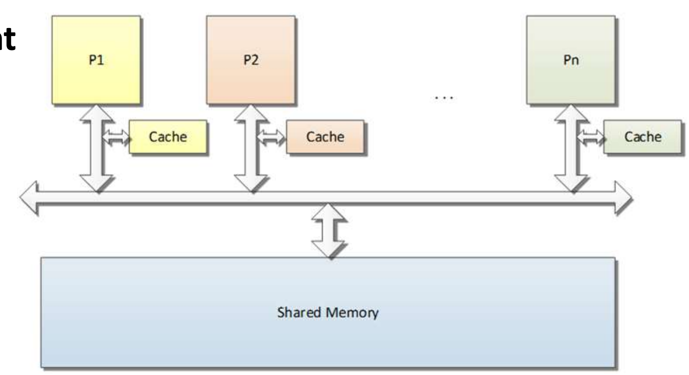
Lower Bounds on the Amortized Time Complexity of Shared Objects

Hagit Attiya, Technion
Arie Fouren, Ono Academic College

1

The Model

- **Asynchronous cache-coherent (CC)** shared-memory model
- **Remote Memory Reference (RMR)** complexity:
  - count only the events that generate process-memory interconnection traffic
  - ignore busy-wait loops on unchanged local variables



2

Step Complexity Measures

- **Amortized step complexity:**
  average number of steps performed by invoked operations (worst case over all possible executions)
  - measures performance of the whole system, not individual operations
  - suitable for lock-free implementations, where operations may never terminate
- **Point contention $\dot{c}$:** # simultaneously active processes

3

Upper Bounds on Lock-free Implementations

[Ruppert 2017]

The amortized step complexity of lock-free stacks, queues, linked lists, doubly-linked lists, binary trees and union-find have an additive factor of point contention $\dot{c}$

➢Is this additive factor of point contention $\dot{c}$ inherent ?

This work:
The additive factor of $\dot{c}$ is inherent for stacks, queues, heaps, linked lists and search trees

4

Our Lower Bounds I

Any implementation of stacks, queues and heaps, using reads, writes and conditional operations (e.g., CAS), has $\Omega(\dot{c})$ amortized RMR complexity, provided $\dot{c} \in O\left(\sqrt{\log\log n}\right)$

- reduction to the $\Omega(\dot{c})$ lower bound on MUTEX **[Kim, Anderson 2012]**, extended to amortized step complexity
- relies only on the abstract definition of shared objects, not on implementation details

5

Our Lower Bounds II

Any implementation of data structures based on a connected graph of nodes (graph-based-set), using 1-revealing primitives (reads, writes, CAS, Test&Set, LL/SC, ...), has $\Omega(\dot{c})$ amortized step complexity, provided $\dot{c} \in O(\sqrt{\log\log n})$

E.g., linked lists, skip lists, binary search trees, B-trees

☑does not require $delete()$ operation

☒implementation-dependent

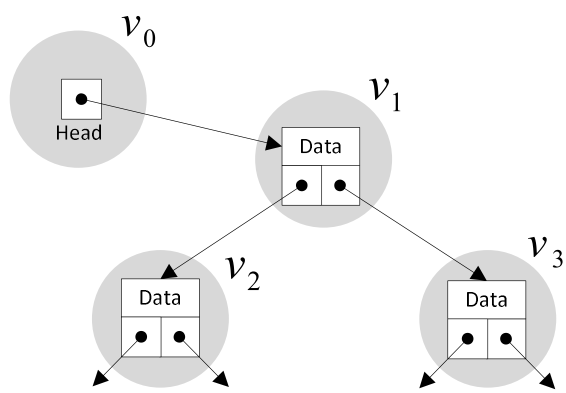
☒bounds only step complexity, not RMRs

6

Graph-based-set

- A shared variable contains data + d pointers to other variables
- Memory graph $G(\alpha) = (V, E)$
- Special node $head$
- One operation $add(e)$

Representation invariant:
$e \in S$ after α if and only if there is a directed path from $S.head$ to $v = e$ in $G(\alpha)$



7

Proof Overview

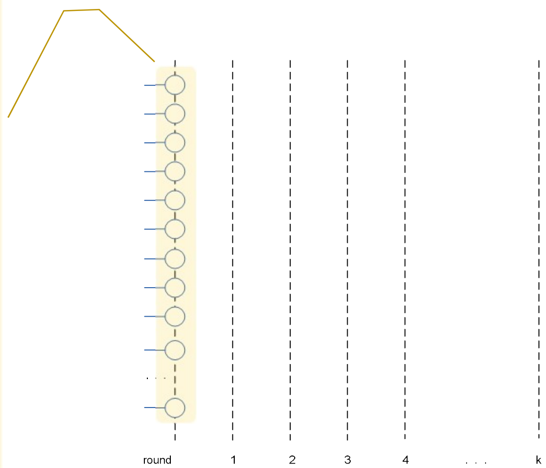
- Execution of $3k$ processes
- Each process p_i invokes $add(id_i)$ once
- $3k$ processes collectively perform $\Omega(k^2)$ steps

⇨ Any implementation of graph-based-set has at least $\Omega(k) = \Omega(\dot{c})$ amortized step complexity

8

Proof Overview

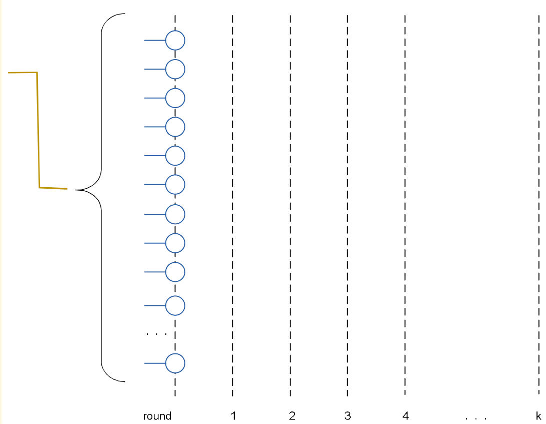
In each round, all processes perform the same type of operation



Proof Overview

Initially, all processes are **invisible** to each other

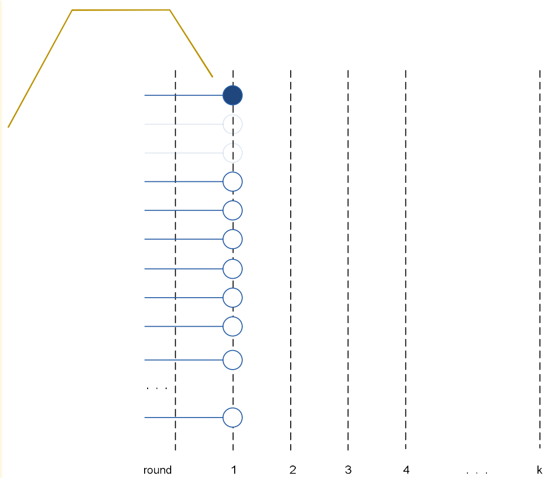
Invisible processes can be removed without affecting other processes



Proof Overview

In each round, each invisible process takes one step

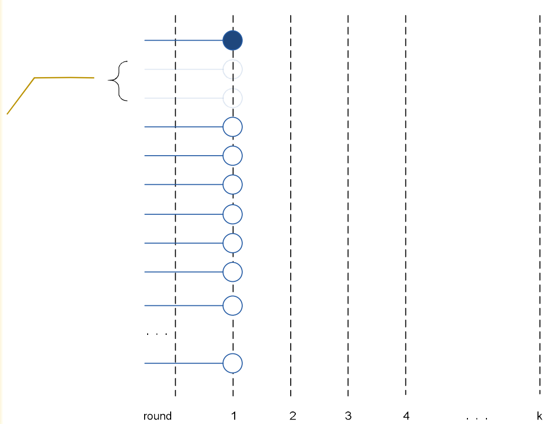
A constant number of invisible processes become **visible** and stop



11

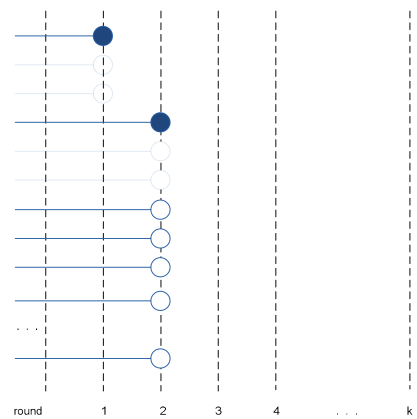
Proof Overview

Some invisible processes are retroactively deleted from the execution to keep other processes invisible



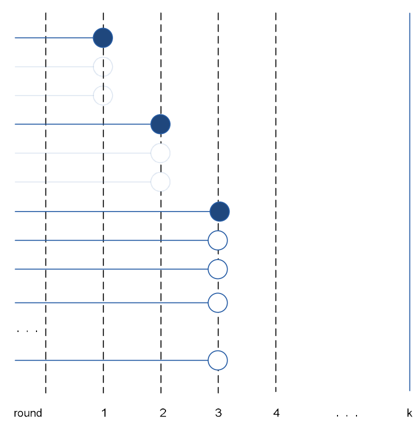
12

Proof Overview



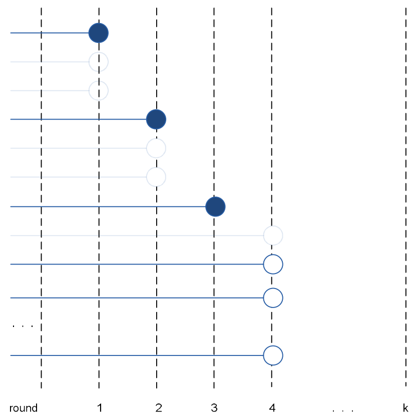
13

Proof Overview



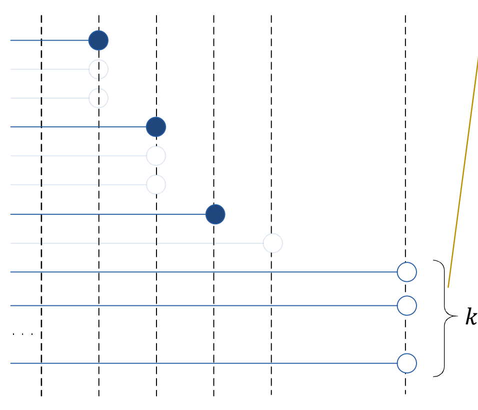
14

Proof Overview



15

Proof Overview



After k rounds, k processes are invisible

Collectively perform $\Omega(k^2)$ steps $\Rightarrow$ $\Omega(k) = \Omega(\dot{c})$ amortized step complexity

16

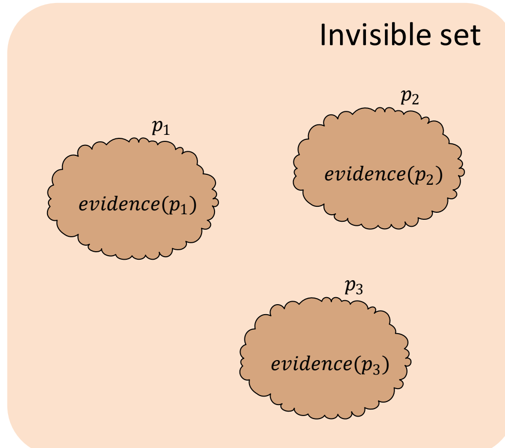
How to keep processes invisible ?

Evidence variables of process p_i

- May contain "traces" of p_i's events
- May change if p_i's events are deleted from the execution

Invisible set of processes P

- Disjoint sets of evidence variables
- Erasing a subset of P is undetected by other invisible processes



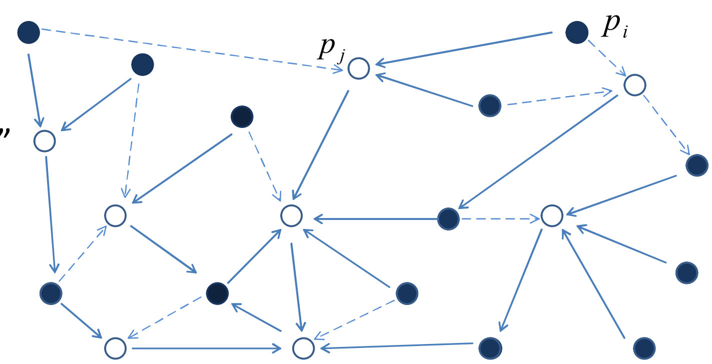
17

How to keep processes invisible ? Eliminating conflicts with previous rounds

Visibility graph

- Nodes: invisible processes
- $p_i \rightarrow p_j$ if p_i is about to "see" an evidence variable of p_j

Turán's theorem
⇨ eliminate conflicts while keeping a constant fraction of the invisible processes

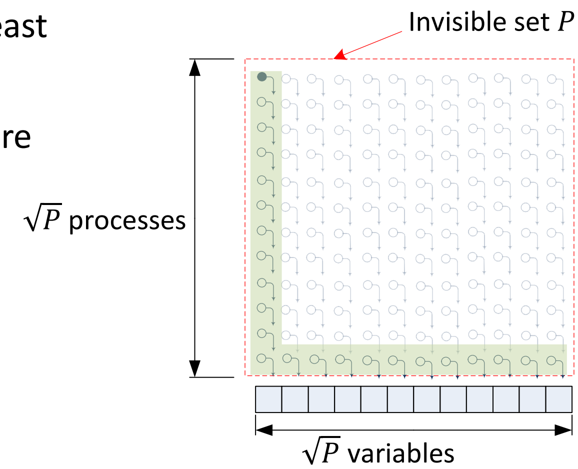


18

How to keep processes invisible? Eliminating conflicts in the same round

- Some variable is accessed by at least $\sqrt{|P|}$ different processes or
- At least $\sqrt{|P|}$ different variables are accessed

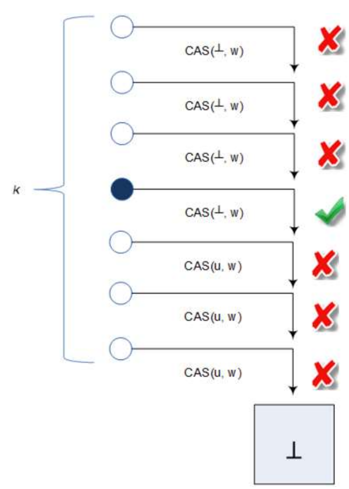
➢In both cases, at least $\sqrt{|P|}$ processes can be kept invisible, since primitives are 1-revealing



19

How to keep processes invisible? 1-Revealing Primitives

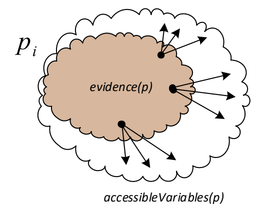
- k operations accessing the same variable can be ordered so that only one succeeds & the rest fail
- The successful process becomes visible & others remain invisible
- Reads, writes, and conditional operations (CAS, LL/SC, Test&Set, …) are 1-revealing

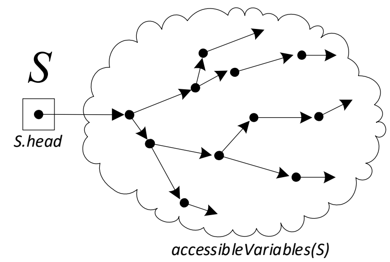


20

How to keep the graph-based-set small ?

- **Accessible variables of a process $\boldsymbol{p_i}$**: there is a directed path from an evidence variable of p_i to these variables in the memory graph G
- **Accessible variables of a graph-based-set $\boldsymbol{S}$:** there is a directed path from $S.head$ to these variables in the memory graph G

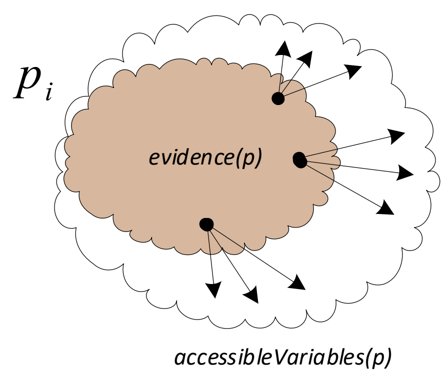




21

How to keep the graph-based-set small ? Keep evidence and accessible sets small

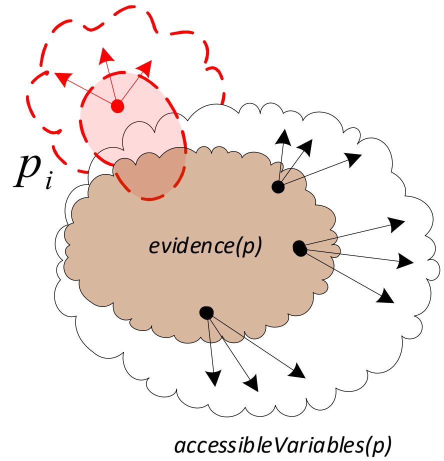
- round r :
  - $|evidence(p)| \leq r$
  - $|accessible(p)| \leq r(d+1)$



22

How to keep the graph-based-set small ? Keep each process' evidence set small

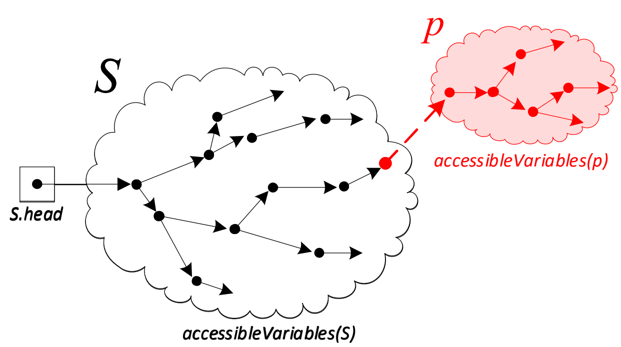
- round r :
  - $|evidence(p)| \leq r$
  - $|accessible(p)| \leq r(d+1)$
- round $r+1$:
  - p accesses at most one variable
  - $|evidence(p)|$ grows at most by 1
  - $|accessibleVariables(p)|$ grows at most by $(d+1)$



23

How to keep the graph-based-set small ?

- round r:
  - S contains r processes
  - $|accessibleVariables(S)| \leq \frac{r(r+1)(d+1)}{2}$
- round $r+1$:
  - at most one process succeeds to add its id to S
  - the number of processes in S grows at most by 1
  - $|accessibleVariables(S)|$ grows at most by $r(d+1)$

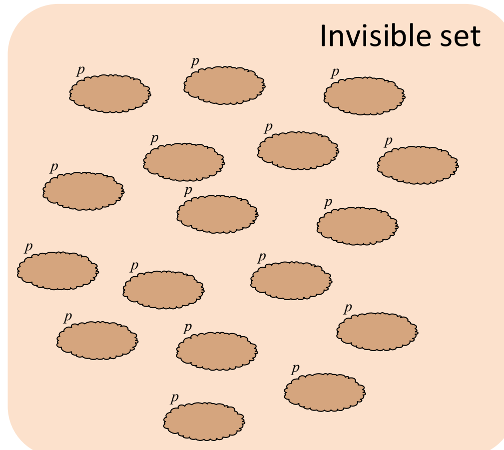


24

How to keep processes invisible ? Summary

- round r:

$$|P_r|$$



How to keep processes invisible ? Summary

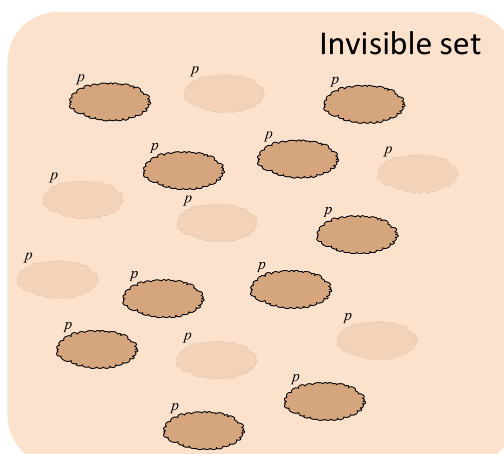
- round r:

$$|P_r|$$

- round $r + 1$:

$$|P_{r+1}| \geq \sqrt{\frac{|P_r|}{2d + 3}}$$

and at most 2 processes become visible



Summary: the lower bound on the amortized step complexity of graph-based-set

Any implementation of **graph-based-set** using only 1-revealing primitives has an execution

- with $\dot{c}$ processes
- each performing a single $add(\,)$ operation
- $\Omega(\dot{c}^2)$ steps by all processes

⇨ $\Omega(\dot{c})$ amortized step complexity of $add(\,)$ operation, provided $\dot{c} \in O\left(\sqrt{\log\log n}\right)$

27

Summary: the lower bound on the amortized step complexity of graph-based data structures

The amortized step complexity of any implementation of

- linked lists
- skip lists
- binary search trees
- B-trees
- other data structures based on graph-based-set

using only 1-revealing primitives is $\Omega(\dot{c})$, provided $\dot{c} \in O\left(\sqrt{\log\log n}\right)$

28

Thank you very much for your attention !

29