CS111, Lecture 17
Trust, Scheduling and Dispatching

Optional reading:

Operating Systems: Principles and Practice (2nd Edition): Chapter 7 up through Section 7.2

😷 masks recommended

Do Now:

1. Congratulate your neighbor on getting through the midterm!
2. Identify 2-3 people, services, or things you both trust and why.

😷 masks strongly recommended

CS111, Lecture 17

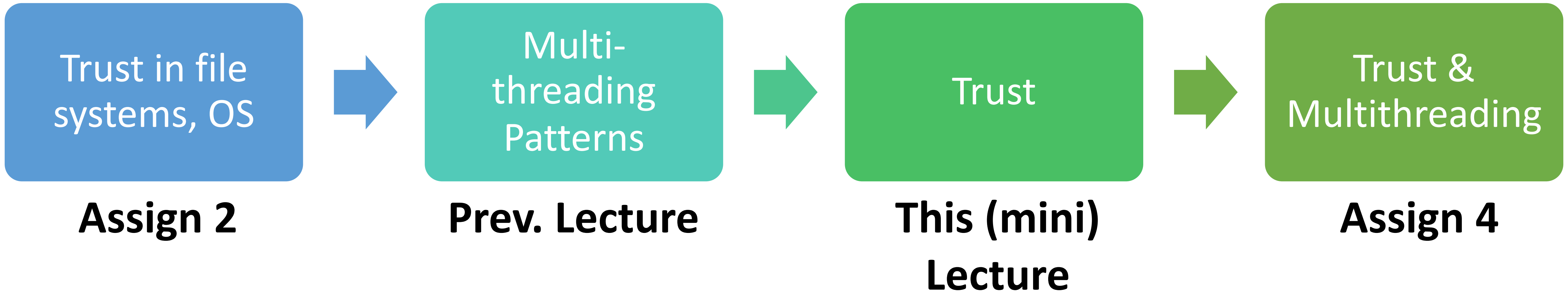
Trust and Operating Systems

Benjamin Xie, Ph.D.

Embedded EthiCS Fellow
benjixie@stanford.edu | benjixie.com

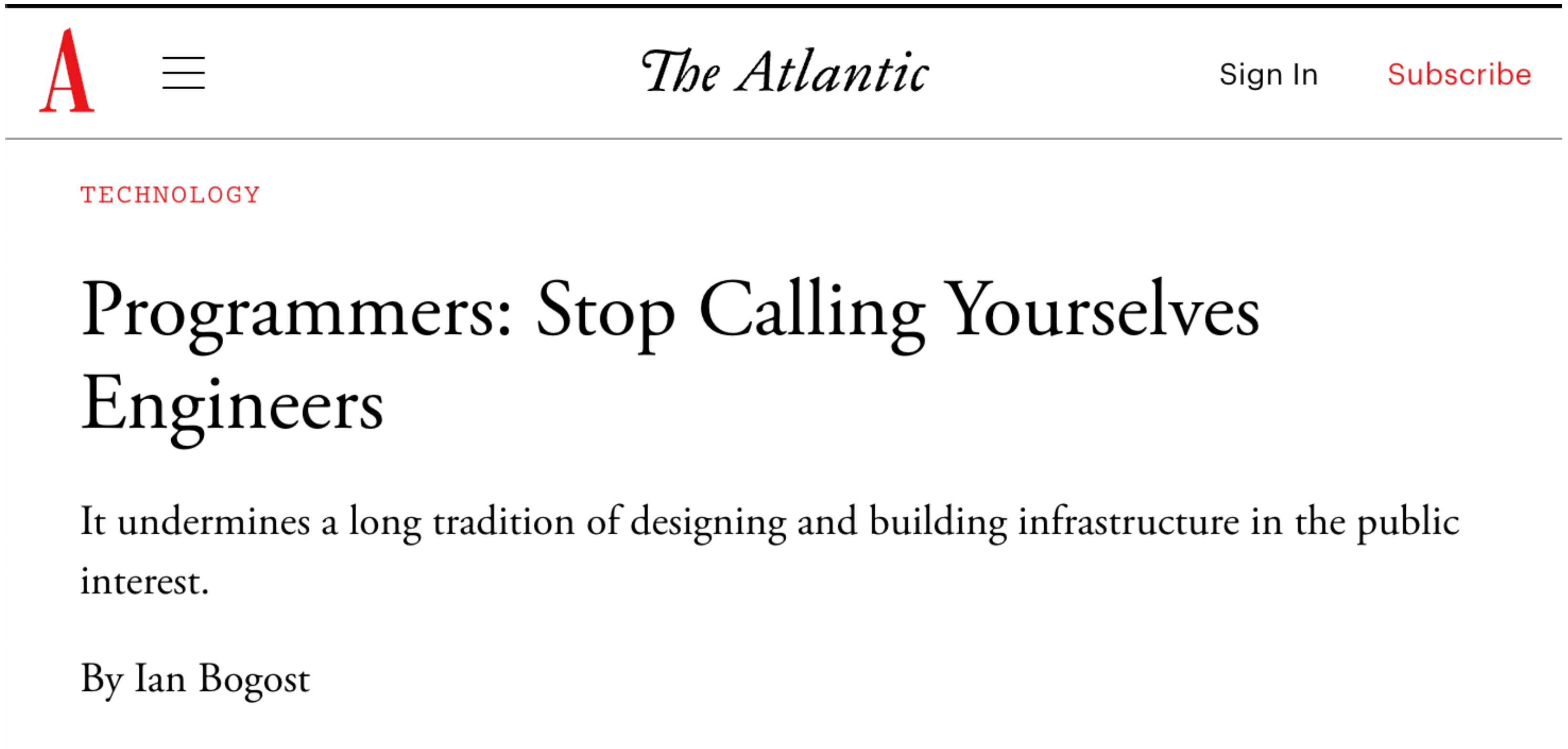
What do you trust? How do you warrant that trust?

CS111 Ethics Topic: Trust



Learning Goals

Understand how trust emerges and manifests in the context of operating systems



TECHNOLOGY

Programmers: Stop Calling Yourselves Engineers

It undermines a long tradition of designing and building infrastructure in the public interest.

By Ian Bogost

Plan For Today

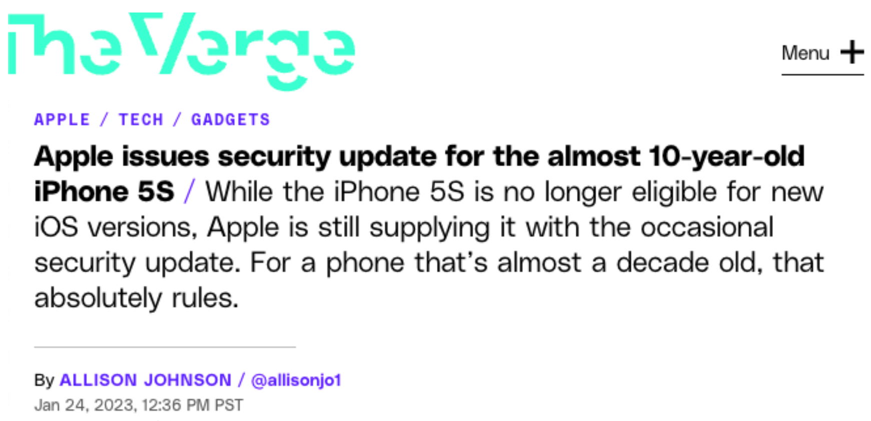
- Motivation: Importance of trust in OS
- What is trust?
- How does trust emerge?
- Example: Trusting Linux

Plan For Today

- **Motivation: Importance of trust in OS**
- What is trust?
- How does trust emerge?
- Example: Trusting Linux

Trust in OS for Standardization

- OS provides efficiency through standardization
- Users rely on technology built on OS
- App developers build off of OS
- OS creators make decisions that ripple far and long



The Verge

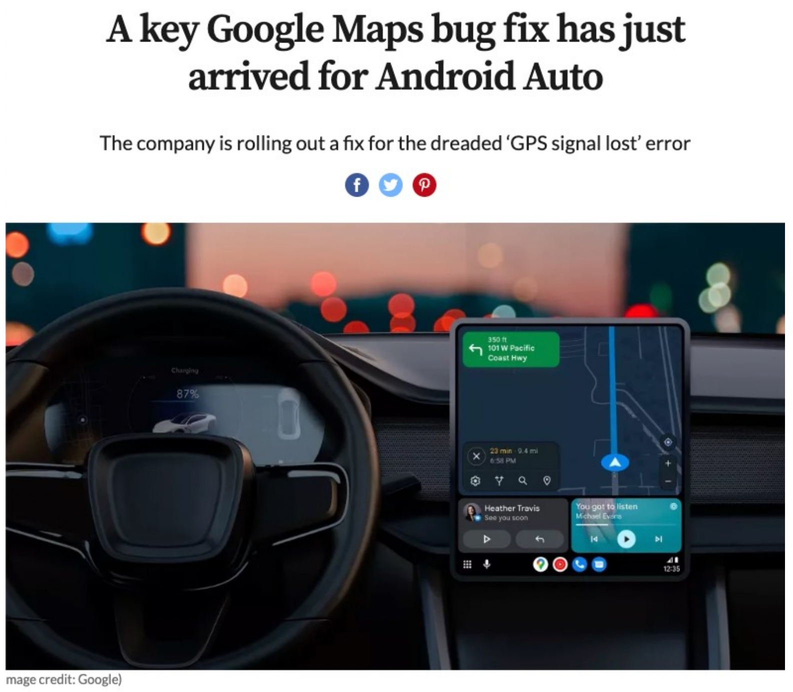
Menu +

APPLE / TECH / GADGETS

Apple issues security update for the almost 10-year-old iPhone 5S / While the iPhone 5S is no longer eligible for new iOS versions, Apple is still supplying it with the occasional security update. For a phone that's almost a decade old, that absolutely rules.

By ALLISON JOHNSON / @allisonjo1

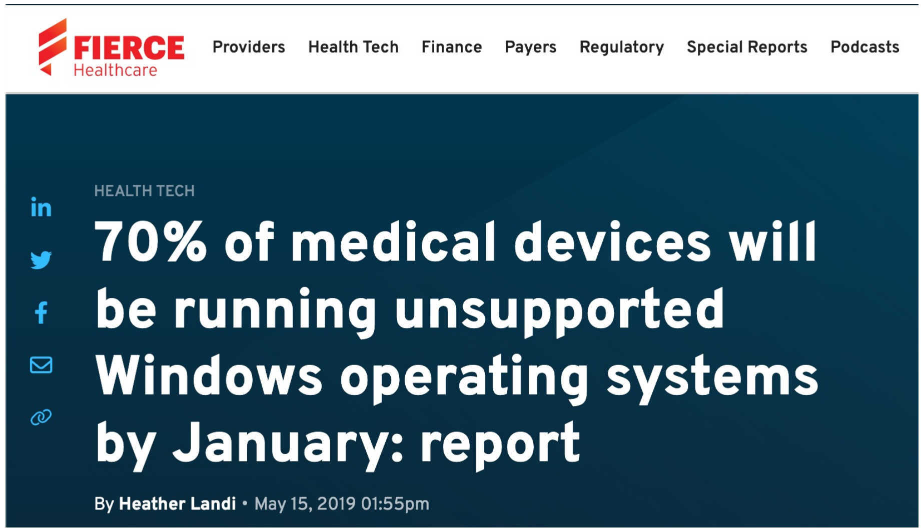
Jan 24, 2023, 12:36 PM PST



A key Google Maps bug fix has just arrived for Android Auto

The company is rolling out a fix for the dreaded 'GPS signal lost' error

mage credit: Google)



FIERCE Healthcare

Providers Health Tech Finance Payers Regulatory Special Reports Podcasts

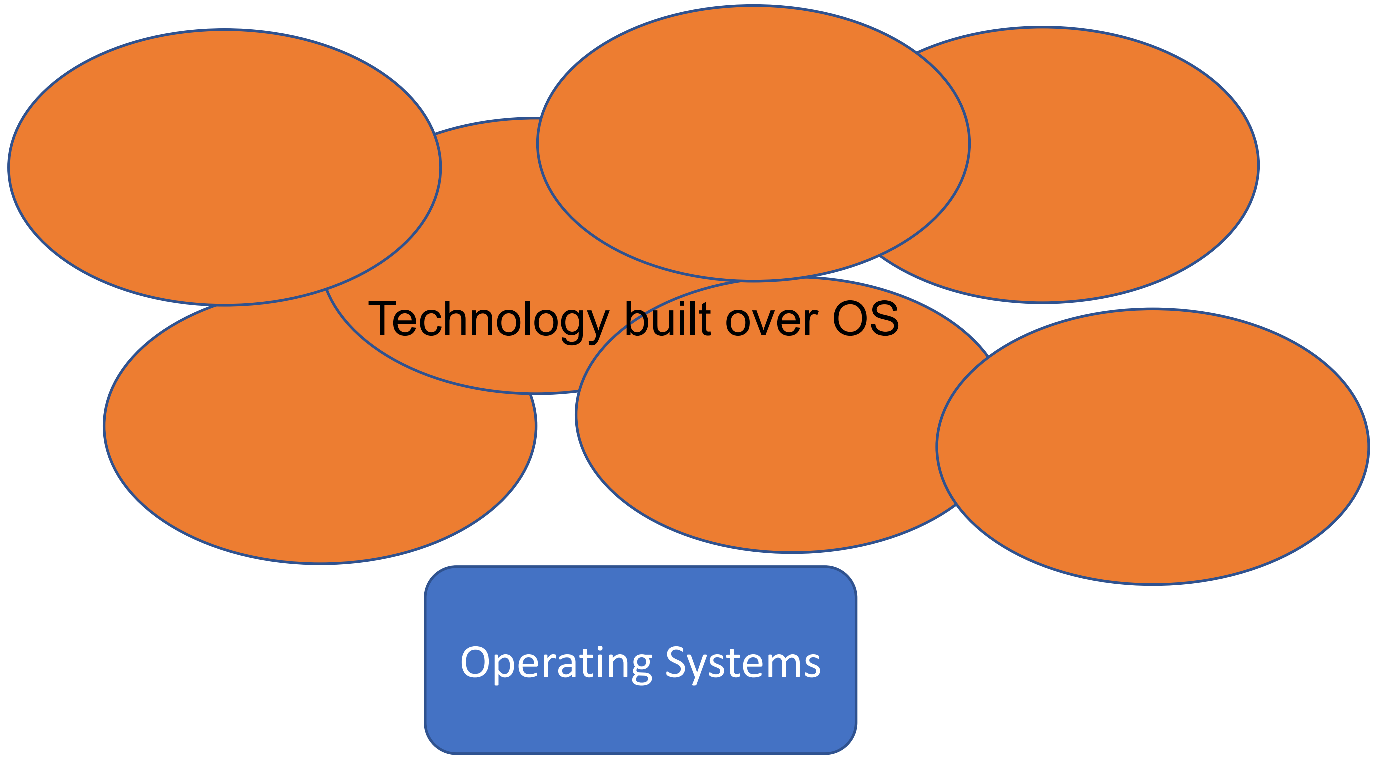
HEALTH TECH

70% of medical devices will be running unsupported Windows operating systems by January: report

By Heather Landi • May 15, 2019 01:55pm

Interaction between apps and OSs

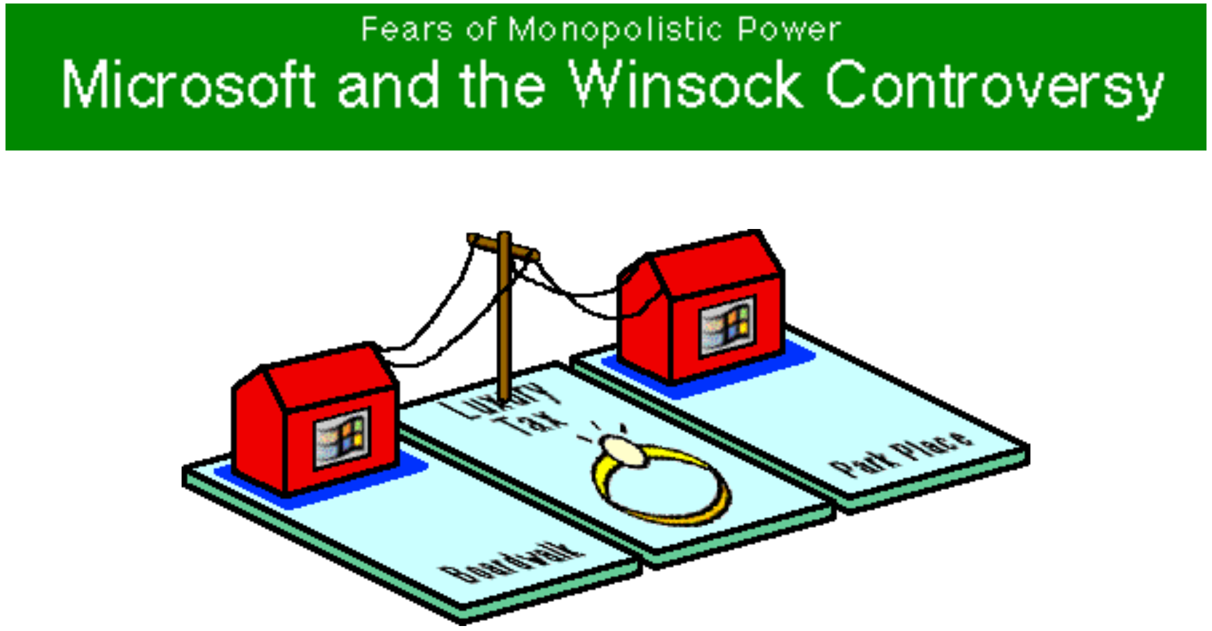
- Developers build off of OS
- OS creators make decisions that ripple far and long
- => changes affect each other



Examples of OS and app interactions

Changes to OS can affect applications

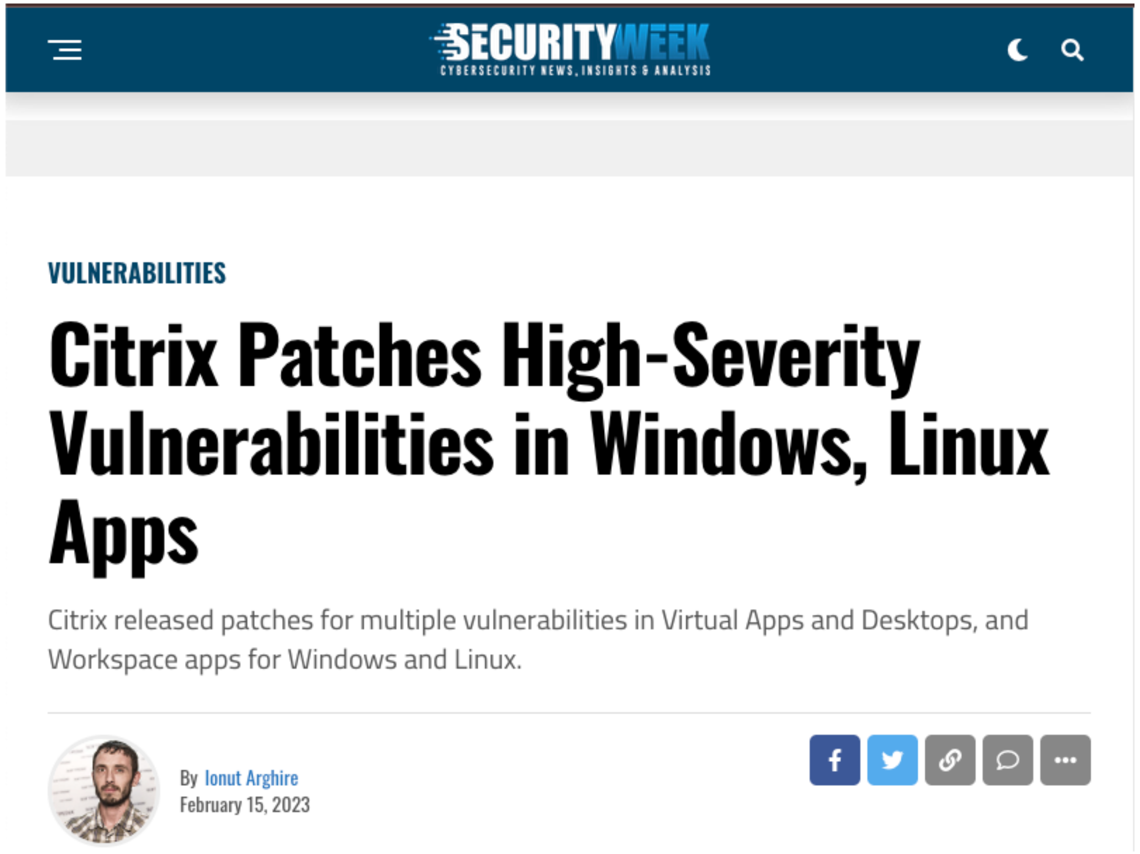
Example: Windows 95 disabled competitors' apps


Fears of Monopolistic Power
Microsoft and the Winsock Controversy

Eric Roberts, CS181

Changes to applications can affect OS usage

Example: virtual desktop introduces vulnerabilities



VULNERABILITIES

Citrix Patches High-Severity Vulnerabilities in Windows, Linux Apps

Citrix released patches for multiple vulnerabilities in Virtual Apps and Desktops, and Workspace apps for Windows and Linux.

By Ionut Arghire
February 15, 2023

Plan For Today

- Motivation: Importance of trust in OS
  - Trust amongst tech users, app developers, and OS developers is intertwined
- **What is trust?**
- How does trust emerge?
- Example: Trusting Linux

Trust as an unquestioning attitude

- Trust is to stop questioning the dependability of a thing
- Efficiency/safety tradeoff:
  - Trust lowers the barrier of monitoring, challenging, checking, and questioning
  - Results in more efficiency
- Involves
  - Intentions
  - Dependence
  - Vulnerability/Risk
- Examples?

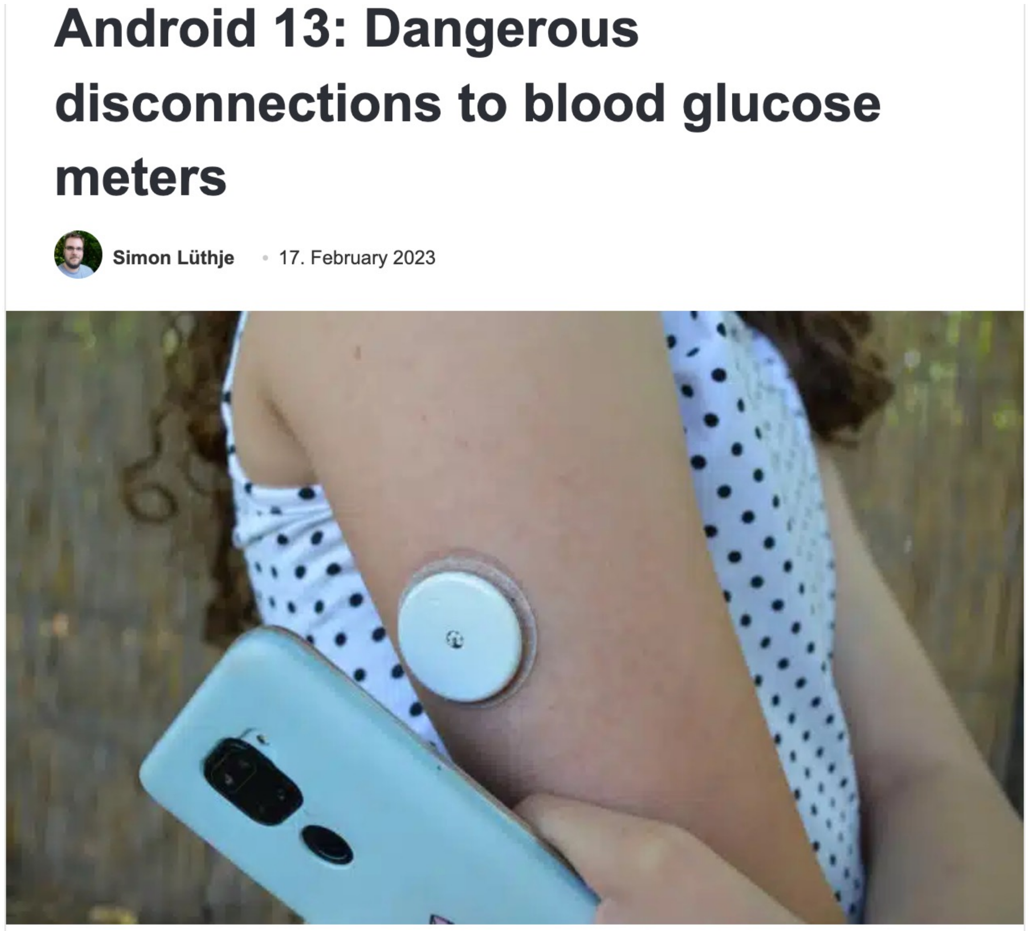
Trusting software is extending agency

- “when we trust, we try to make something a part of our agency, and we are betrayed when our part lets us down. To unquestioningly trust something is to let it in—to attempt to bring it inside one’s practical functioning.”
- Example: glucose monitoring

CT Nguyen: *Trust as an unquestioning attitude*

Risk: Agential Gullibility

- Trusting more than warranted
- Difficult to judge how trust is warranted given how quickly software changes, hard to inspect
- Example: glucose monitoring issues w/ Android update



Android 13: Dangerous disconnections to blood glucose meters

Simon Lüthje • 17. February 2023

Key takeaway

Trust is powerful and necessary, but caries enormous risk. Power of trust is inseparable from vulnerabilities it creates.

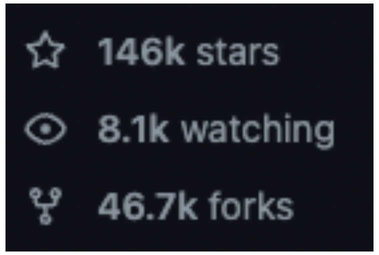
=> Trust (by extending agency) with great care!

Plan For Today

- Motivation: Importance of trust in OS
  - Trust amongst tech users, app developers, and OS developers is intertwined
- What is trust?
  - Extending agency to software
- **How does trust emerge?**
- Example: Trusting Linux

Three paths to trust

1. Assumption: trust absent any cluses to warrant it
    1. E.g. using unknown third party library b/c deadline nearing
2. Inference: reputation is based on past performance
    1. Log of past actions
    2. Trust in brands
    3. Trust in prior versions of software
3. Substitution: structural arrangements that partly substitute need for trust
    1. Often involves separation of code, responsibilities
    2. E.g. user permissions of file system, separating self-driving functionality of car from infotainment

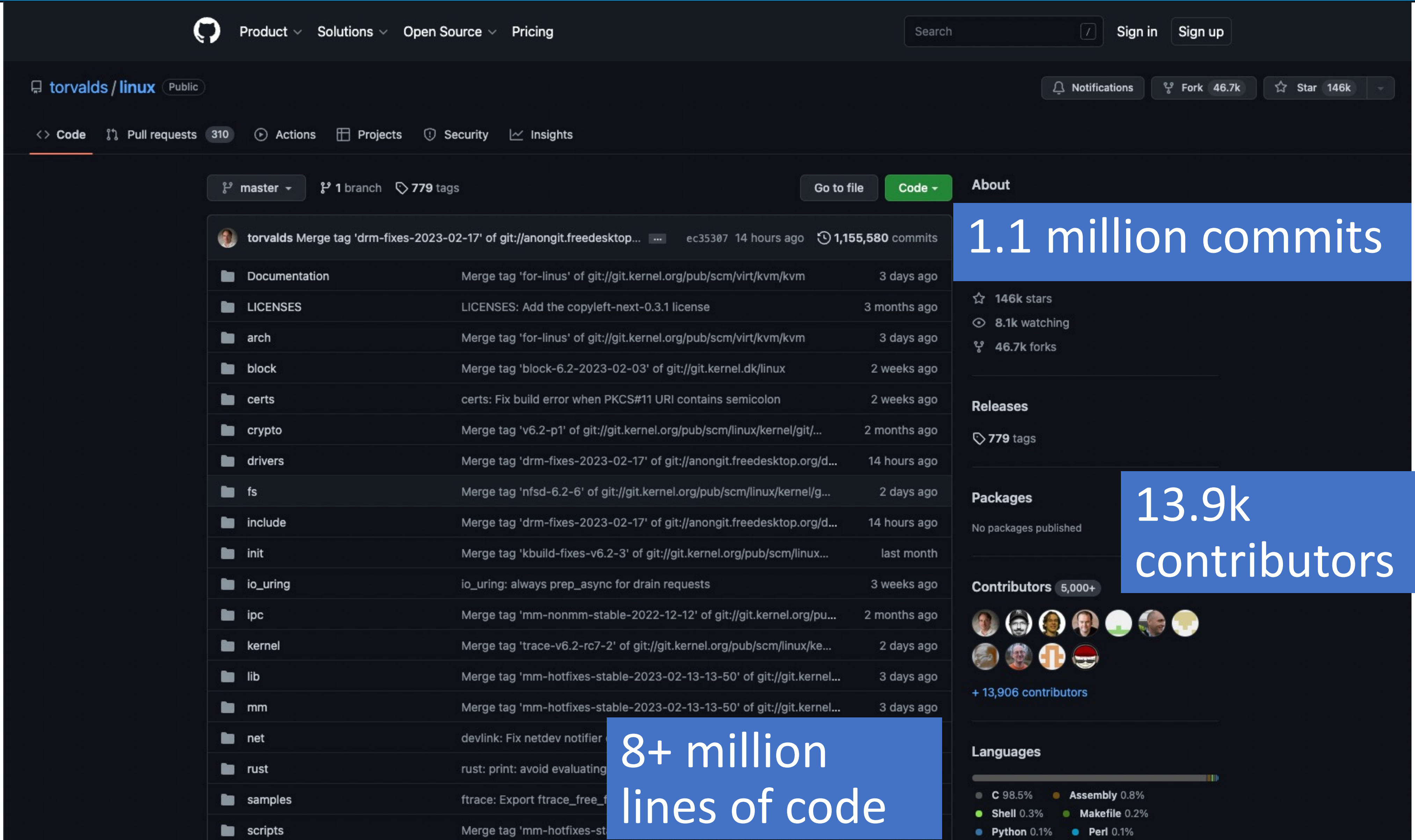


Paul B. de Laat: *How can contributors to open-source communities be trusted? On the assumption, inference, and substitution of trust*

Plan For Today

- Motivation: Importance of trust in OS
  - Trust amongst tech users, app developers, and OS developers is intertwined
- What is trust?
  - Extending agency to software
- How does trust emerge?
  - Assumption, inference, substitution
- **Example: Trusting Linux**

Linux is hard to trust


Product Solutions Open Source Pricing Search Sign in Sign up

torvalds / linux Public

Notifications | Fork 46.7k | Star 146k

Code | Pull requests 310 | Actions | Projects | Security | Insights

master | 1 branch | 779 tags | Go to file | Code

torvalds Merge tag 'drm-fixes-2023-02-17' of git://anongit.freedesktop... ec35307 14 hours ago 1,155,580 commits

Documentation	Merge tag 'for-linus' of git://git.kernel.org/pub/scm/virt/kvm/kvm	3 days ago
LICENSES	LICENSES: Add the copyleft-next-0.3.1 license	3 months ago
arch	Merge tag 'for-linus' of git://git.kernel.org/pub/scm/virt/kvm/kvm	3 days ago
block	Merge tag 'block-6.2-2023-02-03' of git://git.kernel.dk/linux	2 weeks ago
certs	certs: Fix build error when PKCS#11 URI contains semicolon	2 weeks ago
crypto	Merge tag 'v6.2-p1' of git://git.kernel.org/pub/scm/linux/kernel/git/...	2 months ago
drivers	Merge tag 'drm-fixes-2023-02-17' of git://anongit.freedesktop.org/d...	14 hours ago
fs	Merge tag 'nfsd-6.2-6' of git://git.kernel.org/pub/scm/linux/kernel/g...	2 days ago
include	Merge tag 'drm-fixes-2023-02-17' of git://anongit.freedesktop.org/d...	14 hours ago
init	Merge tag 'kbuild-fixes-v6.2-3' of git://git.kernel.org/pub/scm/linux...	last month
io_uring	io_uring: always prep_async for drain requests	3 weeks ago
ipc	Merge tag 'mm-nonmm-stable-2022-12-12' of git://git.kernel.org/pu...	2 months ago
kernel	Merge tag 'trace-v6.2-rc7-2' of git://git.kernel.org/pub/scm/linux/ke...	2 days ago
lib	Merge tag 'mm-hotfixes-stable-2023-02-13-13-50' of git://git.kernel...	3 days ago
mm	Merge tag 'mm-hotfixes-stable-2023-02-13-13-50' of git://git.kernel...	3 days ago
net	devlink: Fix netdev notifier	
rust	rust: print: avoid evaluating	
samples	ftrace: Export ftrace_free_f	
scripts	Merge tag 'mm-hotfixes-st	

About

146k stars
8.1k watching
46.7k forks

Releases

779 tags

Packages

No packages published

Contributors 5,000+

+ 13,906 contributors

Languages

● C 98.5%	● Assembly 0.8%
● Shell 0.3%	● Makefile 0.2%
● Python 0.1%	● Perl 0.1%

1.1 million commits

13.9k contributors

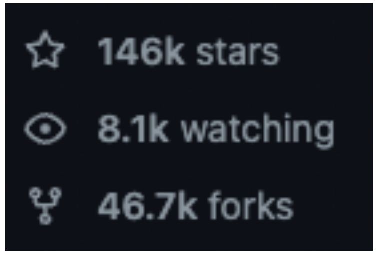
8+ million lines of code

Users Trusting Linux

- Why: People use Linux-based tools to extend their agency
  - Smartphones: Android based on Linux kernel
  - Servers: 13.6% of servers run on Linux
  - Supercomputing: 498/500 supercomputers run on Linux
- How trust emerges?
  - Assumption
    - “never thought about it”
    - ”had no other option”
  - Inference
    - Everyone else uses it.
    - Been around for awhile.
  - Substitution
    - Strong passwords
    - Careful about what data is stored on device

App Developers Trusting Linux

- Why: Standardization and tools of OS enable efficiency
  - High cost to build and maintain new OS
  - LINUX is familiar (UNIX-family of OS), lowers learning time developers
- How trust emerges?
  - Assumption: somewhat rare given affordances to suggest trust
  - Inference
    - Used by other app developers
    - “lots of stars on GitHub”
    - “I met Linus T. and he seemed nice”
  - Substitution
    - code is open source (read it, fork it)
    - “Redundant” checks in code (ex: spurious wakeup)


146k stars
8.1k watching
46.7k forks

OS Developers Trusting Linux

- Why: No single person can build & maintain an OS. Need to extend agency to others to support.
- How trust emerges?
  - Assumption: rarely happens (risks of bugs, exploits)
  - Inference
    - Known in community
    - Previously code submissions were high quality
  - Substitution
    - Formalization: tools and procedures to streamline cooperation
    - Division of roles
    - Decision making: Linus has final authority

"I don't like the idea of having developers do their own updates in my kernel source tree. (...) "there really aren't that many people that I trust enough to give write permissions to the kernel tree."
– Linus Torvalds

Recap

- Trust amongst tech users, app developers, and OS developers is intertwined
- Trust is about extending agency, enabling “unquestioning attitude”
- Trust emerges through assumption, inference, substitution
- Linux kernel to used broadly and large, so users, app developers, OS developers must trust through inference and substitution

Ethics takeaway: Trust is often required, powerful, and dangerous. Key design challenge is how we design structures that enable us to substitute trust.

Topic 3: Multithreading - How can we have concurrency within a single process? How does the operating system support this?

CS111 Topic 3: Multithreading

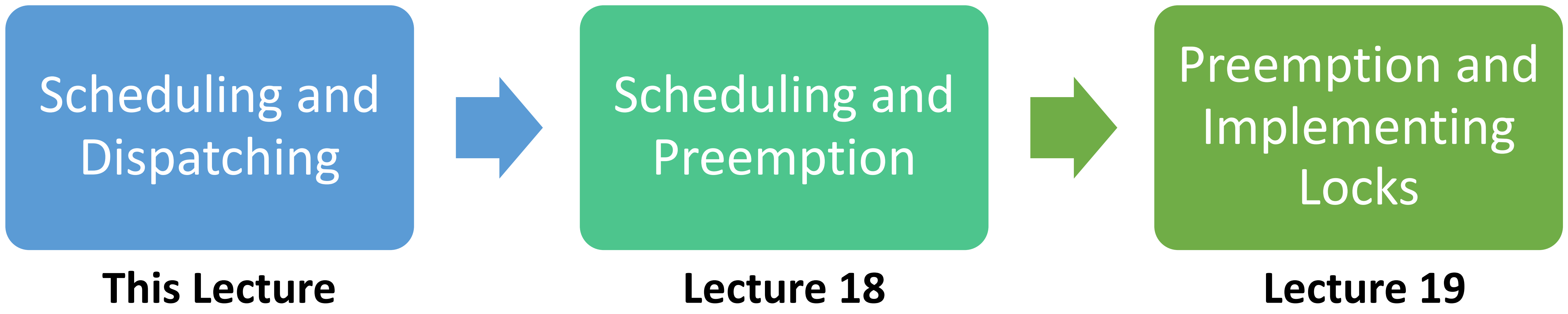
Multithreading - *How can we have concurrency within a single process?* ***How does the operating system support this?***

Why is answering this question important?

- Allows us to see how threads are represented and the fairness challenges for who gets to run next / for how long (next time)
- Shows us what the mechanism looks like for switching between running threads (today and next time)
- Allows us to understand how locks and condition variables are implemented (next week)

assign5: implement your own version of **thread**, **mutex** and **condition_variable**!

CS111 Topic 3: Multithreading, Part 2


Scheduling and Dispatching
Scheduling and Preemption
Preemption and Implementing Locks
This Lecture
Lecture 18
Lecture 19

assign5: implement your own version of **thread**, **mutex** and **condition_variable**!

Learning Goals

- Learn about how the operating system keeps track of threads and processes
- Understand the general mechanisms for switching between threads and when switches occur

Plan For Today

- **Overview:** Scheduling and Dispatching
- Process and Thread State
- Running a Thread
- Switching Between Threads

Plan For Today

- **Overview: Scheduling and Dispatching**
- Process and Thread State
- Running a Thread
- Switching Between Threads

Scheduling And Dispatching

- So far, we have learned about how user programs can create new processes and spawn threads in those processes
- But how does the operating system manage all of this internally? When we spawn a new thread or create a new process, what happens?

Key questions we will answer:

- How does the operating system track info for threads and processes?
- How does the operating system run a thread and switch between threads?
- How does the operating system decide which thread to run next?

Plan For Today

- **Overview:** Scheduling and Dispatching
- **Process and Thread State**
- Running a Thread
- Switching Between Threads

Process and Thread State

Key question #1: How does the operating system track info about threads and processes?

The OS maintains a (private) **process control block ("PCB")** for each process - a set of relevant information about its execution. Lives as long as the process does.

- Information about memory used by this process
- File descriptor table
- Info about threads in this process
- Other misc. accounting and info

Process and Thread State

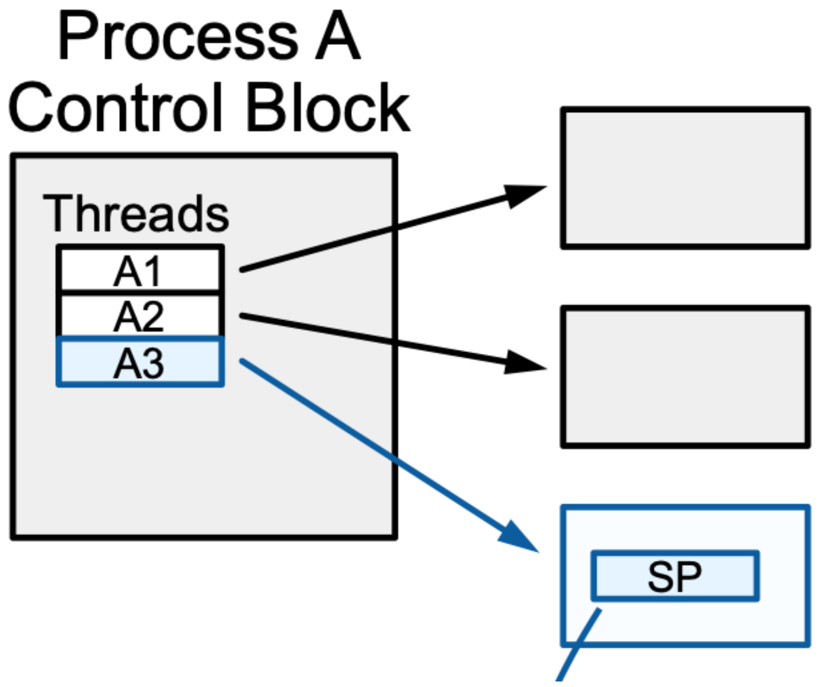
Key question #1: How does the operating system track info about threads and processes?

The OS maintains a (private) **process control block ("PCB")** for each process - a set of relevant information about its execution. Lives as long as the process does.

- Information about memory used by this process
- File descriptor table
- ***Info about threads in this process***
- Other misc. accounting and info

Thread State

- Every process has 1 main thread and can spawn additional threads.
- Threads are the "unit of execution" – processes aren't executed, threads are
- All main info in the PCB (e.g. memory info for the entire process) is relevant to all threads
- Each thread also has some of its own private info (e.g. stack location)
- Recall: there is a register called %rsp that points to the top of the stack ("stack pointer"). Non-running threads must save their %rsp somewhere for later.



Aside: x86-64 Assembly Refresher

- A **register** is a 64-bit space inside a processor core.
- Each core has its own set of registers.
- Registers are like “scratch paper” for the processor. Data being calculated or manipulated is moved to registers first. Operations are performed on registers.
- Registers also hold parameters and return values for functions.
- Some registers have special responsibilities – e.g. **%rsp** always stores the address of the current top of the stack.
- When a thread is being kicked off, it must remember its **%rsp** value so it knows where its stack is the next time it runs. (we’ll see how it remembers other register values later)

Plan For Today

- **Overview:** Scheduling and Dispatching
- Process and Thread State
- **Running a Thread**
- Switching Between Threads

Running a Thread

Key Question #2: How does the operating system run a thread and switch between threads?

- A processor has 1 or more “cores” - Each core contains a complete CPU capable of executing a thread
- Typically have more threads than cores, but most may not need to run at any given point in time (why? They are waiting for something)
- When the OS wants to run a thread, it loads its state (e.g. %rsp and other registers) into a core, and starts or resumes it
- **Problem:** once we run a thread, the OS is not running anymore! (e.g. 1 core) How does it regain control?

Regaining Control

There are several ways control can switch back to the OS:

1. "Traps" (events that require OS attention):
    1. System calls (like **read** or **waitpid**)
    2. Errors (illegal instruction, address violation, etc.)
    3. Page fault (accessing memory that must be loaded in) – more later…
2. "Interrupts" (events occurring outside current thread):
    1. Character typed at keyboard
    2. Completion of disk operation
    3. Timer – to make sure OS eventually regains control

At this point, OS could then decide to run a different thread.

Plan For Today

- **Overview:** Scheduling and Dispatching
- Process and Thread State
- Running a Thread
- **Switching Between Threads**

Switching Between Threads

When the OS regains control, how does it switch to run another thread?

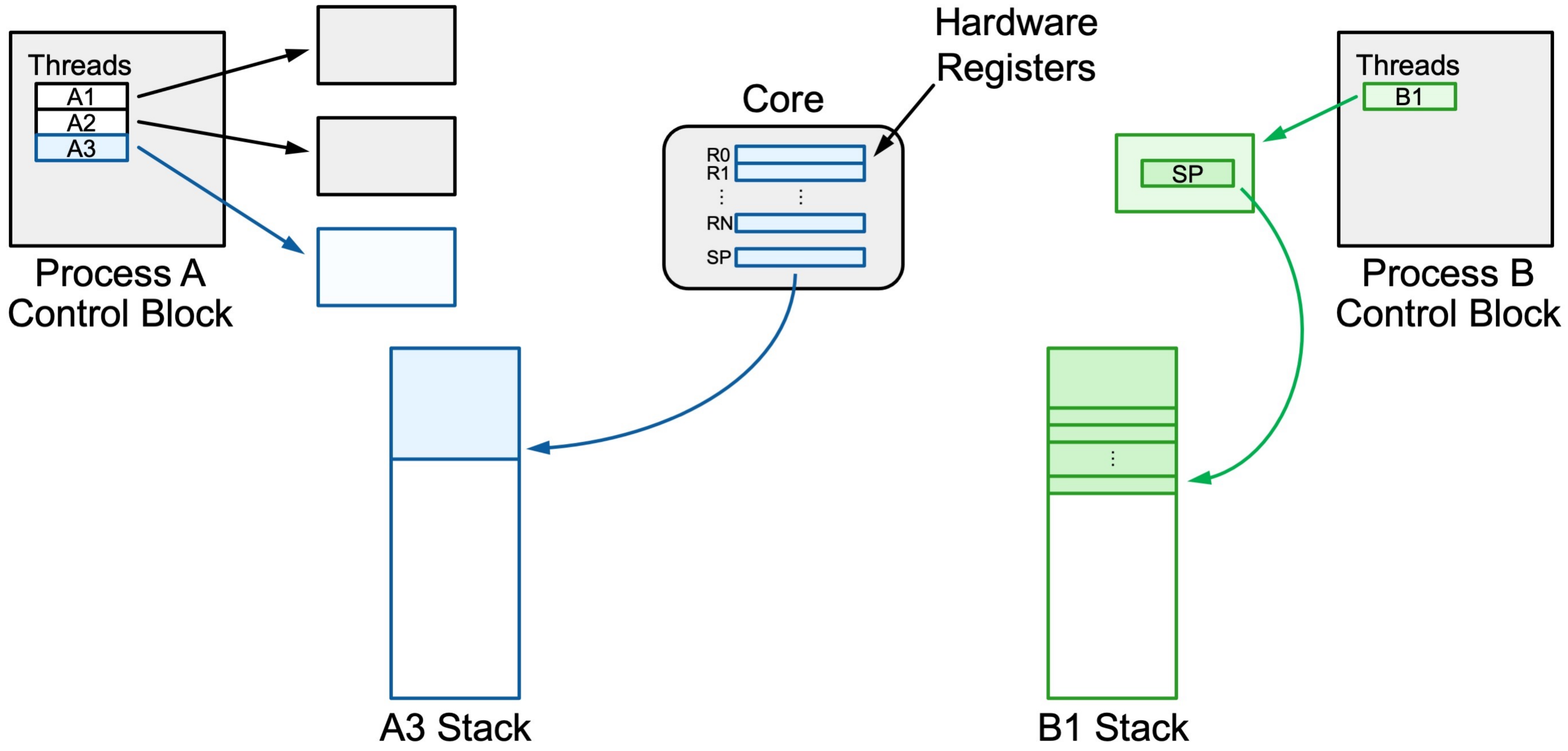
The **dispatcher** is OS code that runs on each core that switches between threads

- Not a thread – code that is invoked to perform the dispatching function
- Lets a thread run, then switches to another thread, etc.
- *Context switch* – changing the thread currently running to another thread. We must save the current thread state (registers) and load in the new thread state.
- Context switches are funky – like running a function that, as part of its execution, switches to a *completely different function in a completely different thread!!*

Demo: context-switch.cc

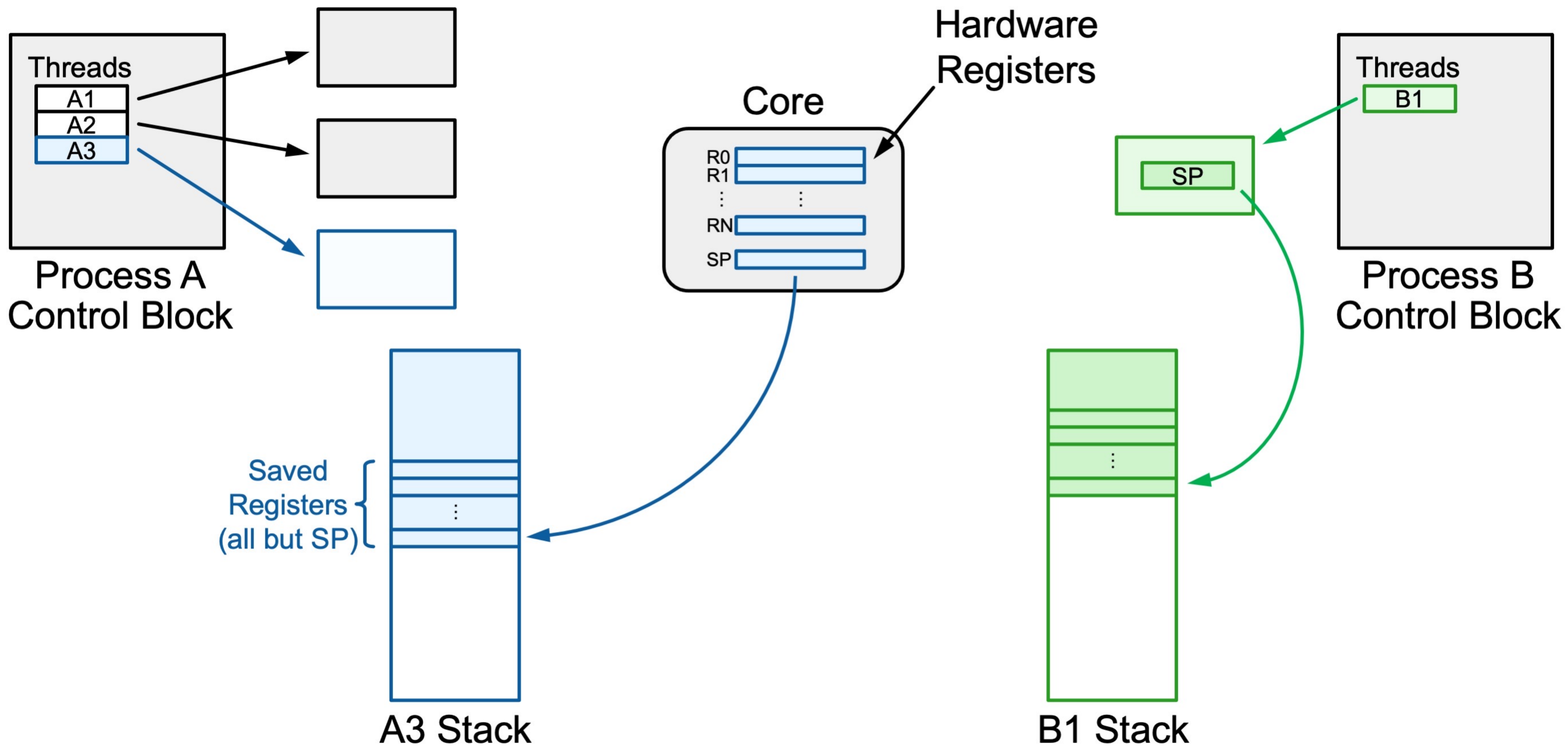
Context Switching

Context switch: how do we switch from thread A3 to thread B1?



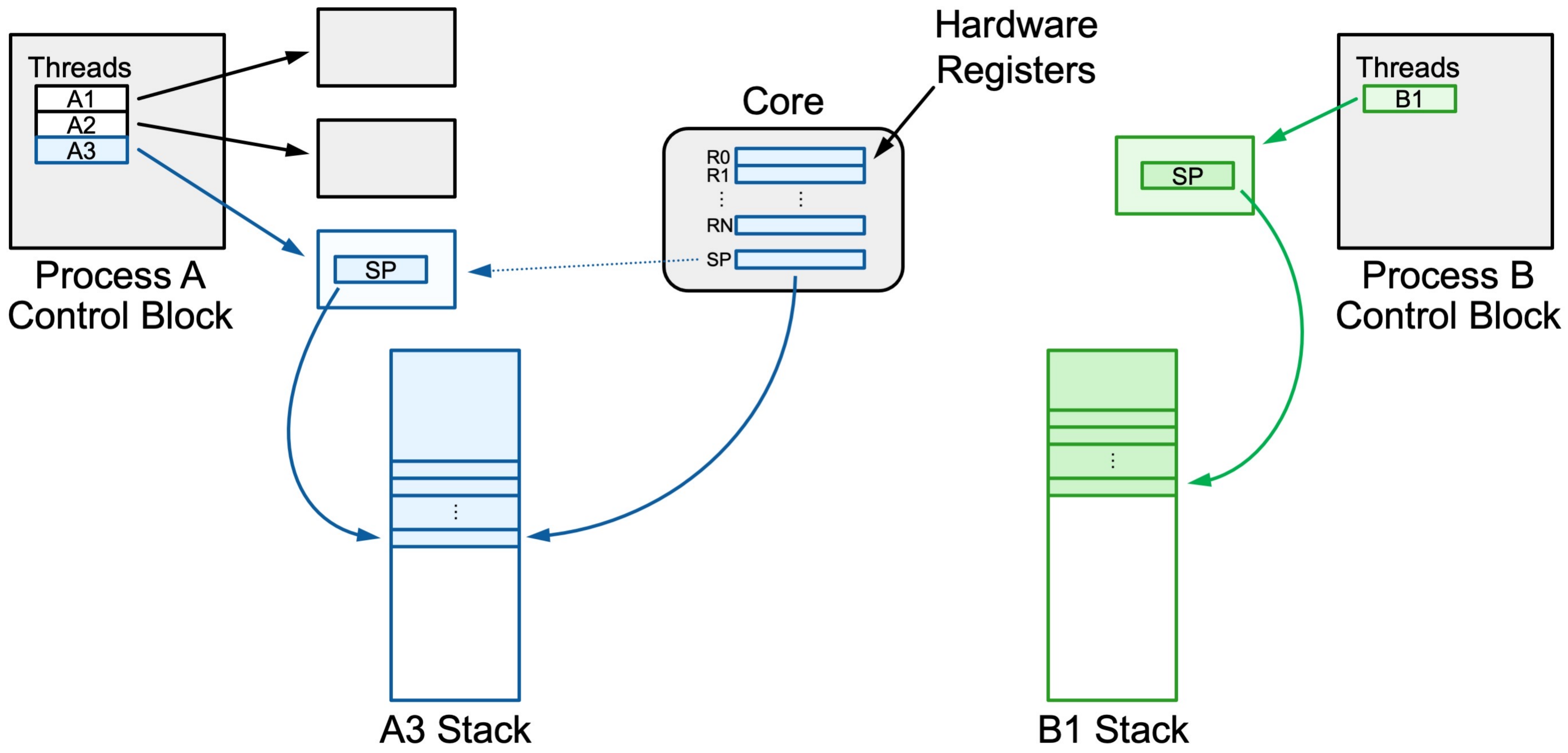
Context Switching

***Step 1:** push all registers besides stack register onto the thread's stack.*



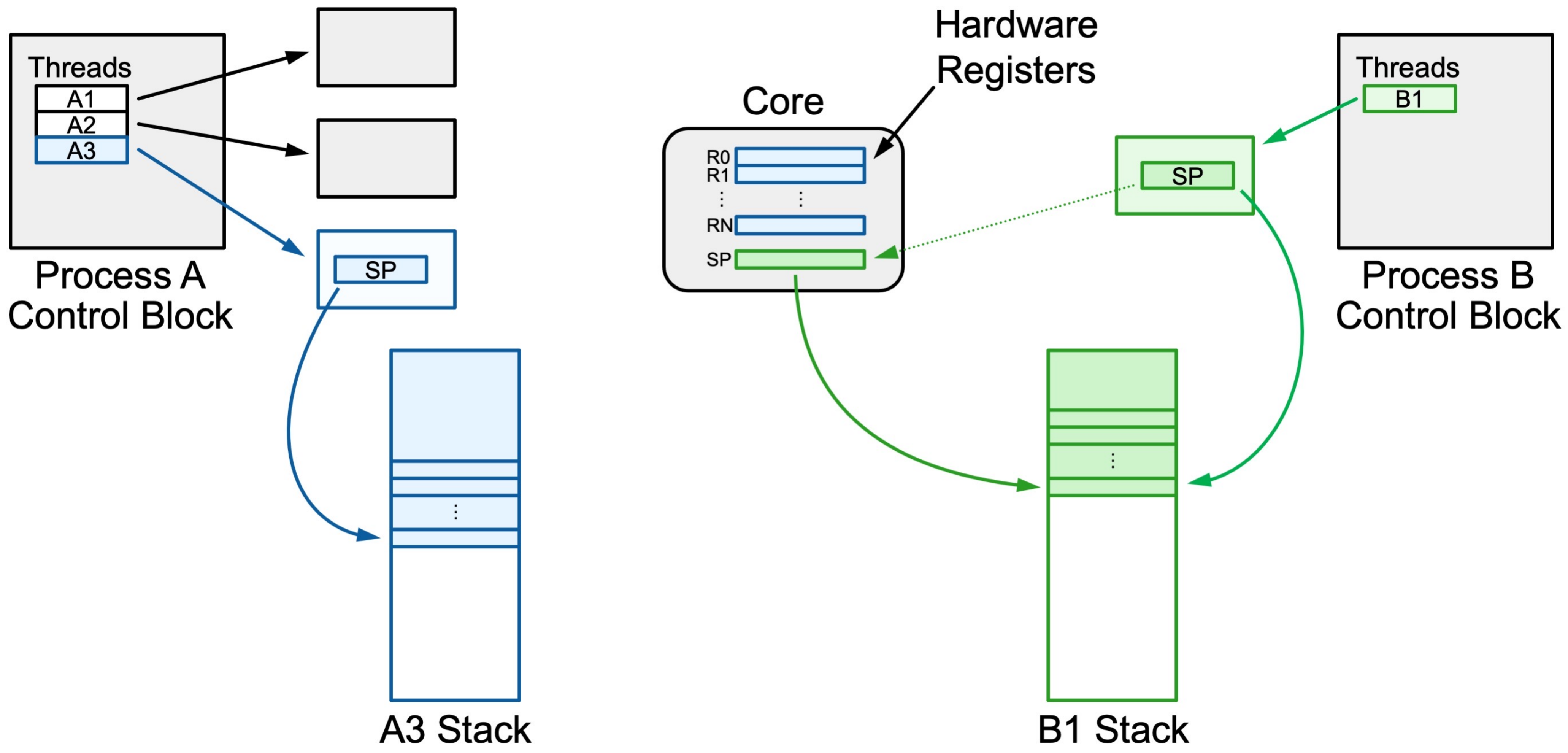
Context Switching

Step 2: *save the stack register into the thread's state space.*



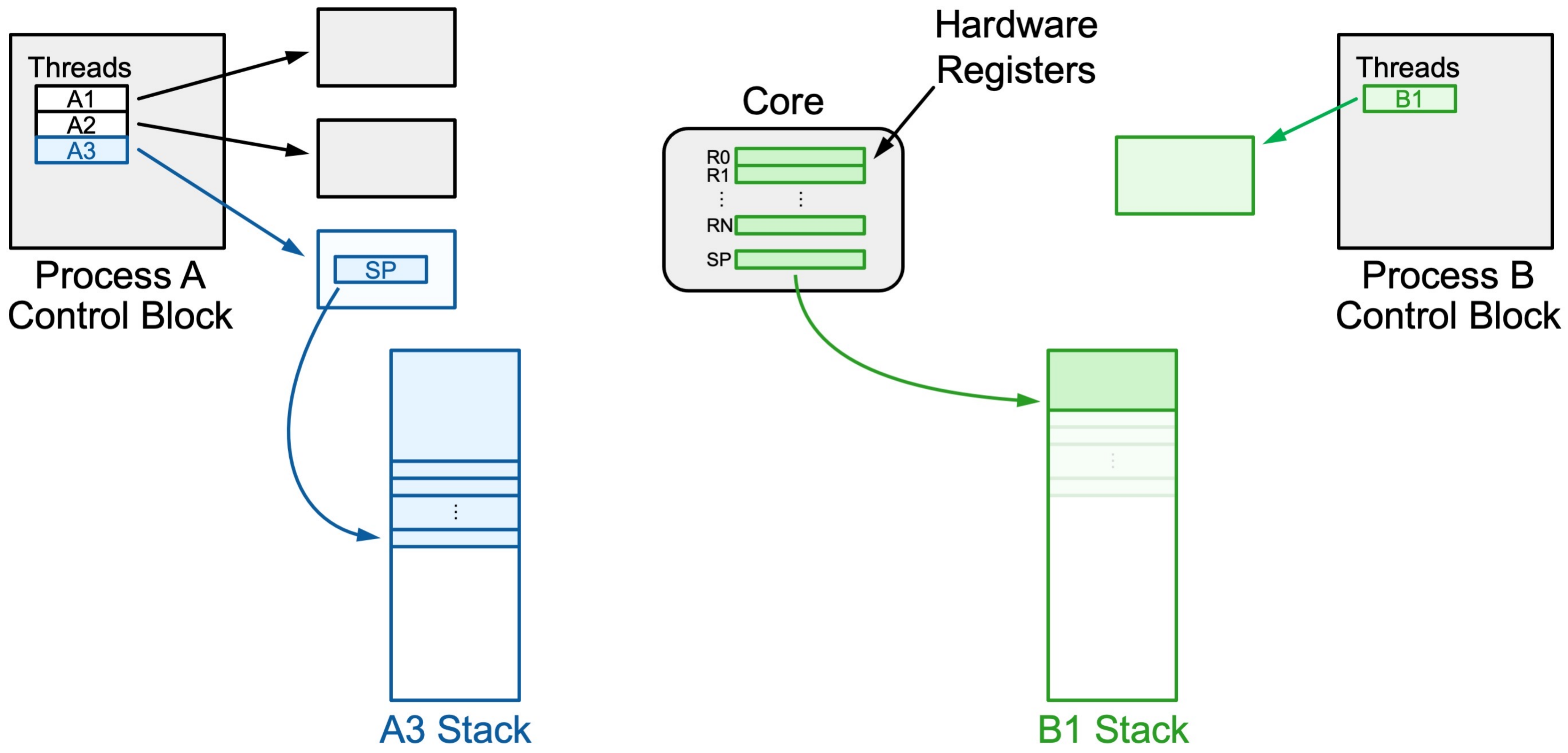
Context Switching

Step 3: *load B1's saved stack register from its thread state space.*



Context Switching

***Step 4:** pop B1's other registers from its stack space.*



Context Switching

A *context switch* means changing the thread currently running to another thread. We must save the current thread state and load in the new thread state.

1. Push all registers besides stack onto current thread's stack
2. Save the current stack register (rsp) into the thread's state space
3. Load the other thread's saved stack register from its state space into rsp
4. Pop registers off the other thread's stack

Super funky: we are calling a function from one thread's stack and execution and returning from it in **another** thread's stack and execution!

Context Switching

```
pushq %rbp
pushq %rbx
pushq %r12
pushq %r13
pushq %r14
pushq %r15
movq %rsp,0x2000(%rdi)
movq 0x2000(%rsi),%rsp
popq %r15
popq %r14
popq %r13
popq %r12
popq %rbx
popq %rbp
ret
```

Context Switching

```
pushq %rbp
pushq %rbx
pushq %r12
pushq %r13
pushq %r14
pushq %r15
movq %rsp,0x2000(%rdi)
movq 0x2000(%rsi),%rsp
popq %r15
popq %r14
popq %r13
popq %r12
popq %rbx
popq %rbp
ret
```

1. Push all registers besides stack onto current thread's stack

Context Switching

```
pushq %rbp
pushq %rbx
pushq %r12
pushq %r13
pushq %r14
pushq %r15
movq %rsp,0x2000(%rdi)
movq 0x2000(%rsi),%rsp
popq %r15
popq %r14
popq %r13
popq %r12
popq %rbx
popq %rbp
ret
```

2. Save the current stack register (rsp) into the thread's state space

Context Switching

```
pushq %rbp
pushq %rbx
pushq %r12
pushq %r13
pushq %r14
pushq %r15
movq %rsp,0x2000(%rdi)
movq 0x2000(%rsi),%rsp
popq %r15
popq %r14
popq %r13
popq %r12
popq %rbx
popq %rbp
ret
```

3. Load the other thread's saved stack register from its state space into rsp

Context Switching

```
pushq %rbp
pushq %rbx
pushq %r12
pushq %r13
pushq %r14
pushq %r15
movq %rsp,0x2000(%rdi)
movq 0x2000(%rsi),%rsp
popq %r15
popq %r14
popq %r13
popq %r12
popq %rbx
popq %rbp
ret
```

4. Pop registers off the other thread’s stack

Context Switching

```
pushq %rbp
pushq %rbx
pushq %r12
pushq %r13
pushq %r14
pushq %r15
movq %rsp,0x2000(%rdi)
movq 0x2000(%rsi),%rsp
popq %r15
popq %r14
popq %r13
popq %r12
popq %rbx
popq %rbp
ret
```

Now we return back to the function in the **new thread** that called **context_switch** previously! (recall: **ret** pops the address off the stack for the instruction we should resume at in the caller)

Context Switching

```
pushq %rbp
pushq %rbx
pushq %r12
pushq %r13
pushq %r14
pushq %r15
movq %rsp,0x2000(%rdi)
movq 0x2000(%rsi),%rsp
popq %r15
popq %r14
popq %r13
popq %r12
popq %rbx
popq %rbp
ret
```

Context Switching

```
pushq %rbp
pushq %rbx
pushq %r12
pushq %r13
pushq %r14
pushq %r15
movq %rsp,0x2000(%rdi)
movq 0x2000(%rsi),%rsp
```

we start executing on one stack…

```
popq %r15
popq %r14
popq %r13
popq %r12
popq %rbx
popq %rbp
ret
```

and end executing on another!

Context Switching

```
pushq %rbp
pushq %rbx
pushq %r12
pushq %r13
pushq %r14
pushq %r15
movq %rsp,0x2000(%rdi)
movq 0x2000(%rsi),%rsp
popq %r15
popq %r14
popq %r13
popq %r12
popq %rbx
popq %rbp
ret
```

We enter via a call from a function in the current thread

We exit to a call from a function in the new thread!

Switching Between Threads

When the OS regains control, how does it switch to run another thread?

- **Key idea:** we must load the thread's state onto a processor core and run it
- State = **registers**
  - Registers store data being manipulated by the core
  - %rsp stores current top of stack
  - Stack remembers what function to continue executing
- If we can load this thread's %rsp + other saved registers, then it can resume right where it left off
- *Context switch* – changing the thread currently running to another thread. We must save the current thread state and load in the new thread state.

Recap

- **Overview:** Scheduling and Dispatching
- Process and Thread State
- Running a Thread
- Switching Between Threads

Lecture 17 takeaway: The OS keeps a process control block for each process and uses it to context switch between threads. To switch we must freeze frame the existing register values and load in new ones.

Next time: how do we decide which thread to run?