Consistency: Strict & Sequential

SWE 622, Spring 2017
Distributed Software Engineering
Review: Real Architectures

N-Tier Web Architectures

Clients

Internet

External Cache

Web Servers

App Servers

Database servers

Internal Cache

Misc Services

Clients
Review: Abstractions & Models

To help design our algorithms and systems, we tend to leverage abstractions and models to make assumptions.

Generally: Stronger assumptions -> worse performance
Weaker assumptions -> more complicated

<table>
<thead>
<tr>
<th>System model</th>
<th>Failure Model</th>
<th>Consistency Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous</td>
<td>Crash-fail</td>
<td>Eventual</td>
</tr>
<tr>
<td>Byzantine</td>
<td>Partitions</td>
<td></td>
</tr>
<tr>
<td>Synchronous</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequential</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generally: Stronger assumptions -> worse performance
Weaker assumptions -> more complicated
HW1 Grades are in Blackboard
Review: HW1

CloudFile cachedFile = findCachedFile(path);
if(cachedFile == null)
{
    cachedFile = getCachedFileAndPutInCache(path);
}
cachedFile.lock.readLock().lock();
//Do stuff
cachedFile.lock.readLock().unlock();
CloudFile cachedFile = findCachedFile(path);
if(cachedFile == null)
{
    cachedFile = getCachedFileAndPutInCache(path);
}
cachedFile.lock.readLock().lock();
//Do stuff
cachedFile.lock.readLock().unlock();
Review: HW1

CloudFile cachedFile = findCachedFile(path);
if(cachedFile == null)
{
    cachedFile = getCachedFileAndPutInCache(path);
}
cachedFile.lock.readLock().lock();
//Do stuff
cachedFile.lock.readLock().unlock();

findCachedFile -> null
Review: HW1

CloudFile cachedFile = findCachedFile(path);
if(cachedFile == null)
{
    cachedFile = getCachedFileAndPutInCache(path);
}
cachedFile.lock.readLock().lock();
//Do stuff
cachedFile.lock.readLock().unlock();
Review: HW1

CloudFile cachedFile = findCachedFile(path);
if(cachedFile == null)
{
    cachedFile = getCachedFileAndPutInCache(path);
}
cachedFile.lock.readLock().lock();
//Do stuff
cachedFile.lock.readLock().unlock();

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>findCachedFile -&gt; null</td>
<td></td>
</tr>
<tr>
<td>getCachedFile -&gt; F1</td>
<td>findCachedFile -&gt; null</td>
</tr>
<tr>
<td>getCachedFile -&gt; F2</td>
<td></td>
</tr>
</tbody>
</table>
Review: HW1

```java
CloudFile cachedFile = findCachedFile(path);
if(cachedFile == null)
{
    cachedFile = getCachedFileAndPutInCache(path);
}
cachedFile.lock.readLock().lock();
//Do stuff
cachedFile.lock.readLock().unlock();
```

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>findCachedFile -&gt; null</td>
<td></td>
</tr>
<tr>
<td>getCachedFile -&gt; F1</td>
<td>findCachedFile -&gt; null</td>
</tr>
<tr>
<td></td>
<td>getCachedFile -&gt; F2</td>
</tr>
<tr>
<td>F1.lock</td>
<td>F2.lock</td>
</tr>
</tbody>
</table>
Review: HW1

CloudFile cachedFile = findCachedFile(path);
if(cachedFile == null)
{
    cachedFile = getCachedFileAndPutInCache(path);
}
cachedFile.lock.readLock().lock();
//Do stuff
cachedFile.lock.readLock().unlock();

Better: correct, but only 1 thread can fetch a new file at a time

synchronized(cache){
    CloudFile cachedFile = findCachedFile(path);
    if(cachedFile == null)
    {
        cachedFile = getCachedFileAndPutInCache(path);
    }
}
cachedFile.lock.readLock().lock();
//Do stuff
cachedFile.lock.readLock().unlock();
CloudFile cachedFile = findCachedFile(path);
if(cachedFile == null)
{
    cachedFile = getCachedFileAndPutInCache(path);
}
cachedFile.lock.readLock().lock();
//Do stuff
cachedFile.lock.readLock().unlock();

Better: correct, but only 1 thread can fetch a new file at a time

synchronized(cache){
    CloudFile cachedFile = findCachedFile(path);
    if(cachedFile == null)
    {
        cachedFile = getCachedFileAndPutInCache(path);
    }
}
cachedFile.lock.readLock().lock();
//Do stuff
cachedFile.lock.readLock().unlock();
Review: HW1

```java
synchronized(cache) {
    CloudFile cachedFile = findCachedFile(path);
}

if (cachedFile == null) {
    cachedFile = downloadFile(path);
} synchronized(cache) {
    if (findCachedFile(path) == null) {
        storeToCache(cachedFile);
    } else {
        cachedFile = findCachedFile(path);
    }
    cachedFile.lock.readLock().lock();
    //Do stuff
    cachedFile.lock.readLock().unlock();
}
```

Best: Release lock, fetch file, then double check that you didn’t accidentally fetch it twice
The Sleeping Barber

• Barber:
  • Cuts 1 person’s hair at a time
  • When finished, dismiss customer. Check waiting room for more customers. If more, then cut next customer’s hair. If no more, take a nap

• Customer:
  • Walks in, sees if barber is napping, if so, wakes barber, else, goes to waiting room
The Sleeping Barber
The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
</table>

[Image showing a table with columns for Barber, Old Customer, and New Customer]
# The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td>Sees barber cutting hair</td>
</tr>
</tbody>
</table>

- Barber: Cutting Hair
- Old Customer: In Chair
- New Customer: Sees barber cutting hair
The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td>Sees barber cutting hair</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Goes to waiting room</td>
</tr>
</tbody>
</table>
# The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td>Sees barber cutting hair</td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td>Goes to waiting room</td>
</tr>
</tbody>
</table>
## The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td>Sees barber cutting hair</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Goes to waiting room</td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td></td>
</tr>
<tr>
<td>Checks waiting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>room</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td>Sees barber cutting hair</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Goes to waiting room</td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td></td>
</tr>
<tr>
<td>Checks waiting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>room</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Escorts to chair, cuts hair</td>
<td></td>
<td>Follows barber to get hair cut</td>
</tr>
</tbody>
</table>
The Sleeping Barber
The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
</table>

The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td></td>
</tr>
</tbody>
</table>

GMU SWE 622 Spring 2017
The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td></td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td>Sees barber cutting hair</td>
</tr>
</tbody>
</table>

J. Bell

GMU SWE 622 Spring 2017
# The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td>Sees barber cutting hair</td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td></td>
</tr>
<tr>
<td>Checks waiting room</td>
<td></td>
<td>Walks slowly to waiting room</td>
</tr>
</tbody>
</table>
# The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td></td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td>Sees barber cutting hair</td>
</tr>
<tr>
<td>Checks waiting room</td>
<td></td>
<td>Walks slowly to waiting room</td>
</tr>
<tr>
<td>Goes to sleep</td>
<td></td>
<td>Sits in waiting room</td>
</tr>
</tbody>
</table>
The Sleeping Barber

Fix: Barber can not check or new customers while customers are entering
<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
</table>

Fix: Barber can not check or new customers while customers are entering
## The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td></td>
</tr>
</tbody>
</table>

Fix: Barber can not check or new customers while customers are entering.
The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td></td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td>Gets lock to check on barber</td>
</tr>
</tbody>
</table>

Fix: Barber can not check or new customers while customers are entering
The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td></td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td>Gets lock to check on barber</td>
</tr>
<tr>
<td>Gets lock to check waiting room (blocked)</td>
<td></td>
<td>Walks slowly to waiting room</td>
</tr>
</tbody>
</table>

Fix: Barber can not check or new customers while customers are entering
# The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td></td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td>Gets lock to check on barber</td>
</tr>
<tr>
<td>Gets lock to check waiting</td>
<td></td>
<td>Walks slowly to waiting room</td>
</tr>
<tr>
<td>room (blocked)</td>
<td></td>
<td>Sits in waiting room</td>
</tr>
</tbody>
</table>

Fix: Barber can not check or new customers while customers are entering
The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td></td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td>Gets lock to check on barber</td>
</tr>
<tr>
<td>Gets lock to check waiting</td>
<td></td>
<td>Walks slowly to waiting room</td>
</tr>
<tr>
<td>room (blocked)</td>
<td></td>
<td>Sits in waiting room</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Releases lock</td>
</tr>
</tbody>
</table>

Fix: Barber can not check or new customers while customers are entering
# The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td></td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td>Gets lock to check on barber</td>
</tr>
<tr>
<td>Gets lock to check waiting room (blocked)</td>
<td>Walks slowly to waiting room</td>
<td>Sits in waiting room</td>
</tr>
<tr>
<td>Acquires lock</td>
<td></td>
<td>Releases lock</td>
</tr>
</tbody>
</table>

Fix: Barber cannot check or new customers while customers are entering
The Sleeping Barber

<table>
<thead>
<tr>
<th>Barber</th>
<th>Old Customer</th>
<th>New Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cutting Hair</td>
<td>In Chair</td>
<td></td>
</tr>
<tr>
<td>Finishes cutting</td>
<td>Leaves</td>
<td>Gets lock to check on barber</td>
</tr>
<tr>
<td>Gets lock to check waiting room (blocked)</td>
<td></td>
<td>Walks slowly to waiting room</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sits in waiting room</td>
</tr>
<tr>
<td>Acquires lock</td>
<td></td>
<td>Releases lock</td>
</tr>
<tr>
<td>Checks waiting room, finds customer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fix: Barber can not check or new customers while customers are entering
Today

• Consistency & Memory Models
• Strict Consistency
• Sequential Consistency
• Distributed Shared Memory
Quiz: What’s the output?

class MyObj {
    int x = 0;
    int y = 0;

    void thread0() {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }
    void thread1() {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}
Quiz: What’s the output?

class MyObj {
    int x = 0;
    int y = 0;

    void thread0() {
        x = 1;
        if(y==0) {
            System.out.println("OK");
        }
    }

    void thread1() {
        y = 1;
        if(x==0) {
            System.out.println("OK");
        }
    }
}
Quiz: What’s the output?

class MyObj {
    int x = 0;
    int y = 0;

    void thread0()
    {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }

    void thread1()
    {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}

"OK"
"OK"
"OK"
Quiz: What’s the output?

class MyObj {
    int x = 0;
    int y = 0;

    void thread0()
    {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }

    void thread1()
    {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}
Quiz: What’s the output?

class MyObj {
    int x = 0;
    int y = 0;

    void thread0() {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }

    void thread1() {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}

"""
"OK"
"OK"
"OK" WTF?
Quiz: What’s the output?

```java
class MyObj {
    int x = 0;
    int y = 0;

    void thread0() {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }

    void thread1() {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}
```

Java Memory Model: Threads are allowed to cache reads and writes
Java Memory Model

CPU 1
- thread0()

CPU 2
- thread1()

Main Memory
Java Memory Model

CPU 1
- thread0()

CPU 2
- thread1()

Main Memory
Java Memory Model

CPU 1

thread0() → 7ns → CPU 1 Cache → 100ns → Main Memory

CPU 2

thread1() → CPU 2 Cache → Main Memory
Java Memory Model

CPU 1
thread0() → CPU 1 Cache → Main Memory
7ns
100ns

CPU 2
thread1() → CPU 2 Cache → Main Memory

Main Memory
Java Memory Model

CPU 1

thread0() -> 7ns -> CPU 1 Cache

CPU 2

thread1() -> CPU 2 Cache

Main Memory

100ns
Quiz: What’s the output?

```java
class MyObj {
    volatile int x = 0;
    volatile int y = 0;

    void thread0() {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }
    void thread1() {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}
```
Quiz: What’s the output?

class MyObj {
    volatile int x = 0;
    volatile int y = 0;

    void thread0()
    {
        x = 1;
        if (y == 0)
            System.out.println("OK");
    }

    void thread1()
    {
        y = 1;
        if (x == 0)
            System.out.println("OK");
    }
}

Volatile keyword: no per-thread caching of variables
Volatile Keyword

CPU 1
- thread0()
- CPU 1 Cache
- 7ns
- 100ns

CPU 2
- thread1()
- CPU 2 Cache
- X

Main Memory
Consistency

• This is a consistency model!

• Constraints on the system state that are observable by applications

• “When I write y=1, any future reads must say y=1”

• … except in Java, if it’s a non-volatile variable

• Clearly, this often comes at a cost (see simple example with `volatile`...)
Strict Consistency

• Each operation is stamped with a global (wall-clock, aka absolute) time

• Rules:
  • Each read sees the latest write
  • All operations on one CPU have time-stamps in execution order
Strict Consistency

class MyObj {
    volatile int x = 0;
    volatile int y = 0;

    void thread0() {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }

    void thread1() {
        y = 1;
        if(x==0)
            if(y==0)
                System.out.println("OK");
    }
}
Strict Consistency

class MyObj {
    volatile int x = 0;
    volatile int y = 0;

    void thread0() {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }

    void thread1() {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}

<table>
<thead>
<tr>
<th></th>
<th>t=0</th>
<th>t=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU0</td>
<td>W(X) 1</td>
<td>R(Y) 1</td>
</tr>
<tr>
<td>CPU1</td>
<td>W(Y) 1</td>
<td>R(X) 1</td>
</tr>
<tr>
<td>CPU0</td>
<td>W(X) 1</td>
<td>R(Y) 0</td>
</tr>
<tr>
<td>CPU1</td>
<td>W(Y) 1</td>
<td>R(X) 1</td>
</tr>
</tbody>
</table>
Strict Consistency

```java
class MyObj {
    volatile int x = 0;
    volatile int y = 0;

    void thread0() {
        x = 1;
        if(y == 0)
            System.out.println("OK");
    }

    void thread1() {
        y = 1;
        if(x == 0)
            System.out.println("OK");
    }
}
```

<table>
<thead>
<tr>
<th></th>
<th>t=0</th>
<th>t=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU0</td>
<td>W(X) 1</td>
<td>R(Y) 1</td>
</tr>
<tr>
<td>CPU1</td>
<td>W(Y) 1</td>
<td>R(X) 1</td>
</tr>
<tr>
<td>CPU0</td>
<td>W(X) 1</td>
<td>R(Y) 0</td>
</tr>
<tr>
<td>CPU1</td>
<td>W(Y) 1</td>
<td>R(X) 1</td>
</tr>
<tr>
<td>CPU0</td>
<td>W(X) 1</td>
<td>R(Y) 0</td>
</tr>
<tr>
<td>CPU1</td>
<td>W(Y) 1</td>
<td>R(X) 0</td>
</tr>
</tbody>
</table>
Strict Consistency

```java
class MyObj {
    volatile int x = 0;
    volatile int y = 0;

    void thread0() {
        x = 1;
        if (y == 0)
            System.out.println("OK");
    }

    void thread1() {
        y = 1;
        if (x == 0)
            System.out.println("OK");
    }
}
```
Sequential Consistency

• Strict consistency is often not practical
  • Requires globally synchronizing clocks

• Sequential consistency gets close, in an easier way:
  • There is some total order of operations so that:
    • Each CPUs operations appear in order
    • All CPUs see results according to that order (read most recent writes)
## Sequential Consistency

### Strict Consistency

<table>
<thead>
<tr>
<th></th>
<th>t=0</th>
<th>t=1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU0</strong></td>
<td>W(X) 1</td>
<td>R(Y) 1</td>
</tr>
<tr>
<td><strong>CPU1</strong></td>
<td>W(Y) 1</td>
<td>R(X) 1</td>
</tr>
</tbody>
</table>

### Sequential Consistency

<table>
<thead>
<tr>
<th></th>
<th>t=0</th>
<th>t=1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU0</strong></td>
<td>W(X) 1</td>
<td>R(Y) 1</td>
</tr>
<tr>
<td><strong>CPU1</strong></td>
<td>W(Y) 1</td>
<td>R(X) 1</td>
</tr>
</tbody>
</table>
Distributed Shared Memory

• How do multiple nodes communicate in a distributed system?
  • Message passing
    • E.g. RMI
  • Shared memory
    • Make it look like there are just a bunch of threads on one machine
Distributed Shared Memory

stack
thread0()

stack
thread1()

heap

DSM
x y
Naïve DSM

• Assume each machine has a complete copy of memory
• Reads from local memory
• Writes broadcast update to other machines, then immediately continue
Naïve DSM

- Assume each machine has a complete copy of memory
- Reads from local memory
- Writes broadcast update to other machines, then immediately continue

```java
class Machine1 {
    DSMInt x = 0;
    DSMInt y = 0;
    static void main(String[] args)
    {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }
}
class Machine2 {
    DSMInt x = 0;
    DSMInt y = 0;
    static void main(String[] args)
    {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}
```
Naïve DSM

- Assume each machine has a complete copy of memory
- Reads from local memory
- Writes broadcast update to other machines, then immediately continue

```java
class Machine1 {
    DSMInt x = 0;
    DSMInt y = 0;

    static void main(String[] args) {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }
}

class Machine2 {
    DSMInt x = 0;
    DSMInt y = 0;

    static void main(String[] args) {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}
```
Naïve DSM

- Assume each machine has a complete copy of memory
- Reads from local memory
- Writes broadcast update to other machines, then immediately continue

```java
class Machine1 {
    DSMInt x = 1;
    DSMInt y = 0;

    static void main(String[] args) {
        x = 1;
        if (y == 0)
            System.out.println("OK");
    }
}

class Machine2 {
    DSMInt x = 1;
    DSMInt y = 0;

    static void main(String[] args) {
        y = 1;
        if (x == 0)
            System.out.println("OK");
    }
}
```
Naïve DSM

• Assume each machine has a complete copy of memory
• Reads from local memory
• Writes broadcast update to other machines, then immediately continue

```java
class Machine1 {
    DSMInt x = 1;
    DSMInt y = 1;

    static void main(String[] args) {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }
}

class Machine2 {
    DSMInt x = 1;
    DSMInt y = 1;

    static void main(String[] args) {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}
```
Naïve DSM

- Assume each machine has a complete copy of memory
- Reads from local memory
- Writes broadcast update to other machines, then immediately continue

```java
class Machine1 {
    DSMInt x = 1;
    DSMInt y = 1;

    static void main(String[] args) {
        x = 1;
        if (y == 0)
            System.out.println("OK");
    }
}

class Machine2 {
    DSMInt x = 1;
    DSMInt y = 1;

    static void main(String[] args) {
        y = 1;
        if (x == 0)
            System.out.println("OK");
    }
}
```
Naïve DSM

- Assume each machine has a complete copy of memory
- Reads from local memory
- Writes broadcast update to other machines, then immediately continue

```java
class Machine1 {
    DSMInt x = 1;
    DSMInt y = 1;
    static void main(String[] args) {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }
}
class Machine2 {
    DSMInt x = 1;
    DSMInt y = 1;
    static void main(String[] args) {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}
```
Naïve DSM

• Assume each machine has a complete copy of memory
• Reads from local memory
• Writes broadcast update to other machines, then immediately continue

```java
class Machine1 {
    DSMInt x = 0;
    DSMInt y = 0;

    static void main(String[] args) {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }
}

class Machine2 {
    DSMInt x = 0;
    DSMInt y = 0;

    static void main(String[] args) {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}
```
Naïve DSM

- Assume each machine has a complete copy of memory
- Reads from local memory
- Writes broadcast update to other machines, then immediately continue

```java
class Machine1 {
    DSMInt x = 1;
    DSMInt y = 1;

    static void main(String[] args) {
        x = 1;
        if(y==0)
            System.out.println("OK");
    }
}

class Machine2 {
    DSMInt x = 1;
    DSMInt y = 1;

    static void main(String[] args) {
        y = 1;
        if(x==0)
            System.out.println("OK");
    }
}
```
Naïve DSM

- Assume each machine has a complete copy of memory
- Reads from local memory
- Writes broadcast update to other machines, then immediately continue

```java
class Machine1 {
    DSMInt x = 0;
    DSMInt y = 0;
    static void main(String[] args) {
        x = 1;
        if(y==0) System.out.println("OK");
    }
}

class Machine2 {
    DSMInt x = 0;
    DSMInt y = 0;
    static void main(String[] args) {
        y = 1;
        if(x==0) System.out.println("OK");
    }
}
```

Is this correct?
Naïve DSM

• Gets even more funny when we add a third host
  • Many more interleaving possible
• Definitely not sequentially consistent
• Who is at fault?
  • The DSM system?
  • The app?
  • The developers of the app, if they thought it would be sequentially consistent.
Sequentially Consistent DSM

• How do we get this system to behave similar to Java’s volatile keyword?

• We want to ensure:
  • Each machine’s own operations appear in order
  • All machines see results according to some total order (each read sees the most recent writes)

• We can say that some observed runtime ordering of operations can be “explained” by a sequential ordering of operations that follow the above rules
Sequentially Consistent DSM

- Each node must see the most recent writes before it reads that same data
- Performance is not great:
  - Might make writes expensive: need to wait to broadcast and ensure other nodes heard your new value
  - Might make reads expensive: need to wait to make sure that there are no pending writes that you haven’t heard about yet
Sequentially Consistent DSM

- Each processor issues requests in the order specified by the program
  - Can’t issue the next request until previous is finished
- Requests to an individual memory location are served from a single FIFO queue
  - Writes occur in single order
  - Once a read observes the effect of a write, it’s ordered behind that write
Sequentially Consistent DSM

CPU 1
- thread0(): 7ns
- CPU 1 Cache
- Main Memory: 100ns

CPU 2
- thread1():
- CPU 2 Cache
- Main Memory
Sequentially Consistent DSM

CPU 1
- thread0()
- CPU 1 Cache

CPU 2
- thread1()
- CPU 2 Cache

Main Memory
- 100ns

FIFO queue
- 1s?

7ns

1s?
Ivy DSM

- Integrated shared Virtual memory at Yale
- Provides shared memory across a group of workstations
- Might be easier to program with shared memory than with message passing
  - Makes things look a lot more like one huge computer with hundreds of CPUs instead of hundreds of computers with one CPU
Ivy Architecture
Ivy Architecture

Each node keeps a cached copy of each piece of data it reads.
Ivy Architecture

Each node keeps a cached copy of each piece of data it reads.

If some data doesn’t exist locally, request it from remote node.
Ivy provides sequential consistency

- Support multiple readers, single writer semantics
- Write invalidate update protocol
- If I write some data, I must tell everyone who has cached it to get rid of their cache
Ivy Architecture

Each node keeps a cached copy of each piece of data it reads.

If some data doesn’t exist locally, request it from remote node.

cached data
x=0

cached data
x=0

cached data
Ivy Architecture

Each node keeps a cached copy of each piece of data it reads.

Write X=1

If some data doesn’t exist locally, request it from remote node.

cached data
x=0

cached data
x=0

cached data
x=0
Ivy Architecture

Each node keeps a cached copy of each piece of data it reads.

Write X=1

If some data doesn’t exist locally, request it from remote node.

cached data

x=1

cached data

x=0

cached data

x=0
Ivy Architecture

Each node keeps a cached copy of each piece of data it reads.

Write $X=1$

If some data doesn’t exist locally, request it from remote node.

Invalidate $x$
Ivy Architecture

Each node keeps a cached copy of each piece of data it reads.

Write $X=1$

If some data doesn’t exist locally, request it from remote node.
Ivy Architecture

Each node keeps a cached copy of each piece of data it reads.

If some data doesn’t exist locally, request it from remote node.
Ivy Architecture

Each node keeps a cached copy of each piece of data it reads.

If some data doesn’t exist locally, request it from remote node.

Read X

cached data

x=1

cached data

cached data

Read X
Each node keeps a cached copy of each piece of data it reads.

If some data doesn’t exist locally, request it from remote node.

Read X

cached data

x=1

Read X

cached data

cached data
Ivy Architecture

Each node keeps a cached copy of each piece of data it reads.

If some data doesn’t exist locally, request it from remote node.

Read X

cached data
x=1

cached data
x=1

cached data
x=1

Read X
Ivy Implementation

- Ownership of data moves to be whoever last wrote it
- There are still some tricky bits:
  - How do we know who owns some data?
  - How do we ensure only one owner per data?
  - How do we ensure all cached data are invalidated on writes?
- Solution: Central manager node
Ivy invariants

• Every piece of data has exactly one current owner
• Current owner is guaranteed to have a copy of that data
• If the owner has write permission, no other copies can exist
• If owner has read permission, it's guaranteed to be identical to other copies
• Manager node knows about all of the copies
• Sounds a lot like CFS + its lock server? :)
Lab: More Concurrency, lock, wait
Dining Philosophers

• N philosophers seated around a circular table
  • One chopstick between each philosopher (N chopsticks)
  • A philosopher picks up both chopsticks next to him to eat
  • Philosophers may not pick up both chopsticks at the same time
• How do they all eat without deadlocking or starving?
Dining Philosophers

• Give each chopstick a lock
• Is this enough?
• Could deadlock!
• Solution: always grab right chopstick first?
  • Nope
• Solution: Index chopsticks, always grab biggest # first?
• Solution: Only get them if they are both free?
Your tasks

• I’ve provided an implementation of Dining Philosophers with a little simulator. It deadlocks. Fix it.
• Implement a small simulator for sleeping barber