

Liana + Anna Lecture!

Heavily edited photo of Margaret Hamilton, 1969



Endgame

Assignment 7 due on Tuesday

The last deadline for re-submits is Thursday

11:59 PM

Project labs are this week and next week

Today

Final project process

Where to go from 107E

Embedded systems route (EE 108, etc)

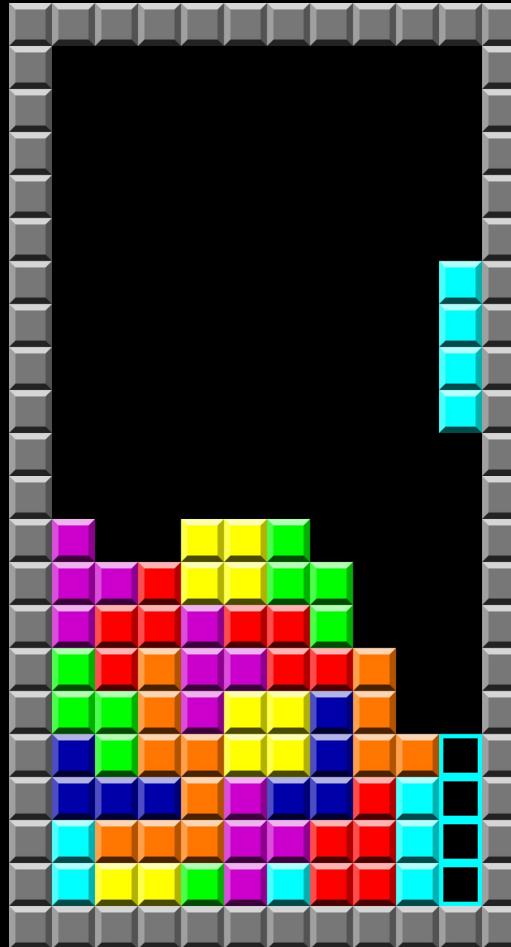
Operating systems route (CS 110)

Final Projects

HOLGRAMSORBUST

HOLGRAMSORBUST

HOLGRAMSORBUST



Pepper's ghost

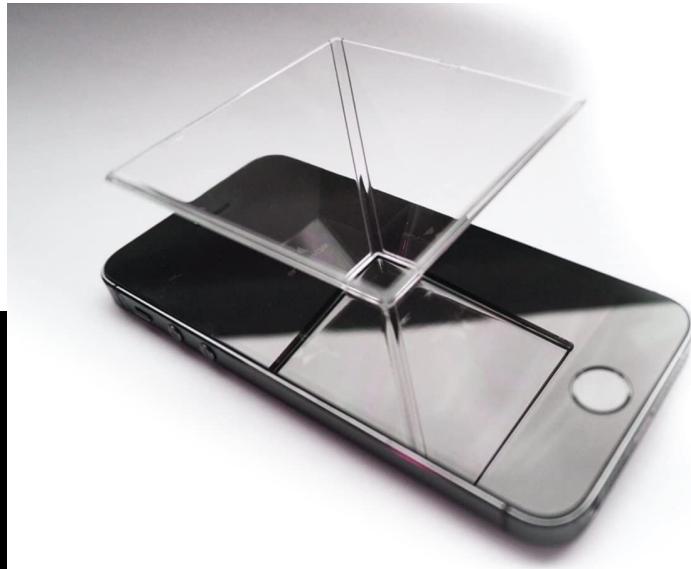
From Wikipedia, the free encyclopedia

For other uses, see [Pepper's ghost \(disambiguation\)](#).

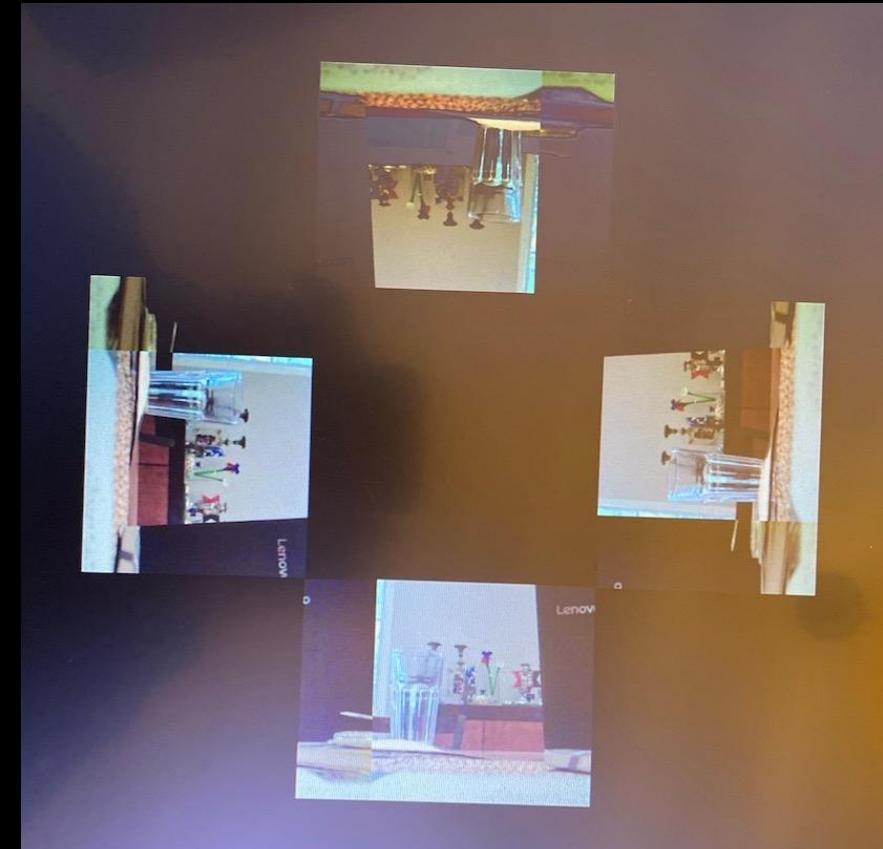
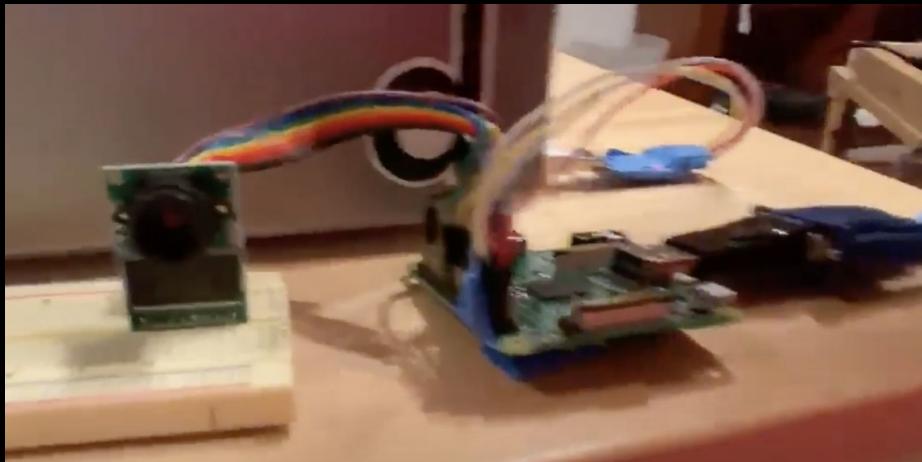
Pepper's ghost is an [illusion](#) technique used in the [theatre](#), [amusement parks](#), [museums](#), [television](#), and [concerts](#). It is named after the English scientist [John Henry Pepper](#) (1821–1900) who began popularizing the effect with a demonstration in 1862.^[1] Examples of the illusion are the Girl-to-Gorilla trick found in old carnival [sideshows](#) and the appearance of "Ghosts" at the [Haunted Mansion](#) and the "Blue Fairy" in [Pinocchio's Daring Journey](#), both at the [Disneyland](#) park in California. [Teleprompters](#) are a modern implementation of Pepper's ghost. The technique was used by Digital Domain for the appearance of [Tupac Shakur](#) onstage with [Dr. Dre](#) and [Snoop Dogg](#) at the 2012 Coachella Music and Arts Festival and [Michael Jackson](#) at the 2014 [Billboard Music Awards](#).

Contents [hide]

- [1 Historic Pepper's ghost effects](#)
- [2 Contemporary technique](#)
- [3 History](#)
 - [3.1 Giambattista della Porta](#)
 - [3.2 Henri Robin and Pierre Séguin](#)
 - [3.3 John Pepper and Henry Dircks](#)
- [4 Modern examples](#)
 - [4.1 Amusement parks](#)
 - [4.2 Museums](#)
 - [4.3 Television, film, and video](#)



Stage setup for Pepper's Ghost. A brightly-lit figure out of the audience's sight below the stage is reflected in a pane of glass placed between the performer and the audience. To the audience, it appears as if the ghost is on stage.





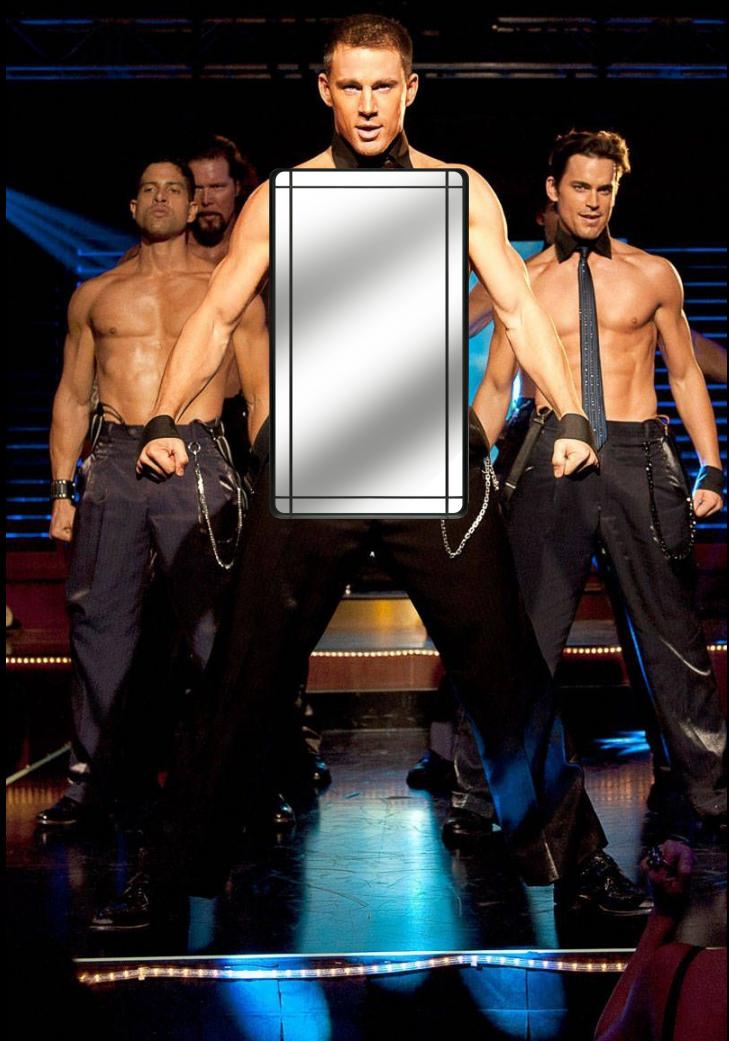
Step #1:

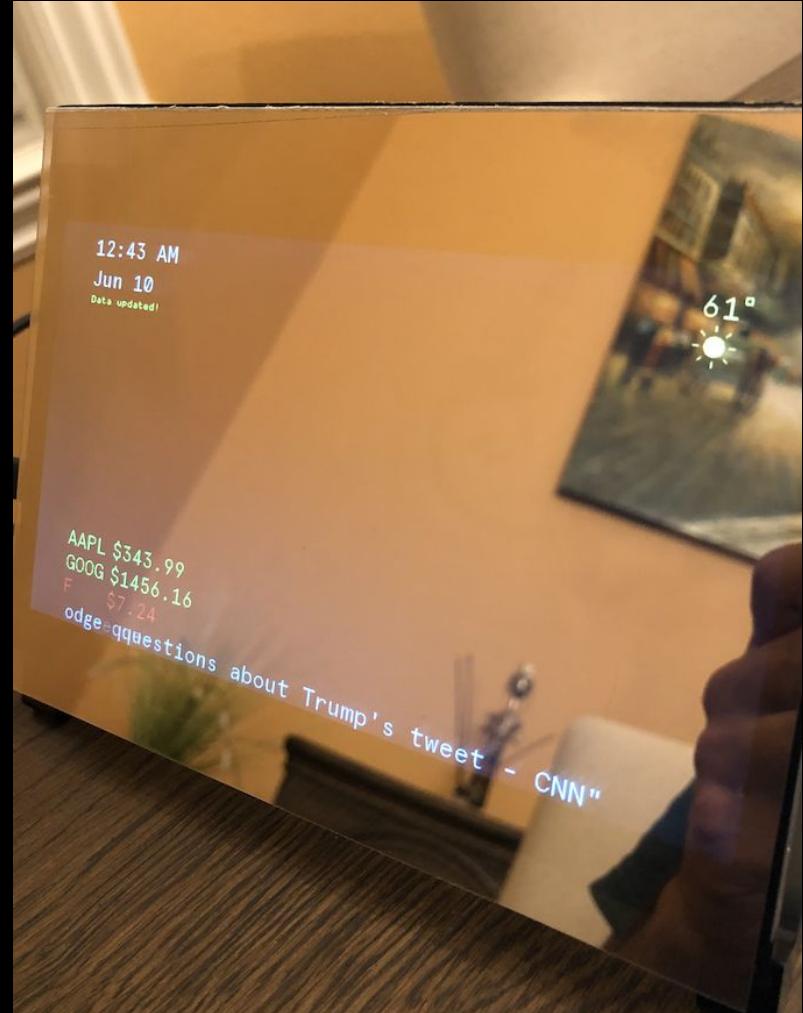
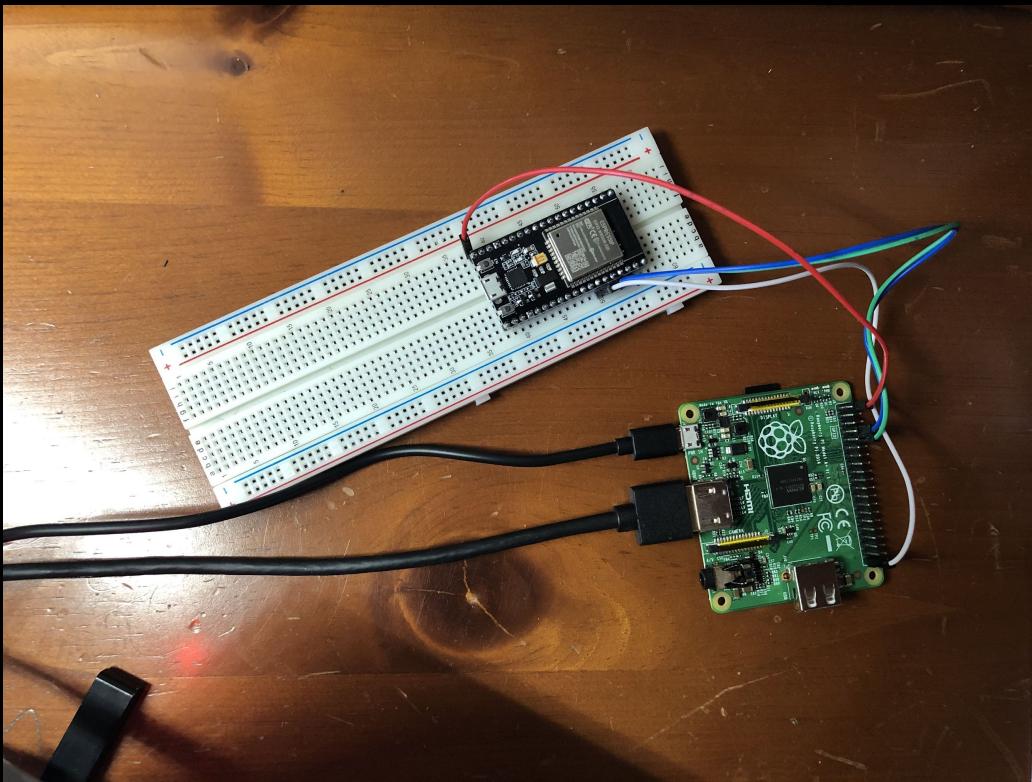
Choose a really easy base project

Step #2

Once you've built the base, you can add more features

magic mirror





project roadmap:

Networking:

- [x] Communication between ESP32 and WiFi
- [x] Communication from ESP32 to Raspberry Pi
- [x] Parsing serial data to internal data

Graphics

- [x] Extend graphics library for modules:

- [x] Date and Time (we should configure this under the hood and let ARM timer do the work OR we can write a proxy server)

- [x] Weather ([Weather API])
 - [x] News Headlines ([News API])
 - [x] Custom Font (could be cool if time)

build off of the existing codebase!!!

```
lkeesing@DESKTOP-7A18ME4: ~ + < - □ ×
File Edit Options Buffers Tools C Help
1#include "console.h"
2#include "gl.h"
3#include "fb.h"
4#include "gl_extra.h"
5#include "weather.h"
6#include "printf.h"
7#include "malloc.h"
8#include "timer.h"
9#include "strings.h"
10#include <stdarg.h>
11
12void console_init(unsigned int width, unsigned int height)
13{
14    // initialize cursor
15    gl_init(width, height, GL_DOUBLEBUFFER);
16    gl_extra_init();
17    gl_clear(GL_BLACK);
18    gl_draw_weather_icon(windy, 0, 0, GL_WHITE);
19    draw_string(100, 100, "hi my name is bob and I work at the\
      button factory", GL_WHITE);
20    gl_swap_buffer();
21    timer_delay(100);|
22
23}
24
25void console_clear(void)
26{
27    gl_clear(GL_BLACK);
28    gl_swap_buffer();
29    gl_clear(GL_BLACK);
30}
```



(the one where liana & anna compare EE & CS)

E•M•B•E•D•D•E•D
S•Y•S•T•E•M•S

"WHEN THEY GO HIGH,
WE GO LOW. -MICHELLE"

- LIANA KEESiNG



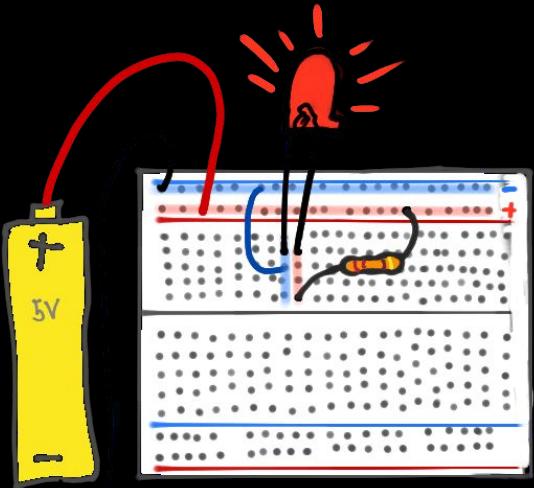
CS107e Week 1

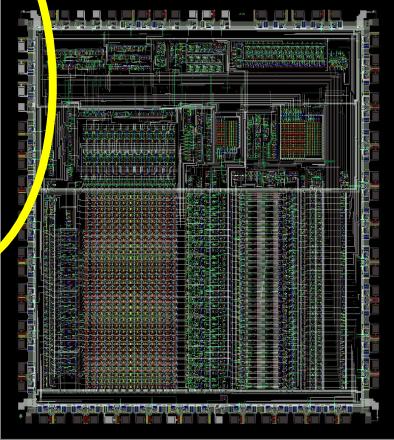
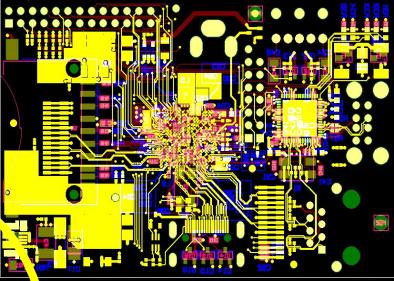
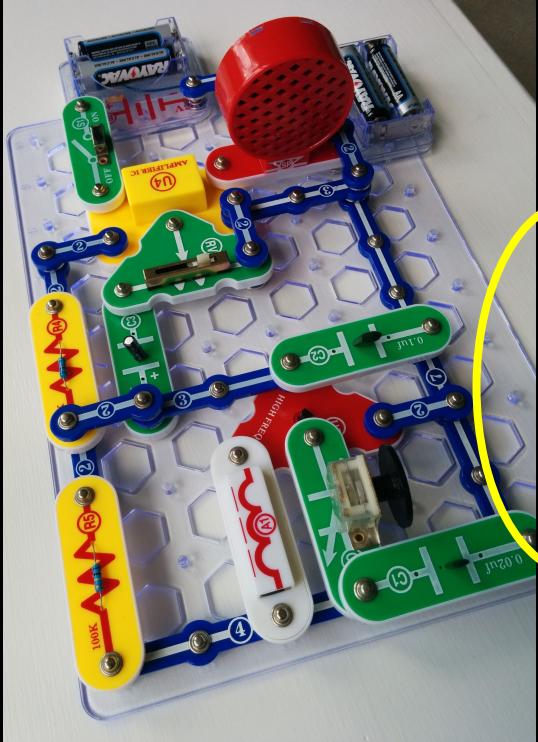
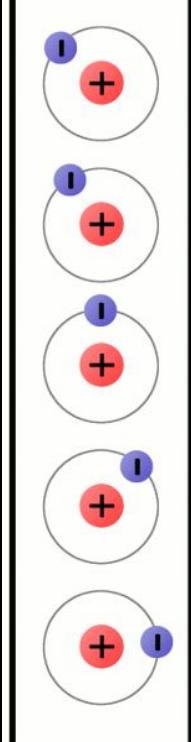
CS107e is Coming Soon! Sign Up.

Announcements

- We will post an

Electrical Engineering





```
lkeesing@DESKTOP-7A1 ~ + - x
lkeesing@DESKTOP-7A18ME4:~/cs107e/cs107e_home/assignments/src/lib$ cat
interrupts_asm.s
@ Assembly code for interrupt vector table and functions
@ to enable/disable IRQ interrupts
on the Raspberry Pi in CS107E.
@
@ Author: Philip Levis
@ Author: Julie Zelenski
@ Last update: 2/20/20

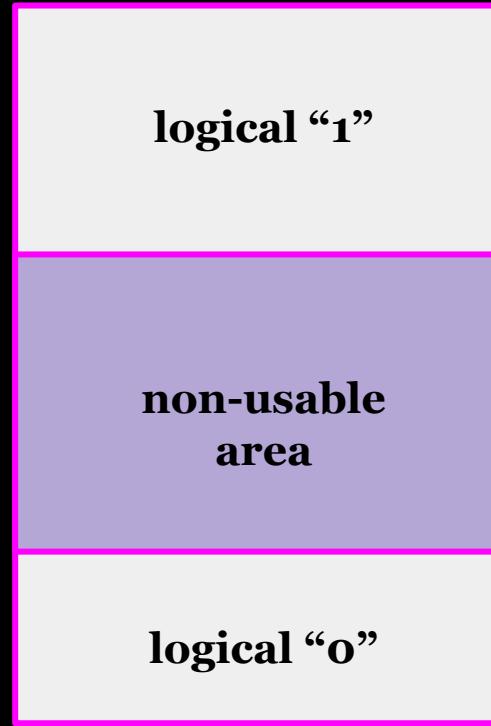
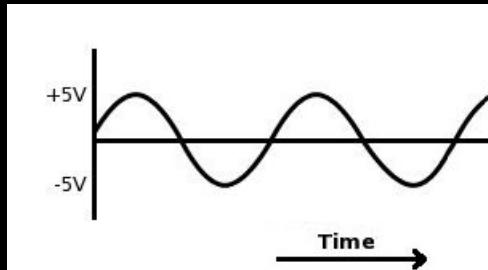
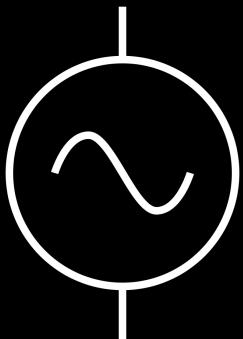
@ Enable/disable interrupts.

@ CPSR = current program status register
lower 8 bits are:
    7 6 5 4 3 2 1 0
    +---+-----+
    |I|F|T|   Mode |
    +---+-----+
@ I : = 0 IRQ enabled, = 1 IRQ disabled
@ F : = 0 FIQ enabled, = 1 FIQ disabled
@ T : = 0 indicates ARM execution, = 1 is thumb execution
@ Mode : current mode
```

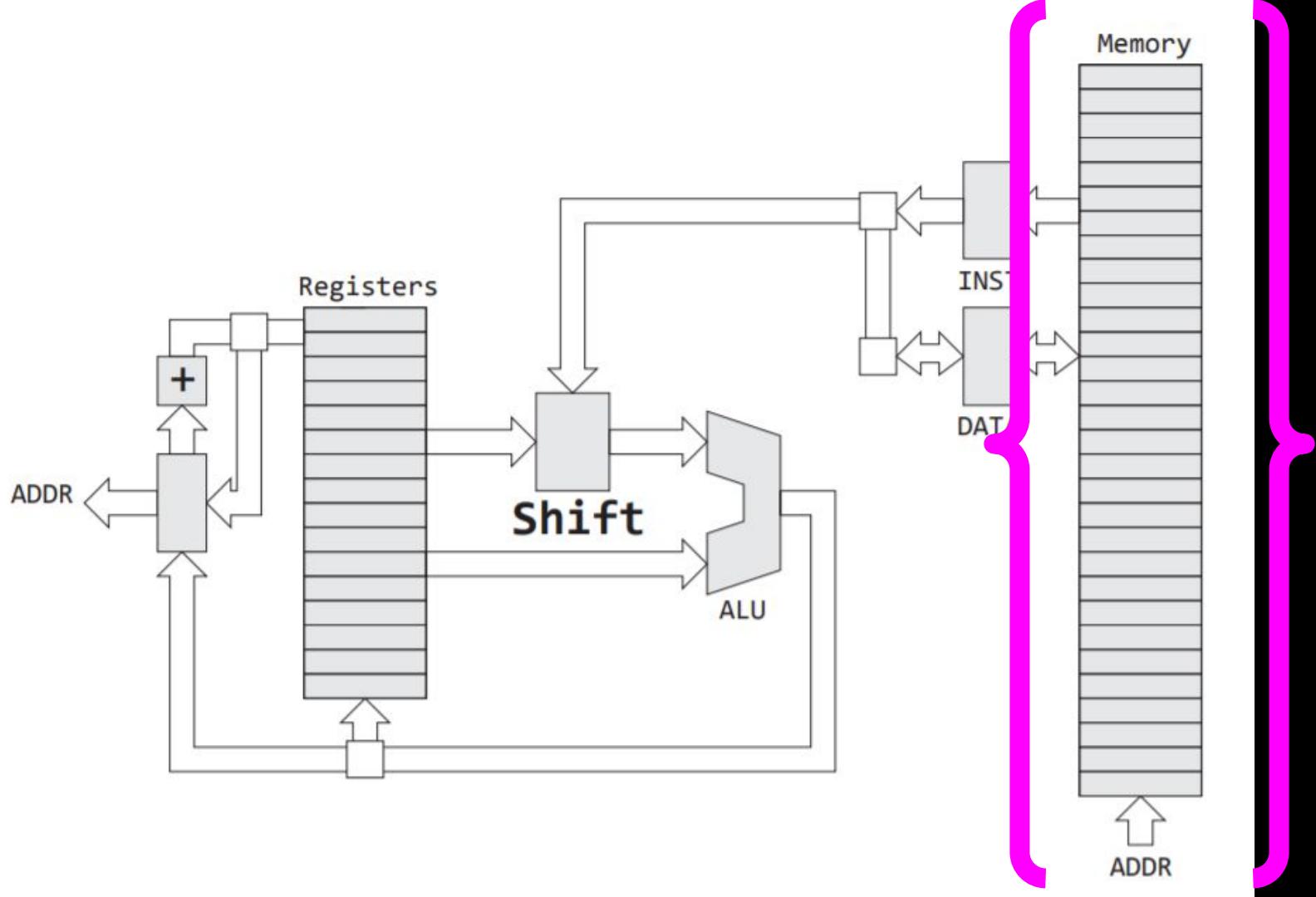
physics
(43 or 63)

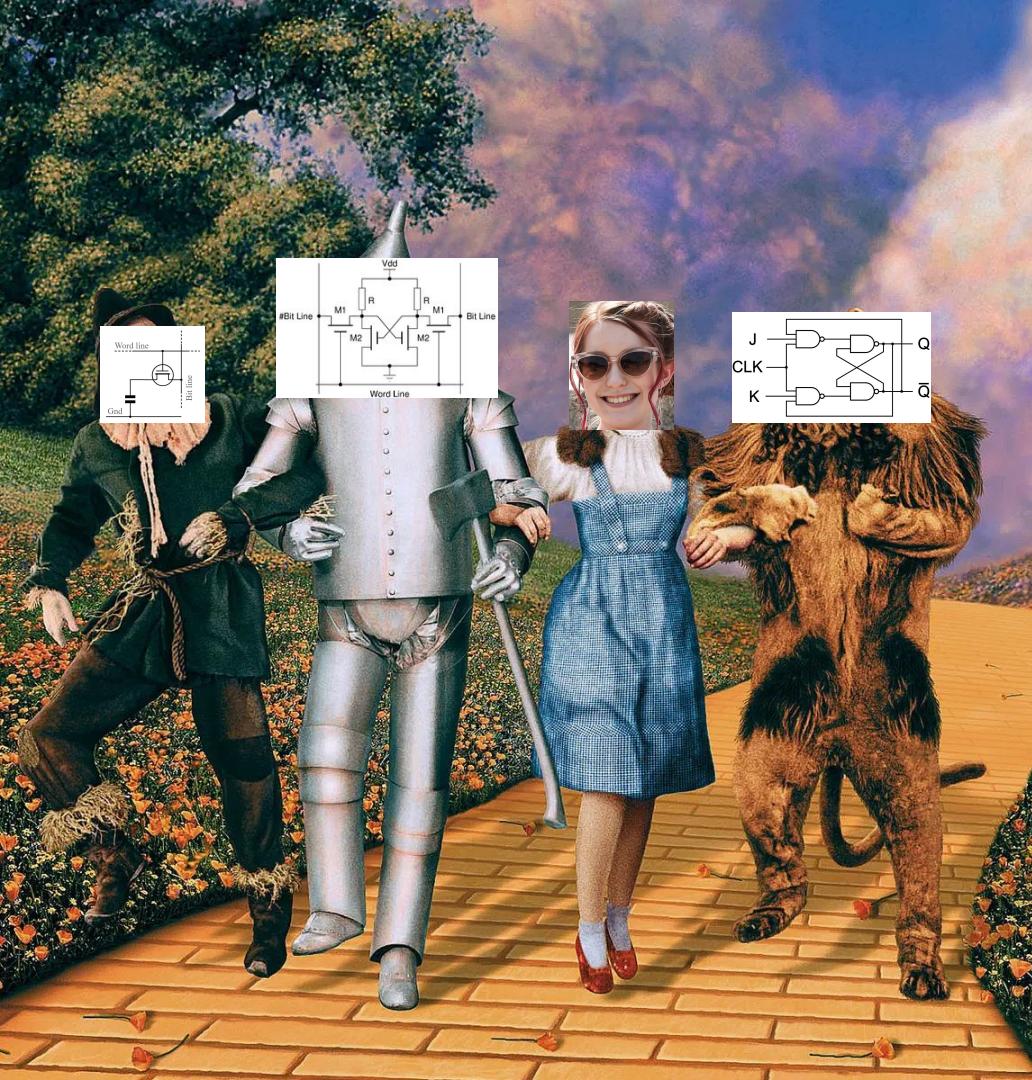
ee
(101a → 108)

ee/cs
(ee18o, cs14oe)



what does “digital” even mean?





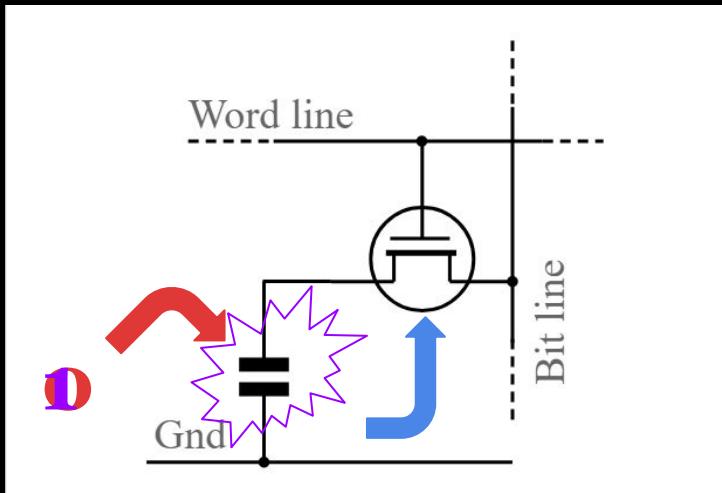
from voltage
differentials
to memory

or,

DRAM and SRAM
and flip-flops,
oh my!

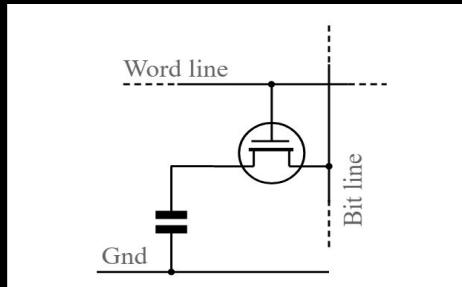
RPi memory is

**Dynamic Random
Access Memory**
transistor + capacitor

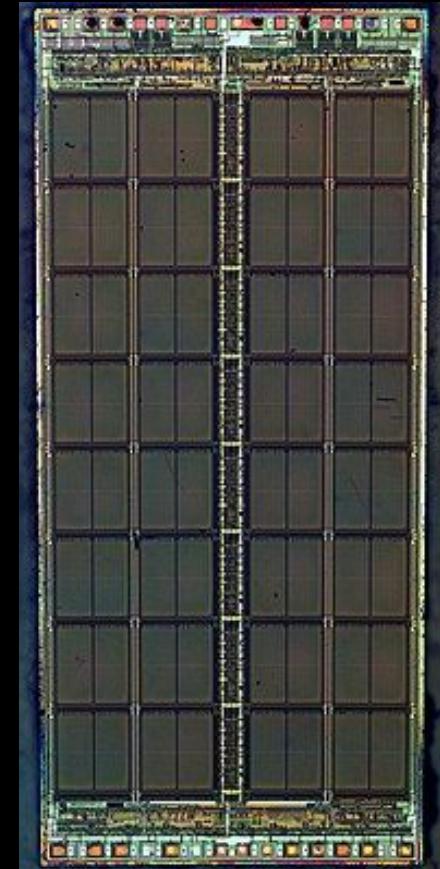
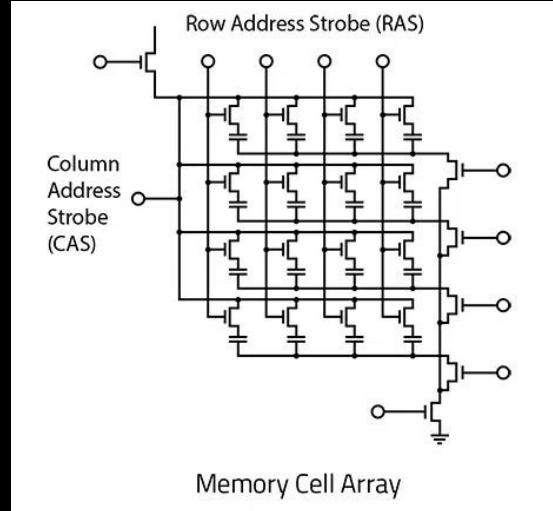


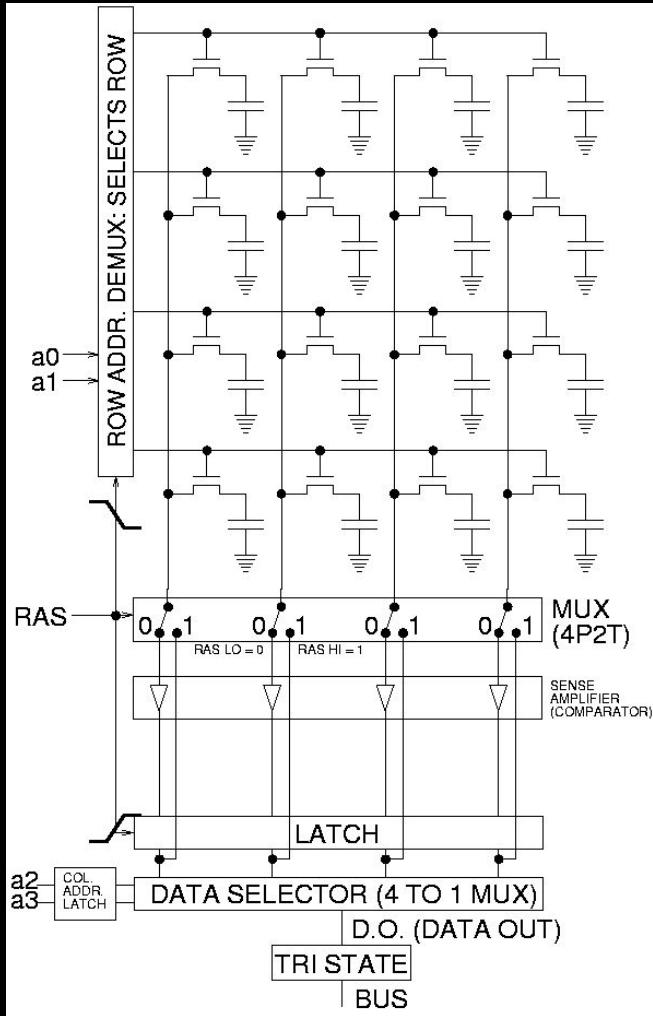
DRAM!

memory cell

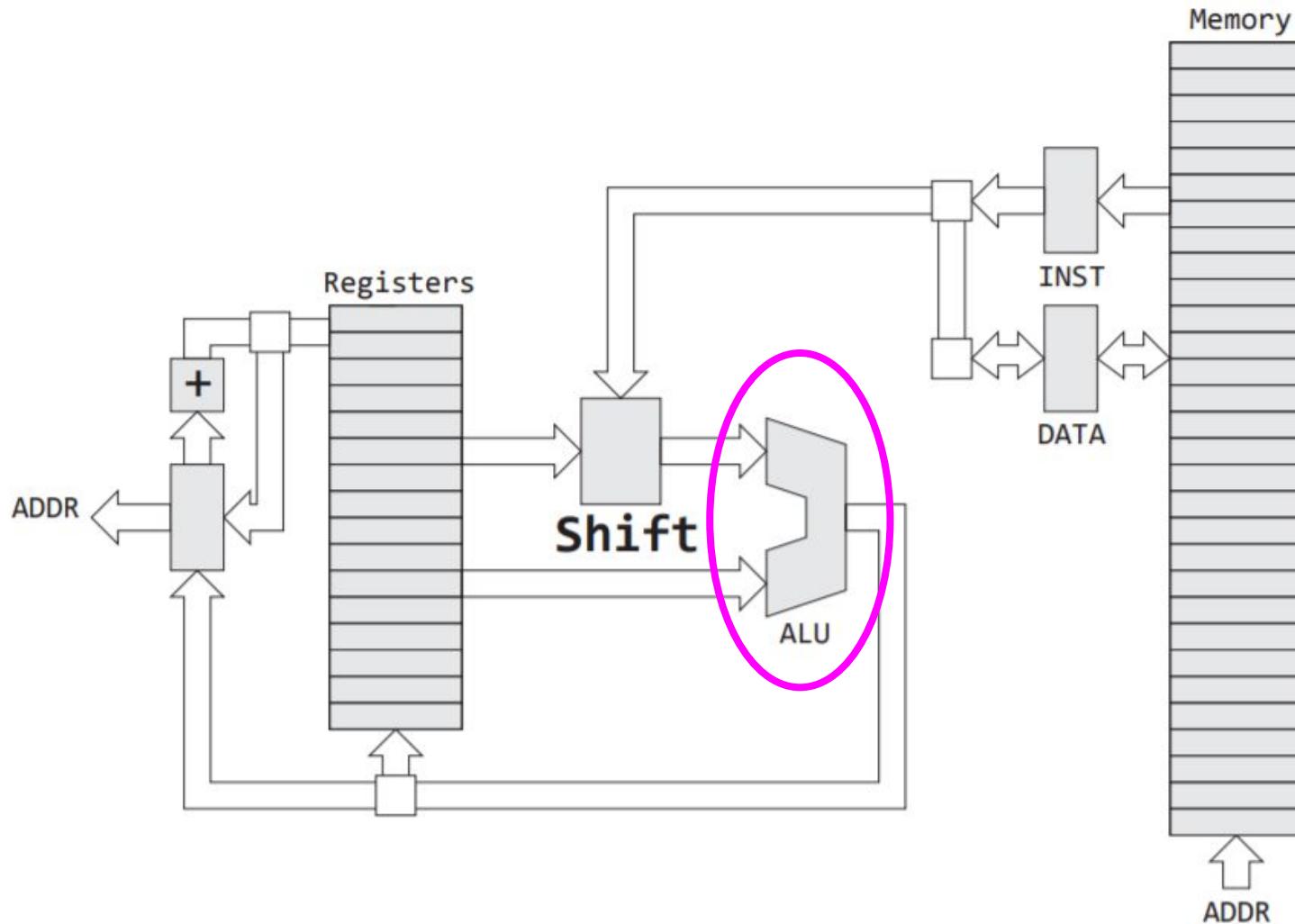


memory cell array



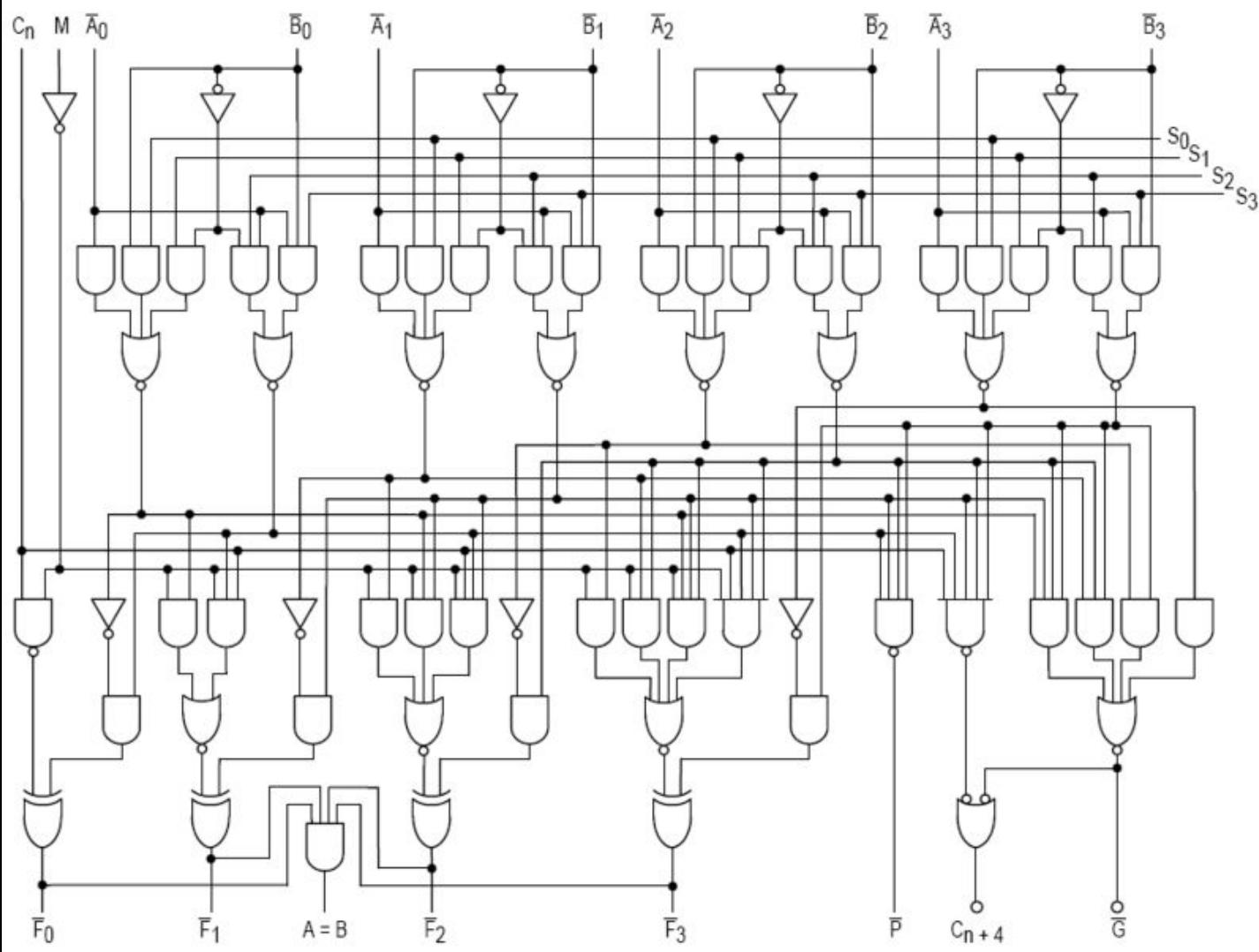


ok actually we cheated here and added a multiplexer and some latches but the basic structure remains



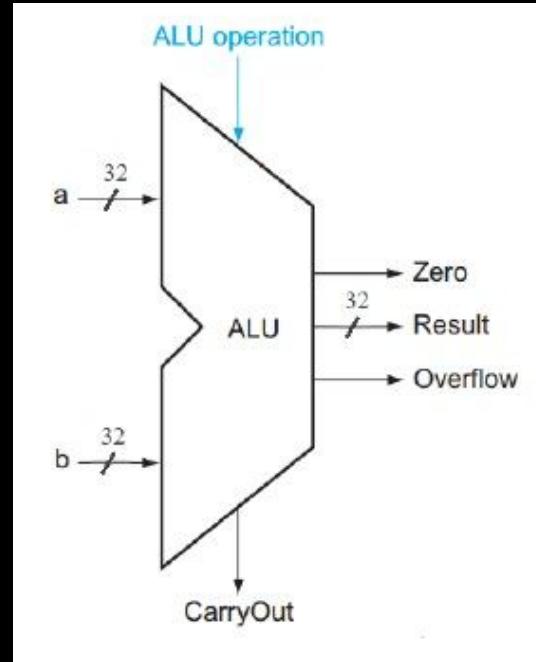


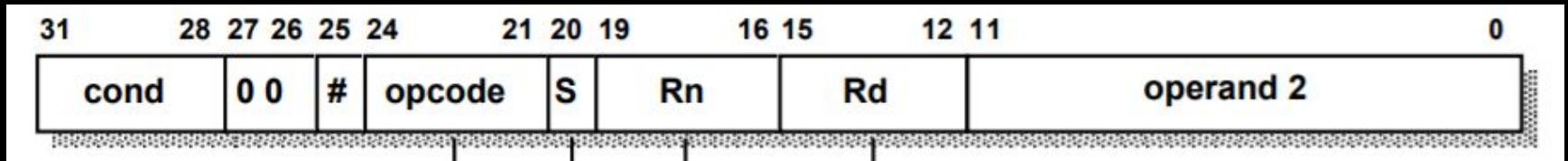
...the ALU isn't magic,
it's just a fancy circuit



(ok that kind of looks like magic.)

but it's really just a digital logic component:





| ALU | Data | Control |
|--------|-----------------------------------|---------|
| input | operands (contents of Rn and op2) | opcode |
| output | result (into Rd) | status |

an HDL like Verilog can explain what the ALU should do!

```
module alu(
    input [31 : 0] A, B, // ALU 8 - BIT Inputs
    input [3 : 0] opcode, // ALU Selection
    input cIn,
    output [31 : 0] ALU_Out, // ALU 8 - BIT
    Output status,
    output CarryOut // Carry OUT Flag
);
    reg [31 : 0] ALU_Result;
    wire [8 : 0] tmp;
    assign ALU_Out = ALU_Result; // ALU OUT
    assign tmp = {1'b0, A} + {1'b0, B};
    assign CarryOut = tmp[8]; // Carryout flag
    always @( *)
        BEGIN
            CASE(opcode)
                4'b0000 : // Logical AND
                    ALU_Result = A & B;
                4'b0001 : // Logical XOR
                    ALU_Result = A ^ B;
                4'b0010 : // Subtraction
                    ALU_Result = A - B;

```

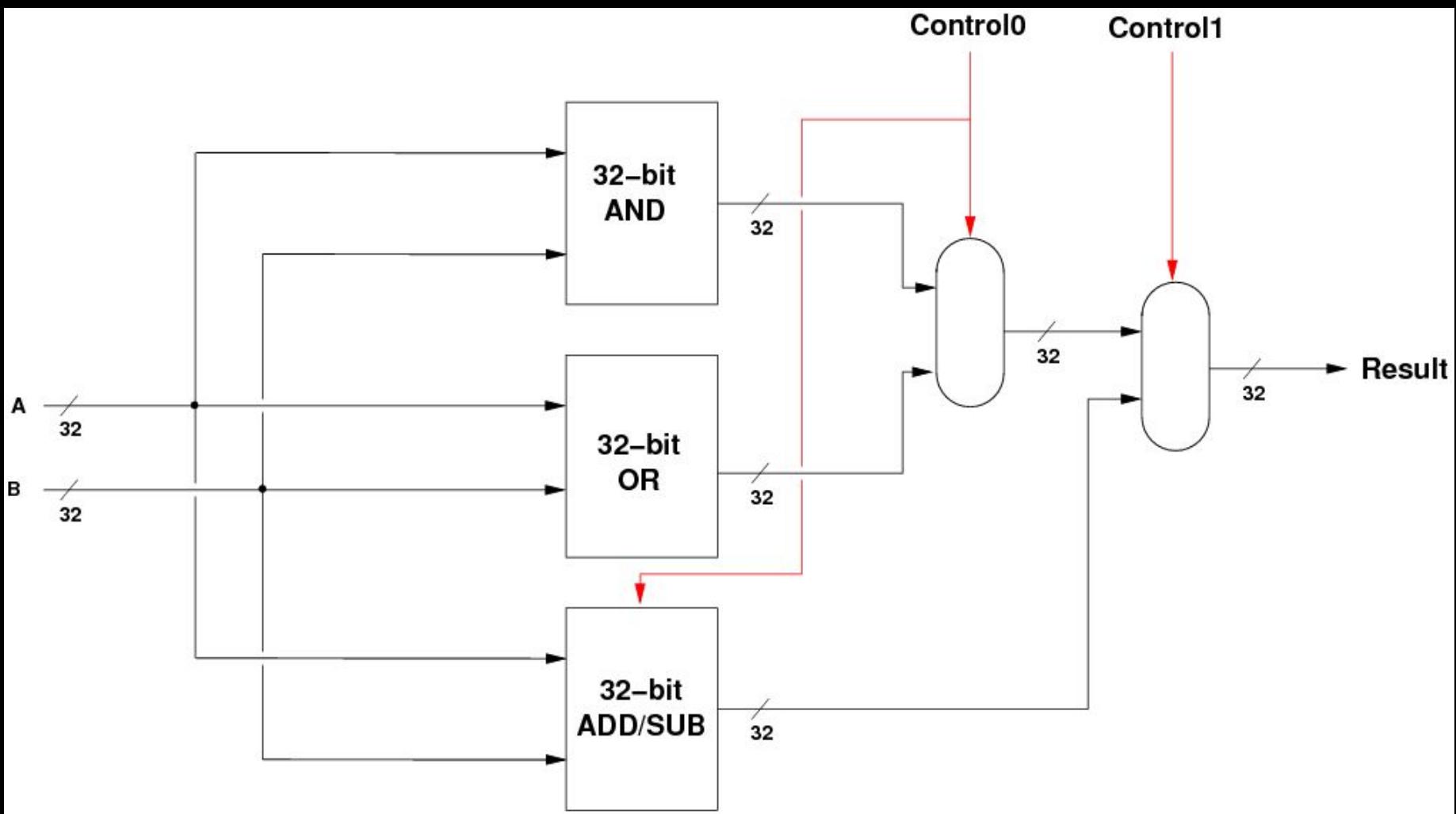
```
                4'b0011 : // Reverse Subtraction
                    ALU_Result = B - A;
                4'b0100 : // Addition
                    ALU_Result = A + B;
                4'b0101 : // Addition with carry
                    ALU_Result = A + B + cIn;
                4'b0110 : // Subtract with carry
                    ALU_Result = A - B + (1 - cIn);
                4'b0111 : // RSC...
                4'b1000 : // TST...
                4'b1001 : // TEQ...
                4'b1010 : // CMP...
                4'b1011 : // CMN...
                4'b1100 : // ORR...
                4'b1101 : // MOV...
                4'b1110 : // BIC...
                4'b1111 : // MVN...
            DEFAULT : ALU_Result = A + B;
        endcase
    END
endmodule
```

| Opcode | Mnemonic | Operation |
|--------|----------|-----------------------------|
| 0000 | AND | Logical AND |
| 0001 | EOR | Logical Exclusive OR |
| 0010 | SUB | Subtract |
| 0011 | RSB | Reverse Subtract |
| 0100 | ADD | Add |
| 0101 | ADC | Add with Carry |
| 0110 | SBC | Subtract with Carry |
| 0111 | RSC | Reverse Subtract with Carry |
| 1000 | TST | Test |
| 1001 | TEQ | Test Equivalence |
| 1010 | CMP | Compare |
| 1011 | CMN | Compare Negated |
| 1100 | ORR | Logical (inclusive) OR |
| 1101 | MOV | Move |
| 1110 | BIC | Bit Clear |
| 1111 | MVN | Move Not |

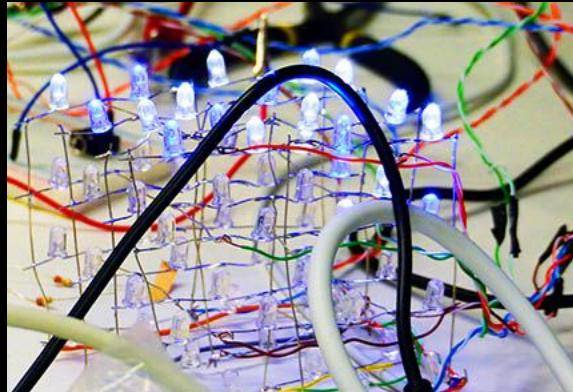
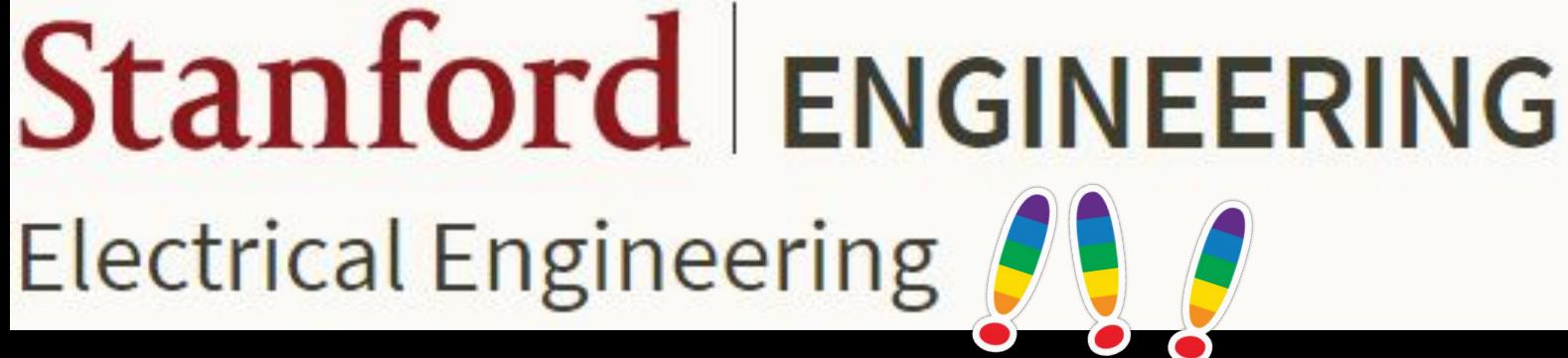
an HDL like Verilog can explain what the ALU should do!

```
module alu(
    input [31 : 0] A, B, // ALU 8 - BIT Inputs
    input [3 : 0] opcode, // ALU Selection
    input cIn,
    output [31 : 0] ALU_Out, // ALU 8 - BIT
    Output status,
    output CarryOut // Carry OUT Flag
);
    reg [31 : 0] ALU_Result;
    wire [8 : 0] tmp;
    assign ALU_Out = ALU_Result; // ALU OUT
    assign tmp = {1'b0, A} + {1'b0, B};
    assign CarryOut = tmp[8]; // Carryout flag
    always @( *)
        BEGIN
            CASE (opcode)
                4'b0000 : // Logical AND
                    ALU Result = A & B;
                4'b0001 : // Logical XOR
                    ALU Result = A ^ B;
                4'b0010 : // Subtraction
                    ALU_Result = A - B;
```

```
        4'b0011 : // Reverse Subtraction
        ALU Result = B - A;
        4'b0100 : // Addition
        ALU_Result = A + B;
        4'b0101 : // Addition with carry
        ALU Result = A + B + cIn
        4'b0110 : // Subtract with carry
        ALU Result = A - B + (1 - cIn);
        4'b0111 : // RSC...
        4'b1000 : // TST...
        4'b1001 : // TEQ...
        4'b1010 : // CMP...
        4'b1011 : // CMN...
        4'b1100 : // ORR...
        4'b1101 : // MOV...
        4'b1110 : // BIC...
        4'b1111 : // MVN...
        DEFAULT : ALU_Result = A + B;
    endcase
    END
endmodule
```



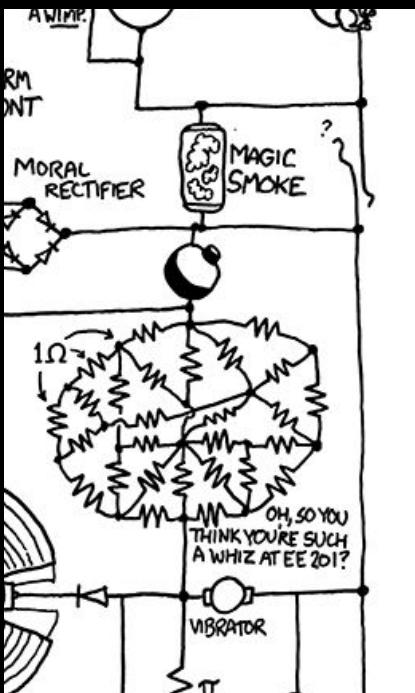
think this stuff is cool?



classes:

EE 108: Digital System Design

Digital circuit, logic, and system design. Digital representation of information. CMOS logic circuits. Combinational logic design. Logic building blocks, idioms, and structured design. Sequential logic design and timing analysis. Clocks and synchronization. Finite state machines. Microcode control. Digital system design. Control and datapath partitioning. Lab. *In



EE 180: Digital Systems Architecture

The design of processor-based digital systems. Instruction sets, addressing modes, data types. Assembly language programming, low-level data structures, introduction to operating systems and compilers. Processor microarchitecture, microprogramming, pipelining. Memory systems and caches. Input/output, interrupts, buses and DMA. System design implementation alternatives, software/hardware tradeoffs. Labs involve the design of processor subsystems and processor-based embedded systems. Formerly EE 108B. Prerequisite: one of CS107 or CS 107E (required) and EE108 (recommended but



Also!

EE109: “Digital Systems Design Lab”

EE271: “Introduction to VLSI Systems”

EE 272A/B: “Design Projects in VLSI Systems I/II”

EE 273: “Digital Systems Engineering”

EE 282: “Computer Systems Architecture”



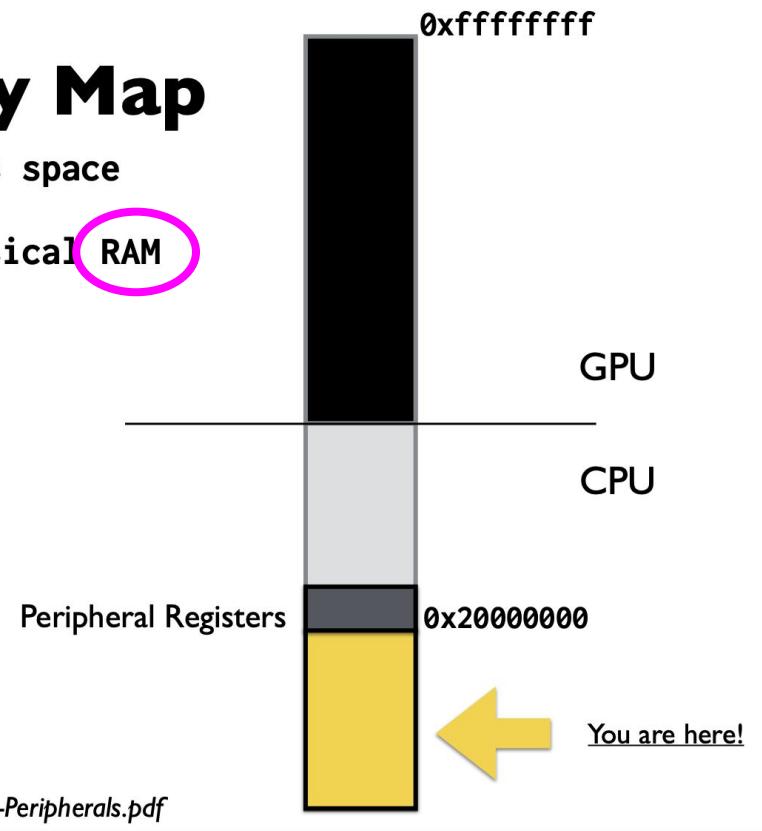
(the one where liana & anna compare EE & CS)

O.P.E.R.A.T.I.N.G S.Y.S.T.E.M.S

Memory Map

32-bit address space

512 MB of physical RAM



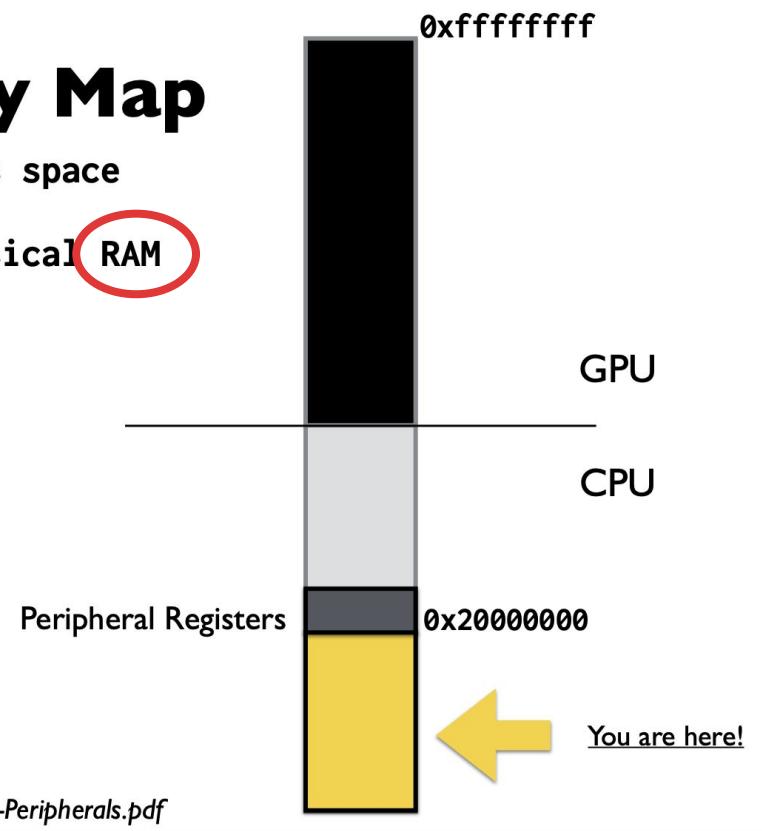
Ref: BCM2835-ARM-Peripherals.pdf



Memory Map

32-bit address space

512 MB of physical RAM



Ref: BCM2835-ARM-Peripherals.pdf

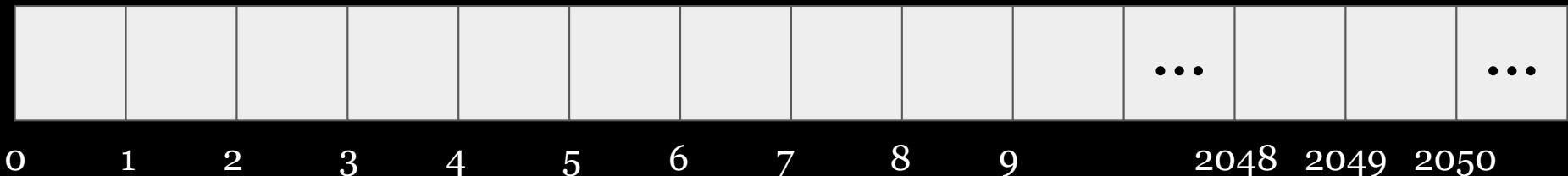
What about the
HARD DRIVE?



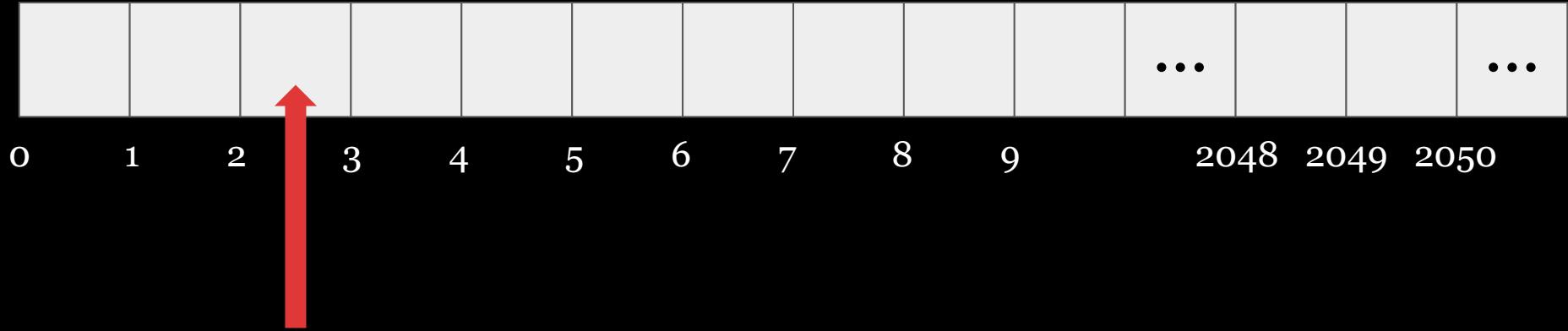
Why file systems are
the coolest thing you'll
ever see



The hard drive is not byte-addressable



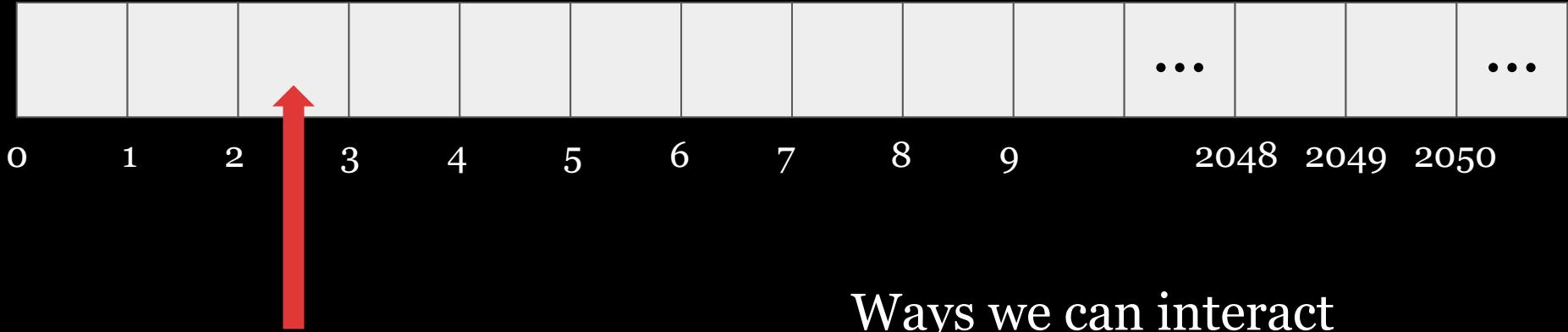
The hard drive is not byte-addressable



Each sector is
512 bytes*

* The size of a sector depends on the hardware, but here we'll assume a sector represents 512 bytes

The hard drive is not byte-addressable



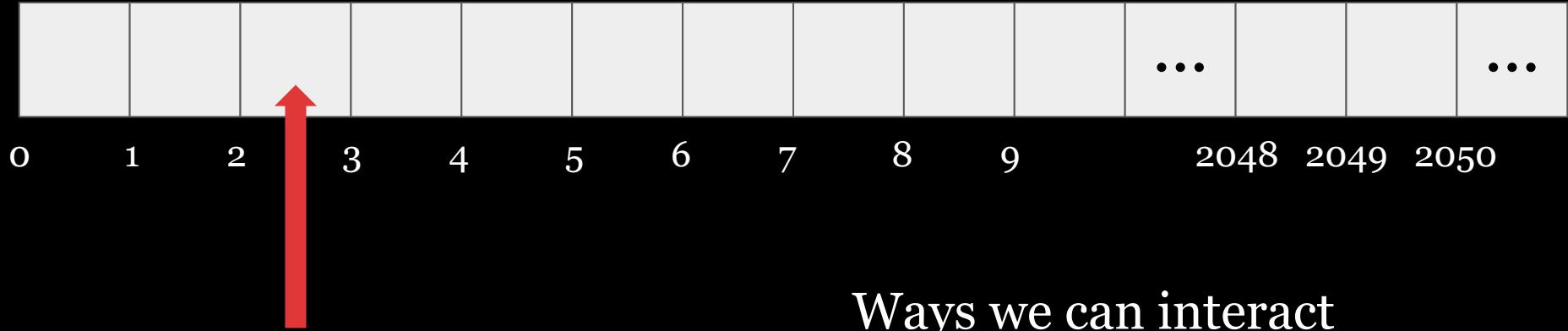
Each sector is
512 bytes*

Ways we can interact
with the hard drive:

readSector
writeSector

* The size of a sector depends on the hardware, but here we'll assume a sector represents 512 bytes

The hard drive is not byte-addressable



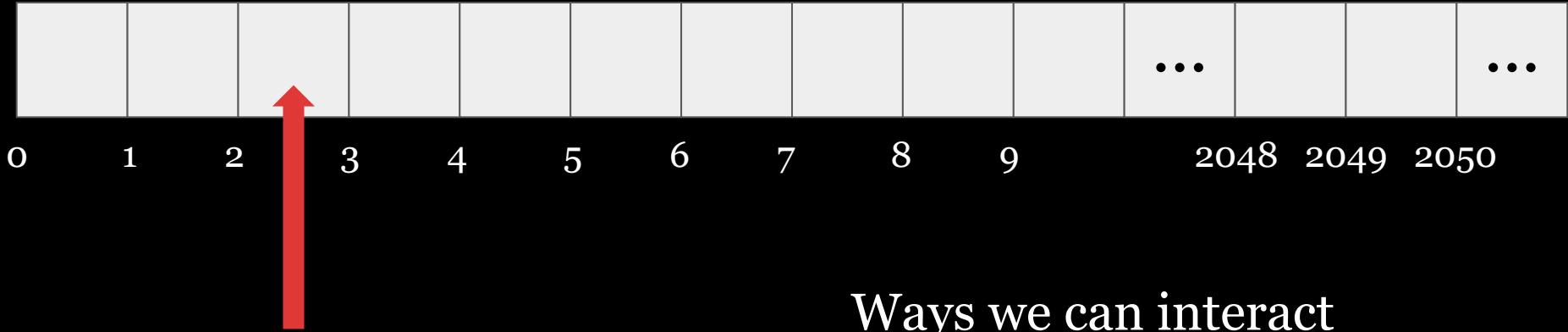
Each sector is
512 bytes*

Ways we can interact
with the hard drive:

`readSector(sectorNum, buf);`
`writeSector(sectorNum, buf);`

* The size of a sector depends on the hardware, but here we'll assume a sector represents 512 bytes

The hard drive is not byte-addressable



Each sector is
512 bytes*

Ways we can interact
with the hard drive:

`readSector(sectorNum, buf);`
`writeSector(sectorNum, buf);`
`char buf[512];`

* The size of a sector depends on the hardware, but here we'll assume a sector represents 512 bytes

So how do we store files
in these sectors?



printf.c (1024 bytes)

```
code code code code
code buf[256] lots
of code %s if else
va_list code code
code snprintf code
code code code code
```

printf.c (1024 bytes)

```
code code code code  
code buf[256] lots  
of code %s if else  
va_list code code  
code snprintf code  
code code code code
```

printf.c
size: 1024
file type: regular
permissions: ...

printf.c data

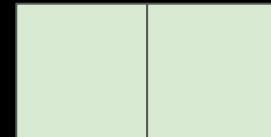


printf.c (1024 bytes)

```
code code code code
code buf[256] lots
of code %s if else
va_list code code
code snprintf code
code code code code
```

printf.c
size: 1024
file type: regular
permissions: ...

printf.c data



0 1 2 3 4 5 6 ... 2048 2049 2050 2051 2052 ...

printf.c (1024 bytes)

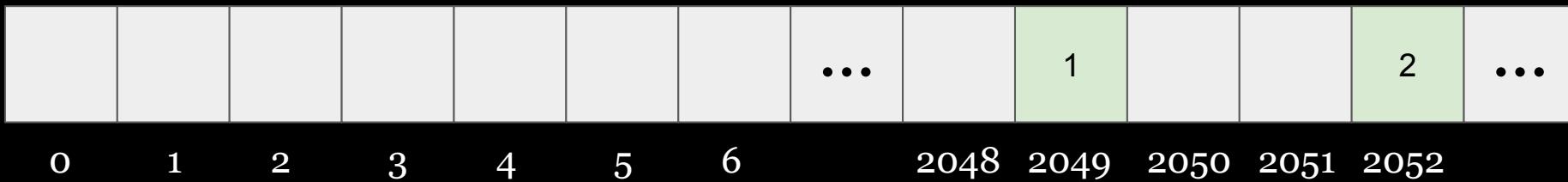
```
code code code code  
code buf[256] lots  
of code %s if else  
va_list code code  
code snprintf code  
code code code code
```

printf.c
size: 1024
file type: regular
permissions: ...

sectors:
2049, 2052

printf.c data

| | |
|---|---|
| 1 | 2 |
|---|---|



printf.c (1024 bytes)

```
code code code code
code buf[256] lots
of code %s if else
va_list code code
code snprintf code
code code code code
```

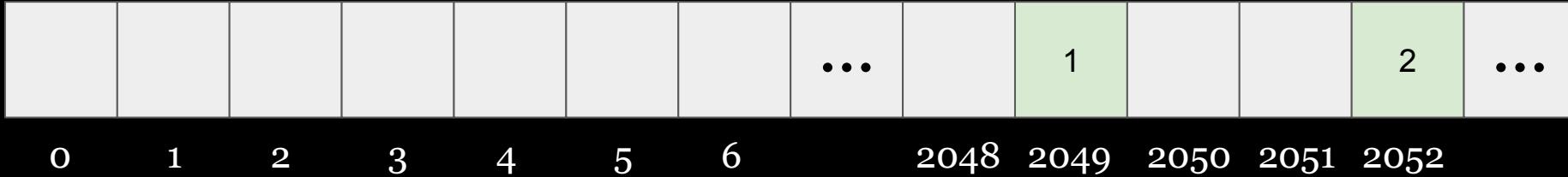
printf.c
size: 1024
file type: regular
permissions: ...

sectors:
2049, 2052

printf.c data

| | |
|---|---|
| 1 | 2 |
|---|---|

What if our file size isn't a
perfect multiple of 512?





inode

printf.c

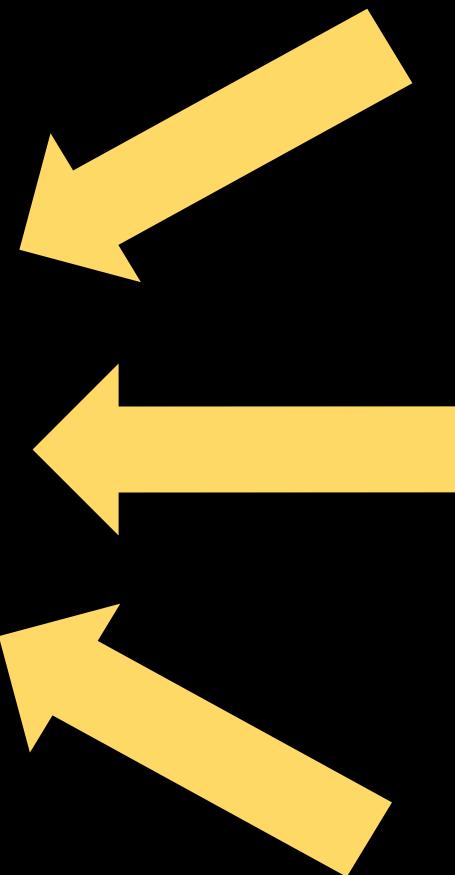
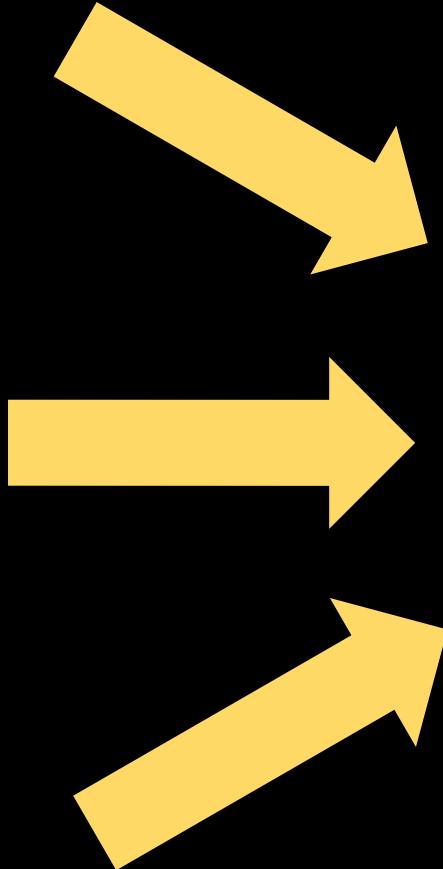
size: 1024

file type: regular

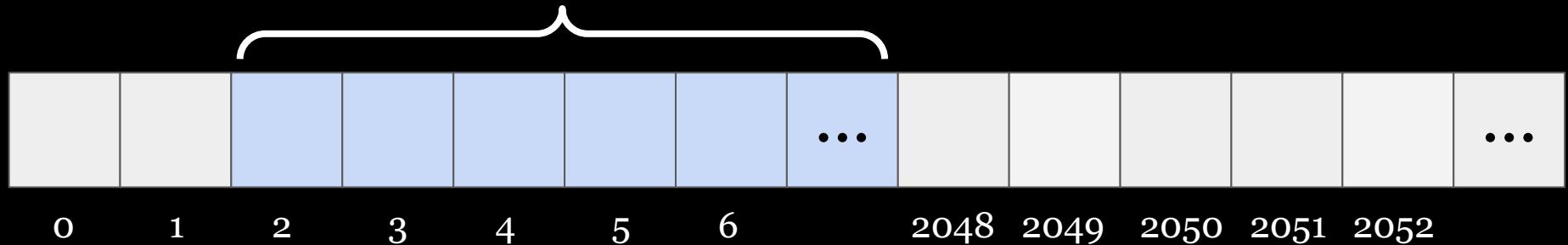
permissions: ...

sectors:

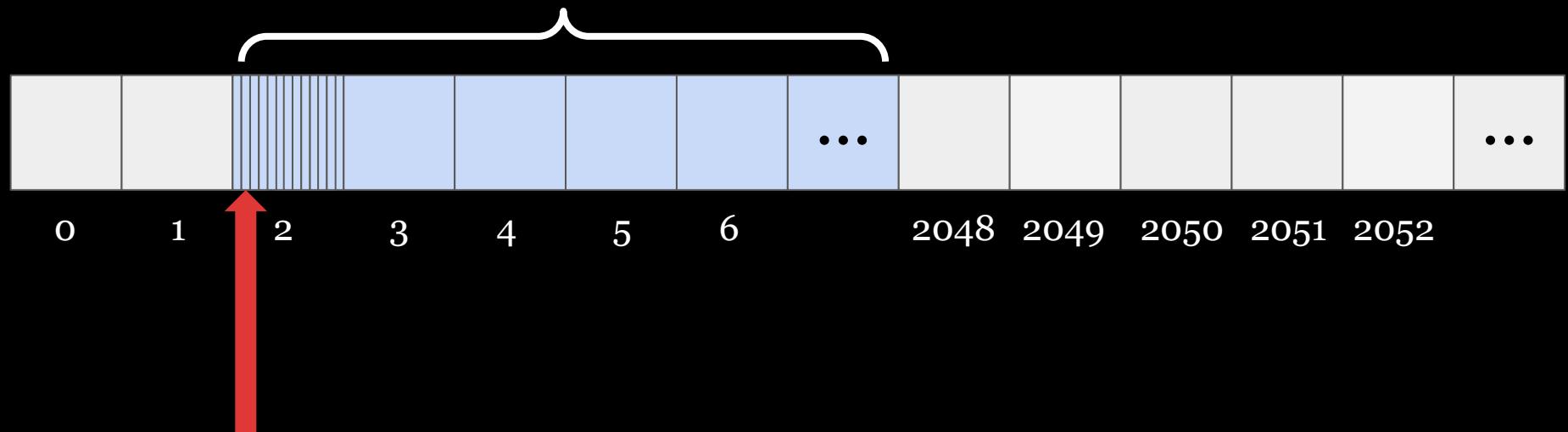
2049, 2052



inode table

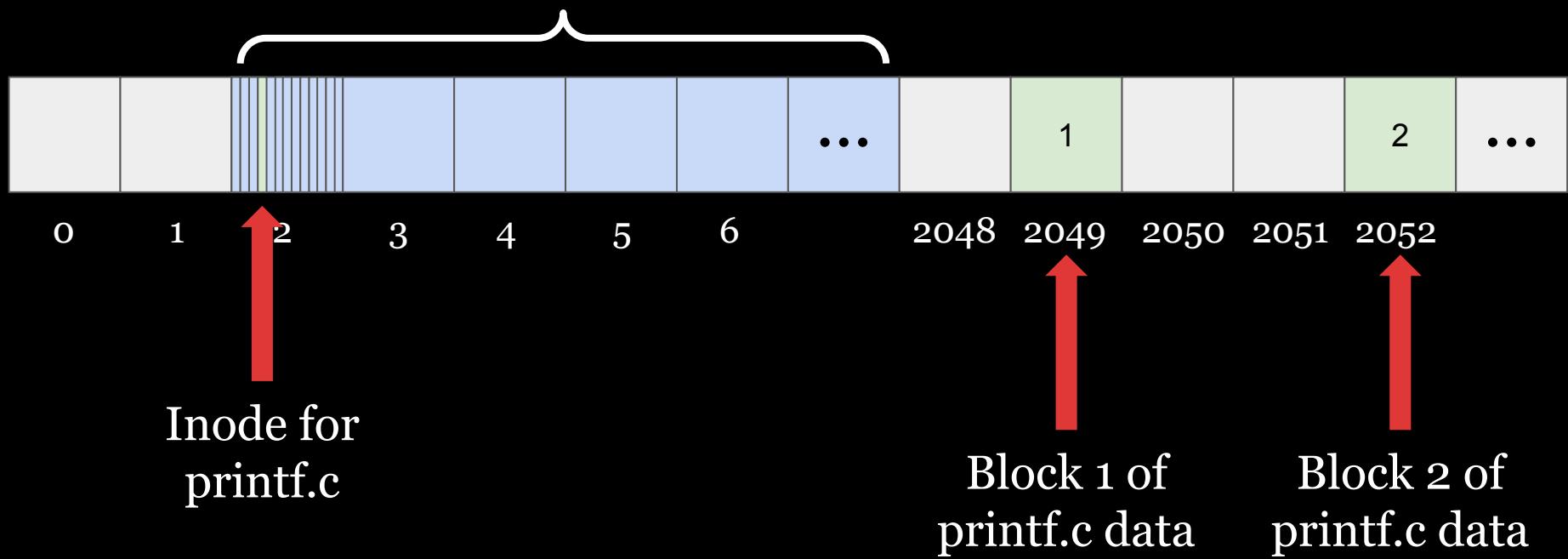


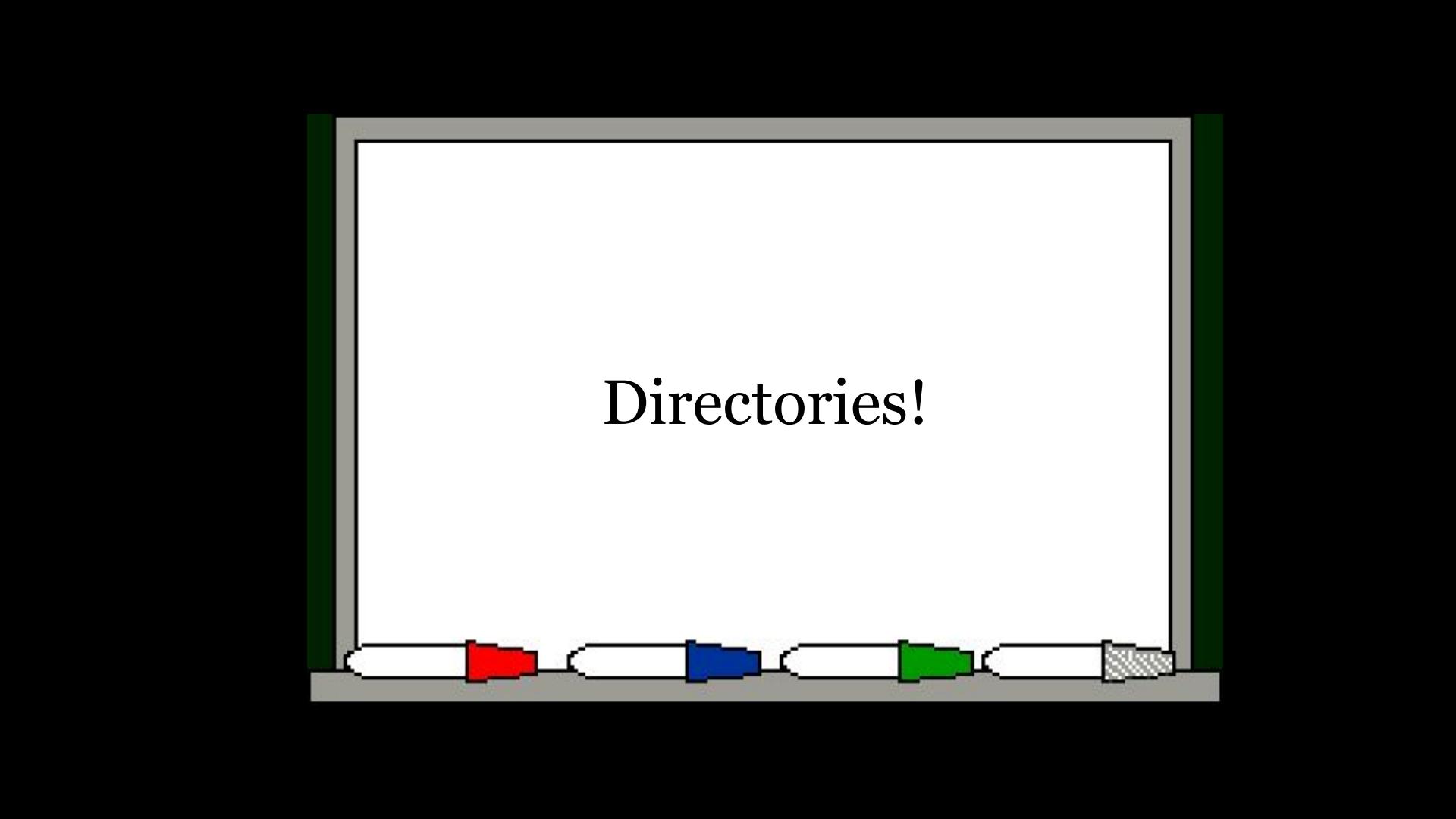
inode table



Each sector
holds 16 inodes

inode table





Directories!

An inode can represent a...

regular file

printf.c

size: 1024

file type: regular

permissions: ...

sectors:

2049, 2052

directory

/

size: 64

file type: directory

permissions: ...

sectors:

2048

/

size: 64

file type: directory

permissions: ...

sectors:

2048

Sector 2048:

| | |
|--------------|---|
| . | 1 |
| .. | 1 |
| printf.c | 4 |
| trombone.mp3 | 8 |

/

size: 64

file type: directory

permissions: ...

sectors:

2048

File name*

Sector 2048:

| | |
|--------------|---|
| . | 1 |
| .. | 1 |
| printf.c | 4 |
| trombone.mp3 | 8 |

*A file can mean a regular file or a directory!

/

size: 64

file type: directory

permissions: ...

sectors:

2048

Sector 2048:

| | |
|--------------|---|
| . | 1 |
| .. | 1 |
| printf.c | 4 |
| trombone.mp3 | 8 |

File name*

The inode
number for
the file

*A file can mean a regular file or a directory!

/

size: 64

file type: directory

permissions: ...

sectors:

2048

Sector 2048:

| | |
|--------------|---|
| . | 1 |
| .. | 1 |
| printf.c | 4 |
| trombone.mp3 | 8 |



. is the current directory, .. is the parent directory

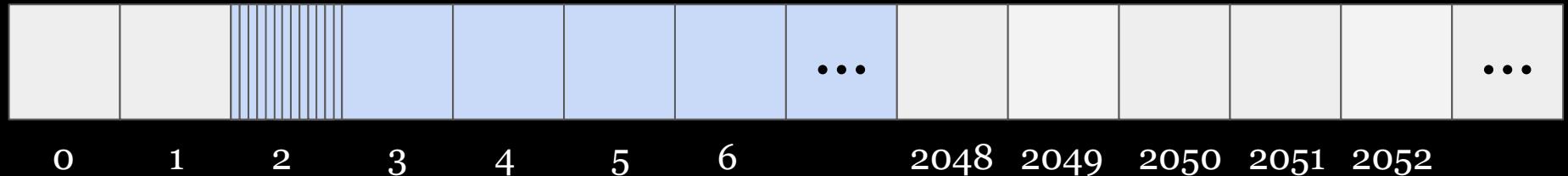
File name*



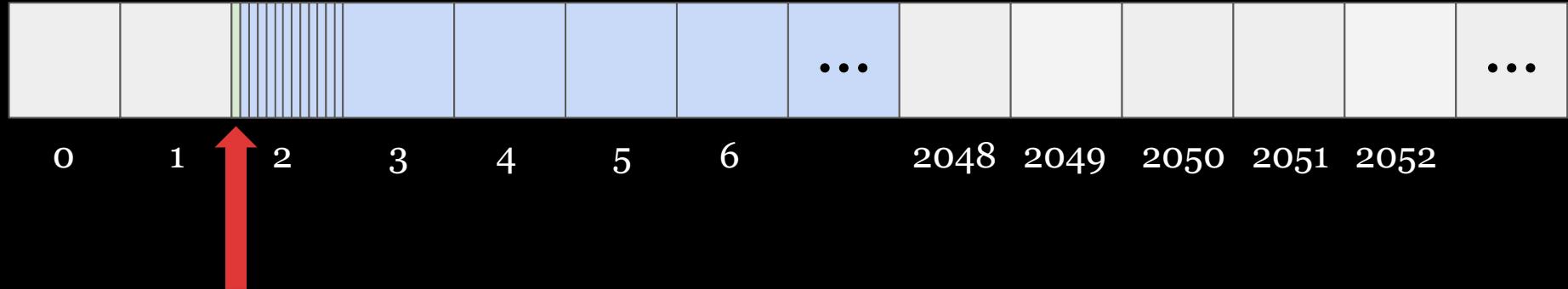
The inode number for the file

*A file can mean a regular file or a directory!

Let's walk through!!



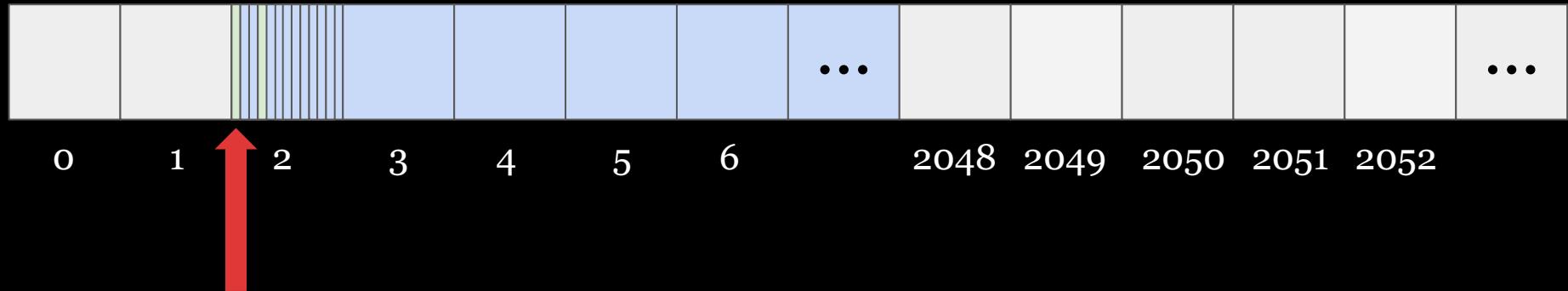
Let's walk through!!



Root inode

The inode representing the root folder
/ is always stored in Inode #1
(sector 2, offset 0)

Let's walk through!!

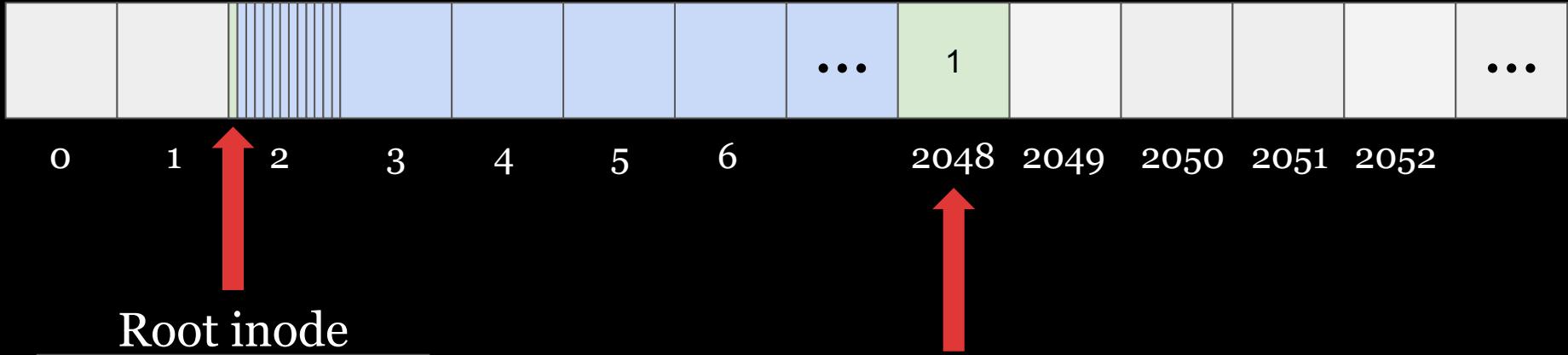


Root inode

/
size: 64
file type: directory
permissions: ...

sectors:
2048

Let's walk through!!



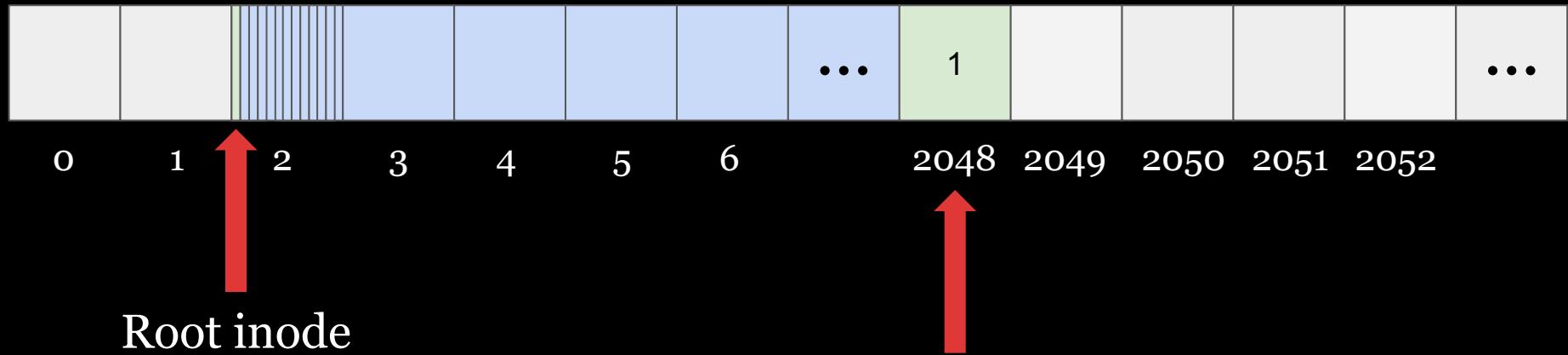
Root inode

/
size: 64
file type: directory
permissions: ...

sectors:
2048

Information about what's in the root directory

Let's walk through!!



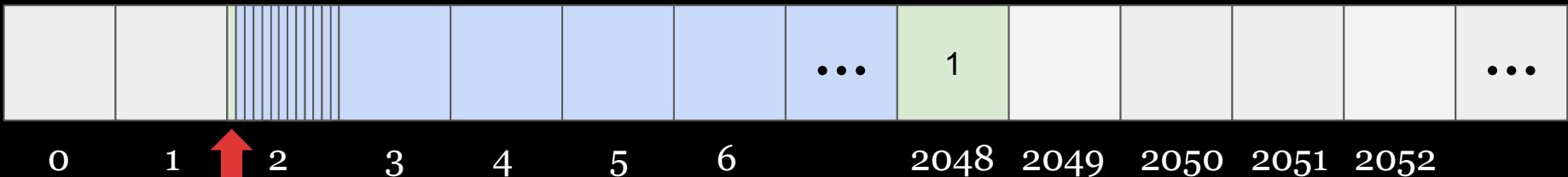
Root inode

/
size: 64
file type: directory
permissions: ...

sectors:
2048

| | |
|--------------|---|
| . | 1 |
| .. | 1 |
| printf.c | 4 |
| trombone.mp3 | 8 |

Let's walk through!!



Root inode

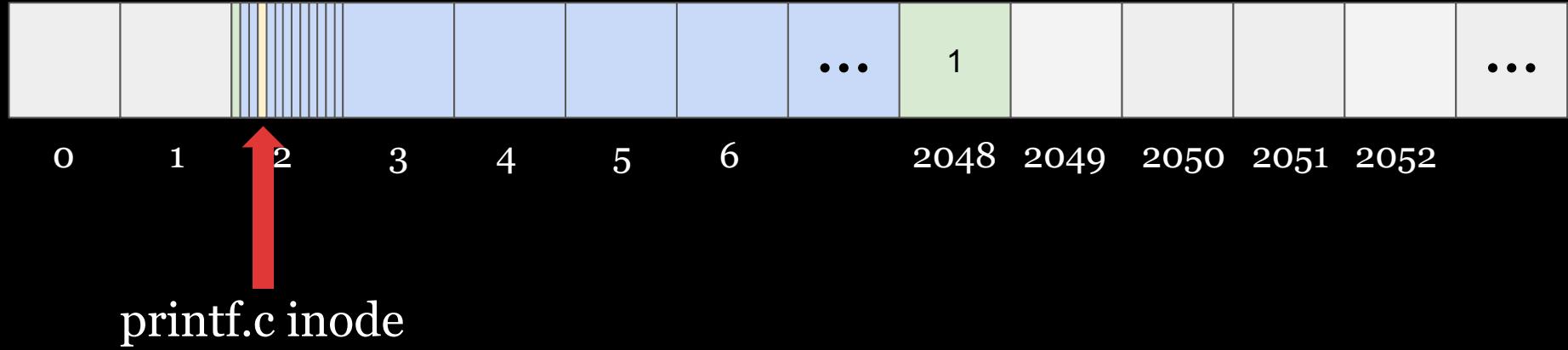
/
size: 64
file type: directory
permissions: ...

sectors:
2048

| | |
|--------------|---|
| . | 1 |
| .. | 1 |
| printf.c | 4 |
| trombone.mp3 | 8 |

If we want to
get to printf.c,
we need to
check Inode #4

Let's walk through!!



printf.c

size: 1024

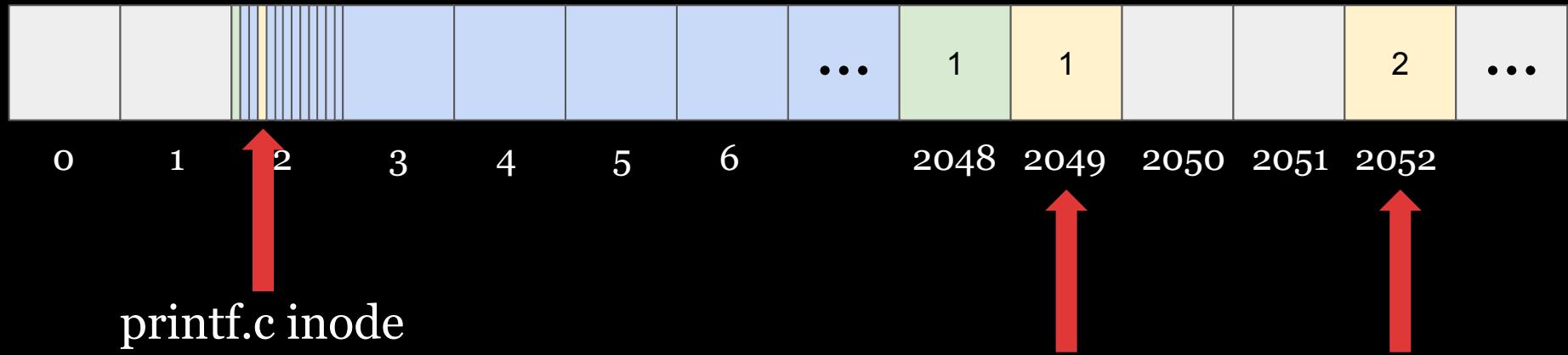
file type: regular

permissions: ...

sectors:

2049, 2052

Let's walk through!!



printf.c
size: 1024
file type: regular
permissions: ...

sectors:
2049, 2052

So you start at Inode #1
(root folder) and you
can get anywhere from
there!





TAKE A DEEP BREATH because
that's how file systems work

inode

printf.c
size: 1024
file type: regular
permissions: ...

sectors:
2049, 2052

There is not
space to store a
ton of sector
numbers here!



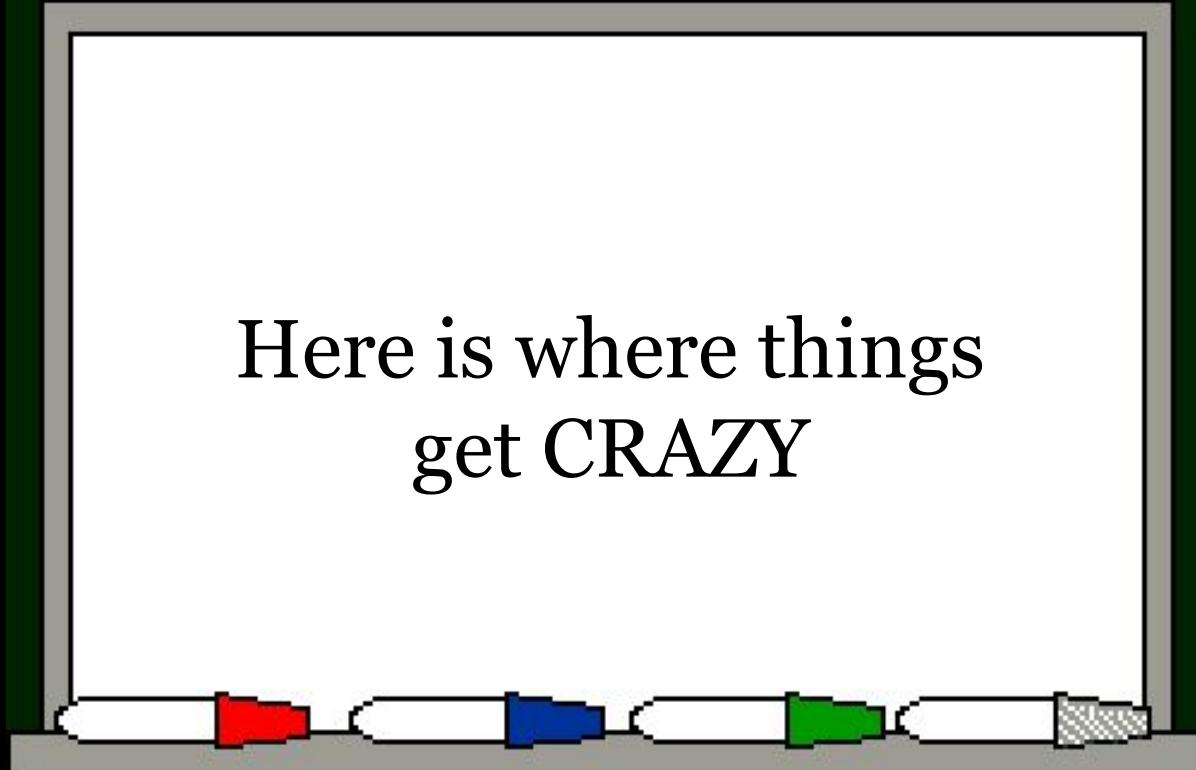
inode

What if we have
more than 8
sectors we want
to store??

printf.c
size: 4098 ←
file type: regular
permissions: ...

sectors:
2049, 2052, 2057,
4923, 2938, 2293,
2939, 2999, ????

This is $512 \times 8 + 2$
(so we need 9
blocks)



Here is where things
get **CRAZY**

printf.c

size: 4098

file type: regular

permissions: ...

sectors:

3013

This one sector tells us where we're storing our sector list!

Sector 3013:

printf.c

size: a lot of bytes
file type: regular
sectors:
3013, 3016, 2059

We can represent really big files like this!

Sector 3013:

| | | | | |
|------|------|------|------|------|
| 2049 | 2052 | 2057 | 4923 | 2938 |
| 2293 | 2939 | 2999 | 8392 | 2343 |
| 2392 | 3994 | 3959 | 3969 | 5939 |
| 3969 | 5924 | 5924 | 3492 | 4969 |
| 2303 | 2084 | 2302 | 2920 | 4392 |
| 2308 | 2383 | 2832 | 3023 | 2302 |
| 2300 | 2380 | 3200 | 2830 | 2332 |
| 3830 | 3828 | 2830 | 2838 | 2090 |

Sector 3016:

| | | | | |
|------|------|------|------|------|
| 7392 | 7373 | 9273 | 2999 | 3020 |
| 2099 | 2929 | 3992 | 7777 | 7372 |
| 7889 | 7483 | 7322 | 3272 | 2939 |
| 8580 | 9588 | 6858 | 2929 | 3922 |
| 3211 | 4811 | 4812 | 8811 | 9111 |
| 9991 | 4818 | 4817 | 2741 | 3757 |
| 3729 | 3702 | 4821 | 4831 | 3819 |
| 3829 | 2839 | 2938 | 2829 | 3829 |

Sector 2059:

printf.c

size: a lot of bytes

file type: regular

sectors:

3013

Sector 3013:

| | | | | |
|------|------|------|------|------|
| 2049 | 2052 | 2057 | 4923 | 2938 |
| 2293 | 2939 | 2999 | 8392 | 2343 |
| 2392 | 3994 | 3959 | 3969 | 5939 |
| 3969 | 5924 | 5924 | 3492 | 4969 |
| 2303 | 2084 | 2302 | 2920 | 4392 |
| 2308 | 2383 | 2832 | 3023 | 2302 |
| 2300 | 2380 | 3200 | 2830 | 2332 |
| 3830 | 3828 | 2830 | 2838 | 2090 |

If a file is REALLY big, each sector can represent a list of
sector lists

Sector 2049:

| | | | | |
|------|------|------|------|------|
| 7392 | 7373 | 9273 | 2999 | 3020 |
| 2099 | 2929 | 3992 | 7777 | 7372 |
| 7889 | 7483 | 7322 | 3272 | 2939 |
| 8580 | 9588 | 6858 | 2929 | 3922 |
| 3211 | 4811 | 4812 | 8811 | 9111 |
| 9991 | 4818 | 4817 | 2741 | 3757 |
| 3729 | 3702 | 4821 | 4831 | 3819 |
| 3829 | 2839 | 2938 | 2829 | 3829 |

Sector 2052:

think this stuff is cool?



Systems

Systems is the study of the design and implementation of computer systems such as compilers, databases, networks, and operating systems. Topics include the hardware/software interface, the networking stack, digital architecture, memory models, optimization, concurrency, privacy, security, distributed and large-scale systems, reliability and fault tolerance, and related algorithms and theoretical topics.

(In CS 110, this takes up a 90 minute lecture, so if it felt a bit too fast-paced here, don't worry!!)

classes:

CS 110: Principles of Computer Systems

1

Principles and practice of engineering of computer software and hardware systems. Topics include: techniques for controlling complexity; strong modularity using client-server design, virtual memory, and threads; networks; atomicity and coordination of parallel activities. Prerequisite: 107.



CS 140: Operating Systems and Systems Programming

Operating systems design and implementation. Basic structure; synchronization and communication mechanisms; implementation of processes, process management, scheduling, and protection; memory organization and management, including virtual memory; I/O device management, secondary storage, and file systems. Prerequisite: CS110.



CS 140E: Operating systems design and implementation

Students will implement a simple, clean operating system (virtual memory, processes, file system) in the C programming language, on a raspberry pi computer and use the result to run a variety of devices and implement a final project. All hardware is supplied by the instructor, and no previous experience with operating systems, raspberry pi, or embedded programming is required.

2

CS 111: Operating Systems Principles

Explores operating system concepts including concurrency, synchronization, scheduling, processes, virtual memory, I/O, file systems, and protection. Available as a substitute for CS110 that fulfills any requirement satisfied by CS110. Prerequisite: CS107.