Barrier Synchronization

Companion slides for
The Art of Multiprocessor Programming
by Maurice Herlihy & 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();
  }
```

Simple Video Game

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

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

Two-Phase Rendering

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

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

Two-Phase Rendering

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

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

Even phases

Two-Phase Rendering

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

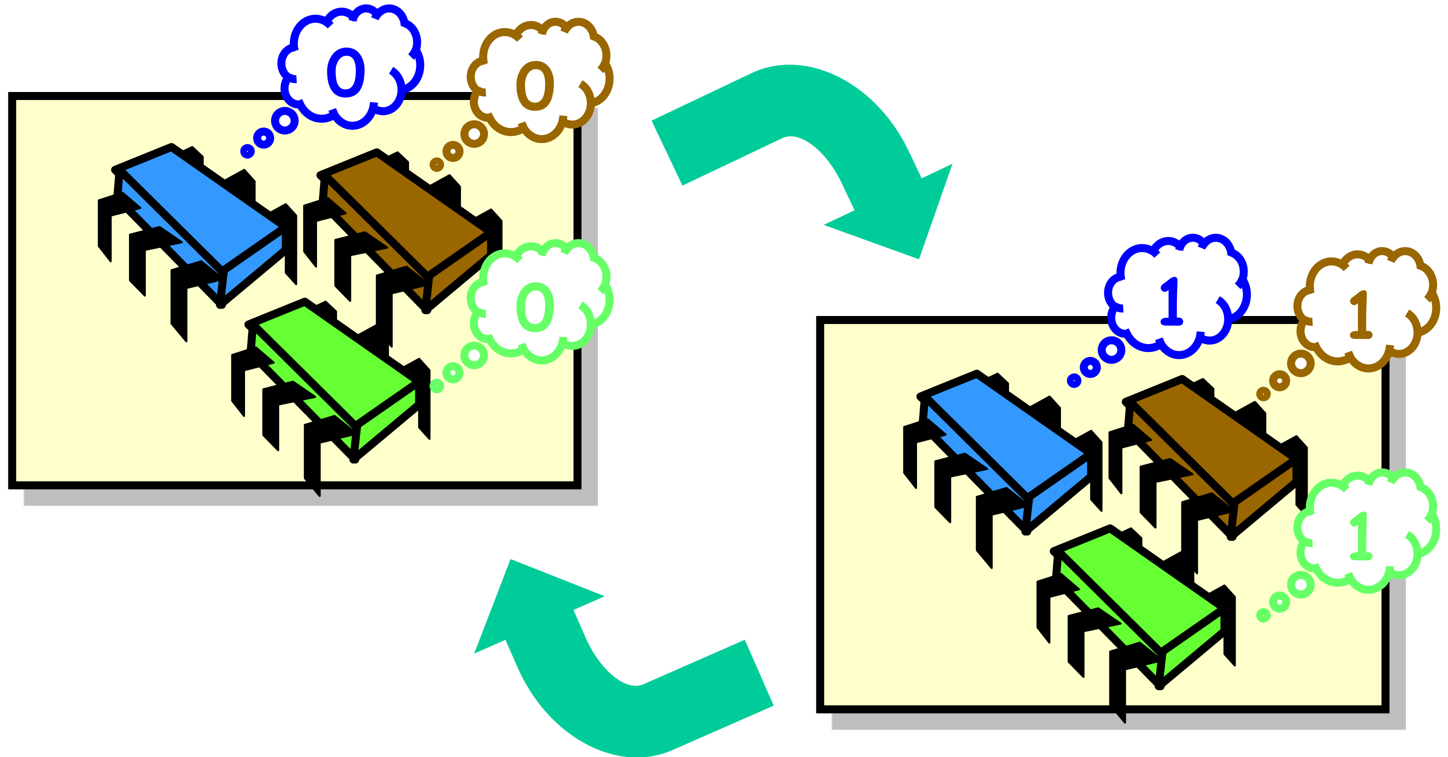
```
while (true) {
  if (phase) {
    frame[1].prepare();
  } else {
    frame[0].prepare();
  }
  phase = !phase;
}
```

odd phases

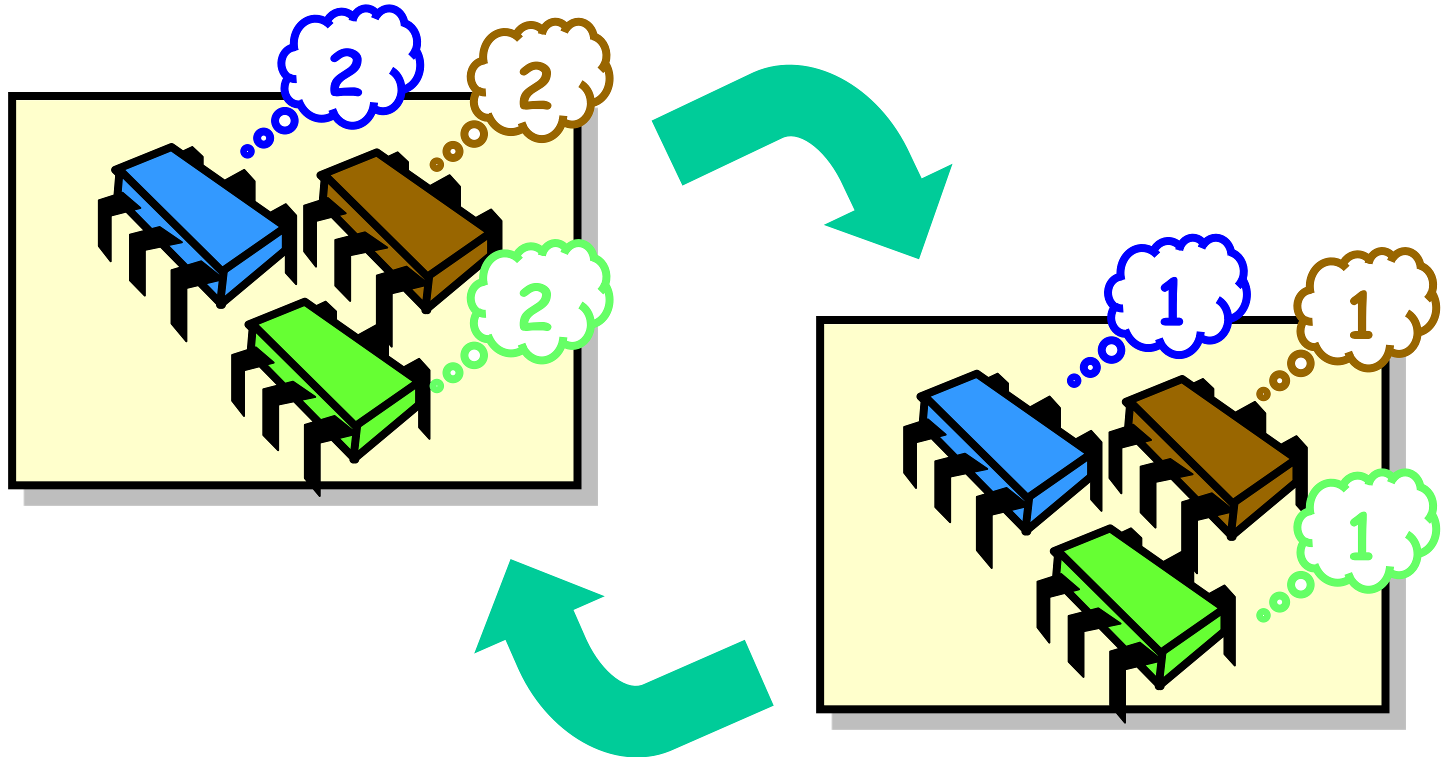
Synchronization Problems

- How do threads stay in phase?
- Too early?
  - "we render no frame before its time"
- Too late?
  - Recycle memory before frame is displayed

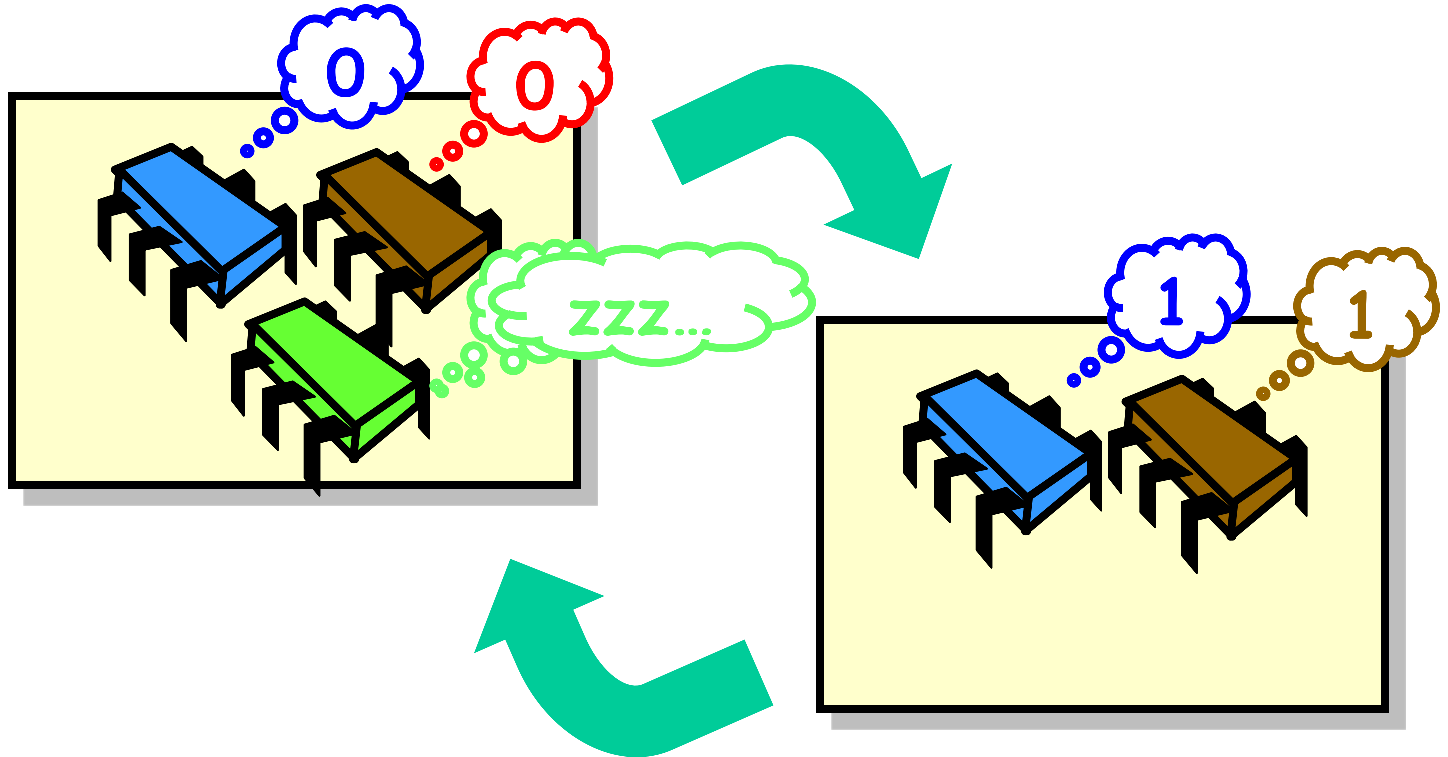
Ideal Parallel Computation



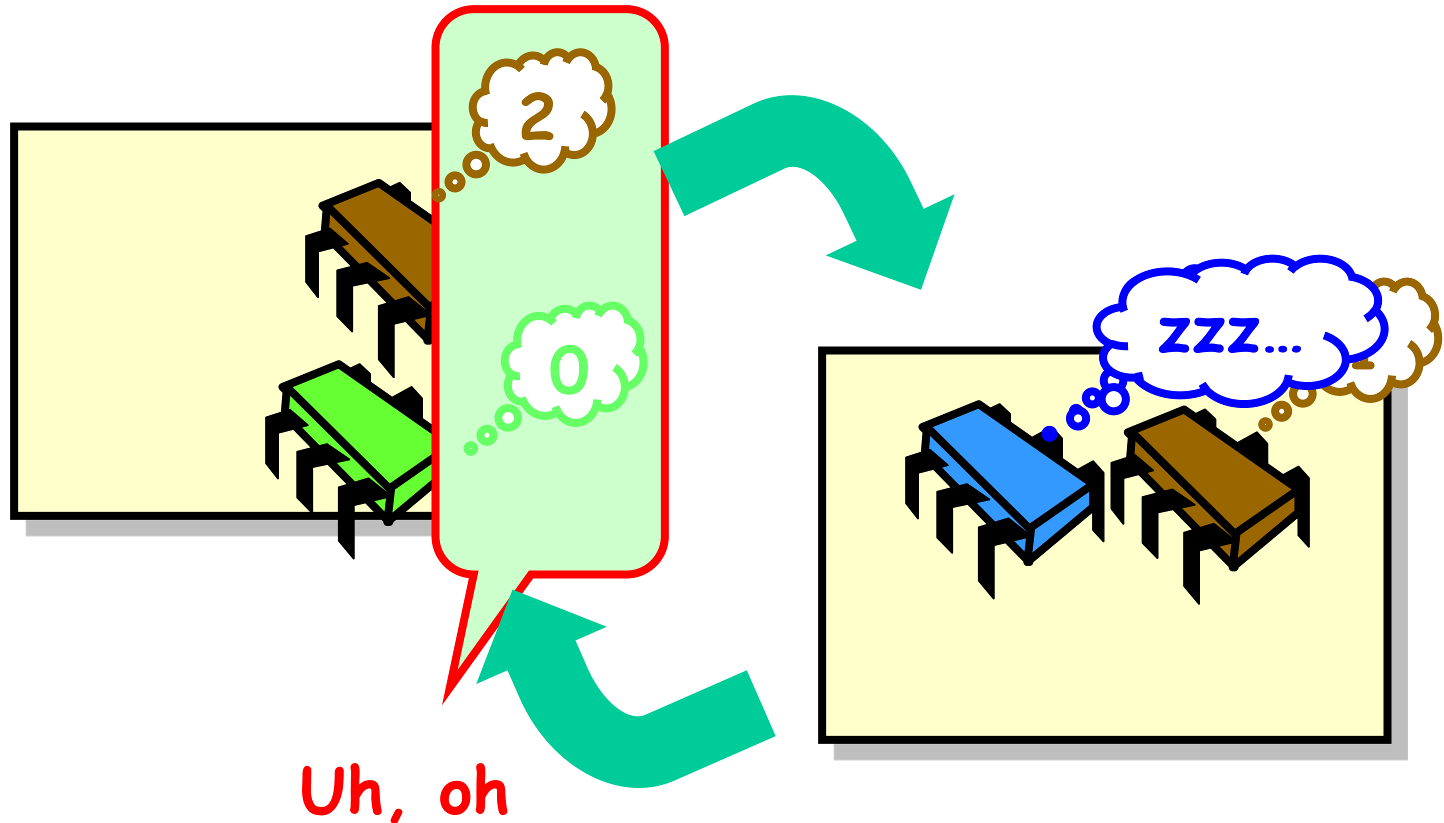
Ideal Parallel Computation



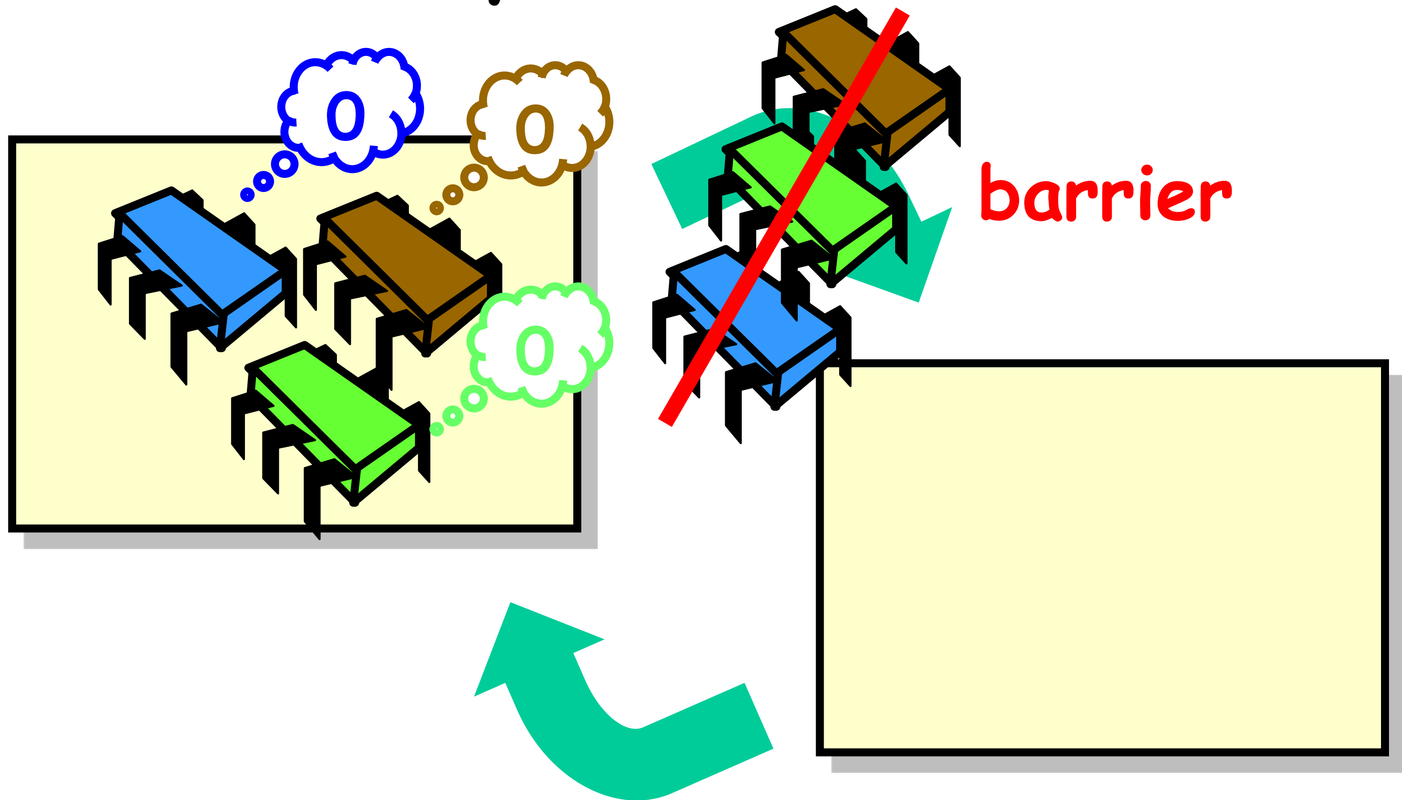
Real-Life Parallel Computation



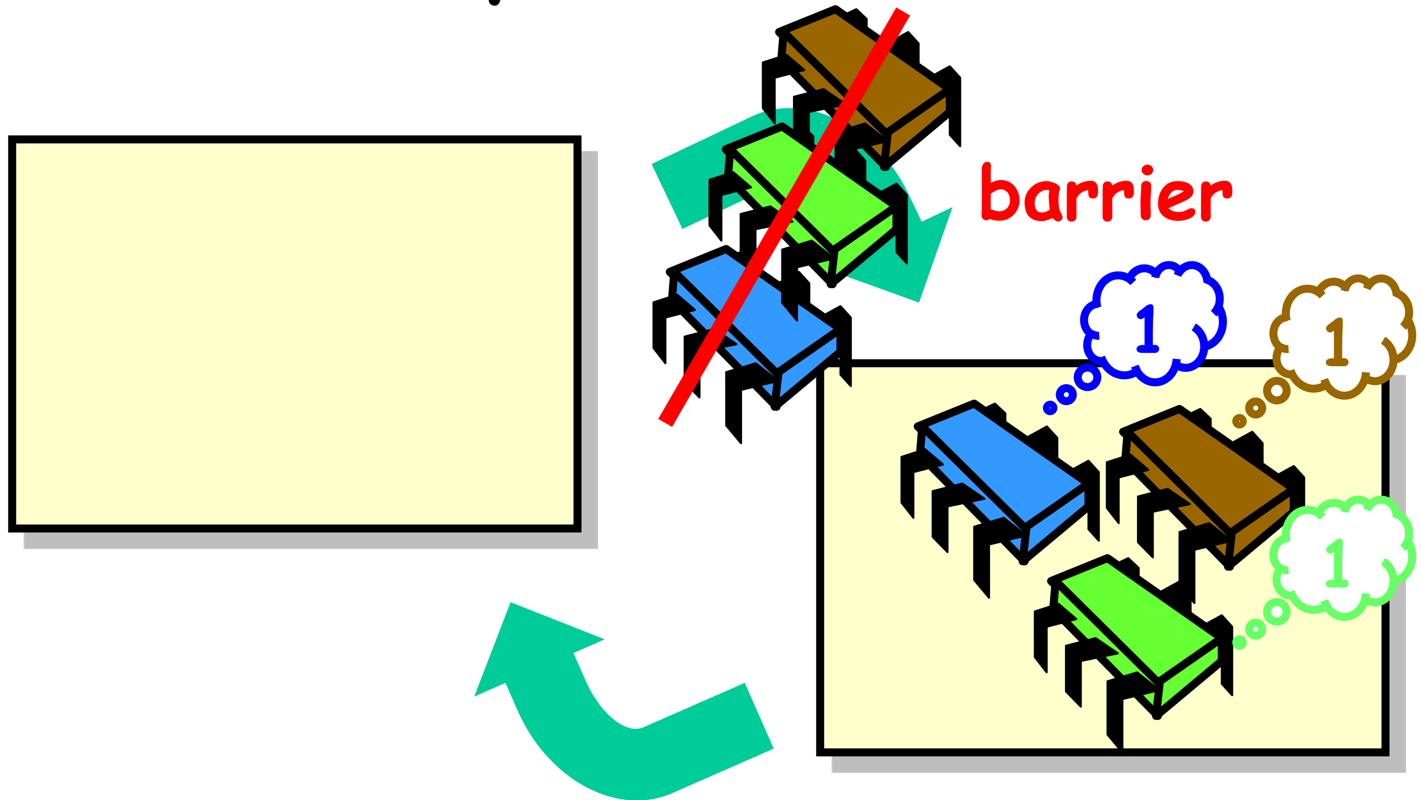
Real-Life Parallel Computation



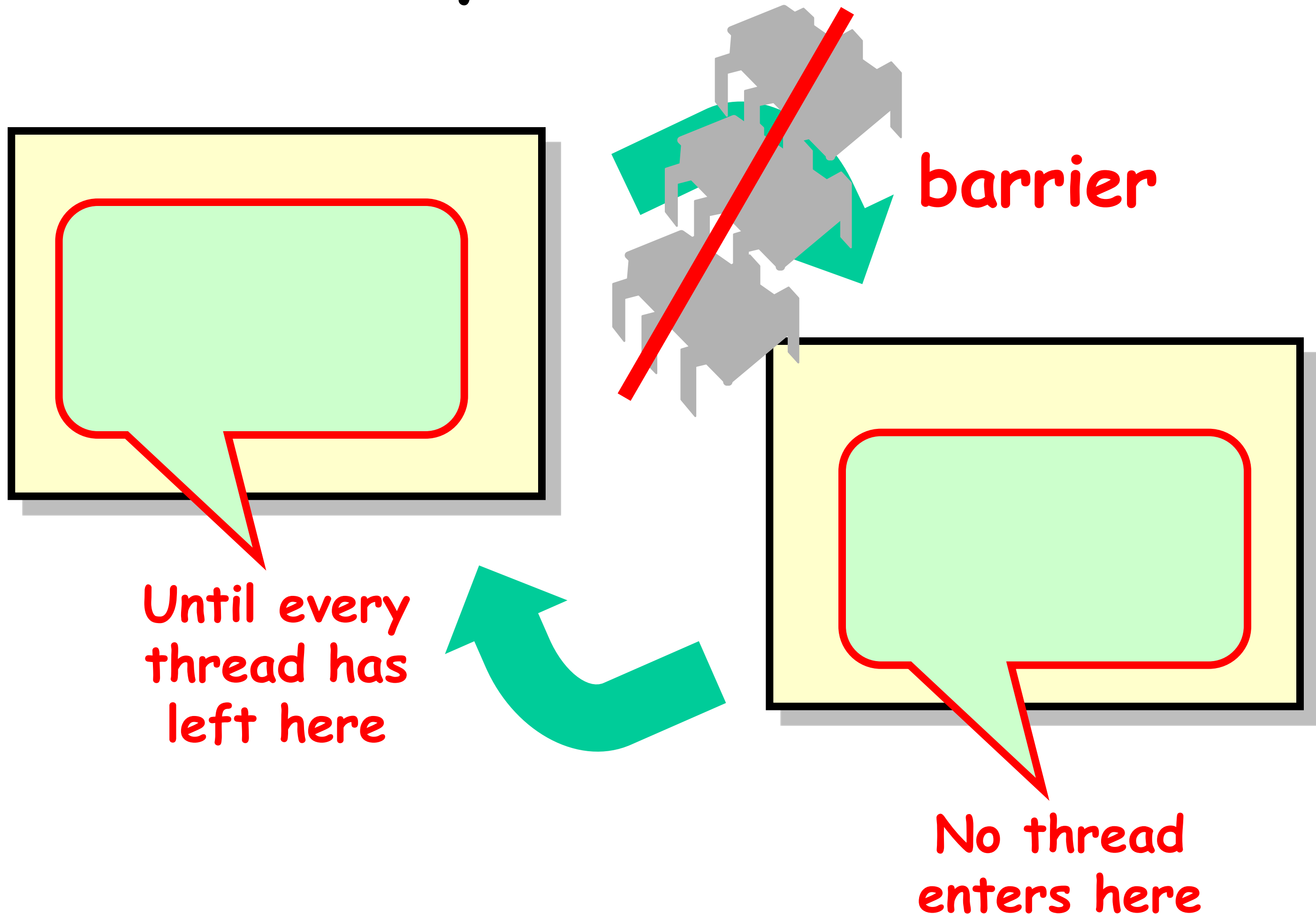
Barrier Synchronization



Barrier Synchronization



Barrier Synchronization



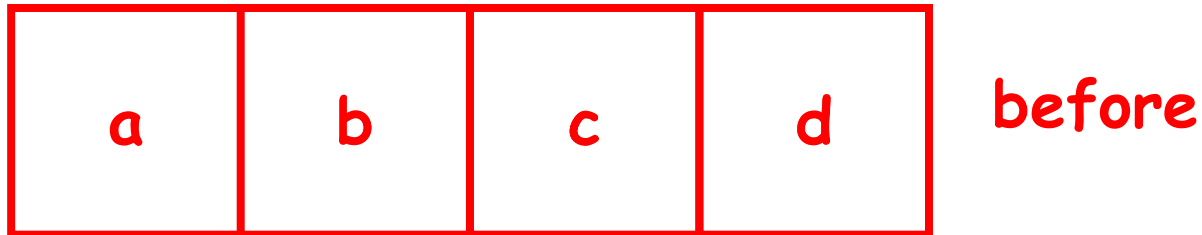
Why Do We Care?

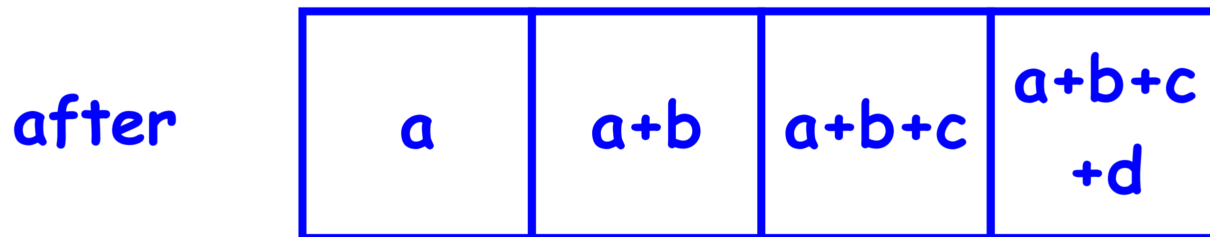
- Mostly of interest to
  - Scientific & numeric computation
- Elsewhere
  - Garbage collection
  - Less common in systems programming
  - Still important topic

Duality

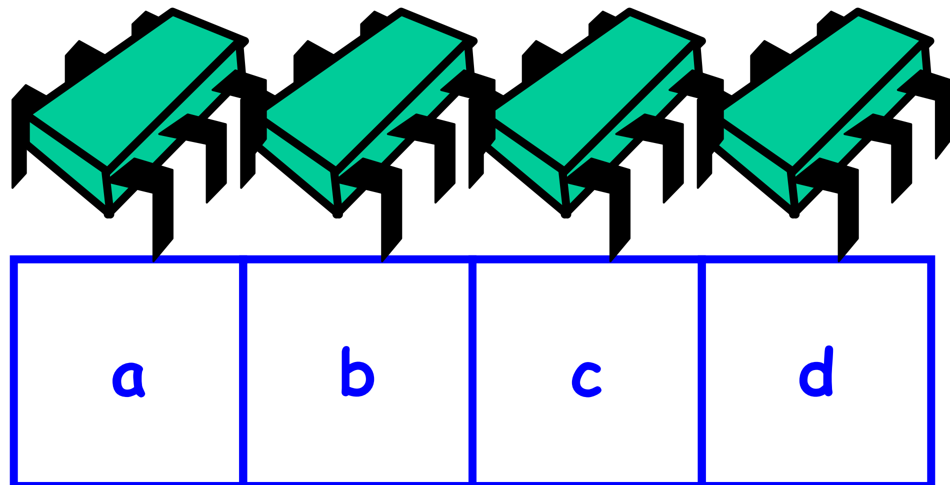
- Dual to mutual exclusion
  - Include others, not exclude them
- Same implementation issues
  - Interaction with caches …
    - Invalidation?
    - Local spinning?

Example: Parallel Prefix



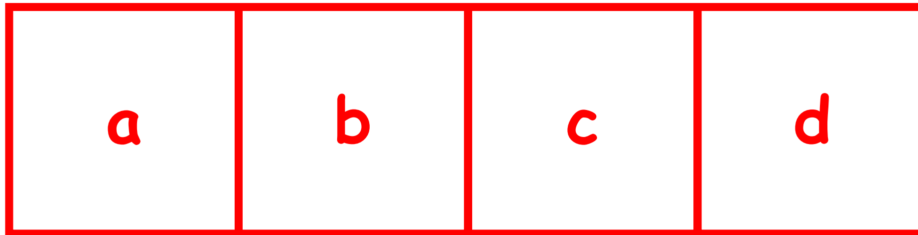


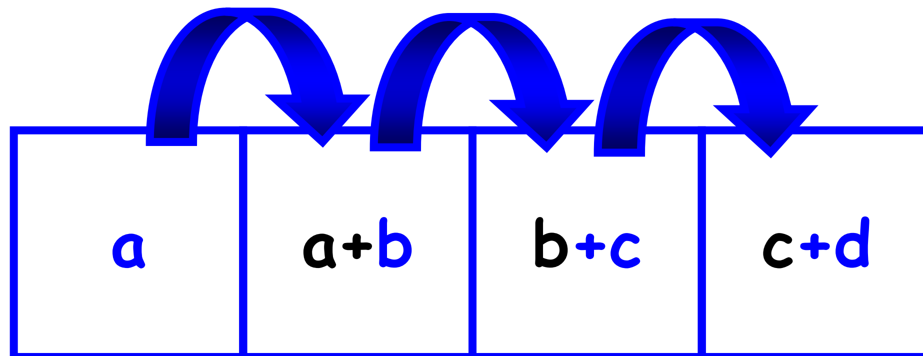
Parallel Prefix



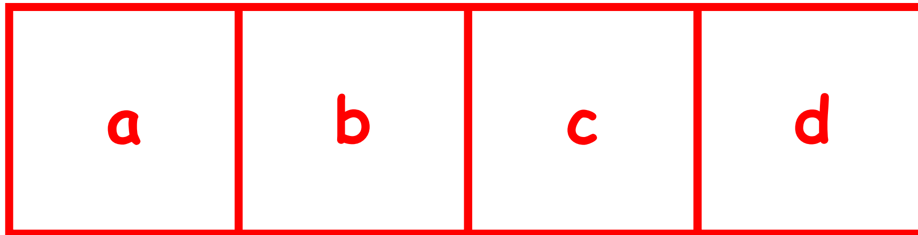
One thread
Per entry

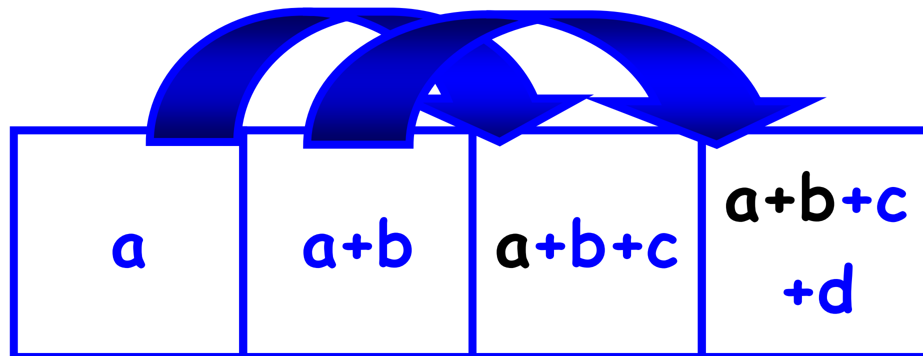
Parallel Prefix: Phase 1





Parallel Prefix: Phase 2





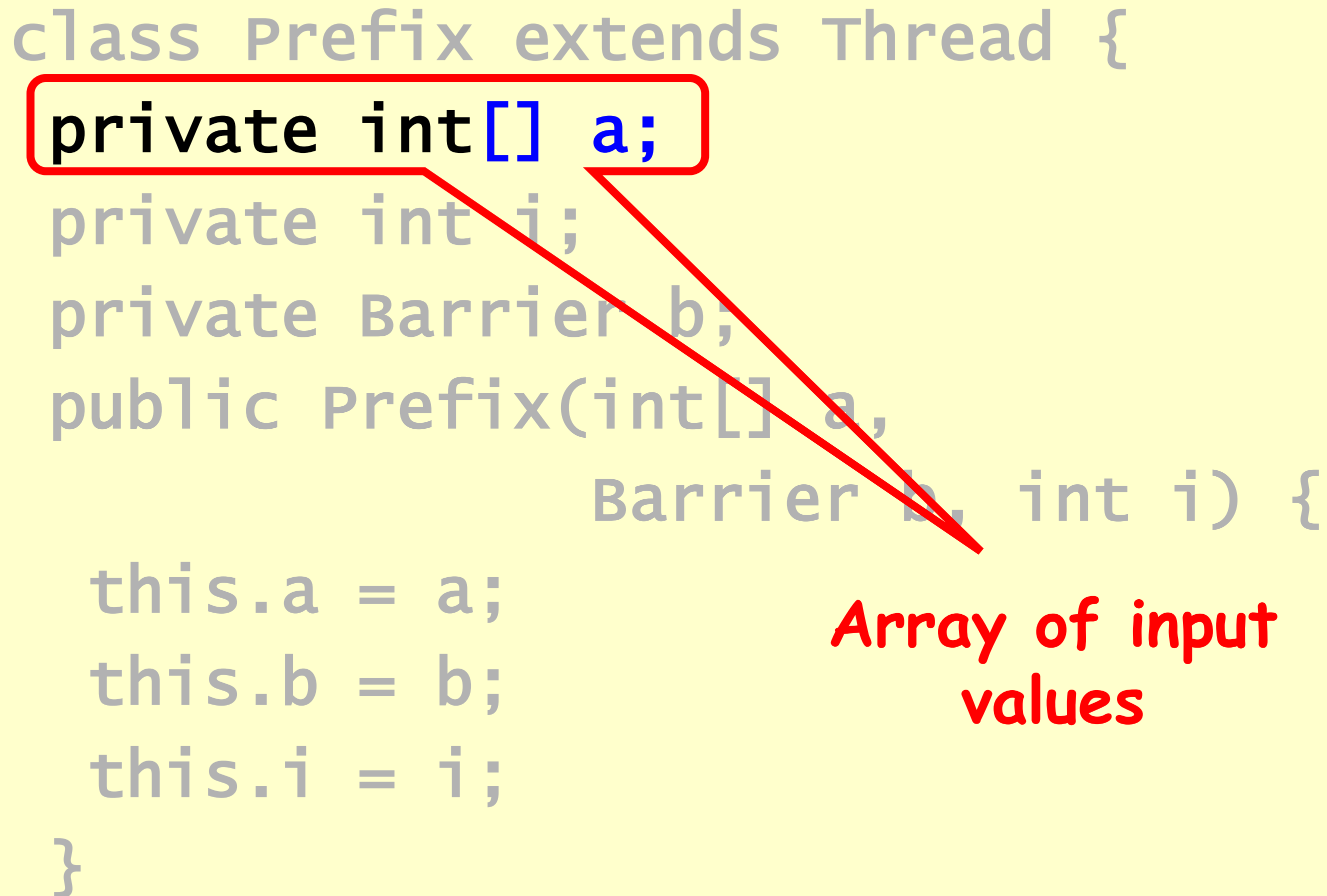
Parallel Prefix

- N threads can compute
  - Parallel prefix
  - Of N entries
  - In $\log_2 N$ rounds
- What if system is asynchronous?
  - Why we need barriers

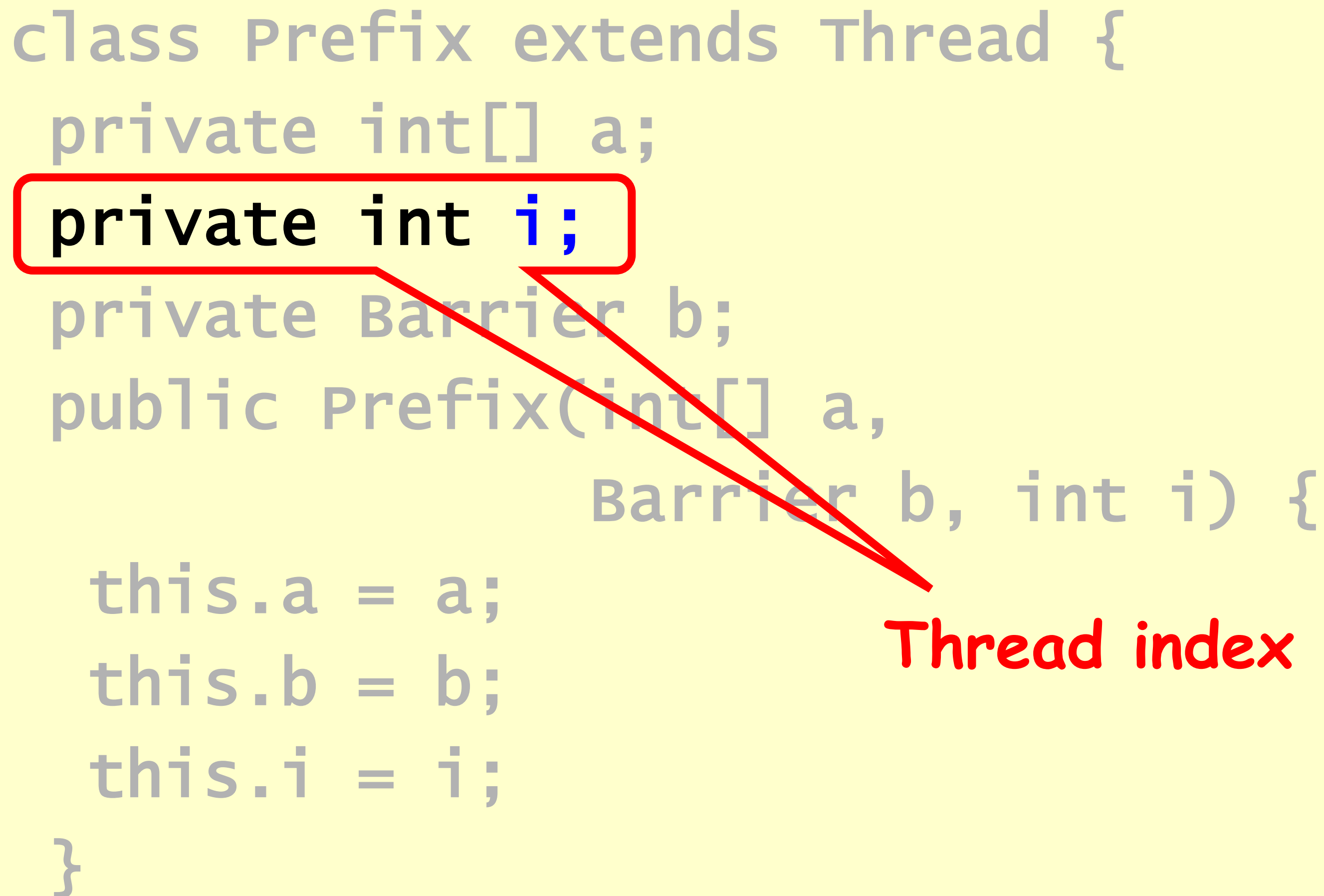
Prefix

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

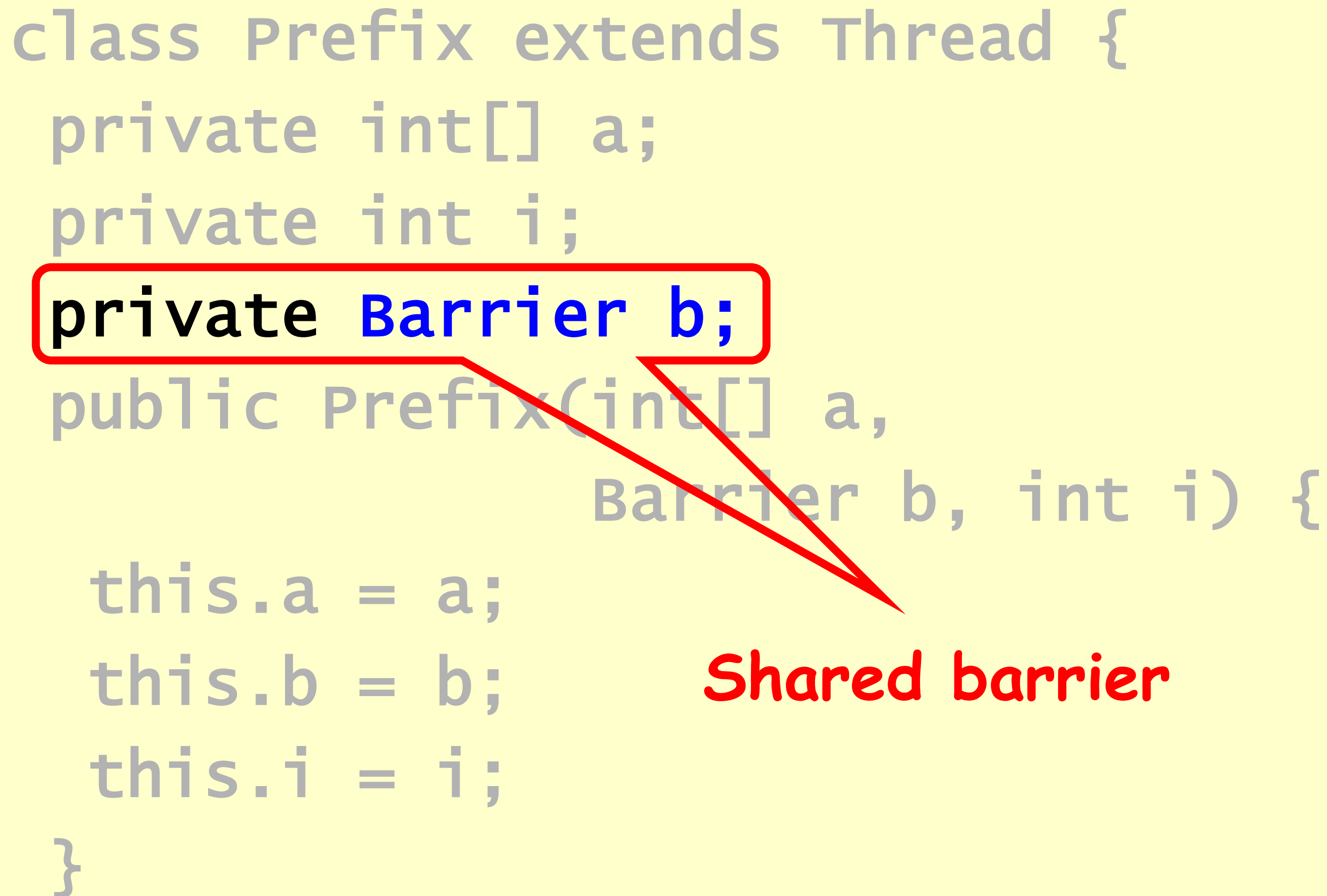
Prefix



Prefix



Prefix




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

Prefix

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

Initialize fields

Where Do the Barriers Go?

```
public void run() {
 int d = 1, sum = 0;
 while (d < N) {
  if (i >= d)
    sum = a[i-d];
  if (i >= d)
   a[i] += sum;
  d = d * 2;
}}}
```

Where Do the Barriers Go?

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

Where Do the Barriers Go?

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

Make sure everyone reads before anyone writes

Where Do the Barriers Go?

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

Make sure everyone reads before anyone writes

Where Do the Barriers Go?

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

Make sure everyone reads before anyone writes

Make sure everyone writes 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){
  this.count.set(this.size = n);
 }
 public void await() {
  if (count.getAndDecrement()==1) {
   this.count.set(this.size);
  } else {
   while (this.count.get() != 0) {}
 }}}}
```

Barriers

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

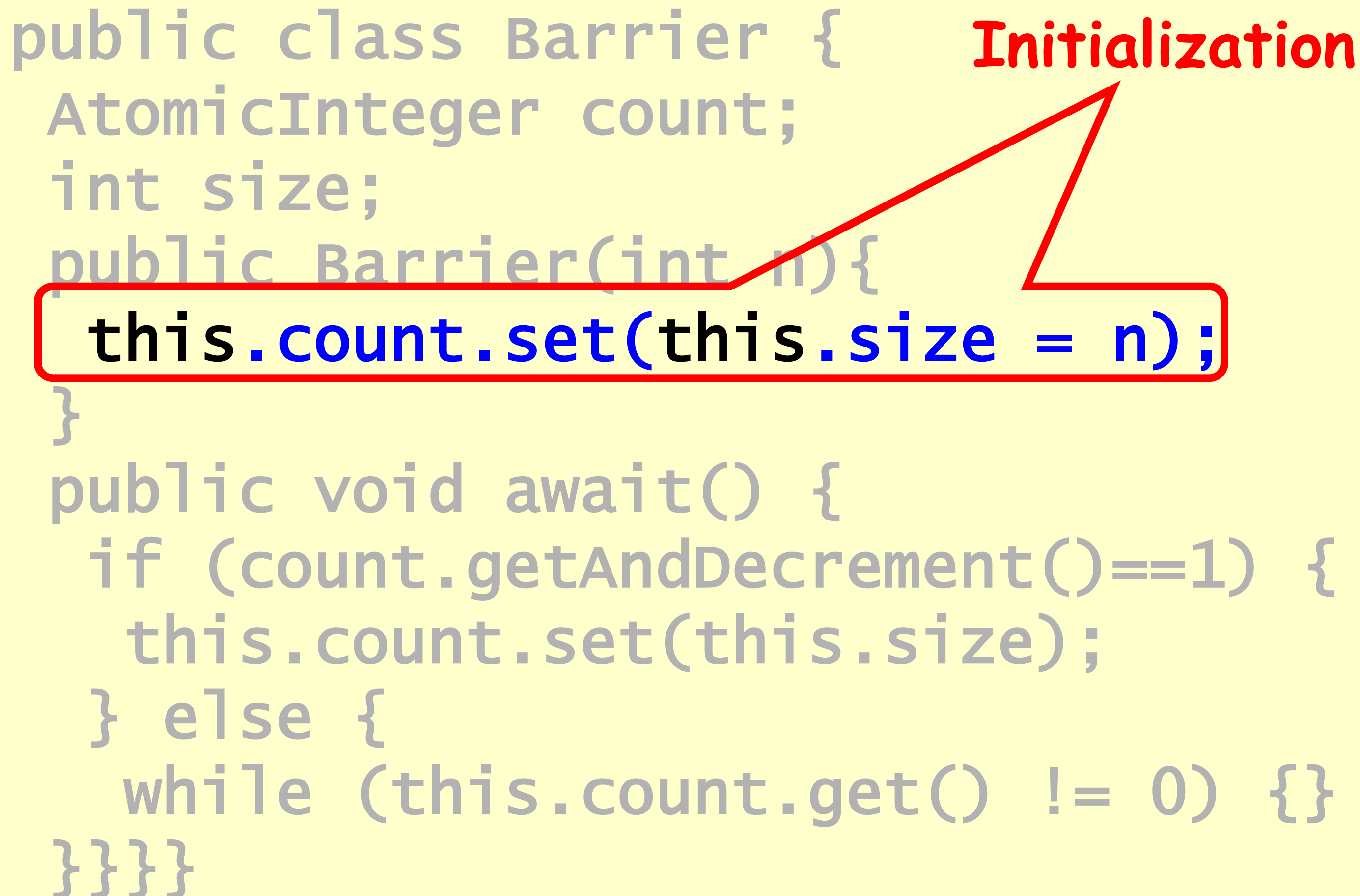
Number threads not yet arrived

Barriers

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

Number threads participating

Barriers



Barriers

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

Principal method

Barriers

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

If I'm last, reset fields for next time

Barriers

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

Otherwise, wait for everyone else

Barriers

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

What's wrong with this protocol?

Reuse

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

Do work

synchronize

repeat

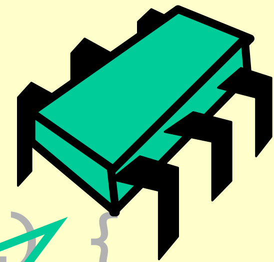
Barriers

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

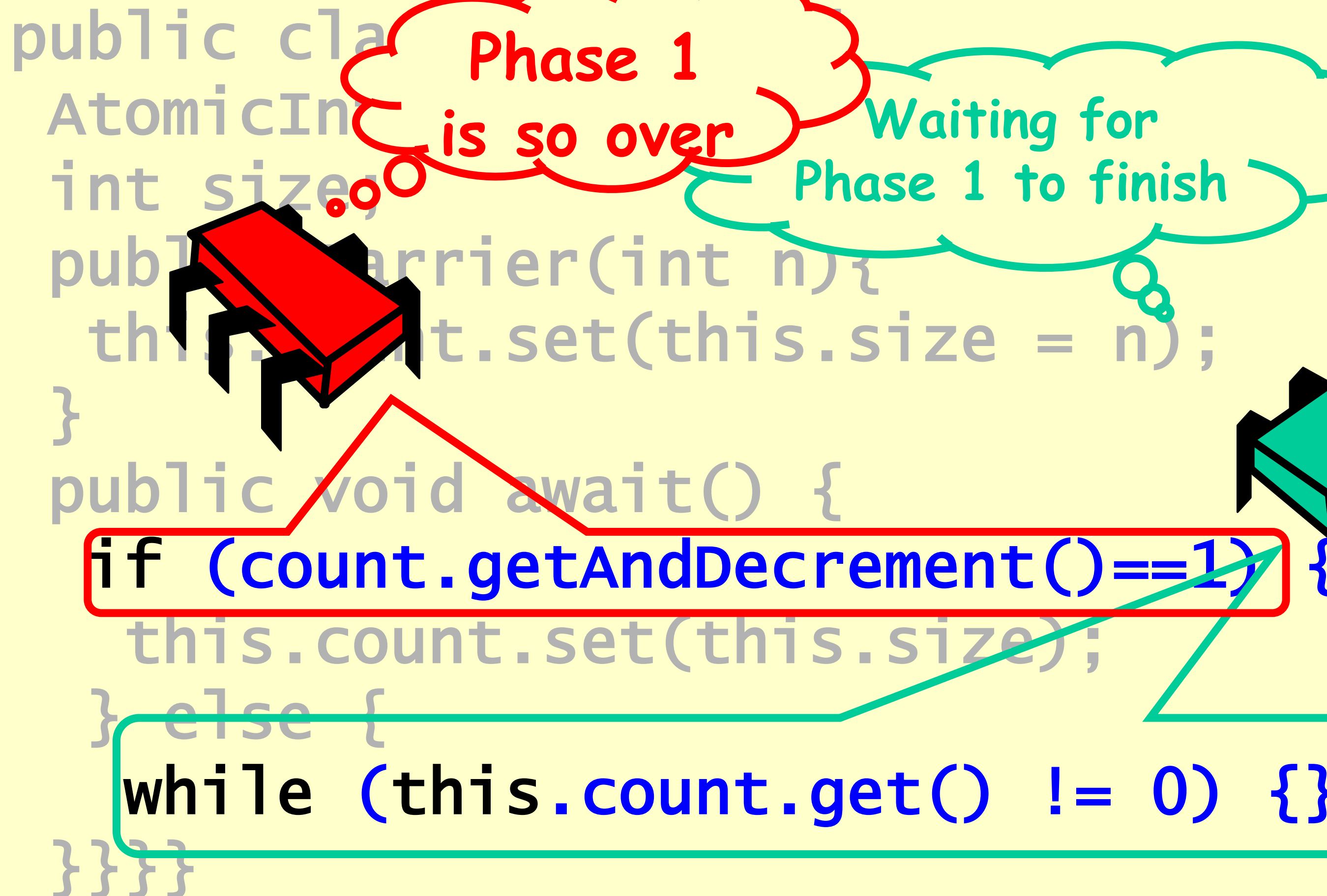
Barriers

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

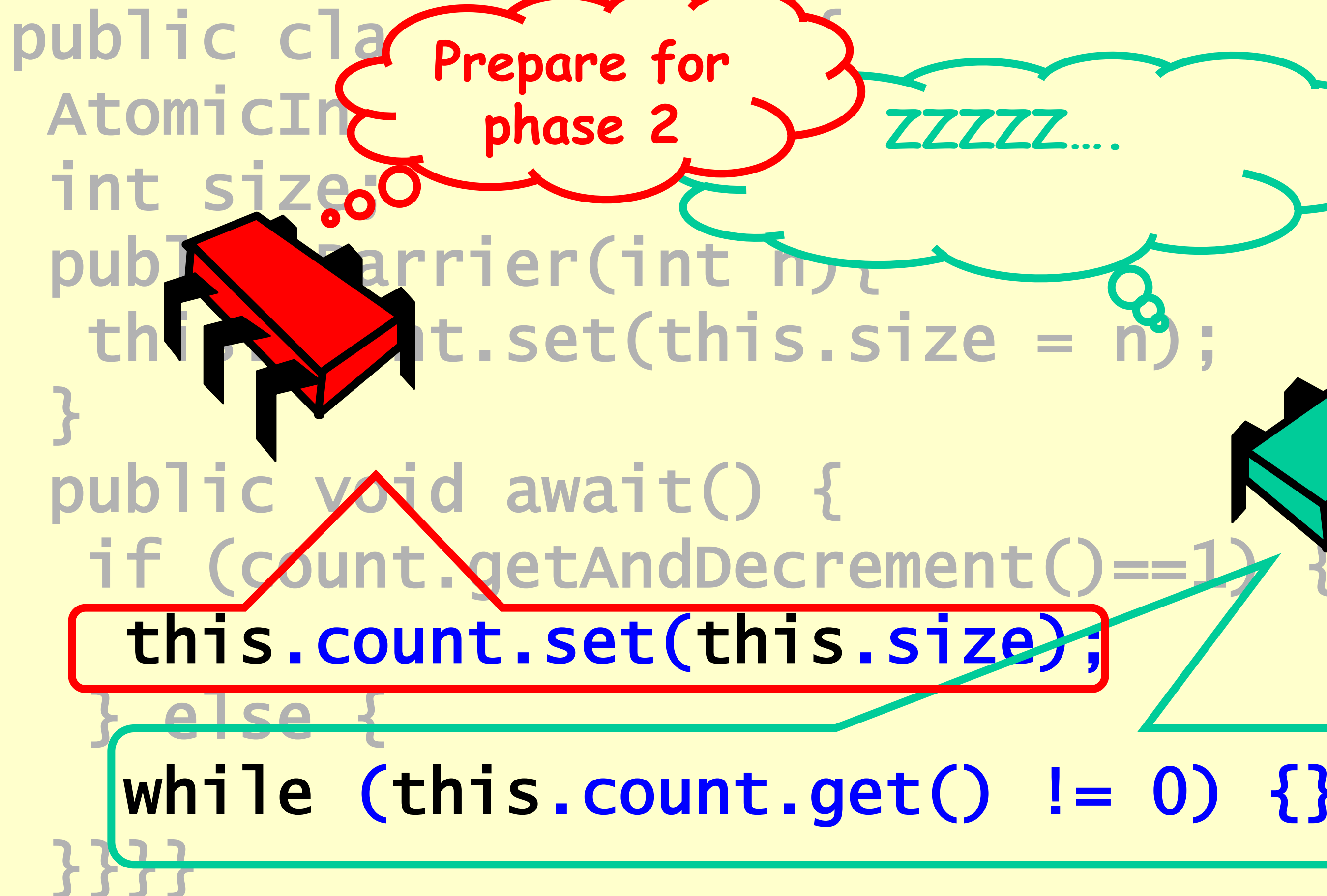
Waiting for Phase 1 to finish



Barriers

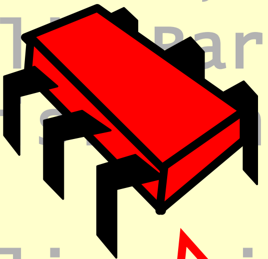


Barriers



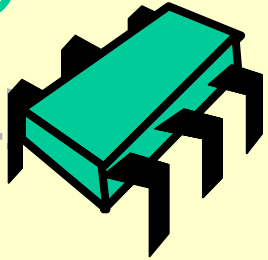
Uh-Oh

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



Waiting for
Phase 2 to finish

Waiting for
Phase 1 to finish



Basic Problem

- One thread "wraps around" to start phase 2
- While another thread is still waiting for phase 1
- One solution:
  - Always use two barriers

Sense-Reversing Barriers

```
public class Barrier {
 AtomicInteger count;
 int size;
 boolean sense = true;

 public void await(boolean mySense) {
  if (count.getAndDecrement()==1) {
   this.count.set(this.size);
   this.sense = !mySense
  } else {
   while (this.sense != mySense) {}
 }}}}
```

Sense-Reversing Barriers

```
public class Barrier {
 AtomicInteger count;
 int size;
 boolean sense = true;

 public void await(boolean mySense) {
  if (count.getAndDecrement()==1) {
   this.count.set(this.size);
   this.sense = mySense
  } else {
   while (this.sense != mySense) {}
 }}}}
```

Completed odd or even-numbered phase?

Sense-Reversing Barriers

```
public class Barrier {
 AtomicInteger count;
 int size;
 boolean sense = true;

 public void await(boolean mySense) {
  if (count.getAndDecrement()==1) {
   this.count.set(this.size);
   this.sense = mySense
  } else {
   while (this.sense != mySense) {}
 }}}}
```

Thread working on odd or even-numbered phase?

Sense-Reversing Barriers

```
public class Barrier {
 AtomicInteger count;
 int size;
 boolean sense = true;

 public void await(boolean mySense) {
  if (count.getAndDecrement()==1) {
   this.count.set(this.size);
   this.sense = mySense
  } else {
   while (this.sense != mySense) {}
 }}}}
```

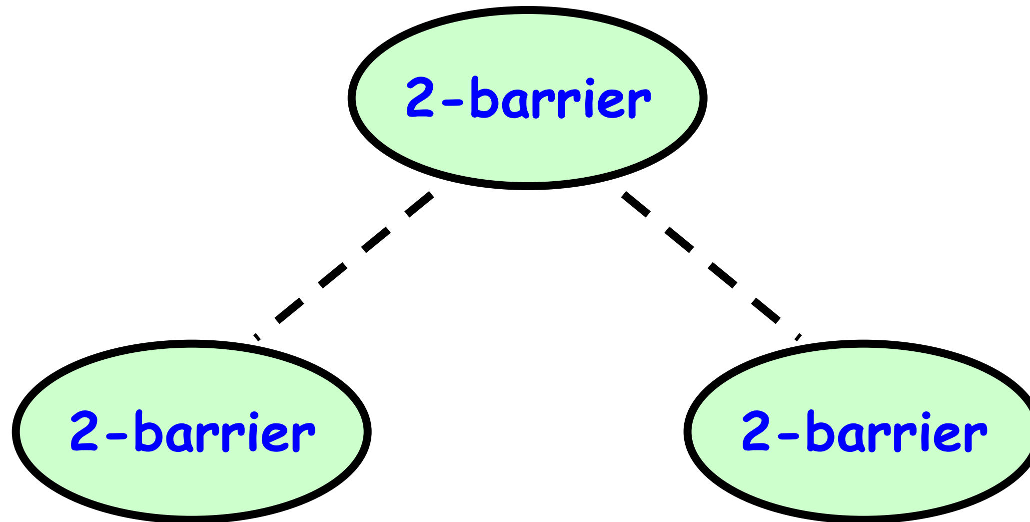
If I'm last, reverse sense for next time

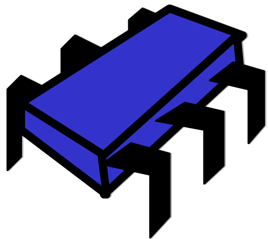
Sense-Reversing Barriers

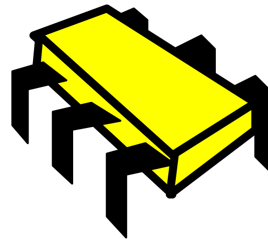
```
public class Barrier {
 AtomicInteger count;
 int size;
 boolean sense = true;

 public void await(boolean mySense) {
  if (count.getAndDecrement()==1) {
   this.count.set(this.size);
   this.sense = mySense
  } else {
   while (this.sense != mySense) {}
 }}}
```

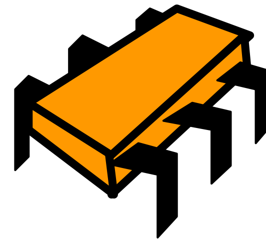
Otherwise, wait for sense to flip

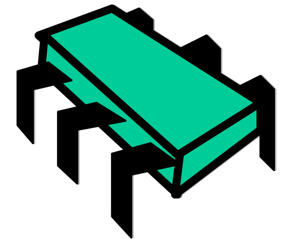
Combining Tree Barriers



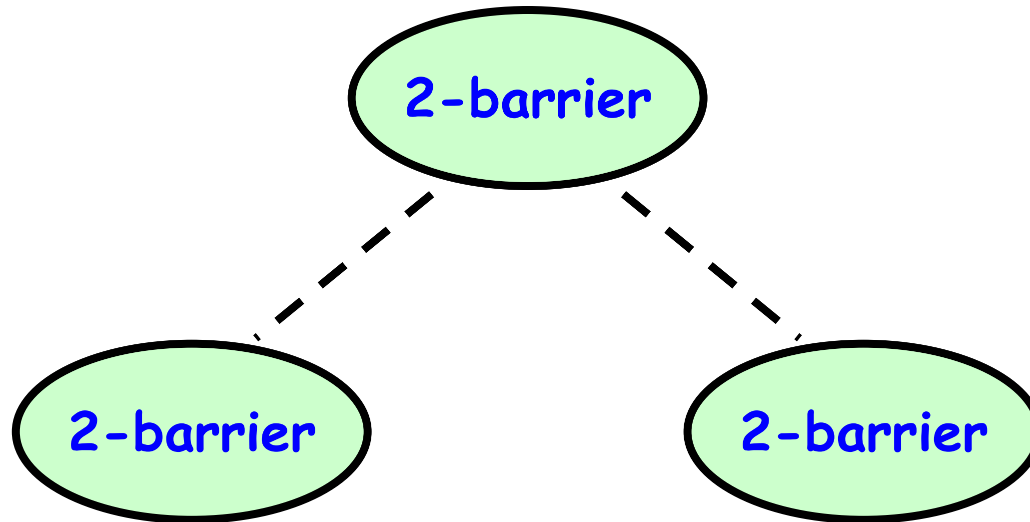


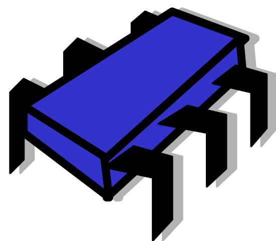
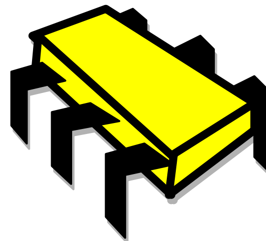
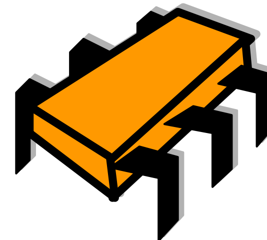
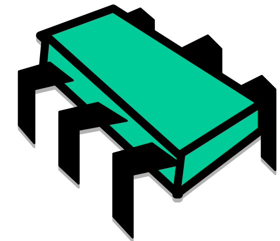




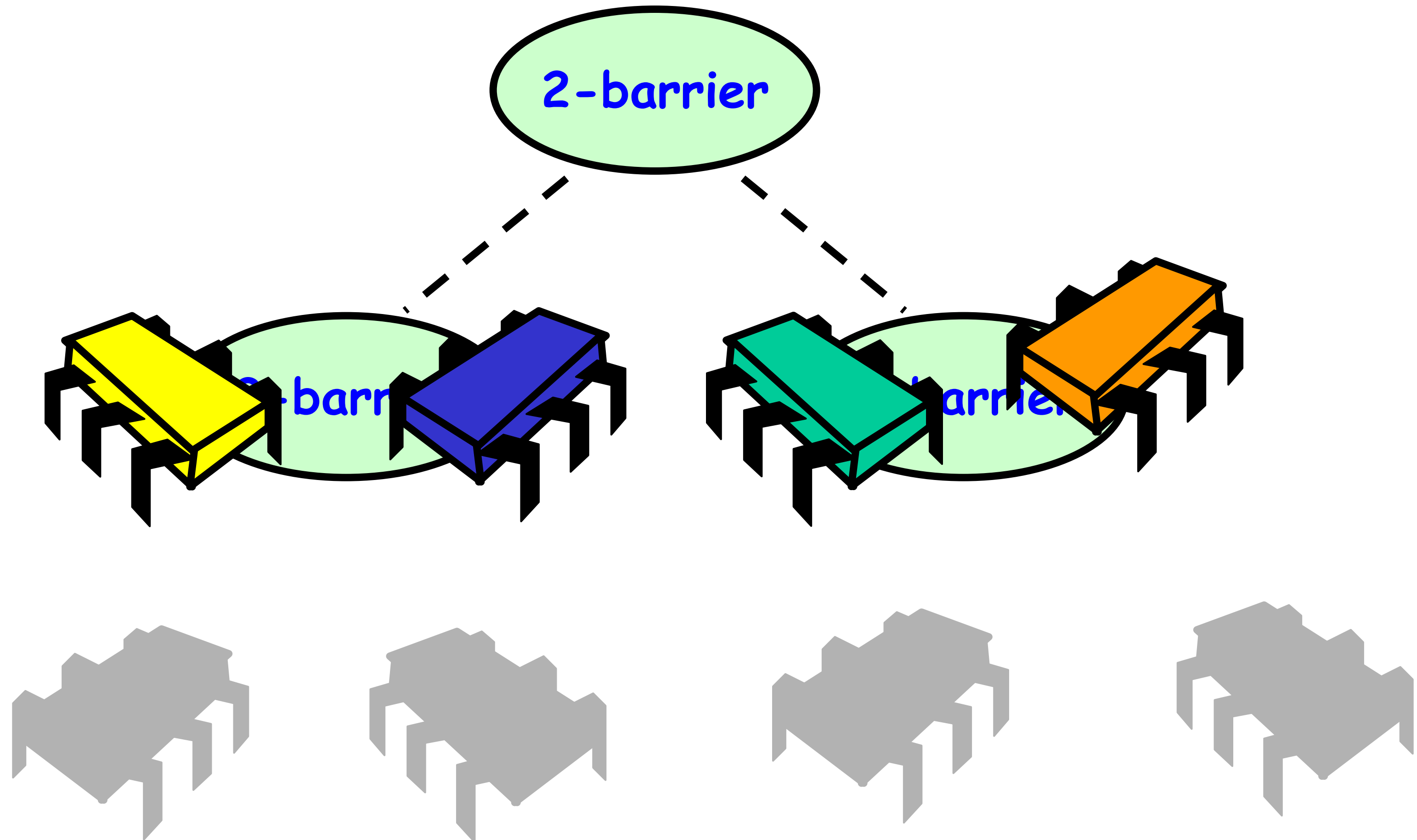


Combining Tree Barriers

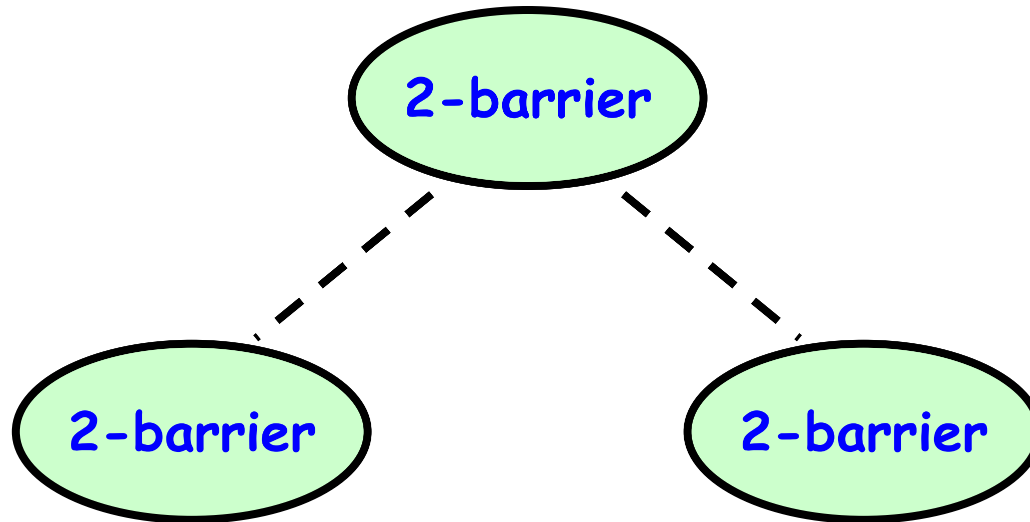


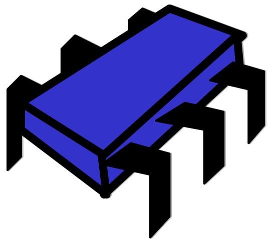





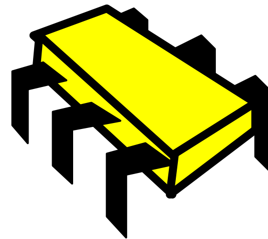
Combining Tree Barriers

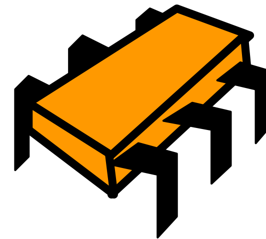


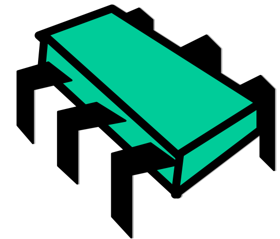
Combining Tree Barriers











Combining Tree Barrier

```
public class CBarrier {
 AtomicInteger count; int size;
 CBarrier parent;

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

Combining Tree Barrier

```
public class CBarrier {
 AtomicInteger count; int size;
 CBarrier parent;

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

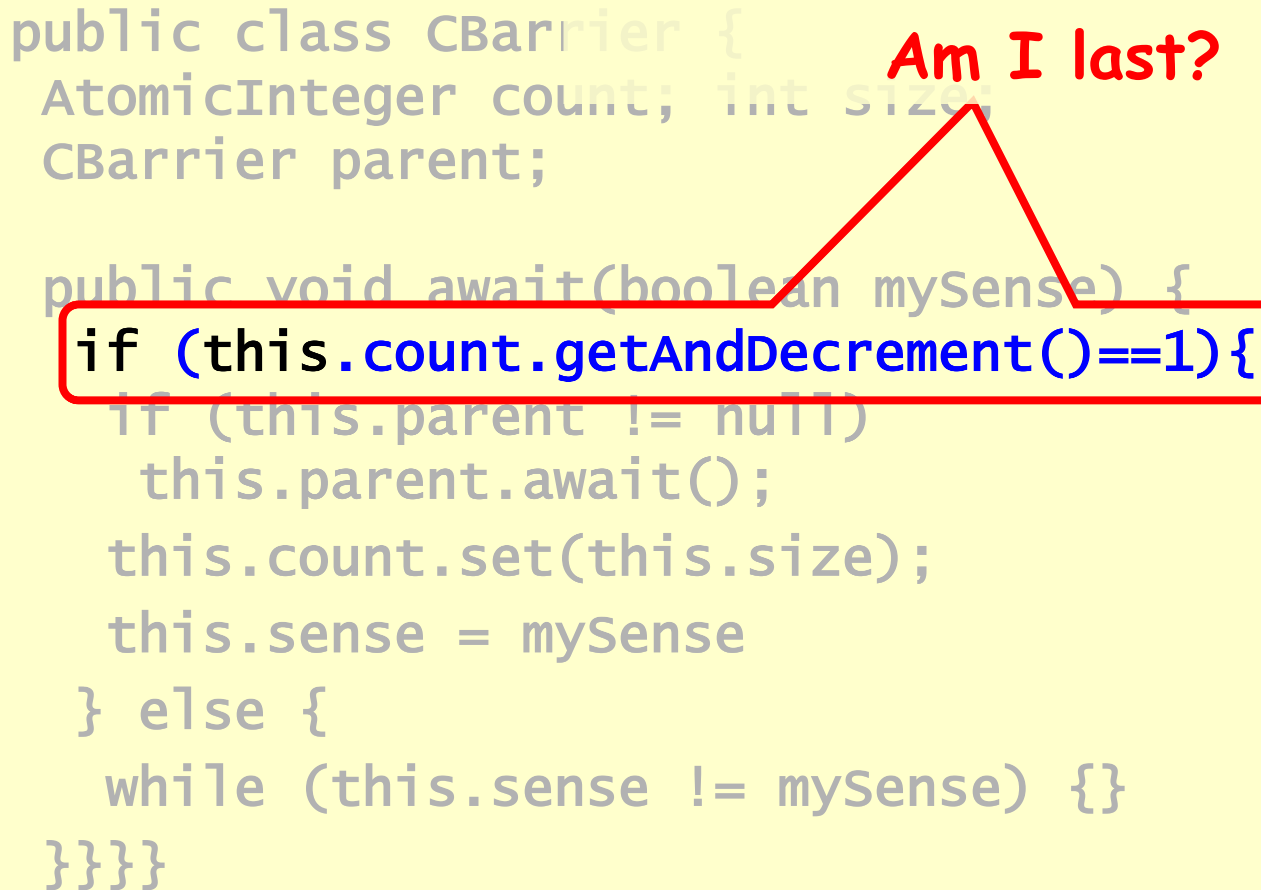
Parent barrier in tree

Combining Tree Barrier

```
public class CBarrier {
 AtomicInteger count; int size;
 CBarrier parent;

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

Thread working on odd or even-numbered phase?

Combining Tree Barrier



Combining Tree Barrier

```
public class CBarrier {
 AtomicInteger count; int size;
 CBarrier parent;

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

Proceed to parent barrier

Combining Tree Barrier

```
public class CBarrier {
 AtomicInteger count; int size;
 CBarrier parent;

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

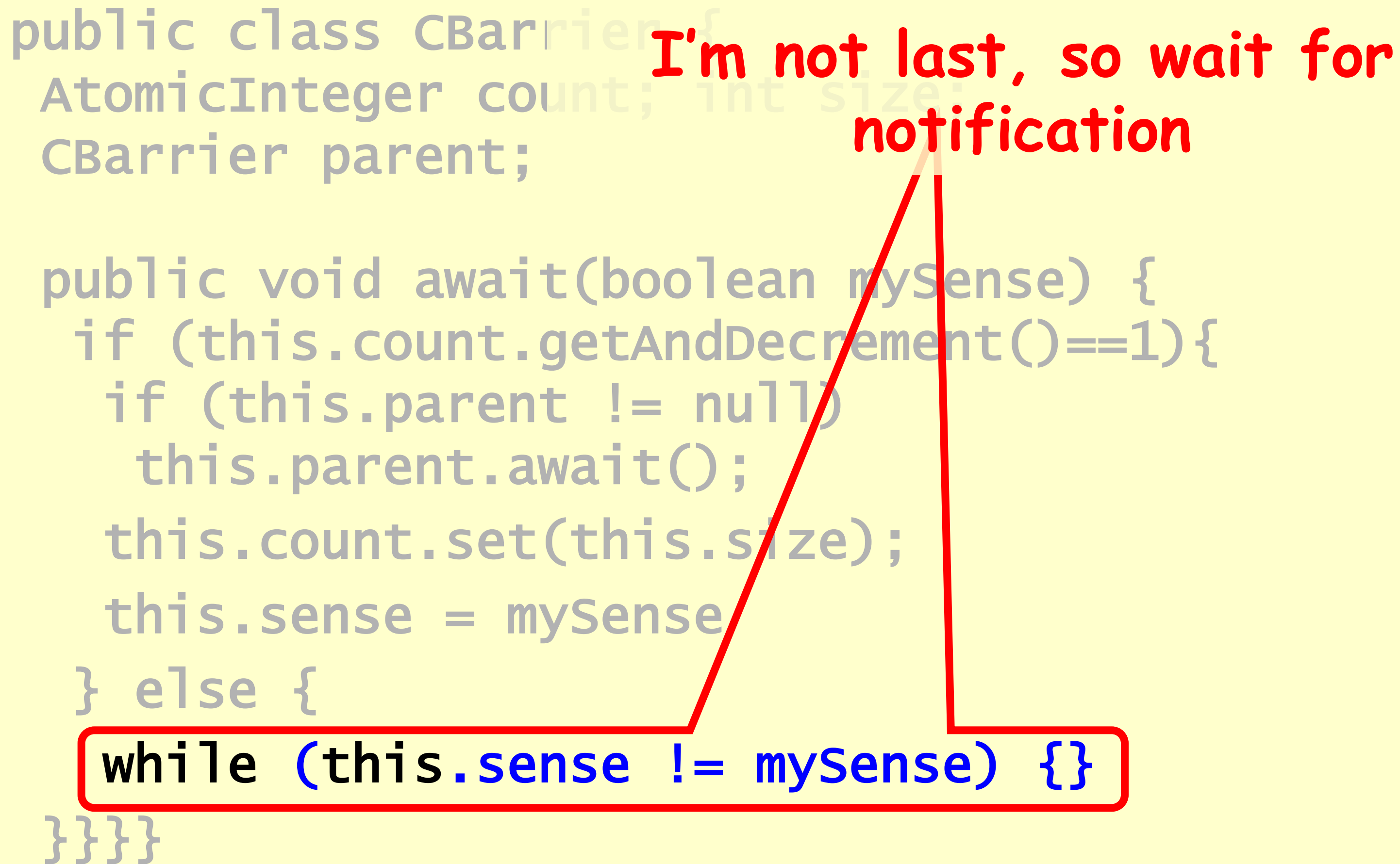
Prepare for next phase

Combining Tree Barrier

```
public class CBarrier {
 AtomicInteger count; int size;
 CBarrier parent;

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

Notify others at this node

Combining Tree Barrier



Combining Tree Barrier

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

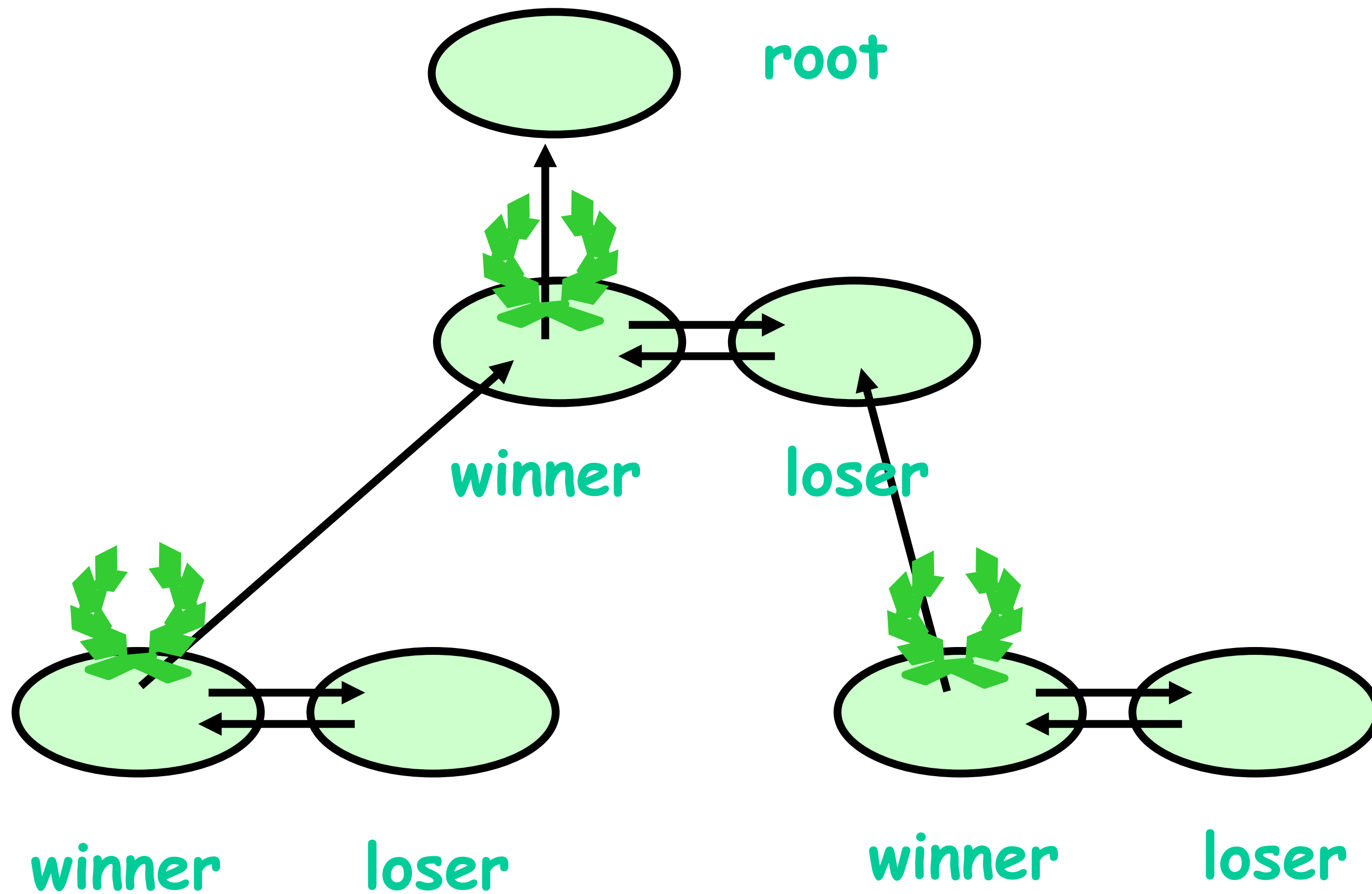
Remarks

- Everyone spins on sense field
  - Local spinning on bus-based (good)
  - Network hot-spot on distributed architecture (bad)
- Not really scalable

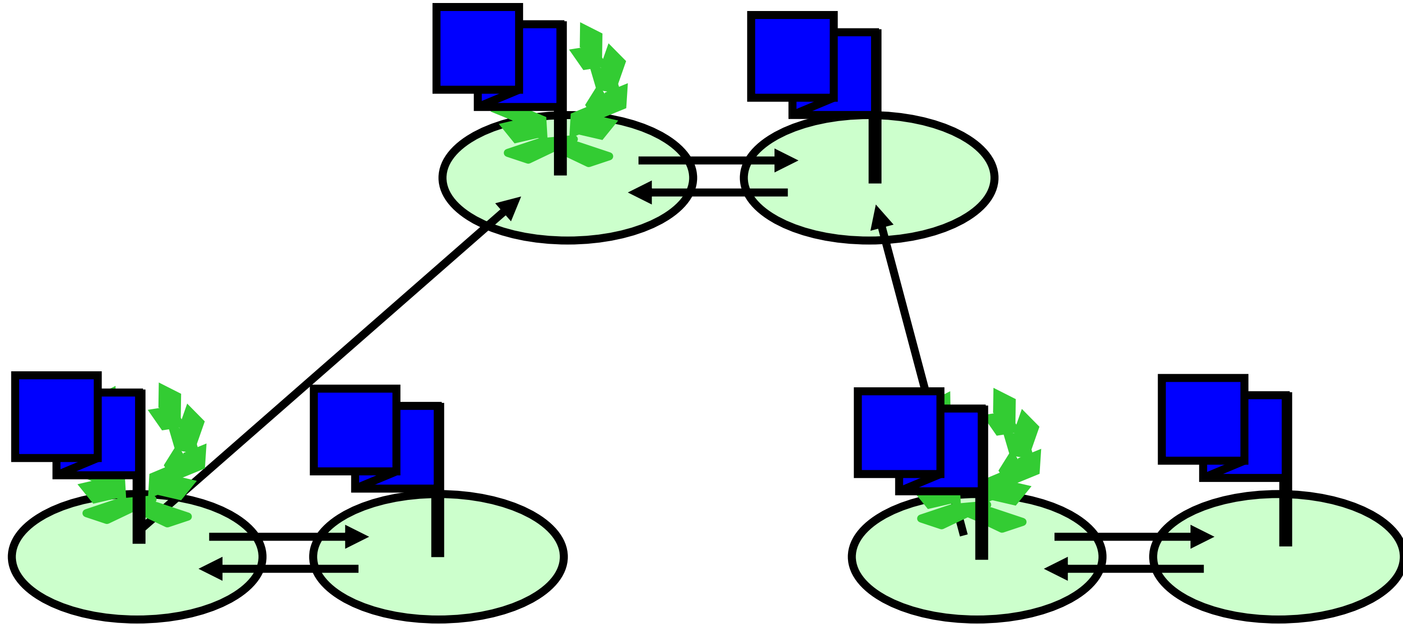
Tournament Tree Barrier

- If tree nodes have fan-in 2
  - Don't need to call getAndDecrement()
  - Winner chosen statically
- At level i
  - If i-th bit of id is 0, move up
  - Otherwise keep back

Tournament Tree Barriers

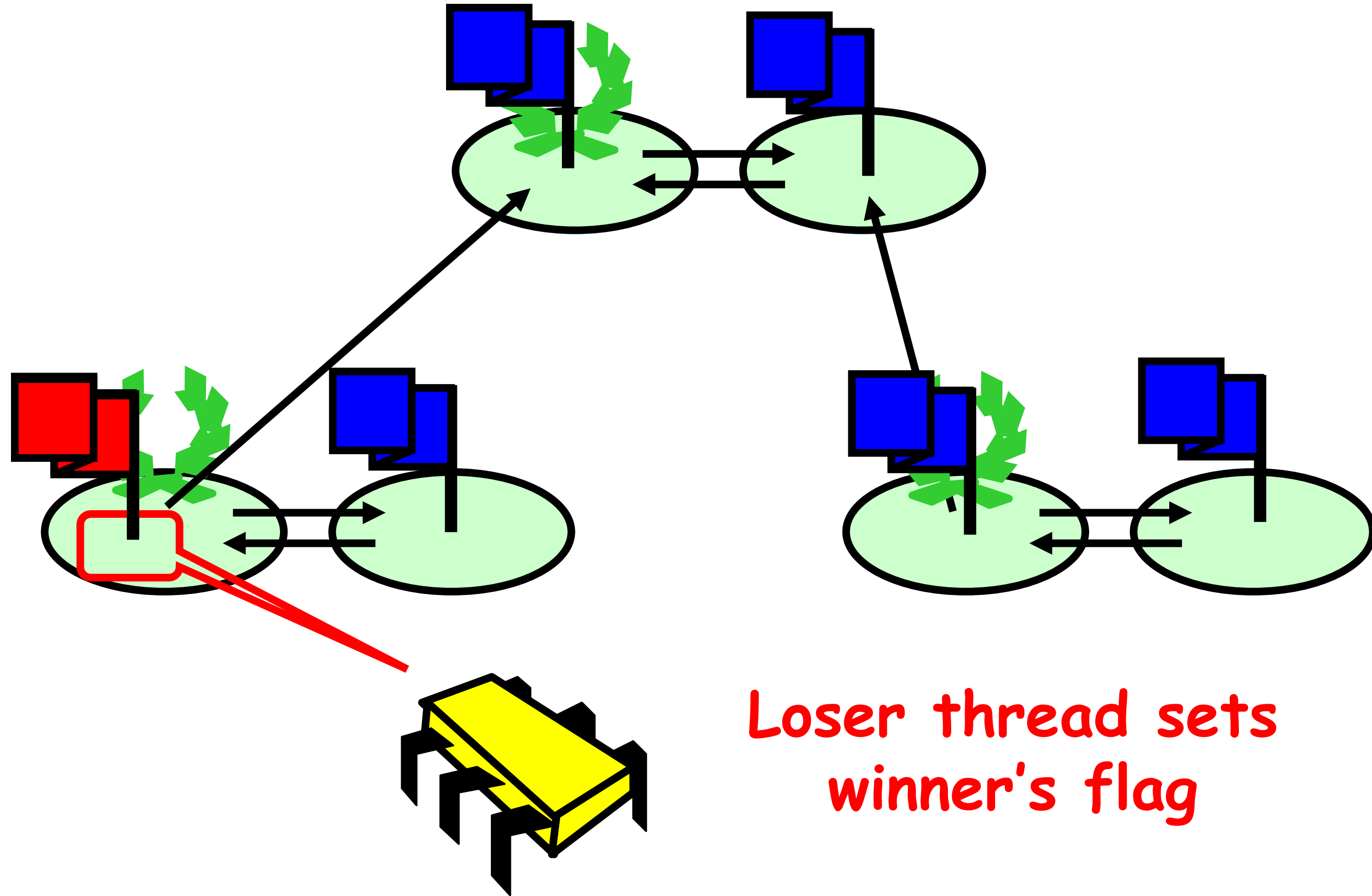


Tournament Tree Barriers

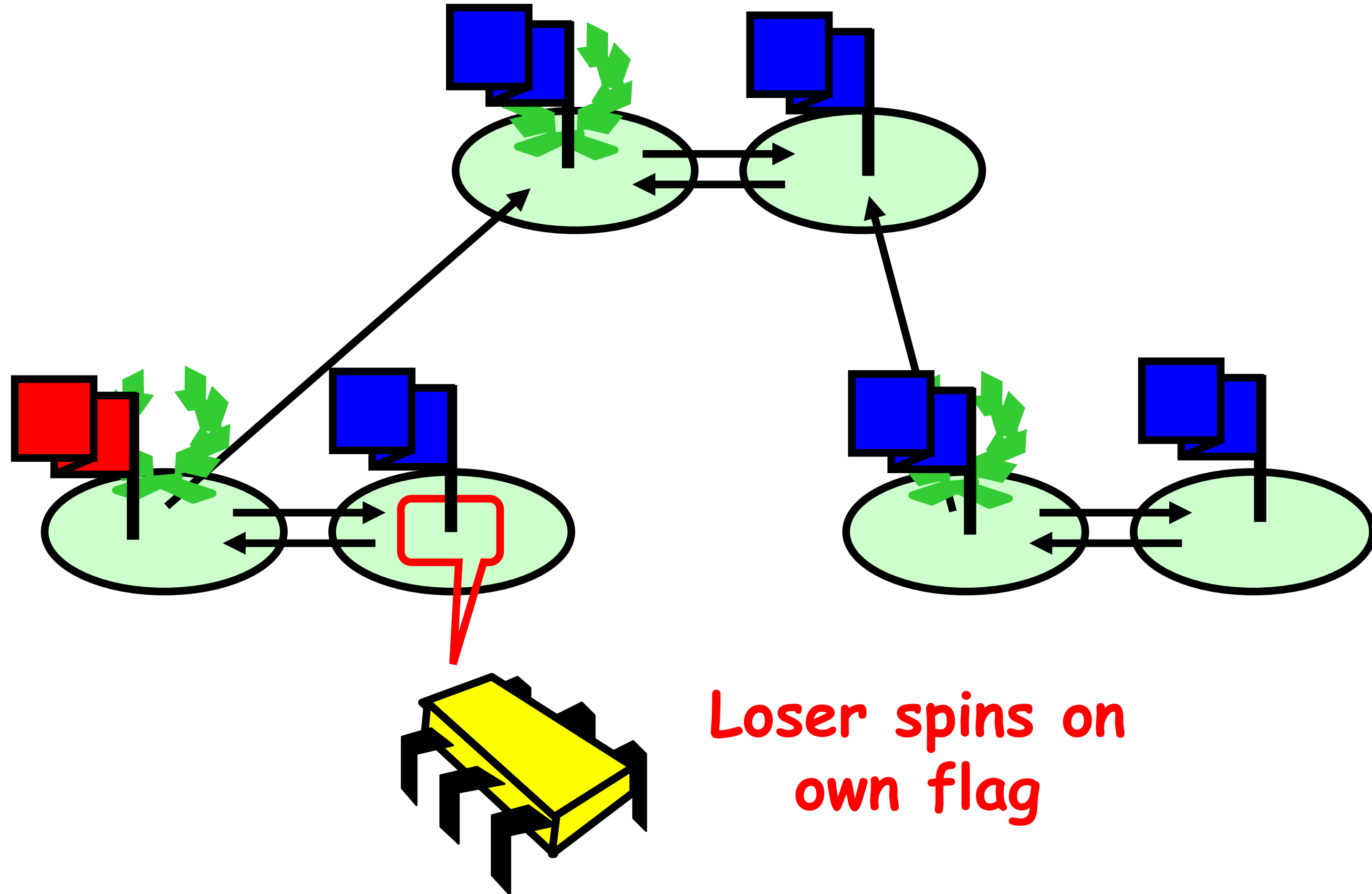


All flags blue

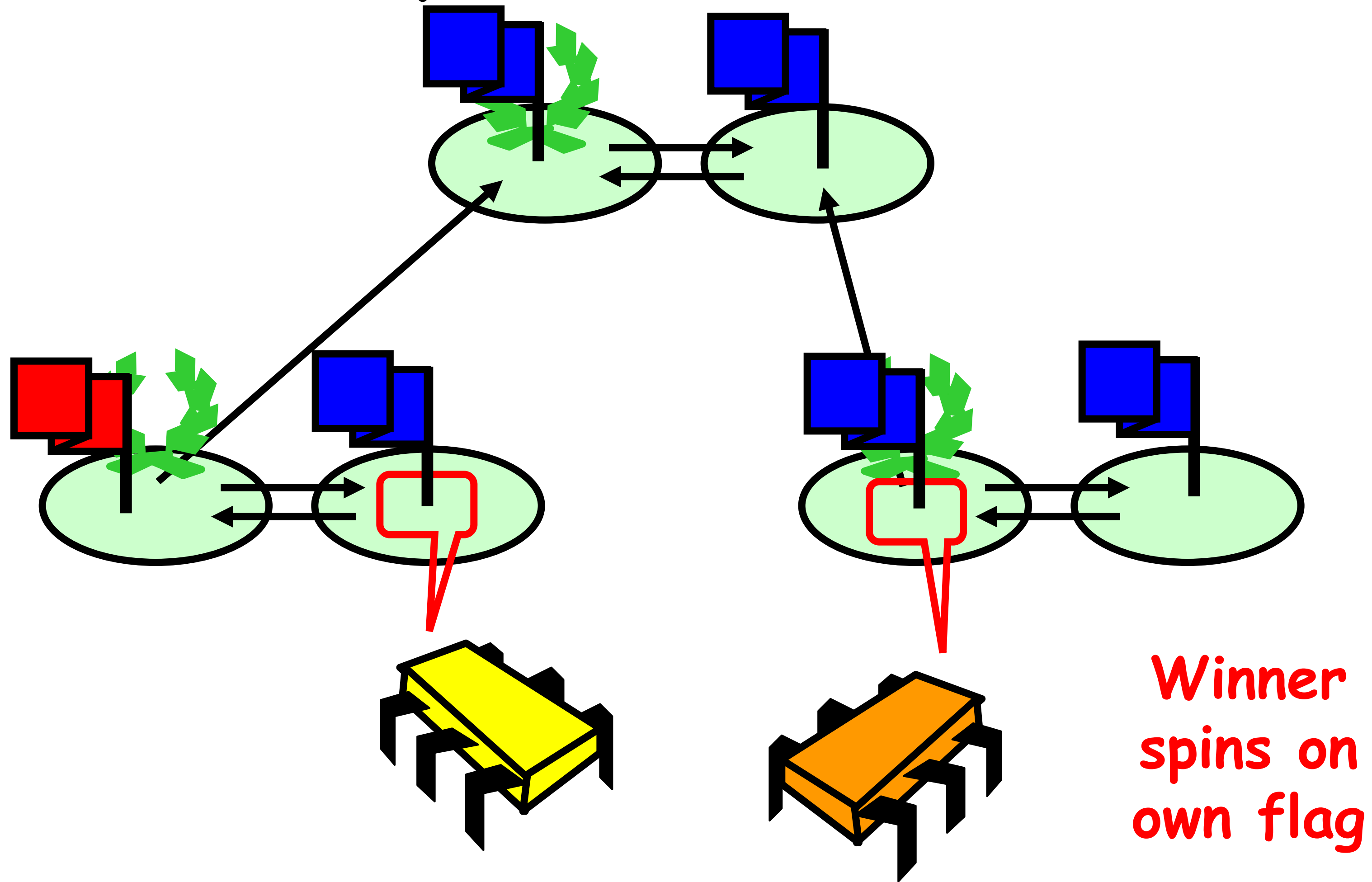
Tournament Tree Barriers



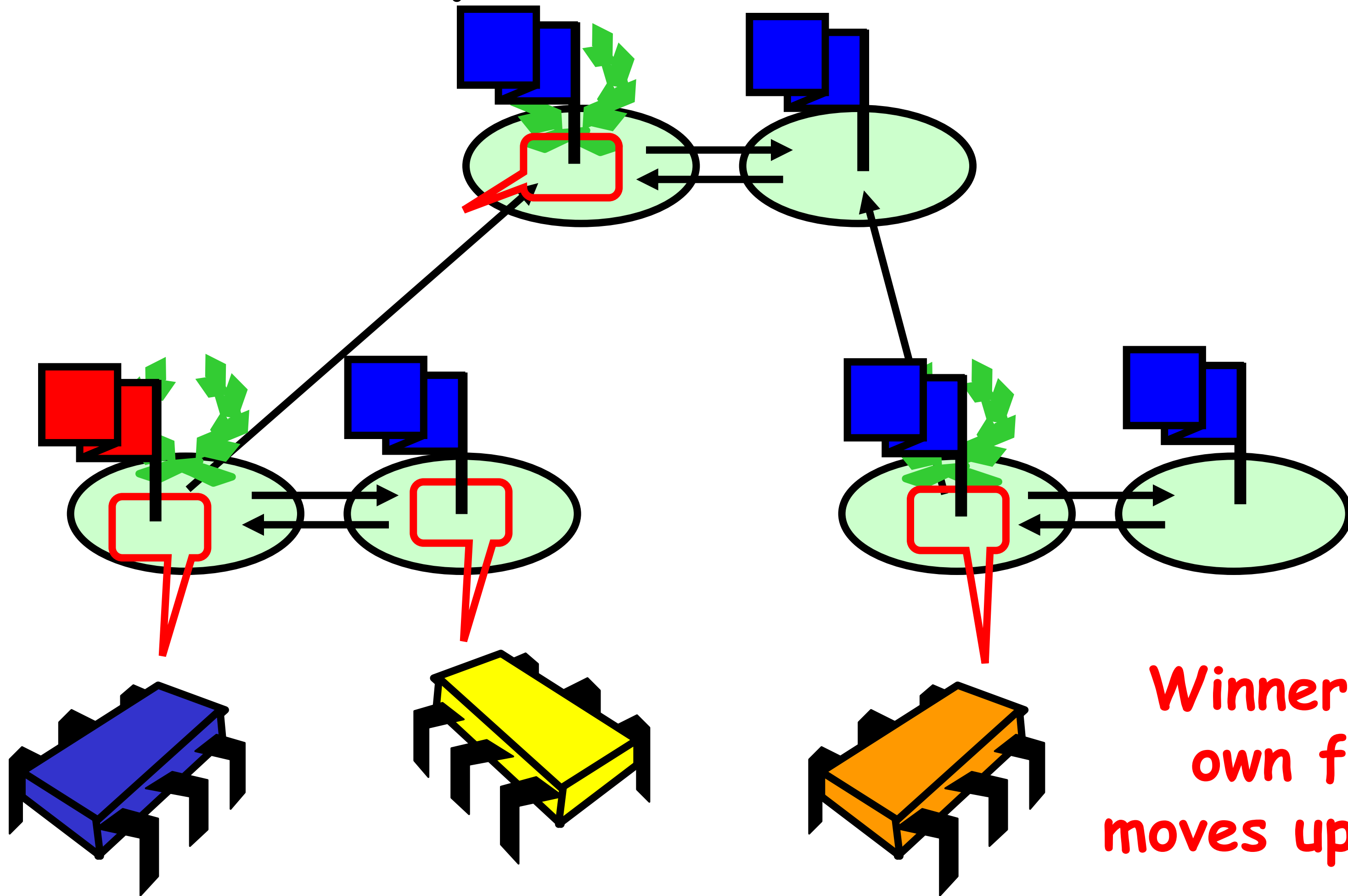
Tournament Tree Barriers



Tournament Tree Barriers

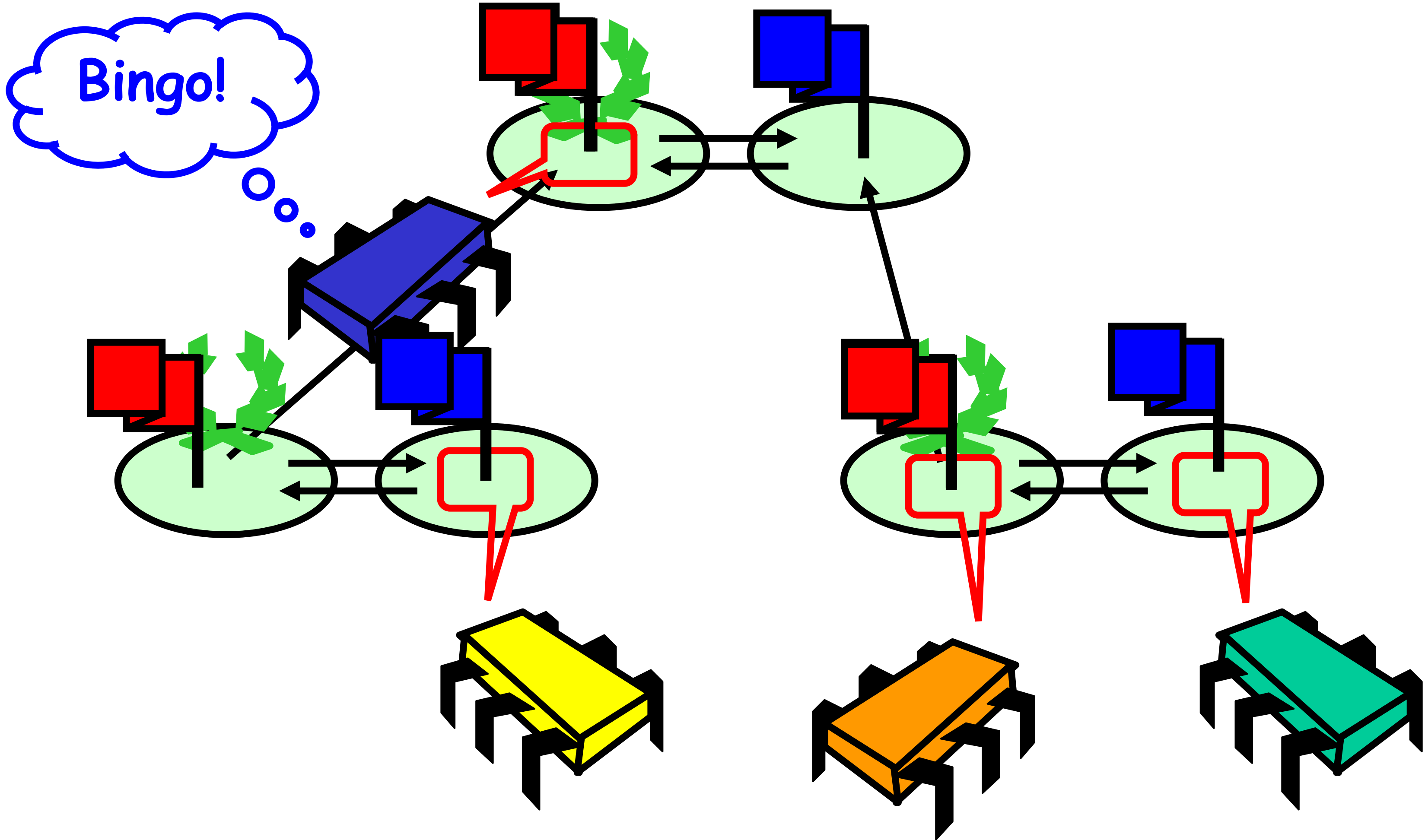


Tournament Tree Barriers

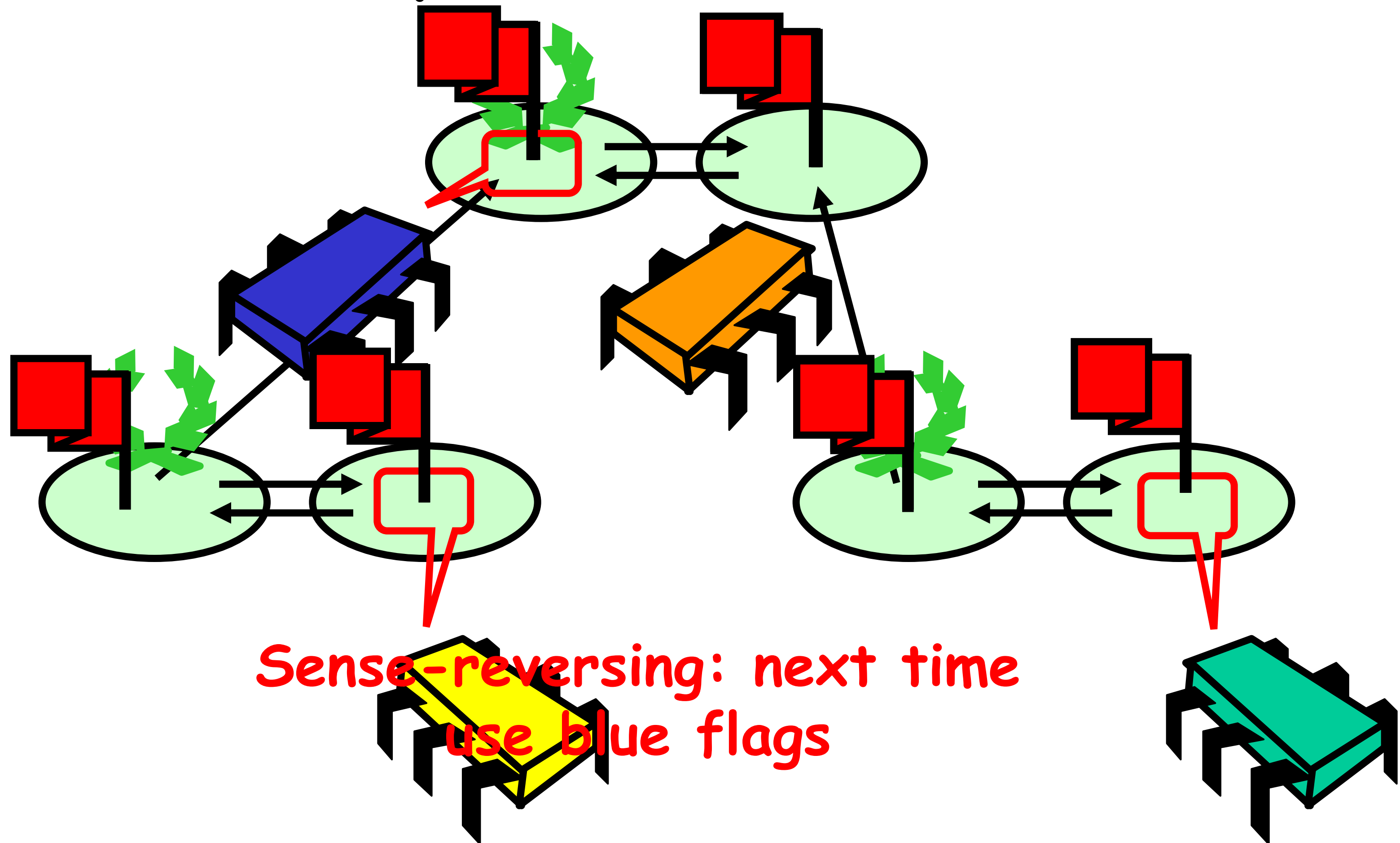


Winner sees own flag, moves up, spins

Tournament Tree Barriers



Tournament Tree Barriers



Tournament Barrier

```
class TBarrier {
  boolean flag;
  TBarrier partner;
  TBarrier parent;
  boolean top;
  …
}
```

Tournament Barrier

```
class TBarrier {
  boolean flag;
  TBarrier partner;
  TBarrier parent;
  boolean top;
  …
}
```

Tournament Barrier

```
class TBarrier {
  boolean flag;
  TBarrier partner;
  TBarrier parent;
  boolean top;
  …
}
```

Tournament Barrier

```
class TBarrier {
  boolean flag;
  TBarrier partner;
  TBarrier parent;
  boolean top;
  …
}
```

Parent (winner) or null (loser)

Tournament Barrier

```
class TBarrier {
  boolean flag;
  TBarrier partner;
  TBarrier parent;
  boolean top;
  …
}
```

Tournament Barrier

```
void await(boolean mySense) {
 if (this.top) {
  return;
 } else if (this.parent != null) {
  while (this.flag != mySense) {};
  this.parent.await(mySense);
  this.partner.flag = mySense;
 } else {
  this.partner.flag = mySense;
  while (this.flag != mySense) {};
}}}
```

Tournament Barrier

```
void await(boolean mySense) {
 if (this.top) {
  return;
 } else if (this.parent != null) {
  while (this.flag != mySense) {};
  this.parent.await(mySense);
  this.partner.flag = mySense;
 } else {
  this.partner.flag = mySense;
  while (this.flag != mySense) {};
}}}
```

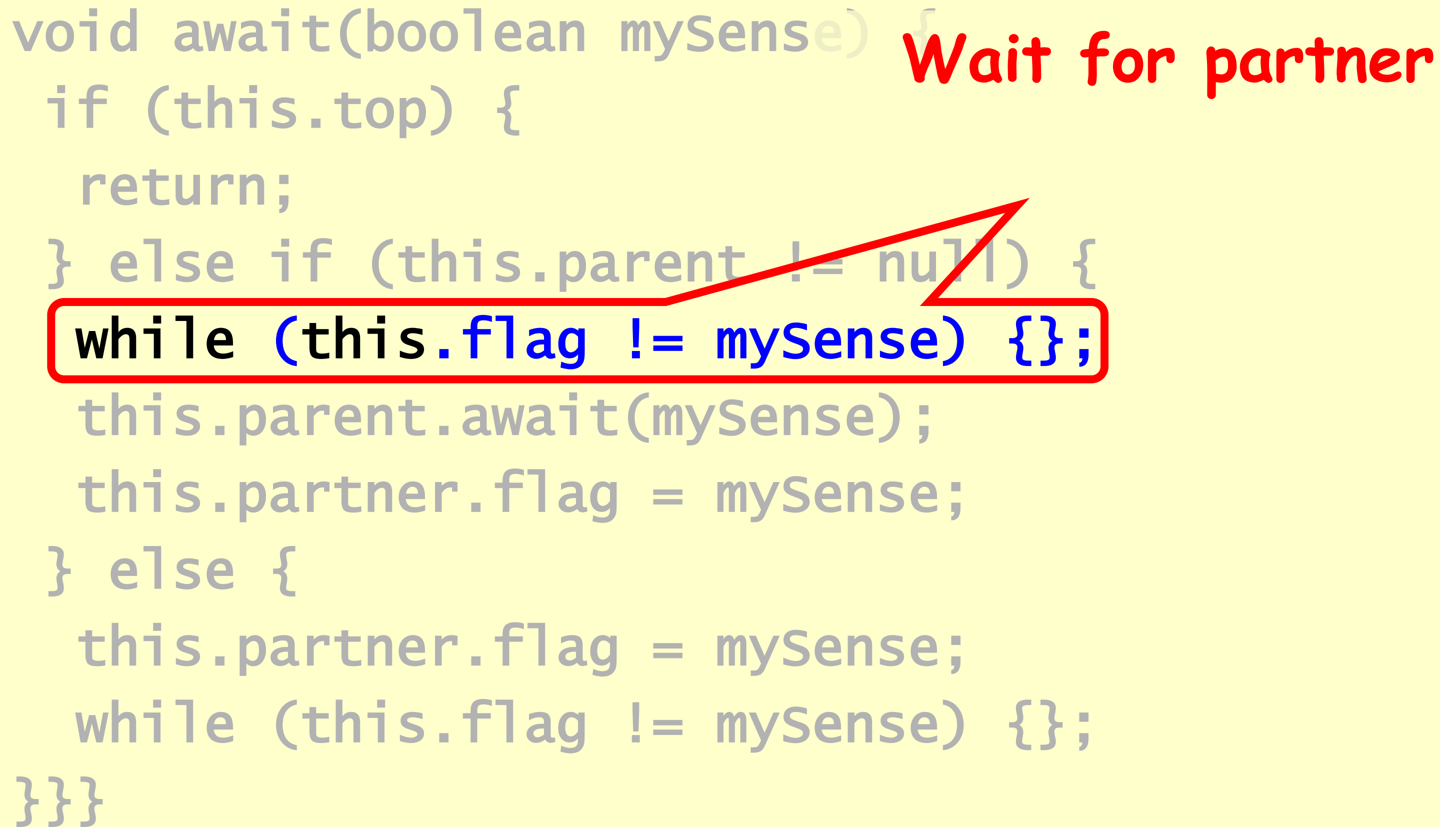
Le root, c'est moi

Tournament Barrier

```
void await(boolean mySense) {
 if (this.top) {
  return;
 } else if (this.parent != null) {
  while (this.flag != mySense) {};
  this.parent.await(mySense);
  this.partner.flag = mySense;
 } else {
  this.partner.flag = mySense;
  while (this.flag != mySense) {};
}}}
```

I am already a winner

Tournament Barrier

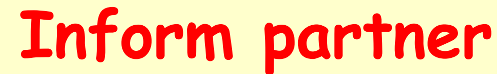


Tournament Barrier

```
void await(boolean mySense) {
 if (this.top) {
  return;
 } else if (this.parent != null) {
  while (this.flag != mySense) {};
  this.parent.await(mySense);
  this.partner.flag = mySense;
 } else {
  this.partner.flag = mySense;
  while (this.flag != mySense) {};
}}}
```

Tournament Barrier

```
void await(boolean mySense) {
 if (this.top) {
  return;
 } else if (this.parent != null) {
  while (this.flag != mySense) {};
  this.parent.await(mySense);
  this.partner.flag = mySense;
 } else {
  this.partner.flag = mySense;
  while (this.flag != mySense) {};
}}}
```



Tournament Barrier

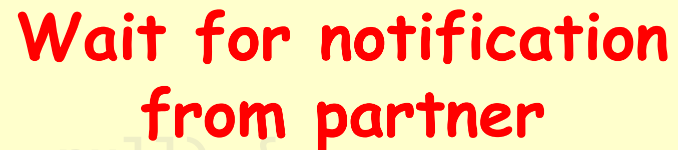
```
void await(boolean mySense) {
 if (this.top) {
  return;
 } else if (this.parent != null) {
  while (this.flag != mySense) {};
  this.parent.await(mySense);
  this.partner.flag = mySense;
 } else {
  this.partner.flag = mySense;
  while (this.flag != mySense) {};
}}}
```

Tournament Barrier

```
void await(boolean mySense) {
 if (this.top) {
  return;
 } else if (this.parent != null) {
  while (this.flag != mySense) {};
  this.parent.await(mySense);
  this.partner.flag = mySense;
 } else {
  this.partner.flag = mySense;
  while (this.flag != mySense) {};
}}}
```

Tournament Barrier

```
void await(boolean mySense) {
 if (this.top) {
  return;
 } else if (this.parent != null) {
  while (this.flag != mySense) {};
  this.parent.await(mySense);
  this.partner.flag = mySense;
 } else {
  this.partner.flag = mySense;
  while (this.flag != mySense) {};
}}}
```



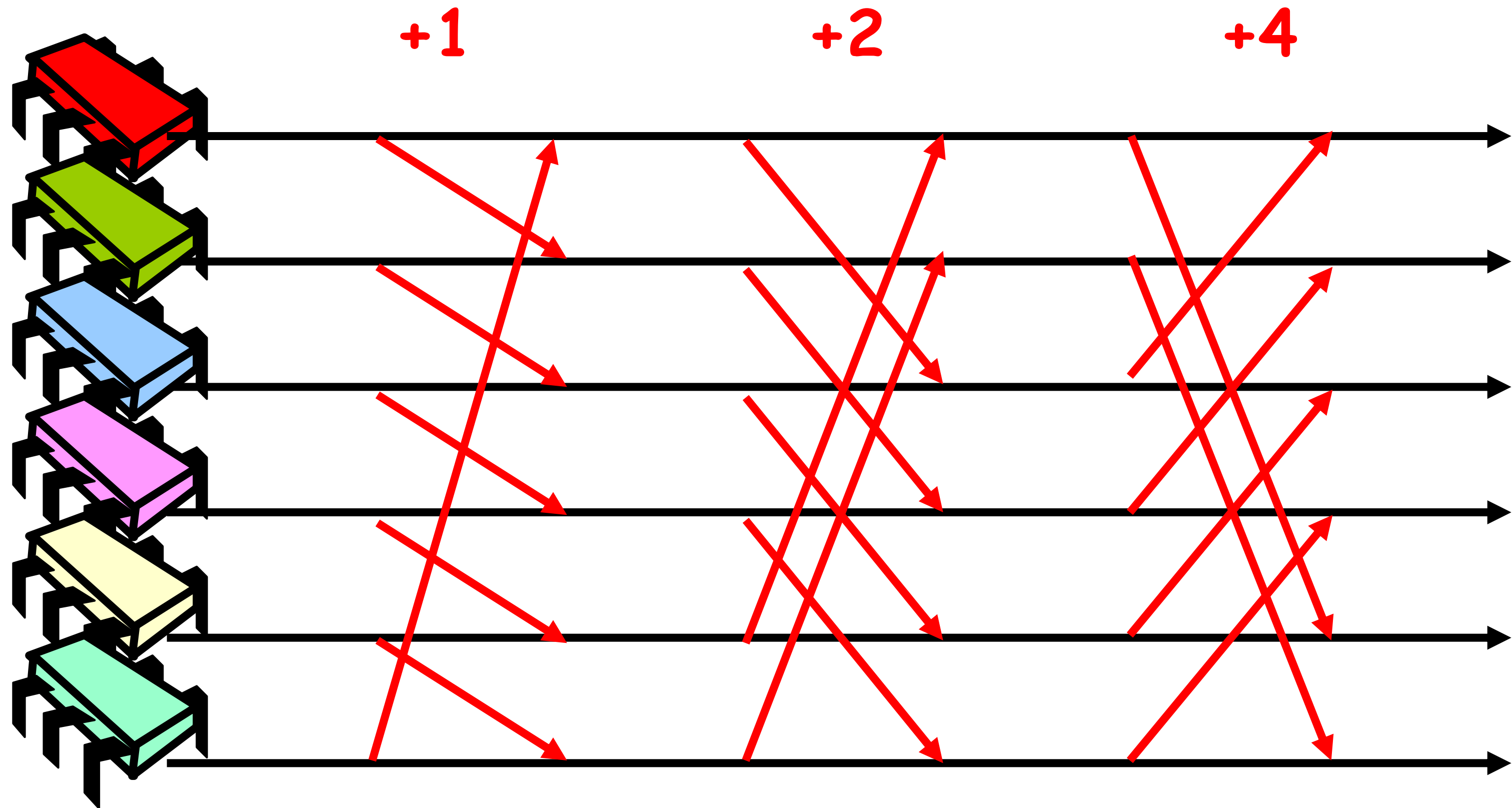
Remarks

- No need for read-modify-write calls
- Each thread spins on fixed location
  - Good for bus-based architectures
  - Good for NUMA architectures

Dissemination Barrier

- At round i
  - Thread A notifies thread $A+2^i \pmod n$
- Requires log n rounds

Dissemination Barrier



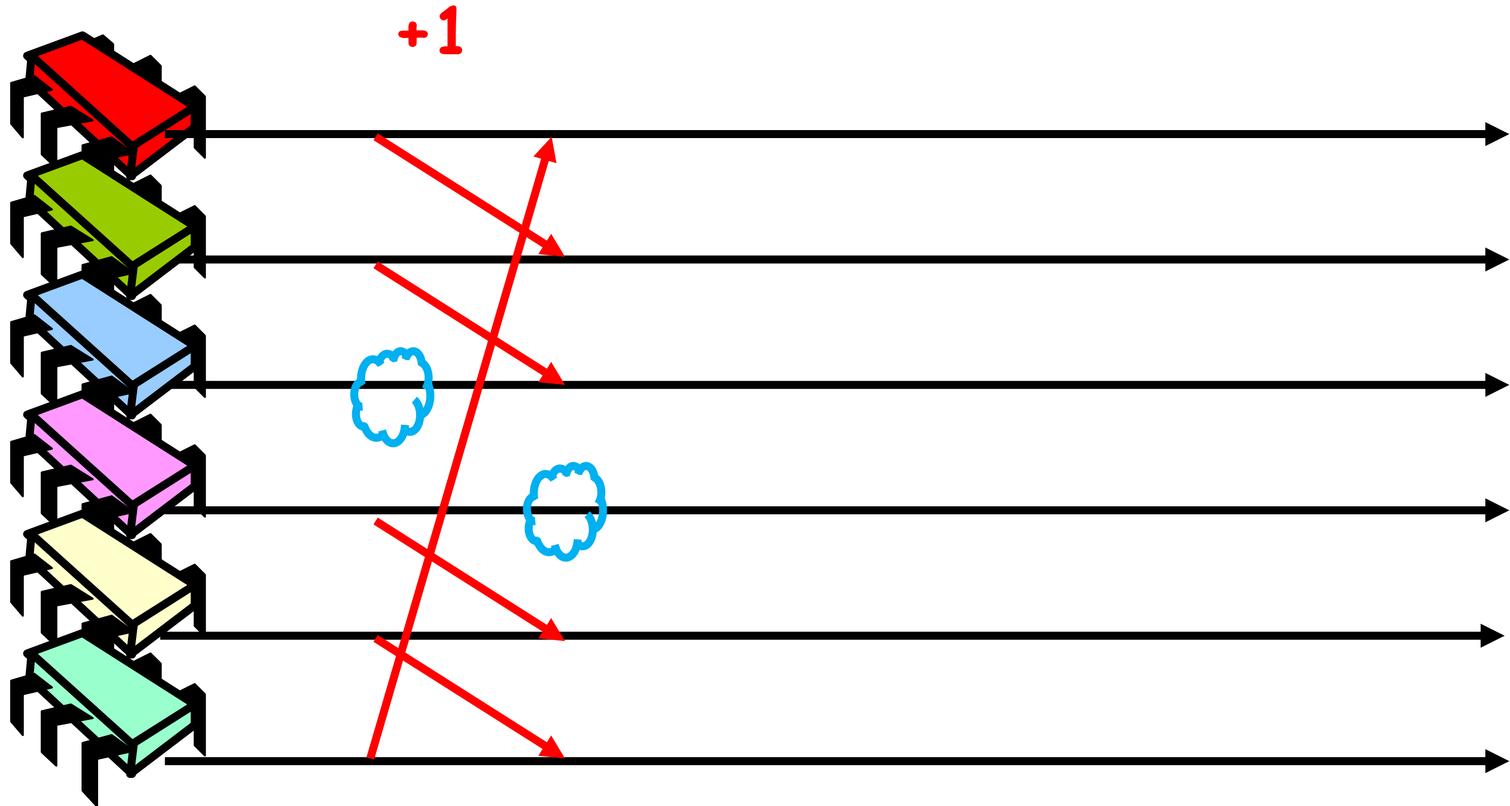
Dissemination Barrier

- If a thread gets delayed,
  - 2 will be delayed at round 1
  - 4 will be delayed at round 2 . . .

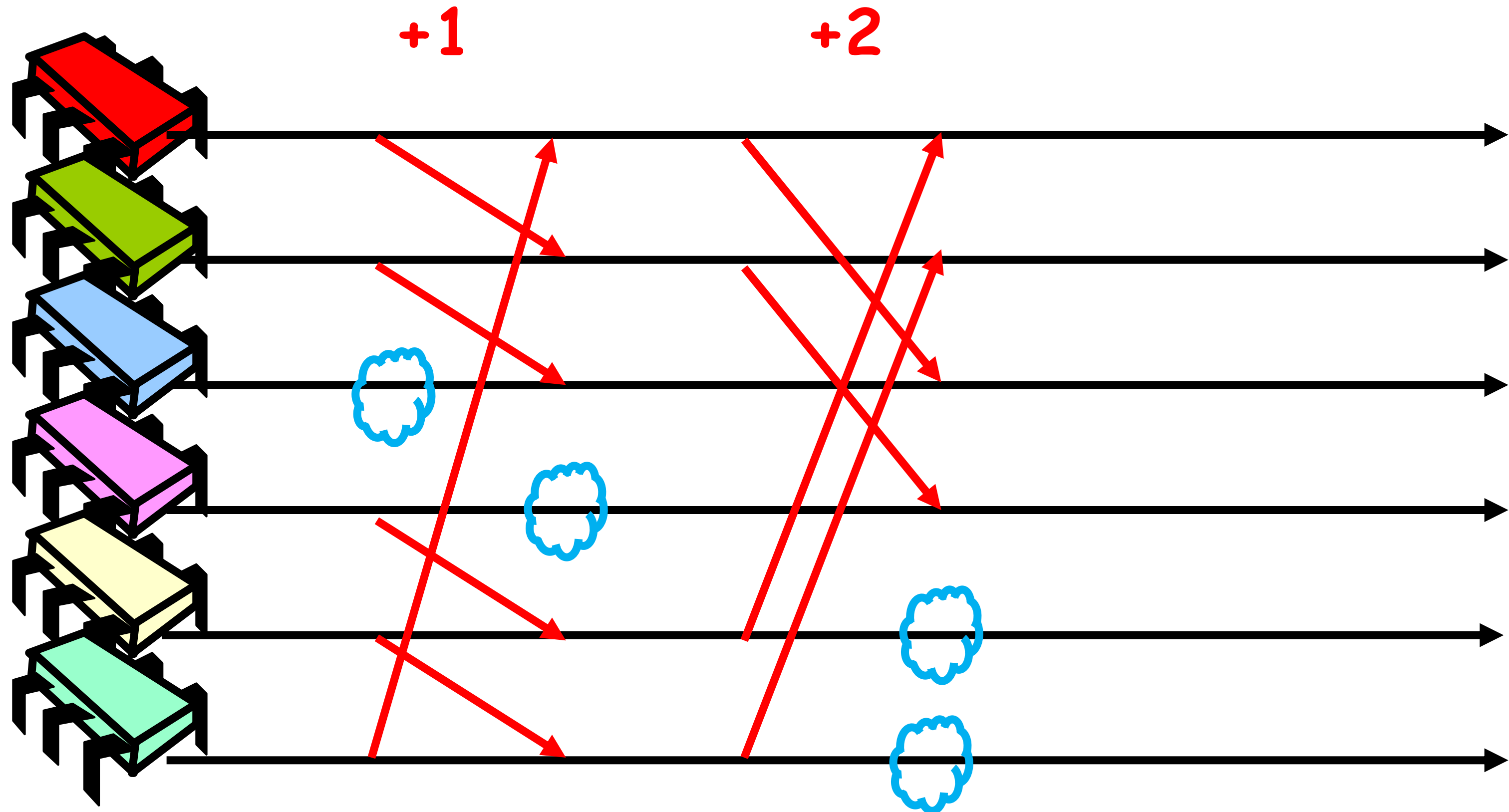
  - 2^i will be delayed at round i (2^i >= n)

- i <= $\log_2 n$

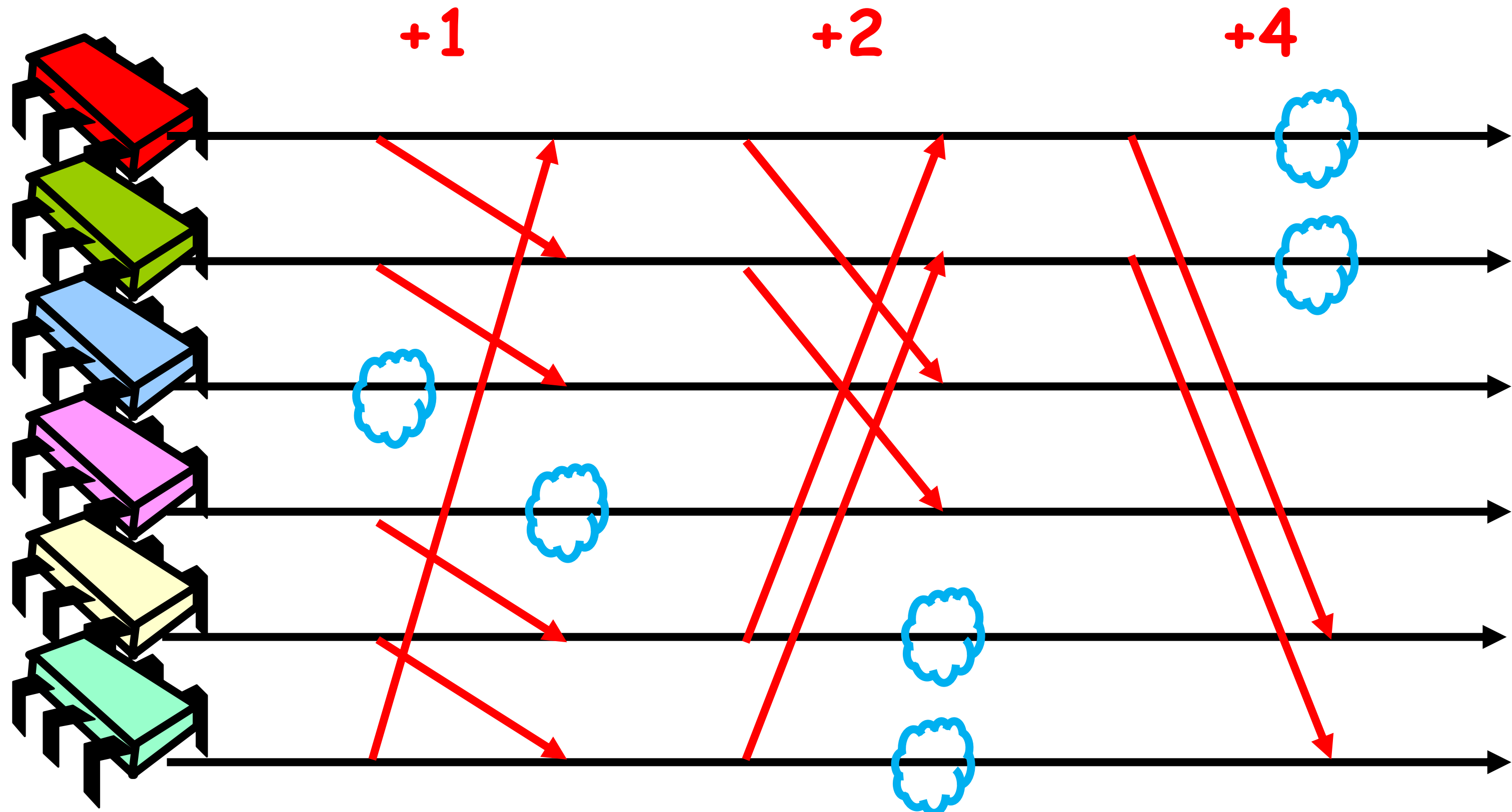
Dissemination Barrier



Dissemination Barrier



Dissemination Barrier



Dissemination Details

- Use two copies of flags
  - Avoids interference
- Use sense-reversing
  - Avoid reinitializing fields
- Thread Arguments
  - Parity of round
  - Sense (flips when round becomes 0)

Dissemination Barrier

```
public class DisBarrier {
 private class Node {
  boolean[] flag = {false, false};
  Node      partner = null;
 }
  private Node[][] nodes;
  …
}
```

Dissemination Barrier

```
public class DisBarrier {
 private class Node {
  boolean[] flag = {false, false};
  Node      partner = null;
 }
  private Node[][] nodes;
  …
}
```

Inner class defines node

Dissemination Barrier

```
public class DisBarrier {
 private class Node {
  boolean[] flag = {false, false};
  Node      partner = null;
 }
  private Node[][] nodes;
  …
}
```

Flags for even and odd phases

Constructor

```
public DisBarrier(int n) {
 nodes = new Node[n][logn];
 int d = 1;
 for (int i = 0; i < n; i++) {
  for (int j = 0; j < logn; j++)
   nodes[i][j].partner
    = nodes[(i+d) % n][j];
   d = 2 * d;
 }}
```

Constructor

```
public DisBarrier(int n) {
 nodes = new Node[n][logn];
 int d = 1;
 for (int i = 0; i < n; i++) {
  for (int j = 0; j < logn; j++)
   nodes[i][j].partner
    = nodes[(i+d) % n][j];
   d = 2 * d;
 }}
```

Array: threads X phases

Constructor

```
public DisBarrier(int n) {
 nodes = new Node[n][logn];
 int d = 1;
 for (int i = 0; i < n; i++) {
  for (int j = 0; j < logn; j++)
   nodes[i][j].partner
    = nodes[(i+d) % n][j];
   d = 2 * d;
 }}
```

Point to partner for this phase

Method

```
void await(int parity,
           boolean mySense) {
 int i = Thread.myIndex();
 for (int r = 0; r < logn; r++) {
  nodes[i][r].partner.flag[parity]
   = mySense;
  while (nodes[i][r].flag[parity]
        != mySense) {}
   }}
```

Method

```
void await(int parity,
            boolean mySense) {
 int i = Thread.myIndex();
 for (int r = 0; r < logn; r++) {
  nodes[i][r].partner.flag[parity]
   = mySense;
  while (nodes[i][r].flag[parity]
          != mySense) {}
   }}
```

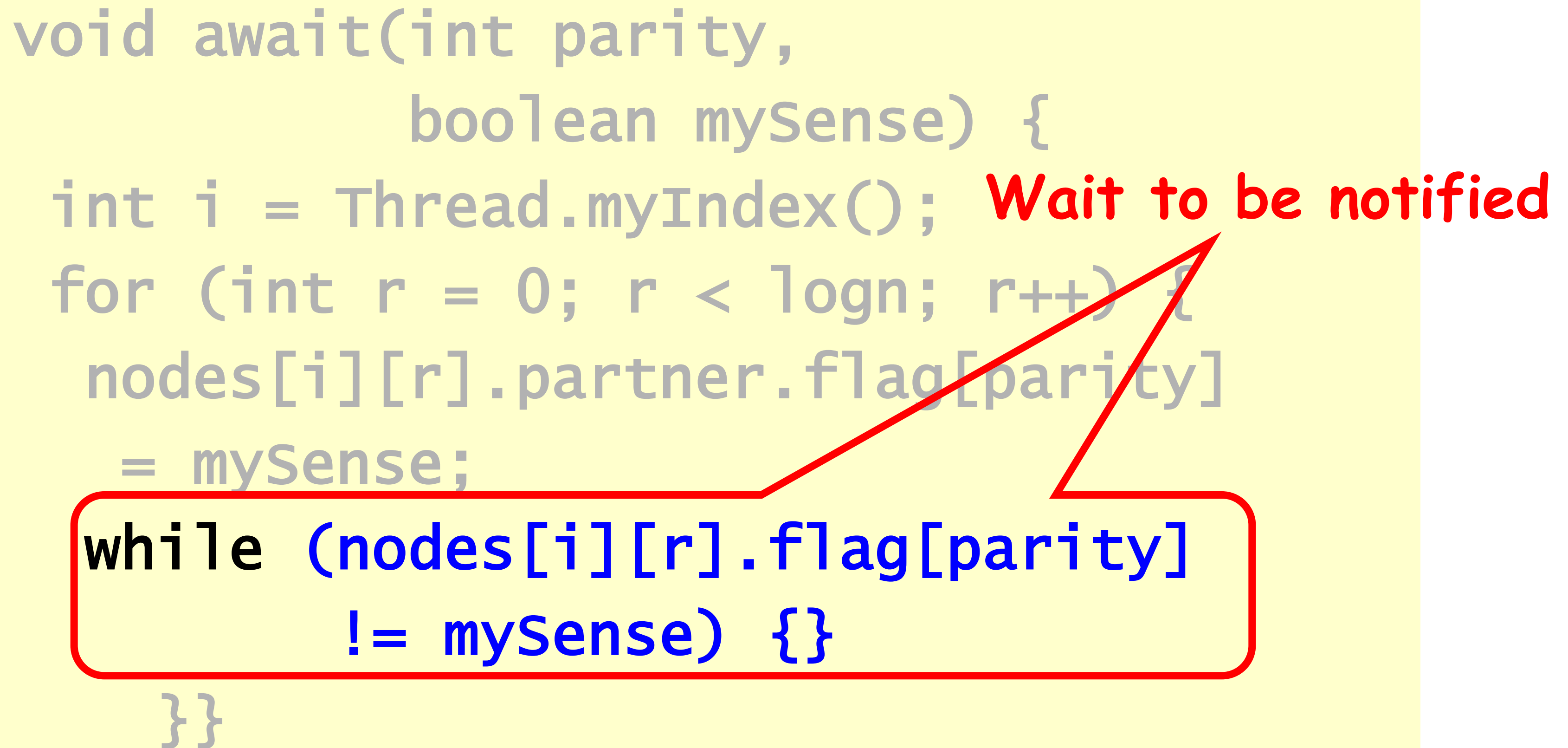
For log number of rounds …

Method

```
void await(int parity,
           boolean mySense) {
 int i = Thread.myIndex();
 for (int r = 0; r < logn; r++) {
  nodes[i][r].partner.flag[parity]
   = mySense;
  while (nodes[i][r].flag[parity]
          != mySense) {}
   }}
```

Notify partner
(use parity to pick flag)

Method



Remarks

- Every thread spins in the same place
  - Good for NUMA implementations
- Works even if n not a power of 2
- Not very space efficient

Ideas

- Sense-reversing
  - Reuse without reinitializing
- Combining tree
  - Like counters, locks …
- Tournament tree
  - Optimized combining tree
- Dissemination barrier
  - Simple, not space-efficient

SOME RIGHTS RESERVED

This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 License.

- **You are free**:
  - **to Share** — to copy, distribute and transmit the work
  - **to Remix** — to adapt the work
- **Under the following conditions**:
  - **Attribution.** You must attribute the work to "The Art of Multiprocessor Programming" (but not in any way that suggests that the authors endorse you or your use of the work).
  - **Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.
- For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to
  - http://creativecommons.org/licenses/by-sa/3.0/.
- Any of the above conditions can be waived if you get permission from the copyright holder.
- Nothing in this license impairs or restricts the author's moral rights.