CS654 Advanced Computer Architecture

Lec 2 - Introduction

Peter Kemper

Adapted from the slides of EECS 252 by Prof. David Patterson
Electrical Engineering and Computer Sciences
University of California, Berkeley

Outline

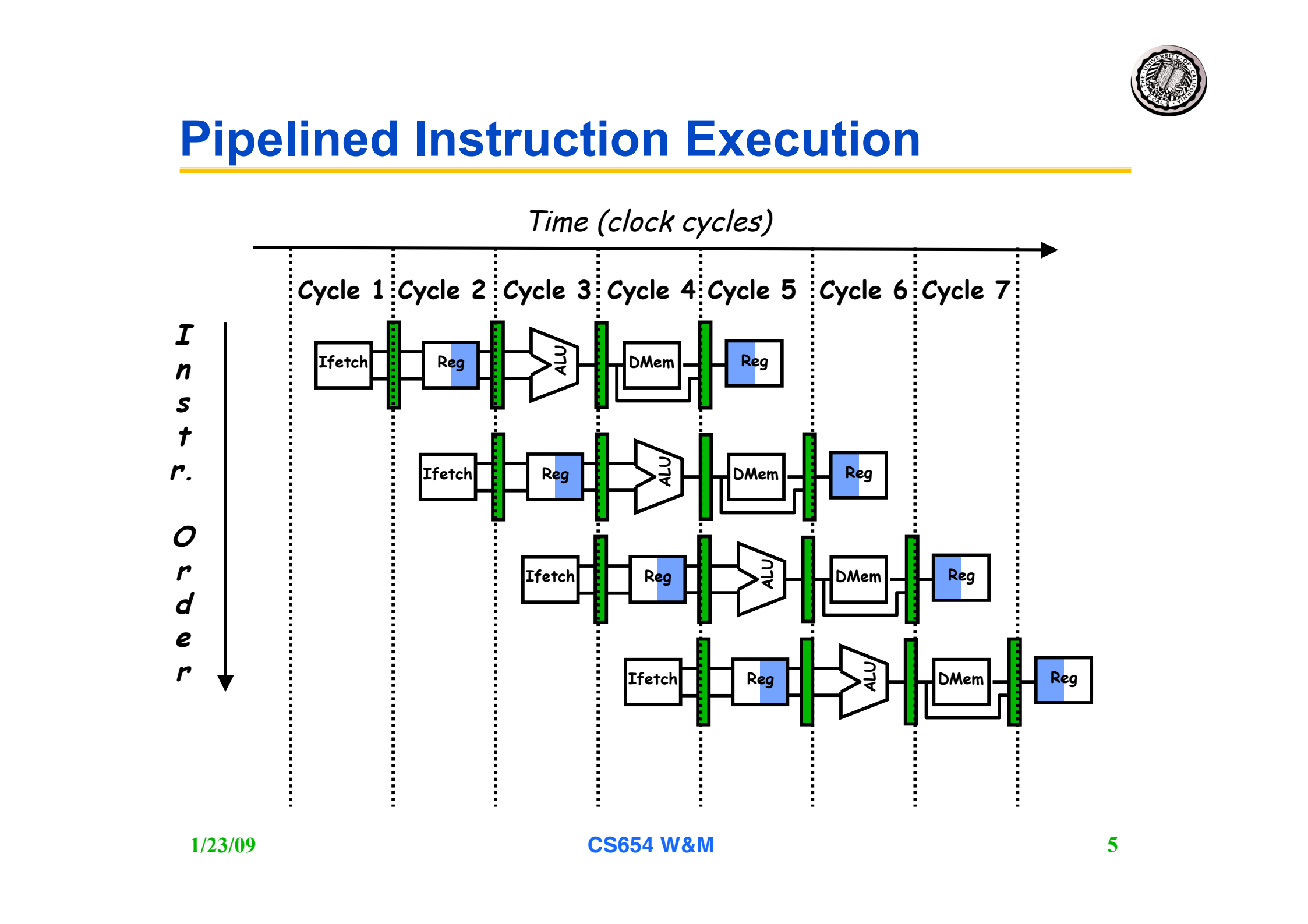
- Computer Science at a Crossroads
- Computer Architecture v. Instruction Set Arch.
- **What Computer Architecture brings to table**
- **Technology Trends**

What Computer Architecture brings to Table

- Other fields often borrow ideas from architecture
- Quantitative Principles of Design
  1. Take Advantage of Parallelism
  2. Principle of Locality
  3. Focus on the Common Case
  4. Amdahl's Law
  5. The Processor Performance Equation
- Careful, quantitative comparisons
  - Define, quantify, and summarize relative performance
  - Define and quantify relative cost
  - Define and quantify dependability
  - Define and quantify power
- Culture of anticipating and exploiting advances in technology
- Culture of well-defined interfaces that are carefully implemented and thoroughly checked

1) Taking Advantage of Parallelism

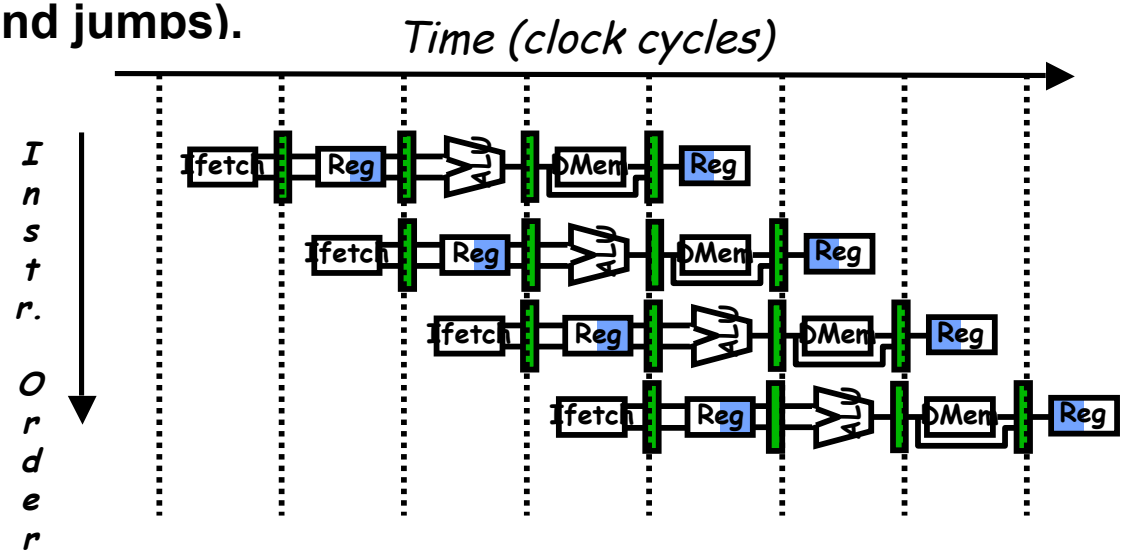
- Increasing throughput of server computer via multiple processors or multiple disks
- Detailed HW design
  - Carry lookahead adders uses parallelism to speed up computing sums from linear to logarithmic in number of bits per operand
  - Multiple memory banks searched in parallel in set-associative caches
- Pipelining: overlap instruction execution to reduce the total time to complete an instruction sequence.
  - Not every instruction depends on immediate predecessor ⇒ executing instructions completely/partially in parallel possible
  - Classic 5-stage pipeline:
    1) Instruction Fetch (Ifetch),
    2) Register Read (Reg),
    3) Execute (ALU),
    4) Data Memory Access (Dmem),
    5) Register Write (Reg)



Pipelined Instruction Execution

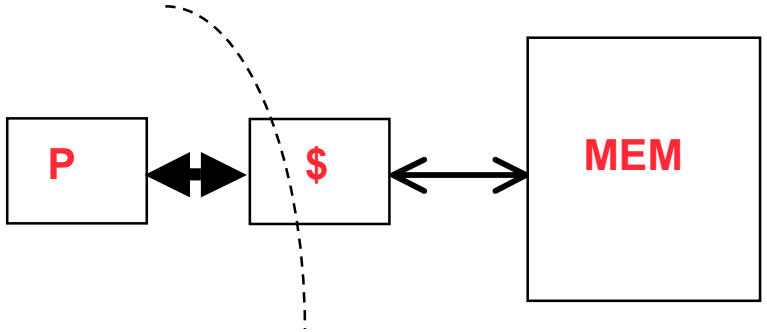
Limits to pipelining

- **Hazards prevent next instruction from executing during its designated clock cycle**
  - **Structural hazards: attempt to use the same hardware to do two different things at once**
  - **Data hazards: Instruction depends on result of prior instruction still in the pipeline**
  - **Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).**



2) The Principle of Locality

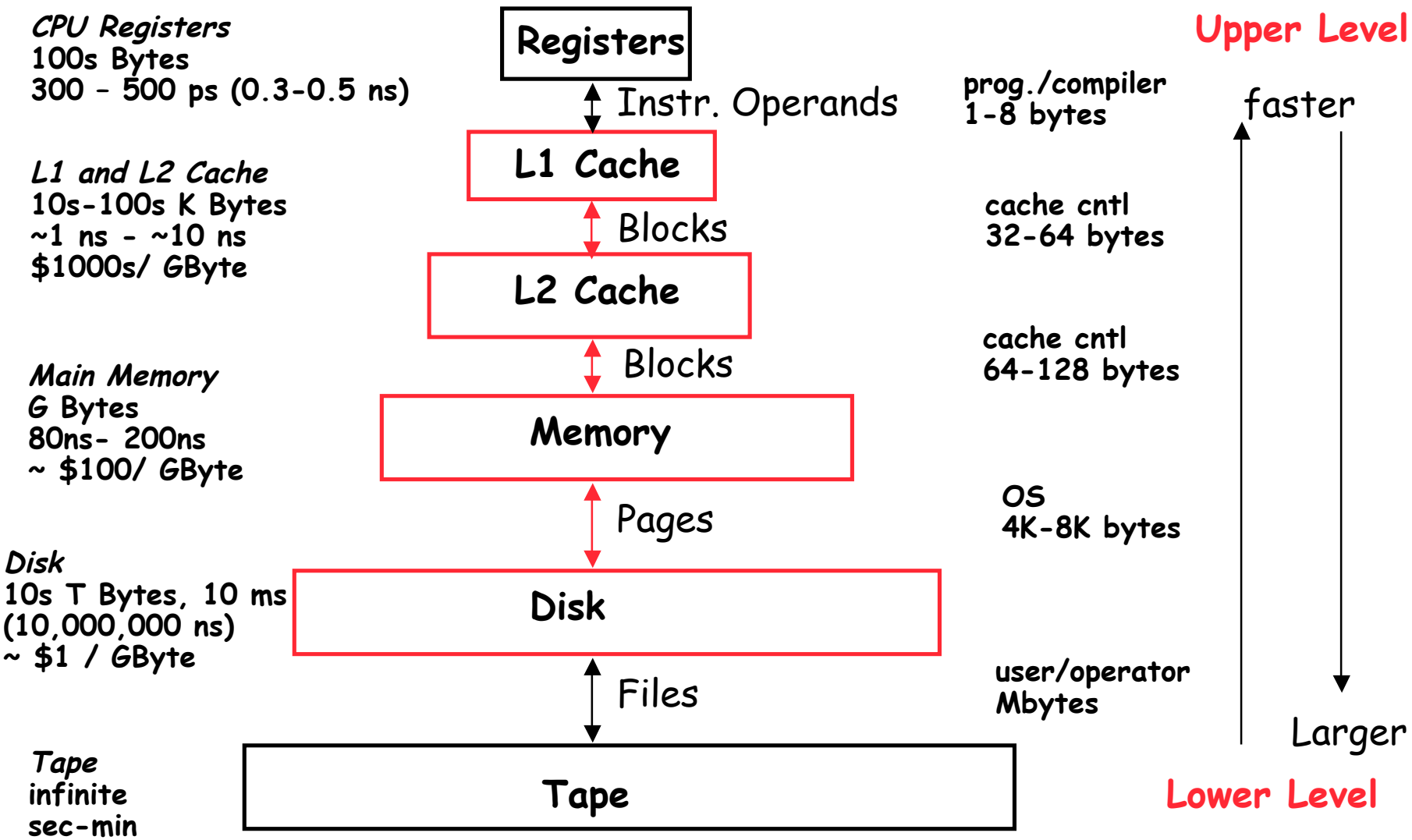
- **The Principle of Locality:**
  - **Program access a relatively small portion of the address space at any instant of time.**
- **Two Different Types of Locality:**
  - **Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)**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



Levels of the Memory Hierarchy

Capacity
Access Time
Cost

Staging
Xfer Unit



CPU Registers
100s Bytes
300 – 500 ps (0.3-0.5 ns)

prog./compiler
1-8 bytes

L1 and L2 Cache
10s-100s K Bytes
~1 ns - ~10 ns
$1000s/ GByte

cache cntl
32-64 bytes

cache cntl
64-128 bytes

Main Memory
G Bytes
80ns- 200ns
~ $100/ GByte

OS
4K-8K bytes

Disk
10s T Bytes, 10 ms
(10,000,000 ns)
~ $1 / GByte

user/operator
Mbytes

Tape
infinite
sec-min
~$1 / GByte

3) Focus on the Common Case

- Common sense guides computer design
  - Since it's engineering, common sense is valuable
- In making a design trade-off, favor the frequent case over the infrequent case
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
  - E.g., If database server has 50 disks / processor, storage dependability dominates system dependability, so optimize it 1st
- Frequent case is often simpler and can be done faster than the infrequent case
  - E.g., overflow is rare when adding 2 numbers, so improve performance by optimizing more common case of no overflow
  - May slow down overflow, but overall performance improved by optimizing for the normal case
- What is frequent case and how much performance improved by making case faster => Amdahl's Law

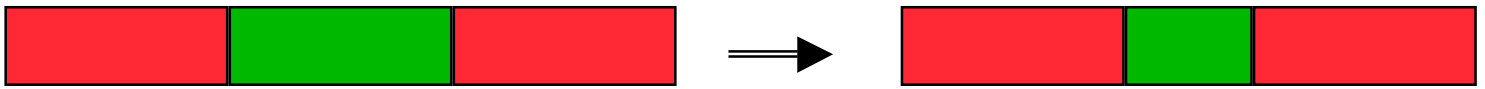
4) Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$



Amdahl's Law example

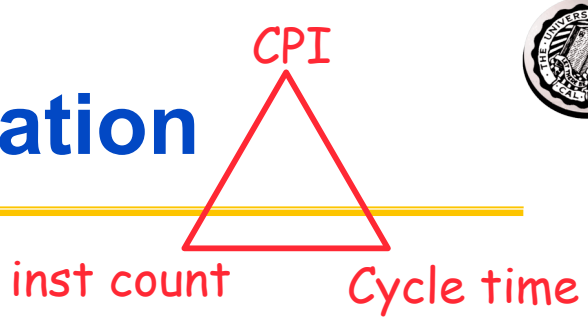
- **New CPU 10X faster**
- **I/O bound server, so 60% time waiting for I/O**

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{\left(1 - 0.4\right) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- **Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster**

5) 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

At this point …

- Computer Architecture >> instruction sets
- Computer Architecture skill sets are different
  - 5 Quantitative principles of design
  - Quantitative approach to design
  - Solid interfaces that really work
  - Technology tracking and anticipation
- Computer Science at the crossroads from sequential to parallel computing
  - Salvation requires innovation in many fields, including computer architecture
- However for CS654, we have to go through the state of the art first:
  - Material:

    read Chapter 1, then Appendix A in Hennessy/Patterson