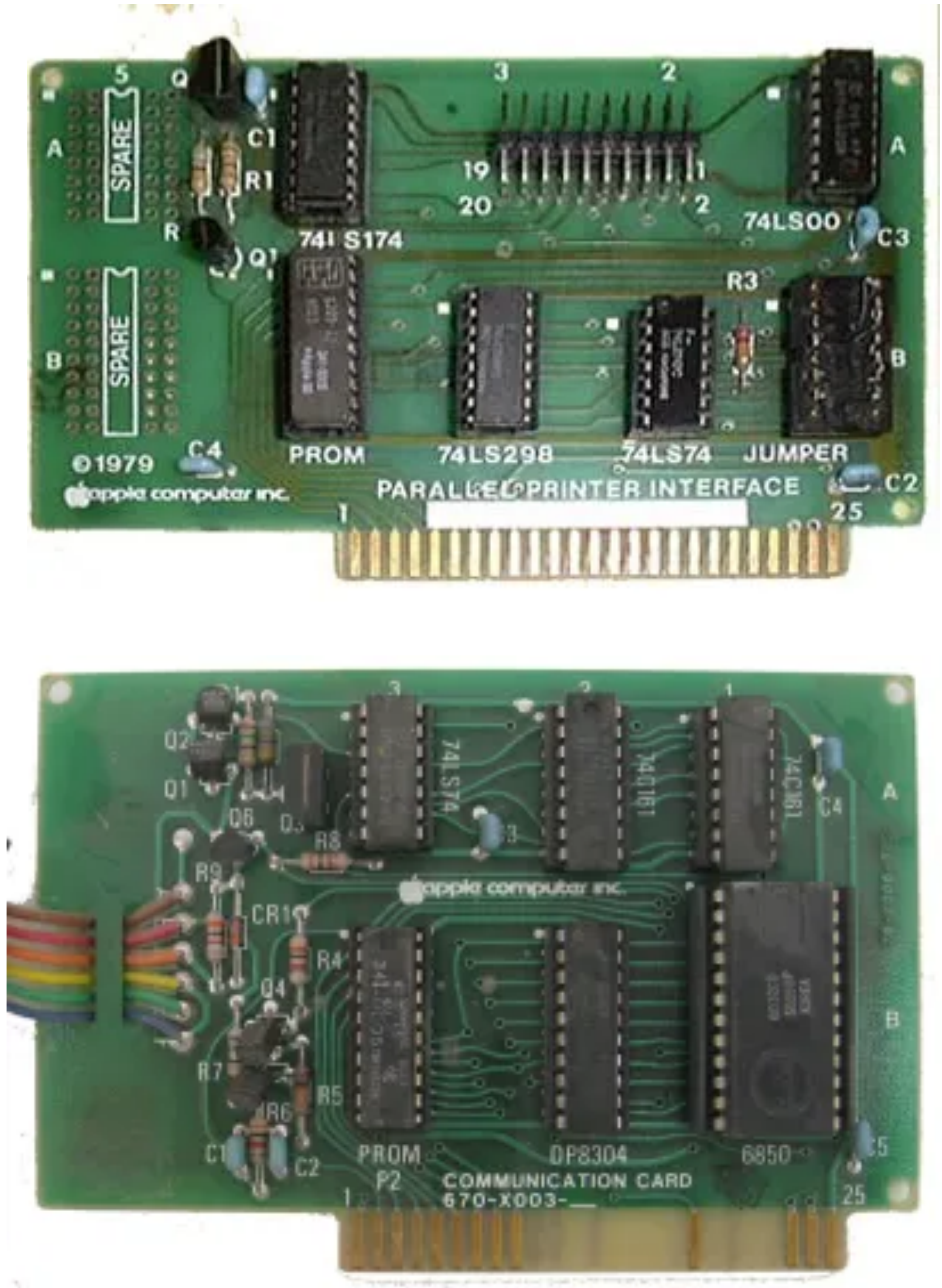
A Pipelined Processor



For further reading: https://www.apple2history.org/history/ah13/

Image Credit: (left) https://en.wikipedia.org/wiki/Apple_II (right) https://www.apple2history.org/history/ah13/

Outline

- Revisiting the pipelined processor
- Metrics for processor performance
- Ideal pipeline conditions

From Friday…

Pipeline registers to shorten clock cycle!

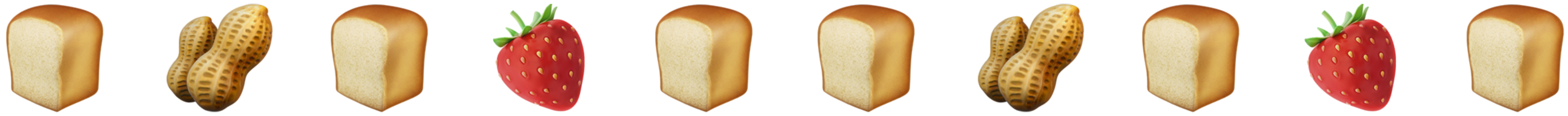
4
Adder
PC
50ps
Instruction Memory
250ps
50ps
Decoder Unit
100ps
MUX
Register File
150ps
MUX
50ps
ALU
200ps
50ps
Data Memory
250ps
MUX
50ps

Adding Pipeline Registers

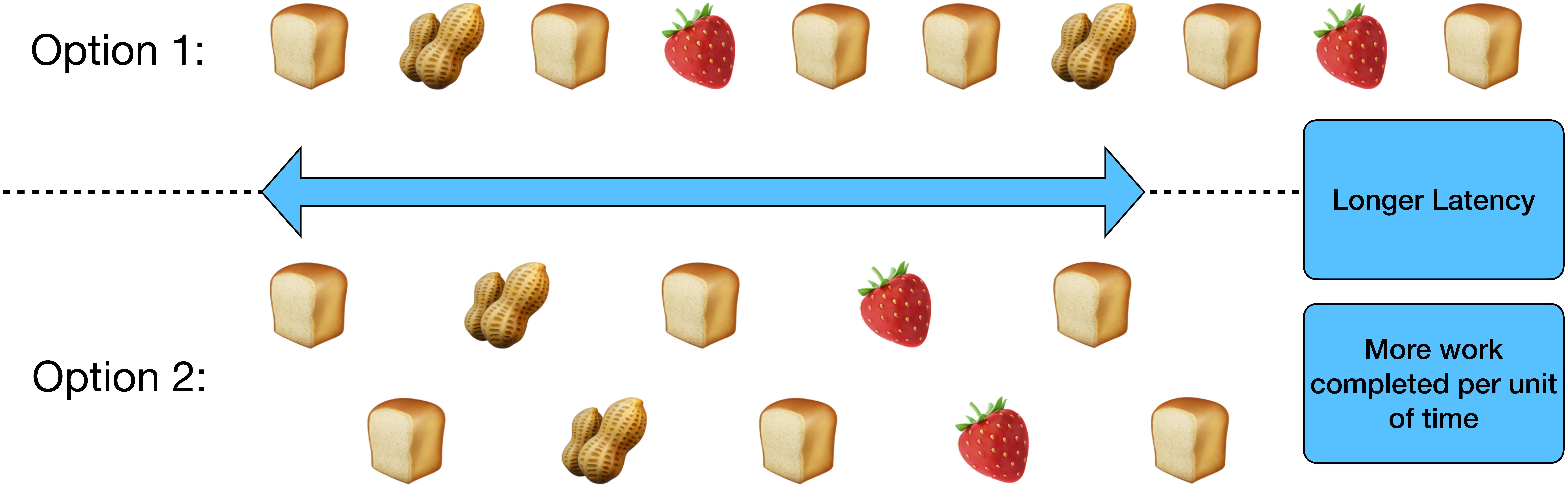
- By adding a state element to the data path, we can reduce our clock cycle time to the *minimum time between register files*
- To perform an instruction, it will now take multiple clock cycles to get through its execution
- If an instruction uses all pipeline stages, then it will take *longer* to execute than in the single stage processor
- Not all instructions use all pipeline stages!

Analyzing Pipeline Performance

- There are two ways to understand performance: *latency* and *throughput*
- Latency describes the time it takes for an operation to complete
- Throughput describes the amount of work completed per unit of time



The Efficient PB&J Construction



Option 1: 🍞 🥜 🍞 🍓 🍞 🍞 🥜 🍞 🍓 🍞

Longer Latency

Option 2:

🍞 🥜 🍞 🍓 🍞

🍞 🥜 🍞 🍓 🍞

More work completed per unit of time

Calculating PB&J Throughput

5 cycles per instruction

Overall CP(S): 8/5 = 1.6

5 instructions in 8 cycles

Pipeline is full!

t0
t1
t2
t3
t4
t5
t6
t7
t8

Chat with your neighbor(s)!

Suppose a pipeline has a depth of 12 and wants to perform 20 steps of execution. What is its throughput? How does the throughput change if it wants to perform 100 steps of execution?

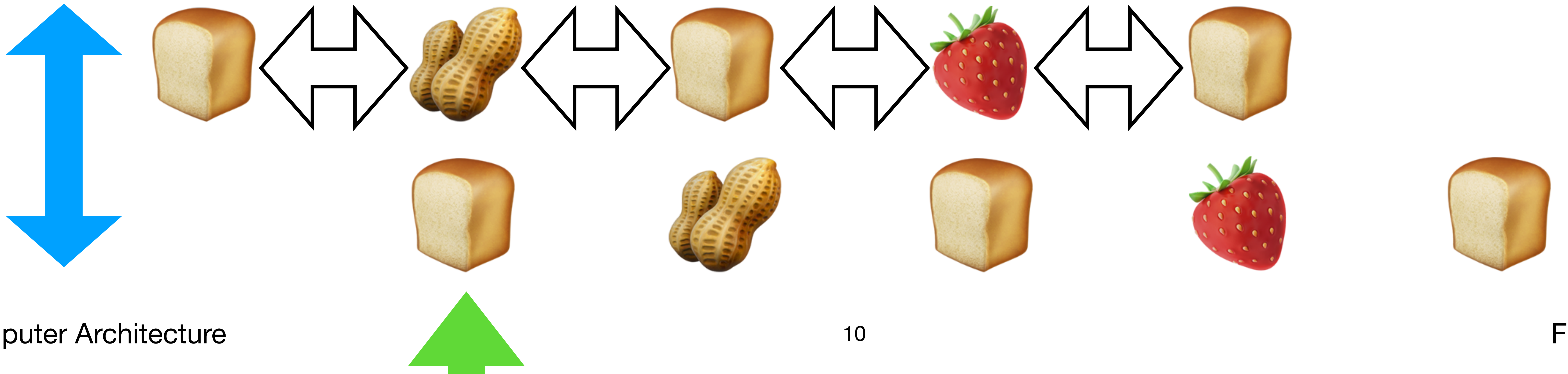
Some Pipelining Observations

- There are several characteristics of the PB&J task that make it ideal for pipelining

Even distribution of latency across stages

We want to produce PB&J sandwiches in perpetuity

At each next unit of time, we can start the next sandwich



Speed-up Due to Pipelining

- Under ideal conditions, we can compute the potential speedup of transforming any task into a sequence of pipelined stages!
- Time Between Instructions$_{\text{pipelined}}$ = Time Between Instructions$_{\text{non-pipelined}}$ / Number of Pipeline Stages
- If our task has five stages, and there is zero time to coordinate our stages, then we should have a 5 times speedup!

Reality of the Pipelined Processor

- Cycle time is decided by the longest stage, shorter executing stages have down time at the end of the clock cycle
- Correctly executing programs should have a well-defined start and end
- Instructions may have dependencies between each other, so we may not always be able to start the next instruction

~~Even distribution of latency across stages~~

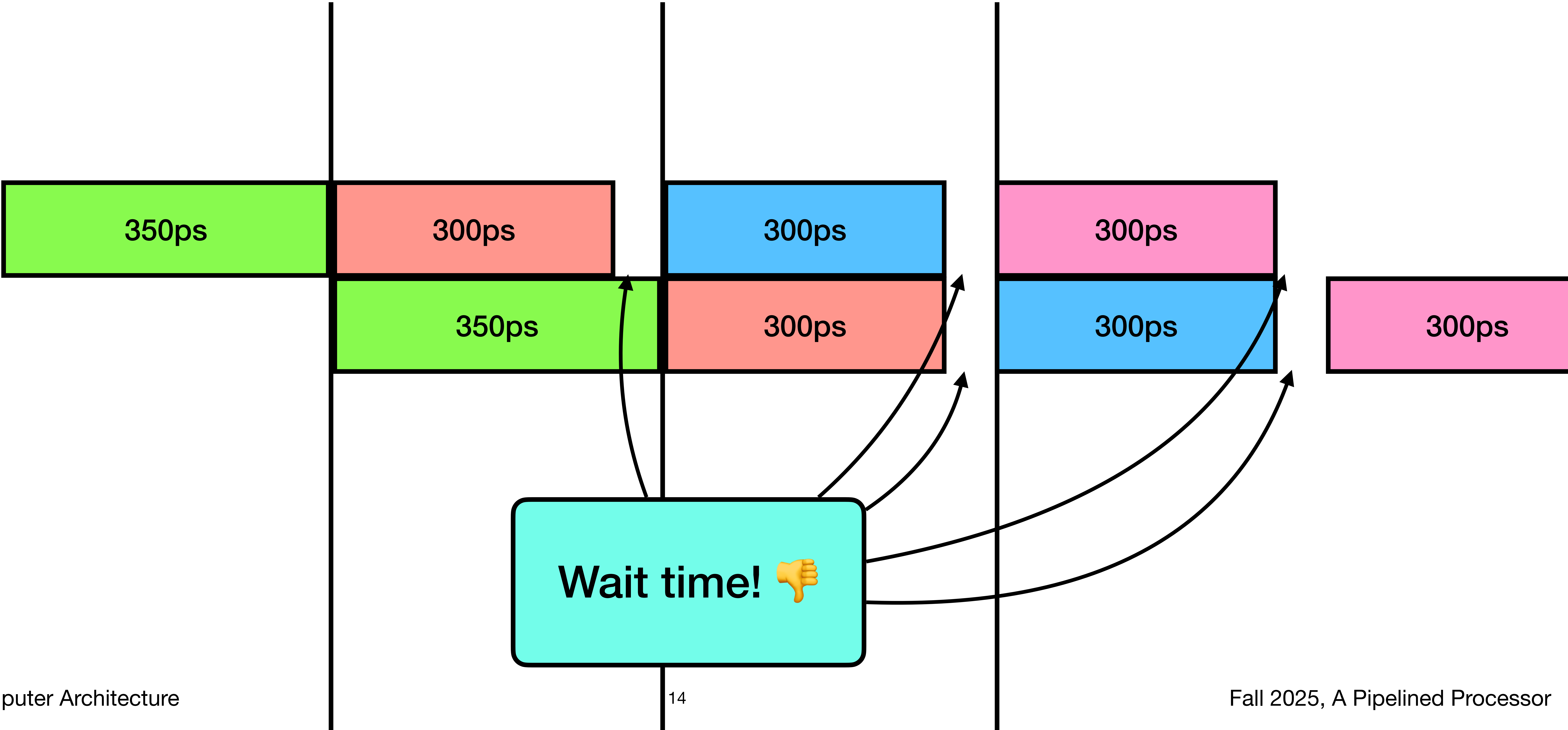
~~We want to produce PB&J sandwiches in perpetuity~~

~~At each next unit of time we can start the next sandwich~~

Wait Time Between Stages

4
Adder
PC
Instruction Memory
350ps
Decoder Unit
MUX
300ps
Register File
MUX
ALU
250ps
Data Memory
300ps
MUX

Wait Time Between Stages



Takeaways

- Performance can be understood as latency and/or throughput ➡️ both are important!
- Throughput describes the amount of work performed per unit of time (i.e., instructions per cycle)
- Pipelining can help improve throughput
- It is difficult to achieve ideal pipelines in practice