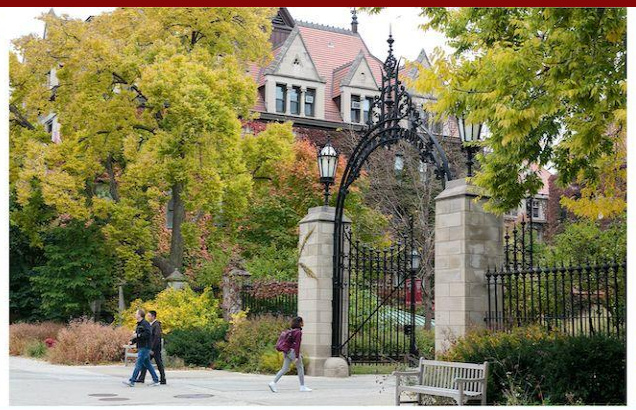
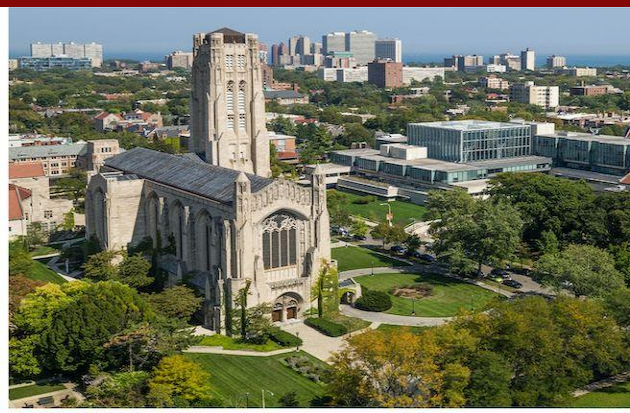
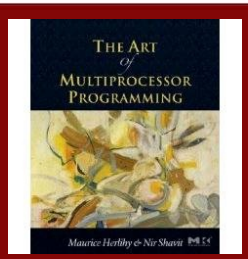
MPCS 52060 - Parallel Programming
M6: Advanced Parallel Patterns and Techniques (Part 2)



Original slides from "The Art of Multiprocessor Programming" by Maurice Herlihy & Nir Shavit with modifications by Lamont Samuels

Data Parallelism

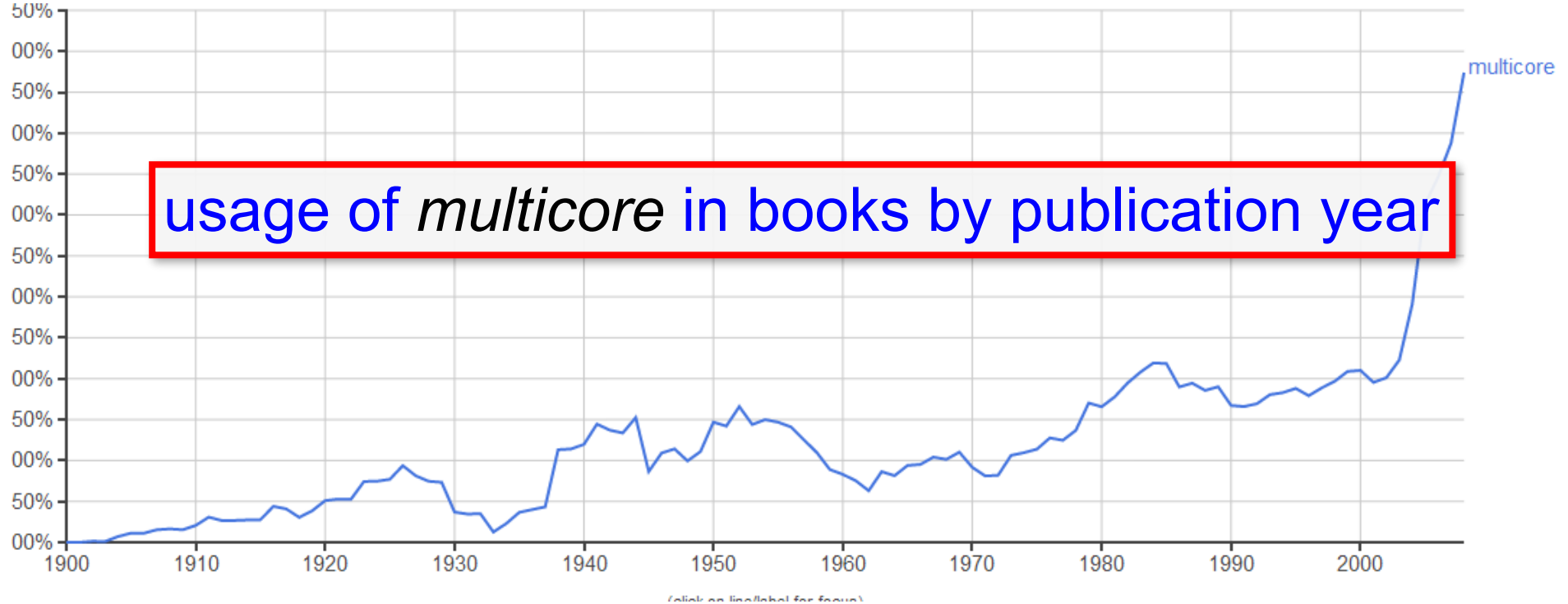
Ever Wonder …

When did the term *multicore* become popular?

A multi-core processor is a single computing component with two or more independent actual central processing units, which are the units that read and execute program instructions.)

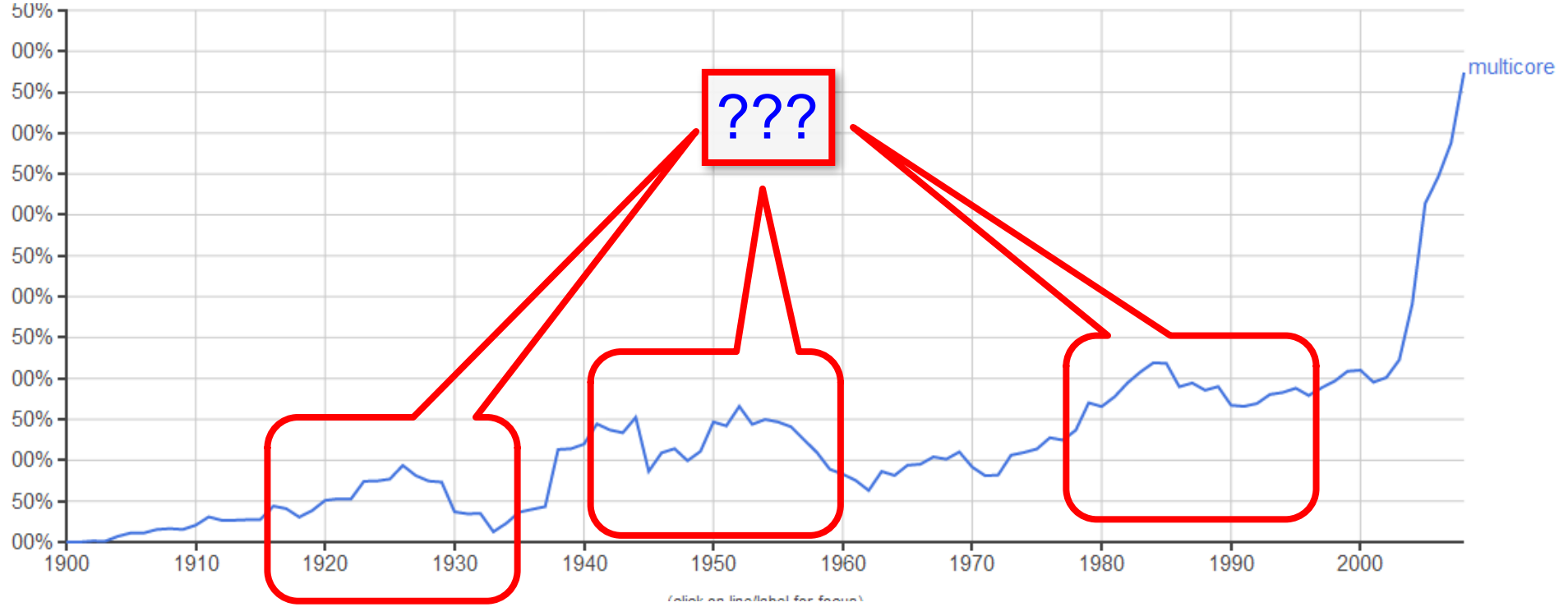
wikipedia

Let's Ask Google Ngram!

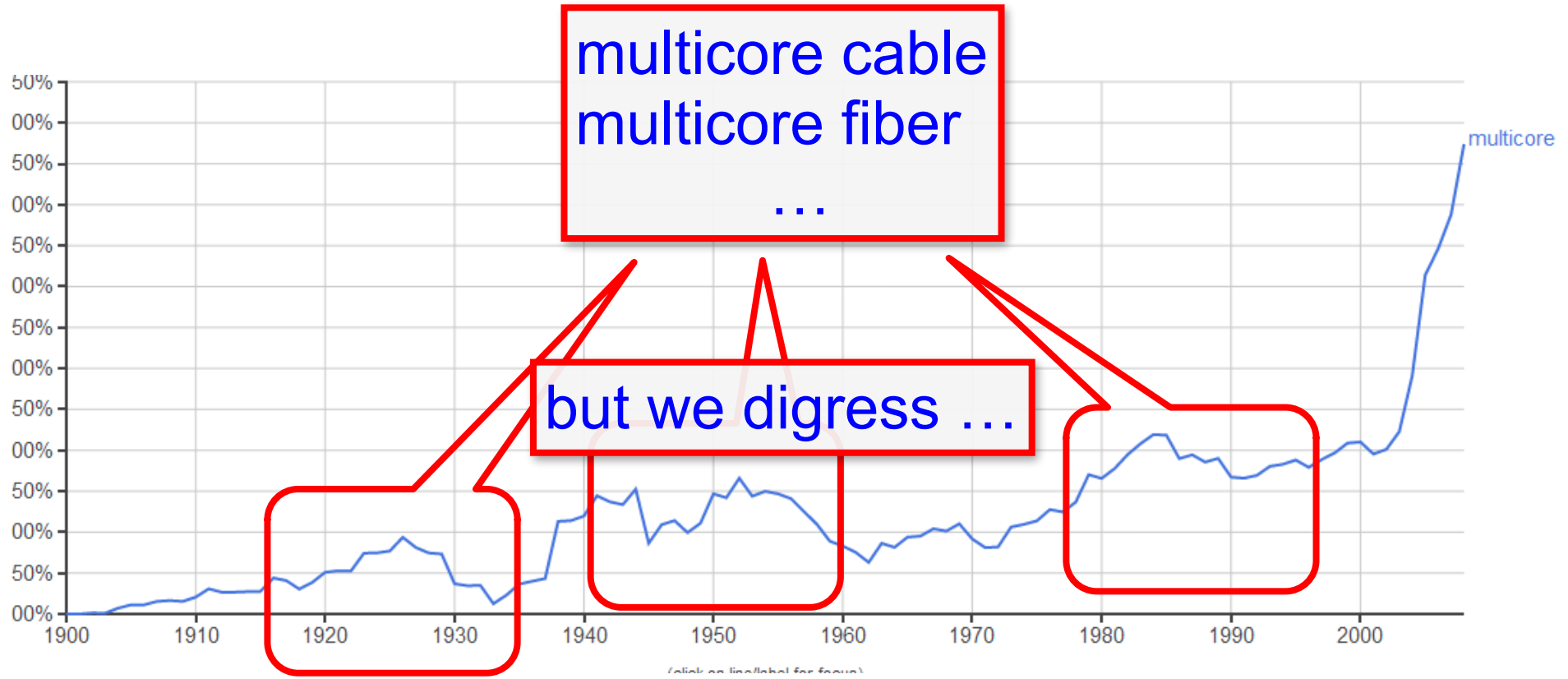


Let's Ask Google Ngram!

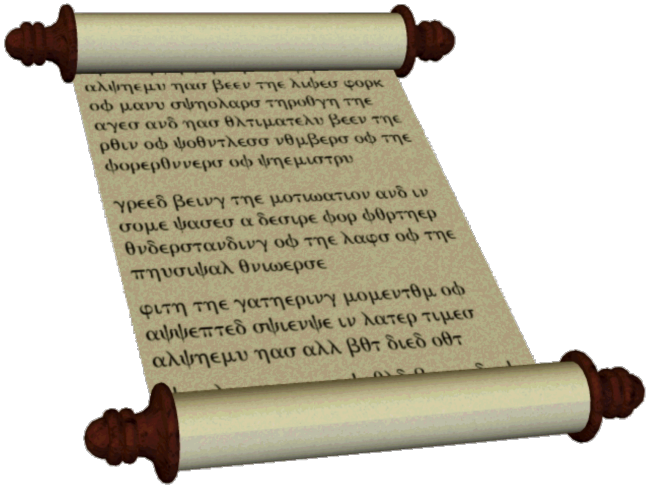
Let's Ask Google Ngram!

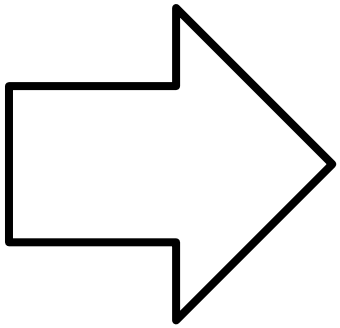


Let's Ask Google Ngram!



WordCount





alpha → 8

bravo → 3

charlie → 9

…

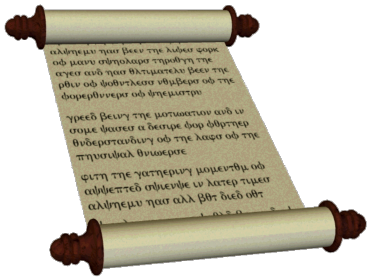
zulu → 1

easy to do sequentially …

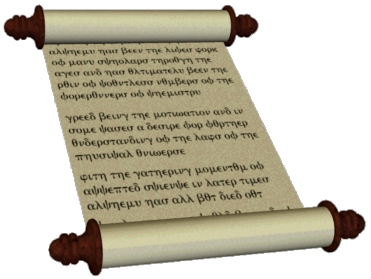
what about in parallel?

MapReduce

split text among *mapping* threads

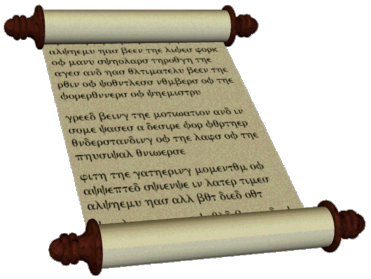


chapter 1



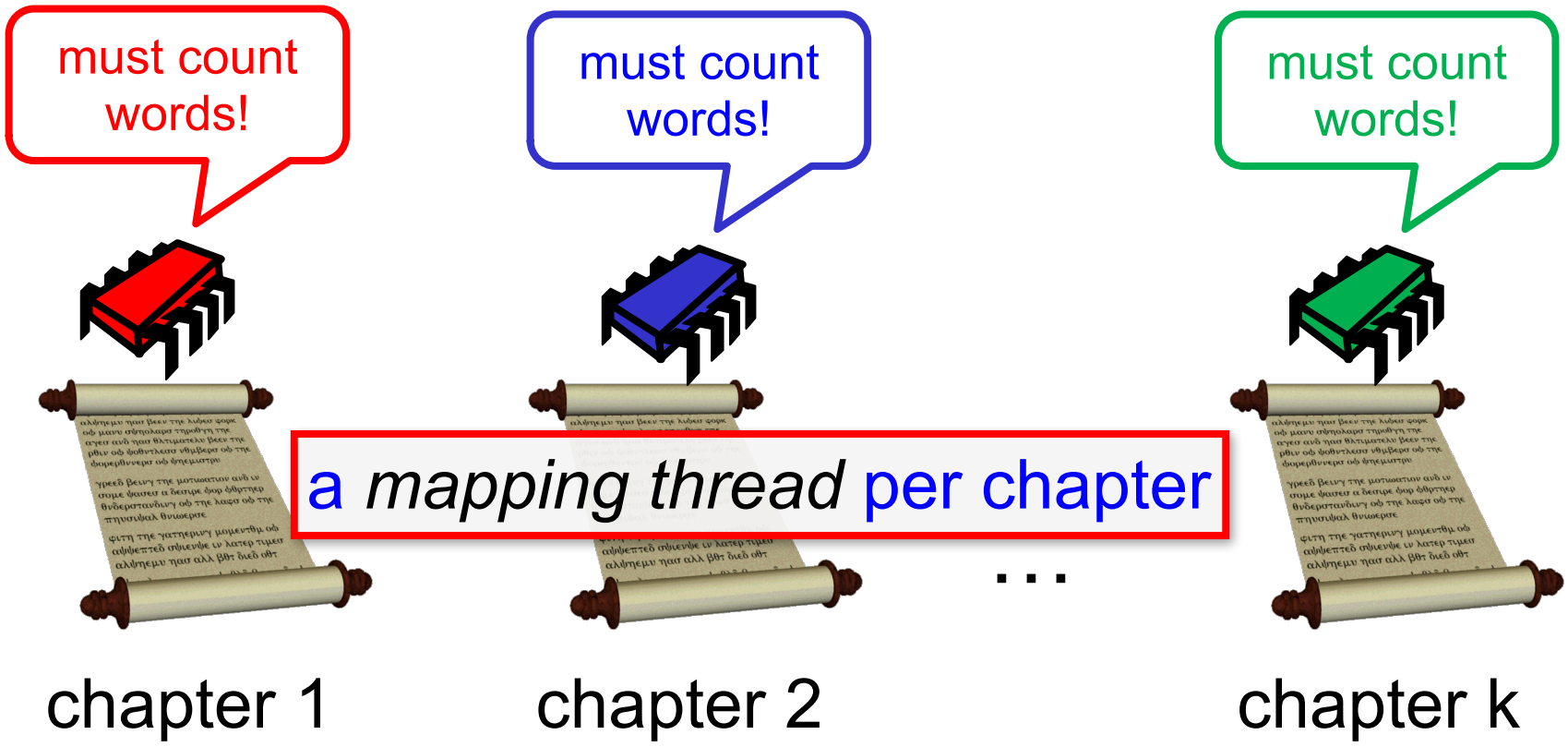
chapter 2

...



chapter k

Map Phase


must count words!
must count words!
must count words!
a *mapping thread* per chapter
...
chapter 1
chapter 2
chapter k

Map Phase

chapter 1

alpha → 9
juliet → 2,
alpha → 1
tango → 4

each mapper thread produces a *stream* …

of *key-value* pairs …

key: word

value: local count

Mapper Class

```
abstract class Mapper<IN, K, V>
    extends RecursiveTask<Map<K, V>> {
  IN input;
  public void setInput(IN anInput) {
    input = anInput;
  }
}
```

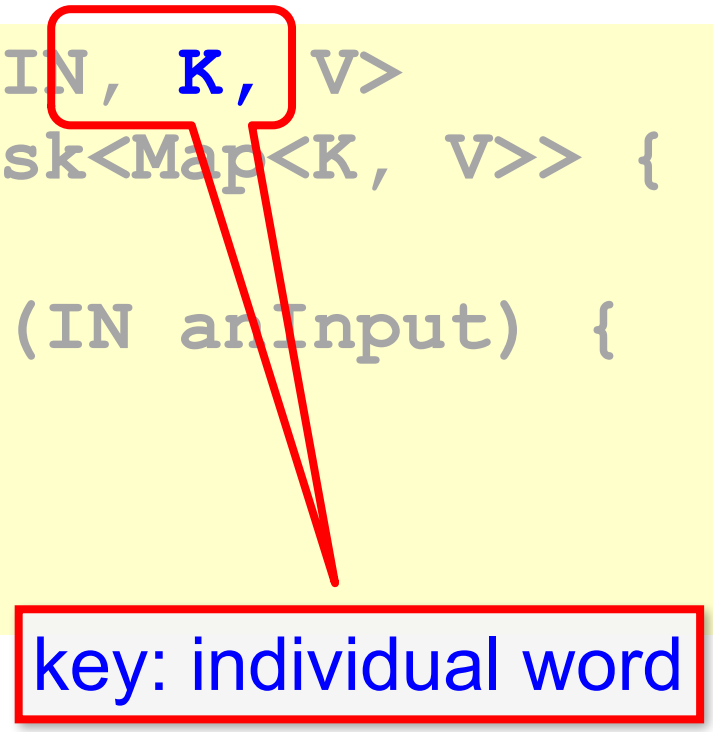
Mapper Class

```
abstract class Mapper<IN, K, V>
    extends RecursiveTask<Map<K, V>> {
  IN input;
  public void setInput(IN anInput) {
    input = anInput;
  }
}
```

input: document fragment

Mapper Class

```
abstract class Mapper<IN, K, V>
    extends RecursiveTask<Map<K, V>> {
  IN input;
  public void setInput(IN anInput) {
    input = anInput;
  }
}
```



Mapper Class

```
abstract class Mapper<IN, K, V>
    extends RecursiveTask<Map<K, V>> {
   IN input;
   public void setInput(IN anInput) {
      input = anInput;
   }
}
```

value: local count

Mapper Class

```
abstract class Mapper<IN, K, V>
    extends RecursiveTask<Map<K, V>> {
  IN input;
  public void setInput(IN anInput) {
    input = anInput;
  }
}
```

a task that runs in parallel with other tasks

Mapper Class

```
abstract class Mapper<IN, K, V>
    extends RecursiveTask<Map<K, V>> {
  IN input;
  public void setInput(IN anInput) {
    input = anInput;
  }
}
```

produces a map: word → count

Mapper Class

```
abstract class Mapper<IN, K, V>
    extends RecursiveTask<Map<K, V>> {
  IN input;
  public void setInput(IN anInput) {
    input = anInput;
  }
}
```

initialize input: which document fragment?

WordCount Mapper

```
class WordCountMapper extends
  mapreduce.Mapper<
      List<String>, String, Long
    > {
  …
  }
```

WordCount Mapper

```
class WordCountMapper extends
   mapreduce.Mapper<
        List<String>, String, Long
     > {
   …
```

document fragment is list of words

WordCount Mapper

```
class WordCountMapper extends
   mapreduce.Mapper<
        List<String>, String, Long
      > {
   …
```

document fragment is list of words

map each word …

WordCount Mapper

```
class WordCountMapper extends
   mapreduce.Mapper<
        List<String>, String, Long
     > {
   …
```

document fragment is list of words

map each word …

to its count in the fragment

WordCount Mapper

```
Map<String,Long> compute() {
  Map<String,Long> map = new HashMap<>();
      for (String word : input) {
        map.merge(word,
                  1L,
                  (x, y) -> x + y);
      }
      return map;
    }
  }
```

WordCount Mapper

```
Map<String,Long> compute() {
  Map<String,Long> map = new HashMap<>();
    for (String word : input) {
      map.merge(word,
                1L,
                (x, y) -> x + y);
    return map;
  }
}
```

the `compute()` method constructs the local word count

WordCount Mapper

```
Map<String,Long> compute() {
  Map<String,Long> map = new HashMap<>();
    for (String word : input) {
      map.merge(word,
                1L,
                (x, y) -> x + y);
    }
    return map;
  }
}
```

create a map to hold the output

WordCount Mapper

```
Map<String,Long> compute() {
  Map<String,Long> map = new HashMap<>();
    for (String word : input) {
      map.merge(word,
                1L,
                (x, y) -> x + y);
    }
    return map;
  }
}
```

examine each word in the document fragment

WordCount Mapper

```
Map<String,Long> compute() {
  Map<String,Long> map = new HashMap<>();
      for (String word : input) {
        map.merge(word,
                  1L,
                  (x, y) -> x + y);
      }
      return map;
    }
  }
```

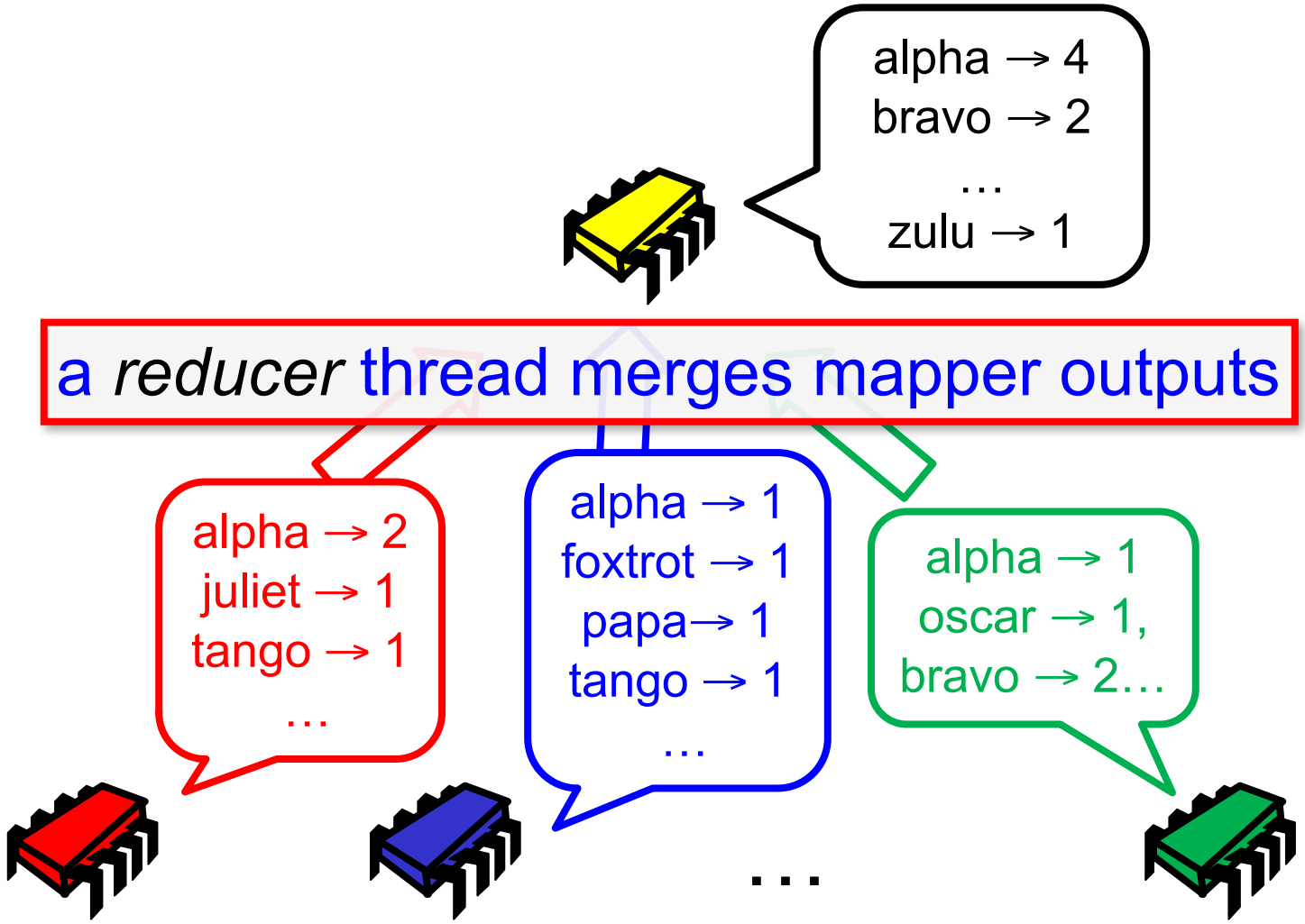
increment that word's count in the map

WordCount Mapper

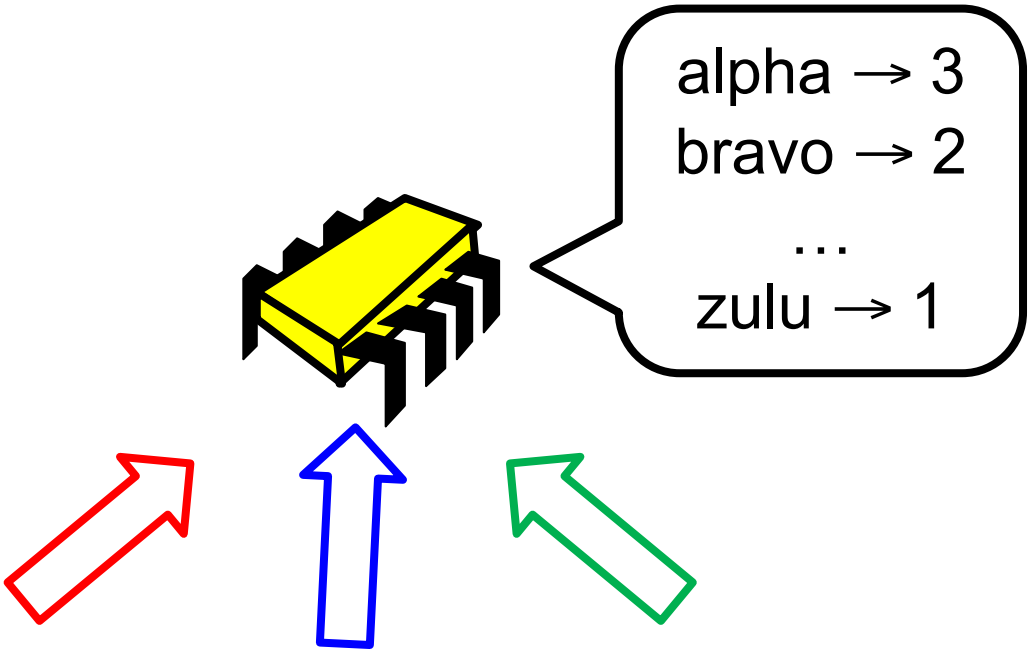
```
Map<String,Long> compute() {
  Map<String,Long> map = new HashMap<>();
      for (String word : input) {
        map.merge(word,
                  1L,
                  (x, y) -> x + y);
      }
      return map;
    }
  }
```

when the local count is complete, return the map

Reduce Phase



Reduce Phase



the reducer task produces a *stream* …

of *key-value* pairs …

key: word

value: word count

Reducer Class

```
abstract class Reducer<K, V, OUT>
    extends RecursiveTask<OUT> {
  K key;
  List<V> valueList;
  public void setInput(
    K aKey,
    List<V> aList) {
    key = aKey;
    valueList = aList;
  }
}
```

Reducer Class

```
abstract class Reducer<K, V, OUT>
    extends RecursiveTask<OUT> {
  K key;
  List<V> valueList;
  public void setInput(
    K aKey,
    List<V> aList) {
    key = aKey;
    valueList = aList;
  }
}
```

each reducer is given a single key (word)

Reducer Class

```
abstract class Reducer<K, V, OUT>
    extends RecursiveTask<OUT> {
  K key;
  List<V> valueList;
  public void setInput(
    K aKey,
    List<V> aList) {
    key = aKey;
    valueList = aList;
  }
}
```

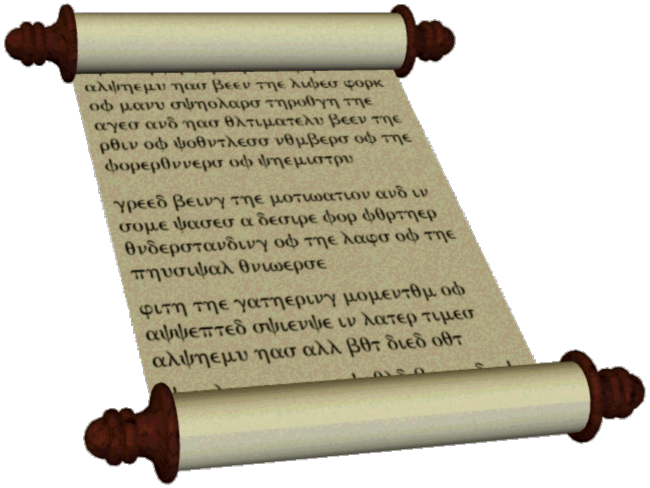
and a list of associated values (word count per fragment)

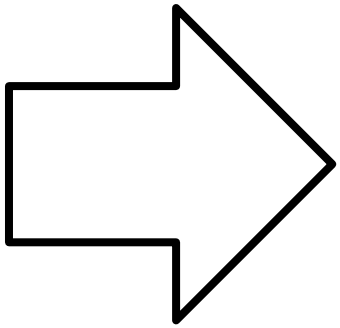
Reducer Class

```
abstract class Reducer<K, V, OUT>
    extends RecursiveTask<OUT> {
  K key;
  List<V> valueList;
  public void setInput(
    K aKey,
    List<V> aList) {
    key = aKey;
    valu
  }
}
```

It produces a single summary value (the total count for that word)

WordCount





$$\begin{pmatrix} 0.037 \\ 0.002 \\ 0.045 \\ \dots \\ 0.000 \end{pmatrix}$$

normalizing document wordcount

gives a *fingerprint* vector

Pseudocode Implementations

- See m6/concurrent/mapreduce/MapReduce.java
- See m6/concurrent/mapreduce/WordCount.java

Barriers

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;
}
```

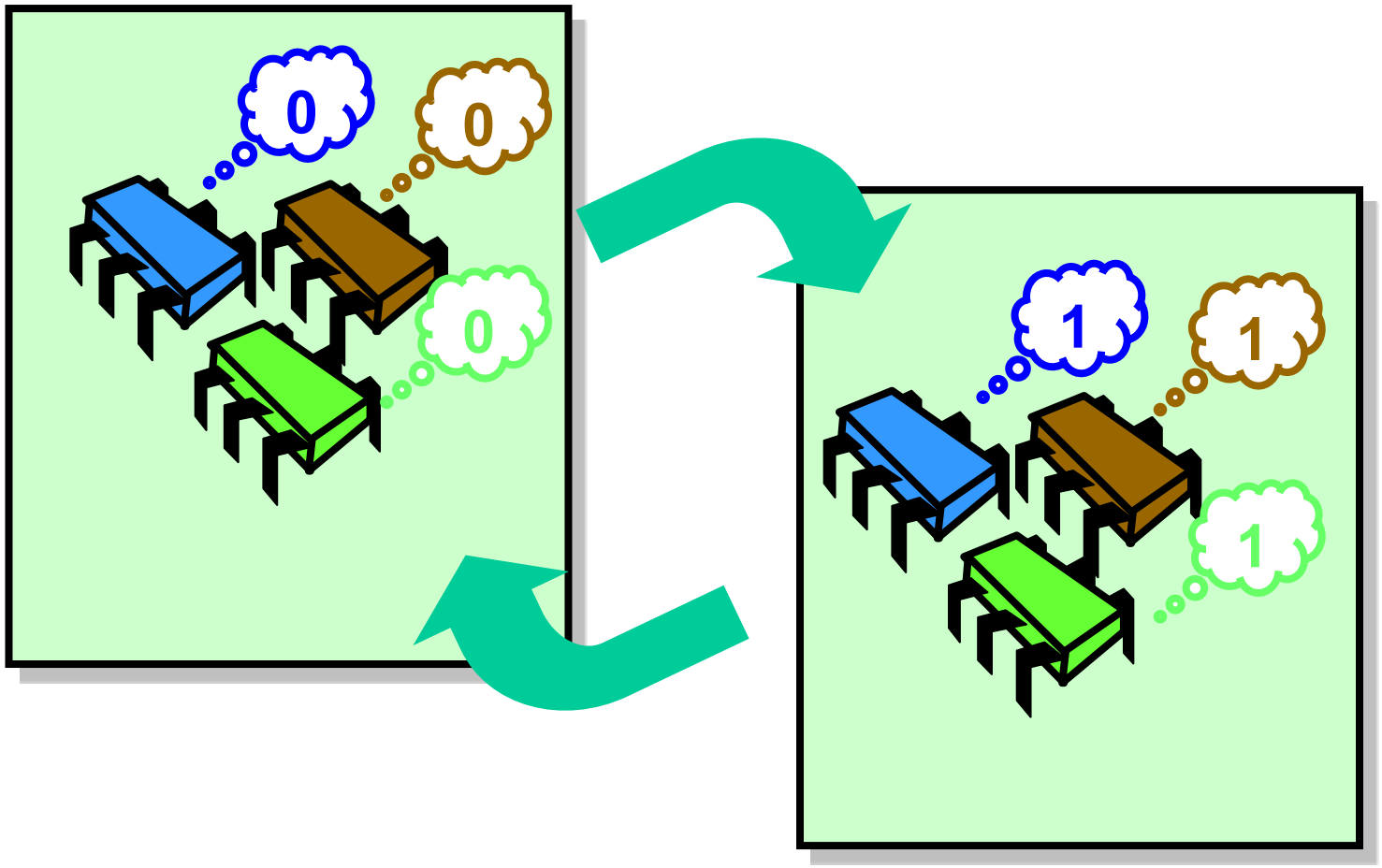
```
while (true) {
  if (phase) {
    frame[1].prepare();
  } else {
    frame[0].prepare();
  }
  phase = !phase;
}
```

odd phases

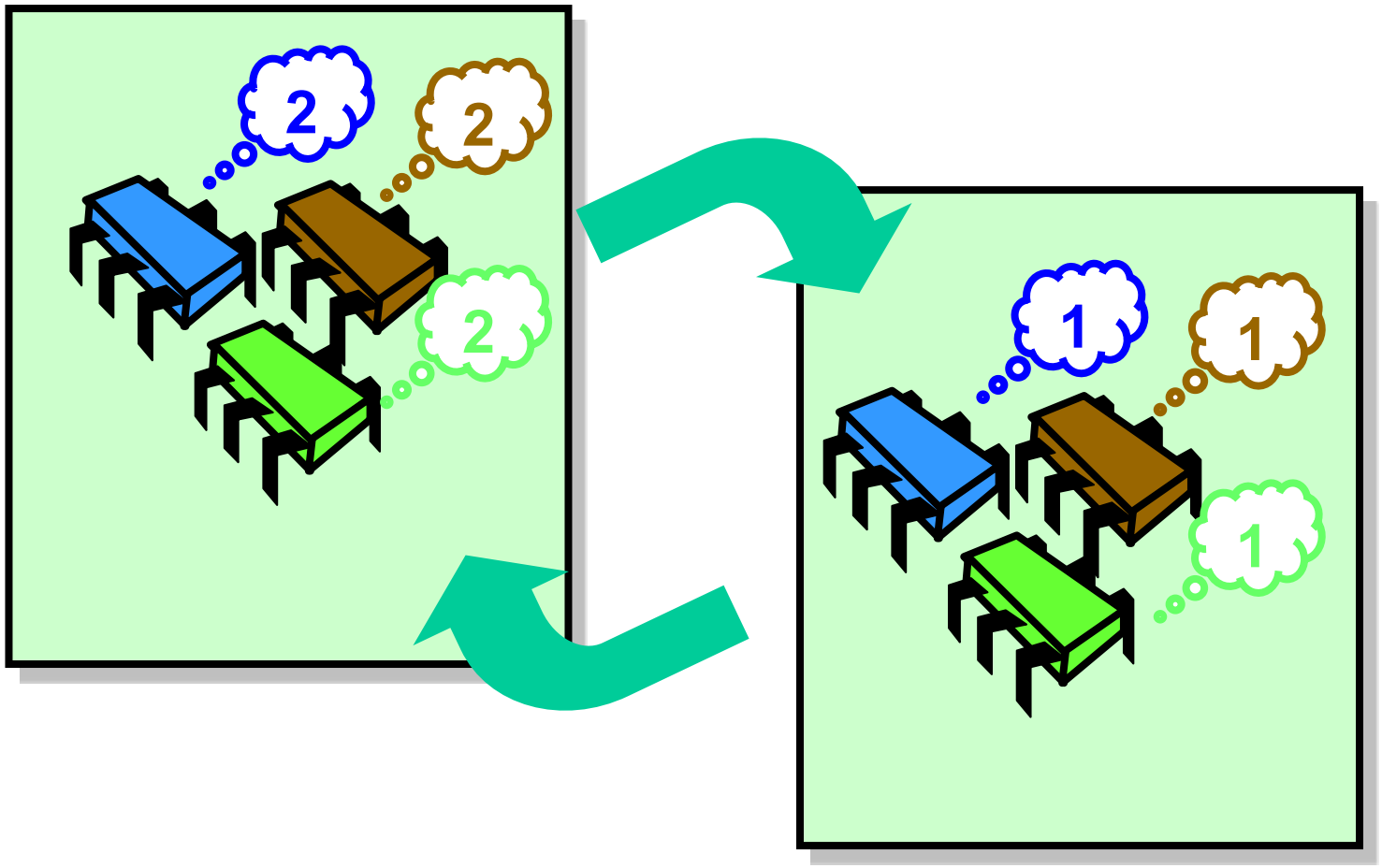
Synchronization Problems

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

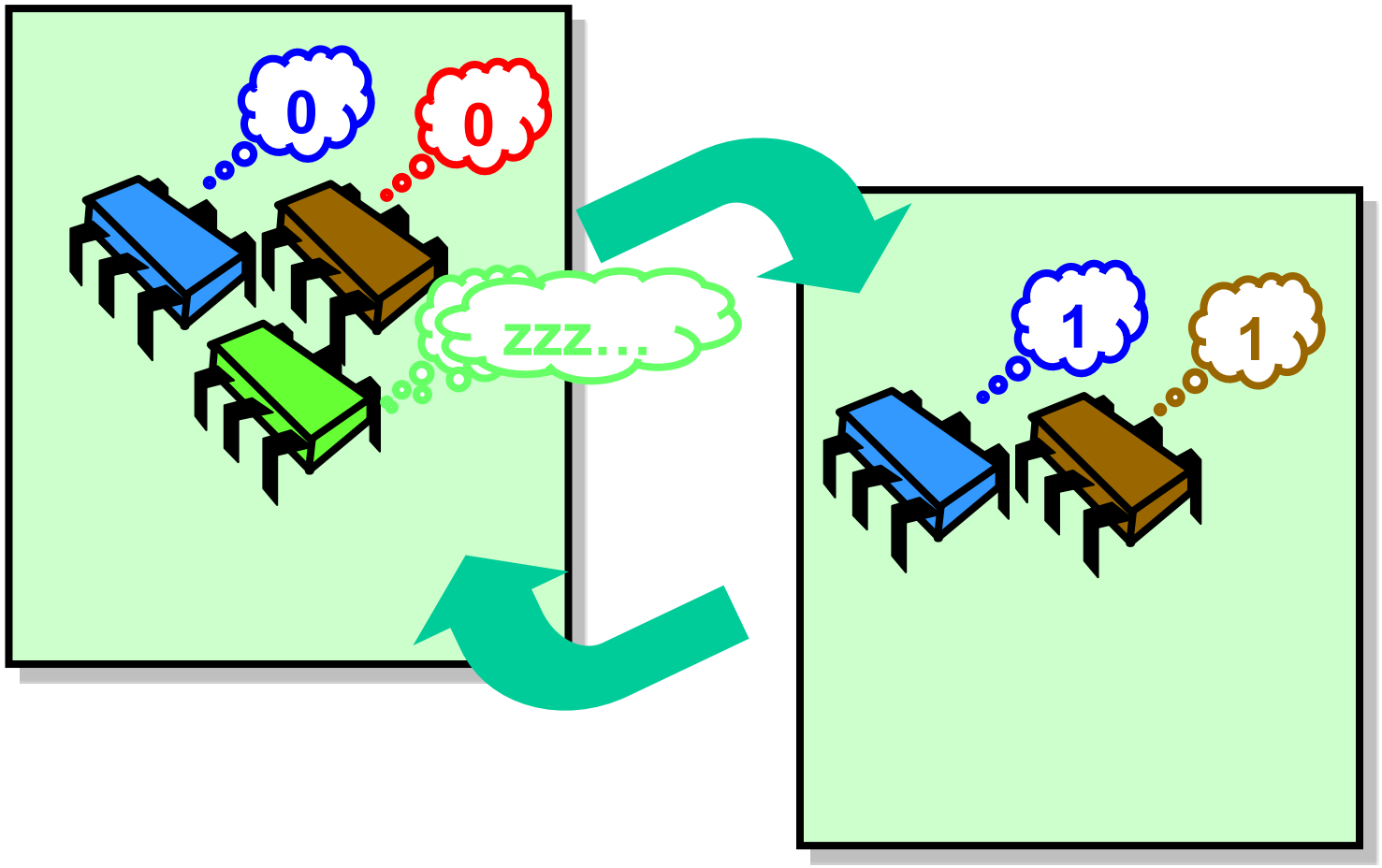
Ideal Parallel Computation



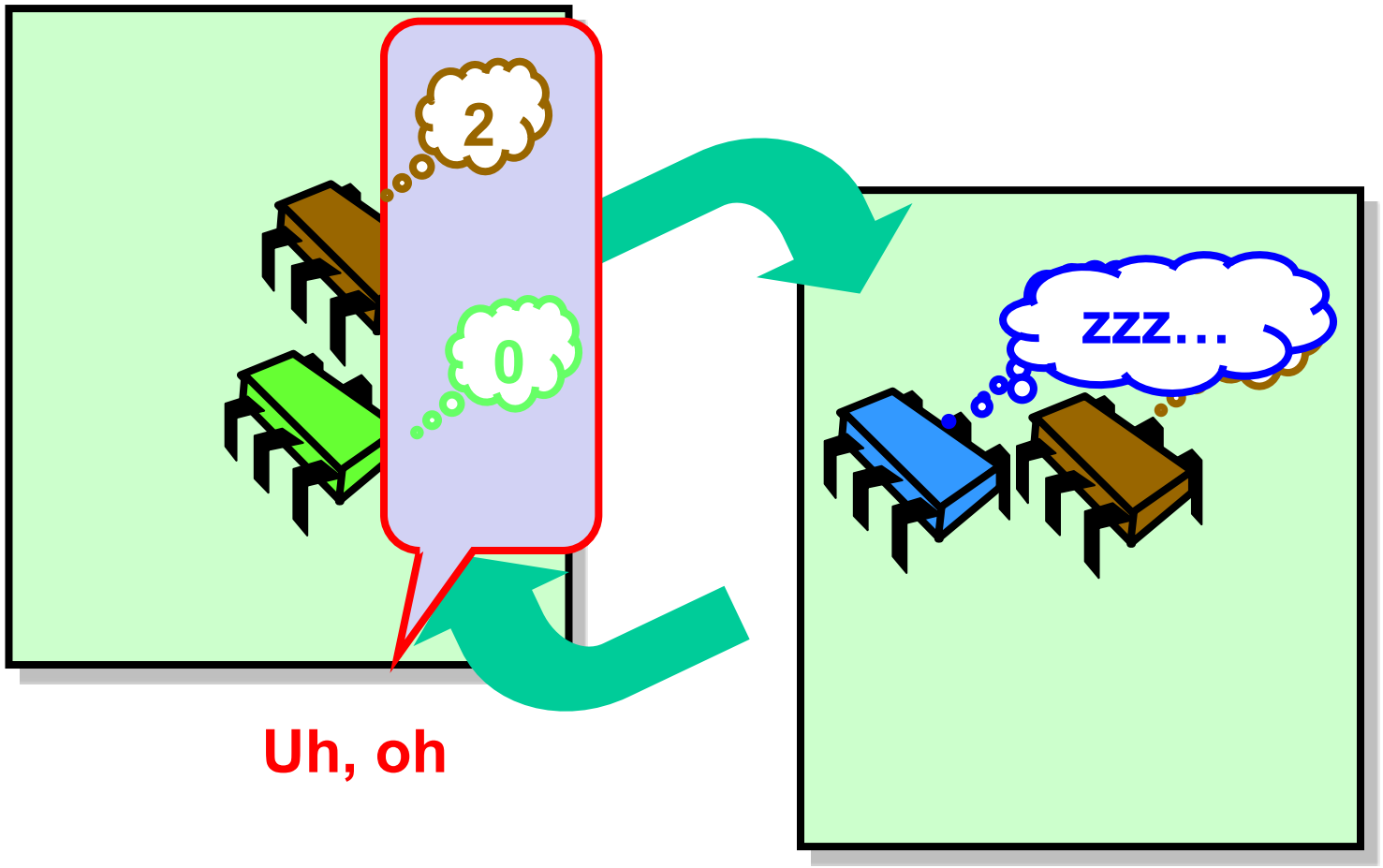
Ideal Parallel Computation



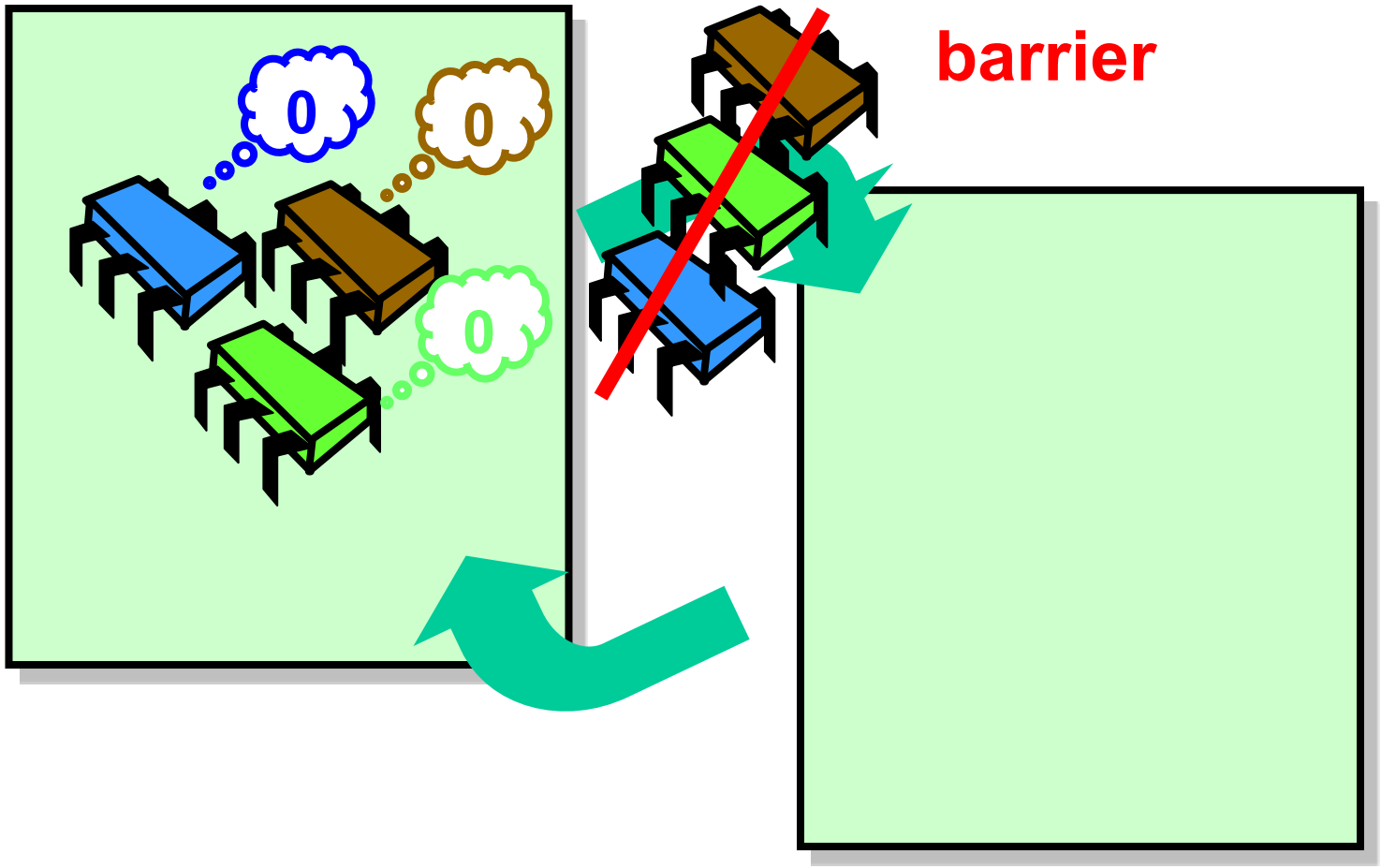
Real-Life Parallel Computation



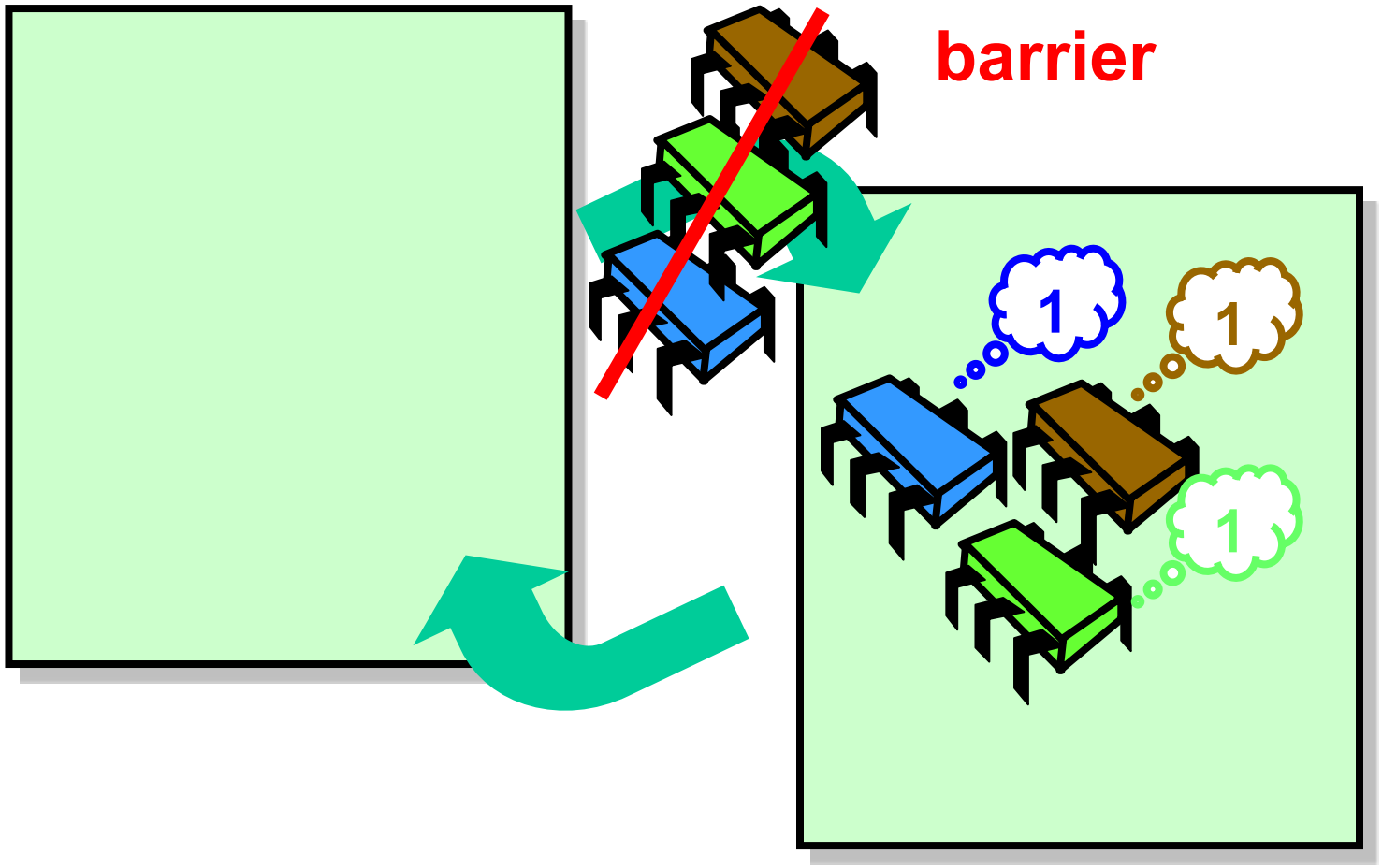
Real-Life Parallel Computation



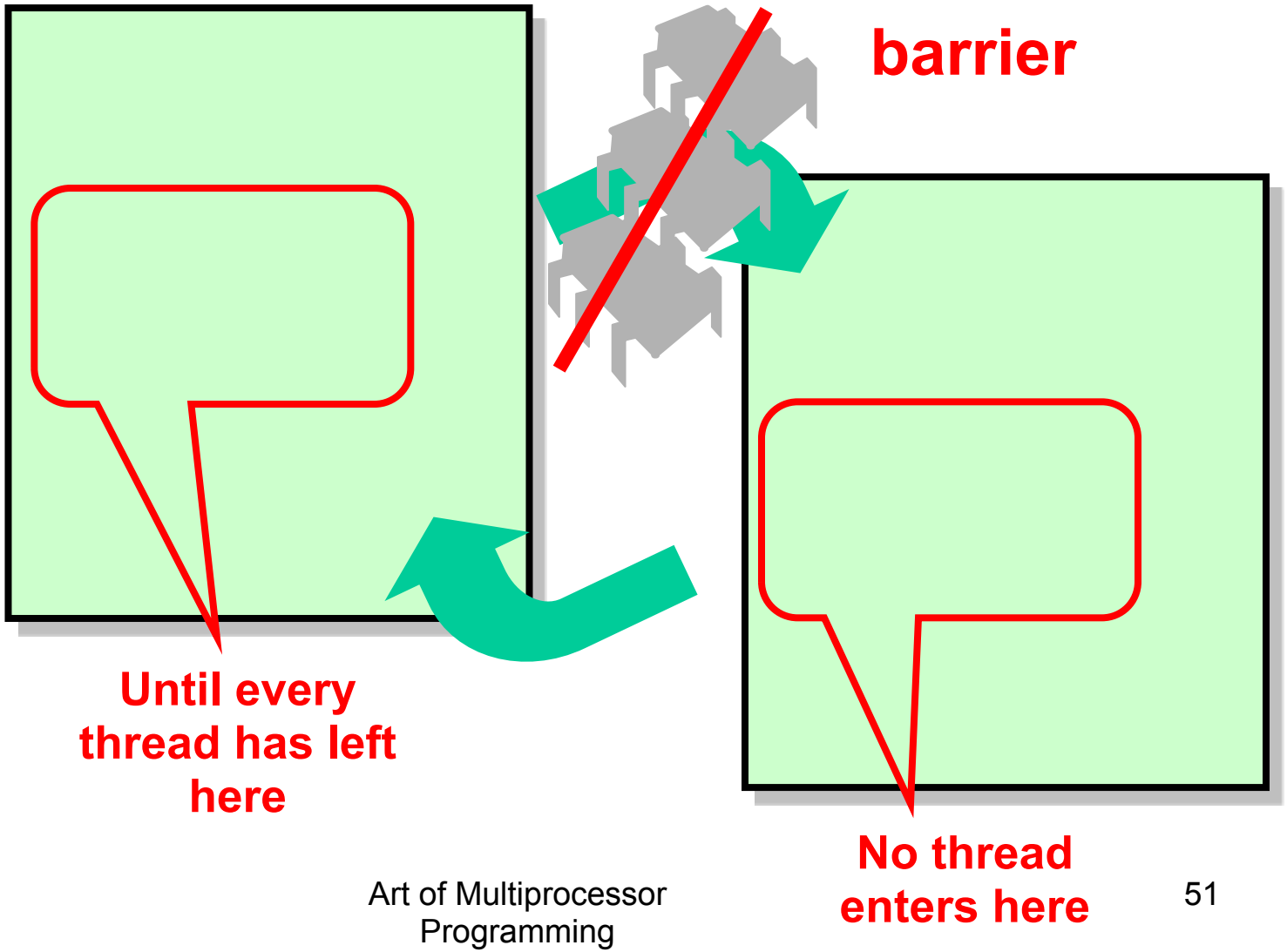
Barrier Synchronization



Barrier Synchronization



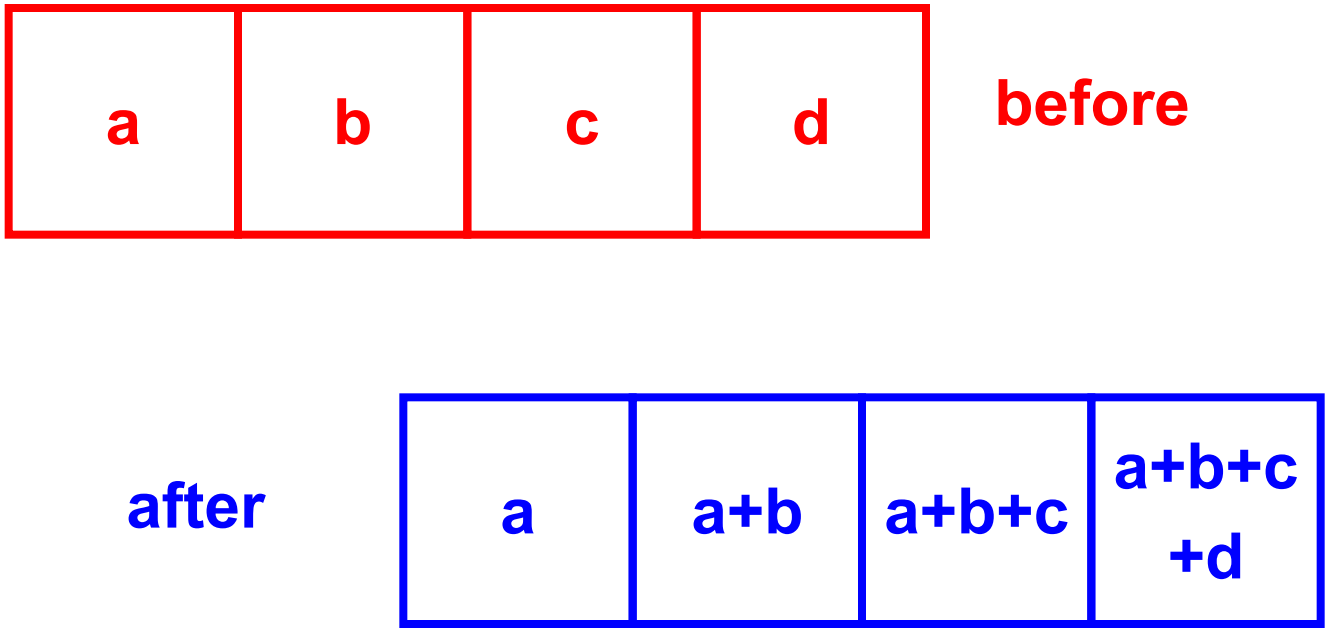
Barrier Synchronization



Why Do We Care?

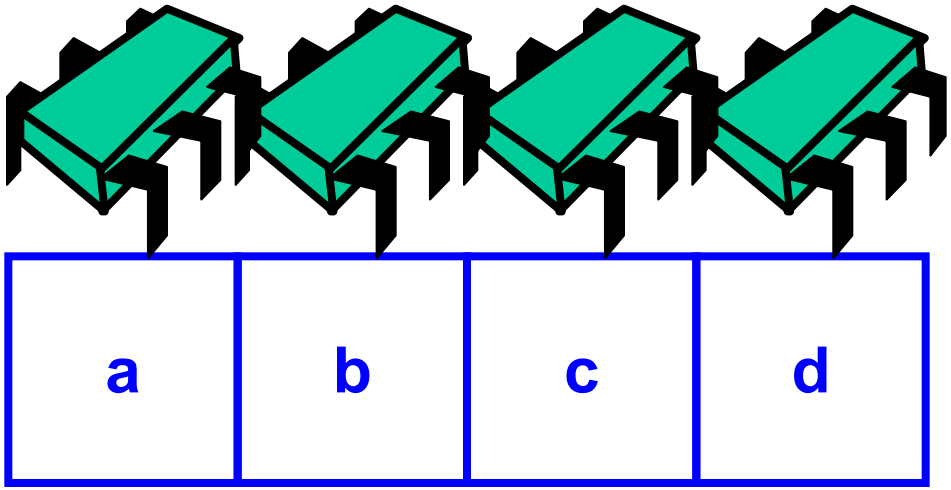
- Mostly of interest to
  - Scientific & numeric computation
- Elsewhere
  - Less common in systems programming
  - Still important topic

Example: Parallel Prefix

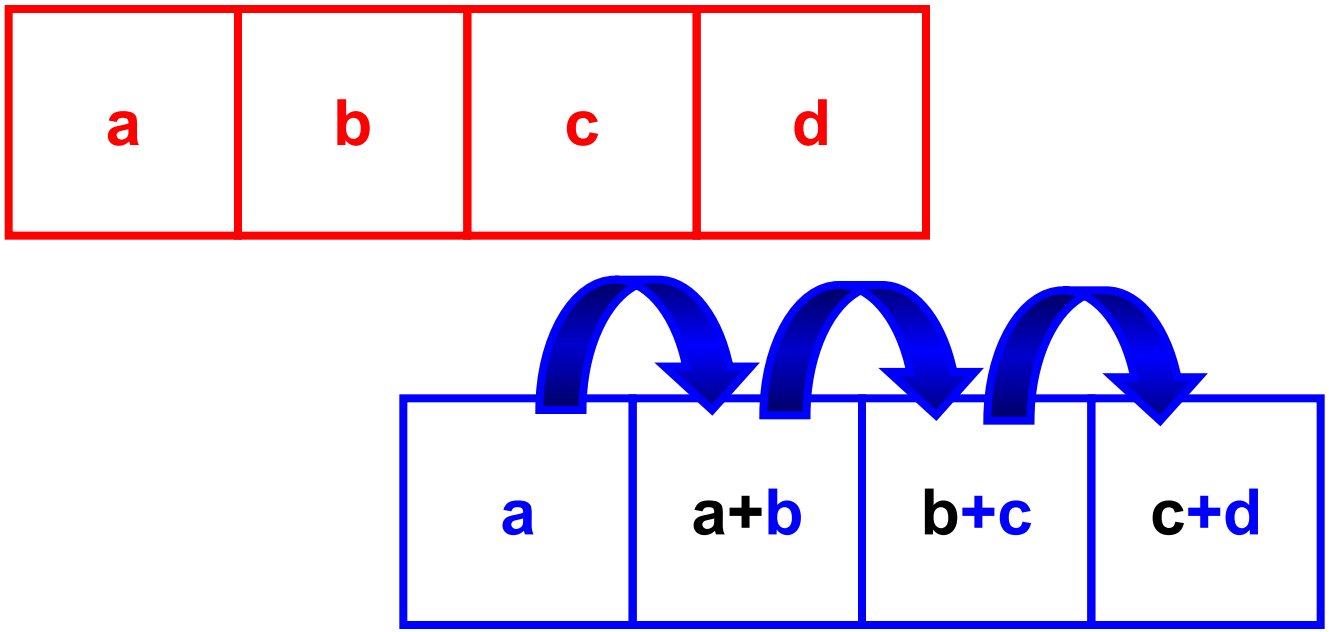


Parallel Prefix

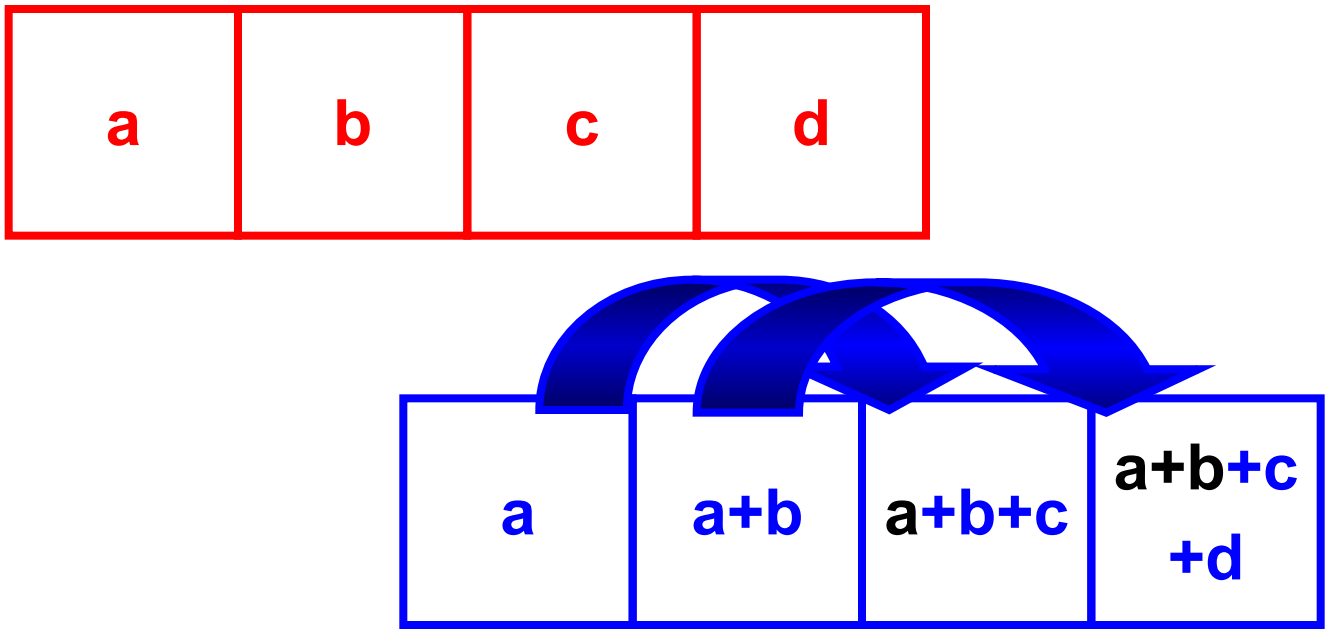
One thread
Per entry



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 {
 int[] a;
 int i;
 Barrier b;
 void Prefix(int[] a,
             Barrier b, int i) {
  a = a;
  b = b;
  i = i;
 }
```

Prefix

```
class Prefix extends Thread {
 int[] a;
 int i;
 Barrier b;
 void Prefix(int[] a,
             Barrier b, int i) {
  a = a;
  b = b;
  i = i;
 }
```

Array of input values

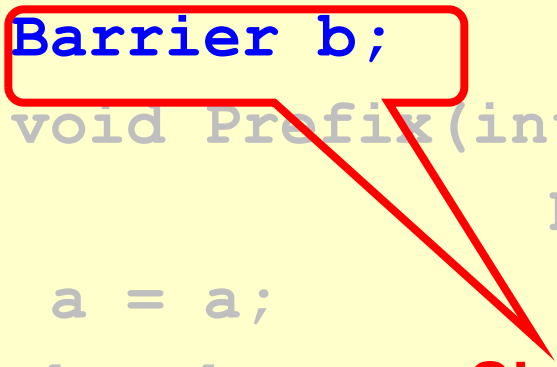
Prefix

```
class Prefix extends Thread {
 int[] a;
 int i;
 Barrier b;
 void Prefix(int[] a,
             Barrier b, int i) {
  a = a;
  b = b;
  i = i;
 }
```

Thread index

Prefix

```
class Prefix extends Thread {
 int[] a;
 int i;
 Barrier b;
 void Prefix(int[] a,
             Barrier b, int i) {
  a = a;
  b = b;
  i = i;
 }
```



Prefix

```
class Prefix extends Thread {
 int[] a;
 int i;
 Barrier b;
 void Prefix(int[] a,
             Barrier b, int i) {
  a = a;
  b = b;
  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;
}}}
```

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

Barriers

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

Number of threads not yet arrived

Barriers

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

Number of threads participating

Barriers

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

Initialization

Barriers

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

Principal method

Barriers

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

If I'm last, reset fields for next time

Barriers

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

Otherwise, wait for everyone else

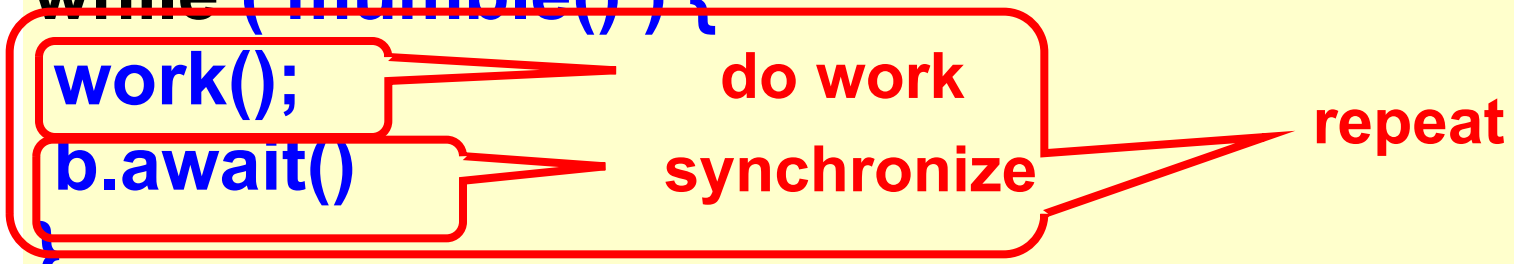
Barriers

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

What's wrong with this protocol?

Reuse

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

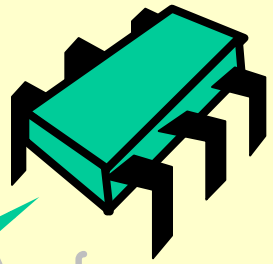


Barriers

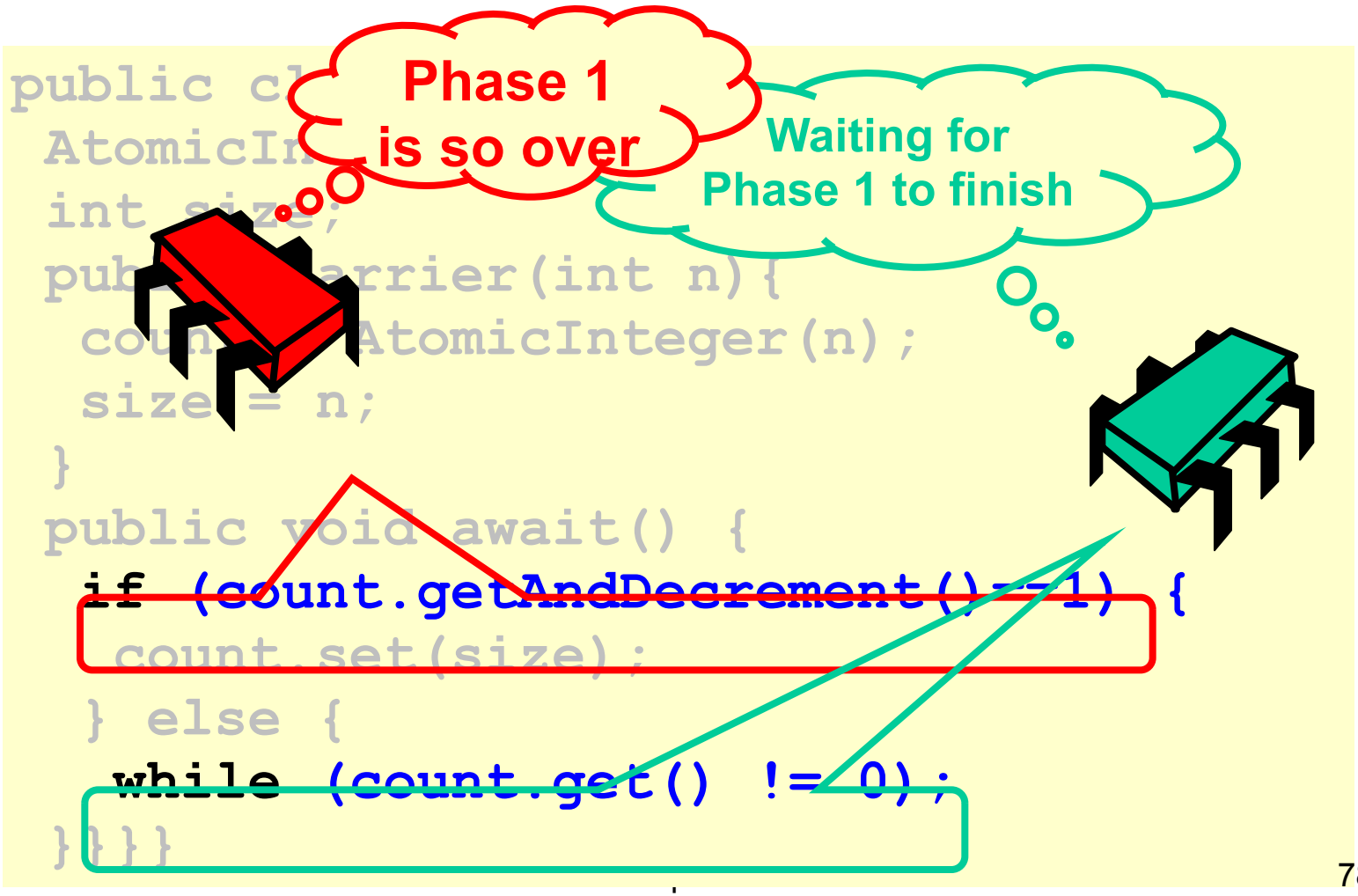
```
public class Barrier {
 AtomicInteger count;
 int size;
 public Barrier(int n){
  count = AtomicInteger(n);
  size = n;
 }
 public void await() {
  if (count.getAndDecrement()==1) {
   count.set(size);
  } else {
   while (count.get() != 0);
 }}}}
```

Barriers

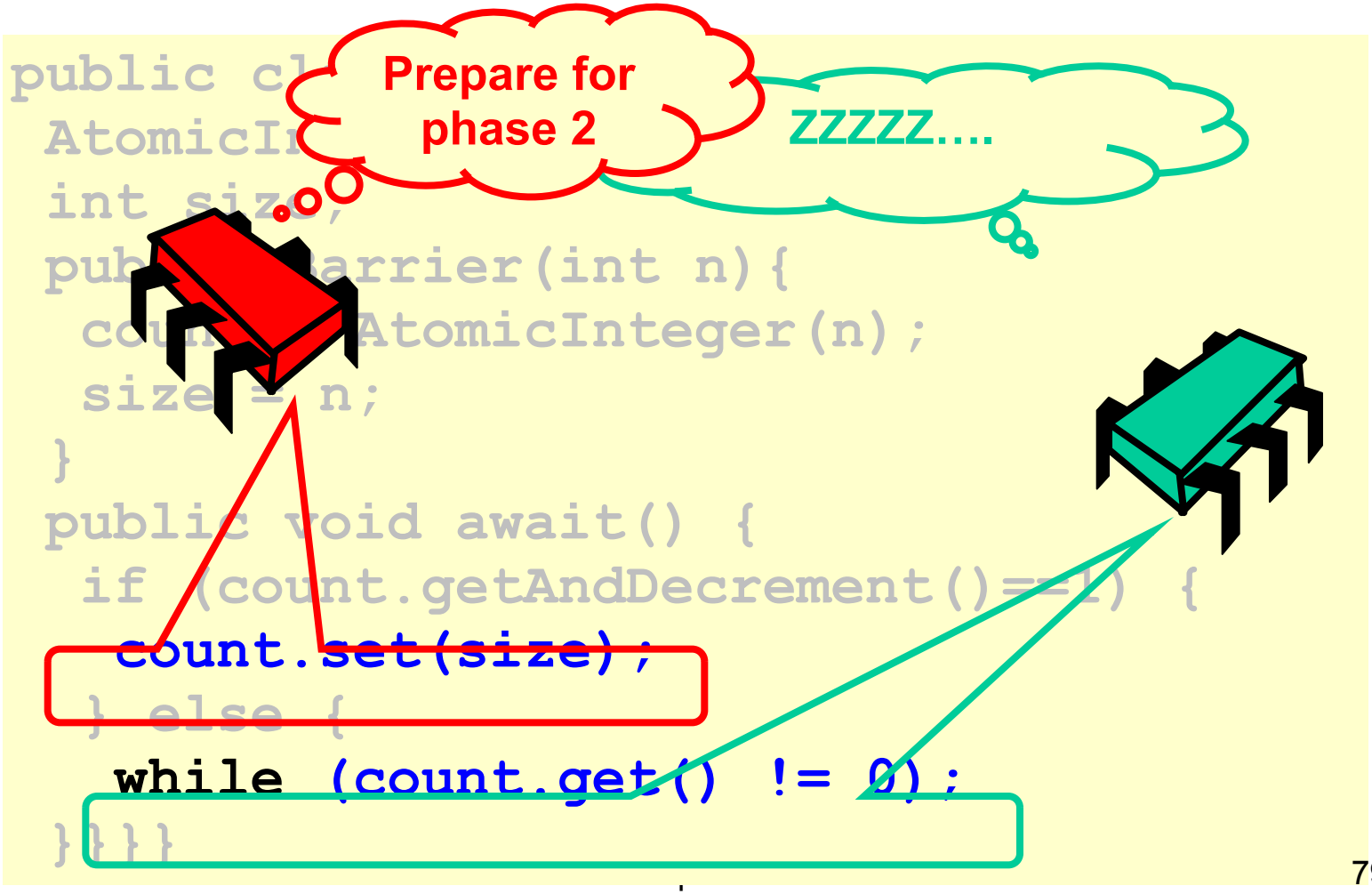
```
public class Barrier {
 AtomicInteger count
 int size;
 public Barrier(int n){
  count = AtomicInteger(n);
  size = n;
 }
 public void await() {
  if (count.getAndDecrement()==1) {
   count.set(size);
  } else {
   while (count.get() != 0);
 }}}}
```



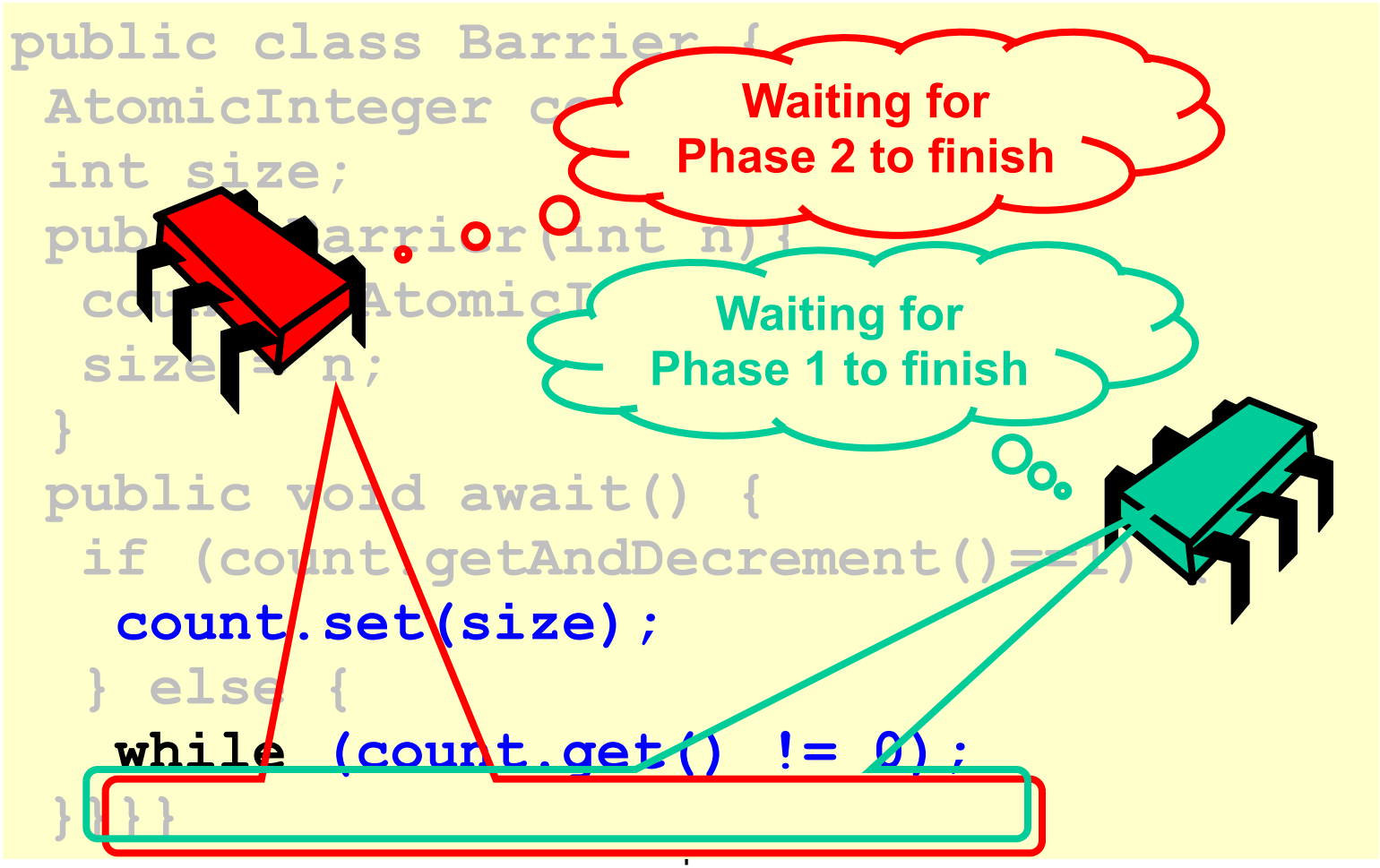
Barriers


Phase 1
is so over
Waiting for
Phase 1 to finish
public cl
AtomicIn
int size;
pub arrier(int n){
coun AtomicInteger(n);
size = n;
}
public void await() {
if (count.getAndDecrement()==1) {
count.set(size);
} else {
while (count.get() != 0);
}}}

Barriers


Prepare for phase 2
ZZZZZ….
public cl
AtomicIn
int size;
pub arrier(int n){
count AtomicInteger(n);
size = n;
}
public void await() {
if (count.getAndDecrement()==1) {
count.set(size);
} else {
while (count.get() != 0);
}}}

Uh-Oh


public class Barrier {
AtomicInteger co
int size;
pub Barrier(int n){
co AtomicI
size = n;
}
public void await() {
if (count.getAndDecrement()==1) {
count.set(size);
} else {
while (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;
 volatile boolean sense = false;
 threadSense = new ThreadLocal<boolean>…

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

Sense-Reversing Barriers

Completed odd or even-numbered phase?

```
public class Barrier {
 AtomicInteger count;
 int size;
 volatile boolean sense = false;
 threadSense = new ThreadLocal<boolean>…

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

Sense-Reversing Barriers

```
public class Barrier {
 AtomicInteger count;
 int size;
 volatile boolean sense = false;
 threadSense = new ThreadLocal<boolean>…

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

Store sense for next phase

Sense-Reversing Barriers

```
public class Barrier {
 AtomicInteger count;
 int size;
 volatile boolean sense = false;
 threadSense = new ThreadLocal<boolean>…

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

Get new sense determined by last phase

Sense-Reversing Barriers

```
public class Barrier {
 AtomicInteger count;
 int size;
 volatile boolean sense = false;
 threadSense = new ThreadLocal<boolean>…

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

If I'm last, reverse sense for next time

Sense-Reversing Barriers

```
public class Barrier {
 AtomicInteger count;
 int size;
 volatile boolean sense = false;
 threadSense = new ThreadLocal<boolean>…

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

Otherwise, wait for sense to flip

Sense-Reversing Barriers

```
public class Barrier {
 AtomicInteger count;
 int size;
 volatile boolean sense = false;
 threadSense = new ThreadLocal<boolean>…

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

Prepare sense for next phase