ECE 570/670

David Irwin

Lecture 3

Administrative Details

- Website: https://courses.umass.edu/eceng570-deirwin/spring23
  - Username: **irwin-ece**, Password: **myclass**
  - *Calendar will change*

- Think about Assignment partners
  - Assignment 0 will be posted on Thursday
    - Fill out Google Form your NetID and NetID of your partner
      - https://forms.gle/ChgyjQehA5hQku5W6
      - I will send back to you a username ("groupXX") and a password to login to the VM you **must** use for the assignments
    - **This must be done by next Sunday!**
  - Ensure you have and can check **@umass.edu** email

- UNIX paper posted for Thursday

Administrative Details

- Piazza setup
  - Will update tomorrow after grad add/drop
- Reviewing system
  - Will add people tomorrow after grad add/drop
  - Link to reviewing system on website
- Moodle setup
  - Only used for posting grades
- VM image posted (ECE670-Spring23.ova)
  - Website resources page
- My Office hours
  - Mondays 2:30pm – 3:30pm
  - Thursdays 9am – 10am

Last Time

- Finished intro
- Started talking about processes and threads

Today's Outline

- Discuss paper
- Continue talking about threads
- Start talking about how to deal with synchronization issues with multiple threads
  - Thread cooperation

"Hints for Computer System Design"

- Technical fellow at Microsoft
- Adjunct Professor at MIT
- Computer architecture, LANs, printers, operating systems, RPCs, programming languages, security, etc.
- Won Turing award in 1992

- Questions? Comments?
- Discussion.

Butler Lampson

“Hints for Computer System Design”

- Divides Hints into 3 categories

- 1. Functionality
- 2. Speed
- 3. Fault-tolerance

Functionality

- Keep it Simple
  - "Do one thing and do it well"
  - "Get it right"
  - "Don't hide power"
  - "Leave it to the client"
- Continuity
  - "Keep interfaces stable"
  - "Keep a place to stand"
- Making things work
  - "Plan to throw one away"
  - "Keep Secrets"
  - "Divide and Conquer"

Speed

- Suggestions
  - Think about “splitting resources” rather than sharing
  - “Cache answers” or use hints
  - “When in doubt use brute force”
  - “Compute in the background”
  - “Use batch processing if possible”
  - “Safety first”
  - “Shed load”

Fault-tolerance

- Suggestions
  - Use end-to-end error recovery
  - Log updates before making them
  - Make actions either atomic or restartable
    - What's another word for restartable?

Consider a web server

- One processor
- Multiple disks
- Tasks
  - 1. Receives multiple, simultaneous requests
    - *Return from accept() with socket ids of new connections*
  - 2. Reads web pages from disk
    - *Receive data by calling recv() on socket id and parse request to find out request HTML web page*
  - 3. Returns on-disk files to requester
    - *Call read() to read the requested HTML file from the file system, and call send() to send the data to the requester*

Three Different Designs

- ***Single-threaded*** – one request at a time, lots of waiting
- ***Event-driven*** - one thread, but interleave requests
- ***Multi-threaded*** – many threads, one request per thread, lots of waiting but scheduler switches between threads

Benefits of Threads

- Thread manager takes care of sharing CPU among threads
  - Thread can issue blocking I/Os, while other threads make progress
  - Private state for each thread
- Applications get a simpler programming model
  - Illusion of a dedicated CPU per thread
- Can isolate requests and schedule them differently

- Any drawbacks?
  - Overhead of context switching (matters in high concurrency servers)
  - Possible state overhead (replicated state between threads)

Threads are useful

- They cannot provide total independence
  - But they are still a useful abstraction!
  - Threads make concurrent programming easier
- Thread system manages sharing the CPU
  - (unlike in the event-driven case)
- Apps can *encapsulate* task state within a thread
  - (e.g. web request state)

Where are threads used?

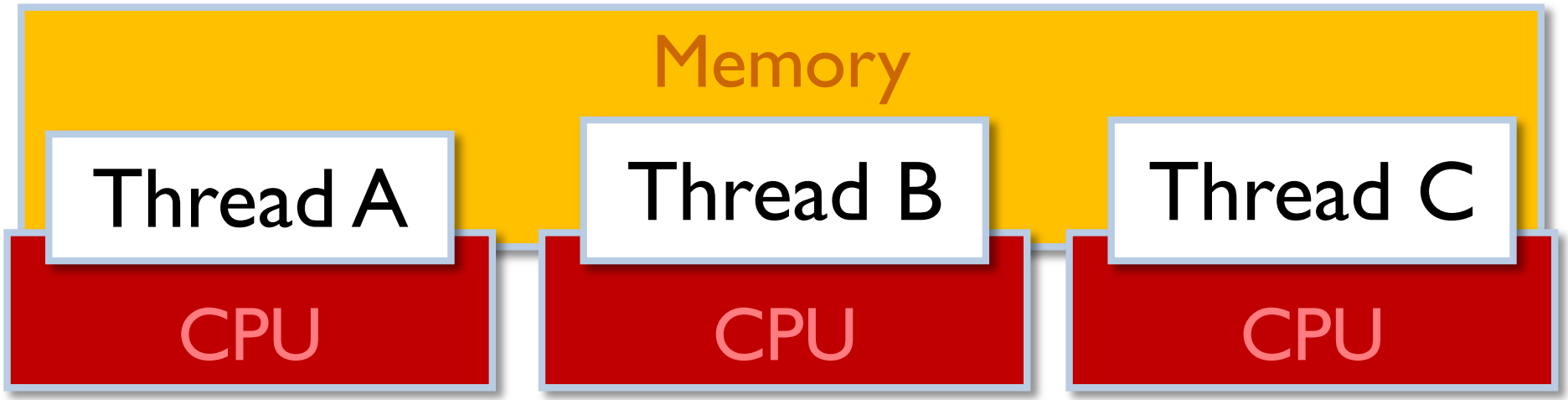
- When a resource is slow, don't want to wait on it
- Windowing system
  - One thread per window, waiting for window input
  - What is slow?
    - Human input, mouse, keyboard
- Network file/web/DB server
  - One thread per incoming request
  - What is slow?
    - Network, disk, remote user (e.g. ATM bank customer)

Where are threads used?

- When a resource is slow, don't want to wait on it
- Operating system kernel (central component of the OS)
  - One thread waits for keyboard input
  - One thread waits for mouse input
  - One thread writes to the display
  - One thread writes to the printer
  - One thread receives data from the network card
  - One thread per disk …
  - *Just about everything except the CPU is slow*

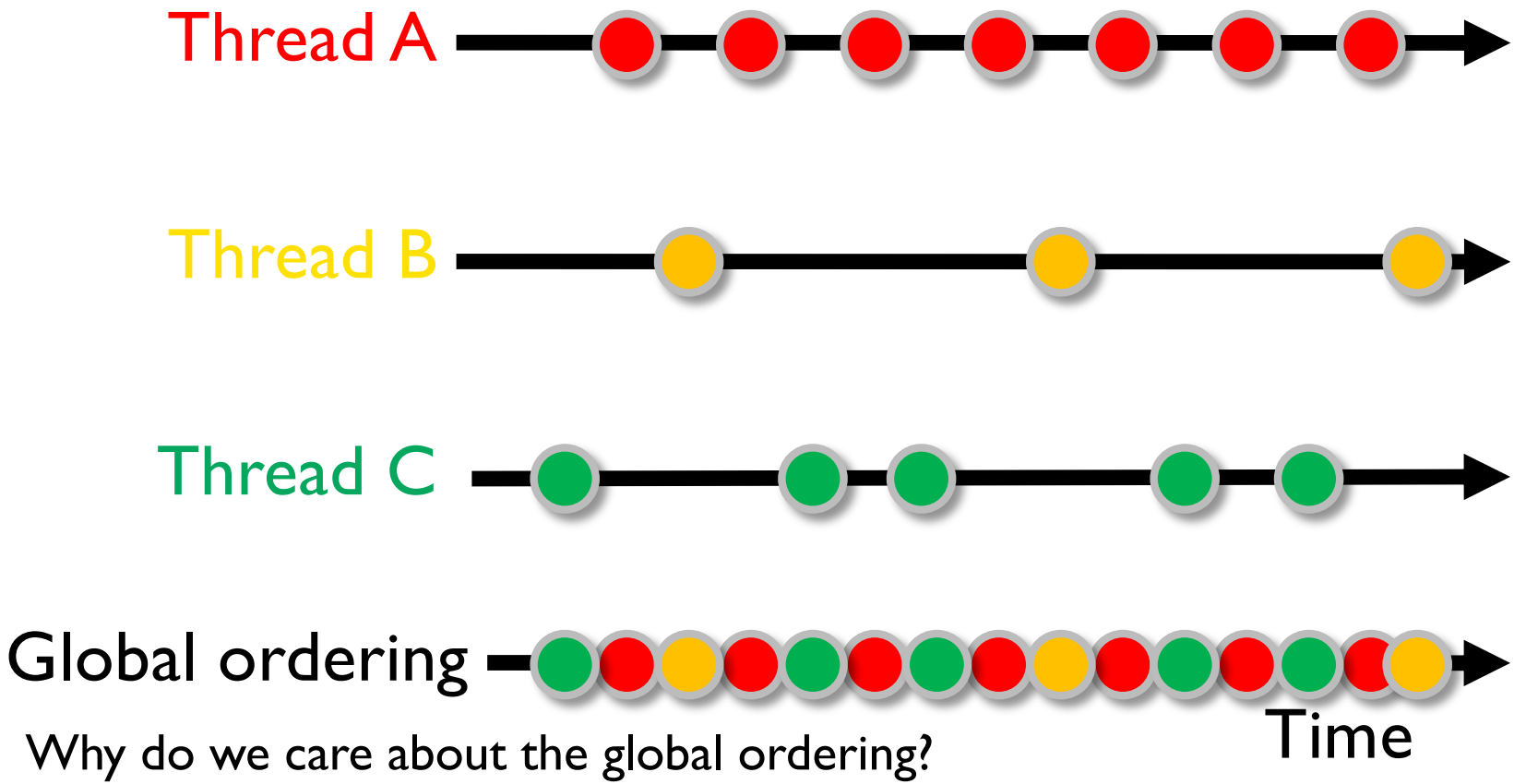
Cooperating threads

- Assume each thread has its own CPU
  - We will relax this assumption later



- CPUs run at unpredictable speeds
  - Source of non-determinism

Non-determinism and Ordering



Why do we care about the global ordering?

- Might have dependencies between events
- Different orderings can produce different results
- Not all operations "commutative" (actually very few)

Why is this ordering unpredictable?

- Can't predict how fast processors will run

Non-determinism example 1

- **Thread A**: `cout << "ABC";`
- **Thread B**: `cout << "123";`
- Possible outputs?
  - "A1BC23", "ABC123", …
- Impossible outputs? Why?
  - "321CBA", "B12C3A", …
- What is shared between threads?
  - Screen, maybe the output buffer

Non-determinism example 2

- `y = 10;`
- **Thread A**: `int x = y+1;`
- **Thread B**: `y = y*2;`
- Possible results?
  - **A** goes first: x = 11 and y = 20
  - **B** goes first: y = 20 and x = 21
- What is shared between threads?
  - Variable y

Non-determinism example 3

- `x = 0;`
- **Thread A**: `x = 1;`
- **Thread B**: `x = 2;`
- Possible results?
  - **B** goes first: x = 1
  - **A** goes first: x = 2
- Is x = 3 possible?

Example 3, continued

- What if "`x = <int>;`" is implemented as
  - `x := x & 0`
  - `x := x | <int>`
- Consider this schedule
  - Thread A: `x := x & 0`
  - Thread B: `x := x & 0`
  - Thread B: `x := x | 1`
  - Thread A: `x := x | 2` Result: x=3!
- The "equals" operator is actually two steps!

Atomic operations

- Must know what operations are **atomic**
  - …before we can reason about cooperation
- **Atomic**
  - Indivisible
  - Happens without interruption
- Between start and end of atomic action…
  - …no events from other threads can occur

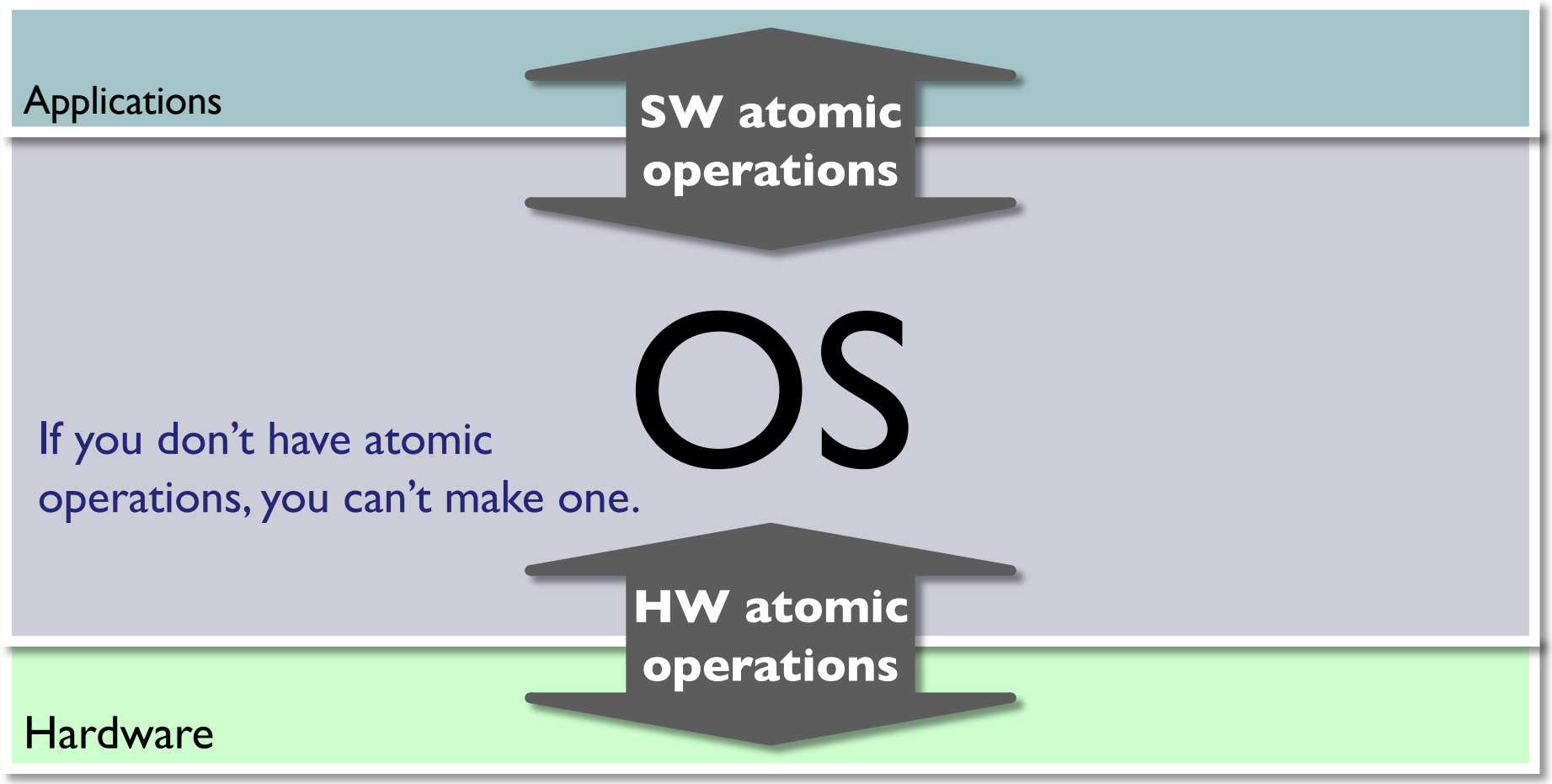
Review of examples

- Print example (ABC, 123)
  - What did we assume was atomic?
  - What if "print" is atomic?
  - What if printing a char was not atomic?
- Arithmetic example (`x=y+1`, `y=y*2`)
  - What did we assume was atomic?

Atomicity in practice

- On *most* (but not all) machines
  - Memory assignment/reference is atomic
  - E.g.: **`a=1, a=b`**
- Many other instructions are not atomic
  - E.g.: double-precision floating point store
  - (often involves two memory operations)

Virtual/physical interfaces



Another example

- Two threads (A and B)
  - A tries to increment i
  - B tries to decrement i

Thread A:

```
i = 0;
while (i < 10){
   i++;
}
print "A done."
```

Thread B:

```
i = 0;
while (i > -10){
   i--;
}
print "B done."
```

Example continued

- Who wins?
- Does someone have to win?

Thread A:

```
i = 0;
while (i < 10){
   i++;
}
print "A done."
```

Thread B:

```
i = 0;
while (i > -10){
   i--;
}
print "B done."
```

Example continued

- Will it go on forever if both threads:
  - Start at about the same time
  - And execute at exactly the same speed?
  - Yes, if each C *statement* is atomic.

Thread A:

```
i = 0;
while (i < 10){
   i++;
}
print "A done."
```

Thread B:

```
i = 0;
while (i > -10){
   i--;
}
print "B done."
```

Example continued

- What if `i++/i--` are not atomic?
  - `tmp := i+1`
  - `i := tmp`
  - (`tmp` is private to A and B)

Example continued

- Non-atomic `i++/i--`
  - If A starts ½ statement ahead, B can win
- How?

```
Thread A: tmpA := i + 1 // tmpA == 1
Thread B: tmpB := i - 1 // tmpB == -1
Thread A: i := tmpA     // i == 1
Thread B: i := tmpB     // i == -1
```

Example continued

- Non-atomic `i++/i--`
  - If A starts ½ statement ahead, B can win
- How?
- Do you need to worry about this?
  - Yes!!! No matter how unlikely

Debugging non-determinism

- Requires **worst-case** reasoning
  - Eliminate **all** ways for program to break
- Debugging is hard
  - Can't test all possible interleavings
  - Bugs may only happen sometimes

- **Heisenbug**
  - Re-running program may make the bug disappear
  - Doesn't mean it isn't still there!