Parallelism

Temporal and spatial parallelism in digital systems

Introduction

Restaurant analogy

Digital systems

Conclusion

Improving performance

A *token* is a group of outputs produced from a group of inputs.

Measures of performance

- Latency
  - The time for one token to be processed
- Throughput
  - The number of tokens processed per unit time

Types of parallelism

- Spatial parallelism
  - Duplicate the hardware to improve throughput
- Temporal parallelism (pipelining)
  - Divide the processing of a token into stages, like an assembly line

A restaurant analogy

- Improving McDonald's Big Mac "performance"

Creating a Big Mac

Steps to create a Big Mac

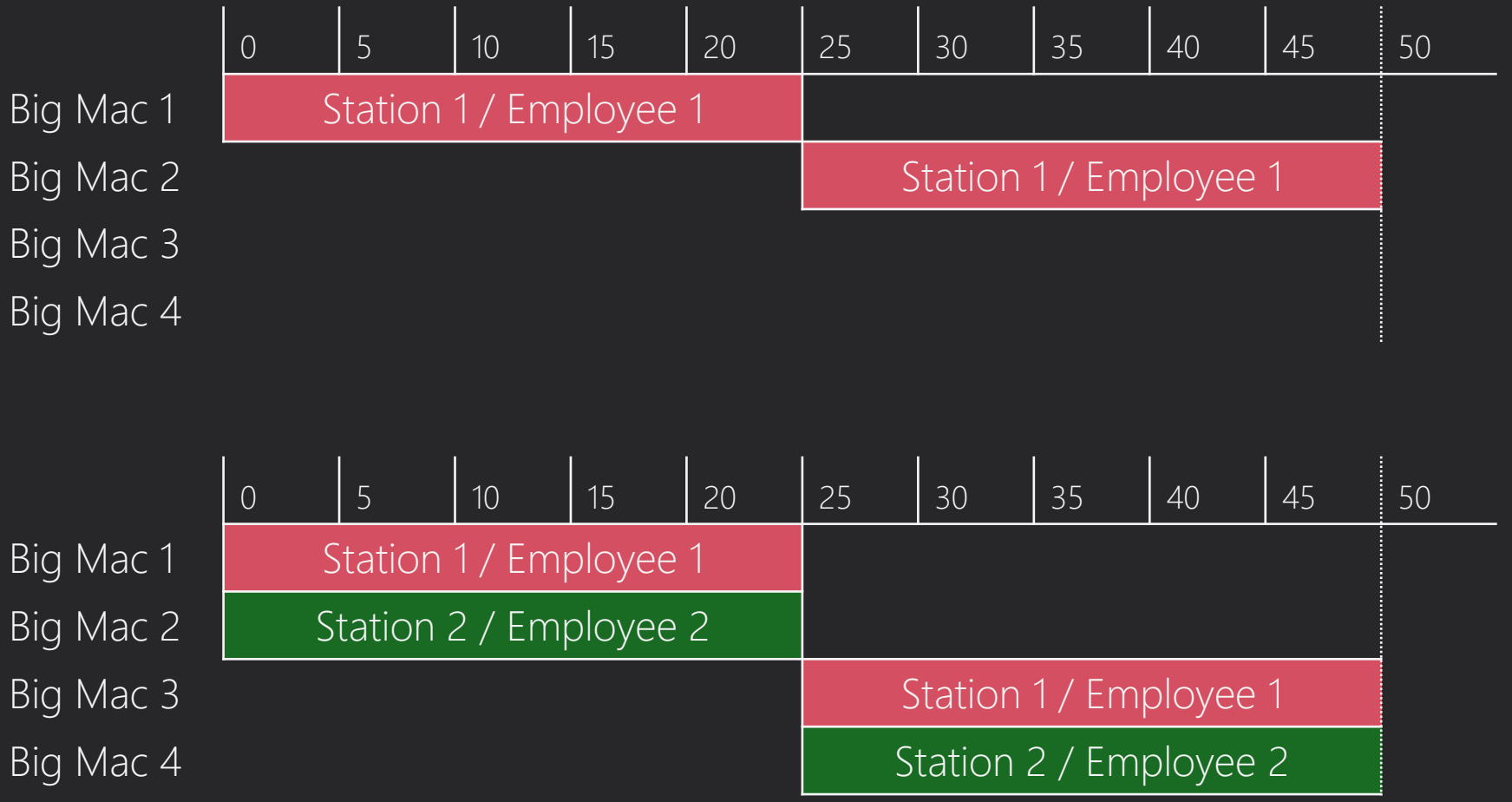
1. Toast the bun (B) (5 seconds)
2. Add the toppings (T) (10 seconds)
3. Add the meat (M) (5 seconds)
4. Close the box (C) (5 seconds)

- The latency for creating a Big Mac is: $5 + 10 + 5 + 5 = 25\ seconds$

- McDonald's does not want customers waiting more than 2 minutes from the time they place their order.

  Since it takes 25 seconds to create a Big Mac, then the throughput is: $4.8\ \frac{Big\ Mac}{2\ min}$

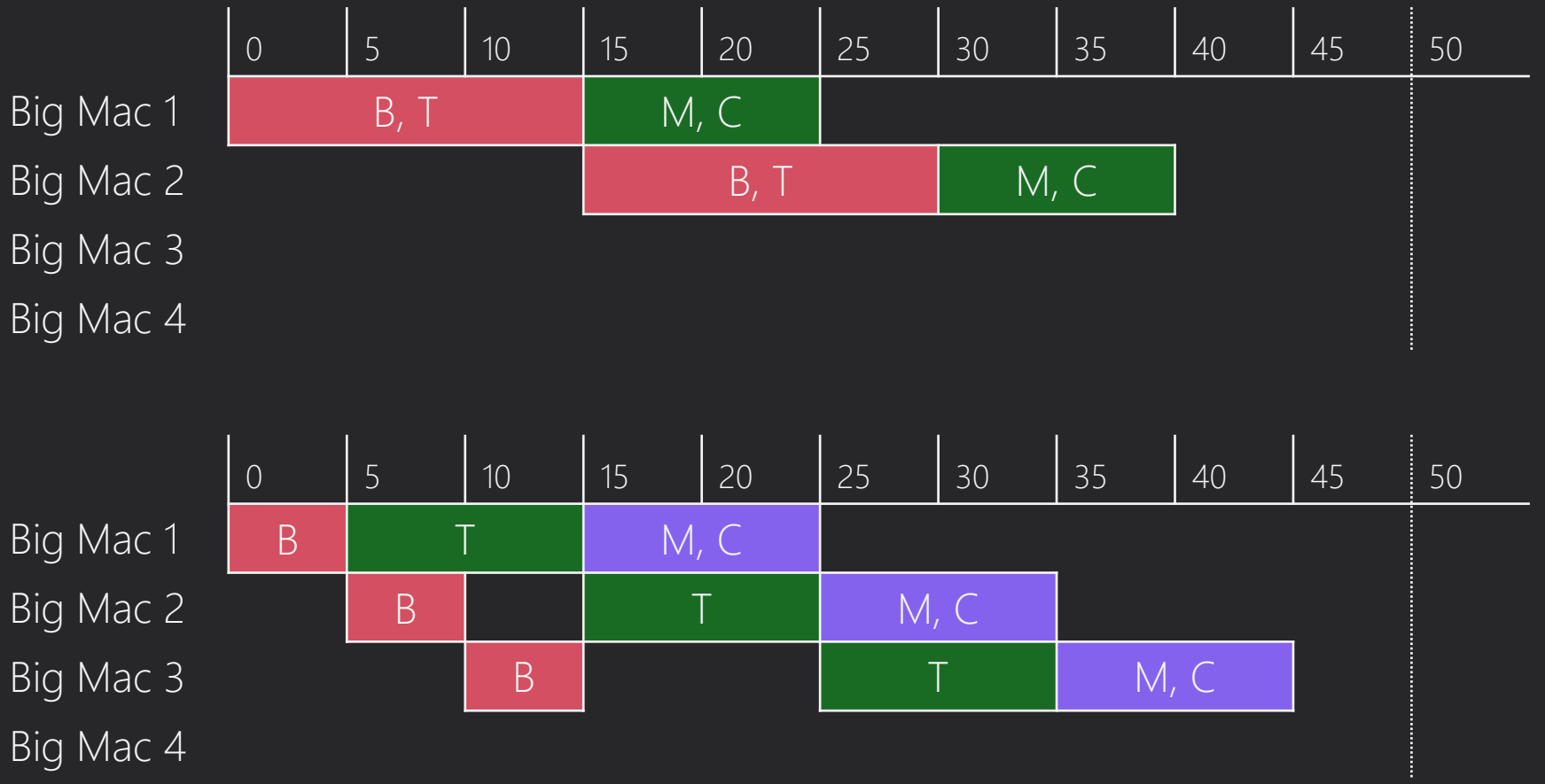
Spatial parallelism (N stations)



- One station

- Two stations
  - Two toasters
  - Two topping trays
  - Two meat trays
  - Two box stacks

Temporal parallelism (1 station)



- Two employees
  - E1: Steps 1 and 2
  - E2: Steps 3 and 4

- Three employees
  - E1: Step 1
  - E2: Step 2
  - E3: Steps 3 and 4

Temporal versus spatial parallelism

Parallelism	Stations	Steps	Latency	Throughput	Notes
None	1	1	L	$1/L$	
Spatial (only)	N	1	L	N/L	• Duplicating a station is expensive (may be infeasible)
Temporal (only)	1	N	L	N/L^{**}	• Balancing the latency of steps is difficult • **only true if steps are balanced • Otherwise, if the longest step is L_{max}, then $throughput = 1/L_{max}$

Synchronous circuits

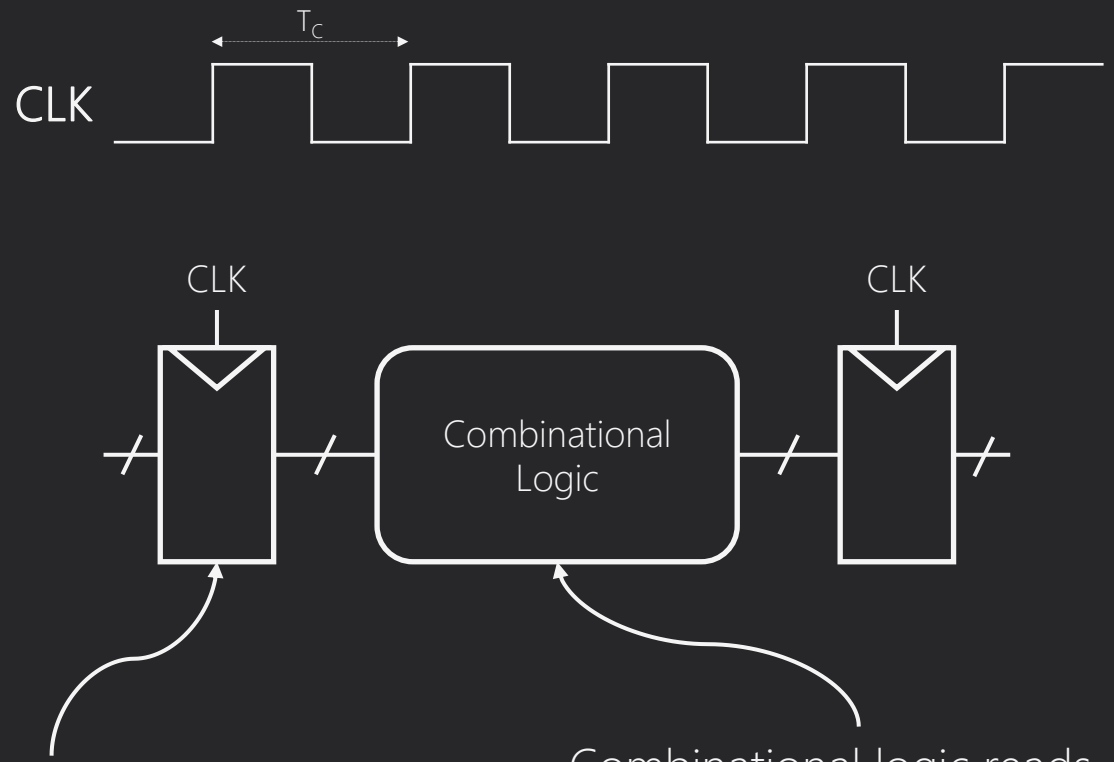
Overheads and constraints

2- and 3-stage pipelines

Pipelining circuits

From analogy to circuits

Synchronous sequential circuits



- The circuit is *synchronized* to the ticks of a clock
- The clock rises to 1 and falls to 0 with some clock period T_C
  - $frequency = {}^{1}/_{T_C}$
  - T_C is a.k.a. *cycle time*
- State changes happen on rising (and/or falling) clock edges

Sequencing overhead and constraints

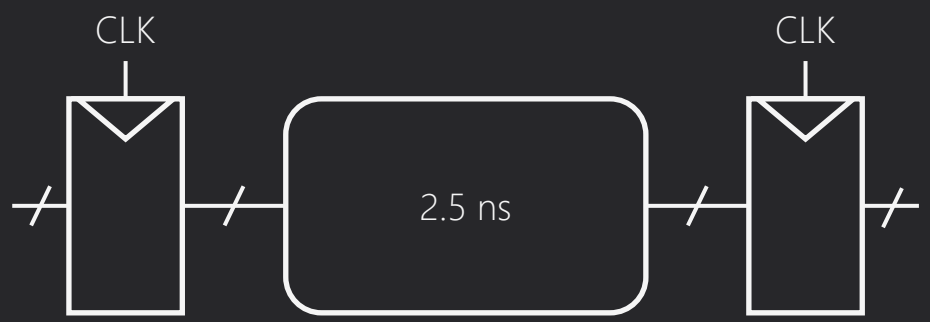
Sequencing overhead

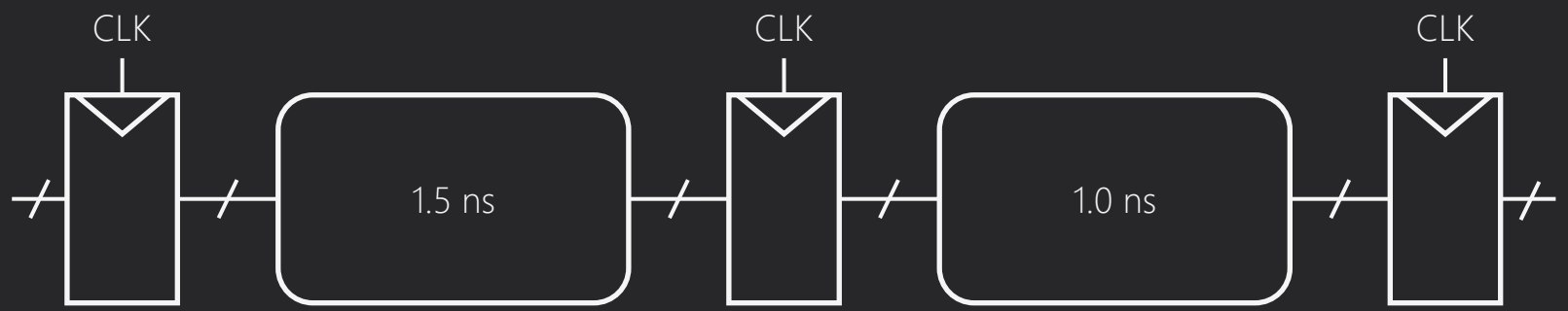
- It takes time to read from a register
  - Setup time (t_{setup})
- It takes time to write to a register
  - Propagation delay (t_{pcq})
- Sequencing overhead is the sum of setup time and propagation delay

The max-delay constraint

- Combinational logic takes time to produce a result (t_{pd})
- The latency of combinational logic must be within the max-delay constraint
  - $t_{pd} \leq T_C - (t_{pcq} + t_{setup})$

Temporal parallelism



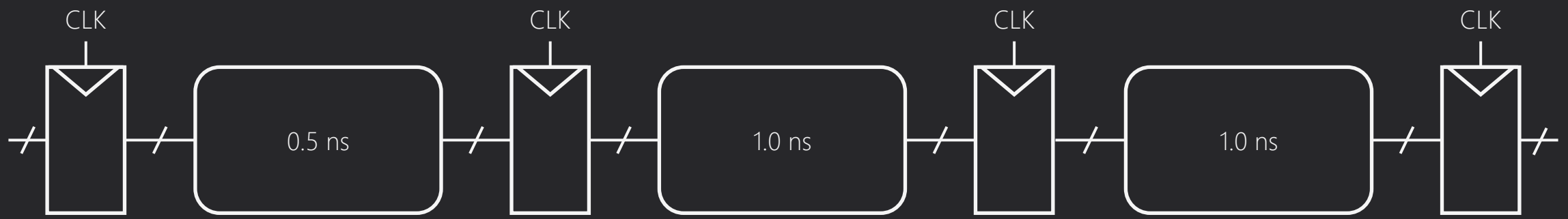


- Assume sequencing overhead of 0.5 ns
- No parallelism
  - $2.5 + 0.5 = 3\ ns$
  - $f = {}^{1}/_{3\ ns} = 333\ MHz$
  - $L = 3\ ns$

- Two-stage pipeline
  - Stage 1: $1.5 + 0.5 = 2\ ns$
  - Stage 2: $1.0 + 0.5 = 1.5\ ns$
  - $f = {}^{1}/_{2\ ns} = 500\ MHz$
  - $L = 2 + 1.5 = 3.5\ ns$

A three-stage pipeline

- Longest stage(s): $1.0 + 0.5 = 1.5\ ns$
- $f = {}^{1}/_{1.5\ ns} = 666\ MHz$
- $L = (0.5 + 0.5) + (1.0 + 0.5) + (1.0 + 0.5) = 4\ ns$



Conclusion

Recapping the important points

Performance trade-offs

- Spatial parallelism
  - Improve throughput by duplicating hardware
  - A trade-off with area

- Temporal parallelism
  - Improve throughput by adding pipeline registers between stages of logic
  - A trade-off with latency (and some area overhead)

Connections to computer architecture

- Instruction pipelining
  - A technique to exploit instruction-level parallelism
  - What about dependencies between instructions?
- Pipelining long-latency operations
  - Low latency: add
  - High latency: floating-point multiply
- Spatial parallelism
  - Decide which hardware is worth duplicating