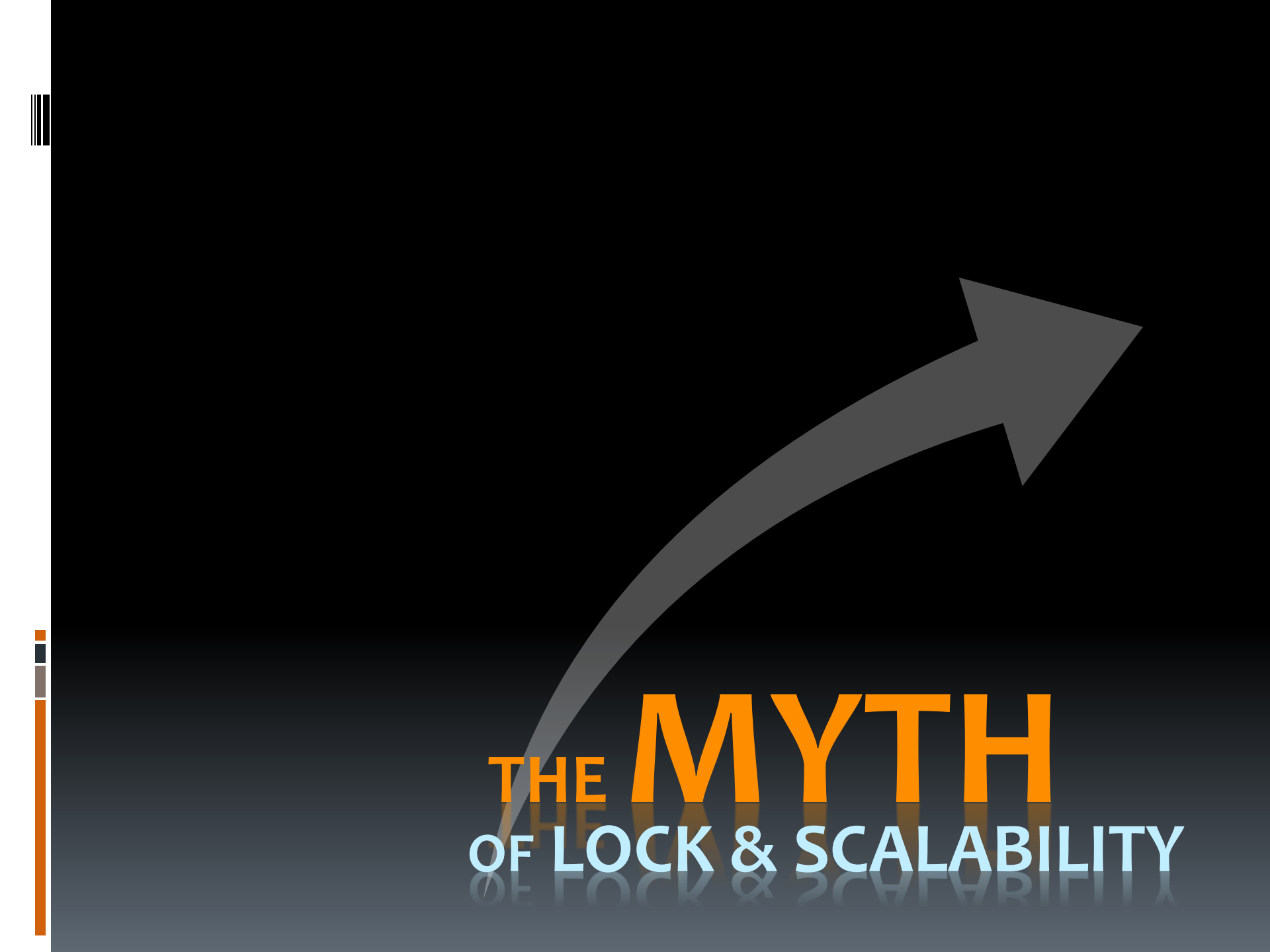
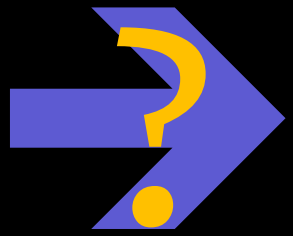

THE MYTH
OF LOCK & SCALABILITY

Introduction to scalability

What is Scalability

More worker  Higher productivity

Wait, why adding more workers?

Why parallel?

The answer is performance

Performance

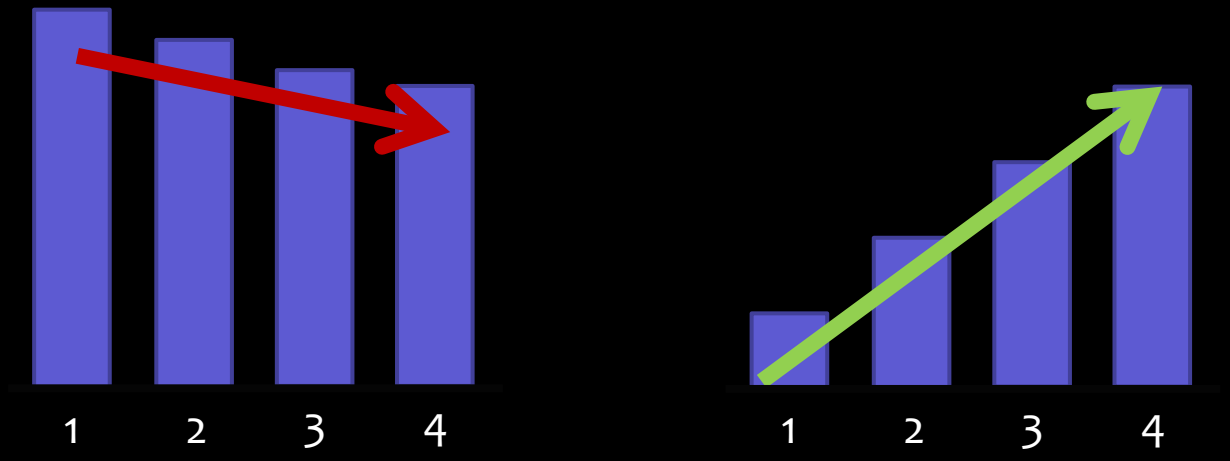
- if performance is not a concern, why not do yourself a favor, just write sequential code, and be happy?

--paulmck

What about scalability?

- The first goal is performance rather than scalability

Why?



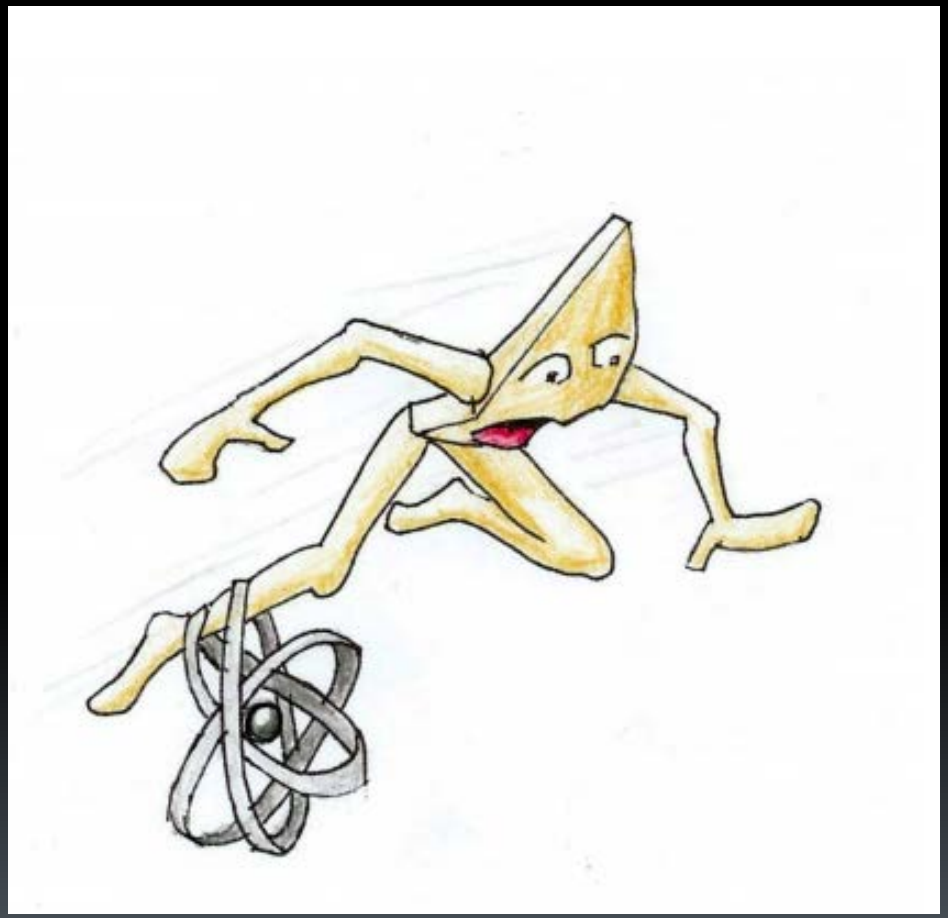
- The easiest way to attain linear scalability is to reduce the performance of each CPU

What affects single CPU performance?

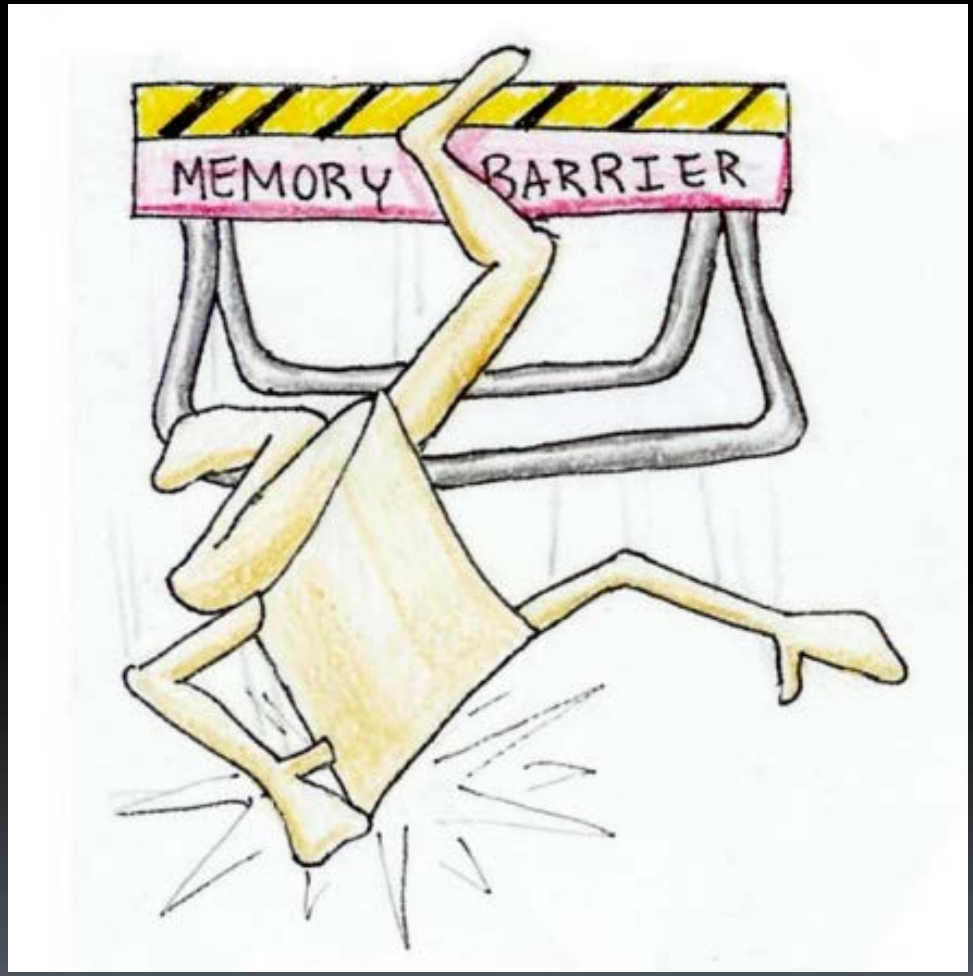
Pipeline Flush

Memory Reference

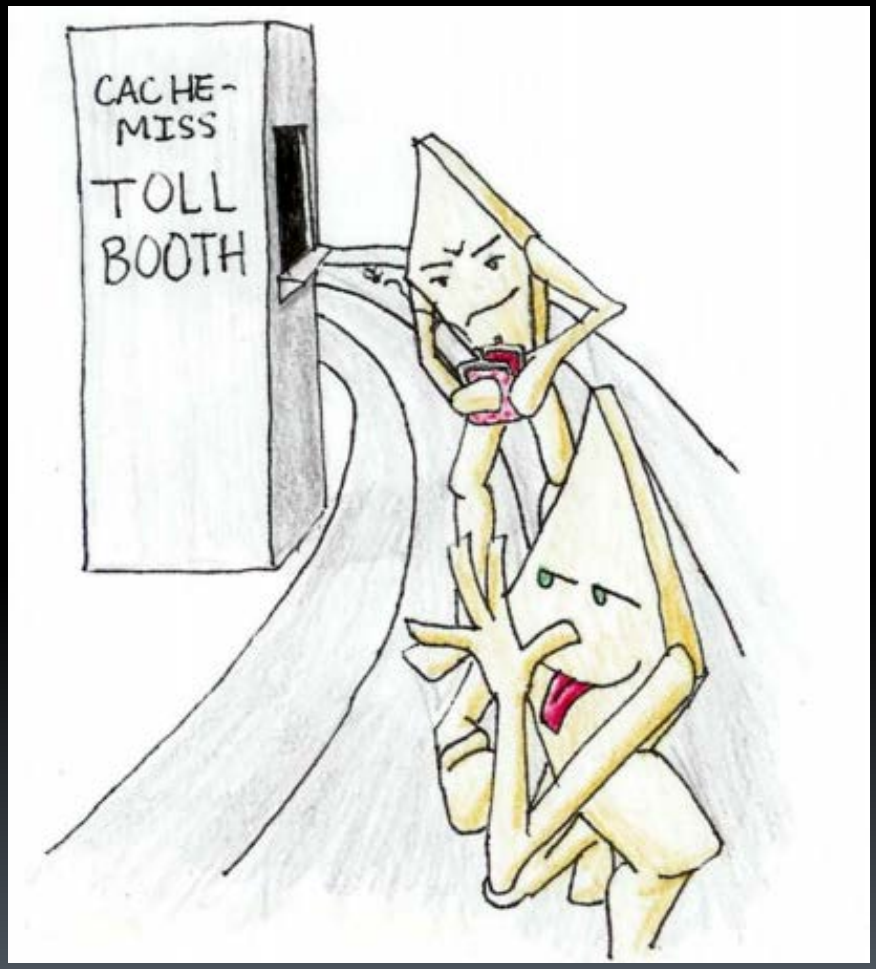
Atomic Operation



Memory Barrier



Cache Miss



I/O


TELE
Please stay on the line. Your call is very important to us...

Summary

- Pipeline Flush
- Memory Reference
- Atomic Operation
- Memory Barrier
- Cache Miss
- I/O

} Communication?

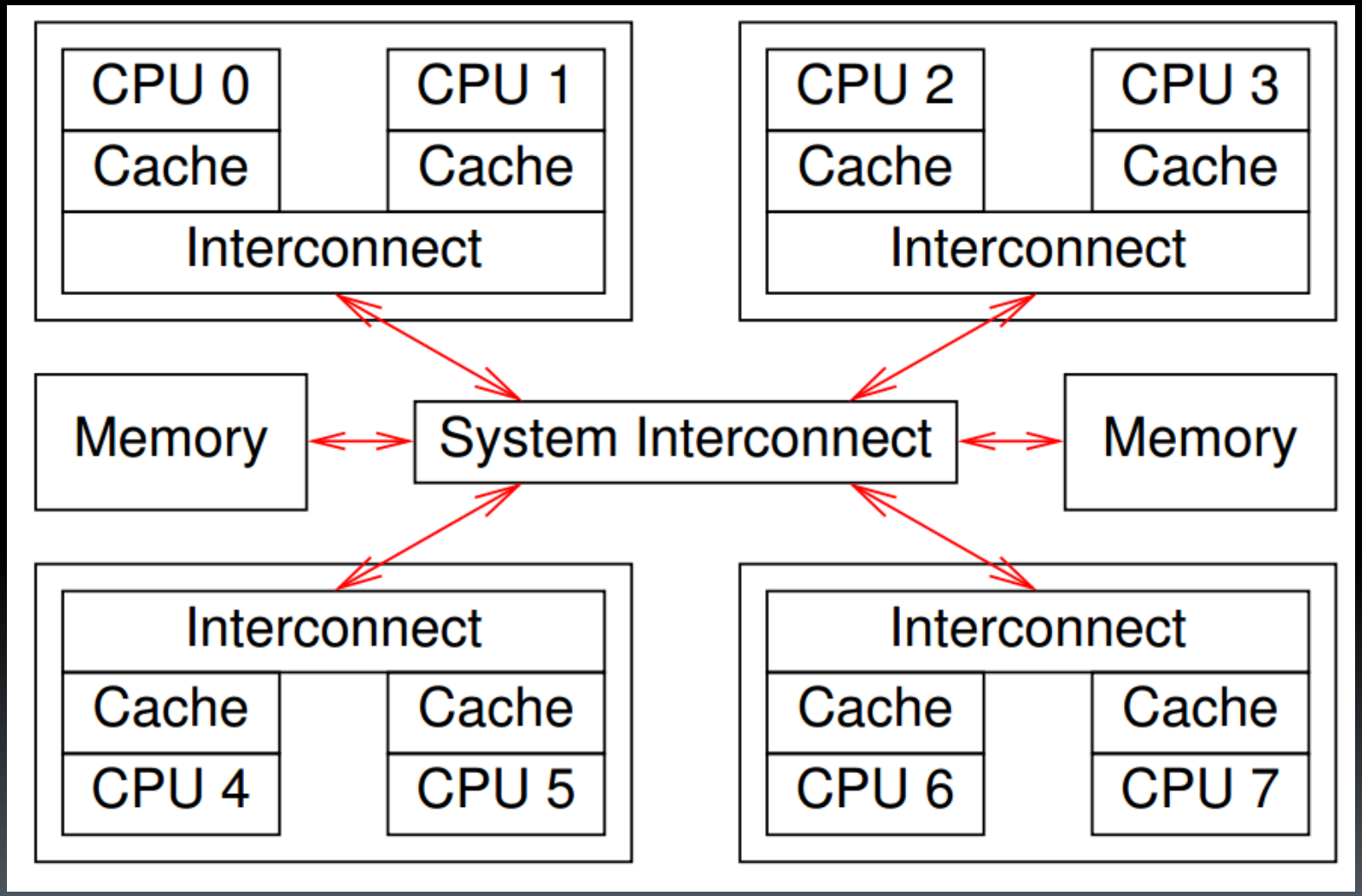
Why Communication?

- Communication is the key to parallel programming
- A thread do not communicate at all need not to be executed

Communication Mechanism

- Message Passing
- Shared Memory
  - Do we really have shared memory?

Real Hardware



Back to Scalability: An ideal model

- A CPU can be in three state
  - Executing
  - **Waiting to send message**
  - **Waiting for a message**

- Pipeline Flush
- Memory Reference
- Atomic Operation
- Memory Barrier
- Cache Miss
- I/O

Waiting in Different Scenario

- Low level
  - Wait for cache invalidation message
  - Wait to gain access to bus
  - …
- High level
  - Spin: Wait for cache invalidation message
  - Sleep: Wait for IPI
  - …

Messages is all what we care about

- Scalability Problem
  - Type I: Waiting to send message

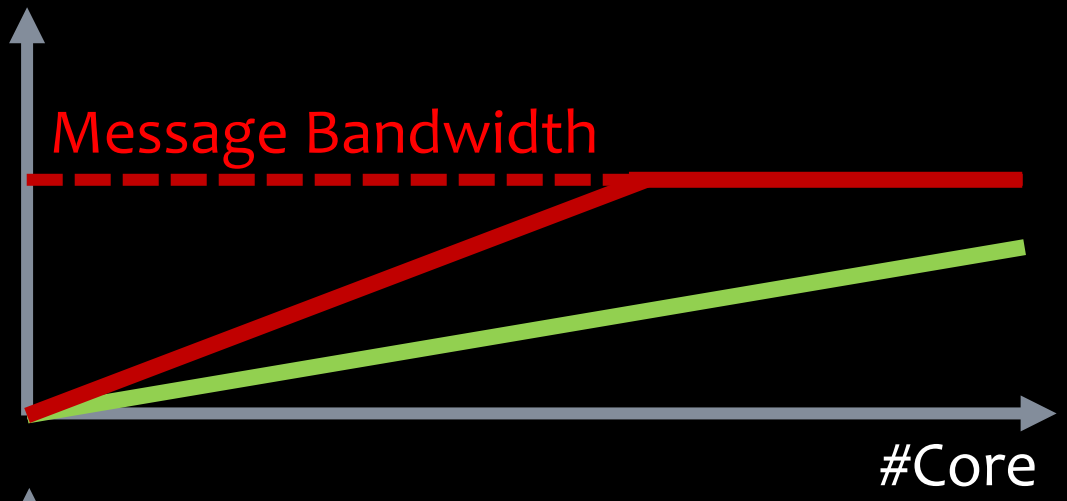
Physical Limitation

Logical Dependency

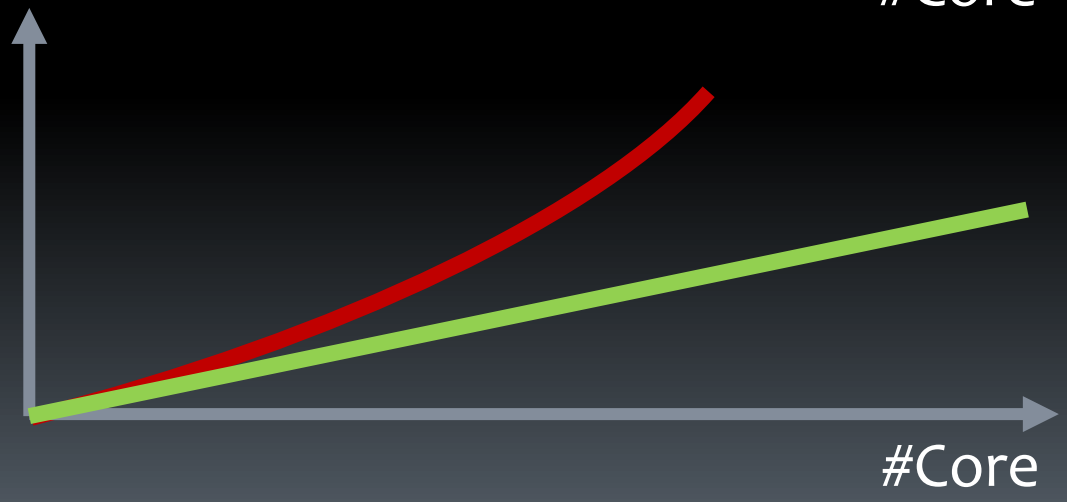
  - Type II: Waiting for a message

What is a scalable program

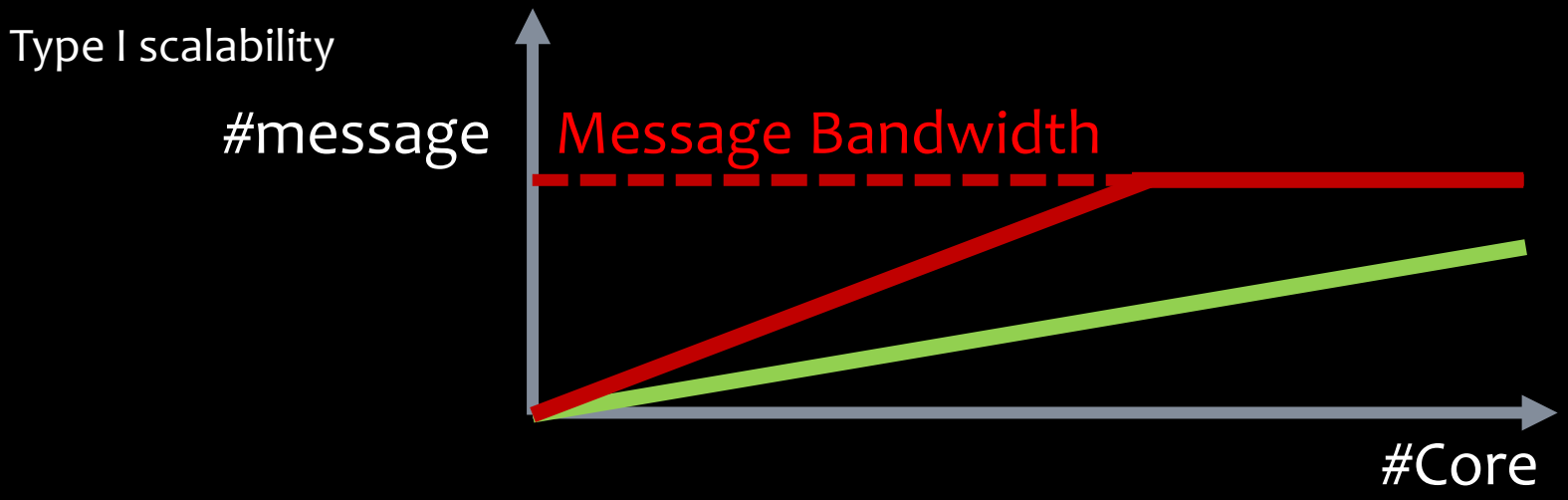
Type I scalability



Type II scalability

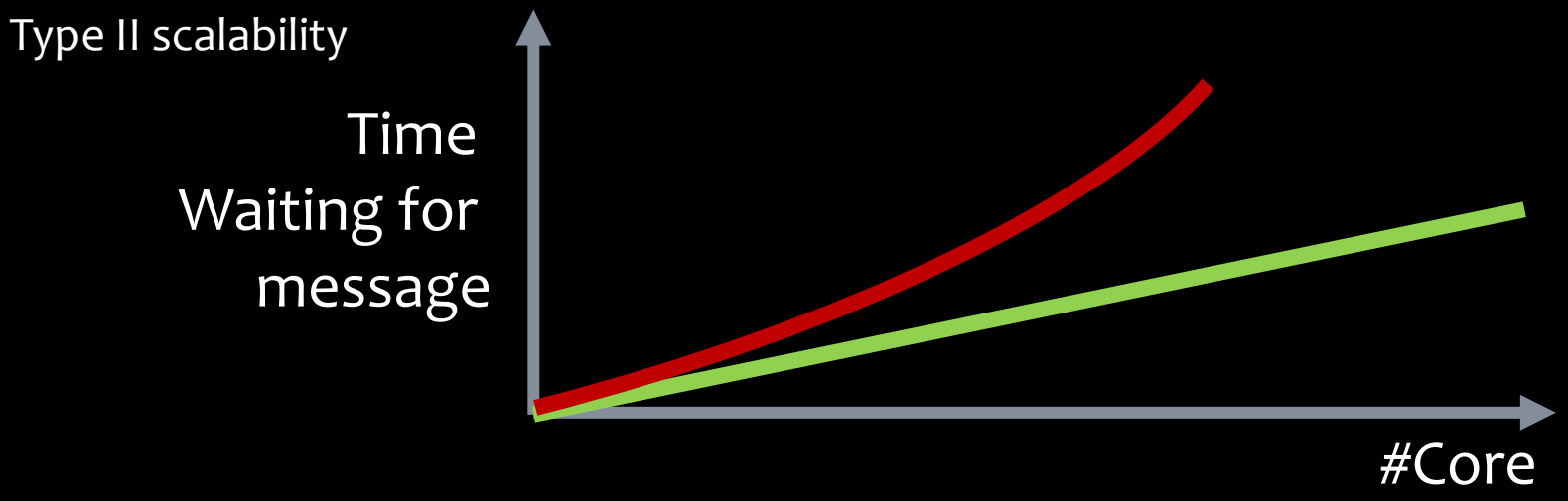


Example



- A program running on a NUMA machine:
  - Read a large file from disk
  - Launch 64 threads to process different part of the data

Example

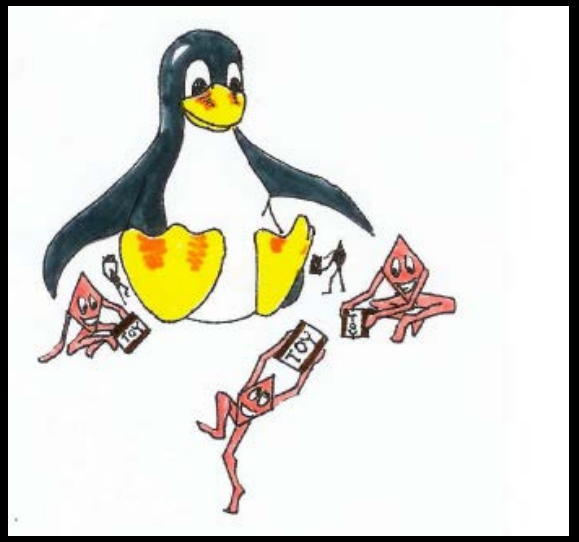


- A program contend on a single lock
- Diagnose:
  - Measure total waiting time to acquire this lock

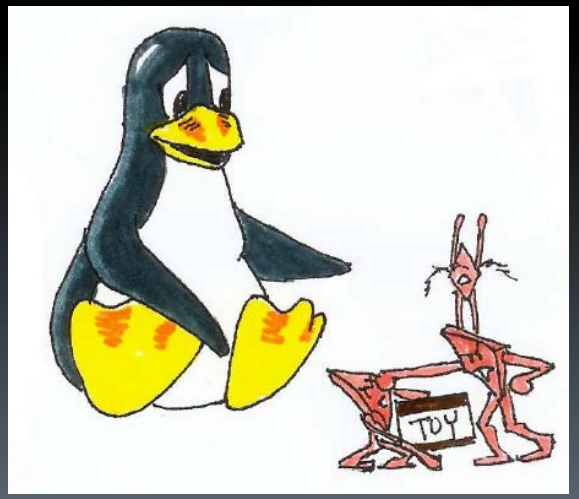
Scalability in ACTION

Parallel Programing Technique

- Partition
  - Data Partition
  - Resource Partition



- Parallel Access Control
  - Lock
  - Transactional Memory
  - RCU



Lock

- **Exclusive Locks**
- Reader-Writer Locks
- Multi-role Locks

Simplest Spinlock

- Lock
  - `while(!test_and_set(A));`
- Unlock
  - `A = 0;`

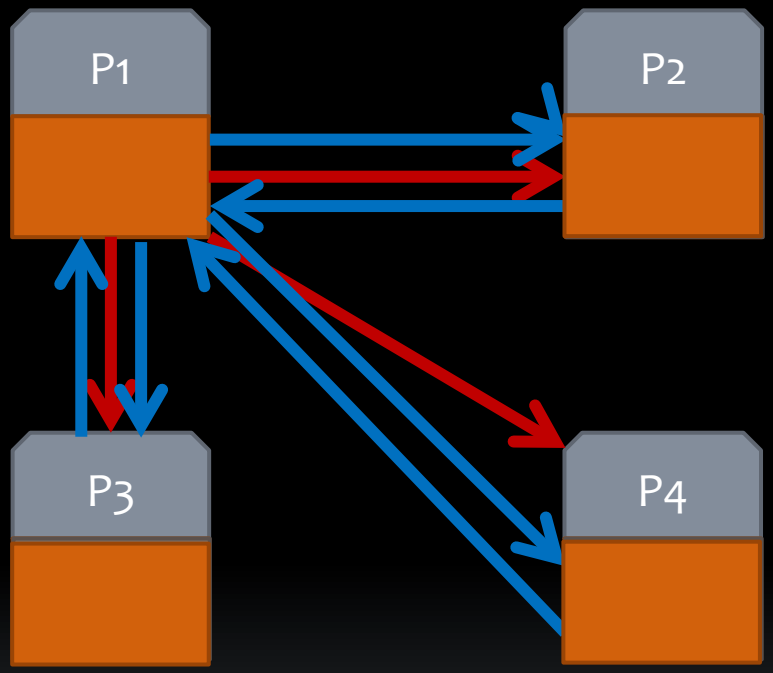
- Problem
  - Cache line Ping-Pong

Spin-on-read

- Lock
  - `while(!test_and_set(A))`
    - `while(!A);`
- Unlock
  - `A = 0;`

What is going on

Test_and_set !!
Failed!!!



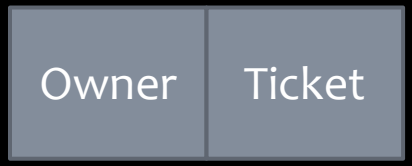
Problem: **TOO MUCH messages in system**

Possible solution :
Back off

- Lock
  - `while(!test_and_set(A))`
    - `while(!A) {`
      - `delay()`
    - `}`
- Unlock
  - `A = 0;`

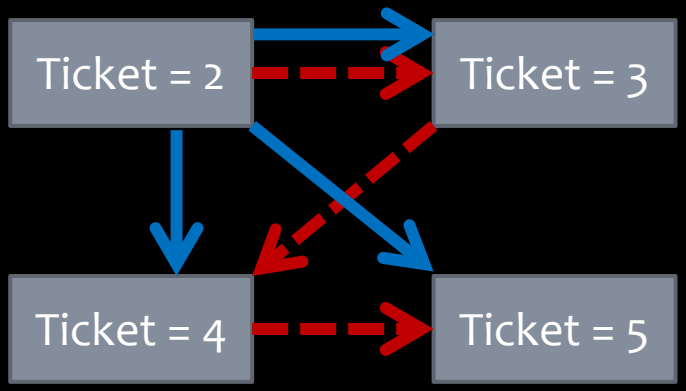
Fair locks: Ticket spin lock

- Lock
  - `MyTicket = fetch_and_add(&Ticket);`
  - `while(Owner != MyTicket);`
- Unlock
  - `Owner++;`



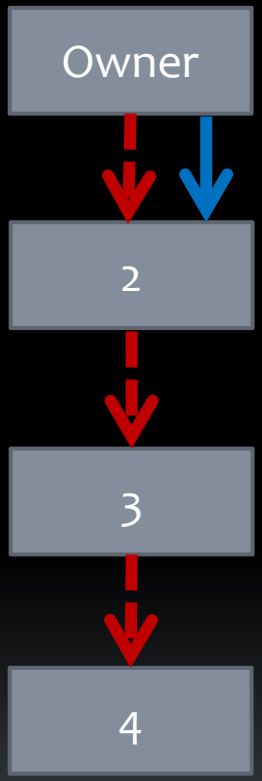
- Fair, but not scalable
- What happened?
  - Fairness guarantee itself is logical dependency !

Fair locks: Ticket spin lock (Cont.)



- Wait for message
  - New owner Messages are sent randomly without care about logical dependency

A solution: MCS lock

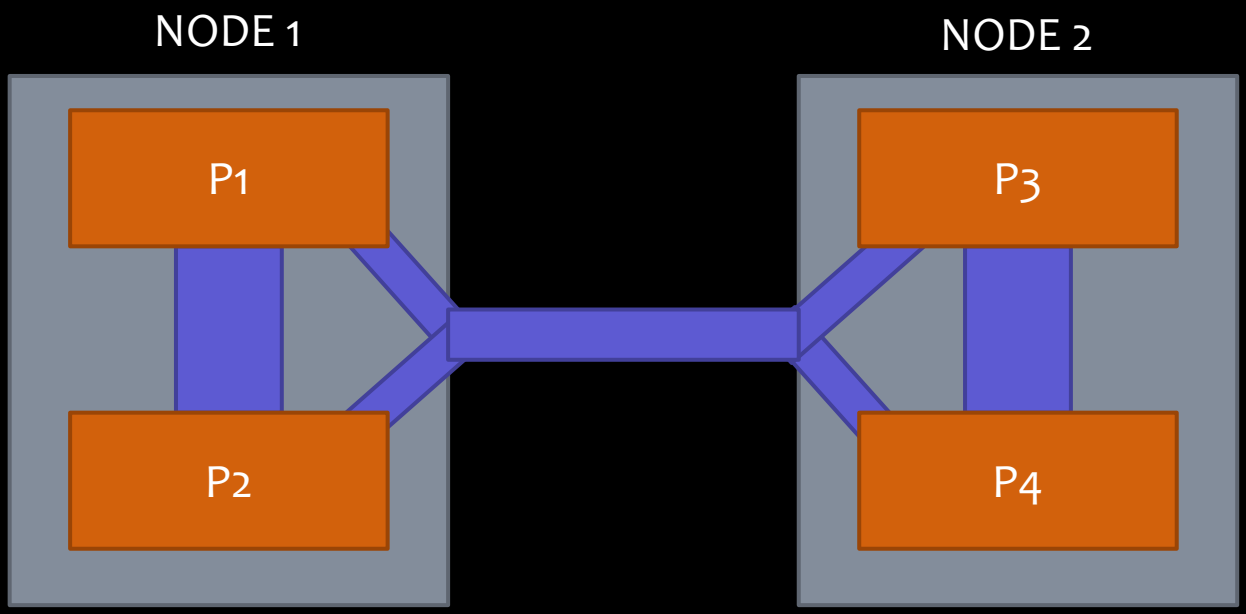


- Lock Failed:
  - Enqueue a node to wait list and spin on its own node
- Unlock:
  - Inform first one by writing on its node
- New owner Message is sent to next one in the list
- Greatly Reduced #message and time to wait messages

Similar locks:

- Anderson
- G. Graunke and S. Thakkar

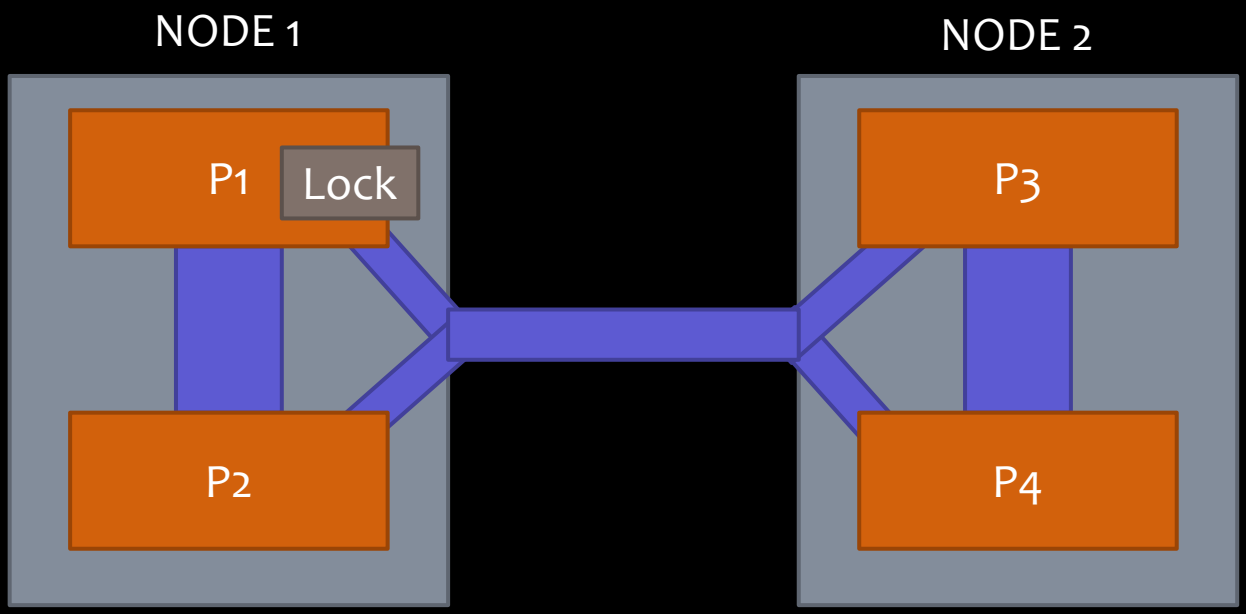
Lock on NUMA system



- Latency: Speed-of-Light on 1.8Ghz → 8cm
- Throughput: Link bandwidth between nodes are limited

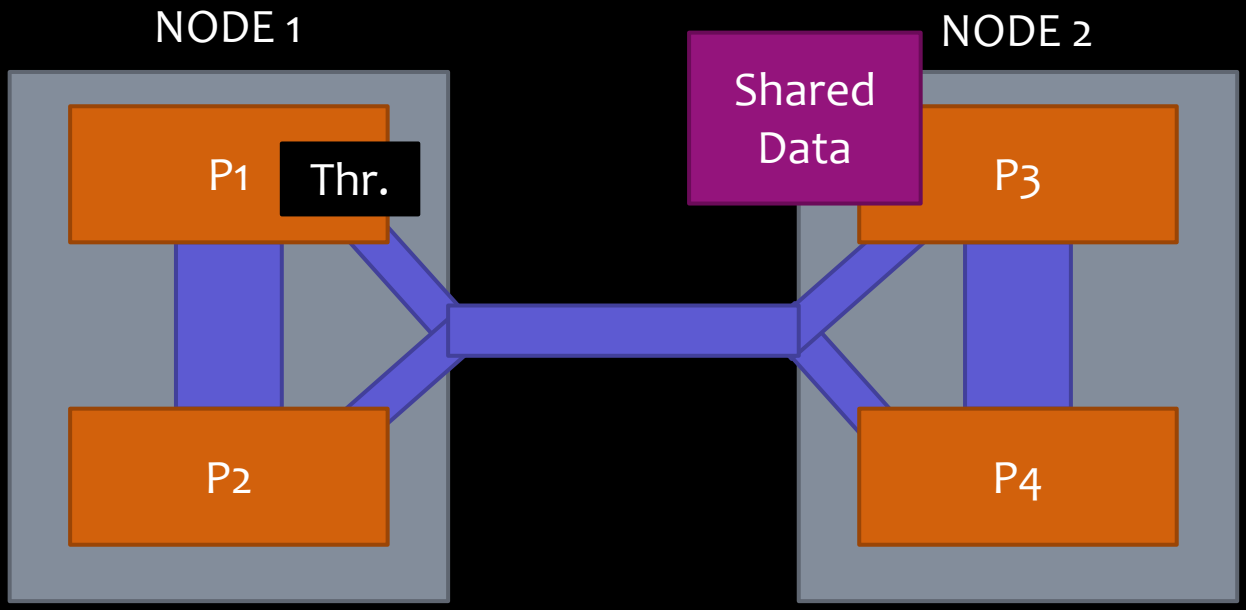
- **Idea: Avoid cross-node messages**

Cohort Lock



- Stay longer on one node
  - Lock acquisition messages
  - Shared data transfer messages

Remote Core Locking



- Migrate critical section
  - Eliminate shared data transfer messages

Atomic Overhead

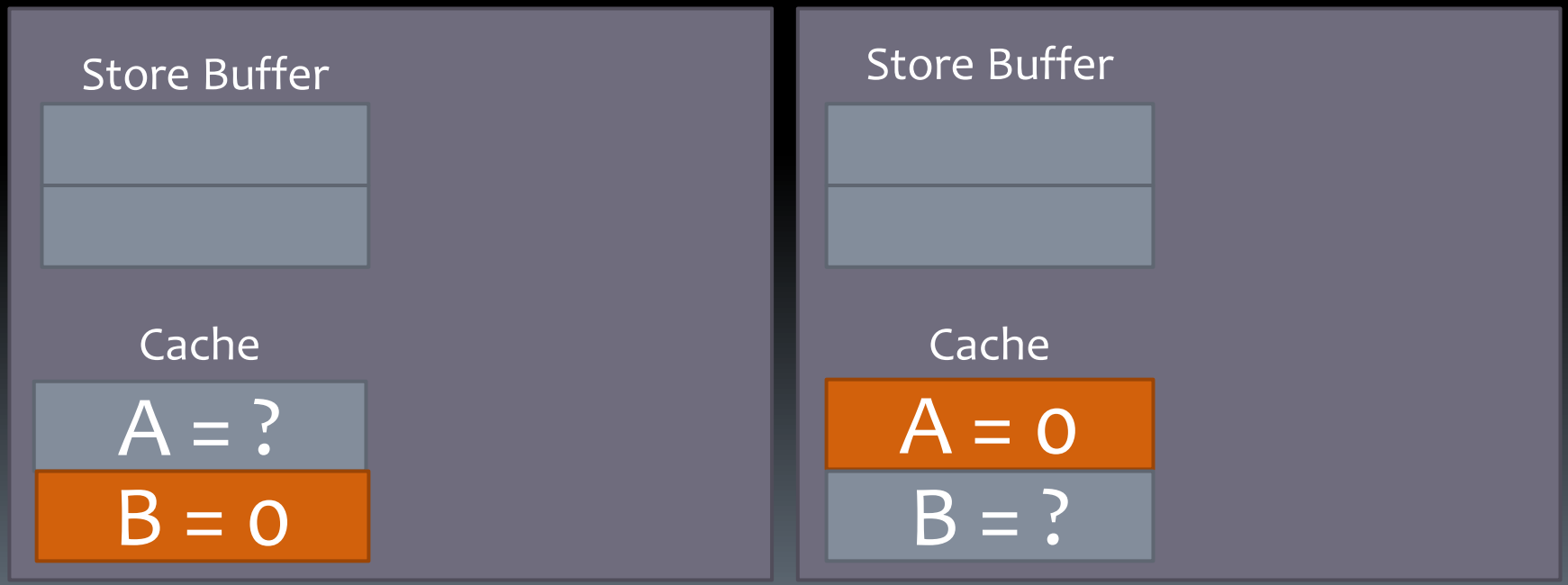
- Atomic operations involve lots of messages
  - Invalidate others' cache line
  - Other processors request for new value
  - …
- Example
  - Latency for a missed atomic inst.: 100cycle
  - Critical Section: 10cycle

10X overhead !!

Memory Barrier

- P1
  A = 1;
  B = 1;

- P2
  while (B == 0)
  continue;
  assert(A == 1);



Memory Barrier

- P1

  A = 1;

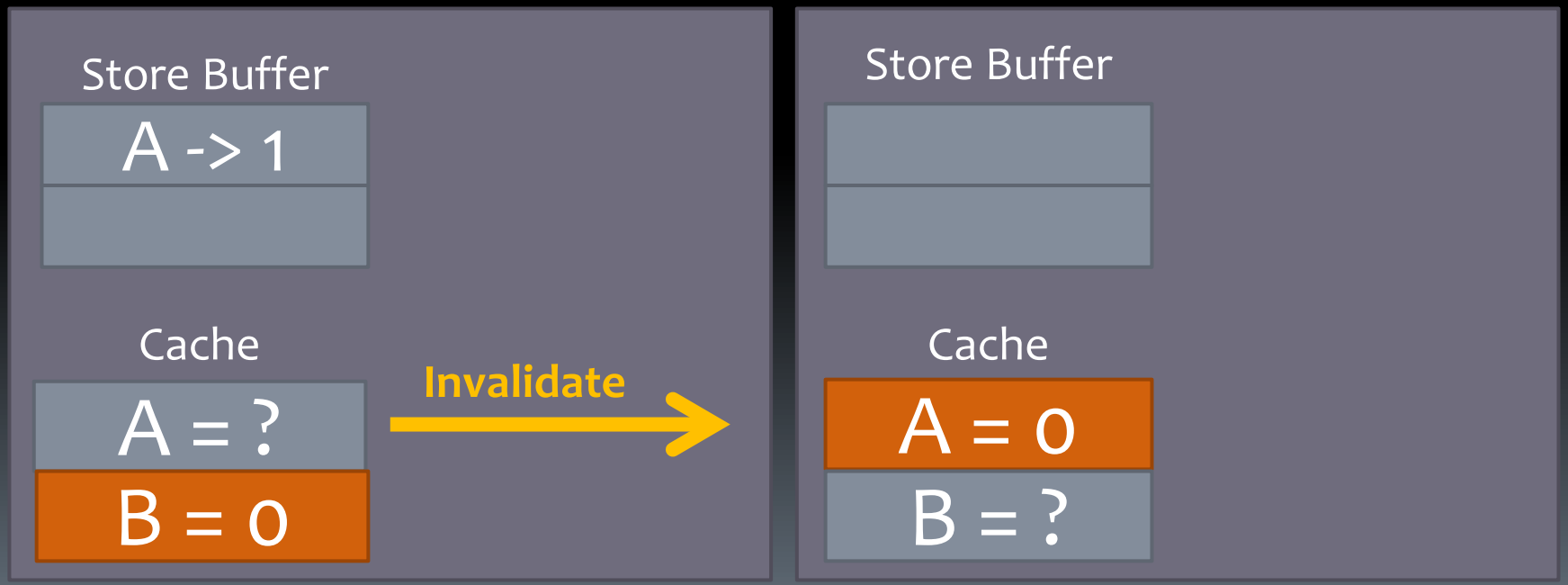
  B = 1;

- P2

  while (B == 0)

  continue;

  assert(A == 1);



Memory Barrier

- P1

  A = 1;

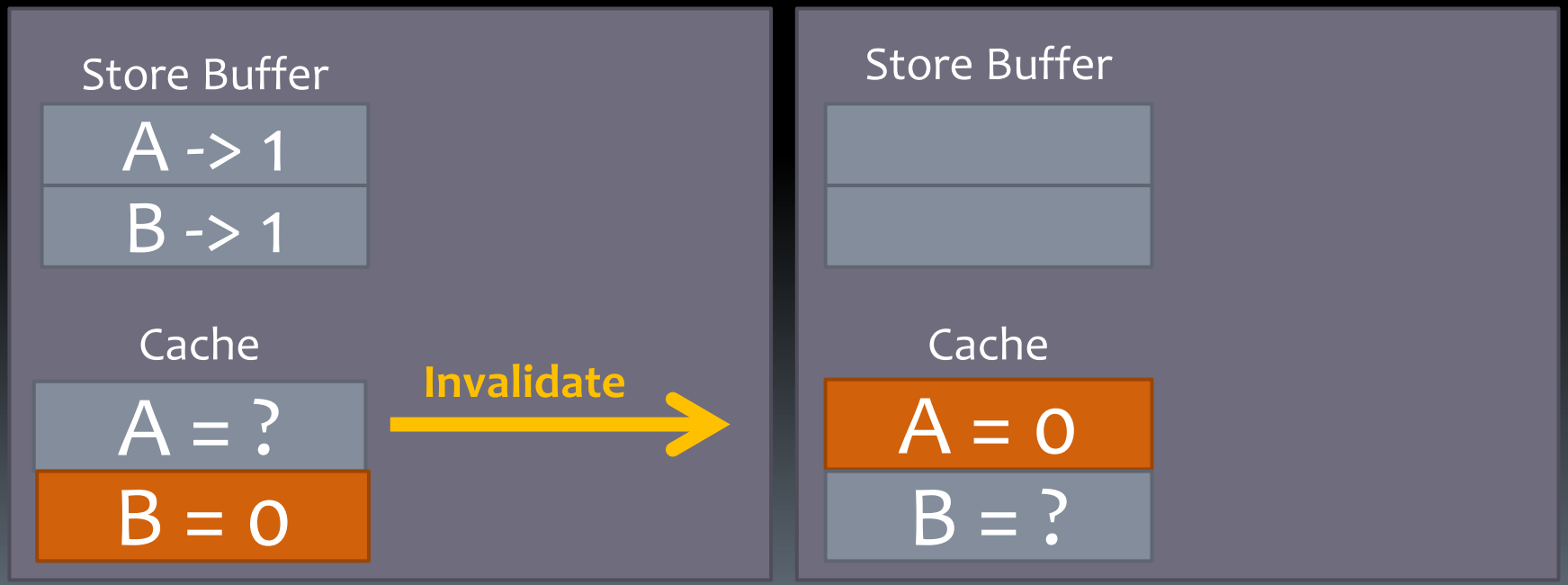
  B = 1;

- P2

  while (B == 0)

  continue;

  assert(A == 1);



Memory Barrier

- P1

  A = 1;

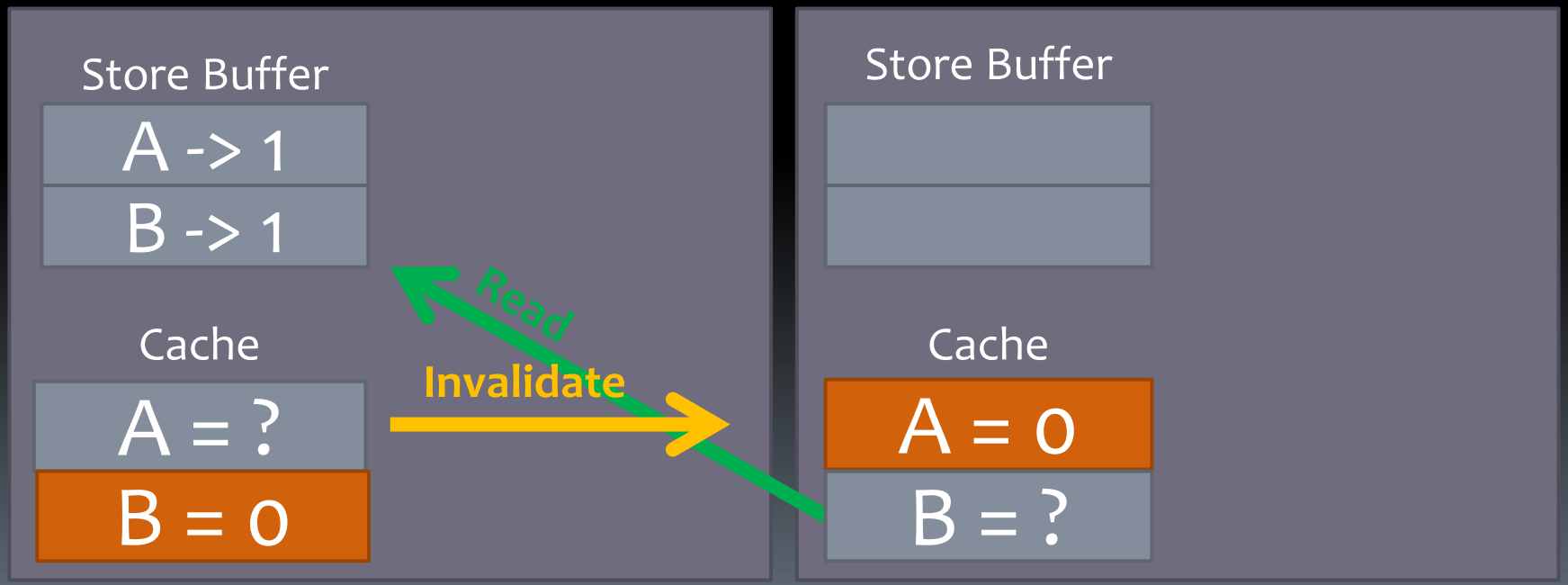
  B = 1;

- P2

  while (B == 0)

  continue;

  assert(A == 1);



Memory Barrier

- P1

  A = 1;

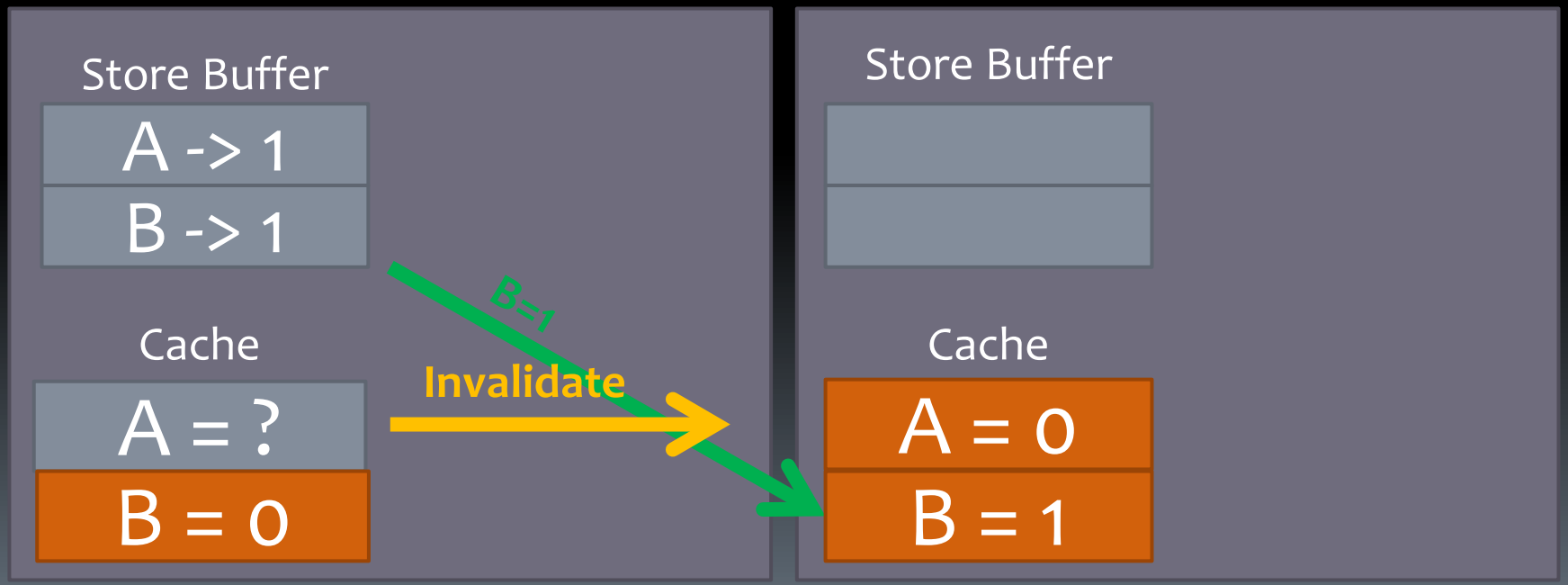
  B = 1;

- P2

  while (B == 0)

  continue;

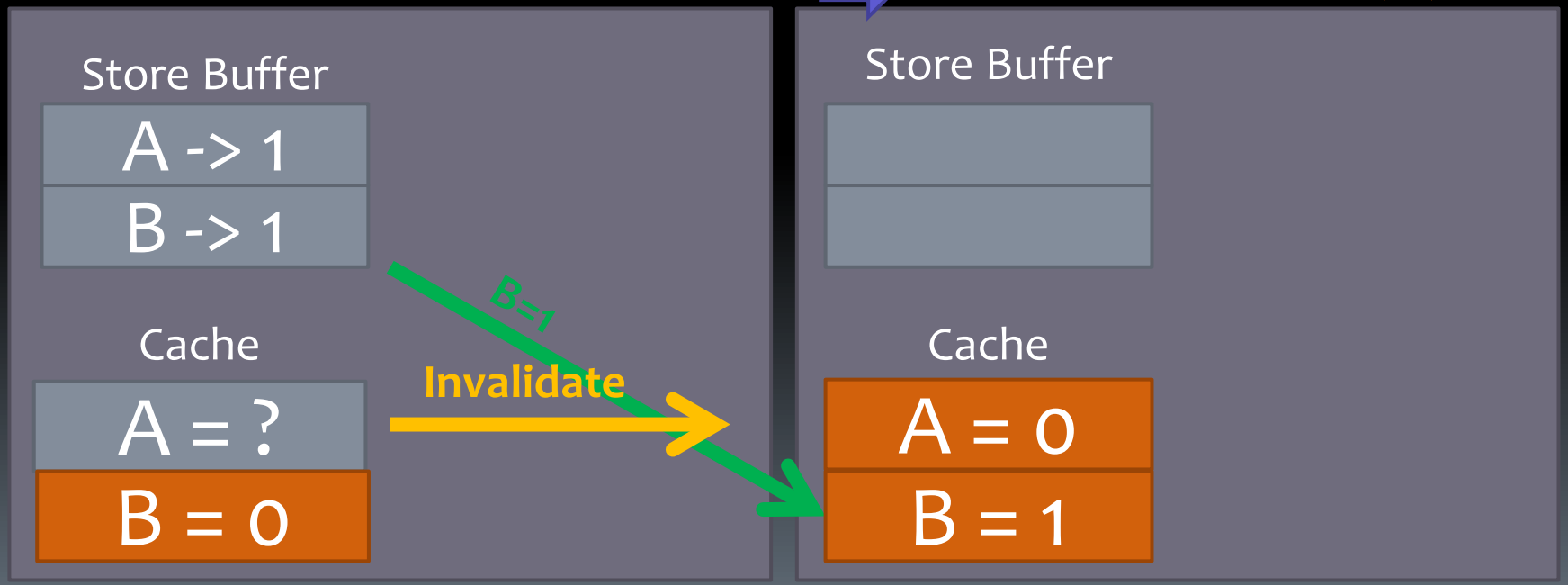
  assert(A == 1);



Memory Barrier

- P1
  A = 1;
  B = 1;

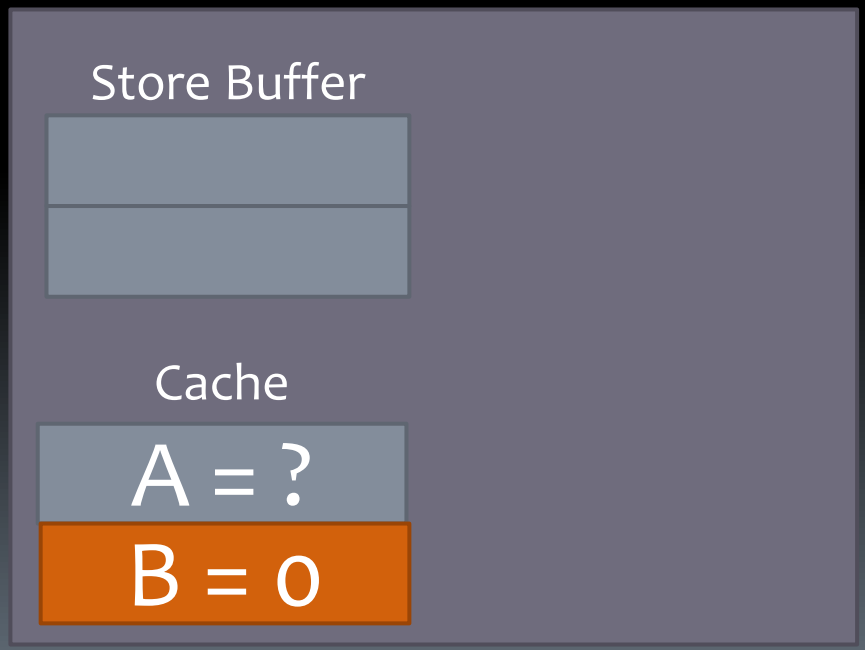
- P2
  while (B == 0)
  continue;
  assert(A == 1);

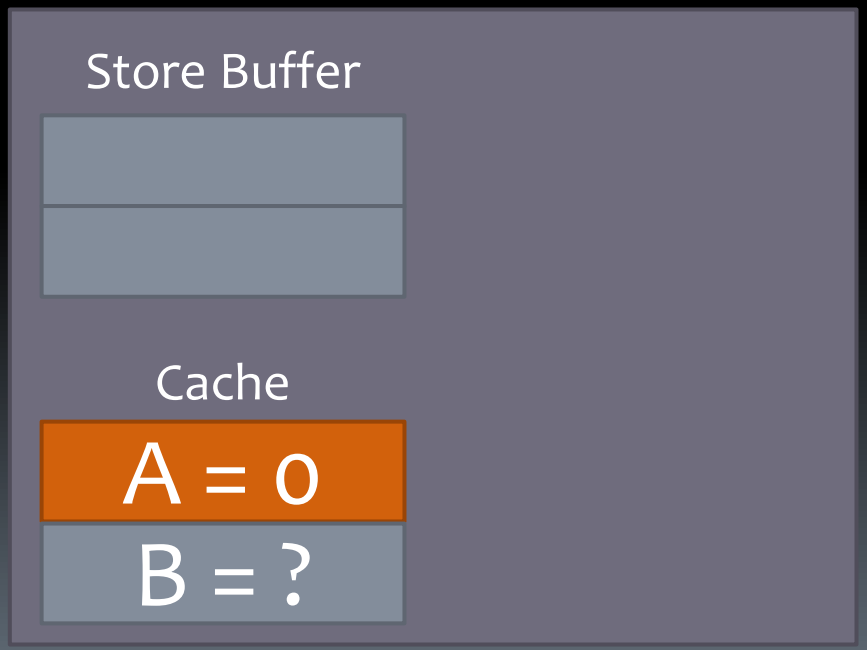


Solution

- P1
  A = 1;
  WMB();
  B=1;

- P2
  while (B == 0)
  continue;
  assert(A == 1);





Solution

- P1

  A = 1;

  WMB();

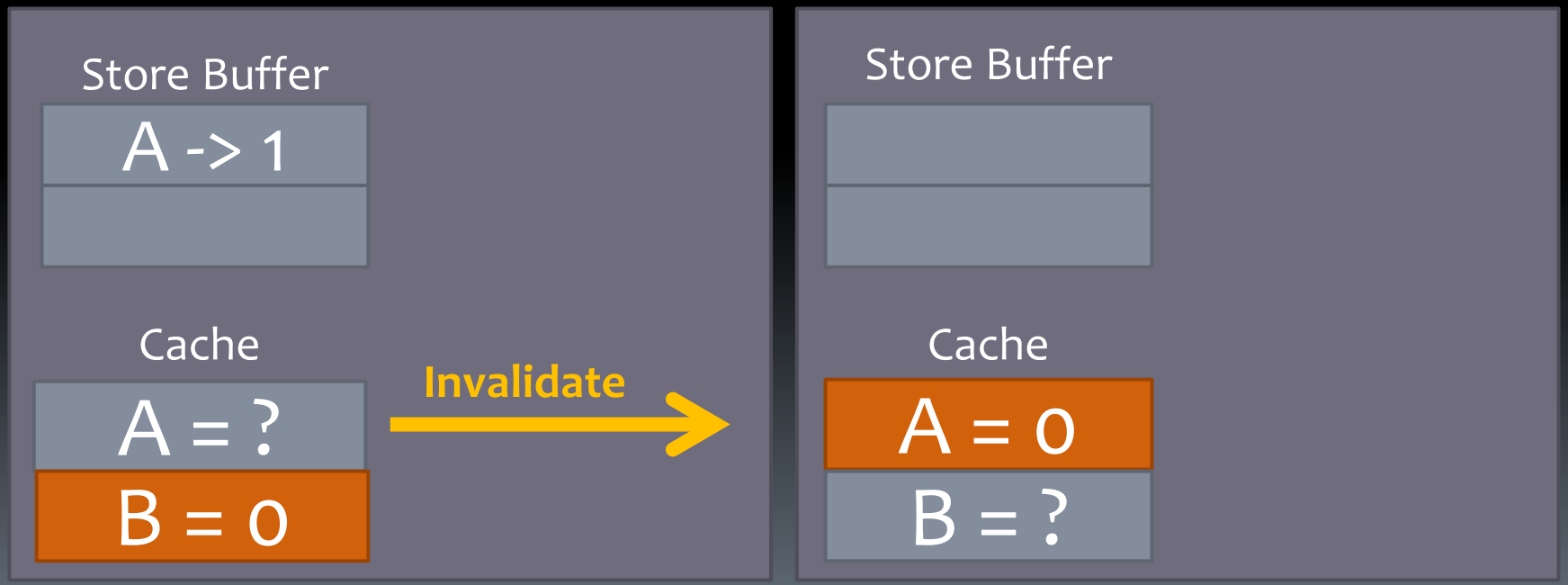
  B=1;

- P2

  while (B == 0)

  continue;

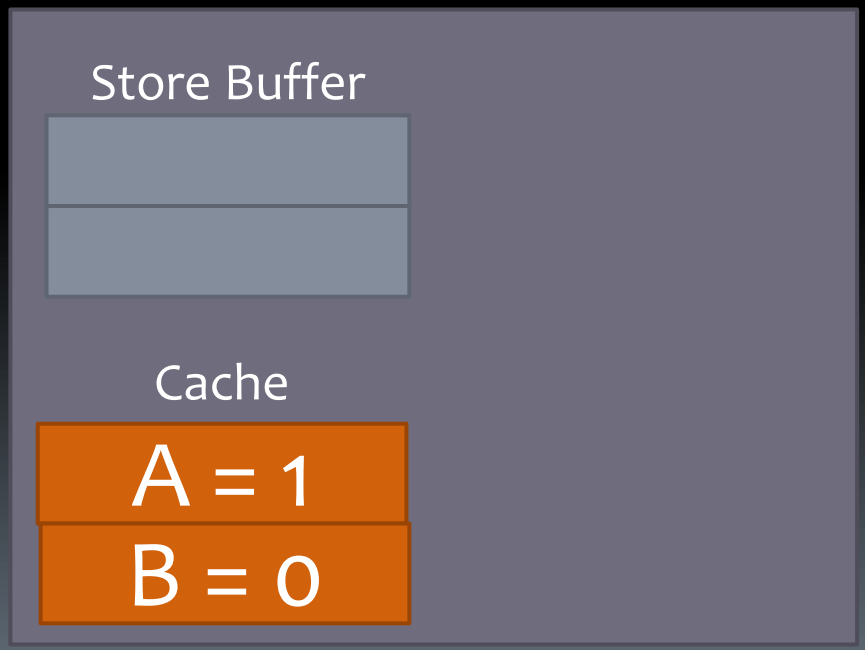
  assert(A == 1);

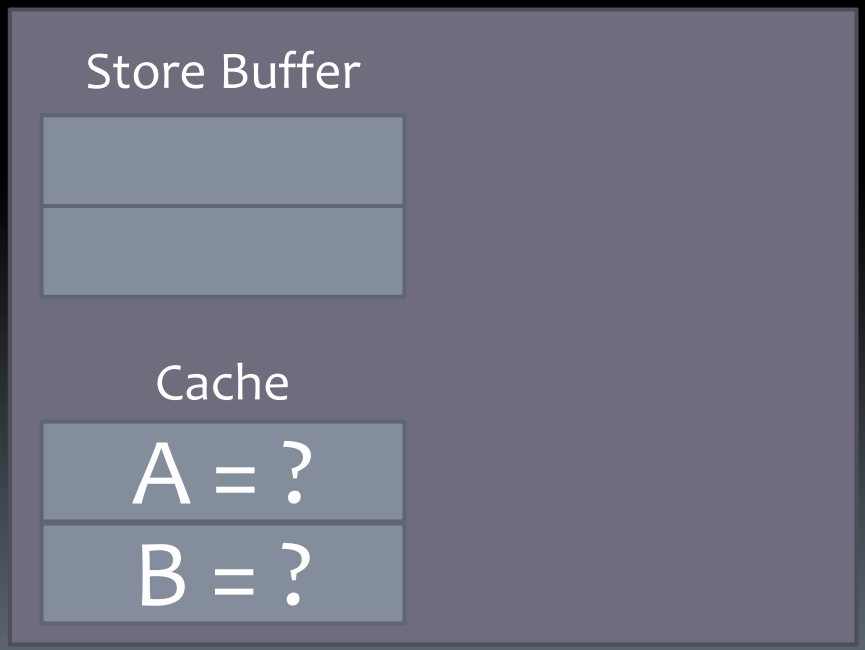


Solution

- P1
  A = 1;
  WMB();
  B=1;

- P2
  while (B == 0)
  continue;
  assert(A == 1);





Back to Lock

- P1

```
Lock();
    A = 1;
    B = 1;
Unlock();
```

- P2

```
Lock();
    if (B == 1)
        assert(A == 1);
Unlock();
```

- Critical section works because Atomic Inst. Implies a Memory Barrier
- Barrier -> Wait For Message

Solution: Biased Lock

- Lock
  - ```
    if (Lock_is_biased_to_me()) {
      return
    } else {
      Make it unbiased / biased to me
      Retry
    }
    ```

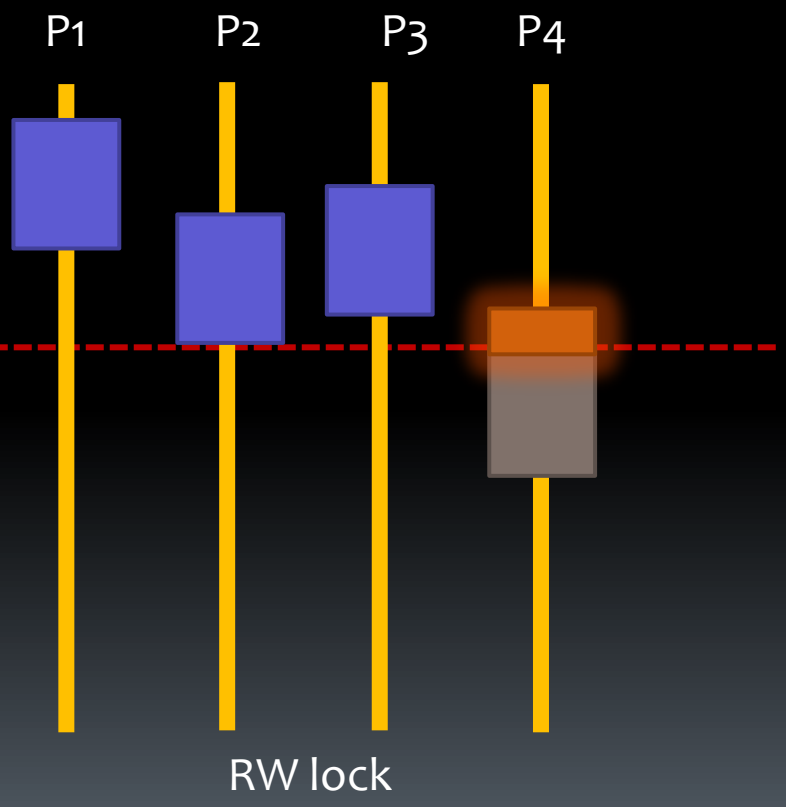
- Default lock implementation in Sun JVM
    ```

Lock

- Exclusive Locks
- **Reader-Writer Locks**
- Multi-role Locks

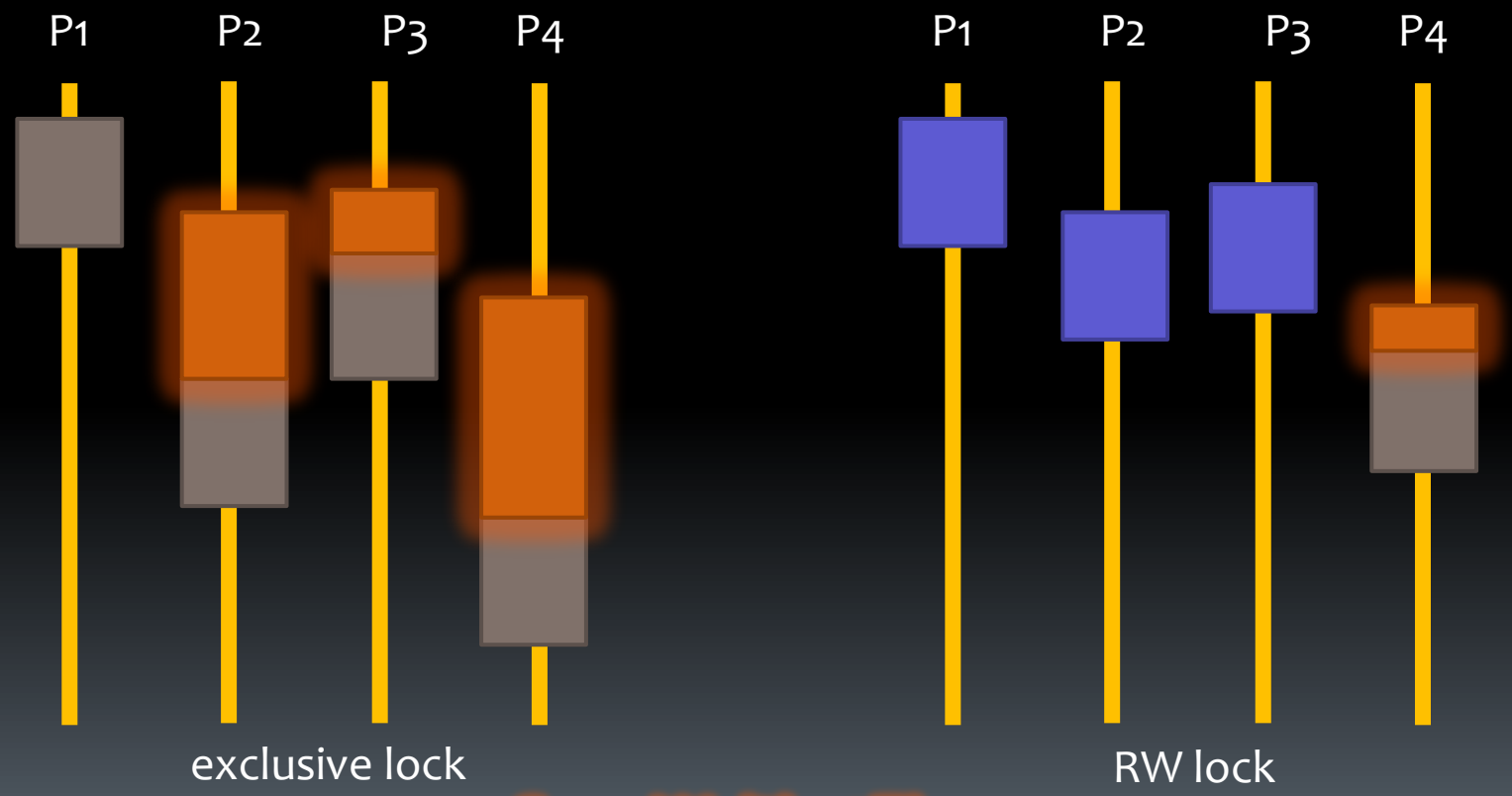
Reader-Writer Locks: senamtic

- Concurrent Readers
- Exclusive Writer



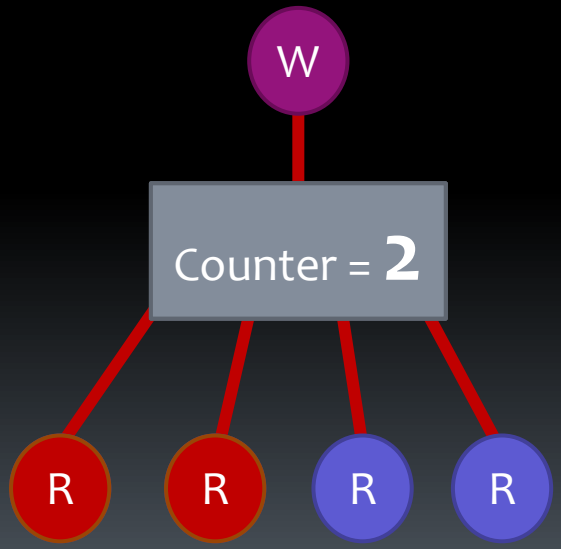
Why RW locks are considered more scalable over exclusive lock?

- Thinking in message passing

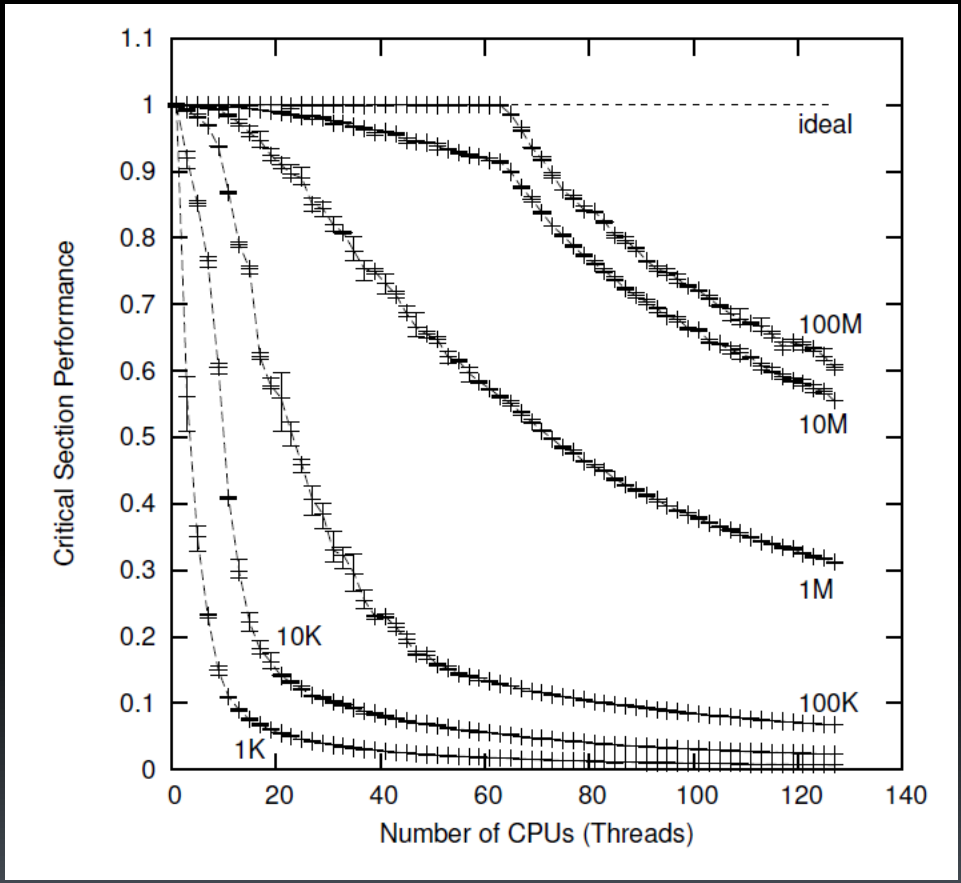


Traditional RW lock

- Reading
  - Counter -> #readers
- Writing
  - Counter = -1



Traditional RW lock performance



Problem

- Reader Lock Acquisition
  - add_and_fetch(&counter)

- Need to wait for message (current #reader)
- Need to send message (next reader)

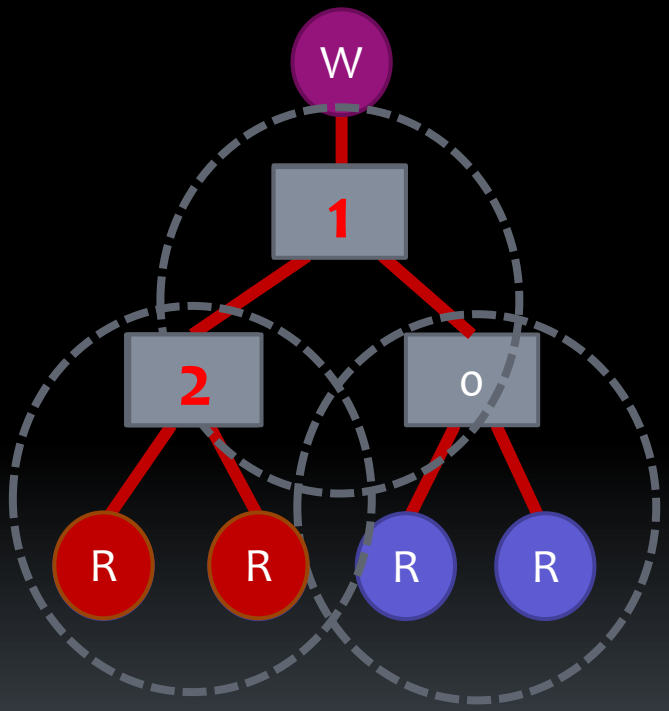
1. Acquisition serialized by messages!!
2. Occupies a great deal of message bandwidth!!

Solutions

- Idea 1
  - Reduce time to wait for message (GOLL)
- Idea 2
  - Reduce #message (BR lock)

GOLL

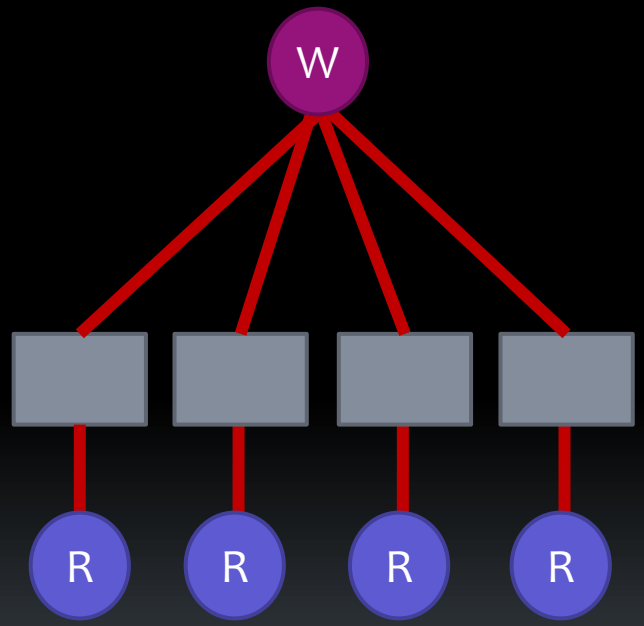
- Reduce time to wait for message



Waiting domain is split into pieces
Reduce #remote messages

BR lock

- Reduce #message



No reader messages if there's no writer

Other Parallel Access Control Technique

- RCU
- Transactional Memory

RCU -> Read-Copy-Update

- Reader

```
rcu_read_lock();
A = Shared_data;
// Read A …
rcu_read_unlock();
```

- Writer

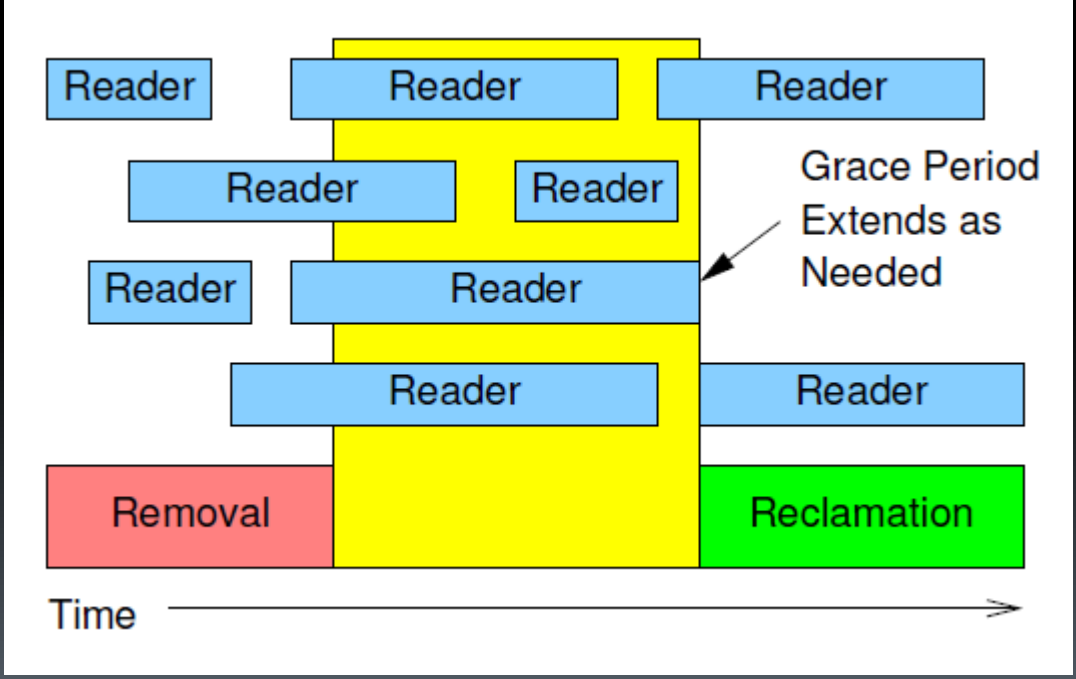
```
B = clone (Shared_data)
//update B
Old_data = Shared_data;
Shared_data = B;
synchronize_rcu();
free(A)
```

Disable Preemption in Reader

Wait others' Context Switch

Why RCU is scalable

- Eliminate most message for reader
- Bulk synchronization



Transactional Memory

- P1

```
Tx_begin();
    A = 1;
    B = 1;
Tx_end();
```

- P2

```
Tx_begin();
    if (B == 1)
        assert(A == 1);
Tx_end();
```

Software TM & Hardware TM

- Software TM
  - Lock
- Hardware TM
  - Add a flag to cache line accessed in transaction
  - Abort: Invalidate cache lines with flags
  - Commit: Remove flags

Why TM is more scalable

- Fine-grained synchronization
  - Relax logical dependency => Reduce time waiting for messages
- Avoid cache message on the lock word

Can locks be replaced by HTM?

- **No, because:**
  - Transaction-size limitations
  - Aborts and rollbacks
  - Lack of forward-progress guarantees
  - Semantic differences:
    - Consider empty critical section

Summary

- Introduction to scalability
  - Why parallel programming
  - Performance vs. Scalability
  - Scalability and Messages
- Scalability in ACTION
  - Locking and Scalability
  - Other Parallel Access Control Techniques