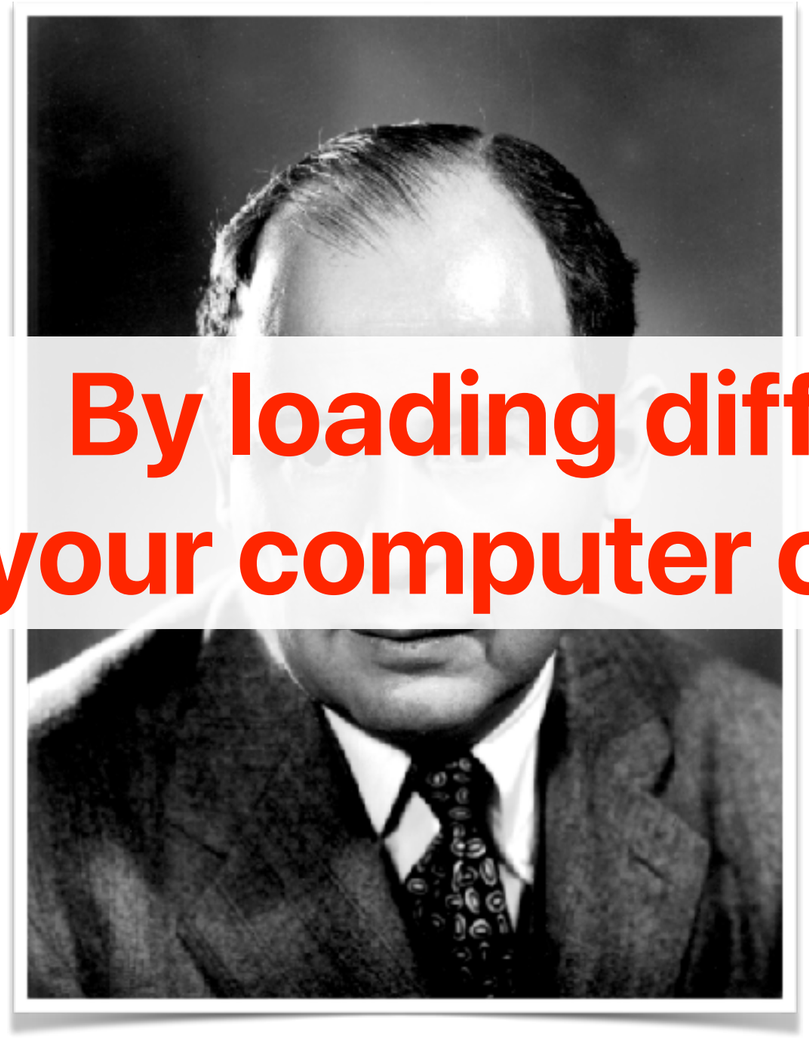
Midterm Review

Hung-Wei Tseng

von Neumann model & Performance equation

von Neumman Architecture



509cbd23

00c2e800

Processor

Program

Instruction

Data

Instruction	Data
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
2ca422a0	00c2f800
130020e4	00000008
00003d24	00c30000
2ca4e2b3	00000008

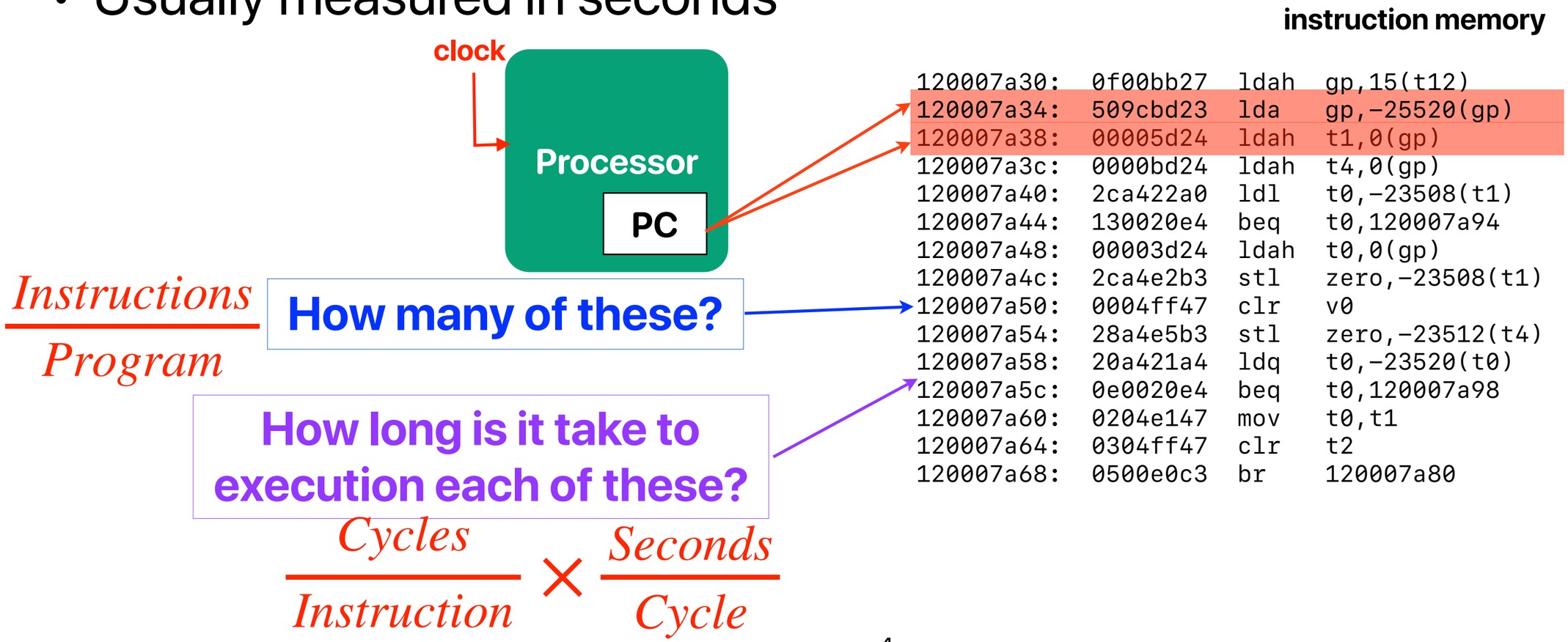
Memory

Storage

By loading different programs into memory, your computer can perform different functions

Execution Time

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



CPU Performance Equation

$$Performance = \frac{1}{Execution\ Time}$$

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

$$ET = IC \times CPI \times CT$$

$$1GHz = 10^9 Hz = \frac{1}{10^9} sec\ per\ cycle = 1\ ns\ per\ cycle$$

$$\frac{1}{Frequency(i.e., clock\ rate)}$$

Performance Equation (X)

- Assume that we have an application composed with a total of **5000000000** instructions, in which **20%** of them are "Type-A" instructions with an average **CPI of 8** cycles, **20%** of them are "Type-B" instructions with an average **CPI of 4** cycles and the rest instructions are "Type-C" instructions with average **CPI of 1** cycle. If the processor runs at **3 GHz**, how long is the execution time?

A. 3.67 sec
B. 5 sec
C. 6.67 sec
D. 15 sec
E. 45 sec

$$ET = (5 \times 10^9) \times \underbrace{(20\% \times 8 + 20\% \times 4 + 60\% \times 1)}_{\textbf{average CPI}} \times \frac{1}{3 \times 10^{-9}} sec = 5$$

$$ET = IC \times CPI \times CT$$

Speedup

- The relative performance between two machines, X and Y. Y is n times faster than X

$$n = \frac{Execution\ Time_X}{Execution\ Time_Y}$$

- The speedup of Y over X

$$Speedup = \frac{Execution\ Time_X}{Execution\ Time_Y}$$

Speedup of Y over X

- Consider the same program on the following two machines, X and Y. By how much Y is faster than X?

	Clock Rate	Instructions	Percentage of Type-A Insts.	CPI of Type-A Insts.	Percentage of Type-B Insts.	CPI of Type-B Insts.	Percentage of Type-C Insts.	CPI of Type-C Insts.
Machine X	3 GHz	500000	20%	8	20%	4	60%	1
Machine Y	5 GHz	500000	20%	13	20%	4	60%	1

A. 0.2
B. 0.25
C. 0.8
D. 1.25
E. No changes

$$ET_Y = (5 \times 10^6) \times (20\% \times 13 + 20\% \times 4 + 60\% \times 1) \times \frac{1}{5 \times 10^{-9}} sec = 4$$

$$Speedup = \frac{Execution\ Time_X}{Execution\ Time_Y}$$

$$= \frac{5}{4} = 1.25$$

What Affects Each Factor in Performance Equation

Summary of CPU Performance Equation

$$Performance = \frac{1}{Execution\ Time}$$

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

$$ET = IC \times CPI \times CT$$

- IC (Instruction Count)
  - ISA, Compiler, algorithm, programming language, **programmer**
- CPI (Cycles Per Instruction)
  - Machine Implementation, microarchitecture, compiler, application, algorithm, programming language, **programmer**
- Cycle Time (Seconds Per Cycle)
  - Process Technology, microarchitecture, **programmer**

Programmer's impact

- By adding the "sort" in the following code snippet, what the programmer changes in the performance equation to achieve **better** performance?

```
std::sort(data, data + arraySize);

for (unsigned c = 0; c < arraySize*1000; ++c) {
        if (data[c%arraySize] >= INT_MAX/2)
            sum ++;
    }
}
```

A. CPI

B. IC — we increased IC, suppose to make the

C. CT performance worse

D. IC & CPI

Demo — programmer & performance

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

$O(n^2)$	Complexity	$O(n^2)$
Same	Instruction Count?	Same
Same	Clock Rate	Same
Better	CPI	Worse

Programmers can also set the cycle time

https://software.intel.com/sites/default/files/comment/1716807/how-to-change-frequency-on-linux-pub.txt

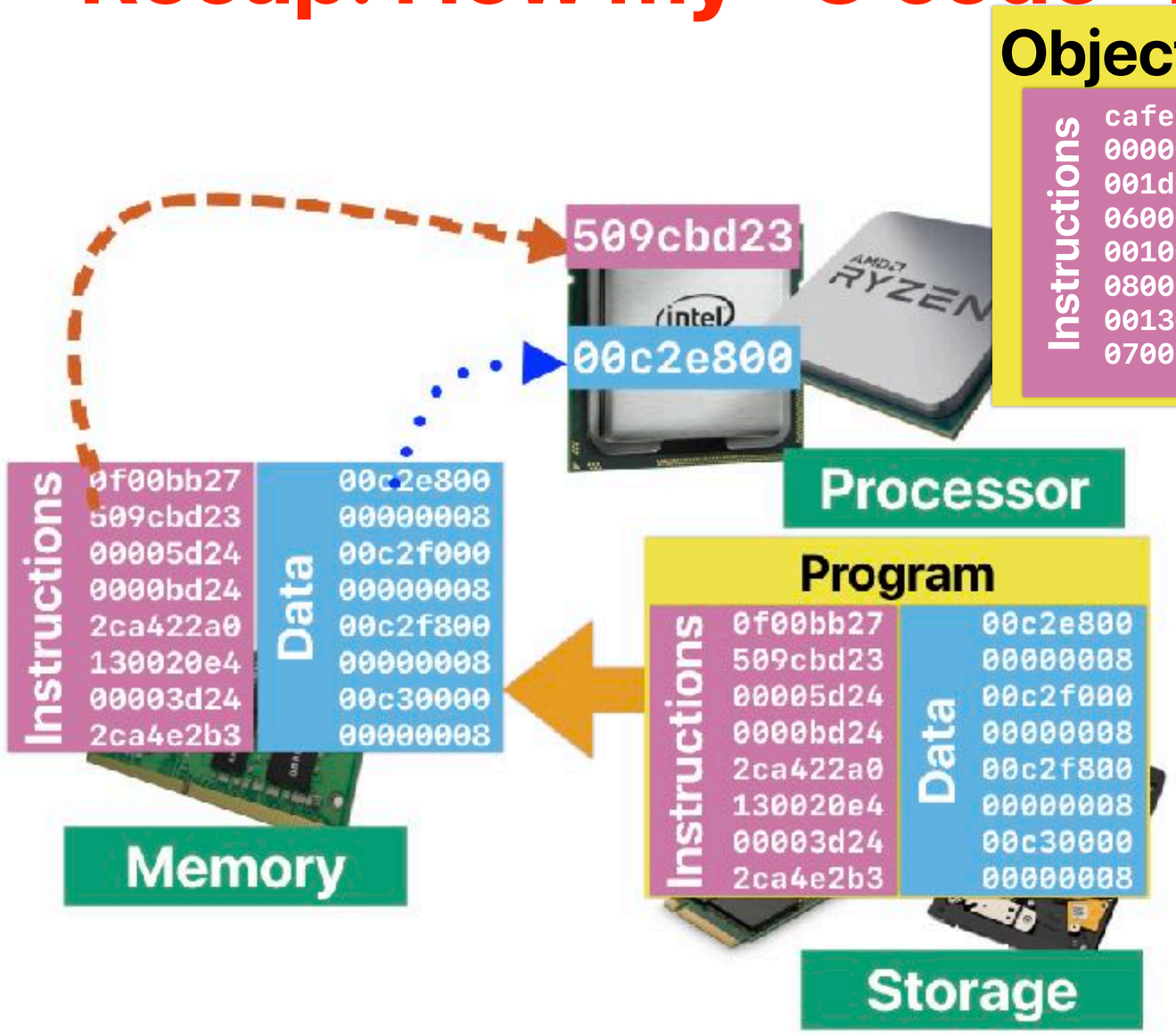
```
==========================================
Subject: setting CPU speed on running linux system


If the OS is Linux, you can manually control the CPU speed by reading and writing some virtual files in the "/proc"

1.) Is the system capable of software CPU speed control?
If the "directory" /sys/devices/system/cpu/cpu0/cpufreq exists, speed is controllable.
-- If it does not exist, you may need to go to the BIOS and turn on EIST and any other C and P state control and vi



2.) What speed is the box set to now?
Do the following:
$ cd /sys/devices/system/cpu
$ cat ./cpu0/cpufreq/cpuinfo_max_freq
3193000
$ cat ./cpu0/cpufreq/cpuinfo_min_freq
1596000

3.) What speeds can I set to?
Do
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
It will list highest settable to lowest; example from my NHM "Smackover" DX58SO HEDT board, I see:
3193000 3192000 3059000 2926000 2793000 2660000 2527000 2394000 2261000 2128000 1995000 1862000 1729000 159600
You can choose from among those numbers to set the "high water" mark and "low water" mark for speed.  If you set "h


4.) Show me how to set all to highest settable speed!
Use the following little sh/ksh/bash script:
$ cd /sys/devices/system/cpu  # a virtual directory made visible by device drivers
$ newSpeedTop=`awk '{print $1}' ./cpu0/cpufreq/scaling_available_frequencies`
$ newSpeedLow=$newSpeedTop  # make them the same in this example
$ for c in ./cpu[0-9]* ; do
>   echo $newSpeedTop >${c}/cpufreq/scaling_max_freq
>   echo $newSpeedLow >${c}/cpufreq/scaling_min_freq
> done
$

5.) How do I return to the default - i.e. allow machine to vary from highest to lowest?
Edit line # 3 of the script above, and re-run it.  Change the line:
$ newSpeedLow=$newSpeedTop  # make them the same in this example
```

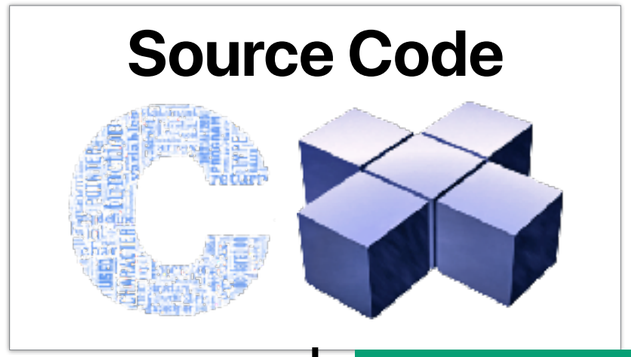
Programming languages

- How many instructions are there in “Hello, world!”

	Instruction count	LOC	Ranking
C	600k	6	1
C++	3M	6	2
Java	~210M	8	5
Perl	10M	4	3
Python	~30M	1	4

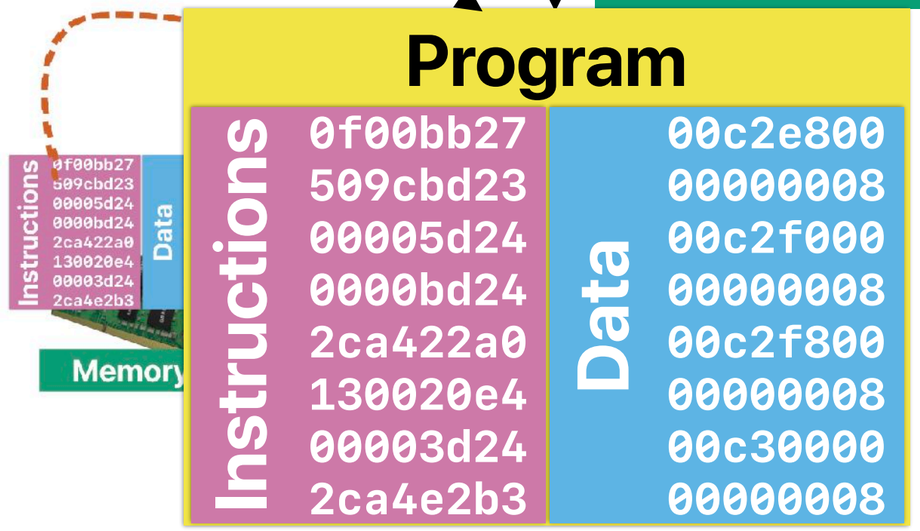
Recap: How my "C code" becomes a "program"





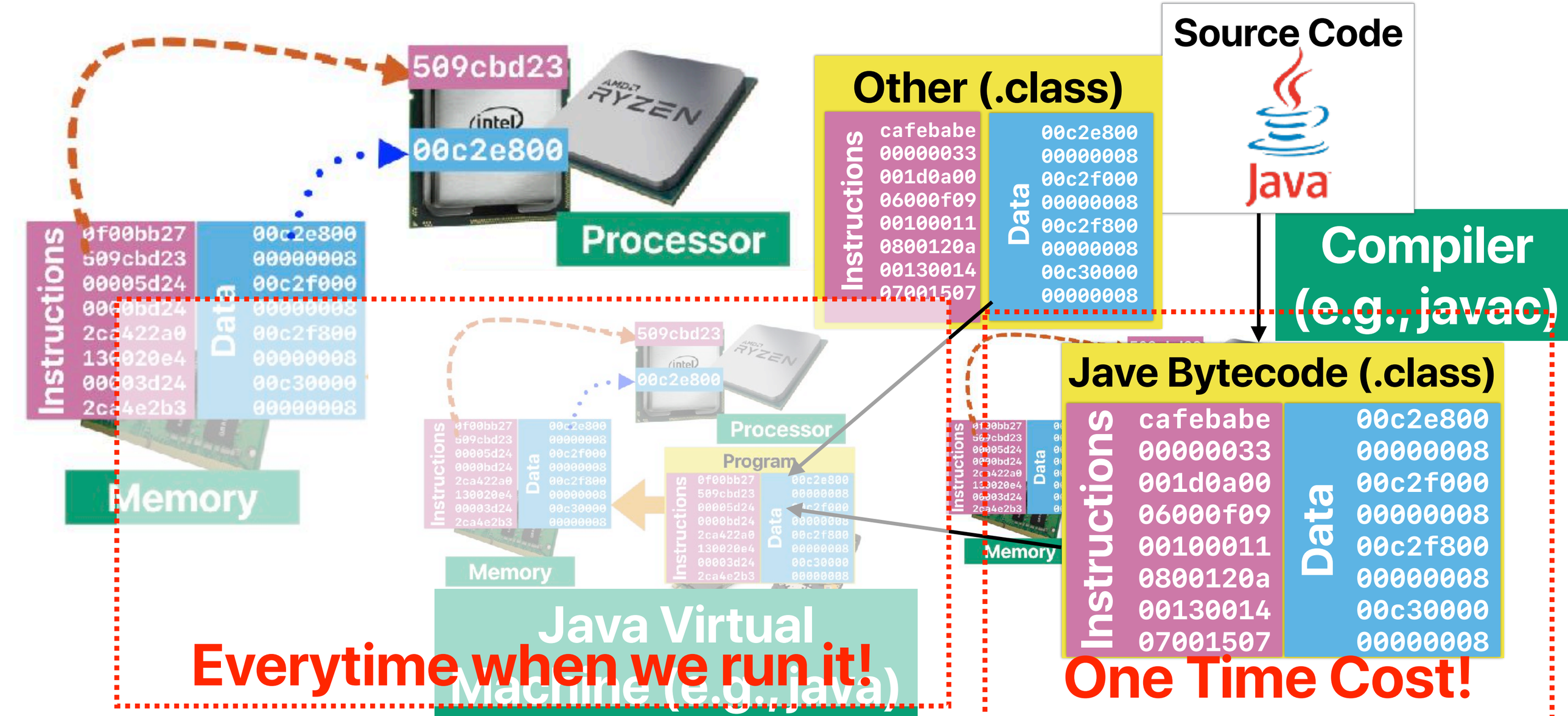
Linker

Compiler (e.g., gcc)

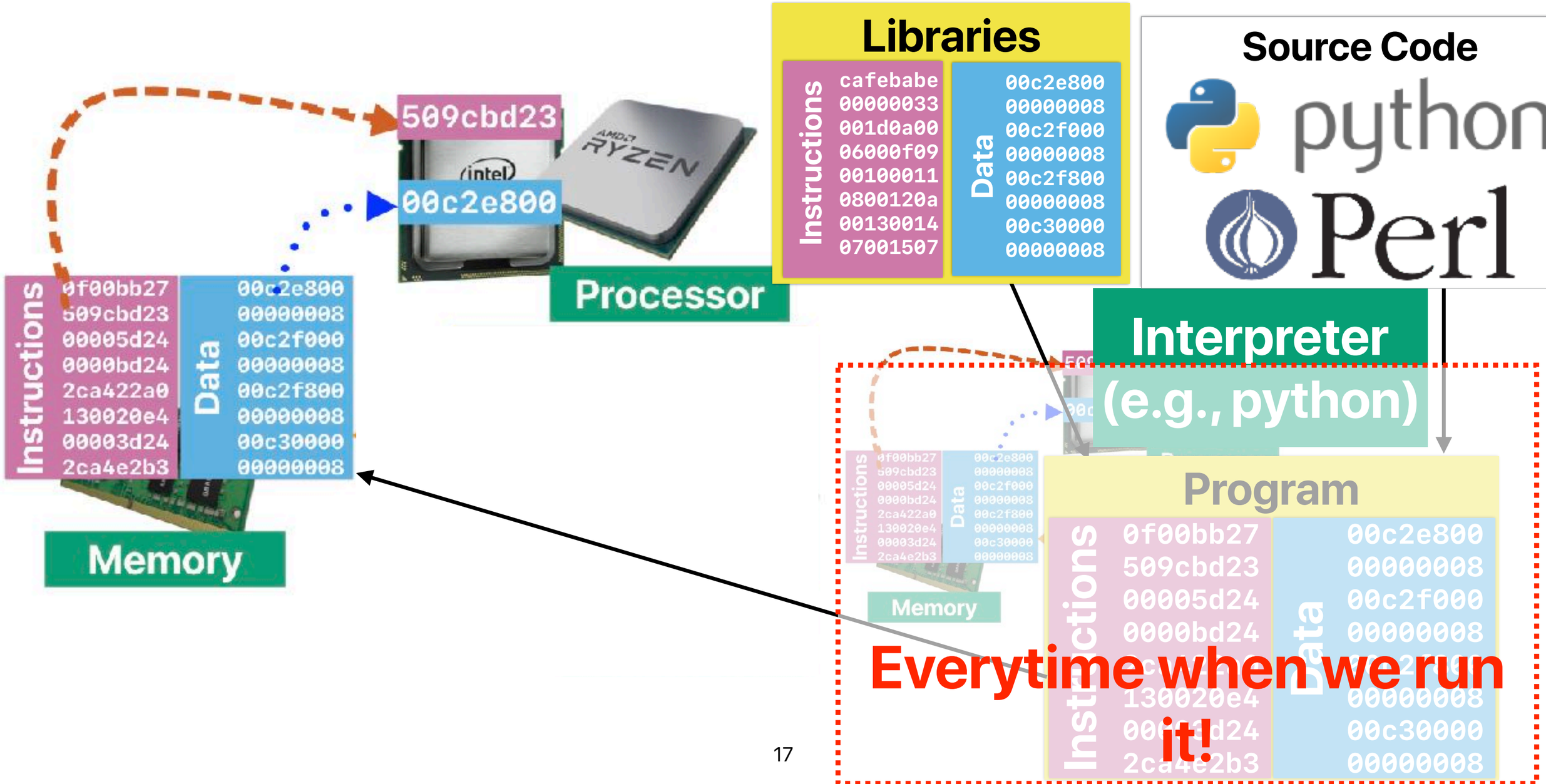


One Time Cost!

Recap: How my "Java code" becomes a "program"



Recap: How my "Python code" becomes a "program"



Revisited the demo with compiler optimizations!

- gcc has different optimization levels.
  - -O0 — no optimizations
  - -O3 — typically the best-performing optimization

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

Demo revisited — compiler optimization

- Compiler can reduce the instruction count, change CPI — with “limited scope”
- Compiler CANNOT help improving “crummy” source code

```
if(option)
    std::sort(data, data + arraySize);
```

Compiler can never add this — only the programmer can!

```
for (unsigned c = 0; c < arraySize*1000; ++c) {
        if (data[c%arraySize] >= INT_MAX/2)
            sum ++;
    }
}
```

How many operations: CISC v.s. RISC

- CISC (Complex Instruction Set Computing)
  - Examples: x86, Motorola 68K
  - Provide **many powerful/complex** instructions
    - Many: more than 1503 instructions since 2016
    - Powerful/complex: an instruction can perform both ALU and memory operations
    - Each instruction takes more cycles to execute
- RISC (Reduced Instruction Set Computer)
  - Examples: ARMv8, RISC-V, MIPS (the first RISC instruction, invented by the authors of our textbook)
  - Each instruction only performs simple tasks
  - Easy to decode
  - Each instruction takes less cycles to execute

RISC-V v.s. x86

- Using the same language, the same source code, regarding the compiled program on x86 and RISC-V, how many of the following statements is/are "generally" correct?
  - ① The RISC-V version would contain more instructions than its x86 version
  - ② The RISC-V version tends to incur fewer memory accesses than its x86 version
  - ③ The RISC-V version needs a processor with higher clock rate than its x86 version if the CPI of both versions are similar
  - ④ The RISC-V version needs a processor with lower CPI than its x86 version if the x86 processor runs at the same clock rate

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Amdahl's Law — and It's Implication in the Multicore Era

H&P Chapter 1.9

M. D. Hill and M. R. Marty. Amdahl's Law in the Multicore Era. In Computer, vol. 41, no. 7, pp. 33-38, July 2008.

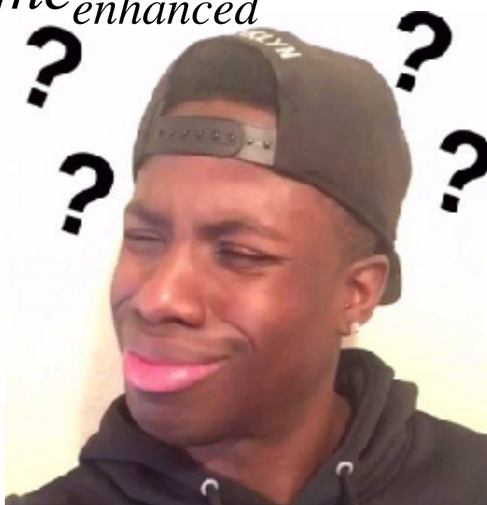
Amdahl's Law

$$Speedup_{enhanced}(f, s) = \frac{1}{(1 - f) + \frac{f}{s}}$$

f — The fraction of time in the original program

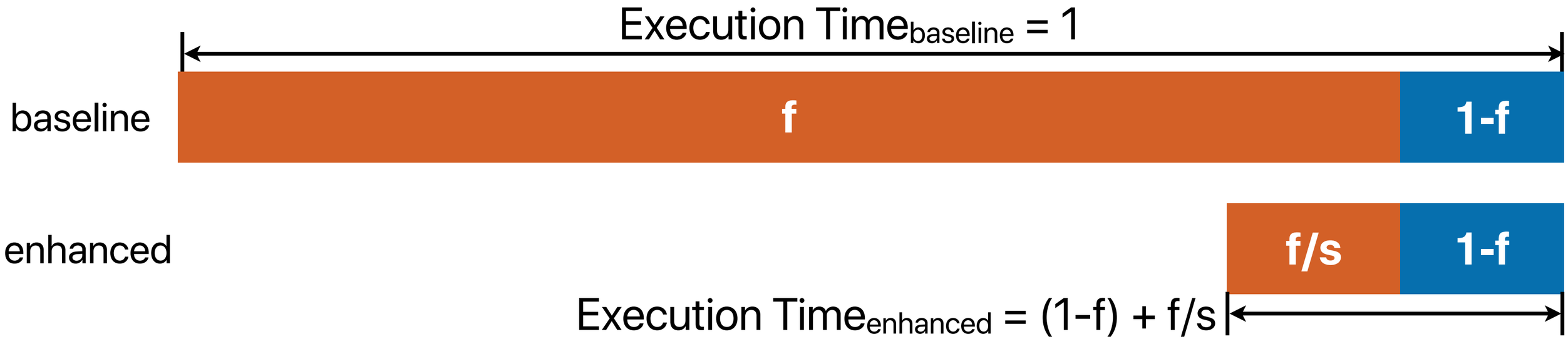
s — The speedup we can achieve on f

$$Speedup_{enhanced} = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}}$$



Amdahl's Law

$$Speedup_{enhanced}(f, s) = \frac{1}{(1 - f) + \frac{f}{s}}$$

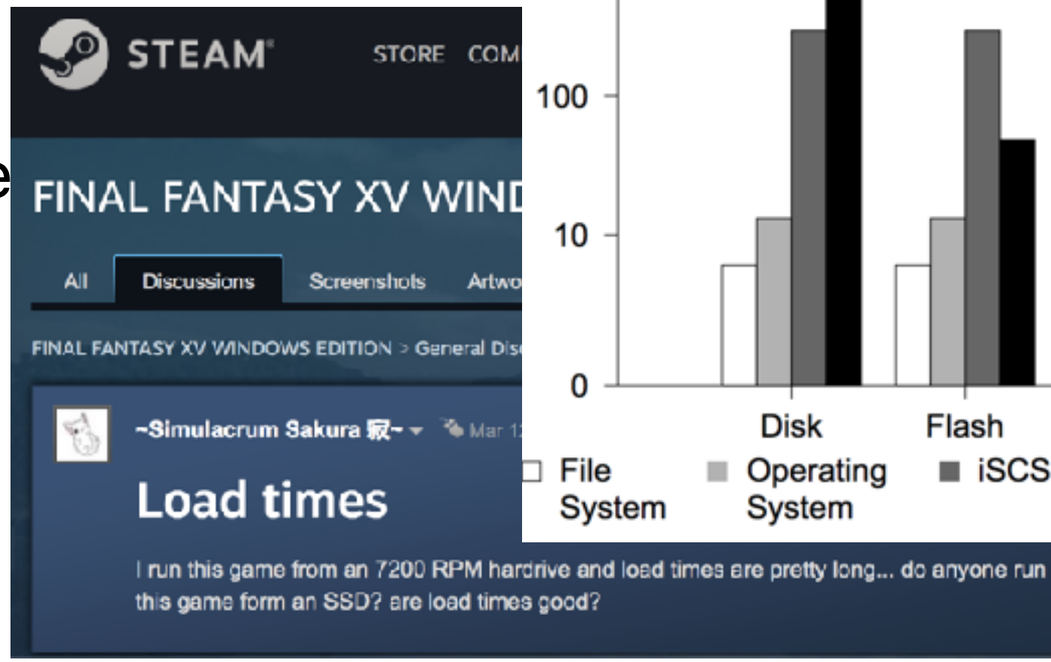


$$Speedup_{enhanced} = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}} = \frac{1}{(1 - f) + \frac{f}{s}}$$

Practicing Amdahl's Law

- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time. By how much can we speed up the map loading process?

  A. ~7x
  B. ~10x
  C. ~17x
  D. ~29x
  E. ~100x



$$Speedup_{enhanced}(95\,\%, 100) = \frac{1}{(1 - 95\%) + \frac{95\,\%}{100}} = 16.81\times$$

Amdahl's Law on 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:

$$Speedup_{enhanced}(f_{Opt1}, f_{Opt2}, s_{Opt1}, s_{Opt2}) = \frac{1}{(1 - f_{Opt1} - f_{Opt2}) + \frac{f\_Opt1}{s\_Opt1} + \frac{f\_Opt2}{s\_Opt2}}$$

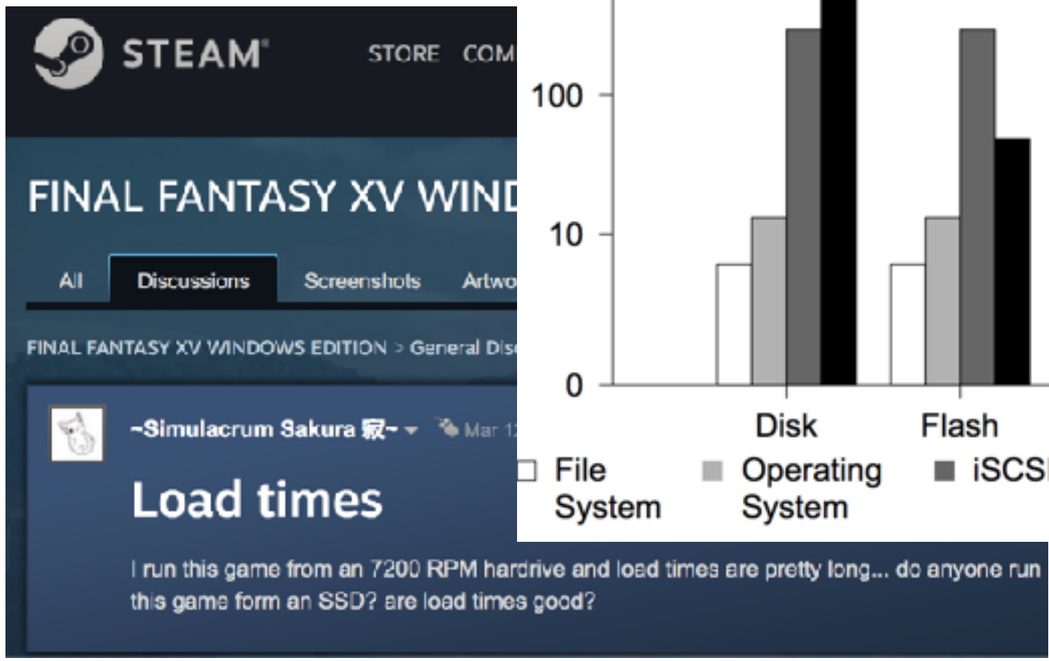
  - If optimization #1 and optimization #2 are not dis-joint:

$$Speedup_{enhanced}(f_{OnlyOpt1}, f_{OnlyOpt2}, f_{BothOpt1Opt2}, s_{OnlyOpt1}, s_{OnlyOpt2}, s_{BothOpt1Opt2}) = \frac{1}{(1 - f_{OnlyOpt1} - f_{OnlyOpt2} - f_{BothOpt1Opt2}) + + \frac{f\_BothOpt1Opt2}{s\_BothOpt1Opt2} + \frac{f\_OnlyOpt1}{s\_OnlyOpt1} + \frac{f\_OnlyOpt2}{s\_OnlyOpt2}}$$

Practicing Amdahl's Law (2)

- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time and a better processor to accelerate the software overhead by 2x. By how much can we speed up the map loading process?

  A. ~7x
  B. ~10x
  C. ~17x
  D. ~29x
  E. ~100x



$$Speedup_{enhanced}(95\,\%, 5\,\%, 100, 2) = \frac{1}{(1 - 95\% - 5\%) + \frac{95\,\%}{100} + \frac{5\,\%}{2}} = 28.98\times$$

Amdahl's Law Corollary #1

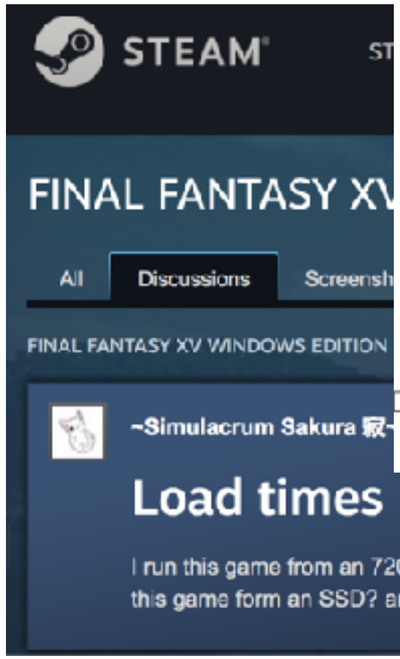
- The maximum speedup is bounded by

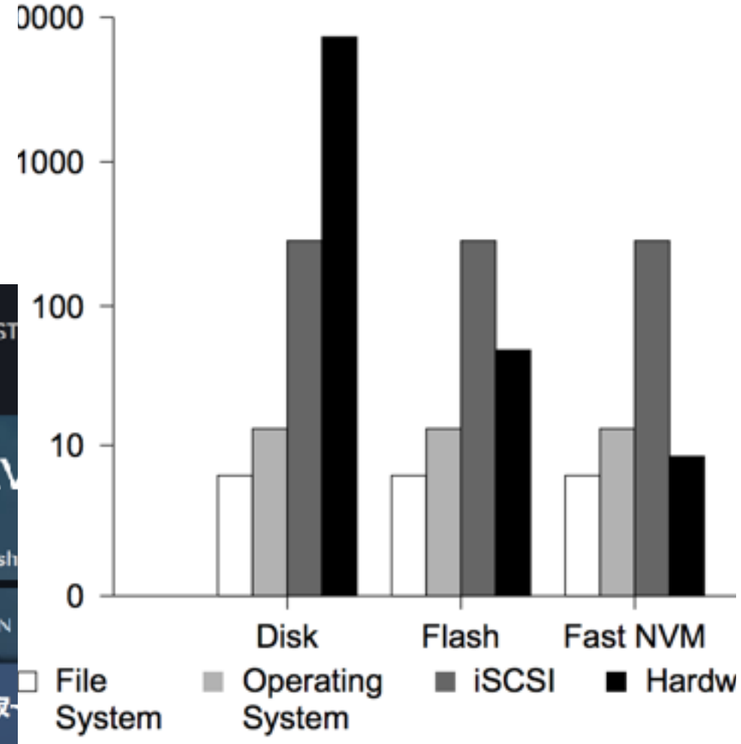
$$Speedup_{max}(f, \infty) = \frac{1}{(1-f) + \frac{f}{\infty}}$$

$$Speedup_{max}(f, \infty) = \frac{1}{(1-f)}$$

Speedup further!

- With the latest flash memory technologies, the system spends 16% of time on accessing the flash, and the software overhead is now 84%. If we want to adopt a new memory technology to replace flash to achieve 2x speedup on loading maps, how much faster the new technology needs to be?
  - A. ~5x
  - B. ~10x
  - C. ~20x
  - D. ~100x
  - E. None of the above





$$Speedup_{max}(16\,\%\,, \infty) = \frac{1}{(1 - 16\%)} = 1.19$$

2x is not possible

Corollary #1 on Multiple Optimizations

- If we can pick just one thing to work on/optimize

$$Speedup_{max}(f_1, \infty) = \frac{1}{(1 - f_1)}$$
$$Speedup_{max}(f_2, \infty) = \frac{1}{(1 - f_2)}$$
$$Speedup_{max}(f_3, \infty) = \frac{1}{(1 - f_3)}$$
$$Speedup_{max}(f_4, \infty) = \frac{1}{(1 - f_4)}$$

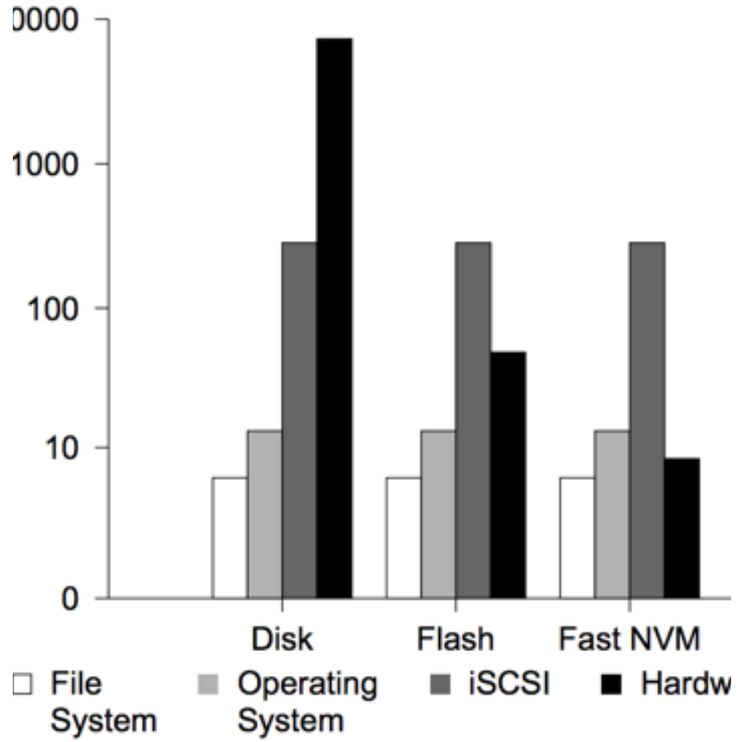
The biggest f_x would lead to the largest $Speedup_{max}$!

Corollary #2 — make the common case fast!

- When f is small, optimizations will have little effect.
- Common == **most time consuming** not necessarily the most frequent
- The uncommon case doesn’t make much difference
- The common case can change based on inputs, compiler options, optimizations you’ve applied, etc.

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.
- — You have to revisit "Amdahl's Law" every time you applied some optimization



Moneta: A High-Performance Storage Array Architecture for Next-Generation, Non-volatile Memories Adrian M. Caulfield, Arup De, Joel Coburn, Todor I. Mollov, Rajesh K. Gupta, and Steven Swanson Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, 2010.

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

$$ET_{new} = \frac{ET_{old} \times 90\%}{9} + ET_{old} \times 10\% \times 10$$

$$ET_{new} = 1.1 \times ET_{old}$$

$$Speedup = \frac{ET_{old}}{ET_{new}} = \frac{ET_{old}}{1.1 \times ET_{old}} = 0.91 \times$$ .....slowdown!

You may not use Amdahl’s Law for this case as Amdahl’s Law does NOT
(1) consider overhead
(2) bound to slowdown

Amdahl's Law on Multicore Architectures

- Symmetric multicore processor with n cores (if we assume the processor performance scales perfectly)

$$Speedup_{parallel}(f_{parallelizable}, n) = \frac{1}{(1 - f_{parallelizable}) + \frac{f\_parallelizable}{n}}$$

Corollary #3, Corollary #4 & Corollary #5

$$Speedup_{parallel}(f_{parallelizable}, \infty) = \frac{1}{(1 - f_{parallelizable}) + \frac{f\_parallelizable}{\infty}}$$

$$Speedup_{parallel}(f_{parallelizable}, \infty) = \frac{1}{(1 - f_{parallelizable})}$$

- Single-core performance still matters — it will eventually dominate the performance
- Finding more "parallelizable" parts is also important
- If we can build a processor with unlimited parallelism — the complexity doesn't matter as long as the algorithm can utilize all parallelism — that's why bitonic sort works!

“Fair” Comparisons

Andrew Davison. Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers. In Humour the Computer, MITP, 1995

Is TFLOPS (Tera FLoating-point Operations Per Second) a good metric?

$$TFLOPS = \frac{\#\ of\ floating\ point\ instructions \times 10^{-12}}{Exection\ Time}$$

$$= \frac{IC \times \%\ of\ floating\ point\ instructions \times 10^{-12}}{IC \times CPI \times CT}$$

$$= \frac{\%\ of\ floating\ point\ instructions \times 10^{-12}}{CPI \times CT}$$

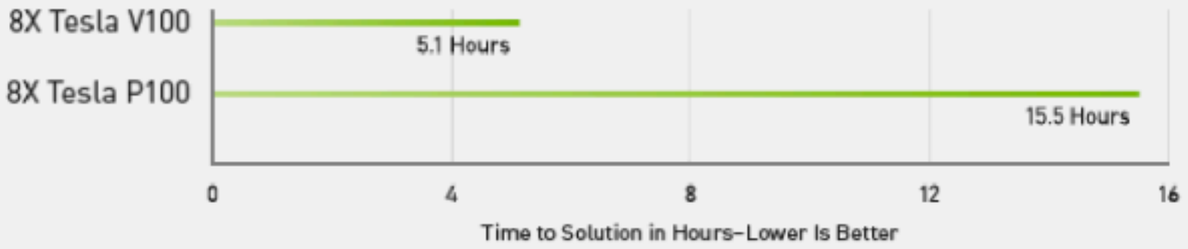
IC is gone!

- 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

12 ways to Fool the Masses When Giving Performance Results on Parallel Computers

- Quote only 32-bit performance results, not 64-bit results.
- Present performance figures for an inner kernel, and then represent these figures as the performance of the entire application.
- Quietly employ assembly code and other low-level language constructs.
- Scale up the problem size with the number of processors, but omit any mention of this fact.
- Quote performance results projected to a full system.
- Compare your results against scalar, unoptimized code on Crays.
- When direct run time comparisons are required, compare with an old code on an obsolete system.
- If MFLOPS rates must be quoted, base the operation count on the parallel implementation, not on the best sequential implementation.
- Quote performance in terms of processor utilization, parallel speedups or MFLOPS per dollar.
- Mutilate the algorithm used in the parallel implementation to match the architecture.
- Measure parallel run times on a dedicated system, but measure conventional run times in a busy environment.
- If all else fails, show pretty pictures and animated videos, and don't talk about performance.

Deep Learning Training in Less Than a Workday



Server Config: Dual Xeon E5-2699 v4 2.6 GHz | 8X NVIDIA® Tesla® P100 or V100 | ResNet-50 Training on MXNet for 90 Epochs with 1.28M ImageNet Dataset.

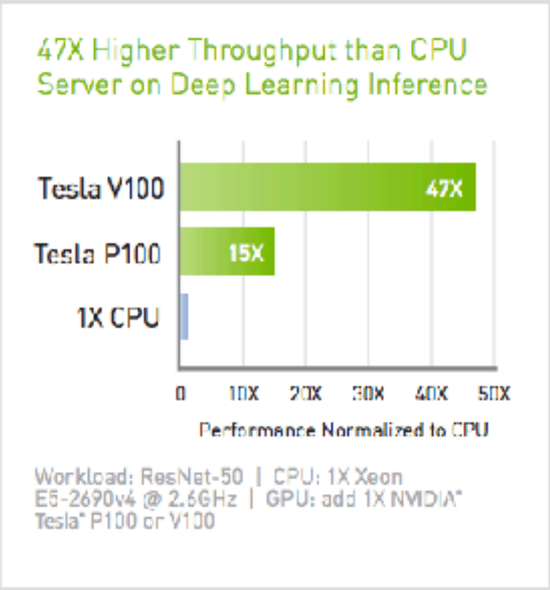
AI TRAINING

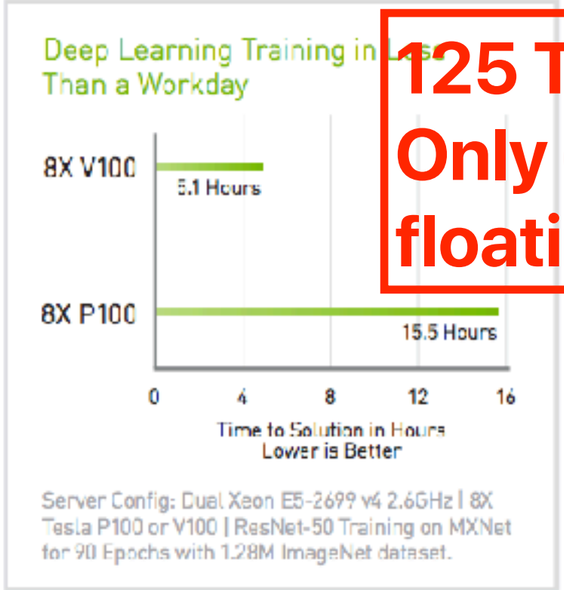
From recognizing speech to training virtual personal assistants and teaching autonomous cars to drive, data scientists are taking on increasingly complex challenges with AI. Solving these kinds of problems requires training deep learning models that are exponentially growing in complexity, in a practical amount of time.

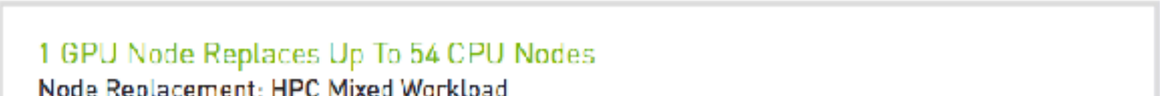
With 640 Tensor Cores, Tesla V100 is the world's first GPU to break the 100 teraFLOPS (TFLOPS) barrier of deep learning performance. The next generation of NVIDIA NVLink™ connects multiple V100 GPUs at up to 300 GB/s to create the world's most powerful computing servers. AI models that would consume weeks of computing resources on previous systems can now be trained in a few days. With this dramatic reduction in training time, a whole new world of problems will now be solvable with AI.

The Most Advanced Data Center GPU Ever Built.

NVIDIA® Tesla® V100 is the world's most advanced data center GPU ever built to accelerate AI, HPC, and graphics. Powered by NVIDIA Volta, the latest GPU architecture, Tesla V100 offers the performance of up to 100 CPUs in a single GPU—enabling data scientists, researchers, and engineers to tackle challenges that were once thought impossible.

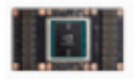






125 TFLOPS Only @ 16-bit floating point

SPECIFICATIONS



	Tesla V100 PCIe	Tesla V100 SXM2
GPU Architecture	NVIDIA Volta	
NVIDIA Tensor Cores	640	
NVIDIA CUDA® Cores	5,120	
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS
GPU Memory	32GB /16GB HBM2	
Memory Bandwidth	900GB/sec	
ECC	Yes	
Interconnect Bandwidth	32GB/sec	300GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink
Form Factor	PCIe Full Height/Length	SXM2
Max Power		

They try to tell it's the better AI hardware

https://blogs.nvidia.com/blog/2017/04/10/ai-drives-rise-accelerated-computing-datacenter/

	K80 2012	TPU 2015	P40 2016
Inferences/Sec <10ms latency	1/13X	1X	2X
Training TOPS	6 FP32	NA	12 FP32
Inference TOPS	6 FP32	90 INT8	48 INT8
On-chip Memory	16 MB	24 MB	11 MB
Power	300W	75W	250W
Bandwidth	320 GB/S	34 GB/S	350 GB/S

What's wrong with inferences per second?

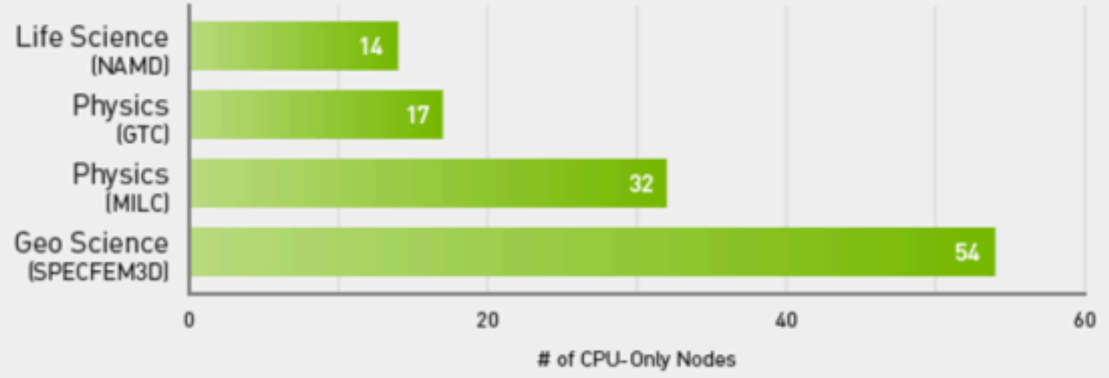
- There is no standard on how they inference
  - What model?
  - What dataset?
- That's why Facebook is trying to promote an AI benchmark — MLPerf

- *Pitfall: For NN hardware, Inferences Per Second (IPS) is an inaccurate summary performance metric.*

Our results show that IPS is a poor overall performance summary for NN hardware, as it's simply the inverse of the complexity of the typical inference in the application (e.g., the number, size, and type of NN layers). For example, the TPU runs the 4-layer MLP1 at 360,000 IPS but the 89-layer CNN1 at only 4,700 IPS, so TPU IPS vary by 75X! Thus, using IPS as the single-speed summary is *even more misleading* for NN accelerators than MIPS or FLOPS are for regular processors [23], so IPS should be even more disparaged. To compare NN machines better, we need a benchmark suite written at a high-level to port it to the wide variety of NN architectures. Fathom is a promising new attempt at such a benchmark suite [3].

1 GPU Node Replaces Up To 54 CPU Nodes

Node Replacement: HPC Mixed Workload



CPU Server: Dual Xeon Gold 6140@2.30GHz, GPU Servers: same CPU server w/ 4x V100 PCIe | CUDA Version: CUDA 9.x| Dataset: NAMD (STMV), GTC (mpi#proc.in), MILC (APEX Medium), SPECFEM3D (four_material_simple_model) | To arrive at CPU node equivalence, we use measured benchmark with up to 8 CPU nodes. Then we use linear scaling to scale beyond 8 nodes.

HIGH PERFORMANCE COMPUTING (HPC)

HPC is a fundamental pillar of modern science. From predicting weather to discovering drugs to finding new energy sources, researchers use large computing systems to simulate and predict our world. AI extends traditional HPC by allowing researchers to analyze large volumes of data for rapid insights where simulation alone cannot fully predict the real world.

Tesla V100 is engineered for the convergence of AI and HPC. It offers a platform for HPC systems to excel at both computational science for scientific simulation and data science for finding insights in data. By pairing NVIDIA CUDA® cores and Tensor Cores within a unified architecture, a single server with Tesla V100 GPUs can replace hundreds of commodity CPU-only servers for both traditional HPC and AI workloads. Every researcher and engineer can now afford an AI supercomputer to tackle their most challenging work.

Choose the right metric — Latency v.s. Throughput/Bandwidth

Latency v.s. Bandwidth/Throughput

- Latency — the amount of time to finish an operation
  - access time
  - response time
- Throughput — the amount of work can be done within a given period of time
  - bandwidth (MB/Sec, GB/Sec, Mbps, Gbps)
  - IOPs
  - MFLOPs

RAID — Improving throughput

MORE SPECS

Model Code (Capaci

Gener

DIMENSION (WxHxD)
100 X 69.85 X 6.8 (mm)

TRIM SUPPORT
Yes

ENCRYPTION SUPPORT
AES 256-bit Encryption (Class 0) TCG/Opal
IEEE1667 (Encrypted drive)

Performance[2]

SEQUENTIAL READ
Up to 550 MB/s

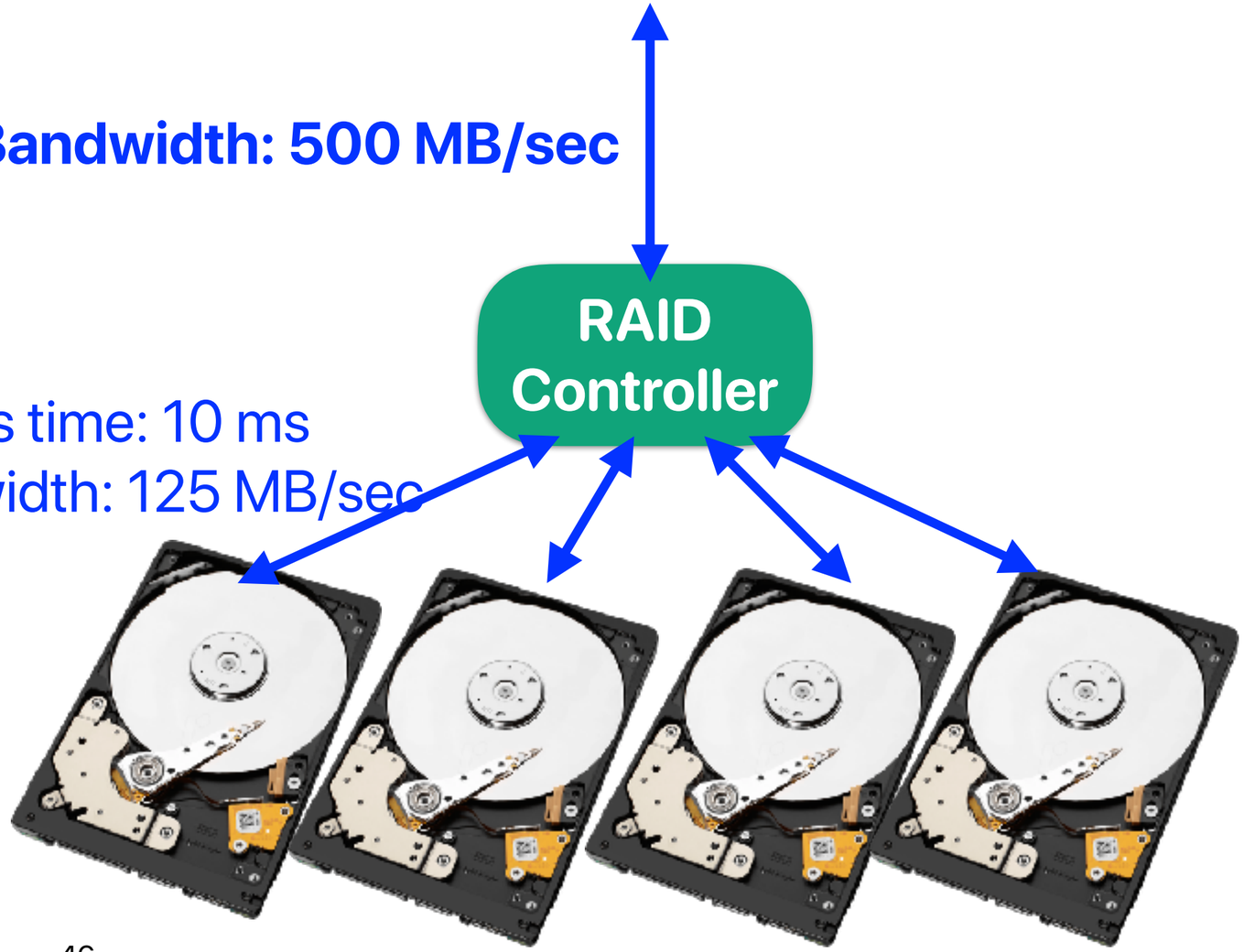
RANDOM WRITE (4KB, QD32)
Up to 89,000 IOPS

Environment

AVERAGE POWER CONSUMPTION (SYSTEM LEVEL)[3]
1,000 GB: Average 2.2 W Maximum 4.0 W
2,000 GB: Average 3.1 W Maximum 4.2 W
4,000 GB: Average 3.1 W Maximum 5.4 W
(Burst mode)

Aggregated Bandwidth: 500 MB/sec

Access time: 10 ms
Bandwidth: 125 MB/sec



The performance between RAID and SSD

- Compare (X) RAID consists of 4x H.D.D. where each has 10 ms access time and 125 MB/sec bandwidth — aggregated bandwidth at 500 MB/Sec (Y) a single SSD with 100 us access time and 550MB/Sec bandwidth. Both accept 4KB data as the smallest request size. If we want to load a program with 100KB code size, how much faster is Y over X at least?

  A. 1x — no speedup
  B. 1.1x
  C. 4x
  D. 4.4x
  E. 100x

$$ET_{HDD_{BestCase}} = 10\ ms$$

$$ET_{SSD_{worst}} = \frac{100KB}{4K} \times 100\ us = 2.5\ ms$$

Latency and Bandwidth trade-off

- Increase bandwidth can hurt the response time of a single task
- If you want to transfer a 2 Peta-Byte video from UCSD
  - 100 miles (161 km) from UCR
  - Assume that you have a 100Gbps ethernet
    - 2 Peta-byte over 167772 seconds = 1.94 Days
    - 22.5TB in 30 minutes
    - Bandwidth: 100 Gbps

Memory Gap Problem

Moore's Law(1)

Present and future

By integrated electronics, I mean
technologies which are referred to
tronics today as well as any additi
result in electronics functions suppli
to miniaturize electronics equipment
creasingly complex electronic functi
space with minimum weight. Sever
evolved, including microassembly
individual components, thin-film
semiconductor integrated circuits.

ICs are increasingly p

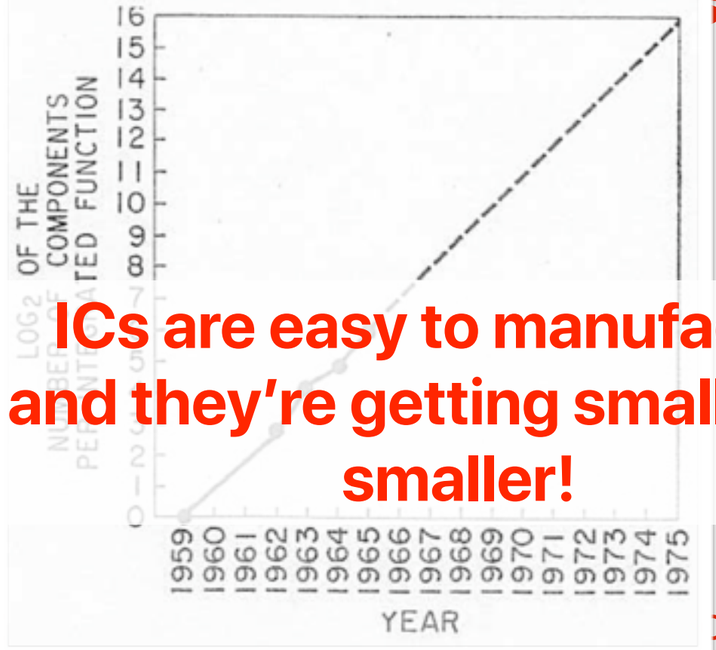
The establishment

ished
or nev
and v
vable
llo, fo
he reli
that
ilure as the

stablish

Increasing the yield

There is no fundamental obstacle to achieving device yields of 100%. At present, packaging costs so far exceed the cost of the semiconductor structure itself that there is no incentive to improve yields, but they can be raised as high as is economically justified. No barrier exists comparable to the thermodynamic equilibrium considerations



ICs are easy to manufacture and they're getting smaller and smaller!

Two-mil squares

With the dimensional tolerances already being
employed in integrated circuits, isolated high-per
formance transistors can be built on centers two
thousandths of an inch apart. Such a two-mil square
can also contain several kilohms of resistance o

ICs are small

Linear circuitry

Integration will not change linear systems as radically as digital systems. Still, a considerable degree of integration will be achieved with linear

ICs are widely applicable

Reliability coun

In almost e
demonstrated h
level of production—low compared to that of discrete components—it offers reduced systems cost, and in many systems improved performance has been realized.

ICs are more reliable

Heat problem

Will it be possible to remove the heat generated by tens of thousands of components in a single silicon chip

Heat is a solvable issue

Moore's
impo
historic

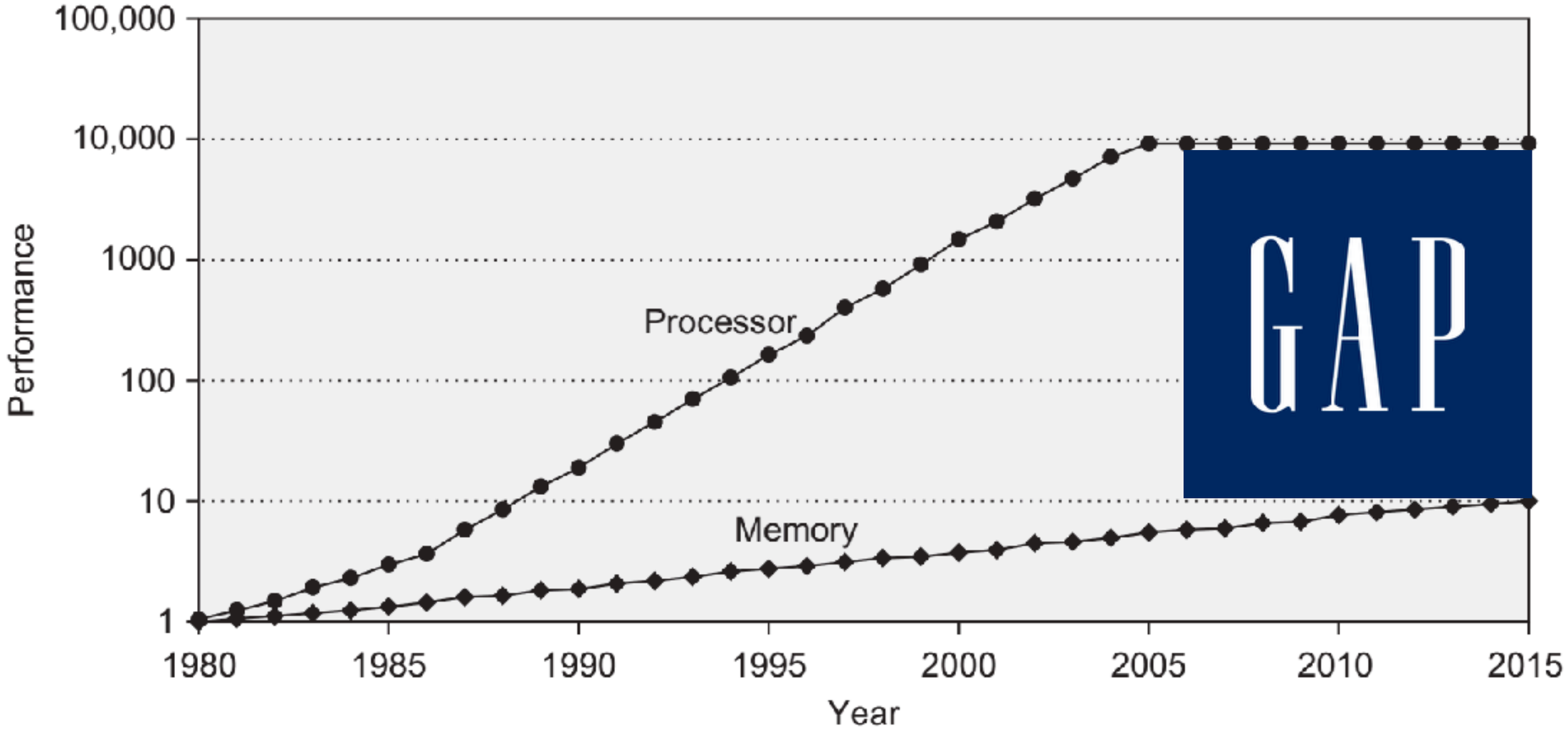
Day of reckoning

Clearly, we will be able to build such component-crammed equipment. Next, we ask under what circumstances we should do it. The total cost of making a particular system function must be minimized. To do so, we could amortize the engineering over several identical items, or evolve flexible techniques for the engineering of large functions so that no disproportionate expense need be borne by a particular array. Perhaps newly devised de-

Designing ICs can be easy

(1) Mo omponents onto integrated circuits', Electronics 38 (8) .

Performance gap between Processor/Memory



Performance of modern DRAM

			Best case access time (no precharge)			Precharge needed
Production year	Chip size	DRAM type	RAS time (ns)	CAS time (ns)	Total (ns)	Total (ns)
2000	256M bit	DDR1	21	21	42	63
2002	512M bit	DDR1	15	15	30	45
2004	1G bit	DDR2	15	15	30	45
2006	2G bit	DDR2	10	10	20	30
2010	4G bit	DDR3	13	13	26	39
2016	8G bit	DDR4	13	13	26	39

Figure 2.4 Capacity and access times for DDR SDRAMs by year of production. Access time is for a random memory word and assumes a new row must be opened. If the row is in a different bank, we assume the bank is precharged; if the row is not open, then a precharge is required, and the access time is longer. As the number of banks has increased, the ability to hide the precharge time has also increased. DDR4 SDRAMs were initially expected in 2014, but did not begin production until early 2016.

The impact of "slow" memory

- Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has "perfect" memory, the CPI is just 1. Now, consider we have DDR4 and the program is well-behaved that precharge is never necessary — the access latency is simply 26 ns. What's the average CPI (pick the most close one)?

  A. 9
  B. 17
  C. 27
  D. 35
  E. 69

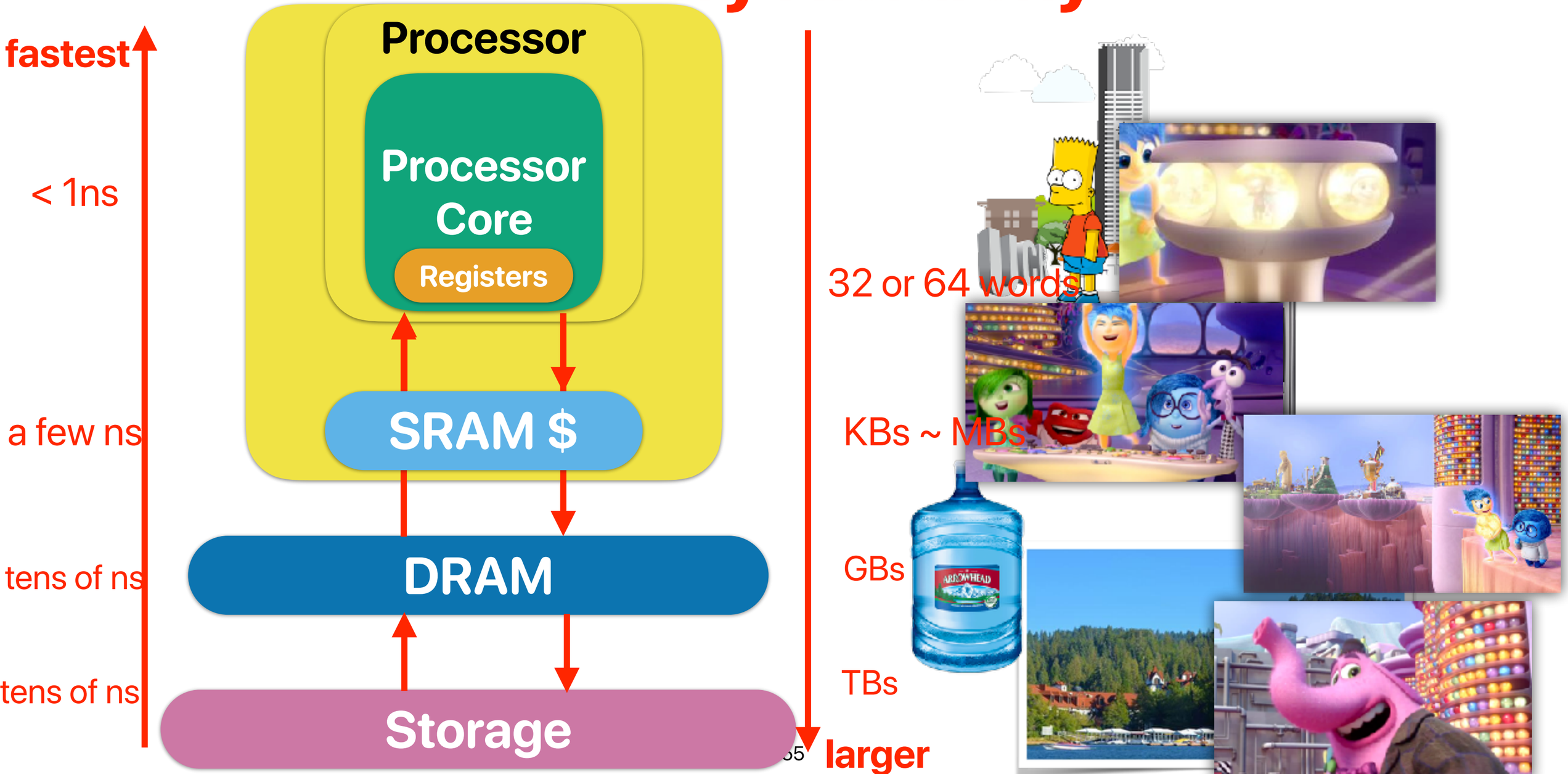
$$1 + \underbrace{100\% \times (52)}_{\text{Fetch Instructions}} + \underbrace{30\% \times 52}_{\text{Access Data}} = 68.6\ cycles$$

Alternatives?

Memory technology	Typical access time	$ per GiB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

Fast, but expensive $$$

Memory Hierarchy



How can memory hierarchy help in performance?

- Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has "perfect" memory, the CPI is just 1. Now, in addition to DDR4, whose latency 26 ns, we also got an SRAM cache with latency of just at 0.5ns and can capture 90% of the desired data/instructions. what's the average CPI (pick the most close one)?

A. 2

B. 4

C. 8

D. 16

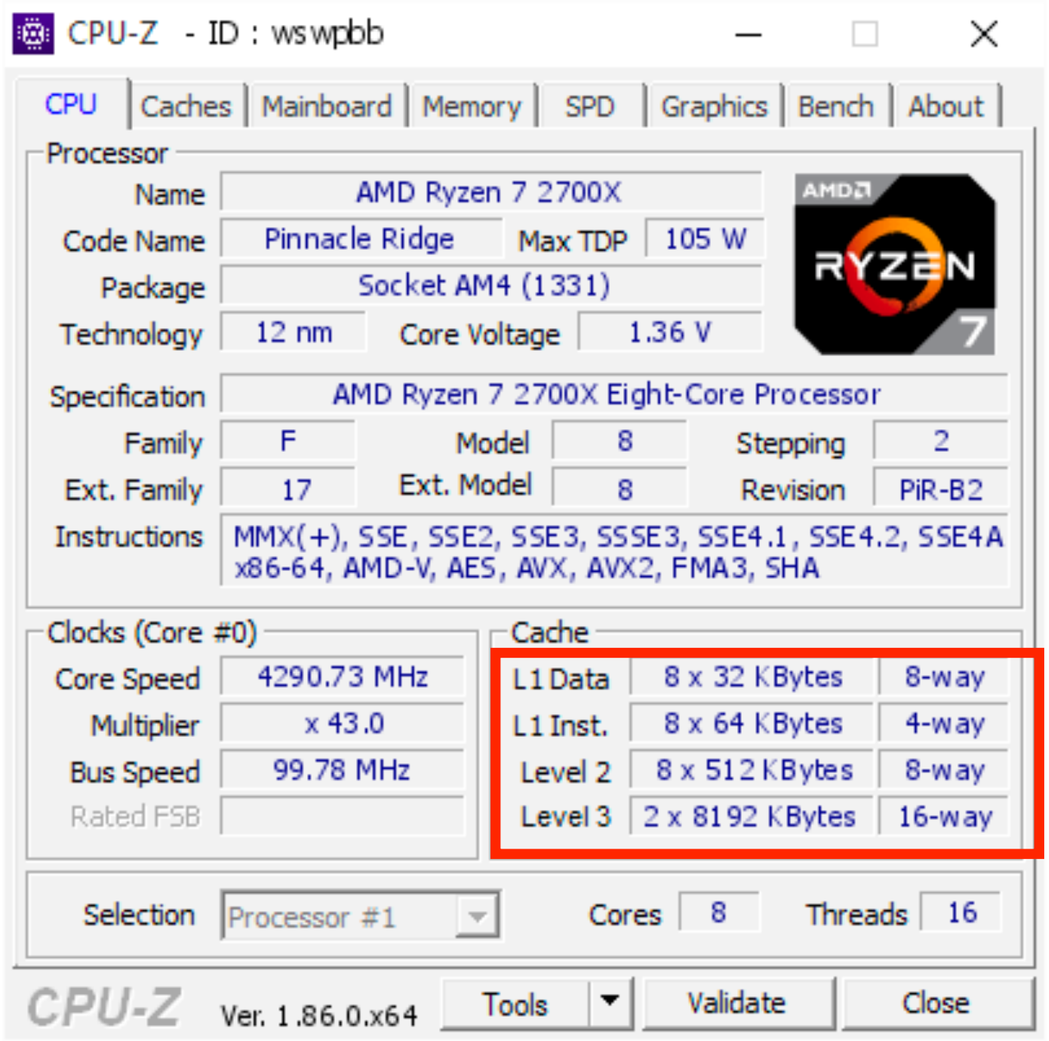
E. 32

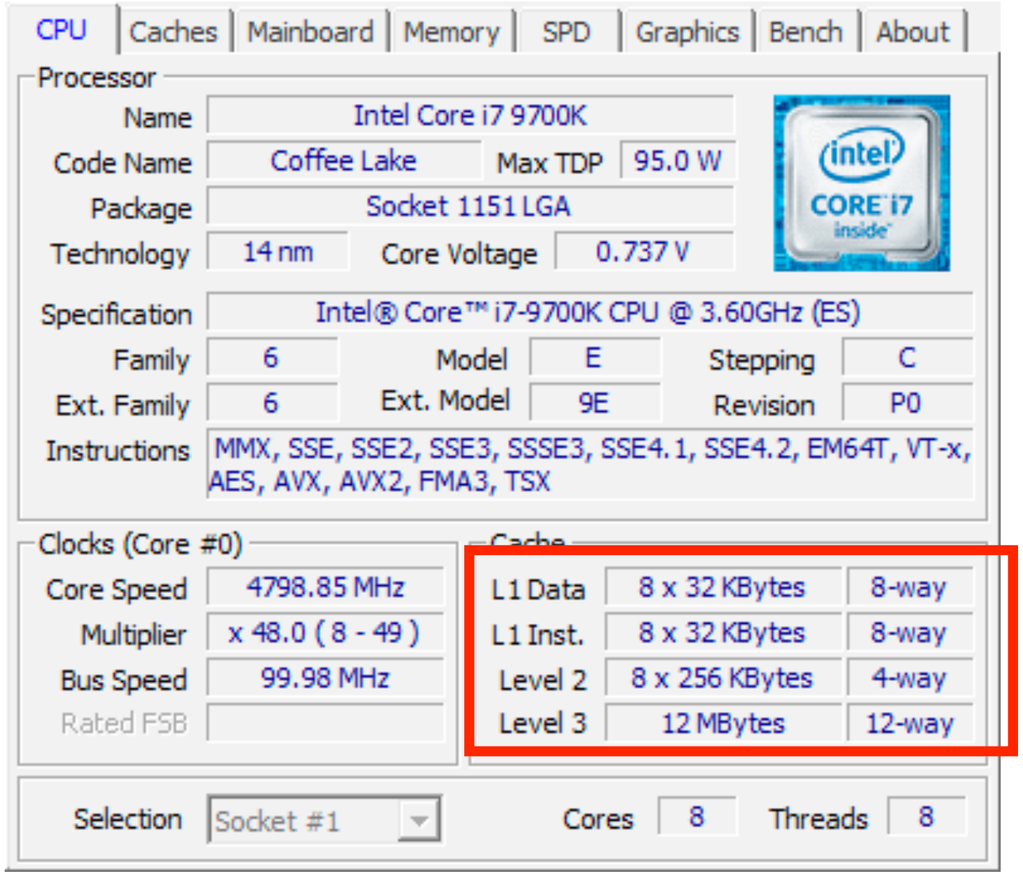
$$1 + (1 - 90\%) \times [100\% \times (52) + 30\% \times 52] = 7.76\ cycles$$

Fetch Instructions

Access Data

L1? L2? L3?


CPU-Z - ID : wswpbb
CPU
Caches
Mainboard
Memory
SPD
Graphics
Bench
About
Processor
Name
AMD Ryzen 7 2700X
Code Name
Pinnacle Ridge
Max TDP
105 W
Package
Socket AM4 (1331)
Technology
12 nm
Core Voltage
1.36 V
Specification
AMD Ryzen 7 2700X Eight-Core Processor
Family
F
Model
8
Stepping
2
Ext. Family
17
Ext. Model
8
Revision
PiR-B2
Instructions
MMX(+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A
x86-64, AMD-V, AES, AVX, AVX2, FMA3, SHA
Clocks (Core #0)
Core Speed
4290.73 MHz
Multiplier
x 43.0
Bus Speed
99.78 MHz
Rated FSB
Cache
L1 Data
8 x 32 KBytes
8-way
L1 Inst.
8 x 64 KBytes
4-way
Level 2
8 x 512 KBytes
8-way
Level 3
2 x 8192 KBytes
16-way
Selection
Processor #1
Cores
8
Threads
16
CPU-Z
Ver. 1.86.0.x64
Tools
Validate
Close


CPU
Caches
Mainboard
Memory
SPD
Graphics
Bench
About
Processor
Name
Intel Core i7 9700K
Code Name
Coffee Lake
Max TDP
95.0 W
Package
Socket 1151 LGA
Technology
14 nm
Core Voltage
0.737 V
Specification
Intel® Core™ i7-9700K CPU @ 3.60GHz (ES)
Family
6
Model
E
Stepping
C
Ext. Family
6
Ext. Model
9E
Revision
P0
Instructions
MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x,
AES, AVX, AVX2, FMA3, TSX
Clocks (Core #0)
Core Speed
4798.85 MHz
Multiplier
x 48.0 (8 - 49)
Bus Speed
99.98 MHz
Rated FSB
Cache
L1 Data
8 x 32 KBytes
8-way
L1 Inst.
8 x 32 KBytes
8-way
Level 2
8 x 256 KBytes
4-way
Level 3
12 MBytes
12-way
Selection
Socket #1
Cores
8
Threads
8

Memory Hierarchy

fastest

< 1ns

a few ns

tens of ns

tens of ns

Processor

Processor Core

Registers

SRAM $

DRAM

Storage

TBs

larger

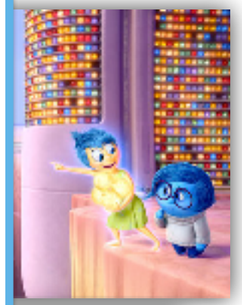
fastest

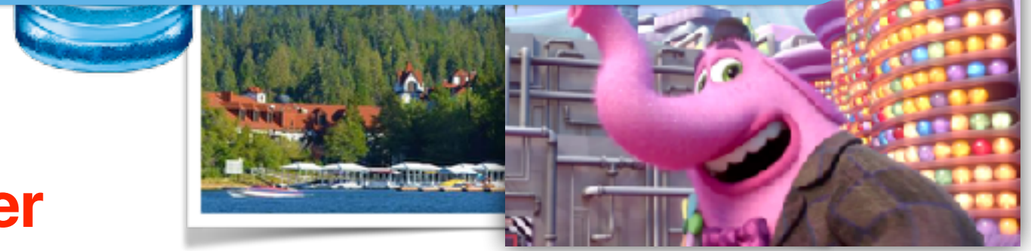
L1 $

L2 $

L3 $

larger





How can deeper memory hierarchy help in performance?

- Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has "perfect" memory, the CPI is just 1. Now, in addition to DDR4, whose latency 26 ns, we also got a 2-level SRAM caches with
  - it's 1st-level one at latency of 0.5ns and can capture 90% of the desired data/ instructions.
  - the 2nd-level at latency of 5ns and can capture 60% of the desired data/instructions

What's the average CPI (pick the most close one)?

A. 2
B. 4
C. 8
D. 16
E. 32

$$1 + (1 - 90\%) \times [10 + (1 - 60\%) \times 52 + 30\% \times (10 + (1 - 60\%) \times 52)] = 5\ cycles$$

Fetch Instructions

Access Data

Why adding small SRAMs would work?

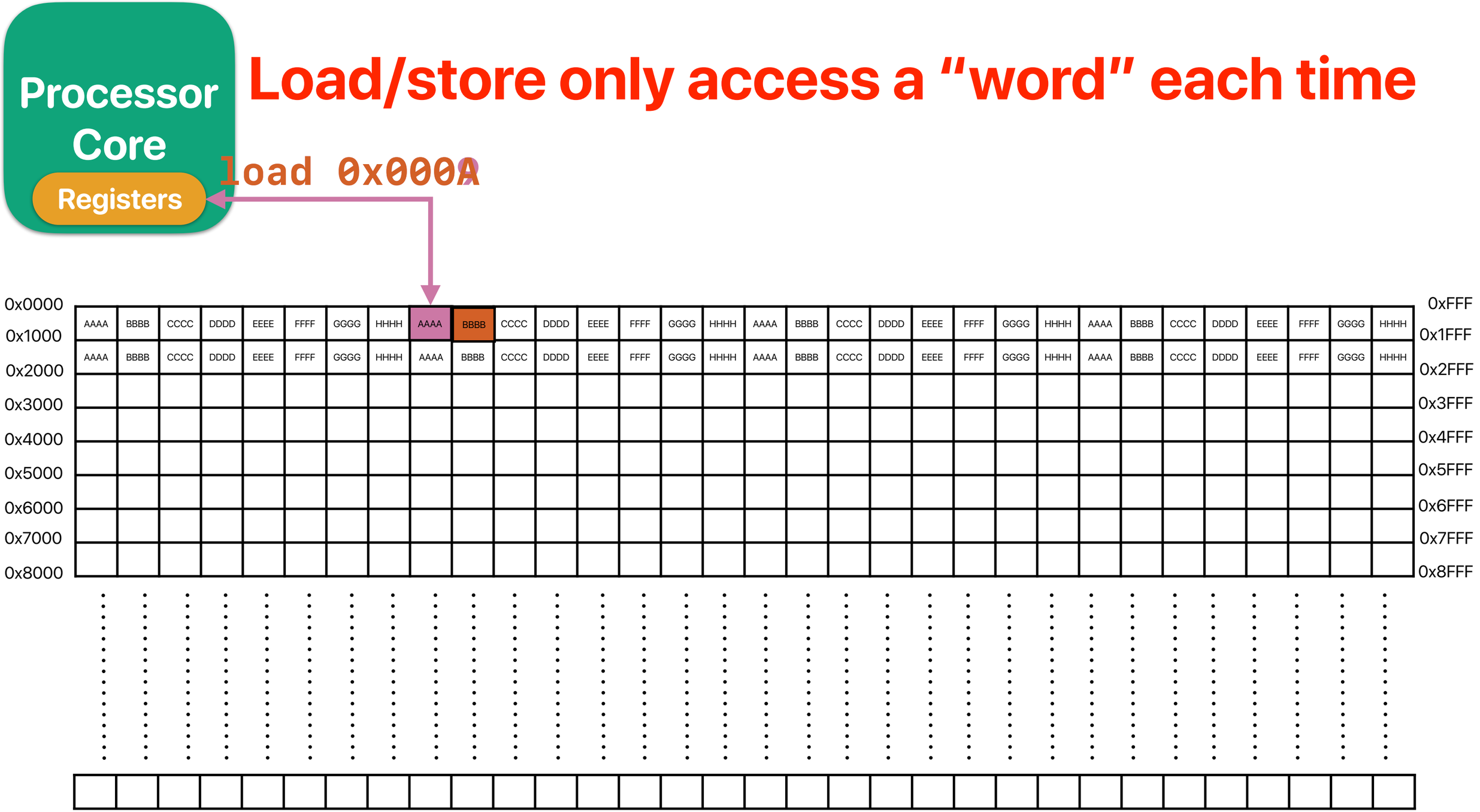
Locality

- Spatial locality — application tends to visit nearby stuffs in the memory
  - Code — the current instruction, and then PC + 4
  - Data — the current element in an array, then the next
- Temporal locality — application revisit the same thing again and again
  - Code — loops, frequently invoked functions
  - Data — the same data can be read/write many times

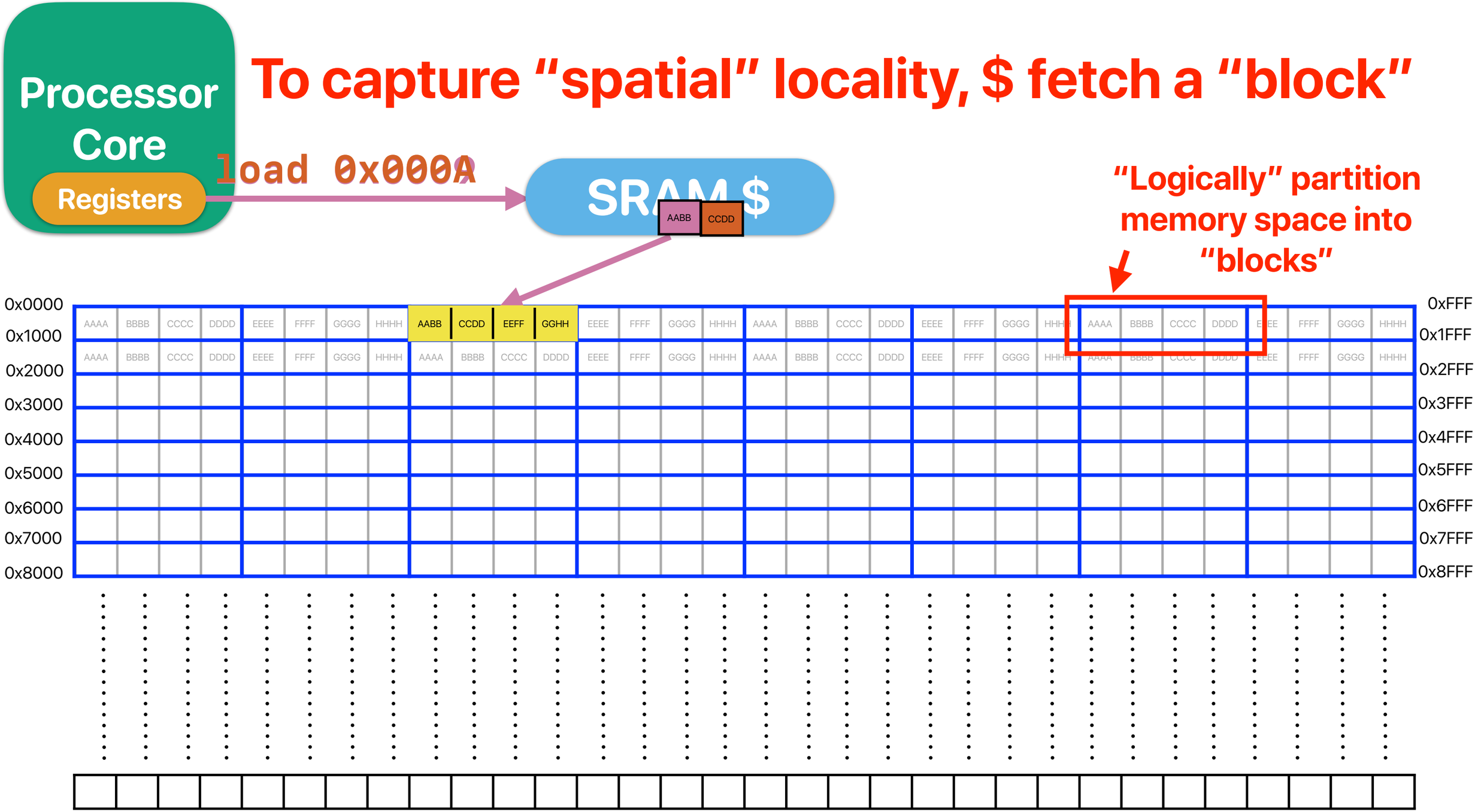
Most of time, your program is just visiting a very small amount of data/instructions within a given window

Architecting the Cache

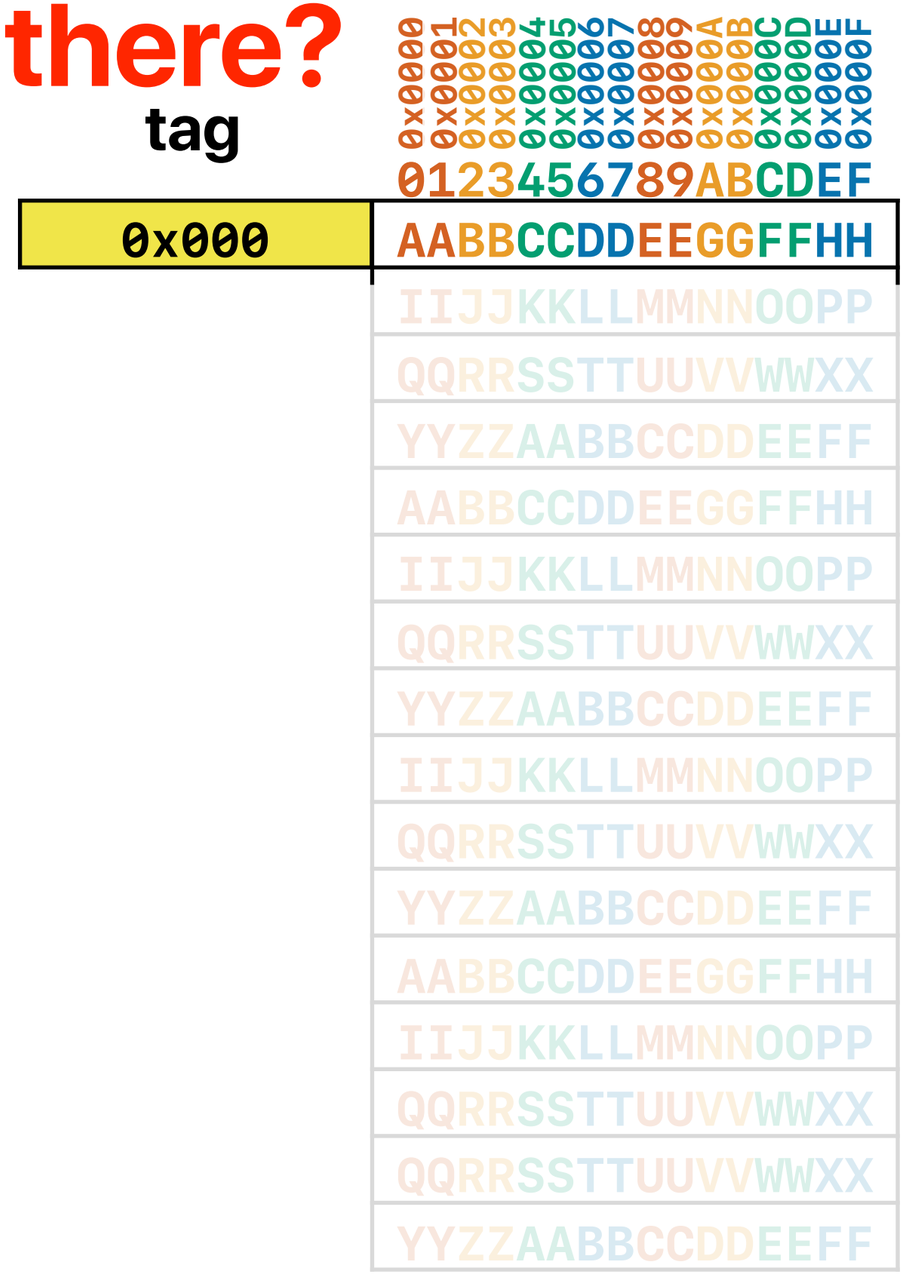
Load/store only access a “word” each time



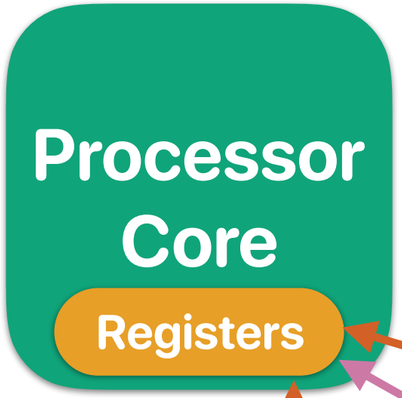
To capture "spatial" locality, $ fetch a "block"



How to tell who is there?



How to tell w



Tell if the block here can be used

Tell if the block here is modified

block offset

tag

load 0x000A

load 0x404A

0x404 not found,
go to lower-level memory

Valid Bit	Dirty Bit	tag	data (0123456789ABCDEF)
1	1	0x000	AABBCCDDEEGGFFHH
1	1	0x001	IIJJKKLLMMNNOOPP
1	0	0xF07	QQRRSSTTUUVVWWXX
0	1	0x100	YYZZAABBCCDDEEFF
1	1	0x310	AABBCCDDEEGGFFHH
1	1	0x450	IIJJKKLLMMNNOOPP
0	1	0x006	QQRRSSTTUUVVWWXX
0	1	0x537	YYZZAABBCCDDEEFF
1	1	0x266	IIJJKKLLMMNNOOPP
1	1	0x307	QQRRSSTTUUVVWWXX
0	1	0x265	YYZZAABBCCDDEEFF
0	1	0x80A	AABBCCDDEEGGFFHH
1	1	0x620	IIJJKKLLMMNNOOPP
1	1	0x630	QQRRSSTTUUVVWWXX
1	0	0x705	QQRRSSTTUUVVWWXX
0	1	0x216	YYZZAABBCCDDEEFF

The complexity of search the matching tag—

$O(n)$— will be slow if our cache size grows!

Can we search things faster?

—hash table! $O(1)$

Hash-like structure — direct-mapped cache

block offset

tag

index

load 0x000A

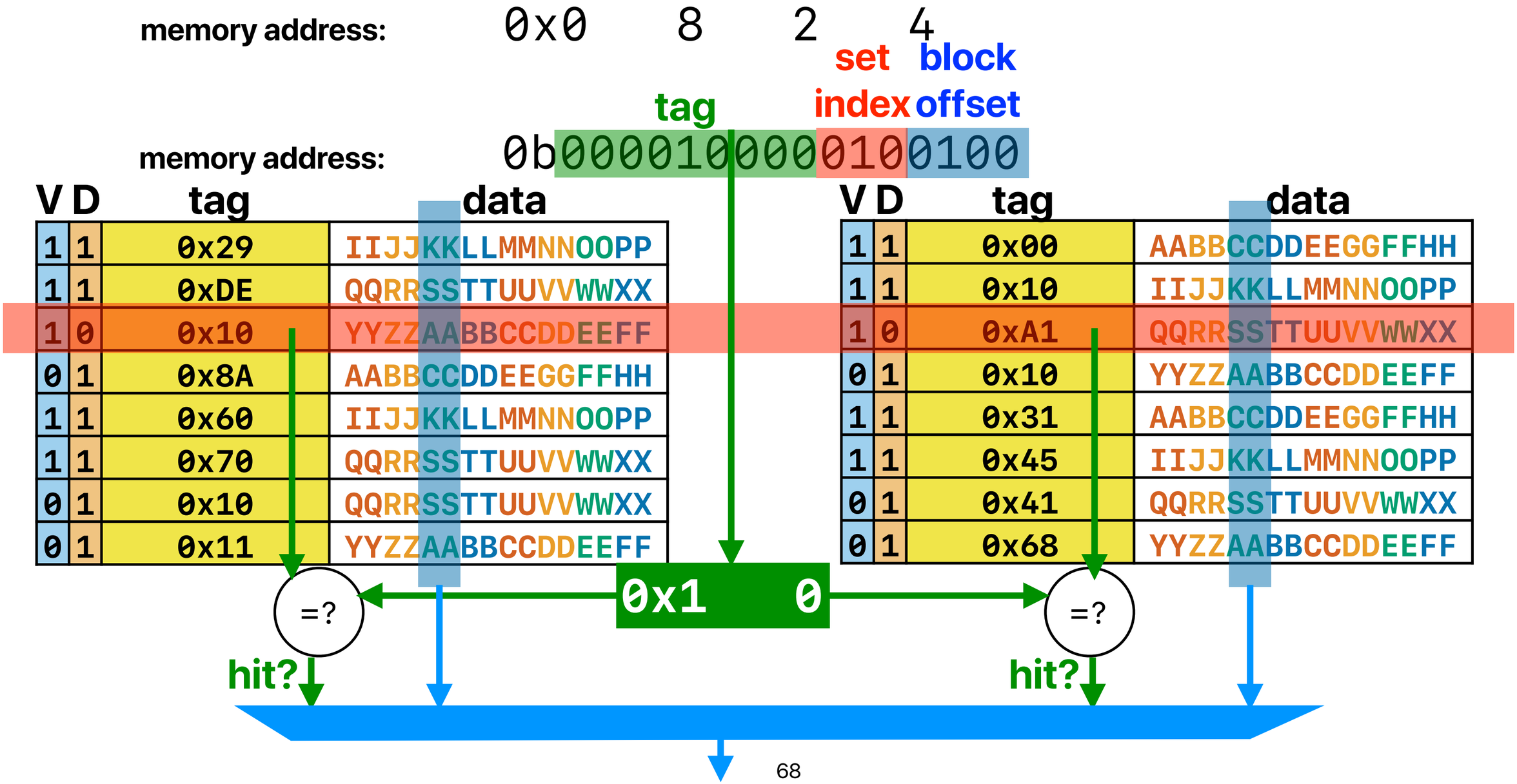
load 0x404A

0x40 not found,
go to lower-level memory

V	D	tag	data
			0123456789ABCDEF
1	1	0x00	AABBCCDDEEGGFFHH
1	1	0x10	IIJJKKLLMMNNOOPP
1	0	0xA1	QQRRSSTTUUVVWWXX
0	1	0x10	YYZZAABBCCDDEEFF
1	1	0x31	AABBCCDDEEGGFFHH
1	1	0x45	IIJJKKLLMMNNOOPP
0	1	0x41	QQRRSSTTUUVVWWXX
0	1	0x68	YYZZAABBCCDDEEFF
1	1	0x29	IIJJKKLLMMNNOOPP
1	1	0xDE	QQRRSSTTUUVVWWXX
0	1	0xCB	YYZZAABBCCDDEEFF
0	1	0x8A	AABBCCDDEEGGFFHH
1	1	0x60	IIJJKKLLMMNNOOPP
1	1	0x70	QQRRSSTTUUVVWWXX
1	0	0x10	QQRRSSTTUUVVWWXX
0	1	0x11	YYZZAABBCCDDEEFF

The biggest issue with hash is — Collision!

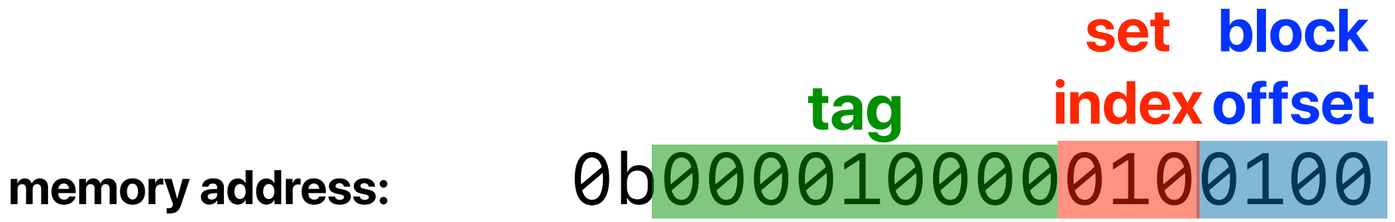
Way-associative cache



C = ABS

- **C**: **C**apacity in data arrays
- **A**: Way-**A**ssociativity — how many blocks within a set
  - N-way: N blocks in a set, A = N
  - 1 for direct-mapped cache
- **B**: **B**lock Size (Cacheline)
  - How many bytes in a block
- **S**: Number of **S**ets:
  - A set contains blocks sharing the same index
  - 1 for fully associate cache

Corollary of C = ABS



- number of bits in **b**lock offset — lg(**B**)
- number of bits in **s**et index: lg(**S**)
- tag bits: address_length - lg(S) - lg(B)
  - address_length is 32 bits for 32-bit machine
- (address / block_size) % S = set index

AMD Phenom II

- L1 data (D-L1) cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block
  - Assume 64-bit memory address

  Which of the following is correct?

A. Tag is 49 bits
B. Index is 8 bits
C. Offset is 7 bits
D. The cache has 1024 sets
E. None of the above

$$C = ABS$$

$$64KB = 2 * 64 * S$$

$$S = 512$$

$$offset = \lg(64) = 6 \text{ bits}$$

$$index = \lg(512) = 9 \text{ bits}$$

$$tag = 64 - \lg(512) - \lg(64) = 49 \text{ bits}$$

intel Core i7

- L1 data (D-L1) cache configuration of Core i7
  - Size 32KB, 8-way set associativity, 64B block
  - Assume 64-bit memory address
  - Which of the following is NOT correct?

A. Tag is 52 bits

B. Index is 6 bits

C. Offset is 6 bits

D. The cache has 128 sets

$$C = ABS$$

$$32KB = 8 * 64 * S$$

$$S = 64$$

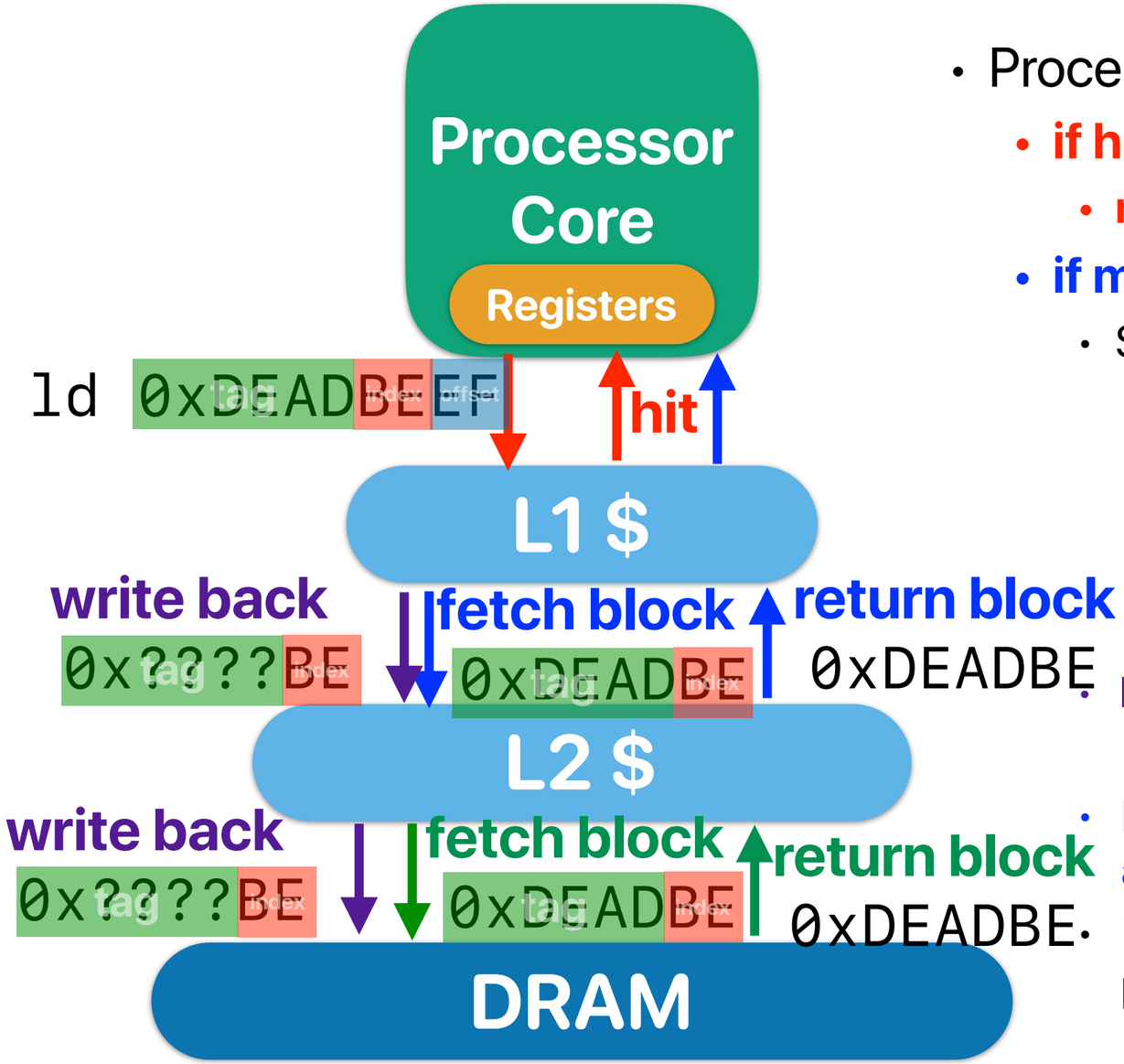
$$offset = lg(64) = 6 \text{ bits}$$

$$index = lg(64) = 6 \text{ bits}$$

$$tag = 64 - lg(64) - lg(64) = 52 \text{ bits}$$

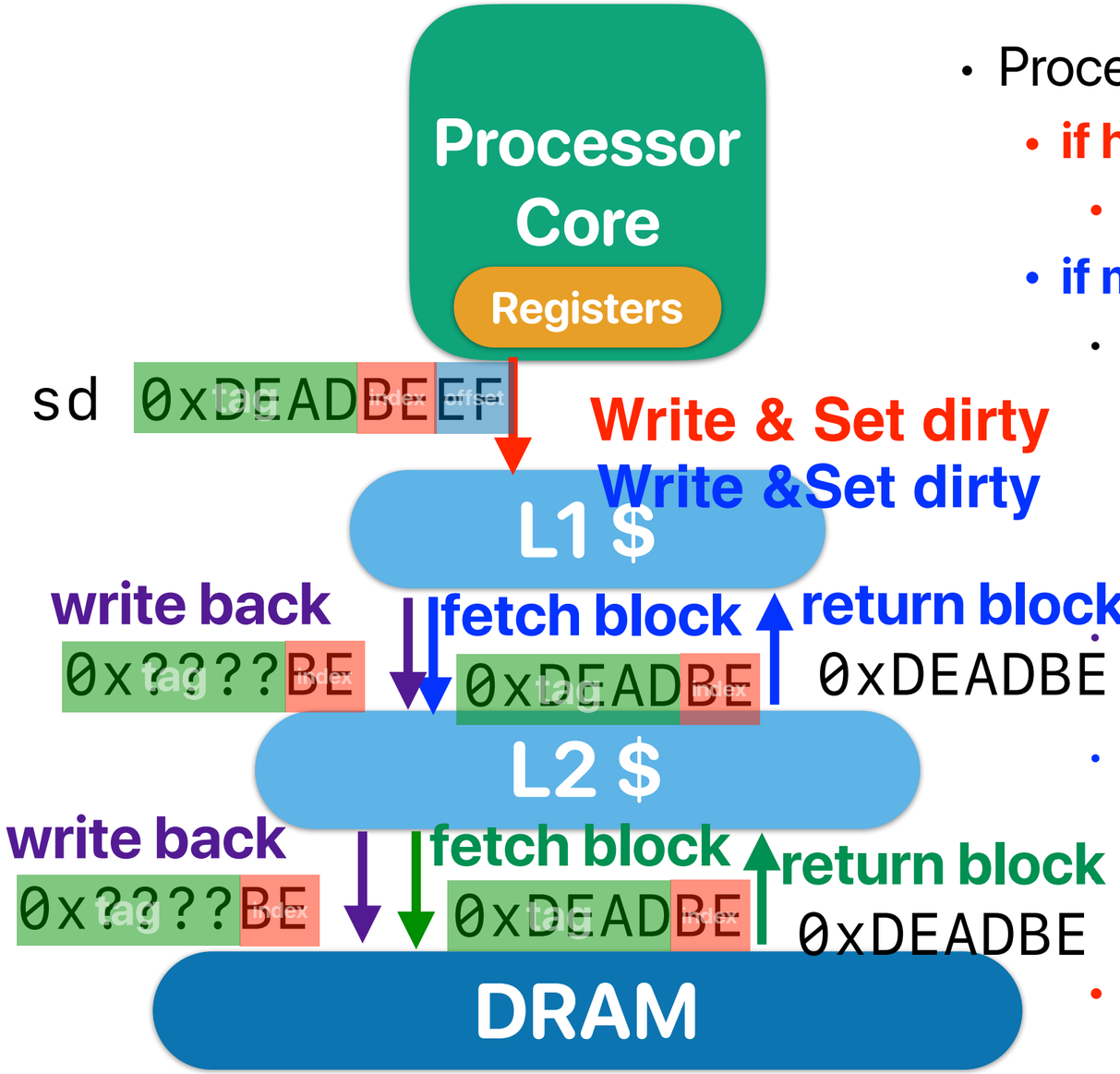
Put everything all together: How cache interacts with CPU

What happens when we read data



- Processor sends load request to L1-$
  - **if hit**
    - **return data**
  - **if miss**
    - Select a victim block
      - If the target "set" is not full — select an empty/invalidated block as the victim block
      - If the target "set is full — select a victim block using some policy
      - LRU is preferred — to exploit temporal locality!
    - If the victim block is "dirty" & "valid"
      - **Write back** the block to lower-level memory hierarchy
    - Fetch the requesting block from lower-level memory hierarchy and place in the victim block
    - If write-back or fetching causes any miss, repeat the same process

What happens when we write data



- Processor sends load request to L1-$
  - **if hit**
    - **return data — set DIRTY**
  - **if miss**
    - Select a victim block
      - If the target "set" is not full — select an empty/invalidated block as the victim block
      - If the target "set is full — select a victim block using some policy
      - LRU is preferred — to exploit temporal locality!
    - If the victim block is "dirty" & "valid"
      - **Write back** the block to lower-level memory hierarchy
    - Fetch the requesting block from lower-level memory hierarchy and place in the victim block
    - If write-back or fetching causes any miss, repeat the same process
    - **Present the write "ONLY" in L1 and set DIRTY**

Performance evaluation considering cache

- If the load/store instruction hits in L1 cache where the hit time is usually the same as a CPU cycle
  - The CPI of this instruction is the base CPI
- If the load/store instruction misses in L1, we need to access L2
  - The CPI of this instruction needs to include the cycles of accessing L2
- If the load/store instruction misses in both L1 and L2, we need to go to lower memory hierarchy (L3 or DRAM)
  - The CPI of this instruction needs to include the cycles of accessing L2, L3, DRAM

How to evaluate cache performance

- $CPI_{Average}$: the average CPI of a memory instruction

$$CPI_{Average} = CPI_{base} + miss\_rate_{L1} * miss\_penalty_{L1}$$

$$miss\_penalty_{L1} = CPI_{accessing\_L2} + miss\_rate_{L2} * miss\_penalty_{L2}$$

$$miss\_penalty_{L2} = CPI_{accessing\_L3} + miss\_rate_{L3} * miss\_penalty_{L3}$$

$$miss\_penalty_{L3} = CPI_{accessing\_DRAM} + miss\_rate_{DRAM} * miss\_penalty_{DRAM}$$

- If the problem is asking for **average memory access time**, transform the CPI values into/from time by multiplying with CPU cycle time!

Cache & Performance

- Application: 80% ALU, 20% Loads
- Assume the 1-cycle L1 hit time allows the CPI to be 1
- L1 I-cache miss rate: 5%, hit time: 1 cycle
- L1 D-cache miss rate: 10%, hit time: 1 cycle
- L2 U-Cache miss rate: 20%, hit time: 10 cycles
- Main memory hit time: 100 cycles
- What's the average CPI?

$$CPI_{Average} = CPI_{base} + miss\_rate * miss\_penalty$$

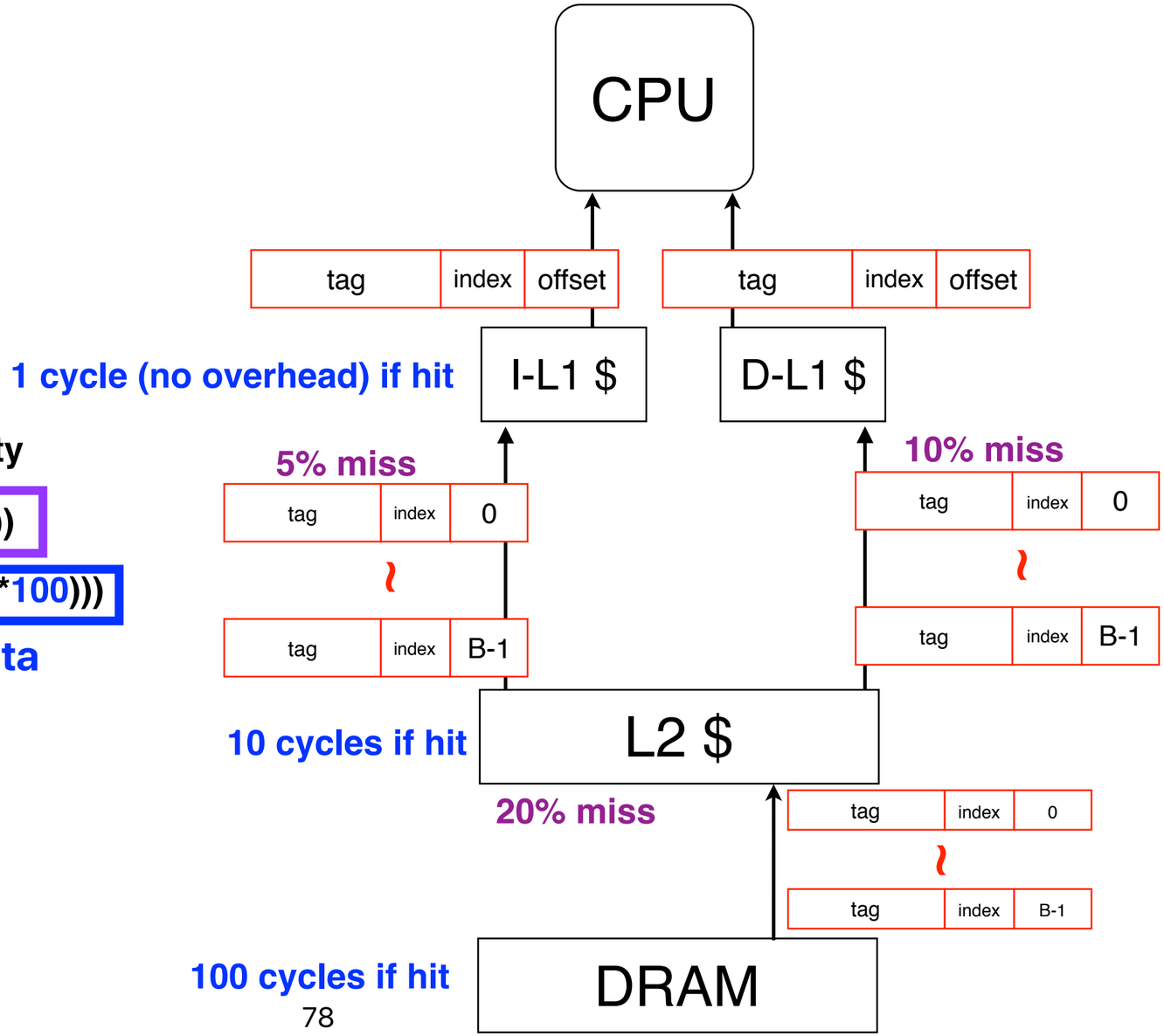
$$= 1 + 100\% * (5\% * (10 + 20\% * (1 * 100)))$$

Fetch Instructions

$$+ 20\% * (10\% * (1) * (10 + 20\% * ((1) * 100)))$$

Access Data

$$= 3.1$$



Cache & Performance

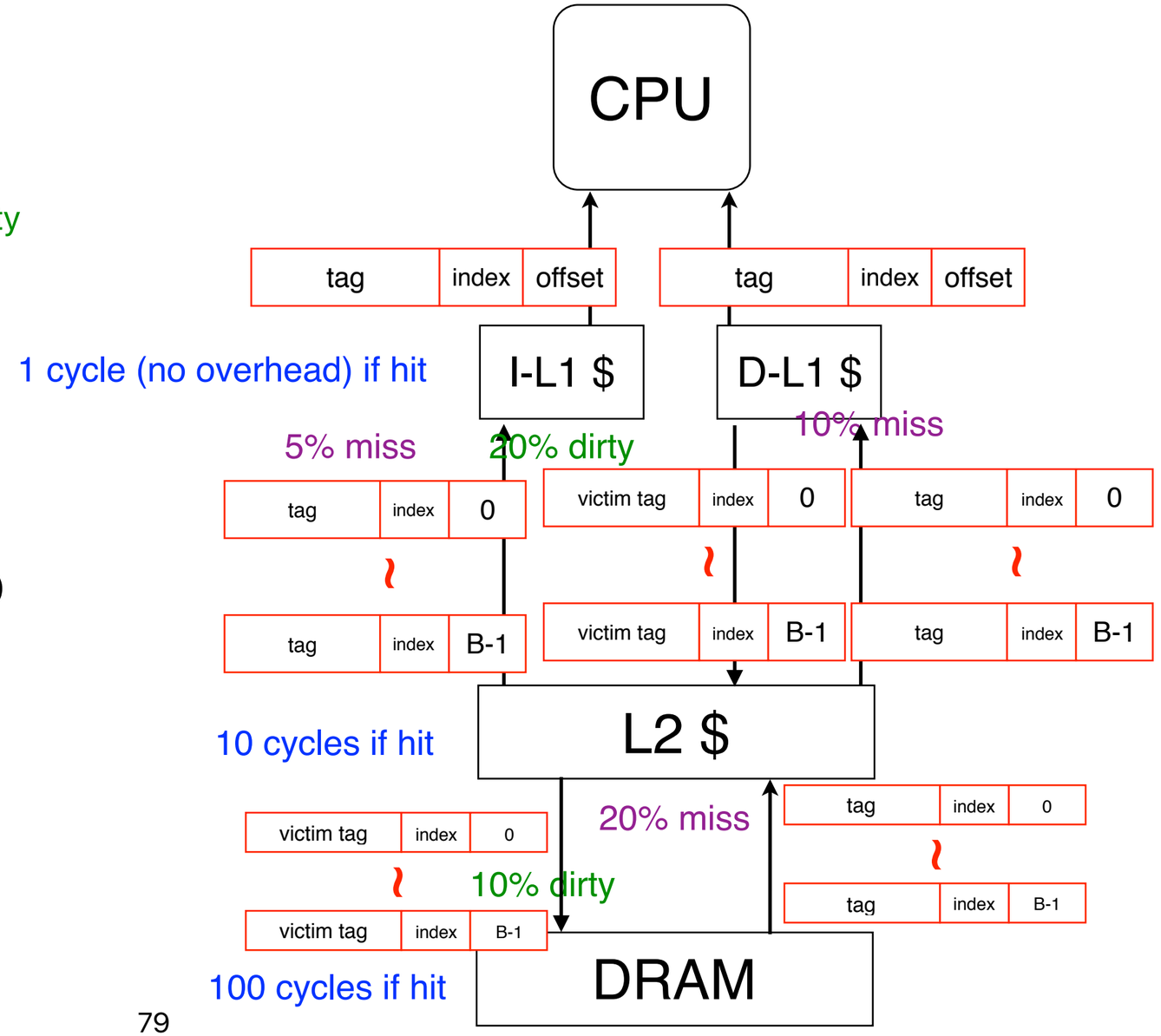
- Application: 80% ALU, 20% Load/Store
- L1 I-cache miss rate: 5%, hit time: 1 cycle
- L1 D-cache miss rate: 10%, hit time: 1 cycle, 20% dirty
- L2 U-Cache miss rate: 20%, hit time: 10 cycles, 10% dirty
- Main memory hit time: 100 cycles
- What's the average CPI?

$CPI_{Average}$= CPI_{base} + miss_rate*miss_penalty

= 1+100%*(5%*(10+20%*((1+10%)*100)))

+20%*(10%* (1+20%)*(10+20%*((1+10%)*100)))

= 3.368



Cause of cache misses

3Cs of misses

- Compulsory miss
  - Cold start miss. First-time access to a block
- Capacity miss
  - The working set size of an application is bigger than cache size
- Conflict miss
  - Required data replaced by block(s) mapping to the same set
  - Similar collision in hash

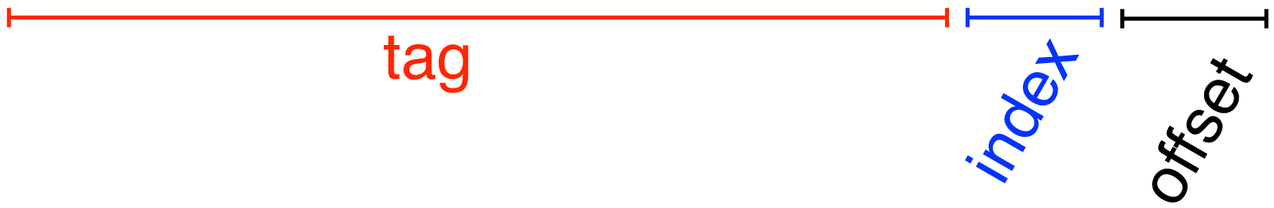
Simulate the cache!

Tips for cache simulation

- Figure out the memory access patterns
  - Address sequences from your code
  - The behavior/locality of the variables/arrays
- Partition the address
  - Use C=ABS
  - Find out tag, index
- Check your current cache content
  - Hit: for the same index, if you can find the same tag there.
  - Otherwise, miss
    - Compulsory misses: you never accessed the same (tag,index) pair before
    - Conflict misses: the tag appeared in the same index before
    - Replace the least recently used block with the requesting block

Simulate a direct-mapped cache

- Consider a direct mapped (1-way) cache with 256 bytes total capacity, a block size of 16 bytes, and the application repeatedly reading the following memory addresses:
  - 0b1000000000, 0b1000001000, 0b1000010000, 0b1000010100, 0b1100010000
    - C = A B S
    - S=256/(16*1) = 16
    - lg(16) = 4 : 4 bits are used for the index
    - lg(16) = 4 : 4 bits are used for the byte offset
    - The tag is 48 - (4 + 4) = 40 bits
    - For example: `0b1000 0000 0000 0000 0000 0000 1000 0000`



Simulate a direct-mapped cache

	V	D	Tag	Data
0	1	0	0b10	
1	1	0	0b10	
2	0	0		
3	0	0		
4	0	0		
5	0	0		
6	0	0		
7	0	0		
8	0	0		
9	0	0		
10	0	0		
11	0	0		
12	0	0		
13	0	0		
14	0	0		
15	0	0		

tag	index		
0b10	0000	0000	compulsory miss
0b10	0000	1000	hit!
0b10	0001	0000	compulsory miss
0b10	0001	0100	hit!
0b11	0001	0000	compulsory miss
0b10	0000	0000	hit!
0b10	0000	1000	hit!
0b10	0001	0000	conflict miss
0b10	0001	0100	hit!

Simulate a 2-way cache

- Consider a 2-way cache with 256 bytes total capacity, a block size of 16 bytes, and the application repeatedly reading the following memory addresses:
  - 0b1000000000, 0b1000001000, 0b1000010000, 0b1000010100, 0b1100010000
    - C = A B S
    - S=256/(16*2) = 8
    - 8 = 2^3 : 3 bits are used for the index
    - 16 = 2^4 : 4 bits are used for the byte offset
    - The tag is 32 - (3 + 4) = 25 bits
    - For example: `0b1000 0000 0000 0000 0000 0000 0001 0000`

Simulate a 2-way cache

	V	D	Tag	Data
0	1	0	0b10	
1	1	0	0b10	
2	0	0		
3	0	0		
4	0	0		
5	0	0		
6	0	0		
7	0	0		

V	D	Tag	Data
0	0		
1	0	0b11	
0	0		
0	0		
0	0		
0	0		
0	0		
0	0		

	tag	index		
0b10	0000	0000	compulsory miss
0b10	0000	1000	hit!
0b10	0001	0000	compulsory miss
0b10	0001	0100	hit!
0b11	0001	0000	compulsory miss
0b10	0000	0000	hit!
0b10	0000	1000	hit!
0b10	0001	0000	hit
0b10	0001	0100	hit!

AMD Phenom II

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 32-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

What's the data cache miss rate for this code?

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

C = ABS

64KB = 2 * 64 * S

S = 512

offset = lg(64) = 6 bits

index = lg(512) = 9 bits

tag = 64 - lg(512) - lg(64) = 49 bits

AMD Phenom II

100% miss rate!

- Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 48-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    c[i] = a[i] + b[i]; /*load a[i], load b[i], store c[i]*/
```

C = ABS
64KB = 2 * 64 * S
S = 512
offset = lg(64) = 6 bits
index = lg(512) = 9 bits
tag = the rest bits

	address in hex	address in binary (tag / index / offset)	tag	index	hit? miss?
load a[0]	0x20000	0b10 0000 0000 0000 0000	0x4	0	compulsory miss
load b[0]	0x30000	0b11 0000 0000 0000 0000	0x6	0	compulsory miss
store c[0]	0x10000	0b01 0000 0000 0000 0000	0x2	0	compulsory miss, evict
load a[1]	0x20004	0b10 0000 0000 0000 0100	0x4	0	conflict miss, evict 0x6
load b[1]	0x30004	0b11 0000 0000 0000 0100	0x6	0	conflict miss, evict 0x2
store c[1]	0x10004	0b01 0000 0000 0000 0100	0x2	0	conflict miss, evict 0x4
⋮	⋮	⋮	⋮	⋮	⋮
load a[15]	0x2003C	0b10 0000 0000 0011 1100	0x4	0	miss, evict 0x6
load b[15]	0x3003C	0b11 0000 0000 0011 1100	0x6	0	miss, evict 0x2
store c[15]	0x1003C	0b01 0000 0000 0011 1100	0x2	0	miss, evict 0x4
load a[16]	0x20040	0b10 0000 0000 0100 0000	0x4	1	compulsory miss
load b[16]	0x30040	0b11 0000 0000 0100 0000	0x6	1	compulsory miss
store c[16]	0x10040	0b01 0000 0000 0100 0000	0x2	1	compulsory miss, evict

AMD Phenom II

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 32-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

What's the data cache miss rate for this code?

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

C = ABS

64KB = 2 * 64 * S

S = 512

offset = lg(64) = 6 bits

index = lg(512) = 9 bits

tag = 64 - lg(512) - lg(64) = 49 bits

Matrix transpose

```
double A[16384], B[16384];
int N=128;
for(i = 0; i < N; i++)
    for(j = 0; j < N; j++)
        B[i*N+j] = A[j*N+i];
// assume load A[j*N+i] and then store B[i*N+j]
// &A[0] is 0x20000, &B[0] is 0x40000
```

What's the access sequence of A[] looks like?

A[0], A[128], A[256], ..., A[127*128], A[1], A[129]..., A[127*128+1], ...

What's the access sequence of B[] looks like?

B[0], B[1], B[2], ...

Improving 3Cs

Improvement of 3Cs

- 3Cs and A, B, C of caches
  - Compulsory miss
    - Increase B: increase miss penalty (more data must be fetched from lower hierarchy)
  - Capacity miss
    - Increase C: increase cost, access time, power
  - Conflict miss
    - Increase A: increase access time and power
- Or modify the memory access pattern of your program!

Programming and memory performance

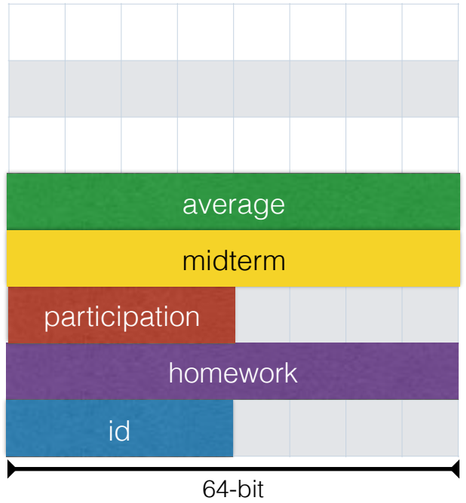
Memory addressing/alignment

- Almost every popular ISA architecture uses "byte-addressing" to access memory locations
- Instructions generally work faster when the given memory address is aligned
  - Aligned — if an instruction accesses an object of size n at address X, the access is **aligned** if $X \bmod n = 0$.
  - Some architecture/processor does not support aligned access at all
  - Therefore, compilers only allocate objects on "aligned" address

The result of `sizeof(struct student)`

- Consider the following data structure:

```
struct student {
    int id;
    double *homework;
    int participation;
    double midterm;
    double average;
};
```



What's the output of `printf("%lu\n",sizeof(struct student))`?

A. 20
B. 28
C. 32
D. 36
E. 40

Loop interchange/fission/fusion

Demo — programmer & performance

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

$O(n^2)$	Complexity	$O(n^2)$
Same	Instruction Count?	Same
Same	Clock Rate	Same
Better	CPI	Worse

AMD Phenom II

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 32-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

What's the data cache miss rate for this code?

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

C = ABS

64KB = 2 * 64 * S

S = 512

offset = lg(64) = 6 bits

index = lg(512) = 9 bits

tag = 64 - lg(512) - lg(64) = 49 bits

What if the code look like this?

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 32-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    c[i] = a[i]; //load a and then store to c
for(i = 0; i < 512; i++)
    c[i] += b[i]; //load b, load c, add, and then store to c
```

What's the data cache miss rate for this code?

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

Loop Fusion

```
/* Before */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];


/* After */
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
    {
        a[i][j] = 1/b[i][j] * c[i][j];
         d[i][j] = a[i][j] + c[i][j];
    }
```

2 misses per access to a & c vs. one miss per access

Blocking

Case study: Matrix Multiplication

```
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Algorithm class tells you it's $O(n^3)$

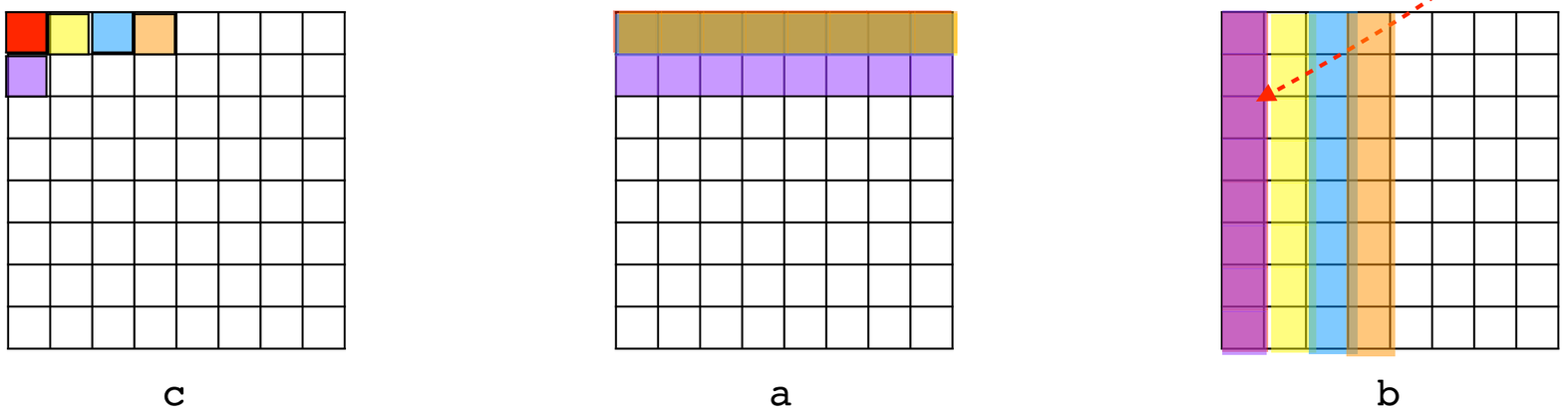
If n=512, it takes about 1 sec

How long is it take when n=1024?

Matrix Multiplication

```
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Very likely a miss if array is large



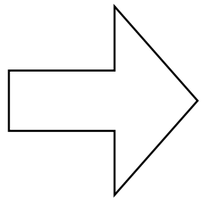
- If each dimension of your matrix is 1024
  - Each row takes 1024*8 bytes = 8KB
  - The L1 $ of intel Core i7 is 32KB, 8-way, 64-byte blocked
  - You can only hold at most 4 rows/columns of each matrix!
  - You need the same row when j increase!

Block algorithm for matrix multiplication

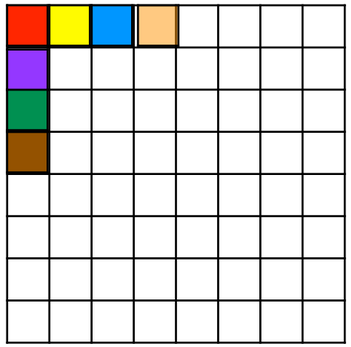
- Discover the cache miss rate
  - valgrind --tool=cachegrind cmd
    - cachegrind is a tool profiling the cache performance
  - Performance counter
    - Intel® Performance Counter Monitor http://www.intel.com/software/pcm/

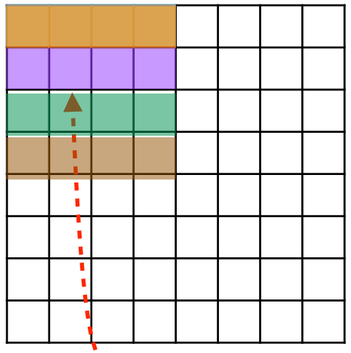
Block algorithm for matrix multiplication

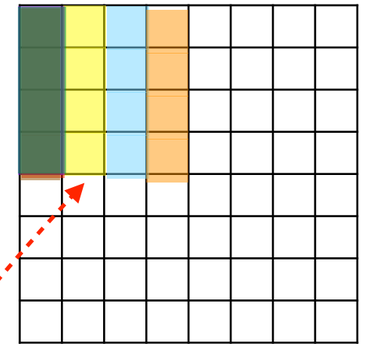
```
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```



```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
              c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```



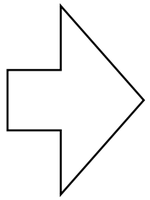




You only need to hold these sub-matrices in your cache

Matrix Transpose

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
              c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```



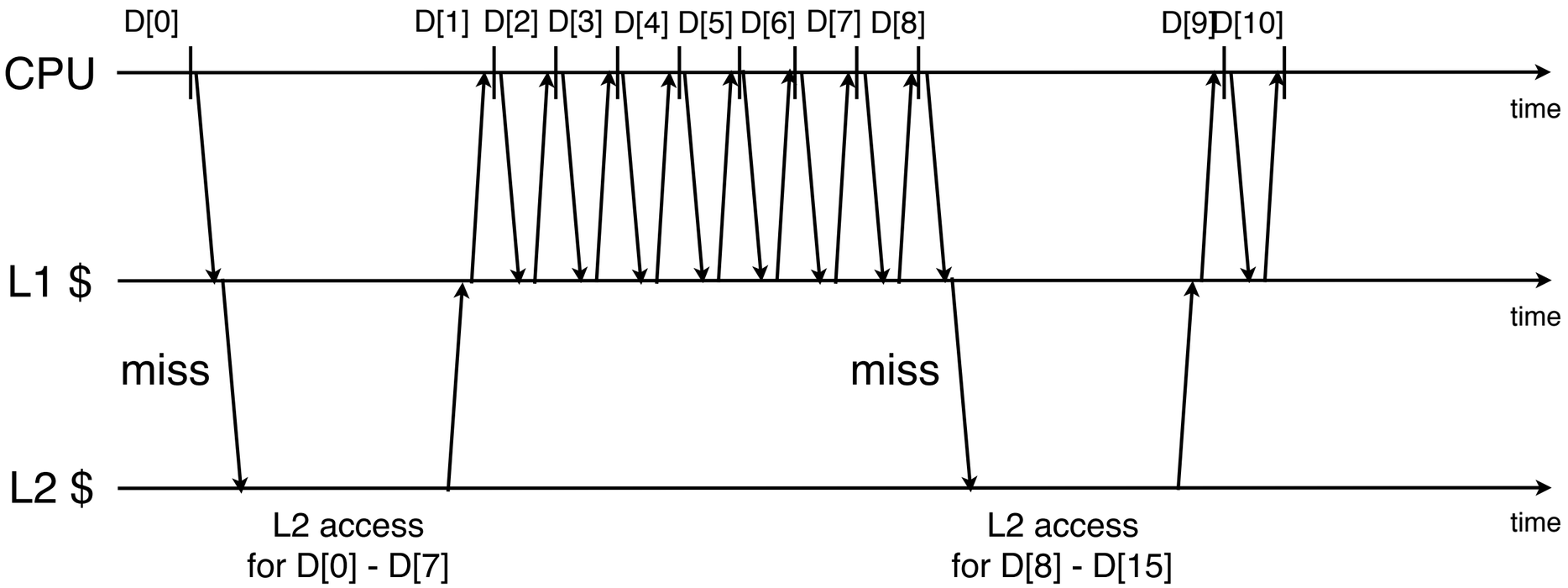
```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}
```

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

Prefetching

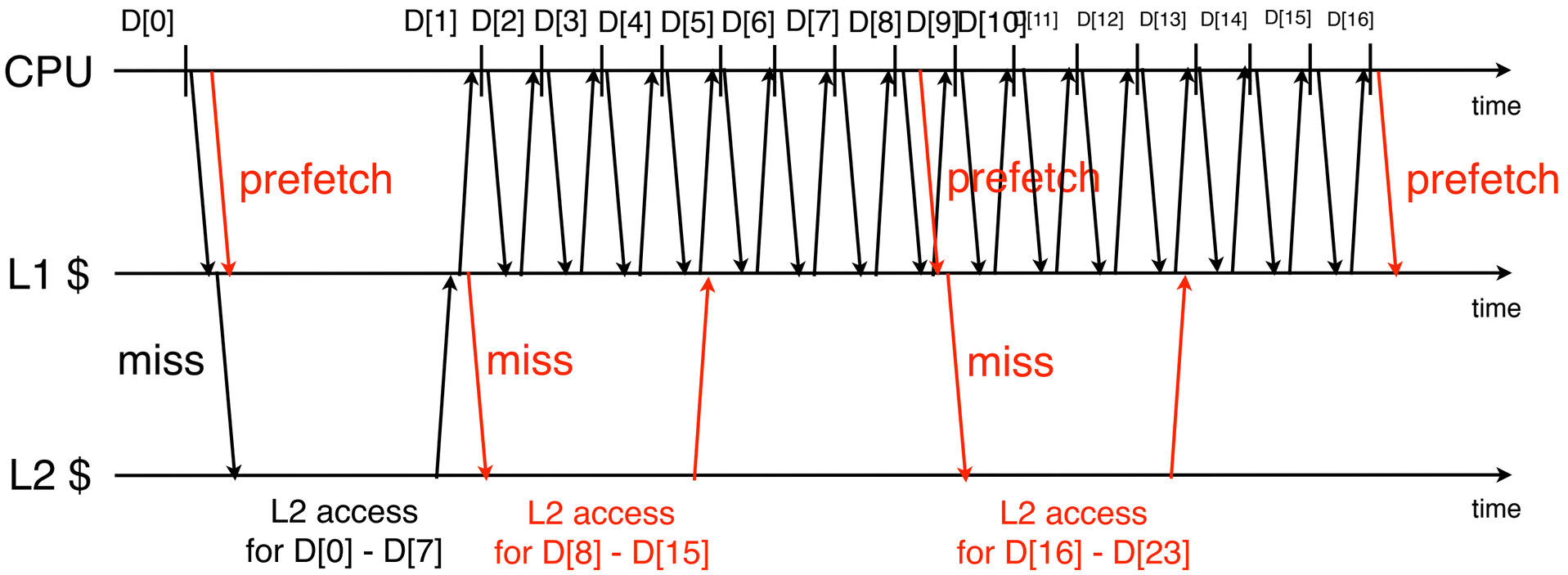
Characteristic of memory accesses

```
for(i = 0;i < 1000000; i++) {
     D[i] = rand();
}
```



Prefetching

```
for(i = 0;i < 1000000; i++) {
      D[i] = rand();
      // prefetch D[i+8] if i % 8 == 0
}
```



Prefetching

- Identify the access pattern and proactively fetch data/instruction before the application asks for the data/instruction
  - Trigger the cache miss earlier to eliminate the miss when the application needs the data/instruction
- Hardware prefetch
  - The processor can keep track the distance between misses. If there is a pattern, fetch miss_data_address+distance for a miss
- Software prefetch
  - Load data into X0
  - Using prefetch instructions

Demo

- x86 provide prefetch instructions
- As a programmer, you may insert `_mm_prefetch` in x86 programs to perform software prefetch for your code
- gcc also has a flag "-fprefetch-loop-arrays" to automatically insert software prefetch instructions

Where can prefetch work effectively?

- How many of the following code snippet can “prefetching” effectively help improving performance?

(1)
```
while(node){
    node = node->next;
}
```
— where the next pointing to is hard to predict

(2)
```
while(++i<100000)
    a[i]=rand();
```

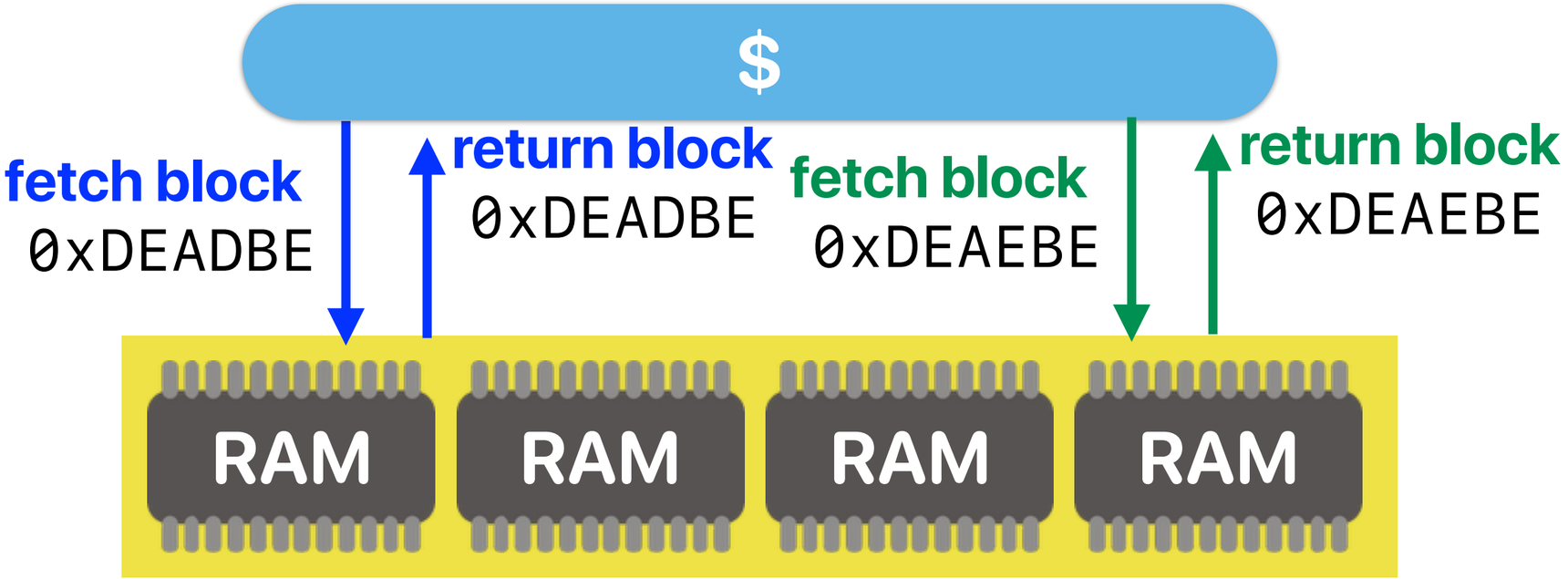
(3)
```
while (root != NULL){
        if (key > root->data)
            root = root->right;

        else if (key < root->data)
            root = root->left;
        else
            return true;
}
```
— where the next node is also hard to predict

(4)
```
    for (i = 0; i < 65536; i++) {
      mix_i = ((i * 167) + 13) & 65536;
      results[mix_i]++;
    }
```
— the stride to the next element is hard to predict…

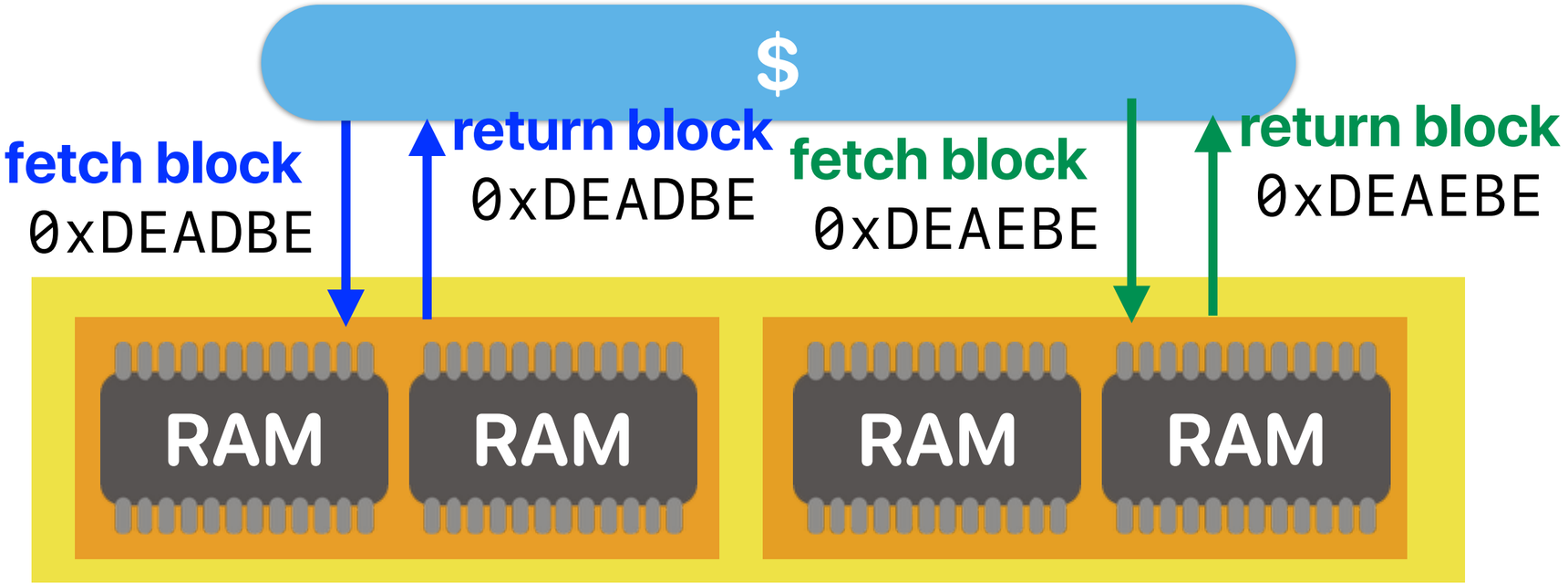
A. 0
B. 1
C. 2
D. 3
E. 4

Advanced Hardware Techniques in Improving Memory Performance

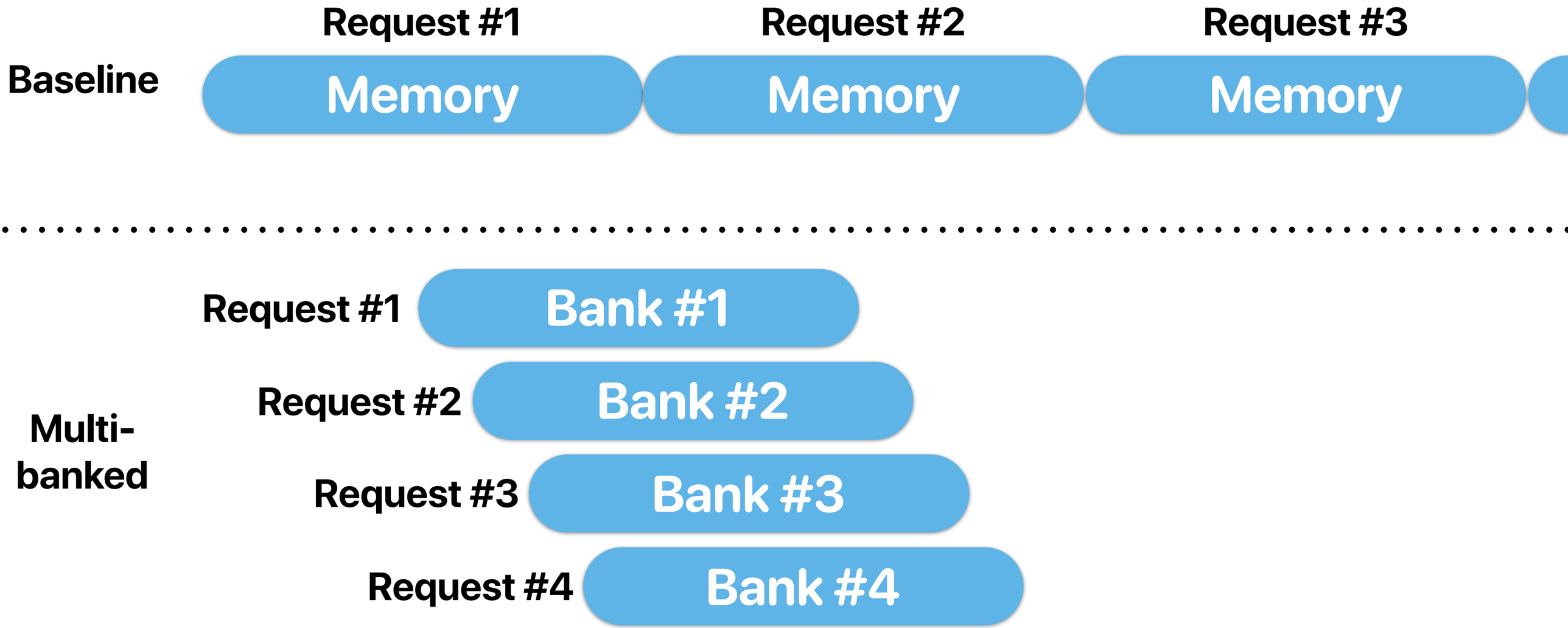
Without banks



Multibanks & non-blocking caches



Pipelined access and multi-banked caches



Pipelined access and multi-banked caches

- Assume each bank in the $ takes 10 ns to serve a request, and the $ can take the next request 1 ns after assigning a request to a bank — if we have 4 banks and we want to serve 4 requests, what's the speedup over non-banked, non-pipelined $? — pick the closest one

  A. 1x — no speedup
  B. 2x
  C. 3x
  D. 4x
  E. 5x

$$ET_{baseline} = 4 \times 10\ ns = 40\ ns$$

$$ET_{banked} = 10\ ns + 3 \times 1\ ns = 13\ ns$$

$$Speedup = \frac{Execution\ Time_{baseline}}{Execution\ Time_{banked}}$$

$$= \frac{40}{13} = 3.08\times$$

Early Restart and Critical Word First

- Don’t wait for full block to be loaded before restarting CPU
  - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called wrapped fetch and requested word first
- Most useful with large blocks
- Spatial locality a problem; often we want the next sequential word soon, so not always a benefit (early restart).

Midterm Logistics

For midterm

- No cheat sheet allowed
- No cheating allowed
- We will have some problems require you to write
- You may bring a calculator
- You should bring pen/pencil/eraser
- My last office hour before midterm — this Wednesday @ 1pm-2pm — friday is cancelled.

Format of the midterm

- Multiple choices * 10 — like your clicker/reading quizzes multiple choices questions
- Short answer question * 5
  - Each answer MUST be less than 30 words
  - Writing more than 30 words is equivalent to writing 0 words
- Homework style free-answer questions * 3
  - You need to clearly write down the original form of the applied equation/formula
  - You need to replace each term accordingly with numbers
  - You will have some credits for right equations even though the final number isn’t correct
  - You will receive 0 credits if we only see the numbers

Sample Midterm

Identify the performance bottleneck

- Why does an Intel Core i7 @ 3.5 GHz usually perform better than an Intel Core i5 @ 3.5 GHz or AMD FX-8350@4GHz?

Sysbench 2014 from http://www.anandtech.com/

A. Because the instruction count of the program are different
B. Because the clock rate of AMD FX is higher
C. Because the CPI of Core i7 is better
D. Because the clock rate of AMD FX is higher and CPI of Core i7 is better
E. None of the above

Amdahl's Law on Multicore Architectures

- Regarding Amdahl's Law on multicore architectures, how many of the following statements is/are correct?
  - ① If we have unlimited parallelism, the performance of each parallel piece does not matter as long as the performance slowdown in each piece is bounded
  - ② With unlimited amount of parallel hardware units, single-core performance does not matter anymore
  - ③ With unlimited amount of parallel hardware units, the maximum speedup will be bounded by the fraction of parallel parts
  - ④ With unlimited amount of parallel hardware units, the effect of scheduling and data exchange overhead is minor

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

How programmer affects performance?

- Performance equation consists of the following three factors
  ① IC
  ② CPI
  ③ CT

  How many can a **programmer** affect?

  A. 0
  B. 1
  C. 2
  D. 3

Demo — programmer & performance

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

- How many of the following make(s) the performance of A better than B?
  - ① IC
  - ② CPI
  - ③ CT
  - A. 0
  - B. 1
  - C. 2
  - D. 3

Fair comparison

- How many of the following comparisons are fair?
  - ① Comparing the frame rates of Halo 5 on AMD RyZen 1600X and civilization on Intel Core i7 7700X
  - ② Using bit torrent to compare the network throughput on two machines
  - ③ Comparing the frame rates of Halo 5 using medium settings on AMD RyZen 1600X and low settings on Intel Core i7 7700X
  - ④ Using the peak floating point performance to judge the gaming performance of machines using AMD RyZen 1600X and Intel Core i7 7700X

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Locality

- Which description about locality of arrays `sum` and `A` in the following code is the most accurate?

```
for(i = 0; i< 100000; i++)
{
    sum[i%10] += A[i];
}
```

A. Access of `A` has temporal locality, `sum` has spatial locality
B. Both `A` and `sum` have temporal locality, and sum also has spatial locality
C. Access of `A` has spatial locality, `sum` has temporal locality
D. Both `A` and `sum` have spatial locality
E. Both `A` and `sum` have spatial locality, and sum also has temporal locality

3Cs and A, B, C

- Regarding 3Cs: compulsory, conflict and capacity misses and A, B, C: associativity, block size, capacity
  How many of the following are correct?
  - ① Increasing associativity can reduce conflict misses
  - ② Increasing associativity can reduce hit time
  - ③ Increasing block size can increase the miss penalty
  - ④ Increasing block size can reduce compulsory misses

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

intel Core i7

- L1 data (D-L1) cache configuration of Core i7
  - Size 32KB, 8-way set associativity, 64B block
  - Assume 64-bit memory address
  - Which of the following is NOT correct?
    A. Tag is 52 bits
    B. Index is 6 bits
    C. Offset is 6 bits
    D. The cache has 128 sets

Virtual indexed, physical tagged cache limits the cache size

- If you want to build a virtual indexed, physical tagged cache with 32KB capacity, which of the following configuration is possible? Assume the system use 4K pages.
  A. 32B blocks, 2-way
  B. 32B blocks, 4-way
  C. 64B blocks, 4-way
  D. 64B blocks, 8-way

When we have virtual memory…

- In a modern x86-64 processor supports virtual memory through, how many memory accesses can an instruction incur?
  A. 2
  B. 4
  C. 6
  D. 8
  E. More than 10

Sample short answer questions (< 30 words)

- What is RISC? What is CISC? List two pros/cons for each
- What are the limitations of compiler optimizations? Can you list two?
- Please define Amdahl's Law and explain each term in it
- Please define the CPU performance equation and explain each term.
- Can you list two things affecting each term in the performance equation?
- What's the difference between latency and throughput? When should you use latency or throughput to judge performance?
- What's "benchmark" suite? Why is it important?
- Why TFLOPS or inferences per second is not a good metrics?

Performance Equation/Speedup

- 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 the processor runs at 1GHz, how long is the execution time? If hardware technology improves the processor clock rate to 2GHz, but making load/store CPI to be 12 cycles, how much is the speedup?

Amdahl's Law for multiple optimizations

- Assume that memory access takes 30% of execution time.
  - Cache can speedup 80% of memory operation by a factor of 4
  - L2 cache can speedup 50% of the remaining 20% by a factor of 2
- What's the total speedup?

Performance evaluation with cache

- Consider the following cache configuration on RISC-V processor:

	I-L1	D-L1	L2	DRAM
size	32K	32K	256K	Big enough
block size	64 Bytes	64 Bytes	64 Bytes	4KB pages
associativity	2-way	2-way	8-way	
access time	1 cycle (no penalty if it's a hit)	1 cycle (no penalty if it's a hit)	10 cycles	100 cycles
local miss rate	2%	10%, 20% dirty	15% (i.e., 15% of L1 misses, also miss in the L2), 30% dirty	
Write policy	N/A	Write-back, write allocate		
Replacement	LRU replacement policy			

The application has 20% branches, 10% loads/stores, 70% integer instructions. Assume that TLB miss rate is 2% and it requires 100 cycles to handle a TLB miss. Also assume that the branch predictor has a hit rate of 87.5%, what's the CPI of branch, L/S, and integer instructions? What is the average CPI?

Cache simulation

- The processor has a 8KB, 256B blocked, 2-way L1 cache. Consider the following code:

```
for(i=0;i<256;i++) {
    a[i] = b[i] + c[i];
// load a[i] and load b[i], store to c[i]
// &a[0] = 0x10000, &b[0] = 0x20000, &c[0] = 0x30000
}
```

- What's the total miss rate? How many of the misses are compulsory misses? How many of the misses are conflict misses?
- How can you improve the cache performance of the above code through changing hardware?
- How can you improve the performance **without** changing hardware?