Memory Hierarchy

2024 Fall ECE454: Computer Systems Programming
Lecture 7

Jon Eyolfson

1.0.0

Matrix Multiply

```
double a[4][4];
double b[4][4];
double c[4][4];

/* Multiply n x n matrices a and b  */
void mm(double *a, double *b, double *c, int n) {
  for (int i = 0; i < n; i++) {
          for (int j = 0; j < n; j++) {
      for (intk = 0; k < n; k++) {
              c[i][j] += a[i][k] * b[k][j]; // actual work
      }
    }
  }
}
```

How much performance improvement can we get by optimizing this code?

We Can Get At Least a 160x Speedup

We don't skip any operations, both implementations have $2n^3$ operations

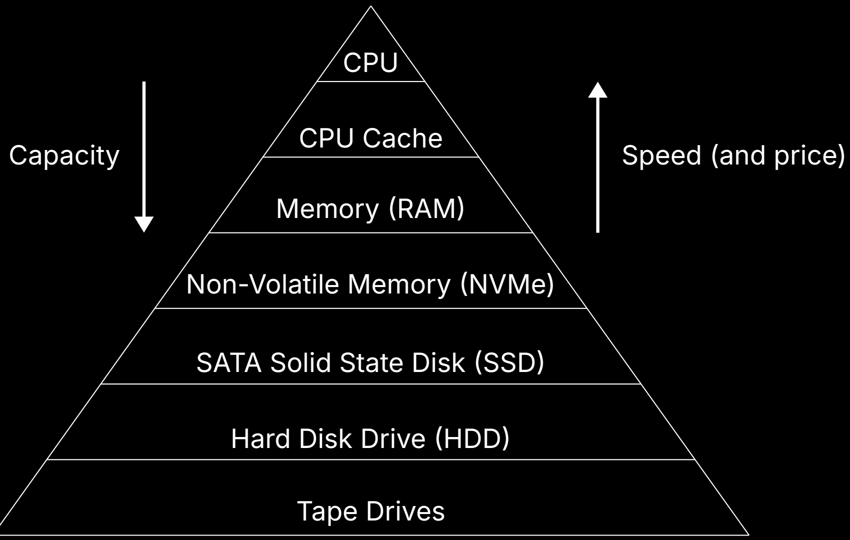
L1 cache reference time is 1-4 ns
  However, L1 cache size $<=$ 64 KB

Main memory reference time = 100 ns, 100x slower!
  However, memory size $>=$ GBs

Some data:
  1 ns = 1/1,000,000,000 second

For a 3.5 GHz CPU (base clock AMD Ryzen 7640U), 1 cycle $\approx$ 0.3 ns

Computer Memory Hierarchy is a Trade-off of Capacity and Speed



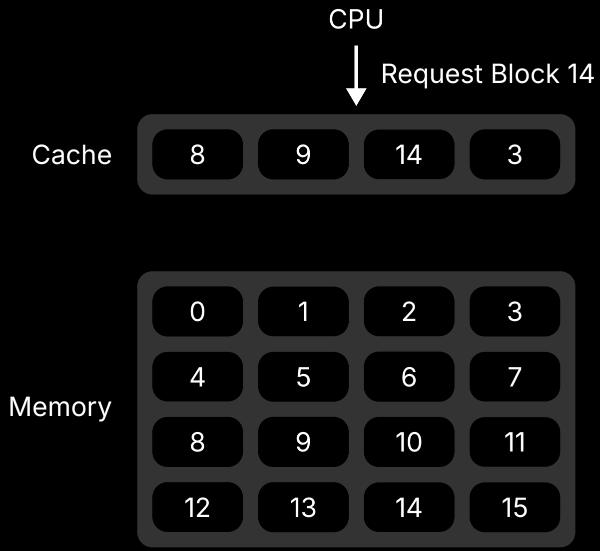
General Cache Mechanics

We transfer blocks of memory at a time

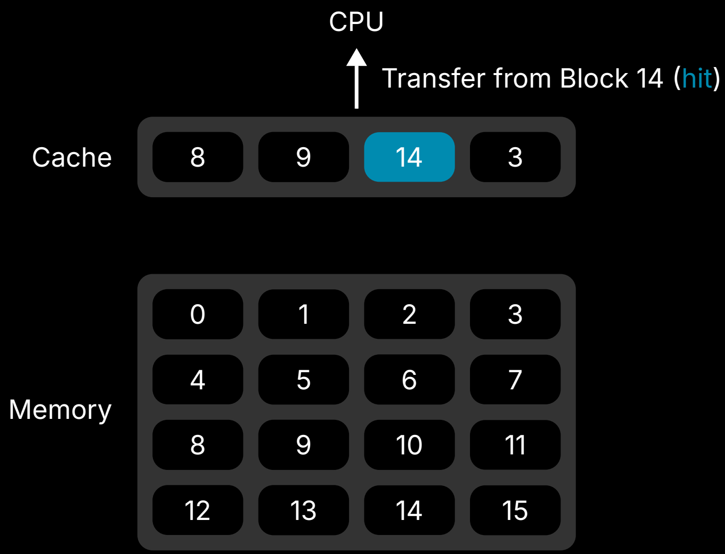
A smaller, faster, more expensive memory caches a subset of the blocks

Larger, slower, cheaper memory viewed as partitioned into fixed-size blocks

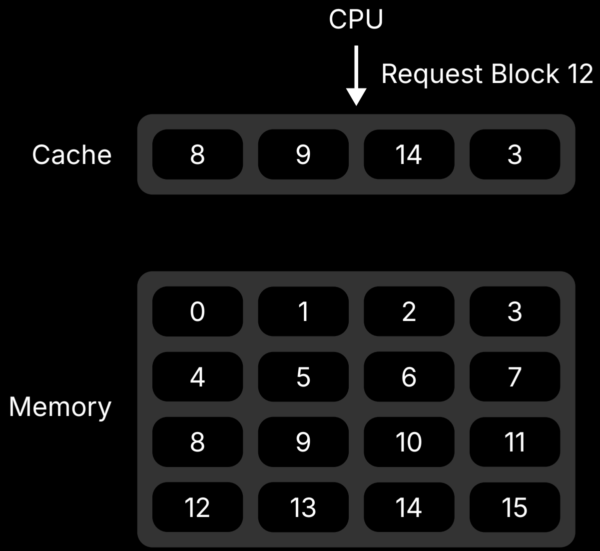
Caching Example



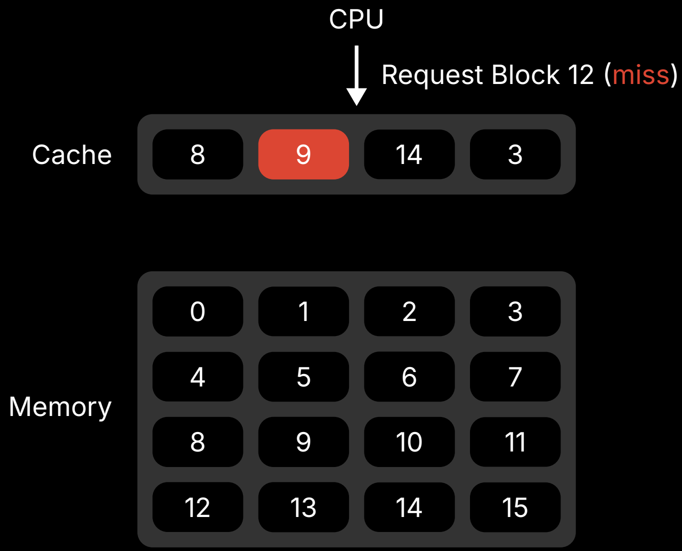
Caching Example



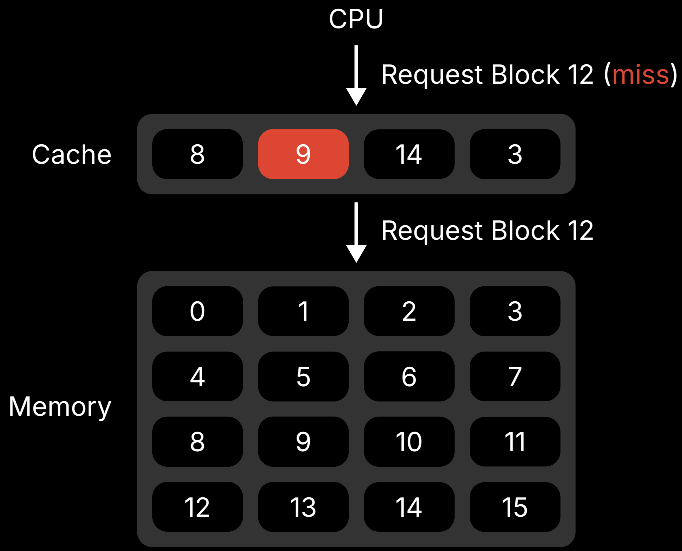
Caching Example



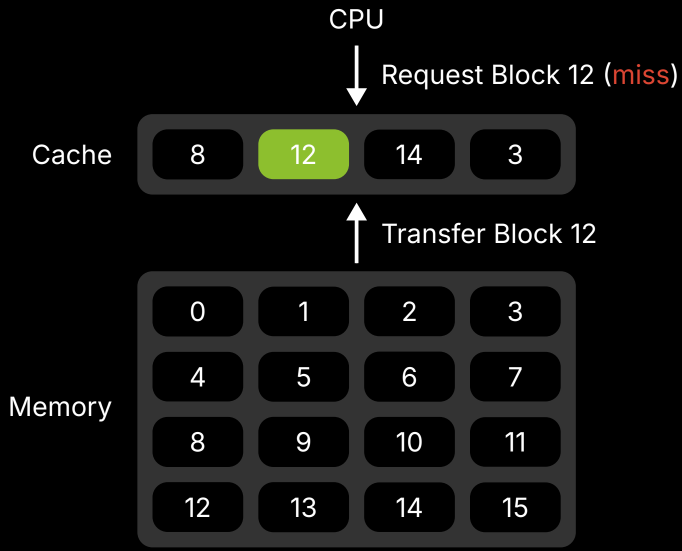
Caching Example



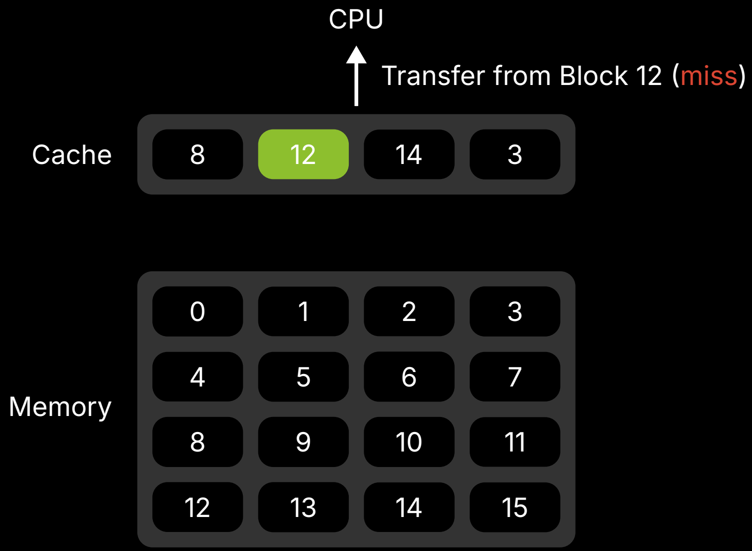
Caching Example



Caching Example



Caching Example



There Are Two Policies

Placement policy: Chooses a set of blocks
where a block (e.g. 12) goes in cache

Replacement policy: Determines which block in set gets evicted (victim)

Cache Performance Metrics

Miss Rate
- Fraction of memory references not found in cache
- miss rate = misses / accesses = 1 - hit rate
- 3-10% for L1, small (e.g., < 1%) for L2, depending on size, etc.

Hit Time
- Time to deliver a line in the cache to the processor
- Includes time to determine whether the line is in the cache
- 1-4 clock cycles for L1, 5-20 clock cycles for L2

Miss Penalty
- Additional time required due to a miss
- Typically 50-400 cycles for main memory

The Numbers in Context

Huge difference between a hit and a miss
  100x between L1 and main memory

Performance with 99% hit rate doubles compared to 97%!
  Say cache hit time = 1 cycle, miss penalty of 100 cycles
  Average access time:
    97% hits: 1 cycle + 0.03 ∗ 100 cycles = 4 cycles
    99% hits: 1 cycle + 0.01 ∗ 100 cycles = 2 cycles

This is why miss (instead of hit) rate is used to think about cache performance, 3% is much worse than 1% miss rate

Cold Cache Misses Can’t Be Avoided

Occurs on first access to a block

Can’t do too much about these (except prefetching—more later)

Two Major Types of Cache Misses

Conflict miss

Placement policy of most hardware caches limit blocks to a small subset (sometimes a singleton) of the available cache slots
- e.g., block i must be placed in slot (i mod 8)

Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot
- e.g., referencing blocks 0, 8, 0, 8, ... would miss every time

Conflict misses are less of a problem today (more later)

Capacity miss

Occurs when the set of active cache blocks is larger than the cache
- Working set is larger than cache size

This is the most significant problem today

Why Caches Work

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

Temporal locality:
Recently referenced items are likely to be referenced soon after

Spatial locality:
Items with nearby addresses tend to be referenced close together

Locality Example

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

Data

Temporal: sum referenced in each iteration

Spatial: close by elements of array a accessed (in stride-1 pattern)

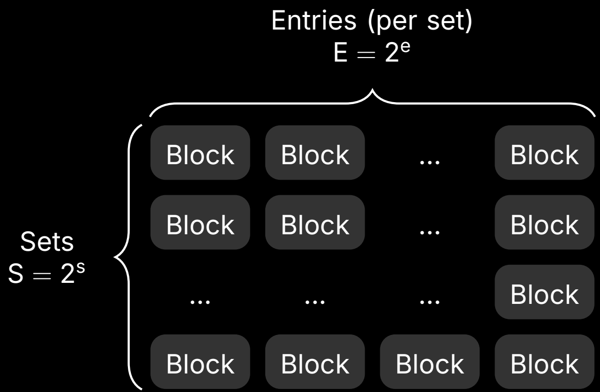
Instructions

Temporal: cycle through loop repeatedly

Spatial: reference close by instructions in sequence

Important to be able to assess the locality in your code!

General Cache Organization (L1, L2, and L3)



We can define the block size as: $B = 2^b$

Therefore, Cache Size $= S \times E \times B = 2^{s+e+b}$

How To Get the Tag from an Address

Address	remaining bits	s bits	b bits
	Tag	Set	Offset

For most modern systems, the cache block size is 64 bytes, b = 6 bits

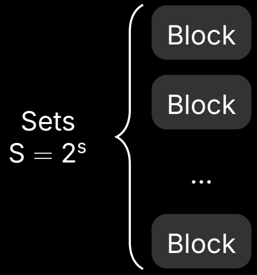
A Cache Entry Stores if It’s Valid, the Tag, and Data

1 bit	tag bits	64 bytes
V	Tag	Data

Direct Mapped Cache (E = 1)

There is only one block per set

That means that only one entry from each tag can be in cache at a time



Example Lookup

Direct mapped cache with 64 sets
- 6 offset bits
- 6 set bits

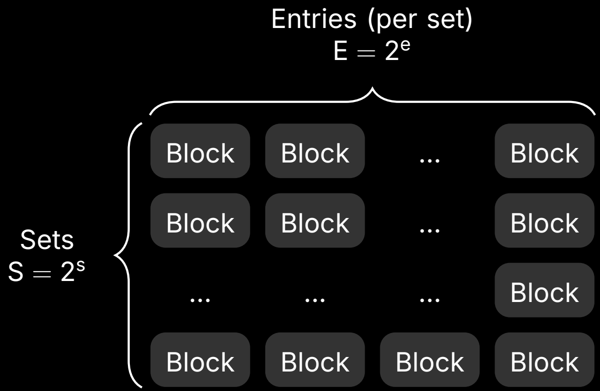
First, we find which set the entry would be in, then see if the tag matches
- If it's not a match, old block is evicted and replaced with entire new block

Address: `0xFEEDFACECAFEBEEF`
- Set: `0b111011` = 59
- Tag: `0xFEEDFACECAFEB`
- Data: `0xFEEDFACECAFEBEC0` - `0xFEEDFACECAFEBEFF`

Adddress: `0x1EC8`
- Set: `0b111011` = 59
- Tag: `0x1`
- Data: `0x1EC0` - `0x1EFF`

The Number of Entries in a Set is Called the Associativity

For an 8-way associative cache, we can store 8 different tags per set



For an access, it scans the set for a matching tag
- If it's not in cache we need to replace an existing block
  - Replacement policies: random, least recently used (LRU), etc.

Example CPU - AMD Ryzen 5 7640U

L1 Cache: 64 KiB (per core)
There's two L1 caches: one for instructions, the other for data,
8-way set associative cache
- Each L1 each is 32 KiB (2^{15}), therefore it has 64 sets (2^{15-6-3})

L2 Cache: 1 MiB (per core)
This cache contains both instructions and data
8-way set associative cache

L3 Cache: 16 MiB (shared)
This cache contains both instructions and data
16-way set associative cache
- Cache latency: 50 cycles

Finding Cache Information on Linux

You can use the following commands:

```
lscpu
lstopo
```

You can also explore in the `/sys` directory

```
ls /sys/devices/system/cpu/cpu0/cache/
```

What About Writes?

- Multiple copies of data exist in L1, L2, main memory, disk
  - Need to ensure consistency
- What to do on a write-hit?
  - Write-through (write to cache and immediately to memory)
  - Write-back (defer write to memory until line is replaced)
    - Need a dirty bit (cache line different from memory or not)
- What to do on a write-miss?
  - Write-allocate (load into cache, update line in cache)
    - Good if more reads and writes to the location follow
  - No-write-allocate (write immediately to memory)
    - For streaming writes (write once and then no reads in the near future)
- Typically:
  - Write-through + No-write-allocate
  - Write-back + Write-allocate

The Best Way to Find Miss Rates is with `perf`

On most systems you can see the number of accesses and misses

```
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-dcache-store-misses                             [Hardware cache event]
  L1-dcache-prefetches                               [Hardware cache event]
  L1-dcache-prefetch-misses                          [Hardware cache event]
  L1-icache-loads                                    [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  L1-icache-prefetches                               [Hardware cache event]
  L1-icache-prefetch-misses                          [Hardware cache event]
```

You can see a list of supported events with `perf list`