ECE570/670

David Irwin

Lecture 2

Administrative Details

- Website: https://courses.umass.edu/eceng570-deirwin/spring23
  - Username: **irwin-ece**, Password: **myclass**
  - *Calendar may change*
  - *Posted VirtualBox and UTM images on Resources page*

- Think about Assignment partners
  - Assignment 0 will be assigned next Thursday
    - Assignment groups of at most 2
    - Will send Google form link via mailing list
      - https://forms.gle/D1SGZcEeZwNeGD7F7
      - Include NetID of both group members in email
      - He will send you back a username and password for the VM

Administrative Details

- Lampson paper for Tuesday
  - Hints for Computer System Design
  - Papers posted on website
- 670 paper reviews
  - Will setup and discuss reviewing system next week
- Posted office hours on website
  - Mondays 2:30pm – 3:30pm
  - Thursdays 9am – 10am

Administrative Details

- Paper Review
  - Briefly summarize Paper
  - What are the paper's strengths and contributions?
  - Why do you think this paper is considered novel/important?
    - Most of these papers are seminal, so they have few weaknesses
    - They were already accepted at the top venues

- See: https://people.inf.ethz.ch/troscoe/pubs/review-writing.pdf

Administrative Details

Overall merit

3. Weak accept

Reviewer expertise

3. Knowledgeable

Paper summary

This paper presents a measurement study of smart home devices and present multiple findings regarding usage patterns, security / privacy risks, and centralization of the IoT ecosystem in regards to content delivery endpoints and trackers / advertising. 200 homes are collected from over 19 days; uniquely, the homes tracked are users of smart home technology who opt in, with traffic from 1200 network connected devices in total.

Comments for author

I enjoyed this paper and applaud the authors for doing some empirical work to study existing smart home users habits, and bring to light useful discussion points on the whole in-home IoT ecosystem.

The study was principled and well organized. Much of the analysis seemed straightforward (diurnal nature, the level of traffic per device type) but the data confirming intuition is useful and their are interesting nuggets especially concerning how centralized the entire ecosystem actually is, which may be cause for concern with regulators.

I think there were some missed opportunities for analysis: for example the data from fingerprinting on this large number of homes merits further study. Number of devices and adoption of IoT, are things that could be studied, moreover, the interaction of the smart home users with the devices and their own opinions would have been a wonderful complement to the tracking data.

In general, the interaction of the users with the smart home devices and how that changes the traffic would be interesting to explore here.

Connection to why this data is useful and what it tells us to do are not made as clear. The MUD discussion was one possibility, but was shallow in terms of recommendations (just asking for more work to be done by administrators).

One concern in this paper is the lack of concrete recommendations or thinking on how the smart home should be constructed in the future to alleviate some of these concerns, or inspire regulators to take action.

I would also challenge the authors to think about how this dataset and study could benefit the community: what can be made available or what tool can be produced that lets other understand smart home usage in the wild? This type of thinking will greatly increase the impact of this paper.

Today's Outline

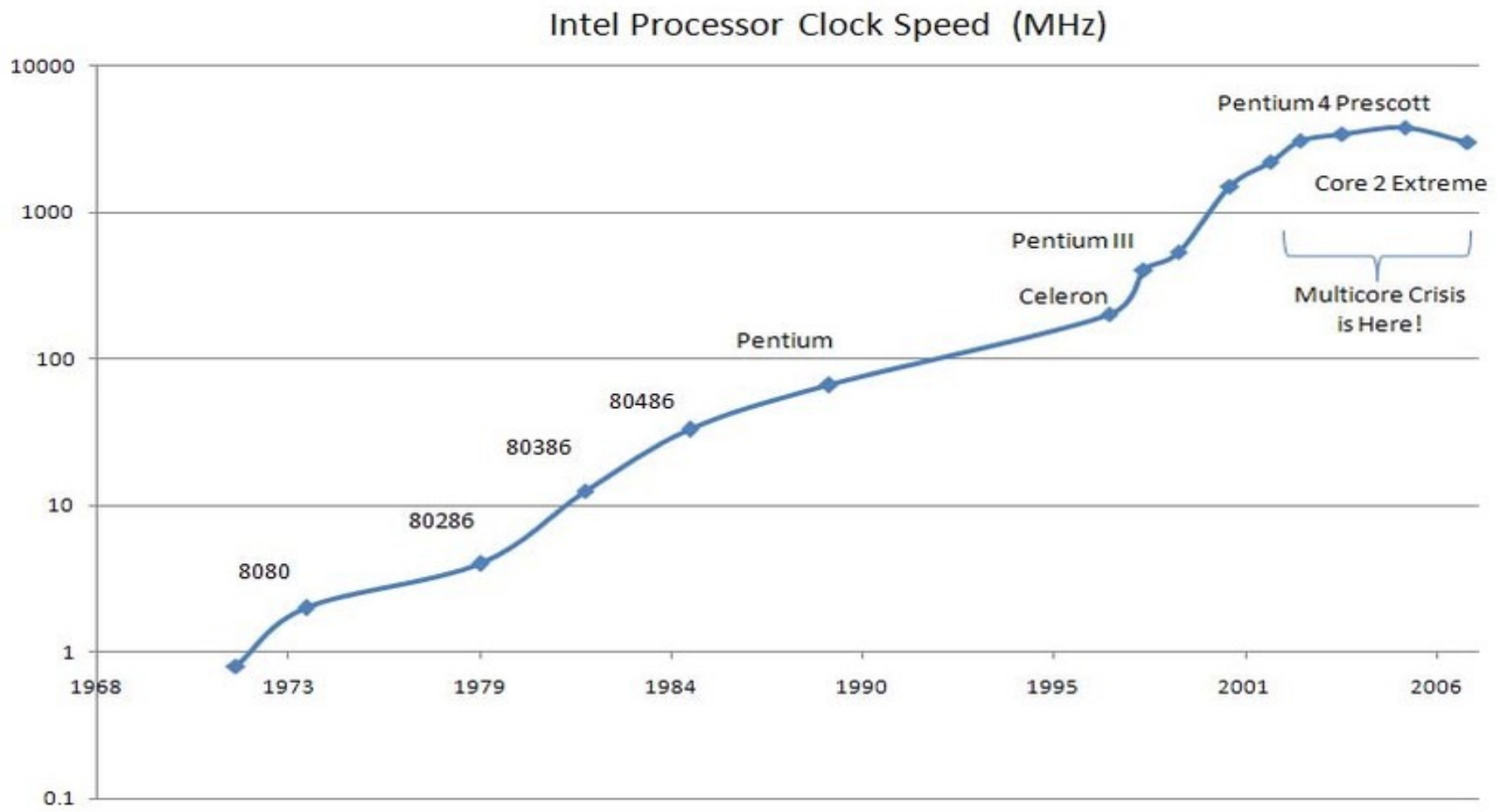
- Historical context wrap-up
  - Class focus – both *how* and *why* operating systems are designed the way they are
  - Will always start from the simplest solution
    - Identify problems with it, fix them, identify more problems, fix them, and so on until we get to modern systems
  - This is how systems research works

- Introduction and review processes and threads

Architectural Trends

- No counterpart in any other sphere of human existence!
  - Transportation:
    - 200 years: horseback (10 mph) to Concorde (1000 mph) = 2 orders of magnitude
  - Communication is closest:
    - 100 years: Pony Express (10 mph) to nearly speed of light (600 million mph) = 7 orders of magnitude

Are We Running Out of Steam?



http://smoothspan.wordpress.com/2007/09/06/a-picture-of-the-multicore-crisis/

Coming Soon…

- Efficiency much more important now
- "New features" must be considered…
  - Ever increasing number of cores
  - Specialized processors (GPUs, CPUs)
  - Serious power/heat constraints
    - Both on-chip and in data centers
    - Computing costs bounded by energy costs
    - Dark silicon
- Other tradeoffs possible
  - Trading computing power for reliability…

Moving on…

- Take-away messages:
  - Operating systems must continually change to keep up with changing hardware and user demands
  - Constant change has led to increased complexity (and functionality)

OS Complexity

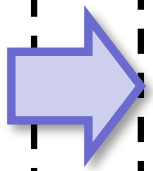
- Lines of code
  - Windows: 50 million
  - Linux 2.6: 6 million
  - (mostly driver code)

- Sources of complexity
  - Multiple instruction streams (processes)
  - Multiple interrupt sources (I/O, timers, faults)
- How can we keep everything straight?

Dealing with complexity

- Program decomposition
  - Functions
  - OO: Classes, types

```
int main () {
   cout << "input: ";
   cin >> input;
   output = sqrt (input);
   output = pow (output,3);
   cout << output << endl;
}
```



```
int main () {
   getInput ();
   computeResult ();
   printOutput ();
}
void getInput () {
   cout << "input: ";
   cin >> input;
}
void computerResult () {
   output = sqrt (input);
   output = pow (output,3);
}
void printOutput () {
   cout << output << endl;
}
```

Intro to processes

- Decompose activities into separate tasks
  - Allow them to run in parallel
  - "Independently" (what does this mean?)

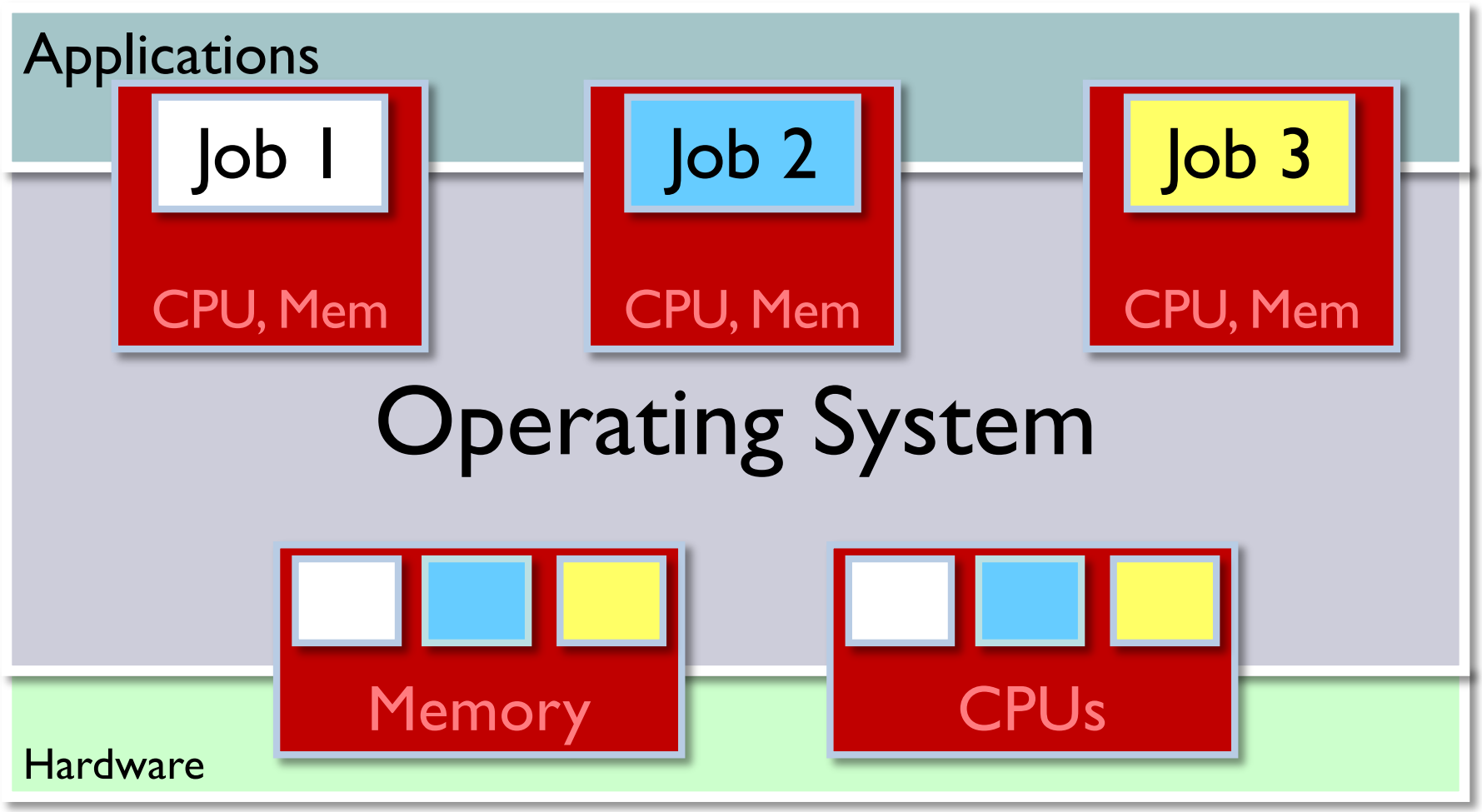
- Key OS abstraction: **processes**
  - Run independently of each other
  - Don't know about others (unless told)
  - Interprocess communication is explicit

Intro to processes

- For any area of OS, should always ask:
  - What interface/abstractions does the hardware provide?
  - What interface/abstractions does the OS provide?

- What is physical reality?
  - Single computer (CPUs + memory)
  - Execute instructions from many programs
- What does an application see?
  - Each app thinks it has its own CPU + memory

Hardware, OS interfaces



What is a process?

- Informal
  - A program in execution
  - Running code + things it can read/write
  - Process **≠ program**
- Formal
  - ≥ 1 **thread(s)** in its own **address space**

Parts of a process

- **Thread**
  - Sequence of executing instructions
  - Active: does things

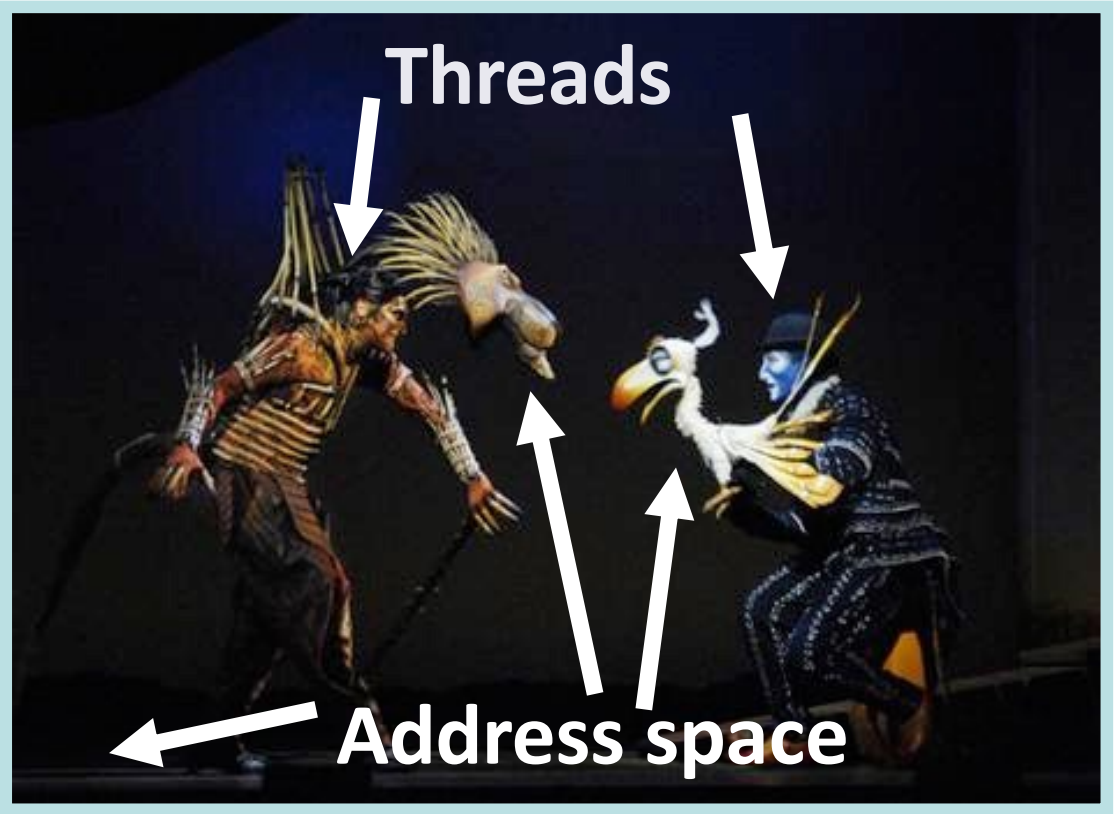
- **Address space**
  - Data the process uses/accesses as it runs
  - Passive: acted upon by threads

Play analogy

- Process is like a play performance
  - Program is like the play's script

What are the threads?

What is the address space?



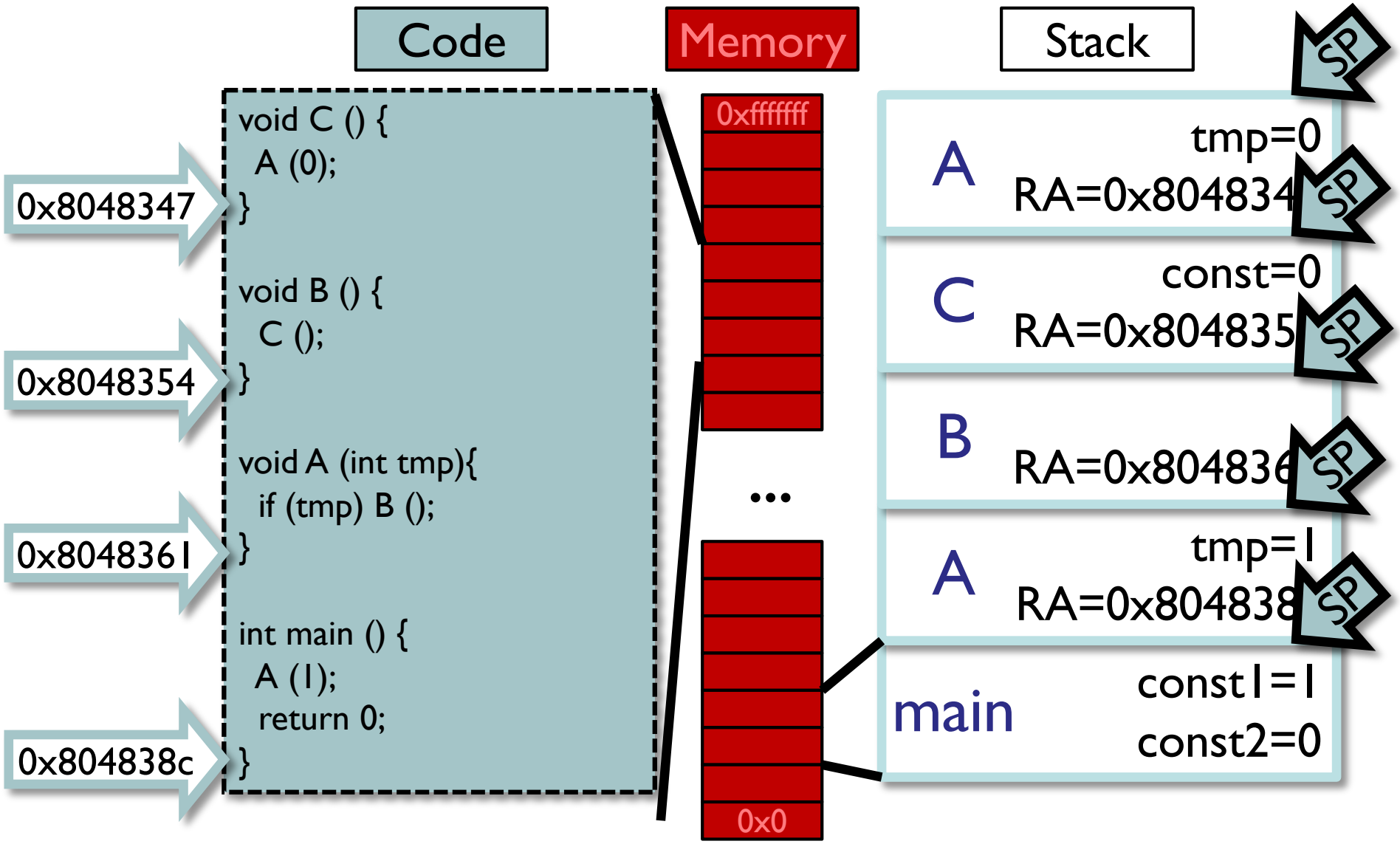
What is in the address space?

- Program code
  - Instructions, also called “text”
- Data segment
  - Global variables, static variables
  - Heap (where “new” memory comes from)
- Stack
  - Where local variables are stored

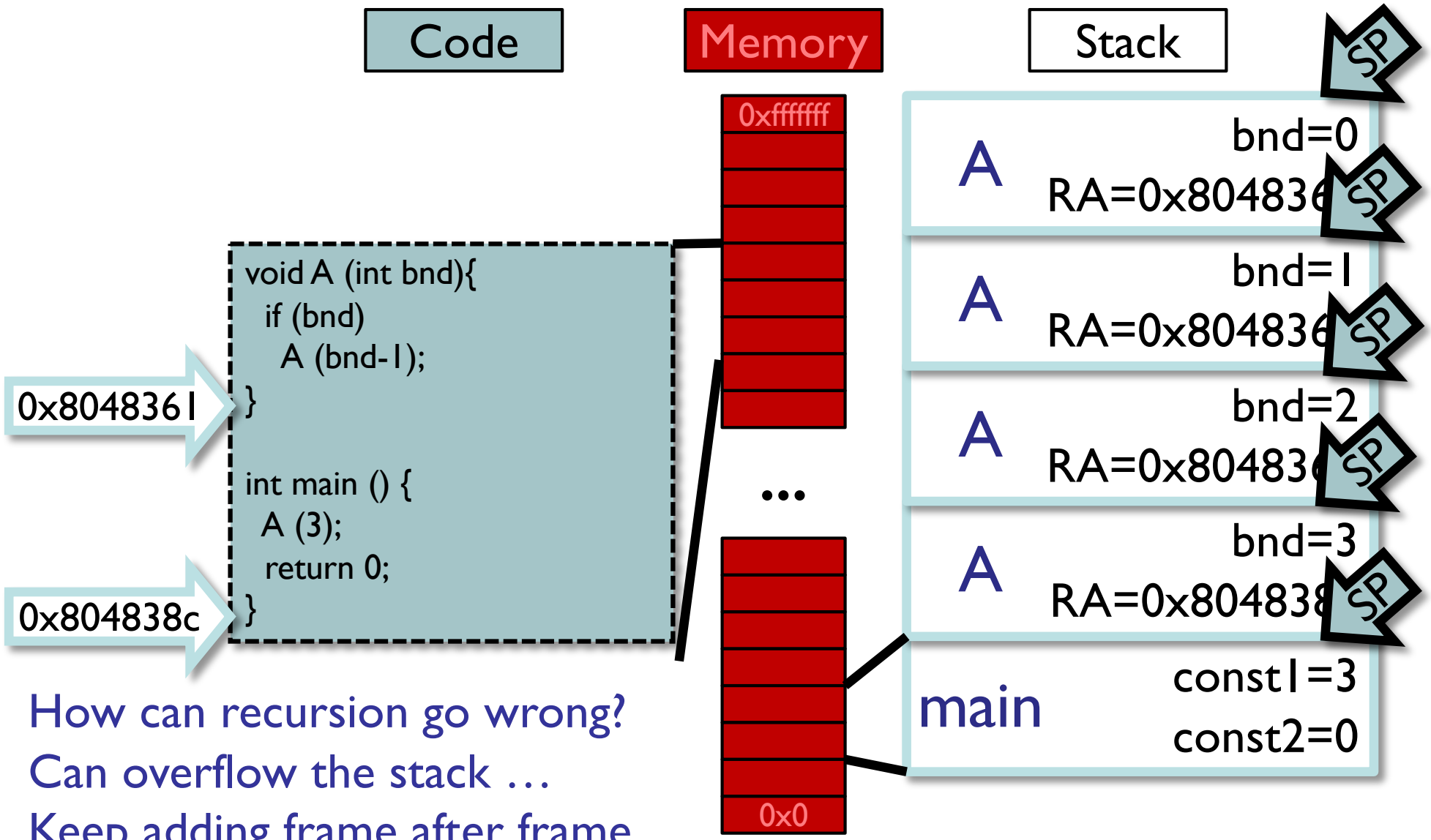
Review of the stack

- Each **stack frame** contains a *function's*…
  - …local variables
  - …parameters
  - …return address
  - …saved values of calling function's registers
- The stack enables recursion

Example stack



The stack and recursion

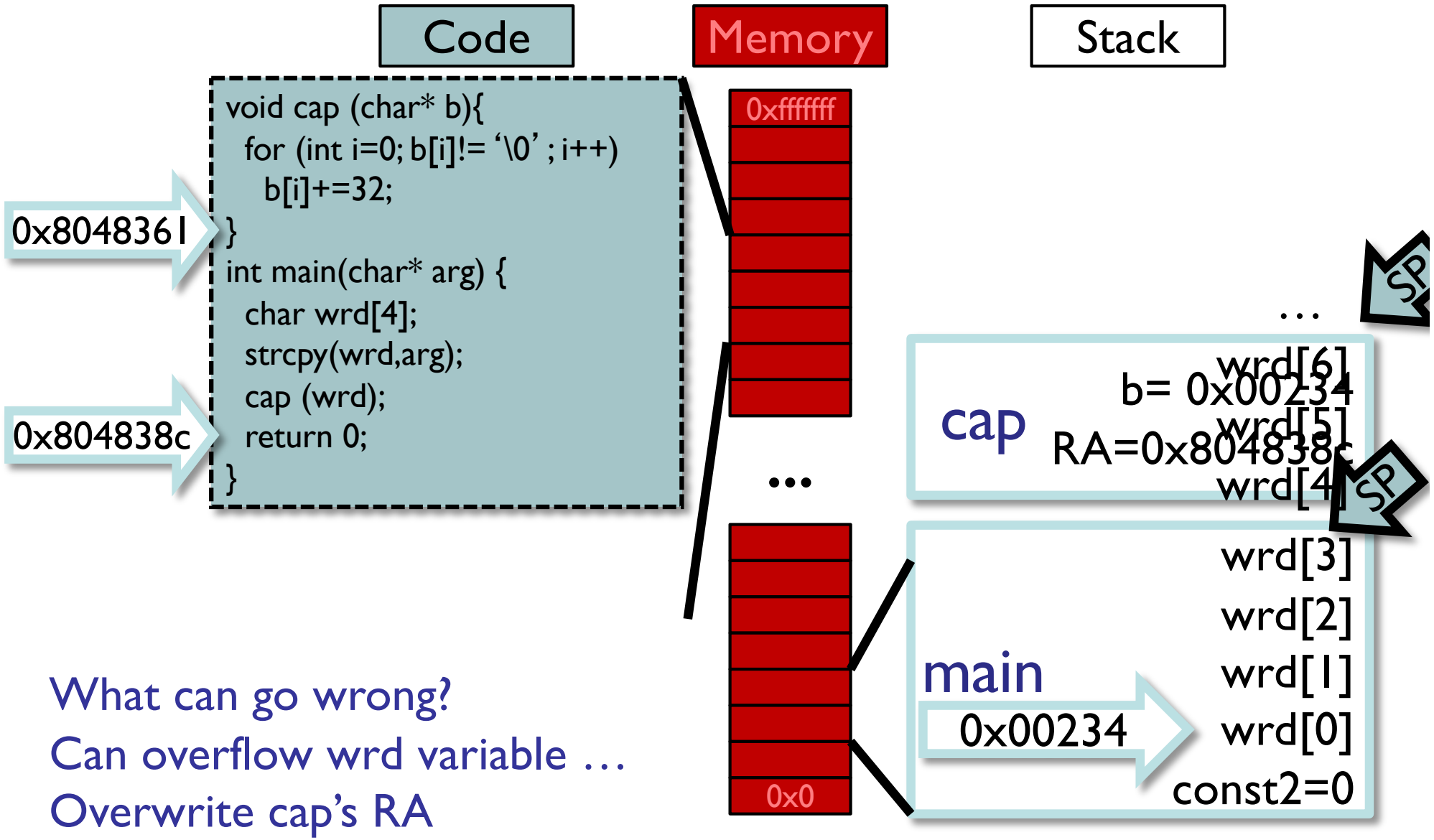


How can recursion go wrong?

Can overflow the stack …

Keep adding frame after frame

The stack and buffer overflows



What can go wrong?

Can overflow wrd variable …

Overwrite cap's RA

What is missing?

- What process state **isn't** in the address space?
  - Everything stored in CPU registers
  - General Purpose Registers
    - Memory that CPU is performing operations on
    - E.g., adding, subtracting, dividing, etc.
  - Special Purpose Registers
    - *Program counter (PC)* – memory address of next instruction to execute
    - *Instruction register (IR)* – memory address of current instruction executing
    - *Stack Pointer (SP)* – memory address of the top of the stack

Multiple threads in an addr space

- Several actors on a single set
  - Sometimes they interact (speak, dance)
  - Sometimes they are apart (different scenes)

Private vs global thread state

- What state is private to each thread?
  - PC (where actor is in his/her script)
  - Stack, SP (actor's mindset)
- What state is shared?
  - Global variables, heap (props on set)
  - Code (like lines of a play)

Looking ahead: concurrency

- **Concurrency**
  - Having multiple threads active at one time
  - Thread is the unit of concurrency
- Primary topics
  - How threads cooperate on a single task
  - How multiple threads can share the CPU
- Big part of 2 Assignments

Looking ahead: address spaces

- **Address space**
  - Unit of "state partitioning"
- Primary topics
  - Many address spaces sharing physical memory
  - Efficiency
  - Safety (protection)
- Main part of Assignment 3

Back to Threads: Thread independence

- Ideal decomposition of tasks:
  - Tasks are **completely independent**
- Is such a pure abstraction really feasible?
  - Word saves a pdf, starts acrobat, which reads the pdf?
  - Running audio player, while compiling your project?
- No, sharing creates dependencies!
  - Software resources (file, address space)
  - Hardware resources (CPU, monitor, keyboard)

True thread independence

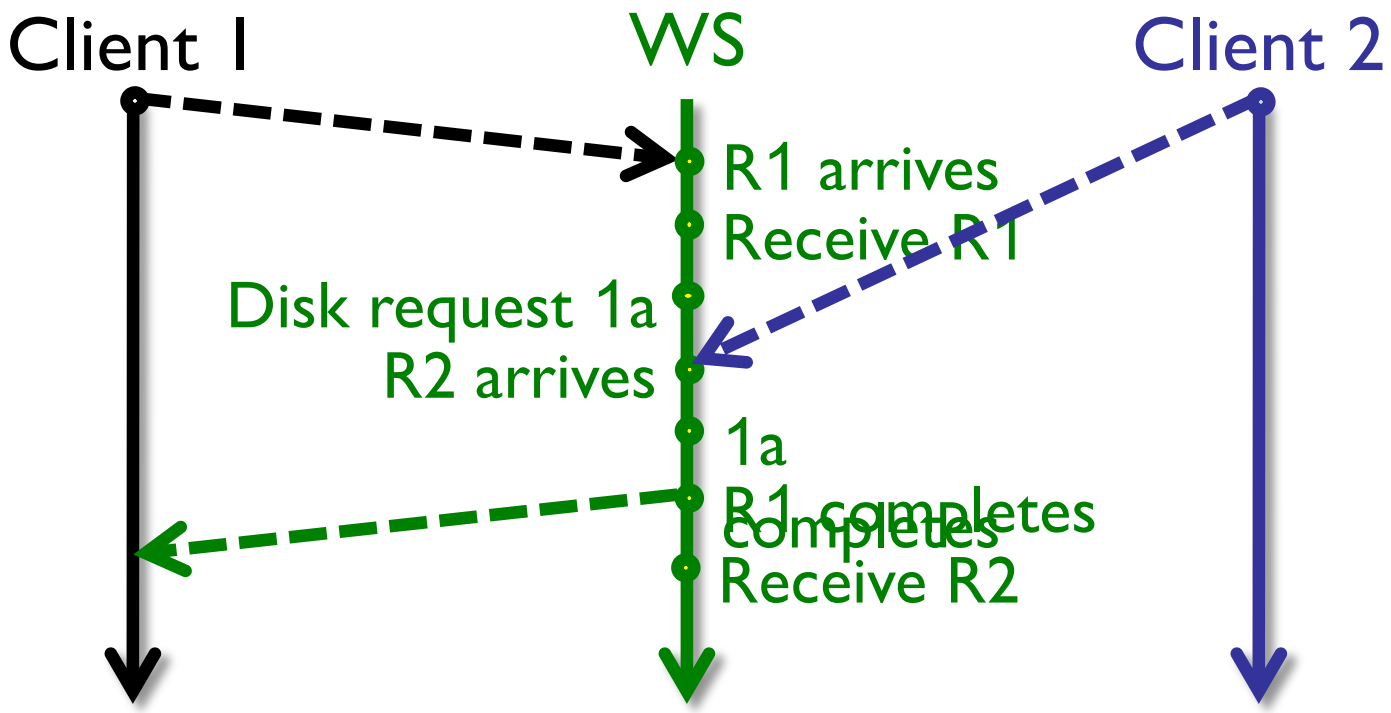
- What would pure independence actually look like?
  - (system with no shared software, hardware resources)
- Multiple computer systems
  - Each running non-interacting programs
  - Technically still share the power grid …
- "Pure" independence is infeasible (and not useful)
  - Tension between software dependencies
- Key question: is the thread abstraction still useful?
  - Easier to have one thread with multiple responsibilities?

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*

Web server (single thread)

- Option 1: could handle requests serially



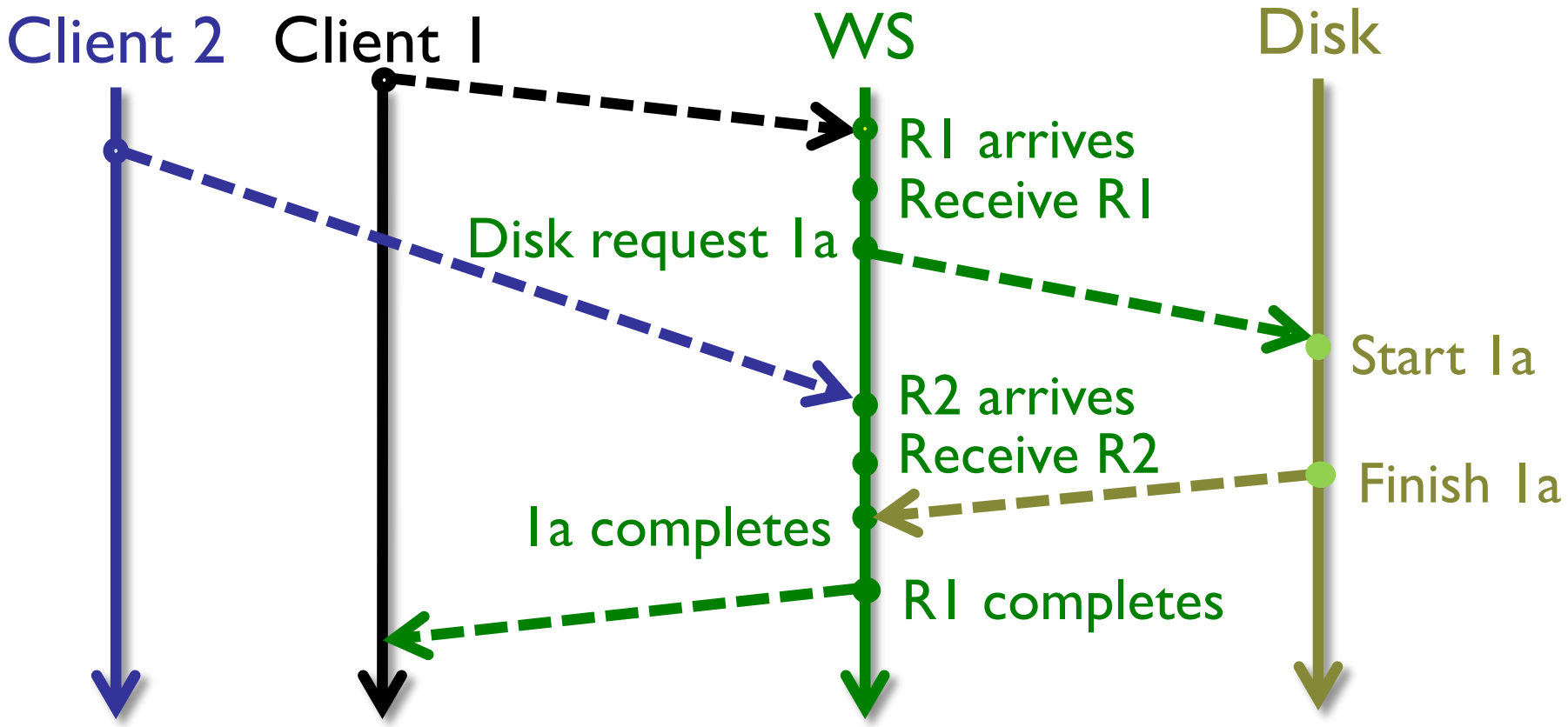
Web server (single thread)

- Option 1: handle one request at a time
  - Request 1 arrives
  - Server receives request 1
  - Server starts disk I/O for request 1a
  - Request 2 arrives
  - Server waits for I/O on request 1a to finish

  - Easy to program, but painfully slow (why?)

Web server (event driven)

- Option 2: use asynchronous I/O



Web server (event driven)

- Option 2: don't wait for I/Os to complete
  - Request 1 arrives
  - Server receives request 1
  - Server starts disk I/O on 1a to satisfy request 1
  - Request 2 arrives
  - Server receives request 2
  - Server starts disk I/O on 2a to satisfy request 2
  - Request 3 arrives
  - Disk I/O on 1a finishes

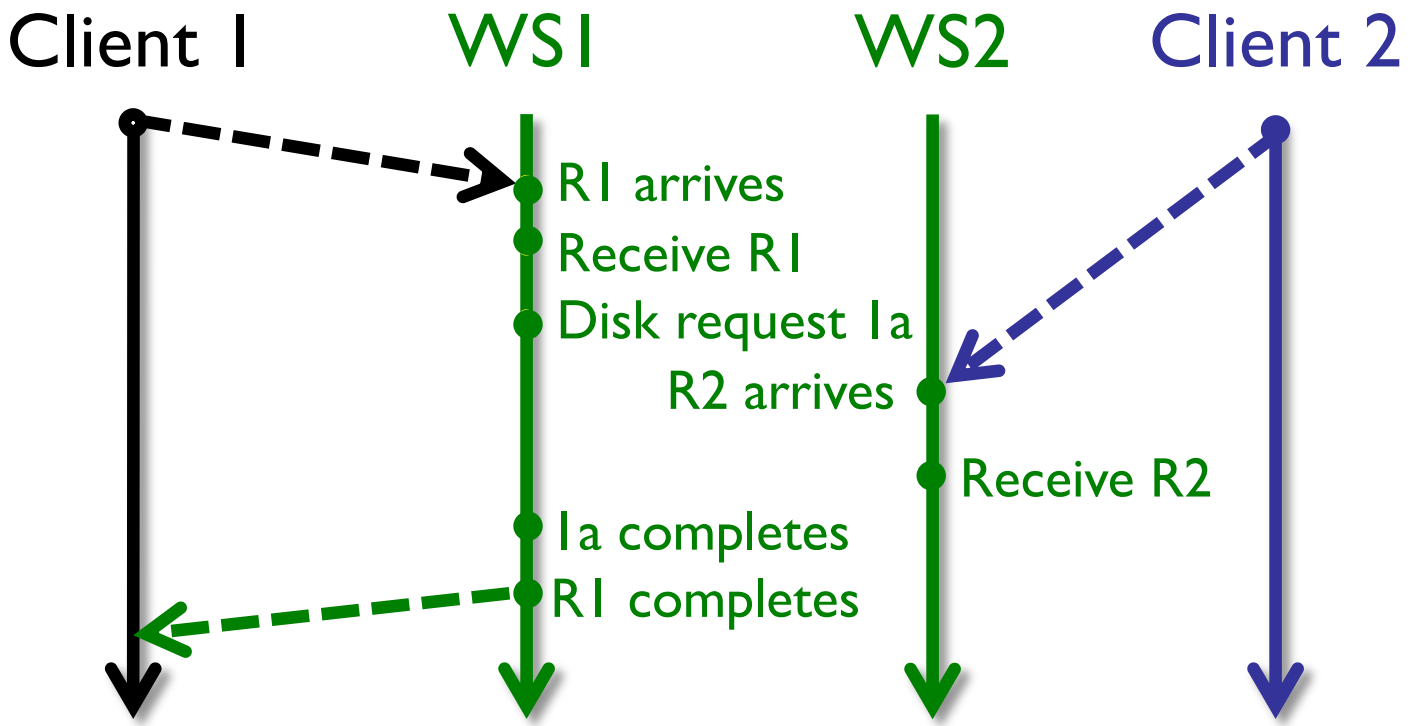
- Fast, but hard to program (why?)

Web server (event driven)

- Must remember lots of extra state!
  - What requests are being serviced, and what stage they're in
  - What disk I/Os are outstanding (and which requests they belong to)
  - Need to check back (poll) to see if outstanding requests have finished
    - Or handle interrupts when they finish
- Easy to get wrong
  - Any delay, delays every request (can kill performance)
  - Request are *highly* dependent on each other
  - Difficult to adjust quality-of-service for each request
    - What if you want to have some requests have higher priority than others?

Web server (multi-threaded)

- Option 3: assign one thread per request



- Where is each request's state stored?

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)