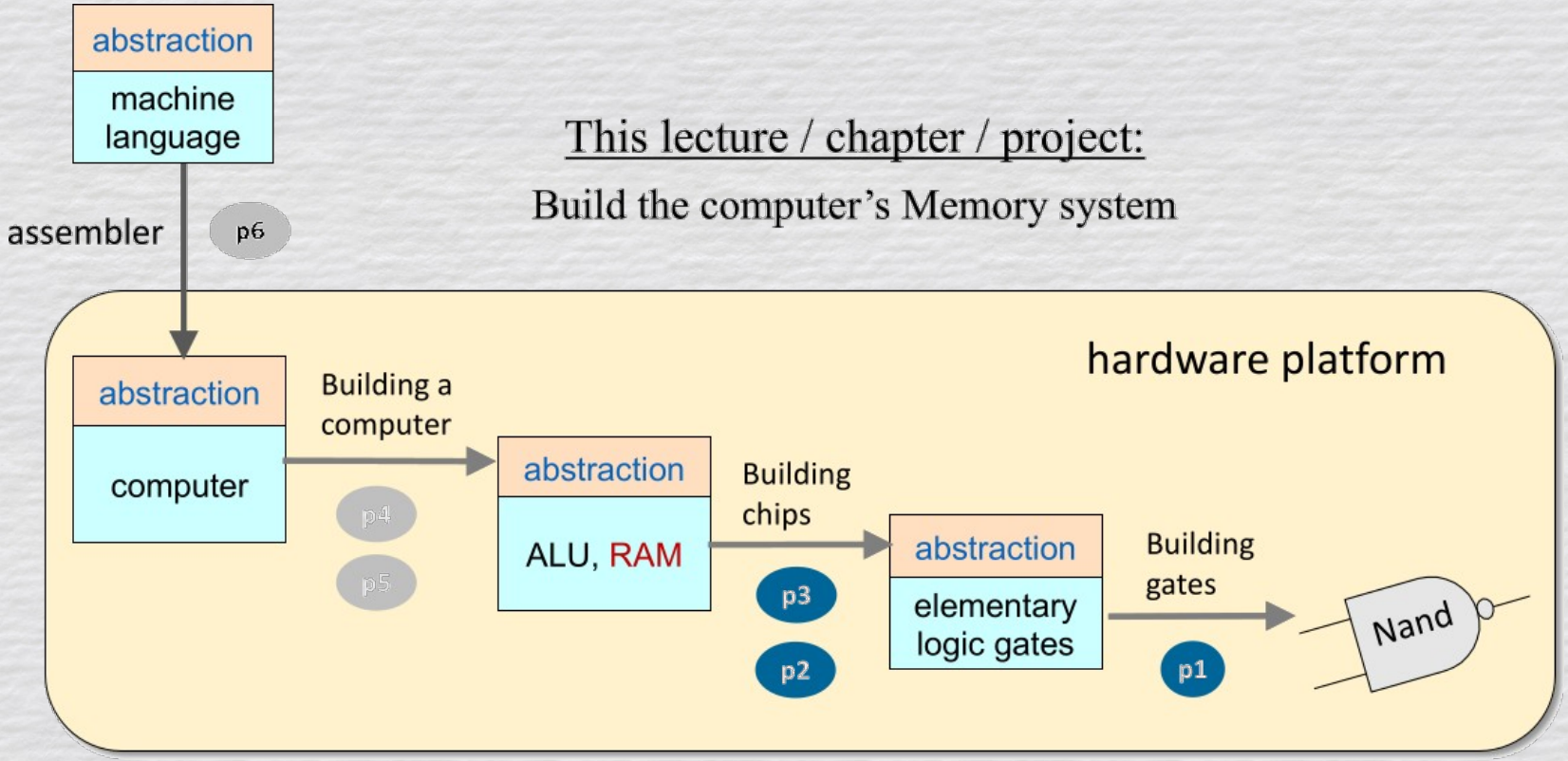
Computer Organization

Sequential Logic

Roadmap


abstraction
machine language
assembler
p6
This lecture / chapter / project:
Build the computer's Memory system
hardware platform
abstraction
computer
Building a computer
p4
p5
abstraction
ALU, RAM
Building chips
p3
p2
abstraction
elementary logic gates
Building gates
p1
Nand

Simple Model - Complex System

- A very common theme that occurs in computer science
- We first design a simple model
  - Logic Gates for this course
- We then proceed to add various layers of abstraction to build up to a more complex system

Combinational Logic

- During the course of Chapter 1, we developed several logic gates including some fairly complex multiplexors
- During Chapter 2, we extended this even further by designing a fairly robust ALU
  - A very simple ALU, but quite elegant in design thanks to the abstractions of Two's Complement and Boolean Logic

Combinational Logic – cont.

- While the current setup we have is quite clever, it is still fairly limited
- We can compute various operations on a fixed number of inputs
  - And16, Dmux8Way, Add16, ALU, etc.
- We cannot due this for a dynamic number of inputs though...

Sequential Logic

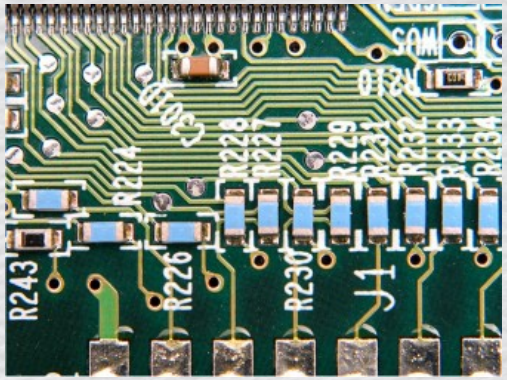
- The answer to our issue is time
- Our current gates are completely time-independent in their current state, but we can solve that via sequential logic
- This allows us to pass the output of our gates back to their inputs giving us a way to track changes over time

Sequential Logic – Software

- Software
  - The ability to store data
    - x = 17
    - int i = 0
  - The ability to solve things sequentially
    - Loops
    - Recursion

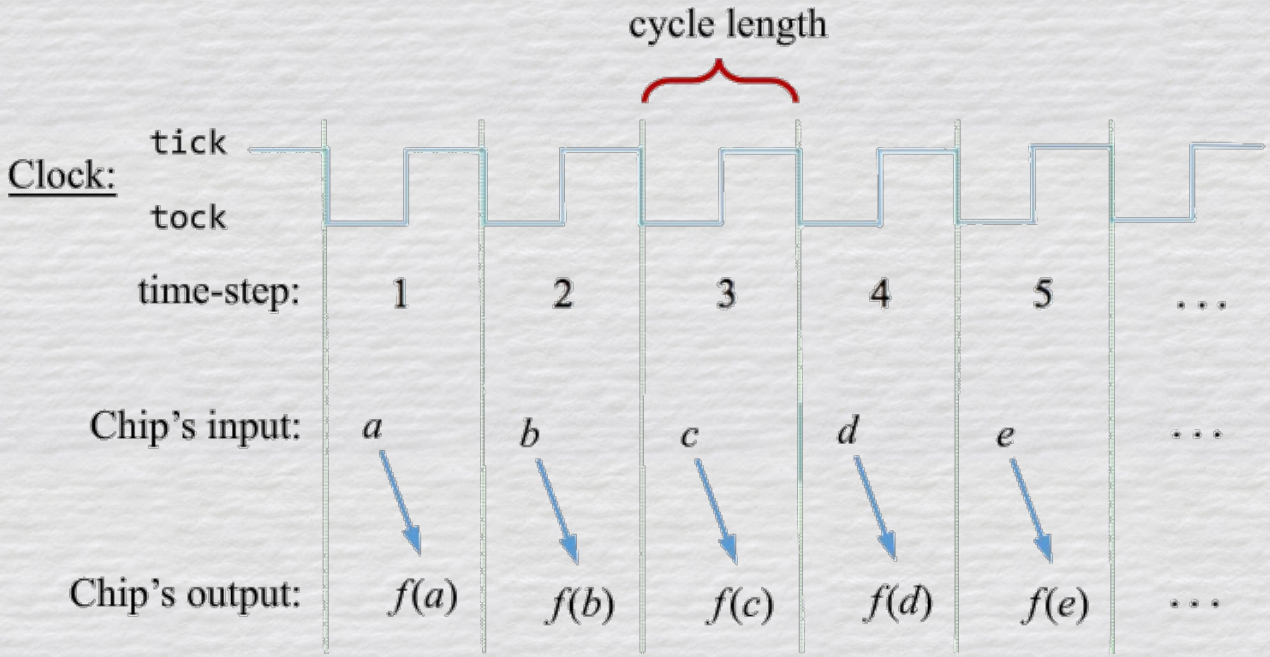
Sequential Logic – Hardware

- Hardware
  - Must be able to handle physical hardware delays when moving data from one chip to another

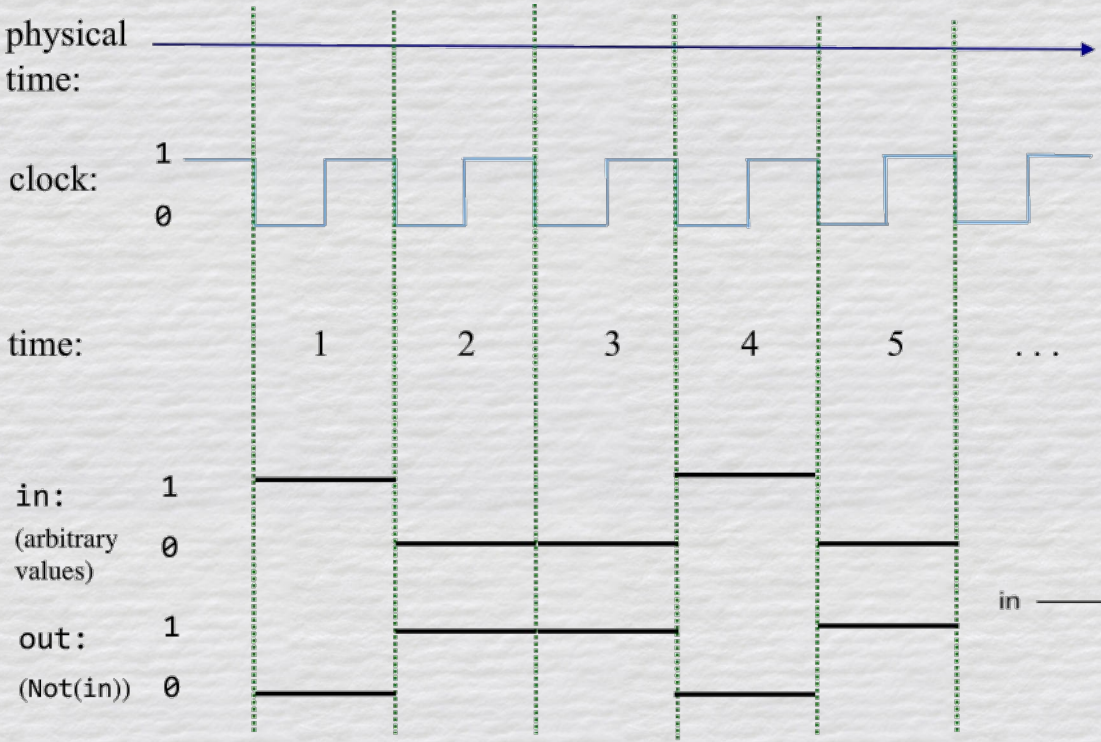


Discrete Time

- Set time cycle slightly greater that maximum delay
- Use outputs only at end of cycle
  - Ignore changes between



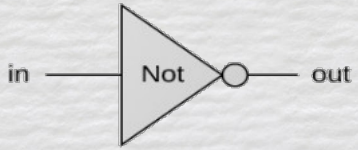
Discrete Time – Ideal



Arrow of time:

Continuous

Discrete time:
Design decision:
Track state changes only when advancing from one time-step to another

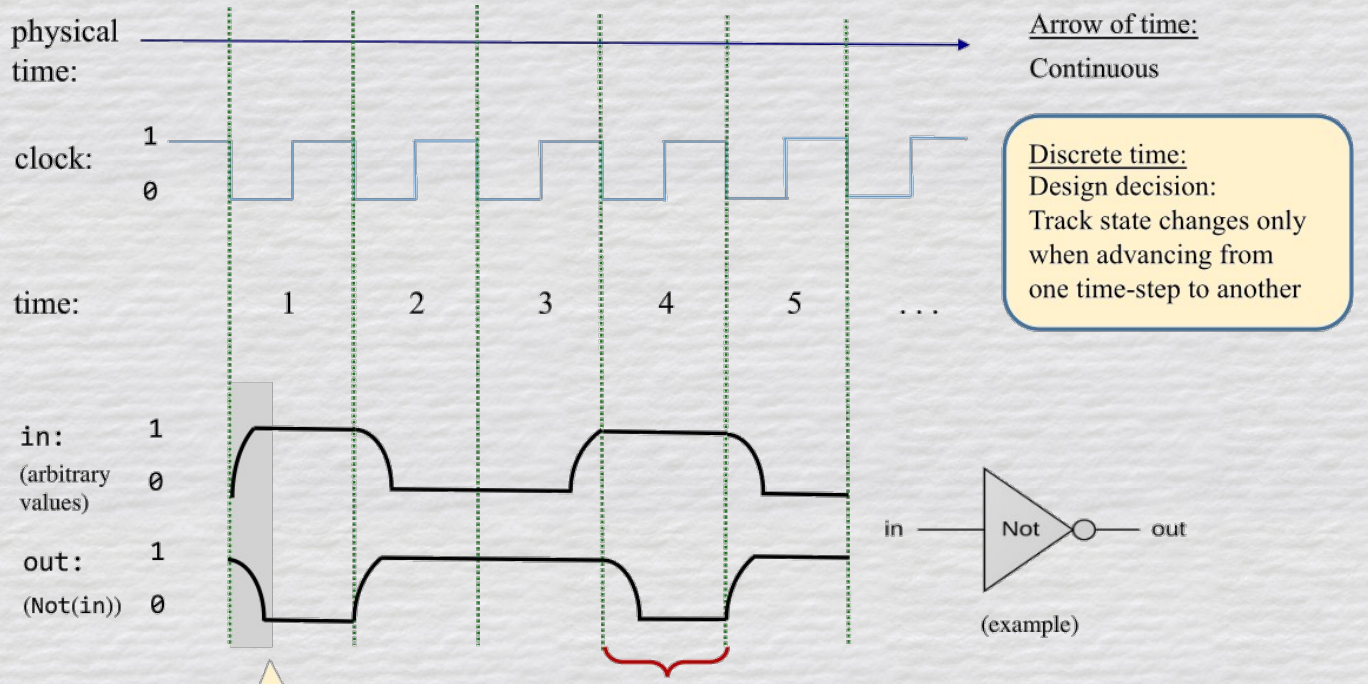


(example)

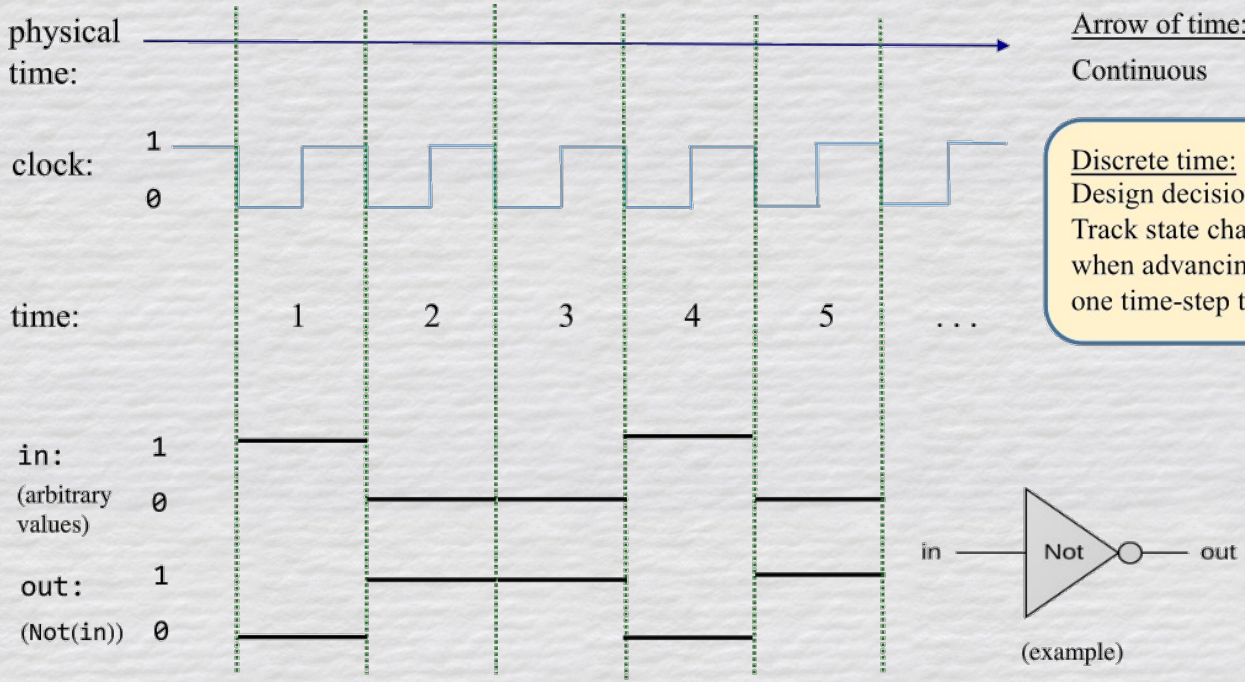
Desired / idealized behavior of the `in` and `out` signals:

That's how we *want* the hardware to behave

Discrete Time – Actual



Discrete Time – Results



Arrow of time:

Continuous

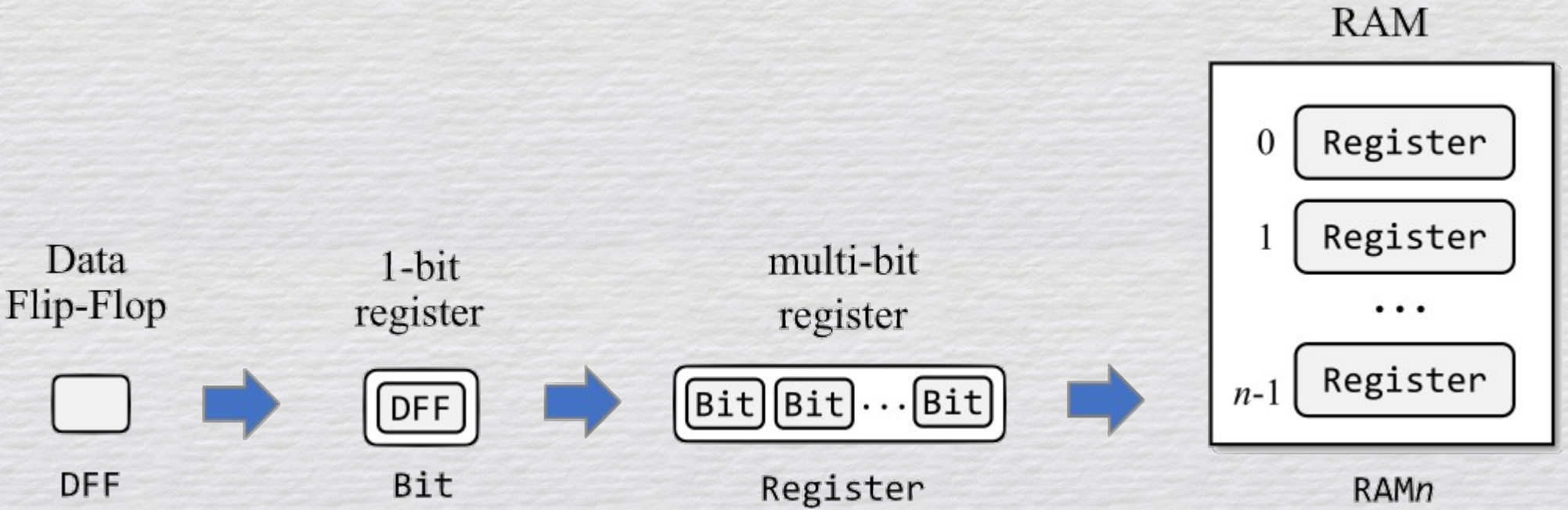
Discrete time:
Design decision:
Track state changes only when advancing from one time-step to another

(example)

Resulting effect:

- Combinational chips react "immediately" to their inputs
- Facilitated by the decision to track changes only at cycle ends

Memory Hierarchy

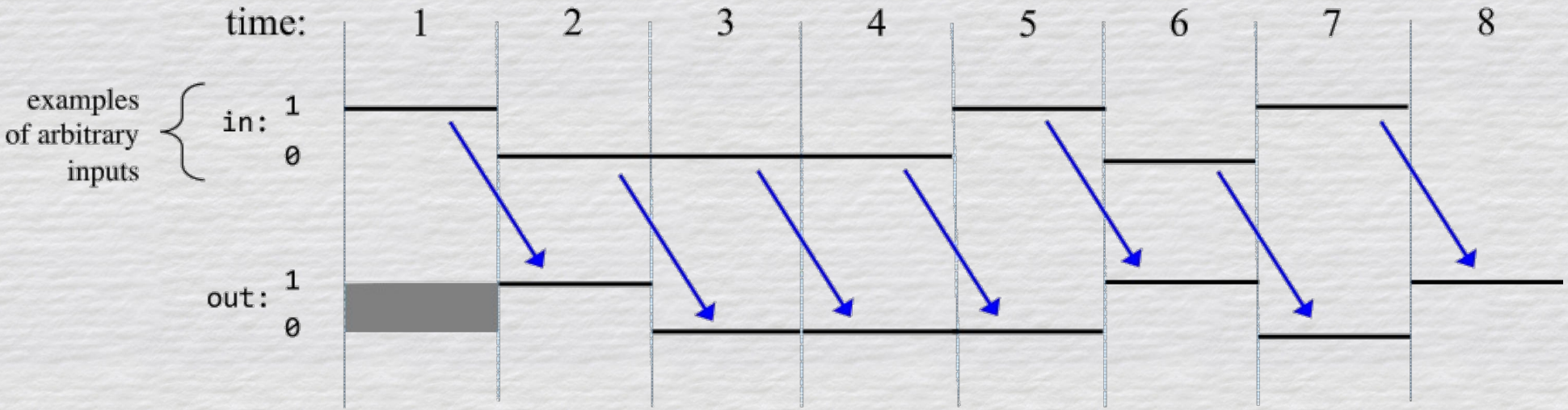


Data Flip Flop

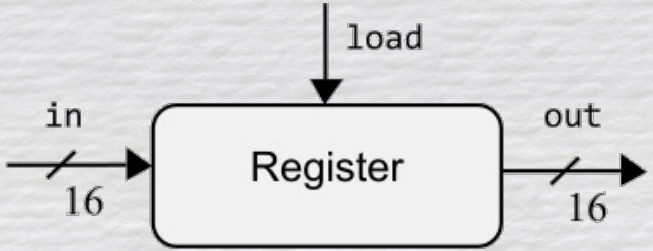
Data Flip Flop (aka *latch*)

The most elementary sequential gate: Outputs the input in the previous time-step

$\texttt{out}(t) = \texttt{in}(t-1)$



Register



Basic abstractions:

- "Loading" a value
- "Storing" a value

1 2 3 4 5 6 7 ... 205 206 207 208 209 210 211 ... time

(cycles)

x = 17 , 17, 17, 17, 17, 17, 17, ... ,

loading | storing

x = 21 , 21, 21, 21, 21, 21, 21, ...

loading | storing