Recap of Last (Couple of) Classes

- Generally useful optimizations
  - Code motion/precomputation
  - Strength reduction
  - Common subexpression elimination and sharing
  - Removing unnecessary procedure calls
  - Loop unrolling
- Optimization blockers (and how to avoid them)
  - Procedure calls
  - Memory aliasing
- Modern processors and instruction-level parallelism
  - Accumulators and loop unrolling: techniques for eliminating sequential dependency and enhancing instruction-level parallelism
- Locality of reference

1

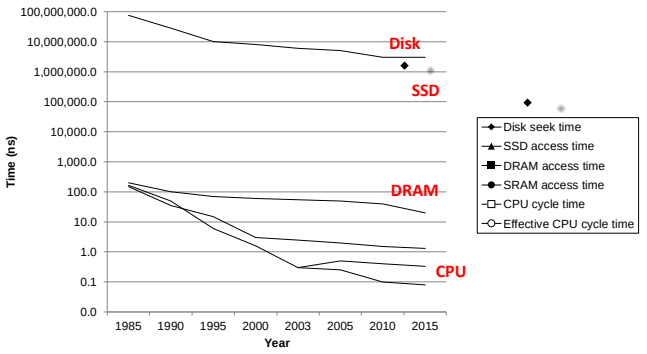
The Memory Hierarchy

- Topics
  - Locality of reference
  - Caching in the memory hierarchy
  - Storage technologies and trends

2

The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.



5

Locality

- Principle of Locality:
  - Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves.
  - Temporal locality: Recently referenced items are likely to be referenced in the near future.
  - Spatial locality: Items with nearby addresses tend to be referenced close together in time.

Locality Example:

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

- **Data**
  - Reference array elements in succession (stride-1 reference pattern): **Spatial locality**
  - Reference `sum` each iteration: **Temporal locality**
- **Instructions**
  - Reference instructions in sequence: **Spatial locality**
  - Cycle through loop repeatedly: **Temporal locality**

6

Locality Example

- Claim: Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.
- Question: Does this function have good locality?

```
int sumarrayrows(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
}
```

7

Locality Example

- Question: Does this function have good locality?

```
int sumarraycols(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
}
```

8

Loop Interchange

- Question: Can you permute the loops so that the function scans the 3-d array `a[]` with a stride-1 reference pattern (and thus has good spatial locality)?

```
int sumarray3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

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

9

Loop Fusion

- Can you improve the temporal locality of this program

```
for (i = 0; i < 100; i++)
    A[i] = C[i];
for (i = 0; i < 100; i++)
    B[i] = C[i]+1;
```

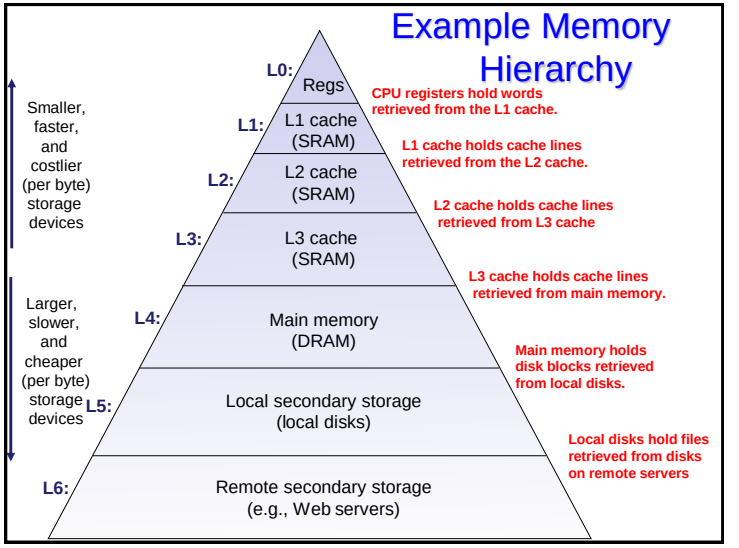
10

Memory Hierarchies

- Some fundamental and enduring properties of hardware and software:
  - Fast storage technologies cost more per byte and have less capacity.
  - The gap between CPU and main memory speed is widening.
  - Well-written programs tend to exhibit good locality.
- These fundamental properties complement each other beautifully.
- They suggest an approach for organizing memory and storage systems known as a memory hierarchy.

11

Example Memory Hierarchy



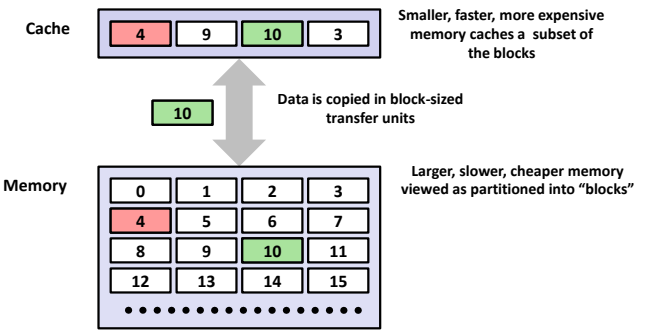
12

Caches

- *Cache:* A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.
- Fundamental idea of a memory hierarchy:
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- 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.

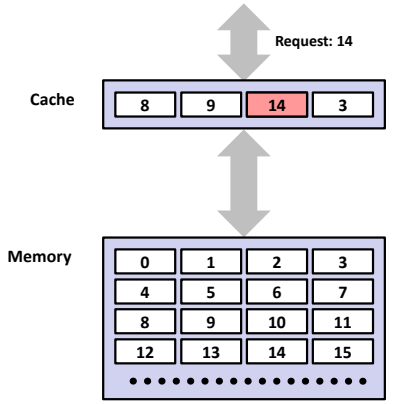
13

General Cache Concepts



14

General Cache Concepts: Hit

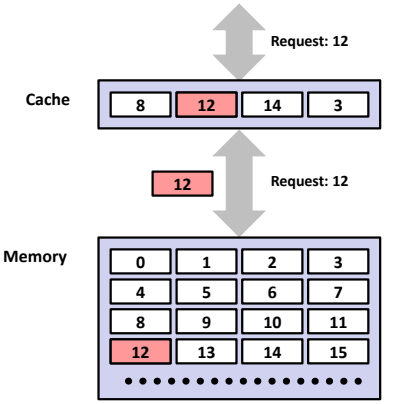


Data in block b is needed

Block b is in cache:
Hit!

15

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)

16

General Caching Concepts: Types of Cache Misses

- Cold (compulsory) miss
  - Cold misses occur because the cache is empty
- Conflict miss
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k
    - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block
    - E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.
- Capacity miss
  - Occurs when the set of active cache blocks (working set) is larger than the cache.

17

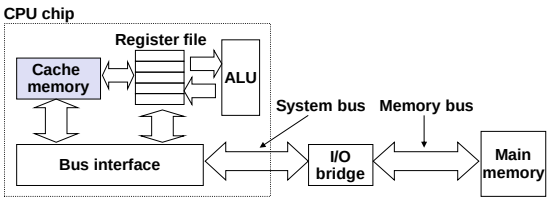
Examples of Caching in the Mem. Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware MMU
L1 cache	64-byte blocks	On-Chip L1	4	Hardware
L2 cache	64-byte blocks	On-Chip L2	10	Hardware
Virtual Memory	4-KB pages	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

18

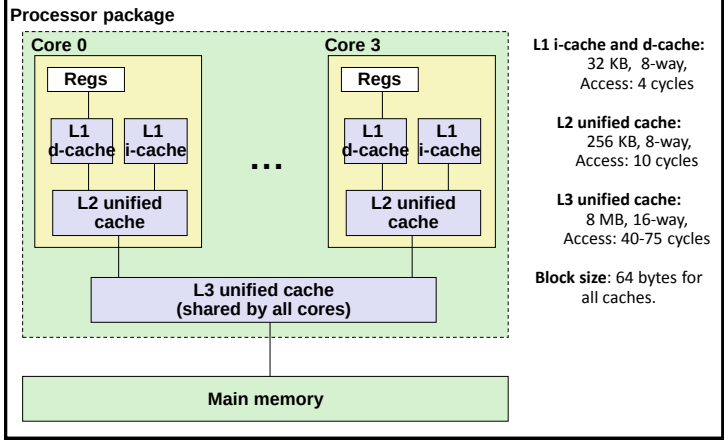
Cache Memories

- Cache memories are small, fast SRAM-based memories managed automatically in hardware
  - Hold frequently accessed blocks of main memory
- CPU looks first for data in cache
- Typical system structure:



19

Intel Core i7 Cache Hierarchy



20

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 quite 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
  - Typical numbers:
    - 4 clock cycle for L1
    - 10 clock cycles for L2
- Miss Penalty
  - Additional time required because of a miss
    - typically 50-200 cycles for main memory (Trend: increasing!)

21

Let's think about those numbers

- 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

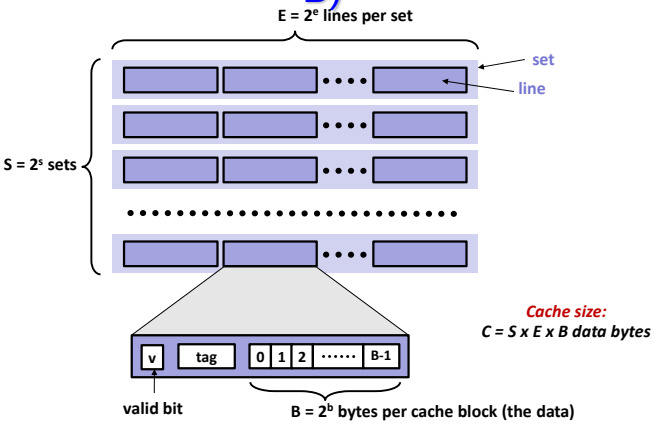
Writing Cache Friendly Code

- Make the common case go fast
  - Focus on the inner loops of the core functions
- Minimize the misses in the inner loops
  - Repeated references to variables are good (temporal locality)
  - Stride-1 reference patterns are good (spatial locality)

Key idea: Our qualitative notion of locality is quantified through our understanding of cache memories

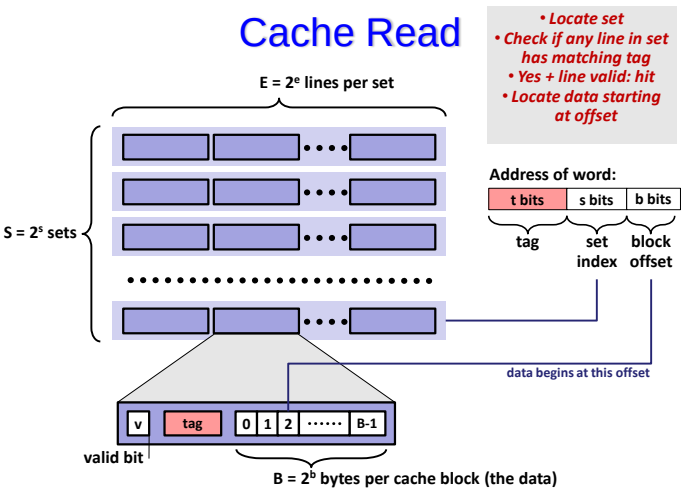
23

General Cache Organization (S, E, B)



24

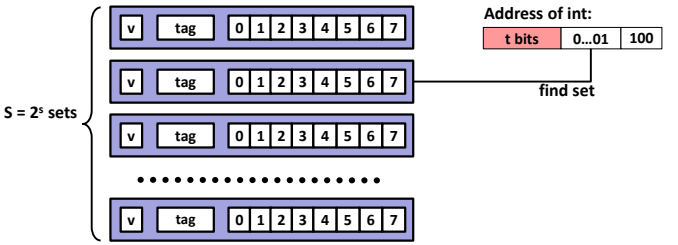
Cache Read



25

Example: Direct Mapped Cache (E = 1)

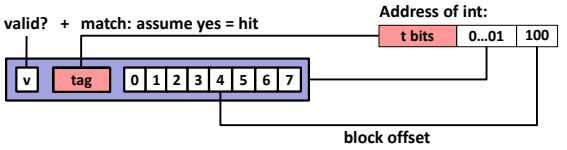
Direct mapped: One line per set
Assume: cache block size 8 bytes



26

Example: Direct Mapped Cache (E = 1)

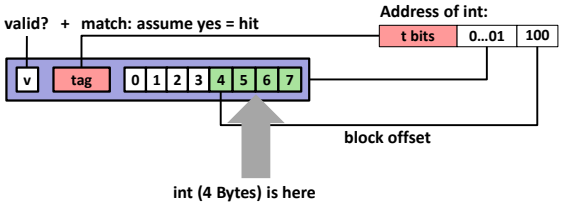
Direct mapped: One line per set
Assume: cache block size 8 bytes



27

Example: Direct Mapped Cache (E = 1)

Direct mapped: One line per set
Assume: cache block size 8 bytes



If tag doesn't match: old line is evicted and replaced

28

Direct-Mapped Cache Simulation

t=1	s=2	b=1
x	xx	x

M=16 bytes (4-bit addresses), B=2 bytes/block,
S=4 sets, E=1 Blocks/set

Address trace (reads, one byte per read):

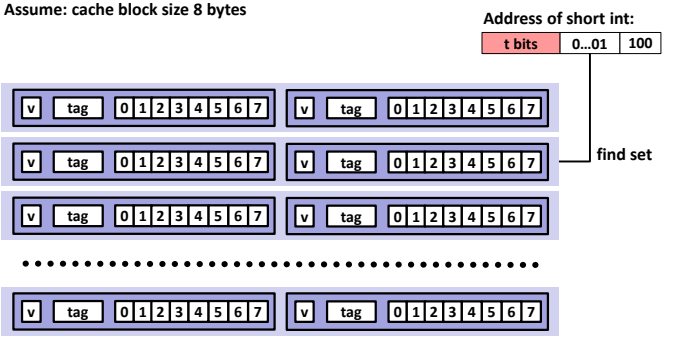
0	**[0000$_2$],**	miss
1	**[0001$_2$],**	hit
7	**[0111$_2$],**	miss
8	**[1000$_2$],**	miss
0	**[0000$_2$]**	miss

	v	Tag	Block
Set 0	1	0	M[0-1]
Set 1			
Set 2			
Set 3	1	0	M[6-7]

29

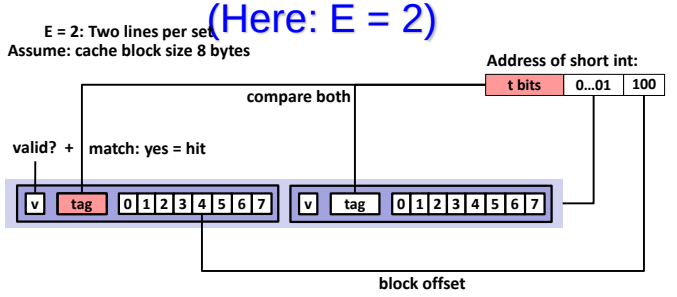
E-way Set Associative Cache (Here: E = 2)

E = 2: Two lines per set
Assume: cache block size 8 bytes



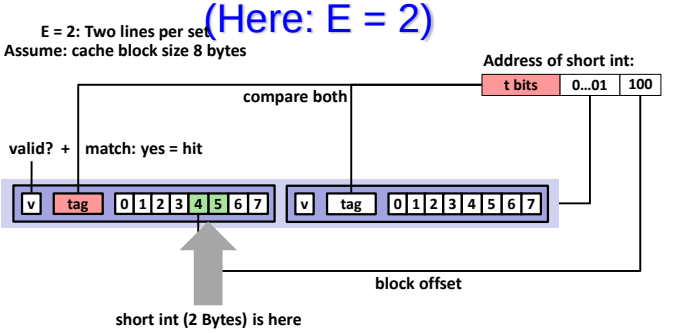
30

E-way Set Associative Cache (Here: E = 2)



31

E-way Set Associative Cache (Here: E = 2)



No match:

- **One line in set is selected for eviction and replacement**
- **Replacement policies: random, least recently used (LRU), …**

32

2-Way Set Associative Cache Simulation

t=2	s=1	b=1
xx	x	x

M=16 byte addresses, B=2 bytes/block,
S=2 sets, E=2 blocks/set

Address trace (reads, one byte per read):

0	**[0000$_2$],**	miss
1	**[0001$_2$],**	hit
7	**[0111$_2$],**	miss
8	**[1000$_2$],**	miss
0	**[0000$_2$]**	hit

	v	Tag	Block
Set 0	1	00	M[0-1]
	1	10	M[8-9]
Set 1	1	01	M[6-7]
	0		

33

What about writes?

- Multiple copies of data exist:
  - L1, L2, L3, Main Memory, Disk
- What to do on a write-hit?
  - Write-through (write immediately to memory)
  - Write-back (defer write to memory until replacement of line)
    - Need a dirty bit (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 writes to the location follow
  - No-write-allocate (writes straight to memory, does not load into cache)
- Typical
  - Write-through + No-write-allocate
  - **Write-back + Write-allocate**

34

Breakout

- Assume a direct-mapped cache of size 8 KByte 64-Byte line. A is located at 0x8000 and C is located at 0x9000. How many misses would you incur? Would this change with a 2-way set associative cache? Assume *i* is retained in a register and the arrays are not initially in the cache.

```
int A[100], C[100];
for (i = 0; i < 100; i++)
    A[i] = C[i];
```

35

Today

- Cache organization and operation
- Performance impact of caches
  - The memory mountain
  - Rearranging loops to improve spatial locality
  - Using blocking to improve temporal locality

36