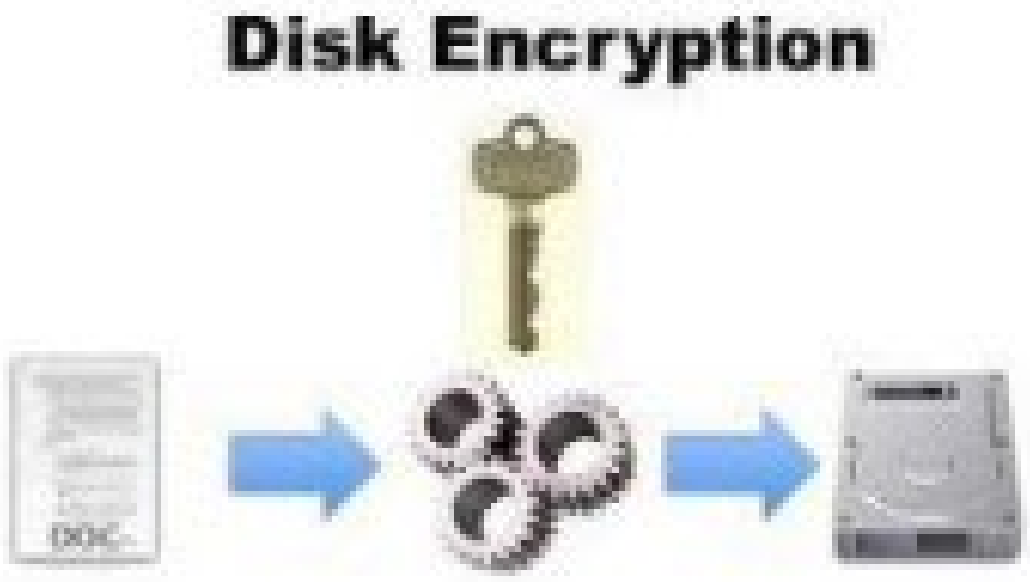
Software Systems

Day 5 - Information Theory, Memory

Intro: Cold Boot Attack

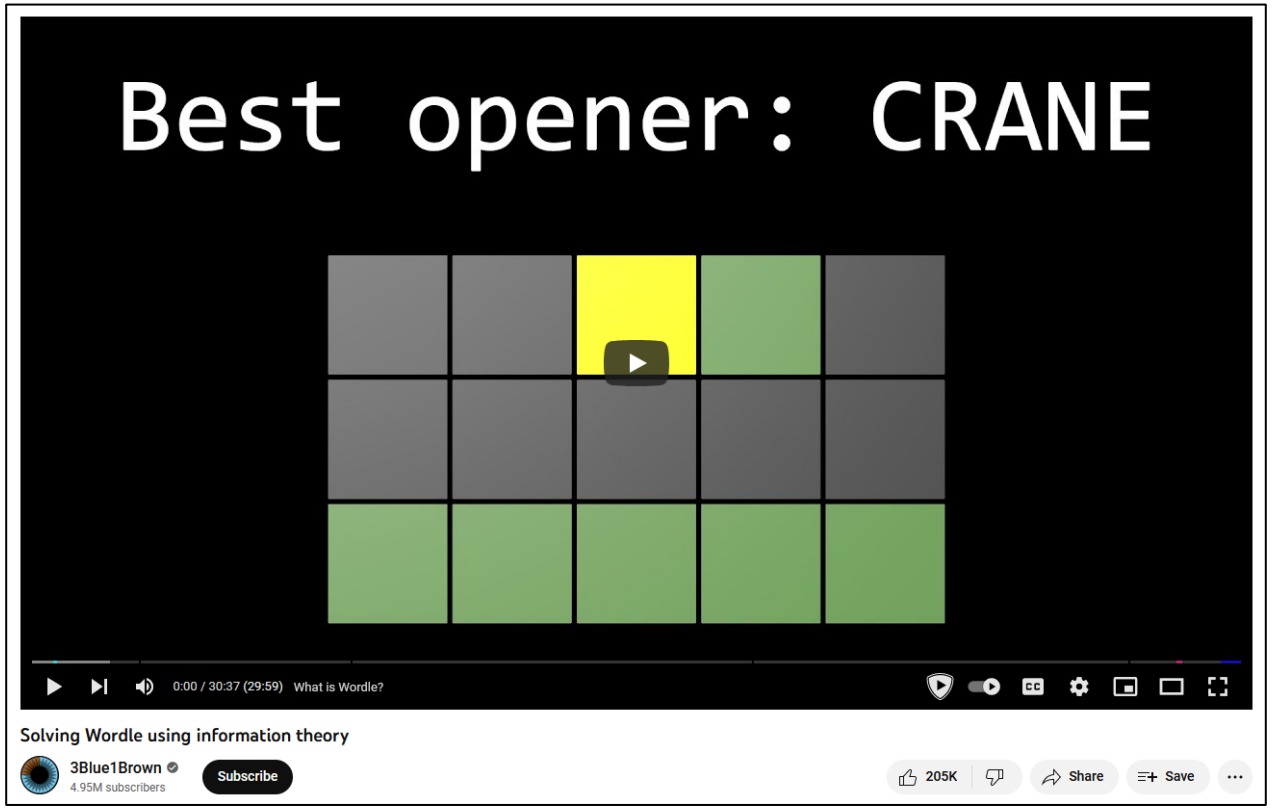
- Think OS Chapter 3 mentions that when you power off the machine, the contents of main memory (RAM) is lost.
- This is...sort of true.

Intro: Cold Boot Attack



Information Theory

- Information theory is useful far outside of computer systems.



Information Theory

- In systems, information theory is useful for:
  - Calculating how many bits of information it takes to store something.
  - Figuring out how to encode different types of data as a series of bits.
  - Calculating how many bits of information you get from something happening.
- So being able to figure out how to calculate information and work with bits is helpful.

Information Theory

- The formula $-\log_2 p$ tells you how many bits are communicated when you observe an event that occurs with probability p.
- Example:
  - You randomly pick one of the following colors: red, orange, yellow, green, blue, purple, pink, brown.
  - I randomly guess one of the colors, and I get it right.
  - I can do that with probability $\frac{1}{8} = 0.125$.
  - That means that I get $-\log_2 0.125 = 3$ bits of information from guessing correctly.
  - If there were only six colors to choose from, I would get ~2.58 bits of information.

Information Theory

- When we say it takes n bits to store some type of data (like characters), we have to:
  - Calculate all of the possible values of that type.
  - Assume that each value is equally likely.
  - Calculate the self-information of one of those values occurring.
  - Round that value up.
- Examples:
  - Six possible colors: ~2.58 bits (takes 3 bits to store one color)
  - Single decimal digit: ~3.32 bits (takes 4 bits to store)
  - Single letter of the English alphabet: ~4.7 bits (takes 5 bits to store)

Information Theory

- But practice, not all values are equally likely.
- "e" appears 13% of the time in English text: ~2.94 bits
- "z" appears 0.074% of the time: ~10.4 bits
- The less likely something is, the more "information" it carries.

Information Theory: Exercise

- Calculate the self-information of the following:
  - The outcome of a fair coin flip
  - The outcome of rolling a fair, 12-sided die.
  - The letter "q" in English (look up letter frequencies for this).
  - Being dealt a full house in poker.
  - Assuming all characters are equally likely, seeing a whitespace character in C (look at the `isspace` function documentation).

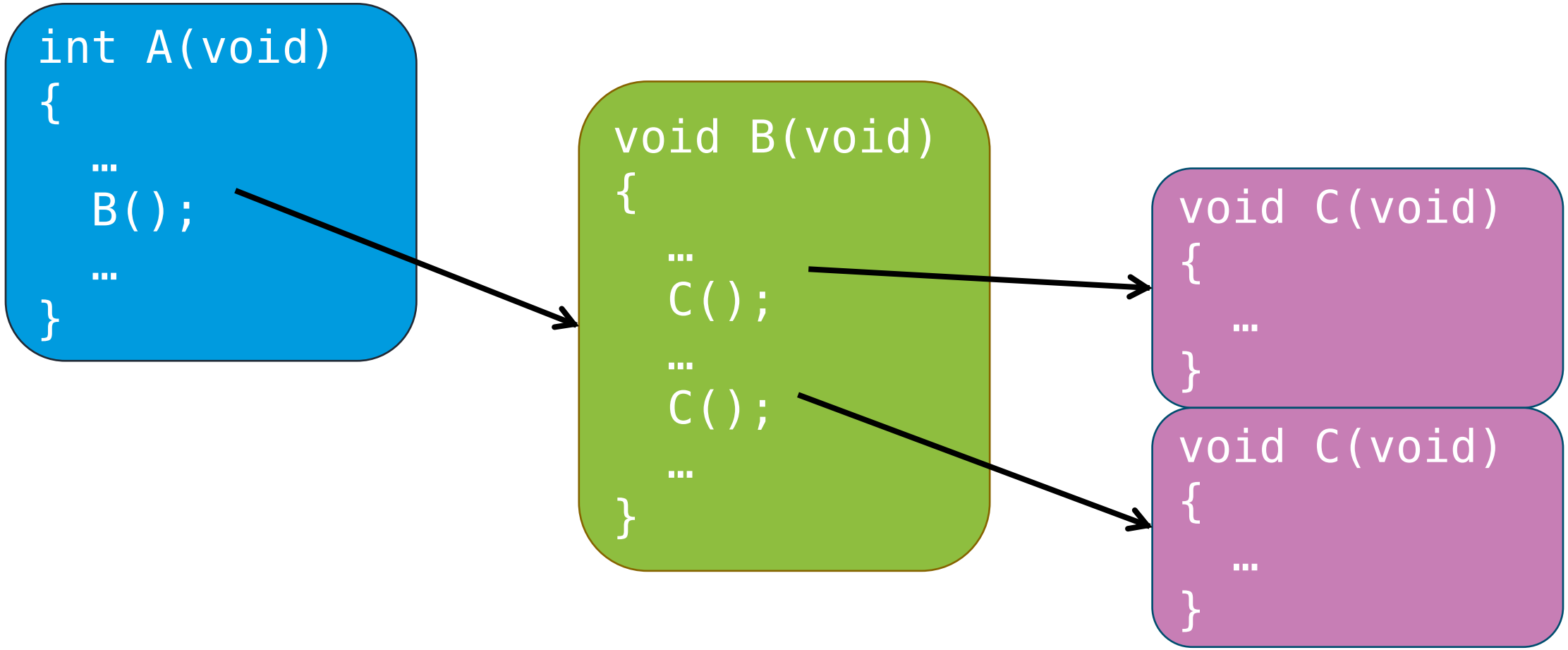
Information Theory: Scale

- There's a difference between KiB (kibibyte) and KB (kilobyte).
  - 1 KiB is 2^10, or 1024 bytes.
  - 1 KB is 10^3, or 1000 bytes.
- This means 1 KiB is about 2.4% larger than a KB, which can add up with larger units.
- Also, this means there's a nice trick to approximately convert between powers of 2 and 10.
  - 2^x is roughly 10^(0.3x).
  - 10^y is roughly 2^(3.3y).

Information Theory: Scale

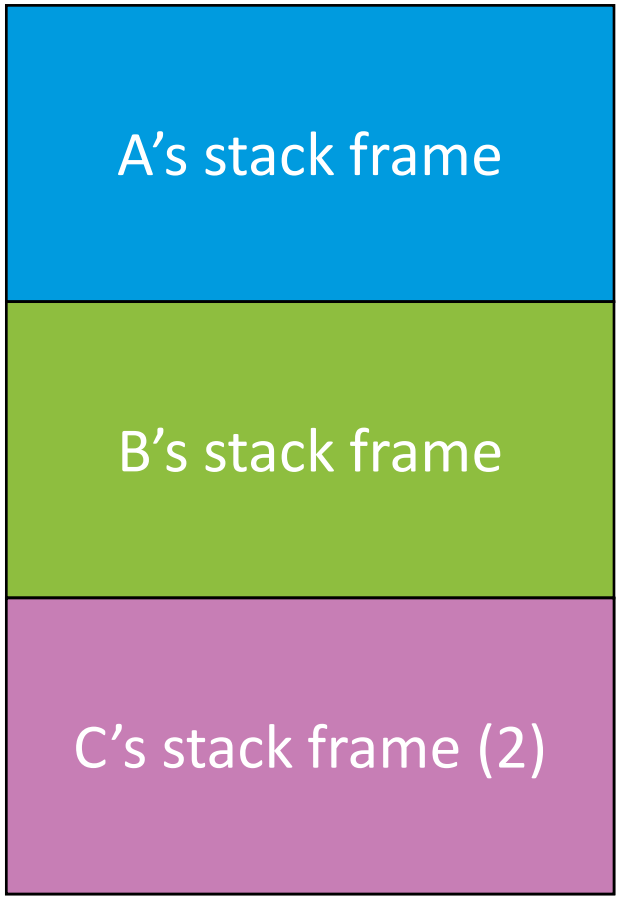
- Let's do another quick exercise:
  - How much larger is 1 GiB than 1 GB?
  - Use the command `ls -l` on your computer to see each file in the directory along with their sizes in bytes.
  - Now use `ls -lh` to see the files with "human-readable" sizes. Is this using KiB/MiB/GiB or KB/MB/GB?

Stack Variables and Return Values



Stack Variables and Return Values

- Each function call has its own stack frame.
- A stack frame tracks:
  - Local variables
  - Parameters to pass to other functions
  - Other temporary space
- Function call: push a new frame onto the stack
- Function return: pop a frame from the stack



Stack Variables and Return Values

- Returning from a function means the data in its stack frame isn't accessible anymore.
- The return value is passed in a special register, `%rax` (or `%eax`).
  - We haven't talked about registers yet, but they're small pieces of memory that the CPU uses to execute machine instructions.

Stack Variables and Return Values

- Write and try to use the function below in a program. What happens?

```
int *add(int x, int y) {
  int z = x + y;
  return &z;
}
```

Stack Variables and Return Values

- The stack frame is gone once add returns. Thus, so is z.

```
int* add(int x, int y) {
  int z = x + y;
  return &z;
}
```

Stack Variables and Return Values

- This is why you typically pass pointers into functions:

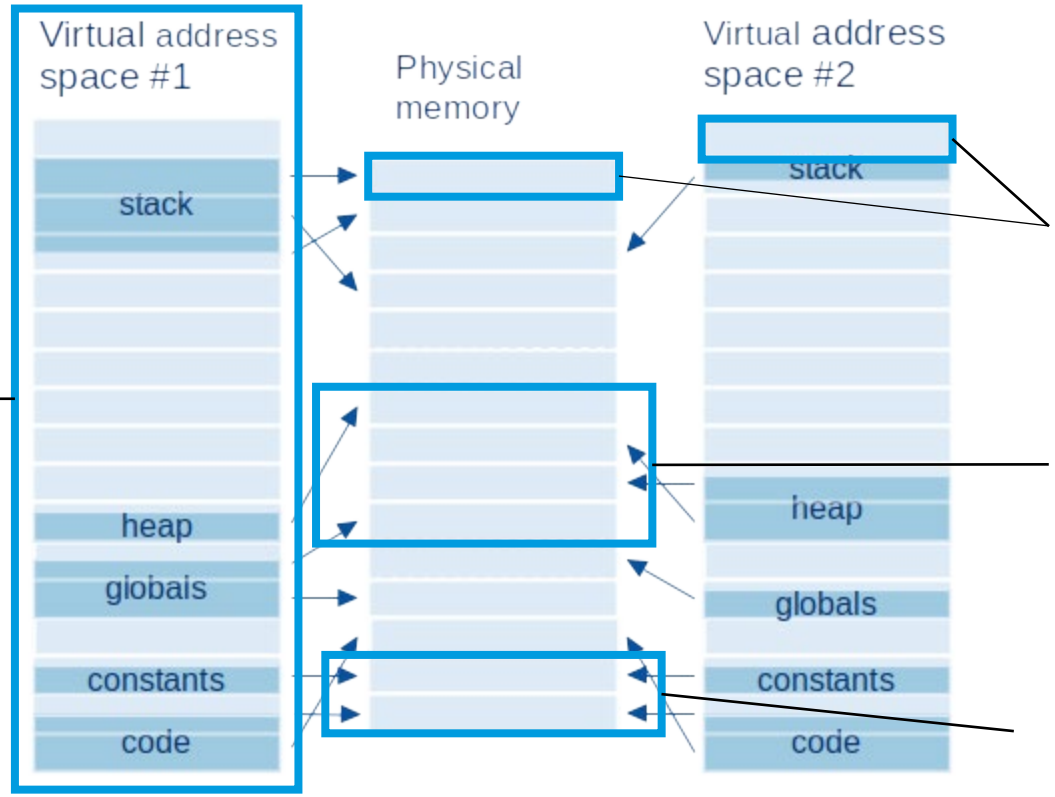
```
void add(int x, int y, int *z) {
  *z = x + y;
}

int main() {
  int z;
  add(42, 47, &z);
  return 0;
}
```

Memory Management

- Virtual memory is an abstraction of physical memory.
- Physical memory:
  - Usually, people mean RAM.
  - In this context, we mean disk space.
  - But RAM plays a role in accessing physical memory.

Memory Management

Every process has its own virtual address space.



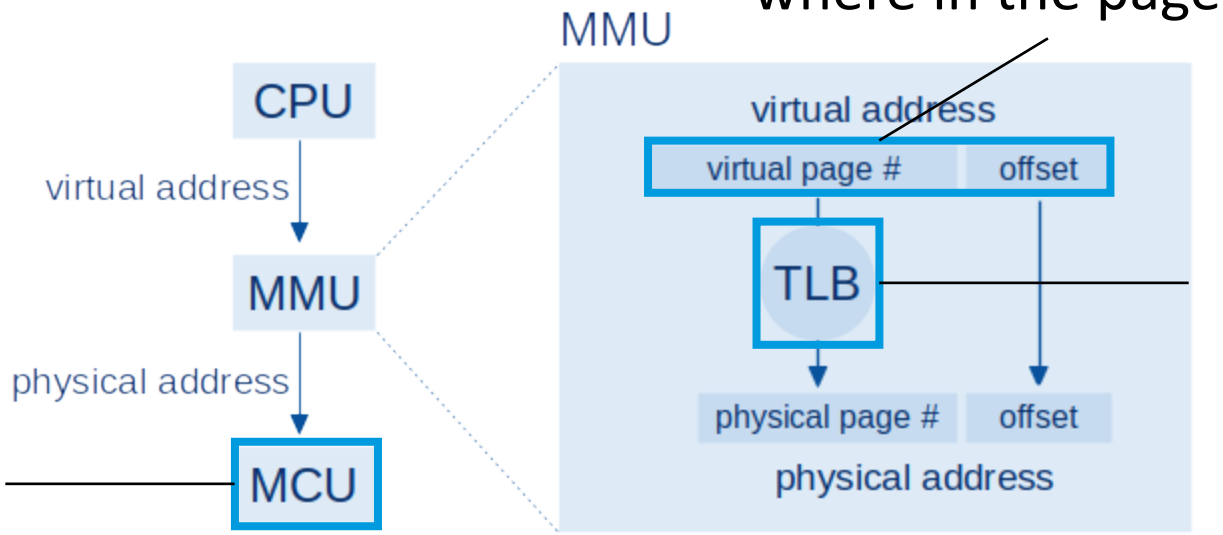
Memory is divided into pages.

Isolation: can't read memory of other processes.

Sharing: can map virtual pages to same physical page.

Memory Management

Page number is translated; offset is where in the page it is.



Look up the corresponding physical page in the kernel's page table.

Get the ~~Infinity Stones~~ actual data from physical memory.

Memory Management: Exercise

- How large is the page size on your computer? On Linux, you can do this with `getconf PAGE_SIZE` to see the result in bytes.
- How much disk space does your machine have? You can use the `df` command to see what disks you are using and `sudo fdisk -l` to see how much space that disk actually has.
- Write these values up on the board.

Memory Management: Exercise

- How many bits will it take to store a physical address (somewhere in your disk space)? Remember, you can only refer to bytes, not bits.
- How many pages will fit into your physical memory?

Memory Management: Exercise

- Check whether your system is 32 or 64 bits with `uname -m`. (x86 is 32 bit, x86_64 is 64 bit).
- This means you have 2^32 or 2^64 addresses in virtual memory.
- How many pages will fit into virtual memory?

Memory Management: Exercise

- Run `lscpu` on your machine.
- There is a line that indicates how many bits are used for a physical and virtual address on your machine. Is it what you expect?

Project 1: Learning C

- Project 1 is an opportunity to deepen your knowledge of one area or application of C.
- You have to build either a reasonably sized piece of software or an educational activity in C.
- Software examples:
  - Implement a small game emulator or virtual machine.
  - Write a simple text editor, shell, or HTTP server.
  - Implement a small part of the compilation process.
- Educational examples:
  - Write a tutorial or assignment designed to teach student what all of the different operators do in C.
  - Try out and teach a different unit testing framework in C.
  - Create at least five practice problems for students at this course level to do.

Project 1: Learning C

- Project 1 **must** be done in a team of 3-4 students.
- It will last just over a month.
- Ask on Discord to discuss ideas or to find partners.
- Proposals are due in a little over a week, so think ahead to then!
- Consider:
  - What do you want to make?
  - What do you want to learn?
  - Who will do what?
  - What resources will you need?