CSE 390B, Winter 2022

Building Academic Success Through Bottom-Up Computing

Sequential Logic, Bloom's Taxonomy

We hope you had a great MLK weekend!

Bloom's Taxonomy, Cornell Note-Taking Method, Circuit Design, Physical Timekeeping, Sequential Logic

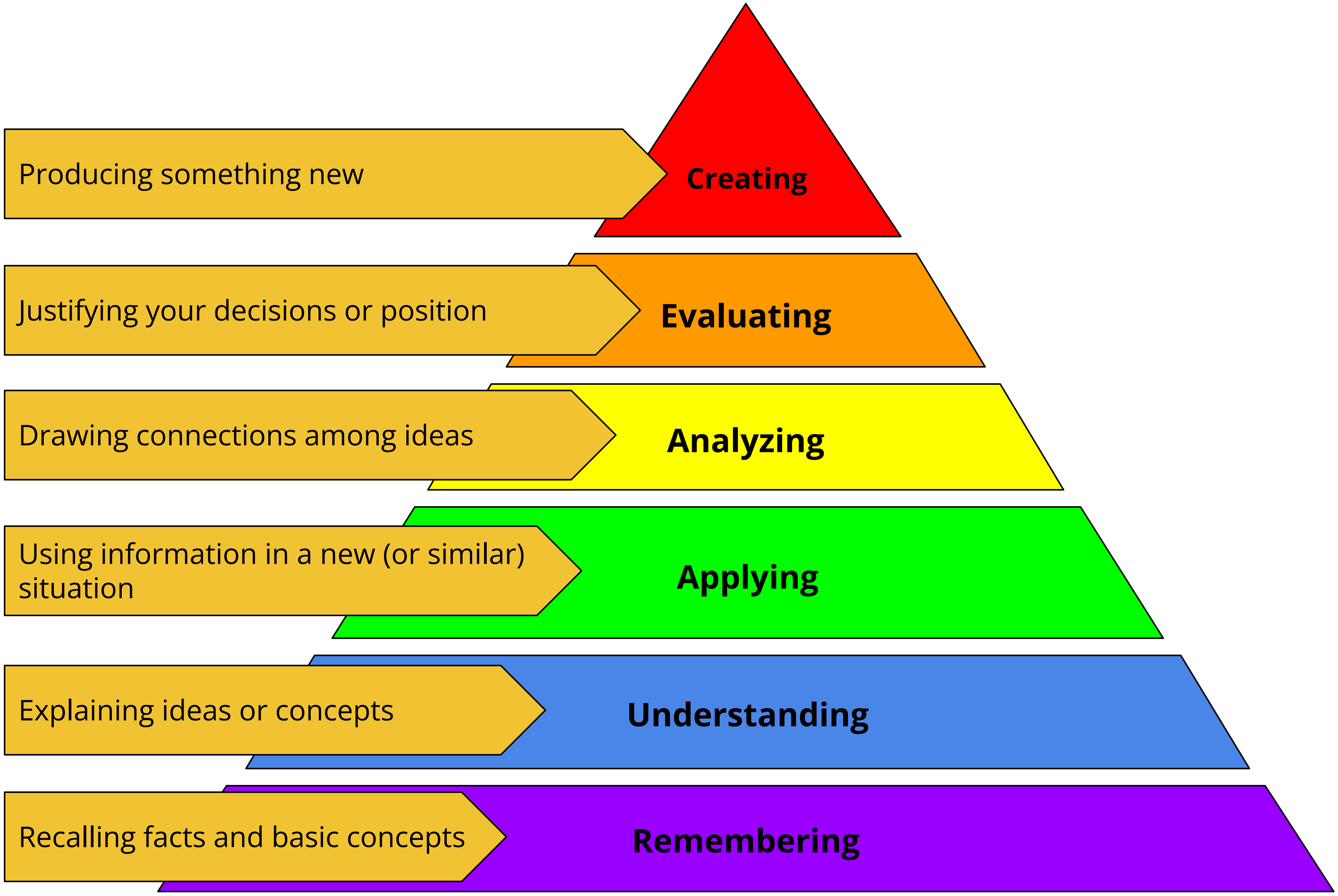
If you are joining virtually, please have your camera turned on!

W UNIVERSITY *of* WASHINGTON

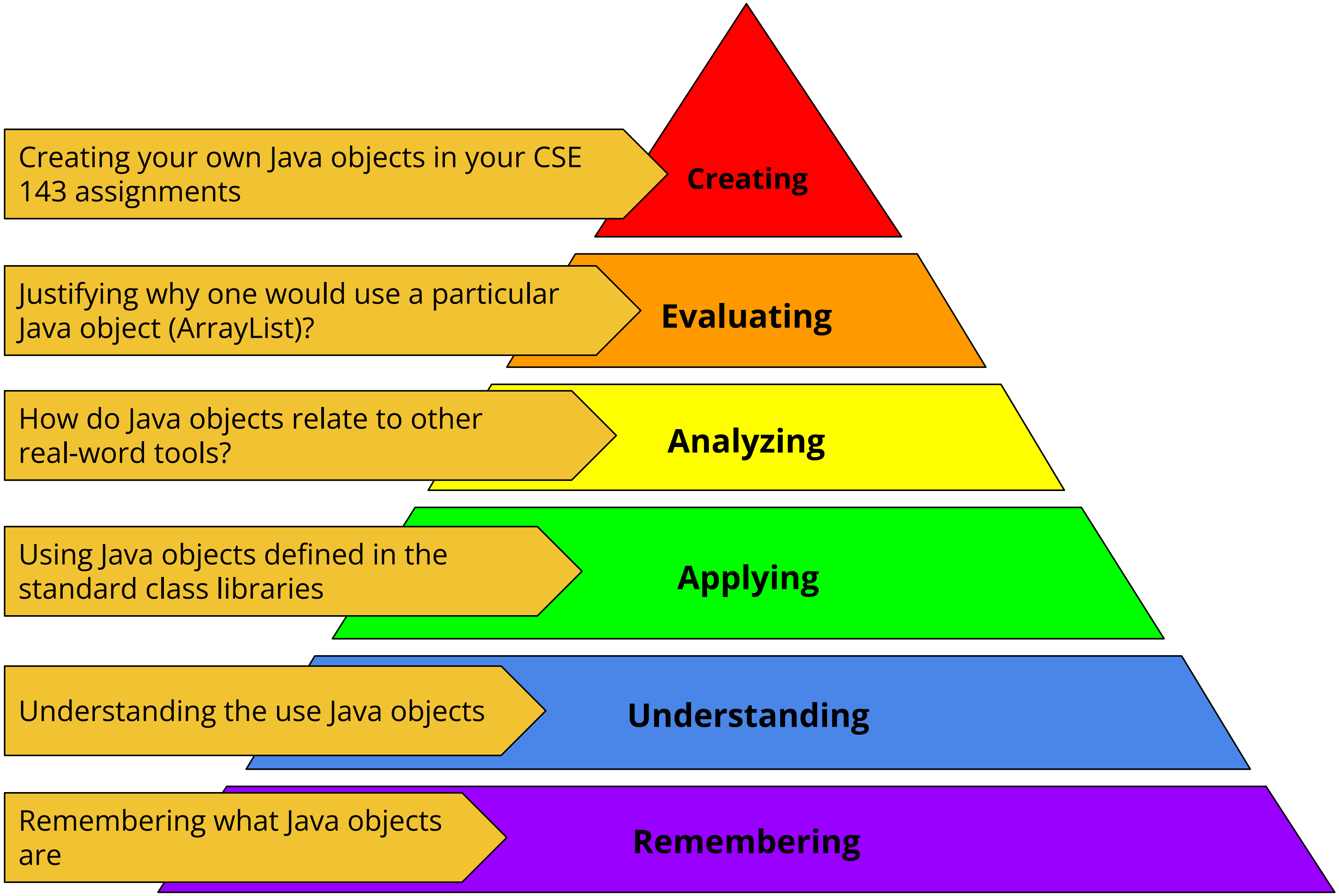
Lecture Outline

- **Bloom's Taxonomy**
  - **Apply levels of learning to coursework**
- Cornell Note-Taking Method
  - System for taking, organizing, and reviewing notes
- Representing Time in Hardware
  - Sequential Logic vs. Combinational Logic
  - Adding a clock
- Sequential Logic
  - Data Flip-Flip (DFF) implementation and examples

Bloom's Taxonomy


Producing something new
Creating
Justifying your decisions or position
Evaluating
Drawing connections among ideas
Analyzing
Using information in a new (or similar) situation
Applying
Explaining ideas or concepts
Understanding
Recalling facts and basic concepts
Remembering

Bloom's Taxonomy in Action


Creating your own Java objects in your CSE 143 assignments
Creating
Justifying why one would use a particular Java object (ArrayList)?
Evaluating
How do Java objects relate to other real-word tools?
Analyzing
Using Java objects defined in the standard class libraries
Applying
Understanding the use Java objects
Understanding
Remembering what Java objects are
Remembering

Lecture Outline

- Bloom's Taxonomy
  - Apply levels of learning to coursework
- **Cornell Note-Taking Method**
  - **System for taking, organizing, and reviewing notes**
- Representing Time in Hardware
  - Sequential Logic vs. Combinational Logic
  - Adding a clock
- Sequential Logic
  - Data Flip-Flip (DFF) implementation and examples

Cornell Note-Taking Method

Questions

Compose a question that corresponds to the notes you took

In what ways is object-oriented programming more extensible than functional programming?

Notes

I. Main Topic
   - Sub point
   - <u>definition</u>
   - example **

II. Object-Oriented Programming
   - Encapsulates the data and the operations for a given data type
   - Provides abstractions - you don't need to know how a car is implemented in order to use it
   - Extensibility - easier to <u>add new data types</u>

III. Functional Programming
   - Extensibility - easier to <u>add new operations</u>

Summary

Object-oriented programming and functional programming are two types of programming paradigms…

Cornell Note-Taking Method

Questions

Compose a question that corresponds to the notes you took

In what ways is object-oriented programming more extensible than functional programming?

Summary

Object-oriented programming and functional programming are two types of programming paradigms…

Applying the Cornell Note-Taking Method

- You are going to try it… TODAY!
- Thursday you will come & contrast Cornell notes with your classmates
- You are also going to try it with one of your other classes as part of Project 3 (more on Thursday)

Lecture Outline

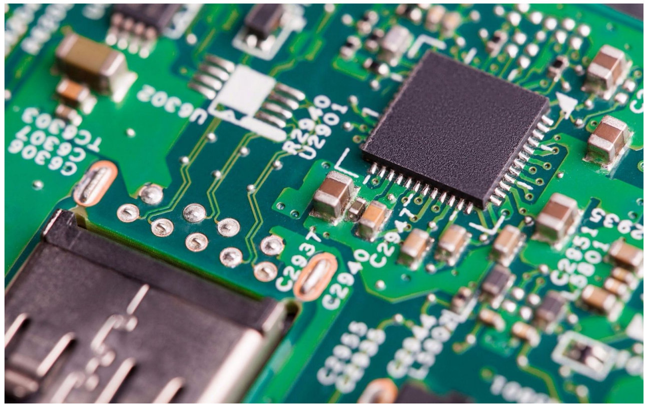
- Bloom's Taxonomy
  - Apply levels of learning to coursework
- Cornell Note-Taking Method
  - System for taking, organizing, and reviewing notes
- **Representing Time in Hardware**
  - **Sequential Logic vs. Combinational Logic**
  - **Adding a clock**
- Sequential Logic
  - Data Flip-Flip (DFF) implementation and examples

Combinational vs. Sequential Logic

- So far, we have ignored “time” in our circuits
- Our chips use **combinational logic**
  - When given inputs, the chip computes its output instantaneously
  - The output is a function of the current inputs, with no memory of previous events
- Today, we’ll start exploring what happens when we consider time, ultimately building to **sequential logic**

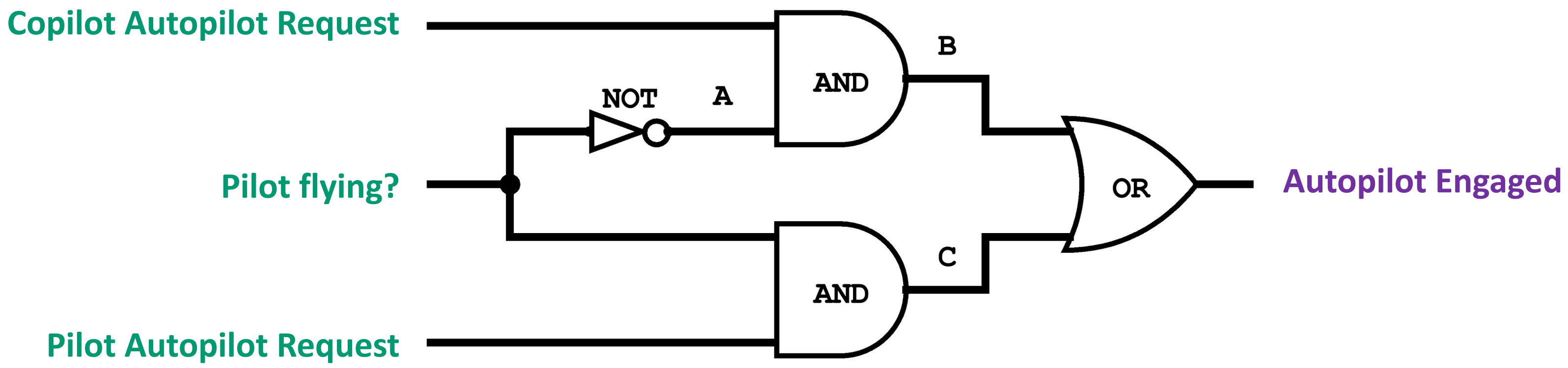
Why Consider Time Now?

- Needed for our abstraction
  - We need to talk about hardware maintaining state for memory
  - We need vocabulary to talk about time
- Needed for our implementation
  - Physical implementations of chips cannot be instantaneous
  - We need to account for physical delays in signal propagation



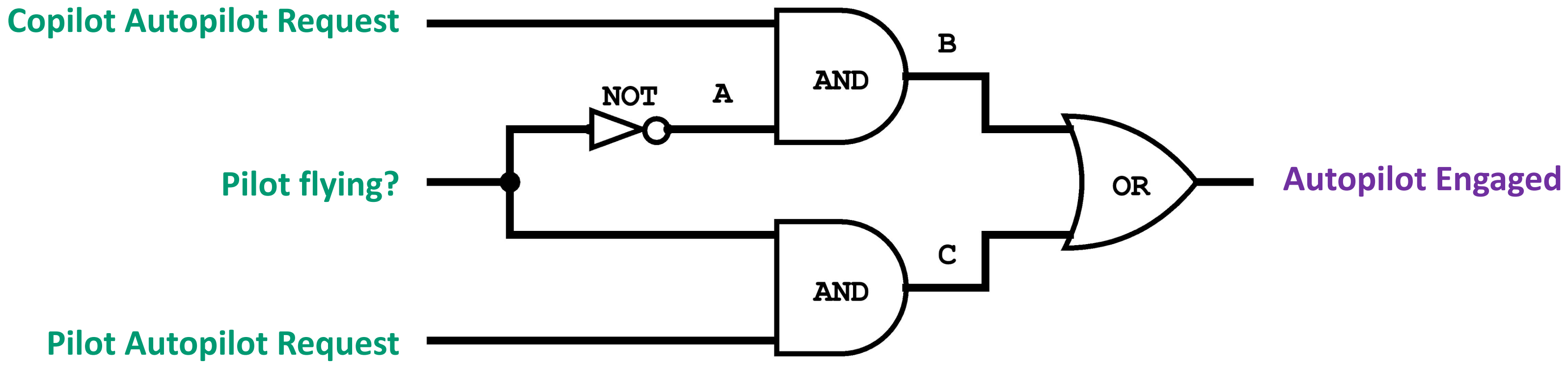
Why Consider Time Now?

- Consider this autopilot control circuit:
  - Either the pilot or copilot is flying at any time
  - The pilot and copilot can separately request autopilot
  - Only the person flying can request autopilot



Why Consider Time Now?

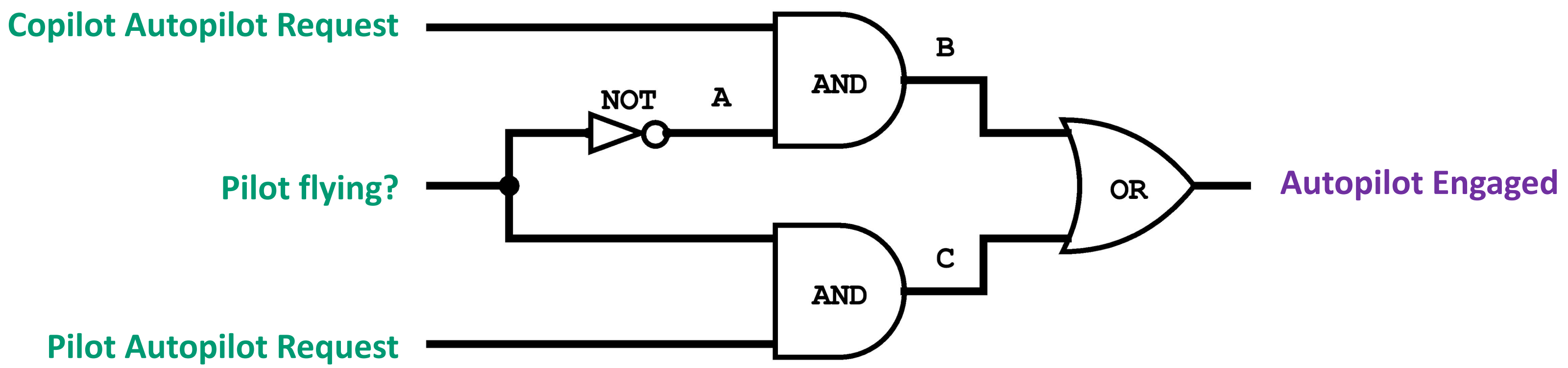
- Consider this autopilot control circuit:
  - Either the pilot or copilot is flying at any time
  - The pilot and copilot can separately request autopilot
  - Only the person flying can request autopilot

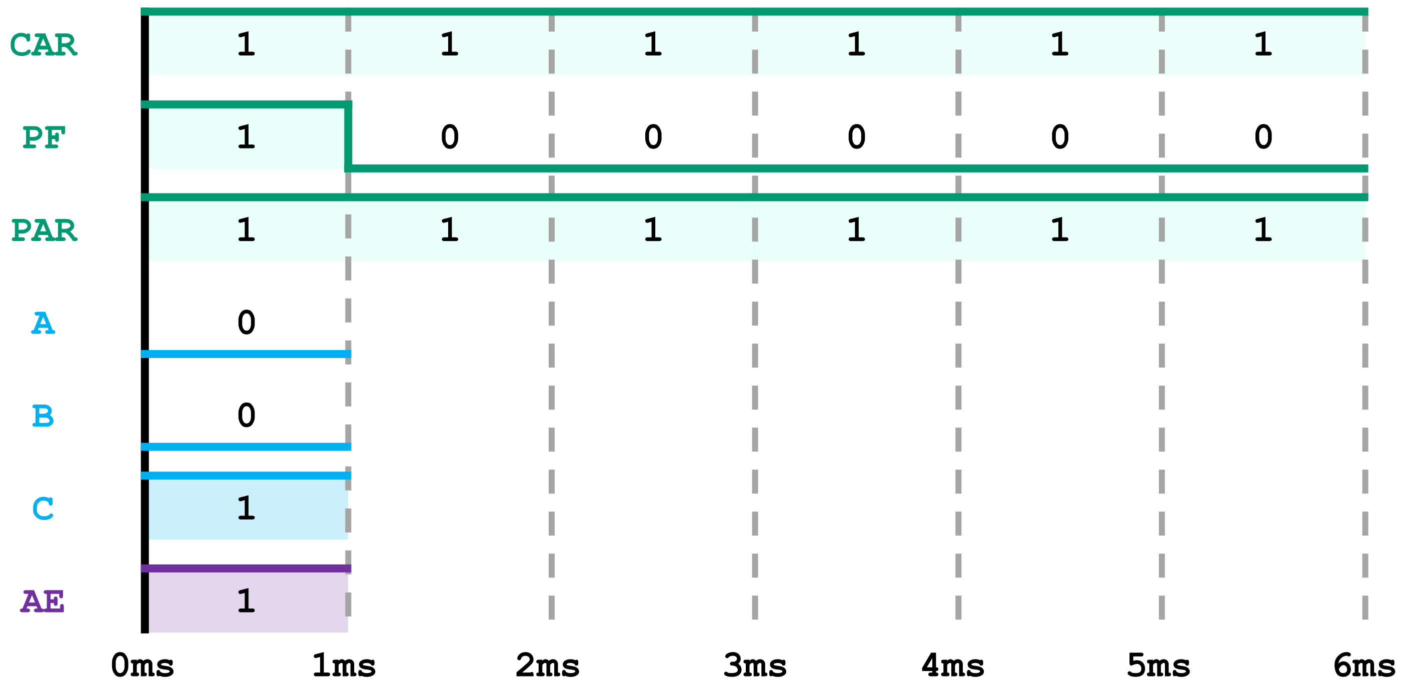


- Let's assume every logic gate takes `1ms` to compute
  - For example, if an input changes at `t=4ms`, the gate will only output the new result at `t=5ms`

Poll Everywhere

Vote at https://pollev.com/cse390b

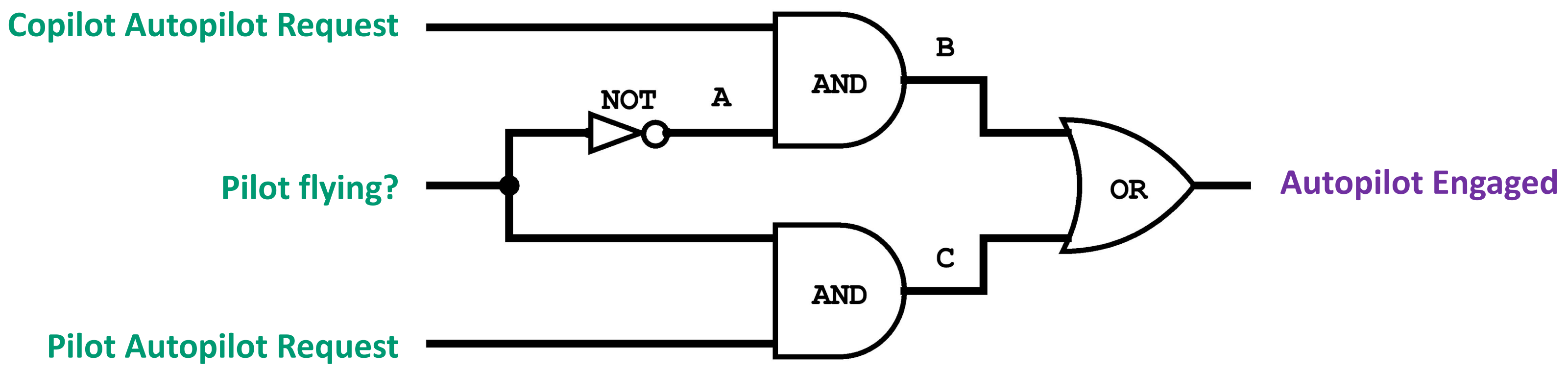


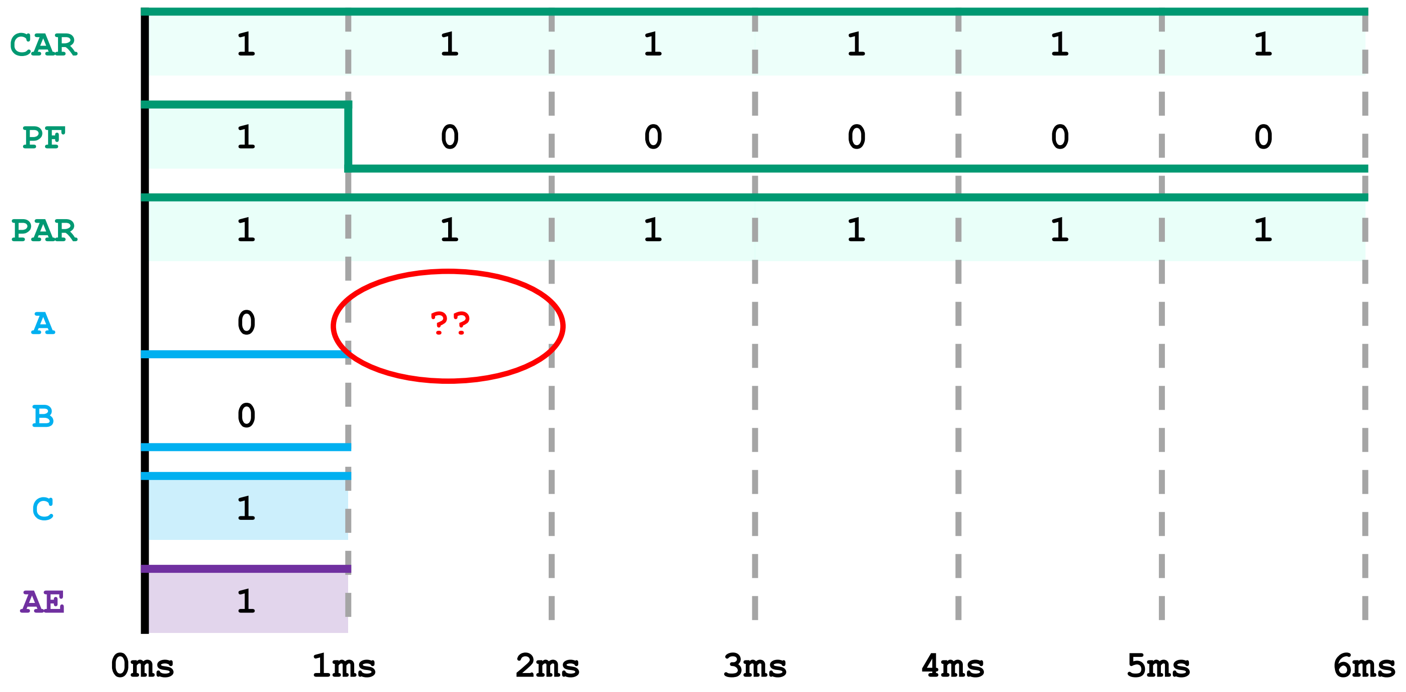


	0–1ms	1–2ms	2–3ms	3–4ms	4–5ms	5–6ms
CAR	1	1	1	1	1	1
PF	1	0	0	0	0	0
PAR	1	1	1	1	1	1
A	0					
B	0					
C	1					
AE	1					

Poll Everywhere

Vote at https://pollev.com/cse390b

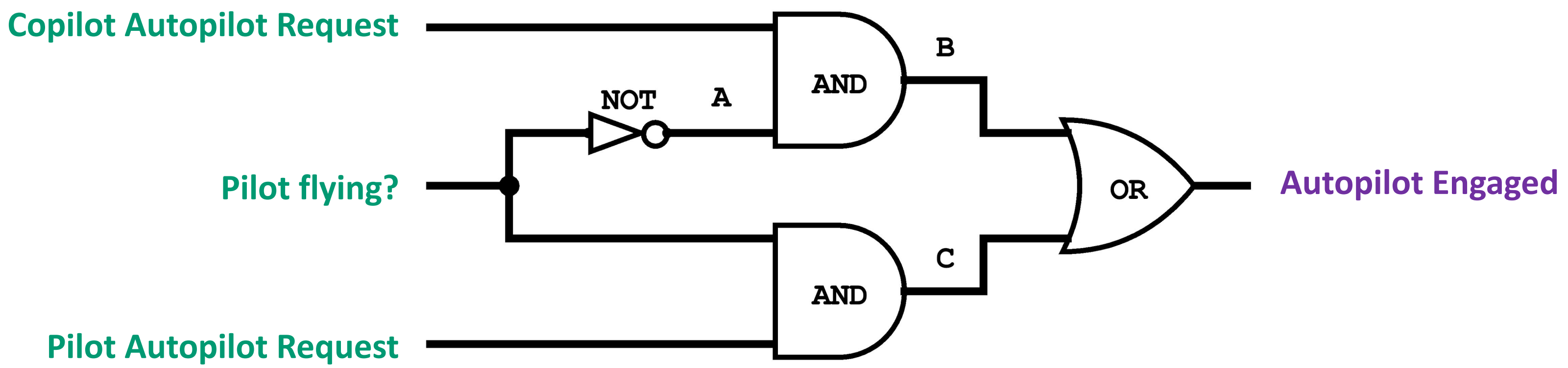


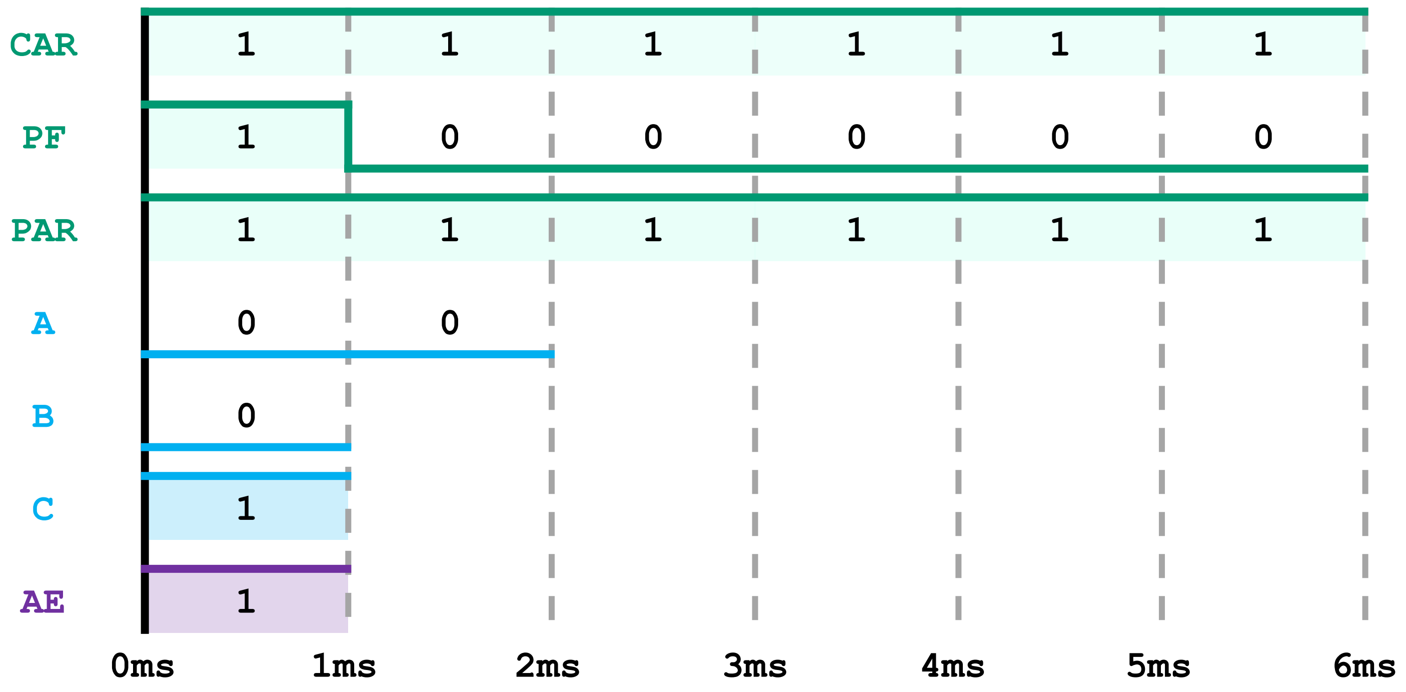


	0–1ms	1–2ms	2–3ms	3–4ms	4–5ms	5–6ms
CAR	1	1	1	1	1	1
PF	1	0	0	0	0	0
PAR	1	1	1	1	1	1
A	0	??				
B	0					
C	1					
AE	1					

Poll Everywhere

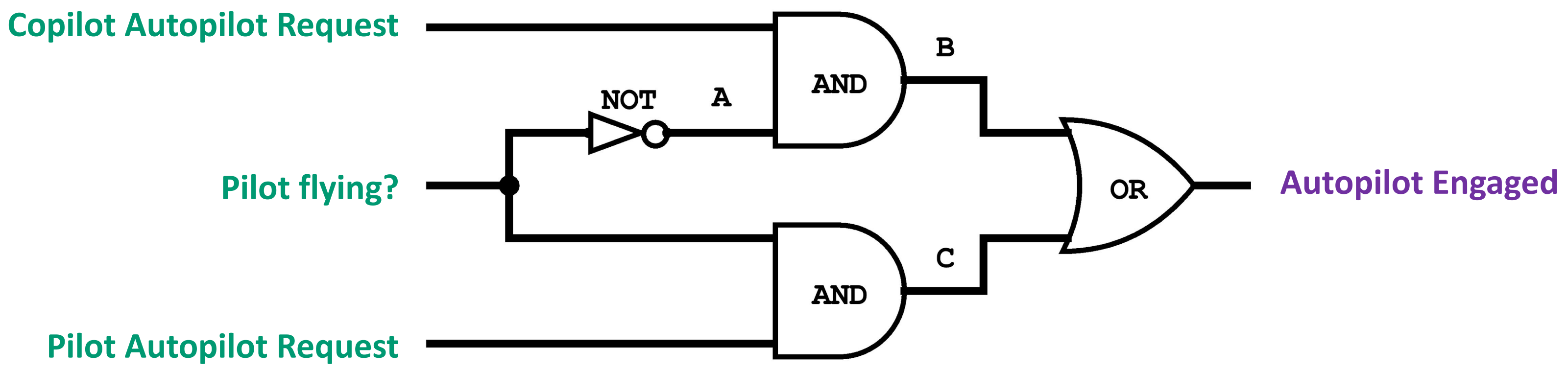
Vote at https://pollev.com/cse390b

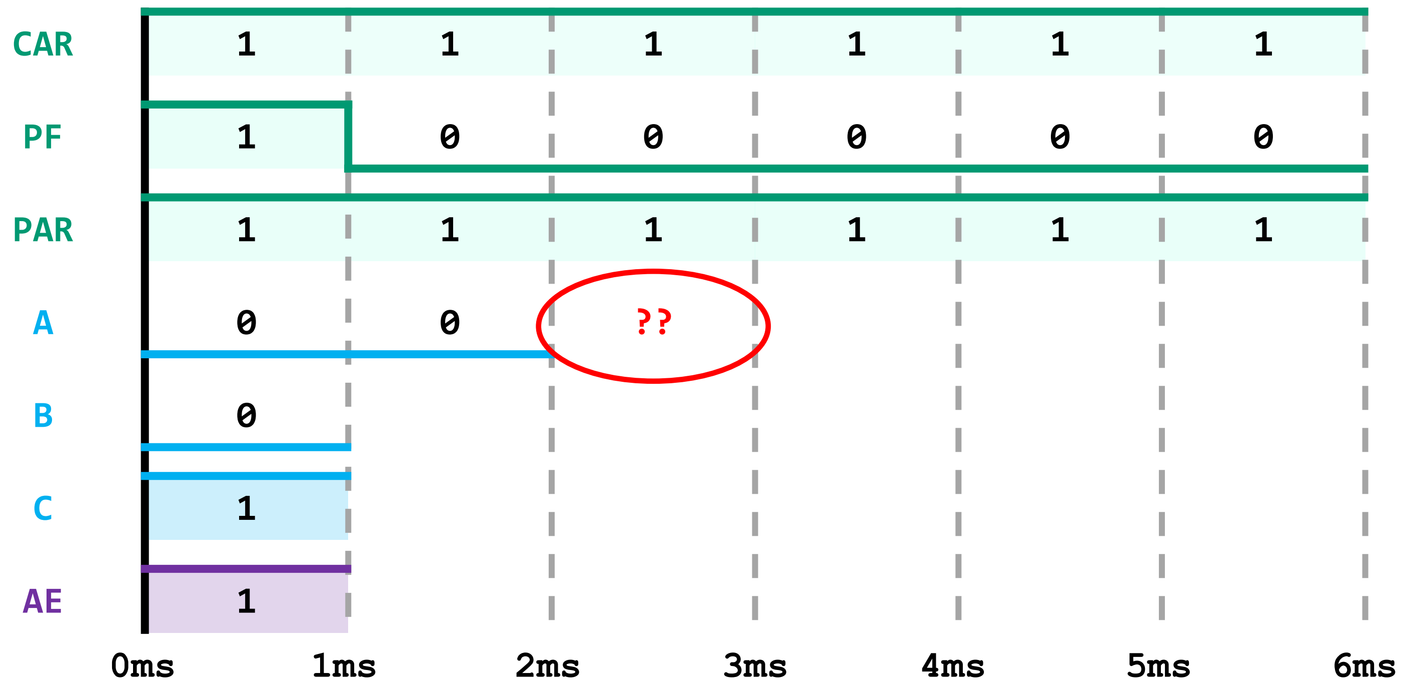




Poll Everywhere

Vote at https://pollev.com/cse390b

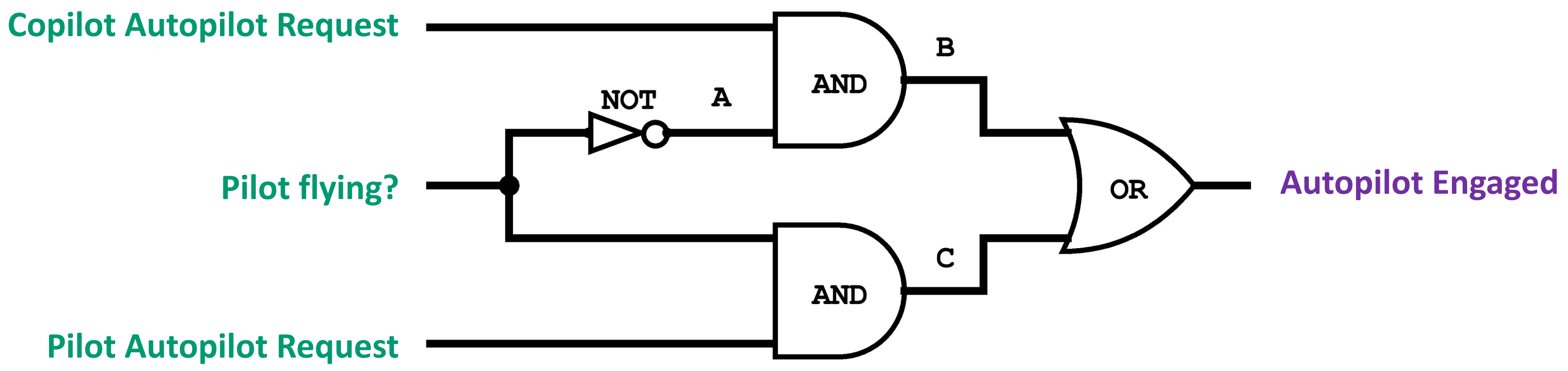


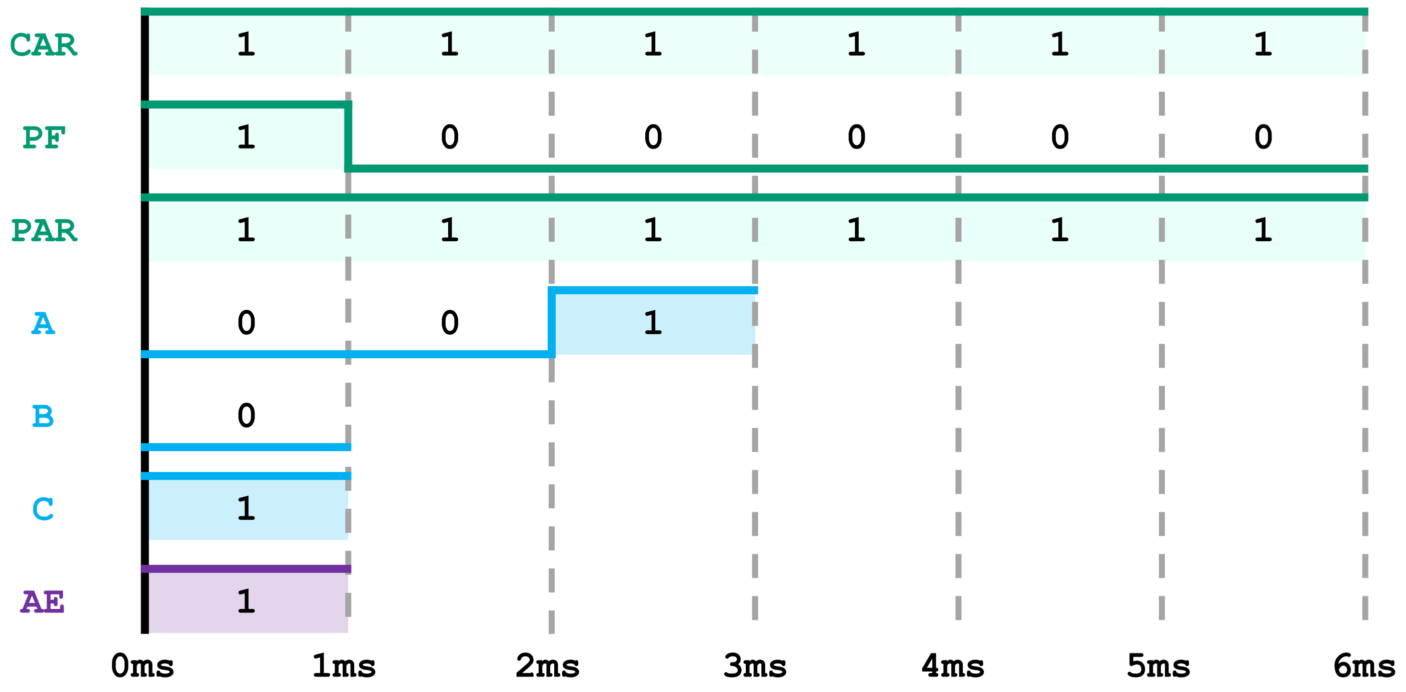


	0ms–1ms	1ms–2ms	2ms–3ms	3ms–4ms	4ms–5ms	5ms–6ms
CAR	1	1	1	1	1	1
PF	1	0	0	0	0	0
PAR	1	1	1	1	1	1
A	0	0	??			
B	0					
C	1					
AE	1					

Poll Everywhere

Vote at https://pollev.com/cse390b

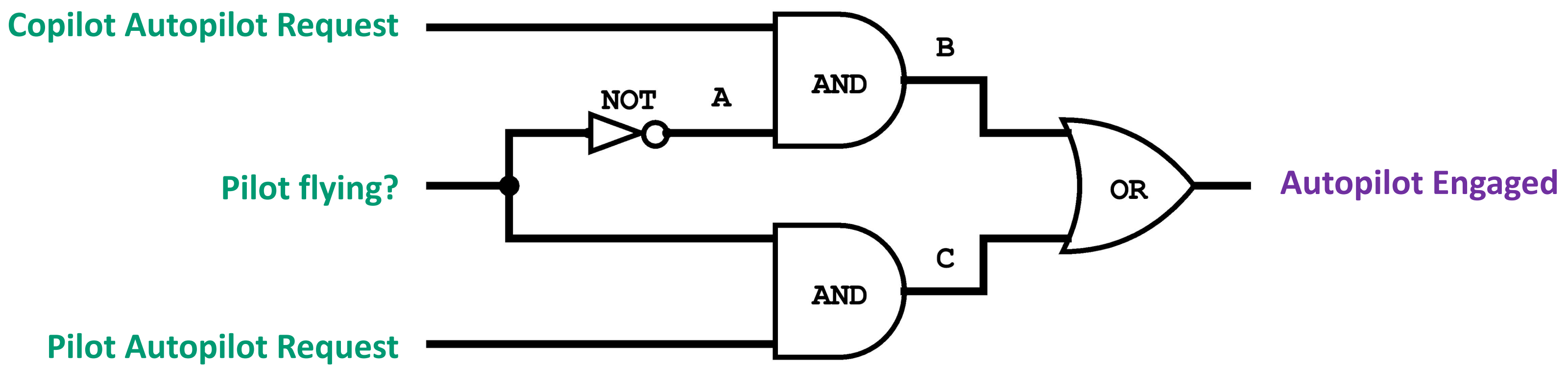


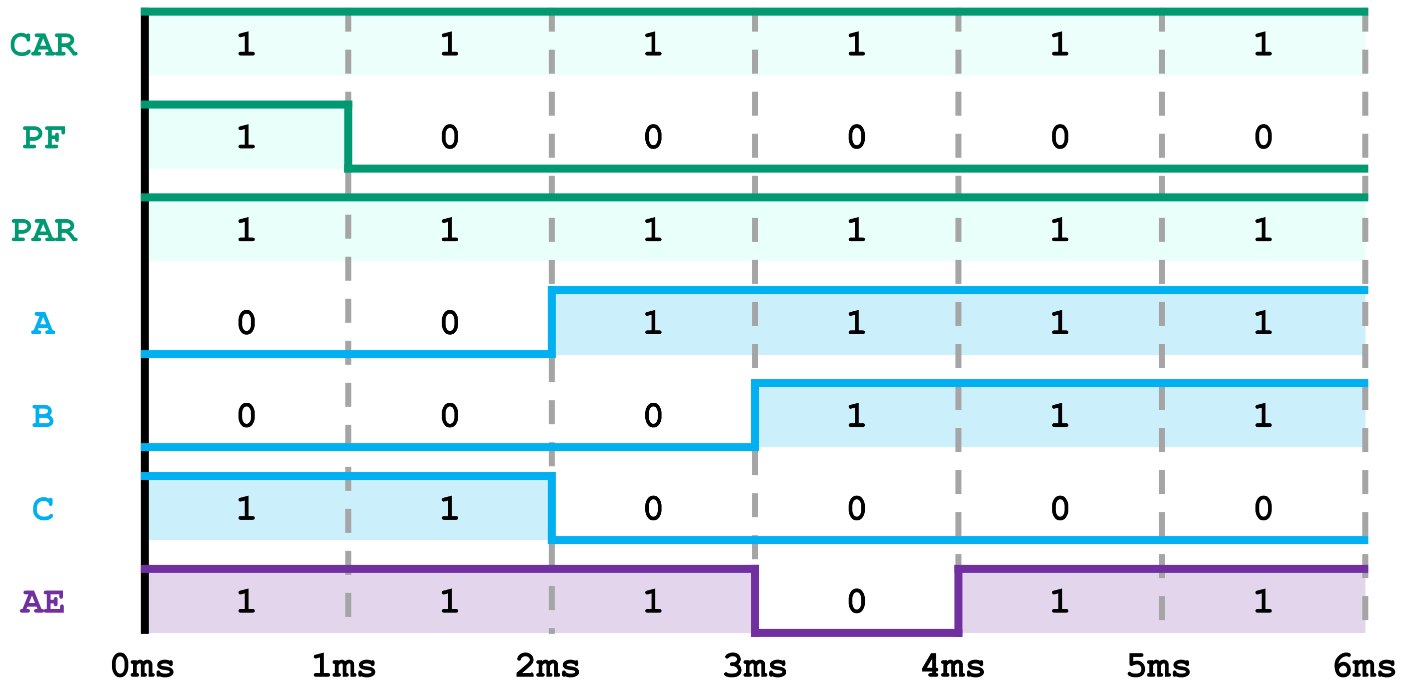


	0–1ms	1–2ms	2–3ms	3–4ms	4–5ms	5–6ms
CAR	1	1	1	1	1	1
PF	1	0	0	0	0	0
PAR	1	1	1	1	1	1
A	0	0	1			
B	0					
C	1					
AE	1					

Poll Everywhere

Vote at https://pollev.com/cse390b

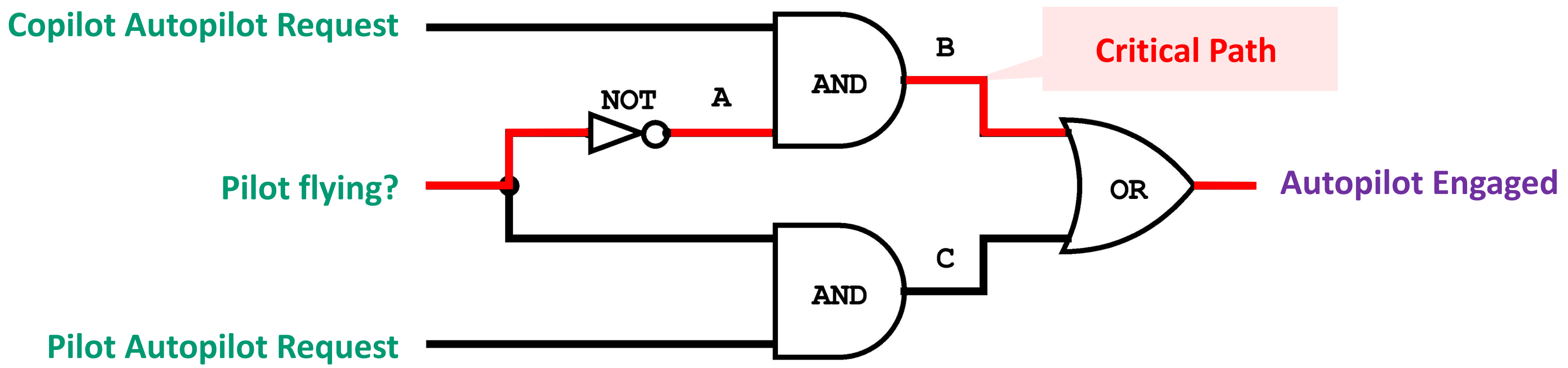


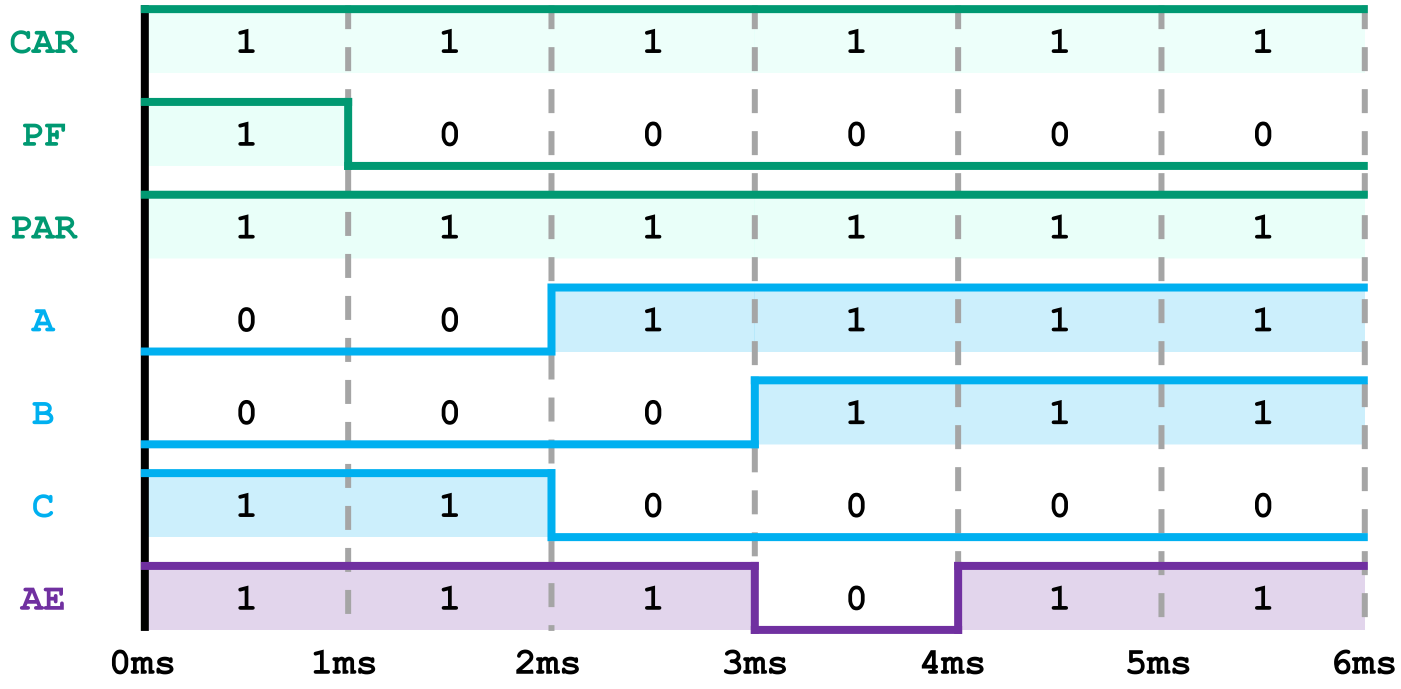


	0–1ms	1–2ms	2–3ms	3–4ms	4–5ms	5–6ms
CAR	1	1	1	1	1	1
PF	1	0	0	0	0	0
PAR	1	1	1	1	1	1
A	0	0	1	1	1	1
B	0	0	0	1	1	1
C	1	1	0	0	0	0
AE	1	1	1	0	1	1

Poll Everywhere

Vote at https://pollev.com/cse390b

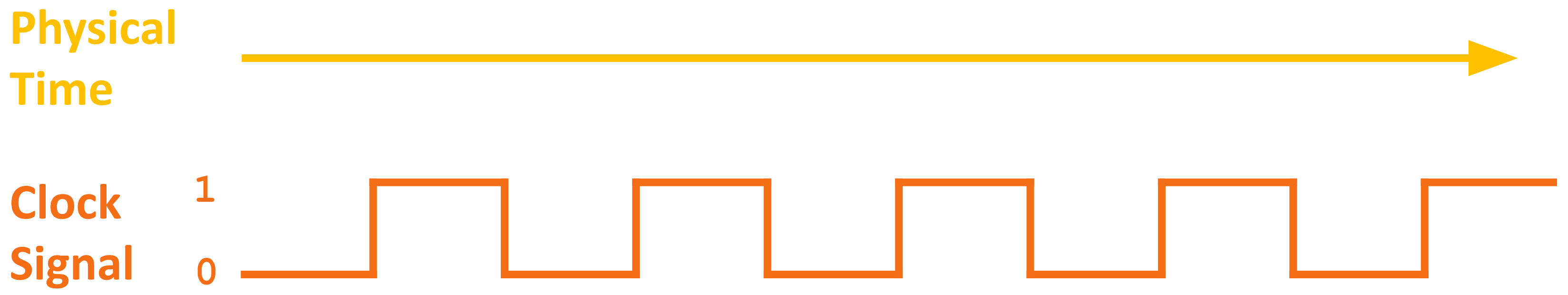




	0–1ms	1–2ms	2–3ms	3–4ms	4–5ms	5–6ms
CAR	1	1	1	1	1	1
PF	1	0	0	0	0	0
PAR	1	1	1	1	1	1
A	0	0	1	1	1	1
B	0	0	0	1	1	1
C	1	1	0	0	0	0
AE	1	1	1	0	1	1

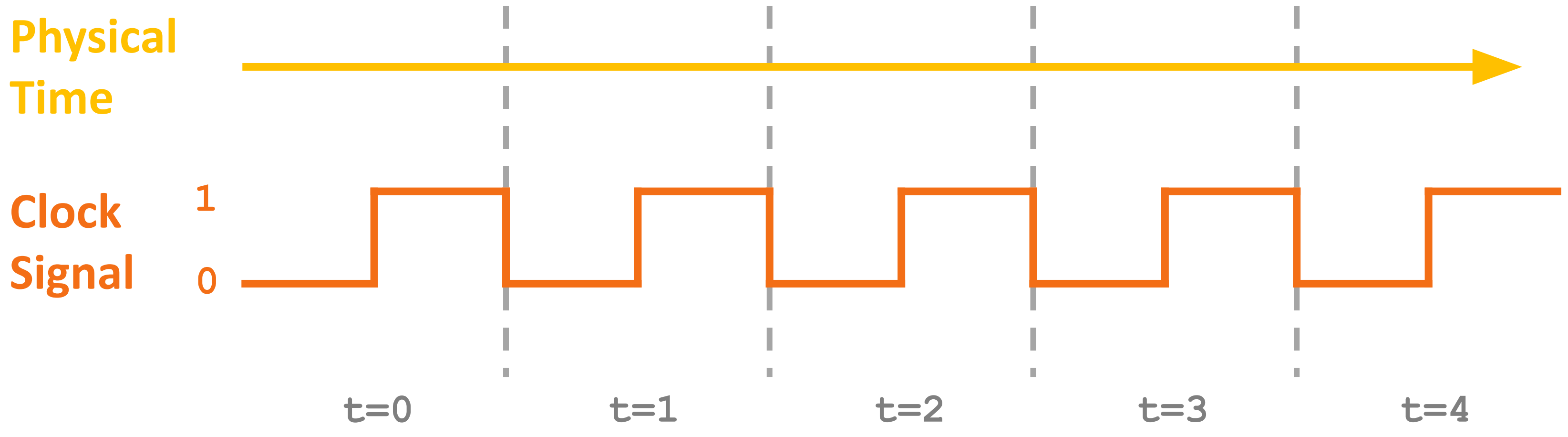
Physical Timekeeping

- Hardware keeps track of time using an alternating signal
  - Creates the idea of **discrete time:** state changes only occur in discrete intervals, right when signal alternates



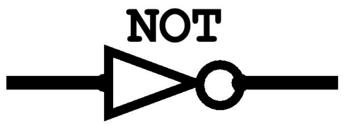
Physical Timekeeping

- Hardware keeps track of time using an alternating signal
  - Creates the idea of **discrete time:** state changes only occur in discrete intervals, right when signal alternates

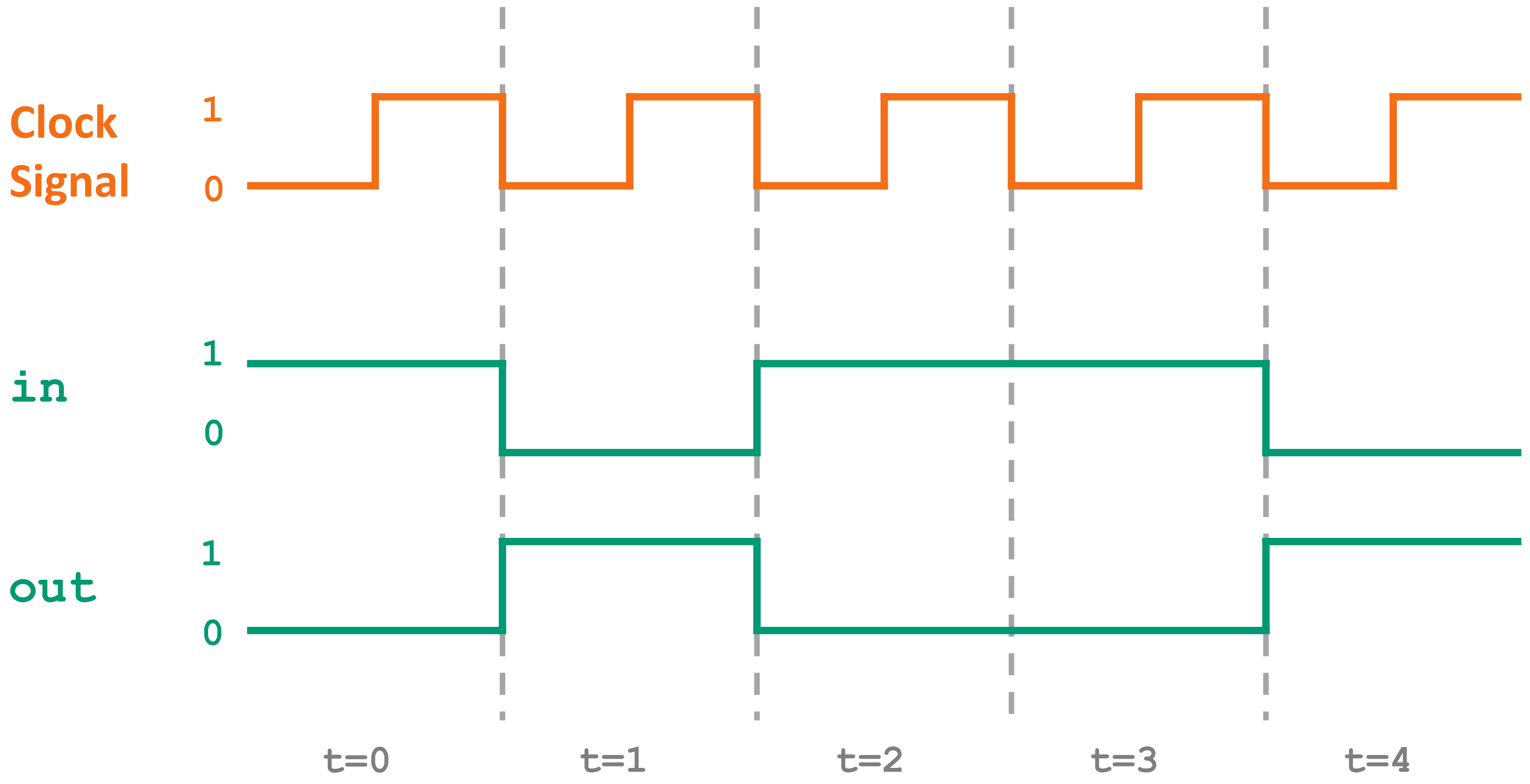


Discrete Time Intervals

Adding a Clock: Ideal

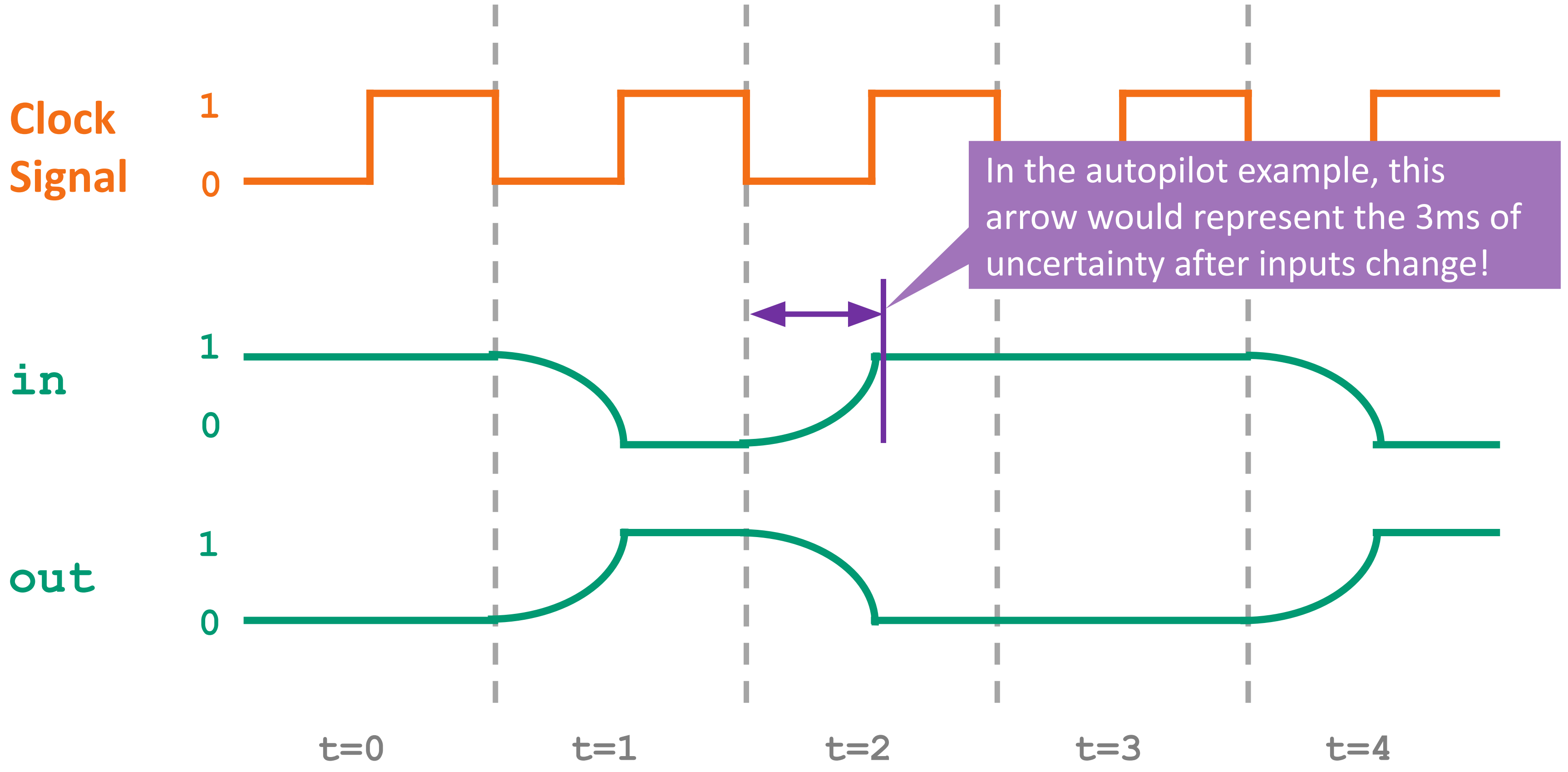


- We want this behavior from a simple, combinational Not gate:



Adding a Clock: Reality

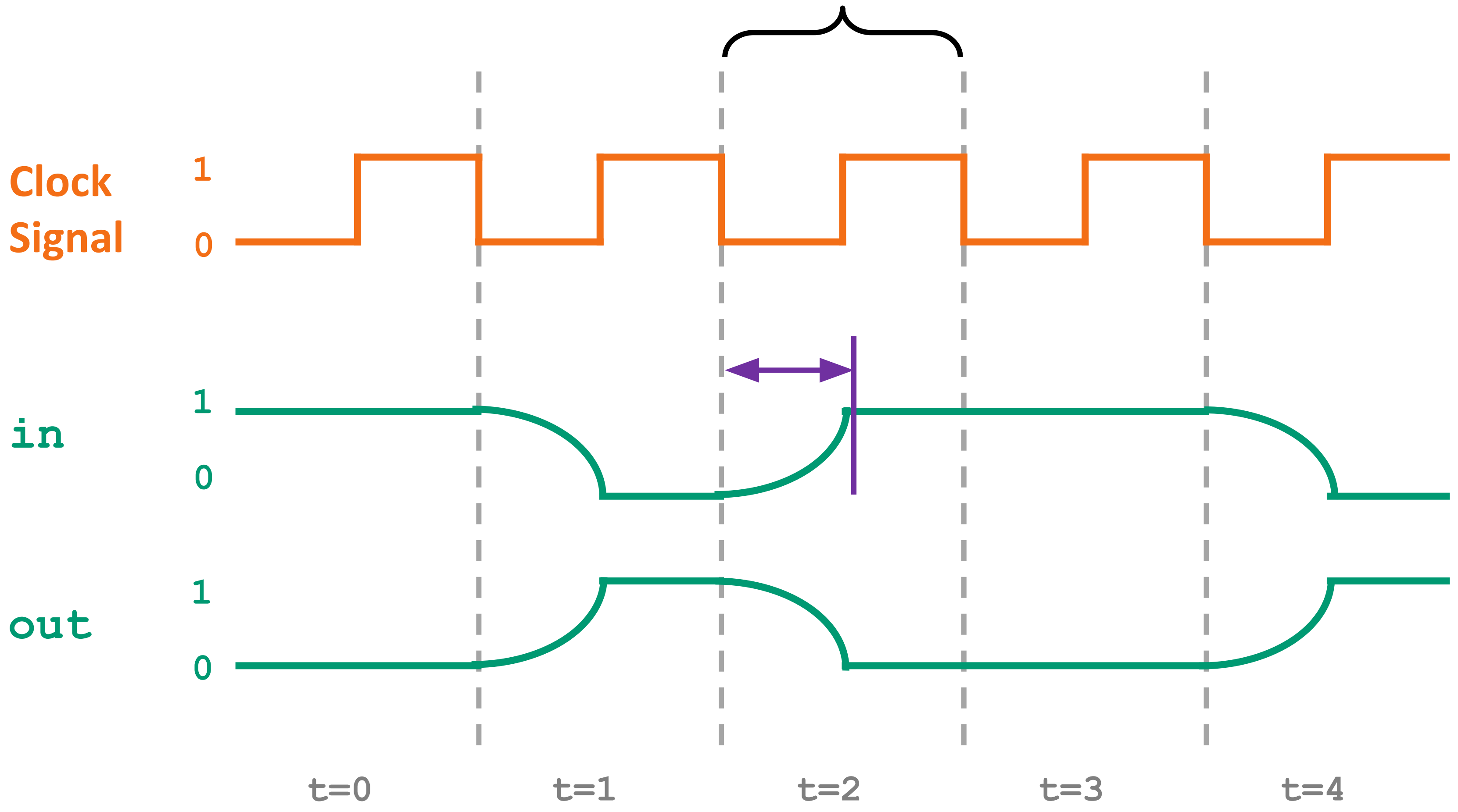
- Combinational logic may be incorrect for a period immediately after inputs change
  - **Computation delays** (logic gates) and **propagation delays** (wires)



In the autopilot example, this arrow would represent the 3ms of uncertainty after inputs change!

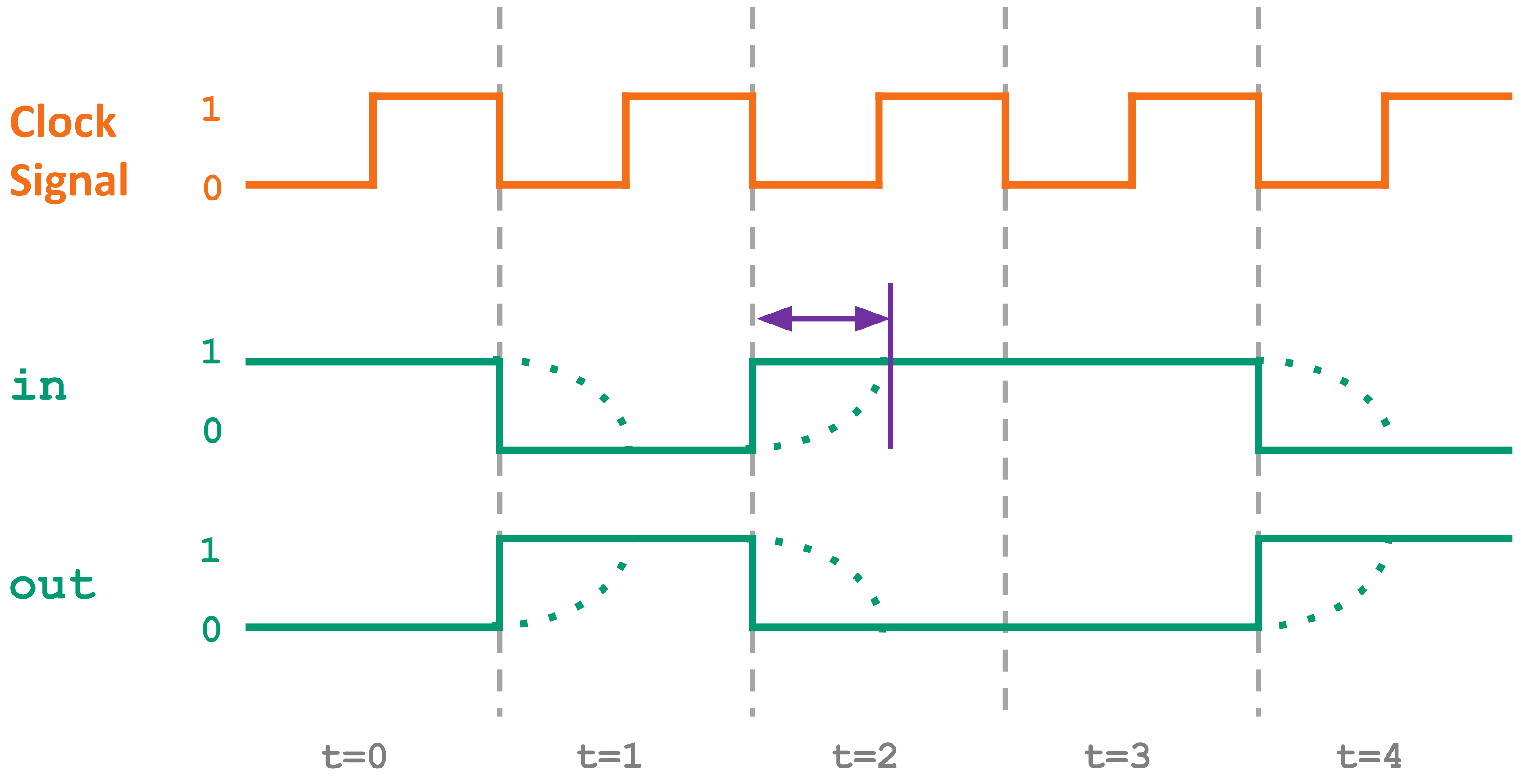
Adding a Clock: Clock Cycles

- Choose a clock cycle length slightly longer than the delays



Adding a Clock: Abstraction

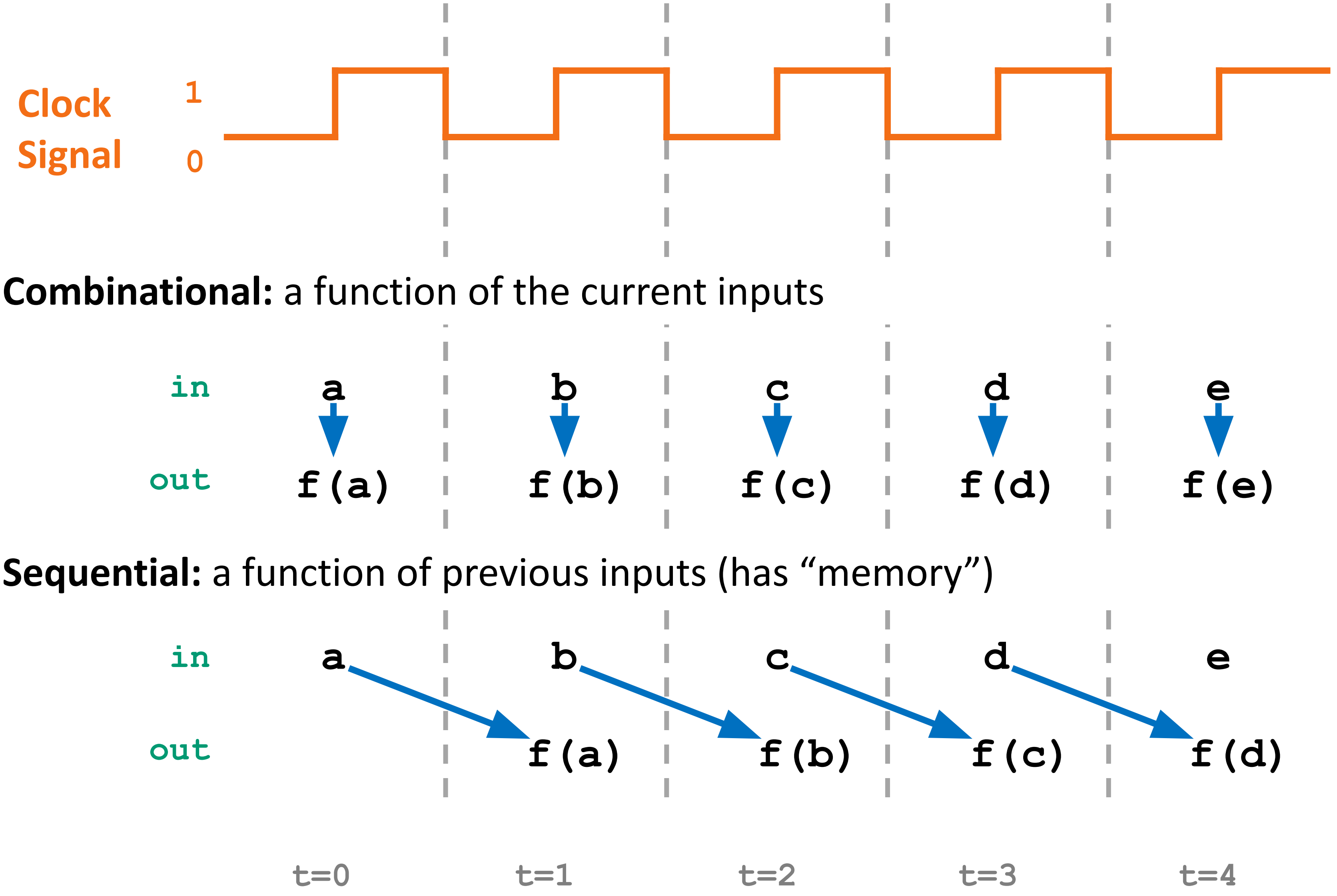
- If we use a long enough clock cycle, we can *pretend* that combinational chips (like Not) work instantly



Lecture Outline

- Bloom's Taxonomy
  - Apply levels of learning to coursework
- Cornell Note-Taking Method
  - System for taking, organizing, and reviewing notes
- Representing Time in Hardware
  - Sequential Logic vs. Combinational Logic
  - Adding a clock
- **Sequential Logic**
  - **Data Flip-Flip (DFF) implementation and examples**

Combinational vs. Sequential Abstraction



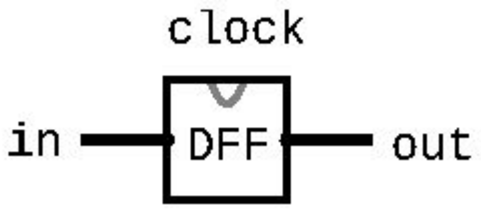
Combinational: a function of the current inputs

Sequential: a function of previous inputs (has "memory")

The Flip-Flop Gate

- Simplest state-keeping component
  - 1-bit input, 1-bit output
  - Wired to the clock signal
  - Always outputs its previous input: **`out(t) = in(t-1)`**

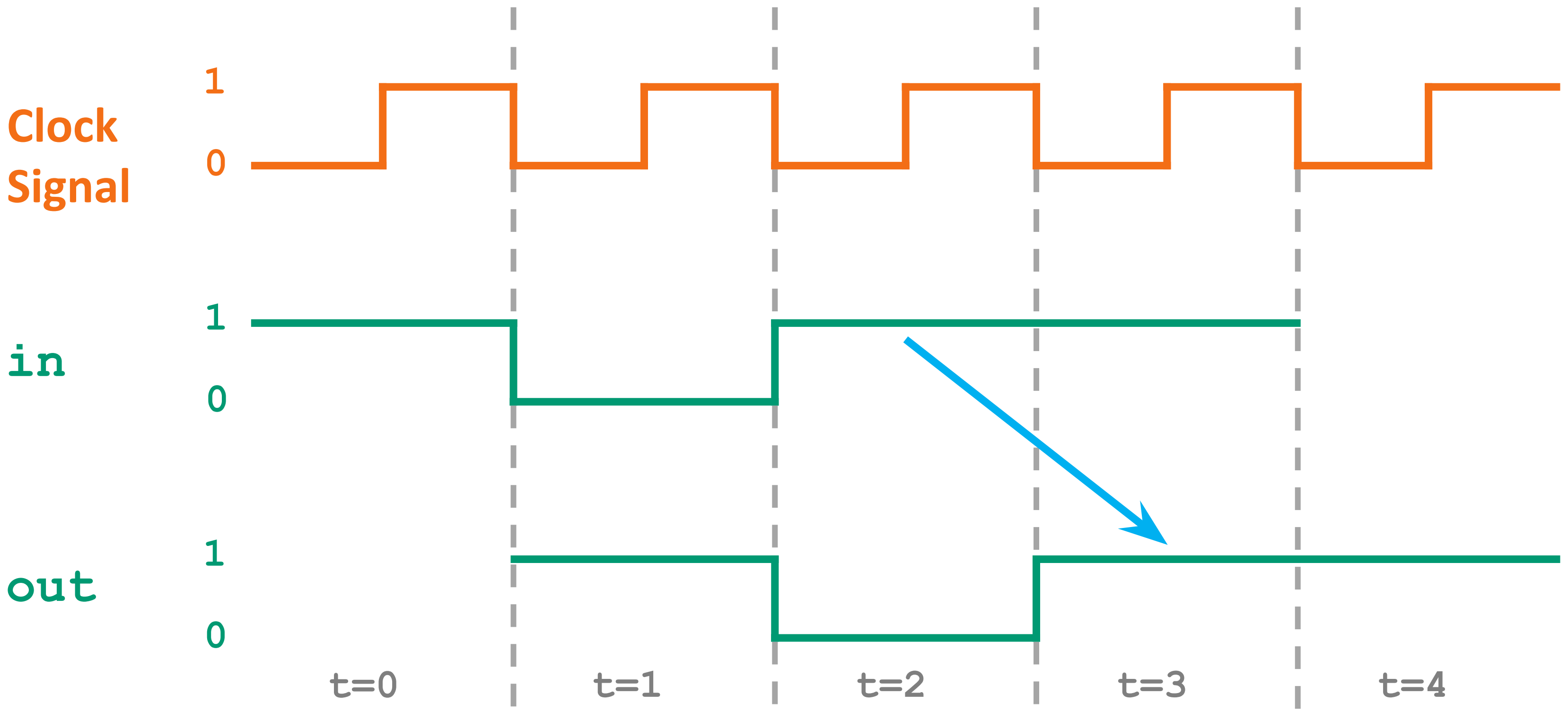
- Implementation: a gate that can flip between two stable states (remembering 0 vs. remembering 1)
  - Gates with this behavior are "Data Flip Flops" (DFFs)

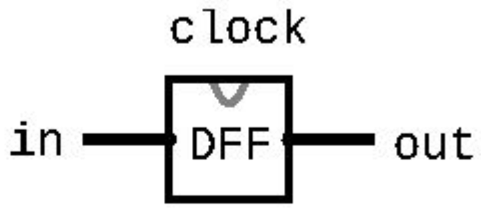


Aside: Treating Flip-Flop as Primitive

- Disclaimer: CAN be made from Nand gates exclusively!
  - But requires wiring them together in a "messy" loop that the hardware simulator can't simulate and isn't very educational
- For simplicity, we will treat Flip-Flop as primitive in the projects
  - Just like Nand, you can use the built-in implementation

Flip-Flop Behavior





Sequential Chips

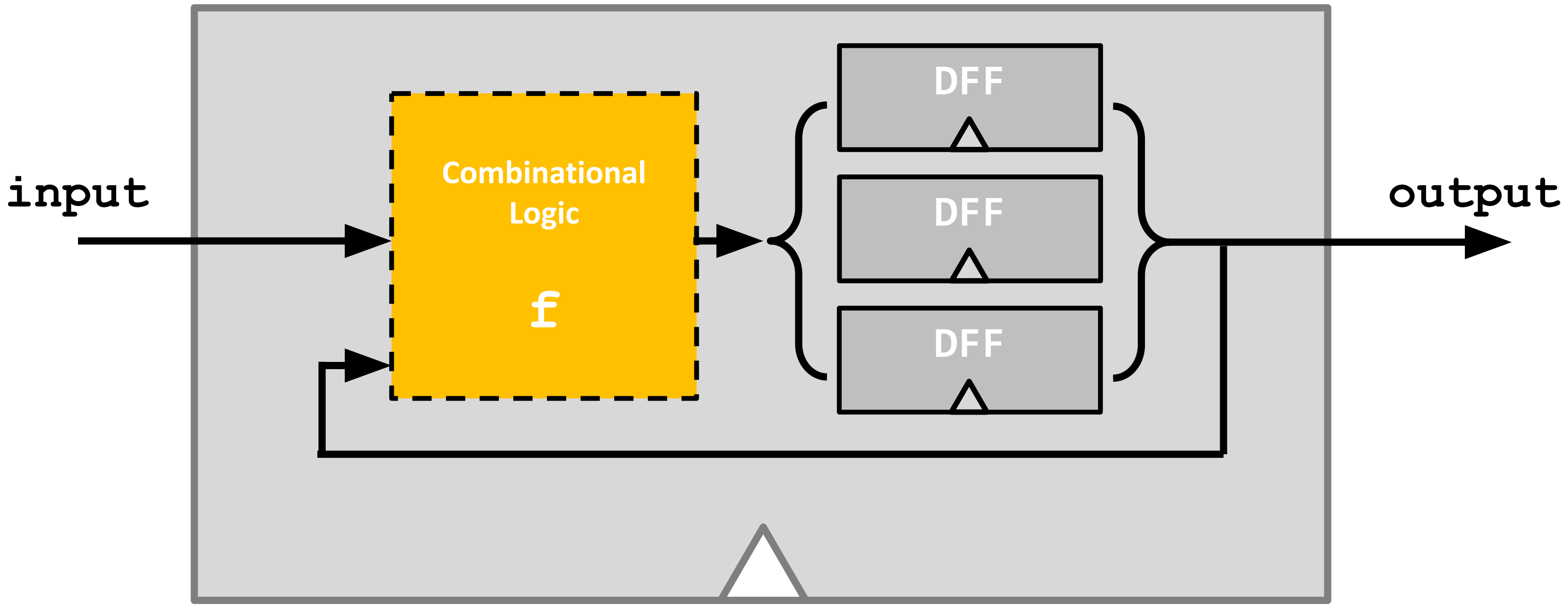
- A category of chips that utilize the clock signal, in addition to any combinational logic

- Capable of:
  - Maintaining state
  - Optionally, acting on that state & current inputs
    - Can incorporate combinational logic as well

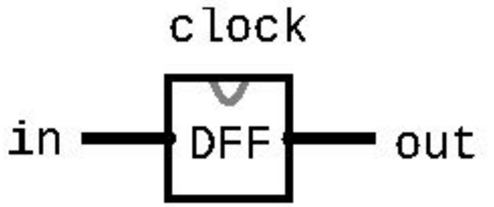
- Constructed from:
  - DFFs
  - Combinational logic (which is entirely constructed from Nand)

Sequential Chips

`output(t) = f(state(t-1), input(t))`



Flip-Flop: Time Series



- DFF Specification:

```
out(t) = in(t-1)
```

in	0	0	1	1	0	1	0	...
out	0	0	0	1	1	0	1	...
time	t=0	t=1	t=2	t=3	t=4	t=5	t=6	...

Example: `out(t=3)` = `in(t=2)`

DFF Example 1: Specification

- Example specification:

```
out(t) = Xor(a(t-1), b(t-1))
```

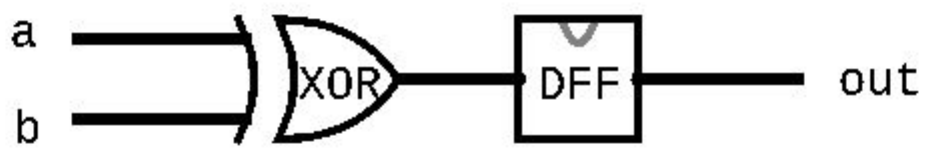
- Takes two inputs, `a` and `b`, and outputs the `Xor` of them
  - Note that out at time `t` is determined by `a` and `b` at time `t-1`
  - We will need to use a DFF!
- Exercise: Draw out the corresponding circuit diagram

DFF Example 1: Circuit Diagram

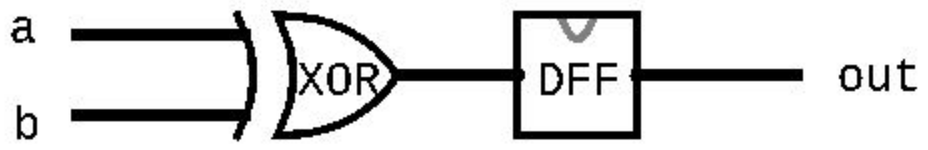
- Example specification:

```
out(t) = Xor(a(t-1), b(t-1))
```

- Circuit diagram:



DFF Example 1: HDL Implementation

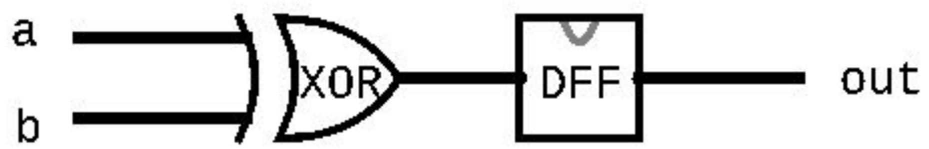


```
CHIP Example1 {
    IN a, b;
    OUT out;

    PARTS:
    Xor(a=a, b=b, out=xorout);
    DFF(in=xorout, out=out);
}
```

DFF Example 1: Time Series

- Example specification:



`out(t) = Xor(a(t-1), b(t-1))`

a	0	0	1	1	1	0	0	...
b	0	1	0	1	1	1	0	...
out	0	0	1	1	0	0	1	...
time	t=0	t=1	t=2	t=3	t=4	t=5	t=6	...

Example: `out(t=3) = Xor(a(t=2), b(t=2))`

DFF Example 2: Specification

- Example specification:

```
out(t) = Xor(out(t-1), in(t-1))
```

- Notice how the specification uses **`out(t-1)`** as an input for **`out(t)`**
  - Implies the necessity of circular wiring, separated by a DFF

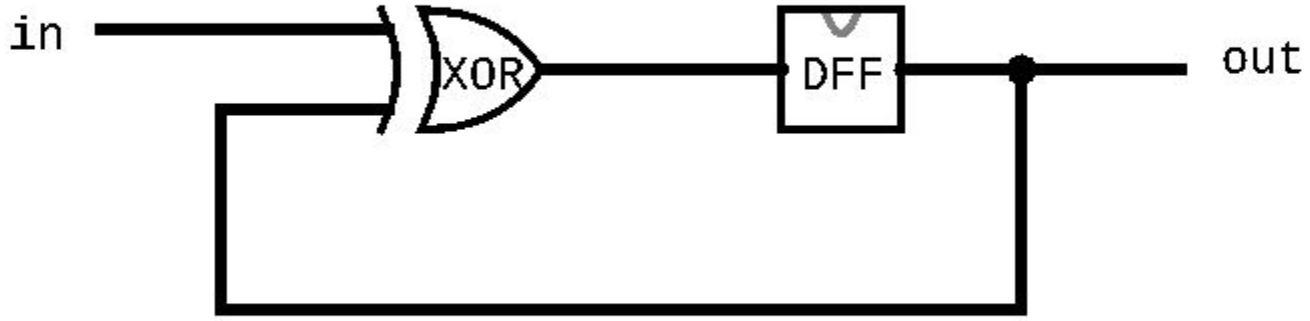
- Exercise: Draw out the corresponding circuit diagram

DFF Example 2: Circuit Diagram

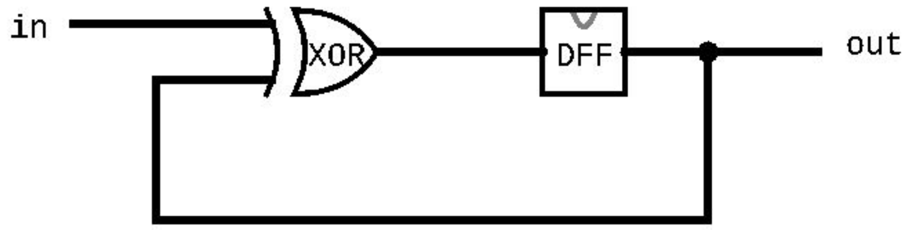
- Example specification:

```
out(t) = Xor(out(t-1), in(t-1))
```

- Circuit diagram:



DFF Example 2: HDL Implementation



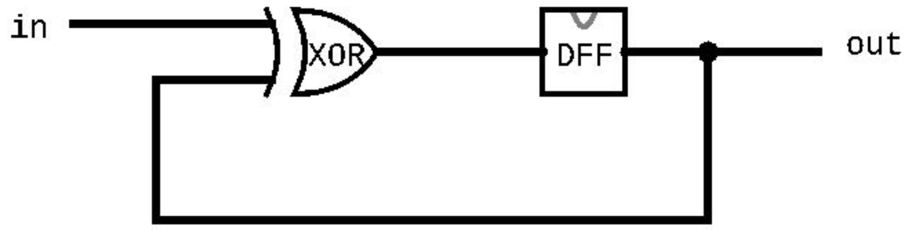
- Example specification:

```
out(t) = Xor(out(t-1), in(t-1))
```

- Exercise: Write out the HDL Implementation

DFF Example 2: HDL Implementation

- Example specification:



out(t) = Xor(out(t-1), in(t-1))

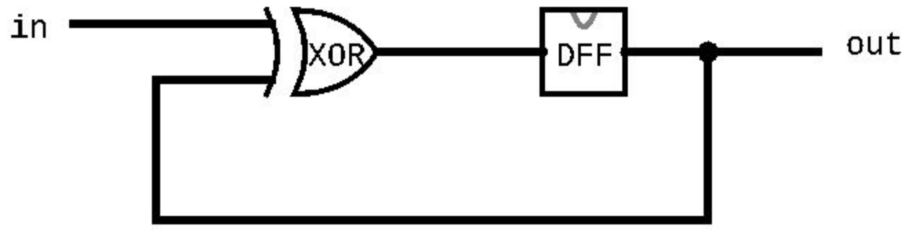
- Exercise: Write out the HDL Implementation

```
CHIP Example2 {
    IN in;
    OUT out;

    PARTS:
    Xor(a=in, b=prevout, out=xorout);
    DFF(in=xorout, out=prevout, out=out);
}
```

DFF Example 2: Time Series

- Example specification:



`out(t) = Xor(out(t-1), in(t-1))`

in	0	0	1	1	1	0	0	...
out	0	0	0	1	0	1	1	...
time	t=0	t=1	t=2	t=3	t=4	t=5	t=6	...

Example: `out(t=1) = Xor(in(t=0), out(t=0))`

DFF Example 3: Specification

- Example specification:

```
out(t) = And(Not(out(t-1)), in(t-1))
```

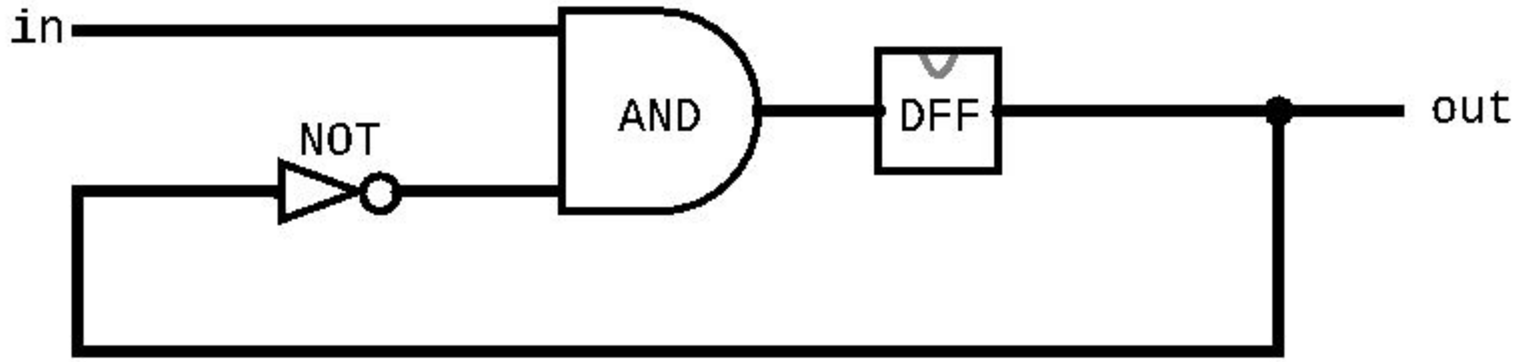
- Exercise: Draw out the corresponding circuit diagram

DFF Example 3: Circuit Diagram

- Example specification:

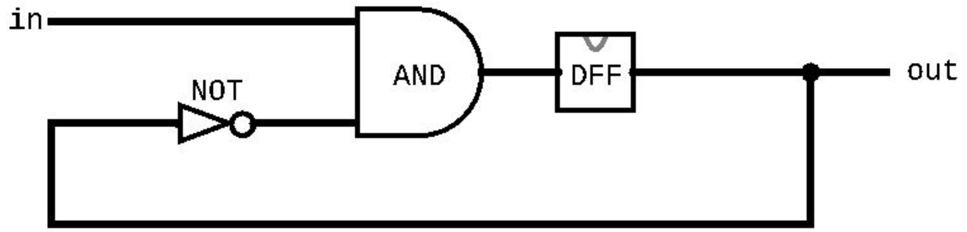
```
out(t) = And(Not(out(t-1)), in(t-1))
```

- Exercise: Draw out the corresponding circuit diagram



DFF Example 3: HDL Implementation

- Example specification:

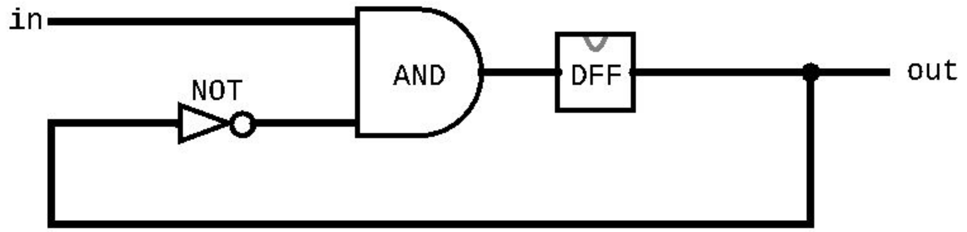


```
out(t) = And(Not(out(t-1)), in(t-1))
```

- Exercise: Write out the HDL Implementation

DFF Example 3: HDL Implementation

- Example specification:



```
out(t) = And(Not(out(t-1)), in(t-1))
```

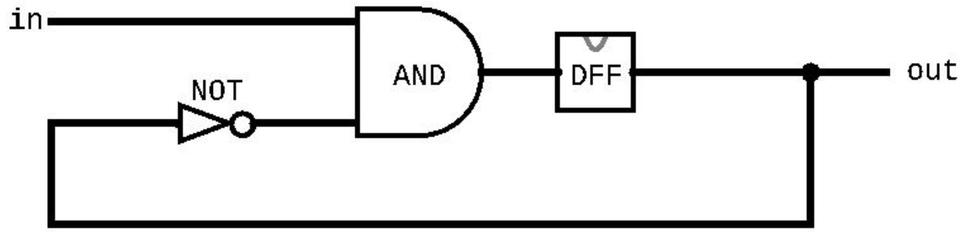
- Exercise: Write out the HDL Implementation

```
CHIP Example3 {
    IN in;
    OUT out;

    PARTS:
    Not(in=prevout, out=notprevout);
    And(a=in, b=notprevout, out=andout);
    DFF(in=andout, out=prevout, out=out);
}
```

DFF Example 3: Time Series

- Example specification:



`out(t) = And(Not(out(t-1)), in(t-1))`

in	1	1	0	1	1	0	0	...
out	0	1	0	0	1	0	0	...
time	t=0	t=1	t=2	t=3	t=4	t=5	t=6	...

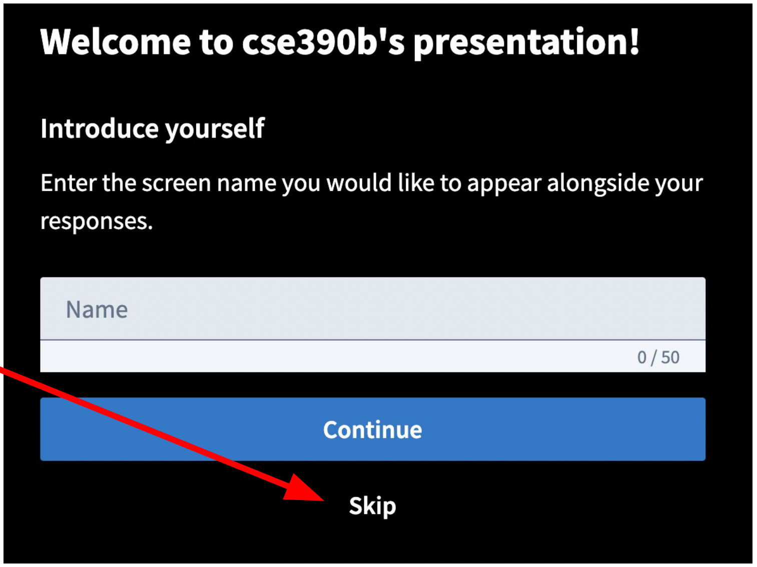
Example: `out(t=1) = And(Not(out(t=0)), in(t=0))`

Poll Everywhere

Vote at https://pollev.com/cse390b

- **Are you available to attend at least one CSE 390B office hours? If not, what days and times would work better?**

- You can choose to respond anonymously by not entering your name (click "Skip")


Welcome to cse390b's presentation!

Introduce yourself

Enter the screen name you would like to appear alongside your responses.

Lecture 5 Reminders

- Project 1 grades released by tonight
  - Feedback can be viewed on Gradescope
- Project 2: Boolean Arithmetic, 24-Hour Time Audit
  - **Due on Thursday (1/20) at 11:59PM PST**
- Office Hours: Zoom links available via Canvas
  - Eric: Tuesdays and Thursdays, 3-4:00pm at CSE2 153
  - Leslie: Wednesdays, 4:30-5pm at CSE2 174
  - Audrey and Sean: Wednesdays, 1:30-2:30pm at CSE2 152
- Continue to ask your questions on the Ed board!