

Concurrency: Multi-core Programming & Data Processing

Barrier Synchronization



Prof. P. Felber

Pascal.Felber@unine.ch

<http://iiun.unine.ch/>

Based on slides by Maurice Herlihy and Nir Shavit

Simple Video Game

- Prepare frame for display
 - By graphics coprocessor

- “Soft real-time” application
 - Need at least 35 frames/second
 - OK to mess up rarely

Simple Video Game

```
while (true) {  
    frame.prepare();  
    frame.display();  
}
```

- What about overlapping work?
 - 1st thread displays frame
 - 2nd prepares next frame

Two-Phase Rendering

1st thread

```

while (true) {
  if (phase) { Even phases
    frame[0].display();
  } else {
    frame[1].display();
  }
  Odd phases
  phase = !phase;
}
  
```

2nd thread

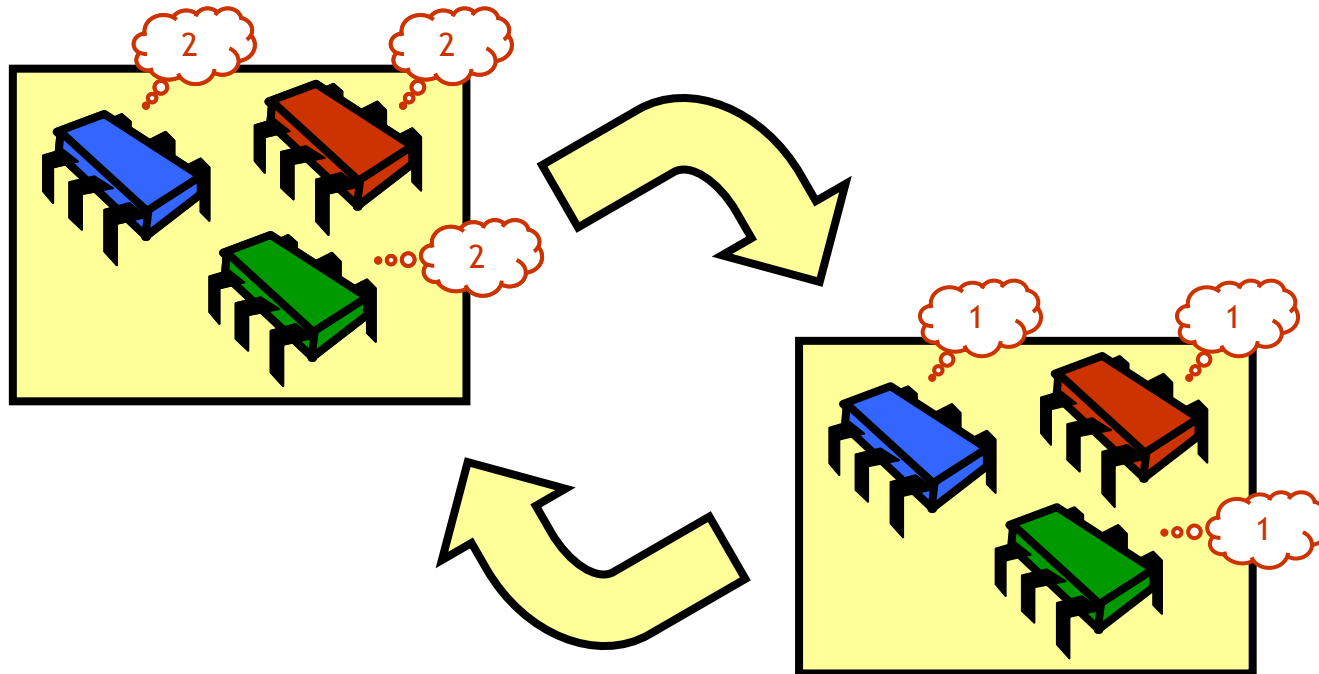
```

while (true) {
  if (phase) {
    frame[1].prepare();
  } else {
    frame[0].prepare();
  }
  phase = !phase;
}
  
```

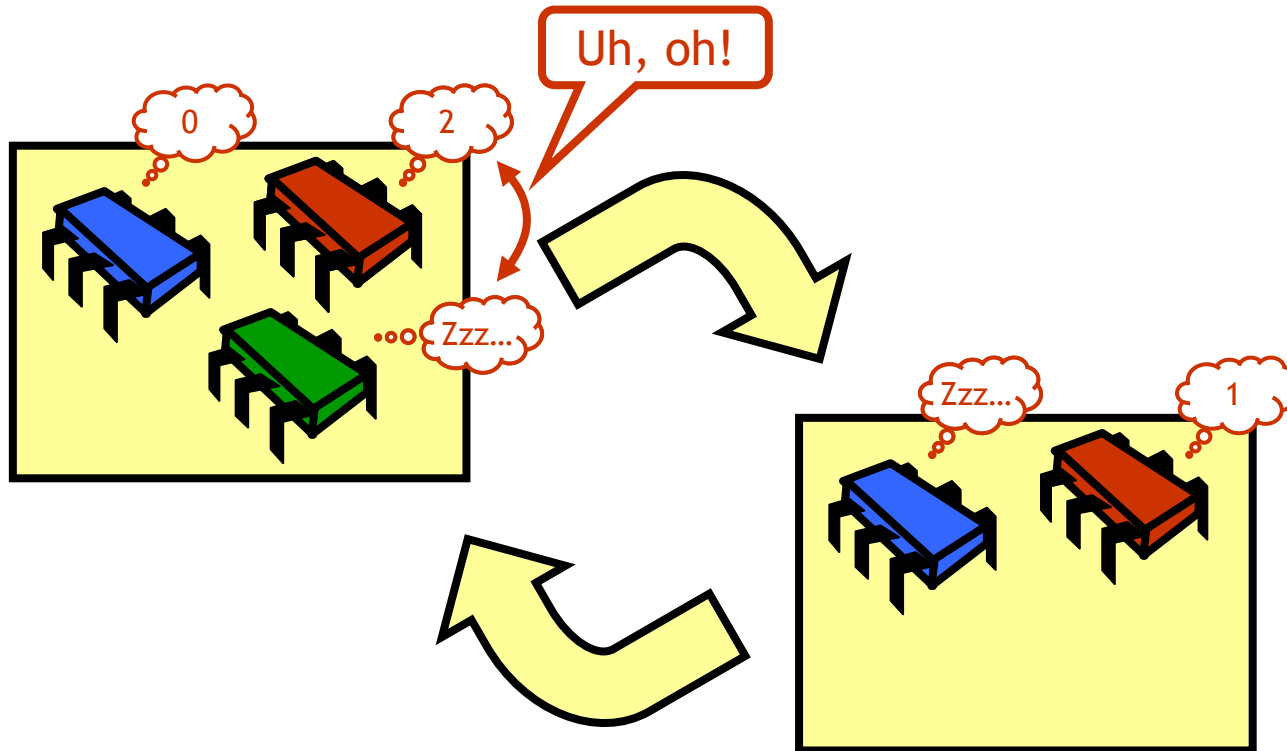
Synchronization Problems

- How do threads stay in phase?
- Too early?
 - Display frame before it is fully rendered
- Too late?
 - Recycle memory before frame is displayed

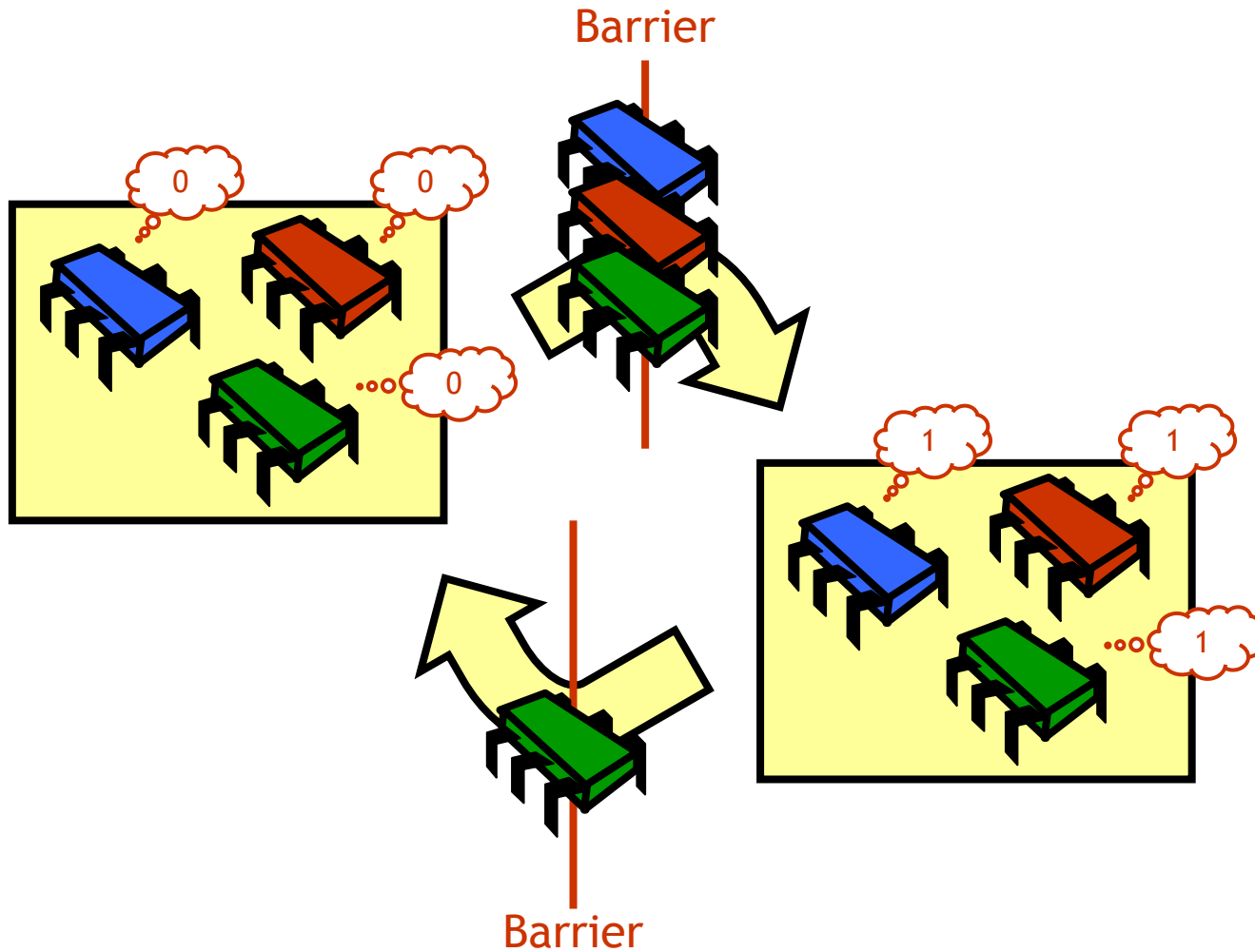
Ideal Parallel Computation



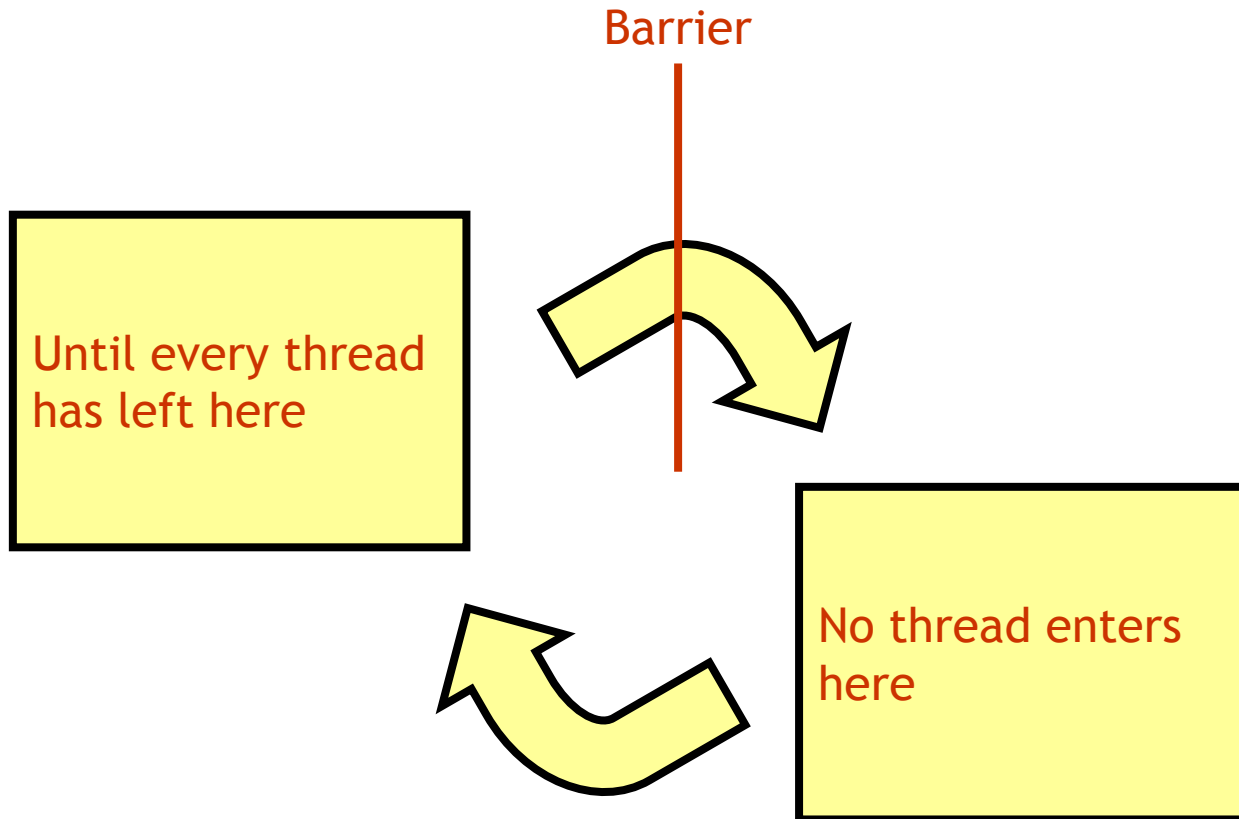
Real-Life Parallel Computation



Barrier Synchronization



Barrier Synchronization



Why Do We Care?

- Mostly of interest to
 - Scientific and numeric computation

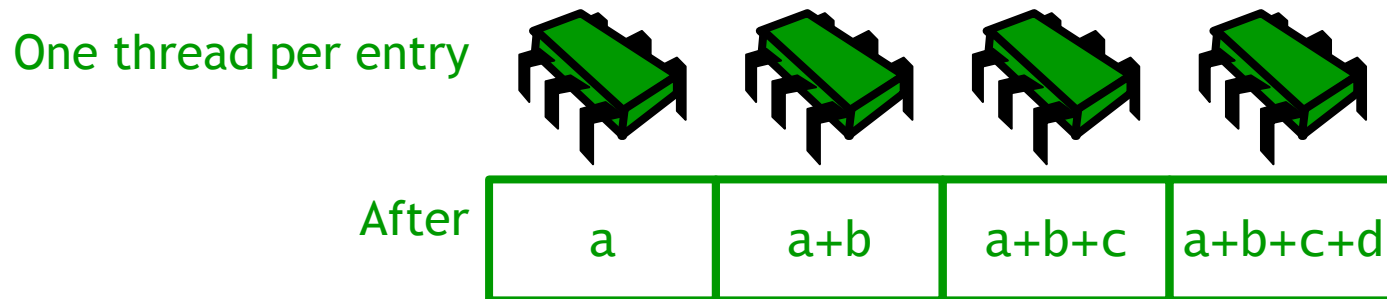
- Elsewhere
 - Garbage collection
 - Less common in systems programming
 - Still important topic

Duality

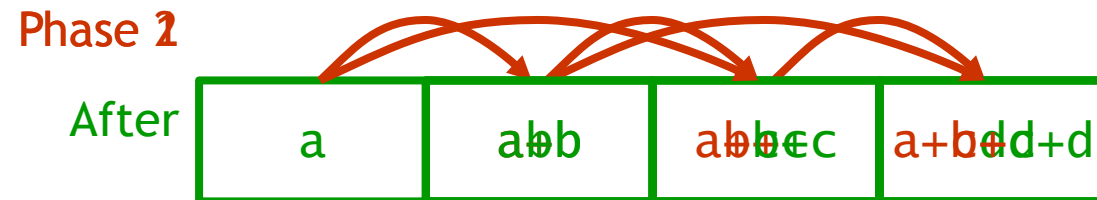
- Dual to mutual exclusion
 - Include others, not exclude them
 - Same implementation issues

- Interaction with caches
 - Invalidation?
 - Local spinning?

Example: Parallel Prefix



Parallel Prefix: Phases



Parallel Prefix

- N threads can compute
 - Parallel prefix
 - Of N entries
 - In $\log_2 N$ rounds




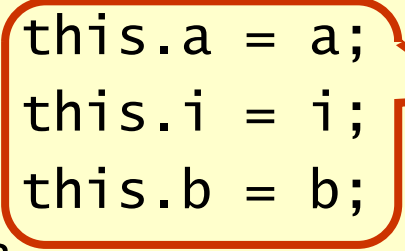
- What if system is asynchronous?
 - That is why we need barriers!

Parallel Prefix

```



class Prefix extends Thread {
  private int[] a;
  private int i;
  private Barrier b;
  public Prefix(int[] a, int i, Barrier b) {
    this.a = a;
    this.i = i;
    this.b = b;
  }
  ...
}

```

 **Array of input values**
 **Thread index**
 **Shared barrier**
 **Initialize fields**

Where Do Barriers Go?

```

public void run() {
    int d = 1, sum = 0;
    while (d < N) {
        if (i >= d)
            sum = a[i-d];
        b.await();  Make sure everyone reads
        if (i >= d) before anyone writes
        a[i] += sum;
        b.await();  Make sure everyone writes
        d = d * 2; before anyone reads
    }
}
  
```


Barrier Implementations

- Cache coherence
 - Spin on locally-cached locations?
 - Spin on statically-defined locations?
- Latency
 - How many steps?
- Symmetry
 - Do all threads do the same thing?

Barriers

```

public class Barrier {
  AtomicInteger count;
  int size;
  public Barrier(int n) {
    count = new AtomicInteger(size = n);
  }
  public void await() {
    if (count.getAndDecrement() == 1)
      count.set(size);
    else
      while (count.get() != 0) { }
  }
}
  
```

Number of threads not yet arrived

Number of threads participating

Initialization

Wait on barrier

If I am last, reset fields for next time

Otherwise, wait for everyone else

What's wrong with this protocol?

Reuse

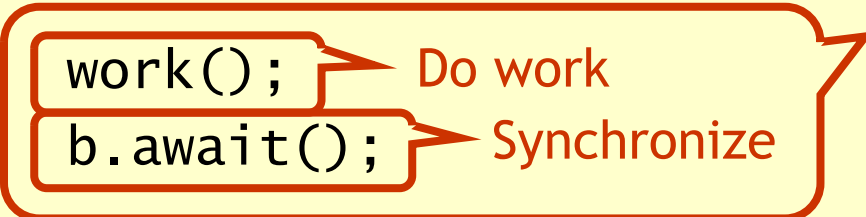
```

{
  ...
  Barrier b = new Barrier(n);
  while (mumble()) {
    work();
    b.await();
  }
}
  
```

Repeat

Do work

Synchronize



Barriers

```

public class Barrier {
    AtomicInteger count;
    int size;
    public Barrier(int n) {
        count = new AtomicInteger(n);
    }
    public void await() {
        if (count.getAndDecrement() == 1)
            count.set(size);
        else
            while (count.get() != 0) { }
    }
}
  
```

Prepare for
phase 2

Waiting
for phase
2 to finish

Phase 1
is over

Waiting
for phase
1 to finish

Zzz...

Basic Problem

- One thread “wraps around” to start phase 2...

...while another thread is still waiting for phase 1

- Solutions
 - Always use two barriers
 - Sense-reversing barrier

Sense-Reversing Barriers

```

public class SenseBarrier {
    AtomicInteger count;
    int size;
    volatile boolean sense = true;
    public void await(boolean mySense) {
        if (count.getAndDecrement() == 1) {
            count.set(size);
            sense = mySense
        } else
            while (sense != mySense) { }
    }
}
  
```

Completed odd or even phase?

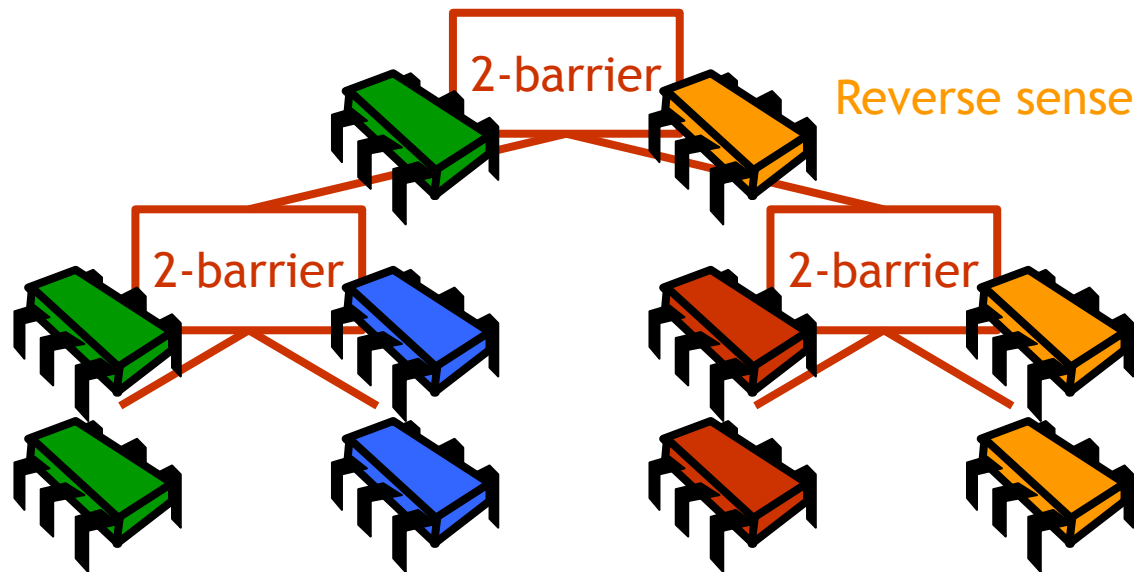
Thread working on odd or even phase?

If last, reverse sense for next time

Otherwise, wait for sense to flip

Combining Tree Barriers

- Goal: decrease contention
 - Processors at leaves
 - Nodes are n -way barriers (in different cache lines)



Combining Tree Barriers

```

public class TreeBarrier {
    AtomicInteger count;
    int size;
    TreeBarrier parent;
    public void await(boolean mySense) {
        if (count.getAndDecrement() == 1) {
            if (parent != null)
                parent.await(mySense);
            count.set(size);
            sense = mySense;
        } else while (sense != mySense) { }
    }
}

```

Parent barrier in tree

Thread working on odd or even phase?

Am I last?

Proceed to parent barrier

Reset barrier and notify others at this node

I am not last, so wait for notification

Combining Tree Barriers

- No sequential bottleneck
 - Parallel `getAndDecrement()` calls
- Low memory contention
 - Same reason
- Cache behavior
 - Everyone spins on `sense` field
 - Local spinning on bus-based architecture
 - Not so good for NUMA

Summary

- Basic barrier must be used carefully
- Sense-reversing
 - Reuse without reinitializing
- Combining tree
 - Less contention, more scalable
- Other barriers
 - Tournament tree
 - Dissemination barrier...