ENGN2219/COMP6719
Computer Systems & Organization

Convener: Shoaib Akram

shoaib.akram@anu.edu.au

Australian National University

Plan: Week 4

Last week: Basics of sequantial circuits

This Week: Finite state machines

This Week: Timing, parallelism & pipelining

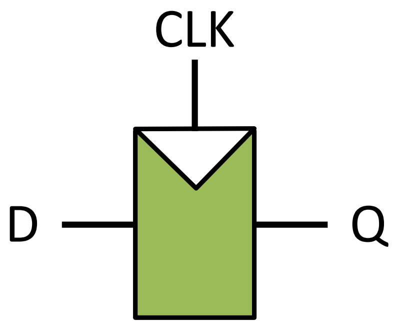
This Week: Gentle introduction to architecture

Two Sync. Sequential Circuits

- Two widely used synchronous sequential circuits
  - Finite state machine (FSM) ✓
  - Pipelines
- To understand pipelining, we need to first understand the timing specification of synchronous sequential circuits

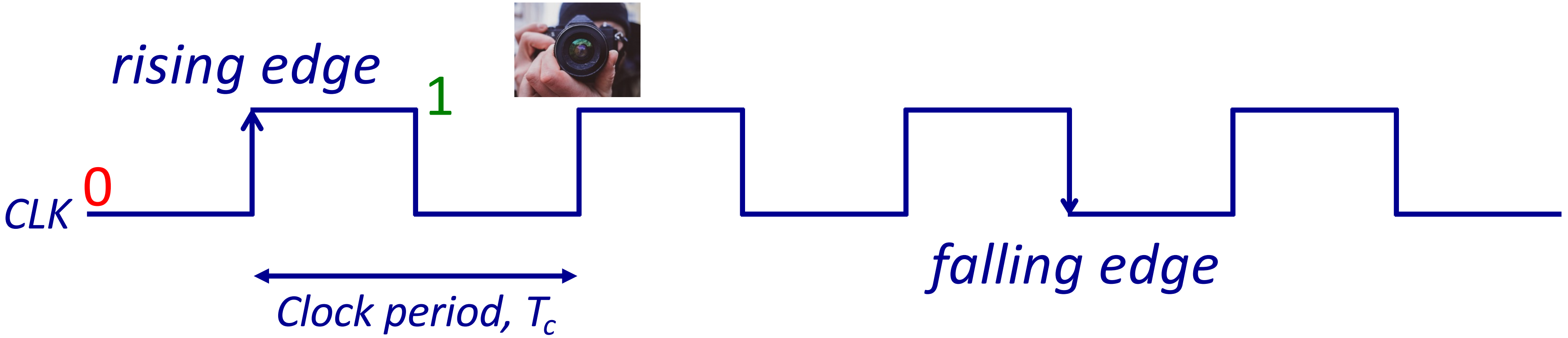
Timing in Sequential Circuits

- We need to understand three aspects of timing specification
  - Clock-to-Q propagation delay
  - Setup time
  - Hold time



Recall the Clock Waveform

- Output does not change instantly when the rising edge arrives
- Input need to stay stable for some time period for the flipflop to take a reliable photograph

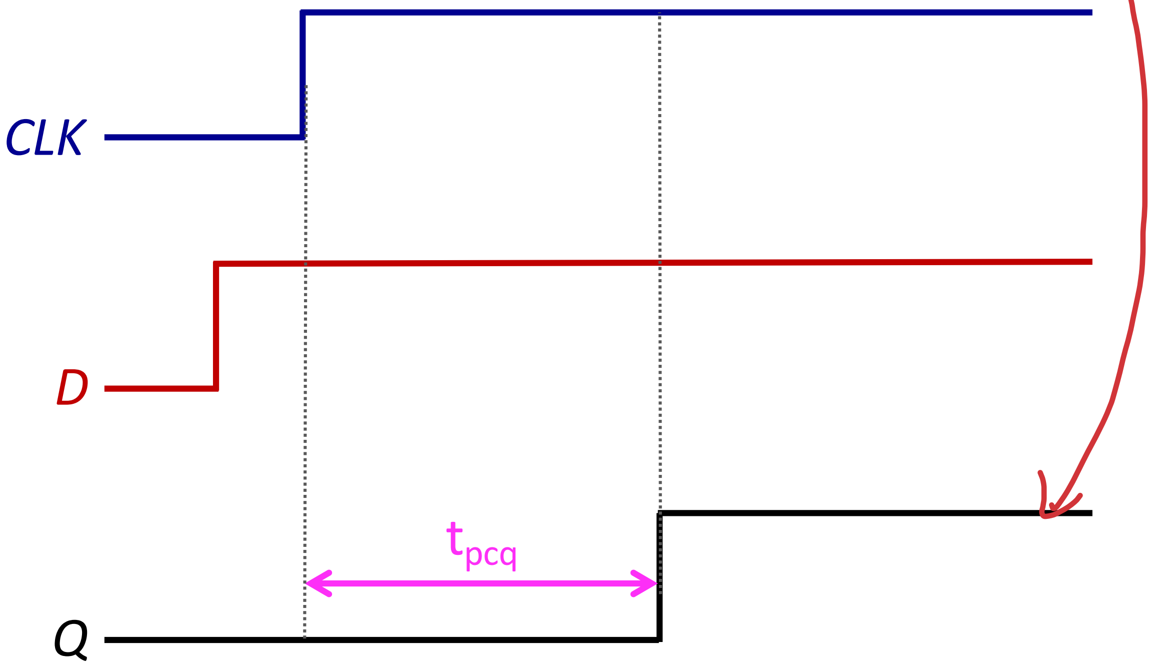


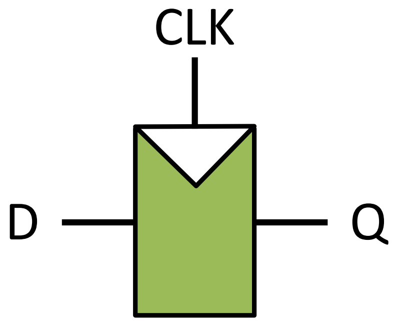
Frequency = $1/T_c$

Clock-to-Q Propagation Delay

When the clock rises, the time it takes for the output to settle to the final value (t_{pcq})

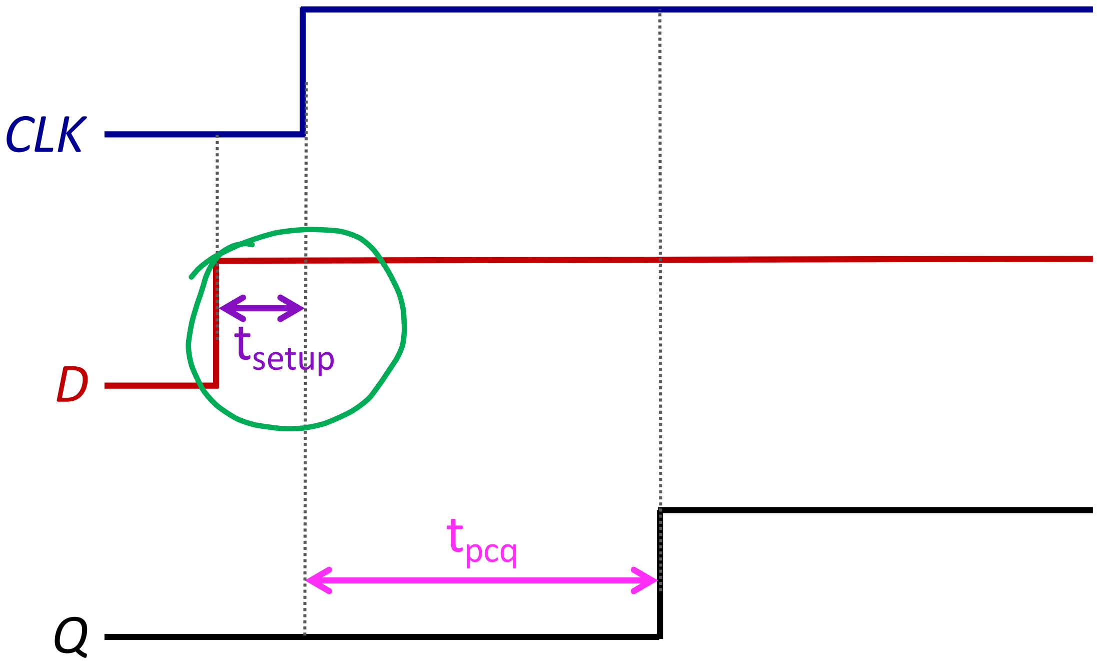
Click

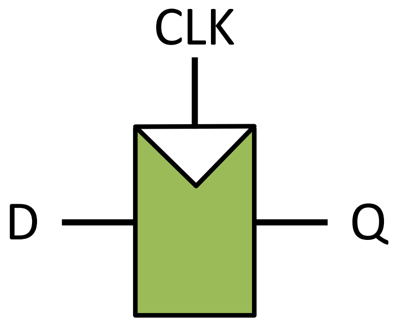




Setup Time

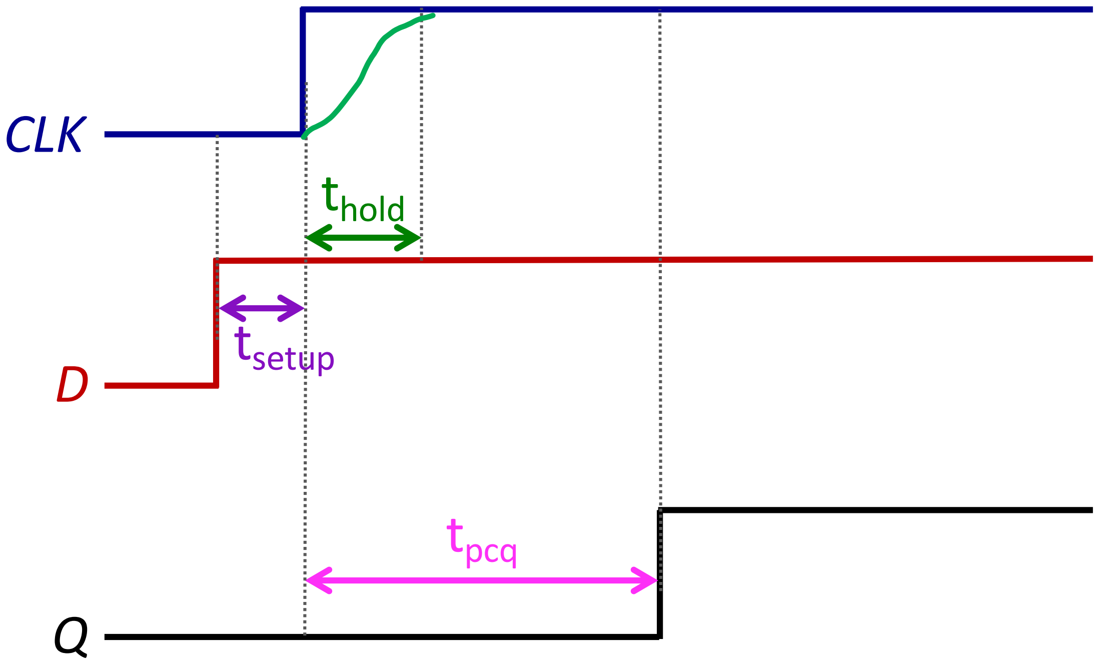
For the circuit to sample its input correctly, the input must have stabilized at least some setup time, t_{setup}, before the rising edge of the clock

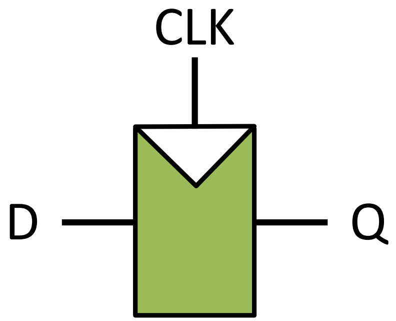




Hold Time

The input must remain stable for at least some hold time (t_{hold}) after the rising edge of the clock





Aperture Time

The sum of the setup and hold times is called the aperture time of the circuit

- Total time for which the input must remain stable

Remark: Hold Time

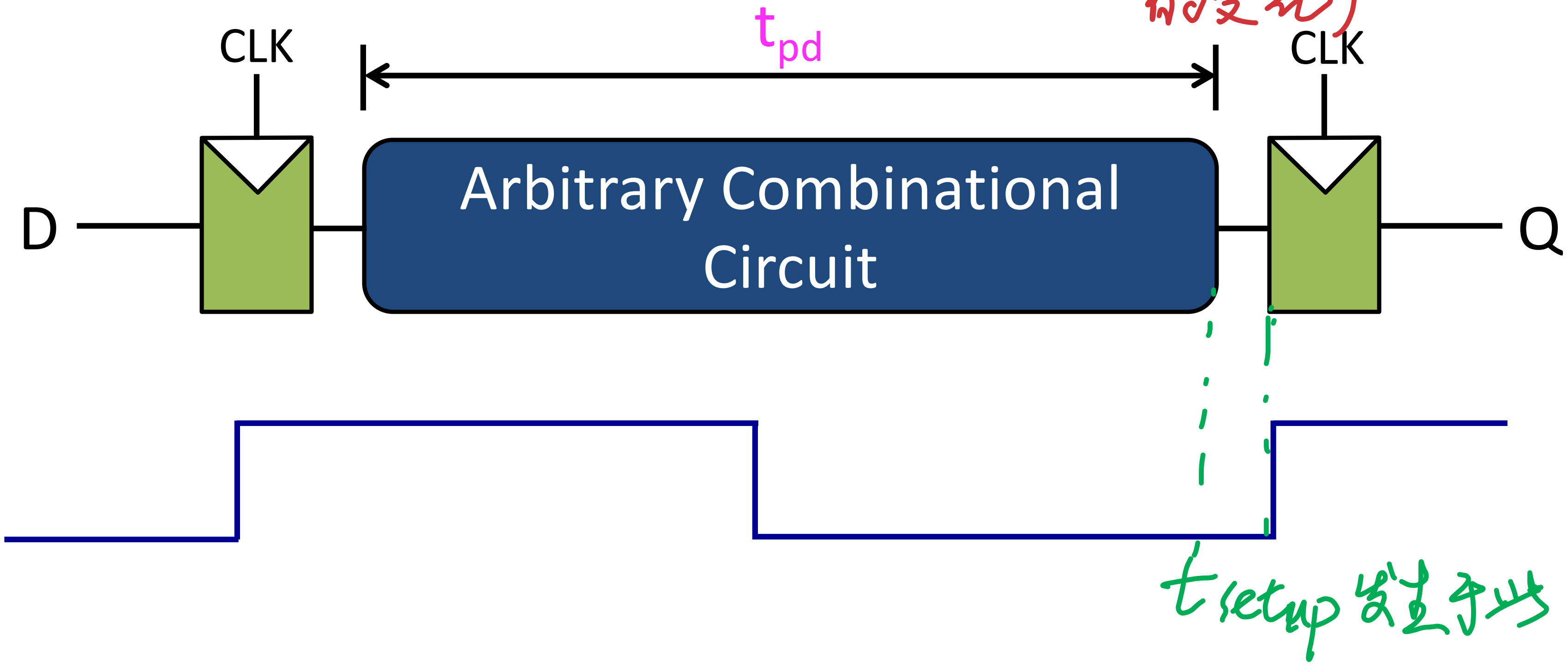
It is a reasonable assumption that modern flipflops have a hold time close to zero (*we can ignore hold time in subsequent discussions*)

Quiz

What is the clock period for the circuit below for it to work correctly?

1. t_{pd}
2. $t_{pd} + t_{pcq}$
3. [✓ 3.] $t_{pd} + t_{pcq} + t_{setup}$

为t_{pd}时段之后的t_{setup}.(计算后结果的变化)



Sequencing Overhead

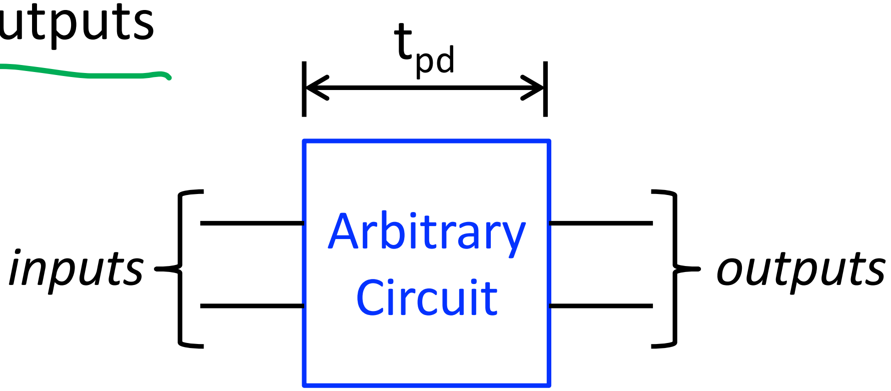
$t_{pcq} + t_{setup}$ is called the sequencing overhead of the flipflop

$T_c = t_{pd} + t_{pcq} + t_{setup}$

- Ideally the entire clock period should be spent on doing useful work in the combinational circuit
- The sequencing overhead of the flipflop cuts into this time

Speed of a Circuit

At a high level, an arbitrary digital circuit processes a group of inputs and produces a group of outputs



We needs metrics to quantify the speed with which we can process inputs to produce outputs (i.e., the performance of a circuit)

- **Latency:** The time required to produce one group of outputs once the inputs arrive (propagation delay, end-to-end latency)
- **Throughput:** The number of input groups processed per unit of time

Example: Latency/Throughput

- What is the latency and throughput for a tray of cookies?
  - Step # 1: Roll cookies (5 minutes)
  - Step # 2: Bake in the oven (15 minutes)
  - Once cookies are baked, start another tray

- Latency (hours/tray): 20 min
- Throughput (trays/hour): 3

Parallelism

Many scenarios in the real-world requires us to increase the throughput of the digital system

- # add operations per second (ALU)
- # instructions per second (CPU)

Parallelism is the key technique digital systems use to increase throughput

- Process several inputs at the same time
- Ideas?

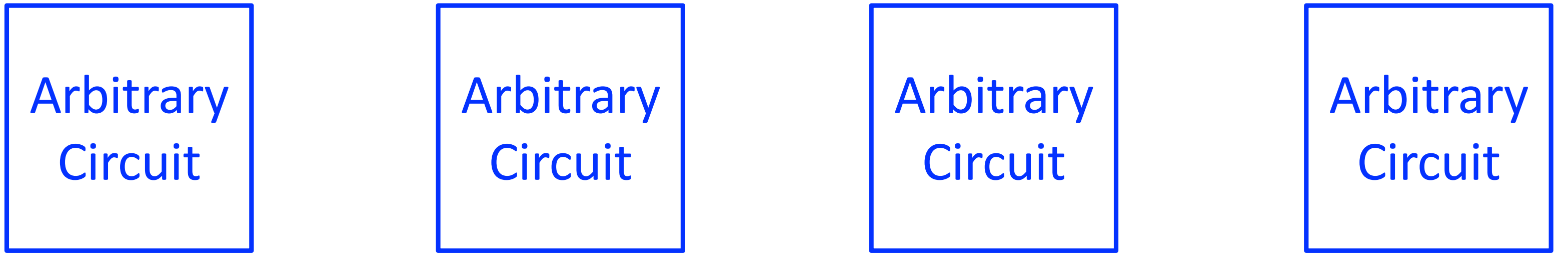
Defining a Task

Task: The process of producing a group of outputs from a group of inputs can be considered a task

- A circuit may need to perform several tasks
- A task can be as simple as adding two numbers
- More complex task is computing a Fourier transform

Spatial Parallelism

Spatial Parallelism: Use multiple copies of hardware (circuit) to get multiple tasks done at the same time



- Suppose a task has a latency of L seconds
- **No spatial parallelism:** Throughput is 1/L (one task per L seconds)
- **N copies of hardware:** Throughput is N/L (N tasks per L seconds)
- Gain in throughput (speedup) = N

Note on Latency

Spatial Parallelism does not improve (reduce) the latency of the circuit. We can finish more tasks per unit of time. But each task still takes L seconds

Temporal Parallelism

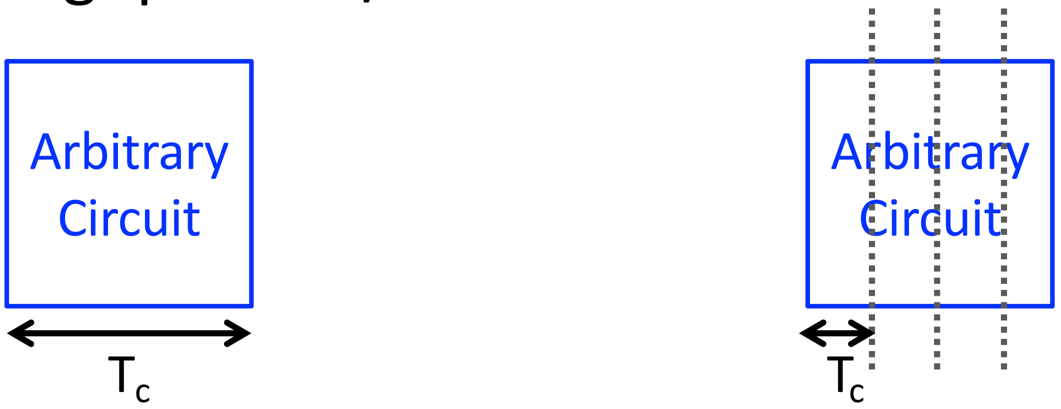
Temporal Parallelism (pipelining): Break down a circuit into stages. Each task passes through all stages. Multiple tasks are spread through stages.

Analogy: *Automotive pipeline*

Work on multiple cars in parallel. Each car goes through all stages. Each stage requires different work. All stages should take roughly the same time for this to work

Pipelining

If a task is broken into N stages, and all stages are of equal length, then the throughput is N/L



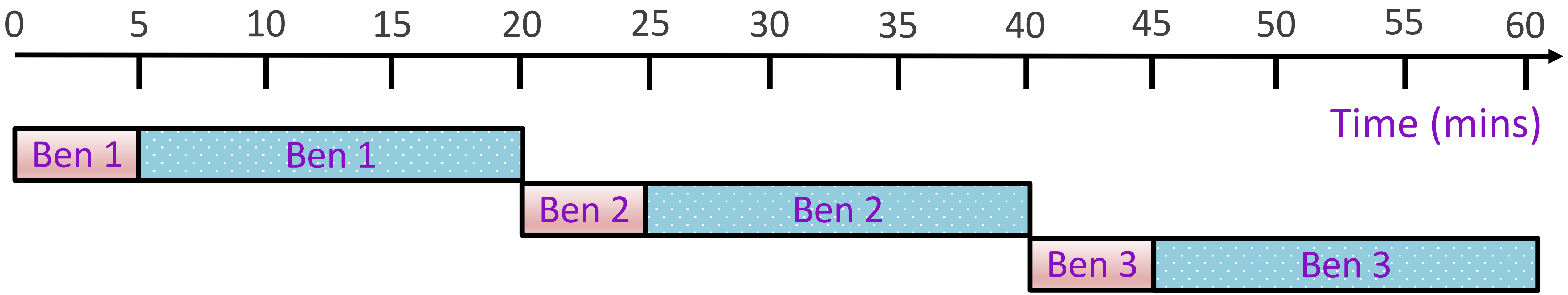
- *The challenge of pipelining is to find stages of equal length*
- *Let’s go back to baking cookies*

Cookie Parallelism

Ben and Jon are making cookies. Let's study the latency and throughput of rolling/baking many cookie trays with

- No parallelism
- Spatial parallelism
- Pipelining
- Spatial parallelism + pipelining

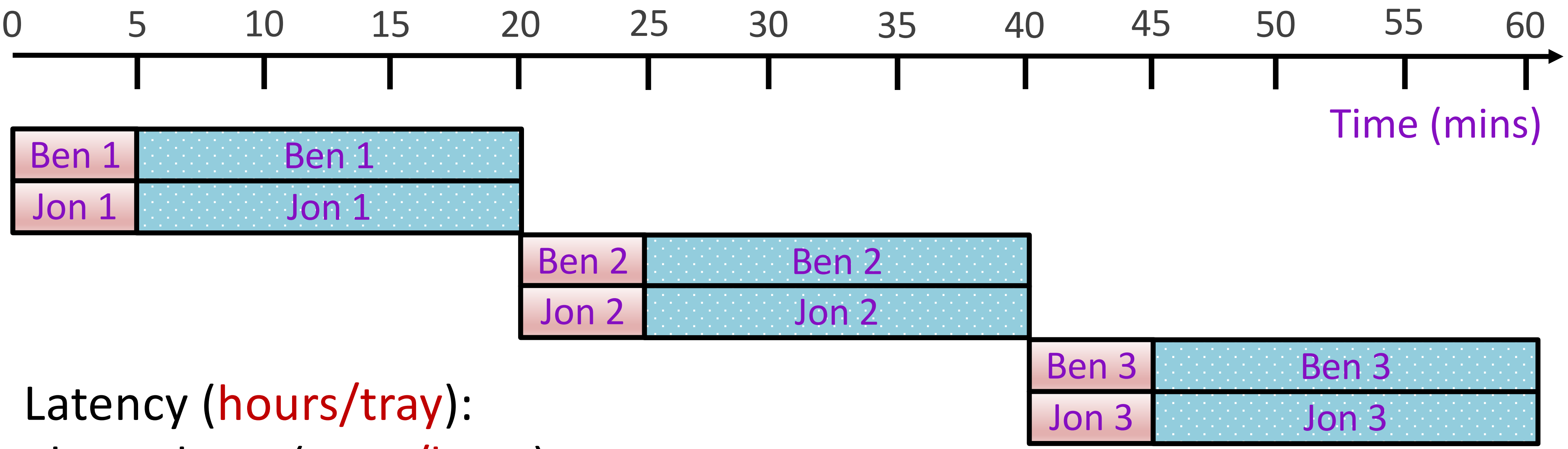
No Parallelism (Ben Only)



Latency (hours/tray):

Throughput (trays/hour):

Spatial Parallelism (Ben & Jon)

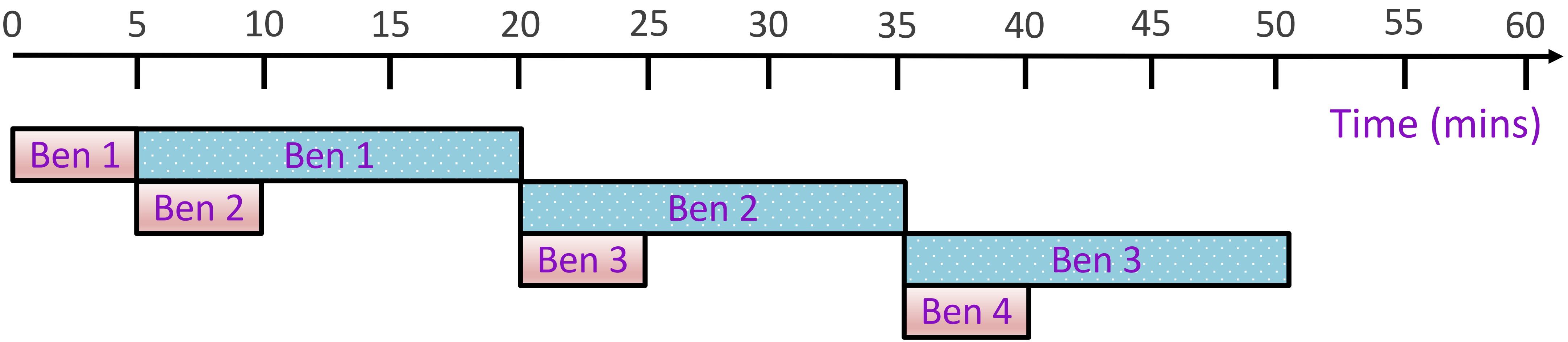


Latency (hours/tray):

Throughput (trays/hour):

Note: Jon owns a tray and oven (hardware duplication)

Pipelining (Ben Only)

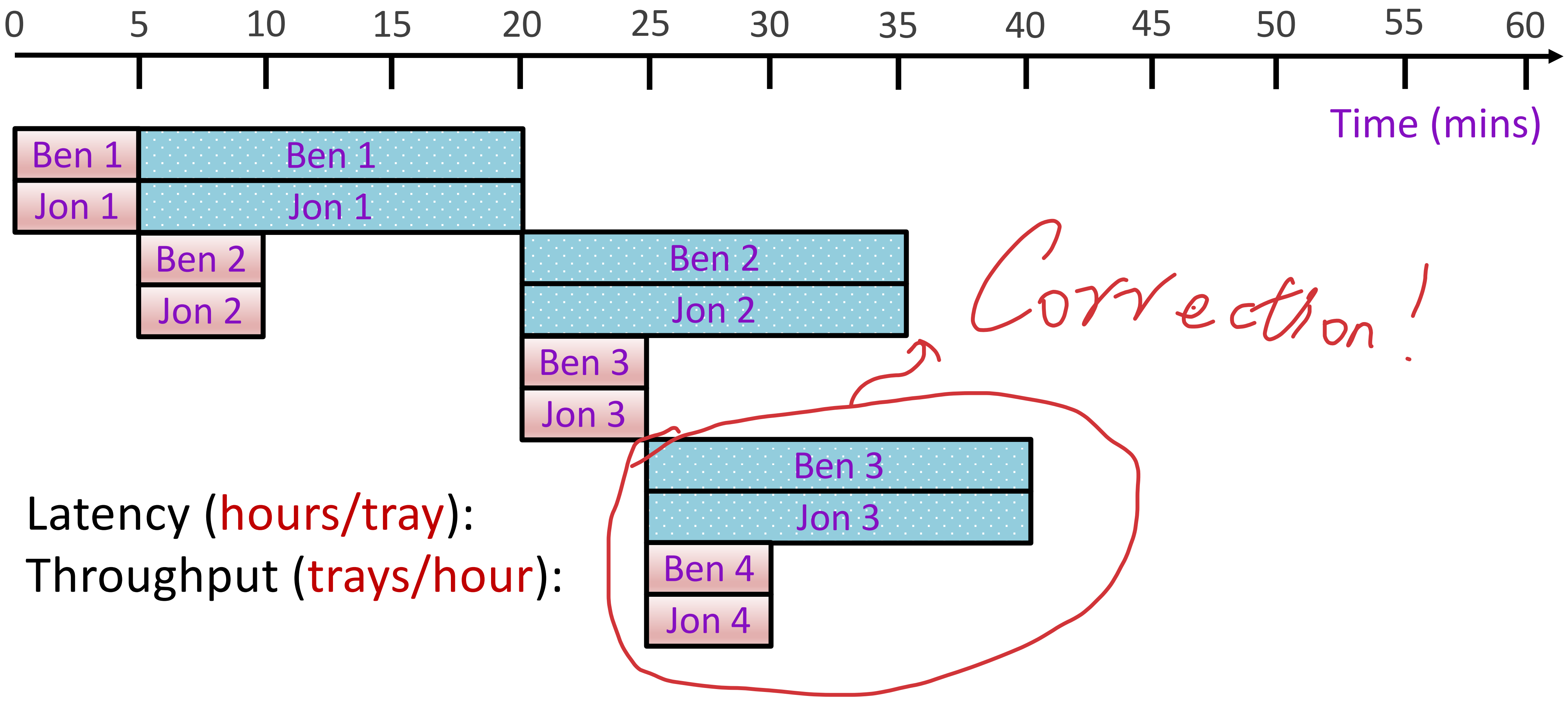


Latency (hours/tray): 20 min

Throughput (trays/hour): 4

Note: Ben decides not to waste a separate tray and oven

Spatial + Temporal Parallelism



Latency (hours/tray):

Throughput (trays/hour):

Answers Explained

- **No parallelism**
  - Latency is clearly 20 minutes (1/3 hours/tray)
  - Throughput is 3 trays per hour
- **Spatial parallelism**
  - Latency remains unchanged as it still takes 20 mins to finish a tray
  - Throughput is doubled via duplication: 6 trays per hour
- **Pipelining**
  - Latency for a single tray remains unchanged
  - Throughput: Ben puts a new tray in the oven every 15 minutes, so the throughput is 4 trays per hour
  - Note that in the first hour, Ben loses 5 minutes to fill the pipeline
- **Spatial parallelism + pipelining**
  - Latency remains unchanged
  - Throughput: Ben & Jon combo puts two trays in the oven every 15 minutes, so the throughput is 8 trays per hour

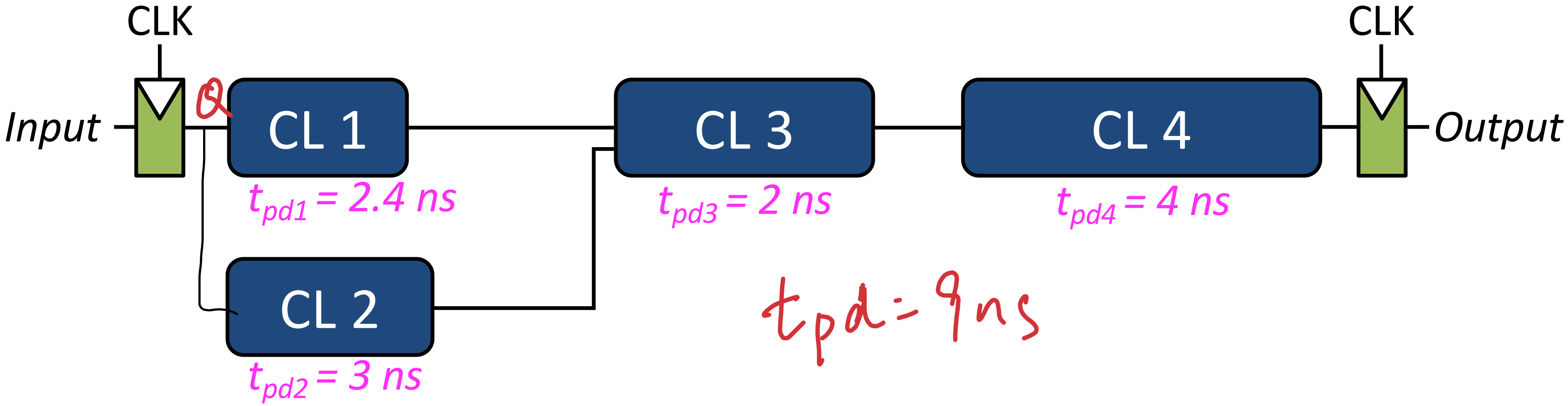
Pipelining Circuits

- Divide a large combinational block/circuit into shorter stages
- Insert registers between the stages
  - The outputs from one stage are copied into a register and communicated to the next stage
- Run the pipelined circuit at a higher clock frequency
  - Each clock cycles, data flows through the pipeline from left to the right
  - Multiple tasks can be spread across the pipeline

Exercise: Now with a circuit

Clock-to-Q propagation delay: 0.3 ns

Setup time : 0.2 ns



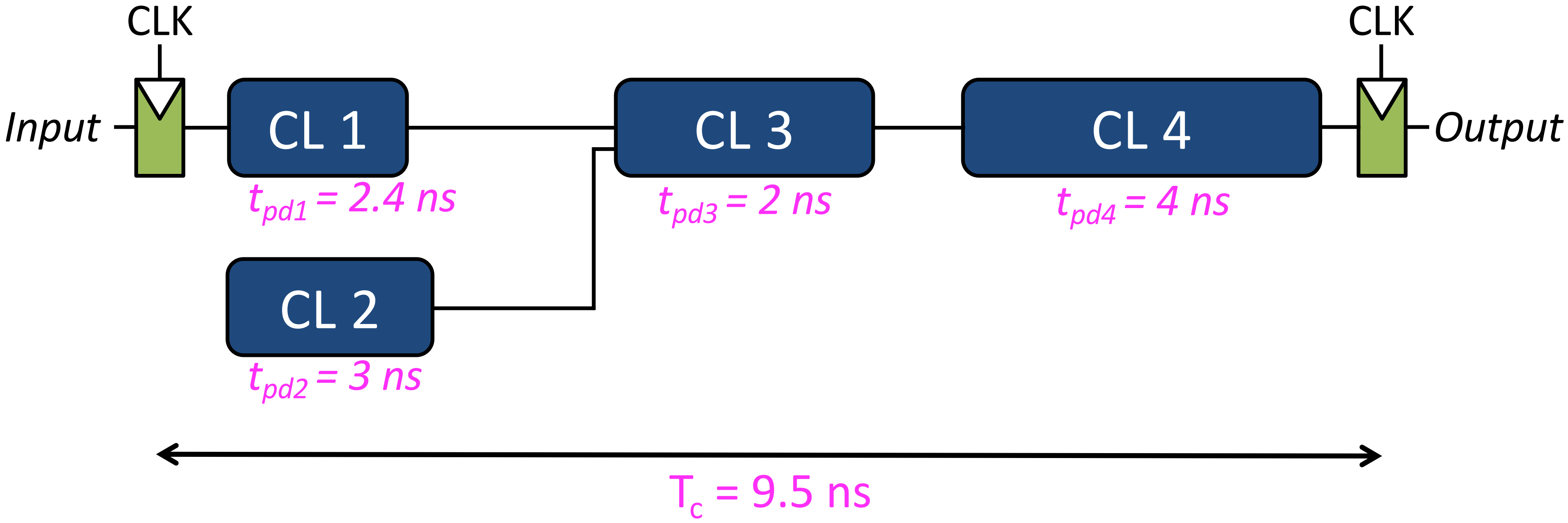
$tpd = 9ns$

$T_c = tpd + setup + tpcq = 9.5ns$

Exercise: Now with a circuit

Clock-to-Q propagation delay: 0.3 ns

Setup time : 0.2 ns

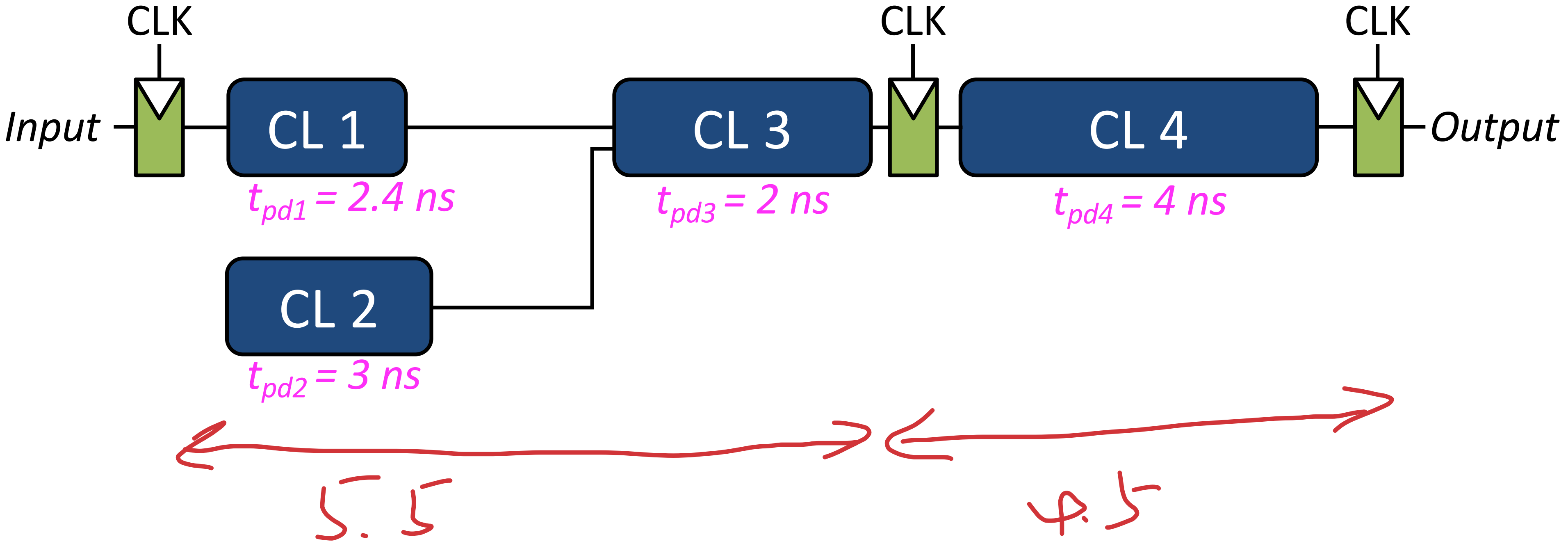


Latency = 9.5 ns

Frequency = 1/9.5 ns = 105 MHz

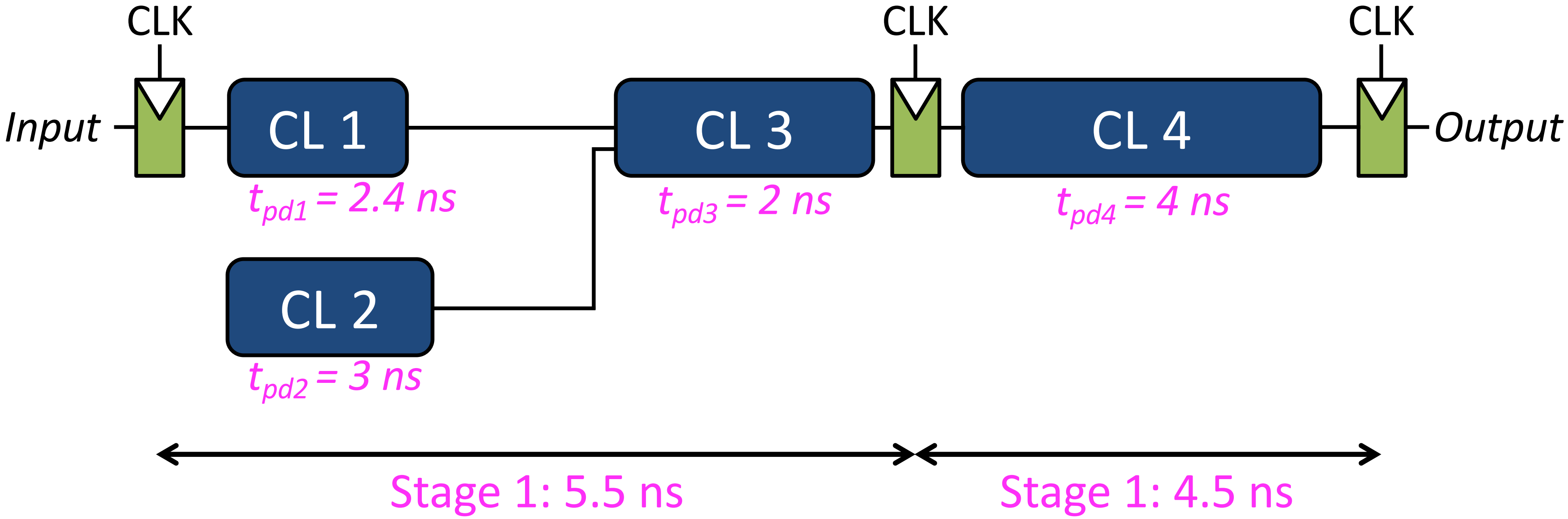
Exercise: 2-stage pipeline

Each task takes *two* clock cycles, but cycle time is reduced



Exercise: 2-stage pipeline

Each task takes *two* clock cycles, but cycle time is reduced

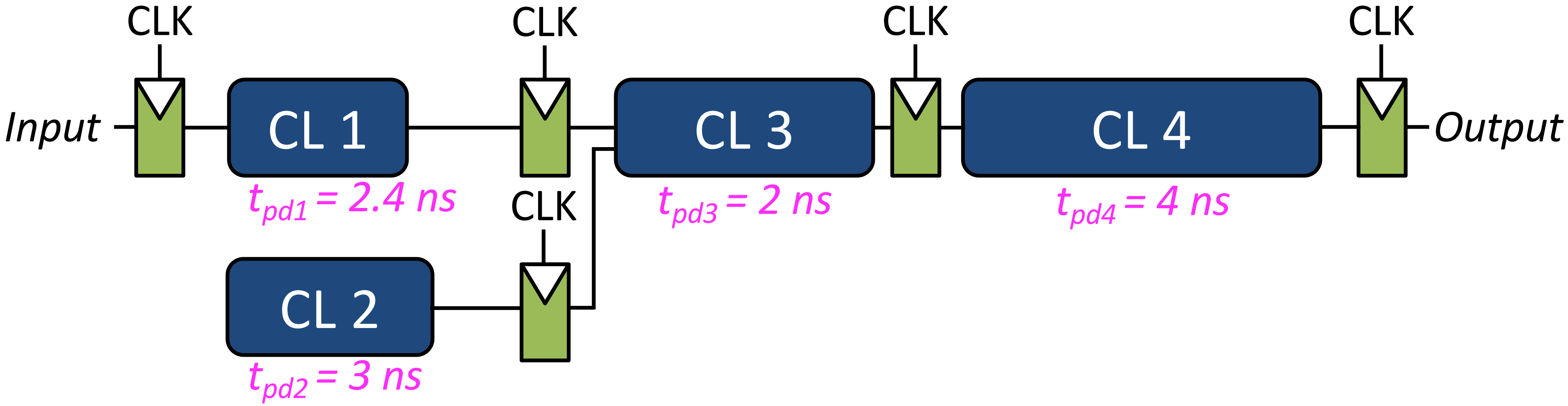


Latency = 2 X 5.5 ns = 11 ns

Frequency = 1/5.5 ns = 182 MHz

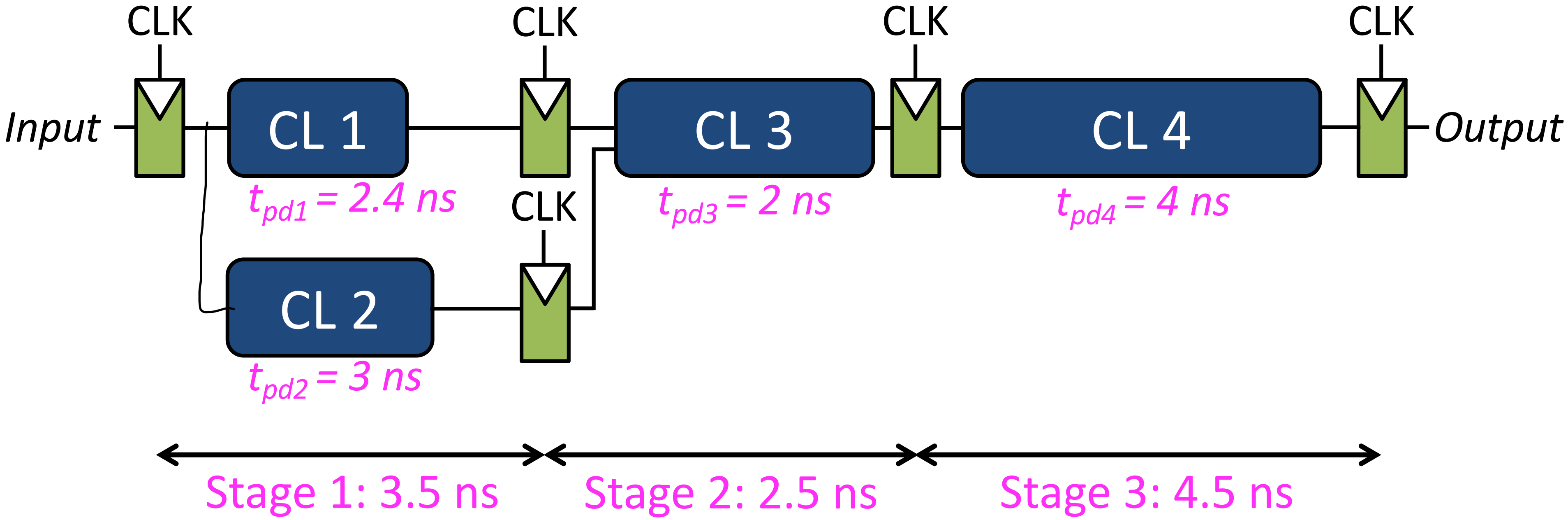
Exercise: 3-stage pipeline

Each task takes *three* clock cycles, but cycle time is further reduced



Exercise: 3-stage pipeline

Each task takes *three* clock cycles, but cycle time is further reduced

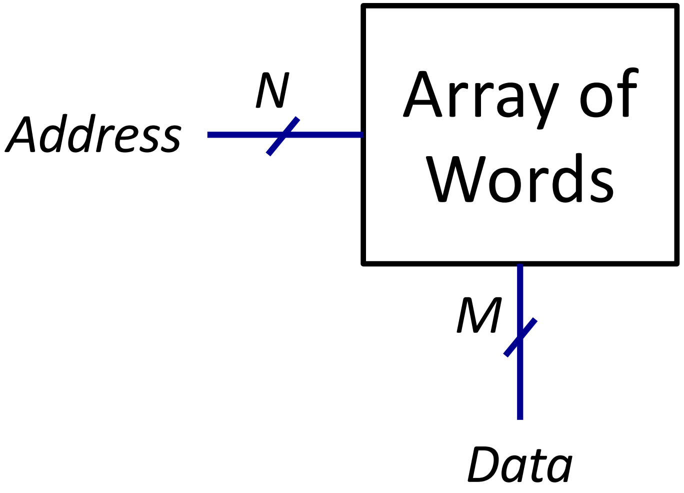


Latency = 3 X 4.5 ns = 13.5 ns

Frequency = 1/4.5 ns = 222 MHz

Memory

- A two-dimensional array of memory cells
  - N-bit address, so 2^N rows
  - Each row is M-bit wide and contains one word of data
  - The array contains 2^N M-bit words



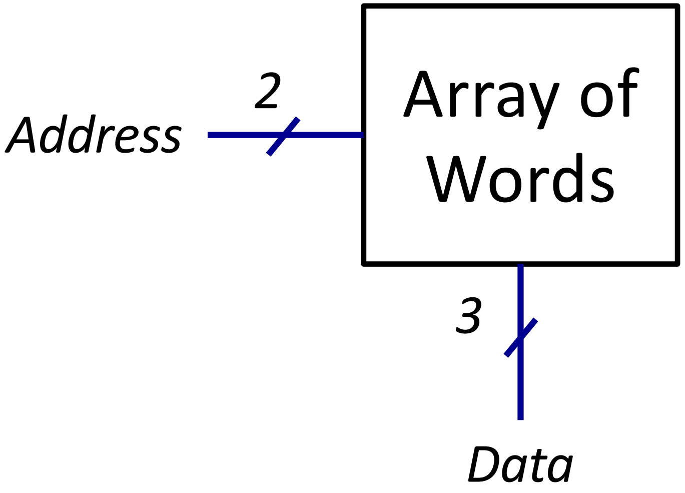
Address and Data

- Data is stored inside memory like the register file
- Address is presented to memory by an external circuit

- Each house is a memory cell (contains data)
- If we know the address, we can reach the house, but address has no physical existence

Example

- 2-bit address and 3-bit words

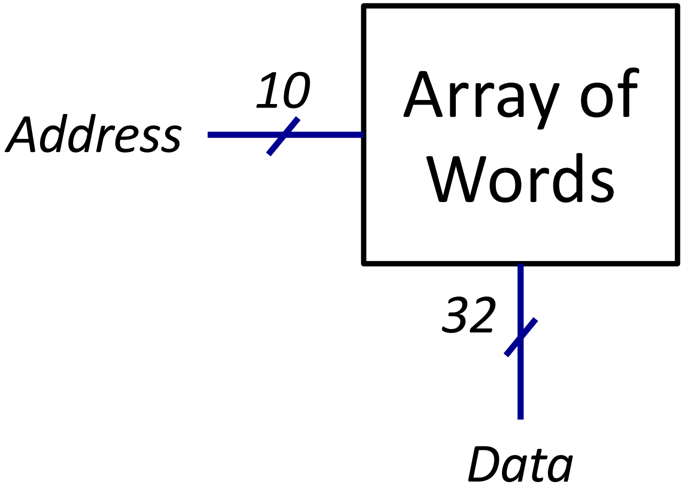


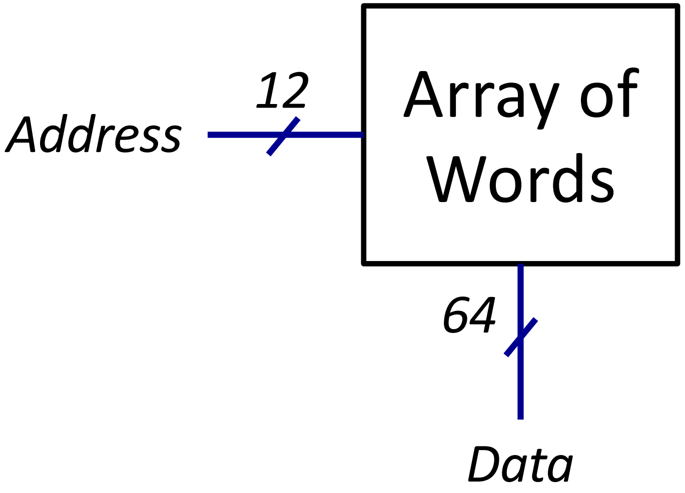
Address	Data		
11	0	1	1
10	1	1	0
01	1	1	1
00	1	0	0

Example

$\text{K bytes} = \frac{\text{K bits}}{8}$

- Size of memory (left) = $2^{10} = 1k.\ 1k \times 32 \rightarrow 32k\ \text{bits} = 4kB$
- Size of memory (right) =





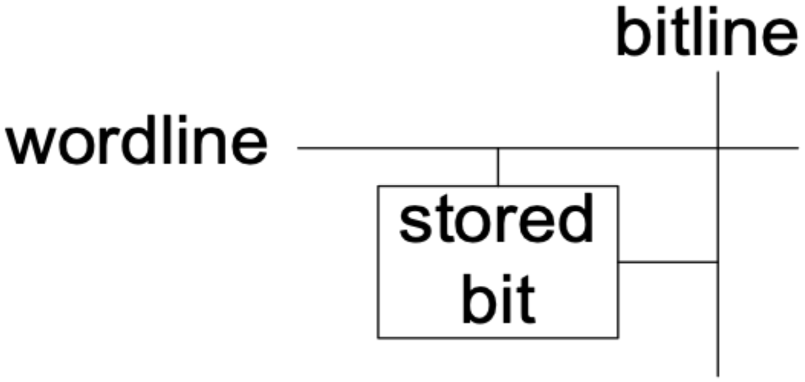
Read/Write Access

- Looking up a word stored at an address in memory is called a *read access* or simply *read*
- Updating a word stored at an address in memory is called a *write access* or simply *write*

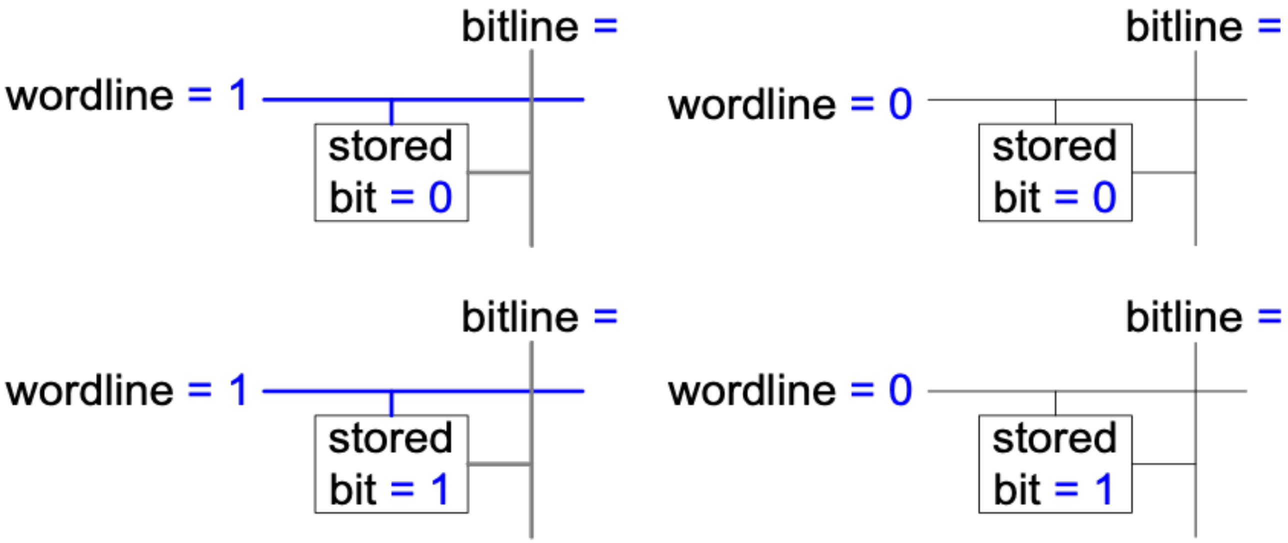
- Typically, a read is called a *memory load* or simply *load* when the word is read from memory into register file
- Similarly, write is called a *memory store* or simply *store*

Memory Cell

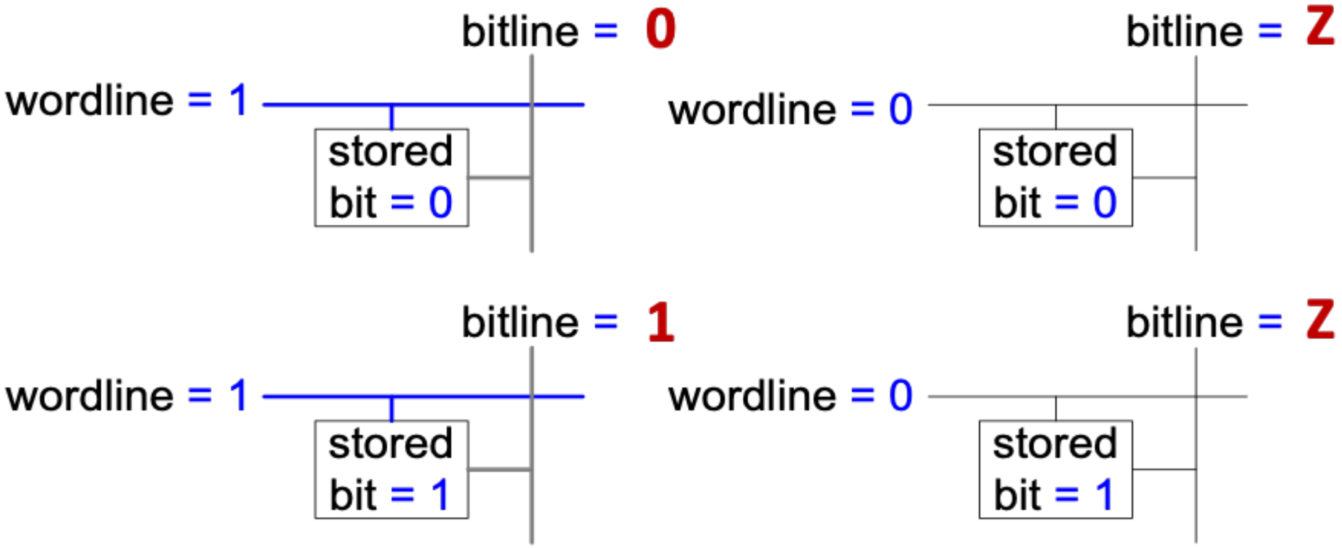
- We call it a bit cell (more technical term)
- A bit cell is connected to a bitline and a wordline
- Each bit cell contains one bit of data
- When the wordline is HIGH, the stored bit transfers to or from the bitline



Example: Bit Cell Operation



Example: Bit Cell Operation



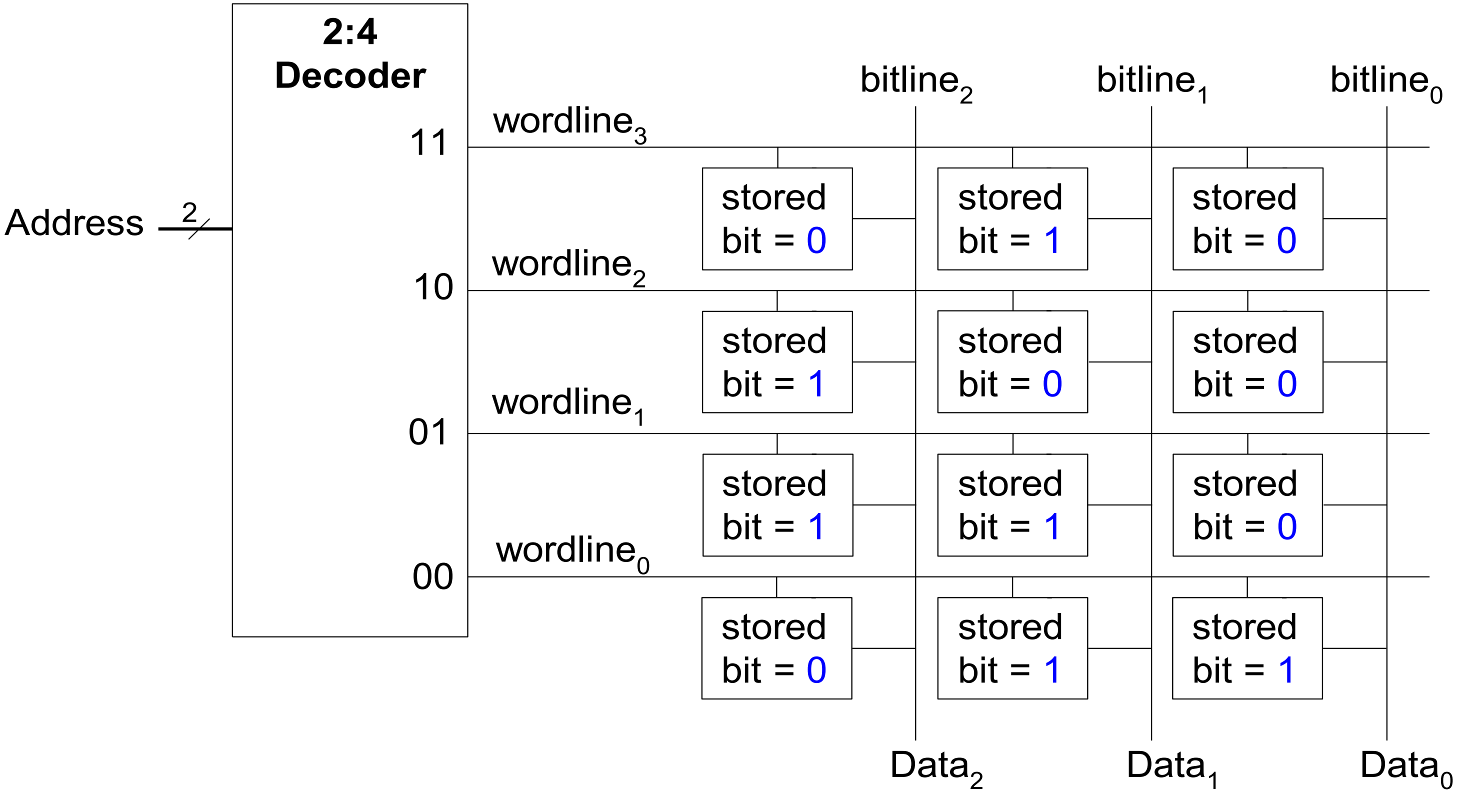
- Z is neither 0 nor 1 in digital electronics
- The wire is cut-off from the circuit (in this case the bit cell)

Reading and Writing Bit Cell

- Read
  - A special circuit is used to bring the bitline in Z state
  - The wordline is then set to HIGH
  - The stored value in the bit cell then drives the bitline
- Write
  - The bitline is driven (set) to 0 or 1
  - The wordline is turned on, connecting the bitline to the stored bit
  - The contents of the bit cell change to 0 or 1

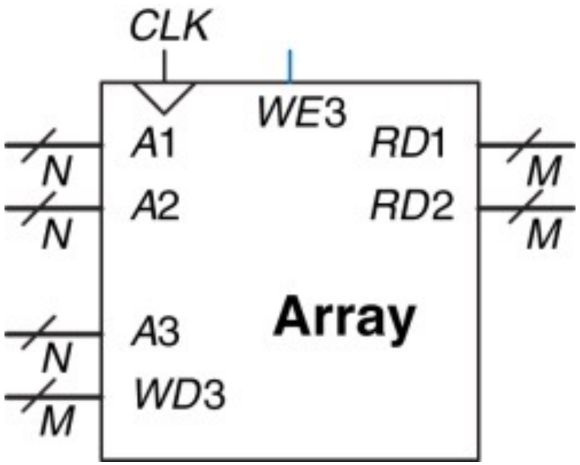
Memory Array Organization

- Decoder drives the wordline HIGH based on the address
- Data on the selected row appears on the bitlines



Memory Ports

- Each memory port gives read and/or write access to one memory address
- Multiported memories can access several address simultaneously
- Example of three-ported memory
  - Port 1 reads the data from address A1 onto the read data output RD1
  - Port 2 reads the data from address A2 onto the read data output RD2
  - Port 3 writes the data from the write data input WD3 into address A3 on the rising clock edge if WE3 is HIGH



Memory Specification

- Size
  - Width
  - Depth
- Ports
  - How many?
  - Type

Random Access Memory

- Random access memory or RAM is a type of memory for which accessing any data word results in the same delay as any other data word
- The main memory in a typical computer (e.g., your laptops) is RAM
- RAM is volatile
    - If the power is removed from the computer, the data in RAM is no longer there

RAM vs. Storage

- Hard disk (left) and tape (right)
  - Sequential access is faster than random access
  - Mechanical movement is required to access data
  - Non-volatile or persistent storage

Memory Classification

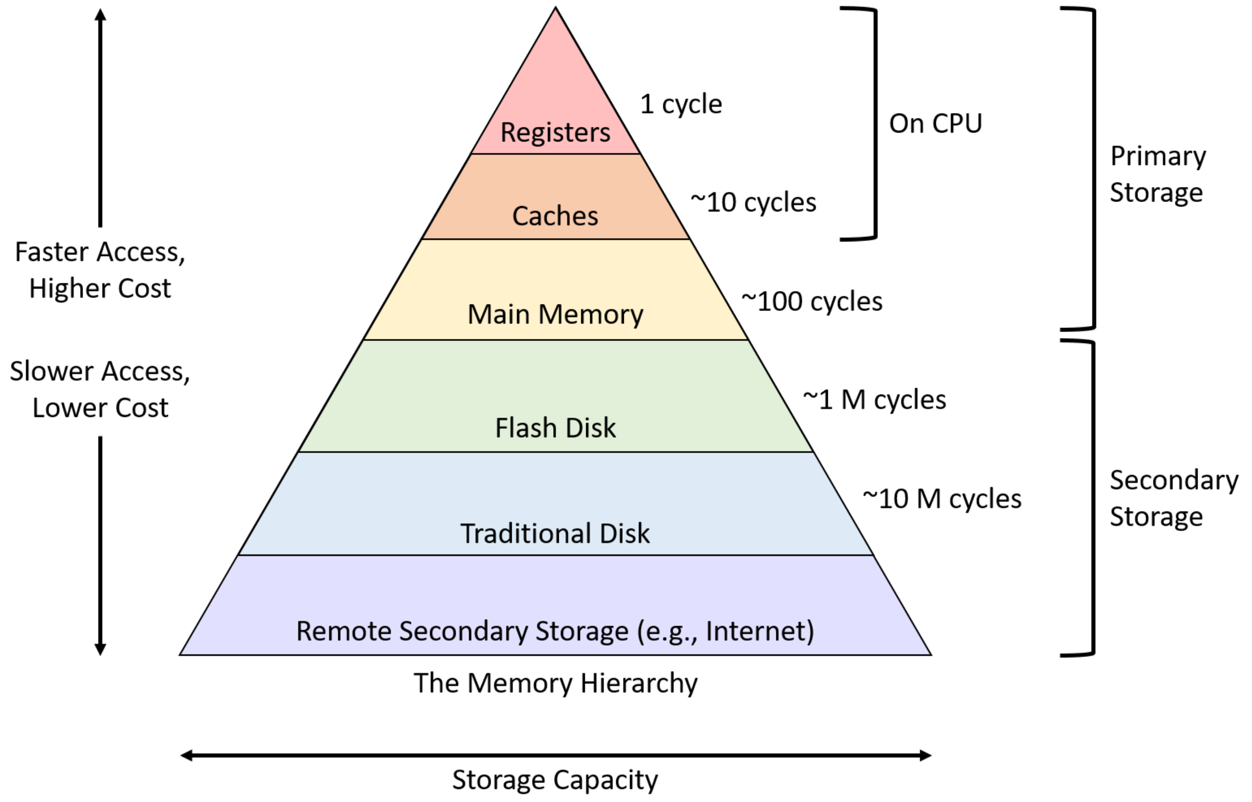
- RAM is classified according to the formation of the bit cells
  - Static RAM stores data bits using a pair of cross-coupled inverters
  - Dynamic RAM stores data bits using the presence or absence of charge on a capacitor (will return to this after the teaching break)

SRAM vs DRAM

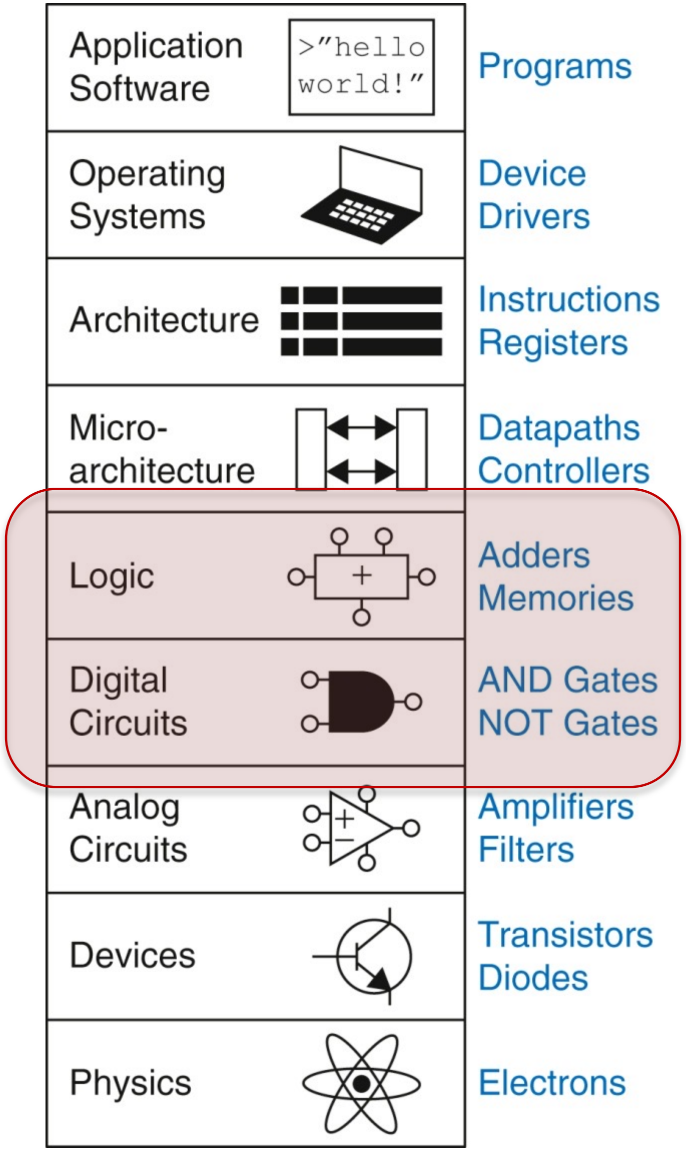
- SRAM is fast and expensive
- Typically, the memory close to the CPU uses the SRAM technology
    - Register file
    - Cache (after the teaching break)
- The memory far from the processor uses the DRAM technology
    - Your computer’s main memory is DRAM

Memory Hierarchy

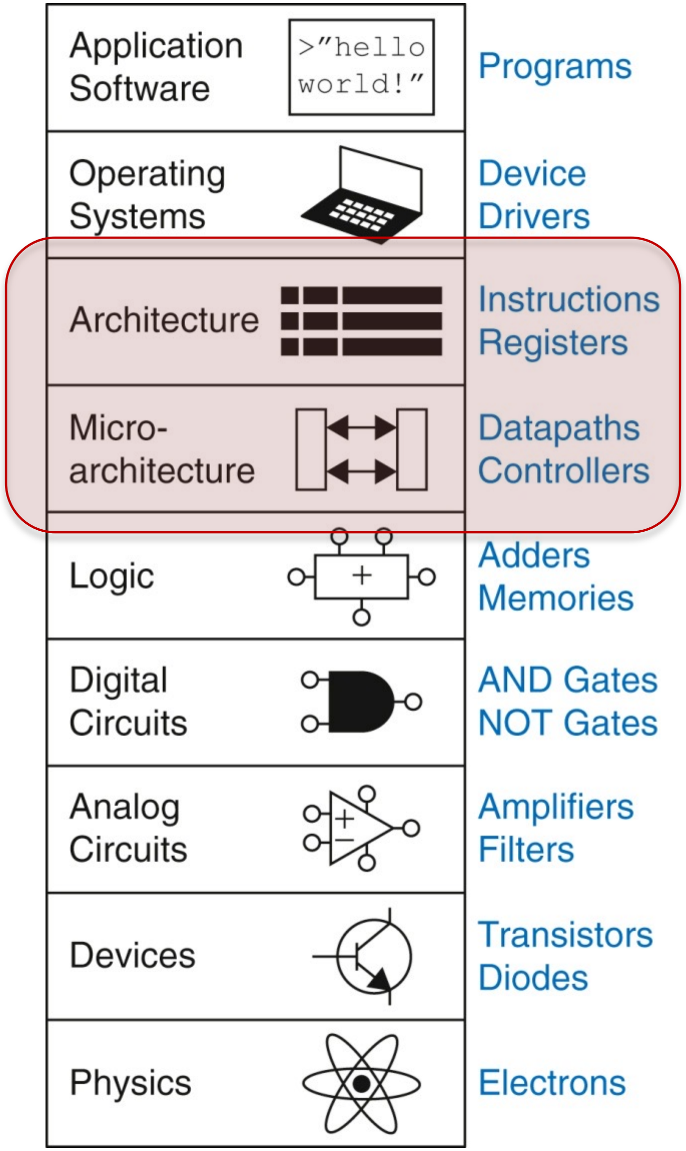
- We will return to this stuff after the teaching break



The Memory Hierarchy


Application Software
>"hello world!"
Programs
Operating Systems
Device Drivers
Architecture
Instructions Registers
Micro-architecture
Datapaths Controllers
Logic
+
Adders Memories
Digital Circuits
AND Gates NOT Gates
Analog Circuits
+
−
Amplifiers Filters
Devices
Transistors Diodes
Physics
Electrons

We are here



Moving up a few abstraction layers?

Week 5 : Architecture

Week 6: Microarchitecture

The Architecture Layer

- Our goal in the first half of the course is to understand and build a processor
- We can specify the combinational circuits and the traffic light controller in English
- How should we write the specification of a processor?
  - We need a systematic approach to manage the complexity of a processor
- The formal specification of a processor (and computer) takes place at the “architecture” abstraction layer

Architecture/ISA

- The architecture or **I**nstruction **S**et **A**rchitecture (**ISA**) is the programmer’s view of the computer
- ISA specifies the set of instructions a computer can perform (think of it like the language of a computer)
  - Instruction = word
  - ISA = vocabulary
- Each instruction specifies
  - Operation (What exactly to do?)
  - Operands (Where to find the data to operate upon?)
- Example: Add two 16-bit binary numbers in registers R1 and R2 (recall the register file from Lab 3 – 4)

Operands

- Operands can be in register and memory
  - Note: The register file alone cannot house all the data programs need
  - Register file is fast and expensive (and hence small)
- If operands can be in memory, then we must have instructions to fetch the operands from memory
  - Load
  - Store
- Is there a third possibility for operand location?
  - From the instruction itself (keep this in mind)

Assembly Language

- Instructions written in a symbolic format so humans can read/understand them easily is called assembly language
  - ADD, SUB, LDR, STR, MUL, ROR, MOV, BIC
  - We will study all of the above ARM instructions
- A sequence of instructions is called assembly code

Remark: Don't worry about what this means! Just want to show how assembly looks

```
SUB     R0,     R1,     R2
ADD     R8,     R4,     R5
ADD     R9,     R6,     R7
SUB     R3,     R8,     R9
```

Machine Language

- Instructions are encoded as 1’s and 0’s in a format called the machine language
  - ADD can be represented by binary code 0000
  - SUB can have a possible encoding of 0001
  - Register R1: 00 (example)
  - Register R2: 01

```
0   0   0   0   1   0   0   0   0   0   1   1   0   0   0   0
0   0   0   1   1   0   0   0   0   0   1   1   1   0   0   0
```

Hypothetical machine code (above) is a sequence of machine language instructions stored in memory

Instruction Format

- An instruction consists of several fields
- Each field has a different meaning
- An instruction format specifies the meaning of each field
- An ISA can have many instruction formats

- **Remark:** The ISA we use in labs (QuAC) and lectures (ARM v4) are fixed-width ISA. Each instruction format in the ISA uses a fixed number of bits (16 or 32)
- **Remark:** A popular ISA (x86) has variable-sized instructions (keep it in mind when you build the CPU and think of the difficulty of implementing such as ISA)

Microarchitecture

- An ISA is a specification
- Microarchitecture is the implementation of an ISA
    - The specific arrangement of registers, memories, ALUs, and other building blocks to form a processor is called microarchitecture
- The same ISA can have many different implementations
    - Tradeoffs in performance, power, price
    - A company X builds two processors for a high-end laptop and a cheap cell phone, respectively, that can both run programs targeting the same ISA named Y

More Examples

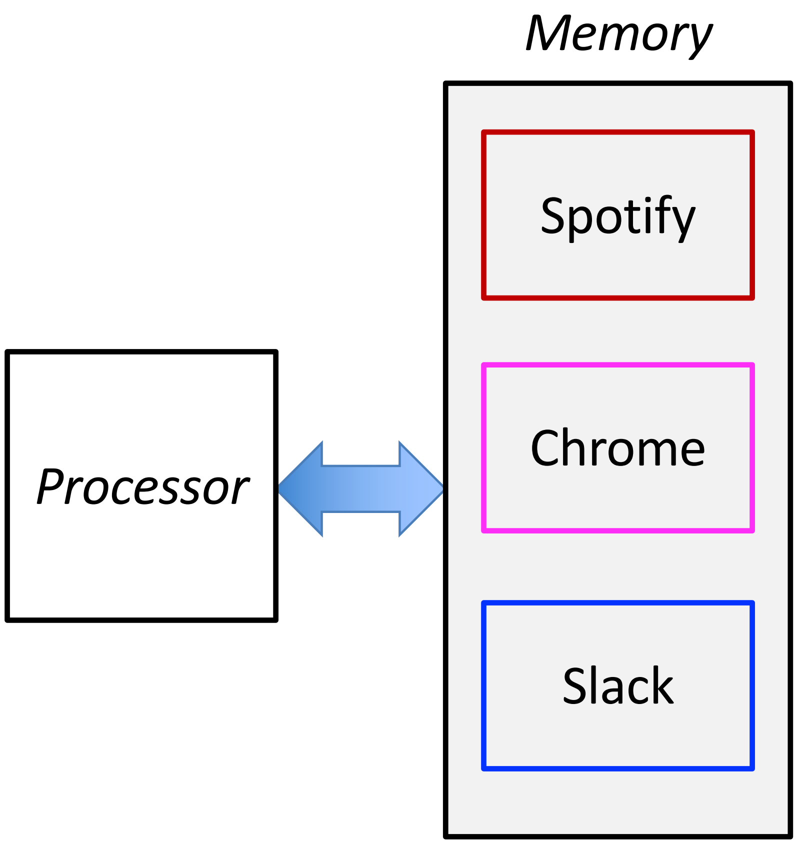
- Intel and AMD build processors targeting the x86 ISA
- QuAC ISA is for teaching purposes
  - Each group/student will build a CPU differently
  - One group may use two adders rather than one; different styles for register file
  - Both are building a processor for the same ISA
  - Their microarchitectures are different

Remarks

- All programs running on a computer use the same instruction set
- All software applications, such as Spotify and Word, are eventually *compiled* into a series of simple instructions
- **Compiler:** A program that transforms a program written in a high-level language (C, C++, Python) to assembly instructions
- **Assembler:** A program that transforms assembly code to machine code

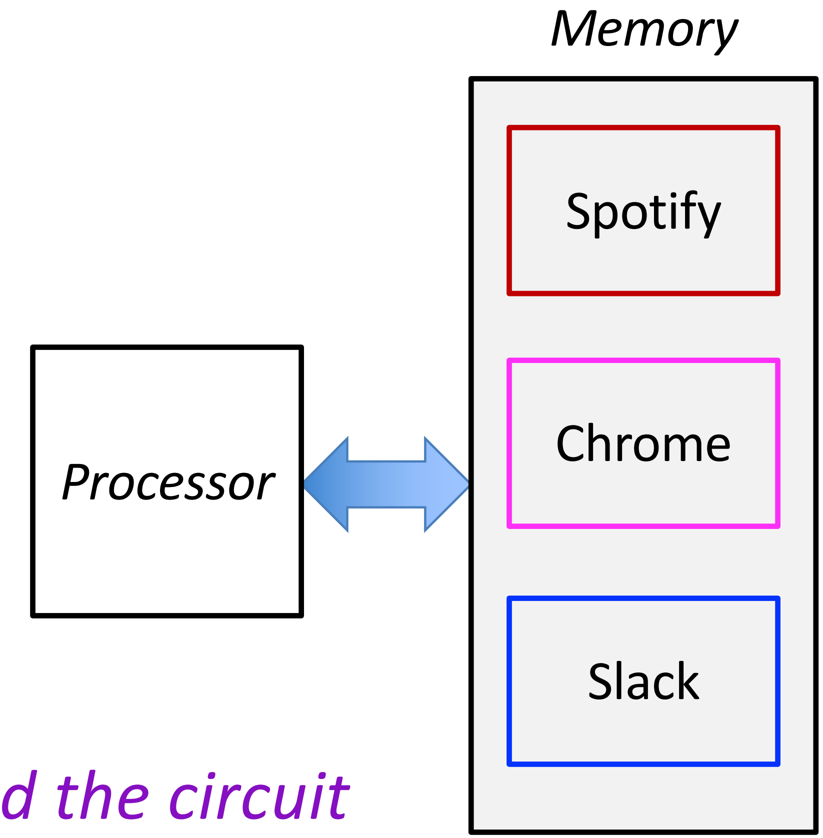
Stored Program Concept

- Two key principles
  - Instructions are represented as binary numbers
  - Programs (*in the form of machine code*) are stored in memory (*like data*)



Stored Program Concept

- How do processors execute machine code?
  - Fetch an instruction from memory
  - Decode the meaning of the instruction
  - Execute the instruction
  - Repeat until done



Your job in the labs/assignment is to build the circuit for fetching, decoding, and executing instructions

Popular ISAs

- Intel x86
  - High-performance desktop/laptop/server
  - Power-hungry (Apple’s recent shift to Apple silicon)
- ARM
  - Popular in the mobile/embedded domain
  - Lecture/textbook focus
  - M1 (Apple) uses ARM architecture
- RISC-V
  - A new open-source ISA gaining momentum
- QuAC
  - Teaching purposes (invented at ANU)

Quiz

- Ben has an excellent idea to make computers more efficient. To try out his idea he first picks a popular existing ISA.
  - *What is the advantage of Ben’s approach to pick an existing ISA instead of developing a new one?*
  - *What is the drawback of picking an existing ISA?*

Quiz

- Ben has an excellent idea to make computers more efficient. To try out his idea he first picks a popular existing ISA.
    - *What is the advantage of Ben’s approach to pick an existing ISA instead of developing a new one?*
        - *Binary compatibility enables existing programs to make use of Ben’s excellent idea without any effort*
        - *Historically, this aspect has led to ISA hegemony, where one popular ISA is dominant*
    - *What is the drawback of picking an existing ISA?*
        - *Think!*