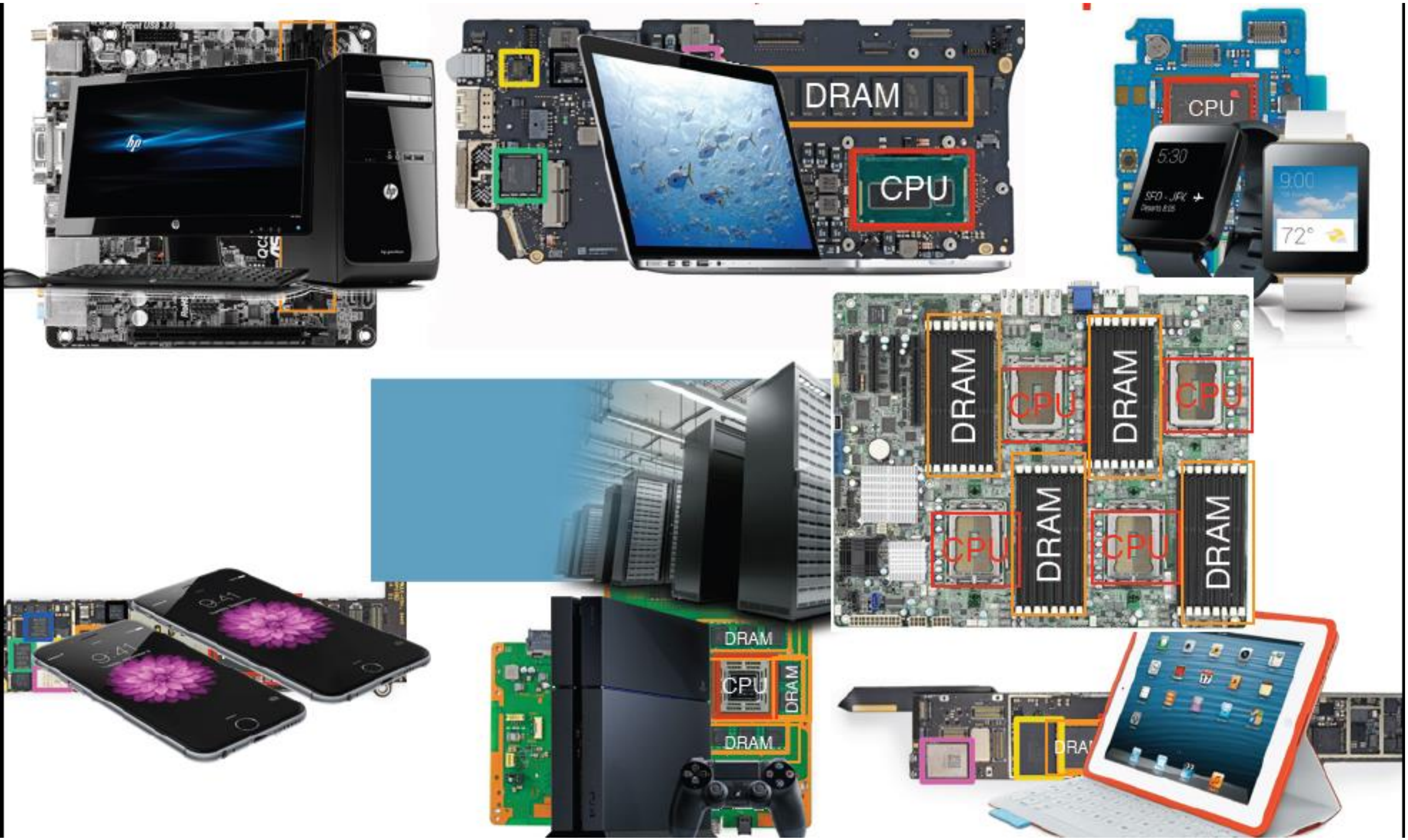
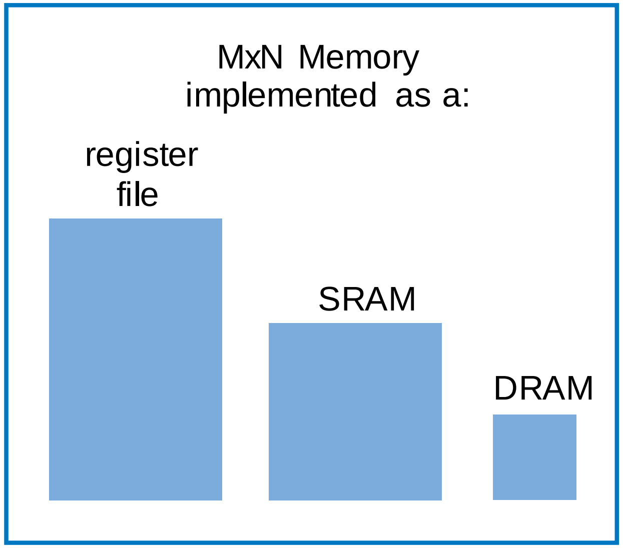
Systems

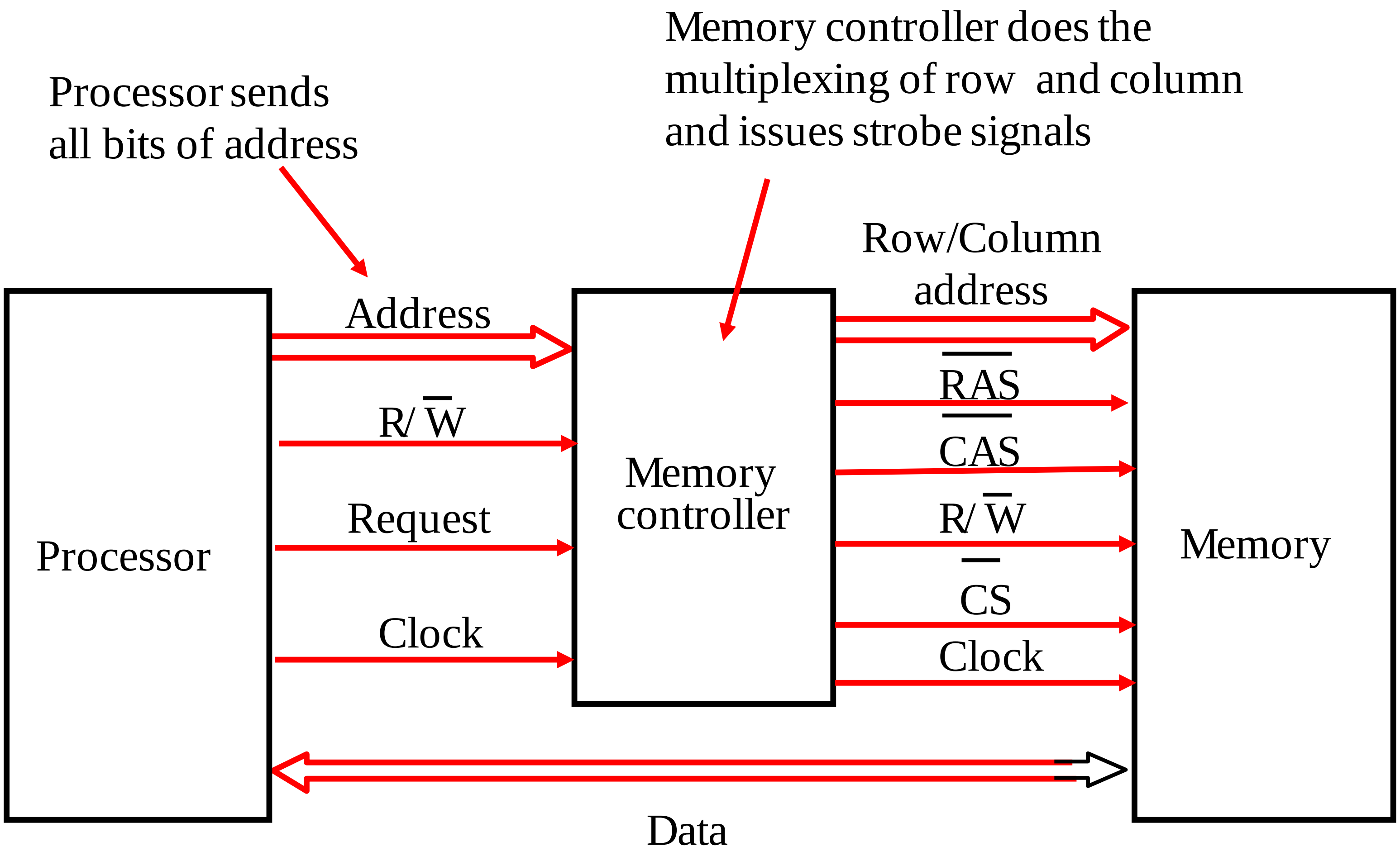

DRAM
CPU
CPU
5:30
SFO - JFK
9:00
72°
DRAM
CPU
DRAM
CPU
CPU
DRAM
CPU
DRAM
9:41
9:41
DRAM
CPU
DRAM
DRAM
DRAM

Comparing Memory

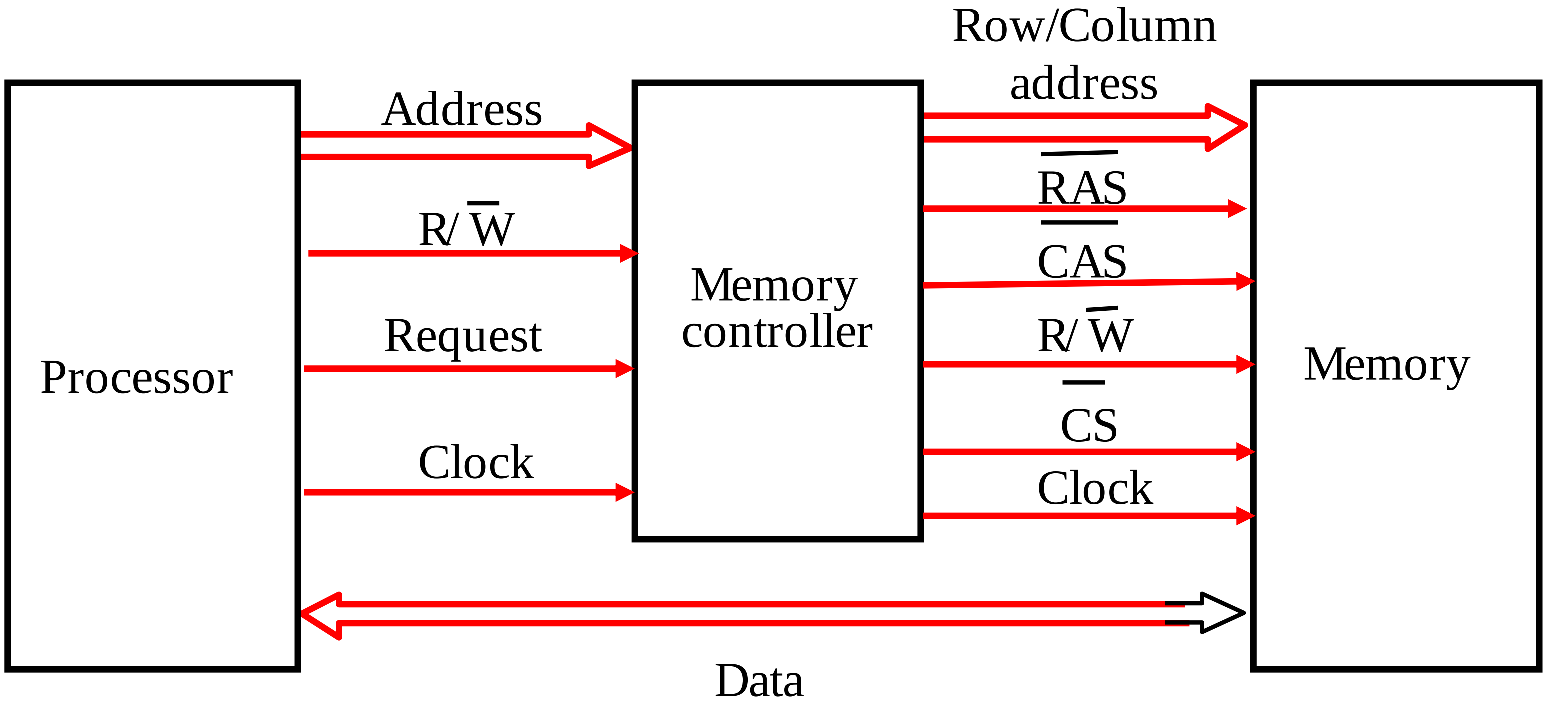
- Register file
  - Fastest
  - But biggest size
- SRAM
  - Fast
  - More compact than register file
- DRAM
  - Slowest (capacitor)
    - And refreshing takes time
  - But very compact (lower cost)
- Use register file for small items, SRAM for large items, and DRAM for huge items
  - Note: DRAM's big capacitor requires a special chip design process, so DRAM is often a separate chip



Size comparison for same number of bits (not to scale)



Use of a memory controller.



Memory controller provides the refresh control if not done on the chip

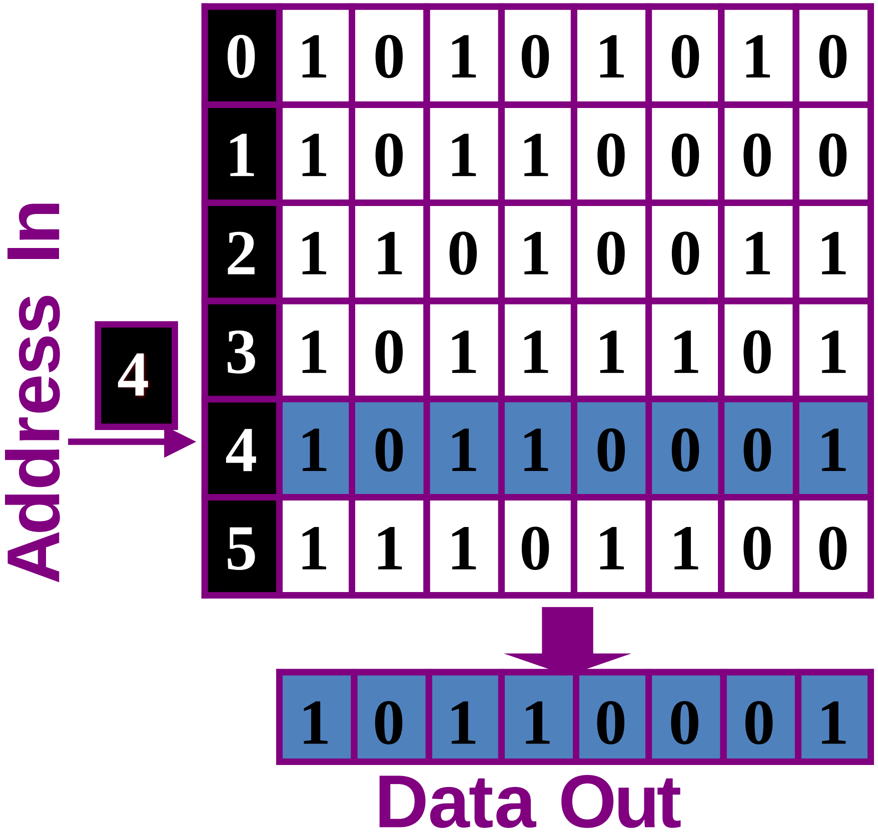
Refreshing typically once every 64 ms. At a cost of .2ms

Less than .4% overhead

Use of a memory controller.

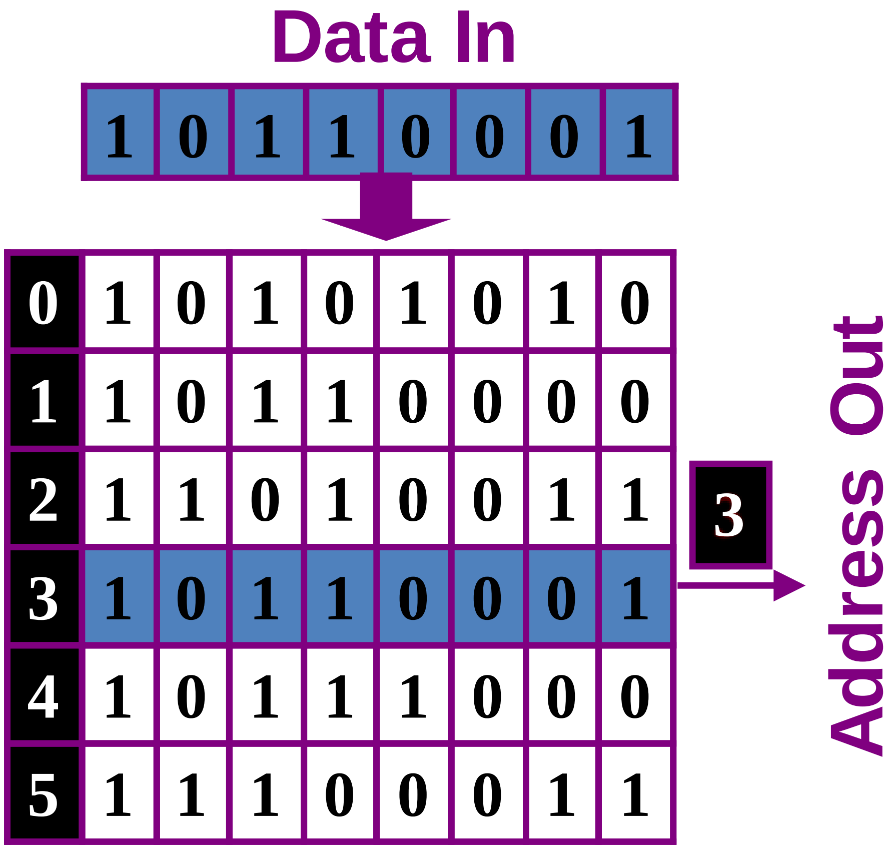
CAM

- CAM vs. RAM



0	1	0	1	0	1	0	1	0
1	1	0	1	1	0	0	0	0
2	1	1	0	1	0	0	1	1
3	1	0	1	1	1	1	0	1
4	1	0	1	1	0	0	0	1
5	1	1	1	0	1	1	0	0

1	0	1	1	0	0	0	1

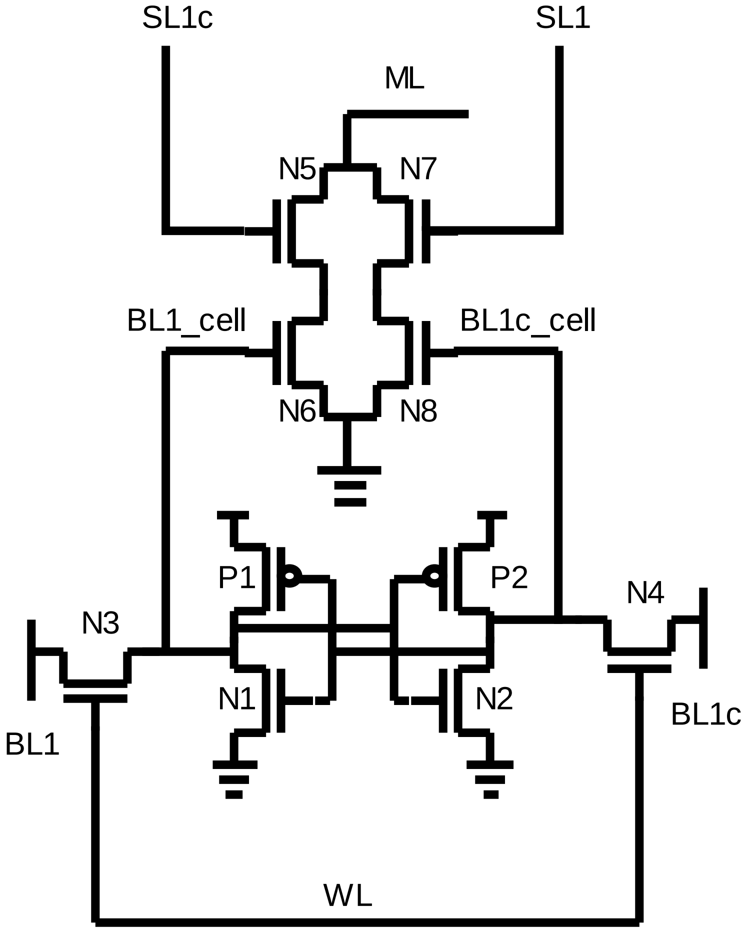


1	0	1	1	0	0	0	1

0	1	0	1	0	1	0	1	0
1	1	0	1	1	0	0	0	0
2	1	1	0	1	0	0	1	1
3	1	0	1	1	0	0	0	1
4	1	0	1	1	1	0	0	0
5	1	1	1	0	0	0	1	1

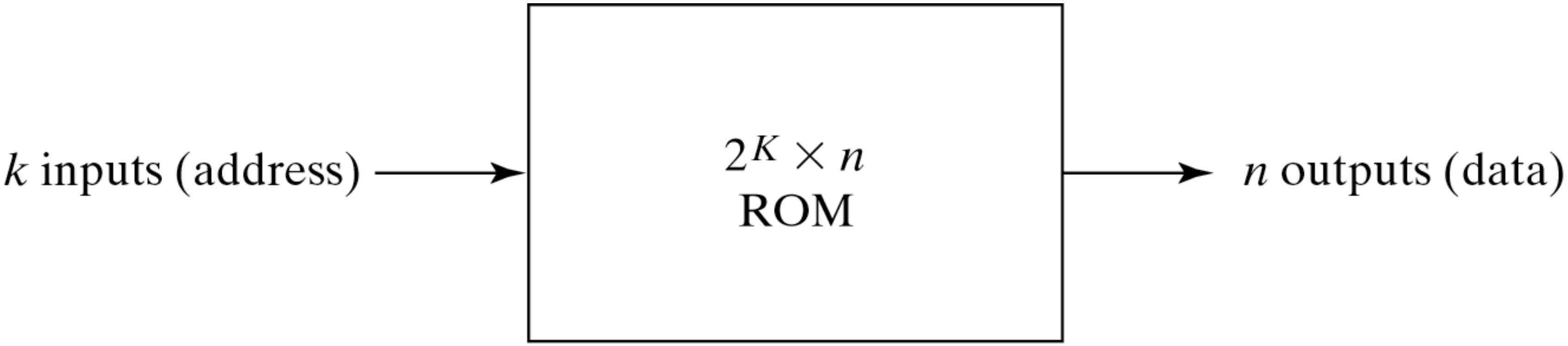
CAM

- Binary CAM Cell
  - ML pre-charged to V_{DD}
  - Match: ML remains at V_{DD}
  - Mismatch: ML discharges



Read-Only Memory

- A block diagram of a ROM is shown below. It consists of k address inputs and n data outputs.
- The number of words in a ROM is determined from the fact that k address input lines are needed to specify 2^k words.

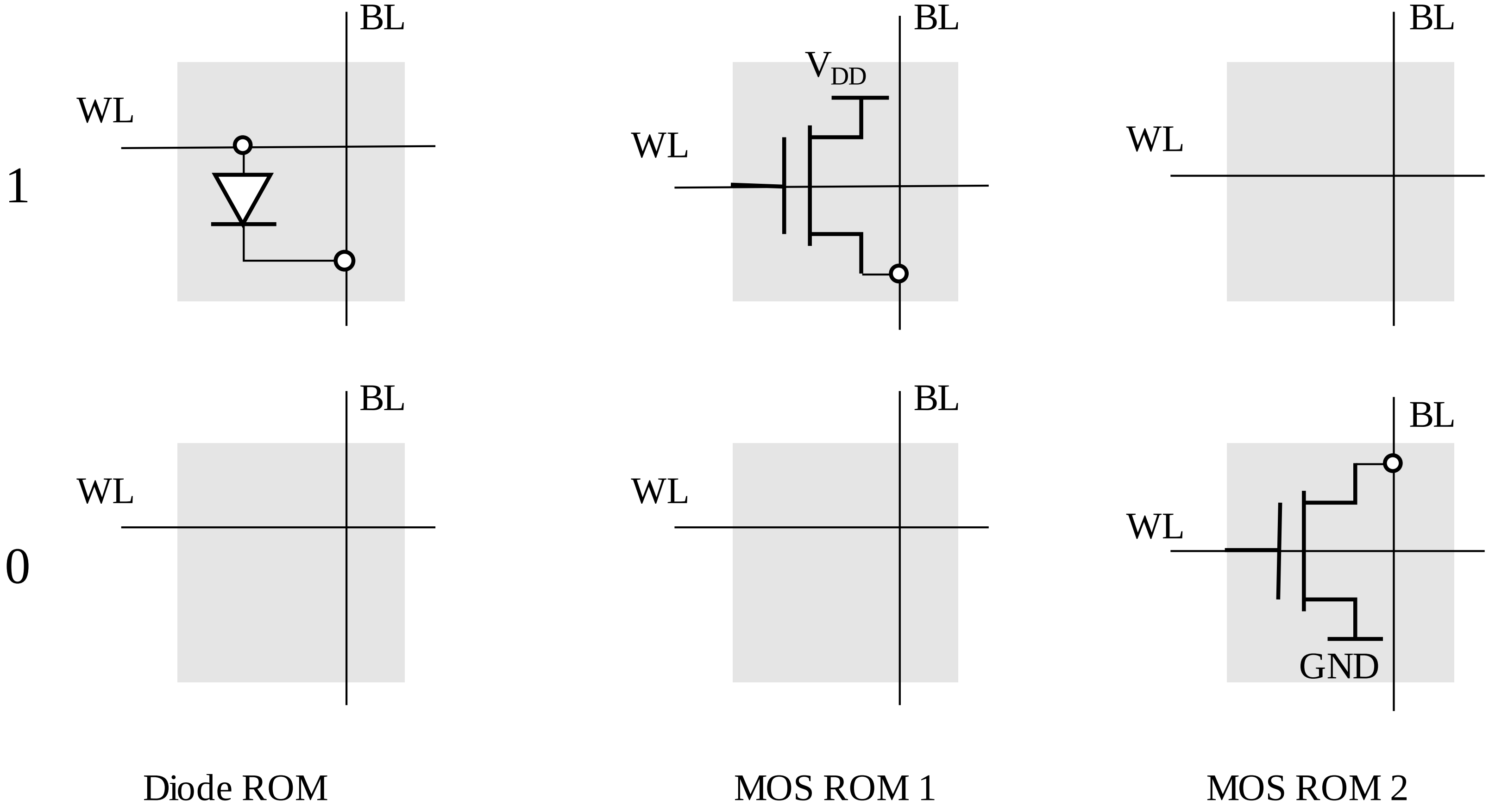


ROM Block diagram 2^k xn Module

Read-Only Memory Cells

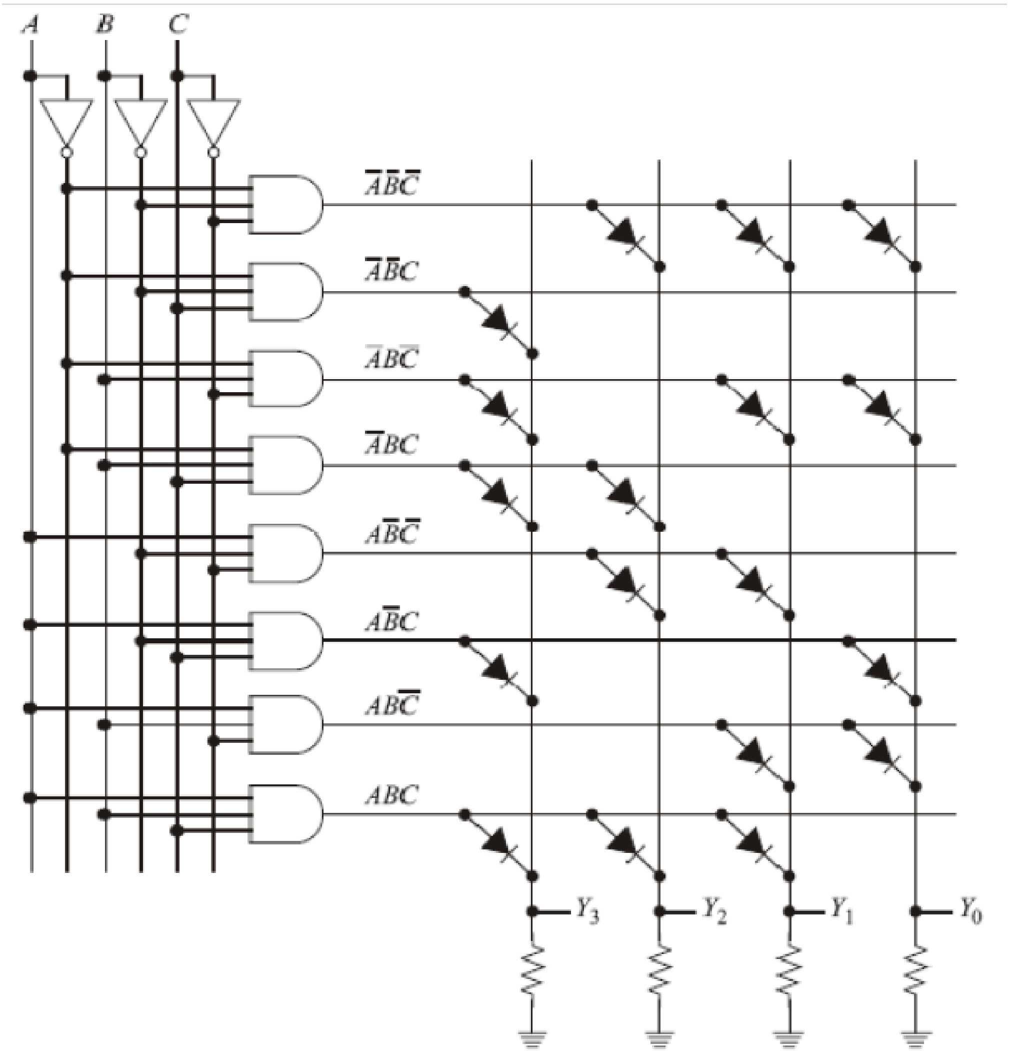
Bit line (BL) is resistively clamped to the ground, so its default value is 0
Diode disadvantage – no electrical isolation between bit and word lines

BL is resistively clamped to VDD, so its default value is 1



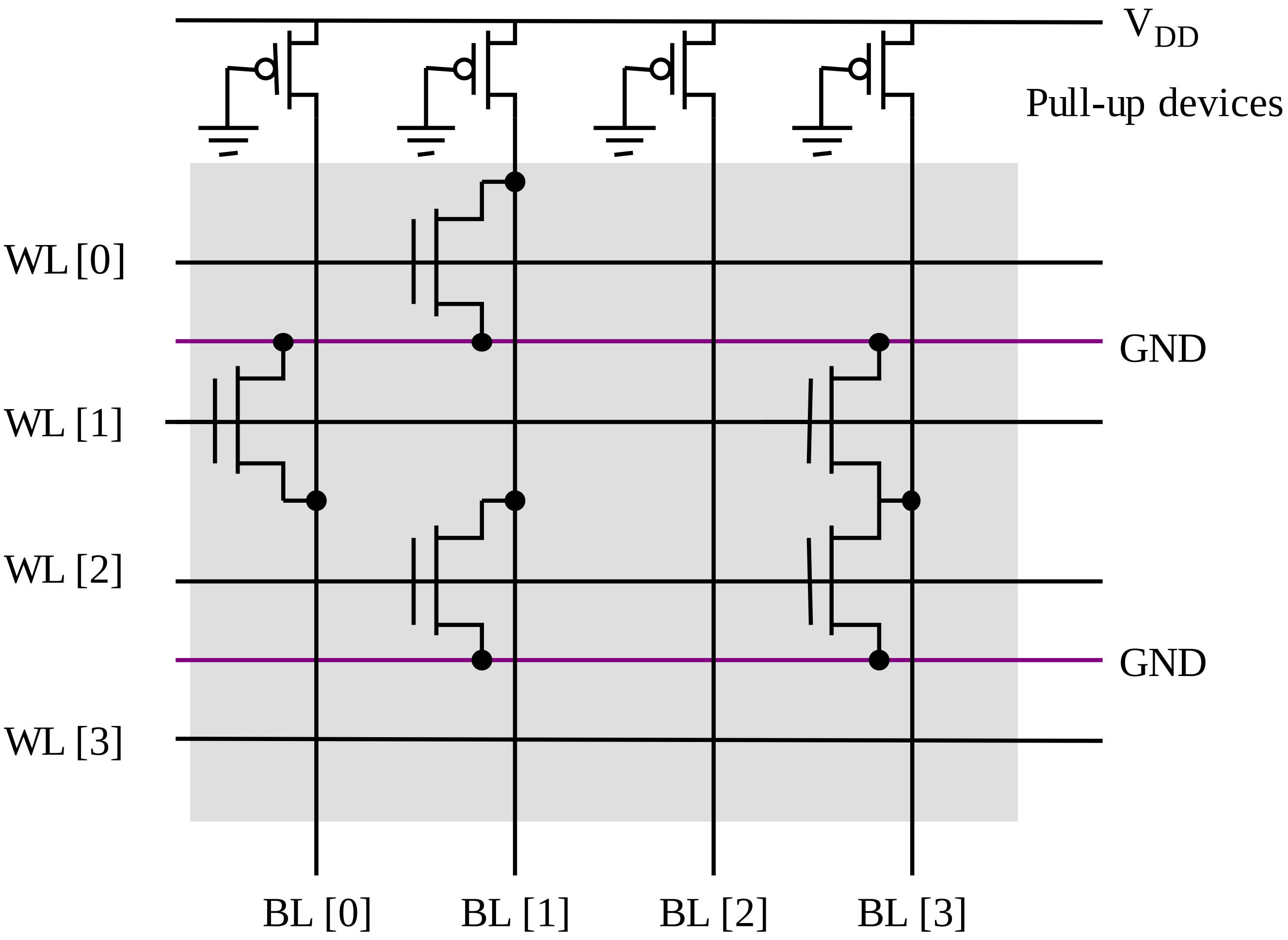
Nonvolatile Memory

ROM
PROM
EPROM

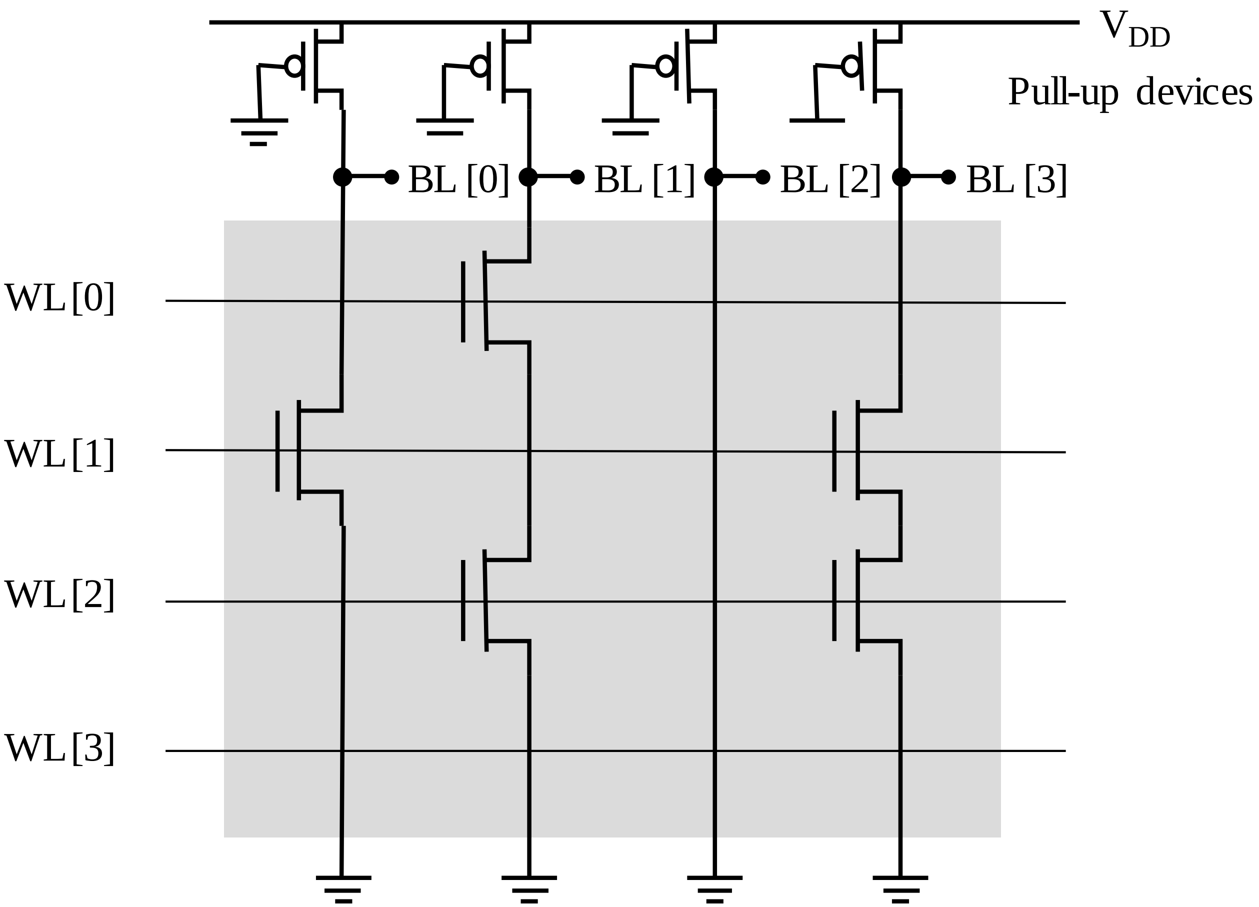


Read-only memory organization, with the fixed contents shown on the right.

MOS NOR ROM

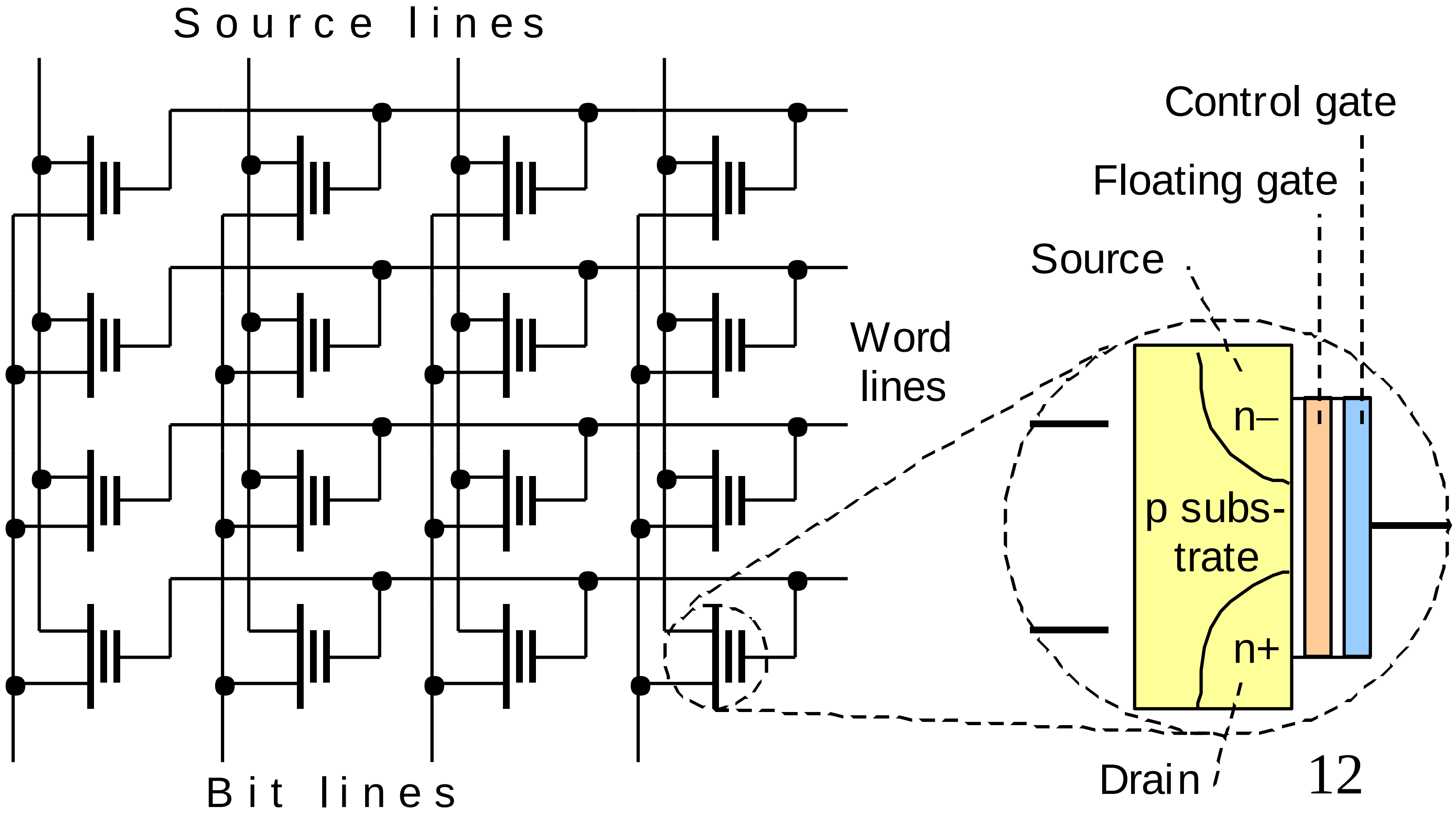


MOS NAND ROM



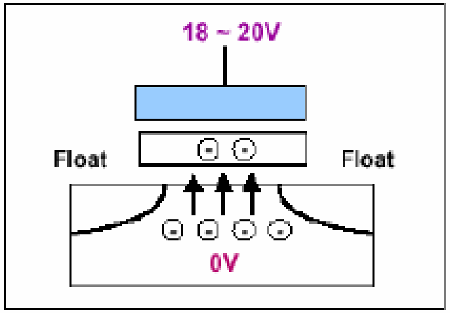
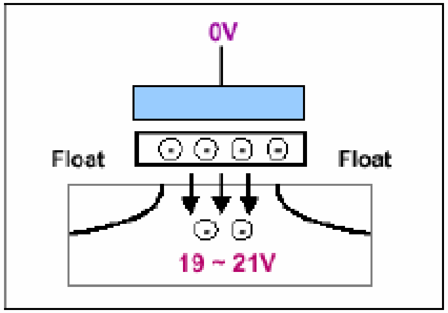
All word lines high by default with exception of selected row

Flash Memory

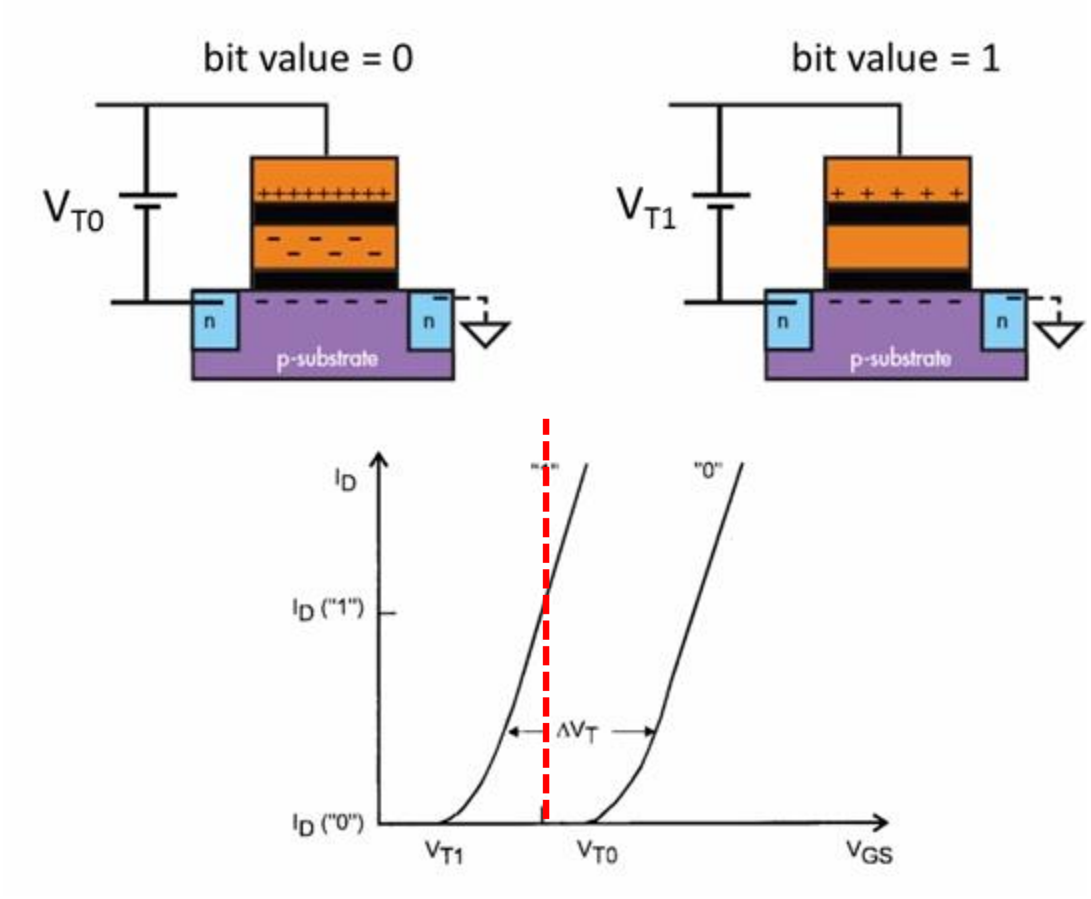


Flash memory

- Flash memory
  - Non volatile read only memory (ROM)
  - Erase Electrically or UV (EPROM)
  - Uses F-N tunneling for program & erase
  - Reads like DRAM (~ns)
  - Writes like DISK (~ms).

Program	Erase
	
● Use F-N Tunneling ● Channel Inversion	● Use F-N Tunneling ● Channel Accumulation

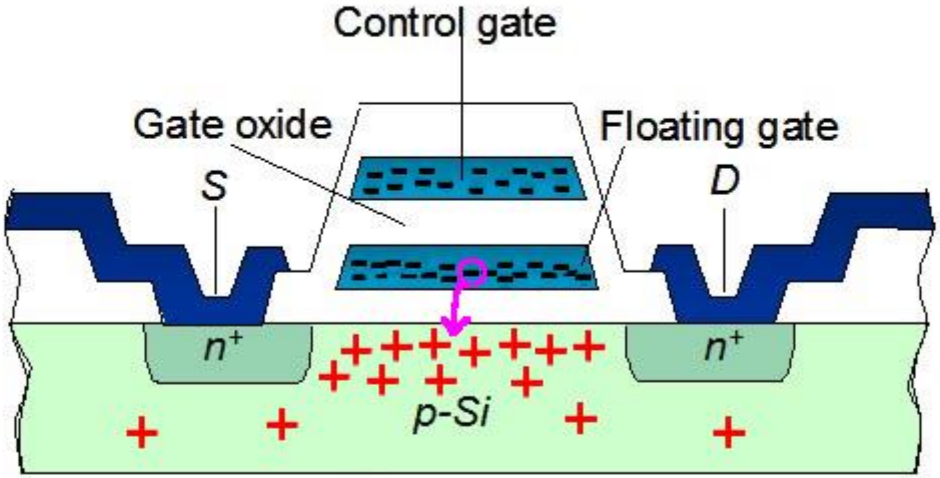
Reading Memory State



Change in Threshold Voltage due to **Screening Effect** of Floating Gate

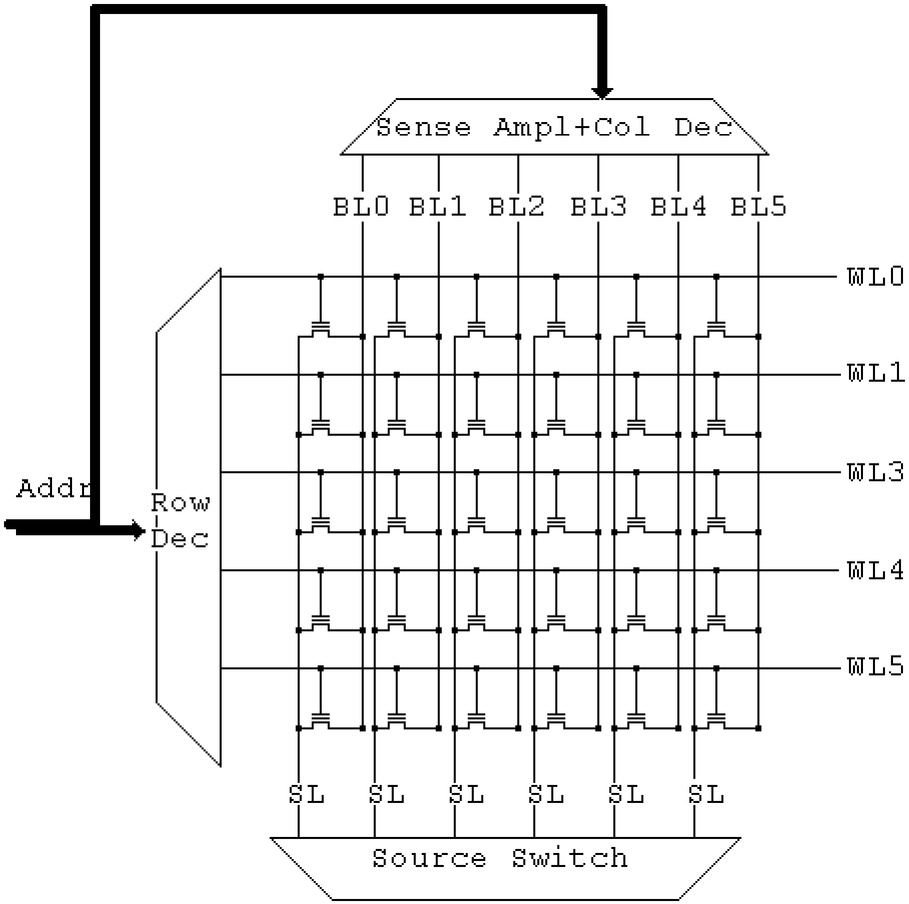
Read mode: Apply intermediate voltage, check whether current is flowing or not

Writing Memory State



Control gate voltage determines whether electrons are injected to, or push/pulled out of floating gate.

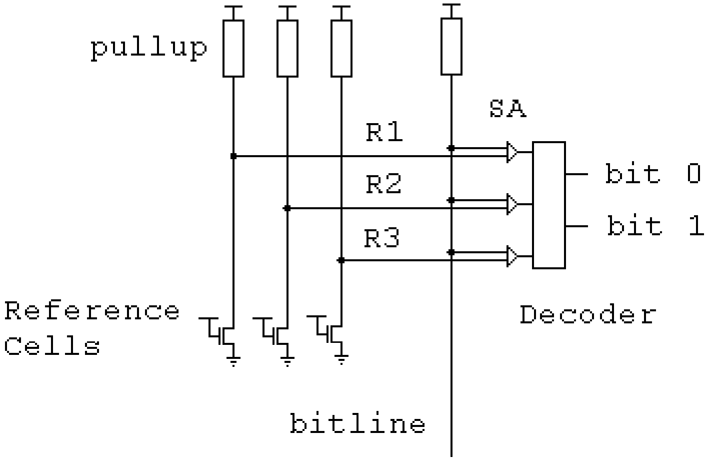
NOR Array



Reading:

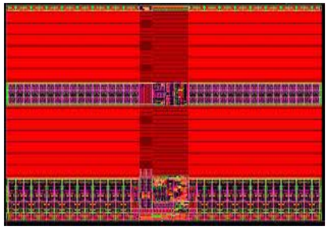
Assert a single word line. The source lines are asserted and the read of the bitline gives the contents of the cell.

Multi-Levels



- By using reference cells set at given levels and comparing them to the value from the bitline, we can determine the value stored.

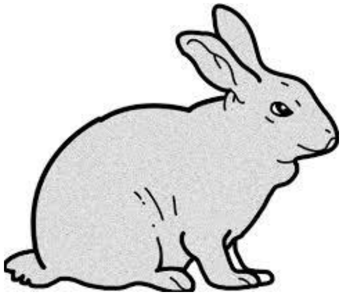
Review of memory technologies



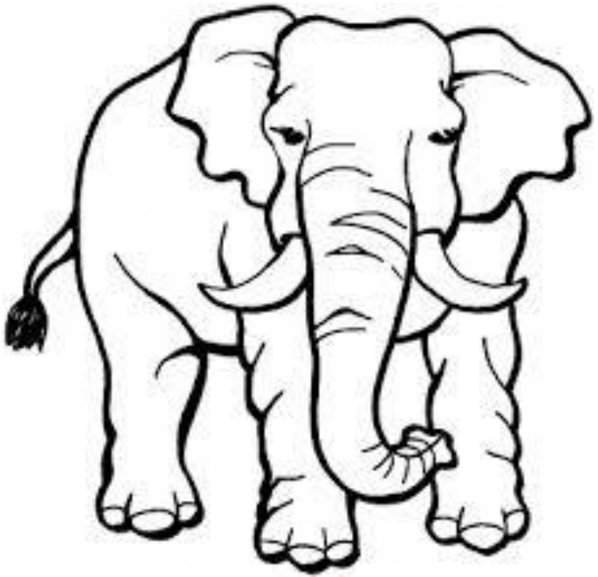
Memory type	SRAM	DRAM	Flash
Speed	Very fast	Slow	Slow
Density	Low	High	Very high
Power	High	Low	Very low
Refresh	No	Yes	No
Mechanism	Bi-stable latch	Capacitor	FN tunneling

Memory Hierarchy

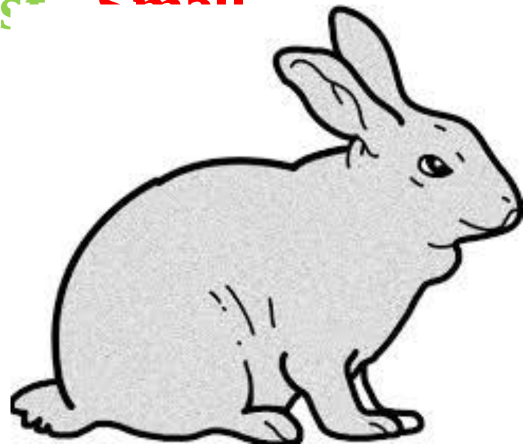
Actual Memory Systems



+



Type1: Fast, Small



Can we achieve?: Fast, Large

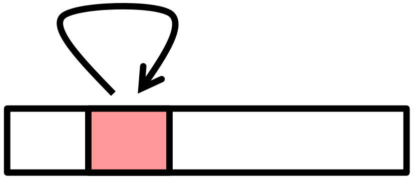
Type2:Slow, Large

(Cache Principle)

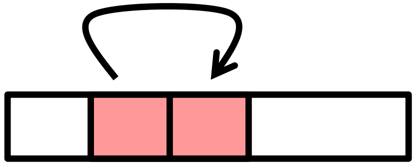
Locality

- Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

- Temporal locality:
  - Recently referenced items are likely to be referenced again in the near future



- Spatial locality:
  - Items with nearby addresses tend to be referenced close together in time



Locality Example

```
sum = 0;
for (i = 0; i < n; i++)
     sum += a[i];
return sum;
```

- Data references
  - Reference array elements in succession

    Spatial locality
  - Reference variable `sum` each iteration.

    Temporal locality
- Instruction references
  - Reference instructions in sequence.

    Spatial locality
  - Cycle through loop repeatedly.

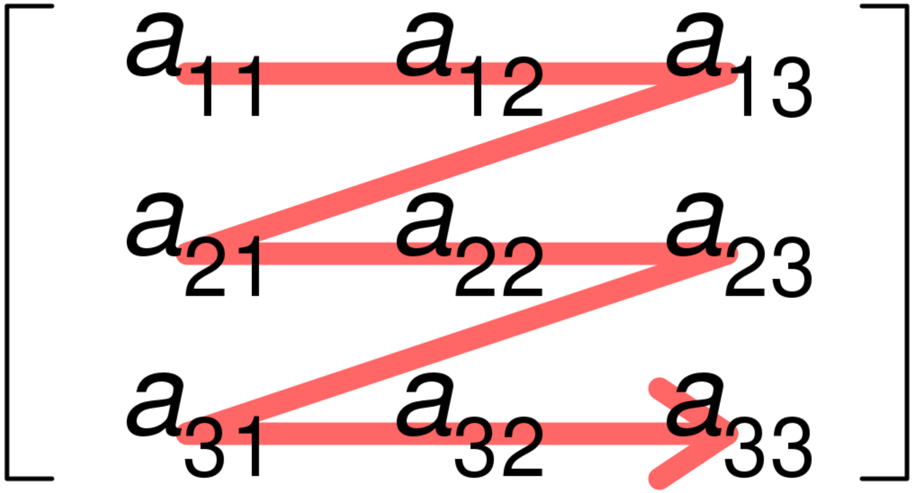
    Temporal locality

Qualitative Estimates of Locality

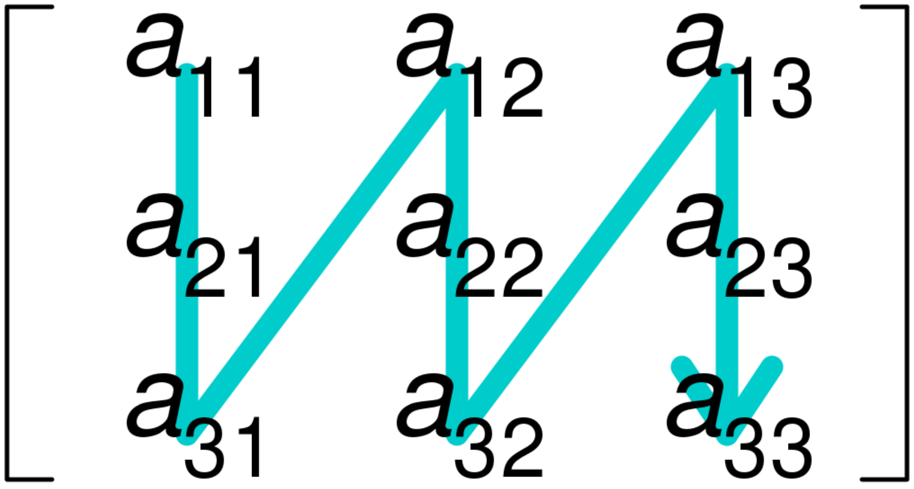
- Claim: Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.
- Question: Does this function have good locality with respect to array a?

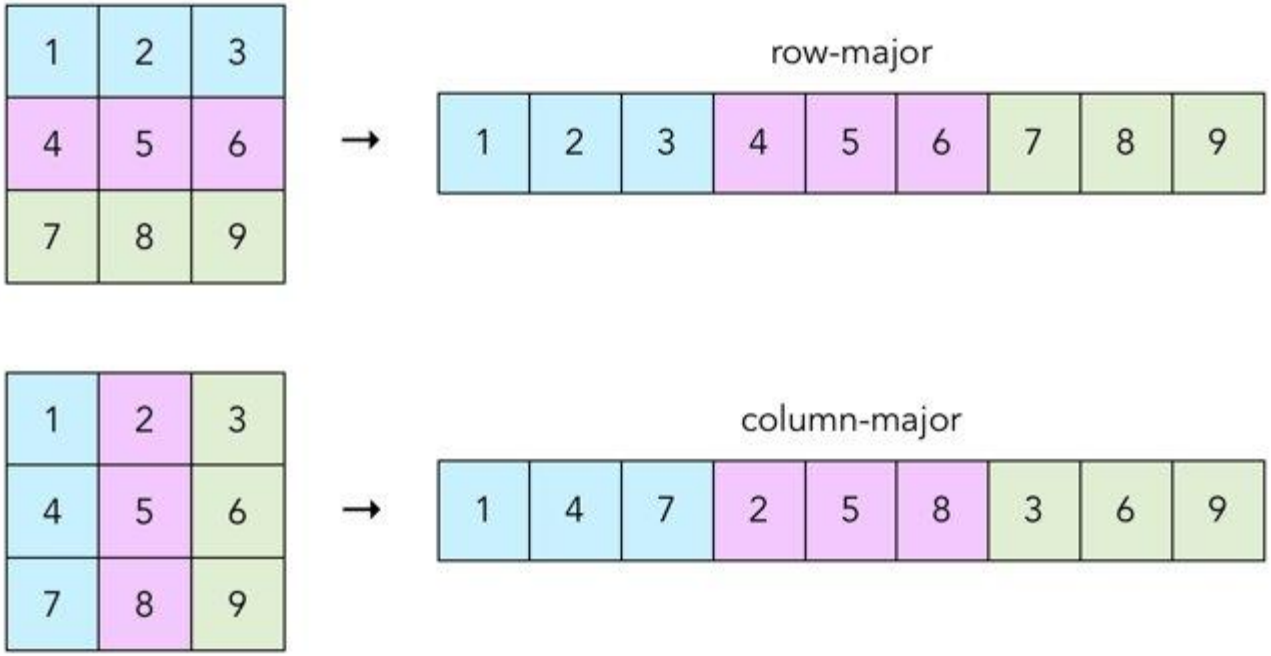
```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Row-major order


a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}

Column-major order


a_{11} a_{12} a_{13}
a_{21} a_{22} a_{23}
a_{31} a_{32} a_{33}

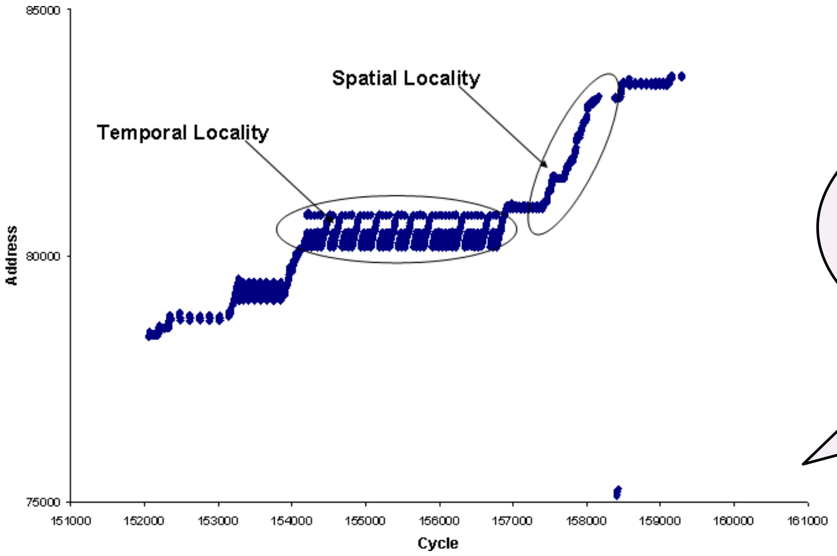

1 2 3
4 5 6
7 8 9
→
row-major
1 2 3 4 5 6 7 8 9
1 2 3
4 5 6
7 8 9
→
column-major
1 4 7 2 5 8 3 6 9

Locality Example

- Question: Does this function have good locality with respect to array `a`?

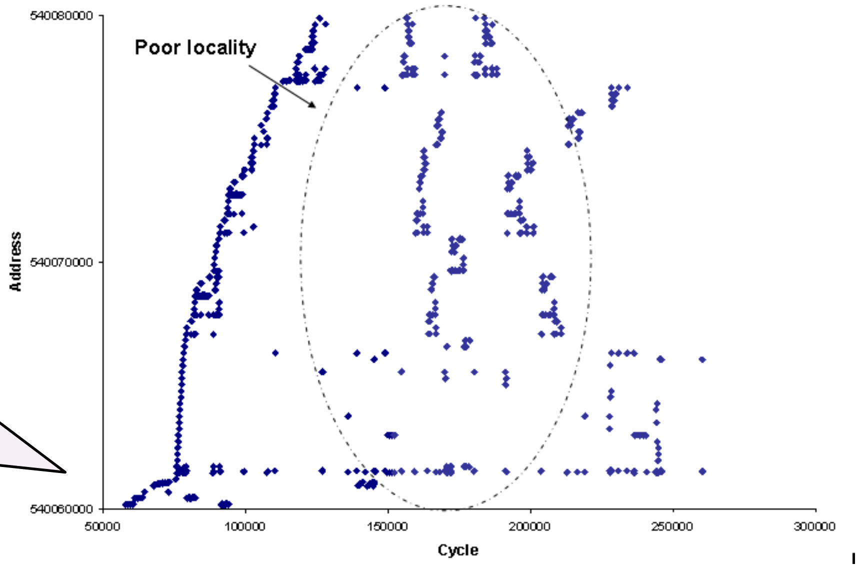
```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Locality reference example



Good locality example:
The X axis corresponds to time (in cycles)
and the Y axis corresponds to memory locations.
The highlighted sections show examples
of temporal and spatial locality.

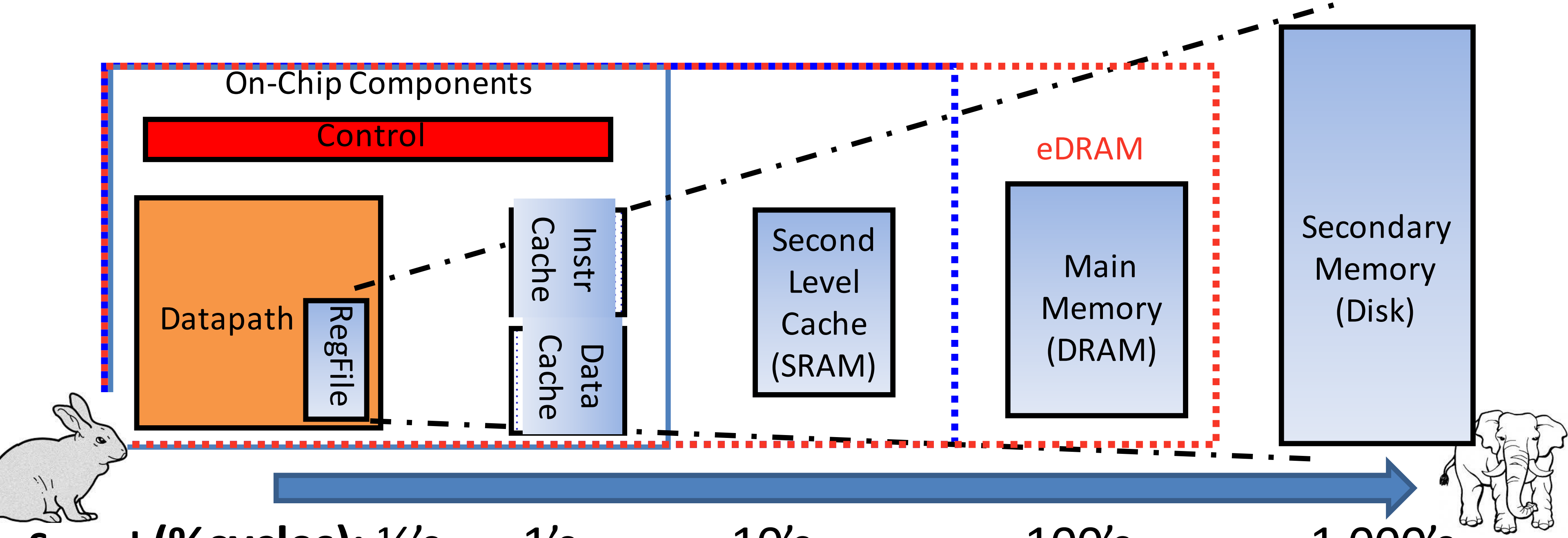
Poor locality example:
We can see that the program accesses a
large memory footprint without a specific order,
giving the pattern poor spatial locality.
We also see little temporal re-use,
which also makes this code cache unfriendly.



Source: http://www.eetimes.com/document.asp?doc_id=1275470

The Memory Hierarchy

- By taking advantage of the principle of locality
  - Can present the user with as much memory as is available in the cheapest technology
  - at the speed offered by the fastest technology



Speed (%cycles):	½'s	1's	10's	100's	1,000's
Size (bytes):	100's	K's	1M's	10G's	10 G's to T's
Speed(ns):	0.5ns	2ns	6ns	100ns	10ms

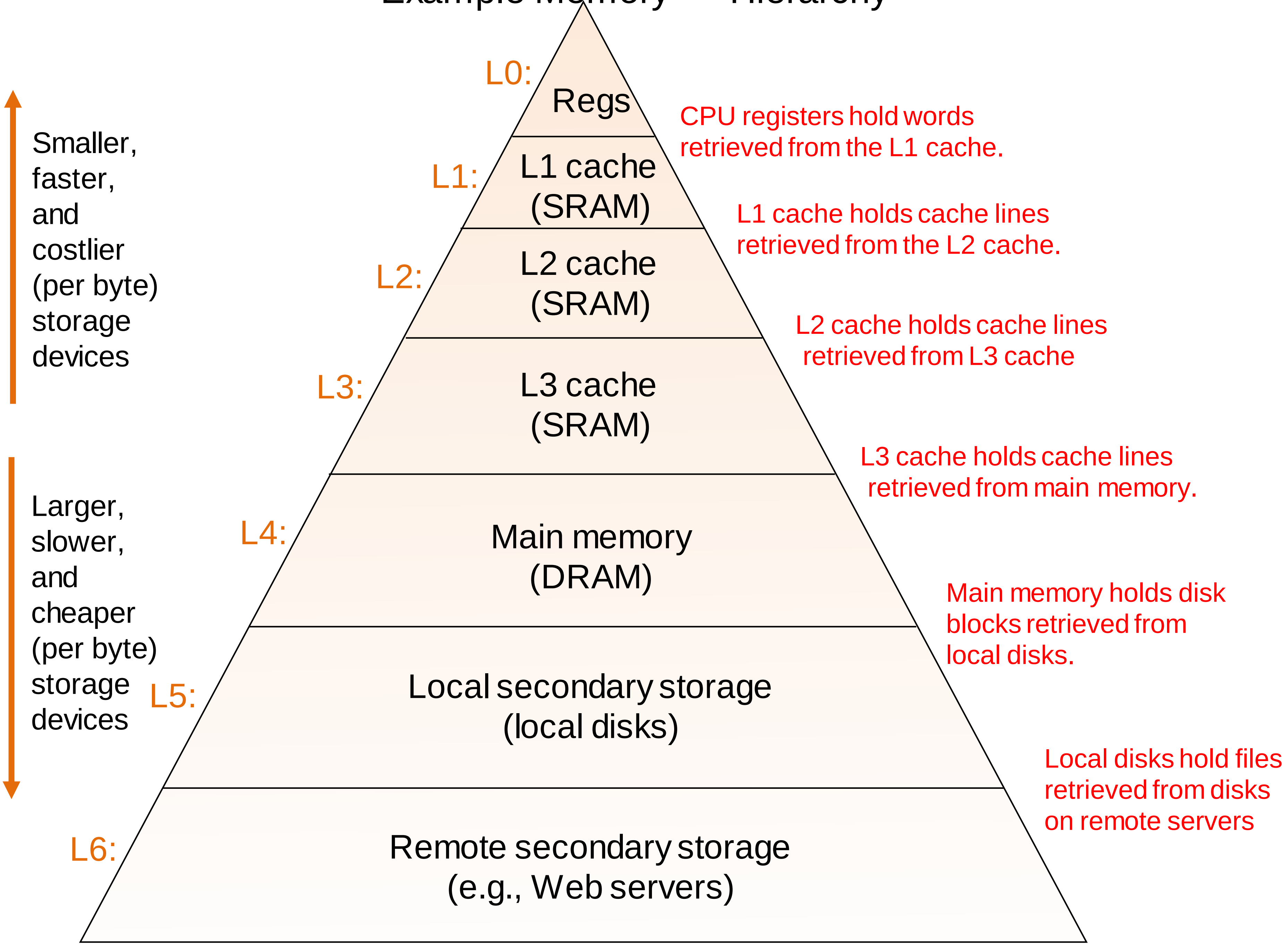
$$$$ $

Cache Design

Maurice Wilkes, “Slave Memories and Dynamic Storage Allocation,” IEEE Trans. On Electronic Computers, 1965

Awards: Turing Award, IEEE John von Neumann Medal, Faraday Medal, Eckert–Mauchly Award, Mountbatten Medal

Example Memory Hierarchy



Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices

L0: Regs

L1: L1 cache (SRAM)

L2: L2 cache (SRAM)

L3: L3 cache (SRAM)

L4: Main memory (DRAM)

L5: Local secondary storage (local disks)

L6: Remote secondary storage (e.g., Web servers)

CPU registers hold words retrieved from the L1 cache.

L1 cache holds cache lines retrieved from the L2 cache.

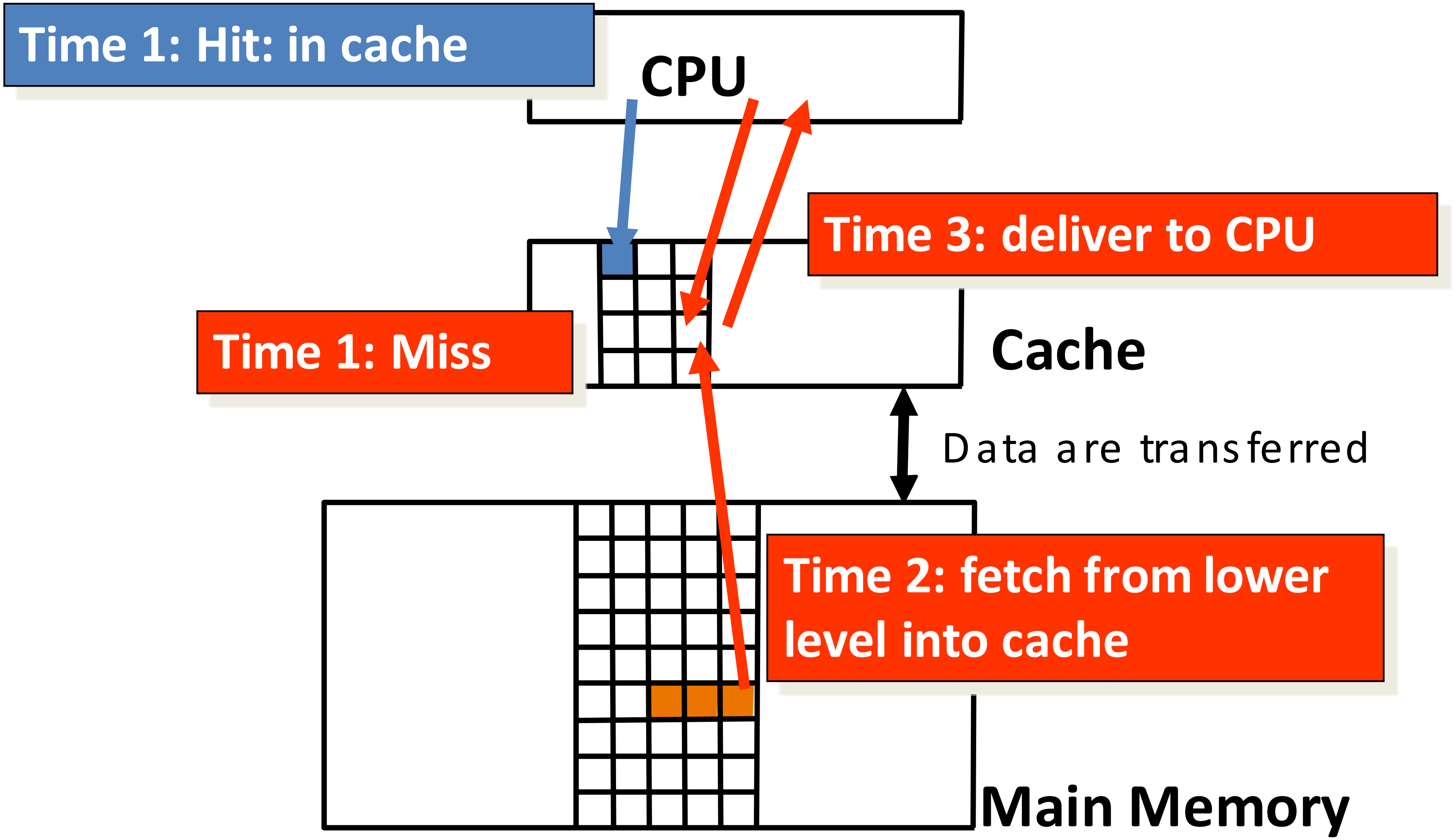
L2 cache holds cache lines retrieved from L3 cache

L3 cache holds cache lines retrieved from main memory.

Main memory holds disk blocks retrieved from local disks.

Local disks hold files retrieved from disks on remote servers

Cache Example



Average Memory Access Time = hit time (t_c) + miss rate (m) * miss penalty(t_p)

$$= t_c + m * t_p$$

Hit time = Time 1 (tc)

Miss penalty (tp)= Time 2 + Time 3

Example: Impact on Performance

- **Suppose a processor executes at**
  - Clock Rate = 1000 MHz (1 ns per cycle)
  - CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control
- Suppose that 10% of memory operations get 50 cycle miss penalty
- CPI = ideal CPI + average stalls per instruction
  = 1.1(cyc) +(0.30 (datamops/ins)
  x 0.10 (miss/datamop) x 50 (cycle/miss))
  = 1.1 cycle + **0.3x0.1x50** cycle =1.1cycle + 1.5 cycle
  = 2. 6
- 58 % of the time the processor
  is stalled waiting for memory!
- a 1% instruction miss rate would add
  an additional 0.5 cycles to the CPI!

Source: David Patterson

Let’s think about those numbers

- Huge difference between a hit and a miss
  - Could be 100x, if just L1 and main memory

- Would you believe 99% hits is twice as good as 97%?
  - Consider:
    cache hit time of 1 cycle
    miss penalty of 100 cycles
  - Average access time:
    97% hits: 1 cycle + 0.03 * 100 cycles = **4 cycles**
    99% hits: 1 cycle + 0.01 * 100 cycles = **2 cycles**

- This is why “miss rate” is used instead of “hit rate”

Hierarchical Latency Analysis

- For a given memory hierarchy level i it has a technology-intrinsic access time of $t_{i,}$ The perceived access time T_i is longer than t_i
- Except for the outer-most hierarchy, when looking for a given address there is
  - a chance (hit-rate h_i) you "hit" and access time is t_i
  - a chance (miss-rate m_i) you "miss" and access time $t_i + T_{i+1}$
  - $h_i + m_i = 1$
- Thus

$$T_i = h_i \cdot t_i + m_i \cdot (t_i + T_{i+1})$$

$$T_i = t_i + m_i \cdot T_{i+1}$$

keep in mind, h_i and m_i are defined to be the hit-rate and miss-rate of just the references that missed at L_{i-1}

Hierarchy Design Considerations

- Recursive latency equation

$$T_i = t_i + m_i \cdot T_{i+1}$$

- The goal: achieve desired T_1 within allowed cost
- $T_i \approx t_i$ is desirable

- Keep m_i low
  - increasing capacity C_i lowers m_i, but beware of increasing t_i
  - lower m_i by smarter management

- Keep T_{i+1} low
  - faster lower hierarchies, but beware of increasing cost
  - introduce intermediate hierarchies as a compromise

Intel Pentium 4 Example

- 90nm P4, 3.6 GHz
- L1 D-cache
  - C_1 = 16K
  - t_1 = 4 cyc int / 9 cycle fp
- L2 D-cache
  - C_2 =1024 KB
  - t_2 = 18 cyc int / 18 cyc fp
- Main memory
  - t_3 = ~ 50ns or 180 cyc
- Notice
  - best case latency is not 1
  - worst case access latencies are into 500+ cycles

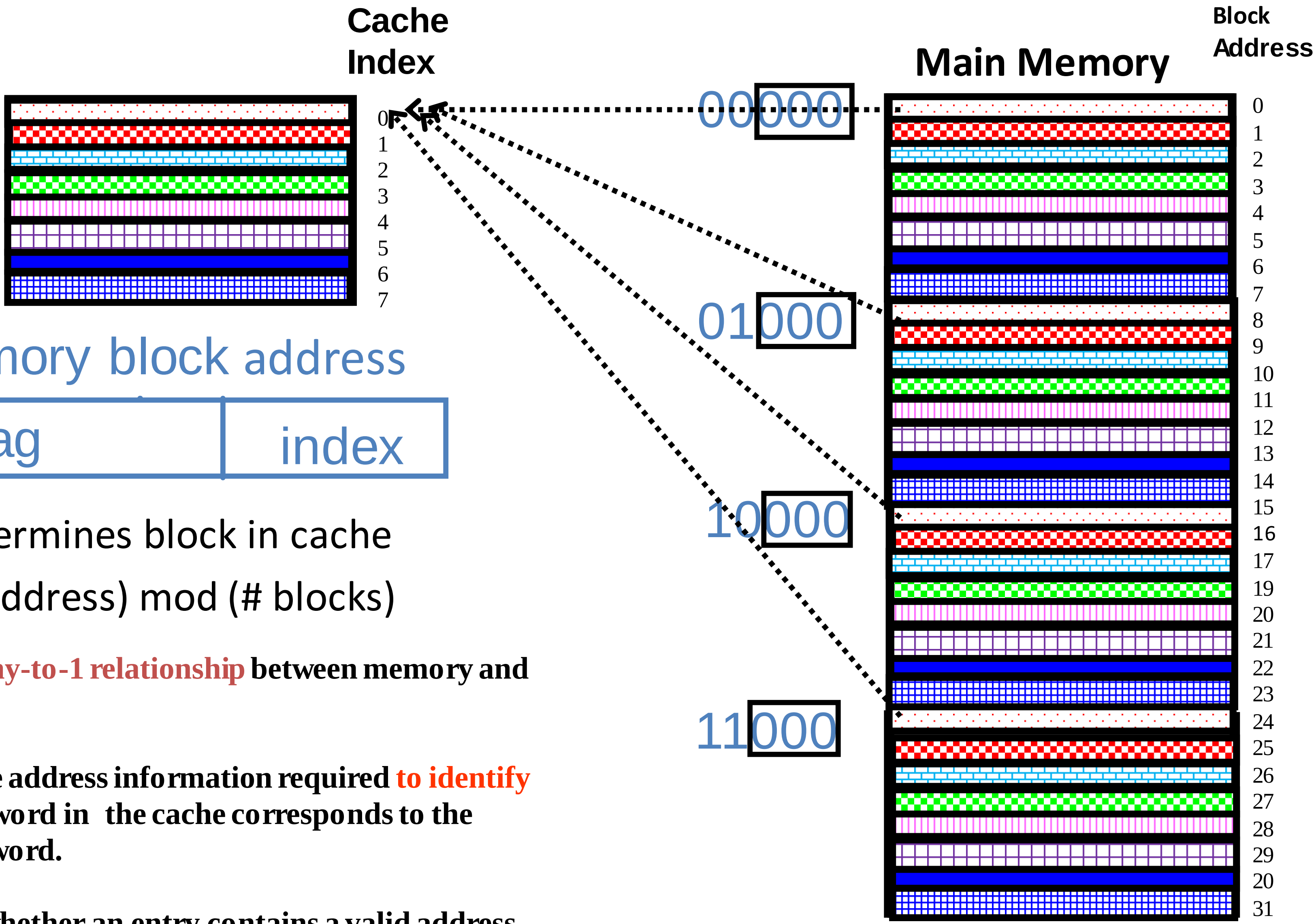
if m_1=0.1, m_2=0.1
T_1=7.6, T_2=36

if m_1=0.01, m_2=0.01
T_1=4.2, T_2=19.8

if m_1=0.05, m_2=0.01
T_1=5.00, T_2=19.8

if m_1=0.01, m_2=0.50
T_1=5.08, T_2=108

Simplest Cache: Direct Mapped



Memory block address

tag	index

- index determines block in cache
- index = (address) mod (# blocks)

There is a Many-to-1 relationship between memory and cache

tags

- contain the address information required to identify whether a word in the cache corresponds to the requested word.

valid bit

- indicates whether an entry contains a valid address

Cache Table Entry

V	tag	Data

Direct Mapped Cache: Temporal Example

lw	$1,10 110 ($0)	Miss: valid	lw	$1,22($0)
lw	$2,11 010 ($0)	Miss: valid	lw	$2,26($0)
lw	$3,10 110 ($0)	Hit!	lw	$3,22($0)

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

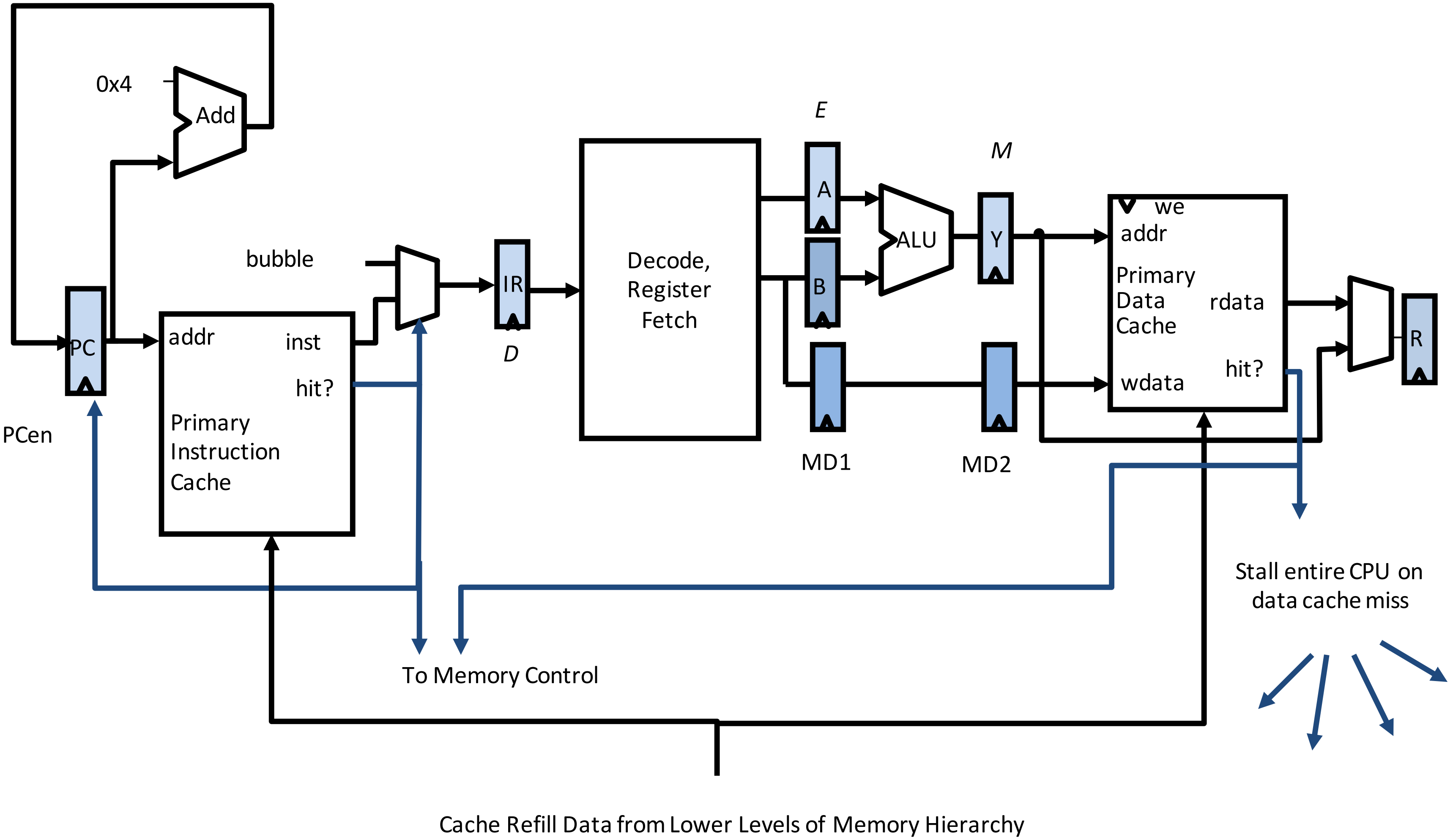
Direct Mapped Cache: Worst case, always miss!

lw	$1,10 110 ($0)	Miss: valid	lw	$1,22($0)
lw	$2,11 110 ($0)	Miss: tag	lw	$2,30($0)
lw	$3,00 110 ($0)	Miss: tag	lw	$3,6($0)

Index	Valid	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	00	Memory[00110]
111	N		

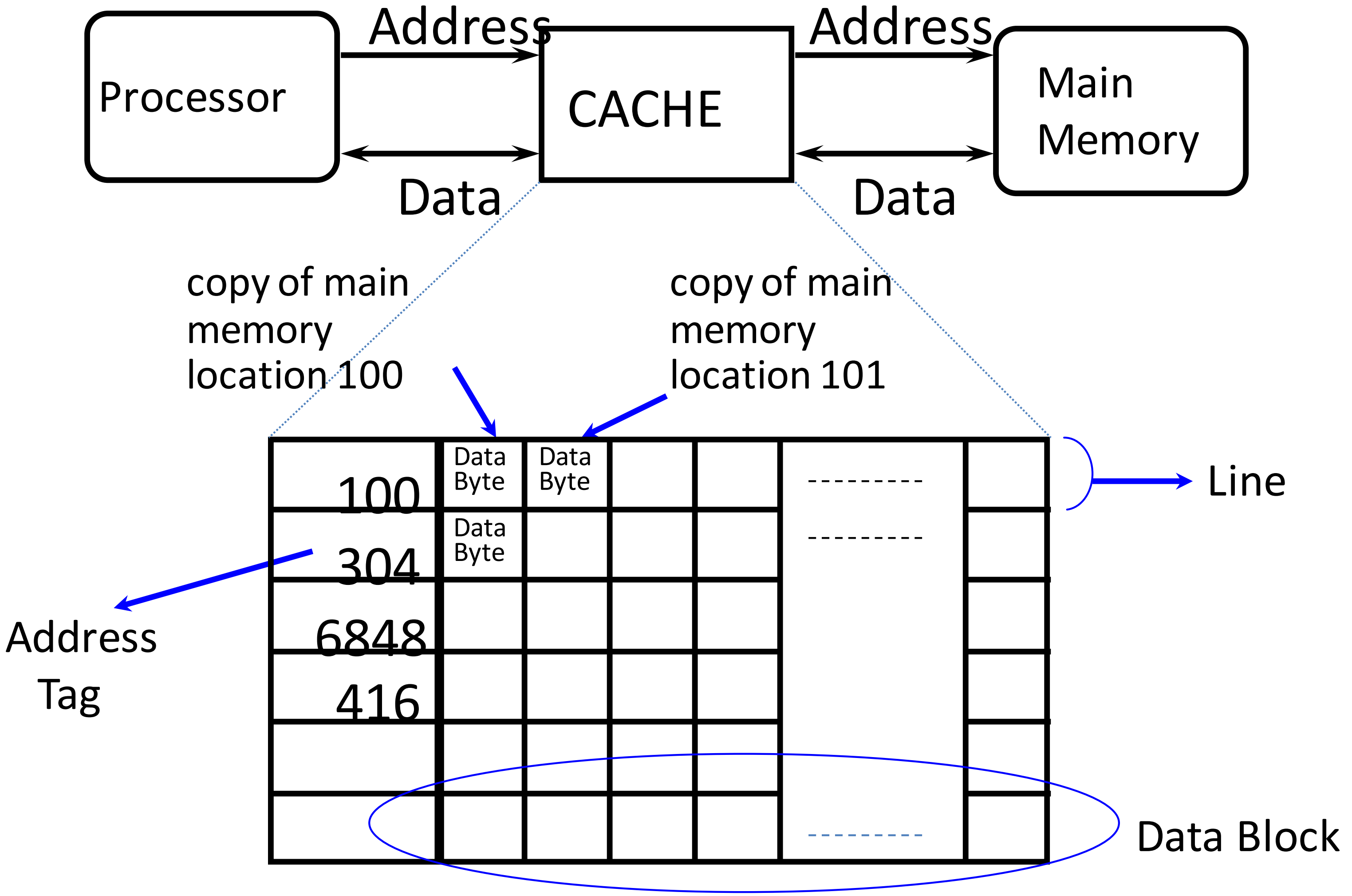
CPU-Cache Interaction

(5-stage pipeline)



Stall entire CPU on data cache miss

Inside a Cache



Processor

Address

Data

CACHE

Address

Data

Main Memory

copy of main memory location 100

copy of main memory location 101

100

304

6848

416

Data Byte

Data Byte

Data Byte

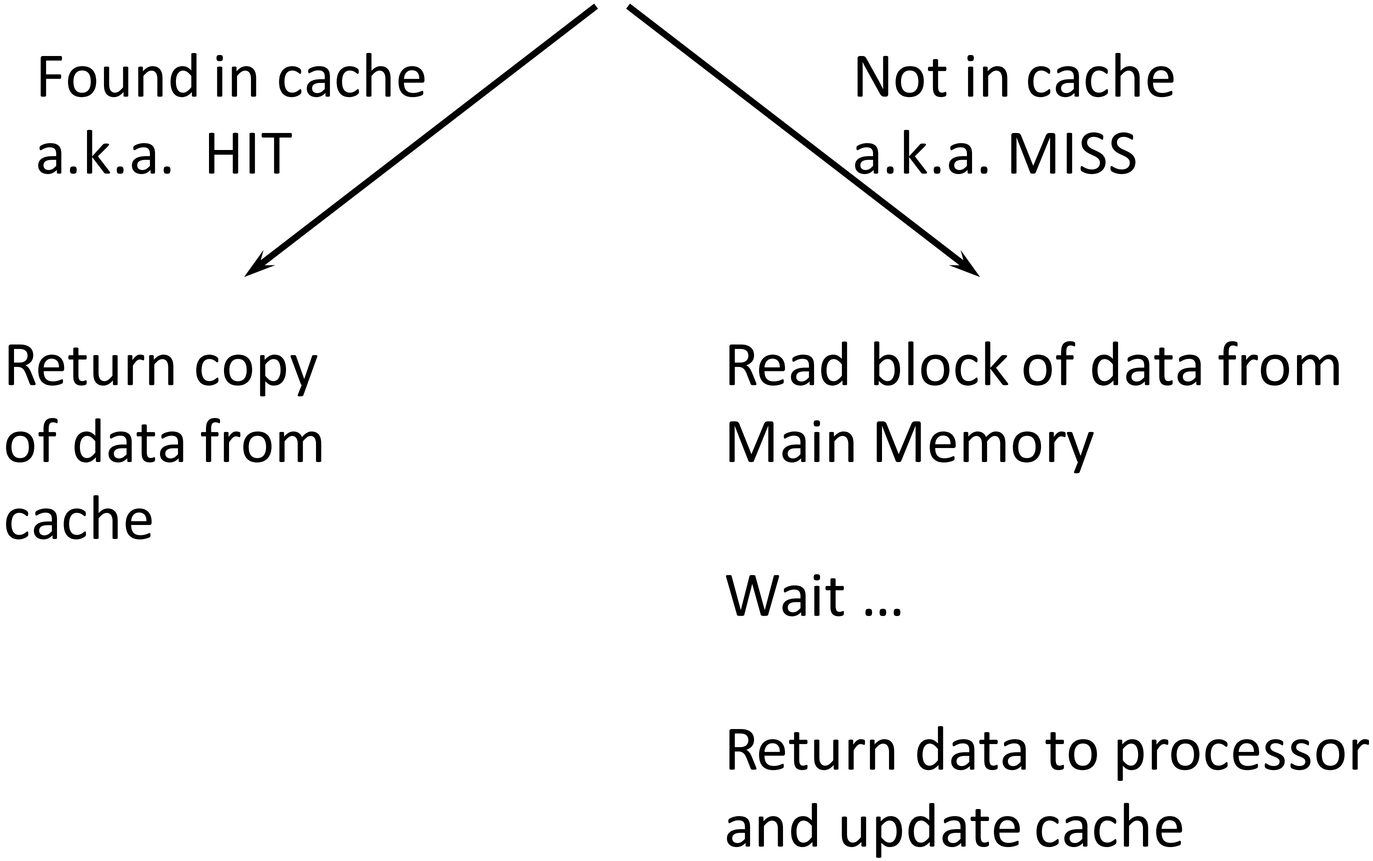
Line

Address Tag

Data Block

Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either



Found in cache
a.k.a. HIT

Not in cache
a.k.a. MISS

Return copy of data from cache

Read block of data from Main Memory

Wait …

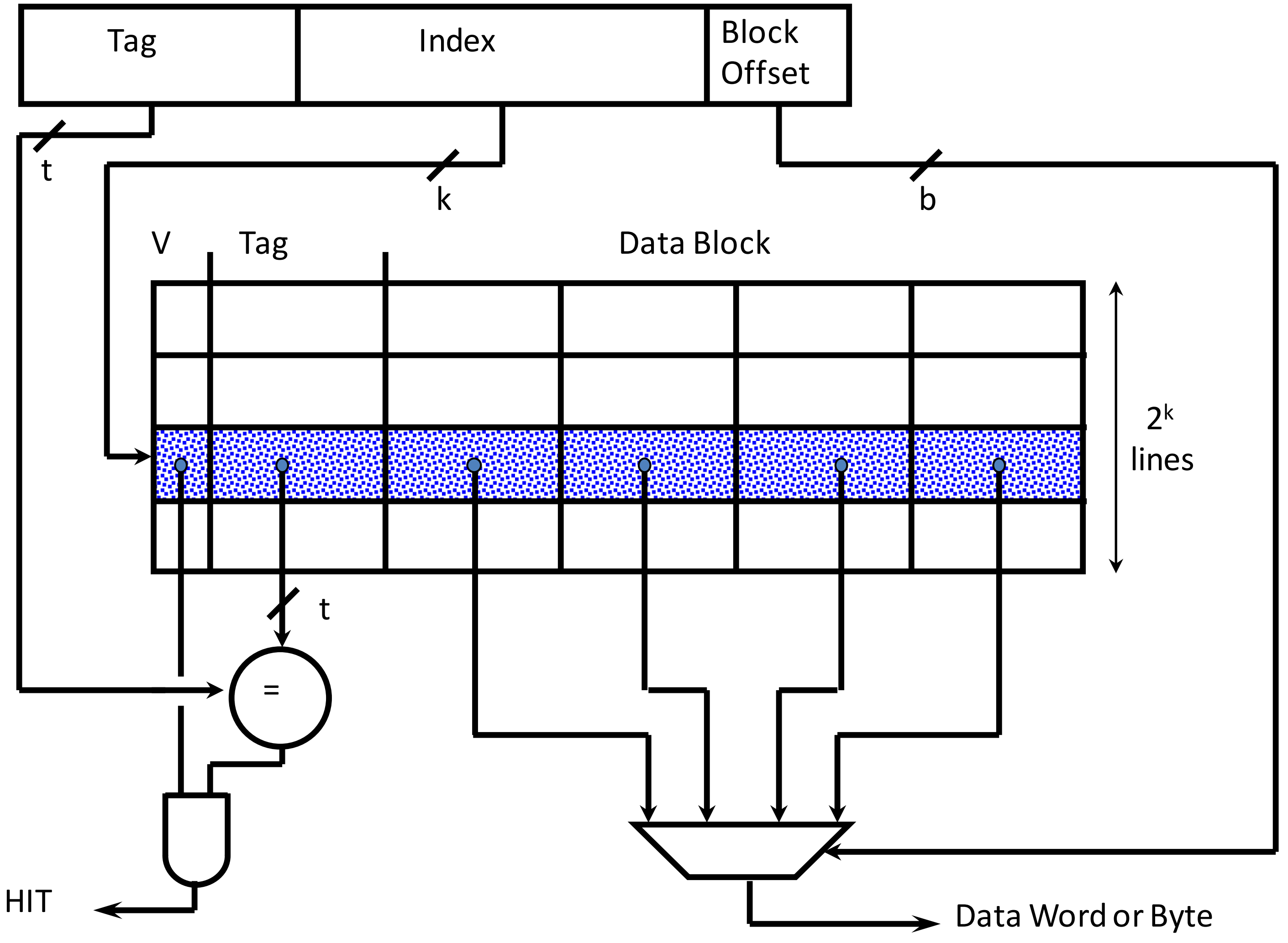
Return data to processor and update cache

Q: Which line do we replace?

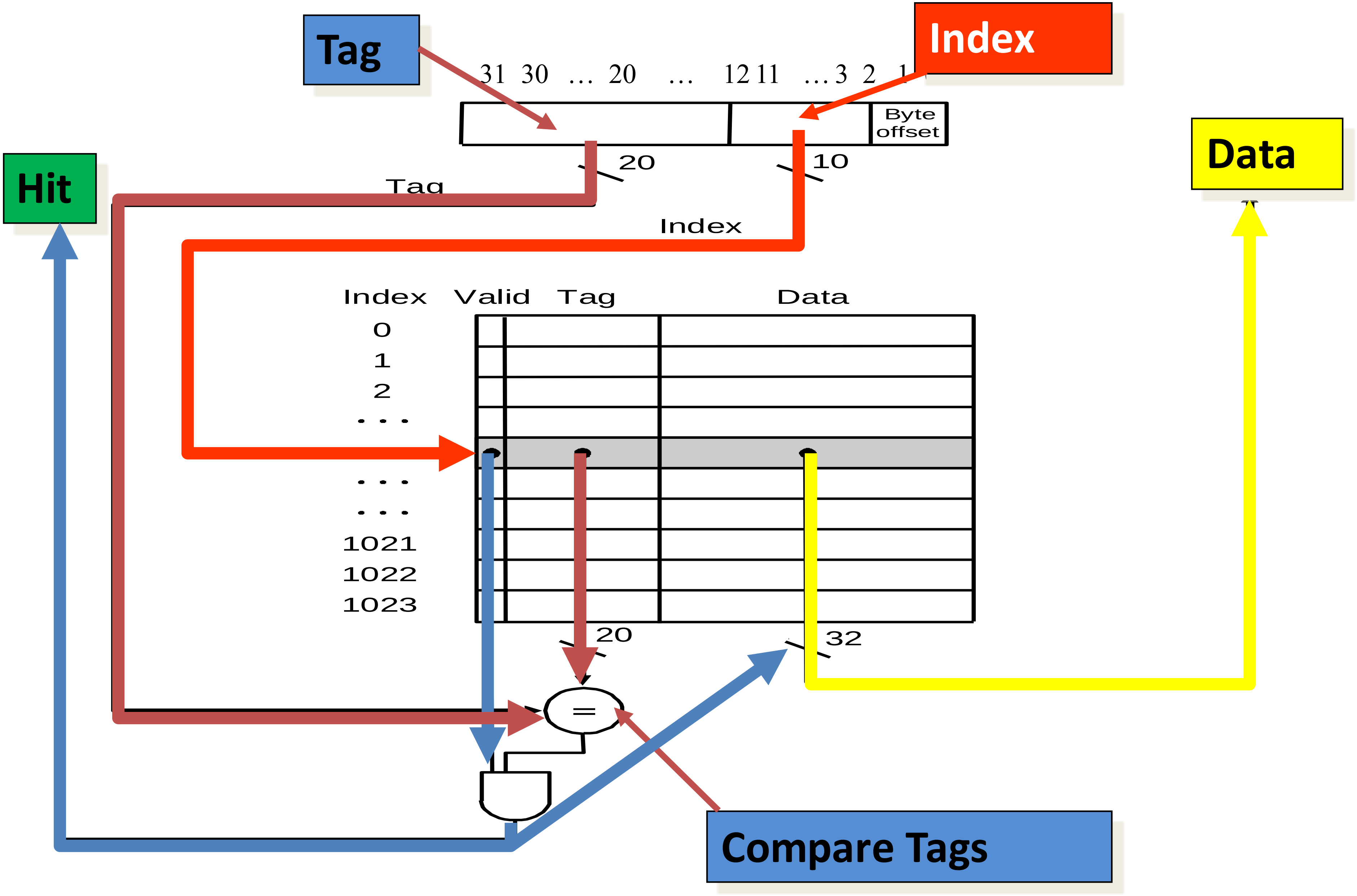
Caching Terminology

- block/line : the unit of information that can be exchanged between two adjacent levels in the memory hierarchy
- cache hit: when CPU finds a requested data item in the cache
- cache miss: when CPU does not find a requested data item in the cache
- miss rate (*m*) : fraction of cache accesses that result in a miss
- hit rate (*h*) : (1- miss rate)fraction of memory access found in the faster level (cache)
- hit time (t_c) : time required to access the level 1 (cache)
- miss penalty (t_p) : time required to replace a data unit in the faster level (cache) + time involved to deliver the requested data to the processor from the cache. (hit time << miss penalty)
- average memory access time (amat)= hit time (t_c) + miss rate (*m*) * miss penalty (t_p) = t_c + (1-*h*) * t_p

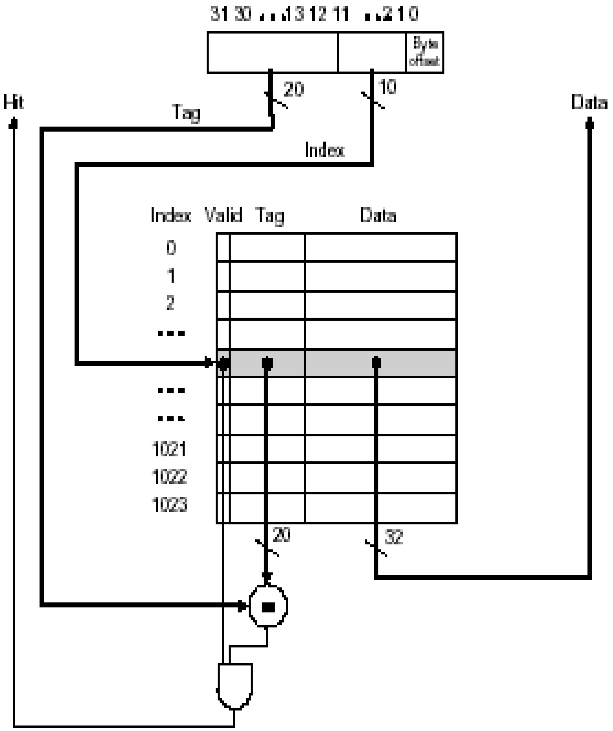
Direct-Mapped Cache



Direct Mapped Cache (4K): MIPS Example



Direct Mapped Cache [example]



- What is the size of cache ?

4K

- If I read

0000 0000 0000 0000 0000 0000 1000 0001

- What is the index number checked ?

- If the number was found, what are the inputs to comparator ?

Direct Mapped Cache [contd…]

- Advantage
  - Simple
  - Fast
- Disadvantage
  - Mapping is fixed !!!

64KB Direct Mapped Cache Example

Nominal Capacity

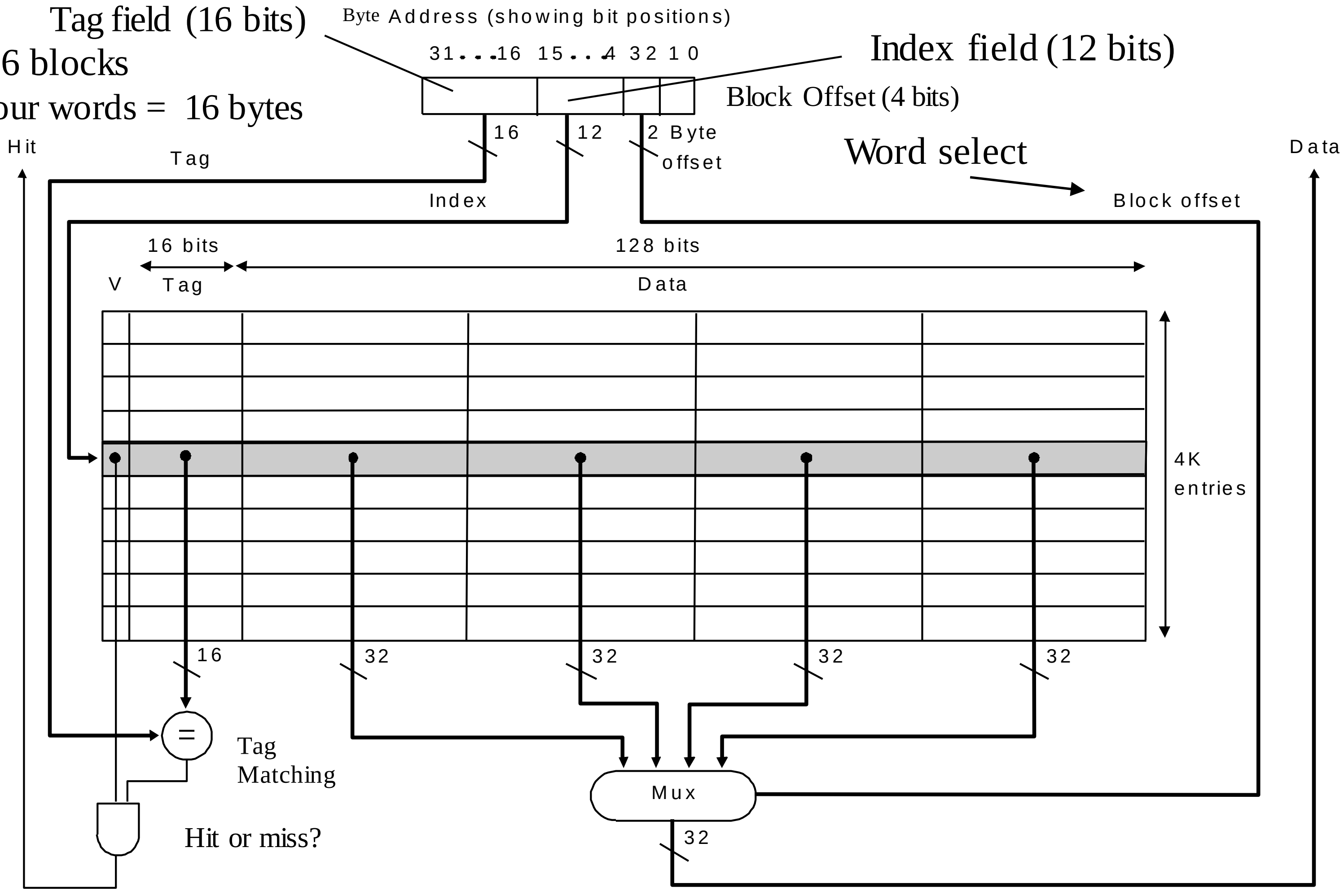
4K= 2^{12} = 4096 blocks

Each block = four words = 16 bytes

Can cache up to 2^{32} bytes = 4 GB of memory

SRAM

Typical cache Block or line size: 32-64 bytes



Block Address = 28 bits

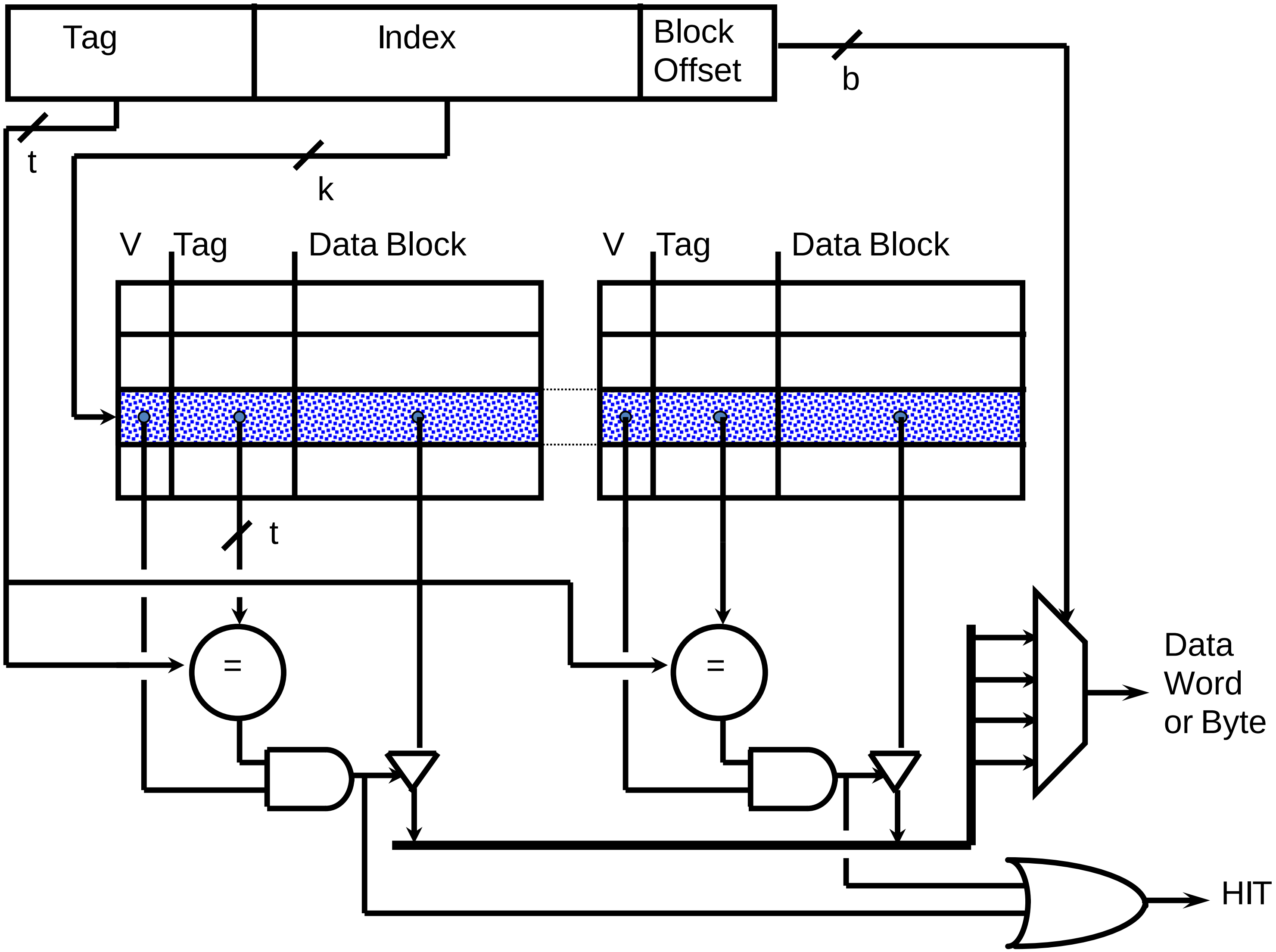
Tag = 16 bits | Index = 12 bits | Block offset = 4 bits

<u>Mapping Function:</u> Cache Block frame number = (Block address) MOD (4096)

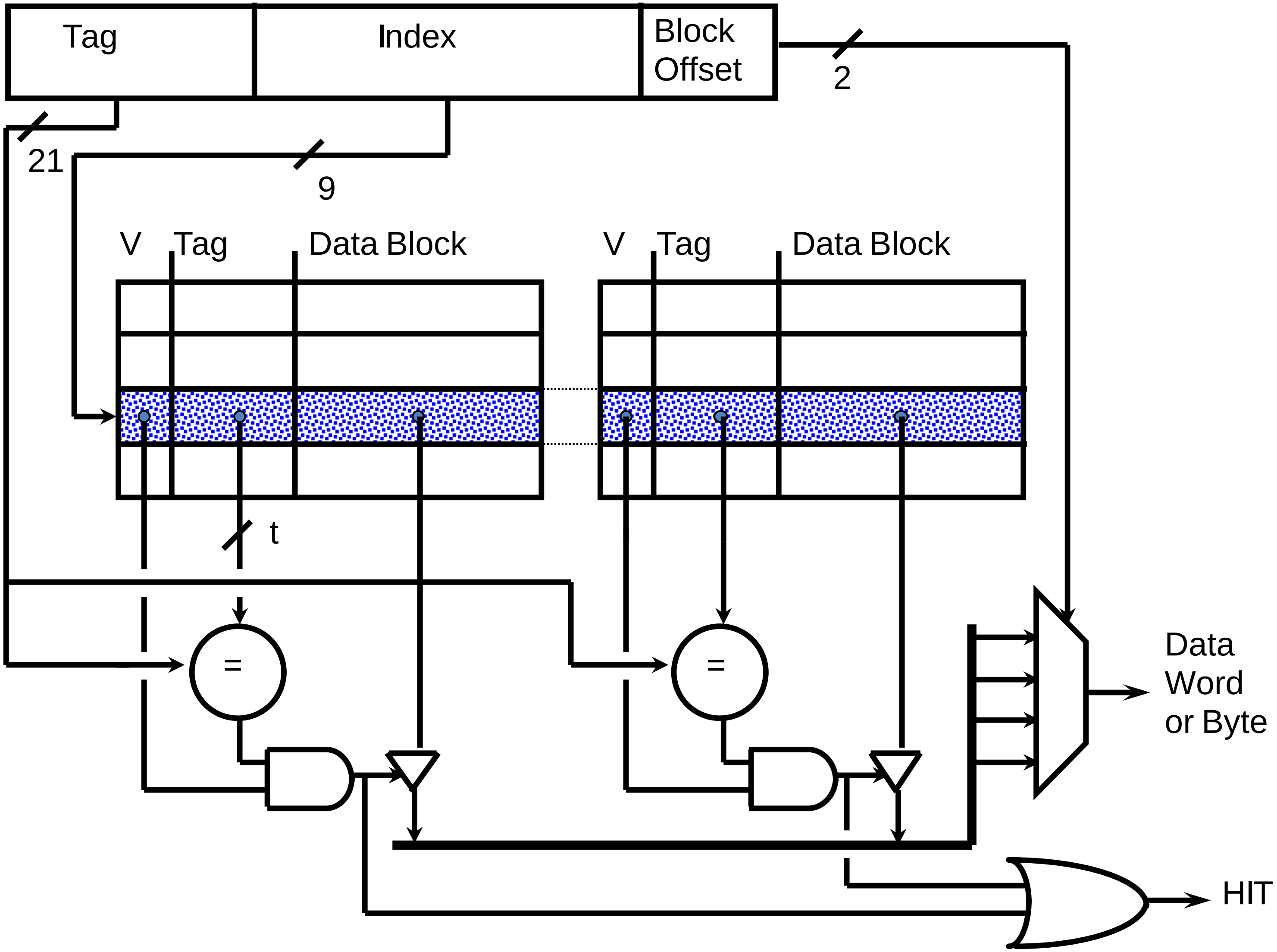
i.e. index field or 12 low bit of block address

Hit Access Time = SRAM Delay + Hit/Miss Logic Delay

2-Way Set-Associative Cache



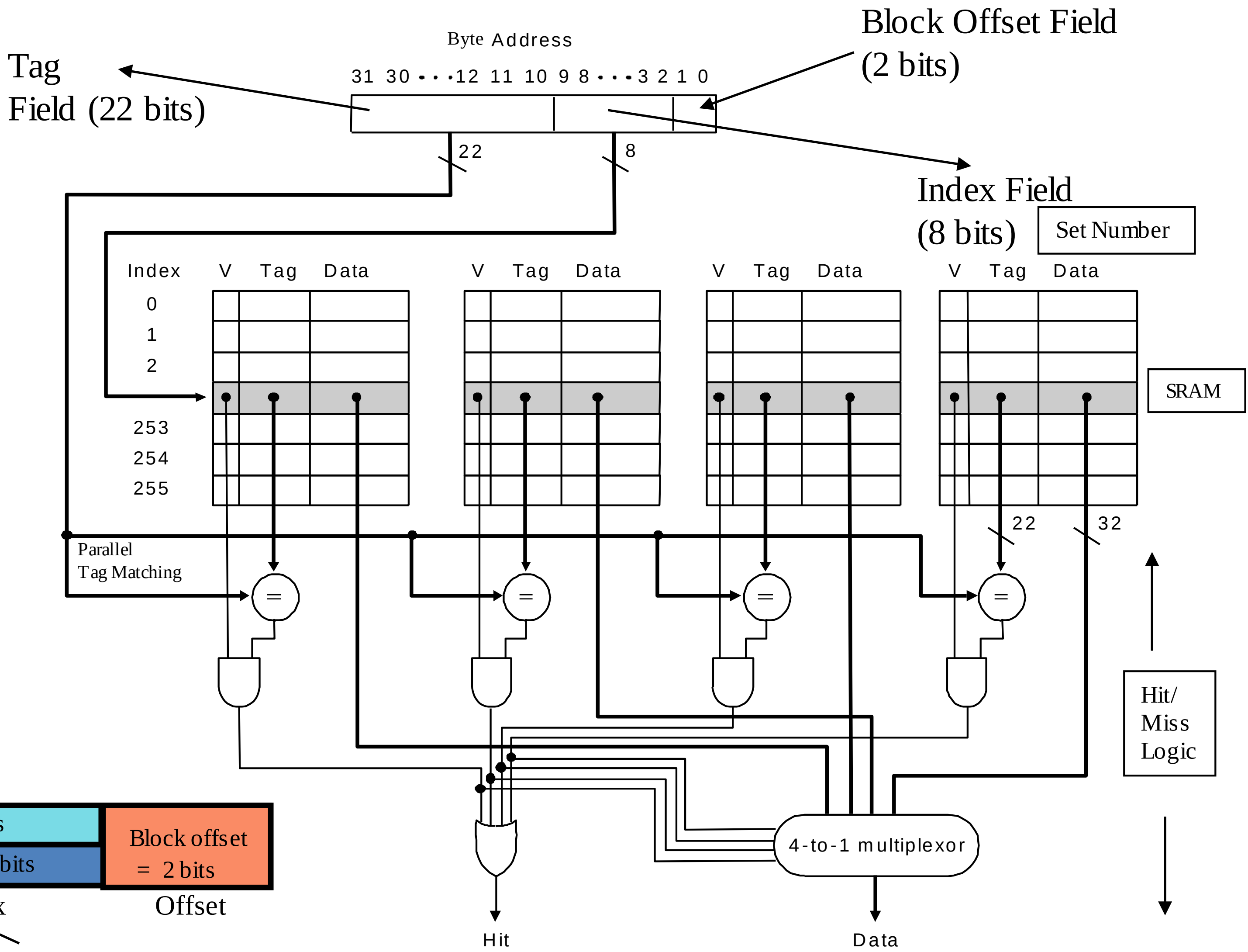
2-Way Set-Associative Cache(4K example)



4K Four-Way Set Associative Cache

4-way set associative
1024 / 4= 2^8= 256 sets

Can cache up to
2^{32} bytes = 4 GB



Block Address = 30 bits

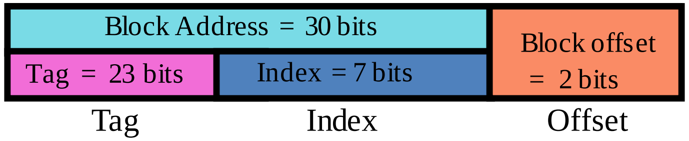
Tag = 22 bits | Index = 8 bits | Block offset = 2 bits

Tag | Index | Offset

Mapping Function: Cache Set Number = index= (Block address) MOD (256)

4K (8, 16-Way ,…. Set Associative Cache

8 way= 4GB=32 bit = 23(tag)+7(index)+2(offset)

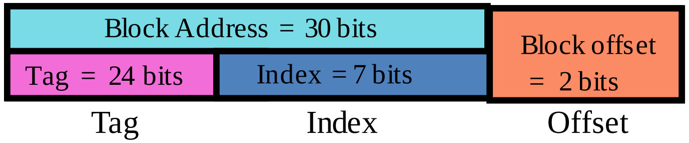


Block Address = 30 bits

Tag = 23 bits | Index = 7 bits | Block offset = 2 bits

Tag | Index | Offset

16 way = 24(tag)+6(index)+2(offset)

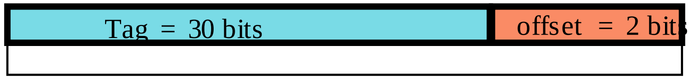


Block Address = 30 bits

Tag = 24 bits | Index = 7 bits | Block offset = 2 bits

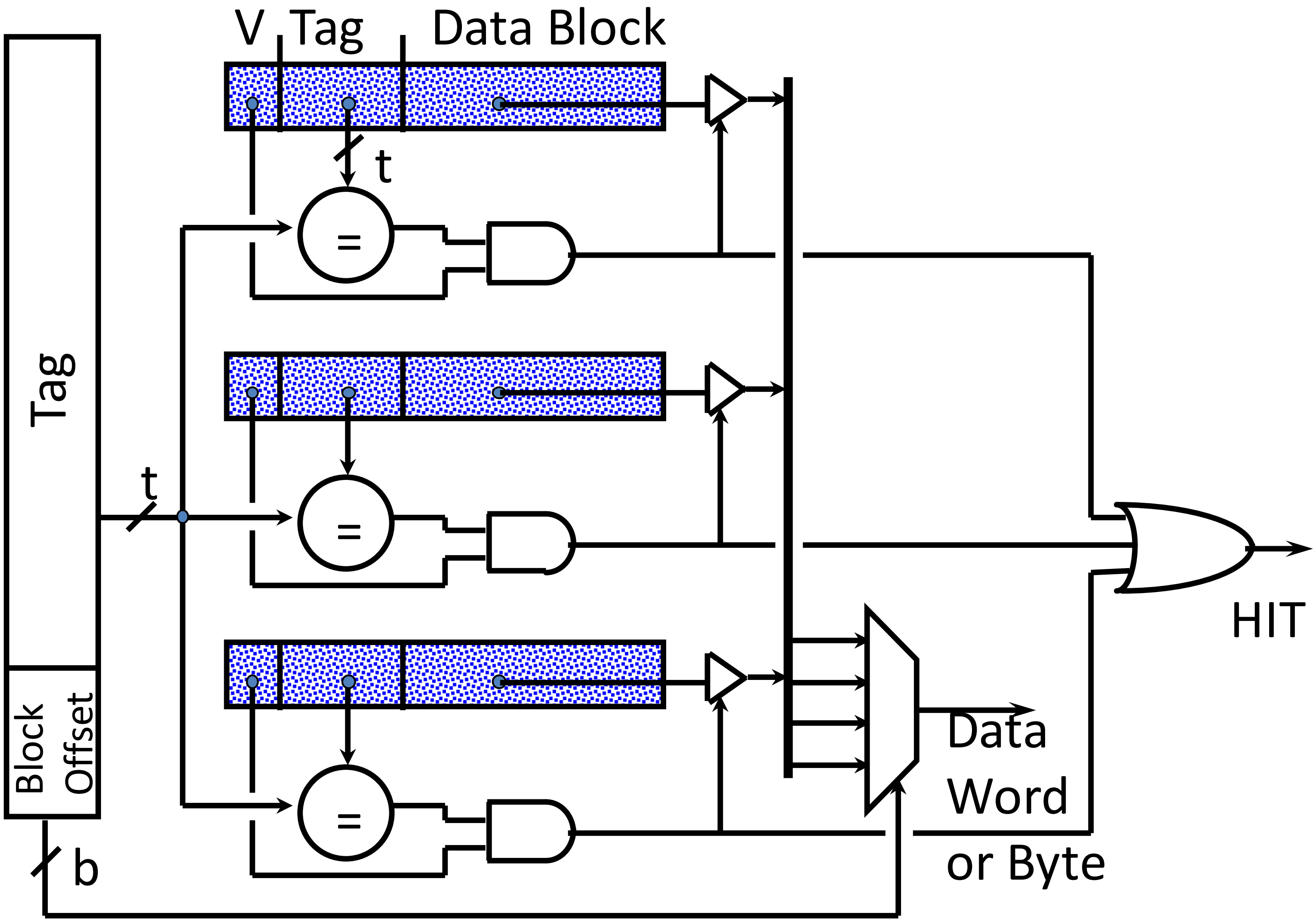
Tag | Index | Offset

Fully Associative = 30(tag)+2(offset)

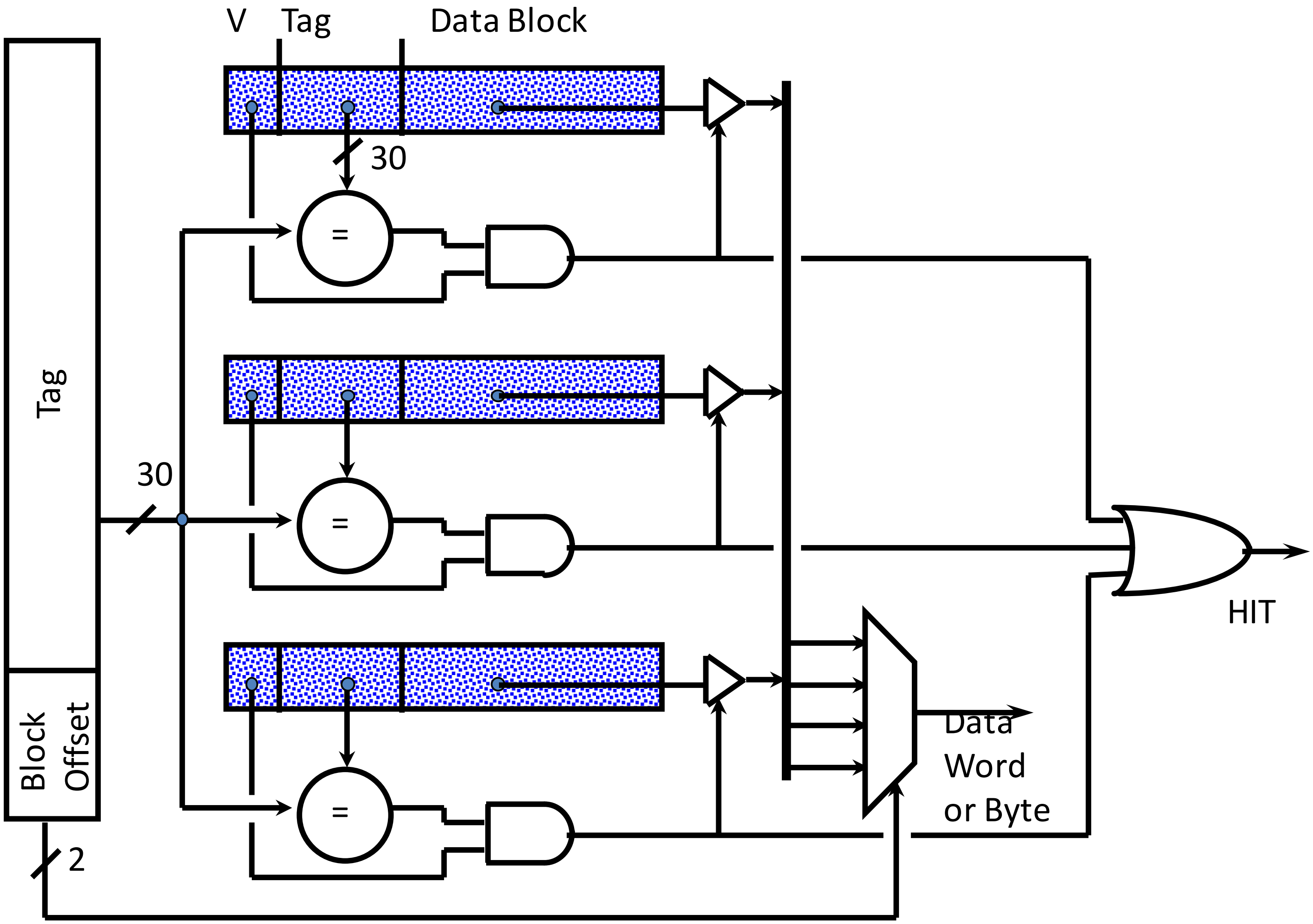


Tag = 30 bits | offset = 2 bits

Fully Associative Cache

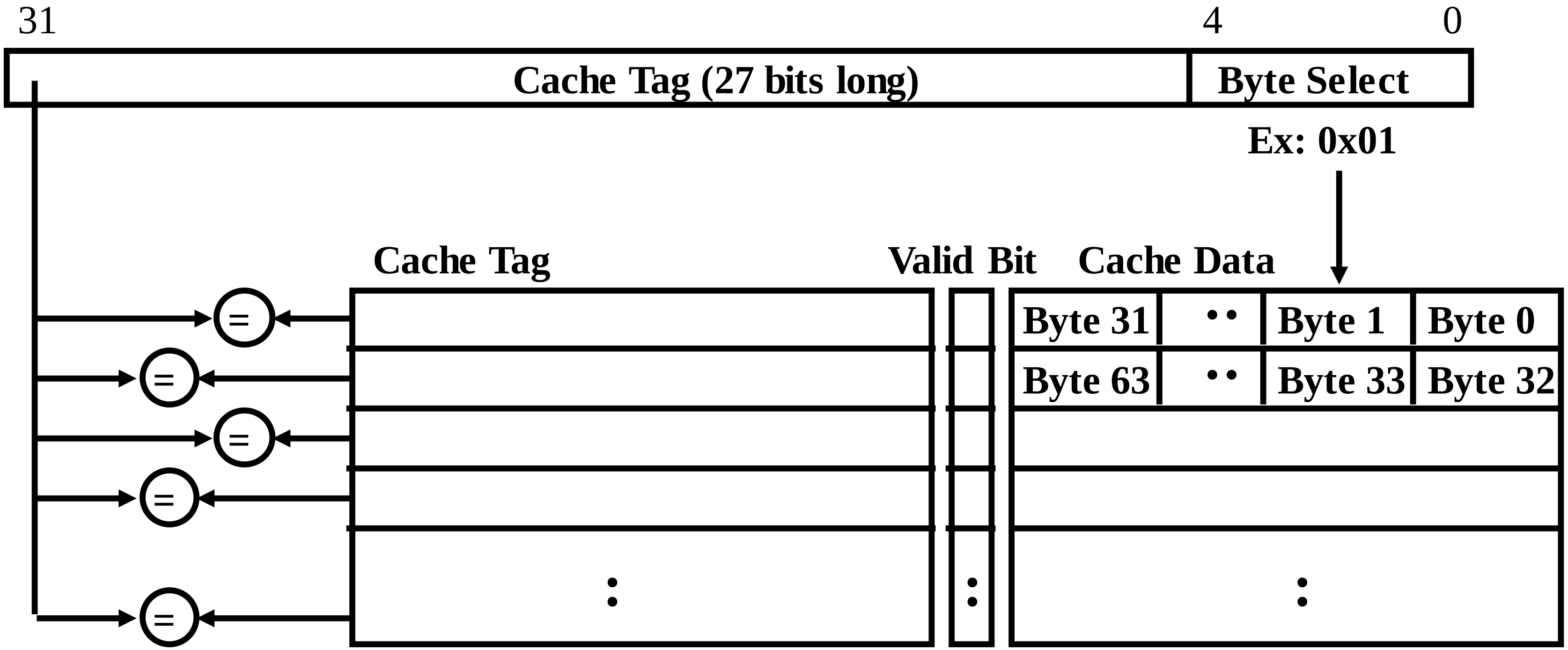


Fully Associative Cache(example)

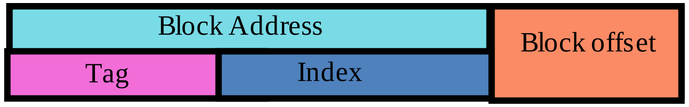


Fully Associative Cache

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



Calculating Number of Cache Bits Needed



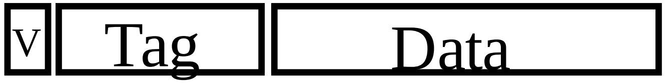
Block Address

Tag

Index

Block offset

Address Fields



V

Tag

Data

Cache Block Frame (or just cache block)

- How many total bits are needed for a direct- mapped cache with 64K Bytes of data and one word blocks, assuming a 32-bit address?

i.e nominal cache Capacity = 64 KB

  - 64 Kbytes = 16 K words = 2^{14} words = 2^{14} blocks
  - Block size = 4 bytes => offset size = $\log_2(4) = 2$ bits,
  - #sets = #blocks = 2^{14} => index size = 14 bits
  - Tag size = address size - index size - offset size = 32 - 14 - 2 = 16 bits
  - Bits/block = data bits + tag bits + valid bit = 32 + 16 + 1 = 49
  - Bits in cache = #blocks x bits/block = 2^{14} x 49 = 98 Kbytes

Number of cache block frames

Actual number of bits in a cache block frame

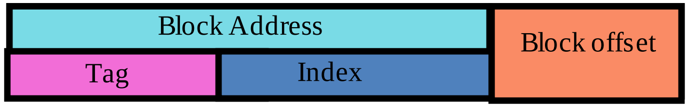
- How many total bits would be needed for a 4-way set associative cache to store the same amount of data?
  - Block size and #blocks does not change.
  - #sets = #blocks/4 = $(2^{14})/4 = 2^{12}$ => index size = 12 bits
  - Tag size = address size - index size - offset = 32 - 12 - 2 = 18 bits
  - Bits/block = data bits + tag bits + valid bit = 32 + 18 + 1 = 51
  - Bits in cache = #blocks x bits/block = 2^{14} x 51 = 102 Kbytes
- Increase associativity => increase bits in cache

More bits in tag

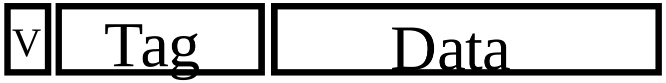
Word = 4 bytes

$1 \text{ k} = 1024 = 2^{10}$

Calculating Cache Bits Needed



Address Fields



Cache Block Frame (or just cache block)

Nominal size

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and 8 word (32 byte) blocks, assuming a 32-bit address (it can cache 2^{32} bytes in memory)?
  - 64 Kbytes = 2^{14} words = $(2^{14})/8 = 2^{11}$ blocks
  - block size = 32 bytes

    => offset size = block offset + byte offset = $\log_2(32) = 5$ bits,
  - #sets = #blocks = 2^{11} => index size = 11 bits
  - tag size = address size - index size - offset size = 32 - 11 - 5 = 16 bits
  - bits/block = data bits + tag bits + valid bit = 8 x 32 + 16 + 1 = 273 bits
  - bits in cache = #blocks x bits/block = 2^{11} x 273 = 68.25 Kbytes
- Increase block size => decrease bits in cache.

Number of cache block frames

Actual number of bits in a cache block frame

Fewer cache block frames thus fewer tags/valid bits

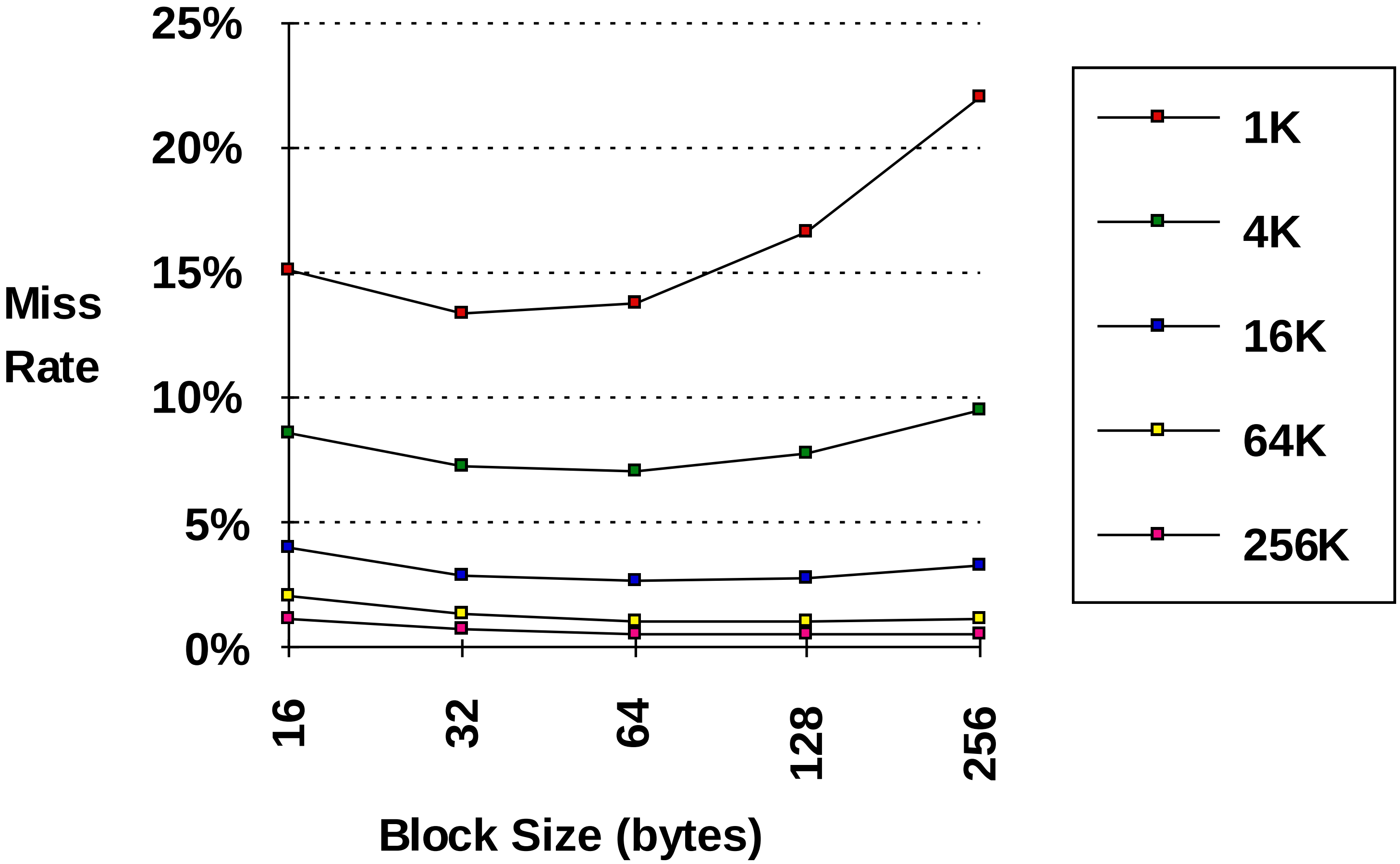
Word = 4 bytes 1 k = 1024 = 2^{10}

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
  - Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant
- Capacity:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity

- Coherence (Invalidation): other process (e.g., I/O) updates memory

Reduce Misses via Larger Block Size



Problem 1

(a) A 64KB, direct mapped cache has 16 byte blocks. If addresses are 32 bits, how many bits are used the tag, index, and offset in this cache?

(b) How would the address be divided if the cache were 4-way set associative instead?

(c) How many bits is the index for a fully associative cache.

Problem 2

An ISA has 44 bit addresses with each addressable item being a byte. Design a four-way set associative cache with each of the four lines in a set containing 64 bytes. Assume that you have 256 sets in the cache.

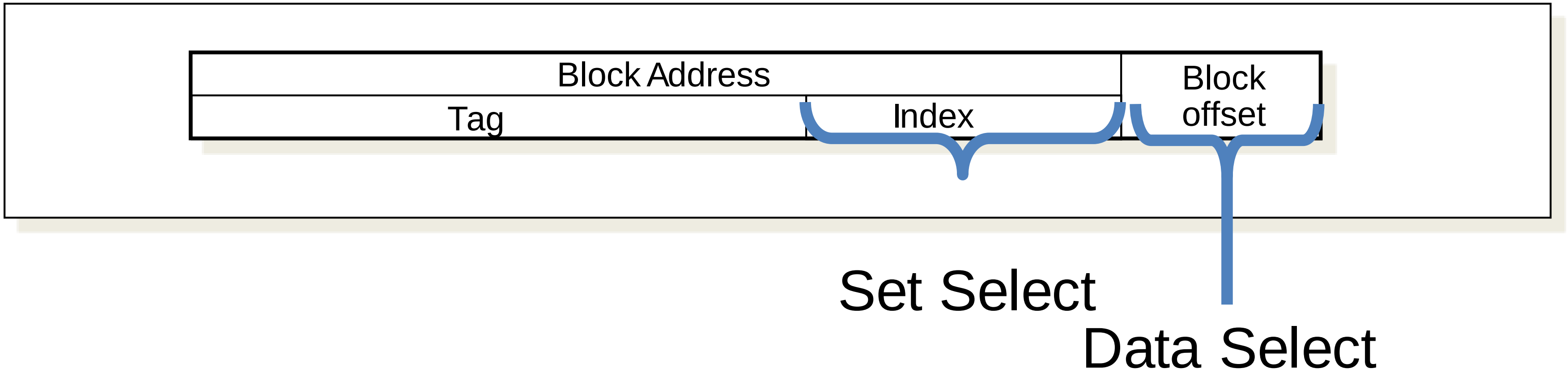
Design Options at Constant Cost

	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	Medium	High
Coherence Miss	Same	Same	Same

Four Questions for Cache Design

- 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)*

How is a block found if it is in the upper level?



- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag

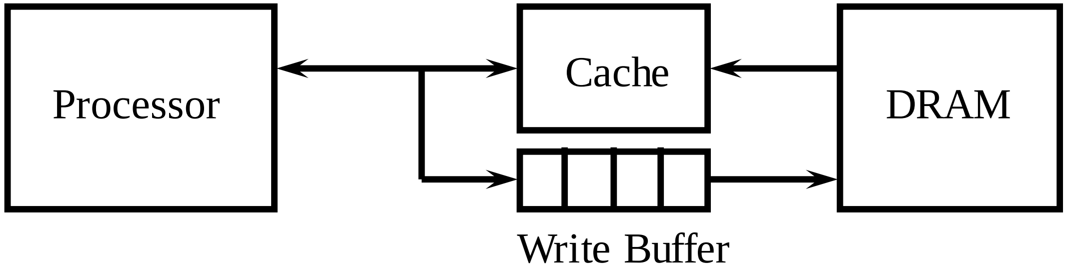
Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

What happens on a write?

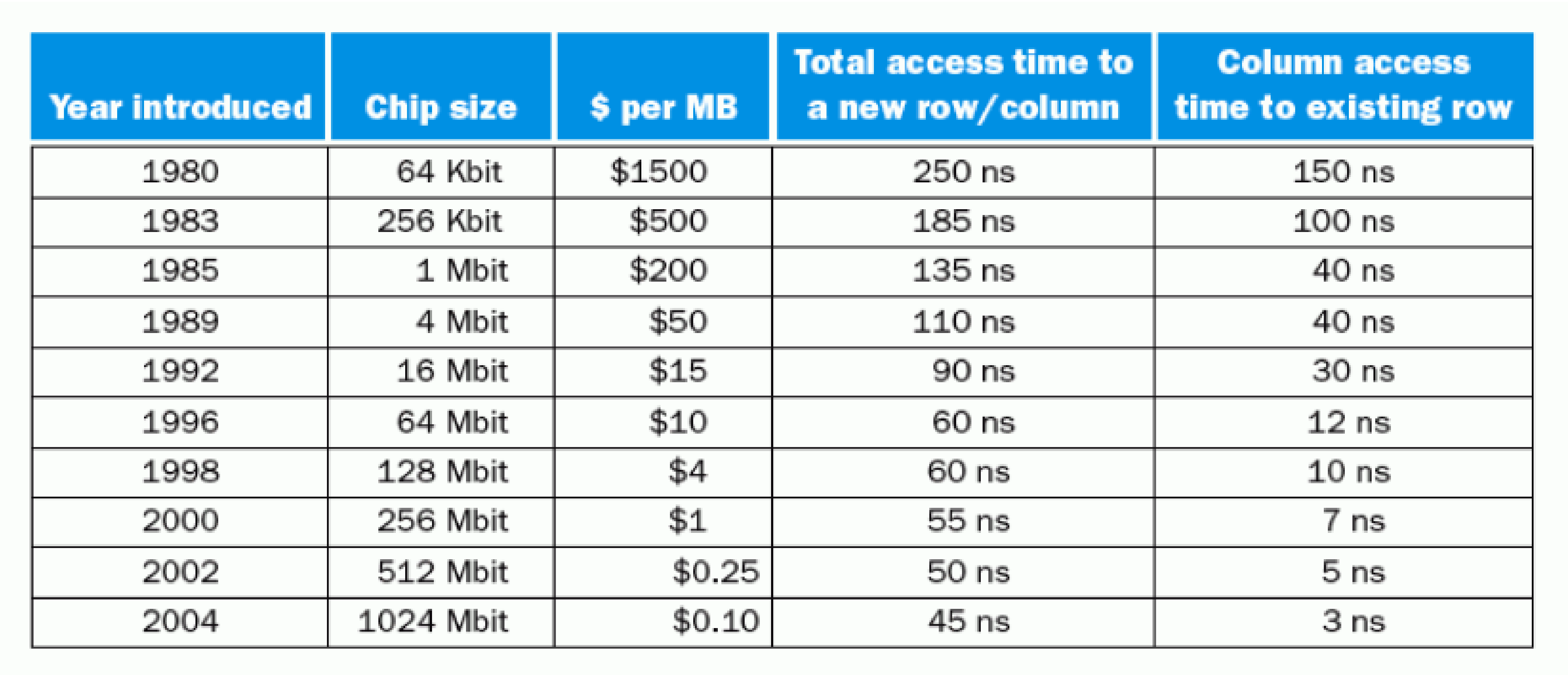
- *Write through*—The information is written to both the block in the cache and to the block in the lower-level memory.
- *Write back*—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?
- WT always combined with write buffers so that don’t wait for lower level memory



Intel Cache Evolution

Problem	Solution	Processor on which feature first appears
External memory slower than the system bus.	Add external cache using faster memory technology.	386
Increased processor speed results in external bus becoming a bottleneck for cache access.	Move external cache on-chip, operating at the same speed as the processor.	486
Internal cache is rather small, due to limited space on chip	Add external L2 cache using faster technology than main memory	486
Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit's data access takes place.	Create separate data and instruction caches.	Pentium
Increased processor speed results in external bus becoming a bottleneck for L2 cache access.	Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.	Pentium Pro
	Move L2 cache on to the processor chip.	Pentium II
Some applications deal with massive databases and must have rapid access to large amounts of data. The on-chip caches are too small.	Add external L3 cache.	Pentium III
	Move L3 cache on-chip.	Pentium 4

Main memory Evolution



Year introduced	Chip size	$ per MB	Total access time to a new row/column	Column access time to existing row
1980	64 Kbit	$1500	250 ns	150 ns
1983	256 Kbit	$500	185 ns	100 ns
1985	1 Mbit	$200	135 ns	40 ns
1989	4 Mbit	$50	110 ns	40 ns
1992	16 Mbit	$15	90 ns	30 ns
1996	64 Mbit	$10	60 ns	12 ns
1998	128 Mbit	$4	60 ns	10 ns
2000	256 Mbit	$1	55 ns	7 ns
2002	512 Mbit	$0.25	50 ns	5 ns
2004	1024 Mbit	$0.10	45 ns	3 ns

Comparison of Cache Sizes

Processor	Type	Year of Introduction	L1 cache[a]	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	
CRAY MTA[b]	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—
Itanium 2	PC/server	2002	32 KB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 KB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 KB/64 KB	1MB	—

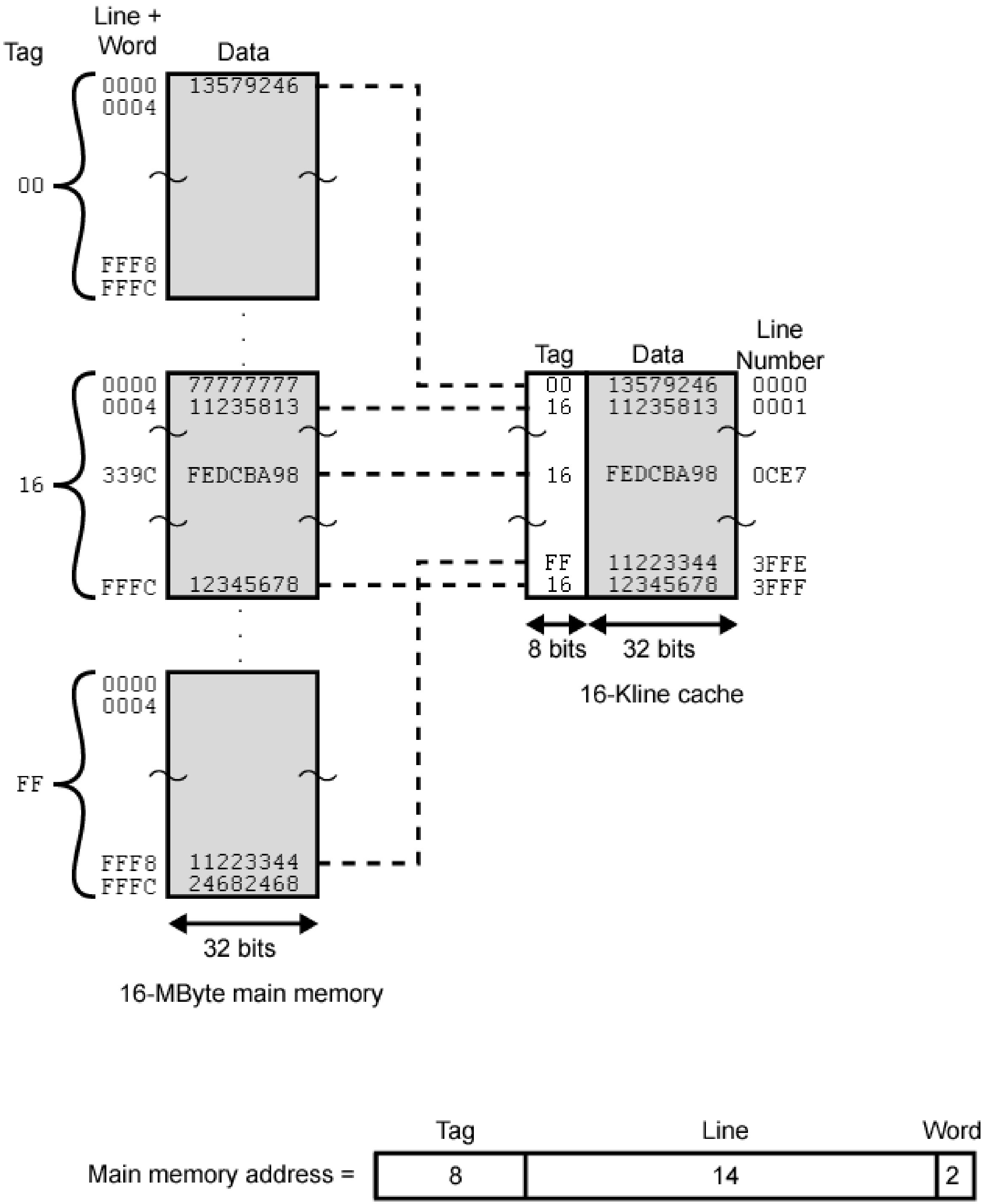
Memory Hierarchy in Power 4/5

Cache	Capacity	Associativity	Line size	Write policy	Repl. alg	Comments
L1 I-cache	64 KB	Direct/2-way	128 B		LRU	Sector cache (4 sectors)
L1 D-cache	32 KB	2-way/4-way	128 B	Write-through	LRU	
L2 Unified	1.5 MB/2 MB	8-way/10-way	128 B	Write-back	Pseudo-LRU	
L3	32 MB/36MB	8-way/12-way	512 B	Write-back	?	Sector cache (4 sectors)

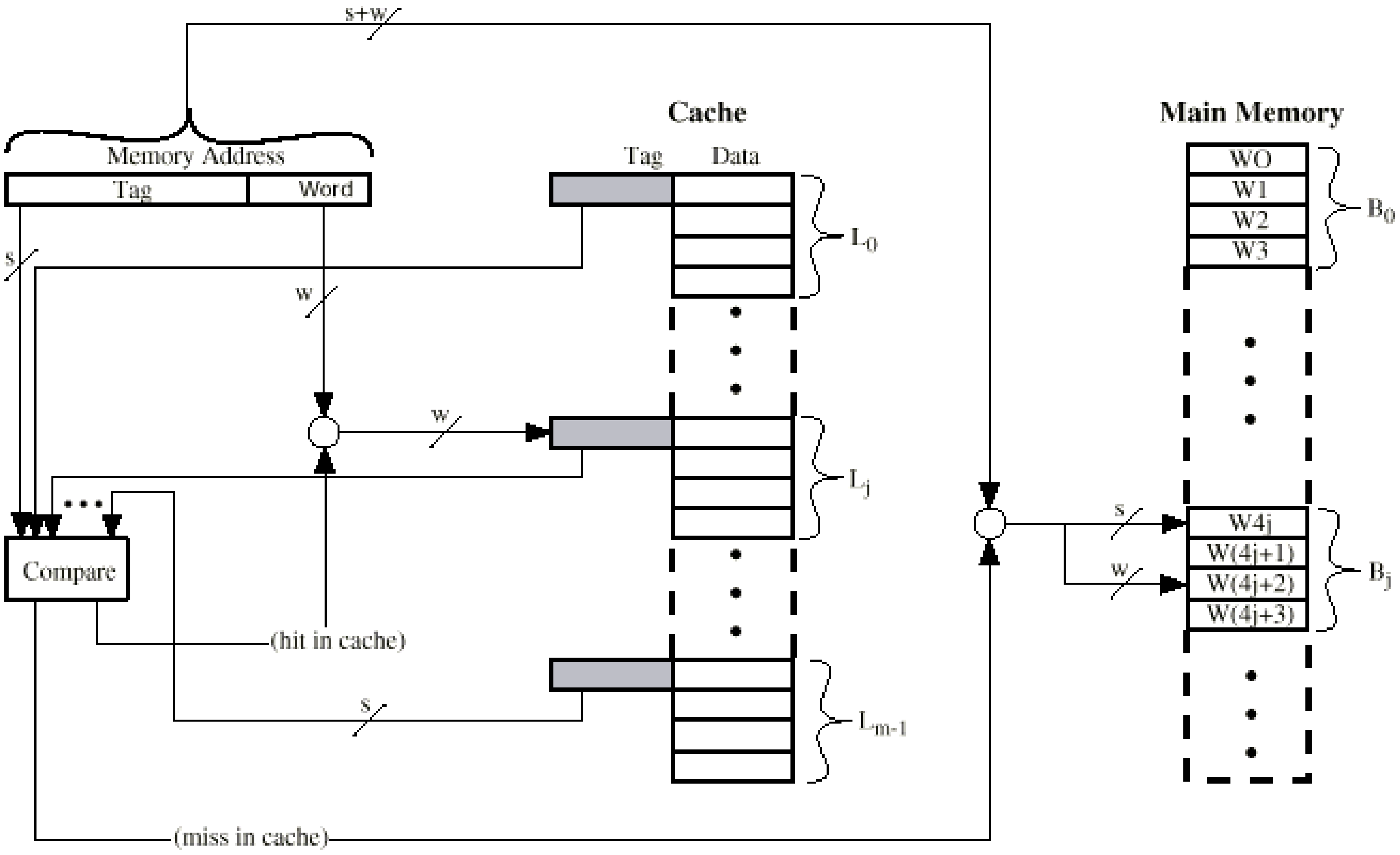
Latency	P4 (1.7 GHz)	P5 (1.9 GHz)
L1 (I and D)	1	1
L2	12	13
L3	123	87
Main Memory	351	220

More examples: Direct Mapping Example (16M, 16K cache)

8(tag)	**14(index)**	**2**



Fully Associative Cache Organization


s+w
Cache
Main Memory
Memory Address
Tag
Word
Tag
Data
WO
W1
W2
W3
B_0
L_0
s
w
w
L_j
Compare
s
w
W4j
W(4j+1)
W(4j+2)
W(4j+3)
B_j
(hit in cache)
s
L_{m-1}
(miss in cache)