Sequential Logic & Memory

Introduction to Computer Systems, Fall 2024

Instructors: Joel Ramirez Travis McGaha

Head TAs: Adam Gorka, Daniel Gearhardt, Ash Fujiyama, Emily Shen

TAs:

Ahmed Abdellah
Angie Cao
August Fu
Caroline Begg
Cathy Cao
Claire Lu
Eric Sungwon Lee
Ethan Weisberg
Garrett O'Malley Kirsch
Hassan Rizwan
Iain Li
Jerry Wang
Juan Lopez
Keith Mathe
Maya Huizar
Meghana Vasireddy
Perrie Quek
Sidharth Roy
Sydnie-Shea Cohen
Vivi Li
Yousef AlRabiah

Poll Everywhere

pollev.com/cis2400

- How are you? Any Questions?

Logistics

- Midterm
  - Covers material all the up to Lecture 10!
  - Double Sided Cheat Sheet of Paper is permissible
- HW 5 is Due Tomorrow!
  - Ask for an extension before hand if you already know you need it.
  - Easier to respond at 6PM than it is at 2AM.

Poll Everywhere

pollev.com/cis2400

Question for Everyone

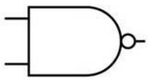
- If recitation next week was a Midterm Review Session, would you prefer that?

Lecture Outline

- Review
  - D Latch & Clock
  - D Flip Flops
- Registers
- Memory at a high level
- Memory using flip flops
- Memory Hierarchy

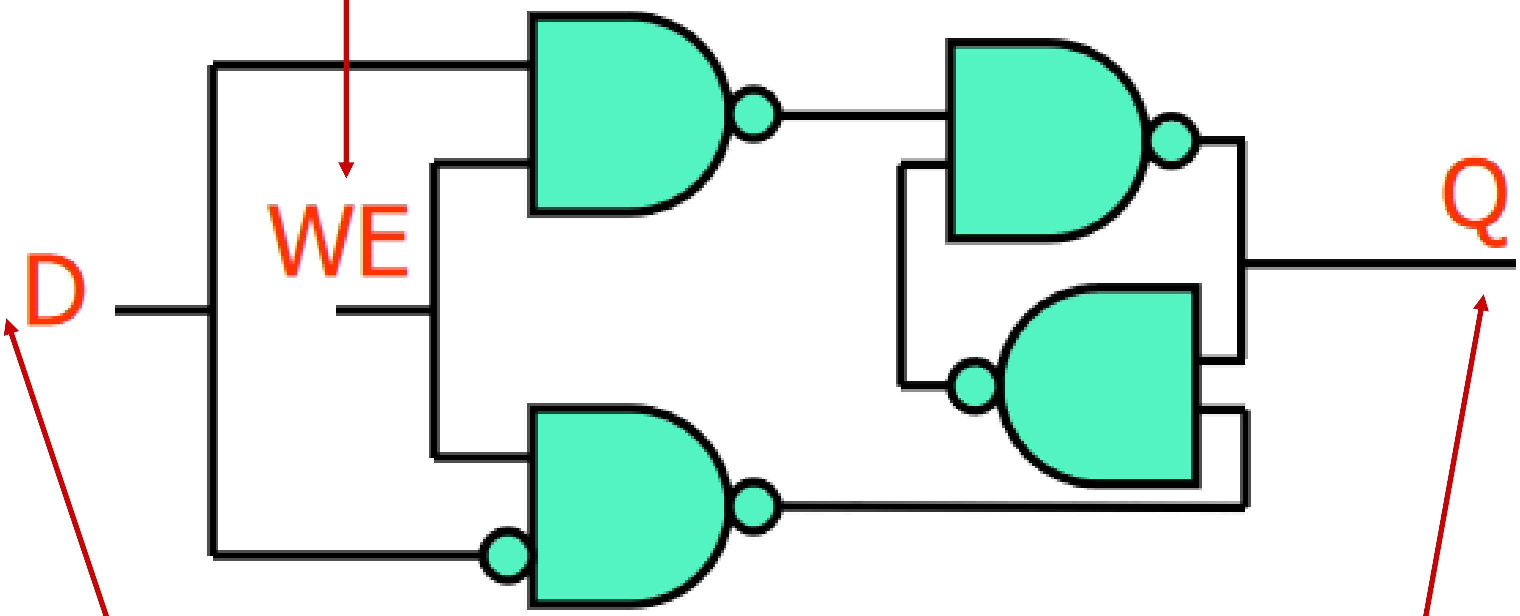
D Latch

NAND



A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

WE is "*Write Enabled*".
If WE is set to 0, the value of D isn't Saved.



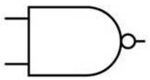
D is the data we are trying to "Save" in the Latch.

Q is the value stored in the Latch.

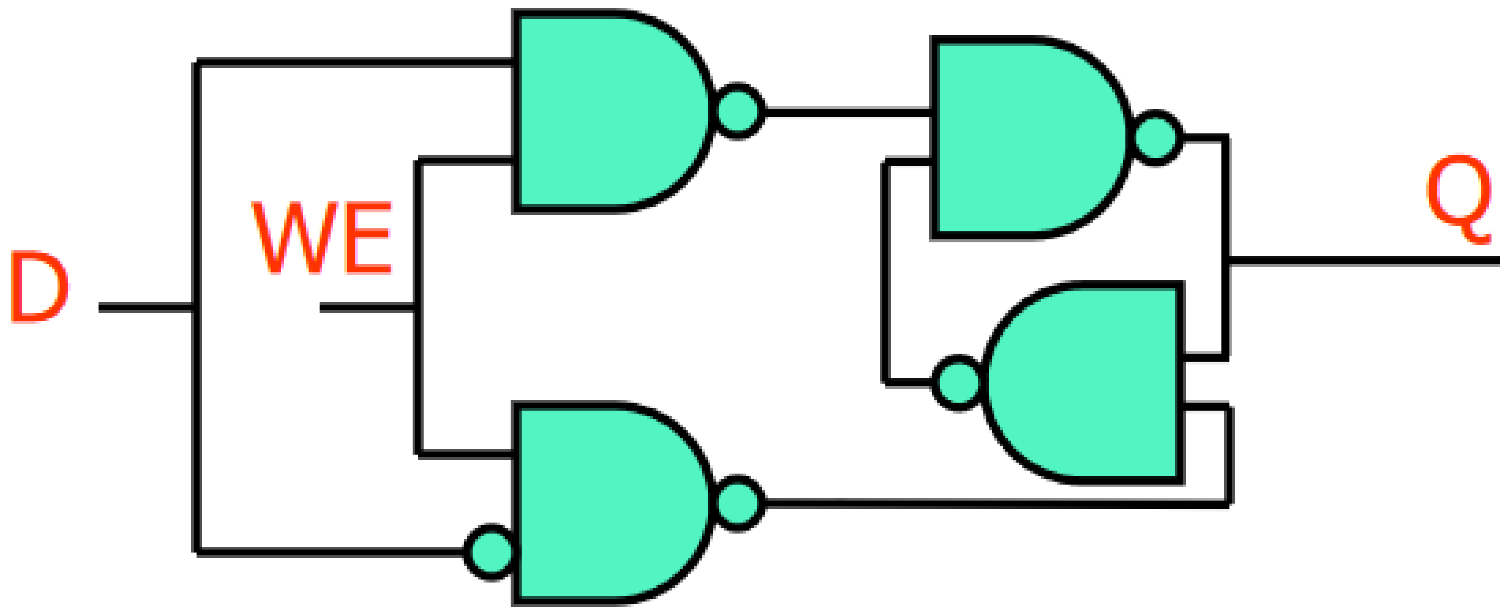
In D Latch terms, ***Q is set to the value of D when WE is 1.***

Some Verbiage

NAND



A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

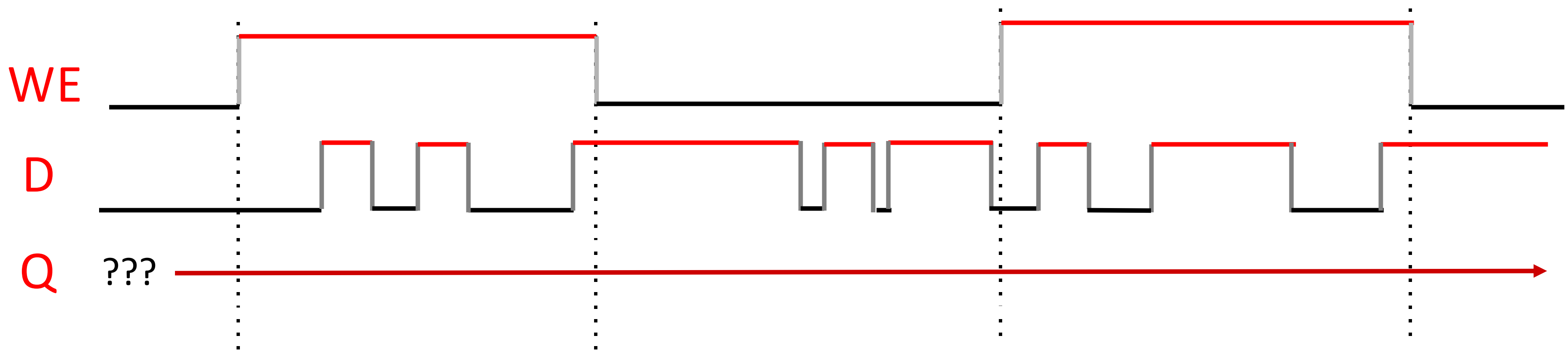


D is 1 equivalent to "D is High"

D is 0 equivalent to "D is Low"

Timing Diagrams Revisited

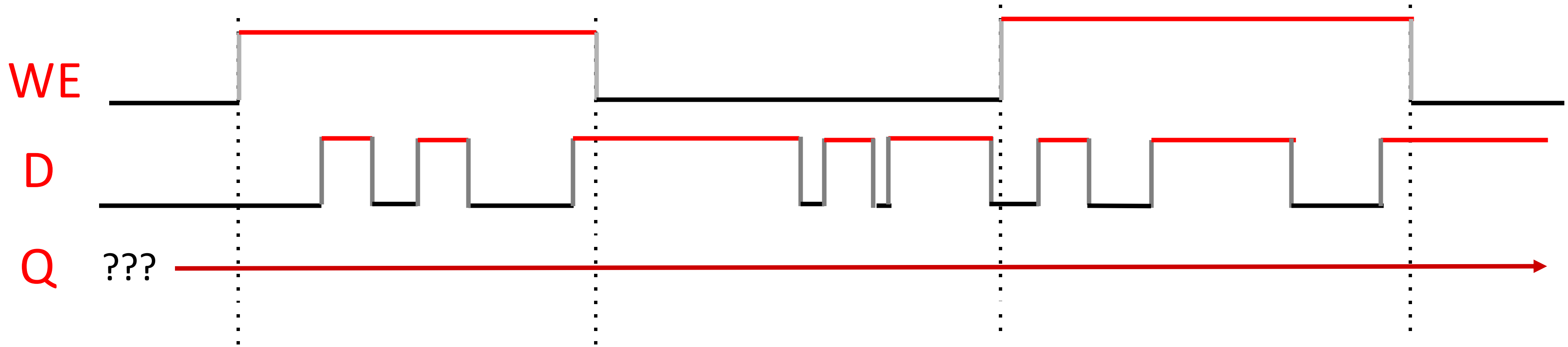
- Diagram to represent how signals (outputs) change



Poll Everywhere

pollev.com/cis2400

- How many times is Q set to be 1 (i.e. High)?



A. 4

B. 5

C. 6

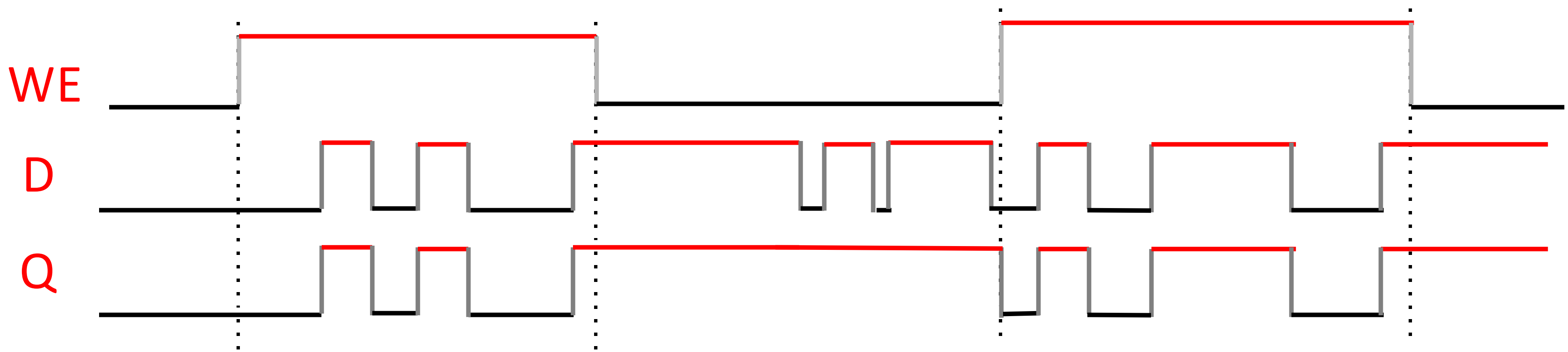
D. 7

E. I can't tell

Poll Everywhere

pollev.com/cis2400

- How many times is Q set to be 1 (i.e. High)?



A. 4

B. 5

C. 6

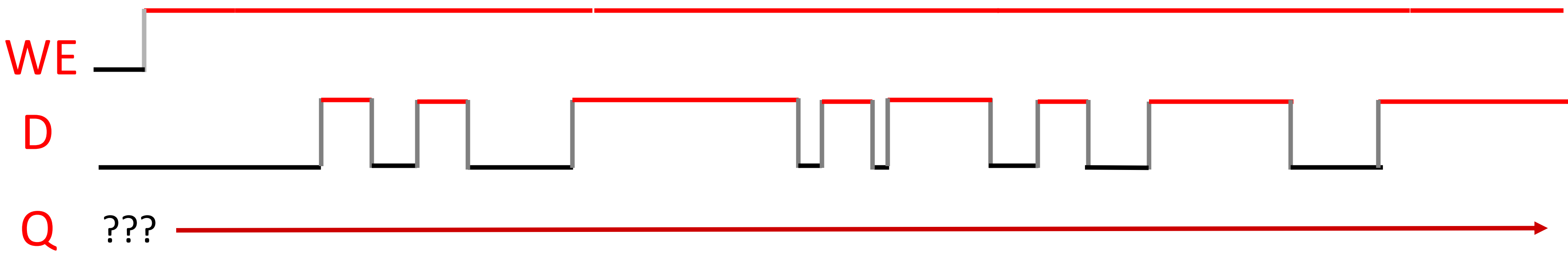
D. 7

E. I can't tell

Poll Everywhere

pollev.com/cis2400

- How many times is Q set to be 1 (i.e. High)?



A. 6

B. 7

C. 8

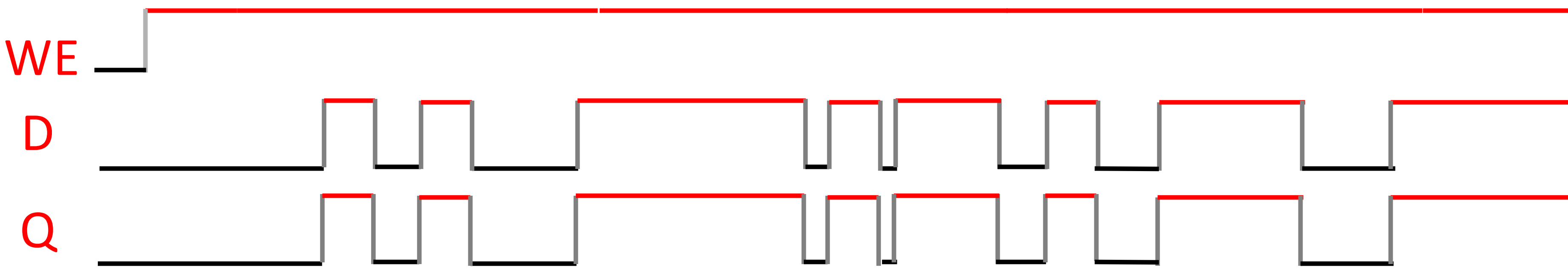
D. 9

E. I can't tell

Poll Everywhere

pollev.com/cis2400

- How many times is Q set to be 1 (i.e. High)?



A. 6

B. 7

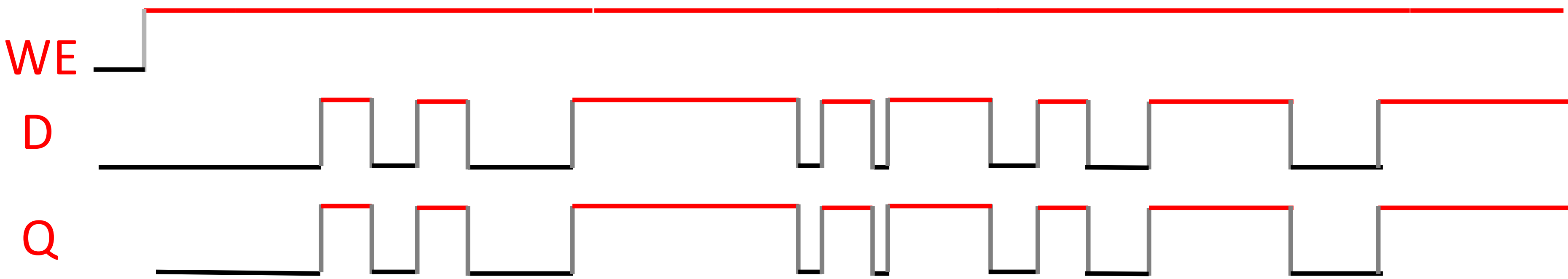
C. 8

D. 9

E. I can't tell

Transparency

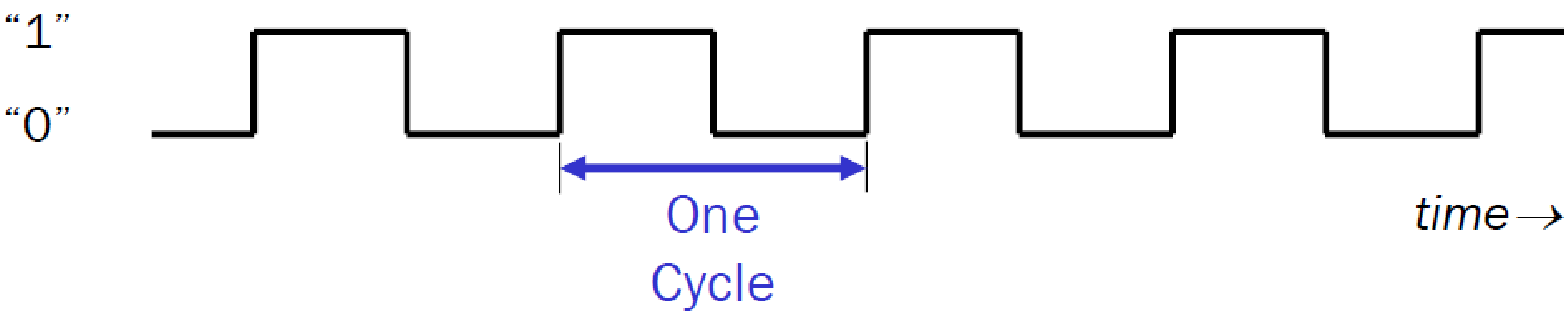
- WE being 1 sets Q equal to D every time!



If *WE* were to be held at a high signal indefinitely,
this would be a Fully Transparent D-Latch.

The Clock Revisted

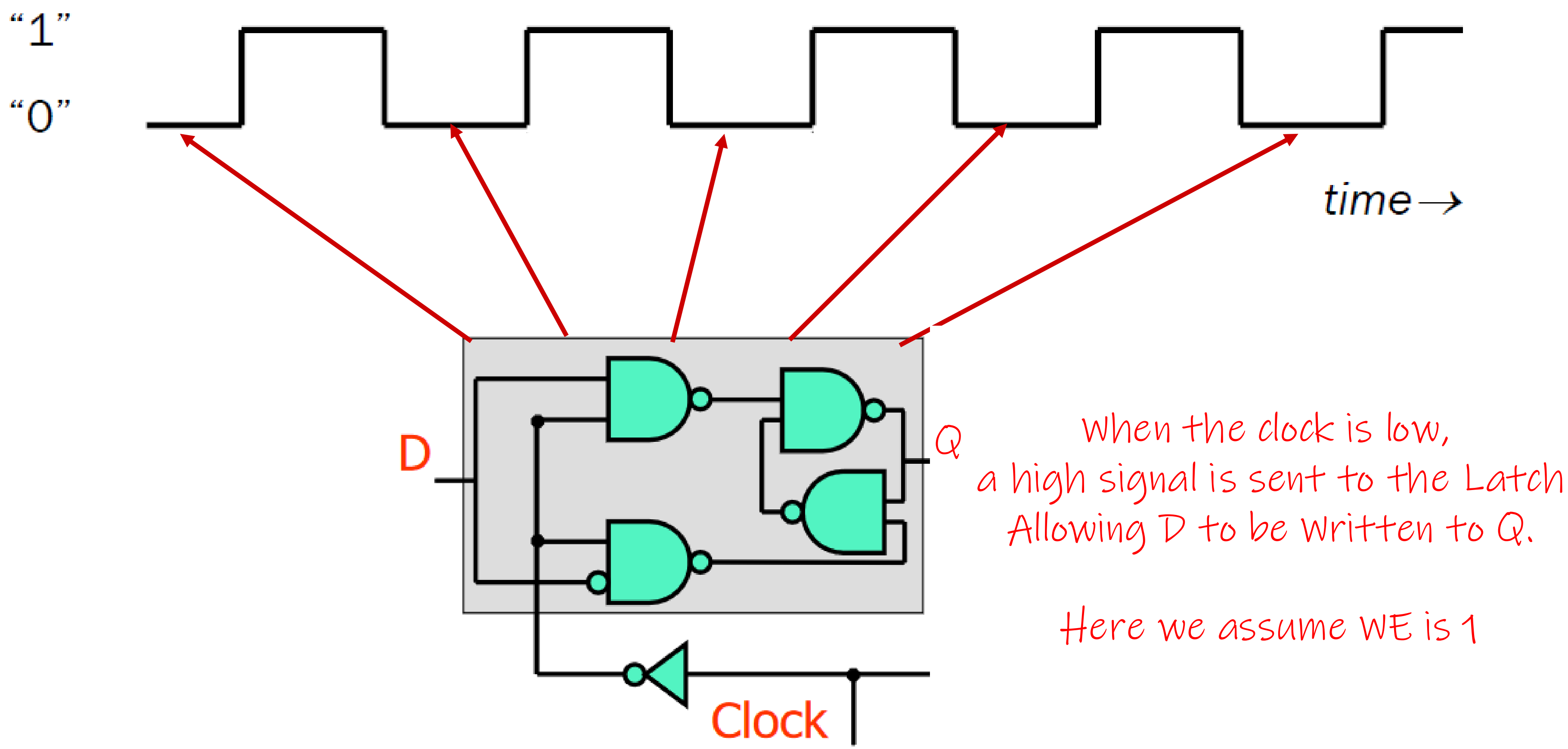
- A regular up & down signal with constant Period



- A **clock** in tandem with **WE** can change when the D-Latch is ***Transparent.***

The Clock & Transparency

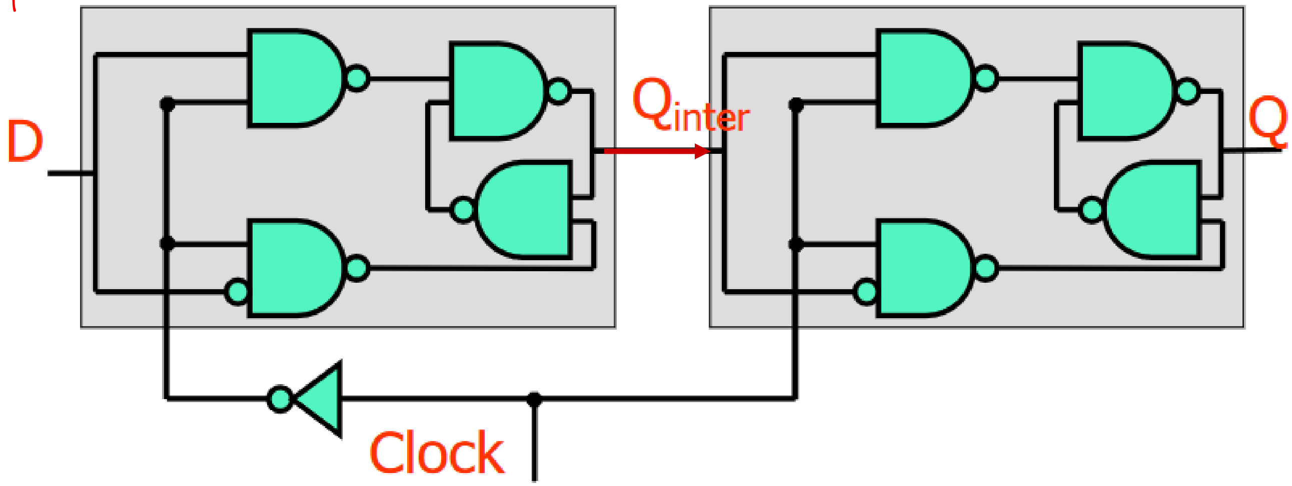
- A regular up & down signal with constant Period



D Flip Flop

Transparent Low

Transparent high



When the clock is LOW, Q_{inter} is set to D

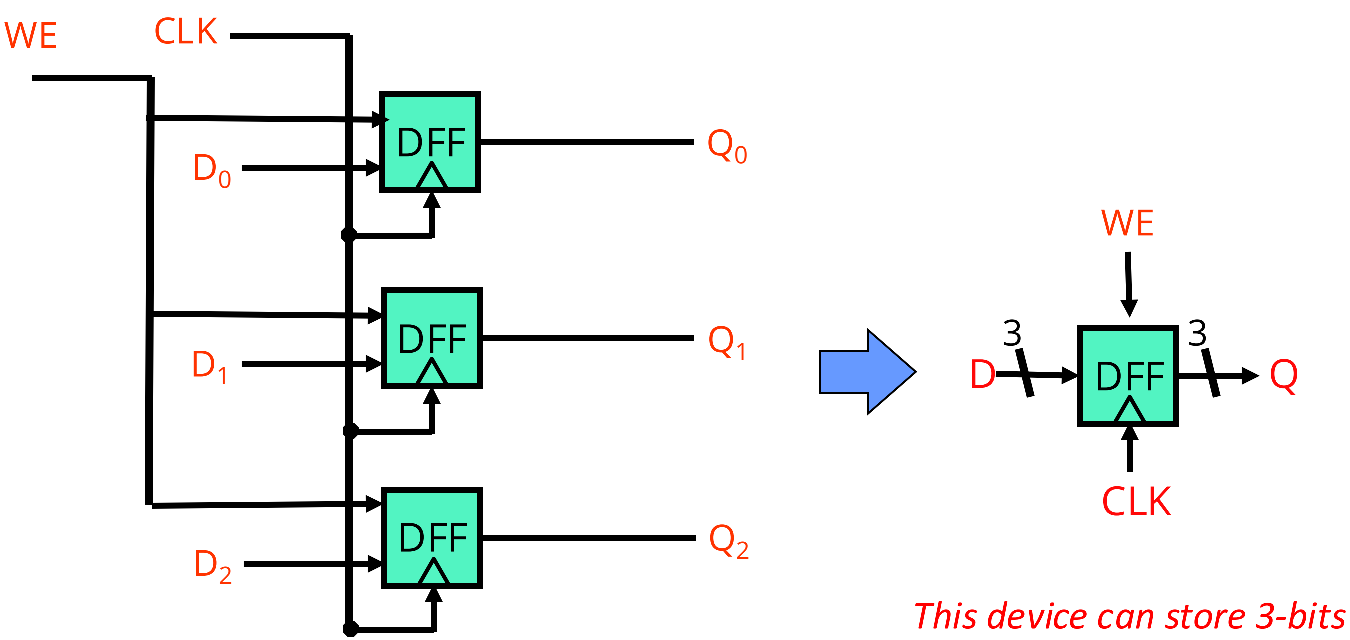
When the clock is High, Q is set to Q_{inter}

Lecture Outline

- Review
  - D Latch & Clock
  - D Flip Flops
- Registers
- Memory at a high level
- Memory using flip flops
- Memory Hierarchy

Register Made of Flip Flops

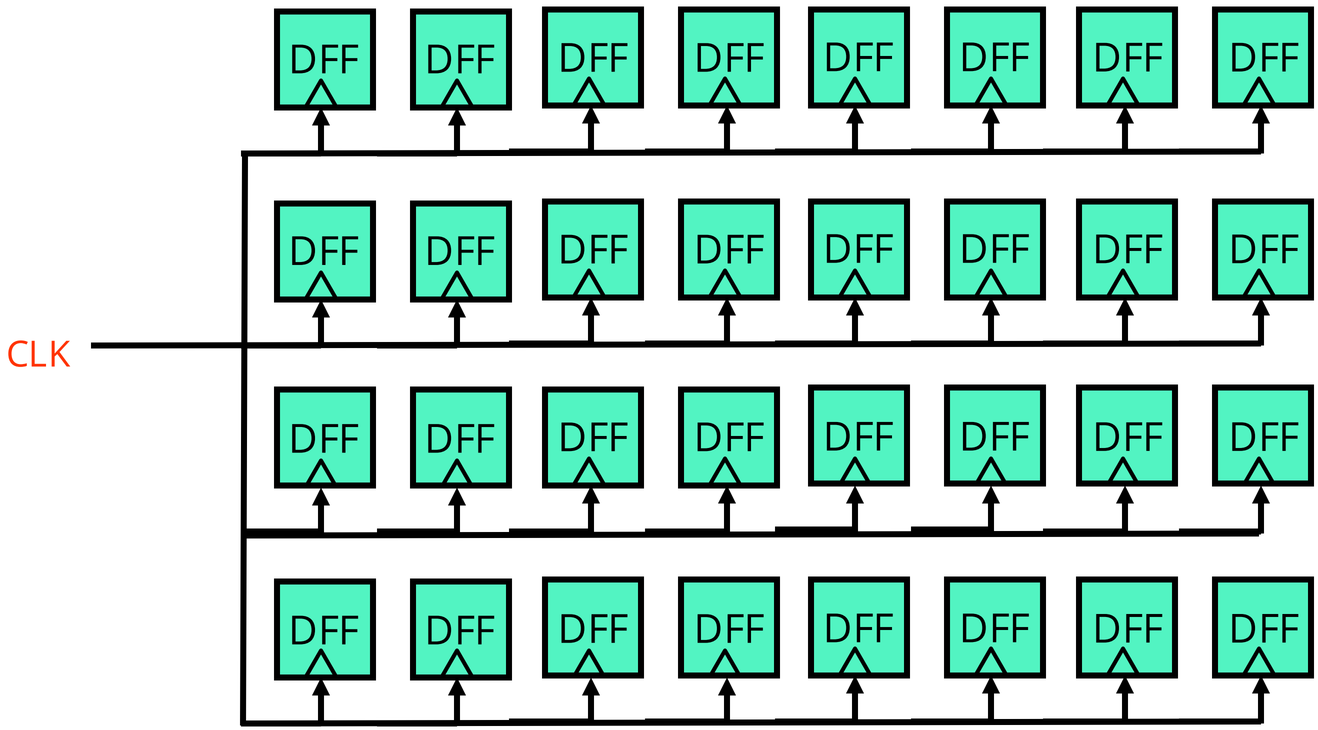
- A collection of D Flip-Flops, controlled by a common CLK signal can be called a register
  - A register is a fixed size multi-bit fast storage location used by a Processor. (More on this over the next few weeks)
  - Many different implementations; but made of same stuff



This device can store 3-bits!

Register Made of Flip Flops

- A collection of D Flip-Flops, controlled by a common CLK signal can be called a register
  - Here is a 32 bit register connected by CMN CLK



Lecture Outline

- Review
  - D Latch & Clock
  - D Flip Flops
- Registers
- Memory at a high level
- Memory using flip flops
- Memory Hierarchy

Memory as an array

- Memory is like a huge array…
  - Stores ***almost all*** the information needed to run a program (Code, variables, strings, …)
  - In a 64-bit machine, this array has **18,446,744,073,709,551,616** indexes
    - Pointers are 8 bytes (64 bits)
      - That's 2^{64} possible values
      - 0xFFFFFFFFFFFF0020 would be a *Full Address*.
  - An index corresponds to a location in memory that contains data
    - (An index in this context is called an **address**)

Memory as an array

- Memory is like a huge array…
  - A location in memory is of fixed size bits
    - Usually 8-bits.
      - RISC-V, x86, and Arm64
    - Previous Version of CIS2400 used LC4
      - ***USED 16 bits***
  - These addresses could be storing variables

```
int a = 0;
int b = 9;
```

Index # *(Address)*	Information *(Data)*
0	0xFFEC
1	0x000A
2	0xFFF1
3	0x0008
4	0xFFFF
5	0x0000
6	0x0102
7	0x0000
8	0xFF32
9	0x2400
10	0x0000
11	0x0009
12	0xF308
13	0x0000
14	0x0000

Memory as an array

- Address Space: The range of possible addresses. Usually, some power of 2.
  - In RISC-V, we can have either 2^{32} or 2^{64} addresses depending on the width of the registers.
  - (i.e. Is the Machine 32 or 64 bits?)
- Addressability: number of bits per location
  - 8-bits on modern computers
  - 8-bits for RISC-V

Index # *(Address)*	Information *(Data)*
0	0xFFEC
1	0x000A
2	0xFFF1
3	0x0008
4	0xFFFF
5	0x0000
6	0x0102
7	0x0000
8	0xFF32
9	0x2400
10	0x0000
11	0x0009
12	0xF308
13	0x0000
14	0x0000

Basic Memory Usage

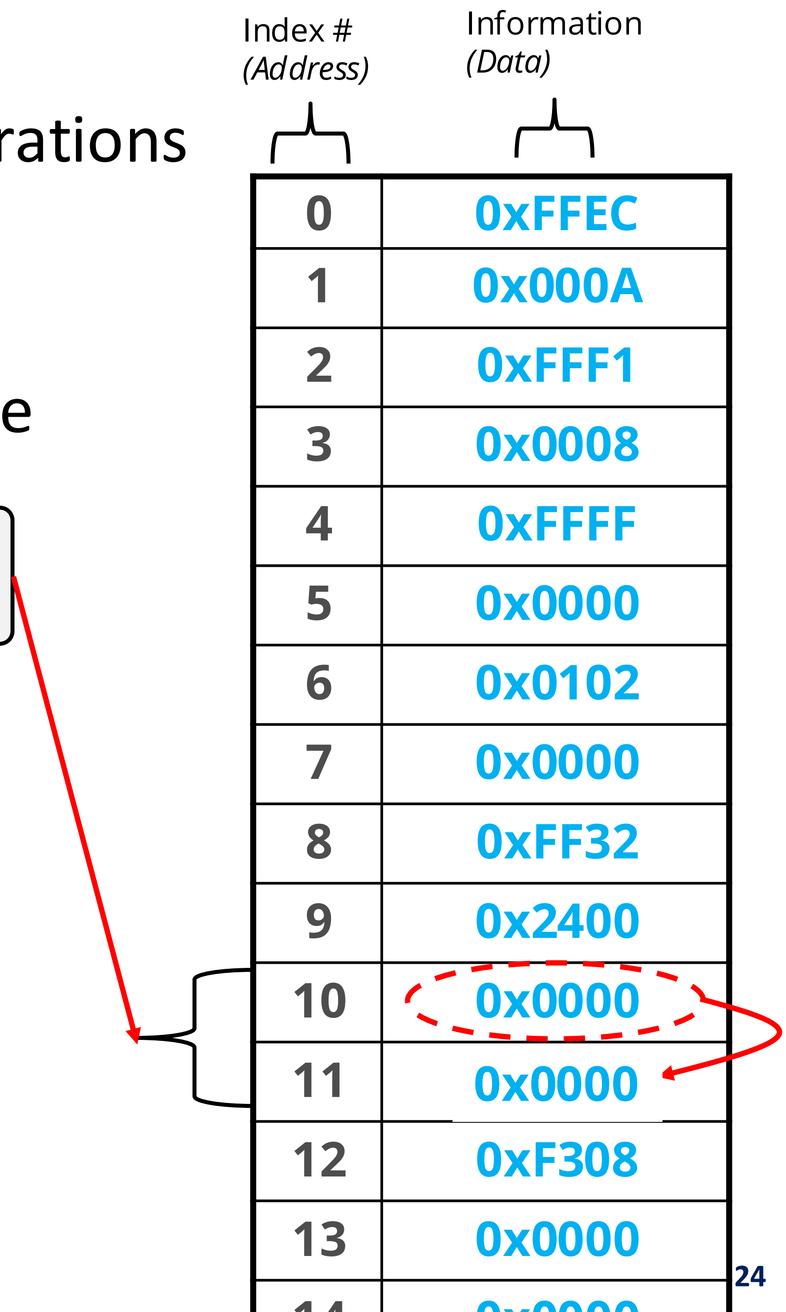
- There are two basic memory operations
  - Selecting a location to **read** from
  - Selecting a location to **write** to
- Consider our example from before

```
int a = 0;
int b = 9;
```

- What if we did

```
b = a;
```

- Done in ***two steps***:
  - Read the value stored in a
  - Store the value in b



Poll Everywhere

pollev.com/cis2400

- If we wanted to use memory that contains 128 different locations, how many bits do we need at minimum to represent an address?

A. 5

B. 7

C. 8

D. 6

E. I'm not sure

Poll Everywhere

pollev.com/cis2400

- If we wanted to use memory that contains 128 different locations, how many bits do we need at minimum to represent an address?

A. 5

B. 7

C. 8

D. 6

E. I'm not sure

7 bits is = 2^7 different values

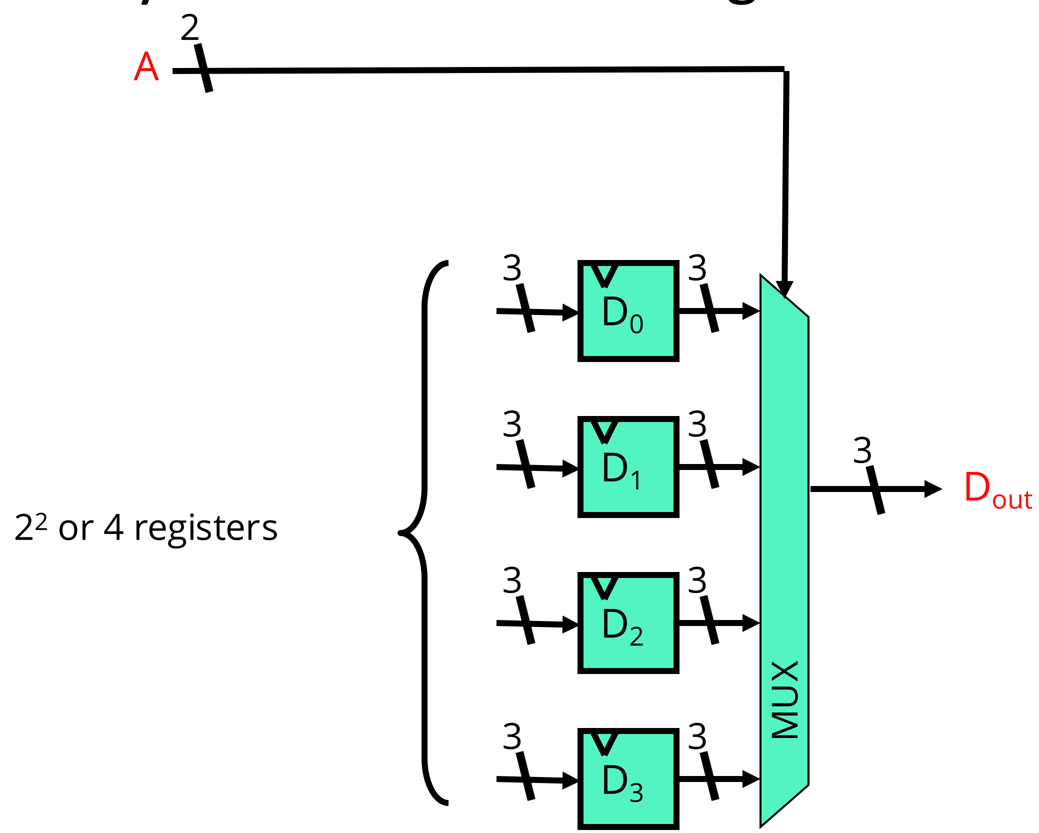
2^7 is 128

Lecture Outline

- Review
  - D Latch & Clock
  - D Flip Flops
- Registers
- Memory at a high level
- Memory using flip flops
- Memory Hierarchy

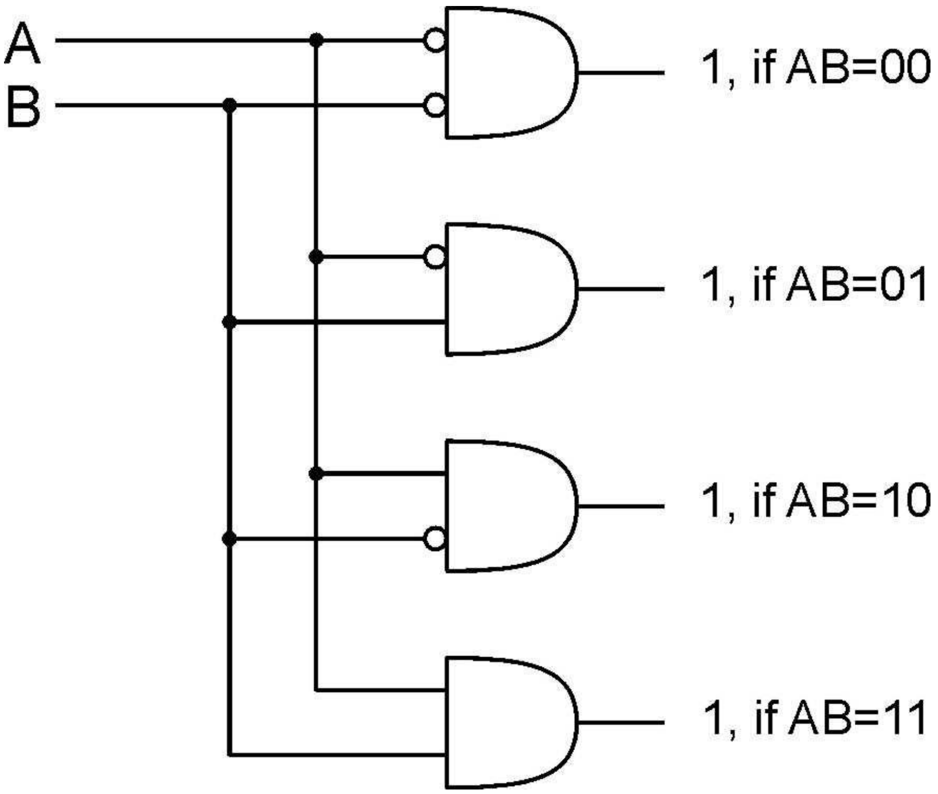
Let's build a simple 2^2 by 3-bit memory

- 2^2 is the amount of locations
- 3-bit is the size of the memory at the location.
- We can implement memory as a collection of registers
- Read operation:



Aside: Decoder

- n inputs, 2^n outputs
  - n = 2 for this example inputs are A and B
- A single output will be 1, the rest will be 0
  - Putting in a binary number will have the corresponding output wire "turn on"
- Sort of a "reverse MUX"
  - Instead of 4-to-1 we are sort of doing 1-to-4

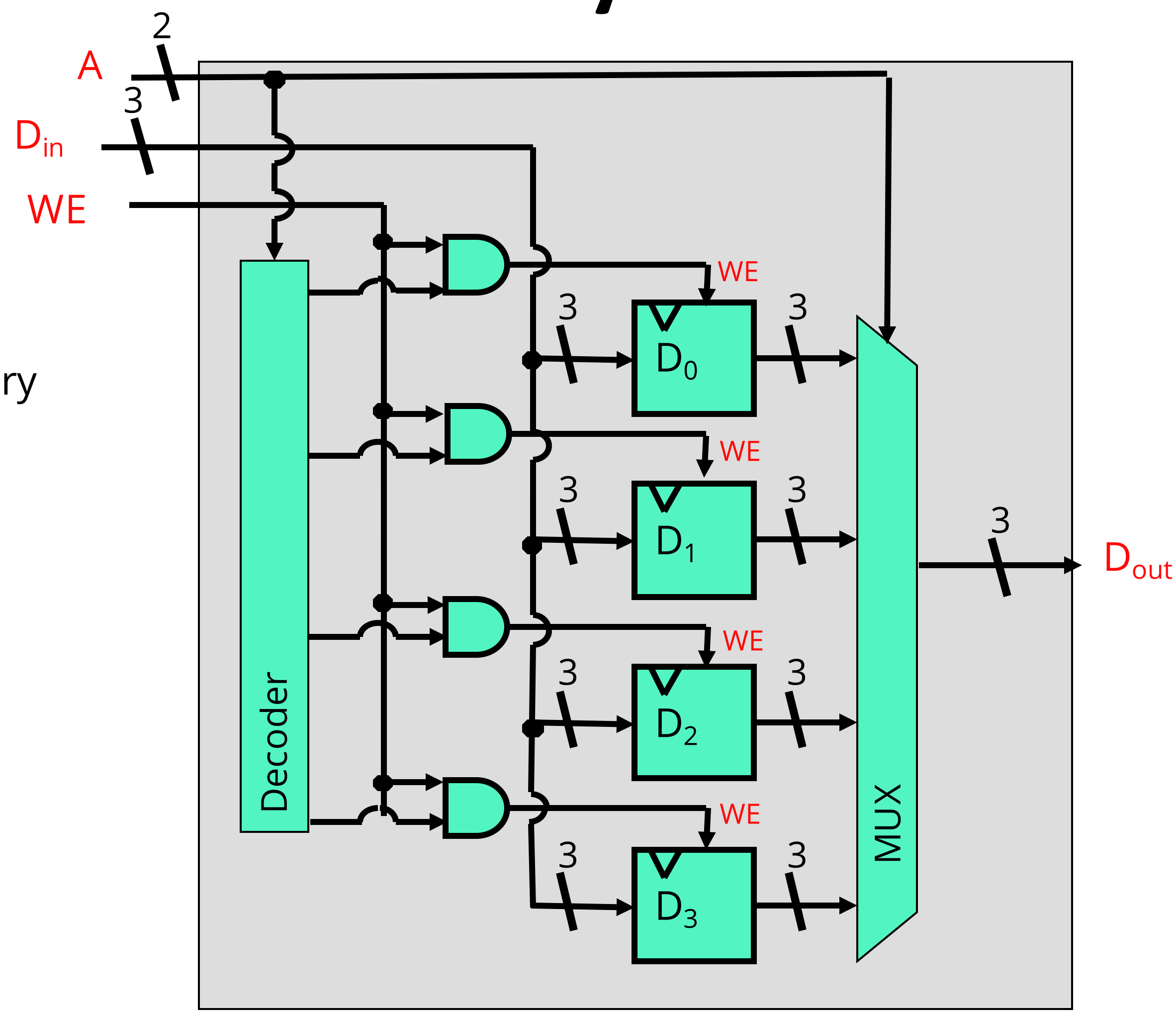


2^2 by 3-bit writeable memory

Write operation

Limitation:
You can only read or write at any given time

This is called "single port" memory



2^2 by 3-bit writeable memory

Write operation

Limitation:
You can only read or write at any given time

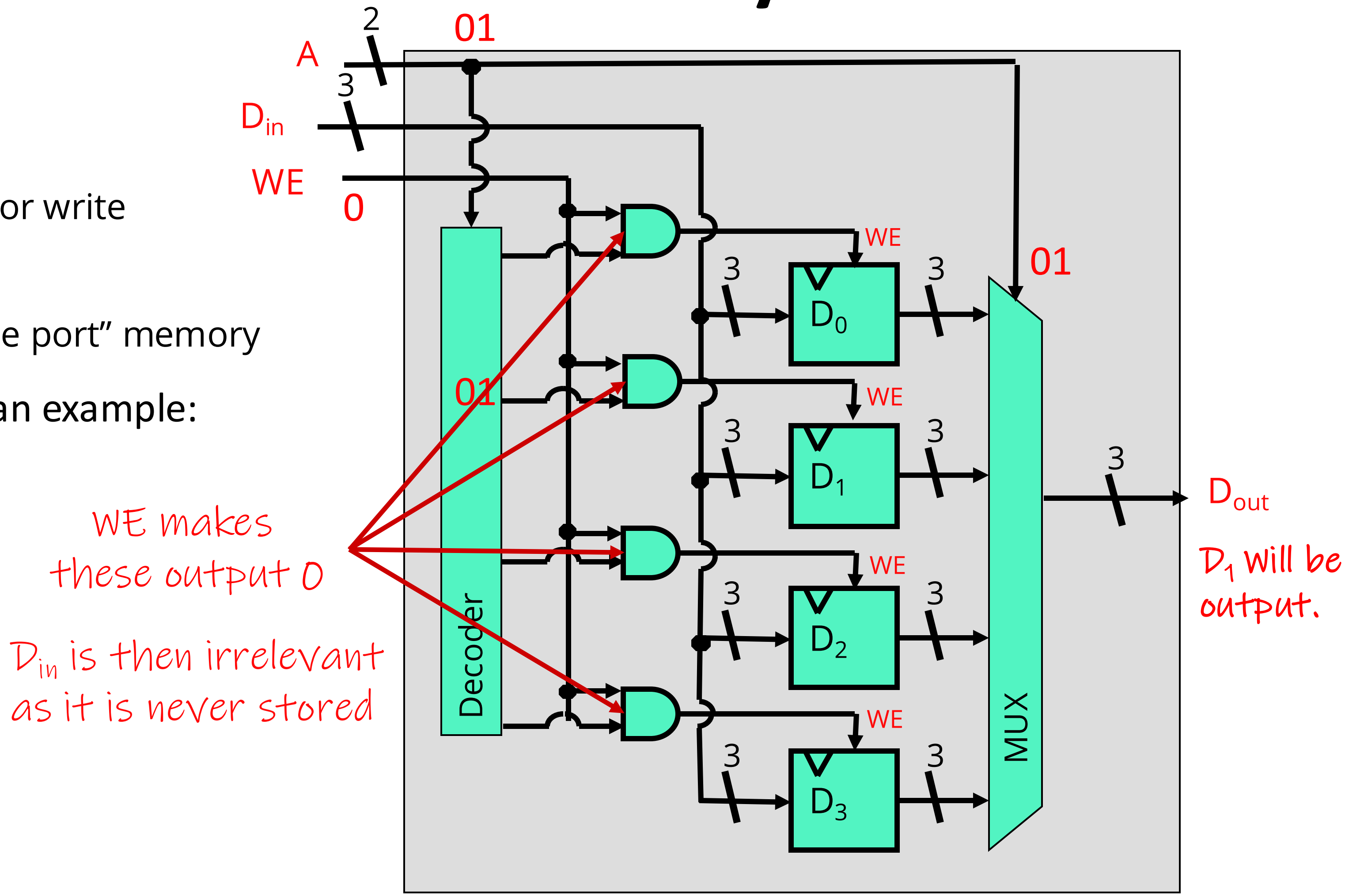
This is called "single port" memory

Let's go through an example:

A = 01

WE = 0

D = ?



Poll Everywhere

pollev.com/cis2400

Which is true?

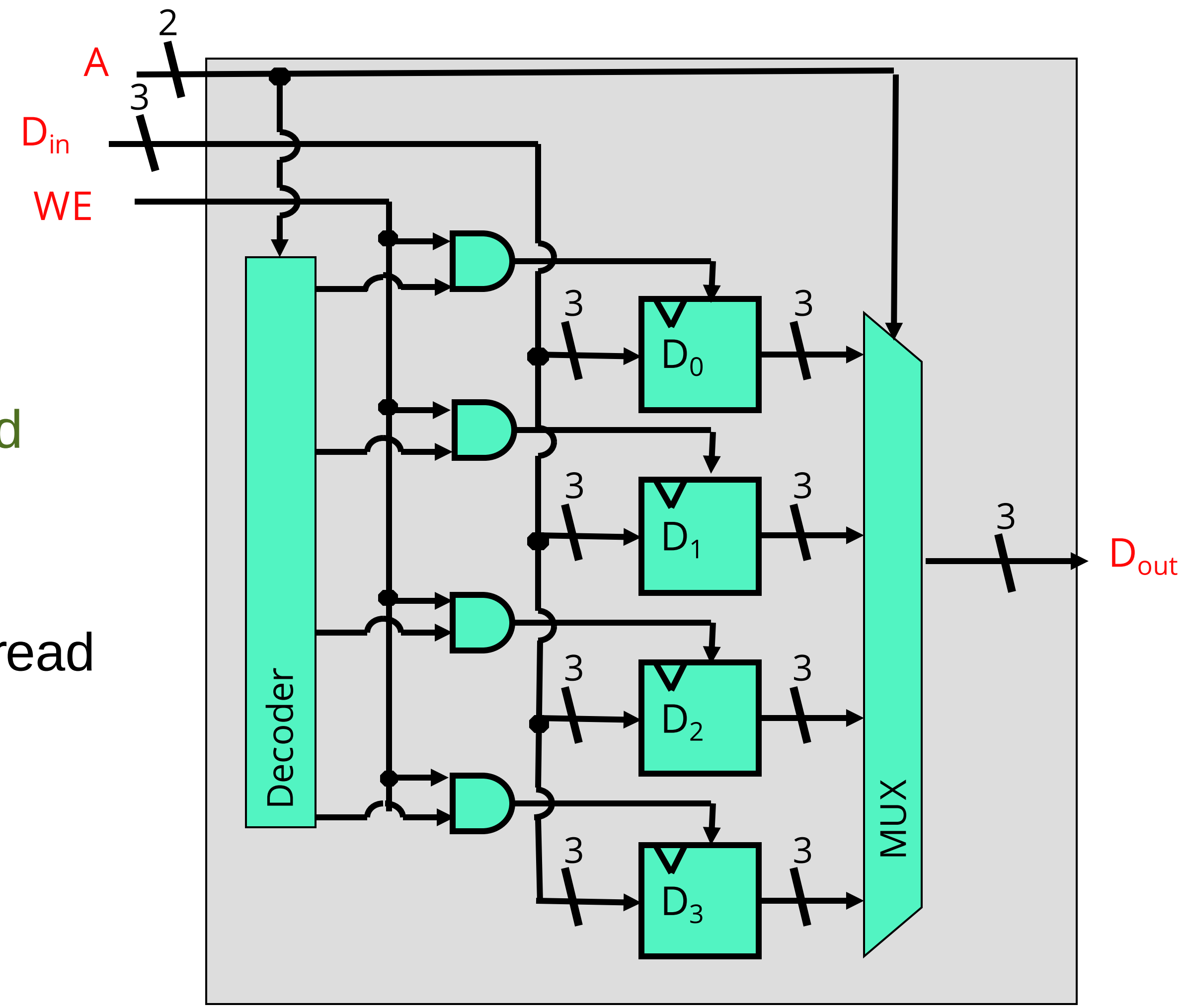
A = 10
WE = 1
D_{in} = 001

A. D_3 is read only

B. D_2 is set to 001 and read

C. D_1 is read

D. D_2 is set to 001 but not read



Poll Everywhere

pollev.com/cis2400

Let's see what happens

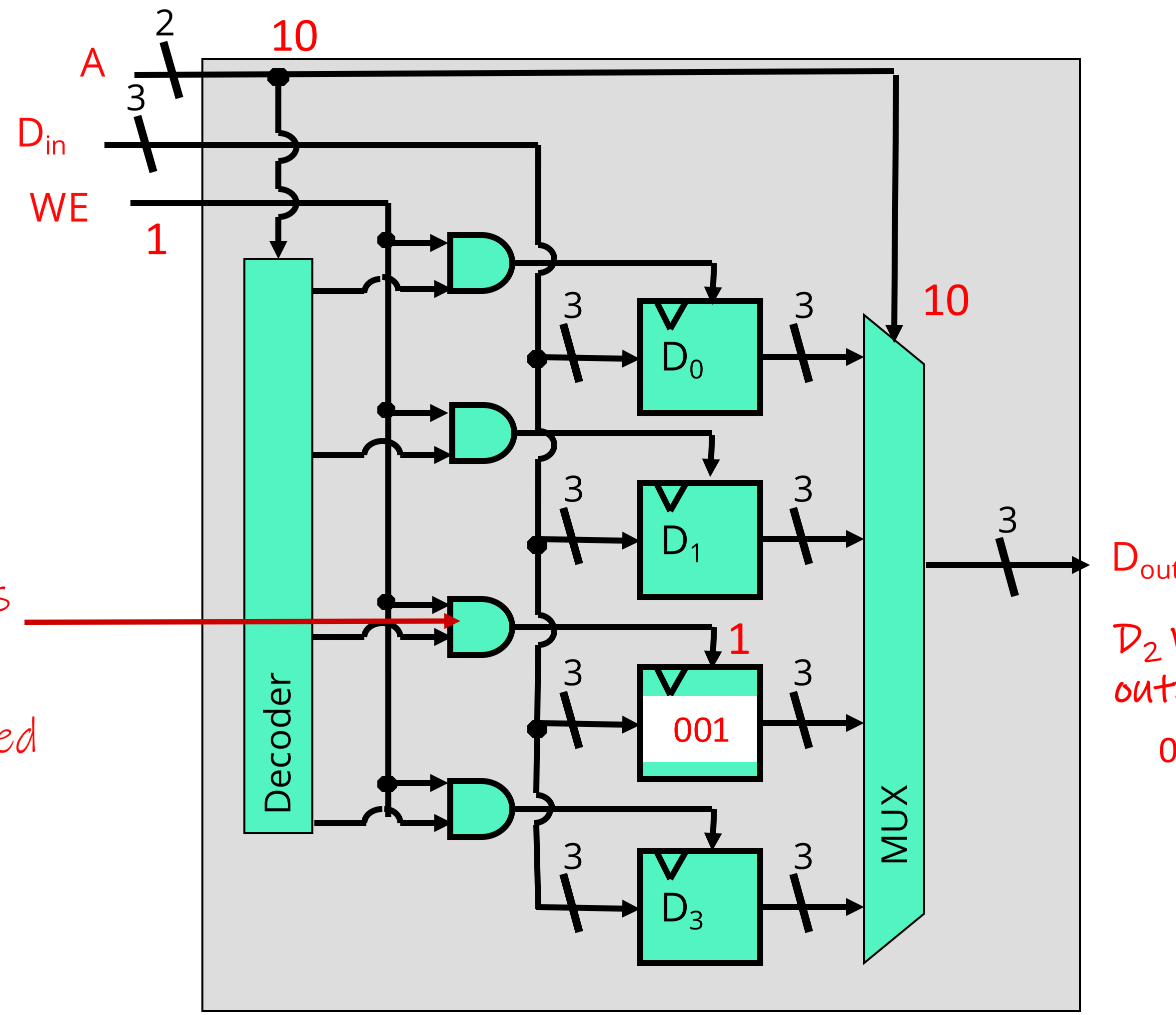
A = 10

WE = 1

D = 001

Decoder makes this output 1

D_{in} is then stored in D_2.



D_2 will be output.

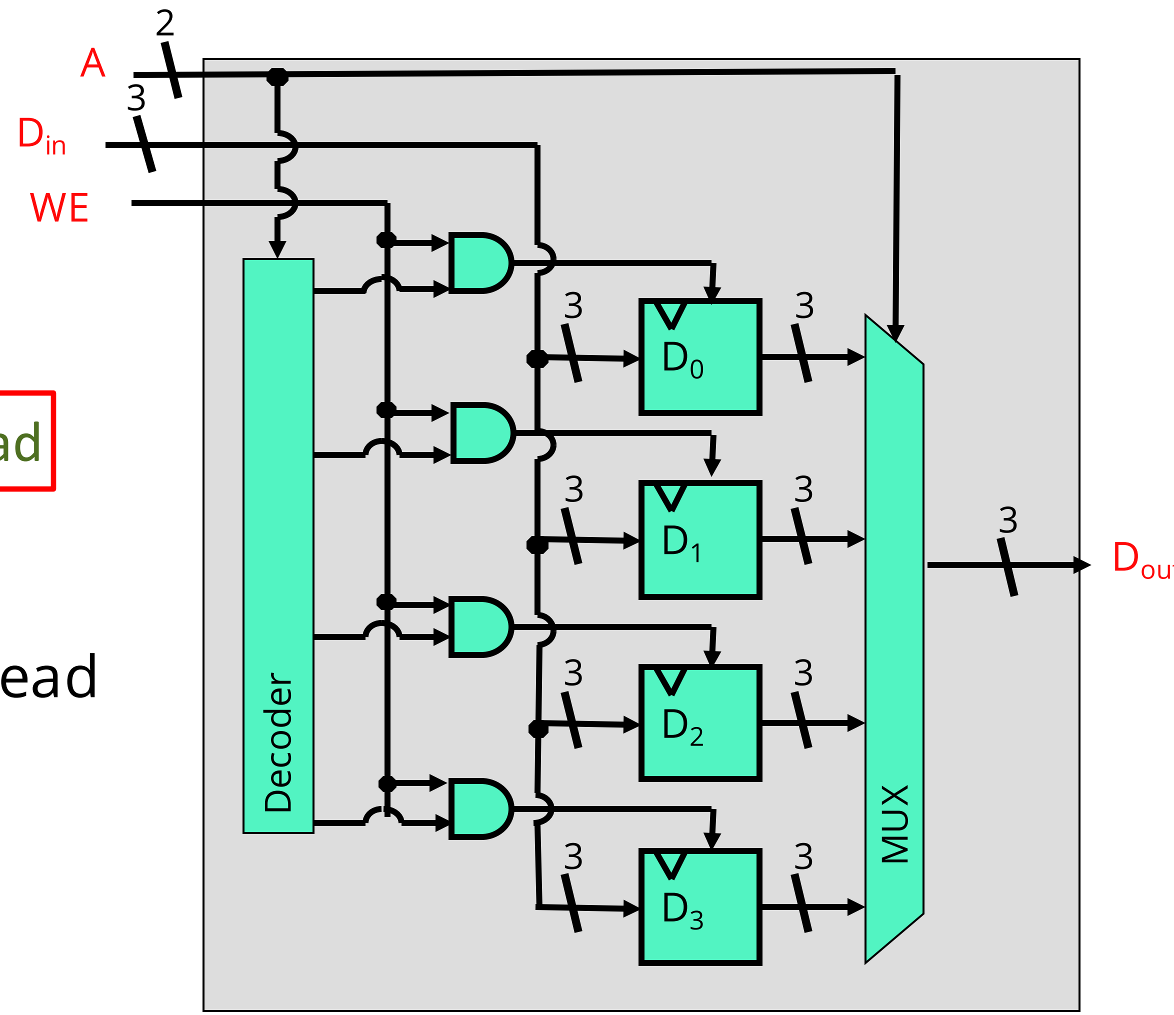
001

Poll Everywhere

pollev.com/cis2400

Which is true?

A = 10
WE = 1
D = 001



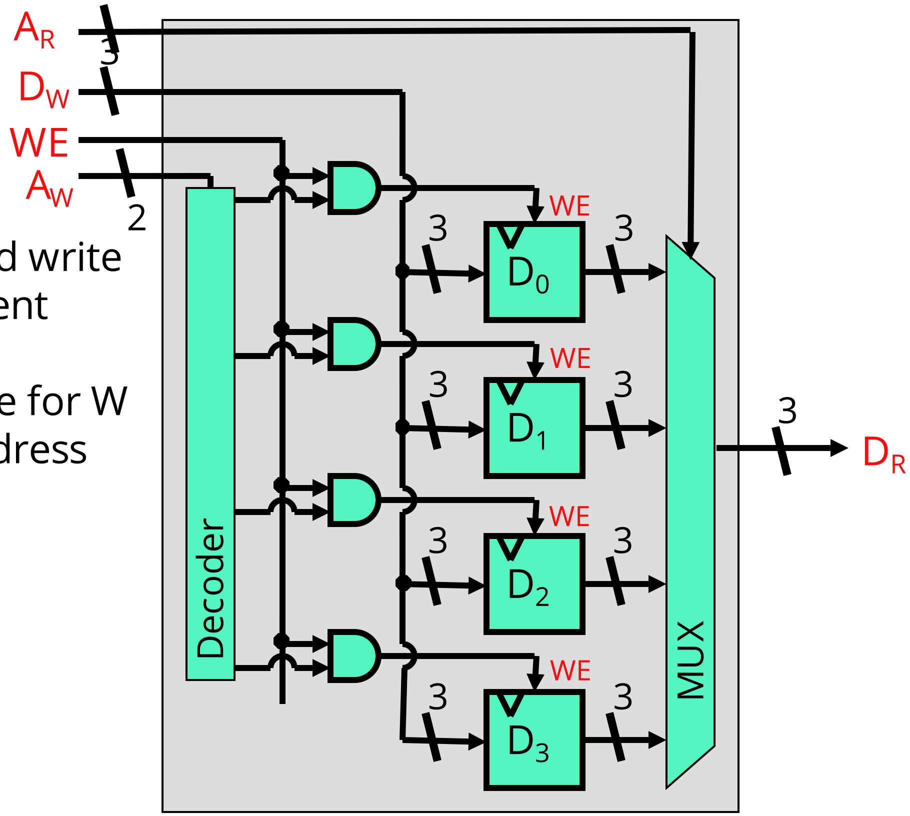
- **A.** D_3 is read only
- **B.** D_2 is set to 001 and read
- C. D_1 is read
- D. D_4 is set to 001 and read

2^2 by 3-bit memory – independent R/W

- Can have independent read/write operations if we take in separate addresses for each.

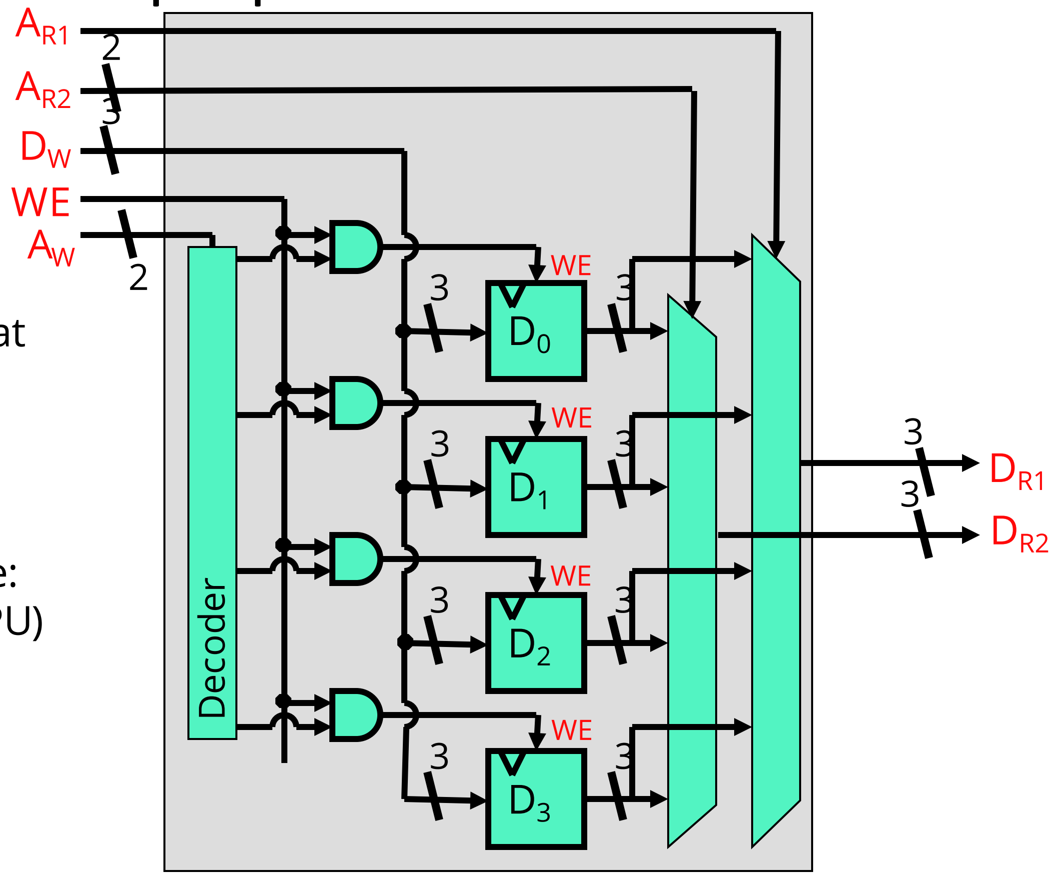
You can read from one address and write to another with this arrangement

1 address line for R & 1 address line for W
Previously we used the same address



2^2 by 3-bit memory – Multiple Reads

- Can read from multiple places at once!



Read from 2 locations at once, write to a third!
(notice 3 address lines)

(We will use this later
In something called the:
"register file" for the CPU)

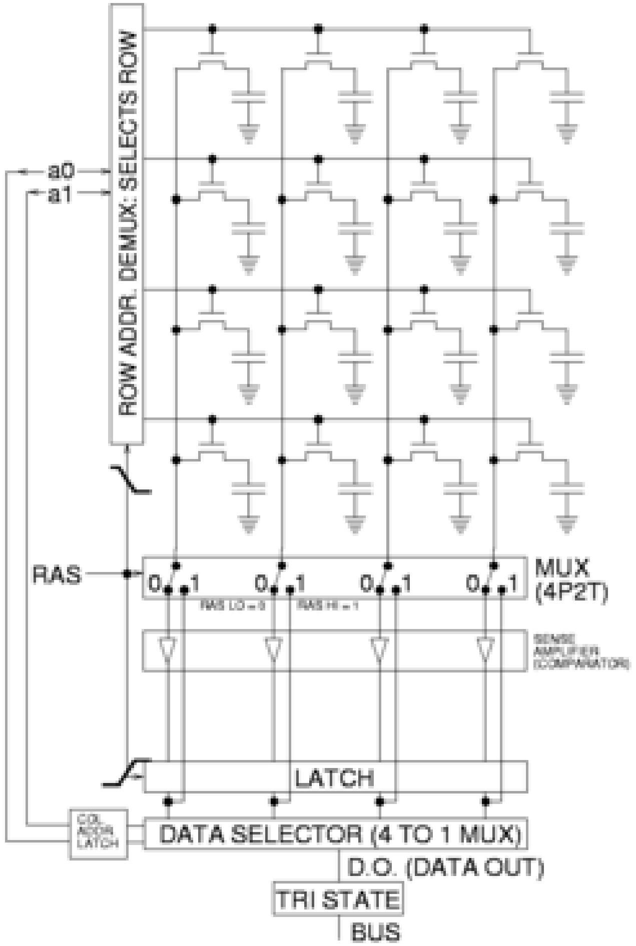
More Memory Details

- The Memory We've Created Would need many transistors, but is a good starting place to understand how it works!
  - Real memory: fewer transistors, denser, relies on analog properties
- The logical structure of all memory is similar
  - Address decoder
  - "Word select line", word write enable
  - "Bit line"
- Two basic kinds of RAM (Random Access Memory)
- Static RAM (SRAM) - 6 transistors per bit
  - We've created a type of SRAM in this presentation, we can do better (6 transistors!)
  - Fast, maintains data as long as power applied
- Dynamic RAM (DRAM) - 1 transistor per bit
  - Denser but slower, relies on "capacitance" to store data, needs constant "refreshing" of data to hold charge on capacitor

Also, non-volatile memories: ROM, PROM, flash, …

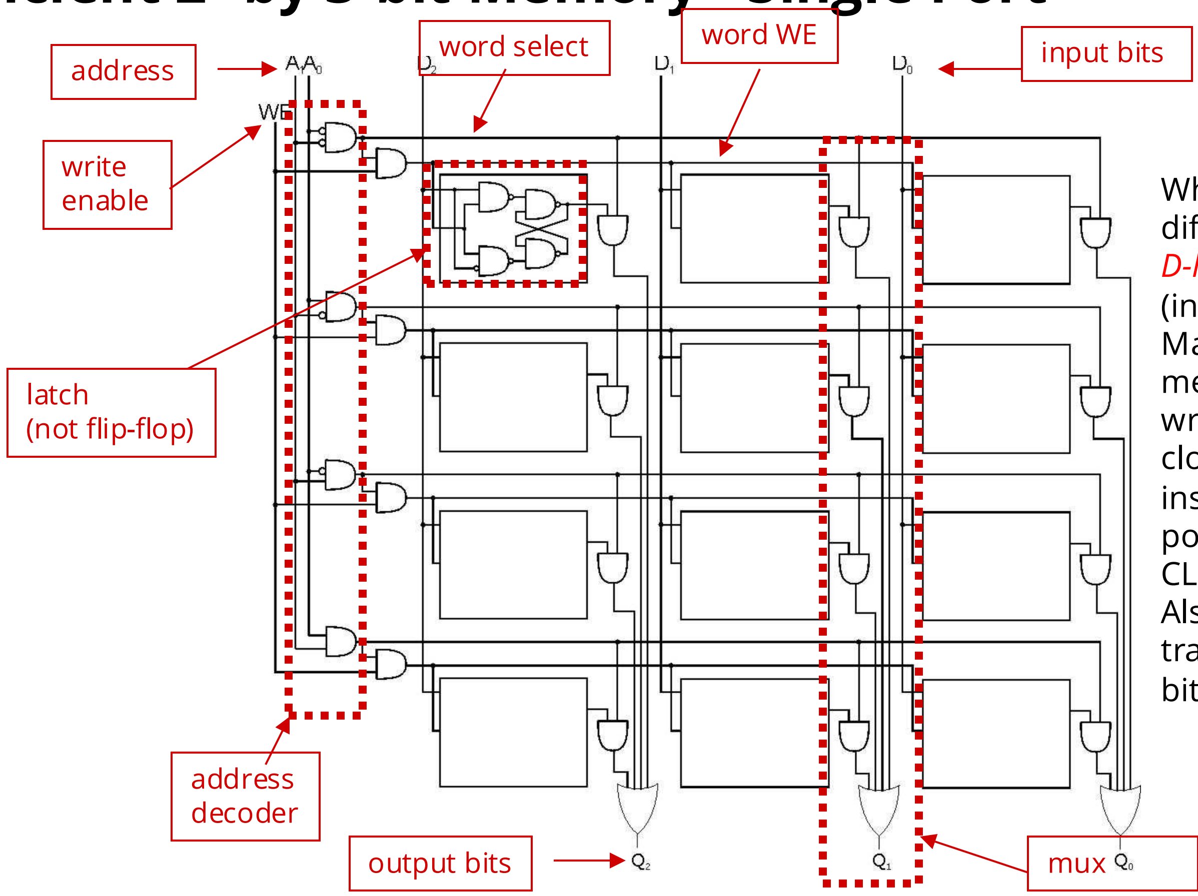
Dynamic RAM

- Information stored as charge on capacitors
- Capacitors *leak* so values have to be 'refreshed' continually
- As memory chips get larger, access times tend to increase. The processor spends more time waiting for data.
  - This is a major issue limiting computer systems performance



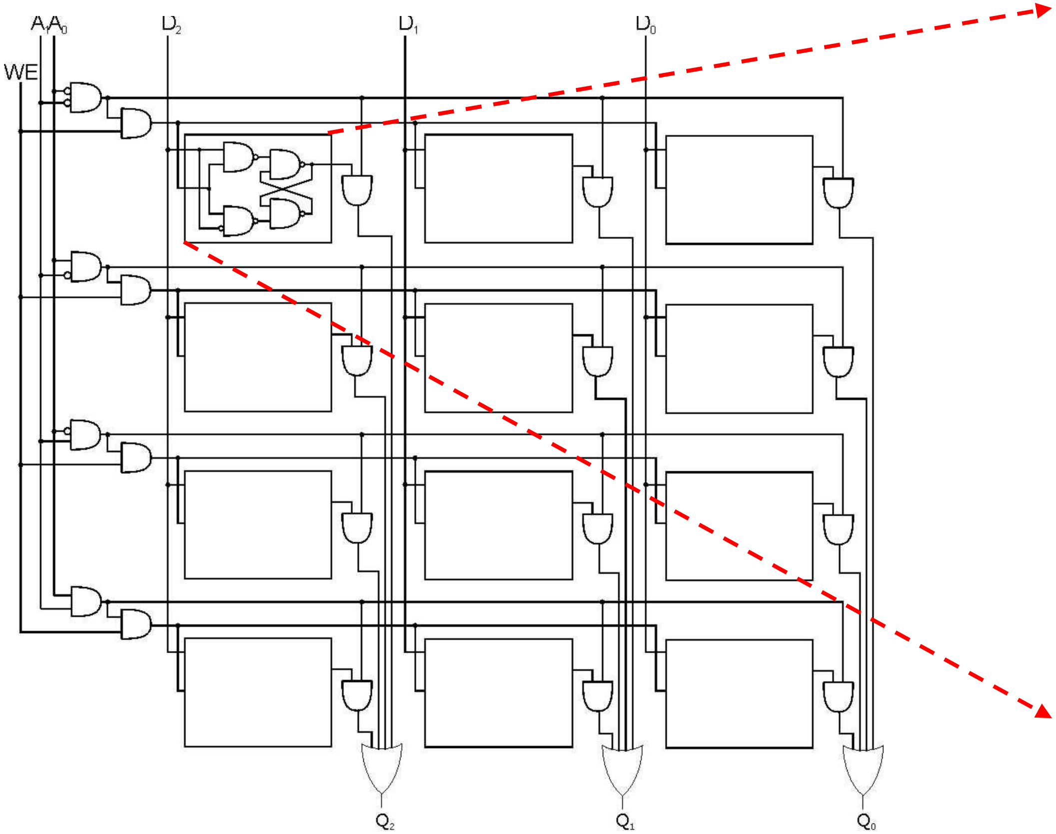
Dynamic RAM

An Efficient 2^2 by 3-bit Memory - Single Port

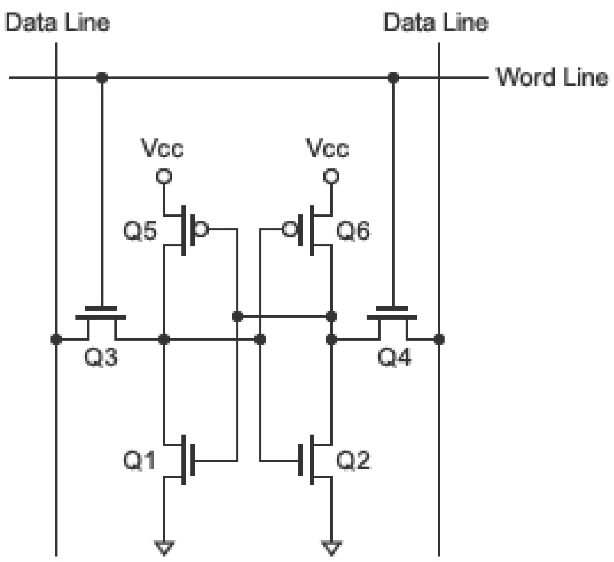


What is different?
D-latch
(instead of D-FF)
Makes this memory writeable when clock is HIGH, instead of on positive edge of CLK
Also uses less transistors per bit!

Efficient 2^2 by 3-bit Memory – Single Port – SRAM Cell



A 6 Transistor D-Latch!



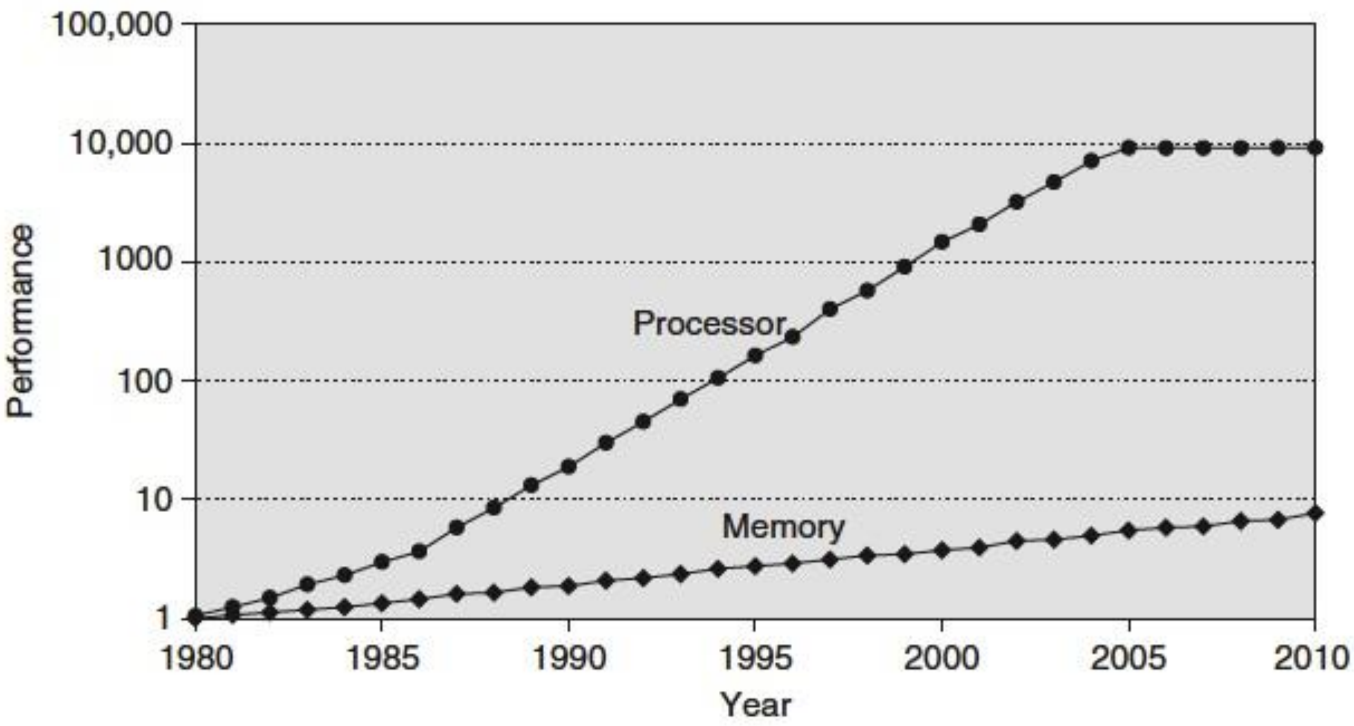
Lecture Outline

- Review
  - D Latch & Clock
  - D Flip Flops
- Registers
- Memory at a high level
- Memory using flip flops
- Memory Hierarchy

Data Access Time

- Data is stored on a physical piece of hardware
- The distance data must travel on hardware affects how long it takes for that data to be processed
- Example: data stored closer to the CPU is quicker to access
  - We will see this as we discuss memory vs registers in RISC-V programming
  - As we go further from the CPU, storage space goes up, but access times increase

Processor-Memory Gap



- Processor speed kept growing ~55% per year
- Time to access memory didn't grow as fast ~7% per year
- Memory access would create a bottleneck on performance

Cache

- Pronounced "cash"

- English: A hidden storage space for equipment, weapons, valuables, supplies, etc.

- Computer: Memory with shorter access time used for the storage of data for increased performance. Data is usually either something frequently and/or recently used.

Principle of Locality

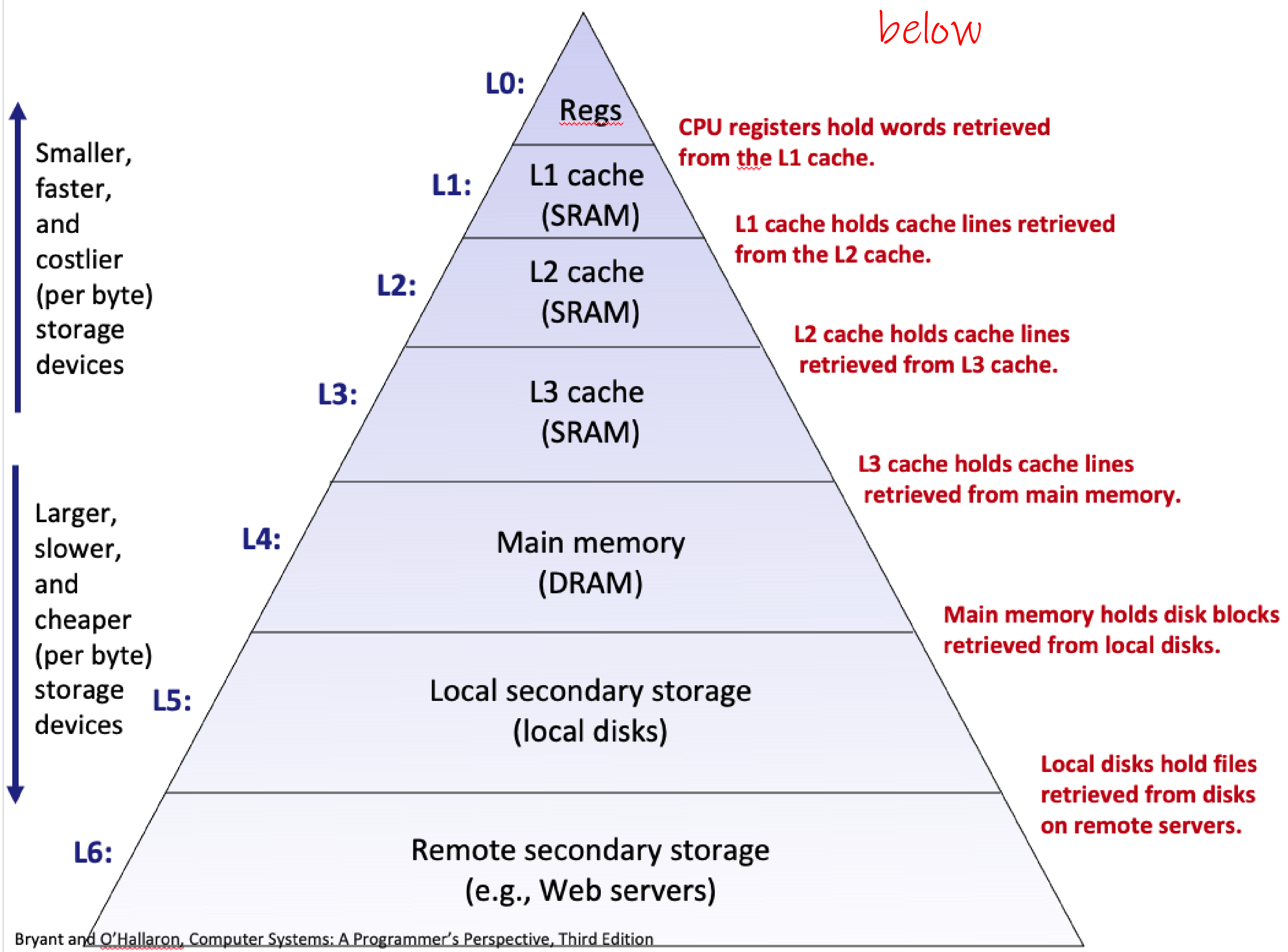
- The tendency for the CPU to access the same set of memory locations over a short period of time

- Two main types:
  - **Temporal Locality**: If we access a portion of memory, we will likely reference it again soon
  - **Spatial Locality**: If we access a portion of memory, we will likely reference memory close to it in the near future.

- Caches take advantage of these tendencies with the cache policies to decide what data is stored in the cache.

Memory Hierarchy

Each layer can be thought of as a "cache" of the layer below



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

And that's it; enjoy your weekend and see you on Tuesday for the Midterm Review!