Algorithms that Adapt to Contention

Hagit Attiya and Arie Fouren

Fast Mutex Algorithm

[Lamport, 1986]

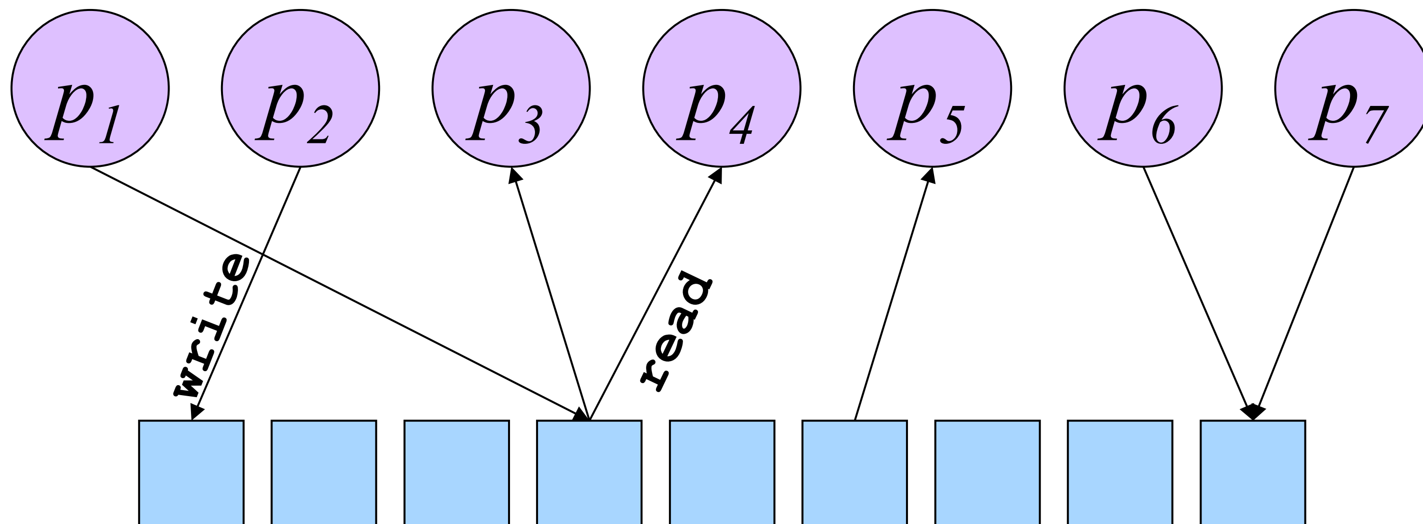
- In a well-designed system, most of the time only a single process tries to get into the critical section…

☝ Will be able to do so in a constant number of steps.

When two processes try to get into the critical section?

May require $O(n)$ steps!

Asynchronous Shared-Memory Systems



Need to collect information in order to coordinate…

When only few processes participate,
reading one by one is prohibitive …

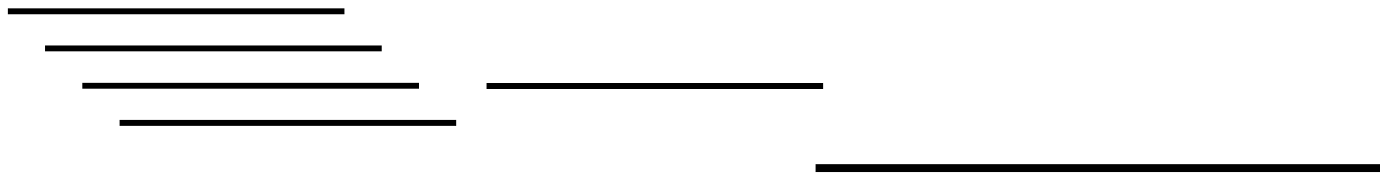
Outline

- How to be adaptive in a global sense?
  - The **splitter** and its applications: renaming and collect.
- How to adapt dynamically?
  - Long-lived **safe agreement** and its applications: **sieve**, renaming and collect.
- Extensions and connections.

Adaptive Step Complexity

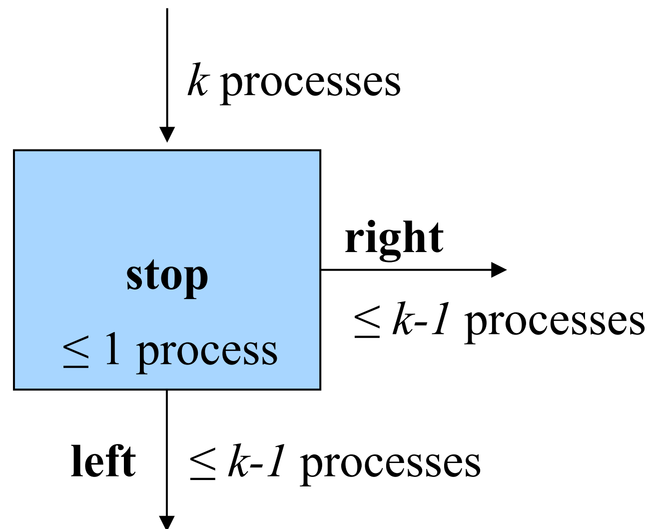
- The step complexity of the algorithm depends only on the number of active processes.

Total contention: The number of processes that (ever) take a step during the execution.



A Splitter

[Moir & Anderson, 1995]



A process stops if it is alone in the splitter.

Splitter Implementation

[Moir & Anderson, 1995][Lamport, 1986]

```
1. X = id_i                        // write your identifier
2. if Y then return( right )
3. Y = true
4. if ( X == id_i )   // check identifier
      then return( stop )
5.else return( left )
```

Splitter Implementation

[Moir & Anderson, 1995][Lamport, 1986]

```
1. X = id_i                  // write your
  identifier
2. if Y then return( right )
3. Y = true
4. if ( X == id_i )   // check identifier
     then return( stop )
5.else return( left )
```

Splitter Implementation

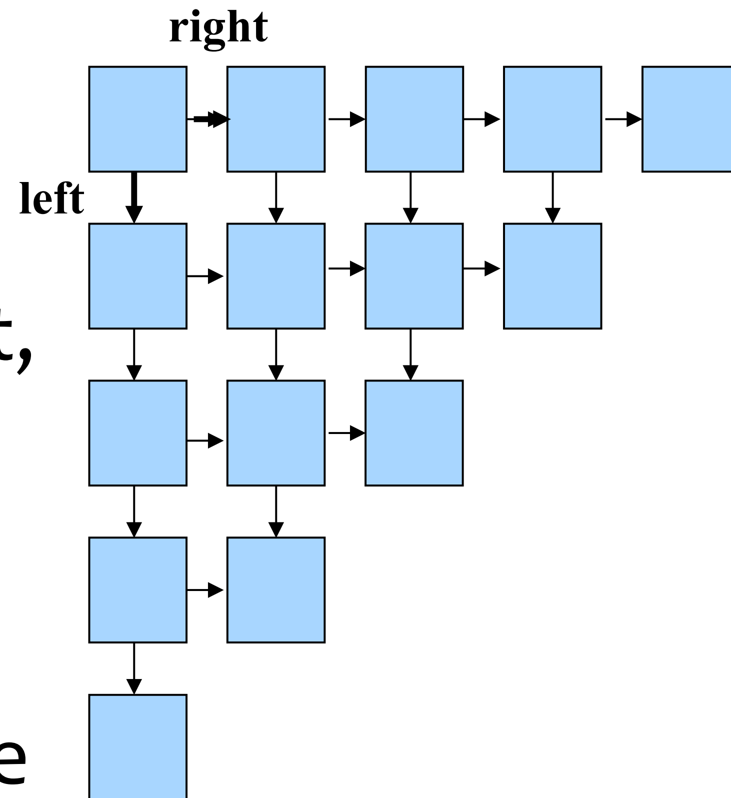
[Moir & Anderson, 1995][Lamport, 1986]

```
1. X = id_i                         // write your
  identifier
2. if Y then return( right )
3. Y = true
4. if ( X == id_i )   // check identifier
     then return( stop )
5.else return( left )
```

Requires ≤ 5 read / write operations, and two shared registers.

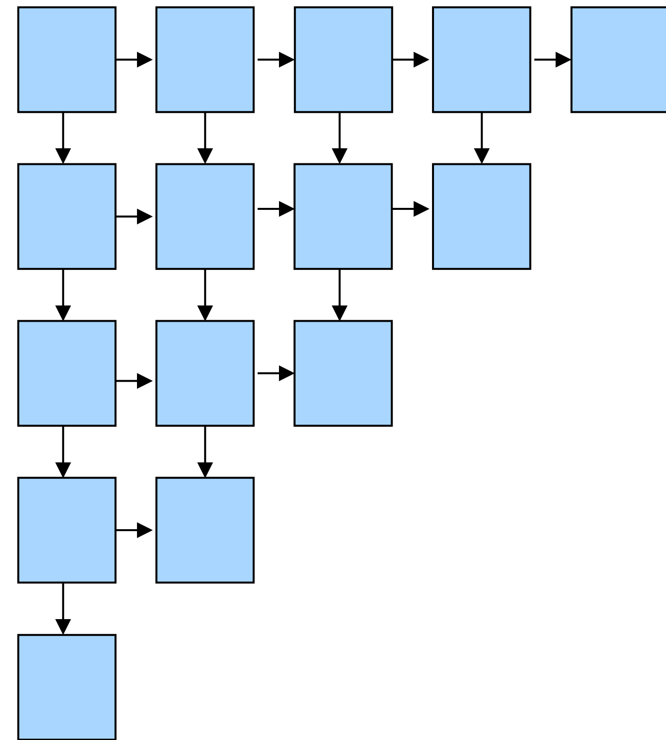
Putting Splitters Together

- A triangular matrix of splitters.
- Traverse array, starting at the top left, according to the values returned by splitters
- Until stopping in some splitter.



Putting Splitters Together

- ≥ one process does not go in each direction.
- ⇨ After ≤ k movements, a process is alone in a splitter.
- ⇨ A process stops at row, column ≤ k
- ⇨ At most $O(k)$ steps.



Renaming

- A process has to **`acquire`** a unique new name
  It may later **`release`** it
- The range of new names must be as small as possible
  - Preferably adaptive: depending only on the number of active processes
  - Must be at least $2k$-1

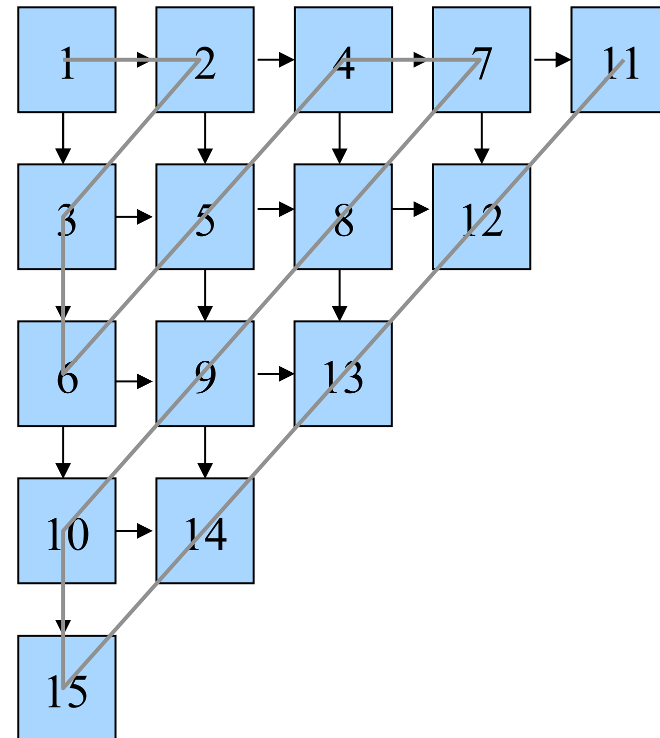
Renaming is a building block for adaptive algorithms

- First obtain names in an adaptive range
- Then apply an ordinary algorithm using these names

Putting Splitters Together: k^2 -Renaming

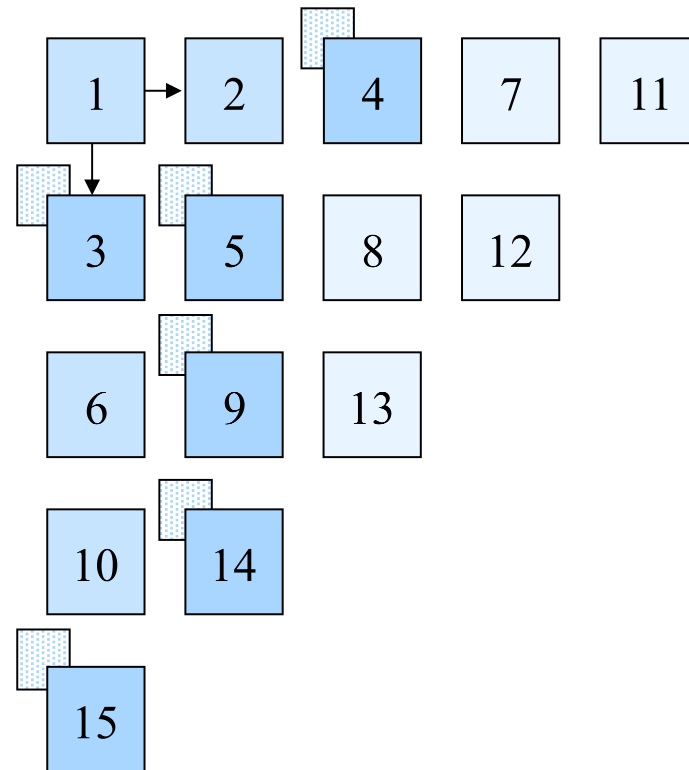
Diagonal association of names with splitters.

⇨ Take a name ≤ k^2 .



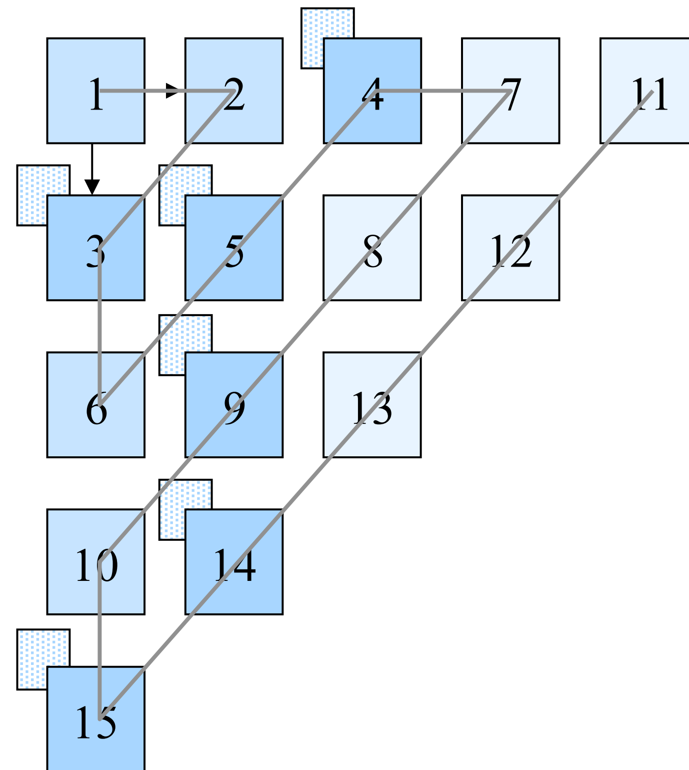
Better Things with a Splitter: Store

- Associate a register with each splitter.
- A process writes its value in the splitter where it stops.
- *Mark* a splitter if accessed by some process.



Better Things with a Splitter: Collect

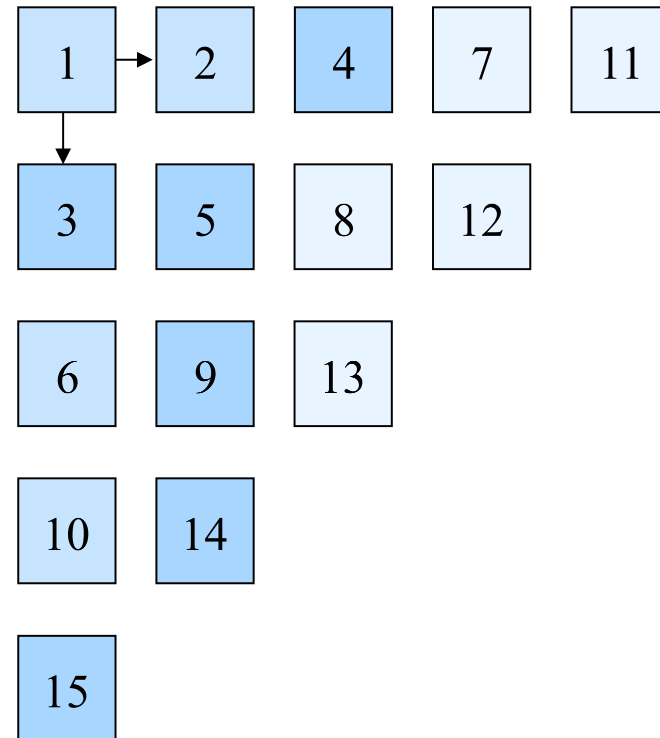
- Associate a register with each splitter.
- The current values can be collected from the associated registers.
- Going in diagonals, until reaching an unmarked diagonal.



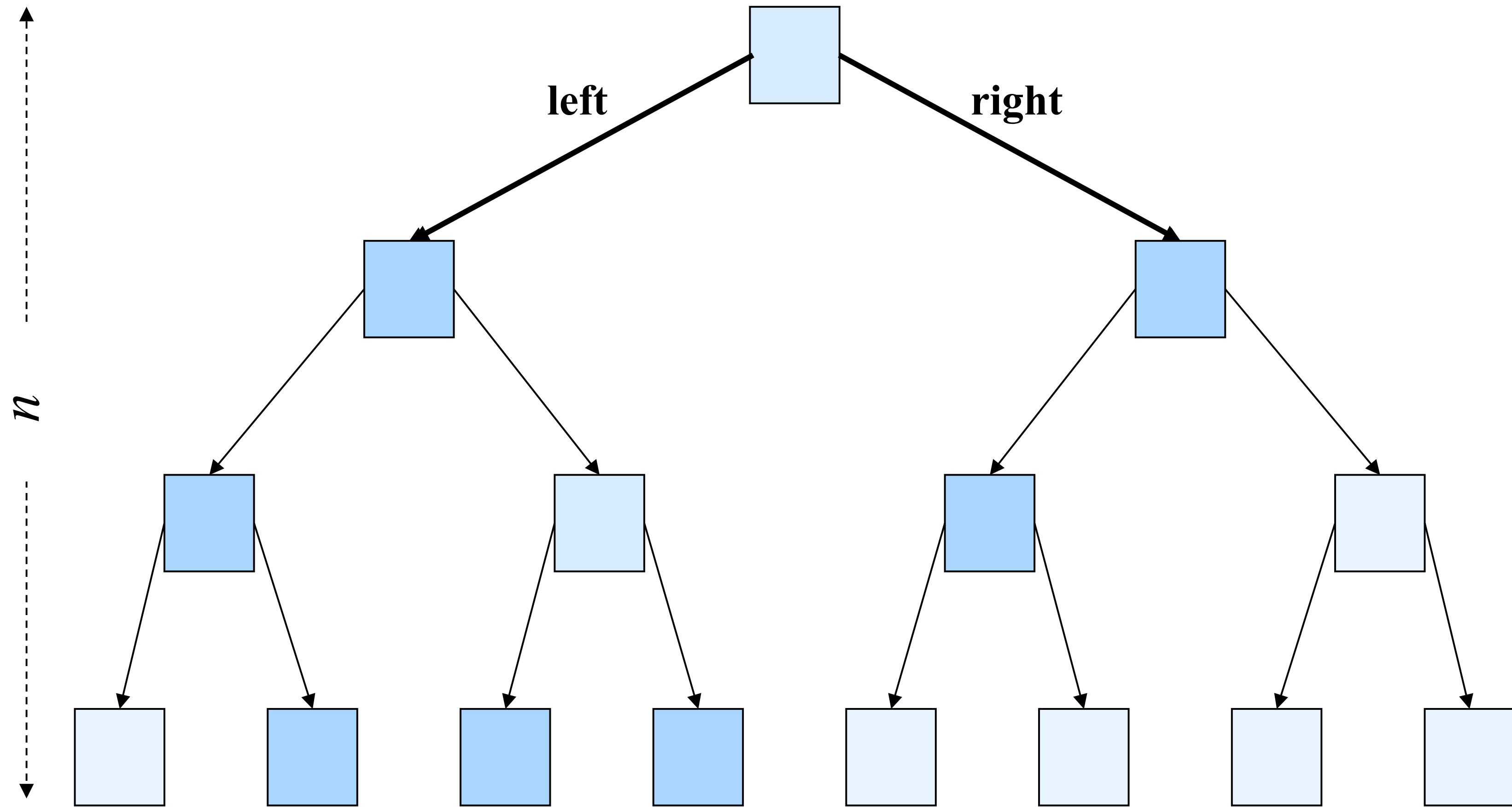
Even Better Things with a Splitter: Store and Collect

- The first store accesses ≤ k splitters.
- A collect may need to access k^2 splitters…

Can we do better?



Binary Collect Tree

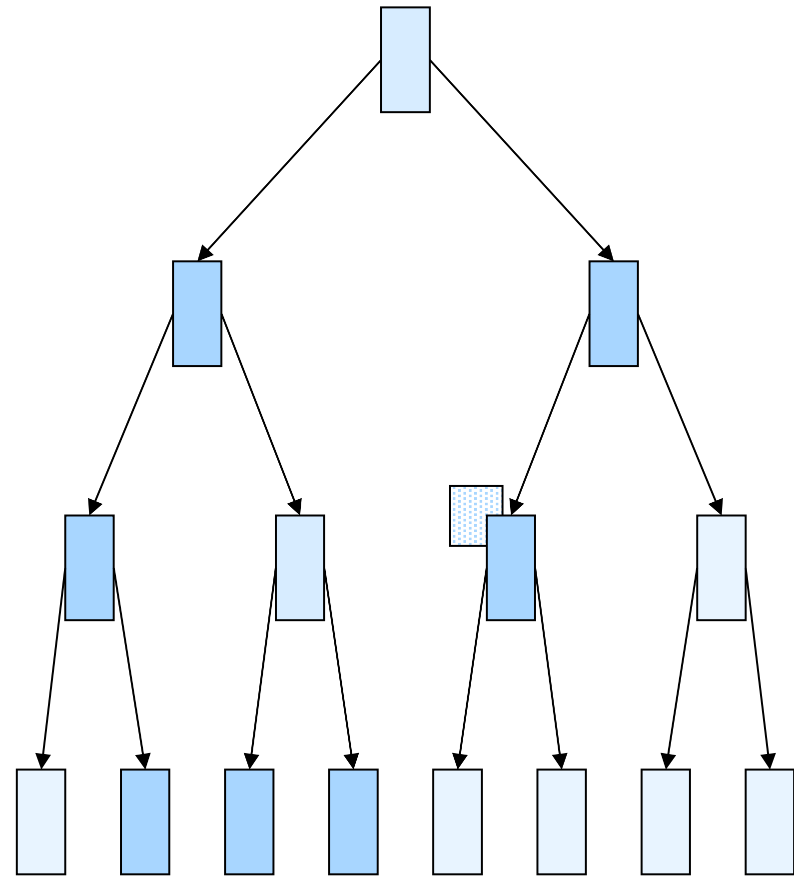


Binary Collect Tree

- To store:

  traverse the tree until stopping in some splitter.

- Later, write in the register associated with this splitter.

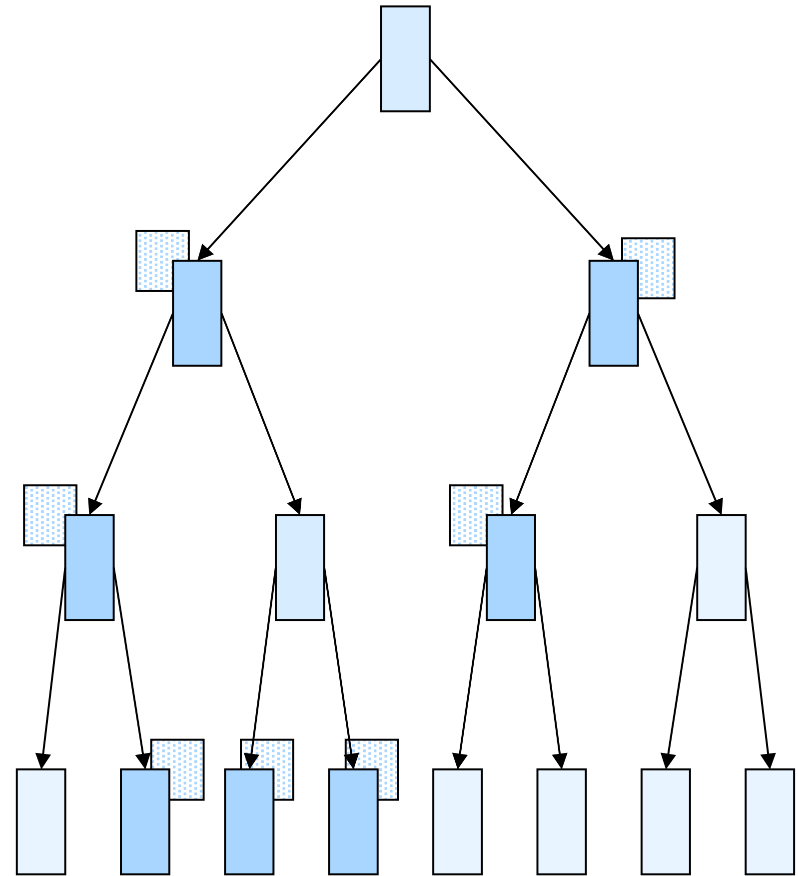


Binary Collect Tree

- To collect:

  DFS traverse the marked tree, and read the associated registers.

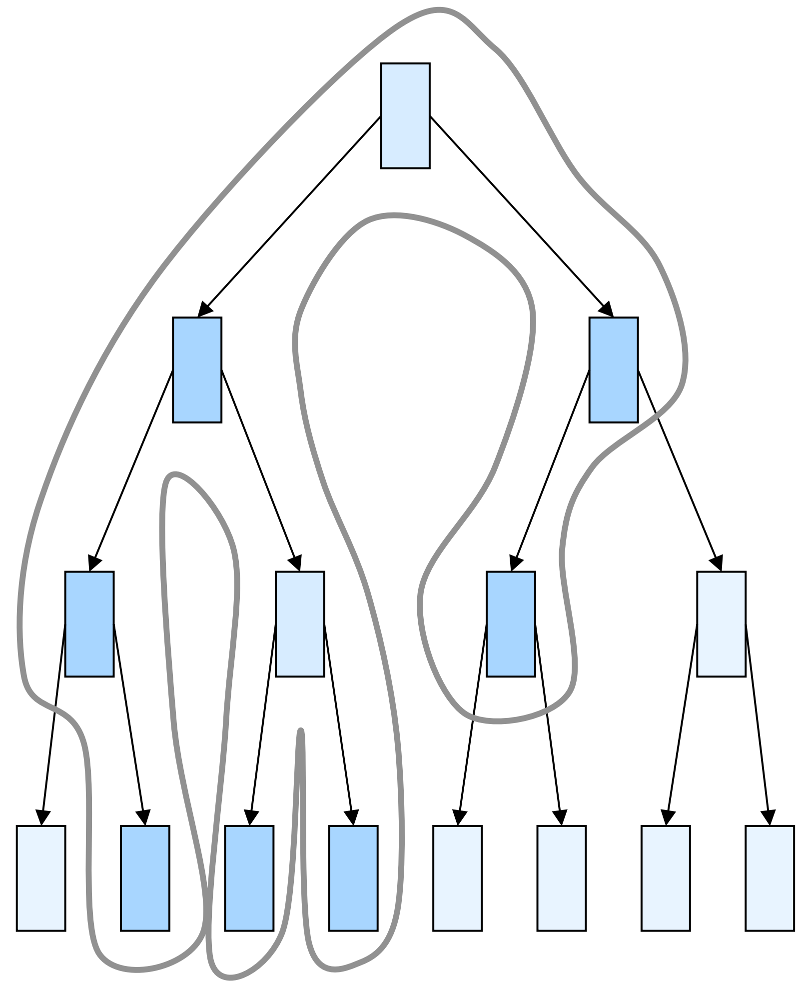
- Marked tree contains ≤ $2k$-1 splitters.



Size of Marked Sub-Tree

In a DFS ordering of the marked sub-tree,

There is an acquired node (where a process stops), between every pair of marked nodes.



Simple Things to Do with a Linear Collect

- Every algorithm with $f(k)$ iterations of collect and store operations can be made adaptive.
  - Atomic snapshots

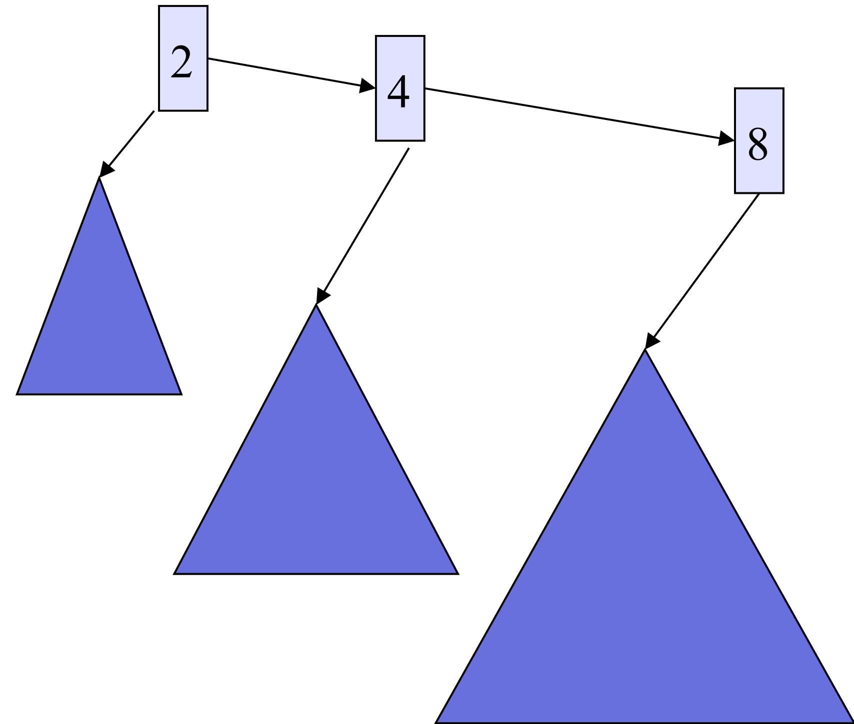
    [Afek et al. 1991]

    $O(k)$ iterations.

    ⇨$O(k^2)$ steps.
  - Renaming.
  - …

More Sophisticated Things to Do with a Linear Collect

- At each *spine* node:
  - Collect.
  - If # processes ≤ label
    - continue **left**
  - Else
    - continue **right**
      remember values.

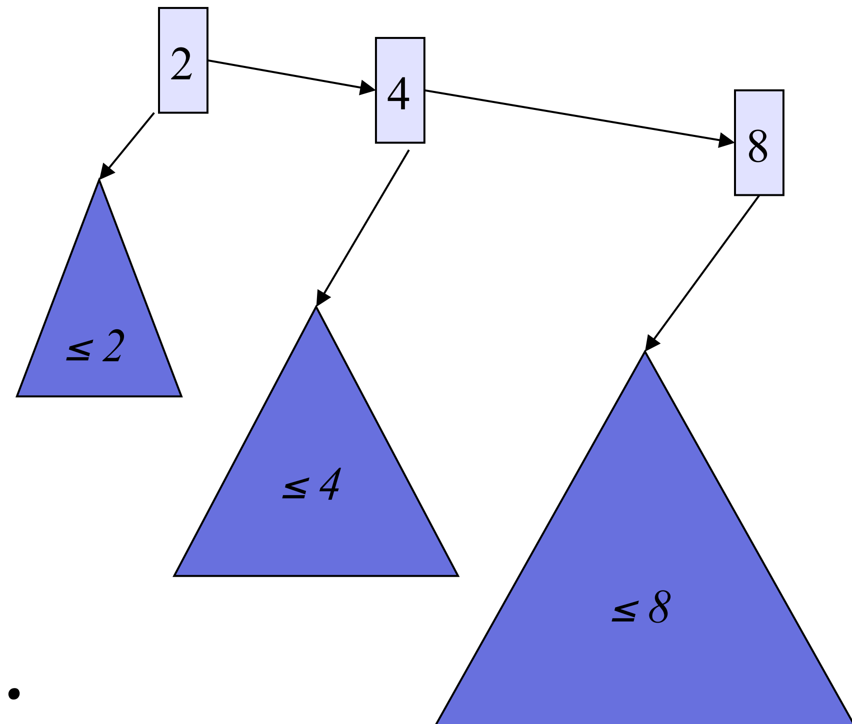


More Sophisticated Things to Do with a Linear Collect

- At most 2, 4, 8, etc. processes move to the left sub tree.

⇨ # participants in a sub-tree is bounded.

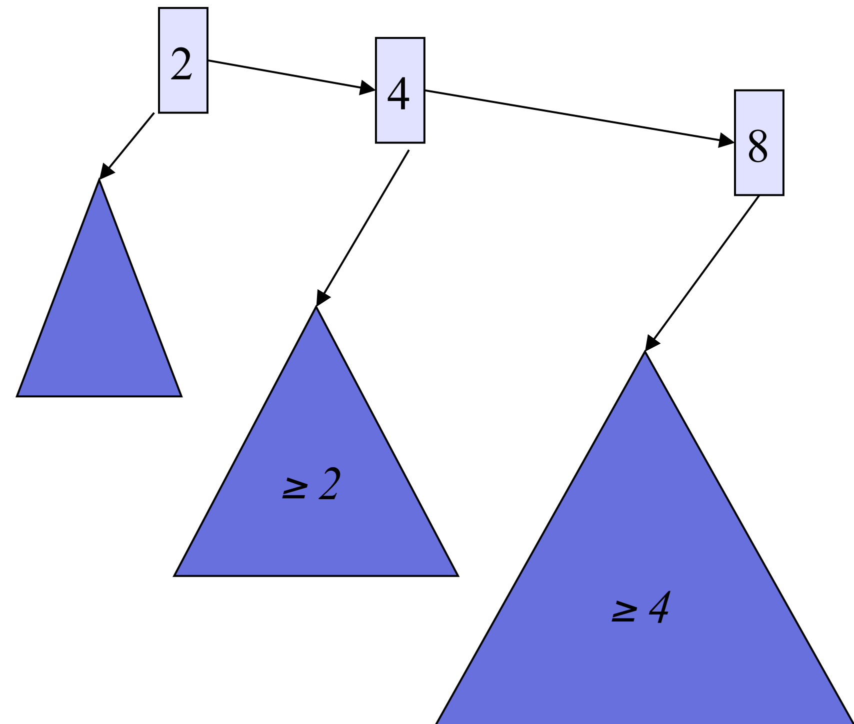
- Perform an ordinary algorithm in sub-tree.



More Sophisticated Things to Do with a Linear Collect

- If move right, at least 2, 4, 8,… participants.

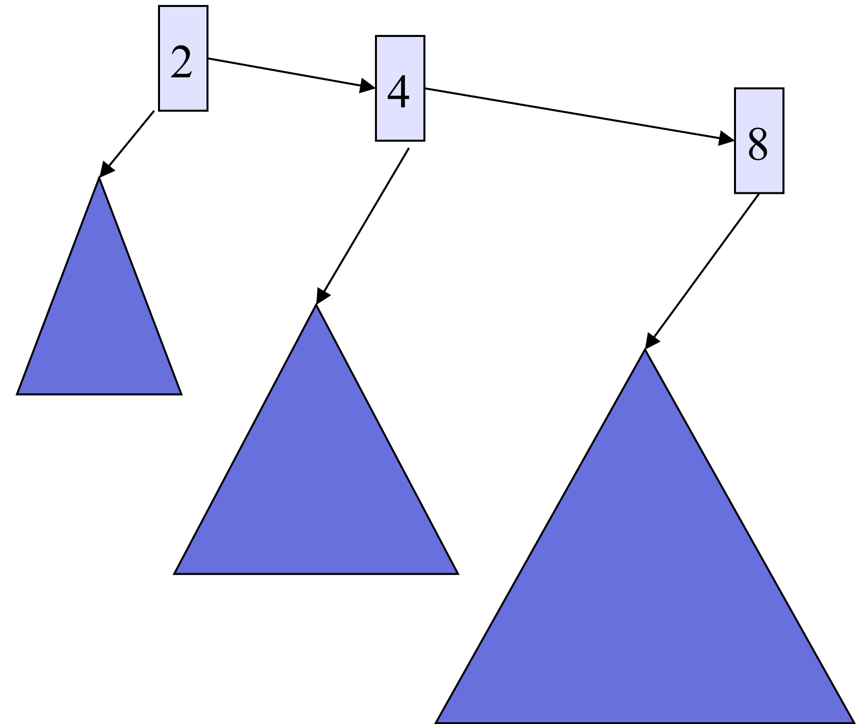
⇨ The extra step complexity is justified.



More Sophisticated Things to Do: Efficient Atomic Snapshot

- E.g., atomic snapshot algorithm.
  [Attiya & Rachman, 1998]

⇨ An $O(k \log k)$ atomic snapshot algorithm.

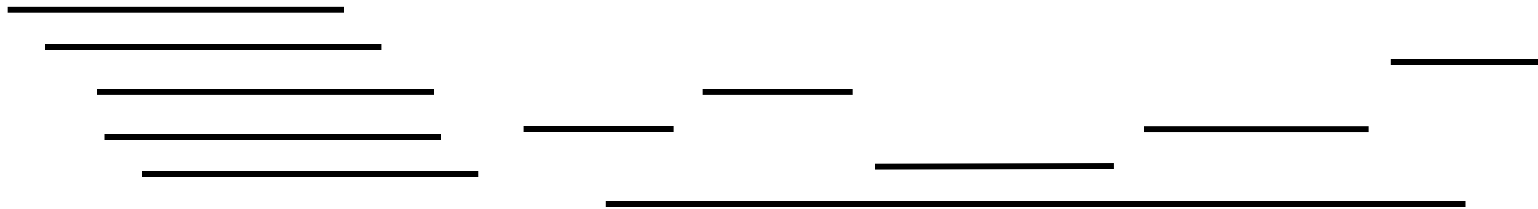


Be More Adaptive?

- In a *long-lived* algorithm…
  …processes come and go.

- What if many processes start the execution, then stop participating?
  …then starts again…
  …then stops again…

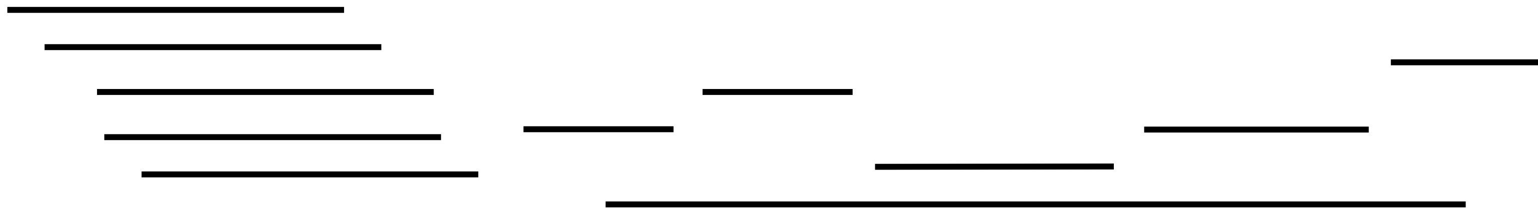
Who's Active Now?



Interval contention during an operation:
The number of processes (ever) taking a step during the operation.

[Afek, Stupp &Touitou, 1999]

Who's Active Now?



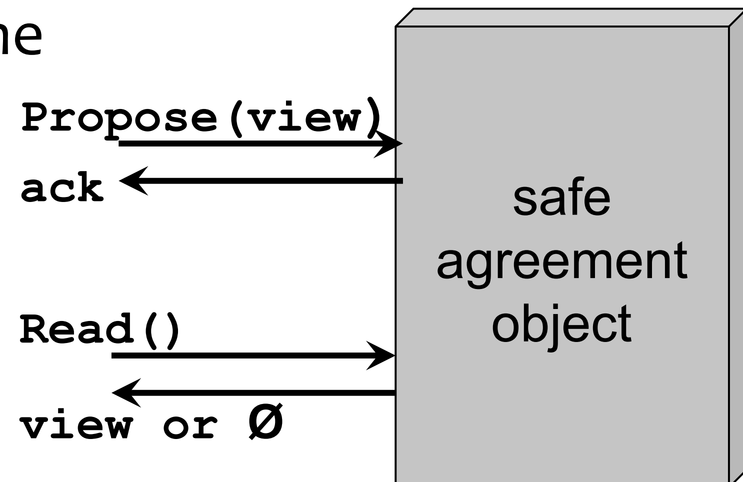
Point contention of an operation:

Max number of processes taking steps together during the operation.

Clearly, point contention ≤ interval contention.

Safe Agreement: Specification

Separate the voting / negotiation on a decision from figuring out the outcome



Two wait-free procedures: **`Propose`** and **`Read`**

Validity of non-Ø views

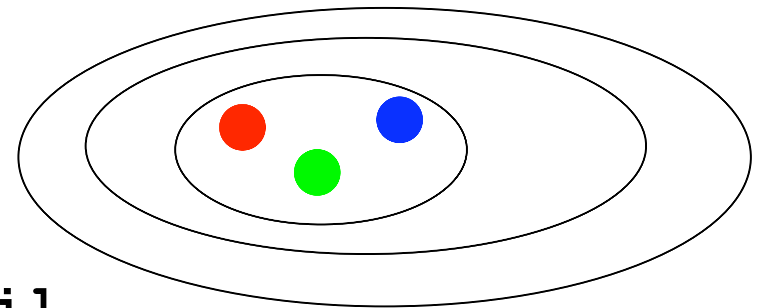
Agreement on non-Ø views returned by **`Read`**

Termination:
If all processes that invoked **`Propose`** return, then **`Read`** returns non-Ø view

Safe Agreement: Implementation

[Borowsky & Gafni]

Use an atomic snapshot object and an array **R**

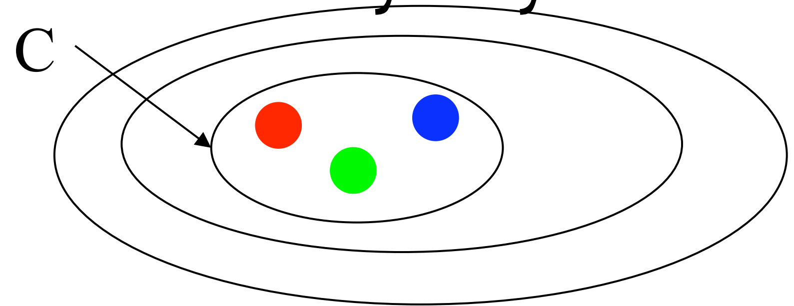


```
Propose( info )
   update( info )
   scan
   write returned view to R[i]
Read() returns view
   find minimal view C written in R
   if all processes in C wrote their view
      return C
   else return Ø
```

U U U S S U U S

Safe Agreement: Safety

Let C be the minimal view returned by any scan



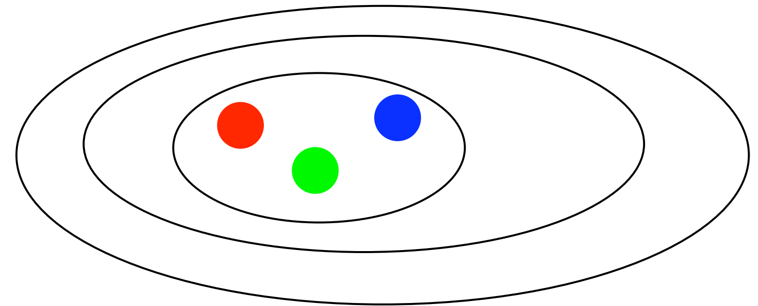
Can prove that all non-∅ views are equal to C

U U U S S U U S

Safe Agreement: Liveness

Clearly, both procedures are wait-free

- But **Read** may return a meaningless value, **Ø**

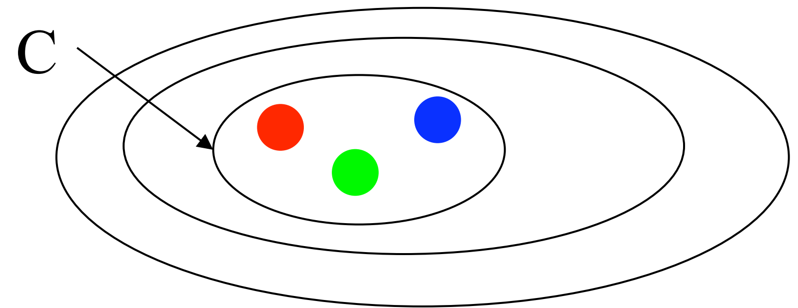


If some process invokes **Propose**, then after all processes that invoke **Propose** return, a **Read** returns a non-Ø value

U U U S S U U S

Safe Agreement: Winners

Even better…



A **Read** by some process in C
returns a non-Ø value

E.g., the last process in C to write its view

These processes are called winners

U U U S S U U S

Safe Agreement & BG Simulation

Safe agreement was introduced by Borowsky & Gafni for fault-tolerant simulation of wait-free algorithms

- Abstracted by Lynch & Rajsbaum

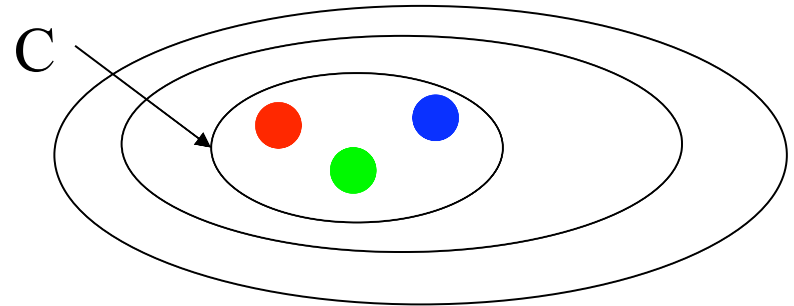
- Different interface
  - `Propose` and `Read` not separated
  - No Ø response for `read`
  - Complicates the simulation

☝ They also missed an interesting feature…

[Attiya & Fouren]

Safe Agreement: Concurrency

All processes in C execute
  `Propose` concurrently
  In particular, all winners

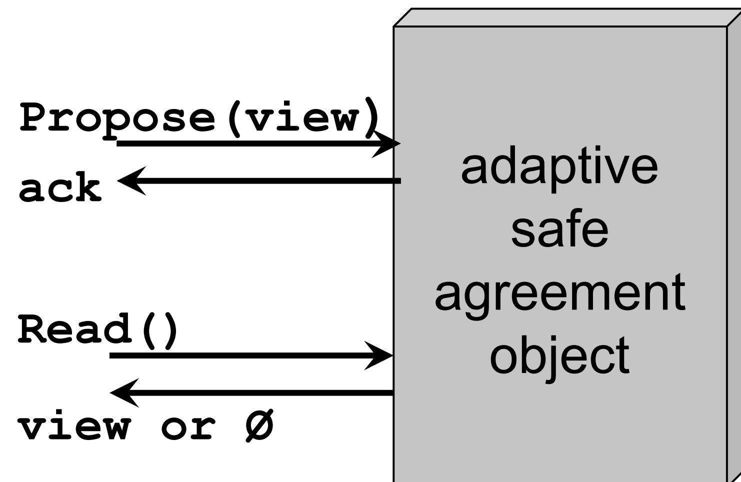


U U U S S U U S

Safe Agreement: Concurrency

All processes in C execute **`Propose`** concurrently

In particular, all winners

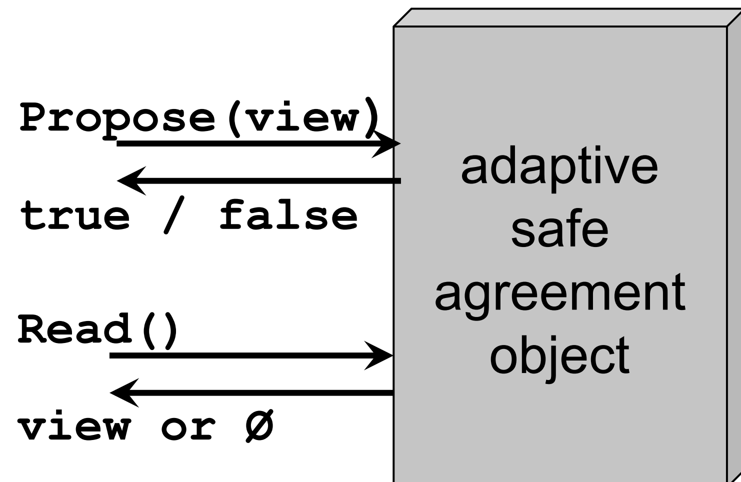


Use a doorway variable **`inside`** to avoid unnecessary update / scan

Safe Agreement: Concurrency

All processes in C execute **`Propose`** concurrently

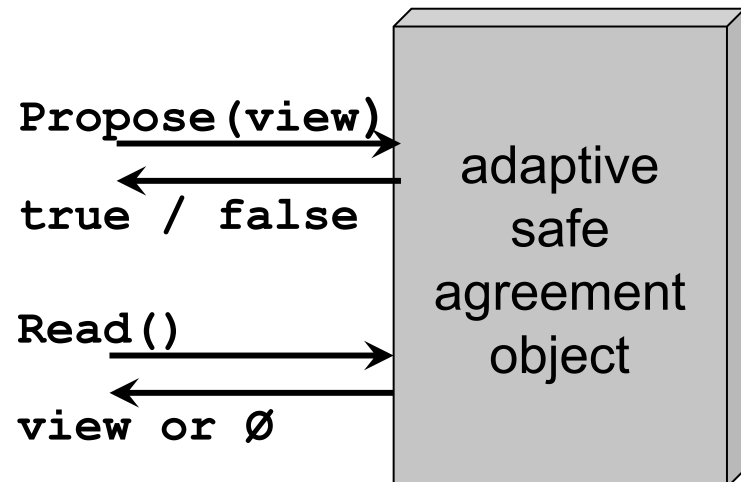
In particular, all winners



Use a doorway variable **`inside`** to avoid unnecessary update / scan

Adaptive Safe Agreement

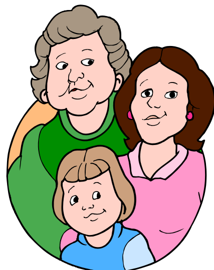
```
Propose( info )
if not inside then
   inside = true
   update( info )
   scan
   write returned view
   return( true )
else return( false )
```

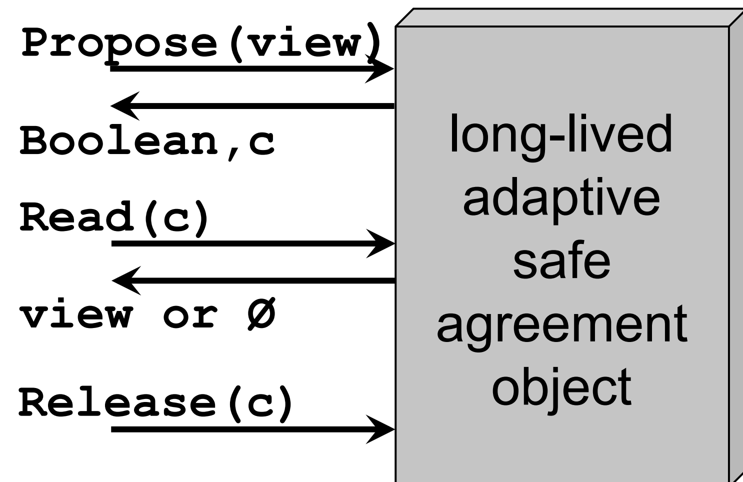


Concurrency: If a process returns **`false`** then some "concurrent" process is accessing the object

Long-Lived Adaptive Safe Agreement

Enhance the interface with
a generation number
(nondecreasing counter)





Validity, agreement and termination as before but relative to a single generation

Concurrency: If a process returns **`false,c`** then some process is concurrently in generation **`c`** of the object

Long-Lived Adaptive Safe Agreement

Synchronization: processes are inside the same generation simultaneously

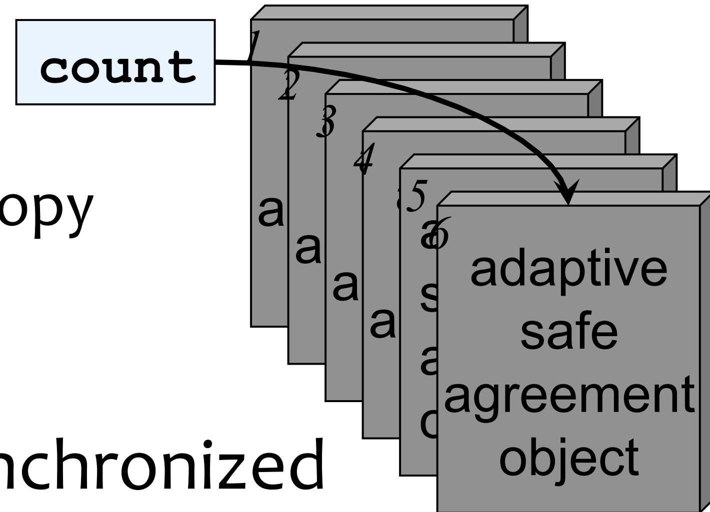
⇨ Their number ≤ point contention

⇨ Can employ algorithms adaptive to total contention within each generation

- e.g., atomic snapshots

Long-Lived Adaptive Safe Agreement: Implementation

- Many copies of one-shot safe agreement
  - **`count`** points to the current copy



- Winners of each copy are synchronized
  - Increase **`count`** by 1.
  - Monotone…

When all processes release a generation, open the next generation by enabling the next copy

Catching Processes with Safe Agreement

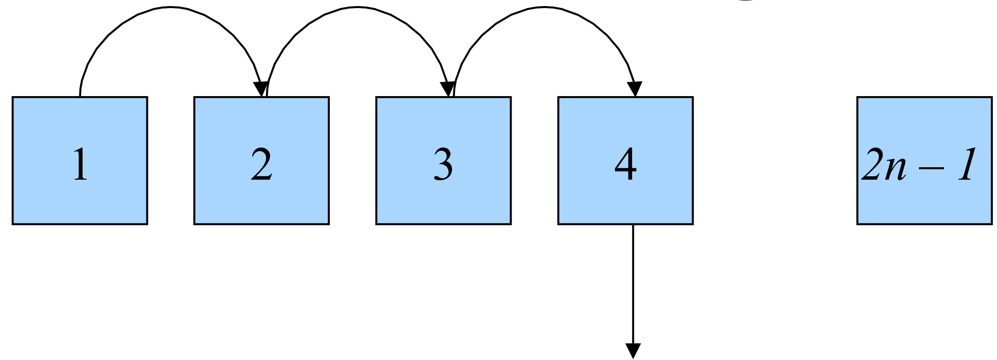
- When processes access an adaptive long-lived safe agreement object simultaneously, at least one wins

- If a process accesses an adaptive long-lived safe agreement object and does not win, some other process is accessing the object

Good for adaptivity…

Things to do with Long-Lived Safe Agreement: Renaming

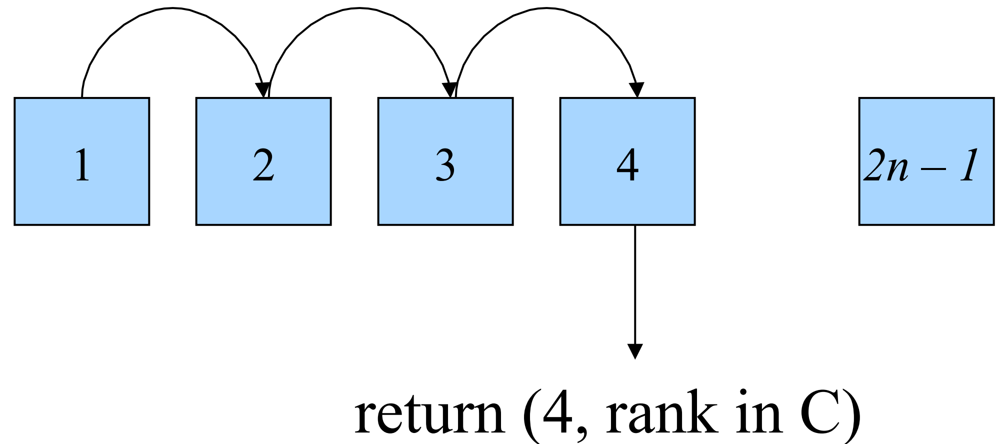
Place objects in a row…



Agreement in each long-lived safe agreement object

⇨ **Uniqueness** of names.

Renaming: Size of Name Space



Concurrency for each long-lived safe agreement object

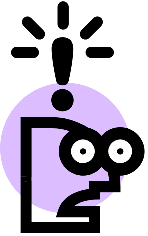
⇨ An object is skipped only due to a concurrent process

⇨ A process skips ≤ r objects

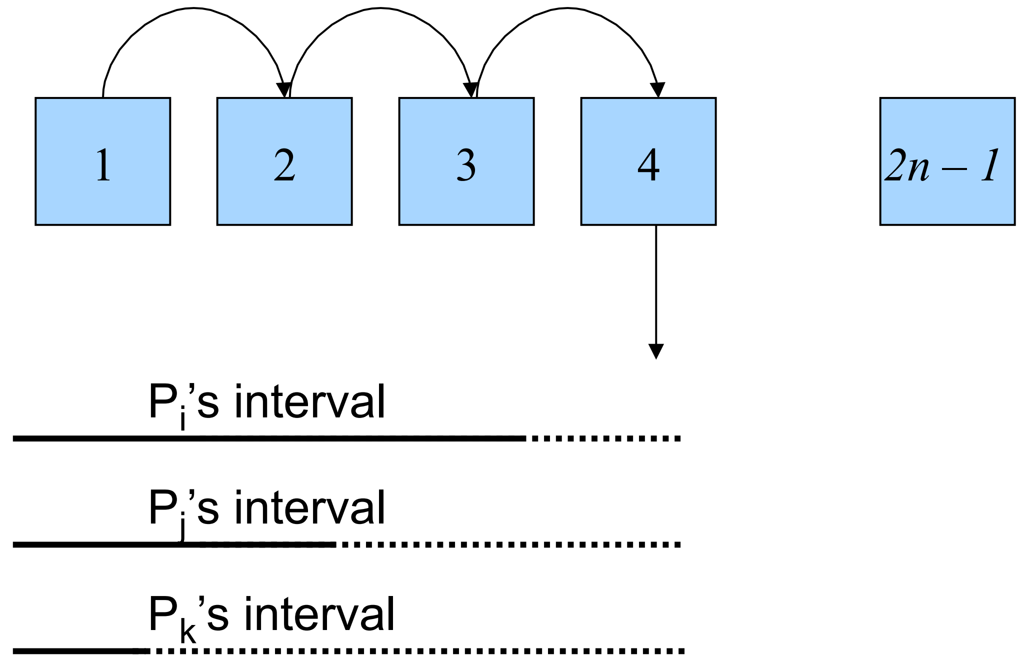
- r is the interval contention

- Range of names ≈ r^2

Renaming: Size of Name Space



We promised point contention



P_i skips because of P_j

⇨ P_j skips because of P_k

⇨ P_k skips because of …

They all overlap

Renaming: Complexity & Size of Name Space

Proof is subtle since a process skips
either due to a concurrent winner
or due to a concurrent non-winner in C
(which it can meet again later in the row)

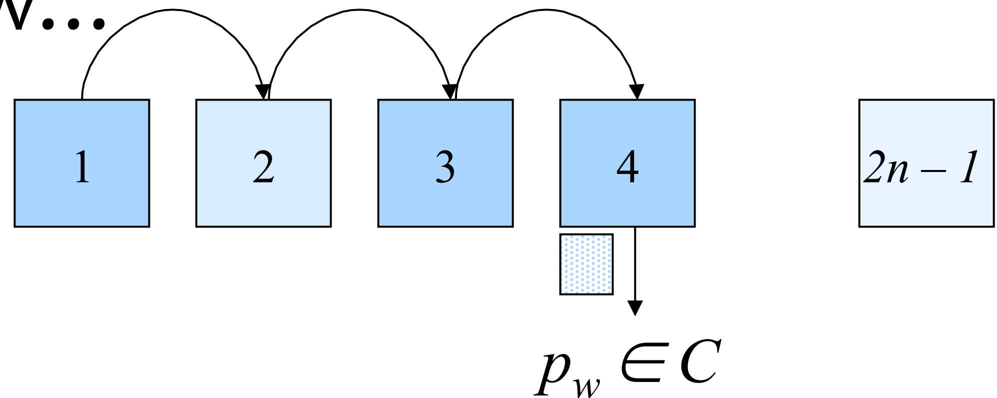
- Use a potential-function proof to show that a process skips ≤ $2k-1$ objects
  - k is the point contention

⇨ Name ≈ k^2

⇨ $f(k)$ step complexity

Store

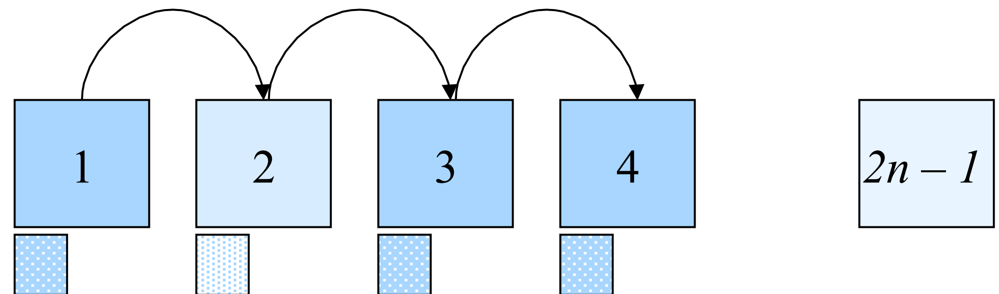
Place objects in a row…



Agreement on set of candidates and uniqueness of copies

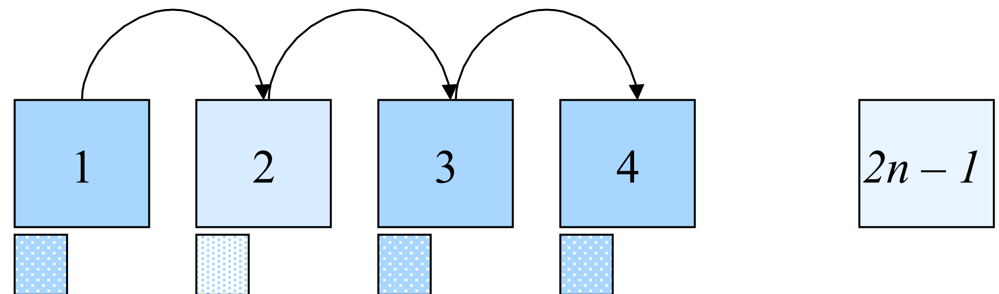
⇨ p_w writes the values of all candidates in a register associated with the sieve.

Collect



- Go over the associated registers and read…

Collect



- p_w and all other operations complete.
- A collect still has to reach the splitter in which p_w has written its value!

Bubble-Up

[Afek, Stupp & Touitou, 2000]

- Before completing an operation, move information from far away objects to the top.

Other Things We can Do

Long-lived adaptive safe agreement objects with bubble-up yield adaptive (to point contention) algorithms for:

- Gathering & collecting information
- Atomic snapshots
- Immediate snapshots
- $(2k\text{-}1)$-renaming (optimal)

Even More…

- The algorithms can be made fully adaptive
  - Step complexity depends on processes really participating, not just "signing in"
    - Especially relevant in renaming-based algorithms

- Can bound their memory requirements
  - But the bounds are not adaptive…

What About Mutex?

- Cannot have adaptive step complexity…
- Can have adaptive system response time.

[Attiya & Bortnikov, 2002]

  - Some techniques are similar.
  - Renaming, adaptive binary tree (bottom-up!)…

Space: The Final Frontier

- Improve the step complexity of the algorithms and reduce their space complexity
  - Lots of improvement recently for total contention
  - E.g., using randomization

- Algorithms whose space complexity is truly adaptive to point contention?
  - Currently, number of registers used depends on total contention
  - Allocated vs. used registers

Space: The New Frontier

- Our results are based on a collect algorithm.
  - Either $O(K^2)$ step complexity (K is total contention),
  - Or exponential space complexity.
- A better collect algorithm?
  - $O(K)$ step complexity, and
  - Polynomial space complexity.
- A lower bound proof?

Other Aspects

- Using stronger primitives (CAS…)
  - Promising for adaptive space complexity

[Afek, Dauber, Touitou] [Herlihy, Luchangco, Moir][Fatourou, Kallimanis]

- More modularity…
  - We made some progress with the long-lived adaptive safe agreement object
  - What about bubble-up?

Lower Bounds

Non-constant number of multi-writer registers is needed for adaptive weak test&set

[Afek, Boxer, Touitou]

⇨ Holds also for renaming and long-lived collect

Non-constant number of multi-writer registers is needed for adaptive generalized weak test&set

[Aguilera, Englert, Gafni]

⇨ Holds also for one-shot collect

Linear number of multi-writer registers is needed for adaptive and efficient one-shot collect

[Attiya, Fich, Kaplan]

Taking a Broader Perspective

Connections with recent research trends:

- Obstruction-free algorithms
  - Adapting to step contention

    [Attiya et al. DISC 2005], [Attiya et al. PODC 2006]

- Abortable / failing objects

- Population-oblivious algorithms

Mostly based on

- Attiya and Fouren,
  Adapting to Point Contention with a Sieve,
  Journal of the ACM, Vol. 50, No. 4 (2003).

- Attiya, Fouren and Gafni,
  An Adaptive Collect Algorithm with Applications,
  Distributed Computing, Vol. 15, No. 2 (2002).

THANK YOU!