MEMORY HIERARCHY

Memory Hierarchy

- Registers
  - In CPU
- Primary Memory
  - May include one or more levels of cache
  - “RAM”
- Secondary memory
  - Magnetic/Optical

CACHE:

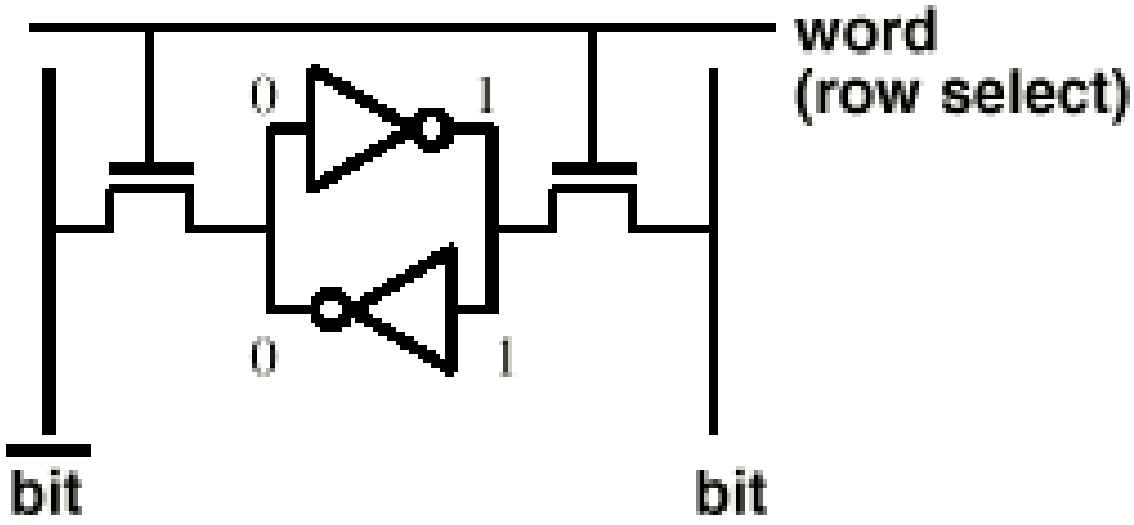
-High Speed Memory (SRAMs)

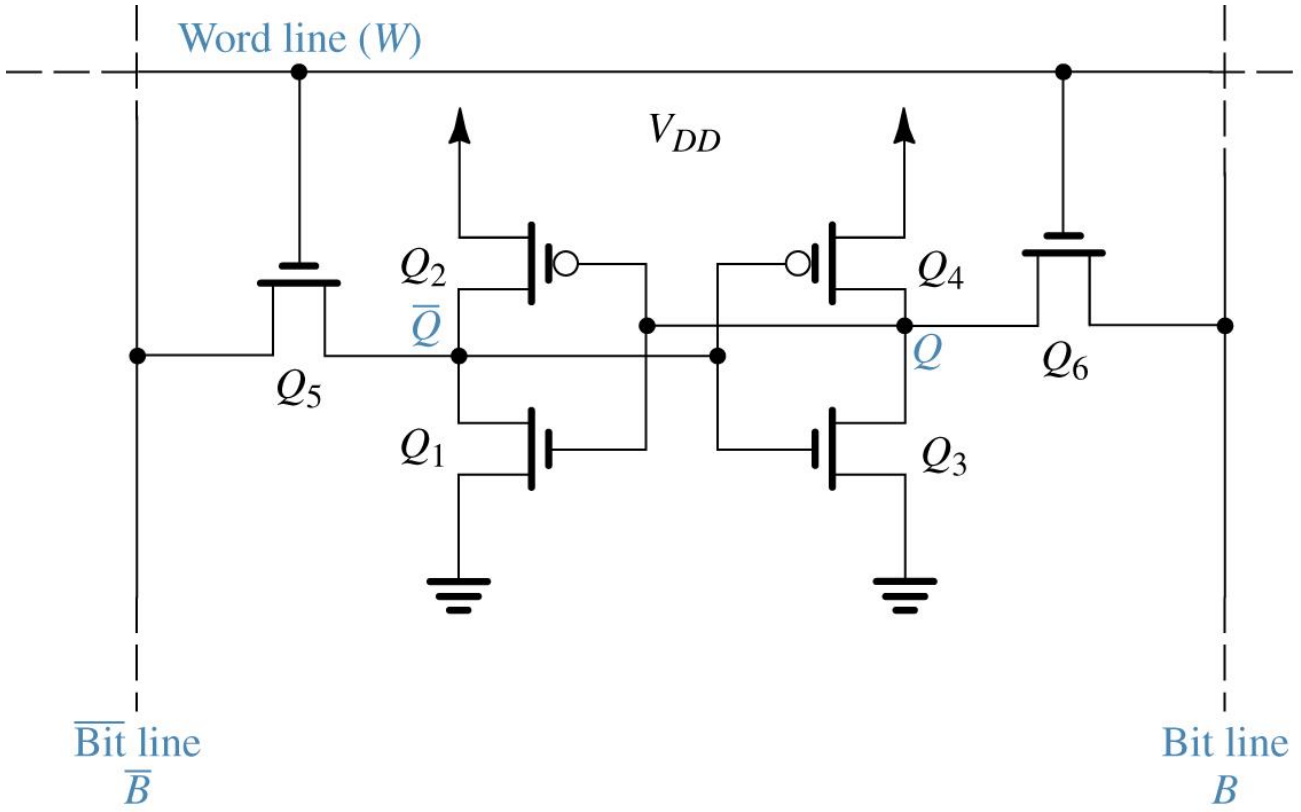
-Small in Size

-High cost

Static RAM Cell

6-Transistor SRAM Cell




Word line (W)
V_{DD}
Q_2
$\overline{Q}$
Q_4
Q
Q_5
Q_6
Q_1
Q_3
$\overline{\text{Bit line}}$
$\overline{B}$
Bit line
B

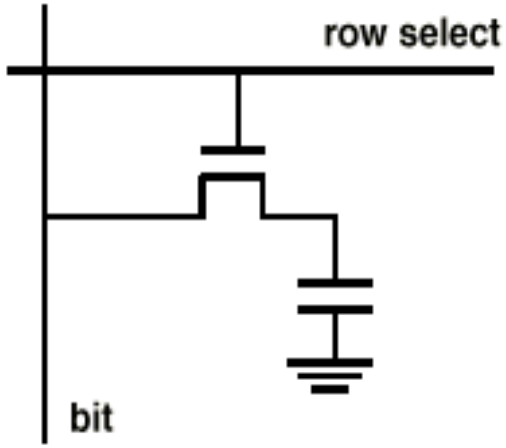
MAIN MEMORY

-High density (DRAMs)

-Low cost

-Slower than Cache.

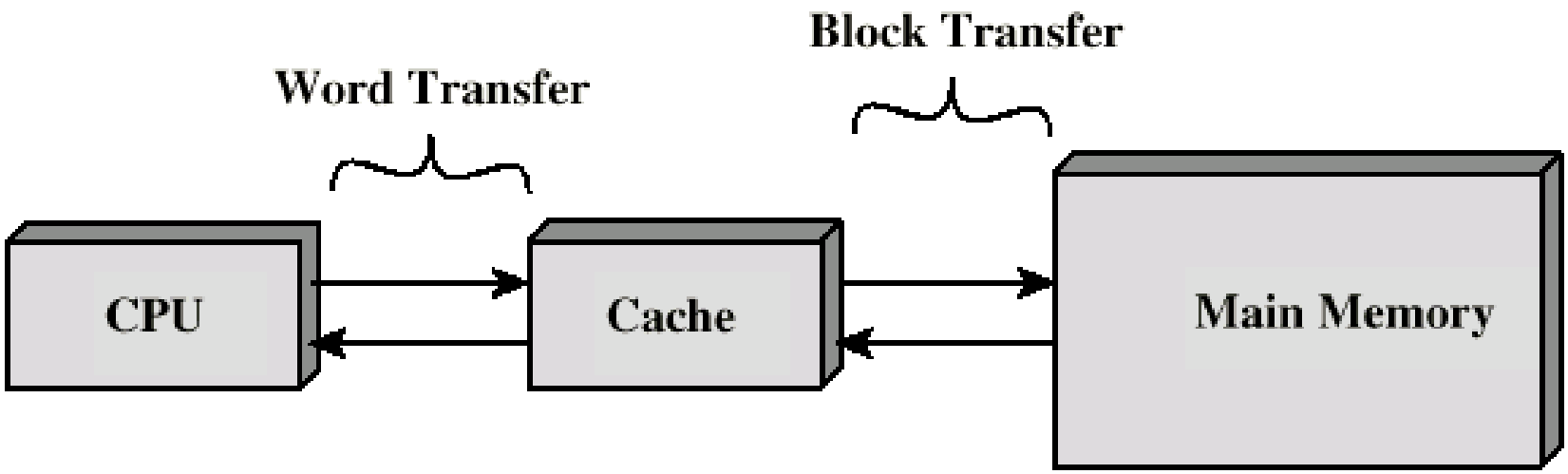
1-Transistor Memory Cell (DRAM)



- Write:
  1. Drive bit line
  2. Select row
- Read:
  1. Precharge bit line to Vdd
  2. Select row
  3. Cell and bit line share charges
     Very small voltage changes on the bit line
  4. Sense (fancy sense amp)
  5. Write: restore the value
- Refresh
  1. Just do a dummy read to every cell.

Cache

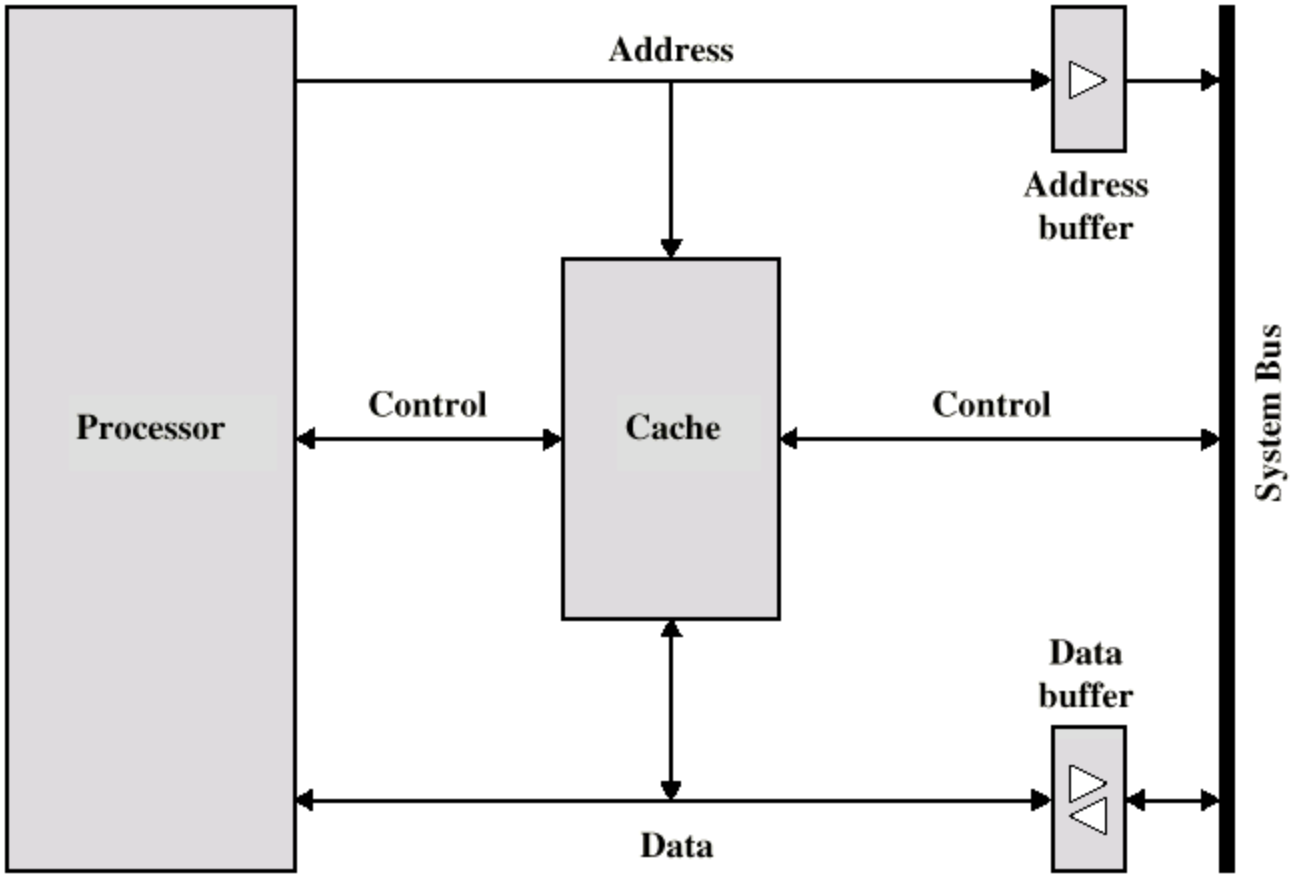
- Small amount of fast memory
- Sits between normal main memory and CPU



Cache operation - overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Typical Cache Organization



Mapping Function

- Cache of 64kByte
- Cache block of 4 bytes
  - i.e. cache is 16k (2^{14}) lines of 4 bytes
- 16MBytes main memory
- 24 bit address
  - (2^{24}=16M)

Direct Mapping

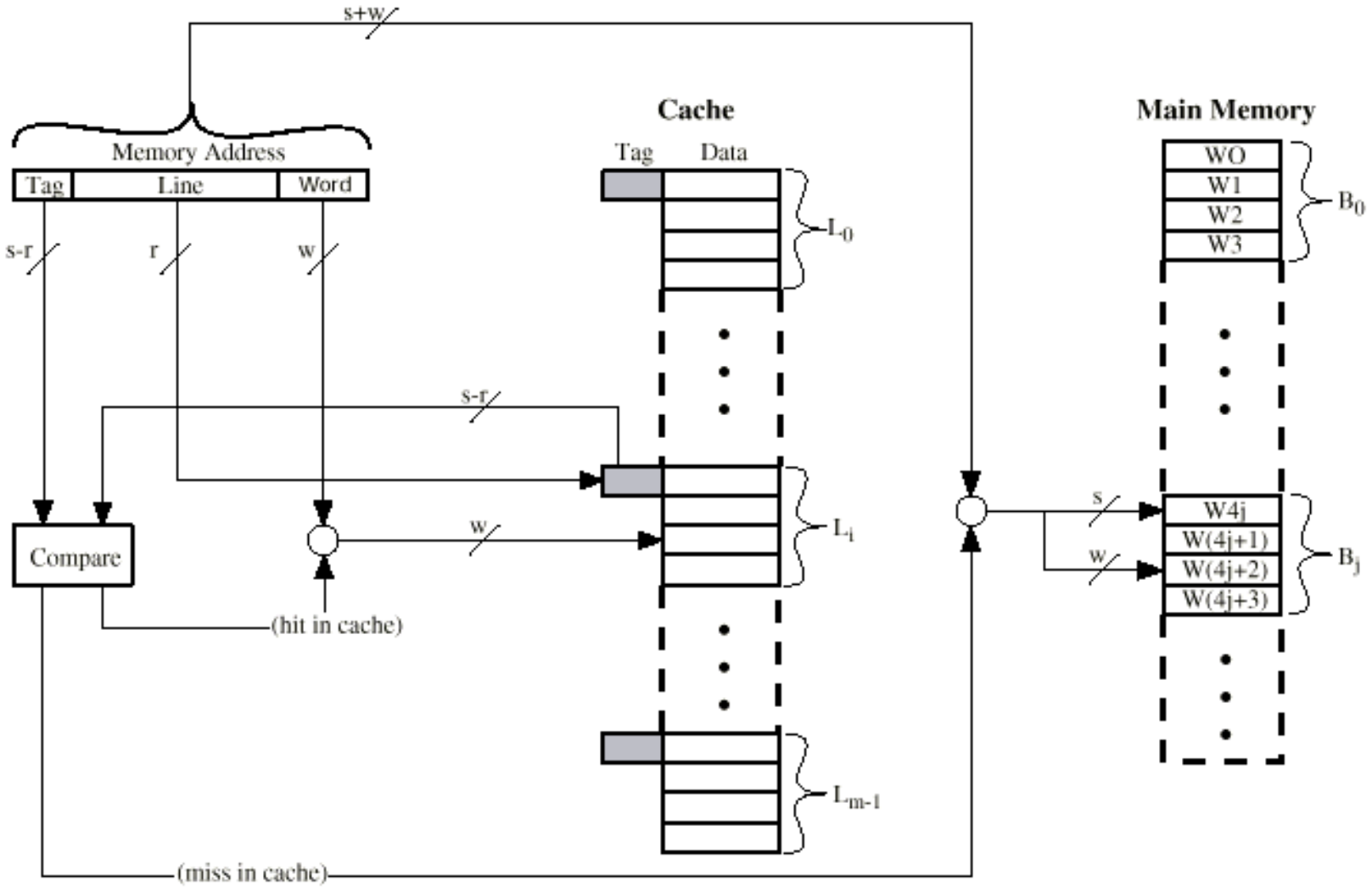
- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- Address is in two parts
- Least Significant w bits identify unique word
- Most Significant s bits specify one memory block
- The MSBs are split into a cache line field r and a tag of s-r (most significant)

Direct Mapping Address Structure

Tag s-r	Line or Slot r	Word w
8	14	2

- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
  - 8 bit tag (=22-14)
  - 14 bit slot or line
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag

Direct Mapping Cache Organization



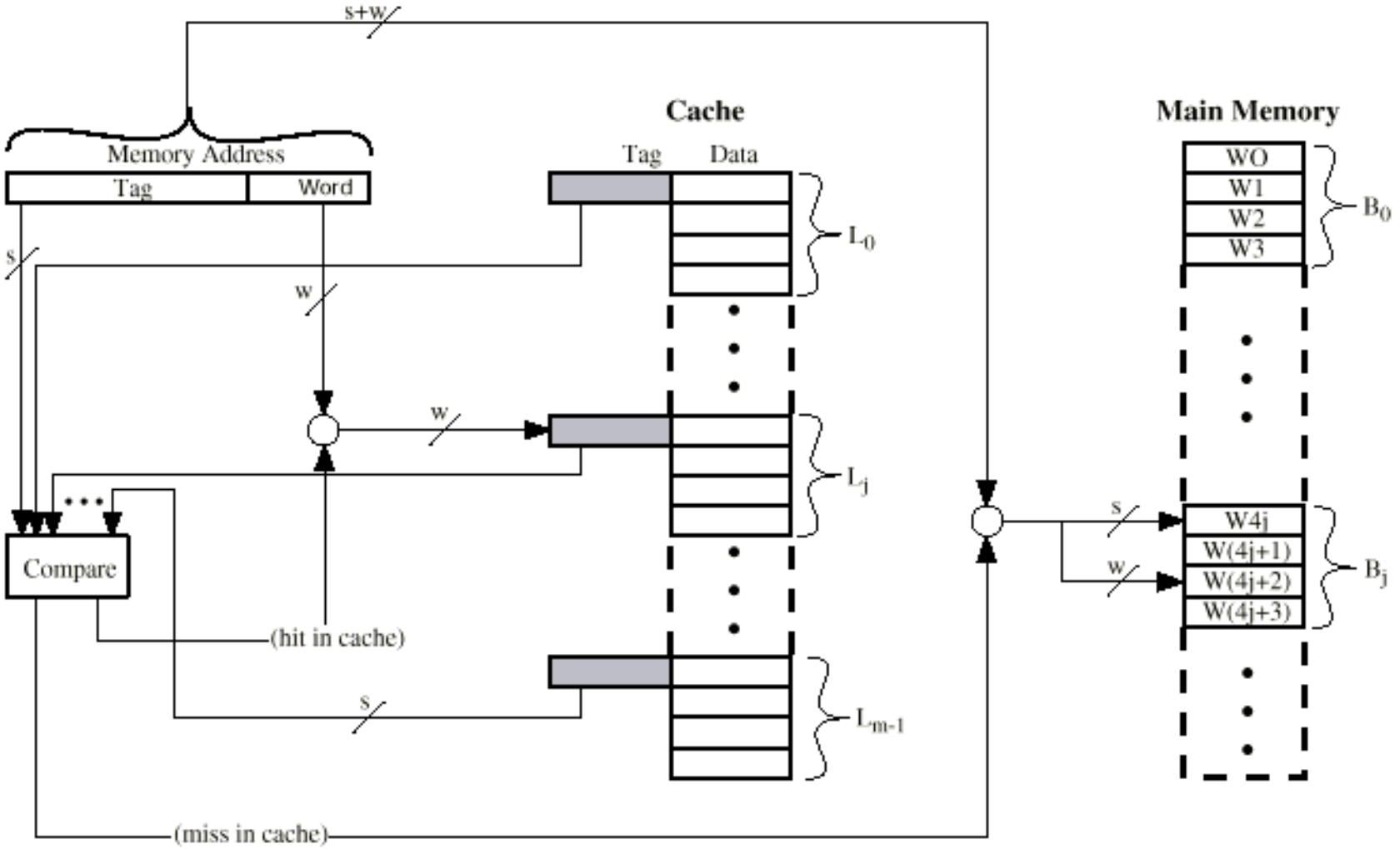
Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

Associative Mapping

- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line’s tag is examined for a match
- Cache searching gets expensive

Fully Associative Cache Organization



Associative Mapping Address Structure

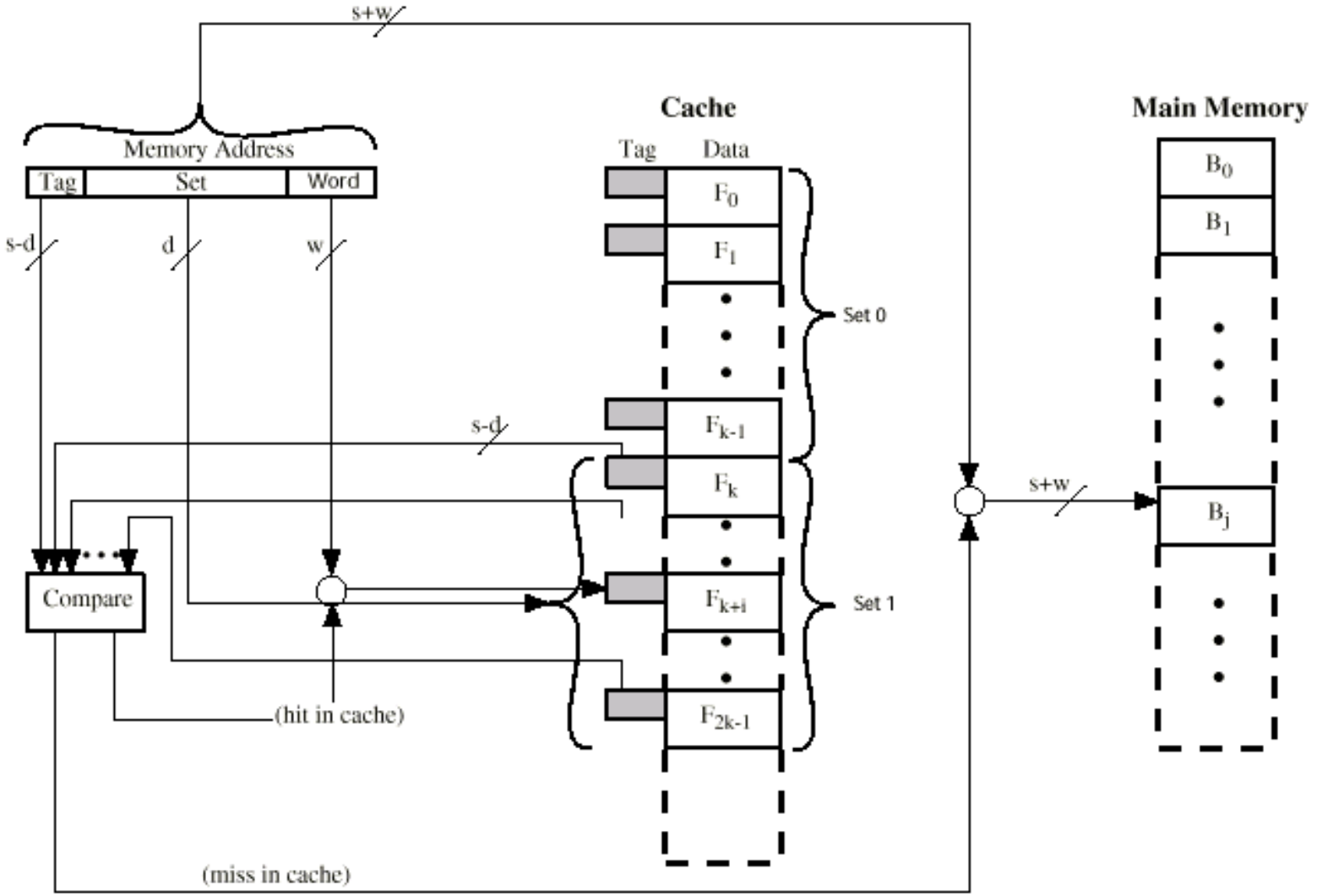
Tag 22 bit	Word 2 bit

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit

Set Associative Mapping

- Cache is divided into a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set

K- Way Set Associative Cache Organization



Set Associative Mapping Address Structure

Tag 9 bit	Set 13 bit	Word 2 bit

- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit

Main Memory 64kB

Block size 8 bytes

Direct Mapped Cache – 32 lines

How is the 16 bit mem add divided

Into what line a byte with add below be stored

0001 0001 0001 1011

1100 0011 0011 0100

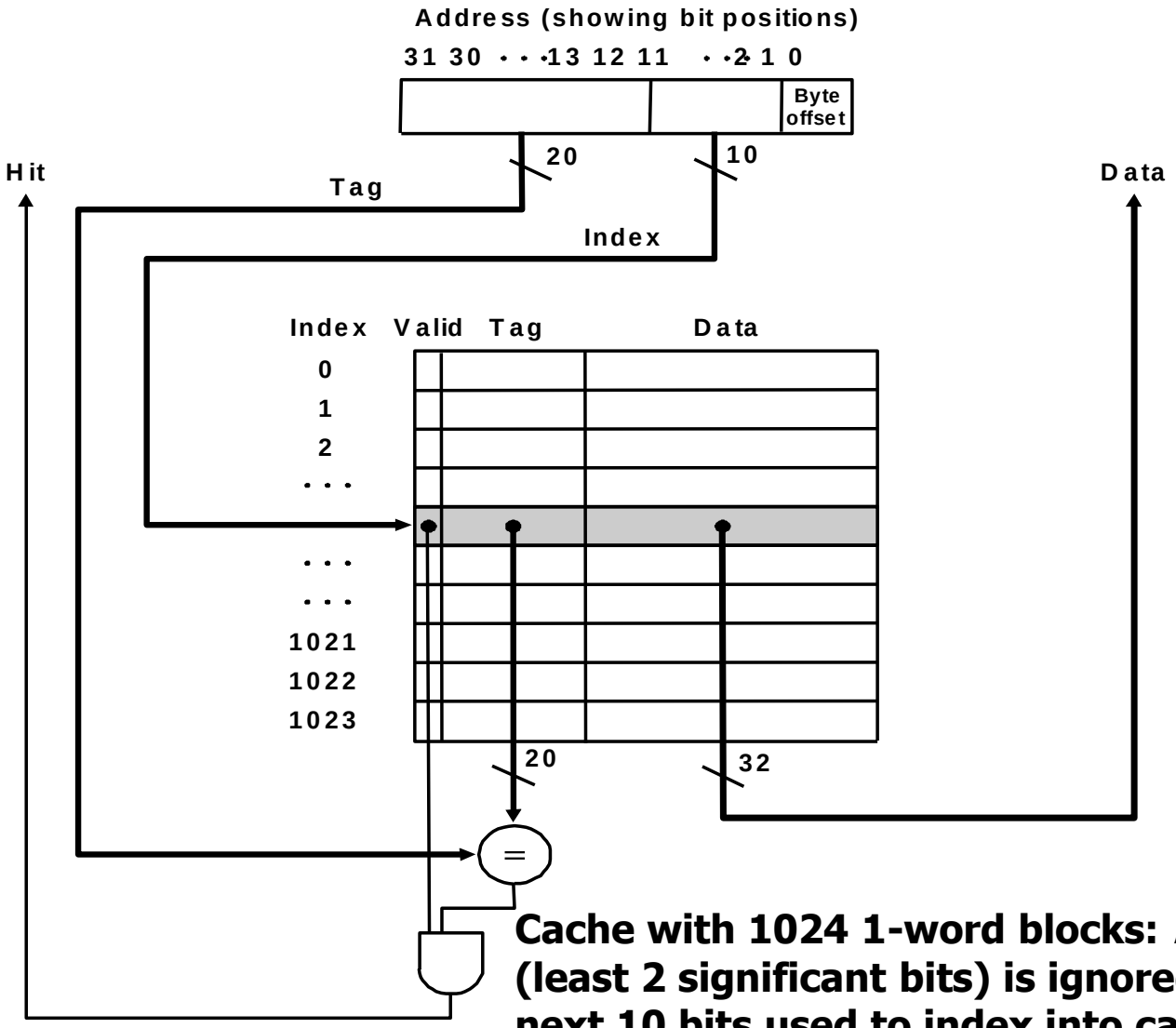
Exploiting Memory Hierarchy

Hit and Miss

- Focus on *any two adjacent* levels – called, *upper* (closer to CPU) and *lower* (farther from CPU) – in the memory hierarchy, because each block copy is always between two adjacent levels
- Terminology:
  - *block*: minimum unit of data to move between levels
  - *hit*: data requested is in upper level
  - *miss*: data requested is not in upper level
  - *hit rate*: fraction of memory accesses that are hits (i.e., found at upper level)
  - *miss rate*: fraction of memory accesses that are not hits
    - miss rate = 1 – hit rate
  - *hit time*: time to determine if the access is indeed a hit + time to access and deliver the data from the upper level to the CPU
  - *miss penalty*: time to determine if the access is a miss + time to replace block at upper level with corresponding block at lower level + time to deliver the block to the CPU

Direct Mapped Cache

- MIPS style:

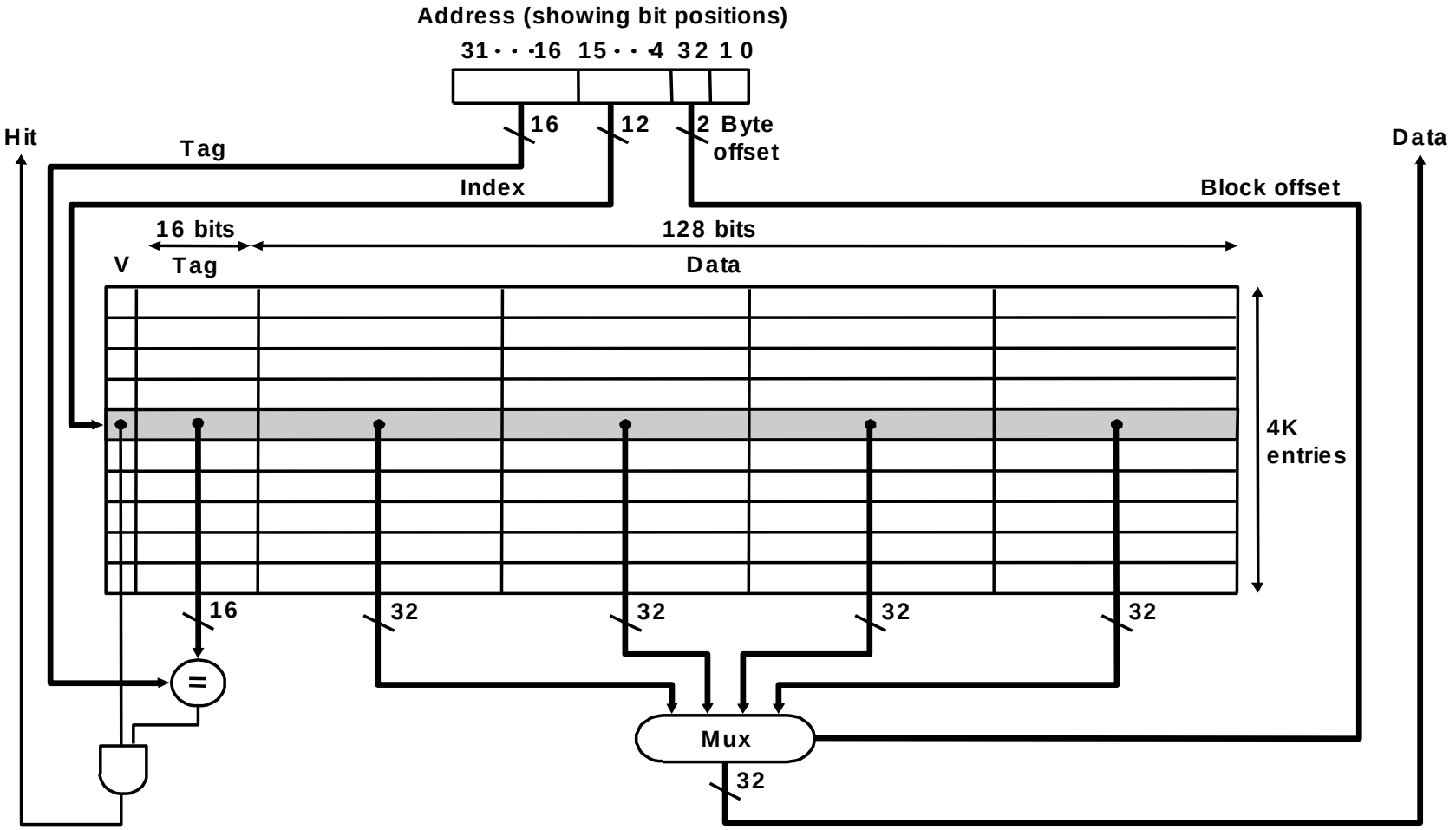


Cache with 1024 1-word blocks: *byte offset* (least 2 significant bits) is ignored and next 10 bits used to index into cache

What kind of locality are we taking advantage of?

Direct Mapped Cache: Taking Advantage of Spatial Locality

- Taking advantage of spatial locality with *larger* blocks:

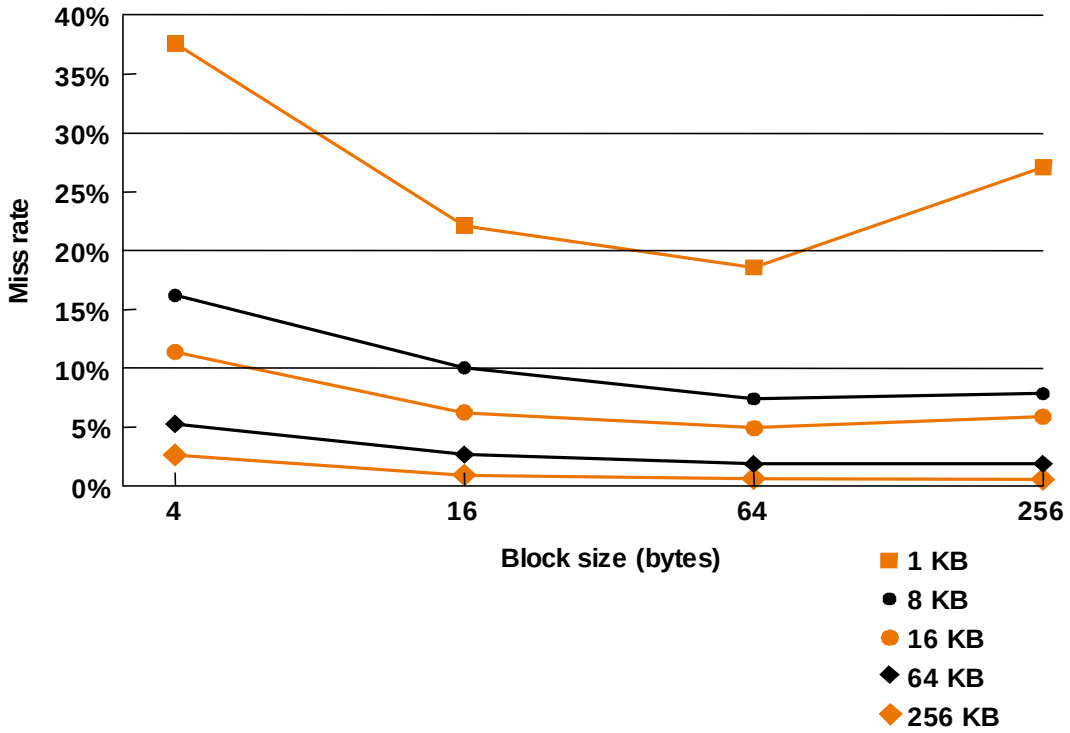


Cache with 4K 4-word blocks: *byte offset* (least 2 significant bits) is ignored, next 2 bits are *block offset*, and the next 12 bits are used to index into cache

Direct Mapped Cache: Taking Advantage of Spatial Locality

- Miss rate falls at first with increasing block size as expected, but, as block size becomes a large fraction of total cache size, miss rate may go up because
  - there are few blocks
  - competition for blocks increases
  - blocks get ejected before most of their words are accessed (*thrashing* in cache)

Miss rate vs. block size for various cache sizes



Example

- *How many total bits are required for a direct-mapped cache with 128 KB of data and 1-word block size, assuming a 32-bit address?*

Example

- *How many total bits are required for a direct-mapped cache with 128 KB of data and 1-word block size, assuming a 32-bit address?*

- Cache data = 128 KB = 2^{17} bytes = 2^{15} words = 2^{15} blocks
- Cache entry size = block data bits + tag bits + valid bit

  = 32 + (32 – 15 – 2) + 1 = 48 bits
- Therefore, cache size = $2^{15} \times 48$ bits = $2^{15} \times (1.5 \times 32)$ bits = 1.5×2^{20} bits = 1.5 Mbits
  - data bits in cache = 128 KB × 8 = 1 Mbits
  - total cache size/actual cache data = 1.5

Example Problem

- *How many total bits are required for a direct-mapped cache with 128 KB of data and 4-word block size, assuming a 32-bit address?*

- Cache size = 128 KB = 2^{17} bytes = 2^{15} words = 2^{13} blocks
- Cache entry size = block data bits + tag bits + valid bit
  = 128 + (32 – 13 – 2 – 2) + 1 = 144 bits
- Therefore, cache size = 2^{13} × 144 bits =
  2^{13} × (1.25 × 128) bits = 1.25 × 2^{20} bits = 1.25 Mbits
  - data bits in cache = 128 KB × 8 = 1 Mbits
  - total cache size/actual cache data = 1.25

Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks $\Rightarrow$ fewer of them
    - More competition $\Rightarrow$ increased miss rate
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help