

## Performance and Cost

- Hennessy Patterson chapter 1-

Politecnico di Milano

v0

Francesco Peverelli <[francesco.peverelli@polimi.it](mailto:francesco.peverelli@polimi.it)>  
Marco D. Santambrogio <[marco.santambrogio@polimi.it](mailto:marco.santambrogio@polimi.it)>  
Politecnico di Milano

# Outline

- Measures to evaluate performance
- Quantifying the design process
- Benchmarking
- Energy/Power
- Cost

# Nowadays...

- Tegra 2
  - Dual-Core ARM Cortex-A9
  - ULP GeForce, 8 cores
- A6
  - Dual-Core based on ARMv7
  - Triple-core PowerVR SGX 543MP3 GPU
- Tegra 3
  - Quad-Core
  - ULP GeForce, 12 cores



ASUS Eee Pad Slider Tablet



Motorola Photon 4G



iPhone 5



HTC One X

# Different Classes of Computers... ... Different needs

- Personal Mobile Device (PMD)
  - Emphasis on energy efficiency and real-time
- Desktop Computing
  - Emphasis on price-performance
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters / Warehouse Scale Computers - Used for “SaaS”
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- Embedded Computers
  - Emphasis: price

# Issues as new opportunities

- Programming has become very difficult
  - Impossible to balance all constraints manually



# Issues as new opportunities

- Programming has become very difficult
  - Impossible to balance all constraints manually
- More computational horse-power than ever before
  - Cores are free

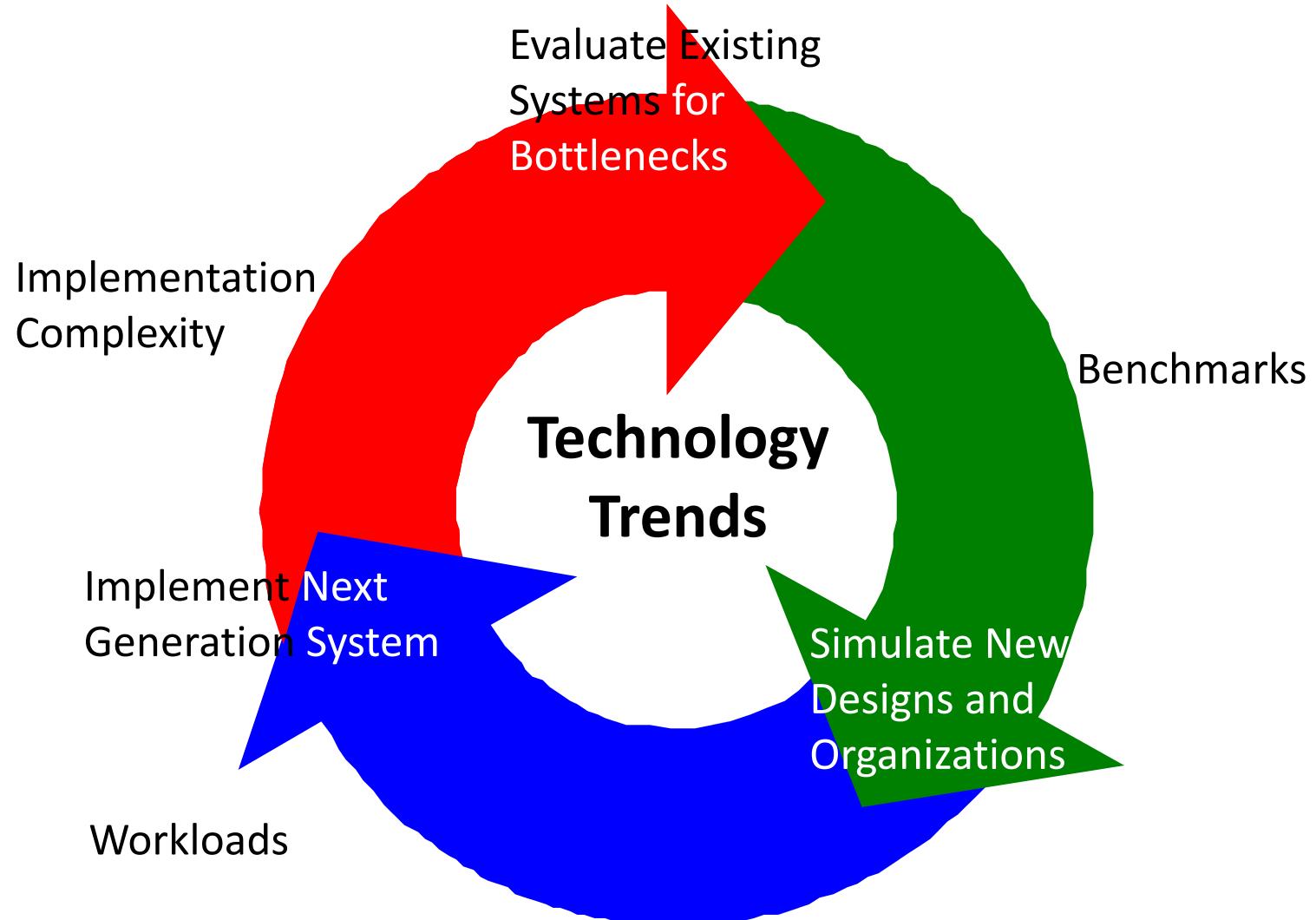


# Issues as new opportunities

- Programming has become very difficult
  - Impossible to balance all constraints manually
- More computational horse-power than ever before
  - Cores are free
- Energy is new constraint
  - Software must become energy and space aware



# Computer Engineering Methodology



# Performance Evaluation

- When we say that one computer is faster than another what do we mean?

# Performance Evaluation

- When we say that one computer is faster than another what do we mean?
  - It depends on what is important

# Performance Evaluation

- When we say that one computer is faster than another what do we mean?
  - It depends on what is important
- Two Metrics:

# Performance Evaluation

- When we say that one computer is faster than another what do we mean?
  - It depends on what is important
- Two Metrics:
  - Computer system user
    - Minimize elapsed time for program execution:  
**response time**:  $\text{execution time} = \text{time\_end} - \text{time\_start}$

# Performance Evaluation

- When we say that one computer is faster than another what do we mean?
  - It depends on what is important
- Two Metrics:
  - Computer system user
    - Minimize elapsed time for program execution:  
**response time**:  $\text{execution time} = \text{time\_end} - \text{time\_start}$
  - Computer center manager
    - Maximize completion rate = #jobs/sec  
**throughput**: total amount of work done in a given time

# Response time vs throughput

- Is throughput = 1/average response time?

# Response time vs throughput

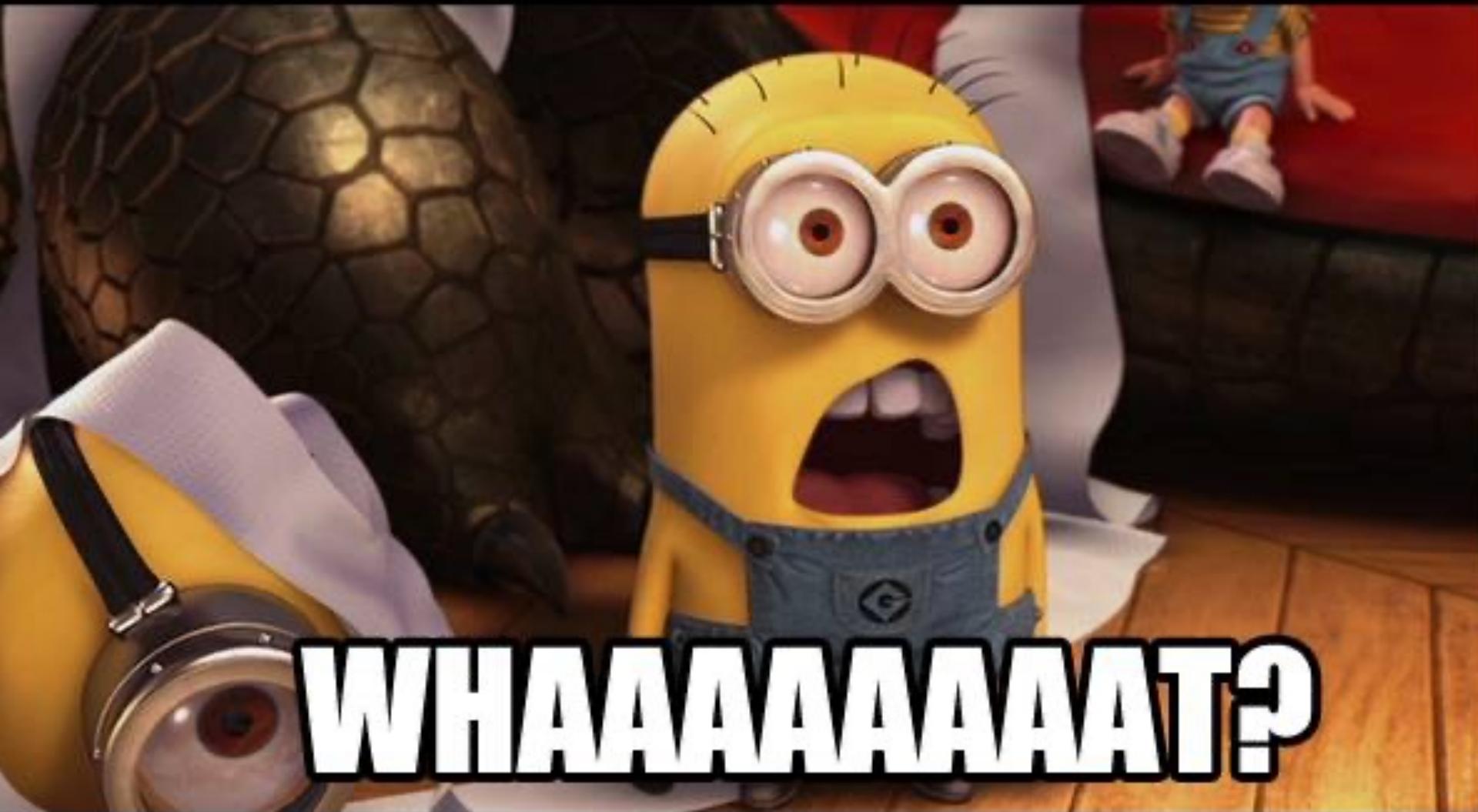
- Is throughput =  $1/\text{average response time}$ ?
  - YES only if NO overlap
  - Otherwise throughput  $> 1/\text{average response time}$

# Response time vs throughput

- Is throughput =  $1/\text{average response time}$ ?
  - YES only if NO overlap
  - Otherwise throughput  $> 1/\text{average response time}$
  - Example:
    - A lunch buffet with 5 stations
    - Each person takes 2 minutes at each station
    - Time per person to fill up the tray is 10 minutes

# Response time vs throughput

- Is throughput =  $1/\text{average response time}$ ?
  - YES only if NO overlap
  - Otherwise throughput  $> 1/\text{average response time}$
  - Example:
    - A lunch buffet with 5 stations
    - Each person takes 2 minutes at each station
    - Time per person to fill up the tray is 10 minutes
    - BUT throughput is 1 person every 2 minutes



**WHAaaaaaat?**

# Response time vs throughput



Time: 0

Station 1

Station 2

Station 3

Station 4

Station 5

2'

2'

2'

2'

2'

# Response time vs throughput



Time: 0



2'

2'

2'

2'

2'

# Response time vs throughput



Time: 2'



Station 3

Station 4

Station 5

2'

2'

2'

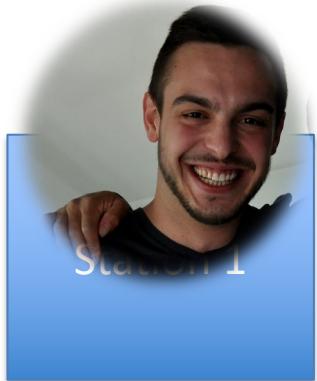
2'

2'

# Response time vs throughput



Time: 4'



2'



2'



2'



2'



2'

# Response time vs throughput



Time: 6'



2'

2'

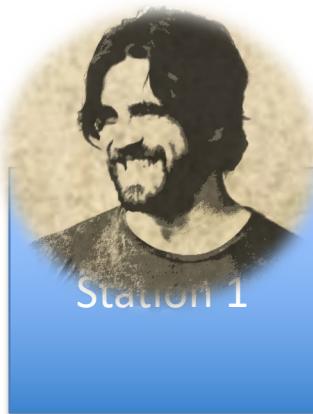
2'

2'

2'

# Response time vs throughput

Time: 8'



2'

2'

2'

2'

2'

# Response time vs throughput

Time: 10'



2'

2'

2'

2'

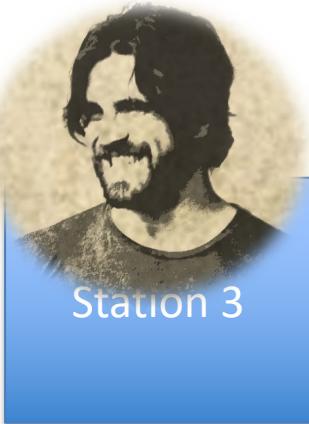
2'

Served:



# Response time vs throughput

Time: 12'



2'

2'

2'

2'

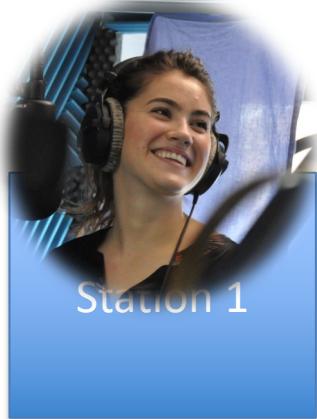
2'

Served:



# Response time vs throughput

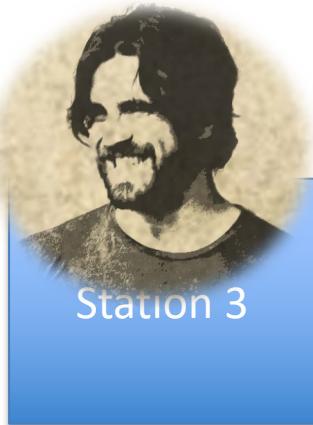
10'



Station 1



Station 2



Station 3



Station 4



Station 5

2'

2'

2'

2'

2'

## Served:



# Response time vs throughput

10'

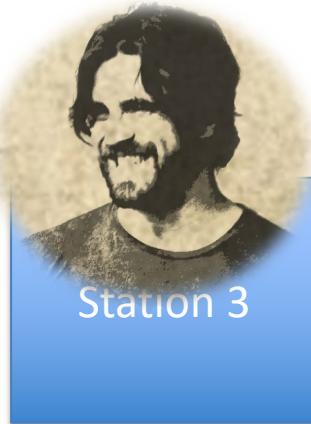
2'



Station 1



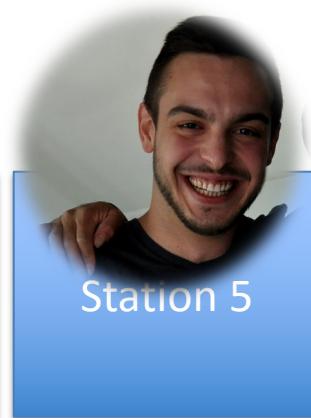
Station 2



Station 3



Station 4



Station 5

2'

2'

2'

2'

2'

Served:



# Response time vs throughput

- Is throughput =  $1/\text{average response time}$ ?
  - YES only if NO overlap
  - Otherwise throughput  $> 1/\text{average response time}$
  - Example:
    - A lunch buffet with 5 stations
    - Each person takes 2 minutes at each station
    - Time per person to fill up the tray is 10 minutes
    - BUT throughput is 1 person every 2 minutes
    - Overlap: 5 people simultaneously filling the tray
    - Without overlap throughput =  $1/10$

# Overview of Factors Affecting Performance

- Algorithm complexity and data set
- Compiler
- Instruction set
- Available operations
- Operating system
- Clock rate
- Memory system performance
- I/O system performance and overhead

# The Bottom Line: Performance (and Cost)

Plane	DC to Paris
Boeing 747	6.5 hours
BAD/Sud Concorde	3 hours

Time to run the task (ExTime)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns ... (Performance)

- Throughput, bandwidth

# The Bottom Line: Performance (and Cost)

Plane	DC to Paris	Speed
Boeing 747	6.5 hours	610 mph
BAD/Sud Concorde	3 hours	1350 mph

Time to run the task (ExTime)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns ... (Performance)

- Throughput, bandwidth

# The Bottom Line: Performance (and Cost)

Plane	DC to Paris	Speed	Passengers
Boeing 747	6.5 hours	610 mph	470
BAD/Sud Concorde	3 hours	1350 mph	132

Time to run the task (ExTime)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns ... (Performance)

- Throughput, bandwidth

# The Bottom Line: Performance (and Cost)

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAD/Sud Concorde	3 hours	1350 mph	132	178,200

Time to run the task (ExTime)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns ... (Performance)

- Throughput, bandwidth

# The Bottom Line: Performance (and Cost)

## **WHAT ABOUT TICKETS PRICES?**

286,700

178,200

Time to run the task (ExTime)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns ... (Performance)

- Throughput, bandwidth

# The Bottom Line: Performance (and Cost)

**WHAT ABOUT TICKETS PRICES?**

**BOEING 100\$ each**

**CONCORD 200\$ each**

286,700

178,200

Time to run the task (ExTime)

- Execution time, response time, latency

Tasks per day, hour, week, sec, ns ... (Performance)

- Throughput, bandwidth

# The Bottom Line: Performance

"X is n times faster than Y" means

$$\frac{\text{ExTime}(Y)}{\text{ExTime}(X)} = \frac{\text{Performance}(X)}{\text{Performance}(Y)}$$

**Speed of Concorde vs. Boeing 747**

$$1350/610 = 2.2$$

**Throughput of Boeing 747 vs. Concorde**

$$286700/178200 = 1.6$$

# Speedup

“X is n% faster than Y”  $\Rightarrow \frac{\text{execution time (y)}}{\text{execution time (x)}} = 1 + \frac{n}{100}$

$$\text{performance}(x) = \frac{1}{\text{execution_time}(x)}$$

“X is n% faster than Y”  $\Rightarrow \frac{\text{performance}(x)}{\text{performance}(y)} = 1 + \frac{n}{100}$

# Speedup

“X is n% faster than Y”  $\Rightarrow \frac{\text{execution time (y)}}{\text{execution time (x)}} = 1 + \frac{n}{100}$

$$\text{performance}(x) = \frac{1}{\text{execution_time}(x)}$$

“X is n% faster than Y”  $\Rightarrow \frac{\text{performance}(x)}{\text{performance}(y)} = 1 + \frac{n}{100}$

Speedup(x,y)= Performance(x)/Performance(y)

# Focus on the Common Case

- Common sense guides computer design
  - Since its engineering, common sense is valuable

# Focus on the Common Case

- Common sense guides computer design
  - Since its 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

- Frequent case is often simpler and can be done faster than the infrequent case
- **What is frequent case and how much performance improved by making case faster**

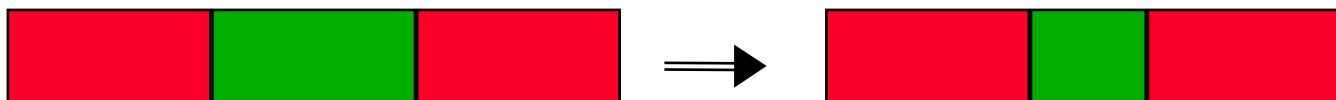
# Frequent case

- Frequent case is often simpler and can be done faster than the infrequent case
- What is frequent case and how much performance improved by making case faster  
=> Amdahl's Law

# Amdahl's Law

Speedup due to enhancement E:

$$\text{Speedup (E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected

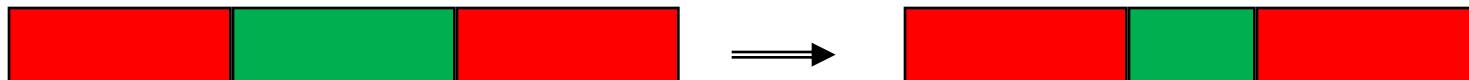
# Amdahl's Law

$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times \left[ (1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right]$$

$$\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Best you could ever hope to do:

$$\text{Speedup}_{\text{maximum}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}})}$$



# Amdahl's Law example

New CPU 10X faster

I/O bound server, so 60% time waiting for I/O

# Amdahl's Law example

New CPU 10X faster

I/O bound server, so 60% time waiting for I/O

$$\begin{aligned}\text{Speedup}_{\text{overall}} &= \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}} \\ &= \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56\end{aligned}$$

# Amdahl's Law example

New CPU 10X faster

I/O bound server, so 60% time waiting for I/O

$$\begin{aligned}\text{Speedup}_{\text{overall}} &= \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}} \\ &= \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56\end{aligned}$$

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

# Exercise on Amdahl's Law

Let's assume that we can improve the CPU speed 5X (with a 5X cost).

Suppose that the CPU is used 50% of the time and that the base CPU cost is 1/3 of the entire system

Is it worth to upgrade the CPU?  
Compare speedup and costs!

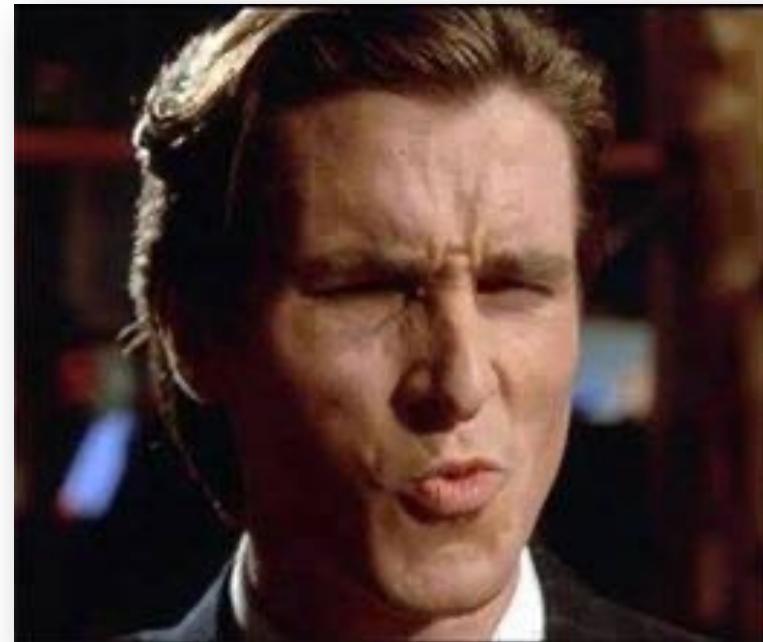
# Solution

- Speedup=  $1/(0.5+0.5/5)=1.67$
- Increased=  $(2/3)+(1/3)*5=2.33$

# Solution

- Speedup=  $1/(0.5+0.5/5)=1.67$
- Increased=  $(2/3)+(1/3)*5=2.33$

It is not worth to  
upgrade the CPU!



# Amdahl's Law

- Expresses the law of diminishing return
- Corollary

If an enhancement is only usable for a fraction of a task we can't speed up the task by more than the reciprocal of 1 minus the fraction

Serves as a guide to how much an enhancement will improve performance and how to distribute resources to improve cost/performance

# Breaking down performance

- A program is broken into instructions
  - Hardware is aware of instructions not programs
- At lower level hardware breaks instructions into clock cycles
  - Lower level state machines change state every cycle

For example

500 MHz P-III runs 500M cycles/sec, 1 cycle = 2 ns

2 GHz P-IV runs 2G cycles/sec, 1 cycle = 0,5 ns

# What is performance for us?

- For computer architects
    - **Response time** = latency due to the completion of a task including disk accesses, I/O activity, OS, ...  
Elapsed time = CPU time + I/O wait
    - **CPU time** = does **not** include I/O wait time and corresponds to CPU  
CPU time = time spent running a program

Or

CPU time (P) = cc needed to execute P x cc time

# CPU time

Processor Performance =  $\frac{\text{Time}}{\text{Program}}$

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

Instructions : Instruction Count, code size  
Program

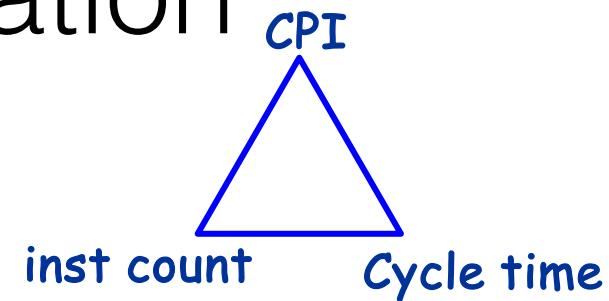
Cycles : CPI  
Instruction

Time : cycle time  
Seconds

# CPU time

- Instruction Count, IC
  - Instructions executed, not static code size
  - Determined by algorithm, compiler, Instruction Set Architecture
- Cycles per instructions, CPI
  - Determined by ISA and CPU organization
  - Overlap among instructions (pipelining) reduces this term
- Time/cycle
  - Determined by technology, organization and circuit design

# Performance equation



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

# Goal of CPU performance

- Minimize time which is the **product**, not isolated terms
- Common error to miss terms while devising optimizations
  - E.g. ISA change to decrease instruction count
  - BUT leads to CPU organization which may make clock slower
- **BOTTOM LINE:** terms are inter-related

# Average CPI

The *average* Clock Cycles per Instruction (CPI) can be defined as:

$$\text{CPI}(P) = \frac{\text{clock cycles needed to exec. } P}{\text{number of instructions}}$$

$$\text{CPUtime} = T_{\text{clock}} * \text{CPI} * N_{\text{inst}} = (\text{CPI} * N_{\text{inst}}) / f$$

# Cycles per instruction

## Average Cycles per Instruction

$$\text{CPI} = (\text{CPU Time} * \text{Clock Rate}) / \text{Instr Count} = \text{Cycles/Instr Count}$$

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^n \text{CPI}_i * I_i$$

## Instruction Frequency

$$\text{CPI} = \sum_{i=1}^n \text{CPI}_i * F_i \quad \text{where} \quad F_i = \frac{I_i}{\text{Instruction Count}}$$

**Invest Resources where time is Spent!**

# Aspects of CPU performance

- The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^n \text{CPI}_i * I_i$$

# Aspects of CPU performance

- The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^n \text{CPI}_i * I_i$$

## Example

OPER
ALU
Load
Store
Branch

# Aspects of CPU performance

- The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^n \text{CPI}_i * I_i$$

## Example

OPER	FREQ
ALU	50%
Load	20%
Store	10%
Branch	20%

# Aspects of CPU performance

- The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^n \text{CPI}_i * I_i$$

## Example

OPER	FREQ	CYCLES
ALU	50%	1
Load	20%	2
Store	10%	2
Branch	20%	2

# Aspects of CPU performance

- The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^n \text{CPI}_i * I_i$$

## Example

OPER	FREQ	CYCLES	CPI
ALU	50%	1	.5
Load	20%	2	.4
Store	10%	2	.2
Branch	20%	2	.4
			1.5

# Aspects of CPU performance

- The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^n \text{CPI}_i * I_i$$

## Example

OPER	FREQ	CYCLES	CPI	COMP
ALU	50%	1	.5	0.5/1.5
Load	20%	2	.4	0.4/1.5
Store	10%	2	.2	0.2/1.5
Branch	20%	2	.4	0.4/1.5
			1.5	

# Aspects of CPU performance

- The CPI can vary among instructions

$$\text{CPU time} = \text{CycleTime} * \sum_{i=1}^n \text{CPI}_i * I_i$$

## Example

OPER	FREQ	CYCLES	CPI	COMP	% TIME
ALU	50%	1	.5	0.5/1.5	33%
Load	20%	2	.4	0.4/1.5	27%
Store	10%	2	.2	0.2/1.5	13%
Branch	20%	2	.4	0.4/1.5	27%
			1.5		

# Other metrics

## MIPS and MFLOPS

- MIPS = millions of instructions per second  
= number of instructions / (execution time  $\times 10^6$ )  
= clock frequency/(CPI  $\times 10^6$ )
- Execution time = Instruction count / (MIPS  $\times 10^6$ )
- Since MIPS is a rate of operations per unit time, performance can be specified as the inverse of execution time, with faster machines having higher MIPS rating
- BUT MIPS has serious shortcomings

# Example

- Execution of a loop ( $a = a + a \times b$ ) 1 million times

ARCH 1 - 1 MHz

ADD 1 cycle

MUL 2 cycles

ARCH2 - 1 MHz

MAC 2 cycles

# Example

- Execution of a loop ( $a = a + a \times b$ ) 1 million times

ARCH 1 - 1 MHz

ADD 1 cycle

MUL 2 cycles

ARCH2 - 1 MHz

MAC 2 cycles

$$\text{CPI}(P) = \frac{\text{clock cycles needed to exec. } P}{\text{number of instructions}}$$

# Example

- Execution of a loop ( $a = a + a \times b$ ) 1 million times

ARCH 1 - 1 MHz

ADD 1 cycle

MUL 2 cycles

CPI=1.5

ARCH2 - 1 MHz

MAC 2 cycles

CPI=2

# Example

- Execution of a loop ( $a = a + a \times b$ ) 1 million times

ARCH 1 - 1 MHz

ADD 1 cycle

MUL 2 cycles

CPI=1.5

MIPS=1/1.5 = 0.66

Ex time = 3 sec

ARCH2 - 1 MHz

MAC 2 cycles

CPI=2

MIPS = 1/2 = 0.5

Ex Time = 2 sec

# MFLOPS

- MFLOPS = Floating Point operations in program  
CPU time  $\times 10^6$
- Assuming FP operations independent of compiler and ISA
  - Often safe for numeric codes: matrix size determines # of FP ops/program
  - However, not always safe:
    - Missing instructions (e.g. FP divide, square rot, sin, cos)
    - Optimizing compilers

# Benchmarks

- Execution time of what program?
- Standard performance test programs:  
benchmarks
  - Programs chosen to measure performance defined by some group
  - Available to the community
  - Run on machines and performance is reported
  - Can compare to reports on other machines
  - Representative?

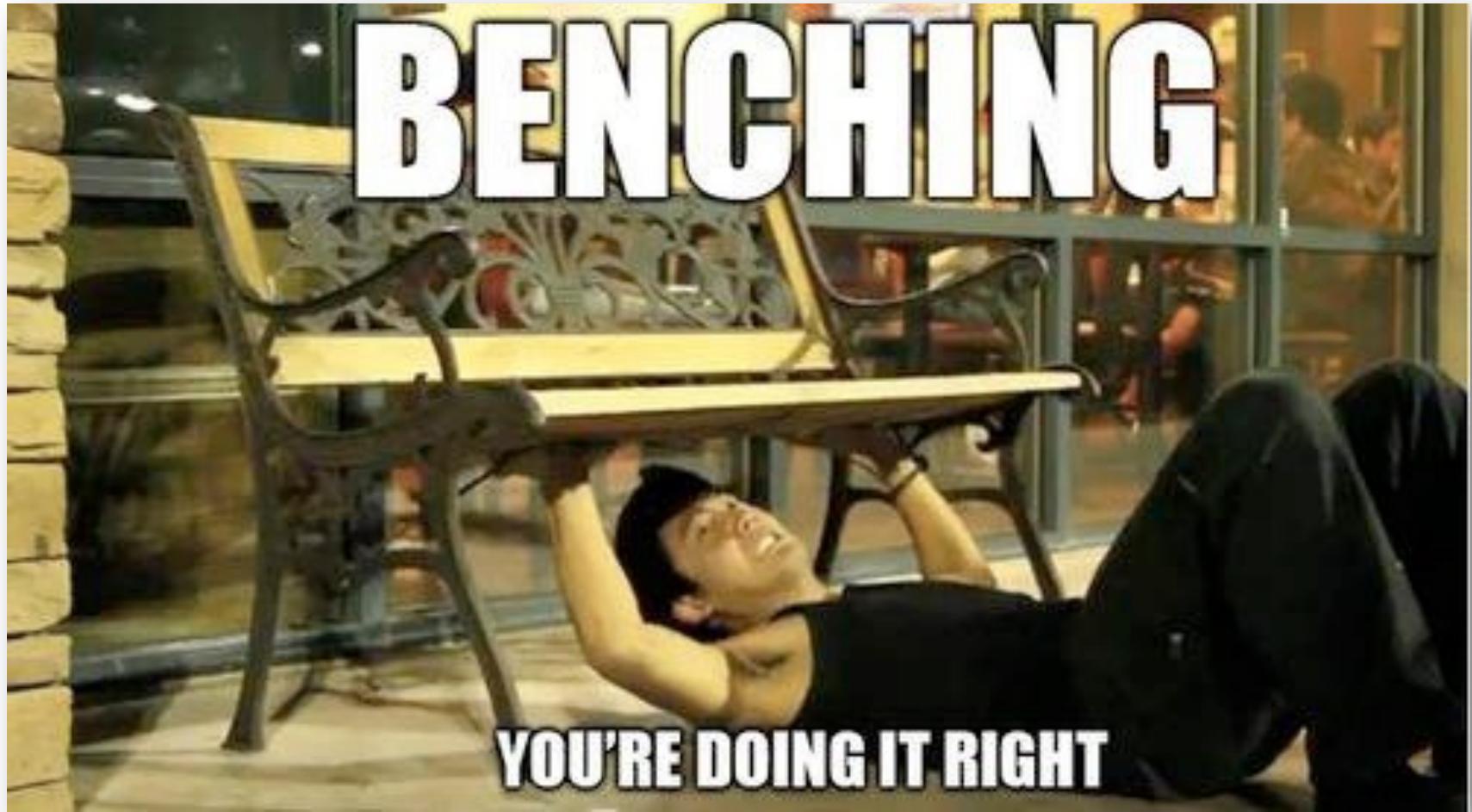
# Types of benchmarks

- Real programs
  - Representative of real workload
  - Only accurate way to characterize performance
  - E.g. C compilers, text processing, other applications
  - Sometimes modified: CPU oriented benchmarks may remove I/O
- Kernels or microbenchmarks
  - “Representative” program fragments
  - Good for focusing on individual features
- Synthetic benchmarks
  - Same philosophy of kernels
  - Try to match the average frequency of operations and operands of a large set of programs
- Instruction mixes (for CPI)

# Benchmarks: SPEC2000

- System Performance Evaluation Cooperative
  - Formed in 89 to combat benchmarketing
  - SPEC89, SPEC92, SPEC95, SPEC2000, SPEC2006 and so on
- 12 integer and 14 floating point programs
- Compute intensive performance of:
  - The CPU
  - The memory architecture
  - The compilers

# Benchmarks pitfalls



# Benchmarks pitfalls

- Benchmark not representative
  - If workload is I/O bound SPECint is useless
- Benchmark is too old
  - Benchmarks age poorly; benchmarking pressure causes vendors to optimize compiler/hardware/software to benchmarks
  - Need to be periodically refreshed

# How to average performance

	Computer A	Computer B	Computer C
P1	1	10	20
P2	1000	100	20
Total	1001	110	40

Which computer is faster?

# Summarize performance

- The simplest approach to summarizing relative performance is to use the total execution time of the two programs
  - B is 9.1 times faster than A for P1 and P2
  - C is 25 times faster than A for P1 and P2
  - C is 2.75 times faster than B for P1 and P2
- Another possibility: arithmetic mean
- Arithmetic mean of times
  - gives the same result:  
 $AM(A) = 100/2 = 500.5$   
 $AM(B) = 110/2 = 55$   
 $500.5/55 = 9.1x$

$$\frac{1}{n} \times \sum_{i=1}^n time(i)$$

Valid only if programs run equally often

# How to average

- If different frequency of execution use the weighted arithmetic mean

$$\left\{ \sum_{i=1}^n weight(i) \times time(i) \right\} \div \frac{1}{n}$$

# Other averages

- Consider for example that a drive takes 30 mph for the first 10 miles and then 90 mph for the next 10 miles, what is the average speed?
- Average speed =  $(30+90)/2$  **WRONG**
- Average speed = total distance/total time  
 $= (20 / (10/30 + 10/90)) = 45 \text{ mph}$
- When dealing with **rates (mph)** do not use arithmetic mean!!
- Another example
  - B1: 10 Minst, 1 MIPS  $\rightarrow$  10 sec
  - B2: 10 Mints, 5 MIPS  $\rightarrow$  2 sec
  - Average would give 3 MIPS BUT....  
 $20 \text{ Minst}/12 \text{ sec} = 1.7 \text{ MIPS}$

# What's best?

- Use arithmetic mean for times
- Use harmonic mean if forced to use rates
  - Speeding up slower benchmarks gives more reward
  - Reflects total execution time
- Use geometric mean if forced to use ratios
- Best: use unnormalized numbers (e.g. CPU time)

# Conclusions

- For better or worse, benchmarks “shape a field”
- Good products created when have:
  - Good benchmarks
  - Good ways to summarize performance
- Given sales is a function in part of performance relative to competition, investment in improving product as reported by performance summary
- If benchmarks/summary inadequate, then choose between improving product for real programs vs. improving product to get more sales
  - Sales almost always wins!
- Execution time is the measure of computer performance!



Where to go next?

# Introducing Energy and Power

# Problem definition

- Energy and power consumption - **Are they a constraint for...**

# Problem definition

- Energy and power consumption - **Are they a constraint for...**
  - Embedded & IoT
  - Mobile devices



# Problem definition

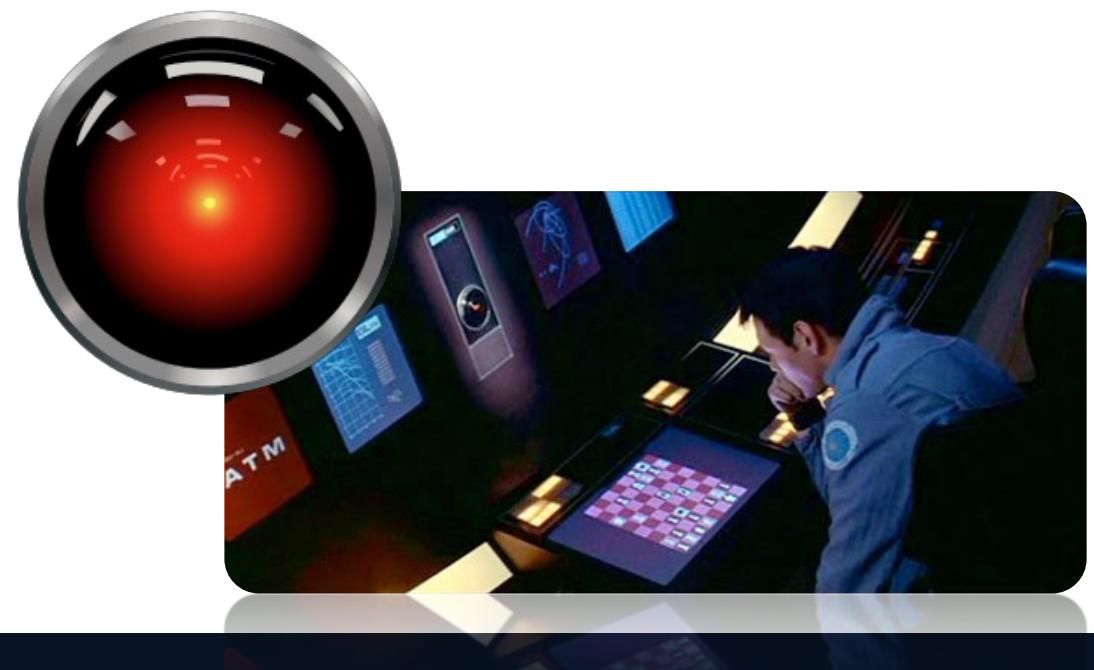
- Energy and power consumption - **Are they a constraint for...**
  - Embedded & IoT
  - Mobile devices

**Batteries capacities**

# Problem definition

- Energy and power consumption - **Are they a constraint for...**
  - Embedded & IoT
  - Mobile devices
  - Desktops
  - Servers
  - HPC Clusters

**Batteries capacities**



# Problem definition

- Energy and power consumption - **Are they a constraint for...**

- Embedded & IoT
- Mobile devices
- Desktops
- Servers
- HPC Clusters

**Batteries capacities**

**Energy costs**

# Problem definition

- Energy and power consumption - **Are they a constraint for...**

- Embedded & IoT
- Mobile devices
- Desktops
- Servers
- HPC Clusters

**Energy and Power  
budgets**

# Problem definition

- Energy and power consumption - **Are they a constraint for...**

- Embedded & IoT
- Mobile devices
- Desktops
- Servers
- HPC Clusters

**Energy and Power budgets**

- Any optimization needs a deep understanding of the phenomenon

Energy and Power, but not only...

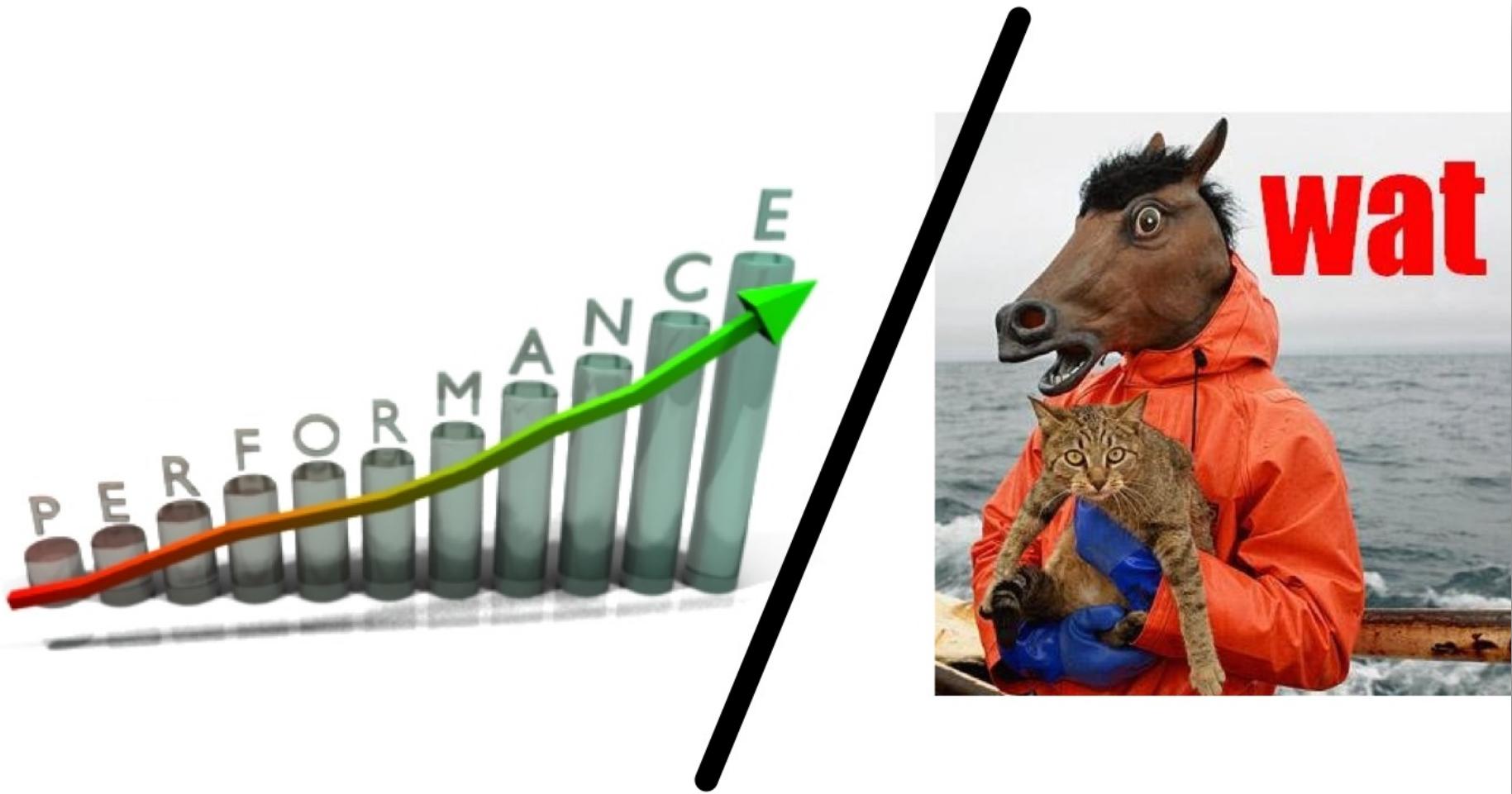
# Power and Energy

- Problem: Get power in, get power out
- Thermal Design Power (TDP)
  - Characterizes sustained power consumption
  - Used as target for power supply and cooling system
  - Lower than peak power, higher than average power consumption
- Clock rate can be reduced dynamically to limit power consumption
- Energy per task is often a better measurement

# Reducing Power

- Techniques for reducing power:
  - Do nothing well
  - Dynamic Voltage-Frequency Scaling
  - Low power state for DRAM, disks
  - Overclocking, turning off cores
- Static power consumption
  - $\text{Current}_{\text{static}} \times \text{Voltage}$
  - Scales with number of transistors
  - To reduce: power gating

# Energy and Power, but not only...



# Other factors?

# Other factors?



# Factors Determining Cost

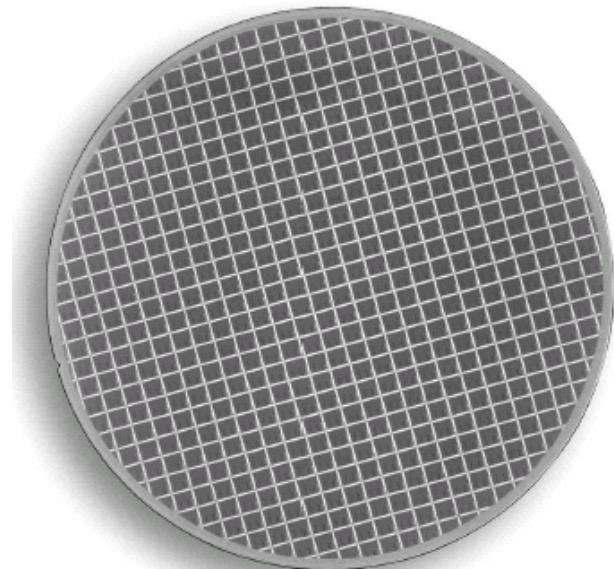
- Cost: amount spent by manufacturer to produce a finished good
- High volume → faster learning curve, increased manufacturing efficiency (10% lower cost if volume doubles), lower R&D cost per produced item
- Commodities: identical products sold by many vendors in large volumes (keyboards, DRAMs) – low cost because of high volume and competition among suppliers

# Trends in Cost

- Cost driven down by learning curve
  - Yield
- DRAM: price closely tracks cost
- Microprocessors: price depends on volume
  - 10% less for each doubling of volume

# Wafers and Dies

An entire wafer is produced and chopped into dies that undergo testing and packaging



# Integrated Circuit Cost

Cost of an integrated circuit =  
(cost of die + cost of packaging and testing) / final test yield

Cost of die = cost of wafer / (dies per wafer x die yield)

Dies/wafer = wafer area / die area -  
 $\pi$  wafer diam / die diag

Die yield = wafer yield x  
 $(1 + (\text{defect rate} \times \text{die area}) / a)^{-a}$

Thus, die yield depends on die area and complexity arising from multiple manufacturing steps ( $a \sim 4.0$ )

# Integrated Circuit Cost

- Integrated circuit

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

- Bose-Einstein formula:

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2} \times \text{Die area}}$$

- Defects per unit area = 0.016-0.057 defects per square cm (2010)

$$\text{Die yield} = \text{Wafer yield} \times 1/(1 + \text{Defects per unit area} \times \text{Die area})^N$$

- N = process-complexity factor = 11.5-15.5 (40 nm, 2010)

# Dependability

- Module reliability
  - Mean time to failure (MTTF)
  - Mean time to repair (MTTR)
  - Mean time between failures (MTBF) = MTTF + MTTR
  - Availability = MTTF / MTBF

# THANK YOU FOR YOUR ATTENTION



more awesome pictures at [THEMETAPICTURE.COM](http://THEMETAPICTURE.COM)