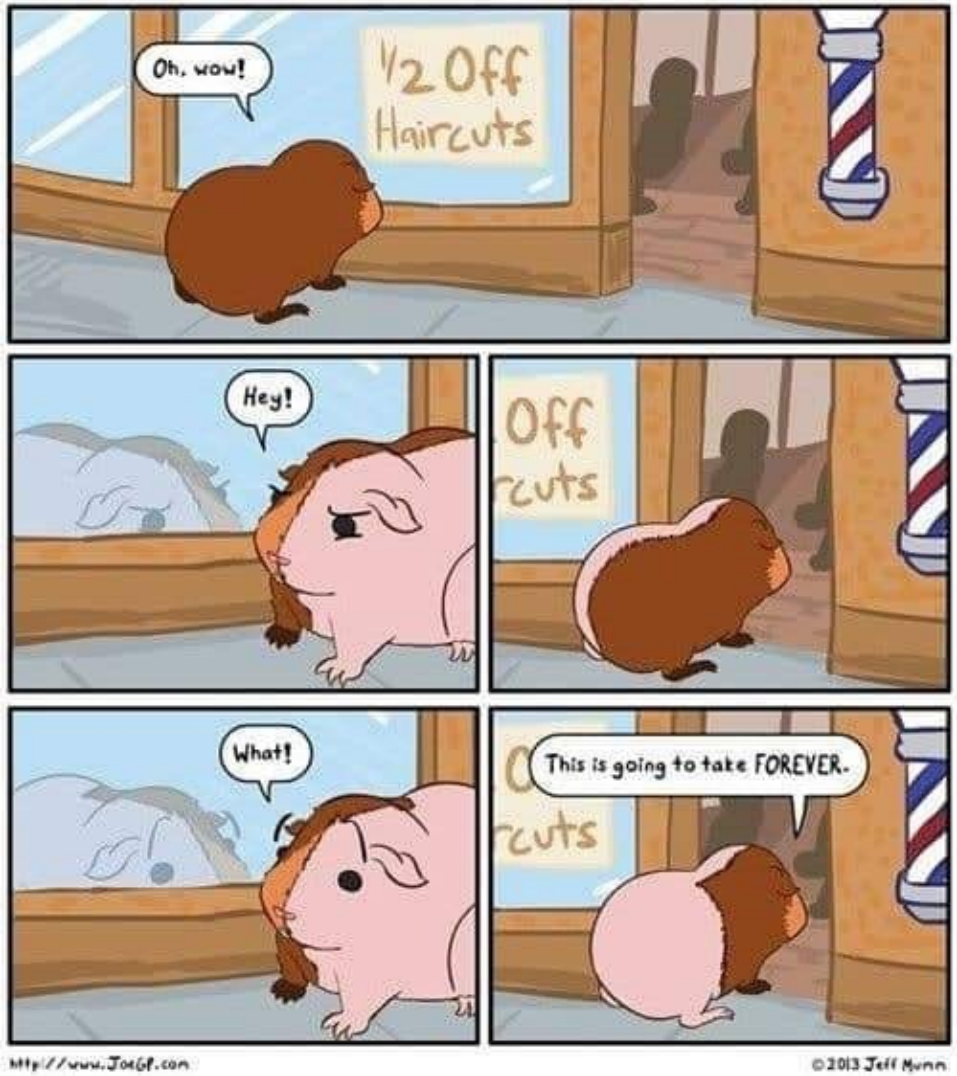
Memory Allocation I

CSE 351 Winter 2022

Instructor:
Sam Wolfson

Teaching Assistants:
Angela Xu
Anirudh Kumar
Catherine Guevara
Dara Stotland
Harrison Bay
Ian Hsiao
Kevin Wang
Mara Kirdani-Ryan
Nick Durand
Sanjana Sridhar



Relevant Course Information

- hw21 due tonight, hw22 due Friday
- Lab 5 released today
  - Writing a memory allocator in C
- Go to section tomorrow (super helpful for lab 5)
- Double-check your midterm grades!

- Final exam: March 15—17
  - Structure will be very similar to the midterm
  - *Not* cumulative, focused on post-midterm material
  - Section 10 will be final review, plus a bit of VM
    - We're doing Memory Allocation this week for the lab

Computing and Vision

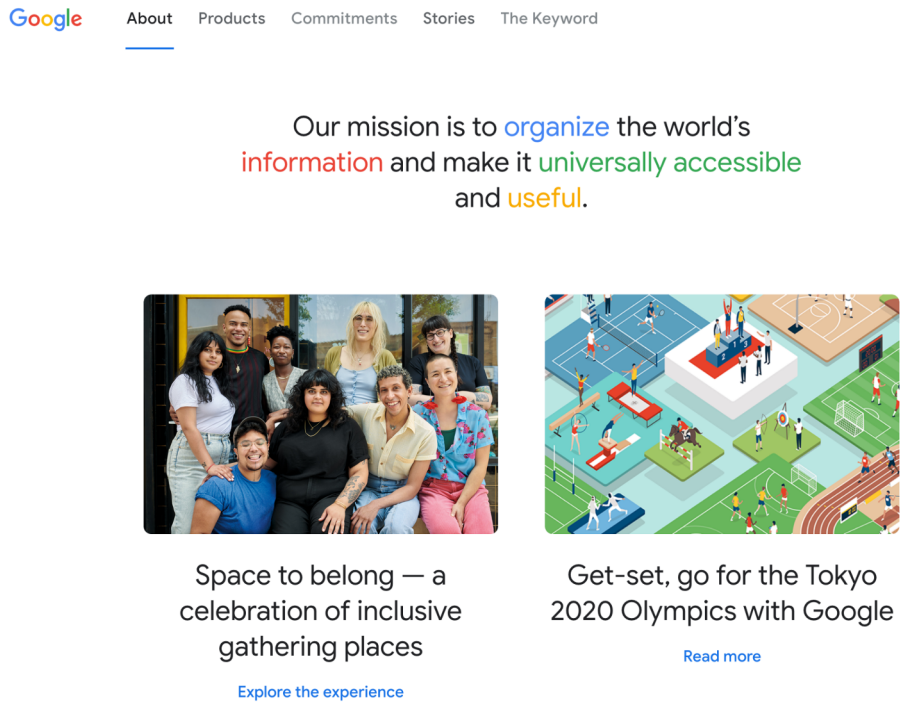
- What's **your** vision for computing?
  - What do you think it will look like in 20 years?

- Is there a **collective** vision?

Computing and 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



- What's the difference here?

What's the ideological vision of computing?

Google About Products Commitments Stories The Keyword

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

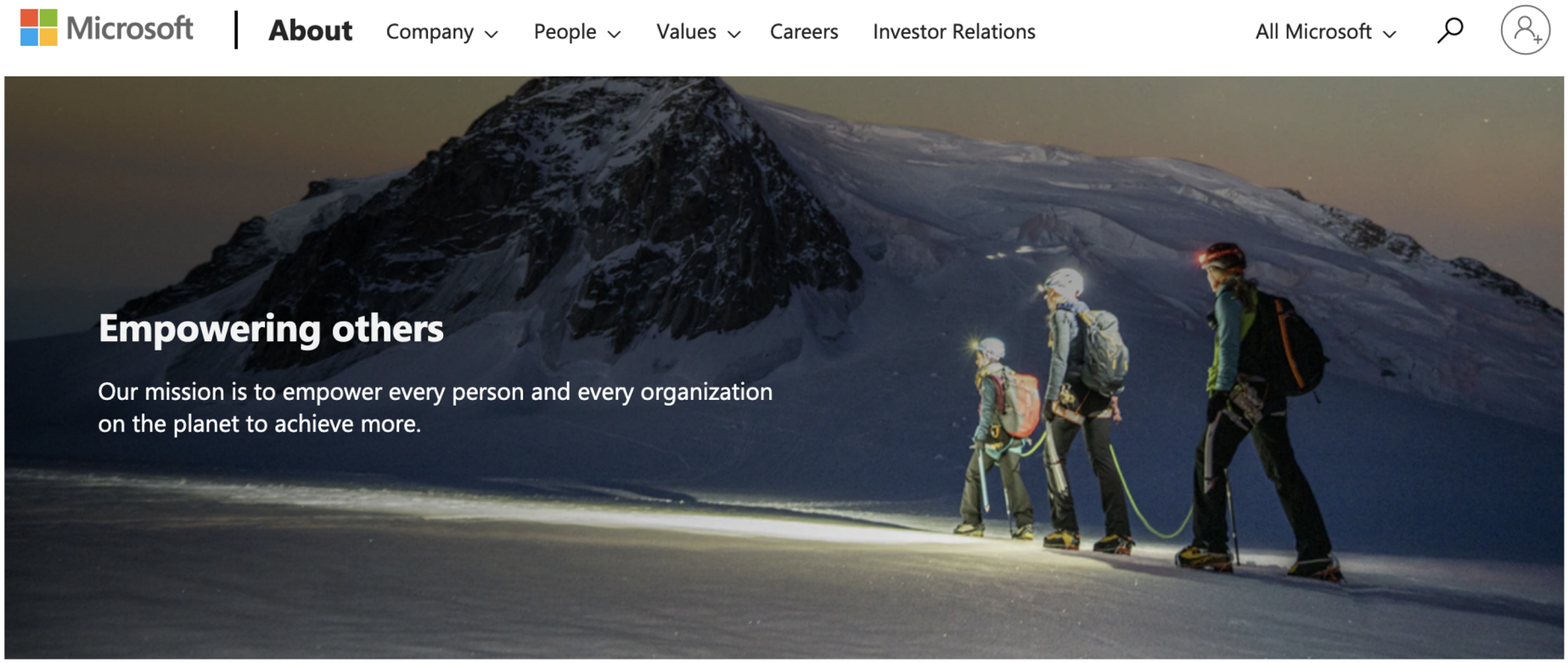
Space to belong — a celebration of inclusive gathering places

Explore the experience

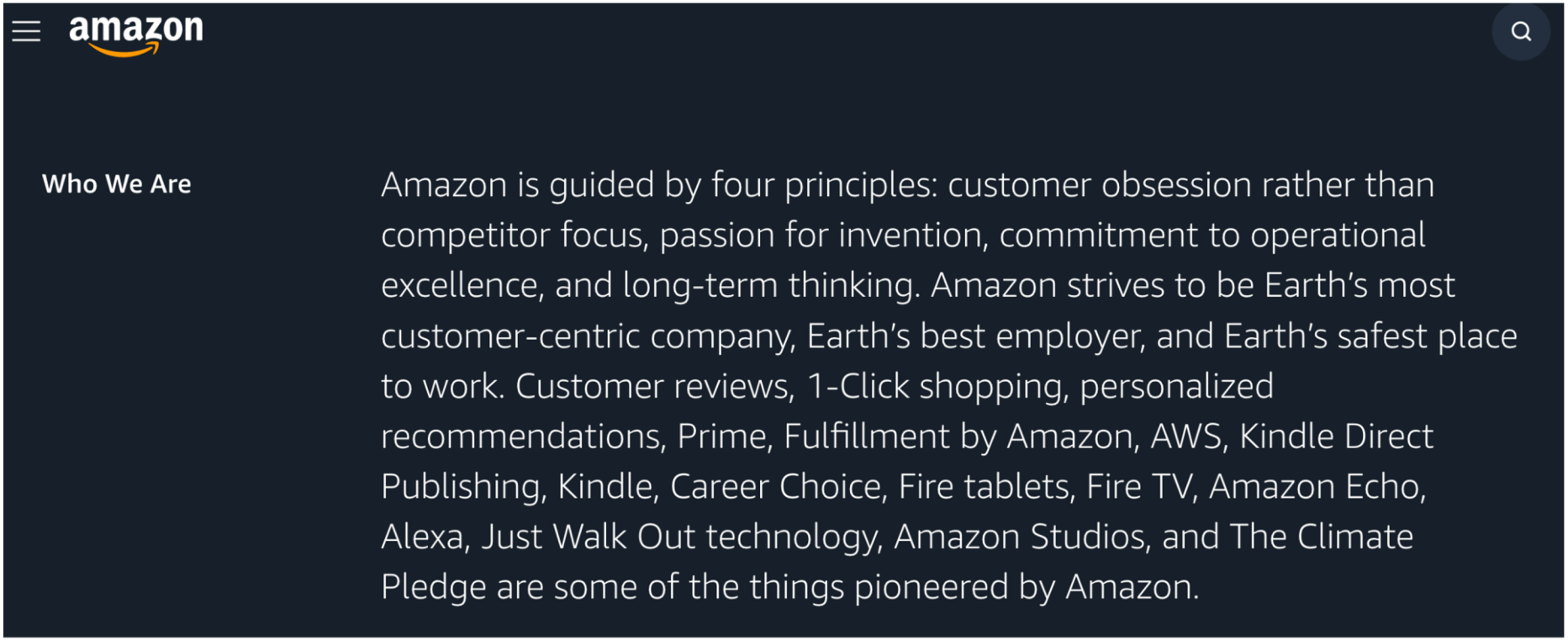
Get-set, go for the Tokyo 2020 Olympics with Google

Read more

not a computer in sight 🤔



not a computer in sight 🤔


amazon
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.

Earth's…

Vision: Operating at a Global Scale

"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

Vision: Operating at a Global Scale

*"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

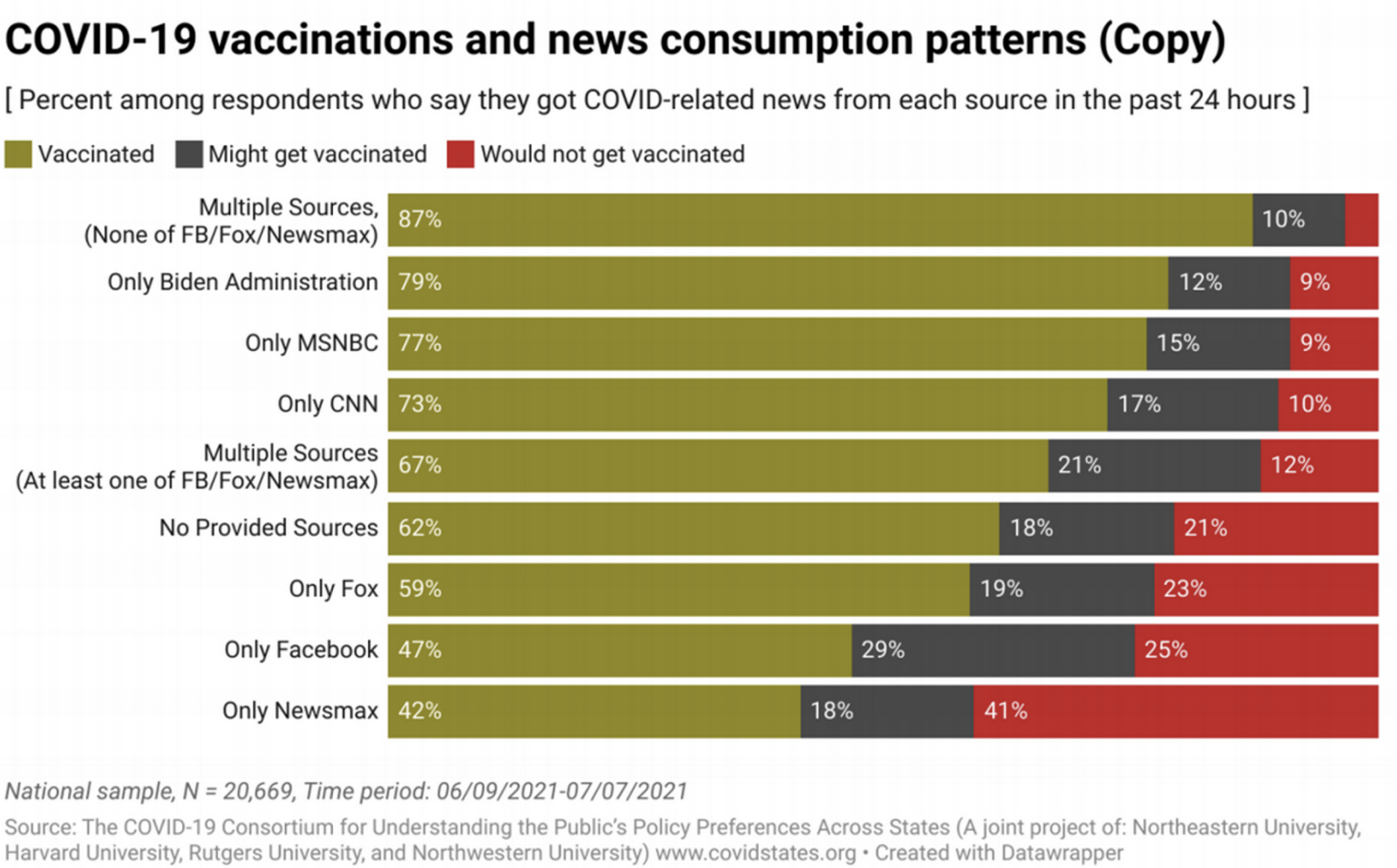
Vision: Operating at a Global Scale

- These sound like noble tasks!
  - Solve the world's important problems
  - Connect the world together
  - Help the world achieve more
  - Be the Earth's best company at everything

- But these are **hard** problems to solve…
  - …and global platforms have global consequences.

- How has this worked out in practice?
  - Are computer scientists uniquely suited to solve these?

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]
Vaccinated
Might get vaccinated
Would not get vaccinated
Multiple Sources, (None of FB/Fox/Newsmax)
87%
10%
Only Biden Administration
79%
12%
9%
Only MSNBC
77%
15%
9%
Only CNN
73%
17%
10%
Multiple Sources (At least one of FB/Fox/Newsmax)
67%
21%
12%
No Provided Sources
62%
18%
21%
Only Fox
59%
19%
23%
Only Facebook
47%
29%
25%
Only Newsmax
42%
18%
41%
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 no bathroom breaks: Amazon's overworked delivery drivers

Drivers report being underpaid and having
in their vehicles to keep up with delivery r

BUSINESS

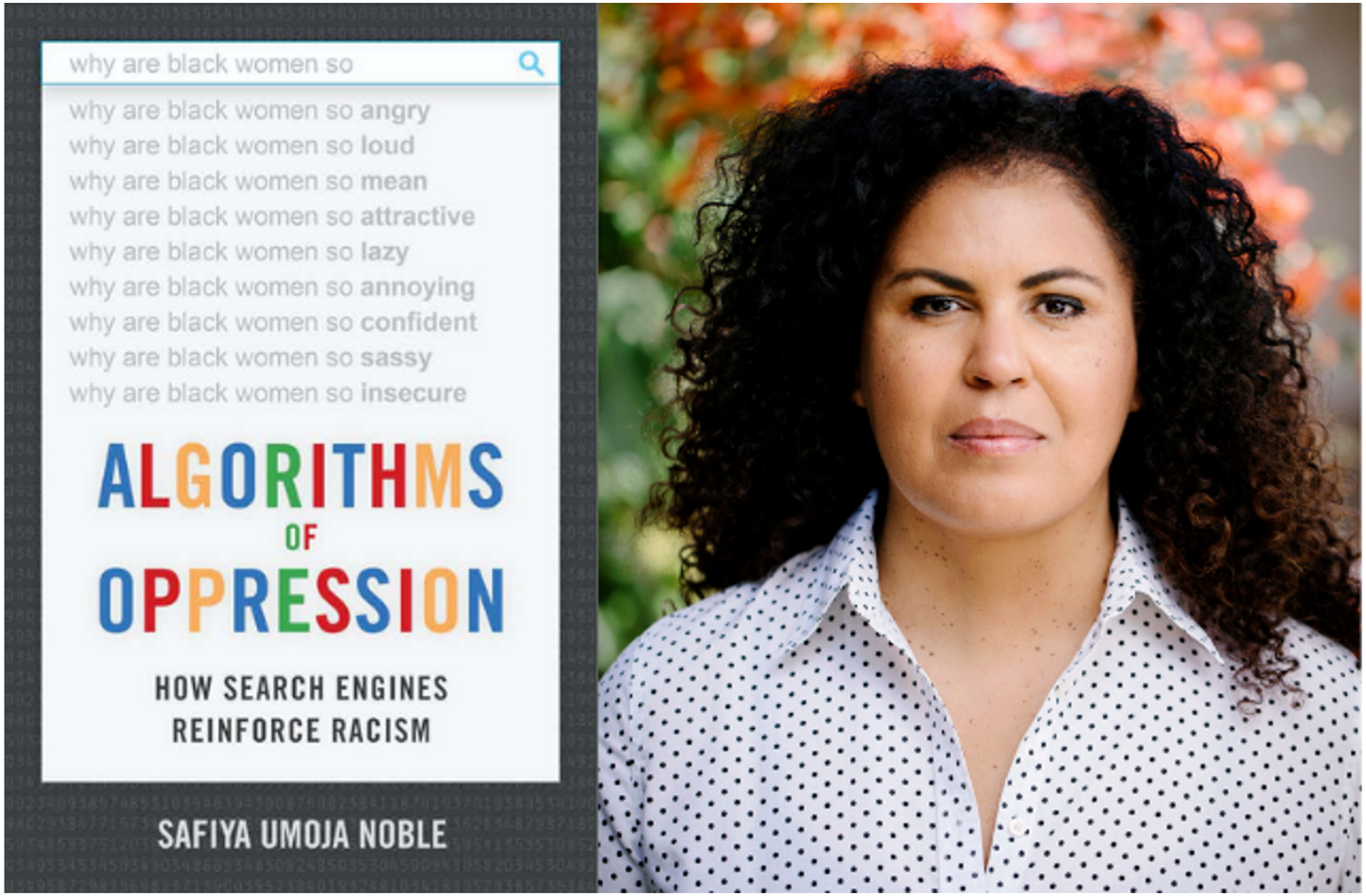
Amazon Reportedly Has Pinkerton Agents Surveil Workers Who Try To Form Unions

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

4-Minute Listen + PLAYLIST

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

They probably mean well!

- Generally, utopian 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."* – Mark Zuckerberg, 2017
- Eliminate poverty, hunger, etc.
- Fulfill the needs of **everyone**

- I'm sure they have good intentions…
  - …but no one has any idea how to operate at scale

Computing at Scale, Revisited

- We had to add a lot of new abstractions and procedures for VM to run multiple processes!
  - If the vision for computing **at large** is scale, are we prepared for the scale that we look to operate at?

- **Why** would computer scientists be uniquely suited to solve global issues?

"I am here to suggest that you voluntarily renounce exercising the power which being an ~~American~~ *computer scientist* 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."

Ivan Illich, *To Hell with Good Intentions*

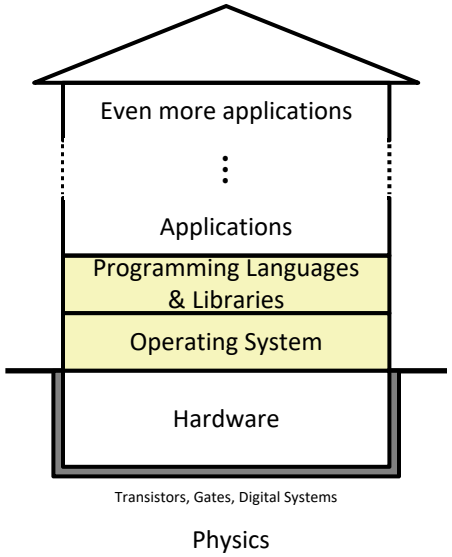
https://www.uvm.edu/~jashman/CDAE195_ESCI375/To%20Hell%20with%20Good%20Intentions.pdf

Computing at Scale, Revisited

- We had to add a lot of new abstractions and procedures for VM to run multiple processes!
  - If the vision for computing **at large** is scale, are we prepared for the scale that we look to operate at?
- You have incredible power and access as computer scientists.
  - What would *you* like to accomplish?
  - Who do *you* want to serve?
- "Move fast and break things" doesn't scale well…

The Hardware/Software Interface

- Topic Group 3: **Scale & Coherence**
  - Caches, Processes, Virtual Memory, **Memory Allocation**



- How do we maintain logical consistency in the face of more data and more processes?
  - How do we support control flow both within many processes and things external to the computer?
  - How do we support data access, including dynamic requests, across multiple processes?

Reading Review

- Terminology:
  - Dynamically-allocated data: malloc, free
  - Allocators: implicit vs. explicit allocators, heap blocks, implicit vs. explicit free lists
  - Heap fragmentation: internal vs. external

- Questions from the Reading?

Multiple Ways to Store Program Data

- Static global data
  - *Fixed size* at compile-time
  - Entire *lifetime of the program* (loaded from executable)
  - Portion is read-only (*e.g.*, string literals)
- Stack-allocated data
  - Local/temporary variables
    - *Can* be dynamically sized (in some versions of C)
  - *Known lifetime* (deallocated on `return`)
- **Dynamic (heap) data**
  - Size known only at runtime (*i.e.*, based on user-input)
  - Lifetime known only at runtime (long-lived data structures)

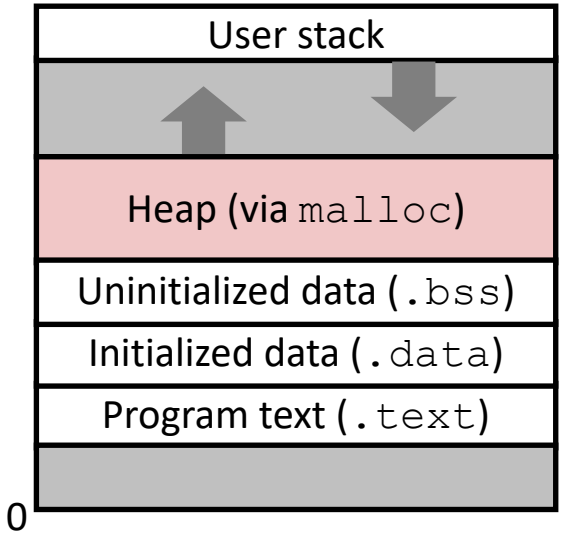
```
int array[1024];

void foo(int n) {
  int tmp;
  int local_array[n];

  int* dyn =
    (int*)malloc(n*sizeof(int));
}
```

Memory Allocation

- **Dynamic memory allocation**
  - **Introduction and goals**
  - **Allocation and deallocation (free)**
  - **Fragmentation**
- Explicit allocation implementation
  - Implicit free lists
  - Explicit free lists (Lab 5)
  - Segregated free lists
- Implicit deallocation: garbage collection
- Common memory-related bugs in C

Dynamic Memory Allocation (Review)

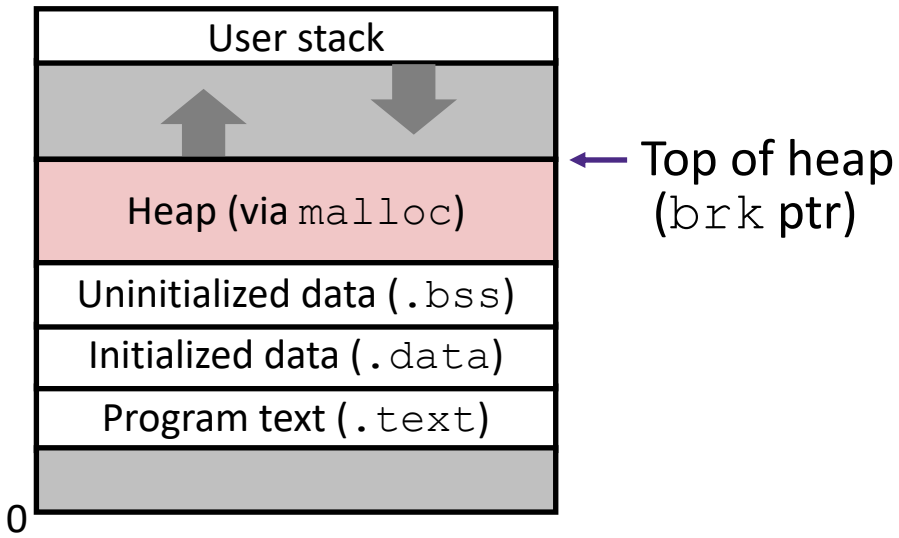
- Programmers use dynamic memory allocators to acquire virtual memory at run time
  - For data structures whose size (or lifetime) is known only at runtime
  - Manage the heap of a process' virtual memory:



- Types of allocators
  - ***Explicit* allocator:** programmer allocates and frees space
    - Example: `malloc` and `free` in C
  - ***Implicit* allocator:** programmer only allocates space (no free)
    - Example: garbage collection in Java, Ruby, and Python

Dynamic Memory Allocation

- Allocator organizes heap as a collection of variable-sized *blocks*, which are either *allocated* or *free*
  - Allocator requests pages in the heap region; virtual memory hardware and OS kernel allocate these pages to the process
  - Application objects are typically smaller than pages, so the allocator manages heap blocks *within* pages
    - (Larger objects handled too; ignored here)



Allocating Memory in C (Review)

- Need to `#include <stdlib.h>`
- **`void*`** `malloc(`**`size_t`** `size)`
  - Allocates a continuous block of `size` bytes of uninitialized memory
  - Returns a pointer to the beginning of the allocated block; `NULL` indicates a failed request
    - Typically aligned to an 8-byte (x86) or 16-byte (x86-64) boundary
    - Returns `NULL` if allocation failed (also sets `errno`) or `size==0`
  - Different blocks not necessarily adjacent
- Good practices:
  - `ptr = (int*) malloc(n*sizeof(int));`
    - `sizeof` makes code more portable
    - `void*` is implicitly cast into any pointer type; explicit typecast will help you catch coding errors when pointer types don't match

Allocating Memory in C (Review)

- Need to `#include <stdlib.h>`
- **`void*`** `malloc(`**`size_t`** `size)`
  - Allocates a continuous block of `size` bytes of uninitialized memory
  - Returns a pointer to the beginning of the allocated block; `NULL` indicates a failed request
    - Typically aligned to an 8-byte (x86) or 16-byte (x86-64) boundary
    - Returns `NULL` if allocation failed (also sets `errno`) or `size==0`
  - Different blocks not necessarily adjacent
- Friends of `malloc`:
  - **`void*`** `calloc(`**`size_t`** `nitems, `**`size_t`** `size)`
    - "Zeros out" allocated block
  - **`void*`** `realloc(`**`void*`** `ptr, `**`size_t`** `size)`
    - Changes the size of a previously allocated block (if possible)
  - **`void*`** `sbrk(`**`intptr_t`** `increment)`
    - Used internally by allocators to grow or shrink the heap

Freeing Memory in C (Review)

- Need to `#include <stdlib.h>`
- **`void`** `free(`**`void*`** `p)`
  - Releases whole block pointed to by `p` to the pool of available memory
  - Pointer `p` must be the address *originally* returned by `m/c/realloc` (*i.e.*, beginning of the block), otherwise system exception raised
  - Don't call `free` on a block that has already been released
  - No action occurs if you call `free(NULL)`

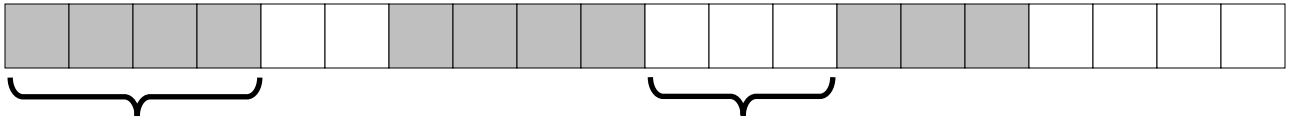
Memory Allocation Example in C

```
void foo(int n, int m) {
  int i, *p;
  p = (int*) malloc(n*sizeof(int)); /* allocate block of n ints */
  if (p == NULL) {                  /* check for allocation error */
    perror("malloc");
    exit(0);
  }
  for (i=0; i<n; i++)               /* initialize int array */
    p[i] = i;
                                    /* add space for m ints to end of p block */
  p = (int*) realloc(p,(n+m)*sizeof(int));
  if (p == NULL) {                  /* check for allocation error */
    perror("realloc");
    exit(0);
  }
  for (i=n; i < n+m; i++)           /* initialize new spaces */
    p[i] = i;
  for (i=0; i<n+m; i++)             /* print new array */
    printf("%d\n", p[i]);
  free(p);                          /* free p */
}
```


= 1 word = 8 bytes

Notation

- We will draw memory divided into *words*
  - Each *word* is 64 bits = 8 bytes
  - Allocations will be in sizes that are a multiple of words (*i.e.*, multiples of 8 bytes)
  - Book and old videos still use 4-byte *word*
    - Holdover from 32-bit version of textbook



Allocated block
(4 words)

Free block
(3 words)

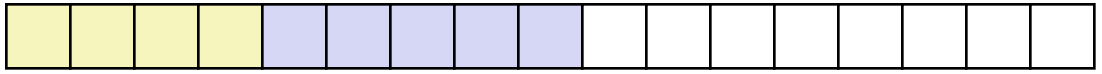

Free word


Allocated word

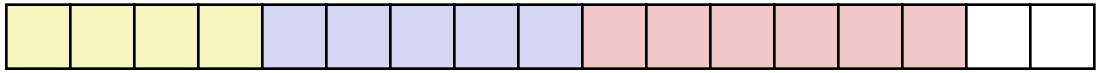
Allocation Example


```
p1 = malloc(32)
```

```
p2 = malloc(40)
```



```
p3 = malloc(48)
```



```
free(p2)
```

```
p4 = malloc(16)
```

Implementation Interface (Review)

- **Applications**
  - Can issue arbitrary sequence of `malloc` and `free` requests
  - Must never access memory not currently allocated
  - Must never free memory not currently allocated
    - Also must only use `free` with previously `malloc`'ed blocks

- **Allocators**
  - Can't control number or size of allocated blocks
  - Must respond immediately to `malloc`
  - Must allocate blocks from free memory
  - Must align blocks so they satisfy all alignment requirements
  - Can't move the allocated blocks

Performance Goals (Review)

- **Goals:** Given some sequence of `malloc` and `free` requests $R_0, R_1, \ldots, R_k, \ldots, R_{n-1}$, maximize throughput and peak memory utilization
  - These goals are often conflicting

1) Throughput

- Number of completed requests per unit time
- Example:
  - If 5,000 `malloc` calls and 5,000 `free` calls completed in 10 seconds, then throughput is 1,000 operations/second

Performance Goals

- Definition: *Aggregate payload* P_k
  - `malloc(p)` results in a block with a *payload* of `p` bytes
  - After request R_k has completed, the *aggregate payload* P_k is the sum of currently allocated payloads
- Definition: *Current heap size* H_k
  - Assume H_k is monotonically non-decreasing
    - Allocator can increase size of heap using `sbrk`

2) Peak Memory Utilization

- Defined as $U_k = (\max_{i \le k} P_i)/H_k$ after k+1 requests
- Goal: maximize utilization for a sequence of requests
- Why is this hard? And what happens to throughput?

Fragmentation (Review)

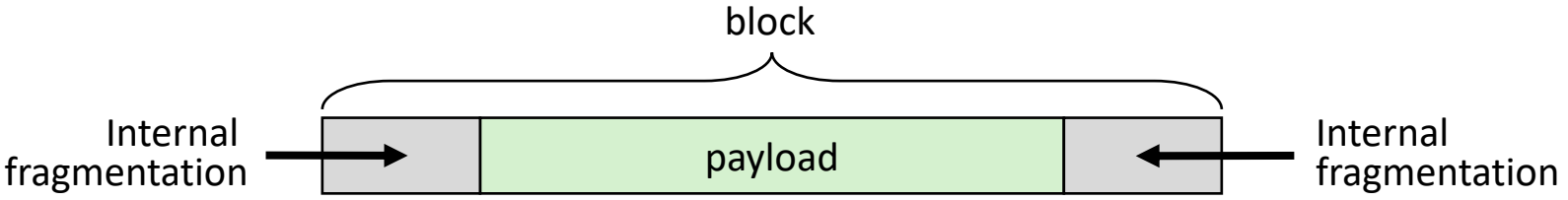
- Poor memory utilization is caused by *fragmentation*
  - Sections of memory are not used to store anything useful, but cannot satisfy allocation requests
  - Two types: *internal* and *external*

- **Recall:** Fragmentation in structs
  - Internal fragmentation was wasted space *inside* of the struct (between fields) due to alignment
  - External fragmentation was wasted space *between* struct instances (*e.g.*, in an array) due to alignment

- Now referring to wasted space in the heap *inside* or *between* allocated blocks

Internal Fragmentation

- For a given block, *internal fragmentation* occurs if payload is smaller than the block

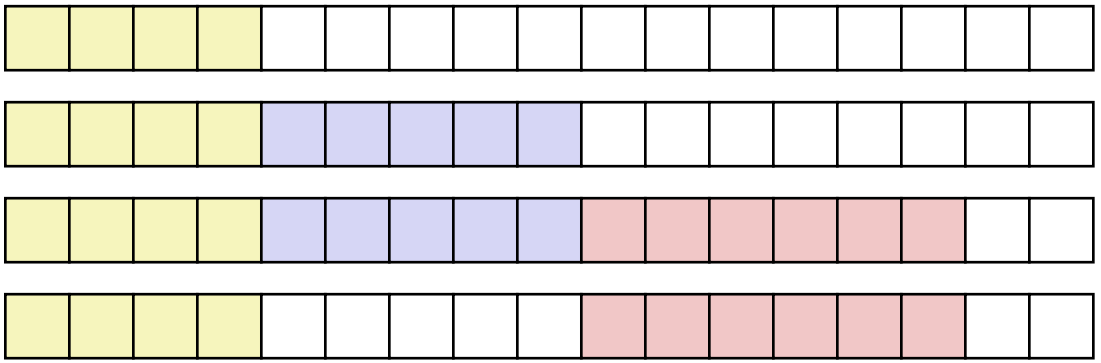


- **Causes:**
  - Padding for alignment purposes
  - Overhead of maintaining heap data structures (inside block, outside payload)
  - Explicit policy decisions (*e.g.*, return a big block to satisfy a small request)
- Easy to measure because only depends on past requests

External Fragmentation

- For the heap, *external fragmentation* occurs when allocation/free pattern leaves "holes" between blocks
  - That is, the aggregate payload is non-continuous
  - Can cause situations where there is enough aggregate heap memory to satisfy request, but no single free block is large enough

```
p1 = malloc(32)
p2 = malloc(40)
p3 = malloc(48)
free(p2)
p4 = malloc(48)
```



Oh no! (What would happen now?)

- Don't know what future requests will be
  - Difficult to impossible to know if past placements will become problematic

Polling Question

- Which of the following statements is FALSE?
  - A. Temporary arrays should *not* be allocated on the Heap
  - B. `malloc` returns an address of a block that is filled with mystery data
  - C. Peak memory utilization is a measure of both internal and external fragmentation
  - D. An allocation failure will cause your program to stop
  - E. We're lost…