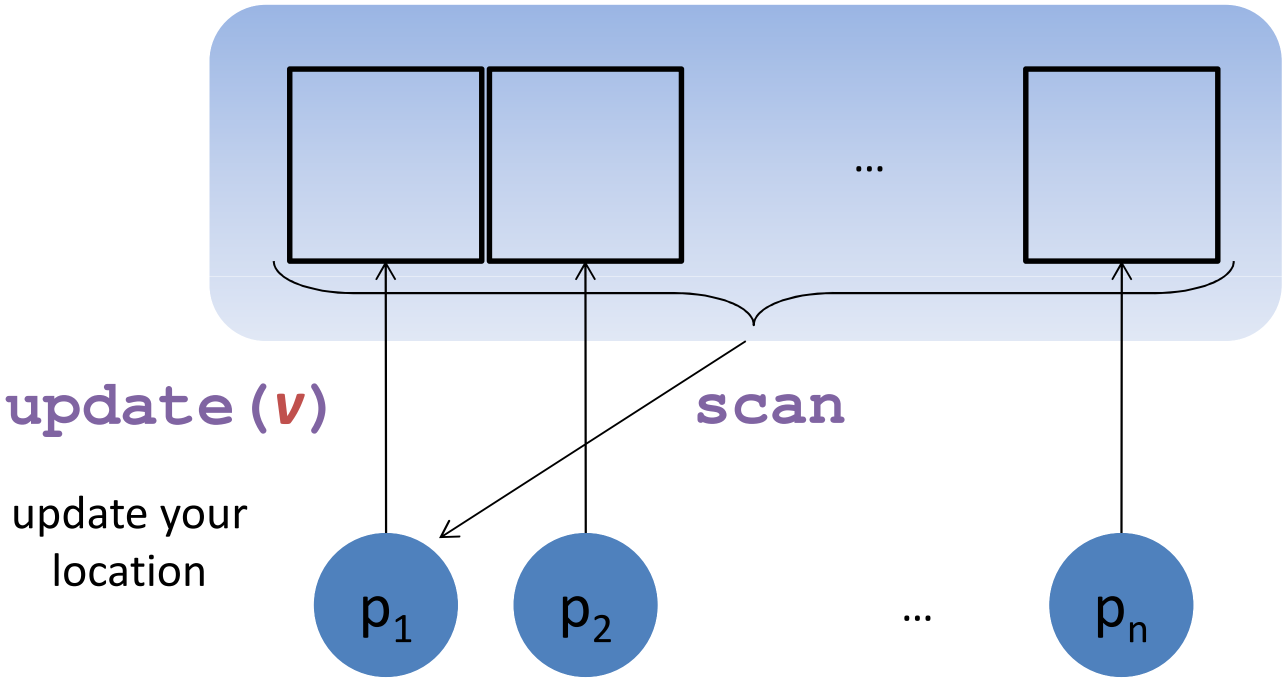
Faster than Optimal Snapshots (for a While)

James Aspnes, Yale University
Hagit Attiya, Technion
Keren Censor-Hillel, MIT
Faith Ellen, University of Toronto

Snapshot Objects



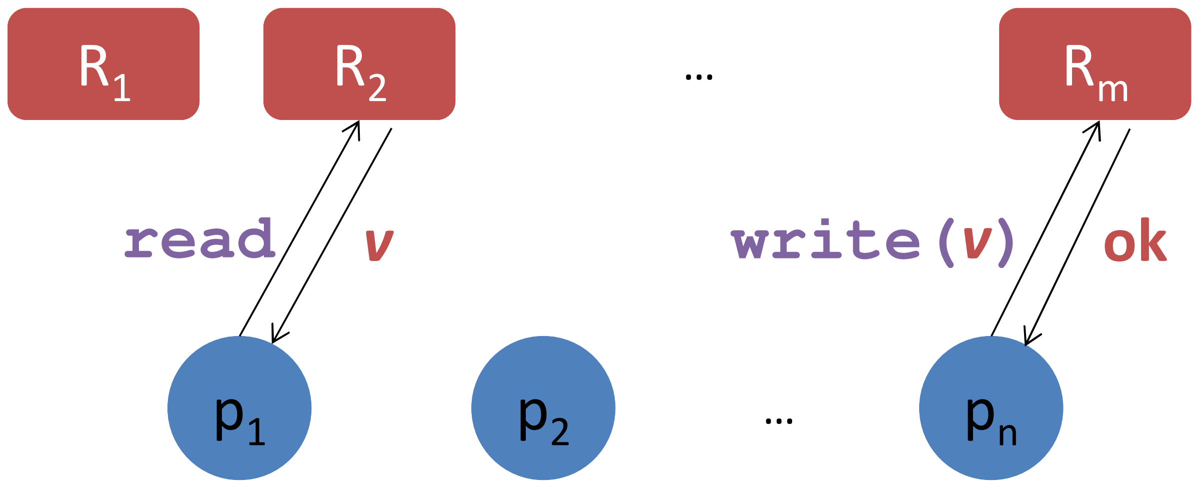
Model

System of ***n*** processes, ***m*** **multi-writer** registers

Asynchronous schedule controlled by an adversary

Crash failures – require **wait-free** implementations

Linearizable implementations



Snapshots - Step Complexity

Using multi-writer registers:

can be done in **O(n)** steps [Inoue and Chen, WDAG 1994]

and requires **Ω(n)** steps [Jayanti, Tan, and Toueg, SICOMP 1996]

Goal:

a faster snapshot implementation (sub-linear)

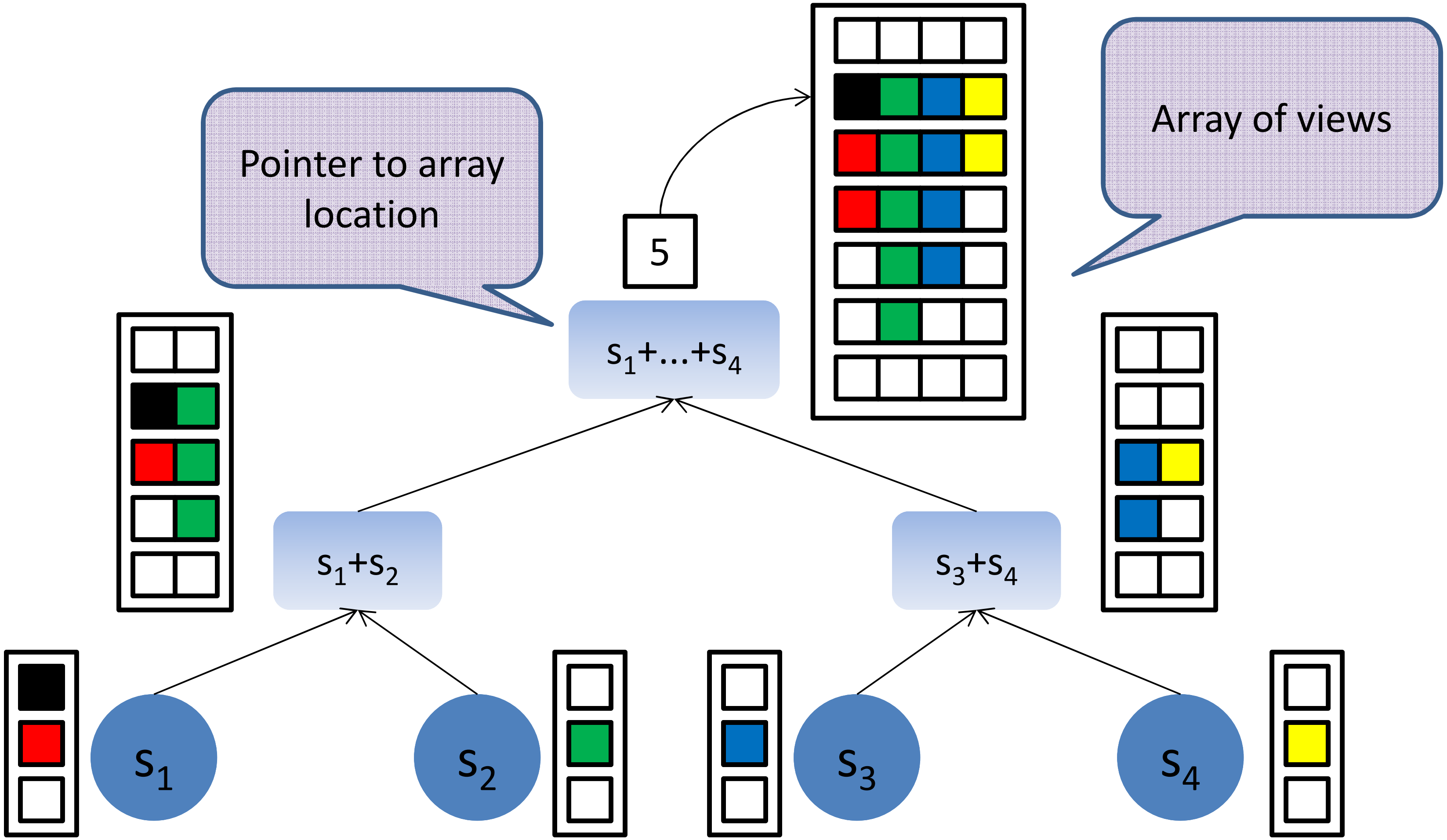
This talk:

snapshot implementation in **$O(\log^3(n))$** steps per operation

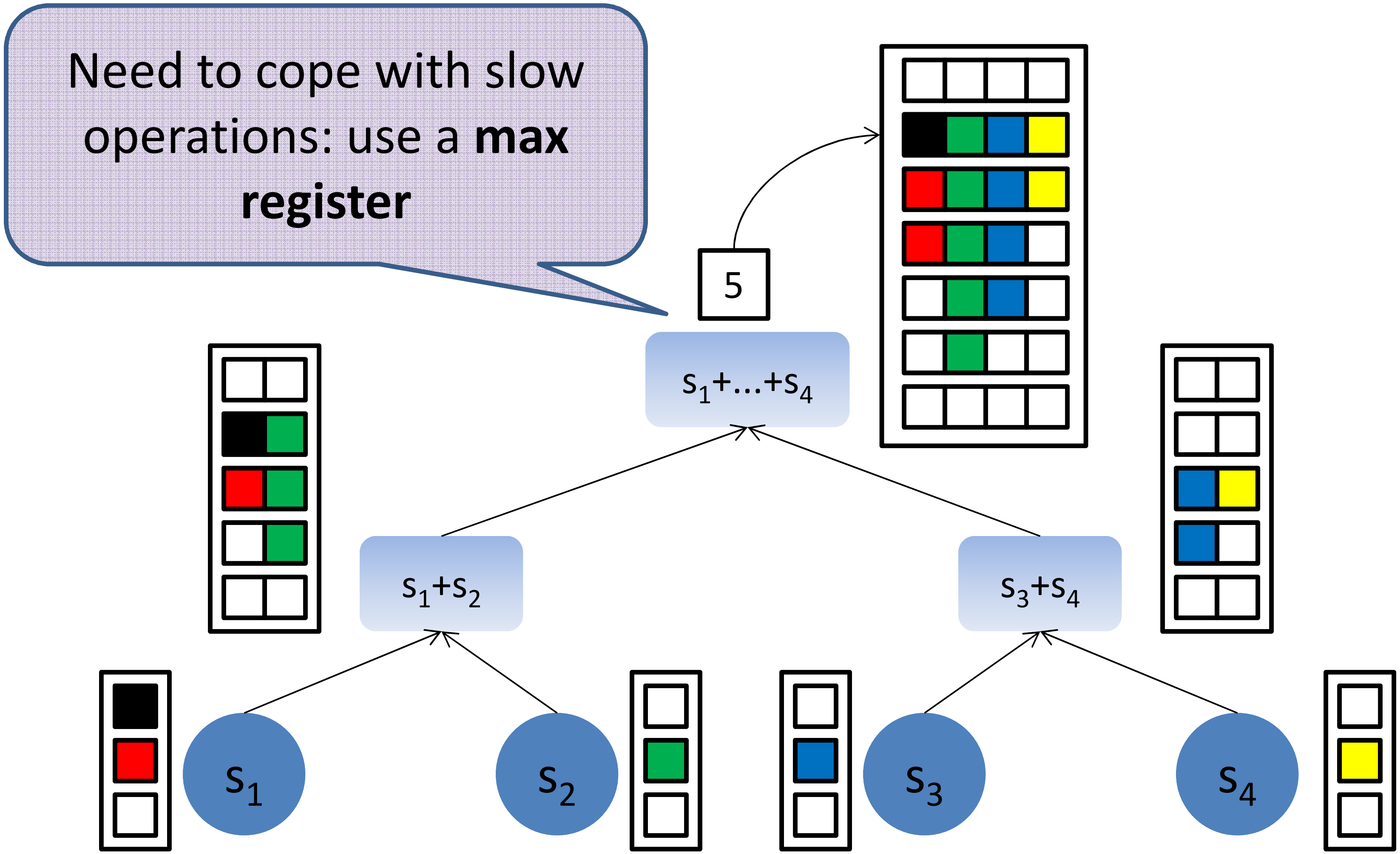
for polynomially many update operations

(limited-use snapshot object)

Tree structure, Updates help Scans



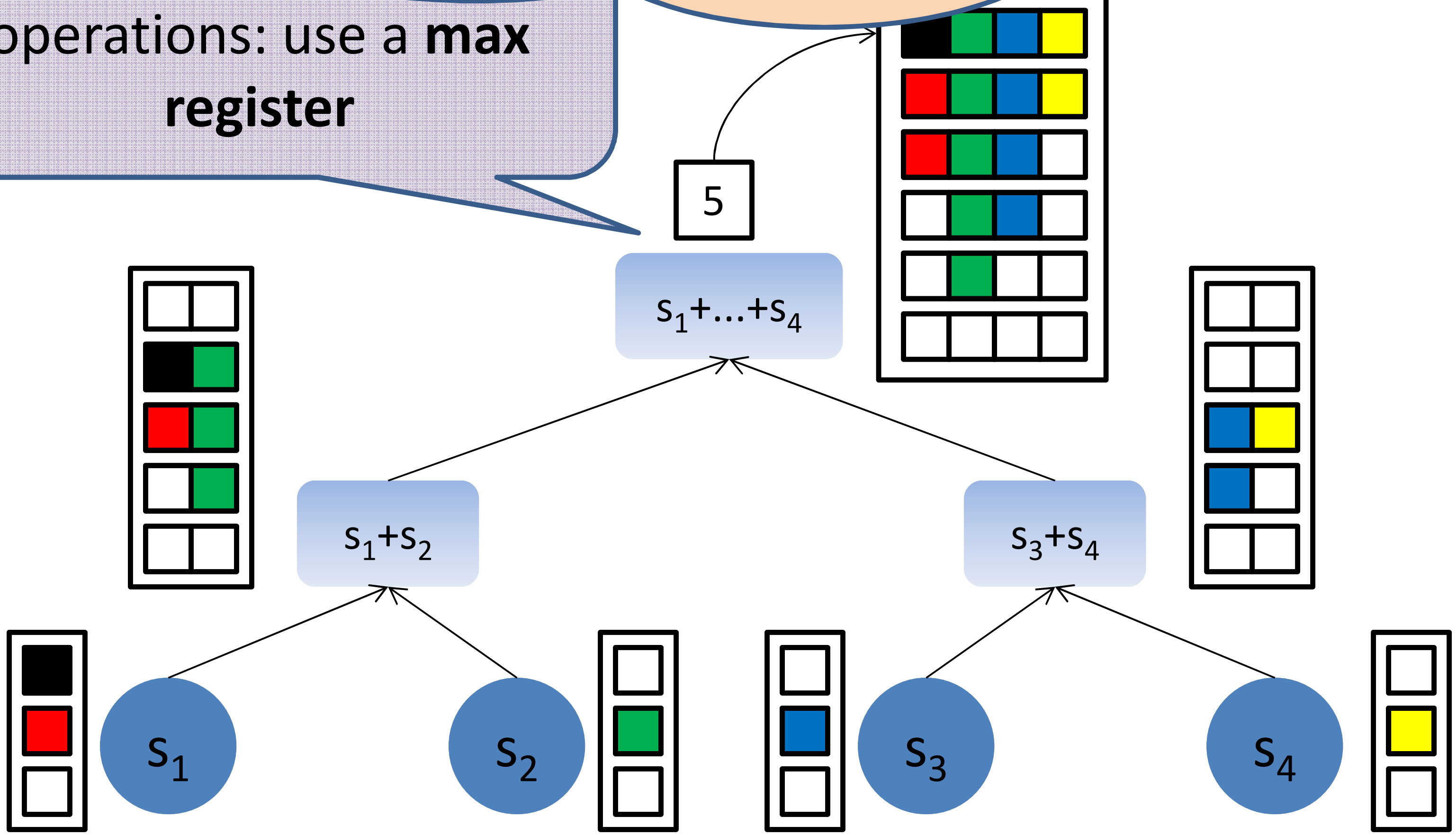
Polylogarithmic snapshots



Max register: returns largest value previously written

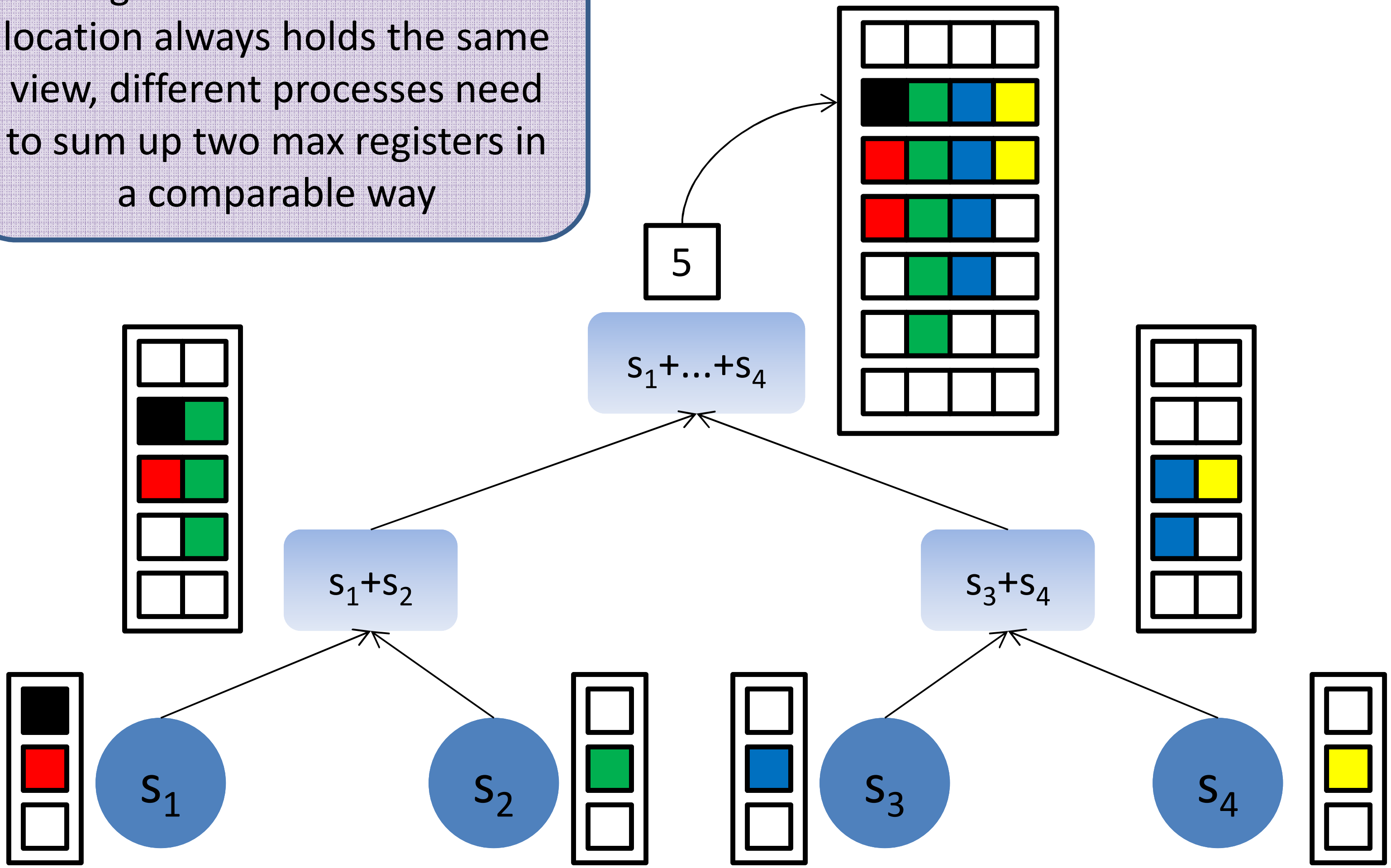
[Aspnes, Attiya, and Censor-Hillel, JACM 2012]

Need to cope … operations: use a **max register**

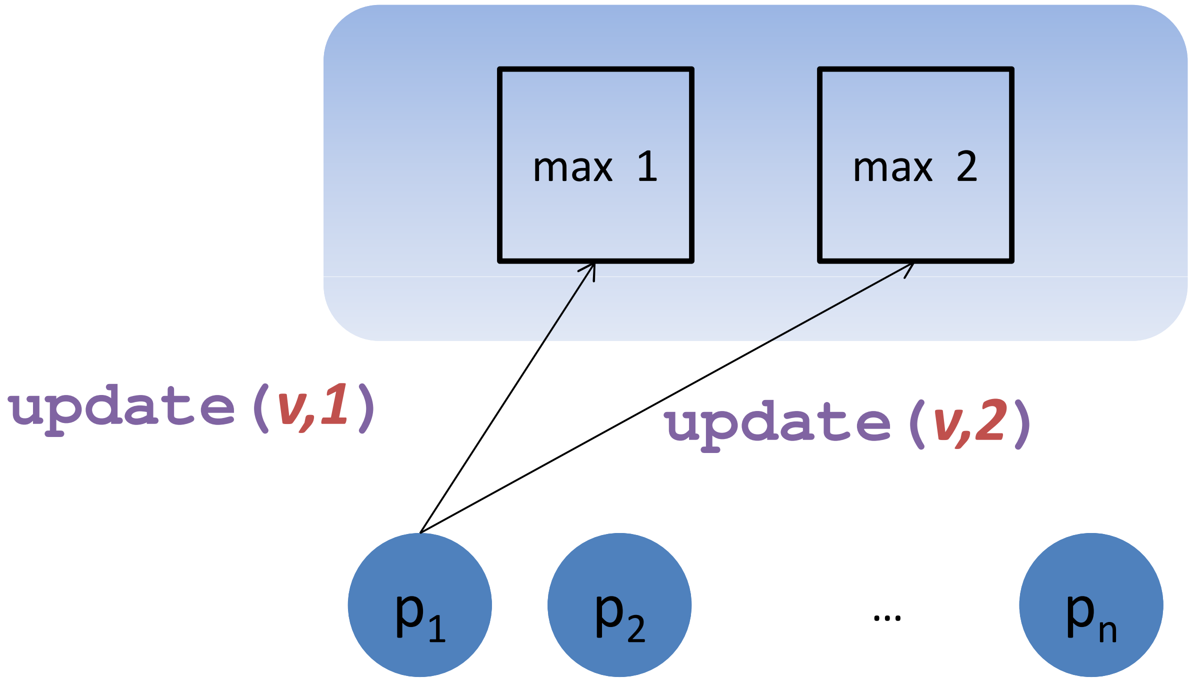


Polylogarithmic snapshots

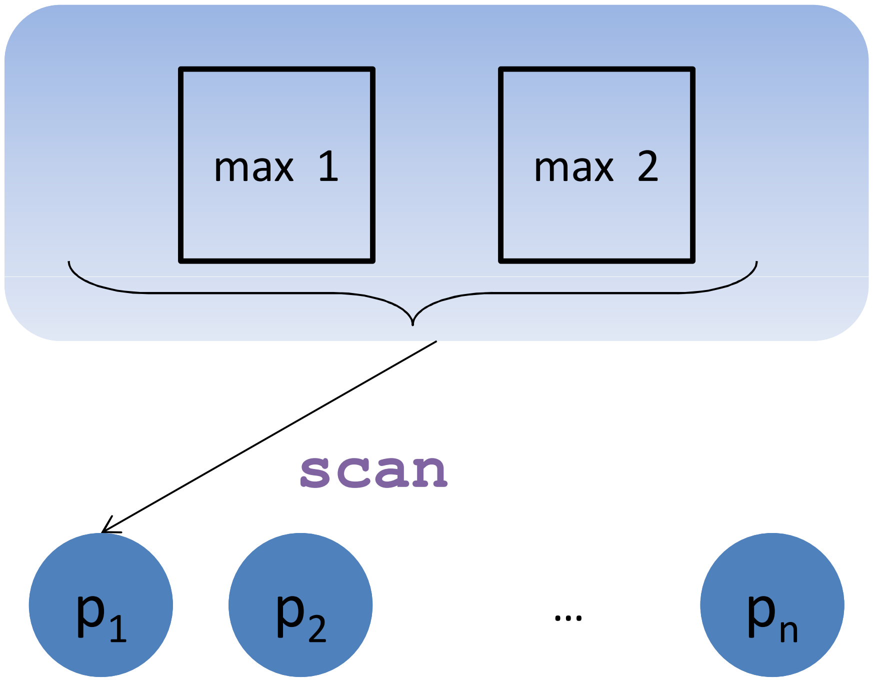
To guarantee that a view location always holds the same view, different processes need to sum up two max registers in a comparable way



2-component max array



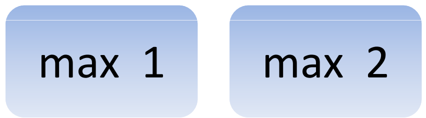
2-component max array



2-component max array

Simply reading one max register and then the other does not work

1. p_1 `read 0`
2. p_2 `write(`***100***`)`
3. p_3 `read 100`



4. p_3 `read 0`
5. p_2 `write(`***100***`)`
6. p_1 `read 100`

p_1 returns `(0,100)`

p_3 returns `(100,0)`

2-component max array

Read max registers again to see if they change

– Might change many times

– What if they were only binary?

(0,0) and (1,1) are comparable with any pair

If you see (0,1) or (1,0) read again

1. p_1 **read 0**
2. p_2 **write(*100*)**
3. p_3 **read 100**

max 1 | max 2

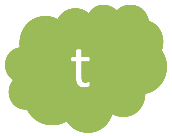
4. p_3 **read 0**
5. p_2 **write(*100*)**
6. p_1 **read 100**

Max register – recursive construction

[Aspnes, Attiya, and Censor-Hillel, JACM 2012]

- $MaxReg_k$ supports values in {0,…,k-1}
  - Built from two $MaxReg_{k/2}$ objects with values in {0,…,k/2-1}
  - and one additional multi-writer register “switch”

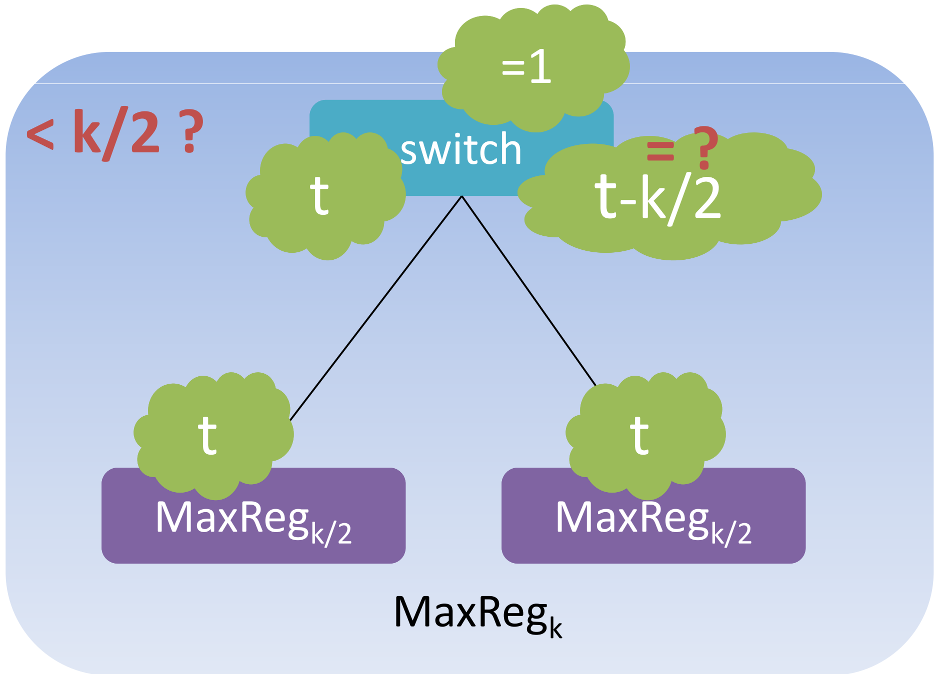
WriteMax



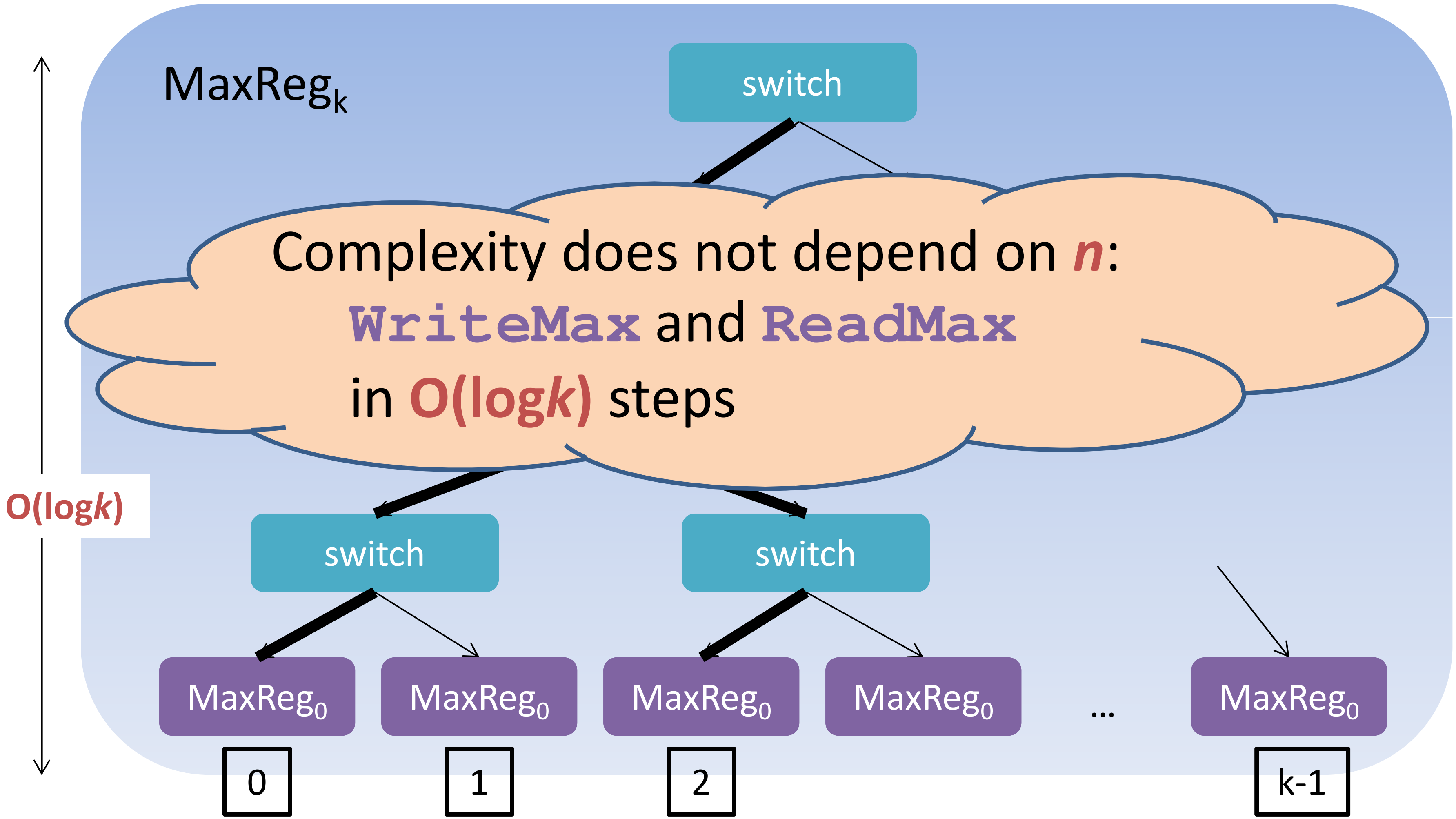
ReadMax

switch=**0** : return **t**

switch=**1** : return **t+k/2**



MaxReg$_k$ unfolded



A 2-component max array

Write

Read

x=ReadMax component 2

switch=**0** :

    WriteMax(x,2) to left subtree

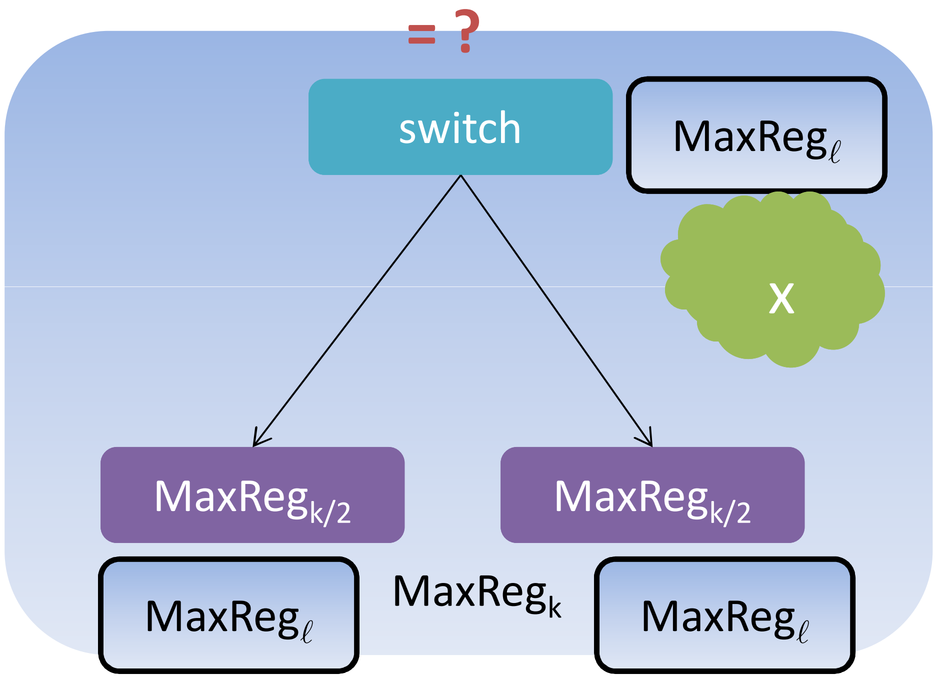
    Return (**Read** left subtree)

switch=**1** :

    x=ReadMax component 2

    WriteMax(x,2) to right subtree

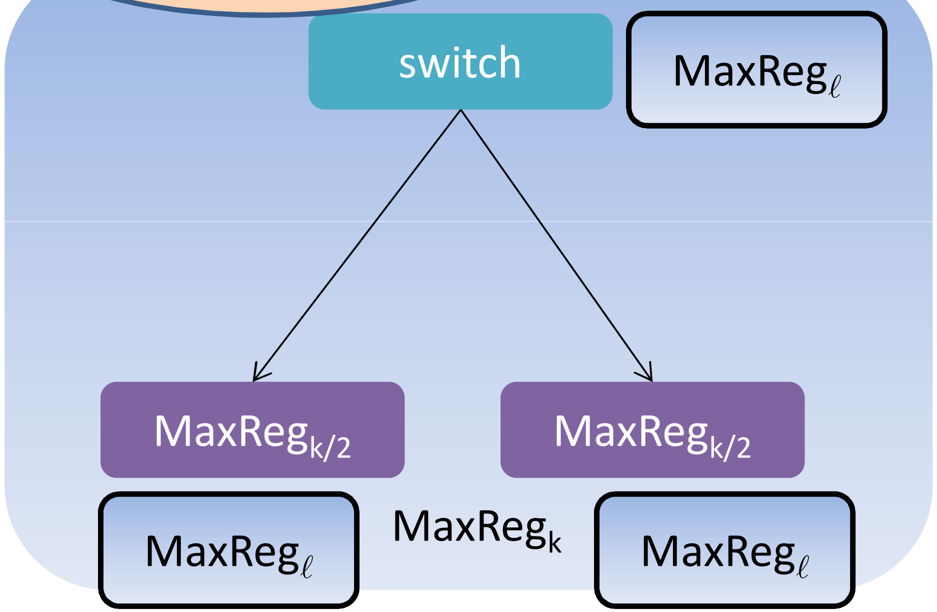
    Return (k/2,0)+(**Read** right subtree)



Key idea:
a reader going right at the switch always sees a value for component 2 that is at least as large as any value that a reader going left sees

Write

Read



x=ReadMax component 2

switch=**0** :

  WriteMax(x,2) to left subtree

  Return (**Read** left subtree)

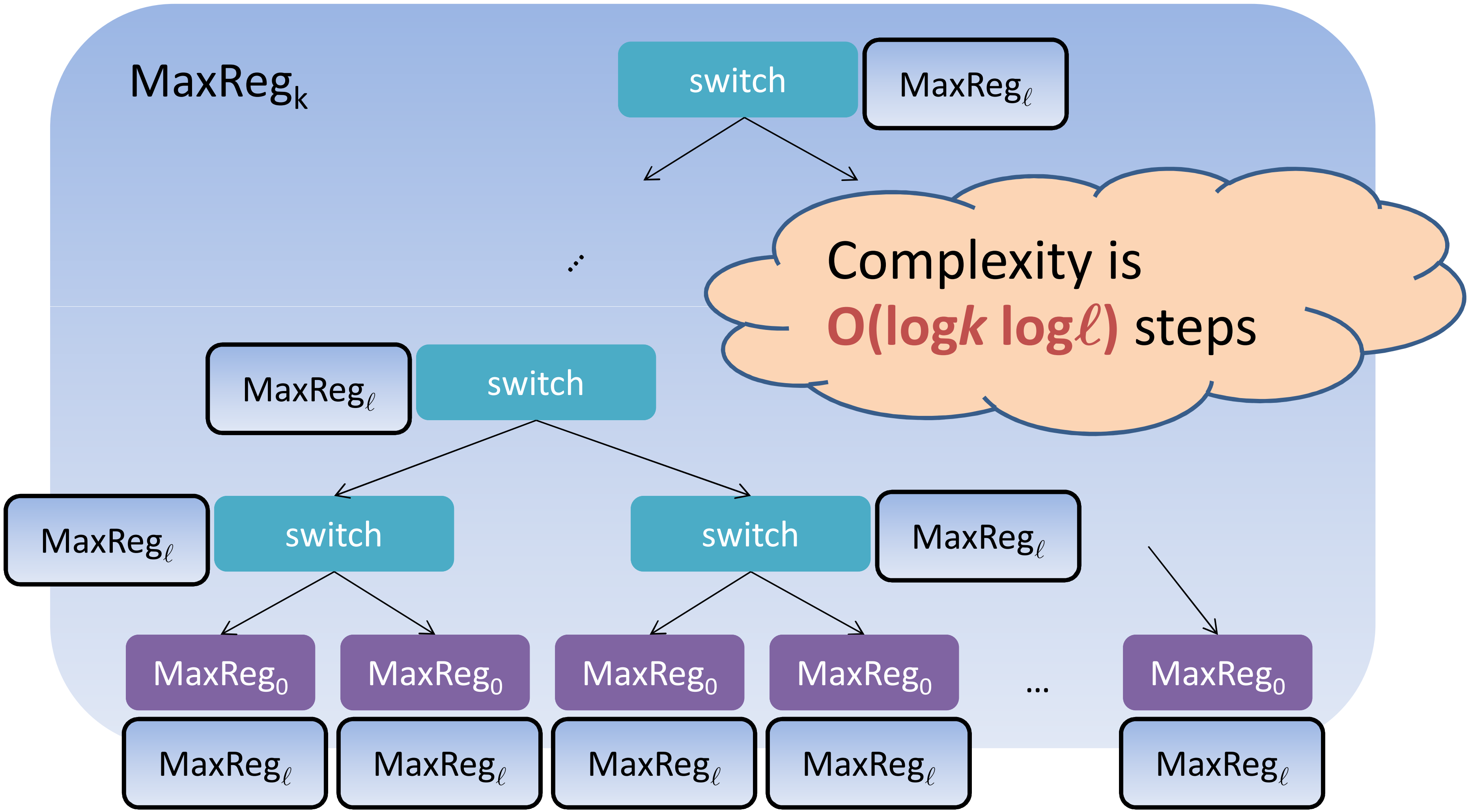
switch=**1** :

  x=ReadMax component 2

  WriteMax(x,2) to right subtree

  Return (k/2,0)+(**Read** right subtree)

A 2-component max array unfolded



Summary

For **b**-limited use snapshot we get **$O(\log^2 b \log n)$** steps

- This is **$O(\log^3(n))$** steps for polynomially many updates

Paper also shows:

- Multi-writer snapshot implementation: every process can update each location
- c-component max arrays

Open problems:

- Snapshot implementations using single-writer registers
- Lower bounds
- Randomized implementations and lower bounds