CSE 390 B Spring 2021

Sequential Logic & Bloom's Taxonomy

Project 1 Reflection, Introduction to Bloom's Taxonomy, Cornell Note-Taking, Continuation of Circuit Design, & Physical Timekeeping

Agenda

- **Project 1 Reflection**
- Bloom's Taxonomy
- Cornell Note-Taking Method
- Representing Time in Hardware
- Sequential Logic

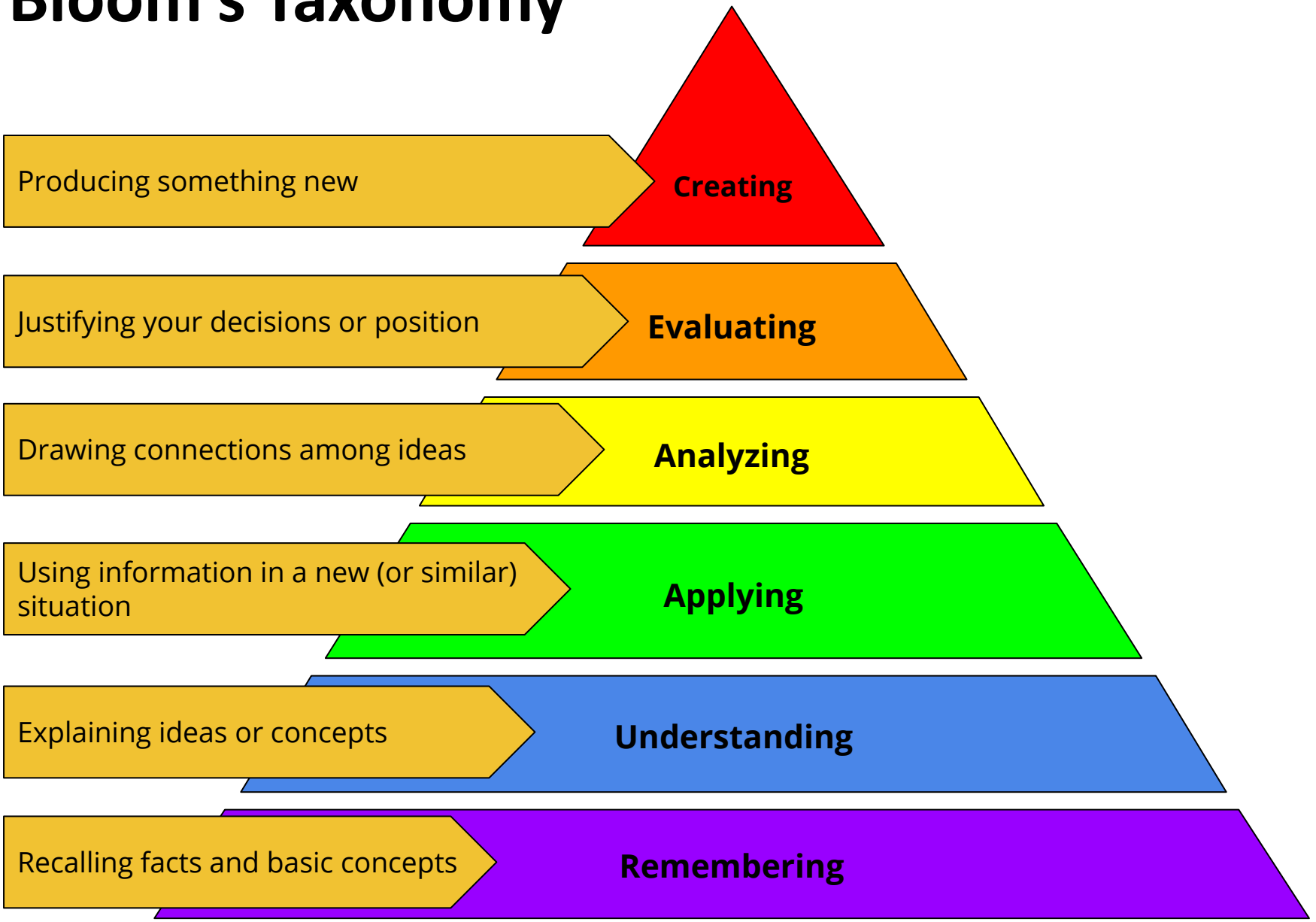
Project 1 Discussion

- What was the most frustrating or difficult part of the Boolean Logic assignment?
- What tools, questions, strategies did you utilize to persist through the assignment?
- What discrepancy did you have been your *estimated* time to complete the assignment and your *actual* time? Why?

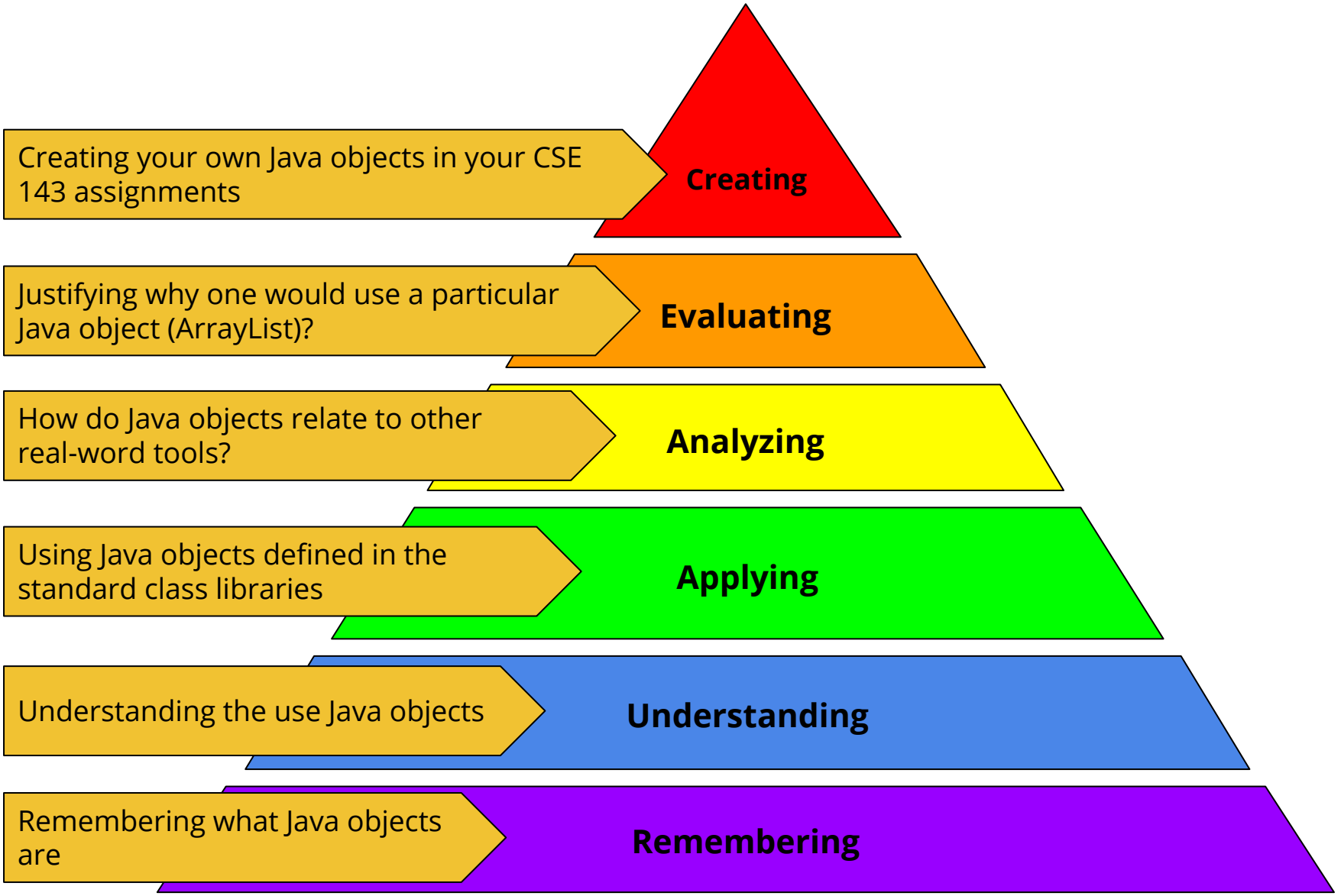
Agenda

- Project 1 Reflection
- **Bloom's Taxonomy**
- Cornell Note-Taking Method
- Representing Time in Hardware
- Sequential Logic

Bloom's Taxonomy



Bloom's Taxonomy in Action



Agenda

- Project 1 Reflection
- Bloom's Taxonomy
- **Cornell Note-Taking Method**
- Representing Time in Hardware
- Sequential Logic

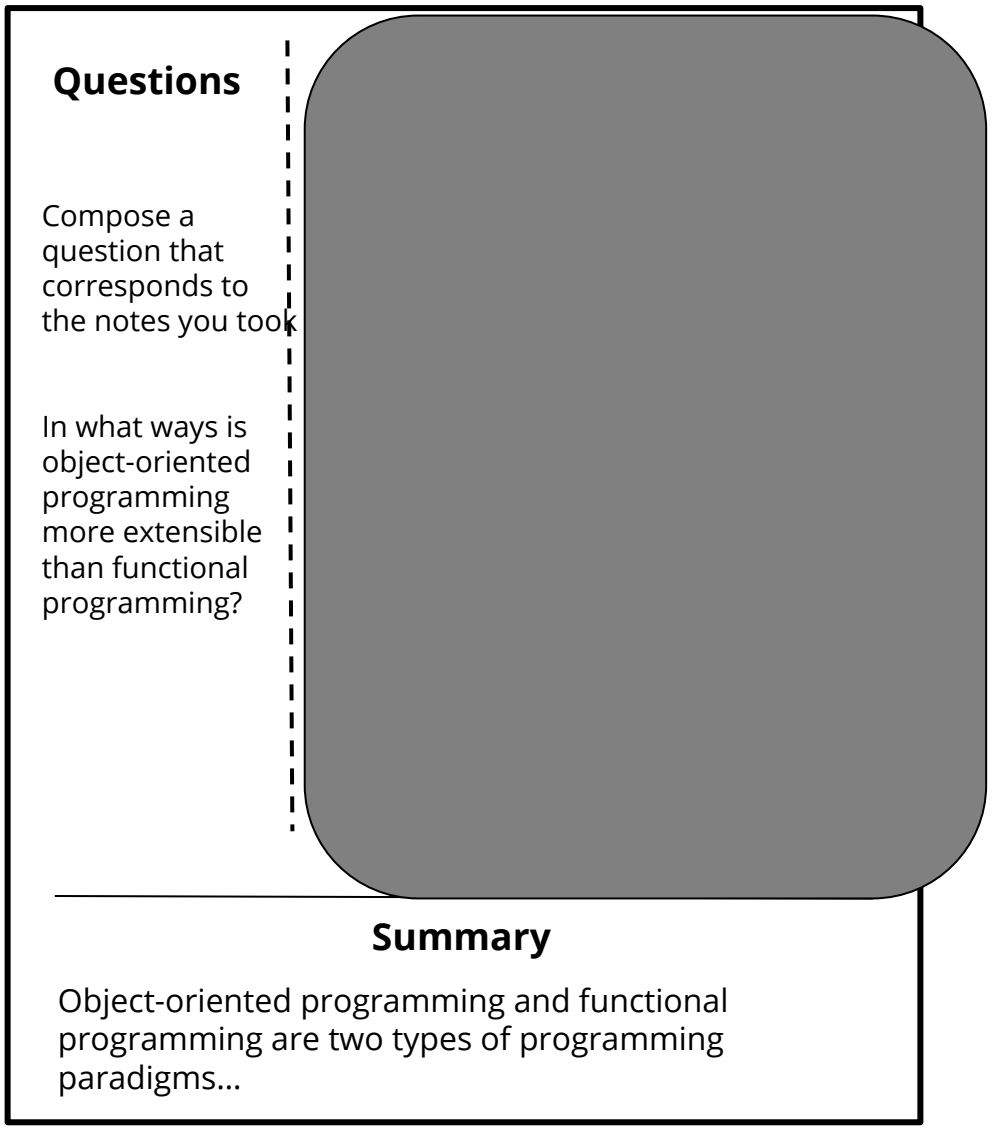
Cornell Note-Taking Method

Questions	Notes
Compose a question that corresponds to the notes you took	I. Main Topic ○ Sub point ○ definition ○ example **
In what ways is object-oriented programming more extensible than functional programming?	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 add new data types
	III. Functional Programming ○ Extensibility - easier to add new operations

Summary

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

Cornell Note-Taking Method



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)

Agenda

- Project 1 Reflection
- Bloom's Taxonomy
- Cornell Note-Taking Method
- **Representing Time in Hardware**
- Sequential Logic

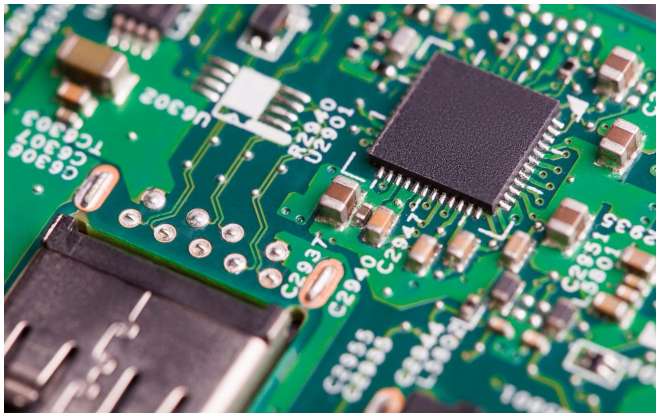
Combinational Logic vs. Sequential Logic

- So far, we ignored "time" in our circuits
- Our chips use **combinational logic**
  - When given inputs, the chip computes its output "instantaneously"
  - The output is a pure function of the current inputs, 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:
  - For memory, we need to talk about hardware maintaining state. To do so, we need a vocabulary to talk about time.

- Needed for Implementation:
  - Physical implementations of chips **cannot be instantaneous**. We need to account for physical delays in signal propagation.

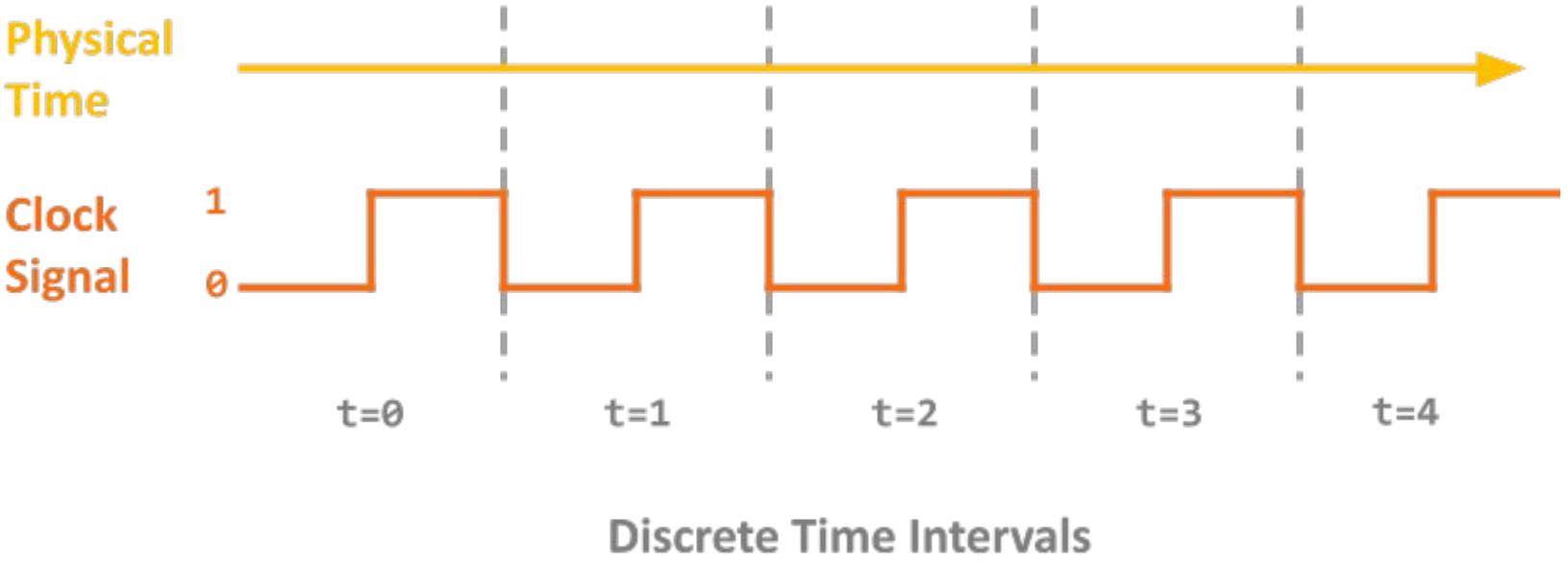


Physical Timekeeping

- In hardware, keep track of time with an alternating signal
  - Creates idea of **discrete time** -- state changes only occur in discrete intervals, right when signal alternates

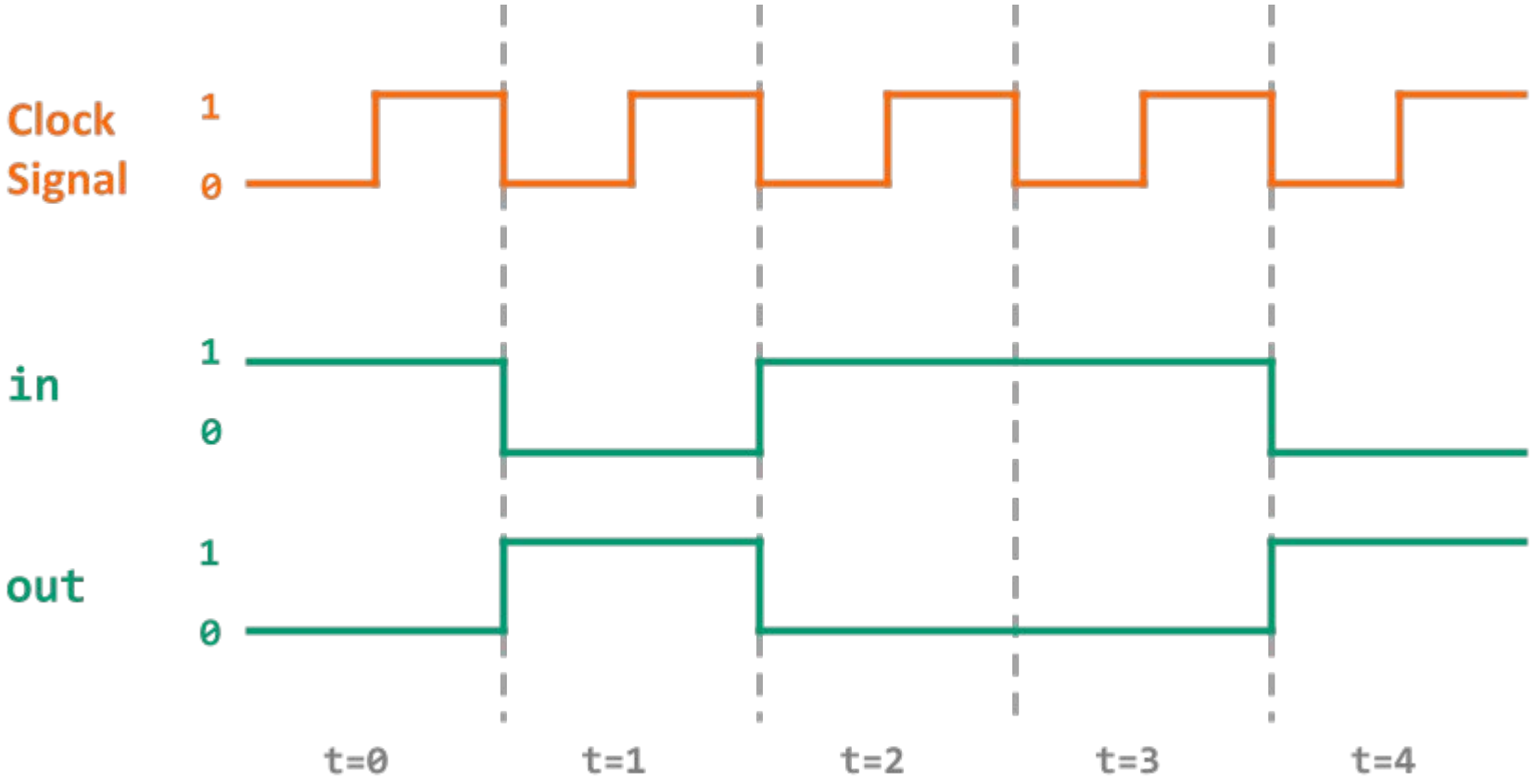
Physical Timekeeping

- In hardware, keep track of time with an alternating signal
  - Creates idea of **discrete time** -- state changes only occur in discrete intervals, right when signal alternates



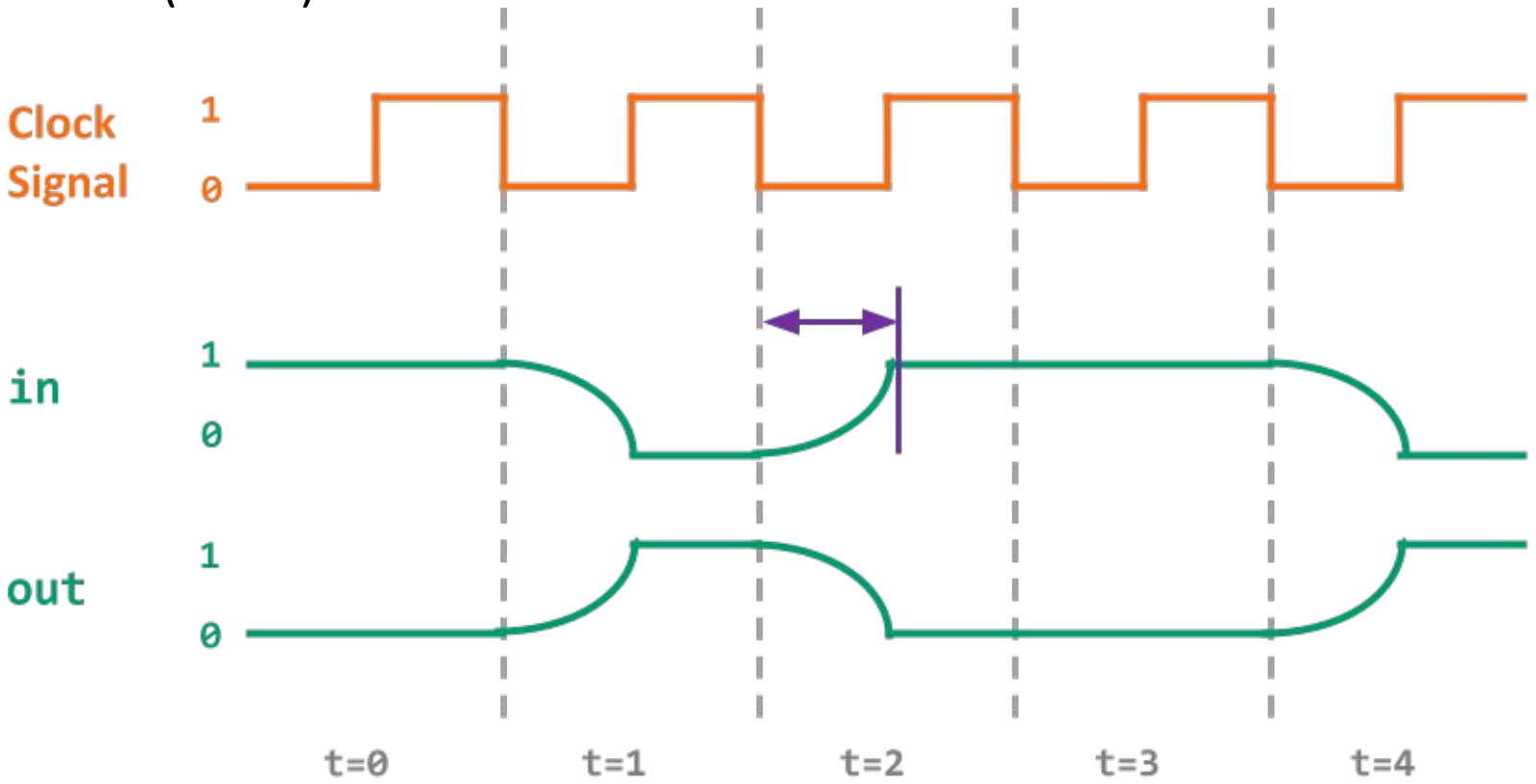
Adding a Clock: Ideal

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



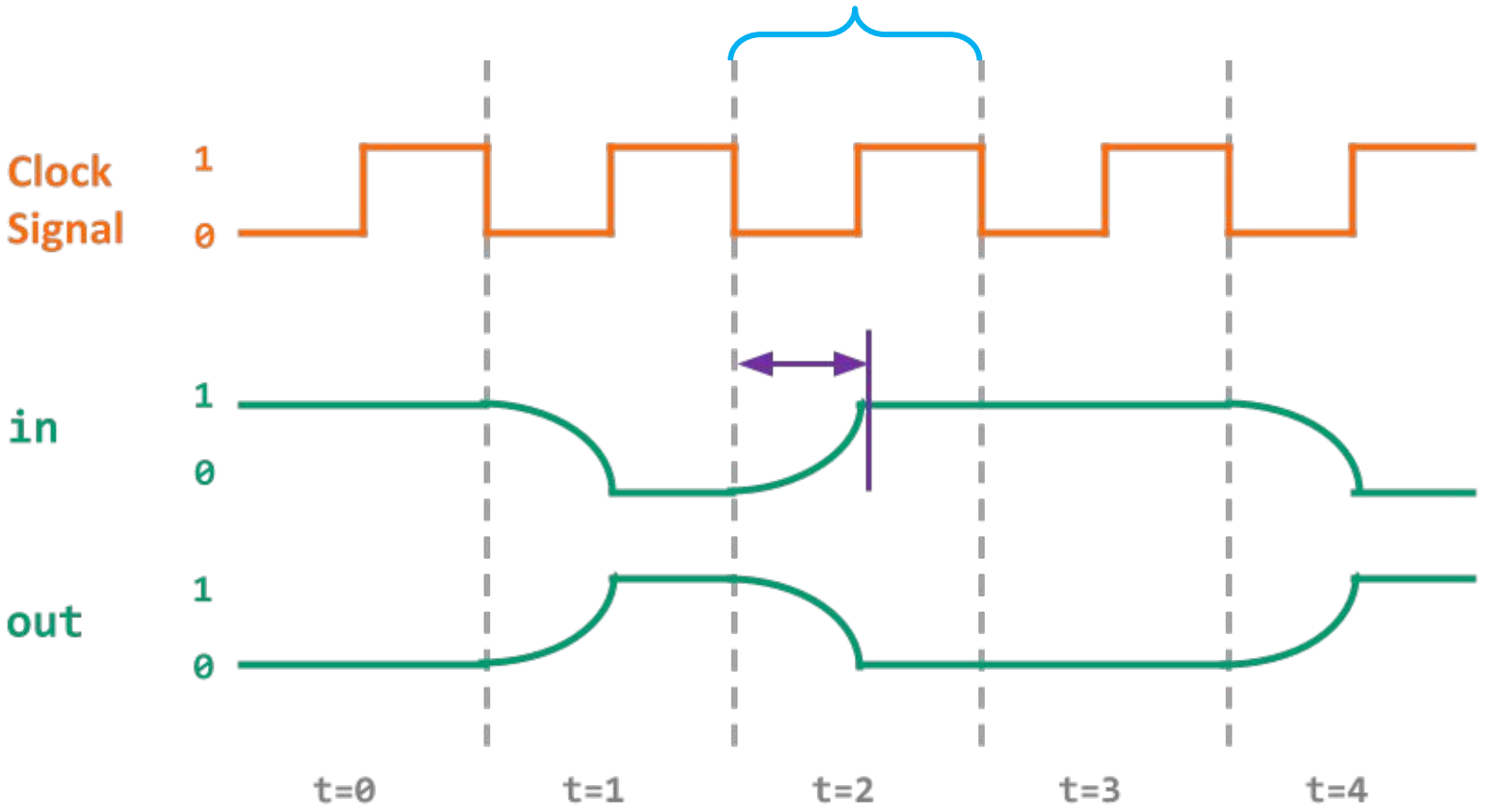
Adding a Clock: Reality

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



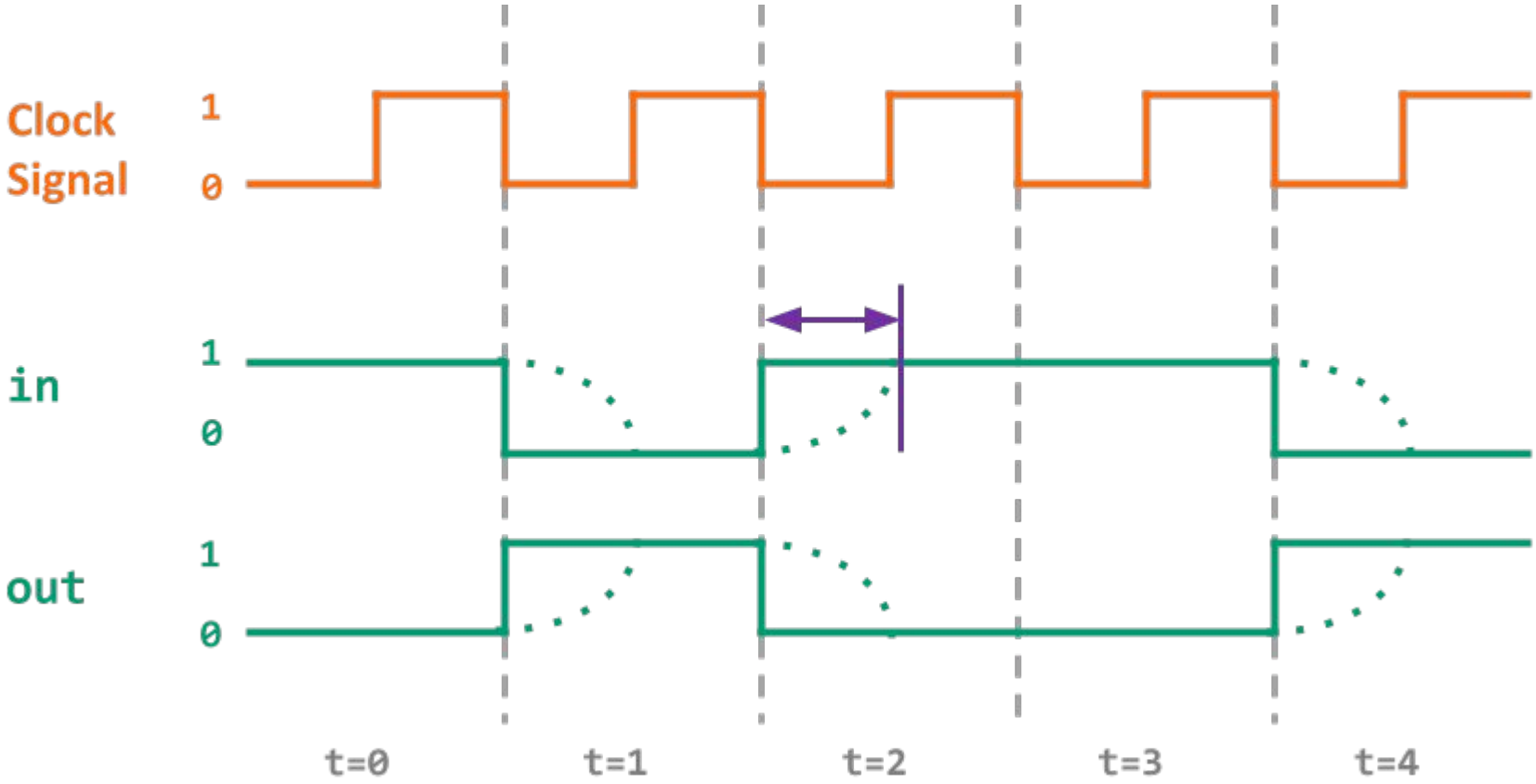
Adding a Clock: Clock Cycles

Choose a clock cycle length slightly longer than the delays

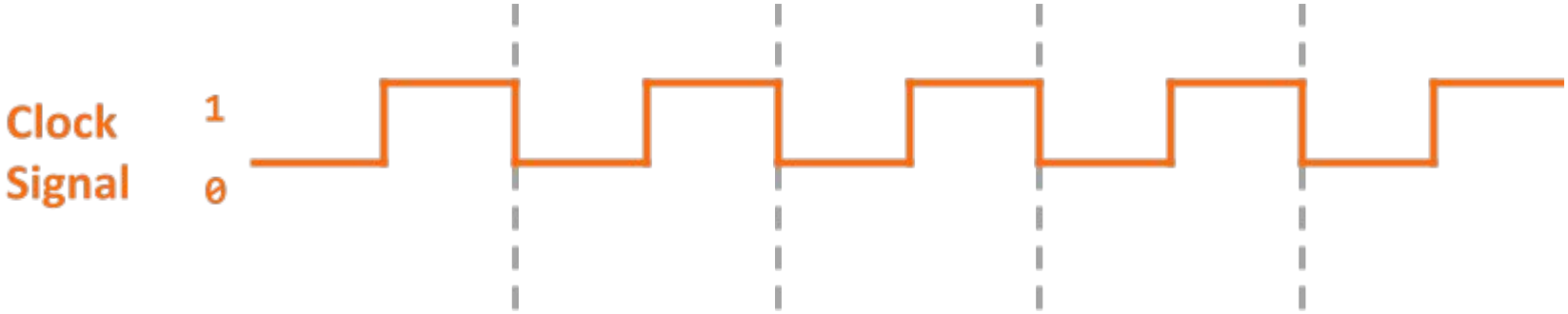


Adding a Clock: Abstraction

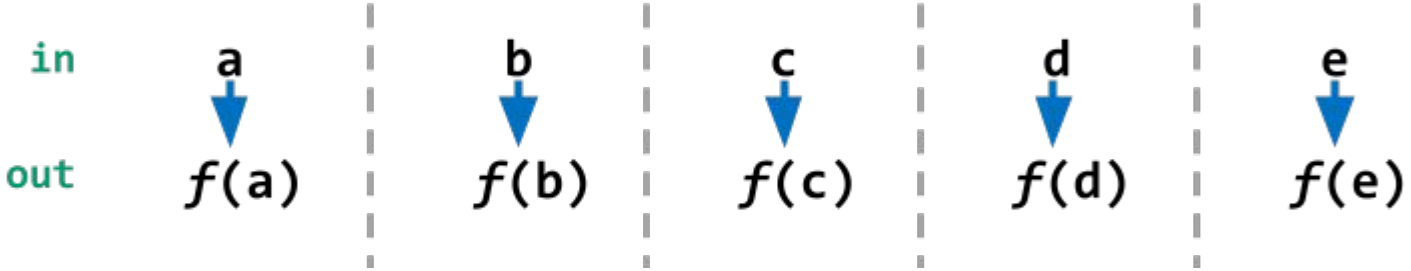
- If we use a long enough clock cycle, we can *pretend* that combinational chips (like Not) work instantly



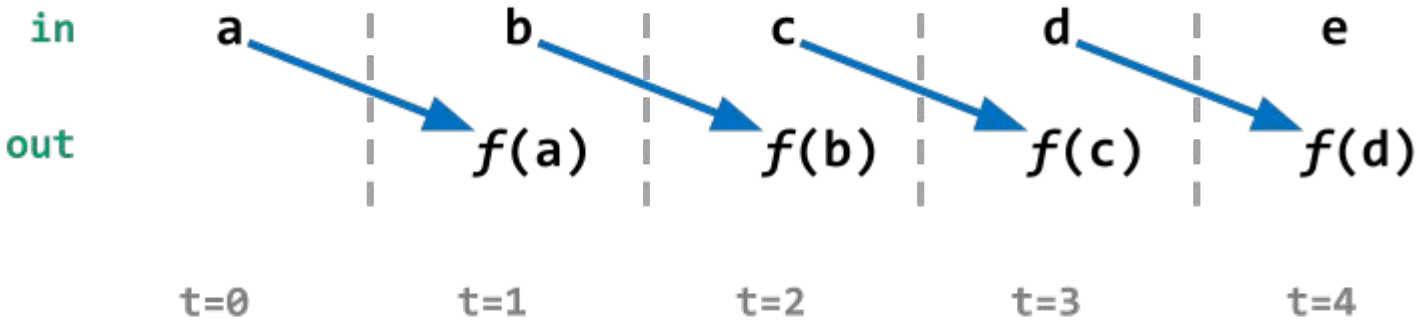
Combinational vs. Sequential Abstraction



Combinational: a function of the inputs from the current time cycle



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



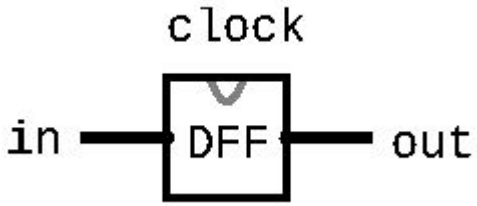
Agenda

- Project 1 Reflection
- Bloom's Taxonomy
- Cornell Note-Taking Method
- Representing Time in Hardware
- **Sequential Logic**

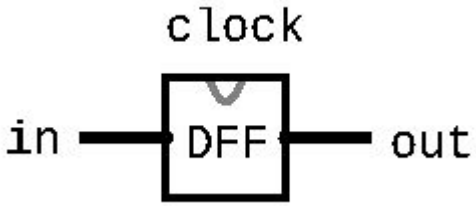
A New Primitive: Flip Flop

- 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"



Flip-Flop Time Series



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

Example:

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

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'll treat Flip-Flop as primitive in homeworks
  - Just like Nand, you can use the built-in implementation

Sequential Chips

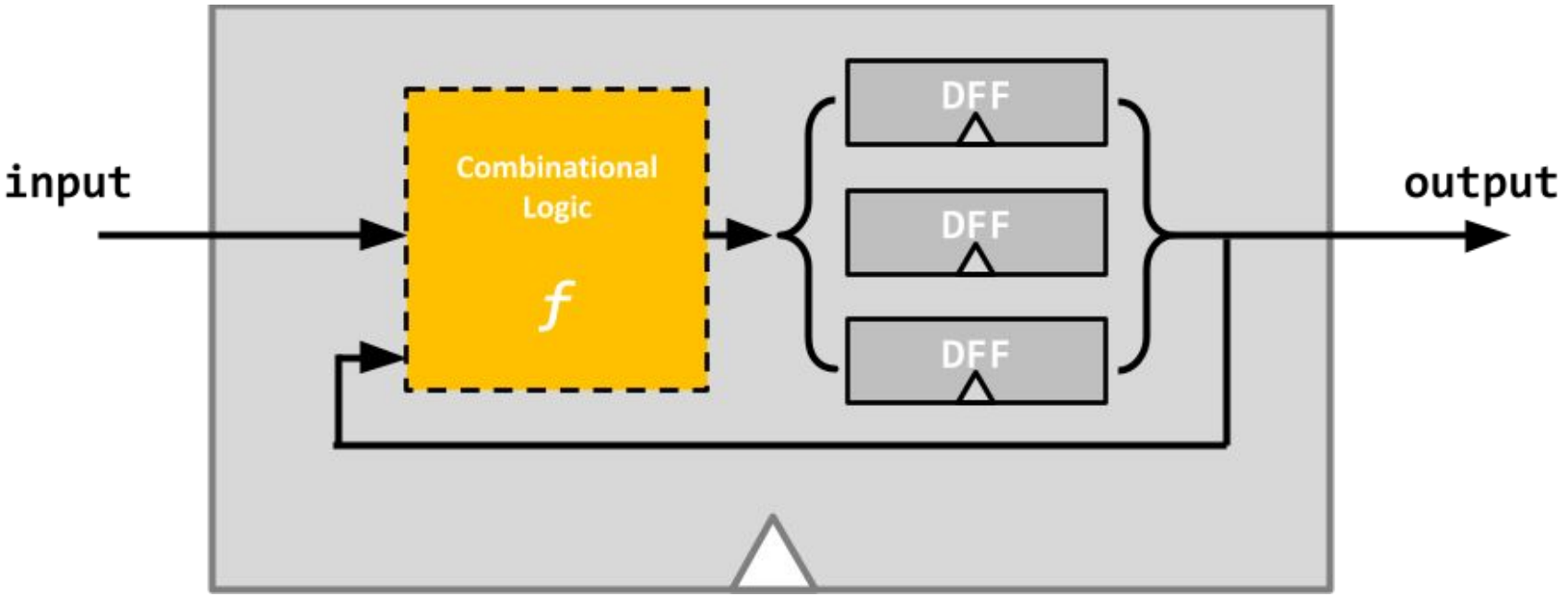
- A category of chips that utilize the clock signal, in addition to any combinational logic

- 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-1))
```



DFF Example 1 Specification

- Example specification:

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

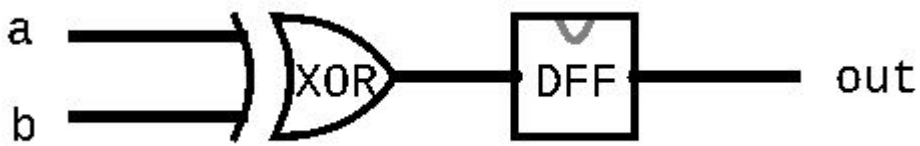
- Takes two inputs, a and b, and outputs the Xor of them
  - Note that out at time t is determined by a and b at time t-1
  - We will need to use a DFF!

DFF Example 1 Circuit Diagram

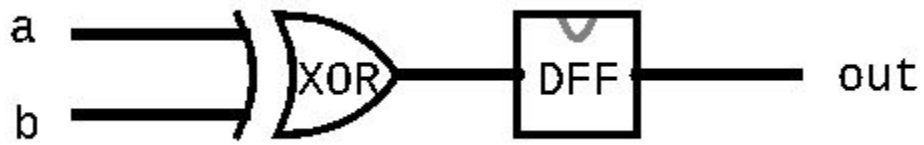
- Example specification:

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

- Circuit diagram:



DFF Example 1 Time Series

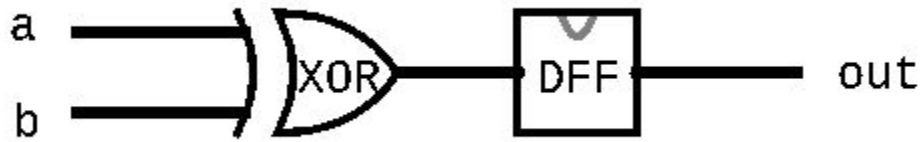


pin	t=0	t=1	t=2	t=3	t=4	t=5	t=6	...
a	0	0	1	1	1	0	0	...
b	0	1	0	1	1	1	0	...
out	0	0	1	1	0	0	1	...

Example:

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

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 2 Specification

- Example specification:

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

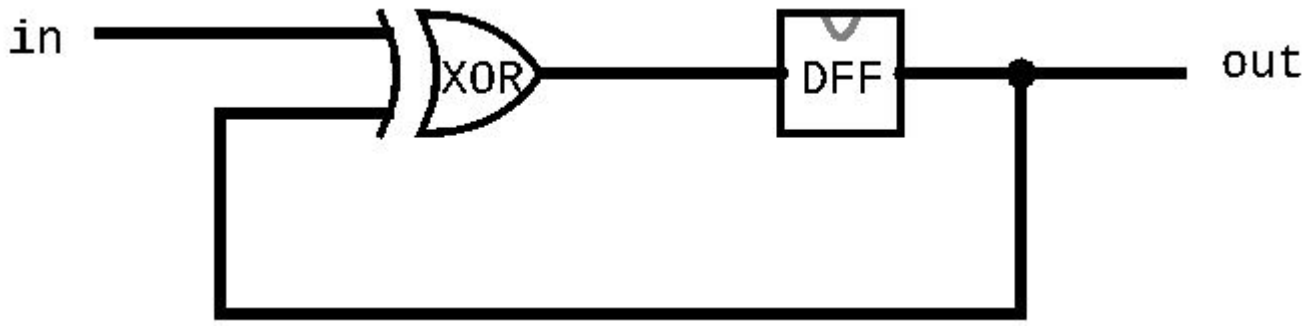
- Notice how the specification uses out(t-1) as an input for out(t)
  - Need some sort of circular wiring, separated by a DFF

DFF Example 2 Circuit Diagram

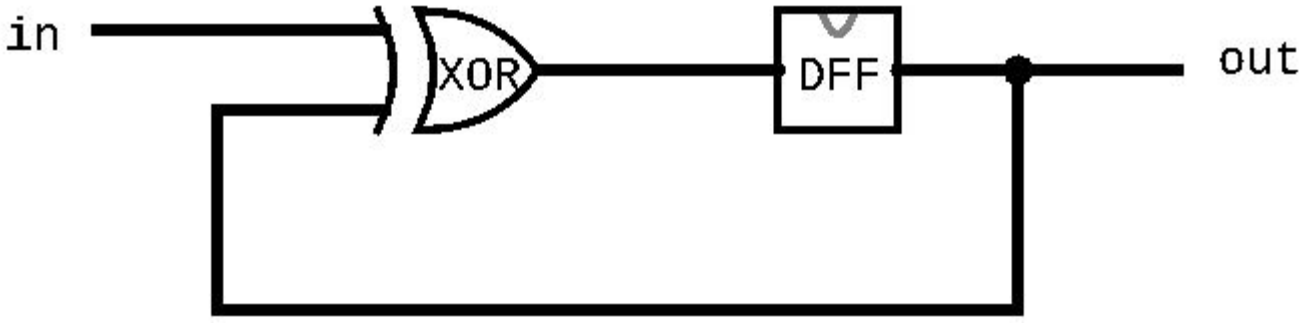
- Example specification:

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

- Circuit diagram:



DFF Example 2 Time Series

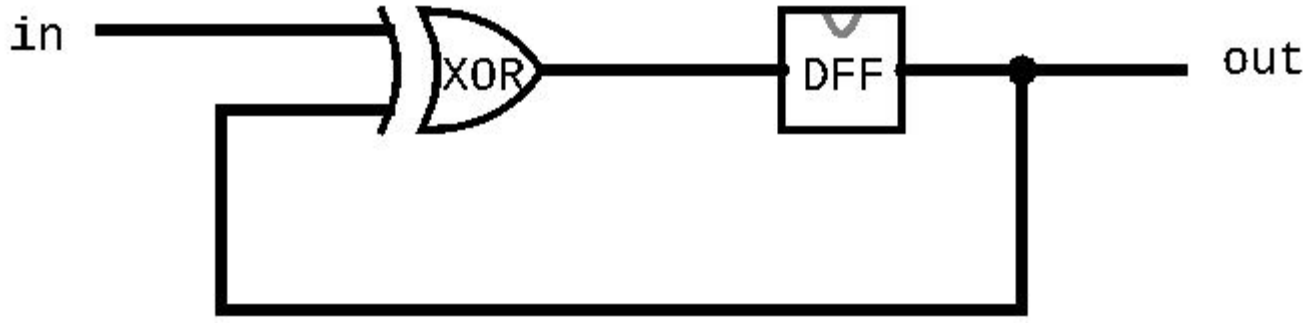


pin	t=0	t=1	t=2	t=3	t=4	t=5	t=6	...
in	0	0	1	1	1	0	0	...
out	0	0	0	1	0	1	1	...

Example:

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

DFF Example 2 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 3 Specification

- Example specification:

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

DFF Example 3 Time Series

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

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

Example:

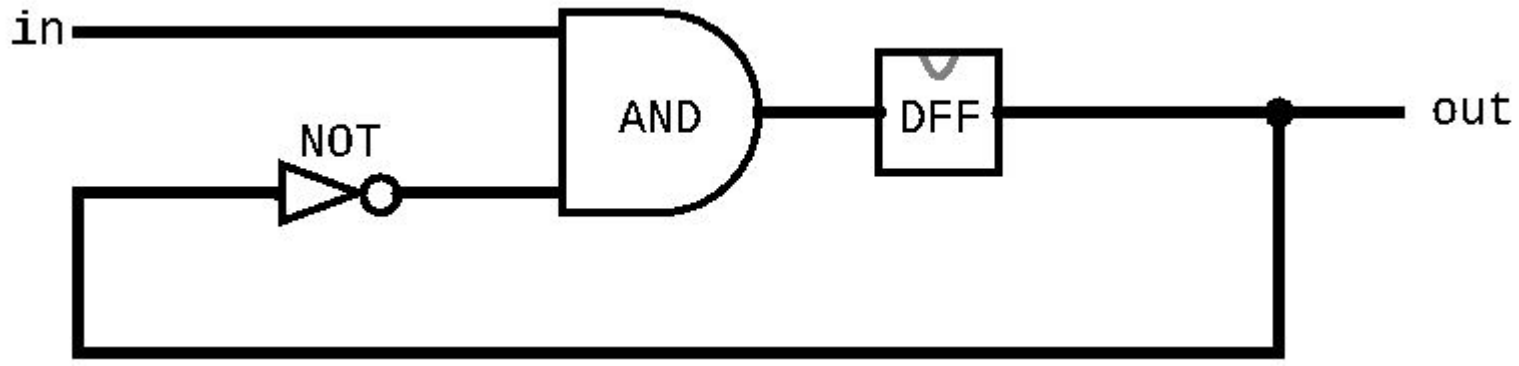
```
out(t=1) = And(Not(out(t=0)), in(t=0))
```

DFF Example 3 Circuit Diagram

- Example specification:

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

- Circuit diagram:



DFF Example 3 HDL Implementation

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

```
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);
}
```

Reminders

- Project 2 is due Thursday 11:59PM PDT
- Margot and Eric have office hours NOW(!) 2:00 - 3:00PM on Tuesdays!
- Message board! Discord!