Optimizations

CSCI 2050U - Computer Architecture

Randy J. Fortier
@randy_fortier

Outline

- Optimizations
  - Pipelining
  - Hyperthreading

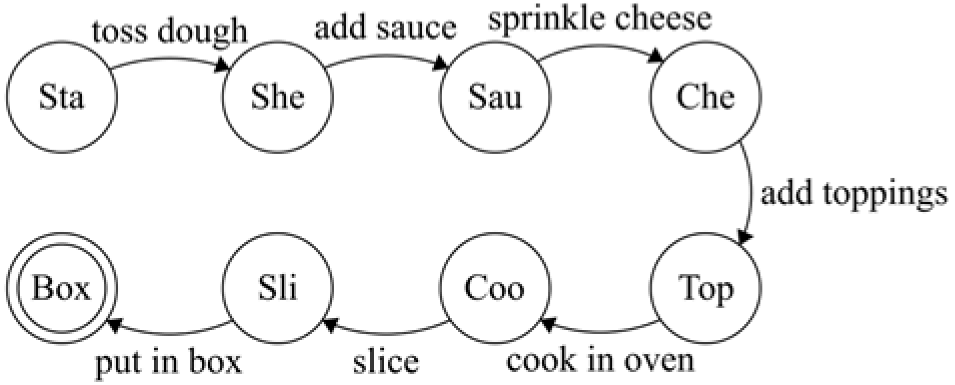
Optimizations

CSCI 2050U - Computer Architecture

Randy J. Fortier
@randy_fortier

Pipelining

- Imagine you work at a pizza place, making pizzas:
  - Toss the dough
  - Add sauce
  - Sprinkle cheese
  - Add toppings
  - Cook in oven
  - Slice
  - Box
- When not busy, you might:


$$\begin{tikzcd}
Sta \arrow[r, bend left, "toss dough"] & She \arrow[r, bend left, "add sauce"] & Sau \arrow[r, bend left, "sprinkle cheese"] & Che \arrow[d, bend left, "add toppings"] \\
Box & Sli \arrow[l, bend left, "put in box"] & Coo \arrow[l, bend left, "slice"] & Top \arrow[l, bend left, "cook in oven"]
\end{tikzcd}$$

Pipelining

- What if it is busy?

Pipelining

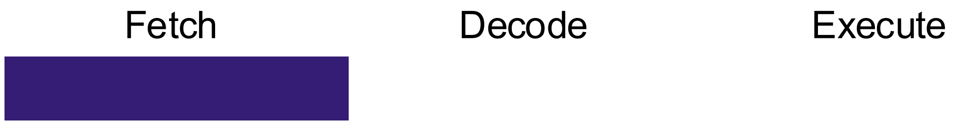
- What if it is busy?
  - The pizza takes a long time to cook in the oven, and you can’t really do anything useful (e.g. slice) that pizza until it has finished cooking
  - Parallelism - while the first pizza is in the oven, you can start the second pizza

Pipelining

- What if it is busy?
  - The pizza takes a long time to cook in the oven, and you can't really do anything useful (e.g. slice) that pizza until it has finished cooking
  - Parallelism - while the first pizza is in the oven, you can start the second pizza
- This is quite similar to executing instructions in a CPU
  - Fetch the instruction and operand values
  - Decode the instruction
  - Execute the instruction

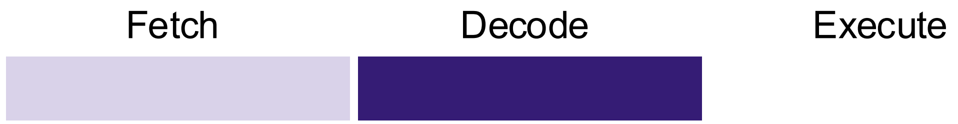
Pipelining

- Without pipelining
  - Fetch the opcode and operand values for the first instruction



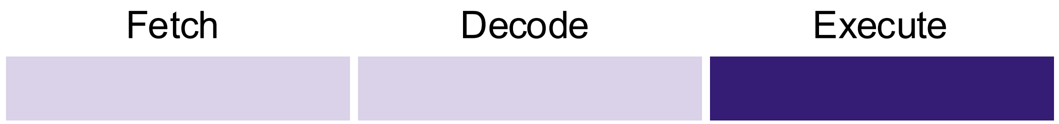
Pipelining

- Without pipelining
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction



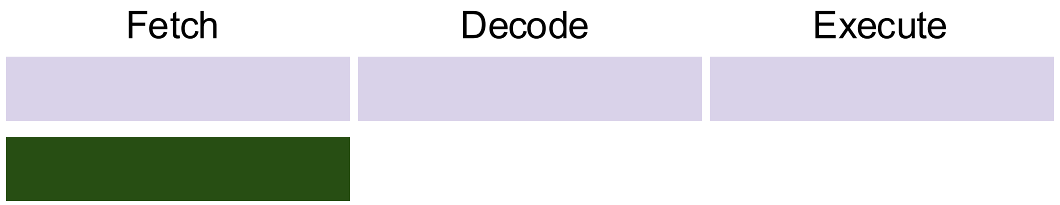
Pipelining

- Without pipelining
    - Fetch the opcode and operand values for the first instruction
    - Decode the first instruction
    - Execute the first instruction



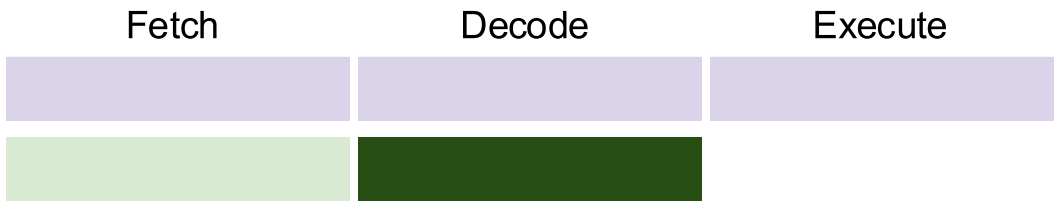
Pipelining

- Without pipelining
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction
  - Execute the first instruction
  - Fetch the opcode and operand values for the second instruction



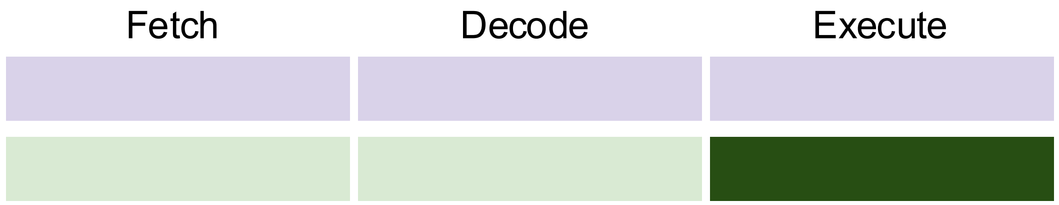
Pipelining

- Without pipelining
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction
  - Execute the first instruction
  - Fetch the opcode and operand values for the second instruction
  - etc.



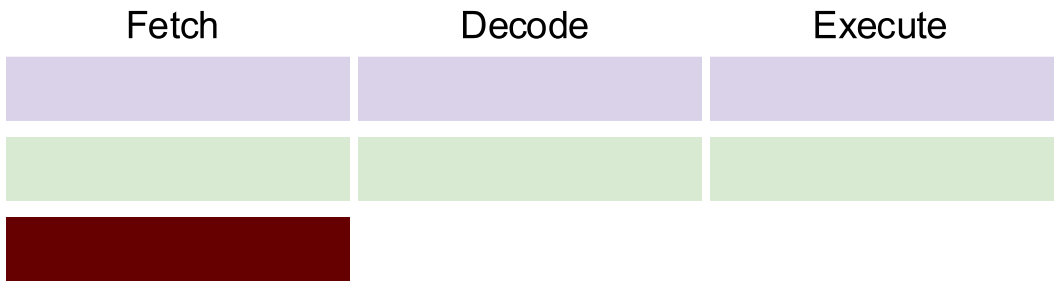
Pipelining

- Without pipelining
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction
  - Execute the first instruction
  - Fetch the opcode and operand values for the second instruction
  - etc.



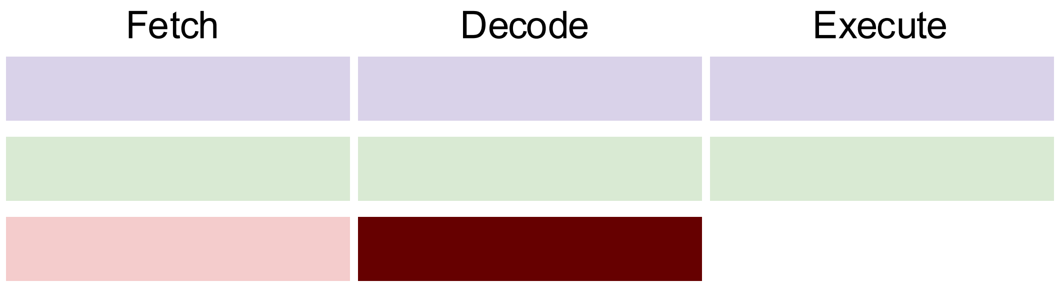
Pipelining

- Without pipelining
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction
  - Execute the first instruction
  - Fetch the opcode and operand values for the second instruction
  - etc.



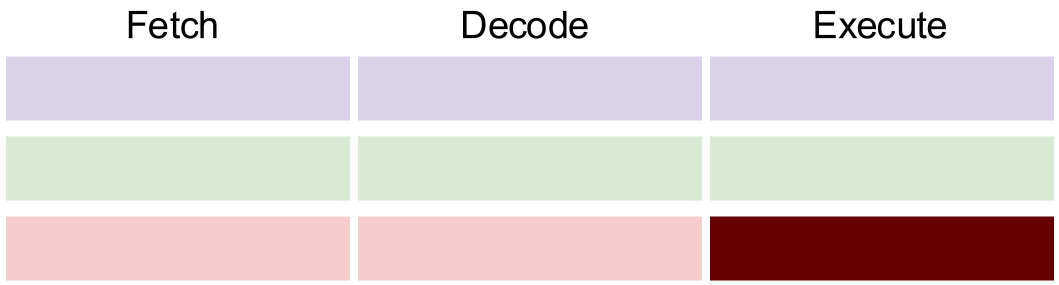
Pipelining

- Without pipelining
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction
  - Execute the first instruction
  - Fetch the opcode and operand values for the second instruction
  - etc.



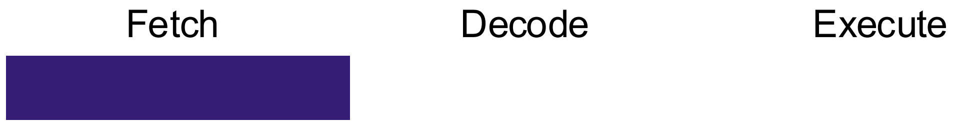
Pipelining

- Without pipelining
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction
  - Execute the first instruction
  - Fetch the opcode and operand values for the second instruction
  - etc.



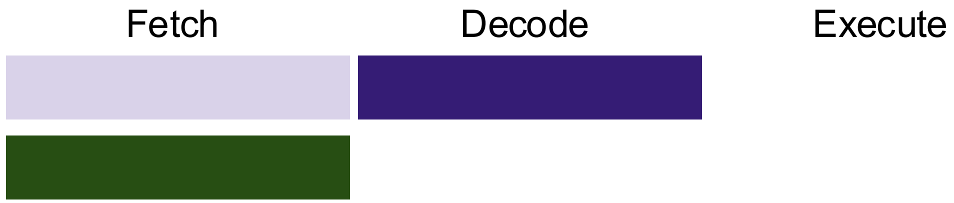
Pipelining

- Pipelining in action
    - Fetch the opcode and operand values for the first instruction



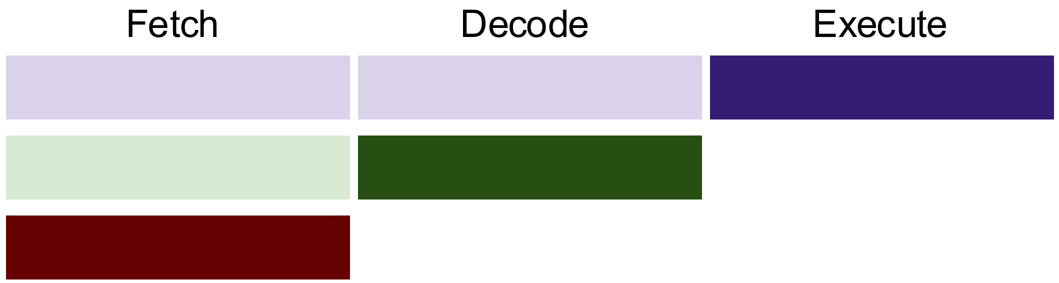
Pipelining

- Pipelining in action
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction, and fetch the second



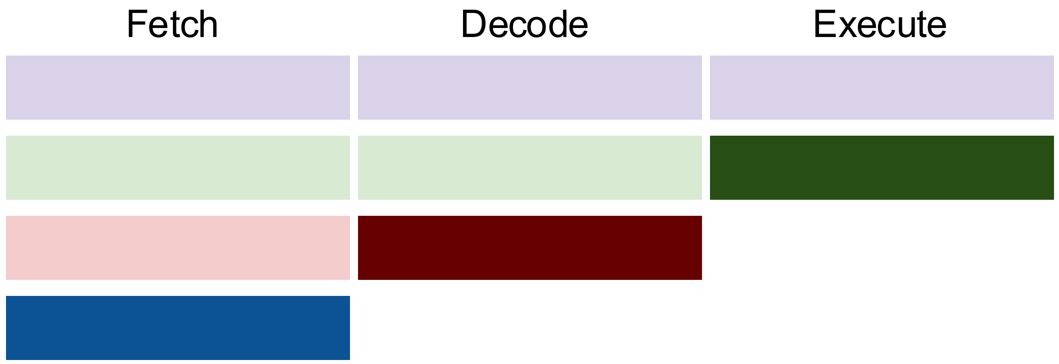
Pipelining

- Pipelining in action
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction, and fetch the second
  - Execute the first, decode the second, and fetch the third



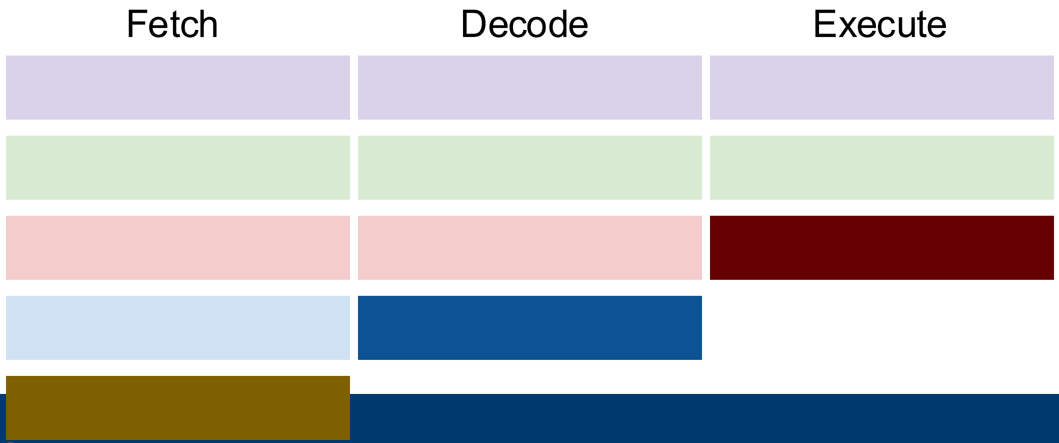
Pipelining

- Pipelining in action
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction, and fetch the second
  - Execute the first, decode the second, and fetch the third
  - etc.



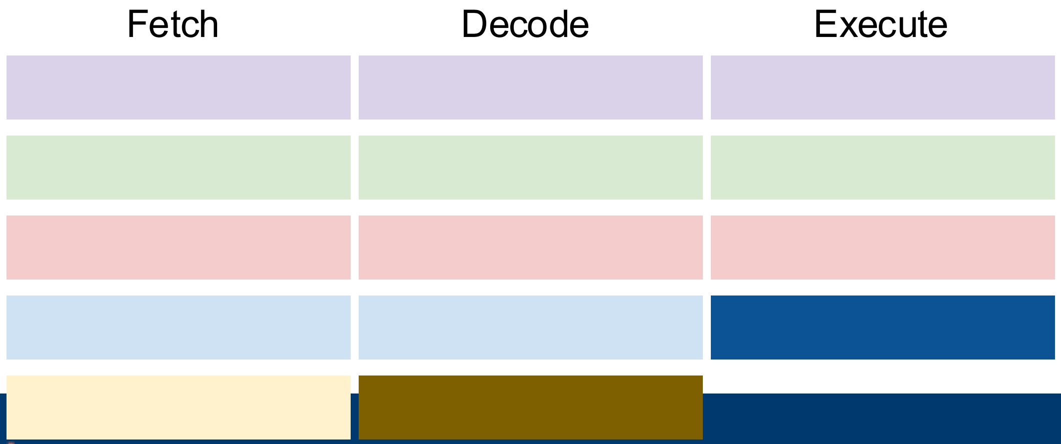
Pipelining

- Pipelining in action
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction, and fetch the second
  - Execute the first, decode the second, and fetch the third
  - etc.



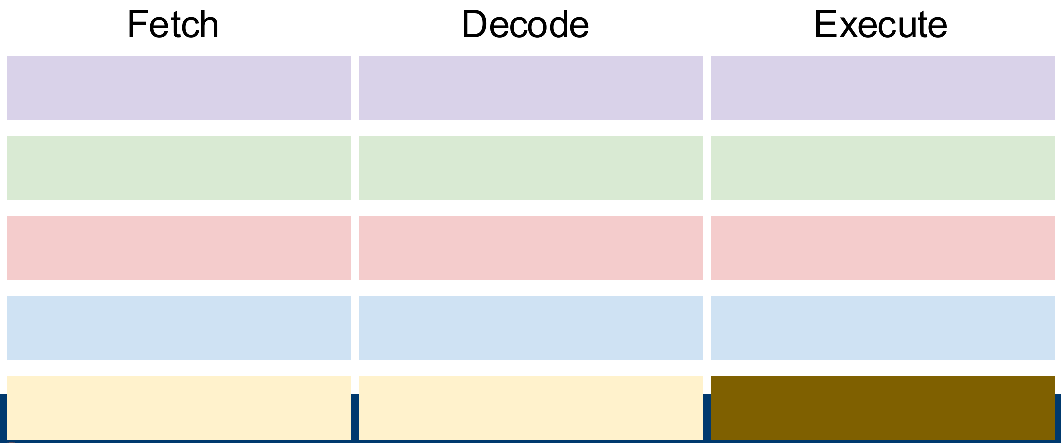
Pipelining

- Pipelining in action
  - Fetch the opcode and operand values for the first instruction
  - Decode the first instruction, and fetch the second
  - Execute the first, decode the second, and fetch the third
  - etc.



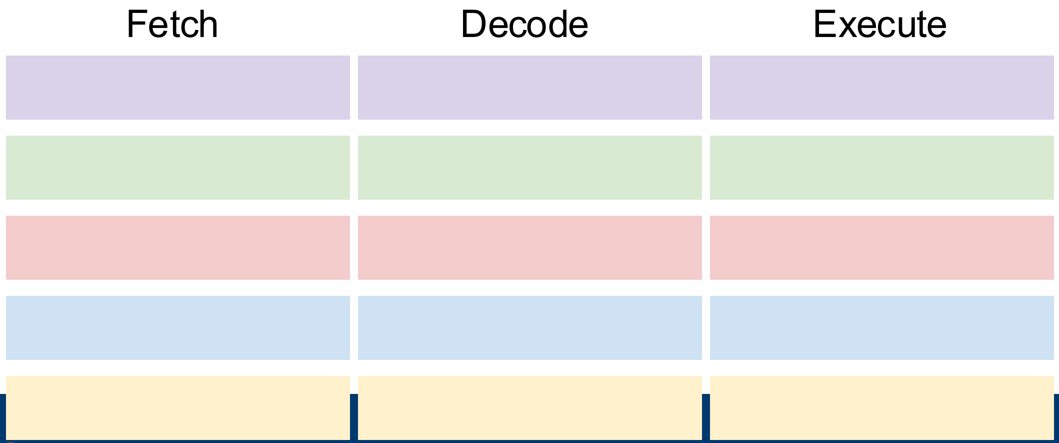
Pipelining

- Pipelining in action
    - Fetch the opcode and operand values for the first instruction
    - Decode the first instruction, and fetch the second
    - Execute the first, decode the second, and fetch the third
    - etc.



Pipelining

- Pipelining in action
    - Fetch the opcode and operand values for the first instruction
    - Decode the first instruction, and fetch the second
    - Execute the first, decode the second, and fetch the third
    - etc.



Pipelining

- But...
  - Pipelining breaks down on branch/jump instructions
  - Pipelining may break down if a subsequent instruction requires the values from a previous instruction

Pipelining

- Branch prediction
    - The CPU guesses which path the program will take, and pre-executes the instructions along that execution path
        - e.g. If a branch has been true for the past 10 iterations of the loop, let's assume it will be true again this time
    - If it guesses wrong, it may need to undo everything it has done since the branch
        - This may be worth it if the predictions are accurate enough

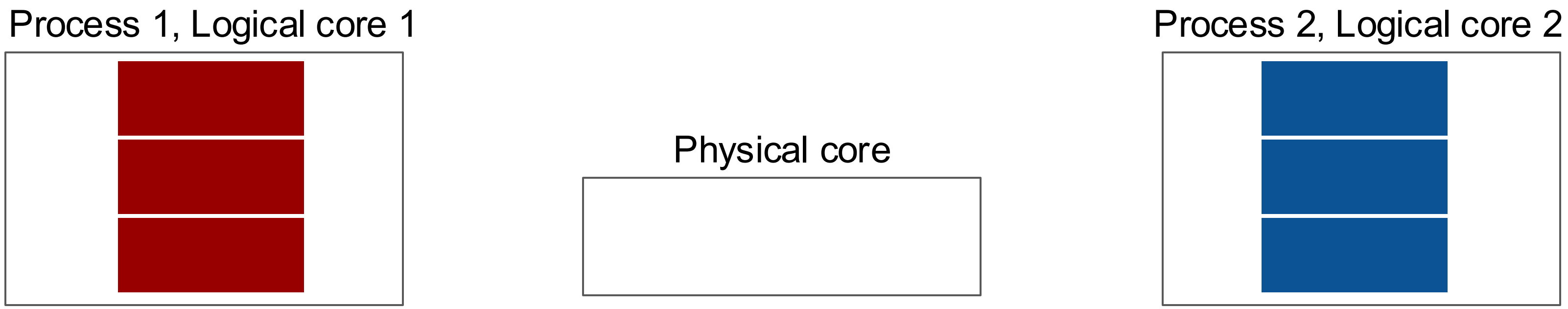
Multi-core CPUs and GPUs

- Multi-core CPUs have become the standard for most devices
  - These are CPUs with multiple physical processing units
    - Multiple ALUs, multiple control units, multiple register sets
  - GPUs are similar, except they have far more numerous, but simpler, cores
- These multi-core systems will be examined further in a future course (CSCI 4060U - Massively Parallel Programming)

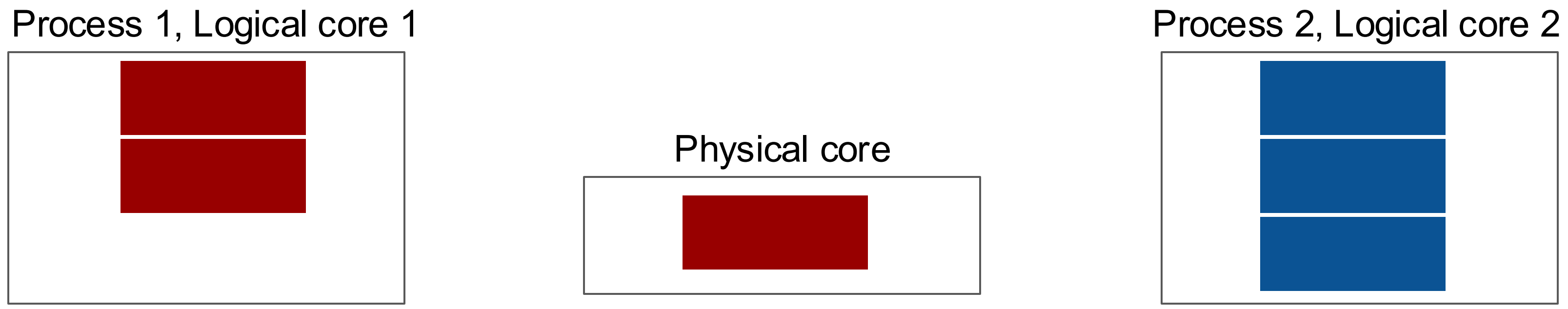
Hyper-threading

- As we’ve seen, adjacent instructions are often dependent
  - Instruction A modifies a value used by instruction B
  - It may not be possible for pipelining to pre-execute instruction B, since the values it needs are not yet ready
- Hyper-threading solves this problem by introducing k *logical cores*
  - Each physical core may be mapped to k logical cores (e.g. 2)
  - A logical core looks like a core to the operating system
  - The CPU may interleave instructions from separate processes/threads in the same core, since they are more likely to be independent

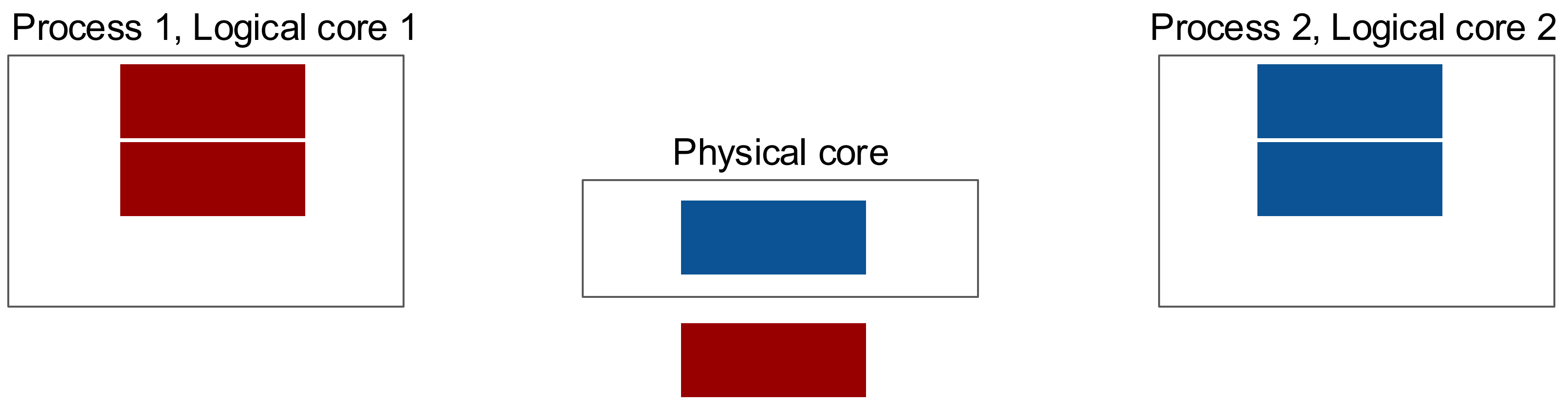
Hyper-threading



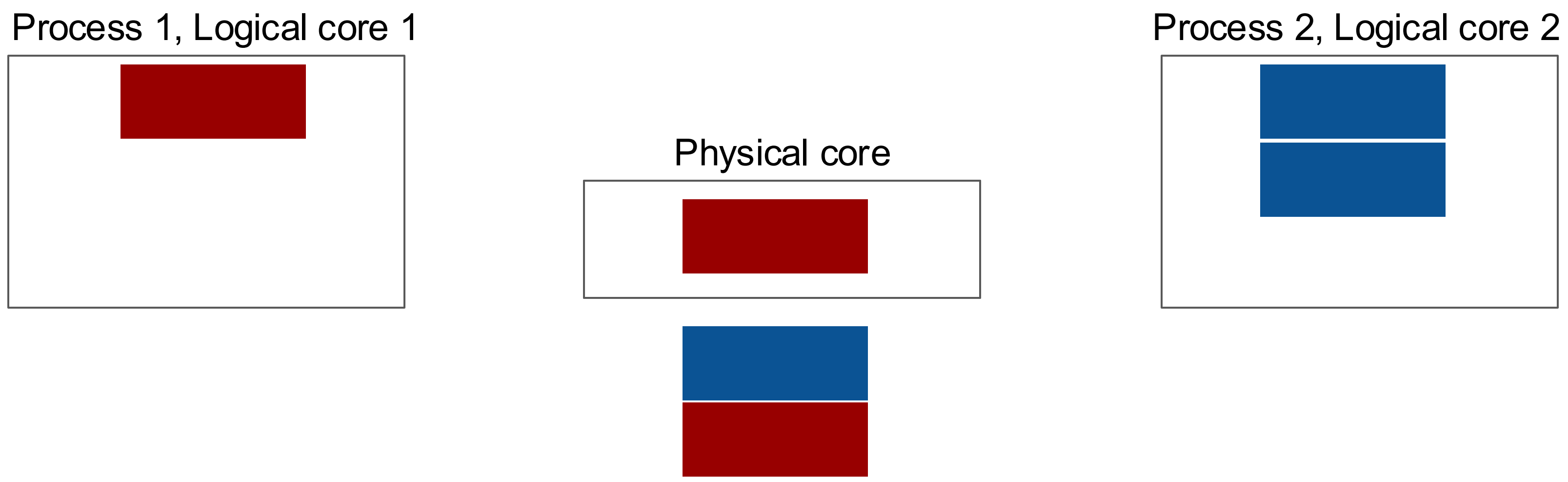
Hyper-threading



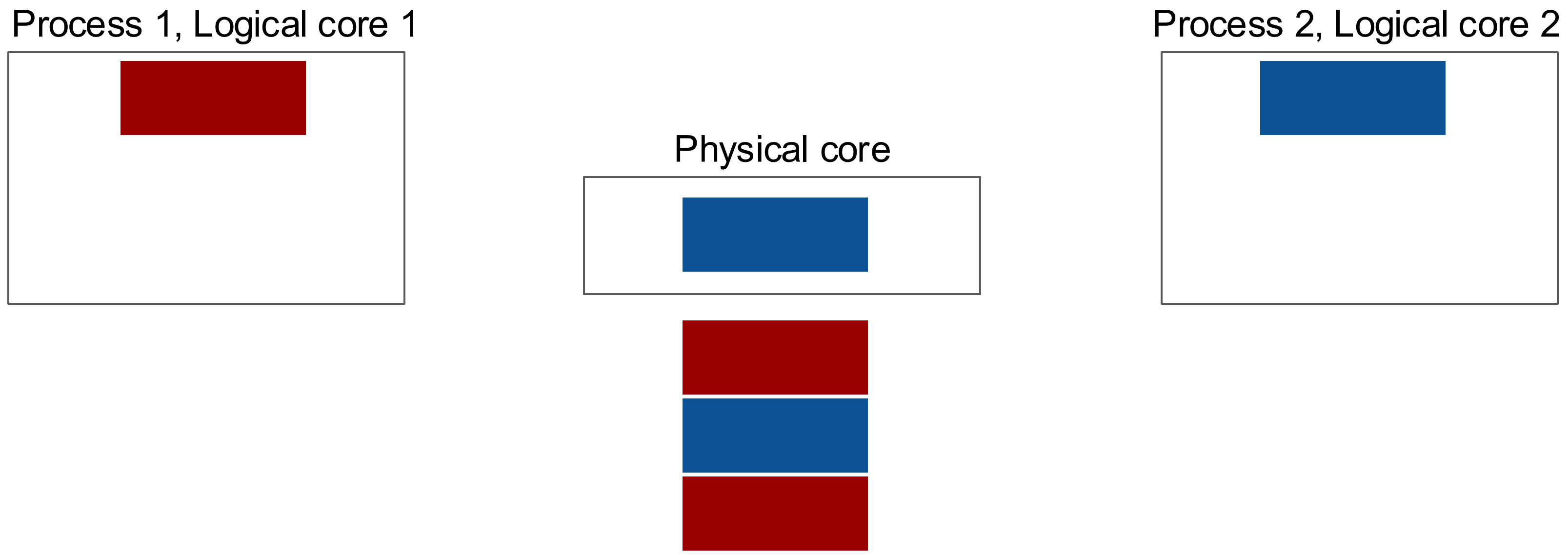
Hyper-threading



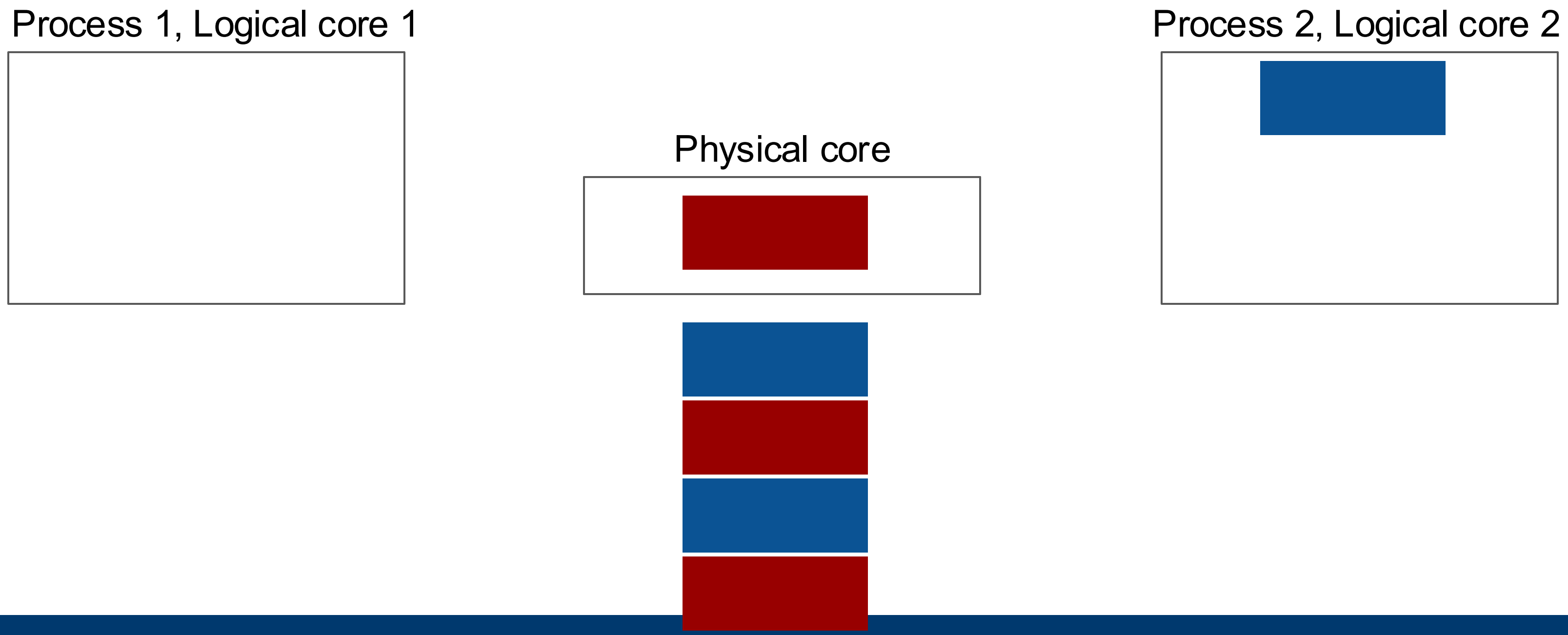
Hyper-threading



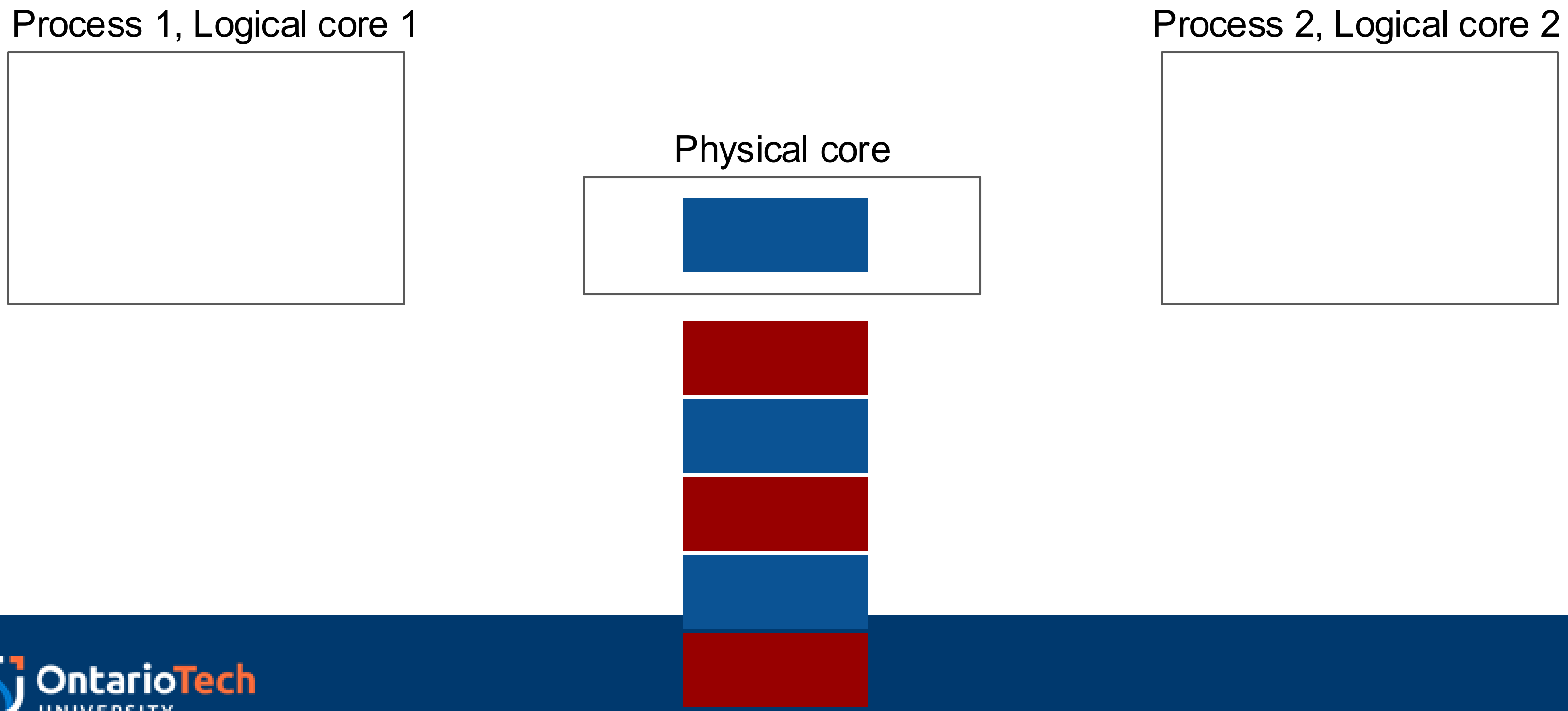
Hyper-threading



Hyper-threading



Hyper-threading



Wrap-Up

- Optimizations
    - Pipelining
    - Hyperthreading

What is next?

- Experiments with light
- Basic principles of quantum mechanics
  - Observer effect
  - Indeterminacy
  - Superposition
  - Entanglement
- Myths about quantum mechanics and quantum computing