Memory & Caches I

CSE 351 Summer 2021

Instructor:

Mara Kirdani-Ryan

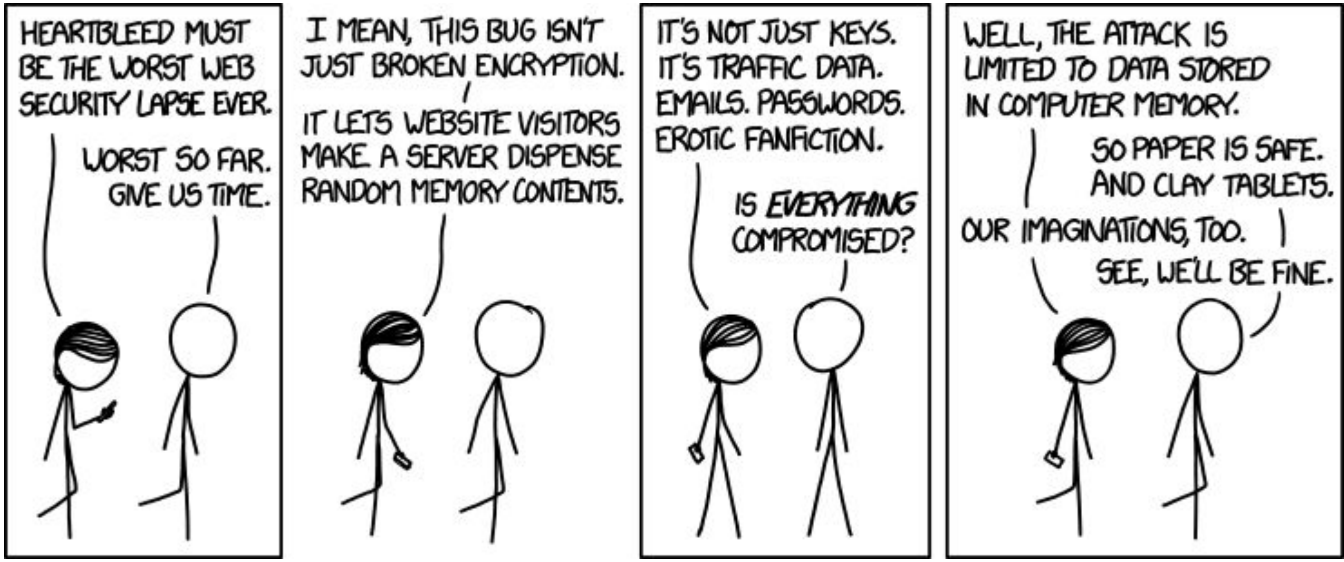
Teaching Assistants:

Kashish Aggarwal

Nick Durand

Colton Jobs

Tim Mandzyuk



Gentle, Loving Reminders

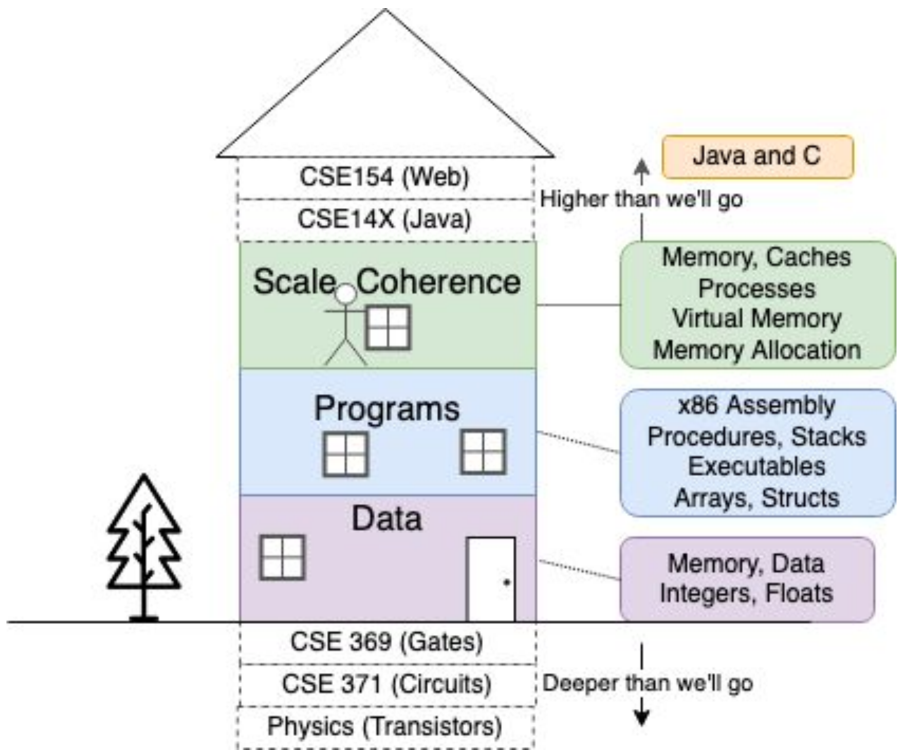
- hw13 due tonight!
- hw14 due Wednesday
- Lab 3 due Friday (7/30)

- Unit Summary 2 Due next Monday (8/2)
  - In-class critique on Wednesday (7/28). Come with something to get feedback on!

- Office hours/email if you want to meet!

Unit 3: Scale, Coherence

- Caches, Process, Virtual Memory
  - Multiple programs? Larger programs?
- Metrics & Structures
- Scale, Automation



Learning Objectives

Understanding this lecture means you can:

- Explain why we have caches
- Define cache misses & hits, and the effect of miss rate on performance
- Define two types of locality, and explain their role in caching
- Determine optimizations by Average Memory Access Time
- Explain how choice of metrics affects the resulting structures

Aside: Units and Prefixes

- Here focusing on large numbers (exponents > 0)
- Note that $10^3 \approx 2^{10}$
- SI prefixes are *ambiguous* if base 10 or 2
- IEC prefixes are *unambiguously* base 2

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

SI Size	Prefix	Symbol	IEC Size	Prefix	Symbol
10^3	Kilo-	K	2^{10}	Kibi-	Ki
10^6	Mega-	M	2^{20}	Mebi-	Mi
10^9	Giga-	G	2^{30}	Gibi-	Gi
10^{12}	Tera-	T	2^{40}	Tebi-	Ti
10^{15}	Peta-	P	2^{50}	Pebi-	Pi
10^{18}	Exa-	E	2^{60}	Exbi-	Ei
10^{21}	Zetta-	Z	2^{70}	Zebi-	Zi
10^{24}	Yotta-	Y	2^{80}	Yobi-	Yi

How to Remember?

- Mnemonics
  - There unfortunately isn't one well-accepted mnemonic
    - But that shouldn't stop you from trying to come with one!
  - **K**iller **M**echanical **G**iraffe **T**eaches **P**et, **E**xtinct **Z**ebra to **Y**odel
  - **K**irby **M**issed **G**anondorf **T**erribly, **P**otentially **E**xterminating **Z**elda and **Y**oshi
  - xkcd: **K**arl **M**arx **G**ave **T**he **P**roletariat **E**leven **Z**eppelins, **Y**o
    - https://xkcd.com/992/
  - You probably only need to remember **K,M,G,T,P**; just have a way to refer to the rest

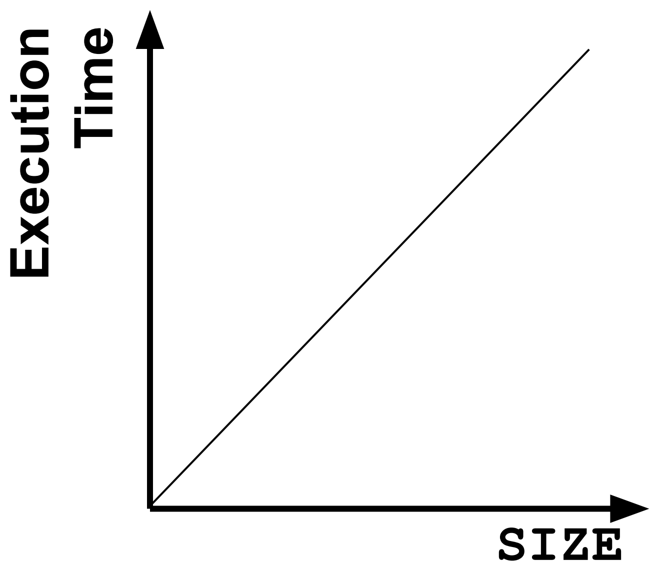
We all care about performance, right?

How does execution time grow with SIZE?

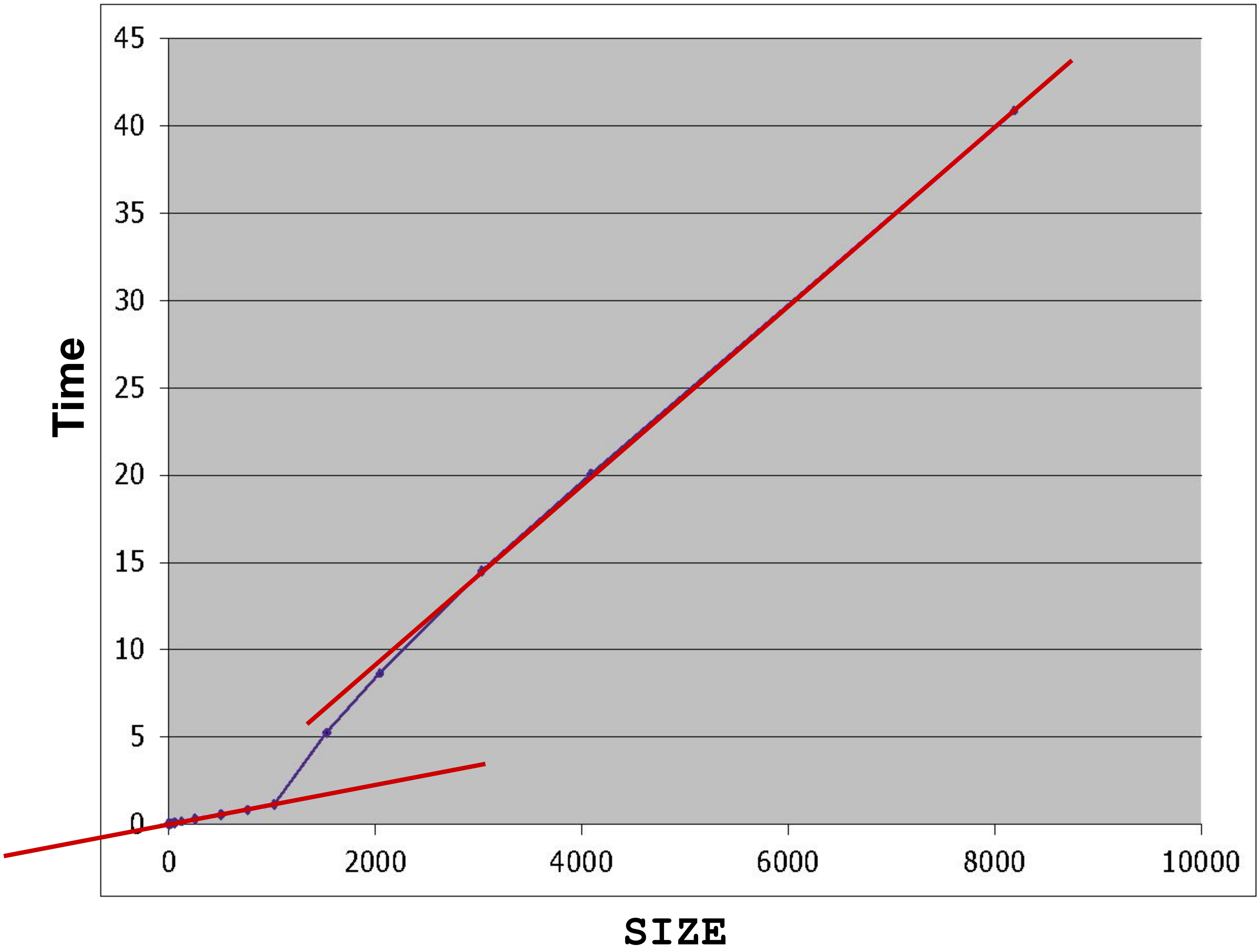
```
int array[SIZE];
int sum = 0;

for (int i = 0; i < 200000; i++) {
  for (int j = 0; j < SIZE; j++) {
    sum += array[j];
  }
}
```

Plot:



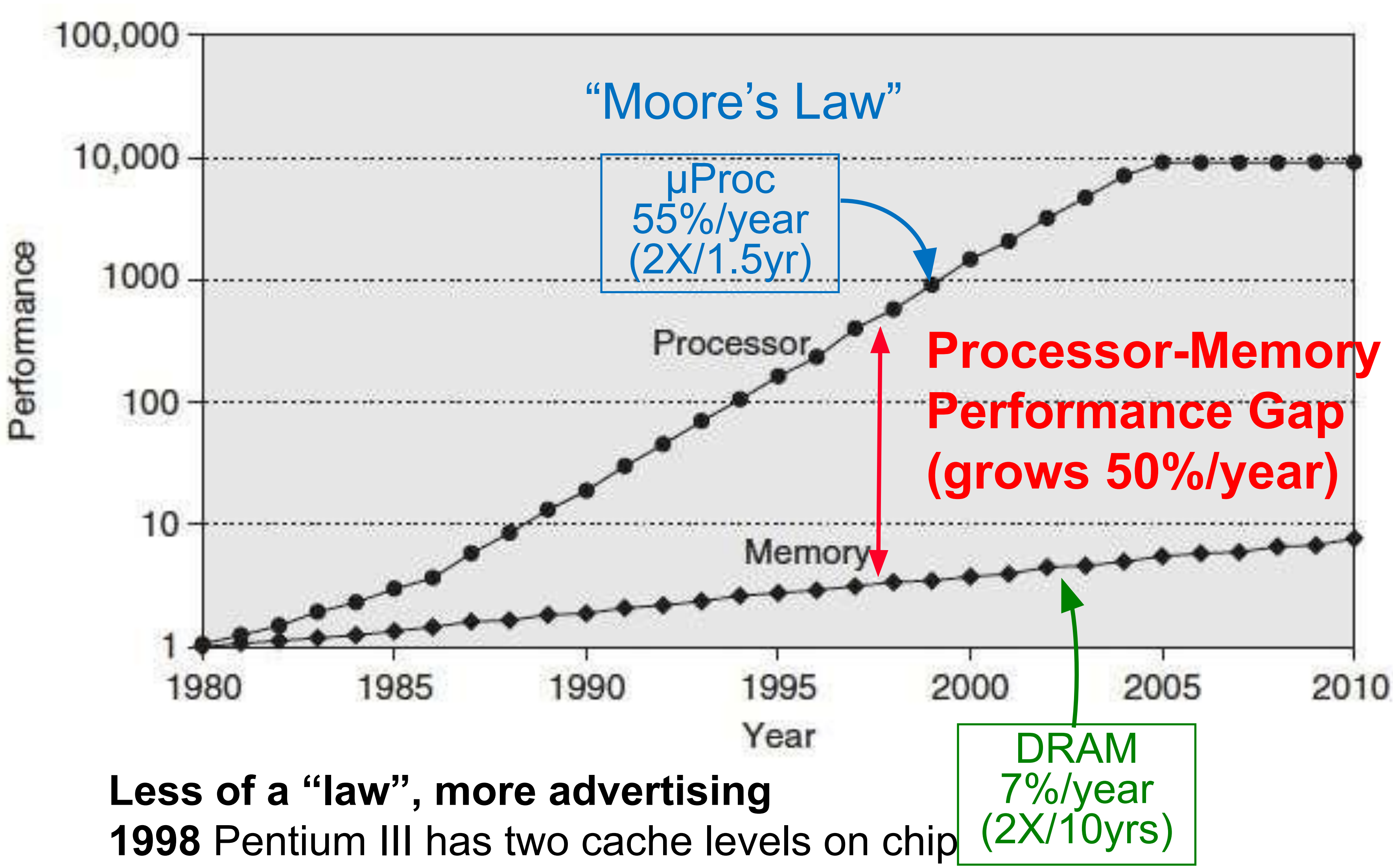
Actual Data



Making memory accesses fast!

- **Cache basics**
- **Principle of locality**
- **Memory hierarchies**
- Cache organization
- Program optimizations that consider caches

Processor-Memory Gap



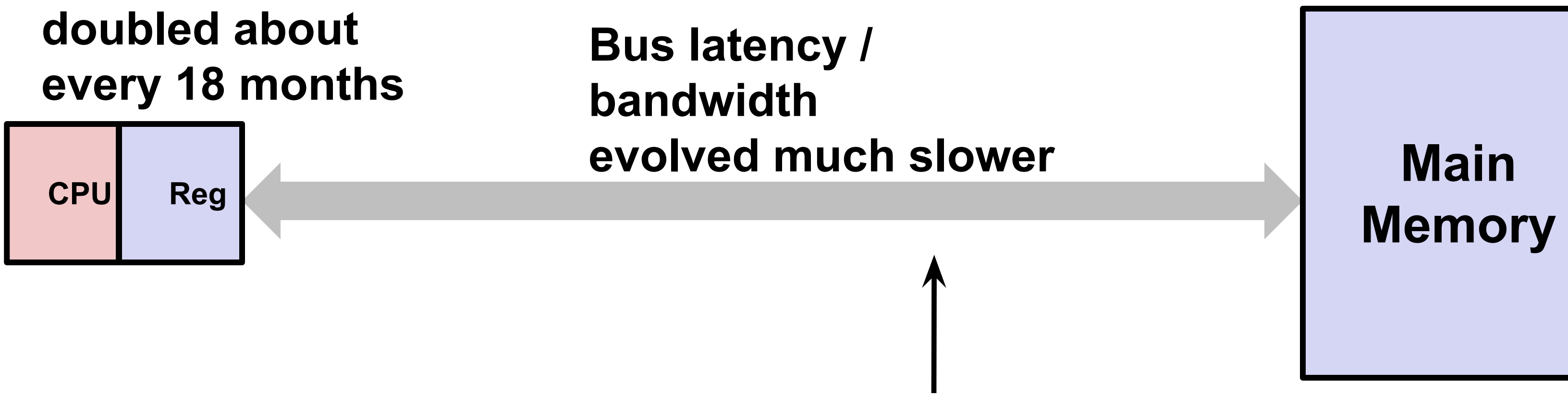
Less of a "law", more advertising

1998 Pentium III has two cache levels on chip

Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months

Bus latency / bandwidth evolved much slower



Core 2 Duo:
Can process at least
256 Bytes/cycle

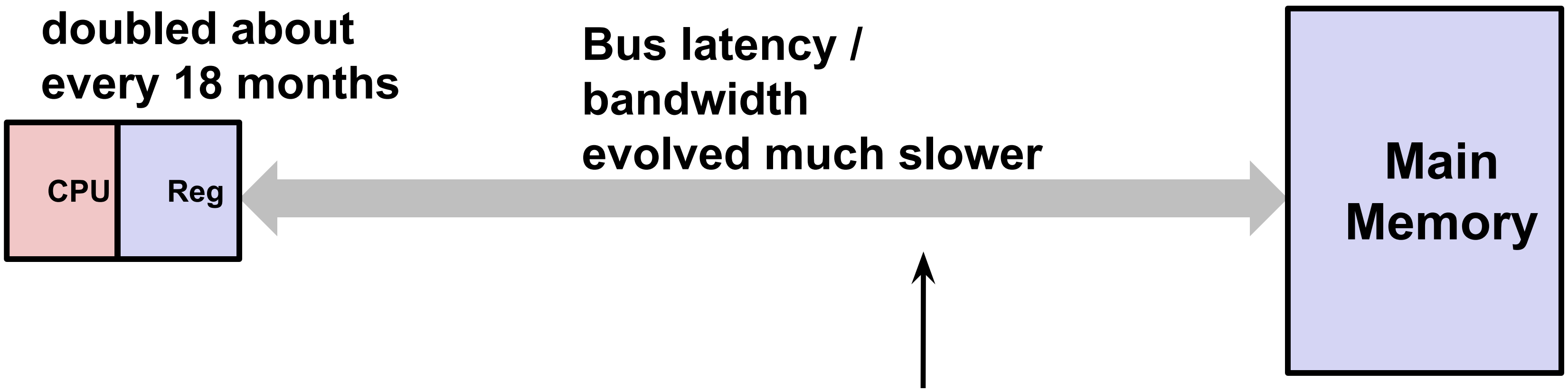
Core 2 Duo:
Bandwidth
2 Bytes/cycle
Latency
100-200 cycles
(30-60ns)

Problem: lots of waiting on memory

cycle: *single machine step (fixed-time)*

Problem: Processor-Memory Bottleneck

Processor performance doubled about every 18 months



Core 2 Duo:
Can process at least
256 Bytes/cycle

Core 2 Duo:
Bandwidth
2 Bytes/cycle
Latency
100-200 cycles (30-60ns)

Solution: Caches

cycle: *single machine step (fixed-time)*

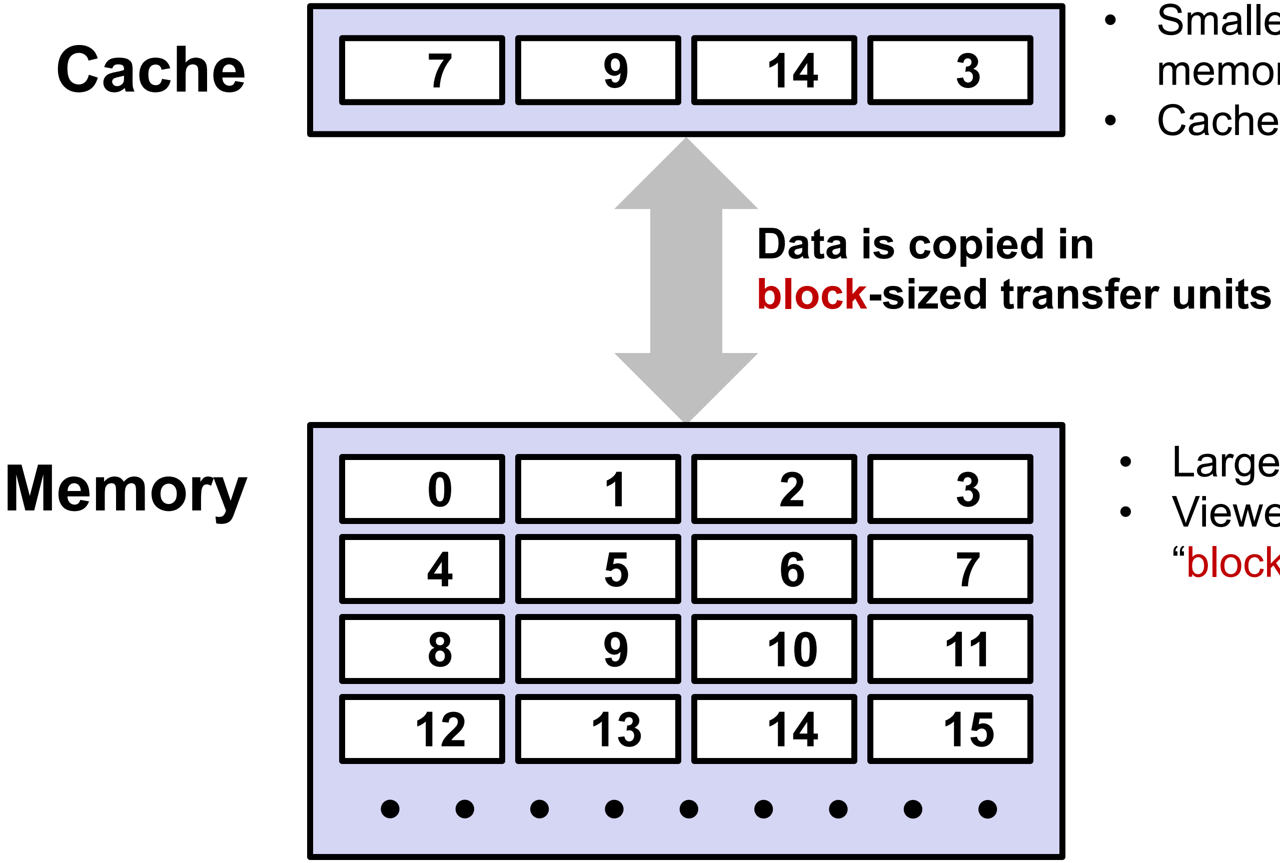
Cache 💰

- Pronunciation: "cash"
  - We abbreviate this as "$"
- English: A hidden storage space for provisions, weapons, and/or treasures
- Computer: Memory with short access time used for the storage of frequently or recently used instructions (i-cache/I$) or data (d-cache/D$)
  - *More generally:* Used to optimize data transfers between any system elements with different characteristics (network interface cache, I/O cache, etc.)

Storage!

FLOUR CITY-FLOWER CITY-CITY OF ROCHESTER, N.Y.®

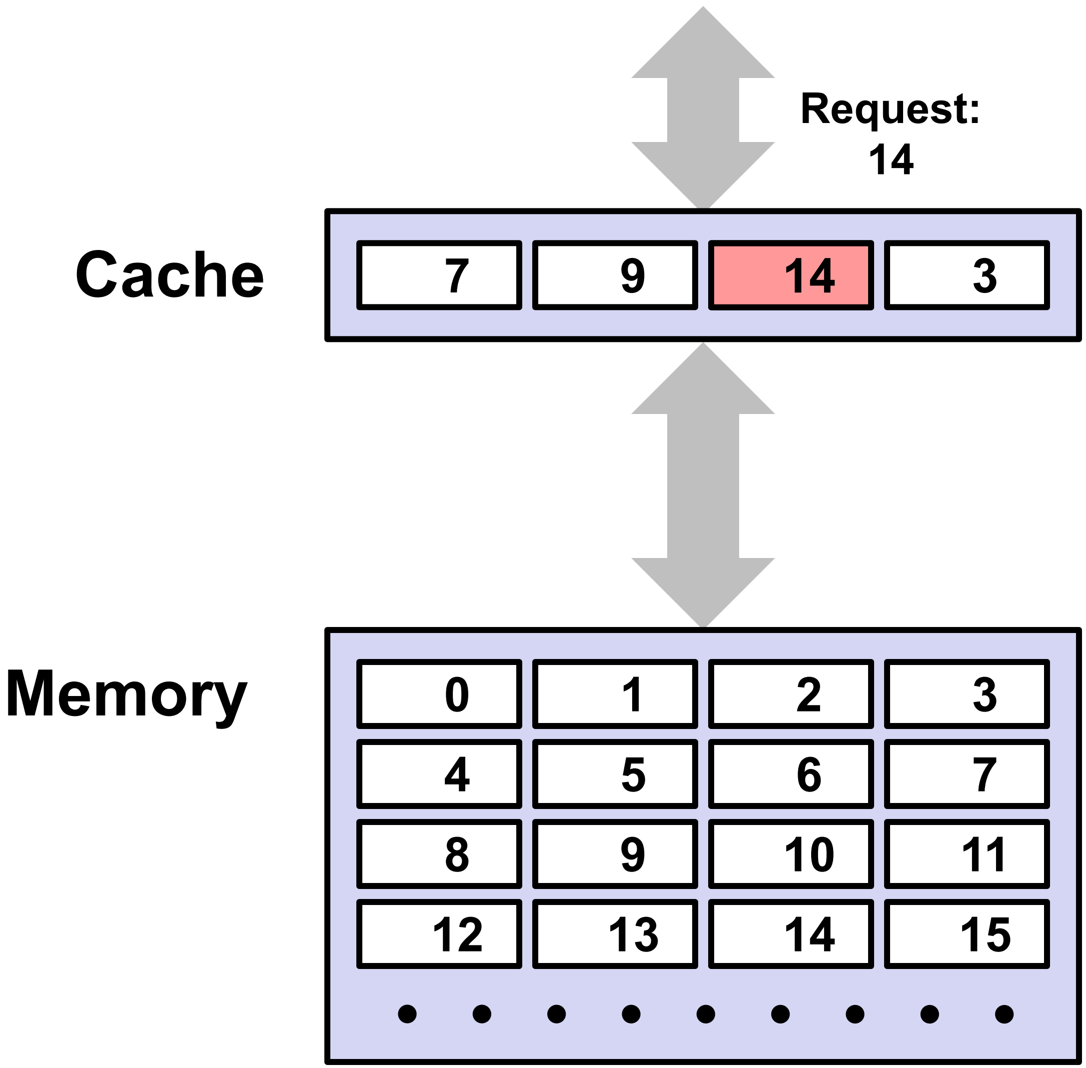
General Cache Mechanics



- Smaller, faster, more expensive memory
- Caches a subset of the blocks

- Larger, slower, cheaper memory.
- Viewed as partitioned into "blocks"

General Cache Concepts: Hit

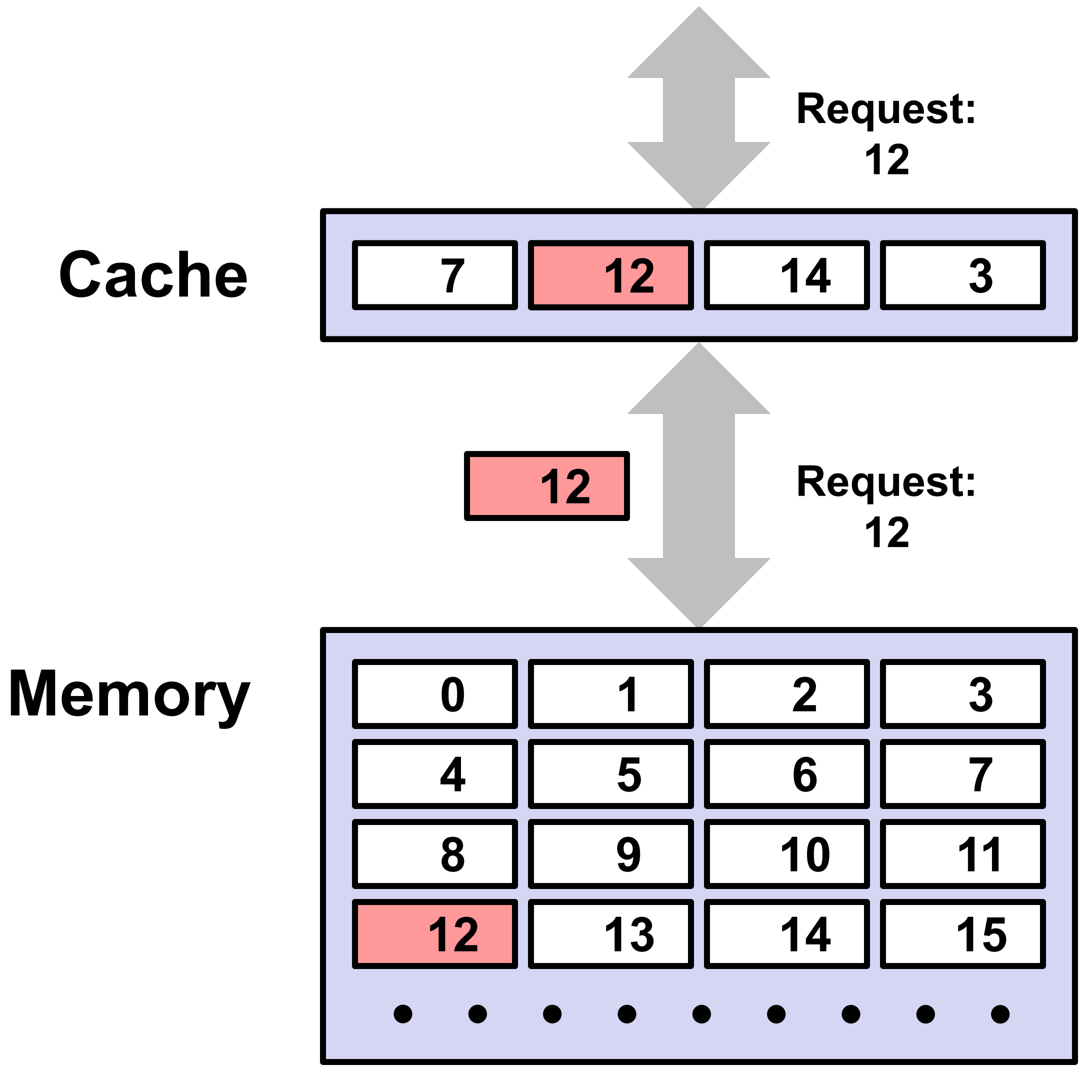


Data in block b is needed

Block b is in cache:
Hit!

Data is returned to CPU

General Cache Concepts: Miss



Data in block b is needed

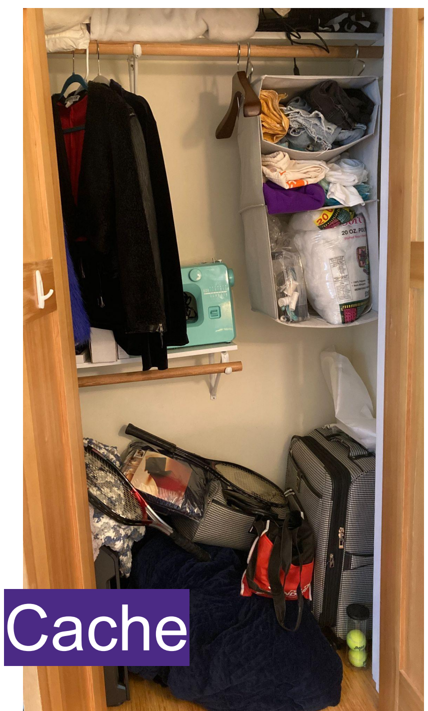
Block b is not in cache:
Miss!

Block b is fetched from memory

Block b is stored in cache
- Placement policy: determines where b goes
- Replacement policy: determines which block gets evicted (victim)

Data returned to CPU

Storage! An analogy



Cache

Main Memory

How do you feel about caches so far?

Why Caches Work

- Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

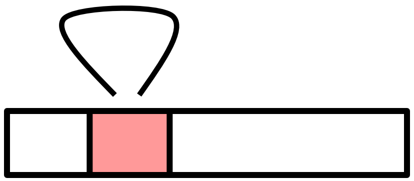
Why Caches Work

- Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

- *Temporal* locality:
  - Recently referenced items are *likely* to be referenced again in the near future

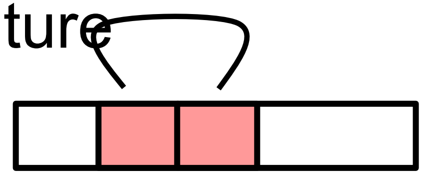
Why Caches Work

- Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently



block

- *Temporal* locality:
  - Recently referenced items are *likely* to be referenced again in the near future



block

- *Spatial* locality:
  - Items with nearby addresses *tend* to be referenced close together in time

- How do caches take advantage of this?

Example: Any Locality?

```
sum = 0;
for (i = 0; i < n; i++)
{
      sum += a[i];
}
return sum;
```

- **Data:**
  - Temporal: `sum` referenced in each iteration
  - Spatial: consecutive elements of array `a[]` accessed
- **Instructions:**
  - Temporal: cycle through loop repeatedly
  - Spatial: reference instructions in sequence

Locality Example #1

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];

    return sum;
}
```

Locality Example #1

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];

    return sum;
}
```

M = 3, N=4

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Access Pattern:
stride = ?

1) a[0][0]
2) a[0][1]
3) a[0][2]
4) a[0][3]
5) a[1][0]
6) a[1][1]
7) a[1][2]
8) a[1][3]
9) a[2][0]
10) a[2][1]
11) a[2][2]
12) a[2][3]

Layout in Memory

a [0] [0]	a [0] [1]	a [0] [2]	a [0] [3]	a [1] [0]	a [1] [1]	a [1] [2]	a [1] [3]	a [2] [0]	a [2] [1]	a [2] [2]	a [2] [3]

76　　92　　108

Note: 76 is just one possible starting address of array a

Locality Example #2

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];

    return sum;
}
```

Locality Example #2

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];

    return sum;
}
```

M = 3, N=4

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Access Pattern:
stride = ?

1) a[0][0]
2) a[1][0]
3) a[2][0]
4) a[0][1]
5) a[1][1]
6) a[2][1]
7) a[0][2]
8) a[1][2]
9) a[2][2]
10) a[0][3]
11) a[1][3]
12) a[2][3]

Layout in Memory

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[2][0]	a[2][1]	a[2][2]	a[2][3]

76 92 108

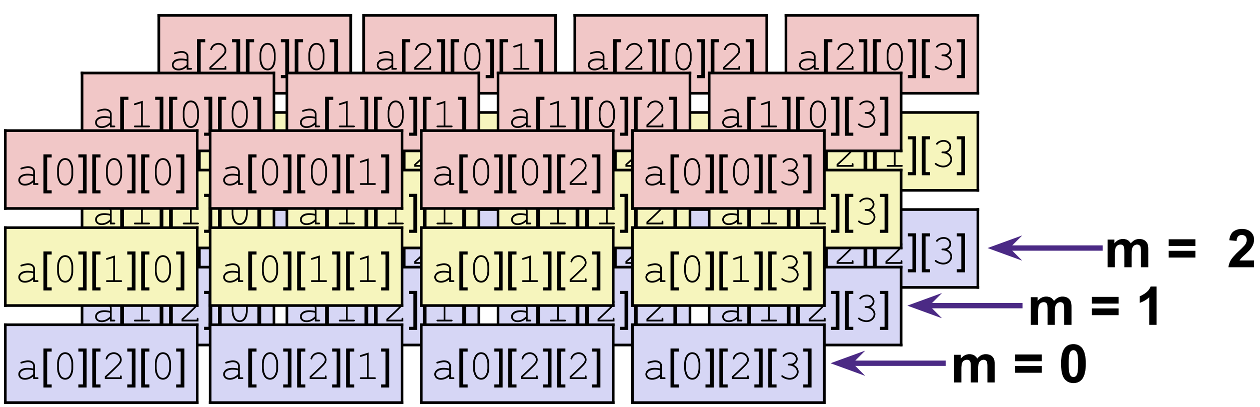
Locality Example #3

```
int sum_array_3D(int a[M][N][L])
{
    int i, j, k, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < L; j++)
            for (k = 0; k < M; k++)
                sum += a[k][i][j];

    return sum;
}
```

- What is wrong with this code?
- How can it be fixed?



Locality Example #3

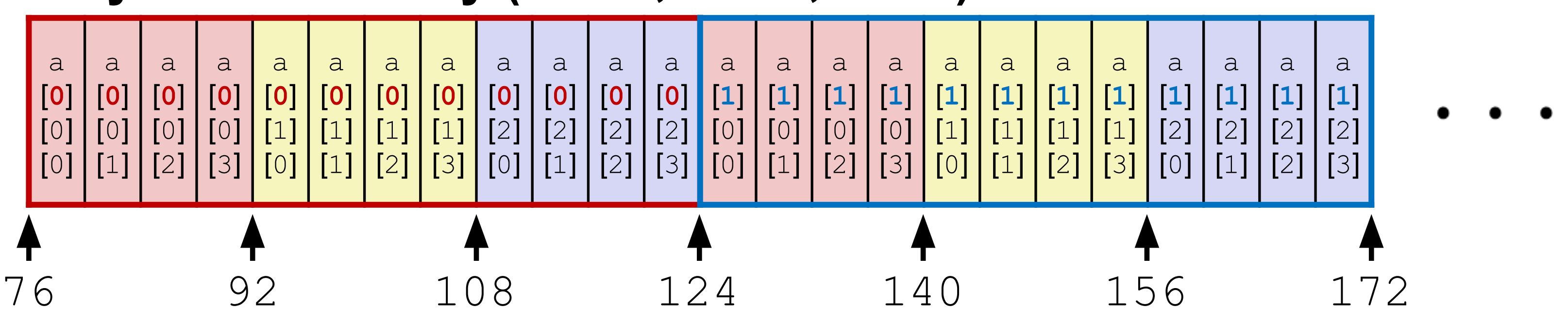
```
int sum_array_3D(int a[M][N][L])
{
    int i, j, k, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < L; j++)
            for (k = 0; k < M; k++)
                sum += a[k][i][j];

    return sum;
}
```

- What is ~~wrong~~ *inefficient* with this code?
- How can it be fixed?

Layout in Memory (M = ?, N = 3, L = 4)



How do you feel about cache locality?

Cache Metrics

- Huge difference between cache hit & misses
  - Could be 100x speed difference between accessing cache and main memory (measured in *clock cycles*)
- Miss Rate (MR)
  - Fraction of memory references not found in cache (misses / accesses) = 1 - Hit Rate
- Hit Time (HT)
  - Time to deliver a block in the cache to the processor
    - Includes time to determine whether the block is in the cache
- Miss Penalty (MP)
  - Additional time required because of a miss

Cache Metrics

- Two things hurt the performance of a cache:
  - Miss rate and miss penalty
- *Average Memory Access Time* (AMAT): average time to access memory considering both hits and misses

  AMAT = Hit time + Miss rate × Miss penalty

  (abbreviated AMAT = HT + MR × MP)

- 99% hit rate twice as good as 97% hit rate!
  - Assume HT of 1 clock cycle and MP of 100 clock cycles
  - 97%: AMAT = 1 + .03(100) = 4
  - 99%: AMAT = 1 + .01(100) = 2

Checking in!

- **Processor specs:** 200 ps clock, MP of 50 clock cycles, MR of 0.02 misses/instruction, and HT of 1 clock cycle

AMAT = 1 + (0.02)(50) = 2 cycles, 400ps

Checking in!

- **Processor specs:** 200 ps clock, MP of 50 clock cycles, MR of 0.02 misses/instruction, and HT of 1 clock cycle

  AMAT = 1 + (0.02)(50) = 2 cycles, 400ps

- Which improvement would be best?

🐶 **190 ps clock**

🐈 **Miss penalty of 40 clock cycles**

🦄 **MR of 0.015 misses/instruction**

🥶 **Help!**

More than one cache? Of course!

- Let's avoid going to memory!
- Typical performance numbers:
  - Miss Rate
    - L1 MR = 3-10%
    - L2 MR = Quite small (usually <1%)
  - Hit Time
    - L1 HT = 4 clock cycles
    - L2 HT = 10 clock cycles
  - Miss Penalty
    - P = 50-200 cycles for missing in L2 & going to main memory
    - Trend: increasing!

Summary

- Memory Hierarchy
  - Successively higher levels contain "most used" data from lower levels
  - Exploits *temporal and spatial locality*
  - Caches are intermediate storage levels used to optimize data transfers between any system elements with different characteristics
- Cache Performance
  - Ideal case: found in cache (hit)
  - Bad case: not found in cache (miss), search next level
  - Average Memory Access Time = HT + MR × MP
    - Hurt by Miss Rate and Miss Penalty

Metrics

Metrics in Computing

- Generally, folks care most about performance
  - Energy-efficiency is more important now!
    - Basically since the plateau in 2004/2005
  - That's why we have so many specialized chips

- Really, this is just efficiency -- making efficient use of the resources that we have
  - Performance: *cycles/instruction, seconds/program*
  - Memory: *bytes/program, bytes/data structure*
  - Efficiency: *performance/watt*

Ideology:

- What's taken for granted?
- What's so assumed to be true that we don't need to ask?

Efficiency & performance tends to be most important

Metrics

- What do we do with metrics?
  - We tend to optimize along them!
  - Especially when jobs/funding depend on better performance along some metric
    - *See all of Intel under "Moore's Law"*

Metrics

- What do we do with metrics?
  - We tend to optimize along them!
  - Especially when jobs/funding depend on better performance along some metric
    - *See all of Intel under "Moore's Law"*
- Sometimes, strange incentives emerge
  - "Minimize the number of bugs on our dashboard"
    - Does it count if we make the bugs invisible?
  - "Make this faster for our demo in a week"
    - Shortcuts might hurt performance at scale
  -

Ok, back to caches

Why do we have caches?

- Memory accesses are expensive!
  - Massive speedups to processors without similar speedups in memory only made the problem worse
  - “Processor-Memory Bottleneck”
- The entire chip industry depends on “a brand new laptop” meaning something
  - Consumers want speedups, engineers should deliver
  - Self-fulfilling, industry taught consumers to believe “faster is better”

Success equated with success along metrics!

Success and metrics

- Let’s say that we choose performance/program:
  - (in reality, performance/program set -- benchmarks)
  - Success means improving performance/program
  - We’ll measure existing performance
  - We’ll think of a bunch of optimizations that would improve performance
  - We’ll select a few to build into the “next version”
  - We’ll build, measure, and define success by the metrics that we’ve chosen

Success and metrics

- Let’s say that we choose profit/year
  - (I’m only a little cynical, it’s more like stock price)
  - Success means earning more profit than last year
  - We’ll think of improvements along this!
    - Reduce expenses, cut staff
    - Sell more things
    - Sell fancier things
    - Make people pay monthly for things they could get for free
- Yes this is overly simplistic.
- It’s usually a bit more complicated!

Success and metrics

- Let’s say that we want to improve participation of minoritized people in computing
  - “minoritized” -- I didn’t choose my oppression
- What might we measure, to see if we’re successful?
  - Percentage of women in STEM majors?
  - Percentage of BIPOC folks in STEM majors?
- If we define success by these metrics, what might go wrong?

What might go wrong?

- We'd need to be careful about how we define participation!
  - Just intro class? People might drop out/leave
  - Just exiting class? People might've moved in

What might go wrong?

- We'd need to be careful about how we define participation!
  - Just intro class? People might drop out/leave
  - Just exiting class? People might've moved in
- What policies might we pursue?
  - Quota recruiting?
  - Mentoring programs?

We might miss things!

Belonging

- **Defn:** Feeling like you *belong* in a space!
  - Seeing people that look like you (vicarious action)
  - Value alignment between you and your discipline
  - Feeling happy in a space! Friends in that space!
  - *Feeling that a space is accessible! Support & Agency!*

Belonging

- **Defn:** Feeling like you *belong* in a space!
  - Seeing people that look like you (vicarious action)
  - Value alignment between you and your discipline
  - Feeling happy in a space! Friends in that space!
  - *Feeling that a space is accessible! Support & Agency!*
- Note that "percentage participation" doesn't say anything about these
  - We might miss this!
  - *Research says that this is a large part of retention*

But wait, weren’t we talking about caches?

Caches

- We defined “locality”, based on observations about existing programs, written by an extremely small subset of the population
  - We built hardware that utilizes locality to improve performance (our metric)
  - We improved performance!

Caches

- We defined "locality", based on observations about existing programs, written by an extremely small subset of the population
  - We built hardware that utilizes locality to improve performance (our metric)
  - We improved performance!
- We also made decisions about "good/bad" code!
  - "Good" code exhibits "good" locality
    - Basically, it's like all the other code
  - "Hierarchies of knowledge through value judgements"

Caches

- What metric are we using to measure success?
  - *Probably performance/program, others*
- What optimizations might we choose?
  - Faster processors?
  - Faster memory?
  - More complexity, for more performance?
- What might we build?
  - *Specialized, expensive hardware that's a huge source of complexity and bugs, that regularly confuses 351 students with its complexity???*

Regardless of what we build, the way that we define success shapes the systems we build!

Choose your metrics carefully!
There’s more to choose than “performance”!
i.e. usability, access, simplicity, agency

Metrics are a “heading”

And, most often, and ideological one.

Best to reevaluate from time to time!

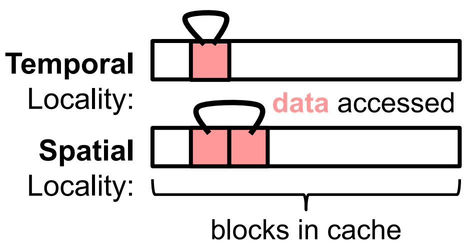
Discuss!

What other metrics affect the design and building of structures?

It doesn't need to be nefarious or neoliberal capitalist!

Emergency Department response time!

Notes Diagrams



Handout: Any Locality?

```
sum = 0;
for (i = 0; i < n; i++)
{
      sum += a[i];
}
return sum;
```

- **Data:**
  - Temporal:
  - Spatial:
- **Instructions:**
  - Temporal:
  - Spatial: