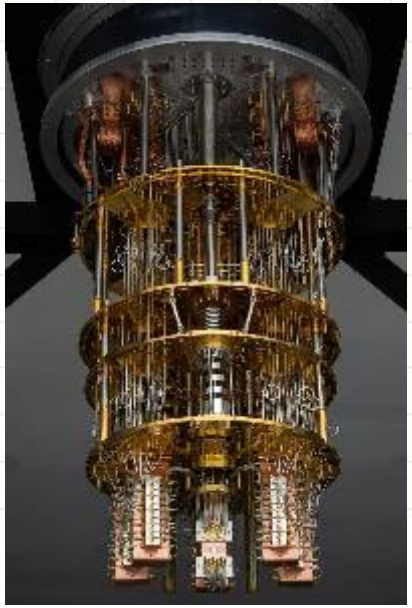
Architecture Concepts: Hardware Foundationsfor Modern Software

What is Computer Architecture?

- The science and art of designing computing platforms
  - hardware, interface, system SW, and programming model
- Objective
  - Achieve a set of design goals
    - Performance
    - Performance/watt
    - etc



The Computer Engineering Hierarchy

- Plenty of Room
  - Top: Leiserson et. al.
    - "There's plenty of room at the Top: What will drive computer performance after Moore's law?", Science, 2020
  - Bottom: Feynmann
    - "There's Plenty of Room at the Bottom: An Invitation to Enter a New Field of Physics", a lecture given at Caltech, 1959

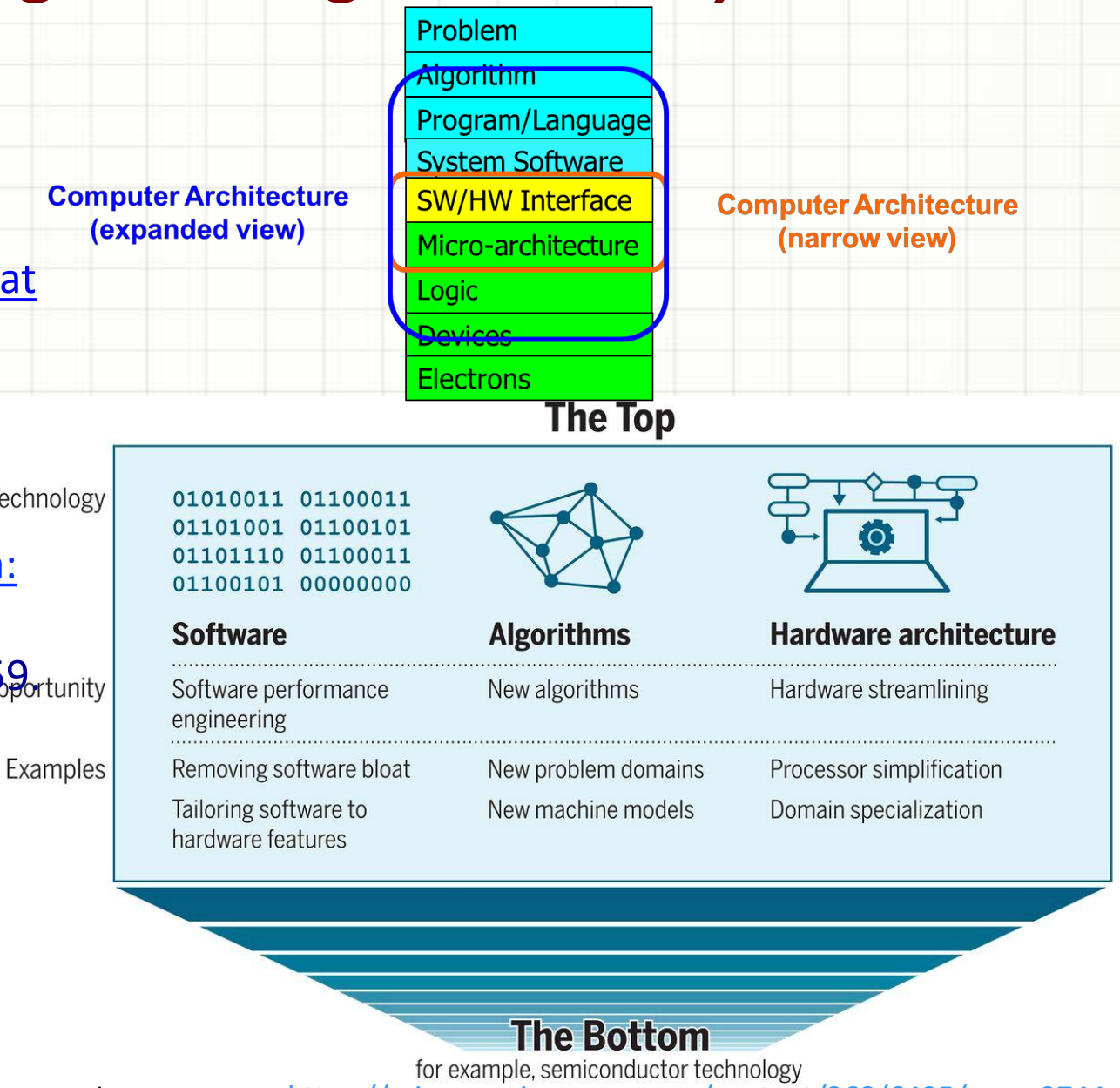
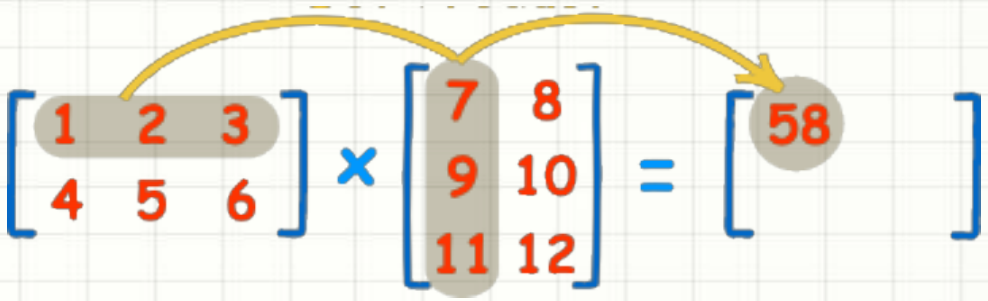


Image source: https://science.sciencemag.org/content/368/6495/eaam9744

Motivating Example

- Matrix Multiplication

```
for i in xrange(4096):
  for j in xrange(4096):
    for k in xrange(4096):
      C[i][j] += A[i][k] *
B[k][j]
```

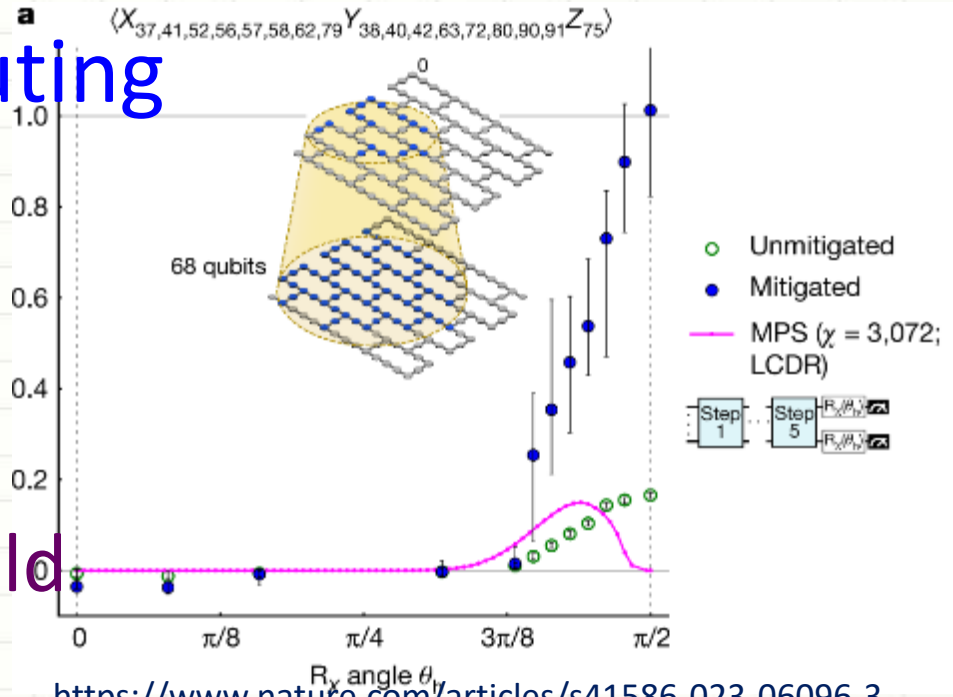


Implementation	Running time (s)	Absolute speedup
Python	25,552.48	1x
Java	2,372.68	11x
C	542.67	47x
Parallel loops	69.80	366x
Parallel divide and conquer	3.80	6,727x
plus vectorization	1.10	23,224x
plus AVX intrinsics	0.41	62,806x

Leiserson+, "There's plenty of room at the Top: What will drive computer performance after Moore's law?", Science, 2020

Motivating Example

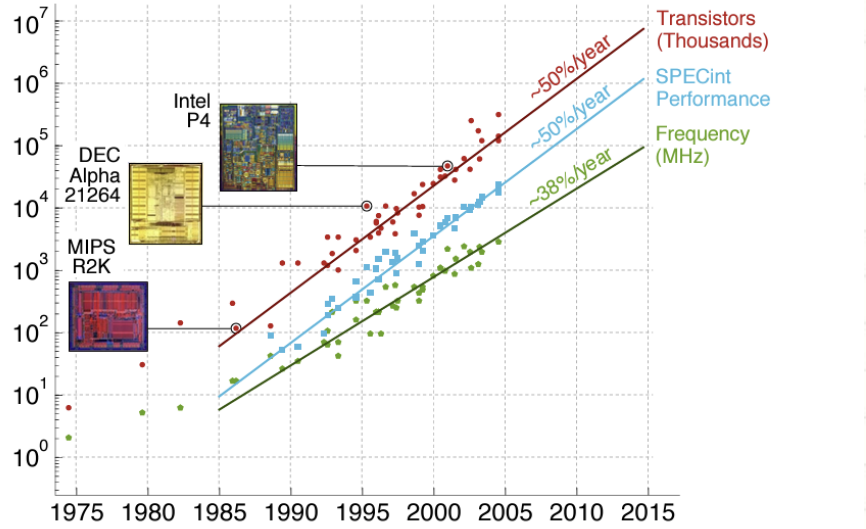
- Fault Tolerance in Quantum Computing
- Utility before FT
  - Evidence for the utility of quantum computing before fault tolerance
  - Time evolution of a 2D transverse-field Ising mode
    - 127 qubit system
    - Demonstrated that a quantum computer can outperform a classical computer.



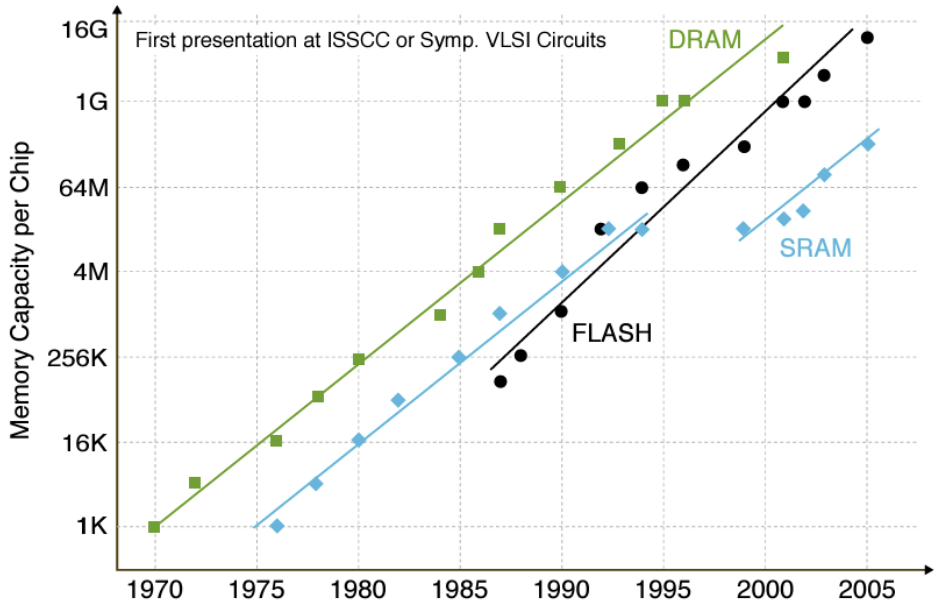
https://www.nature.com/articles/s41586-023-06096-3

Indian Institute of Technology Delhi

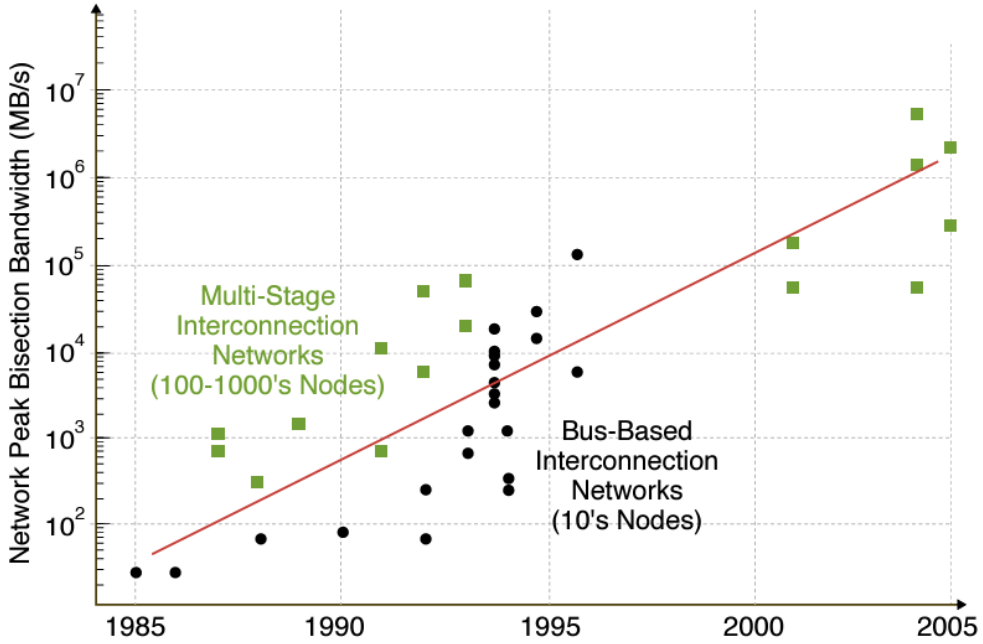
Scaling: Processor, Memory and Network



Data collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, C. Batten



Adapted from K. Itoh et al. "Ultra-Low Voltage Nano-Scale Memories." Spring 2007.

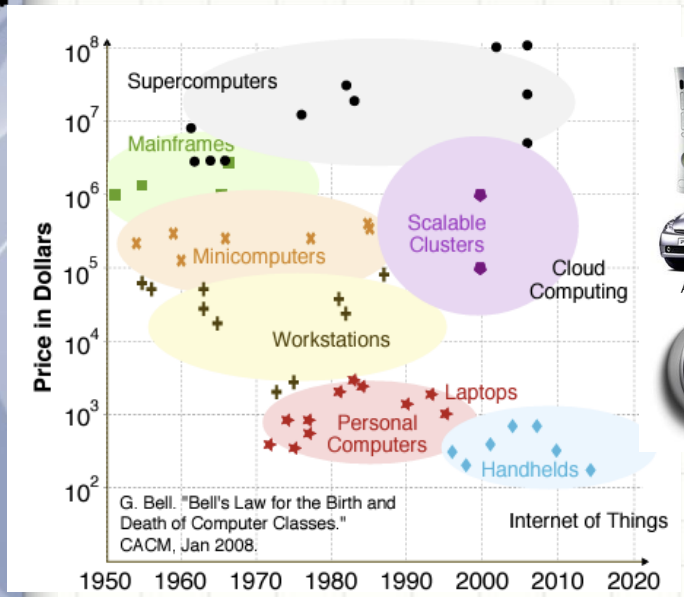


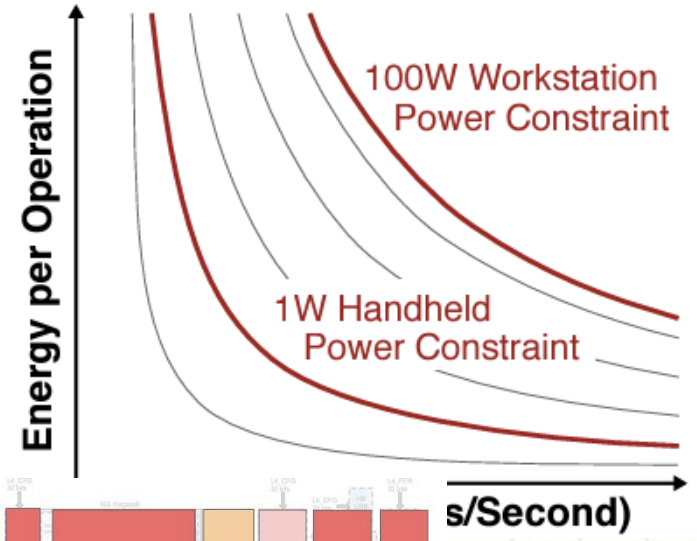
Data from Hennessy & Patterson, Morgan Kaufmann, 2nd & 5th eds., 1996 & 2011; D.E. Culler et al., Morgan Kaufmann, 1999.

What drives CA today

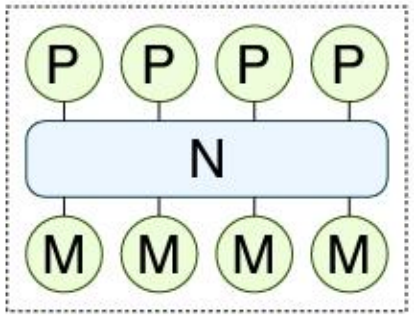
- Diversity in applications
  - Cloud and IoT
- Energy & power constrained systems
- Multiple cores
- Heterogeneous systems-on-chip
- Technology scaling challenges
  - New emerging compute, storage, and communication device technologies

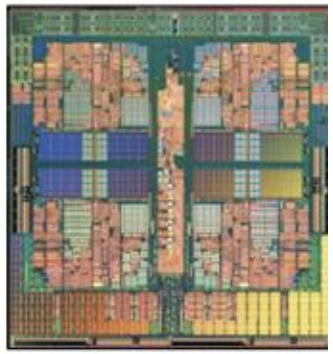
Some Trends

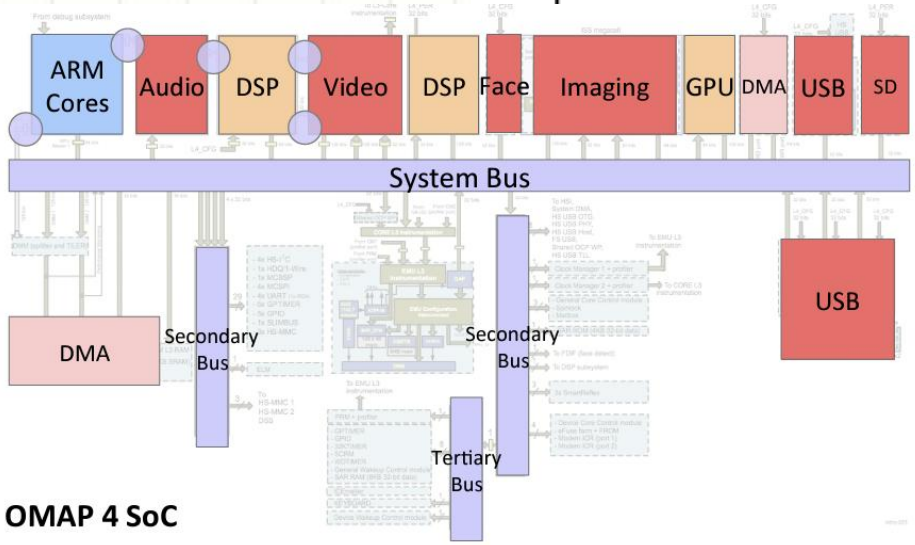


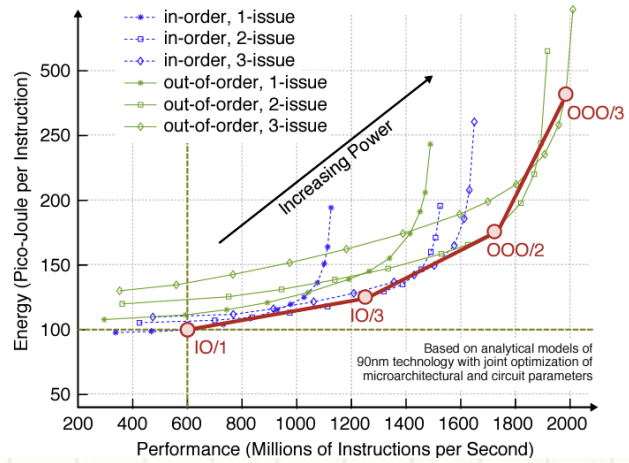


Vertical MOSFETs
Graphene
Carbon Nanotubes
Nanorelays
Quantum Computing
Molecular Computing
Memristers
Phase-Change Mem
Spintronics
3D Integration
Nanophotonics





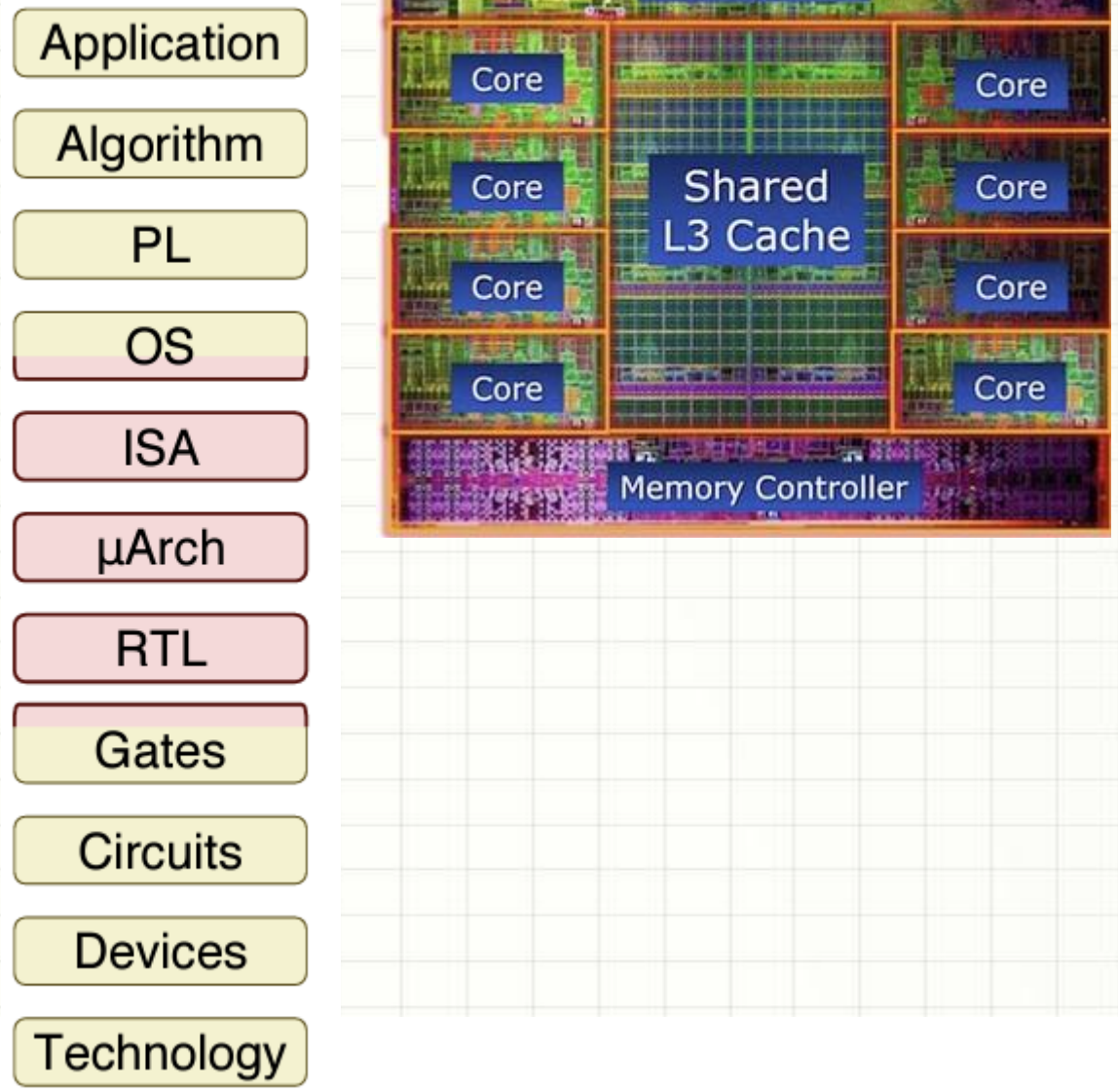




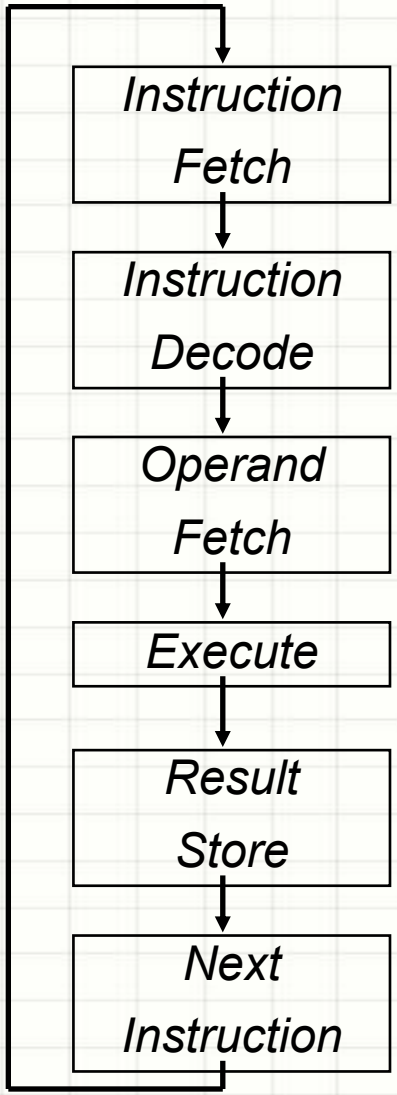
There is plenty of room both at the top and at the bottom
but **much more so**
communicate well between and optimize across
the top and the bottom

Incredibly Complex

- The Design of a Modern Processor
  - Fighter Plane
    - 10 k Parts
  - Intel Sandy Bridge E
    - 2.27 Billion Transistors
- Design Philosophy
  - Modularity
  - Hierarchy
  - Encapsulation
  - Regularity
  - Extensibility
- Design Patterns
  - Control/Datapath split



Fundamental Execution Cycle



Obtain instruction from program storage

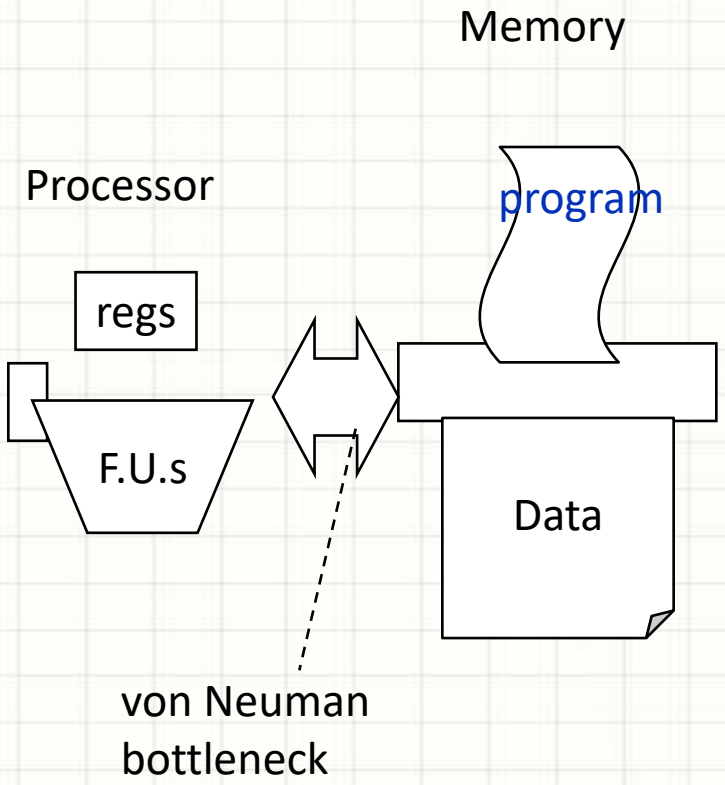
Determine required actions and instruction size

Locate and obtain operand data

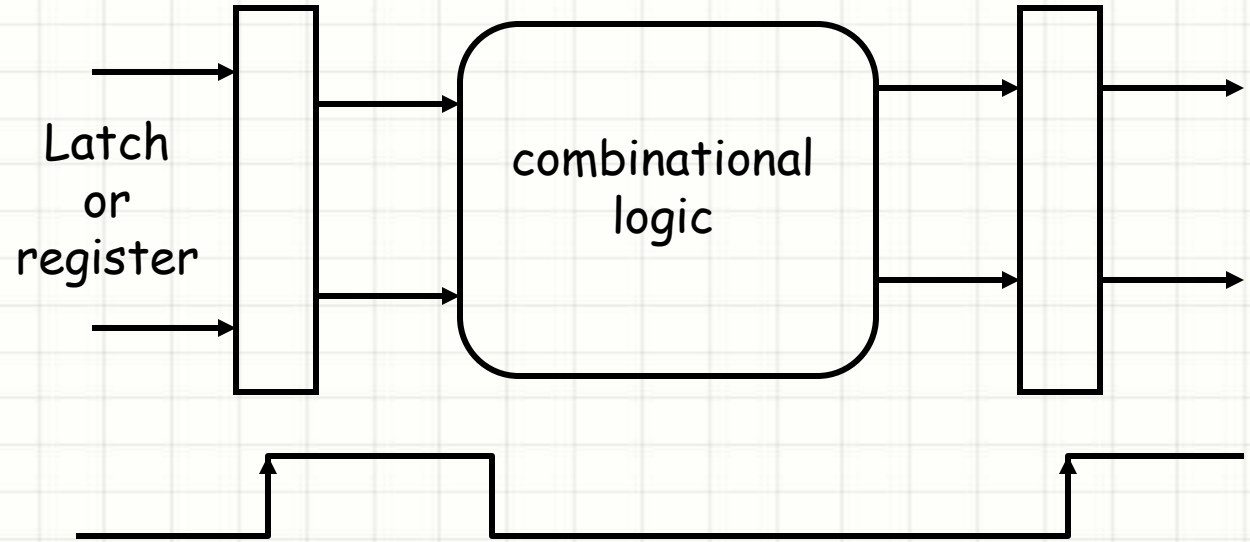
Compute result value or status

Deposit results in storage for later use

Determine successor instruction



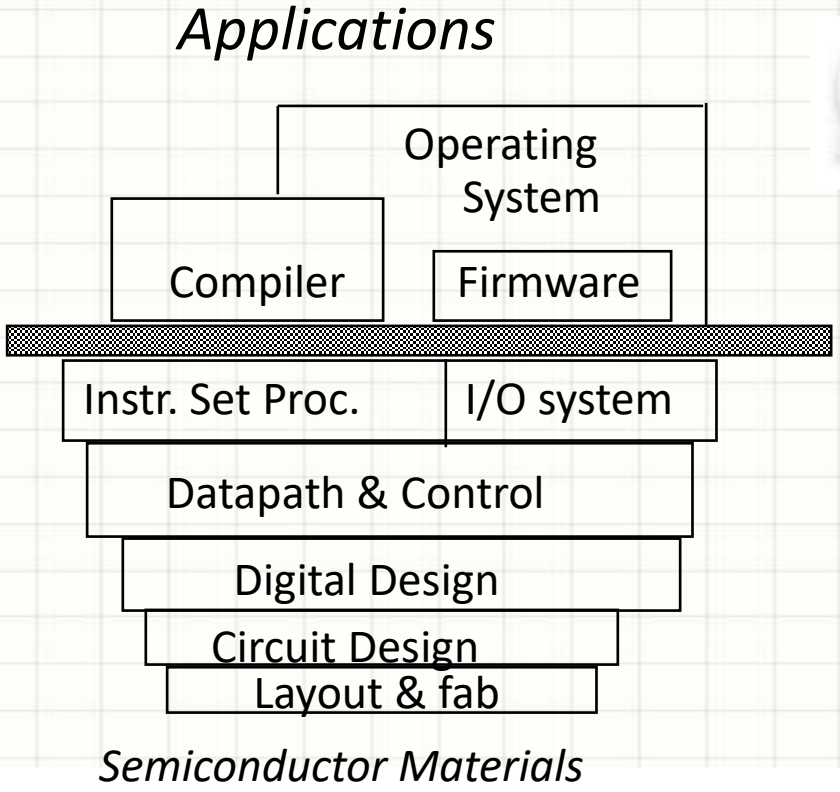
What's a Clock Cycle?



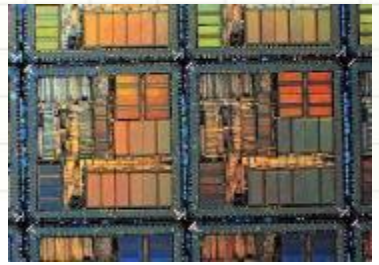
- 10 levels of gates
  - clock propagation, wire lengths, drivers

What is "Computer Architecture"?

- Coordination of many *levels of abstraction*
- Under a rapidly *changing set of forces*
- Design, Measurement, *and* Evaluation

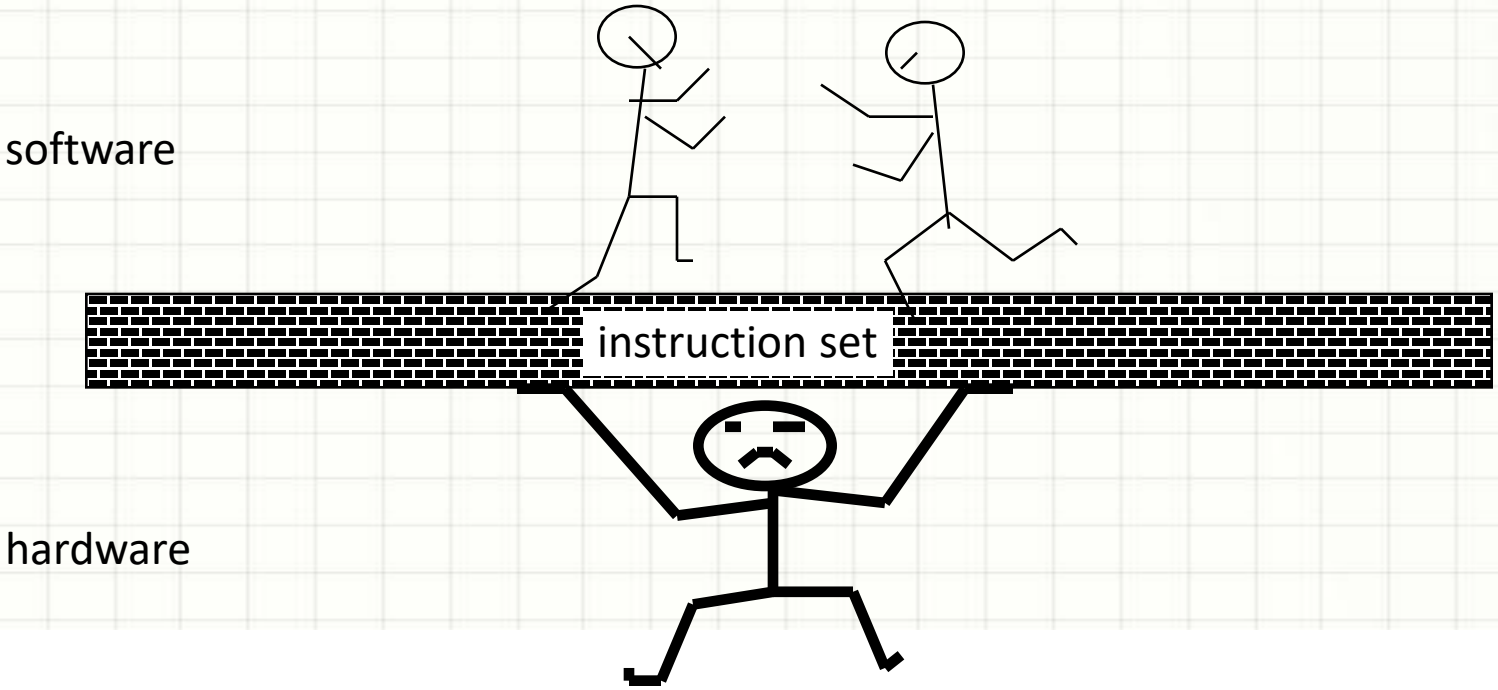


Instruction Set Architecture

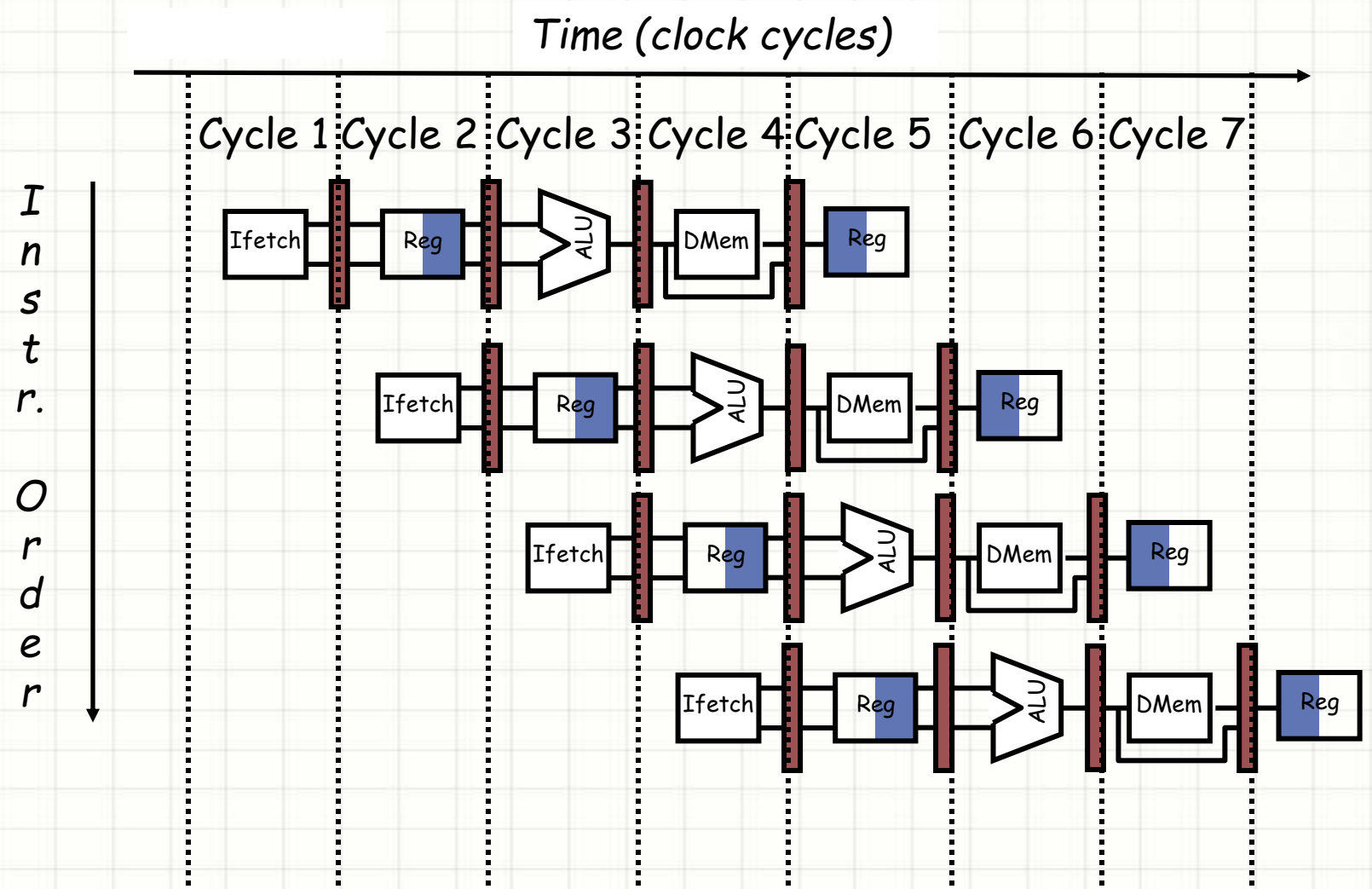


The Instruction Set: a Critical Interface

- Properties of a good abstraction
  - Lasts through many generations (portability)
  - Used in many different ways (generality)
  - Provides convenient functionality to higher levels
  - Permits an efficient implementation at lower levels

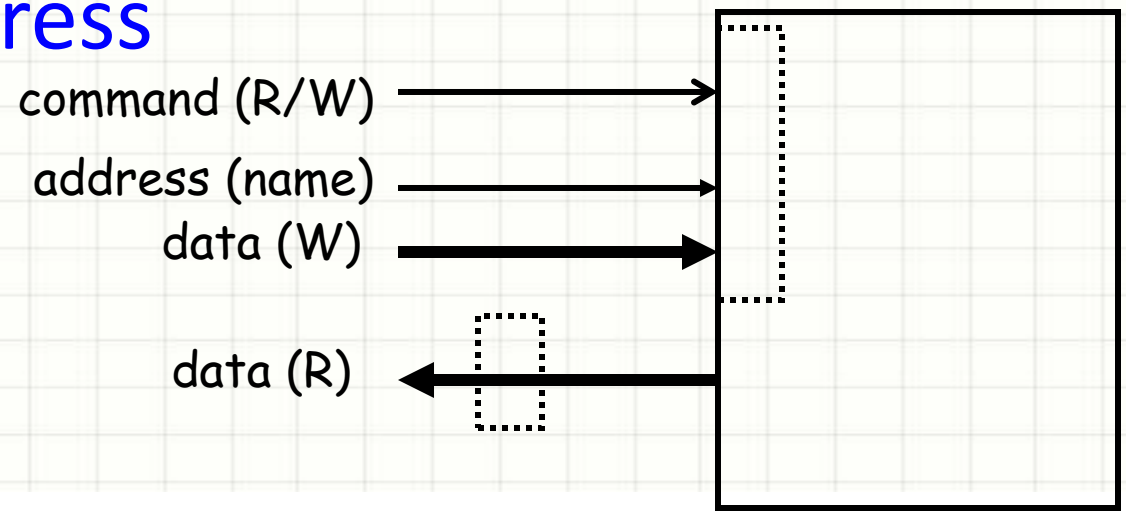


Pipelined Instruction Execution

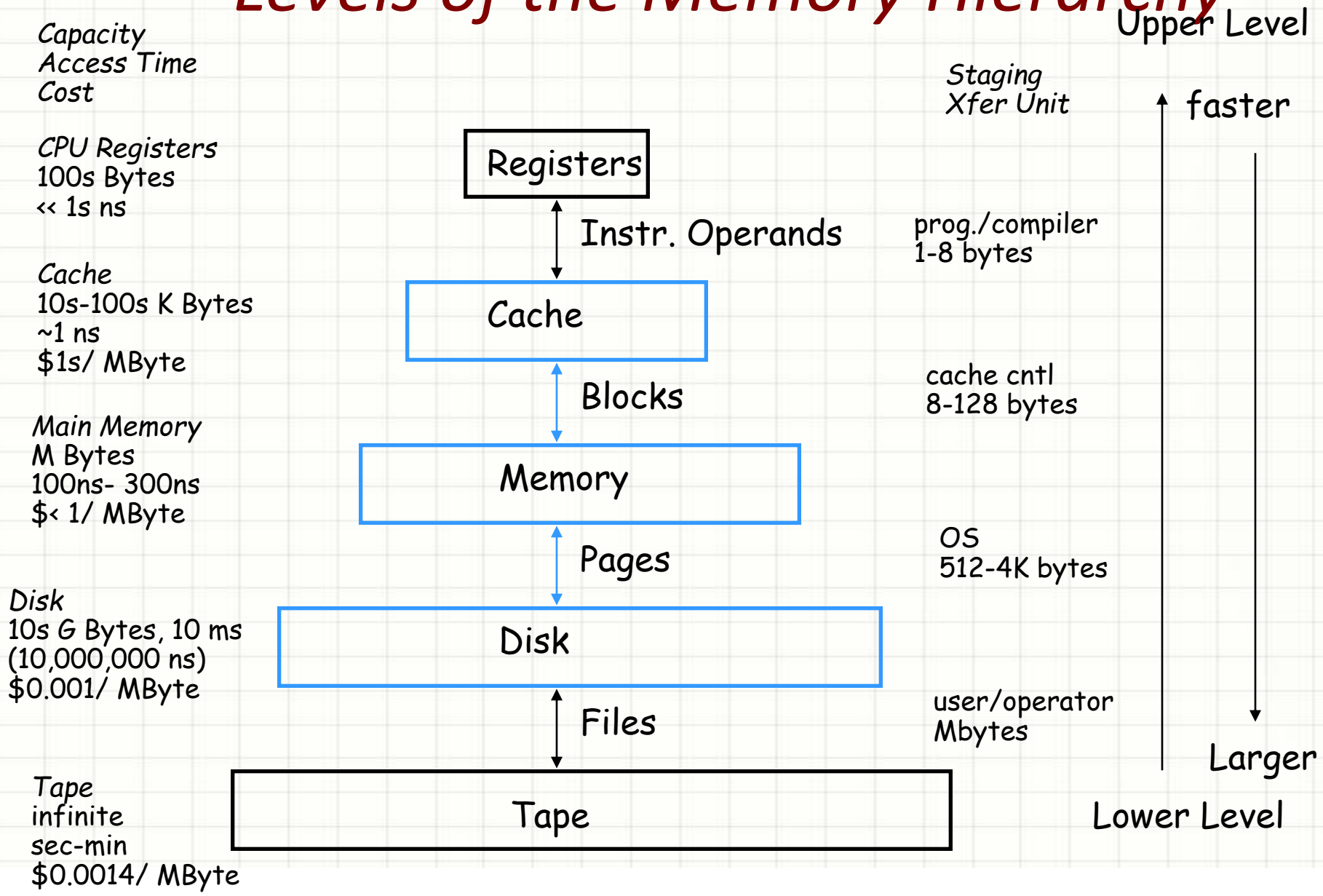


The Memory Abstraction

- Association of <name, value> pairs
  - typically named as byte addresses
  - often values aligned on multiples of size
- Sequence of Reads and Writes
- Write binds a value to an address
- Read of addr returns most recently written value bound to that address

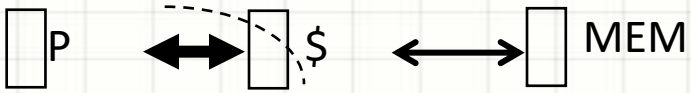


Levels of the Memory Hierarchy



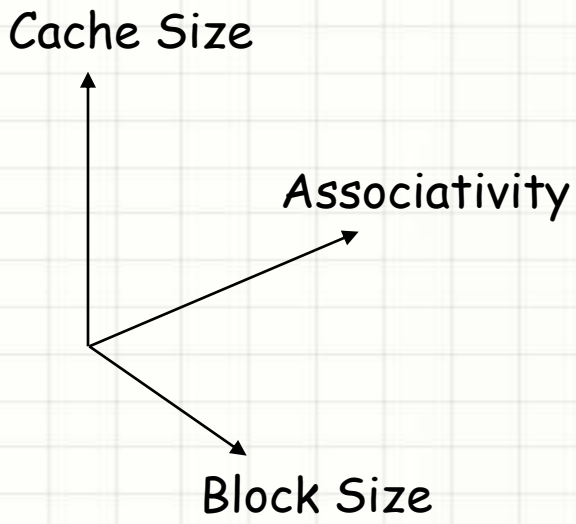
The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 30 years, HW relied on locality for speed

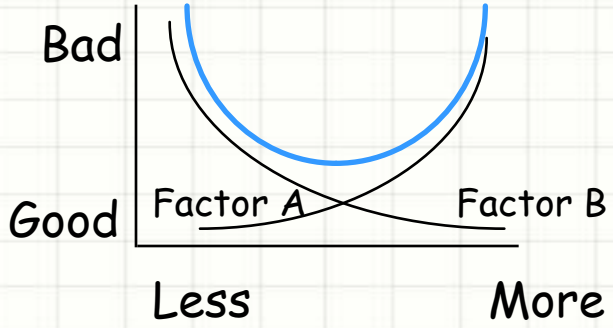


The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back

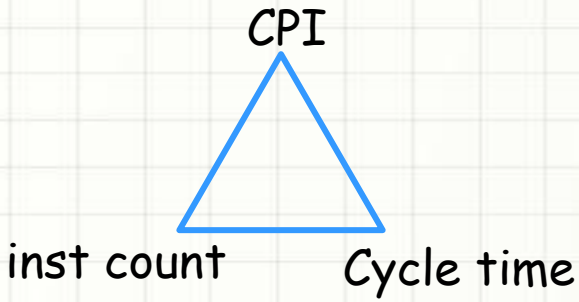


- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins



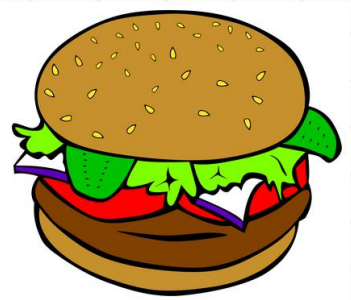
Processor performance equation

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$



	Inst Count	CPI	Clock Rate
Program	**X**		
Compiler	**X**	**(X)**	
Inst. Set.	**X**	**X**	
Organization		**X**	**X**
Technology			**X**

So, what does performance depend on …



- **#instructions in the program**
  - Depends on the compiler
- **Frequency**
  - Depends on the transistor technology and the architecture
    - If we have more pipeline stages, then the time to traverse each stage reduces roughly proportionally
    - Given that each stage needs to be processed in one clock cycle, smaller the stage, higher the frequency
    - To increase the frequency, we simply need to increase the number of pipeline stages
- **IPC**
  - Depends on the architecture and the compiler
  - A large part of this book is devoted to this aspect.

How to improve performance?

- There are **3** factors:
  - IPC, #instructions, and frequency
  - #instructions is dependent on the compiler → not on the architecture
- Let us look at IPC and frequency
- IPC
  - What is the IPC of an in-order pipeline?

1 if there are no stalls, otherwise < 1

Methods to increase IPC

- Forwarding
- Having more not-taken branches in the code
- Faster instruction and data memories