COL380

Introduction to
Parallel & Distributed Programming

Consensus

- Consensus number:
  - ➡ Number of threads that can reach consensus
  - ➡ Distributed, No pre-determined priority, Wait free
- Consensus number of atomic registers is 1
- Consensus number of CompareAndSwap is ∞

c = Consensus(v)

```
int CAS(int* address, int expect,  int newval):
    Atomically:
        old  = *address
        if(old == expect): *address = newval
        return old
```

Peterson's Algorithm

Initially want0 = false
want1 = false

Thread 0

1 want0 = true
2 turn = 1
3 4 while (want1 && turn == 1);

{// critical section
}

want0 = false

want1:false
Or turn:0

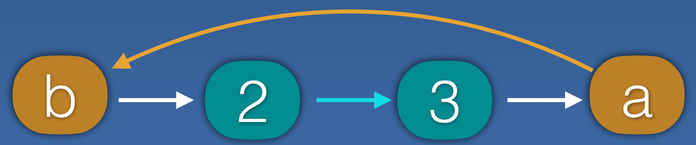
Thread 1

a want1 = true
b turn = 0
c d while (want0 && turn == 0);

{ // critical section
}

want1 = false

want0:false
Or turn:1



Bakery Algorithm

Initially want[0..k] = false
token[0..k] = 0

Thread "ID"

want[ID] = true
token[ID] = 1 + max(token)

while (∃ other s.t. want[other] && token[other] "<" token[ID];
{// critical section
}

want[ID] = false

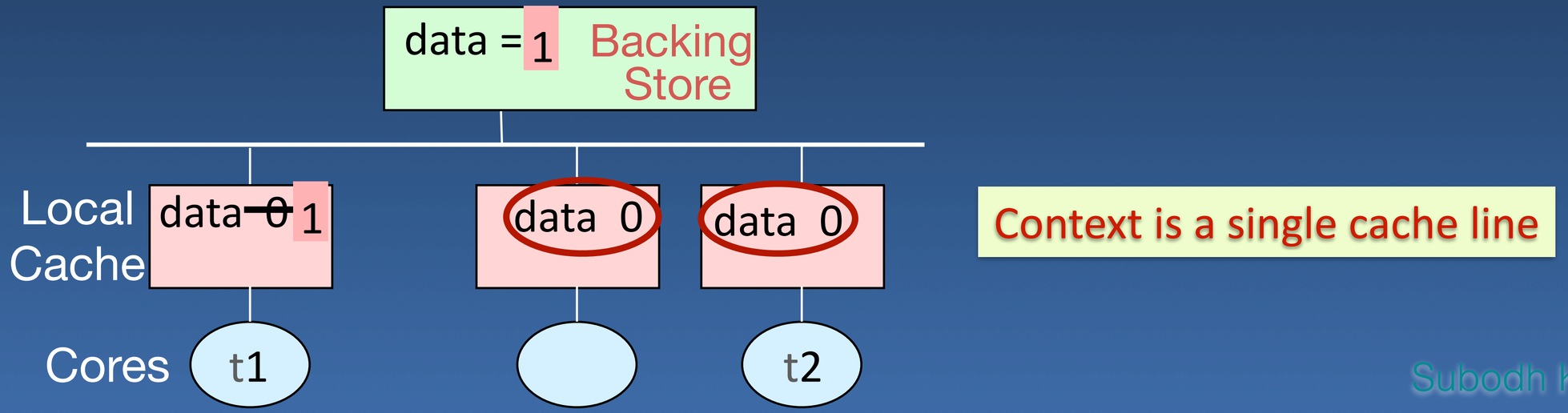
Cache Coherence

- Different copies of ‘same location’ may not have the same value
  - ➡ thread1 and thread2 both have cached copies of data
- t1 writes data=1
  - ➡ But may not “write through” to memory
- t2 reads data, but gets the “stale” copy
  - ➡ This may happen even if t2 read an updated value of another variable

Disallow conflict

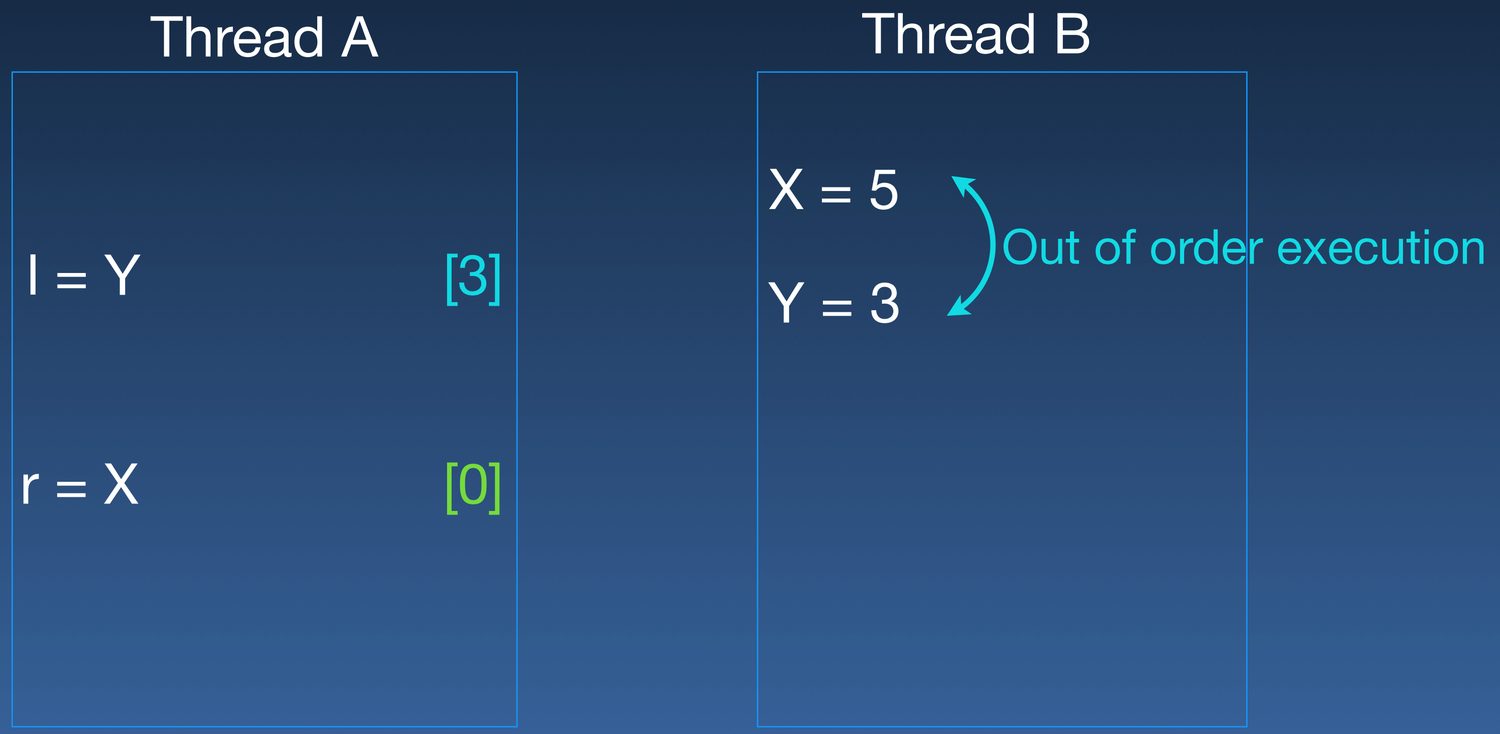
Invalidate first

Monitor access



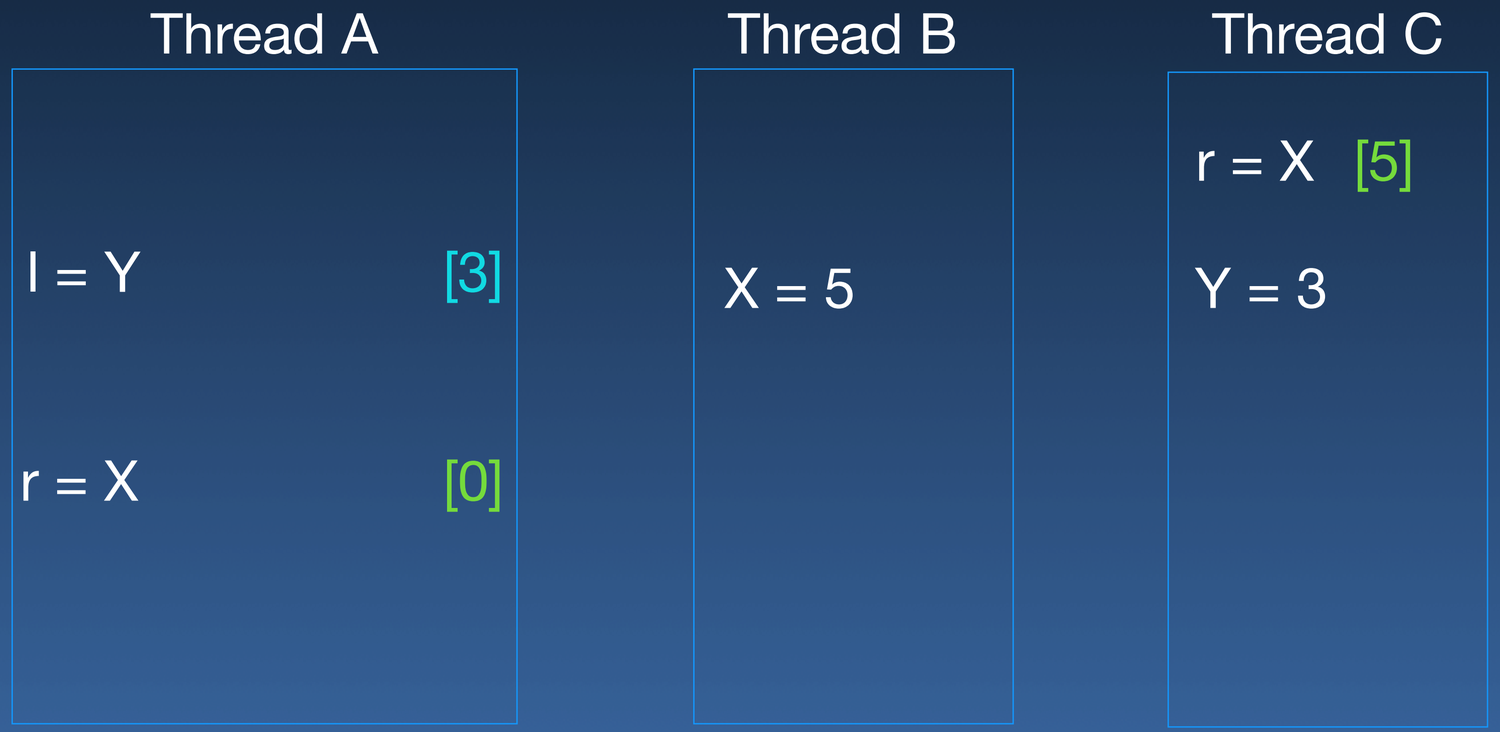
Coherence *vs* Consistency

Initially: X = 0; Y = 0;



Coherence *vs* Consistency

Initially: X = 0; Y = 0;



Memory Consistency

- Consistency is about global state (not per-variable)
- Compiler sees only local memory dependencies in reordering code
  - ➡ Can allocate a register or stack
- Architecture batches memory operations out of order
  - ➡ X=1, Y=1, X=2
  - ➡ Second write to X may happen before Y's
  - ➡ 1st write may never happen
- The network can also reorder two write messages
- Need support from hardware, compiler ..
- Guide compiler, Use synchronization, Manage race