The Teleportation Design Pattern

Nachshon Cohen
Maurice Herlihy
Erez Petrank
Elias Wald

Hardware Transaction Memory

- Simple to use: atomic{ … };
- Fast: hardware implementation

Available on Intel and IBM processors

- But: hardware transactions are *best effort*
  - Limited resources, thus spurious failures
  - Dependency on external events (e.g., interrupts)
  - Contention causes significant waste

How to use HTM efficiently for building concurrent data structures?

Agenda

- Black Box Approach and Limitations
- Teleport Design Pattern
- Applying teleport to lock coupling
- Applying teleport to hazard pointers
- Measurements

Black Box Approach

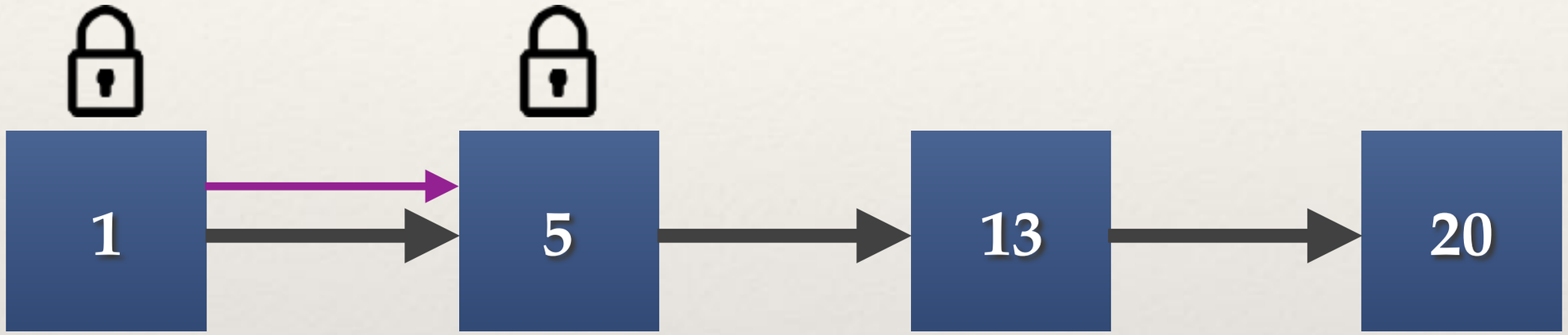
```
start:
   switch(start_htm())
   case SUCCESS:
        operation
     commit_htm();
   case FAILURE:
     goto start;
```

- May fail repeatedly

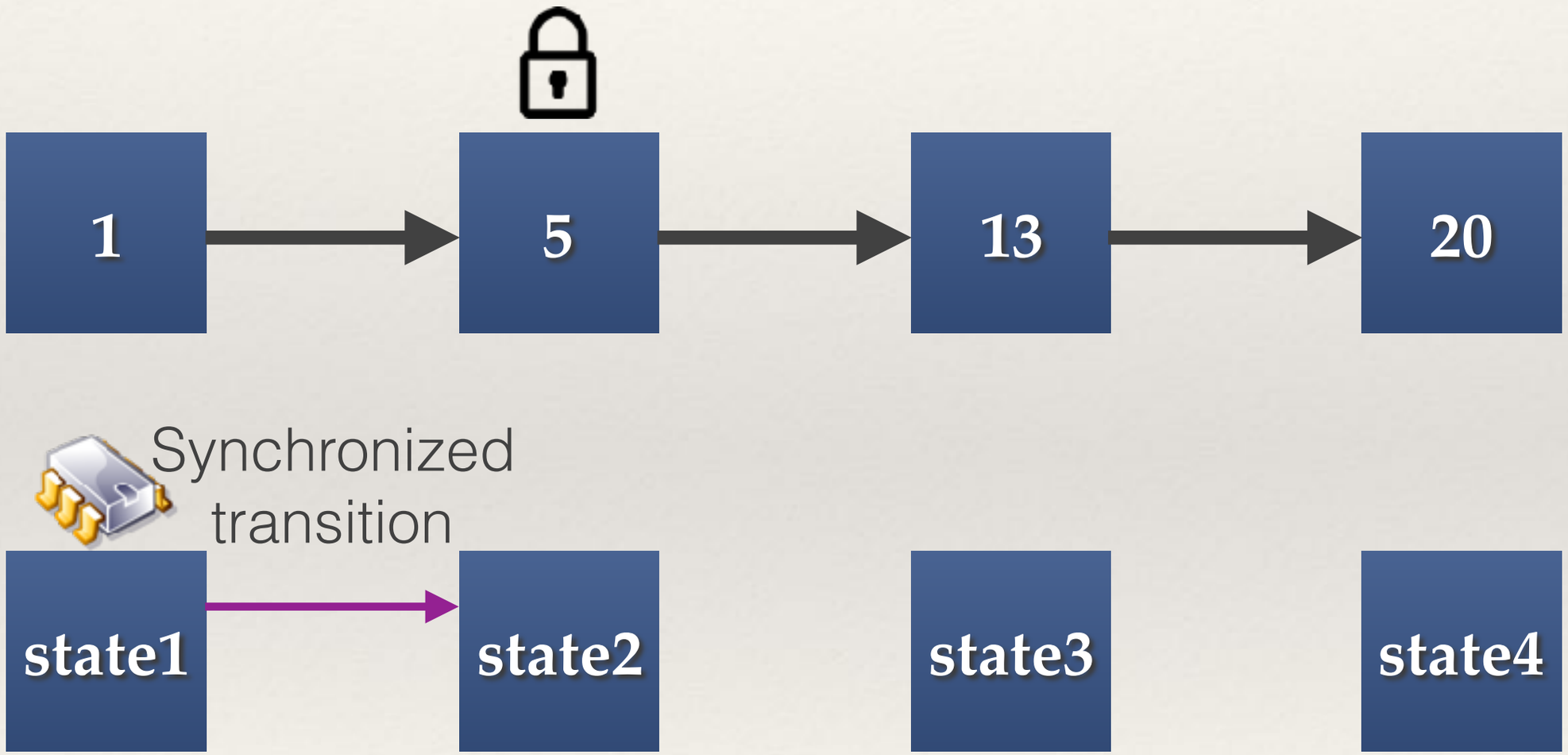
```
switch(start_htm())
 case SUCCESS:
      operation
   commit_htm();
 case FAILURE:
   globalLock.acquire()
   operation
```

- Lock elision
- May execute sequentially

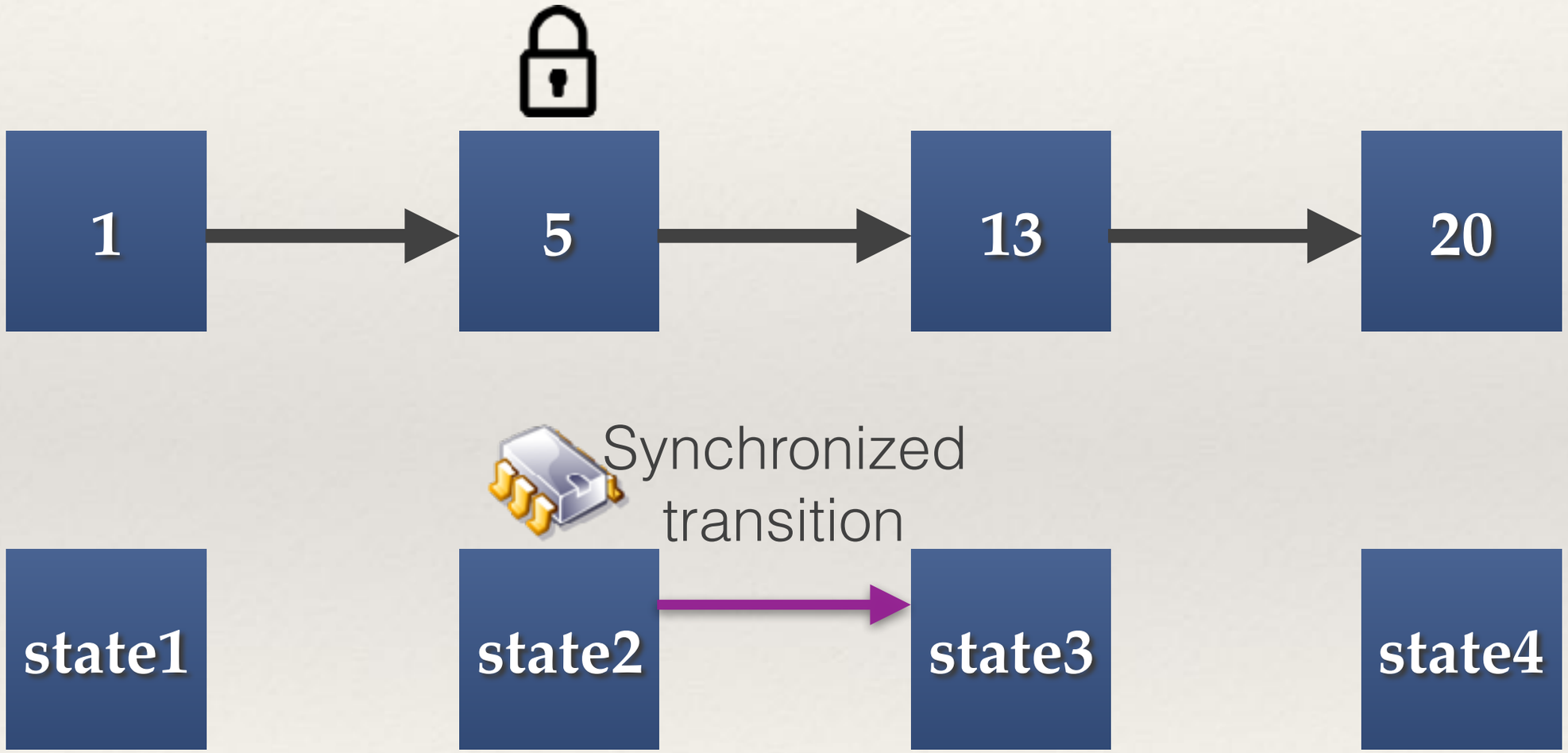
Example: Lock-Coupling Linked List



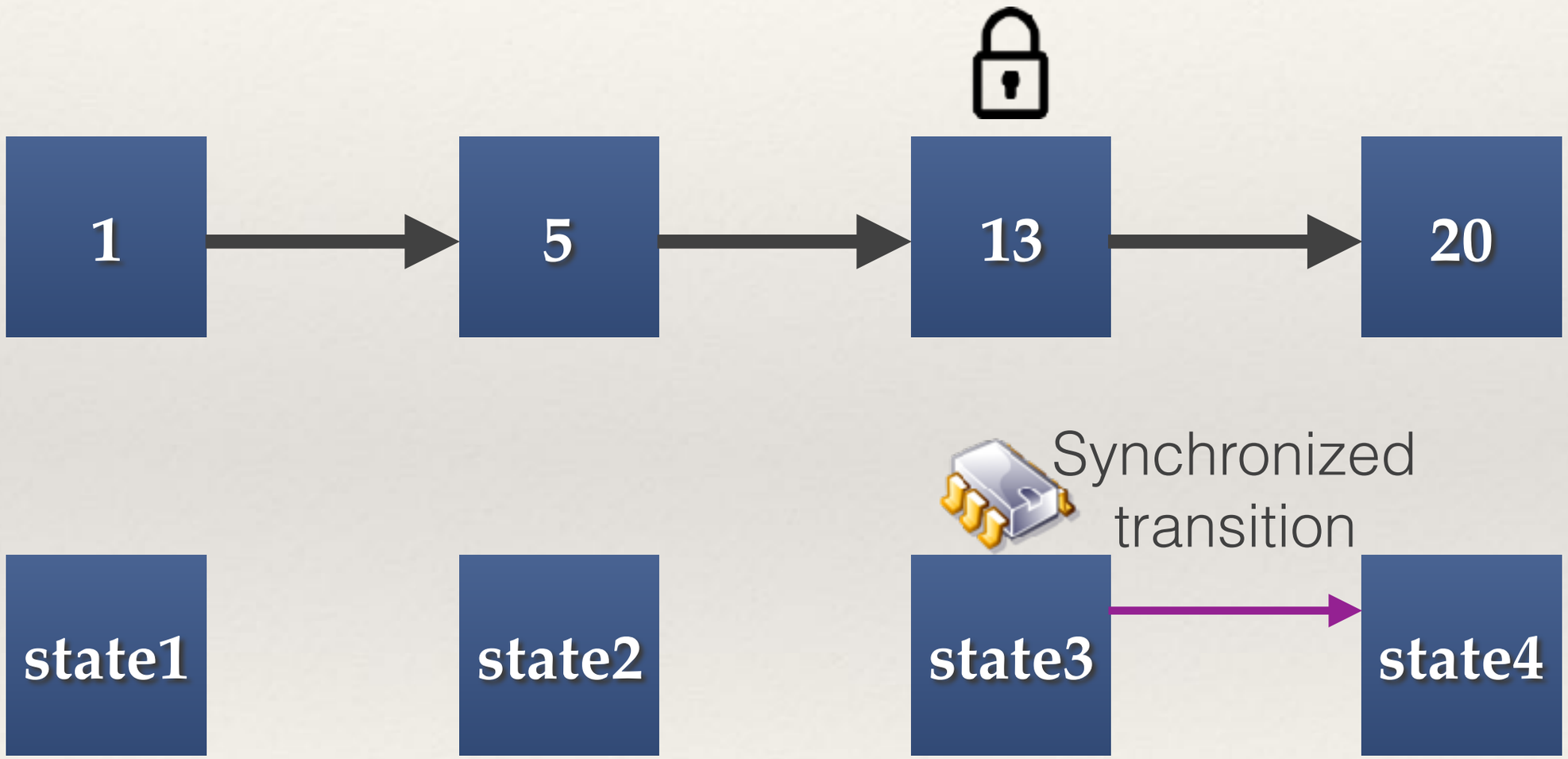
Abstract Concurrent Data Structure



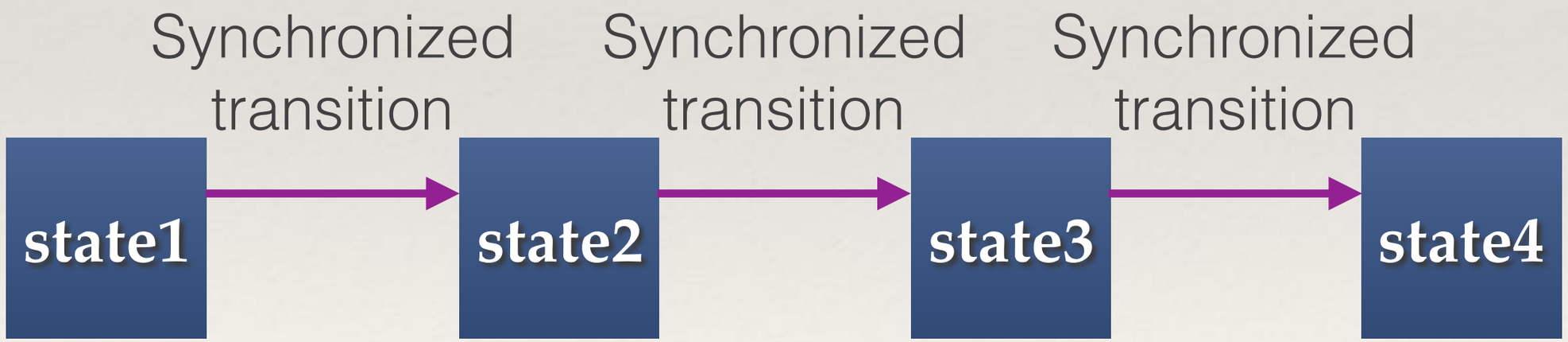
Abstract Concurrent Data Structure



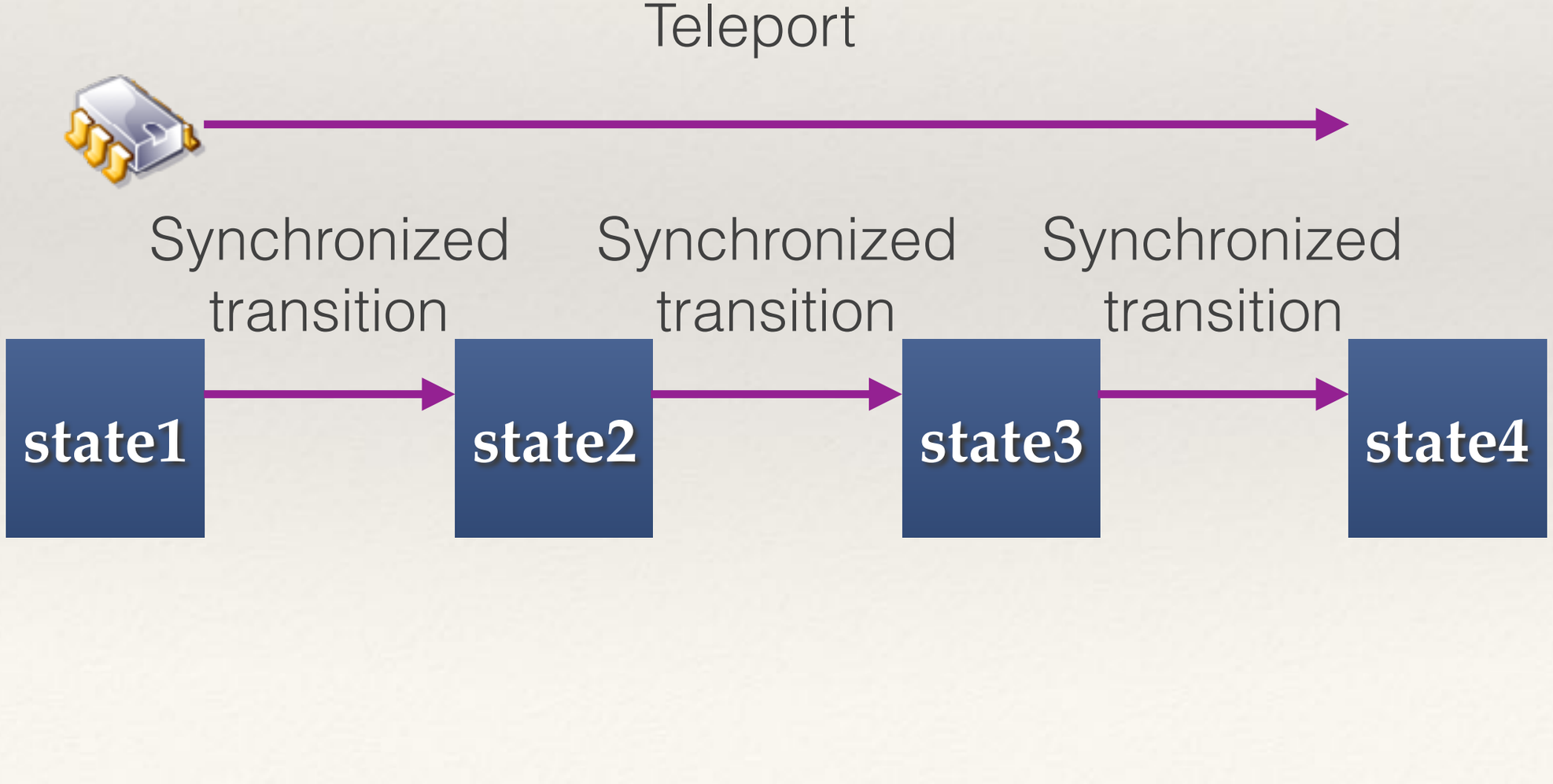
Abstract Concurrent Data Structure



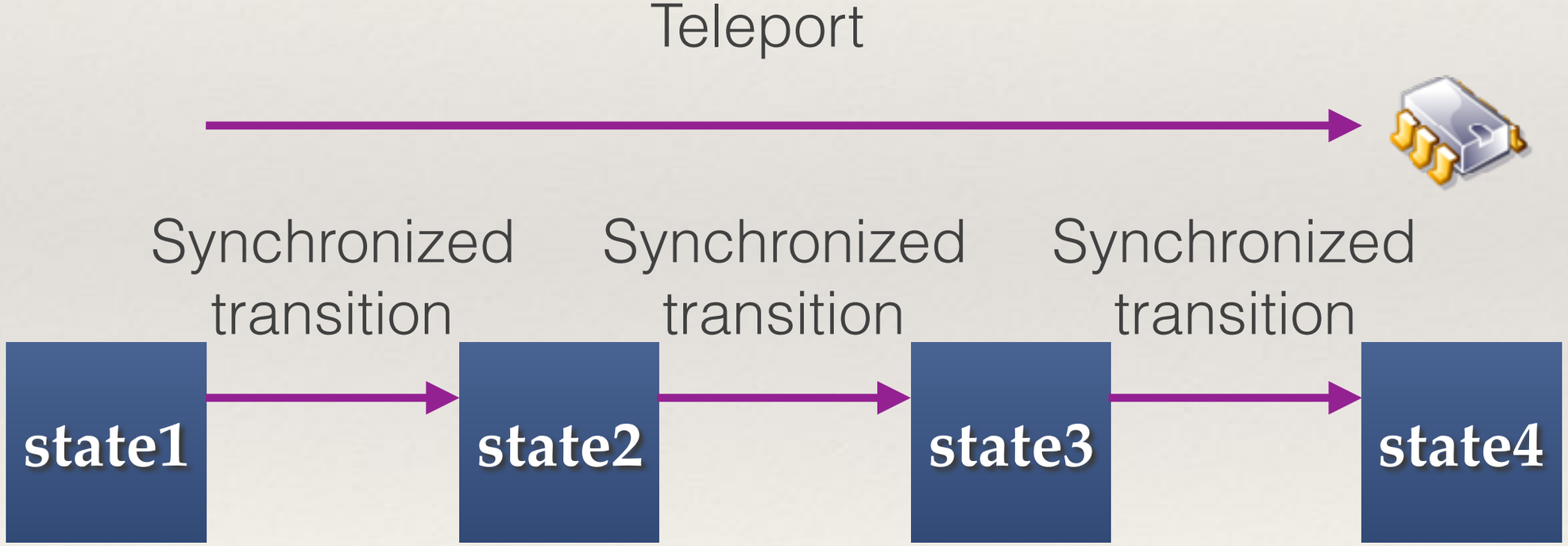
Abstract Concurrent Data Structure



Teleportation Design Pattern



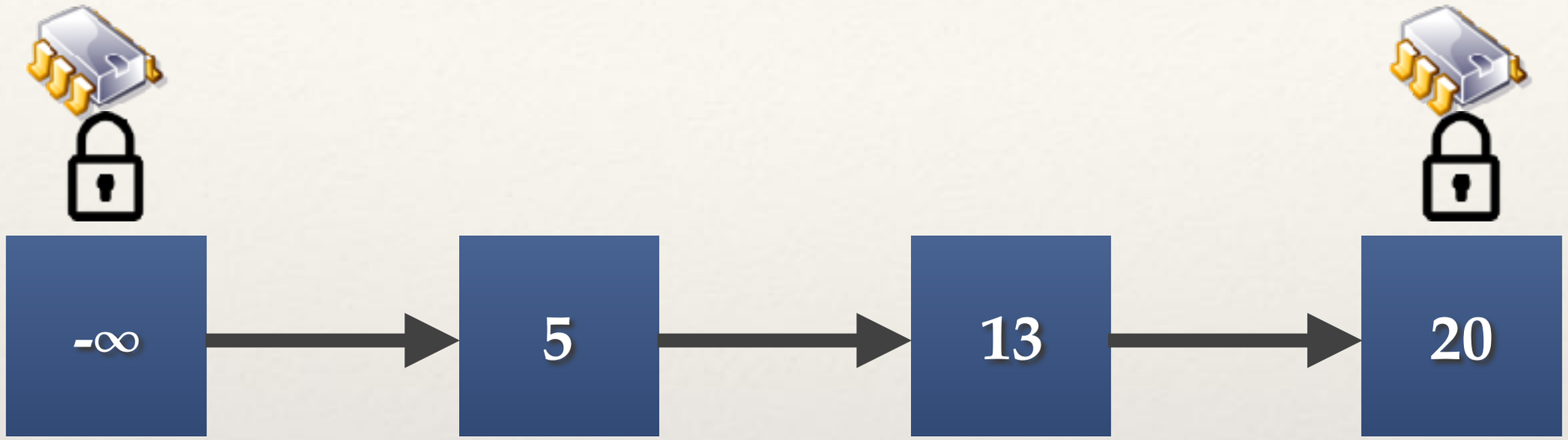
Teleportation Design Pattern



Adaptive Strategy

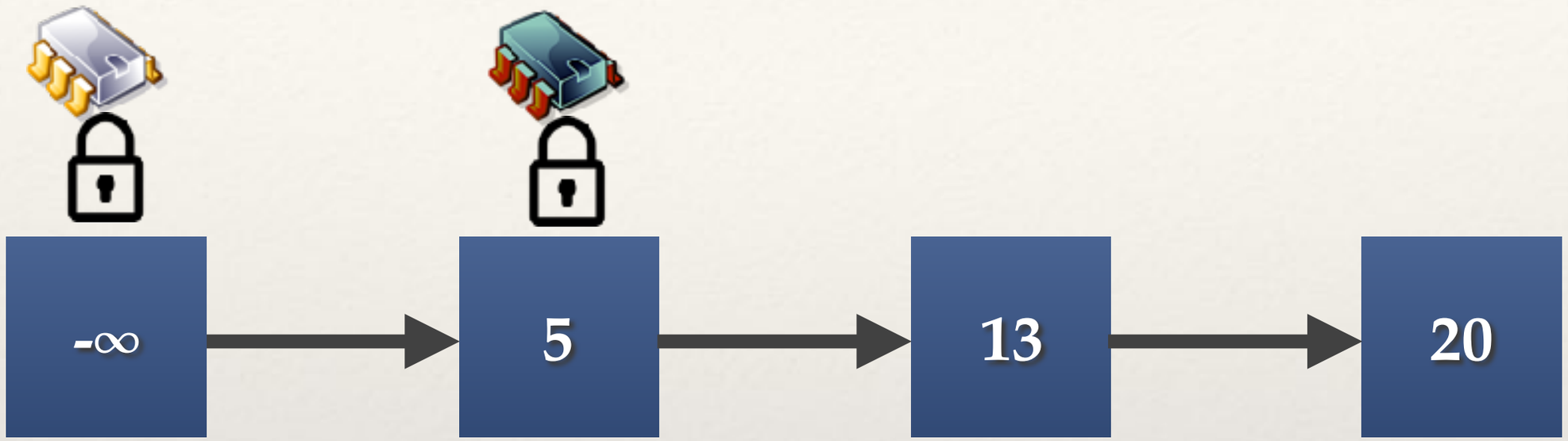
- Dynamically adjusts teleportation length
  - If successful: increases attempted length
  - If failed: decreases attempted length
    - If failed repeatedly:
      - Single transition, using baseline algorithm (fallback)
- Use *additive increase, multiplicative decrease* strategy
  - Minor effect on runtime

Applying Teleportation: Lock Coupling



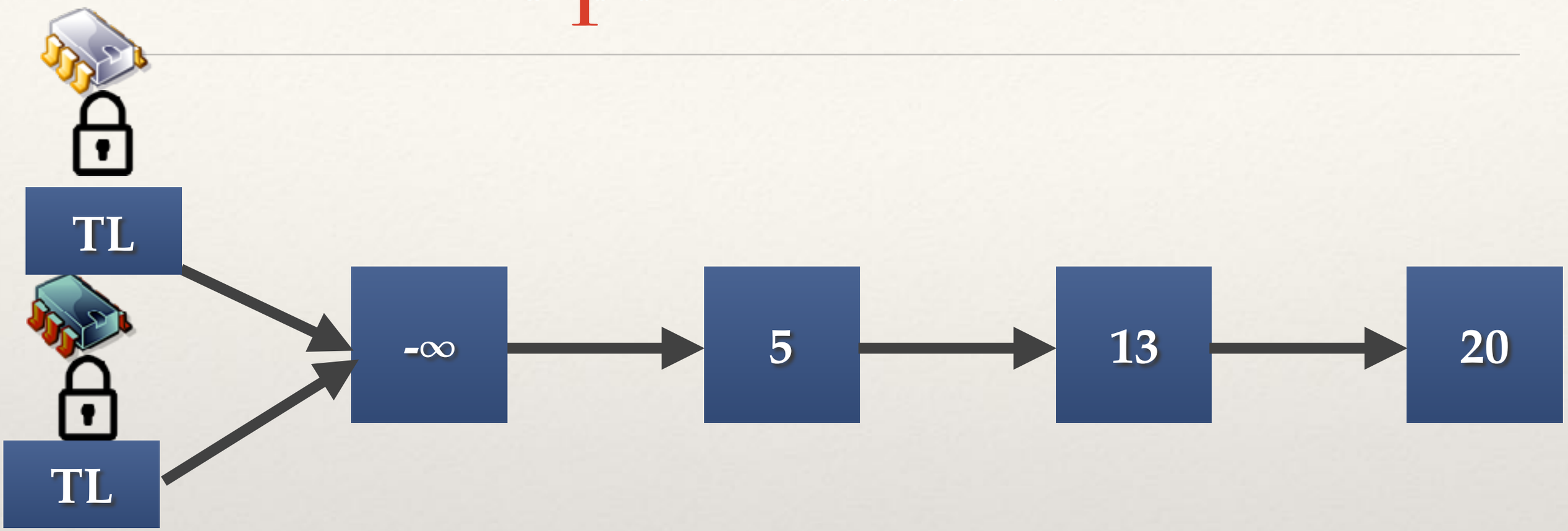
- Teleport locks
- Simple to develop
- Faster: significantly reduces acquires/releases

Optimizations



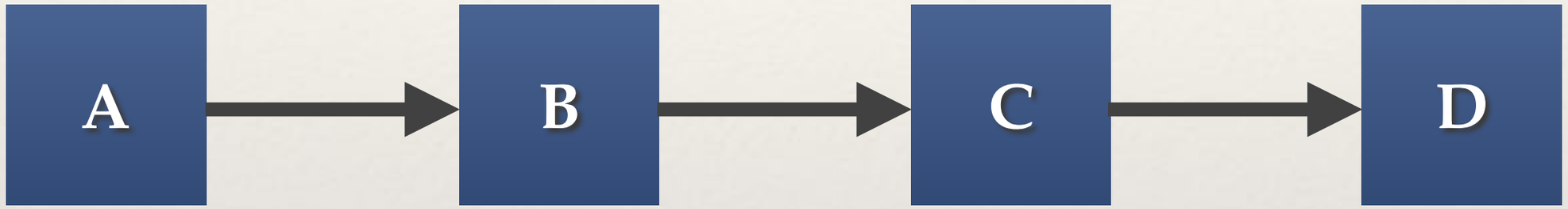
- Optimization 1: can bypass locked nodes
  - Signifiant improvement if thread asleep

Optimizations



- Optimization 1: can bypass locked nodes
  - Signifiant improvement if thread asleep
- Optimization 2: reduce contention on sentinel node

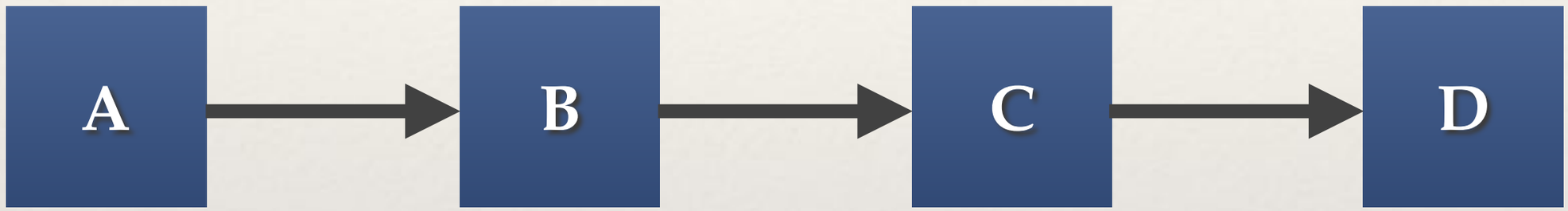
Applying Teleportation: Hazard Pointers



- Applicable to lock free algorithms?
  - Transition is seemly fast
  - But: memory management is **expensive**

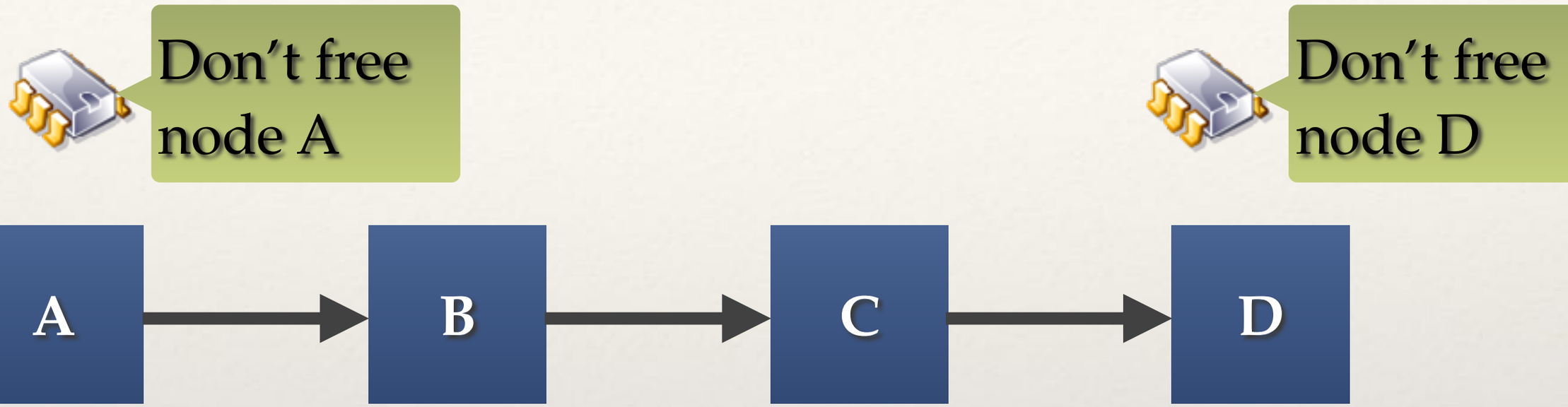
Applying Teleportation: Hazard Pointers

Don't free node A



- Hazard pointers: a lock free memory reclamation
- Requires memory fence per transition, thus slow
- Progress guarantee: don't wait for delayed threads

Teleporting Hazard Pointers

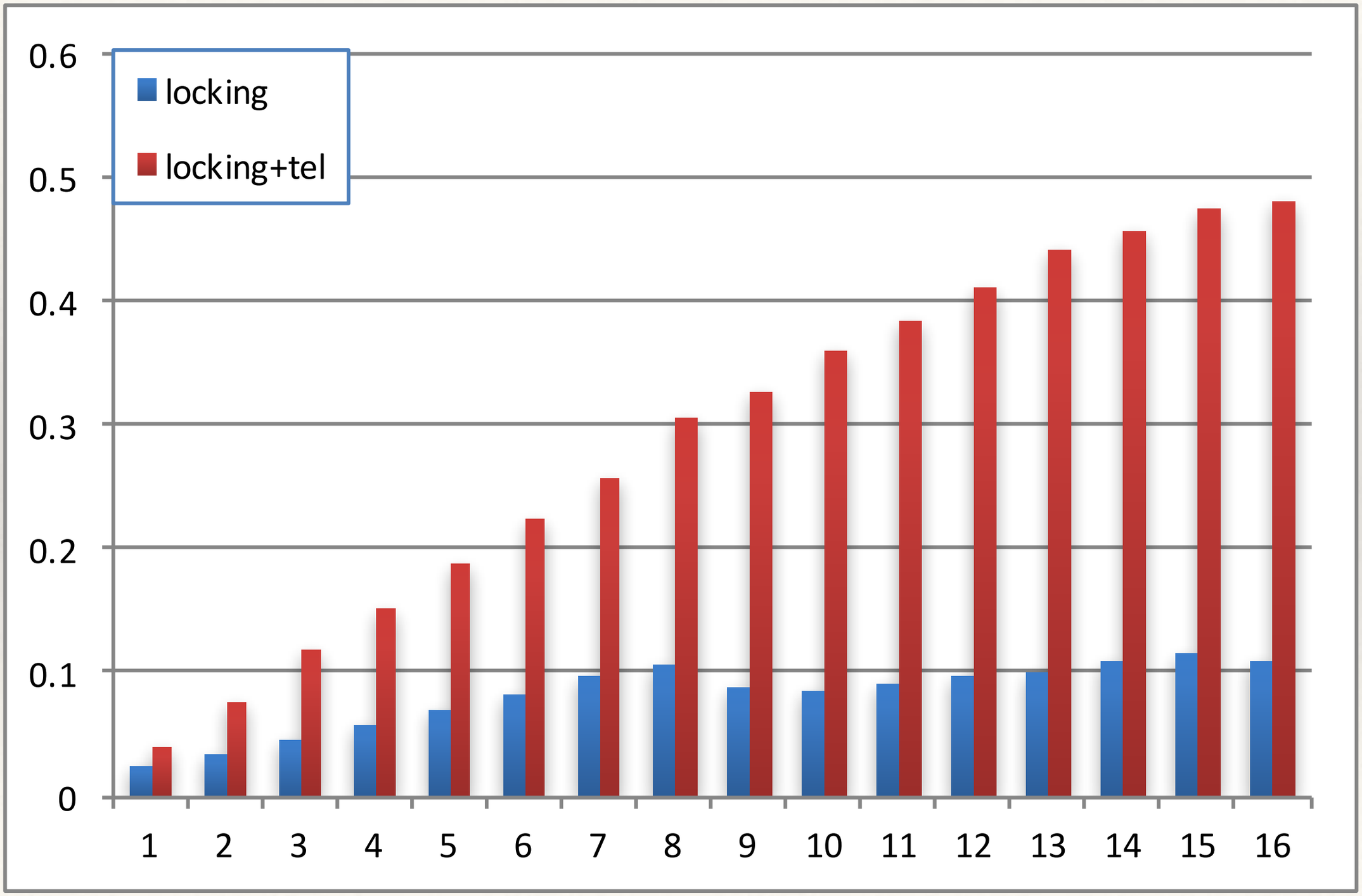


- Hazard pointer "teleports" to destination node
- Reduce memory fences on intermediate nodes

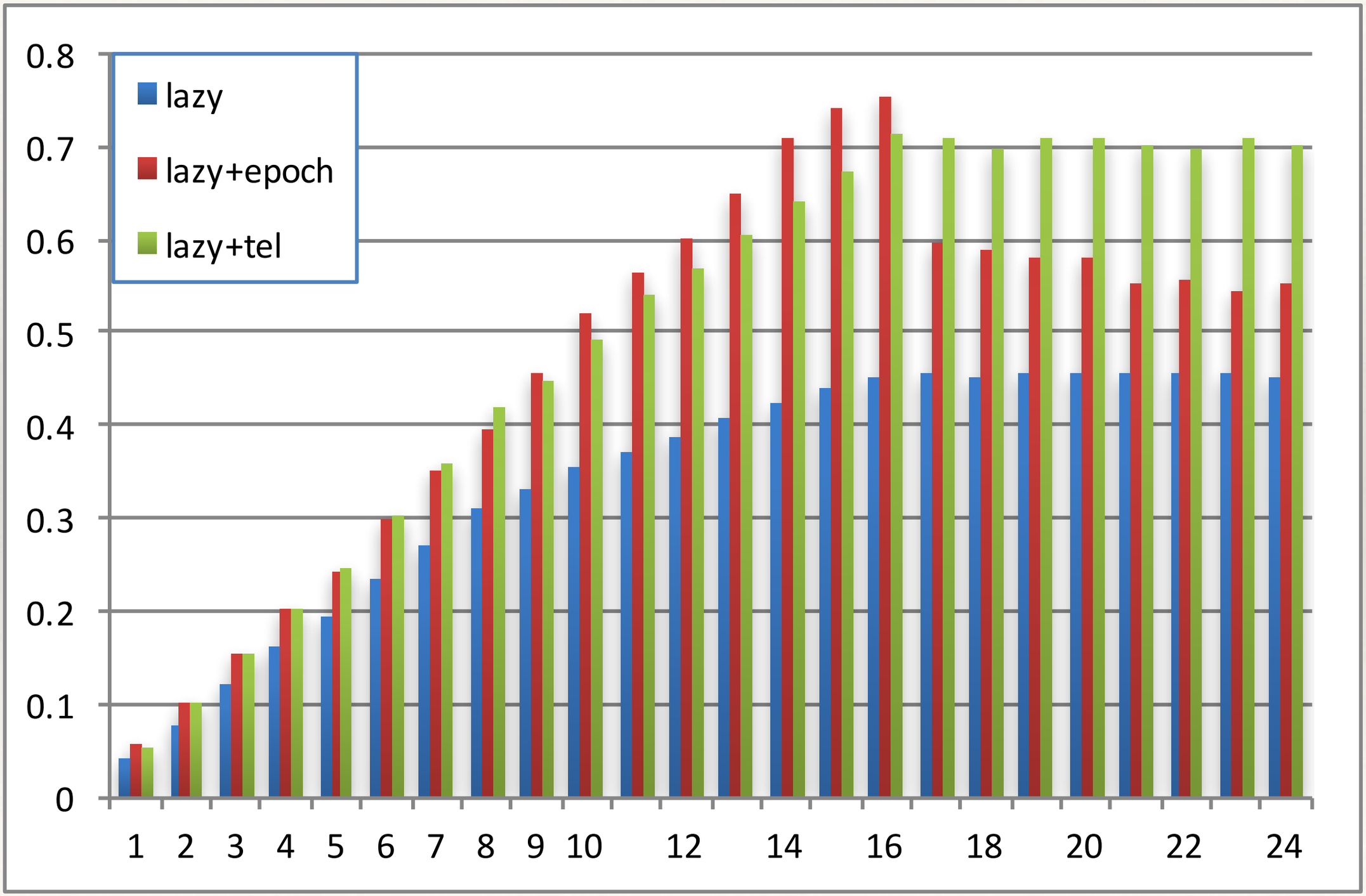
Measurements

- Data structures:
  - Linked list: lock-coupling, lazy, lock-free
  - Skip list: lazy
  - Tree: lock-coupling with range queries
- Machine: 8-core Broadwell (16 hardware threads)

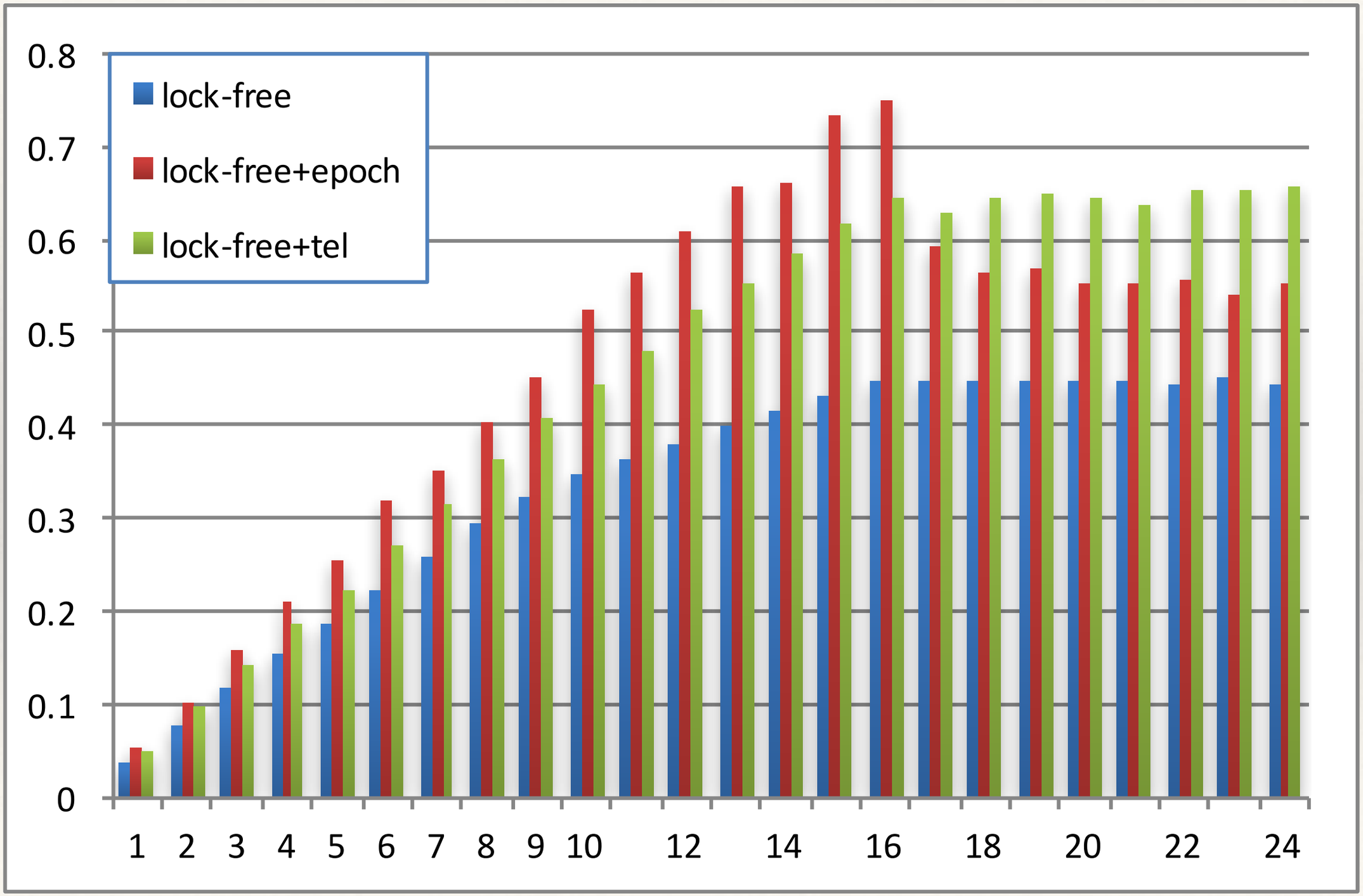
Linked List: Lock-Coupling



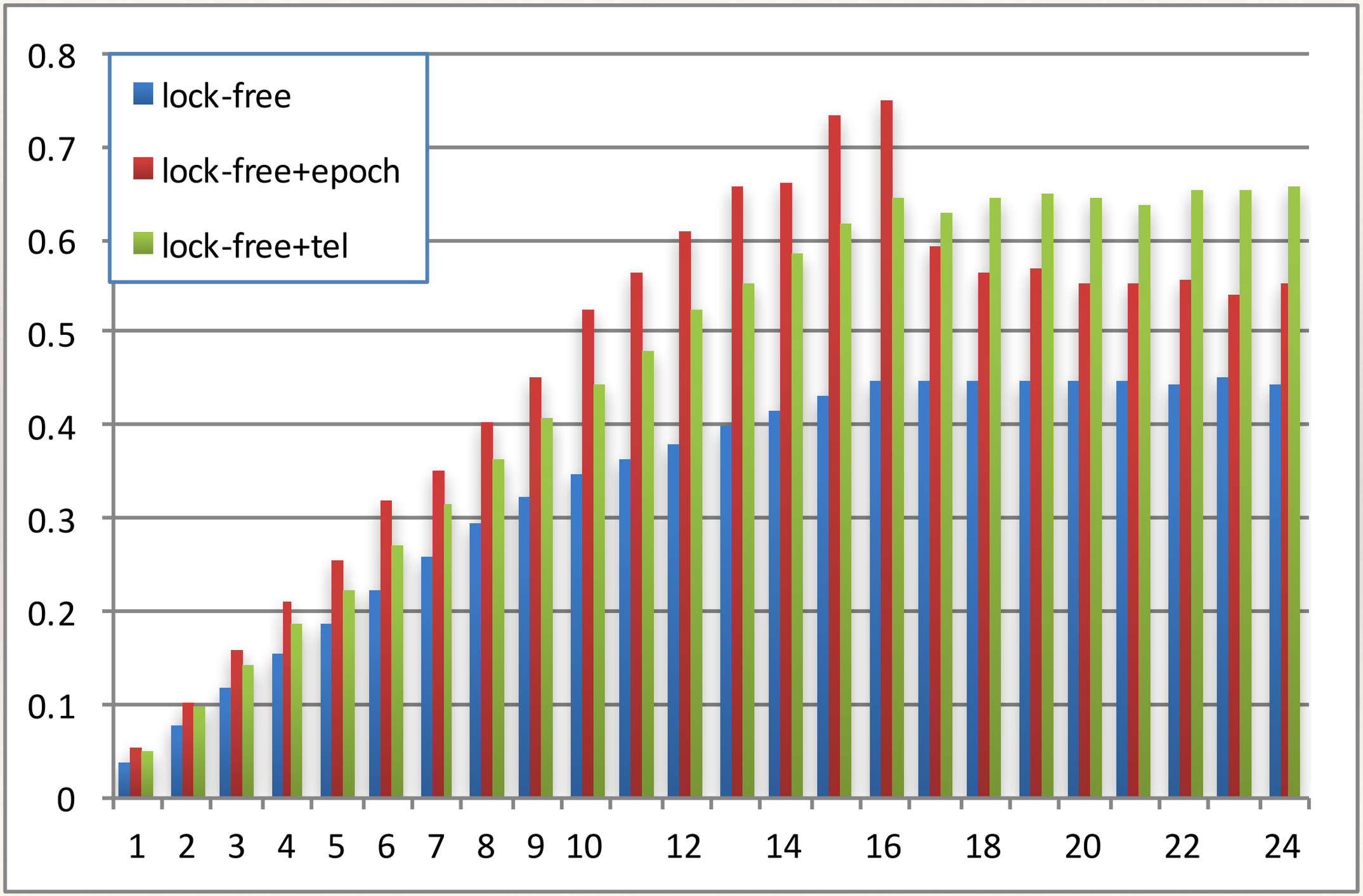
Linked List: Lazy+HP



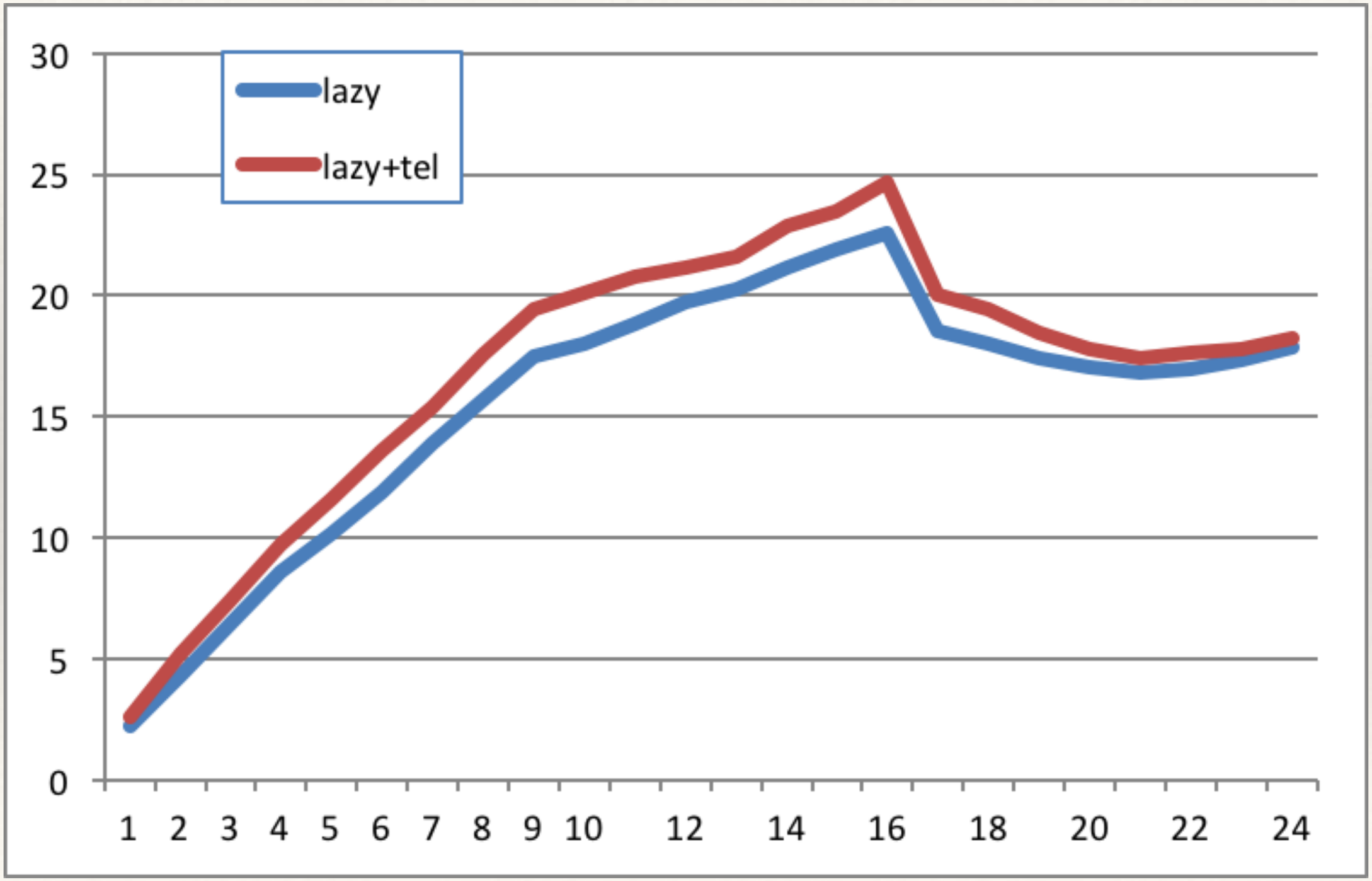
Linked List: Lock-Free+HP



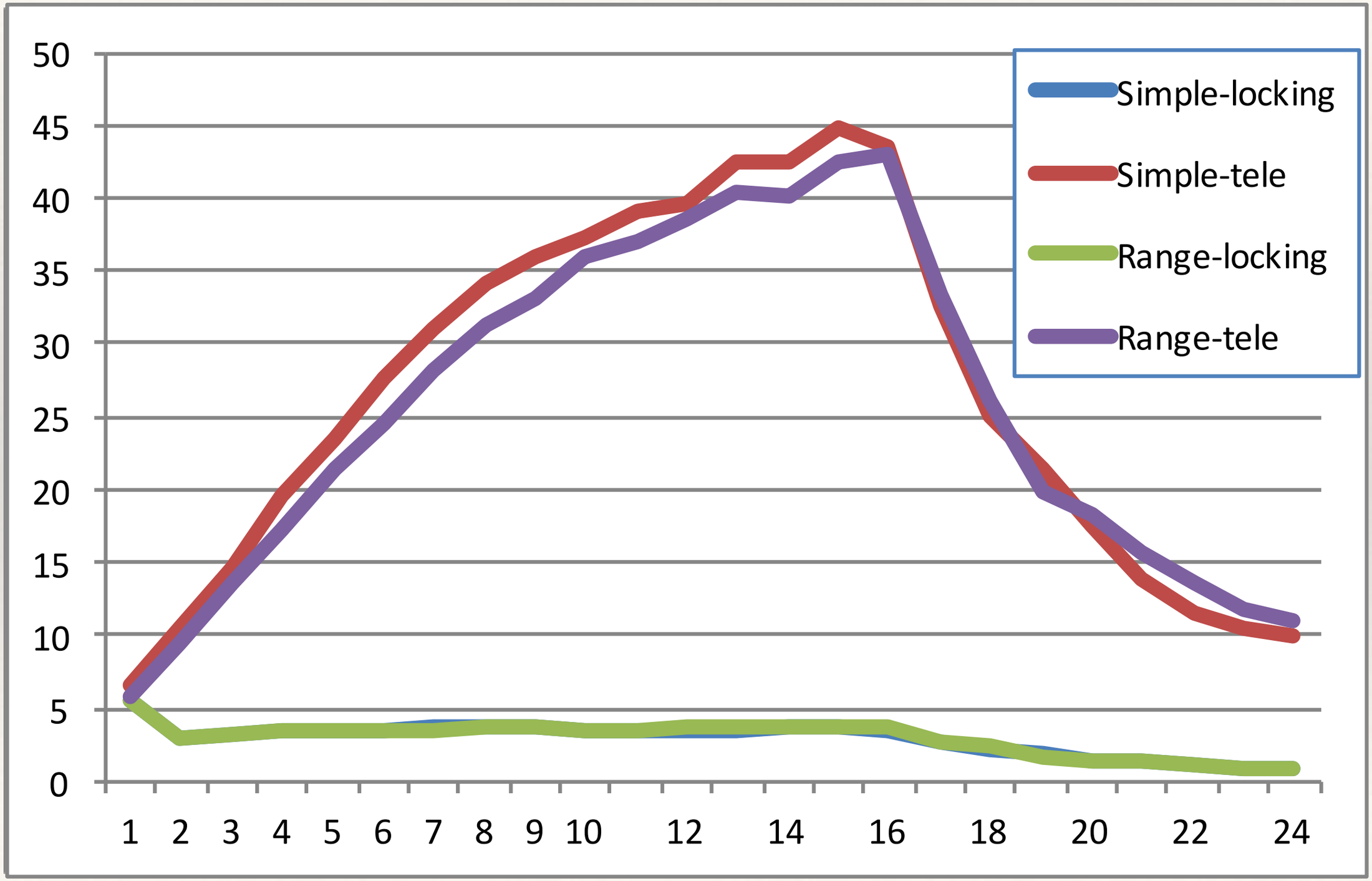
Linked List: Lock-Free+HP



Lazy Skip List + HP



Lock Coupling Tree with Range Queries



Summary

- Teleportation design pattern: “teleport” across multiple synchronous transitions
- Adaptive strategy matches best-effort hardware transactions
- Usage examples
  - Lock Coupling: teleport locks
  - Lazy / Lock-Free: teleport hazard pointers