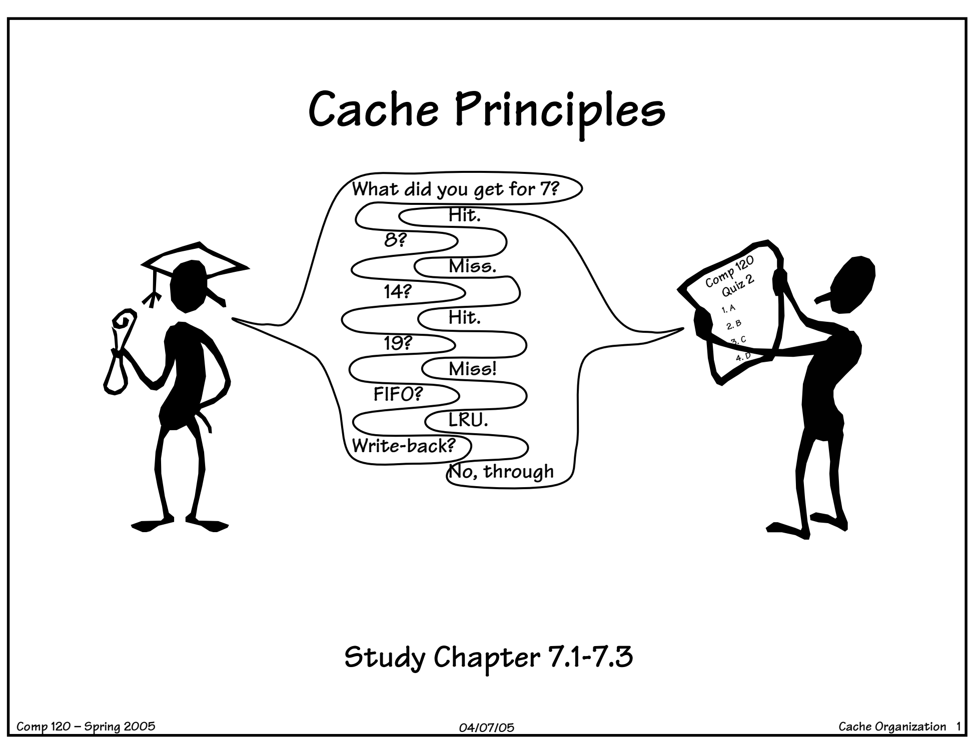
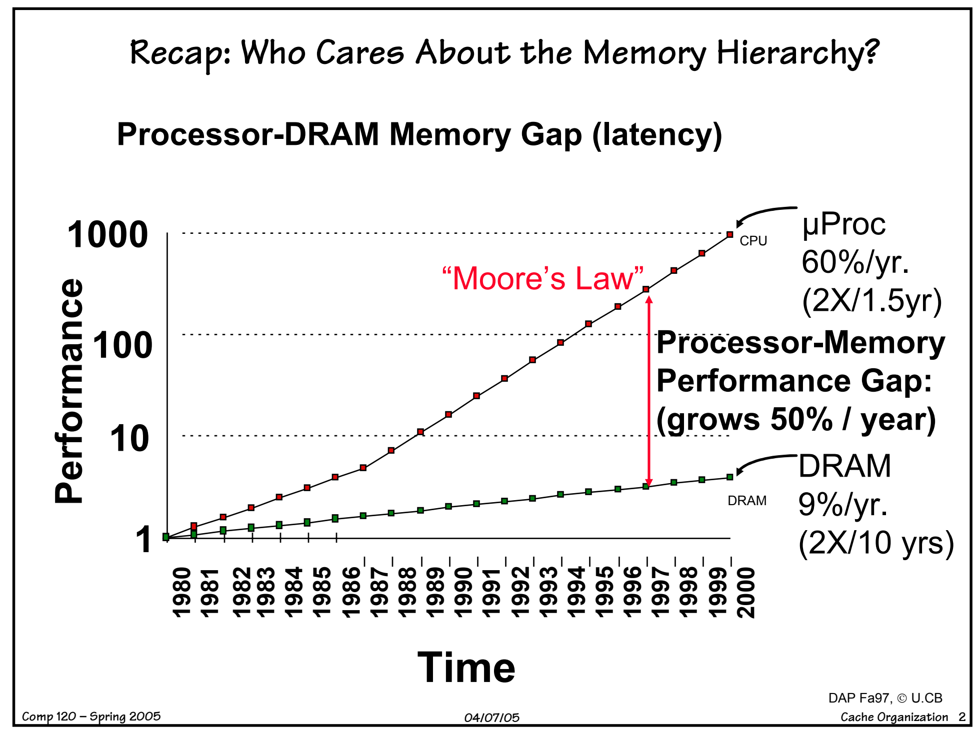
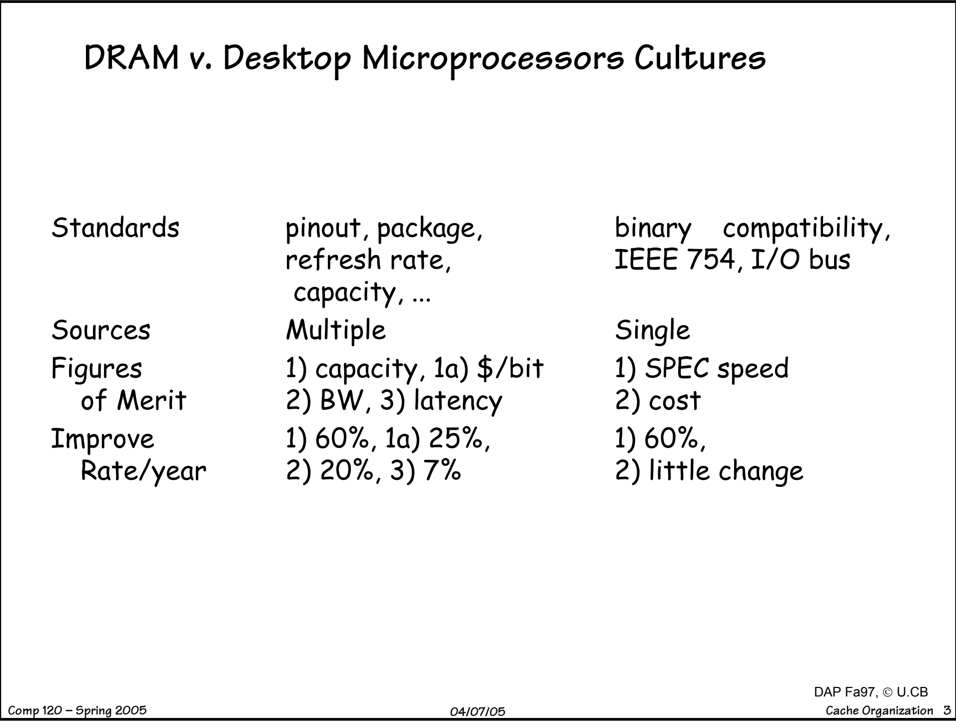
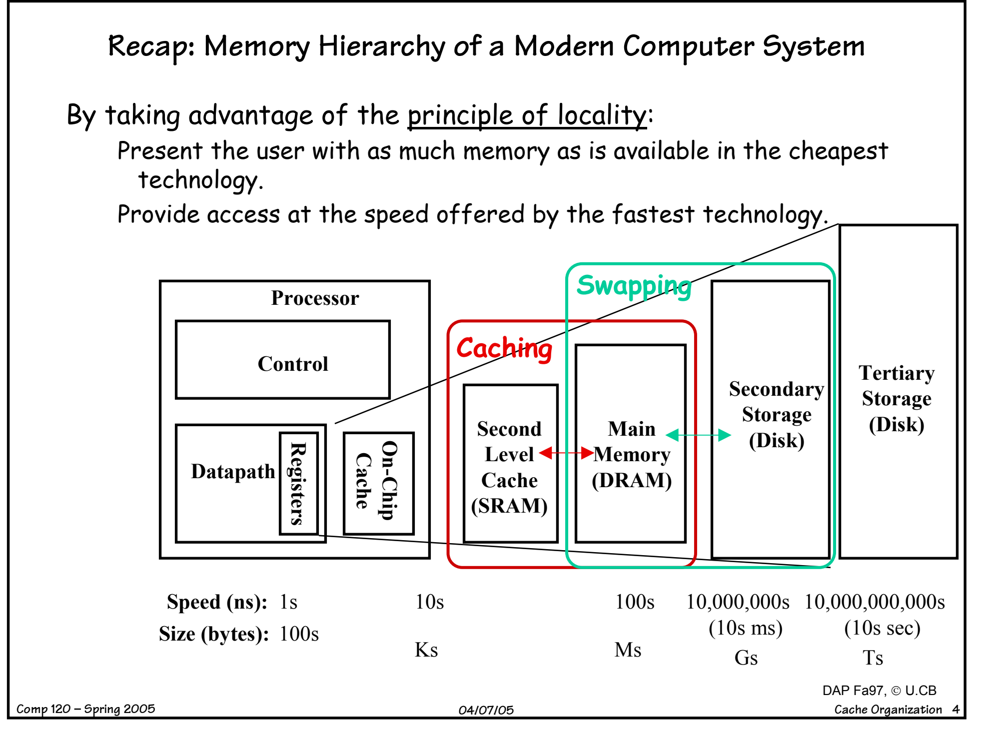

Cache Principles
What did you get for 7?
Hit.
8?
Miss.
14?
Hit.
19?
Miss!
FIFO?
LRU.
Write-back?
No, through
Comp 120
Quiz 2
Study Chapter 7.1-7.3
Comp 120 – Spring 2005
04/07/05
Cache Organization 1


Recap: Who Cares About the Memory Hierarchy?
Processor-DRAM Memory Gap (latency)
1000
100
10
1
Performance
"Moore's Law"
CPU
µProc
60%/yr.
(2X/1.5yr)
Processor-Memory
Performance Gap:
(grows 50% / year)
DRAM
DRAM
9%/yr.
(2X/10 yrs)
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
Time
DAP Fa97, © U.CB
Comp 120 – Spring 2005
04/07/05
Cache Organization 2


DRAM v. Desktop Microprocessors Cultures

Standards	pinout, package, refresh rate, capacity, ...	binary compatibility, IEEE 754, I/O bus
Sources	Multiple	Single
Figures of Merit	1) capacity, 1a) $/bit 2) BW, 3) latency	1) SPEC speed 2) cost
Improve Rate/year	1) 60%, 1a) 25%, 2) 20%, 3) 7%	1) 60%, 2) little change



Recap:

Two Different Types of Locality:

- **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
- **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

By taking advantage of the principle of locality:

- Present the user with as much memory as is available in the cheapest technology.
- Provide access at the speed offered by the fastest technology.

DRAM is slow but cheap and dense:

- Good choice for presenting the user with a BIG memory system

SRAM is fast but expensive and not very dense:

- Good choice for providing the user FAST access time.

DAP Fa97, © U.CB

Cache Review

Caches improve AVERAGE memory access time

$$t_{ave} = \alpha t_c + (1-\alpha)(t_c + t_m) = t_c + (1-\alpha)t_m$$

where
- t_c = access time of a Cache HIT
- t_m = additional access time for a Cache MISS
- α = HIT ratio (ratio of references found in cache)
- $(1-\alpha)$ = MISS ratio

e.g. 10ns+0.02x100ns=12ns (98%) (caching)

e.g. 100ns+0.000002x10ms=120ns (99.9998%) (swapping)

Cache Principle

IDEA: Suppose you're making a sequence of data requests. For each request you supply some sort of tag (or key) and the cache give you back the data associated with that tag. If there is a predictable pattern to the sequence, it's possible to build a cache that quickly responds to many (or, if we're lucky, most) requests.

A cache is usually some sort of local storage that can be accessed very quickly because of its small size and/or high-bandwidth connection, i.e., much more quickly then the usual mechanism for responding to requests.

This request is made only when tag can't be found in cache, i.e., with probability $1-\alpha$, where α is the probability that the cache has the data (a "hit")

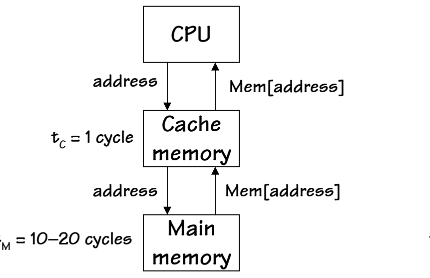
requestor —tag→ Data store t_M ; Data store —data→ requestor

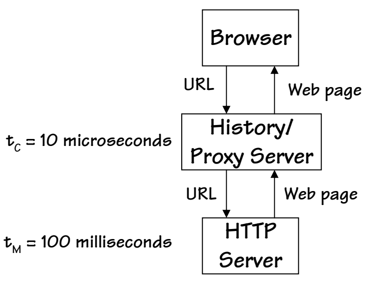
requestor —tag→ Cache $t_C \ll t_M$ —tag→ Data store t_M ; data returned back through Cache to requestor

Access time: t_M

Average access time: $t_C + (1-\alpha)t_M$

Real-world Cache Applications





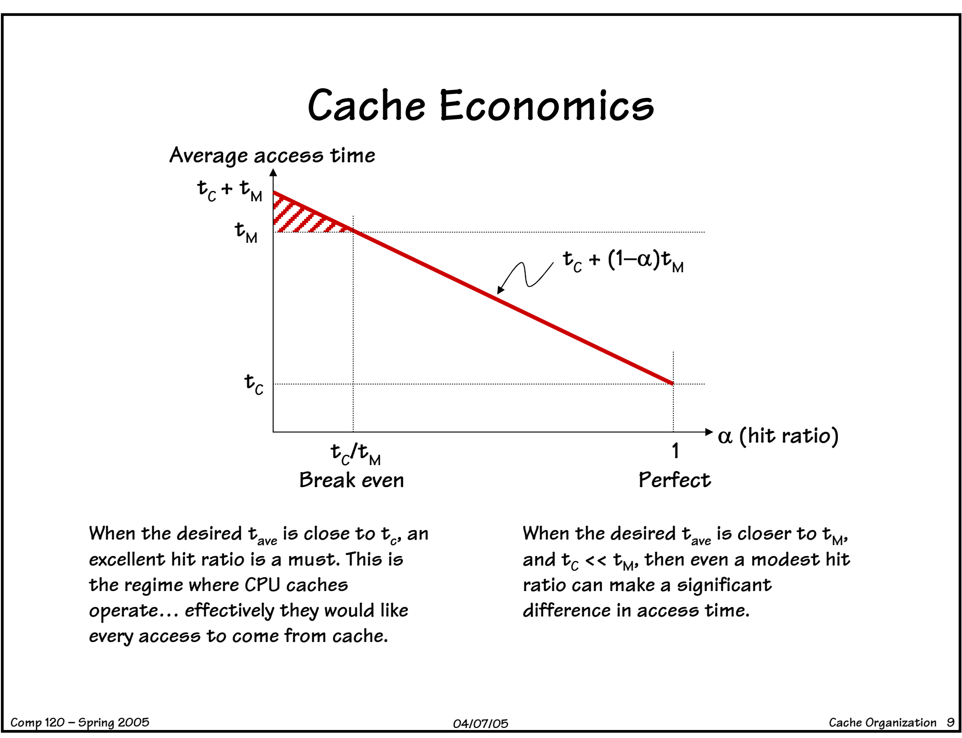
Memory system: assuming a hit ratio of 95% and t_M=20 cycles, average access time is

$t_{ave} = 1 + (1 - .95)\ 20 = 2$ cycles. Cache is worthwhile, but still not as fast as we'd like (need high hit rates).

World-Wide Web: links, back/forward navigation aid prediction. Just fair cache performance reduces effective t_M (reduces latency).

$t_{ave} = .01 + (1 - .2)\ 100 = 80$ ms.

BONUS: frees up bandwidth to server by reducing contention (win-win).



Taking Advantage of Access Patterns

Spatial locality:

Access to X suggests that locations "near" X might soon be referenced.

• Return a block of "adjacent" locations for each request,
e.g., a block of consecutive memory locations, several linked web pages or a JAR file of related java classes.
Amortize long latencies, leverage good throughput

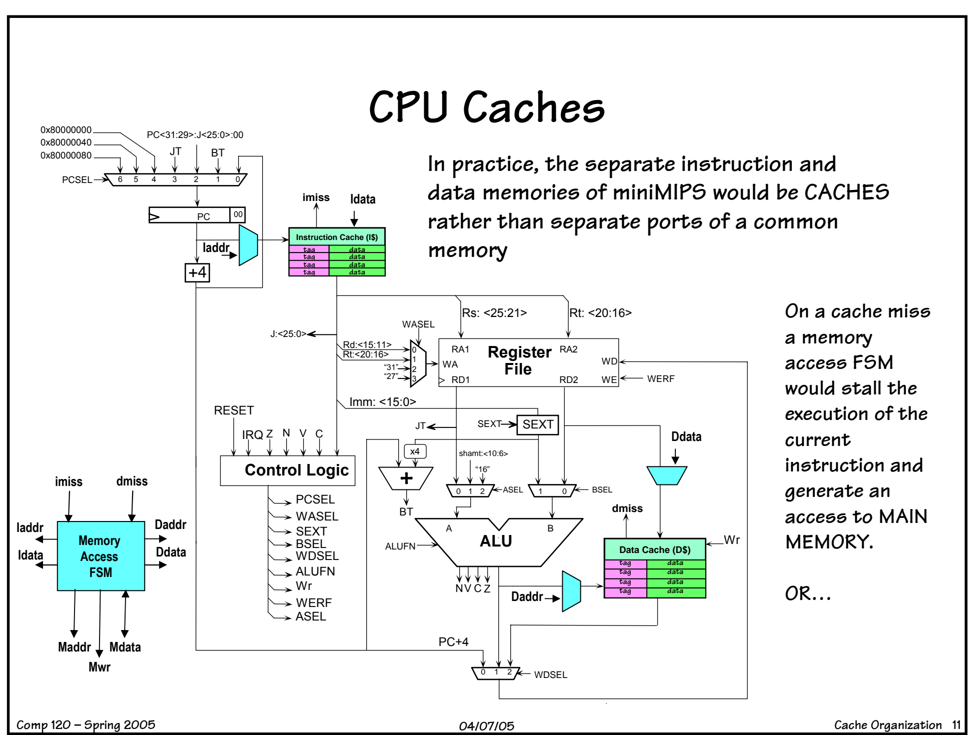
• Speculatively prefetch other nearby blocks (do this as background task so as not to preempt useful work)

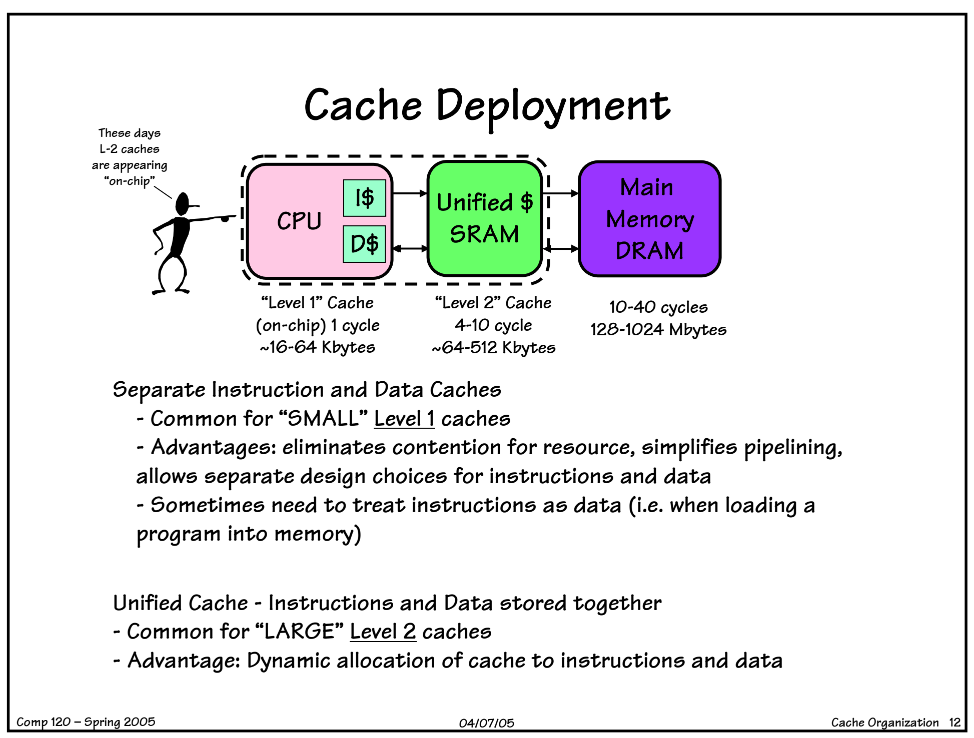
Temporal locality:

Access to X suggests subsequent accesses to X.

• Keep recent accesses in cache (Obvious, but effective).

• Impacts what we throw out of cache. According to temporal locality, the least recently used cache element is the least likely to be accessed in the future. This observation works well when the cache is large compared to the number of requests issued before revisiting a location.


CPU Caches
In practice, the separate instruction and data memories of miniMIPS would be CACHES rather than separate ports of a common memory
On a cache miss a memory access FSM would stall the execution of the current instruction and generate an access to MAIN MEMORY.
OR...
Instruction Cache (I$)
Data Cache (D$)
Register File
Control Logic
ALU
Memory Access FSM
imiss
dmiss
Iaddr
Idata
Daddr
Ddata
Maddr
Mdata
Mwr
PCSEL
WASEL
SEXT
BSEL
WDSEL
ALUFN
Wr
WERF
ASEL
RESET
Comp 120 – Spring 2005
04/07/05
Cache Organization 11


Cache Deployment
These days L-2 caches are appearing "on-chip"
CPU
I$
D$
Unified $ SRAM
Main Memory DRAM
"Level 1" Cache (on-chip) 1 cycle ~16-64 Kbytes
"Level 2" Cache 4-10 cycle ~64-512 Kbytes
10-40 cycles 128-1024 Mbytes
Separate Instruction and Data Caches
- Common for "SMALL" Level 1 caches
- Advantages: eliminates contention for resource, simplifies pipelining, allows separate design choices for instructions and data
- Sometimes need to treat instructions as data (i.e. when loading a program into memory)
Unified Cache - Instructions and Data stored together
- Common for "LARGE" Level 2 caches
- Advantage: Dynamic allocation of cache to instructions and data
Comp 120 – Spring 2005
04/07/05
Cache Organization 12

Level-2 Cache Reward

Does Level-2 Cache help?

$$t_{ave} = t_{c1} + \beta_1 t_{c2} + \beta_1 \beta_2 t_m$$

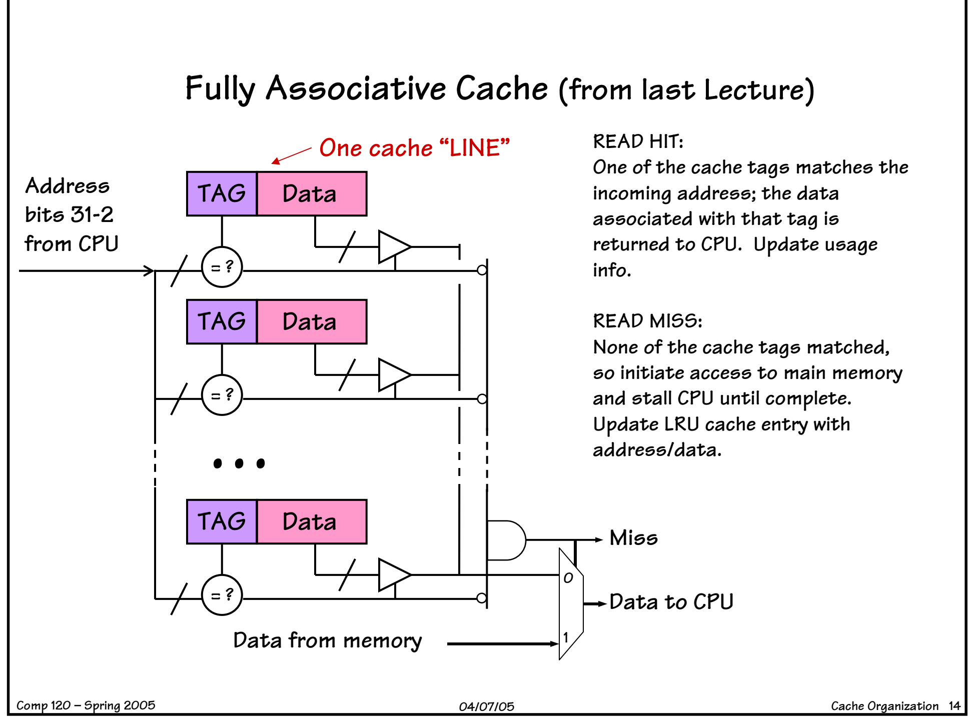
$$t_{ave} = t_{c1} + \beta_1 t_m$$

e.g. 80% of Level-2 request hit (99% of original)

1+0.05x10+0.05x0.2x100=2.5

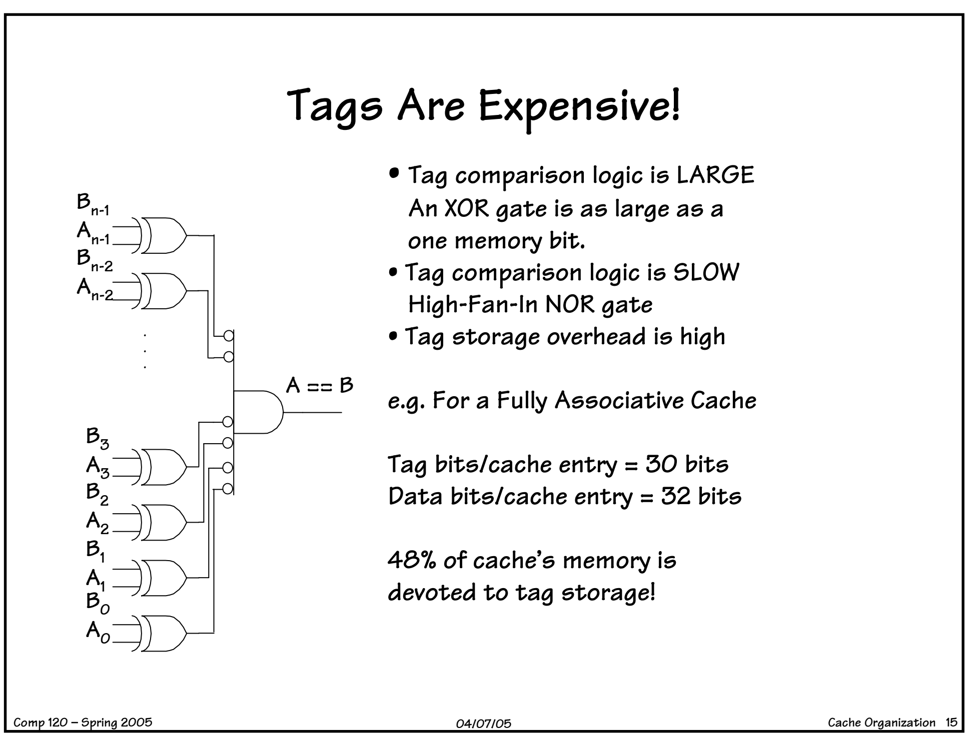
1+0.05x100=6

Fully Associative Cache (from last Lecture)



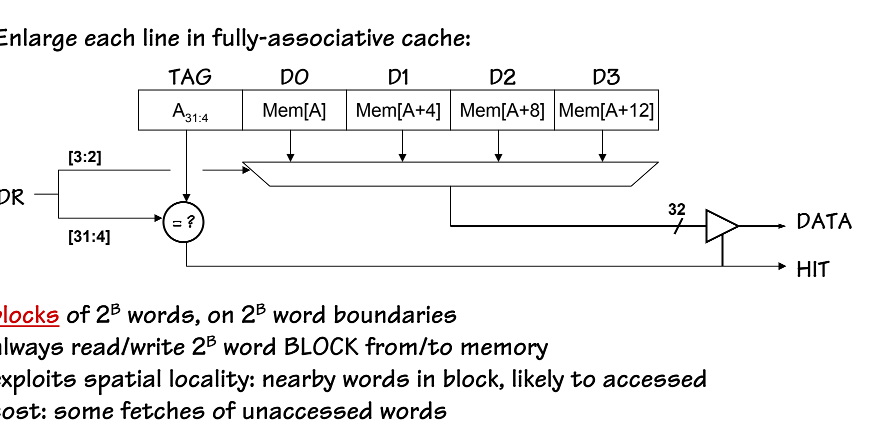
READ HIT:
One of the cache tags matches the incoming address; the data associated with that tag is returned to CPU. Update usage info.

READ MISS:
None of the cache tags matched, so initiate access to main memory and stall CPU until complete. Update LRU cache entry with address/data.



Amortize Tag Costs: More Data/Tag

Enlarge each line in fully-associative cache:

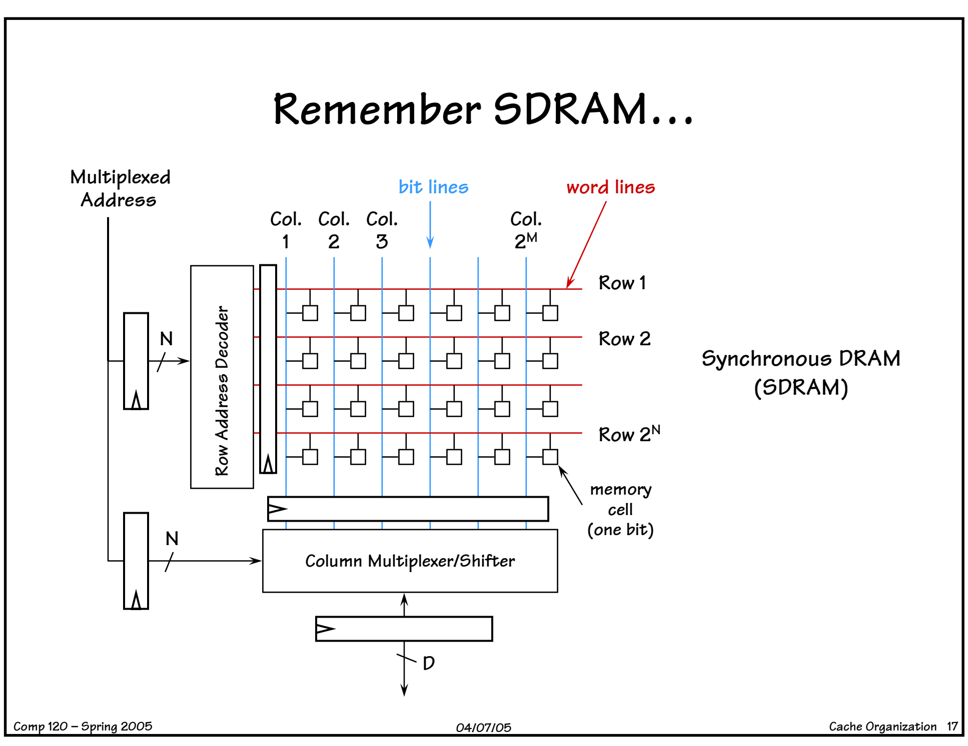


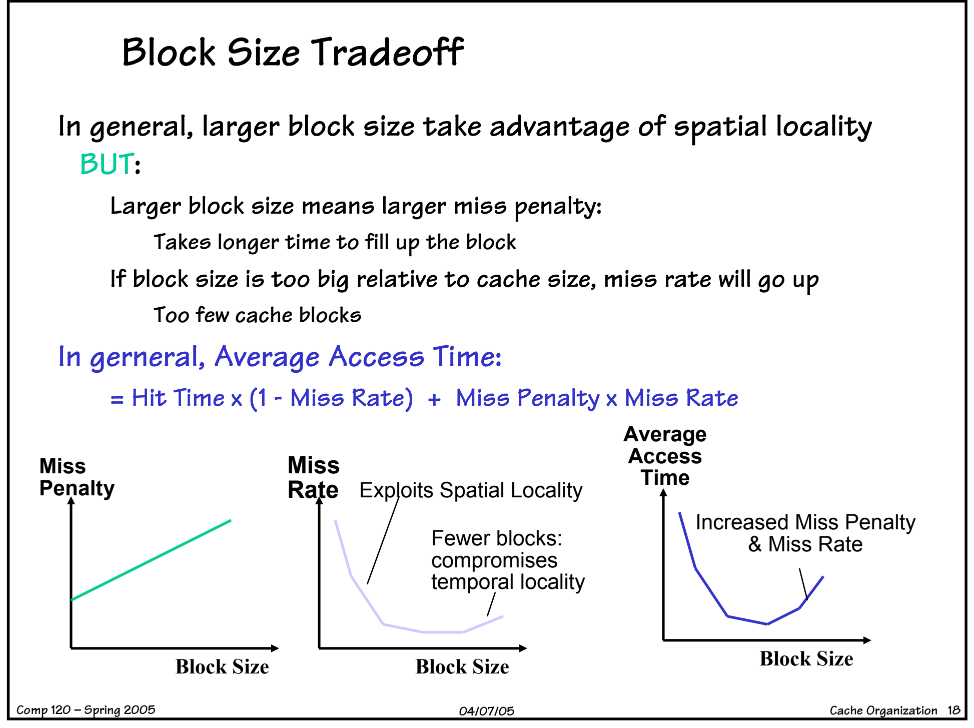
- blocks of 2^B words, on 2^B word boundaries
- always read/write 2^B word BLOCK from/to memory
- exploits spatial locality: nearby words in block, likely to accessed
- cost: some fetches of unaccessed words
- BIG WIN if path to memory is wide, or sequential accesses are fast (SDRAM, see next slide)

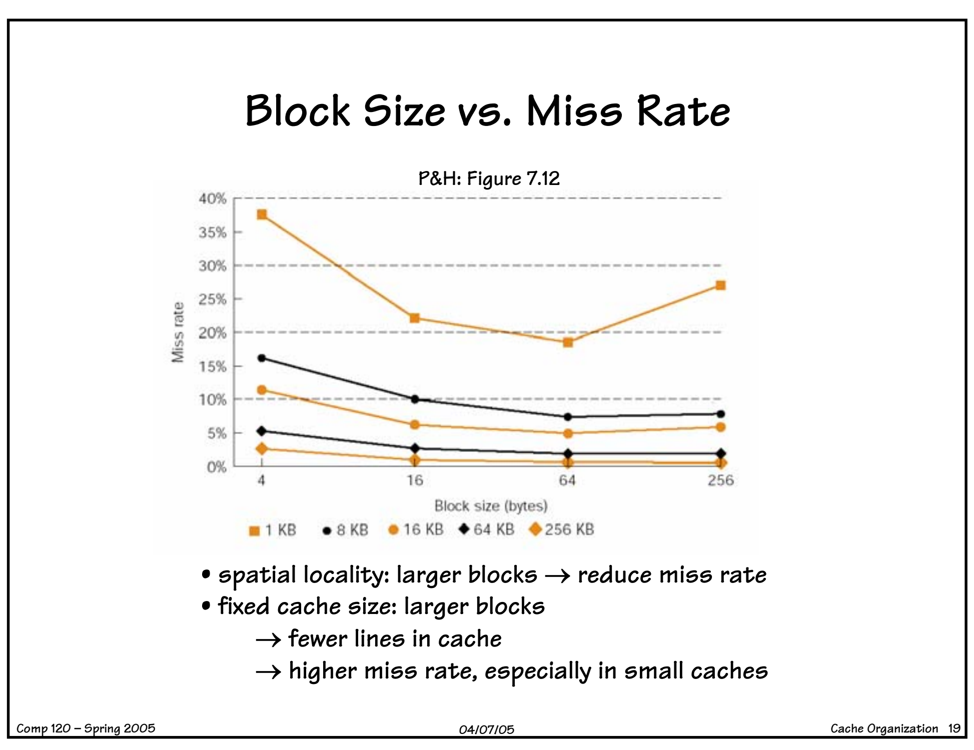
Tag bits/cache entry = (30 – 2) bits

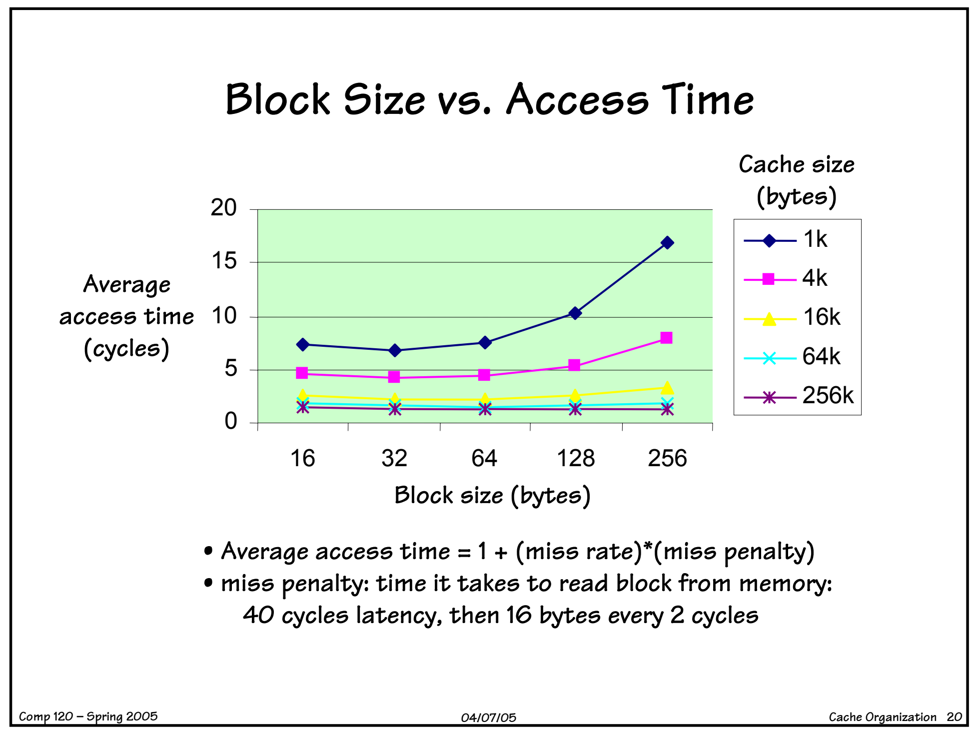
Data bits/cache entry = 4*32 bits Only 18% of cache's memory used for tags

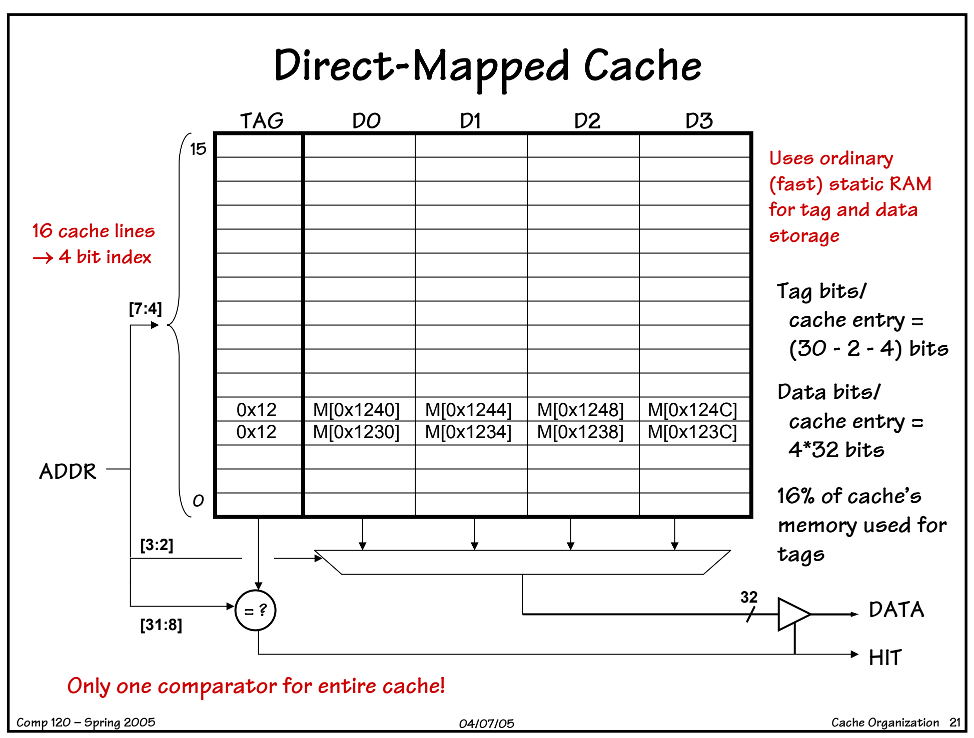
Comp 120 – Spring 2005 04/07/05 Cache Organization 16


Remember SDRAM…
Multiplexed Address
bit lines
word lines
Col. 1
Col. 2
Col. 3
Col. 2M
Row 1
Row 2
Row 2N
N
Row Address Decoder
Synchronous DRAM (SDRAM)
memory cell (one bit)
N
Column Multiplexer/Shifter
D
Comp 120 – Spring 2005
04/07/05
Cache Organization 17


Block Size Tradeoff
In general, larger block size take advantage of spatial locality
BUT:
Larger block size means larger miss penalty:
Takes longer time to fill up the block
If block size is too big relative to cache size, miss rate will go up
Too few cache blocks
In gerneral, Average Access Time:
= Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate
Miss Penalty
Block Size
Miss Rate
Exploits Spatial Locality
Fewer blocks: compromises temporal locality
Block Size
Average Access Time
Increased Miss Penalty & Miss Rate
Block Size
Comp 120 – Spring 2005
04/07/05
Cache Organization 18


Block Size vs. Miss Rate
P&H: Figure 7.12
40%
35%
30%
25%
20%
15%
10%
5%
0%
Miss rate
4
16
64
256
Block size (bytes)
1 KB
8 KB
16 KB
64 KB
256 KB
• spatial locality: larger blocks → reduce miss rate
• fixed cache size: larger blocks
→ fewer lines in cache
→ higher miss rate, especially in small caches
Comp 120 – Spring 2005
04/07/05
Cache Organization 19


Block Size vs. Access Time
Cache size (bytes)
20
15
10
5
0
Average access time (cycles)
16
32
64
128
256
Block size (bytes)
1k
4k
16k
64k
256k
• Average access time = 1 + (miss rate)*(miss penalty)
• miss penalty: time it takes to read block from memory:
40 cycles latency, then 16 bytes every 2 cycles
Comp 120 – Spring 2005
04/07/05
Cache Organization 20



Fully-Assoc. vs. Direct-mapped

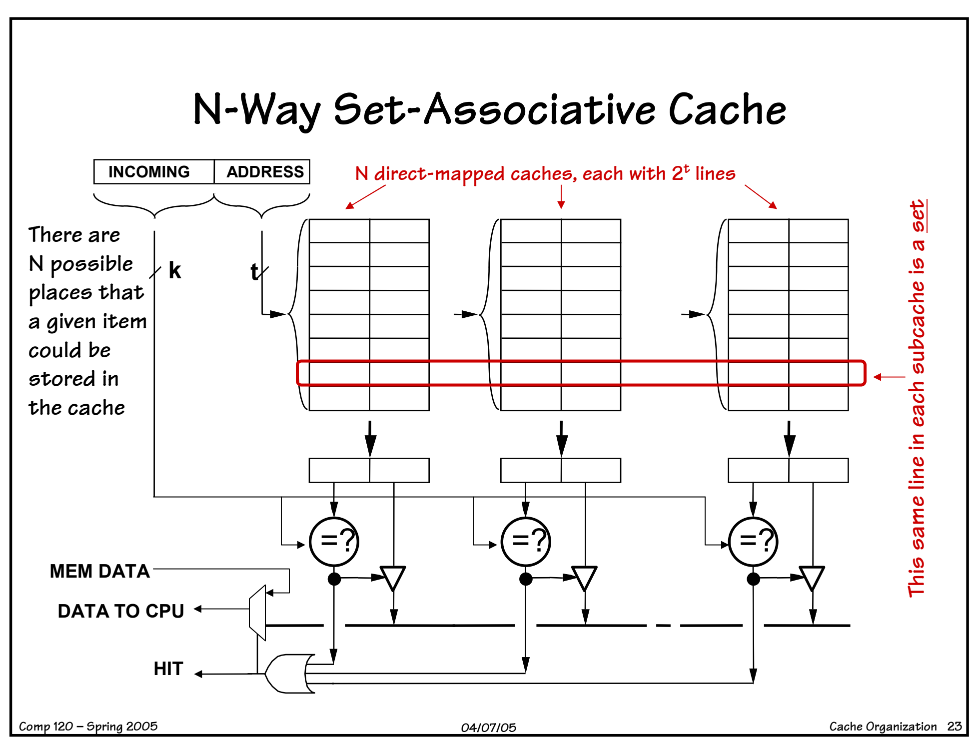
Fully-associative N-line cache:

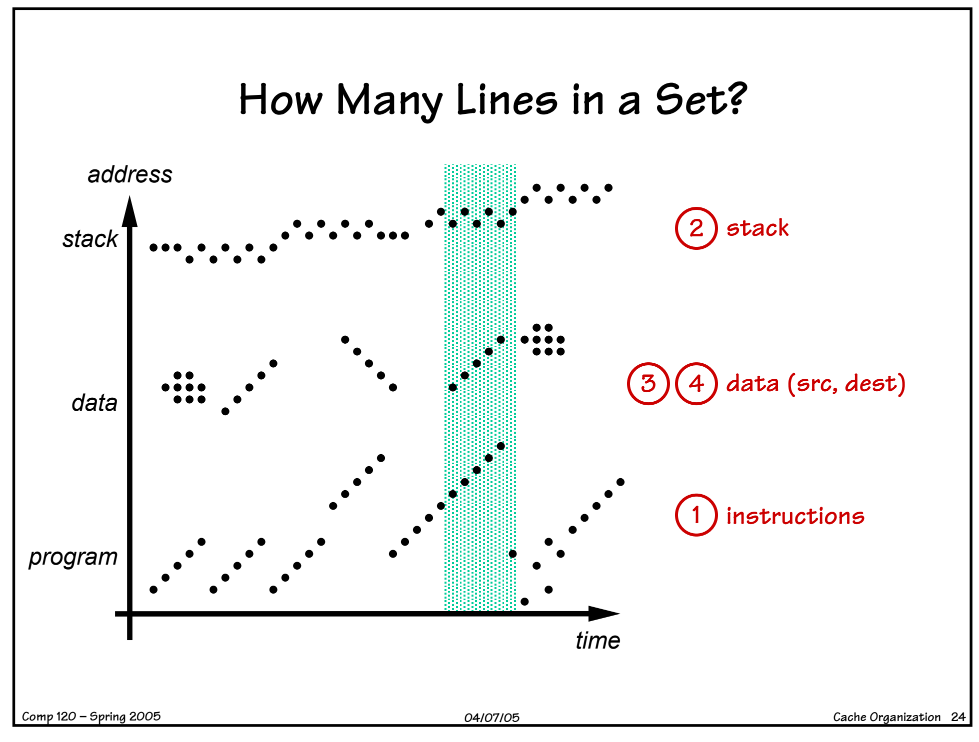
- N tag comparators, registers used for tag/data storage ($$$)
- Location A can be stored in ANY of the N cache lines; no "collisions"
- Replacement strategy (e.g., LRU=Least Recently Used) used to pick which line to use when loading new word(s) into cache

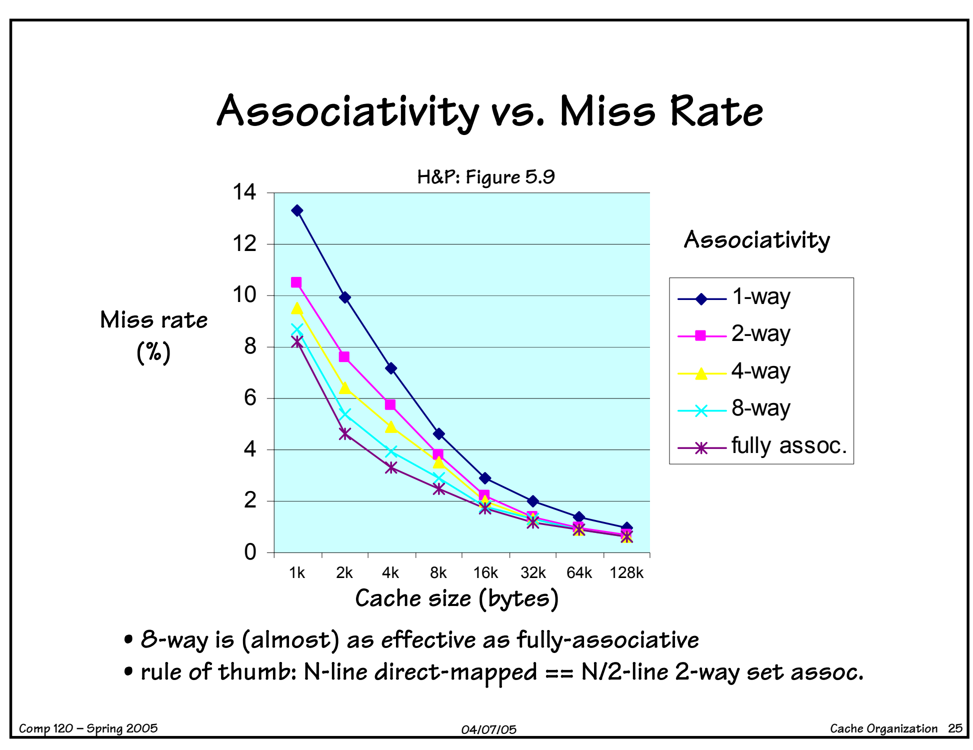
Direct-mapped N-line cache:

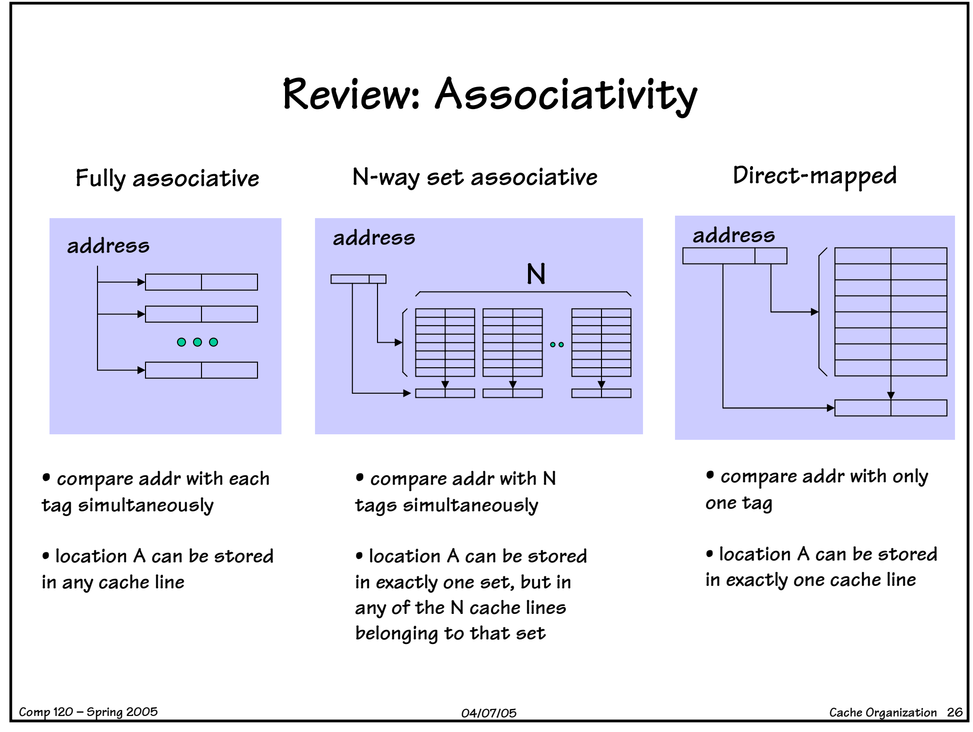
- 1 tag comparator, SRAM used for tag/data storage ($)
- Location A is stored in a SPECIFIC line of the cache determined by its address; address "collisions" possible
- Replacement strategy not needed: each word can only be cached in one specific cache line

Comp 120 – Spring 2005 04/07/05 Cache Organization 22


N-Way Set-Associative Cache
INCOMING
ADDRESS
N direct-mapped caches, each with 2^t lines
There are N possible places that a given item could be stored in the cache
k
t
This same line in each subcache is a set
=?
MEM DATA
DATA TO CPU
HIT
Comp 120 – Spring 2005
04/07/05
Cache Organization 23


How Many Lines in a Set?
address
stack
data
program
time
2 stack
3 4 data (src, dest)
1 instructions
Comp 120 – Spring 2005
04/07/05
Cache Organization 24


Associativity vs. Miss Rate
H&P: Figure 5.9
Miss rate (%)
14
12
10
8
6
4
2
0
1k
2k
4k
8k
16k
32k
64k
128k
Cache size (bytes)
Associativity
1-way
2-way
4-way
8-way
fully assoc.
• 8-way is (almost) as effective as fully-associative
• rule of thumb: N-line direct-mapped == N/2-line 2-way set assoc.
Comp 120 – Spring 2005
04/07/05
Cache Organization 25


Review: Associativity
Fully associative
N-way set associative
Direct-mapped
address
address
N
address
• compare addr with each tag simultaneously
• location A can be stored in any cache line
• compare addr with N tags simultaneously
• location A can be stored in exactly one set, but in any of the N cache lines belonging to that set
• compare addr with only one tag
• location A can be stored in exactly one cache line
Comp 120 – Spring 2005
04/07/05
Cache Organization 26

A Summary on Sources of Cache Misses

Compulsory (cold start or process migration, first reference): first access to a block

- "Cold" fact of life: not a whole lot you can do about it
- Note: If you are going to run "billions" of instruction, Compulsory Misses are insignificant

Conflict (collision):

- Multiple memory locations mapped to the same cache location
- Solution 1: increase cache size
- Solution 2: increase associativity

Capacity:

- Cache cannot contain all blocks access by the program
- Solution: increase cache size

Invalidation: other process (e.g., I/O) updates memory

Sources of Cache Misses Answer

	Direct Mapped	**N-way Set Associative**	**Fully Associative**
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low	Medium	High
Invalidation Miss	Same	Same	Same

Note:
If you are going to run "billions" of instruction, Compulsory Misses are insignificant.

More $ Next Time

1) Replacement Strategies – what cache line do we replace on a miss? What bookkeeping logic is required?
2) Writes to Cache – should we update memory on writes or only the cache?
3) Cache bookkeeping – overhead for keeping track of least-recently used cache items, "dirty" cache lines, etc.
4) Cache Simulations – How do the choices of block-size, associativity, replacement strategy, and writing policy effect cache performance