

Computer Systems

CSE 410 Winter 2022
12 – Memory Organization and Caches

1

Memory and Caches

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

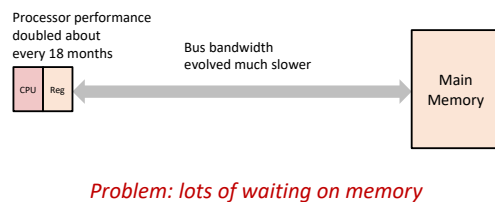
2

Making memory accesses fast!

- What we want: Memories that are
 - **Big**
 - Fast
 - Cheap
- Hardware: Pick any two
- So we'll be clever...
 - Pick "fast and cheap" (but not big), and
 - "big and cheap" (but not fast)

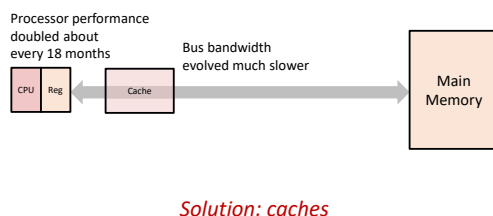
3

Problem: Processor-Memory Bottleneck



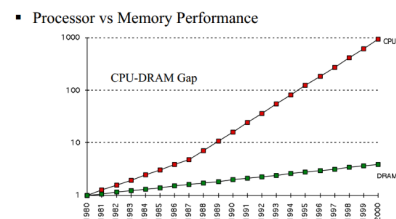
4

Problem: Processor-Memory Bottleneck



5

Cycle time vs. Memory Access Time



1980: no cache in microprocessor;
1995 2-level cache

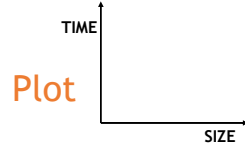
<https://www.hardwaredtimes.com/difference-between-l1-l2-and-l3-cache-how-does-cpu-cache-work/>

6

How does cache affect performance?

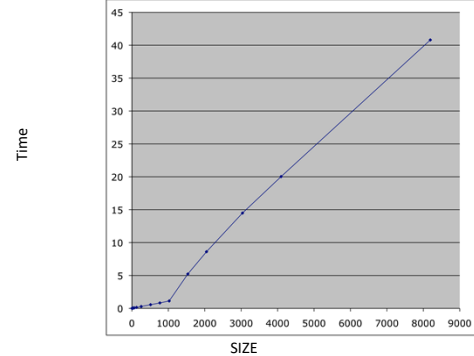
```
int array[SIZE];
int A = 0;

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



7

Actual Data



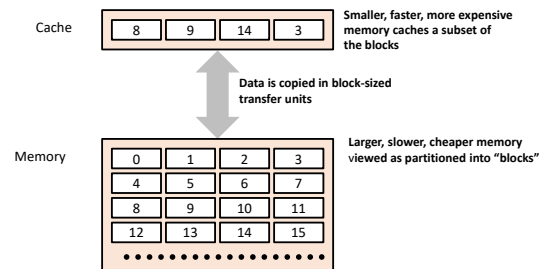
8

Cache

- **English definition:** a hidden storage space for provisions, weapons, and/or treasures
- **CSE definition:** computer memory with short access time used for the storage of frequently or recently used instructions or data (i-cache and d-cache)
- Used to optimize data transfers between system elements with different characteristics (main memory cache, network interface cache, I/O cache, etc.)

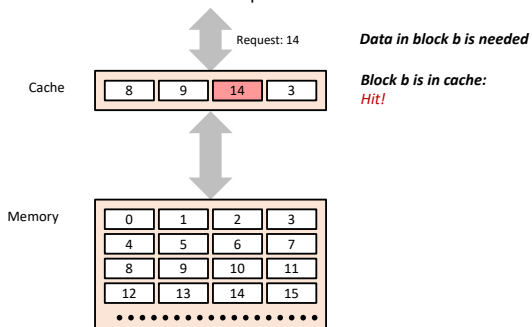
9

General Cache Mechanics



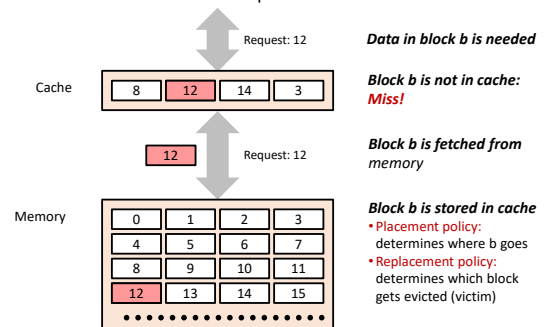
10

General Cache Concepts: **Hit**



11

General Cache Concepts: **Miss**




12

Cache Organization Questions

- How can the processor quickly determine whether a memory reference is a hit or a miss?
- If there's a miss, where in the cache should we put the data we have to retrieve from the layer(s) above?
 - Which data item currently in the cache should we overwrite with the new data?
- On a read miss, should the new data be put in the cache at all?
- On a write miss, should the written data be (a) put in the cache, or (b) written to the higher layer(s), or (c) both?

13

Core i7-8700K (2017)

CPU				Caches	Mainboard	Memory	SPD	Graphics	Bench	About
Processor										
Name		Intel Core i7								
Code Name		Coffee Lake		Max TDP		95.0 W				
Package		Socket 1151 LGA								
Technology		14 nm		Core Voltage		0.516 V				
Specification										
Intel® Core™ i7-8700K CPU @ 3.70GHz (E5)										
Family		6		Model		E		Stepping		A
Ext. Family		6		Ext. Model		9E		Revision		U0
Instructions		MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3, TSX								
Clocks (Core #0)										
Core Speed		4696.57 MHz								
Multiplier		x 47.0 (8 - 47)								
Bus Speed		99.93 MHz								
Rated FSB										
Cache										
L1 Data		6 x 32 KBytes		8-way						
L1 Inst.		6 x 32 KBytes		8-way						
Level 2		6 x 256 KBytes		4-way						
Level 3		12 MBytes		16-way						
Selection		Socket #1		Cores		6		Threads		12
CPU-Z Ver. 1.80.1.x64 Tools Validate Close										

14

Core i7-8700K (2017)

CPU

Caches

Mainboard

Memory

SPD

Graphics

Bench

About

L1 D-Cache

Size

32 KBytes

x 6

Descriptor

8-way set associative, 64-byte line size

L1 I-Cache

Size

32 KBytes

x 6

Descriptor

8-way set associative, 64-byte line size

L2 Cache

Size

256 KBytes

x 6

Descriptor

4-way set associative, 64-byte line size

L3 Cache

Size

12 MBytes

Descriptor

16-way set associative, 64-byte line size

Size

Descriptor

Speed

CPU-Z

Ver. 1.80.1.x64

Tools

Validate

Close

15

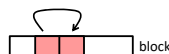
Memory and Caches

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

16

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
- **Spatial locality:**
 - Items with nearby addresses *tend* to be referenced close together in time
- How do caches take advantage of this?



17

Example: Locality?

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

- **Data:**
 - Temporal: `sum` referenced in each iteration
 - Spatial: array `a[]` accessed in stride-1 pattern (and blocks are larger than one word)
- **Instructions:**
 - Temporal: cycle through loop repeatedly
 - Spatial: reference instructions in sequence
- A high cache hit rate is essential to good performance
- Both the hardware designer and the programmer are concerned with it
- Being able to assess the locality of code is a crucial skill for a programmer

18

Another Locality Example

```
int sum_array_2d(int a[M][N])
{
    int i, j, sum = 0;

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

    return sum;
}
```

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

19

Another Locality Example

```
int sum_array_2d(int a[M][N])
{
    int i, j, sum = 0;

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

    return sum;
}
```

i=0 j=0	i=0 j=1	i=0 j=2	i=1 j=0	i=1 j=1	i=1 j=2	i=2 j=0	i=2 j=1	i=2 j=2
[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]	[2][0]	[2][1]	[2][2]
...

- Array is stored in *row major order*
- Accesses are in *column major order*
- No spatial locality

20

Memory and Caches

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

21

Cost of Cache Misses

- Huge difference between a hit and a miss
 - Could be 100x, if just L1 and main memory
- Would you believe 99% hits is twice as good as 97%?
 - Consider:
 - Cache hit time of 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 rate” is used instead of “hit rate”

22

Cache Performance Metrics

- Miss Rate
 - Fraction of memory references not found in cache (misses / accesses)
 - = 1 - hit rate
 - Typical numbers (in percentages):
 - 3% - 10% for L1
 - Can be worse for higher level
- Hit Time
 - Time to deliver a line in the cache to the processor
 - Includes time to determine whether the line is in the cache
 - Typical hit times: 1 - 2 clock cycles for L1
- Miss Penalty
 - Additional time required because of a miss
 - Typically 50 - 200 cycles

23

Memory Hierarchies

- Some fundamental and enduring properties of hardware and software systems:
 - Faster storage technologies almost always cost more per byte and have lower capacity
 - The gaps between memory technology speeds are widening
 - True for: registers ↔ cache, cache ↔ DRAM, DRAM ↔ disk, etc.
 - Well-written programs tend to exhibit good locality
- These properties complement each other beautifully
- They suggest an approach for organizing memory and storage systems known as a memory hierarchy

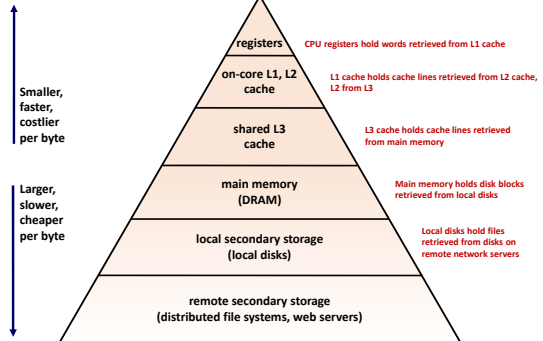
24

Memory Hierarchies

- Fundamental idea of a memory hierarchy:
 - Each level, k , serves as a cache for the larger, slower, level $k+1$, behind it
- Why do memory hierarchies work?
 - Because of locality, programs tend to access the data at level k more often than they access the data at level $k+1$.
 - Thus, the storage at level $k+1$ can be slower, and thus larger and cheaper per bit.
- Big Idea:** The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

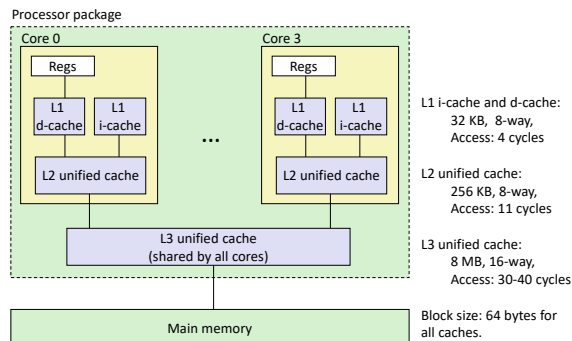
25

An Example Memory Hierarchy



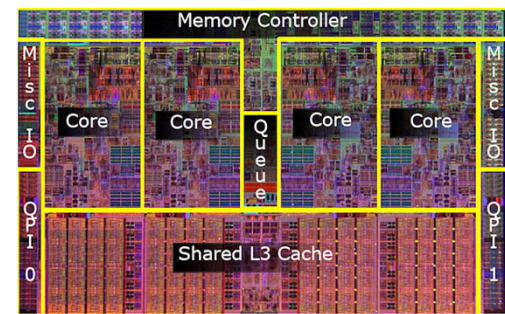
26

Intel Core i7 Cache Hierarchy



27

Intel i7 Die



28

Memory and Caches

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

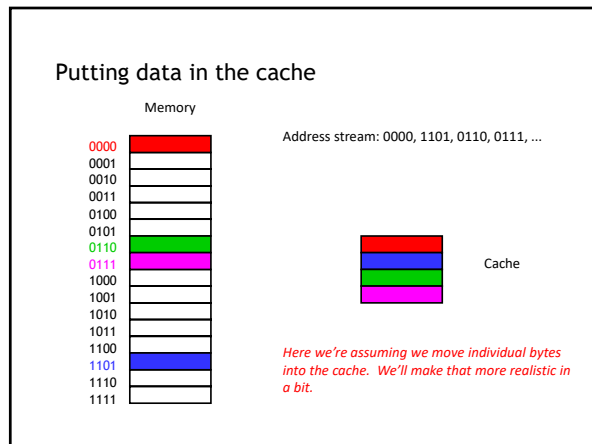
29

Core i7-8700K (2017)

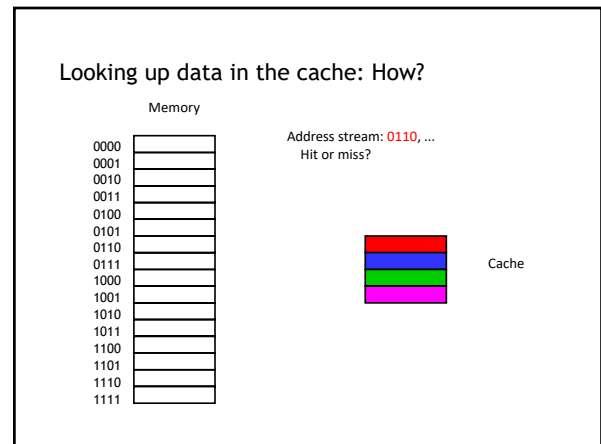
CPU Caches			
L1D-Cache	Size	32 KBytes	x 6
	Descriptor	8-way set associative, 64-byte line size	
L1I-Cache	Size	32 KBytes	x 6
	Descriptor	8-way set associative, 64-byte line size	
L2 Cache	Size	256 KBytes	x 6
	Descriptor	4-way set associative, 64-byte line size	
L3 Cache	Size	12 MBytes	
	Descriptor	16-way set associative, 64-byte line size	
	Size		
	Descriptor		
	Speed		

CPU-Z Ver. 1.80.1.x64 Tools Validate Close

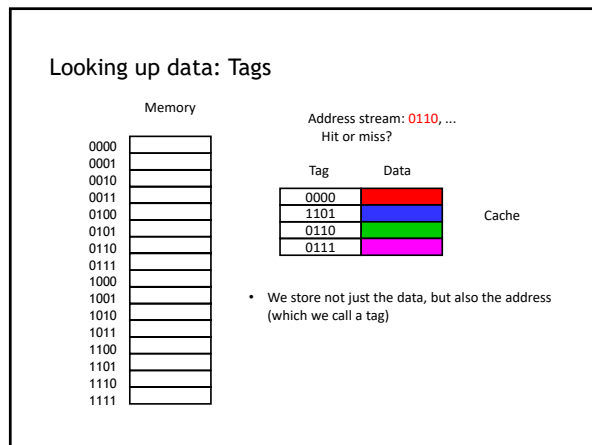
30



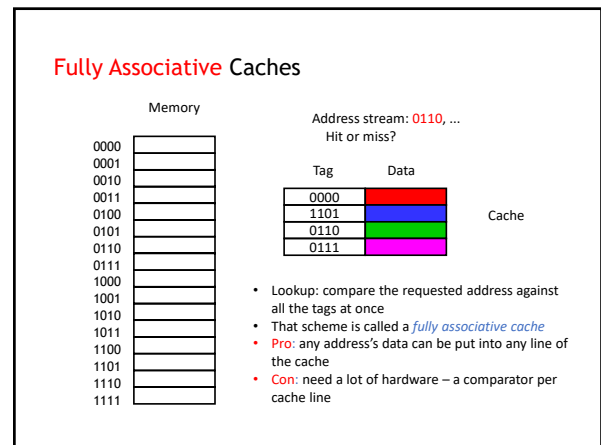
31



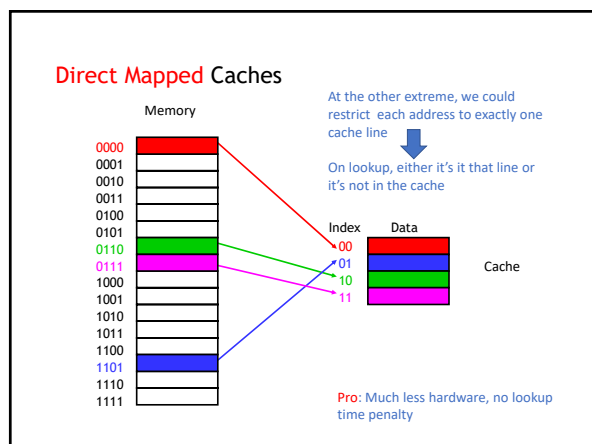
32



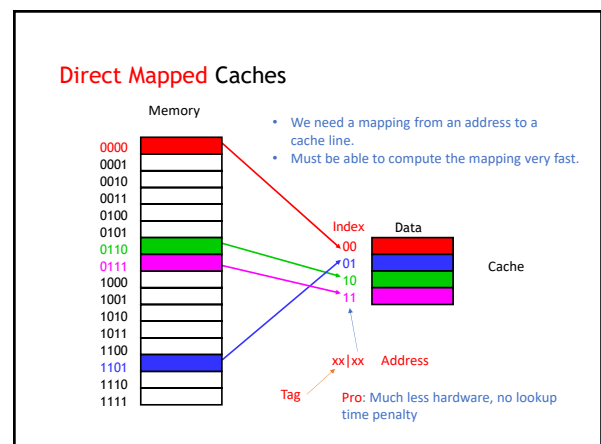
33



34

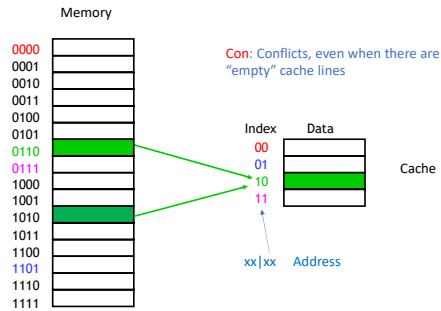


35



36

Direct Mapped Caches



37

Address \rightarrow tag|index

- Q: Why use low order bits for index and high bits for tag, rather than vice versa
 - tag | index, or
 - index | tag
 - A: Programs tend to sweep through memory sequentially
 - e.g., instructions
 - e.g., arrays
 - AND by using the low order bits as the index, sequential memory makes use of the entire cache
 - 00 | 00
 - 00 | 01
 - 00 | 11
 - 01 | 00
 - 01 | 01
 - 01 | 10
 - 01 | 11
- tag | index vs. index | tag

38

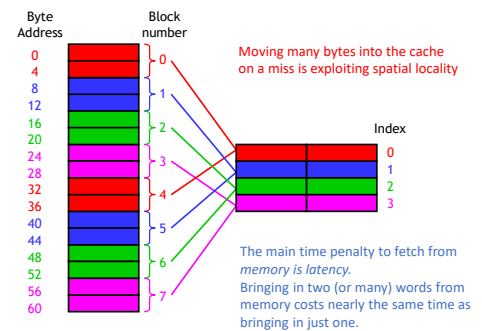
A small optimization

- We don't need to save the low order bits of the address in the tag storage, because they're implied by the line number



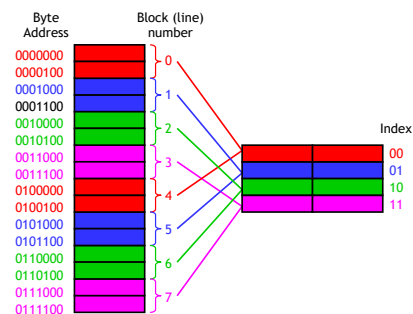
39

Exploiting Spatial Locality – Cache Blocks



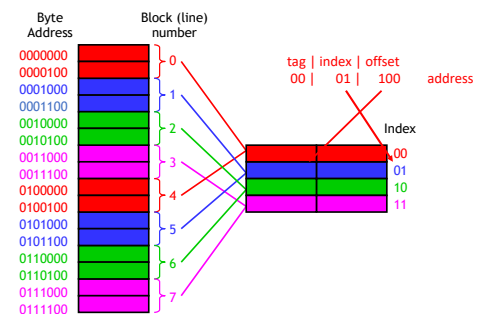
40

What's a cache block? (or *cache line*)



41

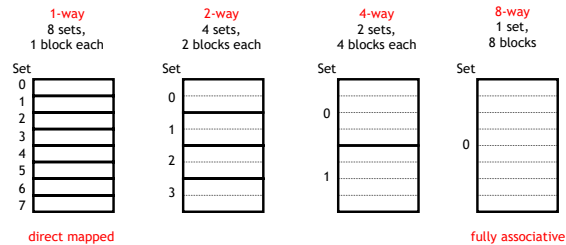
What's a cache block? (or *cache line*)



42

Set Associative Caches

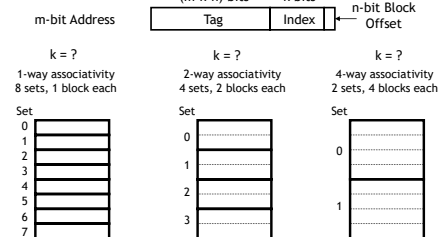
- What if we could store data in *any* place in the cache?
 - That's called a *fully associate cache*
- But that might slow down caches... so we do something in between.



43

Example placement in set-associative caches

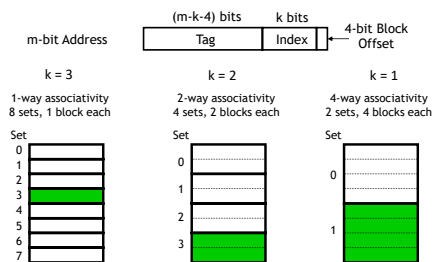
- Where would data from address 0x1833 be placed?
 - Block size is 16 bytes.
- 0x1833 in binary is 00...01 1000 0011 0011.



44

Example placement in set-associative caches

- Where would data from address 0x1833 be placed?
 - Block size is 16 bytes.
- 0x1833 in binary is 00...01 1000 0011 0011.



45

Block replacement

- Any empty block in the correct set may be used for storing data.
 - How can the CPU tell if a block is empty or not?
- What should happen if there are no empty blocks?
- Replace something, of course, but what?
- Caches typically use schemes related to *least-recently-used*
 - Guess that the future will look like the past

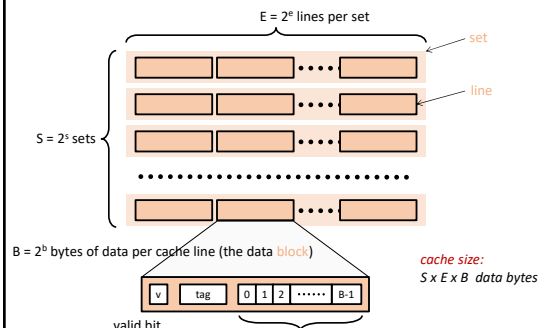
46

Memory and Caches

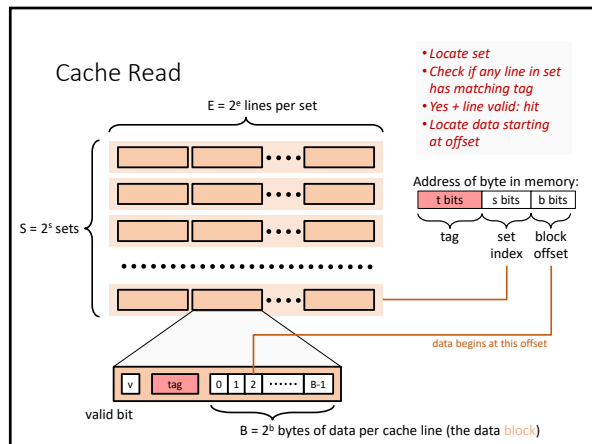
- Cache basics
- Principle of locality
- Memory hierarchies
- Cache organization (part 2)
- Program optimizations that consider caches

47

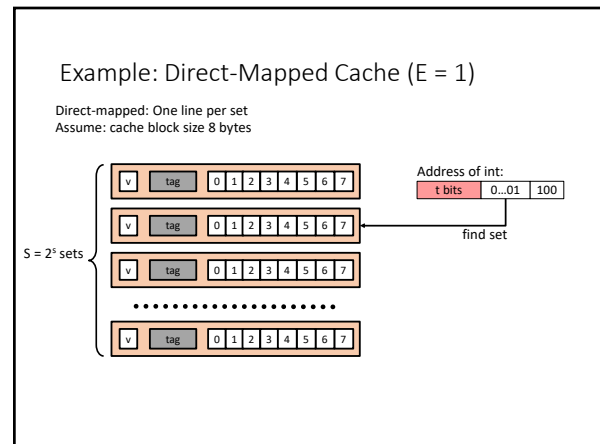
General Cache Organization (S, E, B)



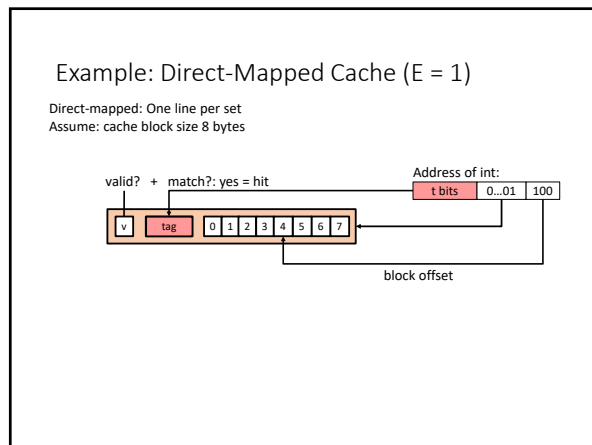
48



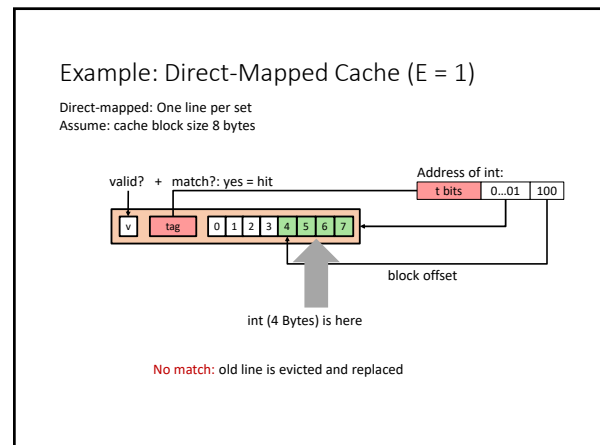
49



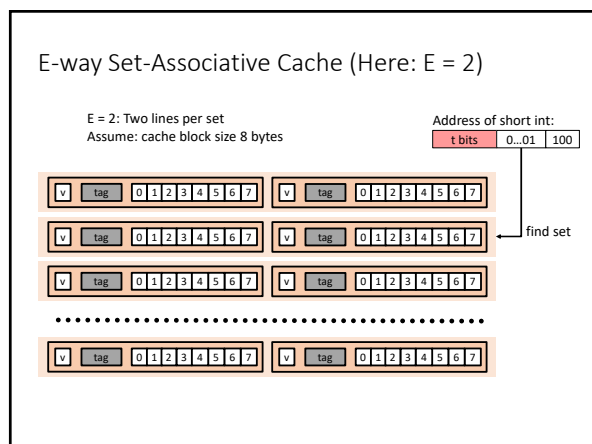
50



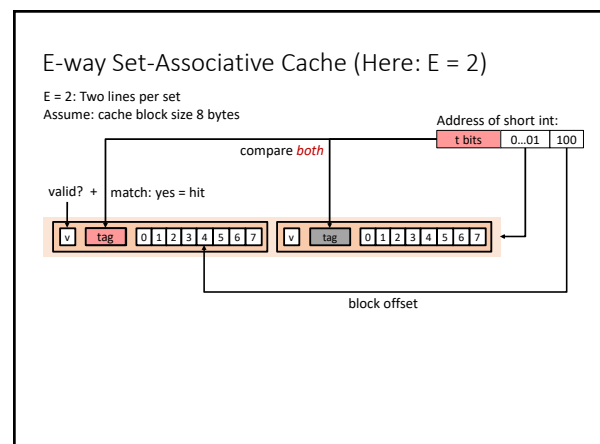
51



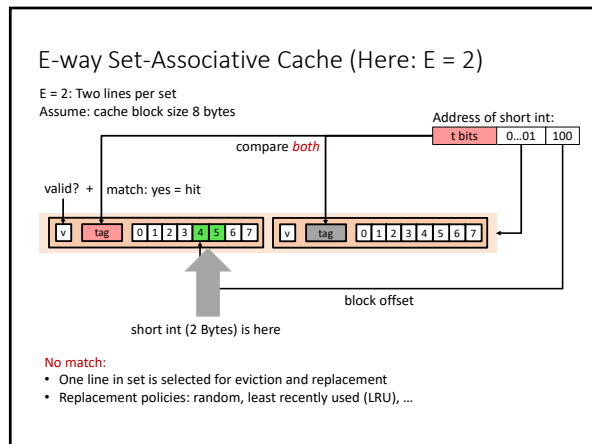
52



53



54



55

Types of Cache Misses

- **Cold (compulsory) miss**
 - Occurs on first access to a block
- **Conflict miss**
 - Most hardware caches limit blocks to a small subset (sometimes just one) of the available cache slots
 - if one (e.g., block i must be placed in slot (i mod size)), **direct-mapped**
 - if more than one, n-way **set-associative** (where n is a power of 2)
 - 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
- **Capacity miss**
 - Occurs when the set of active cache blocks (the **working set**) is larger than the cache (just won't fit)

56

What about writes?

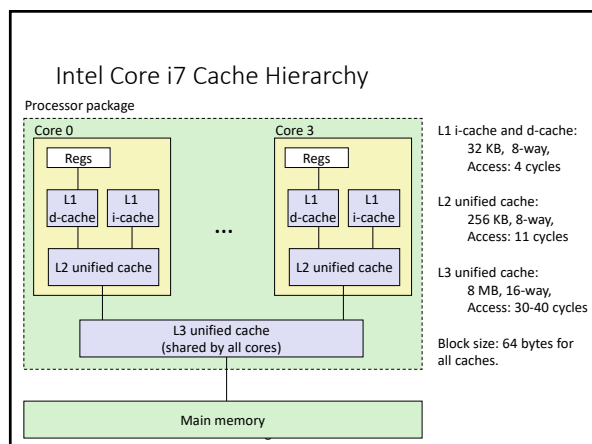
- Multiple copies of data exist:
 - L1, L2, possibly L3, main memory
- What is the main problem with that?

57

What about writes?

- Multiple copies of data exist:
 - L1, L2, possibly L3, main memory
- What to do on a write-hit?
 - Write-through** (write immediately to memory)
 - Write-back** (defer write to memory until line is evicted)
 - Need a **dirty bit** to indicate if line is different from memory or not
- What to do on a write-miss?
 - Write-allocate** (load into cache, update line in cache)
 - Good if more writes to the location follow
 - No-write-allocate** (just write immediately to memory)
- Typical caches:
 - Write-back + Write-allocate, usually
 - Write-through + No-write-allocate, occasionally

58



59

Core i7-8700K (2017)

CPU Caches Mainboard Memory SPD Graphics Bench About

L1D-Cache
Size: 32 KBytes x 6
Descriptor: 8-way set associative, 64-byte line size

L1I-Cache
Size: 32 KBytes x 6
Descriptor: 8-way set associative, 64-byte line size

L2 Cache
Size: 256 KBytes x 6
Descriptor: 4-way set associative, 64-byte line size

L3 Cache
Size: 12 MBytes
Descriptor: 16-way set associative, 64-byte line size

Size
Descriptor
Speed

CPU-Z Ver. 1.80.1.x64 Tools Validate Close

60

Memory and Caches

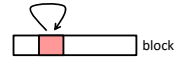
- Cache basics
- Principle of locality
- Memory hierarchies
- Cache organization
- Program optimizations that consider caches

61

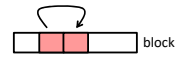
Why Caches Work

- **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



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



62

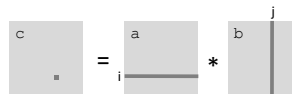
Optimizations for the Memory Hierarchy

- Write code that has good locality
 - Spatial: access data contiguously
 - Temporal: make sure access to the same data is not too far apart in time
- How to achieve?
 - Proper choice of algorithm
 - Loop transformations

63

Example: Matrix Multiplication

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

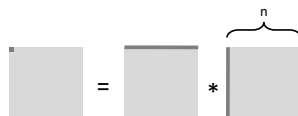


64

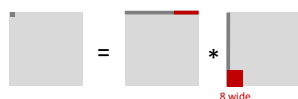
(Simplistic) Cache Miss Analysis

- Assume:
 - Matrix elements are doubles
 - Cache block = 64 bytes = 8 doubles
 - Cache size $C \ll n$ (much smaller than n)

- First iteration:
 - $n/8 + n = 9n/8$ misses (omitting matrix c)



- Afterwards **in cache:** (schematic)

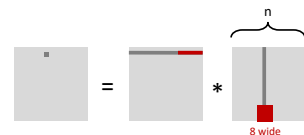


65

(Simplistic) Cache Miss Analysis

- Assume:
 - Matrix elements are doubles
 - Cache block = 64 bytes = 8 doubles
 - Cache size $C \ll n$ (much smaller than n)

- Other iterations:
 - Again: $n/8 + n = 9n/8$ misses (omitting matrix c)



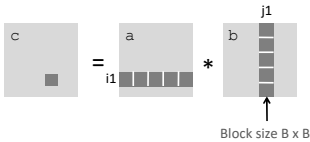
- Total misses:
 - $9n/8 * n^2 = (9/8) * n^3$

66

Blocked Matrix Multiplication

```
c = (double *) calloc(sizeof(double), n*n);

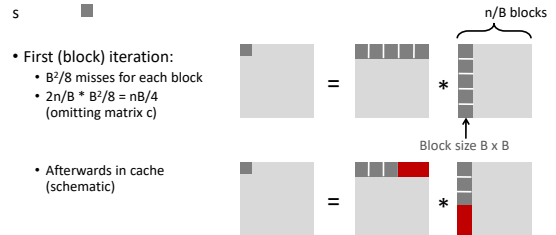
/* Multiply n x n matrices a and b */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i+=B)
        for (j = 0; j < n; j+=B)
            for (k = 0; k < n; k+=B)
                /* B x B mini matrix multiplications */
                for (il = i; il < i+B; il++)
                    for (jl = j; jl < j+B; jl++)
                        for (kl = k; kl < k+B; kl++)
                            c[il*n + jl] += a[il*n + kl]*b[kl*n + jl];
}
```



67

(Simplistic) Cache Miss Analysis

- Assume:
 - Cache block = 64 bytes = 8 doubles
 - Cache size $C \ll n$ (much smaller than n)
 - Three blocks fit into cache: $3B^2 < C$



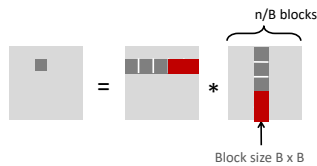
68

(Simplistic) Cache Miss Analysis

- Assume:
 - Cache block = 64 bytes = 8 doubles
 - Cache size $C \ll n$ (much smaller than n)
 - Three blocks fit into cache: $3B^2 < C$

- Other (block) iterations:
 - Same as first iteration
 - $2n/B * B^2/8 = nB/4$

- Total misses:
 - $nB/4 * (n/B)^2 = n^3/(4B)$



69

Summary

- No blocking: $(9/8) * n^3$
- Blocking: $1/(4B) * n^3$
- (No blocking)/Blocking = $9B/2$
 - If $B = 8$ ratio is $36x$
 - If $B = 16$ ratio is $72x$
- Suggests using largest possible block size B (but limit $3B^2 < C$)
- Reason for this difference:
 - Matrix multiplication has inherent temporal locality:
 - Input data: $3n^2$, computation $2n^3$
 - Every array element used $O(n)$ times!
- But program has to be written properly

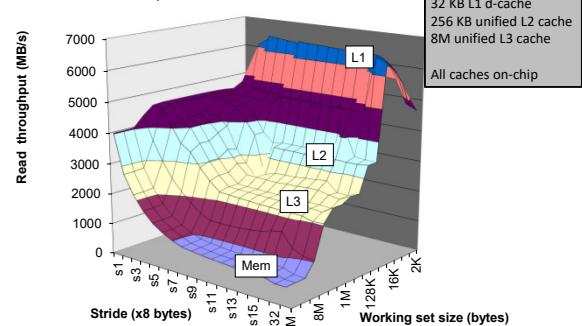
70

Cache-Friendly Code

- Programmer can optimize for cache performance
 - How data structures are organized
 - Enhance spatial locality
 - How data are accessed (e.g., nested loop structure)
 - Enhance spatial locality
 - Enhance temporal locality
- All systems favor "cache-friendly code"
 - Keep working set reasonably small (temporal locality)
 - Use small strides (spatial locality)
 - Focus on inner loop code
- Getting absolute optimum performance is very platform specific
 - Cache sizes, line sizes, associativities, etc.

71

The Memory Mountain



72