

# Outline

- Definition of performance
- Execution time
- What affects your performance

# Performance

# What do you want for a computer?

- Latency/Execution time
- Frame rate
- Responsiveness
- Real-time
- Throughput
- Cost
- Volume
- Weight
- Battery life
- Low power/low temperature
- Reliability

# How about running a single program

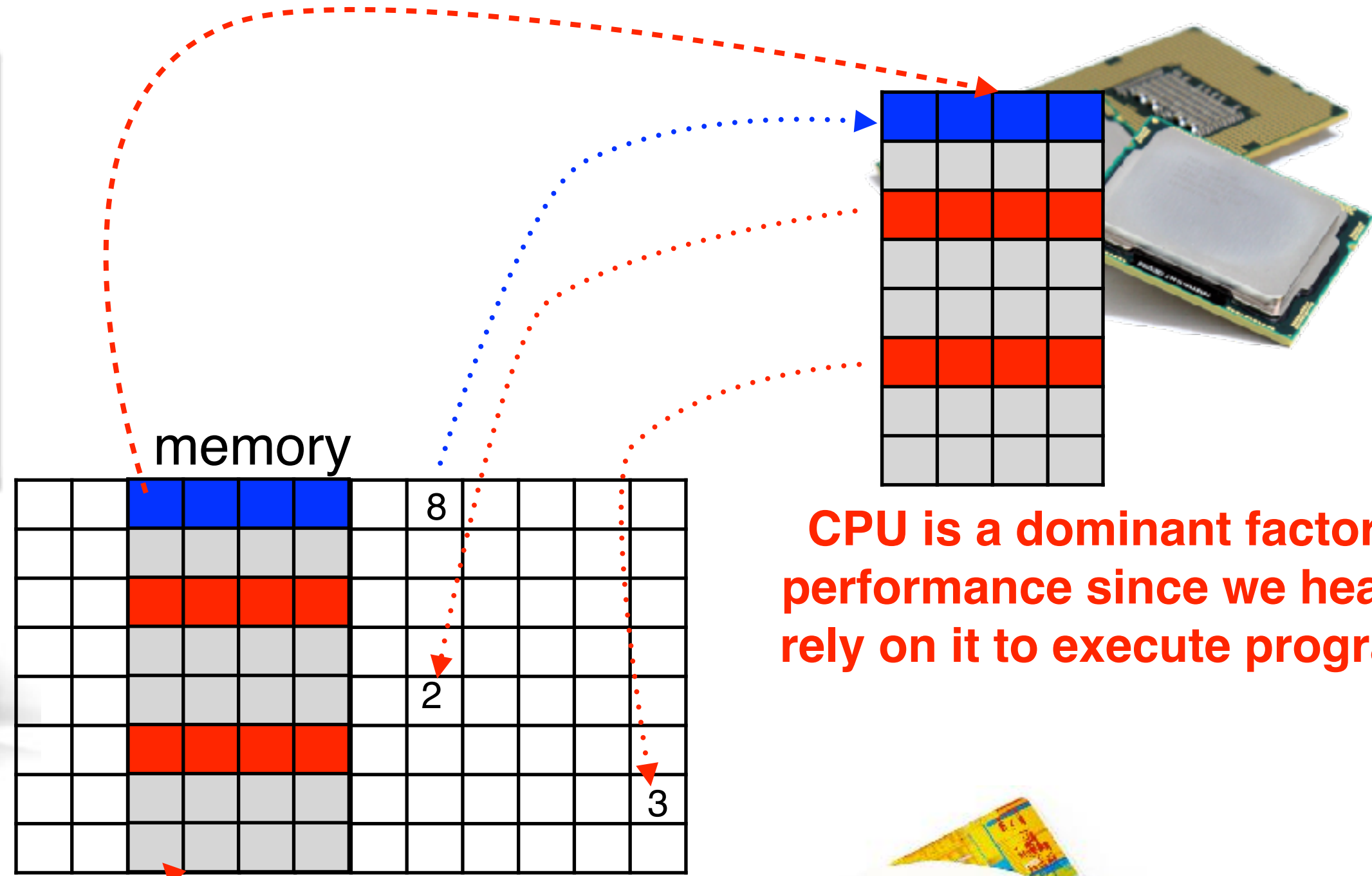
- Latency/Execution time
- Frame rate
- Responsiveness
- Real-time
- Throughput
- Cost
- Volume
- Weight
- Battery life
- Low power/low temperature
- Reliability

The most direct measurement  
of performance



# Evaluating the execution time of a program

# Recap: Von Neumann architecture



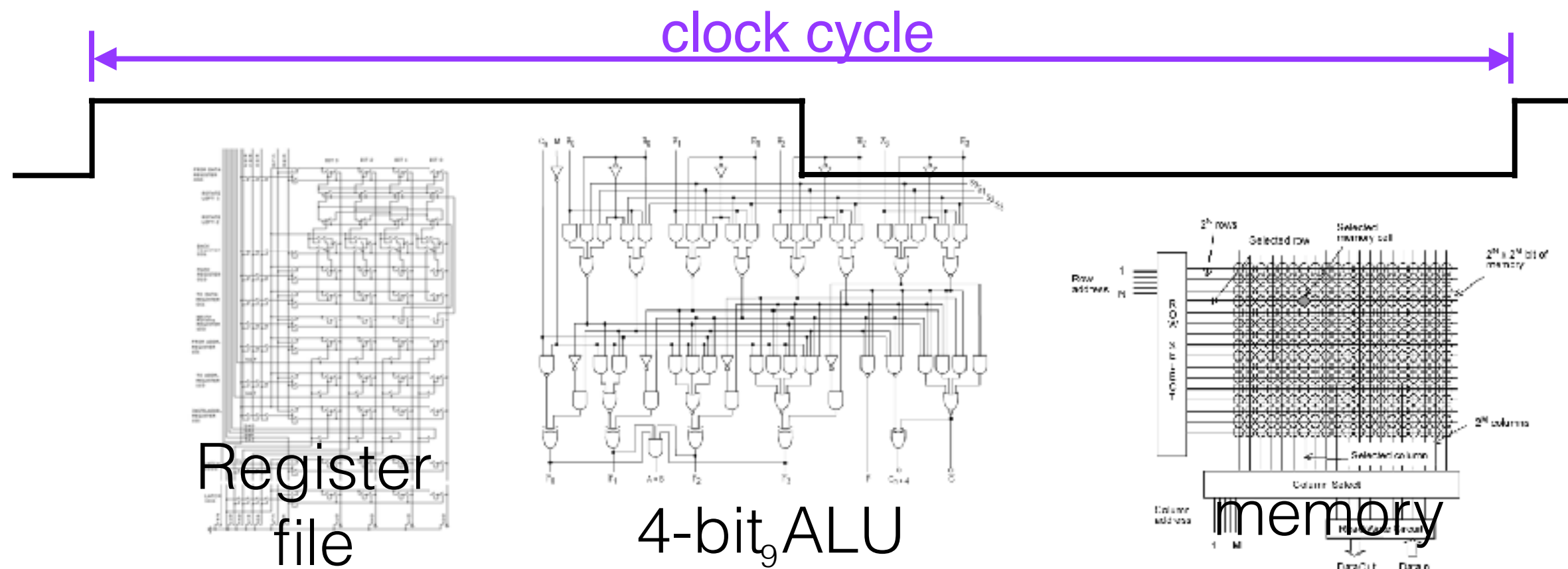
**CPU is a dominant factor of performance since we heavily rely on it to execute programs**

**By pointing "PC" to different part of your memory, we can perform different functions!**



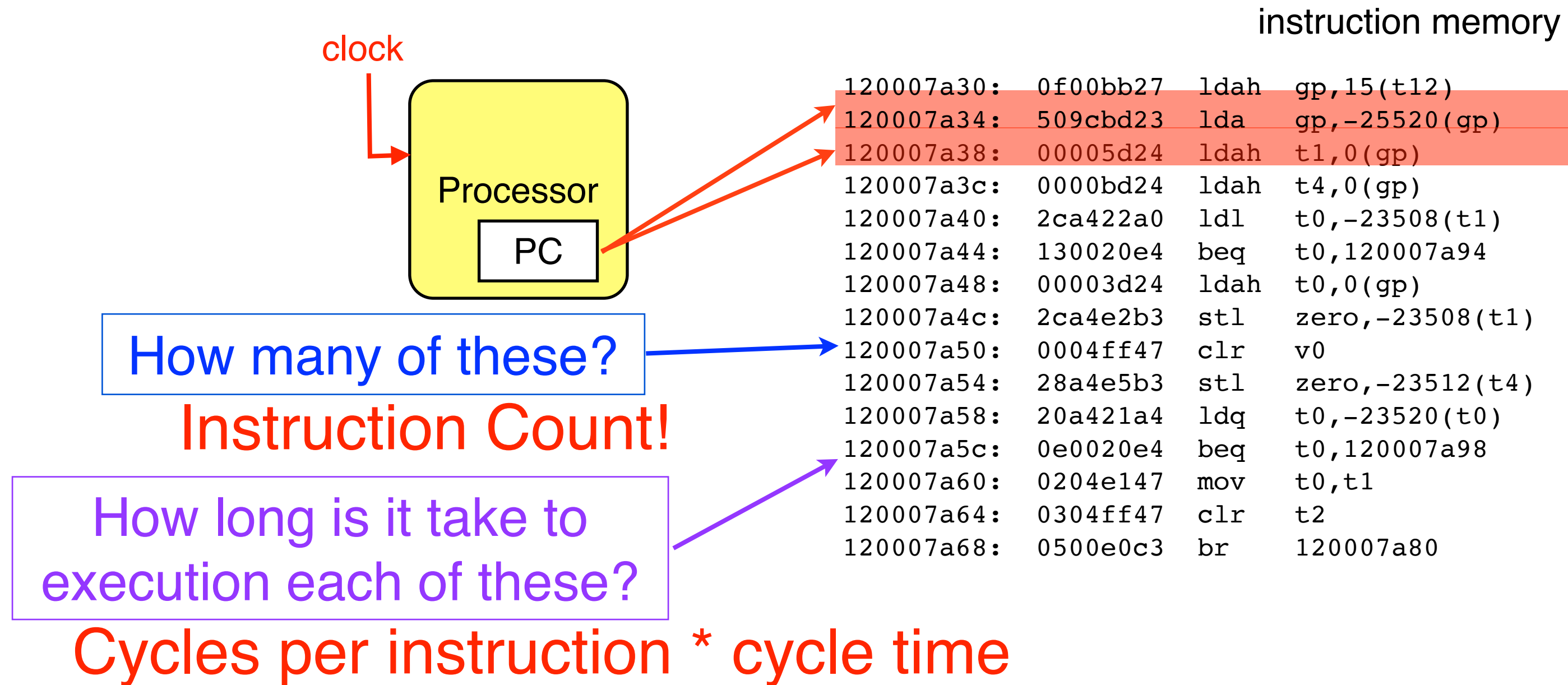
# Recap: Clock — synchronizing hardware components

- A hardware signal defines when data for any specific component is ready to use by others
  - Think about the clock in real life!
- We use edge-triggered clocking
  - Values stored in the sequential logic is updated only on a clock edge



# Execution Time

- The simplest kind of performance
- Shorter execution time means better performance
- Usually measured in seconds





# Performance Equation

$$\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

How many instruction  
are executed?

How long is it take to execute  
each instruction

- $ET = IC * CPI * CT$
- IC (Instruction Count)
- CPI (Cycles Per Instruction)
- CT (Seconds Per Cycle)
- 1 Hz = 1 second per cycle; 1 GHz = 1 ns per cycle



# Relative performance

- Can be confusing
  - A runs in 12 seconds
  - B runs in 20 seconds
  - We know A is faster, but
    - $A/B = .6$  , so A is 40% faster, or 1.4X faster, or B is .40% slower
    - $B/A = 1.67$ , so A is 67% faster, or 1.67X faster, or B is 67% slower
- Needs a precise definition

# Speedup

- Compare the relative performance of the baseline system and the improved system
- Definition

$$\text{Speedup} = \frac{\text{Execution time}_{\text{baseline}}}{\text{Execution time}_{\text{improved system}}}$$

$$\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

# What affects performance

# Demo: programmer & performance

- Row-major, column major
  - How do you know this?
- Let's identify where the performance gain is from!
  - Using “performance counters”
  - You may use “perf stat” in linux
  - You can also create your own functions to obtain counter values
    - <https://github.ncsu.edu/htseng3/CSC456/tree/master/performance>

# Applications

- Different applications can have different CPIs on the same machine

# Compiler

- Compiler can change the combination of instructions and lead to different CPIs, instruction counts.



# Summary: Performance Equation

$$\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

- $ET = IC * CPI * \text{Cycle Time}$
- IC (Instruction Count)
  - ISA, Compiler, algorithm, programming language
- CPI (Cycles Per Instruction)
  - Machine Implementation, microarchitecture, compiler, application, algorithm, programming language
- Cycle Time (Seconds Per Cycle)
  - Process Technology, microarchitecture, programmer

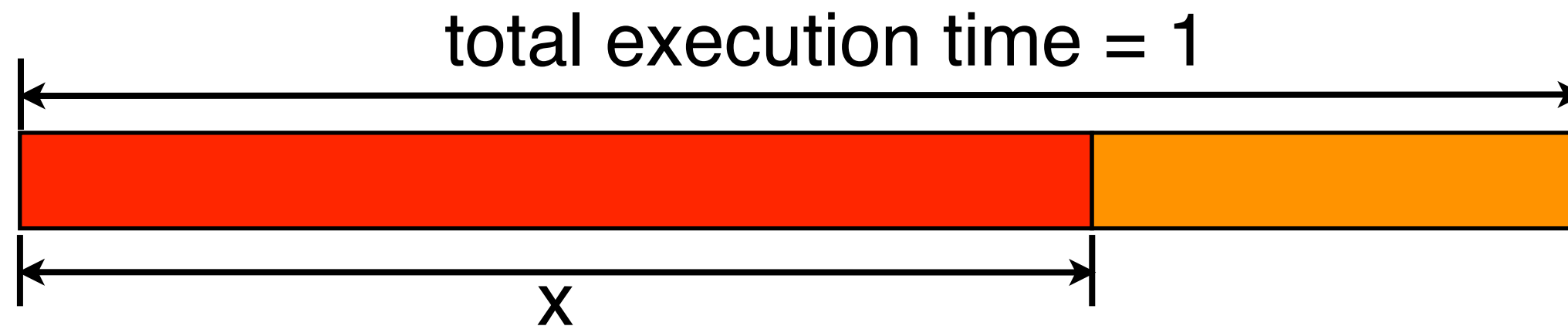
# Amdahl's Law

# Amdahl's Law

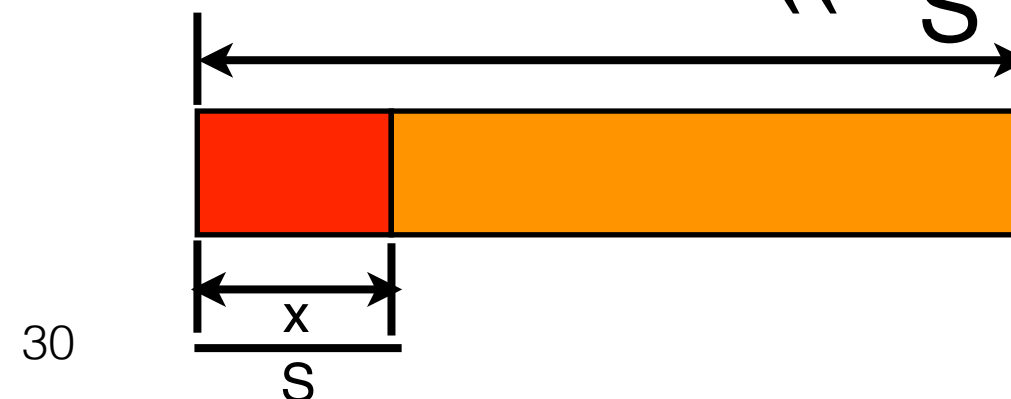


$$\text{Speedup} = \frac{1}{\left(\frac{x}{S}\right) + (1-x)}$$

- x: the fraction of “execution time” that we can speed up in the target application
- S: by how many times we can speedup x



$$\text{total execution time} = \left(\frac{x}{S}\right) + (1-x)$$



# Performance Example

- Assume that we have an application composed with a total of **500000** instructions, in which **20%** of them are the load/store instructions with an average **CPI of 6** cycles, and **the rest** instructions are integer instructions with average **CPI of 1** cycle.
- If we double the CPU **clock rate** to **4GHz** but keep using the same memory module, the average CPI for **load/store instruction** will become **12** cycles. What's the performance improvement after this change?

How much time in load/store?

$$500000 * (0.2 * 6) * 0.5 \text{ ns} = 300000 \text{ ns} \quad 60\%$$

How much time in the rest?

$$500000 * (0.8 * 1) * 0.5 \text{ ns} = 200000 \text{ ns} \quad 40\%$$

# Performance Example

- Assume that we have an application composed with a total of **500000** instructions, in which **20%** of them are the load/store instructions with an average **CPI of 6** cycles, and **the rest** instructions are integer instructions with average **CPI of 1** cycle.
- If we double the CPU **clock rate** to **4GHz** but keep using the same memory module, the average CPI for **load/store instruction** will become **12** cycles. What's the performance improvement after this change?

$$\text{Speedup} = \frac{1}{\frac{0.4}{2} + (1-0.4)}$$

$$\text{Speedup} = \frac{1}{0.8} = 1.25$$

# Amdahl's Law: Revisited

# Performance Example

- Assume that we have an application composed with a total of **500000** instructions, in which **20%** of them are the load/store instructions with an average **CPI of 6** cycles, and **the rest** instructions are integer instructions with average **CPI of 1** cycle.
- If we double the CPU **clock rate** to **4GHz** but keep using the same memory module, the average CPI for **load/store instruction** will become **12** cycles. What's the performance improvement after this change?

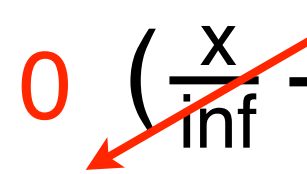
$$\text{Speedup} = \frac{1}{\frac{0.4}{2} + (1-0.4)}$$

$$\text{Speedup} = \frac{1}{0.8} = 1.25$$

# Amdahl's Corollary #1

- Maximum possible speedup  $S_{\max}$ , if we are targeting  $x$  of the program.

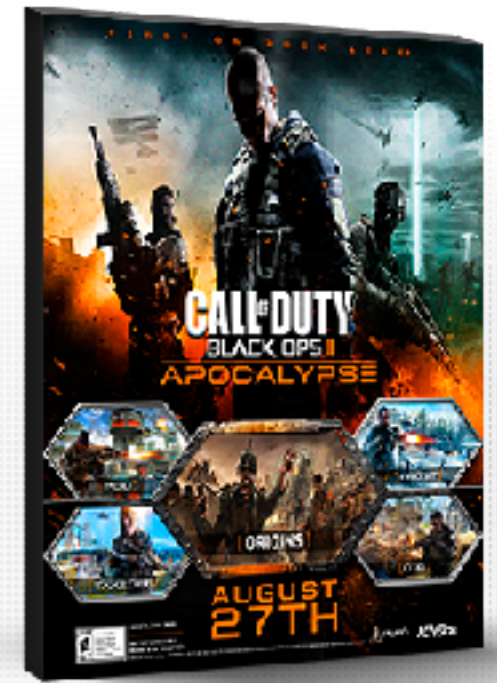
$$S = \text{infinity}$$

$$S_{\max} = \frac{1}{\left(\frac{x}{\text{inf}} + (1-x)\right)}$$
$$S_{\max} = \frac{1}{(1-x)}$$




# Maximum of speedup

- Call of Duty Black Ops II loads a zombie map for 10 minutes on my current machine, and spends **20%** of this time in integer instructions
- How much faster must you make the integer unit to make the map loading **5 minutes** faster?



$$S_{\max} = \frac{1}{(1-x)}$$
$$1.25 = \frac{1}{(1-20\%)}$$

**2x is not possible.**

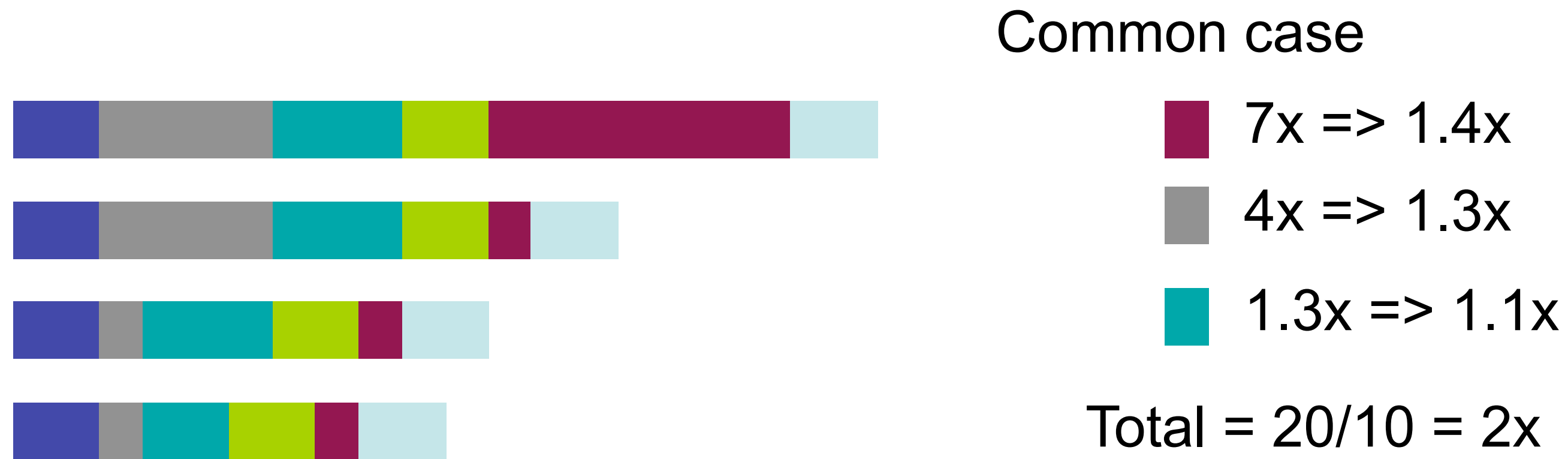
# Amdahl's Corollary #2

- Make the **common case** fast (i.e.,  $x$  should be large)!
- Common == **most time consuming** not necessarily the **most frequent**
- The uncommon case doesn't make much difference
- Be sure of what the common case is
- The common case can change based on inputs, compiler options, optimizations you've applied, etc.

# Identify the most time consuming part

- Compile your program with -pg flag
- Run the program
  - It will generate a gmon.out
  - `gprof your_program gmon.out > your_program.prof`
- It will give you the profiled result in `your_program.prof`

# If we repeatedly optimizing our design based on Amdahl's law...



- With optimization, the common becomes uncommon.
- An uncommon case will (hopefully) become the new common case.
- Now you have a new target for optimization.

# Demo

- Quicksort takes a lot of time if we want to sort a 300M array
- GPU gives you 10x speed up!
- New bottleneck emerges!

# Don't hurt non-common part too mach

- If the program spend 90% in A, 10% in B. Assume that an optimization can accelerate A by 9x, by hurts B by 10x...
- Assume the original execution time is T. The new execution time

$$T_{\text{new}} = \frac{T \times 0.9}{9} + T \times 0.1 \times 10$$

$$T_{\text{new}} = 1.1T$$

$$\text{Speedup} = \frac{T}{1.1T} = 0.91$$

# Amdahl's Corollary #3

- Assume that we have an application, in which  $x$  of the execution time in this application can be fully parallelized with  $S$  processors. What's the speedup if we use a  $S$ -core processor instead of a single-core processor?

$$S_{\text{par}} = \frac{1}{\frac{x}{S} + (1-x)}$$

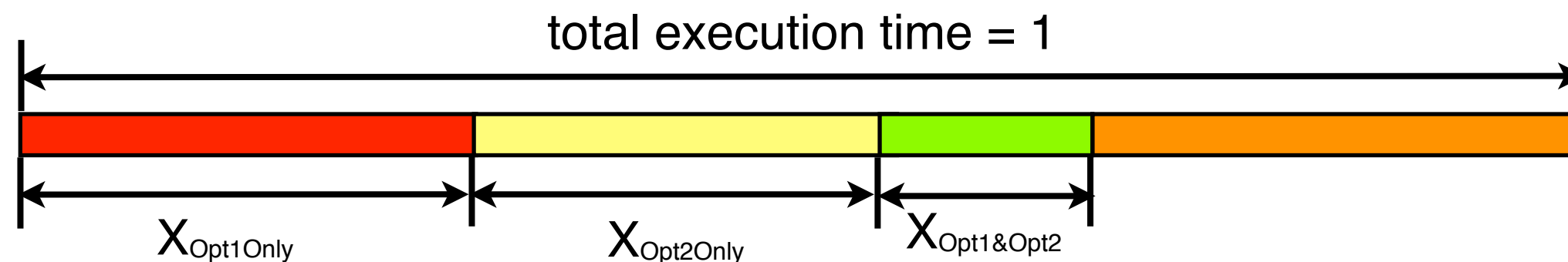
# Multiple optimizations

- We can apply Amdahl's law for multiple optimizations
- These optimizations must be dis-joint!
  - If optimization #1 and optimization #2 are dis-joint:

$$\text{Speedup} = \frac{1}{(1 - X_{\text{Opt1}} - X_{\text{Opt2}}) + \frac{X_{\text{Opt1}}}{S_{\text{Opt1}}} + \frac{X_{\text{Opt2}}}{S_{\text{Opt2}}}}$$

- If optimization #1 and optimization #2 are not dis-joint:

$$S = \frac{1}{(1 - X_{\text{Opt1Only}} - X_{\text{Opt2Only}} - X_{\text{Opt1\&Opt2}}) + \frac{X_{\text{Opt1}}}{S_{\text{Opt1Only}}} + \frac{X_{\text{Opt2}}}{S_{\text{Opt2Only}}} + \frac{X_{\text{Opt1\&Opt2}}}{S_{\text{Opt1\&Opt2}}}}$$





# Amdahl's Law for multicore processors

- Assume that we have an application, in which 50% of the application can be fully parallelized with 2 processors. Assuming 80% of the parallelized part can be further parallelized with 4 processors, what's the speed up of the application running on a 4-core processor?

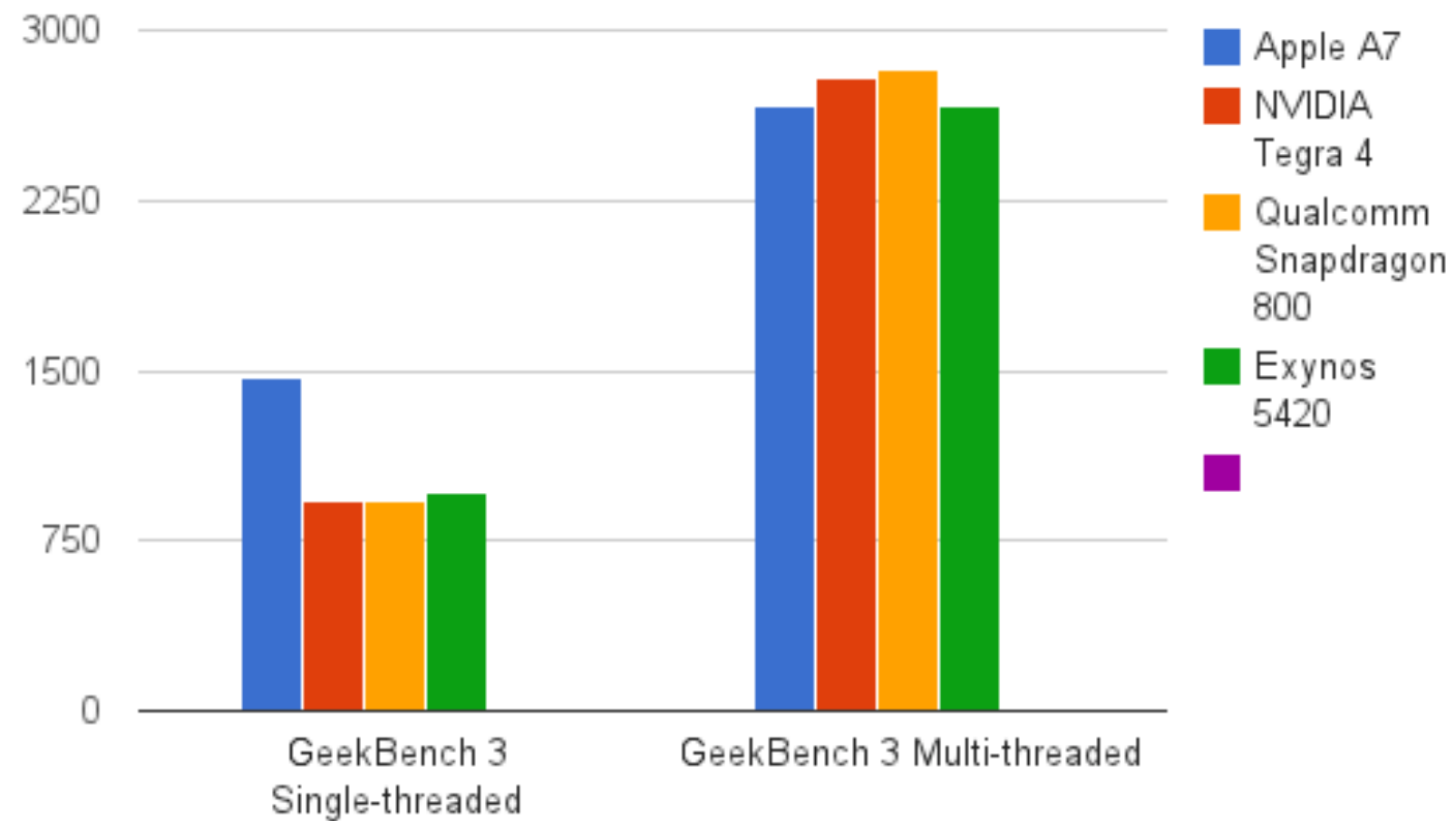
Code can be optimized for 2-core =  $50\% * (1 - 80\%) = 10\%$

Code can be optimized for 4-core =  $50\% * 80\% = 40\%$

$$\text{Speedup}_{\text{quad}} = \frac{1}{(1 - 0.5) + \frac{0.10}{2} + \frac{0.40}{4}} = 1.54$$

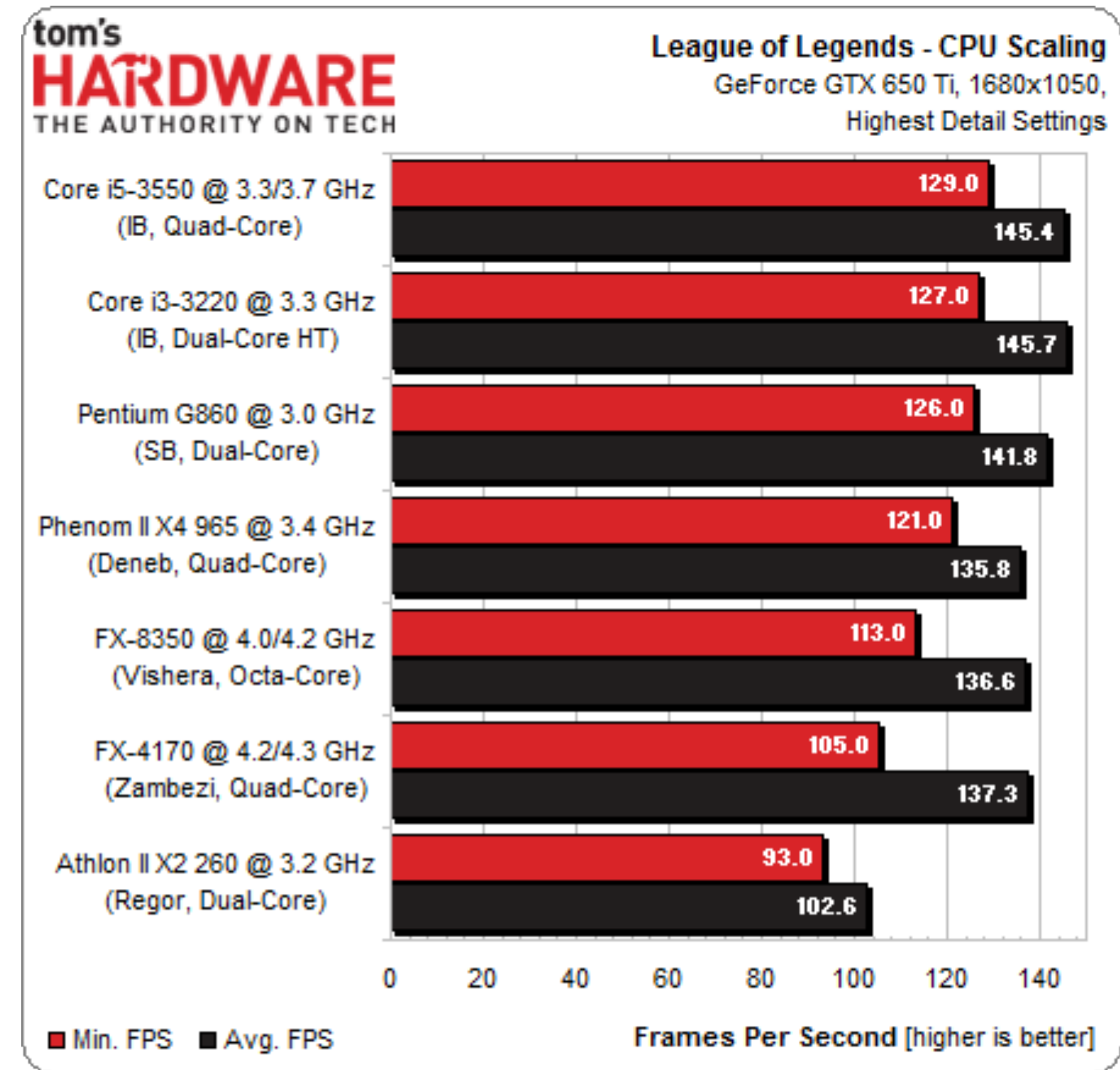
# Case study: more cores?

- If you cannot make your mobile Apps multithreaded, Apple A7 is the best



# Case study: LOL

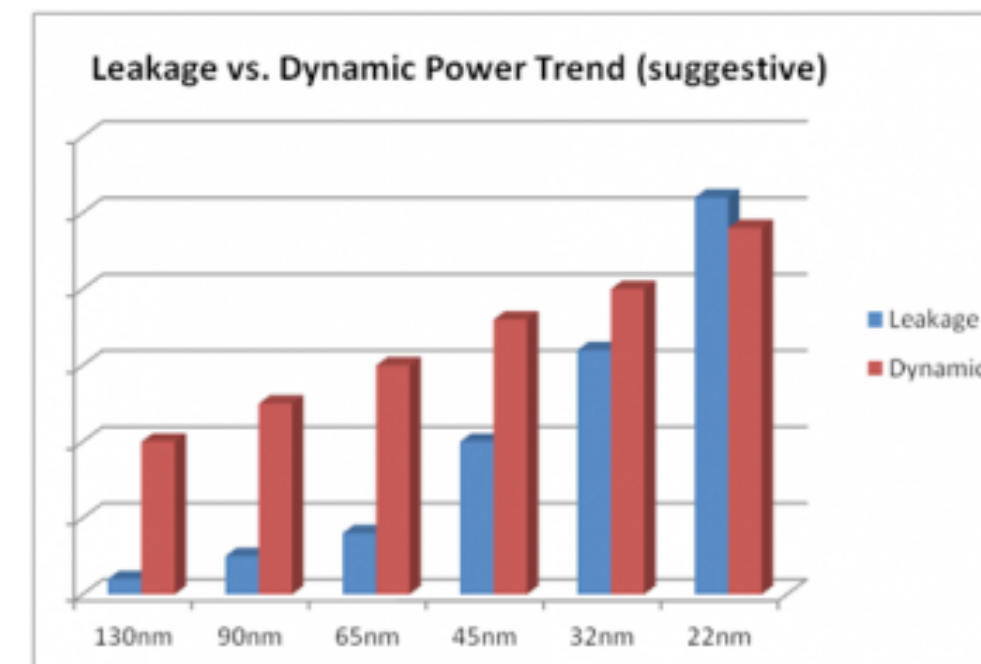
- Corollary #2
- The CPU is not the main performance bottleneck
- CPU parallelism doesn't help, either
- You might consider
  - GPU
  - network
  - storage (loading maps)



# Power & Energy

# Power

- Dynamic power:  $P = aCV^2f$ 
  - $a$ : switches per cycle
  - $C$ : capacitance
  - $V$ : voltage
  - $f$ : frequency, usually linear with  $V$
  - Doubling the clock rate consumes more power than a quad-core processor!
- Static/Leakage power becomes the dominant factor in the most advanced process technologies.
- Power is the direct contributor of “heat”
  - Packaging of the chip
  - Heat dissipation cost



# Energy

- $\text{Energy} = P * ET$
- The electricity bill and battery life is related to energy!
- Lower power does not necessary means better battery life if the processor slow down the application too much

# Double Clock Rate or Double the Processors?

- Assume 60% of the application can be fully parallelized with 2-core or speedup linearly with clock rate. Should we double the clock rate or duplicate a core?

$$\text{Speedup}_{2\text{-core}} = \frac{1}{(1 - 0.6) + \frac{0.6}{2}} = 1.43$$

$$\text{Power}_{2\text{-core}} = 2x$$

$$\text{Energy}_{2\text{-core}} = 2 * [1/(1.43)] = 1.39$$

$$\text{Speedup}_{2x\text{Clock}} = 2$$

$$\text{Power}_{2x\text{Clock}} = 8x$$

$$\text{Energy}_{2x\text{Clock}} = 8 / 2 = 4$$

# Other important metrics



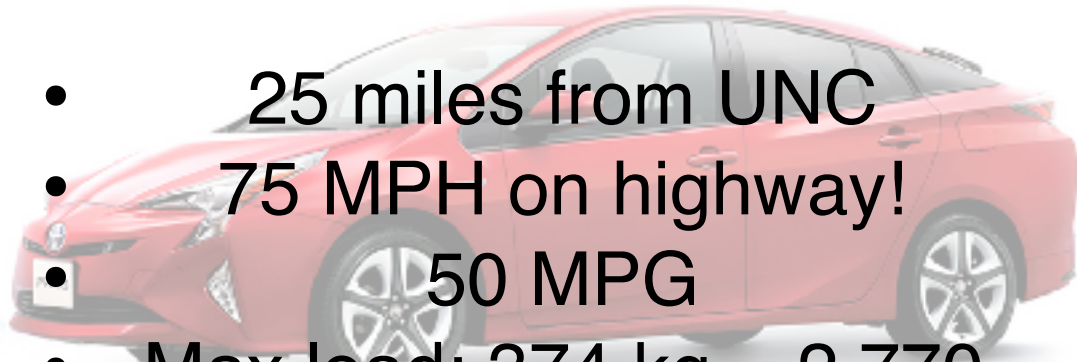

# Bandwidth

- The amount of work (or data) during a period of time
  - Network/Disks: MB/sec, GB/sec, Gbps, Mbps
  - Game/Video: Frames per second
- Also called “throughput”
- “Work done” / “execution time”

# Response time and BW trade-off

- Increase bandwidth can hurt the response time of a single task
- If you want to transfer a 2 Peta-Byte video from UNC
  - 25 miles from NCSU
  - Assume that you have a 100Gbps ethernet
    - 2 Peta-byte over 167772 seconds = 1.94 Days
    - 22.5TB in 30 minutes
    - Bandwidth: 100 Gbps

# Or ...

	Toyota Prius	10Gb Ethernet
	 <ul style="list-style-type: none"><li>• 25 miles from UNC</li><li>• 75 MPH on highway!</li><li>• 50 MPG</li><li>• Max load: 374 kg = 2,770 hard drives (2TB per drive)</li></ul>	
bandwidth	1.53TB/sec	100 Gb/s or 12.5GB/sec
latency	1 hour	2 Peta-byte over 167772 seconds = 1.94 Days
response time	You see nothing in the first hour	You can start watching the movie as soon as you get a frame!

# Reliability

- Mean time to failure (MTTF)
  - Average time before a system stops working
  - Very complicated to calculate for complex systems
- Hardware can fail because of
  - Electromigration
  - Temperature
  - High-energy particle strikes

# GFLOPS (Giga Floating-point Operations Per Second)

- MIPS does not include instruction count!
  - Cannot compare different ISA/compiler
  - Different CPI of applications, for example, I/O bound or computation bound
  - If new architecture has more IC but also lower CPI?

	GFLOPS	clock rate
XBOX One	1310	1.75 GHz
PS4	1843	1.6 GHz
Core i7 EE 3970X + AMD Radeon 6990	5099	3.5 GHz

## Is GFLOPS (Giga Floating-point Operations Per Second) a good metric?

- Cannot compare different ISA/compiler
  - What if the compiler can generate code with fewer instructions?
  - What if new architecture has more IC but also lower CPI?
- Does not make sense if the application is not floating point intensive

$$\begin{aligned} \text{GFLOPS} &= \frac{\# \text{ of floating point instructions} / 10^9}{\text{Execution Time}} \\ &= \frac{\text{IC} \times \% \text{ of floating point instructions}}{\text{IC} \times \text{CPI} \times \text{Cycle Time} \times 10^9} = \frac{\text{Clock Rate} \times \% \text{ FP ins.}}{\text{CPI} \times 10^9} \end{aligned}$$