A Randomized Concurrent Algorithm for Disjoint Set Union

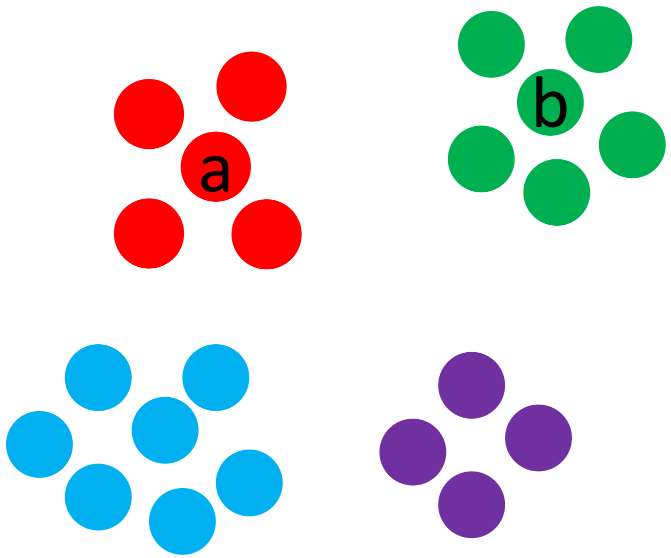
SIDDHARTHA V. JAYANTI

ROBERT E. TARJAN

PRINCETON UNIVERSITY

Set Union Data Type

Maintain n nodes in **disjoint sets**



$\boldsymbol{Unite(x, y)}$ – update operation

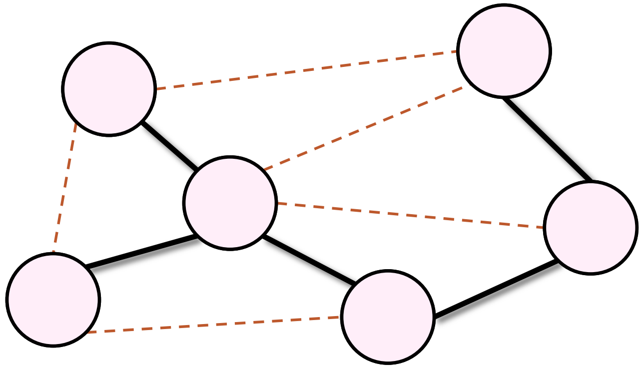
$\boldsymbol{SameSet(x, y)}$ – query operation

Initially nodes are in singleton sets.

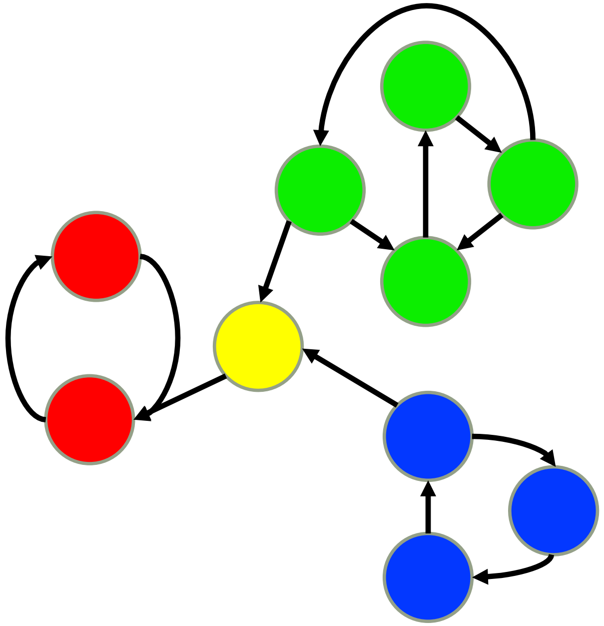
$SameSet(a, b)$	False
$Unite(a, b)$	True
$SameSet(a, b)$	True

Applications

- FORTRAN compilers: COMMON and EQUIVALENCE statements
- Incremental **connected components**
- Finding dominators in flow graphs
- **Spanning tree / forest**
- Percolation
- Strongly Connected Components
- Model Checking



Spanning Trees



Finding Strongly Connected Components

Road Map

- Review famous sequential algorithm
- Previous Attempt at Concurrent Data Structure
- Our Concurrent Data Structure

Implementation
[Galler and Fischer]

Set = Rooted Tree with **parent** pointers

$\{a, b, c, d, e, f, g\}$

$\{x, y, z\}$

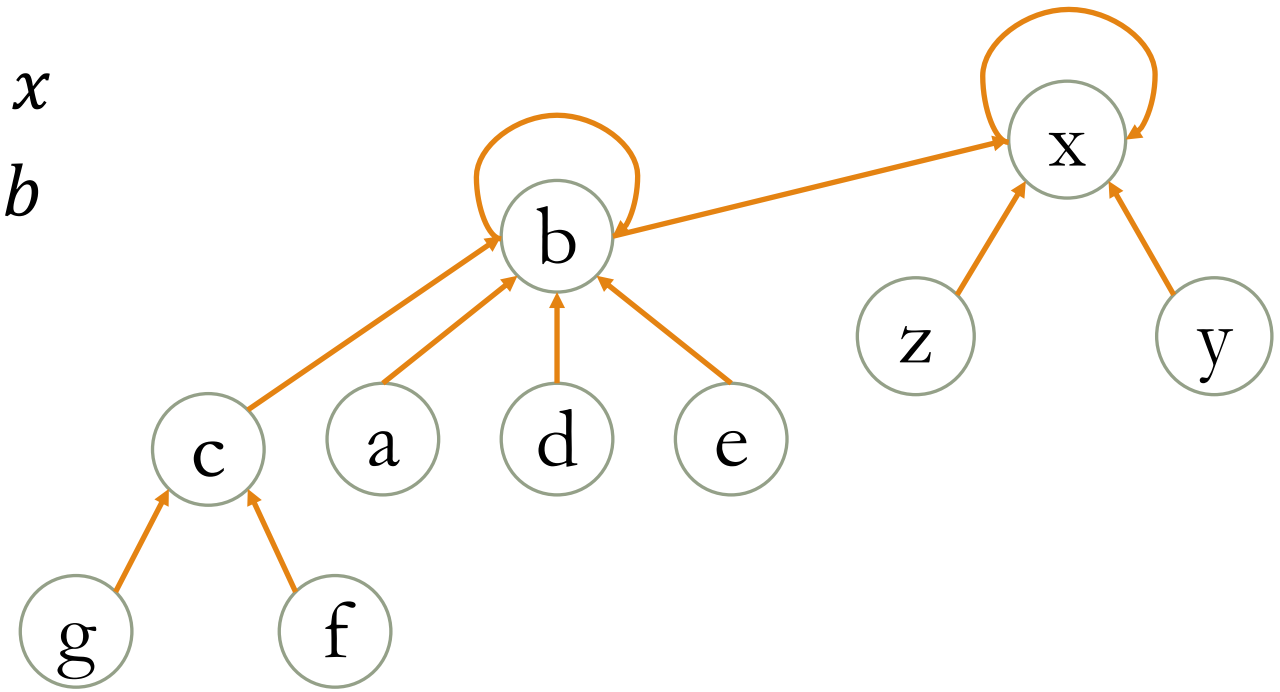
Primitives:

$\boldsymbol{Link(b, x)}$: make $b.parent = x$

$\boldsymbol{or}\ x.parent = b$

$\boldsymbol{Find(c)}$: return root of c's tree

(returns x in this case)



Implementation

$\boldsymbol{SameSet(x, y)}$:

$u = Find(x)$

$v = Find(y)$

$return\ (u = v)$

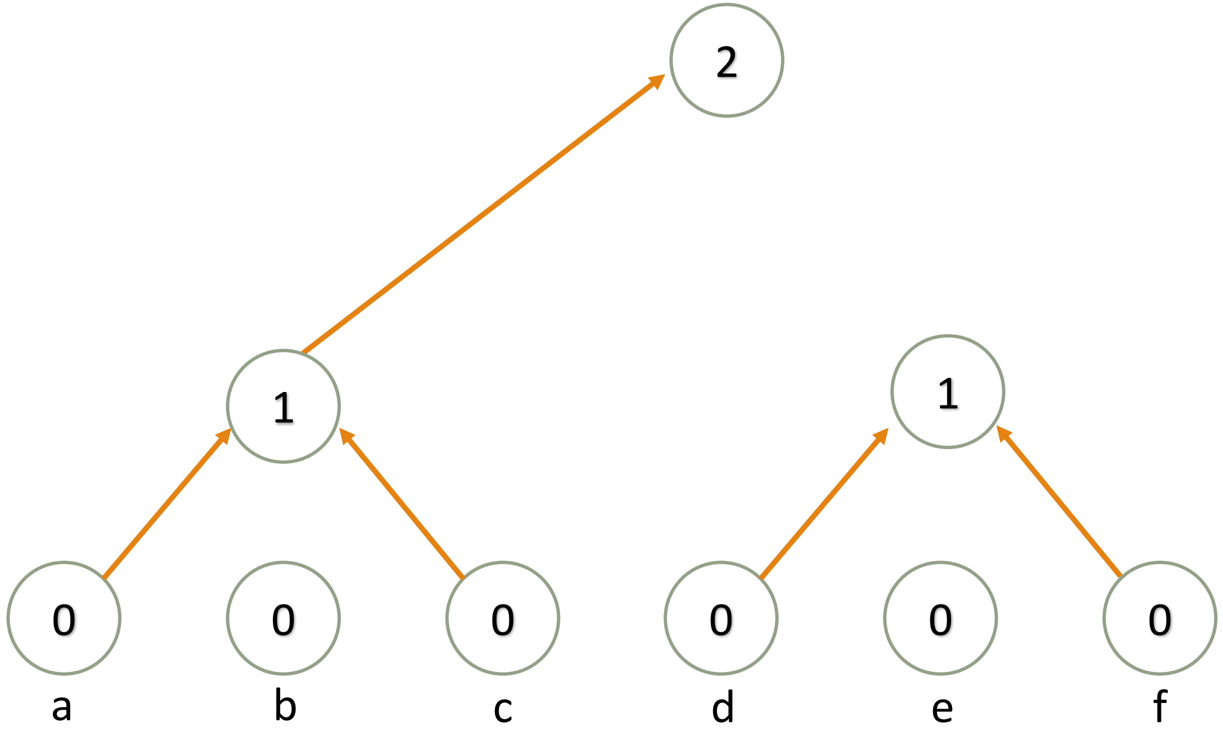
$\boldsymbol{Unite(x, y)}$:

$u = Find(x)$

$v = Find(y)$

$if\ (u \neq v)\ Link(u, v)$

Linking by Rank



Link(b, c)

Link (a, b)

Link (e, f)

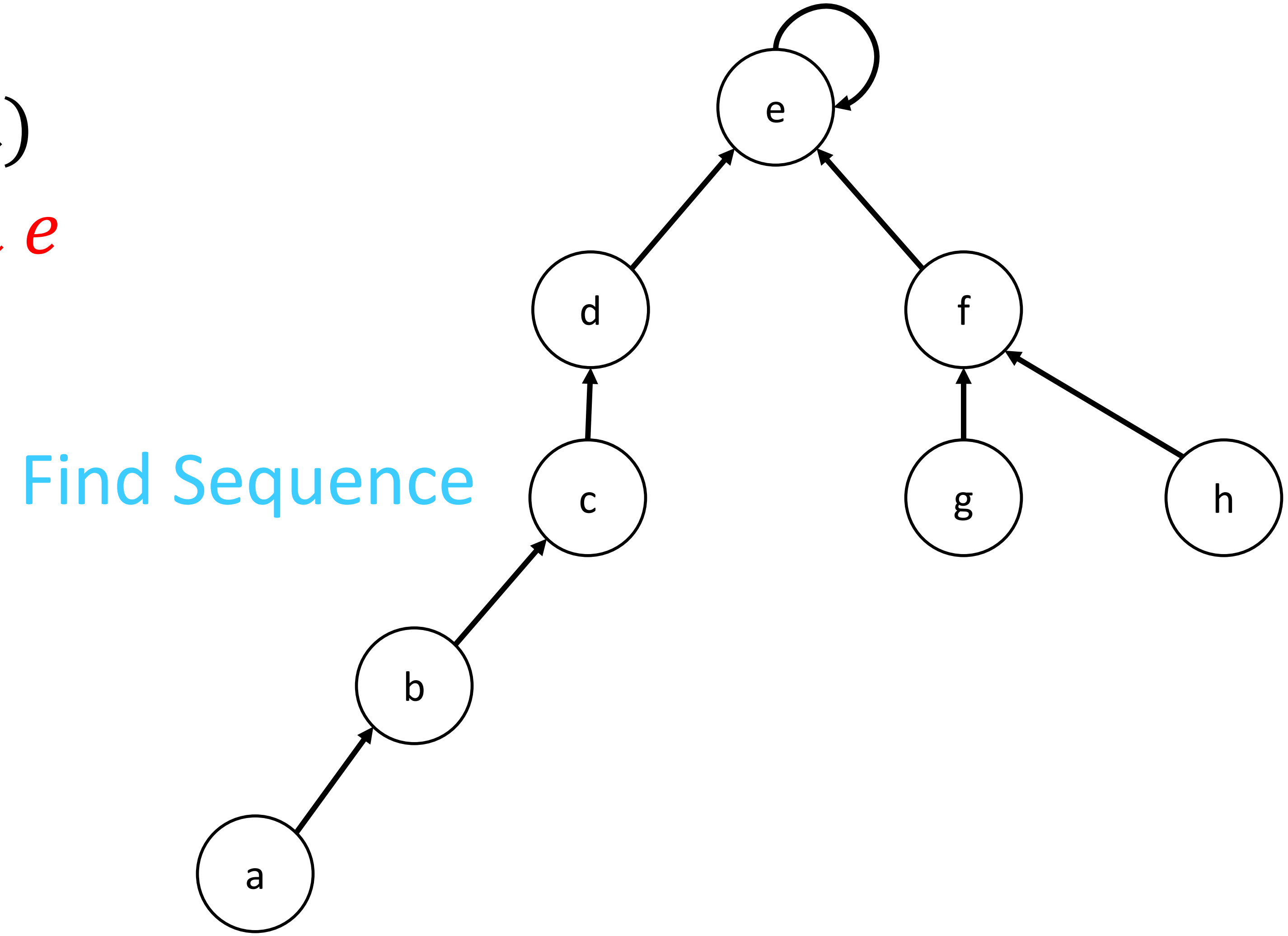
Link(d, e)

Link(b, e)

Find

$$\boldsymbol{Find(a)}$$

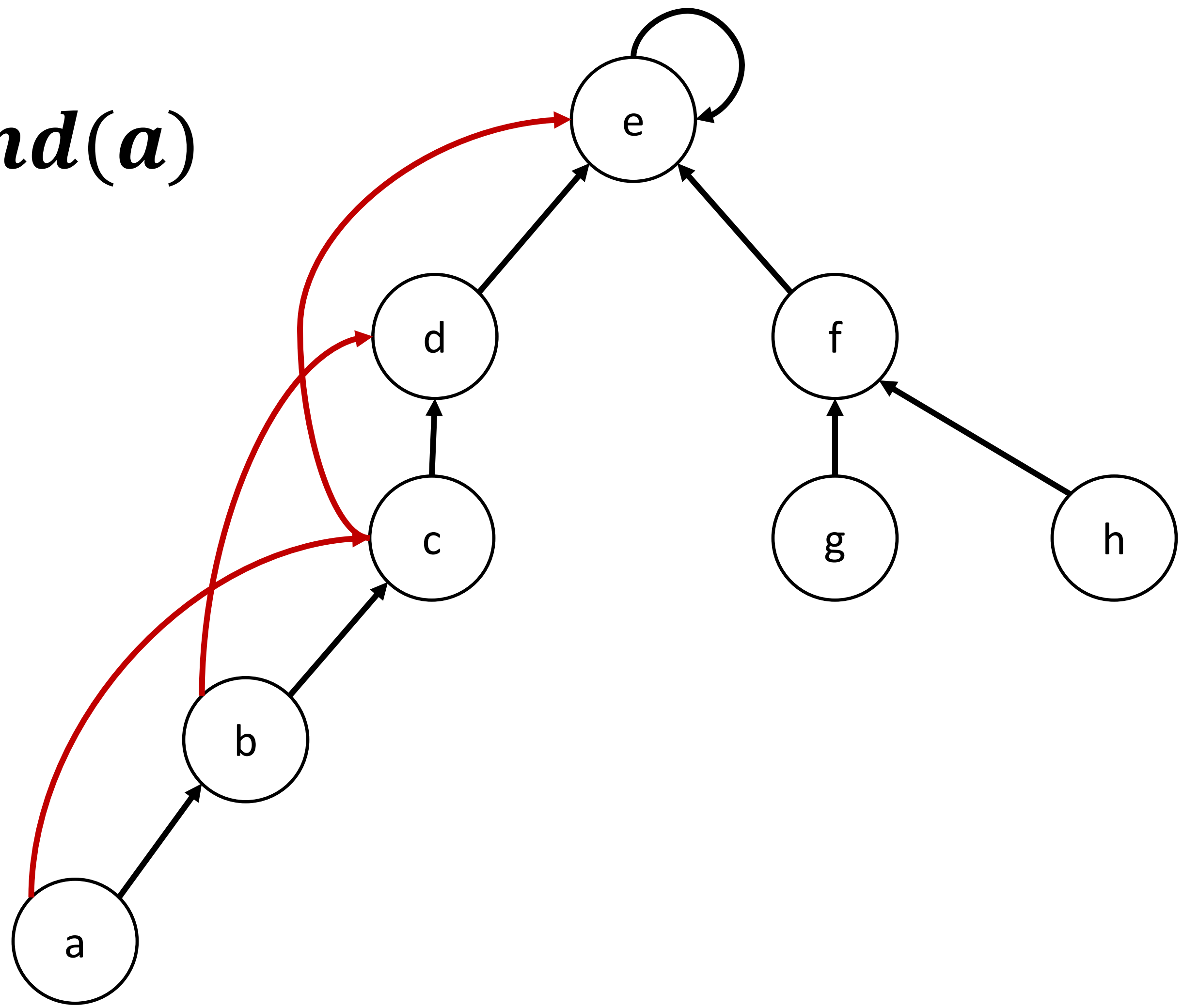
$return\ e$



Find with Splitting

$\boldsymbol{SplittingFind(a)}$

$return\ e$



Ackermann's Function

- $A_{\mathrm{k}}(n)$ – highly super-exponential function
- $A_4(2)$ more than number of particles in observable universe
- $\alpha(n, d) = \min\{k > 0 | A_k(d) > n\}$
- $\alpha(n, d)$ is practically bounded by 4

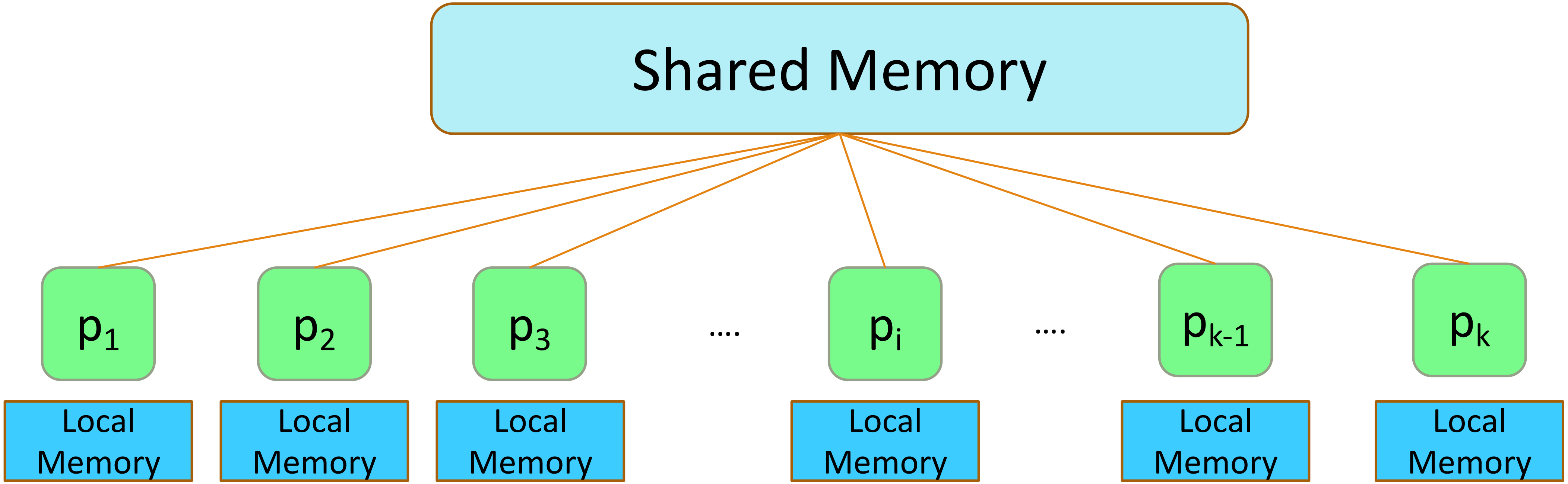
Time Complexity [Tarjan, van Leeuwen 1984]

Find with Splitting	Linking by Rank	Amortized Time per Operation
	✓	
✓		
✓	✓	

m – number of operations, n – number of nodes

Computational Model

Asynchronous Shared Memory Machine

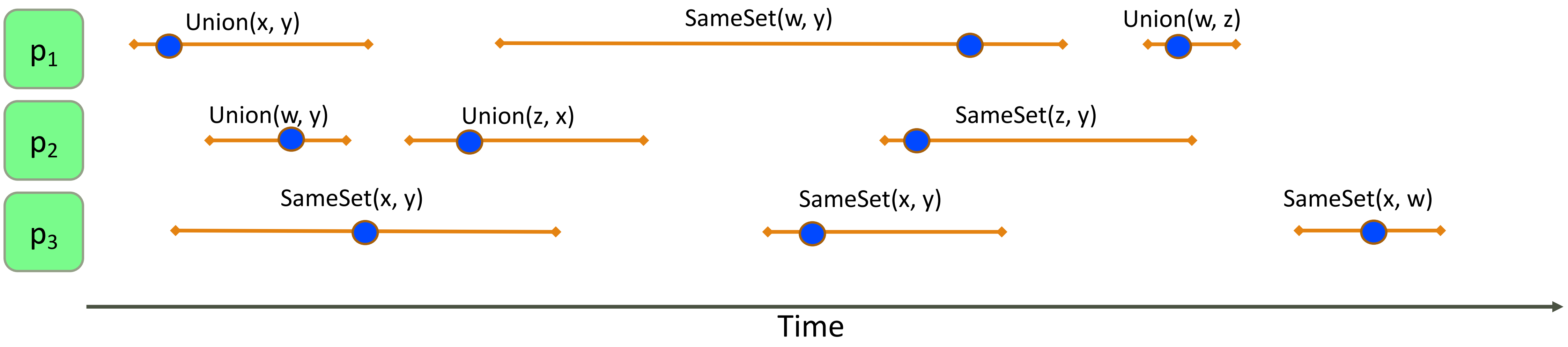


Compare and Swap

procedure CAS(x, x_0, x_1)
    if $x = x_0$ **then**
        $x \leftarrow x_1$
        return $true$
    else
        return $false$

Correctness Criteria

Linearizability [Herlihy, Wing 1990]:



Wait-Freedom [Herlihy 1991]:

Each p_i should complete operation in a bounded number of its steps.

Work

- $\boldsymbol{W_j}$ = number instructions executed by **p**$_j$

- **Total work** $W = \sum_{j=1}^{k} W_j$

- For sequential algorithm, work = time

Previous Algorithm
[Anderson and Woll, 1991]

Extends linking by rank algorithm

$\boldsymbol{n}$ nodes

$\boldsymbol{m}$ operations

$\boldsymbol{p}$ processes

Amortized work per operation $\Theta(\alpha(m,1)+p)$?

Hard to maintain **both** rank and parent

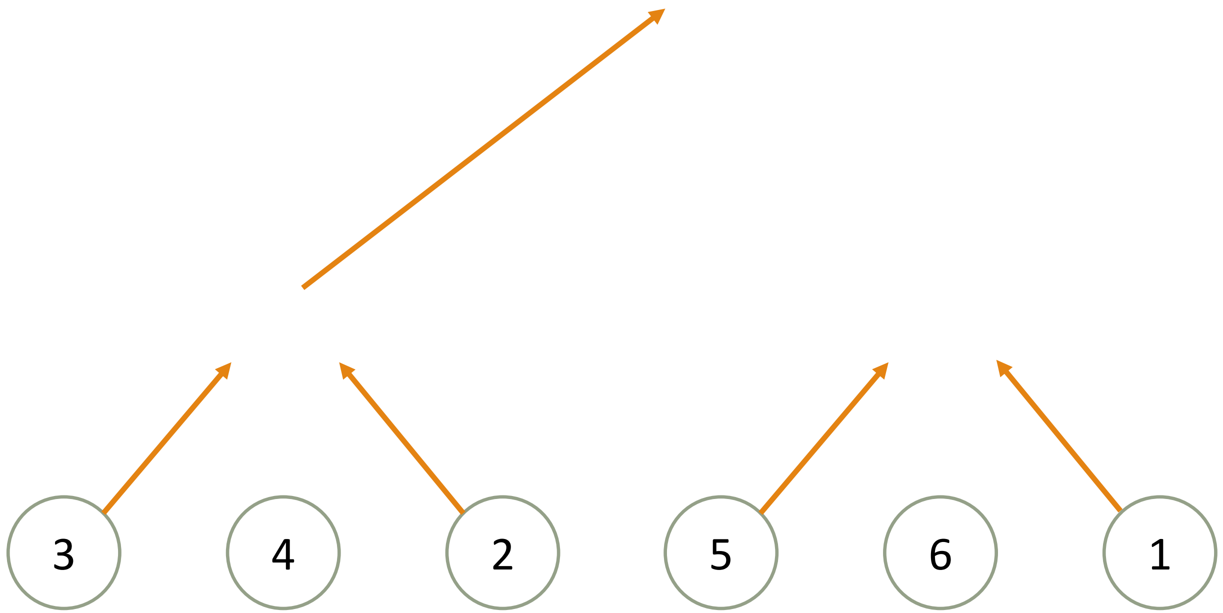
per operation work is linear in p

Linking by ID

[Goel, Khanna, Larkin, Tarjan 2014]

- The nodes are given IDs $1,2,\ldots,n$ uniformly at **random**

- Link winner determined by **fixed ID** rather than **changing rank.**



Link(4, 2)

Link(3, 4)

Link(6, 1)

Link(5, 6)

Link(4, 6)

Time Complexity [Goel, Khanna, Larkin, Tarjan 2014]

Find with Splitting	Linking by ID	Expected-amortized Time per Op
		$\Theta(n)$
	✓	$\Theta(\log n)$
✓		$\Theta\left(log_{1+\frac{m}{n}} n\right)$
✓	✓	$\Theta\left(\alpha\left(n, \frac{m}{n}\right)\right)$ (optimal in cell-probe model)

The same results in expectation!

Concurrent Link(u, v)

Link(u, v)

$\quad$ *if* $(v < u)$ *swap(u, v)*

$\quad$ return $\text{CAS}(u.parent, u, v)$

- CAS succeeds iff u is a root
- v is possibly not a root

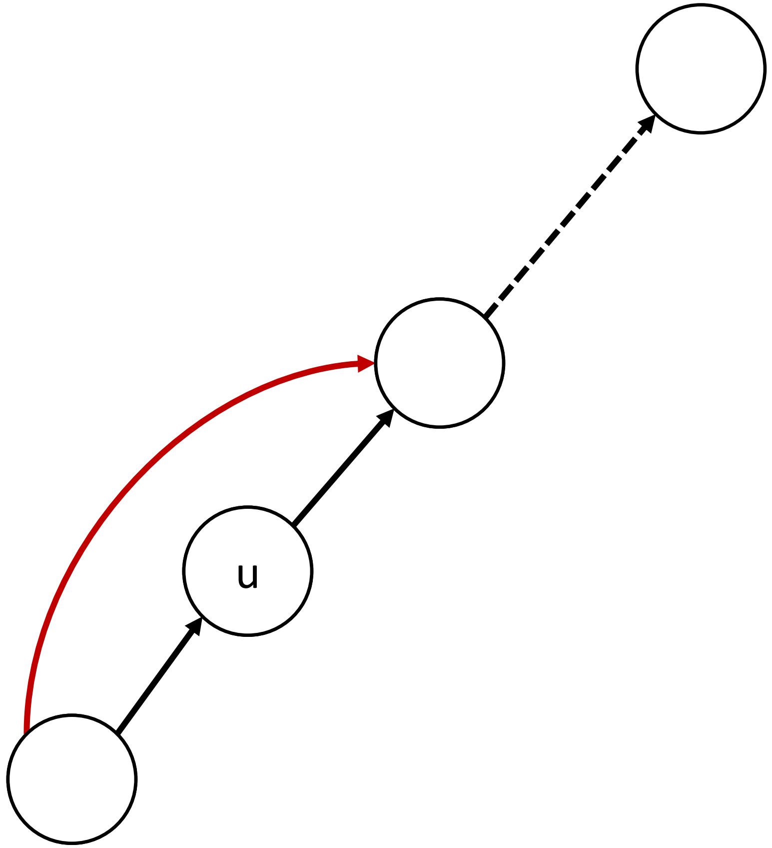
Concurrent Find(x)

$Find(x)$

```
u = x

while (u not root)*
        v = u.parent, w = v.parent
        CAS(u.parent, v, w)
        u = v

return u
```



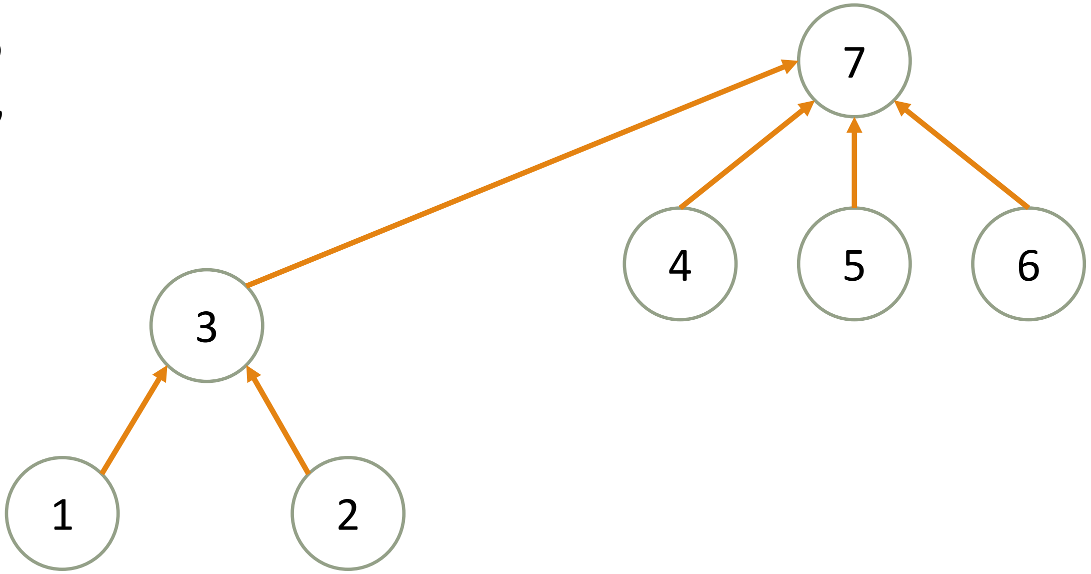
Difficulty with Parallelization

Computation can be invalidated

p_1 SameSet(1, 2)

Root of 1 is 3
Root of 2 is 7
return false

p_2 Unite(3, 7)



Unite Implementation

Unite(x,y)

u = Find(x)

v = Find(y)

if (**u = v**), *return false*

if Link(u, v)*, *return true*

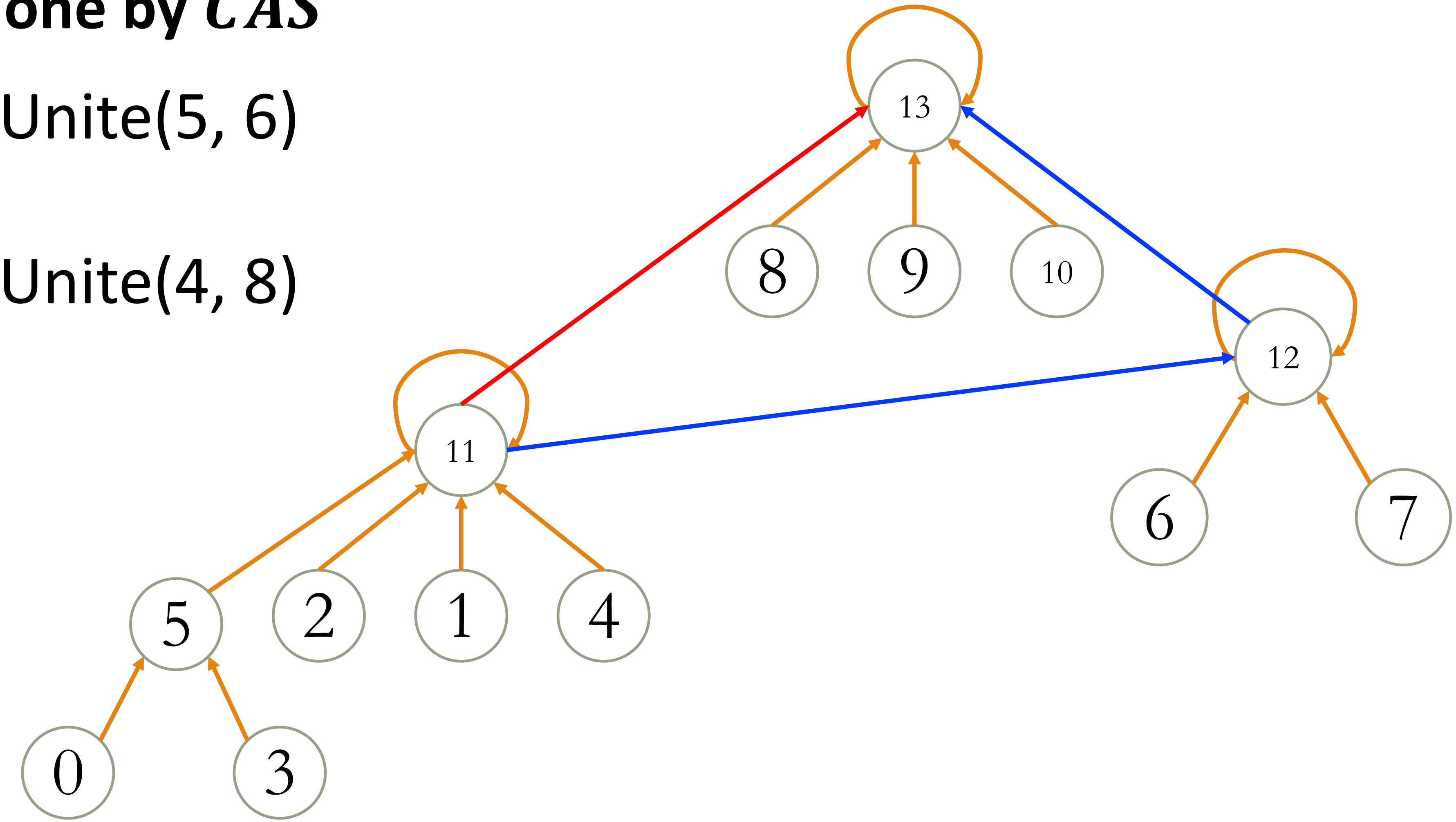
TRY AGAIN (occurs at most n times)

Unite Implementation

Links done by CAS

p₁ Unite(5, 6)

p₂ Unite(4, 8)



SameSet Implementation

SameSet(x,y)

u = Find(x)

v = Find(y) *

if (**u = v)**, *return* ***true***

else if (**u** still root), *return* ***false***

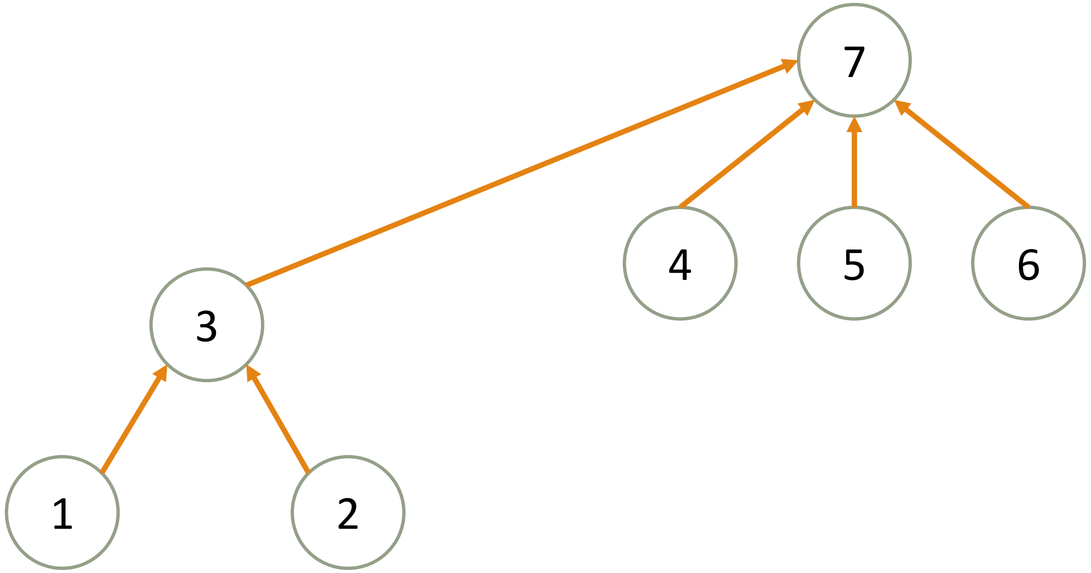
TRY AGAIN (occurs at most n times)

Problem Fixed

p_1 SameSet(1, 2)

Root of 1 is ~~3~~ 7
Root of 2 is 7
3 not root!
return true

p_2 Unite(3, 7)



Main Theorem

m operations , *n* nodes , *p* processors

Expected-amortized work per operation

$$\Theta\left(\alpha\left(n, \frac{m}{np}\right) + \log\left(\frac{np}{m} + 1\right)\right)$$

*assuming ID order and linearization order are independent

Main Theorem Part 2

m operations , n nodes , p processors

Worst-case work per operation **whp**

$$\mathbf{O}(\log n)$$

*assuming ID order and linearization order are independent

Current State-of-the-Art

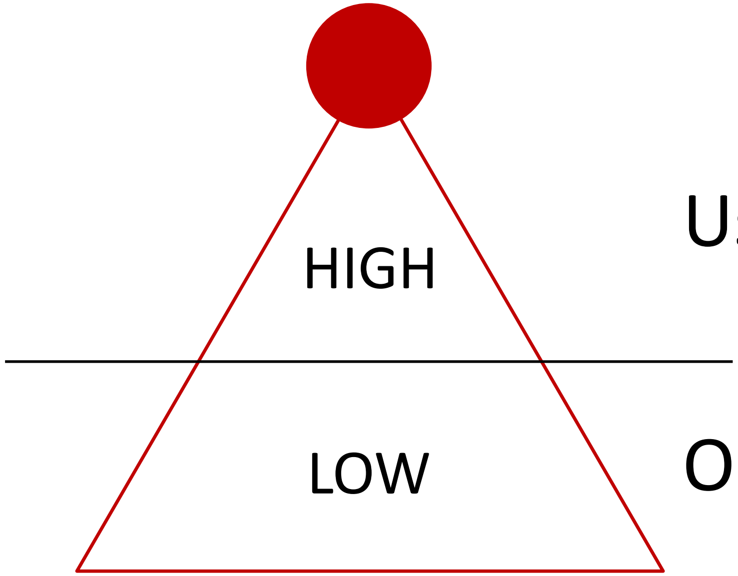
- Randomized algorithm with same efficiency under **no assumption**
- **Deterministic algorithm** (only a loglog p extra overhead!)
- We think work bound is **optimal**, we have shown a lower bound:

$$\boldsymbol{\Omega}\left(\alpha\left(n,\frac{m}{n}\right)+\operatorname{loglog}\left(\frac{np}{m}+1\right)\right)$$

Thanks!

Upper Bound Proof Idea

- Define d = $\frac{m}{np}$
- If d > 1, extend sequential analysis
- If d < 1



Use d > 1 argument

O(log p) height

Lower Bound Example

- Let us consider the case $\mathbf{\Theta}(m) = \mathbf{\Theta}(n) = \mathbf{\Theta}(p)$

- Perform $SameSet(x, x)$ with each processor (random x)

- Per operation work = $\log p$

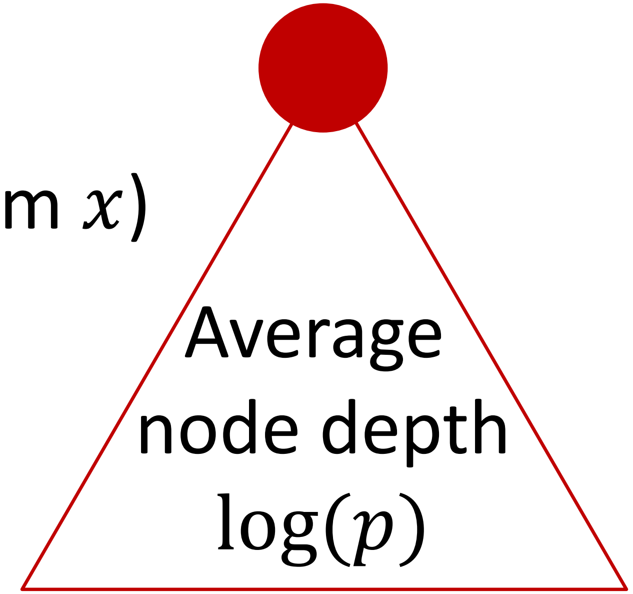
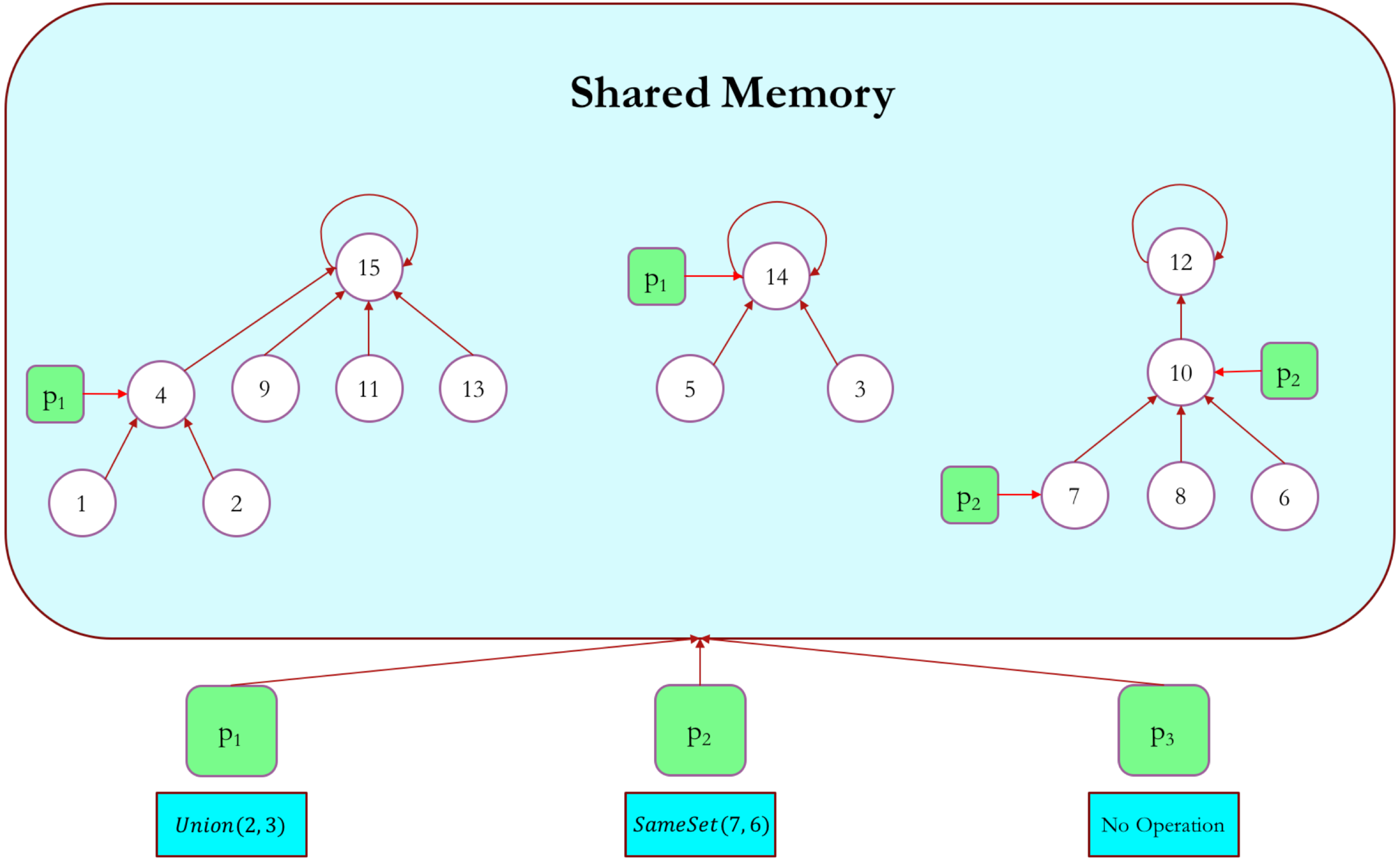


Illustration of our Solution



Correctness Criteria

Linearizability [Herlihy, Wing 1990]:

Each operation appears to take effect instantaneously at some point between its invocation and return.

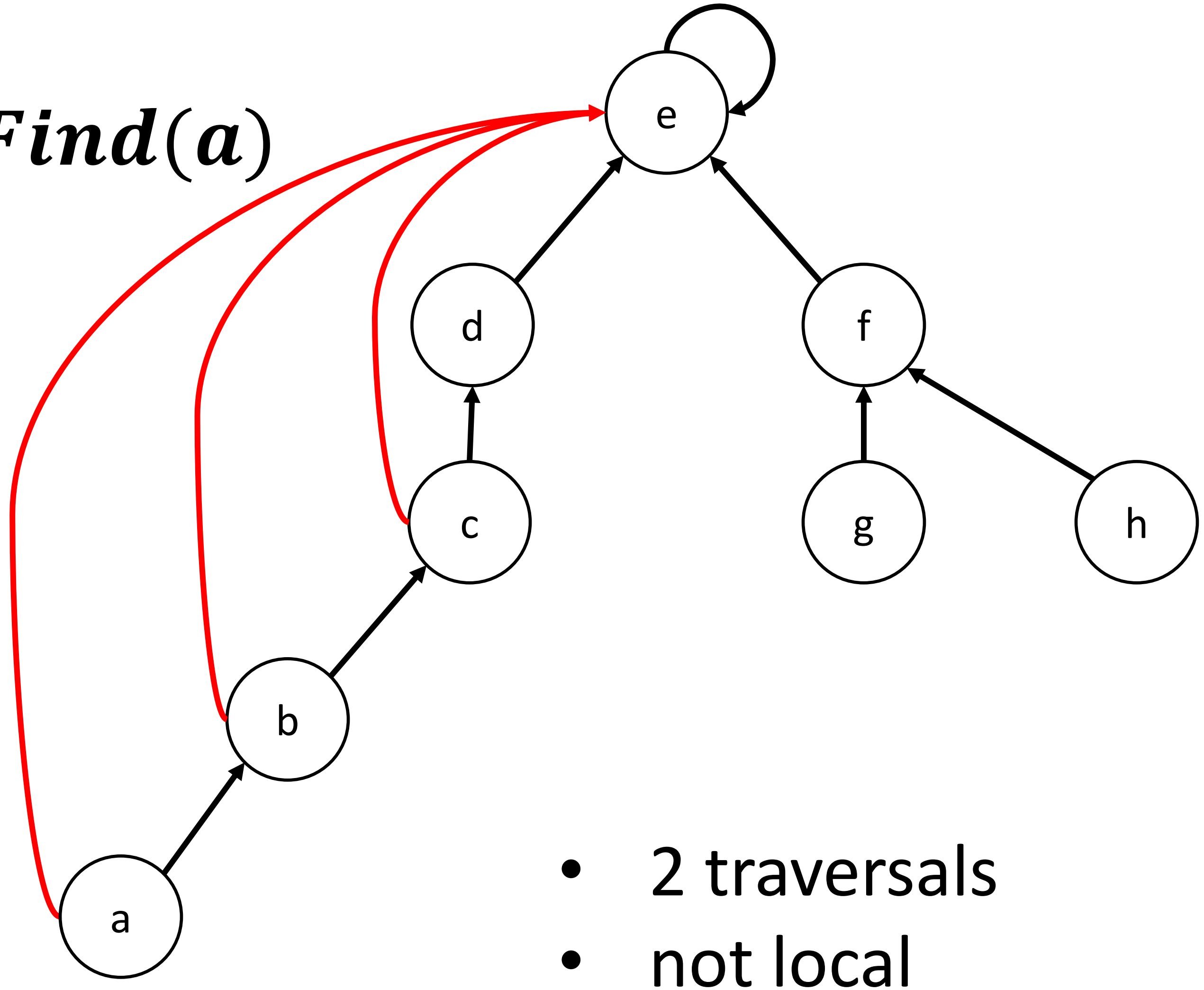
Wait-Freedom [Herlihy 1991]:

Each process completes each operation in a finite number of its own steps.

Find with Compression

$\boldsymbol{CompressFind(a)}$

$return\ e$



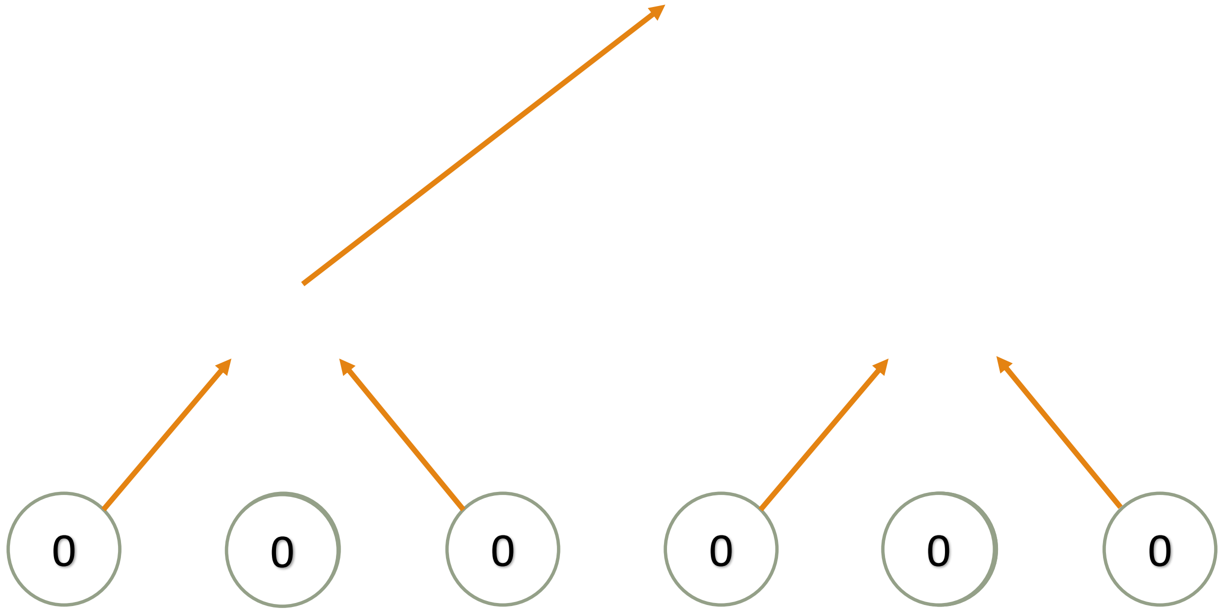
- 2 traversals
- not local

Time Complexity [Goel, Khanna, Larkin, Tarjan 2014]

Find with Splitting	Linking by randomized ID	Expected Time per Operation
	✓	
✓		
✓	✓	

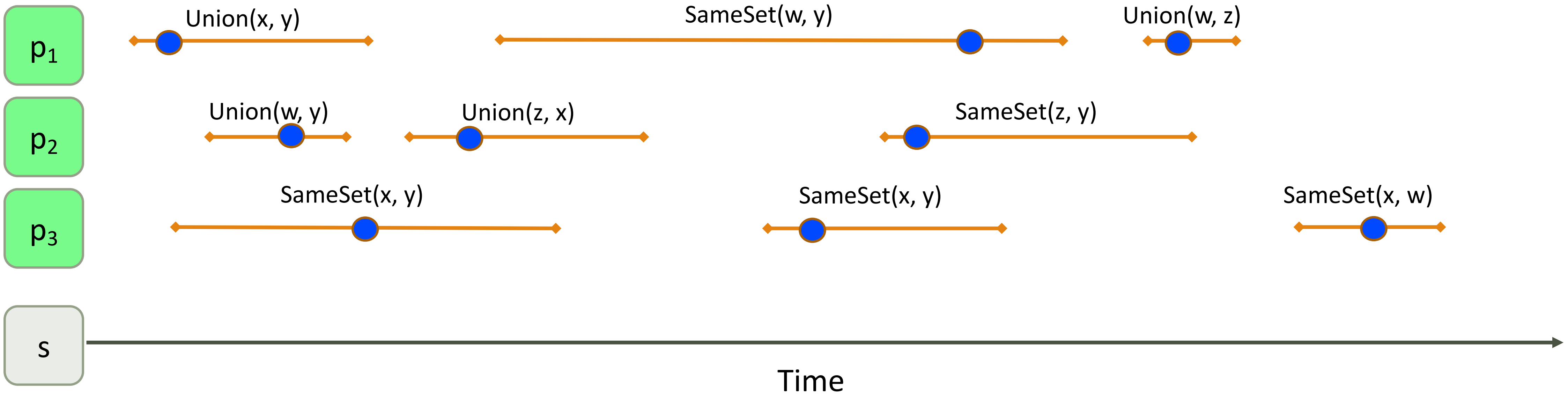
The same efficiency results carry over in Expectation!

Linking by Rank



Correctness Criteria

Linearizability [Herlihy, Wing 1990]:



Wait-Freedom: Each p_i should be able to complete its operation in a bounded number of its own steps.

Goal

Algorithm with work sub-linear in p.

Approach

- Use linking by ID instead
- Only parent pointers change in this case