Liana + Anna Lecture!

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

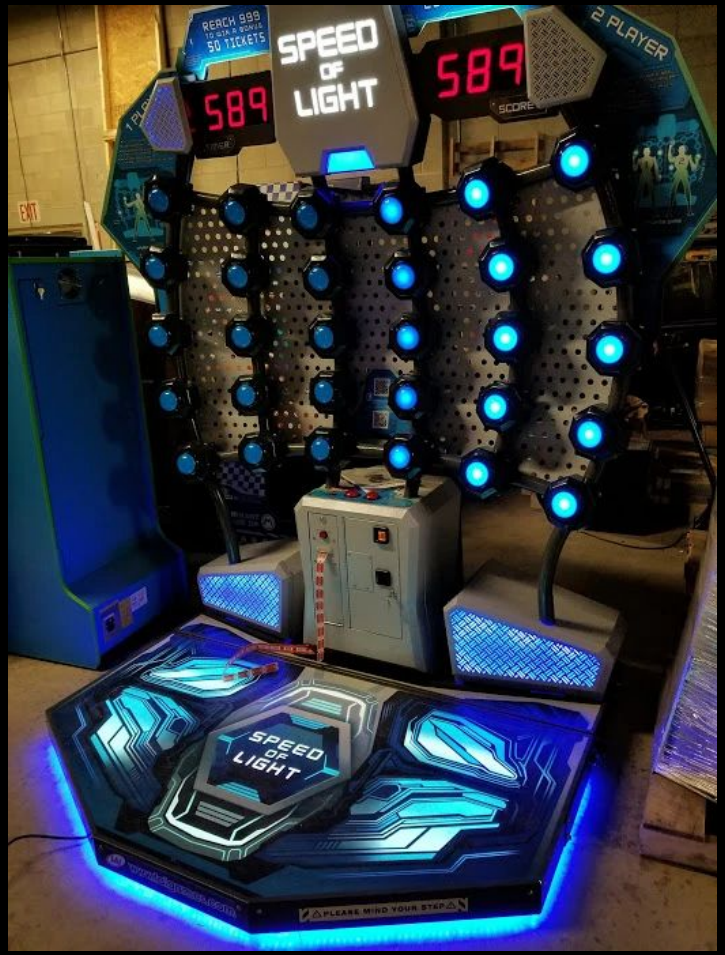
Final project process

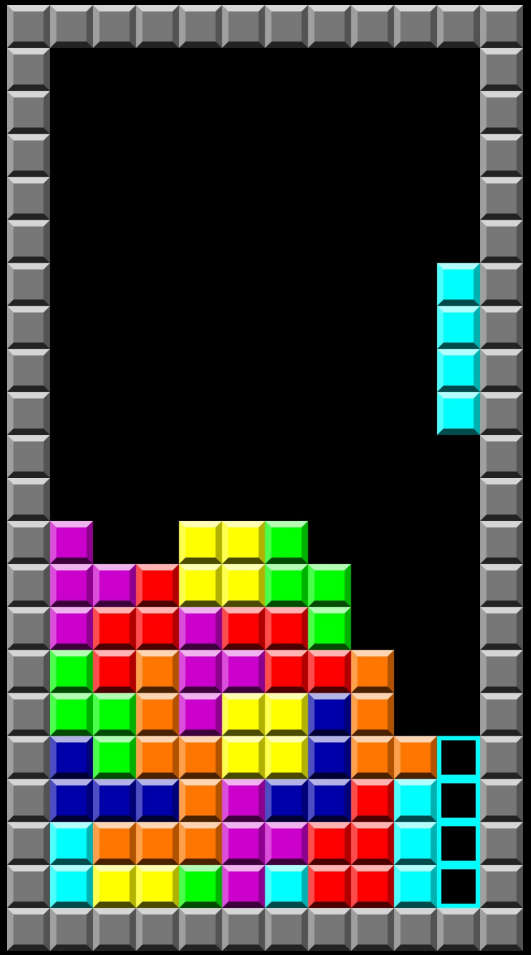
Where to go from 107E

- Embedded systems route (EE 108, etc)
- Operating systems route (CS 110)

Final Projects


HOLOGRAMSORBUST
HOLOGRAMSORBUST
HOLOGRAMSORBUST


SPEED OF LIGHT
589
589
2 PLAYER
SCORE
SPEED OF LIGHT
PLEASE MIND YOUR STEP

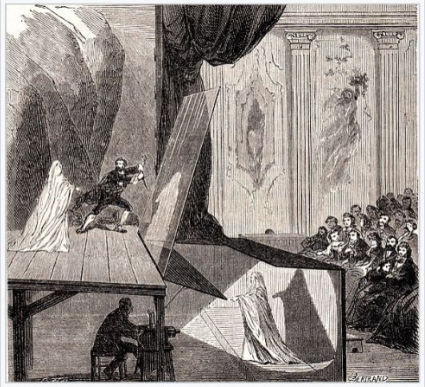


Pepper's ghost

From Wikipedia, the free encyclopedia

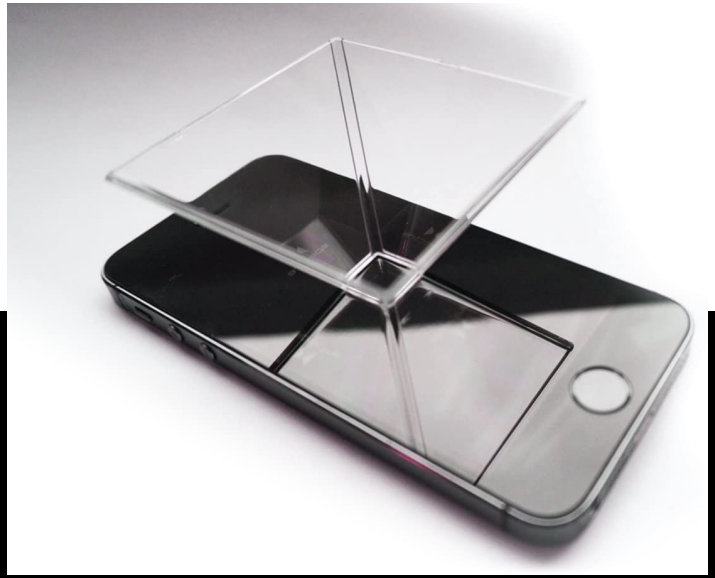
For other uses, see Pepper's ghost (disambiguation).

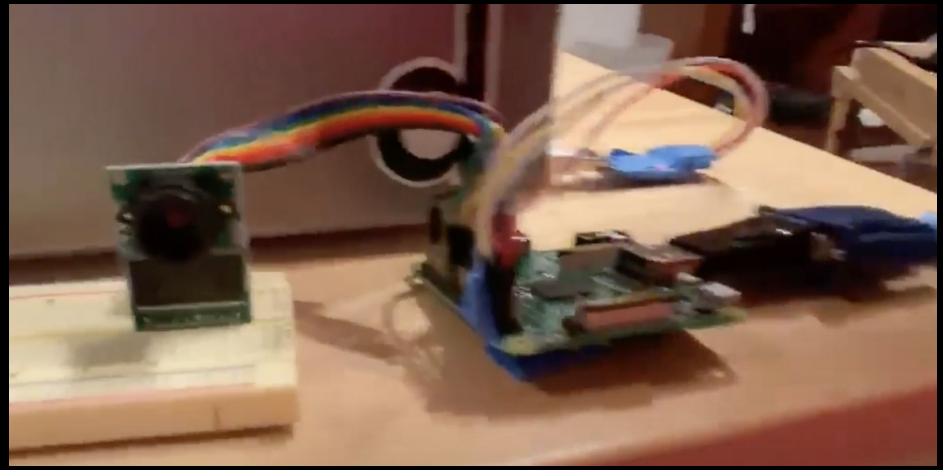
Pepper's ghost is an illusion technique used in the theatre, amusement parks, museums, television, and concerts. It is named after the English scientist John Henry Pepper (1821–1900) who began popularizing the effect with a demonstration in 1862.[1] Examples of the illusion are the Girl-to-Gorilla trick 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.



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.

Contents [hide]



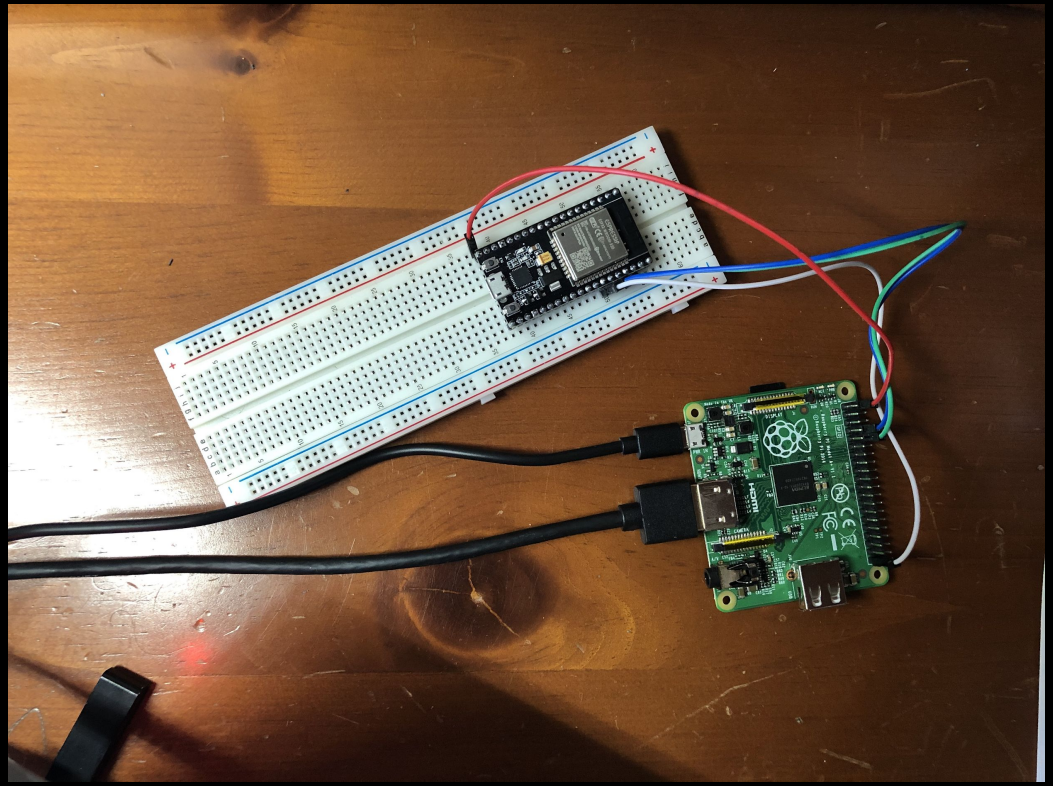


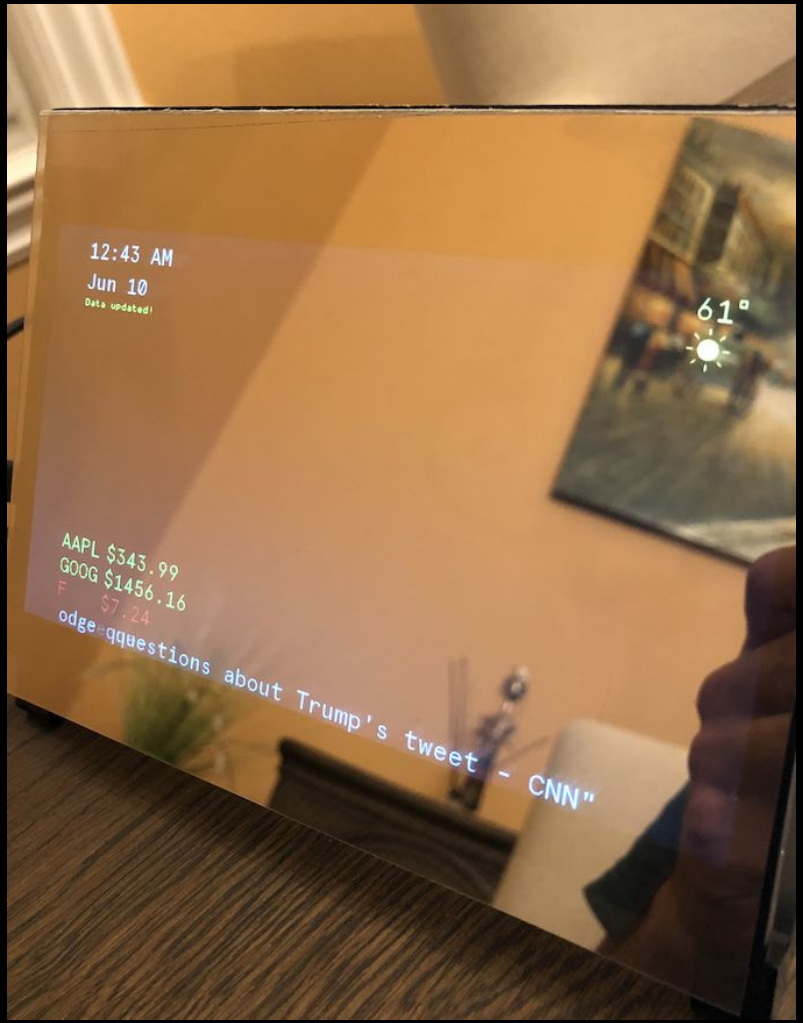
```
132             fb_pixels[one_third_height - height_width_difference - row][two_thirds_width - (col + 3)] = hologr
133             fb_pixels[one_third_height - height_width_difference - row][two_thirds_width - (col + 4)] = hologr
134             fb_pixels[one_third_height - height_width_difference - row][two_thirds_width - (col + 5)] = hologr
135             fb_pixels[one_third_height - height_width_difference - row][two_thirds_width - (col + 6)] = hologr
136             fb_pixels[one_third_height - height_width_difference - row][two_thirds_width - (col + 7)] = hologr
137         }
138         for (int col = 0; col < hologram_width; col += 8) {
139             fb_pixels[one_third_height - height_width_difference - (row + 1)][two_thirds_width - col] = hologr
140             fb_pixels[one_third_height - height_width_difference - (row + 1)][two_thirds_width - (col + 1)] =
141             fb_pixels[one_third_height - height_width_difference - (row + 1)][two_thirds_width - (col + 2)] =
142             fb_pixels[one_third_height - height_width_difference - (row + 1)][two_thirds_width - (col + 3)] =
143             fb_pixels[one_third_height - height_width_difference - (row + 1)][two_thirds_width - (col + 4)] =
144             fb_pixels[one_third_height - height_width_difference - (row + 1)][two_thirds_width - (col + 5)] =
145             fb_pixels[one_third_height - height_width_difference - (row + 1)][two_thirds_width - (col + 6)] =
146             fb_pixels[one_third_height - height_width_difference - (row + 1)][two_thirds_width - (col + 7)] =
147         }
148         for (int col = 0; col < hologram_width; col += 8) {
149             fb_pixels[one_third_height - height_width_difference - (row + 2)][two_thirds_width - col] = hologr
150             fb_pixels[one_third_height - height_width_difference - (row + 2)][two_thirds_width - (col + 1)] =
151             fb_pixels[one_third_height - height_width_difference - (row + 2)][two_thirds_width - (col + 2)] =
152             fb_pixels[one_third_height - height_width_difference - (row + 2)][two_thirds_width - (col + 3)] =
```

Step #1:
Choose a really easy base project

Step #2
Once you've built the base, you can add more features

magic mirror



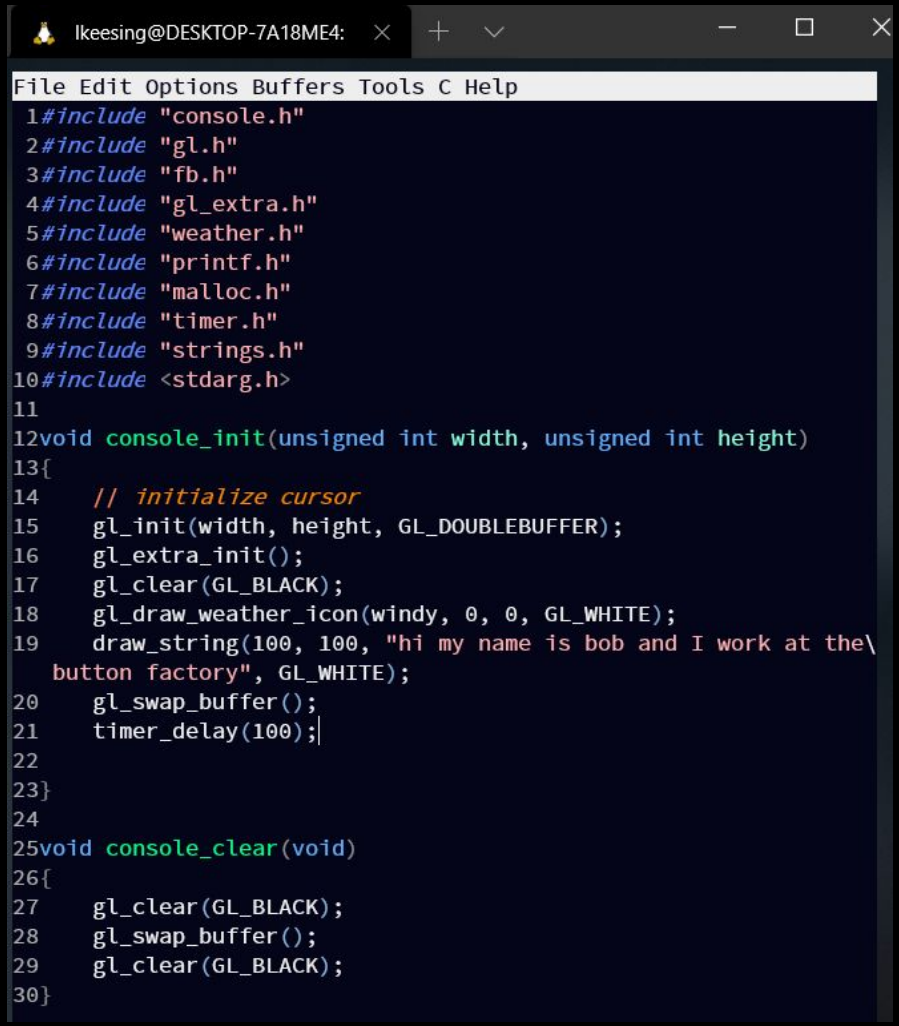

12:43 AM
Jun 10
Data updated!
61°
AAPL $343.99
GOOG $1456.16
F $7.24
odge=qquestions about Trump's tweet - CNN"

project roadmap:

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

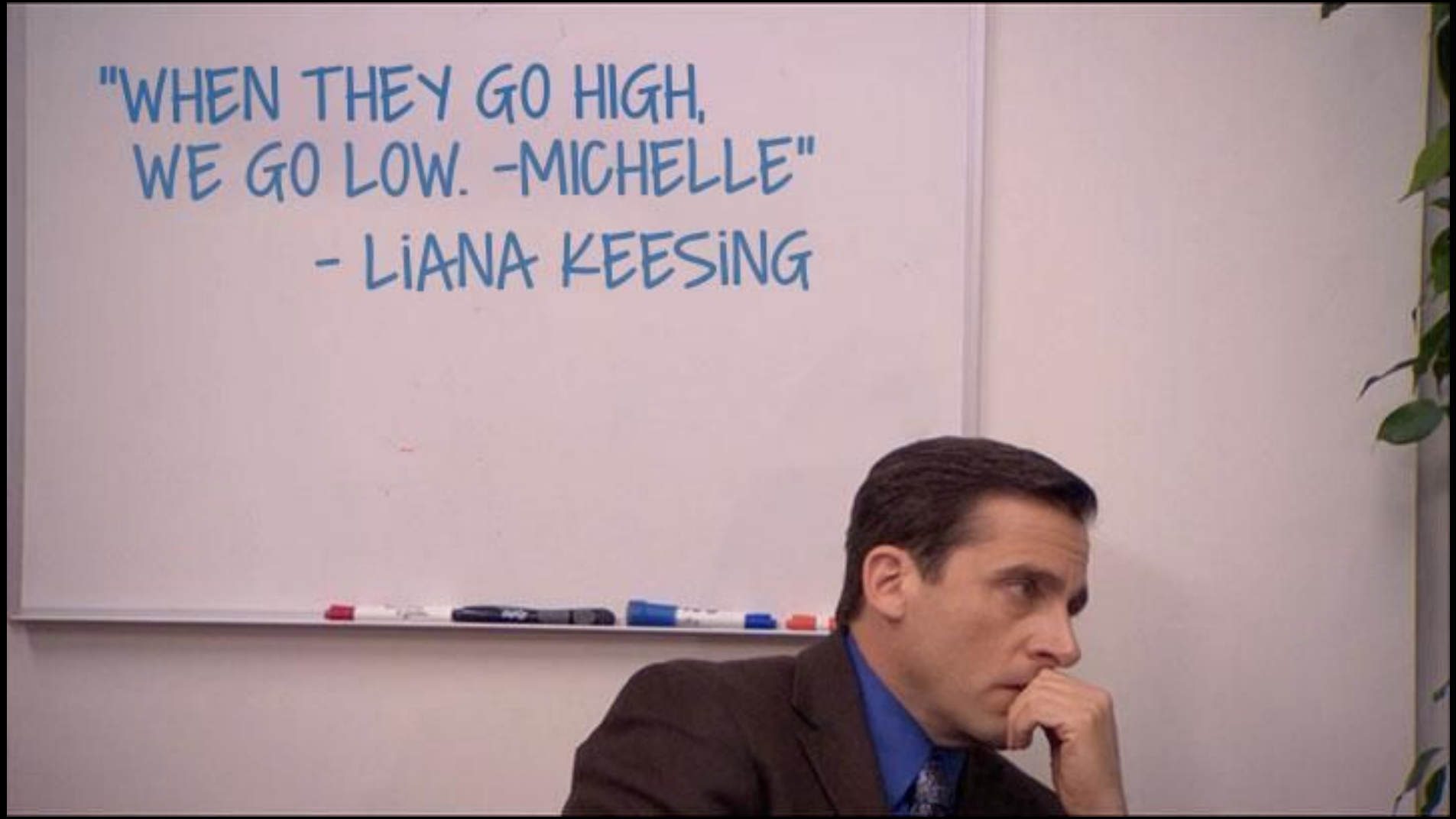
Graphics
[x] Extend graphics library for modules:
    [x] Date and Time (we should configure this under the hood and let ARM timer do the work OR we can write a proxy server)
    [x] Weather ([Weather API]
    [x] News Headlines ([News API]
    [x] Custom Font (could be cool if time)

build off of the existing codebase!!!




12:04 PM
Jun 19
81°
Testimony In Judiciary Hearing On Police -


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

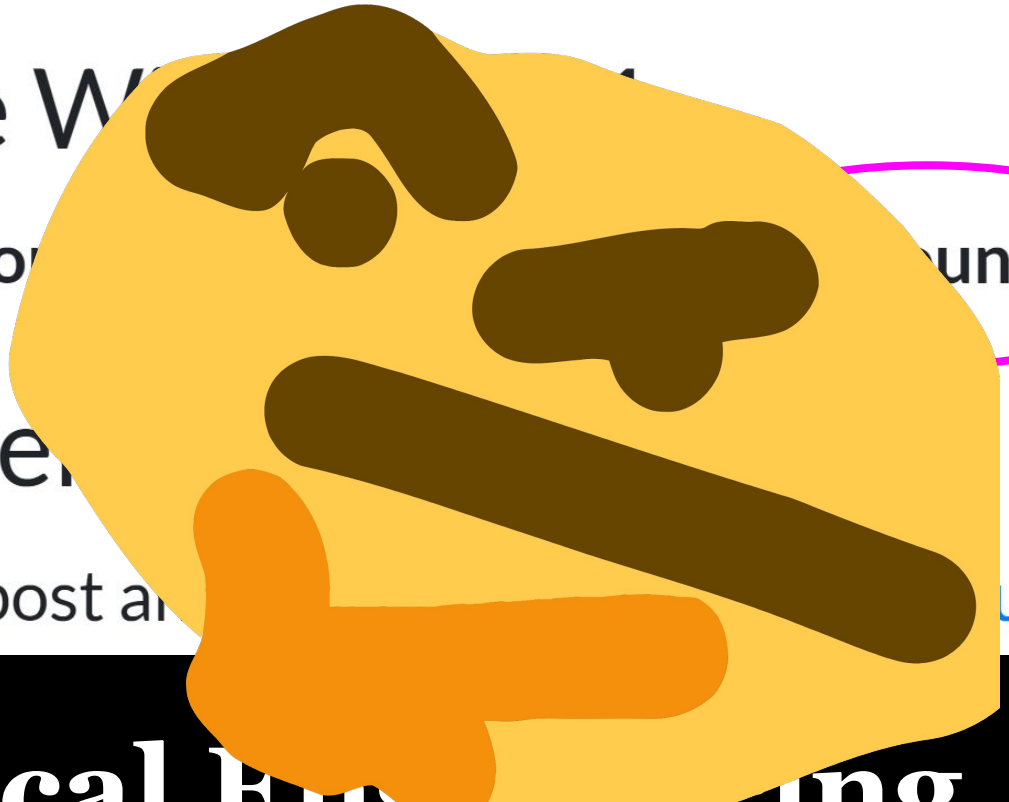

"WHEN THEY GO HIGH,
WE GO LOW. -MICHELLE"
- LIANA KEESING

CS107e W

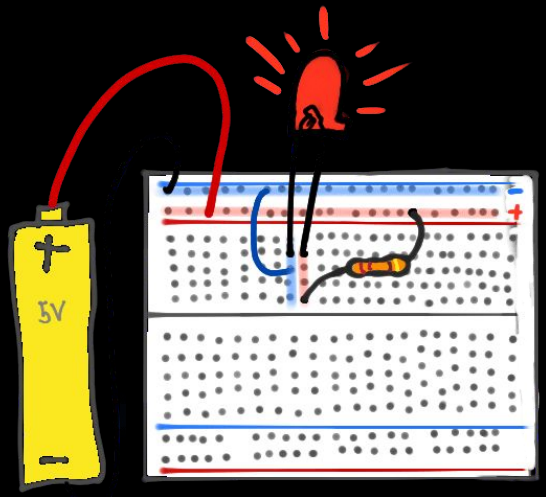
CS107e is Co und Up.

Announce

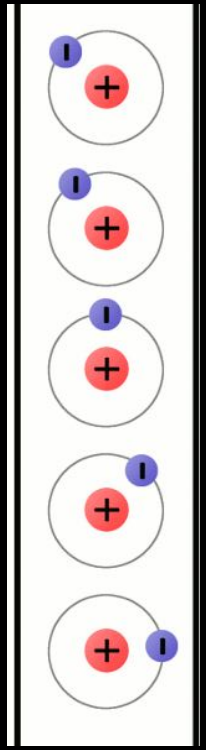
- We will post a um .

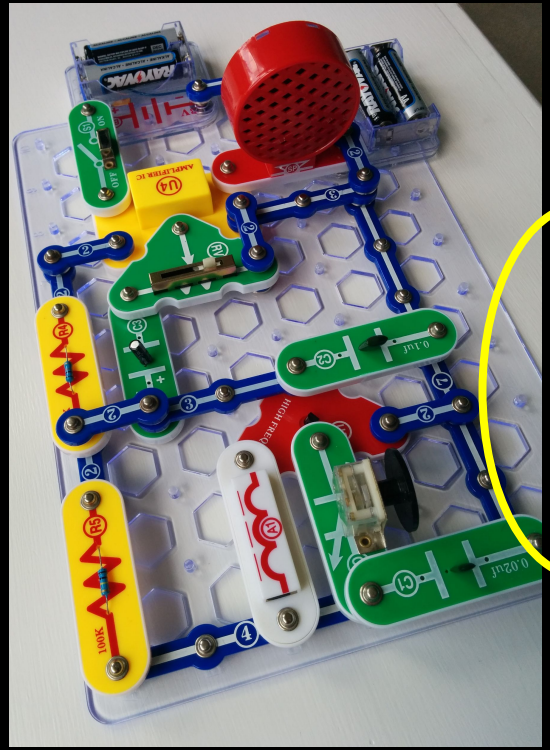


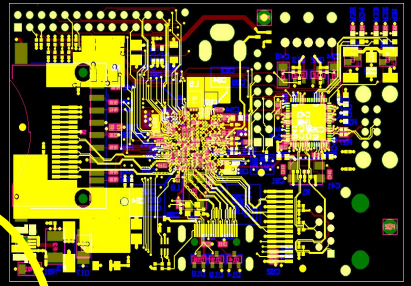
Electrical Engi ing

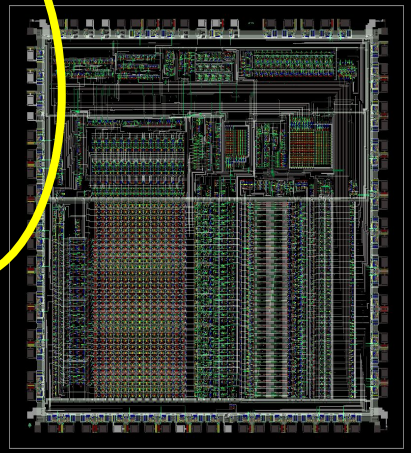

5V


MacBook Pro



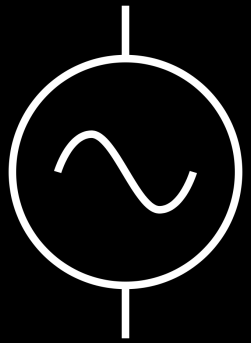


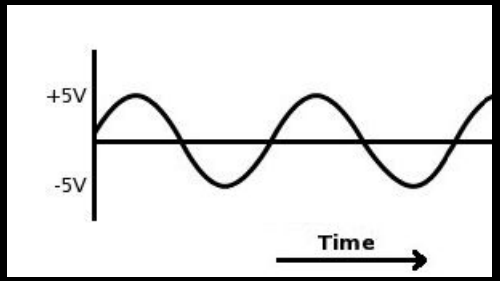




```
lkeesing@DESKTOP-7A18ME4:~/cs107e/cs107e_home/assignments/src/lib$ cat interrupts_asm.s
@ Assembly code for interrupt vector table and functions
@ to enable/disable IRQ interrupts on the Rapsberry Pi in CS107E.
@
@ Author:       Philip Levis
@ Author:       Julie Zelenski
@ Last update: 2/20/20

@ 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 (ee180, cs140e)**





logical “1”

non-usable area

logical “0”

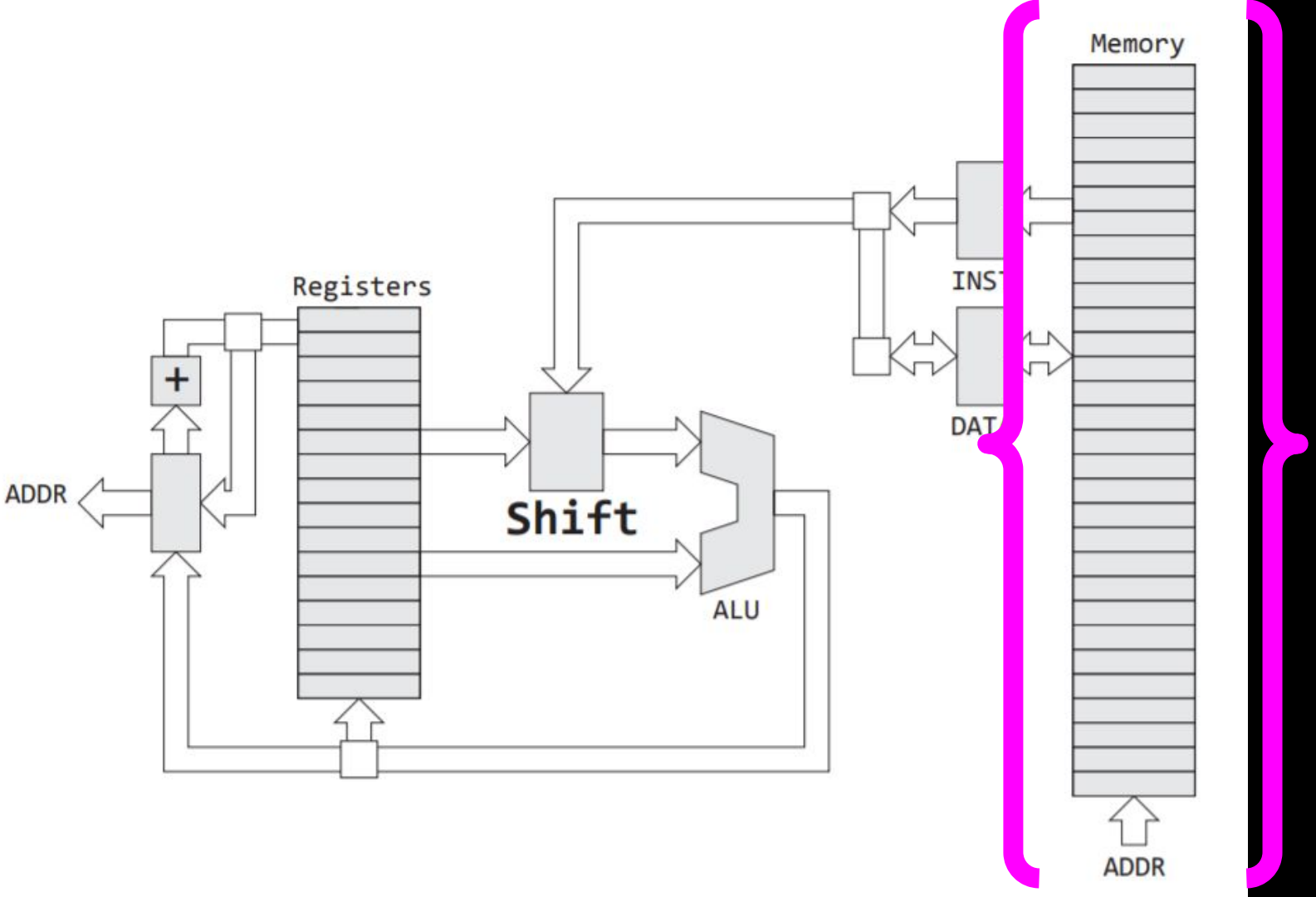
V_{CC} = 5V

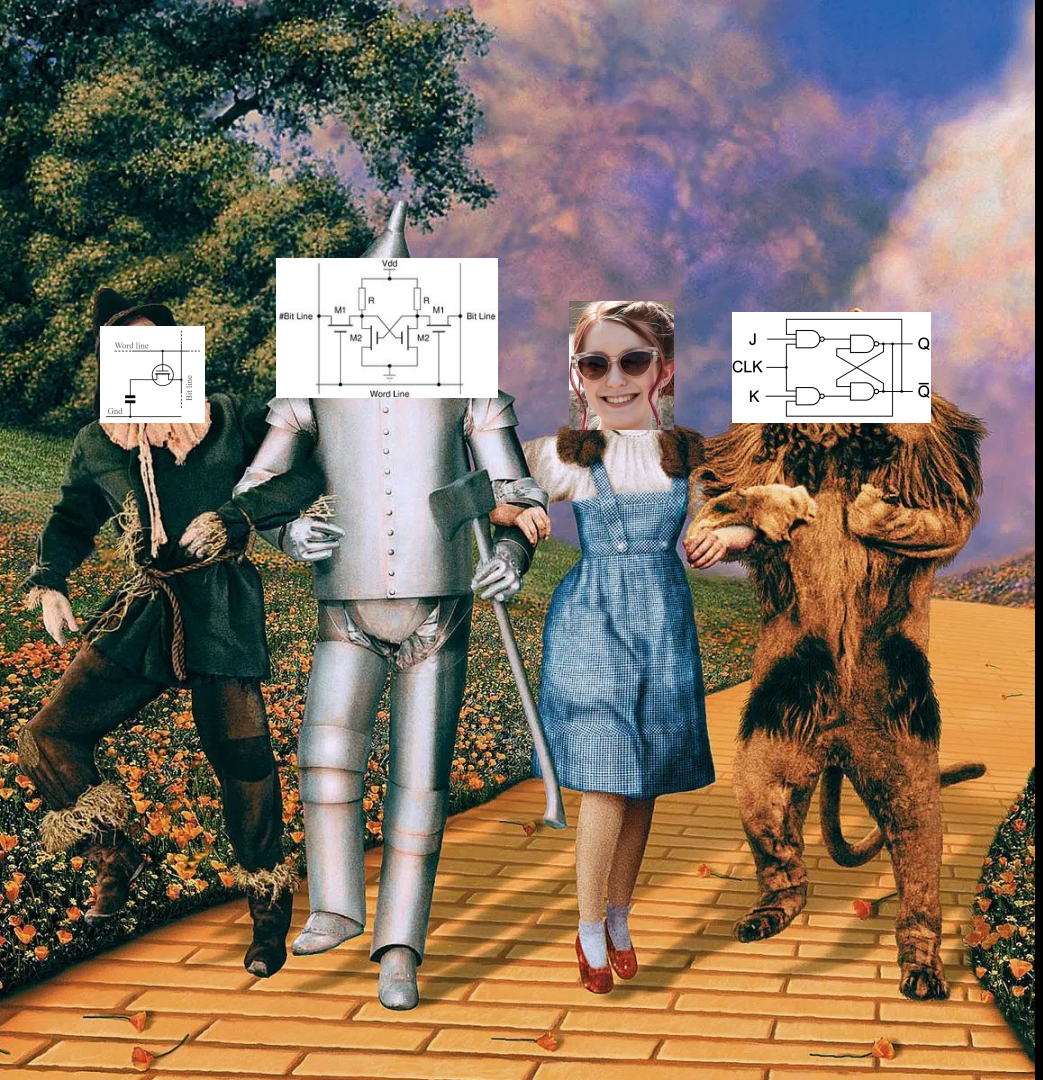
V_{ON} (min) = 3.0V

V_{OFF} (max) = 0.8V

0V

what does “digital” even mean?


Memory
Registers
INST
DAT
+
ADDR
Shift
ALU
ADDR



from voltage differentials to memory

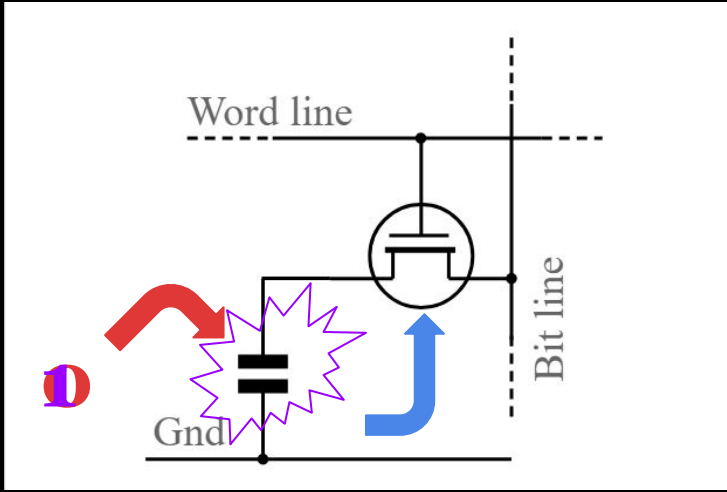
or,

DRAM and SRAM and flip-flops, oh my!

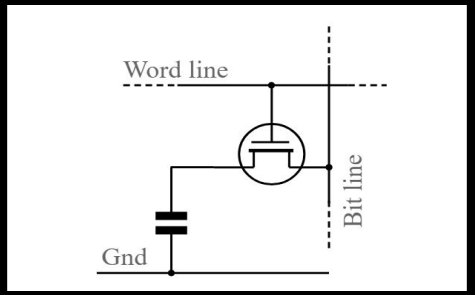
RPi memory is

Dynamic Random Access Memory

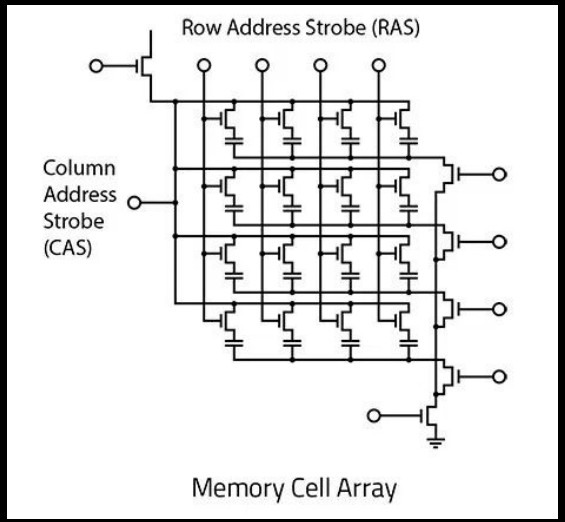
transistor + **capacitor**



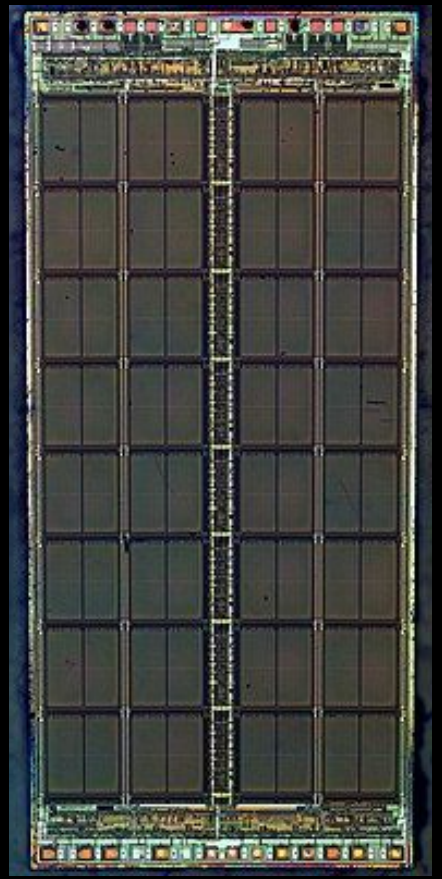
memory cell

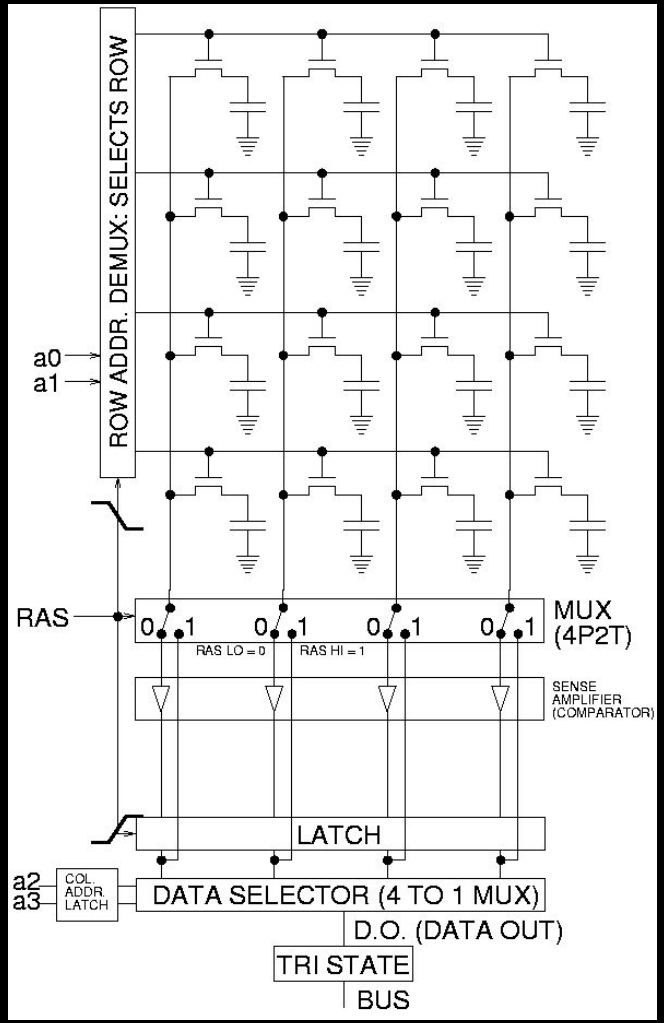


memory cell array

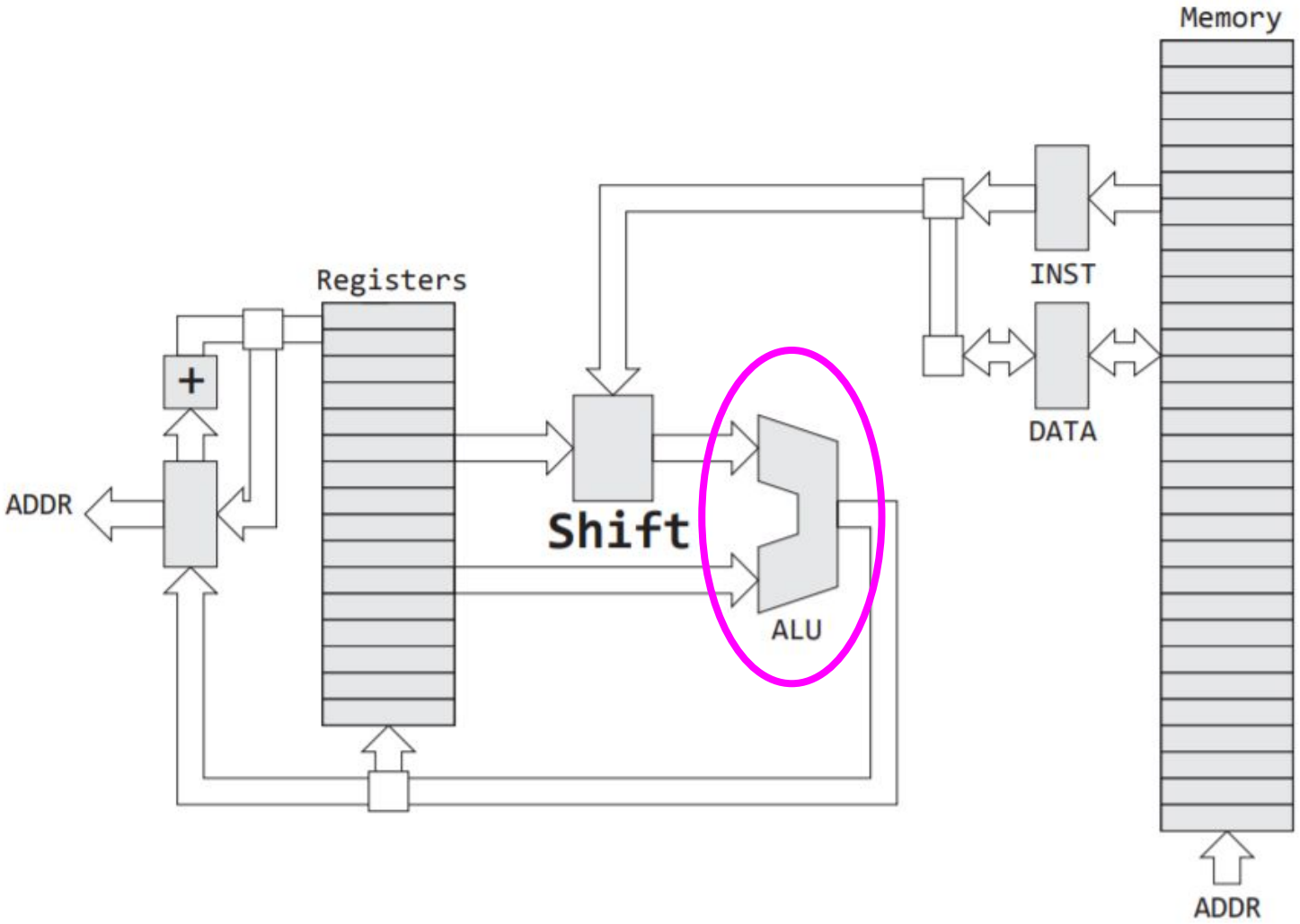


DRAM!

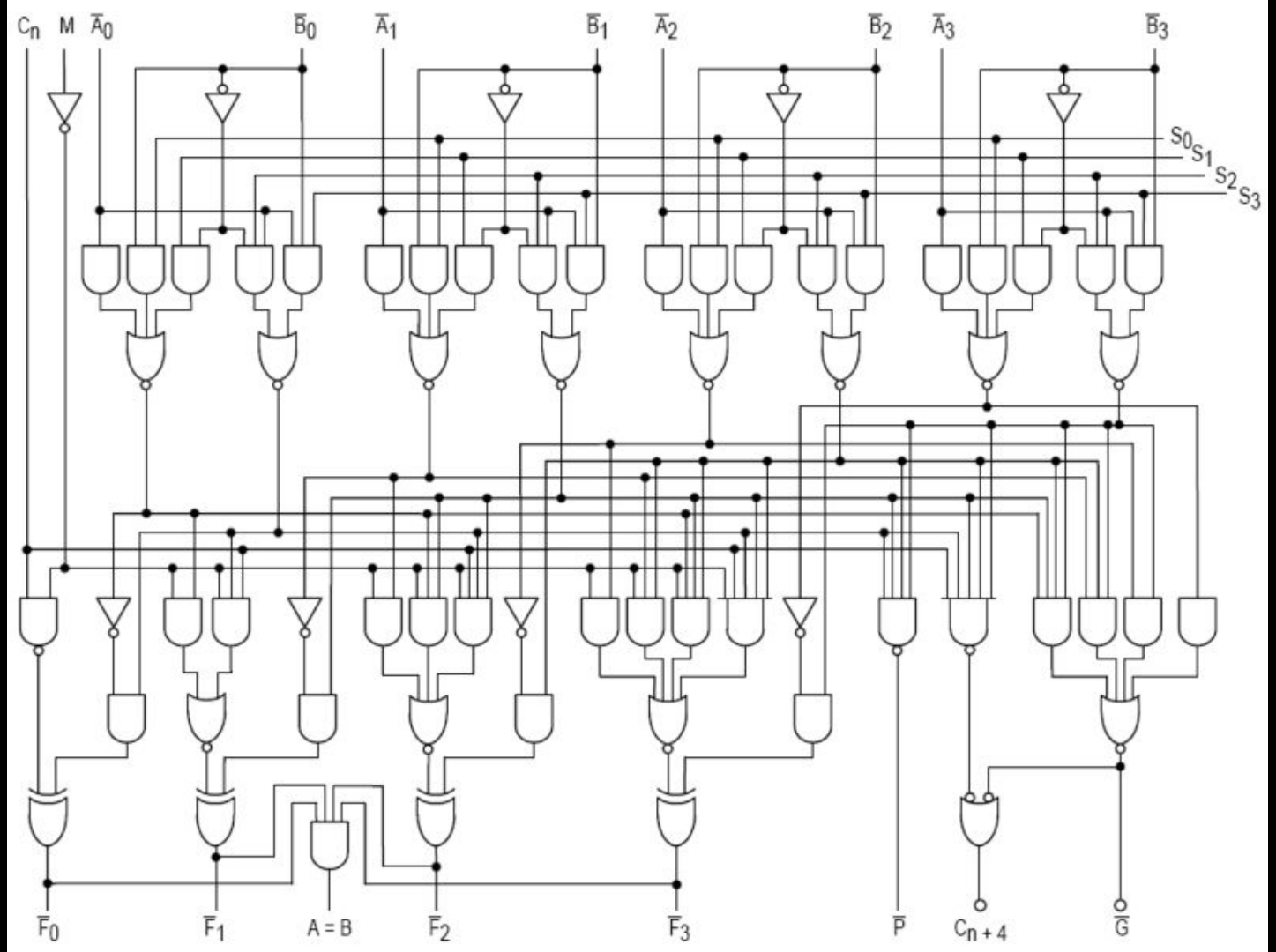




ok actually we cheated here and added a multiplexer and some latches but the basic structure remains

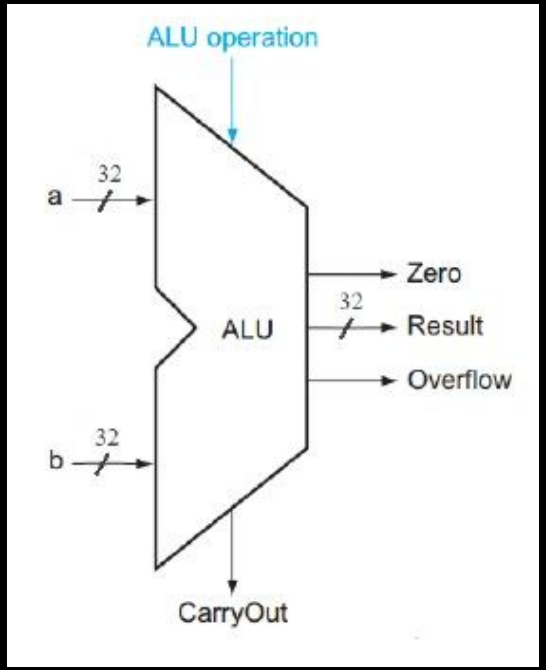

Memory
INST
DATA
Registers
+
ADDR
Shift
ALU
ADDR

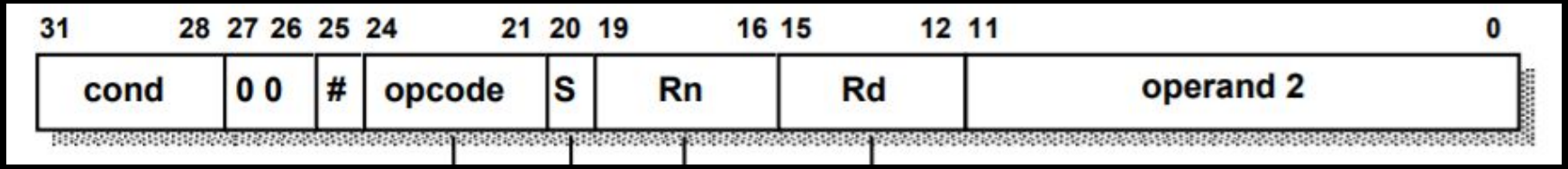

...the ALU isn't magic,
it's just a fancy circuit


C_n
M
$\overline{A}_0$
$\overline{B}_0$
$\overline{A}_1$
$\overline{B}_1$
$\overline{A}_2$
$\overline{B}_2$
$\overline{A}_3$
$\overline{B}_3$
S_0
S_1
S_2
S_3
$\overline{F}_0$
$\overline{F}_1$
A = B
$\overline{F}_2$
$\overline{F}_3$
$\overline{P}$
C_{n+4}
$\overline{G}$

(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

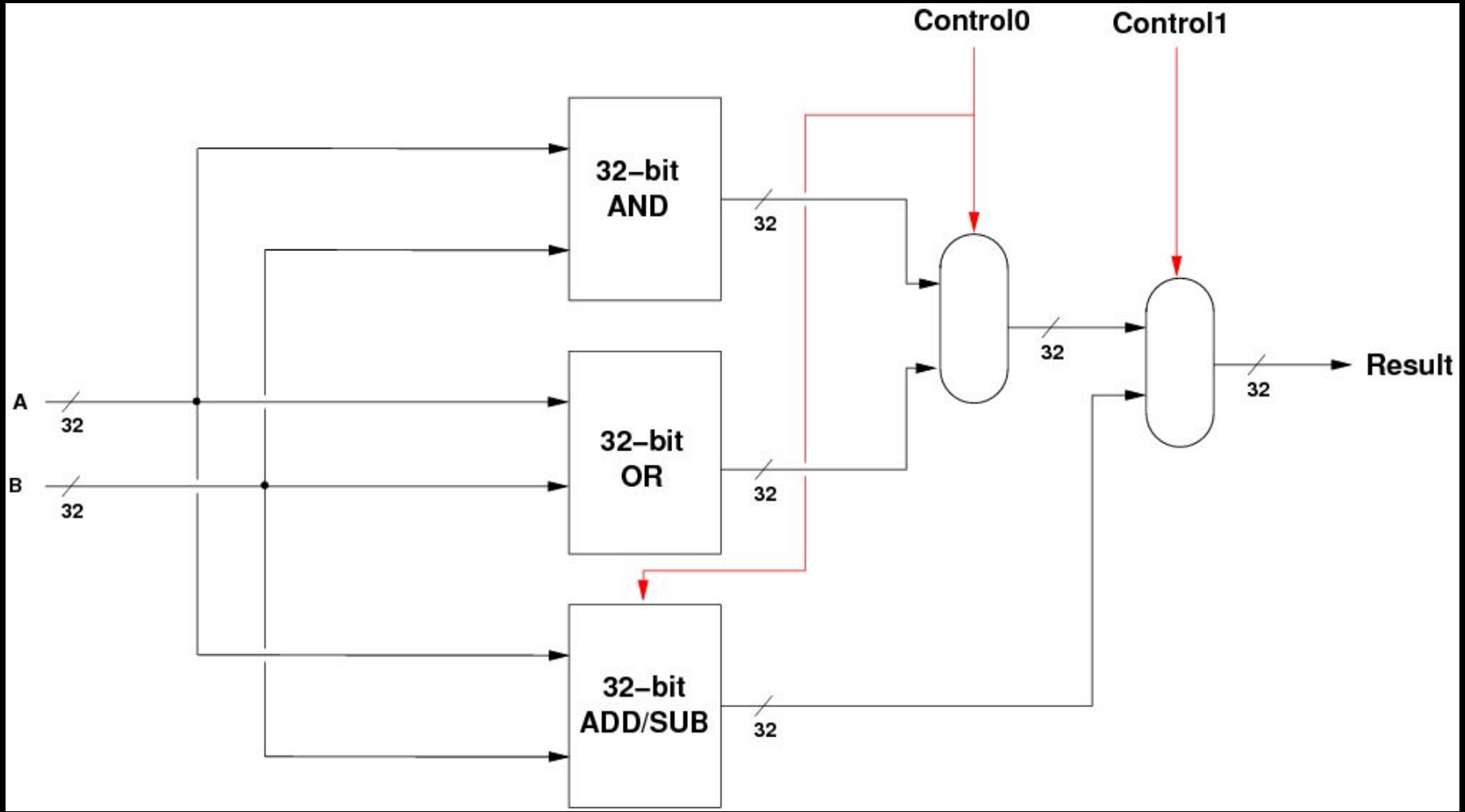
an HDL like Verilog can explain what the ALU should do!

```
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!

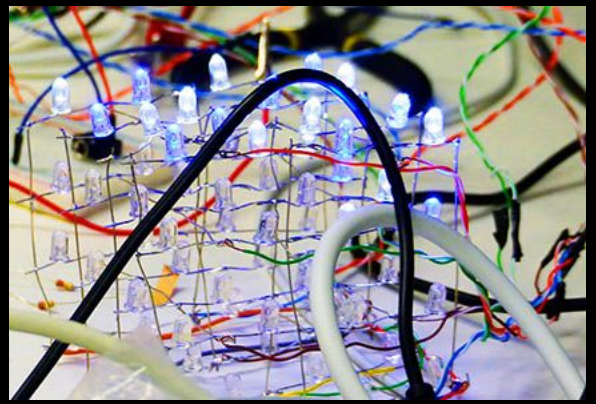
```
module alu(
input [31 : 0] A, B, // ALU 8 - BIT Inputs
input [3 : 0] opcode, // ALU Selection
input cIn,
output [31 : 0] ALU_Out, // ALU 8 - BIT
Output status,
output CarryOut // Carry OUT Flag
);
reg [31 : 0] ALU_Result;
wire [8 : 0] tmp;
assign ALU_Out = ALU_Result; // ALU OUT
assign tmp = {1'b0, 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
```


Control0
Control1
32–bit
AND
32–bit
OR
32–bit
ADD/SUB
A
B
32
32
32
32
32
32
32
Result

think this stuff is cool?

Stanford | ENGINEERING

Electrical Engineering



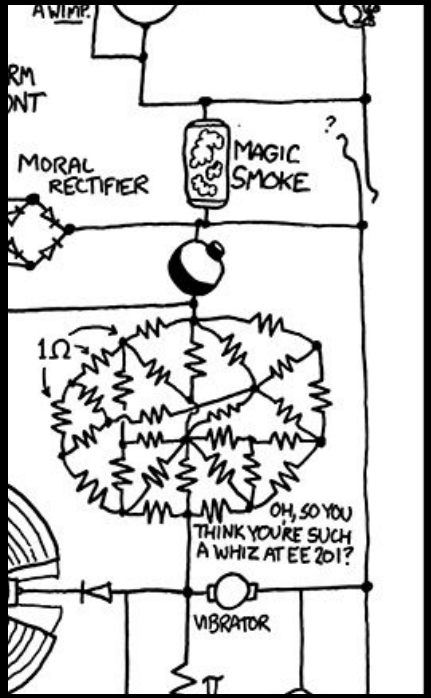
classes:

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

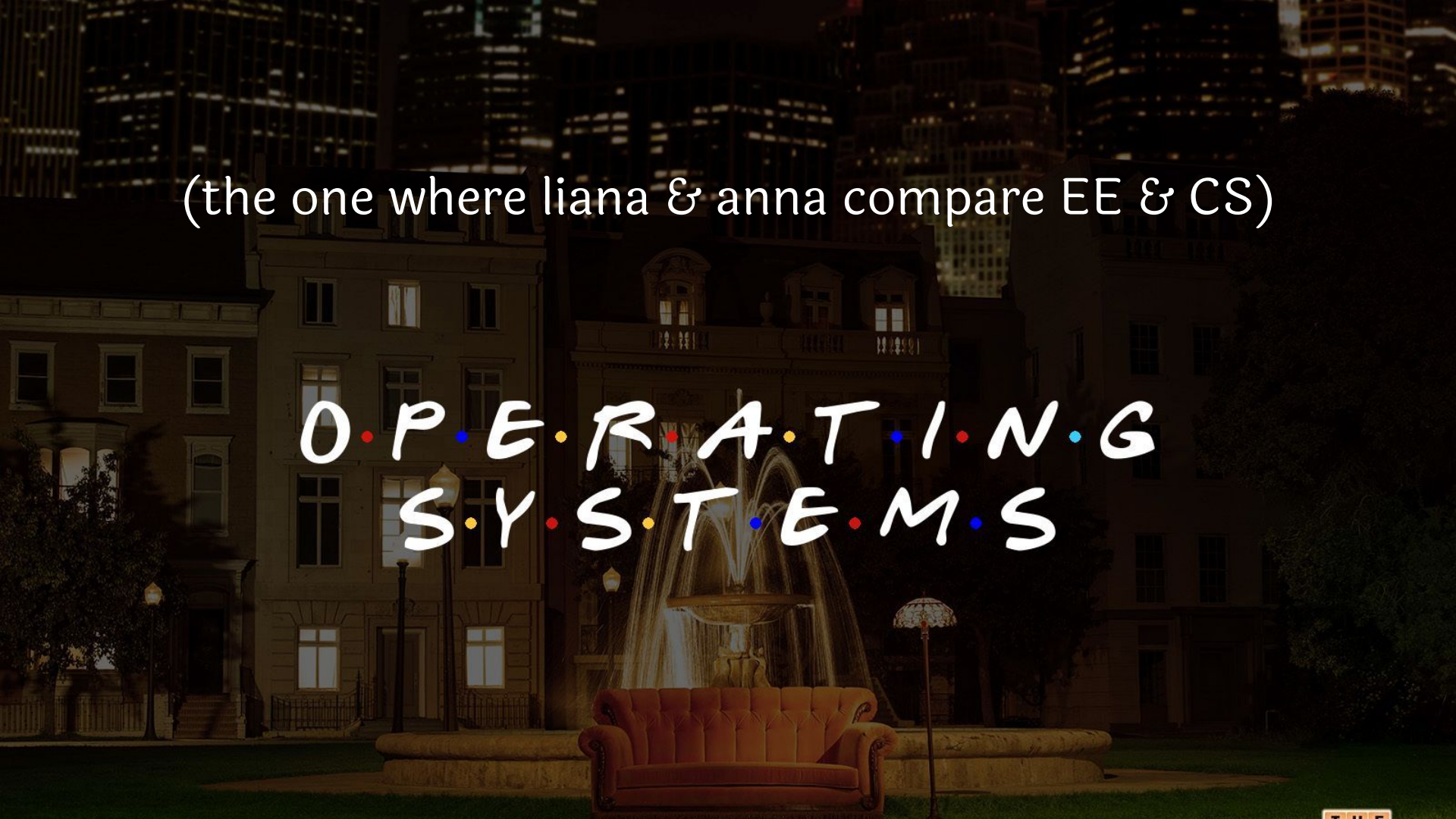
EE 180: Digital Systems Architecture

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!

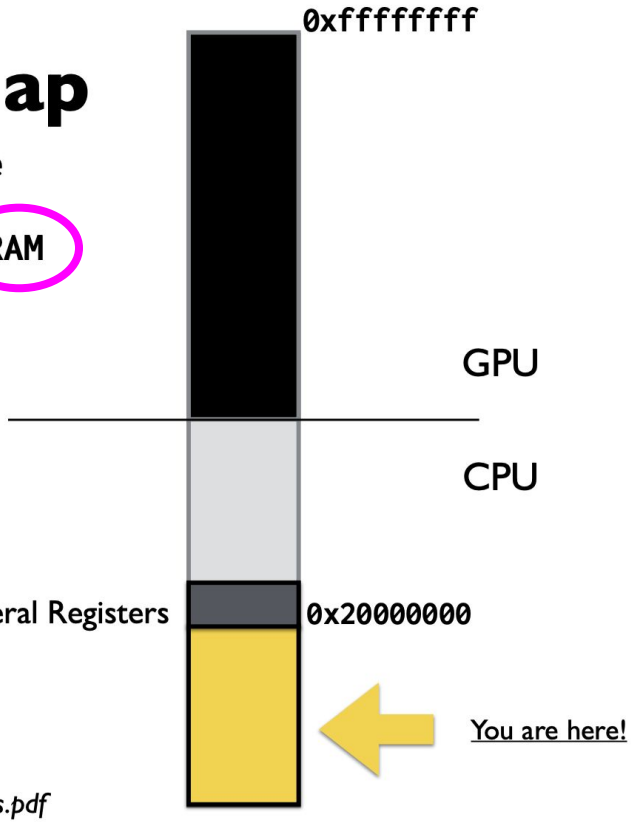
EE109: “Digital Systems Design Lab”
EE271: “Introduction to VLSI Systems”
EE 272A/B: “Design Projects in VLSI Systems I/II”
EE 273: “Digital Systems Engineering”
EE 282: “Computer Systems Architecture”


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

Memory Map

32-bit address space

512 MB of physical RAM

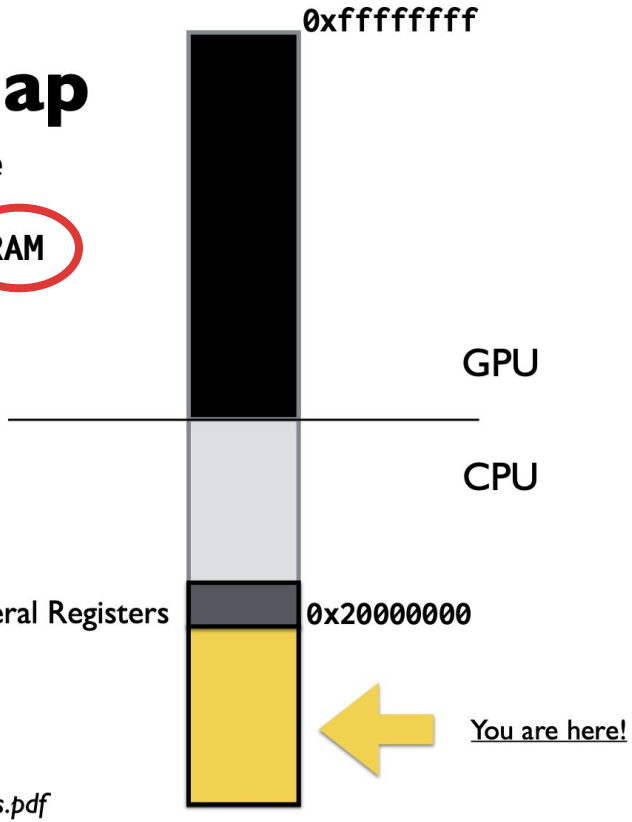


Ref: BCM2835-ARM-Peripherals.pdf

Memory Map

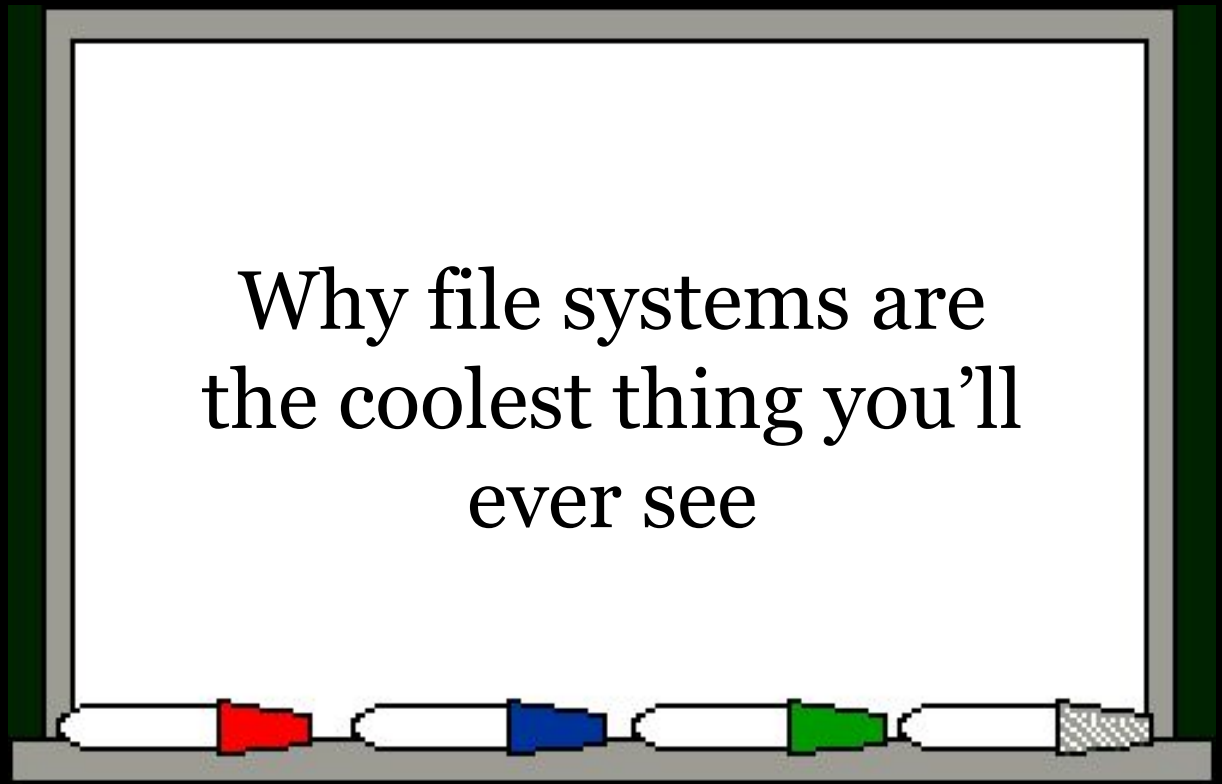
32-bit address space

512 MB of physical RAM

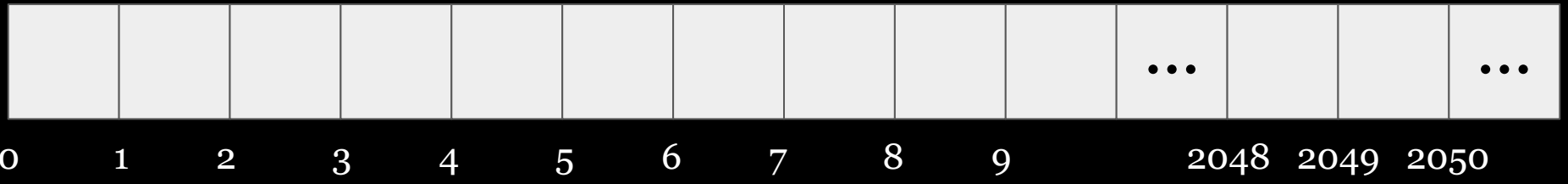


Ref: BCM2835-ARM-Peripherals.pdf

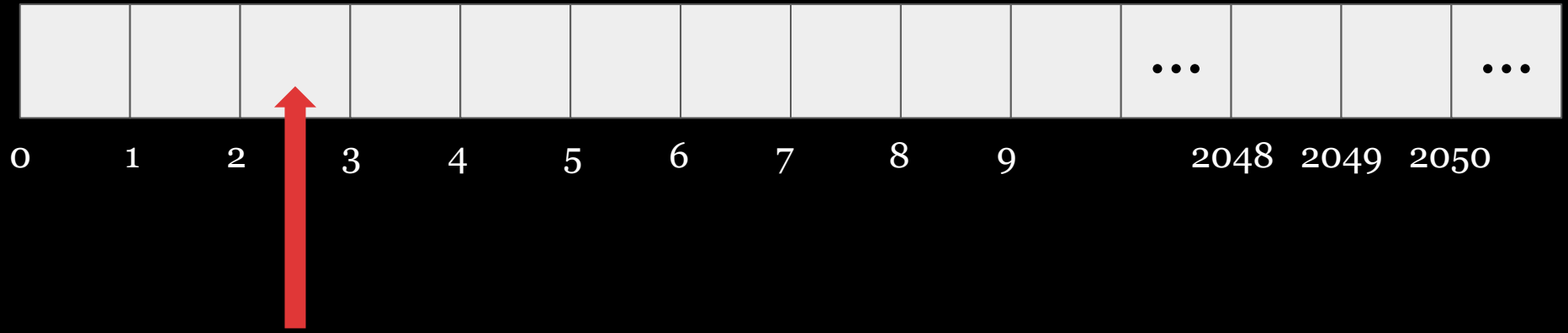
What about the HARD DRIVE?


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

The hard drive is not byte-addressable



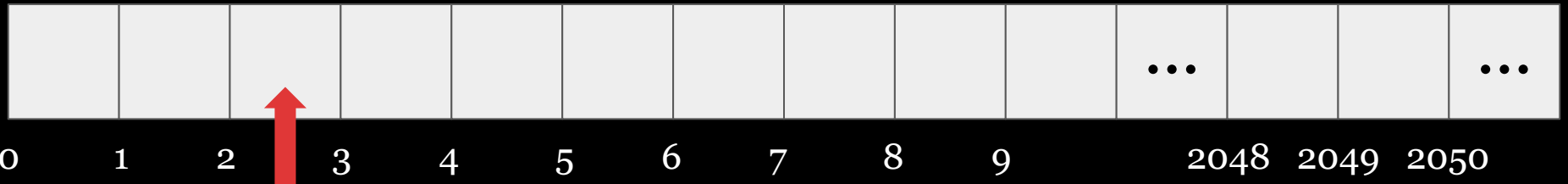
The hard drive is not byte-addressable



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

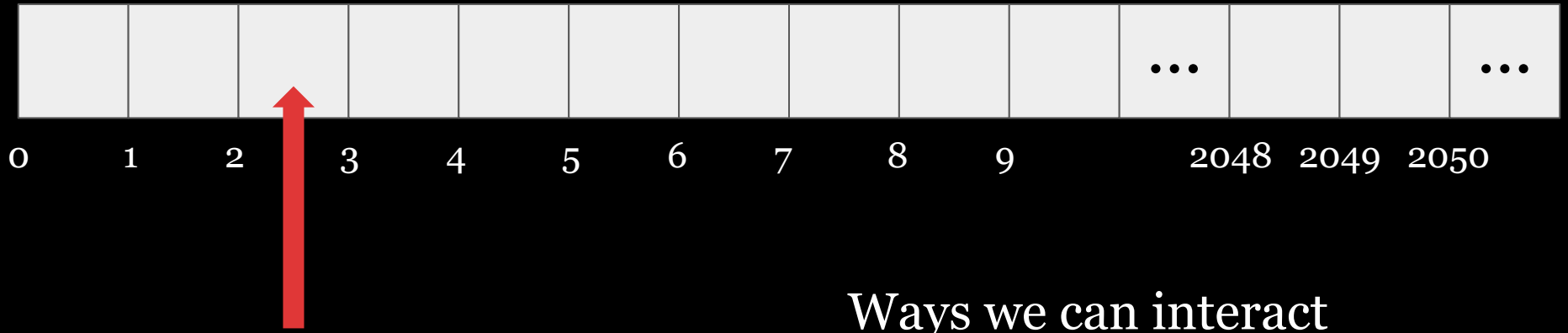


Each sector is
512 bytes*

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

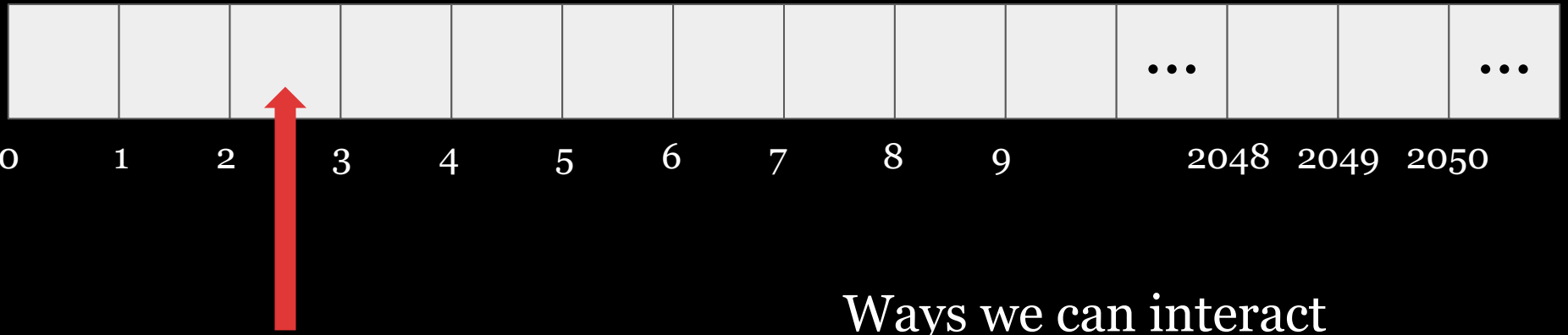


Each sector is
512 bytes*

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*

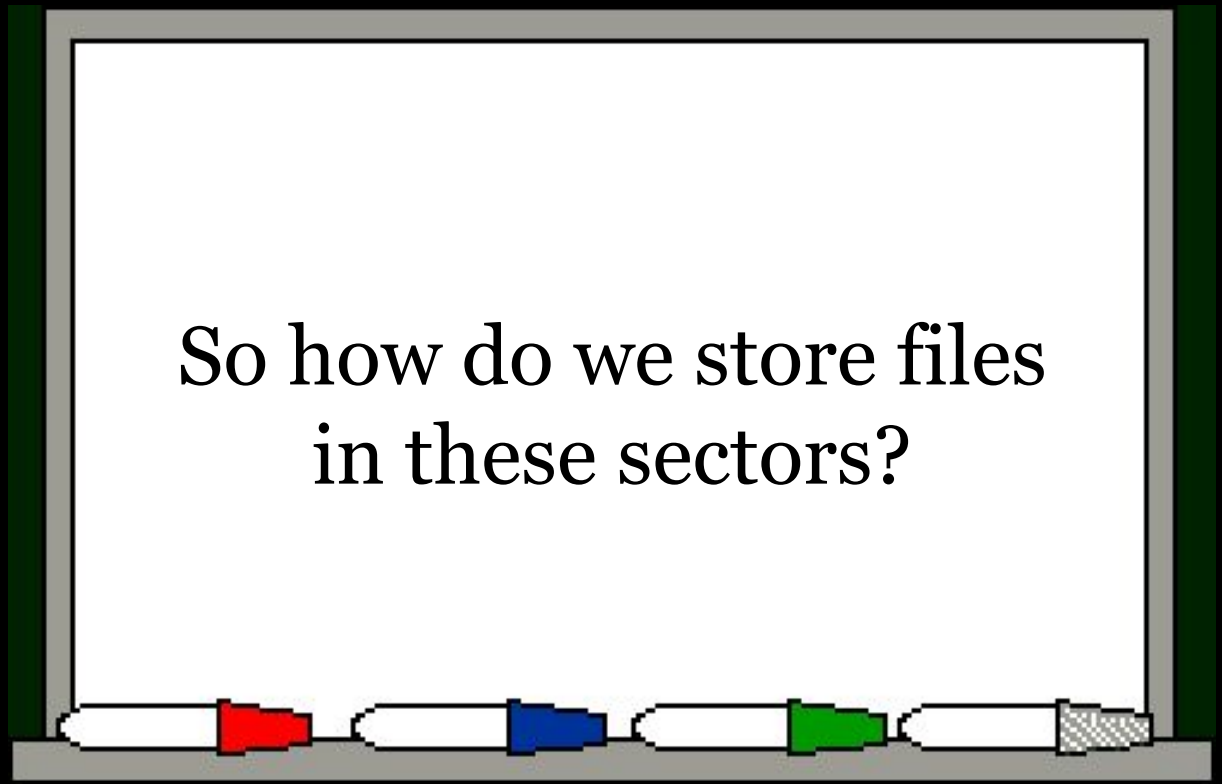
Ways we can interact
with the hard drive:

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)

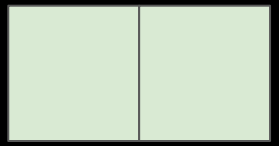
```
code code code code
code buf[256] lots
of code %s if else
va_list code code
code snprintf code
code code code code
```

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

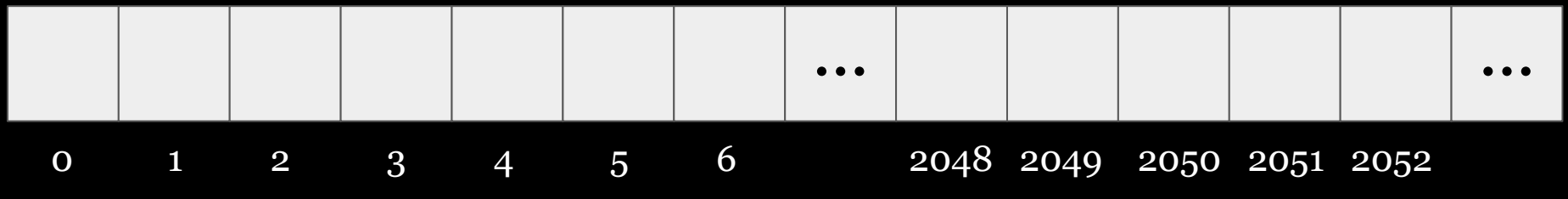


printf.c
(1024 bytes)

```
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



printf.c
(1024 bytes)

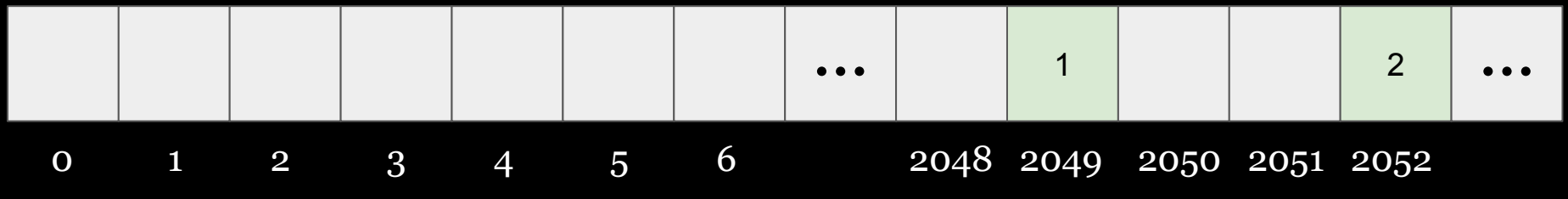
```
code code code code
code buf[256] lots
of code %s if else
va_list code code
code snprintf code
code code code code
```

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

sectors:
2049, 2052

printf.c data

1	2



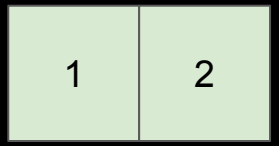
printf.c
(1024 bytes)

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

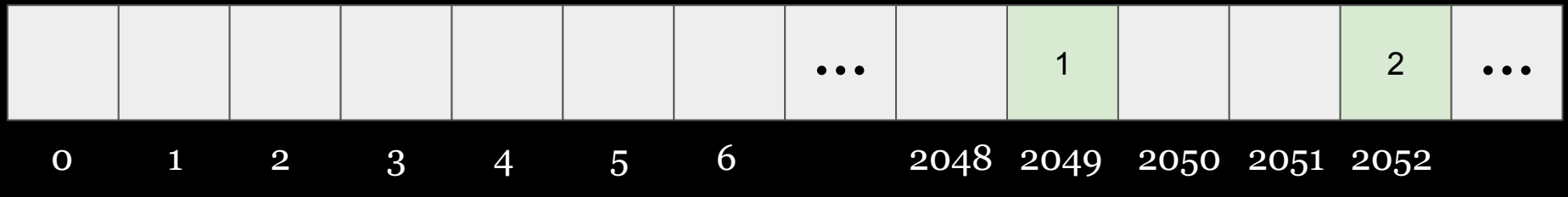
printf.c
size: 1024
file type: regular
permissions: ...

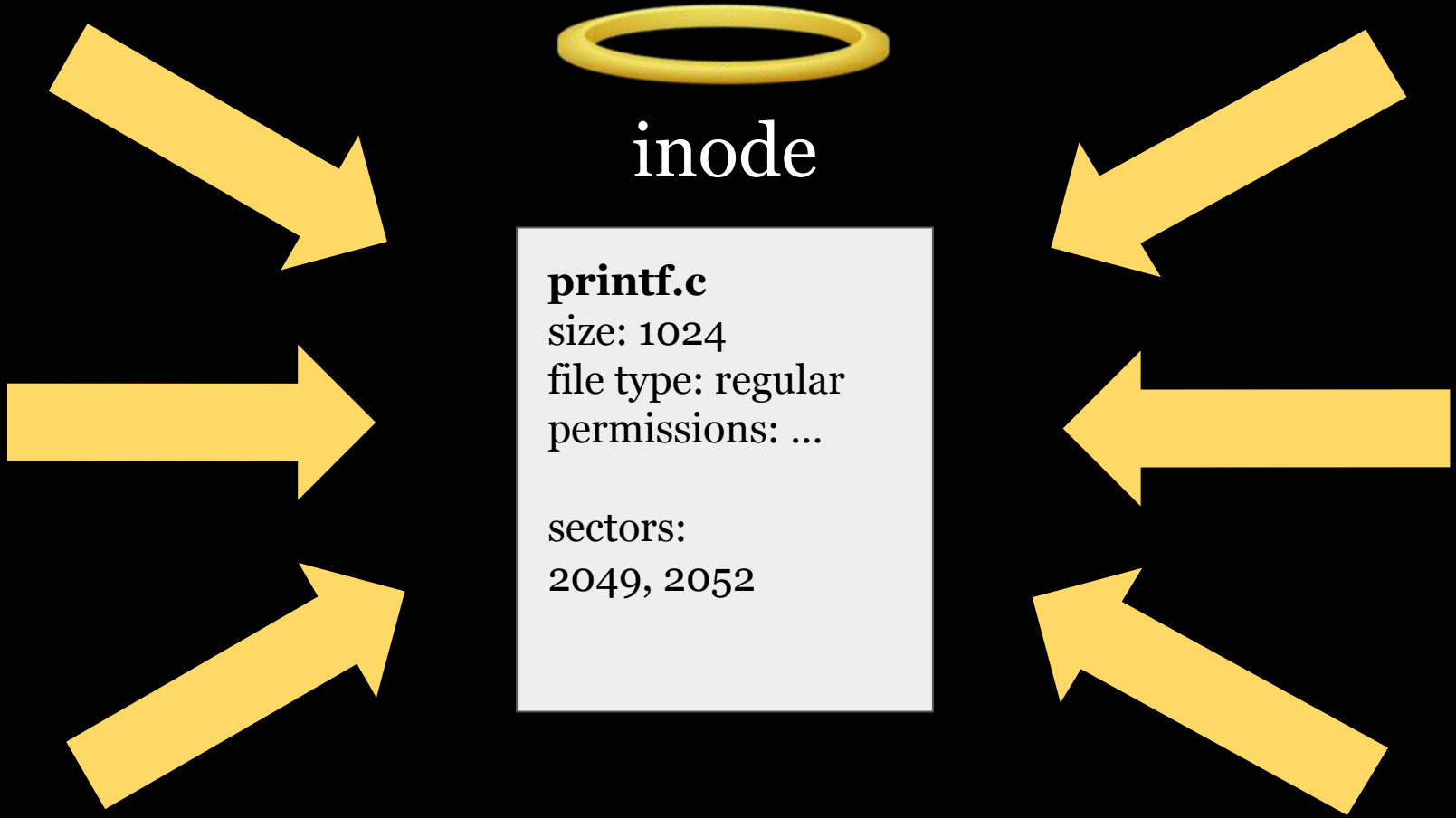
sectors:
2049, 2052

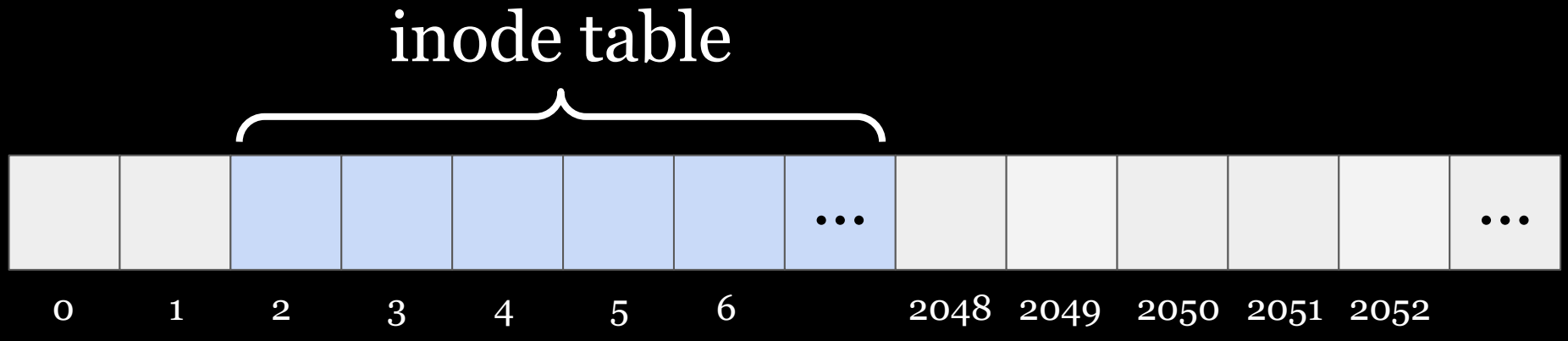
printf.c data

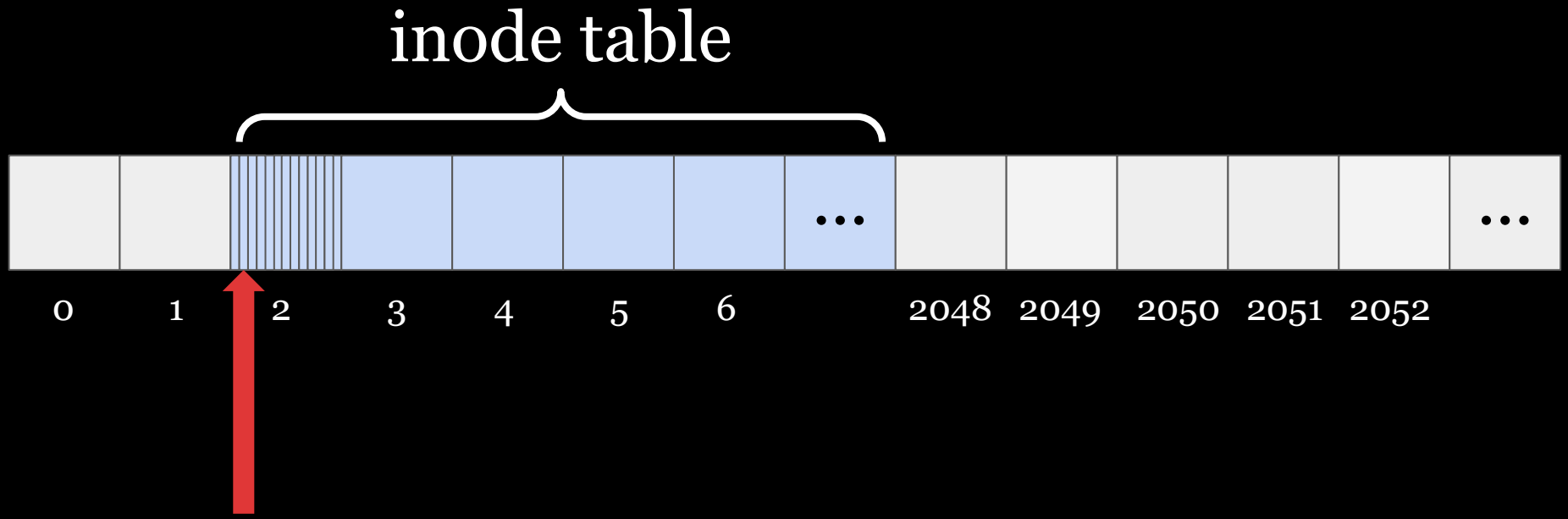


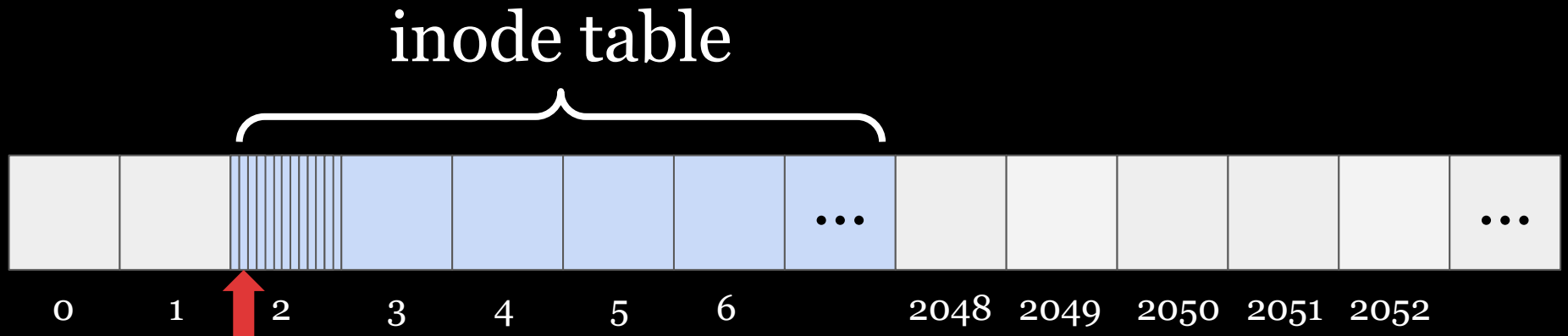
What if our file size isn't a perfect multiple of 512?




inode
printf.c
size: 1024
file type: regular
permissions: ...
sectors:
2049, 2052

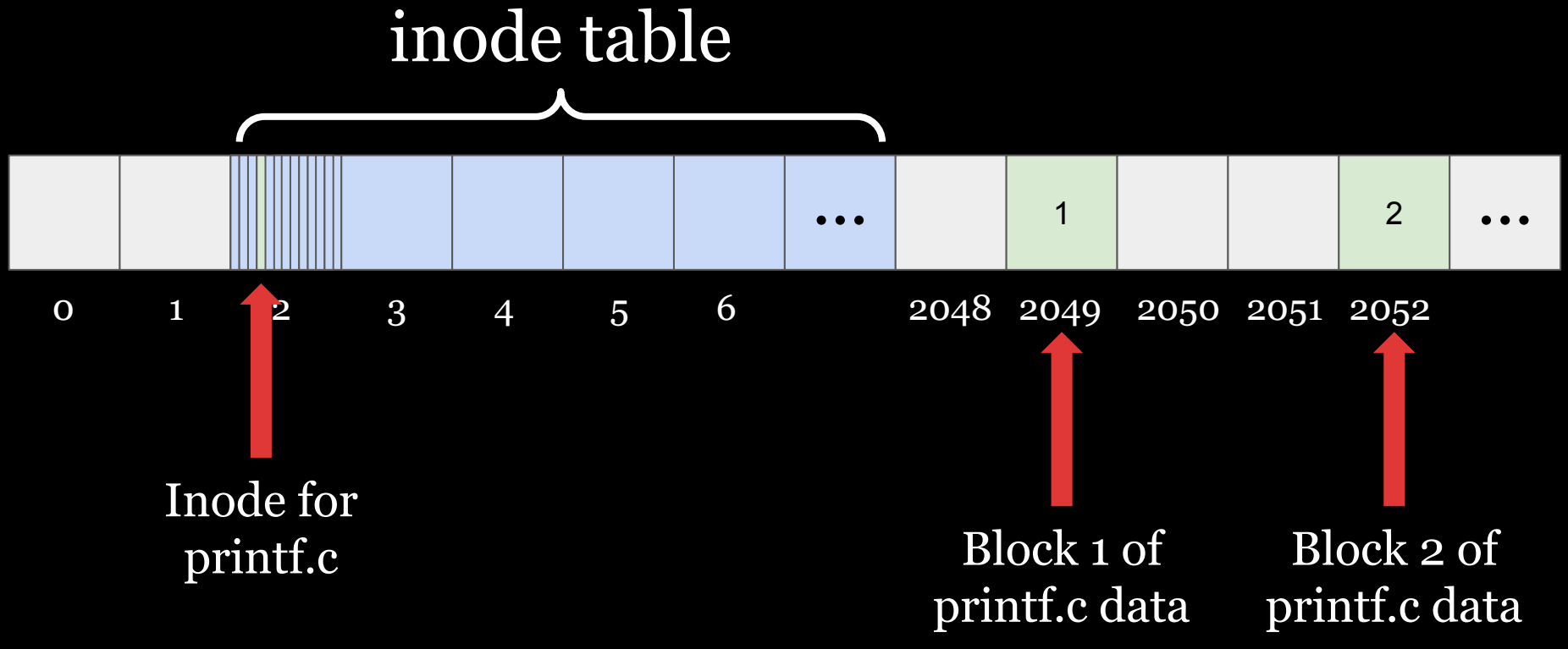

inode table
...
...
0
1
2
3
4
5
6
2048
2049
2050
2051
2052

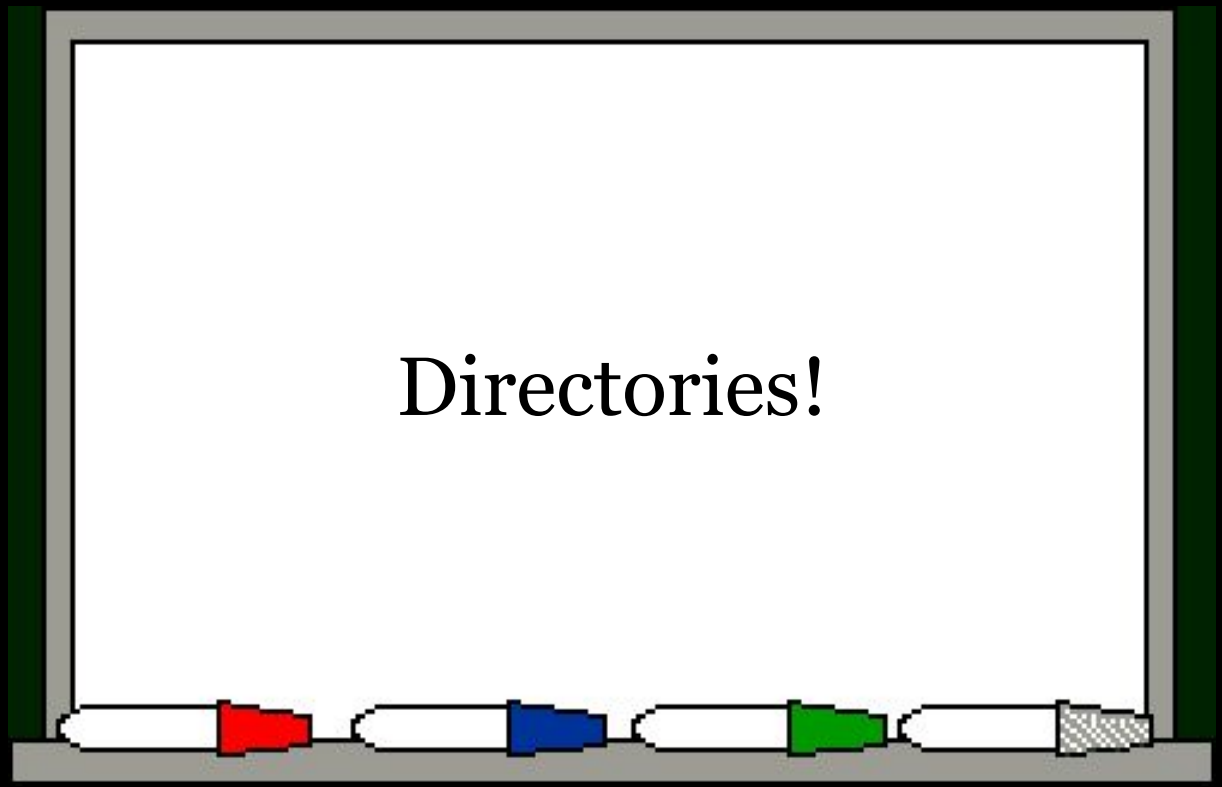

inode table
...
...
0
1
2
3
4
5
6
2048
2049
2050
2051
2052
Each sector
holds 16 inodes



Each sector holds 16 inodes

Inode numbering: Sector 2 has inodes 1-16, Sector 3 has inodes 17-32, etc.


inode table
...
1
2
...
0
1
2
3
4
5
6
2048
2049
2050
2051
2052
Inode for printf.c
Block 1 of printf.c data
Block 2 of printf.c data


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

Sector 2048:

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

File name*

*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

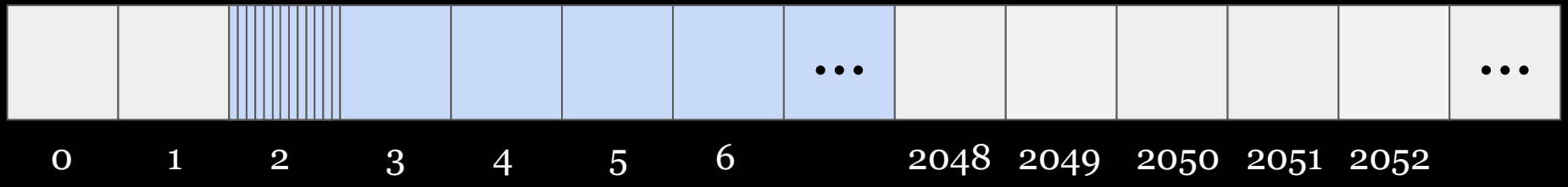
. is the current directory, .. is the parent directory

File name*

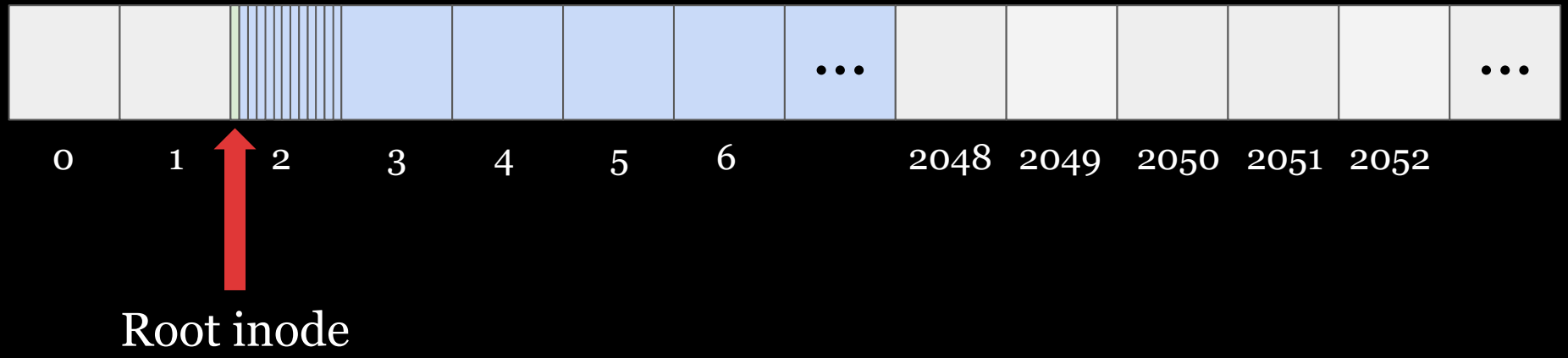
The inode number for the file

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

Let’s walk through!!

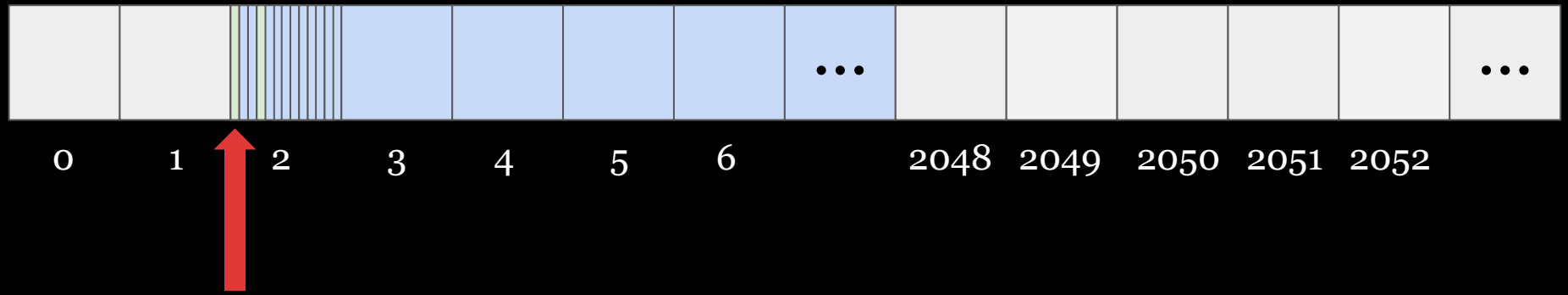


Let’s walk through!!



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

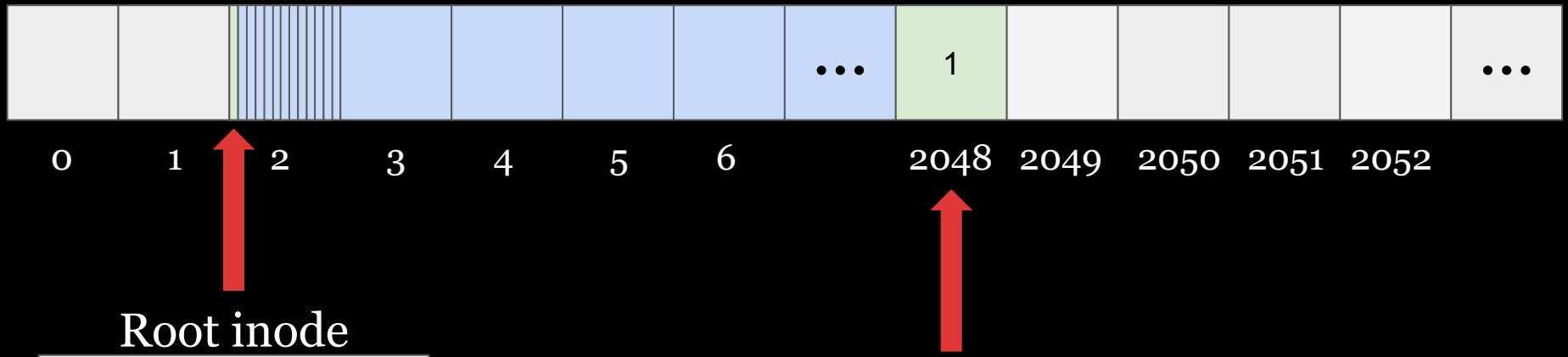
Let’s walk through!!



/
size: 64
file type: directory
permissions: …

sectors:
2048

Let’s walk through!!



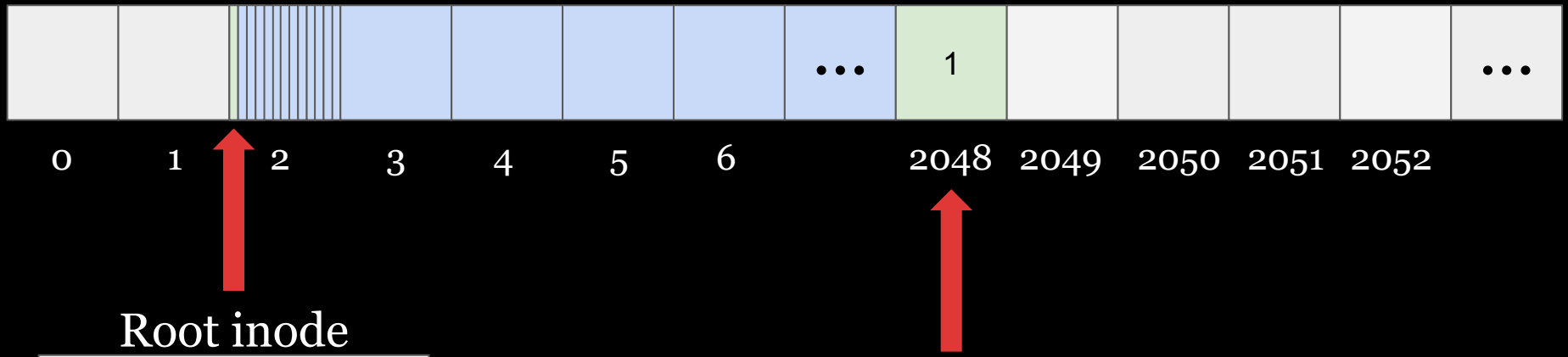
Root inode

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

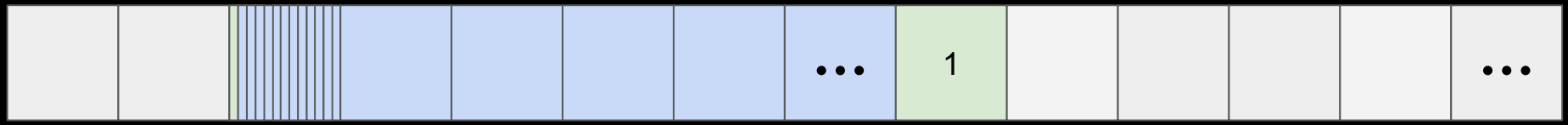
sectors:
2048

Information about what’s in the root directory

Let’s walk through!!



Let's walk through!!



Root inode

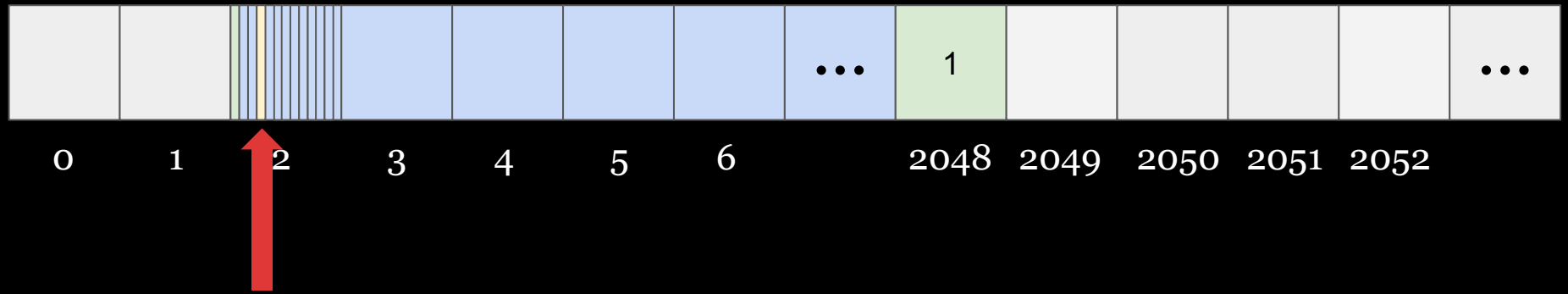
/
size: 64
file type: directory
permissions: ...

sectors:
2048

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

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

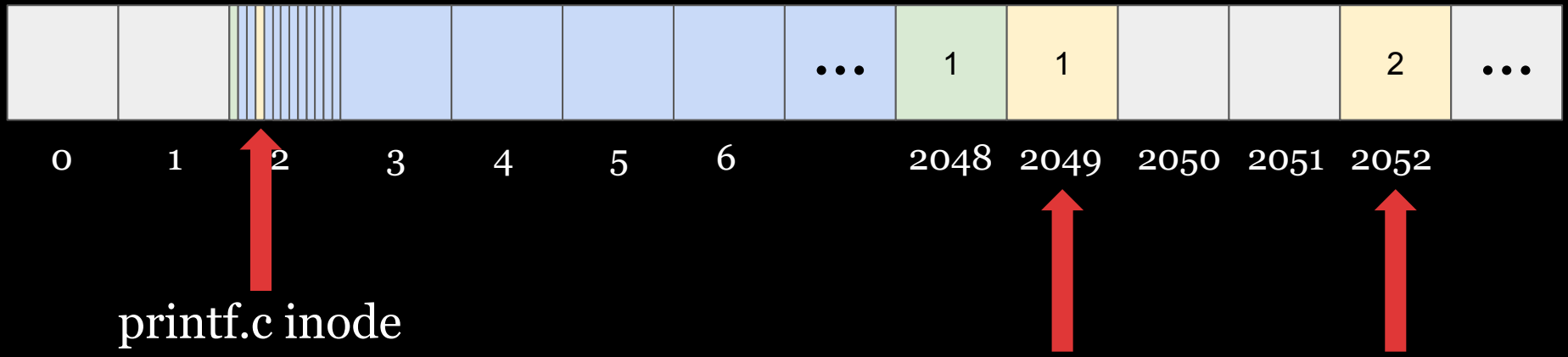
Let’s walk through!!



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

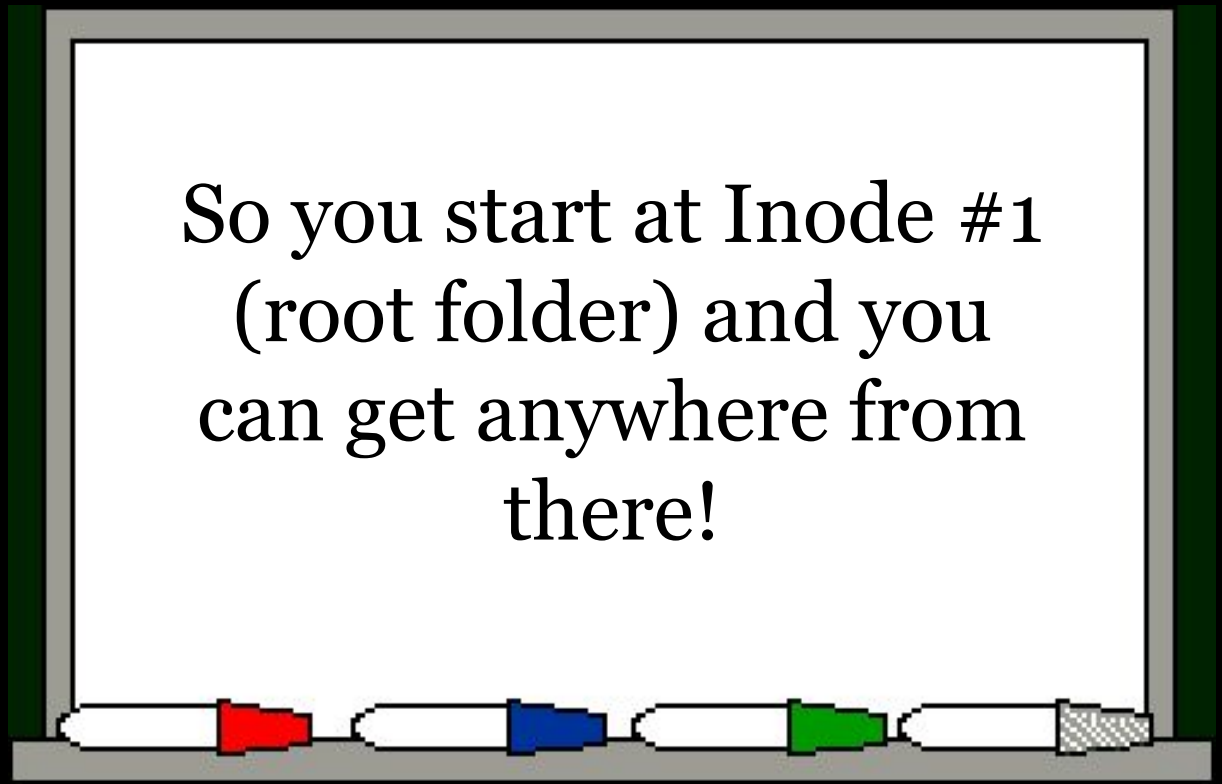
sectors:
2049, 2052

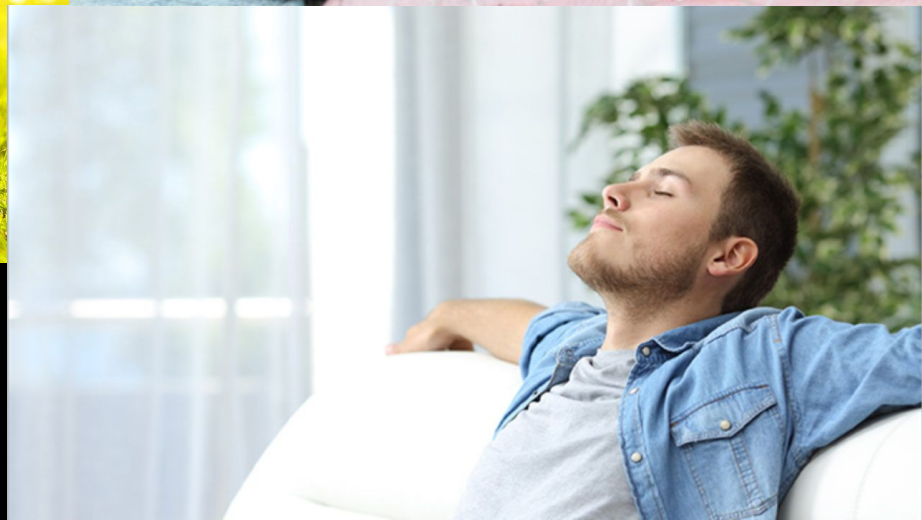
Let’s walk through!!



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

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

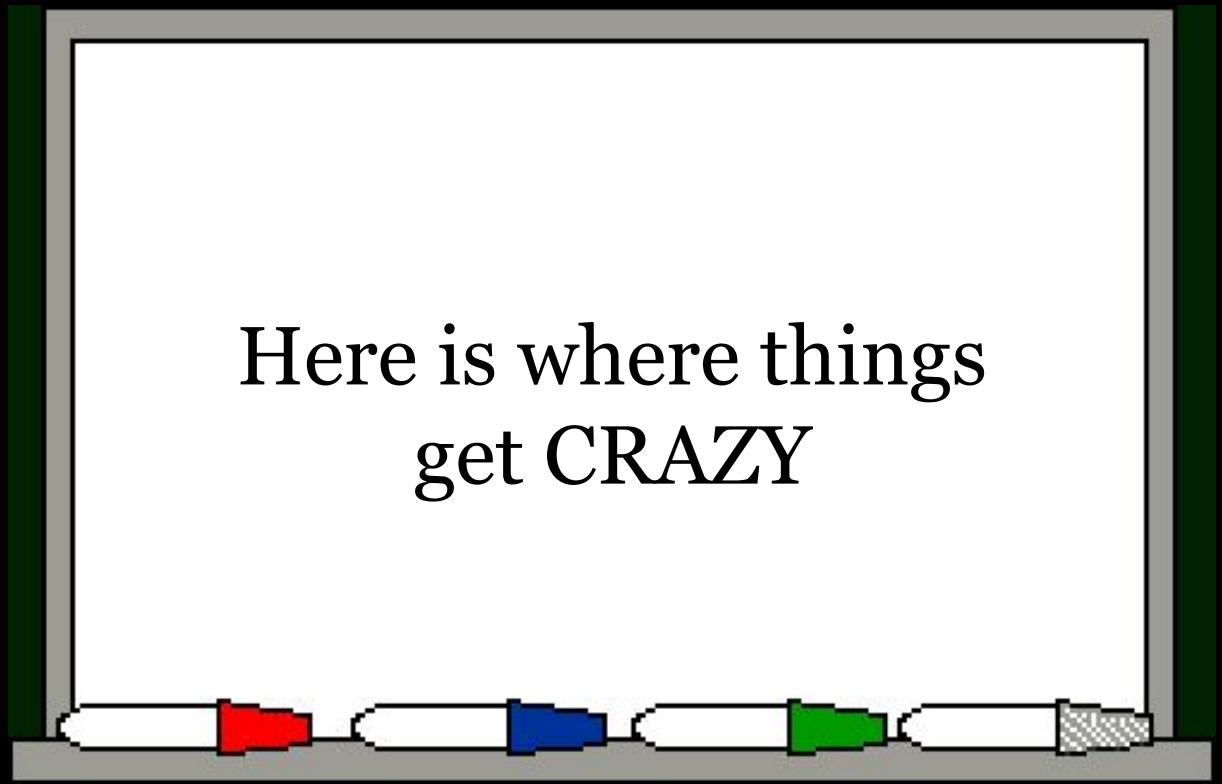
printf.c
size: 1024
file type: regular
permissions: ...

sectors:
2049, 2052

inode

This is 512*8 + 2 (so we need 9 blocks)

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


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:

2049	2052	2057	4923	2938
2293	2939	2999	8392	…

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:

2131	3281	4811	4812	4381
3821	3829	3813	3391	3911
313	...			

printf.c
size: a lot of bytes
file type: regular
sectors:
3013

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

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

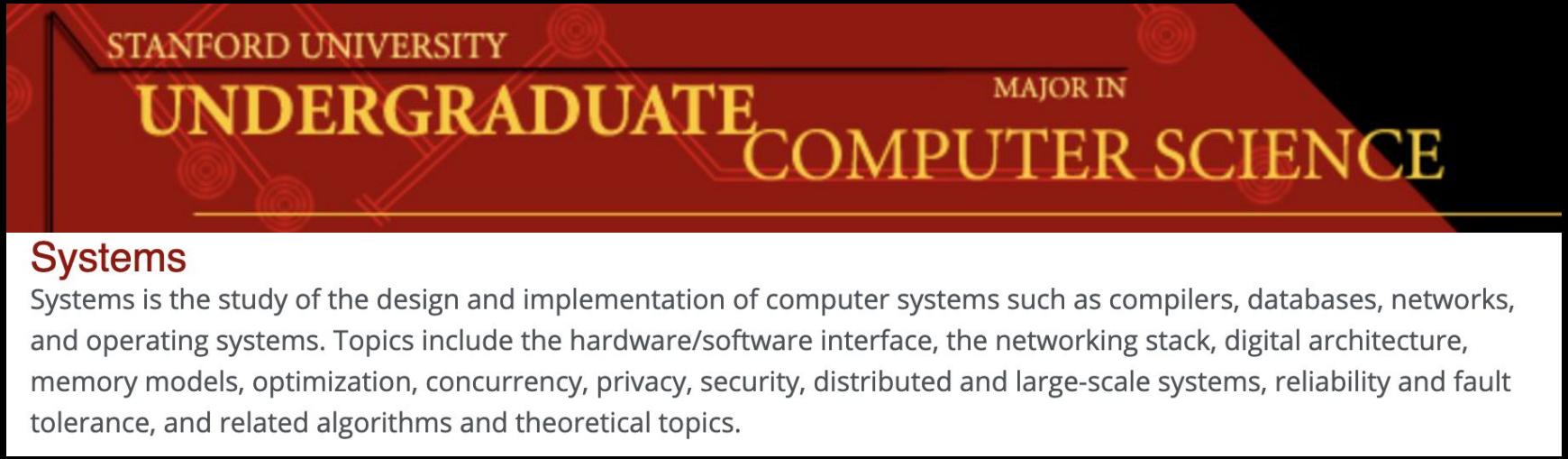
Sector 2049:

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:

2131	3281	4811	4812	4381
3821	3829	3813	3391	3911
313	...			

think this stuff is cool?



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

classes:

1

CS 110: Principles of Computer Systems

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 rasberry 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.