

Distributed Algorithms for Mobile Networks

Sébastien Tixeuil

LIP6 - CNRS 7606 - UPMC Sorbonne Universités
Sebastien.Tixeuil@lip6.fr

Thanks to

- Quentin Bramas
- Arnaud Casteigts
- Arnaud Labourel
- Giuseppe Prencipe
- Eric Ruppert
- Nicola Santoro

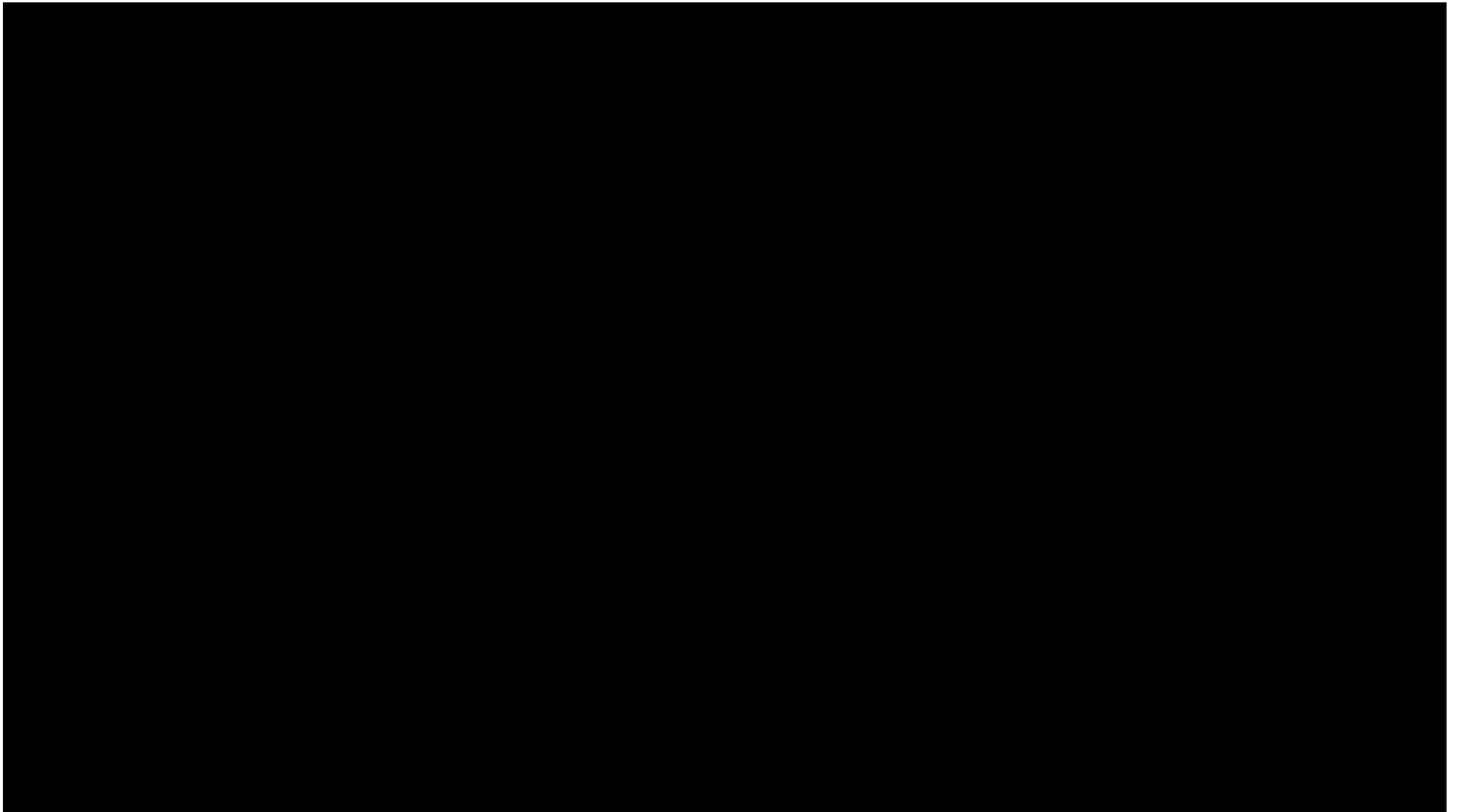
This Lecture is **NOT** about

- **Centralized** algorithms
- Distributed **heuristics**
- **Mobility-unrelated** dynamic networks

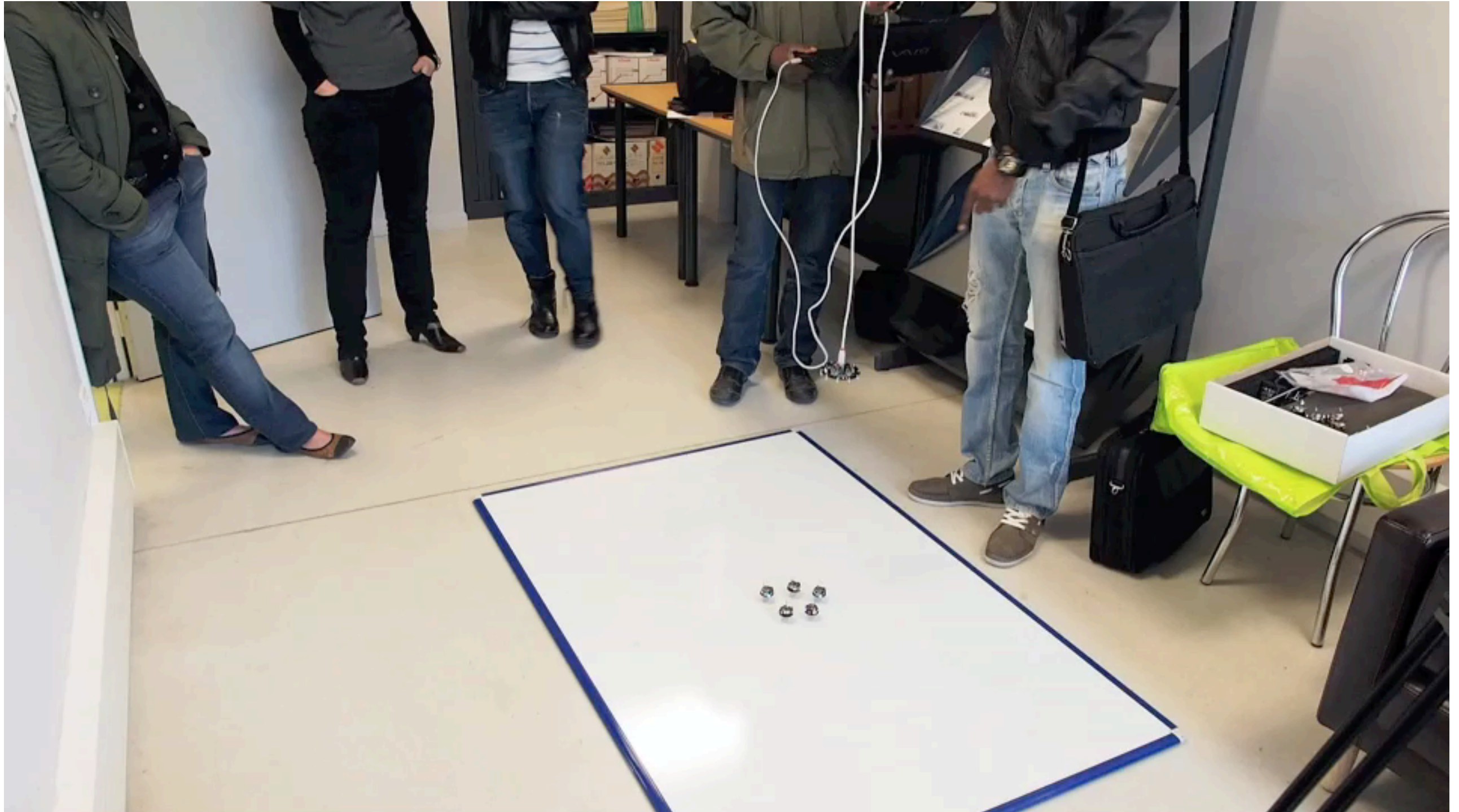
[illegible]

Two Domains

Passively Mobile Networks



Actively Mobile Networks

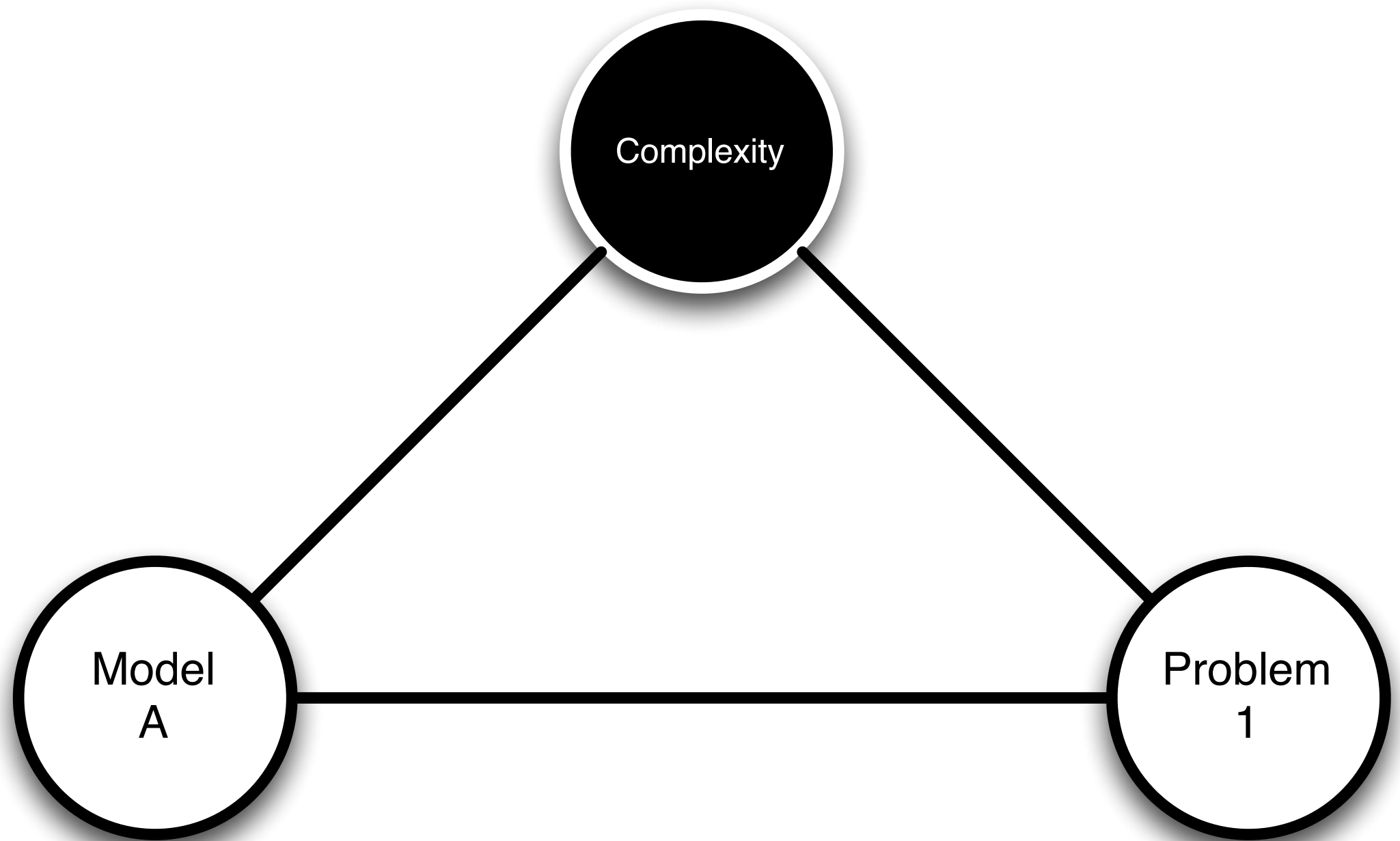


Outline

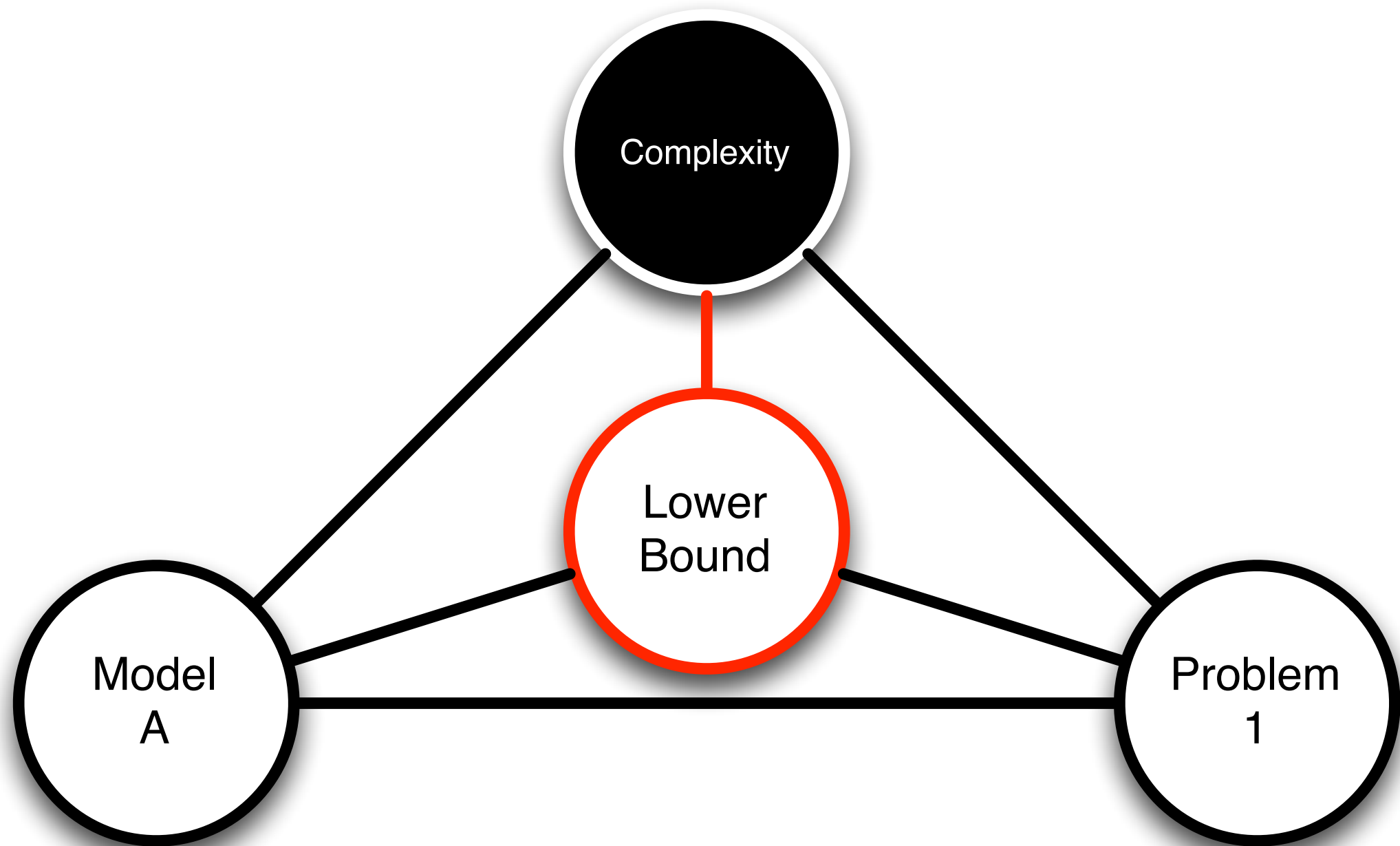
- **Passively mobile networks**
 - Static Algorithms for Dynamic Networks
 - Dynamic Network Models and Algorithms
- **Actively mobile networks**
 - Agents vs. Robots

Three Approaches

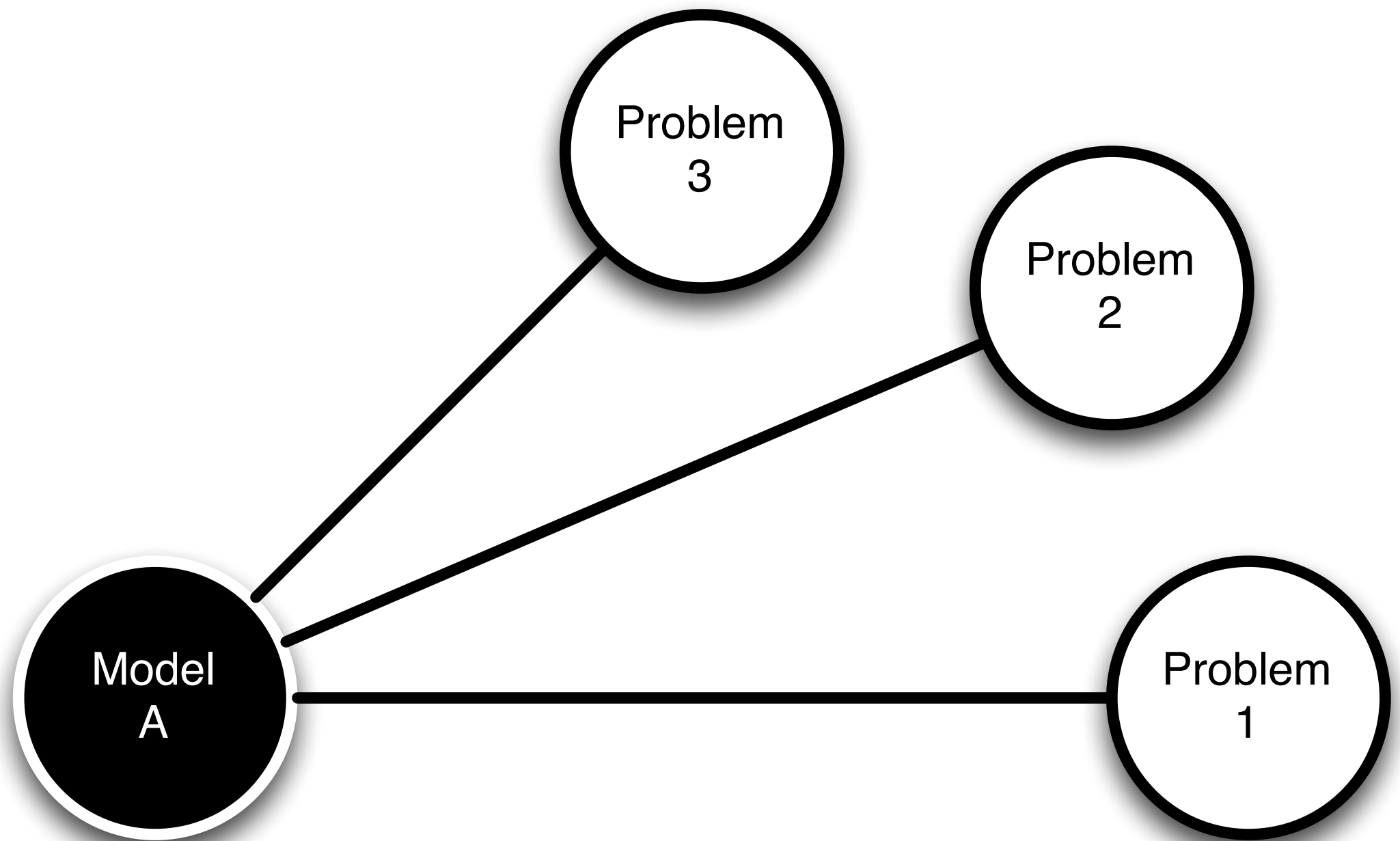
Complexity-driven



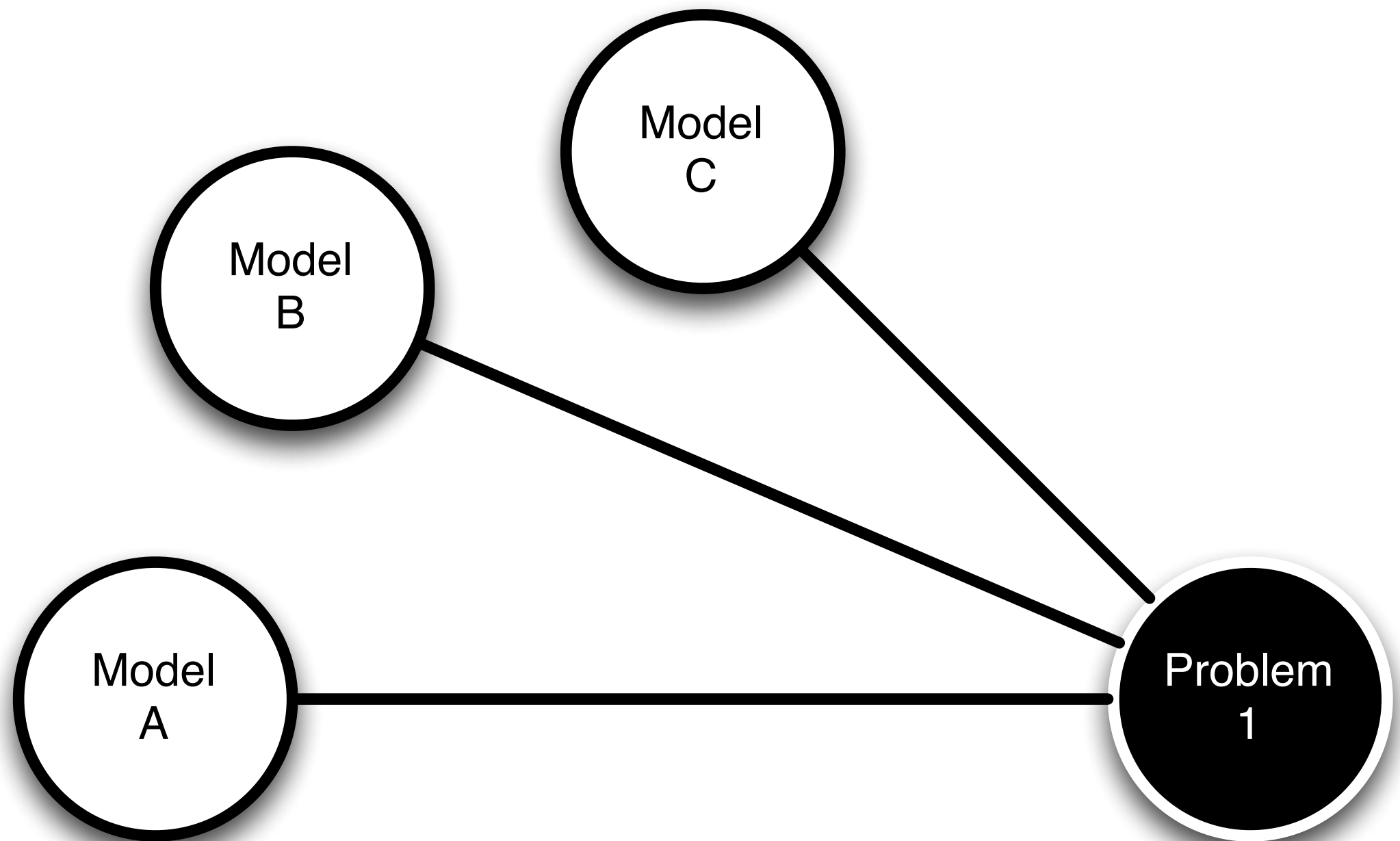
Complexity-driven



Model-driven



Problem-driven



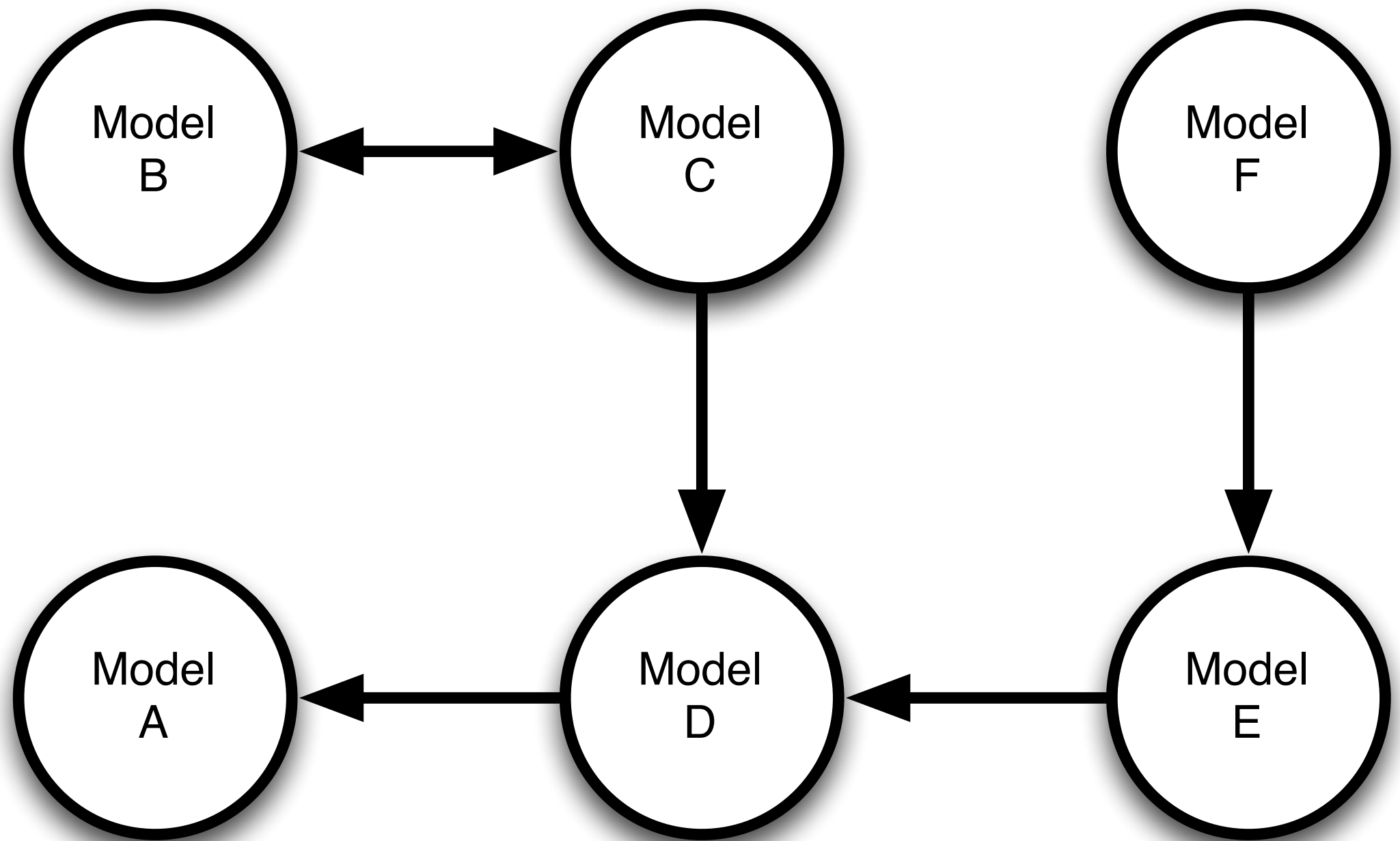
- Landing gear's frozen
- I lost my radar
- I'm out of fuel
- Lost a wing
- Lost the other one



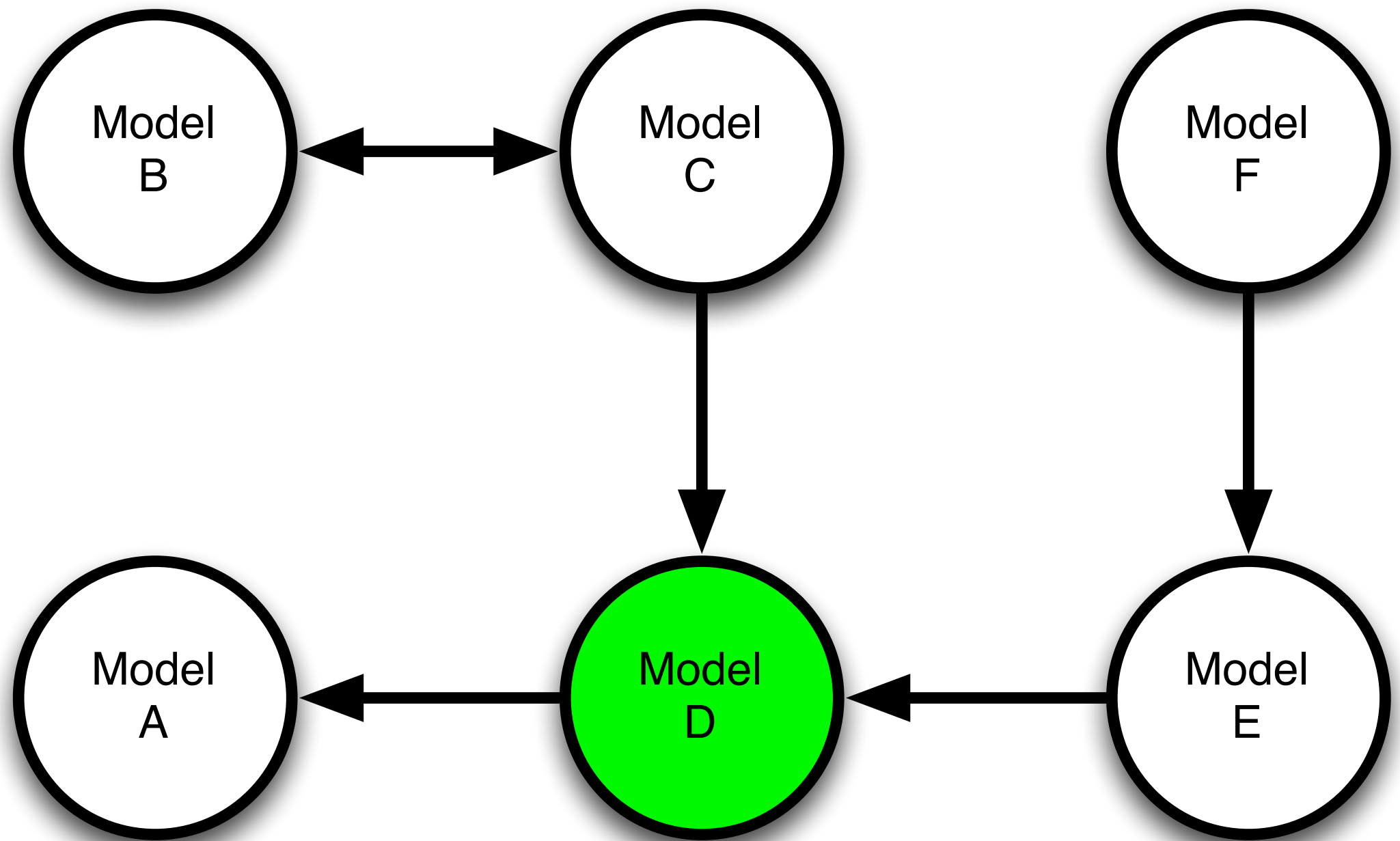
- looking good
- right for line-up
- doing fine
- Ok you can do it



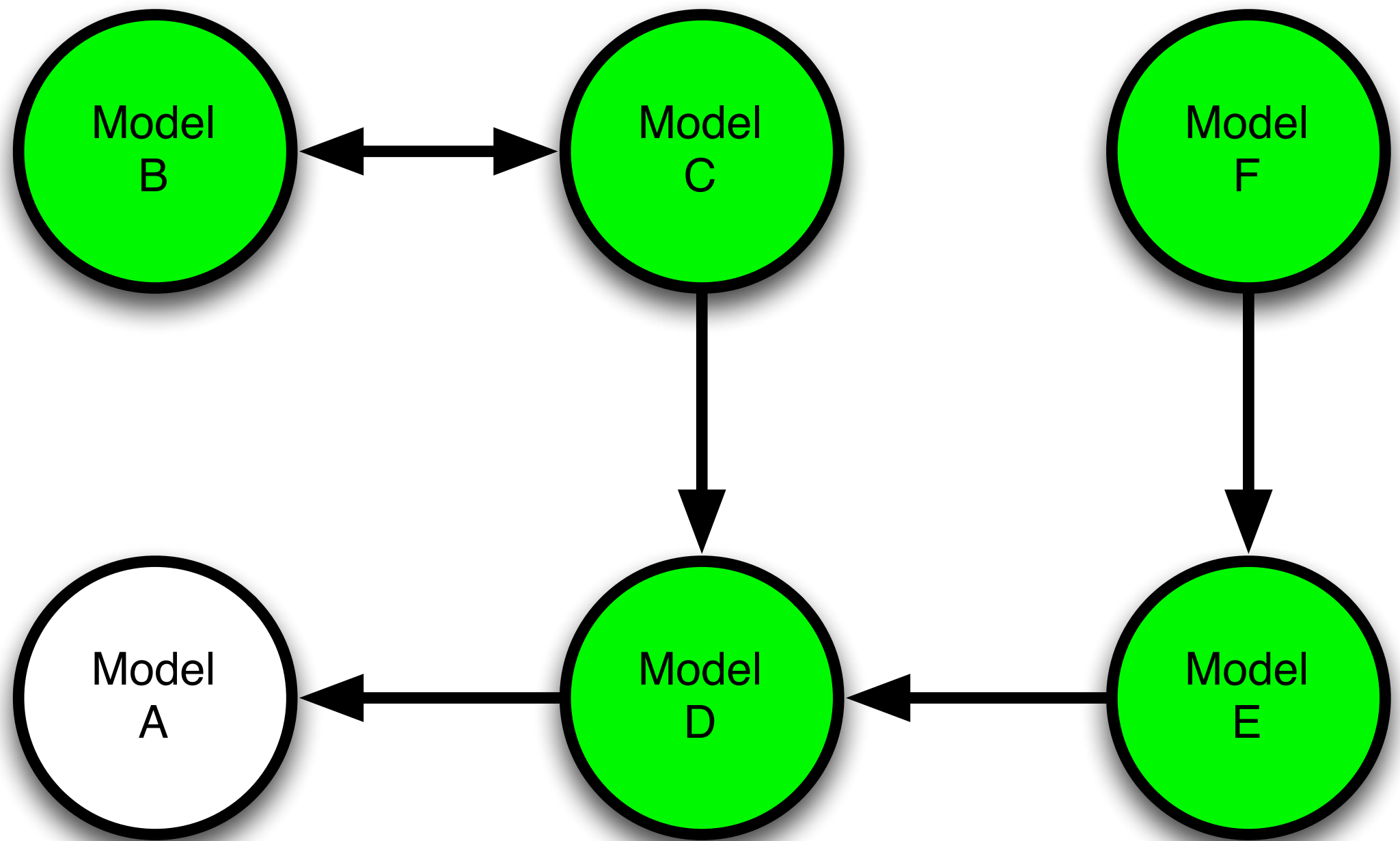
A Map of Models



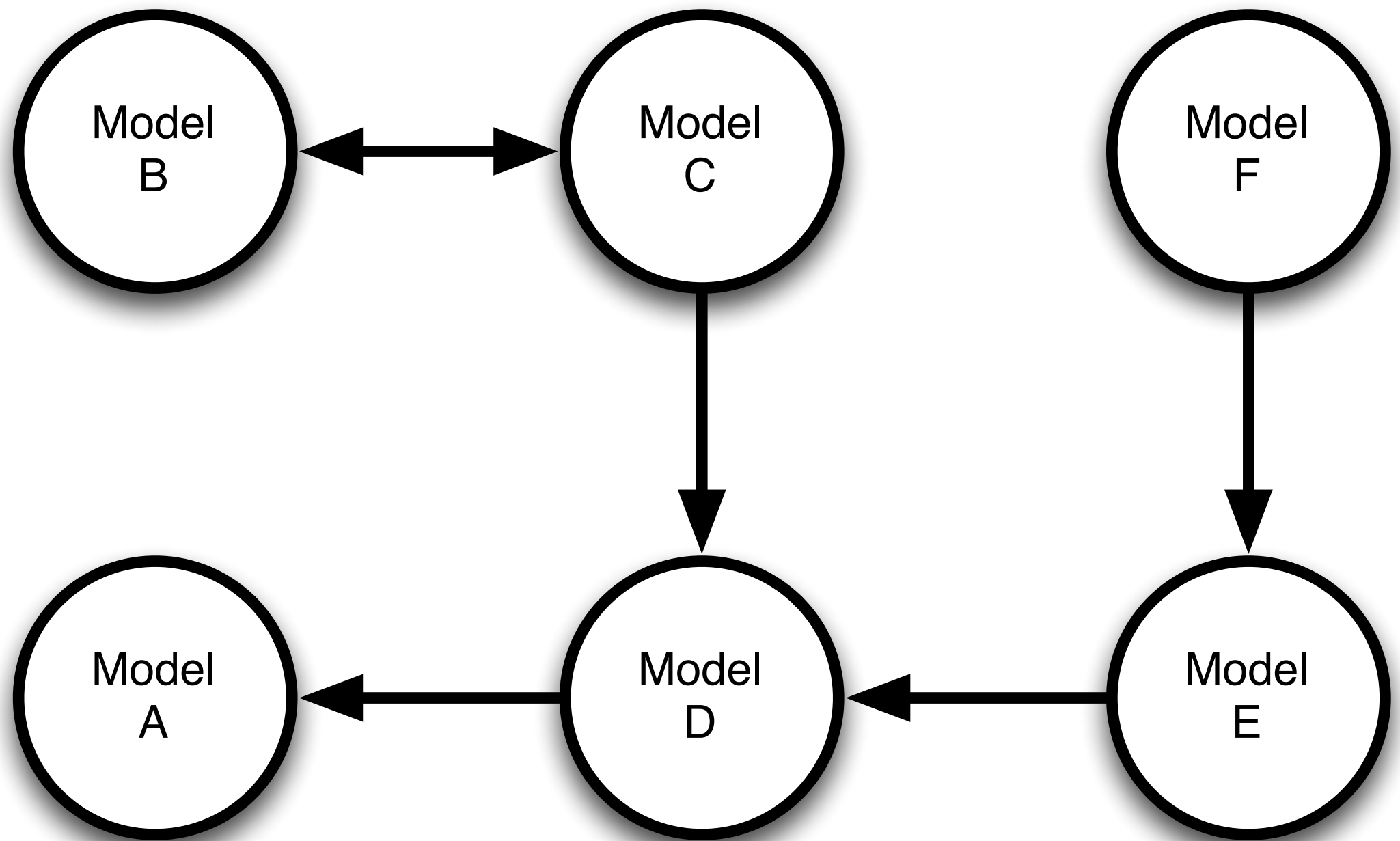
A Map of Models



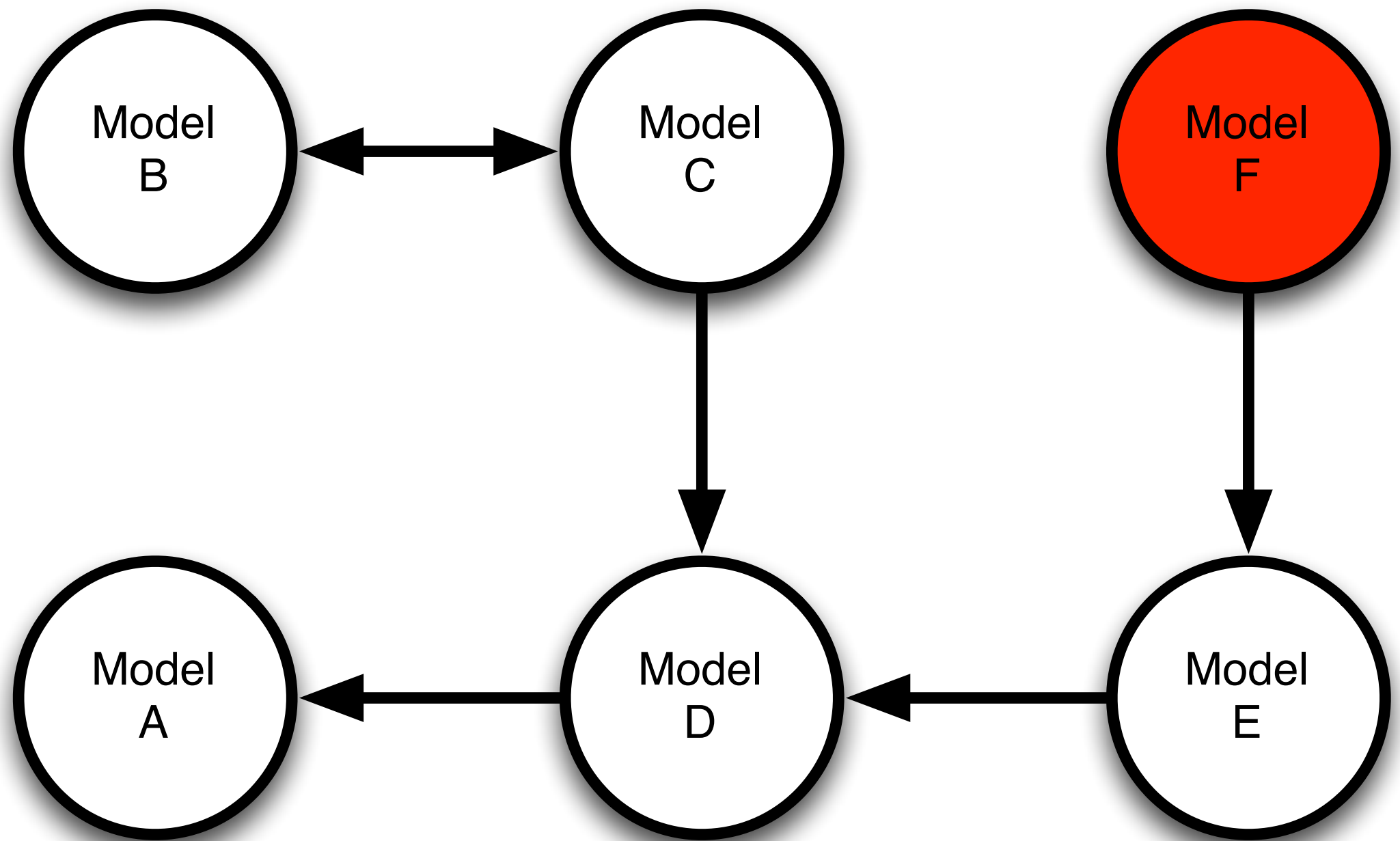
A Map of Models



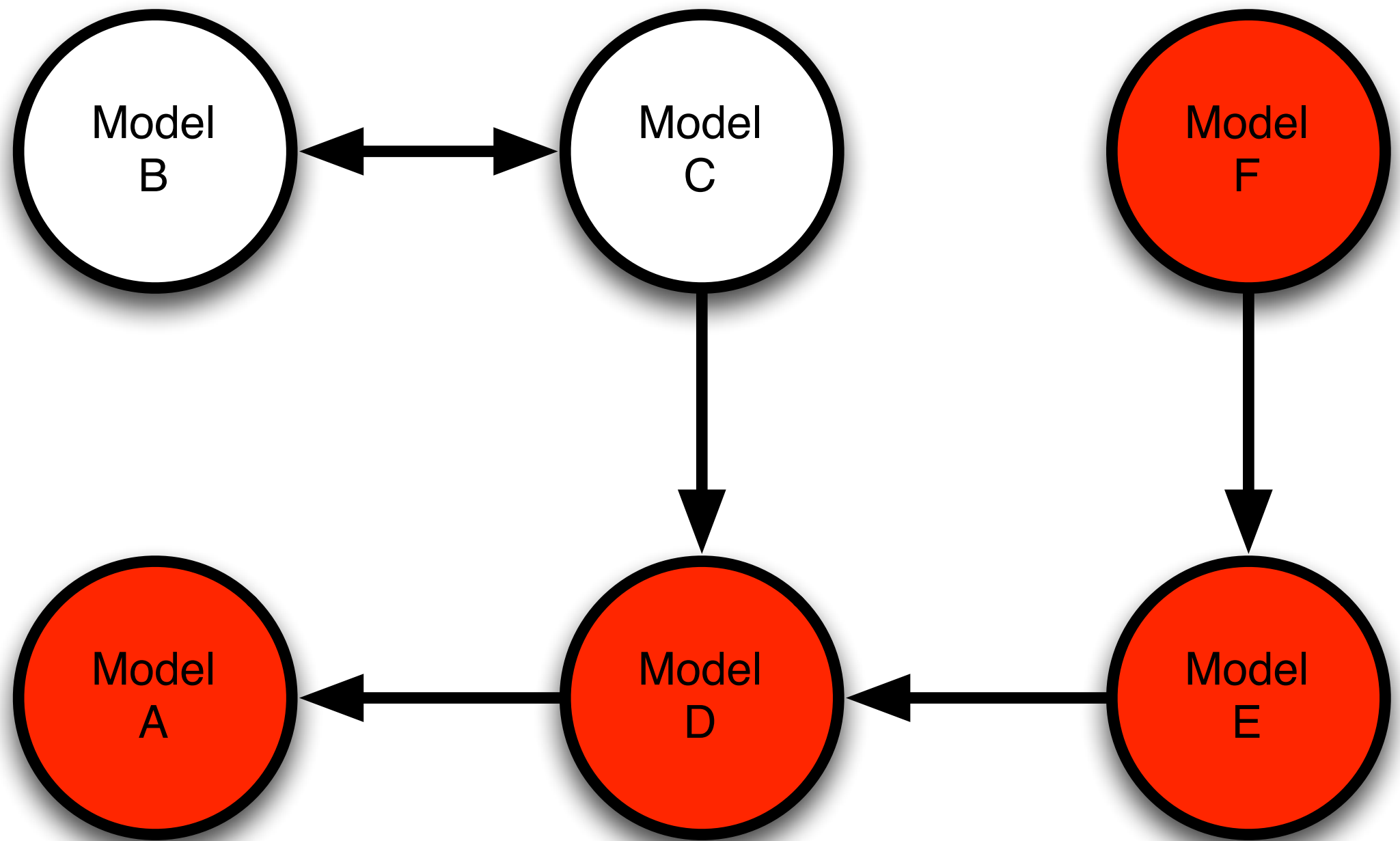
A Map of Models



A Map of Models

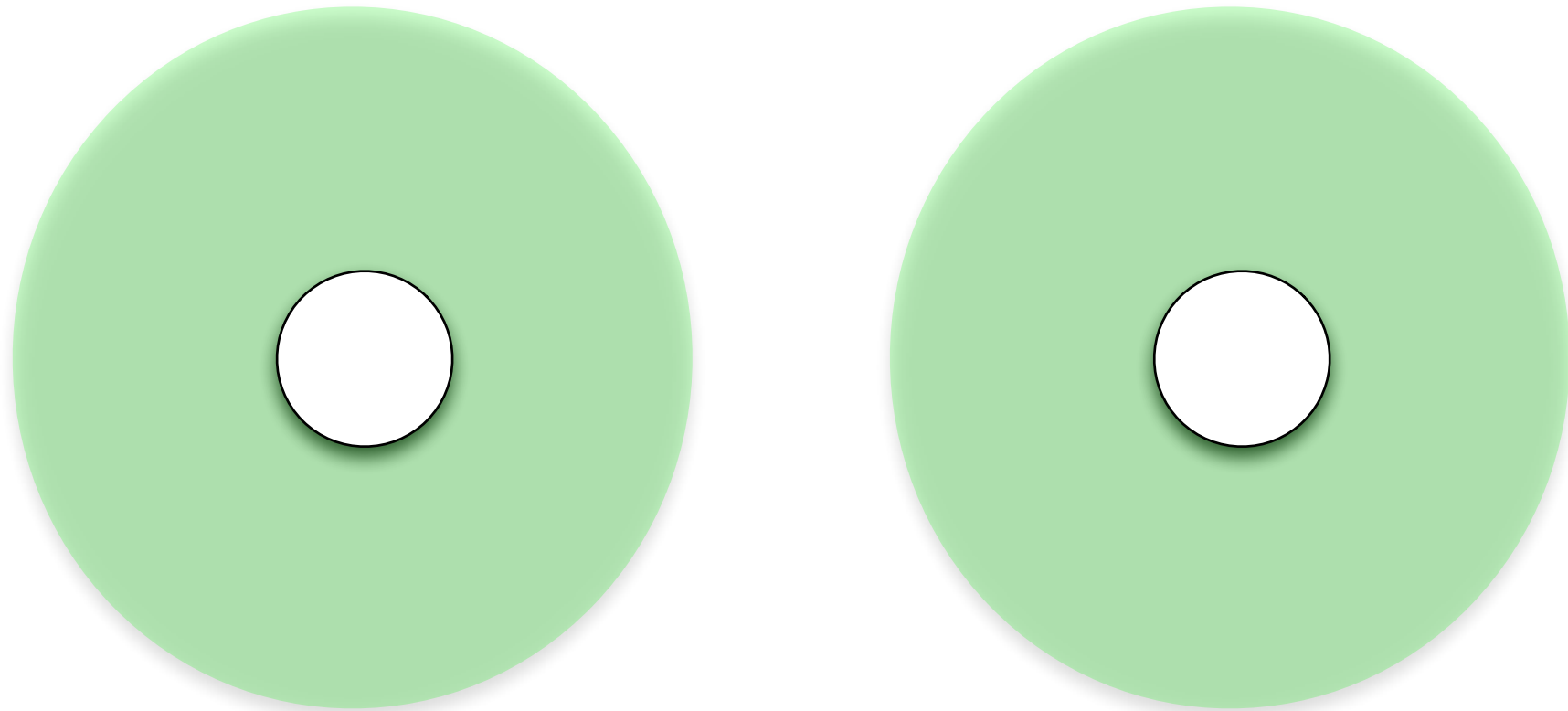


A Map of Models

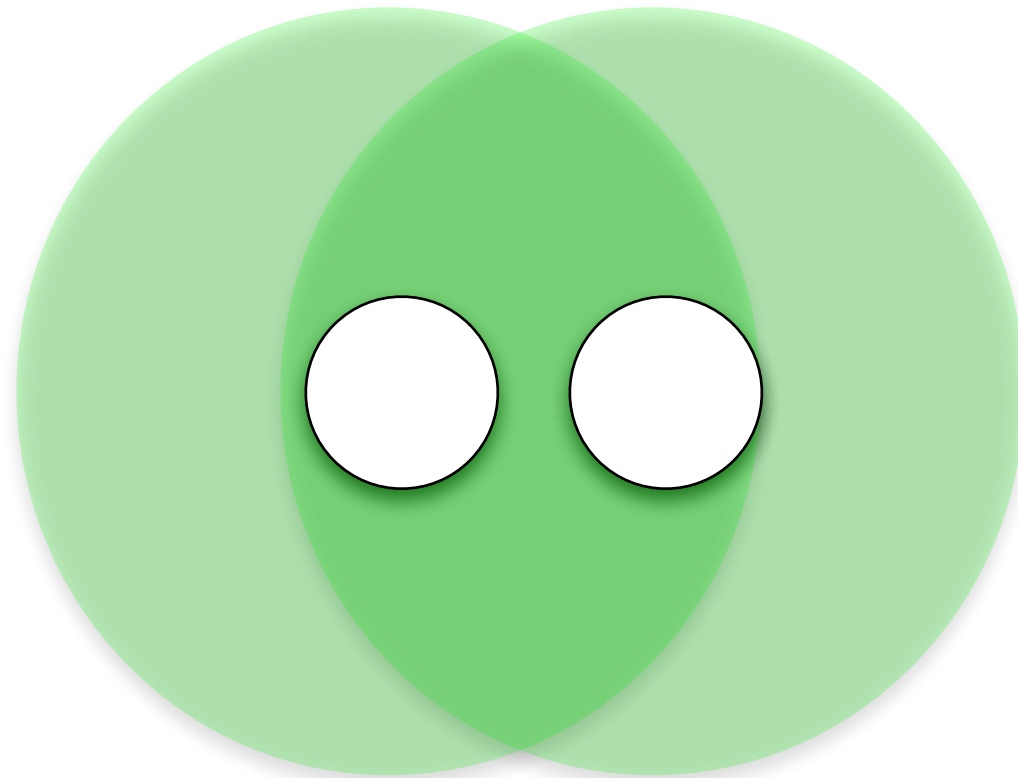


Passively Mobile Networks

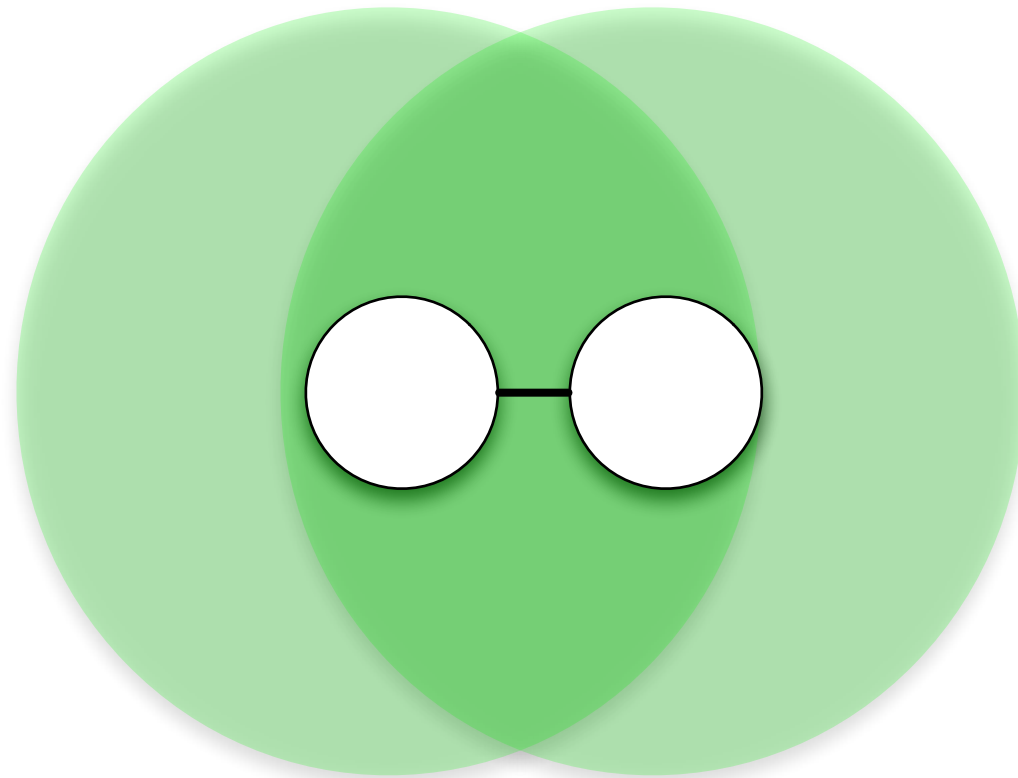
Mobility-induced Dynamic Networks



Mobility-induced Dynamic Networks

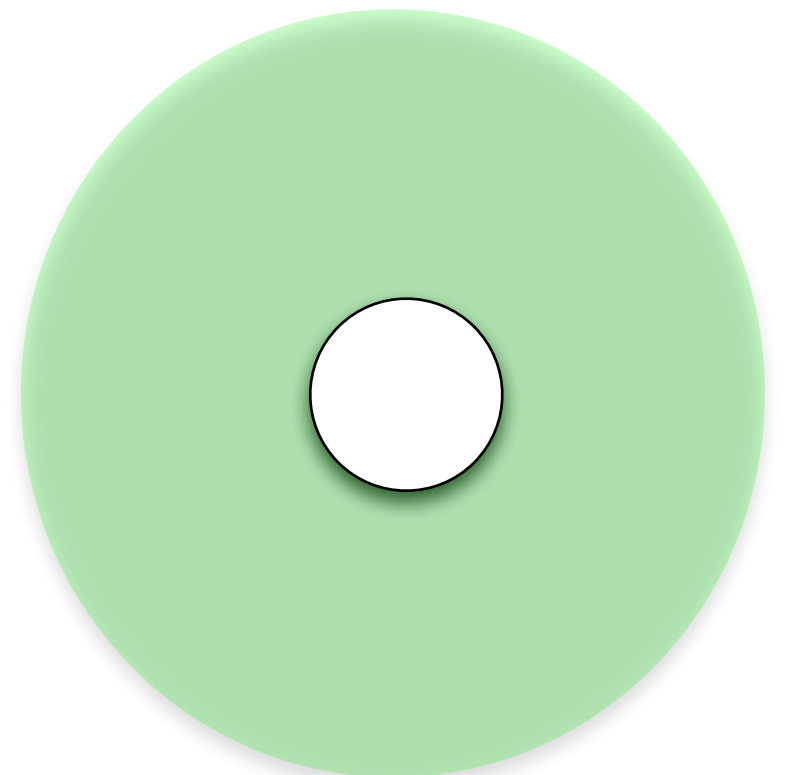
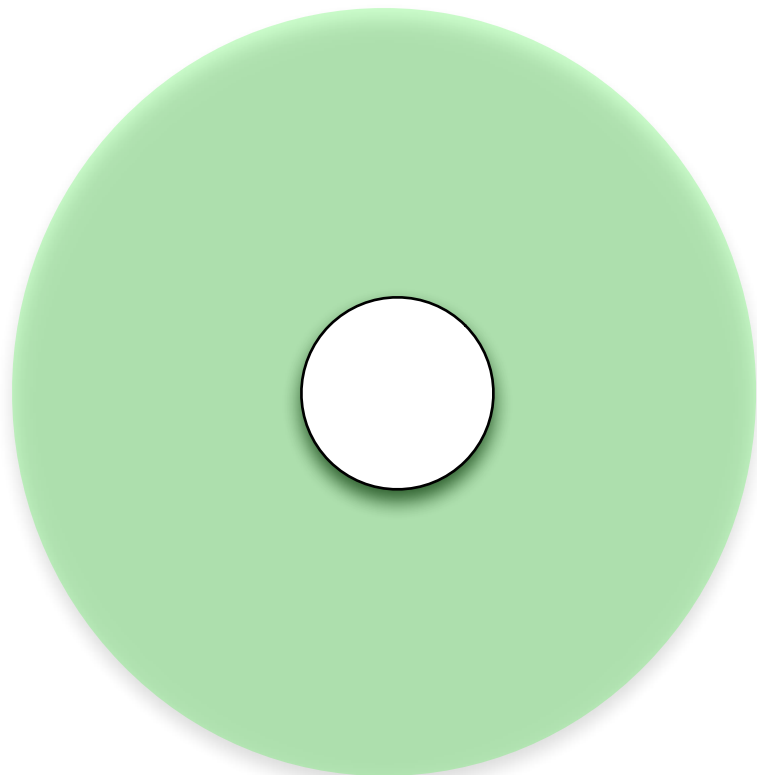


Mobility-induced Dynamic Networks

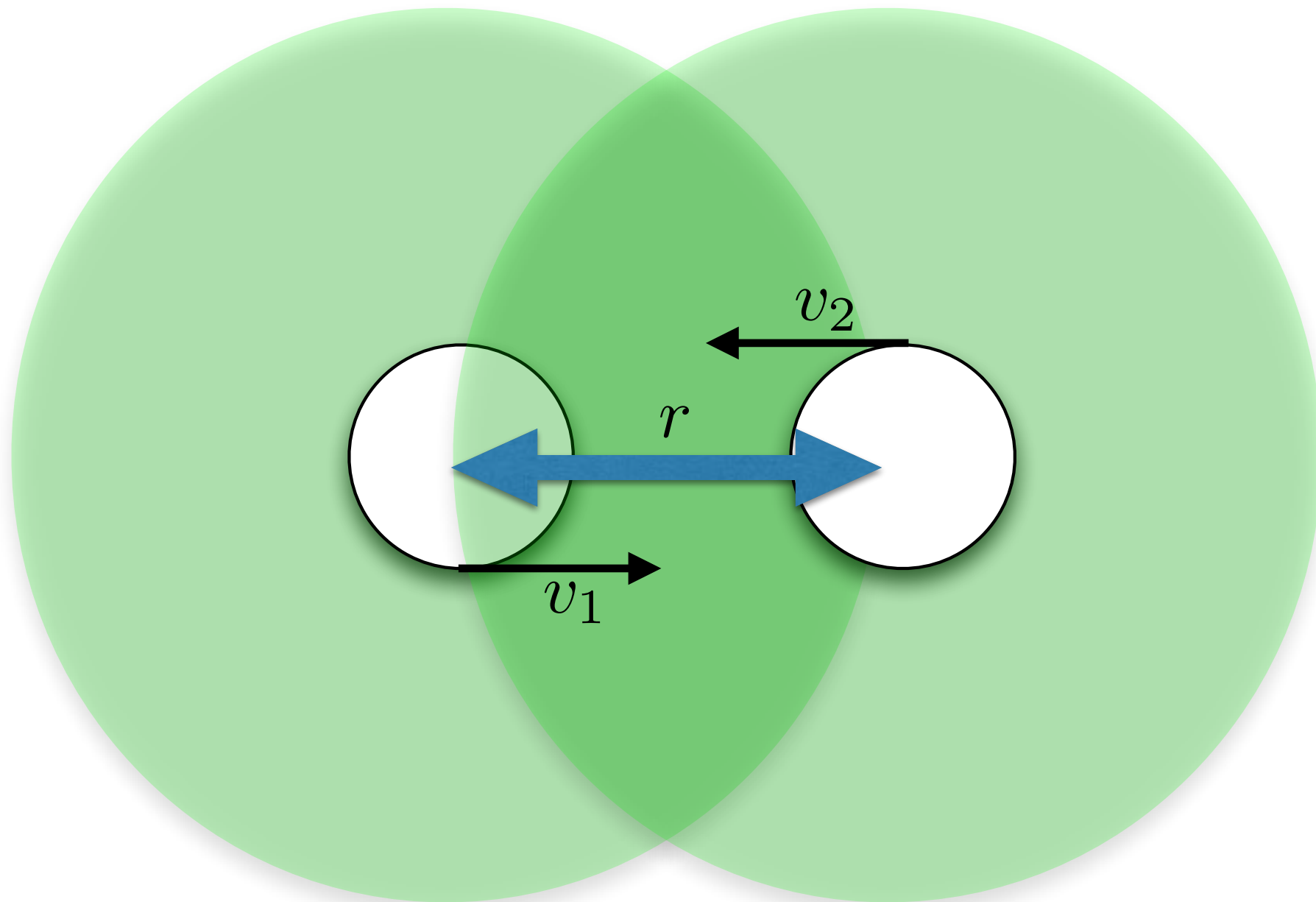


Static Algorithms for Mobile Networks

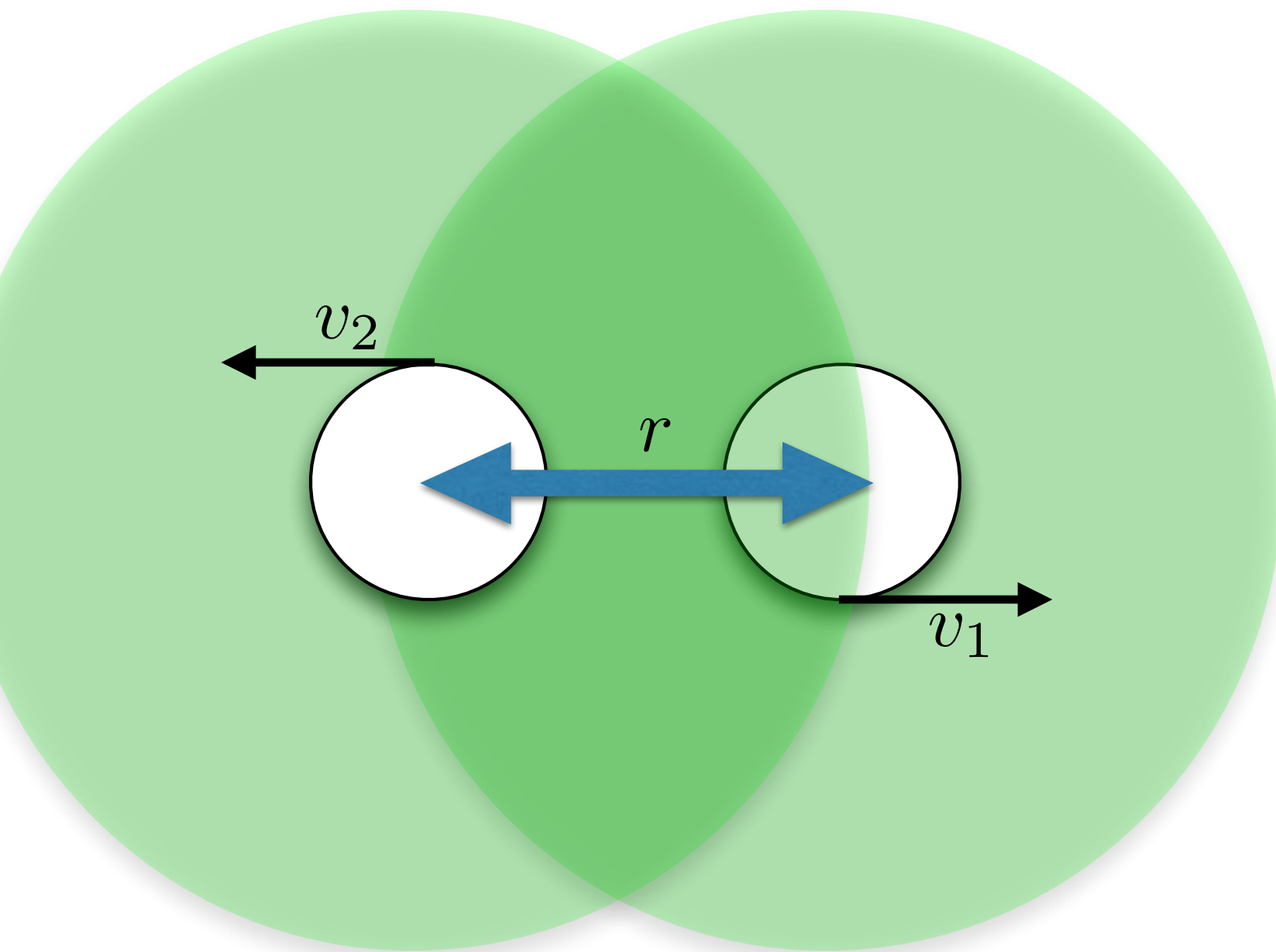
Link Lifetime



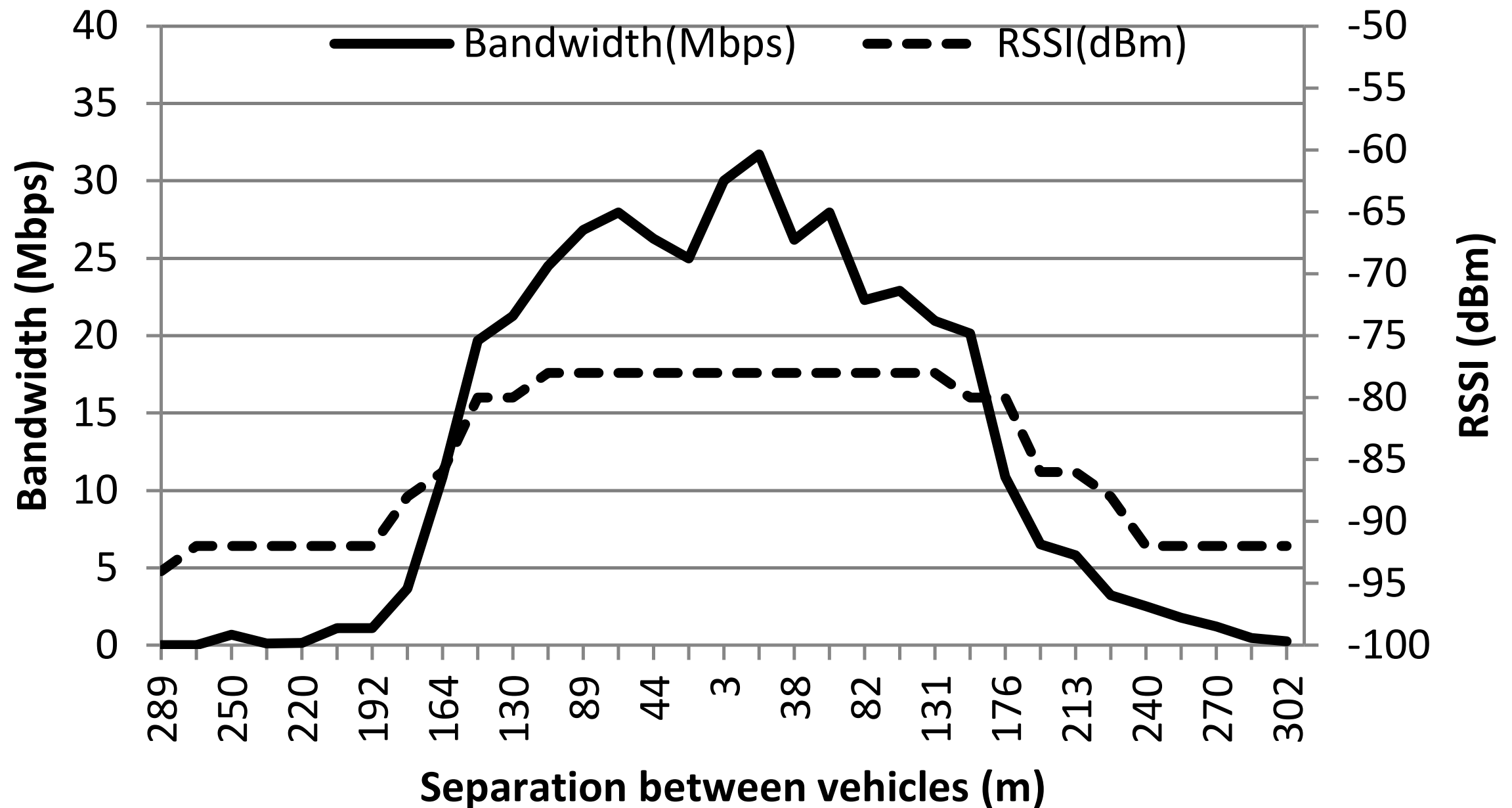
Link Lifetime



Link Lifetime

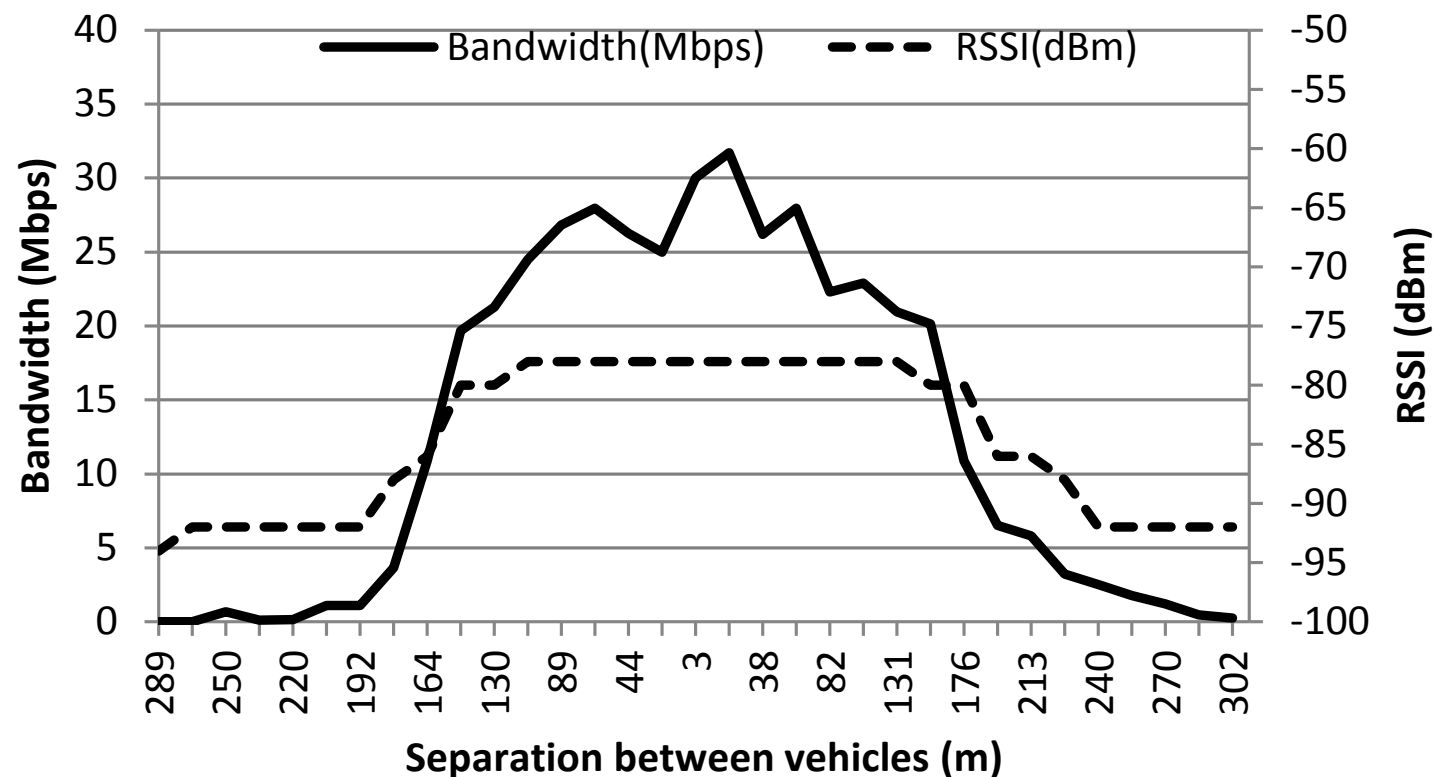


Link Lifetime



On the use of WiMAX and Wi-Fi to provide in-vehicle connectivity and media distribution.
Lerotholi S. Mojela ; Marthinus J. Booysen, Industrial Technology (ICIT), 2013 IEEE International Conference on

Link Lifetime



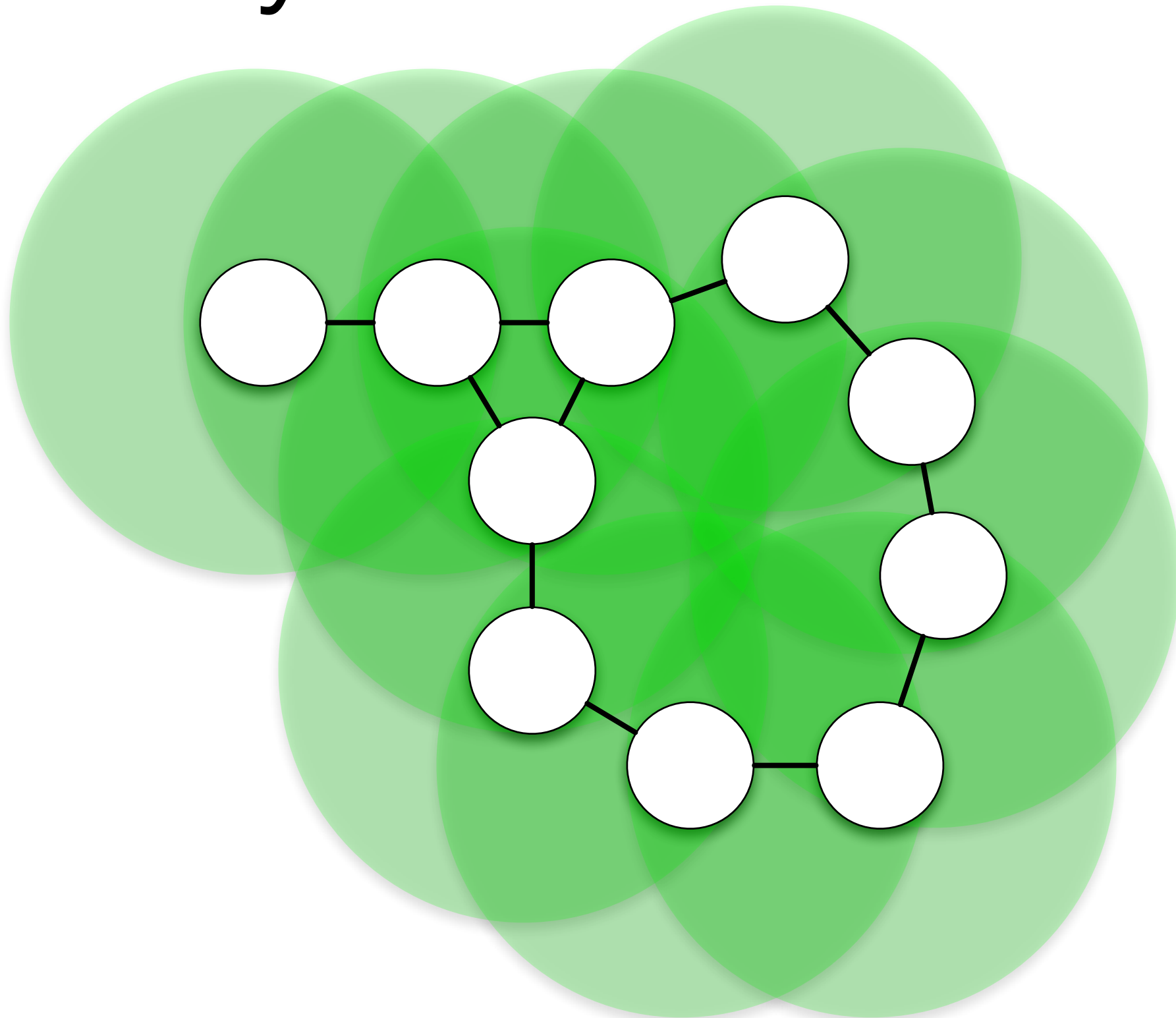
On the use of WiMAX and Wi-Fi to provide in-vehicle connectivity and media distribution. Lerotholi S. Mojela ; Marthinus J. Booysen, Industrial Technology (ICIT), 2013 IEEE International Conference on

Fig. 4b shows a graph of the two vehicles traveling in opposite directions at an average relative speed of 64 km/h in an urban area. The average contact time recorded was 33s and the average maximum communication range was found to be 302 m with an average bandwidth of 13.7 Mbps per test run taken over the period of established contact, average jitter of 1.88 ms and an average of 51.7 MB data transferred per contact period. The maximum peak bandwidth of 31.7 Mbps

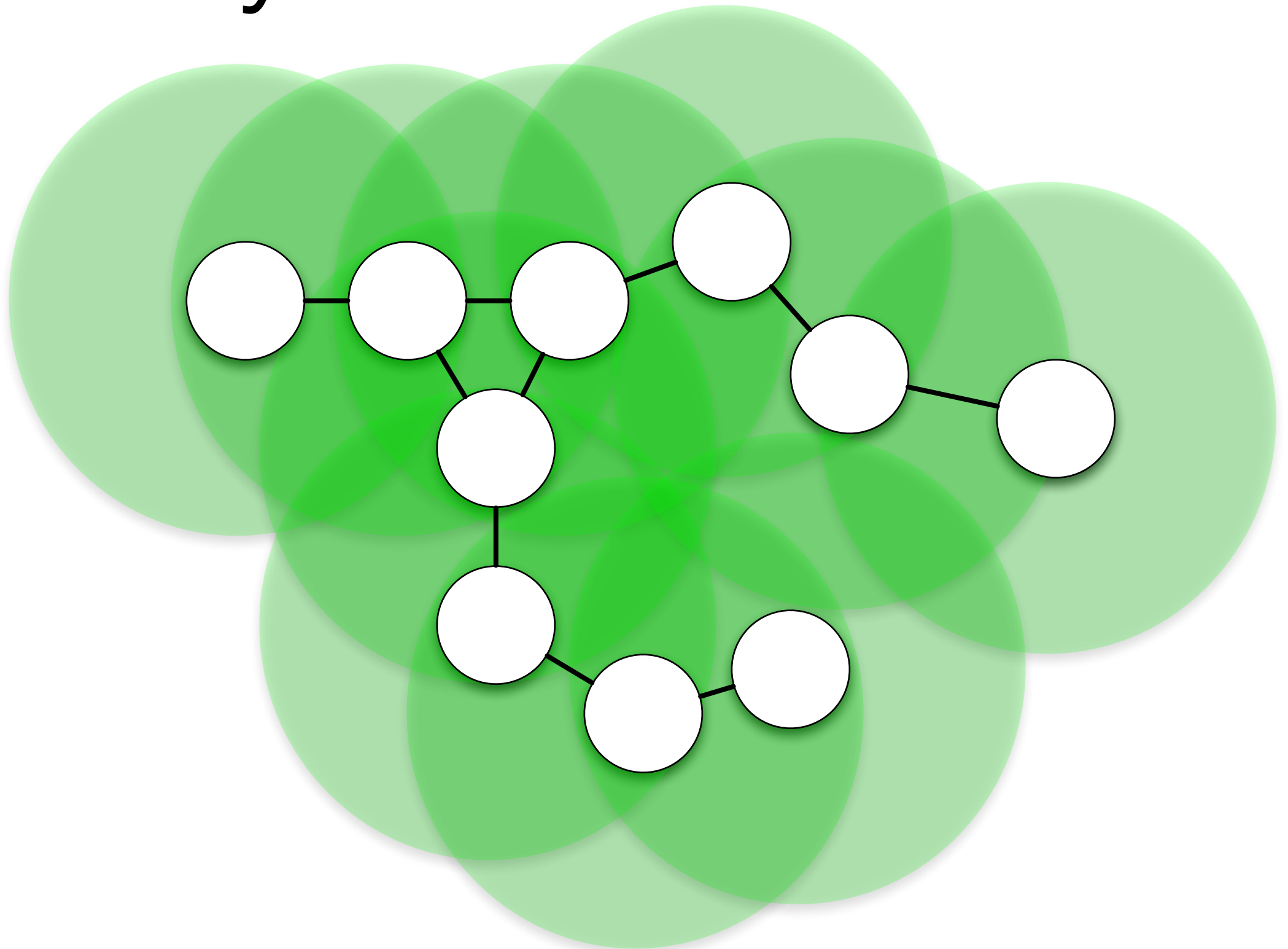
Round-trip Time

| | | Barcelona✖ | Paris✖ | Tokyo✖ | Toronto✖ | Washington✖ |
|--------------|--|-------------|-------------|-------------|-------------|-------------|
| Amsterdam✖ | | ● 44.173ms | ● 11.876ms | ● 242.819ms | ● 96.619ms | ● 93.342ms |
| Auckland✖ | | ● 295.323ms | ● 280.111ms | ● 241.541ms | ● 199.854ms | ● 211.48ms |
| Berlin✖ | | ● 49.71ms | ● 20.041ms | ● 275.013ms | ● 103.027ms | ● 113.909ms |
| Copenhagen✖ | | ● 43.745ms | ● 25.574ms | ● 272.459ms | ● 117.751ms | ● 100.623ms |
| Dallas✖ | | ● 143.543ms | ● 114.941ms | ● 145.957ms | ● 76.173ms | ● 34.194ms |
| London✖ | | ● 35.431ms | ● 4.542ms | ● 240.519ms | ● 85.102ms | ● 75.484ms |
| Los Angeles✖ | | ● 157.289ms | ● 141.484ms | ● 112.265ms | ● 58.689ms | ● 71.881ms |
| Moscow✖ | | ● 74.806ms | ● 55.67ms | ● 299.067ms | ● 137.808ms | ● 144.68ms |
| New York✖ | | ● 130.6ms | ● 77.551ms | ● 171.641ms | ● 17.78ms | ● 47.161ms |
| Paris✖ | | ● 31.085ms | — | ● 260.569ms | ● 103.015ms | ● 107.643ms |
| Stockholm✖ | | ● 59.785ms | ● 27.894ms | ● 285.462ms | ● 123.358ms | ● 115.657ms |
| Tokyo✖ | | ● 267.753ms | ● 268.811ms | — | ● 172.841ms | ● 186.818ms |

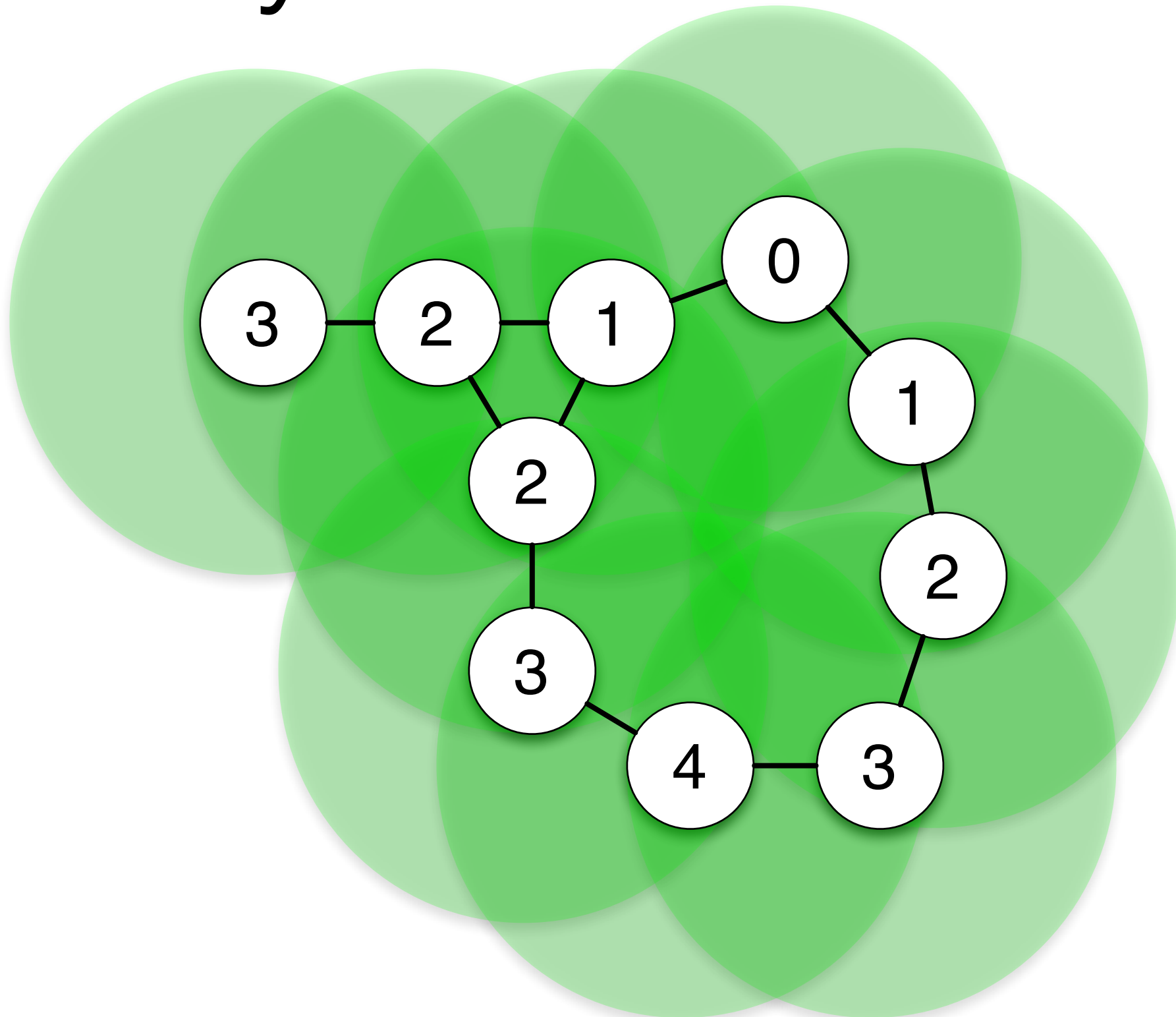
Mobility vs. Global State



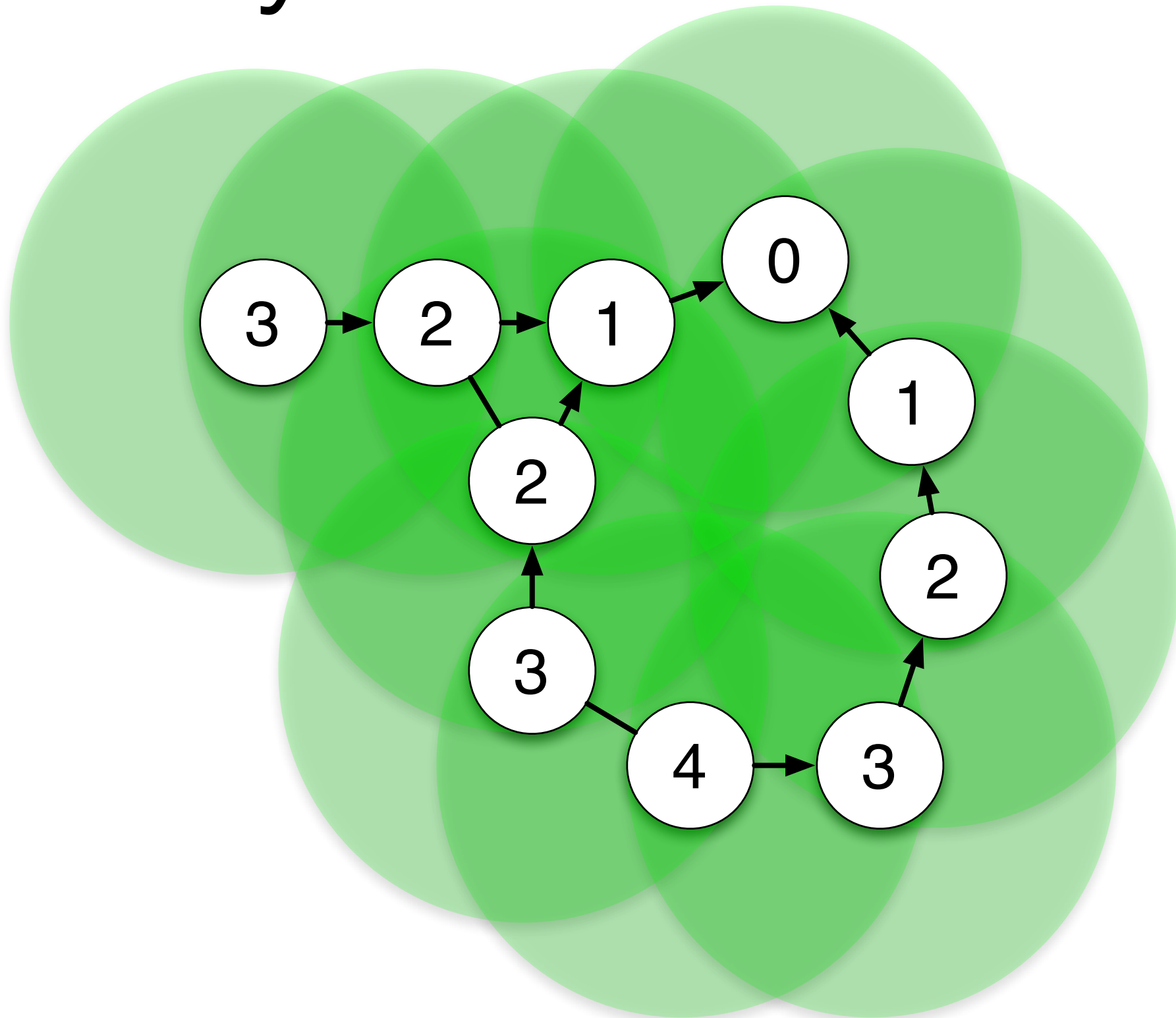
Mobility vs. Global State



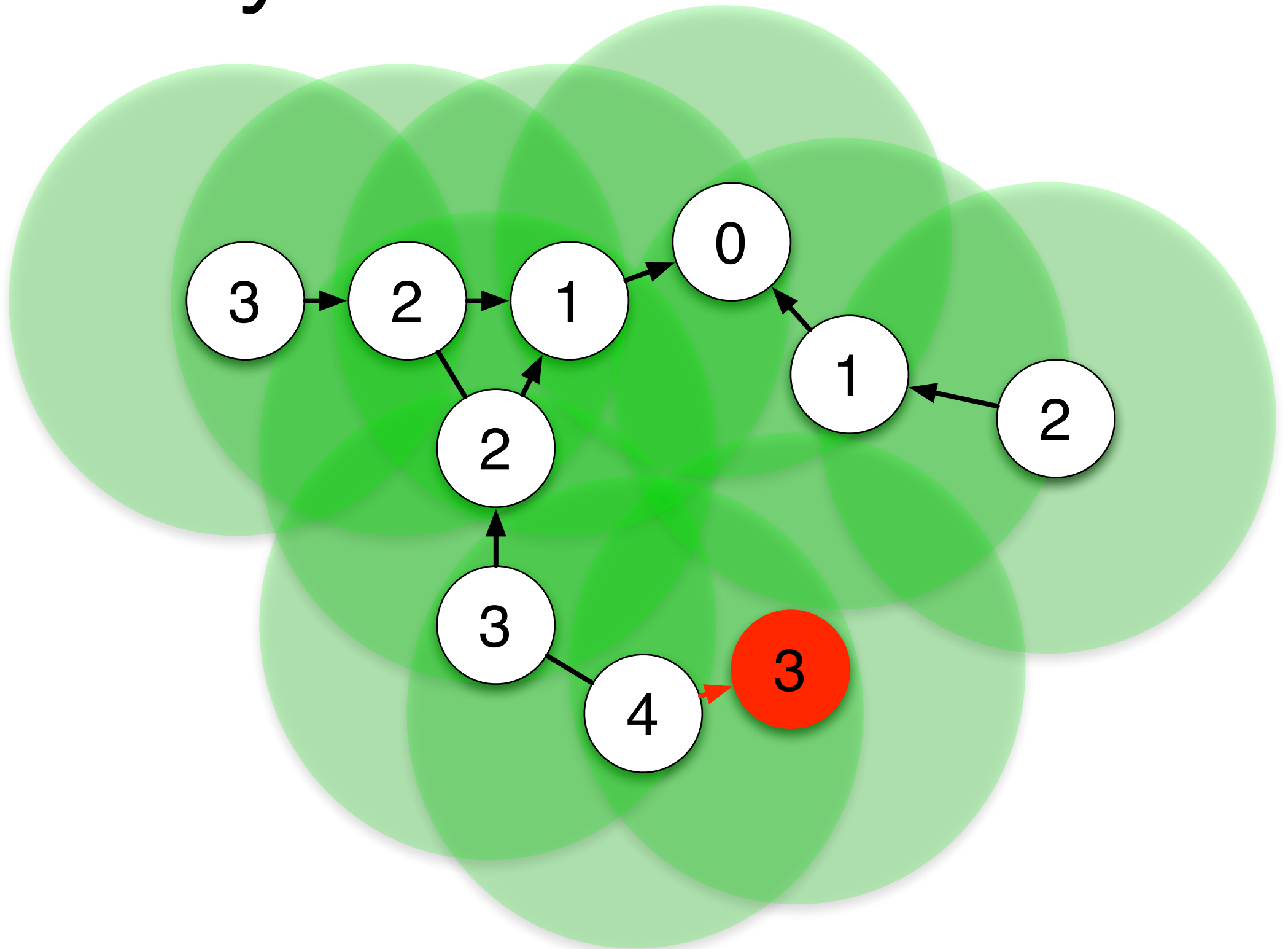
Mobility vs. Global State



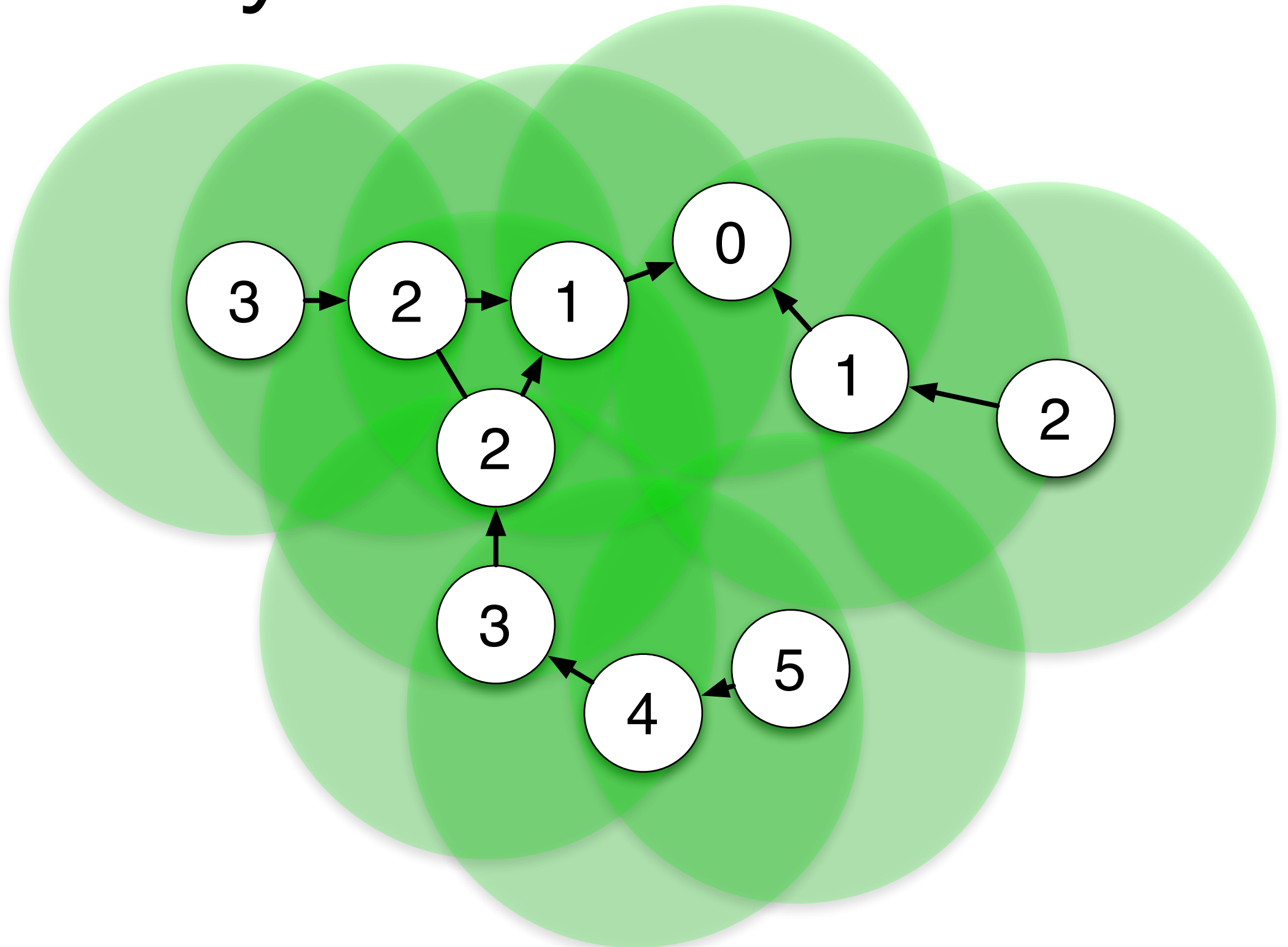
Mobility vs. Global State



Mobility vs. Global State

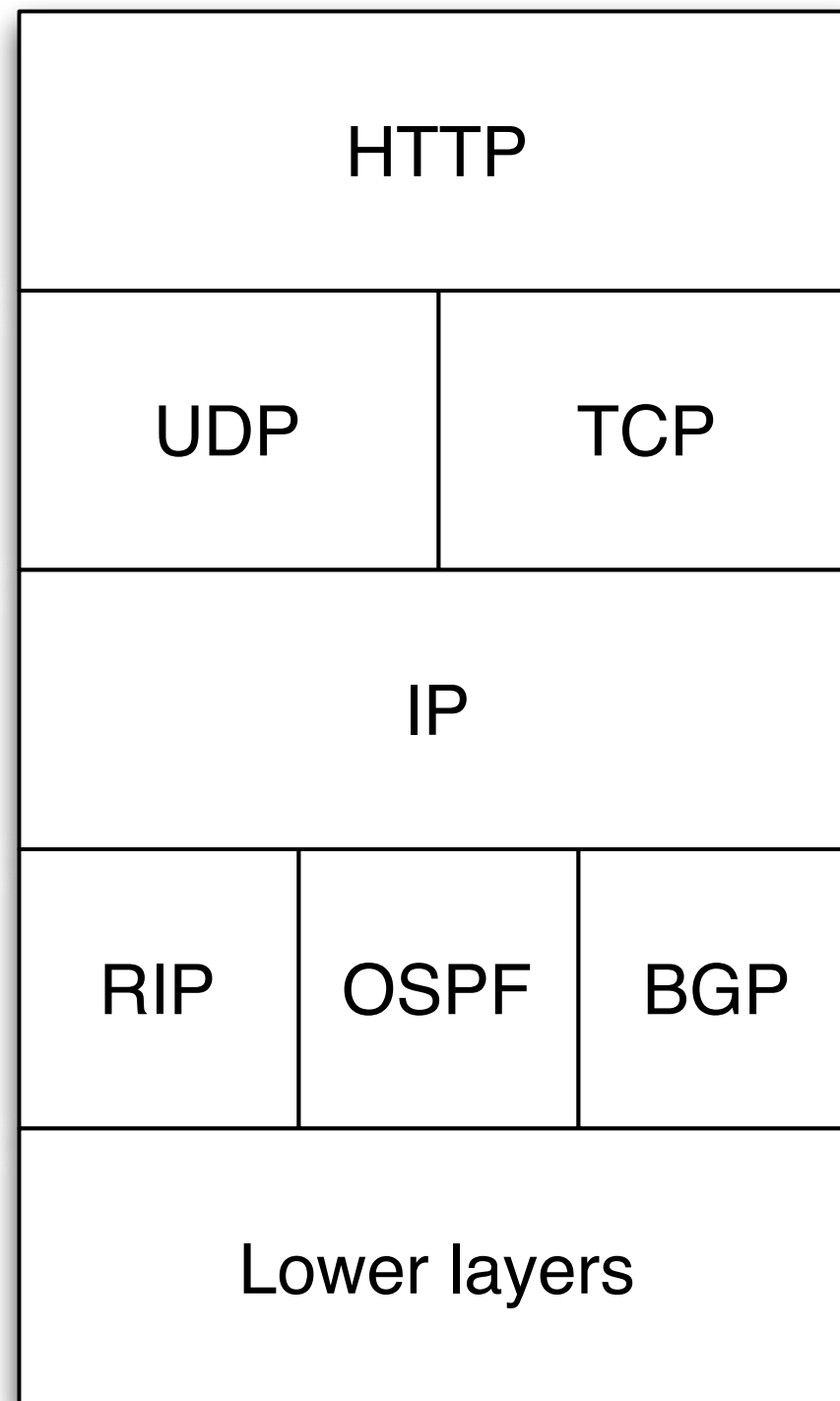


Mobility vs. Global State

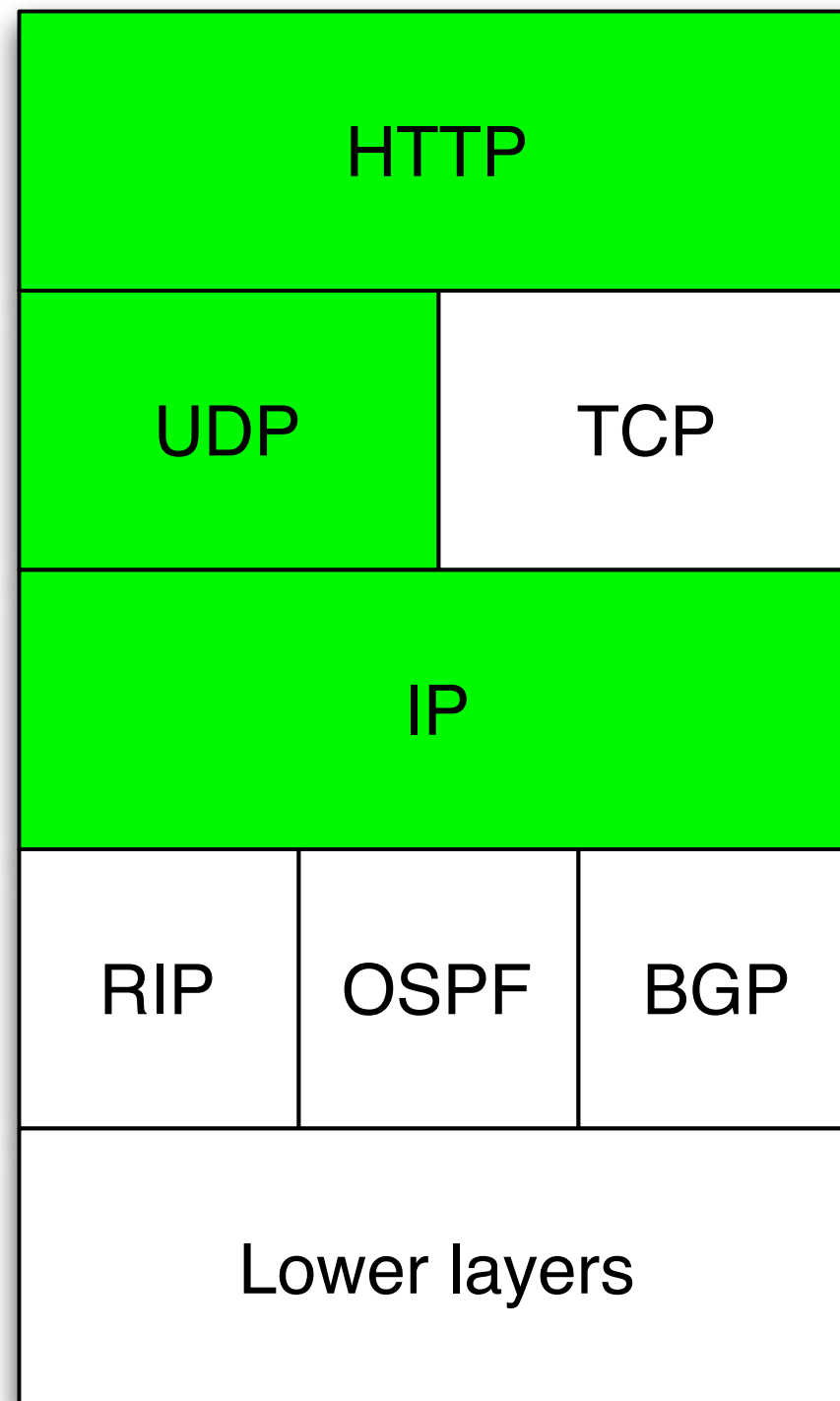


Stateless Algorithms

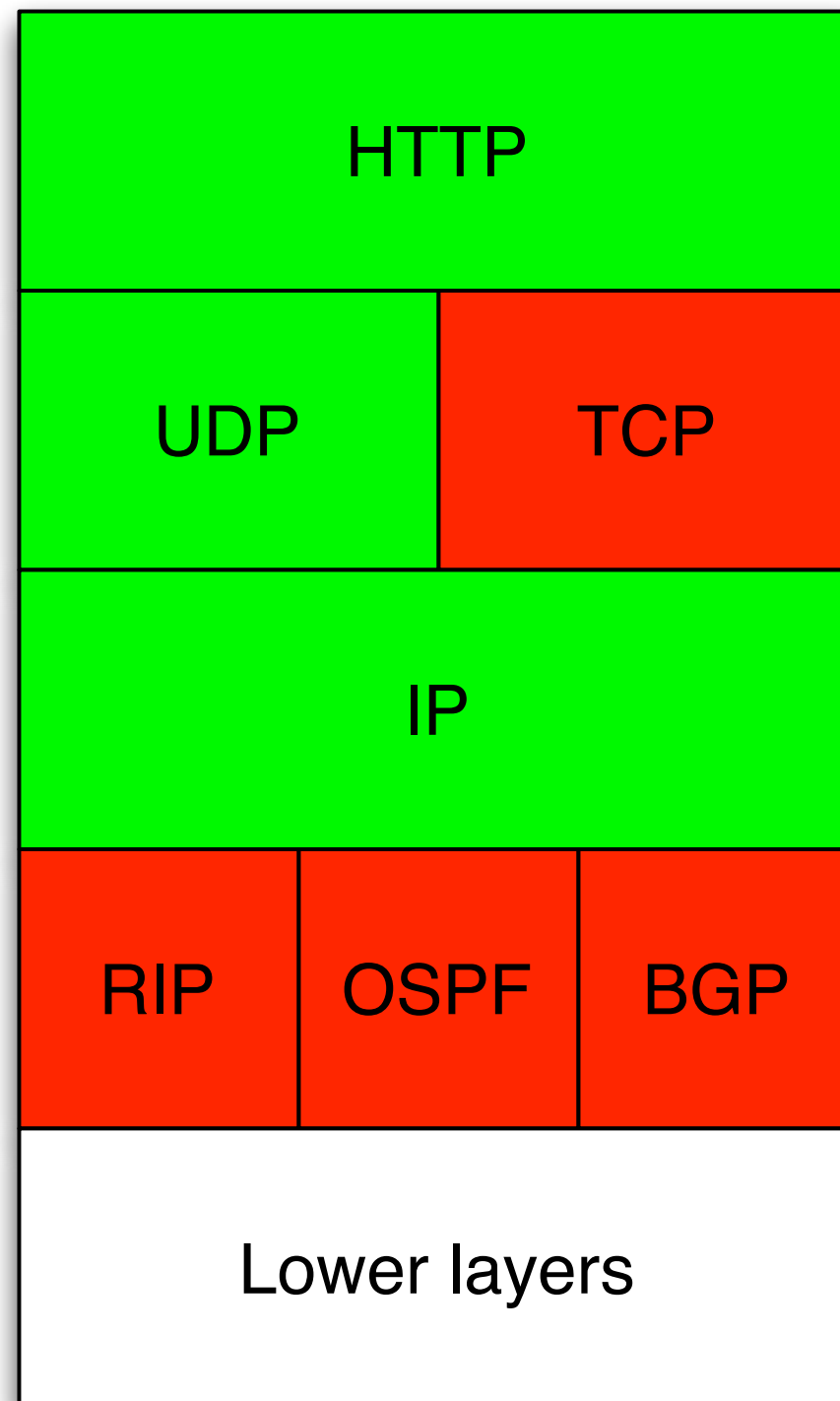
Statelessness



Statelessness



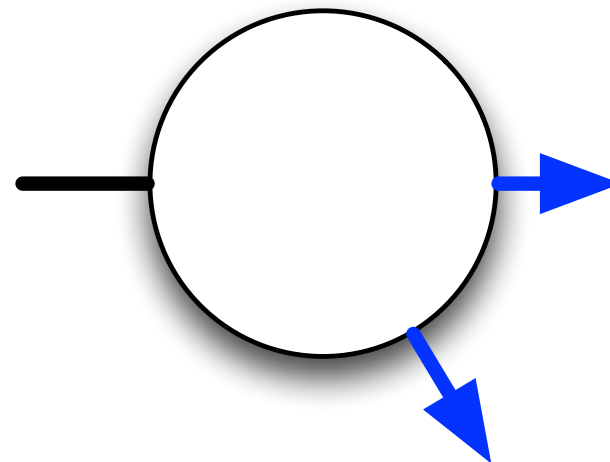
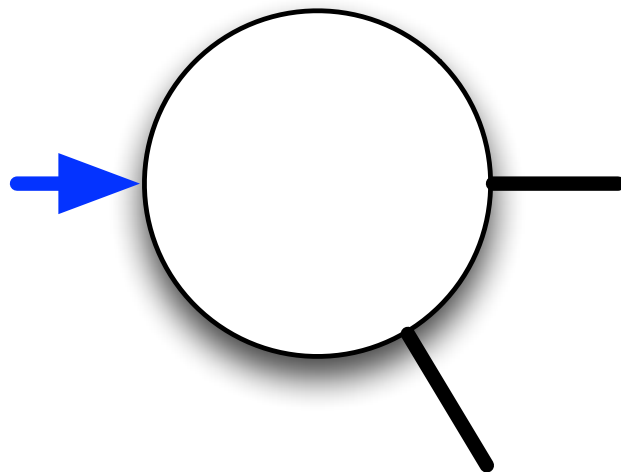
Statelessness



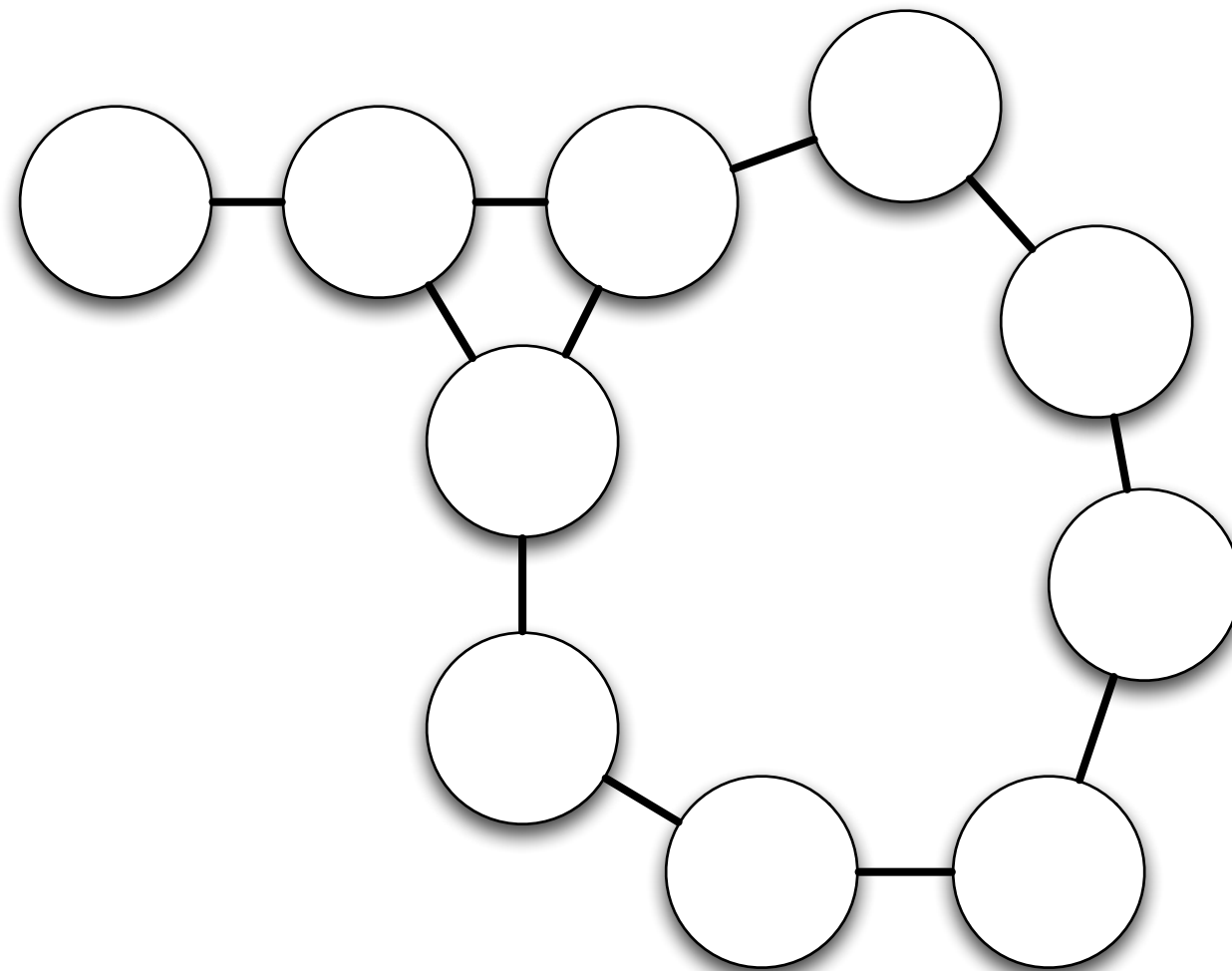
Stateless Routing

A routing algorithm is **stateless** if it is designed such that devices store *no information* about messages *between transmissions*. It is **stateful** otherwise.

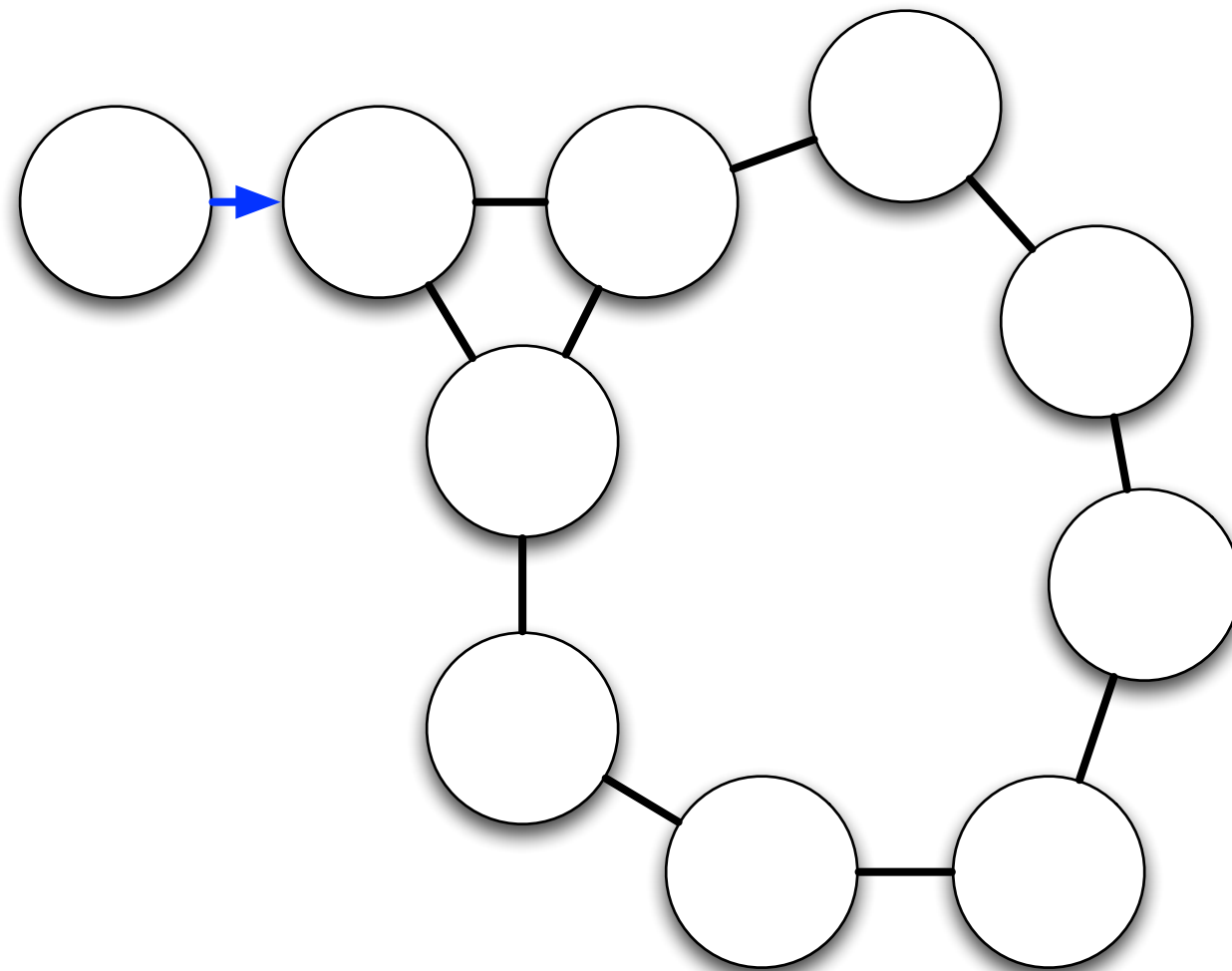
Flooding



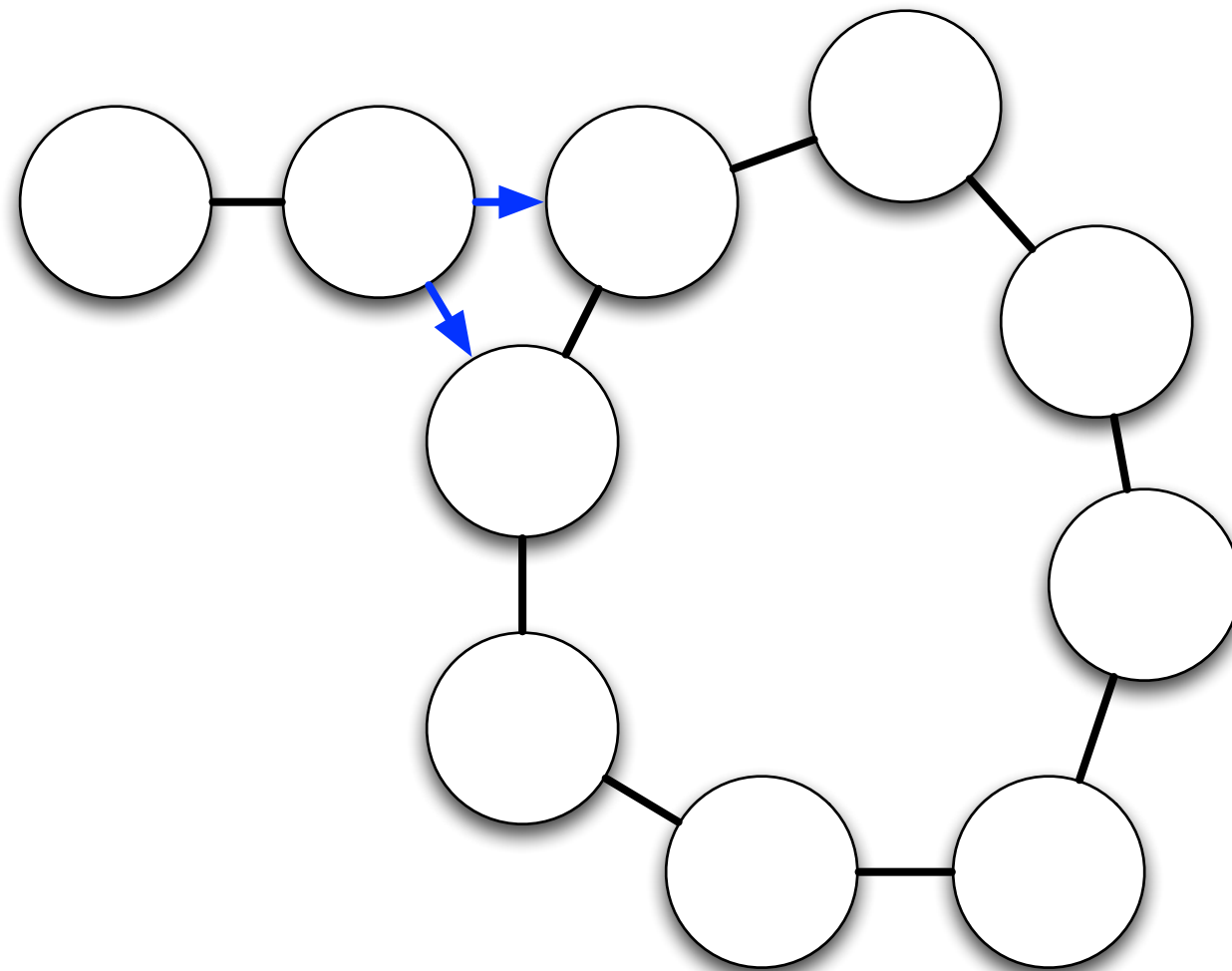
Stateless Flooding



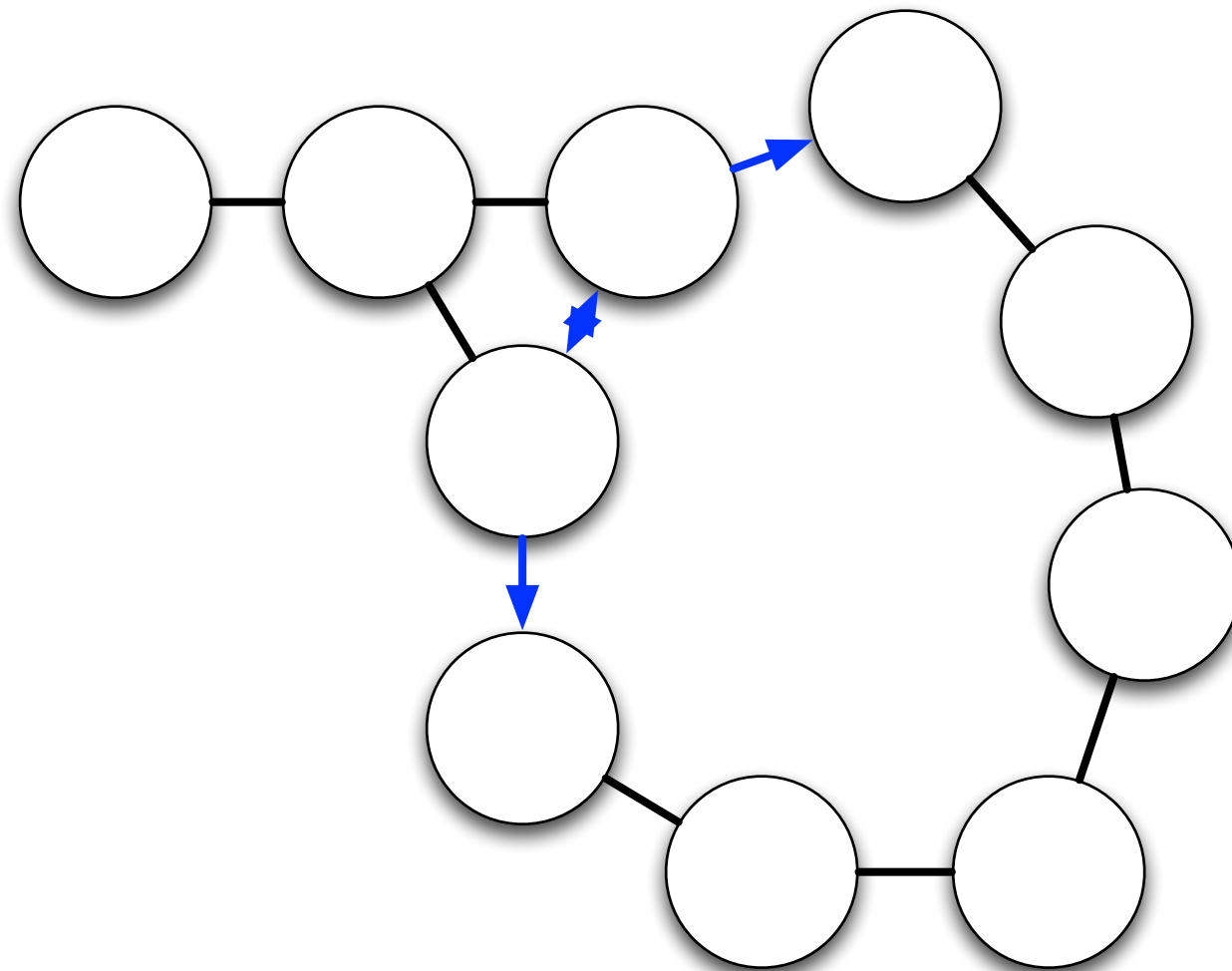
Stateless Flooding



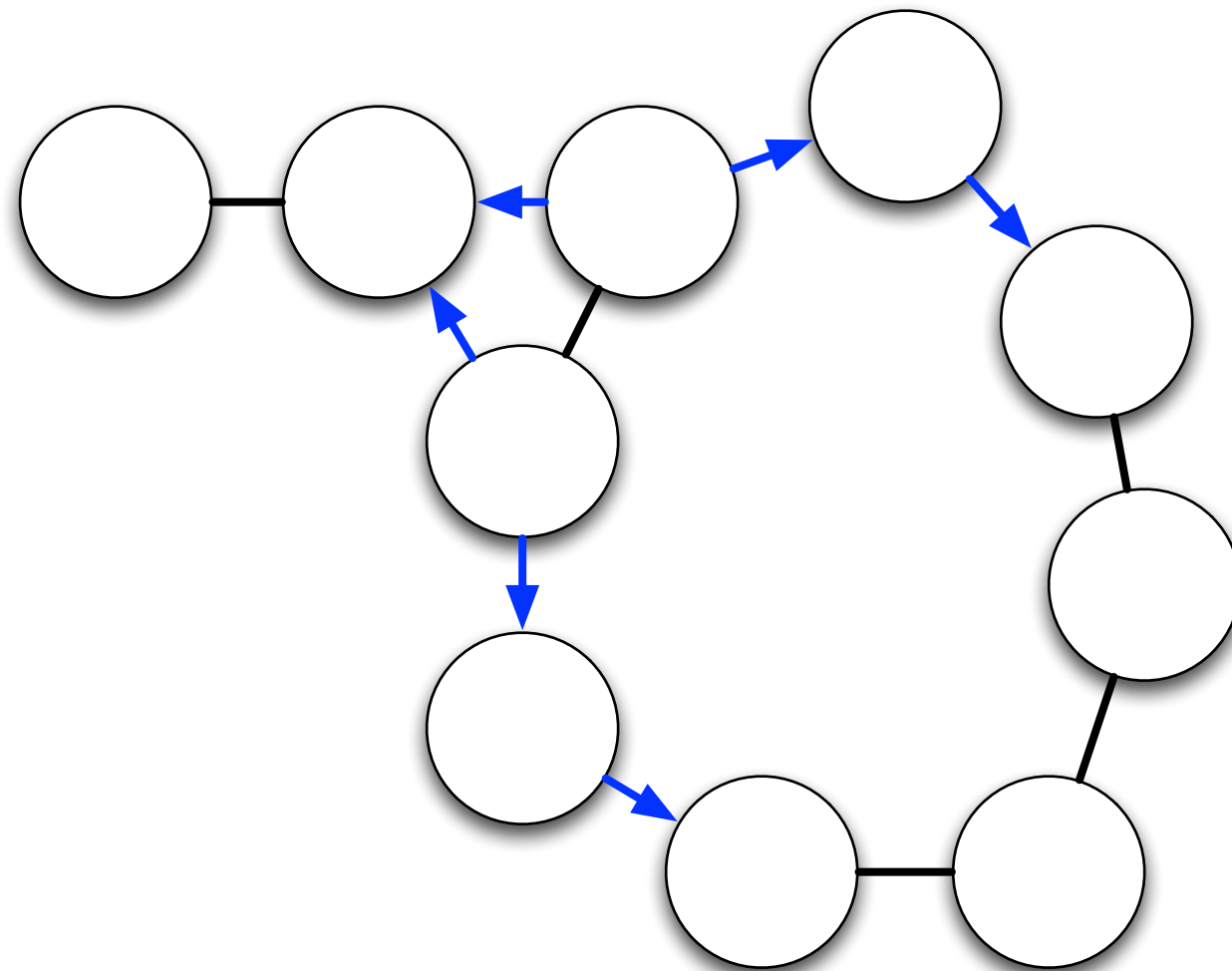
Stateless Flooding



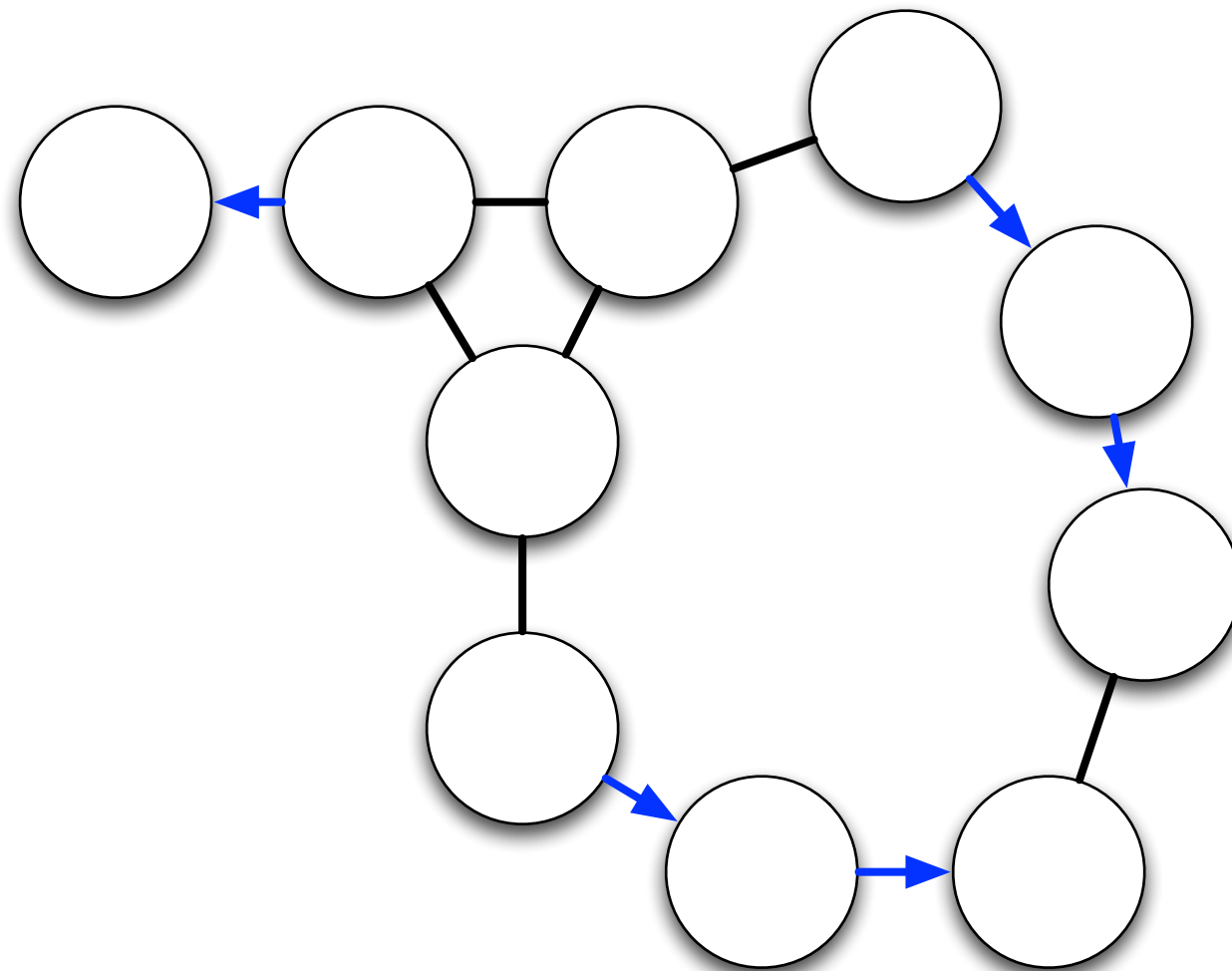
Stateless Flooding



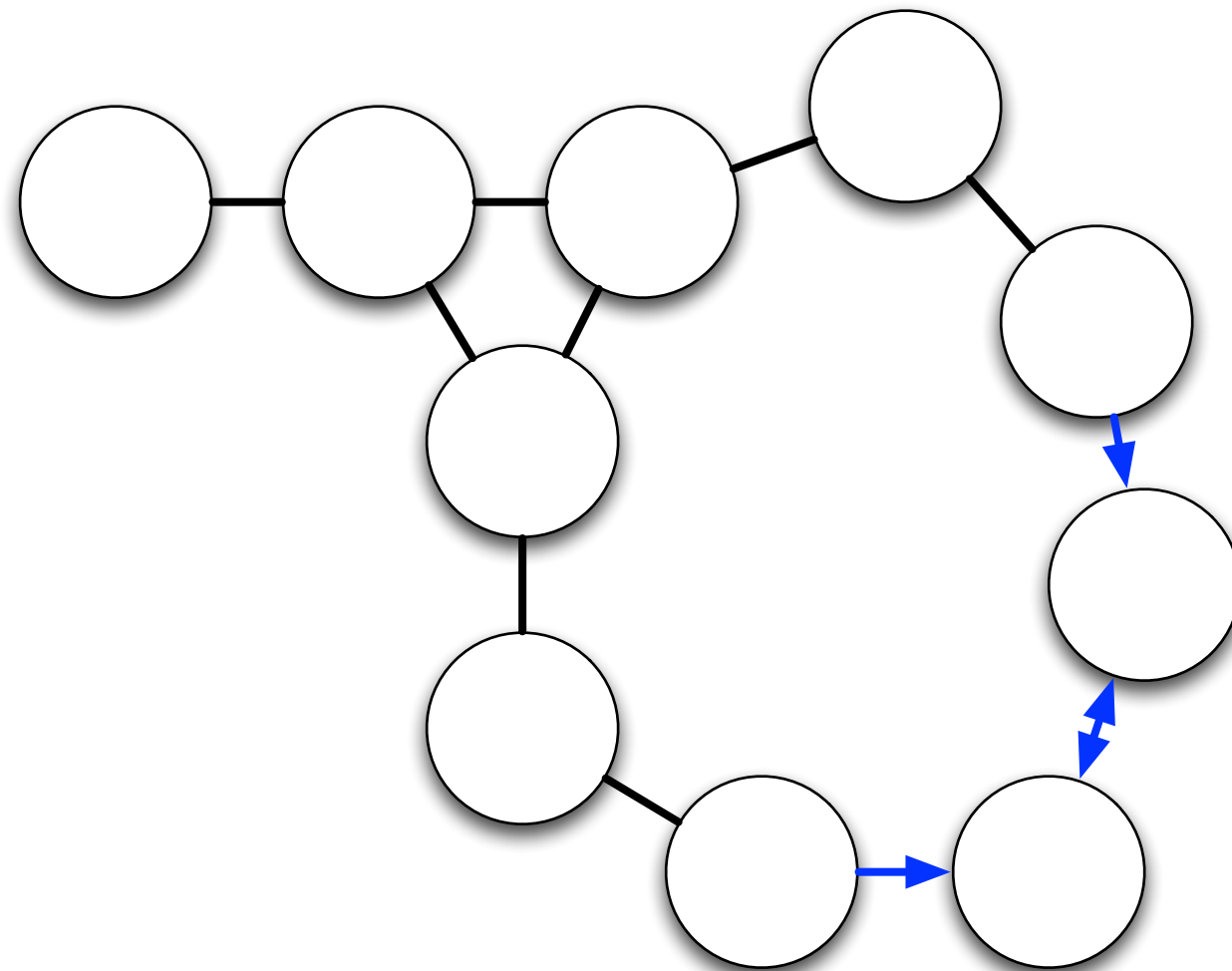
Stateless Flooding



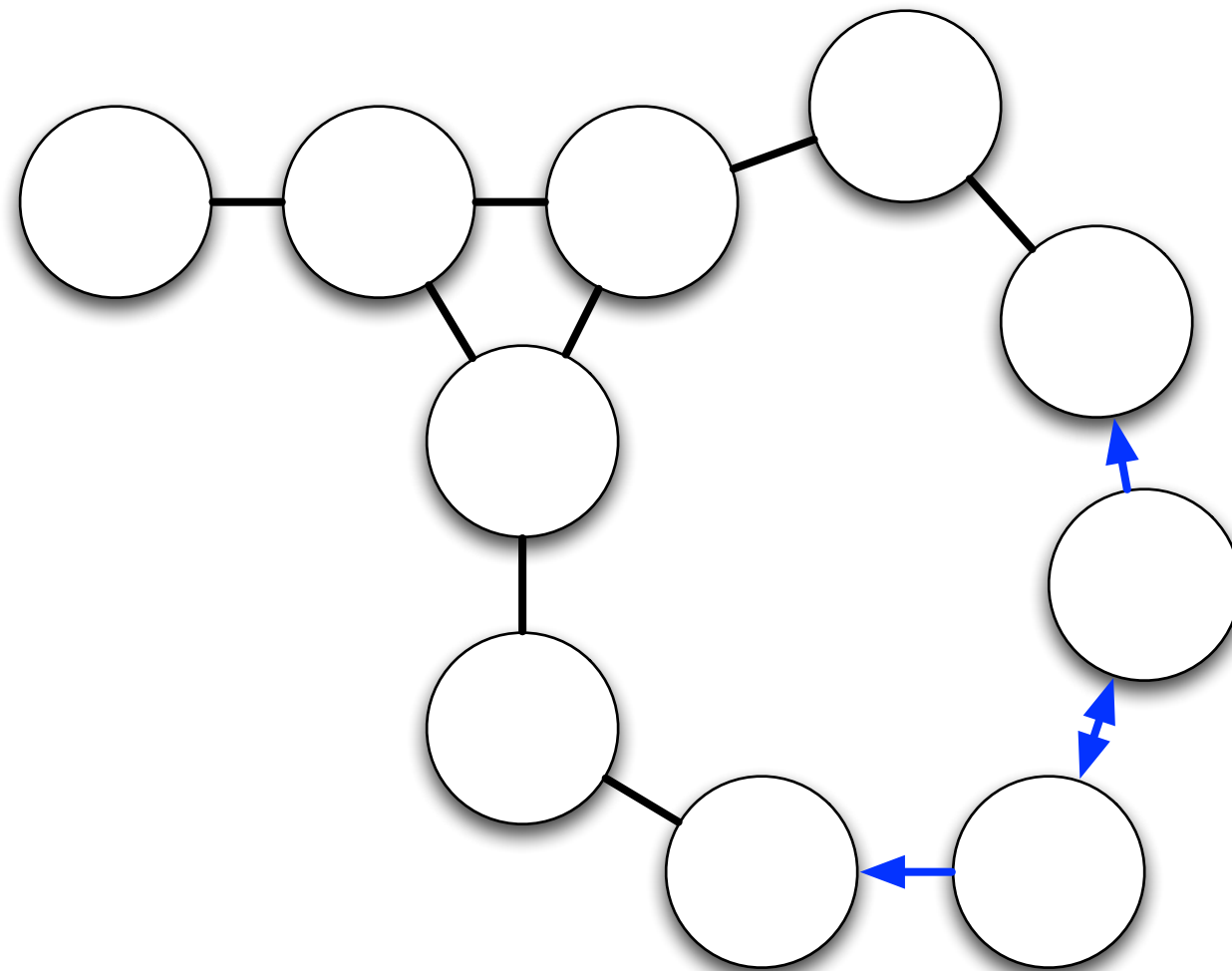
Stateless Flooding



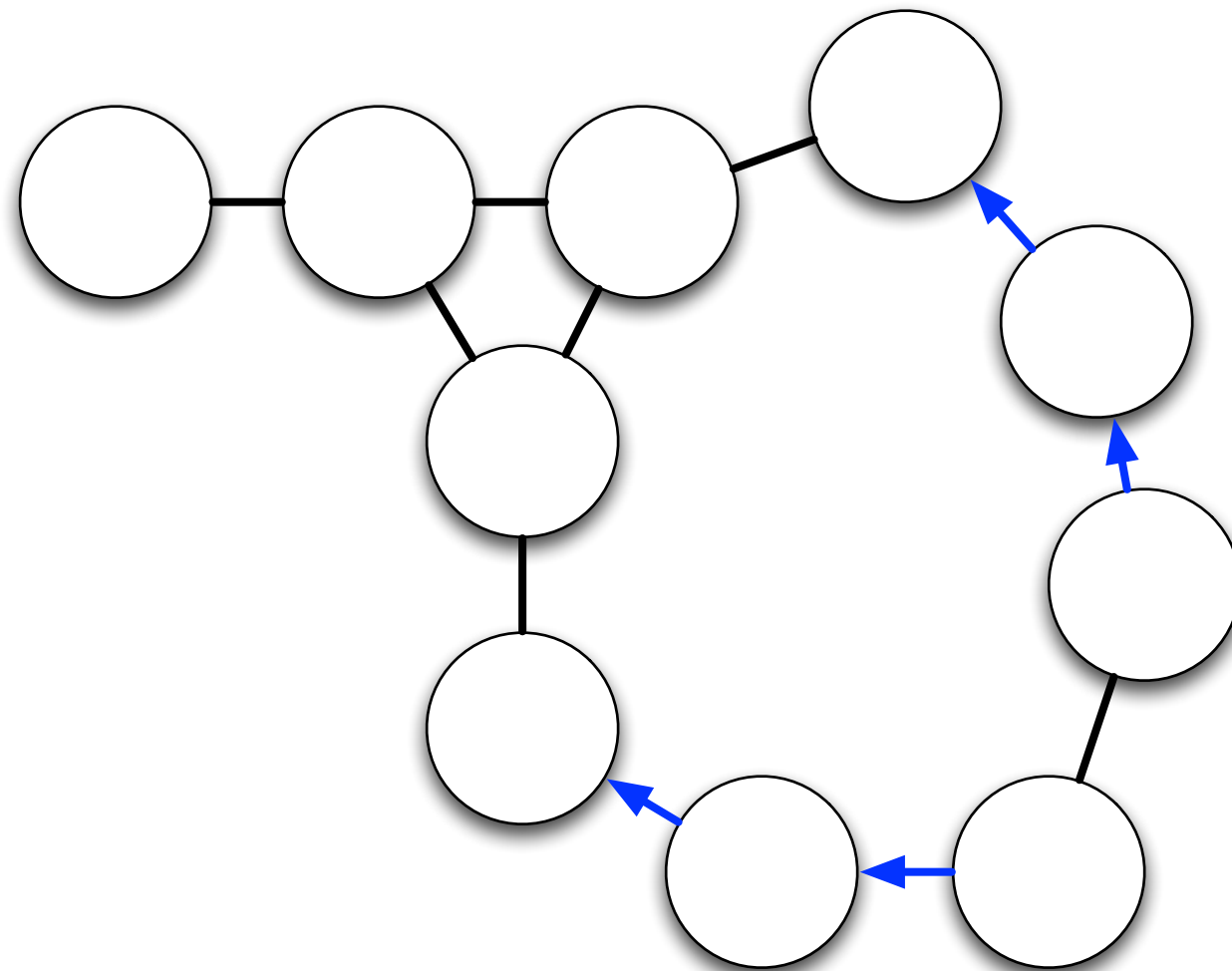
Stateless Flooding



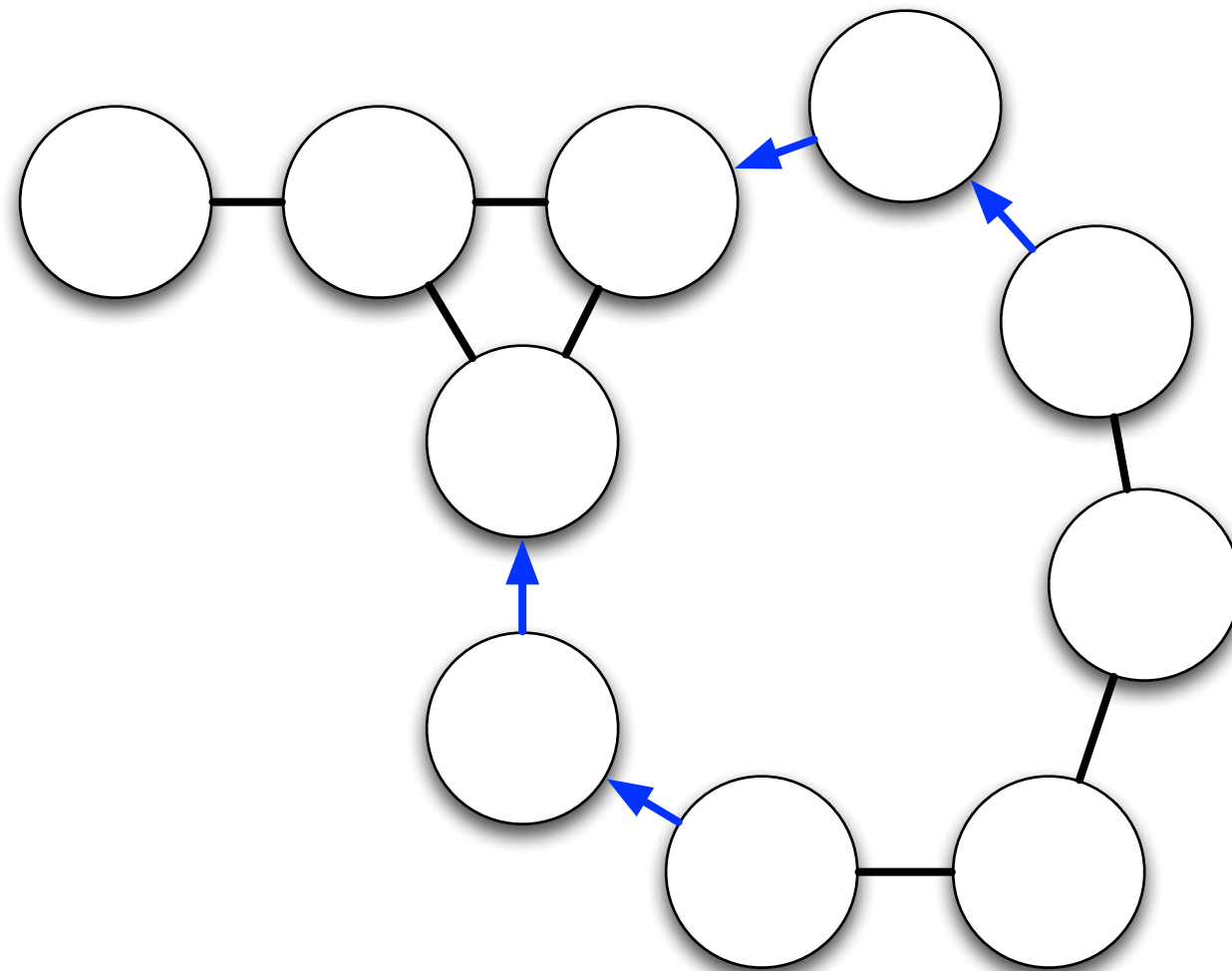
Stateless Flooding



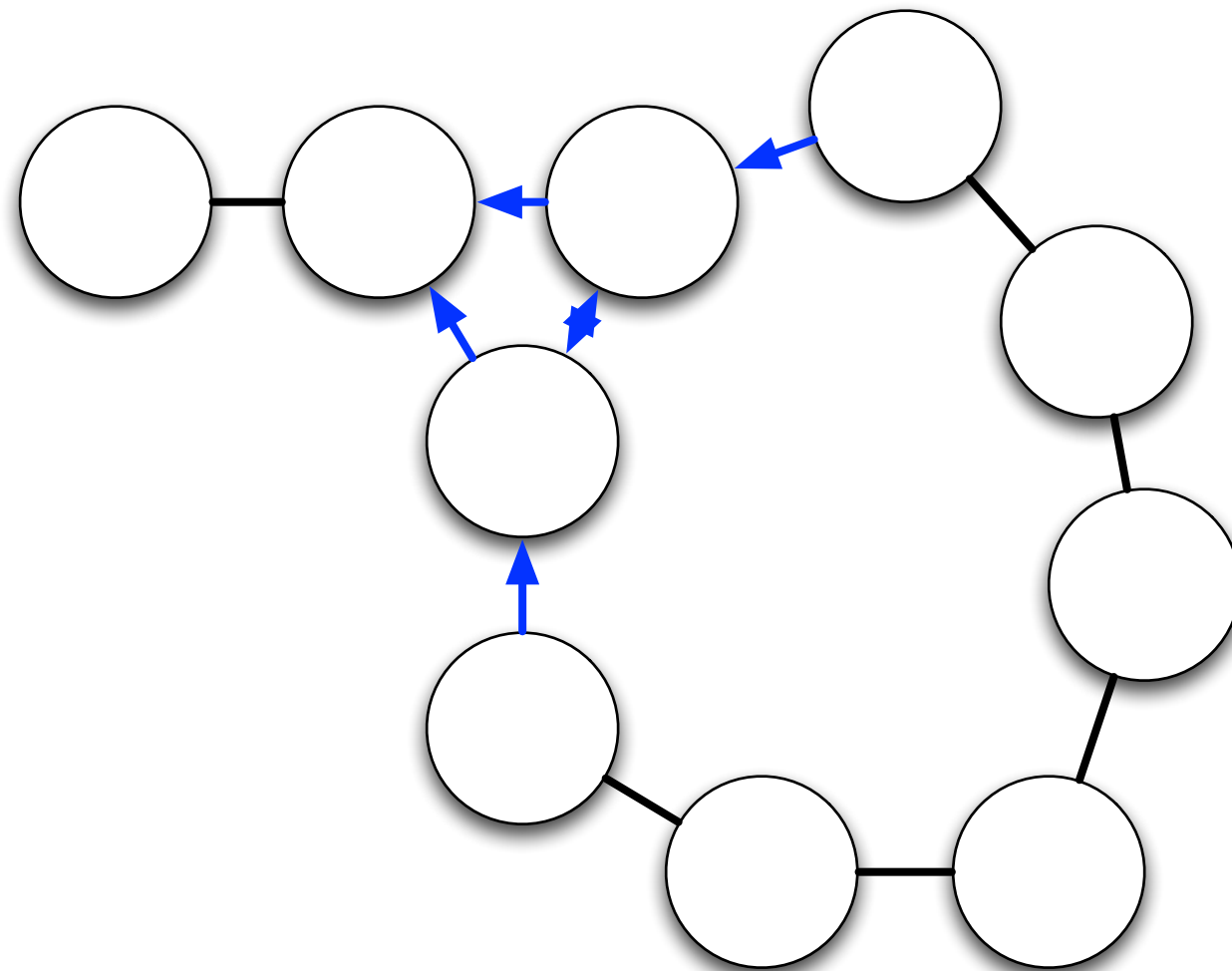
Stateless Flooding



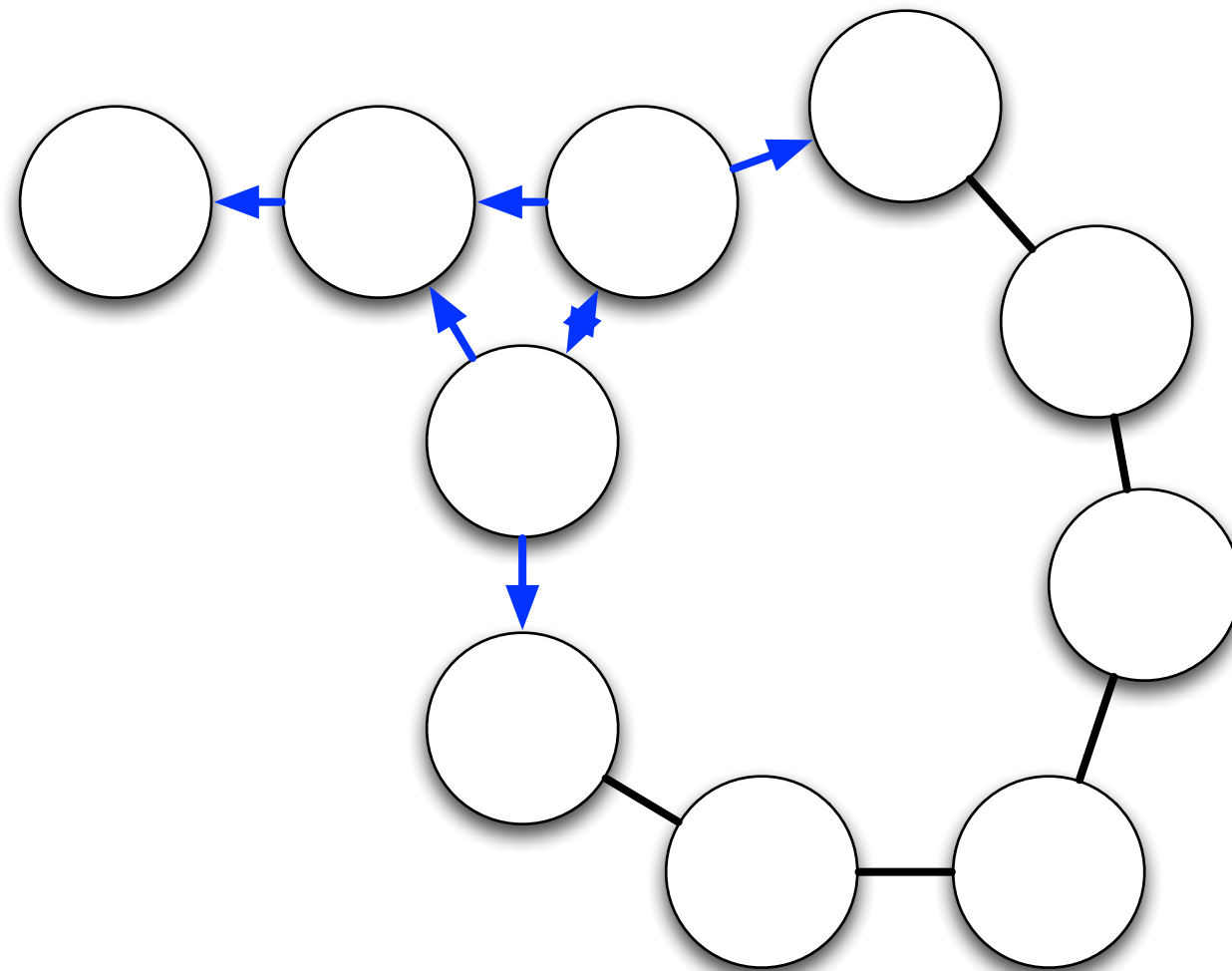
Stateless Flooding



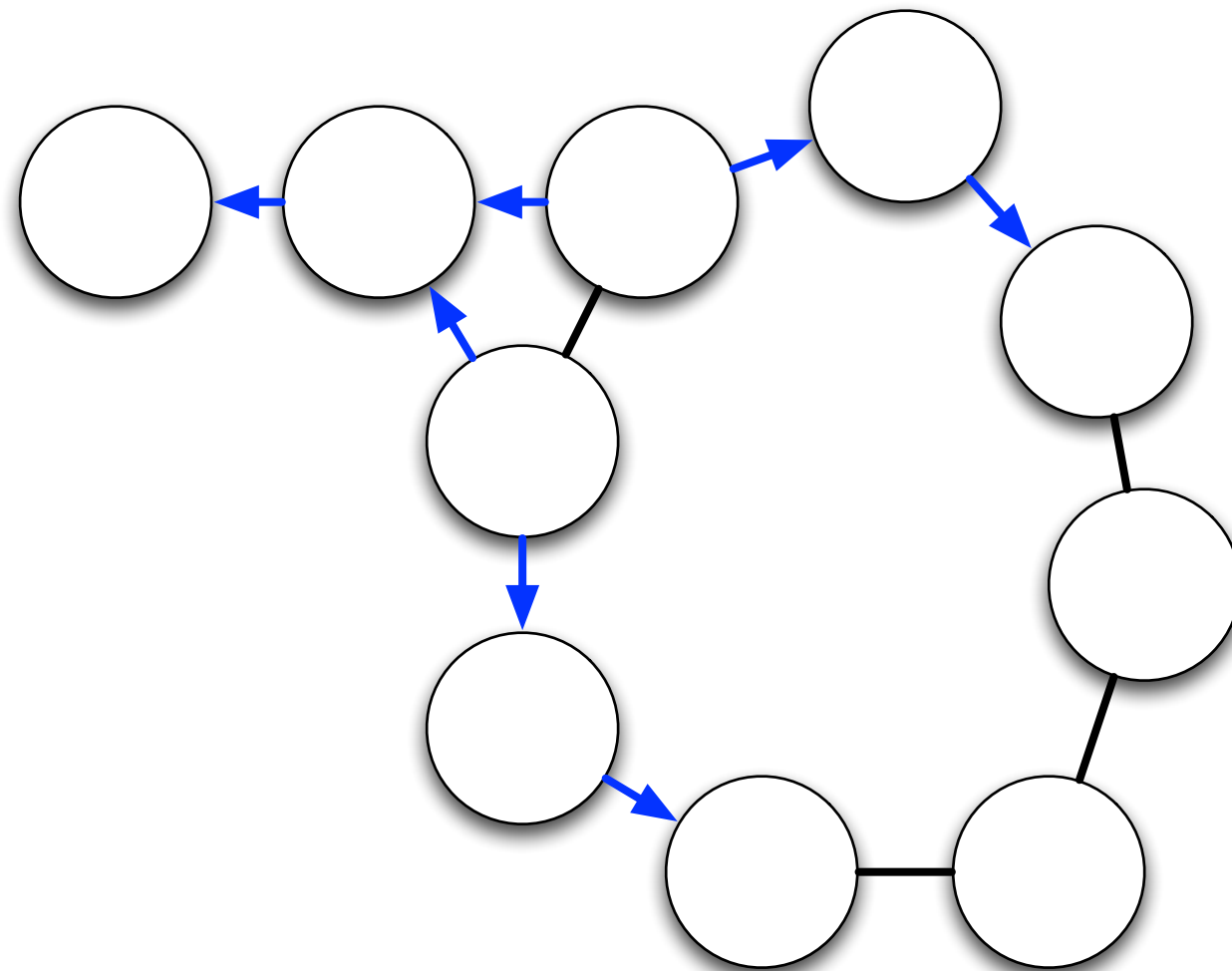
Stateless Flooding



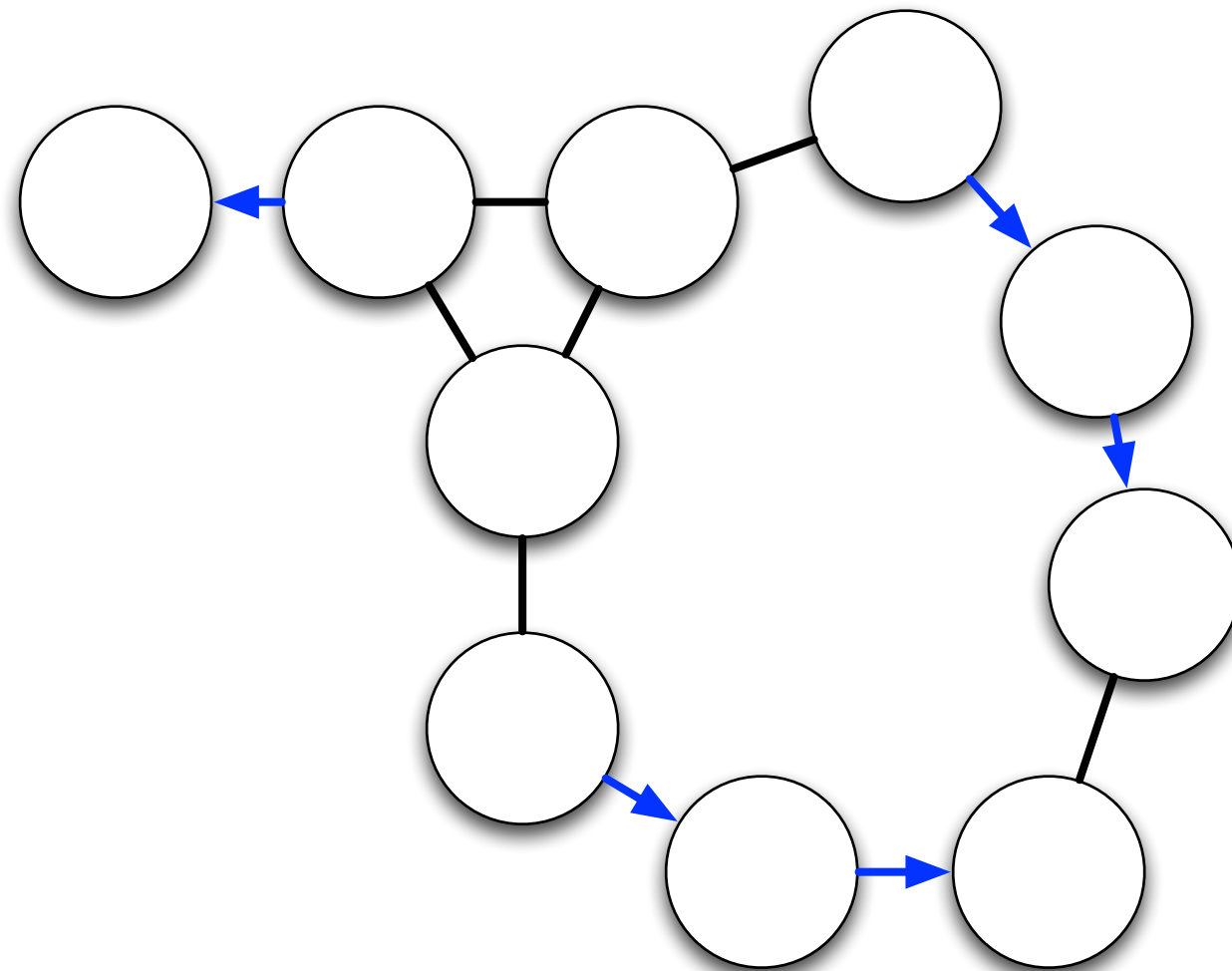
Stateless Flooding



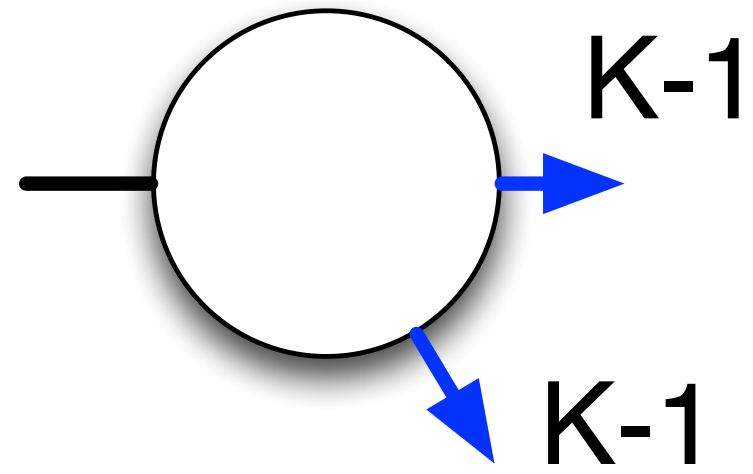
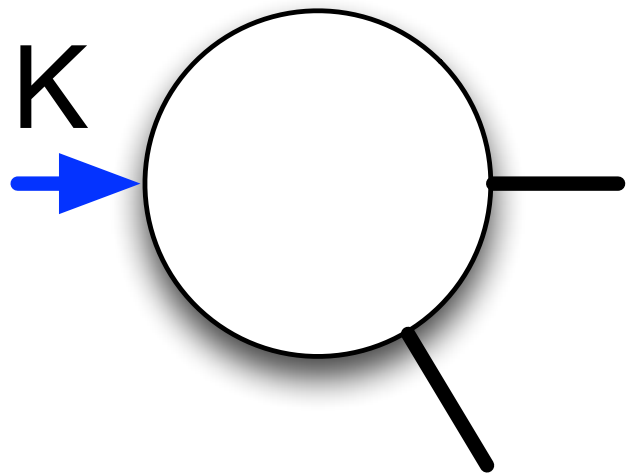
Stateless Flooding



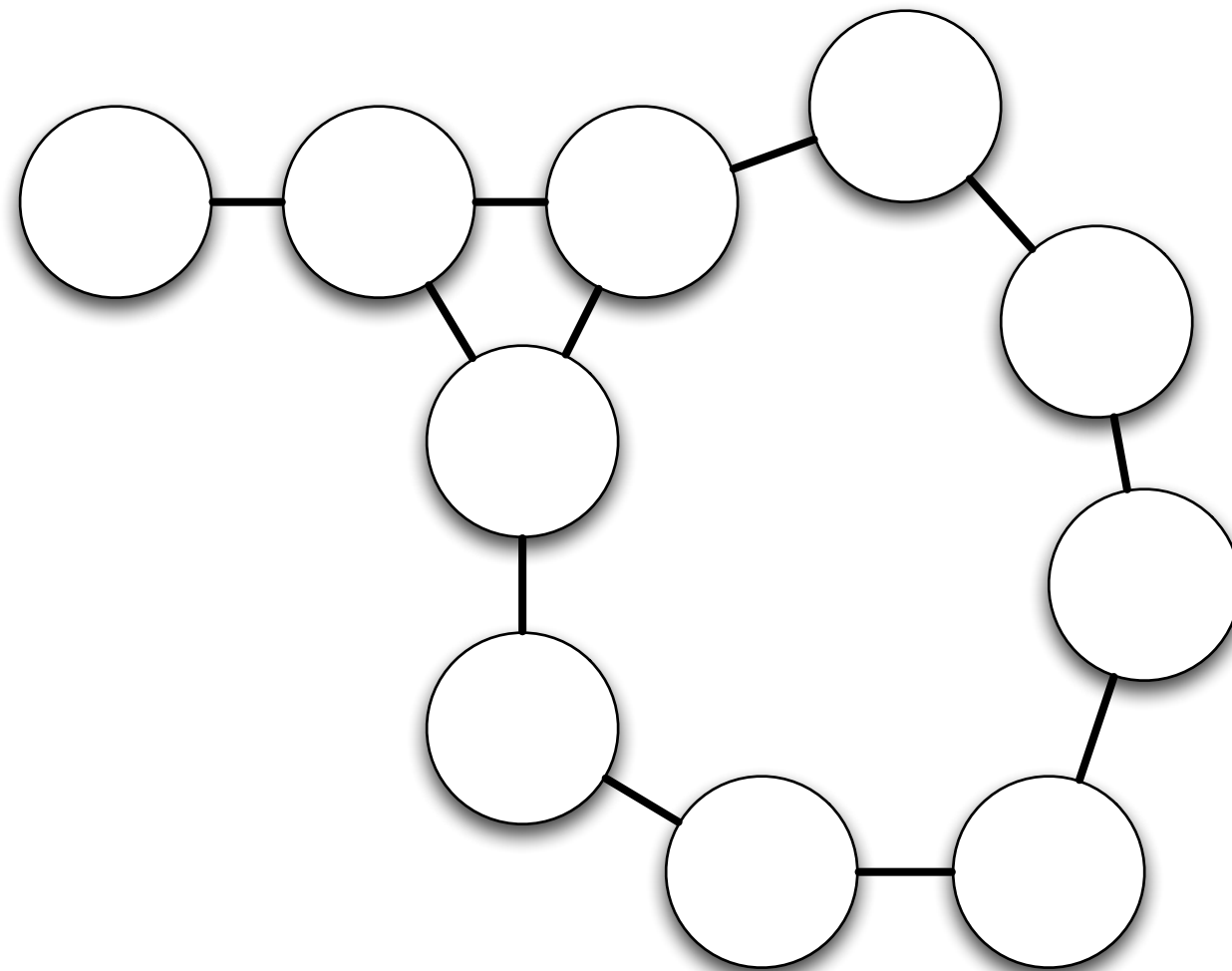
Stateless Flooding



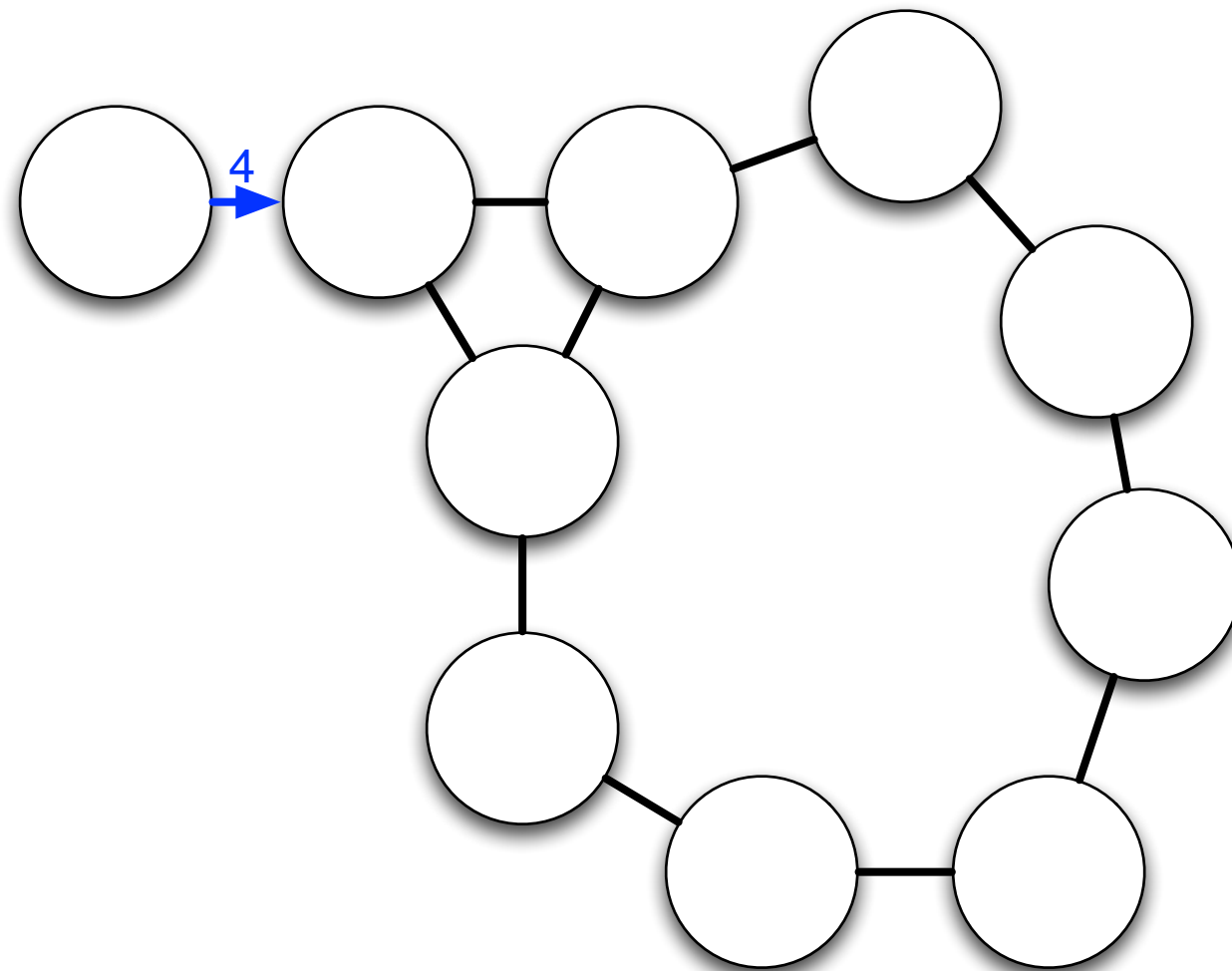
Flooding v2



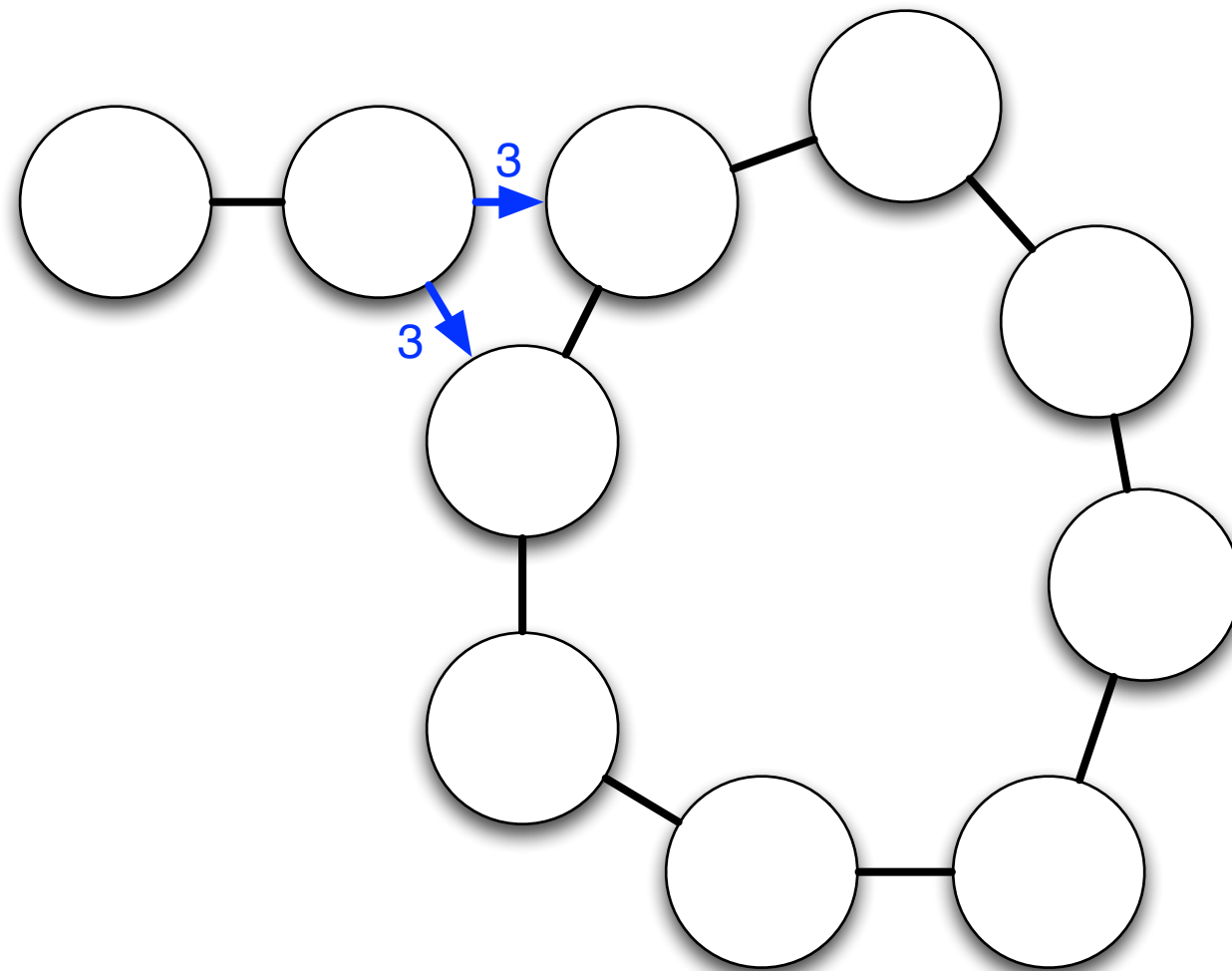
TTL Flooding



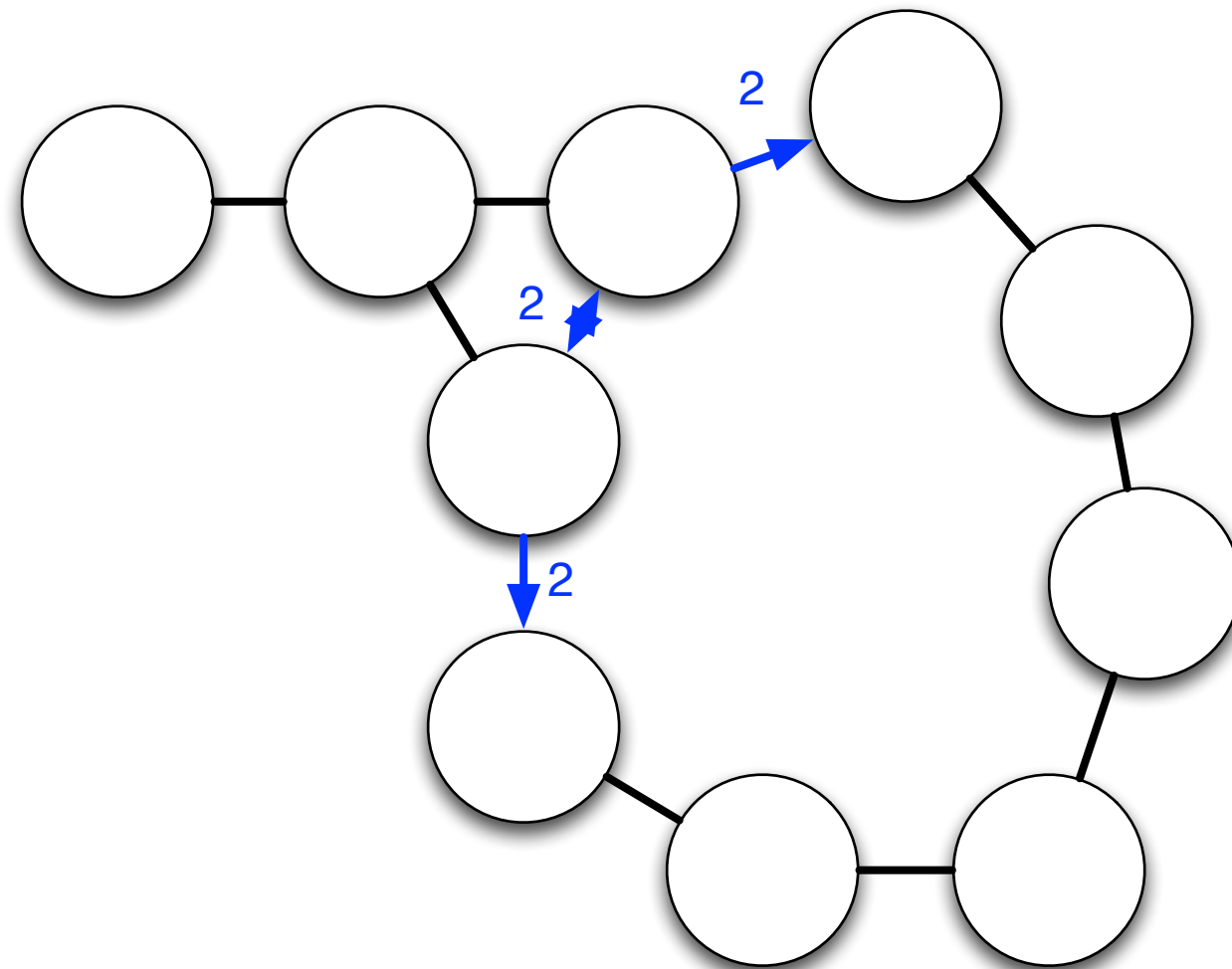
TTL Flooding



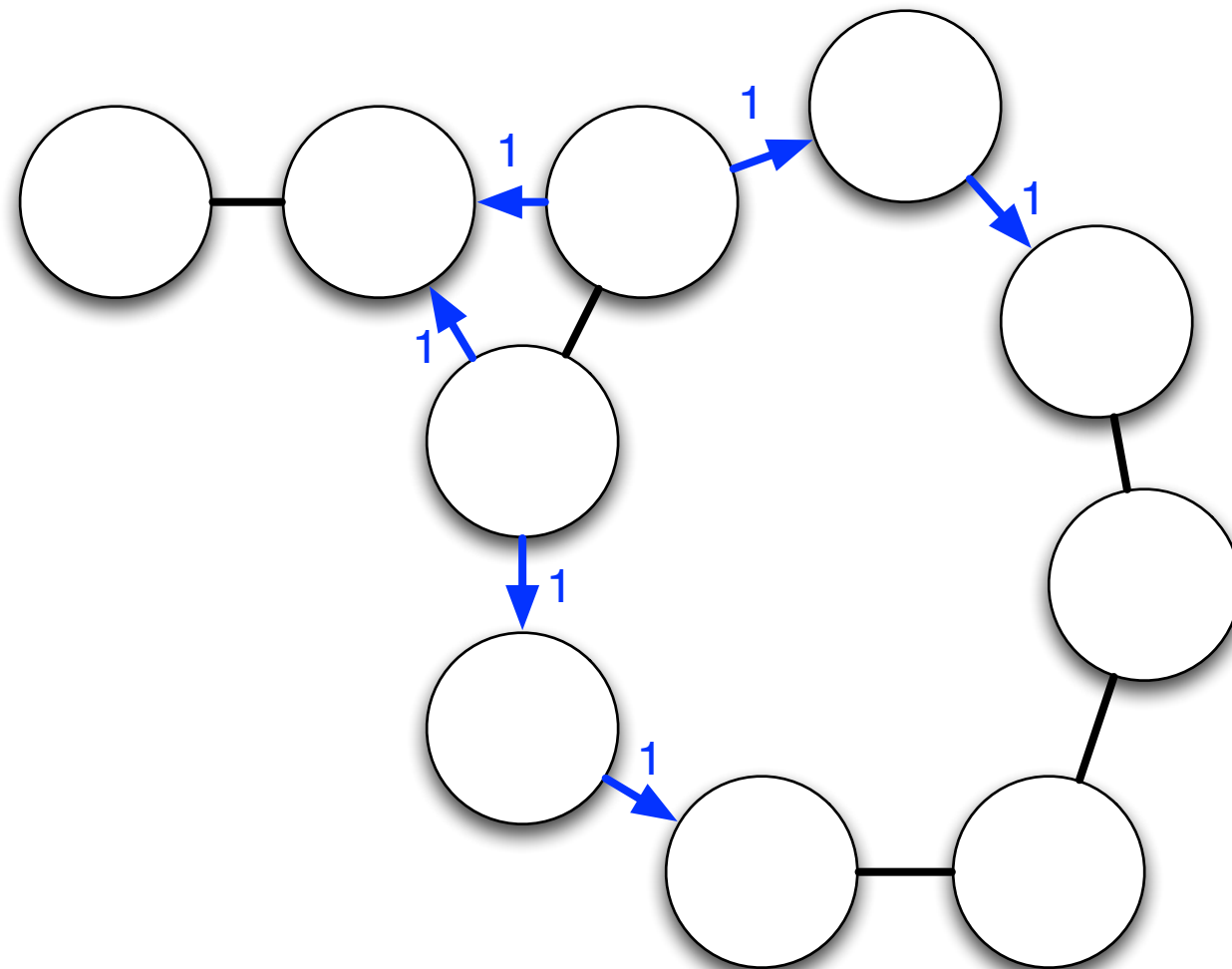
TTL Flooding



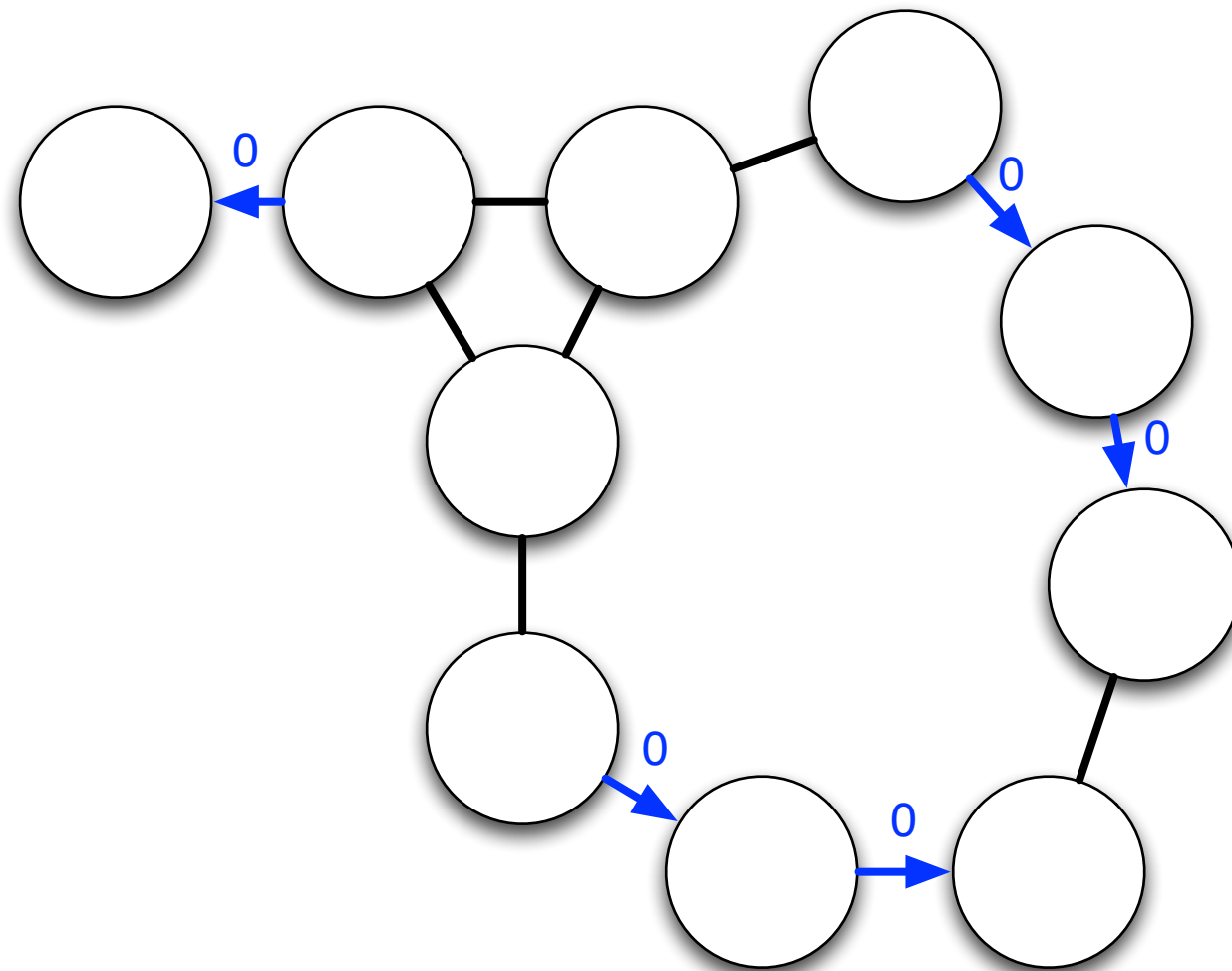
TTL Flooding



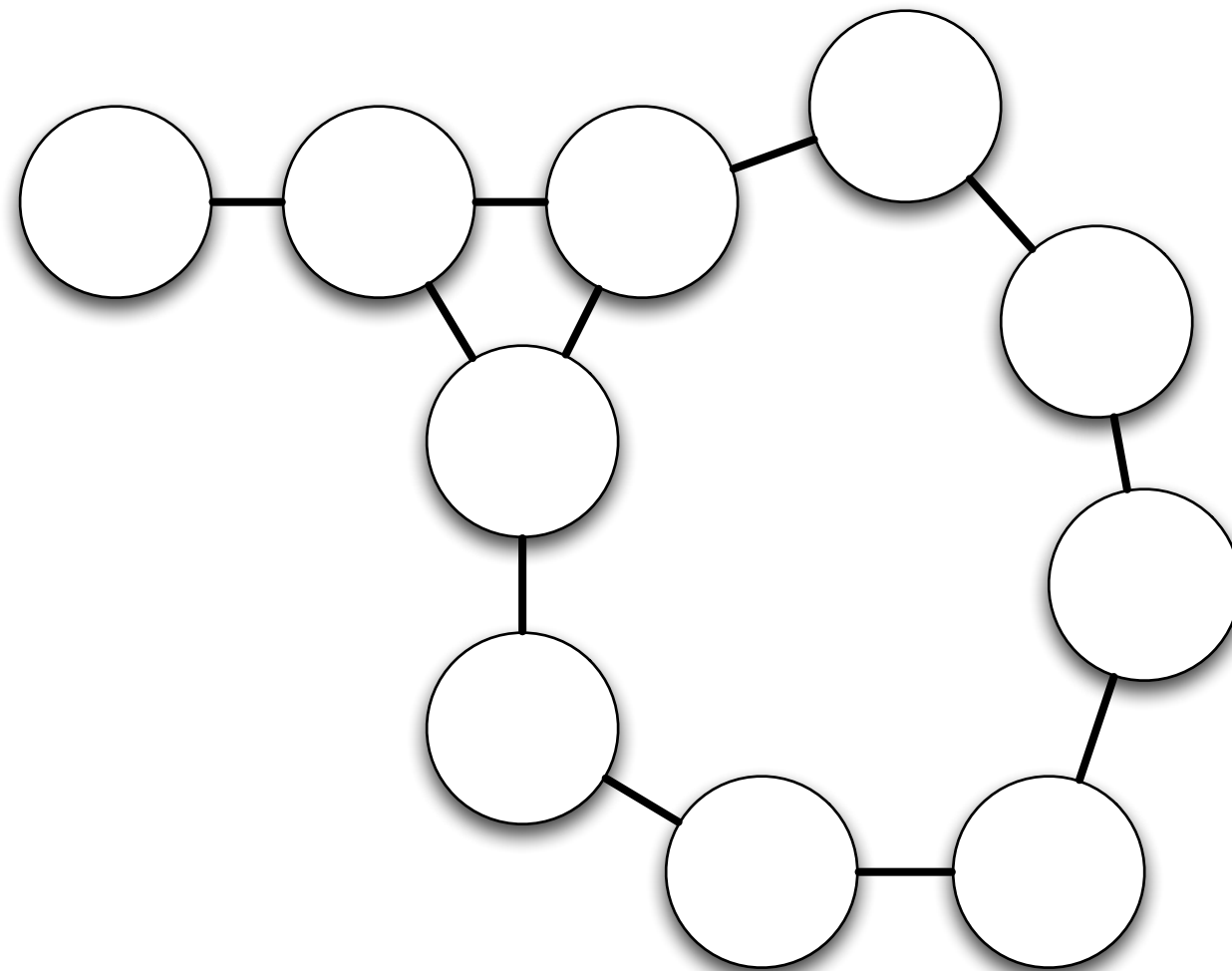
TTL Flooding



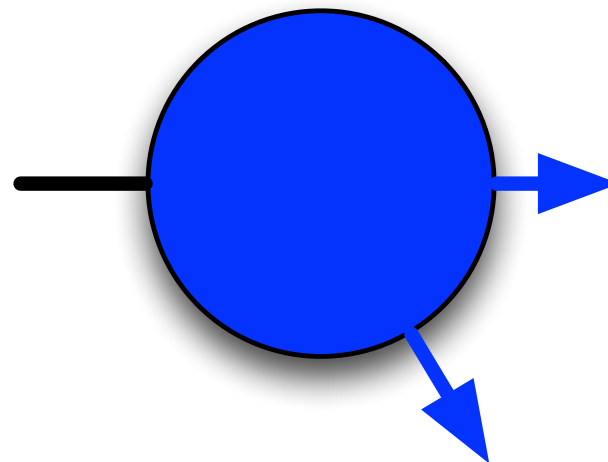
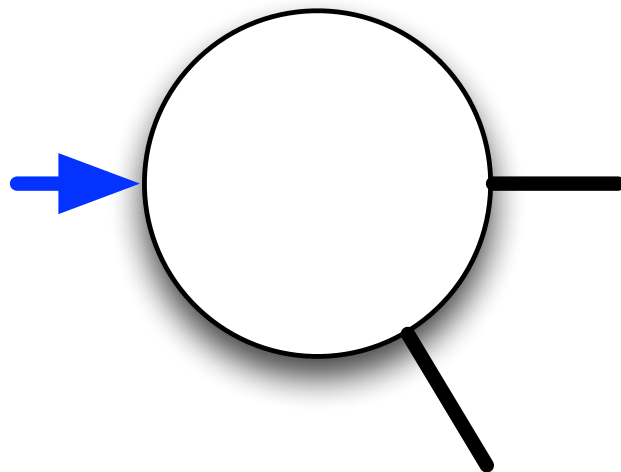
TTL Flooding



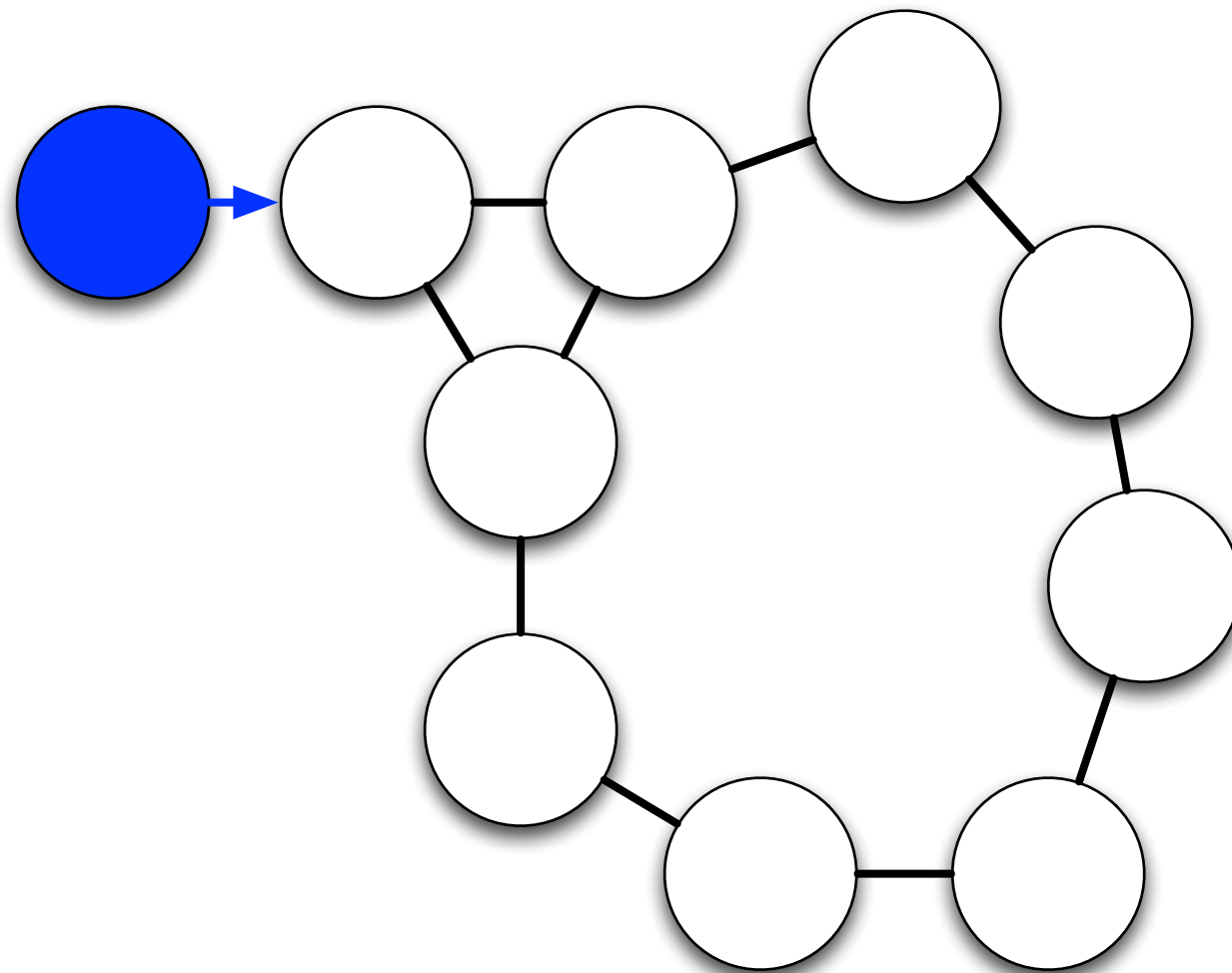
TTL Flooding



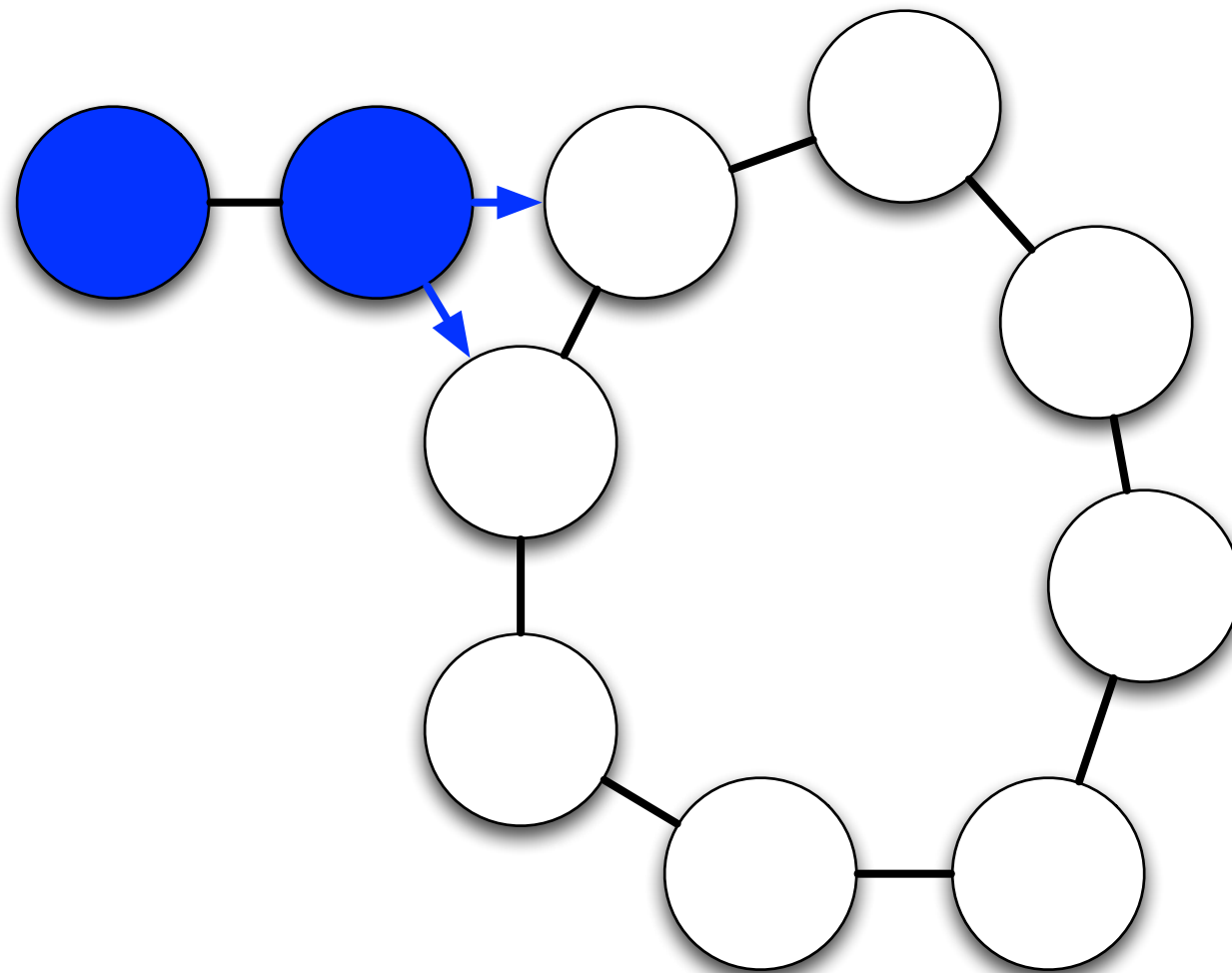
Flooding v3



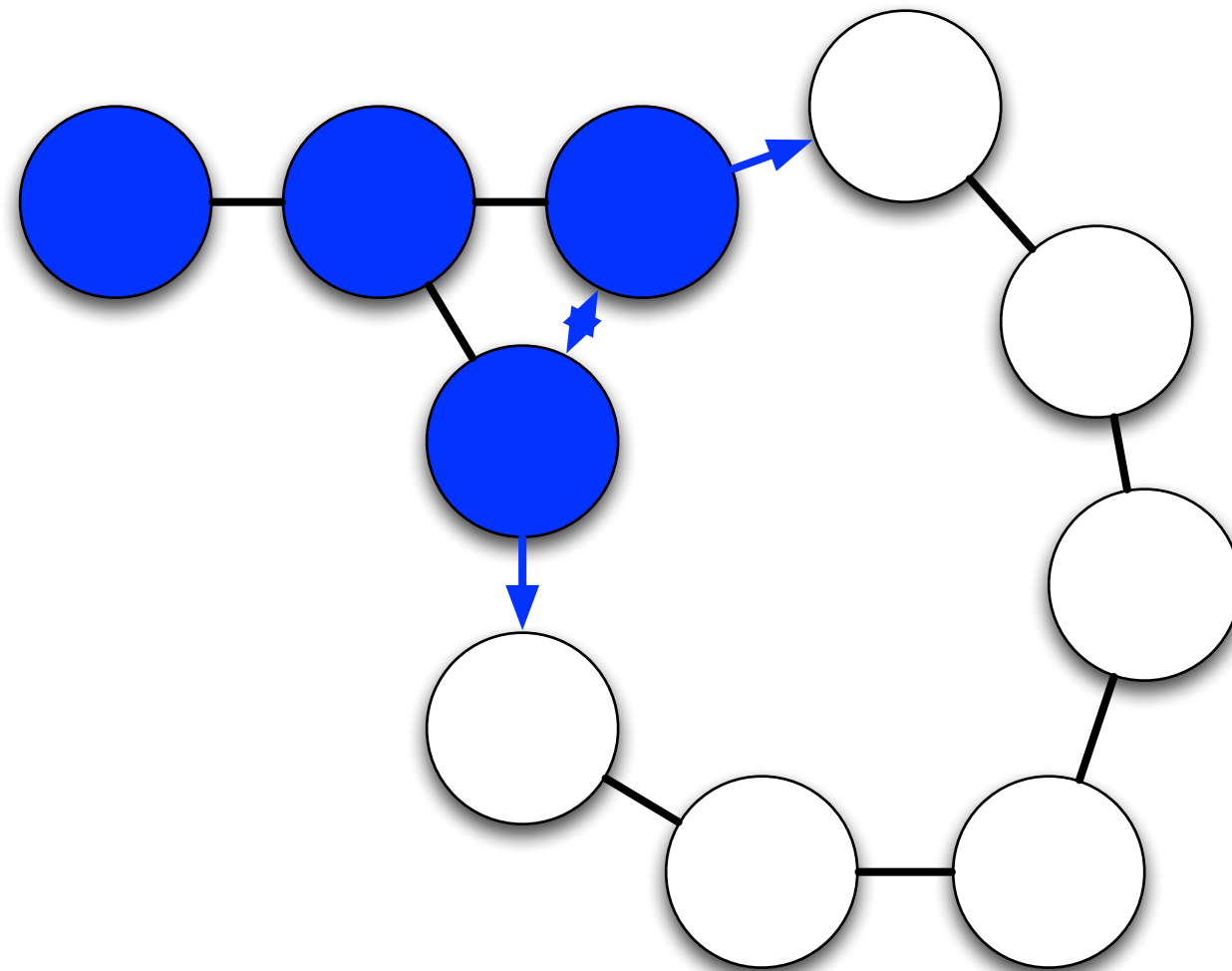
Stateful Flooding



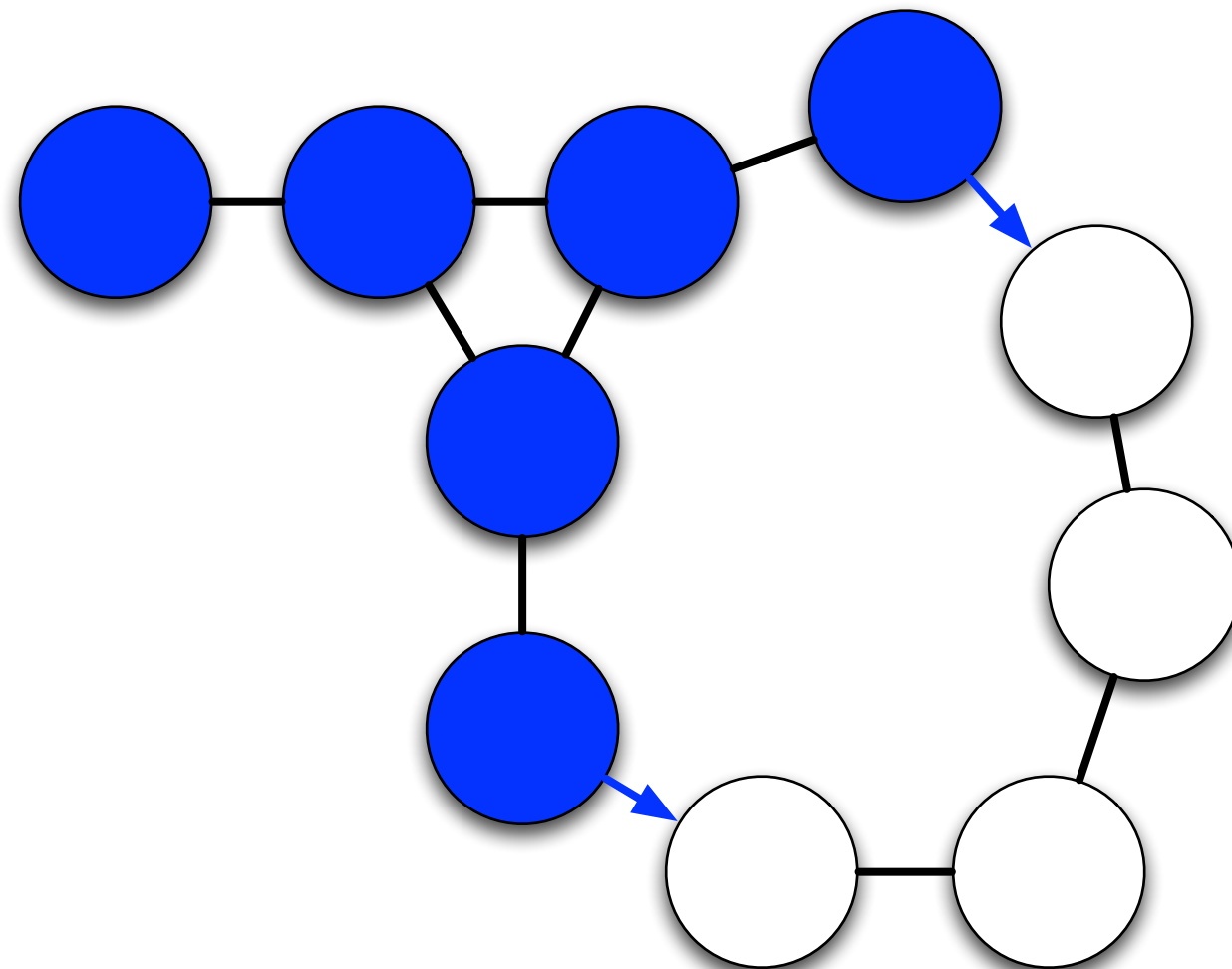
Stateful Flooding



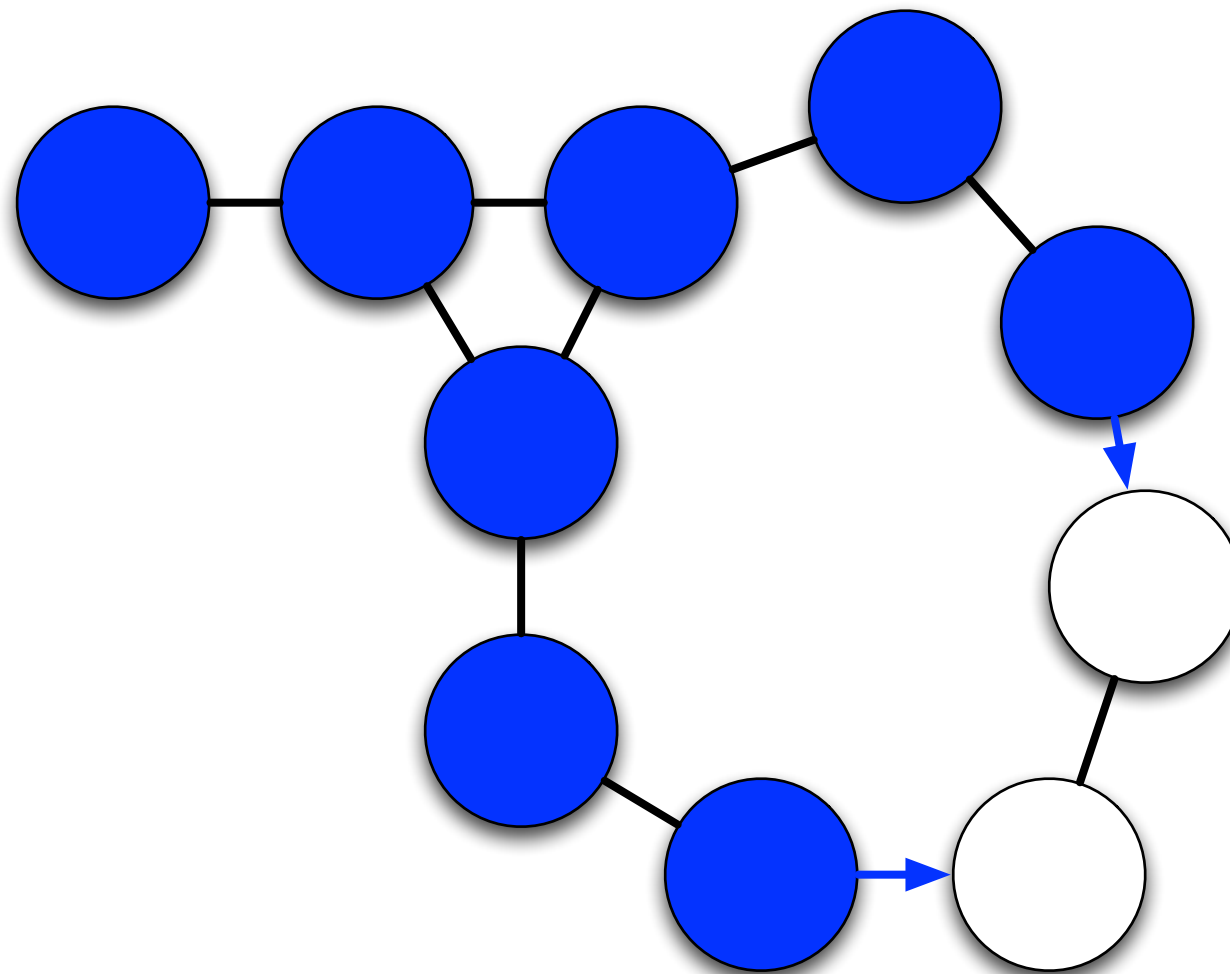
Stateful Flooding



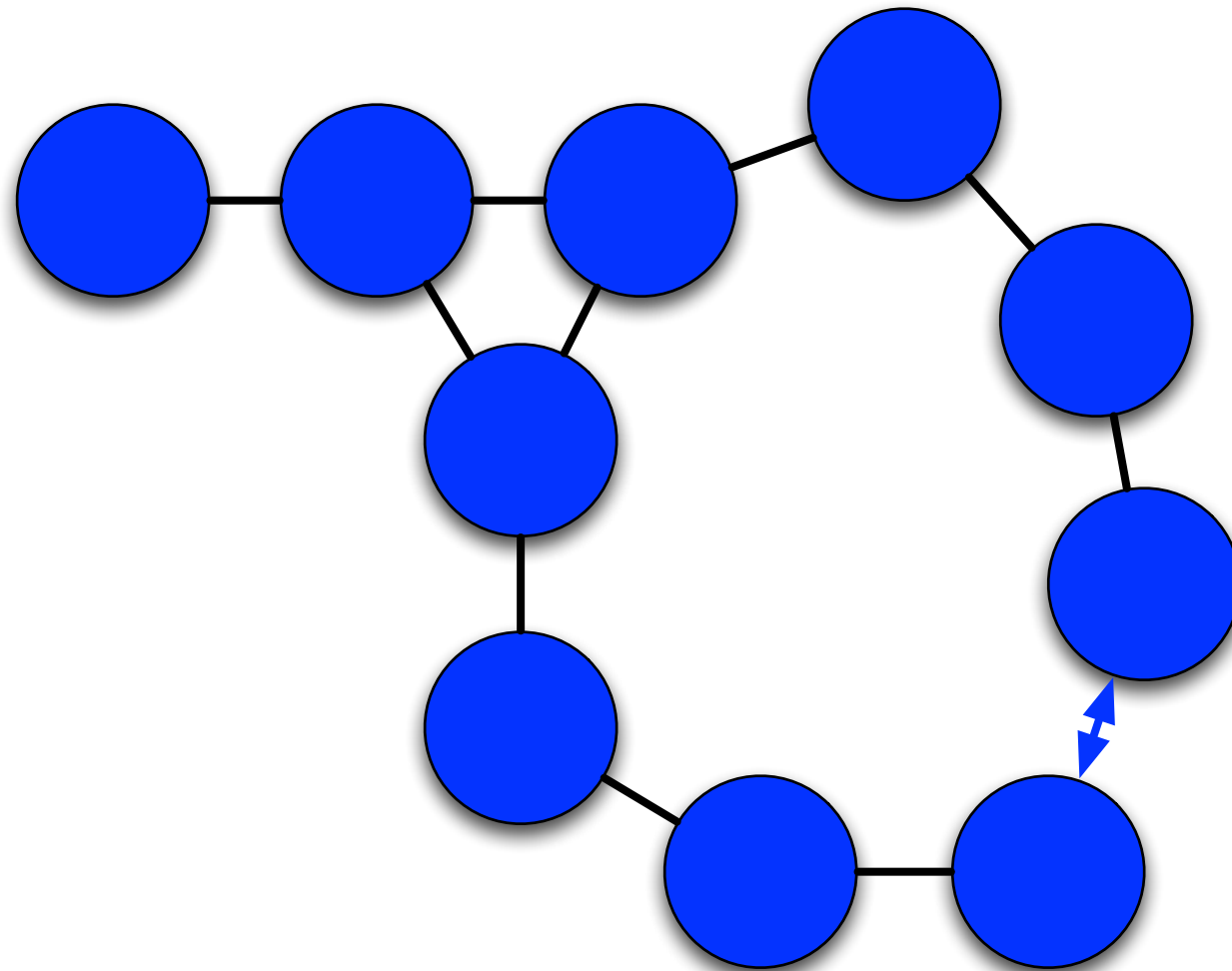
Stateful Flooding



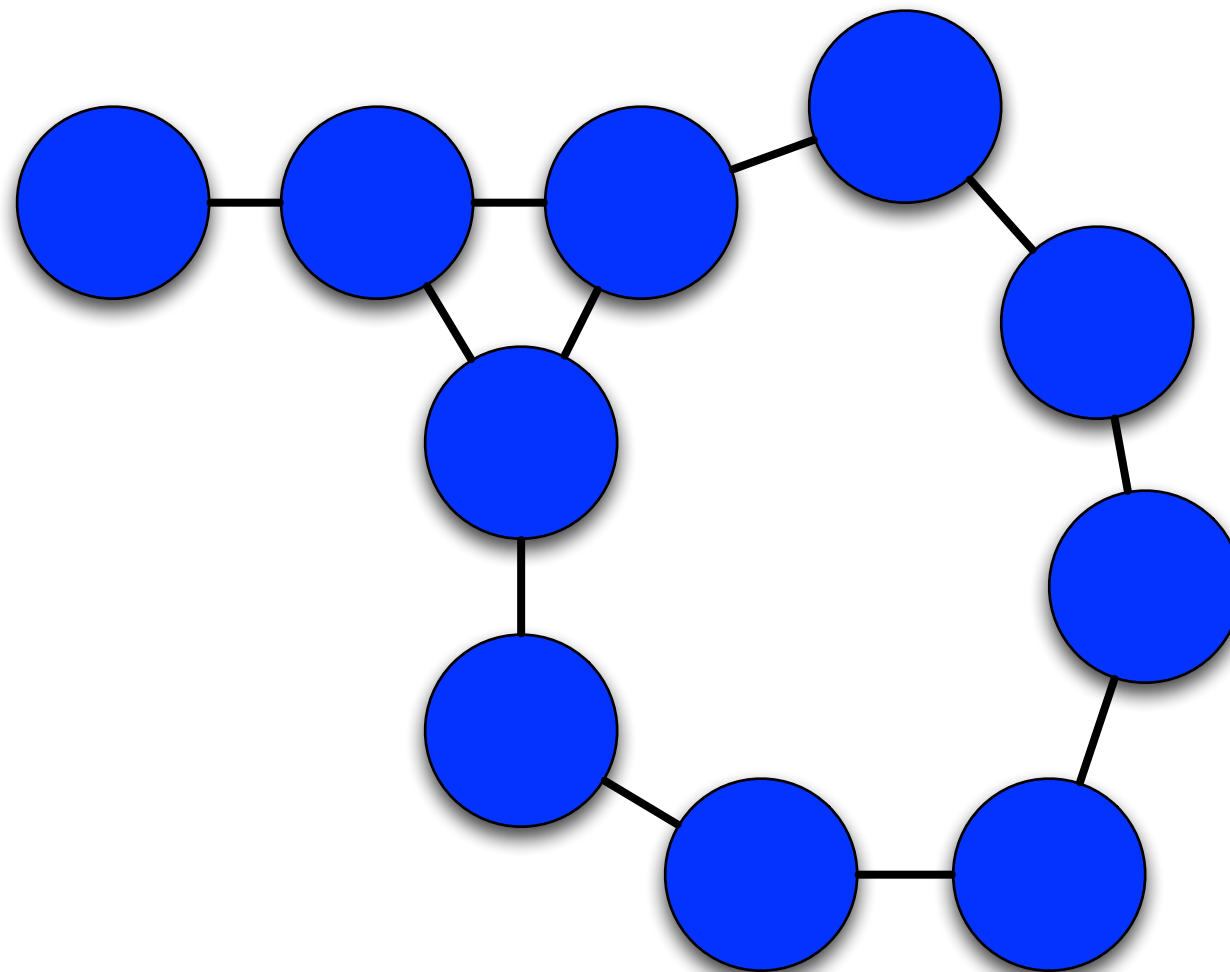
Stateful Flooding



Stateful Flooding



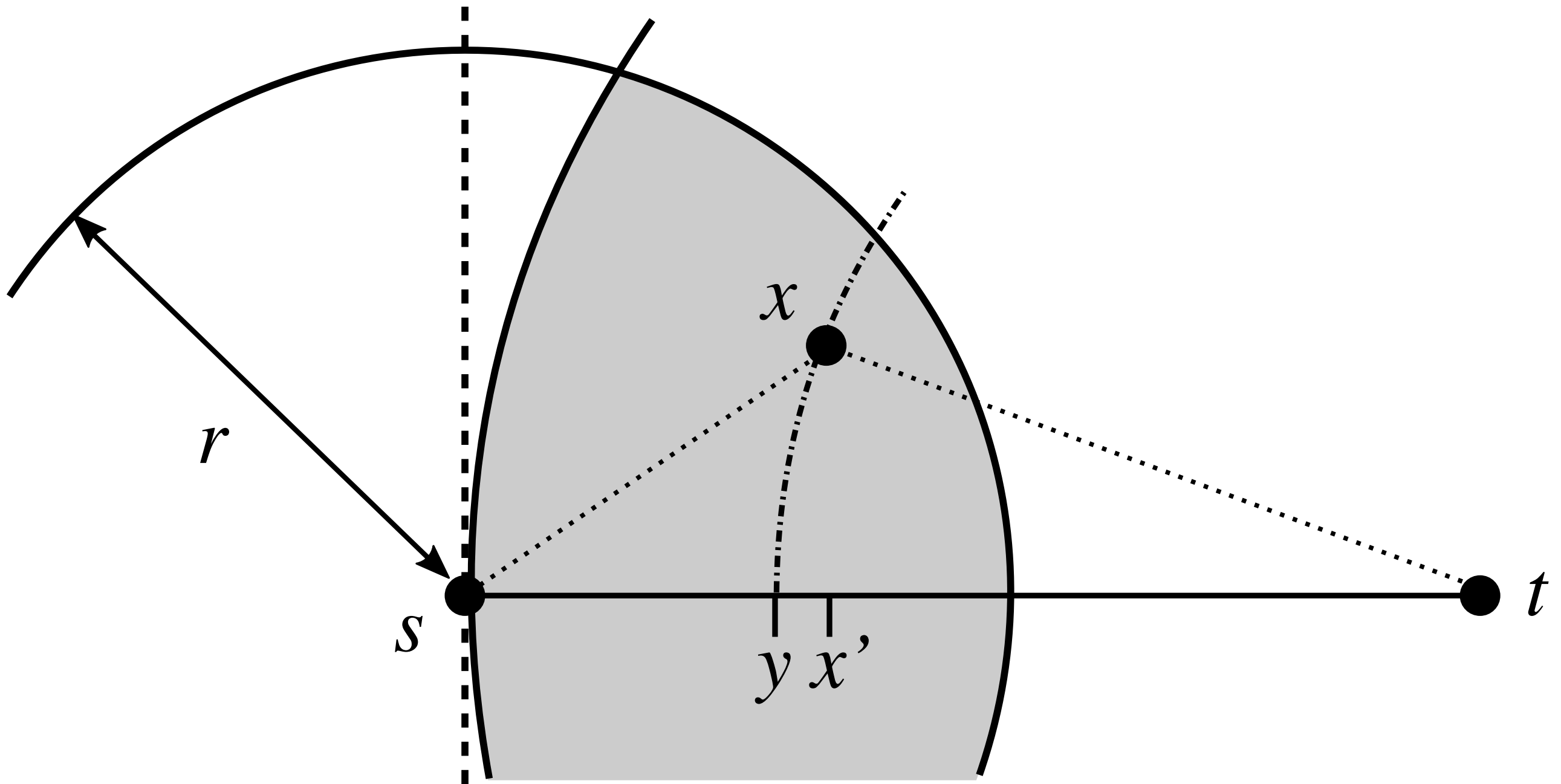
Stateful Flooding



Geometric Routing

- Each node is aware of its *coordinates* (and those of its neighbors)
- The message contains the coordinates of the destination
- **Goal:** deliver the message to the destination *without routing tables*

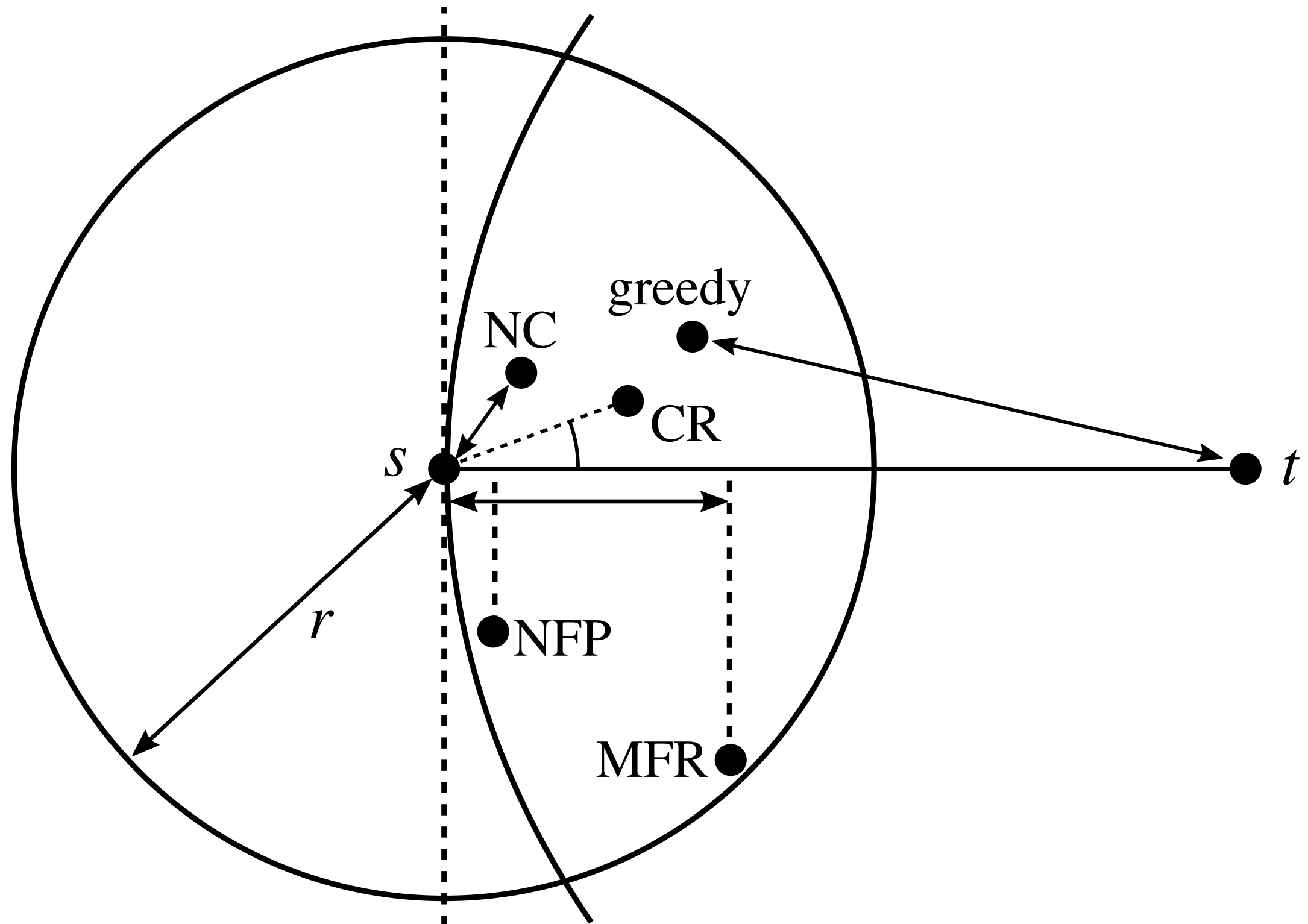
Progress vs. Distance



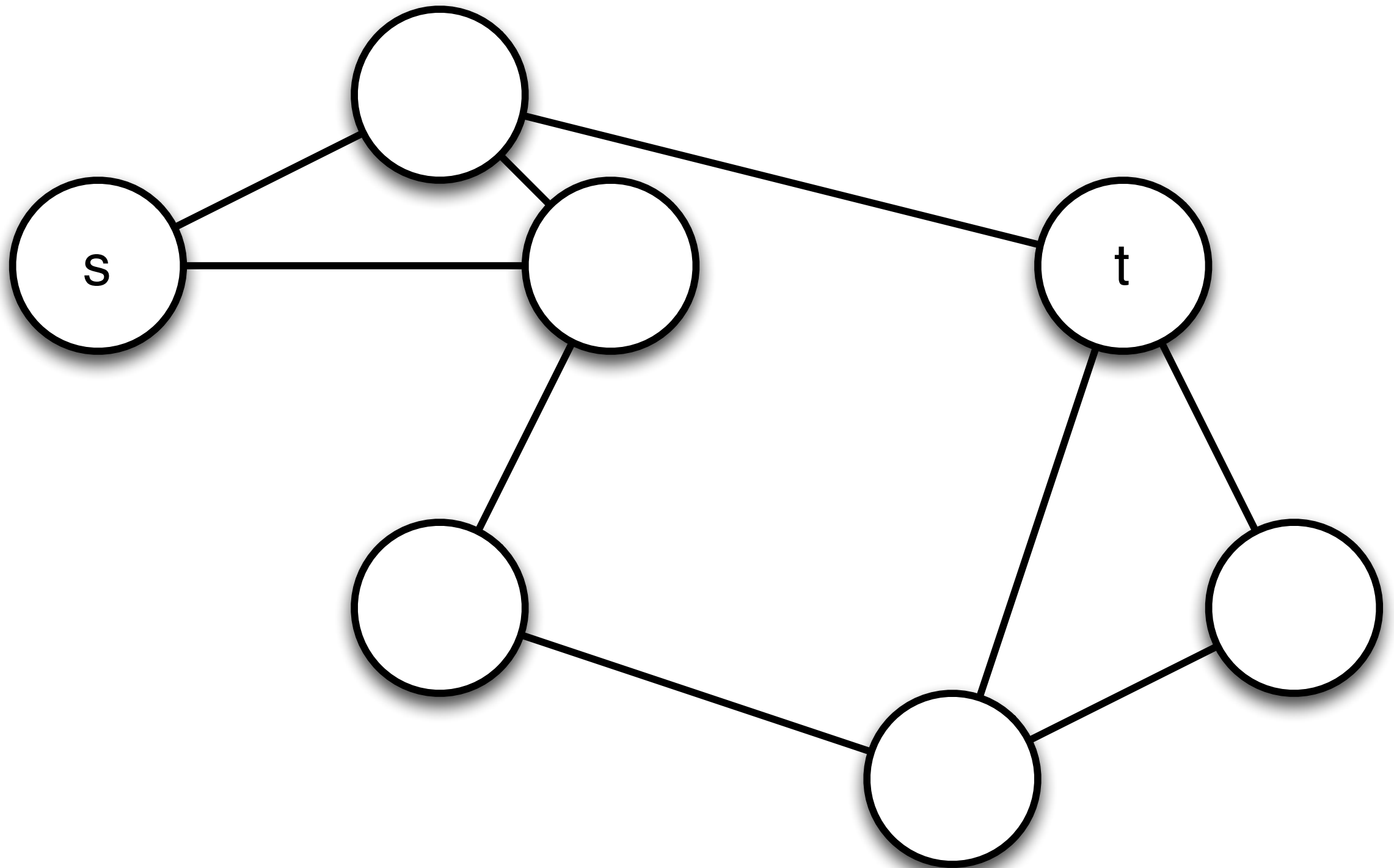
Which Criterion?

- **MFR**: most forwarding progress
- **CR**: minimize angular criterion
- **Greedy**: minimize distance to destination
- **NC**: nearest closer
- **NFP**: nearest with forwarding progress

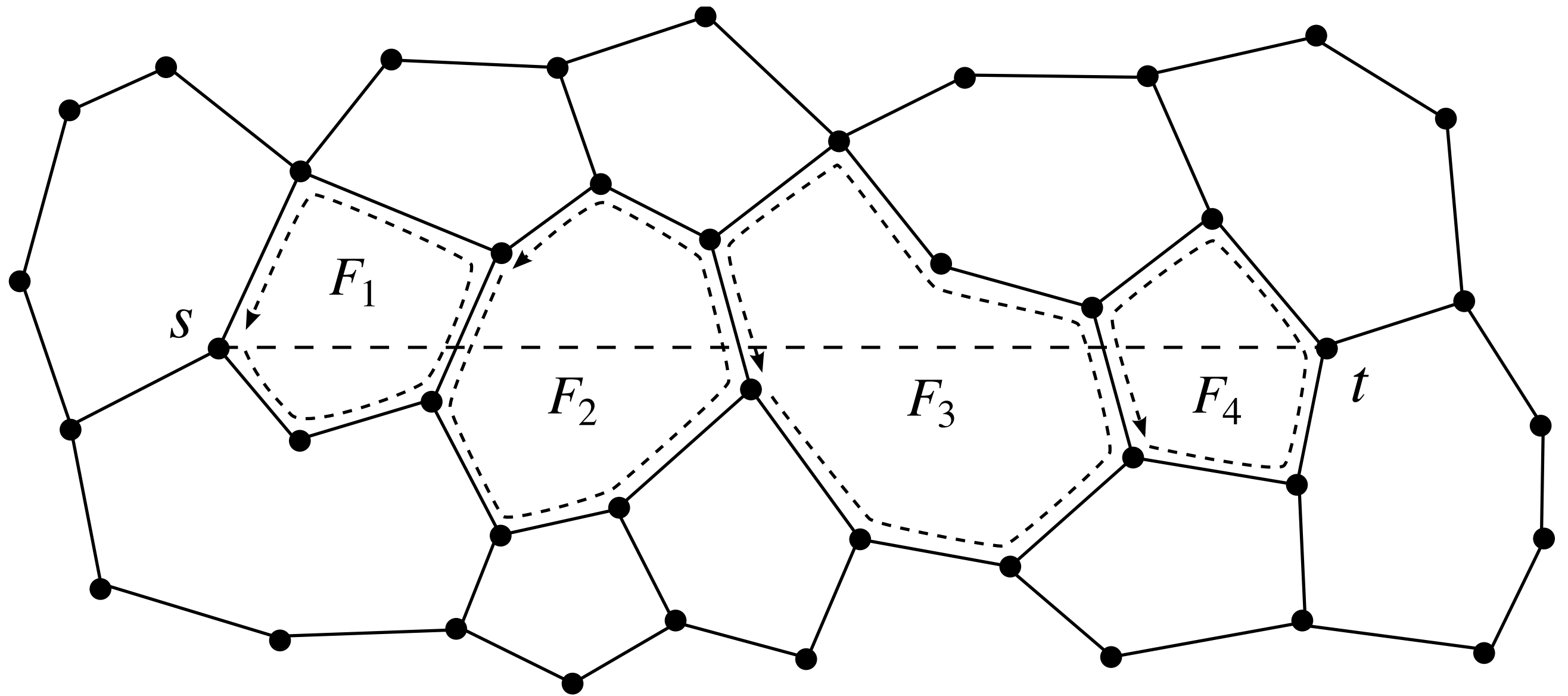
Which Criterion?



Delivery Guarantee?

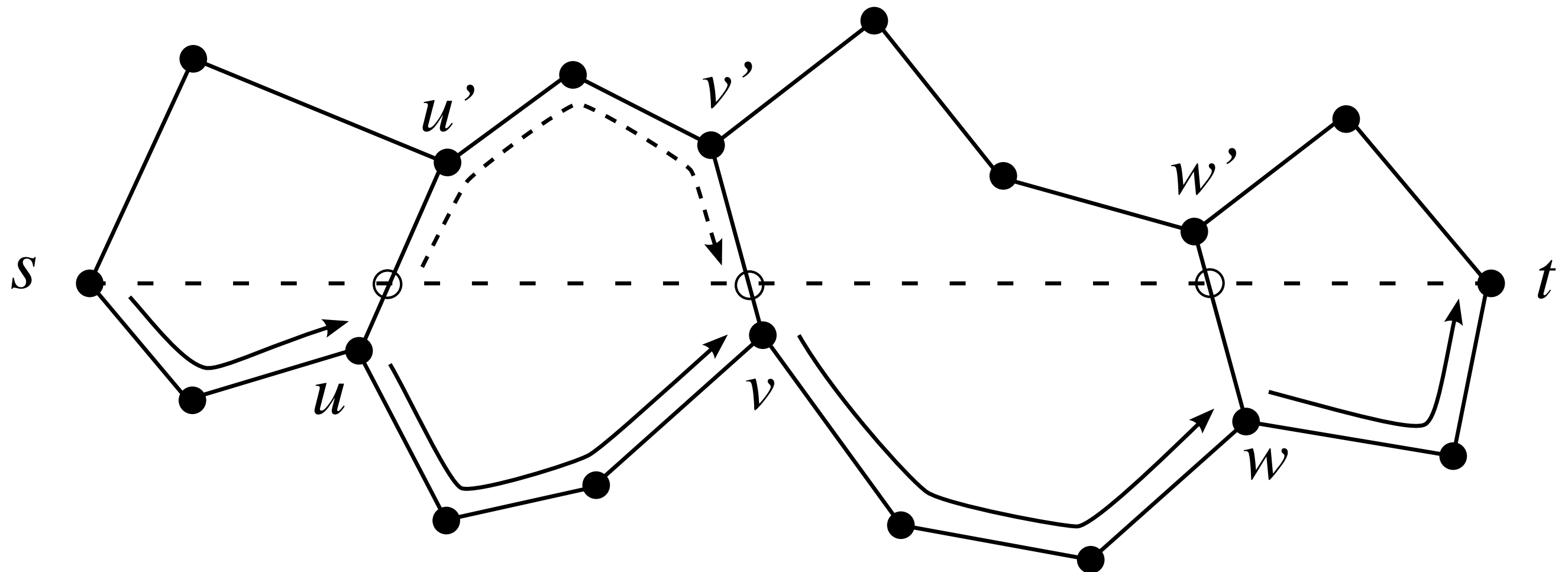


Planar Graph Routing

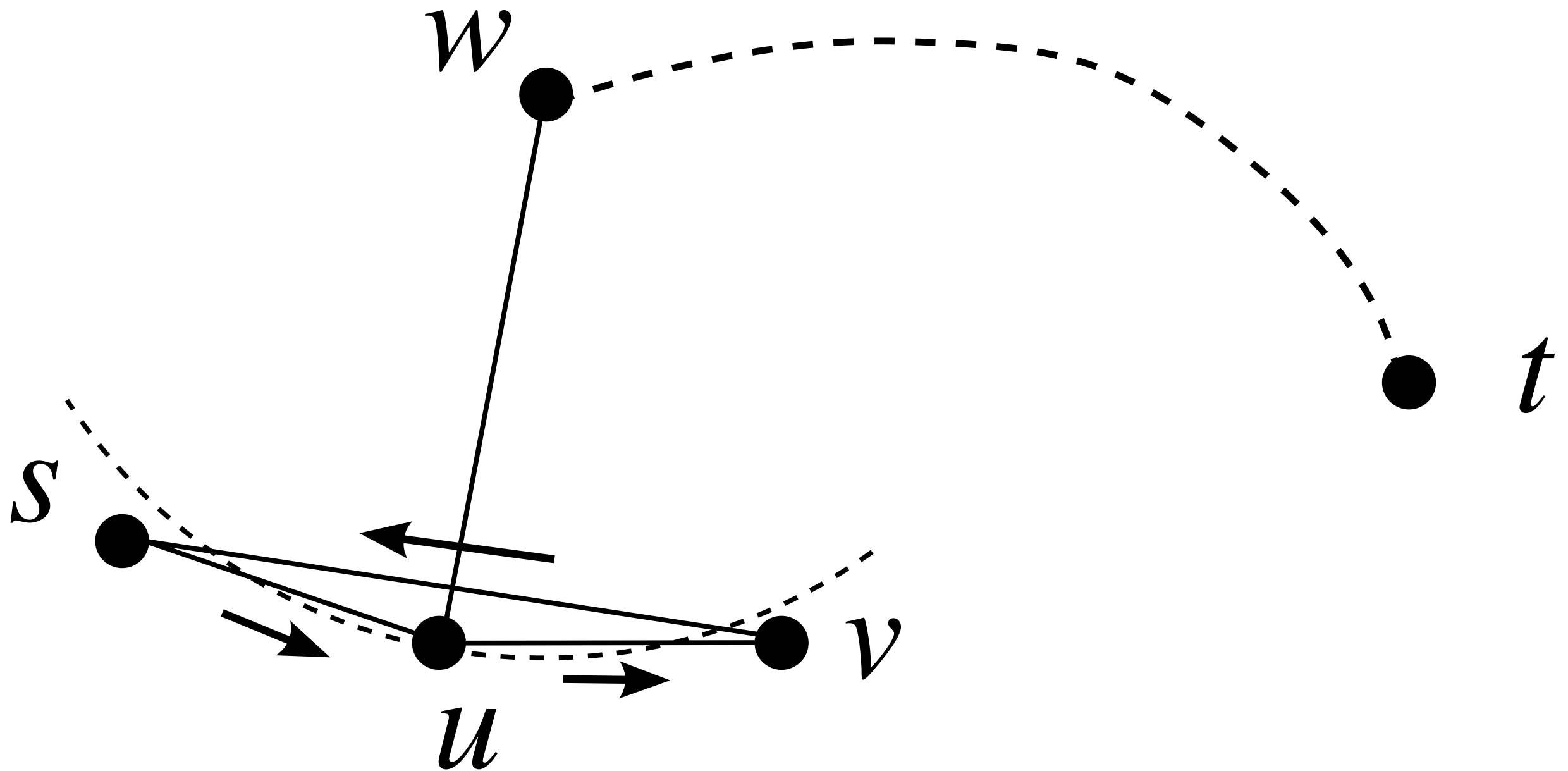


Bose, P.; Morin, P.; Stojmenovic, I.; Urrutia, J. (1999). "Routing with guaranteed delivery in ad hoc wireless networks". Proc. of the 3rd international workshop on discrete algorithms and methods for mobile computing and communications (DIALM '99). pp. 48–55.

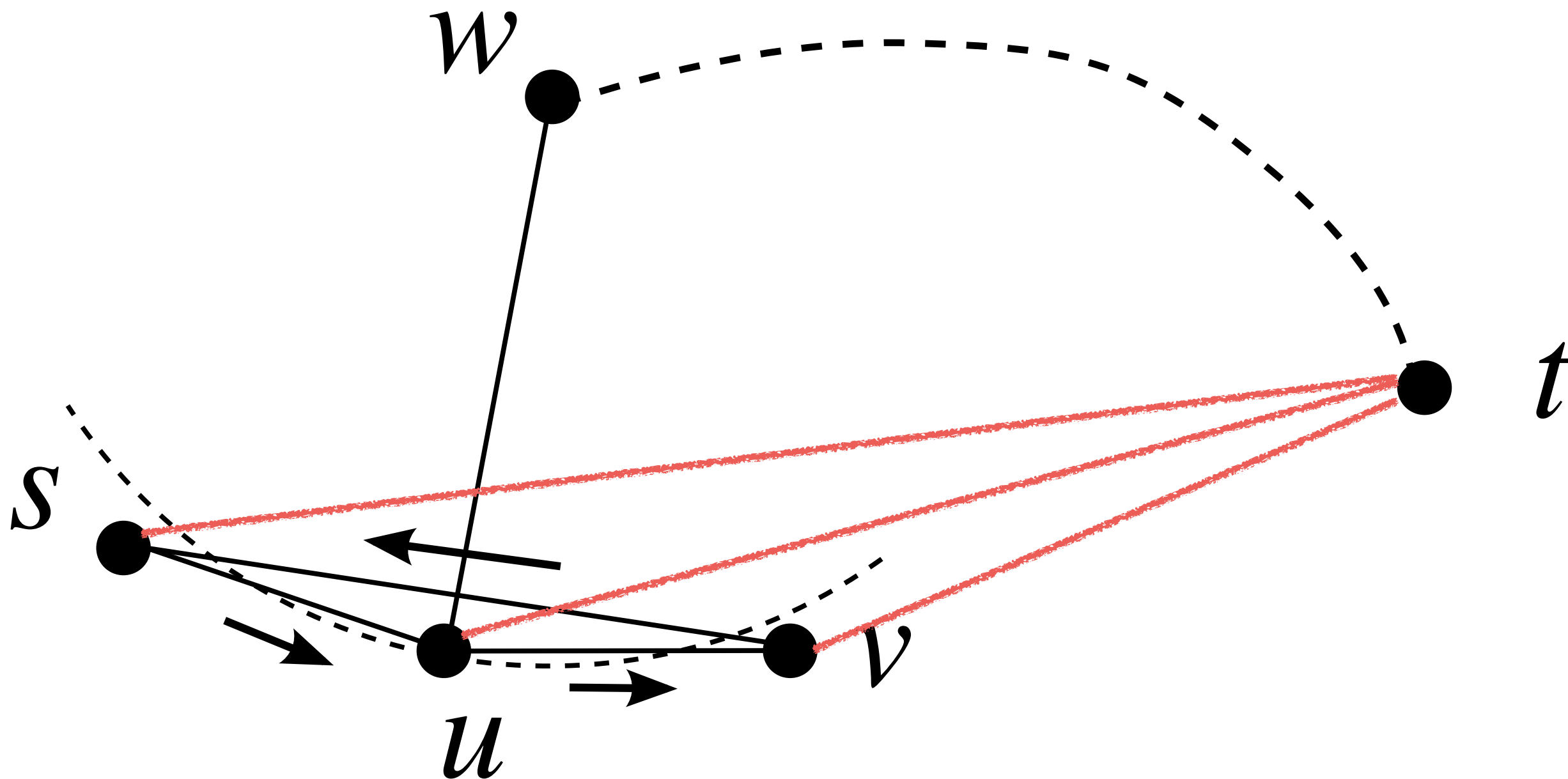
Face Routing



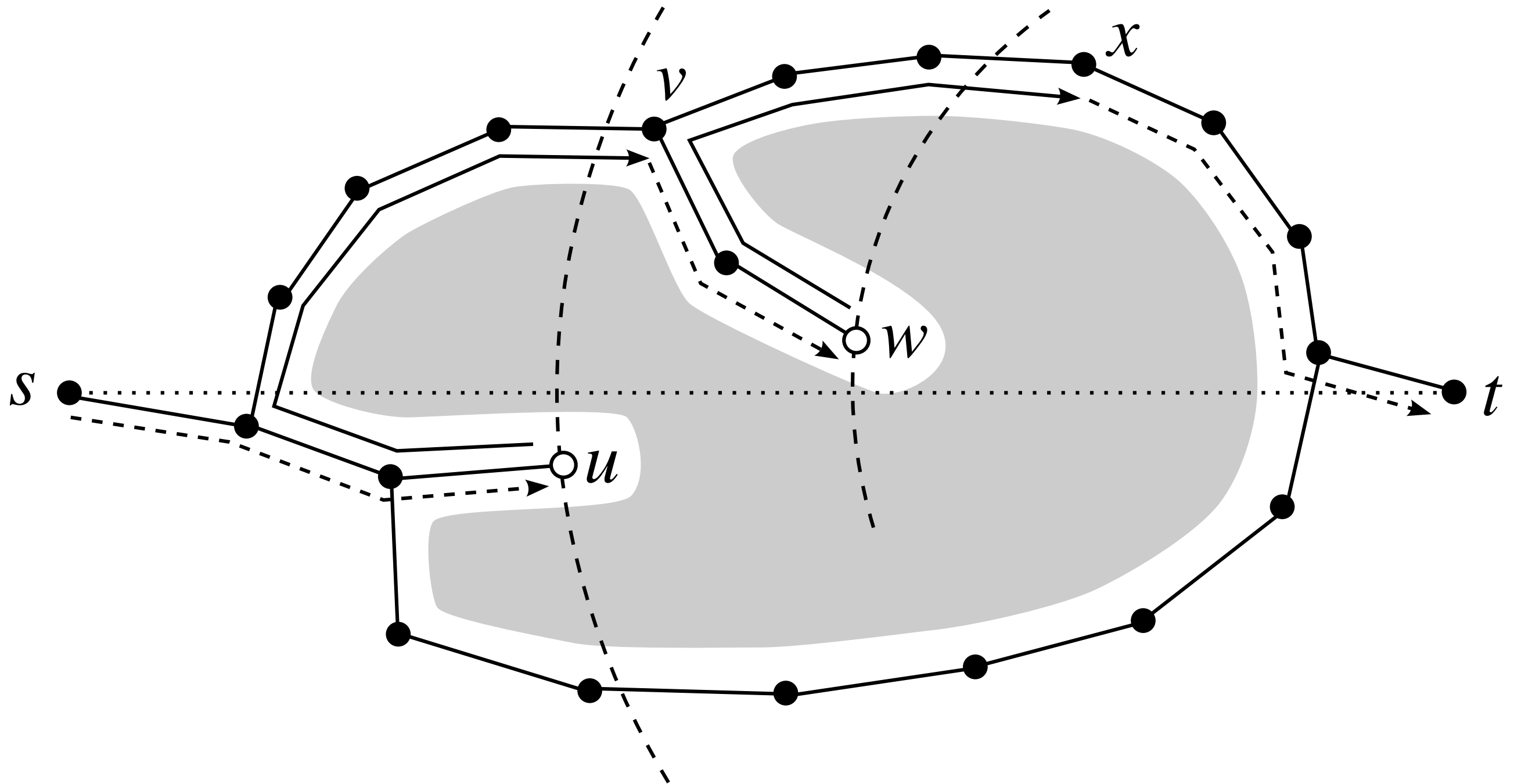
Planar Graphs!



Planar Graphs!



Greedy / Face / Greedy



Self-stabilization

Self-stabilization 101



Example

$$U_0 = a$$

$$U_{n+1} = \frac{U_n}{2} \text{ if } U_n \text{ is even}$$

$$U_{n+1} = \frac{3U_n+1}{2} \text{ if } U_n \text{ is odd}$$

Example

$$U_0 = a$$

$$U_{n+1} = \frac{U_n}{2} \text{ if } U_n \text{ is even}$$

$$U_{n+1} = \frac{3U_n+1}{2} \text{ if } U_n \text{ is odd}$$

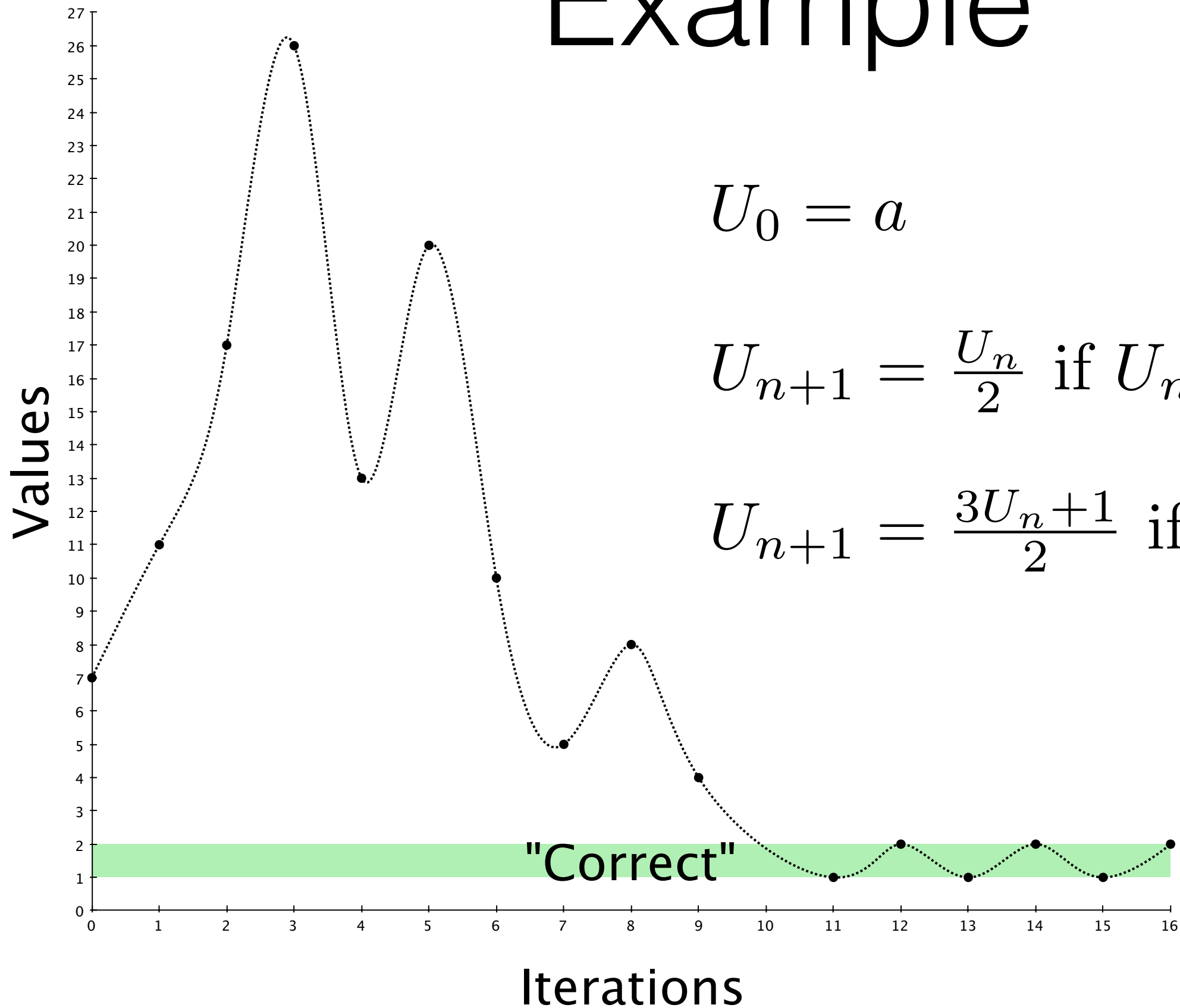
| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|-------|---|----|----|----|----|----|----|---|---|---|----|----|----|
| U_n | 7 | 11 | 17 | 26 | 13 | 20 | 10 | 5 | 8 | 4 | 2 | 1 | 2 |

Example

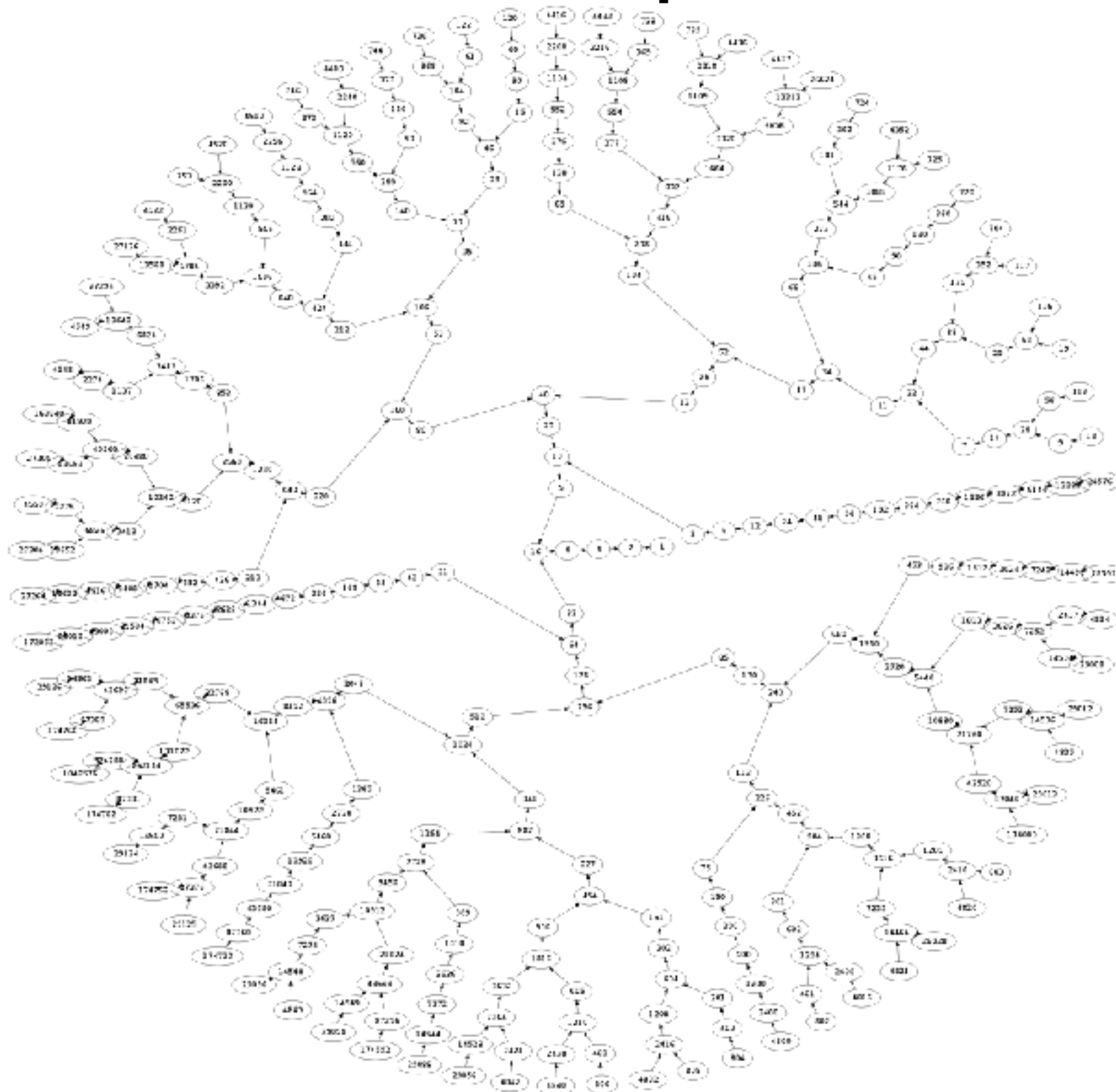
$$U_0 = a$$

$$U_{n+1} = \frac{U_n}{2} \text{ if } U_n \text{ is even}$$

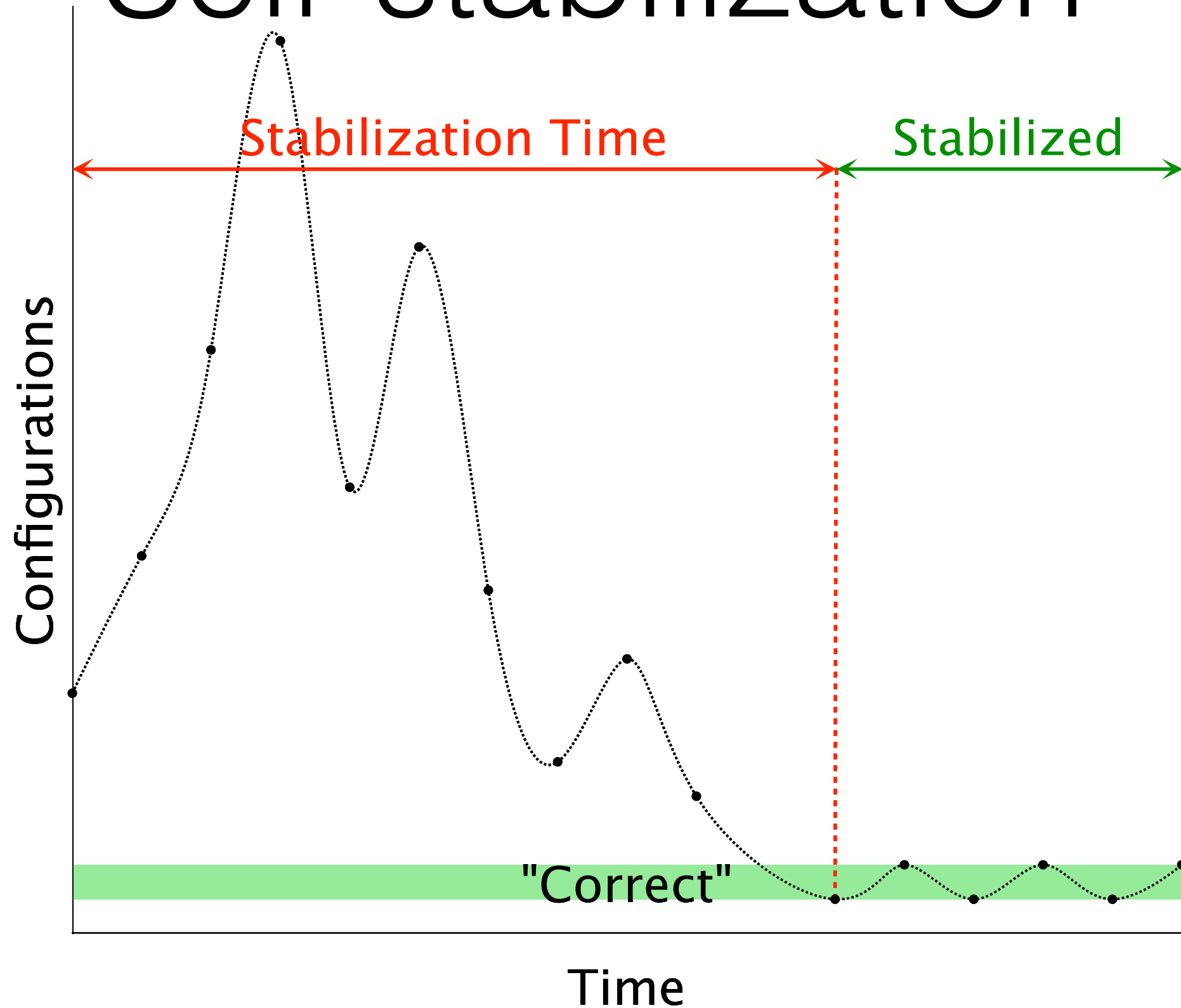
$$U_{n+1} = \frac{3U_n+1}{2} \text{ if } U_n \text{ is odd}$$



Example

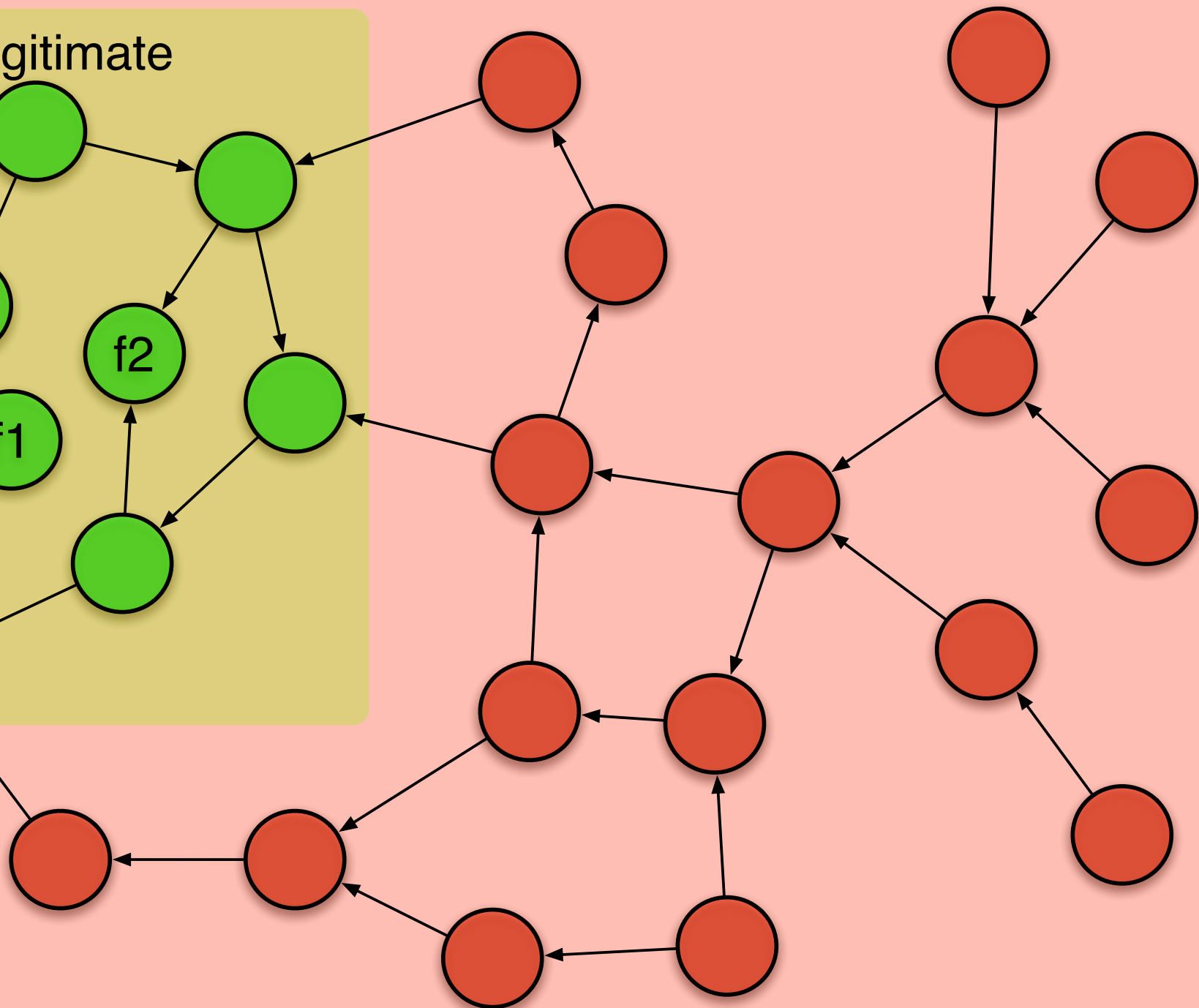
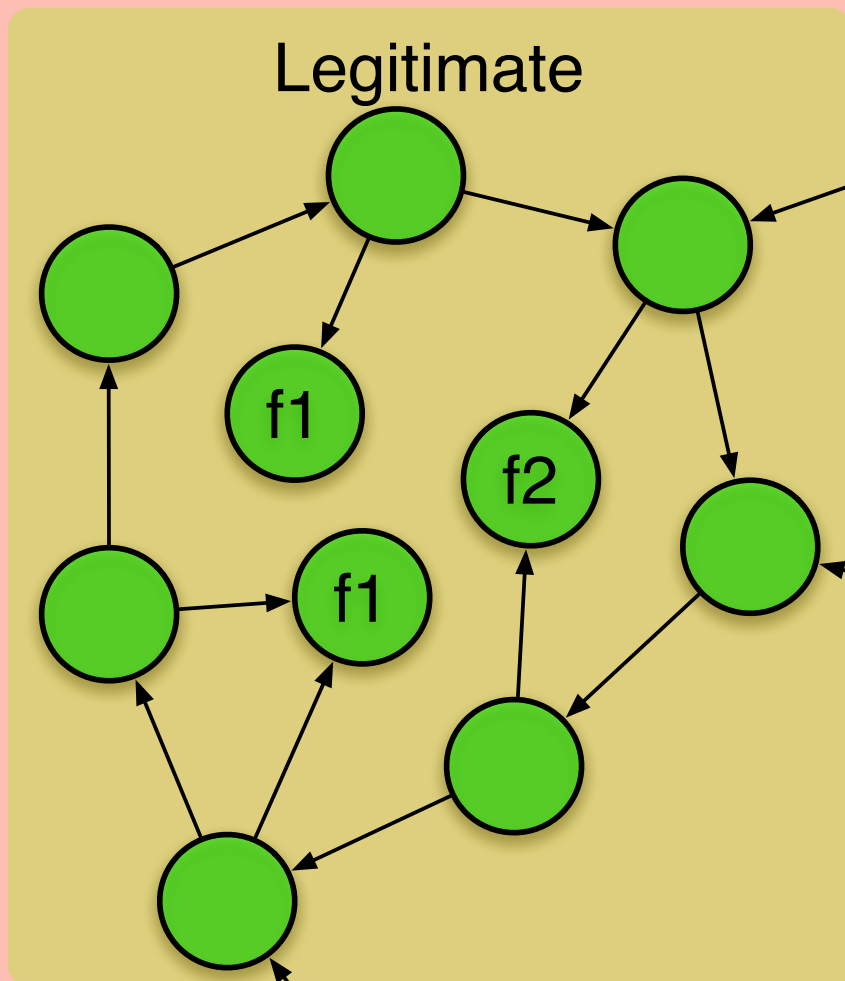


Self-stabilization

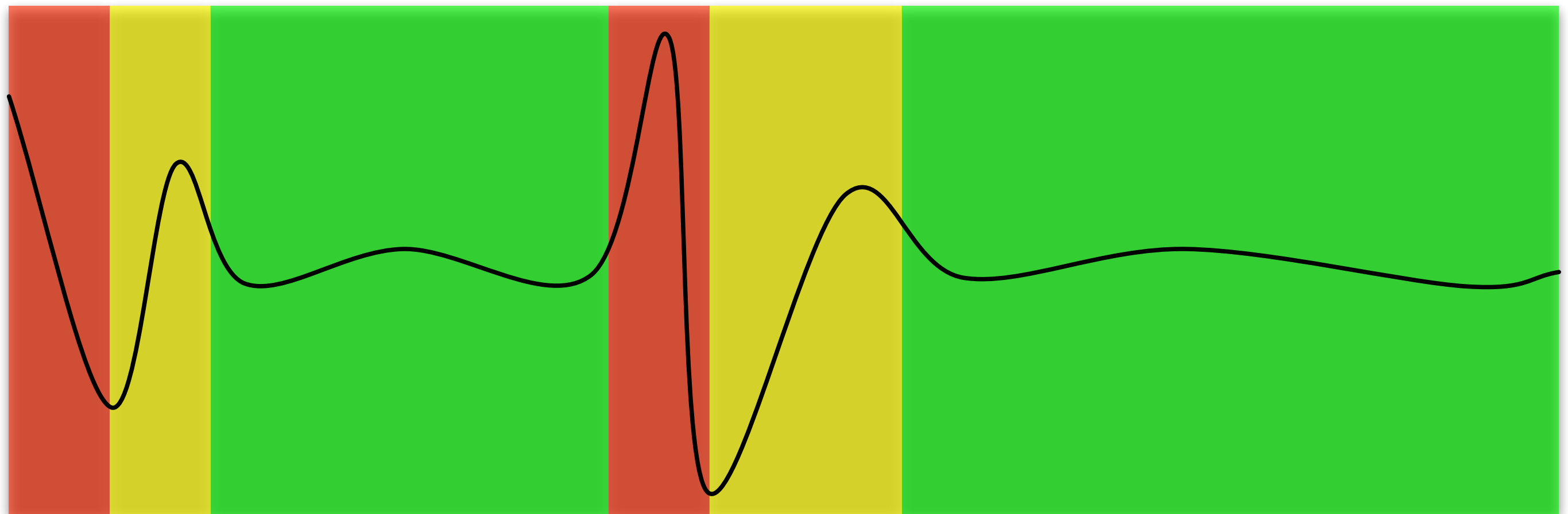


Self-stabilization

Arbitrary



Self-stabilization



Distributed Systems

- **Configuration:** product of the local states of system components
- **Execution:** interleaving of the local executions of the system components

Distributed Systems

- **Classical:** Starting from a particular initial configuration, the system immediately exhibits correct behavior
- **Self-stabilizing:** Starting from any initial configuration, the system eventually reaches a configuration from which its behavior is correct

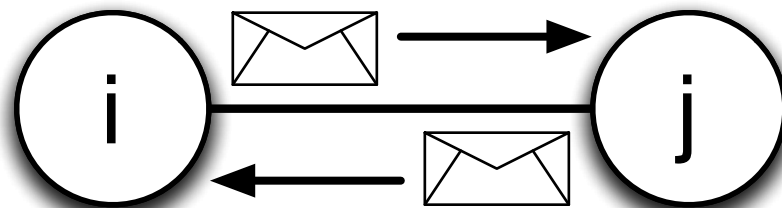
Distributed Systems

- **Self-stabilizing**: Starting from **any** initial configuration, the system **eventually** reaches a configuration from which its behavior is correct
 - Defined by *Dijkstra* in 1974
 - Advocated by *Lamport* in 1984 to address fault-tolerant issues
 - Stale states due to **mobility** can be recovered!

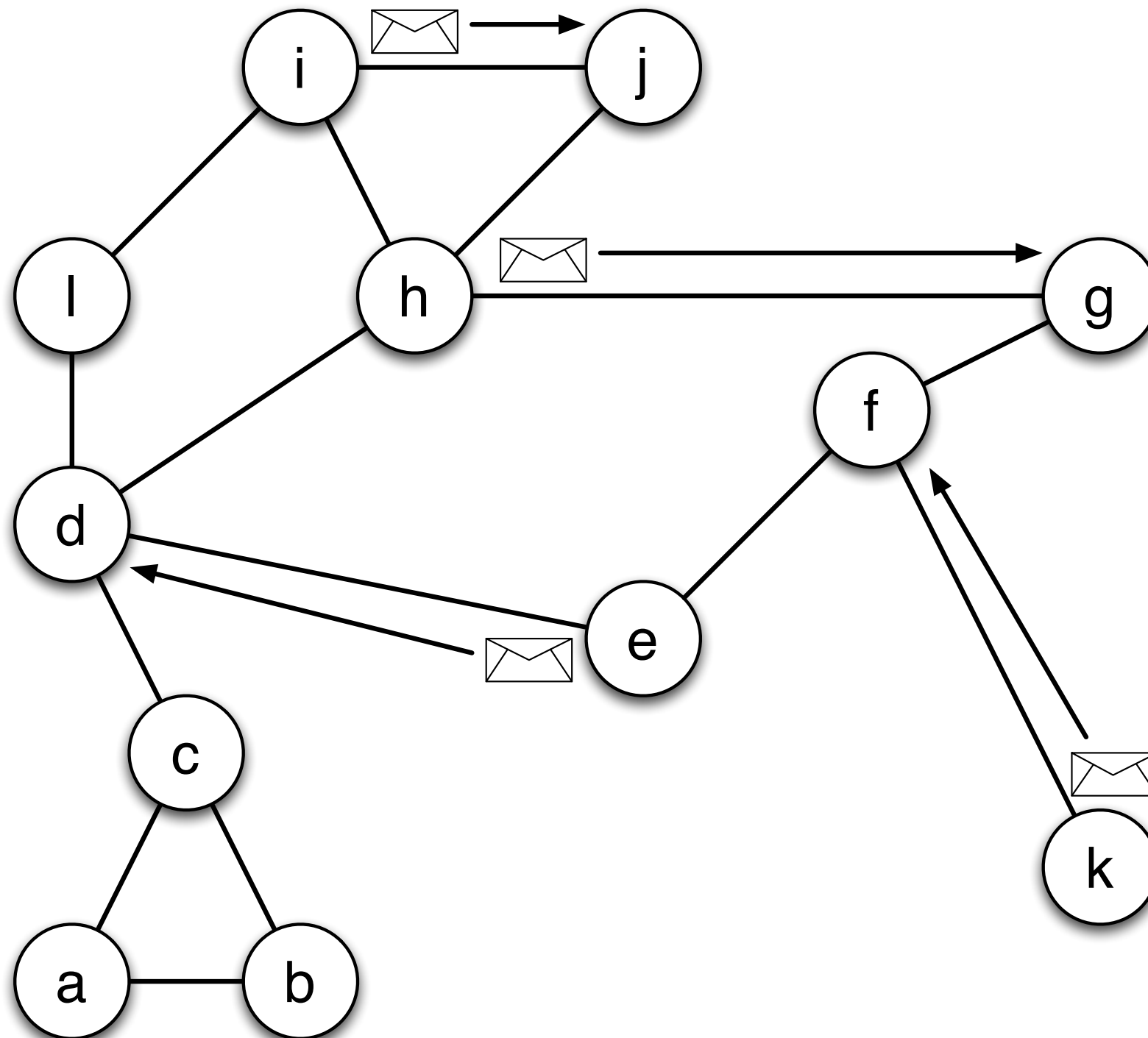
Configurations

```
int x = 0;  
...  
if( x == 0 ) {  
    // code assuming x equals 0  
}  
else {  
    // code assuming x does not equal 0  
}
```

Configurations



Configurations



Hypotheses

Atomicity

- A «*stabilizing*» sequential program

```
int x = 0;
```

```
...
```

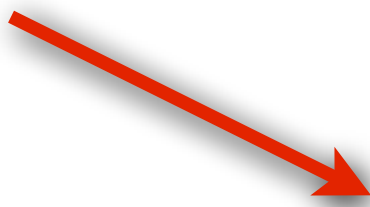
```
while( x == x ) {  
    x = 0;  
    // code assuming x equals 0  
}
```

Atomicity

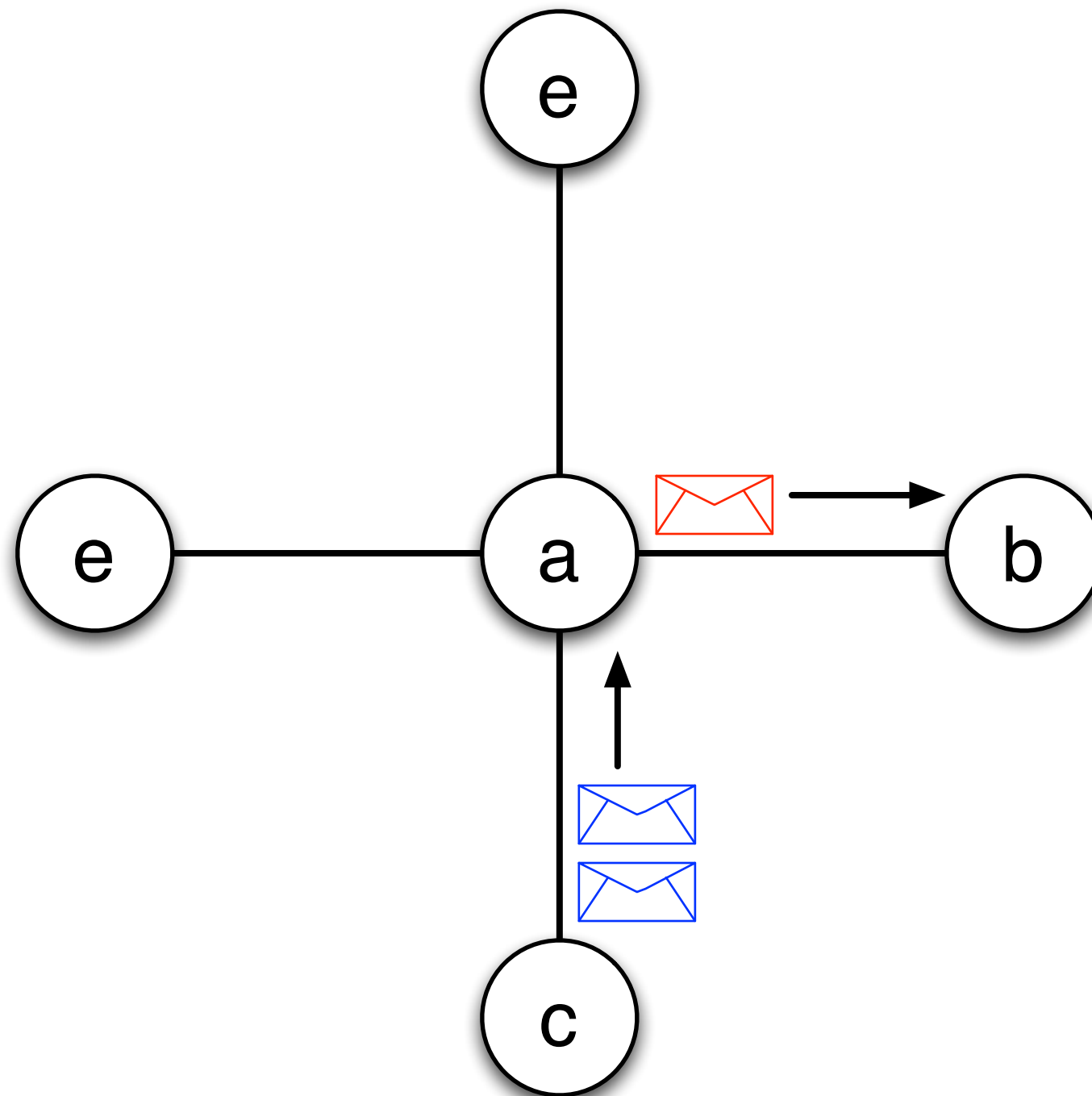
- A «*stabilizing*» sequential program

```
0  iconst_0
1  istore_1
2  goto 7
5  iconst_0
6  istore_1
7  iload_1
8  iload_1
9  if_icmpeq 5
```

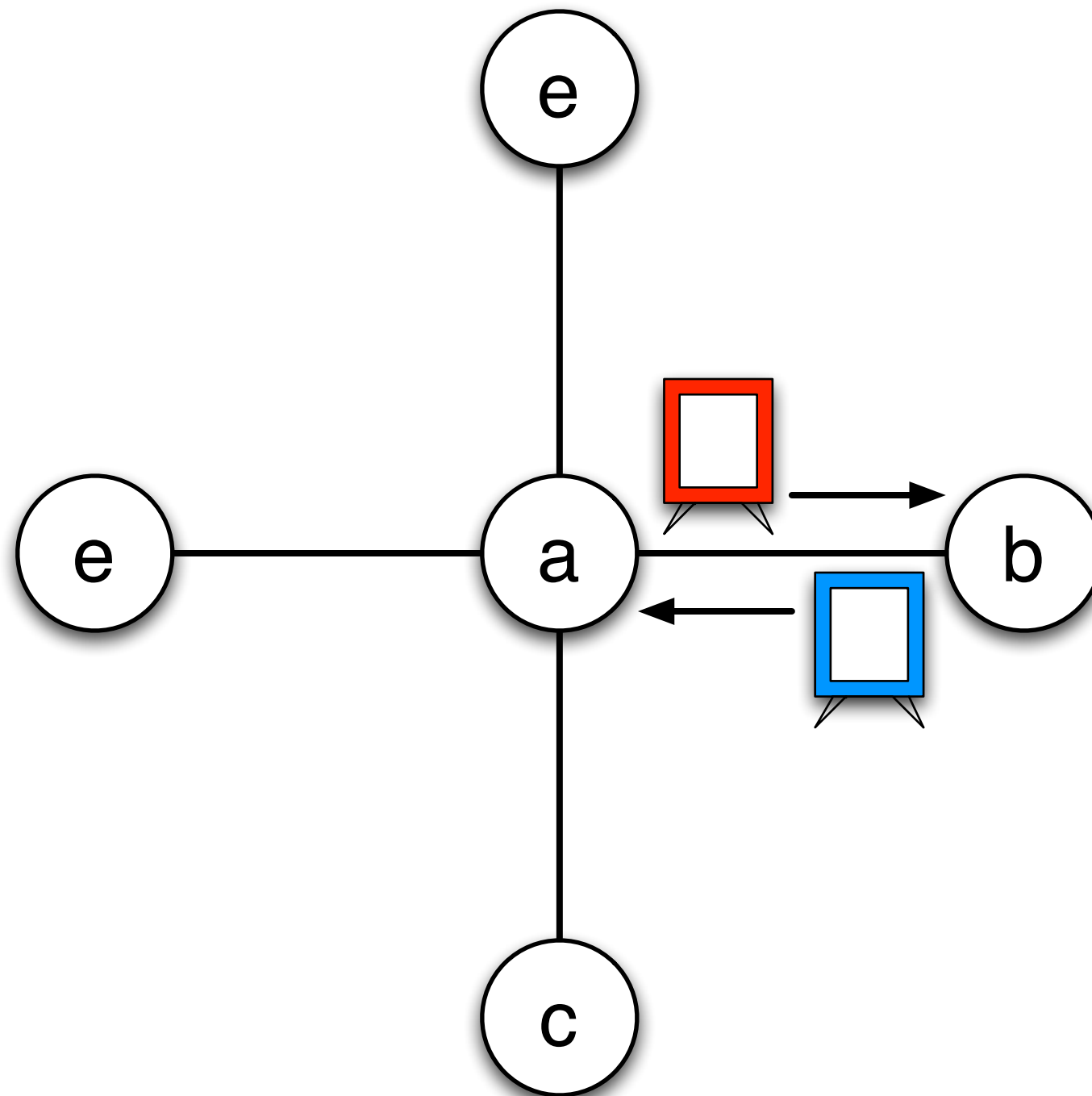
Problem



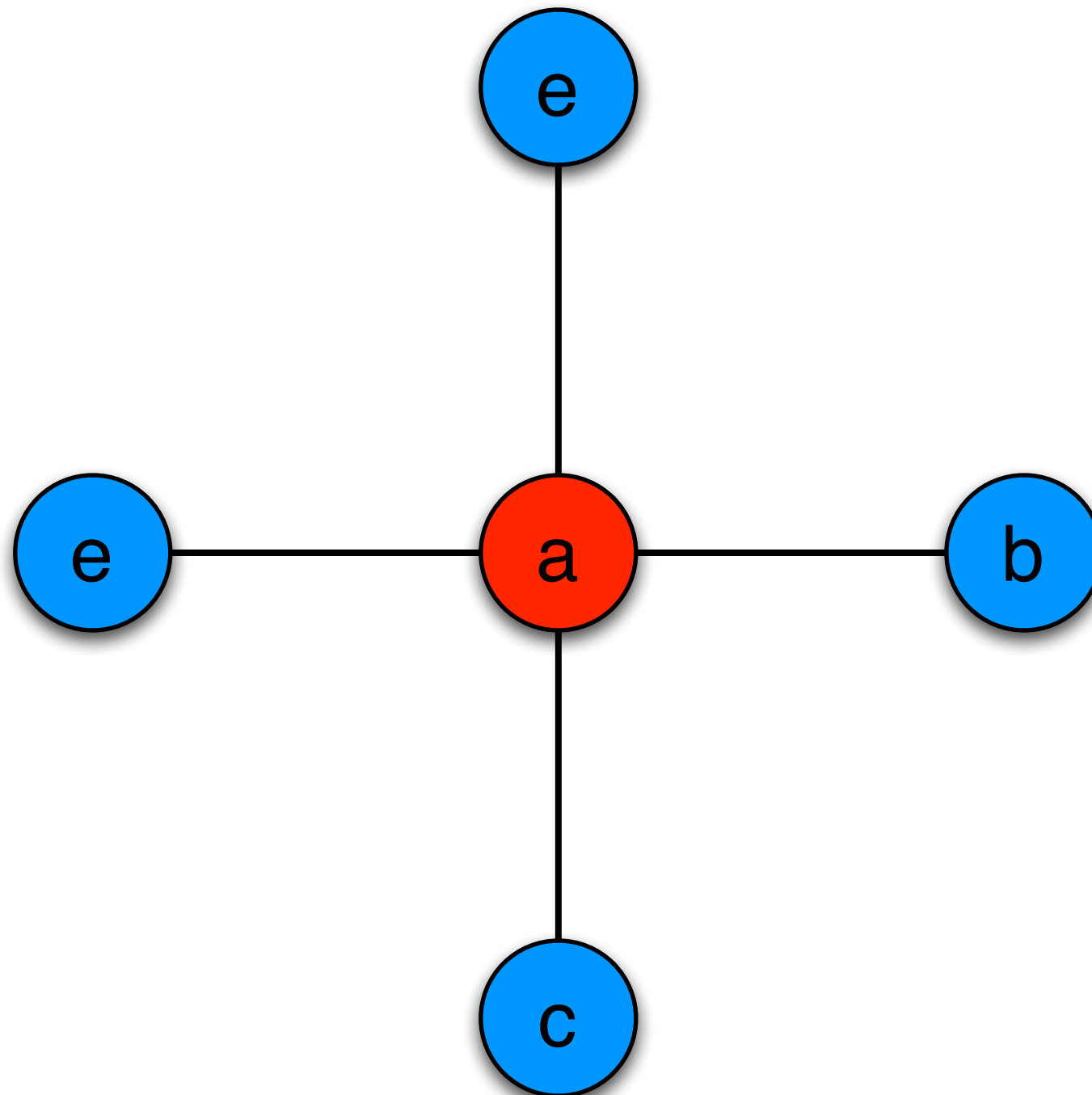
Communications



Communications



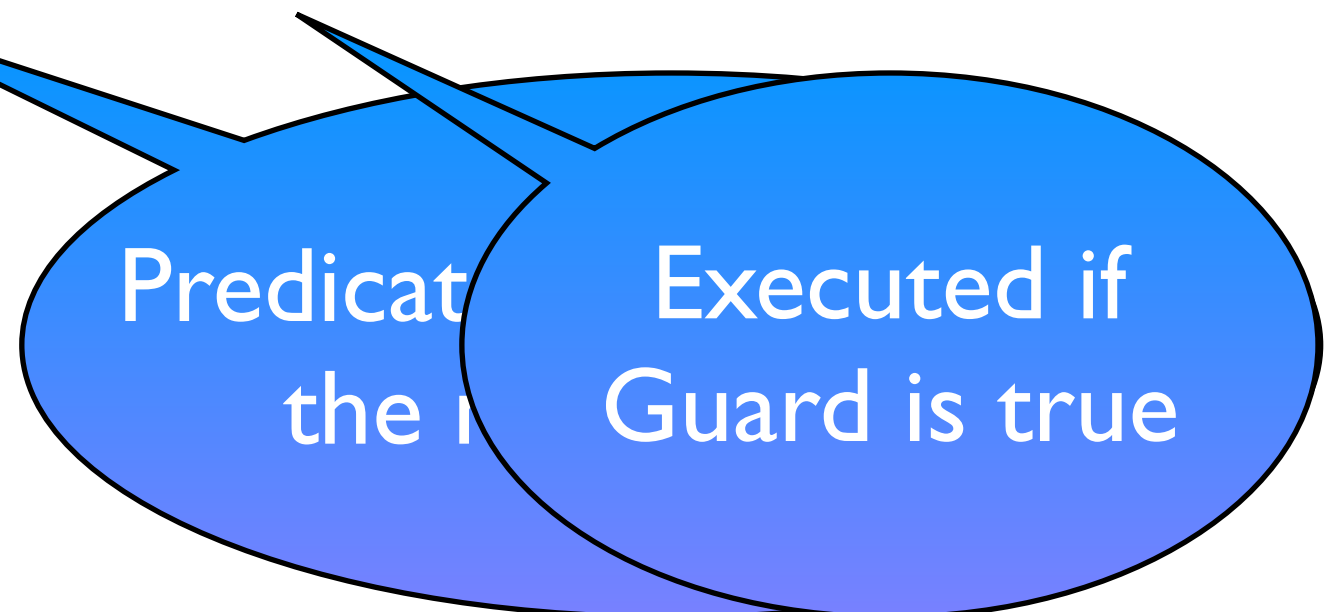
Communications



Example

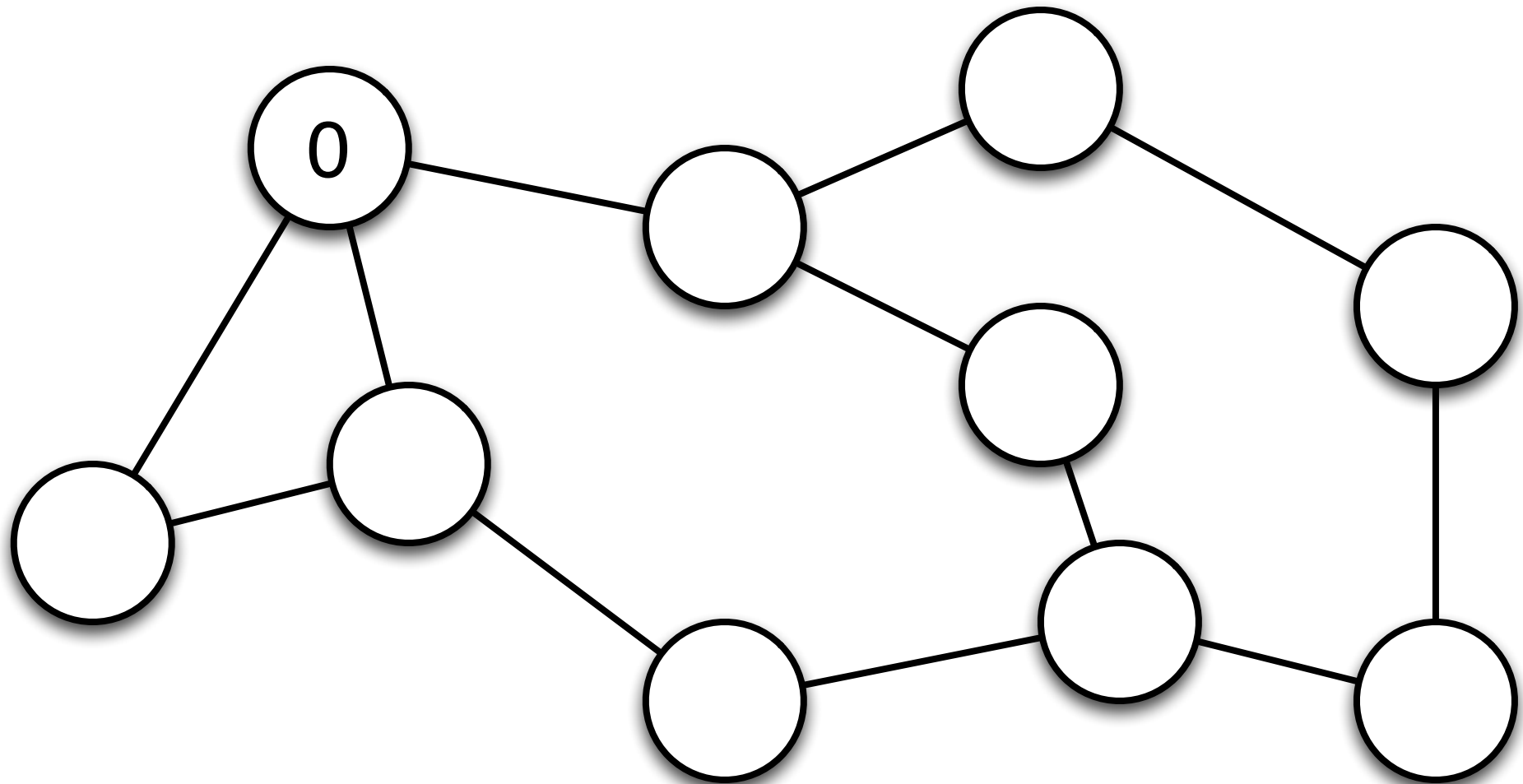
- **Shared memory:** in one atomic step, read the state of all neighbors and write own state
- **Guarded command**

Guard \rightarrow *Action*

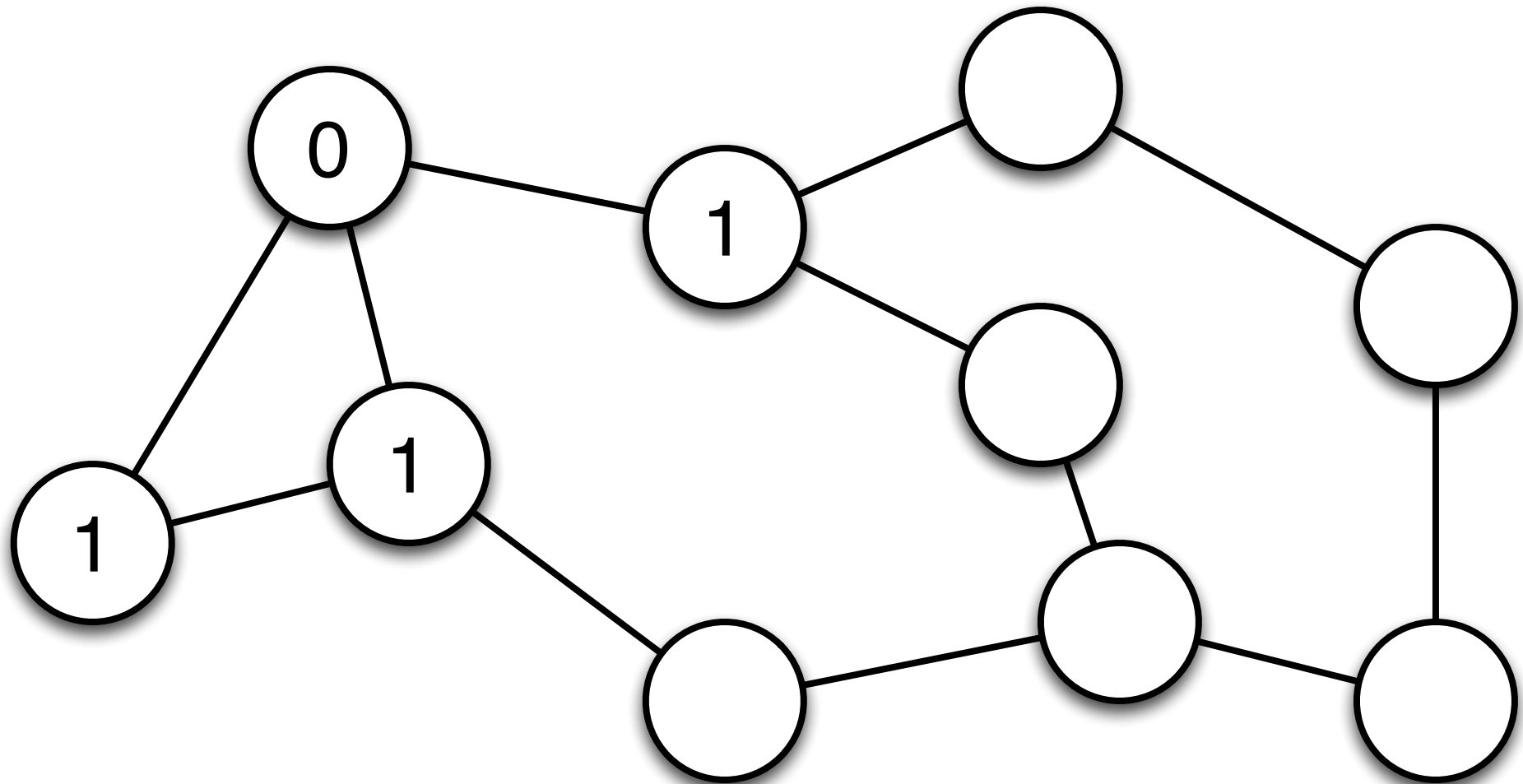


Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$

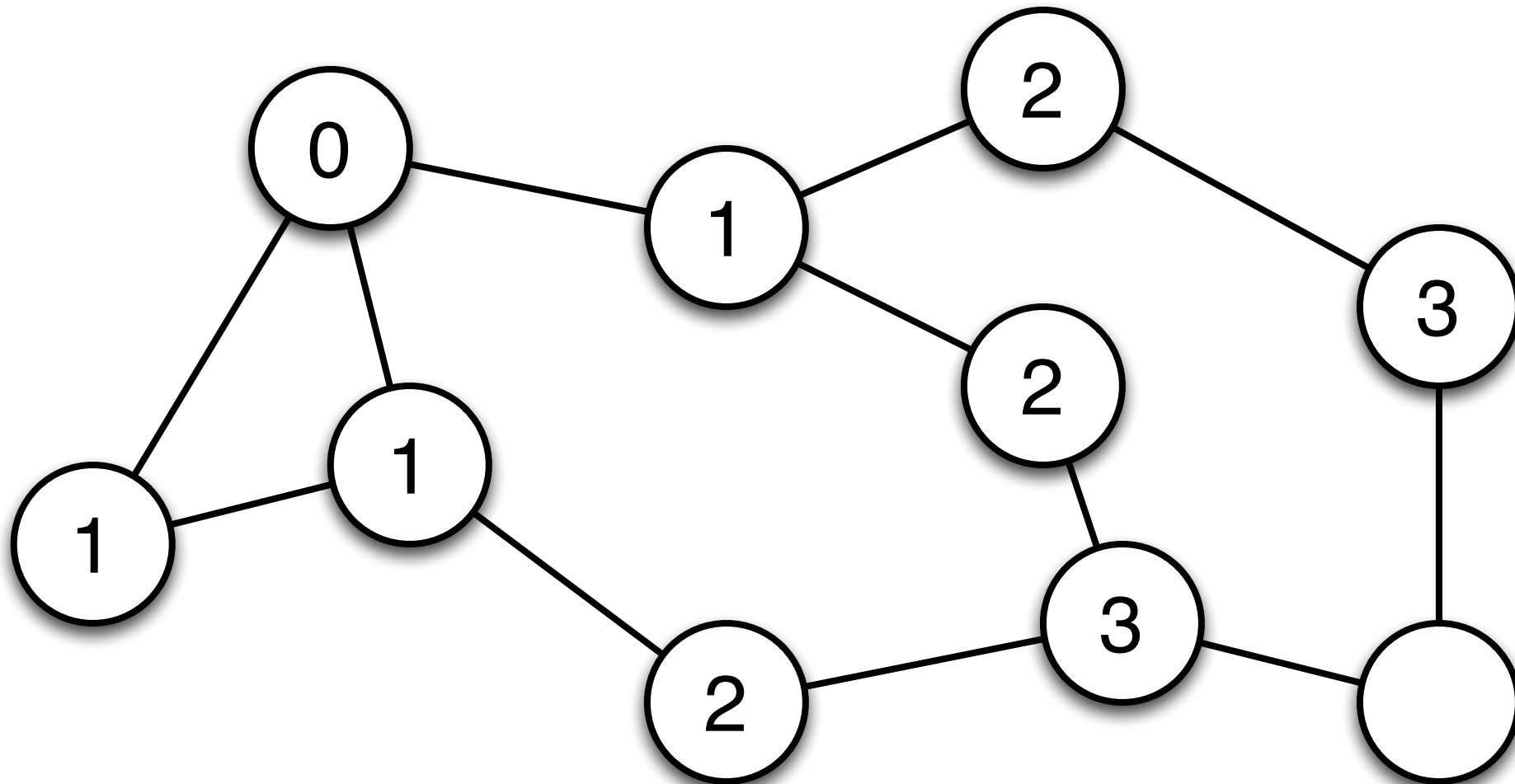


Example

$$true \rightarrow Distance_i := Min_{j \in Neighbors_i} \{Distance_j + 1\}$$


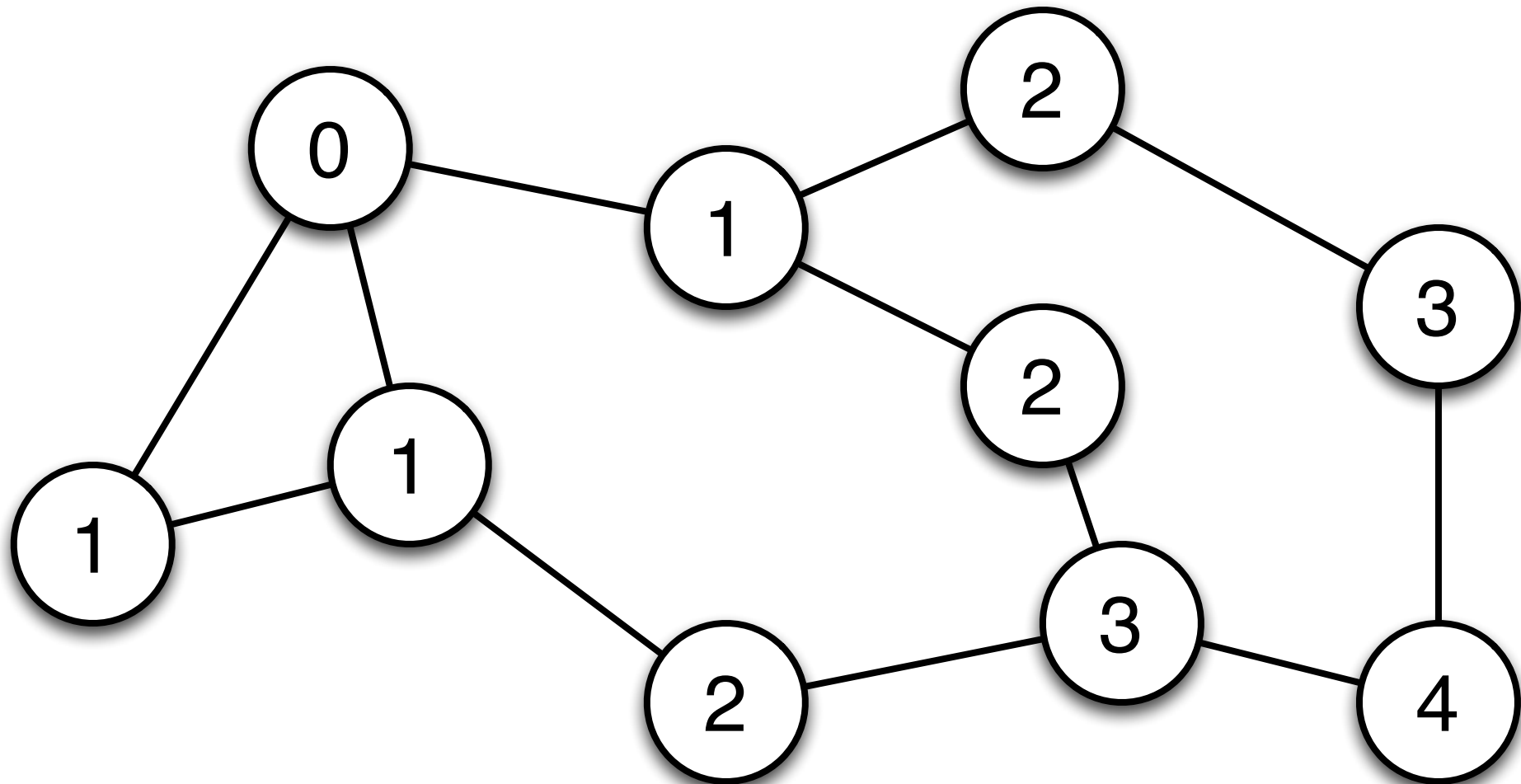
Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$



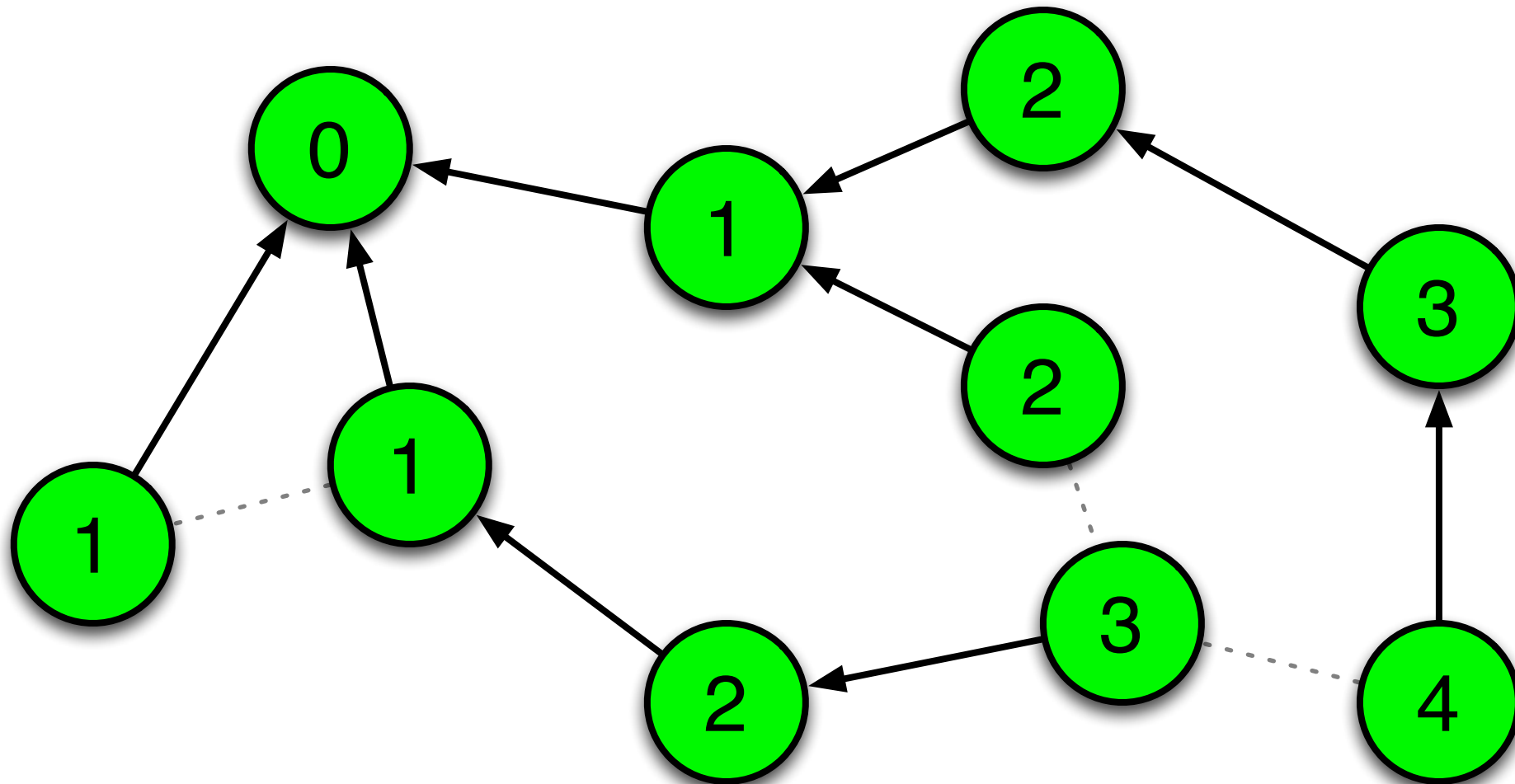
Example

$true \rightarrow Distance_i := Min_{j \in Neighbors_i} \{Distance_j + 1\}$



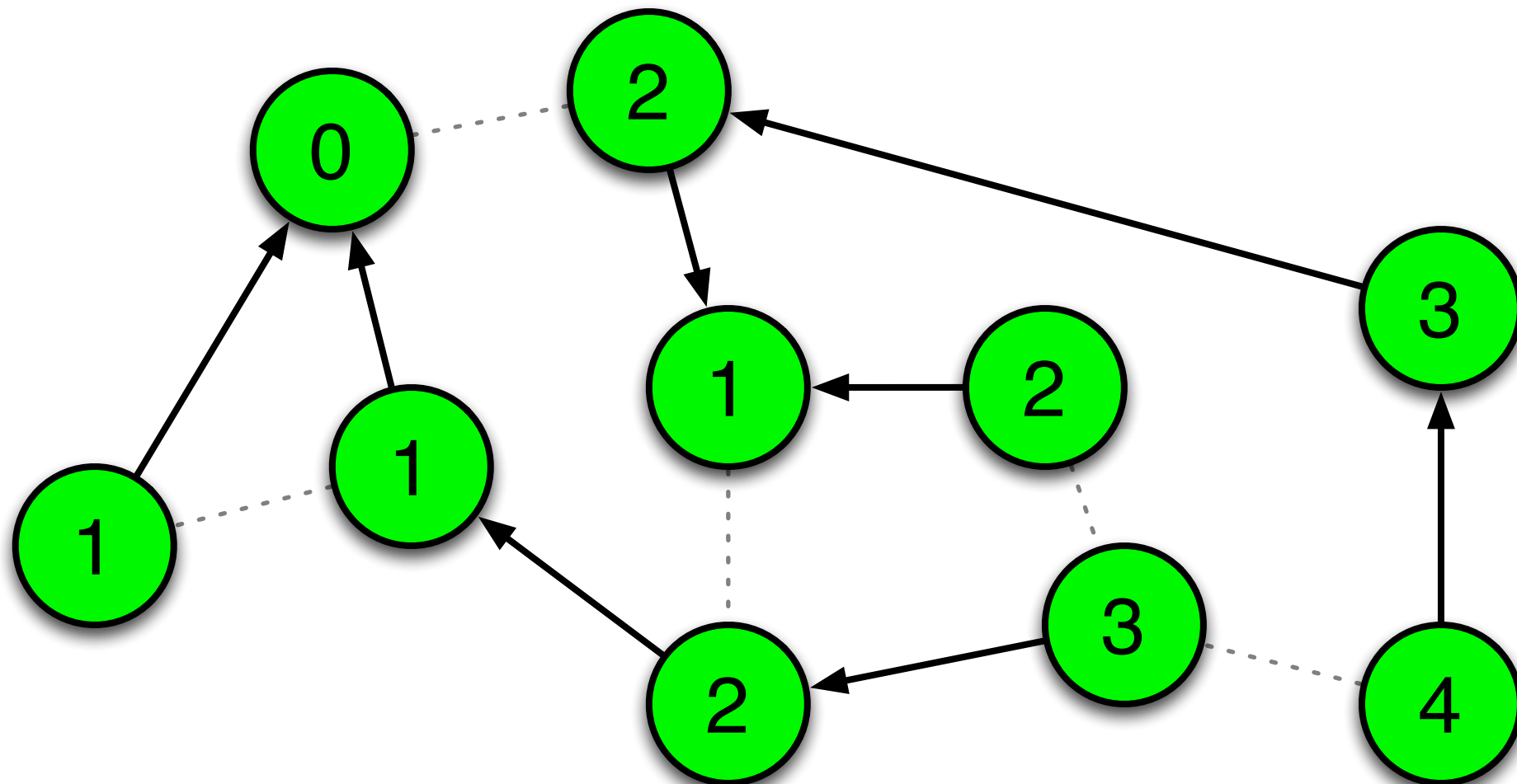
Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$



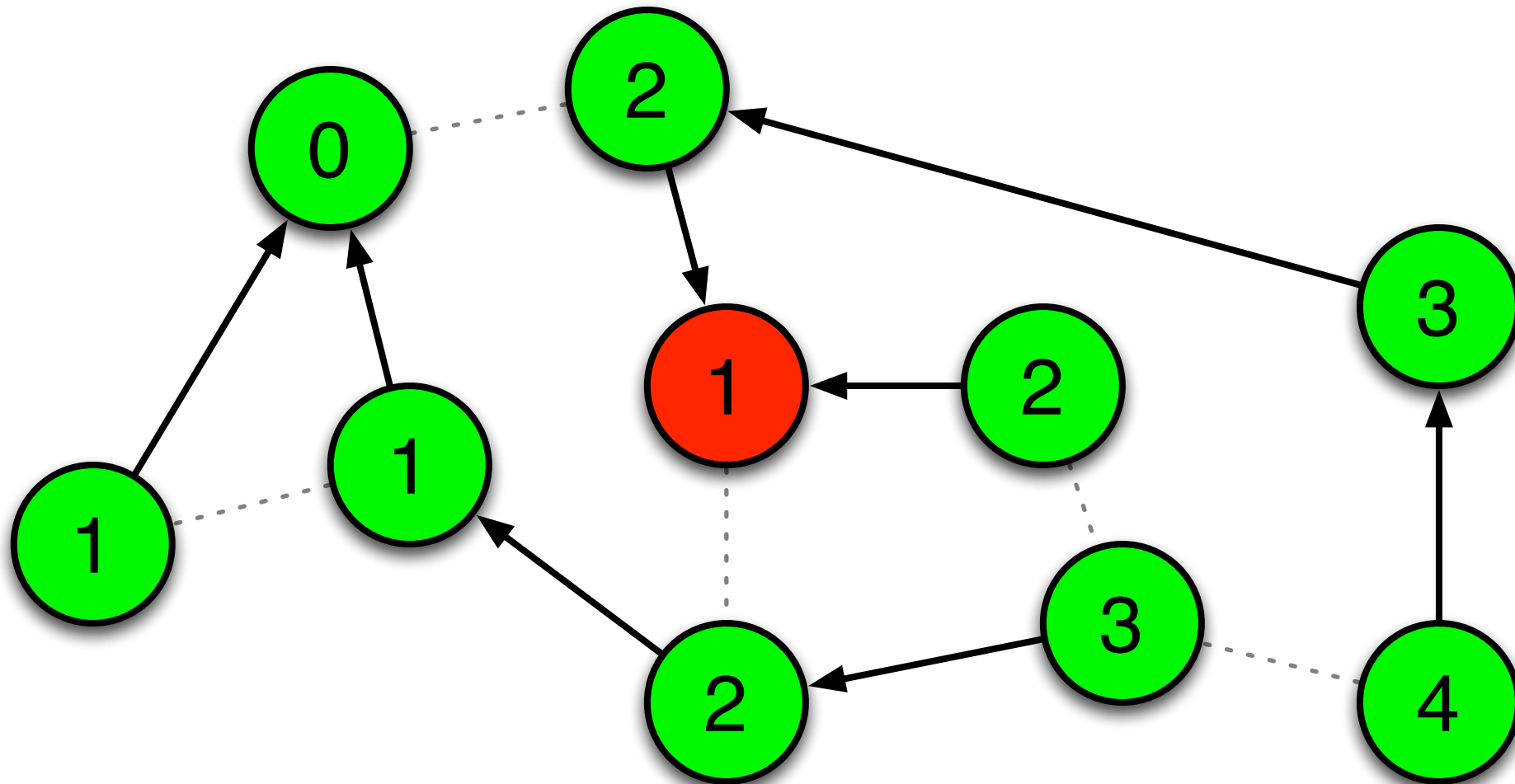
Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$



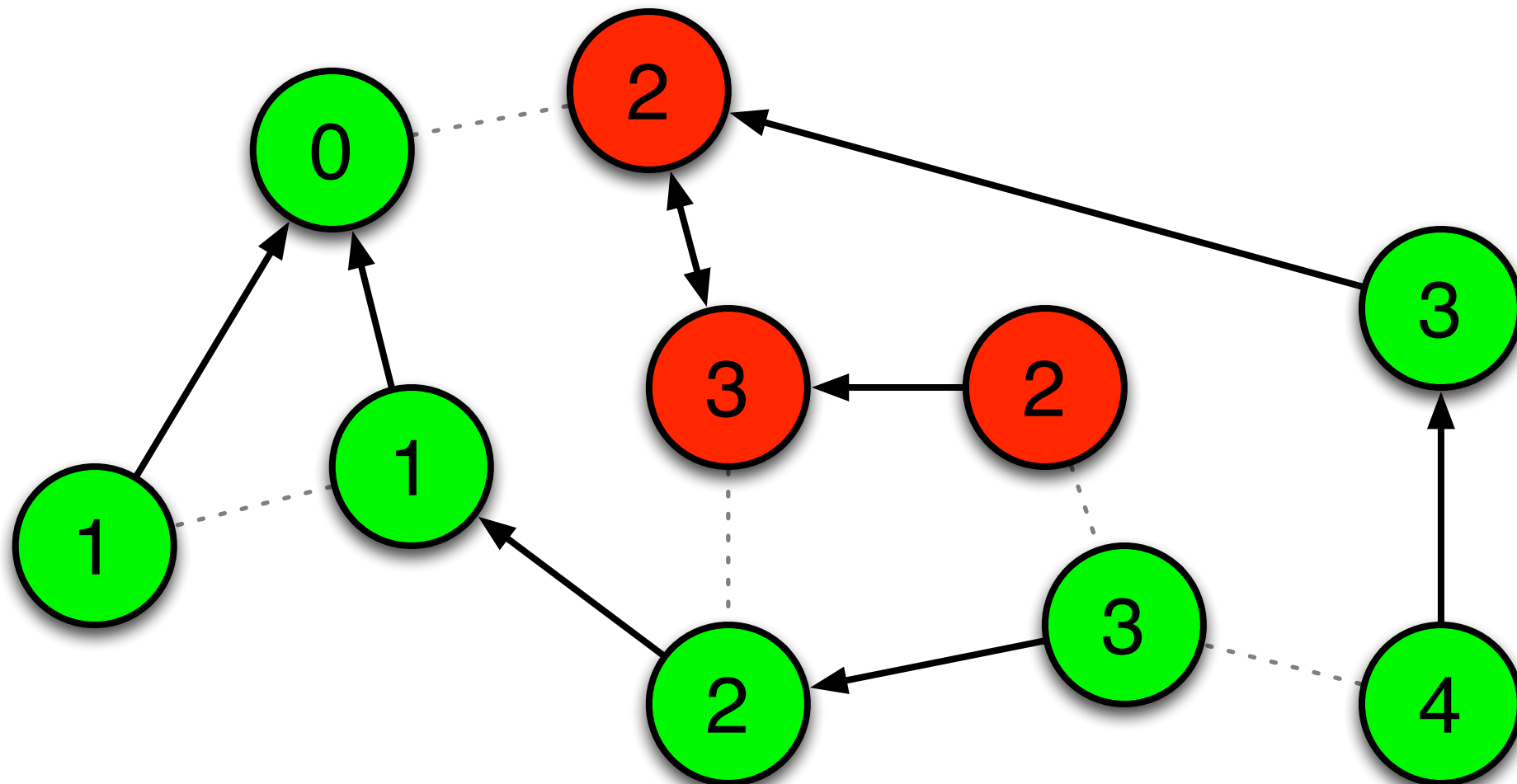
Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$



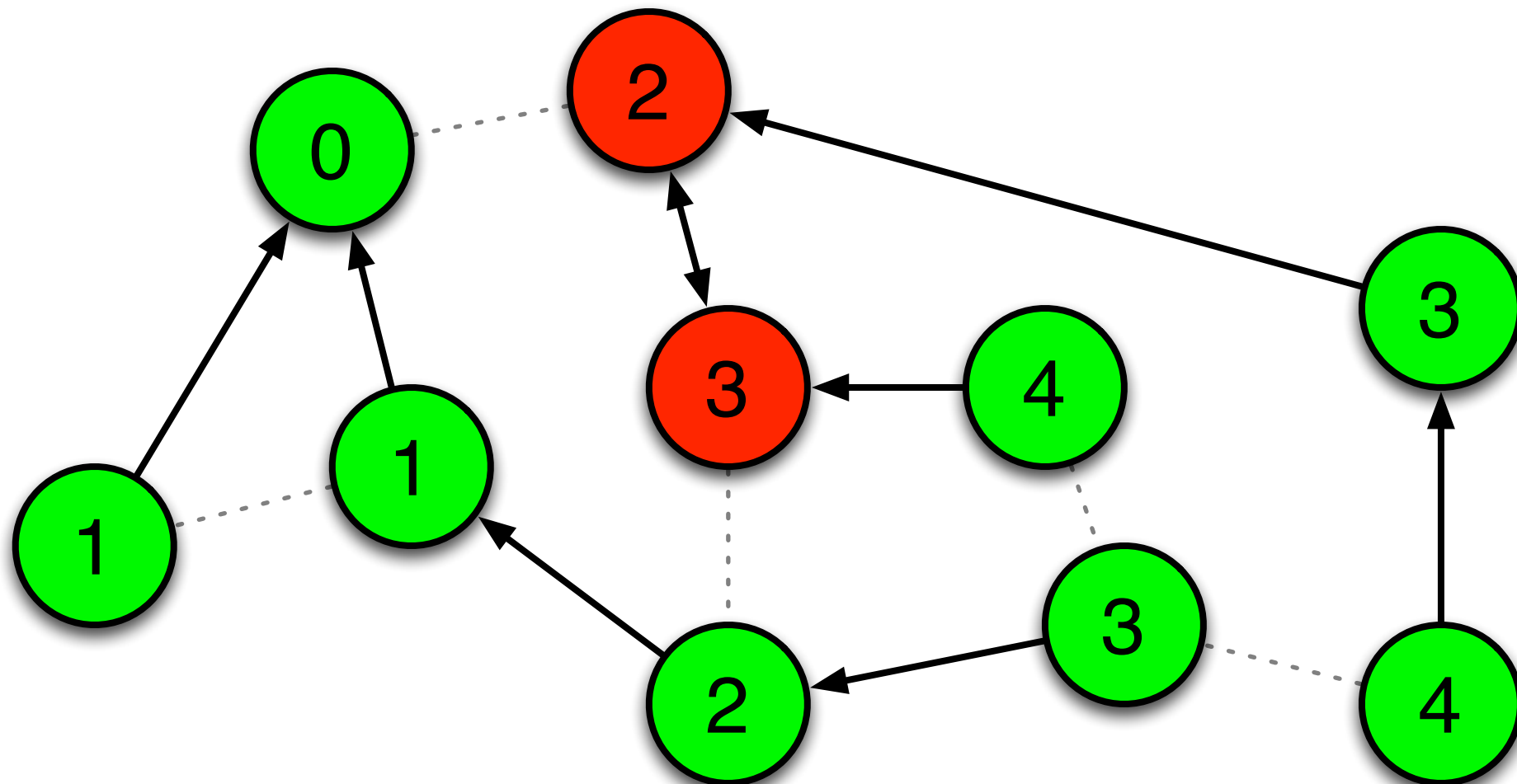
Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$



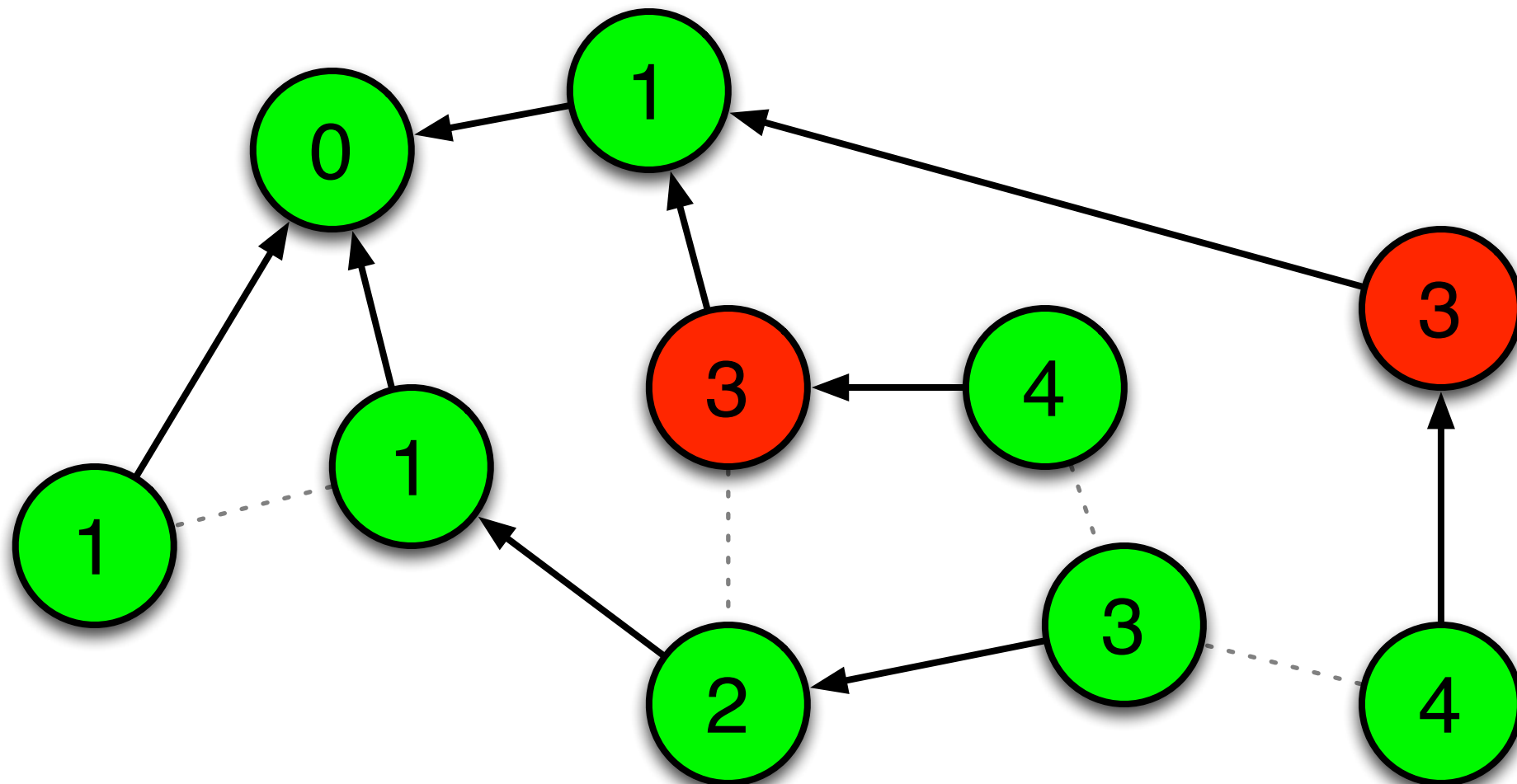
Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$



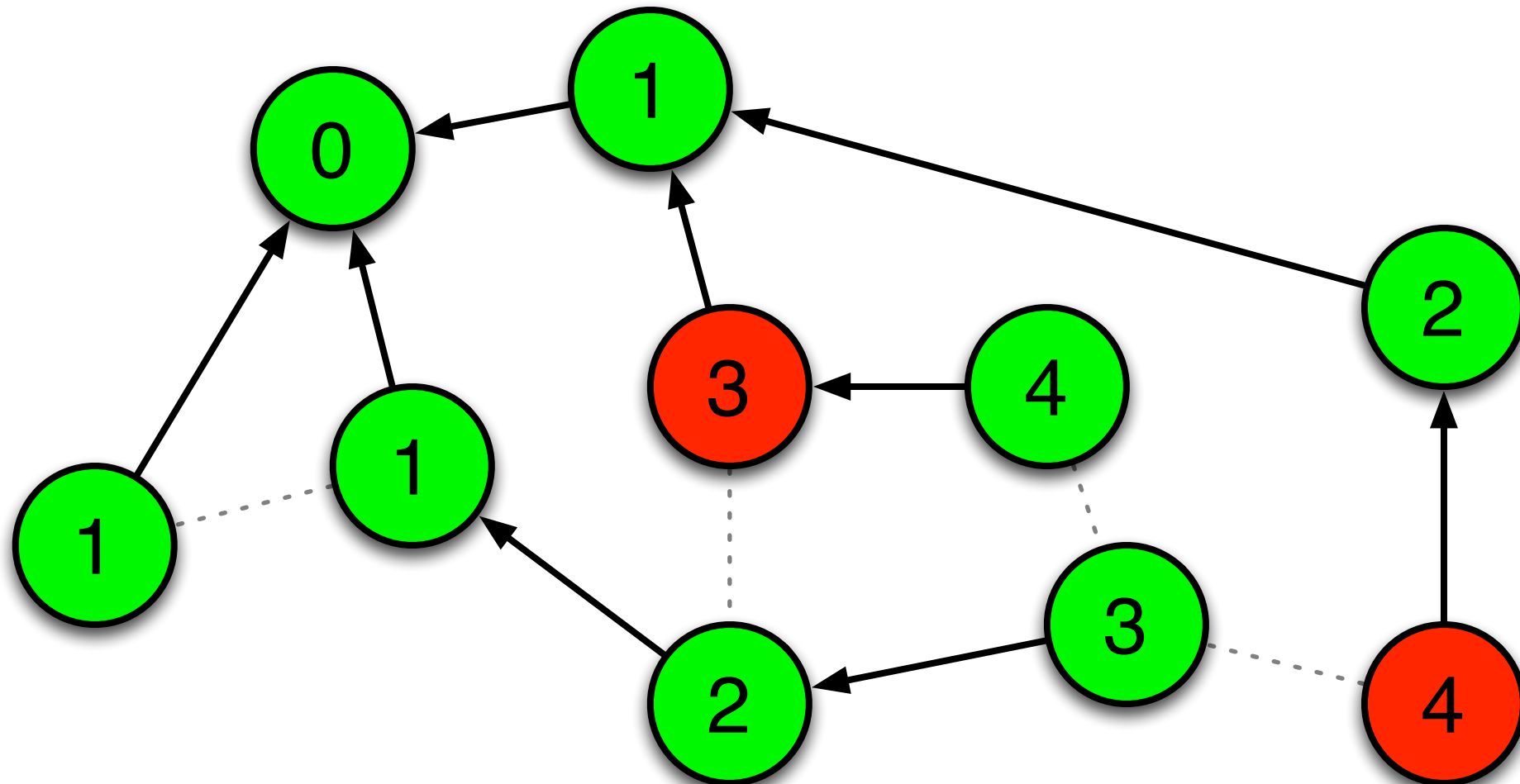
Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$



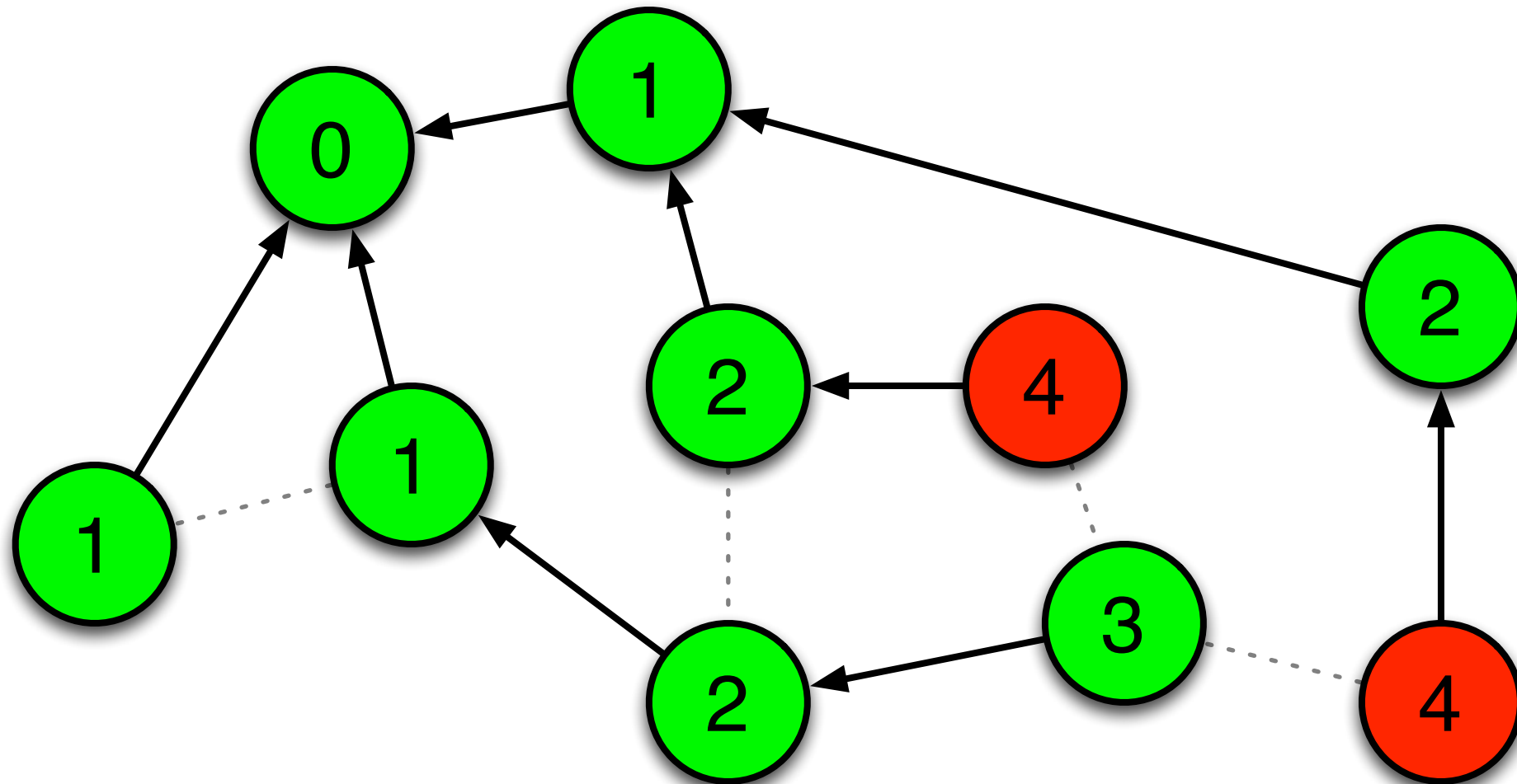
Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$



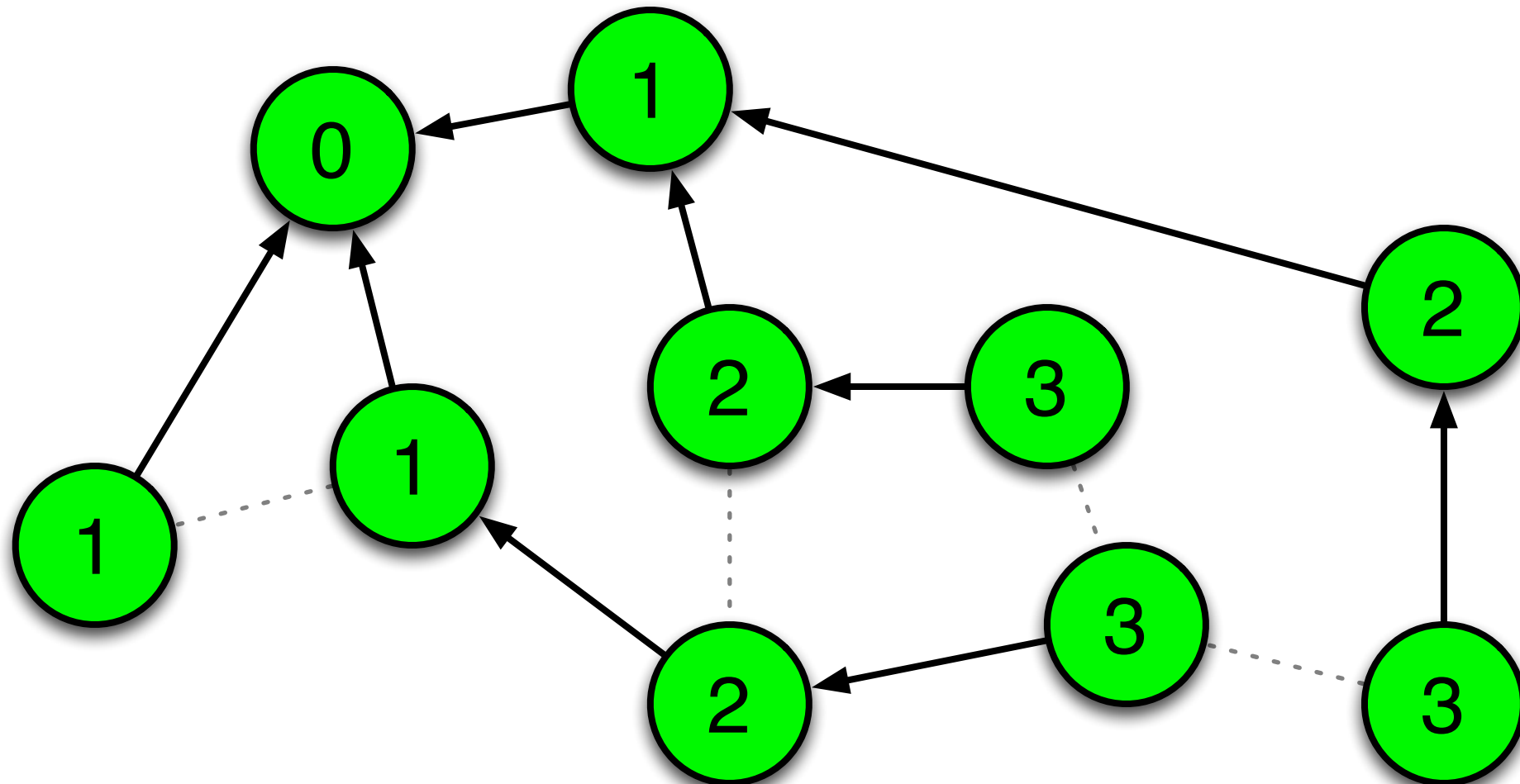
Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$



Example

$true \rightarrow Distance_i := \text{Min}_{j \in \text{Neighbors}_i} \{Distance_j + 1\}$



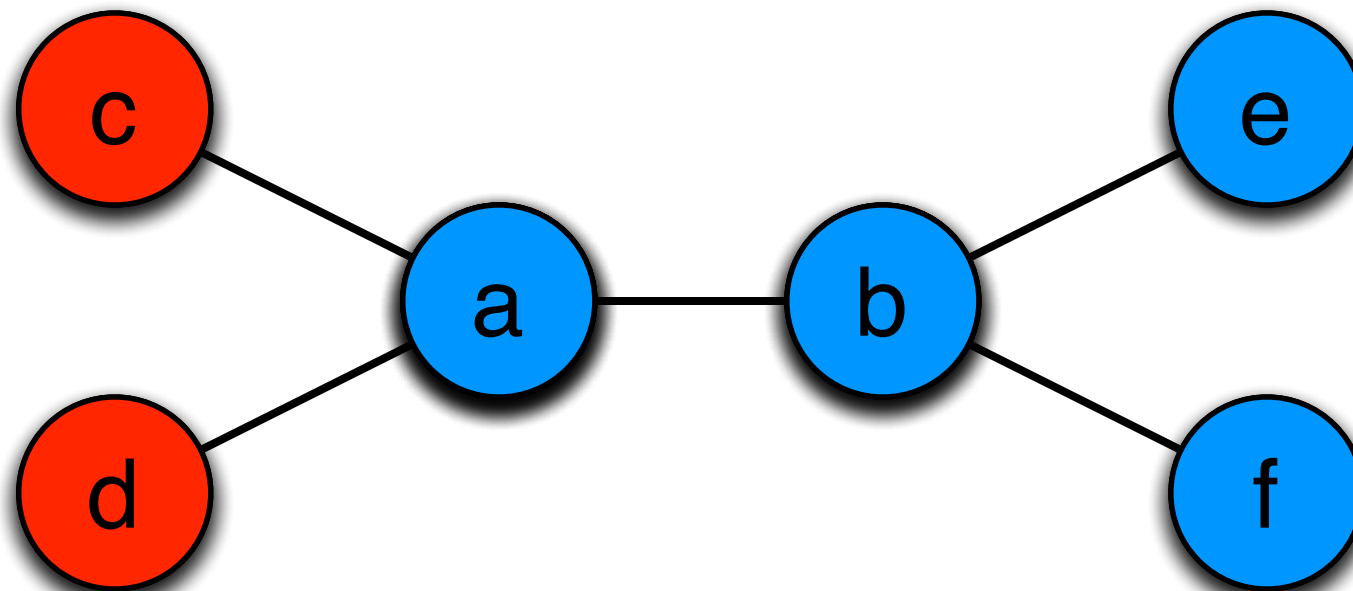
Scheduling

- **Scheduler** (a.k.a. **Daemon**): the *daemon* chooses among activatable processes those that will execute their actions
- can be seen as an *adversary* whose role is to prevent stabilization

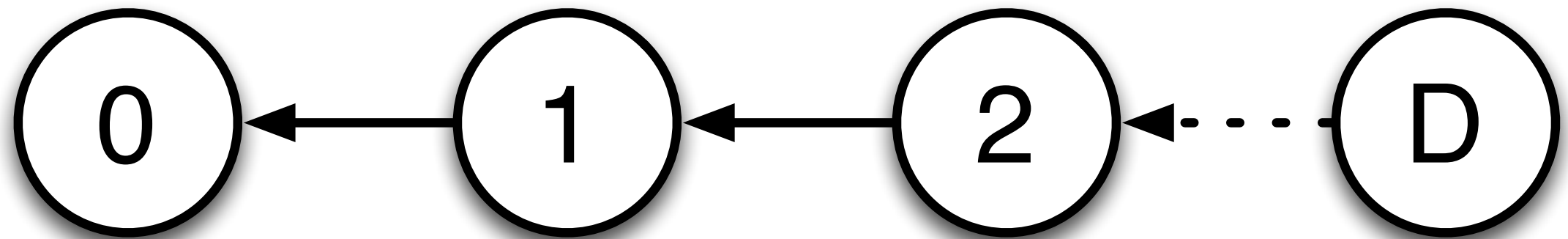
Spatial Scheduling

$true \rightarrow color_i := Min\{\Delta \setminus \{color_j | j \in Neighbors_i\}\}$

$$\Delta = \{\text{red circle}, \text{blue circle}, \text{yellow circle}, \text{green circle}\}$$

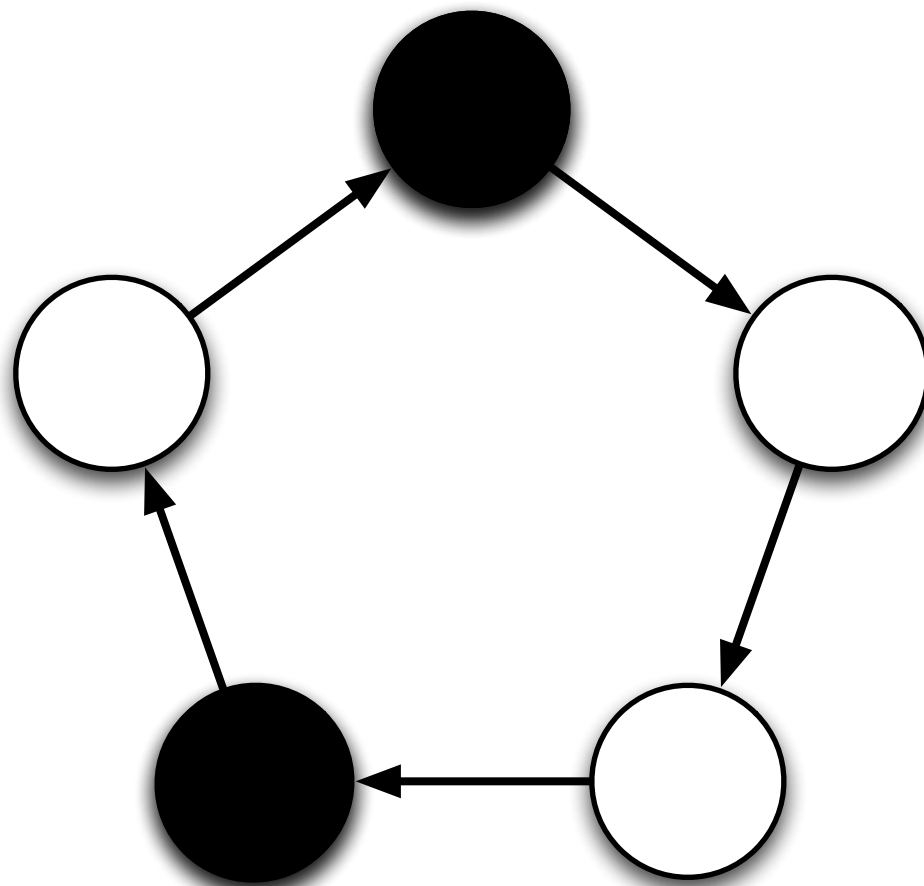


Spatial Scheduling



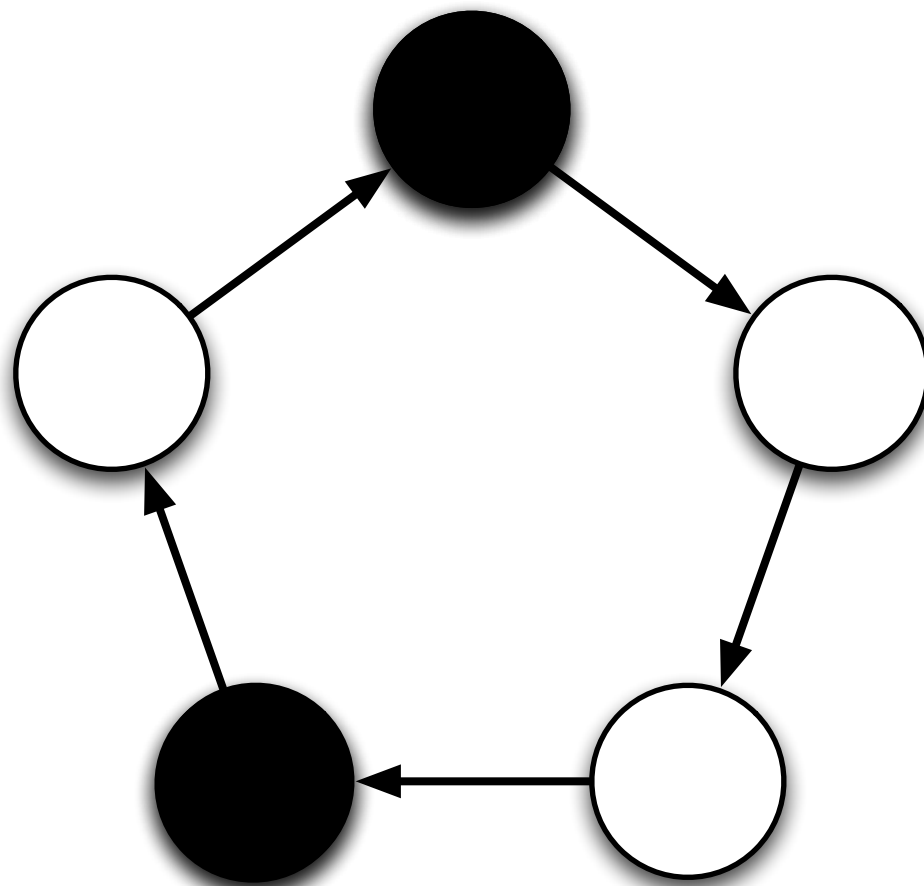
Temporal Scheduling

token \rightarrow *pass token to left neighbor with probability* $\frac{1}{2}$
token = ● *no token* = ○

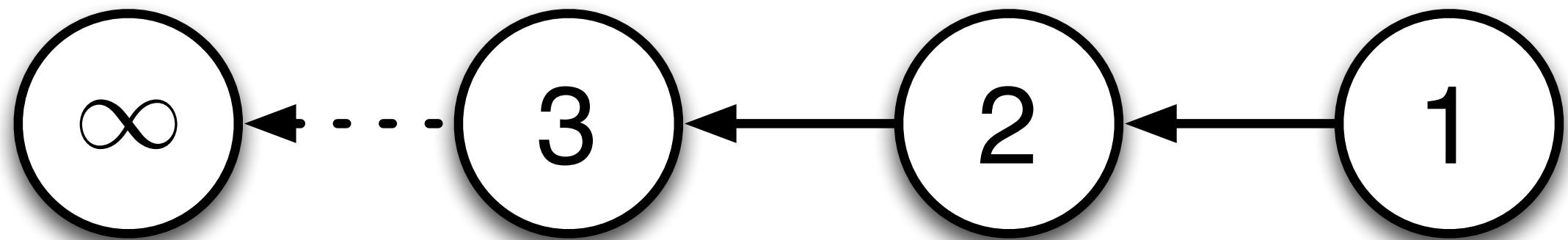


Temporal Scheduling

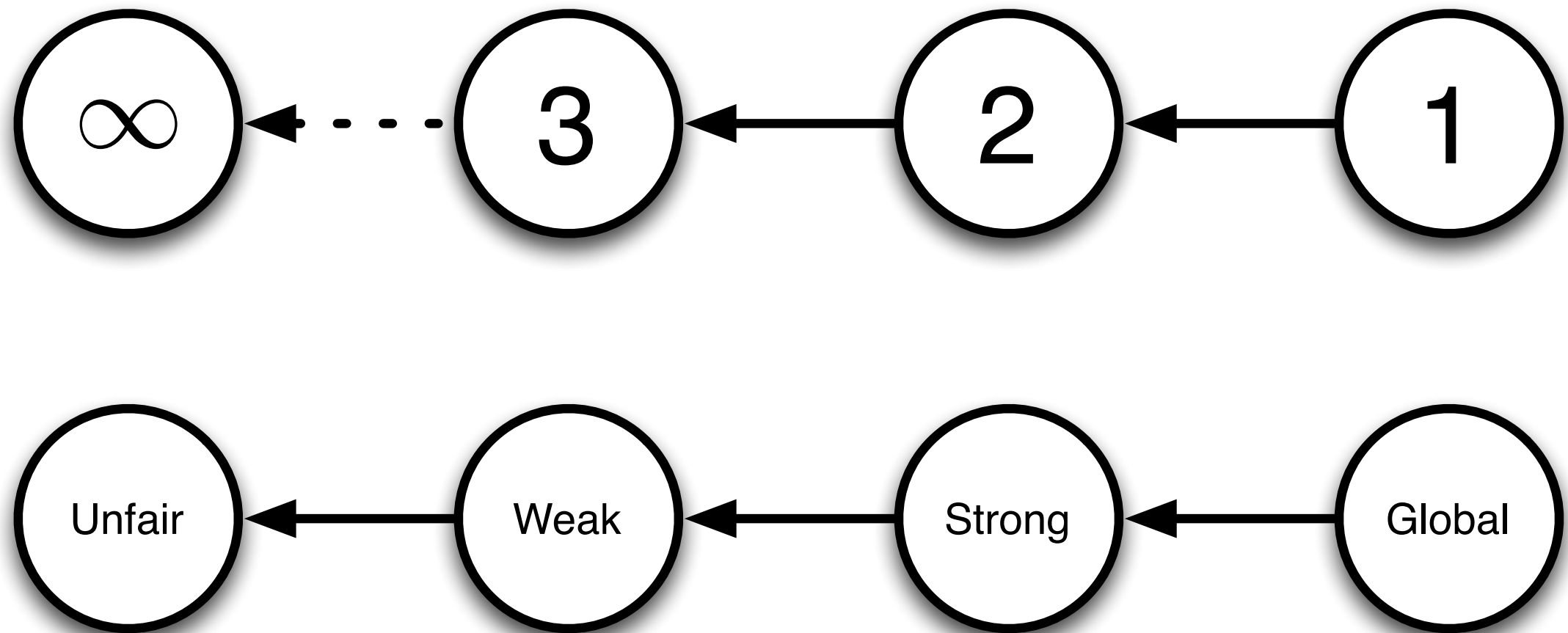
token \rightarrow *pass token to left neighbor with probability* $\frac{1}{2}$
token = ● *no token* = ○



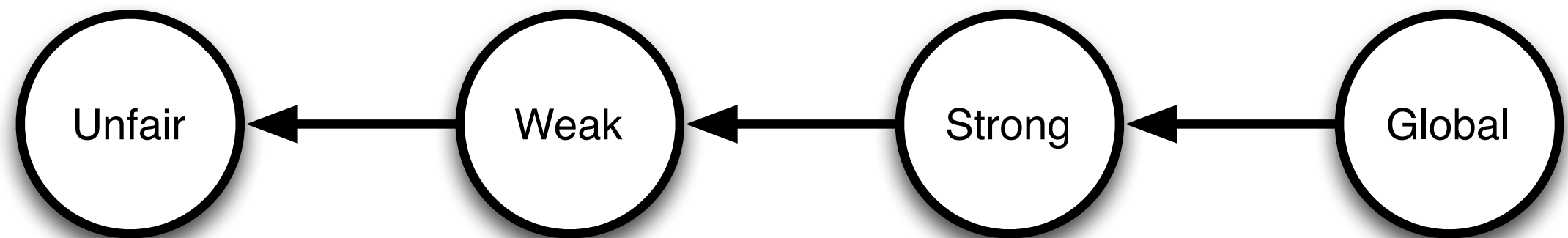
Temporal Scheduling



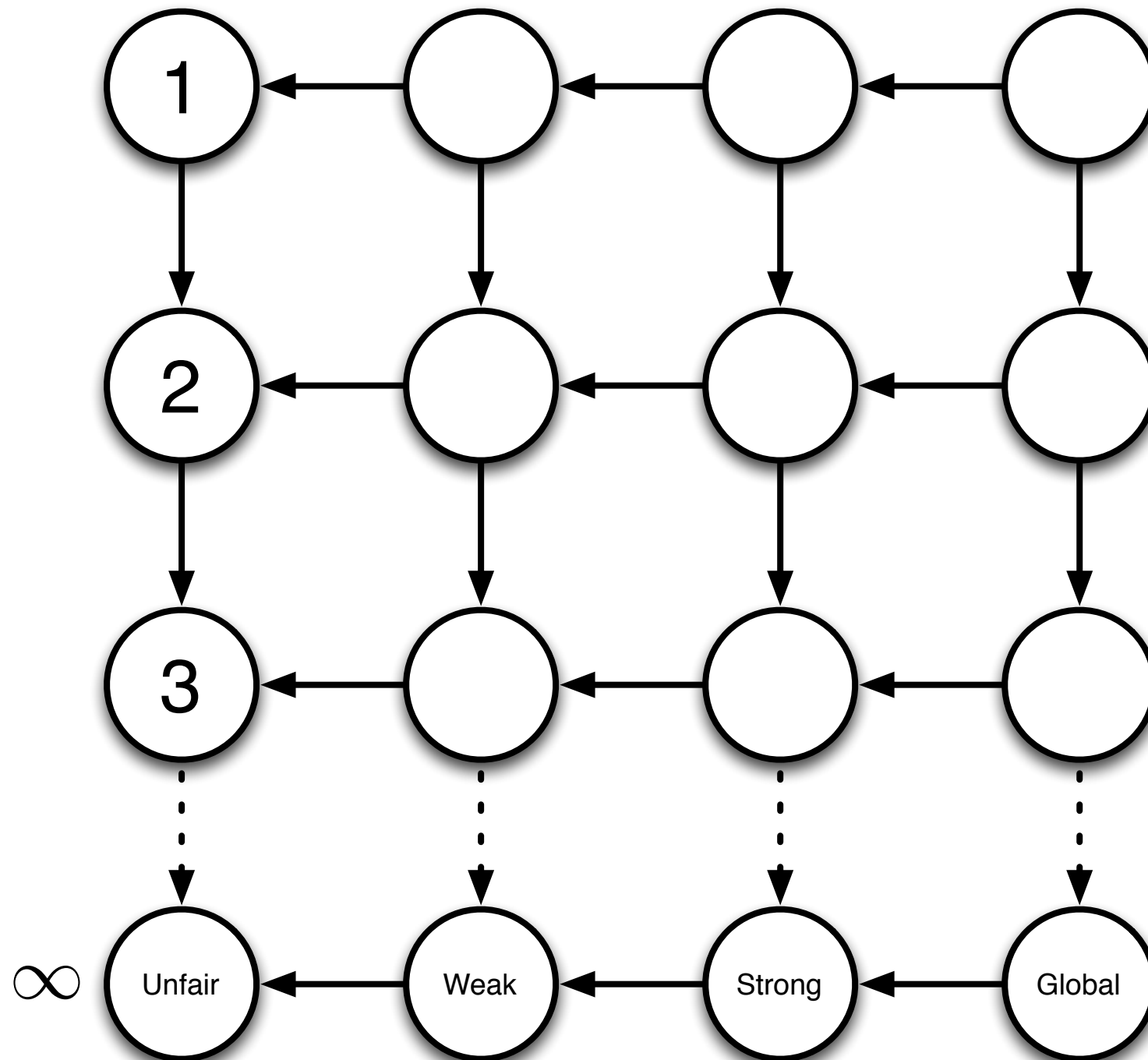
Temporal Scheduling



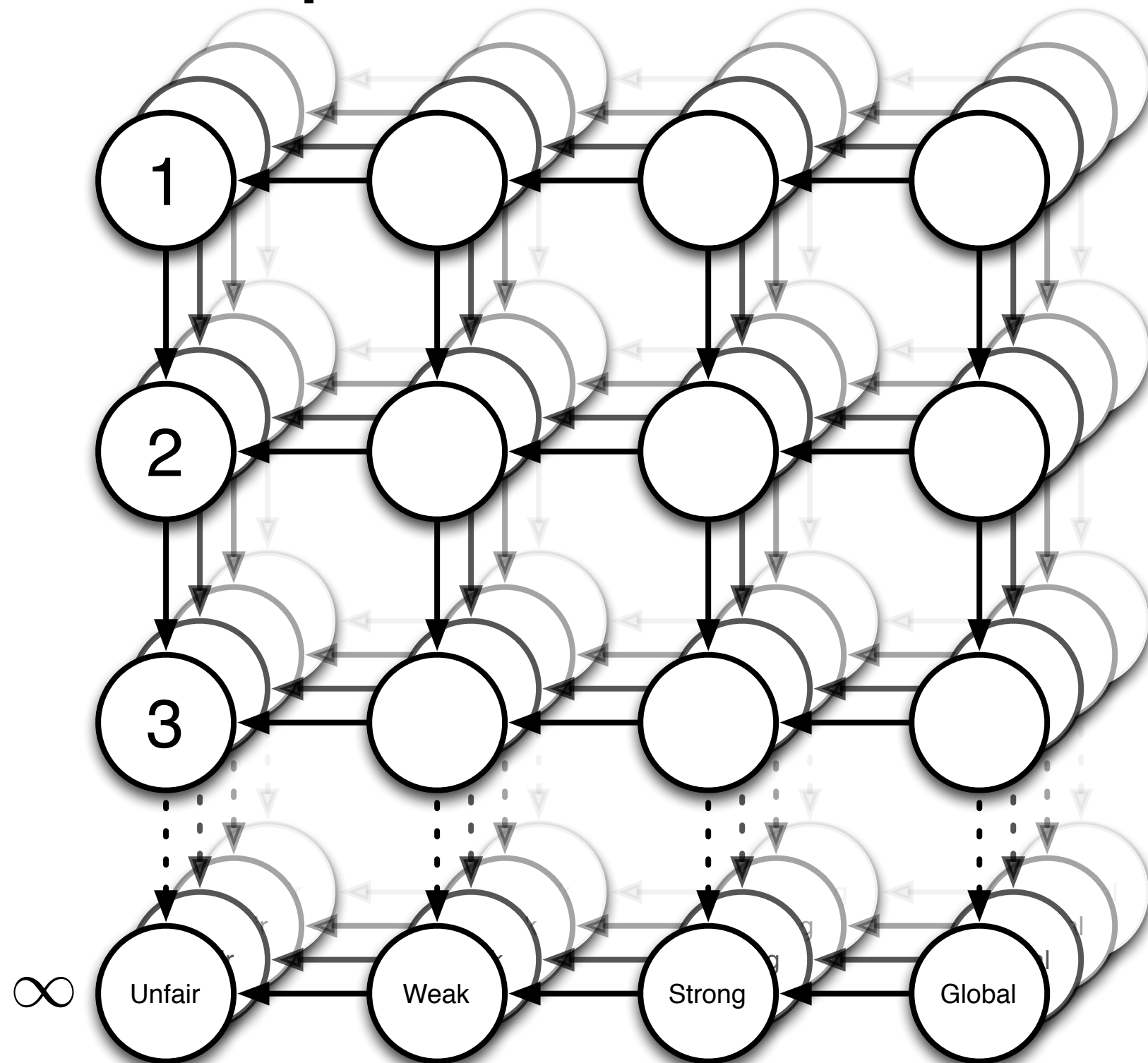
A Map of Daemons



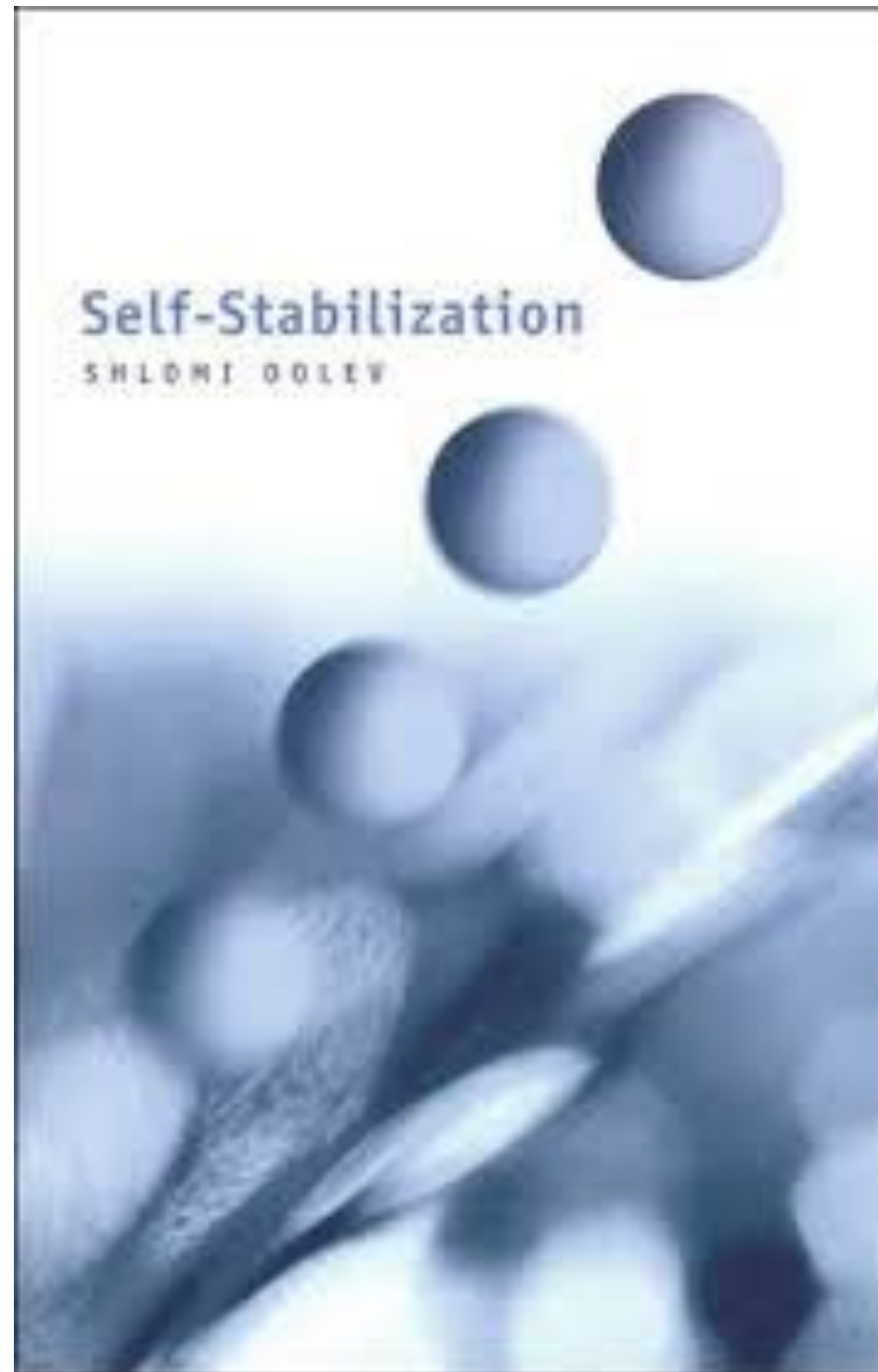
A Map of Daemons



A Map of Daemons



Self-stabilization



Population Protocols

Population Protocols



Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, René Peralta:
Computation in networks of passively mobile finite-state sensors. Distributed
Computing 18(4): 235-253 (2006)

Population Protocols



Population Protocols

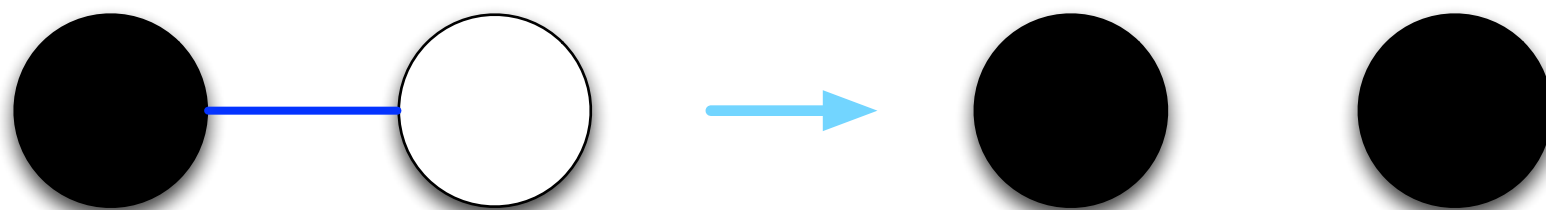


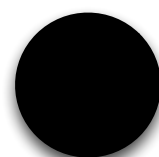
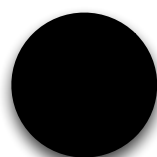
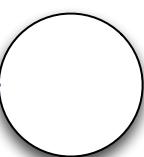
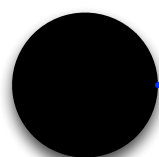
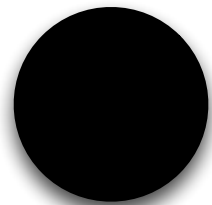
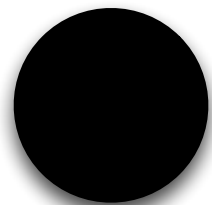
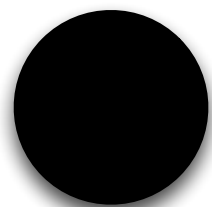
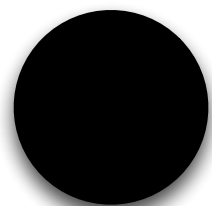
Population Protocols

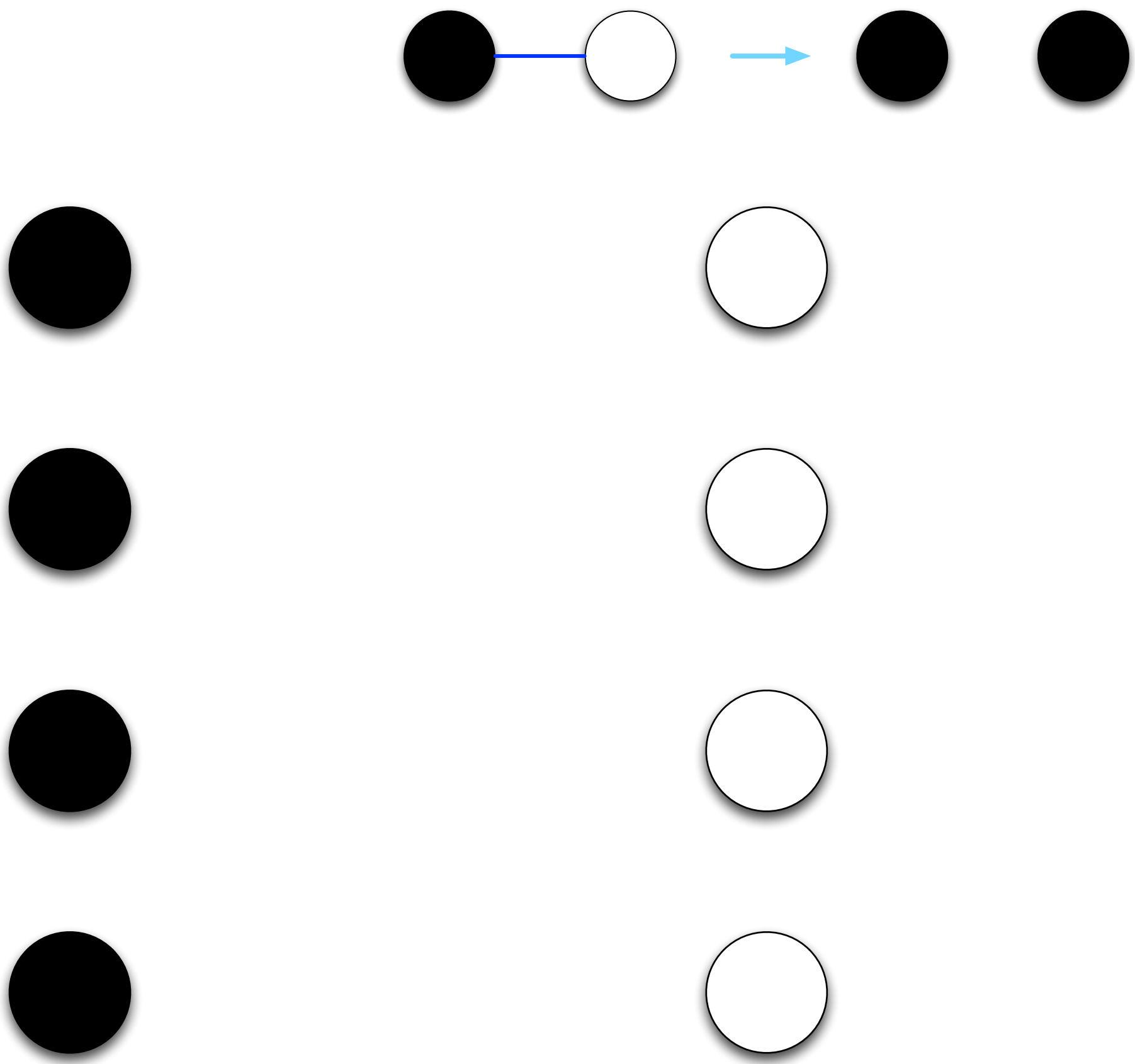
- **Definition**

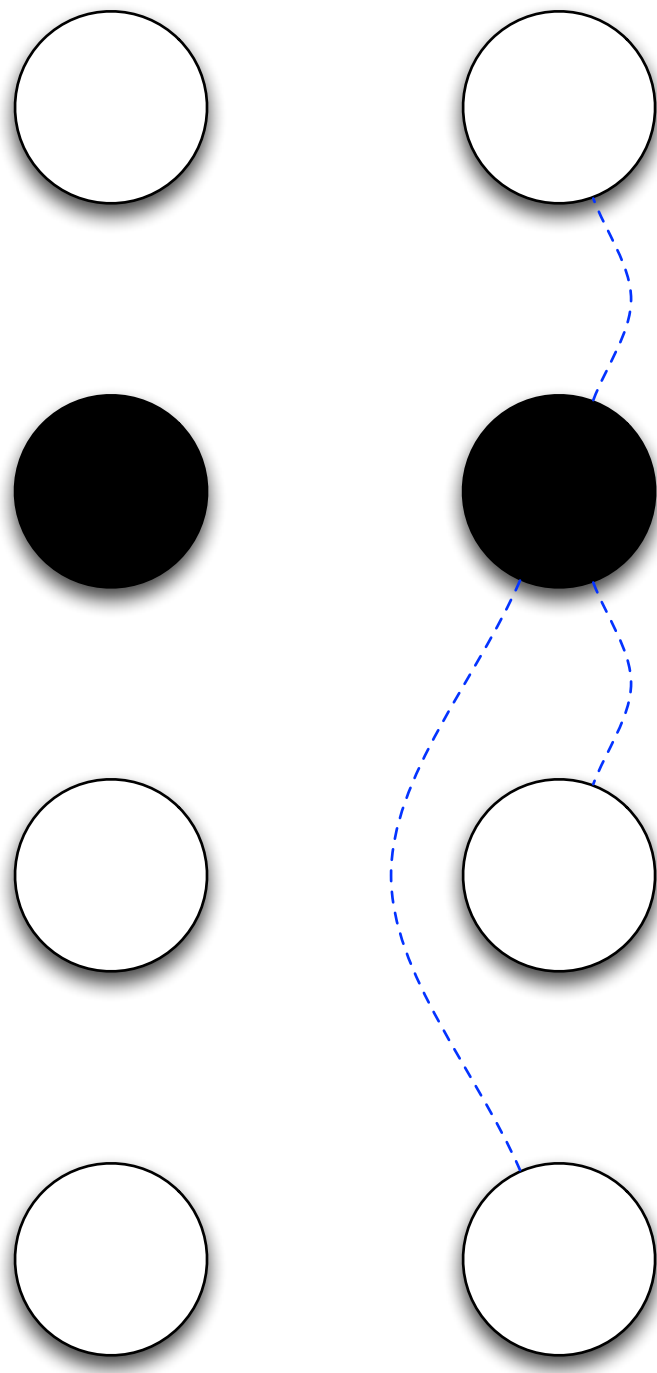
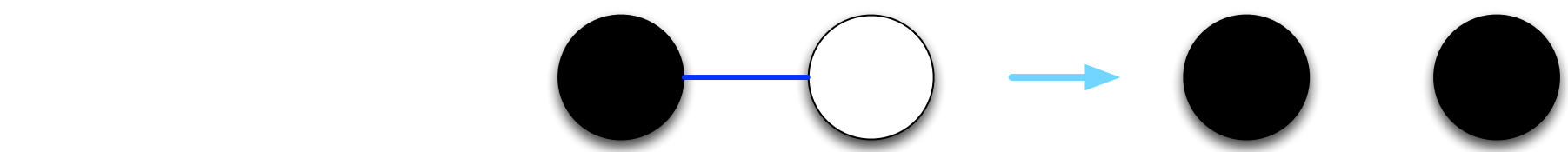
- *A Population Protocol* is a 6-tuple **(X,Y,Q,I,O,T)**
 - **X**: Set of inputs
 - **Y**: Set of outputs
 - **Q**: Set of states
 - **I**: Input mapping function, $X \longrightarrow Q$
 - **O**: Output mapping function, $Q \longrightarrow O$
 - **T**: Transition function, $Q \times Q \longrightarrow Q \times Q$

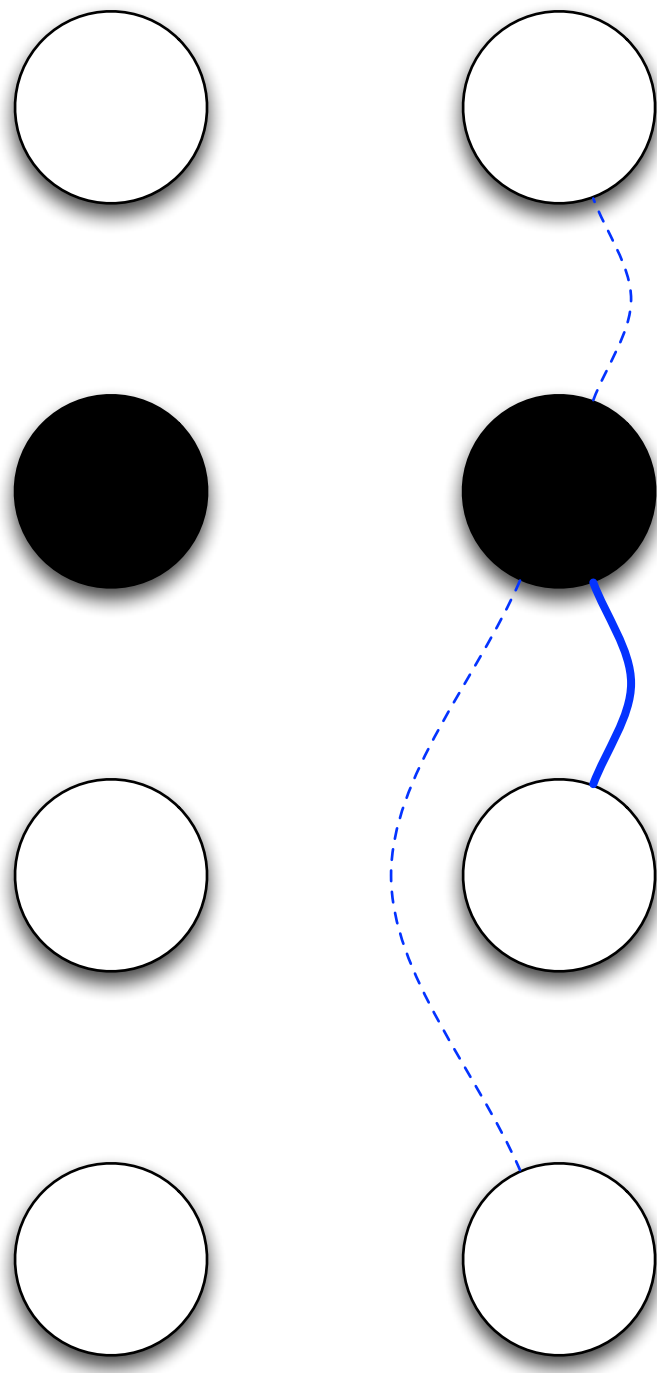
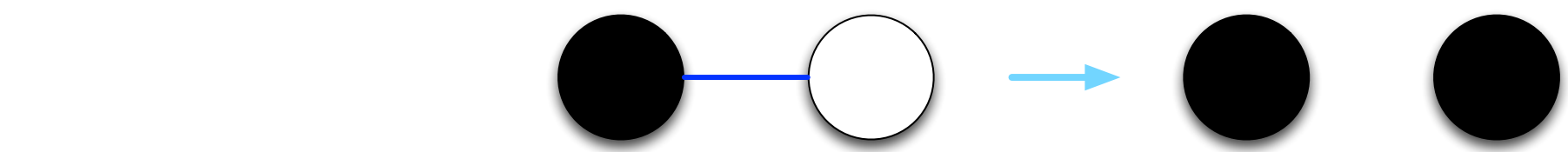
Example 1

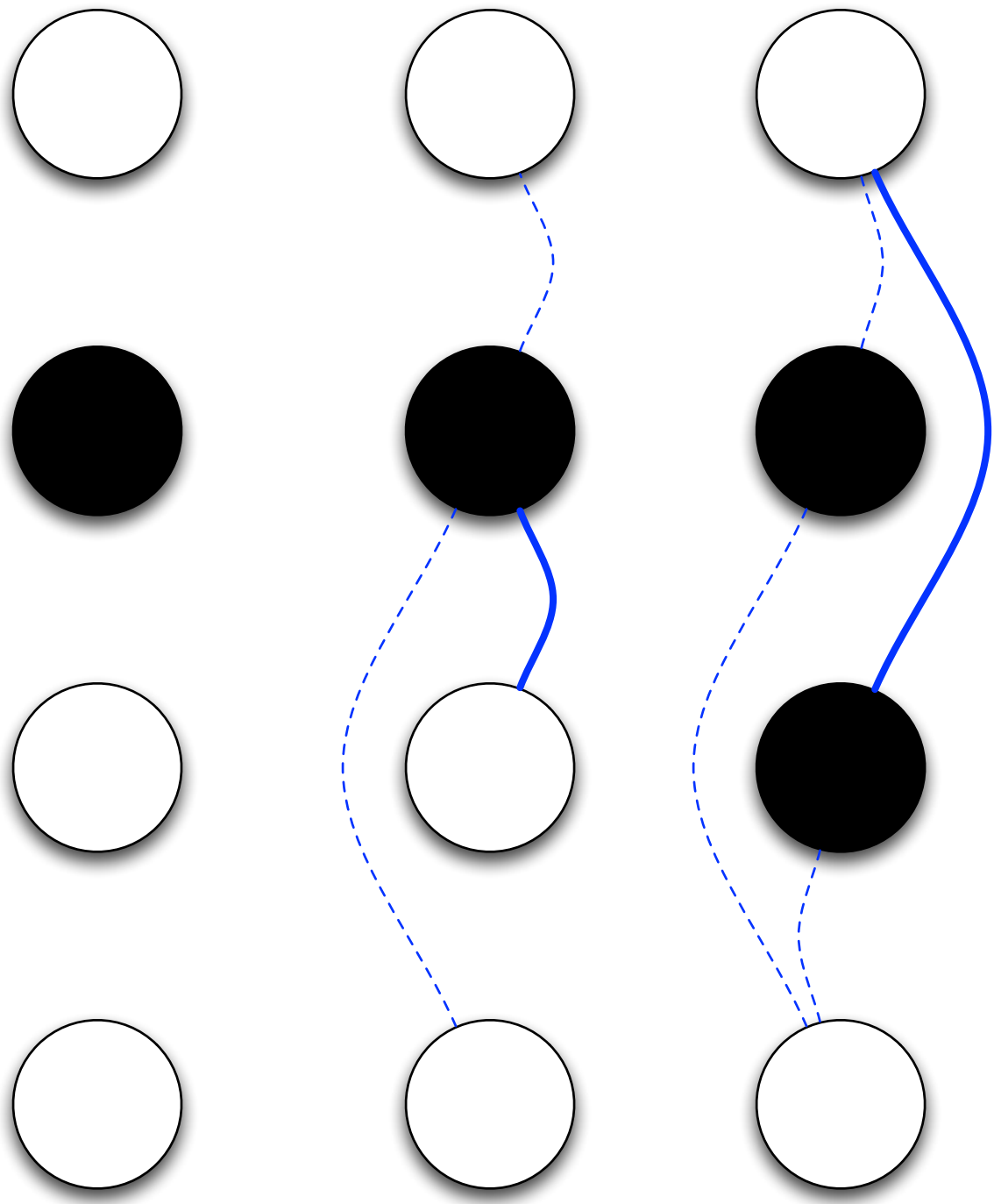
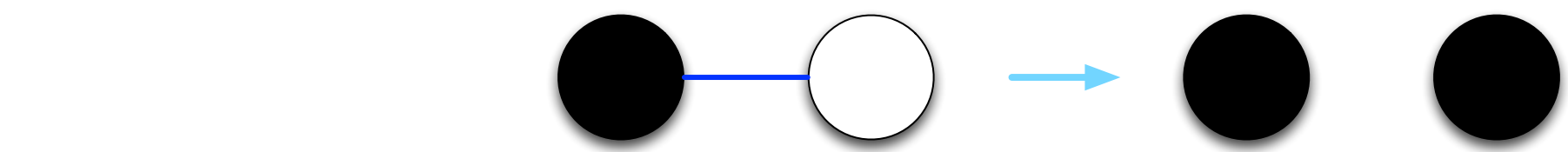


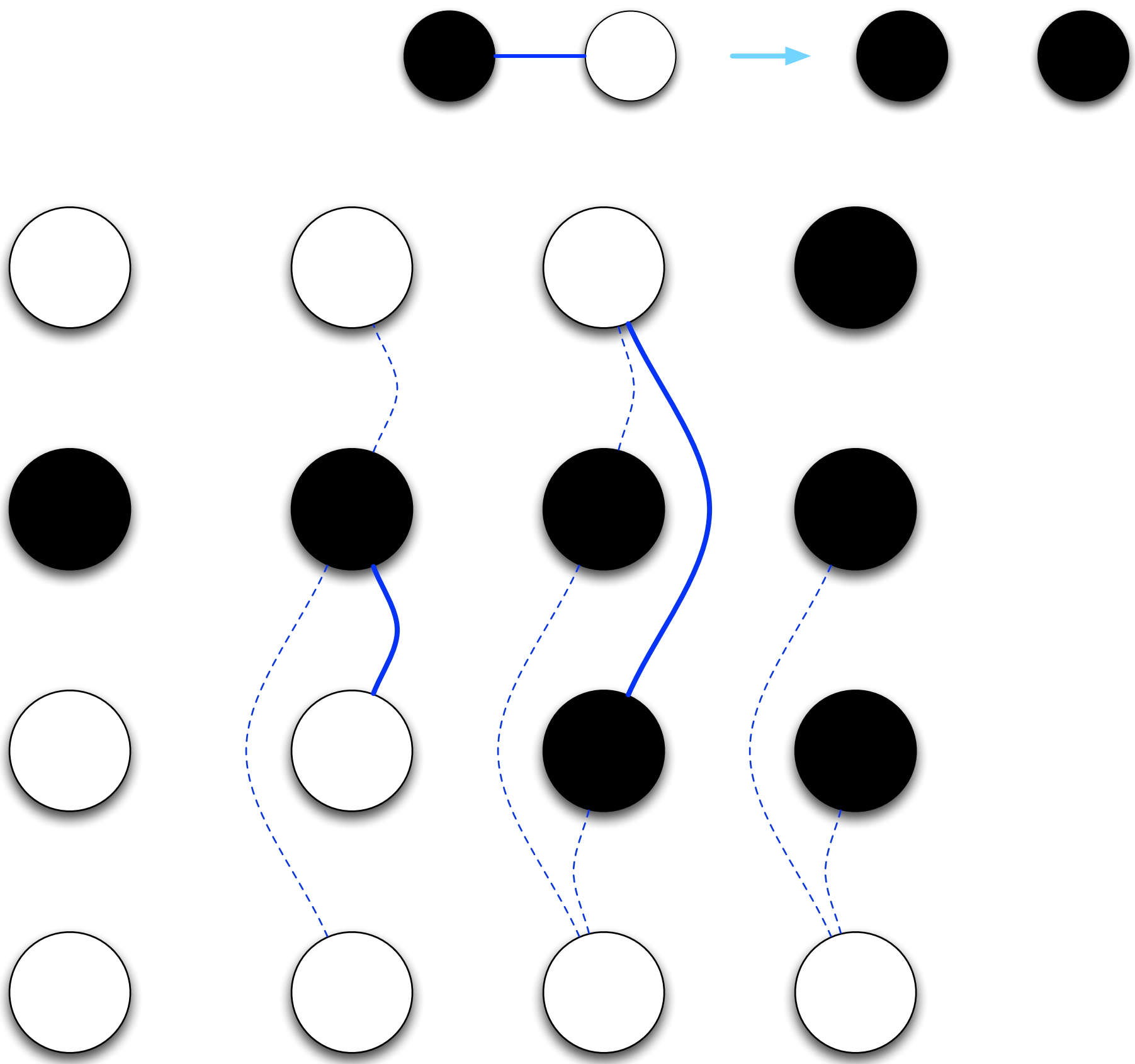


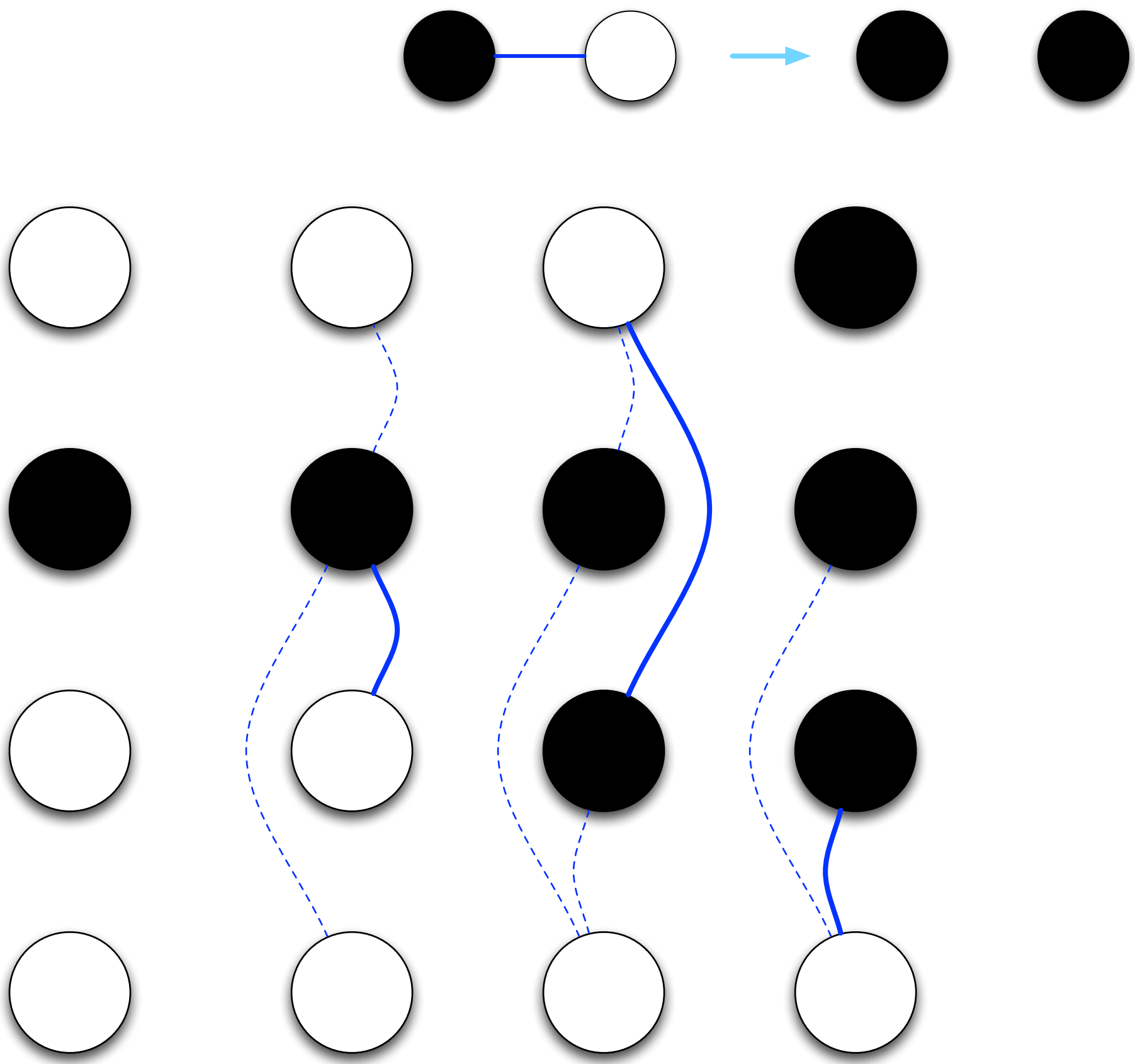


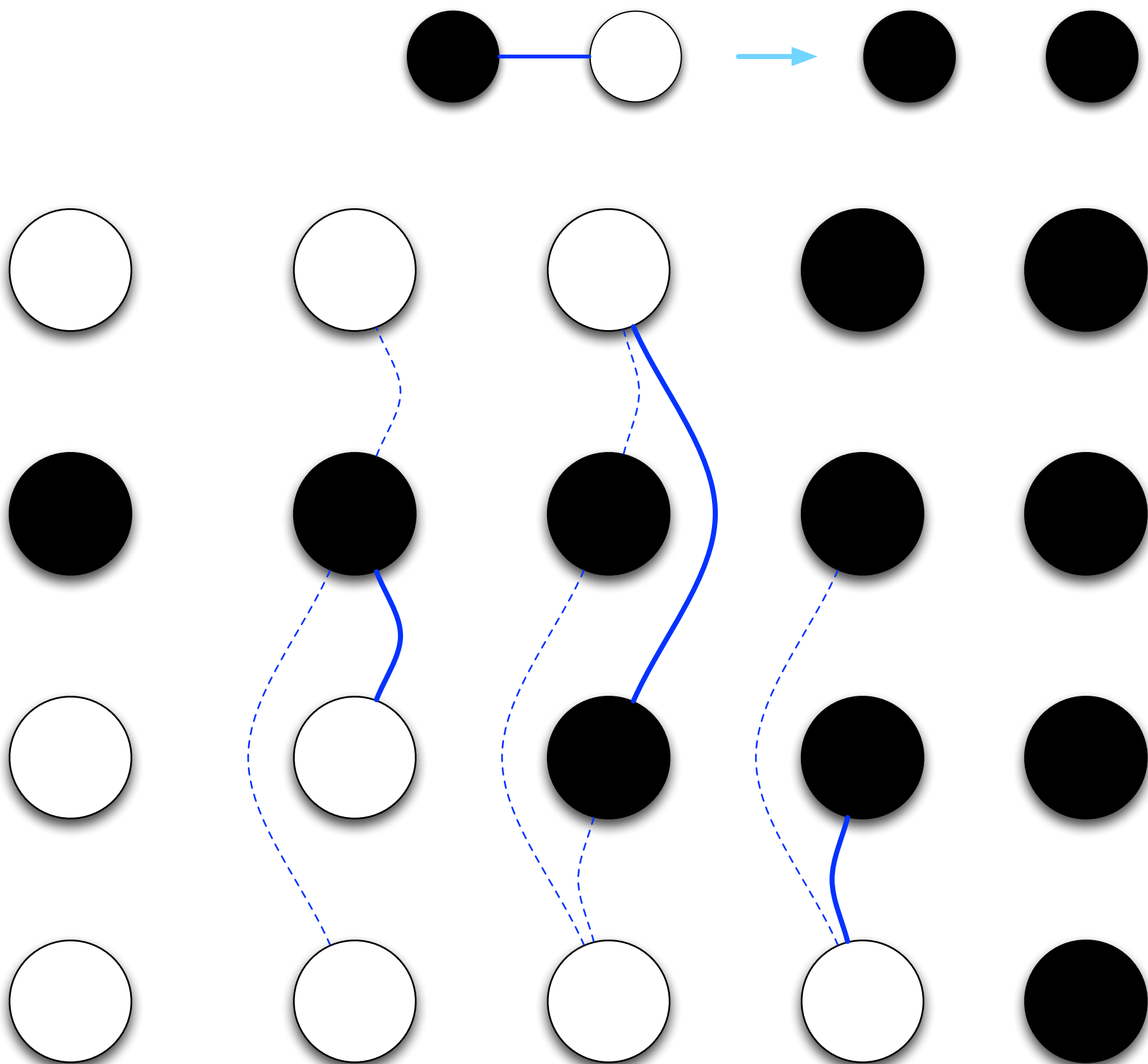




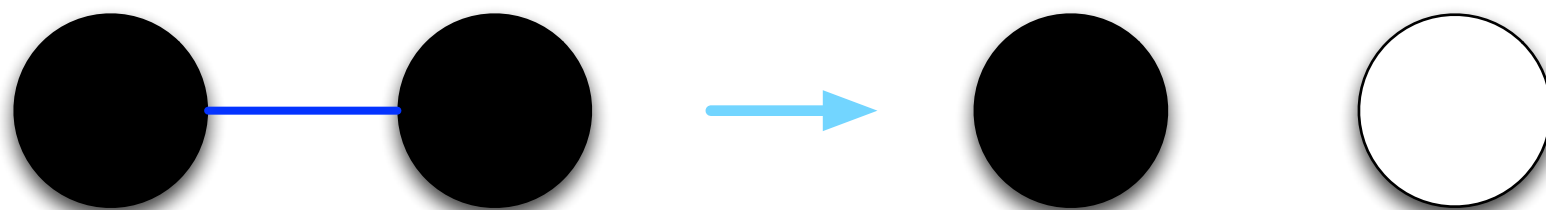


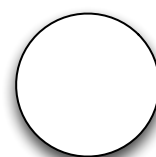
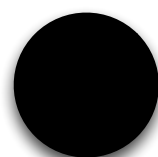
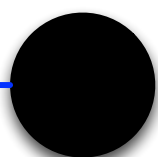
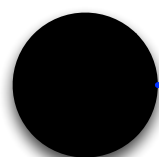
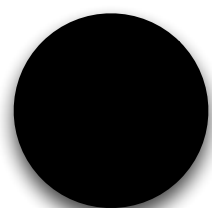
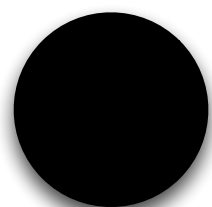
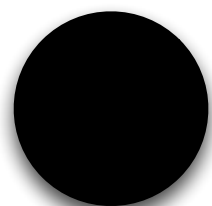
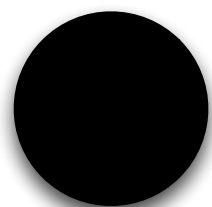


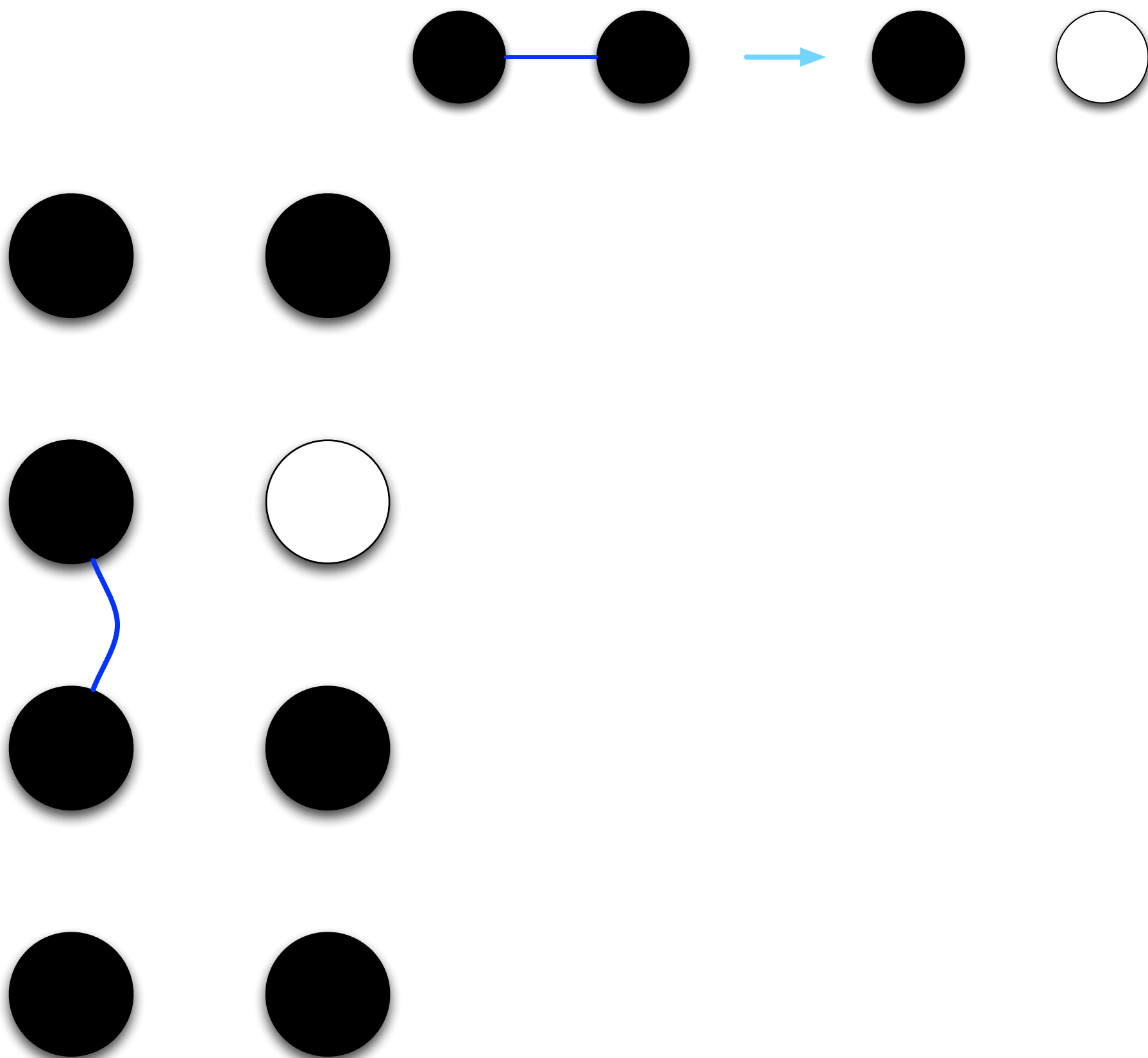


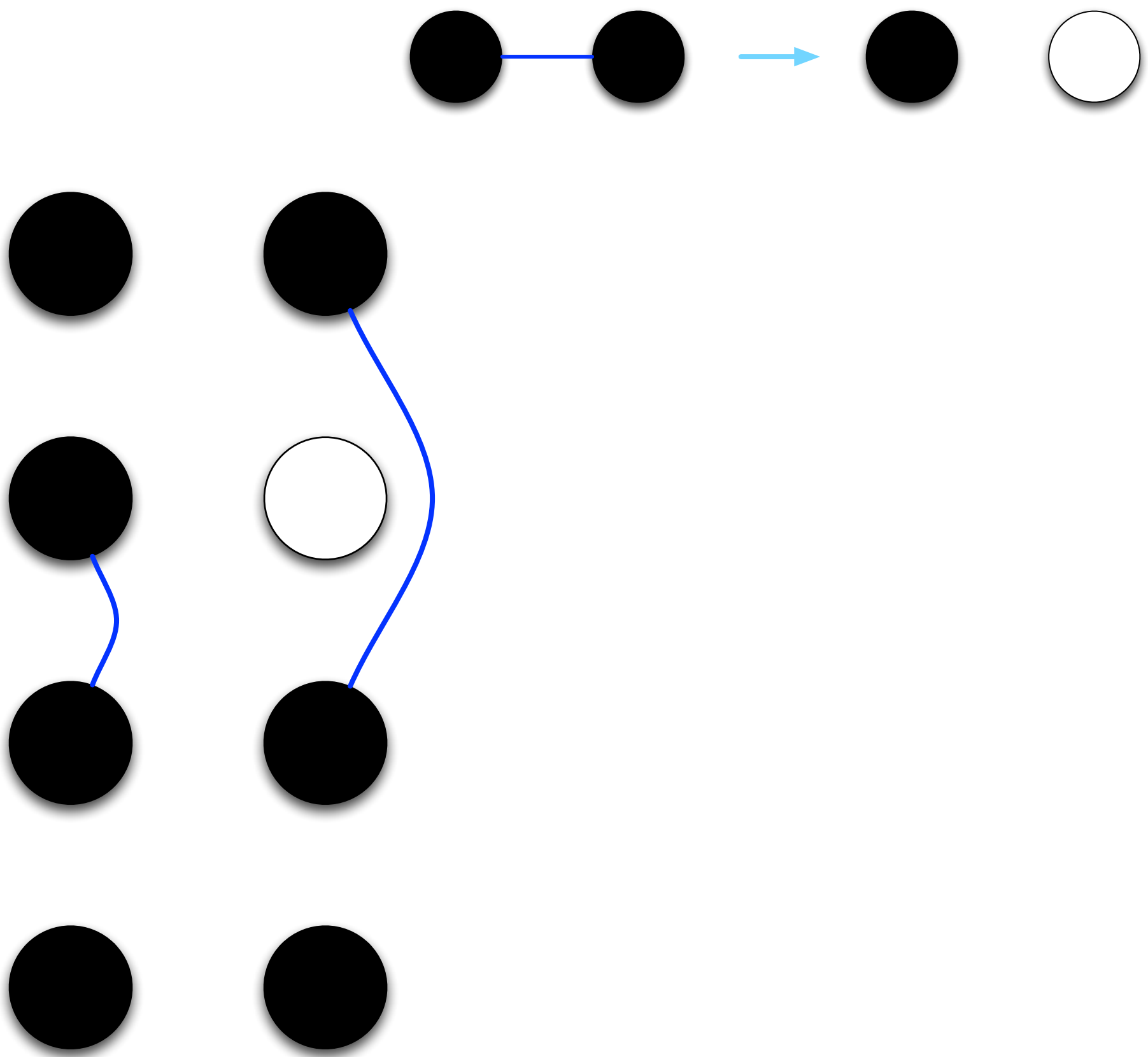


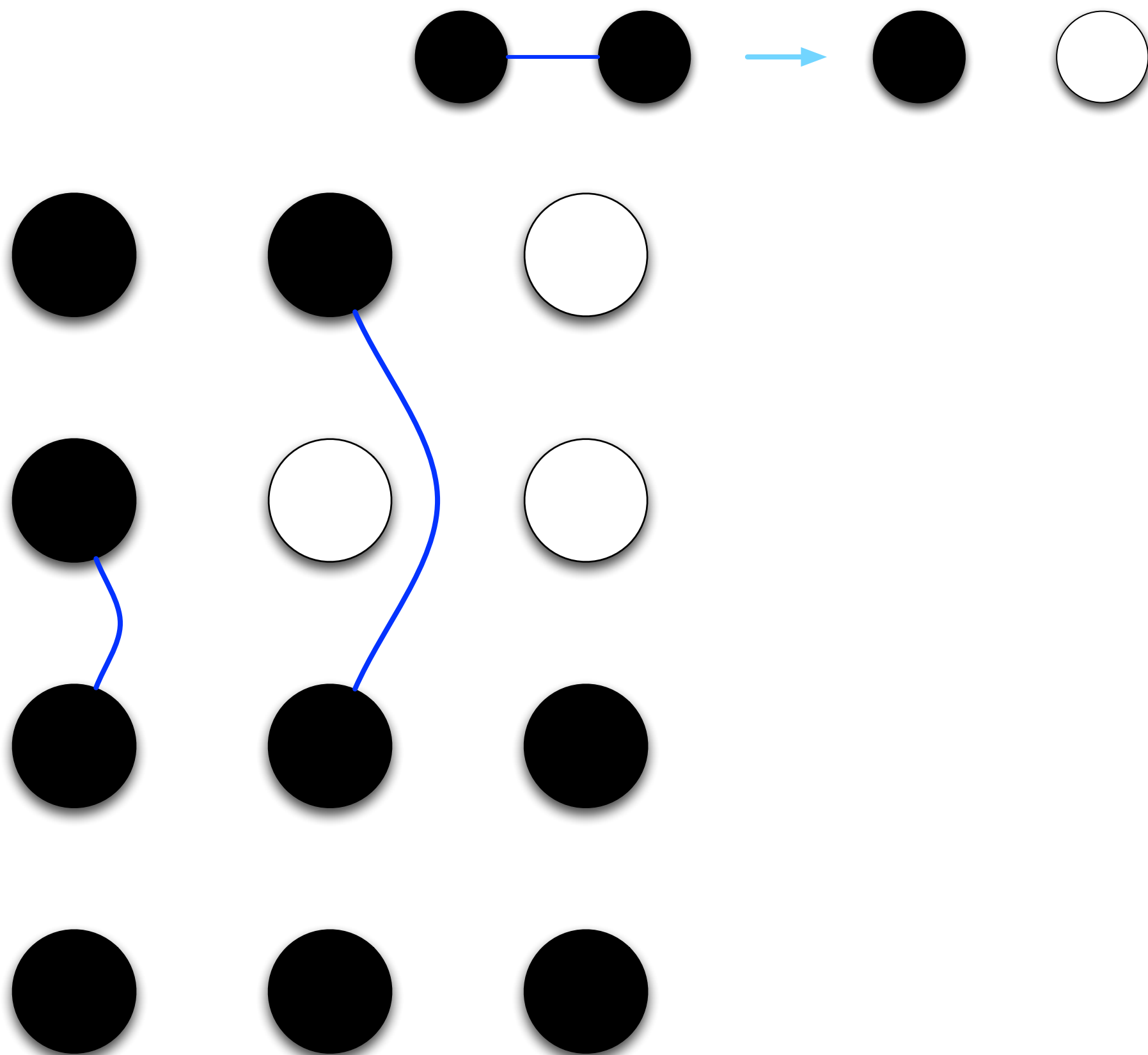
Example 1b

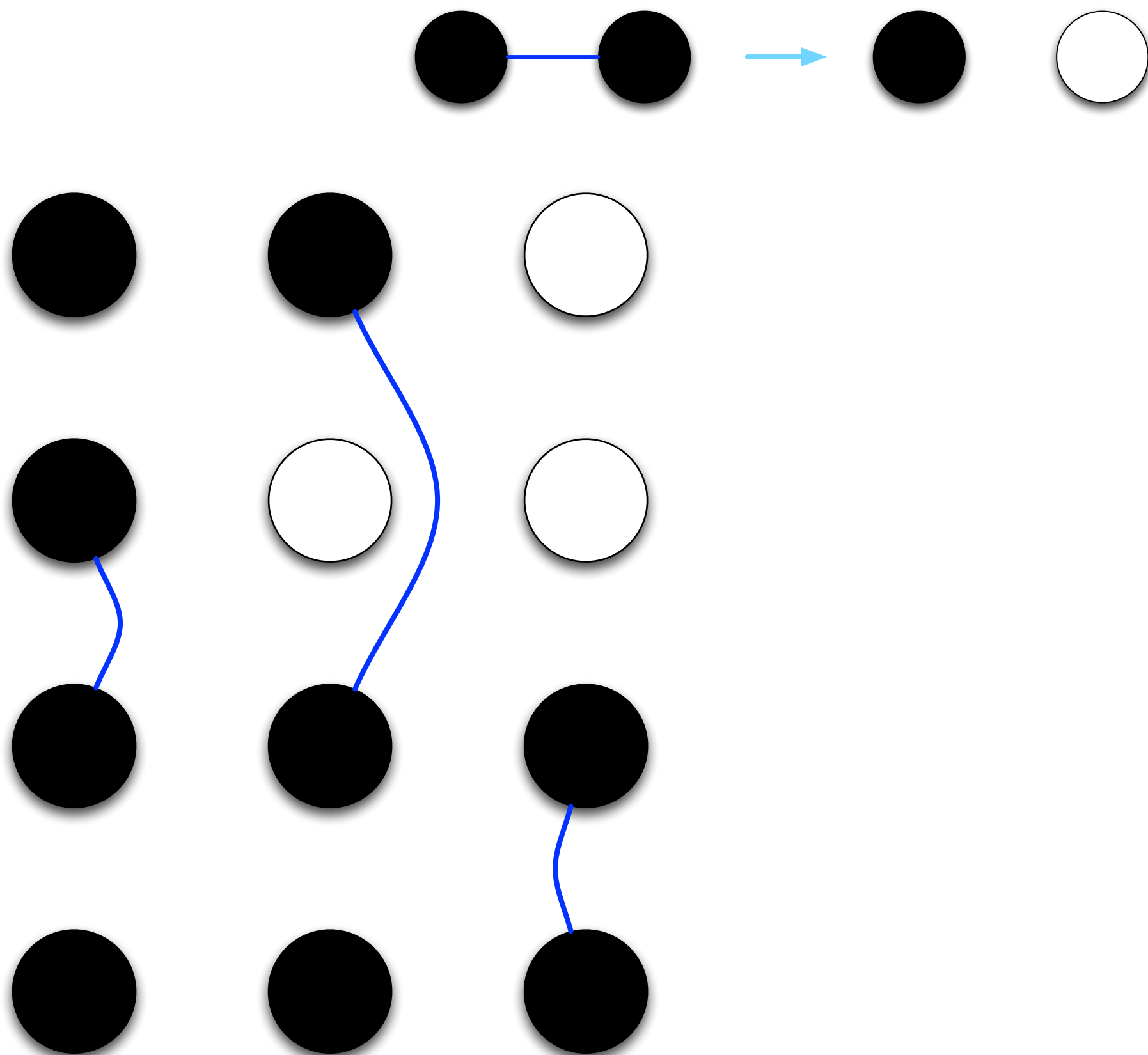


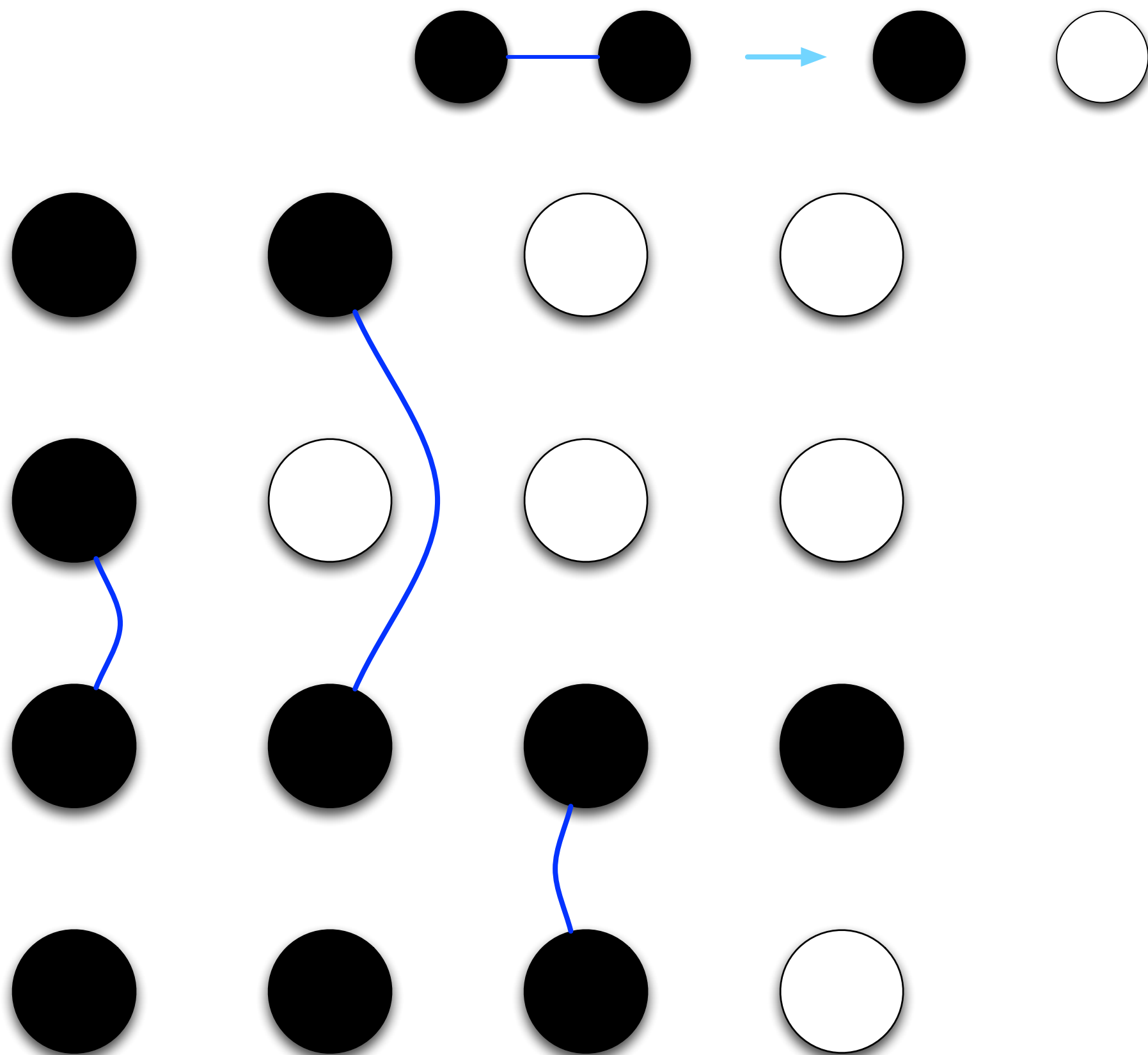




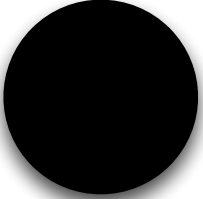
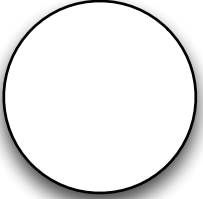


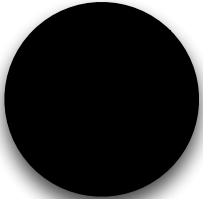
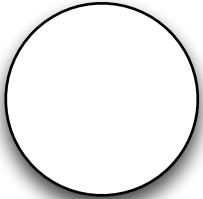




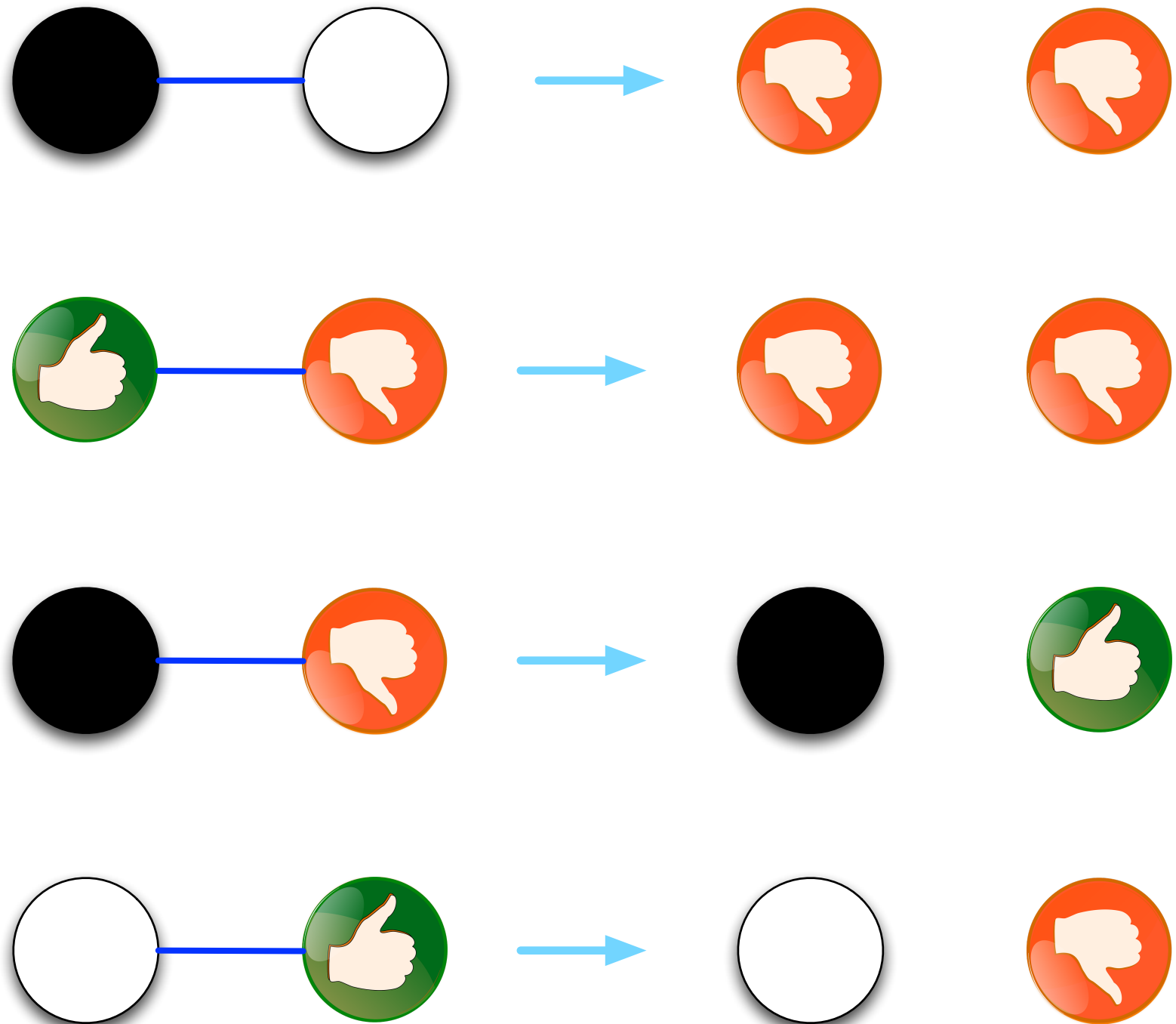


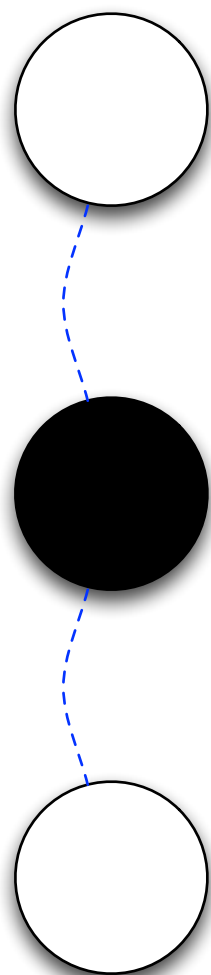
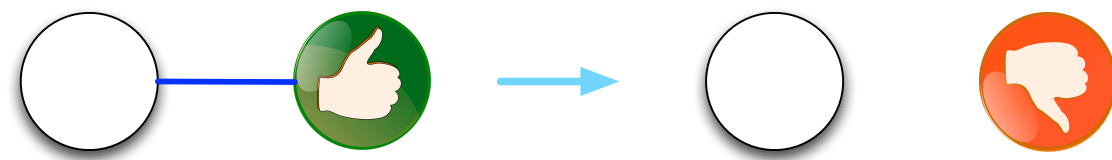
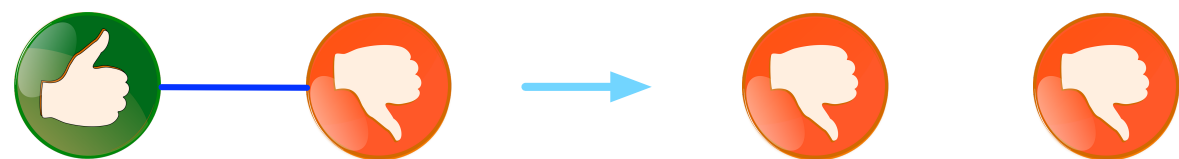
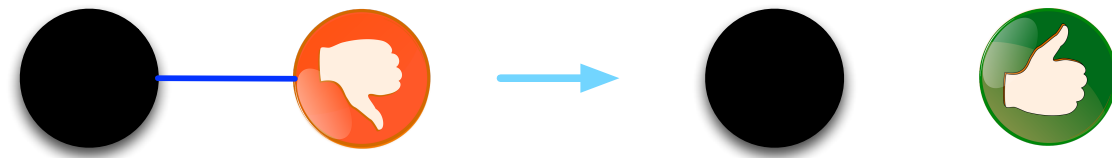
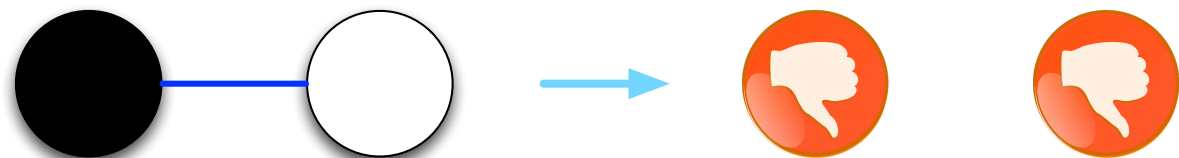


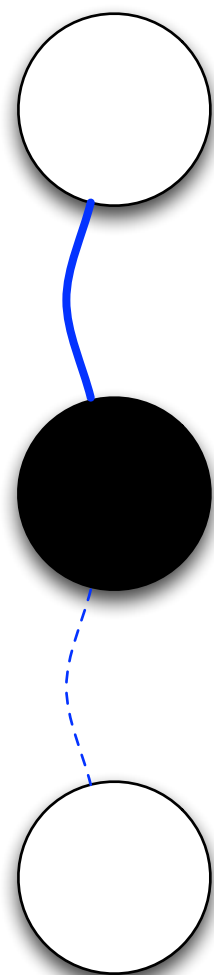
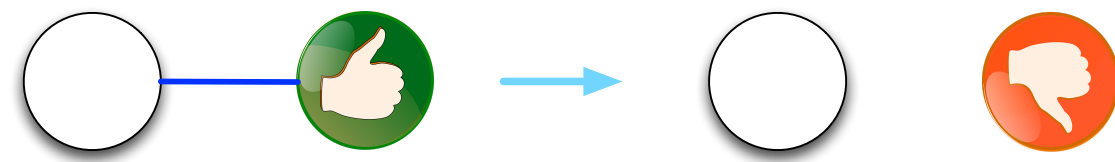
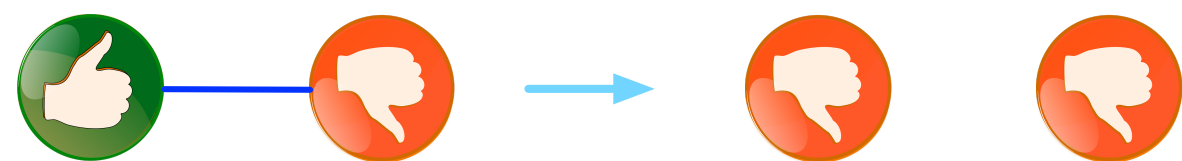
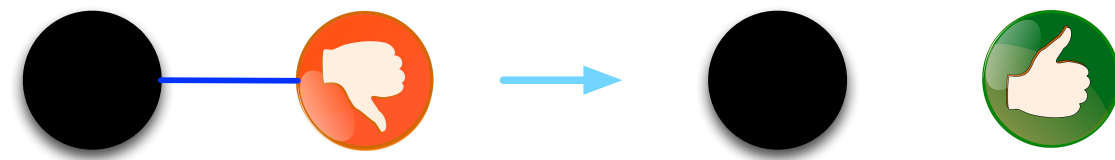
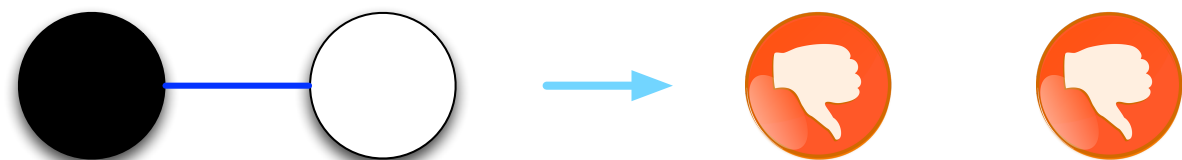
Example 2

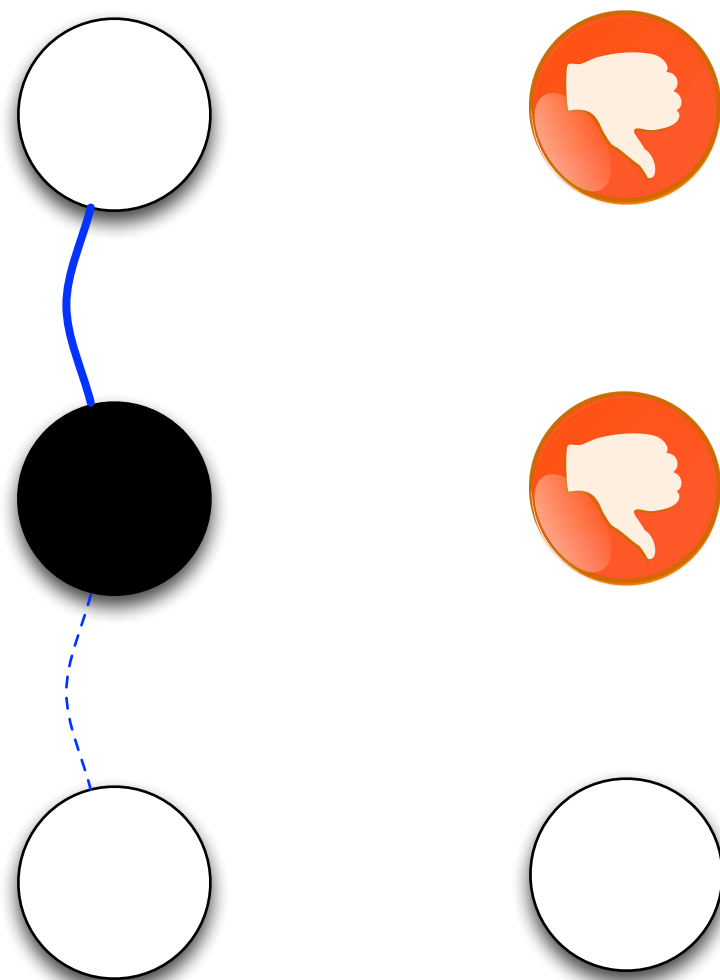
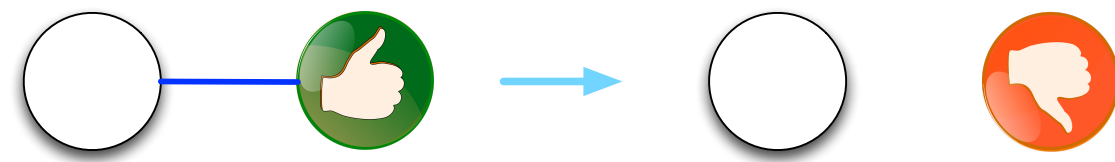
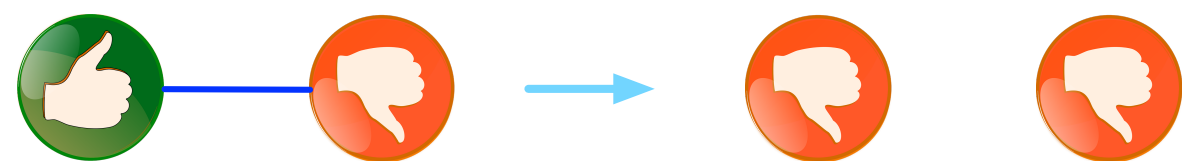
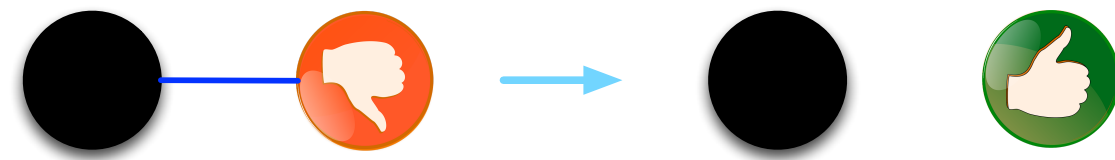
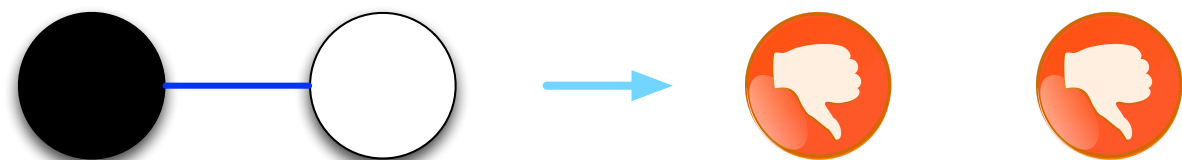
- **Inputs:**  
- **Outputs:**  
- **#**  **<** **#**  **?**

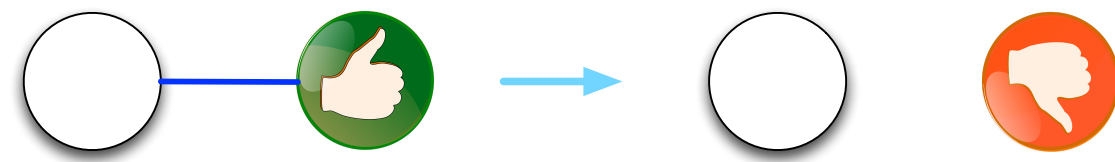
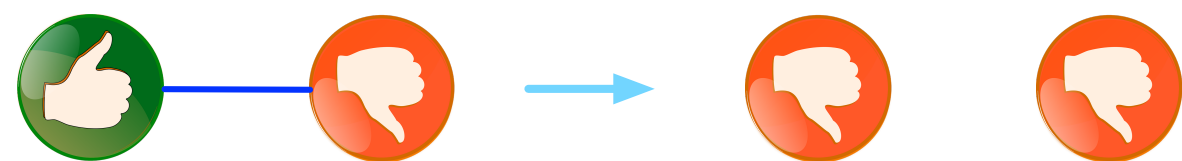
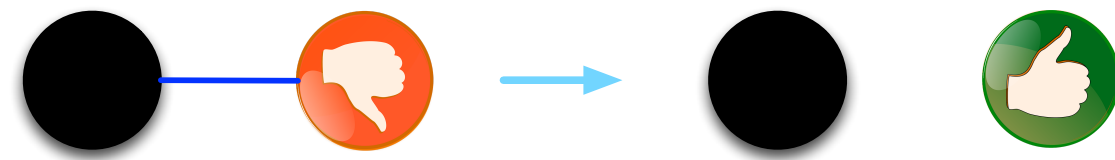
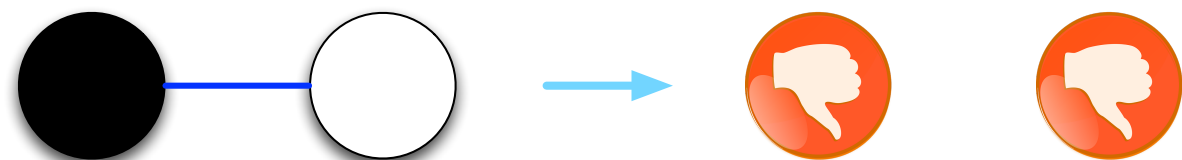
Example 2

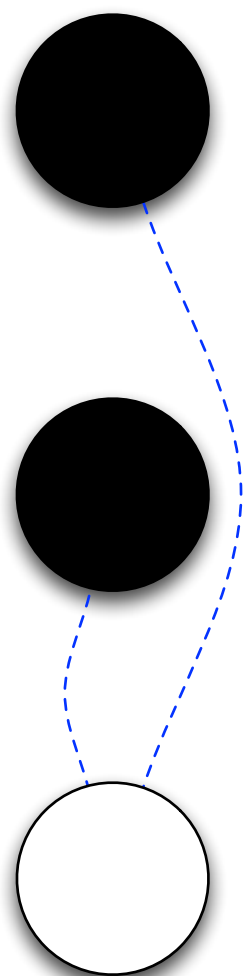
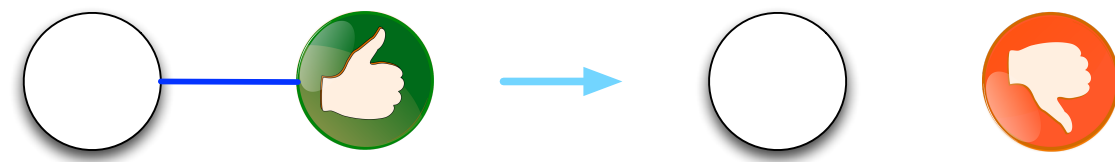
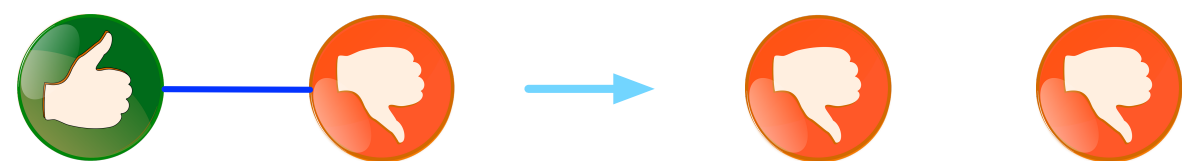
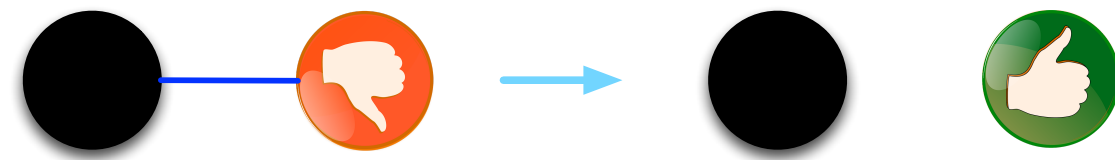
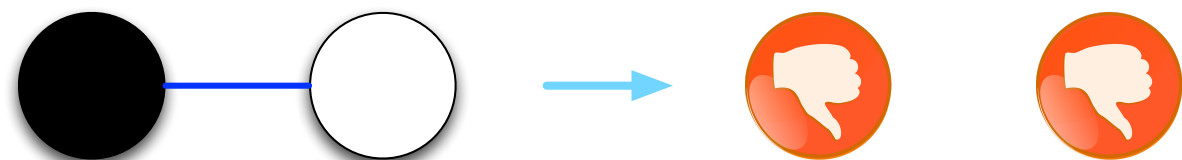


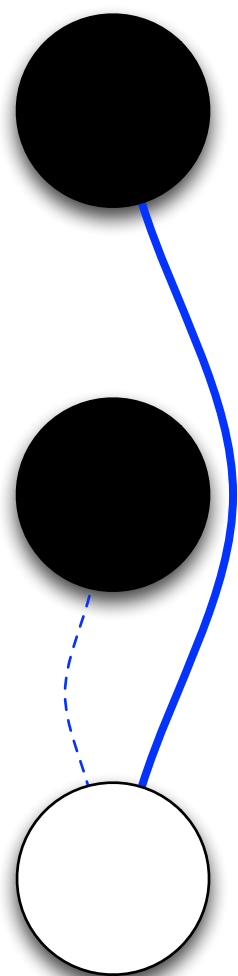
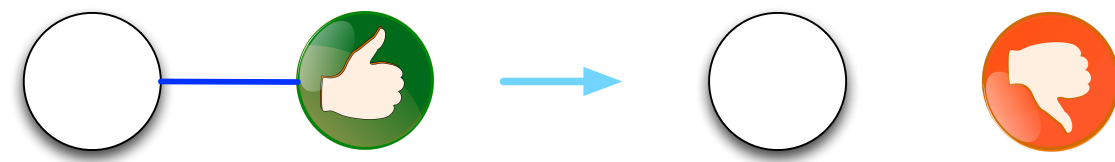
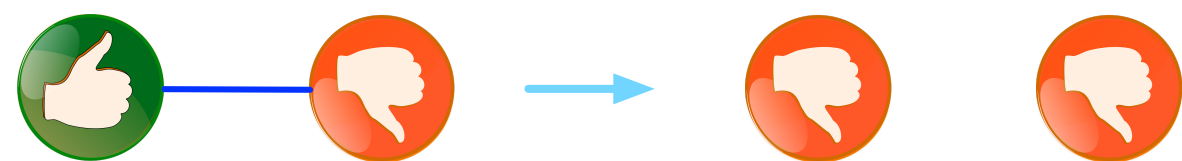
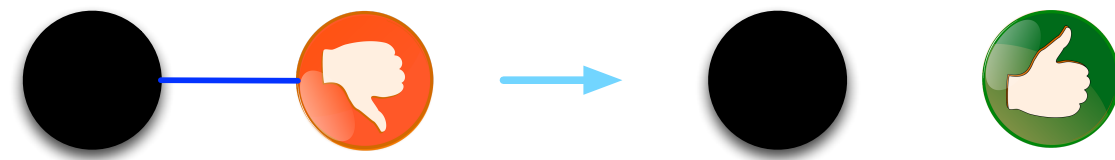
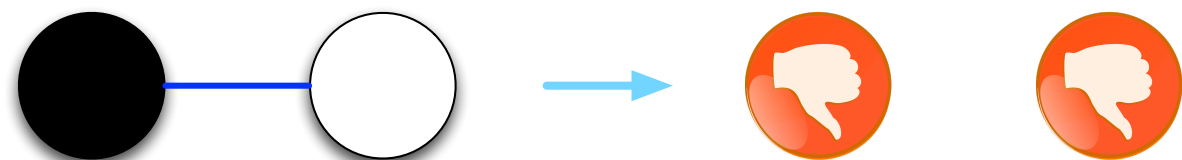


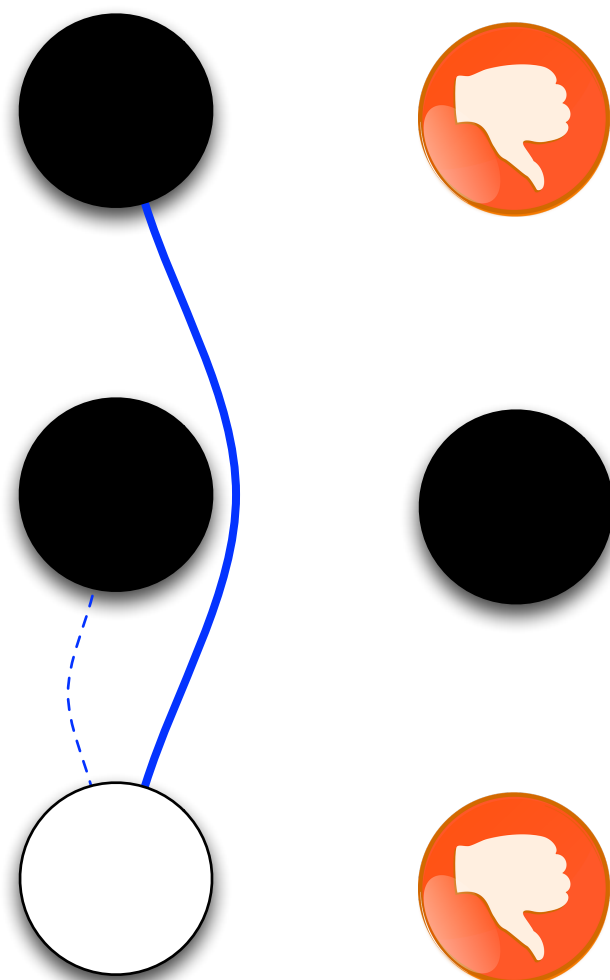
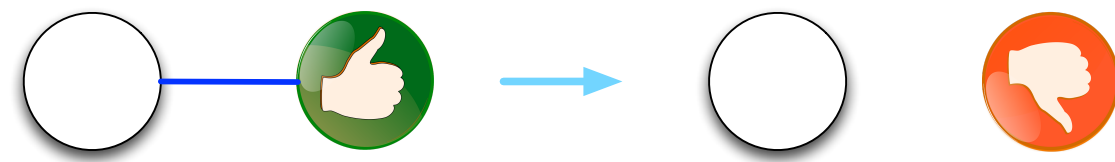
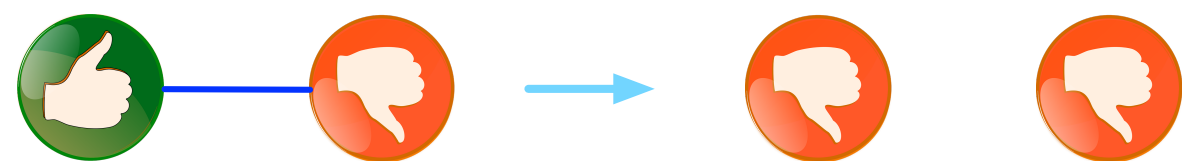
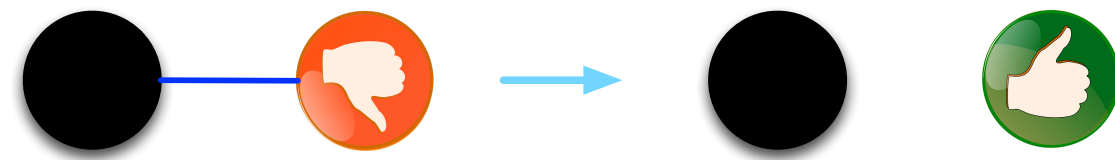
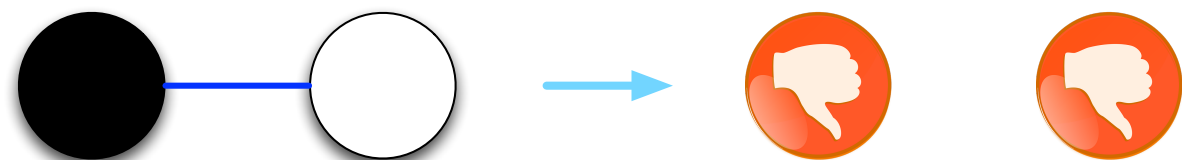


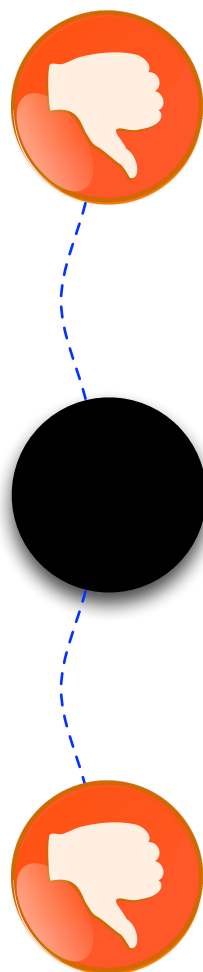
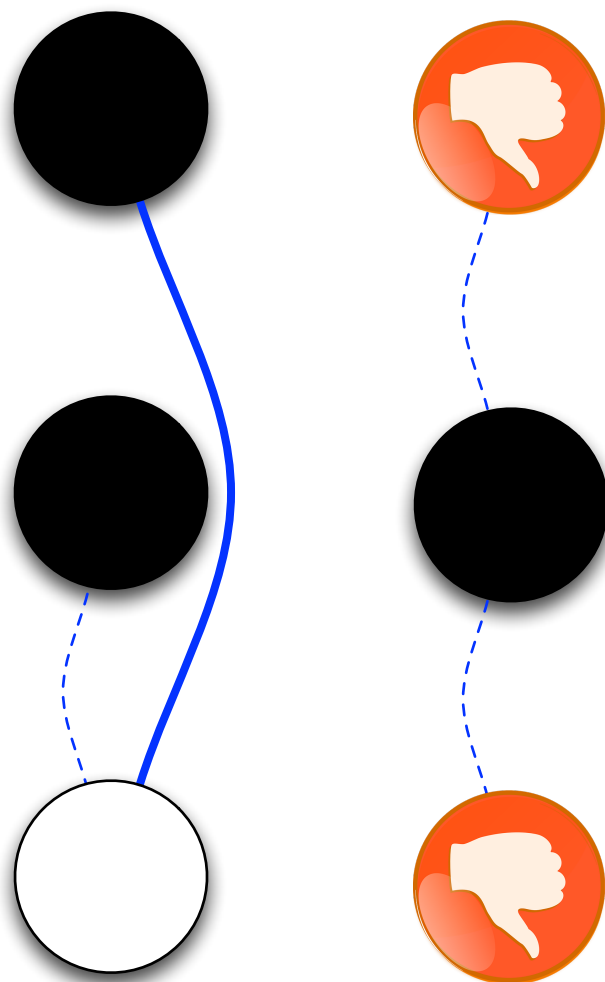
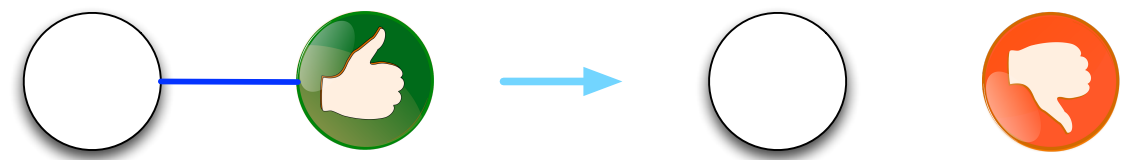
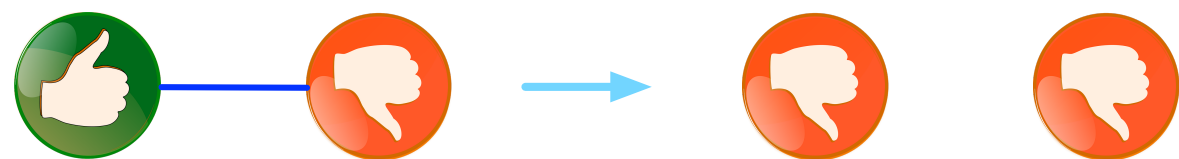
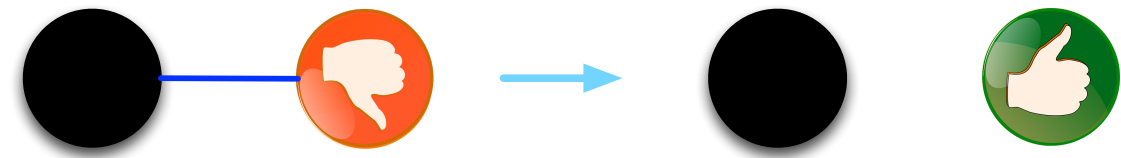
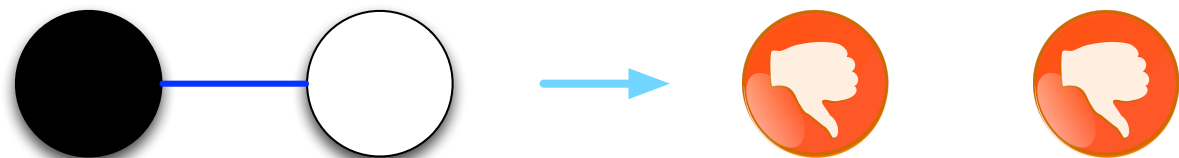


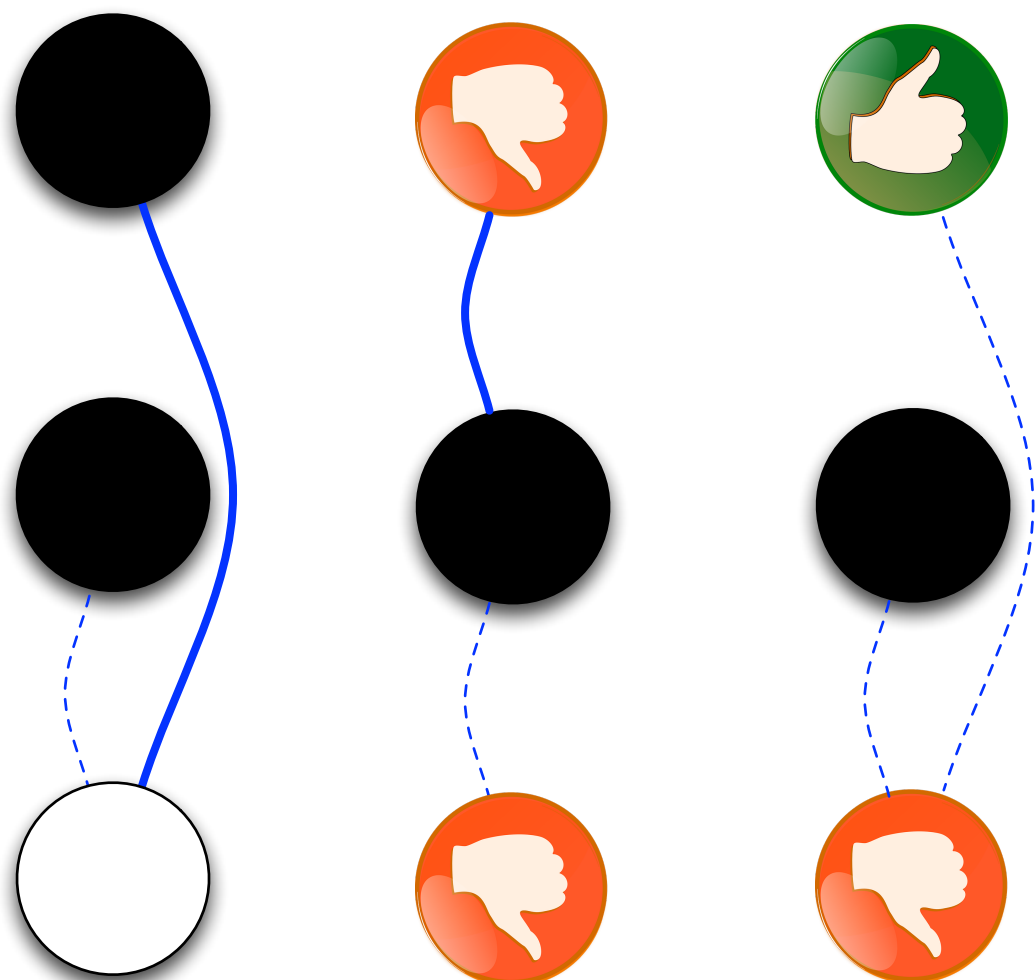
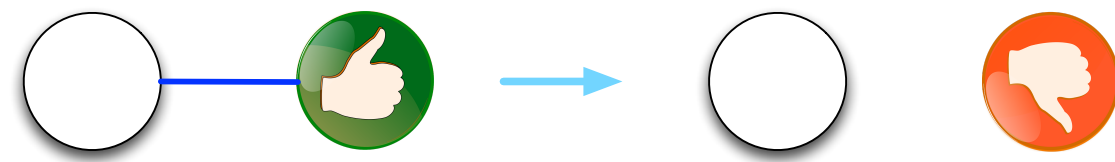
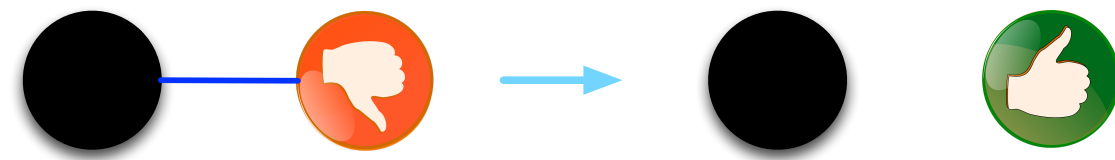
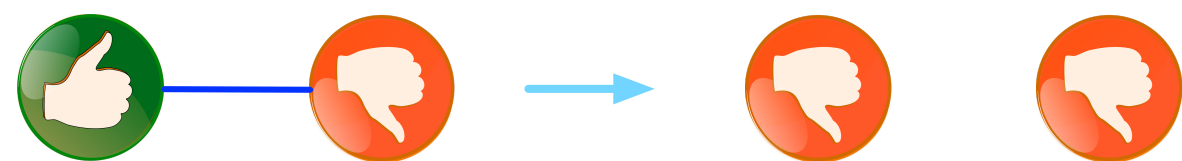
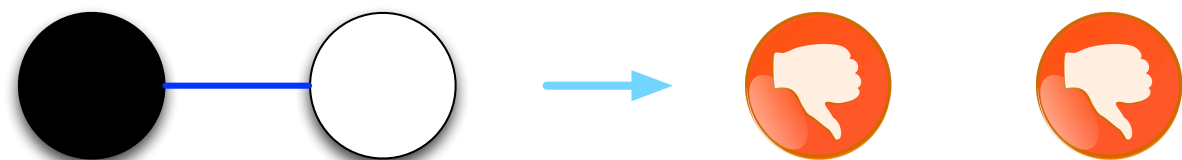


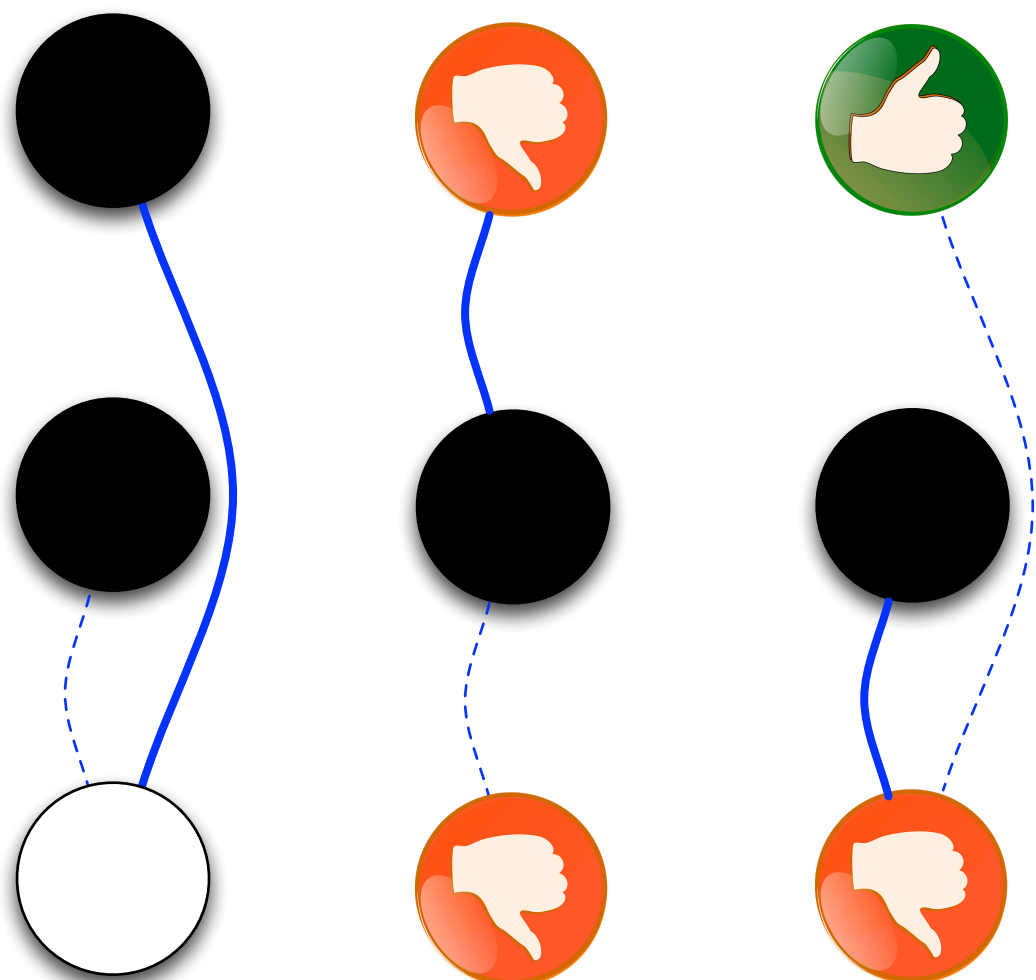
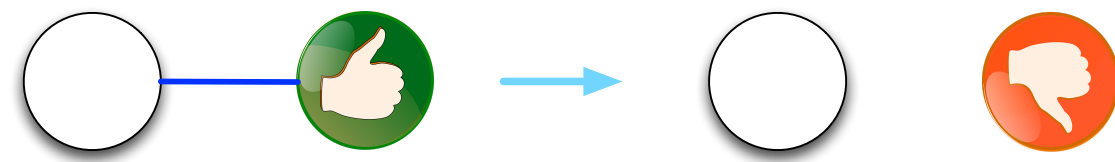
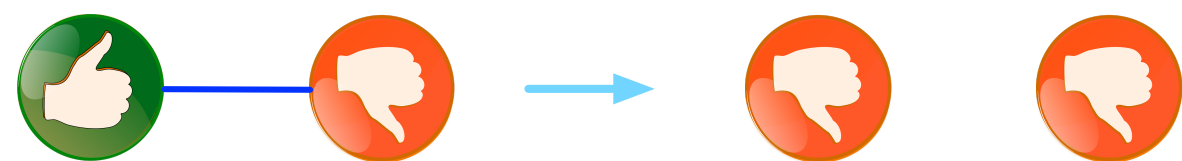
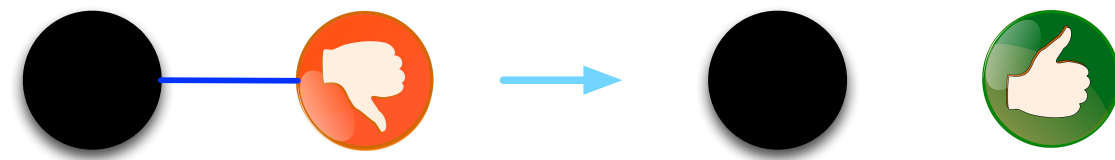
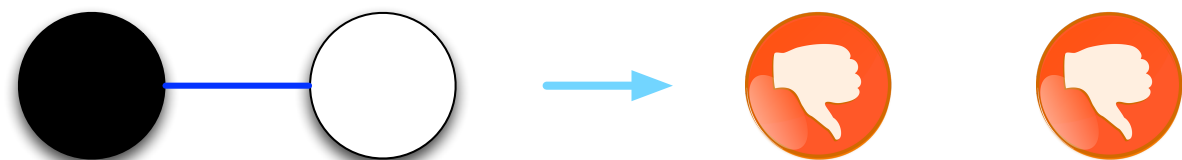


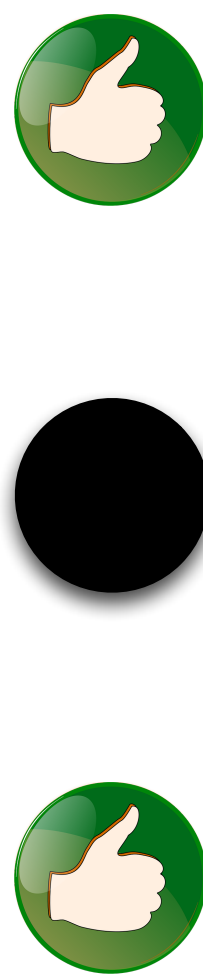
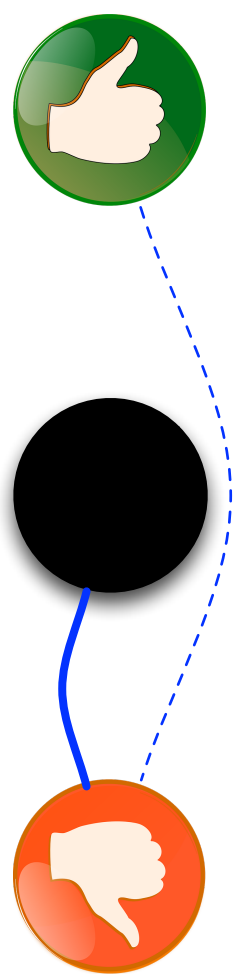
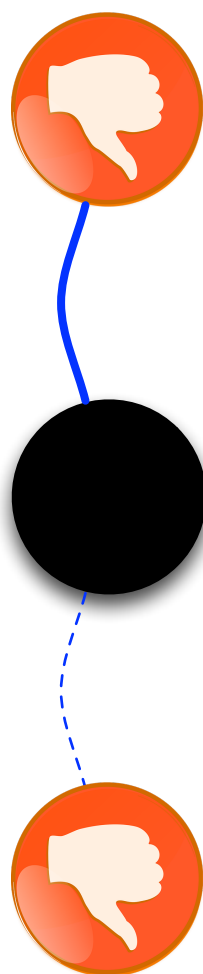
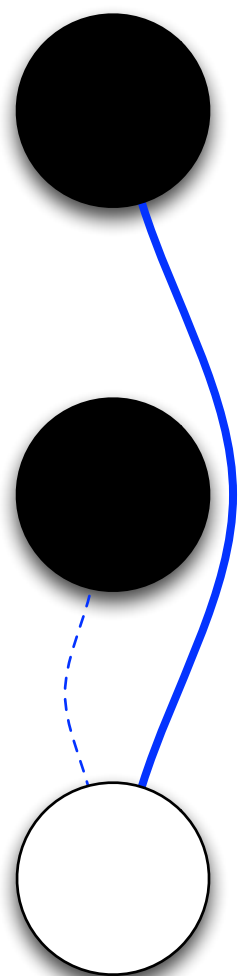
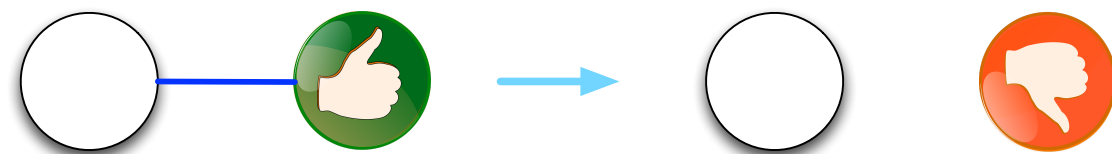
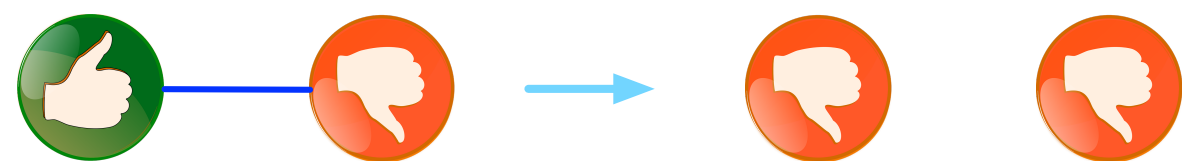
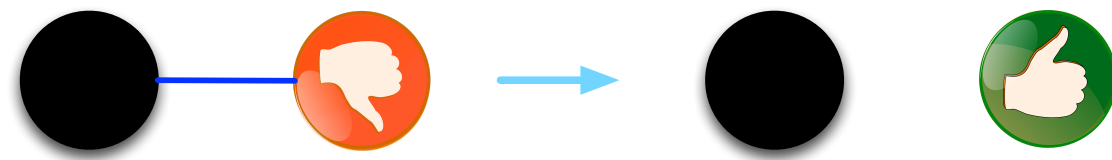
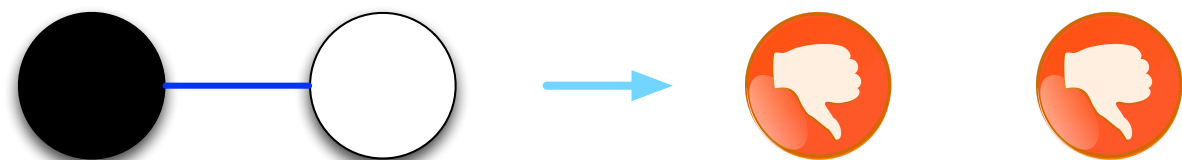


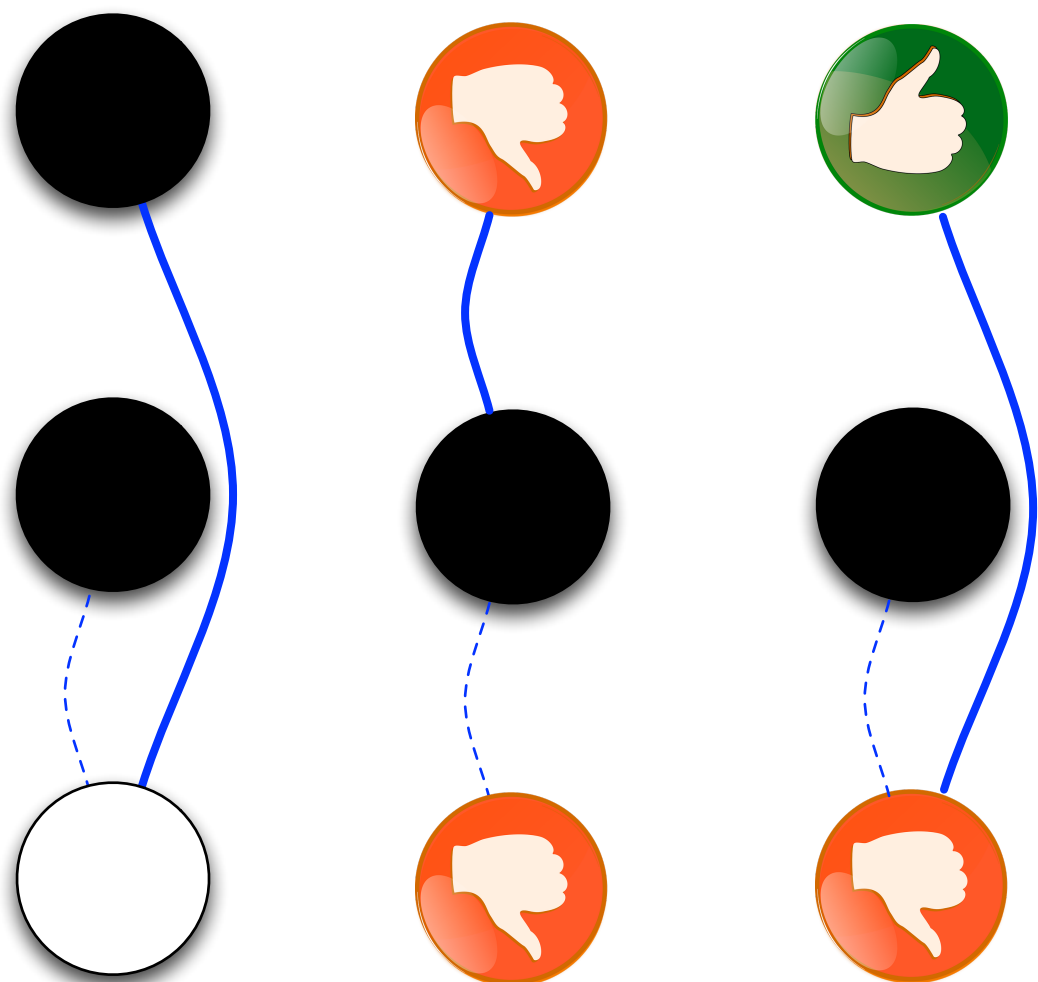
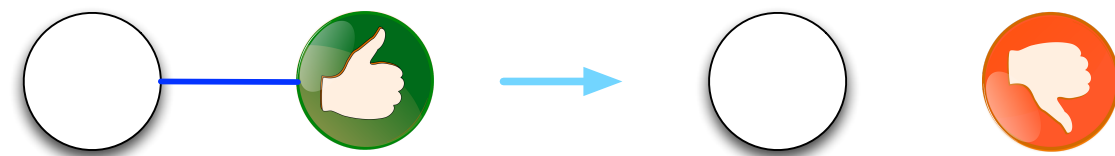
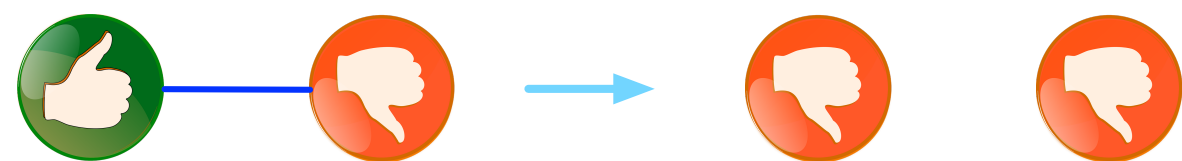
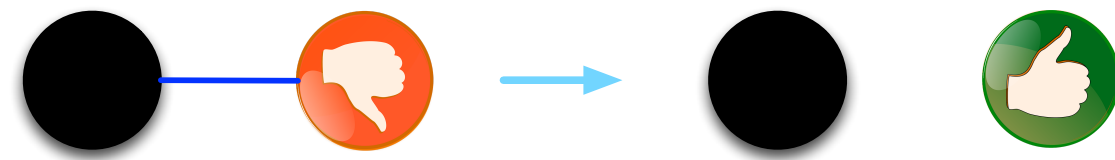
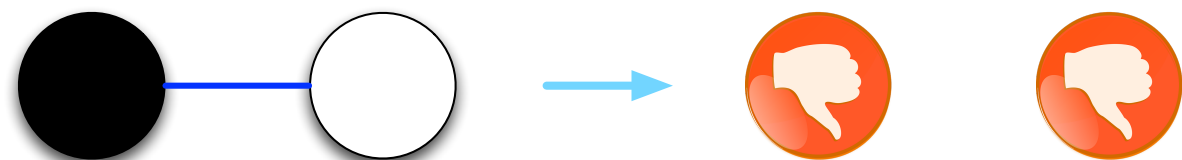


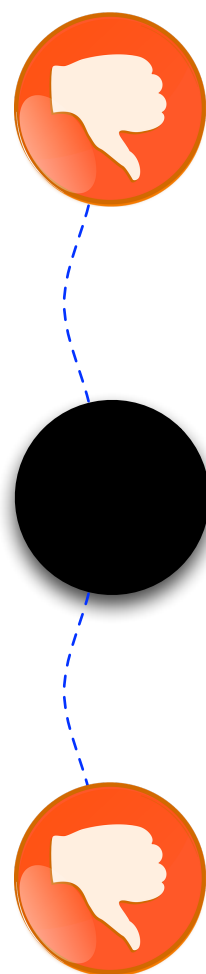
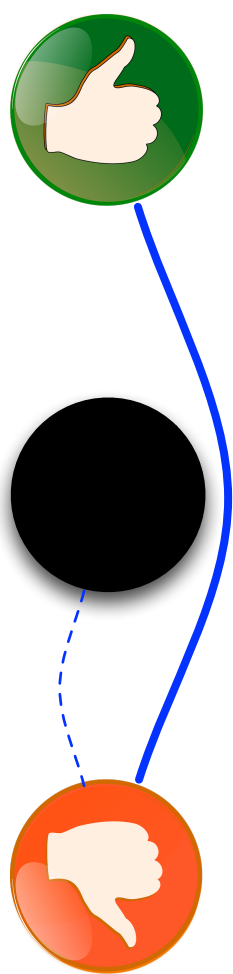
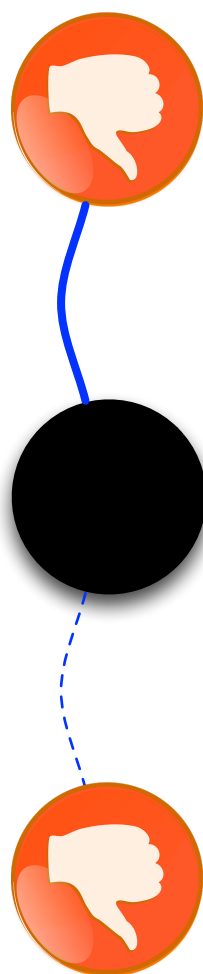
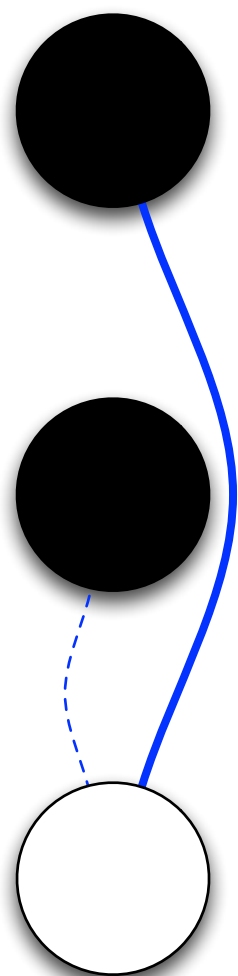
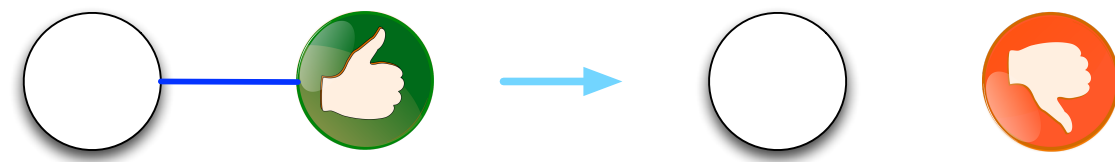
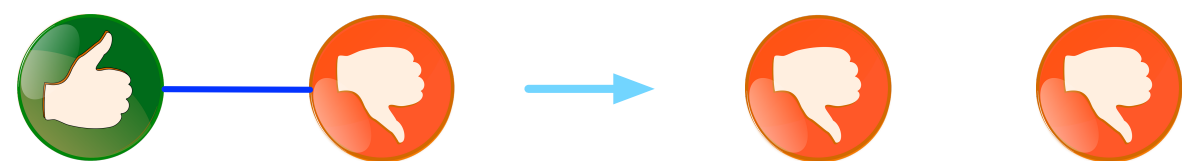
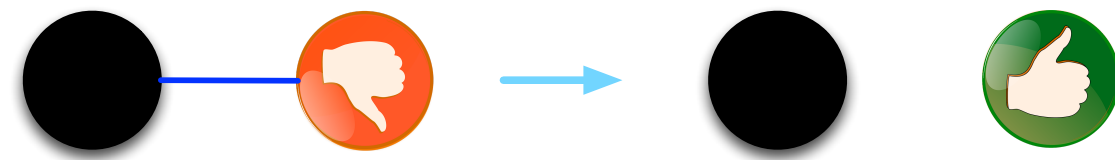
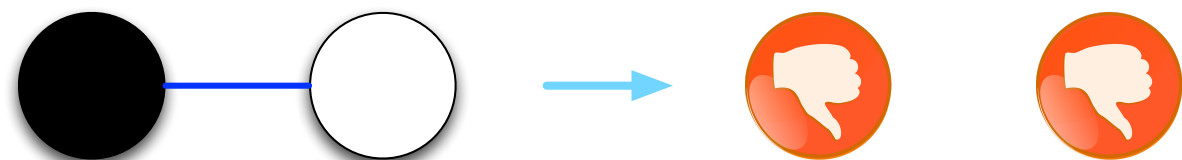


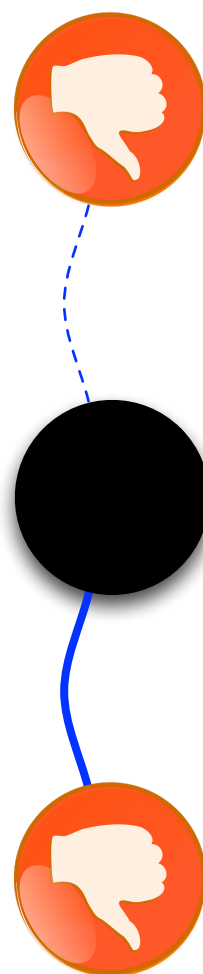
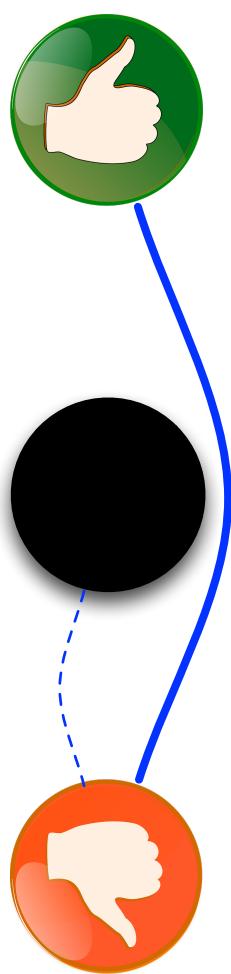
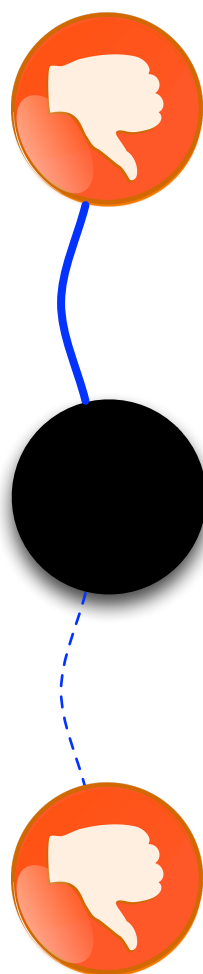
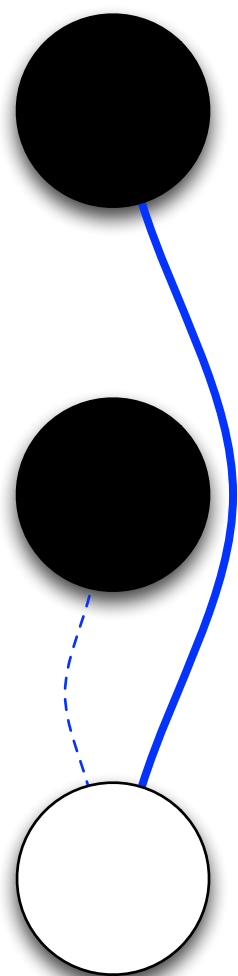
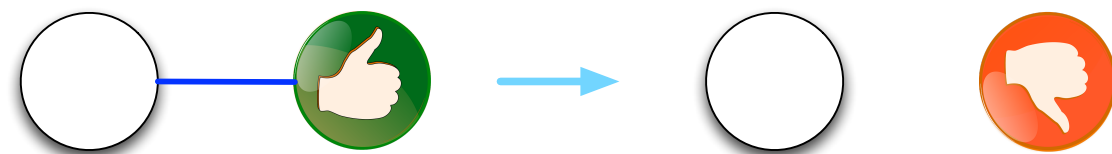
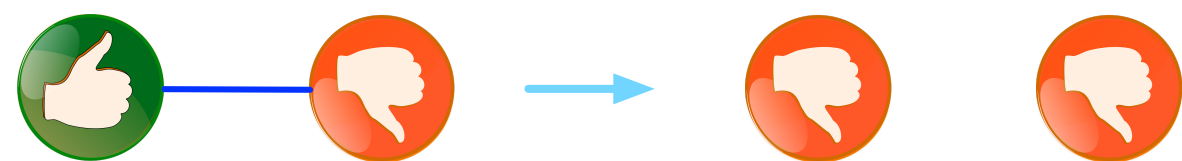
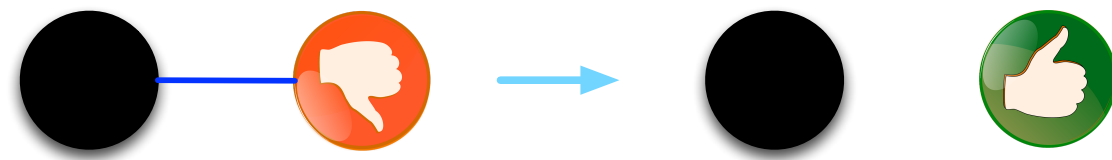
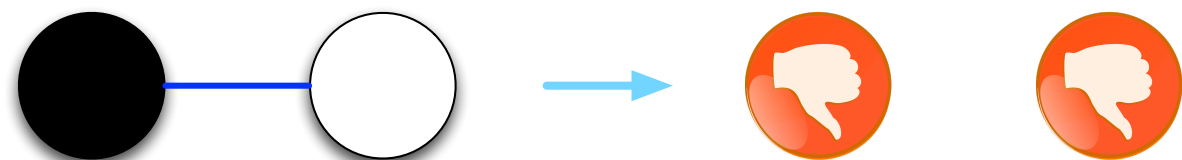


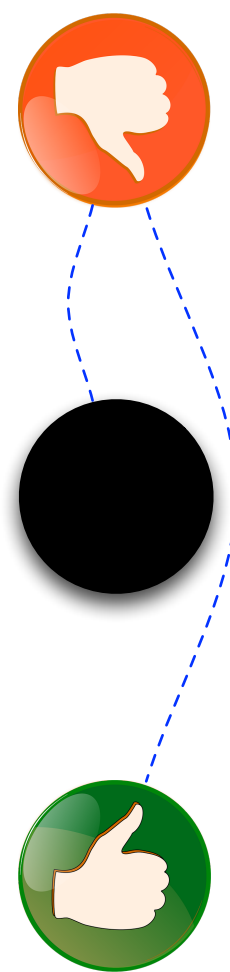
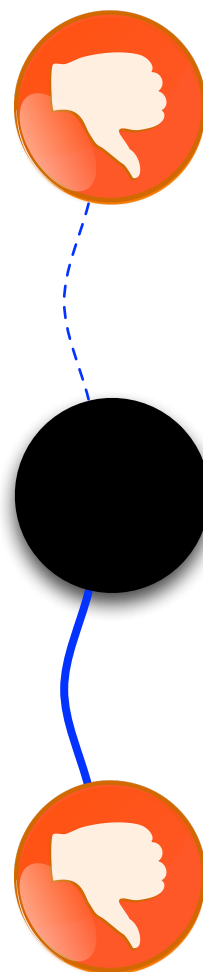
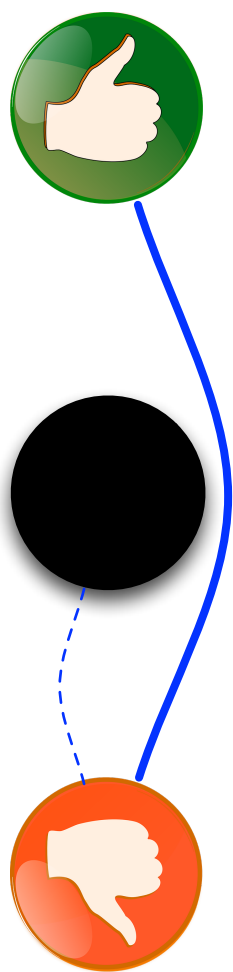
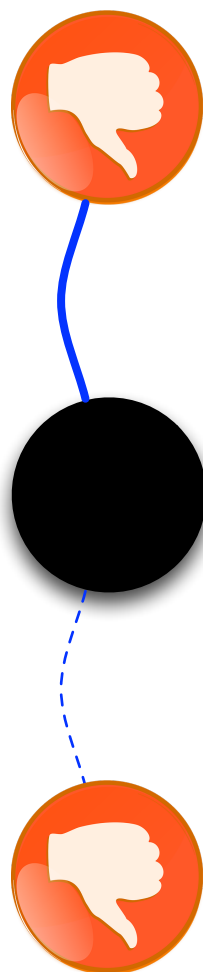
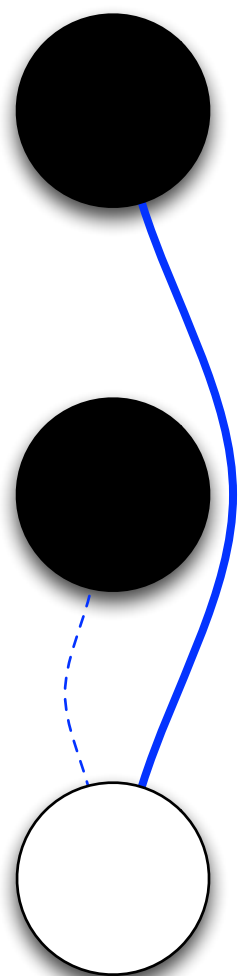
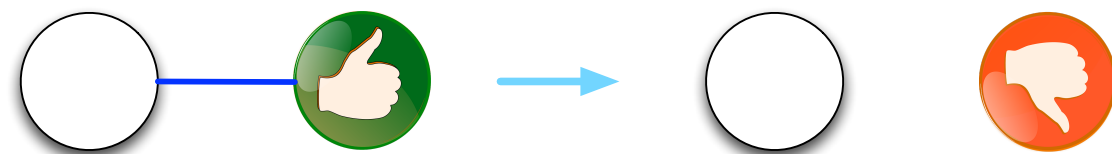
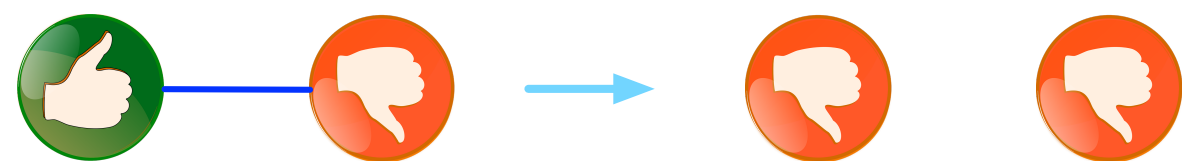
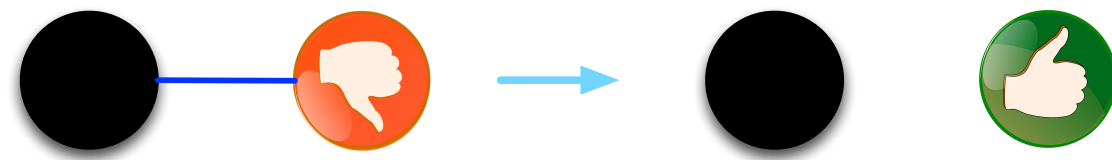
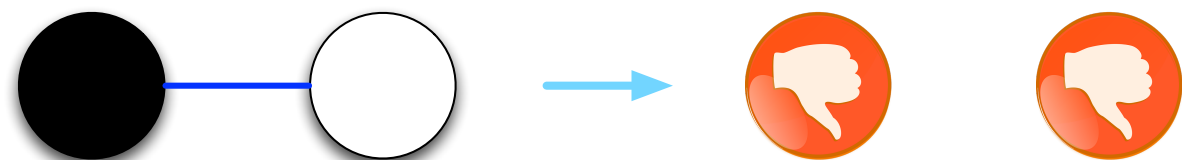


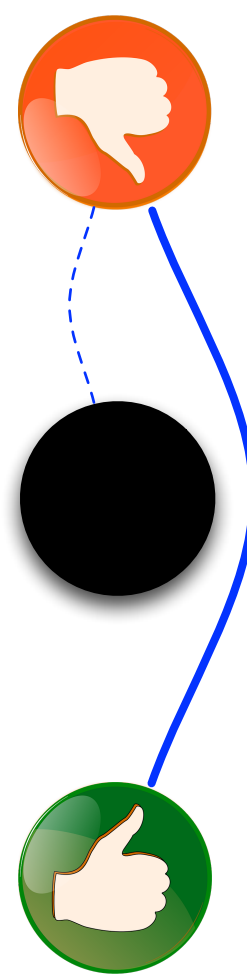
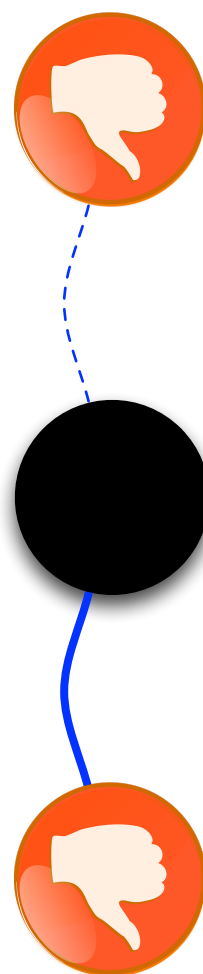
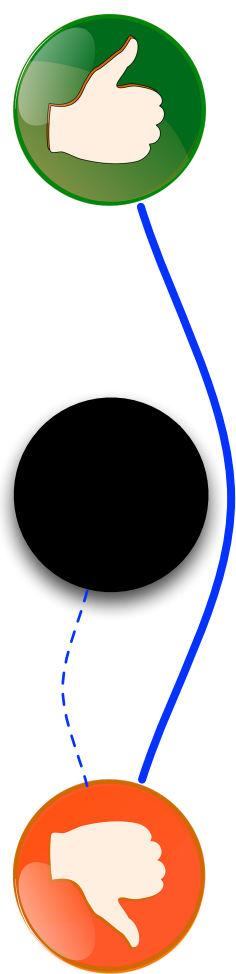
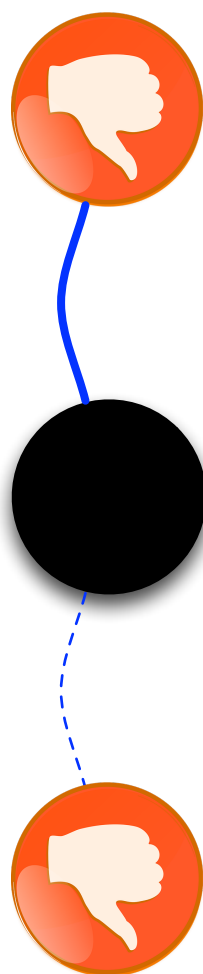
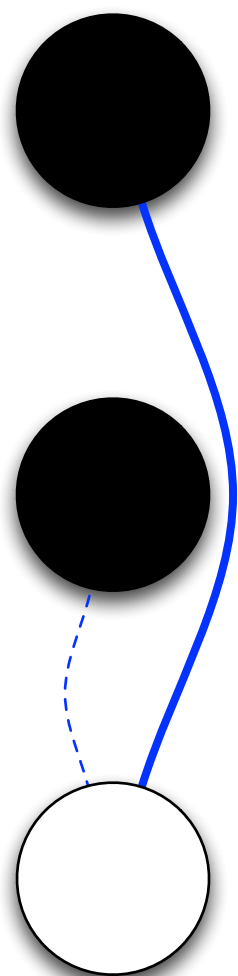
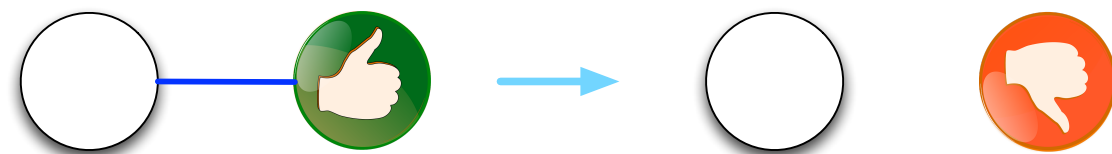
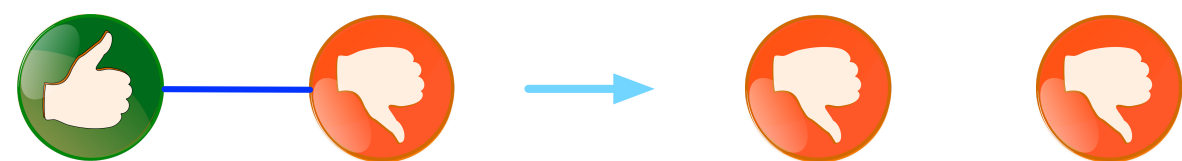
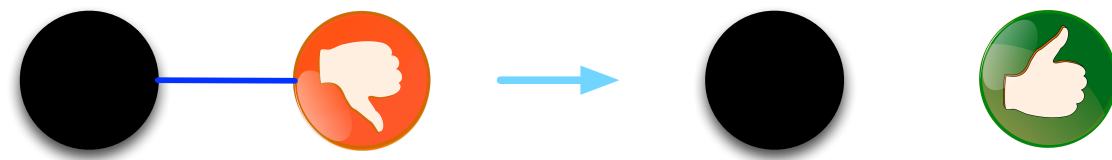
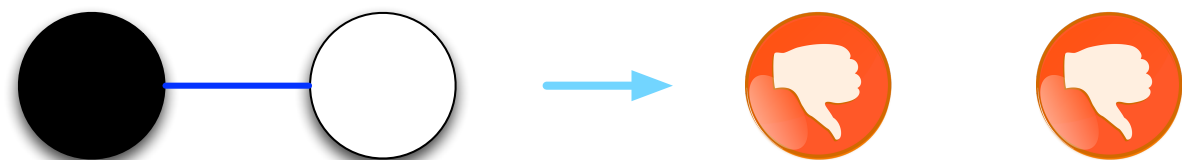


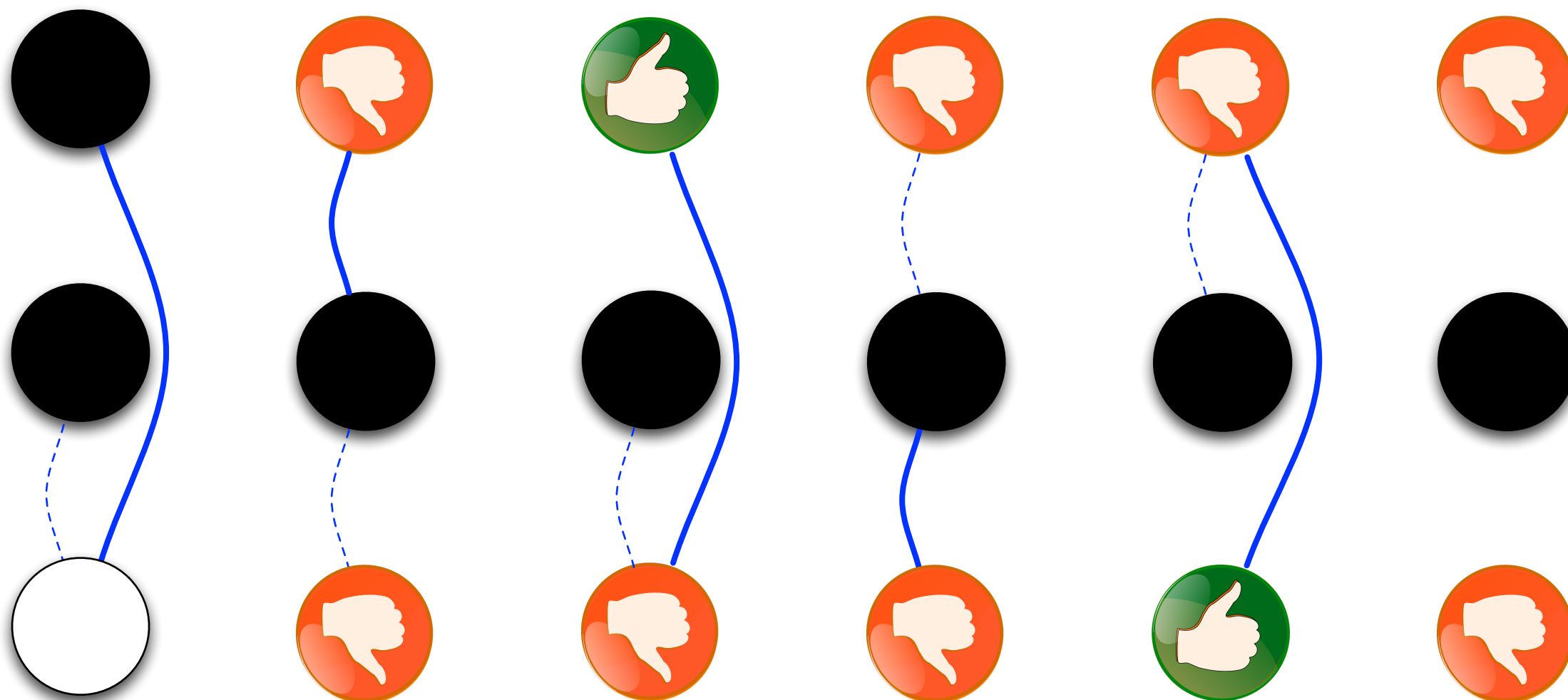
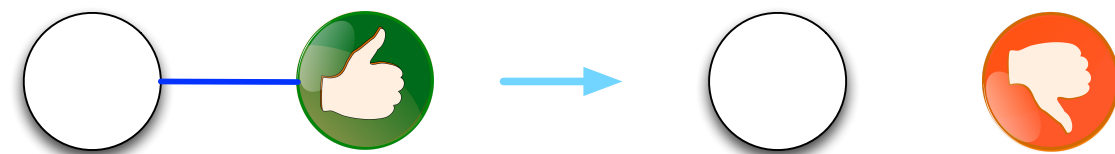
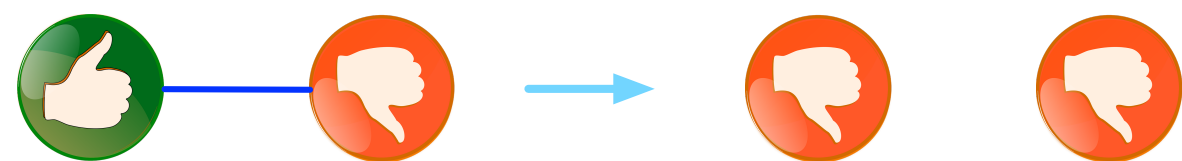
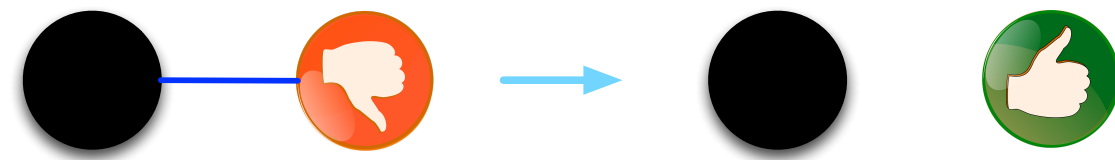
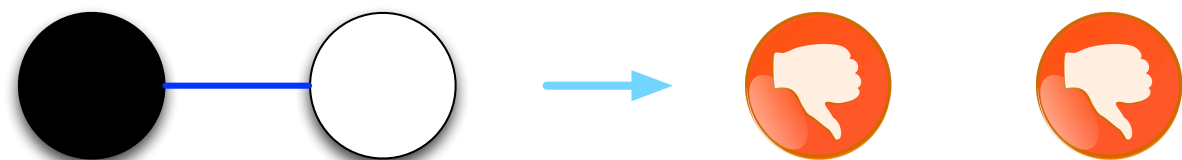




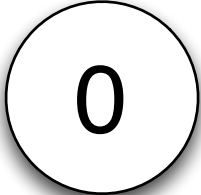



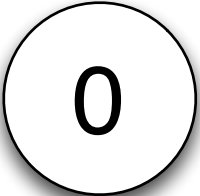



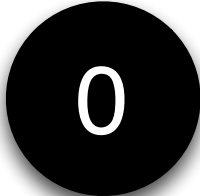







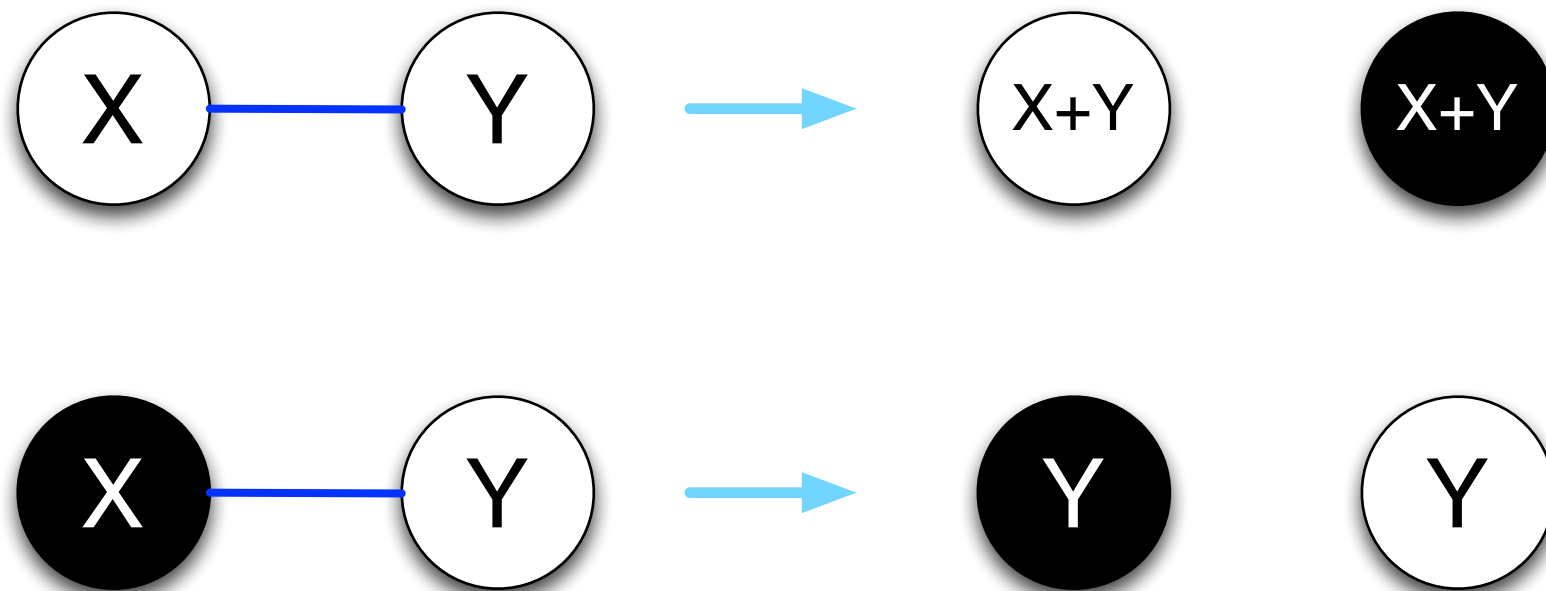


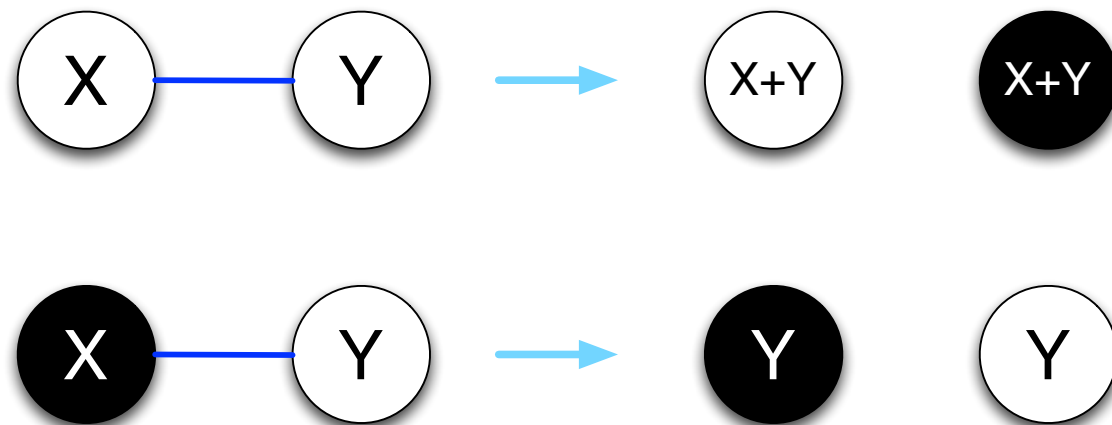


Example 3

- **Inputs:**    
- **Outputs:**        
- **Sum mod 4 ?**

Example 3





0

1

2

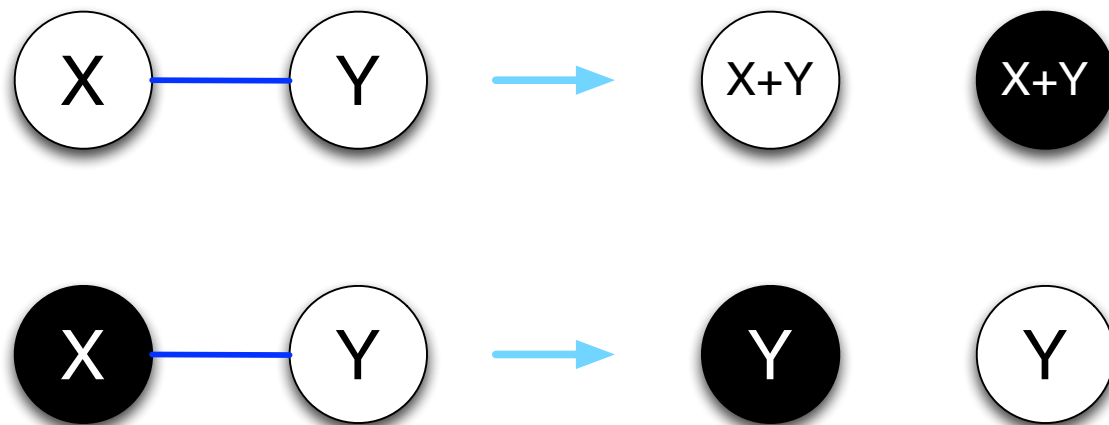
3

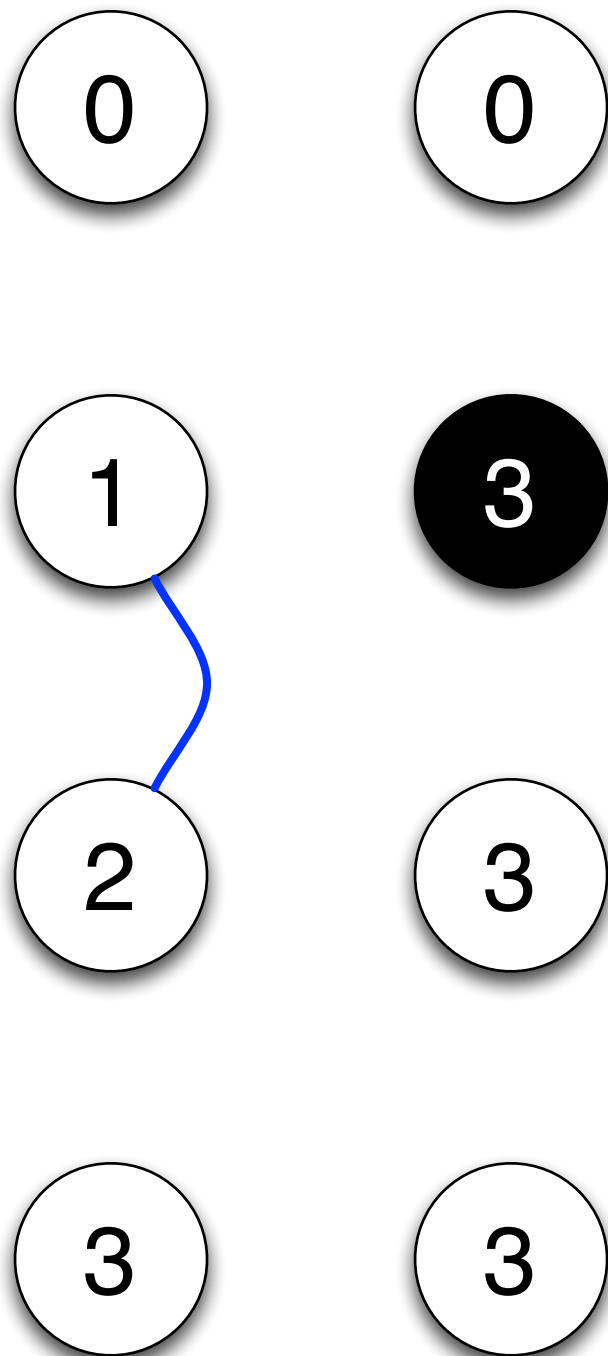
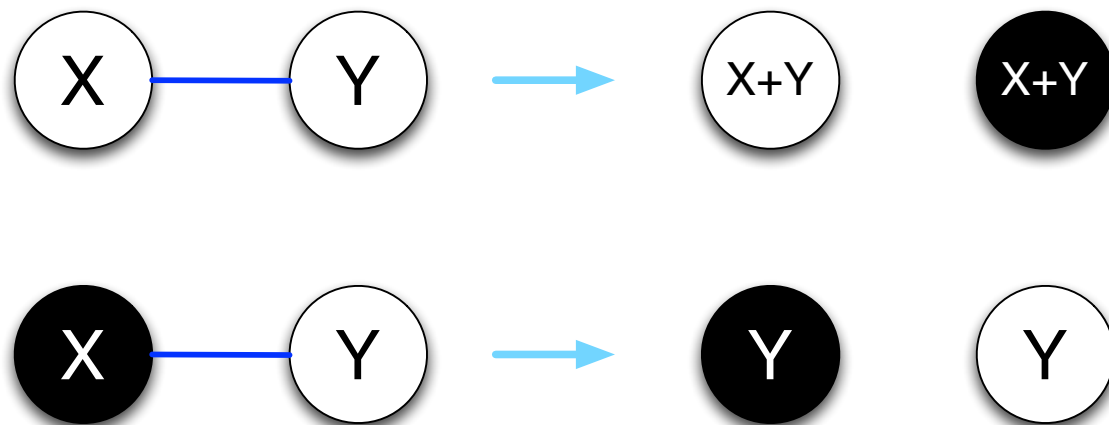
0

1

2

3





0

1

2

3

0

3

3

3

X

Y



X+Y

X+Y

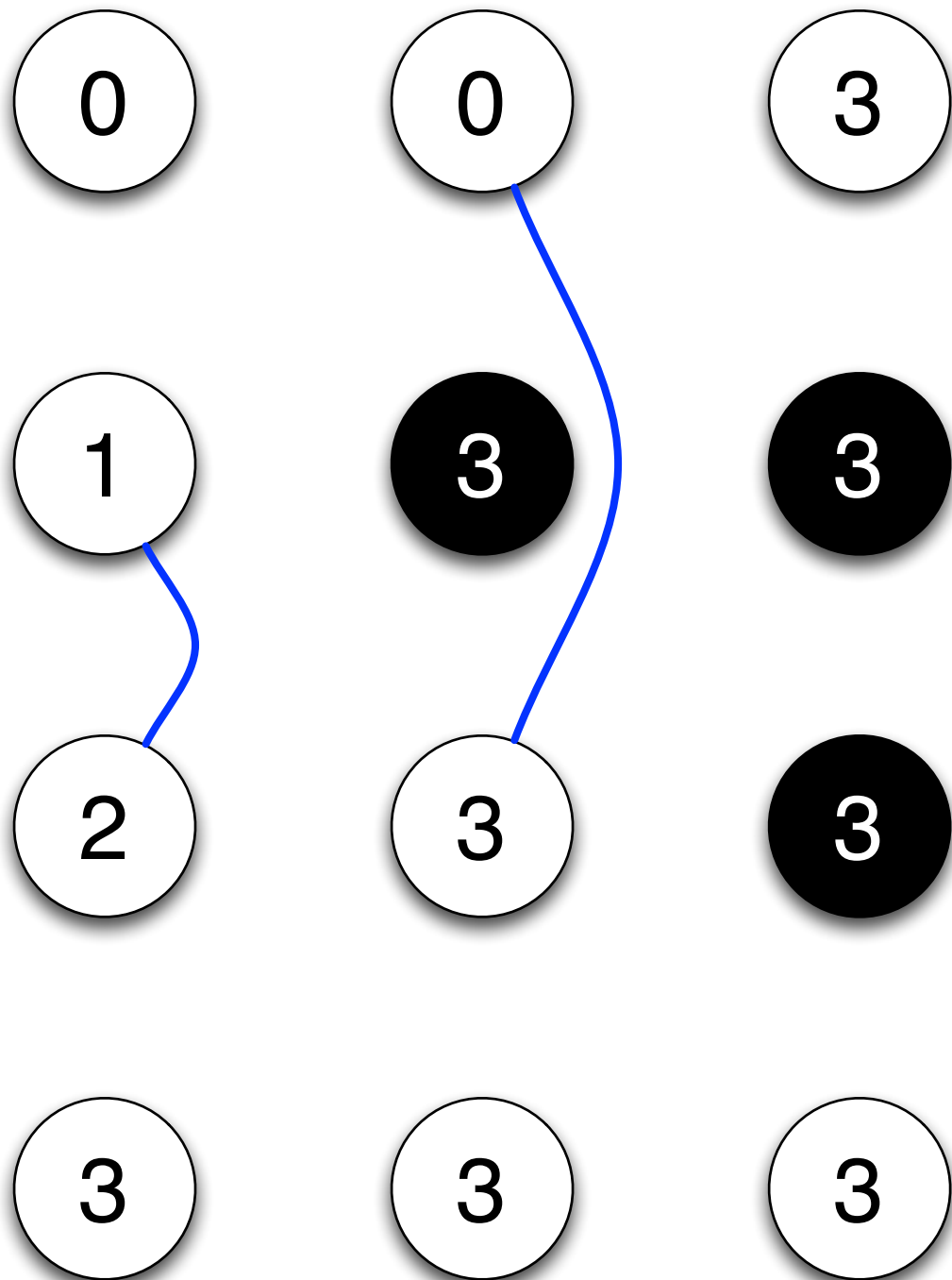
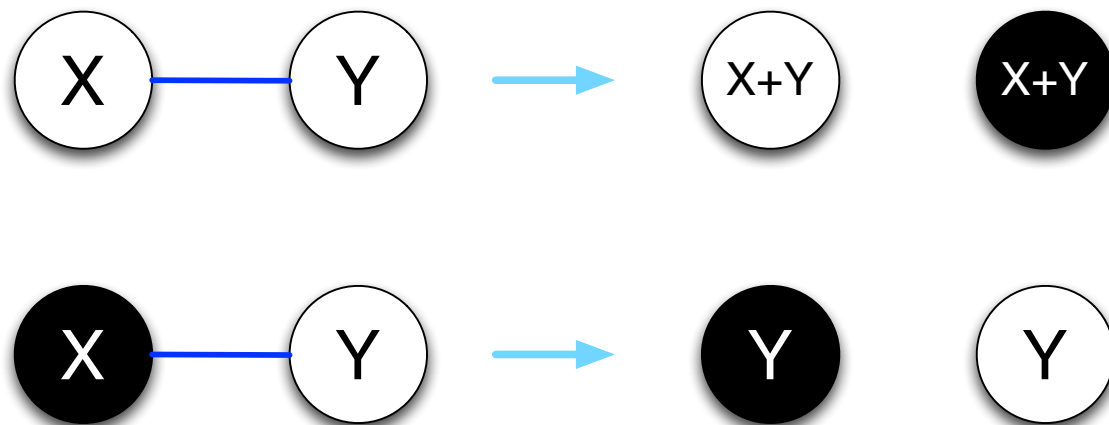
X

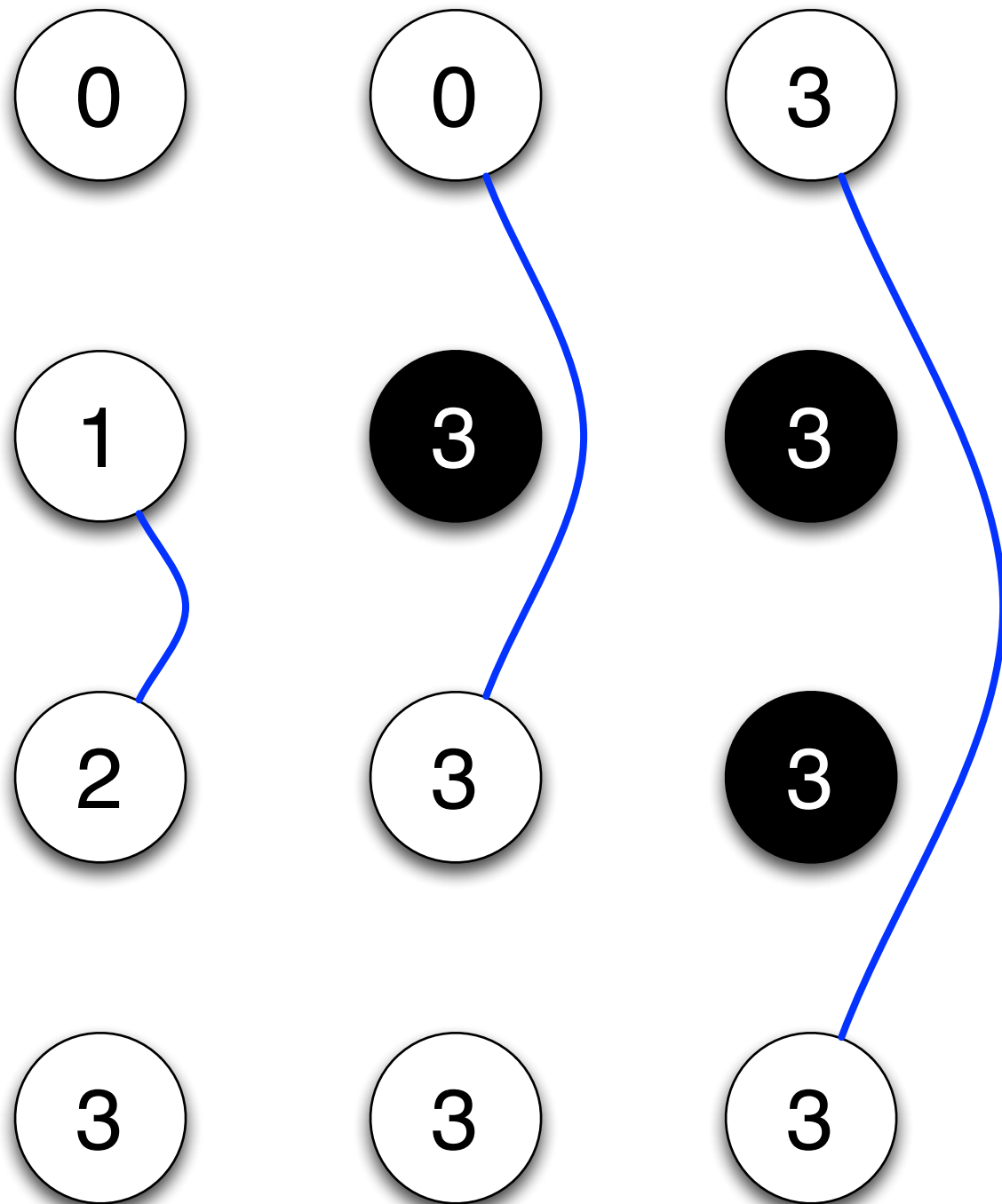
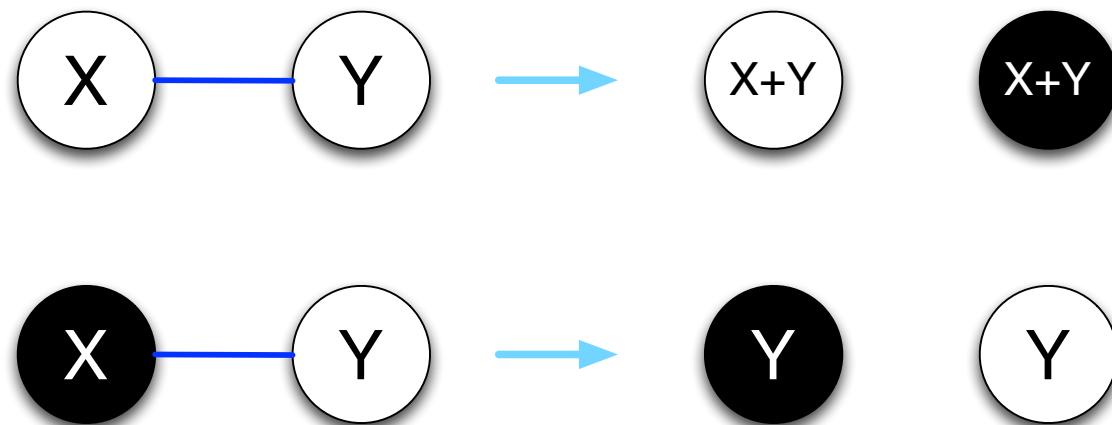
Y

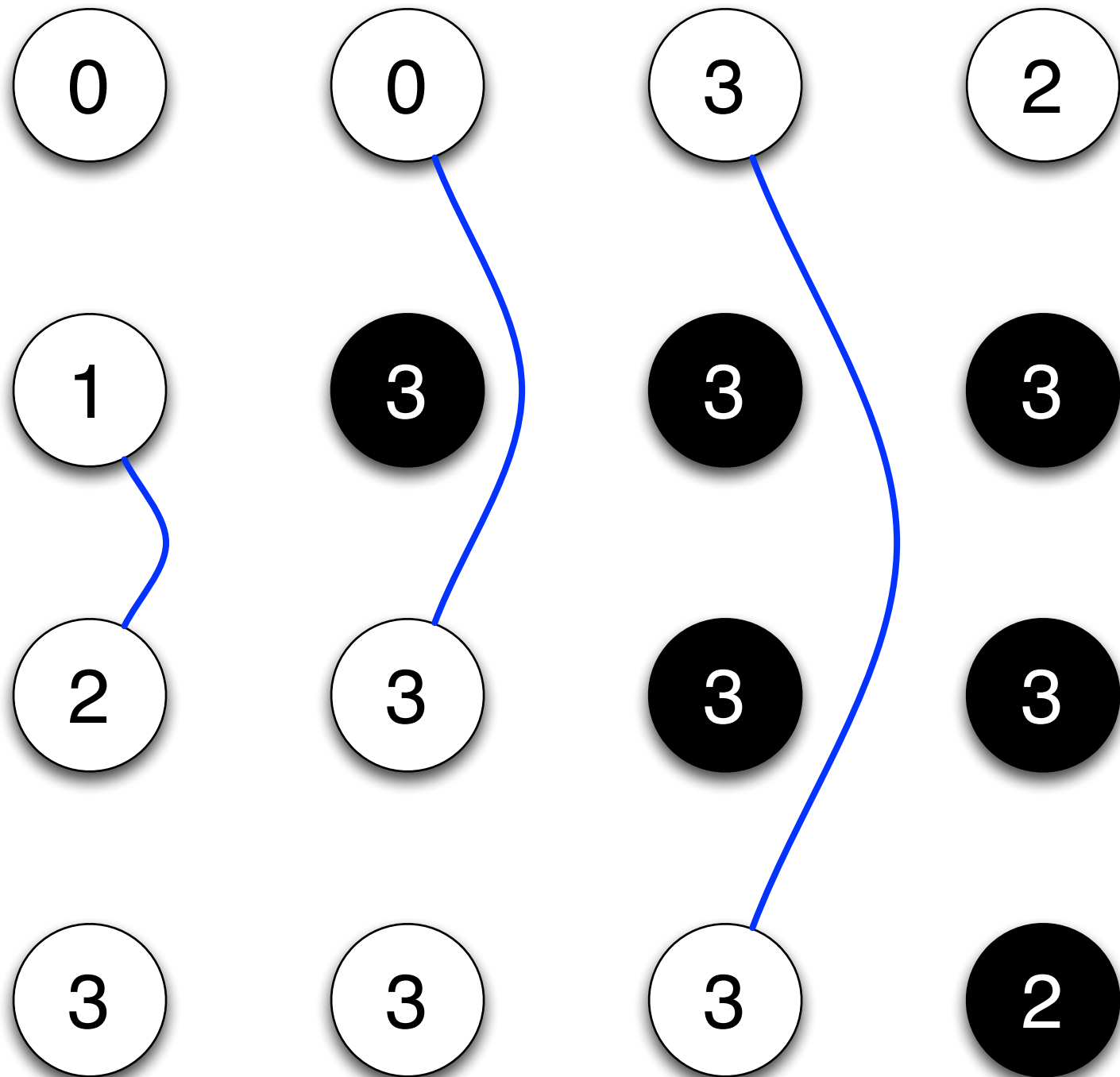
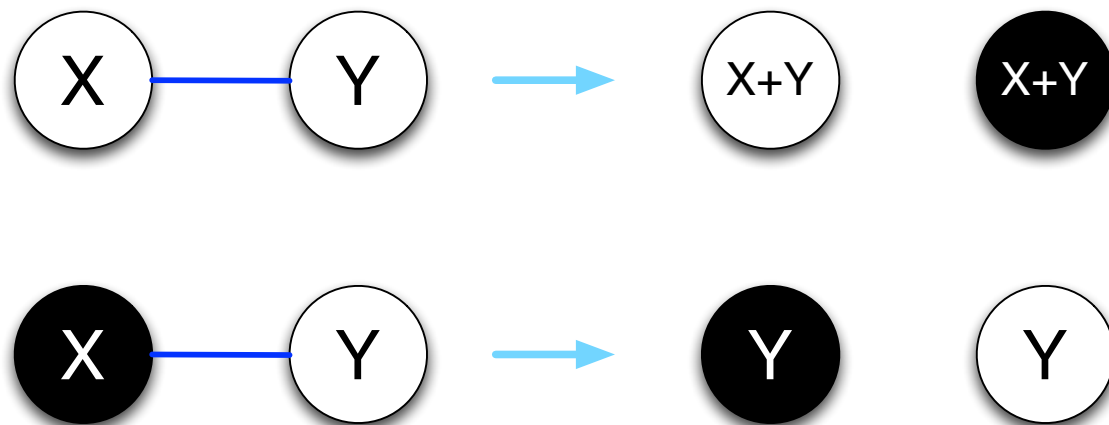


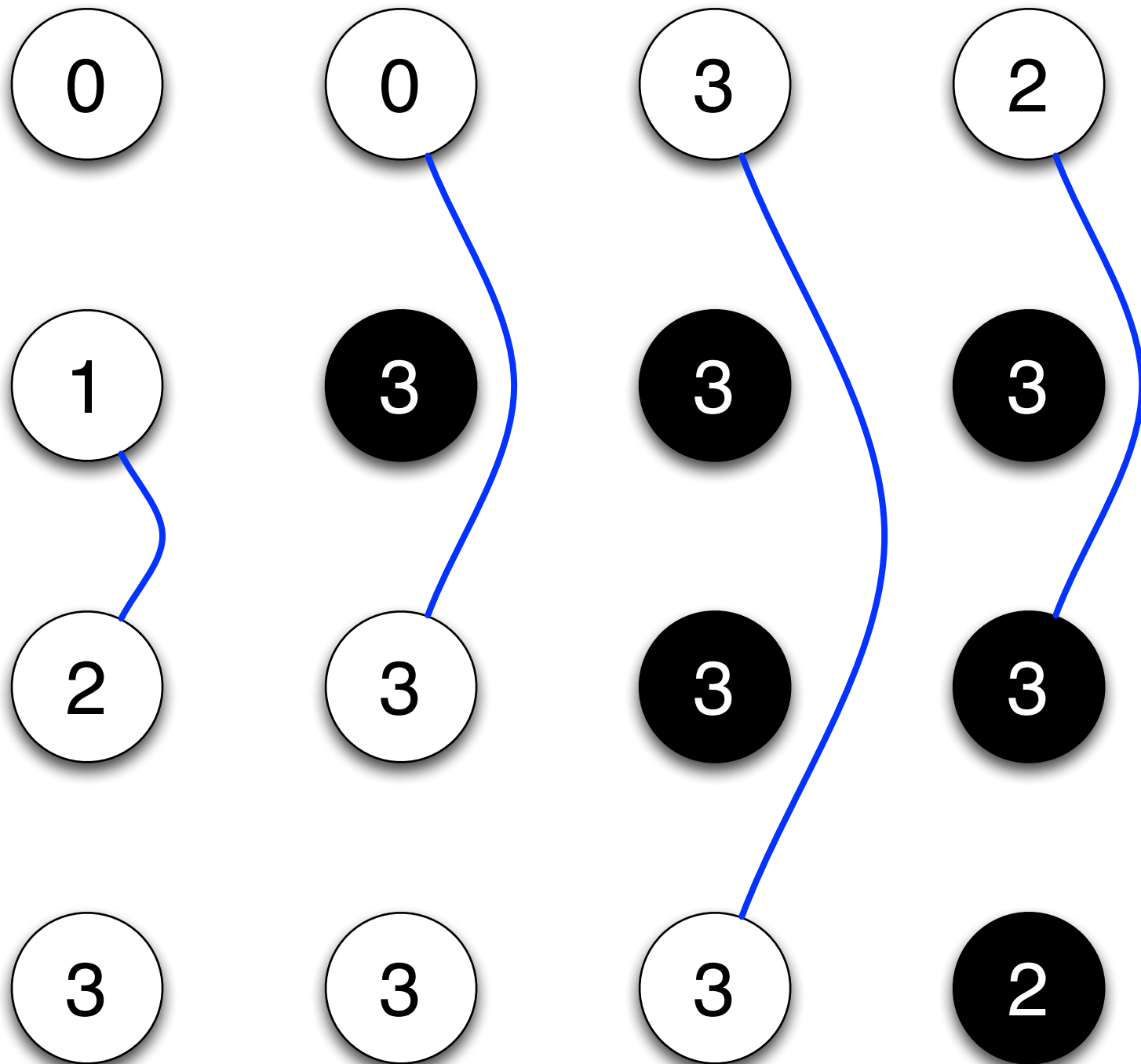
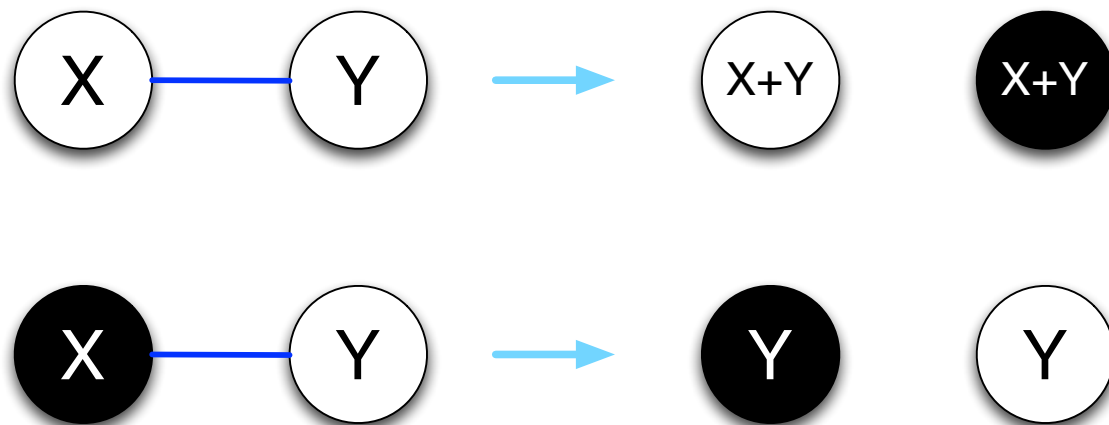
Y

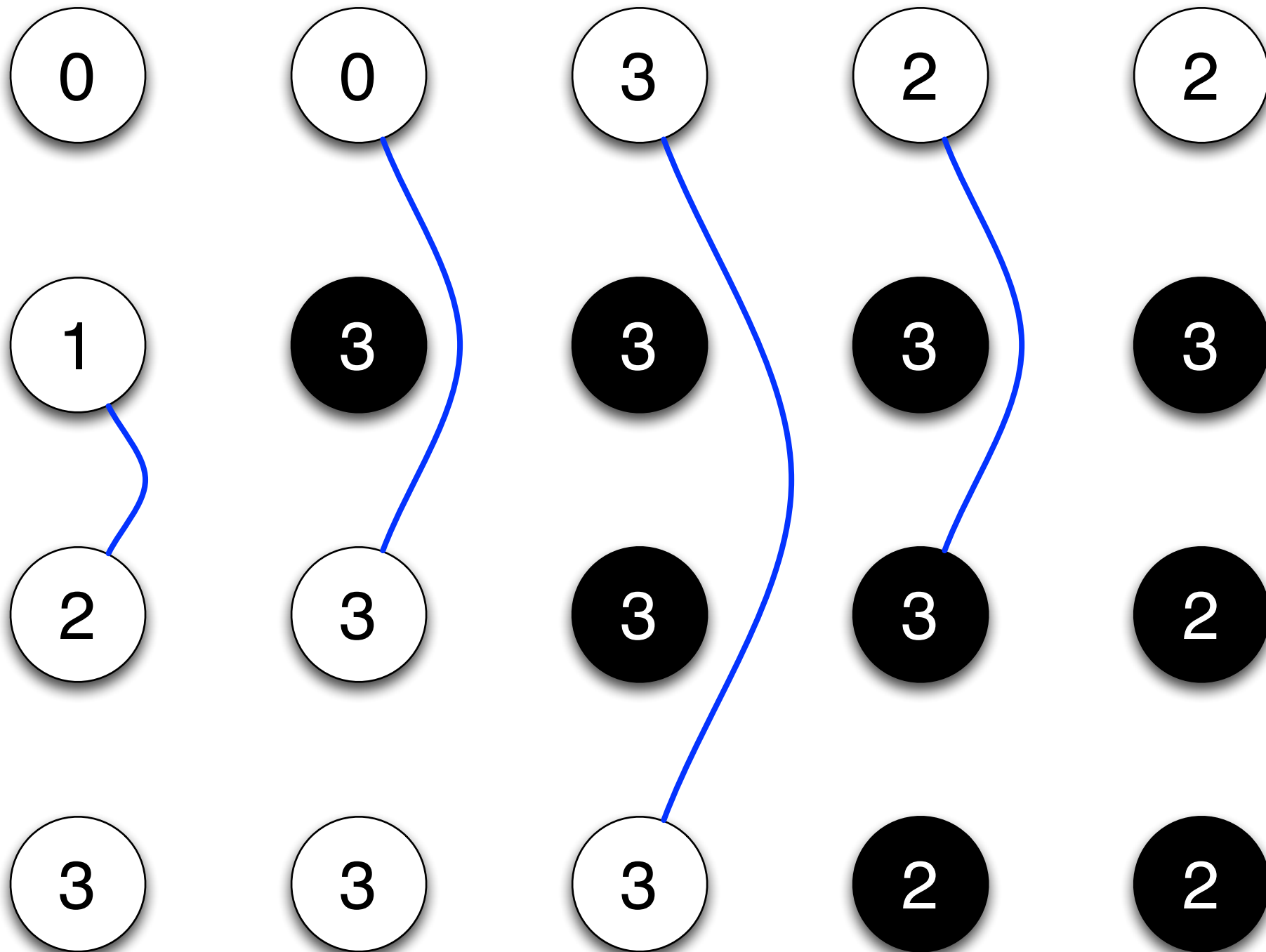
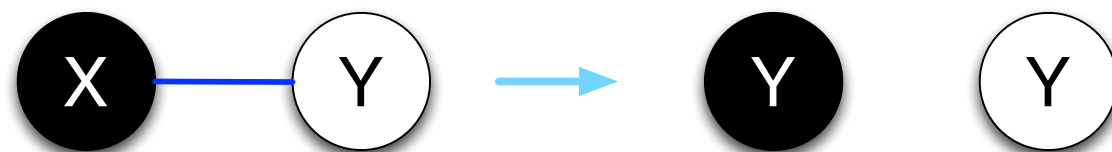
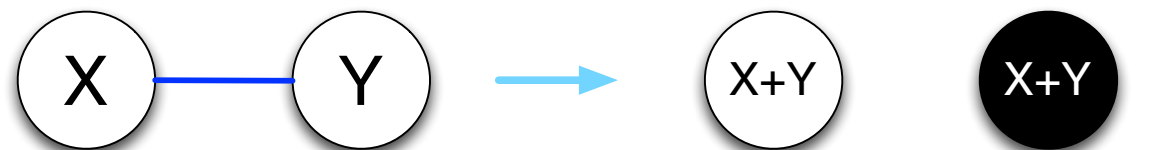
Y

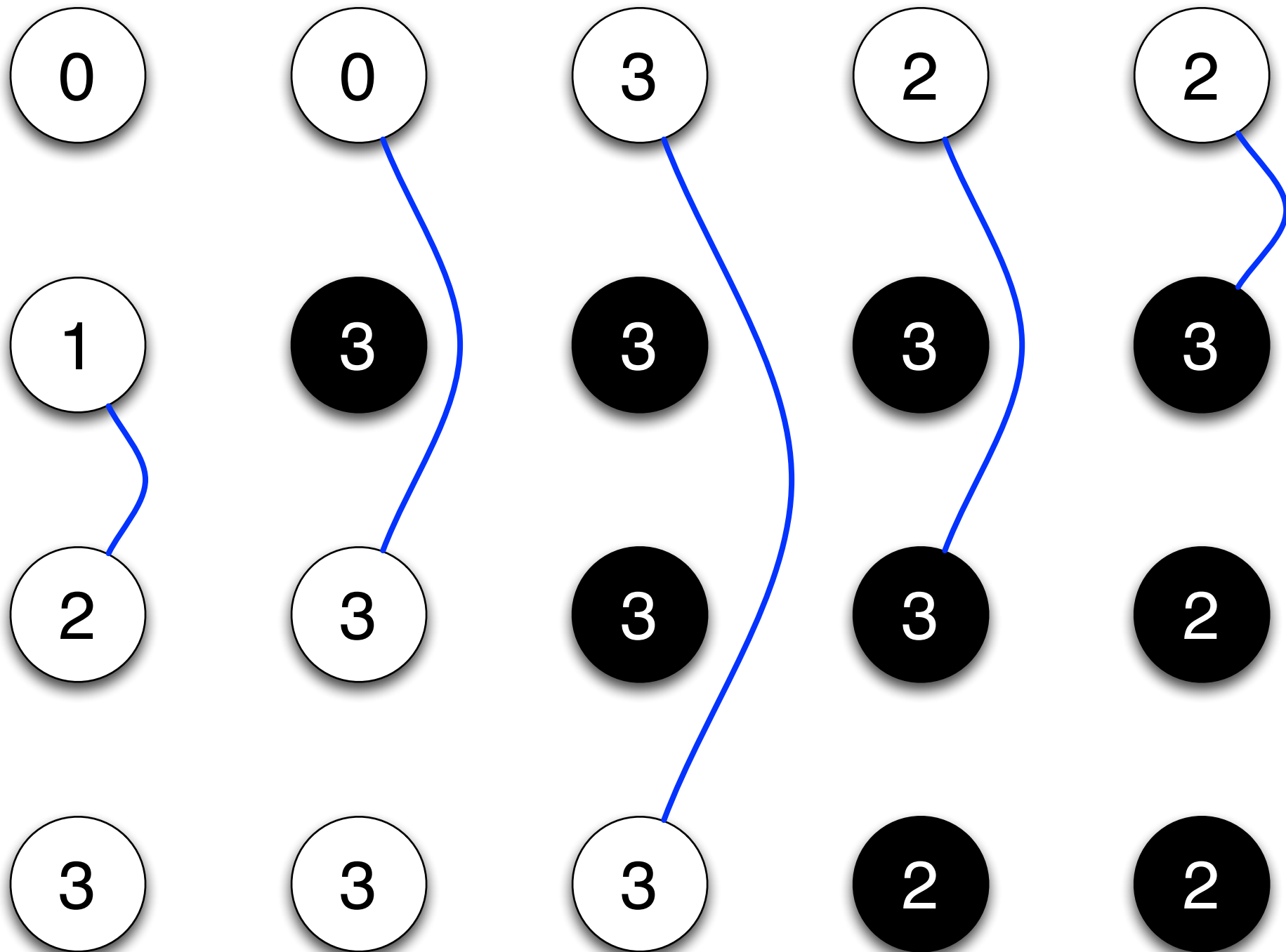
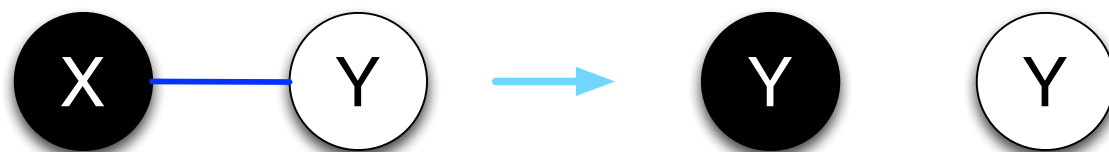
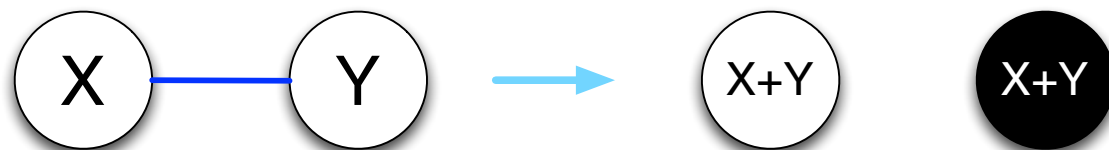


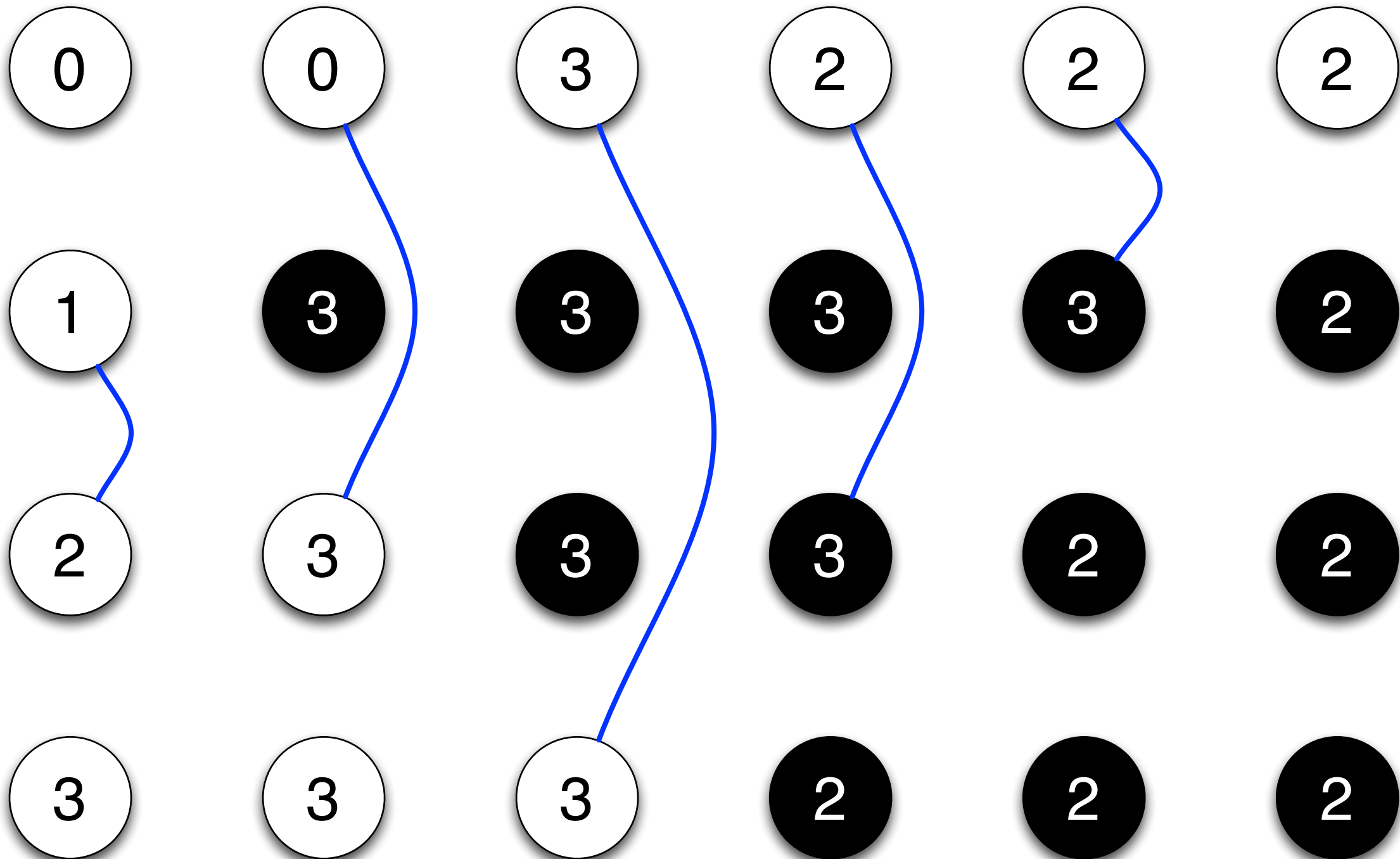
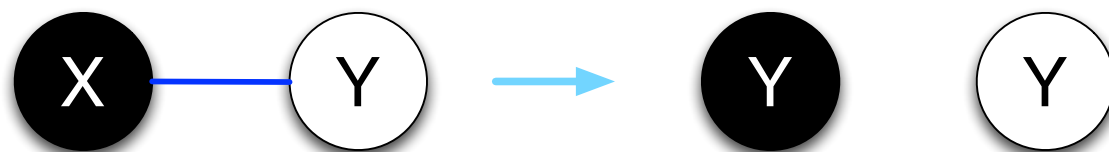
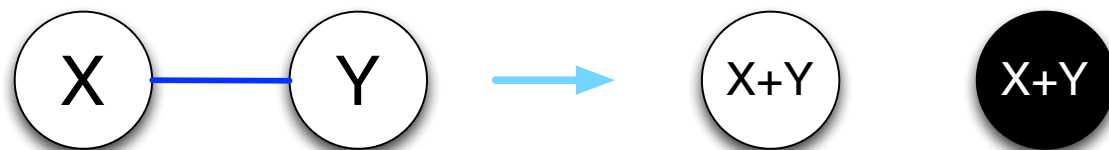




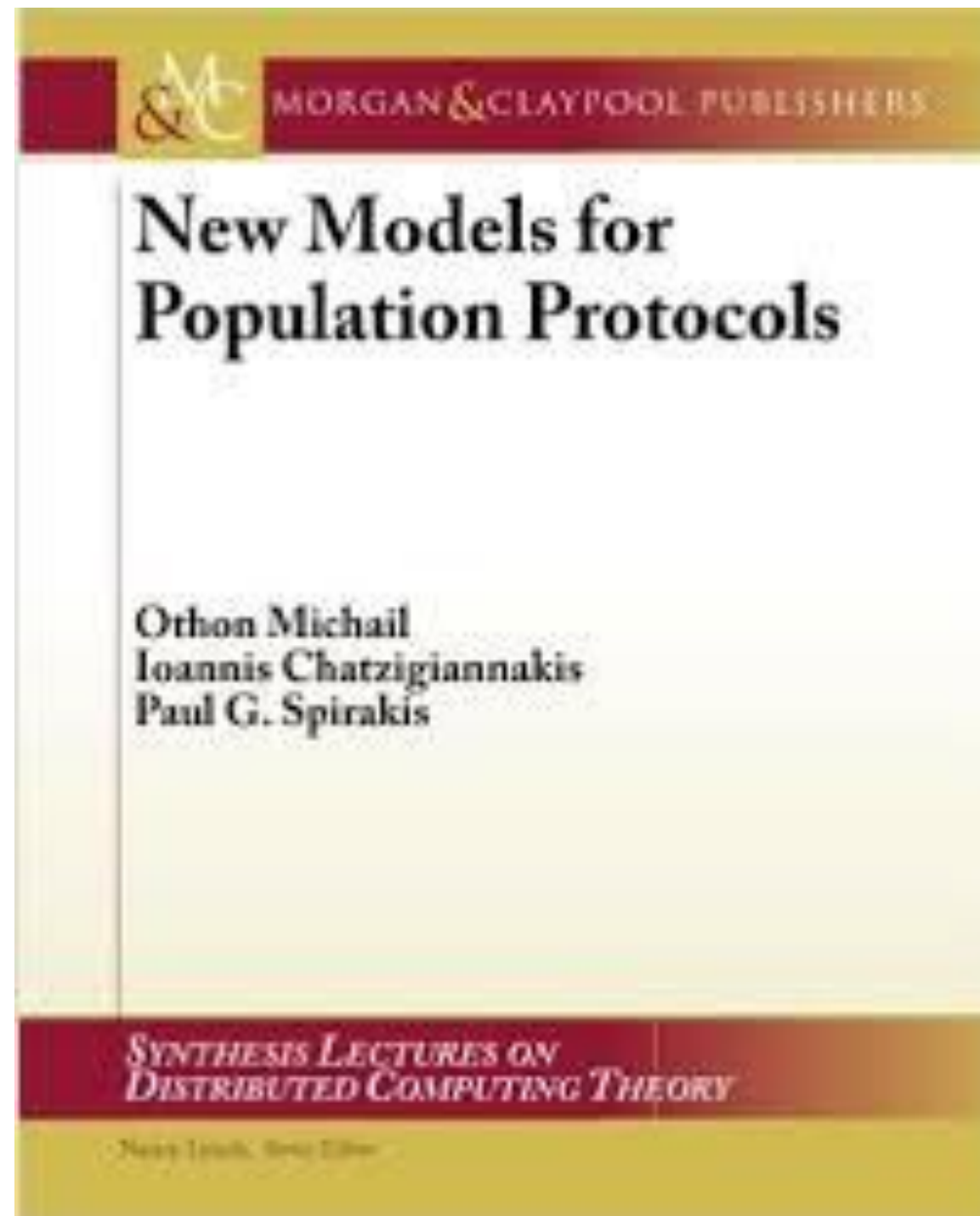








Population Protocols



Dynamic Graphs

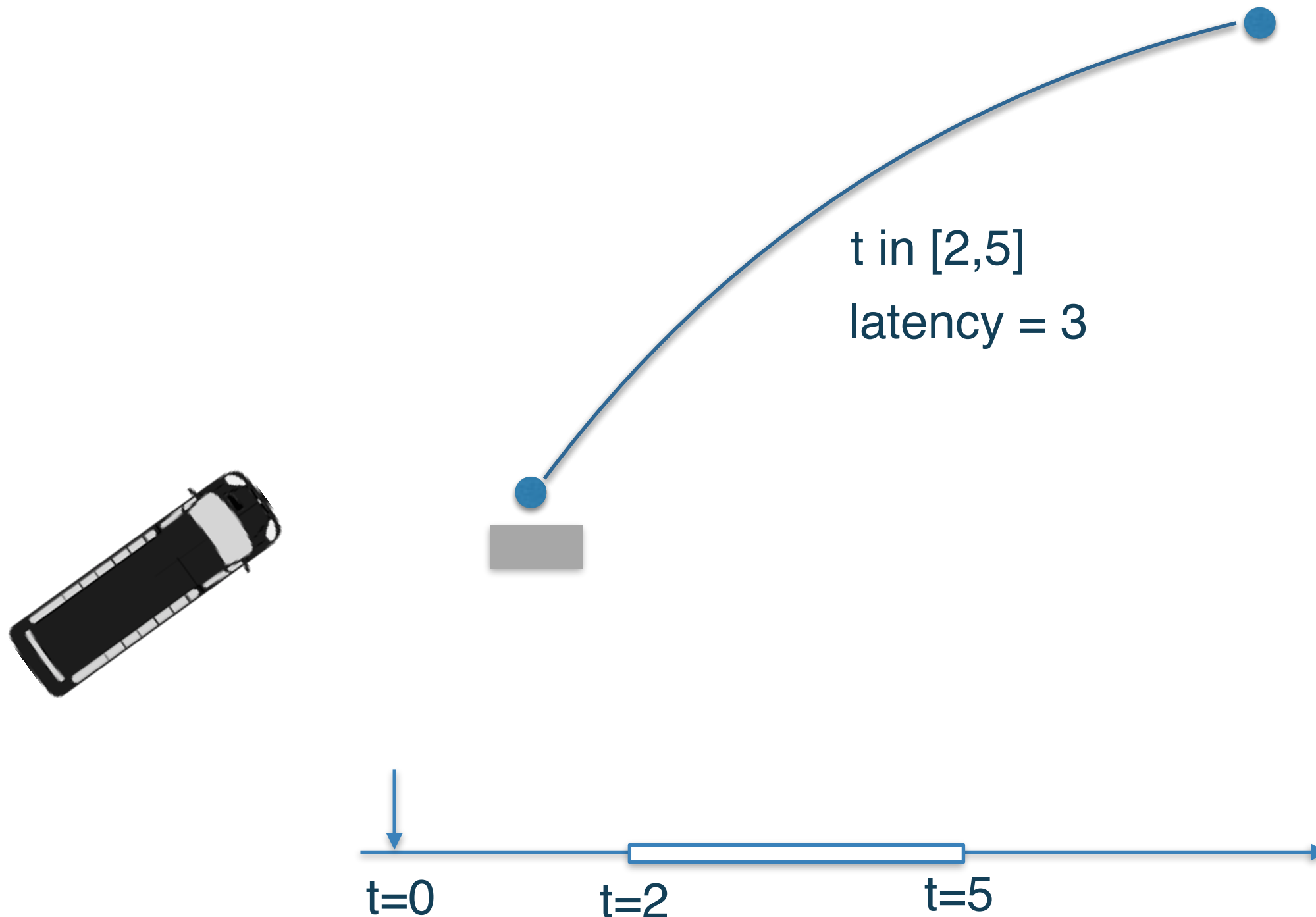
Time-varying Graphs

- A *time-varying graph* (TVG) is a 5-tuple $(\mathbf{V}, \mathbf{E}, \mathbf{T}, \mathbf{p}, \mathbf{l})$
 - \mathbf{V} : set of nodes
 - \mathbf{E} : (labelled) set of edges
 - \mathbf{T} : lifetime, $\mathbf{T} \subseteq \mathcal{T}$
 - \mathbf{p} : presence function, $\mathbf{E} \times \mathbf{T} \longrightarrow \{0, 1\}$
 - \mathbf{l} : latency function, $\mathbf{E} \times \mathbf{T} \longrightarrow \mathcal{T}$

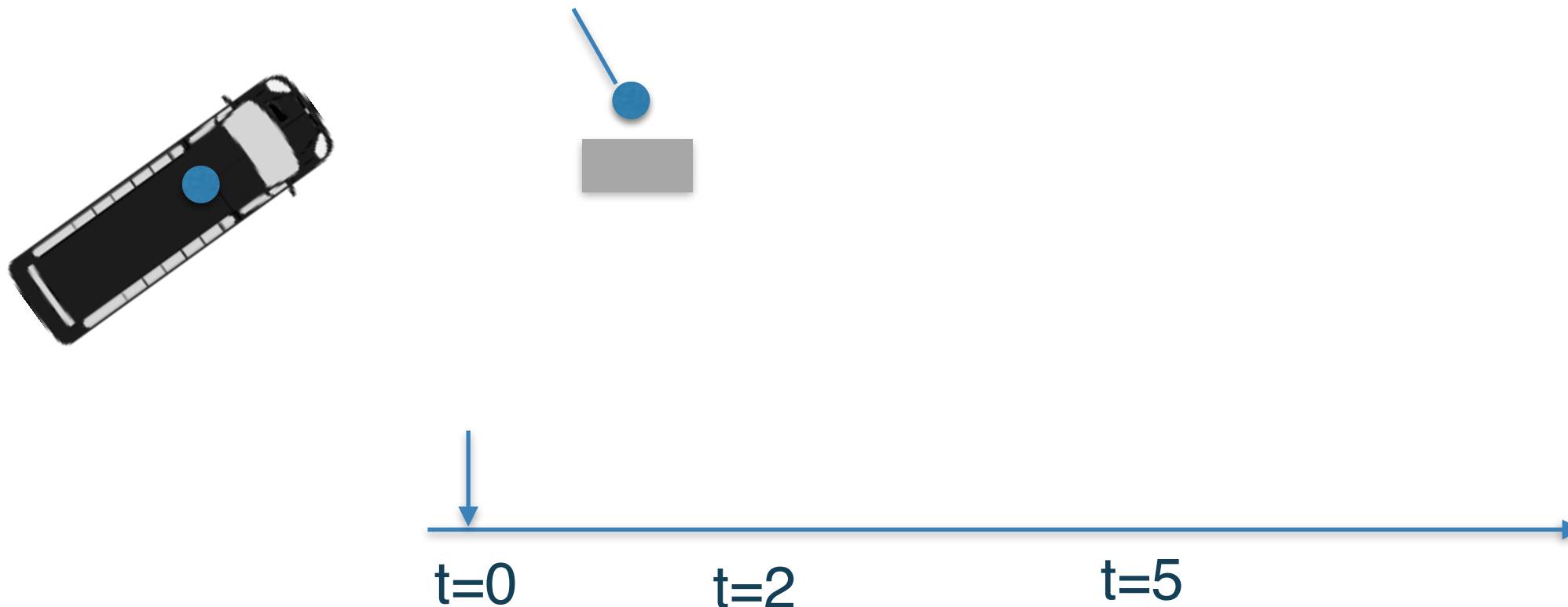
Time-varying Graphs

- A *time-varying graph* (TVG) is a 5-tuple $(\mathbf{V}, \mathbf{E}, \mathbf{T}, \mathbf{p}', \mathbf{l}')$
 - \mathbf{V} : set of nodes
 - \mathbf{E} : (labelled) set of edges
 - \mathbf{T} : lifetime, $\mathbf{T} \subseteq \mathcal{T}$
 - \mathbf{p}' : *node* presence function, $\mathbf{V} \times \mathbf{T} \longrightarrow \{0, 1\}$
 - \mathbf{l}' : *node* latency function, $\mathbf{V} \times \mathbf{T} \longrightarrow \mathcal{T}$

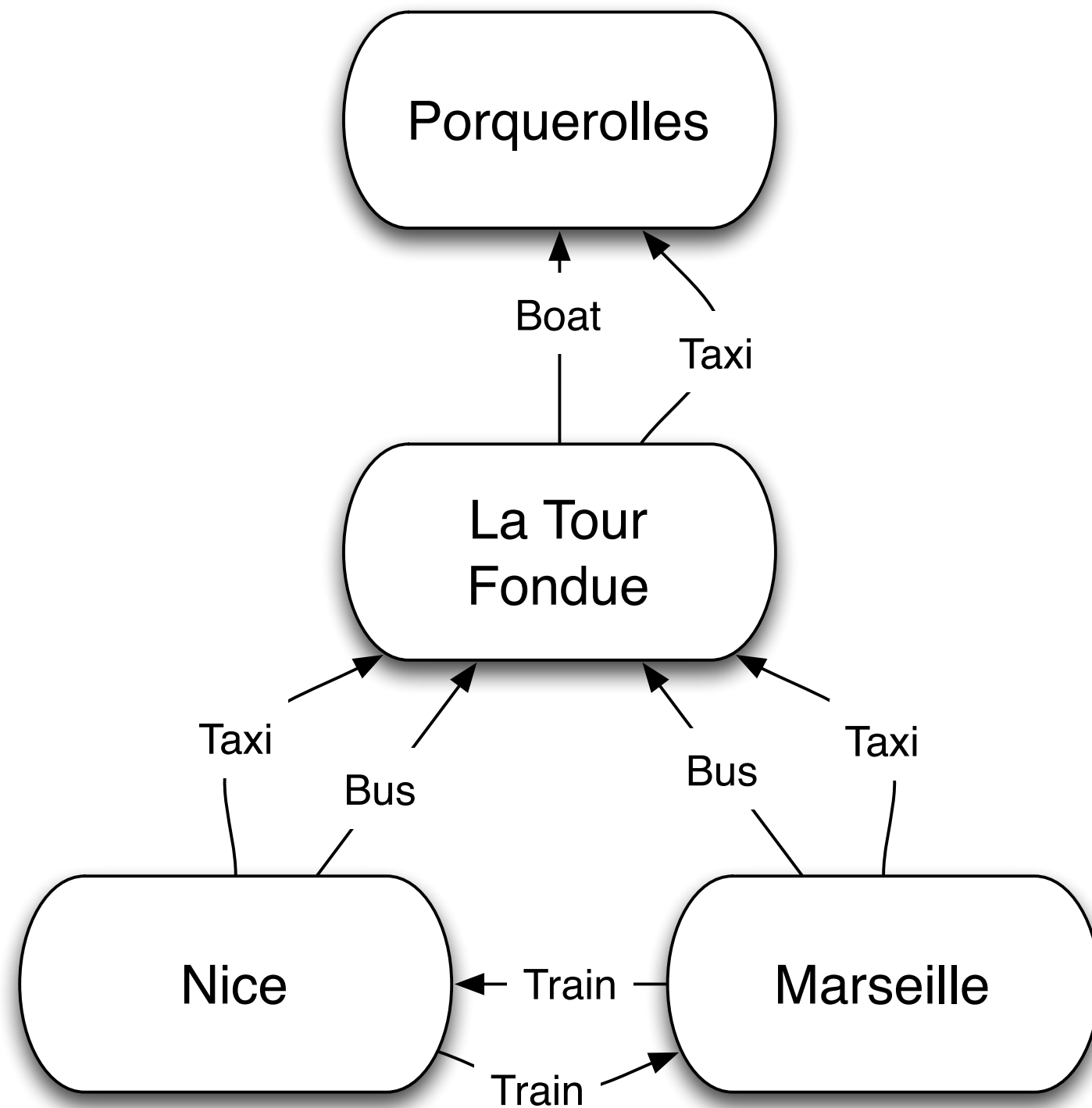
Time-varying Graphs



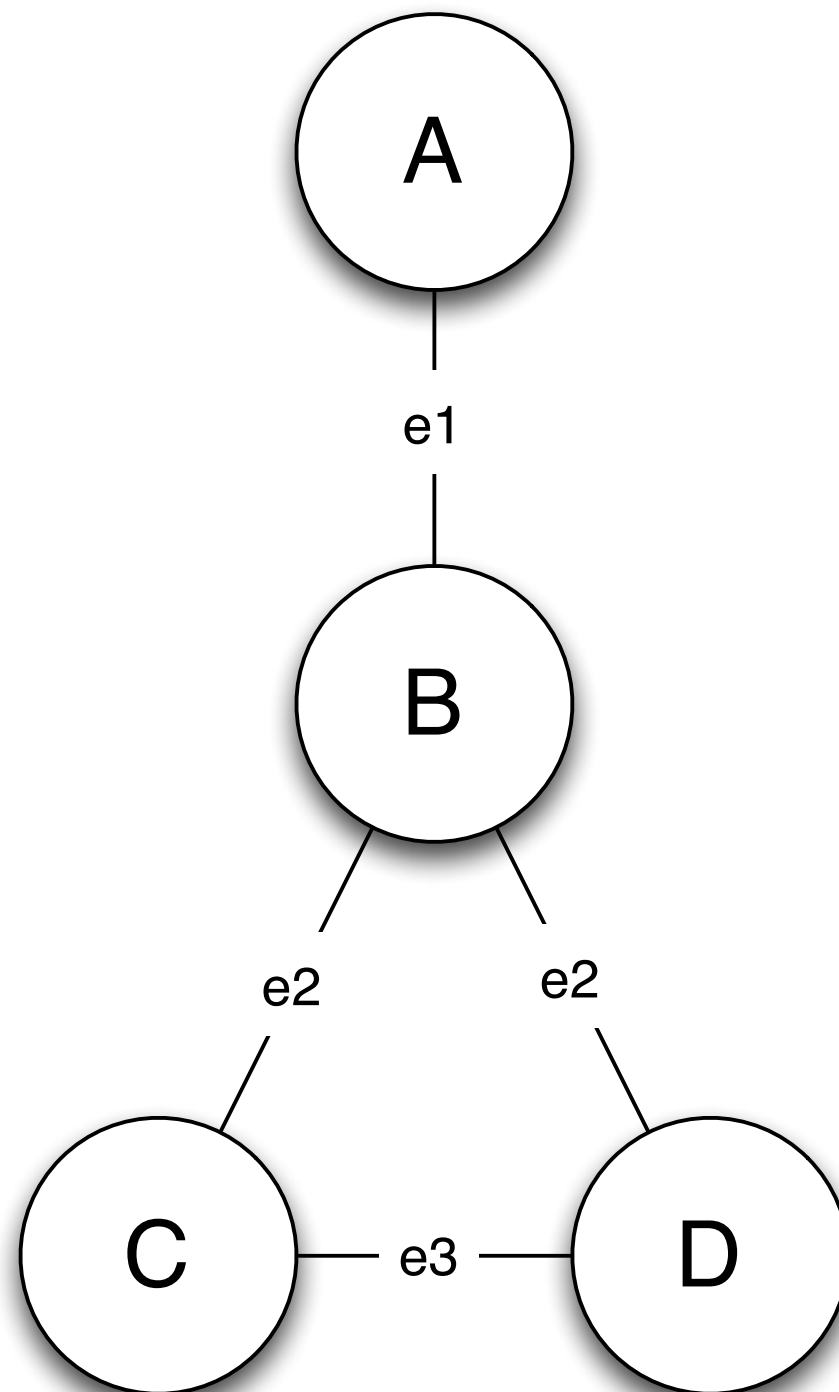
Evolving Graphs



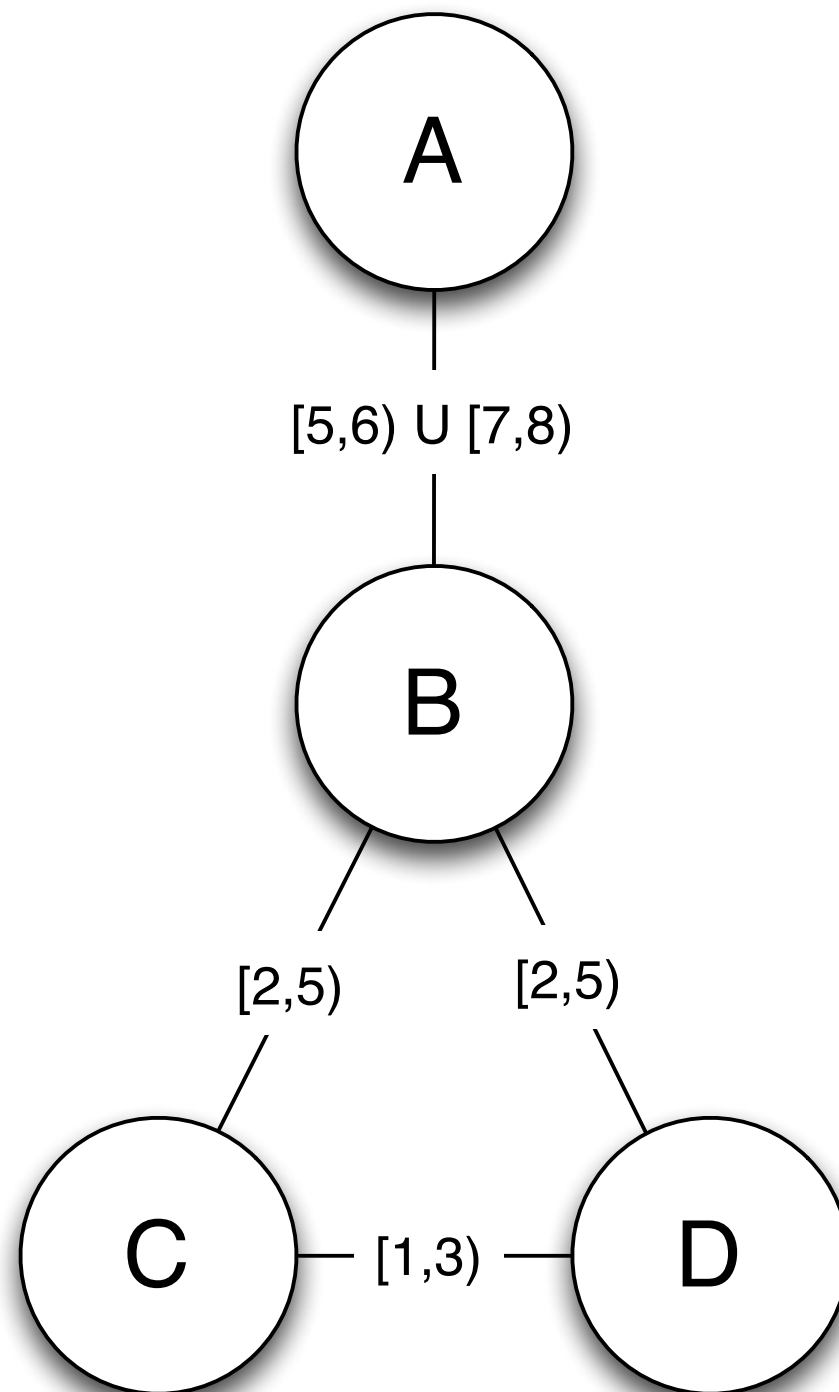
Example



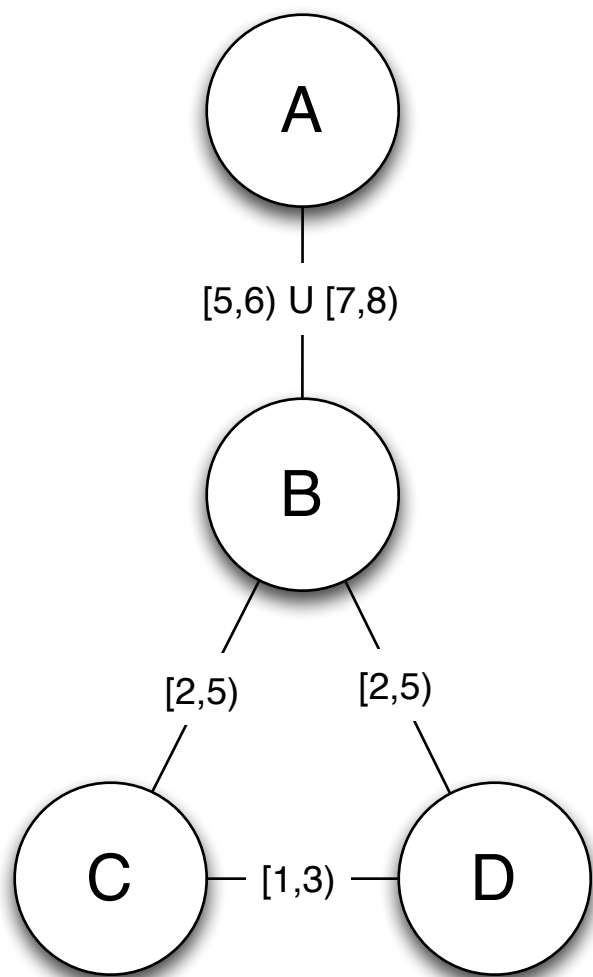
Example



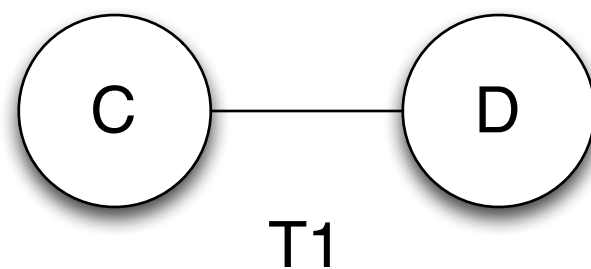
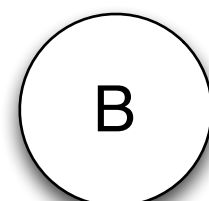
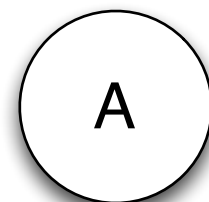
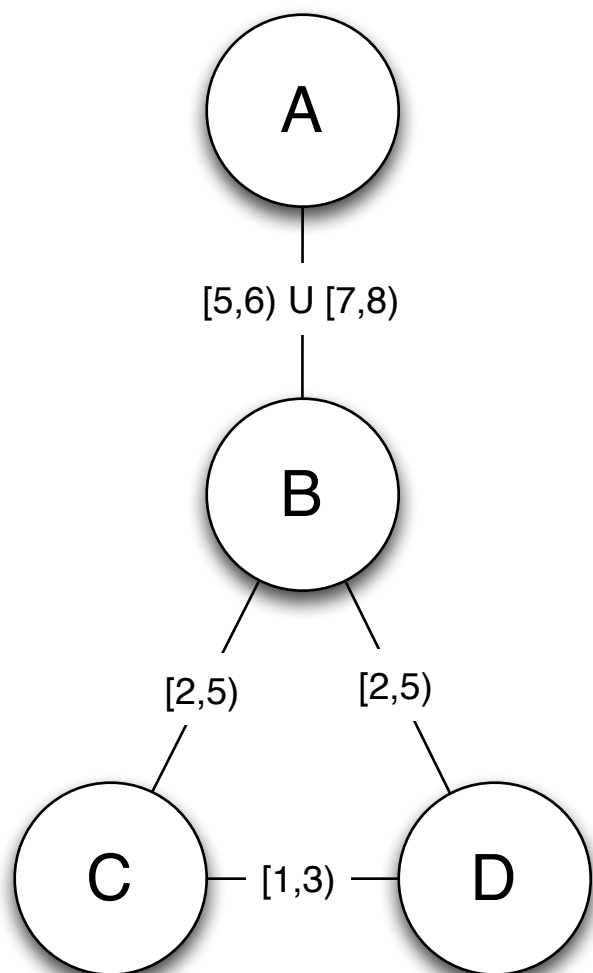
Example



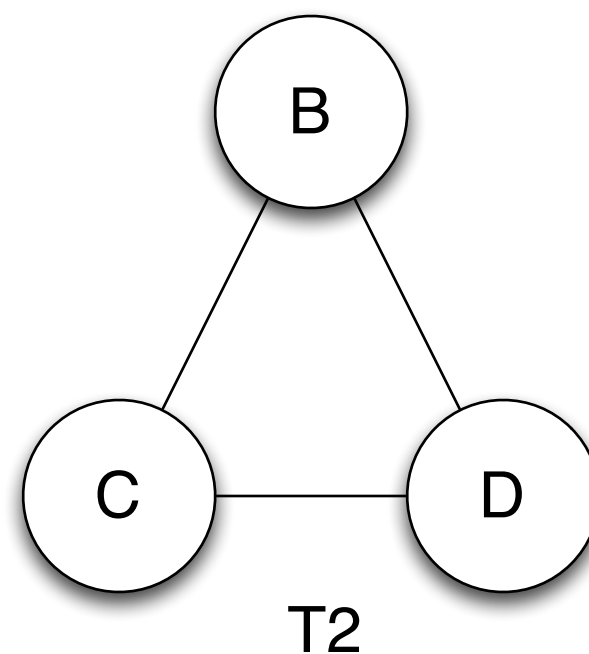
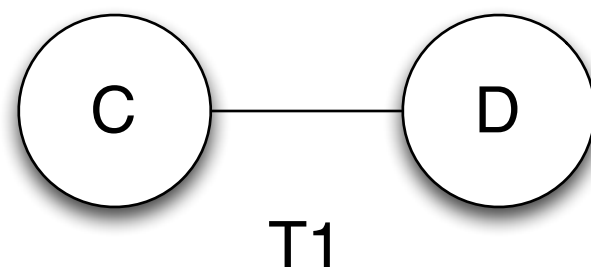
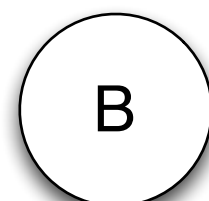
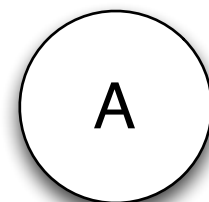
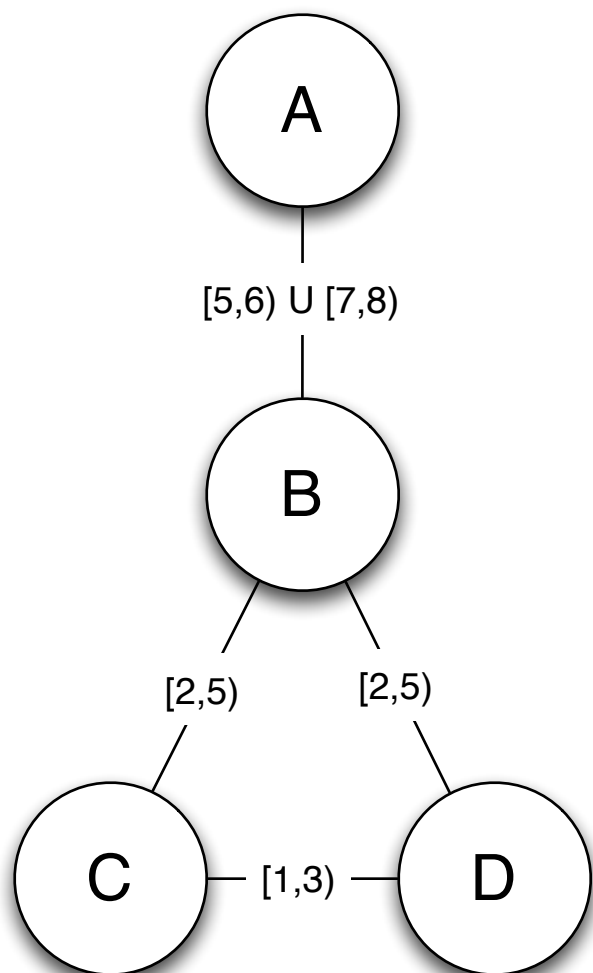
Evolving Graphs



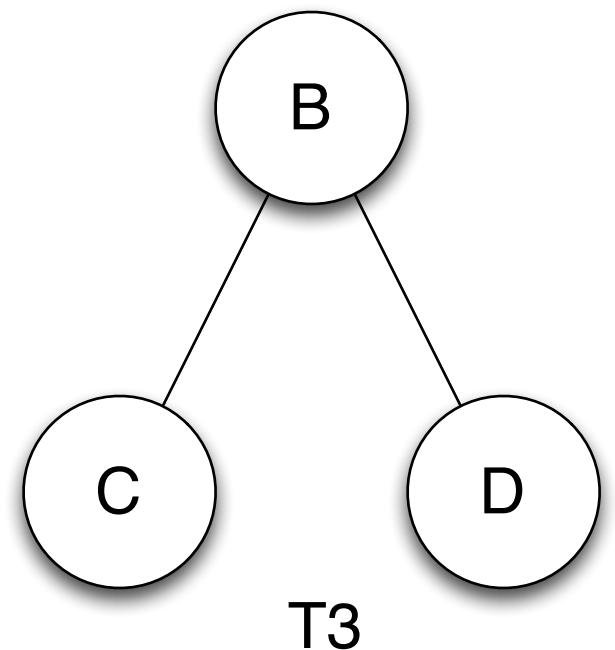
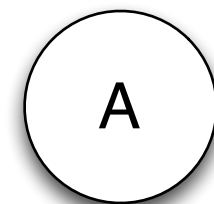
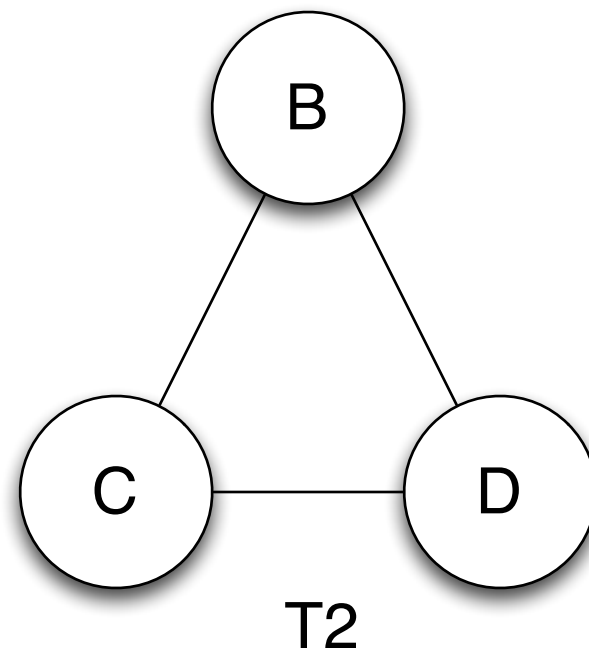
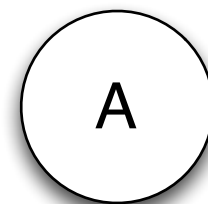
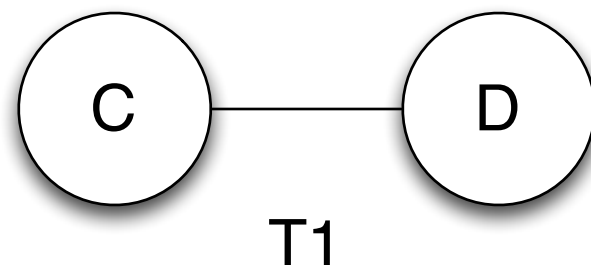
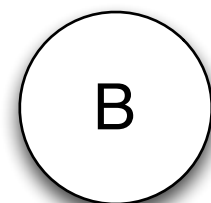
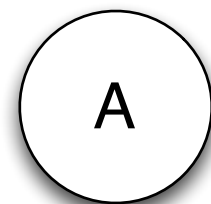
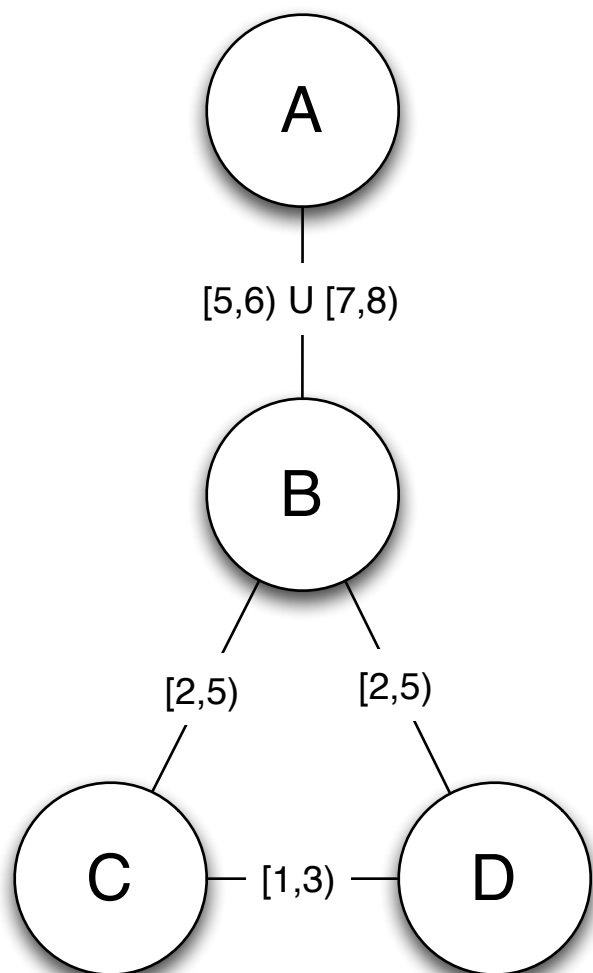
Evolving Graphs



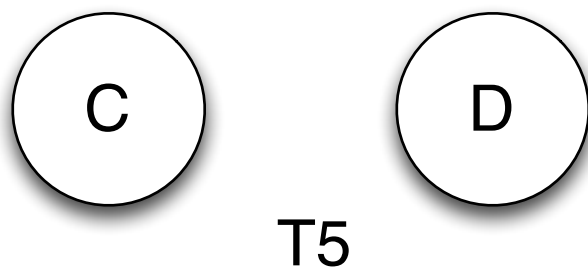
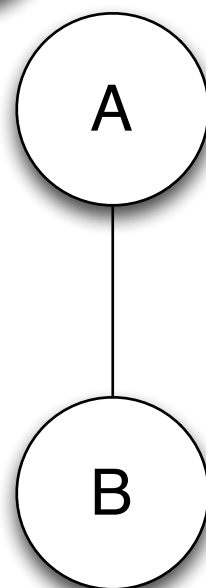
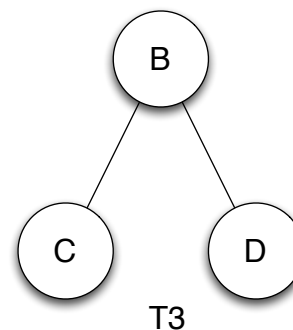
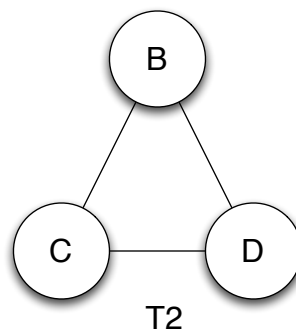
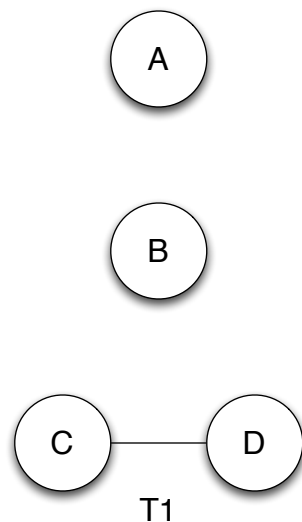
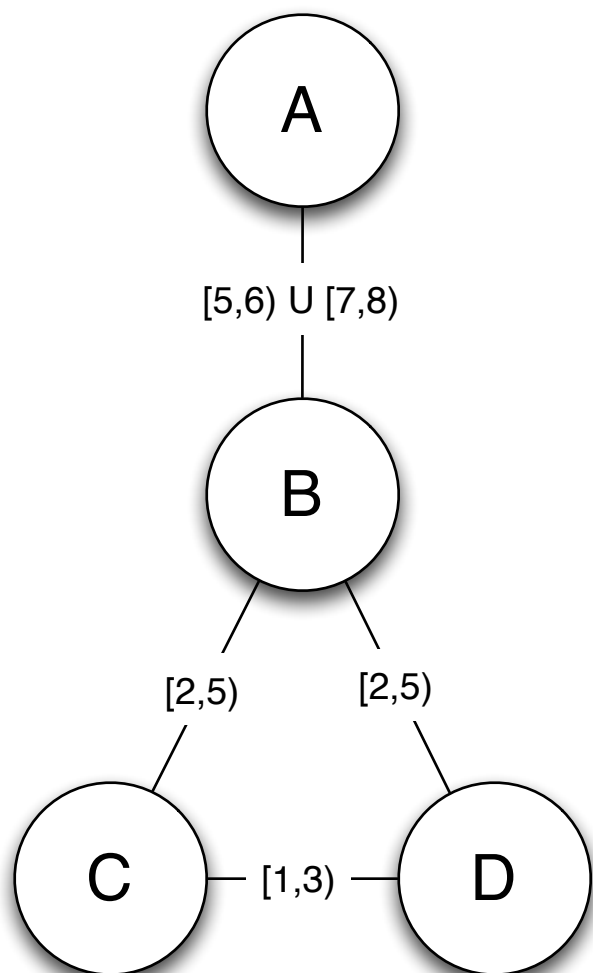
Evolving Graphs



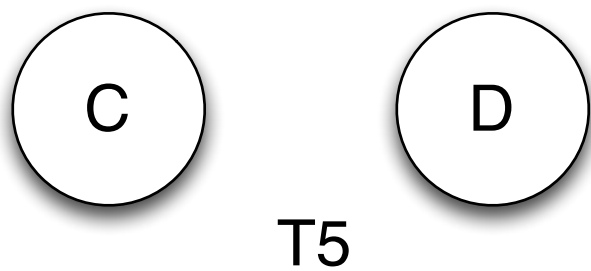
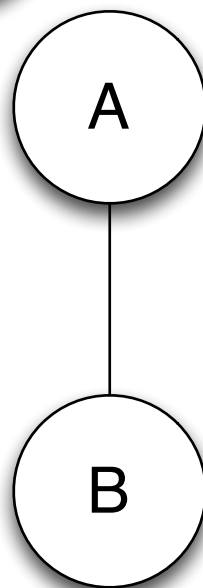
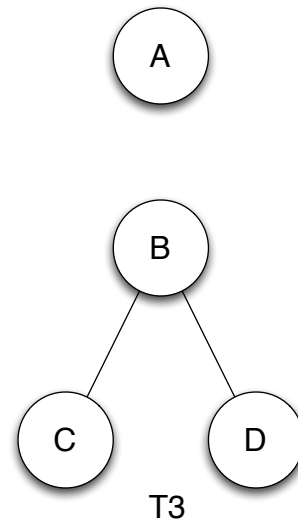
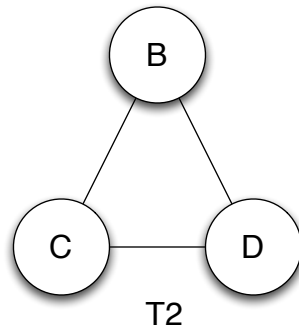
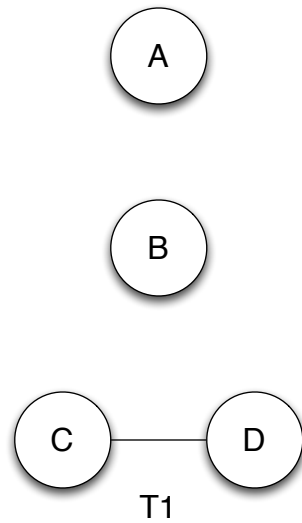
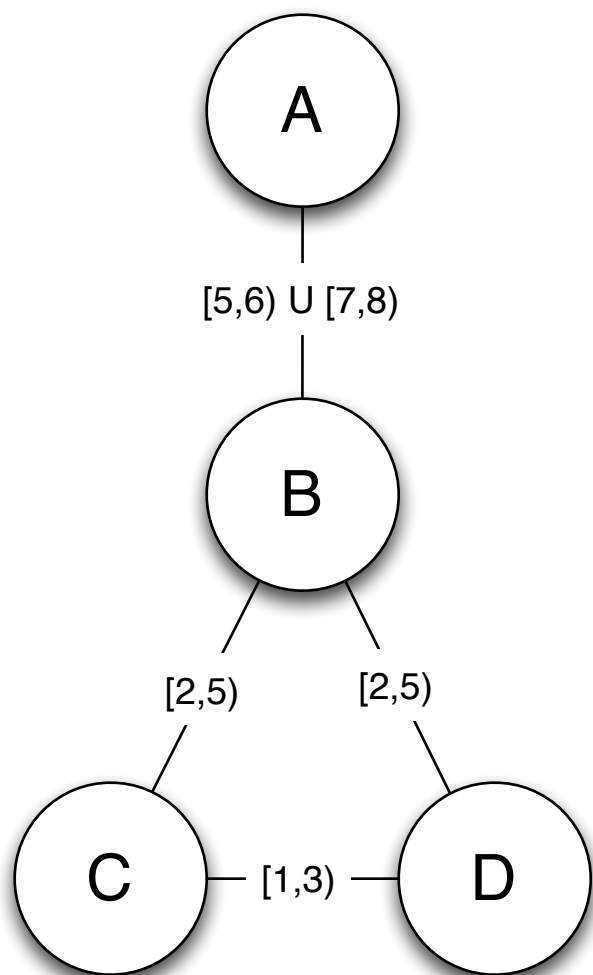
Evolving Graphs



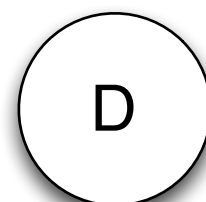
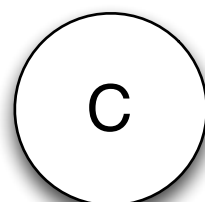
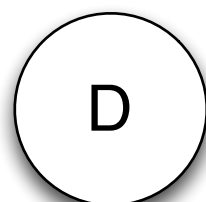
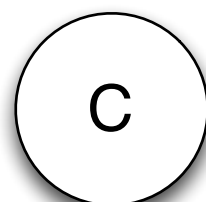
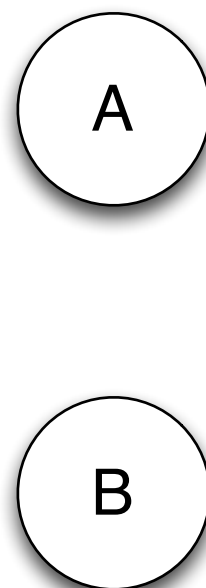
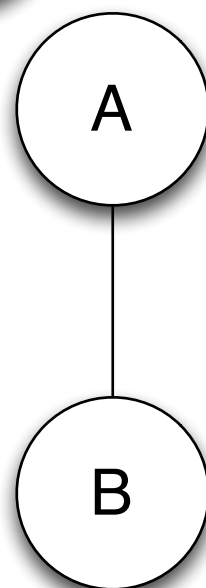
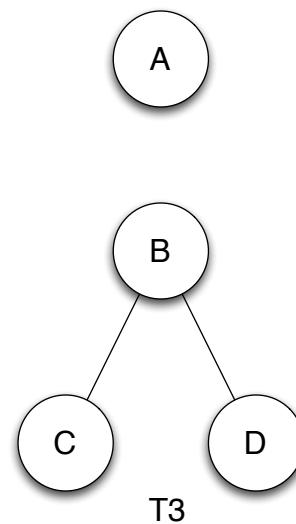
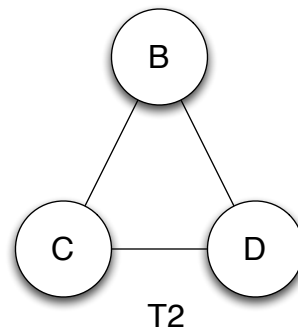
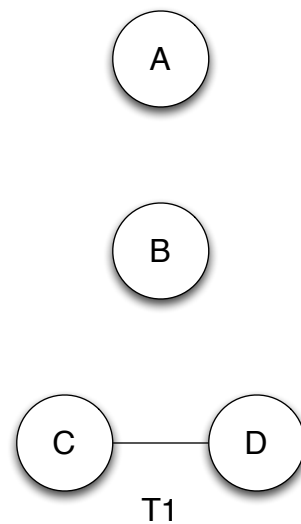
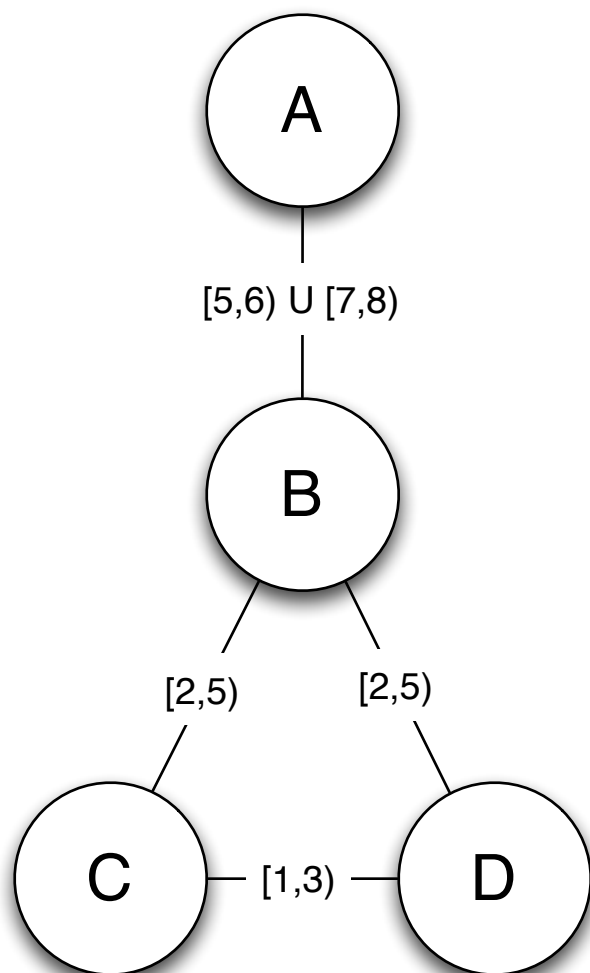
Evolving Graphs



Evolving Graphs



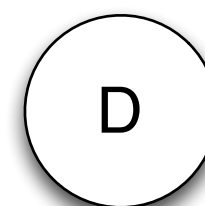
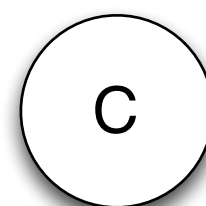
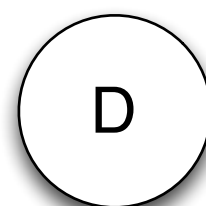
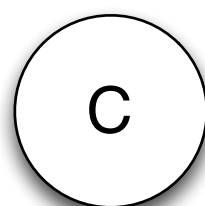
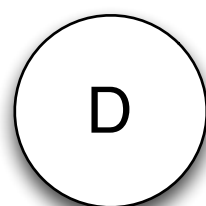
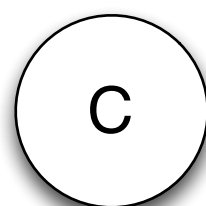
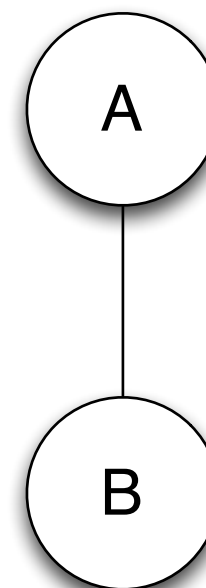
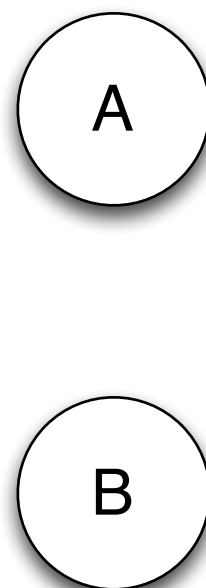
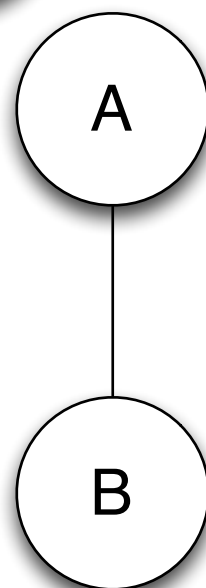
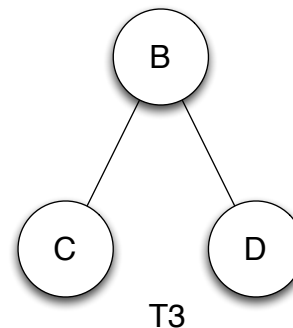
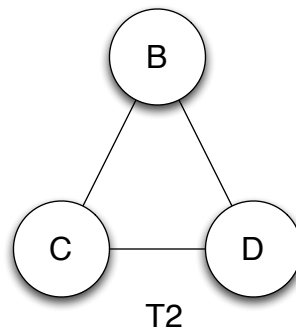
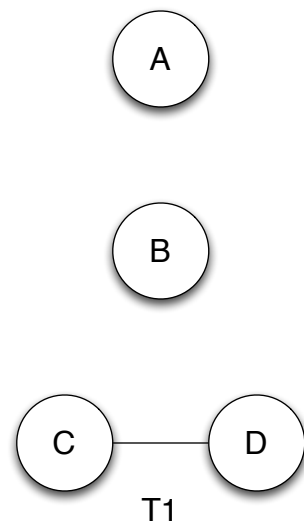
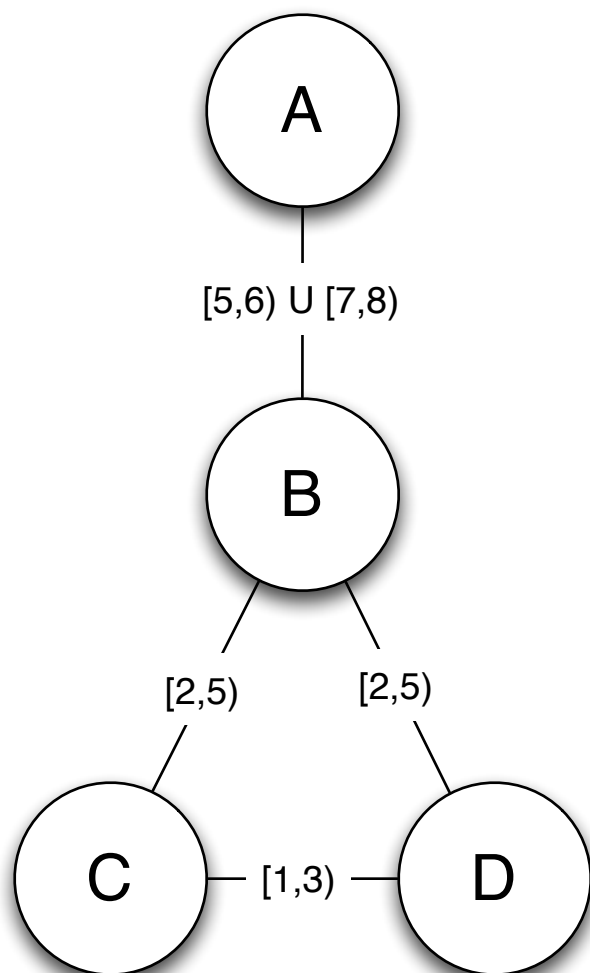
Evolving Graphs



T5

T6

Evolving Graphs

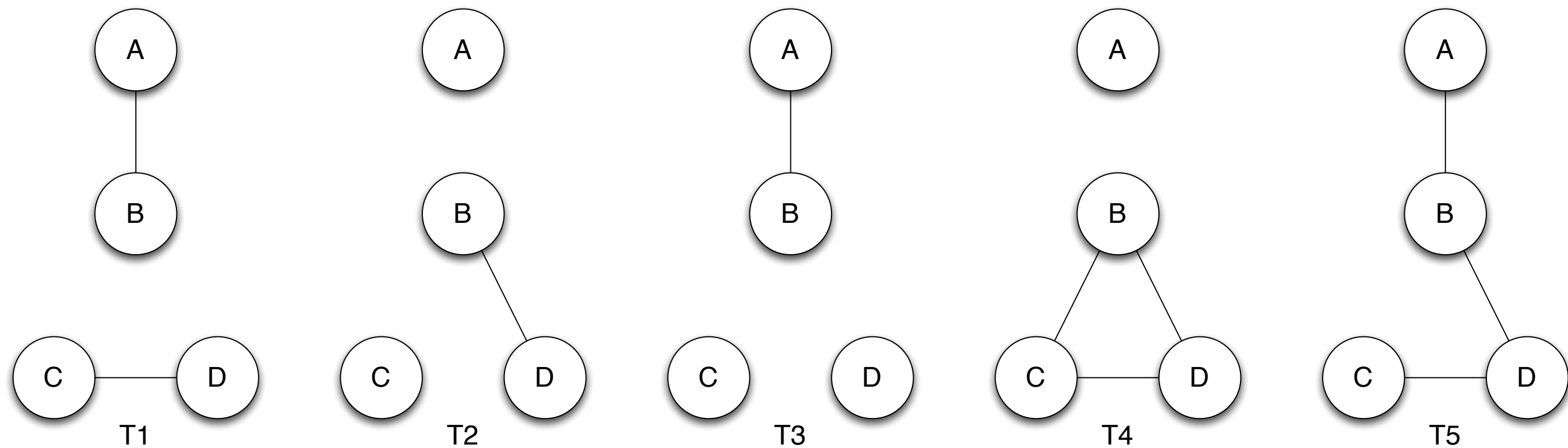


T5

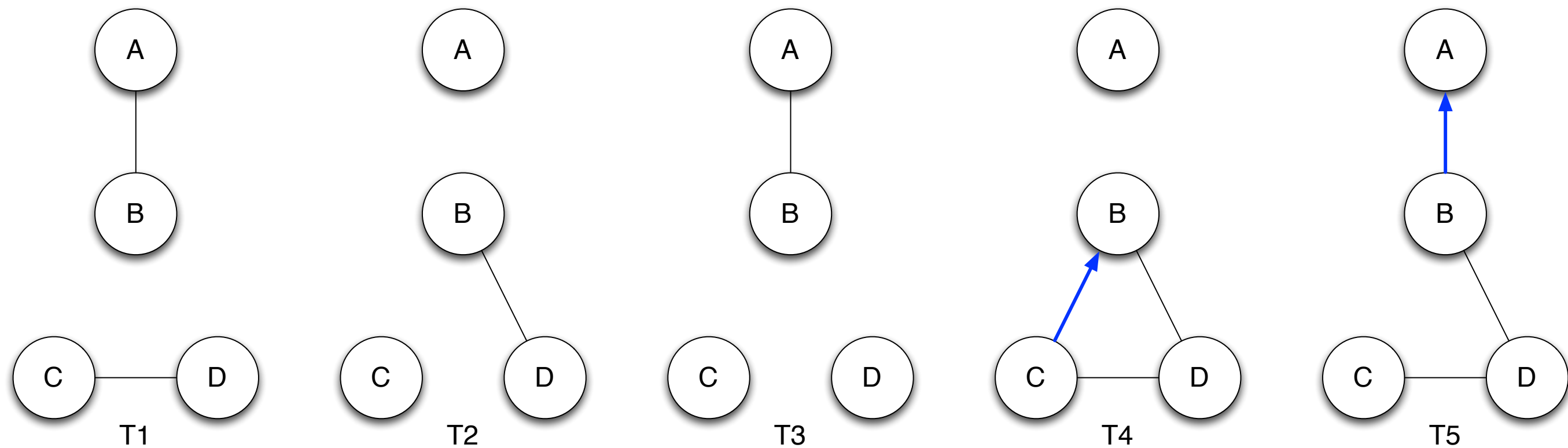
T6

T7

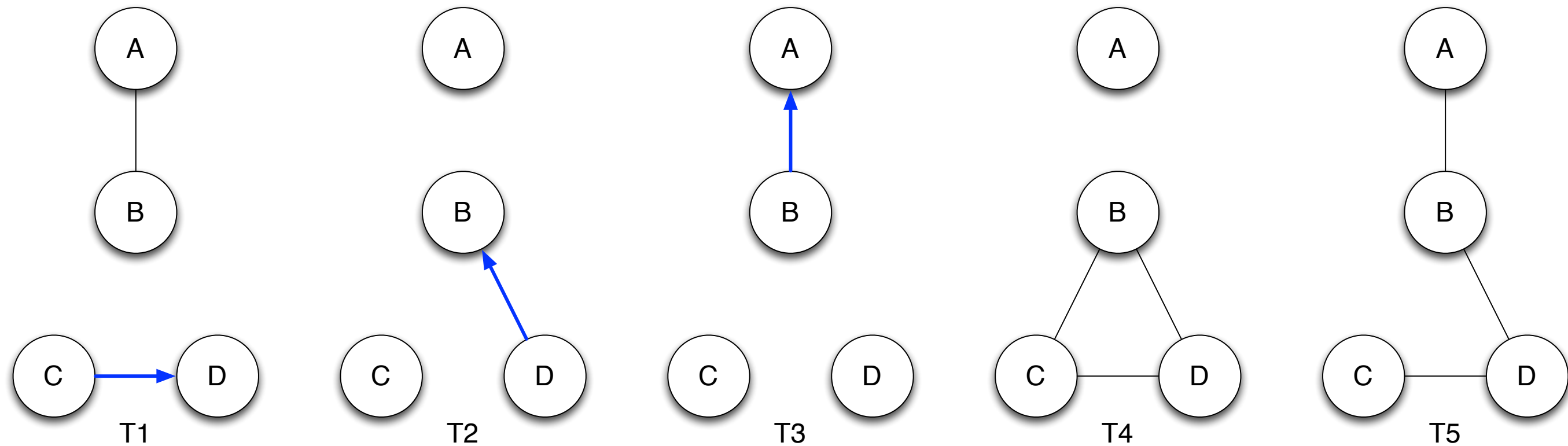
Journeys from C to A



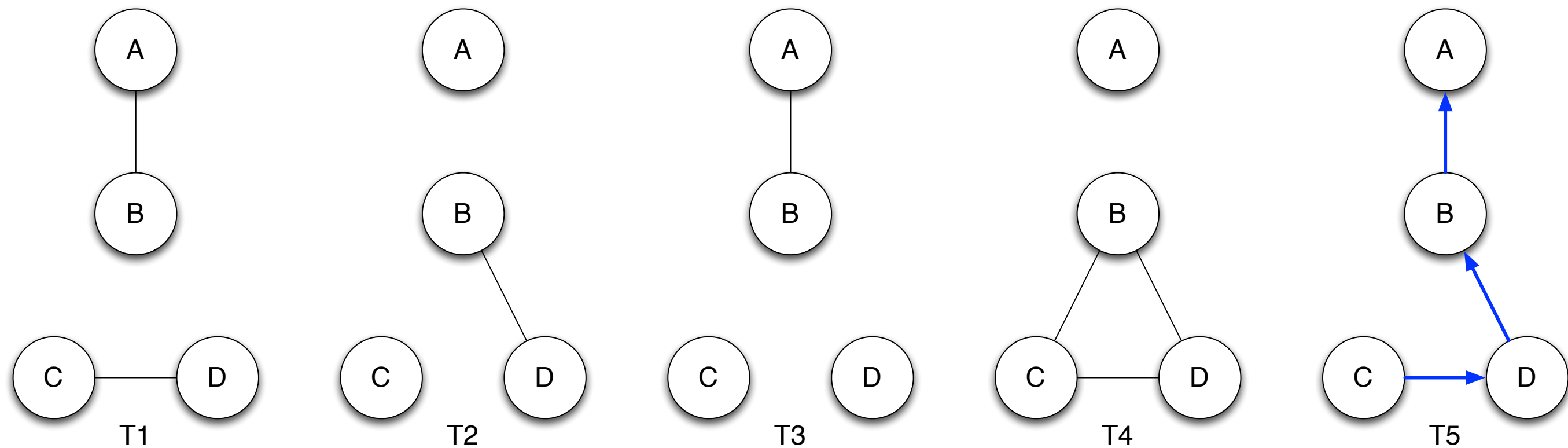
Shortest Journey



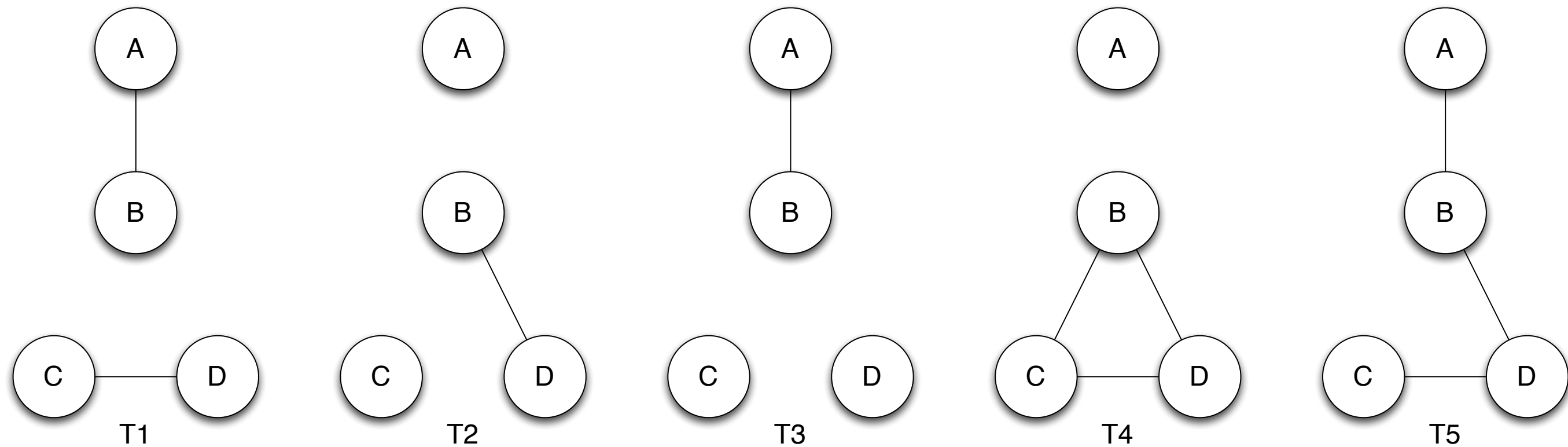
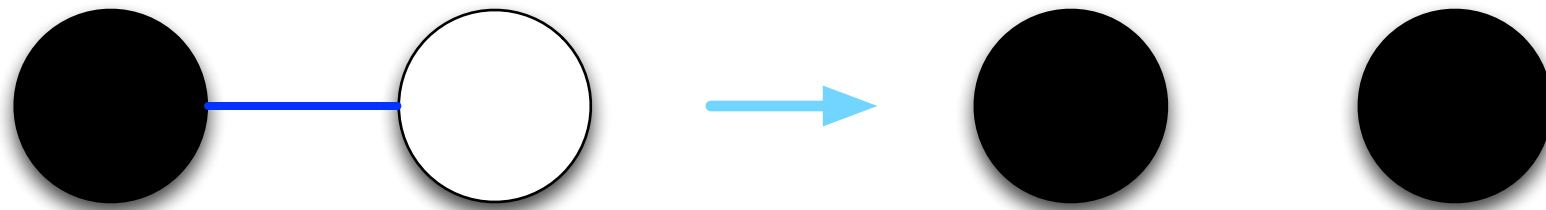
Foremost Journey



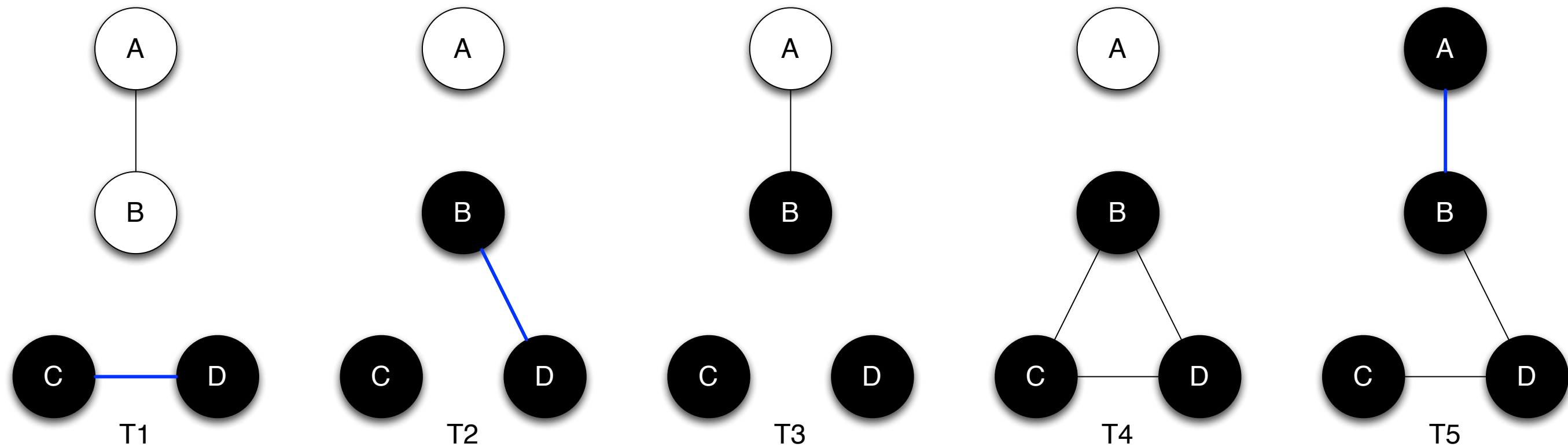
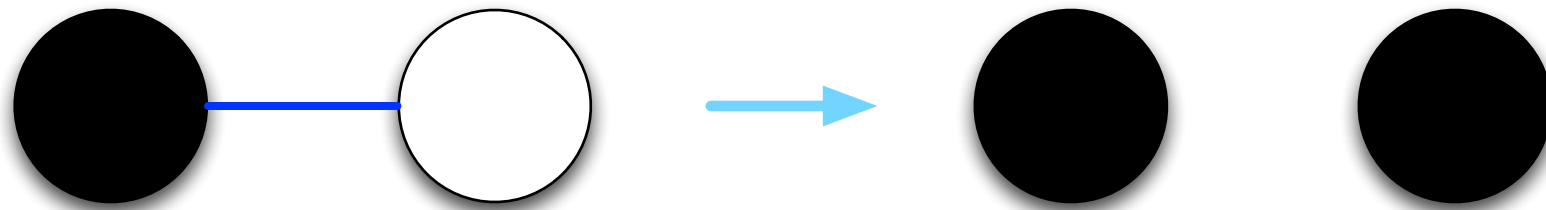
Fastest Journey



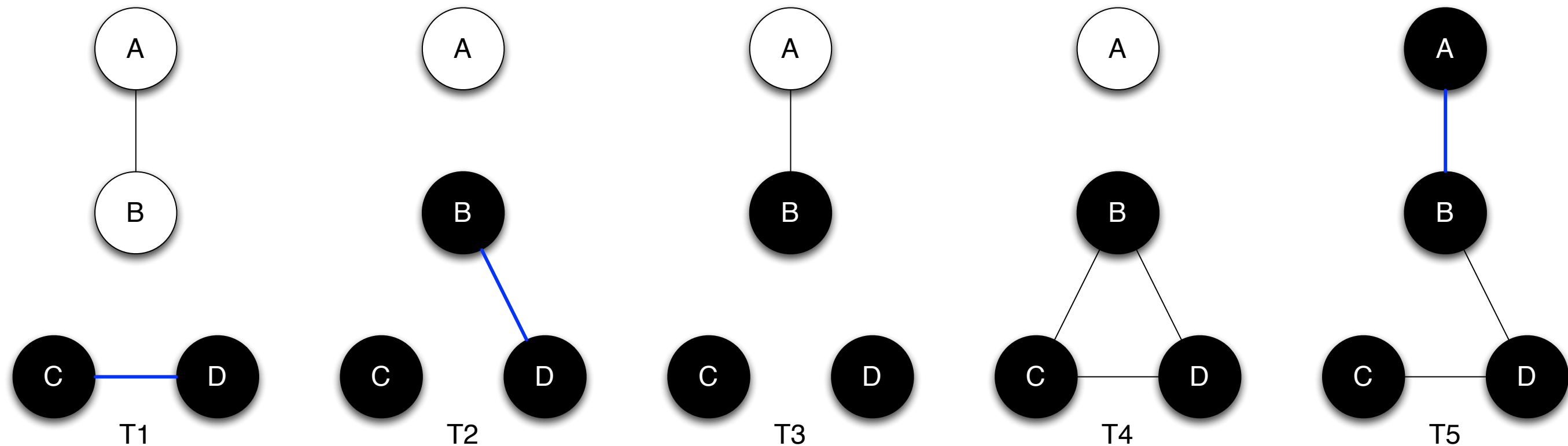
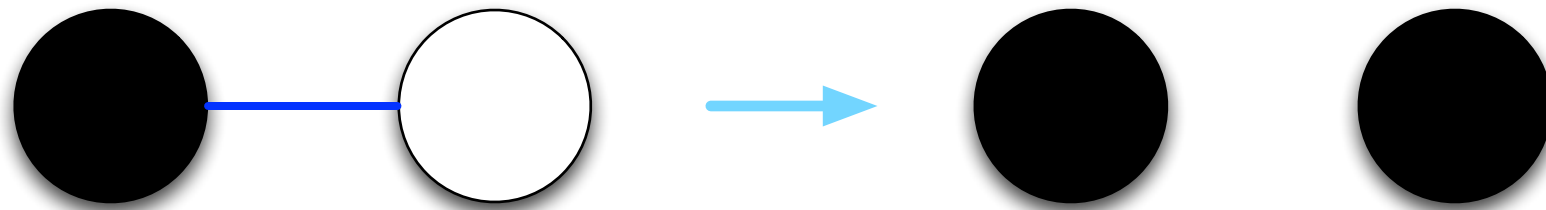
Condition for Broadcast?



Condition for Broadcast?

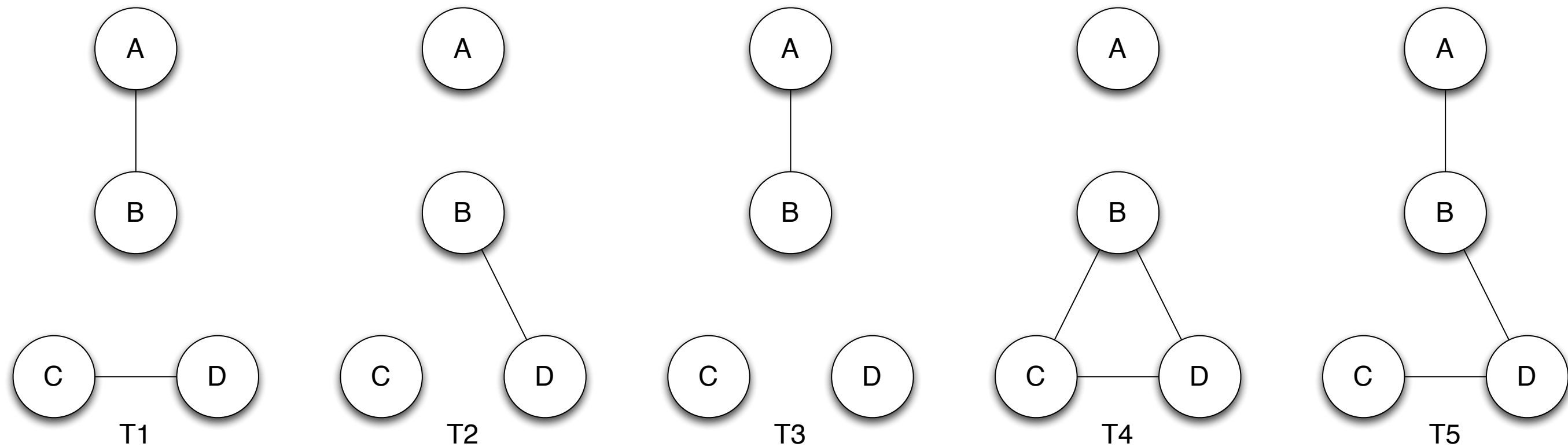
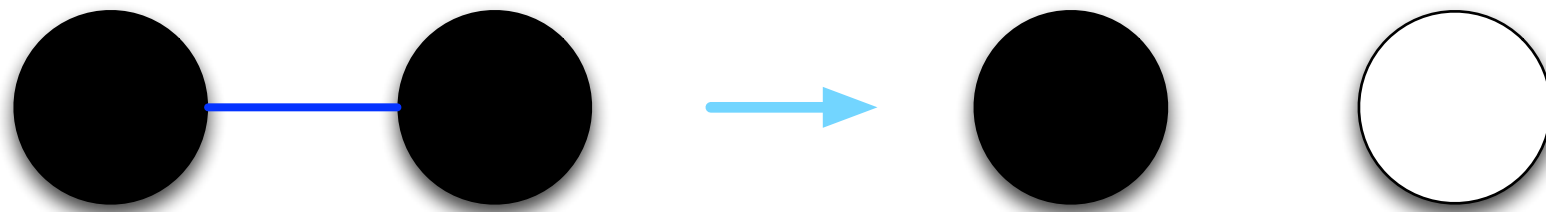


Condition for Broadcast?

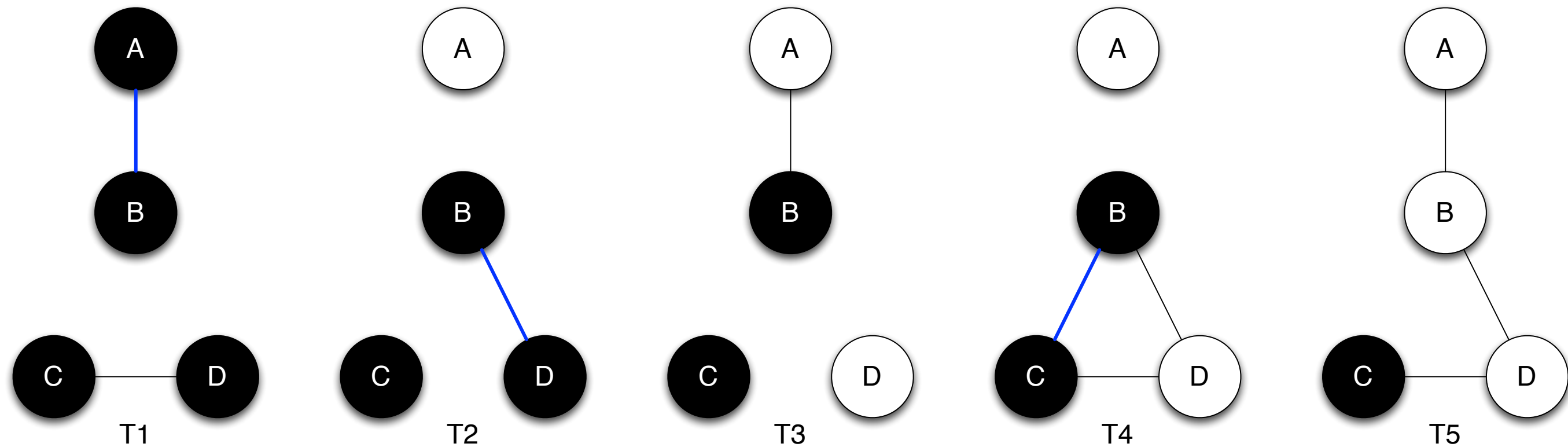
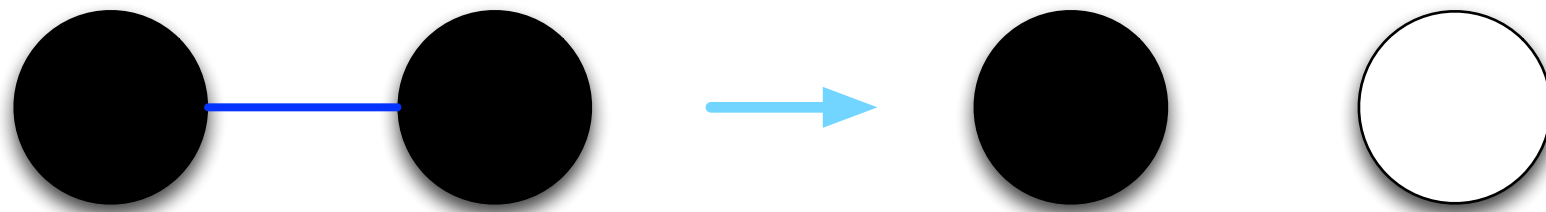


There exists a node (**C**) from which a journey reaches every other node

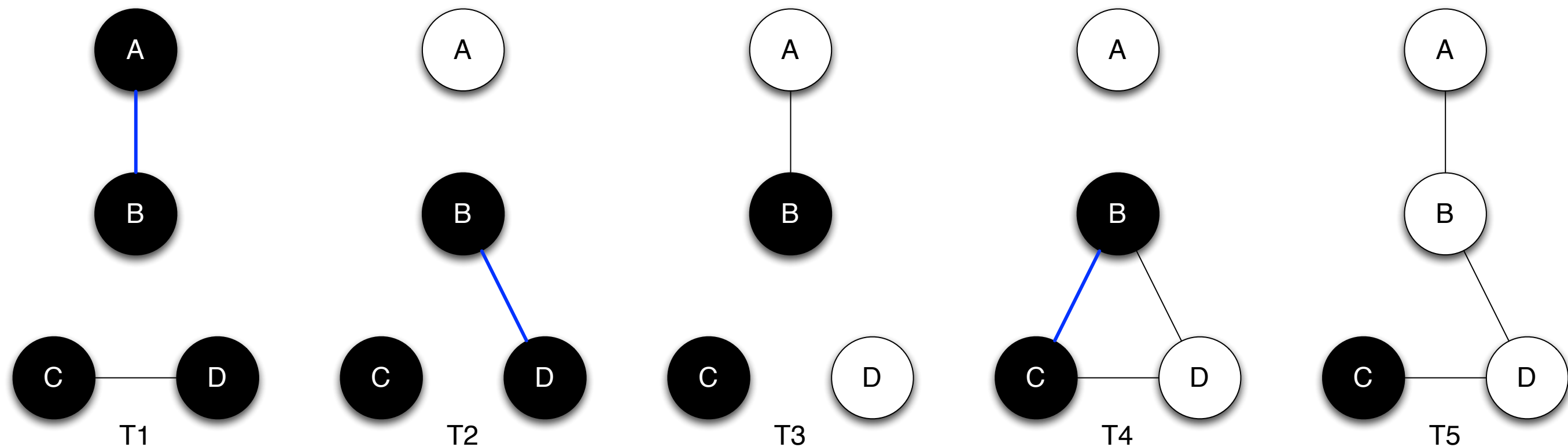
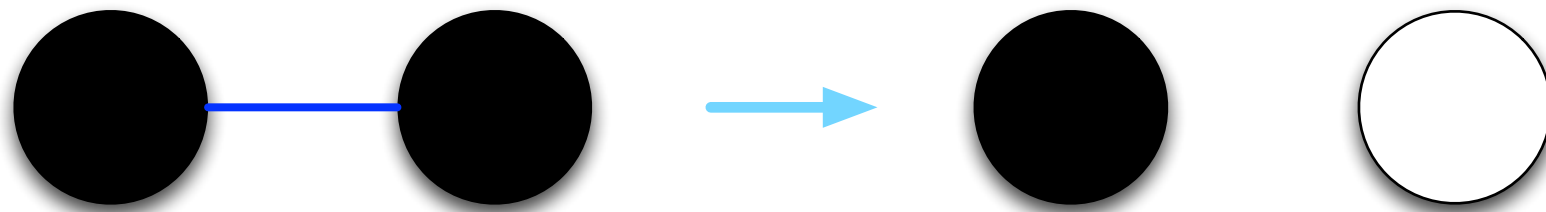
Condition for Election?



Condition for Election?

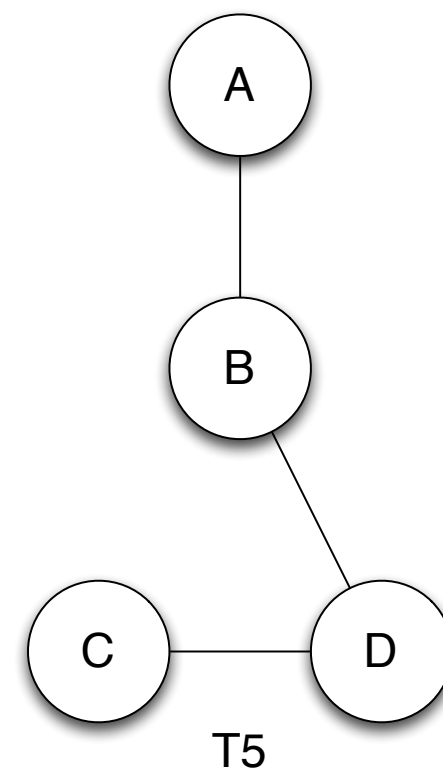
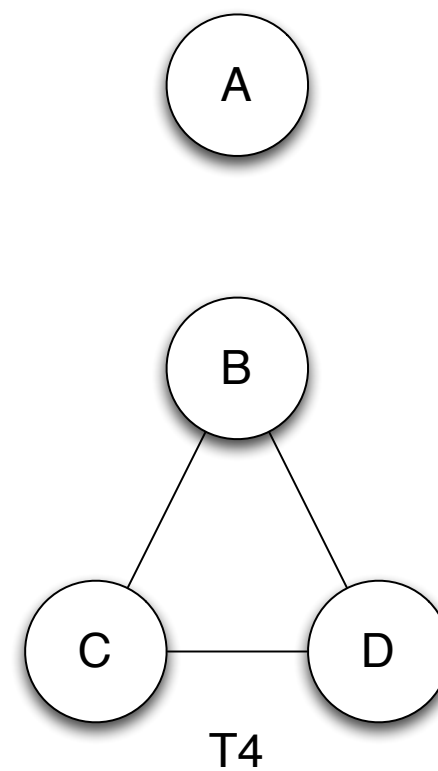
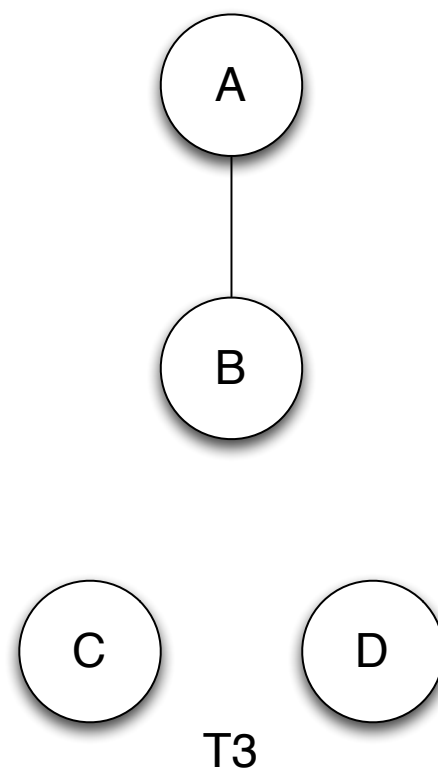
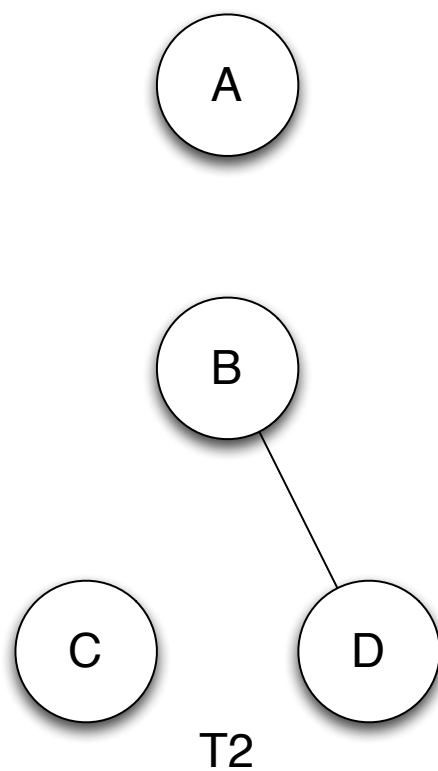
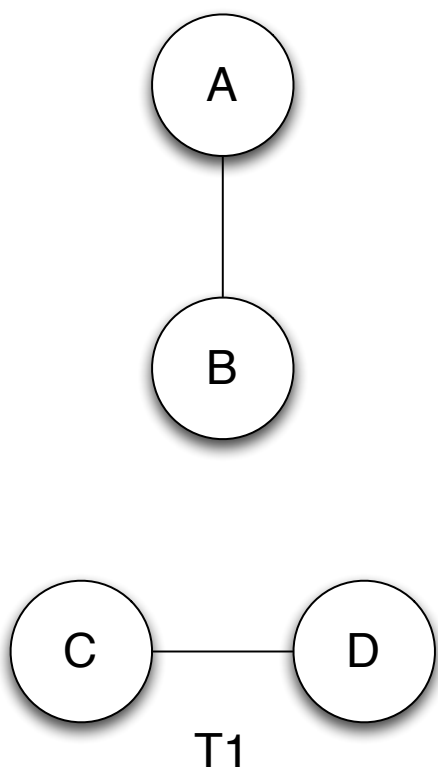
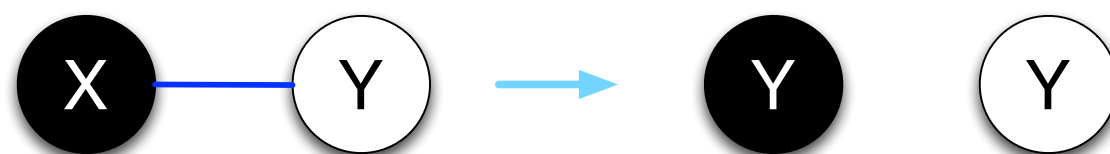
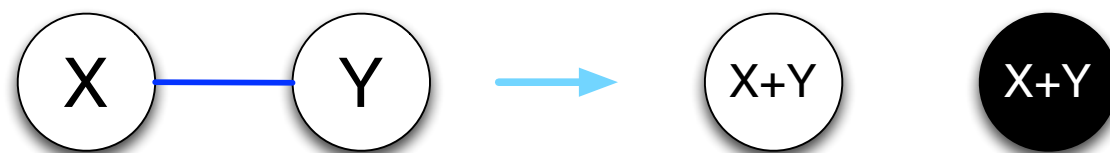


Condition for Election?

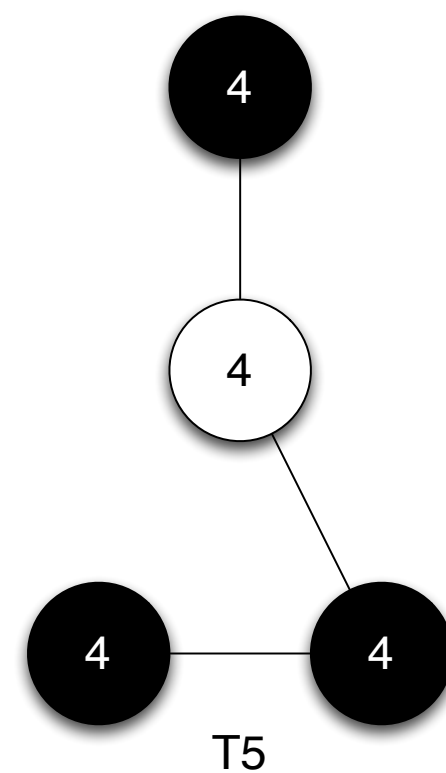
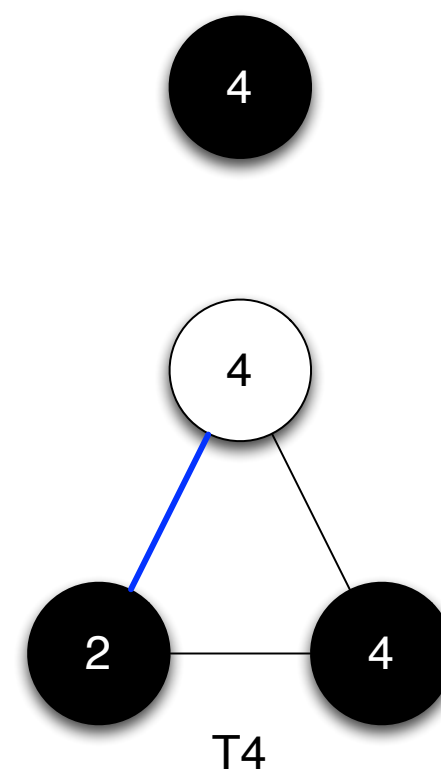
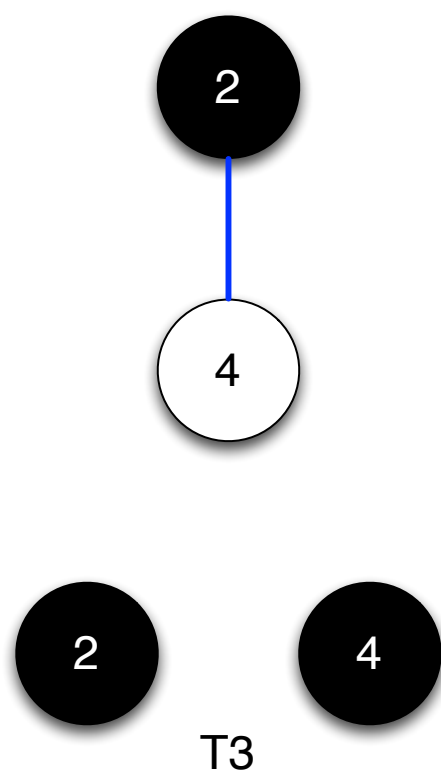
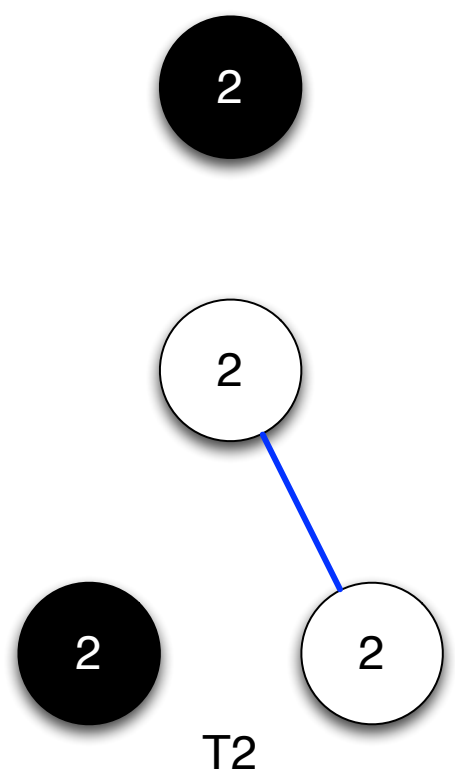
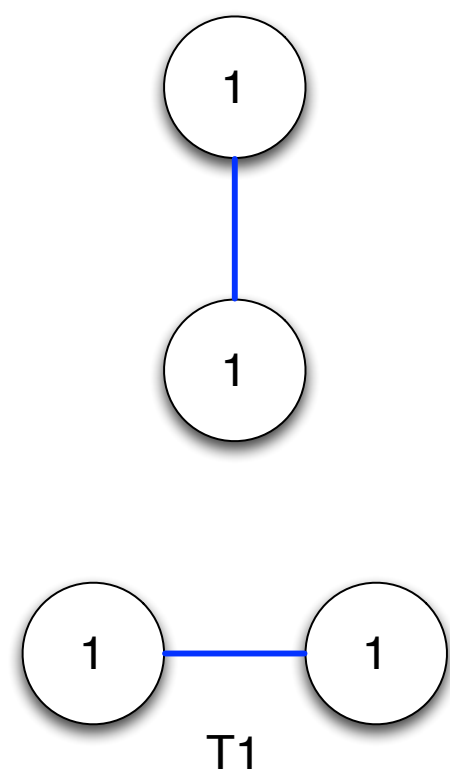
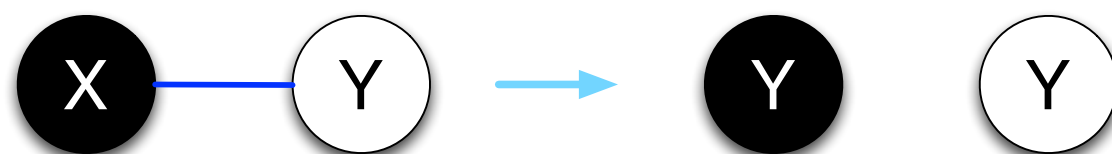
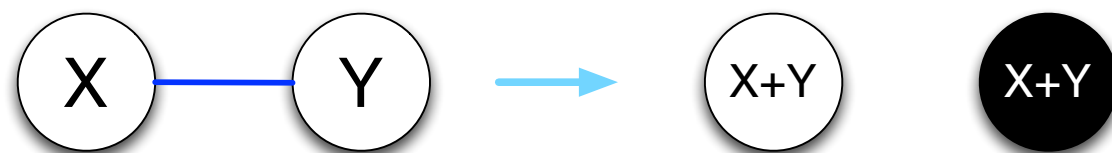


There exists a node (**C**) such that there exists a journey from every other node to it

Condition for Global Calculus?

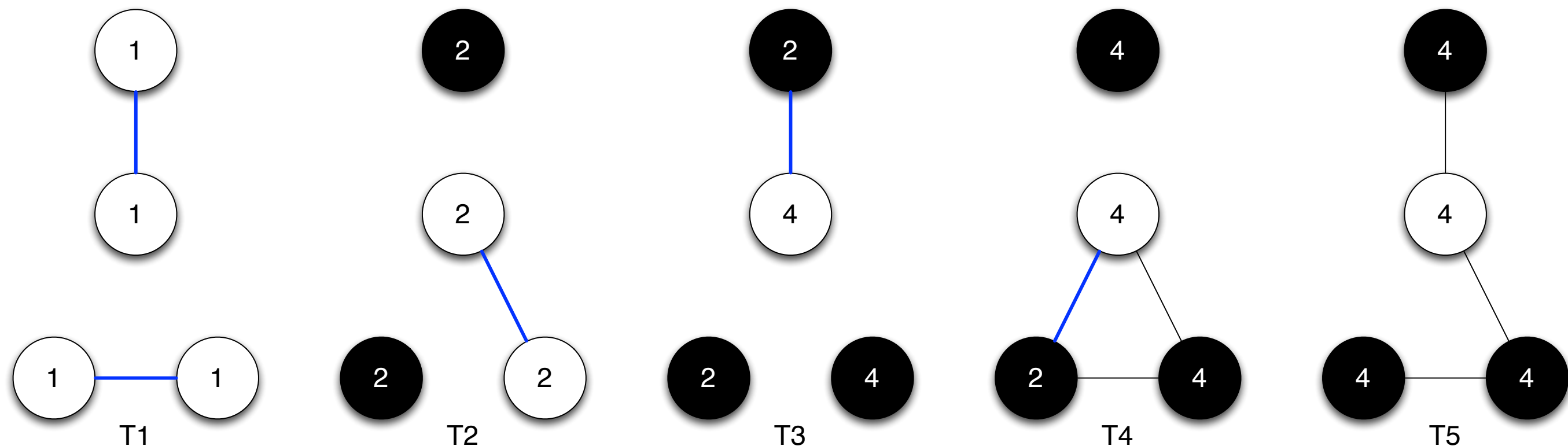


Condition for Global Calculus?



Condition for Global Calculus?

There exists a node (**Center**) such that there exists a journey from every other node to it *and back*



Connectivity Classes

- There exists a node r from which a journey reaches every other node $1 \rightsquigarrow *$
- There exists a node r such that there exists a journey from every other node to it $* \rightsquigarrow 1$
- There exists a node r such that there exists a journey from every other node to to and back $1 \overset{\leftarrow}{\rightsquigarrow} *$

More Classes

- There exists a journey between any two nodes $* \rightsquigarrow *$
- There exists a roundtrip journey between any two nodes $* \rightsquigarrow^* *$
- There exists a journey between any two nodes infinitely often $* \overset{\mathcal{R}}{\rightsquigarrow} *$
- Every edge appears infinitely often $\bullet \overset{\mathcal{R}}{\text{---}} \bullet$

More Classes

- Every edge appears infinitely often, and there is an upper bound between two occurrences

• $\underline{\mathcal{B}}$ •

- Every edge appears infinitely often with some period p

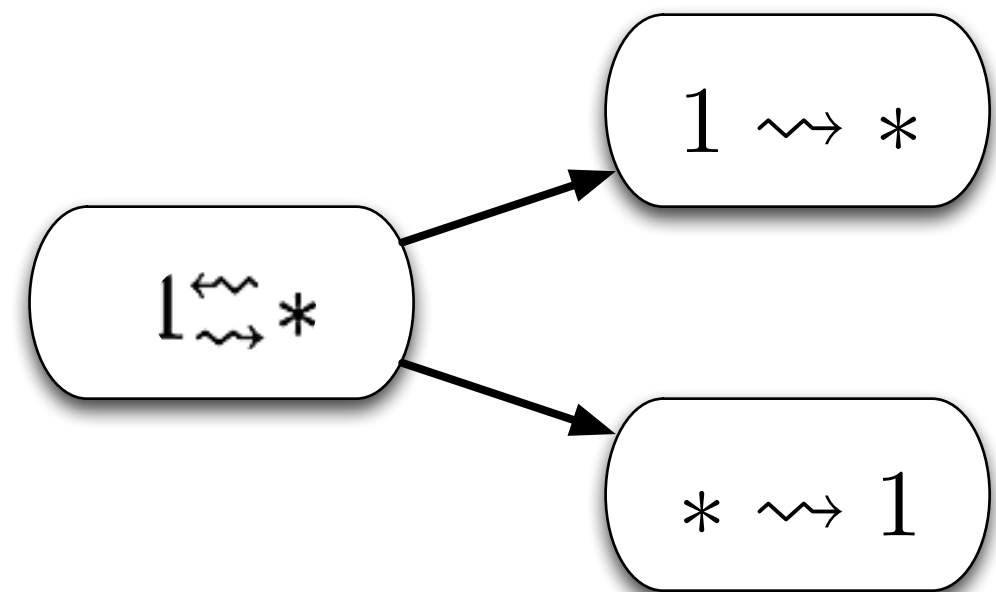
• $\underline{\mathcal{P}}$ •

More Classes

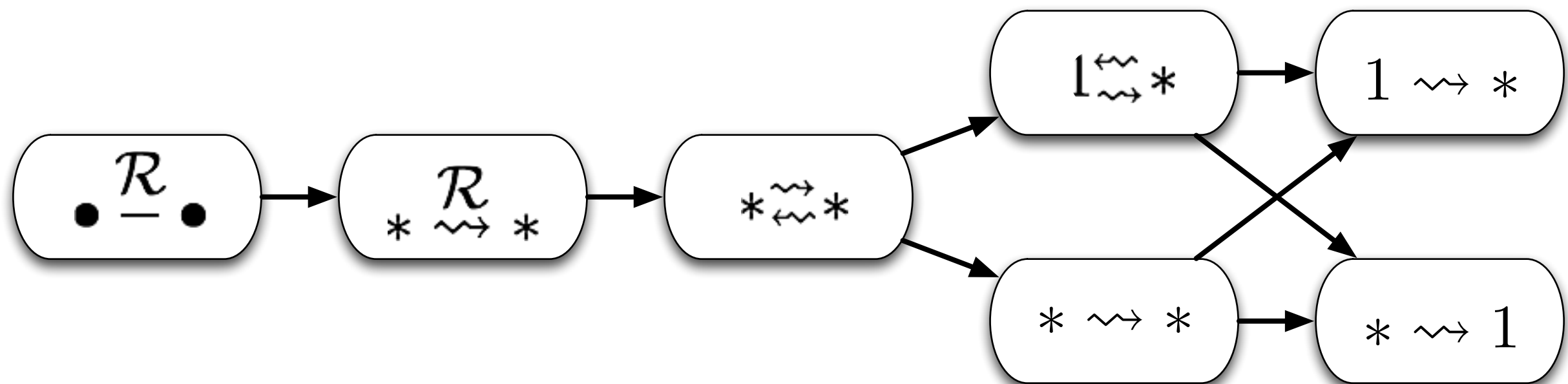
- At any time, the graph is connected
- Every spanning subgraph lasts at least T time units
- Every edge appears infinitely often, and the underlying graph is a clique

\mathcal{R}
* — *

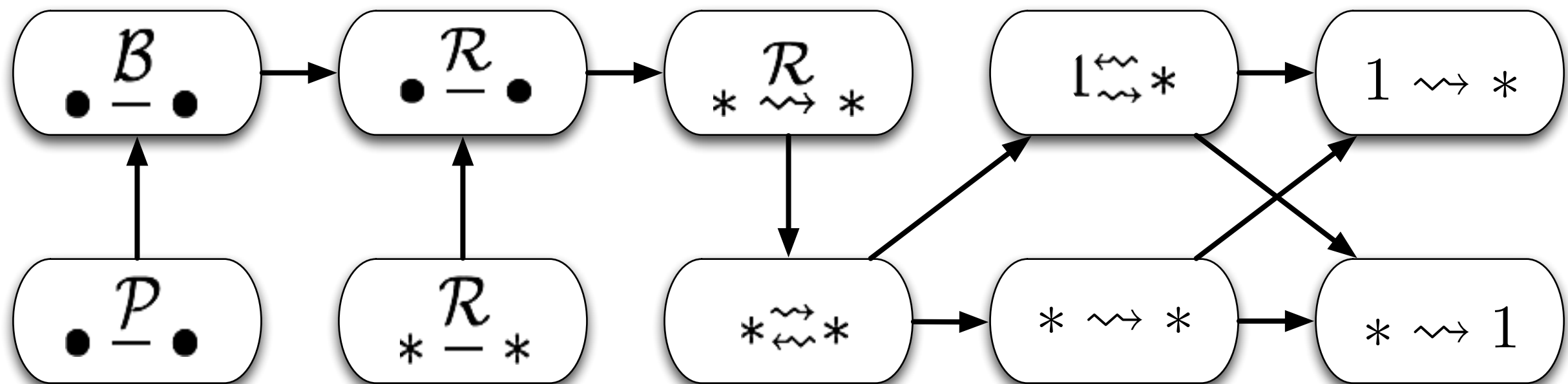
A Classification



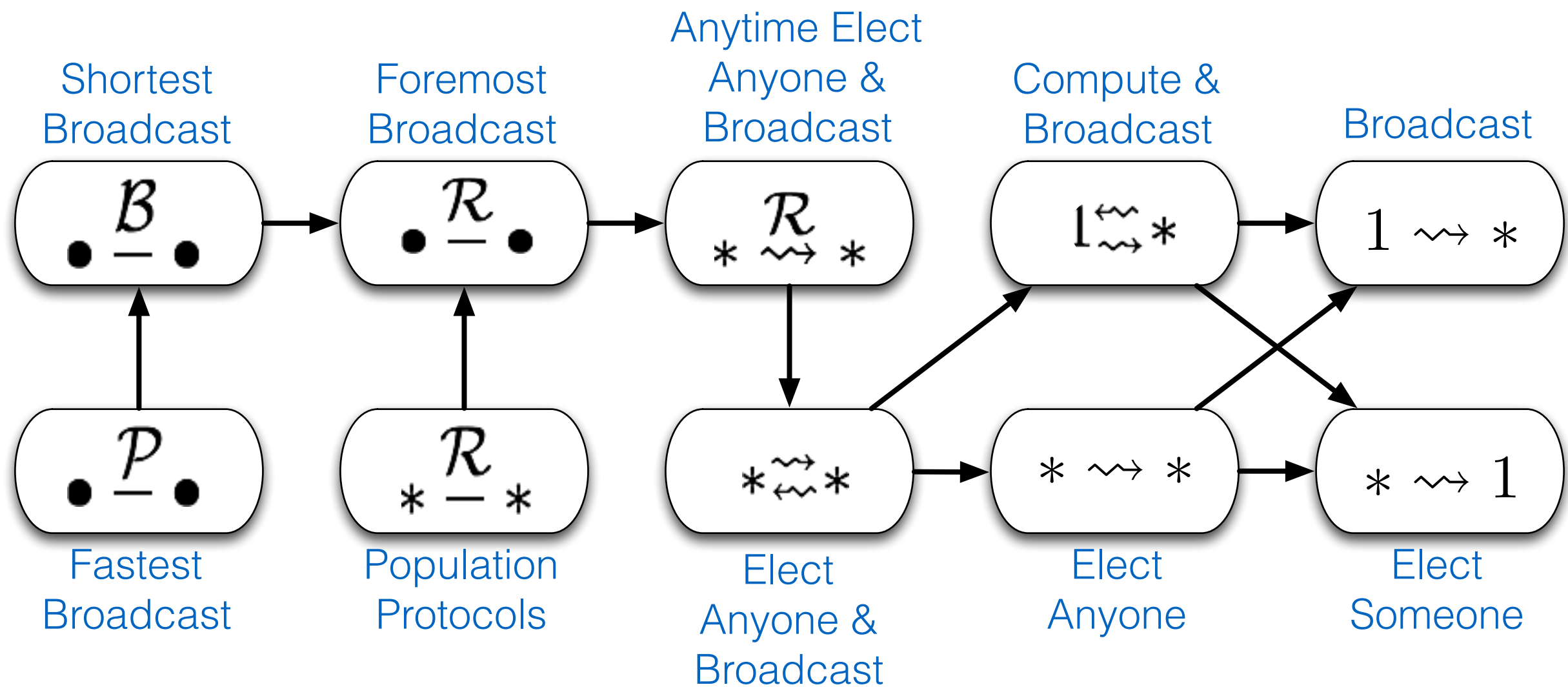
A Classification



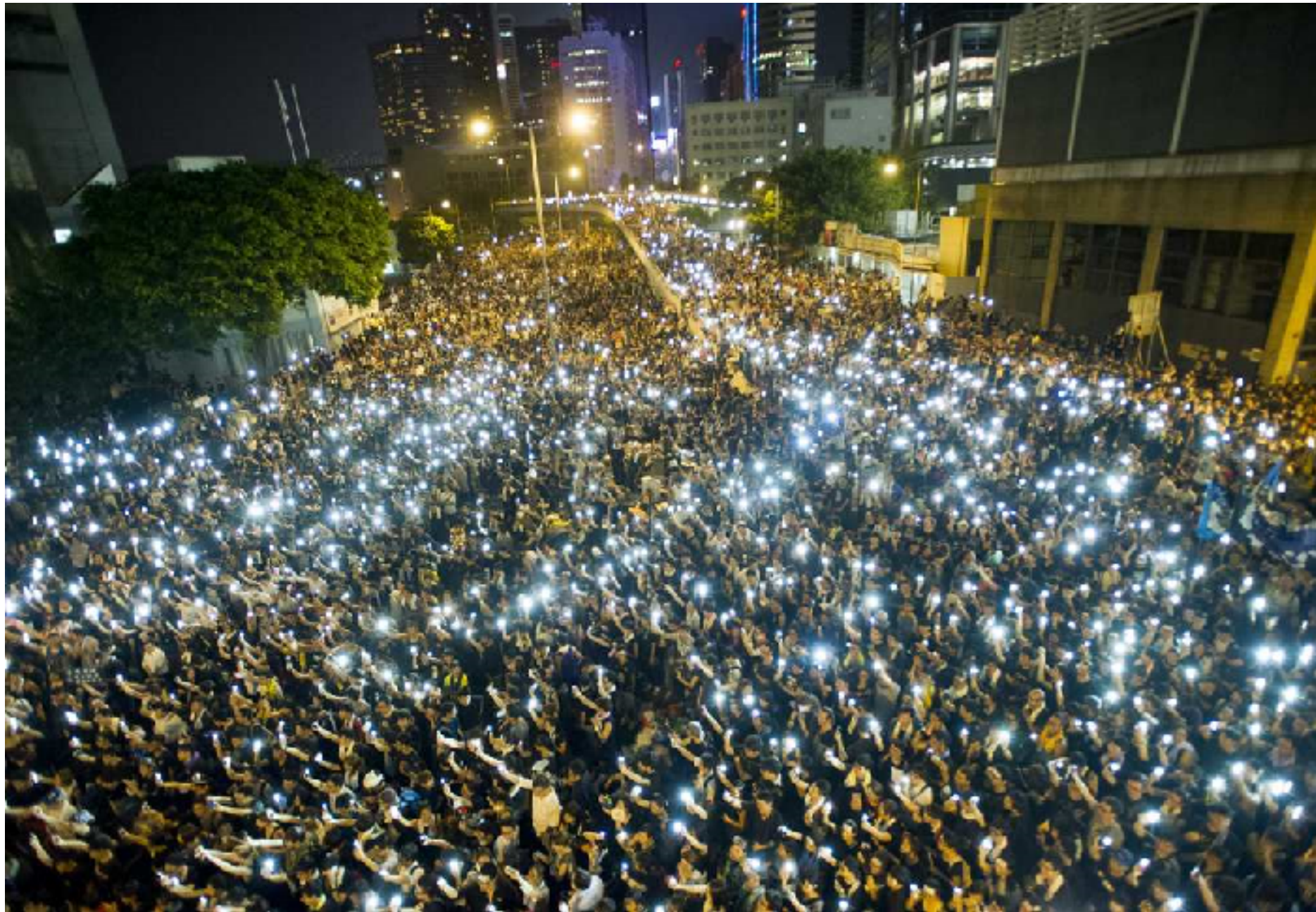
A Classification



A Classification



Arnaud Casteigts, Paola Flocchini, Bernard Mans, Nicola Santoro: Shortest, Fastest, and Foremost Broadcast in Dynamic Networks. Int. J. Found. Comput. Sci. 26(4): 499-522 (2015)



Reliable communication despite Byzantine failures in dynamic networks

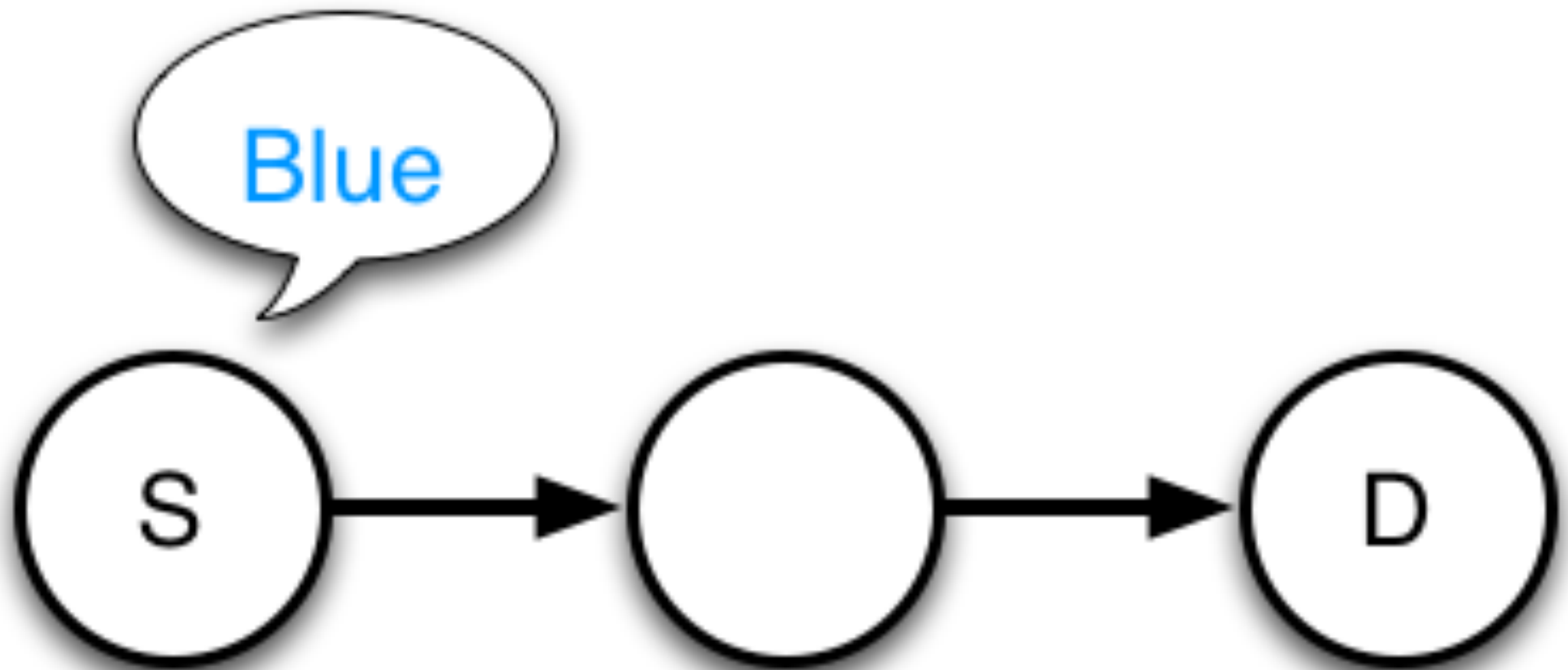
Alexandre Maurer, Sébastien Tixeuil, Xavier Défago: Communicating Reliably in Multihop Dynamic Networks Despite Byzantine Failures. SRDS 2015: 238-245

Context

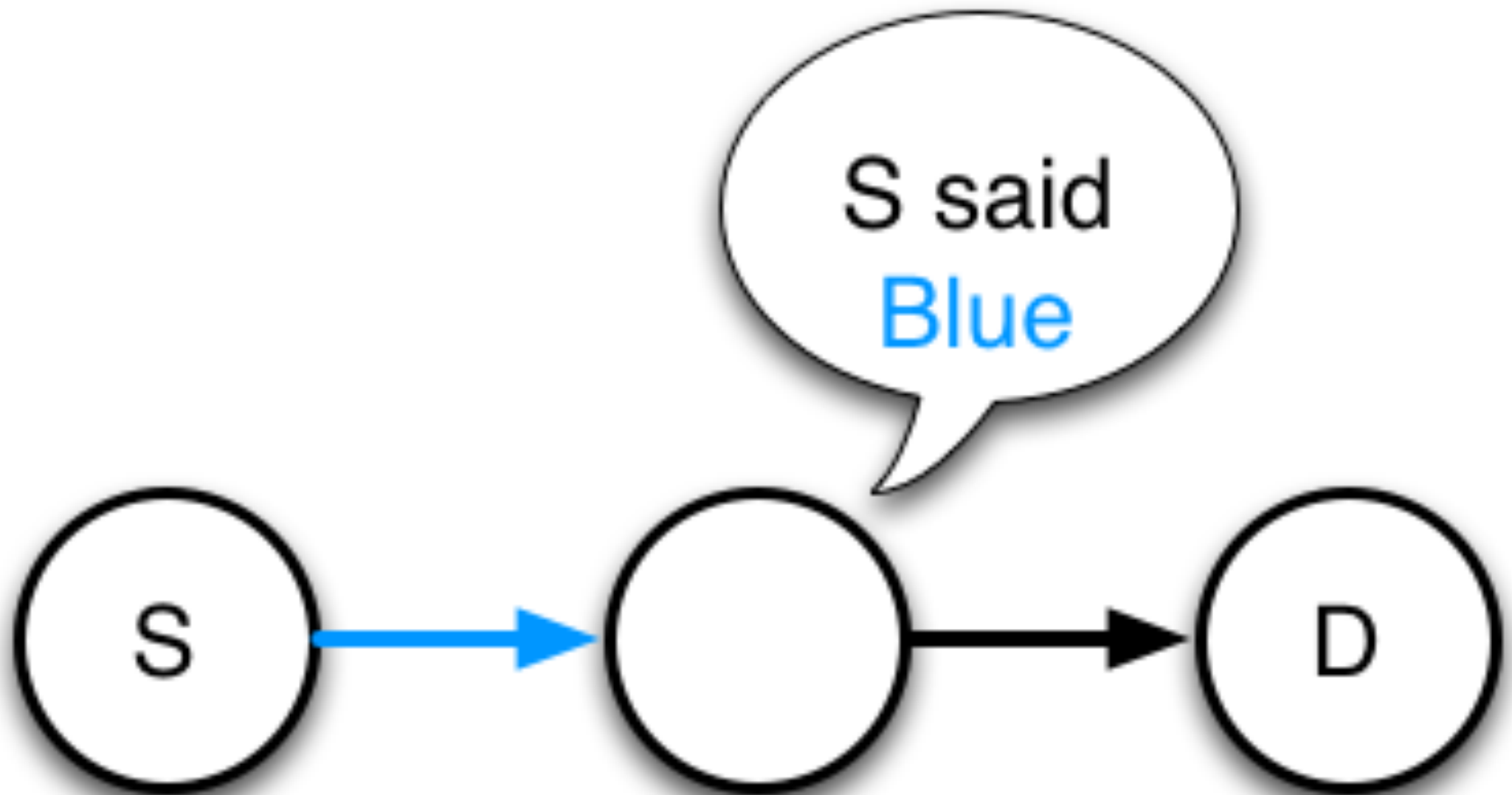


Information broadcast in multi hop networks

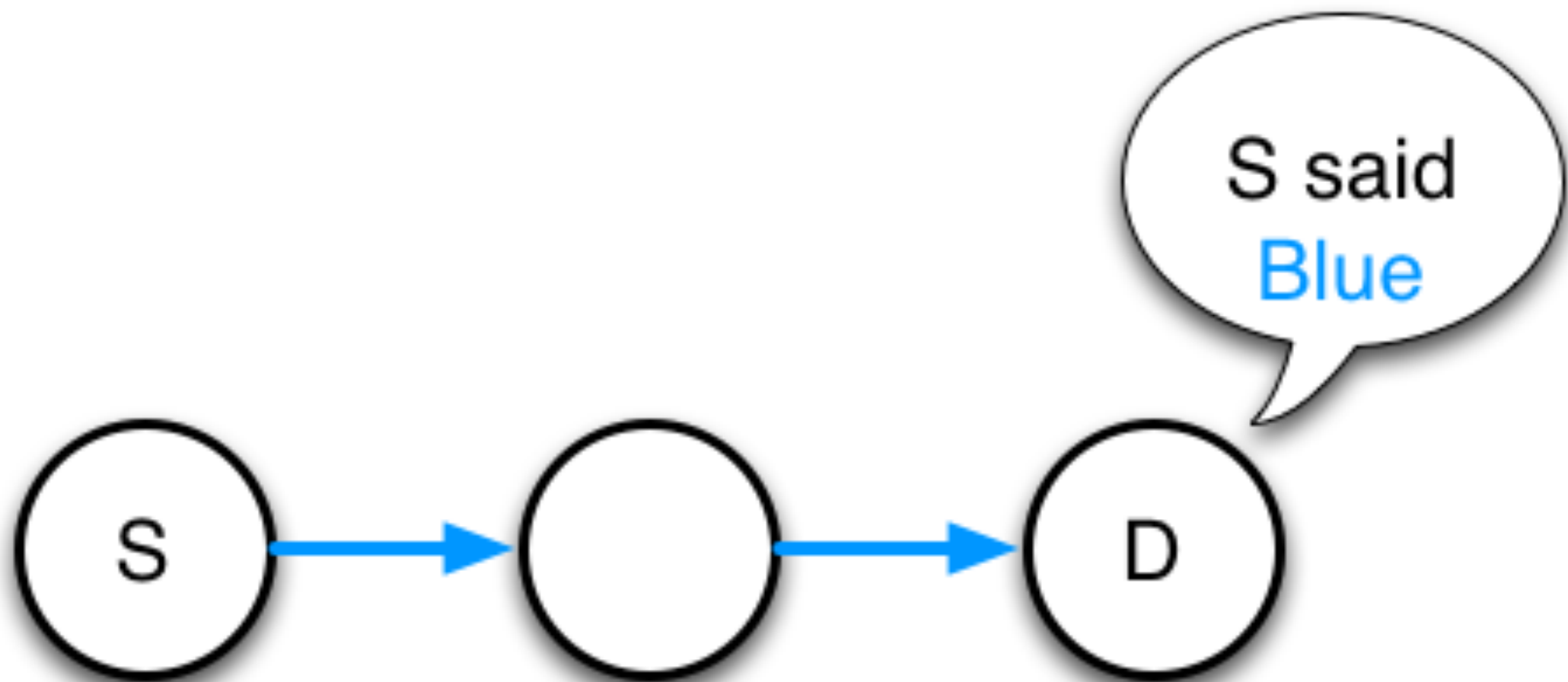
Example



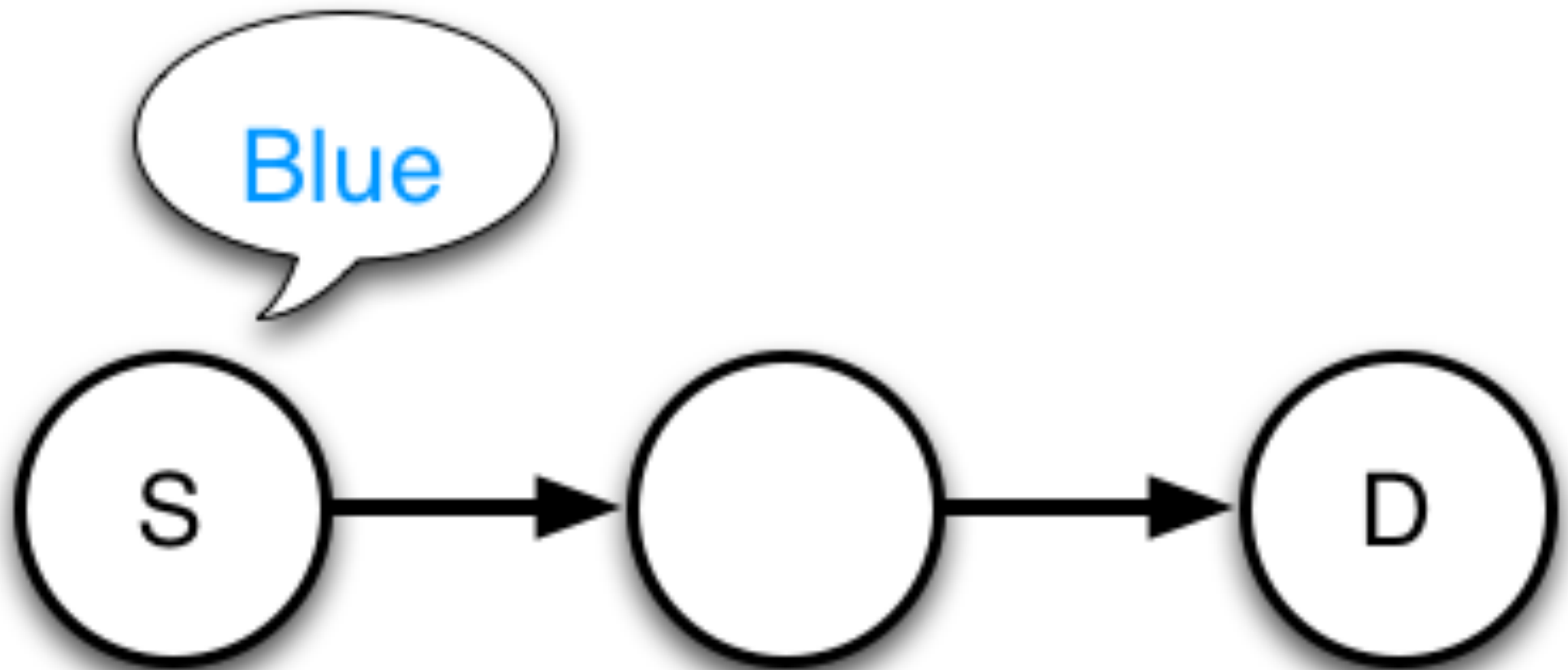
Example



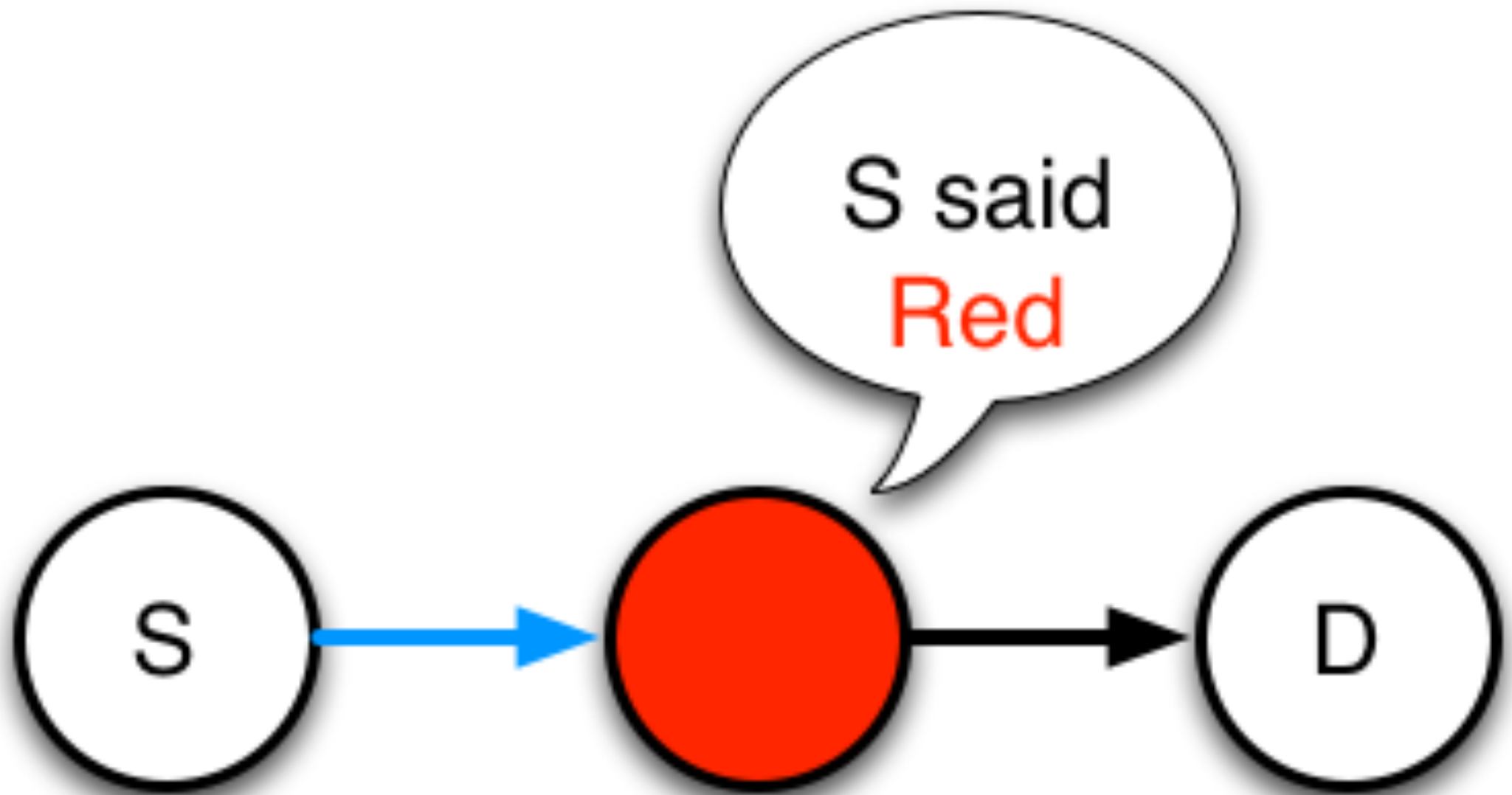
Example



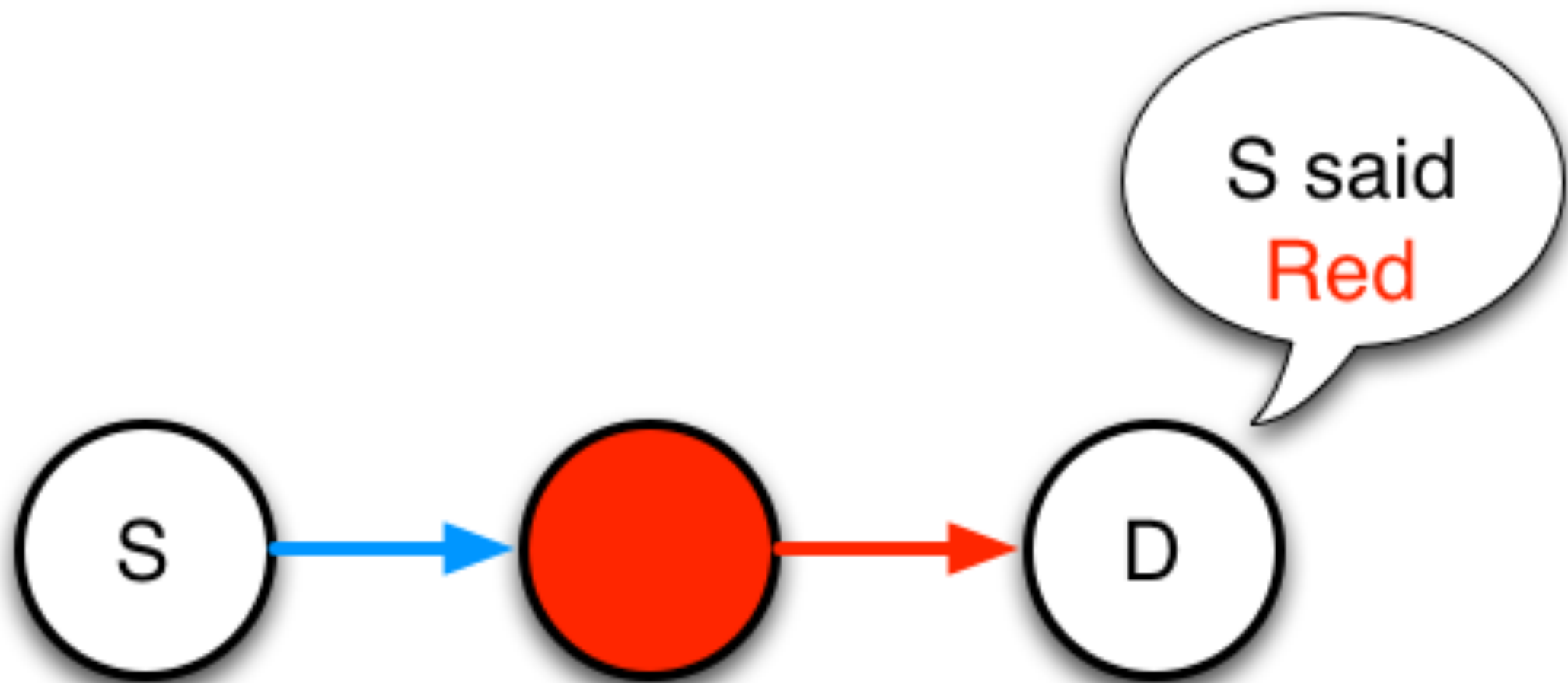
Example



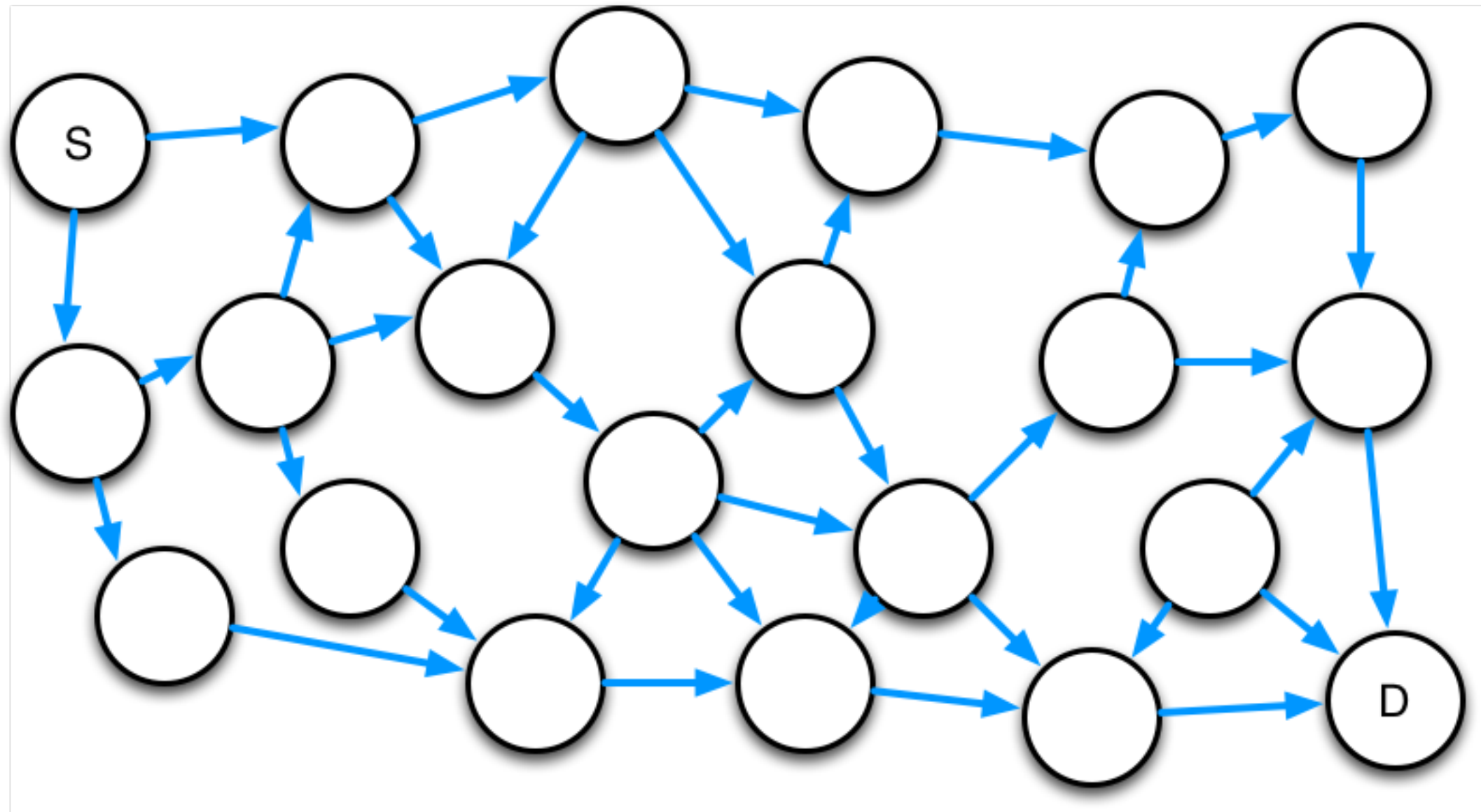
Example



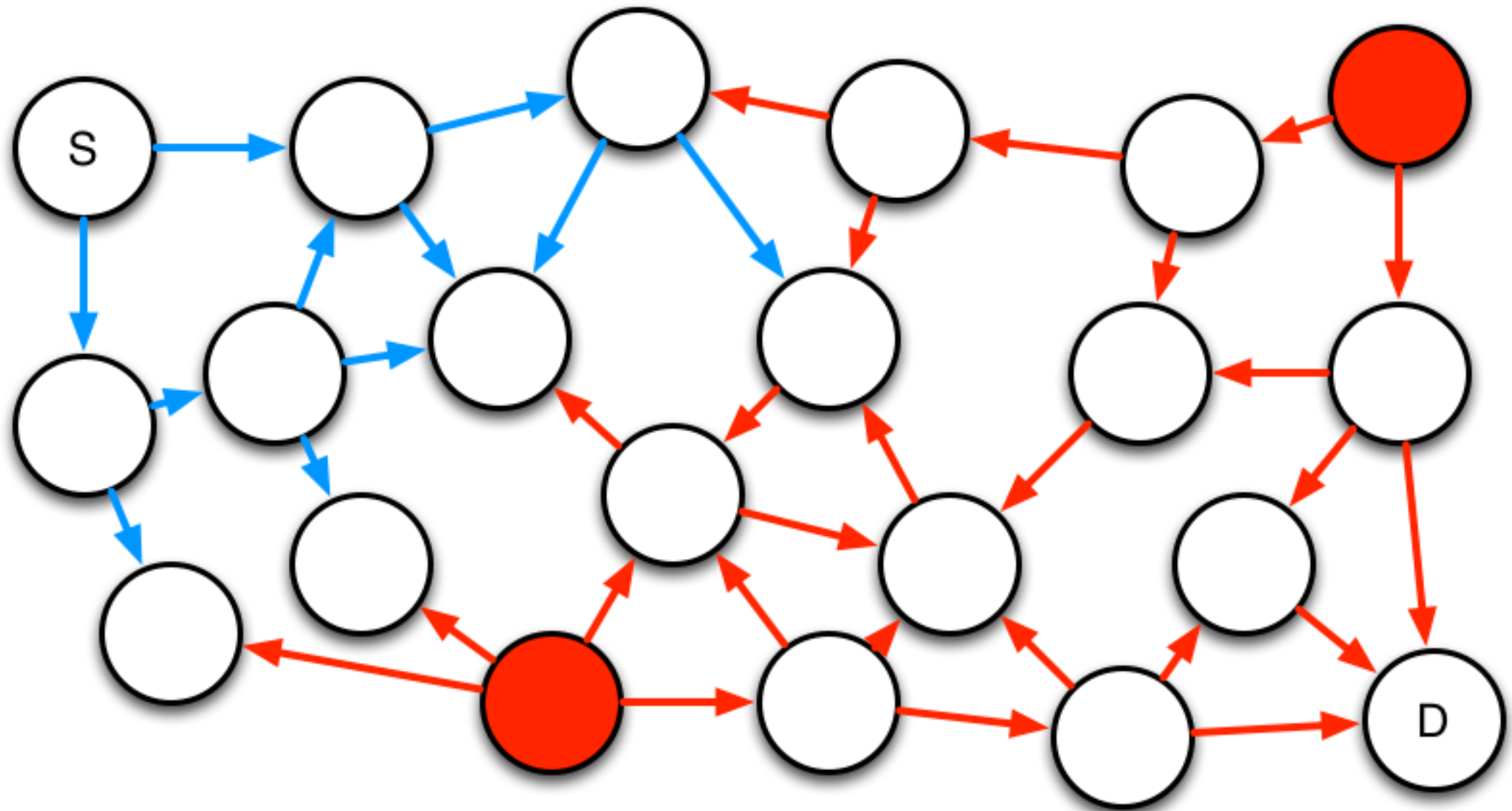
Example



Information Broadcast



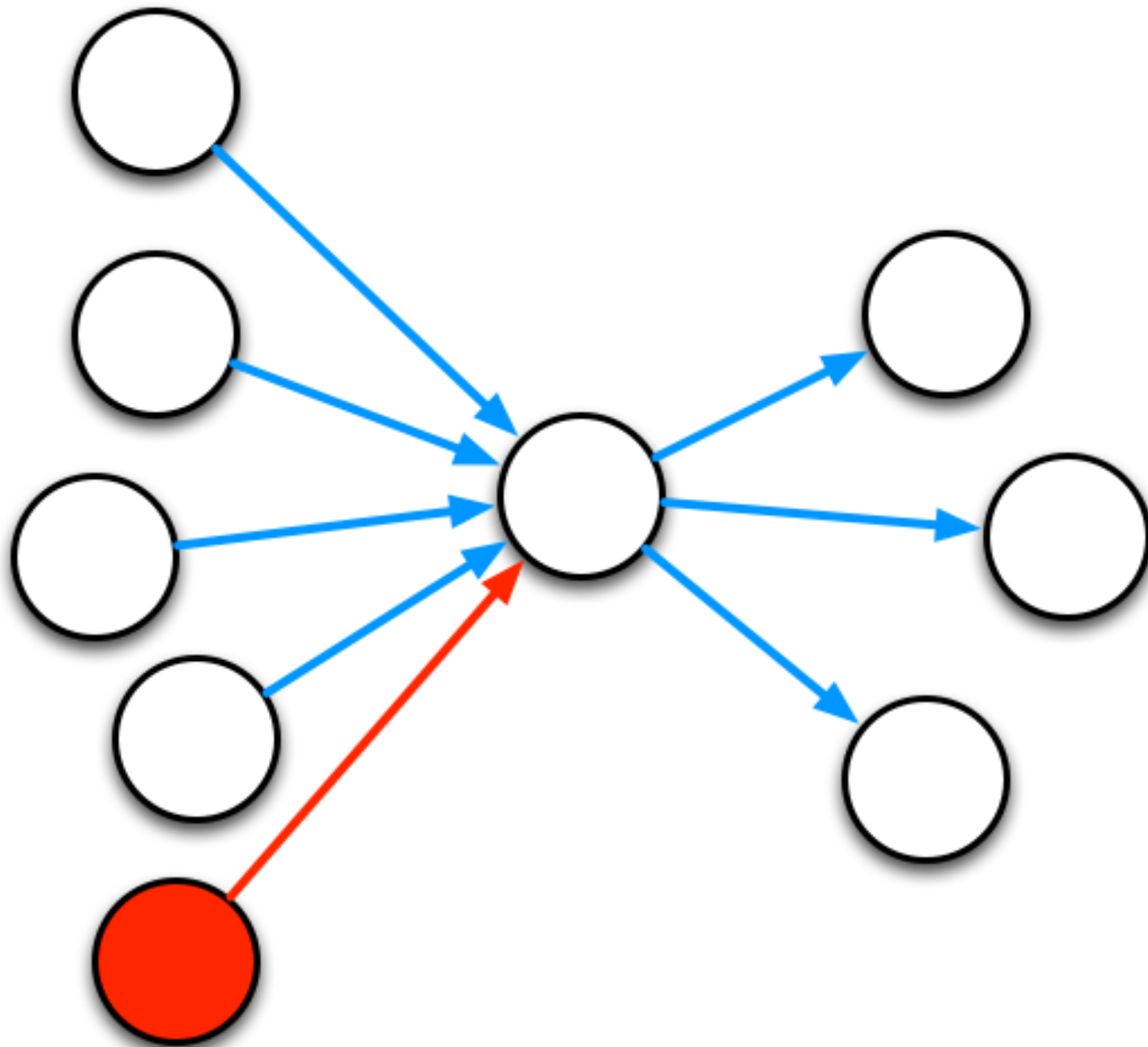
Information Broadcast



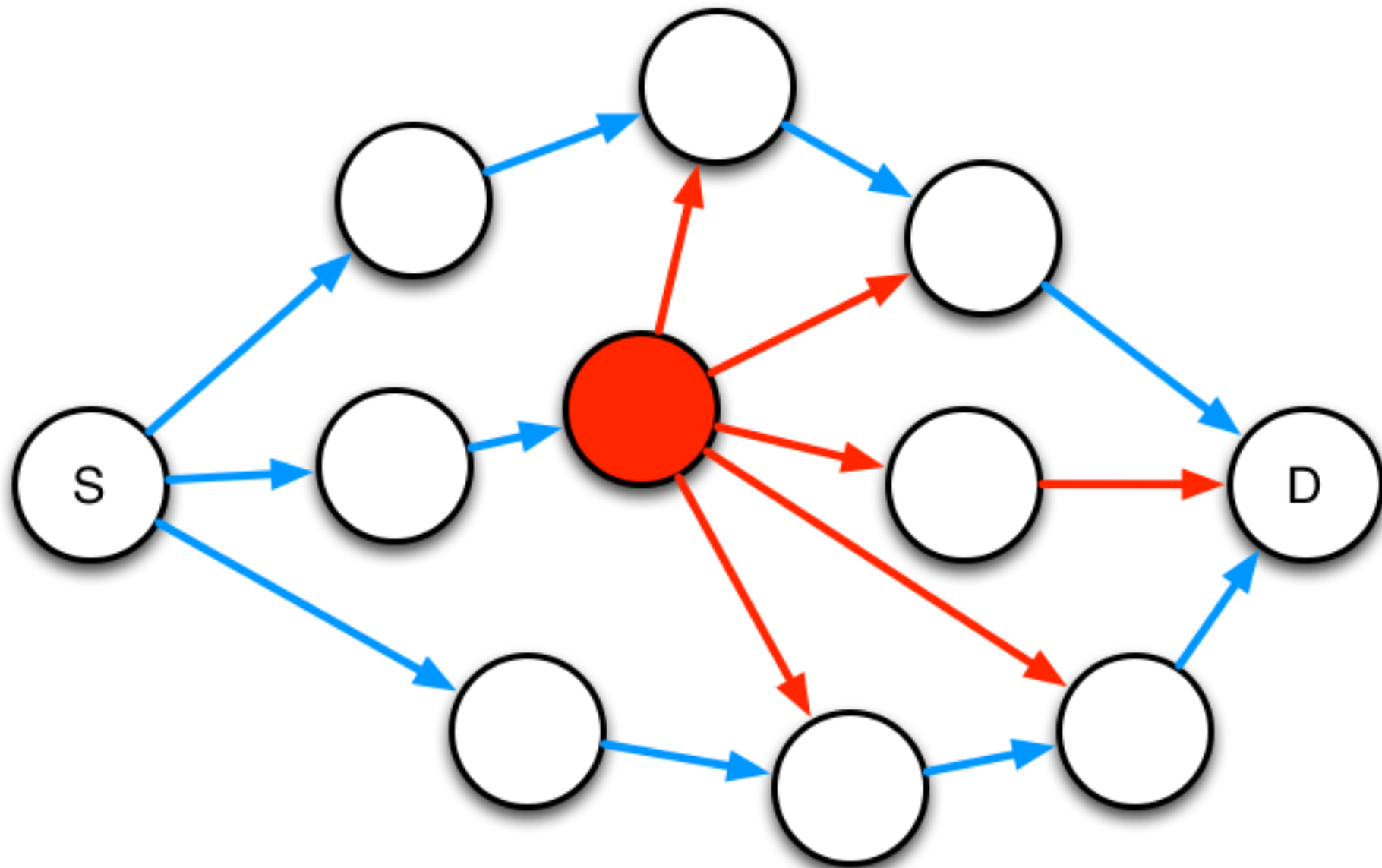
Objective

- **Broadcast** algorithms resilient to **Byzantine** Failures
 - No **false** message ever accepted
 - **Correct** messages always received

Local Vote



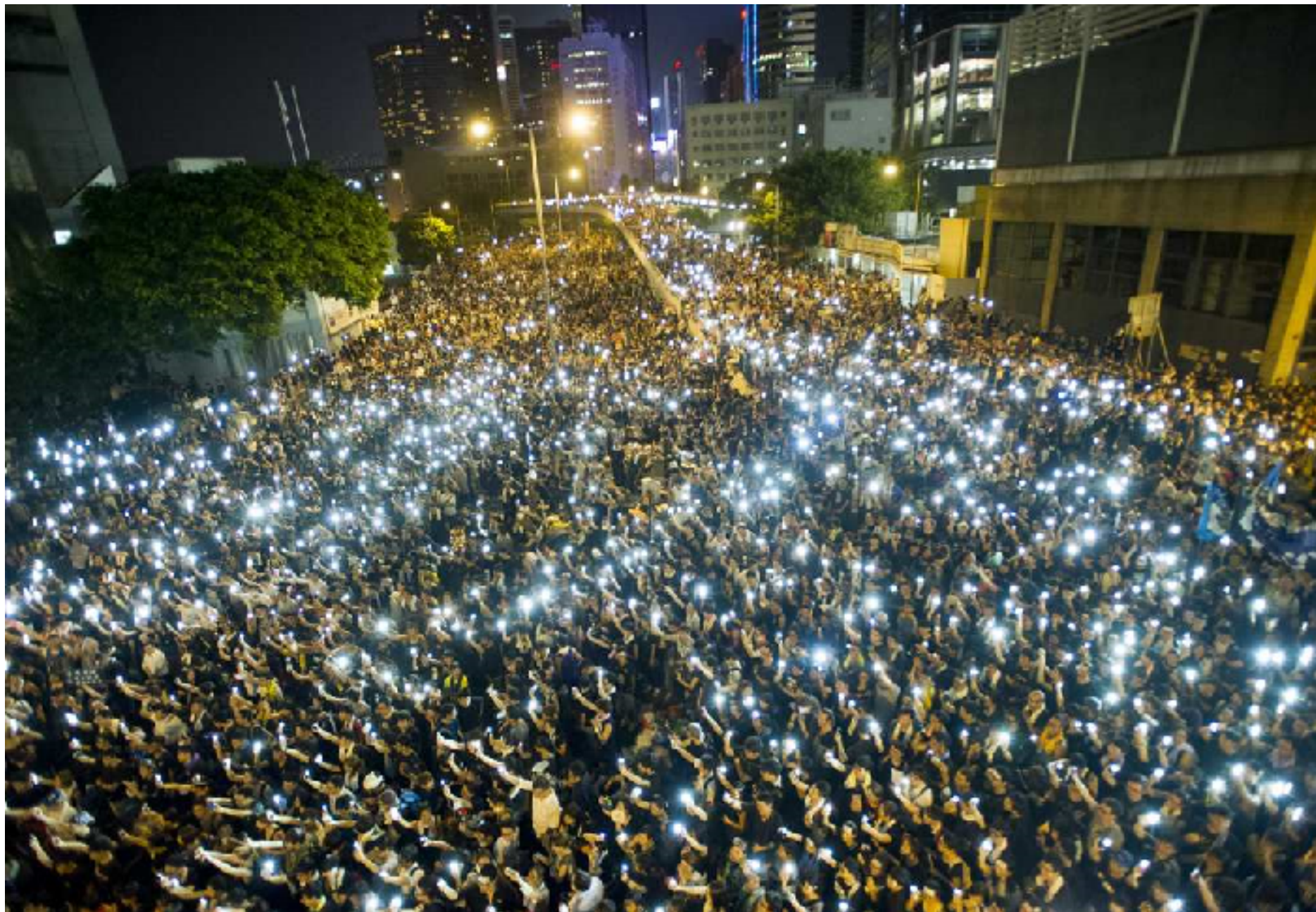
Vote on Multiple Paths



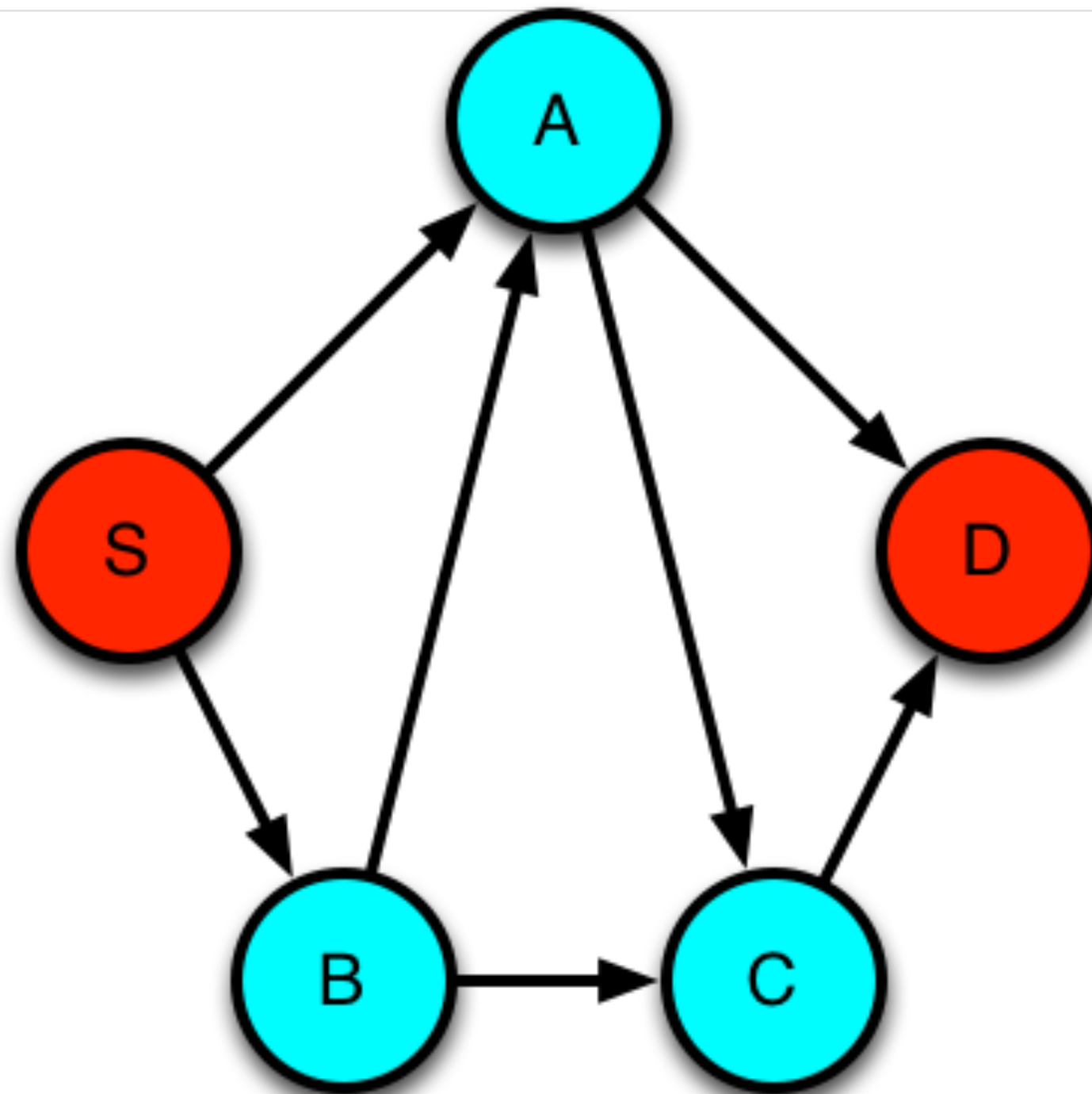
Condition for reliable communication in static networks

- k = number of Byzantine nodes
- **Condition:** $2k+1$ node-disjoint paths between the source and the destination

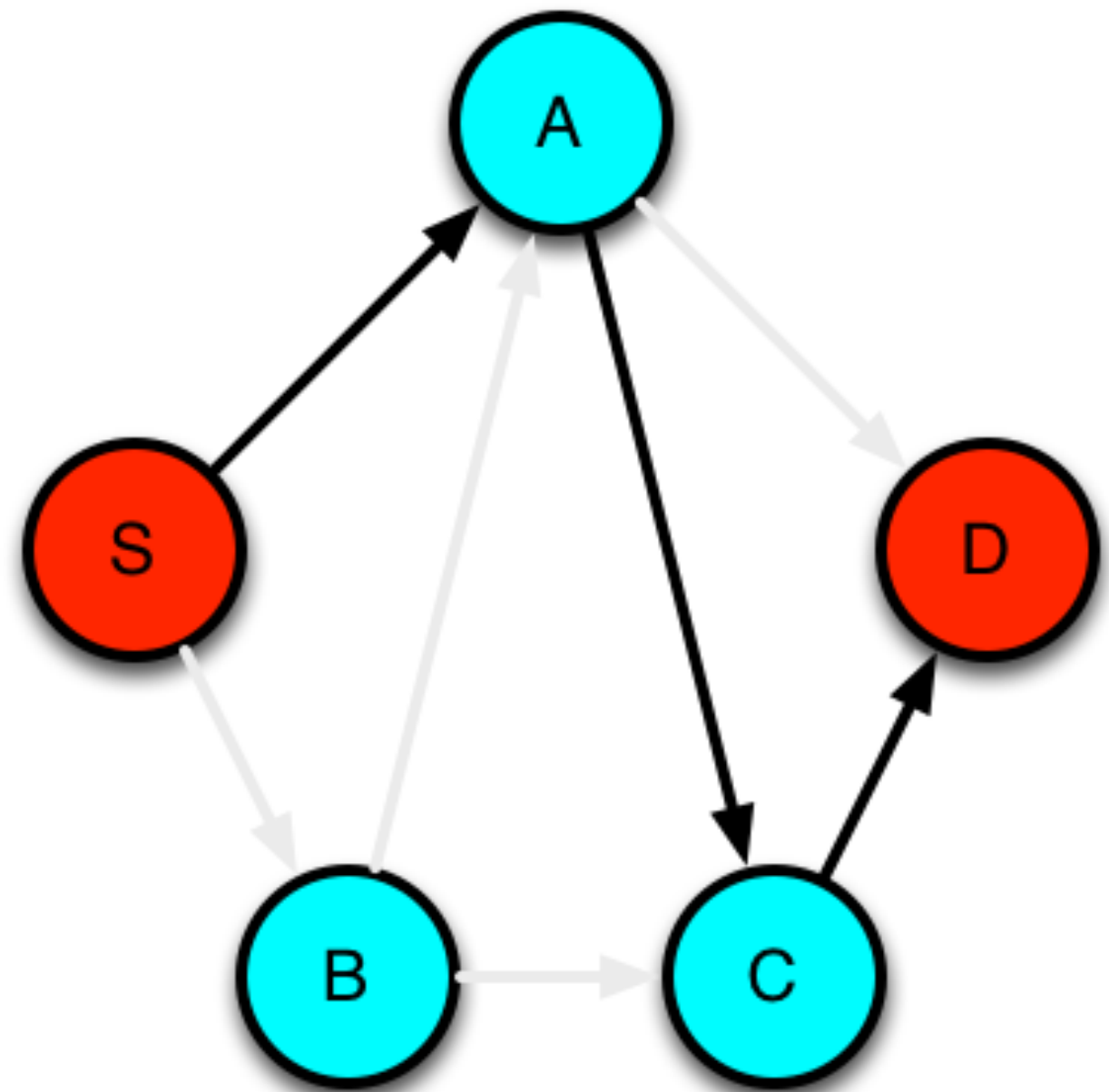
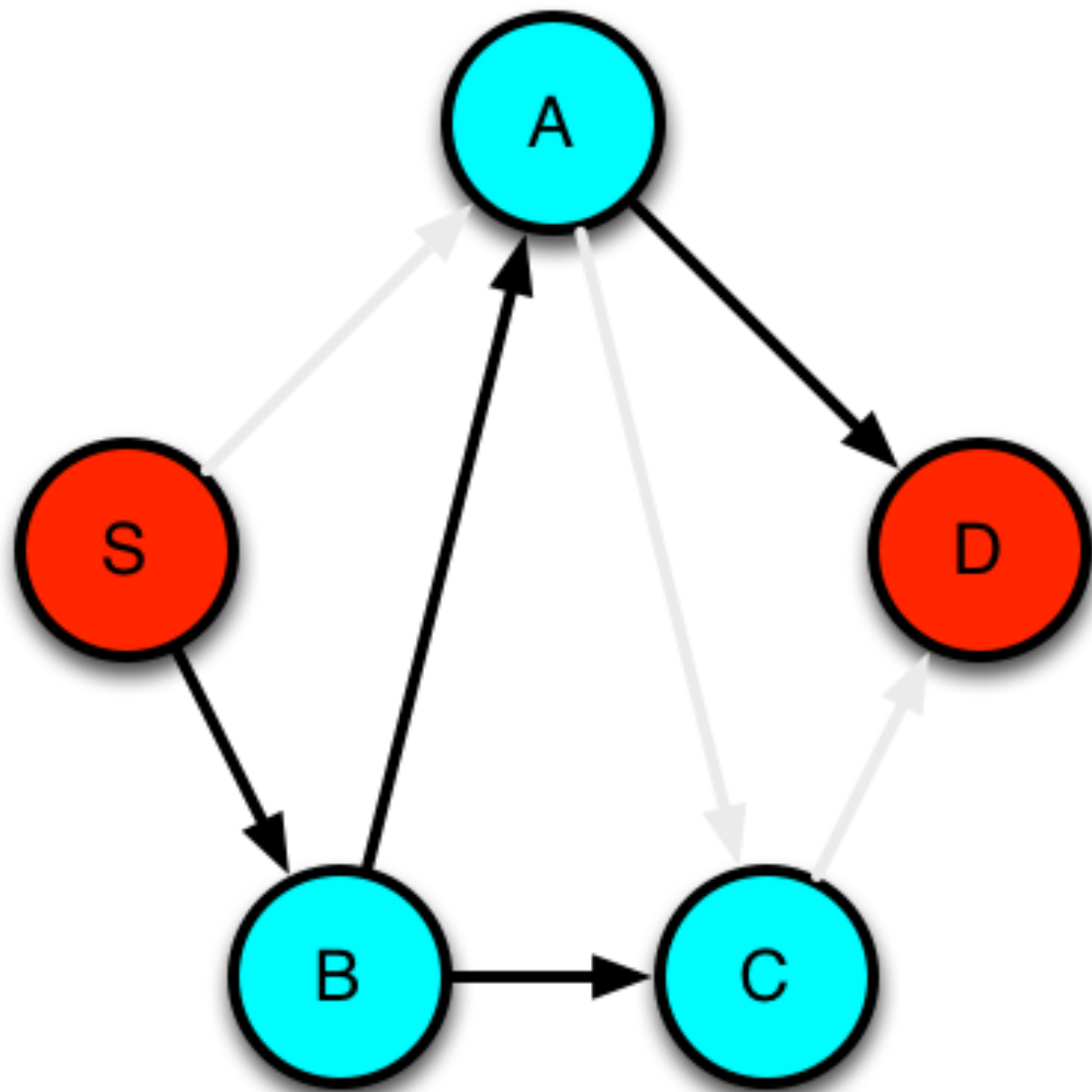
Enter Dynamic Networks



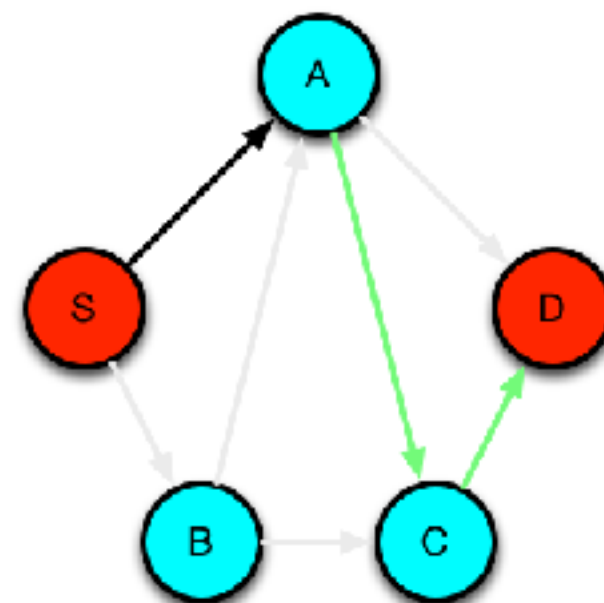
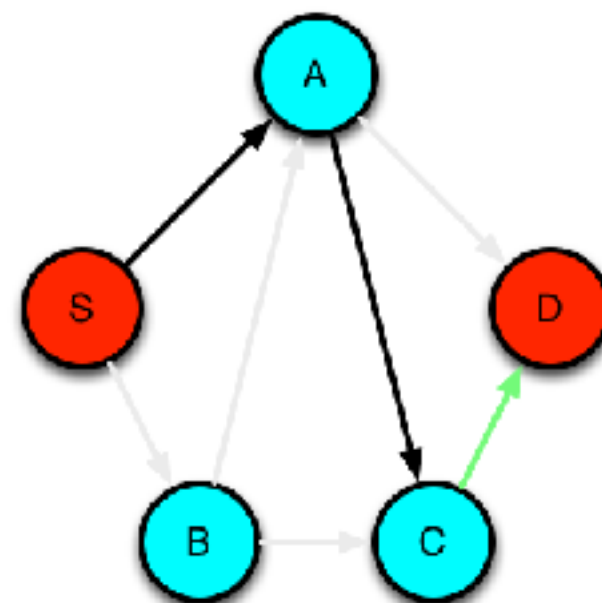
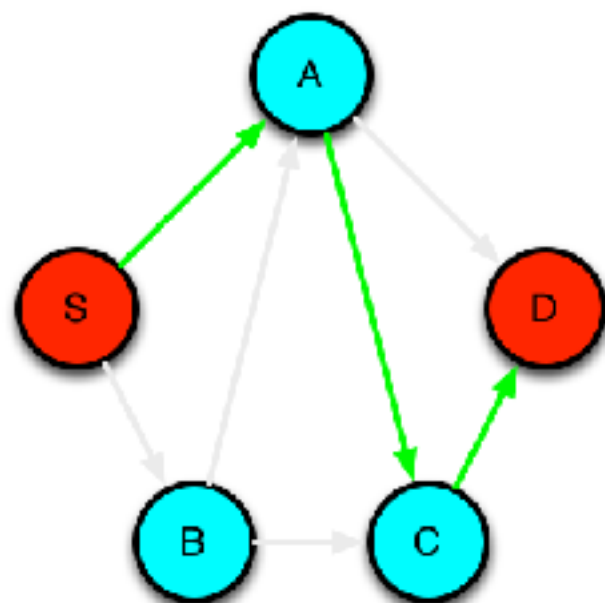
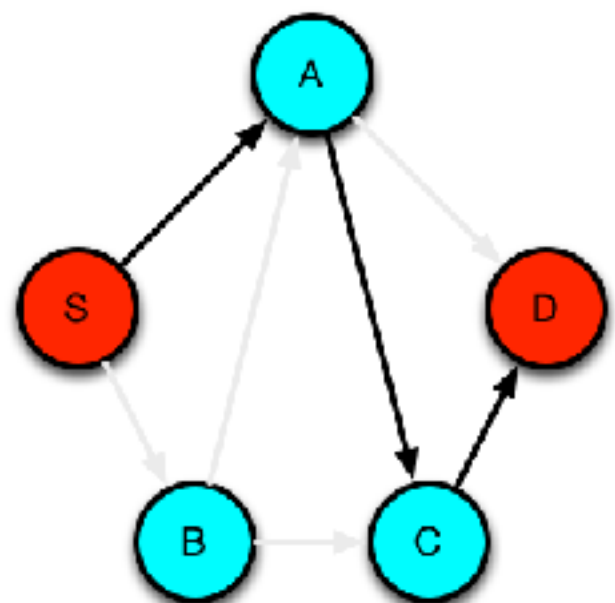
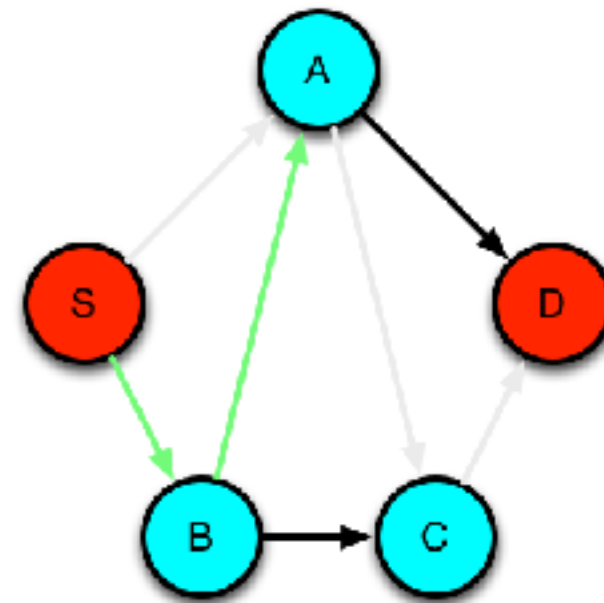
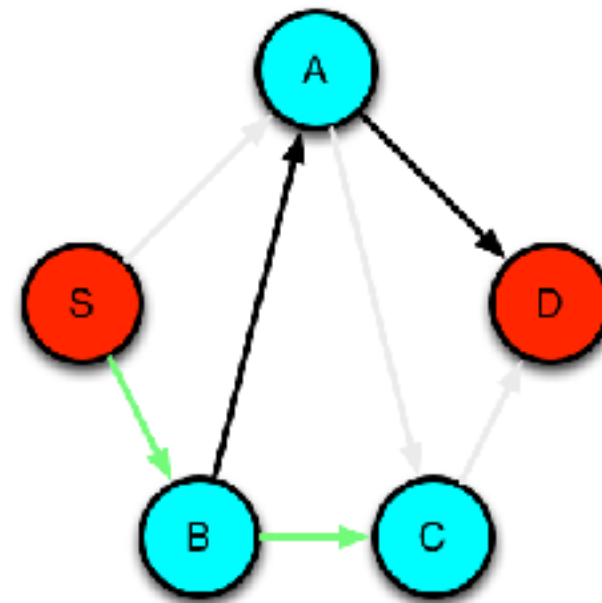
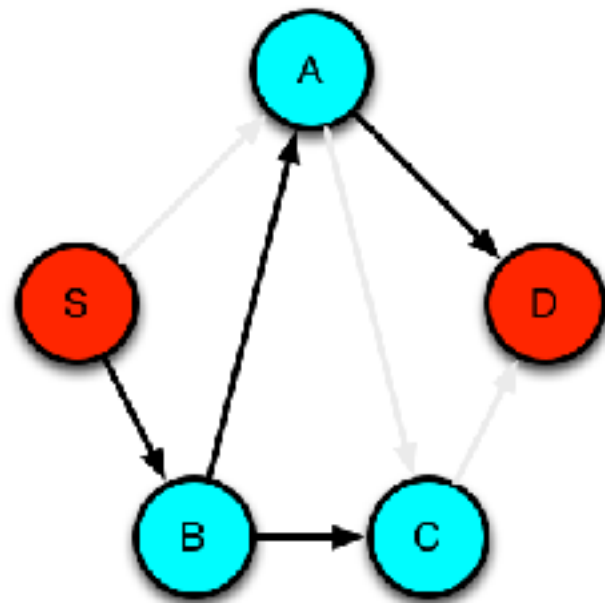
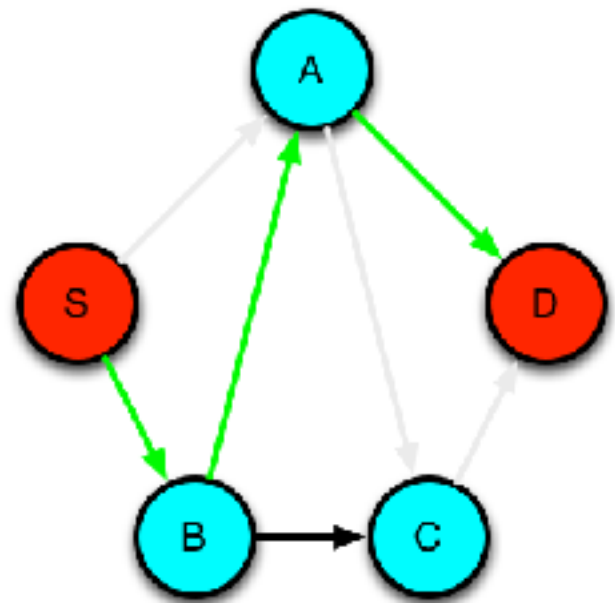
Menger's Theorem



Menger's Theorem



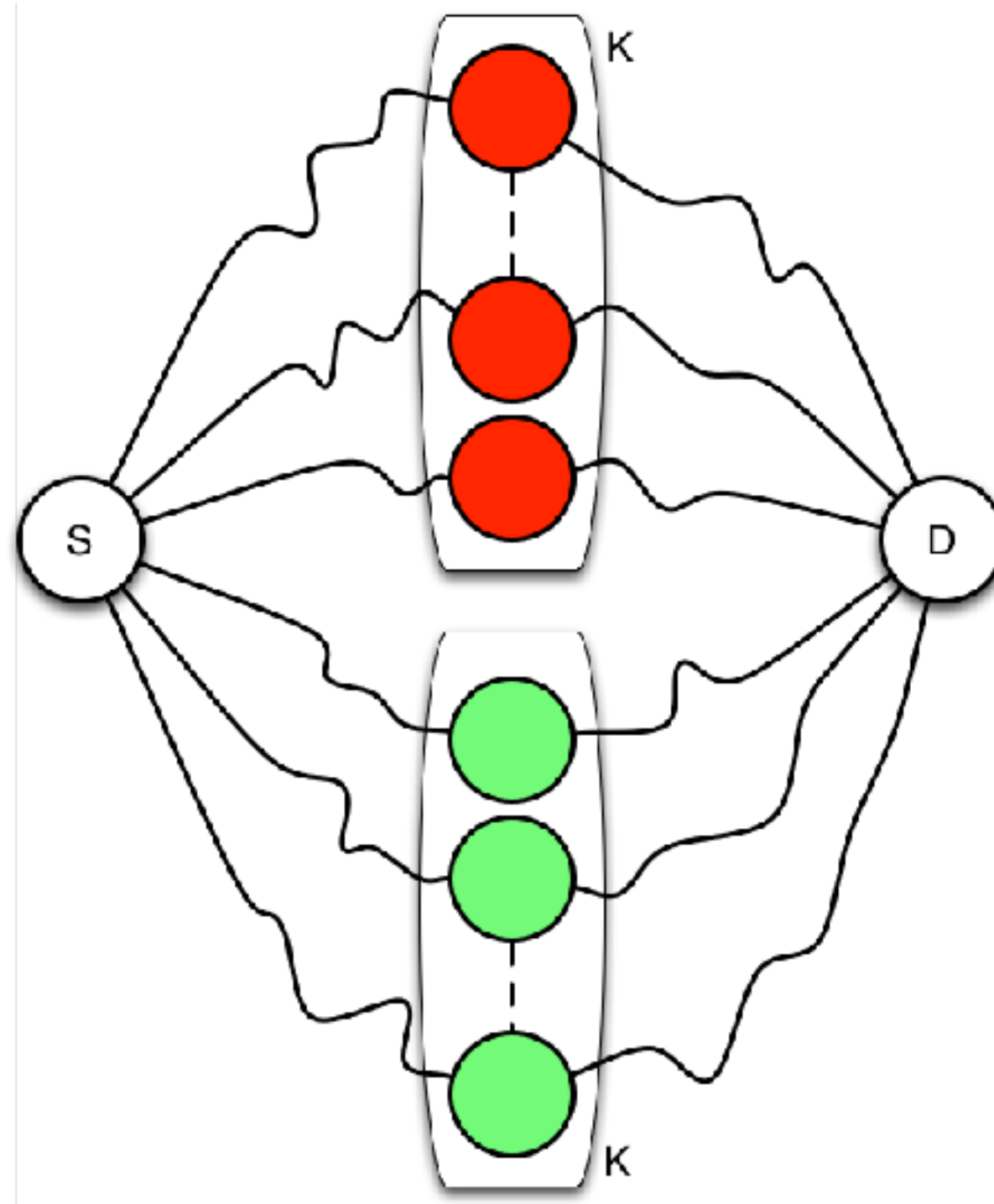
Menger's Theorem



Condition in Dynamic Networks

- k =number of Byzantine nodes
- **Condition:** $2k+1$ nodes must be removed to cut all dynamic paths

Necessary Condition



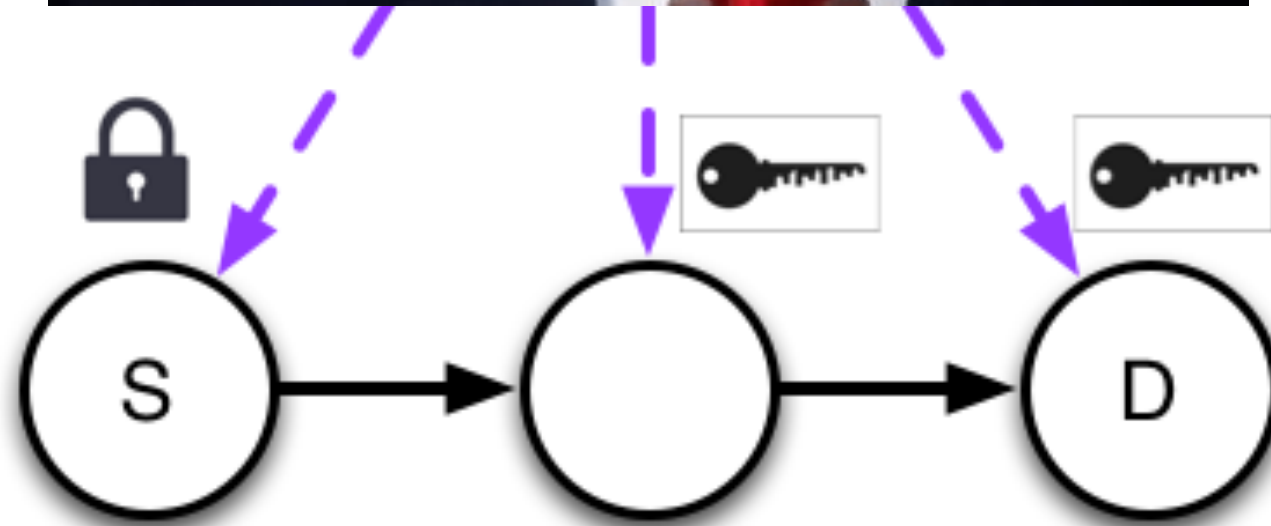
Sufficient Condition

- Send the message through *all* journeys
- Register the journeys
- When a set of journeys that cannot be cut by $2k$ nodes is collected, accept the message

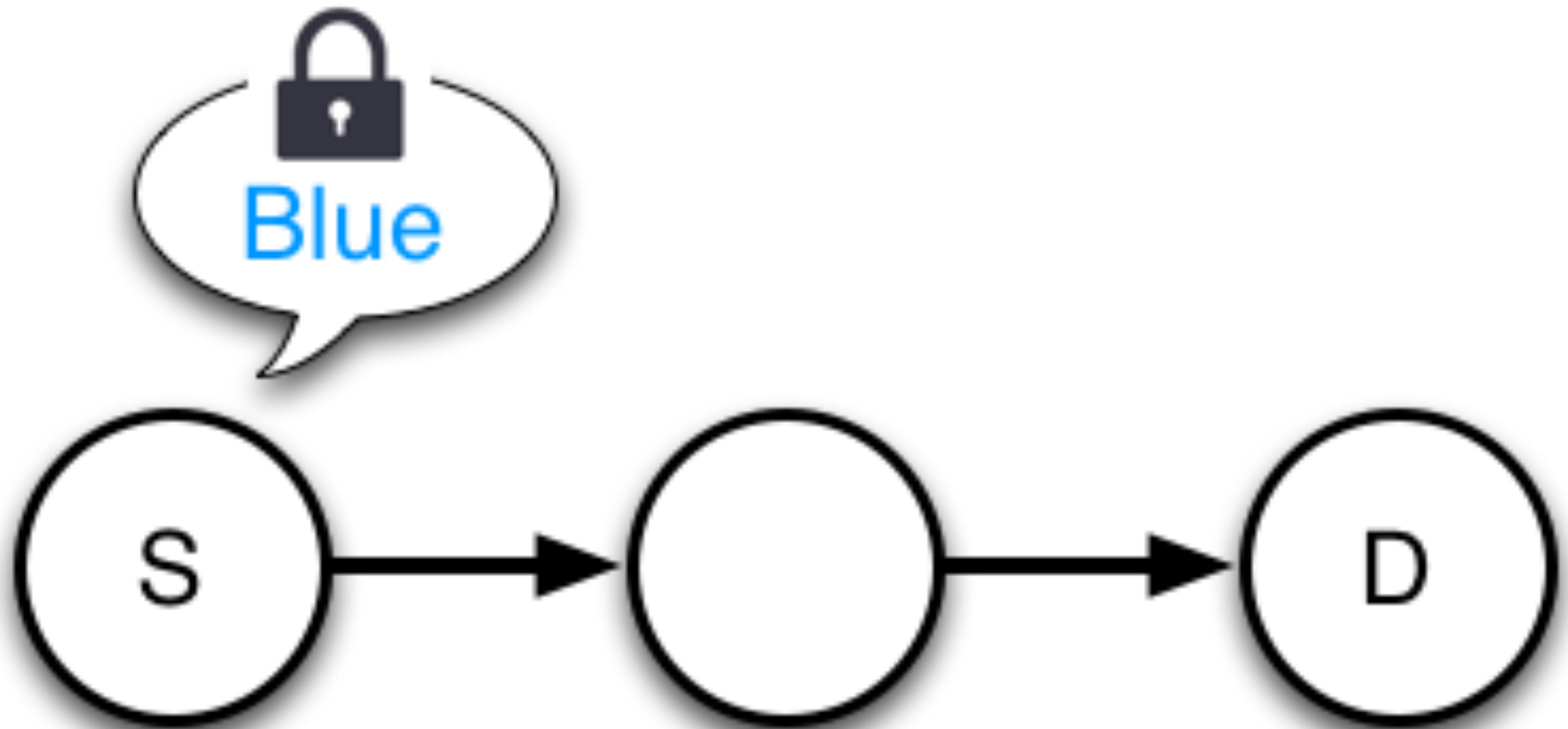
Why not Cryptography?



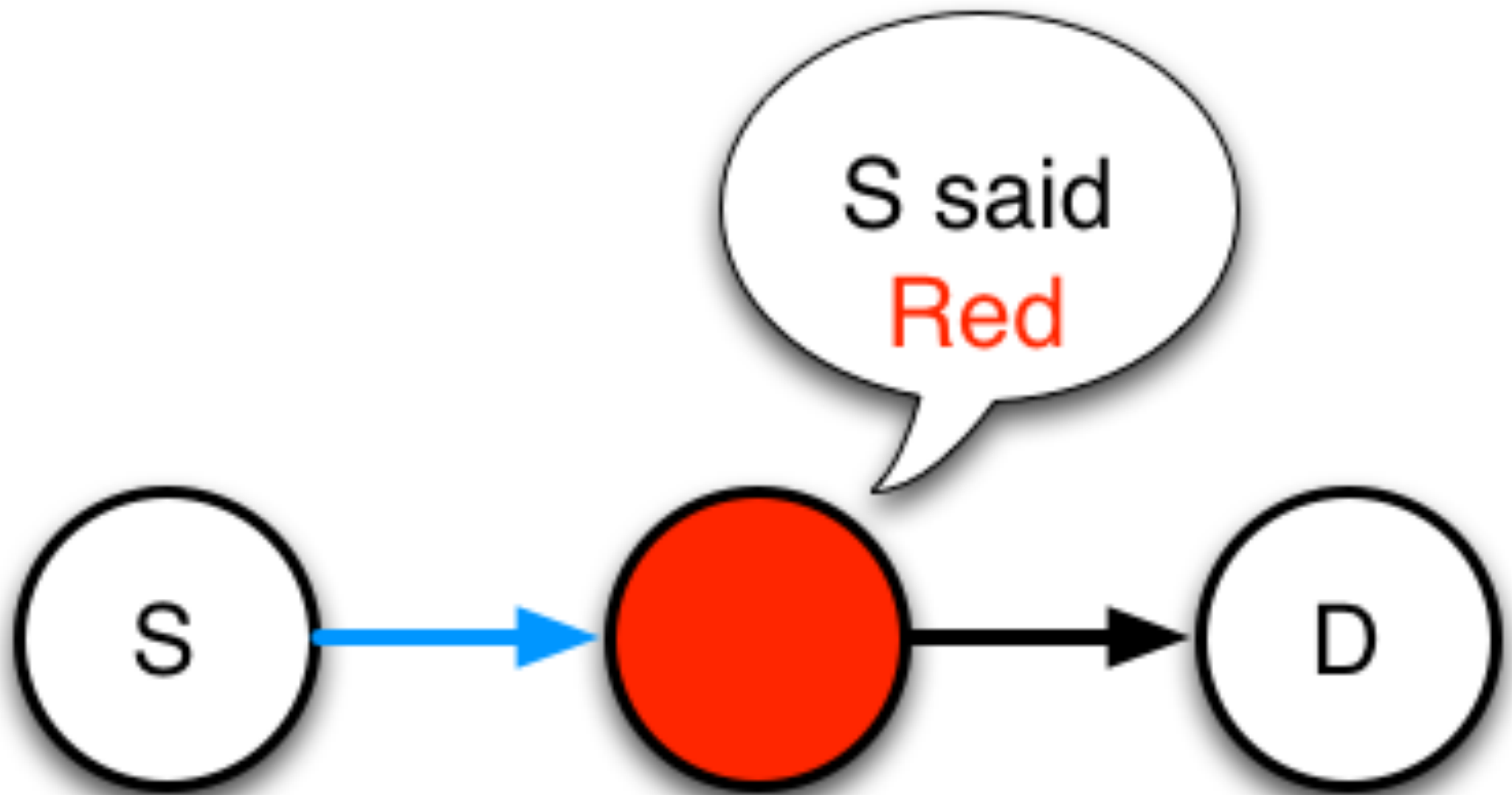
PKI



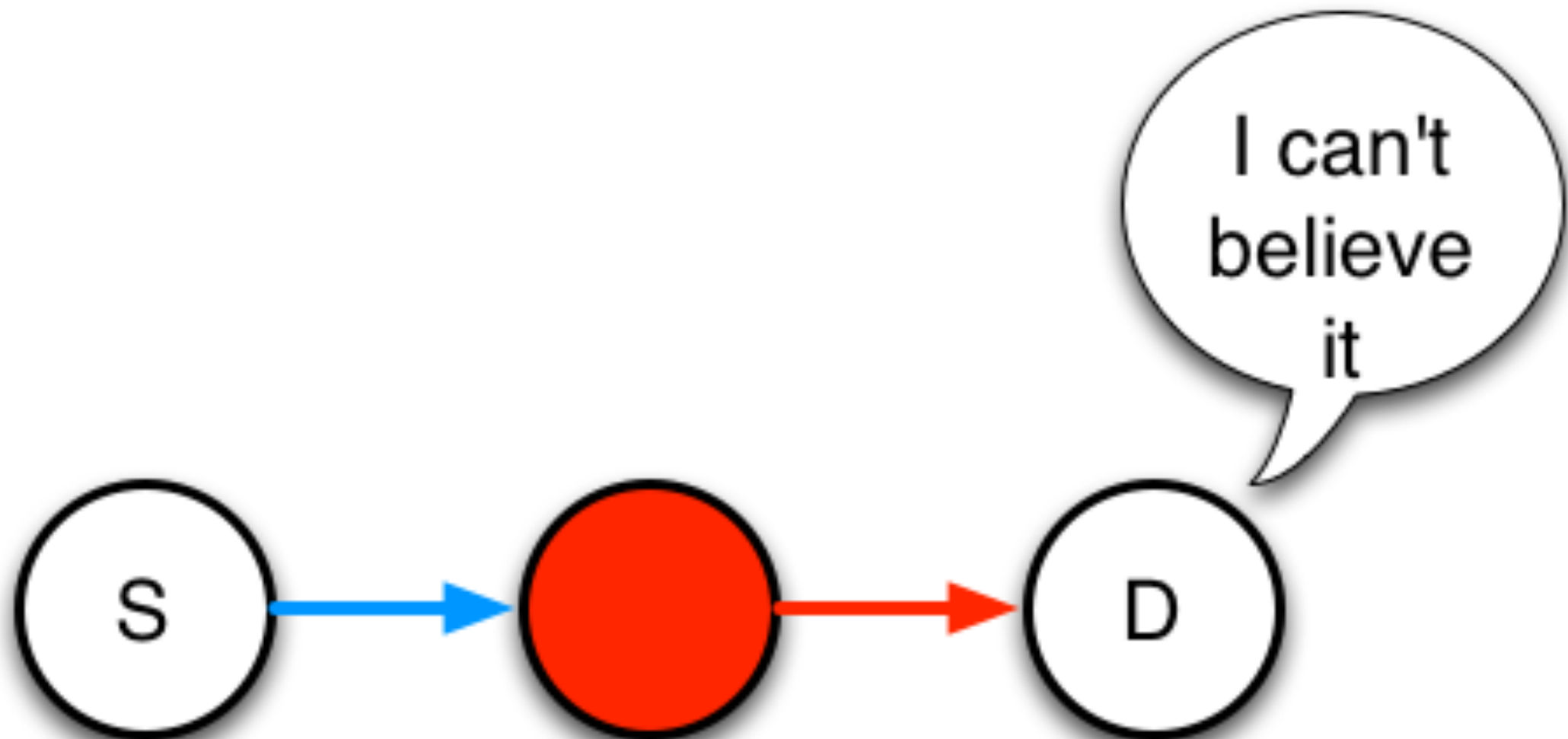
Example



Example



Example



Trusted third party



Trusted keys



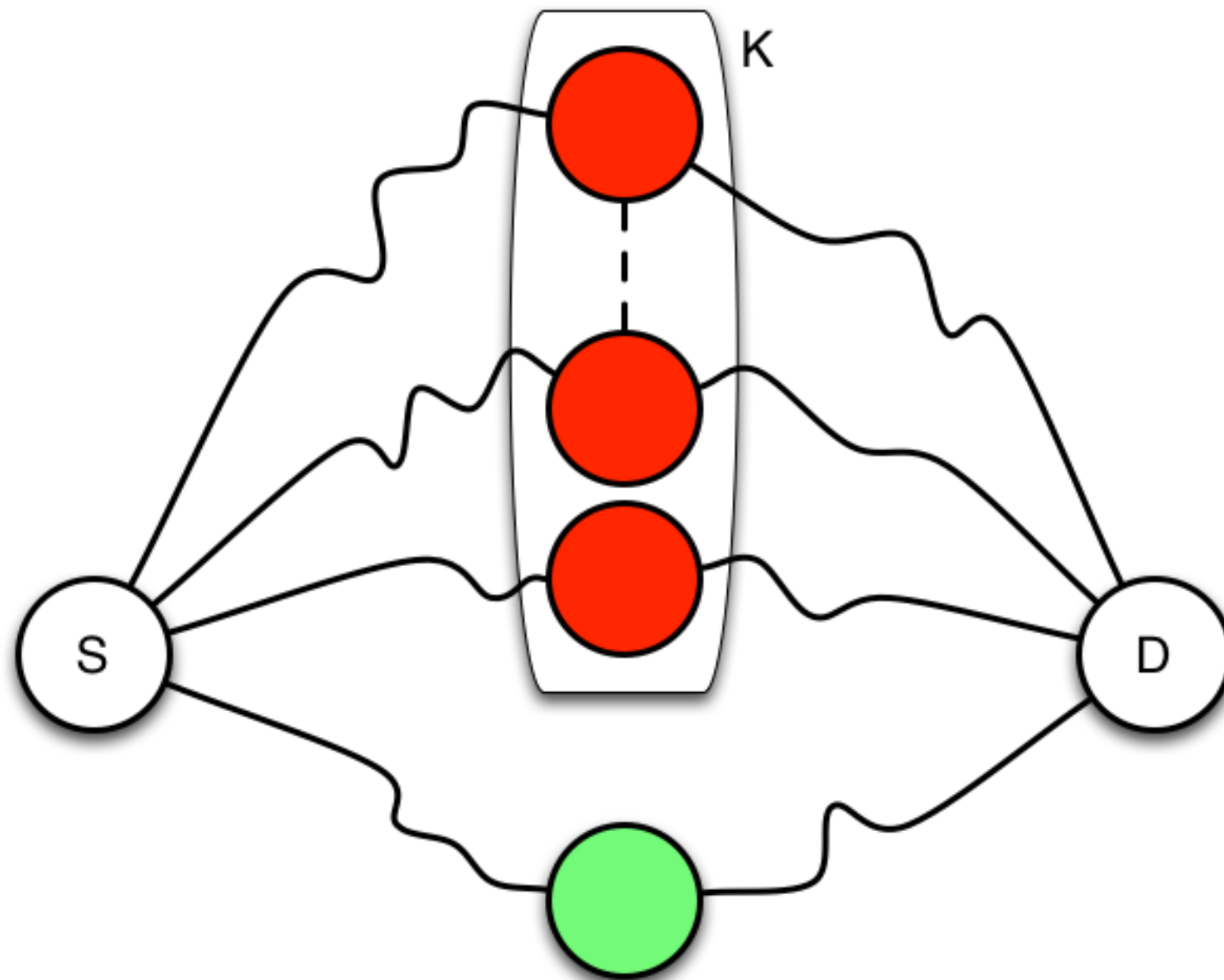
Trusted Software



Condition in Dynamic Networks with Cryptography

- k = number of Byzantine nodes
- **Condition:** $k+1$ nodes must be removed to cut all journeys

Necessary Condition with Cryptography



Sufficient Condition with Cryptography

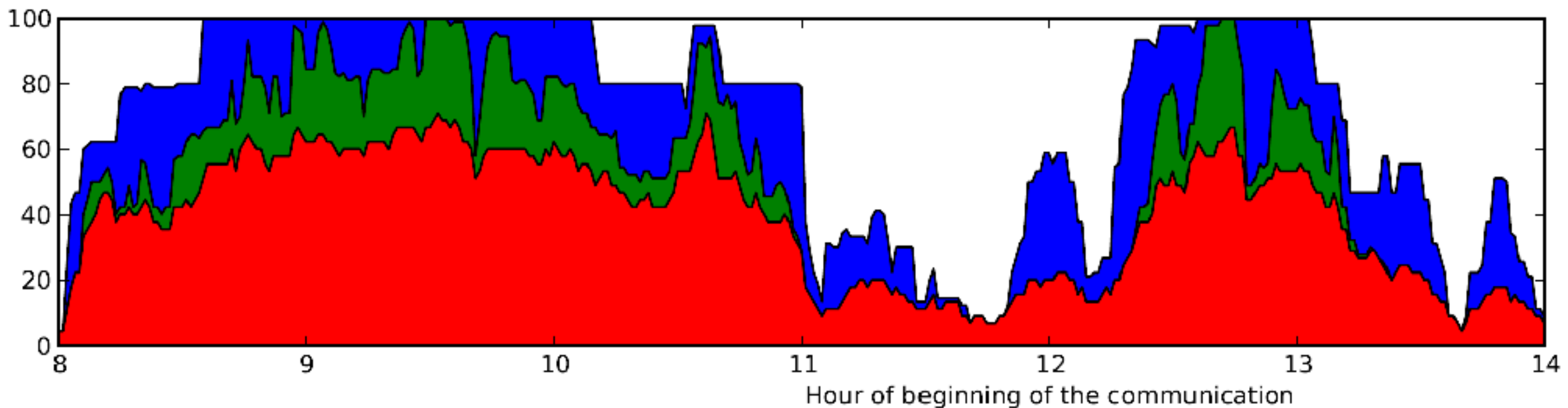
- Send the message through all journeys
- When a cryptographically acceptable message arrives, accept it

Case Studies

- Participants in a conference
- Agents in the subway

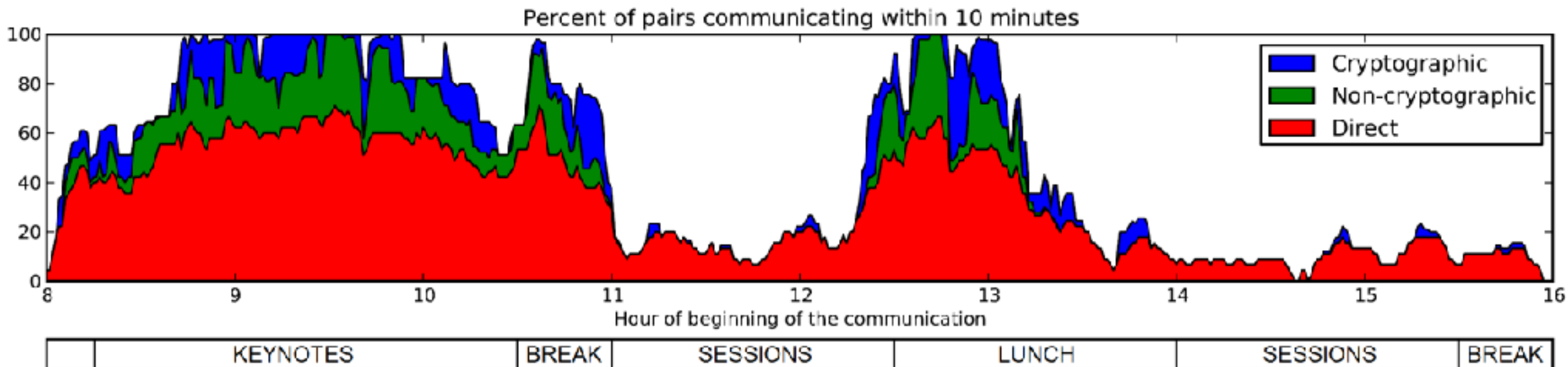
Participants Interacting in a Conference

Percent of pairs communicating within 10 minutes



- Unreliable multihop communication
- Reliable multi hop communication
- Direct communication

Participants Interacting in a Conference

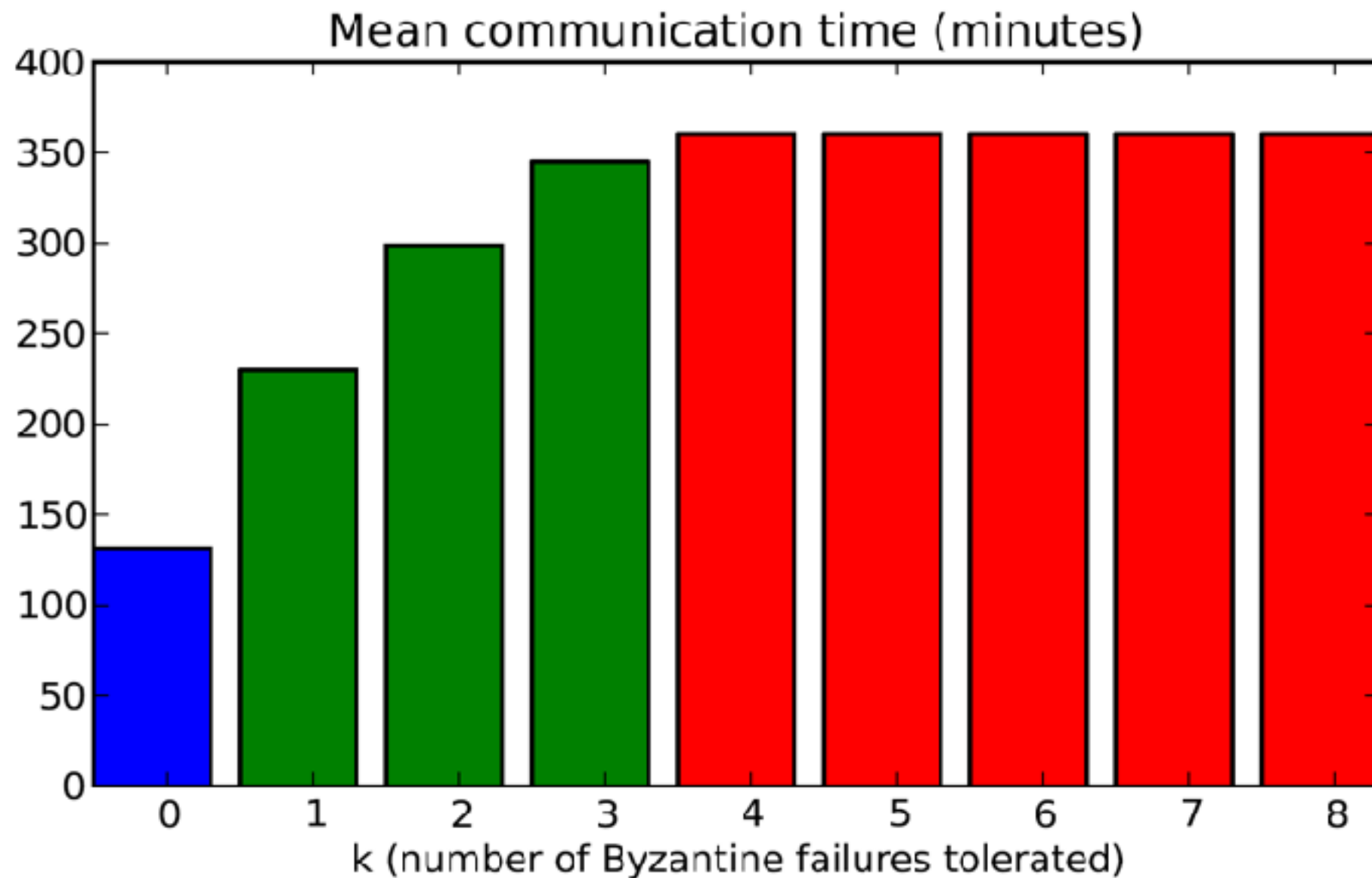


- Cryptographic multihop communication
- Non-Cryptographic multi hop communication
- Direct communication

Paris Subway Users



Paris Subway Users

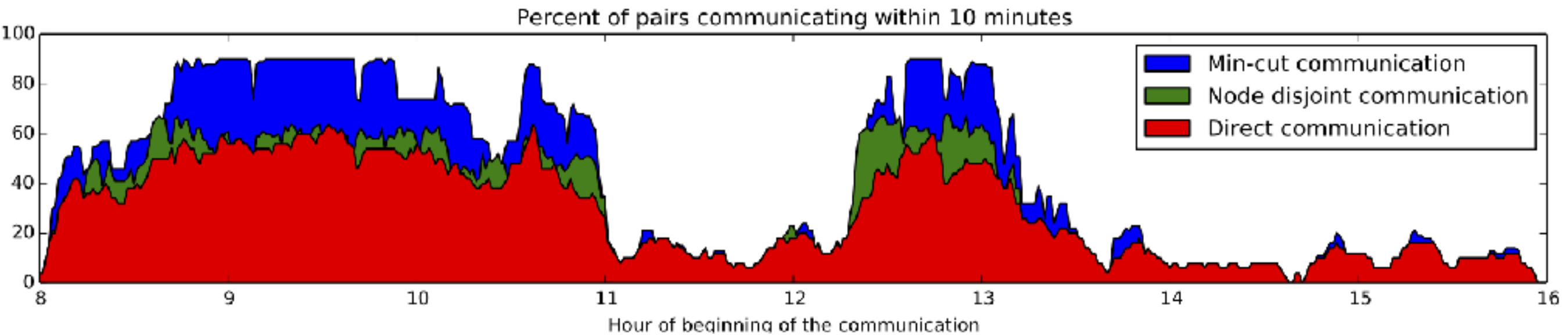


- Unreliable multihop communication
- Reliable multi hop communication
- Direct communication

Conclusion

- Classical node-disjoint paths cannot be used as « if and only if » condition
- Falling back to (dynamic) Min-cut works

IF vs. IFF

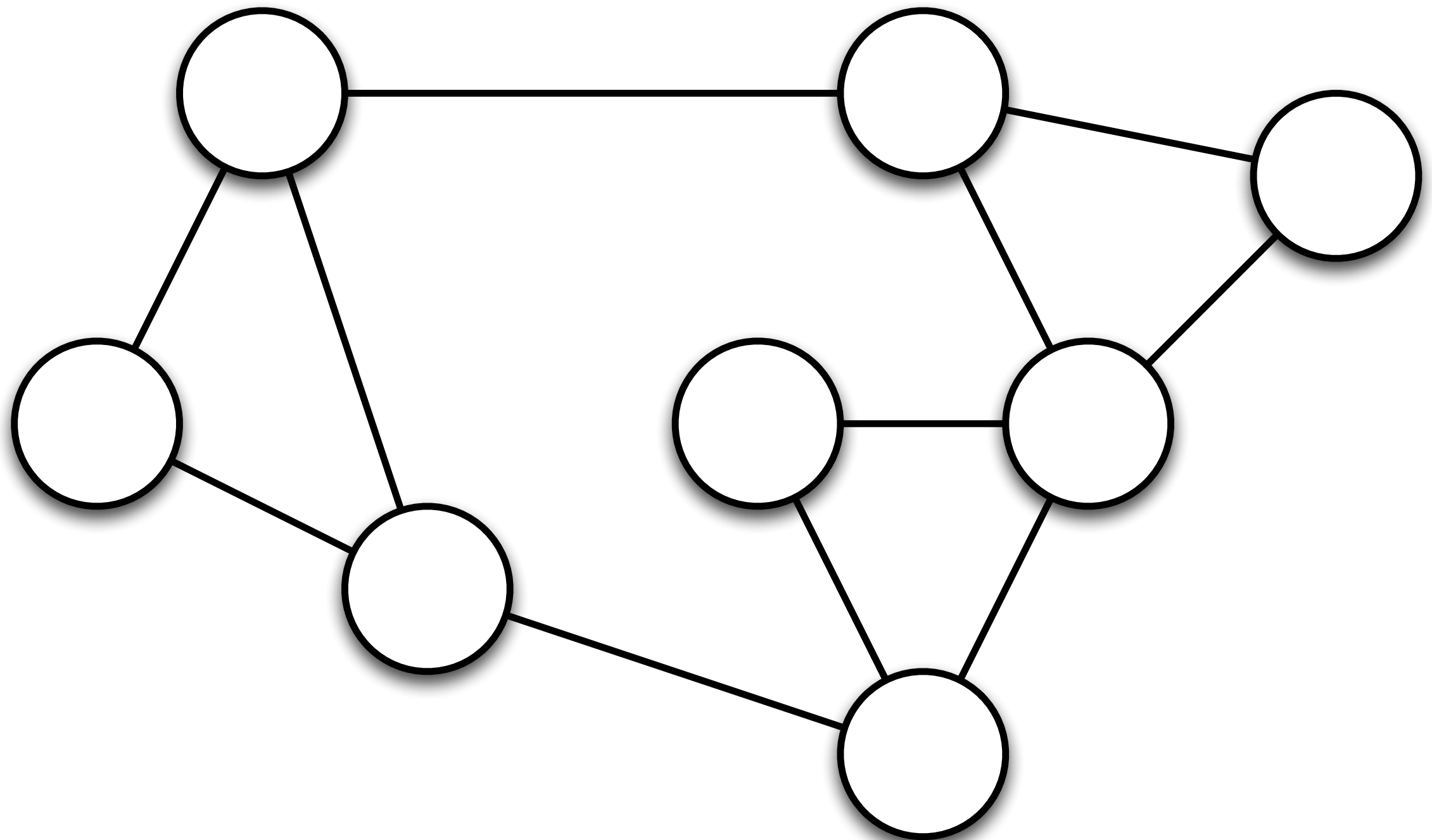


- Min-cut multihop communication
- Node-disjoint multihop communication
- Direct communication

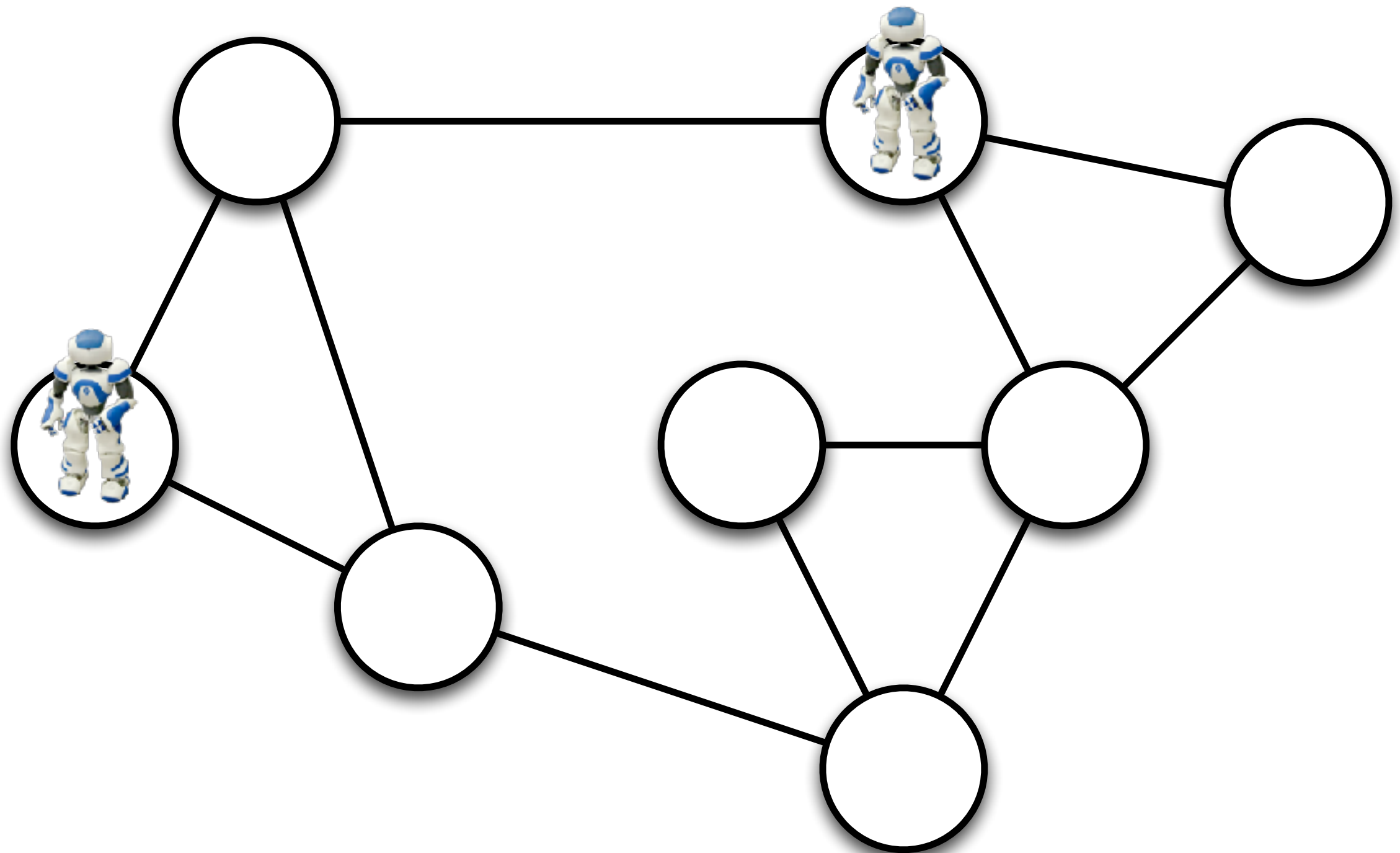
Actively Mobile Networks

Mobile Agents

Mobile Agents



Mobile Agents



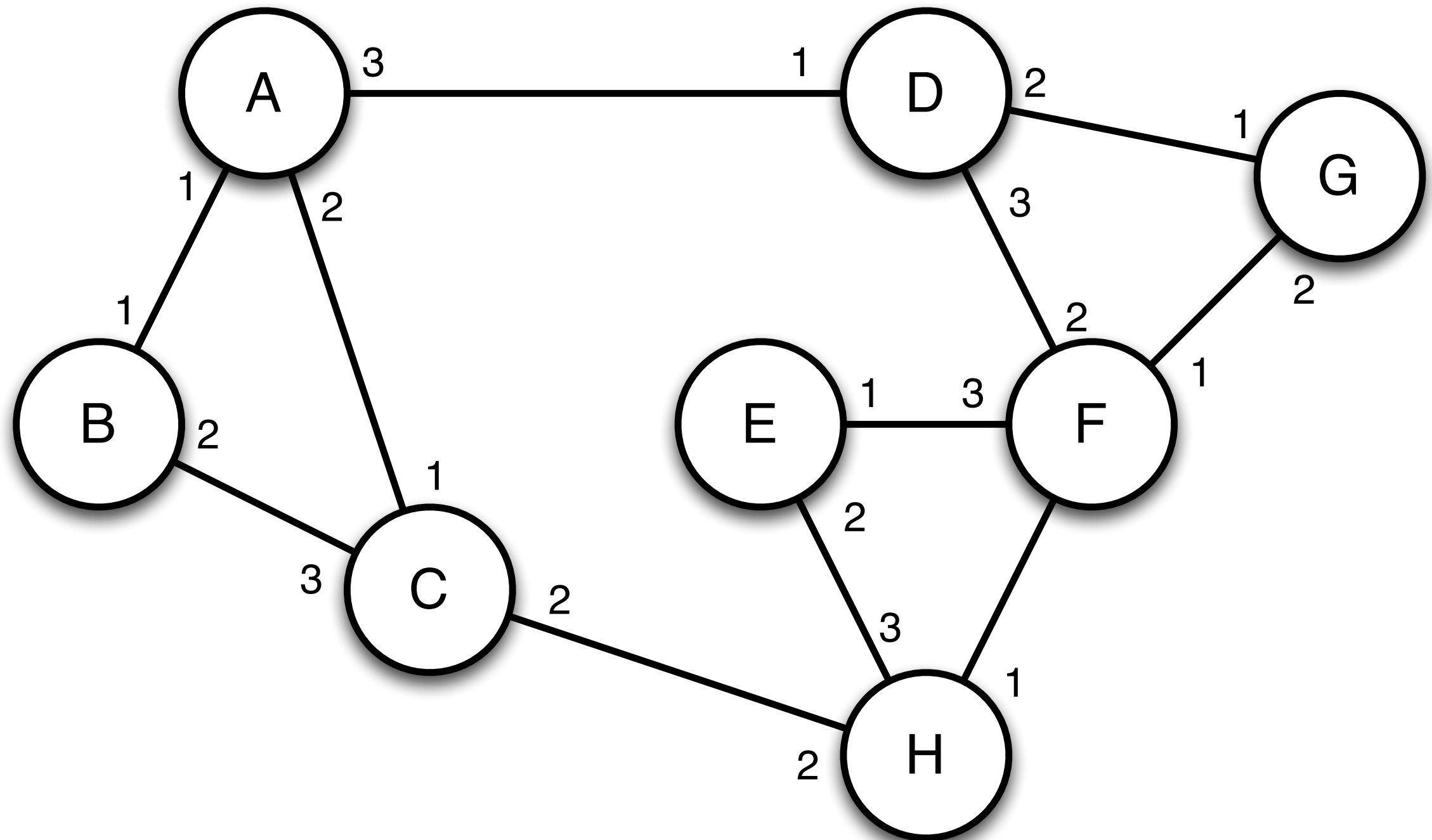
Problems to Solve

- **Exploration** (perpetual or with stop)
- **Mapping**
- **Rendez-vous**
- **Black hole search**
- **Capturing an intruder**

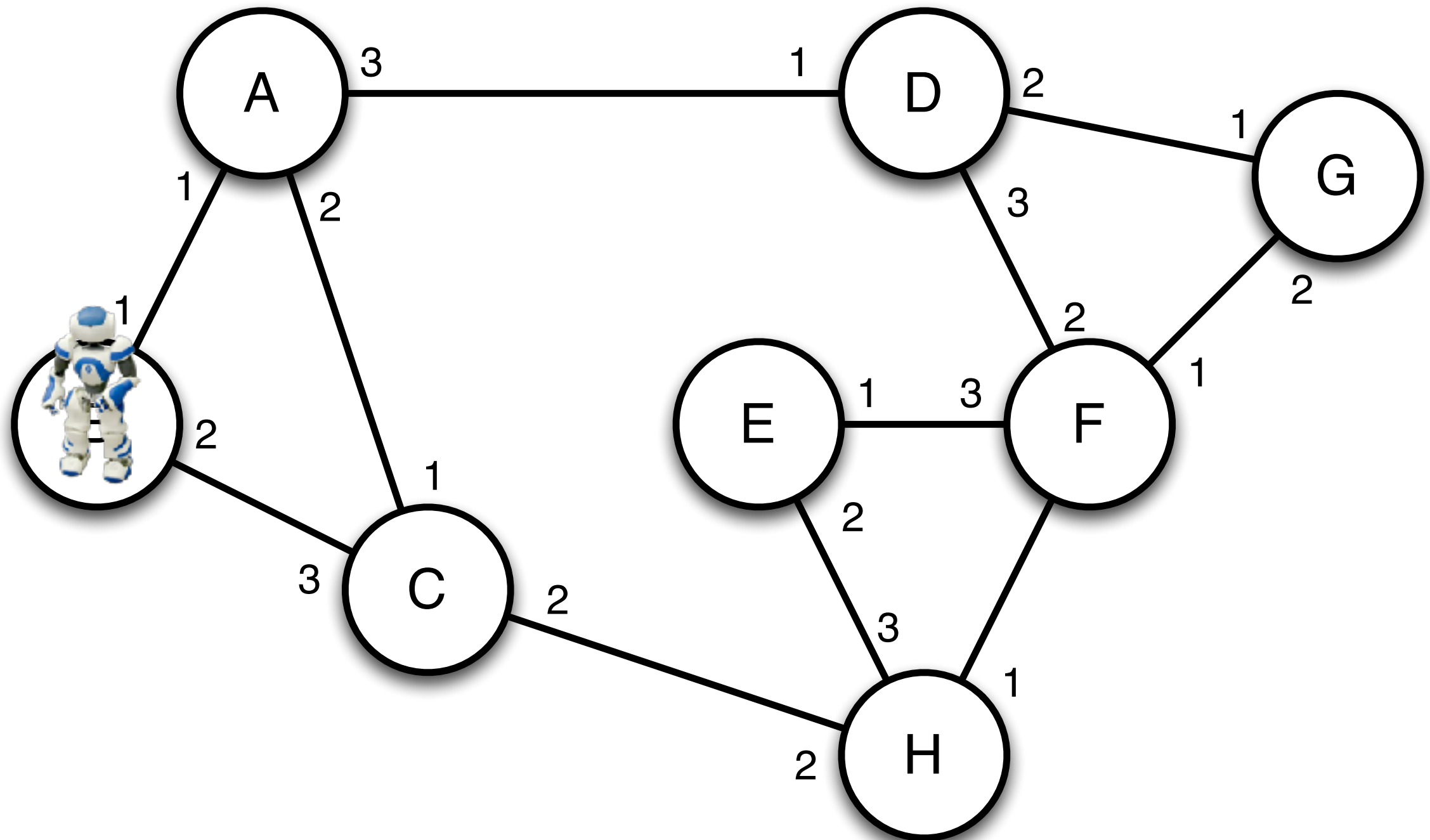
Models

- **Network** (anonymous vs. ID based)
- **Agents** (anonymous vs. ID based)
- **Synchrony**
- **Initial** (structural) **knowledge**
- **Communications** (none, pebbles, whiteboards)
- Agent **memory** (infinite, bounded, constant)

Mapping



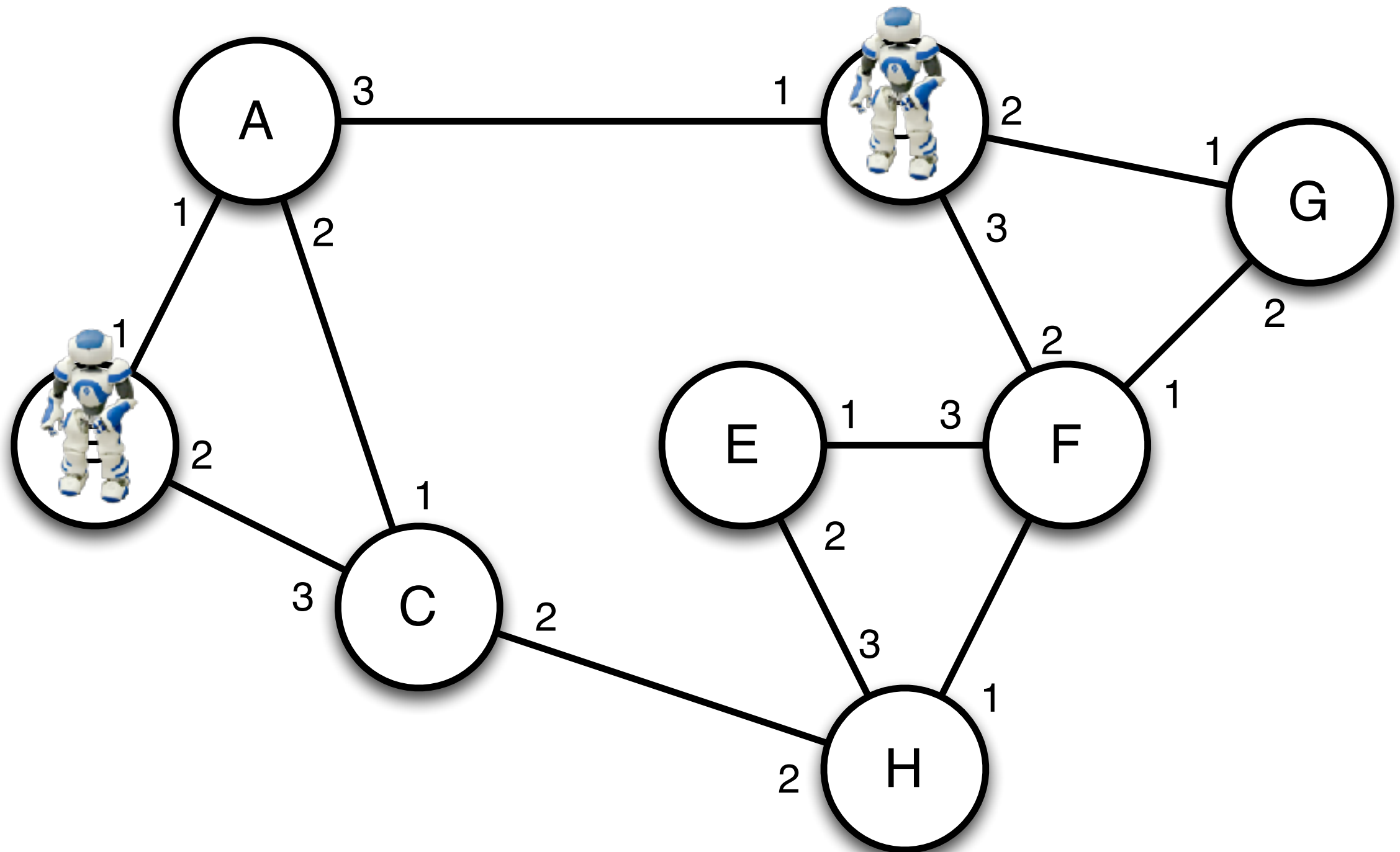
Mapping



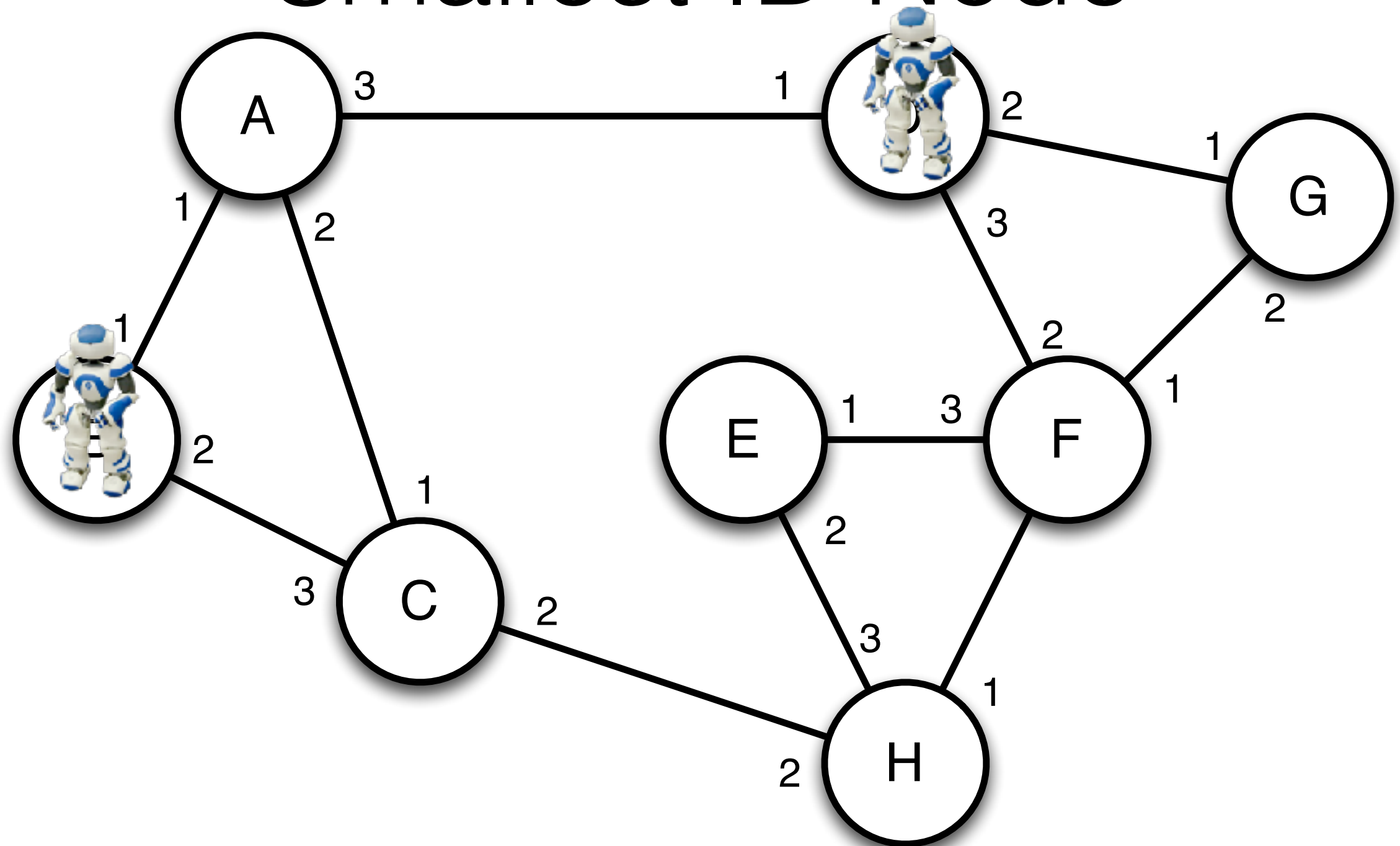
Rendez-vous

- **Two** (or more) **mobile agents must meet** in a graph
- They start on **distinct** locations
- **Computability?**
- **Complexity?**

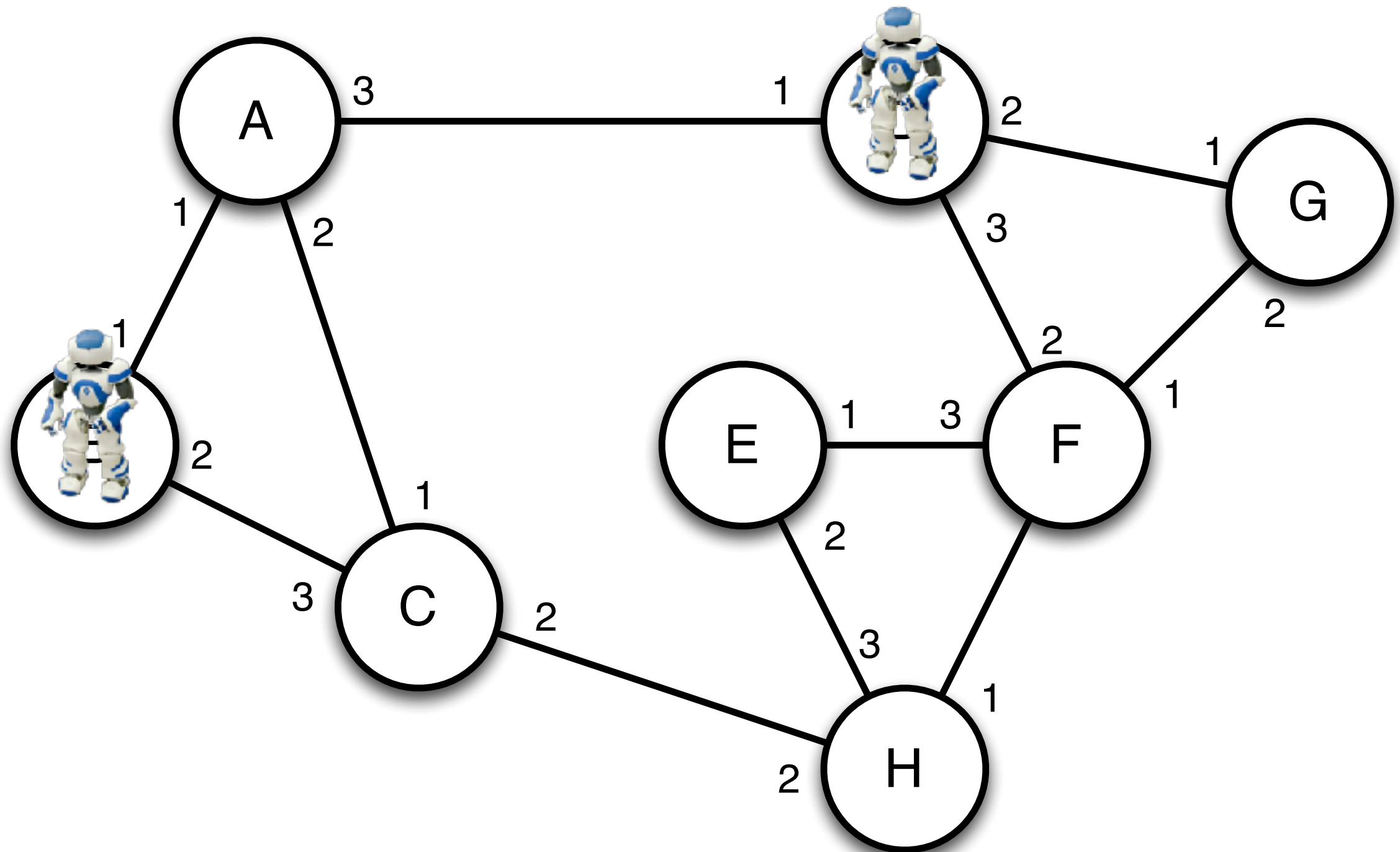
Rendez-vous in ID Graphs



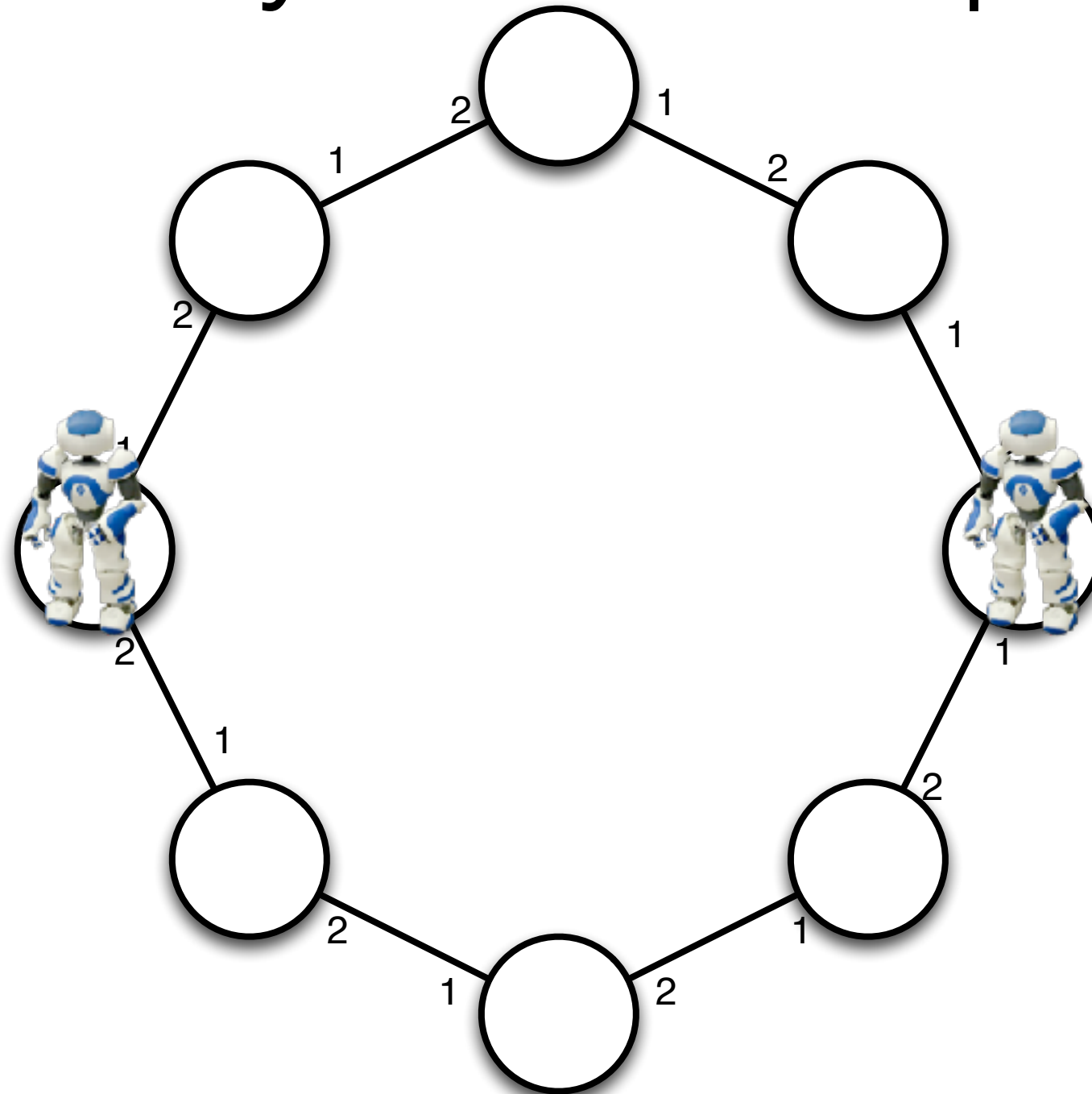
DFS to find Smallest ID Node



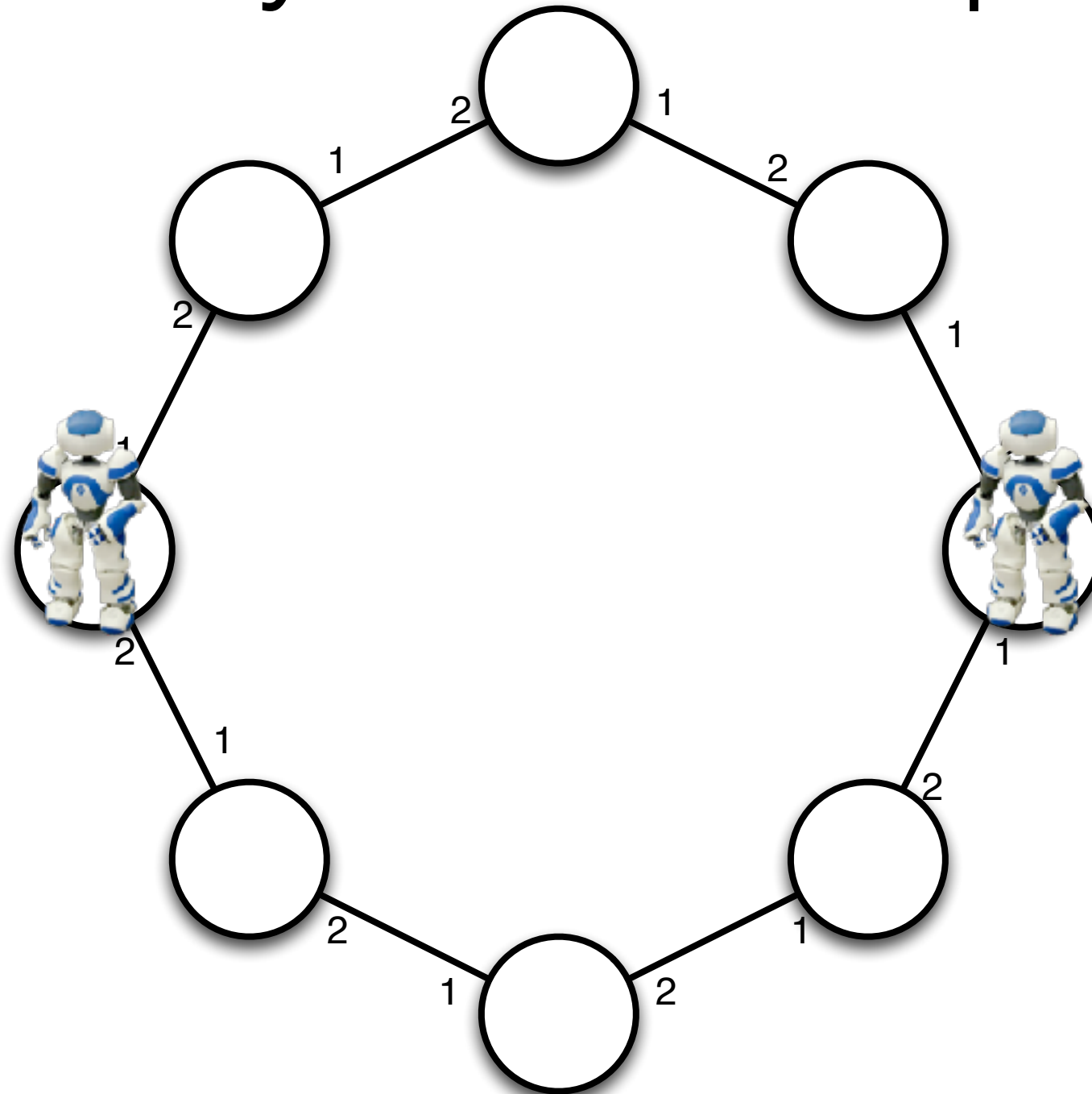
Rendez-vous in A



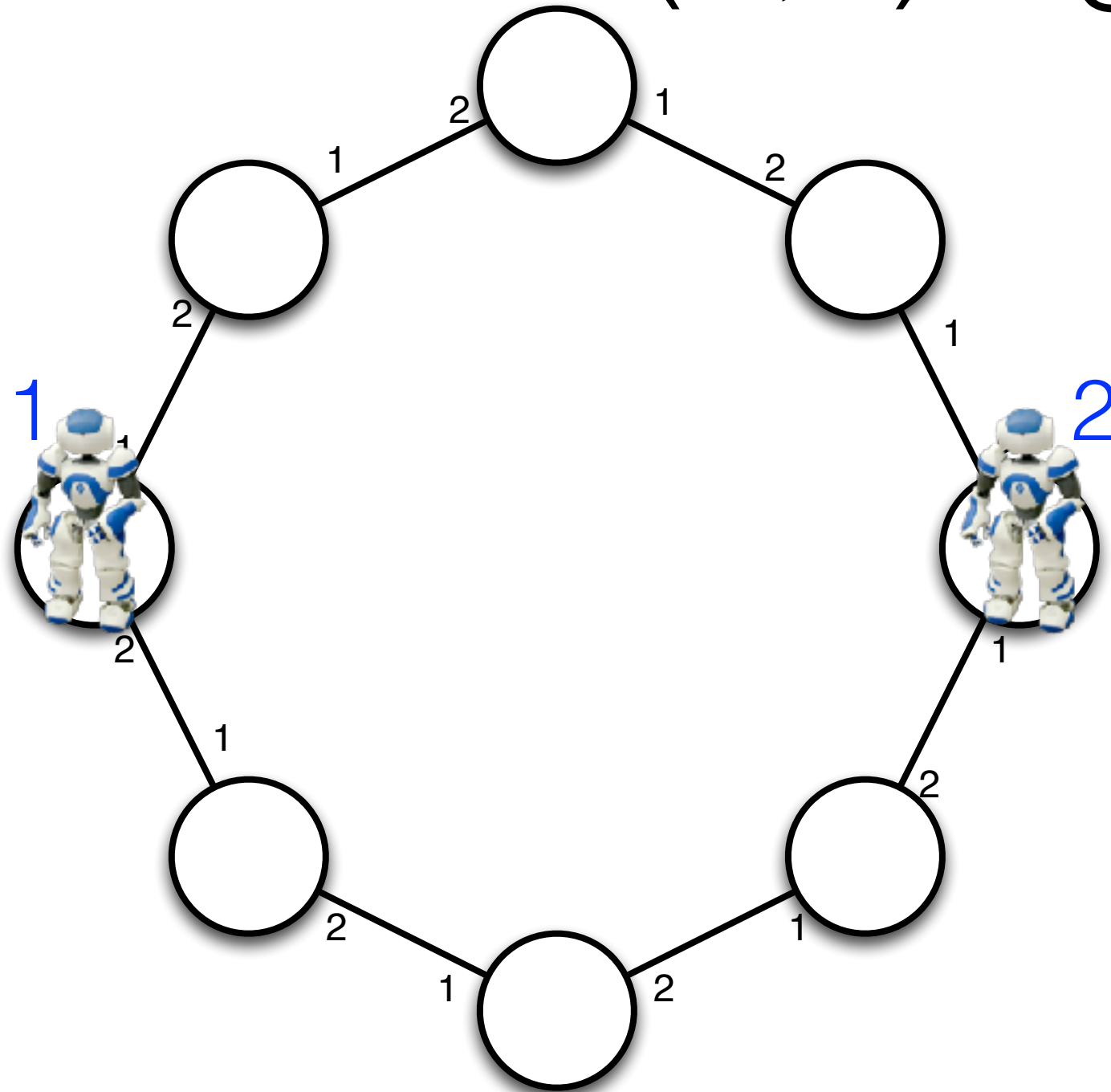
Rendez-vous in Anonymous Graphs



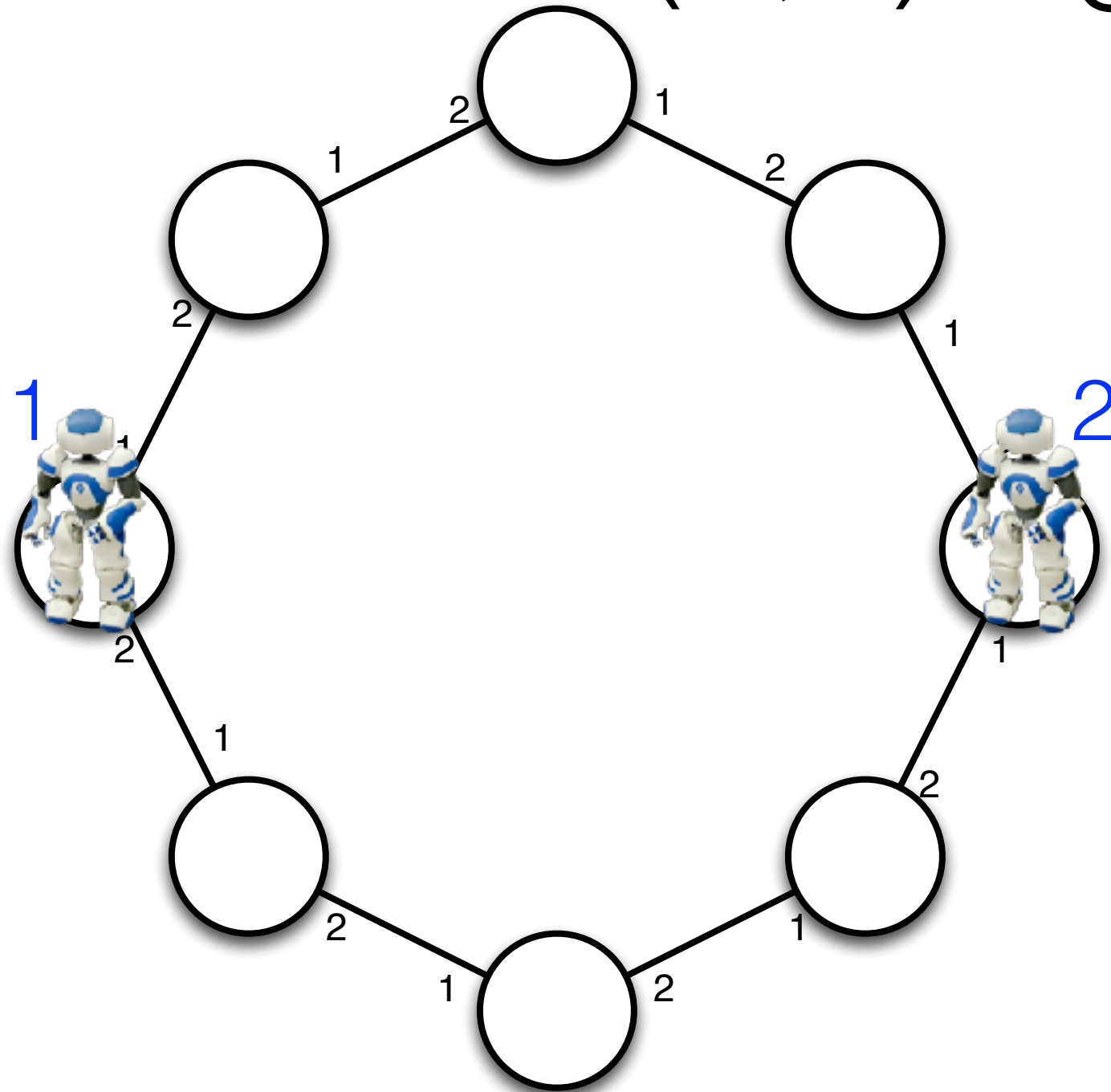
Rendez-vous in Anonymous Graphs



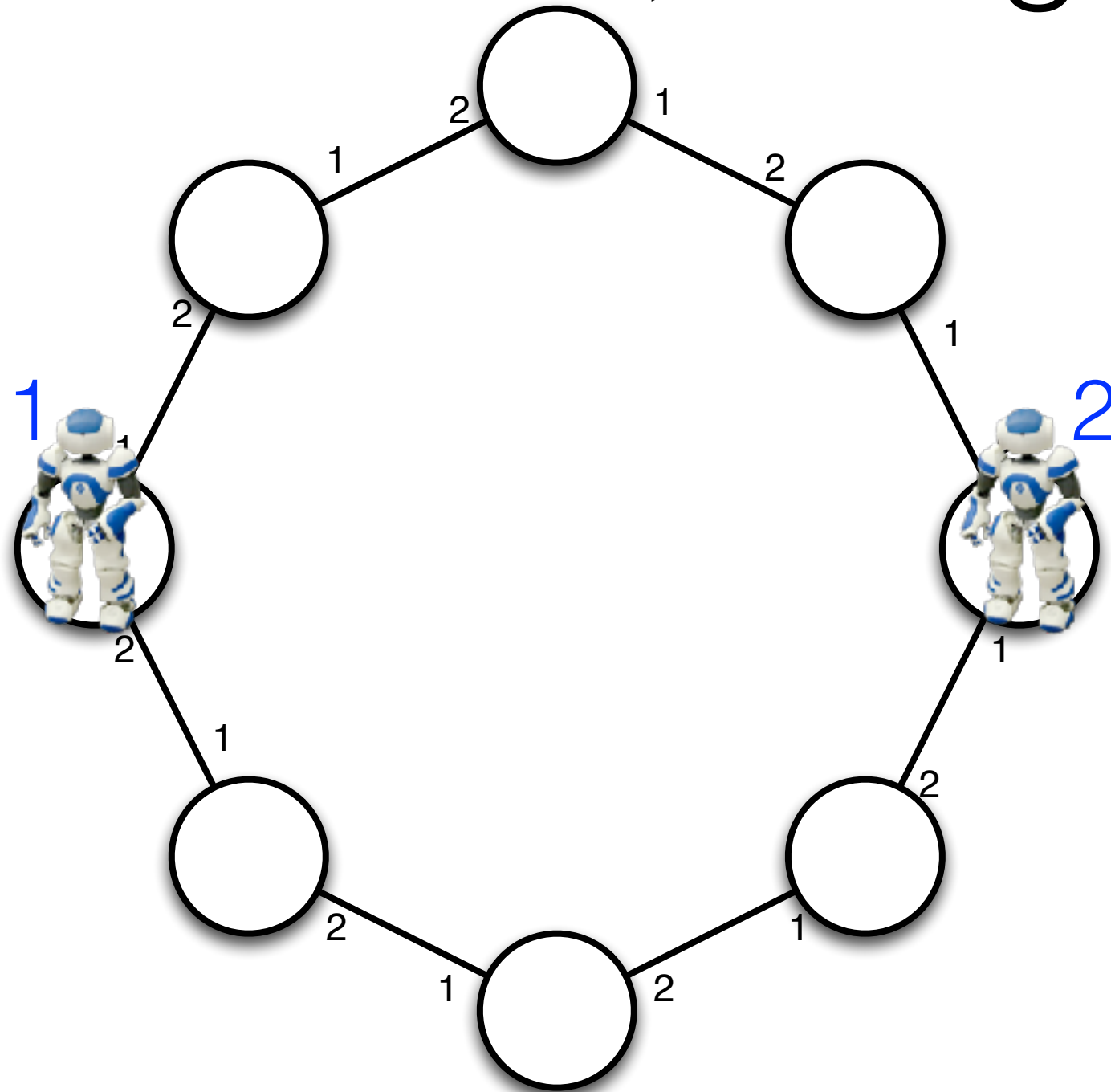
Anonymous Graphs with Known ID (1,2) Agents



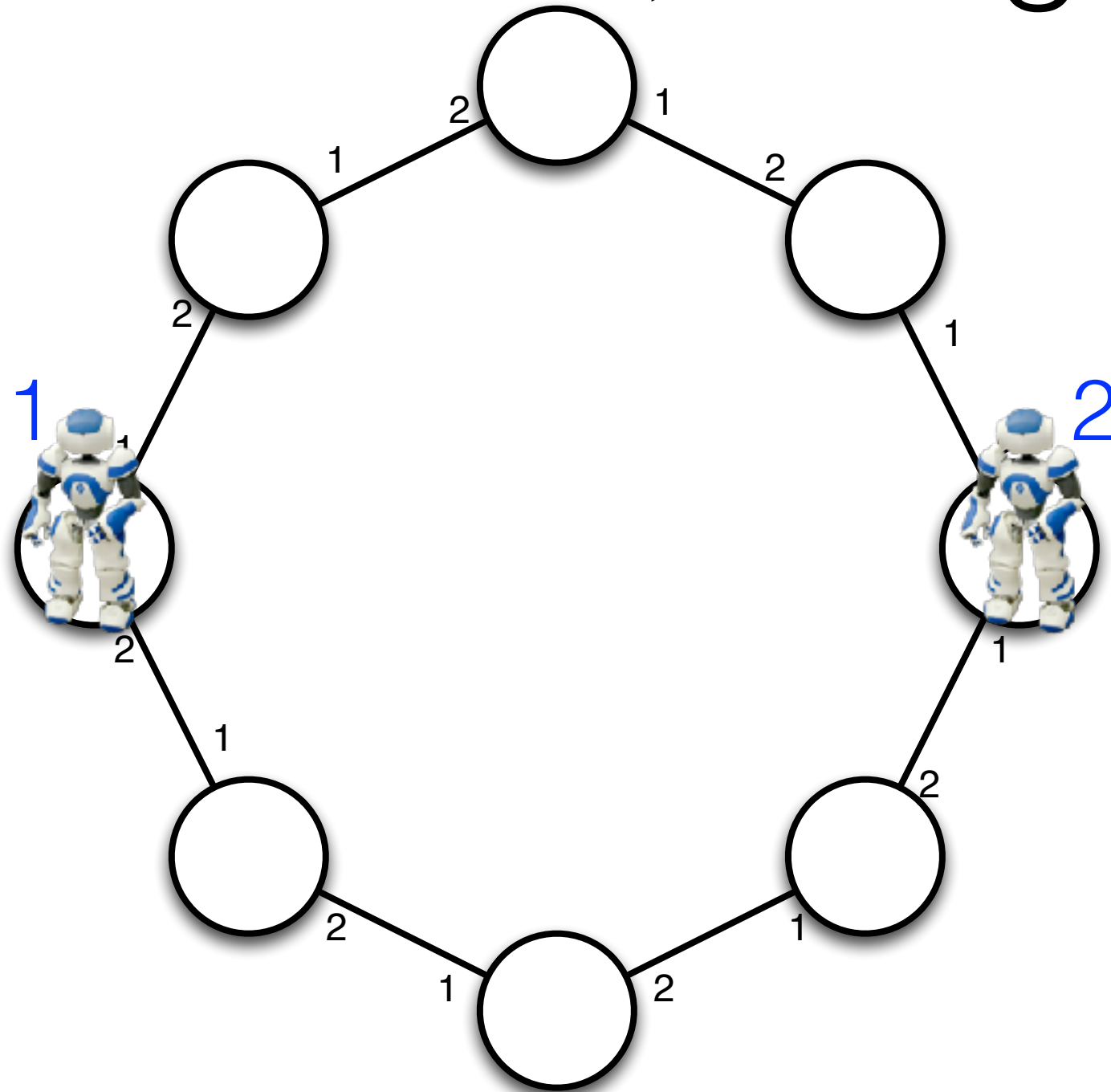
Anonymous Graphs with Known ID (1,2) Agents



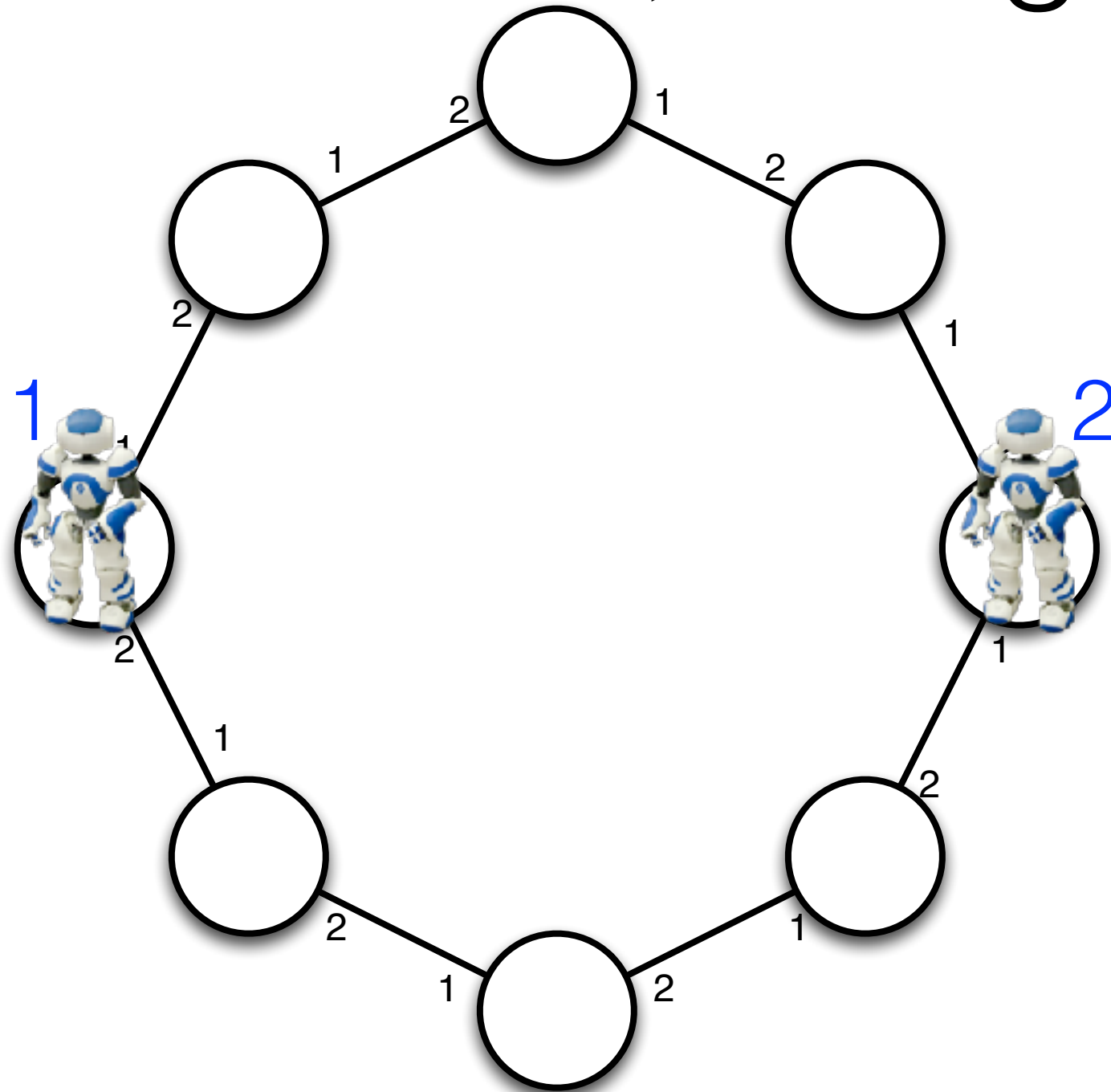
Anonymous Graphs with Known N, ID Agents



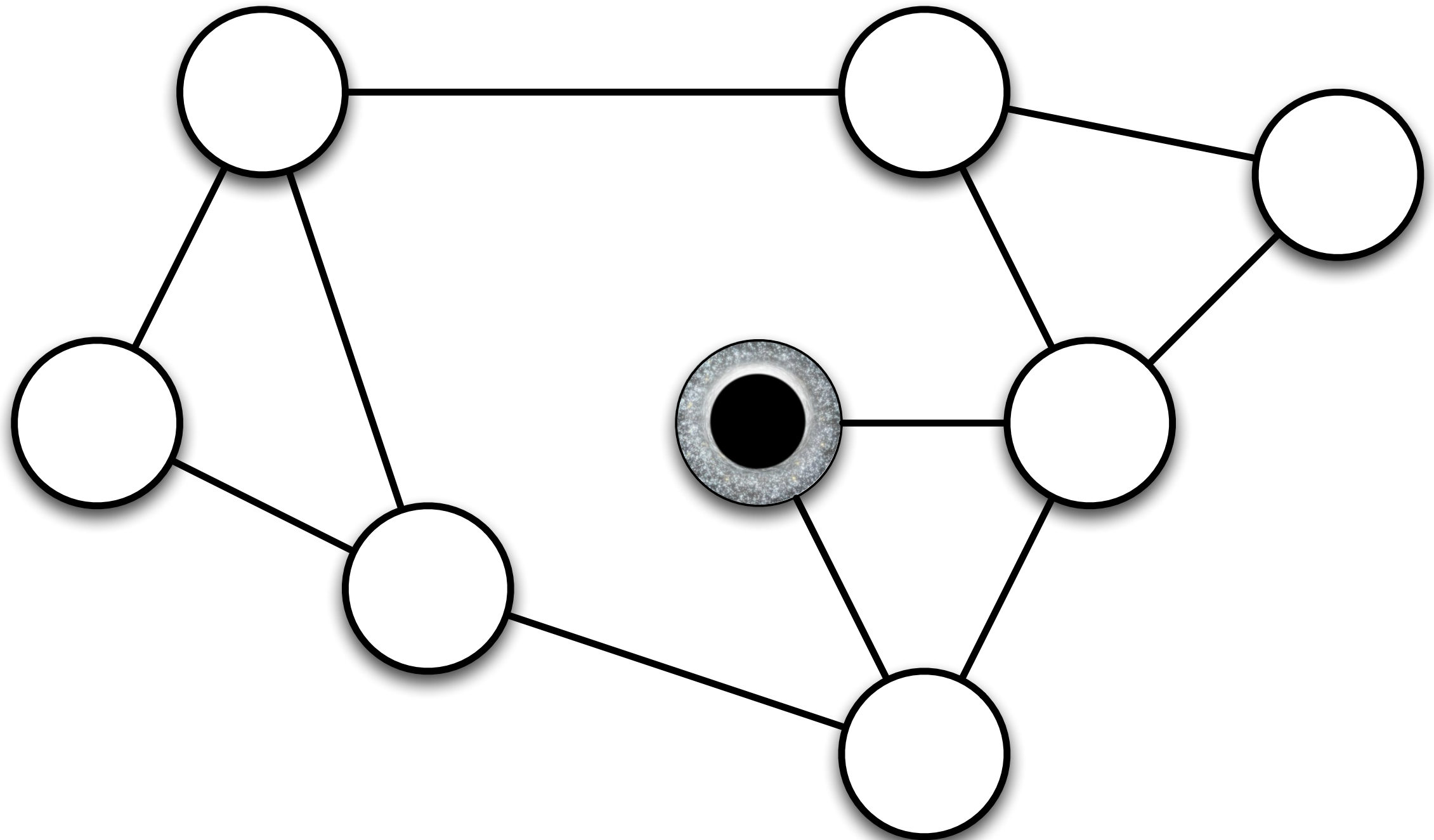
Anonymous Graphs with Known N, ID Agents



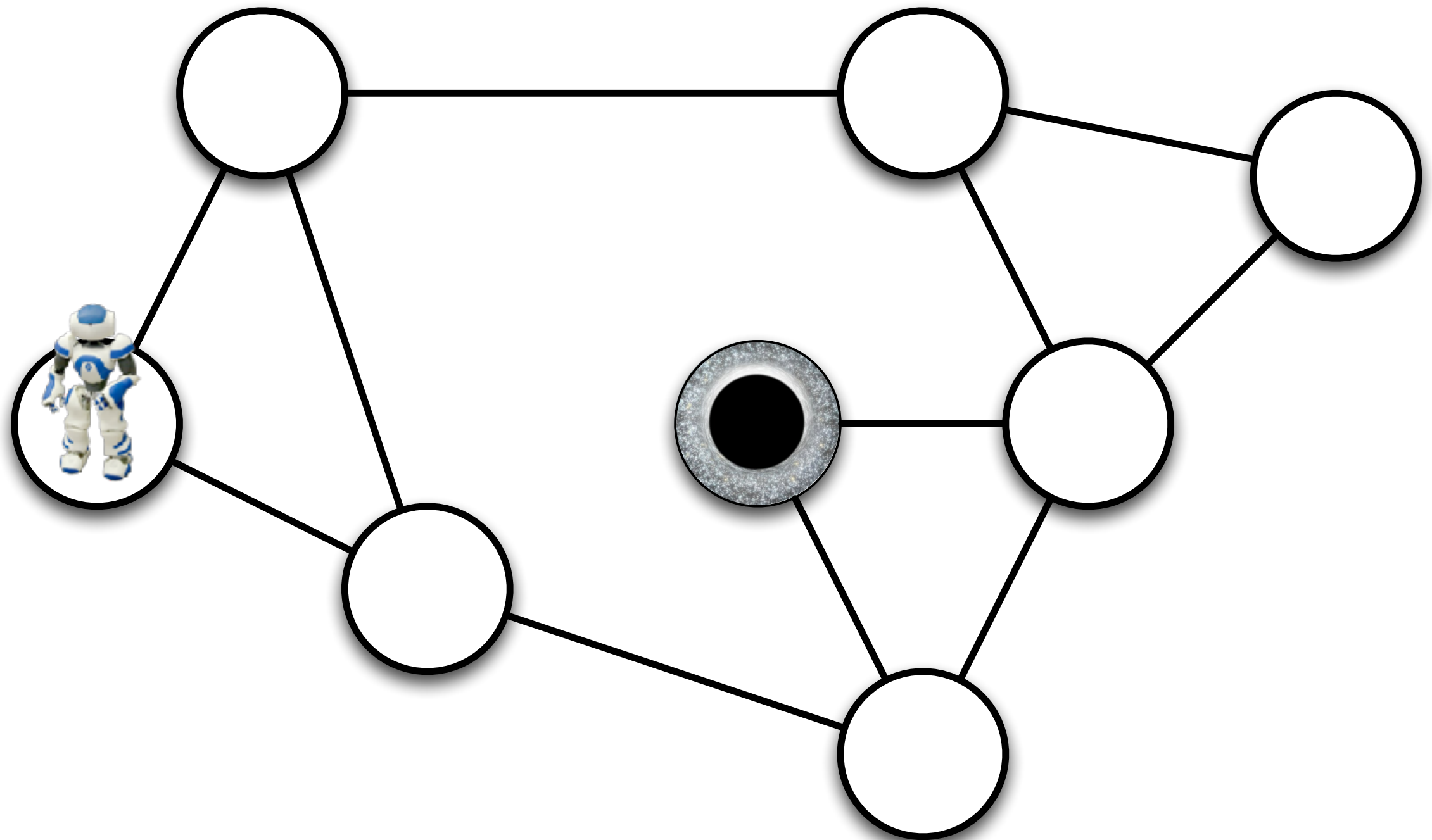
Anonymous Graphs with Known N, ID Agents



Black Hole Search



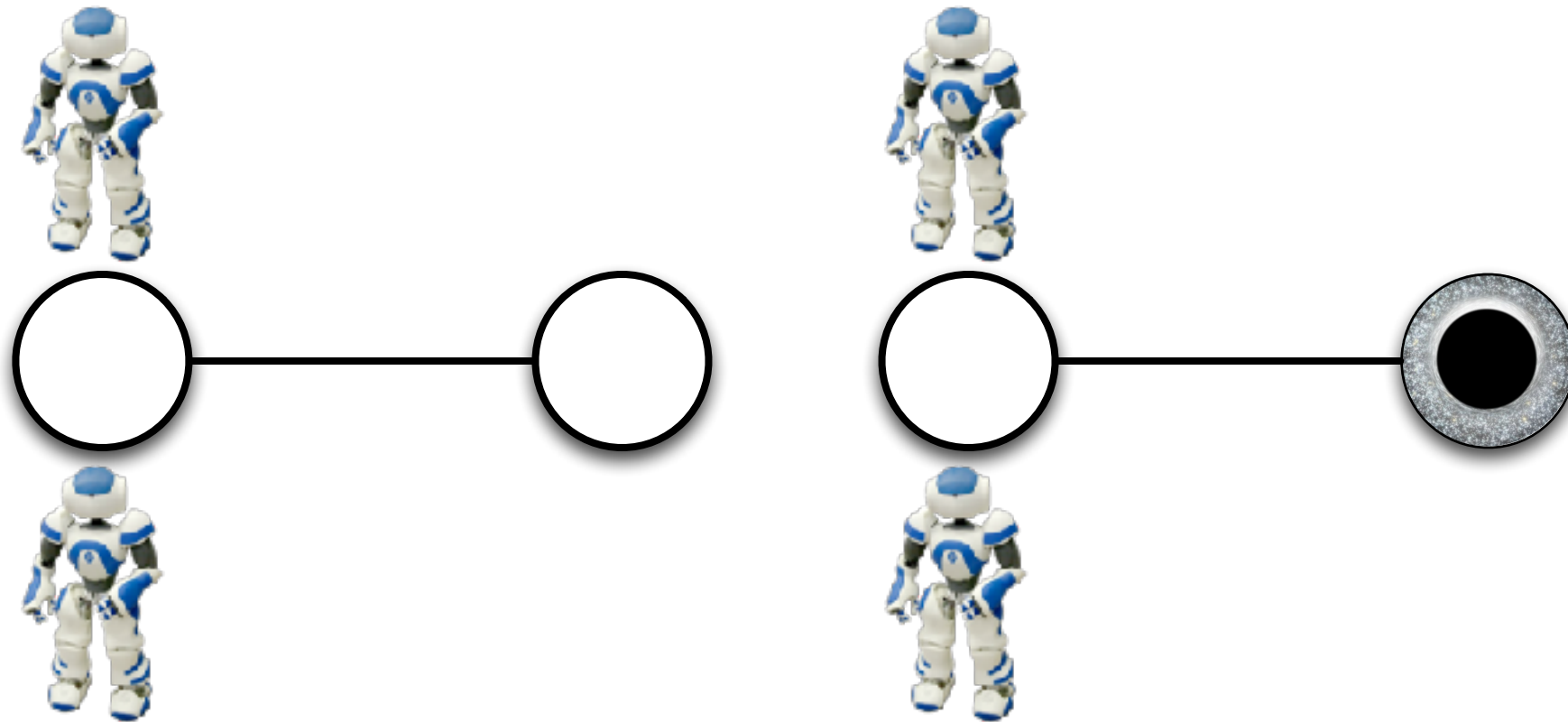
Black Hole Search



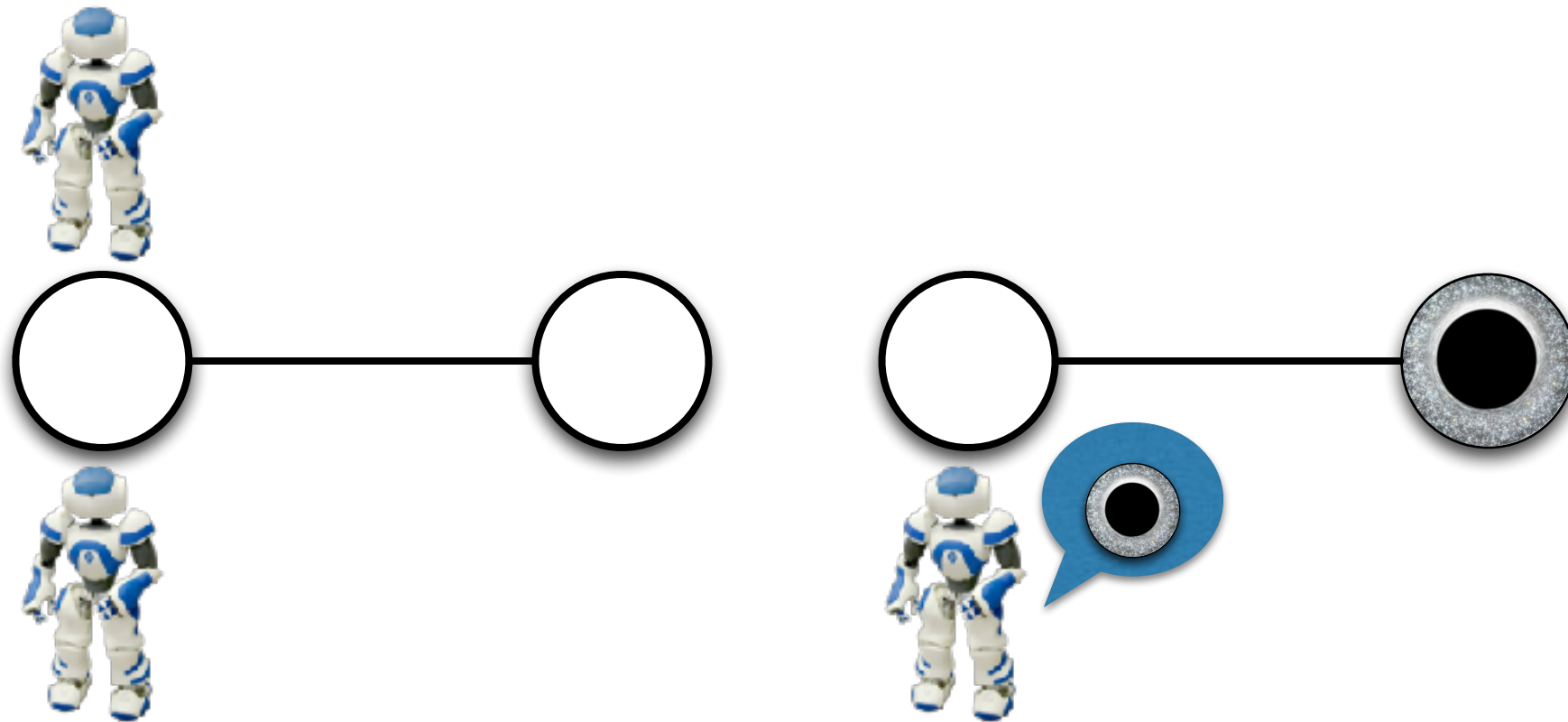
Black Hole Search

- A **single** black hole in the graph
- The black hole **does not disconnect** the graph
- Identify each **adjacent edge**
- **Minimize** #agents, #moves

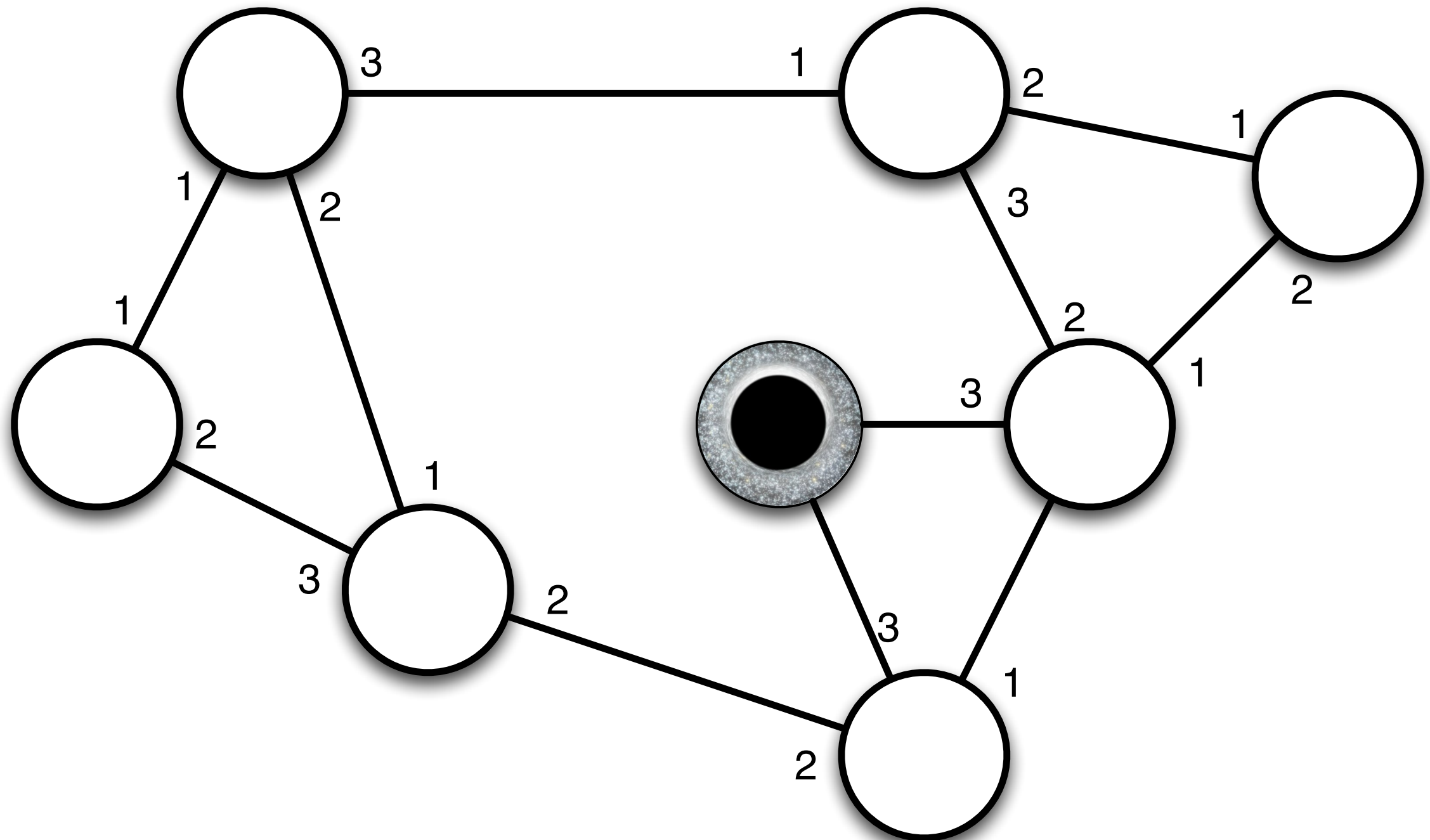
Synchronous Agents



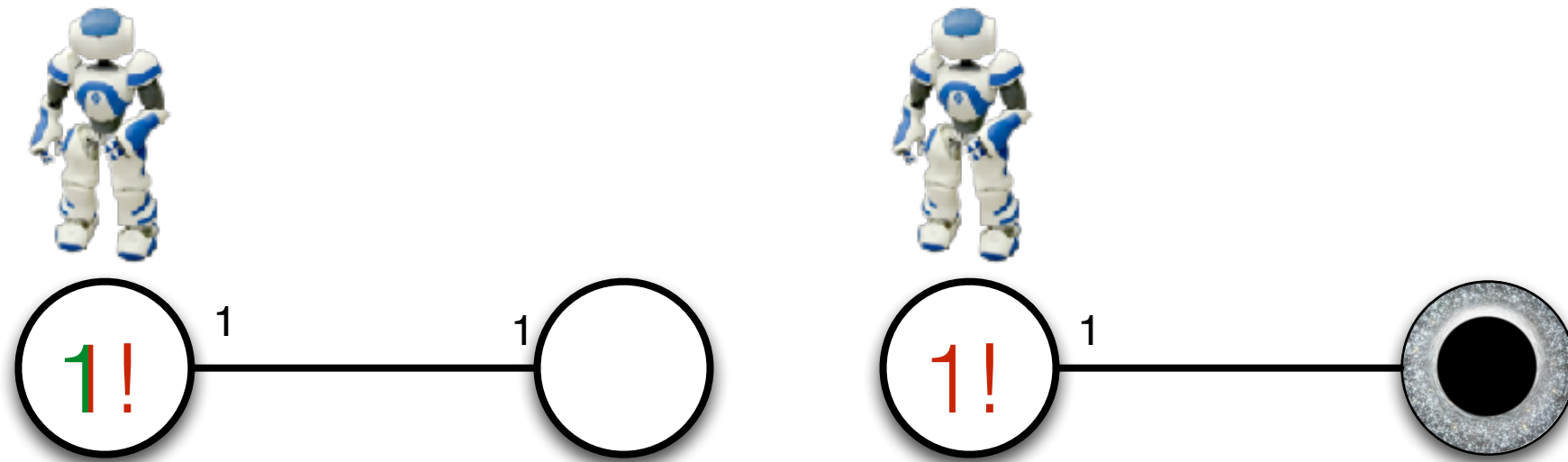
Synchronous Agents



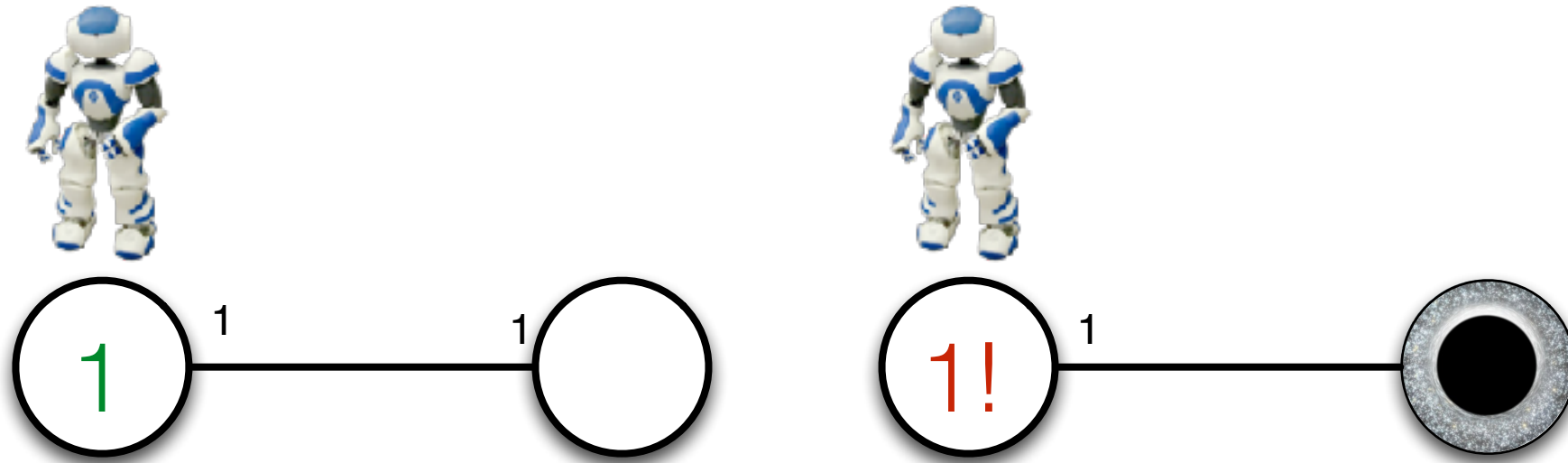
Asynchronous Black Hole Search



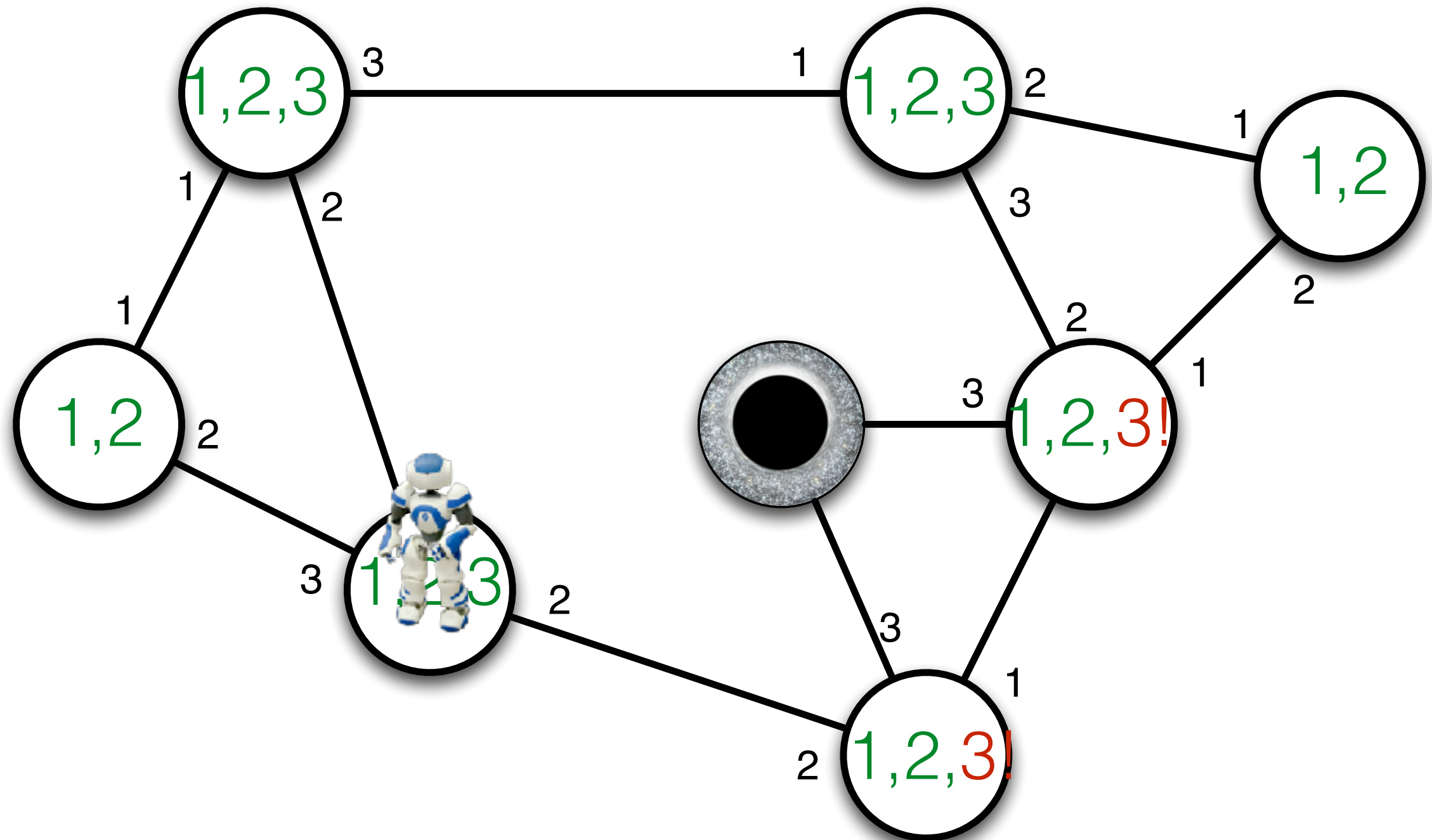
Asynchronous Black Hole Search



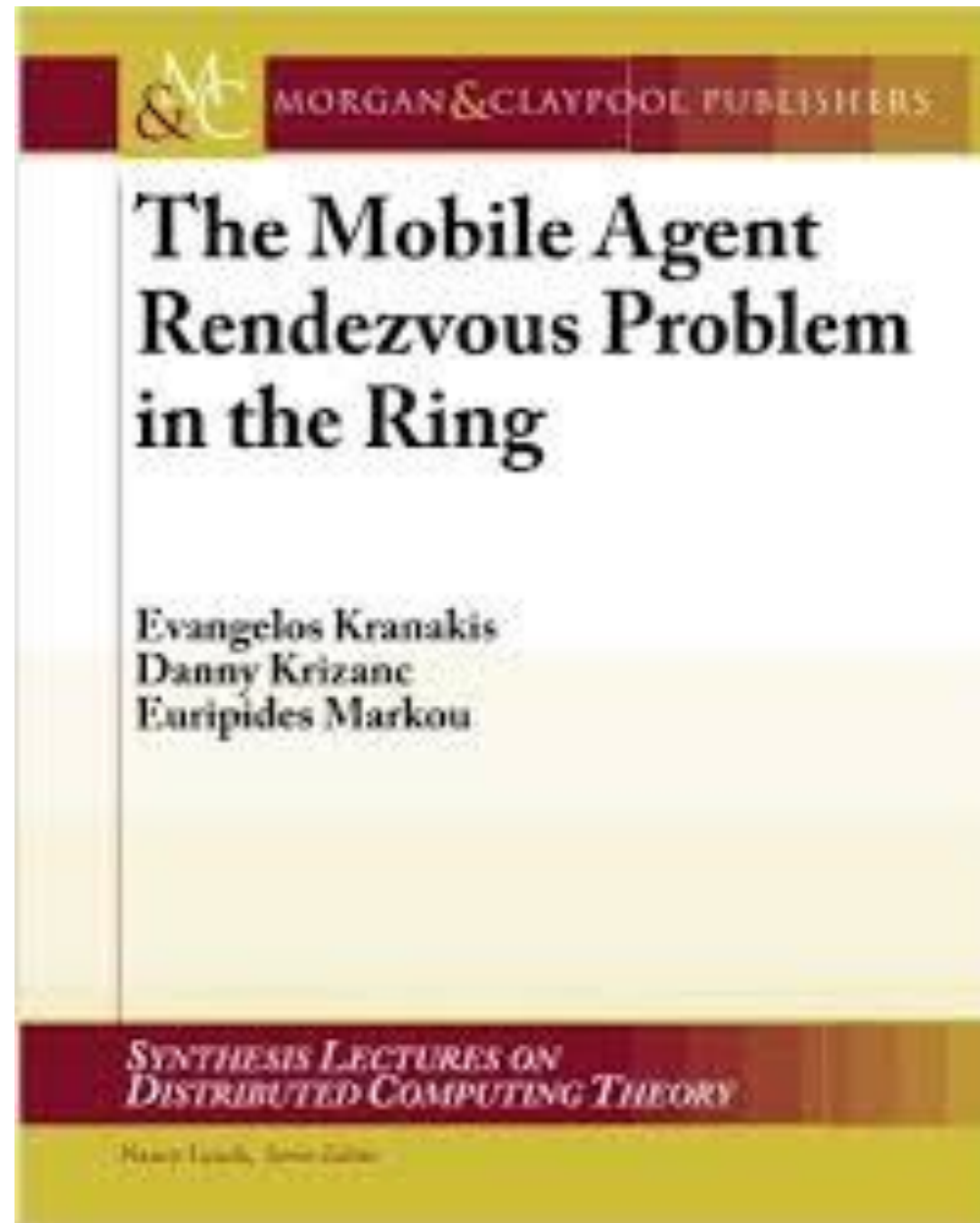
Asynchronous Black Hole Search



Asynchronous Black Hole Search



Mobile Agents

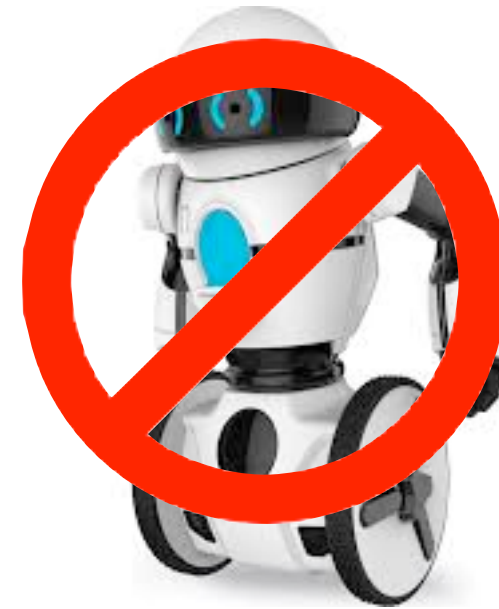


Mobile Robots

Mobile Robots



Mobile Robots



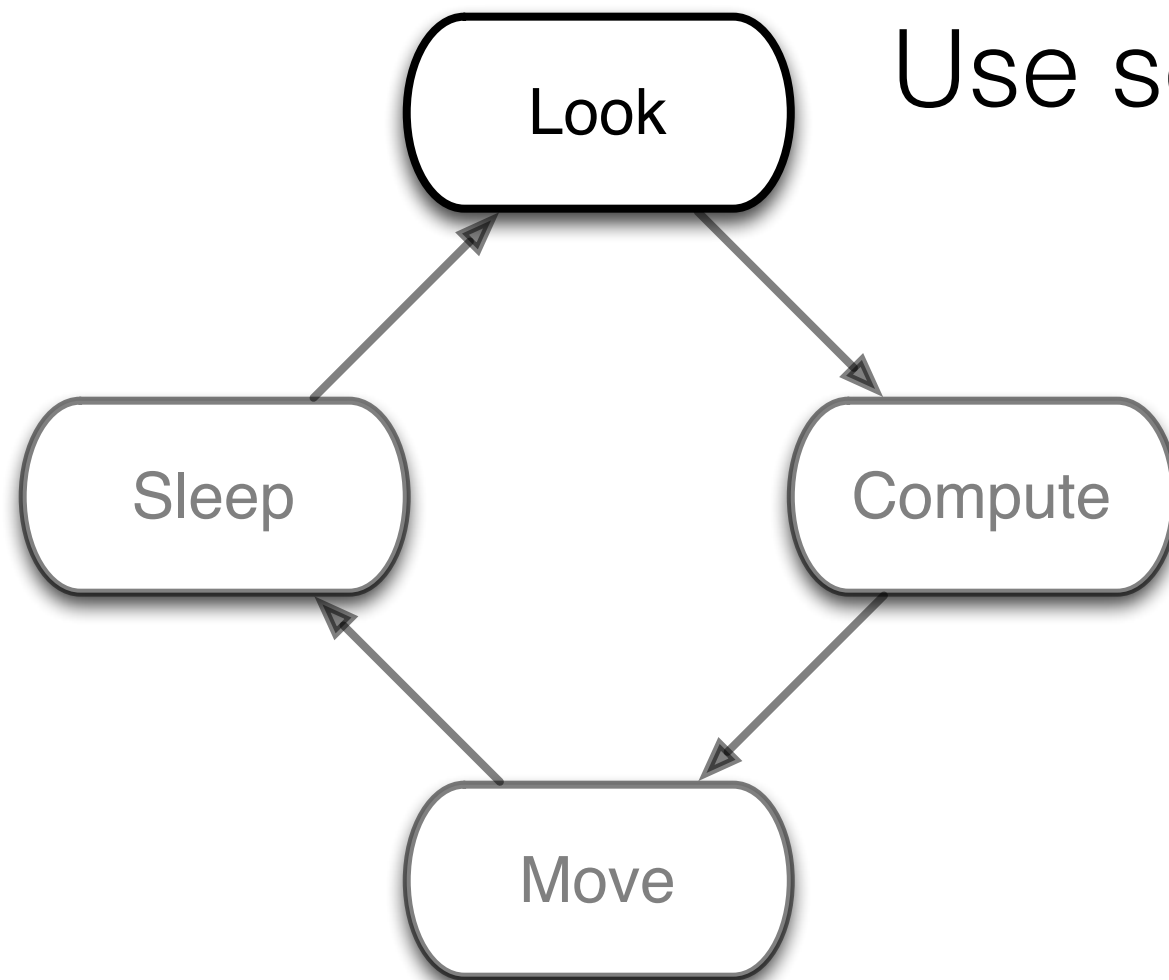
Mobile Robots



Mobile Robots

- **Autonomous** (no central control)
- **Homogeneous** (run same algorithm)
- **Identical** (indistinguishable)
- **Silent** (no explicit communication)

Robot Life Cycle

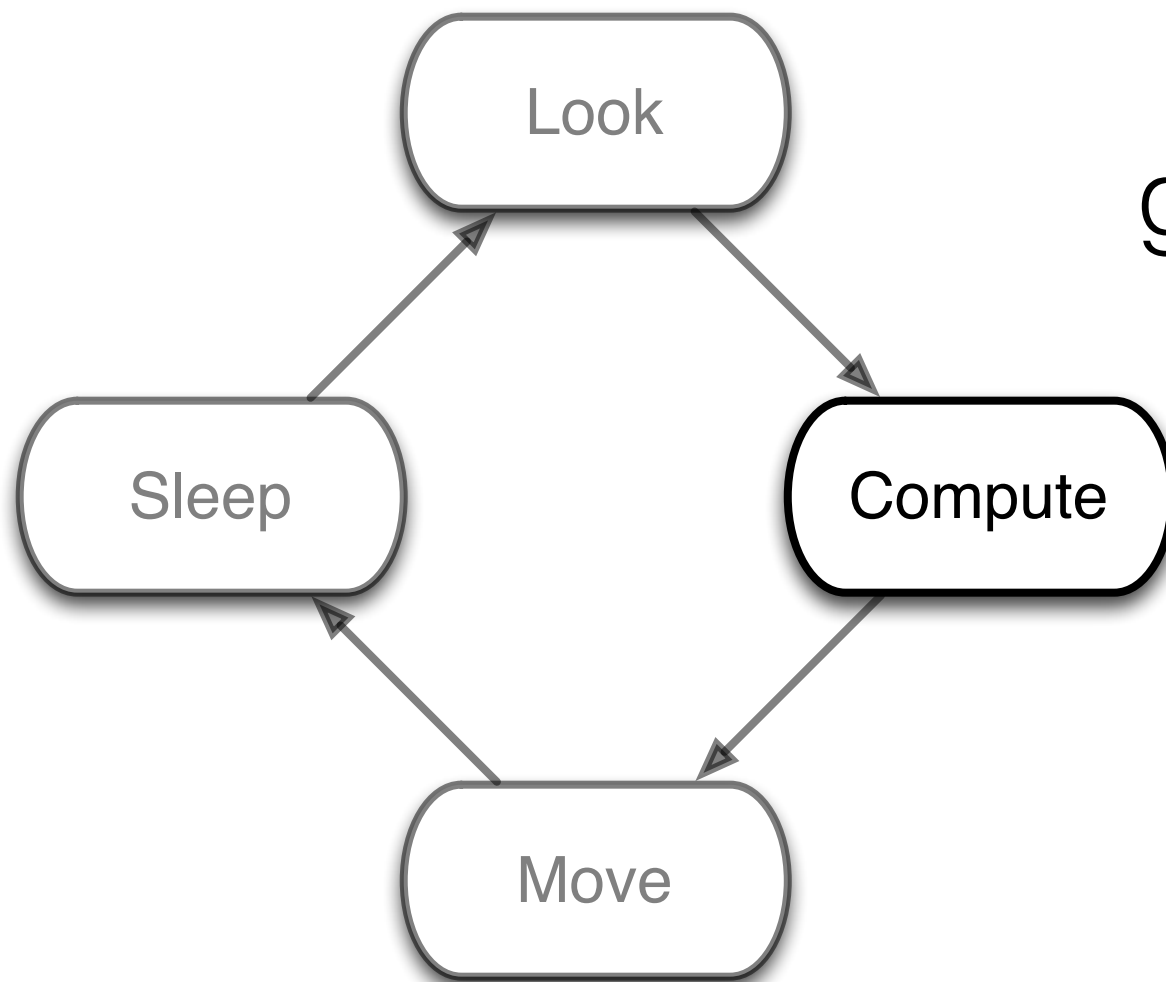


Use sensors to observe the world,
get a **snapshot**



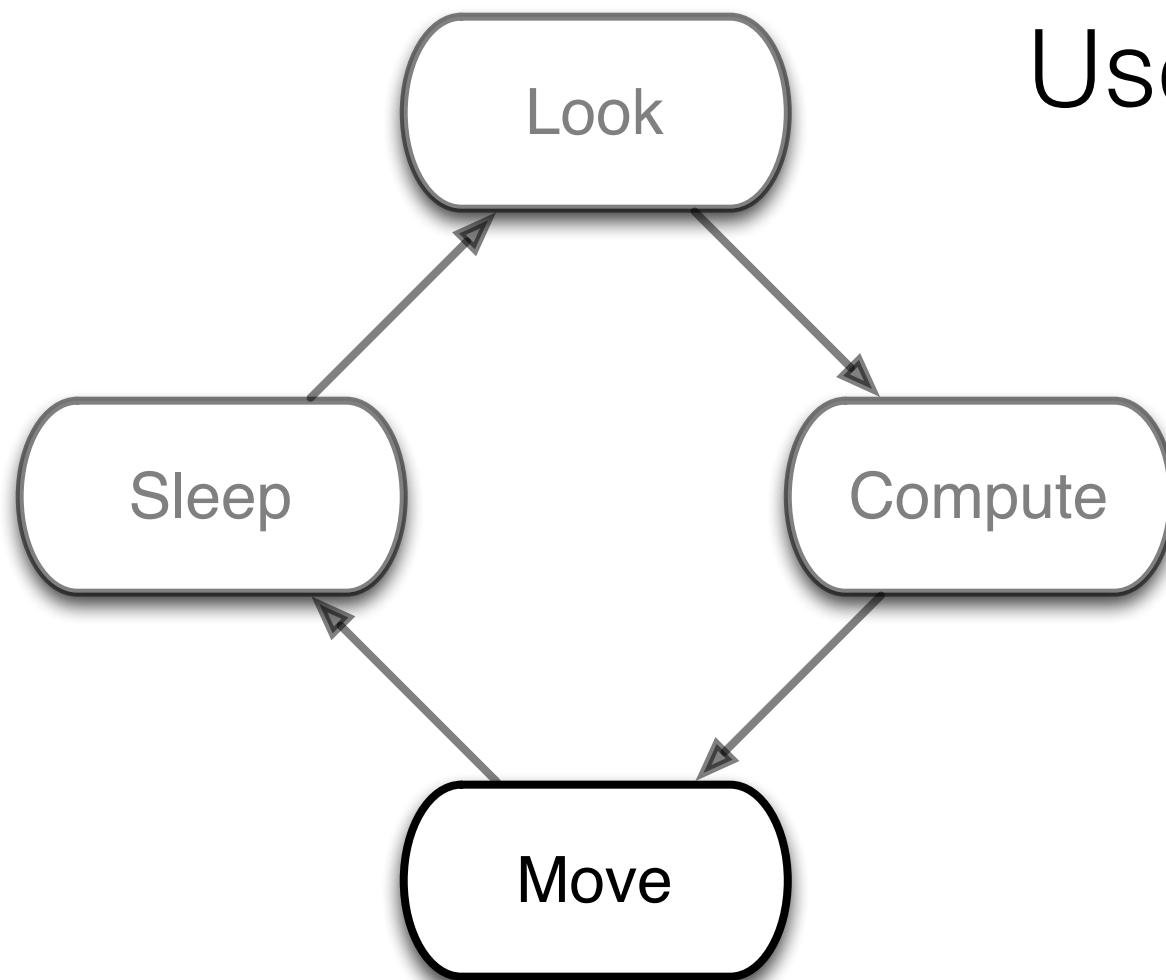
Robot Life Cycle

Execute the algorithm,
get a **destination point**

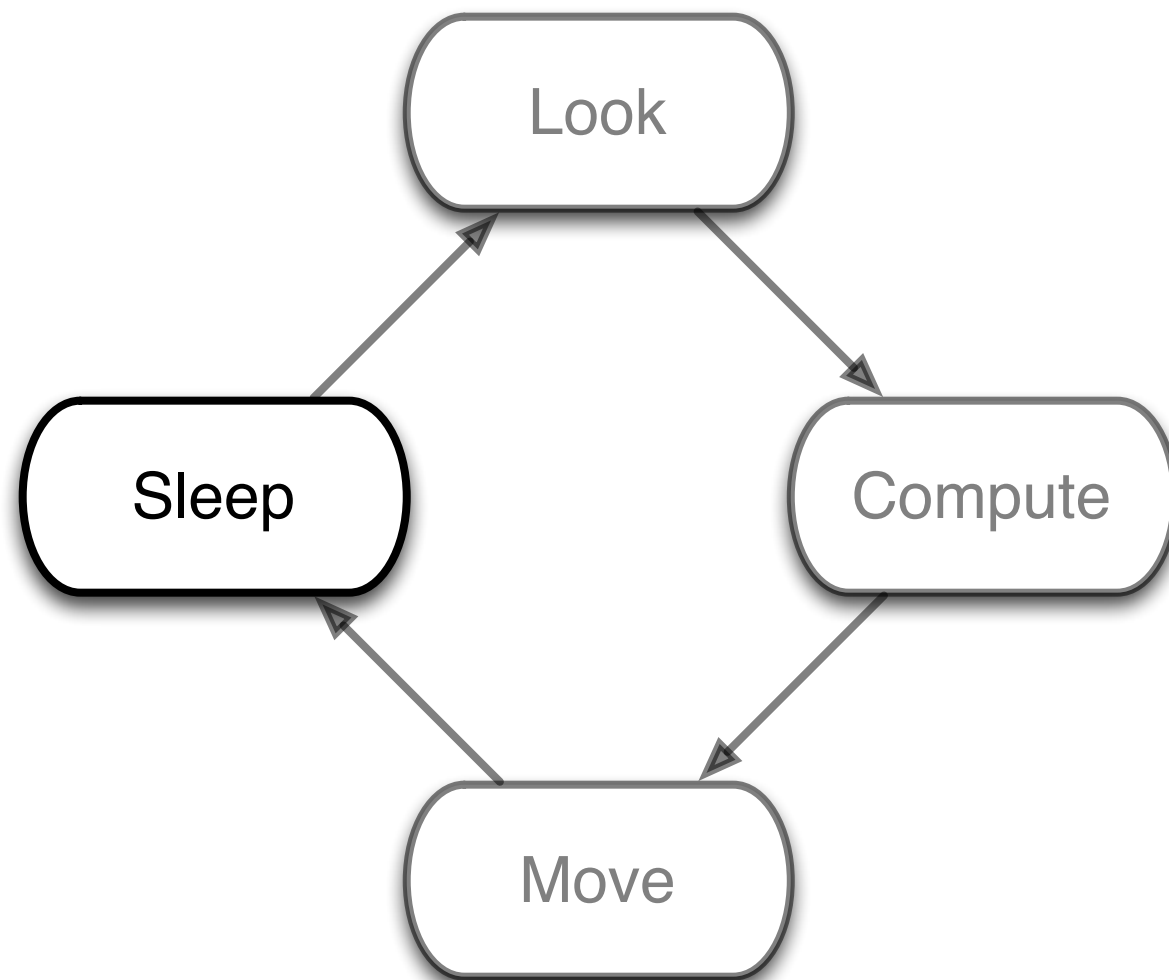


Robot Life Cycle

Use motors to **move toward** the destination point



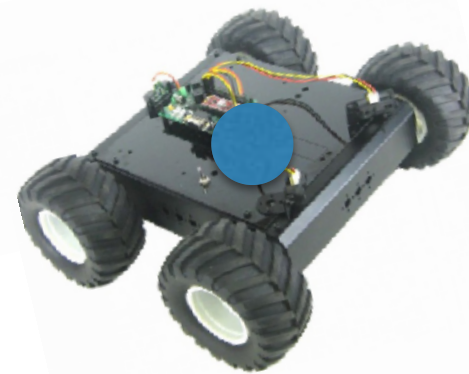
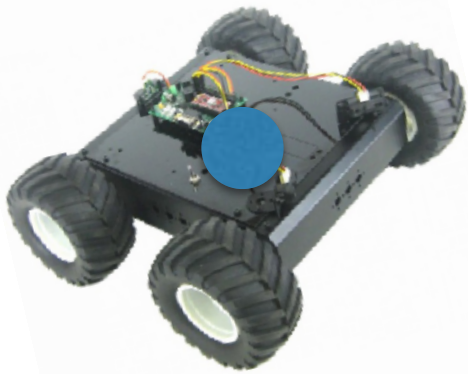
Robot Life Cycle



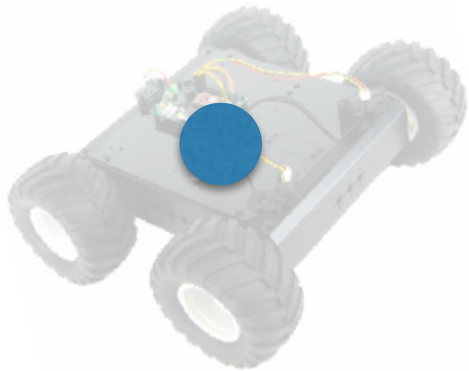
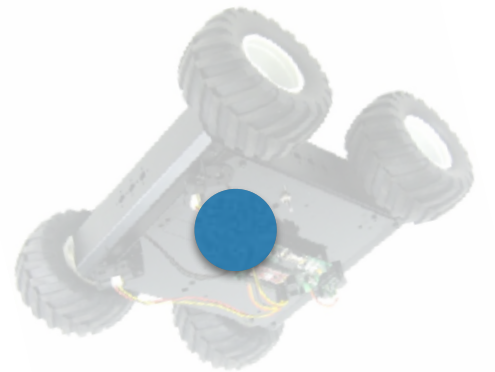
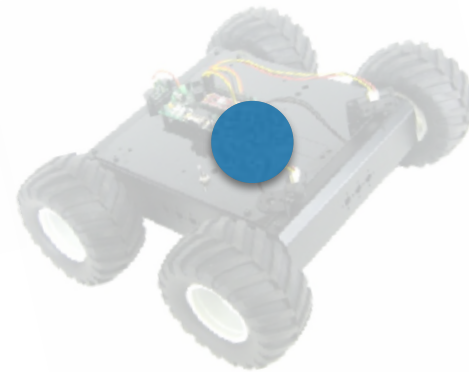
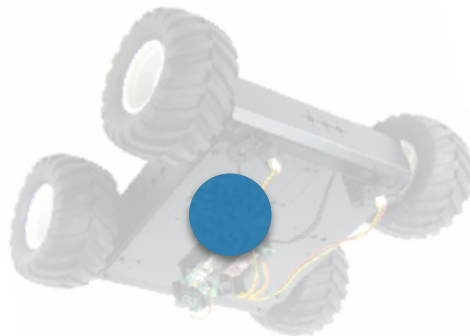
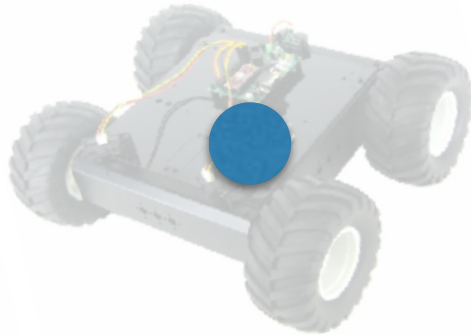
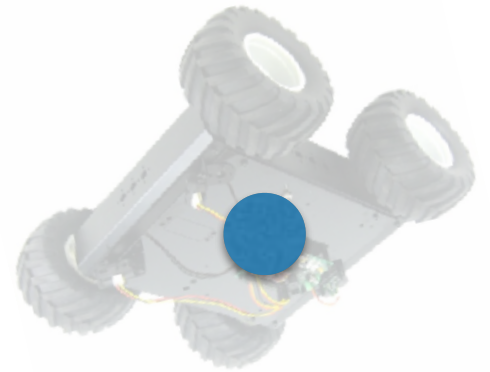
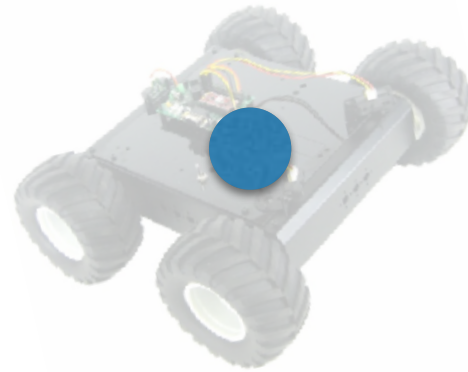
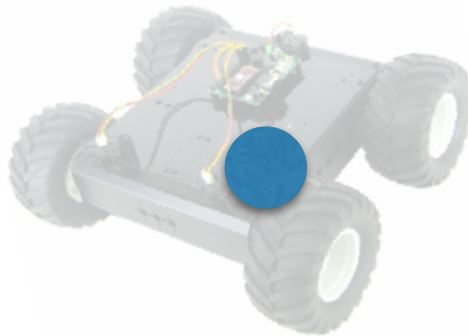
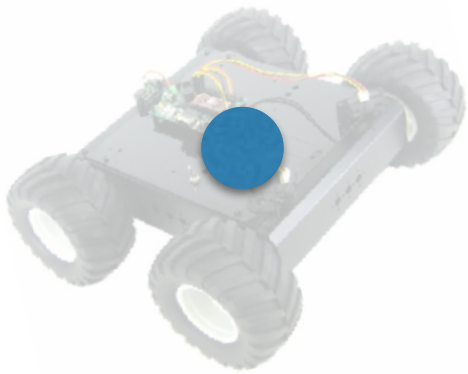
remain **idle** for a while



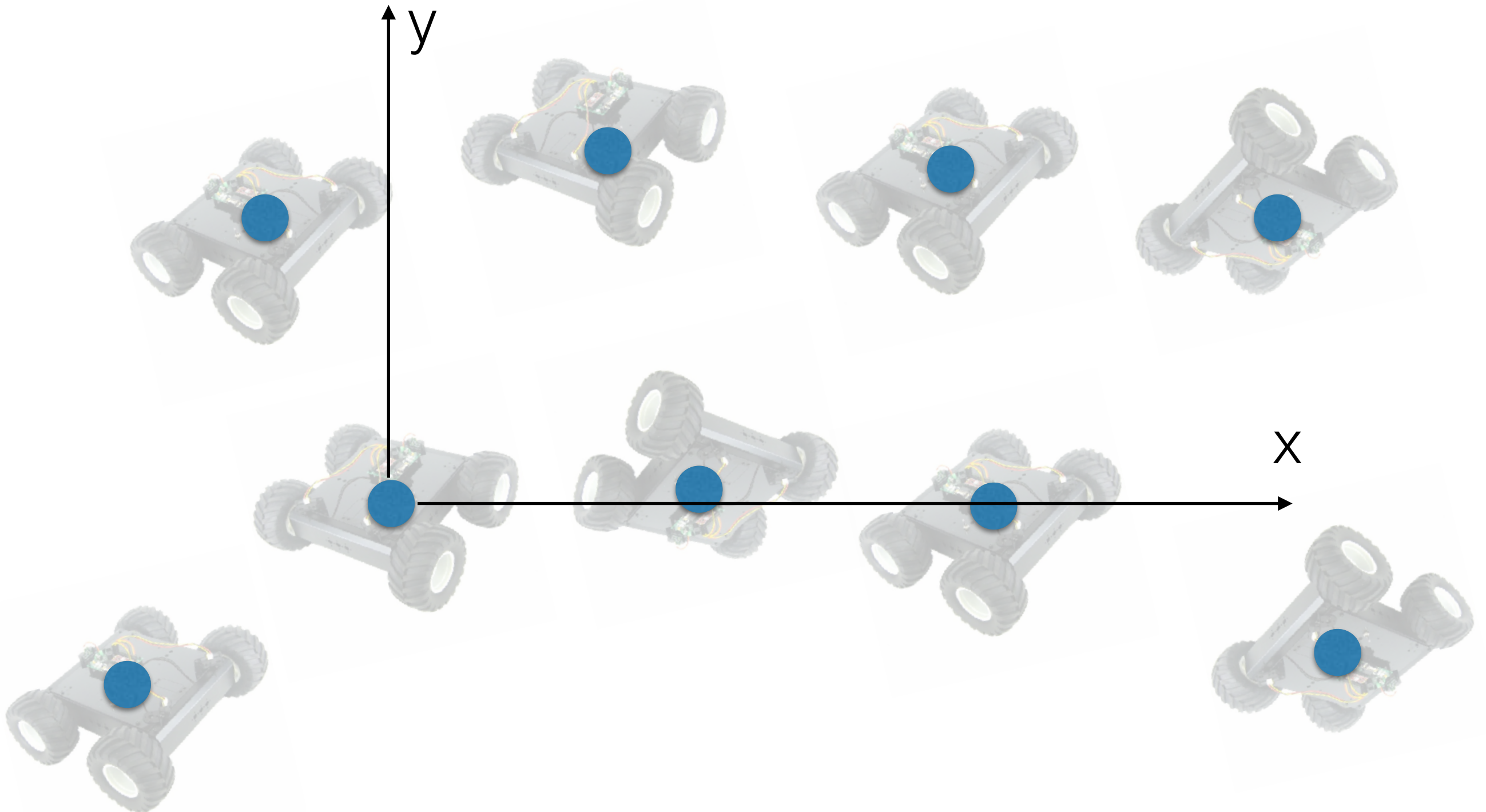
Visibility



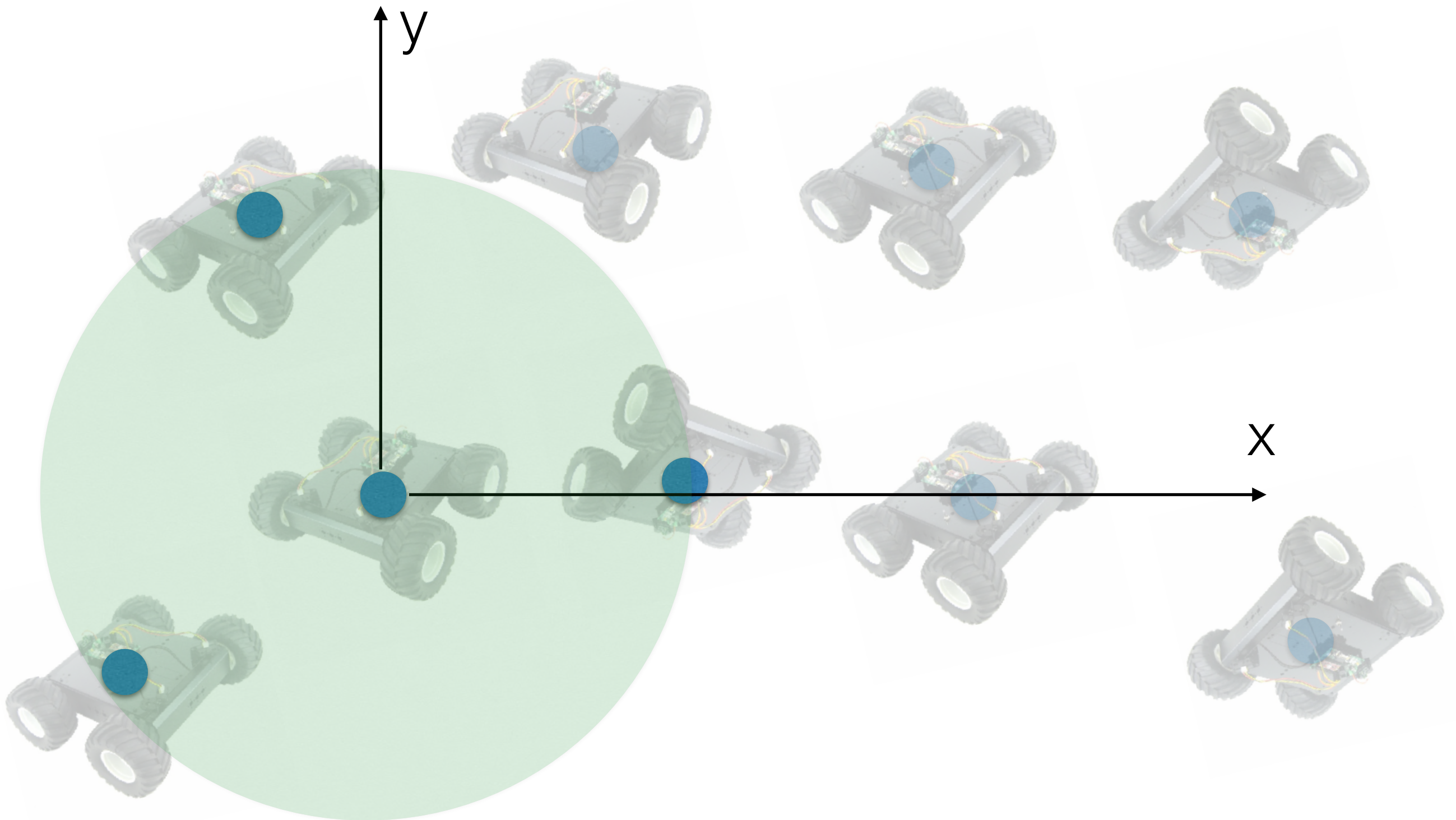
Visibility



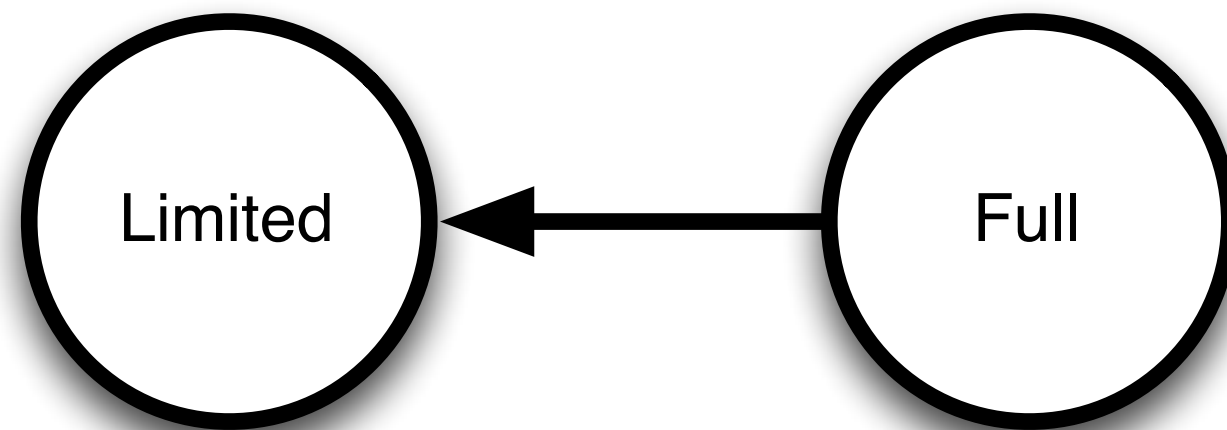
Visibility



Limited Visibility



Visibility



Multiplicity Detection

How many robots do you see?

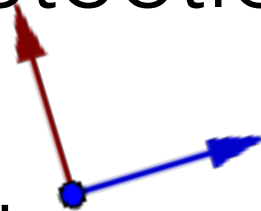
- No detection

1

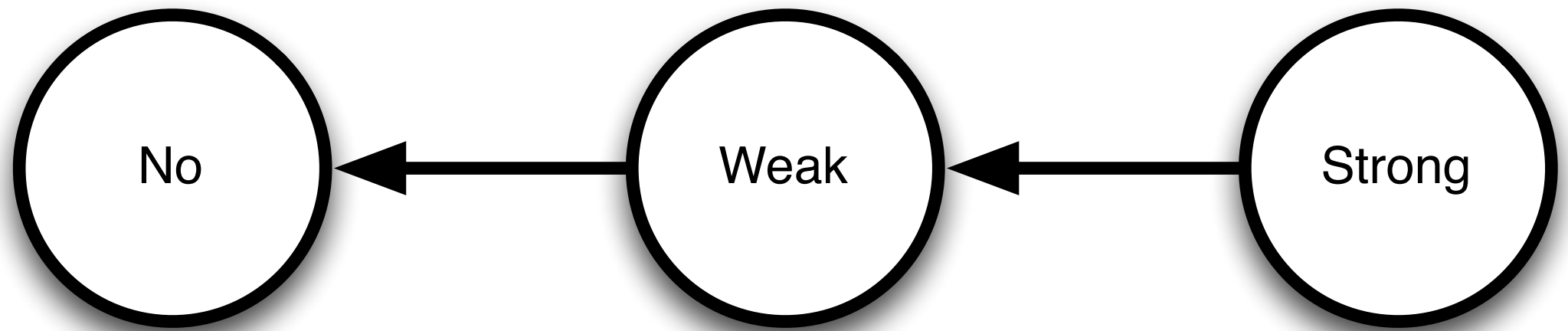
- Weak multiplicity detection

>1

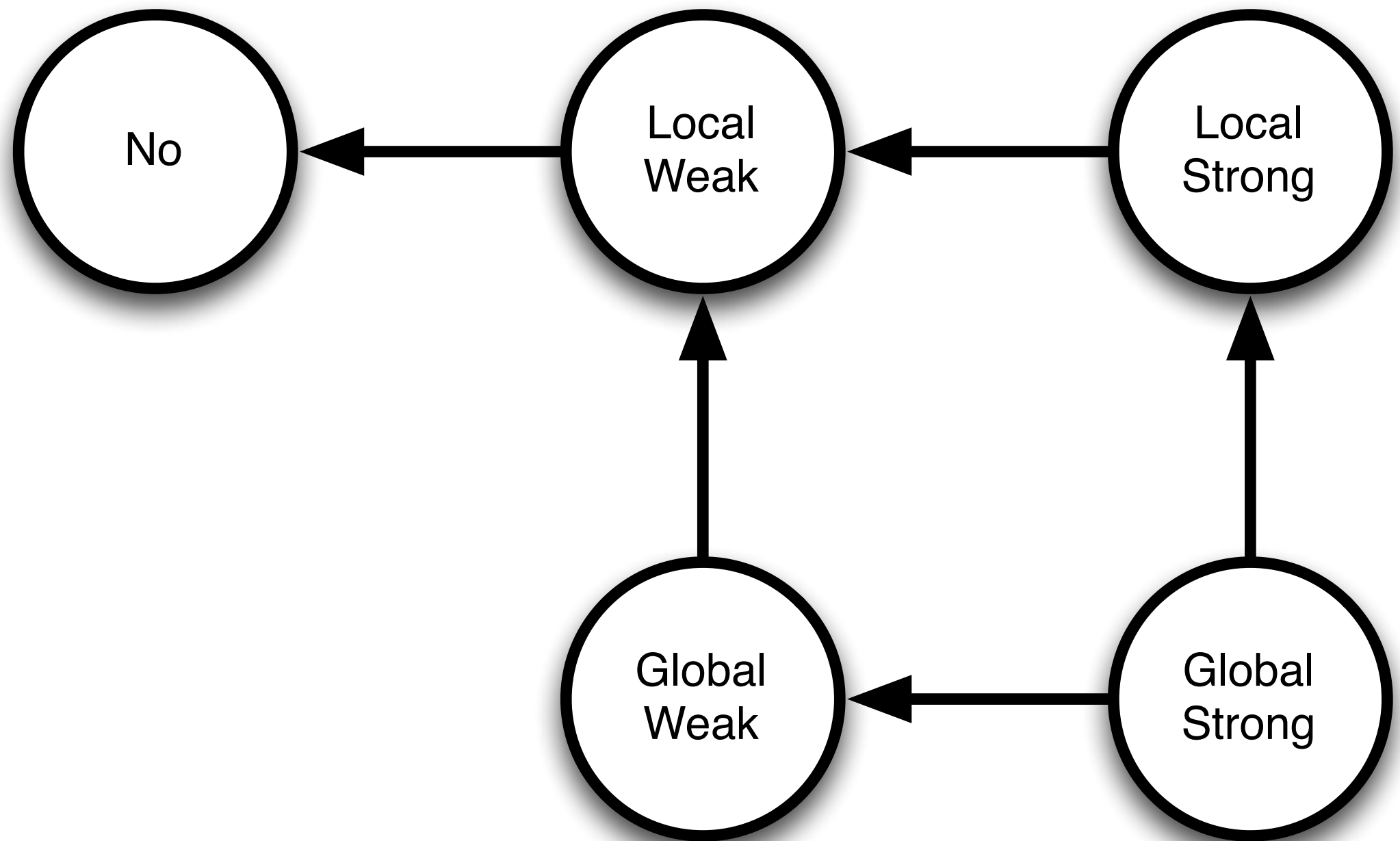
- Strong multiplicity detection



Multiplicity



Multiplicity

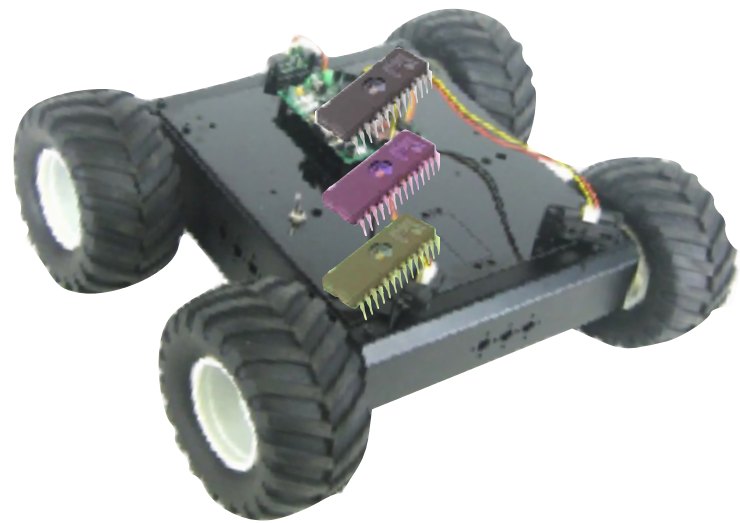


Memory

Algorithm

Persistent Memory

Volatile Memory



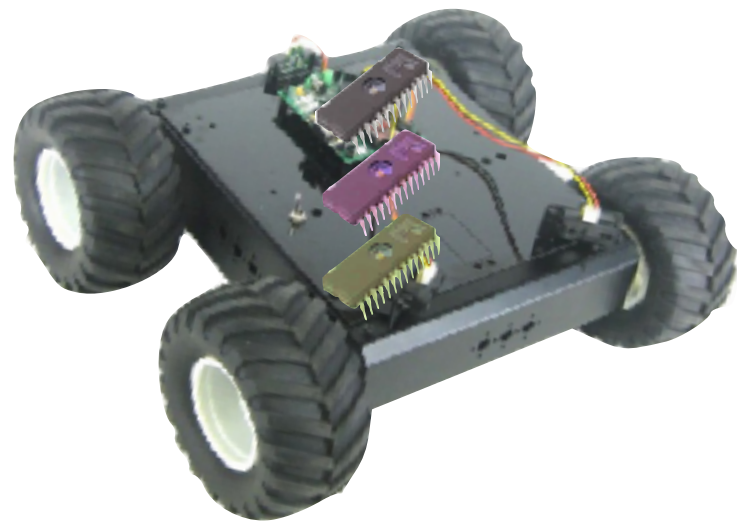
Oblivious Robot Memory

Algorithm

Persistent Memory

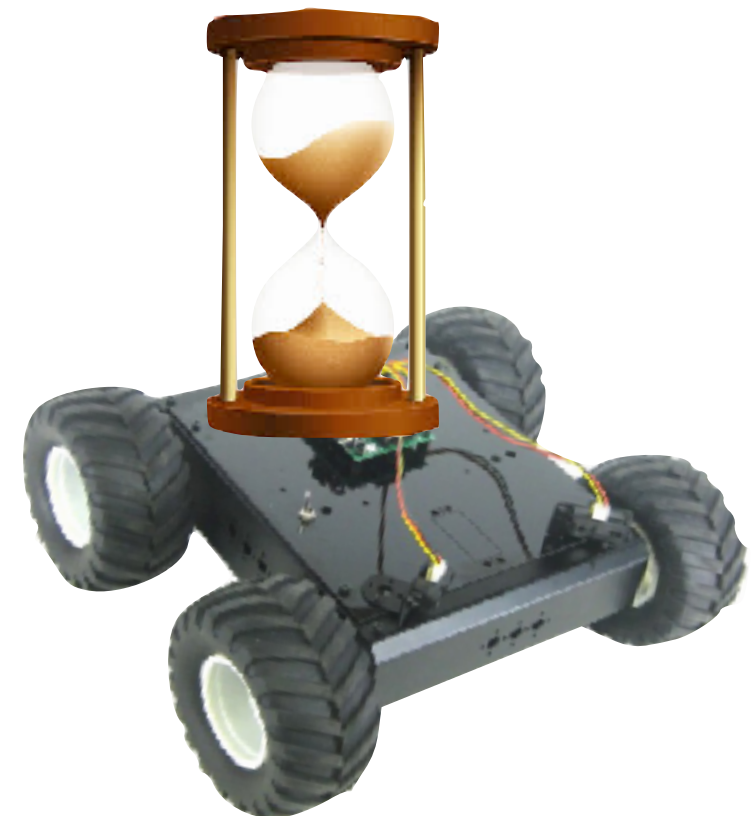
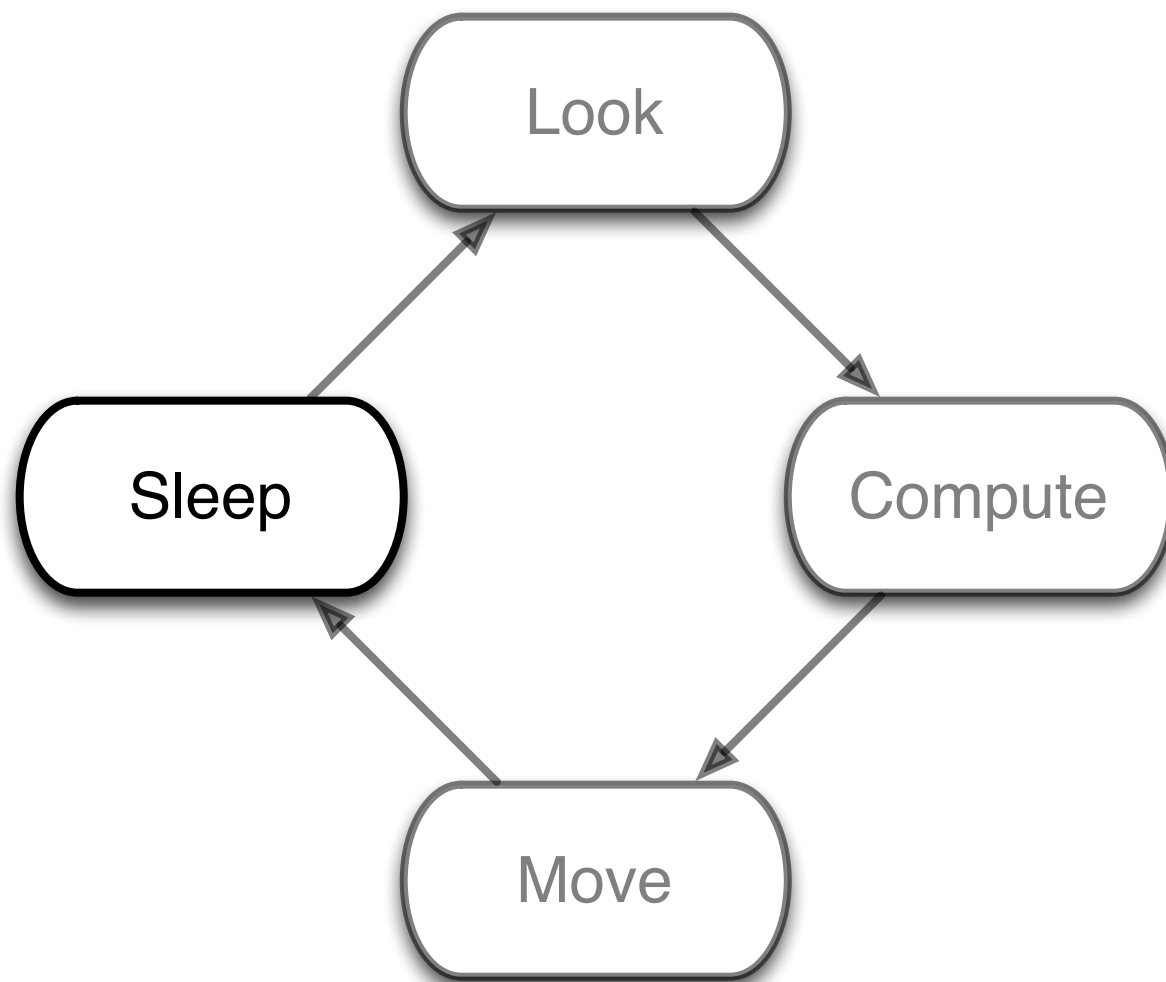


Volatile Memory

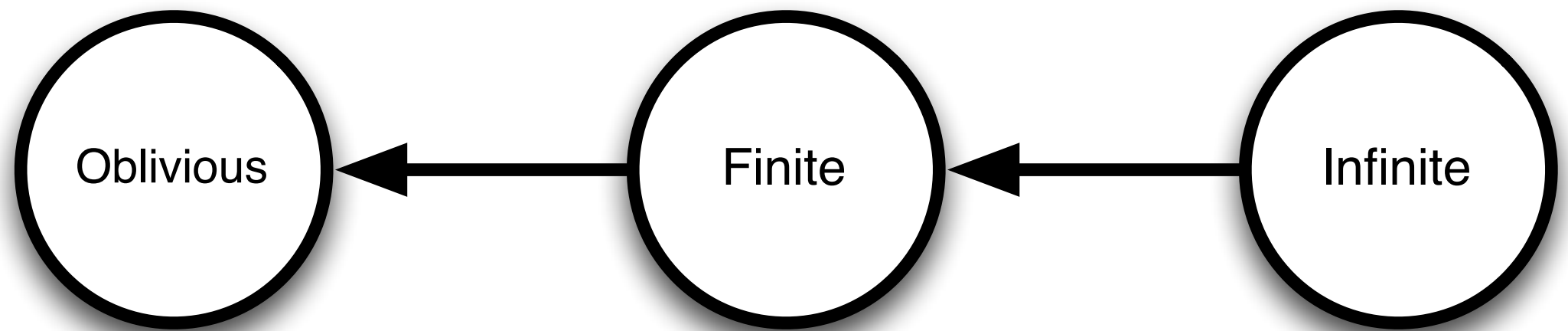


Oblivious Robot Life Cycle

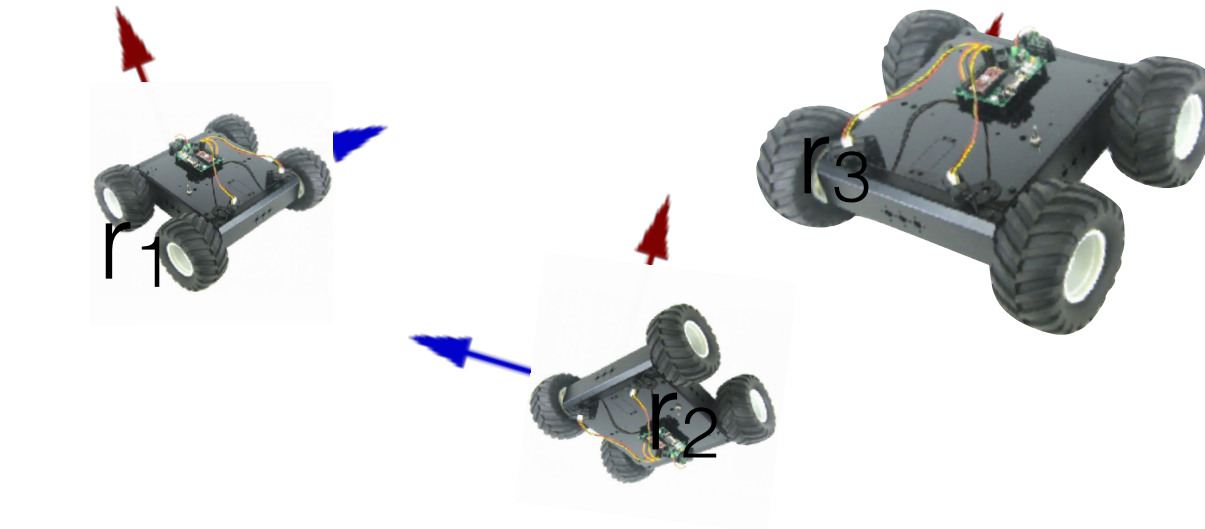
remain **idle** for a while,
forget about the past



Memory



Scheduling

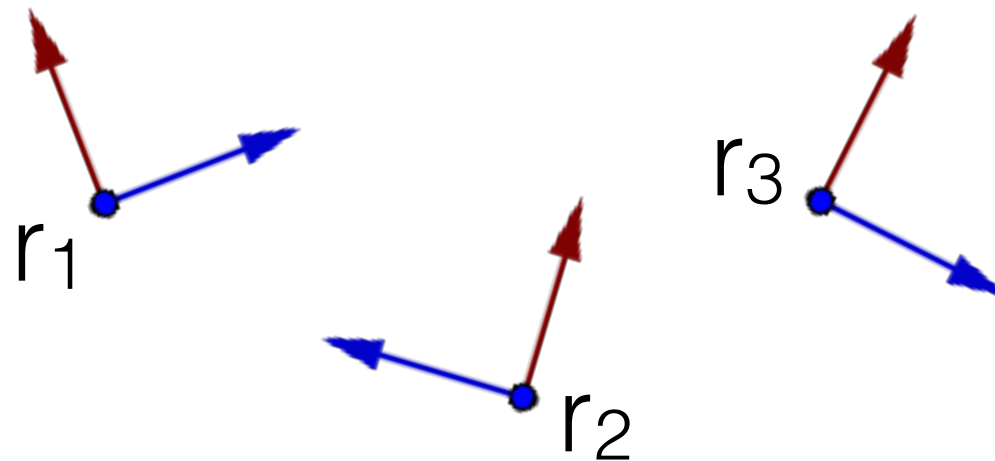


↻ Look → Compute → Move

FSYNC

| | 1 | 2 | 3 | 4 | 5 | 6 |
|-------|---|---|---|---|---|---|
| r_1 | ↻ | ↻ | ↻ | ↻ | ↻ | ↻ |
| r_2 | ↻ | ↻ | ↻ | ↻ | ↻ | ↻ |
| r_3 | ↻ | ↻ | ↻ | ↻ | ↻ | ↻ |

Scheduling

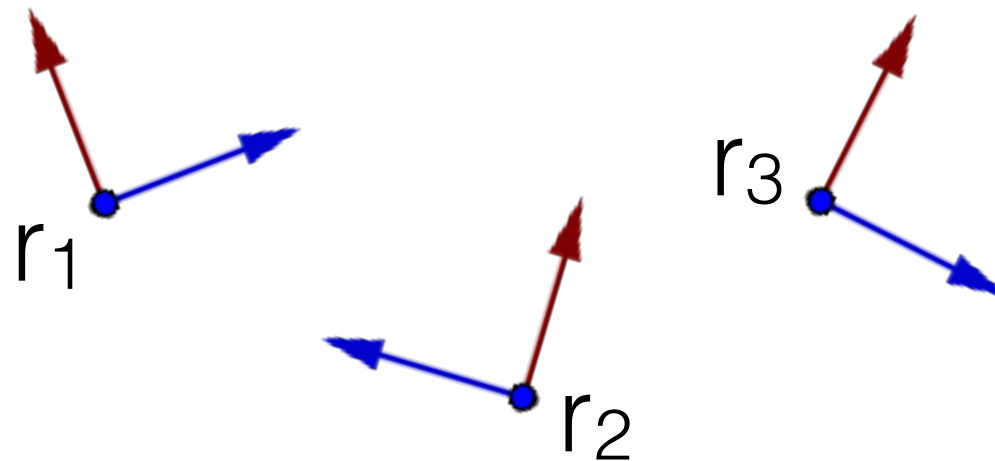


↻ Look → Compute → Move

SSYNC

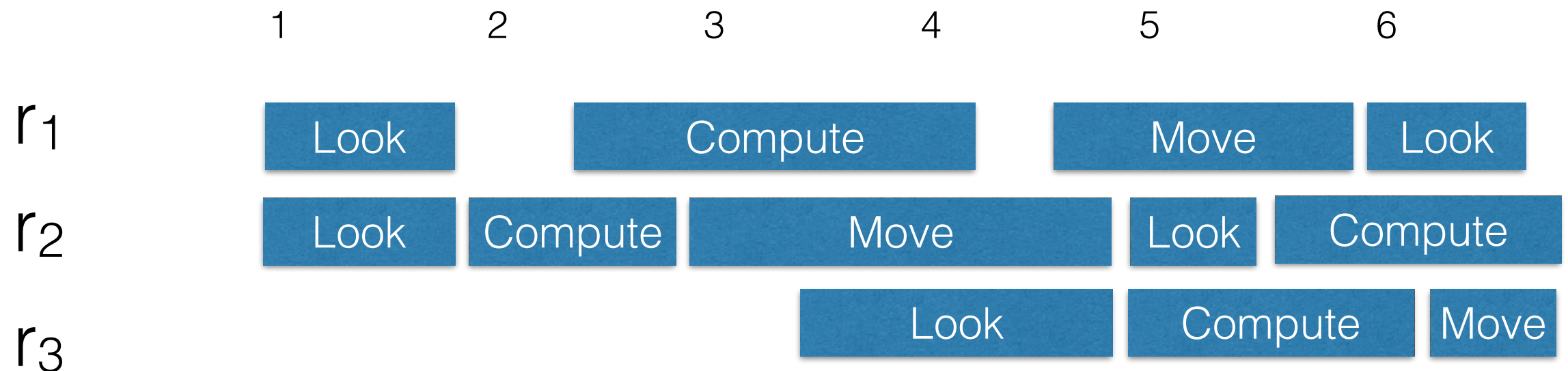
| | 1 | 2 | 3 | 4 | 5 | 6 |
|-------|---|---|---|---|---|---|
| r_1 | ↻ | | ↻ | ↻ | ↻ | ↻ |
| r_2 | ↻ | ↻ | | ↻ | ↻ | |
| r_3 | ↻ | | ↻ | | | ↻ |

Scheduling

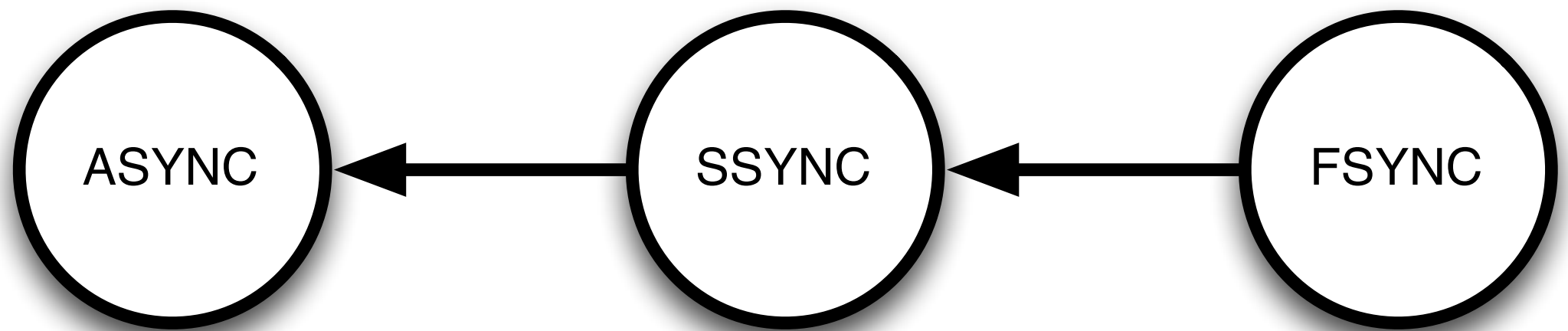


↻ Look → Compute → Move

ASYNC

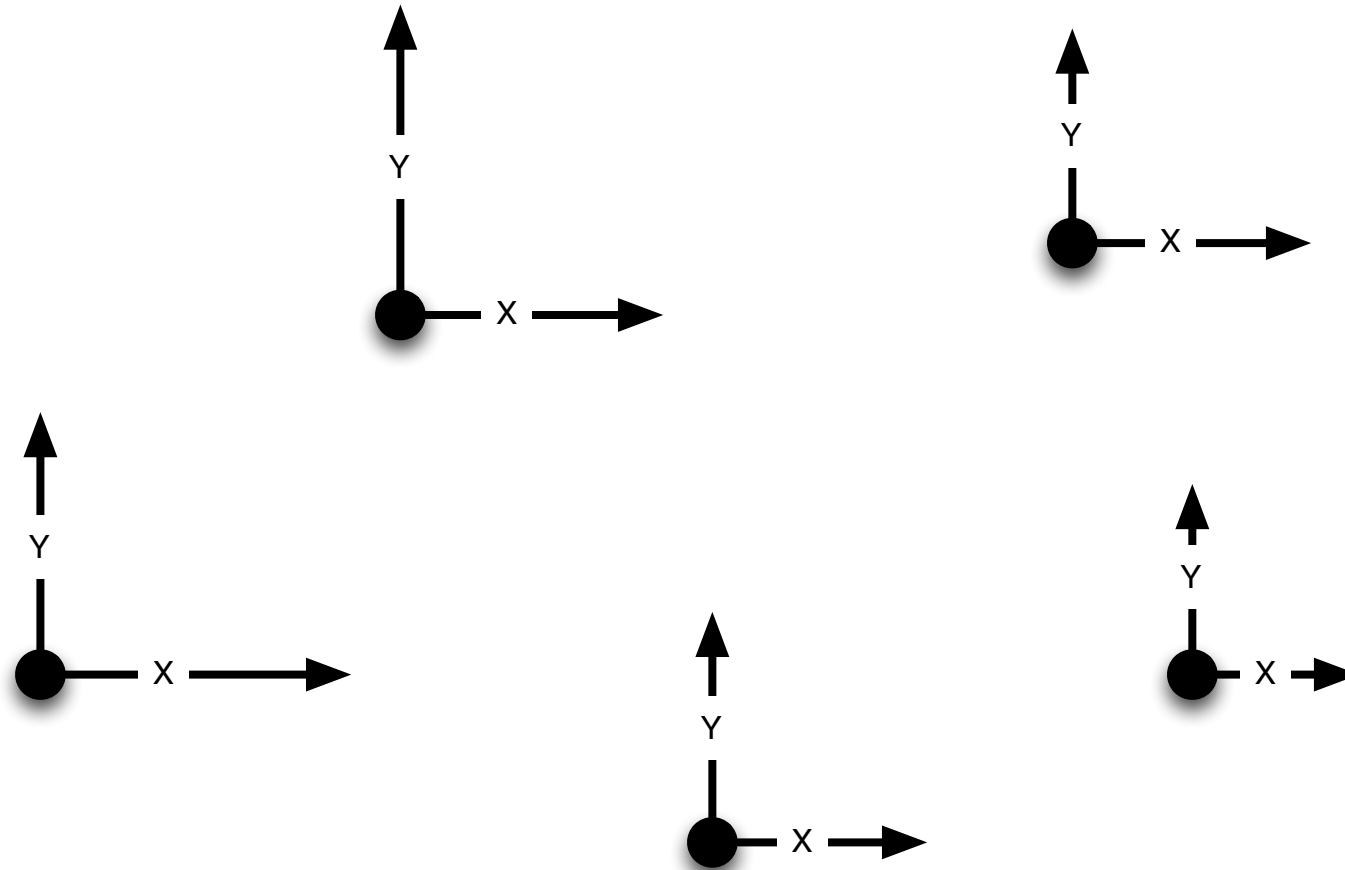


Scheduling



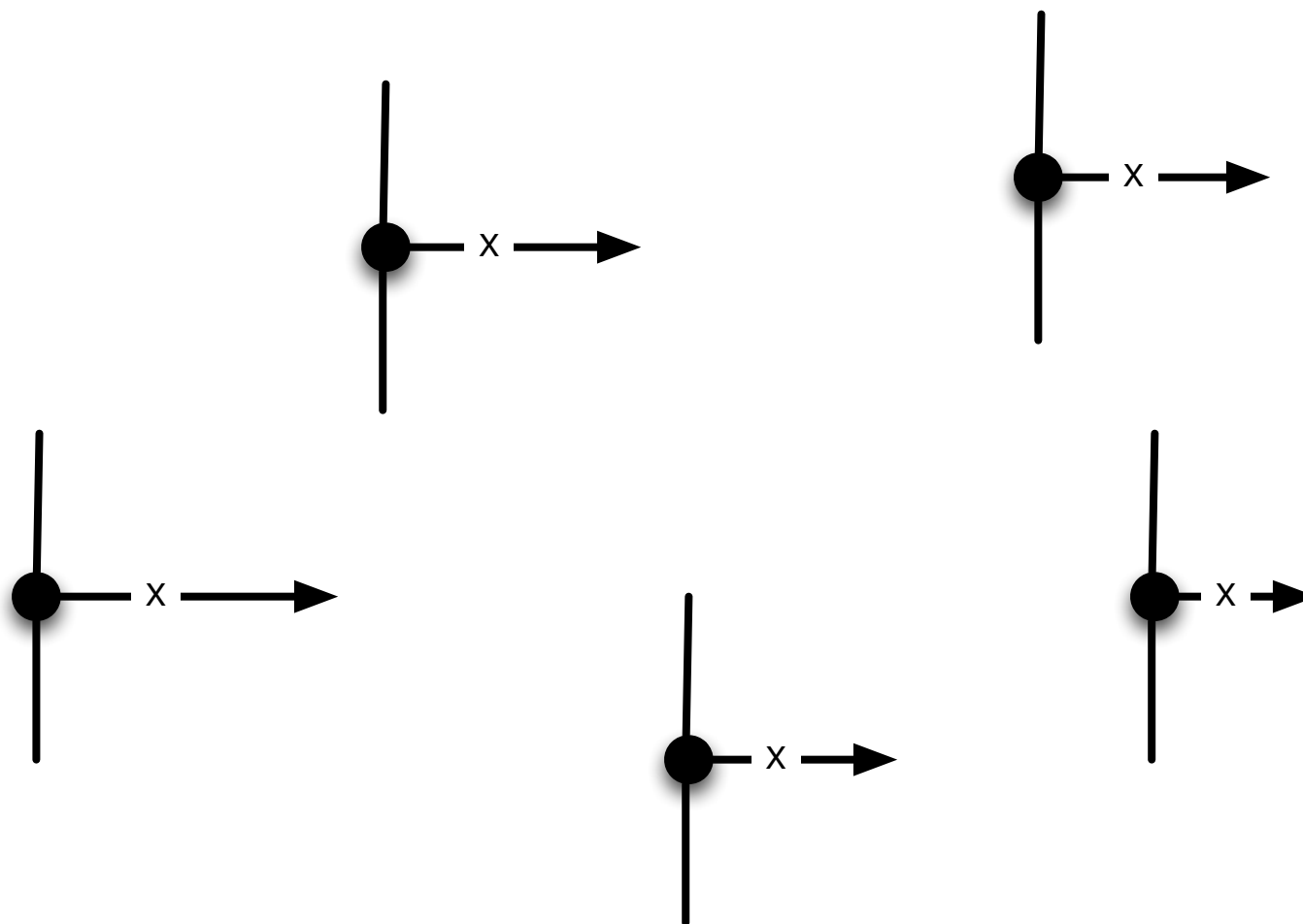
Two Axes

Direction and Orientation

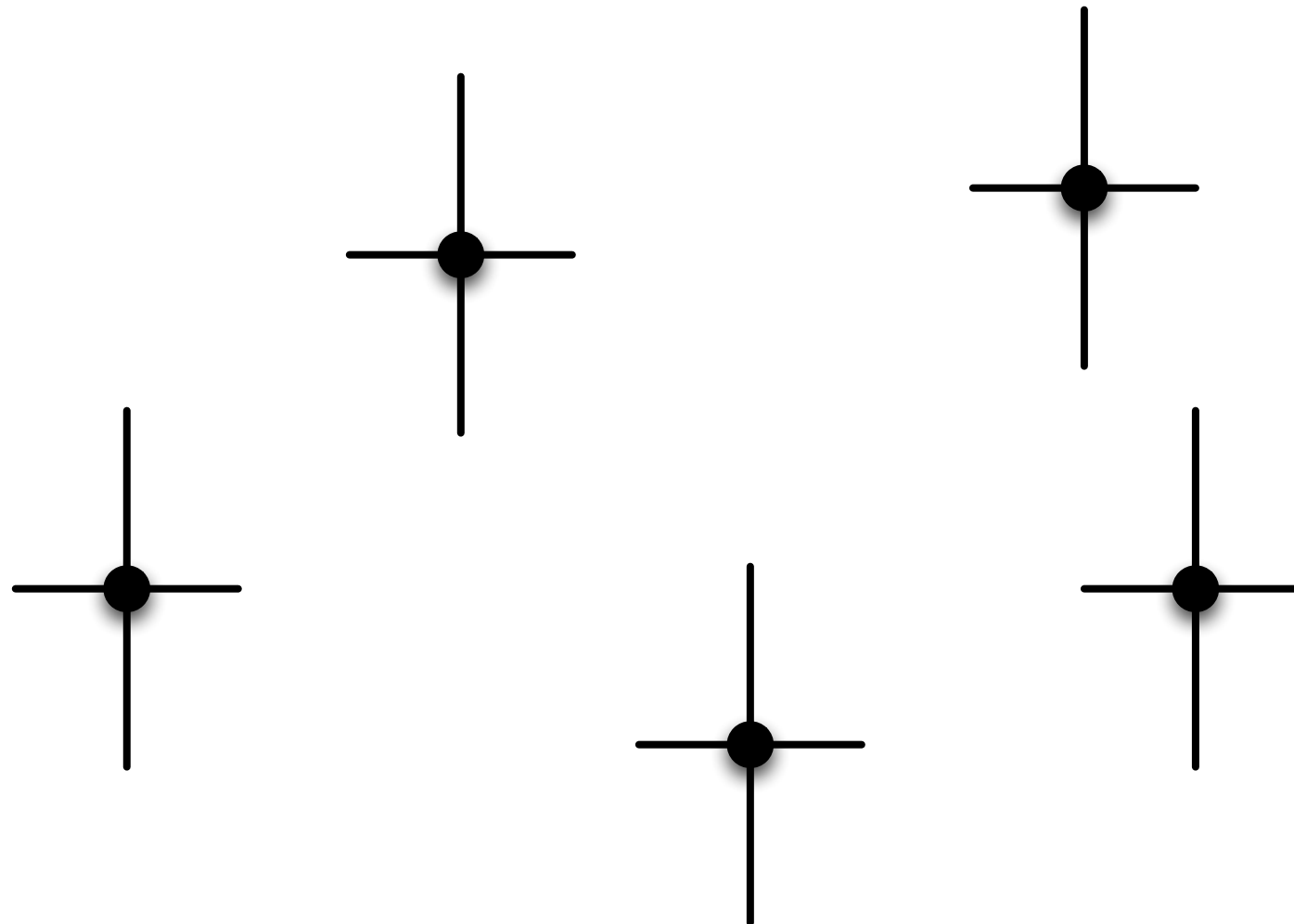


One Axis

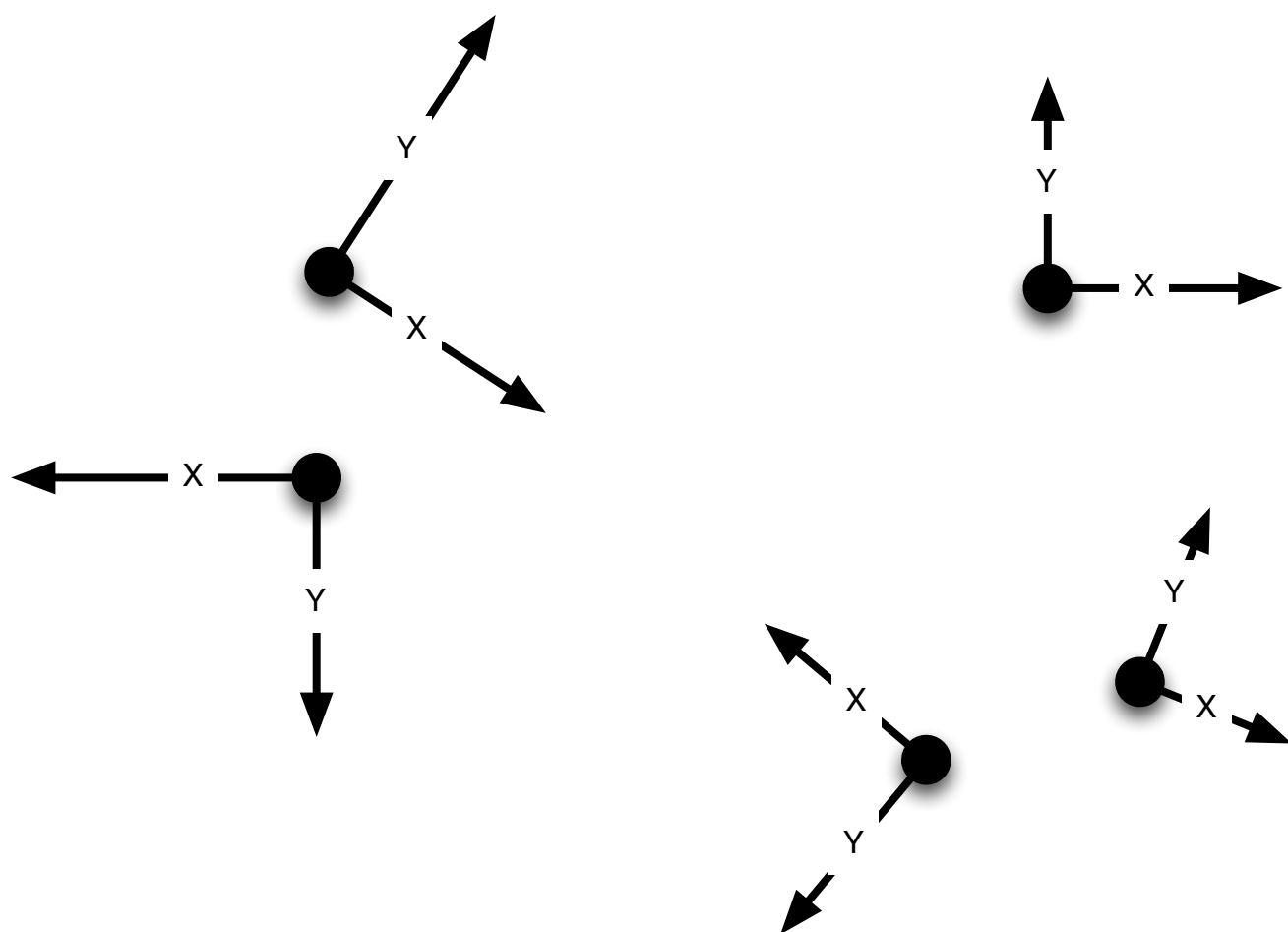
Direction and Orientation



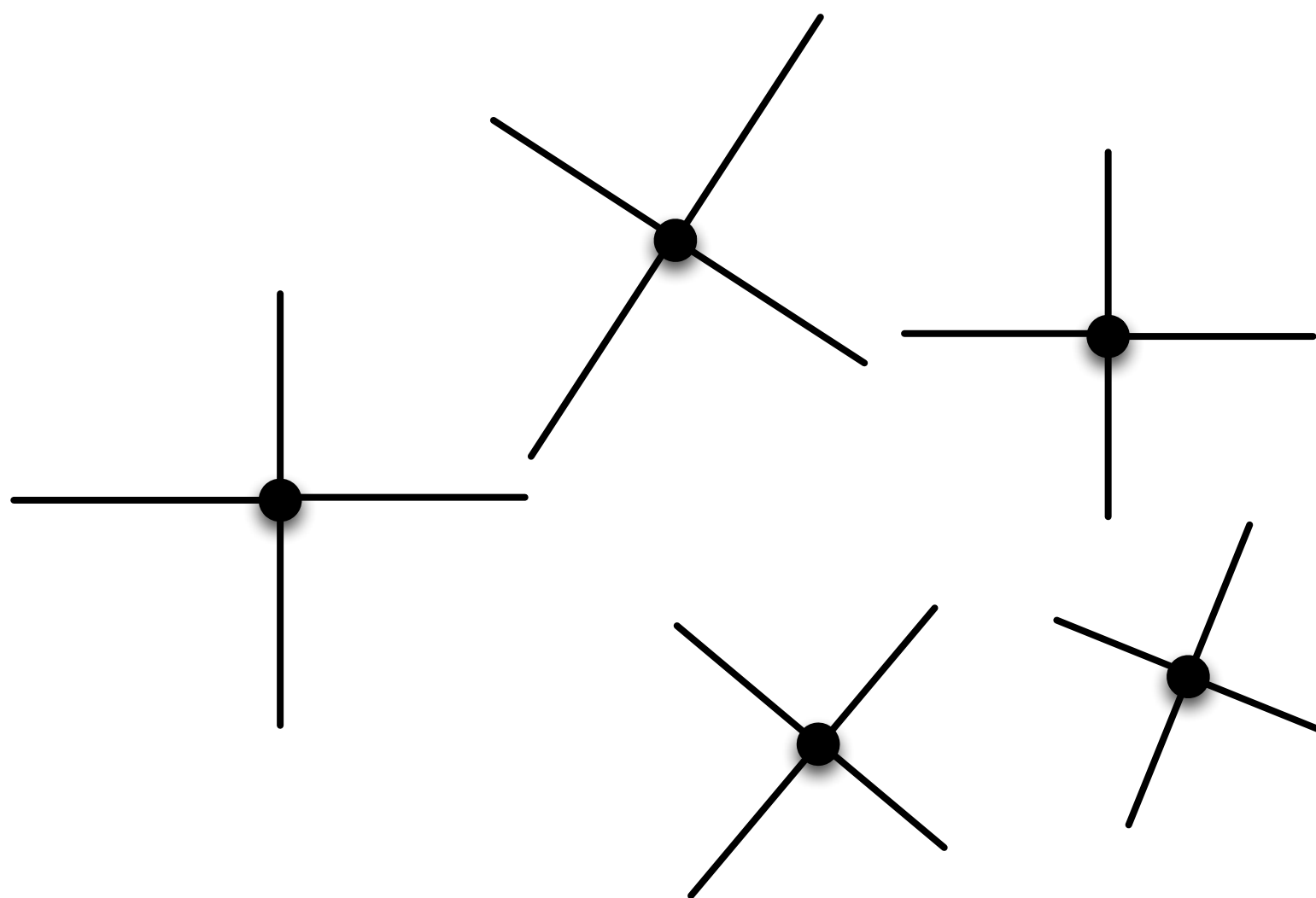
Two Axes Direction



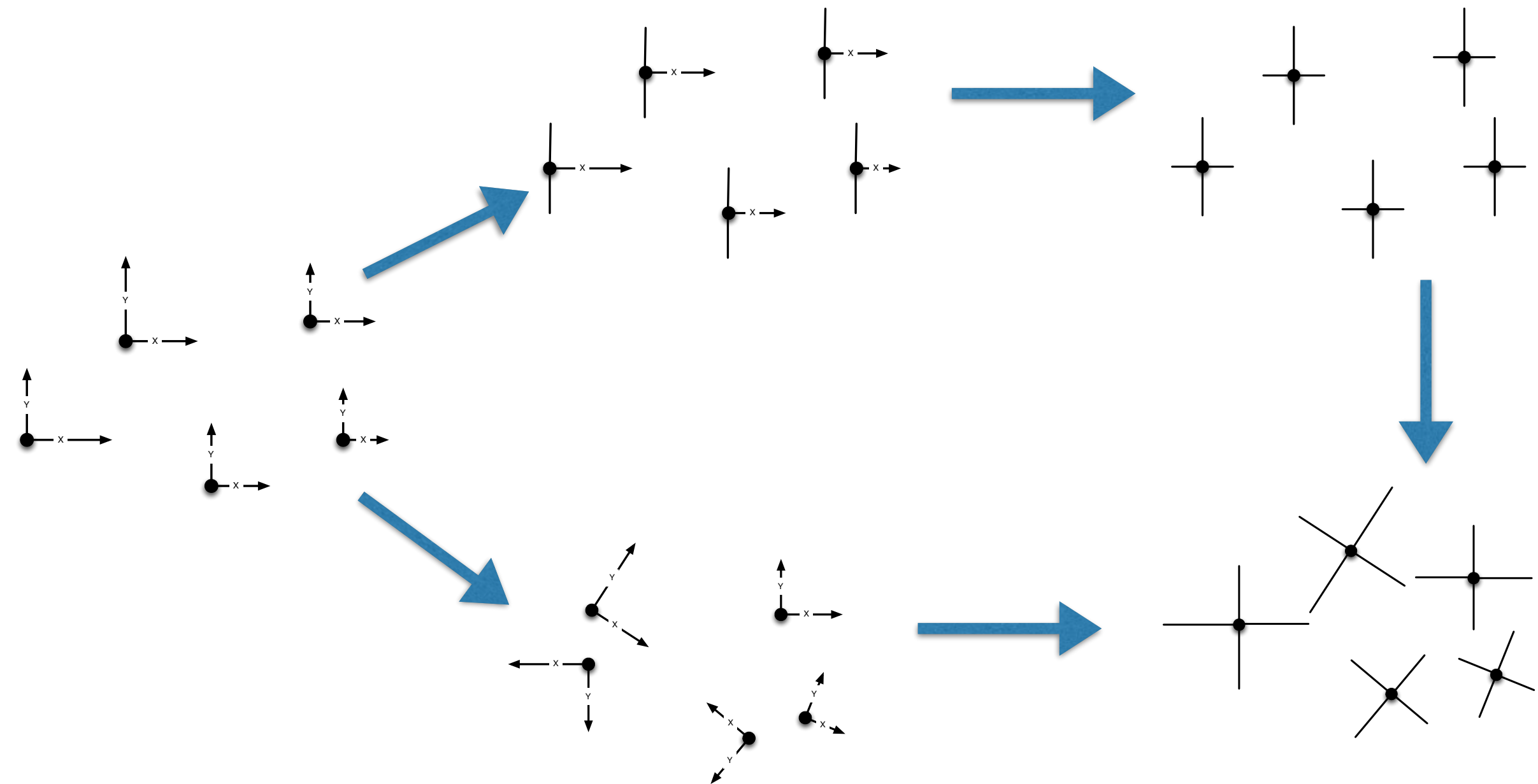
Chirality



No Agreement



Overview

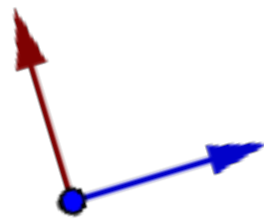


Scattering

Scattering

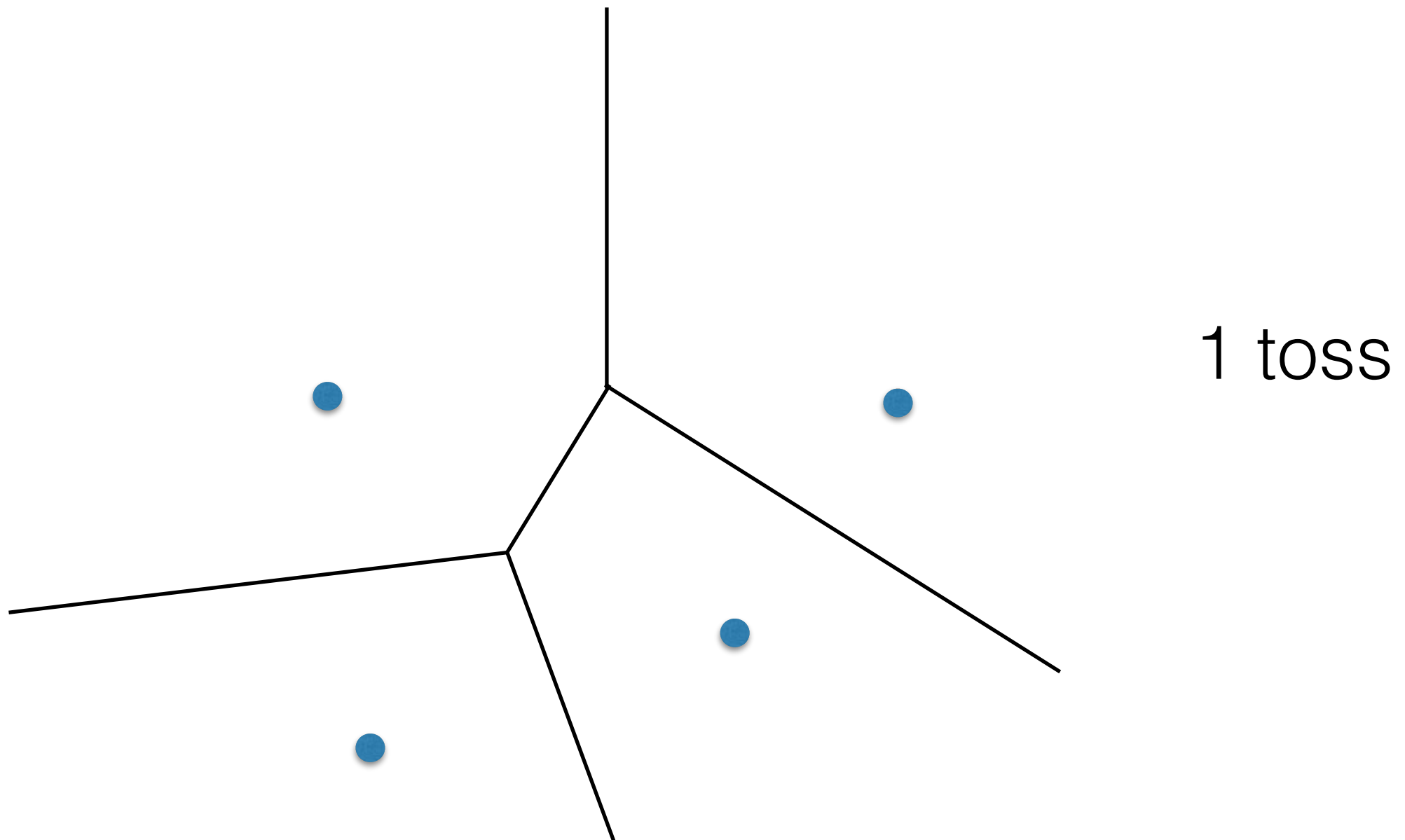
No two robots should occupy the same position

- No deterministic solution



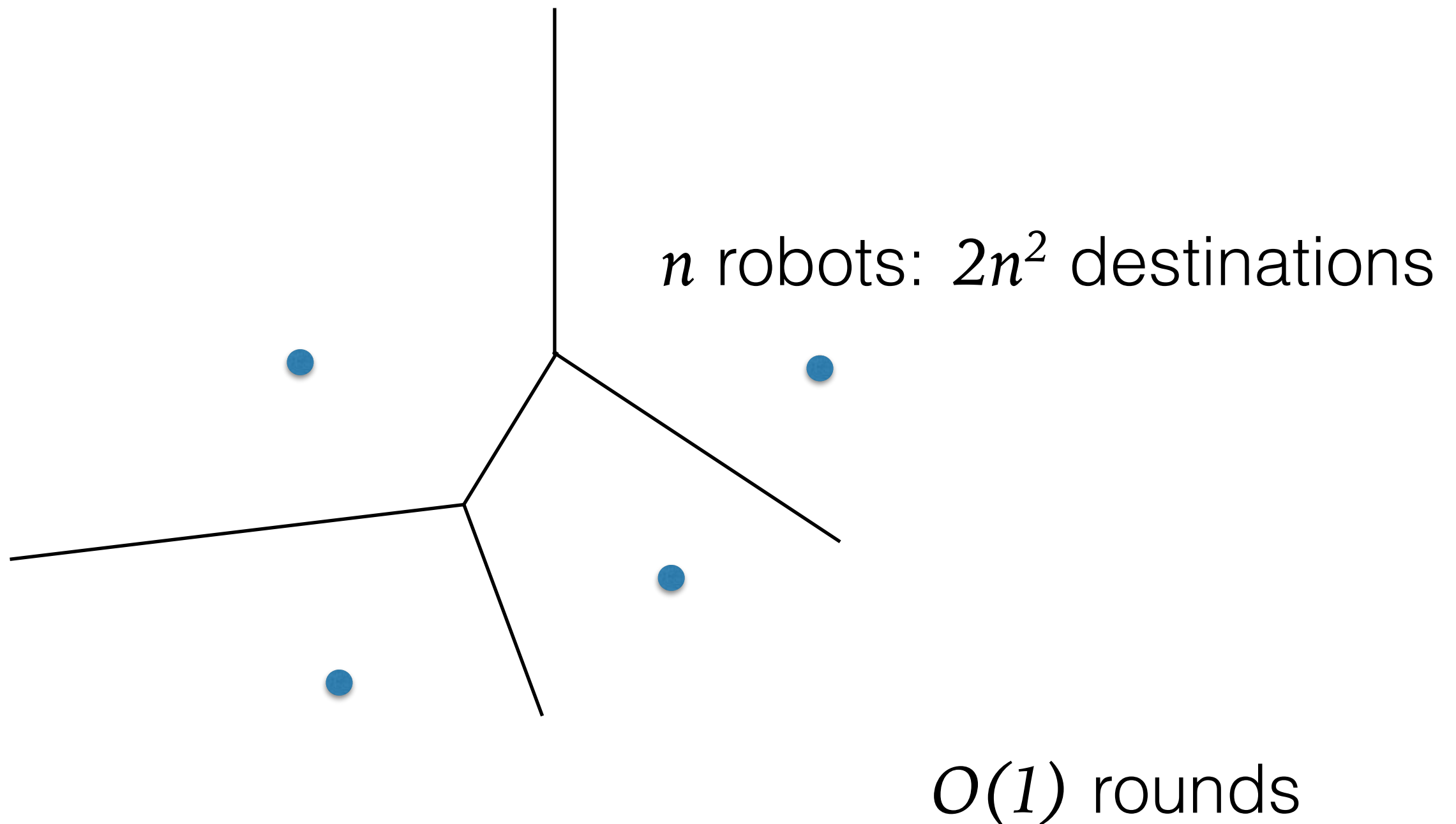
- No termination without multiplicity detection

Scattering



$O(\log(n)\log\log(n))$ rounds

Scattering



How Many Tosses?

Minimum?

Influence of multiplicity detection?

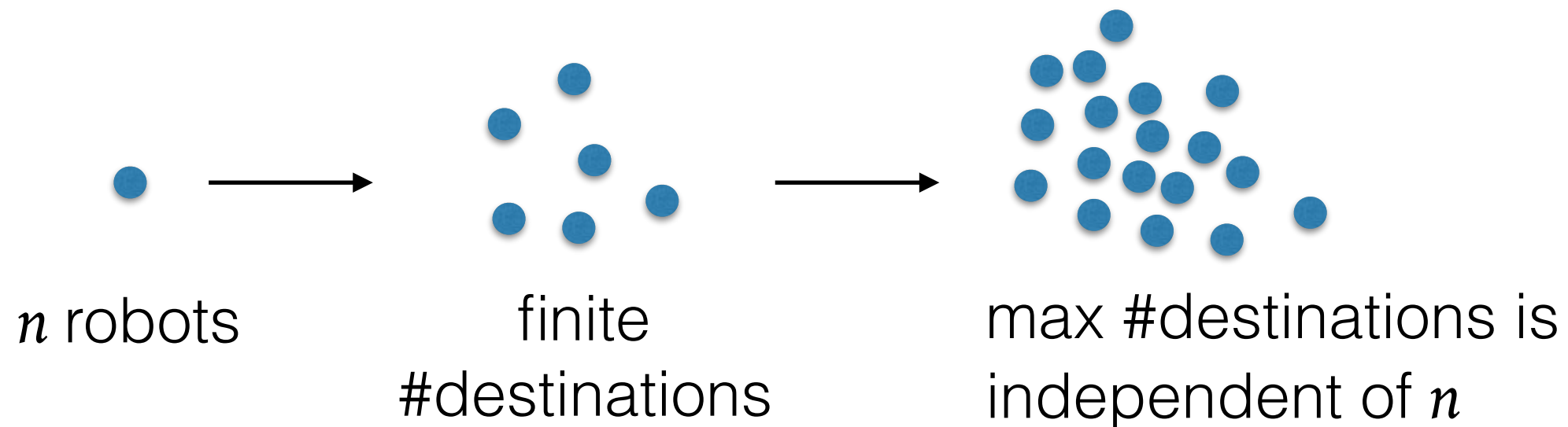
Relationship with scattering speed?

Optimal Speed

With strong multiplicity detection:

Algorithm with optimal #tosses terminates in $O(1)$ rounds

Without strong multiplicity detection:

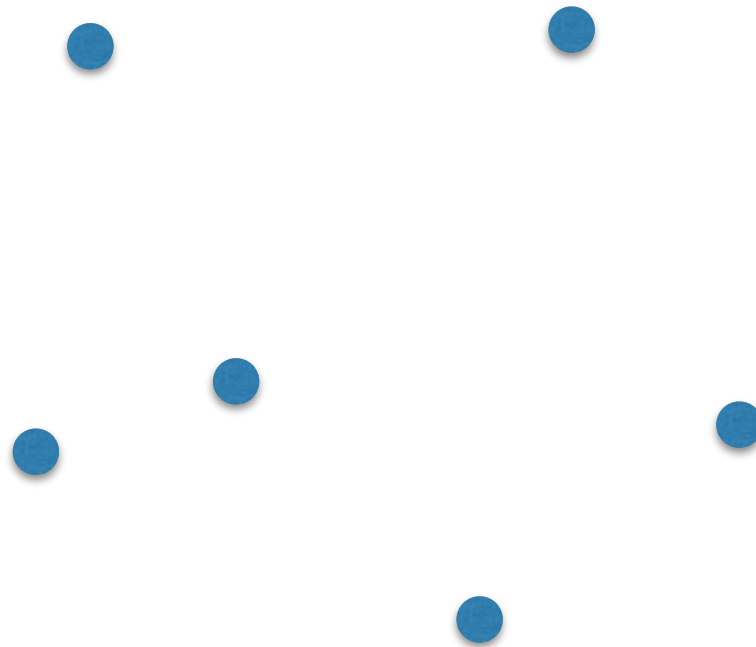


$O(1)$ rounds scattering of n robots is impossible

How fast can we go?

Gathering

Gathering

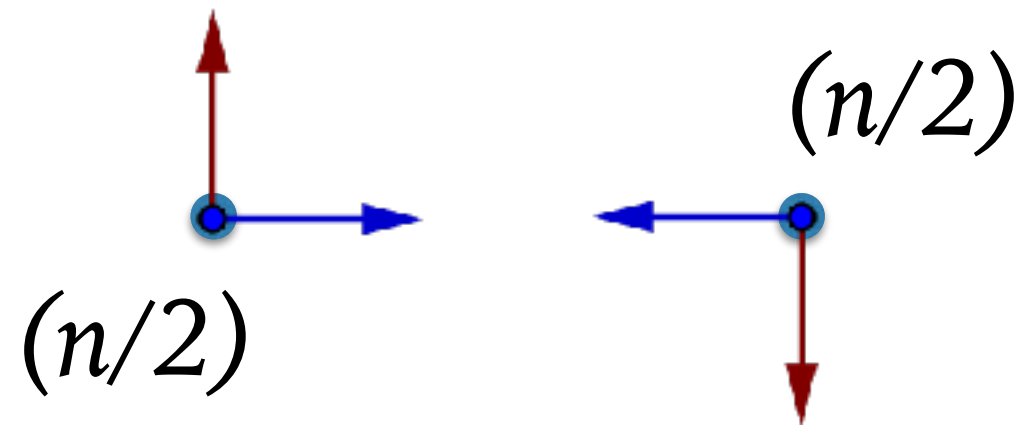


Gathering



Gathering

Impossible for two robots



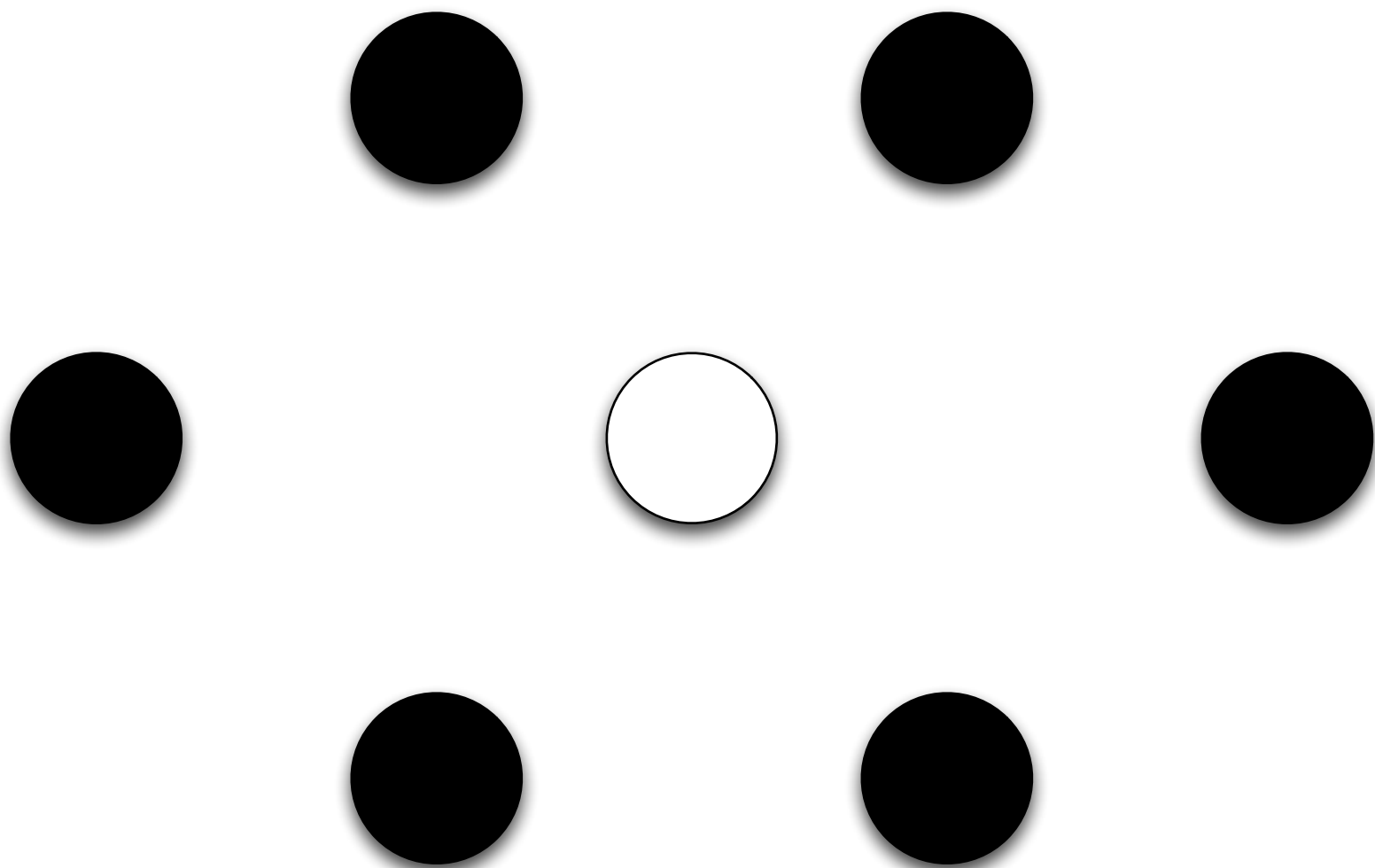
A bivalent configuration

Gathering vs. Convergence

- **Gathering**: robot must **reach** the same point in finite time
- **Convergence**: robots must get **closer** at time goes by

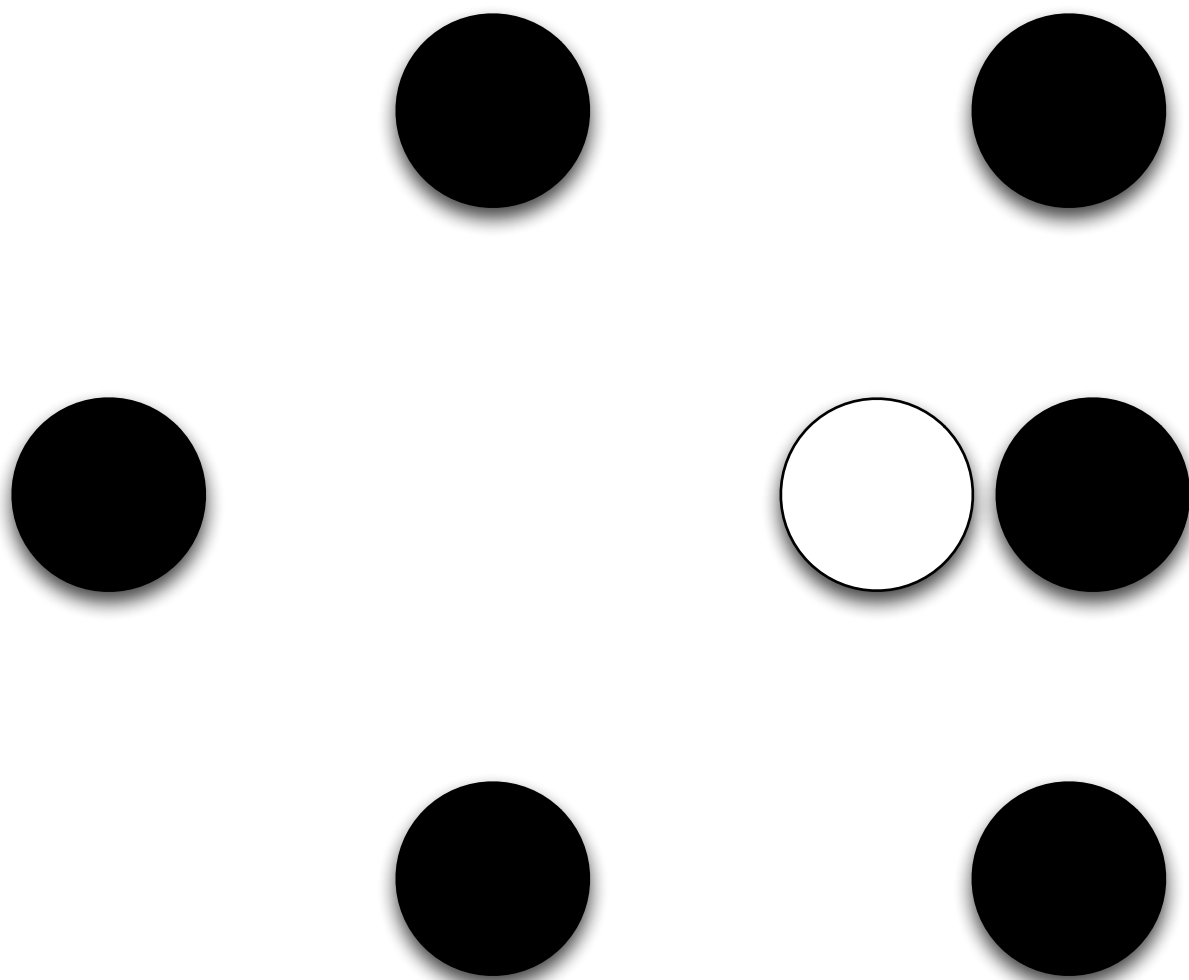
Center of Gravity

$$\vec{c}[t] = \frac{1}{n} \sum_{i=1}^n \vec{r}_i[t]$$



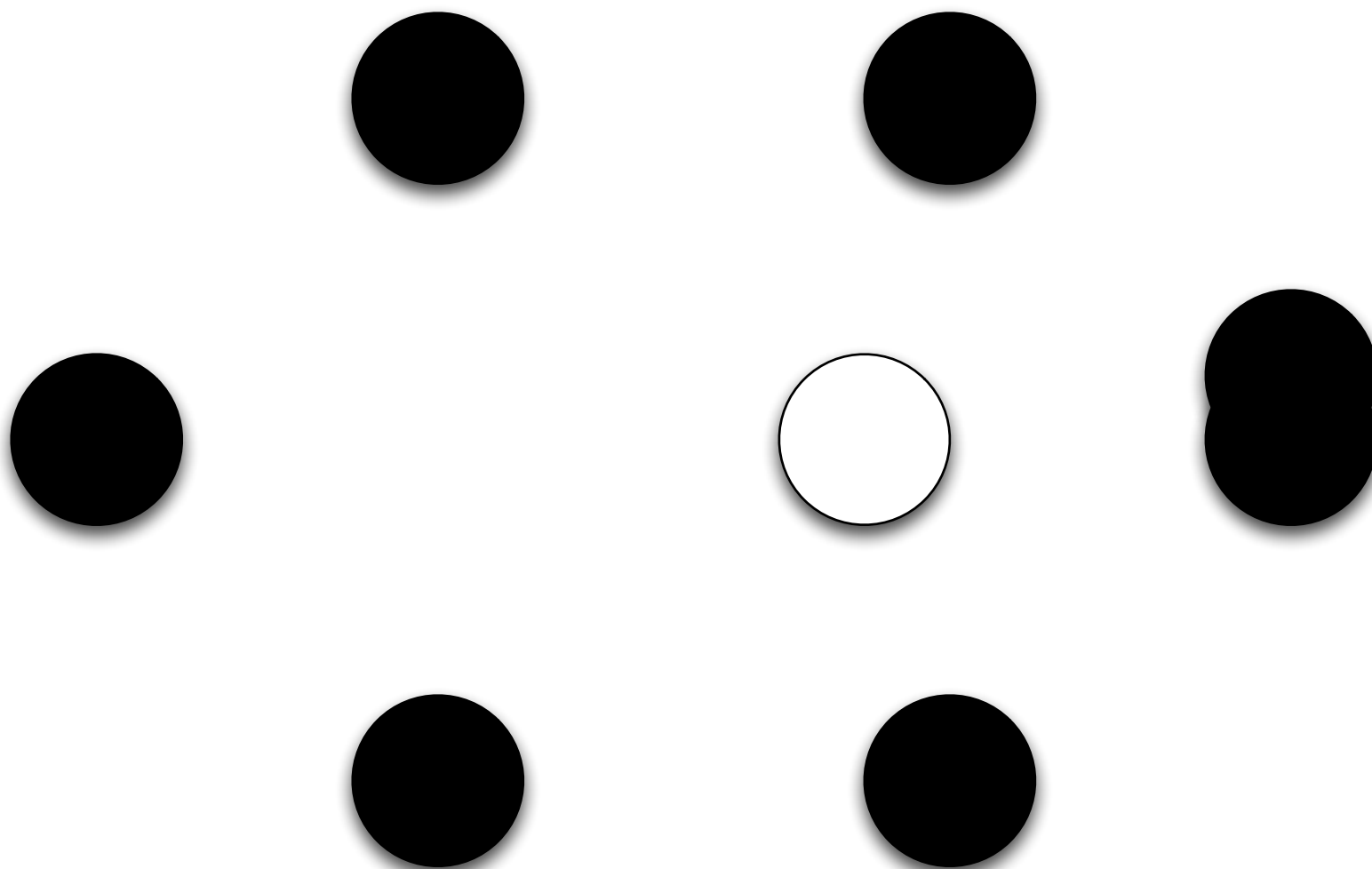
Center of Gravity

$$\vec{c}[t] = \frac{1}{n} \sum_{i=1}^n \vec{r}_i[t]$$



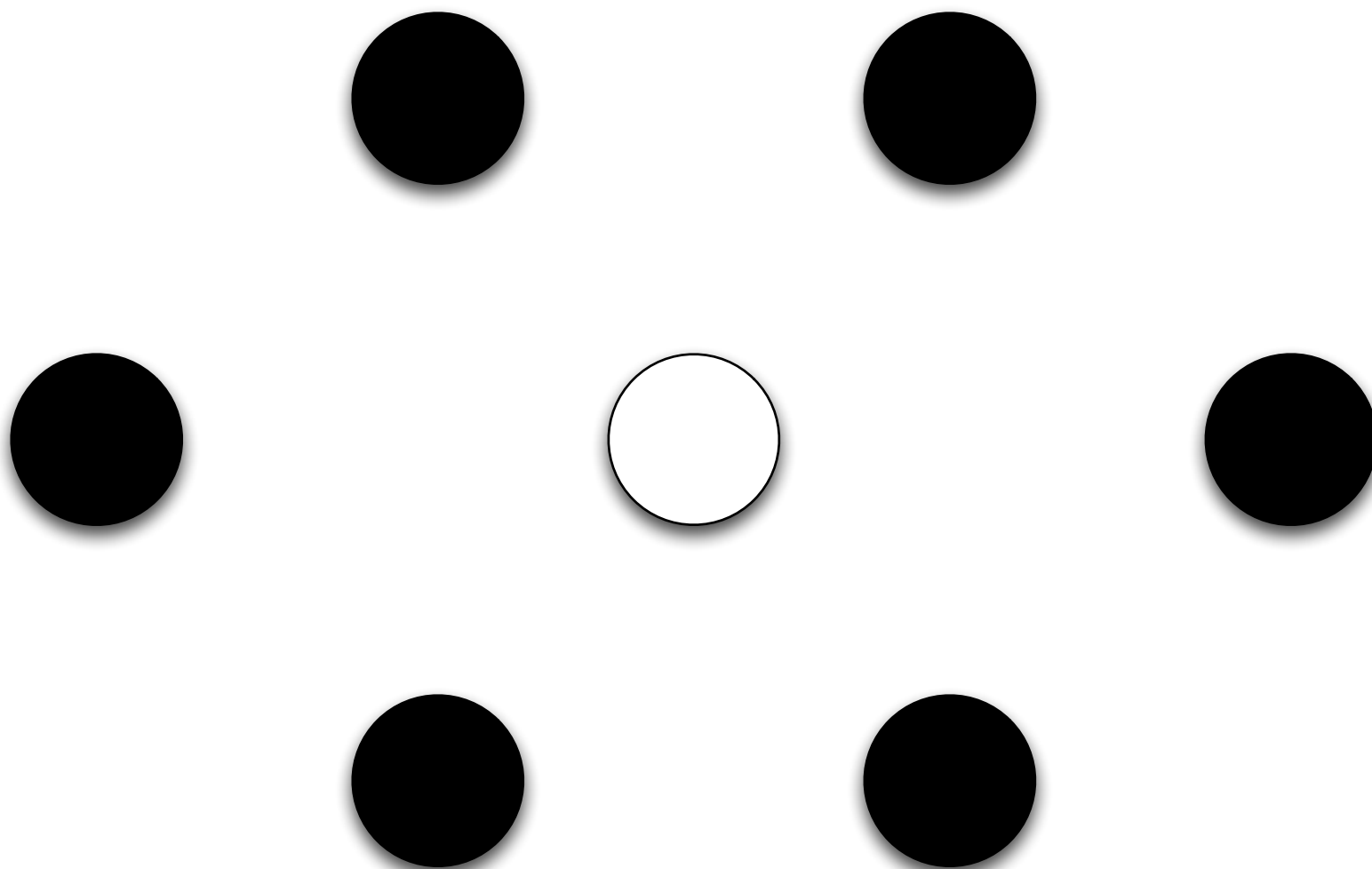
Center of Gravity

$$\vec{c}[t] = \frac{1}{n} \sum_{i=1}^n \vec{r}_i[t]$$



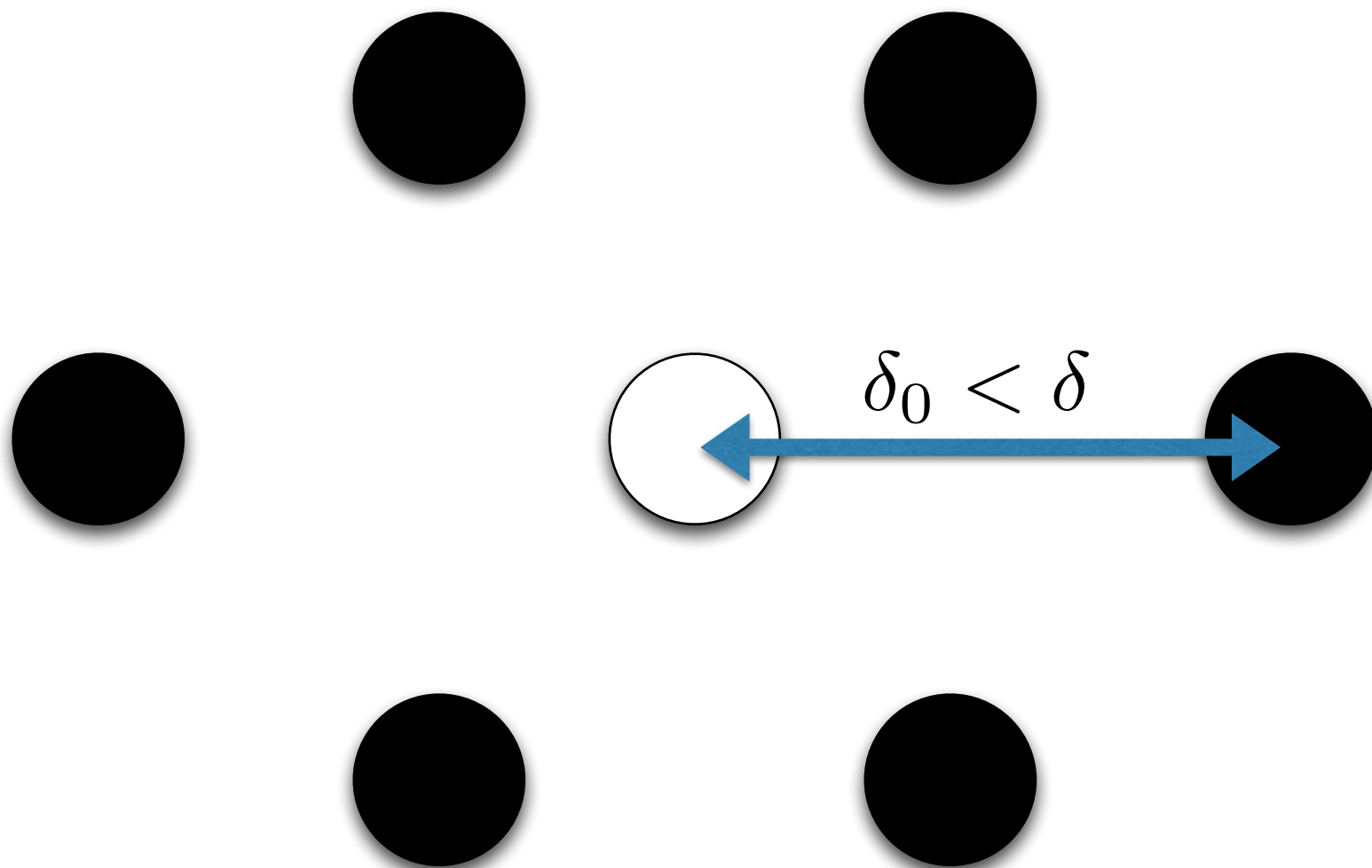
Center of Gravity of Positions

$$\vec{c}[t] = \frac{1}{p} \sum_{i=1}^p \vec{p}_i[t]$$



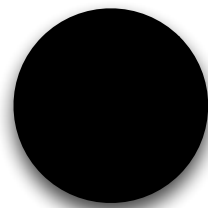
FSYNC Gathering

$$\vec{c}[t] = \frac{1}{p} \sum_{i=1}^p \vec{p}_i[t]$$

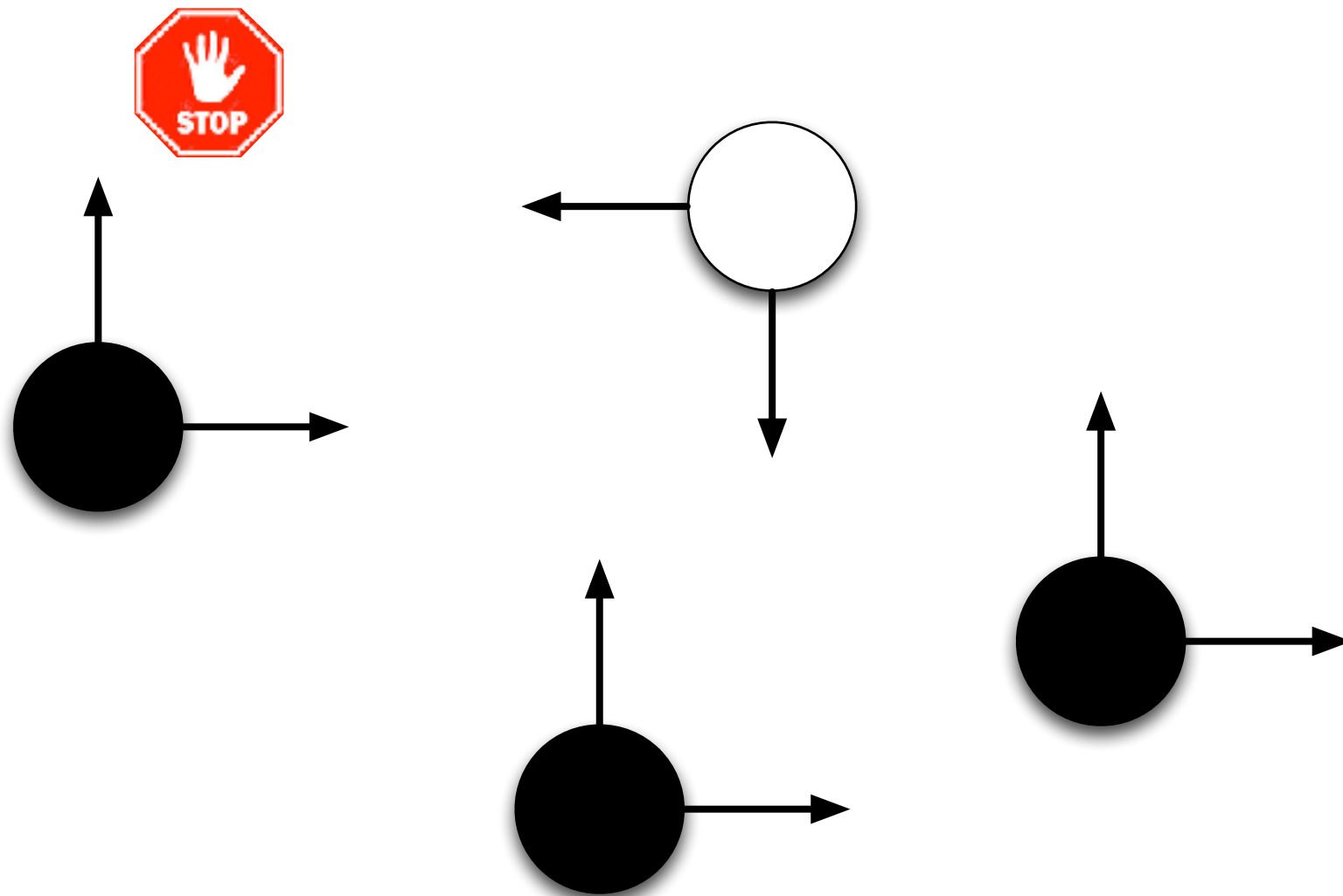


FSYNC Gathering

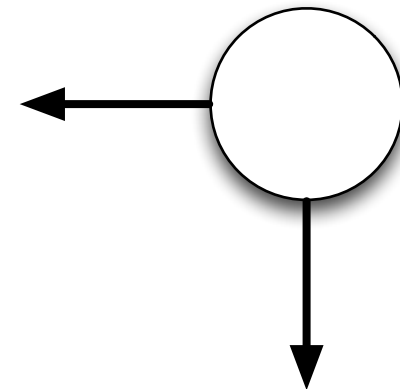
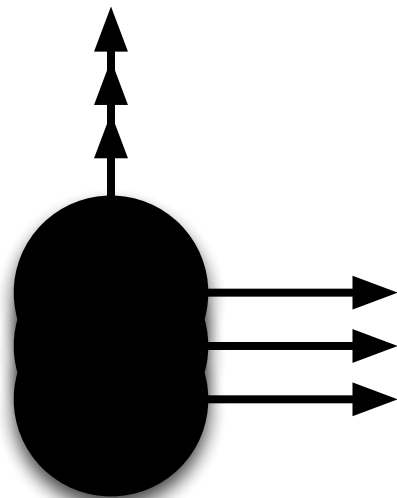
$$\vec{c}[t] = \frac{1}{p} \sum_{i=1}^p \vec{p}_i[t]$$



SSYNC Gathering?



SSYNC Gathering?



Convergence & Gathering

| | Convergence | 2-Gathering | n-Gathering | n-Gathering +MD | n-Gathering +MD+WF |
|-------|-------------|-------------|-------------|--------------------|-----------------------|
| FSYNC | Yes | Yes | Yes | Yes | Yes |
| SSYNC | Yes | No | No | Yes | Yes |
| ASYNC | Yes | No | No | Yes | ? |

Convergence & Gathering

| | Convergence | 2-Gathering | n-Gathering | n-Gathering +MD | n-Gathering +MD+WF |
|-------|-------------|-------------|-------------|--------------------|-----------------------|
| FSYNC | Yes | Yes | Yes | Yes | Yes |
| SSYNC | Yes | No | No | Yes | Yes |
| ASync | Yes | No | No | Yes | ? |

Reuven Cohen and David Peleg. *Convergence Properties of the Gravitational Algorithm in Asynchronous Robot Systems*. SIAM J. Comput. 34(6): 1516-1528 (2005)

Convergence & Gathering

| | Convergence | 2-Gathering | n-Gathering | n-Gathering +MD | n-Gathering +MD+WF |
|-------|-------------|-------------|-------------|--------------------|-----------------------|
| FSYNC | Yes | Yes | Yes | Yes | Yes |
| SSYNC | Yes | No | No | Yes | Yes |
| ASync | Yes | No | No | Yes | ? |

Ichiro Suzuki, Masafumi Yamashita: Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. SIAM J. Comput. 28(4): 1347-1363 (1999)

Convergence & Gathering

| | Convergence | 2-Gathering | n-Gathering | n-Gathering +MD | n-Gathering +MD+WF |
|-------|-------------|-------------|-------------|--------------------|-----------------------|
| FSYNC | Yes | Yes | Yes | Yes | Yes |
| SSYNC | Yes | No | No | Yes | Yes |
| ASync | Yes | No | No | Yes | ? |

Guisepppe Prencipe. *Impossibility of gathering by a set of autonomous mobile robots.*
Theor. Comput. Sci. 384(2-3): 222-231 (2007)

Convergence & Gathering

| | Convergence | 2-Gathering | n-Gathering | n-Gathering +MD | n-Gathering +MD+WF |
|-------|-------------|-------------|-------------|--------------------|-----------------------|
| FSYNC | Yes | Yes | Yes | Yes | Yes |
| SSYNC | Yes | No | No | Yes | Yes |
| ASync | Yes | No | No | Yes | ? |

Thibaut Balabonski, Amélie Delga, Lionel Rieg, Sébastien Tixeuil, Xavier Urbain:
Synchronous Gathering Without Multiplicity Detection: A Certified Algorithm. SSS
2016: 7-19

Convergence & Gathering

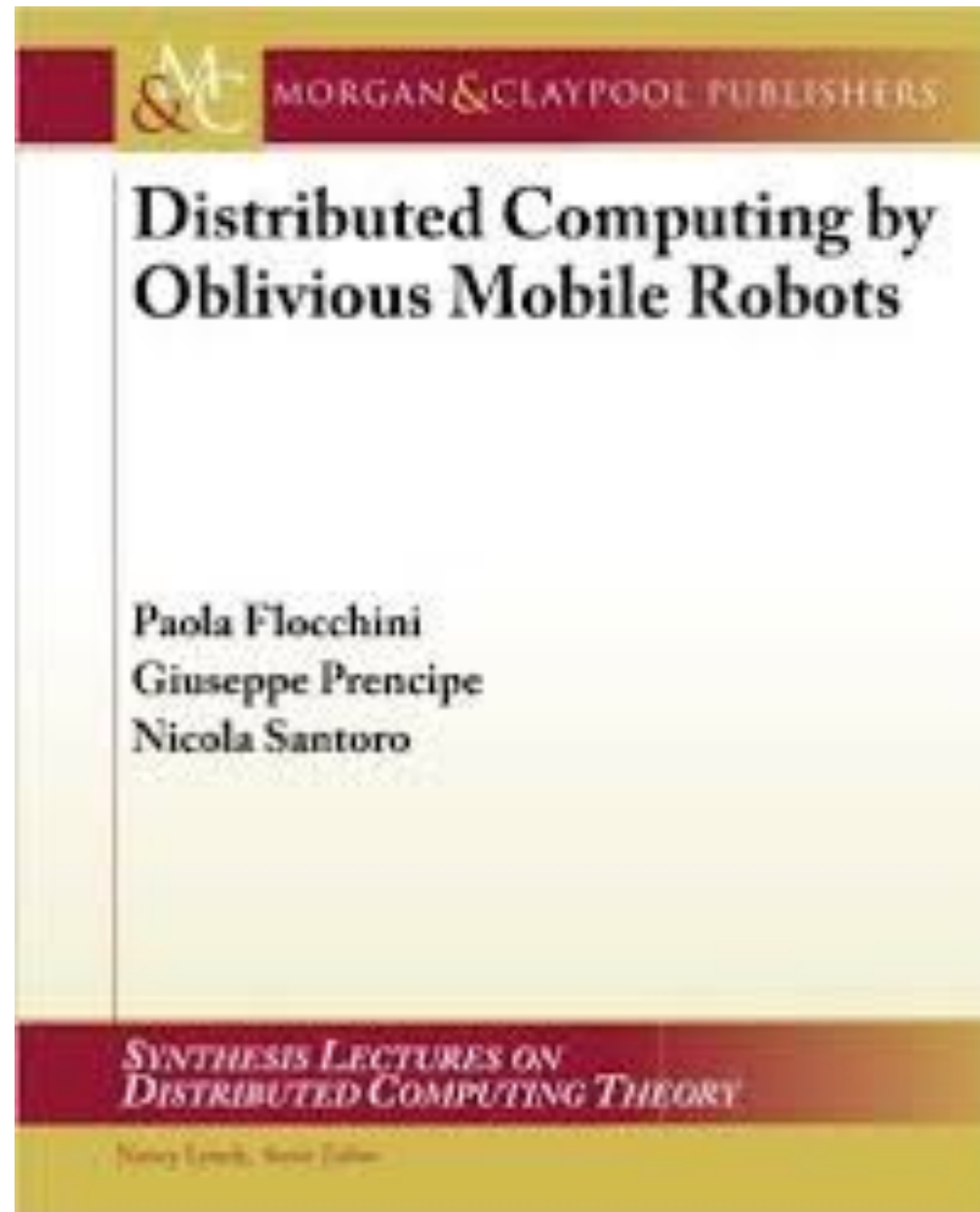
| | Convergence | 2-Gathering | n-Gathering | n-Gathering +MD | n-Gathering +MD+WF |
|-------|-------------|-------------|-------------|--------------------|-----------------------|
| FSYNC | Yes | Yes | Yes | Yes | Yes |
| SSYNC | Yes | No | No | Yes | Yes |
| ASync | Yes | No | No | Yes | ? |

Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro. *Distributed Computing by Mobile Robots: Gathering*. SIAM J. Comput. 41(4): 829-879 (2012)

Convergence & Gathering

| | Convergence | 2-Gathering | n-Gathering | n-Gathering +MD | n-Gathering +MD+WF |
|-------|-------------|-------------|-------------|--------------------|-----------------------|
| FSYNC | Yes | Yes | Yes | Yes | Yes |
| SSYNC | Yes | No | No | Yes | Yes |
| ASync | Yes | No | No | Yes | ? |

Mobile Robots



Conclusion

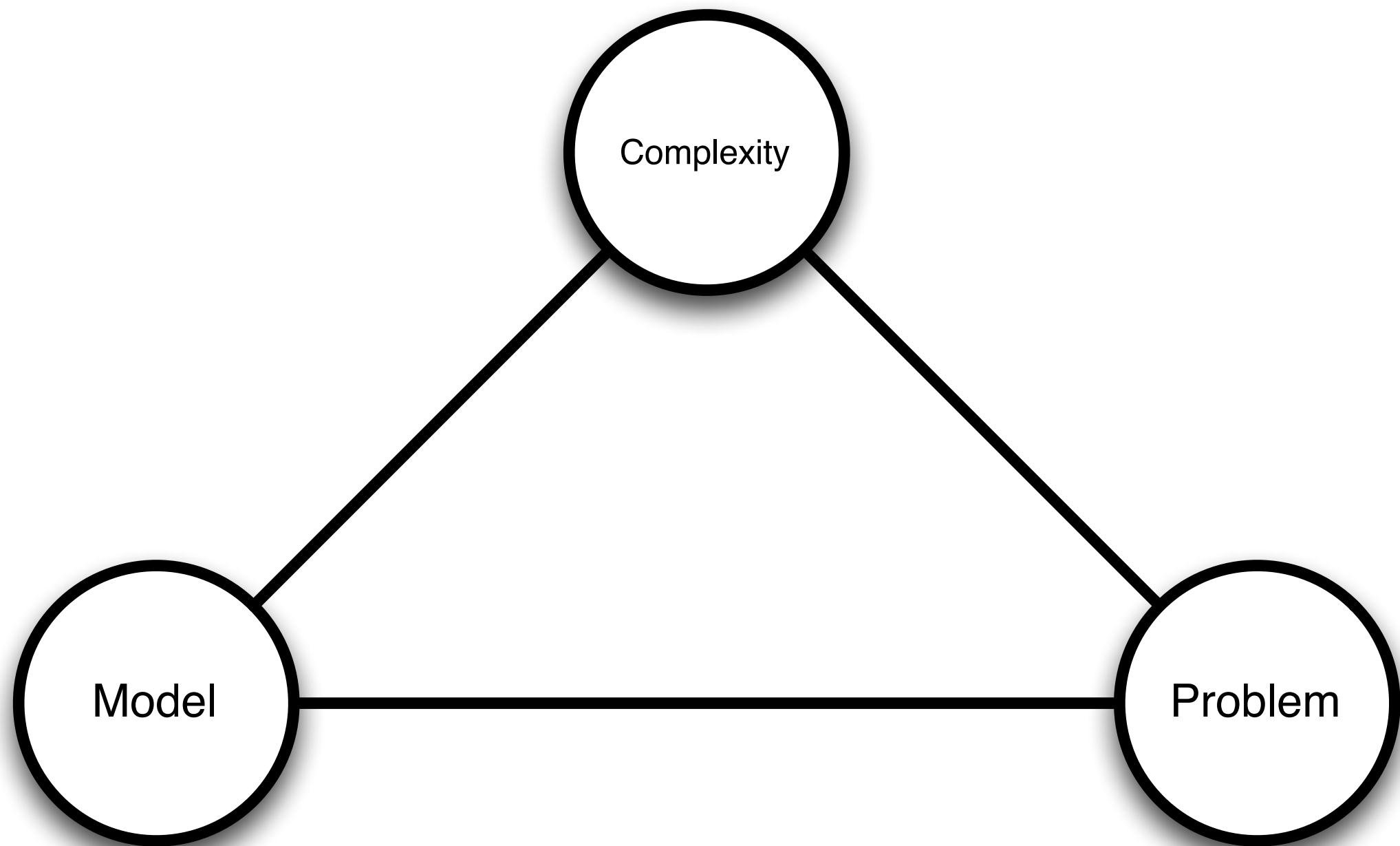
Mobility as an Adversary

- Can corrupt the distributed state of a network
- Can reduces communication capacity
- Can increase uncertainty
- Can increase protocol complexity

Mobility as a Friend

- Mobility can be the solution to the problem
- Mobility can improve efficiency
- Mobility can promote simplicity

Mobile Networks



Thank You