Memory Hierarchy

1

Programmers wants unlimited fast memory

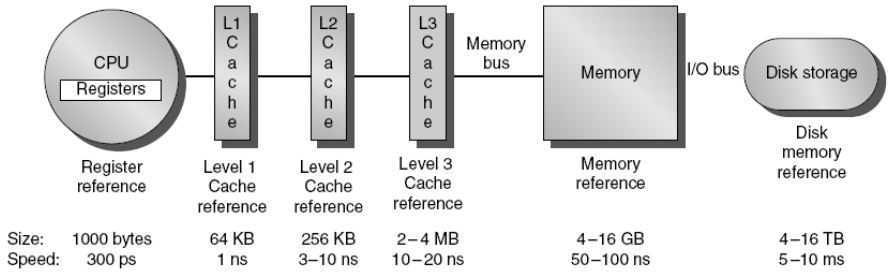
An economical solution is a *memory hierarchy,*

- which takes advantage of locality and trade-offs in the cost-performance of memory technologies.
- The *principle of locality,* says that programs do not access all code or data uniformly.
  - Locality occurs in time *(temporal locality)* and
  - in space *(spatial locality).*

2

2

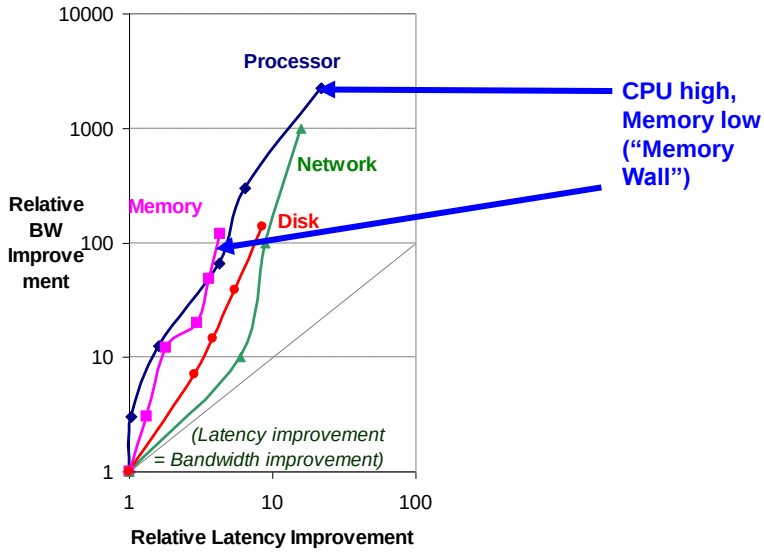
Levels of the Memory Hierarchy



3

3

Latency Lags Bandwidth (last ~20 years)



4

4

Intel i7 (1)

Intel Core i7 can generate two data memory references per core each clock cycle

- with four cores and a 3.2 GHz clock rate, the i7 can generate a peak of 25.6 billion 64-bit data memory references per second,

- to a peak instruction demand (for four cores) of about 12.8 billion 128-bit instruction references per second;
- total peak bandwidth of 409.6 GB/sec

Intel i7 (2)

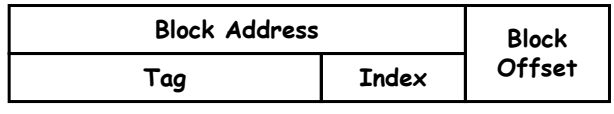
- In contrast, the peak bandwidth to DRAM main memory is only 6% of this
  - 25 GB/sec

Intel i7 (3)

- This incredible bandwidth is achieved
  - by multiporting and pipelining the cache accesses;
  - by the use of multiple levels of caches,
  - by using a separate instruction and data cache at the first level,
  - By using separate first- and sometimes second-level caches per core.

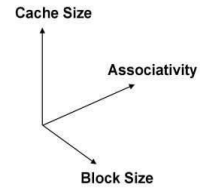
Cache memory basics

- When a word is not found in the cache, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Lower level may be another cache or the main memory
  - Also fetch the other words contained within the *block*
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets



Summary: The Cache Design Space

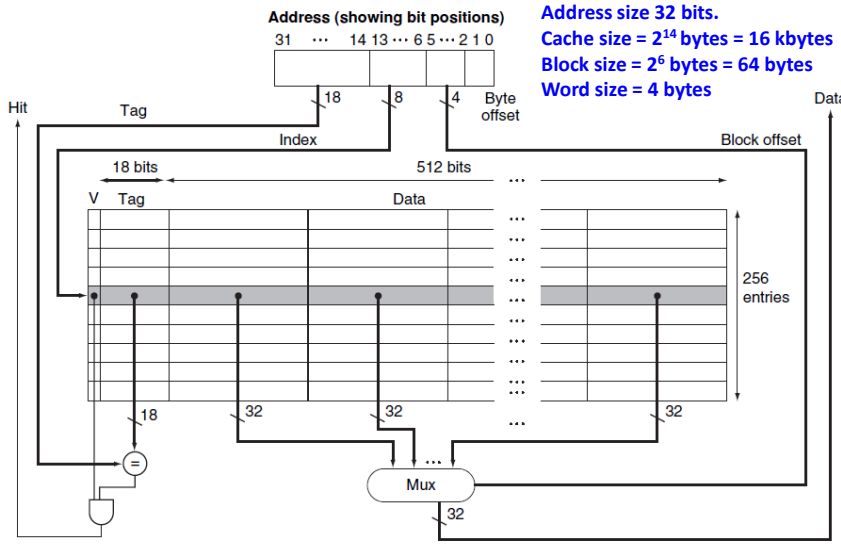
- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation



The cache performance depends also by:
- **Locality features of the program**
- **Compiler**
- **Input values**

9

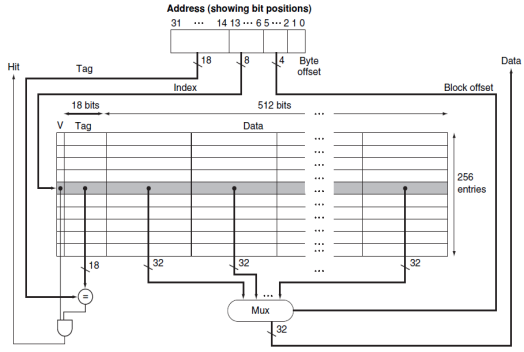
Direct access cache memory (1)



Address size 32 bits.
Cache size = 2^{14} bytes = 16 kbytes
Block size = 2^{6} bytes = 64 bytes
Word size = 4 bytes

10

Direct access cache memory (2)

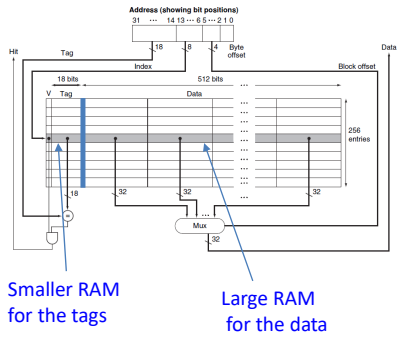


Address size 32 bits.
Cache size = 2^{14} bytes = 16 kbytes
Block size = 2^{6} bytes = 64 bytes
Word size = 4 bytes

Number of pages = $2^{32}/2^{14}$ pages= 2^{18} pages
tag field is 18 bits wide
Number of cache blocks = Cache size/block size
= 2^{8} blocks

and the index field is 8 bits wide, while a 4-bit field (bits 5–2) is used to index the block and select the word from the block using a 16-to-1 multiplexor.

Direct access cache memory (3)



Smaller RAM for the tags

Large RAM for the data

Address size 32 bits.

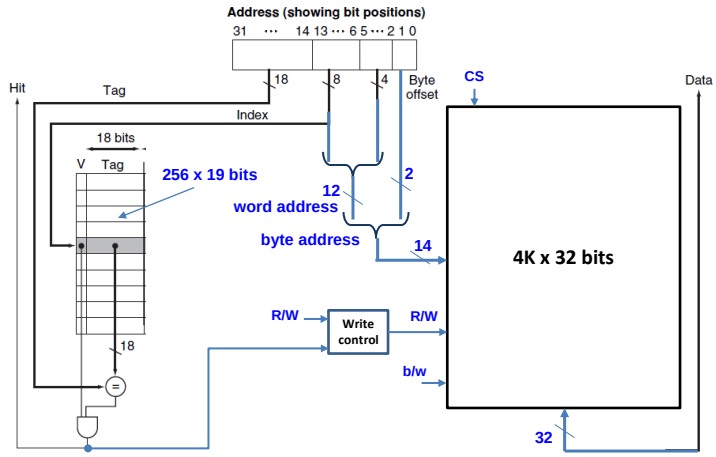
Cache size = 2^{14} bytes = 16 kbytes
Block size = 2^{6} bytes = 64 bytes
Word size = 4 bytes

Number of pages = $2^{32}/2^{14}$ pages= 2^{18} pages
tag field is 18 bits wide
Number of cache blocks = Cache size/block size = 2^{8} blocks

In the implementation, caches use a separate large RAM for the data and a smaller RAM for the tags, with the block offset supplying the extra address bits for the large data RAM.

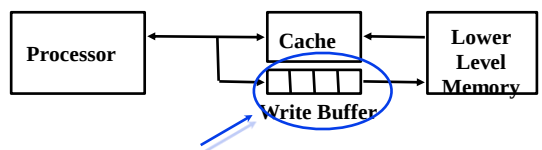
Direct access cache memory

an implementation



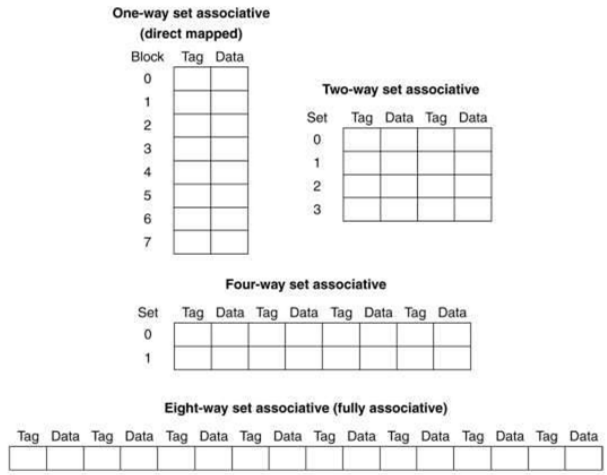
Cache memory basics

- *n* sets => *n-way set associative*
  - *Direct-mapped cache* => one block per set
  - *Fully associative* => one set
- Writing to cache: two strategies
  - *Write-through*
    - Immediately update lower levels of hierarchy
  - *Write-back*
    - Only update lower levels of hierarchy when an updated block is replaced
  - Both strategies use *write buffer* to make writes asynchronous



Holds data awaiting write-through to lower level memory

Cache Memory – Associative Cache

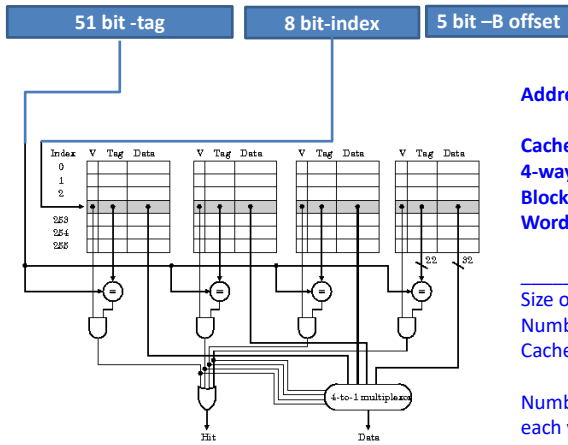


15

15

Cache Memory – Set Associative Cache

Block based – L1 cache



Address size 64 bits.

Cache size = 32 kbytes
4-way set associative cache
Block size = 2^5 bytes = 32 bytes
Word size = 8 bytes

Size of each way = cache size/4= 8 kbytes
Number of cache blocks for each way = Cache size/(4 *block size) = 2^8 blocks

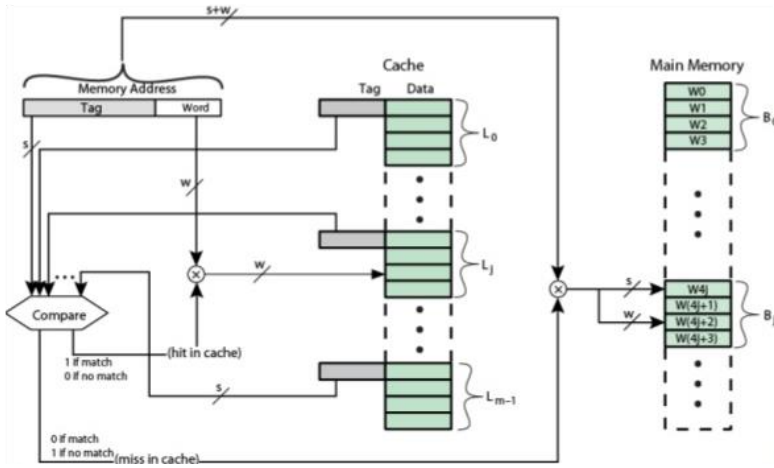
Number of pages = size of memory/ Size of each way = $2^{64}/2^{13}$ pages= 2^{51} pages

tag field is 51 bits wide

16

16

Cache Memory – Fully Associative Cache

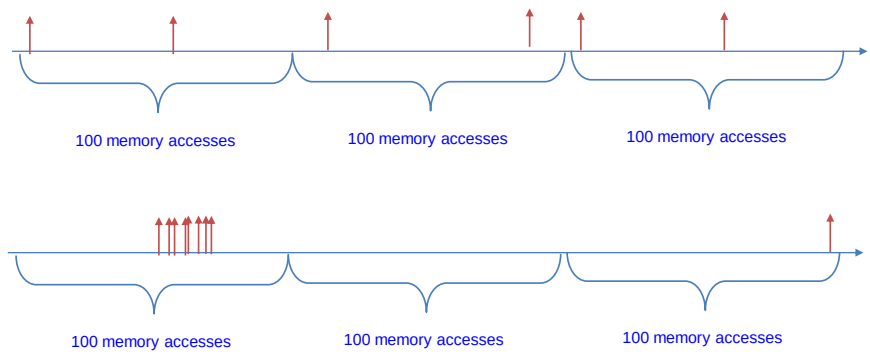


17

17

Cache effects: unbalanced distribution of misses

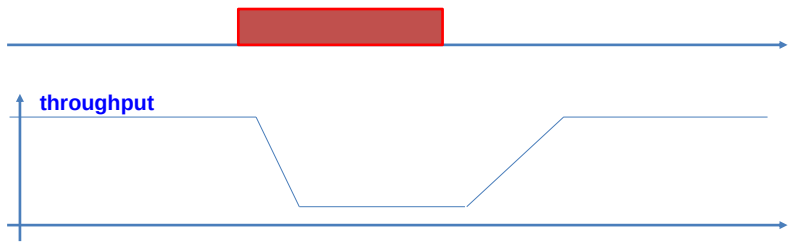
2% of misses



18

18

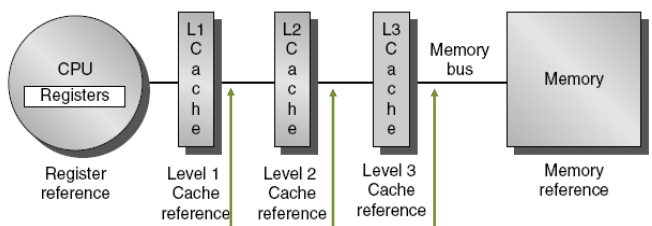
Cache effects:
unbalanced distribution of misses



19

19

Cache effects:
reads and writes on the memory bus



Miss percentage: 5 2 2

Reads blocks (15%) for 100 accesses: 5 1 1 (6%)

Writes (15%) for 100 accesses: 15 15 15 (94%)

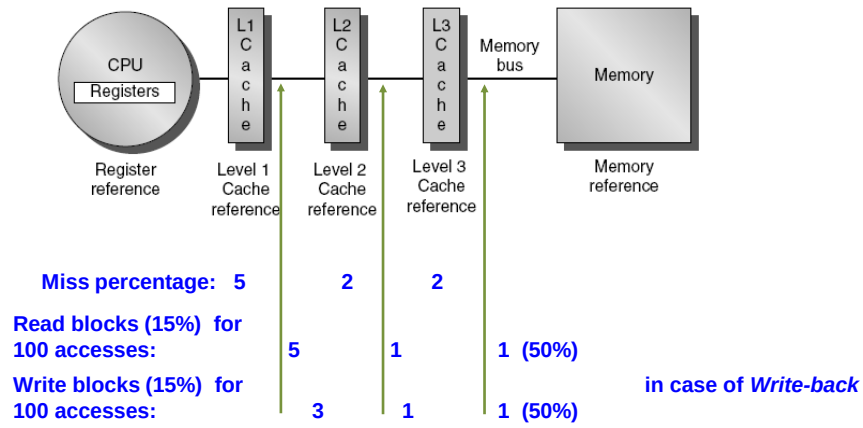
in case of *Write-through*

Write-through doesn't use temporal and spatial localities of writes

20

20

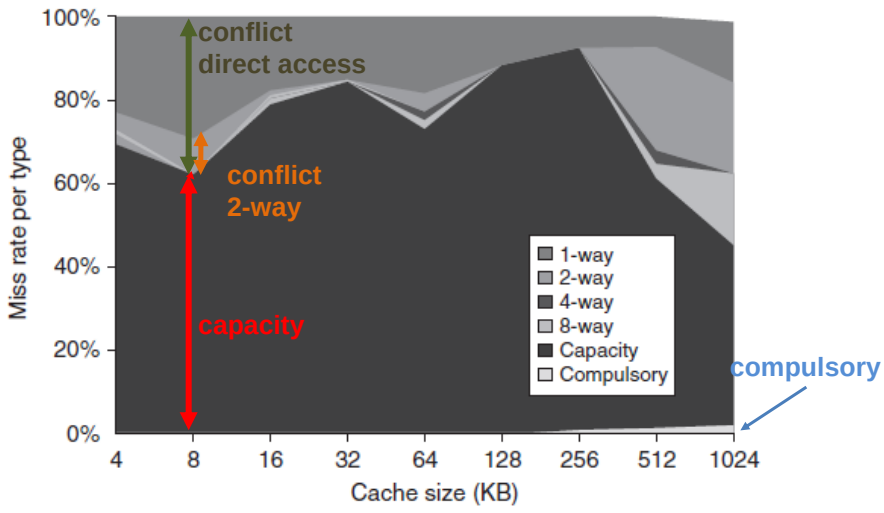
Cache effects:
reads and writes on the memory bus



Cache memory basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

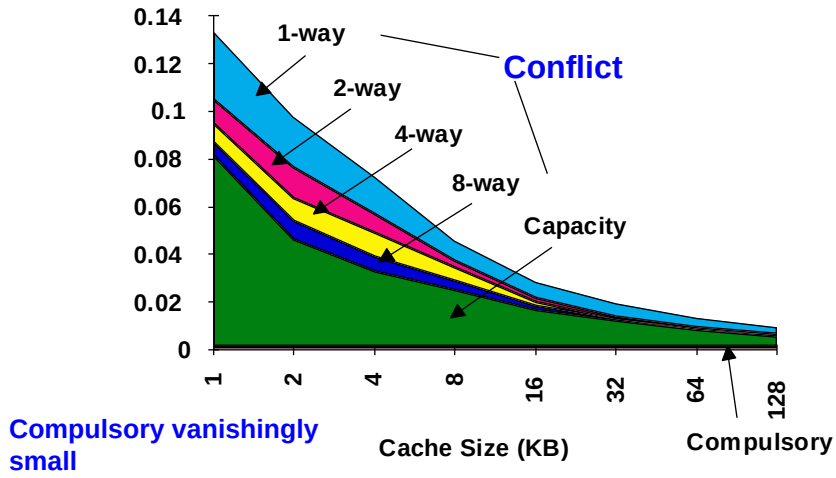
Compulsory, capacity and conflict misses



23

23

3Cs Absolute Miss Rate (SPEC92)

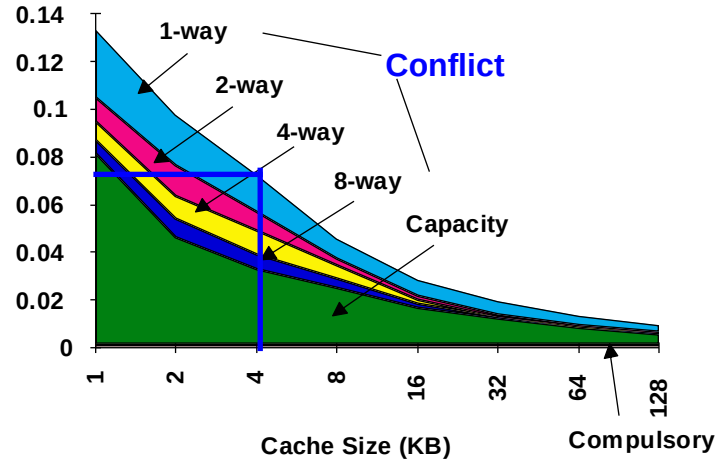


24

24

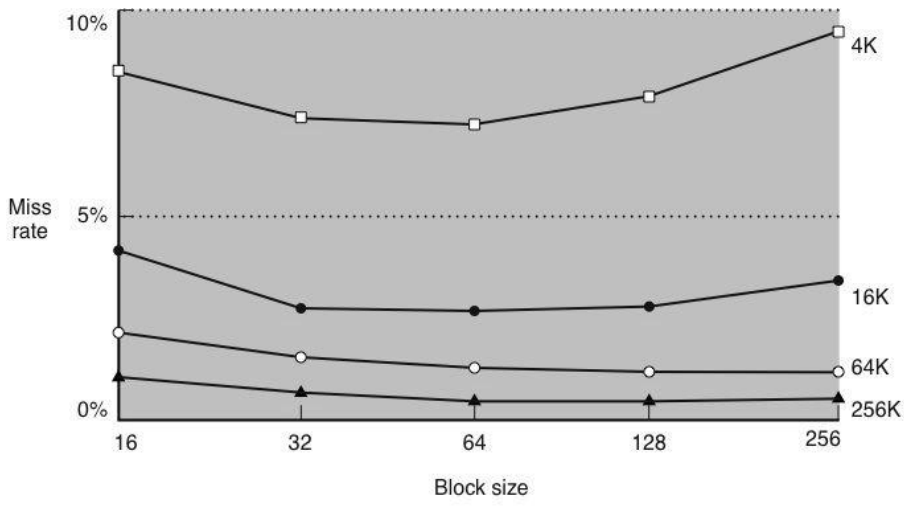
2:1 Cache Rule

miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2



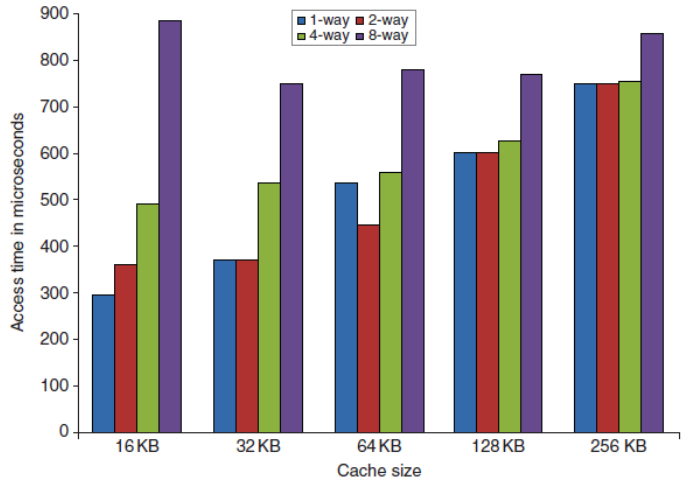
25

25



26

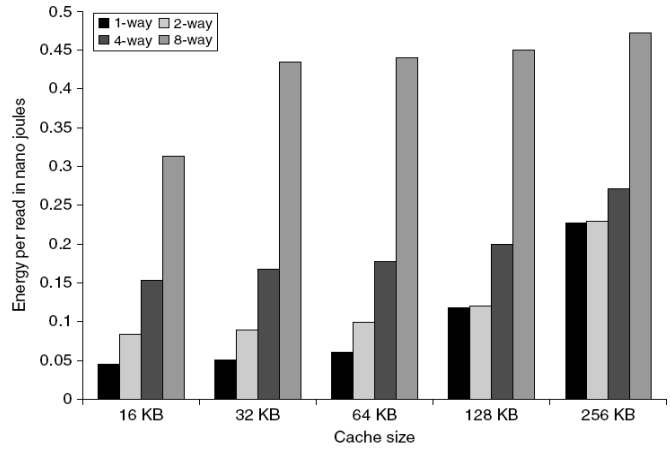
Access times increase as cache size and associativity are increased



27

27

Energy per read vs. size and associativity



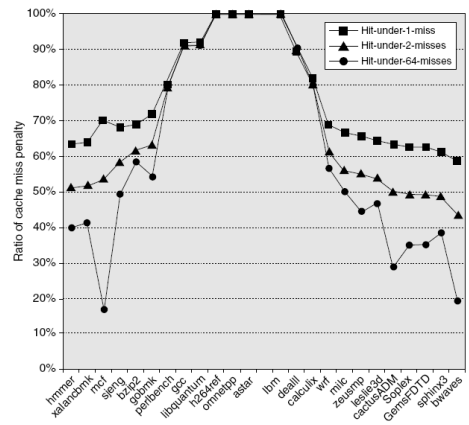
28

Way Prediction

- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - "Way selection"
  - Increases mis-prediction penalty

Nonblocking Caches

- Allow hits before previous misses complete
  - "Hit under miss"
  - "Hit under multiple miss"
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty

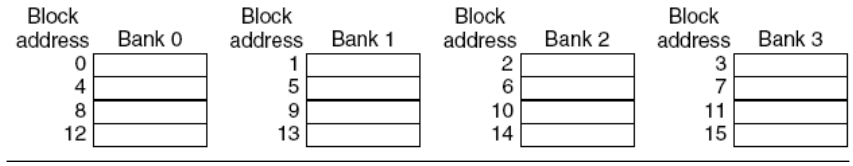


The study was done assuming a model based on a single core of an Intel i7 running the SPEC2006 benchmarks.

Multibanked Caches

- Organize cache as independent banks to support simultaneous accesses
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2

- Interleave banks according to block address



31

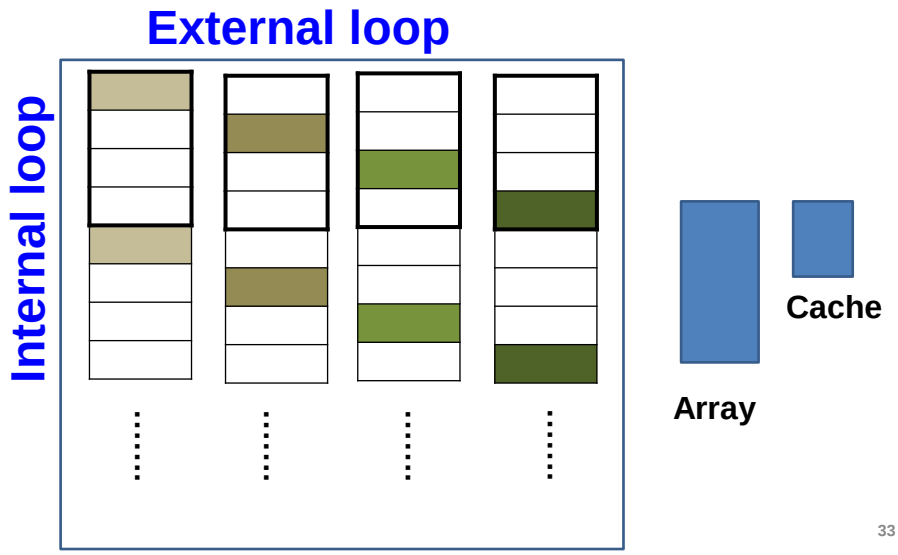
Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order

- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

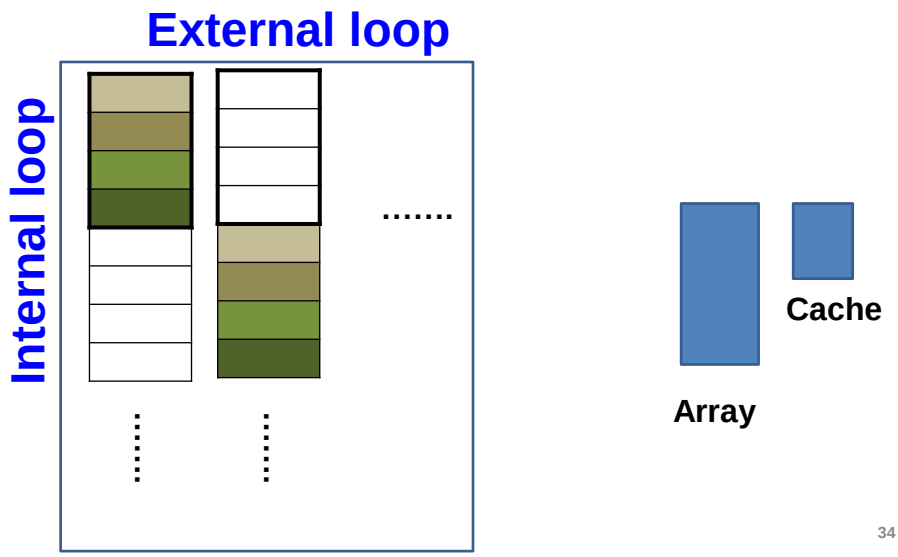
32

An example of nested loops



33

An example of nested loops



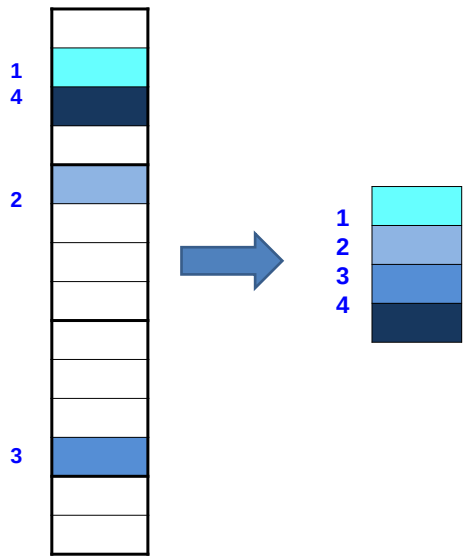
34

Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order
- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

35

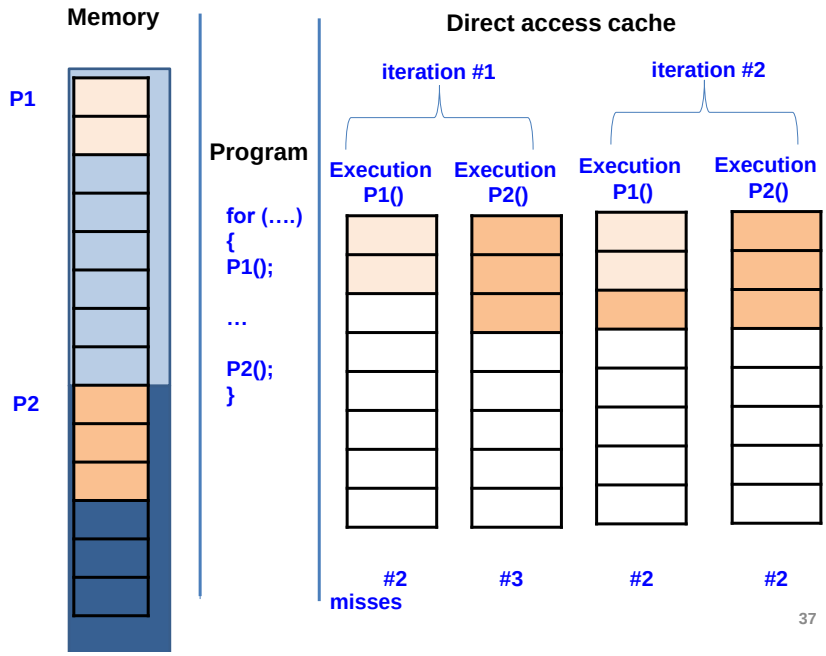
Improve temporal locality



36

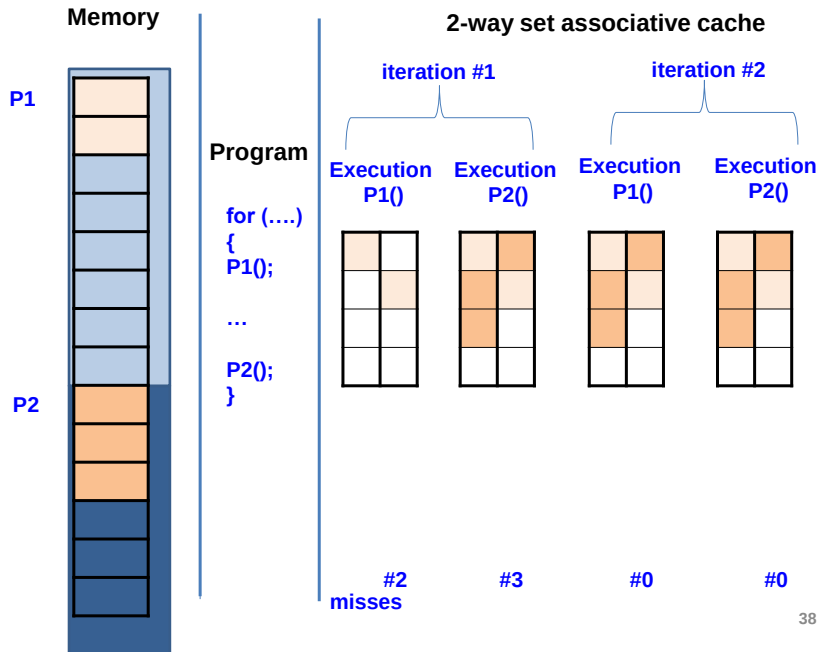
36

Conflict misses



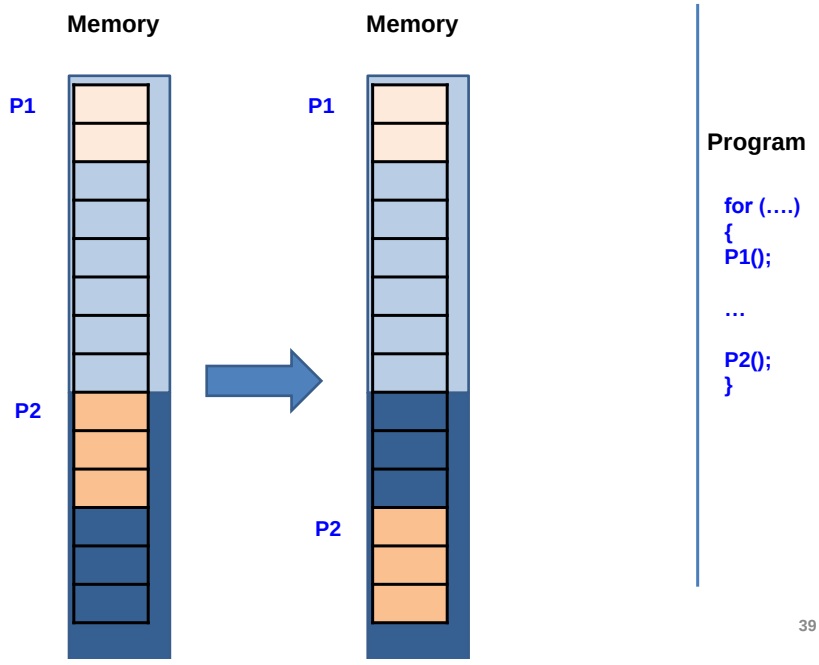
37

Hardware solution



38

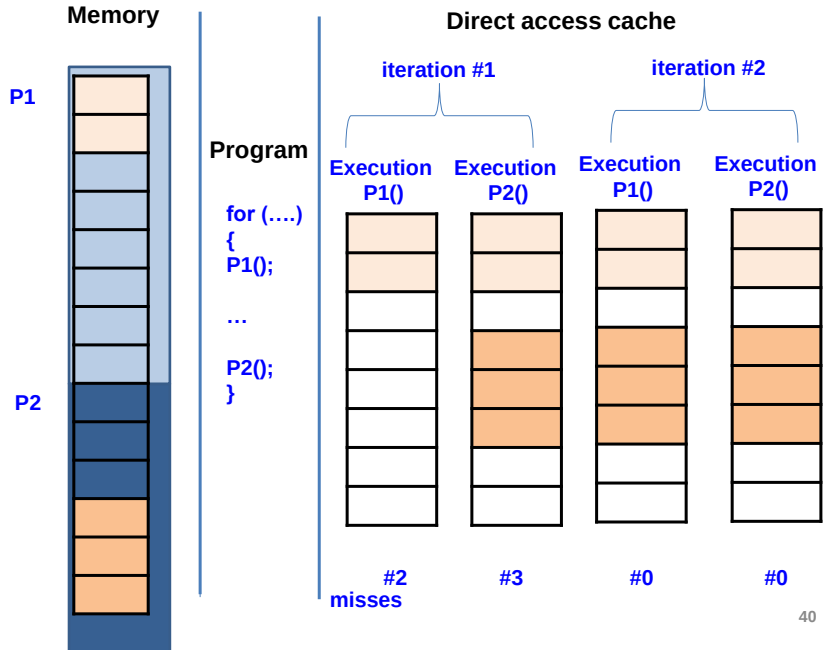
Software solution



39

39

Software solution



40

40