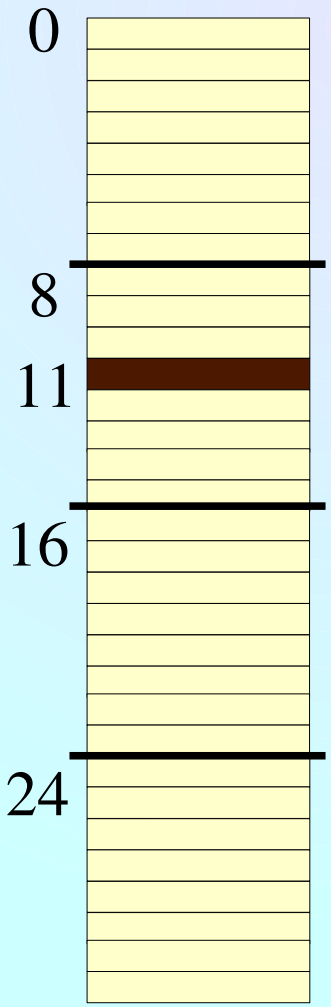
Memory Hierarchy

- Two principles:
  - Smaller is faster
  - Principle of locality
- Processor speed grows much faster than memory speed
- Registers – Cache – Memory – Disk
  - Upper level vs. lower level
- Cache design

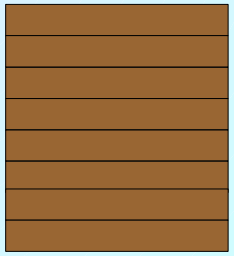
Cache Design Questions

- Cache is arranged in terms of blocks
  - To take advantage of spatial locality
- Design choices:
  - **Q1: block placement** –w here to place a block in upper level?
  - **Q2: block identification** – how to find a block in upper level?
  - **Q3: block replacement** –which bl ock to replace on a miss?
  - **Q4: write strategy** –wh at happens on a write?

Block Placement: Fully Associative



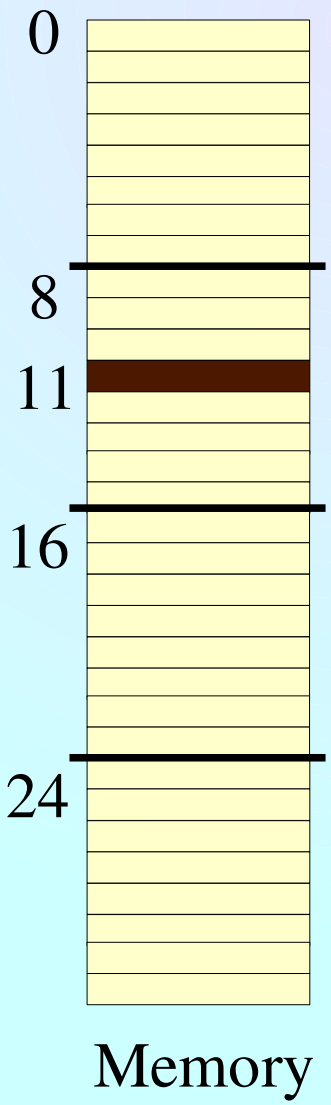
Memory



Cache

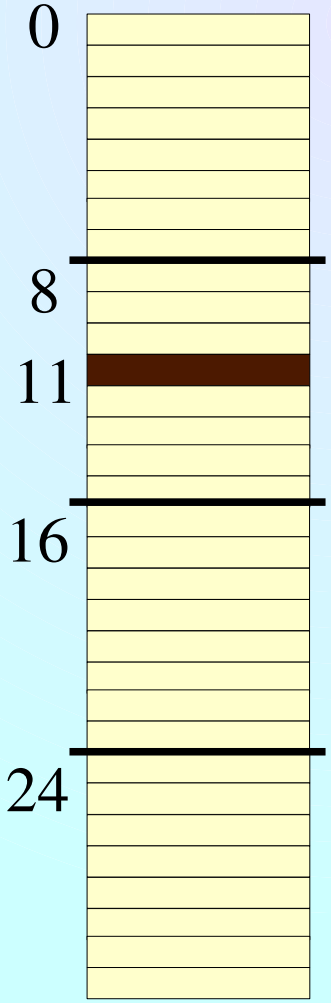
Block 11 can go anywhere

Block Placement: Direct

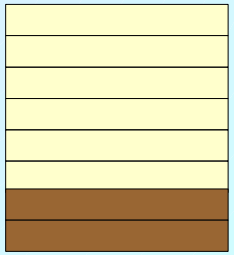


Block 11 can go only in block number 11 mod 8

Block Placement: Set Associative



Memory



Cache

Block 11 can go in set number 11 mod 4

Continuum of Choices

- Memory has *n* blocks, cache has *m* blocks
- Fully associative is the same as set associative with one set (*m*-way set associative)
- Direct placement is the same as 1-way set associative (with *m* sets)
- Most processors use direct, 2-way/4-way set associative

Block Identification

- How many different blocks of memory can be mapped (at different times) to a cache block?
- Fully associative: *n*
- Direct: *n/m*
- k-way set associative: *k*n/m*
- Each cache block has a tag saying which block of memory is currently present in it
  - A valid bit is set to 0 if no memory block is in the cache block currently

Block Identification (continued)

- How many bits for the tag?

  $\log_2(k*n/m)$

- How many sets in cache?

  m/k

- How many bits to identify the correct set?

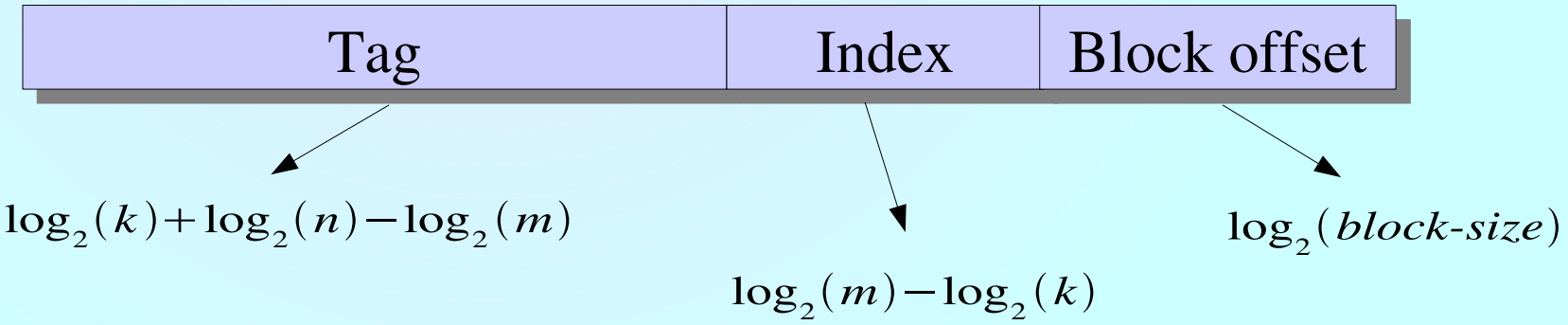
  $\log_2(m/k)$

Block Identification (continued)

- How many blocks in memory?

  n , $\log_2(n)$ *to represent block number in memory*

- Given a memory address:

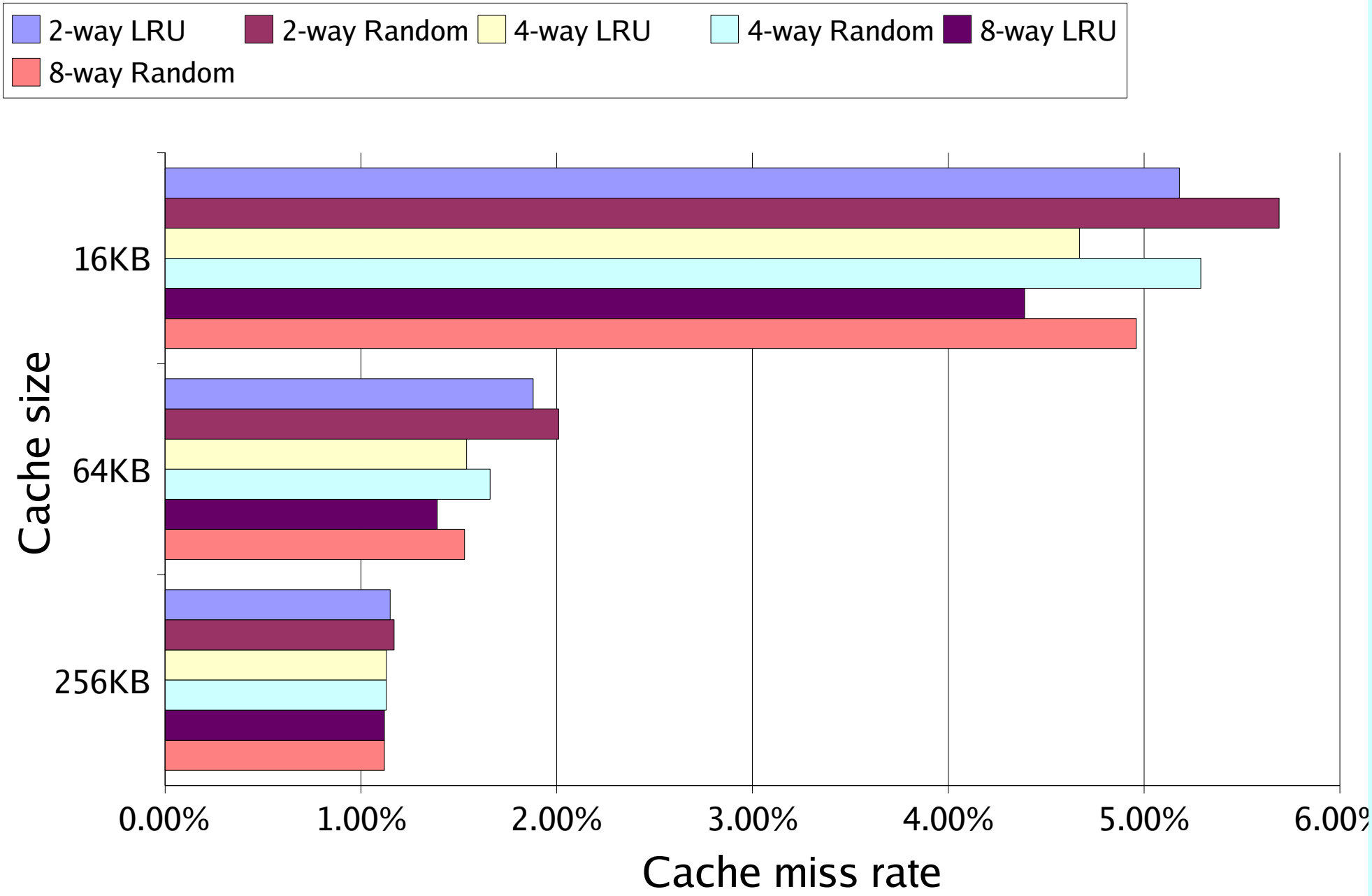


  - Select set using index, block from set using tag
  - Select location from block using block offset
  - tag + index = block address

Block Replacement Policy

- Cache miss ==> bring block onto cache
  - What if no free block in set?
  - Need to replace a block
- Possible policies:
  - Random
  - Least-Recently Used (LRU)
    - Lesser miss-rate, but harder to implement

Replacement Policy Performance



Write Strategy

- Reads are dominant
  - All instructions are read
  - Even for data, loads dominate over stores
- Reads can be fast
  - Can read from multiple blocks while performing tag comparison
  - Cannot do the same with writes
- Should pay attention to write performance too!

When do Writes go to Memory?

- **Write through:** each write is mirrored to memory also
  - Easier to implement
- **Write back:** write to memory only when block is replaced
  - Faster writes
  - Some writes do not go to memory at all!
  - But, read miss may cause more delay
    - Block being replaced has to be written back
    - Optimize using dirty bit
  - Also, bad for multiprocessors and I/O

Write Stalls

- In write through, may have to stall waiting for write to complete
  - Called a **write stall**
  - Can employ a **write buffer** to enable the processor to proceed during the write-through

What to do on a Write Miss?

- **Write-allocate** (or, fetch on write): load block on a cache miss during a write
- **No-write allocate** (or, write around): just write directly to main memory
- Write-allocate usually goes with write-back, and no-write allocate goes with write-through

The Alpha AXP 21064 Cache

- 34-bit physical address
  - 29 bits for block address
  - 5 bits for block offset
- 8 KB cache, direct-mapped
  - 8 bits for index
  - 29 – 8 = 21 bits for tag

Steps in Memory Read

- Four steps:
  - Step-1: CPU puts out the address
  - Step-2: Index selection
  - Step-3: Tag comparison, read from data
  - Step-4: Data returned to CPU (assuming hit)
- This takes two cycles

Steps in Memory Write

- Write-through policy is used
- Write buffer with four entries
  - Each entry can have up to 4 words from the same block
  - Write merging: successive writes to the same block use the same write-buffer entry

Some More Details

- What happens on a miss?
  - Cache sends signal to CPU asking it to wait
  - No replacement policy required (direct mapped)
  - Write miss ==> write-around
- 8KB separate instruction cache

Separate versus Unified Cache

- Direct-mapped cache, 32-byte blocks, SPEC92, on DECstation 5000
- Unified cache has twice the size of I-cache or D-cache
- 75% instruction references

	I-Cache	D-Cache	U-Cache
1KB	3.06%	24.61%	13.34%
2KB	2.26%	20.57%	9.78%
4KB	1.78%	15.94%	7.24%
8KB	1.10%	10.19%	4.57%
16KB	0.64%	6.47%	2.87%
32KB	0.39%	4.82%	1.99%
64KB	0.15%	3.77%	1.35%
128KB	0.02%	2.88%	0.95%

Miss-rates