CPS104
Computer Organization and Programming
Lecture 18: Cache Memory

Nov. 1, 1999

Dietolf (Dee) Ramm

http://www.cs.duke.edu/~dr/cps104.html

©GK&DR Fall 1999

Outline of Today's Lecture

- Direct Mapped Cache (review).
- Two-Way Set Associative Cache
- Fully Associative cache
- Replacement Policies
- Write Strategies

©GK&DR Fall 1999

Direct Mapped Cache

For a Cache of 2^M bytes with block size of 2^L bytes

- There are 2^{M-L} cache blocks,
- Lowest L bits of the address are Block-Offset bits
- Next (M - L) bits are the Cache-Index.
- The last (32 - M) bits are the Tag bits.

Data Address

Cache-Index = (<Address> Mod (Cache_Size))/ Block_Size

Block-Offset = <Address> Mod (Block_Size)

Tag = <Address> / (Cache_Size)

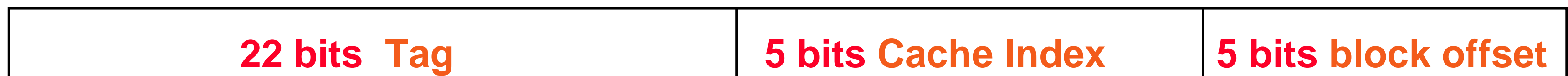
©GK&DR Fall 1999

Example: 1-KB Cache with 32B blocks:

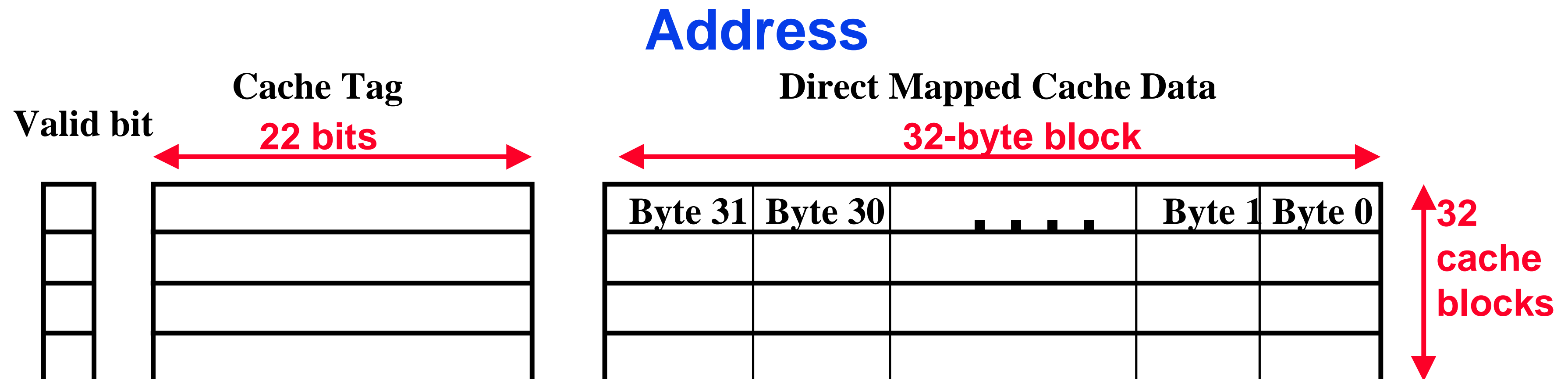
Cache Index = (<Address> Mod (1024))/ 32

Block-Offset = <Address> Mod (32)

Tag = <Address> / (1024)



Address



$1K = 2^{10} = 1024$

$2^5 = 32$

©GK&DR Fall 1999

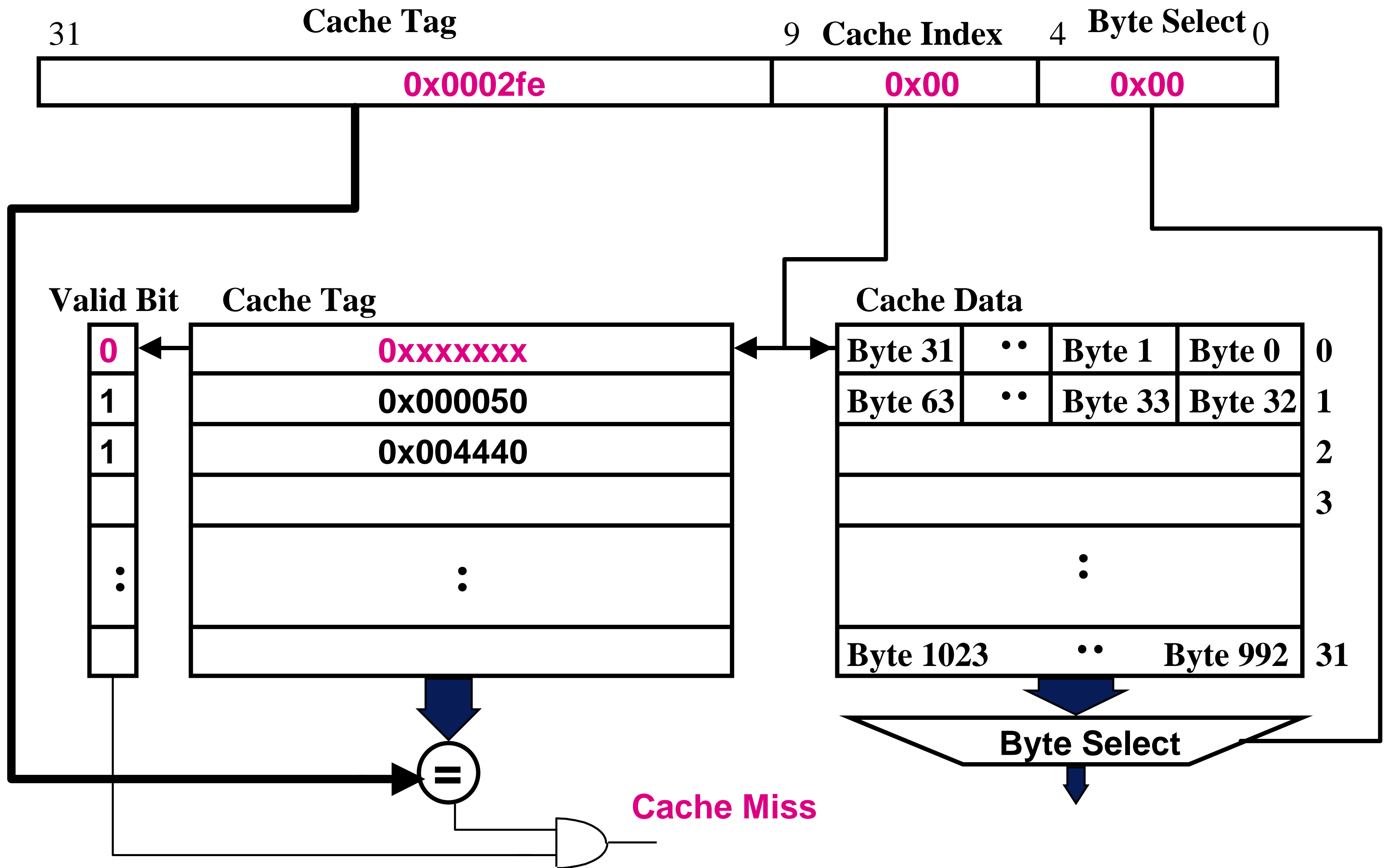
Example: 1KB Direct Mapped Cache with 32B Blocks

° For a 1024 (2^{10}) byte cache with 32-byte blocks:

- The uppermost 22 = (32 - 10) address bits are the Cache Tag
- The lowest 5 address bits are the Byte Select (Block Size = 2^5)
- The next 5 address bits (bit5 - bit9) are the Cache Index

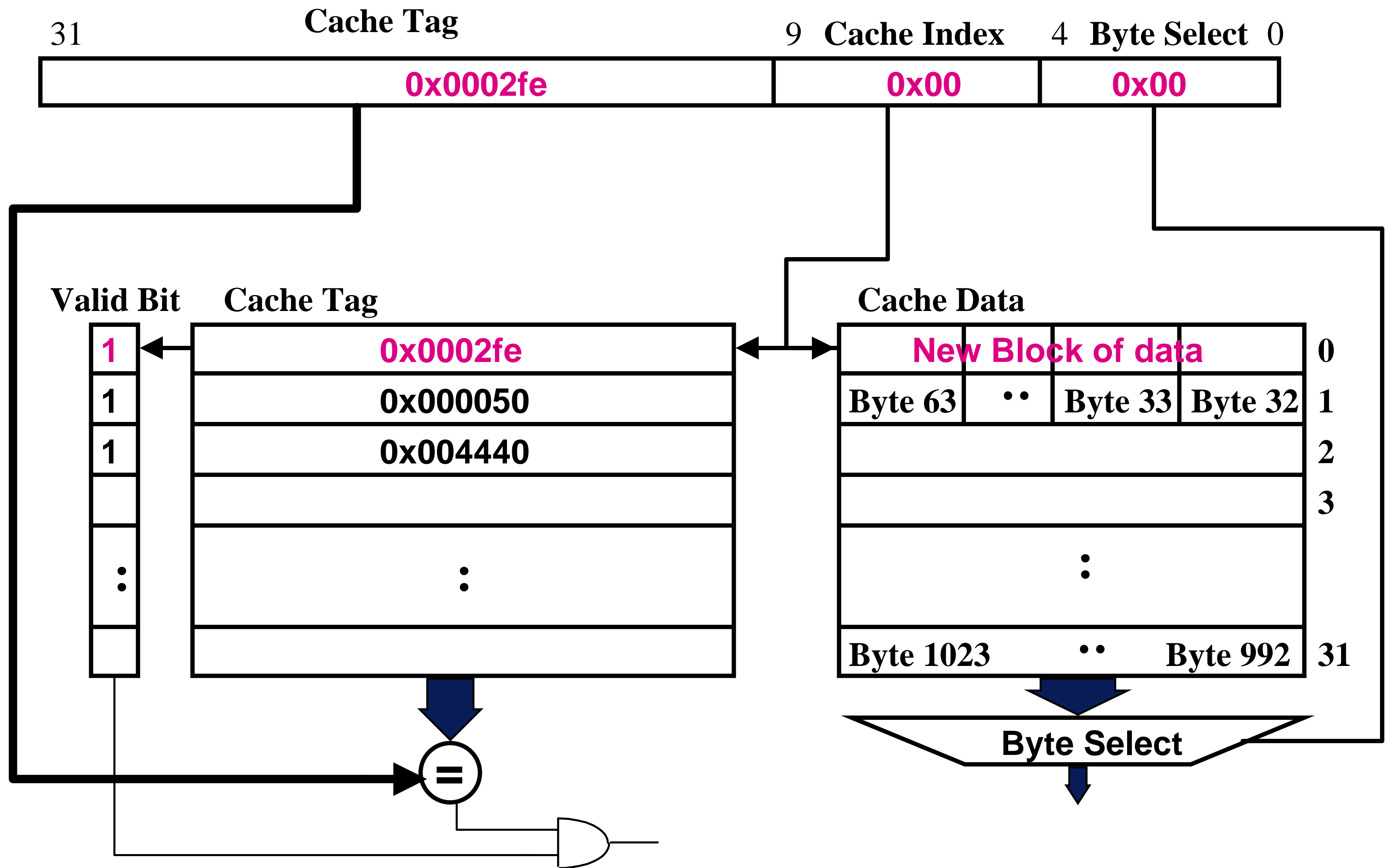
©GK&DR Fall 1999

Example: 1K Direct Mapped Cache



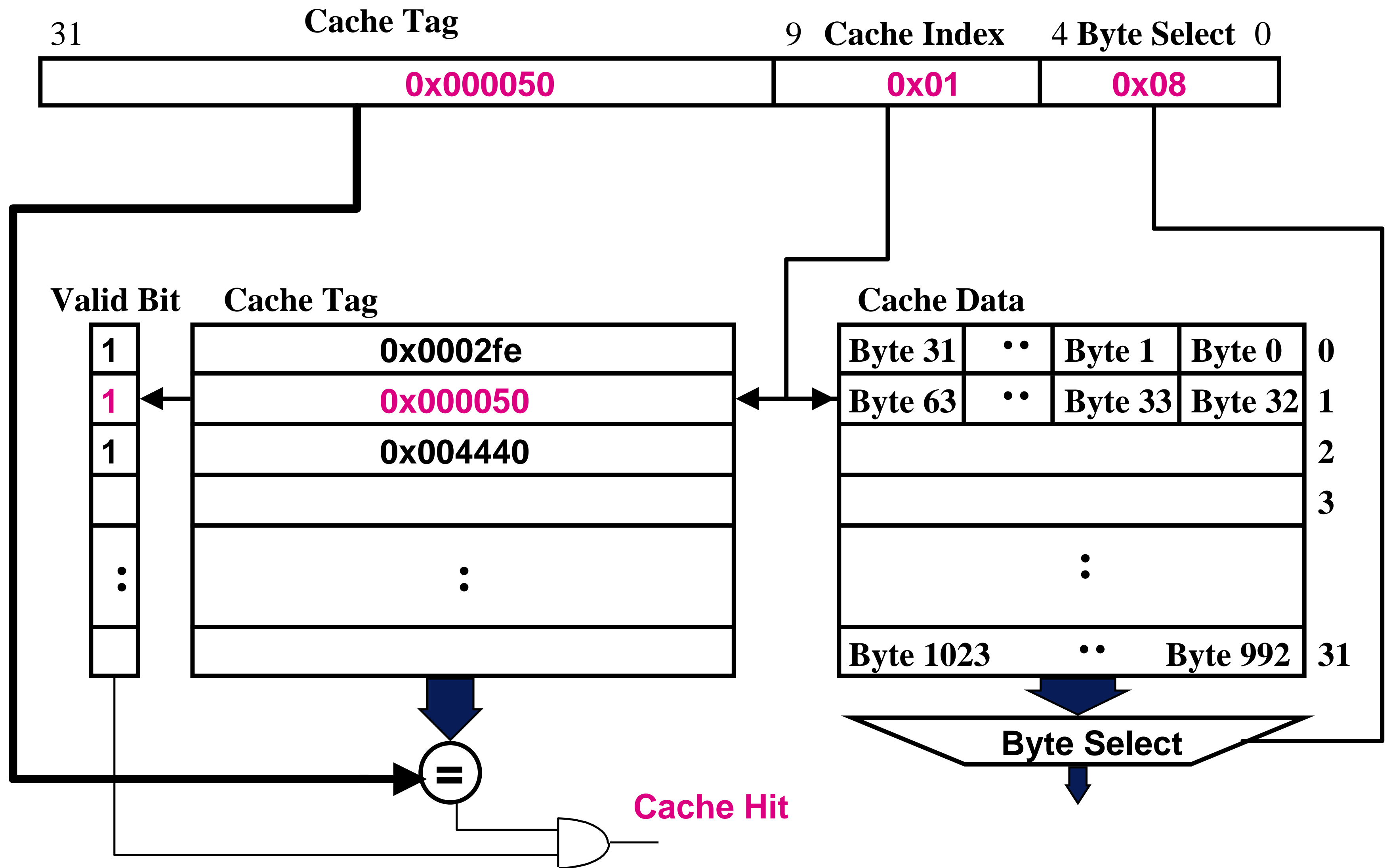
©GK&DR Fall 1999

Example: 1K Direct Mapped Cache



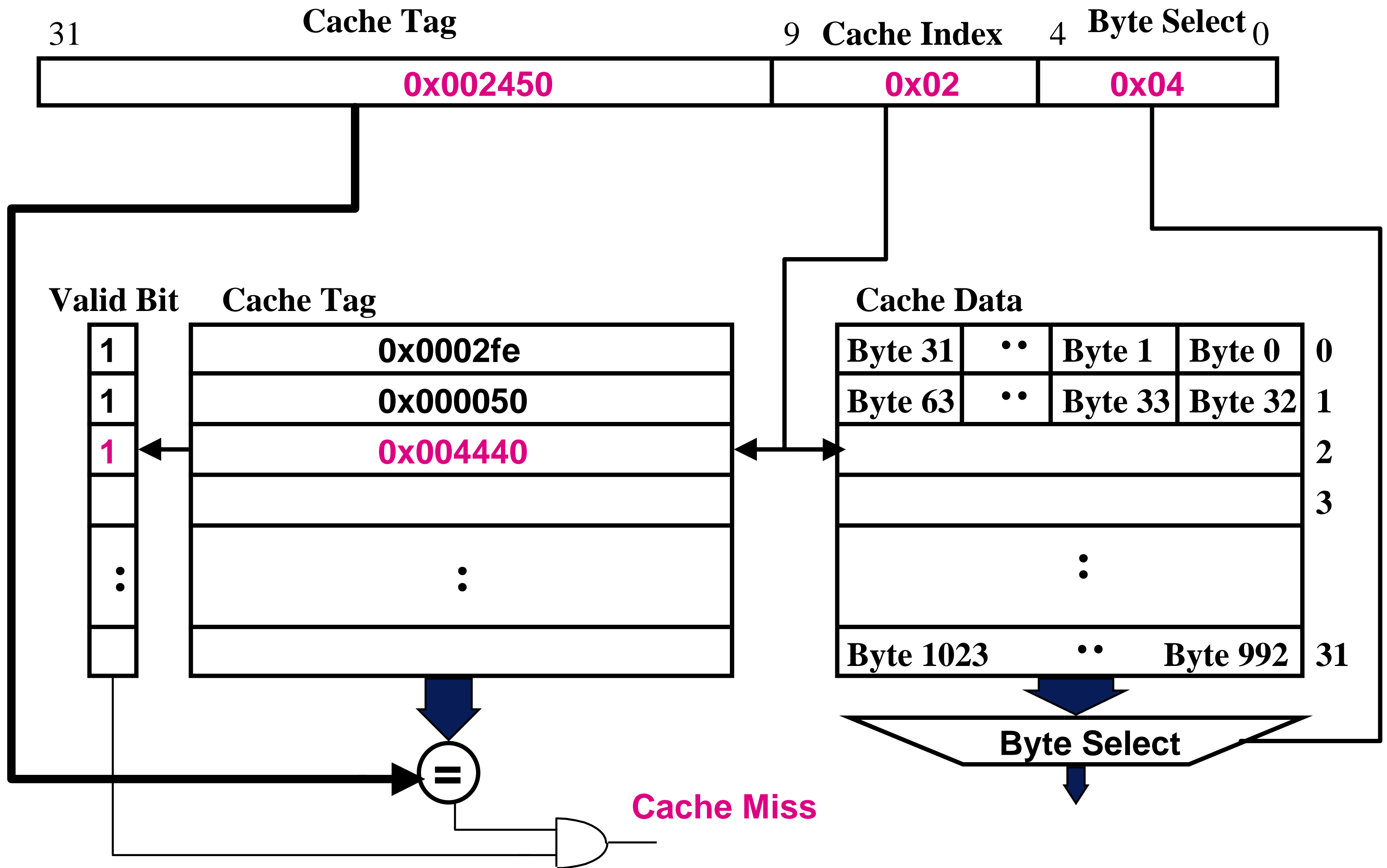
©GK&DR Fall 1999

Example: 1K Direct Mapped Cache



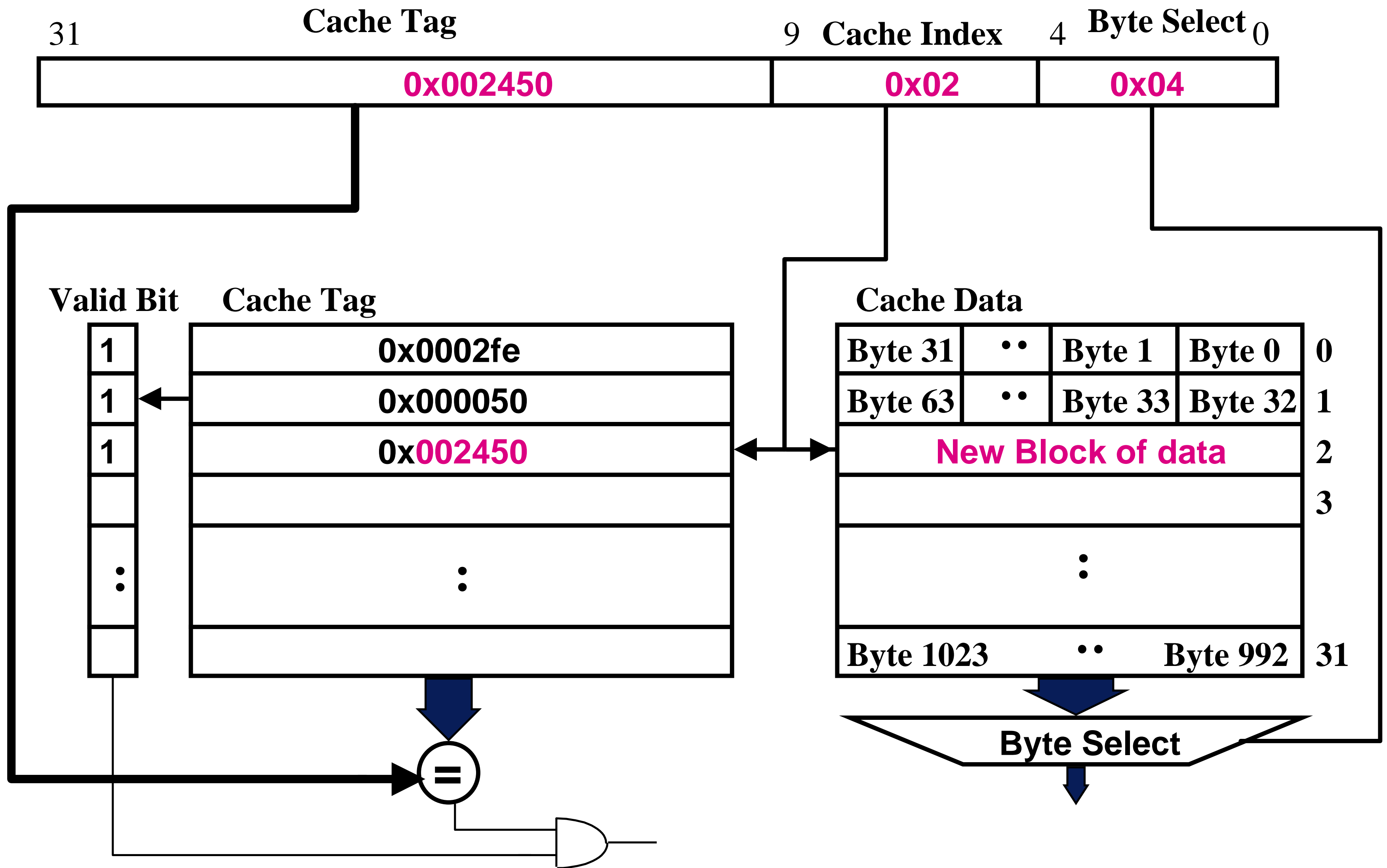
©GK&DR Fall 1999

Example: 1K Direct Mapped Cache



©GK&DR Fall 1999

Example: 1K Direct Mapped Cache

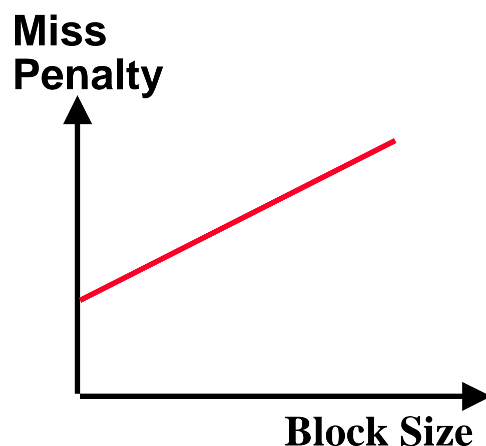


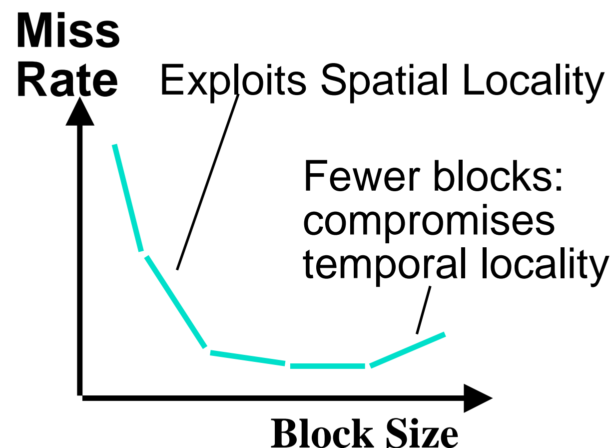
©GK&DR Fall 1999

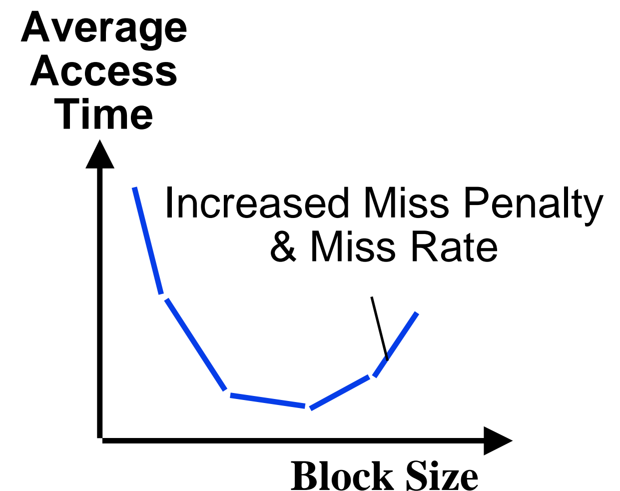
Block Size Tradeoff

° 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 general, Average Access Time:
  • Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate





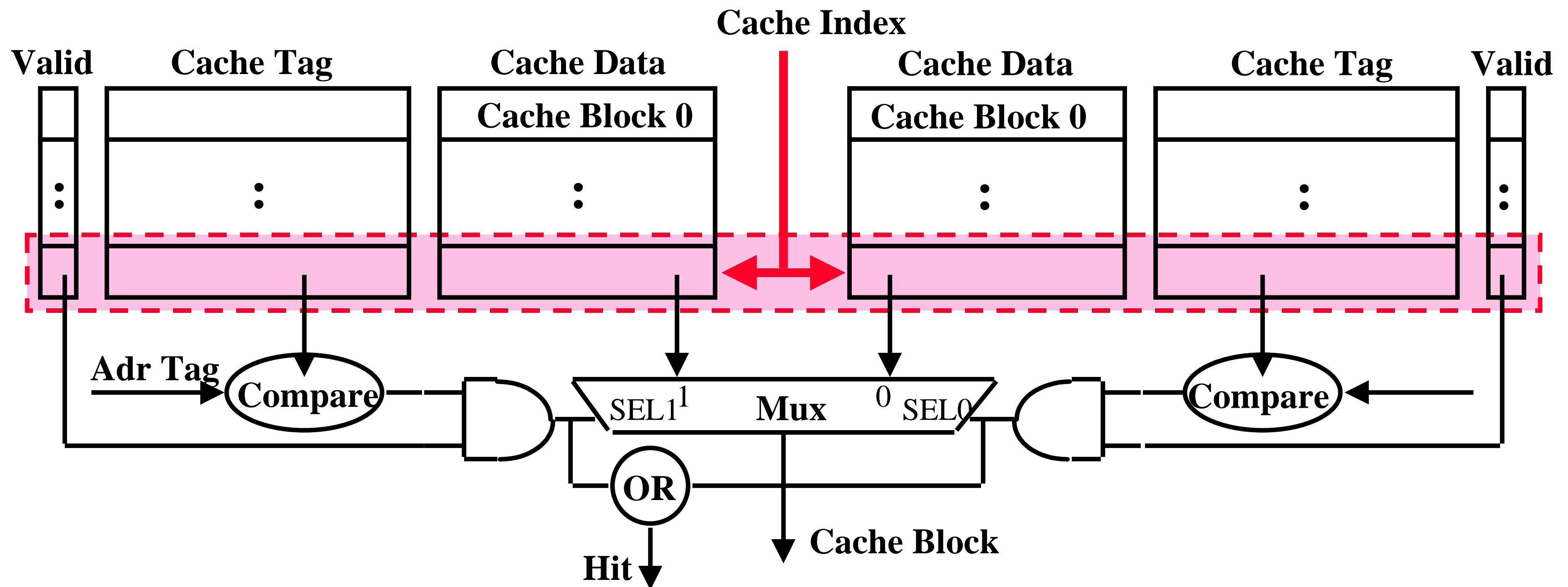


©GK&DR Fall 1999

A N-way Set Associative Cache

° N-way set associative: N entries for each Cache Index
  - N direct mapped caches operating in parallel

° Example: Two-way set associative cache
  - Cache Index selects a "set" from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result



©GK&DR Fall 1999

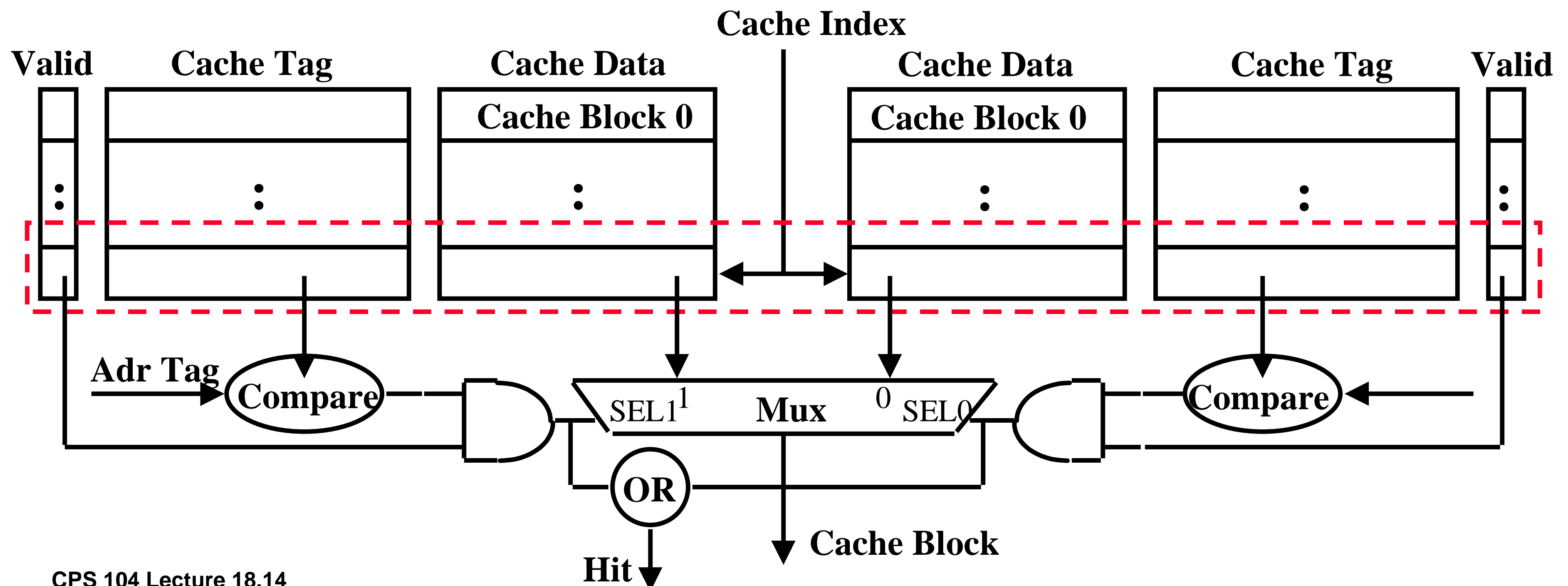
Advantages of Set associative cache

- Higher Hit rate for the same cache size.
- Fewer Conflict Misses.
- Can can have a larger cache
  but keep the index smaller
  (same size as virtual page index)

©GK&DR Fall 1999

Disadvantage of Set Associative Cache

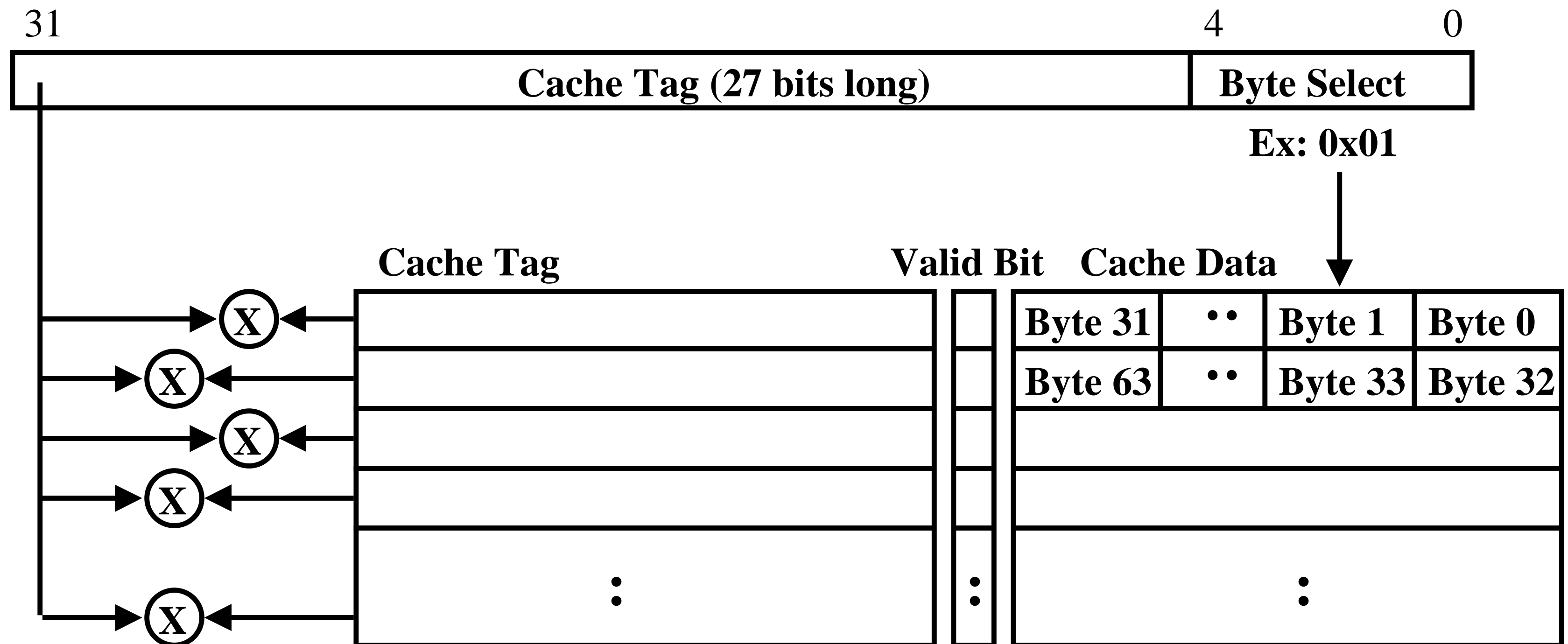
- N-way Set Associative Cache versus Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss decision and set selection
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.



©GK&DR Fall 1999

And yet Another Extreme Example:
Fully Associative cache

- **Fully Associative Cache** -- push the set associative idea to its limit!
  - Forget about the Cache Index
  - Compare the Cache Tags of all cache entries in parallel
  - Example: Block Size = 32B blocks, we need N 27-bit comparators
- By definition: Conflict Miss = 0 for a fully associative cache



©GK&DR Fall 1999

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

° **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

©GK&DR Fall 1999

Sources of Cache Misses

	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(er)	Medium	High
Invalidation Miss	Same	Same	Same

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

©GK&DR Fall 1999

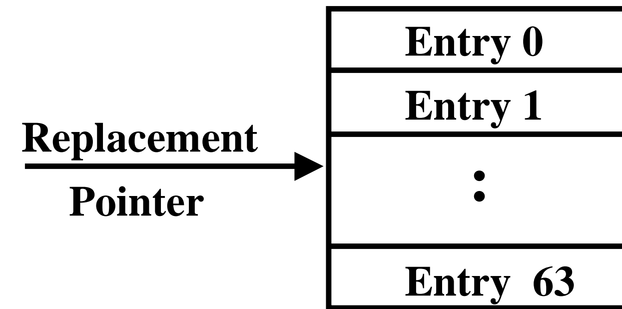
The Need to Make a Decision!

- **Direct Mapped Cache:**
  - Each memory location can only mapped to 1 cache location
  - No need to make any decision :-)
    - Current item replaces the previous item in that cache location
- **N-way Set Associative Cache:**
  - Each memory location have a choice of N cache locations
- **Fully Associative Cache:**
  - Each memory location can be placed in ANY cache location
- Cache miss in a N-way Set Associative or Fully Associative Cache:
  - Bring in new block from memory
  - Throw out a cache block to make room for the new block
  - We need to make a decision on which block to throw out!

©GK&DR Fall 1999

Cache Block Replacement Policy

- **Random Replacement:**
  - Hardware randomly selects a cache block out of the set and replaces it.
- **Least Recently Used:**
  - Hardware keeps track of the access history
  - Replace the entry that has not been used for the longest time.
  - For two way set associative cache one needs one bit for LRU replacement.
- **Example of a Simple "Pseudo" Least Recently Used Implementation:**
  - Assume 64 Fully Associative Entries
  - Hardware replacement pointer points to one cache entry
  - Whenever an access is made to the entry the pointer points to:
    - Move the pointer to the next entry
    - Otherwise: do not move the pointer



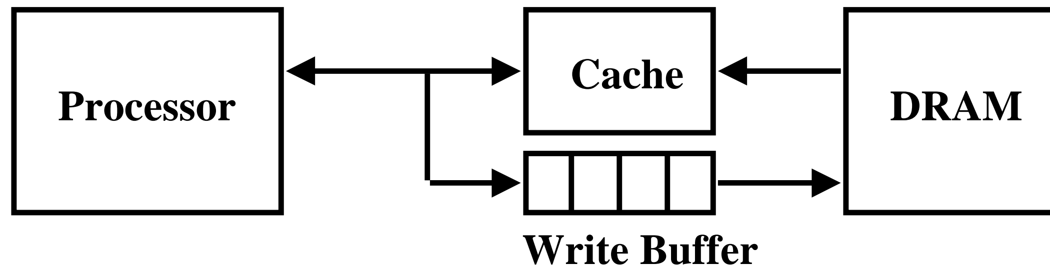
©GK&DR Fall 1999

Cache Write Policy: Write Through versus Write Back

- Cache read is much easier to handle than cache write:
  - Instruction cache is much easier to design than data cache
- Cache write:
  - How do we keep data in the cache and memory consistent?
- Two options (decision time again :-)
  - Write Back: write to cache only. Write the cache block to memory when that cache block is being replaced on a cache miss.
    - Need a "dirty bit" for each cache block
    - Greatly reduce the memory bandwidth requirement
    - Control can be complex
  - Write Through: write to cache and memory at the same time.
    - What!!! How can this be? Isn't memory too slow for this?

©GK&DR Fall 1999

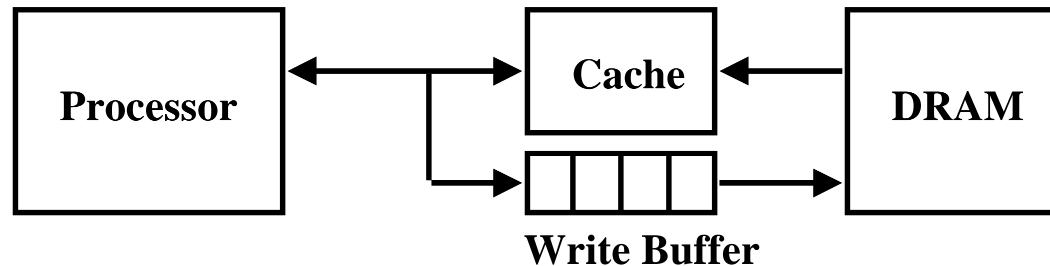
Write Buffer for Write Through



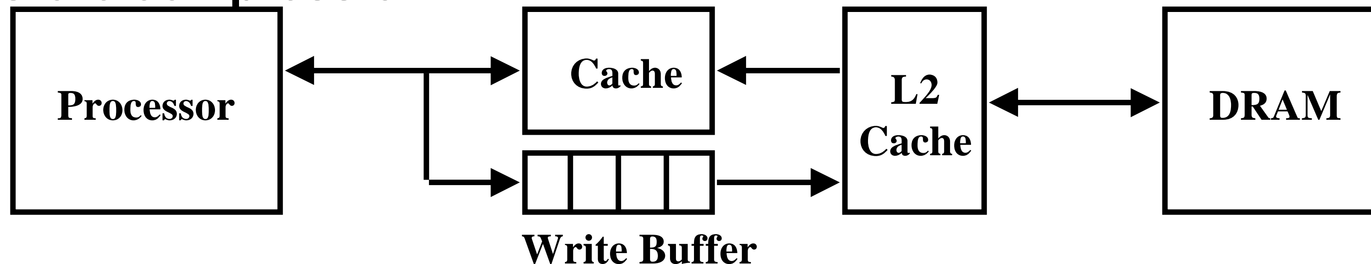
- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer's nightmare:
  - Store frequency (w.r.t. time) > 1 / DRAM write cycle
  - Write buffer saturation

©GK&DR Fall 1999

Write Buffer Saturation



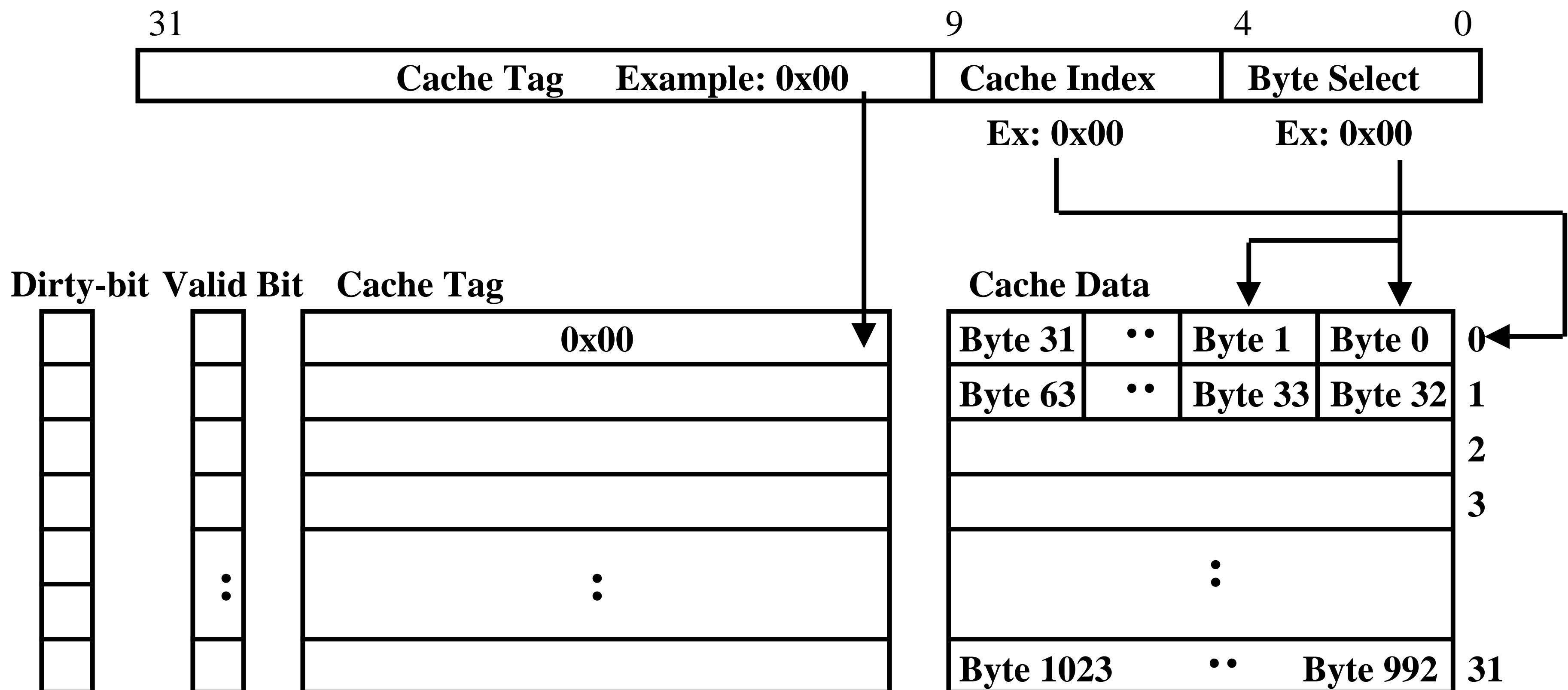
- Store frequency (w.r.t. time) > 1 / DRAM write cycle
  - If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
    - Store buffer will overflow no matter how big you make it
    - The CPU Cycle Time << DRAM Write Cycle Time
- Solution for write buffer saturation:
  - Use a write back cache
  - Install a second level (L2) cache:
  - store compression



©GK&DR Fall 1999

Write Allocate versus Not Allocate

° Assume: a 16-bit write to memory location 0x0 and causes a miss
  - Do we read in the block?
    - Yes: Write Allocate
    - No: Write Not Allocate



©GK&DR Fall 1999

Four Questions for Memory Hierarchy Designers

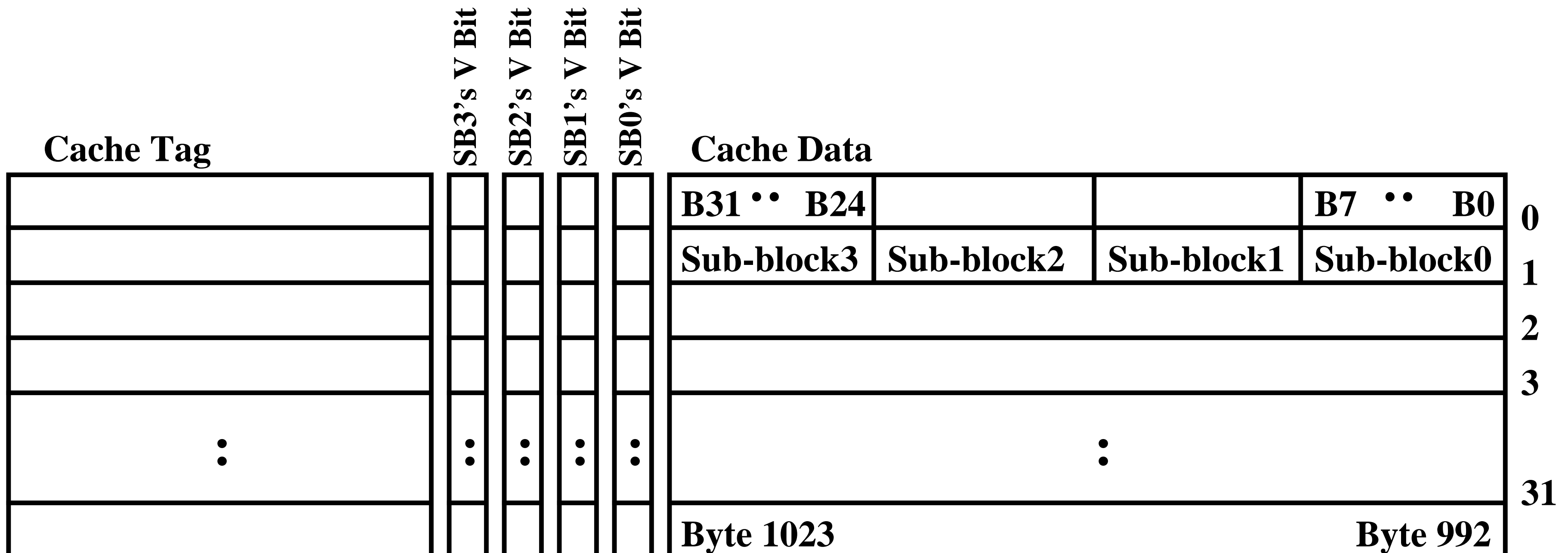
- **Q1:** **Where can a block be placed in the upper level?**
  (Block placement)
- **Q2:** **How is a block found if it is in the upper level?**
  (Block identification)
- **Q3:** **Which block should be replaced on a miss?**
  (Block replacement)
- **Q4:** **What happens on a write?**
  (Write strategy)

©GK&DR Fall 1999

What is a Sub-block?

- Sub-block:
  - Share one cache tag between all sub-blocks in a block
  - A unit within a block that has its own valid bit
  - Example: 1 KB Direct Mapped Cache, 32-B Block, 8-B Sub-block
    - Each cache entry will have: 32/8 = 4 valid bits

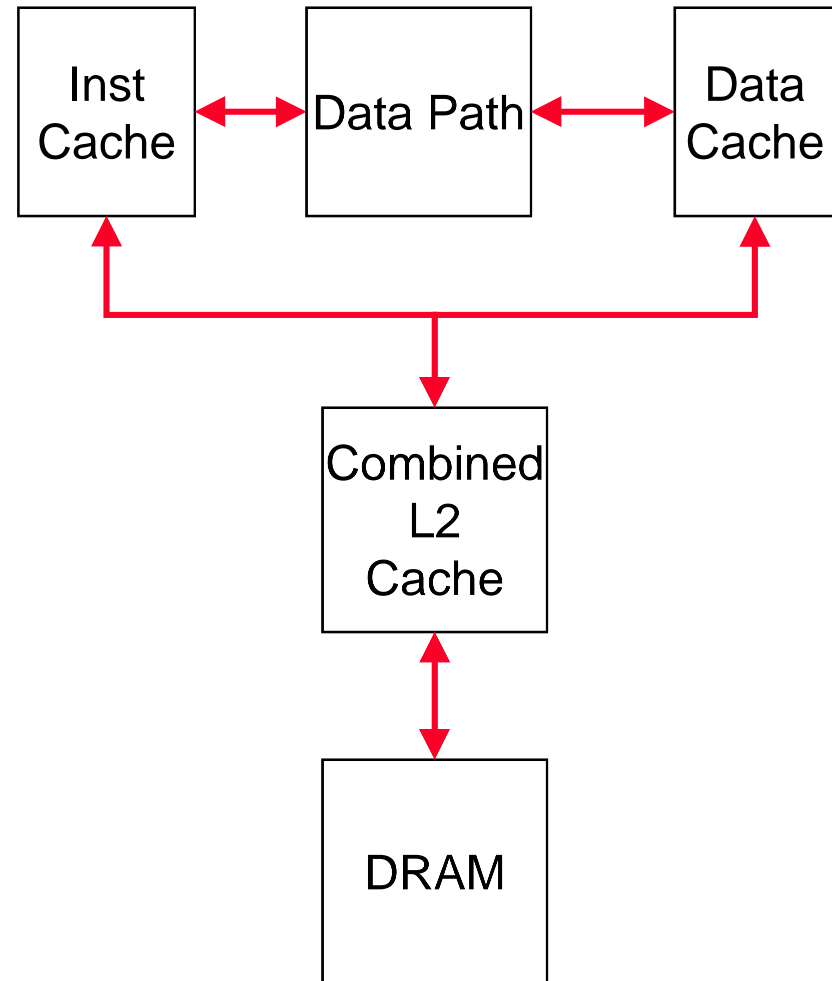
- Write miss: only the bytes in that sub-block is brought in.
  - reduce cache fill bandwidth (penalty).



©GK&DR Fall 1999

Separate Instruction and Data Caches

- Separate Inst & Data Caches
  - Harvard Architecture
- Can access both at same time
- Combined L2
  - L2 >> L1



©GK&DR Fall 1999

Cache Performance

CPU time = (CPU_execution_clock_cycles + Memory_stall_clock_cycles) x clock_cycle_time

Memory_stall_clock_cycles = Memory_accesses x Miss_rate x Miss_penalty

Example

- Assume every instruction takes 1 cycle
- Miss penalty = 20 cycles
- Miss rate = 10%
- 1000 total instructions, 300 memory accesses
- Memory stall cycles? CPU clocks?

©GK&DR Fall 1999

Cache Performance

- Memory Stall cycles = 300 * 0.10 * 20 = 600
- CPUclocks = 1000 + 600 = 1600

- 60% slower because of cache misses!

©GK&DR Fall 1999

Improving Cache Performance

1. Reduce the miss rate,

2. Reduce the miss penalty, or

3. Reduce the time to hit in the cache.

©GK&DR Fall 1999

Reducing Misses

° **Classifying Misses: 3 Cs**

- ***Compulsory*—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses*.**
  (Misses in Infinite Cache)
- ***Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.**
  (Misses in Size X Cache)
- ***Conflict*—If the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*.**
  (Misses in N-way Associative, Size X Cache)

©GK&DR Fall 1999

Cache Performance

- Your program and caches
- Can you affect performance?
- Think about 3Cs

©GK&DR Fall 1999

Reducing Misses by Compiler Optimizations

- Instructions
  - Reorder procedures in memory so as to reduce misses
  - Profiling to look at conflicts
  - McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks
- Data
  - ***Merging Arrays***: improve spatial locality by single array of compound elements vs. 2 arrays
  - ***Loop Interchange***: change nesting of loops to access data in order stored in memory
  - ***Loop Fusion***: Combine 2 independent loops that have same looping and some variables overlap
  - ***Blocking***: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows

©GK&DR Fall 1999

Merging Arrays Example

```
/* Before */
int val[SIZE];
int key[SIZE];

/* After */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

Reducing conflicts between val & key

©GK&DR Fall 1999

Loop Interchange Example

```
/* Before */
for (k = 0; k < 100; k = k+1)
  for (j = 0; j < 100; j = j+1)
        for (i = 0; i < 5000; i = i+1)
              x[i][j] = 2 * x[i][j];

/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
        for (j = 0; j < 100; j = j+1)
              x[i][j] = 2 * x[i][j];
```

Sequential accesses Instead of striding through memory every 100 words

©GK&DR Fall 1999

Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
        a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
        d[i][j] = a[i][j] + c[i][j];


/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {     a[i][j] = 1/b[i][j] * c[i][j];
        d[i][j] = a[i][j] + c[i][j];}
```

2 misses per access to a & c vs. one miss per access

©GK&DR Fall 1999

Blocking Example

```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
      {r = 0;
       for (k = 0; k < N; k = k+1)
            r = r + y[i][k]*z[k][j];
       x[i][j] = r;
      };
```

- Two Inner Loops:
  - Read all NxN elements of z[ ]
  - Read N elements of 1 row of y[ ] repeatedly
  - Write N elements of 1 row of x[ ]
- Capacity Misses a function of N & Cache Size:
  - 3 NxN => no capacity misses; otherwise ...
- Idea: compute on BxB submatrix that fits

©GK&DR Fall 1999

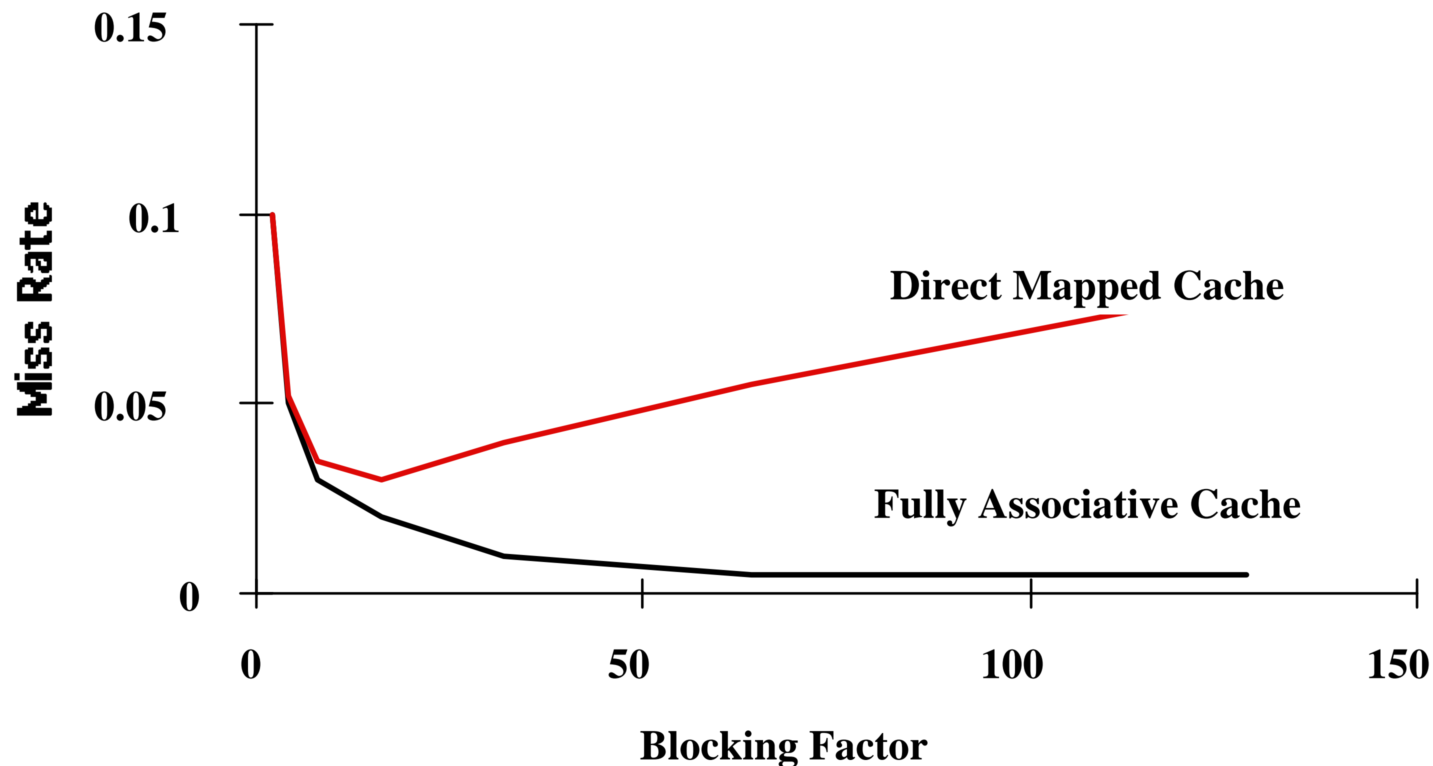
Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
    for (j = jj; j < min(jj+B-1,N); j = j+1)
      {r = 0;
       for (k = kk; k < min(kk+B-1,N); k = k+1) {
         r = r + y[i][k]*z[k][j];};
       x[i][j] = x[i][j] + r;
      };
```

- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- B called *Blocking Factor*
- Conflict Misses Too?

©GK&DR Fall 1999

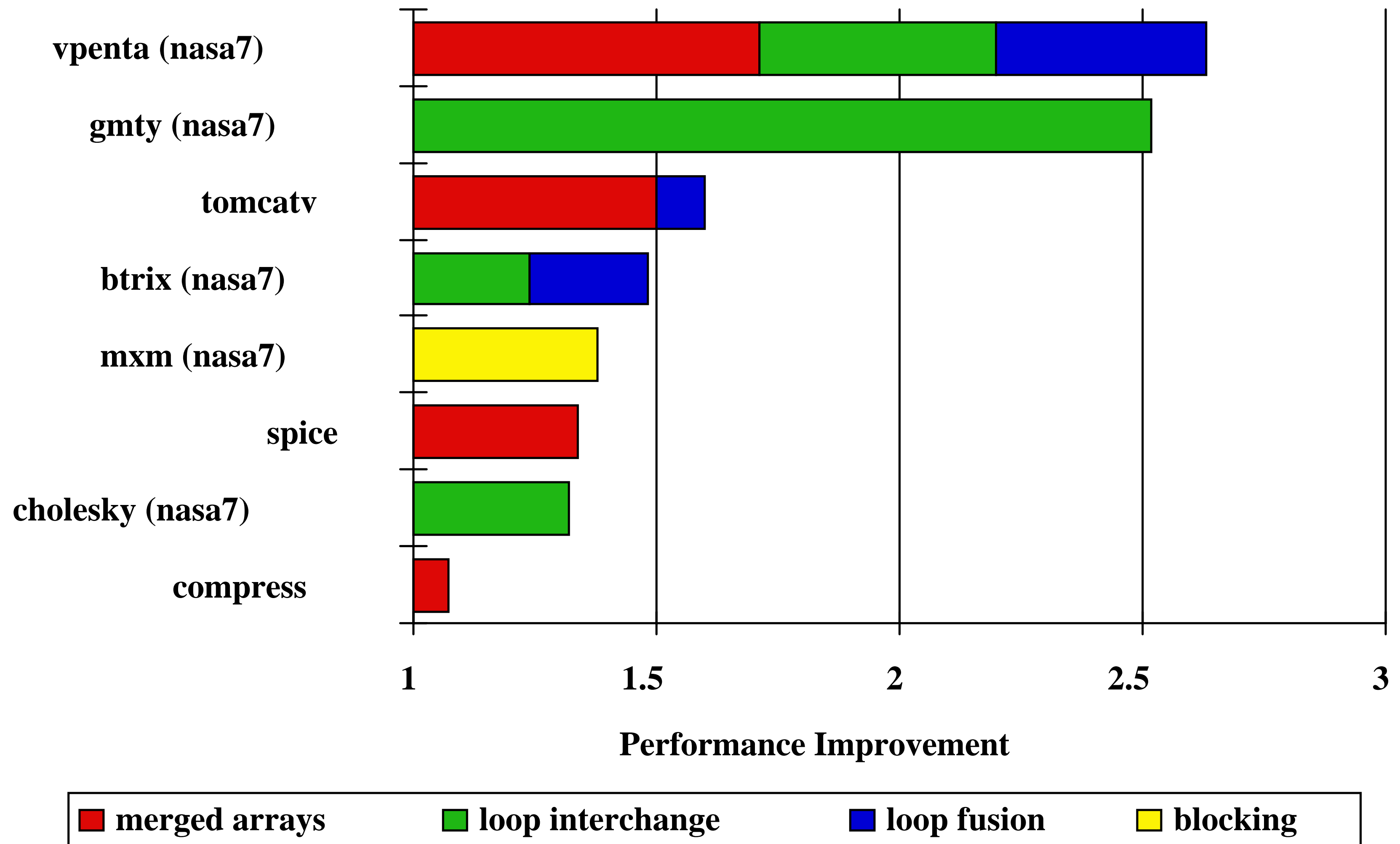
Reducing Conflict Misses by Blocking



- Conflict misses in caches not FA vs. Blocking size
  - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

©GK&DR Fall 1999

Summary of Compiler Optimizations to Reduce Cache Misses



©GK&DR Fall 1999

Summary

° Cost Effective Memory Hierarchy

° Split Instruction and Data Cache

° 4 Questions

° CPU cycles/time, Memory Stall Cycles

° Your programs and cache performance

<u>Next Time</u>

° Virtual Memory

©GK&DR Fall 1999