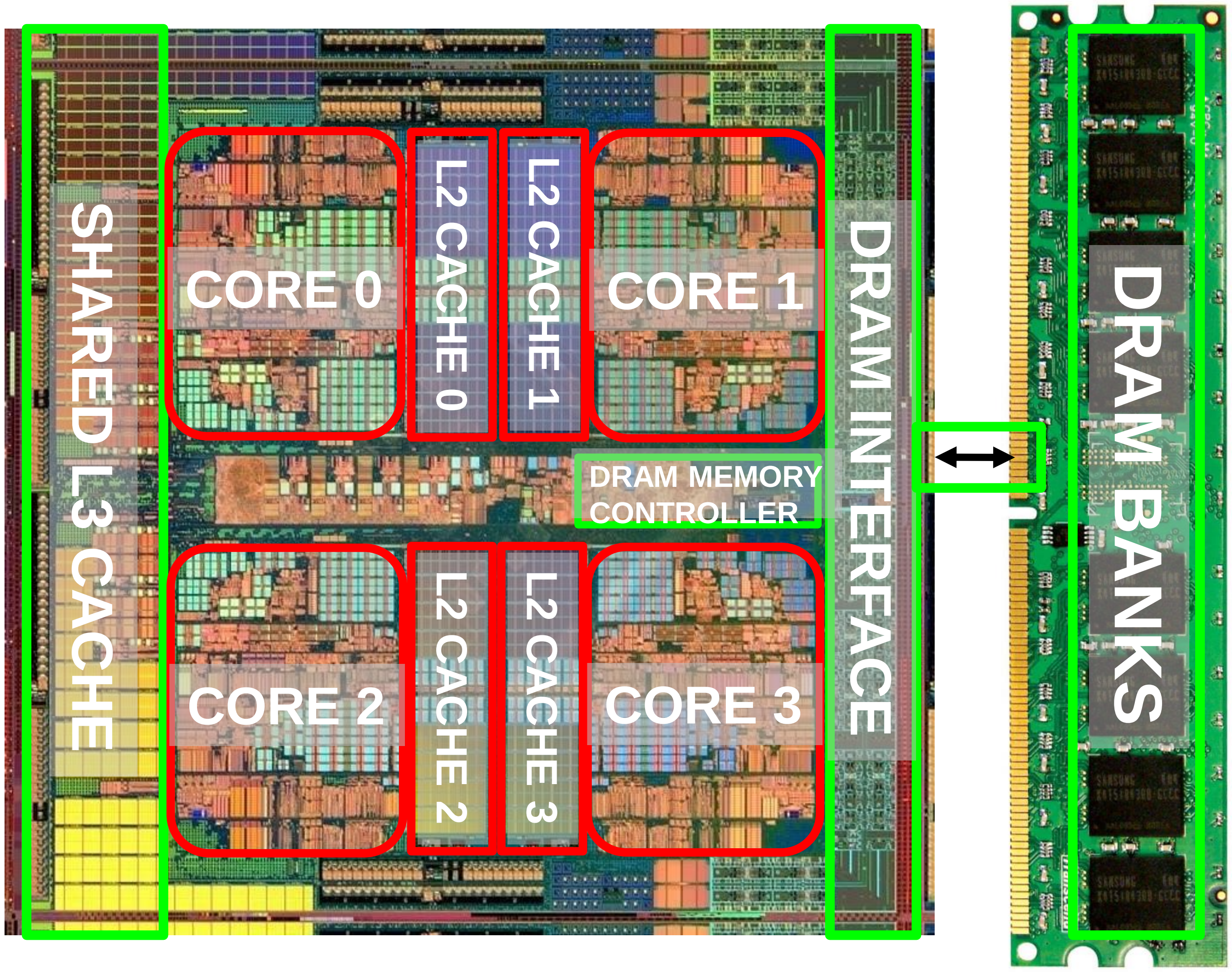
Principles of designing memory hierarchy and Caches

Slides adapted by: Dr Sparsh Mittal

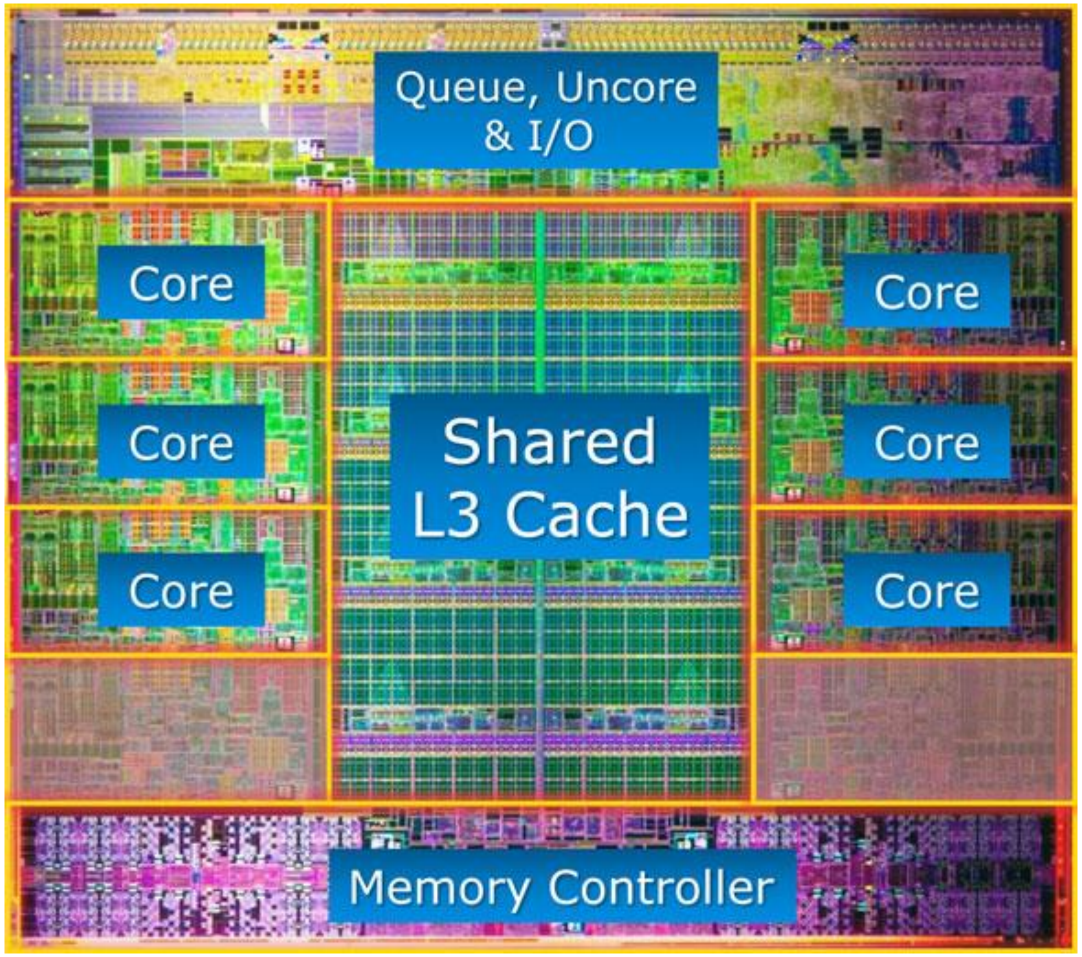
Courtesy: MK Publishers, Dr Sarangi and others

Examples of processor chip



L1, L2, L3 cache => L1 is closest and L3 is farthest from core

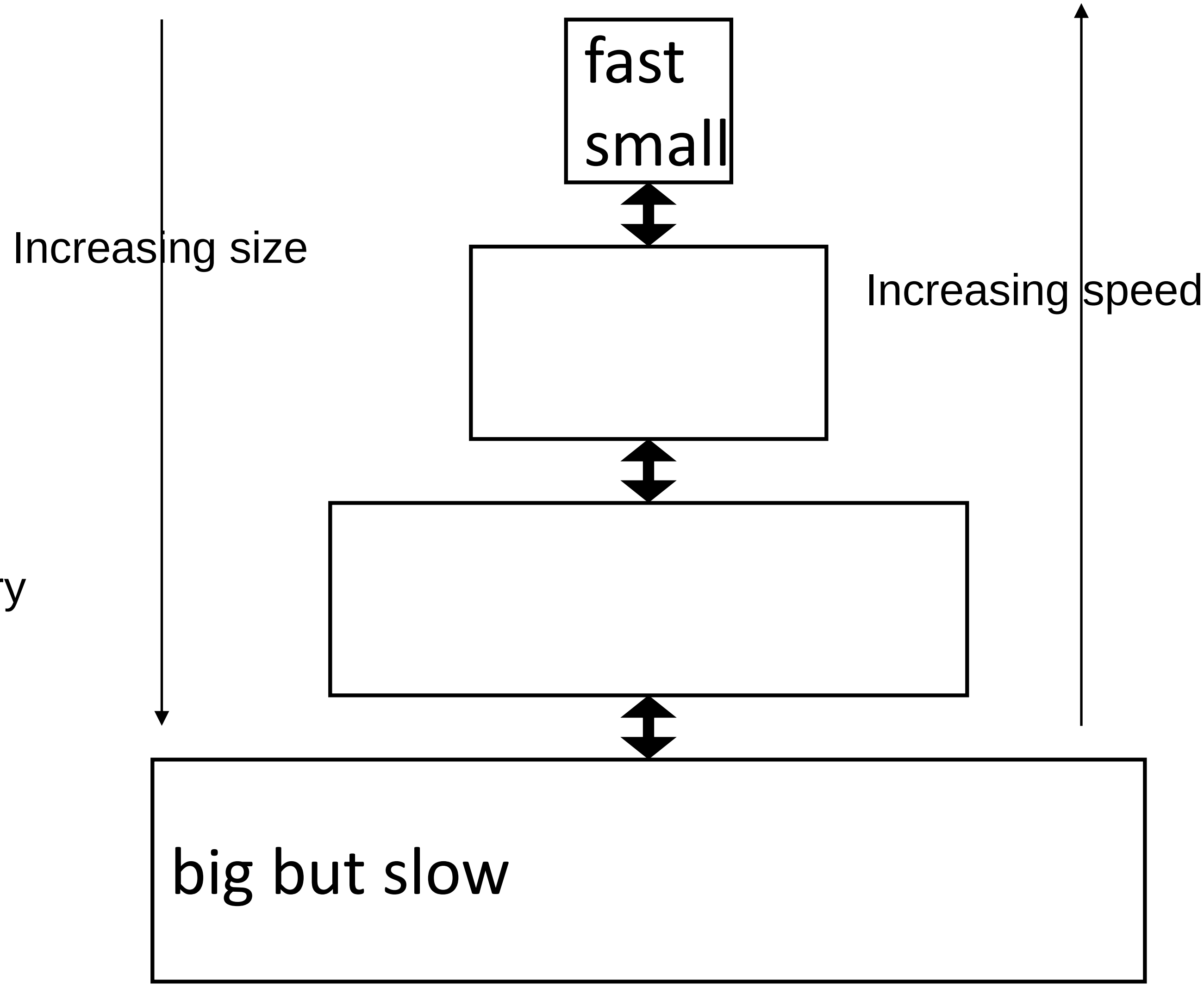
Examples of processor chip



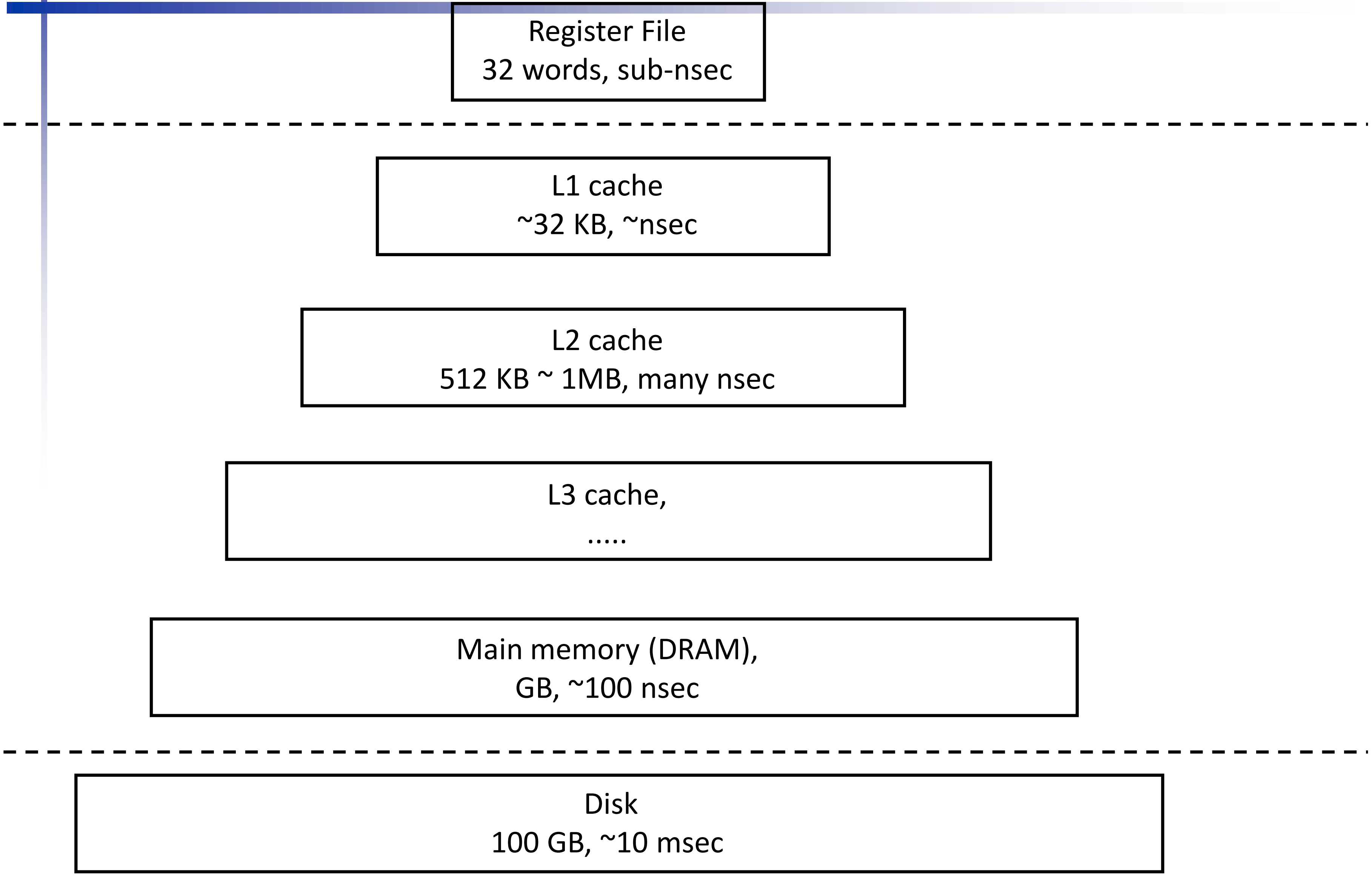
Intel's 32nm Sandy Bridge Core i7-3960X
15MB L3 cache

Why memory "hierarchy"?

Personal bag	L1 cache
Book-shelf	L2 cache
Institute library	Main memory
Inter-institute collection	Hard disk



Example of Memory Hierarchy



Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

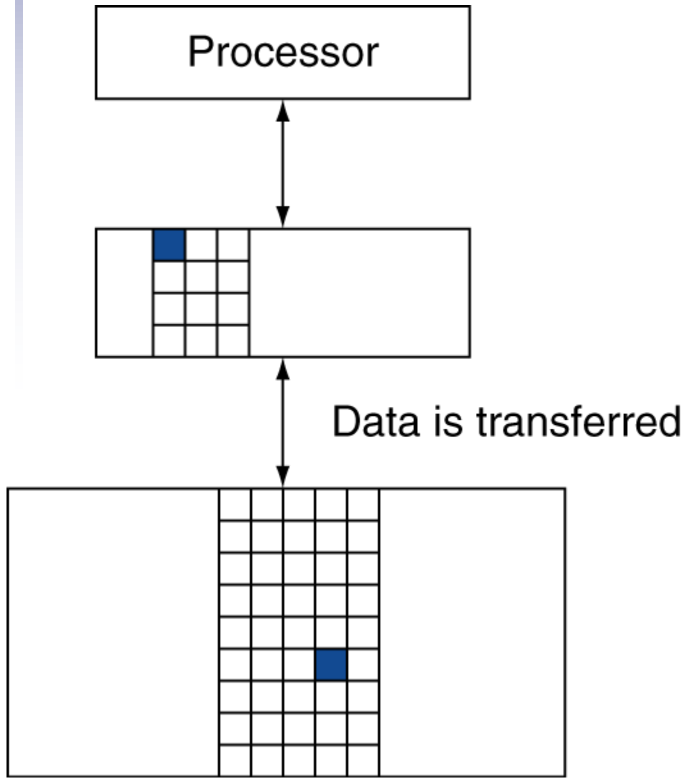
Question on spatial locality

- We have multiple numbers which can be stored in array or dynamic linked-list. If we access these numbers sequentially, which of the two data-structures gives higher spatial locality in memory?
- Answer: Array.

Memory Technology

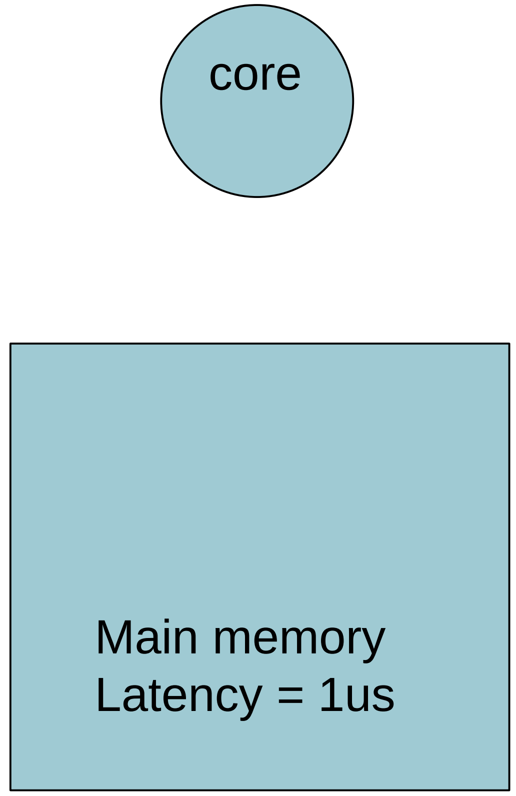
- Static RAM (SRAM)
  - 0.5ns – 2.5ns
- Dynamic RAM (DRAM)
  - 50ns – 70ns
- Magnetic disk
  - 5ms – 20ms
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

Memory Hierarchy Levels



- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      = 1 – hit ratio
  - Then accessed data supplied from upper level

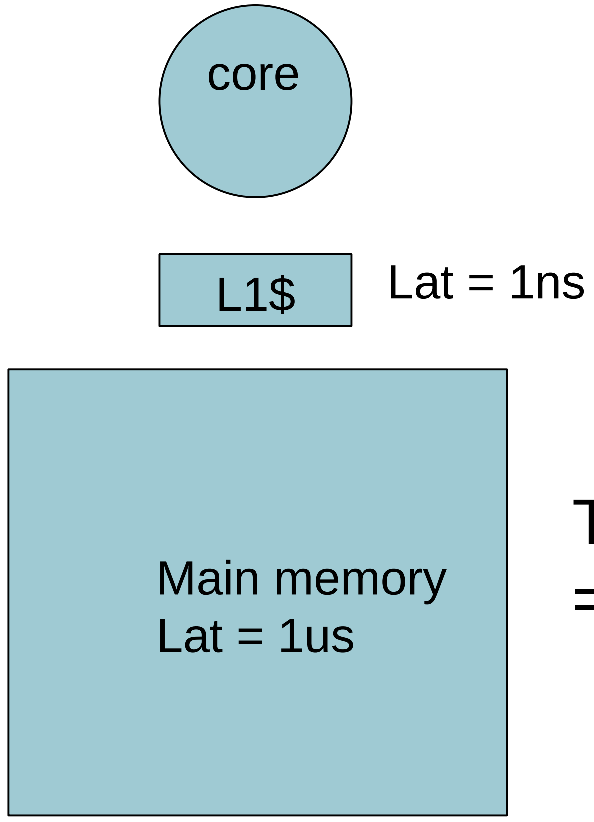
Why memory "hierarchy"?



Assume 1 million requests

Total time with no cache
= 1 million * 1us = 1 sec

Why memory hierarchy



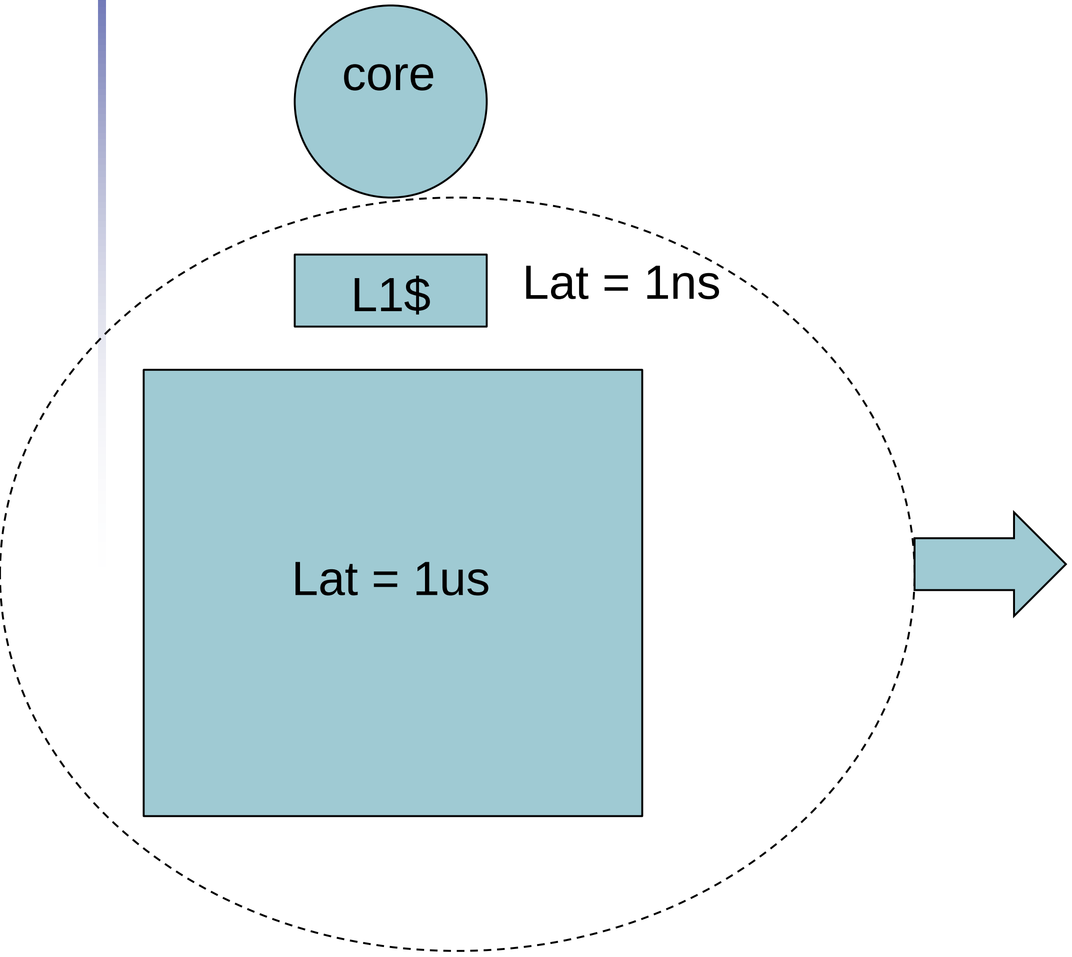
Assume 1 million requests

Total time with cache
= (HitRate*1ns + MissRate * 1us) * 10^6

With increasing hit-rate, total access time approaches that of L1$

Cache Hit/miss = element found or not found in cache

Why memory hierarchy



It appears to be a memory
- As fast as L1 cache
- As large as main memory

Approaches the characteristic of Ideal memory

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Multilevel Caches

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some systems include L-3 and L-4 cache

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = **9**

Example (cont.)

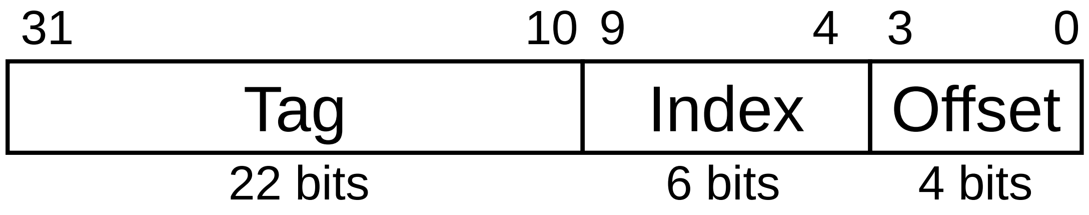
- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = **3.4**
- Performance ratio = 9/3.4 = 2.6

Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 block size smaller than L-2 block size

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - We only need the high-order bits called tag



- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache organization: associativity

An example: bookshelf

- Assume I have a bookshelf with N spaces
- I can organize it differently
- Organization 1: Keep a book anywhere

- Plus: ease of arranging
- Minus: more time in retrieving (searching)

- Organization 2: Make two partitions
- Partition 1: research books
- Parition 2: teaching books

- Plus: time required for searching reduced
- Minus: >N/2 research books cannot fit

- Organization 3: Make N partitions
- A book can go to only one space, belonging to its topic.

- Plus: time required for searching constant
- Minus: >1 book on any topic cannot fit

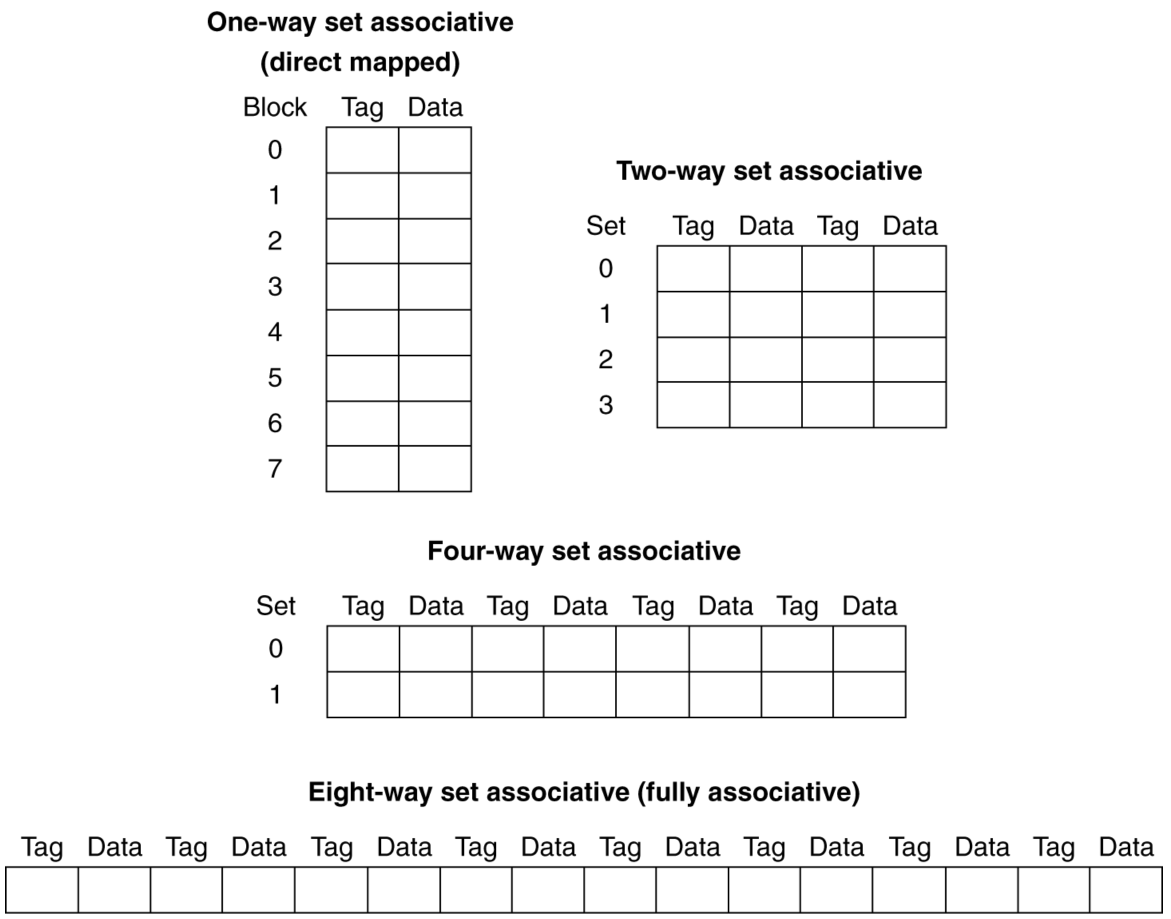
Spectrum of Associativity

- Direct-mapped (1 way, k-sets)
- Set-associative (m way, k/m sets)
- Fully-associative (1 set, k-way)

# of ways = associativity

Spectrum of Associativity

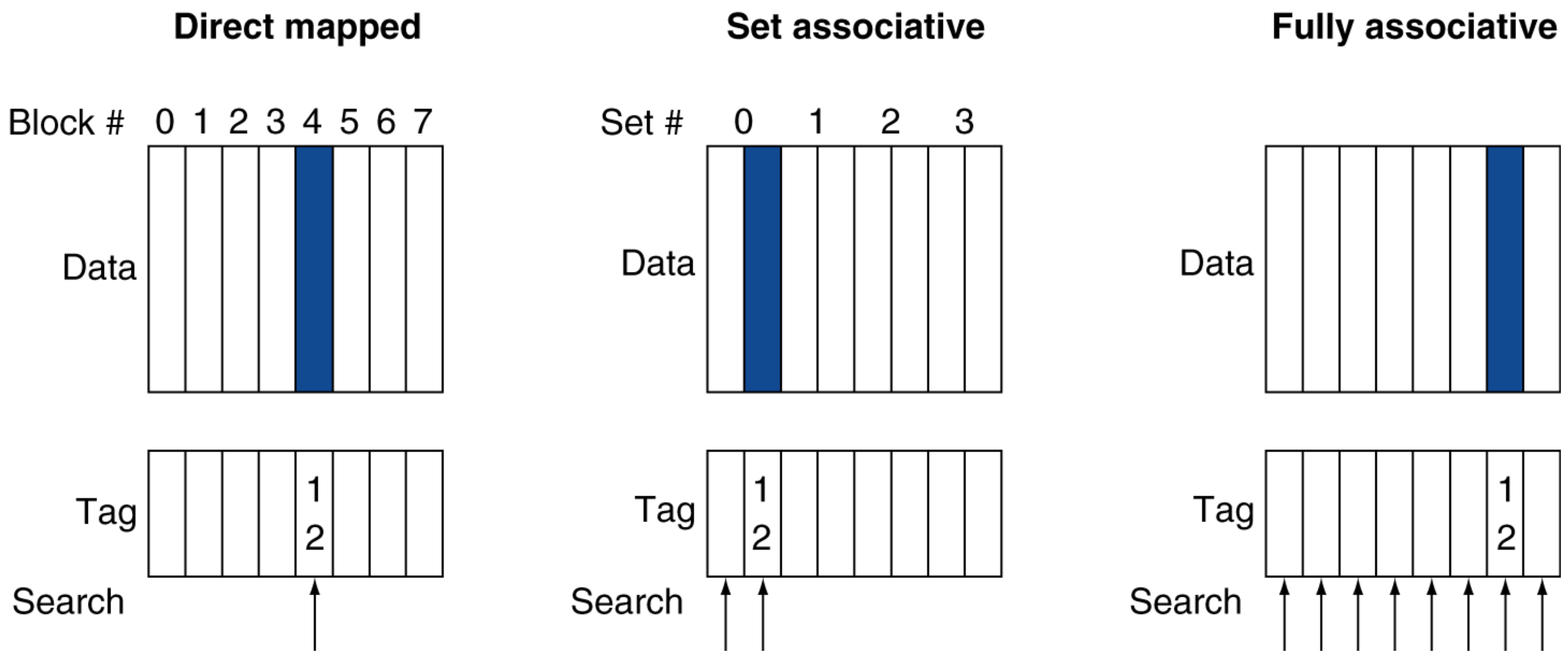
- For a cache with 8 entries



One-way set associative (direct mapped)
Block
Tag
Data
0
1
2
3
4
5
6
7
Two-way set associative
Set
Tag Data Tag Data
0
1
2
3
Four-way set associative
Set
Tag Data Tag Data Tag Data Tag Data
0
1
Eight-way set associative (fully associative)
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

# of ways = associativity

Associative Cache Example



Cache organization

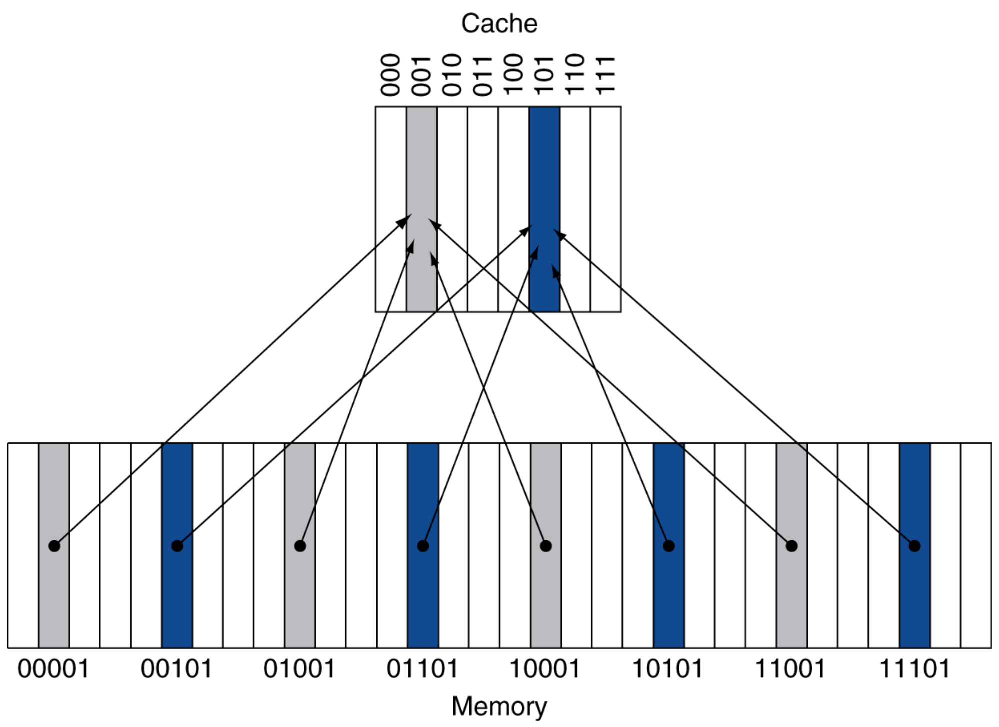
- Cache size
- = (# of blocks) * (block size)
- = (# of sets) * (# of ways) * (block size)

- Example: block size = 64B, # of ways =16, cache size = 4MB

- => # of sets = 4096

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice

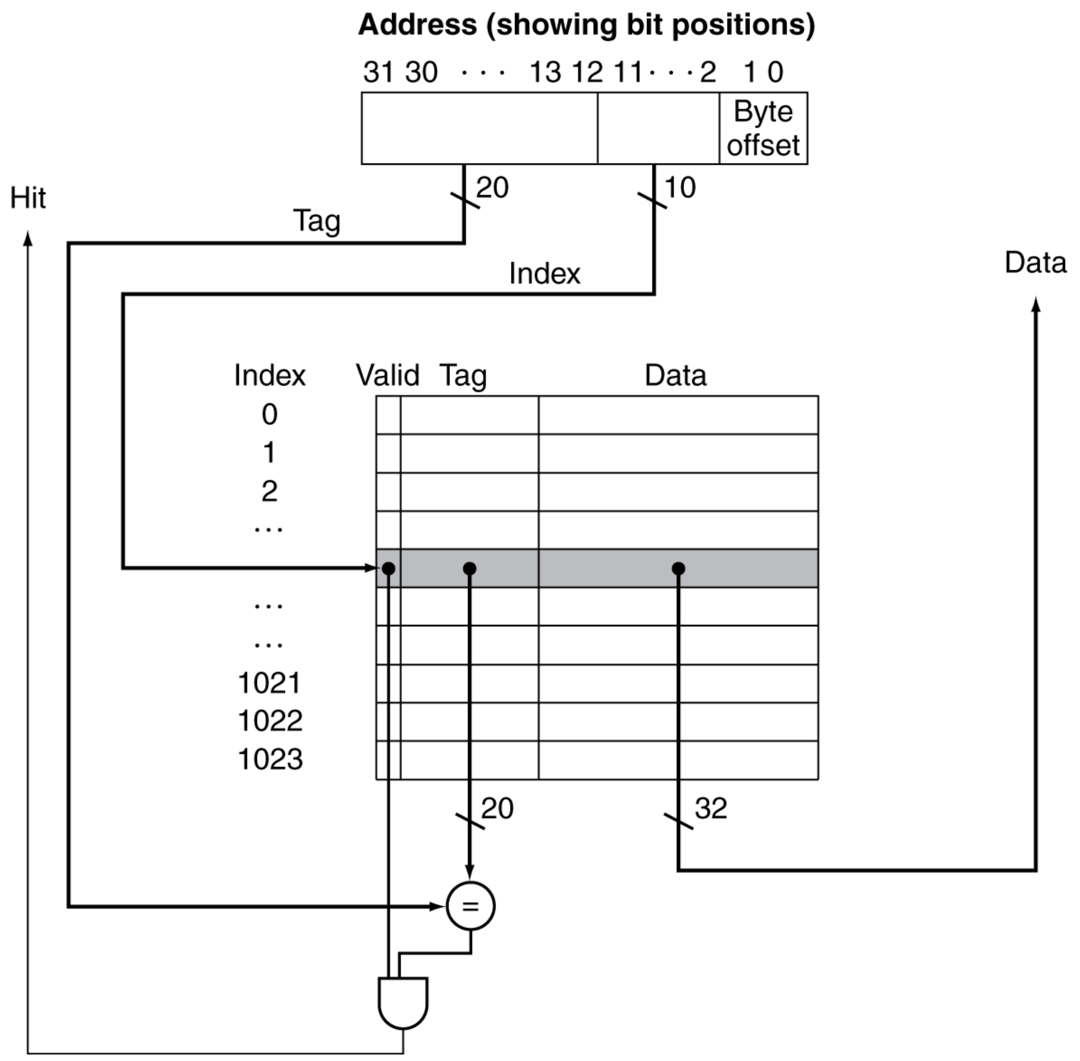


- #Blocks is a power of 2

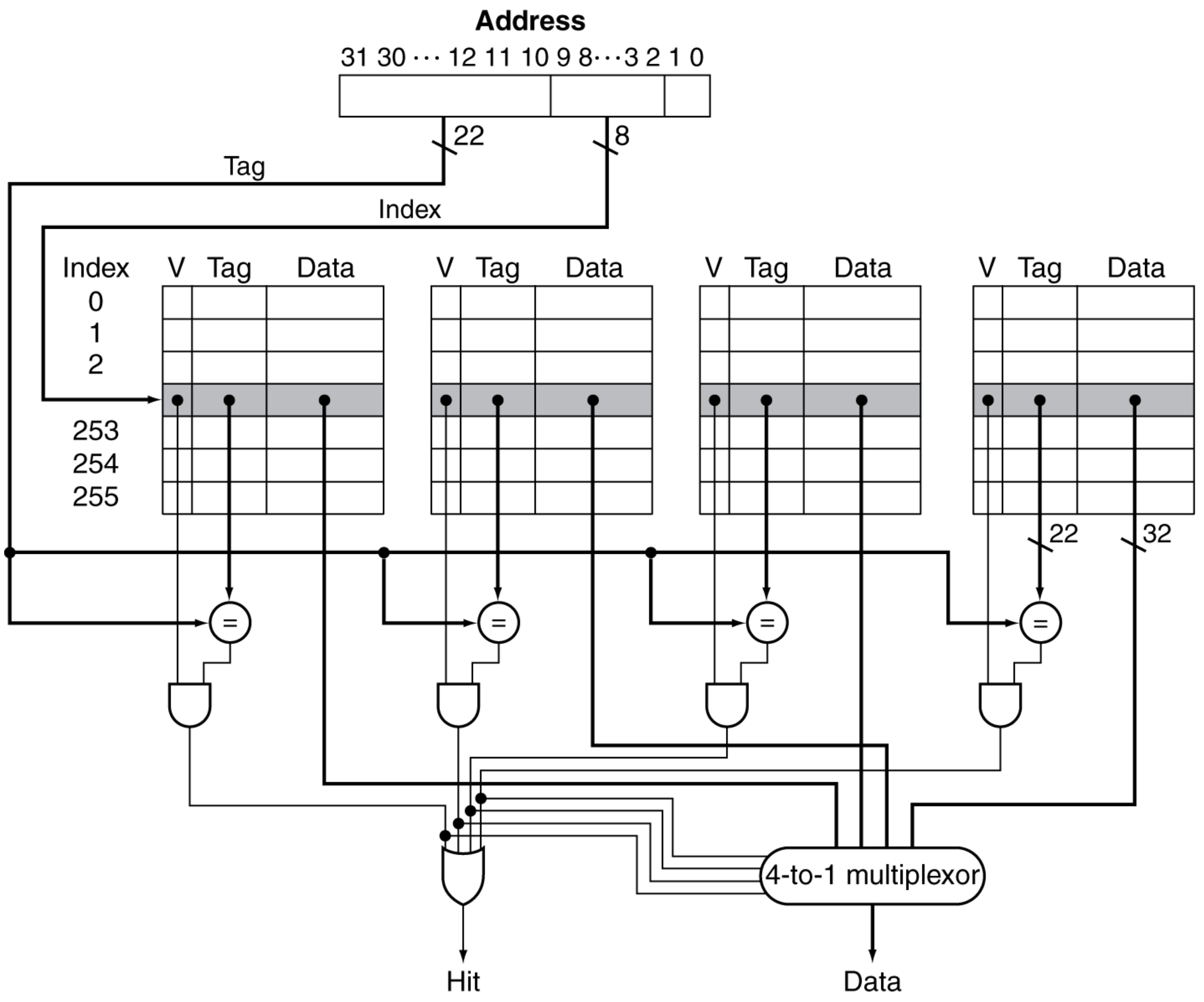
Associative Caches

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - 1 Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Direct-mapped Cache



Set Associative Cache



Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

Associativity Example

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

- Fully associative

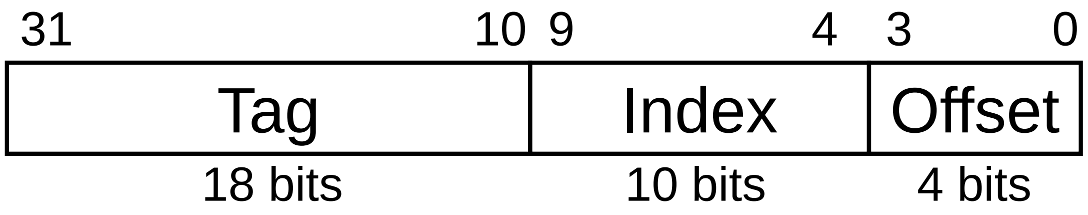
Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Example of cache parameters

- Direct-mapped, write-back, write allocate
- Block size: 4 words (16 bytes)
- Cache size: 16 KB (1024 blocks)
- 32-bit byte addresses
- Valid bit and dirty bit per block
- Blocking cache
  - CPU waits until access is complete



Basic Cache Operations

- lookup → Check if the memory location is present
- data read → read data from the cache
- data write → write data to the cache
- insert → insert a block into a cache
- replace → find a candidate for replacement
- evict → throw a block out of the cache

Definitions

local miss rate	It is equal to the number of misses in a cache at level *i* divided by the total number of accesses at level *i*.
global miss rate	It is equal to the number of misses in a cache at level *i* divided by the total number of memory accesses.
working set	The amount of memory, a given program requires in a time interval.

Types of Misses

- Compulsory Misses
  - Misses that happen when we read in a piece of data for the first time.
- Conflict Misses
  - Misses that occur due to the limited amount of associativity in a set associative or direct mapped cache. Example: Assume that 5 blocks (accessed by the program) map to the same set in a 4-way associative cache. Only 4 out of 5 can be accommodated.
- Capacity Misses
  - Misses that occur due to the limited size of a cache. Example: Assume the working set of a program is 10 KB, and the cache size is 8 KB.

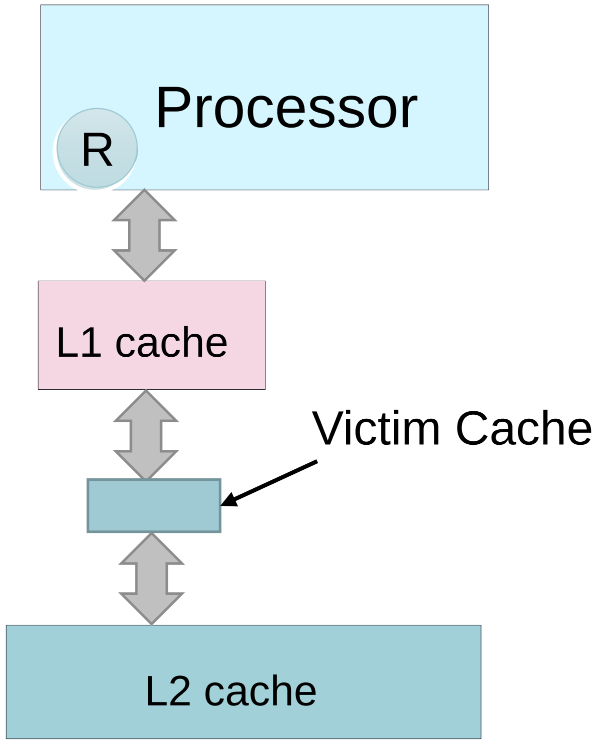
Mitigating Compulsory Misses

- Increase the block size. We can bring in more data in one go, and due to **spatial locality** the number of misses might go down.
- Try to guess the memory locations that will be accessed in the near future. **Prefetch** (fetch in advance) those locations. We can do this for example in the case of array accesses.

Mitigating Conflict Misses

- Increase the associativity of the cache (at the cost of **latency** and **power**)
- We can use a smaller fully associative cache called the ***victim cache*** . Any line that gets displaced from the main cache can be put in the victim cache. The processor needs to check both the L1 and victim cache, before proceeding to the L2 cache.
- Write programs in a **cache friendly** way.

Victim Cache



Mitigating Capacity Misses

- Increase the size of the cache

Understanding working of cache with an example

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache write hit policies

1. Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

2. Write-Back

- On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

Summary of Write Policies

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state

Cache replacement policies

Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set

- Least-recently used (LRU): Choose the one unused for the longest time
  - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity
- FIFO
- Optimal: replaces the element which will be accessed farthest in future.
  - Requires future knowledge, cannot be implemented in real-time

Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

Cache hierarchy in commercial processors

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available