Lower Bounds in Distributed Computing

Petr Kuznetsov

Telecom ParisTech

(supported by Hagit Attiya, Technion)

EPIT 2017

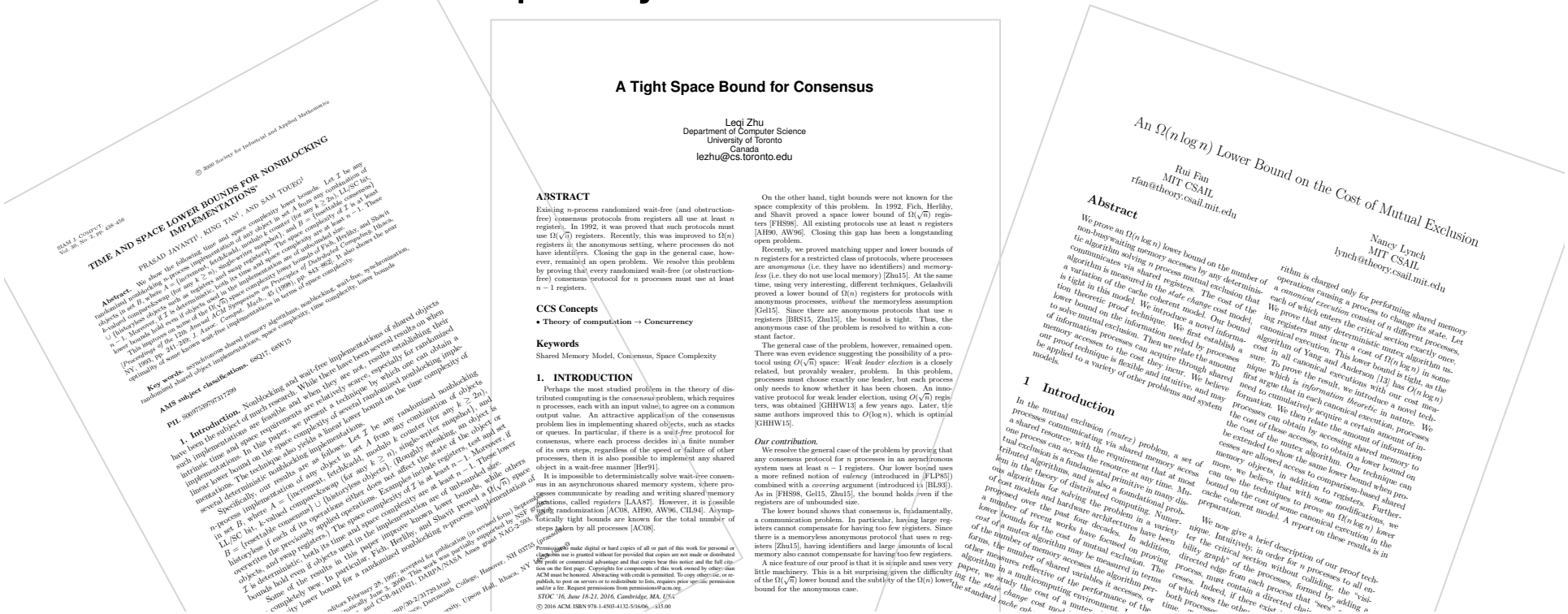
Île Porquerolles, France

Lower and upper bounds

- If something can be computed, what are the costs?
- Complexity metrics:
  - ✓time (e.g., number of steps)
  - ✓space (number of shared memory locations)
  - ✓typically asymptotic wrt n (system size)
- Lower bounds $\Omega(f(n))$ – required cost
- Upper bounds $O(f(n))$ – exhibited cost
- Tight lower bounds: $f(n)=g(n)$

This class

- Covering/valency arguments
  - ✓Time/space complexity of perturbable objects
  - ✓Space complexity of obstruction-free consensus
- Information theory: encoder/decoder
  - ✓Total work complexity of mutual exclusion



TIME AND SPACE LOWER BOUNDS FOR NONBLOCKING IMPLEMENTATIONS*

A Tight Space Bound for Consensus

Leqi Zhu
Department of Computer Science
University of Toronto
Canada
lezhu@cs.toronto.edu

An $\Omega(n \log n)$ Lower Bound on the Cost of Mutual Exclusion

Rui Fan
MIT CSAIL
rfan@theory.csail.mit.edu

Nancy Lynch
MIT CSAIL
lynch@theory.csail.mit.edu

Part I

Covering

The idea

Processes need to communicate to get their work done

Several processes may cover the same location

- An operation must write to enough distinct locations before terminating
- Otherwise, the operation is not visible

INFORMATION AND COMPUTATION 107, 171–184 (1993)

Bounds on Shared Memory for Mutual Exclusion*

JAMES E. BURNS

Georgia Institute of Technology, Atlanta, Georgia 30332

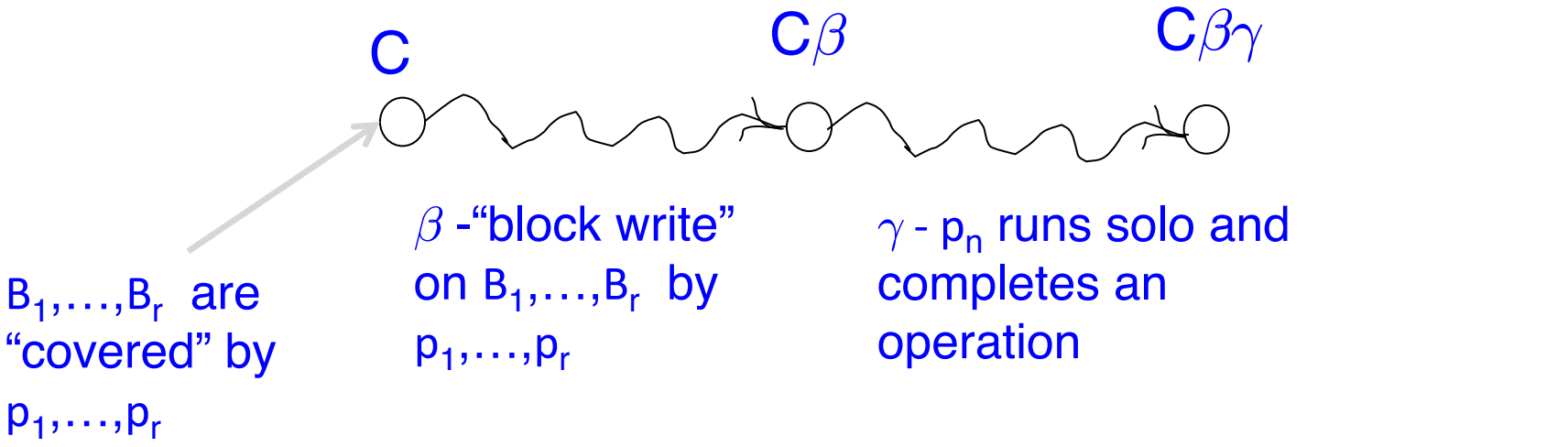
AND

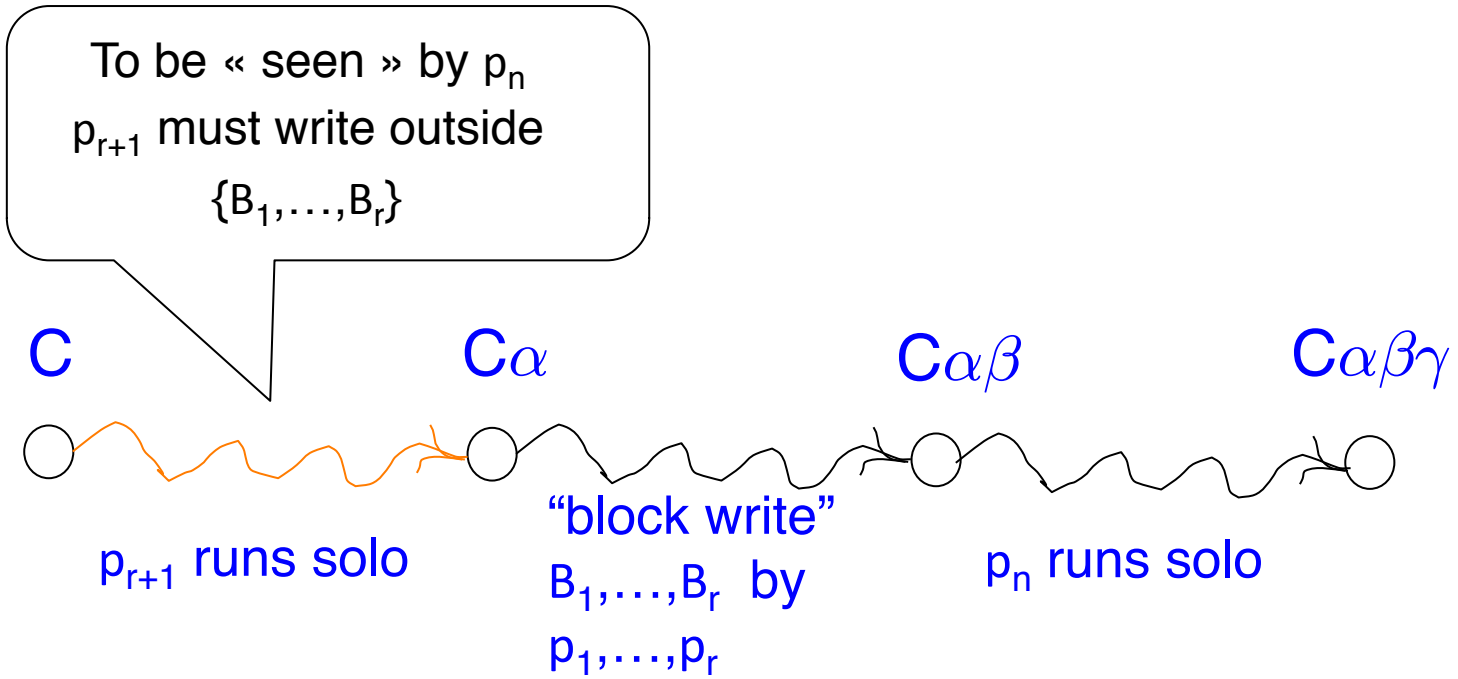
NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

The shared memory requirements of Dijkstra's mutual exclusion problem are examined. It is shown that n binary shared variables are necessary and sufficient to solve the problem of mutual exclusion with guaranteed global progress for n processes using only atomic reads and writes of shared variables for communication.

Covering and block writes





I.1: Perturbable complexity

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 438–456

TIME AND SPACE LOWER BOUNDS FOR NONBLOCKING IMPLEMENTATIONS*

PRASAD JAYANTI†, KING TAN†, AND SAM TOUEG‡

Abstract. We show the following time and space complexity lower bounds. Let $\mathcal{I}$ be any randomized nonblocking n-process implementation of any object in set A from any combination of objects in set B, where $A = \{$increment, fetch&add, modulo k counter (for any $k \geq 2n$), LL/SC bit, k-valued compare&swap (for any $k \geq n$), single-writer snapshot$\}$, and $B = \{$resettable consensus$\}$ $\cup$ $\{$historyless objects such as registers and swap registers$\}$. The space complexity of $\mathcal{I}$ is at least $n-1$. Moreover, if $\mathcal{I}$ is deterministic, both its time and space complexity are at least $n-1$. These lower bounds hold even if objects used in the implementation are of unbounded size.

This improves on some of the $\Omega(\sqrt{n})$ space complexity lower bounds of Fich, Herlihy, and Shavit [*Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*, Ithaca, NY, 1993, pp. 241–249; *J. Assoc. Comput. Mach.*, 45 (1998), pp. 843–862]. It also shows the near optimality of some known wait-free implementations in terms of space complexity.

Key words. asynchronous shared memory algorithms, nonblocking, wait-free, synchronization, randomized shared object implementations, space complexity, time complexity, lower bounds

AMS subject classifications. 68Q17, 68W15

PII. S0097539797317299

1. Introduction. Nonblocking and wait-free implementations of shared objects have been the subject of much research. While there have been several results on when such implementations are feasible and when they are not, results establishing their intrinsic time and space requirements are relatively scarce, especially for randomized implementations. In this paper, we present a technique by which one can obtain a linear lower bound on the space complexity of several randomized nonblocking implementations. The technique also yields a linear lower bound on the time complexity of several deterministic nonblocking implementations.

Specifically, our results are as follows. Let $\mathcal{I}$ be any randomized nonblocking n-process implementation of any object in set A from any combination of objects in set B, where $A = \{$increment, fetch&add, modulo k counter (for any $k \geq 2n$), LL/SC bit, k-valued compare&swap (for any $k \geq n$), single-writer snapshot$\}$, and $B = \{$resettable consensus$\} \cup \{$historyless objects$\}$. (Roughly speaking, an object is historyless if each of its operations either does not affect the state of the object or overwrites the previously applied operations. Examples include registers, test and set objects, and swap registers.) The space complexity of $\mathcal{I}$ is at least $n-1$. Moreover, if $\mathcal{I}$ is deterministic, both its time and space complexity are at least $n-1$. These lower bounds hold even if objects used in the implementation are of unbounded size.

Some of the results in this paper improve known lower bounds, while others are completely new. In particular, Fich, Herlihy, and Shavit proved a $\Omega(\sqrt{n})$ space

The result (simplified)

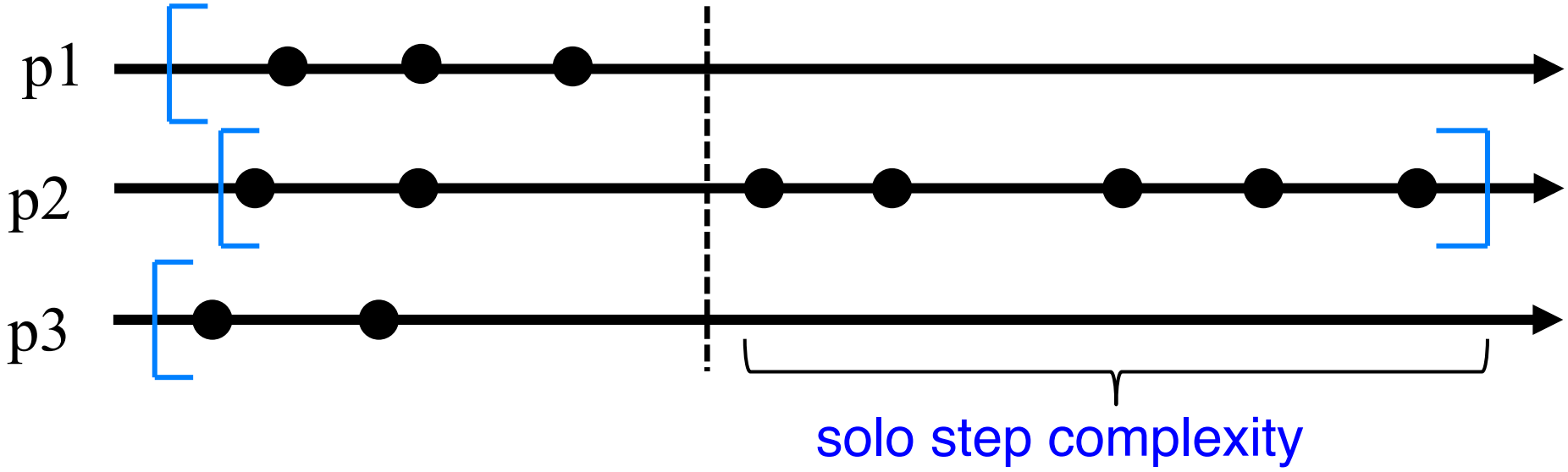
Any linearizable obstruction-freedom implementation of a counter from read-write registers has

- $\geq$ n-1 space complexity
- $\geq$ n-1 solo step complexity

Space and (solo-step) time complexity is Θ(n) [Attiya et al., JACM'09]

Obstruction-freedom

An operation is guaranteed to return if it runs in isolation (no step contention) for sufficiently long



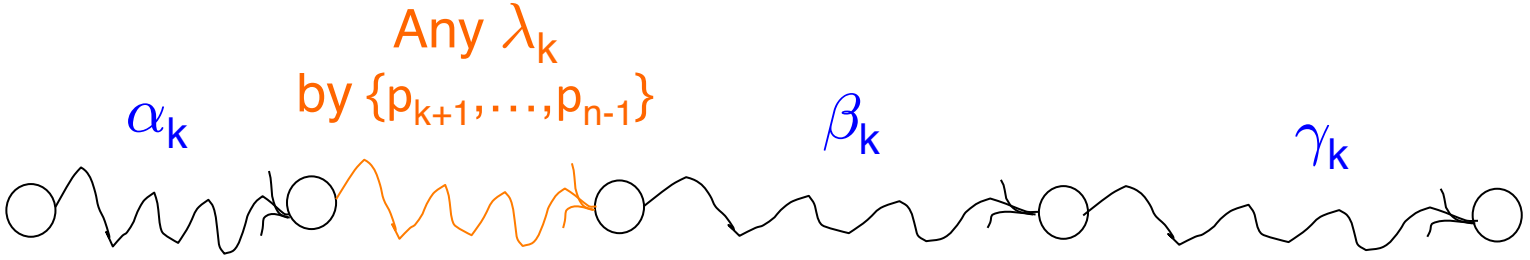
OF read-write implementations:
consensus, CAS, counters,…

Model assumptions

- Processes $\Pi=\{p_1,\ldots,p_n\}$ communicate via reading and writing to shared base objects
- Every process is assigned a deterministic (counter) algorithm
- Every process runs inc() operations, one after another
  - ✓ The state of the system is determined by a schedule --- a sequence in Π^*

Induction hypothesis

For all k=0,…,n-1, there exist schedules α_k, β_k,γ_k such that:



by $\{p_1,\ldots,p_{n-1}\}$

"block write" to $B_1,\ldots,B_k$ by $p_1,\ldots,p_k$

by p_n: only $B_1,\ldots,B_k$ are accessed (at most one inc())

p_n does not notice λ_k

k=n-1 - we are done!

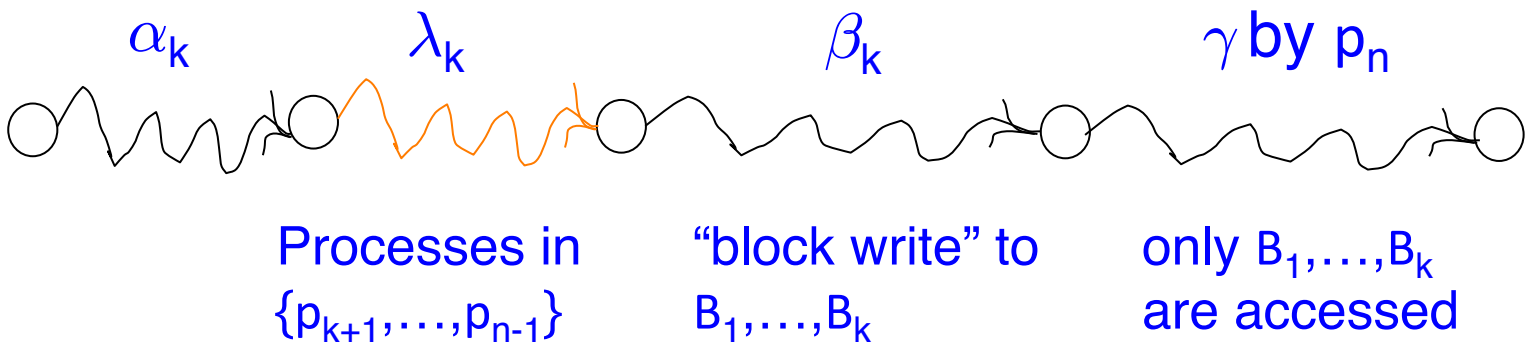
Base case

- k=0
- $\alpha_0=\beta_0=\gamma_k=\epsilon$ (empty schedules)

Suppose the hypothesis holds for

$0 \leq k<n-1$

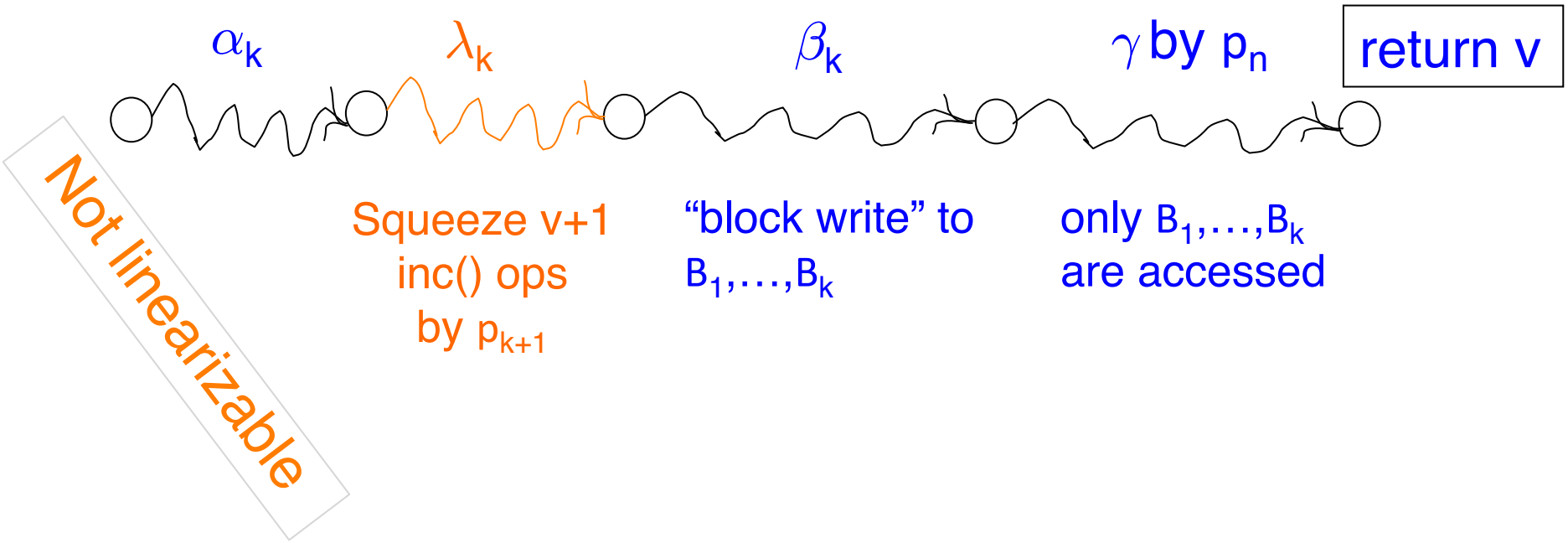
Induction step: λ_k can "perturb" p_n



Claim: p_n must access some B_{k+1} outside $\{B_1,\ldots,B_k\}$ before returning in γ

Suppose not: p_n returns v only accessing $B_1,\ldots,B_k$

p_n can be "perturbed" by λ_k



- α_{k+1} - extend α_k until p_{k+1} is about to write to some B_{k+1} outside $\{B_1,\ldots,B_k\}$
- β_k=$p_1,\ldots,p_{k+1}$
- γ_{k+1} - extend γ_k until p_n accesses B_{k+1}

Perturbable objects

There exists an assignment of operations such that for every schedule $\alpha\beta\gamma$ such that

- α and β do not contain p_n
- β is by a proper subset of $\{p_1 \ldots p_{n-1}\}$
- $\gamma \in p_n^*$ and p_n runs exactly one operation in $\alpha\beta\gamma$
- For some $p_l \in \{p_1,\ldots,p_{n-1}\}$-PSET($\beta$), $\exists\ \lambda \in p_l^*$: p_n does not complete its operation or returns a different response in $\alpha\lambda\beta\alpha$ and $\alpha\beta\alpha$

$\Omega(n)$ time/space hold for *randomized* implementations of *perturbable* objects (CAS, counters, atomic snapshots) from historyless primitives (read-write, swap, …)

I.2: Space complexity of consensus: valence and covering

A Tight Space Bound for Consensus

Leqi Zhu
Department of Computer Science
University of Toronto
Canada
lezhu@cs.toronto.edu

ABSTRACT

Existing n-process randomized wait-free (and obstruction-free) consensus protocols from registers all use at least n registers. In 1992, it was proved that such protocols must use $\Omega(\sqrt{n})$ registers. Recently, this was improved to $\Omega(n)$ registers in the anonymous setting, where processes do not have identifiers. Closing the gap in the general case, however, remained an open problem. We resolve this problem by proving that every randomized wait-free (or obstruction-free) consensus protocol for n processes must use at least $n-1$ registers.

CCS Concepts

• **Theory of computation** → **Concurrency**

Keywords

Shared Memory Model, Consensus, Space Complexity

1. INTRODUCTION

Perhaps the most studied problem in the theory of distributed computing is the *consensus* problem, which requires n processes, each with an input value, to agree on a common output value. An attractive application of the consensus problem lies in implementing shared objects, such as stacks or queues. In particular, if there is a *wait-free* protocol for consensus, where each process decides in a finite number of its own steps, regardless of the speed or failure of other processes, then it is also possible to implement any shared object in a wait-free manner [Her91].

It is impossible to deterministcally solve wait-free consensus in an asynchronous shared memory system, where processes communicate by reading and writing shared memory locations, called *registers* [LAA87]. However, it is possible using randomization [AC08, AH90, AW96, CIL94]. Asymptotically tight bounds are known for the total number of steps taken by all processes [AC08].

On the other hand, tight bounds were not known for the space complexity of this problem. In 1992, Fich, Herlihy, and Shavit proved a space lower bound of $\Omega(\sqrt{n})$ registers [FHS98]. All existing protocols use at least n registers [AH90, AW96]. Closing this gap has been a longstanding open problem.

Recently, we proved matching upper and lower bounds of n registers for a restricted class of protocols, where processes are *anonymous* (i.e. they have no identifiers) and *memoryless* (i.e. they do not use local memory) [Zhu15]. At the same time, using very interesting, different techniques, Gelashvili proved a lower bound of $\Omega(n)$ registers for protocols with anonymous processes, *without* the memoryless assumption [Gel15]. Since there are anonymous protocols that use n registers [BRS15, Zhu15], the bound is tight. Thus, the anonymous case of the problem is resolved to within a constant factor.

The general case of the problem, however, remained open. There was even evidence suggesting the possibility of a protocol using $O(\sqrt{n})$ space: *Weak leader election* is a closely related, but provably weaker, problem. In this problem, processes must choose exactly one leader, but each process only needs to know whether it has been chosen. An innovative protocol for weak leader election, using $O(\sqrt{n})$ registers, was obtained [GHHW13] a few years ago. Later, the same authors improved this to $O(\log n)$, which is optimal [GHHW15].

Our contribution.

We resolve the general case of the problem by proving that any consensus protocol for n processes in an asynchronous system uses at least $n-1$ registers. Our lower bound uses a more refined notion of *valency* (introduced in [FLP85]) combined with a *covering* argument (introduced in [BL93]). As in [FHS98, Gel15, Zhu15], the bound holds even if the registers are of unbounded size.

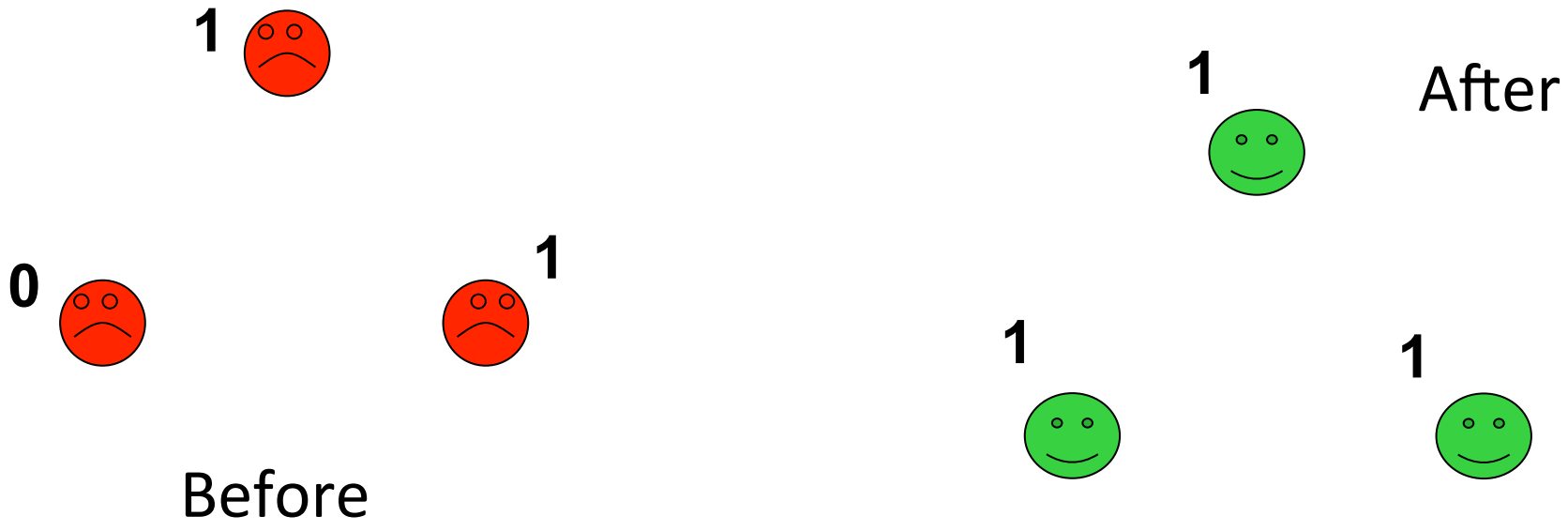
The lower bound shows that consensus is, fundamentally, a communication problem. In particular, having large registers cannot compensate for having too few registers. Since

The result

Space complexity of any obstruction-free binary consensus implementation is n-1

Θ(n) space complexity [AGHK09,BRS15]

(applies to randomized consensus too)



Refined model assumptions

- Processes $\Pi=\{p_1,\ldots,p_n\}$ communicate via reading and writing to shared base objects
- Every process is assigned a deterministic algorithm
- Configuration
  - ✓ Local state for each process
  - ✓ State of each register
- Initial configuration
  - ✓ Input assignment
  - ✓ Registers in initial states
- Configuration C and schedule $\alpha \in \Pi^*$ define a configuration Cα

Valence of a configuration

C is *v-valent* (for v in {0,1}) if v is decided in every extension of C

C is *bivalent* if both 0 and 1 can be decided in extensions of C

- Every configuration is 0-valent, or 1-valent, or bivalent.
- If some process decides v in C, then C is v-valent
- No process can decide in a bivalent run

Refined valence of a configuration

Let C be a configuration and P$\subseteq \Pi$

P is *v-valent from C* (for v in {0,1}) if v is decided in every P-only extension Cα

P is *bivalent from C* if both 0 and 1 can be decided in P-only extensions of C

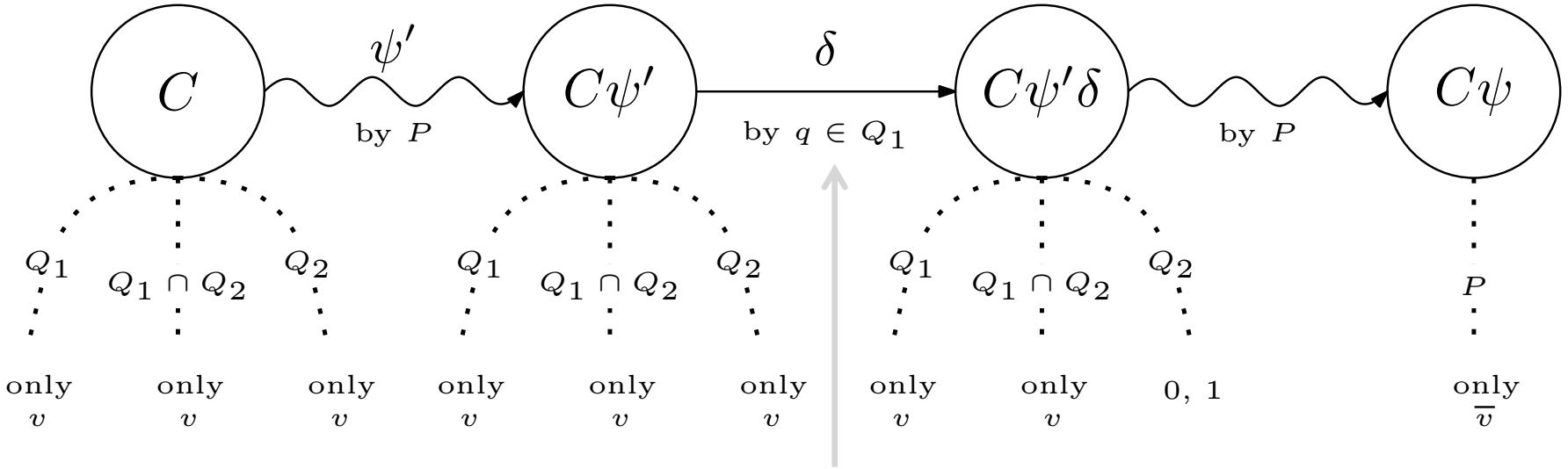
For all C and non-empty P:

- P is 0-valent, or 1-valent, or bivalent from C
- If P is v-valent in C, then any non-empty P'$\subseteq$ P is v-valent from C

Valence: more properties

Lemma 1 If P ($|P| \geq 3$) is bivalent from C, then there exists a P-only schedule φ and $z \in P$ such that P-{z} is bivalent from $C\varphi$

Proof: Take $z_1, z_2 \in P$, let $Q_1 = P - \{z_1\}$ and $Q_2 = P - \{z_2\}$, $Q_1 \cap Q_2 \neq \emptyset$



Q_1 and Q_2 are v-valent

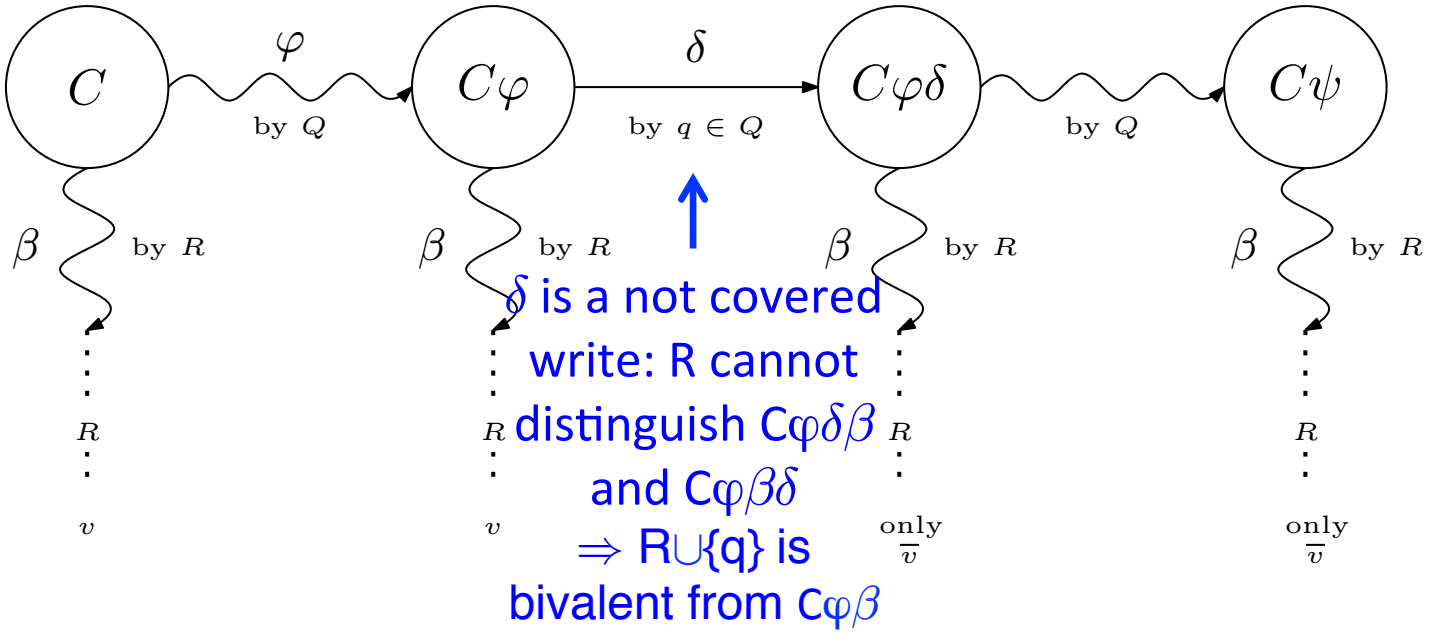
"Critical" step of q, let $q \in Q_1$: Q_2 is bivalent

Q_1 and Q_2 are (1-v)-valent

Valence and read-write: more properties

Lemma 2 Let P be bivalent from C, and β be a block write (in C) by R$\subseteq$ P. Then every deciding schedule from C by z not in P should contain a write not covered by R in C.

Lemma 3 Let P-R be bivalent from C, and β be a block write by non-empty R$\subseteq$ P. If Q=P-R is bivalent from C, then there exists Q-only schedule φ and q $\in$ Q such that R$\cup${q} is bivalent from C$\varphi\beta$.



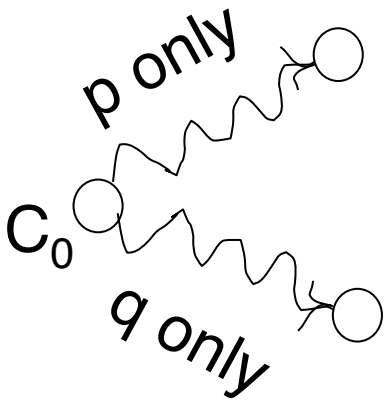
The goal

If P, |P|$\geq$ 2, is bivalent from C, then there exists P-only α and Q$\subseteq$ P, |Q|=2, such that

- Q is bivalent from Cα
- Every process in P-Q covers a distinct register in Cα

Proof by induction on |P|:

- Base case |P|=2: any initial configuration from which some P={p,q} is bivalent



For |P|=n we are done: n-2 distinct registers are covered

Induction step

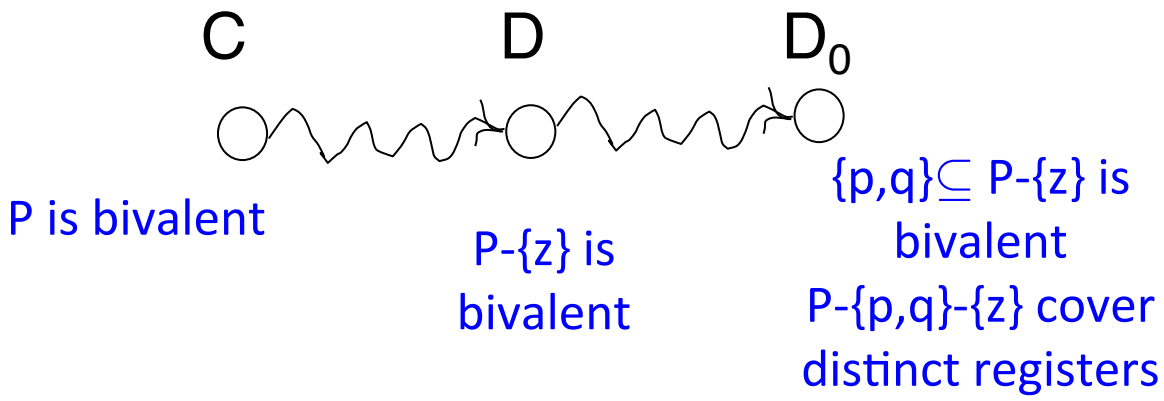
P (of size $\geq$3) is bivalent from C, the claim hold for subsets of size |P|-1

By Lemma 1, for some z$\in$ P and P-only γ, P-{z} is bivalent from D=Cγ

z is left for future (larger) cover

By induction hypothesis, there exist D_0, extension of D, and a pair $Q_0 \subseteq$ P-{z} such that

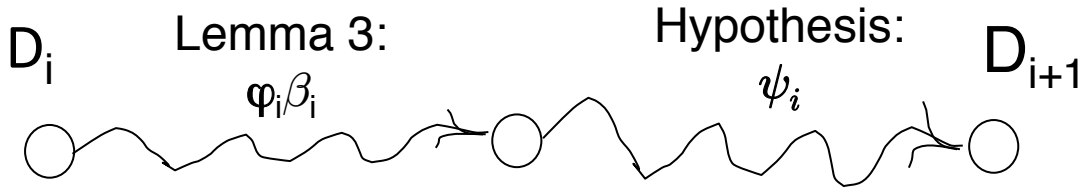
- Q_0 is bivalent from D_0
- P-{z}-Q_0 cover distinct registers in D_0



Induction step (ctd.)

Applying Lemma 3 and induction hypothesis repeatedly we get an infinite (P-{z})-only extension $D_0 \rightarrow D_1 \rightarrow D_3 \rightarrow \ldots$

- D_i: some pair $Q \subseteq$ P-{z} is bivalent and the rest cover IPI-2 registers



$Q \subseteq$ P is bivalent
P-Q-{z} cover distinct registers

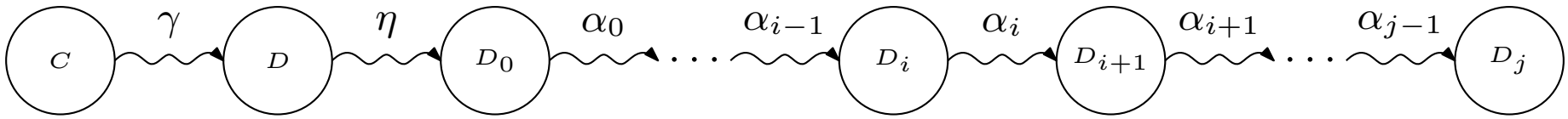
$(P\text{-}Q\text{-}z) \cup \{q\}$ is bivalent

$\{p',q'\} \subseteq$ P-{z} is bivalent
P-{p',q'}-{z} cover distinct registers

Induction step (ctd.)

But there are finitely many registers!

Some D_i and D_j cover the same set of IPI-3!



Q_i is bivalent
P-{z}-Q_i cover

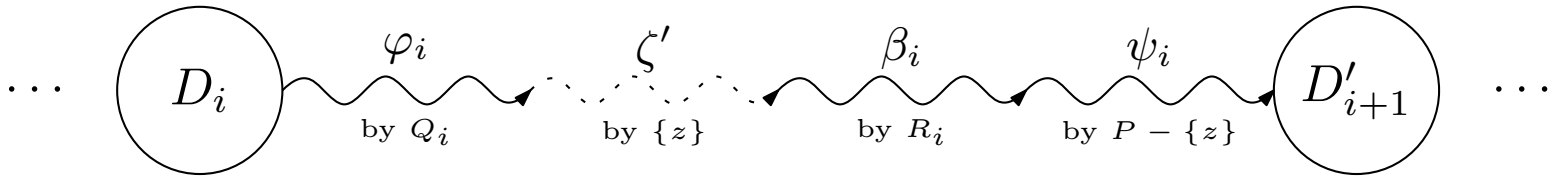
Q_{i+1} is bivalent
P-{z}-Q_{i+1} cover
the same set

Induction step (ctd.)

Run z from $D_i\varphi_i$ until it decides: $D_i\varphi_i\zeta$

z must write to a not covered register (Lemma 2), stop just before: $D_i\varphi_i\zeta'$

No (P-{z}) extension of $D_i\varphi_i\,\zeta'\beta_i$ can see the difference from $D_i\varphi_i\beta_i$



All writes by z are hidden by the block write

indistinguishable from D_{i+1} for P-{z}

Continue by P-{z} until D'_j and get a cover by |P|-2 processes!

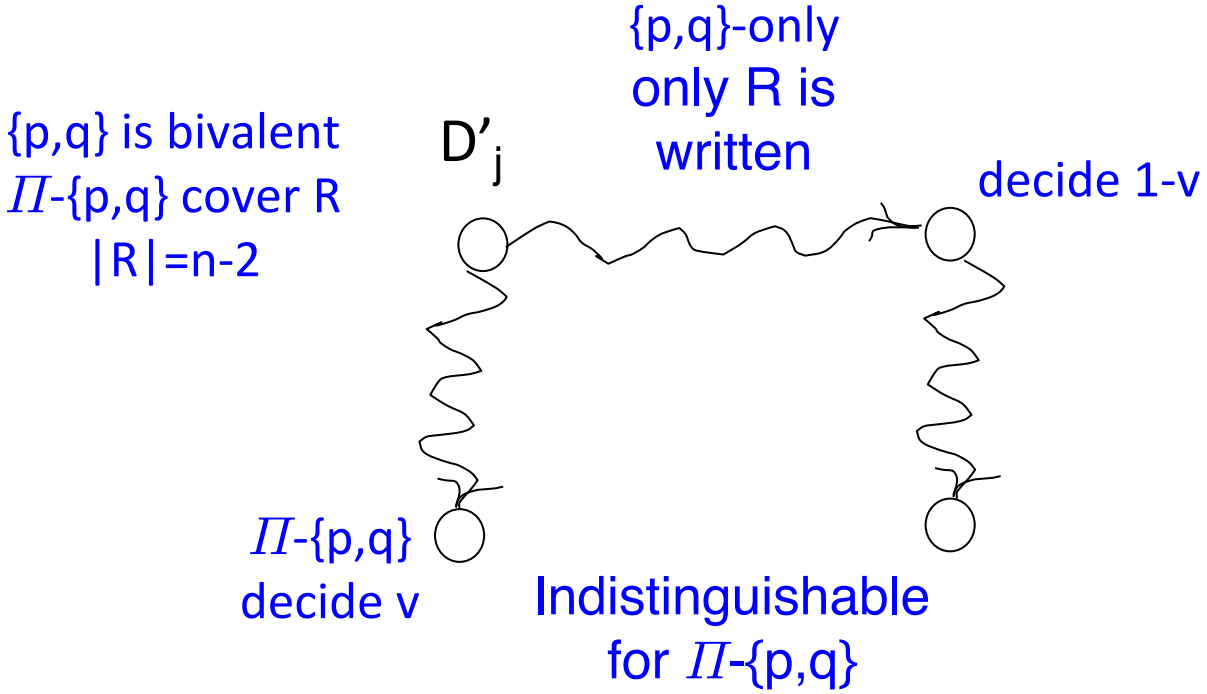
For k=n, n-2 distinct registers are covered!

Finally!

Take k=n

Let {p,q} be bivalent in D_j'

{p,q}-only extension of D_j' must write to a non-covered register



n-1 registers covered!

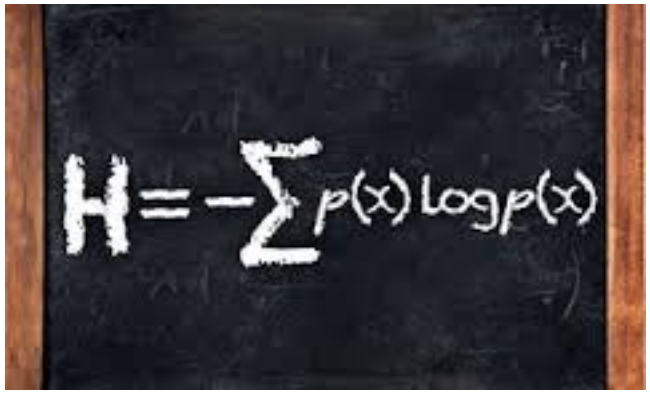
Wrapping up: covering/valence

- Perturbable objects (CAS, counters, AS,…)
  - ✓“pure covering” assuming long-lived (perturbable) operations: time&space
- One-shot (non-perturbable) consensus
  - ✓Covering&valence: space
- The proofs are about constructing a (worst-case) run

And now for something completely different

Part II

Information theory



Ω(n log n) total work in mutual exclusion: The encoder/decoder argument

An $\Omega(n \log n)$ Lower Bound on the Cost of Mutual Exclusion

Rui Fan
MIT CSAIL
rfan@theory.csail.mit.edu

Nancy Lynch
MIT CSAIL
lynch@theory.csail.mit.edu

Abstract

We prove an $\Omega(n \log n)$ lower bound on the number of non-busywaiting memory accesses by any deterministic algorithm solving n process mutual exclusion that communicates via shared registers. The cost of the algorithm is measured in the *state change* cost model, a variation of the cache coherent model. Our bound is tight in this model. We introduce a novel information theoretic proof technique. We first establish a lower bound on the information needed by processes to solve mutual exclusion. Then we relate the amount of information processes can acquire through shared memory accesses to the cost they incur. We believe our proof technique is flexible and intuitive, and may be applied to a variety of other problems and system models.

1 Introduction

rithm is charged only for performing shared memory operations causing a process to change its state. Let a *canonical execution* consist of n different processes, each of which enters the critical section exactly once. We prove that any deterministic mutex algorithm using registers must incur a cost of $\Omega(n \log n)$ in some canonical execution. This lower bound is tight, as the algorithm of Yang and Anderson [13] has $O(n \log n)$ cost in all canonical executions with our cost measure. To prove the result, we introduce a novel technique which is *information theoretic* in nature. We first argue that in each canonical execution, processes need to cumulatively acquire a certain amount of information. We then relate the amount of information processes can obtain by accessing shared memory to the cost of those accesses, to obtain a lower bound on the cost of the mutex algorithm. Our technique can be extended to show the same lower bound when processes are allowed access to comparison-based shared memory objects, in addition to registers. Further-

The result

Total work of any n-process mutual exclusion algorithm is

$$\Omega(n \log n)$$

The number of (non busy-waiting) memory accesses performed by $p_1,..,p_n$ to enter CS

- ✓ $\approx$ remote memory references (RMRs) in CC and DSM memory models [GW12,…]

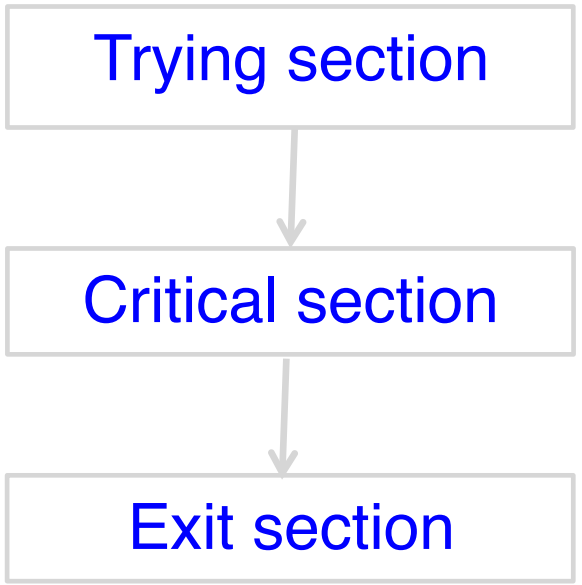
- Tight [Yang&Anderson, 95]:

$$\Theta(n \log n)$$

- Holds even for stronger primitives (CAS, …)

Mutual exclusion

- No two processes are in their critical sections (CS) at the same time
- **Deadlock-freedom**: at least one process in its trying section (TS) eventually enters its CS
  - ✓assuming no process fails or stays in its CS forever



Peterson's mutual exclusion

```
// initialization
level[0..n-1] = {-1};      // current level of processes 0…n-1
waiting[0..n-2] = {-1}; // the waiting process in each level
                           // 0…n-2

// code for process i that wishes to enter CS
for (m = 0; m < n-1; m++) {
    level[i] = m;
    waiting[m] = i;
    while(waiting[m] == i &&(exists k ≠ i: level[k] ≥ m)) {
        // busy wait
    }
}
// critical section
level[i] = -1; // exit section
```

Total work $O(n^3)$

The idea: acquiring information incurs costs

- A canonical execution: every process enters CS exactly once
- Processes (cumulatively) must learn about the order of CSs
- Getting O(C) bits of information $\equiv$ performing O(C) work

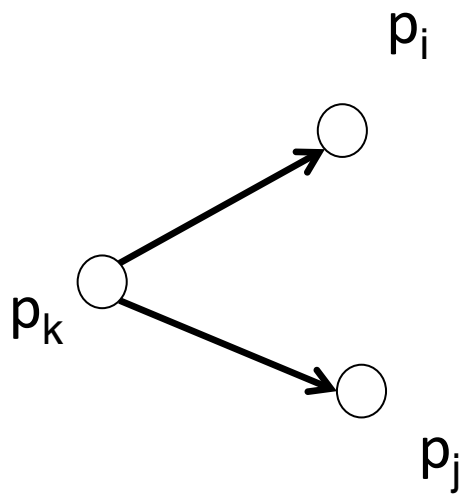
n! distinct orders $\equiv \Omega$(n log n) work

Visibility graph of a canonical run

p_i "sees" $p_j \equiv$ the CS of p_i is causally preceded by CS of p_j

- p_j left CS before p_i started its CS

Claim: in a canonical run, for every p_i and p_j, at least one sees the other



Let p_i and p_j "miss" each other

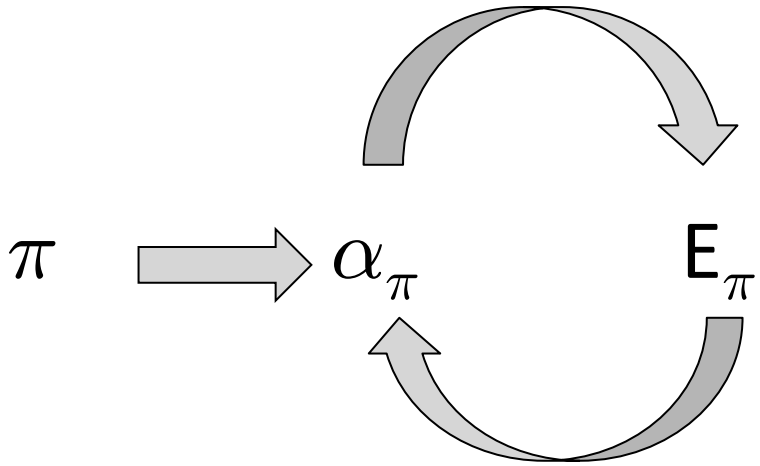
can be driven to their CSs simultaneously

Proof outline

1. **Construction step**: for each permutation $\pi=\pi_1,\ldots,\pi_n$, build a (canonical) run α_π with order π of CS accesses

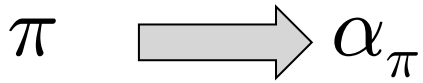
2. **Encoding step**: for each α_π, produce a binary string E_π of length O(cost(α_π))

3. **Decoding step:** reproduce α_π given E_π



- $\{E_\pi\}$ is a code of $\{\alpha_\pi\}$
- Some codeword E_π has length Ω(n log n)
- The cost of π is Ω(n log n)

1. Construction



α_π is constructed iteratively:

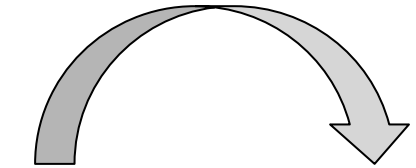
- α_1: p_{π_1} enters CS and exits
- From α_i to α_{i+1}: add a complete run of $p_{\pi i+1}$ so that no process, so that $p_{\pi 1},\ldots,p_{\pi i}$ do not see it
  - ✓The trickiest part: maintain a partial order on metasteps $(M_i,\preceq_i)$
- $(M_n,\preceq_n) \Rightarrow \alpha_\pi$

Metastep (on the same register):

- Set of reads
- Set of writes
- Winning write

Signature: counts of reads and writes

2. Encoding



α_π $\quad$ E_π

$(M_n, \preceq_n) \rightarrow E_\pi$

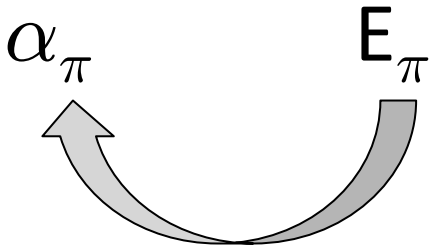
Only metasteps are encoded

The cost of a metastep with k processes is O(k)

E_π uses O(k) bits per metastep with k processes

p_1	...	p_n
1		1
		1
0		0
..	..	..

3. Decoding



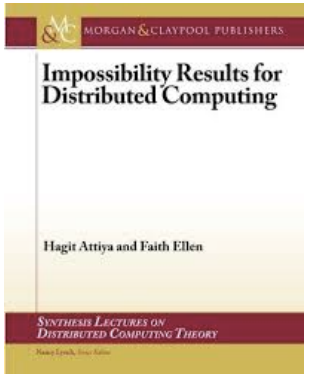
Iteratively compute the minimal “unexecuted” step:

- Given the algorithm and the current state of each process
- Compute the metastep’s composition

$$\Rightarrow (M_n, \preceq_n)$$

Wrapping up

- Not everything (computable sequentially) can be computed in a distributed way
  - ✓Computable problems are subject to complexity bounds
- There are many lower bounds but fewer techniques
  - ✓Covering/valence/potential functions
  - ✓Information theory
  - ✓Combinatorial arguments



(Some) open questions

- From n-1 to n for consensus/perturbable objects
- (Solo) time complexity for OF consensus
- Time/space for k-set consensus
  - ✓Only anonymous case is explored
  - ✓ Ω(\sqrt{nm/k}) for m-OF k-set consensus
  - ✓ Θ(n+m-k) for repeated m-OF k-set consensus
- Other system parameters
  - ✓Progress guarantees
  - ✓Contention
- Other metrics/beyond worst-case
  - ✓Throughput? (amortized complexity? scheduling?)
- Computability bounds
  - ✓Tight impossibility results: what is the weakest model for a given pb?

Merci beaucoup!

Questions?

Induction hypothesis

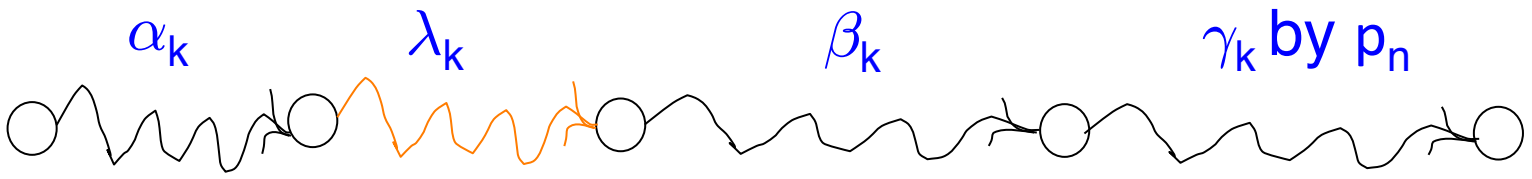
For all k=0,…,n-1, there exist schedules α_k, β_k,γ_k such that:

- α_k and β_k do not contain p_n
- For k=0, $\beta_k=\epsilon$ (empty), for k>0, $\beta_k=p_1,\ldots,p_k$
- $\gamma_k \in p_n^*$
- The set of objects accessed by p_n in $\alpha_k\beta_k\gamma_k$ is $\{B_1,\ldots,B_k\}$
- p_n performs at most one inc() in $\alpha_k\beta_k\gamma_k$
- Let λ_k be any schedule of $\{p_{k+1},\ldots,p_{n-1}\}$. Then p_n cannot distinguish $\alpha_k\lambda_k\beta_k\alpha_k$ and $\alpha_k\beta_k\alpha_k$

k=n-1 - we are done!

Induction step

- α_k and β_k do not contain p_n
- β_k is a block write by $p_1,\ldots,p_k$ to registers $B_1,\ldots,B_k$
- $\gamma_k \in p_n^*$ and p_n performs at most one operation in $\alpha_k\beta_k\gamma_k$
- Let λ_k be any schedule of $\{p_{k+1},\ldots,p_{n-1}\}$. Then p_n cannot distinguish $\alpha_k\lambda_k\beta_k\alpha_k$ and $\alpha_k\beta_k\alpha_k$



Processes in $\{p_{k+1},\ldots,p_{n-1}\}$

"block write" to $B_1,\ldots,B_k$

only $B_1,\ldots,B_k$ are accessed

Proof of Lemma 1

- Take $z_1, z_2 \in P$
- Let $Q_1 = P - \{z_1\}$ and $Q_2 = P - \{z_2\}$, $Q_1 \cap Q_2 \neq \emptyset$
- Let $Q_1 \cap Q_2$ decides $v \in \{0,1\}$ from C
  - ✓ If some Q_i decides 1-v from C - we are done

C is bivalent: 1-v is decided in Cψ

There must be a critical step in ψ: from Cψ' ("only v is decided by Q_1 and Q_2") to Cψ'δ ("some Q_i decides 1-v")

Let z (taking this step δ) be in Q_1:

- Q_1 is v-valent from Cψ'z
- Q_2 is bivalent from Cψ'z