SFU SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 431 Distributed Systems

Fall 2019

Barriers

https://www.cs.sfu.ca/~keval/teaching/cmpt431/fall19/

Instructor: Keval Vora

Reading

- [AMP] Chapter 17
- [Paper] Algorithms for Scalable Synchronization on Shared Memory Multiprocessors: http://web.mit.edu/6.173/www/currentsemester/readings/R06-scalable-synchronization-1991.pdf

Barriers

- Synchronization method to pause all threads at a point

```
scary_code();
barrier();
really_scary_code();
```

- Fundamental synchronization primitive: must be efficient
- Minimize communication, writes, bus contention, etc.

Barrier Implementation

Issues?

```
1  public class SimpleBarrier implements Barrier {
2    AtomicInteger count;
3    int size;
4    public SimpleBarrier(int n){
5      count = new AtomicInteger(n);
6      size = n;
7    }
8    public void await() {
9      int position = count.getAndDecrement();
10     if (position == 1) {
11       count.set(size);
12     } else {
13       while (count.get() != 0);
14     }
15   }
16 }
```

Deadlock!
Not Reusable

Sense-Reversing Barrier

```
1  public SenseBarrier(int n) {
2    count = new AtomicInteger(n);
3    size = n;
4    sense = false;
5    threadSense = new ThreadLocal<Boolean>() {
6      protected Boolean initialValue() { return !sense; };
7    };
8  }
9  public void await() {
10   boolean mySense = threadSense.get();
11   int position = count.getAndDecrement();
12   if (position == 1) {
13     count.set(size);
14     sense = mySense;
15   } else {
16     while (sense != mySense) {}
17   }
18   threadSense.set(!mySense);
19 }
```

ThreadLocal<Bool>

Can two thread be in different barrier calls?

Sense-Reversing Barrier

```
1  public SenseBarrier(int n) {
2    count = new AtomicInteger(n);
3    size = n;
4    sense = false;
5    threadSense = new ThreadLocal<Boolean>() {
6      protected Boolean initialValue() { return !sense; };
7    };
8  }
9  public void await() {
10   boolean mySense = threadSense.get();
11   int position = count.getAndDecrement();
12   if (position == 1) {
13     count.set(size);
14     sense = mySense;
15   } else {
16     while (sense != mySense) {}
17   }
18   threadSense.set(!mySense);
19 }
```

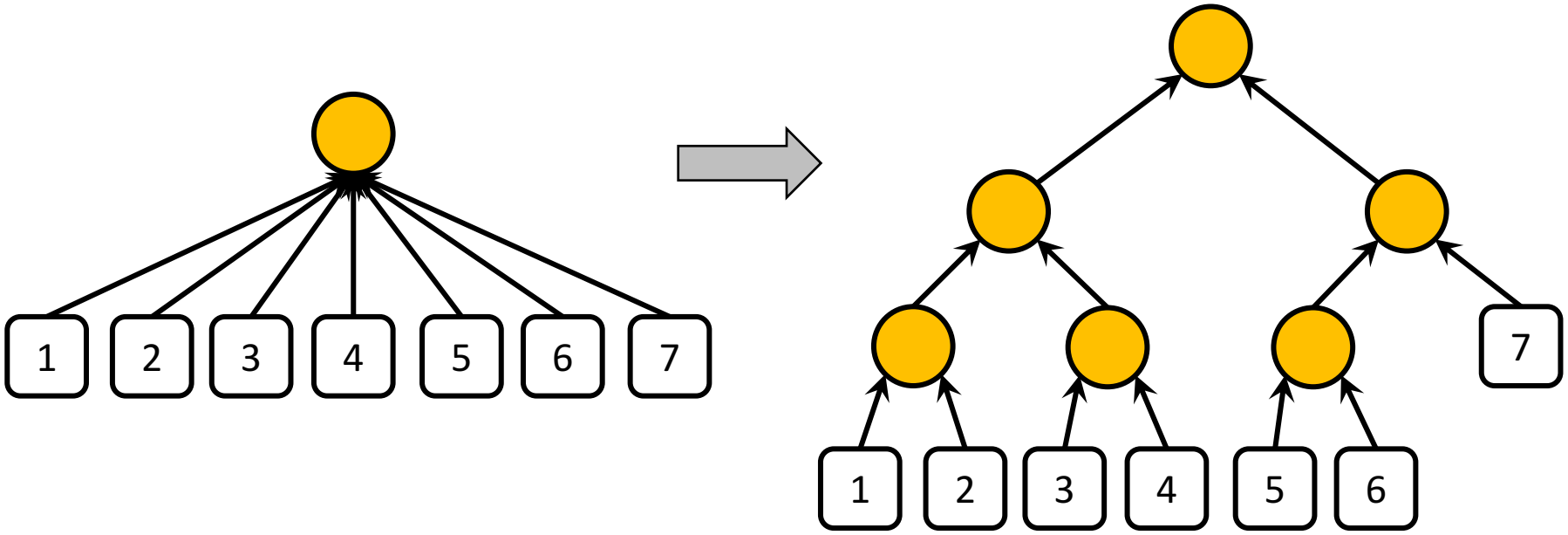
Issues?

Traffic

- Barrier is a fundamental building block – so we should optimize at lower-most level (i.e., reads and writes)
- O(P) traffic on the bus
  - 2P read/write transactions to update count
  - 2 write transactions to write sense and reset count
  - P-1 transactions to read updated sense
- Serialization on a single shared variable
  - Latency is O(P)
  - Can we do better?

Combining Tree Barrier

- Reduces memory contention by spreading memory accesses across multiple barriers



```
1  public class TreeBarrier implements Barrier {
2    int radix;
3    Node[] leaf;
4    int leaves;
5    ThreadLocal<Boolean> threadSense;
6    public TreeBarrier(int n, int r) {
7      radix = r;
8      leaves = 0;
9      leaf = new Node[n / r];
10     int depth = 0;
11     threadSense = new ThreadLocal<Boolean>() {
12       protected Boolean initialValue() { return true; };
13     };
14     // compute tree depth
15     while (n > 1) {
16       depth++;
17       n = n / r;
18     }
19     Node root = new Node();
20     build(root, depth - 1);
21   }
22   // recursive tree constructor
23   void build(Node parent, int depth) {
24     if (depth == 0) {
25       leaf[leaves++] = parent;
26     } else {
27       for (int i = 0; i < radix; i++) {
28         Node child = new Node(parent);
29         build(child, depth - 1);
30       }
31     }
32   }
```

Issues?

```
public void await() {
  int me = ThreadID.get();
  Node myLeaf = leaf[me / radix];
  myLeaf.await();
}
```

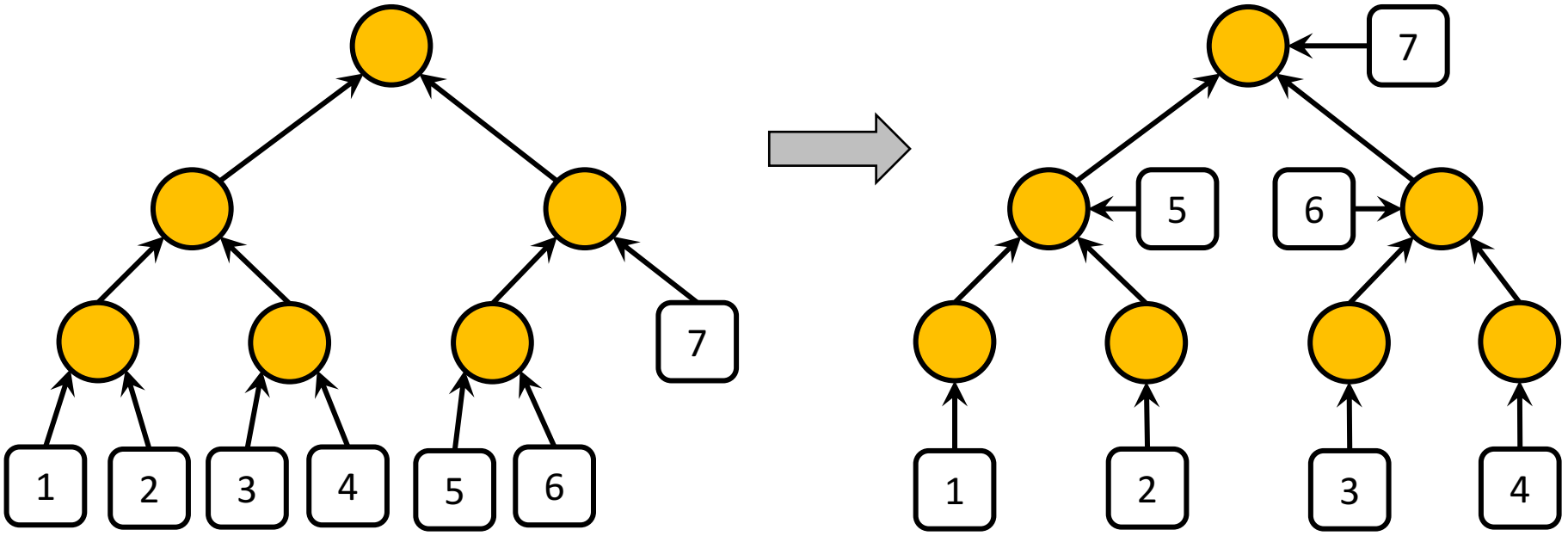
```
1    private class Node {
2      AtomicInteger count;
3      Node parent;
4      volatile boolean sense;
5      public Node() {
6        sense = false;
7        parent = null;
8        count = new AtomicInteger(radix);
9      }
10     public Node(Node myParent) {
11       this();
12       parent = myParent;
13     }
14     public void await() {
15       boolean mySense = threadSense.get();
16       int position = count.getAndDecrement();
17       if (position == 1) {  // I'm last
18         if (parent != null) { // Am I root?
19           parent.await();
20         }
21         count.set(radix);
22         sense = mySense;
23       } else {
24         while (sense != mySense) {};
25       }
26       threadSense.set(!mySense);
27     }
28   }
29 }
```

Combining Tree Barriers

- Removed contention on shared variable
  - Count variables separated for each node

- Unpredictable communication
  - Last thread at every level proceeds towards parent
  - Think NUMA architectures

Static Tree Barriers



```
1  public class StaticTreeBarrier implements Barrier {
2    int radix;
3    boolean sense;
4    Node[] node;
5    ThreadLocal<Boolean> threadSense;
6    int nodes;
7    public StaticTreeBarrier(int size, int myRadix) {
8      radix = myRadix;
9      nodes = 0;
10     node = new Node[size];
11     int depth = 0;
12     while (size > 1) {
13       depth++;
14       size = size / radix;
15     }
16     build(null, depth);
17     sense = false;
18     threadSense = new ThreadLocal<Boolean>() {
19       protected Boolean initialValue() { return !sense; };
20     };
21   }
22   // recursive tree constructor
23   void build(Node parent, int depth) {
24     if (depth == 0) {
25       node[nodes++] = new Node(parent, 0);
26     } else {
27       Node myNode = new Node(parent, radix);
28       node[nodes++] = myNode;
29       for (int i = 0; i < radix; i++) {
30         build(myNode, depth - 1);
31       }
32     }
33   }
34   public void await() {
35     node[ThreadID.get()].await();
36   }
37 }
```

```
1    public Node(Node myParent, int count) {
2      children = count;
3      childCount = new AtomicInteger(count);
4      parent = myParent;
5    }
6    public void await() {
7      boolean mySense = threadSense.get();
8      while (childCount.get() > 0) {};
9      childCount.set(children);
10     if (parent != null) {
11       parent.childDone();
12       while (sense != mySense) {};
13     } else {
14       sense = !sense;
15     }
16     threadSense.set(!mySense);
17   }
18   public void childDone() {
19     childCount.getAndDecrement();
20   }
```

Barriers

- Dissemination Barriers
- Tournament Barriers
- Butterfly Barriers

  …

- [Paper] Algorithms for Scalable Synchronization on Shared Memory Multiprocessors: http://web.mit.edu/6.173/www/currentsemester/readings/R06-scalable-synchronization-1991.pdf

Case Study: Sum

- We have a large number of thread (think millions)
- Each thread has an integer
- The goal is to compute sum of all integers

- Barriers!