Algorithms for Scalable Synchronization on Shared Memory Multiprocessors

by

John M. Mellor Crummey

Michael L. Scott

Presentation by

Joe Izraelevitz

Tim Kopp

Synchronization Primitives

- Spin Locks
  - Used for mutual exclusion around critical sequential sections
  - Spin locks used for nonscheduled processes, basic primitive for building complicated locks
- Barriers
  - Used to ensure all threads have left a parallel section
  - Similar to pthreads join()

Spin Locks: Evaluation

- Fairness
  - Starvation
  - FIFO Ordering
- Scalability
  - Network load
  - Space
- Hardware Requirements
  - Cache Coherence
  - Instruction Set
- Overhead

- Spin Locks
  - Test and Set
  - Ticket Lock
  - Array based Queuing Lock
  - MCS Lock

Test and Set Lock

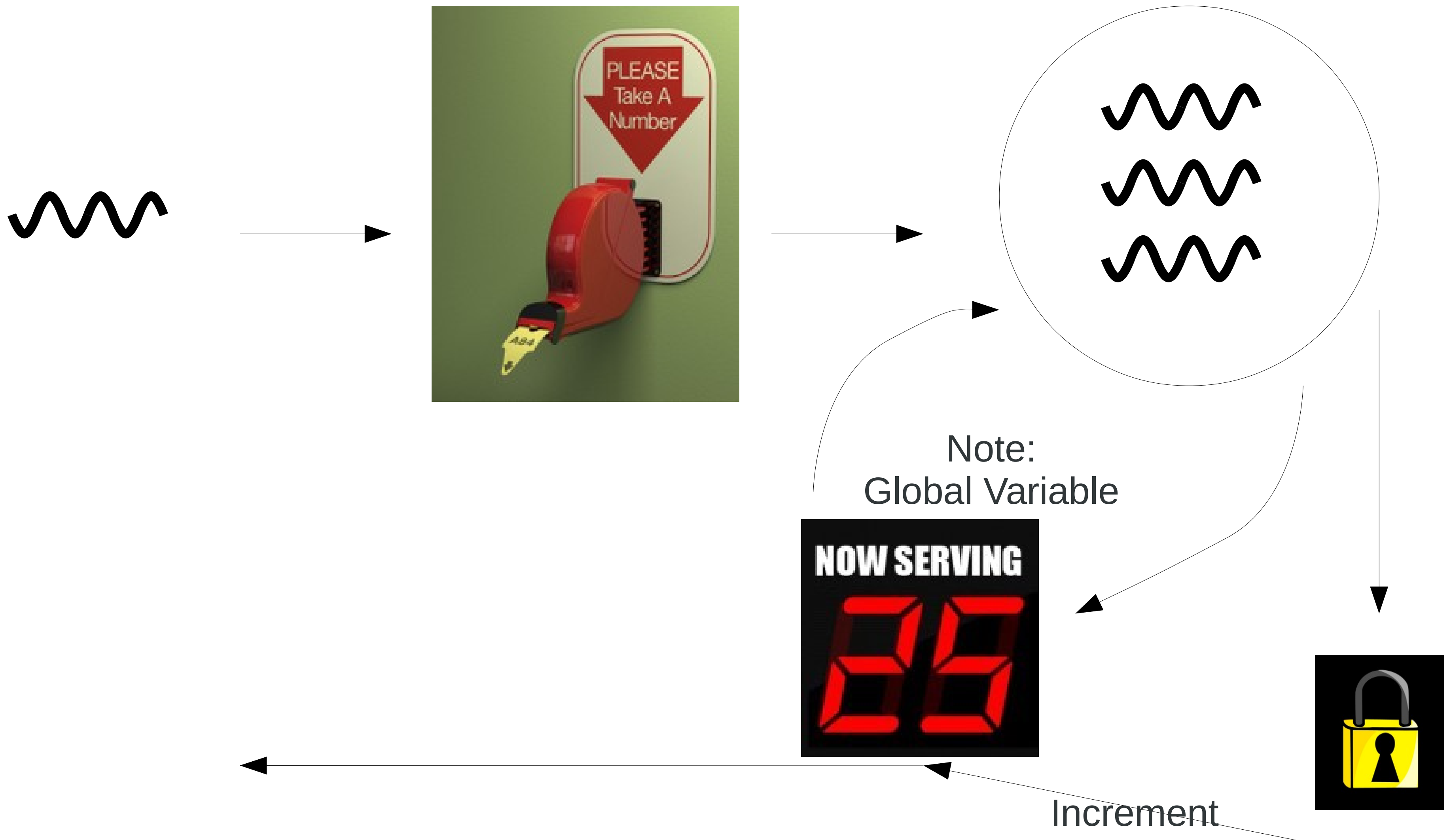
```
type lock = (unlocked, locked)

procedure acquirelock (L : lock)
    delay : integer = 1
    while (testandset (L) == locked) // returns old value
    pause delay // pause for short time
    delay = delay*2    // exponential back off

procedure releaselock (L : lock)
    lock = unlocked
```

Spin Locks: Evaluation

- Fairness
  - Starvation
  - FIFO Ordering
- Scalability
  - Network load
  - Space
- Hardware Requirements
  - Cache Coherence
  - Instruction Set
- Overhead

- Improvements
  - Backoff
    - Exponential
  - test and test_and_set

Ticket Lock



Ticket Lock

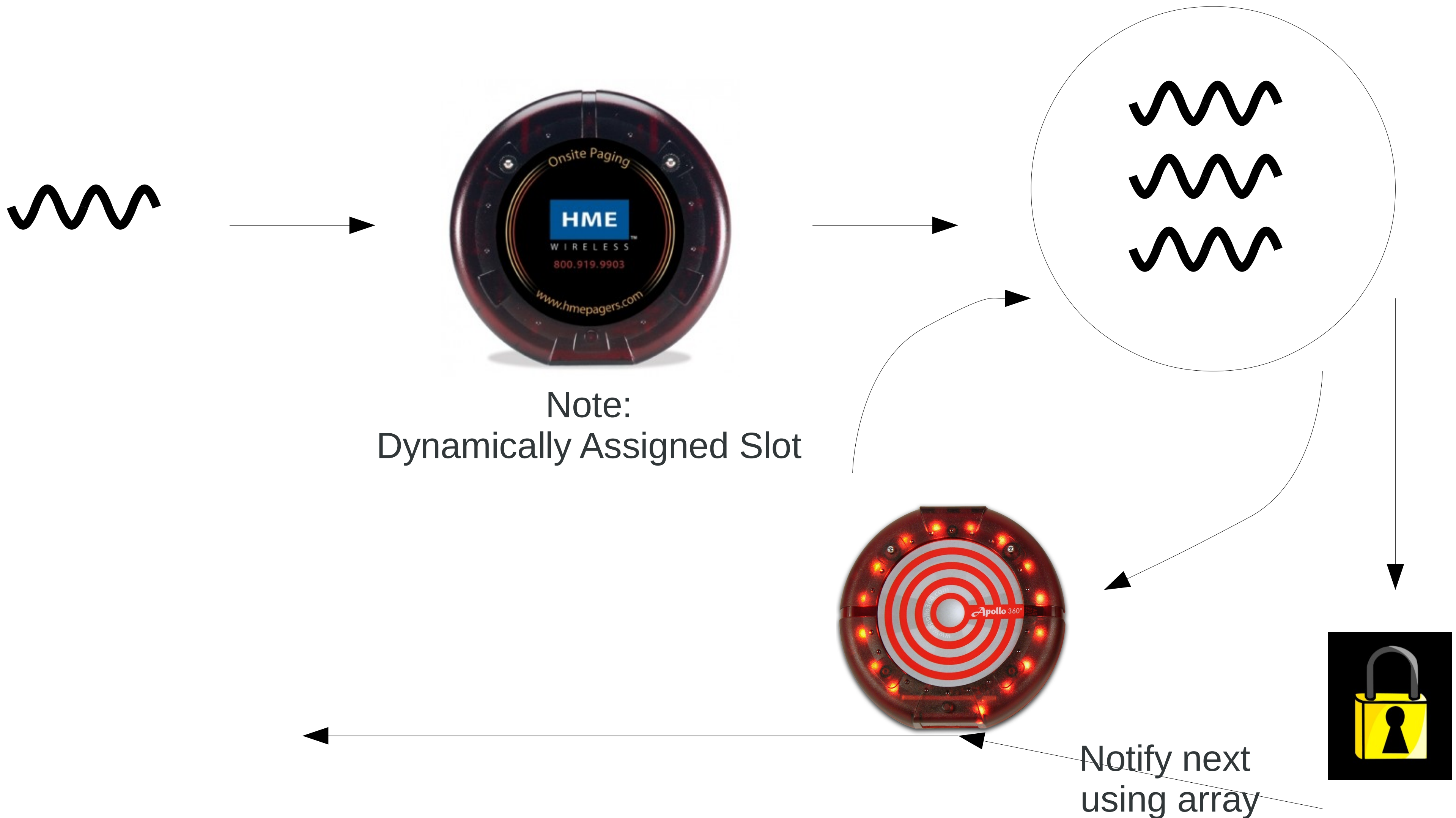
```
type lock = record
    nextticket : integer = 0 // the next ticket to be issued
    nowserving : integer = 0 // ticket number of thread with lock

procedure acquirelock (L : lock)
    myticket : integer = fetchandincrement(L->nextticket)
        // returns old value, arithmetic overflow is harmless
    loop
        pause (myticket – (L->nowserving) )
            // consume this many units of time
            // on most machines subtraction works correctly
            // despite overflow
        if (L->nowserving == myticket)
            return

procedure releaselock (L : lock)
    L->nowserving = L->nowserving + 1
```

Spin Locks: Evaluation

- Fairness
  - Starvation
  - FIFO Ordering
- Scalability
  - Network load
  - Space
- Hardware Requirements
  - Cache Coherence
  - Instruction Set
- Overhead

- Improvements
  - Backoff
    - Exponential
    - Proportional

Array Based Queuing Lock



Array Based Queuing Lock (Andersen)

```
type lock : record
    slots : array [0...numprocs-1] of (haslock, mustwait)
       = (haslock, mustwait, mustwait, ... mustwait)
         // each element of slots should lie in a different memory module or cache line
    nextslot : integer = 0


// function argument myplace points to a private variable in an enclosing scope
procedure acquirelock (L : lock, myplace : integer)
    myplace = fetchandincrement (L->nextslot)
              // returns old value, this provides my assigned slot to spin on
    if (myplace mod numprocs == 0) // decrement periodically to avoid overflow
         atomicadd (L->nextslot, -numprocs)
    myplace = myplace mod numprocs
    repeat while (L->slots[myplace] == mustwait) // spin until L->slots[myplace]==haslock
    L->slots[myplace] == mustwait // init for next time


procedure releaselock (L : lock, myplace : integer)
    L->slots[(myplace+1) mod numprocs] = haslock // give lock to next thread
```

Spin Locks: Evaluation

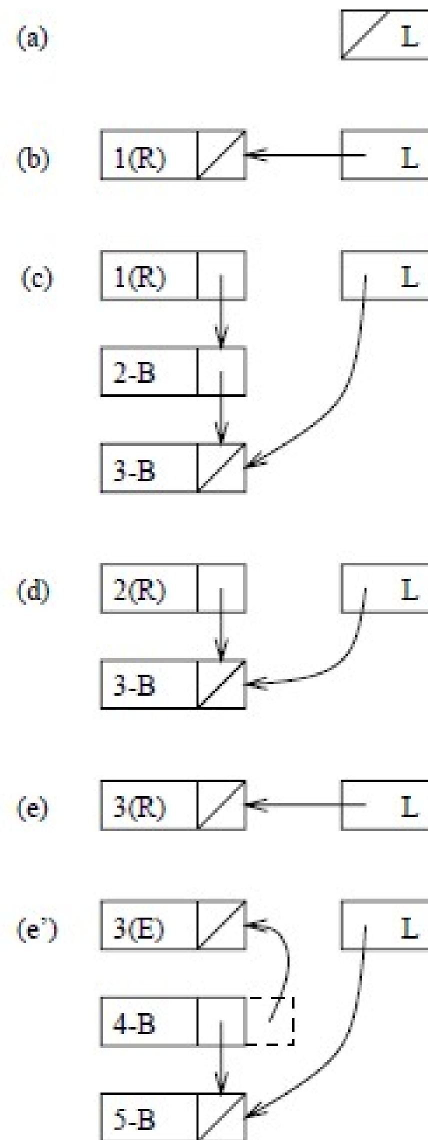
- Fairness
  - Starvation
  - FIFO Ordering
- Scalability
  - Network load
  - Space
- Hardware Requirements
  - Cache Coherence
  - Instruction Set
- Overhead

MCS Lock

MCS Lock

```
type qnode = record

     next : qnode

     locked : Boolean

type lock = qnode

// parameter I below points to a qnode record allocated in an enclosing scope in shared

// memory locally accessible to the invoking processor

procedure acquirelock (L : lock, I : qnode)

     I->next = nil

     predecessor : qnode = fetchandstore (L, I)

     if (predecessor != nil) // queue was non empty

          I->locked = true

          predecessor->next = I

          repeat while (I->locked) // spin on local qnode


procedure releaselock (L : lock, I : qnode)

     if (I->next == nil ) // no known successor

          if ( compareandswap (L, I, nil) )

               return // compareandswap succeeds if no thread has enqueued, we can exit
          // else, we spin until next thread registers with us, then unlock it and exit

          repeat while (I->next == nil)

     I->next->locked = false
```

MCS Lock



Spin Locks: Evaluation

- Fairness
  - Starvation
  - FIFO Ordering
- Scalability
  - Network load
  - Space
- Hardware Requirements
  - Cache Coherence
  - Instruction Set
- Overhead

Advantage Comparison

Lock	FIFO	Network Traffic	Space	Instruction Set	Overhead Ratio (single CPU compared to TSL)
Test and Set	No	O(n*t)	O(k)	test_and_set	1.0
Ticket	Yes	O(n*t)	O(k)	fetch_and_ increment	1.11
Array Queue	Yes	O(n)	O(#cpus)	fetch_and_ add	1.88
MCS	Yes	O(n)	O(n)	compare_ and_swap	2.04

* n = number of threads contending for lock
t = number of polls made in spin
k = constant

Speed Comparison

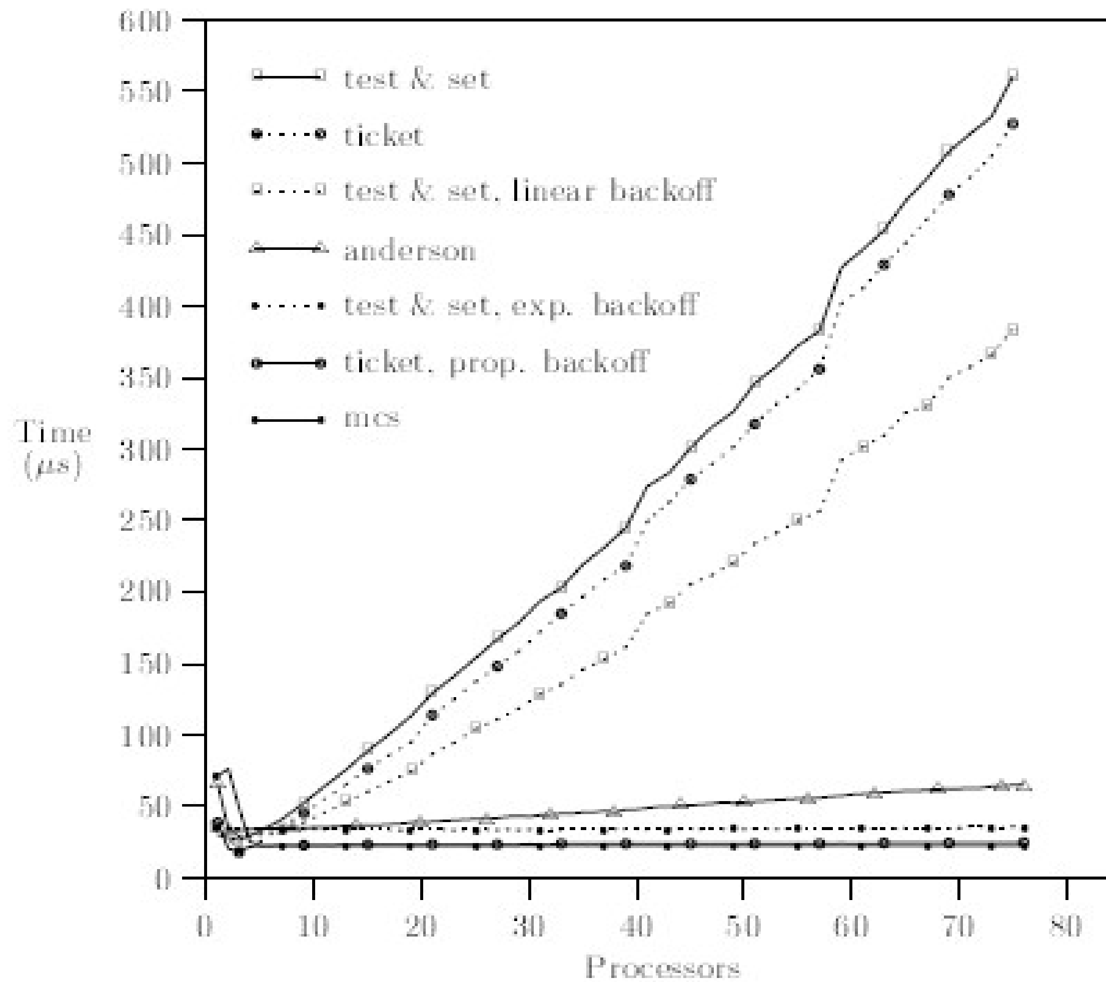


Figure 4: Performance of spin locks on the Butterfly (empty critical section).

Overhead Comparison

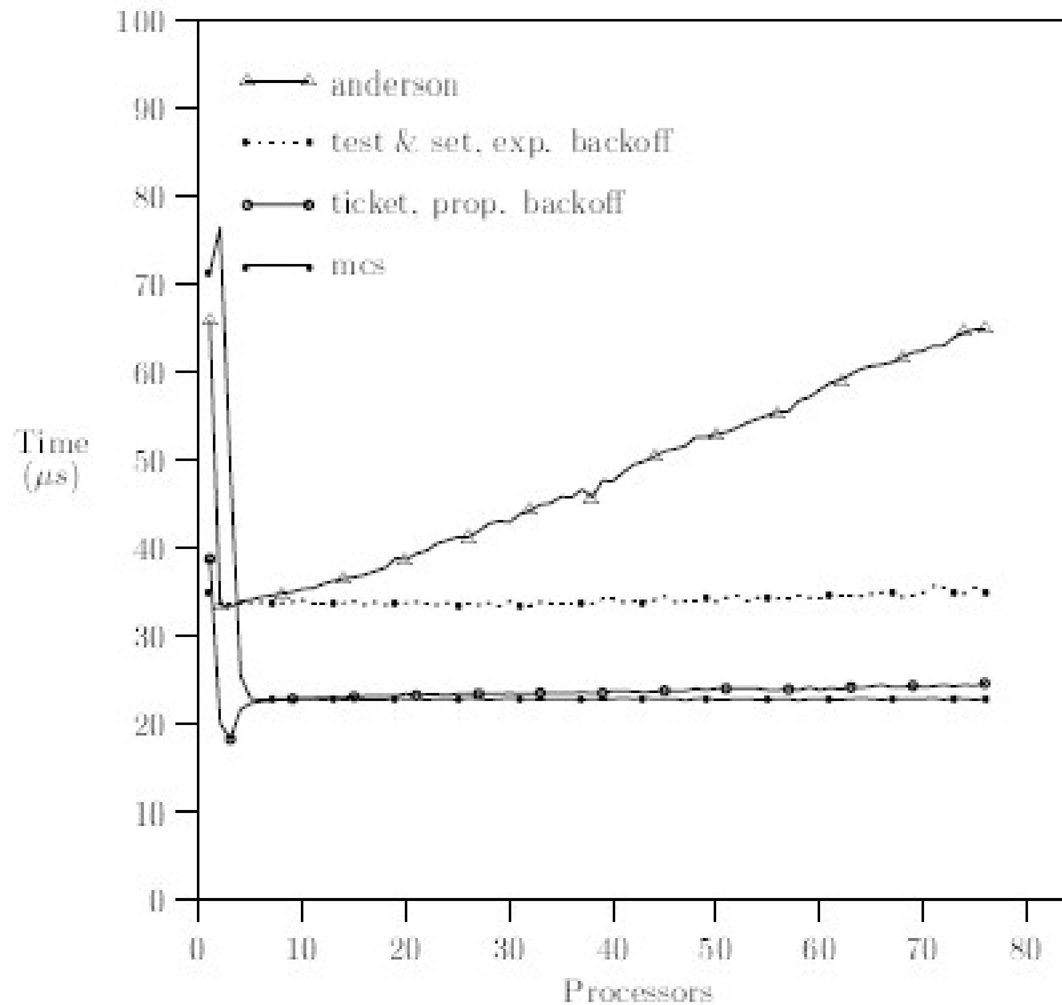
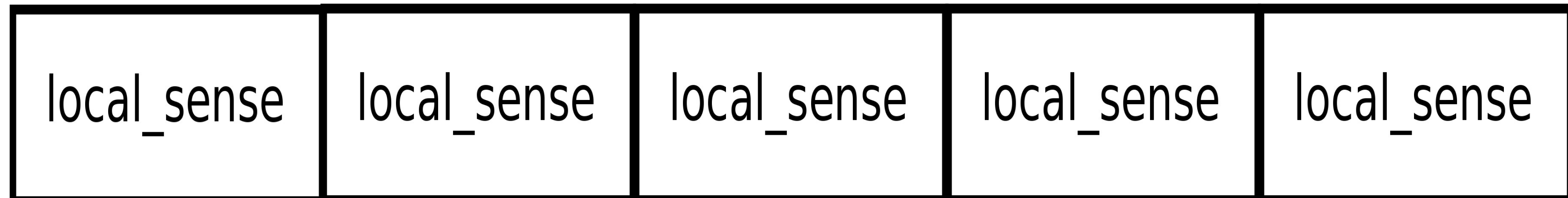


Figure 5: Performance of selected spin locks on the Butterfly (empty critical section).

Barriers

Centralized Barrier (Naive)

Centralized Barrier (Sense Reversal)



Centralized Barrier

- Ops on critical path = O(P)
- Space = O(1)
- Network transactions = O(P) with broadcast coherent caches, unbounded otherwise
- fetch_and_increment needed

Centralized Barrier

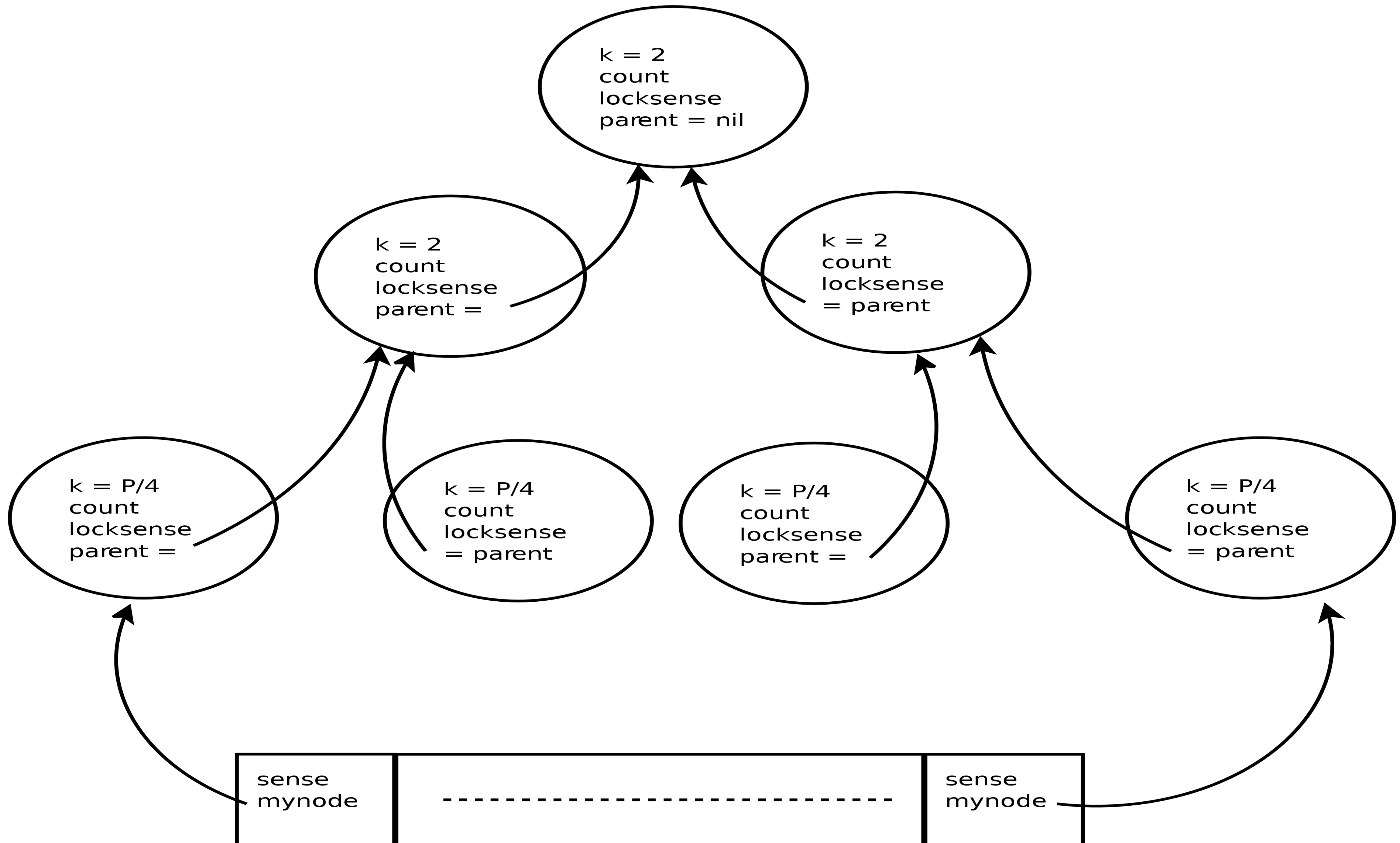
Pros

- Simple
- Less traffic on broadcast-based cache-coherent architectures
- Constant Space

Cons

- Many busy-wait accesses – significant network traffic
- Memory contention
- Cache thrashing
- Does not scale

Software-Combining Tree Barrier



Software Combining Tree Barrier

- Nodes scattered across different memory units or cache lines
- Ops on critical path = O(log P)
- Space = O(P)
- Network transactions = O(P) with broadcast coherent caches, unbounded otherwise
- fetch_and_increment needed

Software Combining Tree Barrier

Pros

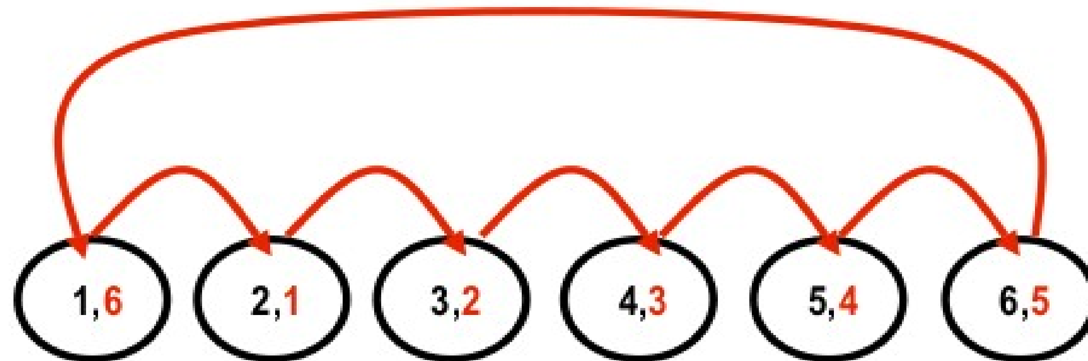
- Significant decrease in memory contention
- Prevents tree saturation in multi-stage interconnects

Cons

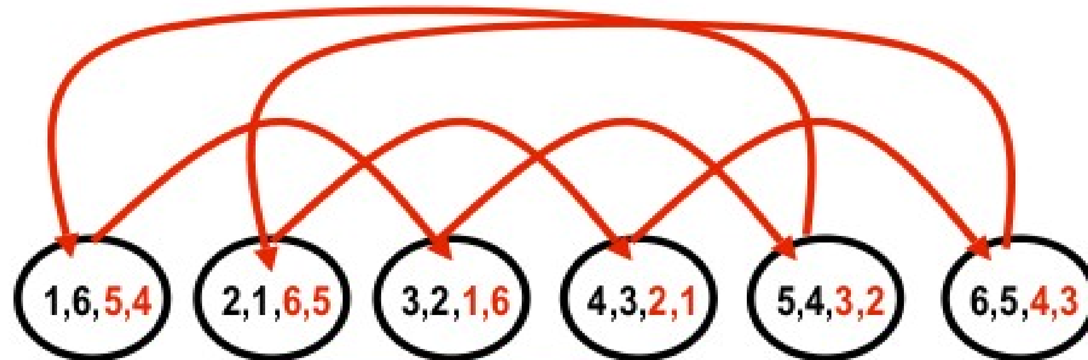
- Processors spin on memory locations that cannot be statically determined
- Processors spin on memory locations that other processes spin on

Dissemination Barrier

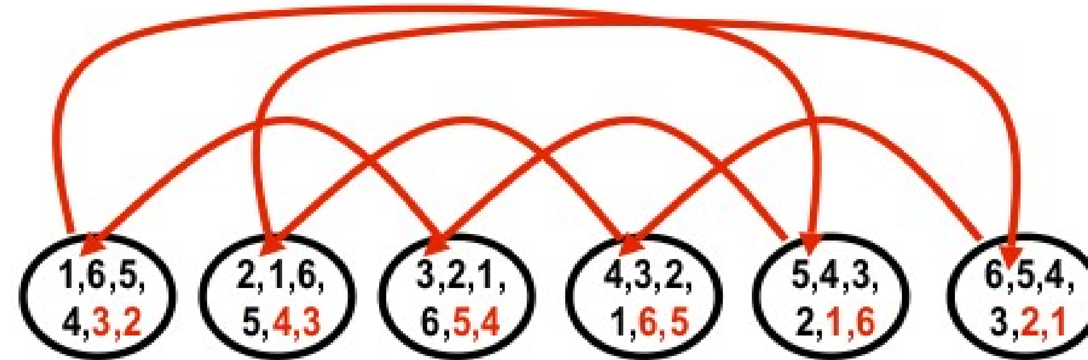
Round 1



Round 2



Round 3



Dissemination Barrier

- Uses two sets of variables to avoid double spinning
- Network transactions = O(P log P)
- Ops in critical path = O(log P)
- Space = O(P log P)
- No need for fetch_and_Φ instructions

Dissemination Barrier

Pros

- All spinning is local
- Eliminates remote spinning

Cons

- With distributed shared memory and no coherent caches, must scatter state across memory banks

Tournament Barrier

- Combining tree of fan-in two
- Rather than waiting on a node, losers drop out and wait on shared variable
- Winners statically determined
- Tournament winner changes shared state

Tournament Barrier

- Network transactions = O(P)
- Ops in critical path = O(log P)
- Space = O(P) or O(P log P)
- No need for fetch_and_Φ instructions

Tournament Barrier

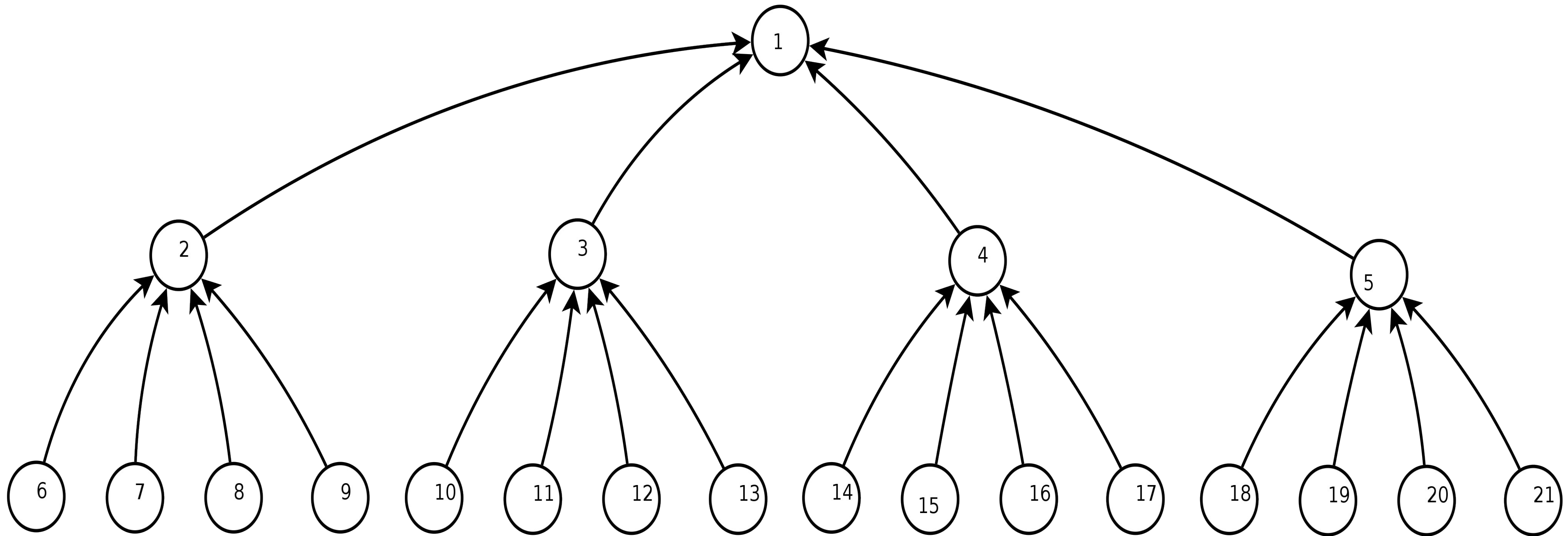
Pros

- Efficient when broadcast is used for cache consistency
- Eliminates remote spinning

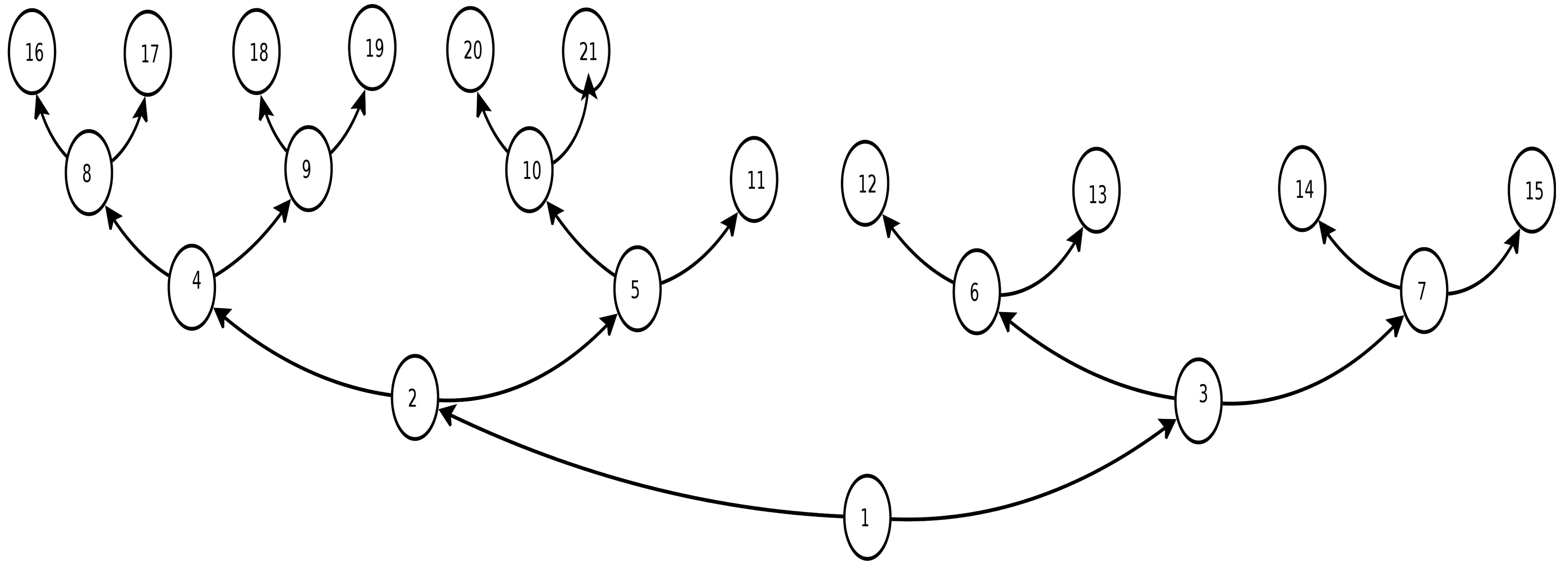
Cons

- Heavy interconnect traffic if no coherent caches

MCS Barrier - Arrival



MCS Barrier - Wake



MCS Barrier

- Network transactions = 2p-2 = O(P), theoretical minimum without broadcast
- Space = O(P)
- Ops on critical path = O(log P)
- No need for fetch_and_Φ instructions

MCS Barrier

- With coherent caches, could replace wakeup with spin on global variable
- Minimal network traffic

Advantage Comparison

Barrier	Critical Path Length	# of Network Transactions	Space	Instruction Set
Centralized (count)	O(P)	O(P) or Unbounded	O(1)	fetch_and_ increment
Software Combining Tree	O(log P)	O(P) or Unbounded	O(P)	fetch_and_ increment
Dissemination	O(log P)	O(P log P)	O(P log P)	Atomic read/write
Tournament	O(log P)	O(P)	O(P log P)	Atomic read/write
MCS Tree	O(log P)	O(P)	O(P)	Atomic read/write

Speed Comparison (Non Cache Coherent)

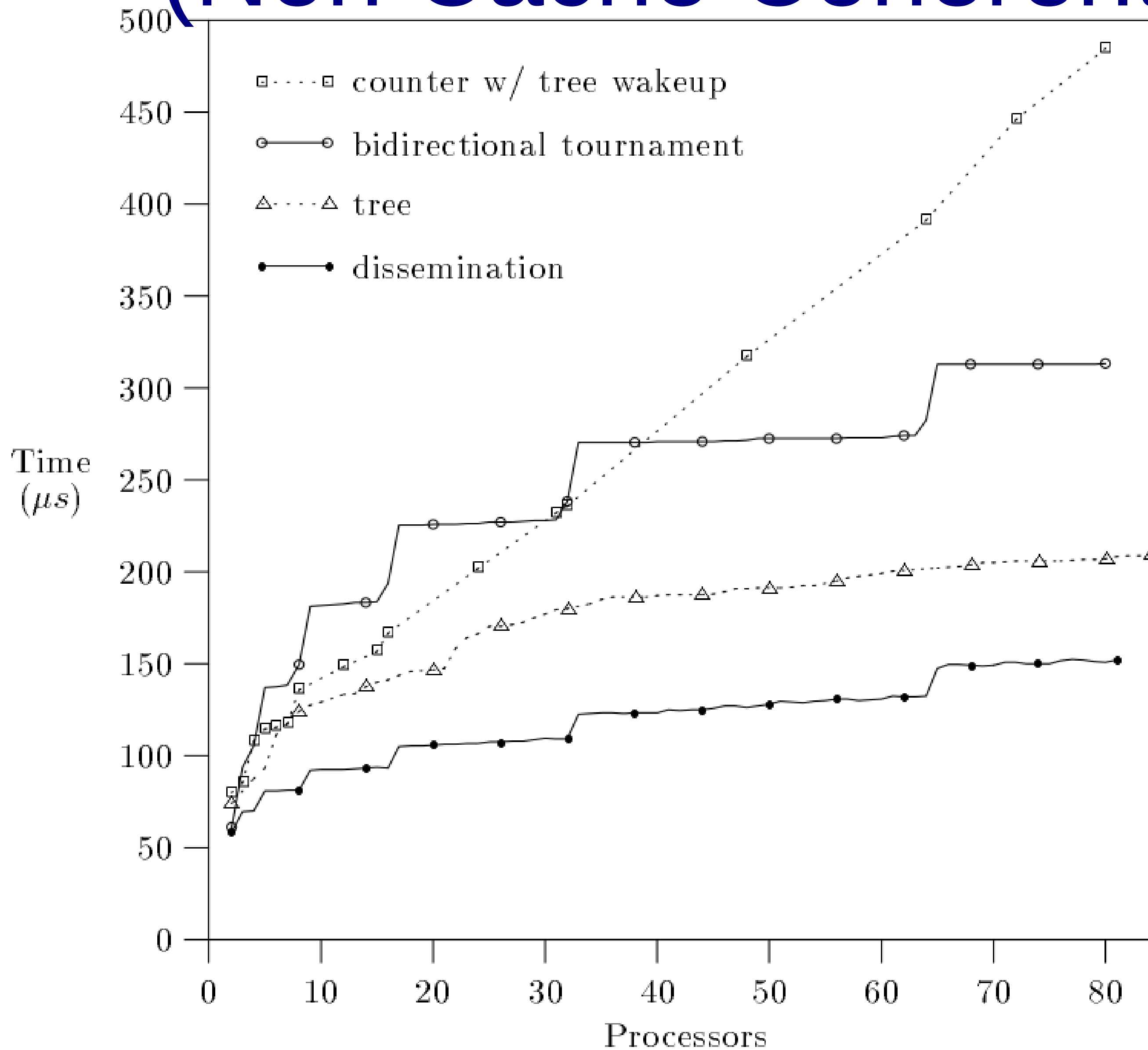


Figure 9: Performance of selected barriers on the Butterfly.

Speed Comparison (Cache Coherent)

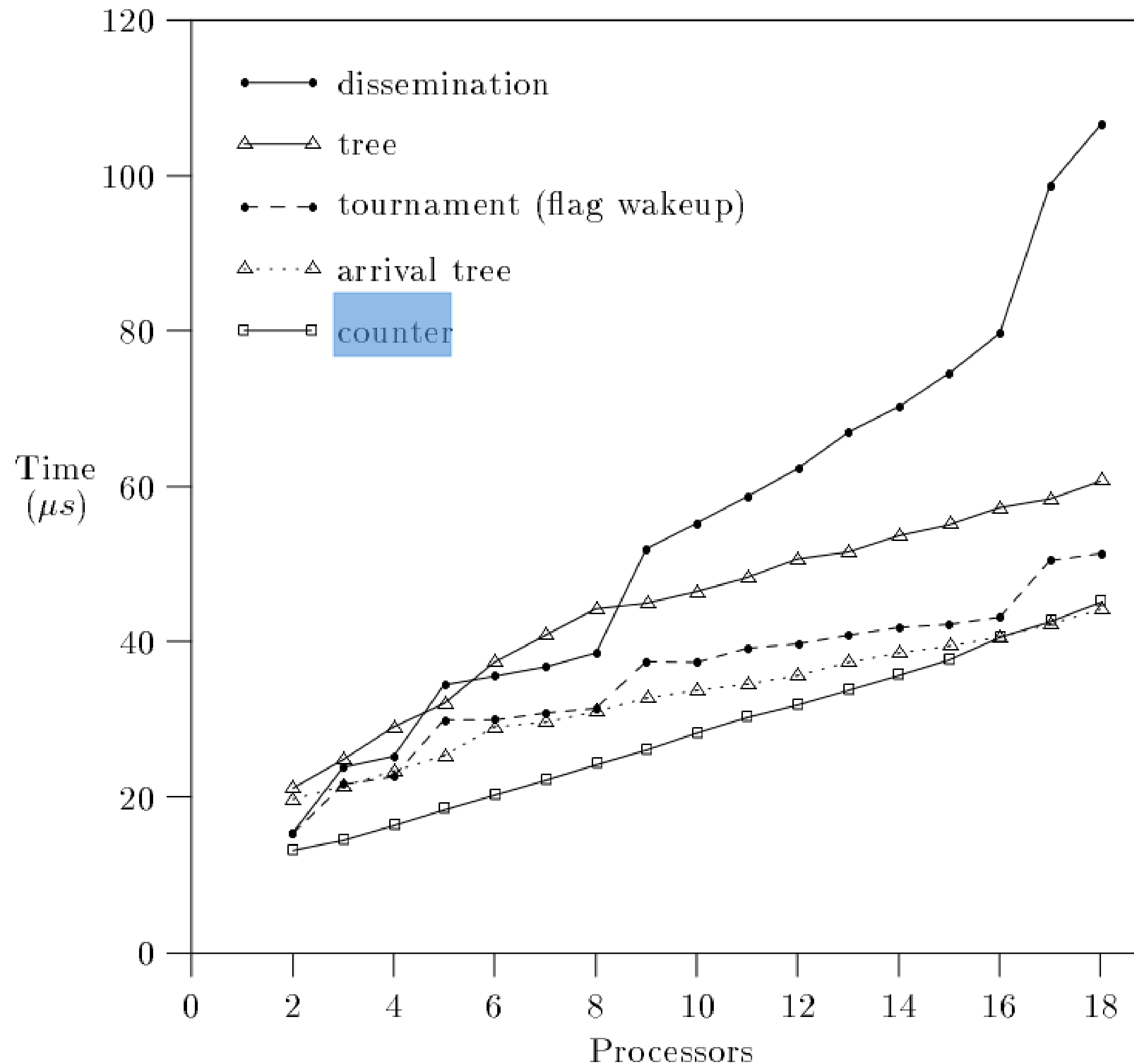


Figure 10: Performance of barriers on the Symmetry.

Questions?

Images Cited

- The Electronic Avenue Inc. "Employee and Guest Pagers." http://www.posequipment.net/guest-pagers.html (2013). (Pager on image)
- Finding the Humor. http://findingthehumor.com/kids-2/number/ (2013). (Now Serving image)
- Genex Disability. http://www.genexdisability.com/social-security-disability-blog/bid/59559/Social-Security-Field-Offices-Face-Challenges-With-Servicing-Customers (2013). (Please take a number image)
- Open Clip Art. http://openclipart.org/detail/22179/lock-by-nicubunu (2013). (Lock image)
- Mellor Crummey, John. Comp 422 Lecture Slides, Rice University (Dissemination Barrier Image)
- Mellor Crummey, John M. and Michael L. Scott. "Algorithms for Scalable Synchronization on Shared Memory Multiprocessors." *ACM Trans. on Computer Systems.* February 1991. (Algorithm Comparison Charts)
- PRLOG. "Fajitas and 'Ritas Keeps Customers Festive with GuestCall paging from HME Wireless." (2010). http://www.prlog.org/11082855-fajitas-and-ritas-keeps-customers-festive-with-guestcall-paging-from-hme-wireless.html (Pager off image)