CS150 - EE141/241A
Fall 2014
Digital Design and Integrated Circuits

Instructors:
John Wawrzynek and Vladimir Stojanovic

Lecture 17

Outline

- *Memory Block Introduction*
- *Cache Introduction*
- *Cache Design and Optimization*

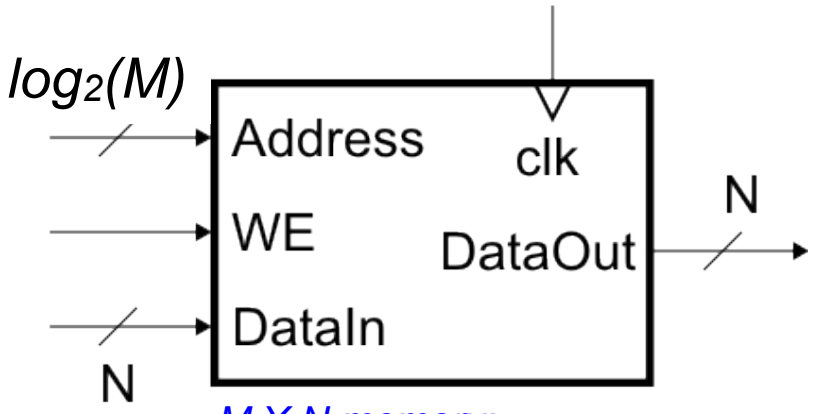
Memory Block Introduction

Memory-Block Basics

- Uses:

  Whenever a large collection of state elements is required.

  - data & program storage
  - general purpose registers
  - data buffering
  - table lookups
  - CL implementation



M X N memory:

Depth = M, Width = N.

M words of memory, each word N bits wide.

- Basic Types:
  - RAM - random access memory
  - ROM - read only memory
  - EPROM, FLASH - electrically programmable read only memory

Memory Components Types:

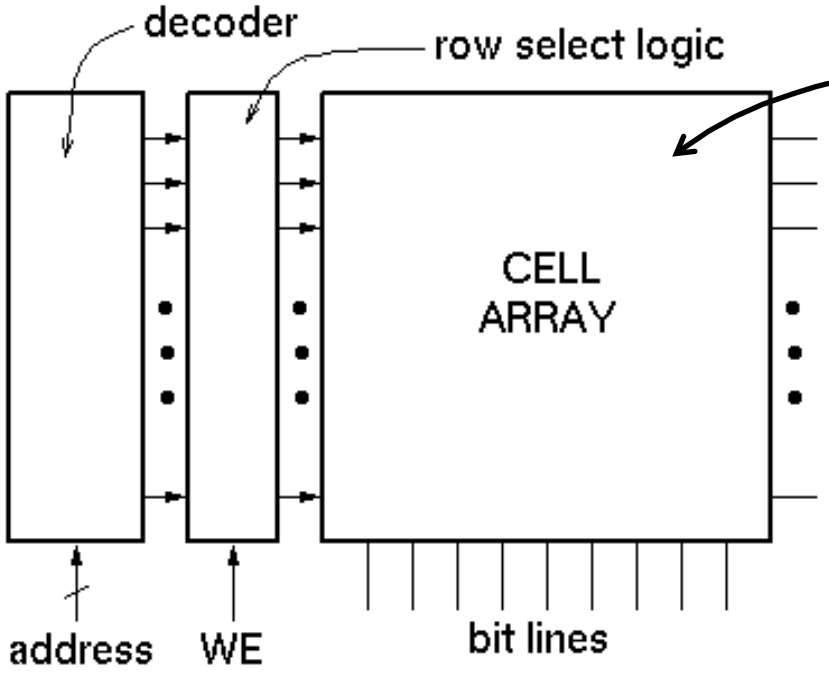
- Volatile:
  - Random Access Memory (RAM):
    - DRAM "dynamic"
    - SRAM "static"

Focus Today

- Non-volatile:
  - Read Only Memory (ROM):
    - Mask ROM "mask programmable"
    - EPROM "electrically programmable"
    - EEPROM "erasable electrically programmable"
    - FLASH memory - similar to EEPROM with programmer integrated on chip

All these types are available as stand alone chips or as blocks in other chips.

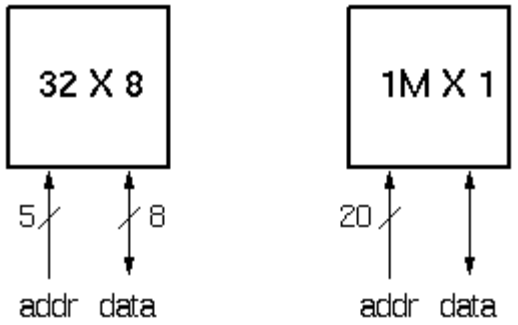
Standard Internal Memory Organization



2-D arrary of bit cells. Each cell stores one bit of data.

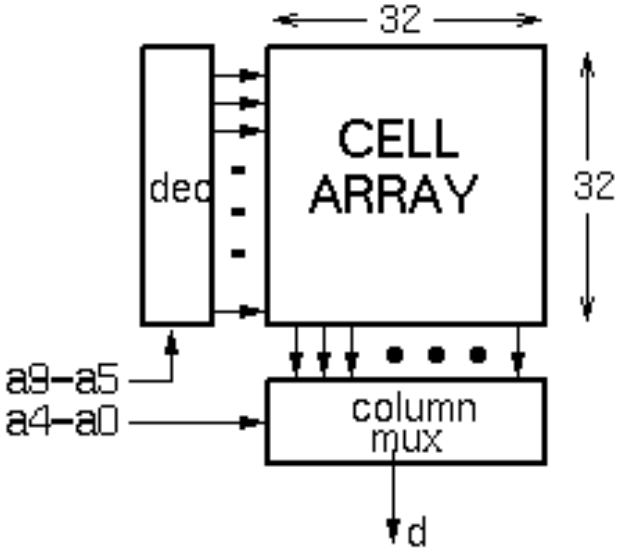
Special circuit tricks are used for the cell array to improve storage density.

- RAM/ROM naming convention:
  - examples: 32 X 8, "32 by 8" => 32 8-bit words
  - 1M X 1, "1 meg by 1" => 1M 1-bit words

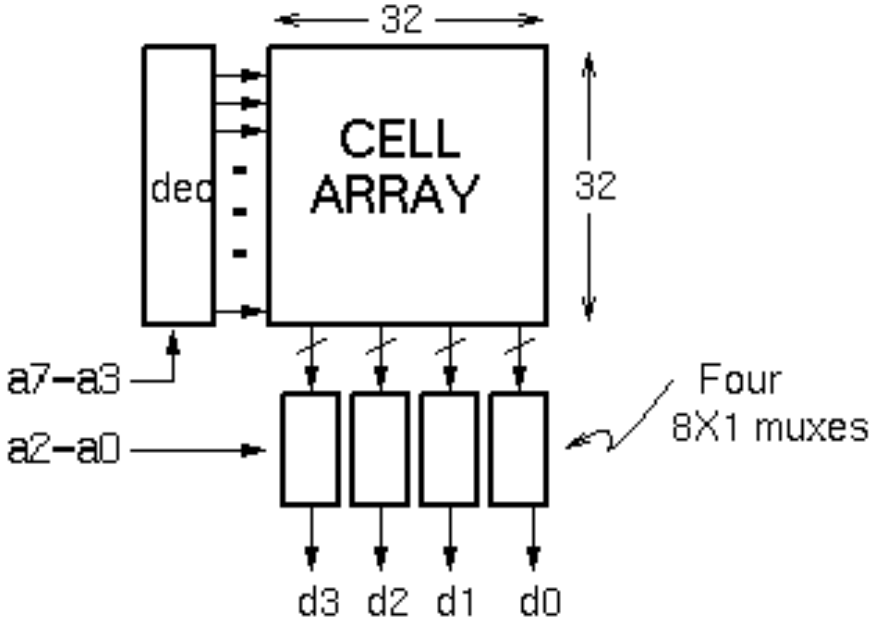


Column MUX in ROMs and RAMs:

- Permits input/output data widths different from row width.
- Controls physical aspect ratio
  - Important for physical layout and to control delay on wires.



1K X 1 ROM



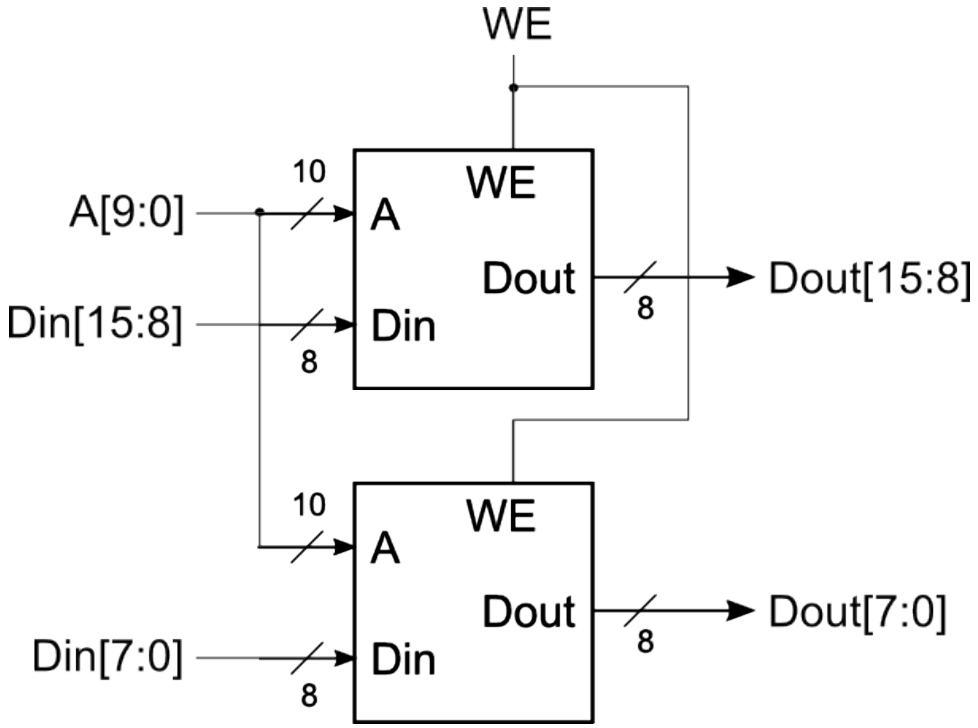
256 X 4 ROM

Technique illustrated for read operation. Similar approach for write.

Cascading Memory-Blocks

How to make larger memory blocks out of smaller ones.

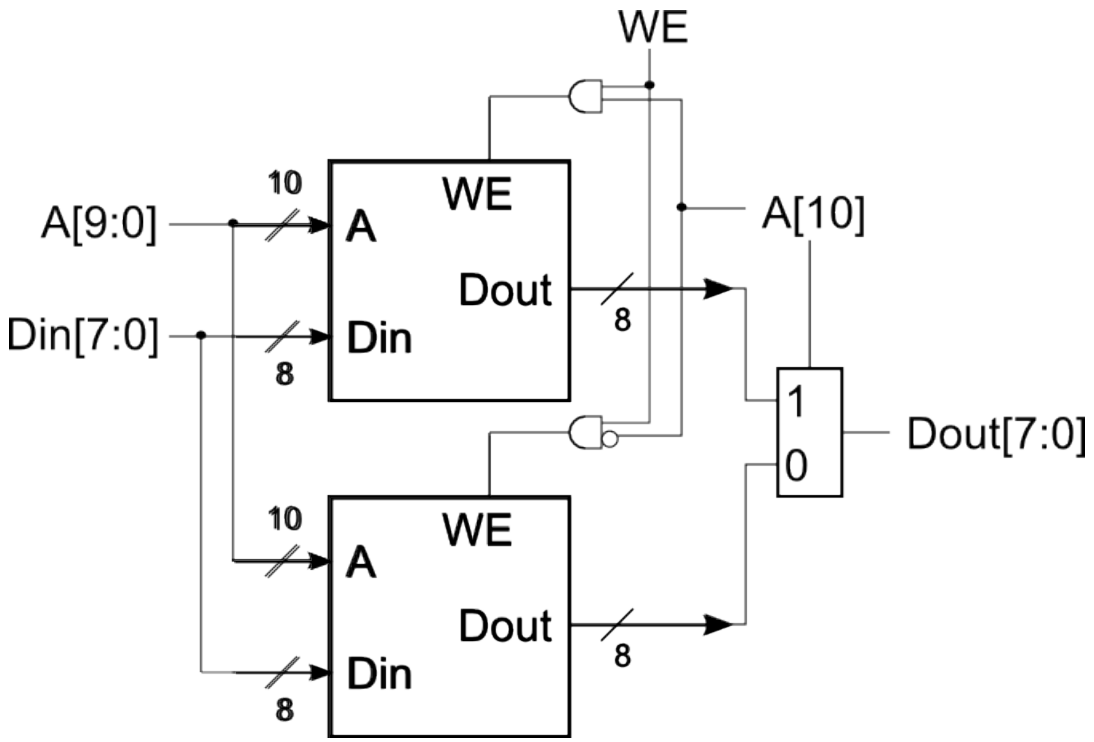
Increasing the width. Example: given 1Kx8, want 1Kx16



Cascading Memory-Blocks

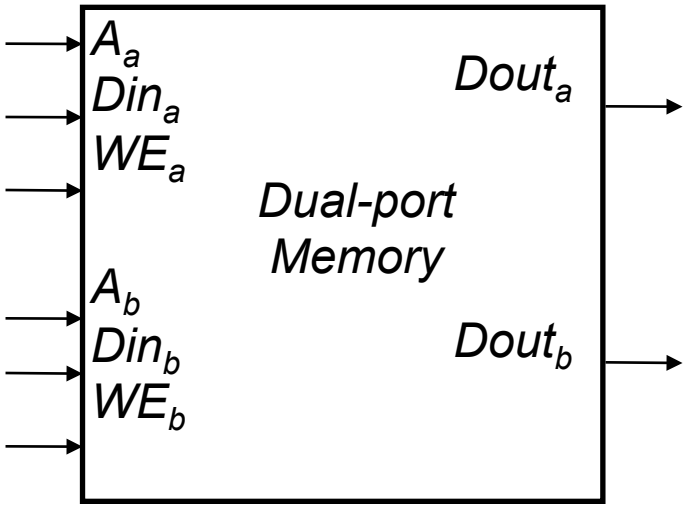
How to make larger memory blocks out of smaller ones.

Increasing the depth. Example: given 1Kx8, want 2Kx8



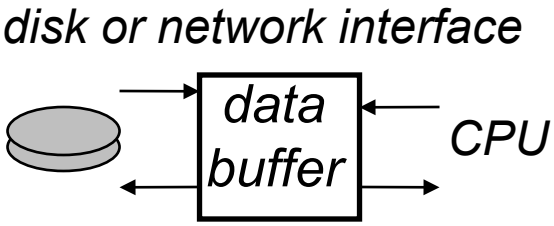
Multi-ported Memory

- Motivation:
  - Consider CPU core register file:
    - 1 read or write per cycle limits processor performance.
    - Complicates pipelining. Difficult for different instructions to simultaneously read or write regfile.
    - Common arrangement in pipelined CPUs is 2 read ports and 1 write port.



- *dual-porting allows both sides to simultaneously access memory at full bandwidth.*

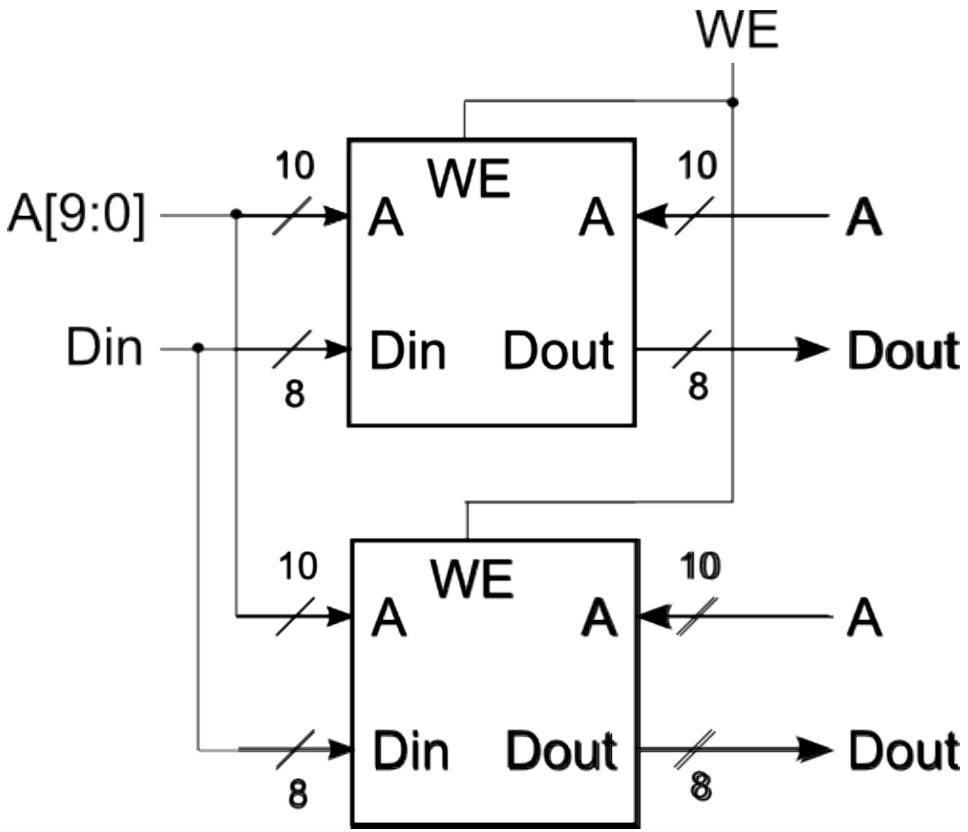
– *I/O data buffering:*



Adding Ports to Primitive Memory Blocks

Adding a read port to a simple dual port (SDP) memory.

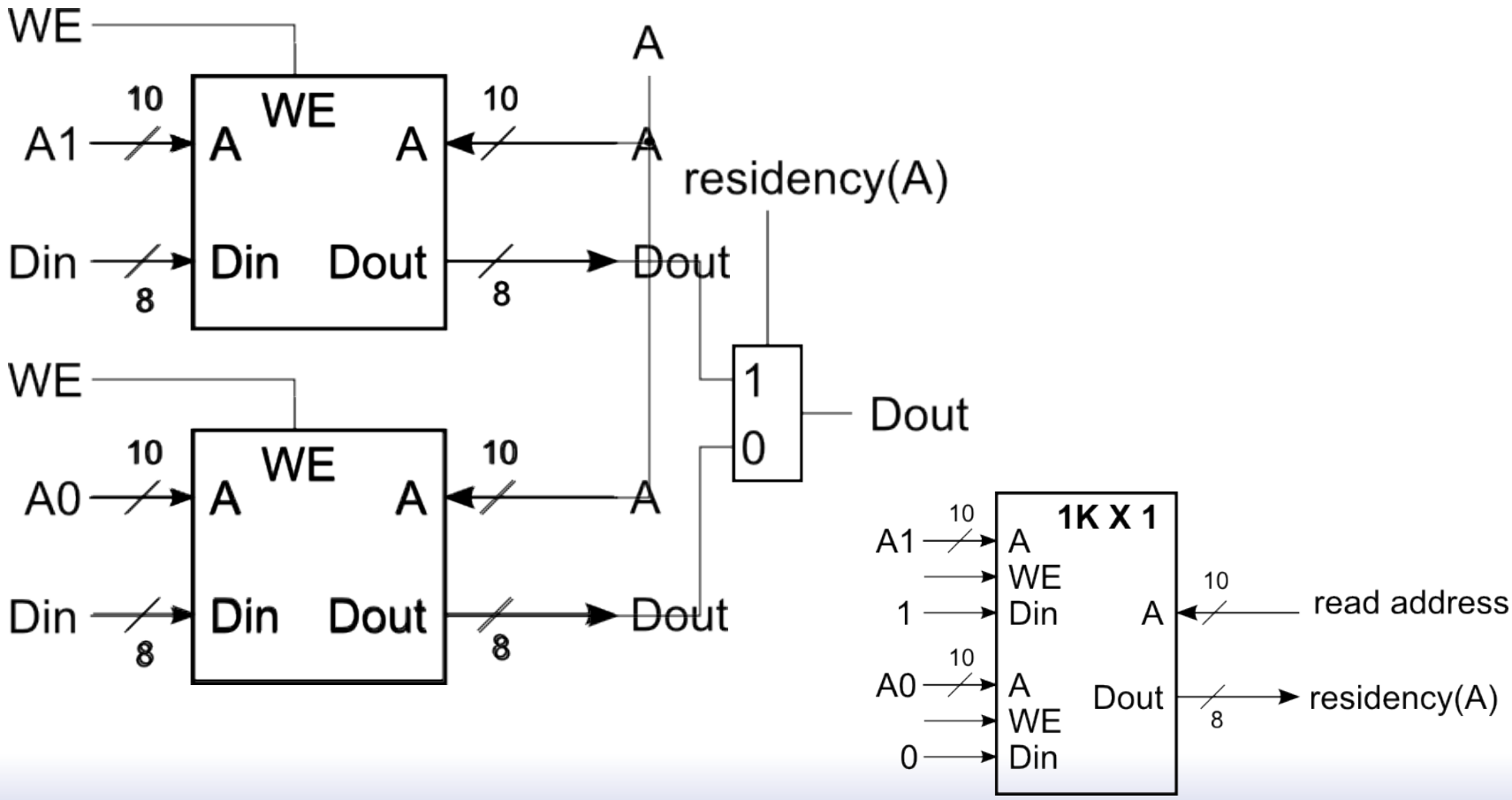
Example: given 1Kx8 SDP, want 1 write & 2 read ports.



Adding Ports to Primitive Memory Blocks

How to add a write port to a simple dual port memory.

Example: given 1Kx8 SDP, want 1 read & 2 write ports.



Virtex-5 LX110T memory blocks.

Distributed RAM using LUTs among the CLBs.

Block RAMs in four columns.

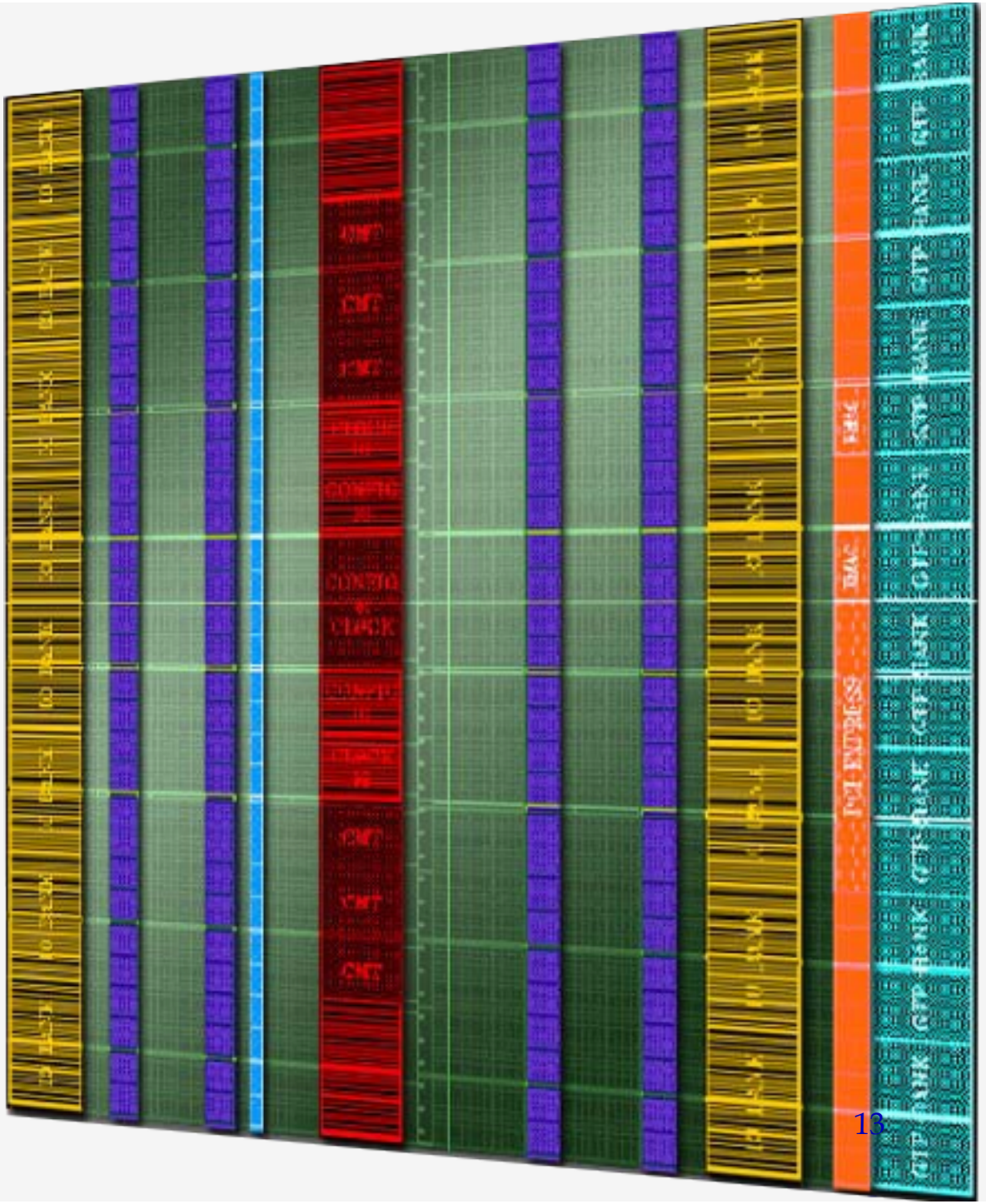
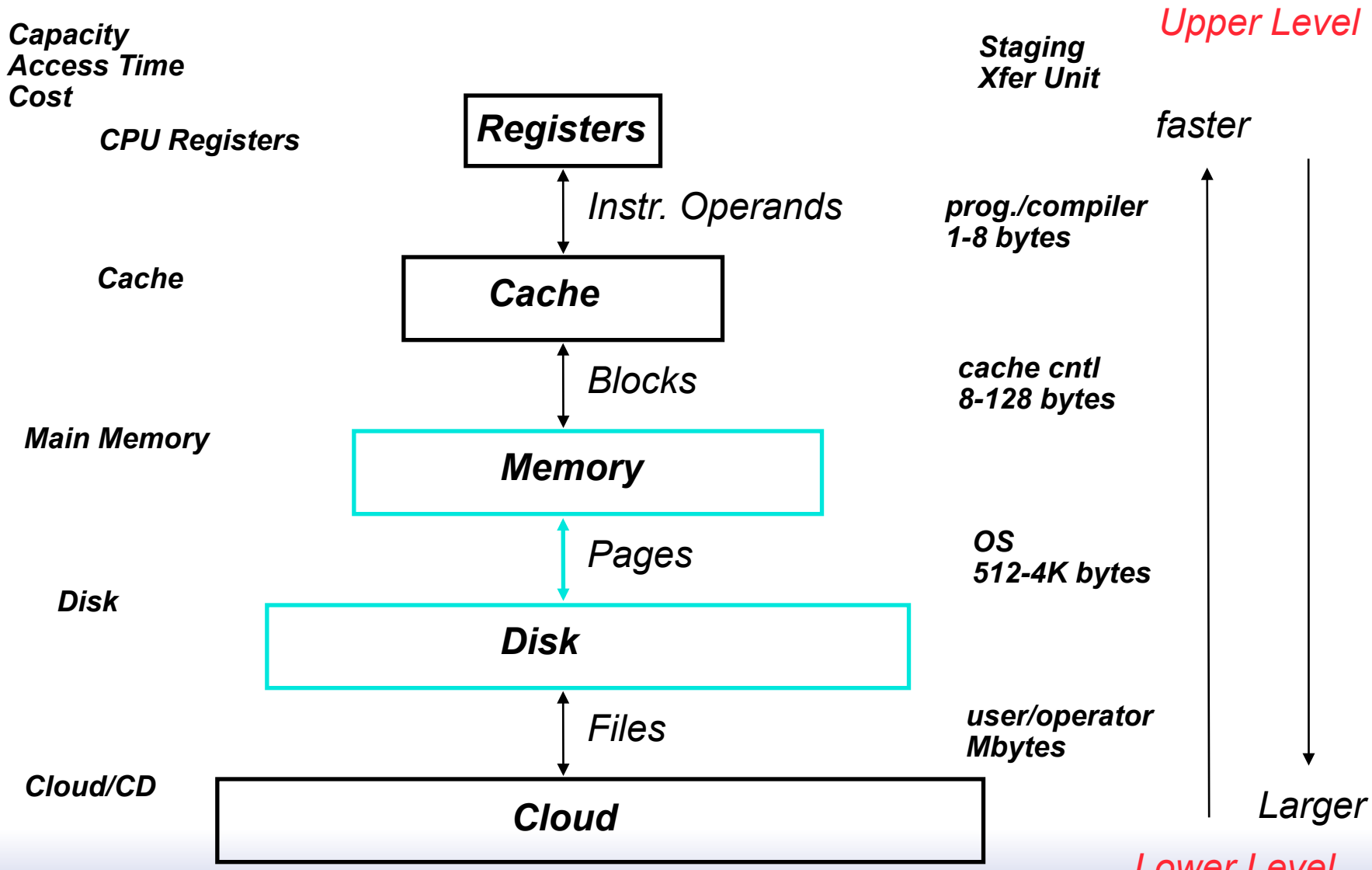


Table 1: **Virtex-5 FPGA Family Members**

Device	Configurable Logic Blocks (CLBs)			DSP48E Slices[2]	Block RAM Blocks			CMTs[4]	PowerPC Processor Blocks	Endpoint Blocks for PCI Express	Ethernet MACs[5]	Max RocketIO Transceivers[6]		Total I/O Banks[8]	Max User I/O[7]
	Array (Row x Col)	Virtex-5 Slices[1]	Max Distributed RAM (Kb)		18 Kb[3]	36 Kb	Max (Kb)					GTP	GTX		
XC5VLX30	80 x 30	4,800	320	32	64	32	1,152	2	N/A	N/A	N/A	N/A	N/A	13	400
XC5VLX50	120 x 30	7,200	480	48	96	48	1,728	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX85	120 x 54	12,960	840	48	192	96	3,456	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX110	160 x 54	17,280	1,120	64	256	128	4,608	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX155	160 x 76	24,320	1,640	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX220	160 x 108	34,560	2,280	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX330	240 x 108	51,840	3,420	192	576	288	10,368	6	N/A	N/A	N/A	N/A	N/A	33	1,200
XC5VLX20T	60 x 26	3,120	210	24	52	26	936	1	N/A	1	2	4	N/A	7	172
XC5VLX30T	80 x 30	4,800	320	32	72	36	1,296	2	N/A	1	4	8	N/A	12	360
XC5VLX50T	120 x 30	7,200	480	48	120	60	2,160	6	N/A	1	4	12	N/A	15	480
XC5VLX85T	120 x 54	12,960	840	48	216	108	3,888	6	N/A	1	4	12	N/A	15	480
XC5VLX110T	160 x 54	17,280	1,120	64	296	148	5,328	6	N/A	1	4	16	N/A	20	680
XC5VLX155T	160 x 76	24,320	1,640	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX220T	160 x 108	34,560	2,280	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX330T	240 x 108	51,840	3,420	192	648	324	11,664	6	N/A	1	4	24	N/A	27	960
XC5VSX35T	80 x 34	5,440	520	192	168	84	3,024	2	N/A	1	4	8	N/A	12	360
XC5VSX50T	120 x 34	8,160	780	288	264	132	4,752	6	N/A	1	4	12	N/A	15	480
XC5VSX95T	160 x 46	14,720	1,520	640	488	244	8,784	6	N/A	1	4	16	N/A	19	640
XC5VSX240T	240 x 78	37,440	4,200	1,056	1,032	516	18,576	6	N/A	1	4	24	N/A	27	960
XC5VTX150T	200 x 58	23,200	1,500	80	456	228	8,208	6	N/A	1	4	N/A	40	20	680
XC5VTX240T	240 x 78	37,440	2,400	96	648	324	11,664	6	N/A	1	4	N/A	48	20	680
XC5VFX30T	80 x 38	5,120	380	64	136	68	2,448	2	1	1	4	N/A	8	12	360
XC5VFX70T	160 x 38	11,200	820	128	296	148	5,328	6	1	3	4	N/A	16	19	640
XC5VFX100T	160 x 56	16,000	1,240	256	456	228	8,208	6	2	3	4	N/A	16	20	680
XC5VFX130T	200 x 56	20,480	1,580	320	596	298	10,728	6	2	3	6	N/A	20	24	840
XC5VFX200T	240 x 68	30,720	2,280	384	912	456	16,416	6	2	4	8	N/A	24	27	960

Cache Introduction

Recall: Levels of the Memory Hierarchy



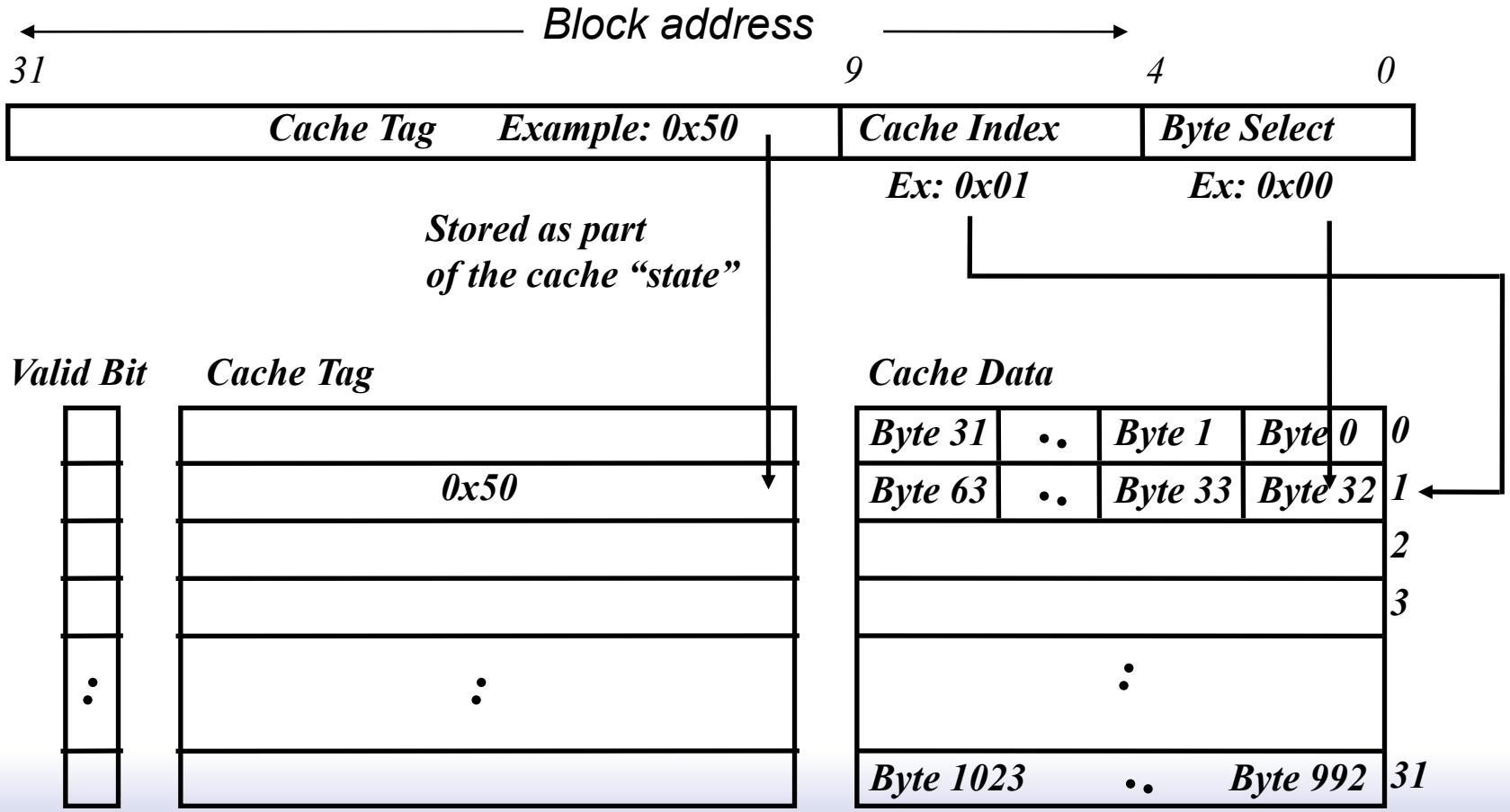
Review from 61C

- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.
- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.
- DRAM is slow but cheap and dense:
  - Good choice for presenting the user with a BIG memory system
- SRAM is fast but expensive and not very dense:
  - Good choice for providing the user FAST access time.

Example: 1 KB Direct Mapped Cache with 32 B Blocks

For a 2^N byte cache:

- **The uppermost (32 - N) bits are always the Cache Tag**
- **The lowest M bits are the Byte Select (Block Size = 2^M)**



Block Size Tradeoff

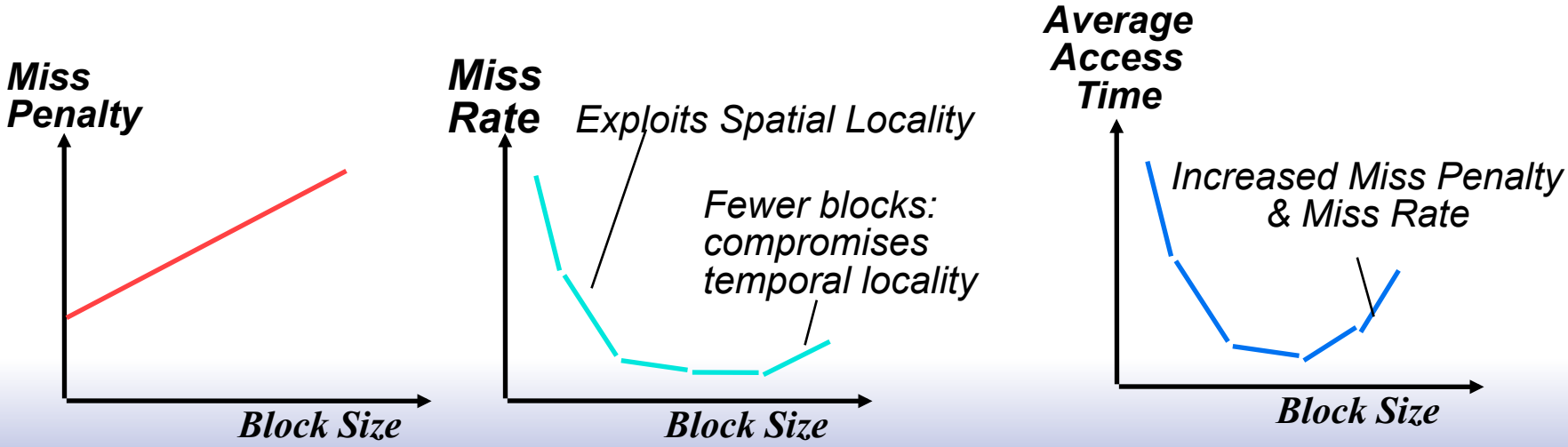
In general, larger block size take advantage of spatial locality

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

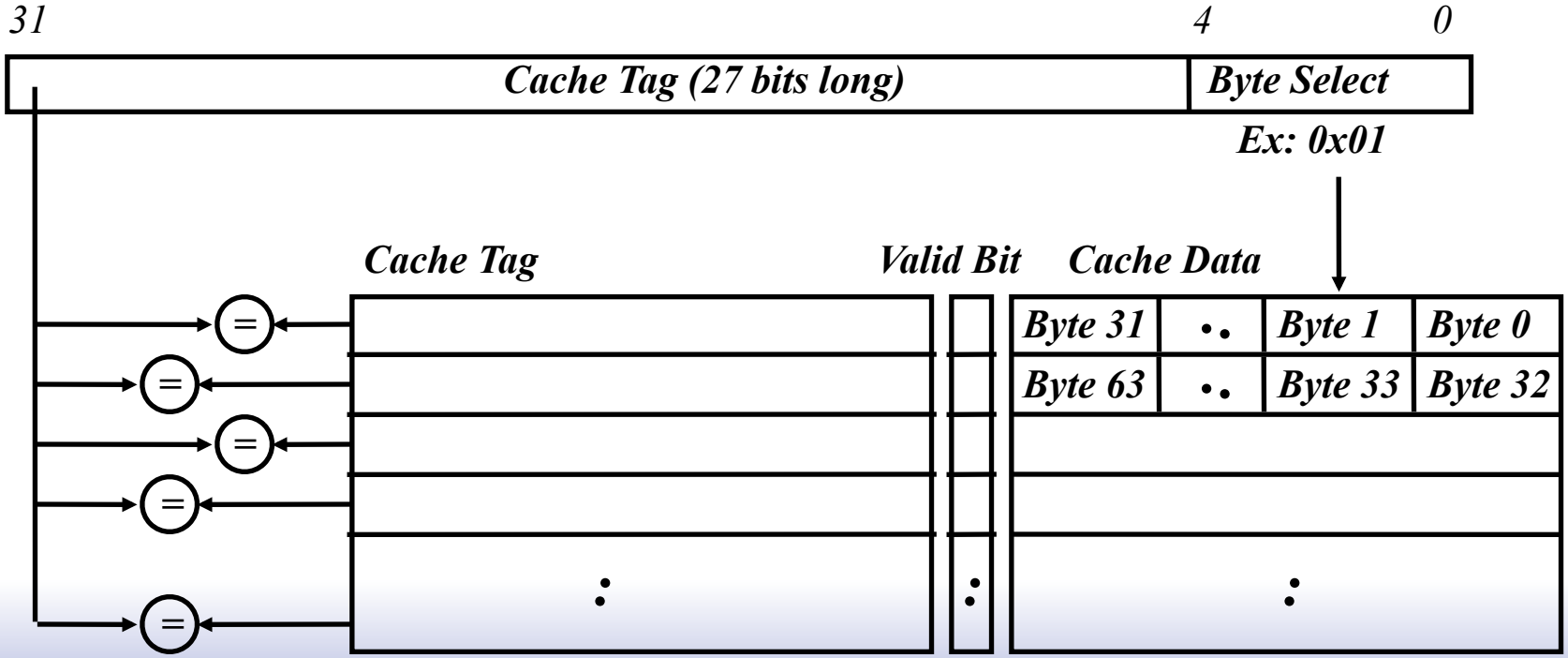


Extreme Example: Fully Associative

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



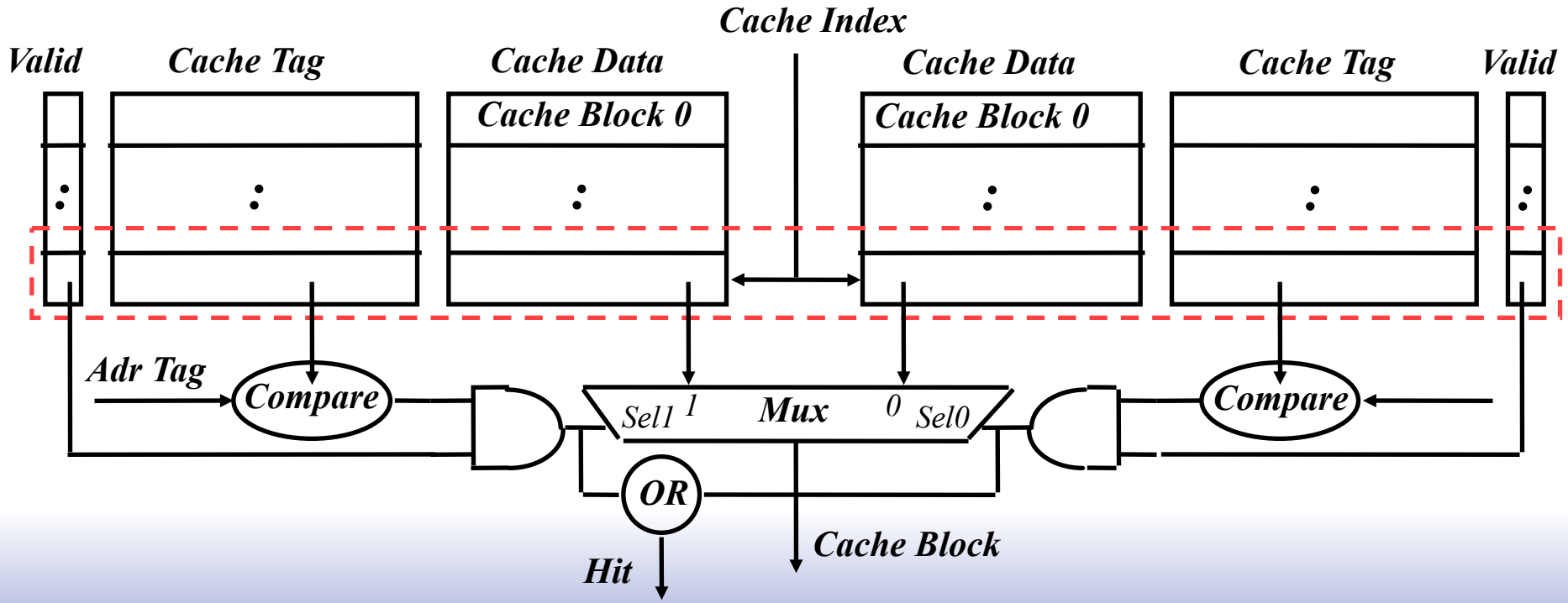
Set Associative Cache

N-way set associative: N entries for each Cache Index

- N direct mapped caches operates in parallel

Example: Two-way set associative cache

- Cache Index selects a "set" from the cache
- The two tags in the set are compared to the input in parallel
- Data is selected based on the tag result



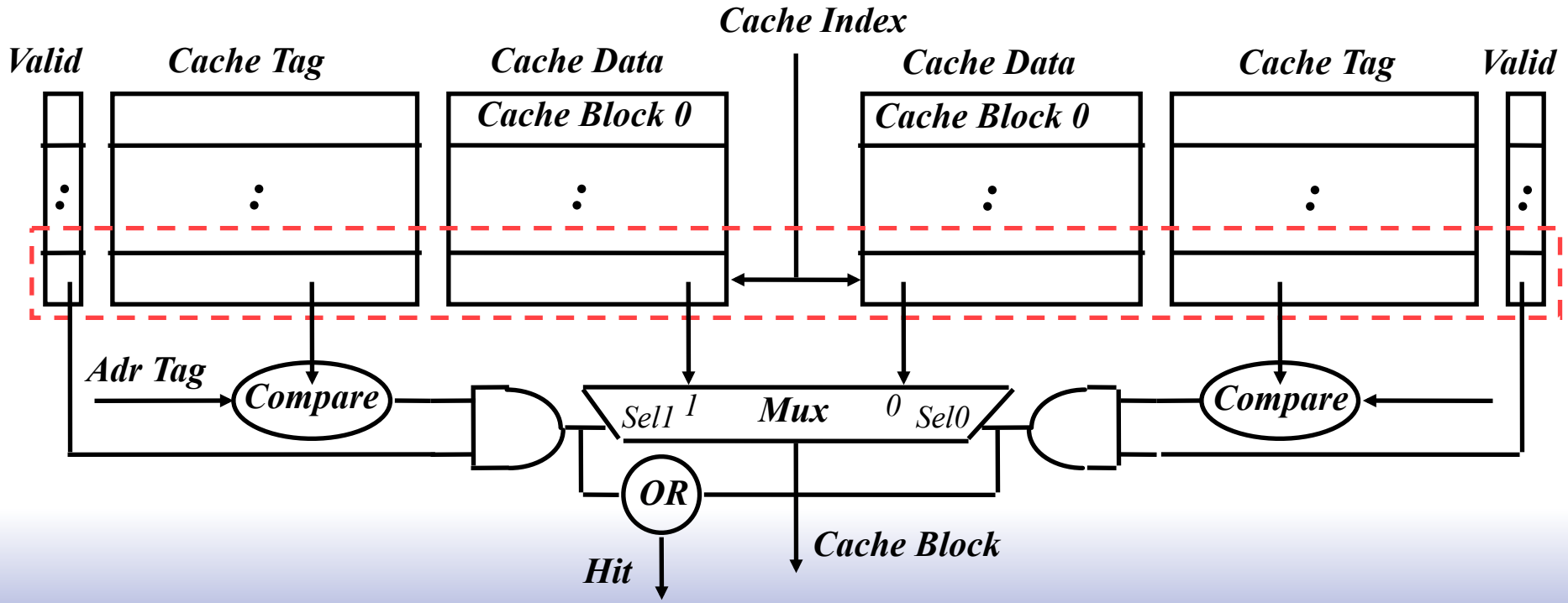
Disadvantage of Set Associative Cache

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.

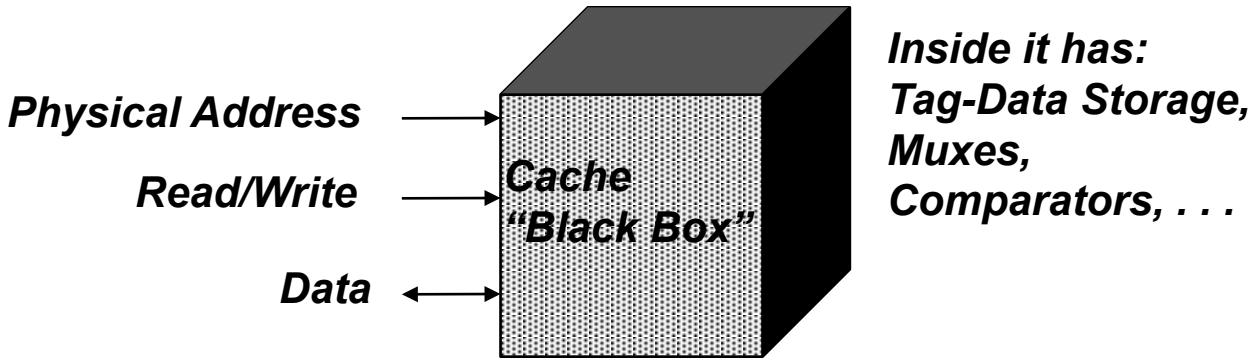


Cache Design and Optimization

How do you Design a Cache?

Set of Operations that must be supported

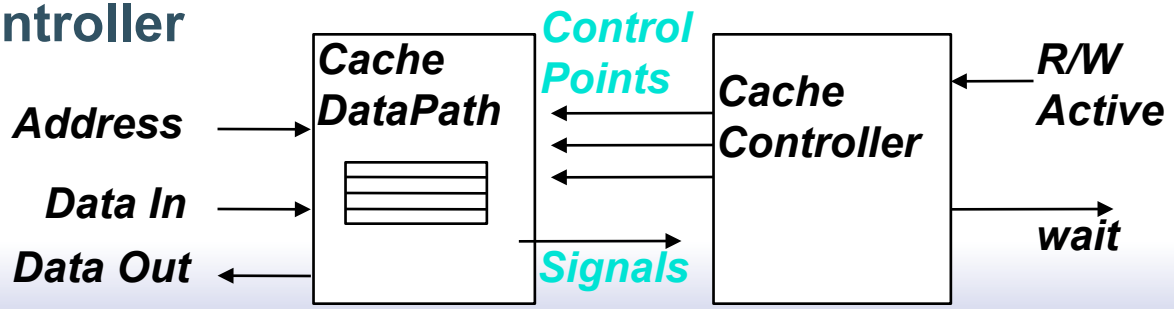
- read: data <= Mem[Physical Address]
- write: Mem[Physical Address] <= Data



Determine the internal register transfers

Design the Datapath

Design the Cache Controller



Performance = Intr. Count x Clock Freq x (ideal CPI + stalls)

Average Memory Access time =
Hit Time + *Miss Rate* x *Miss Penalty*

Improving Cache Performance: 3 general options

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache (although this is very often 1 cycle).

Summary on Sources of Cache Misses

Compulsory (cold start, 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**

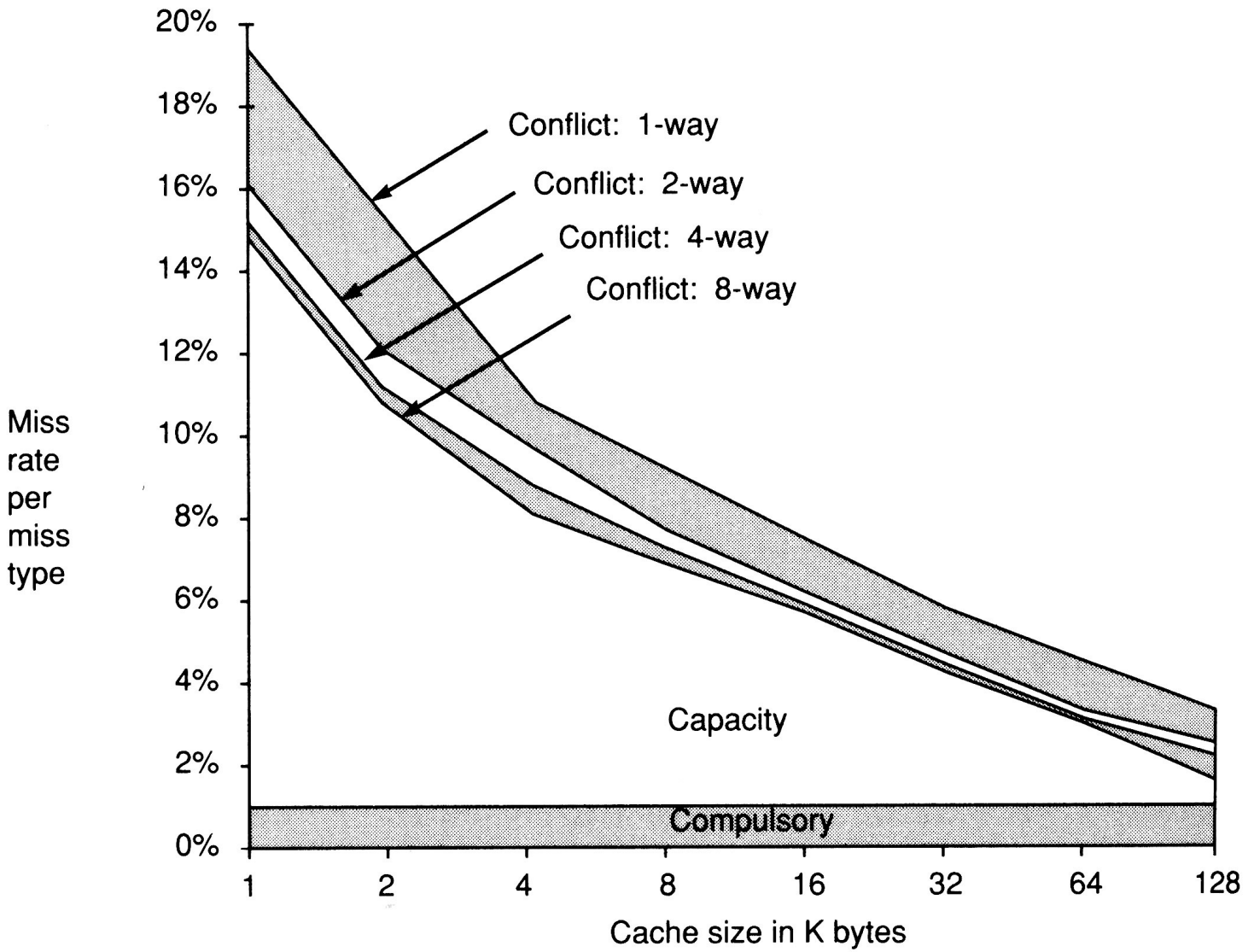
Conflict (collision):

- **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


20%
18%
16%
14%
12%
10%
8%
6%
4%
2%
0%
Miss rate per miss type
Conflict: 1-way
Conflict: 2-way
Conflict: 4-way
Conflict: 8-way
Capacity
Compulsory
1
2
4
8
16
32
64
128
Cache size in K bytes

4 Questions for Caches (and Memory Hierarchy)

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)

Q1: Where can a block be placed in the upper level?

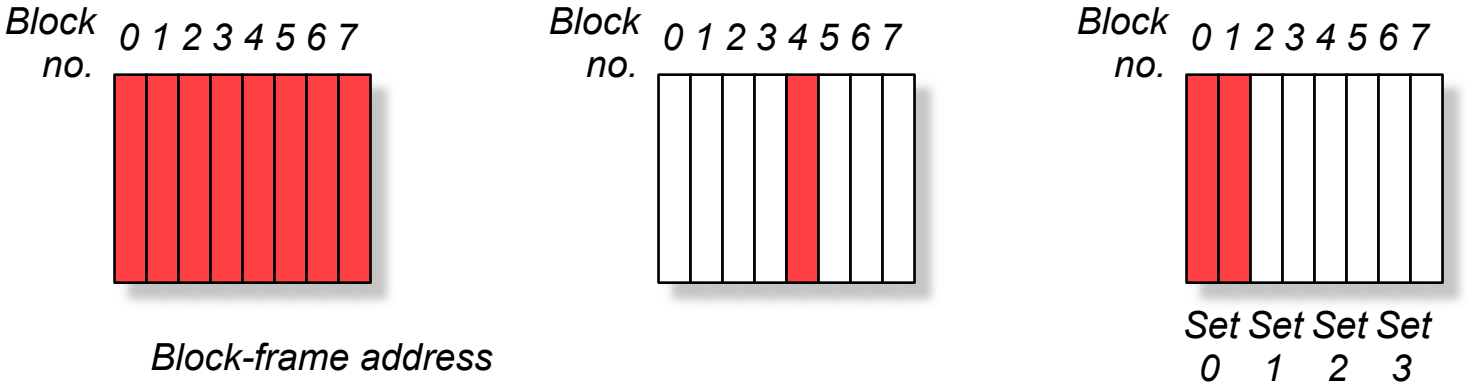
Block 12 placed in 8 block cache:

- **Fully associative, direct mapped, 2-way set associative**
- **S.A. Mapping = Block Number Modulo Number Sets**

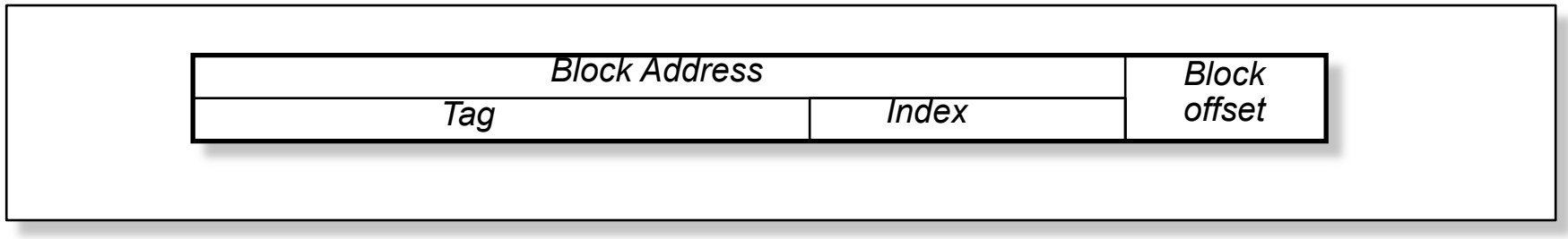
Fully associative: block 12 can go anywhere

Direct mapped: block 12 can go only into block 4 (12 mod 8)

Set associative: block 12 can go anywhere in set 0 (12 mod 4)



Q2: 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

Q3: 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%

Miss-rate

Q4: 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.

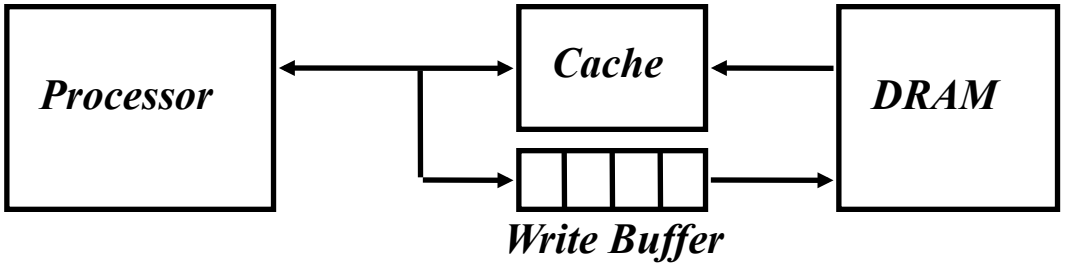
- is block clean or dirty?

Pros and Cons of each?

- WT: read misses cannot result in writes
- WB: no writes of repeated writes (saves energy)

WT always combined with write buffers so that don't wait for lower level memory

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

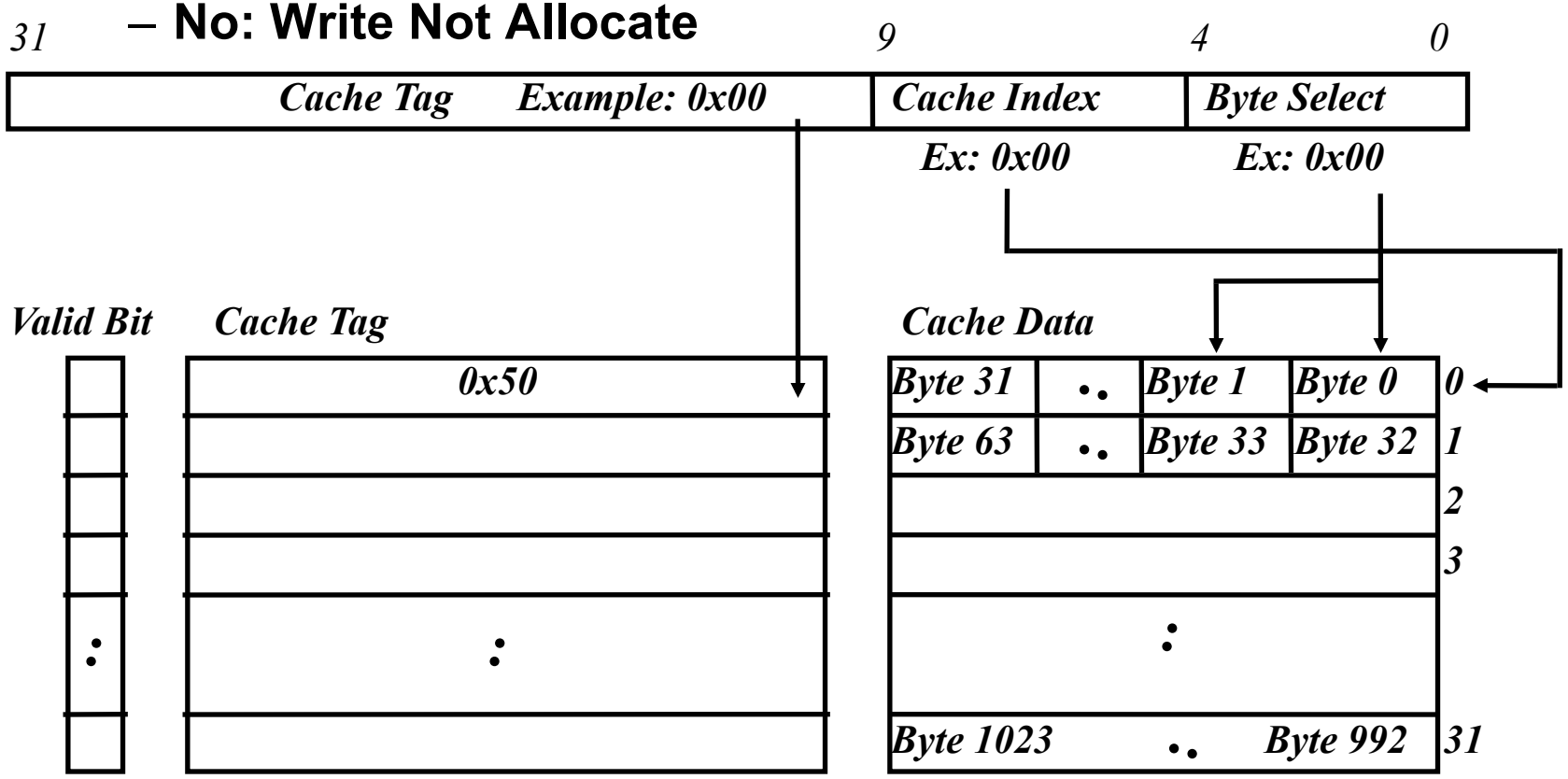
Memory system designer's nightmare:

- Store frequency (w.r.t. time) > 1 / DRAM write cycle
- Write buffer saturation

Write-miss Policy: 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**



Usually: Write-back cache uses write allocate.
Write-through cache uses no-write allocate.