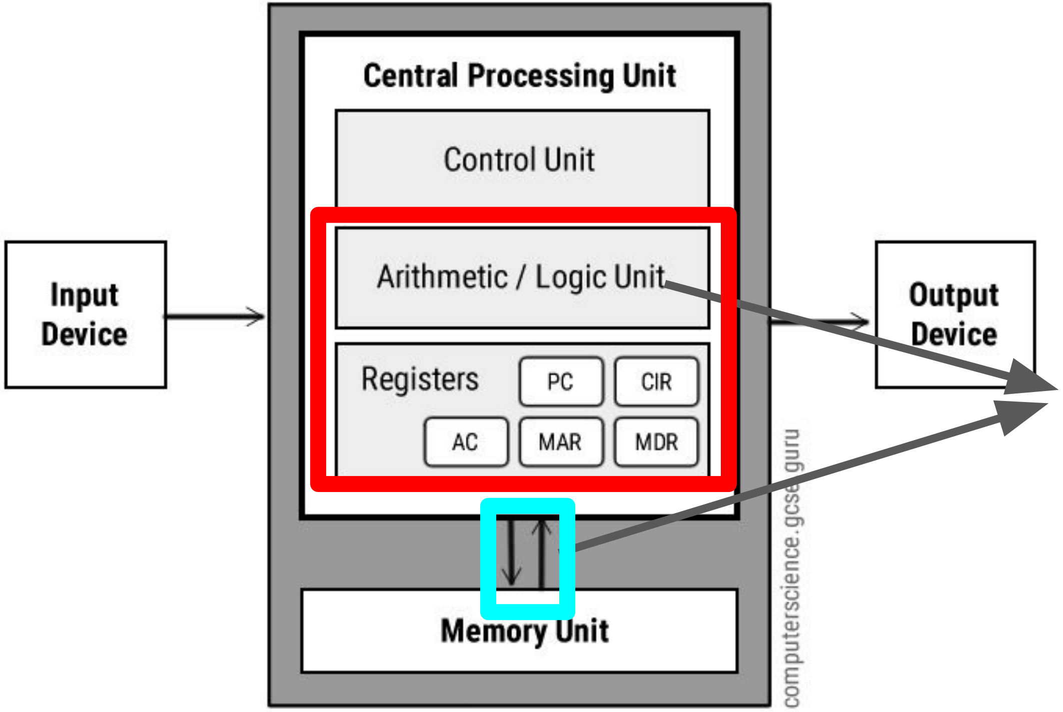
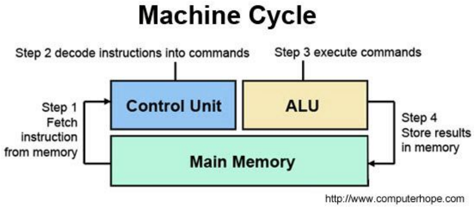
Hardware for parallelism: saturate all your pipelines

Hardware details: von Neumann architecture




Machine Cycle
Step 2 decode instructions into commands
Step 3 execute commands
Step 1 Fetch instruction from memory
Control Unit
ALU
Step 4 Store results in memory
Main Memory
http://www.computerhope.com

Computation and data retrieval are different physical hardware locations

Computation and communication can be overlapped!
CPUs: compiler **usually** takes care of this
GPUs: compilers **sometimes** do this

https://www.computerscience.gcse.guru/theory/von-neumann-architecture

Overlapping cycles to avoid stalls

Loop with stalls annotated

```
@label Loop
a = A[i]
# Stall
c = a + x
# Stall
# Stall
A[i] = c
a1 = A[i-1]
# Stall
c1 = a1 + x
# Stall
# Stall
A[i-1] = c1
a2 = A[i-2]
# Stall
c2 = a2 + x
# Stall
# Stall
A[i-2] = c2
a3 = A[i-3]
# Stall
c3 = a3 + x
# Stall
# Stall
A[i-3] = c3
i = i - 4
i > 4 && @goto Loop
```

The stalls are really like the dryer, folding cycles. The processors are not idle, but rather other functional units are processing, and the result is not ready.

Re-ordered loop

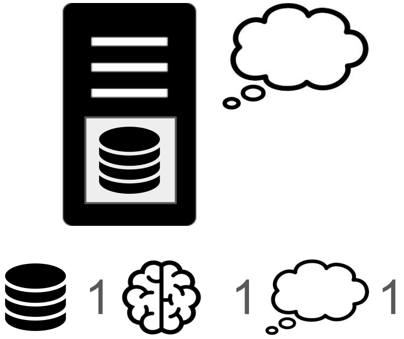
```
@label Loop
a  = A[i]
a1 = A[i-1]
a2 = A[i-2]
a3 = A[i-3]
c  = a  + x
c1 = a1 + x
c2 = a2 + x
c3 = a3 + x
A[i]   = c
A[i-1] = c1
A[i-2] = c2
A[i-3] = c3
i = i - 4
i > 4 && @goto Loop
```

- How many stalls? 0
- How many overhead cycles: 2
- How many cycles are actually work: 12

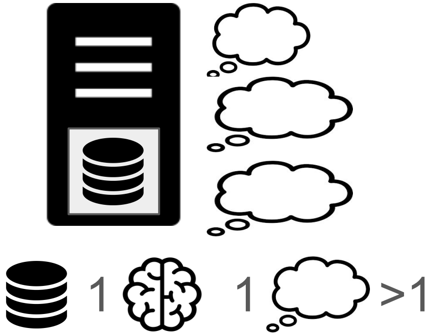
Source:
https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/pipelining/index.html

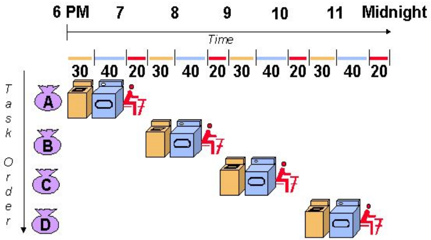
Multithreading is multitasking: another way to overlap hardware resource use/stalls

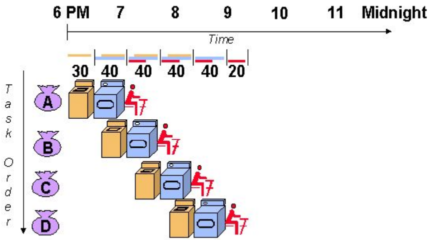
Single core



Hyperthreading





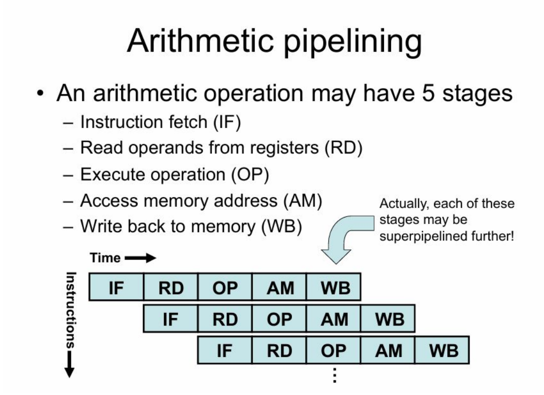


Most consumer laptops have 4 or 8 cores:
-4 or 8 threads is multi-threading
-8 or 16 threads is hyperthreading (2 threads per cores: multitasking)

If every thread is hyperoptimized and uses every hardware all the time (practically impossible on CPU) : hyperthreading would give no benefit (and even slow things down)
Practically: per-case basis

Pipelining would not work if instead of washers + dryers, you were only using a single machine!

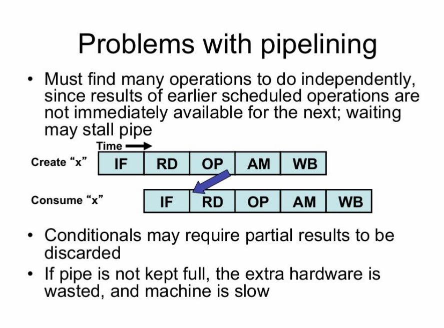
Pipelining: when hyperthreading can make sense



Arithmetic pipelining

- An arithmetic operation may have 5 stages
  - Instruction fetch (IF)
  - Read operands from registers (RD)
  - Execute operation (OP)
  - Access memory address (AM)
  - Write back to memory (WB)

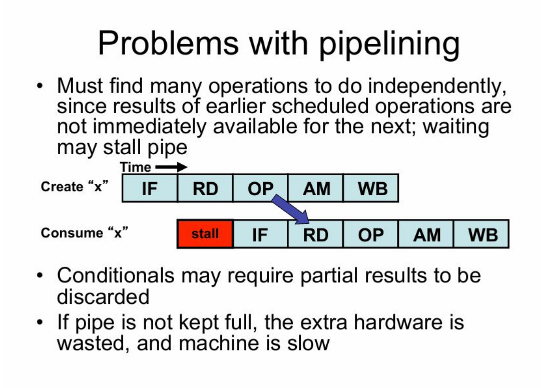
Actually, each of these stages may be superpipelined further!



Problems with pipelining

- Must find many operations to do independently, since results of earlier scheduled operations are not immediately available for the next; waiting may stall pipe

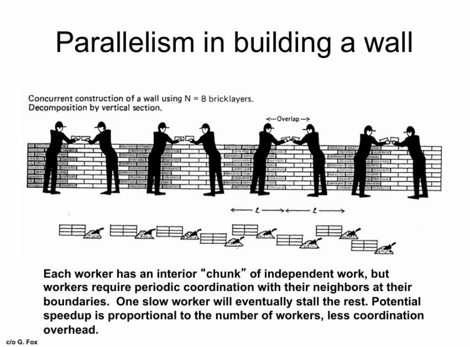
- Conditionals may require partial results to be discarded
- If pipe is not kept full, the extra hardware is wasted, and machine is slow



Problems with pipelining

- Must find many operations to do independently, since results of earlier scheduled operations are not immediately available for the next; waiting may stall pipe

- Conditionals may require partial results to be discarded
- If pipe is not kept full, the extra hardware is wasted, and machine is slow

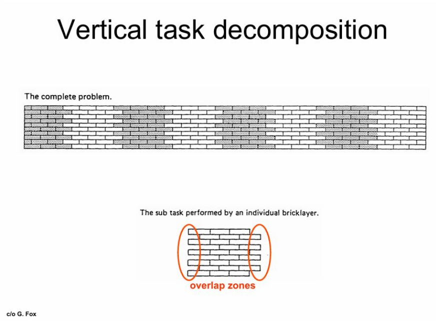


Parallelism in building a wall

Concurrent construction of a wall using N = 8 bricklayers.
Decomposition by vertical section.

Each worker has an interior "chunk" of independent work, but workers require periodic coordination with their neighbors at their boundaries. One slow worker will eventually stall the rest. Potential speedup is proportional to the number of workers, less coordination overhead.

c/o G. Fox

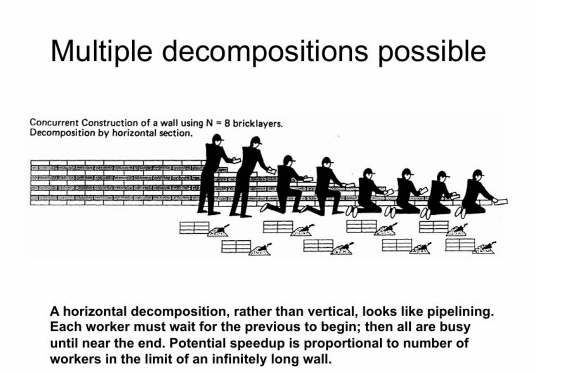


Vertical task decomposition

The complete problem.

The sub task performed by an individual bricklayer.

c/o G. Fox



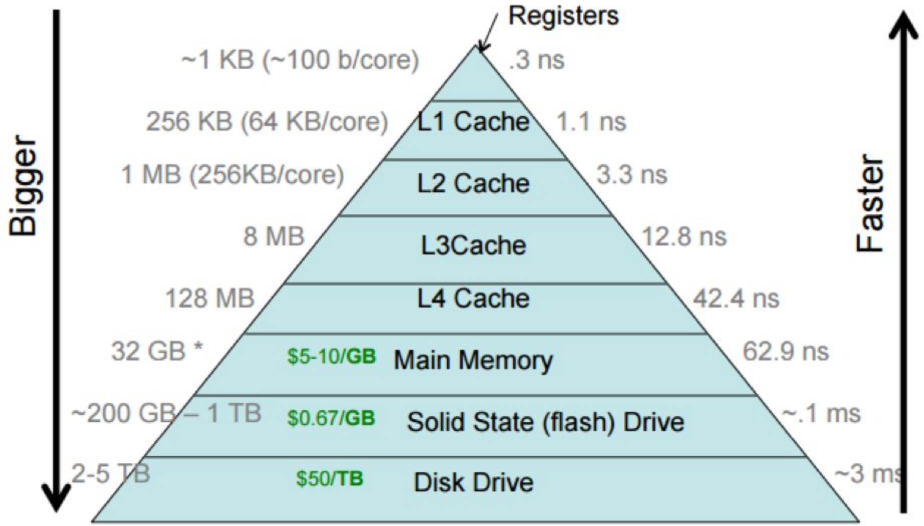
Multiple decompositions possible

Concurrent Construction of a wall using N = 8 bricklayers.
Decomposition by horizontal section.

A horizontal decomposition, rather than vertical, looks like pipelining. Each worker must wait for the previous to begin; then all are busy until near the end. Potential speedup is proportional to number of workers in the limit of an infinitely long wall.

Source: David Keyes, KAUST, Introduction to High Performance Computing
A Blue Waters Online Course Fall 2016, unit 1 part 1 slides

Cache levels and memory locations: the farther the data, the higher the latency

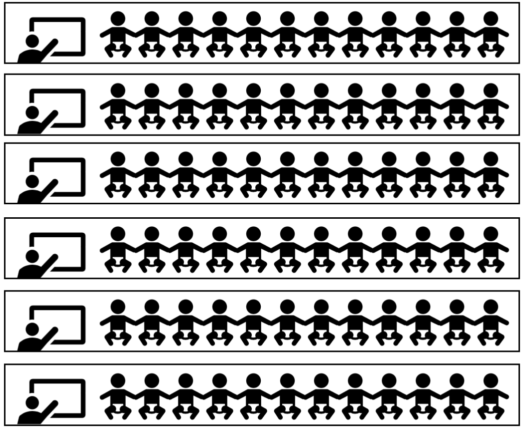


https://cs.brown.edu/courses/csci1310/2020/notes/l10.html

Types of processors: CPU vs GPU - MIT students vs kindergarten classes

CPU

GPU



- Kindergarten classes operate in groups of 32, directed by a teacher (large quantity)
- Grad students operate independently (large quality)
- Classes on the GPU need to receive an instruction from a grad student what to do!