Virtual Memory I

CSE 351 Summer 2021

Instructor:

Mara Kirdani-Ryan

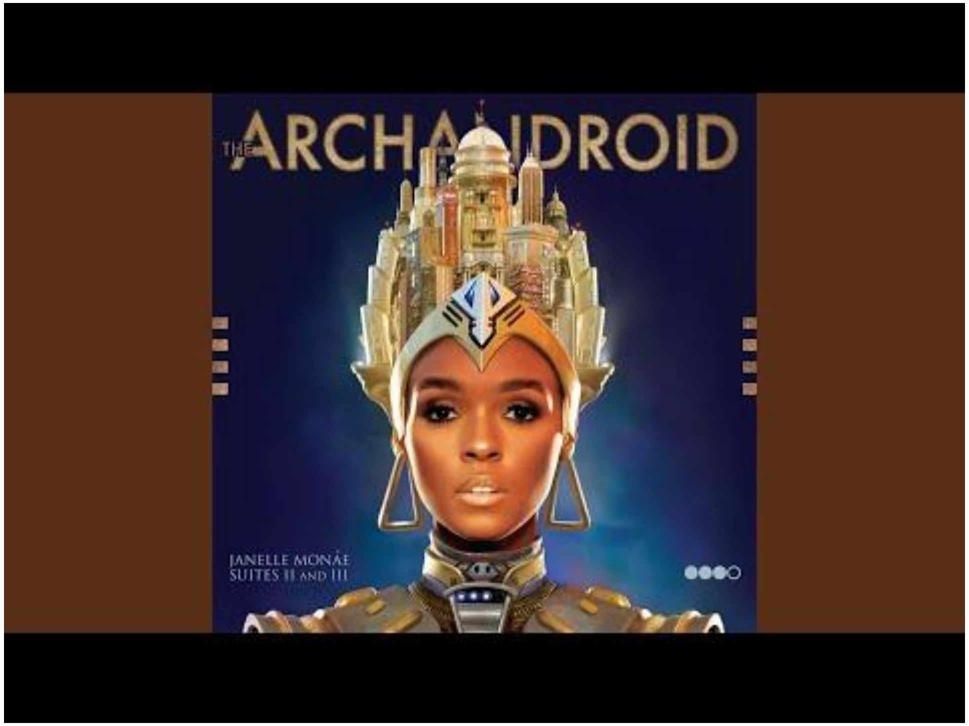
Teaching Assistants:

Kashish Aggarwal

Nick Durand

Colton Jobes

Tim Mandzyuk



Gentle, Loving Reminders

- hw17 due tonight!
- hw18, 19 due Wednesday (8/11)
  - Lots of virtual memory

- Lab 4 due Monday (8/9)
  - All about caches!

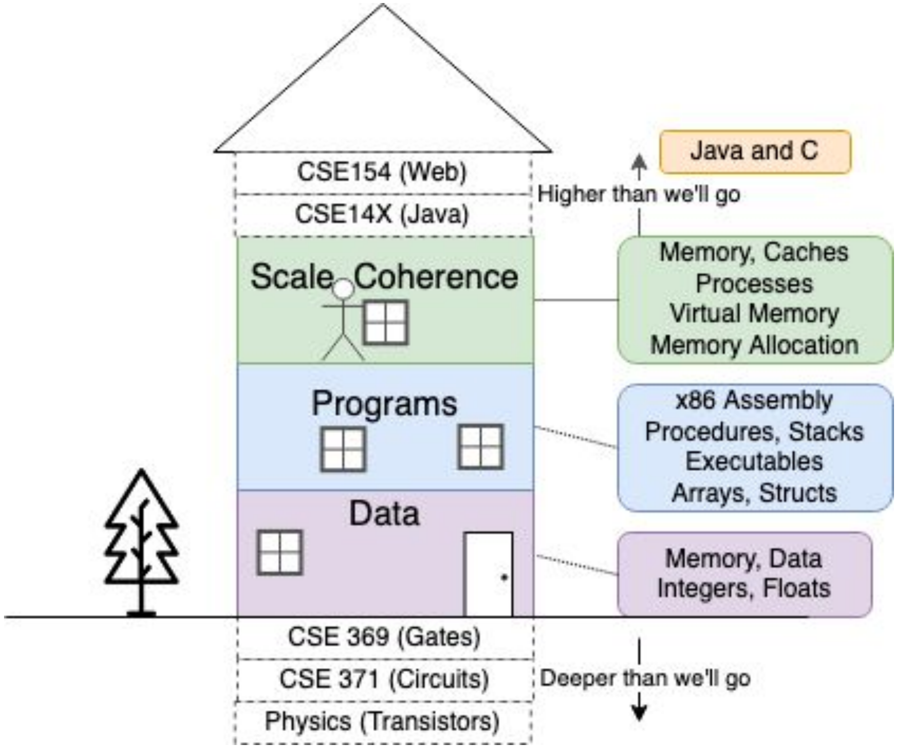
Learning Objectives

Understanding this lecture means you can:

- Explain the purpose of virtual memory, and how indirection is used to achieve abstraction
- Problematize visions of computing grandeur, especially around “good intentions”

Unit 3: Scale, Coherence

- Caches, Process, **Virtual Memory**
  - Multiple programs? Larger programs?
- Metrics & Structures
- Scale, Automation



Virtual Memory (VM*)

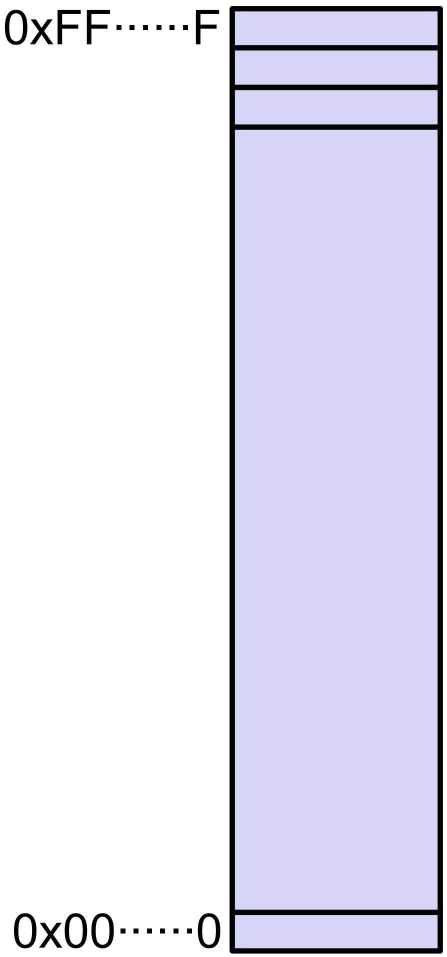
- **Overview and motivation**
- **VM as a tool for caching**
- Address translation
- VM as a tool for memory management
- VM as a tool for memory protection

Warning: Virtual memory is pretty complex, but crucial for understanding how processes work and for debugging performance

**Not to be confused with "Virtual Machine" which is a whole other thing.*

Memory is *virtual!*

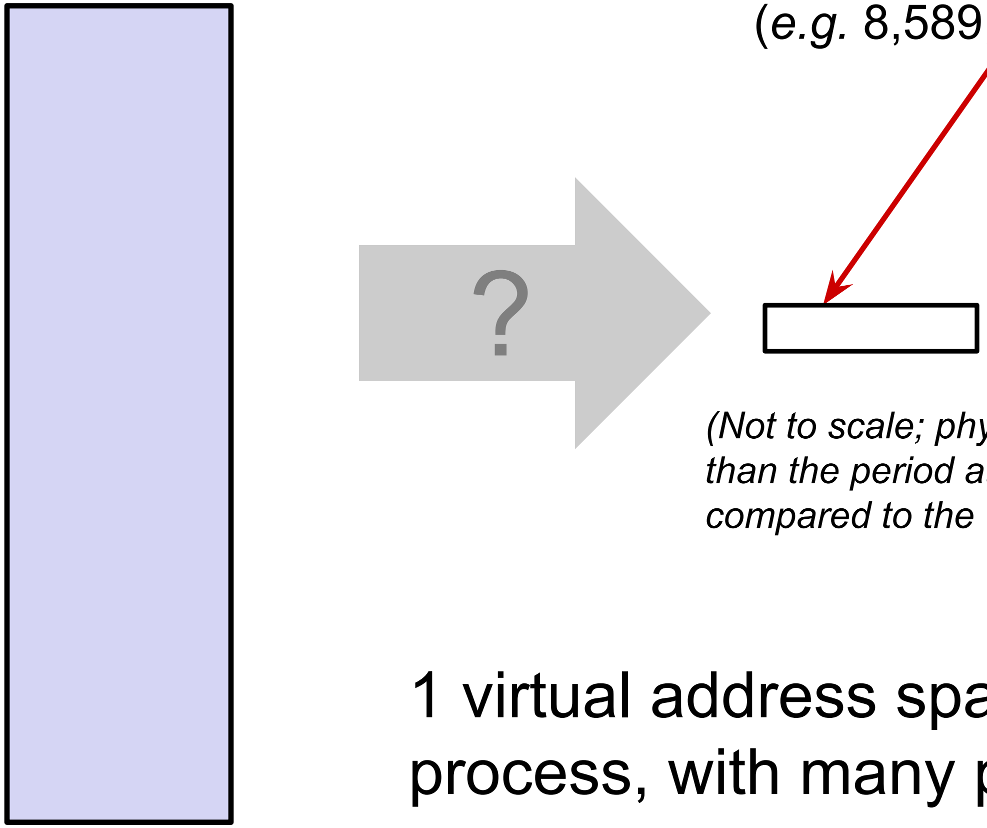
- Programs refer to virtual memory addresses
  - `movq (%rdi),%rax`
  - Conceptually memory is just a very large array of bytes
  - System provides private address space to each process
- Allocation: Compiler and run-time system
  - Where different program objects should be stored
  - All allocation within single virtual address space
- But…
  - We *probably* don't have 2^w bytes of physical memory
  - We *certainly* don't have 2^w bytes of physical memory ***for every process***
  - Processes should not interfere with one another



Problem 1: How Does Everything Fit?

64-bit virtual addresses can address several exabytes (18,446,744,073,709,551,616 bytes)

Physical main memory offers a few gigabytes (*e.g.* 8,589,934,592 bytes)



(Not to scale; physical memory would be smaller than the period at the end of this sentence compared to the virtual address space.)

1 virtual address space per process, with many processes…

Problem 2: Memory Management

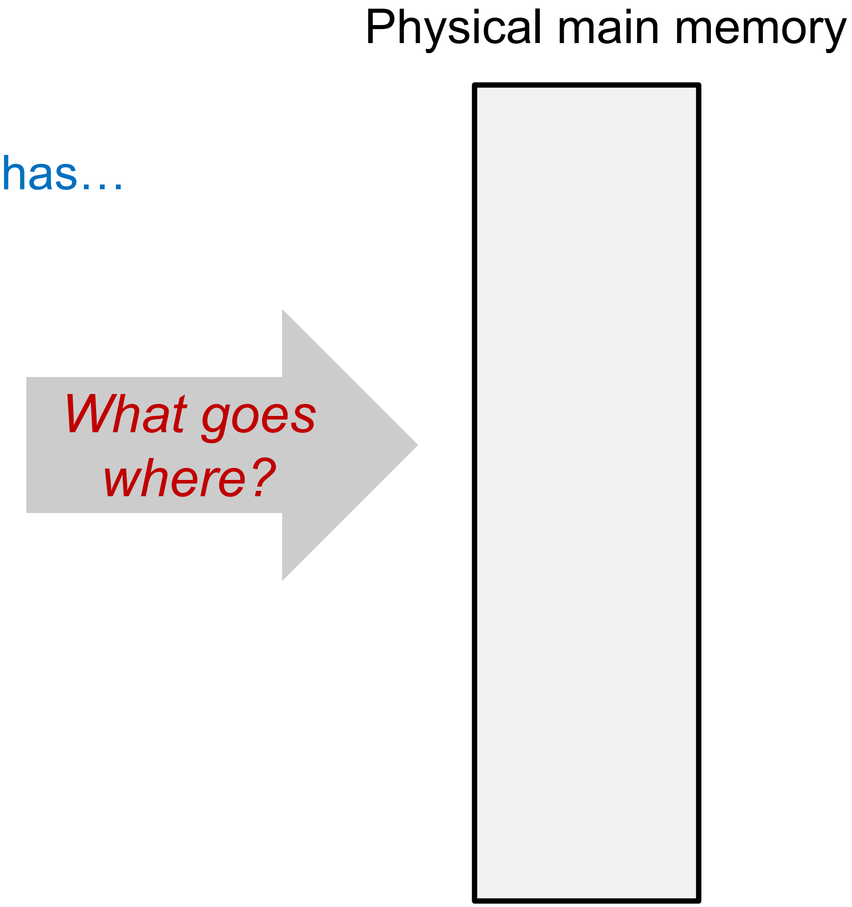
We have multiple processes:

Process 1
Process 2
Process 3
…
Process n

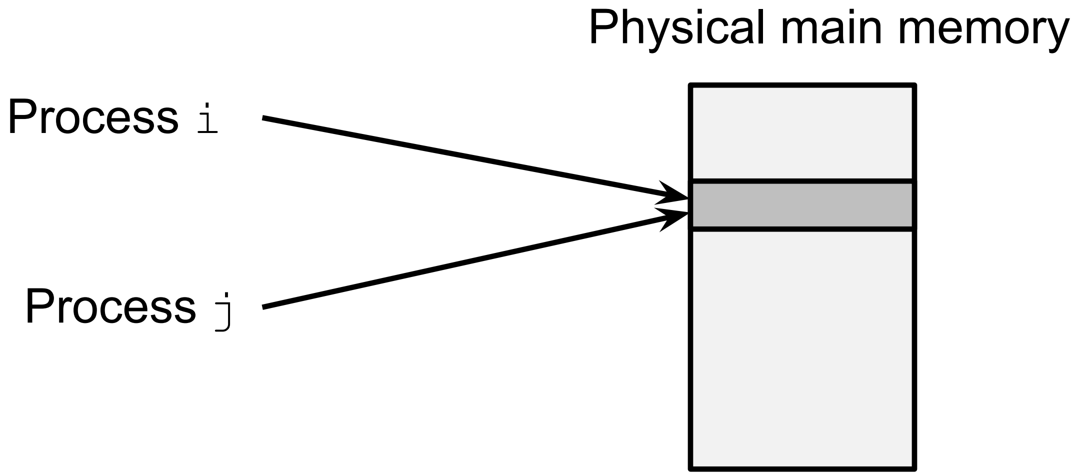
X

Each process has…

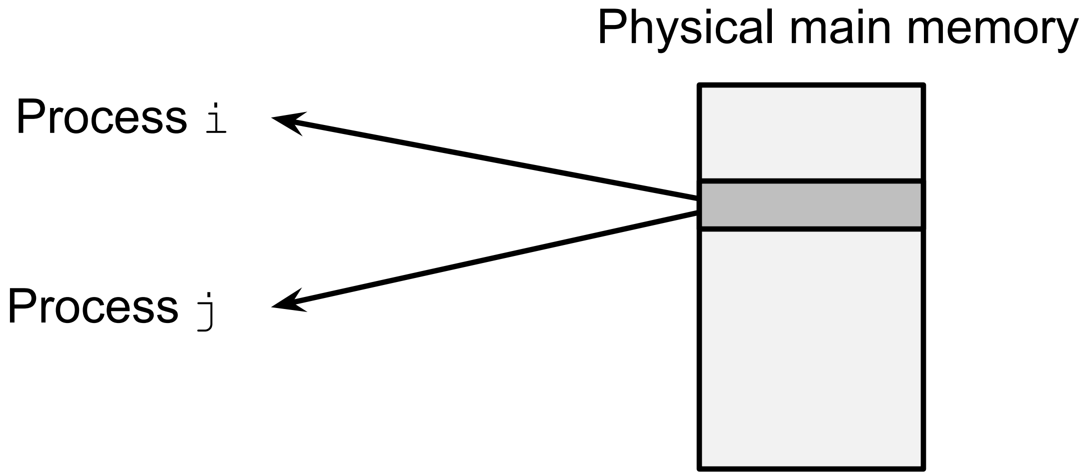
stack
heap
`.text`
`.data`
…



Problem 3: How To Protect



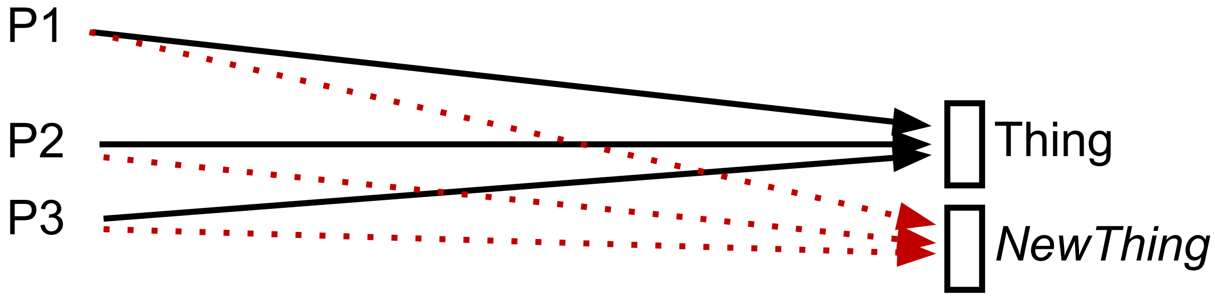
Problem 4: How To Share?



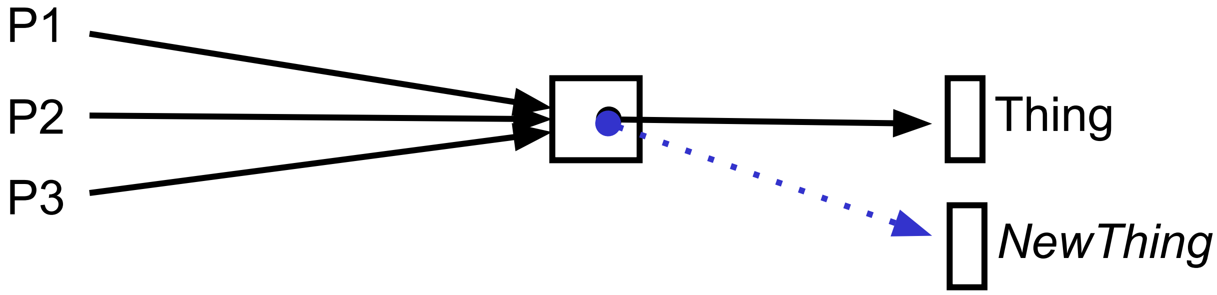
How can we solve these problems?

- "Any problem in computer science can be solved by adding another level of **indirection**."
  – *David Wheeler, inventor of the subroutine*

- Without Indirection



- With Indirection



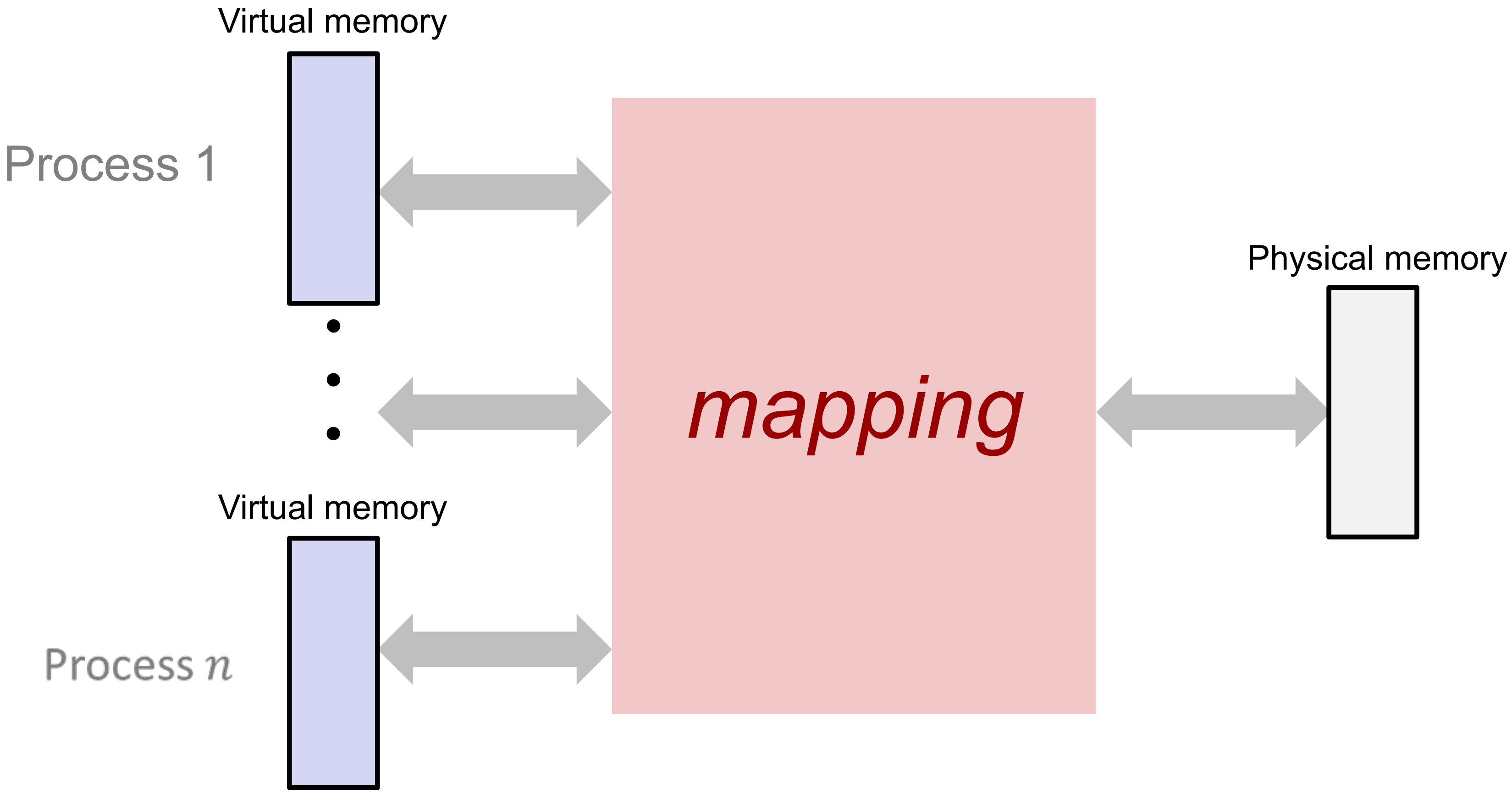
What if I want to move Thing?

Indirection

- *Indirection*: The ability to reference something using a name, reference, or container instead of the value itself. A flexible mapping between a name and a thing allows changing the thing without notifying holders of the name.
  - Adds some work (now have to look up 2 things instead of 1)
  - But don't have to track all uses of name/address (single source!)

- Examples:
  - **Phone system:** cell phone number portability
  - **Domain Name Service (DNS):** translation from name to IP address
  - **Call centers:** route calls to available operators, etc.
  - **Dynamic Host Configuration Protocol (DHCP):** local network address assignment

Indirection in Virtual Memory



- Each process gets a private virtual address space
- Solves the previous problems!

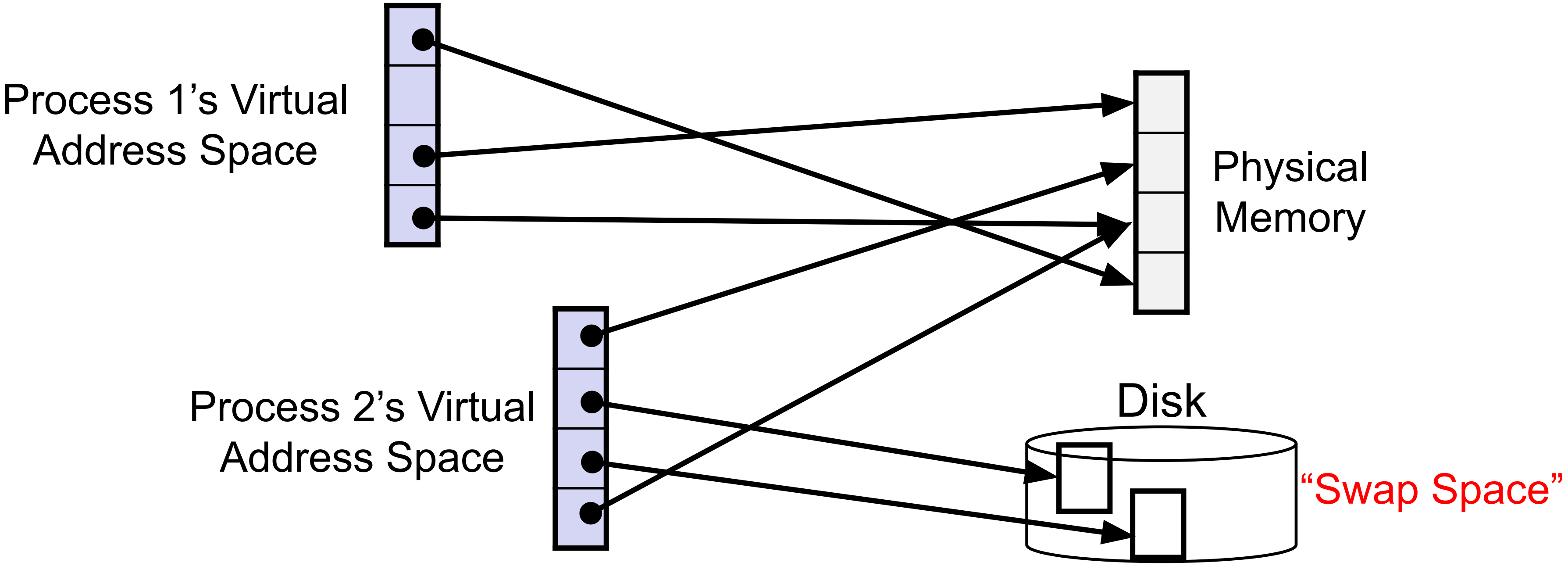
Address Spaces

- Virtual address space: Set of $\mathrm{N} = 2^n$ virtual addr
  - {0, 1, 2, 3, …, N-1}
- Physical address space: Set of $\mathrm{M} = 2^m$ physical addr
  - {0, 1, 2, 3, …, M-1}

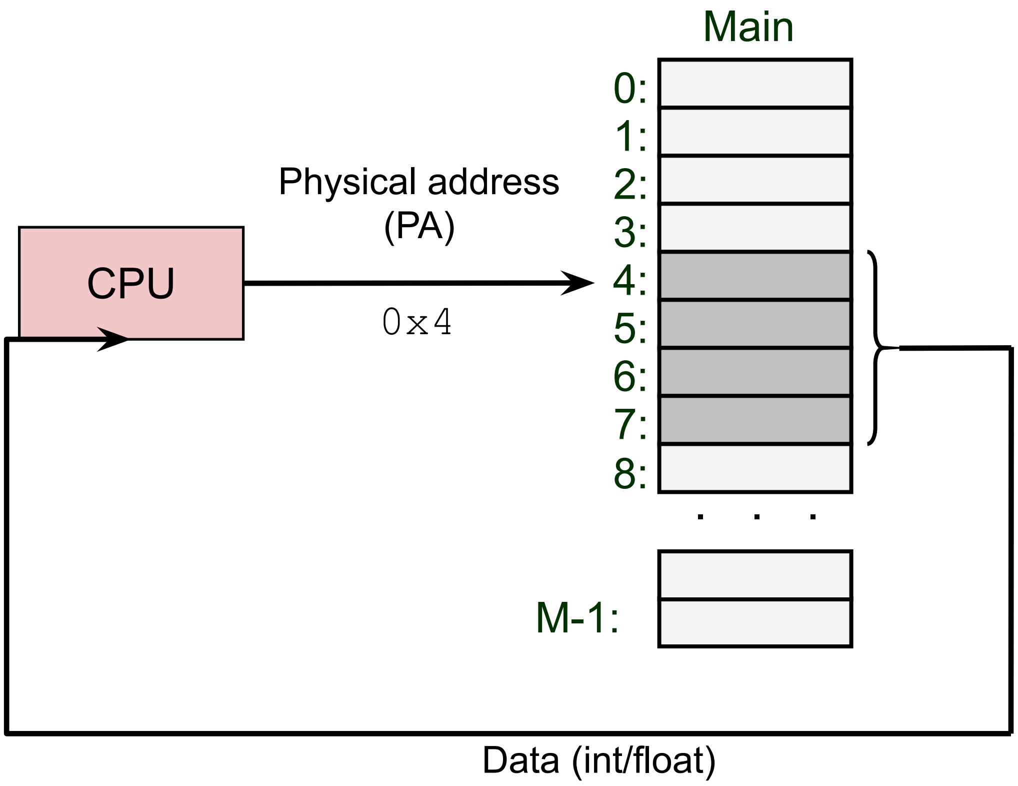
- Every byte in main memory has:
  - one physical address (PA)
  - zero, one, *or more* virtual addresses (VAs)

Mapping

- A virtual address (VA) can be mapped to either physical memory or disk
  - Unused VAs may not have a mapping
  - VAs from *different* processes may map to same location in memory/disk

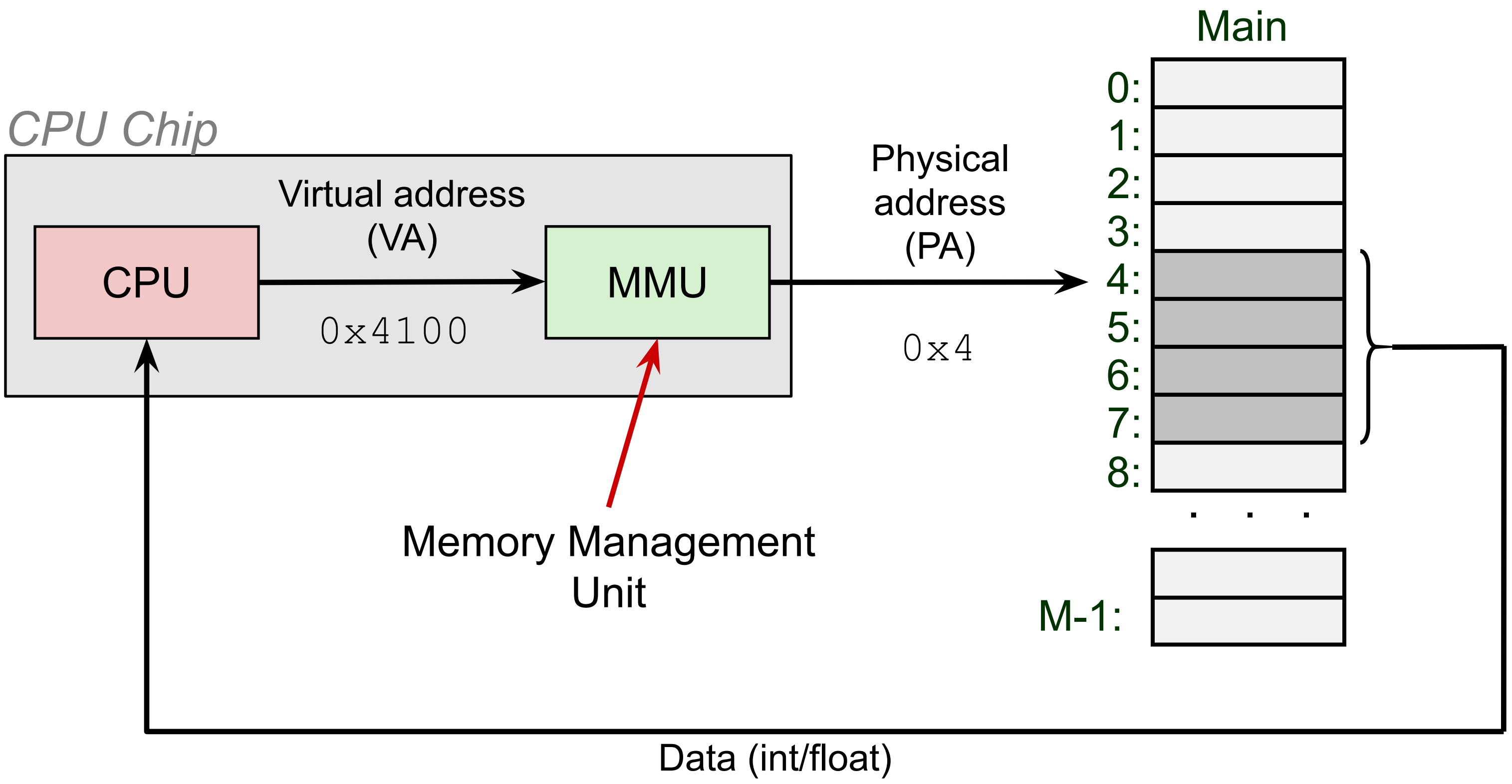

Process 1's Virtual Address Space
Physical Memory
Process 2's Virtual Address Space
Disk
"Swap Space"

A System Using Physical Addressing



- Used in “simple” systems with (usually) just one process:
  - Embedded microcontrollers in devices like cars, elevators, and digital picture frames

A System Using Virtual Addressing



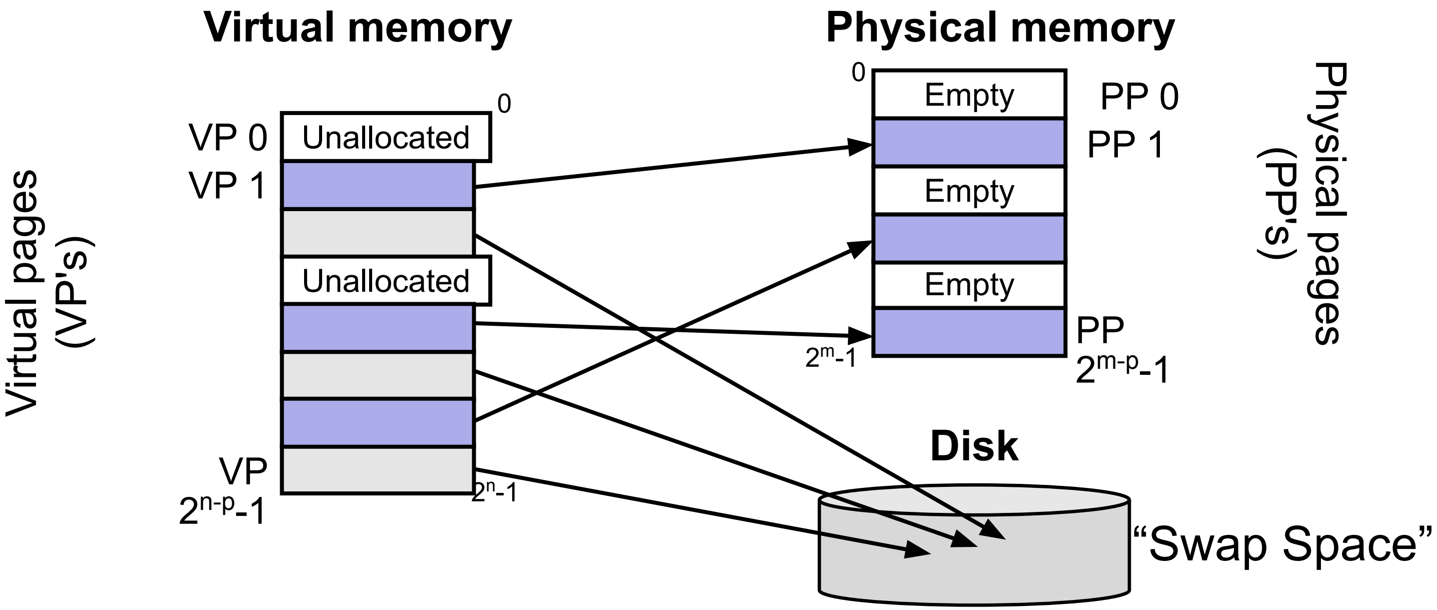
- Physical addresses are *invisible to programs*
  - Used in all modern desktops, laptops, smartphones…
  - “Classic” CS idea, made visible

Why Virtual Memory (VM)?

- Efficient use of limited main memory (RAM)
  - Use RAM as a cache for the parts of a virtual address space
    - Some non-cached parts stored on disk
    - Some (unallocated) non-cached parts stored nowhere
  - Keep only active areas of virtual address space in memory
    - Transfer data back and forth as needed
- Simplifies memory management for programmers
  - Each process "gets" the same full, private linear address space
- Isolates address spaces (protection)
  - One process can't interfere with another's memory
    - They operate in *different address spaces*
  - User process cannot access privileged information
    - Different sections of address spaces have different permissions

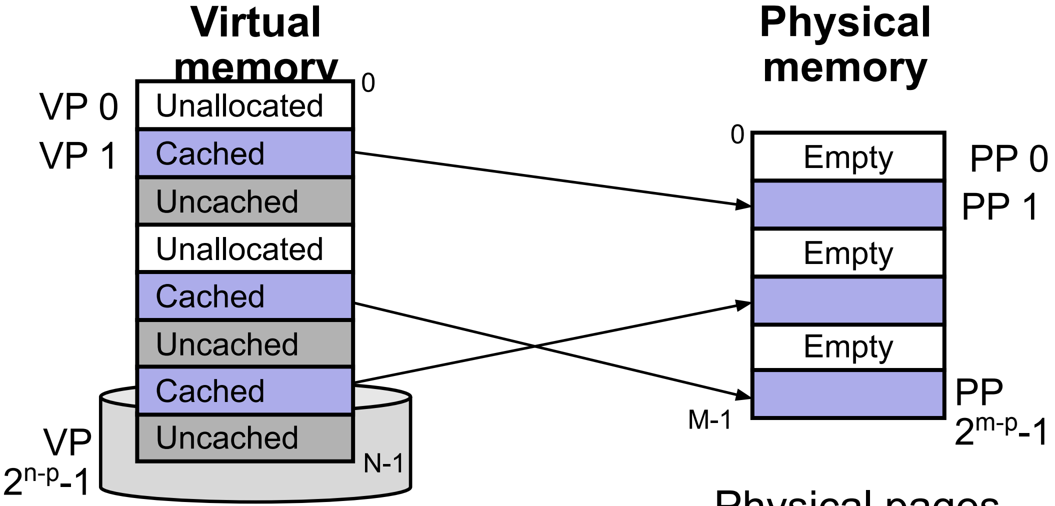
VM and the Memory Hierarchy

- Think of virtual memory as array of $\mathrm{N} = 2^n$ contiguous bytes
- *Pages* of virtual memory are usually stored in physical memory, but sometimes spill to disk
  - Pages are another unit of aligned memory (size is $\mathrm{P} = 2^p$ bytes)
  - Each virtual page can be stored in *any* physical page (no fragmentation!)



or: VM as DRAM Cache for Disk

- Think of virtual memory as an array of $\mathrm{N} = 2^n$ contiguous bytes stored *on a disk*
- Then physical main memory is used as a *cache* for the virtual memory array
  - These “cache blocks” are called *pages* (size is $\mathrm{P} = 2^p$ bytes)

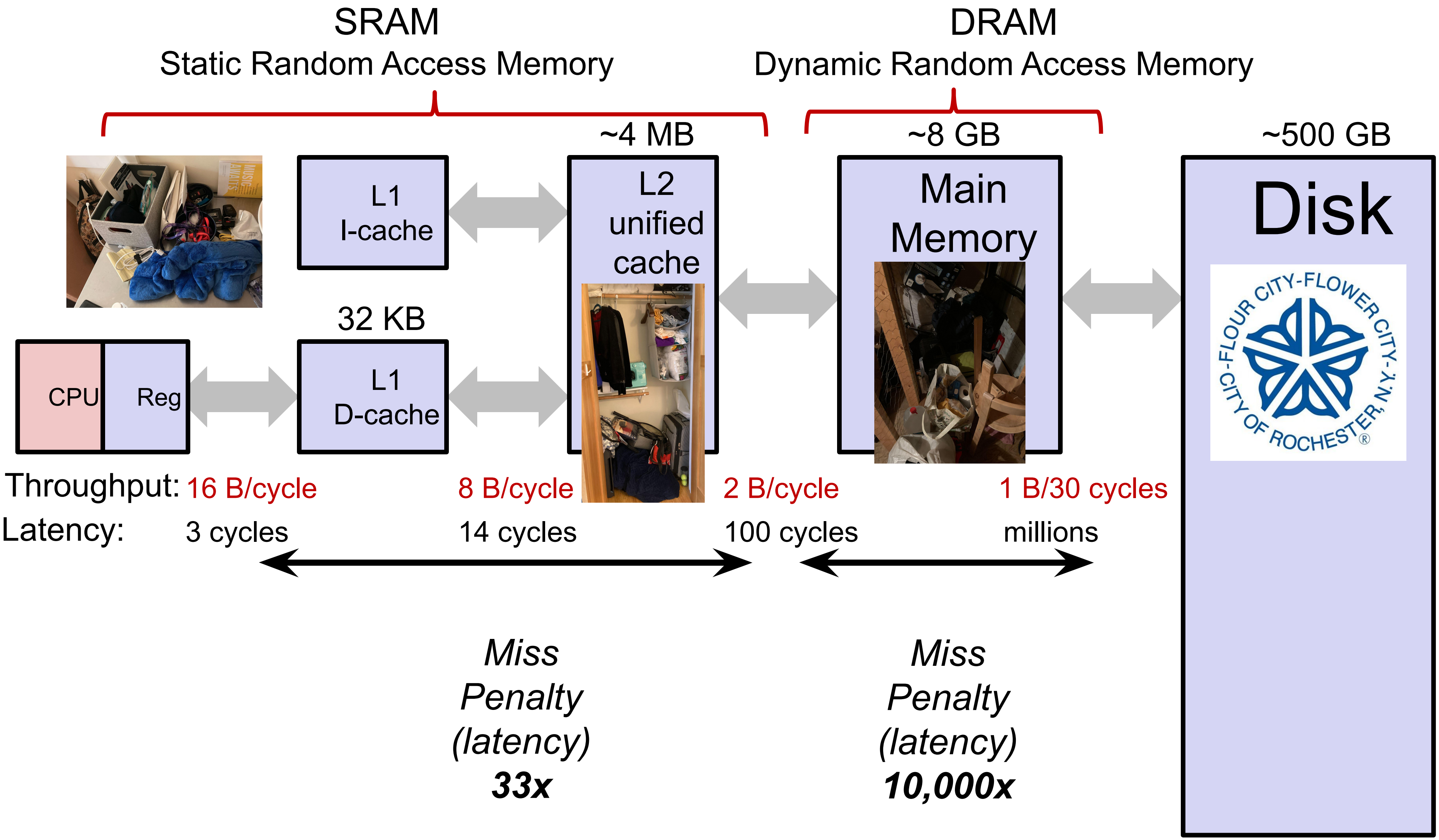


Virtual pages (VPs) “stored on disk”

Physical pages (PPs) cached in DRAM

Memory Hierarchy: Core 2 Duo

Not drawn to scale



SRAM
Static Random Access Memory

DRAM
Dynamic Random Access Memory

CPU Reg

32 KB

L1 I-cache

L1 D-cache

~4 MB

L2 unified cache

~8 GB

Main Memory

~500 GB

Disk

Throughput: 16 B/cycle 8 B/cycle 2 B/cycle 1 B/30 cycles

Latency: 3 cycles 14 cycles 100 cycles millions

Miss Penalty (latency) ***33x***

Miss Penalty (latency) ***10,000x***

Virtual Memory Design Consequences

- Large page size: typically 4-8 KiB or 2-4 MiB
  - *Can* be up to 1 GiB (for "Big Data" apps on big computers)
  - Compared with 64-byte cache blocks
- Fully associative
  - Any virtual page can be placed in any physical page
  - Requires a "large" mapping function – different from CPU caches
- Fancy, expensive replacement algorithms in OS
  - Too complicated and open-ended to be implemented in hardware
- *Write-back* rather than *write-through*
  - *Really* don't want to write to disk every time we write to memory
  - Some things may never end up on disk (*e.g.* stack for short-lived process)

Why does VM work on RAM/disk?

- Avoids disk accesses because of *locality*
  - Same reason that L1 / L2 / L3 caches work

- Set of virtual pages that a program is "actively" accessing at any point is called its *working set*
  - If (*working set of one process ≤ physical memory*):
    - Good performance for one process (after compulsory misses)
  - If (*working sets of all processes > physical memory*):
    - ***Thrashing:*** Performance meltdown where pages are swapped between memory and disk continuously (CPU always waiting or paging)
    - Why adding RAM speeds up computer performance

Summary

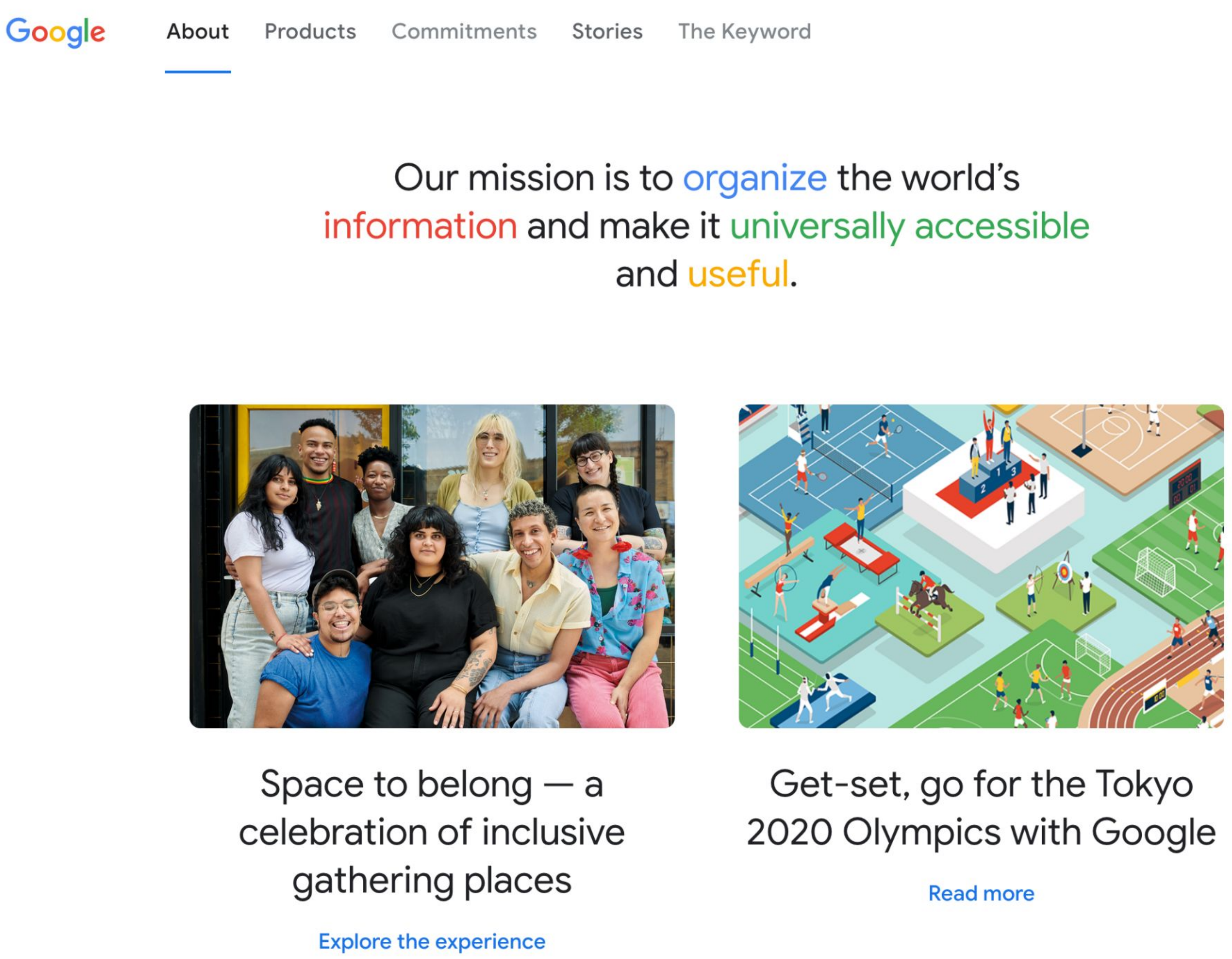
- Virtual memory provides:
  - Ability to use limited memory (RAM) across multiple processes
  - Illusion of contiguous virtual address space for each process
  - Protection and sharing amongst processes

Computing and Vision

What's your vision for computing?
Is there a collective vision?

"To provide free and easy access to a vast array of knowledge, ideas, and information by supporting lifelong learning and a love of reading, so that everyone in our community is empowered, informed, and enriched."

Seattle Public Library Mission, 2002



Our mission is to organize the world's information and make it universally accessible and useful.

Space to belong — a celebration of inclusive gathering places

Explore the experience

Get-set, go for the Tokyo 2020 Olympics with Google

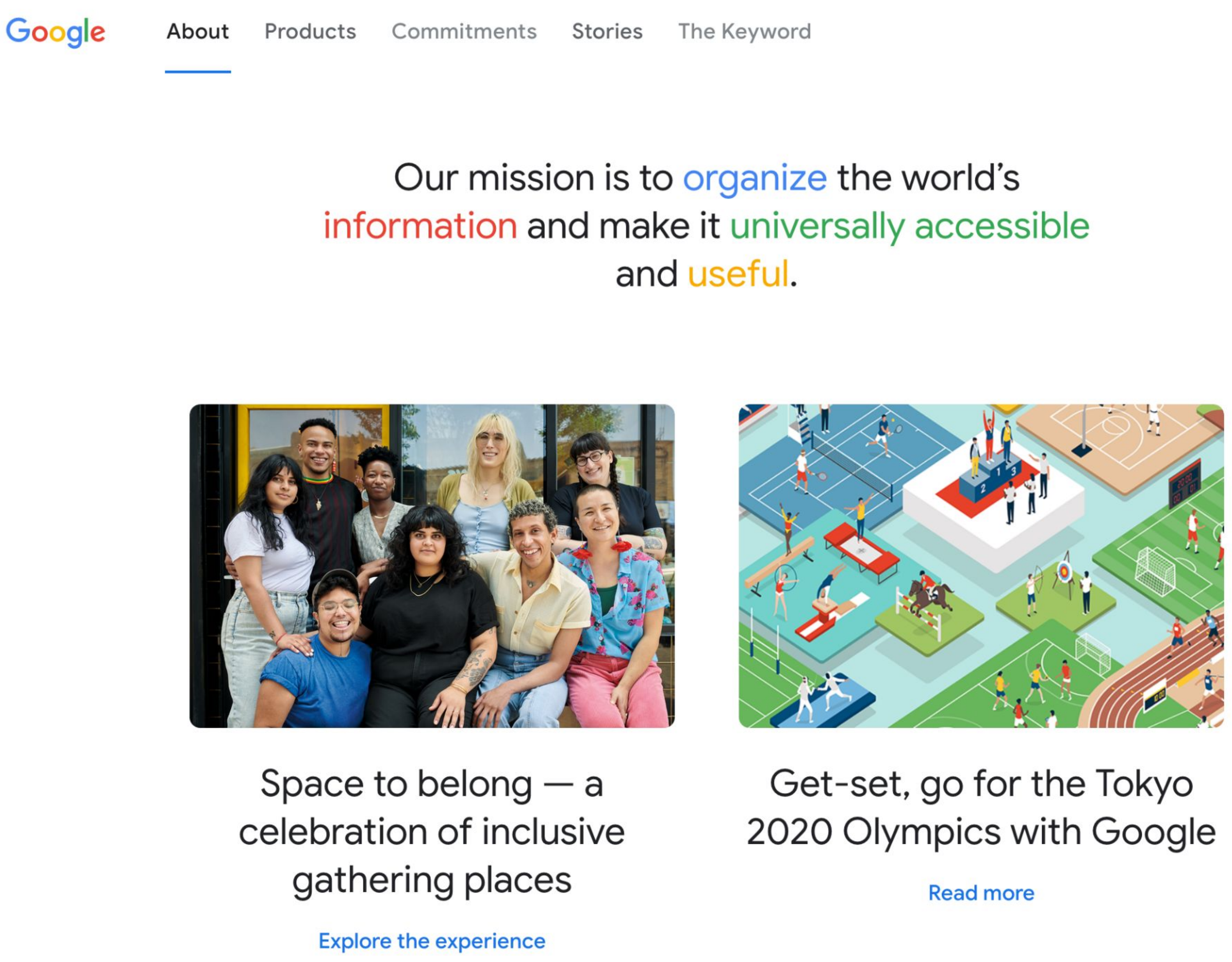
Read more

The difference is scale!

Seattle, versus the world.

What’s the ideological vision of computing?

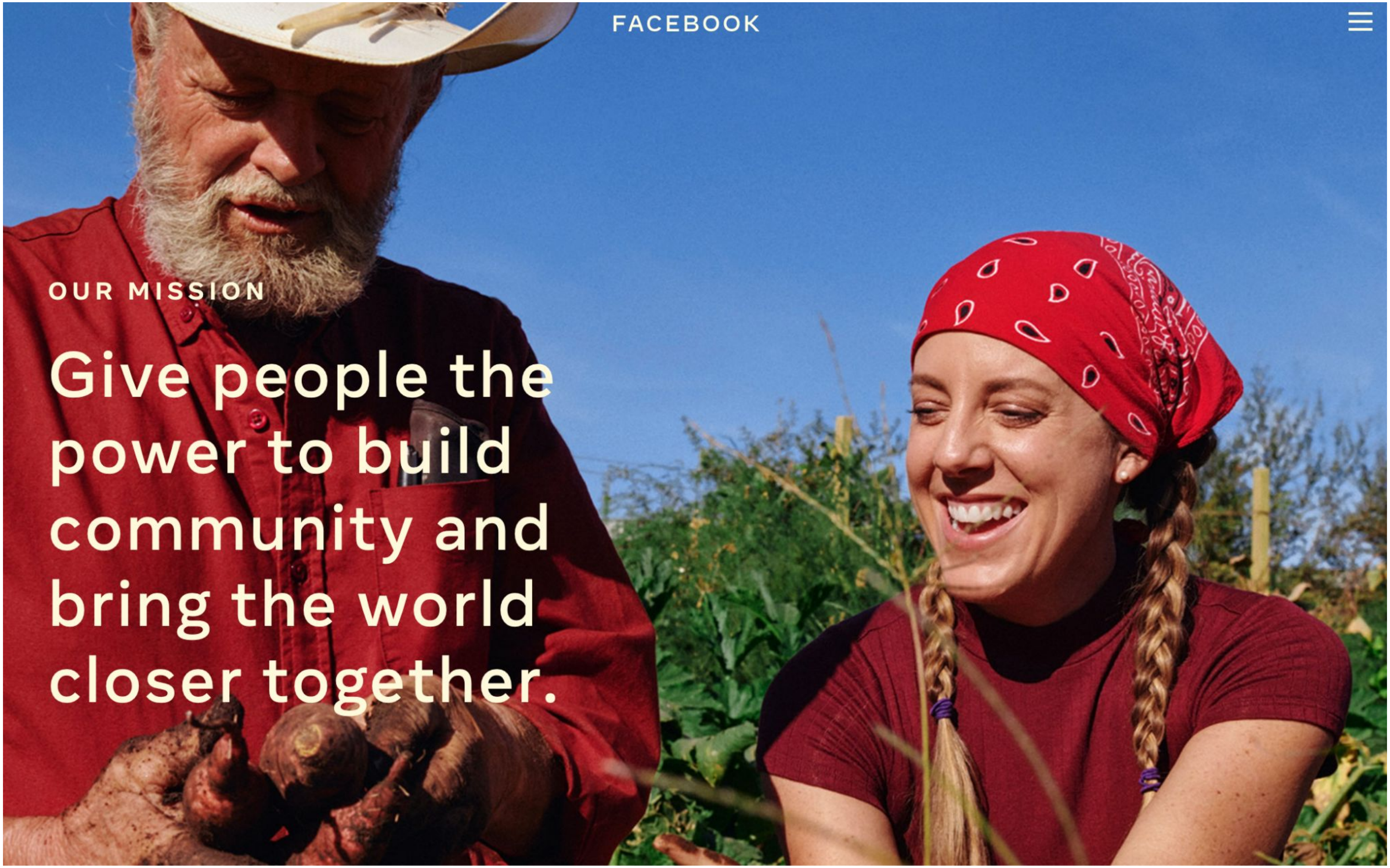
Ideology: What’s so true, that you don’t even need to ask?



Our mission is to organize the world's information and make it universally accessible and useful.

Space to belong — a celebration of inclusive gathering places

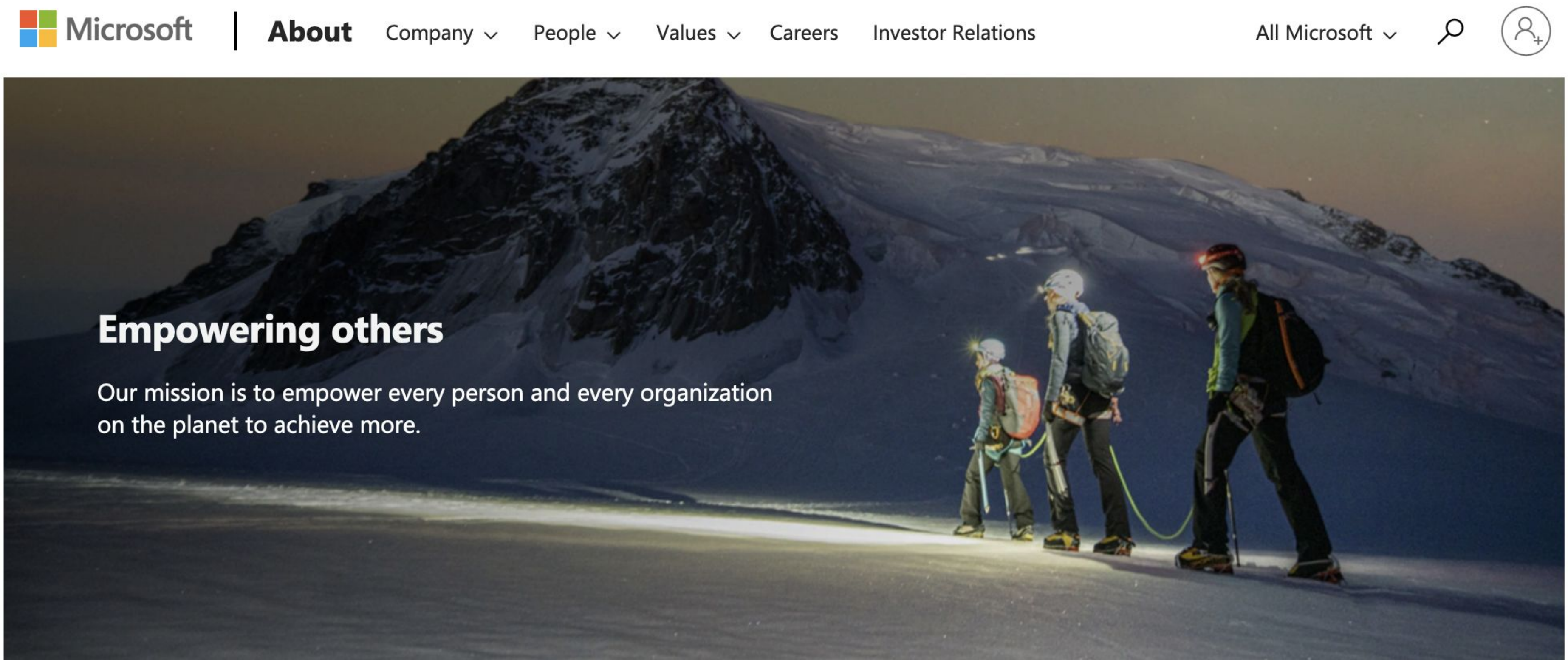
Get-set, go for the Tokyo 2020 Olympics with Google



OUR MISSION

Give people the power to build community and bring the world closer together.

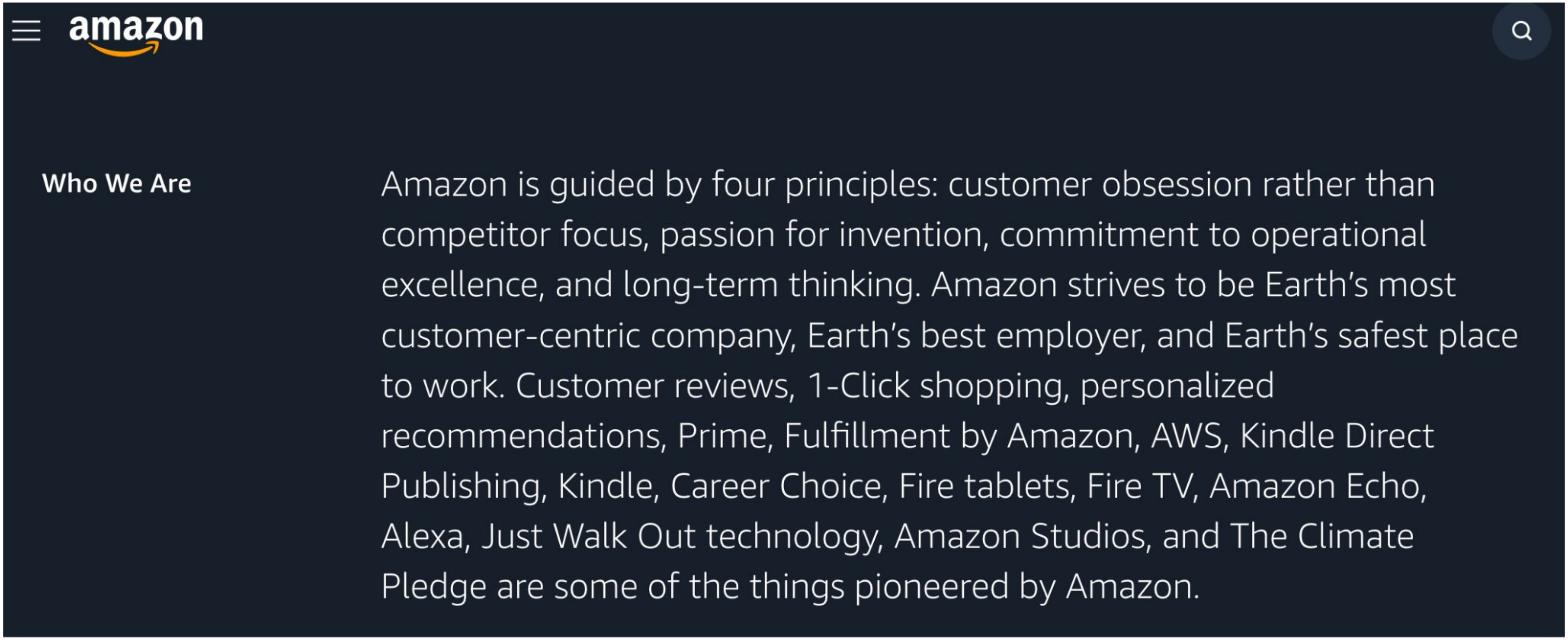
Quite literally, not a computer in sight



Empowering others

Our mission is to empower every person and every organization on the planet to achieve more.

Quite literally, not a computer in sight



Who We Are

Amazon is guided by four principles: customer obsession rather than competitor focus, passion for invention, commitment to operational excellence, and long-term thinking. Amazon strives to be Earth's most customer-centric company, Earth's best employer, and Earth's safest place to work. Customer reviews, 1-Click shopping, personalized recommendations, Prime, Fulfillment by Amazon, AWS, Kindle Direct Publishing, Kindle, Career Choice, Fire tablets, Fire TV, Amazon Echo, Alexa, Just Walk Out technology, Amazon Studios, and The Climate Pledge are some of the things pioneered by Amazon.

Again, "Earth's…"

Vision: Operating on a global & universal scale

What might this mean?

"In my very long term worldview, our software understands deeply what you're knowledgeable about, what you're not, and how to organize the world so that the world can solve important problems"

Larry Page, Google Founder, 2013

"Our greatest opportunities are now global — like spreading prosperity and freedom, promoting peace and understanding, lifting people out of poverty, and accelerating science. Our greatest challenges also need global responses — like ending terrorism, fighting climate change, and preventing pandemics. Progress now requires humanity coming together not just as cities or nations, but also as a global community….in times like these, the most important thing we at Facebook can do is develop the social infrastructure to give people the power to build a global community that works for all of us."

Mark Zuckerberg, 2017

Vision:
Operating on a global & universal scale, with Big Tech at the helm

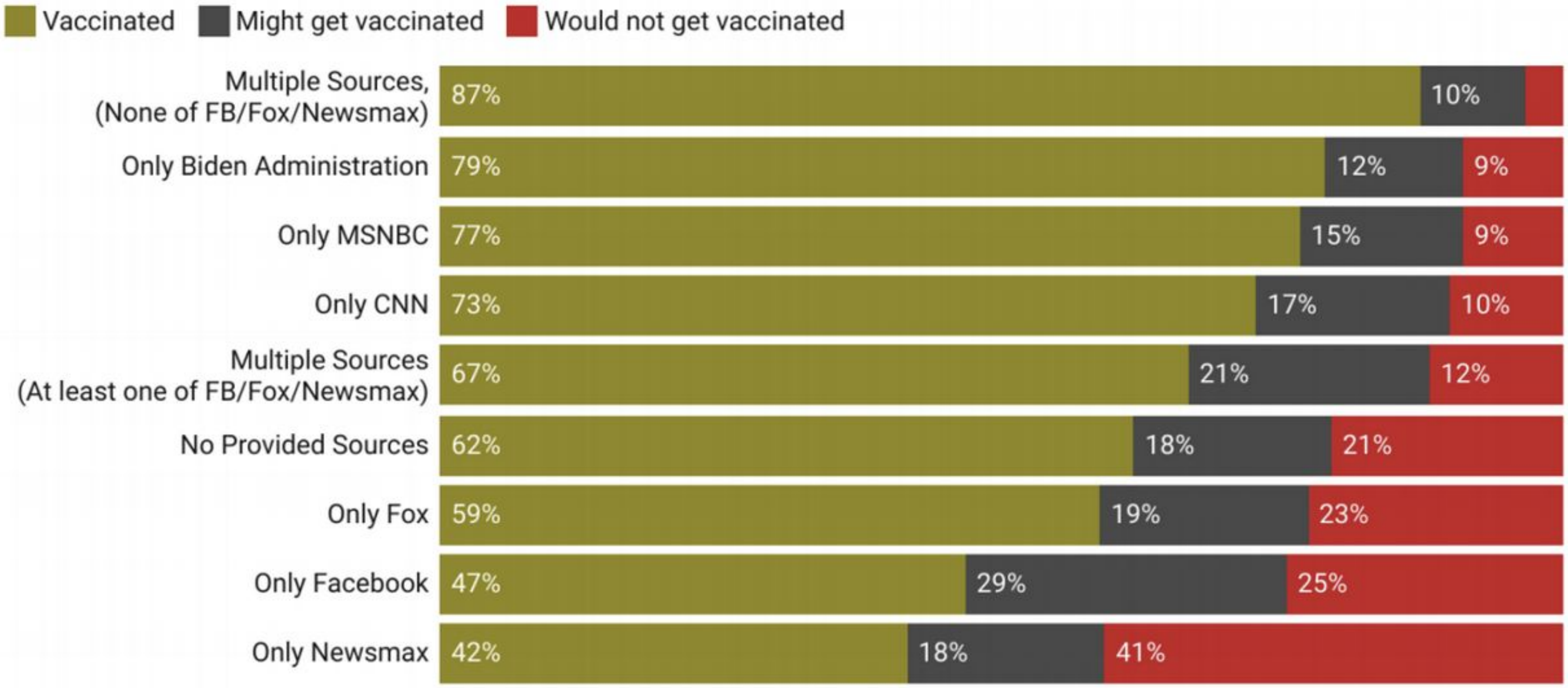
The ingredients of totalitarianism

- Strong, charismatic idealism

Facebook

COVID-19 vaccinations and news consumption patterns (Copy)

[Percent among respondents who say they got COVID-related news from each source in the past 24 hours]



National sample, N = 20,669, Time period: 06/09/2021-07/07/2021

Source: The COVID-19 Consortium for Understanding the Public's Policy Preferences Across States (A joint project of: Northeastern University, Harvard University, Rutgers University, and Northwestern University) www.covidstates.org • Created with Datawrapper

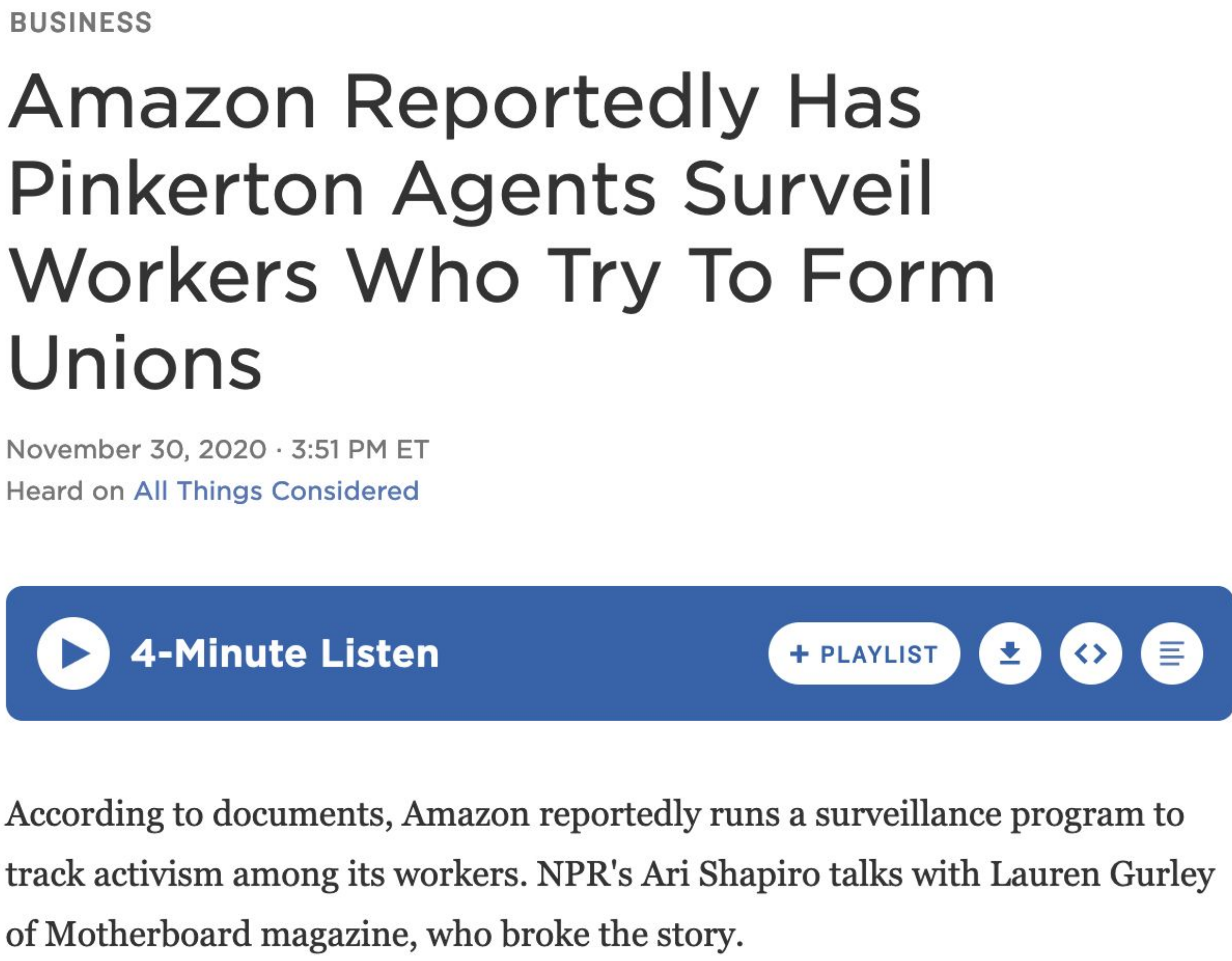
Amazon

Amazon

14-hour days and r breaks: Amazon's delivery drivers

Drivers report being underpaid an in their vehicles to keep up with de

▲ An Amazon delivery driver loads a van outside of a distribution facility on 2 February 2021 in Hawthorne, California. Photograph: Patrick T Fallon/AFP/Getty Images



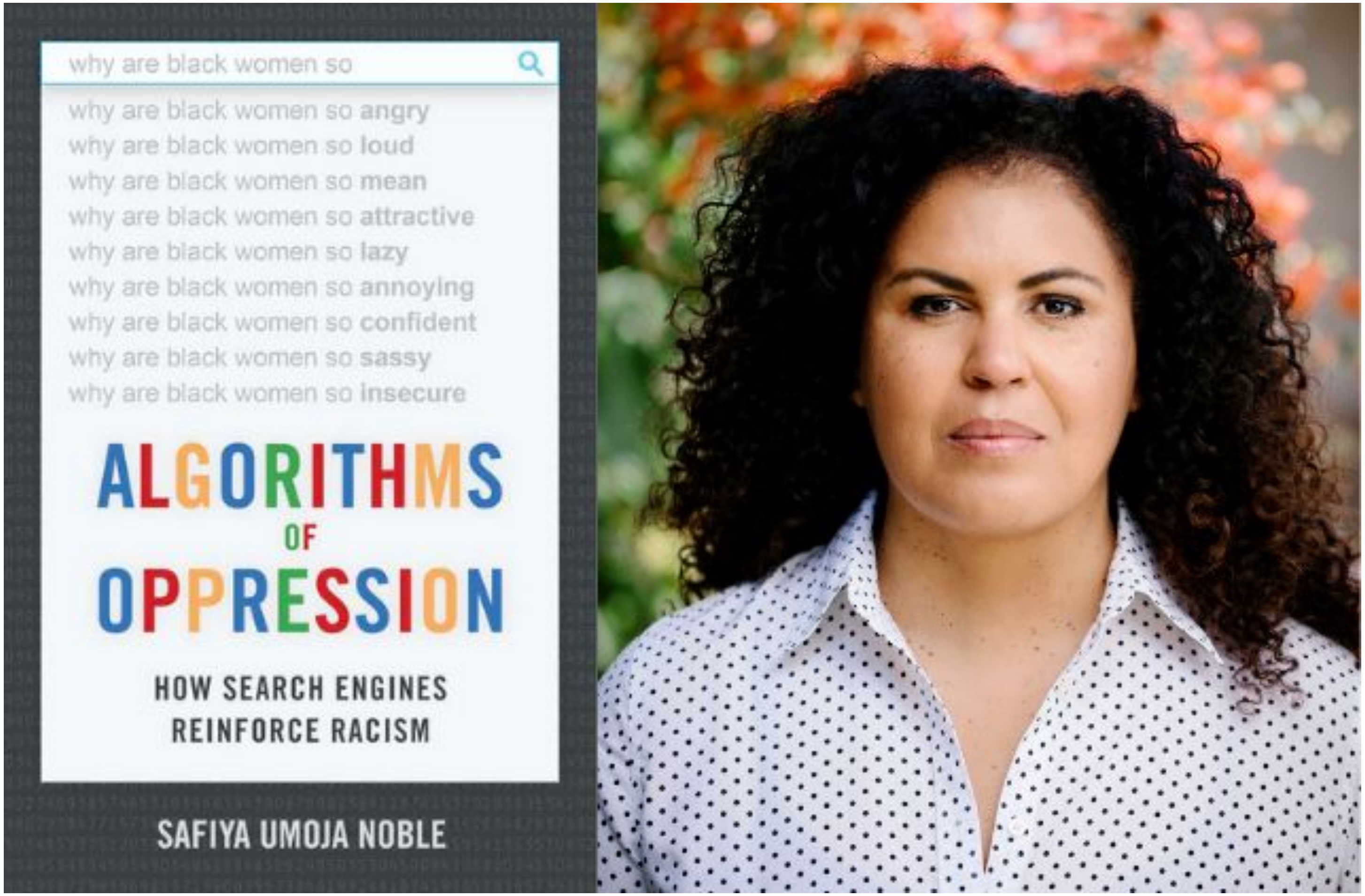
BUSINESS

Amazon Reportedly Has Pinkerton Agents Surveil Workers Who Try To Form Unions

November 30, 2020 · 3:51 PM ET
Heard on All Things Considered

According to documents, Amazon reportedly runs a surveillance program to track activism among its workers. NPR's Ari Shapiro talks with Lauren Gurley of Motherboard magazine, who broke the story.

Google


why are black women so
why are black women so angry
why are black women so loud
why are black women so mean
why are black women so attractive
why are black women so lazy
why are black women so annoying
why are black women so confident
why are black women so sassy
why are black women so insecure
ALGORITHMS OF OPPRESSION
HOW SEARCH ENGINES REINFORCE RACISM
SAFIYA UMOJA NOBLE

“Microsoft is trying to steal your data just as much as Google, they’re just not as popular”

“Everything’s driven by enterprise adoption, because that’s where the money is. Individuals aren’t prioritized”

I’m sure they mean well…

I’m sure they mean well

- Generally, utopia vision from Big Tech Leaders
    - “In the future, technology is going to...free us up to spend more time on the things we all care about, like enjoying and interacting with each other and expressing ourselves in new ways” *Zuckerberg, 2017*
    - Eliminate poverty, hunger,
    - Fulfil the needs of everyone

- I’m sure they have good intentions
    - But, no one has any idea how to operate at scale

VM's implemented to account for computing at scale.

Multi-process machines warrant virtual memory.

If the vision is scale, are we prepared for the scale that we look to operate at?

"I am here to suggest that you voluntarily renounce exercising the power which being an ~~American~~ *technologist* gives you. I am here to entreat you to freely, consciously and humbly give up the legal right you have to impose your benevolence on ~~Mexico~~ *the world*. I am here to challenge you to recognize your inability, your powerlessness and your incapacity to do the "good" which you intended to do.

I am here to entreat you to use your money, your status and your education to travel ~~in Latin America~~ *around the world*. Come to look, come to climb our mountains, to enjoy our flowers. Come to study. But do not come to help."

You have unprecedented power and access as technologists.

What would you like to accomplish?
Who do you want to serve?

Ideally, better than "move fast and break things"

Fork-Exec

Note: the return values of `fork` and `exec*` should be checked for errors

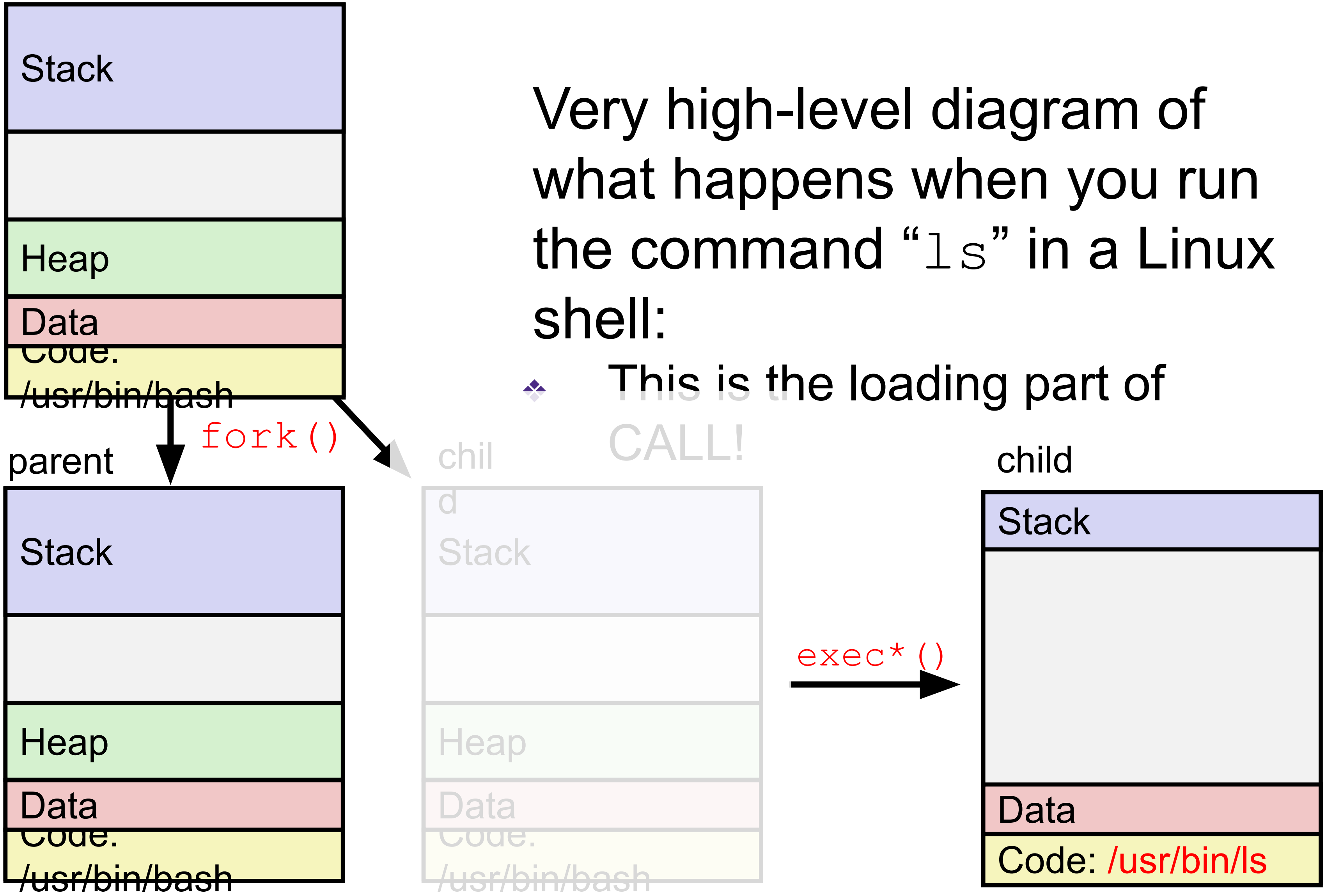
- fork-exec model:
  - `fork()` creates a copy of the current process
  - `exec*()` replaces the current process' code and address space with the code for a different program
    - Whole family of `exec` calls – see **`exec(3)`** and

```
// Example arguments: path="/usr/bin/ls",
//     argv[0]="/usr/bin/ls", argv[1]="-ahl", argv[2]=NULL
void fork_exec(char *path, char *argv[]) {
   pid_t fork_ret = fork();
   if (fork_ret != 0) {
      printf("Parent: created a child %d\n", fork_ret);
   } else {
      printf("Child: about to exec a new program\n");
      execv(path, argv);
   }
   printf("This line printed by parent only!\n");
}
```

Exec-ing a new program

Very high-level diagram of what happens when you run the command “ls” in a Linux shell:

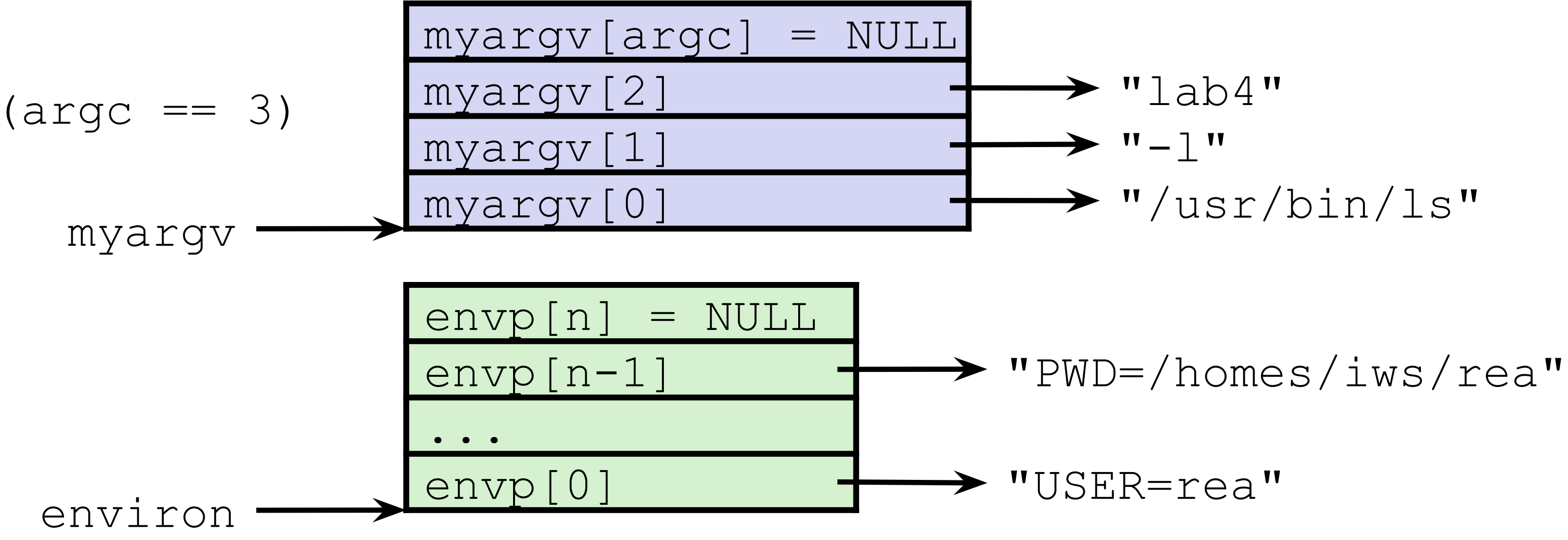
- This is the loading part of CALL!



execve Example

This is extra (non-testable) material

Execute "/usr/bin/ls -l lab4" in child process using current environment:

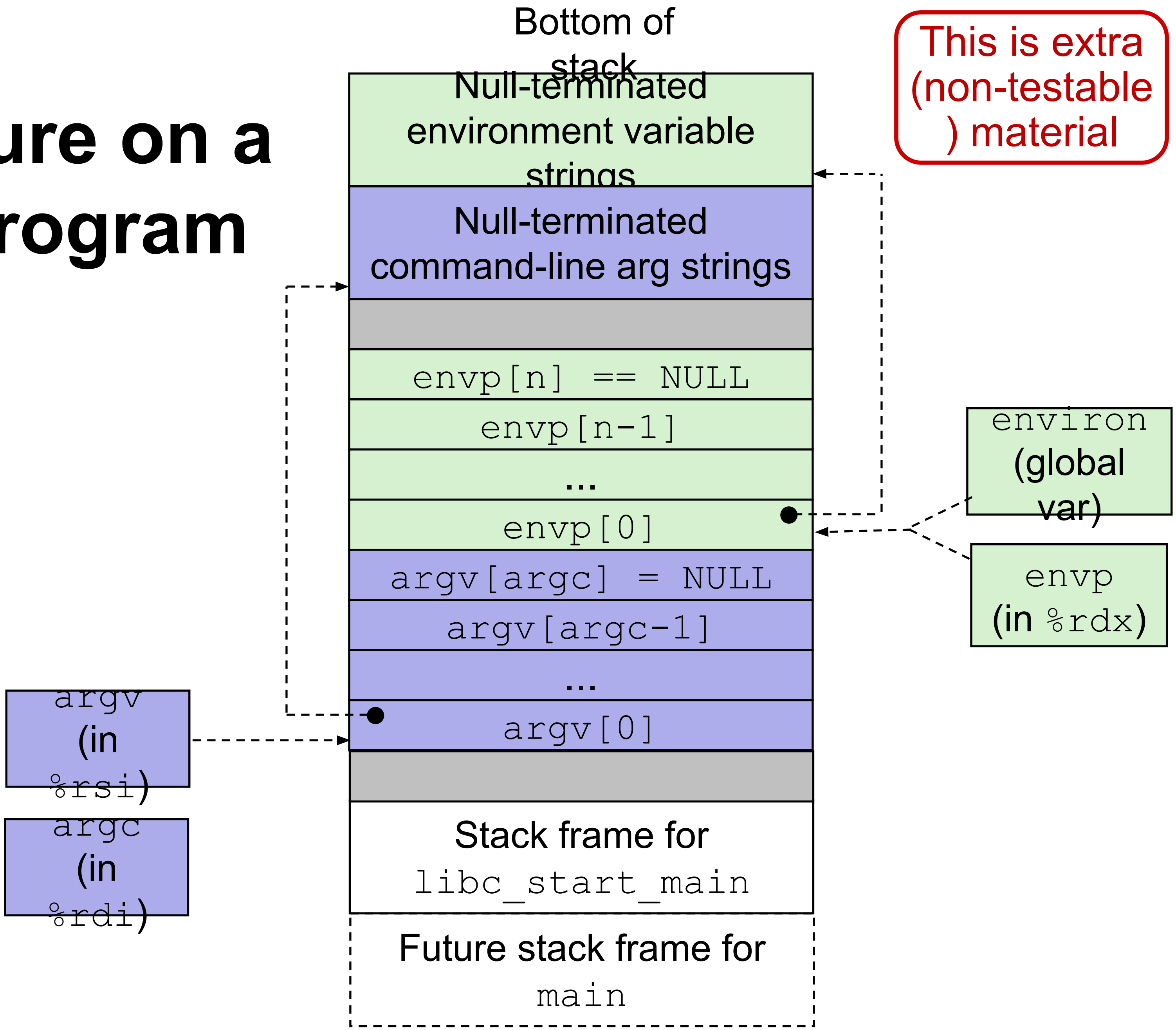


```
if ((pid = fork()) == 0) {   /* Child runs program */
   if (execve(myargv[0], myargv, environ) < 0) {
      printf("%s: Command not found.\n", myargv[0]);
      exit(1);
   }
}
```

Run the printenv command in a Linux shell to see your own environment variables

Stack Structure on a New Program Start

This is extra (non-testable) material



`exit`: Ending a process

- **`void`** `exit(`**`int`** `status)`
  - Explicitly exits a process
    - Status code: 0 is used for a normal exit, nonzero for abnormal exit

- The `return` statement from `main()` also ends a process in C
  - The return value is the status code

Processes

- Processes and context switching
- Creating new processes
  - `fork()`, `exec*()`, and `wait()`
- **Zombies**

Zombies

- A terminated process still consumes system resources
  - Various tables maintained by OS
  - Called a "zombie" (a living corpse, half alive and half dead)
- *Reaping* is performed by parent on terminated child
  - Parent is given exit status information and kernel then deletes zombie child process
- What if parent doesn't reap?
  - If any parent terminates without reaping a child, then the orphaned child will be reaped by `init` process (pid of 1)

`wait`: Synchronizing with Children

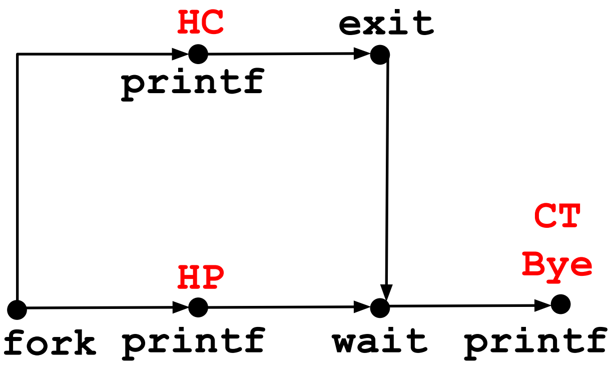
- **`int`** `wait(`**`int *`**`child_status)`
  - Suspends current process (*i.e.* the parent) until one of its children terminates
  - Return value is the PID of the child process that terminated
    - *On successful return, the child process is reaped*
  - If `child_status != NULL`, then the `*child_status` value indicates why the child process terminated
    - Special macros for interpreting this status – see **`man wait(2)`**
- **Note:** If parent process has multiple children, `wait` will return when *any* of the children terminates

wait: Synchronizing with Children

```
void fork_wait() {
   int child_status;

   if (fork() == 0) {
      printf("HC: hello from child\n");
      exit(0);
   } else {
      printf("HP: hello from parent\n");
      wait(&child_status);
      printf("CT: child has terminated\n");
   }
   printf("Bye\n");
}
```

forks.c



Feasible output:
HC
HP
CT
Bye

Infeasible output:
HP
CT
Bye
HC

Example: Zombi

```
void fork7() {
    if (fork() == 0) {
        /* Child */
        printf("Terminating Child, PID =
%d\n",
               getpid());
        exit(0);
    } else {
        printf("Running Parent, PID = %d\n",
               getpid());
        while (1); /* Infinite loop */
    }
```

forks.c

```
linux> ./forks 7 &
[1] 6639
Running Parent, PID = 6639
Terminating Child, PID = 6640
linux> ps
  PID TTY          TIME CMD
 6585 ttyp9    00:00:00 tcsh
 6639 ttyp9    00:00:03 forks
 6640 ttyp9    00:00:00 forks <defunct>
 6641 ttyp9    00:00:00 ps
linux> kill 6639
[1]    Terminated
linux> ps
  PID TTY          TIME CMD
 6585 ttyp9    00:00:00 tcsh
 6642 ttyp9    00:00:00 ps
```

- `ps` shows child process as "defunct"
- Killing parent allows child to be reaped by `init`

Example: Non-terminating Child

```
void fork8() {
   if (fork() == 0) {
      /* Child */
      printf("Running Child, PID = %d\n",
             getpid());
      while (1); /* Infinite loop */
   } else {
      printf("Terminating Parent, PID = 
%d\n",
             getpid());
      exit(0);
   }
}
```

forks.c

```
linux> ./forks 8
Terminating Parent, PID = 6675
Running Child, PID = 6676
linux> ps
  PID TTY          TIME CMD
 6585 ttyp9    00:00:00 tcsh
 6676 ttyp9    00:00:06 forks
 6677 ttyp9    00:00:00 ps
linux> kill 6676
linux> ps
  PID TTY          TIME CMD
 6585 ttyp9    00:00:00 tcsh
 6678 ttyp9    00:00:00 ps
```

- ○ Child process still active even though parent has terminated
- ○ Must kill explicitly, or else will keep running indefinitely

Process Management Summary

- `fork` makes two copies of the same process (parent & child)
  - Returns different values to the two processes
- `exec*` replaces current process from file (new program)
  - Two-process program:
    - First `fork()`
    - **if** (pid == 0) { *\/* child code *\/* } **else** { *\/* parent code *\/* }
  - Two different programs:
    - First `fork()`
    - **if** (pid == 0) { execv(…) } **else** { *\/* parent code *\/* }
- `wait` or `waitpid` used to synchronize parent/child