Chapter 5-1

Large and Fast: Exploiting Memory Hierarchy

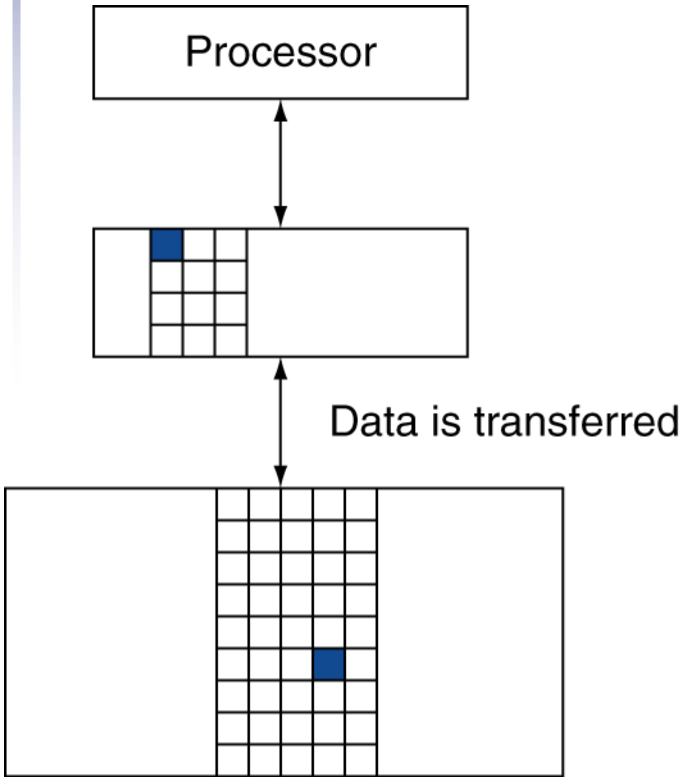
Principle of Locality

- Locality: Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - *e.g.*, instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - *e.g.*, sequential instruction access, array data

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Memory Hierarchy Levels



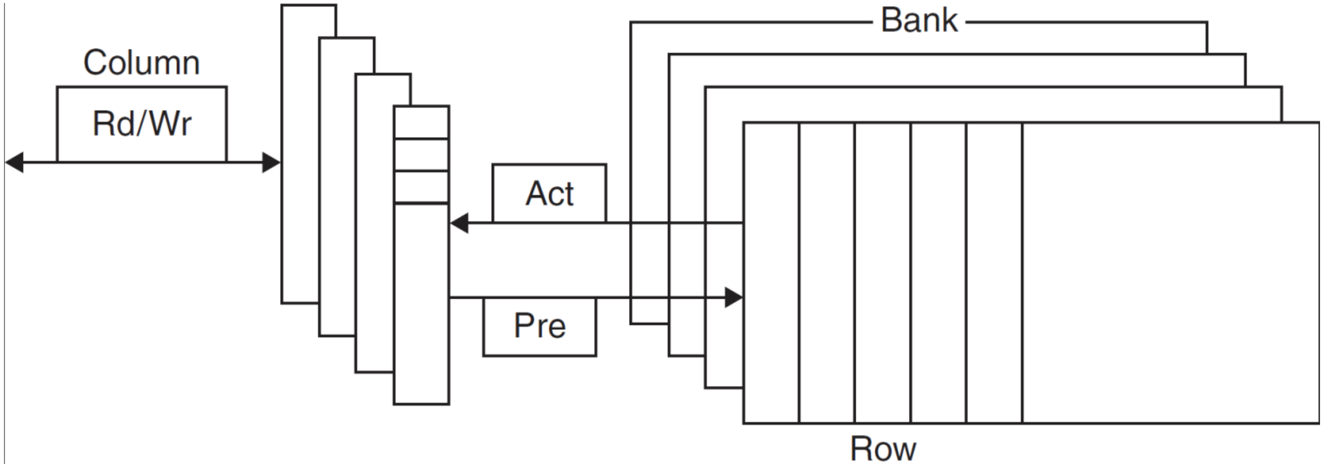
- Block (aka line): unit of copying
  - May be multiple words
- Hit, if accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- Miss, if accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      = 1 – hit ratio
  - Then accessed data supplied from upper level

Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $2000 – $5000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $20 – $75 per GB
- Magnetic disk
  - 5ms – 20ms, $0.20 – $2 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

DRAM Technology

- Data stored as a charge in a capacitor
  - Single transistor used to access the charge
  - Must periodically be refreshed
    - Read contents and write back
    - Performed on a DRAM "row"

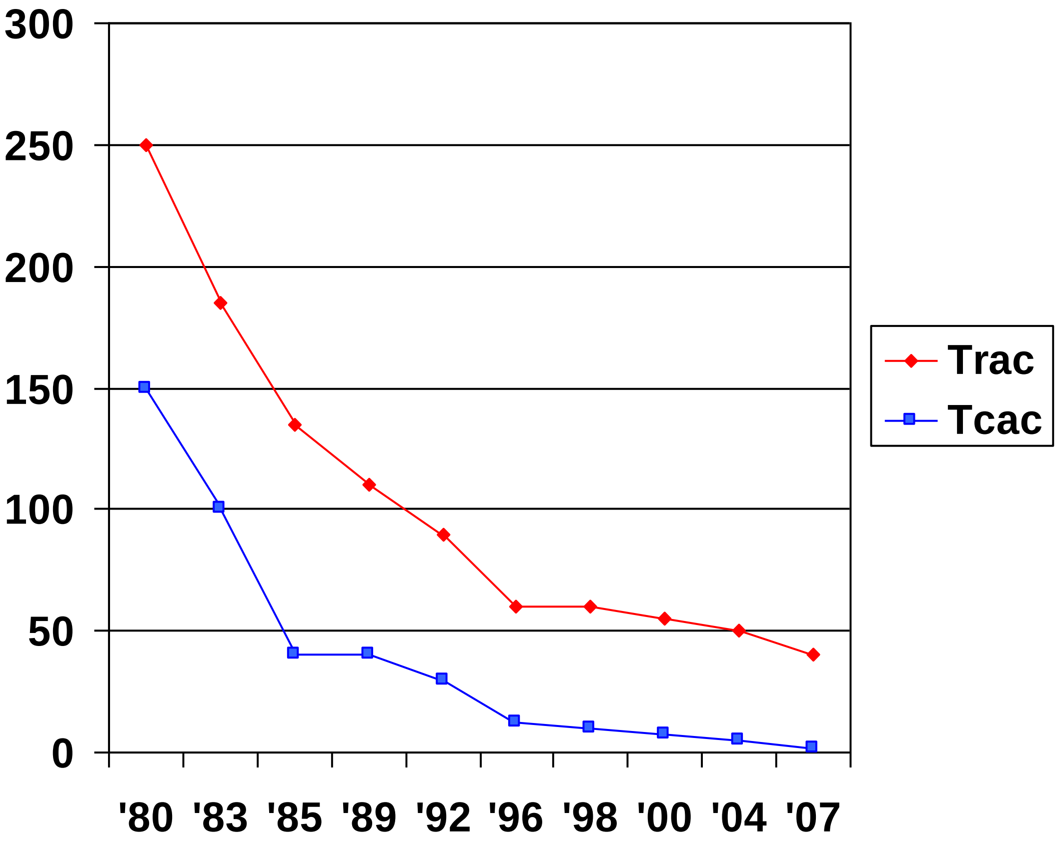


Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

DRAM Generations

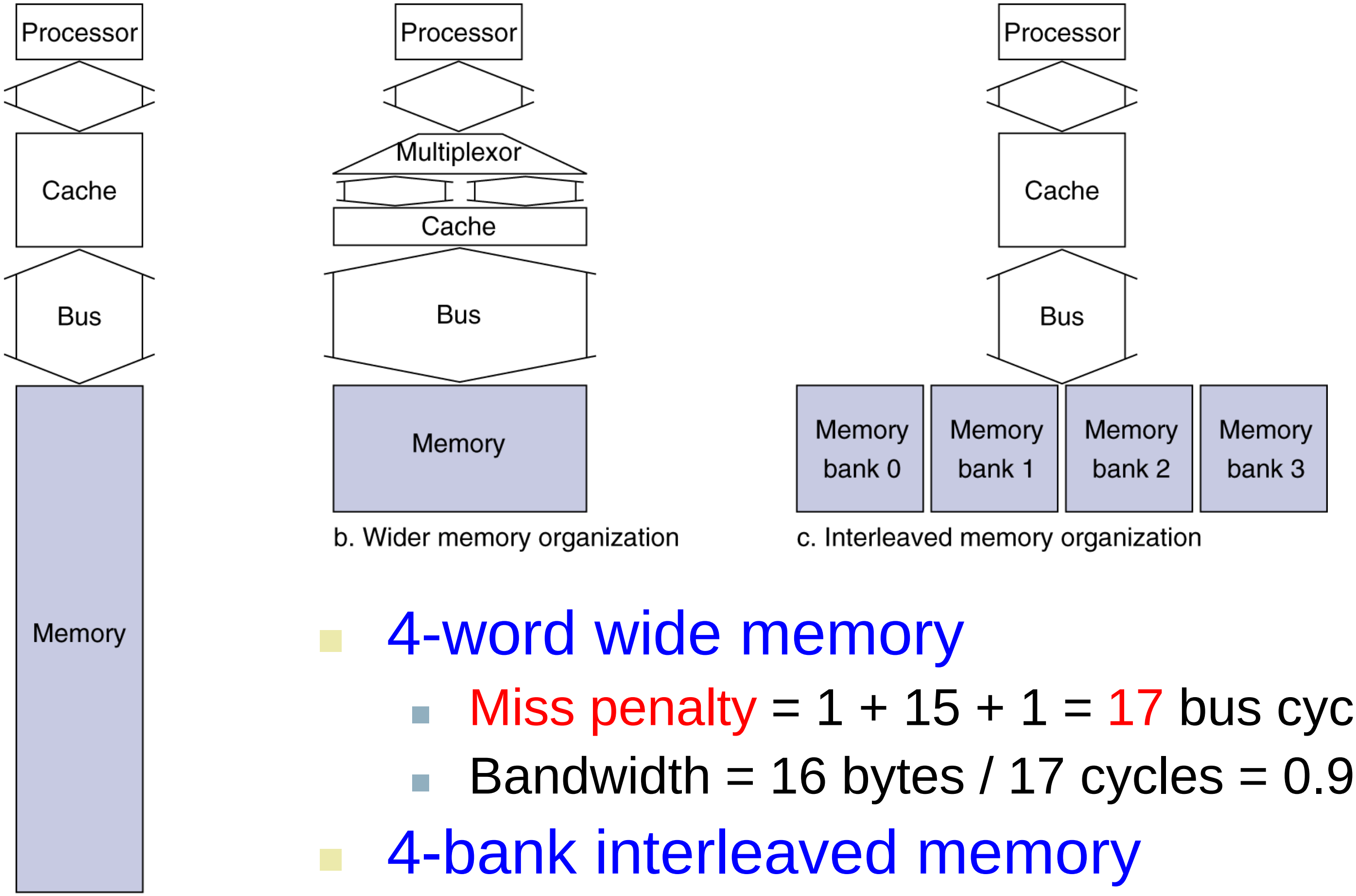
Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



DRAM Performance Factors

- Row buffer
  - Allows several words to be read and refreshed in parallel
- Synchronous DRAM
  - Allows for consecutive accesses in bursts without needing to send each address
  - Improves bandwidth
- DRAM banking
  - Allows simultaneous access to multiple DRAMs
  - Improves bandwidth

Increasing Memory Bandwidth



b. Wider memory organization

c. Interleaved memory organization

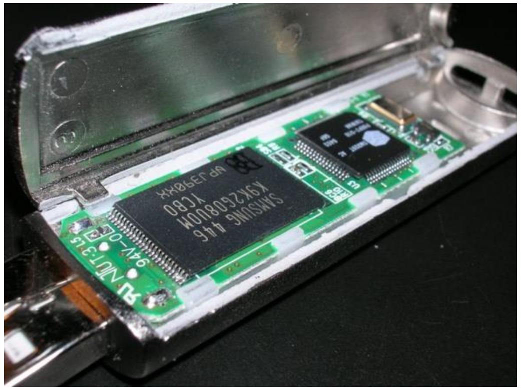
a. One-word-wide memory organization

- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Flash Storage

§6.4 Flash Storage

- Nonvolatile semiconductor storage
  - 100× – 1000× faster than disk
  - Smaller, lower power, more robust
  - But more $/GB (between disk and DRAM)

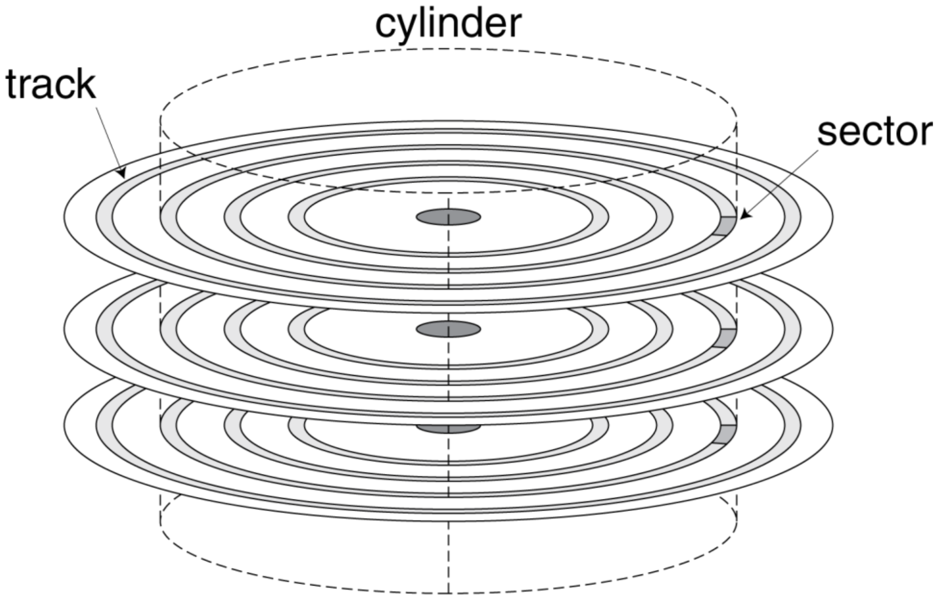


Flash Types

- NOR flash: bit cell like a NOR gate
  - Random read/write access
  - Used for instruction memory in embedded systems
- NAND flash: bit cell like a NAND gate
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, …
- Flash bits wears out after 1000's of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks

Disk Storage

- Nonvolatile, rotating magnetic storage



Disk Sectors and Access

- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
  - Synchronization fields and gaps
- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek: move the heads
  - Rotational latency
  - Data transfer
  - Controller overhead

Disk Access Example

- Given
  - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk
- Average read time
  - 4ms seek time
    + ½ / (15,000/60) = 2ms rotational latency
    + 512 / 100MB/s = 0.005ms transfer time
    + 0.2ms controller delay
    = 6.2ms
- If actual average seek time is 1ms
  - Average read time = 6.2 – 3 = 3.2ms

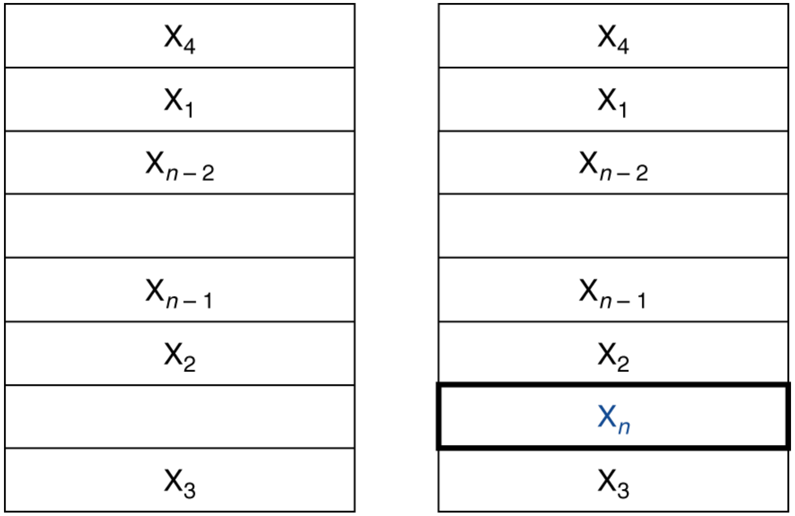
Disk Performance Issues

- Manufacturers quote average seek time
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times
- Smart disk controller allocate physical sectors on disk
  - Present logical sector interface to host
  - SCSI, ATA, SATA
- Disk drives include caches
  - Prefetch sectors in anticipation of access
  - Avoid seek and rotational delay

Cache Memory

§5.3 The Basics of Caches

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$

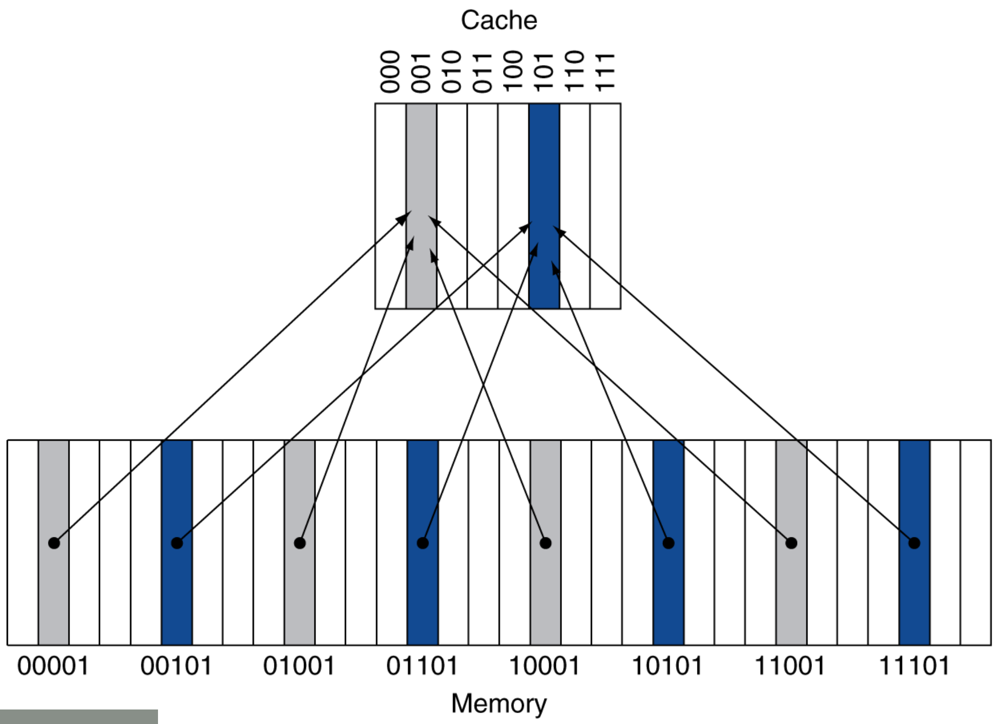


a. Before the reference to X_n b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	**10 110**	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	**11 010**	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	**10 000**	Miss	000
3	**00 011**	Miss	011
16	10 000	Hit	000

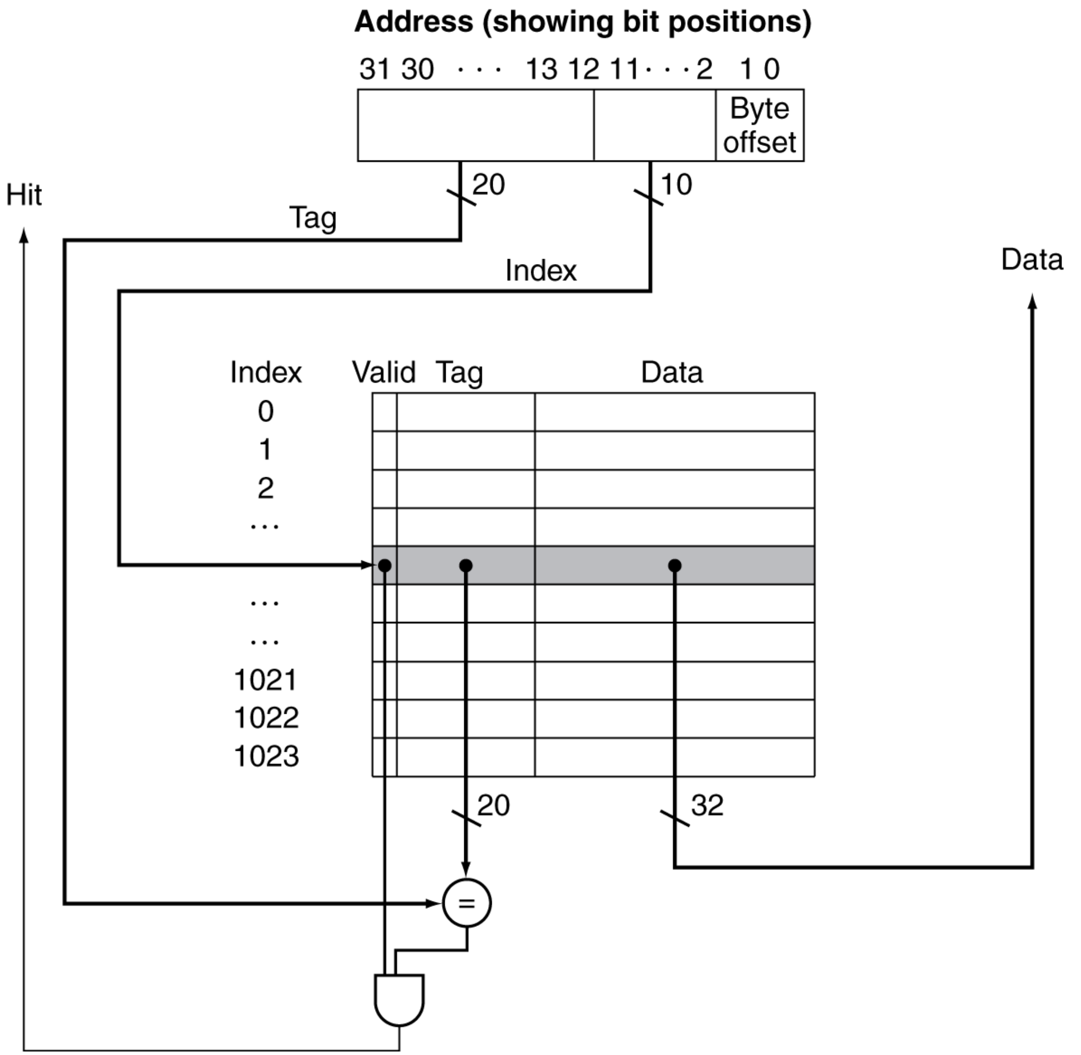
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

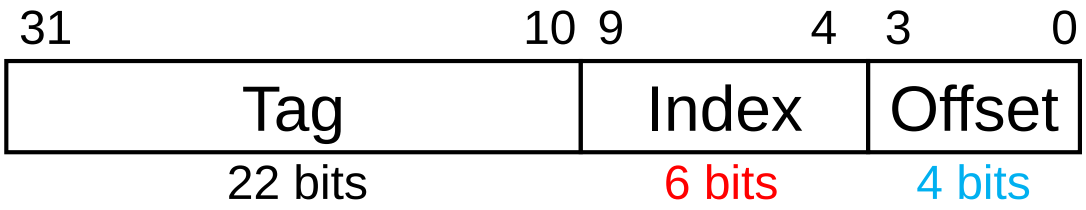
Address Subdivision



Example: Larger Block Size

- 64 blocks (= 2^6), 16 bytes/block (2^4)
  - To what block number does byte-address **1200** map?
- Block addr. = $\lfloor 1200/16 \rfloor = 75$
- Block # = (Block addr.) mod (# of blocks)

  = 75 modulo 64 = 11



31 … 10	9 … 4	3 … 0
Tag	Index	Offset
22 bits	6 bits	4 bits

Block Size Considerations

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

Block Size Considerations

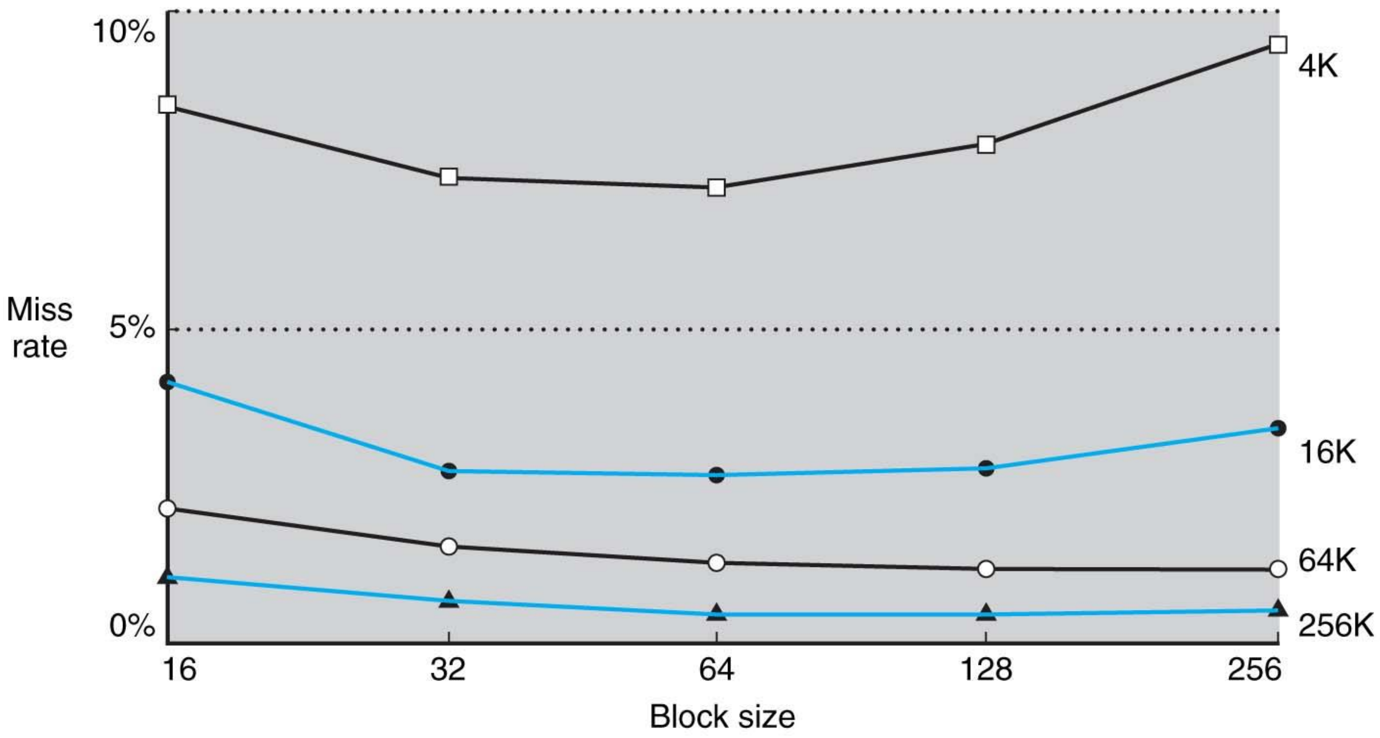
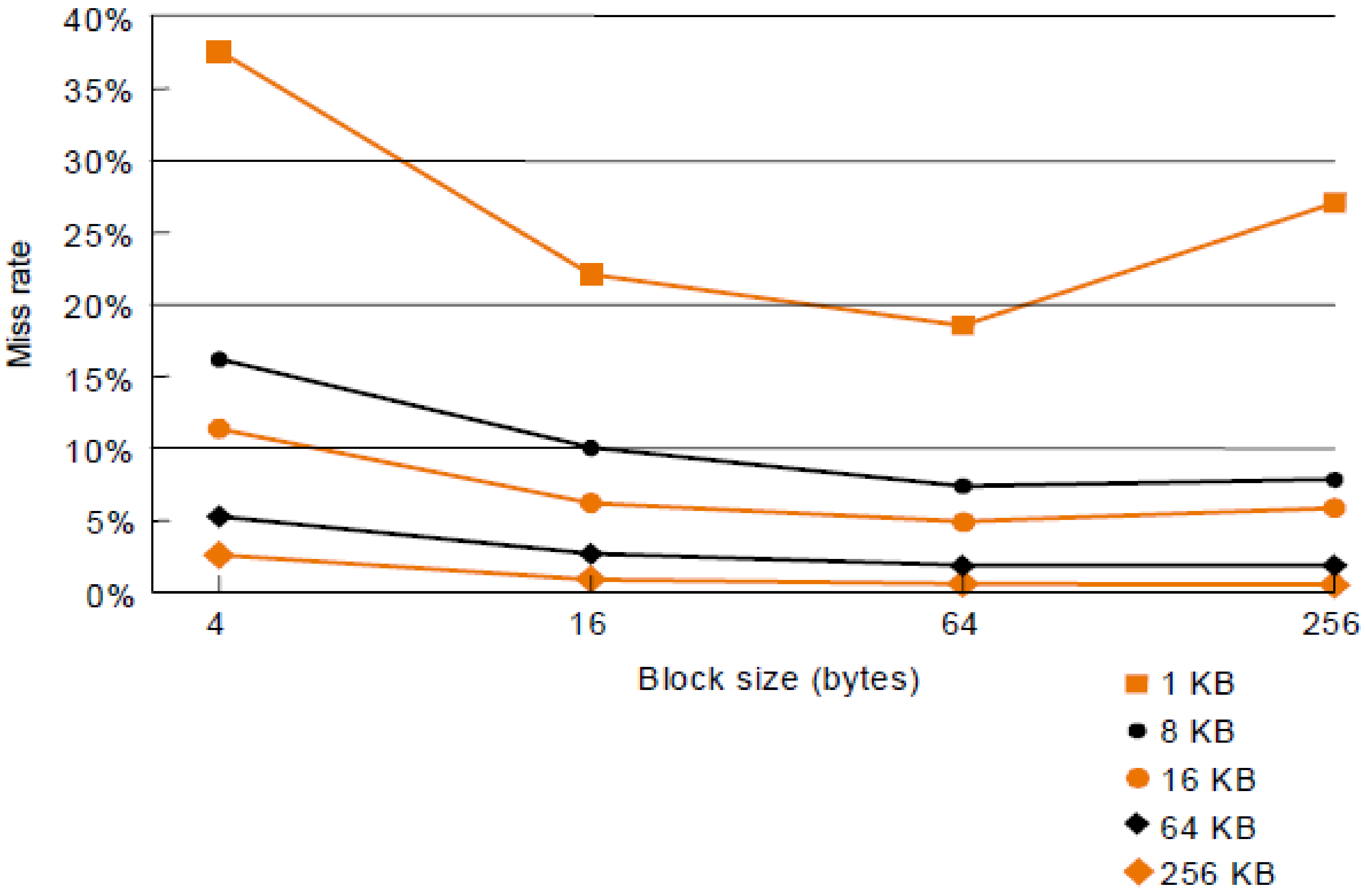


FIGURE 5.11. Miss rate versus block size. Note that the miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size (This figure is independent of associativity, discussed soon). Unfortunately, SPEC2000 traces would take too long if block size were included, so this data is based on SPEC92.

Block Size Considerations



Block Size Performance

- Simplified model:
  - execution time = (execution cycles + stall cycles) × cycle time
  - stall cycles = # of instructions × miss ratio × miss penalty
- Two ways of improving performance
  - decreasing the miss ratio (*e.g.*, use Associativity)
  - decreasing the miss penalty (*e.g.*, use Multi-level cache)

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
    - Stall the CPU pipeline (need PC-4)
    - Fetch block from next level of hierarchy (*i.e.*, read from memory and then write to cache)
    - If instruction cache miss, then restart instruction fetch
    - If data cache miss, then complete data access

Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - *e.g.*, if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - **Effective CPI = 1 + 0.1×100 = 11**
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Back

- Alternative: On data-write hit, just update the block in cache
    - Keep track of whether each block is dirty
- When a dirty block is replaced
    - Write it back to memory
    - Can use a write buffer to allow replacing block to be read first

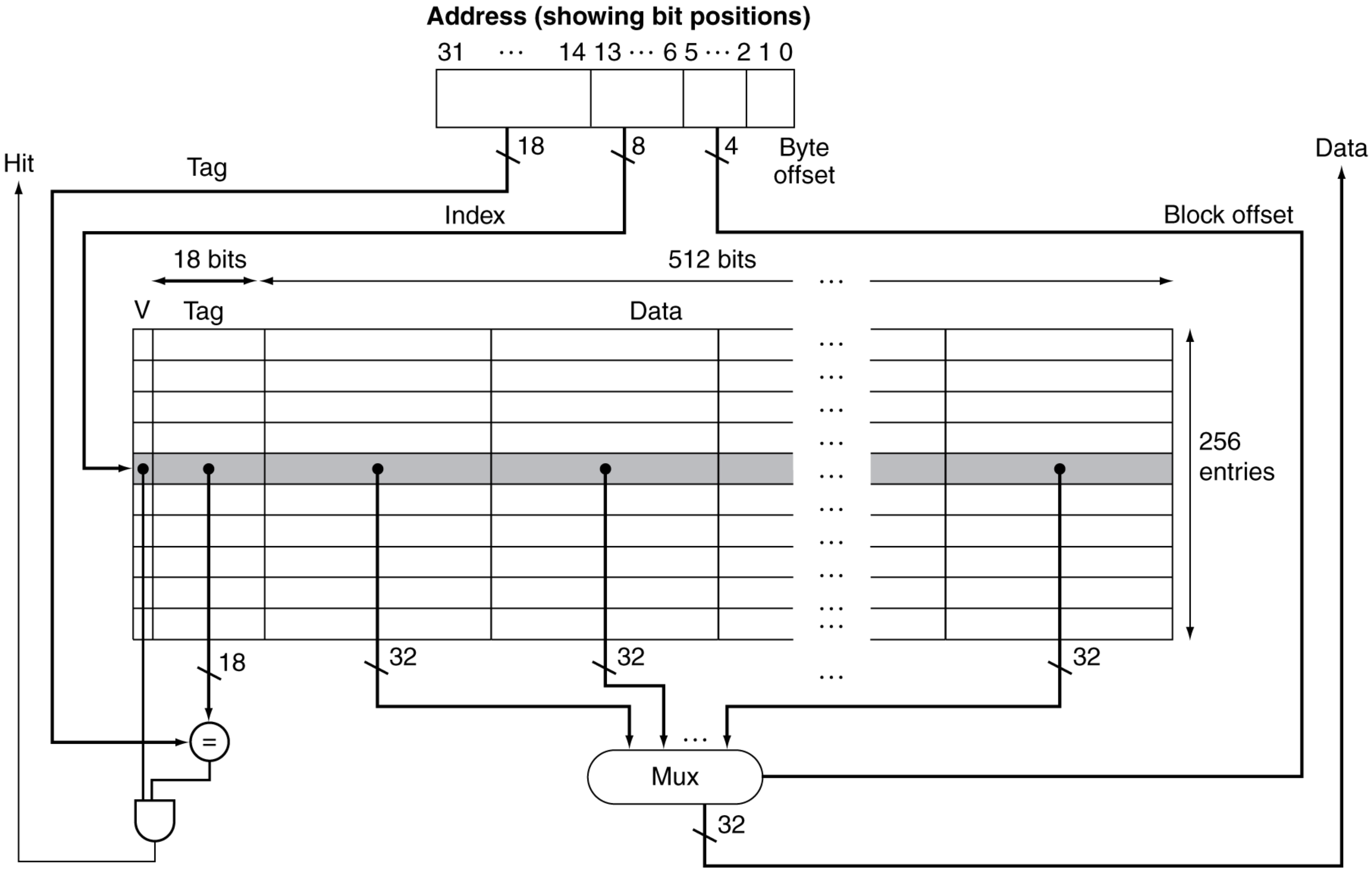
Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss (write allocate): fetch the block
  - Write around (write not allocate): don't fetch the block
    - Since programs often write a whole block before reading it (*e.g.*, initialization)
- For write-back
  - Use write allocate to fetch the block
    - because the data in memory and in cache may be different, thus need a write buffer

Example: Intrinsity FastMATH

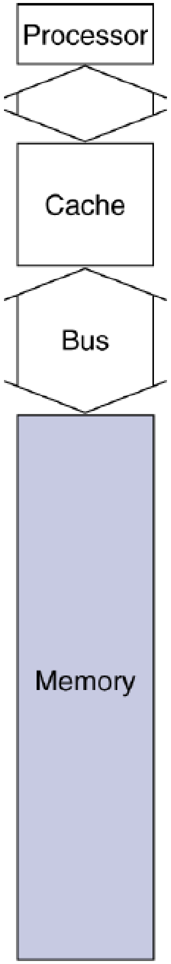
- Embedded MIPS processor (DSP-like)
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

Example: Intrinsity FastMATH



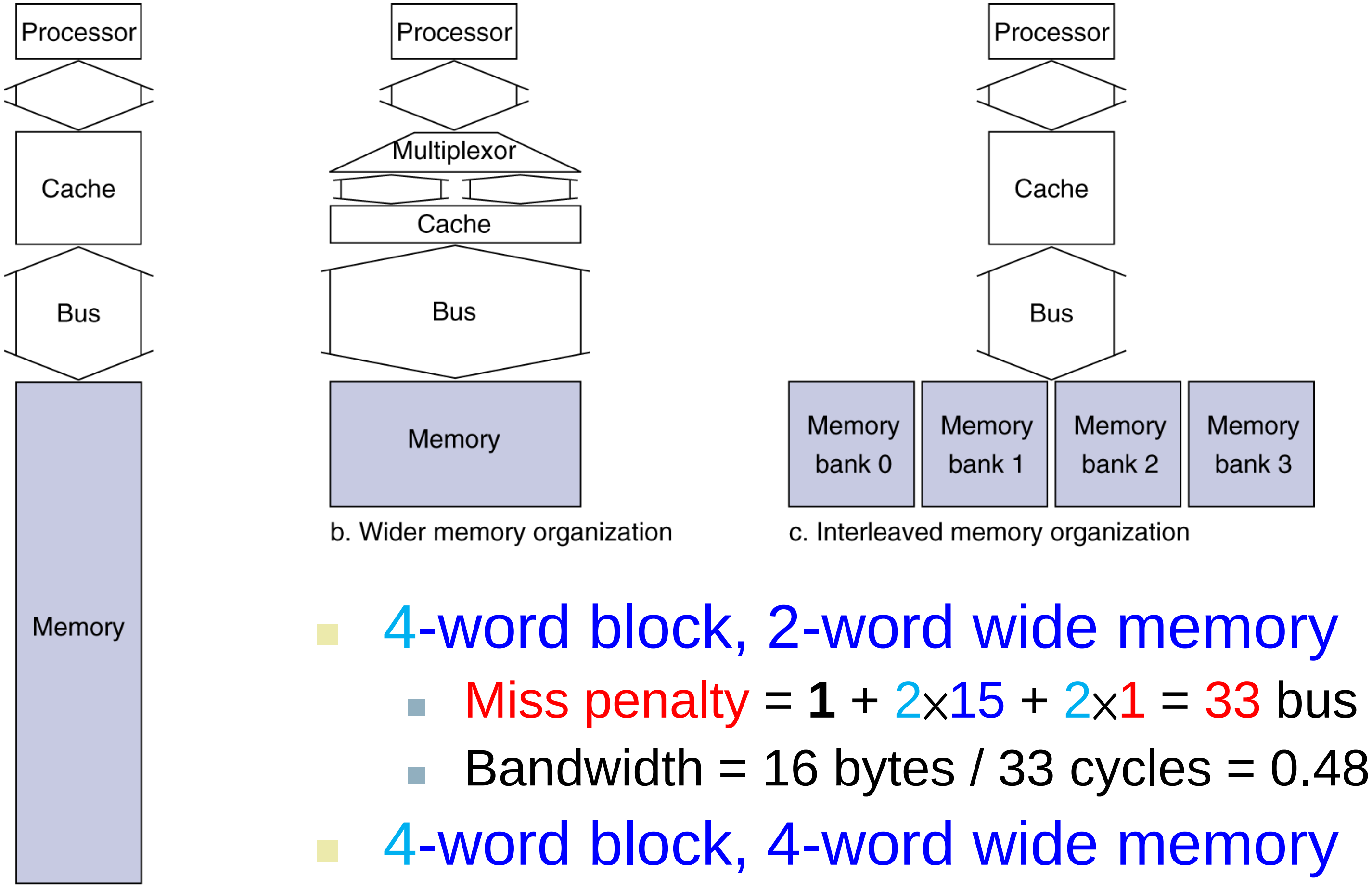
Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (*e.g.*, 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock
- Example: cache block read
  - **1** bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
  - Miss penalty = **1** + 4×15 + 4×1
    = 65 bus cycles
  - Bandwidth = 16 Bytes / 65 cycles
    = 0.25 B / cycle



a. One-word-wide memory organization

Increasing Memory Bandwidth



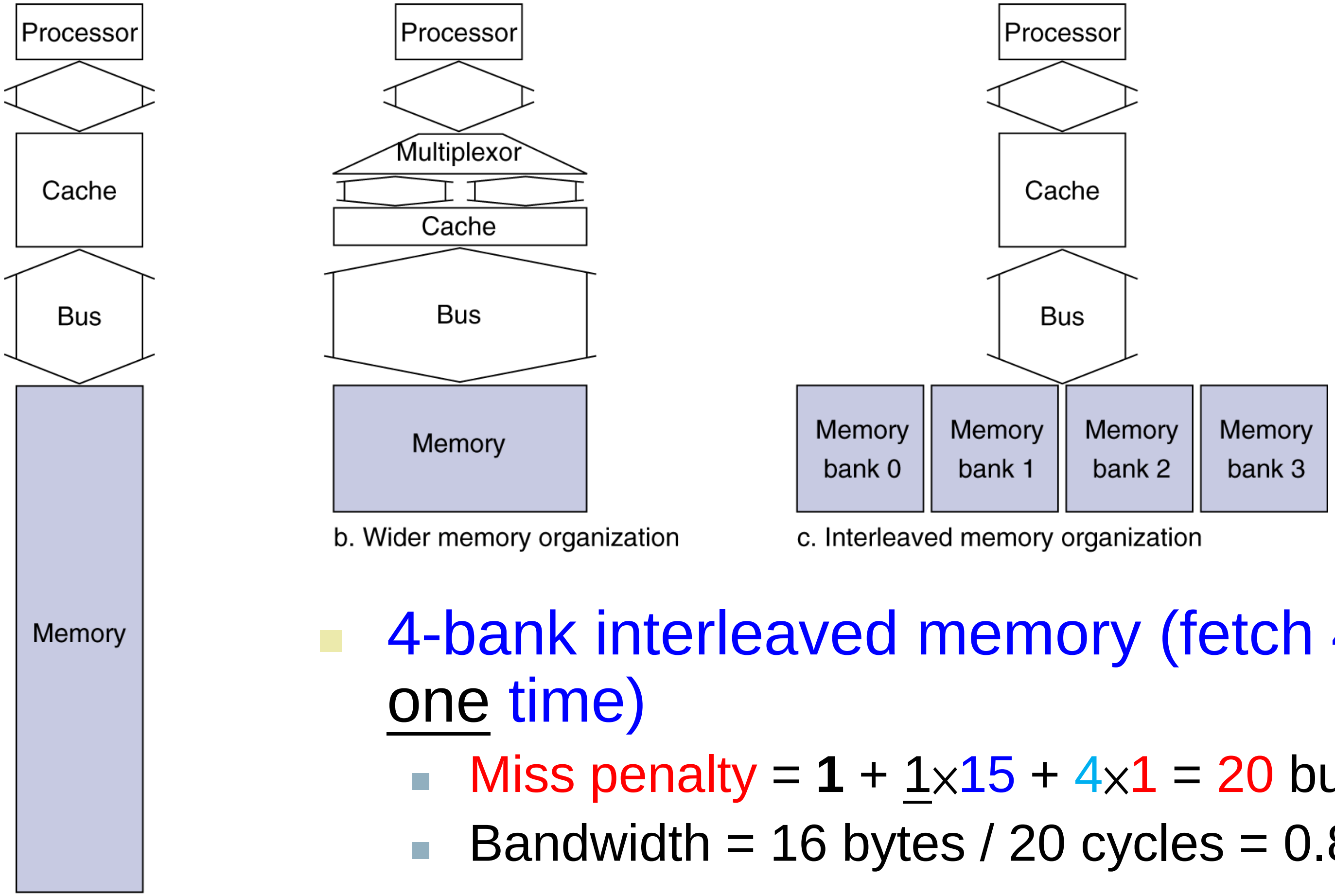
b. Wider memory organization

c. Interleaved memory organization

a. One-word-wide memory organization

- 4-word block, 2-word wide memory
  - Miss penalty = **1** + 2×15 + 2×1 = 33 bus cycles
  - Bandwidth = 16 bytes / 33 cycles = 0.48 B / cycle
- 4-word block, 4-word wide memory
  - Miss penalty = **1** + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B / cycle

Increasing Memory Bandwidth



b. Wider memory organization

c. Interleaved memory organization

a. One-word-wide memory organization

- 4-bank interleaved memory (fetch 4 words in one time)
  - Miss penalty = **1** + $\underline{1}\times 15$ + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Measuring Cache Performance

- Components of CPU time = (1) + (2)
  - Program execution cycles (1)
    - Includes cache hit time
  - Memory stall cycles (2) from cache misses
    = 2a (for Read) + 2b (for Write)
- Assumption of using (2a) only:

$$\text{Memory stall cycles}$$

$$= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
    (ideal means no cache miss nor memory stall)
  - Load & stores are 36% of instructions
- Miss cycles per instruction (Miss CPI = 3.44)
  - I-cache: 0.02 × 100 = **2**
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster
  - Miss CPI/ Actual CPI = 3.44/5.44 = 63% (stall proportion)

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 1

    (ideal means no cache miss nor memory stall)
  - Load & stores are 36% of instructions
- Miss cycles per instruction (Miss CPI = 3.44)
  - I-cache: 0.02 × 100 = **2**
  - D-cache: 0.36 × 0.04 × 100 = 1.44
- Actual CPI = 1 + 2 + 1.44 = 4.44
  - Ideal CPU is 4.44/2 =2.22 times faster
  - Miss CPI/ Actual CPI = 3.44/4.44 = 77% (stall proportion)

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Performance Summary

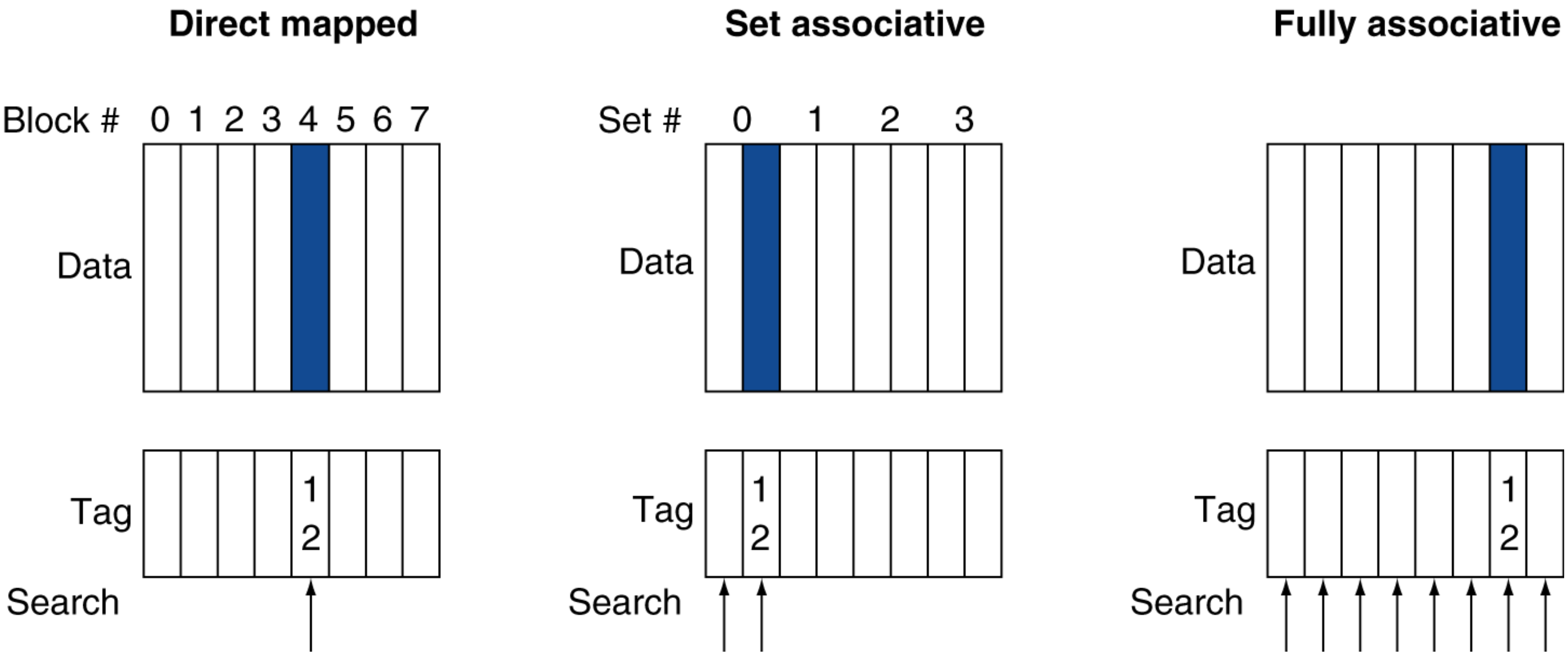
- When CPU performance increased
  - Miss penalty becomes more significant
- Sol.-1: Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Sol.-2: Increasing clock rate
  - Memory stalls account for more CPU cycles

⇒ Can't neglect cache behavior when evaluating system performance

Associative Caches

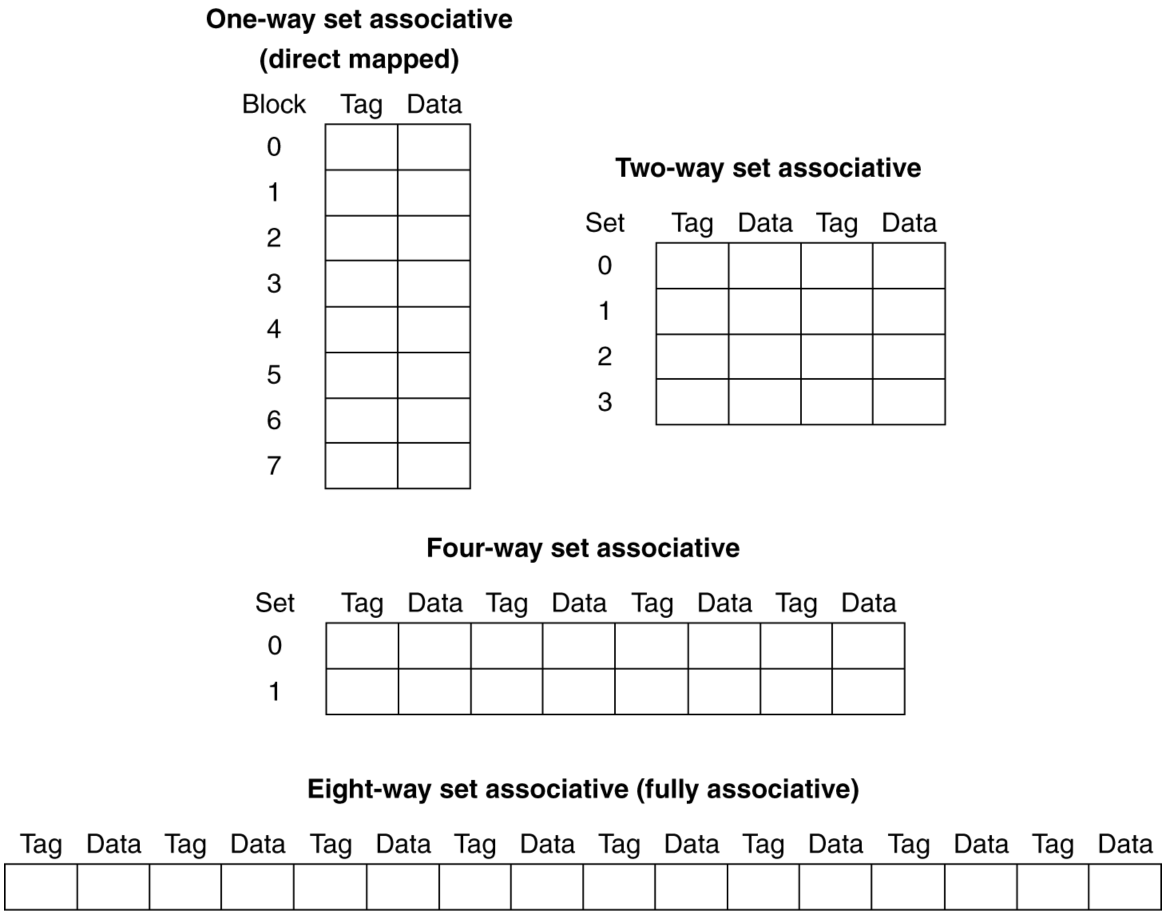
- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Associative Cache Example



Spectrum of Associativity

- For a cache with 8 entries



One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence:
  - 0, 8, 0, 6, 8

Block -Addr	Cache Block
0	0 = (0 mod 4)
6	2 = (6 mod 4)
8	0 = (8 mod 4)

- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	Mem[0]			
8	0	miss	Mem[8]			
0	0	miss	Mem[0]			
6	2	miss	Mem[0]		Mem[6]	
8	0	miss	Mem[8]		Mem[6]	

Associativity Example

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	**0 = (0 mod 2)**	miss	**Mem[0]**			
8	**0 = (8 mod 2)**	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	**0 = (6 mod 2)**	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

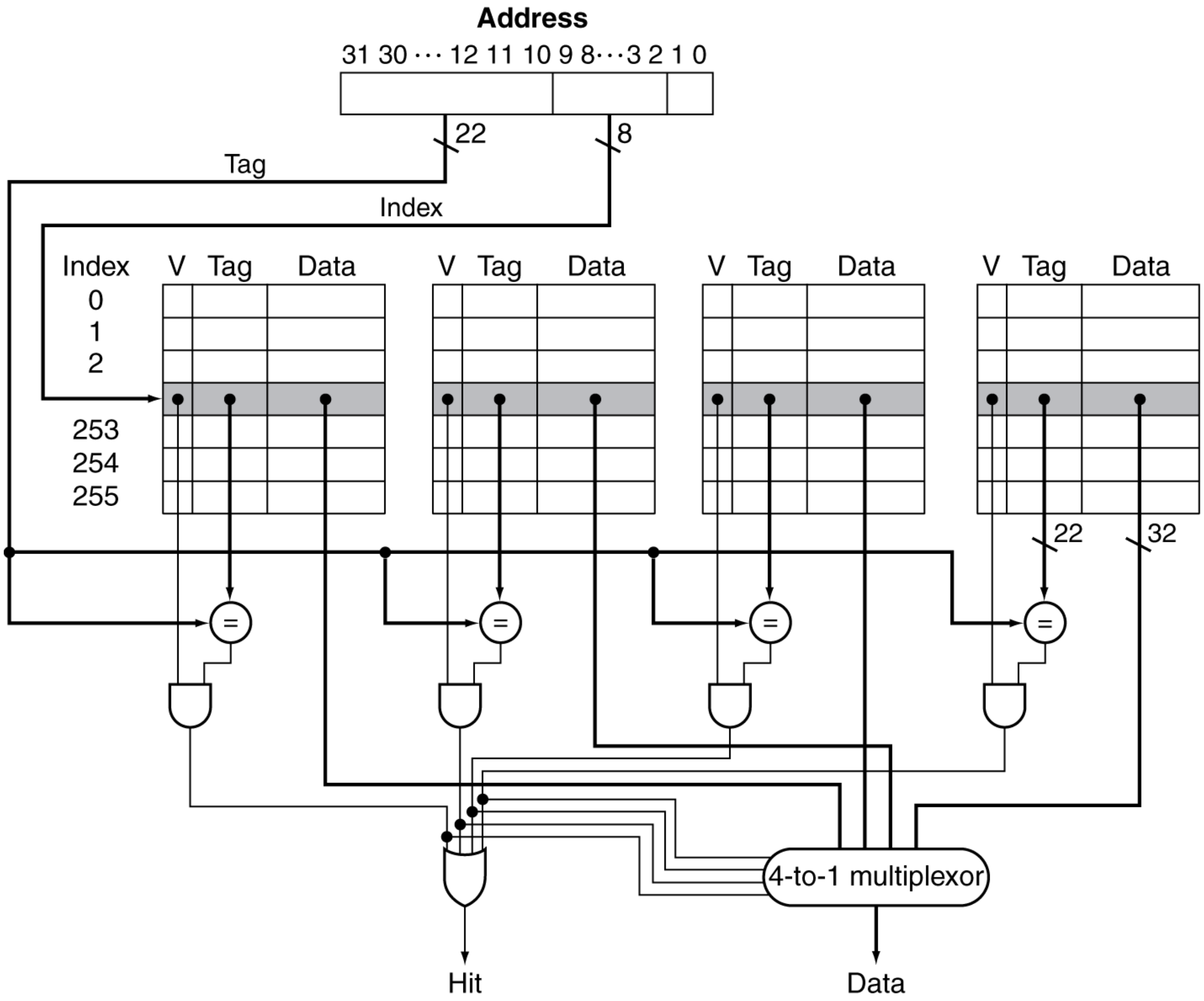
- Fully associative (no need of cache index)

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system (Intrinsity FastMath) with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Set Associative Cache Organization



Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set to replace using one of the following rules.
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Multilevel Caches

- Primary cache (Level-1, L-1) attached to CPU
  - Small, but fast
- Level-2 cache services (L-1) misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

Multilevel Cache Example

- Given
  - CPU base CPI = **1**,
    clock rate = 4GHz (= 0.25ns/clock cycle)
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache (L-1 cache)
  - Miss penalty = 100ns / 0.25ns = **400** cycles
  - Effective CPI = **1** + 0.02 × **400** = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns / 0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

Multilevel Cache Considerations

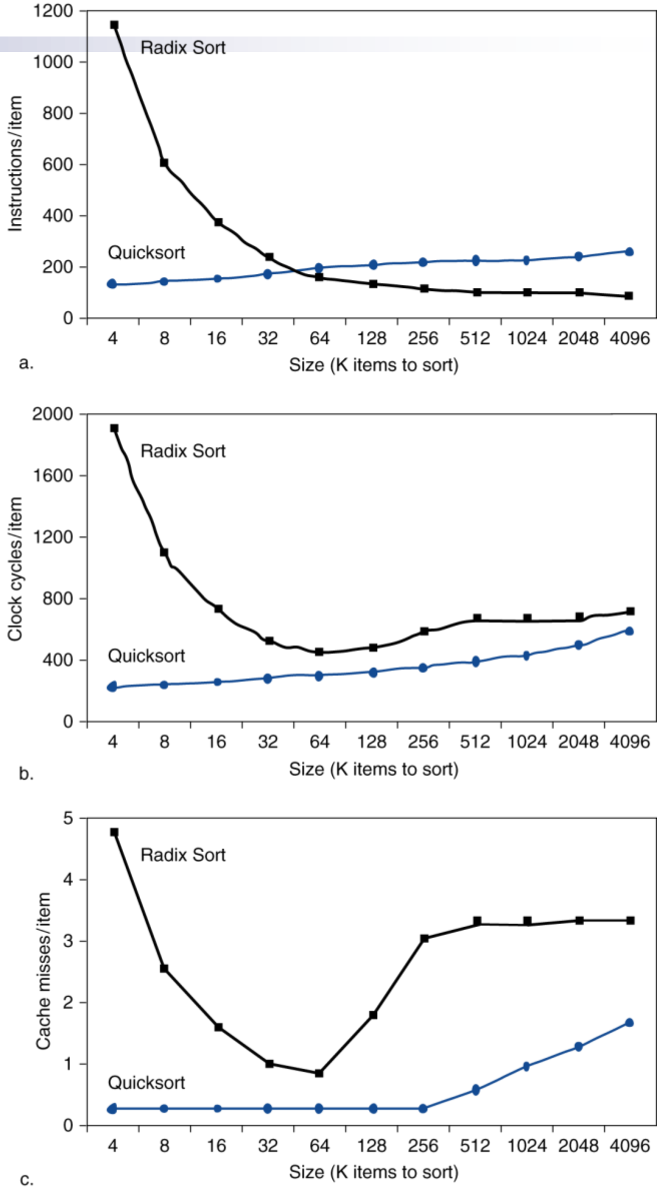
- Primary cache (L-1 cache)
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyze
  - Use system simulation

Interactions with Software

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access



Y-axis in the right figures:
(a): Instr. Executed / Item
(b): Clock Cycles / Item
(c): Cache Misses / Item

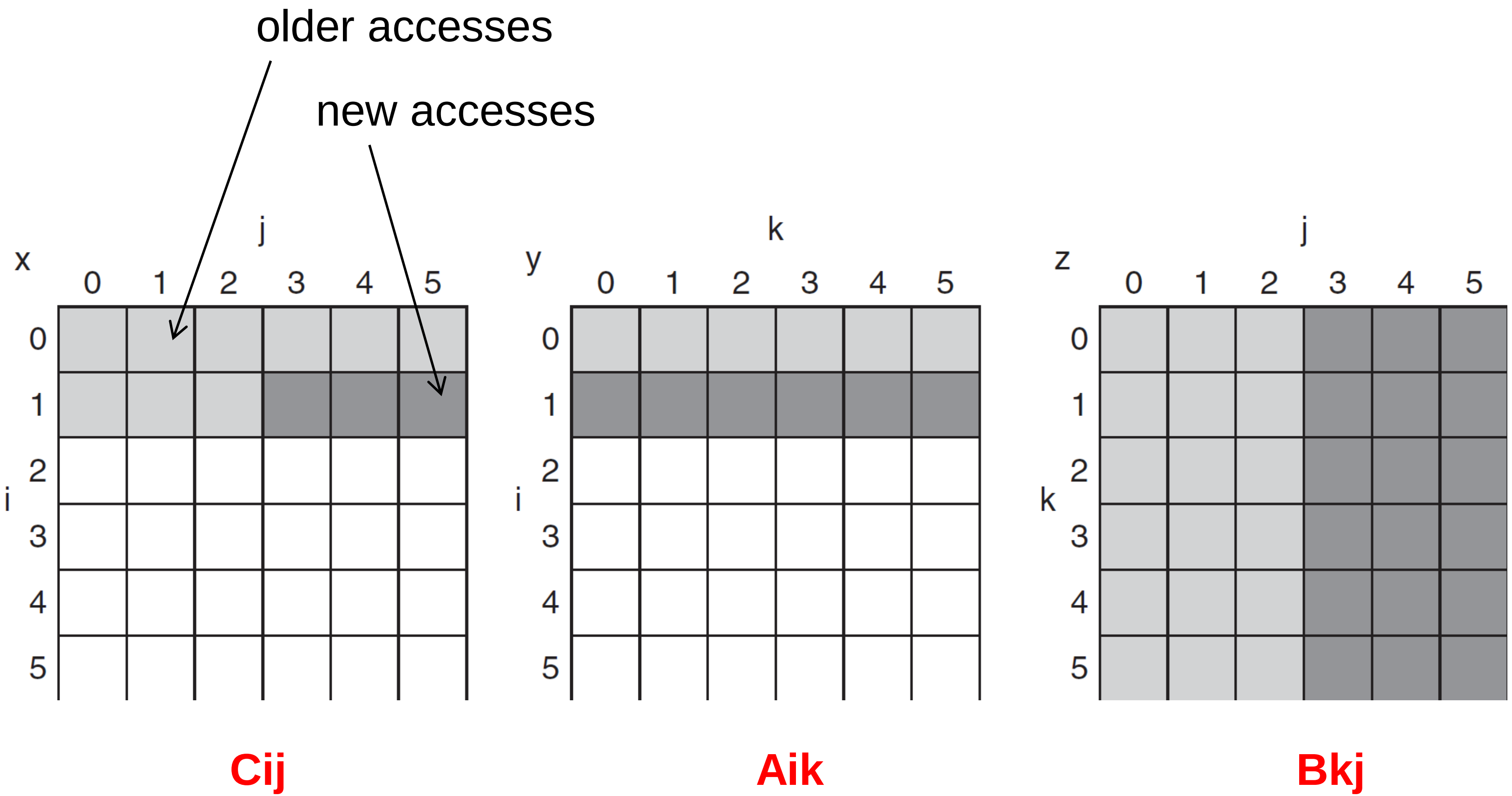
Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM (Double-Precision General Matrix Multiplication):

```
for (int j = 0; j < n; ++j)
  {
    double cij = C[i+j*n];
    for( int k = 0; k < n; k++ )
         cij += A[i+k*n] * B[k+j*n];
    C[i+j*n] = cij;
  }
```

DGEMM Access Pattern

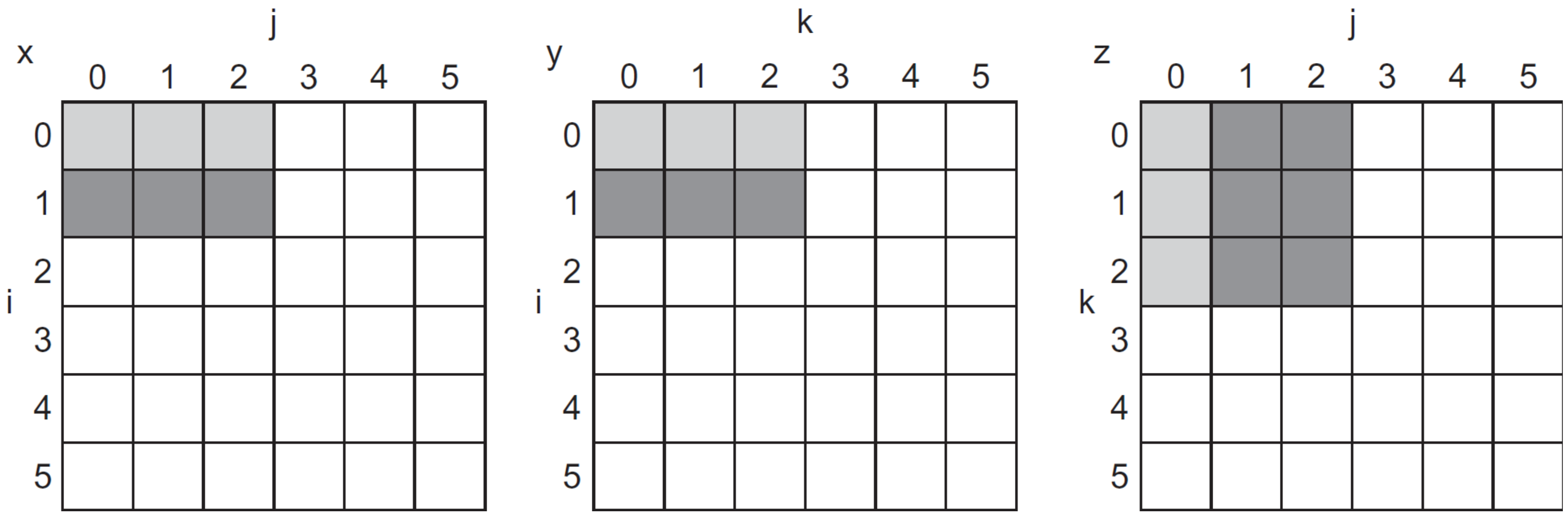
- C, A, and B arrays, each having a size of 32x32, each element is 8 Bytes (or 64-bit)

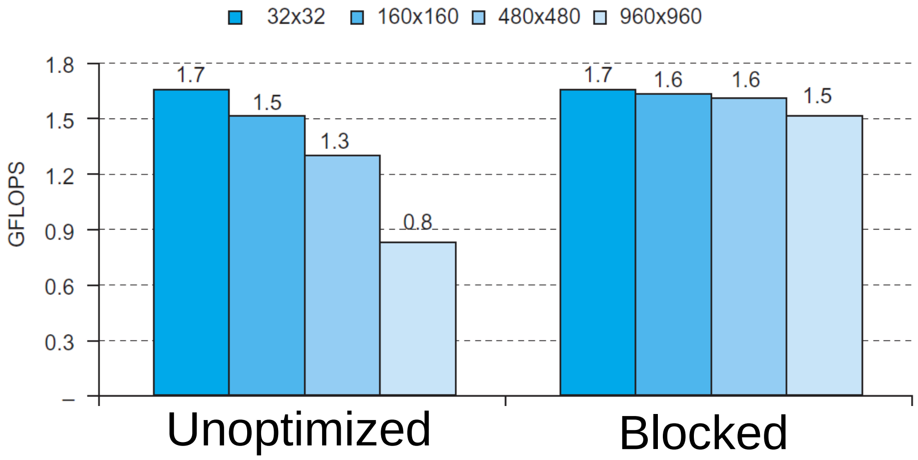


Cache Blocked DGEMM

```
#define BLOCKSIZE 32
void do_block (int n, int si, int sj, int sk, double *A, double
*B, double *C)
{
 for (int i = si; i < si+BLOCKSIZE; ++i)
  for (int j = sj; j < sj+BLOCKSIZE; ++j)
  {
   double cij = C[i+j*n];/* cij = C[i][j] */
   for( int k = sk; k < sk+BLOCKSIZE; k++ )
   cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
  C[i+j*n] = cij;/* C[i][j] = cij */
 }
}
void dgemm (int n, double* A, double* B, double* C)
{
 for ( int sj = 0; sj < n; sj += BLOCKSIZE )
  for ( int si = 0; si < n; si += BLOCKSIZE )
   for ( int sk = 0; sk < n; sk += BLOCKSIZE )
    do_block(n, si, sj, sk, A, B, C);
}
```

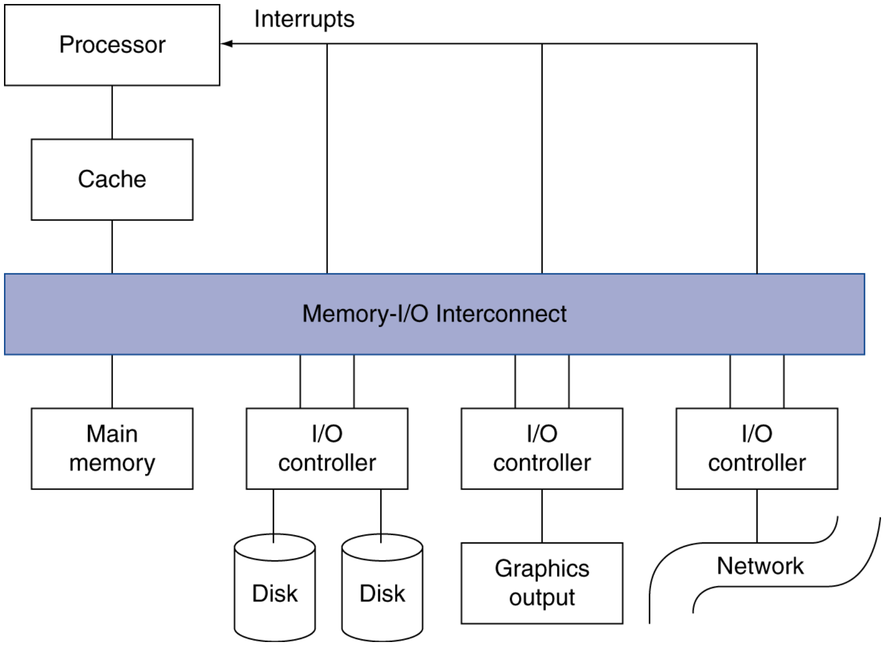
Blocked DGEMM Access Pattern





Input/Output Devices

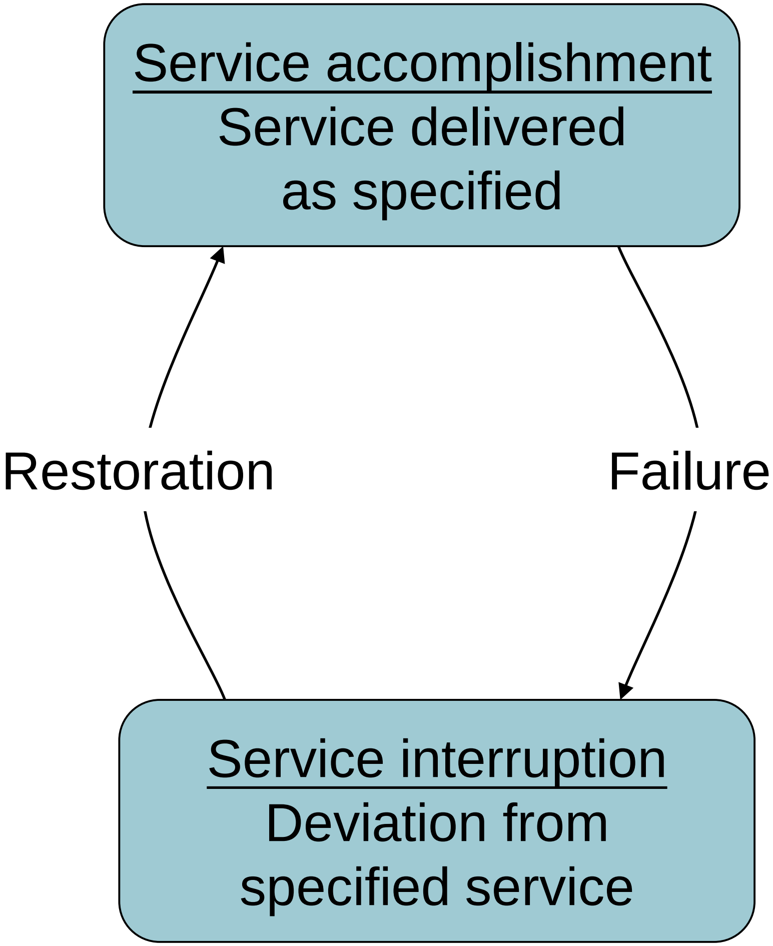
- I/O (Input/Output) devices can be characterized by
  - Behaviour: input, output, storage
  - Partner: human or machine
  - Data rate: bytes/sec, transfers/sec
- I/O bus connections



I/O System Characteristics

- Dependability is important
  - Particularly for storage devices
- Performance measures
  - Latency (response time)
  - Throughput (bandwidth)
  - Desktops & embedded systems
    - Mainly interested in response time & diversity of devices
  - Servers
    - Mainly interested in throughput & expandability of devices

Dependability


Service accomplishment
Service delivered as specified
Restoration
Failure
Service interruption
Deviation from specified service

- Fault: failure of a component
  - May or may not lead to system failure

Dependability Measures

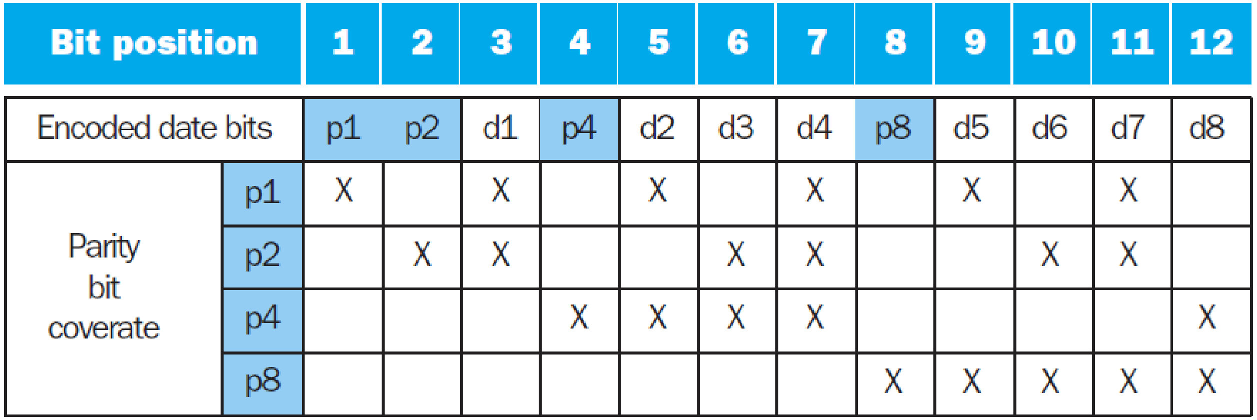
- Reliability: mean time to failure (MTTF)
- Service interruption: mean time to repair (MTTR)
- Mean time between failures
  - MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR)
- Improving Availability
  - Increase MTTF: fault avoidance, fault tolerance, fault forecasting
  - Reduce MTTR: improved tools and processes for diagnosis and repair

The Hamming SEC Code

- Hamming distance
    - Number of bits that are different between two bit patterns
- Minimum distance = 2 provides single-bit error detection
    - *e*.*g*. parity code
- Minimum distance = 3 provides single error correction, 2-bit error detection

Encoding SEC

- To calculate Hamming code:
  - Number bits from 1 on the left
  - Mark bit positions that are a power 2 as parity bits
  - Each parity bit checks certain data bits:



Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded date bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverate	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

Decoding SEC

- Value of parity bits indicates which bits are in error
  - Use numbering from encoding procedure
  - *e.g.*
    - Parity bits p8 p4 p2 p1 = 0 0 0 0 indicates no error
    - Parity bits p8 p4 p2 p1 = 1 0 1 0 indicates bit at the 10th position (*i.e*., d6) was flipped (or in error)

SEC/DED Code

- Add an additional parity bit for the whole word (p_n)
- Make Hamming distance = 4
- Decoding:
  - Let H = SEC parity bits
    - H even, p_n even, no error
    - H odd, p_n odd, correctable single bit error
    - H even, p_n odd, error in p_n bit
    - H odd, p_n even, double error occurred
- Note: ECC DRAM uses SEC/DED with 8 bits protecting each bit in a 64-bit block