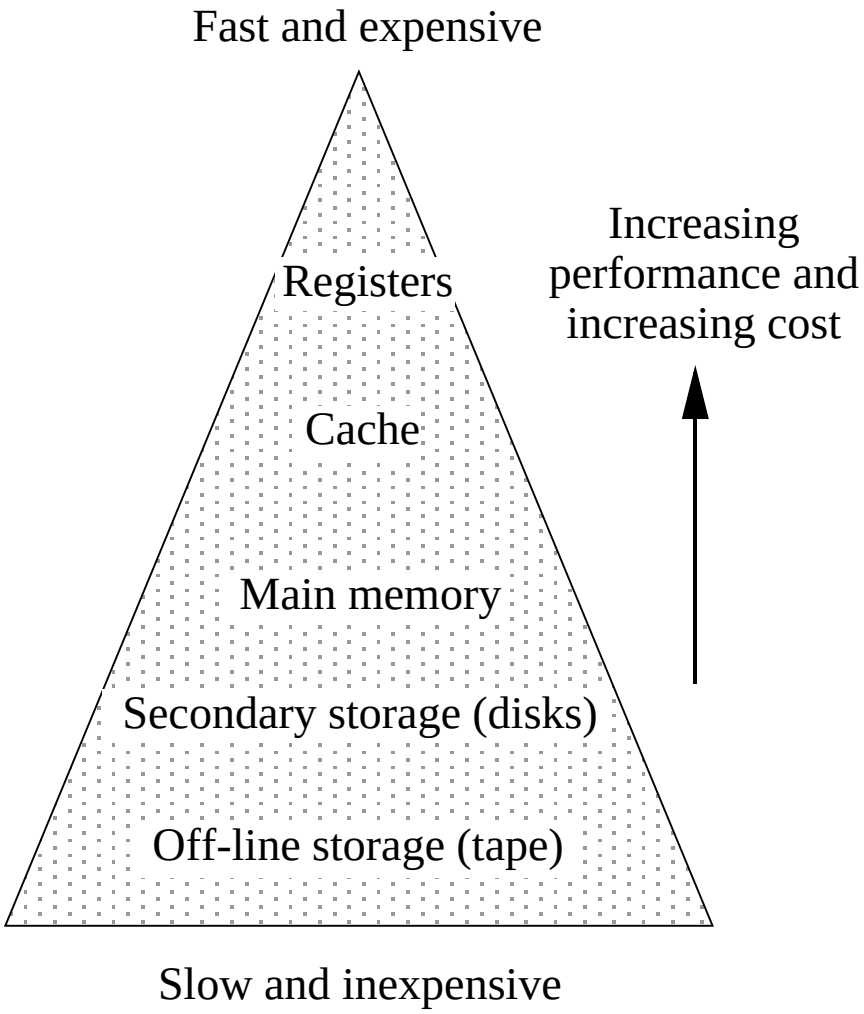
Principles of Computer Architecture

Miles Murdocca and Vincent Heuring

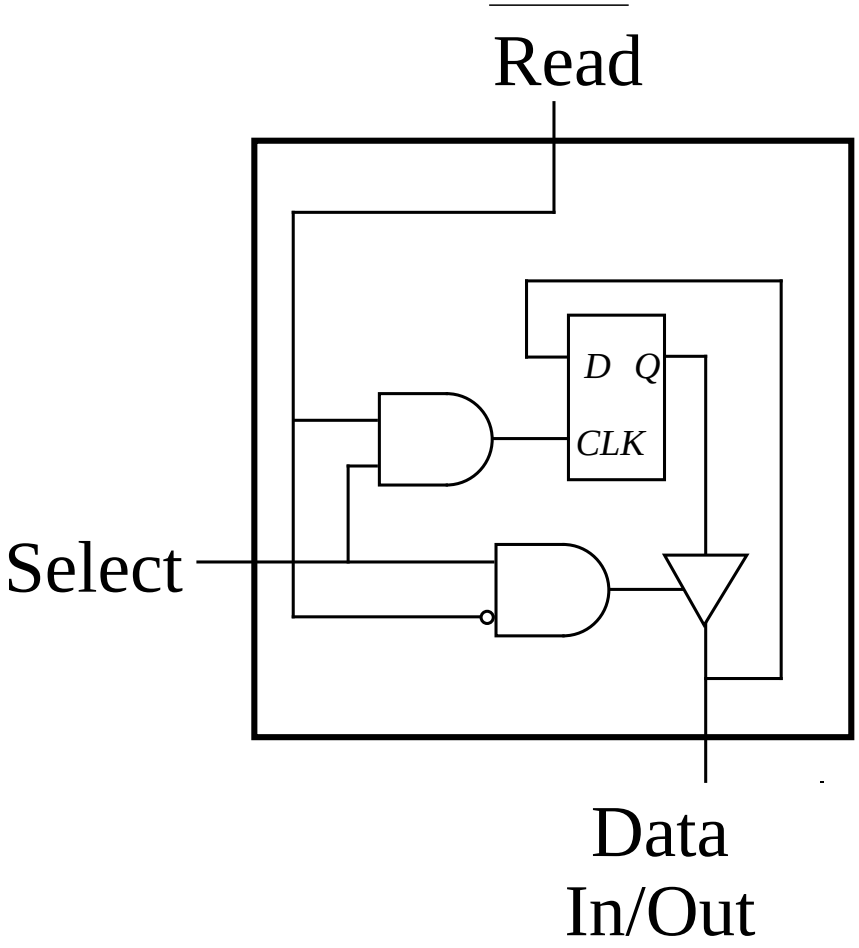
Chapter 7: Memory

Chapter Contents

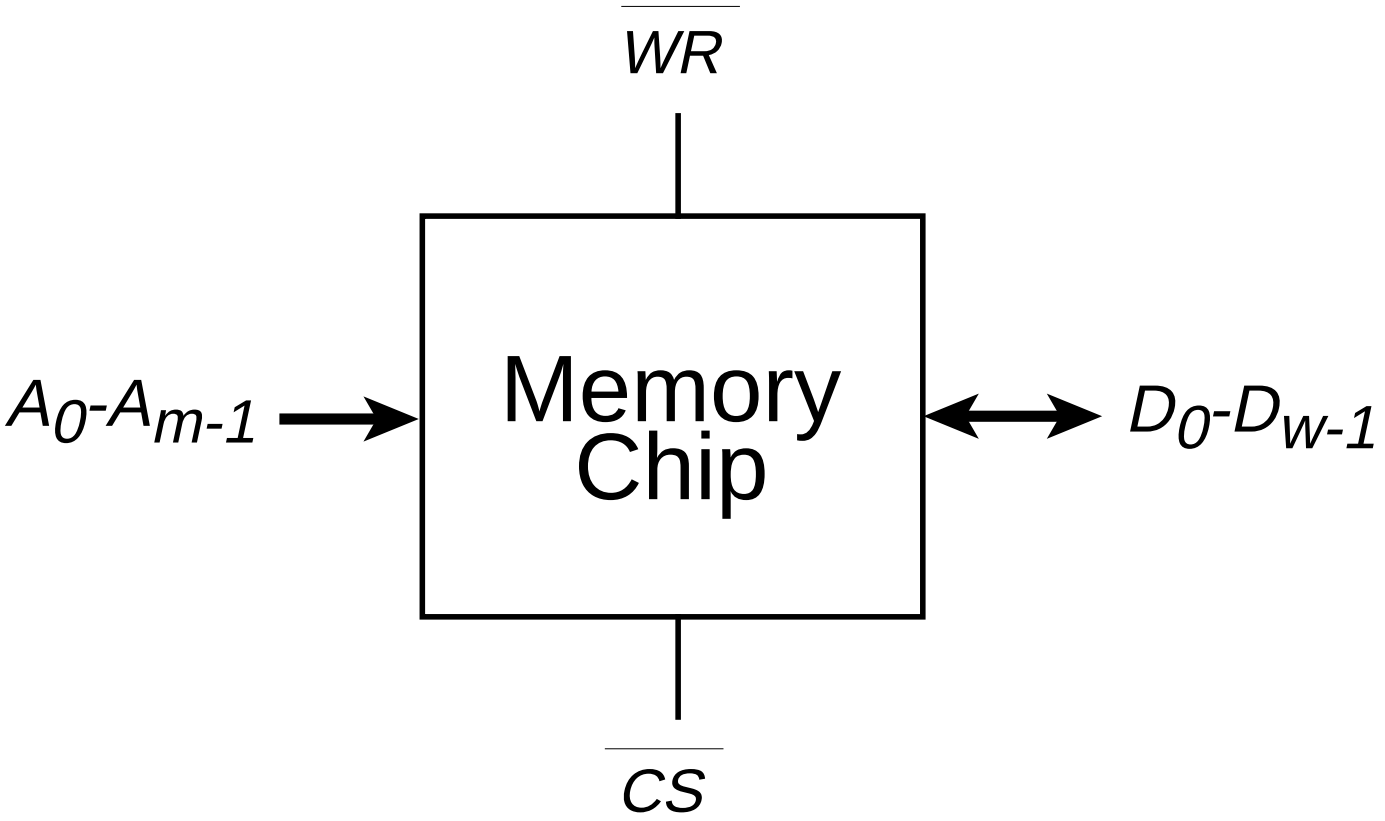
The Memory Hierarchy



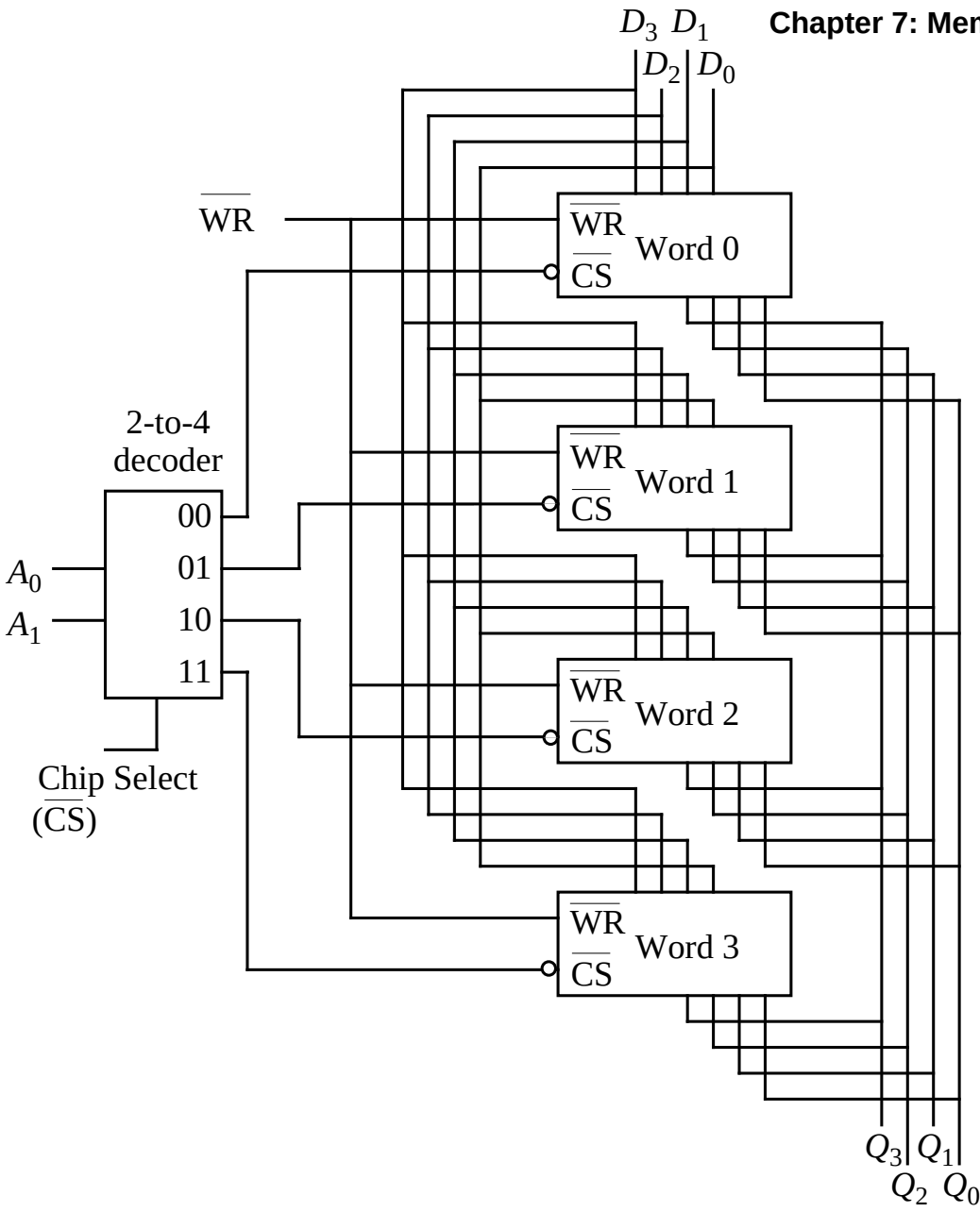
Functional Behavior of a RAM Cell



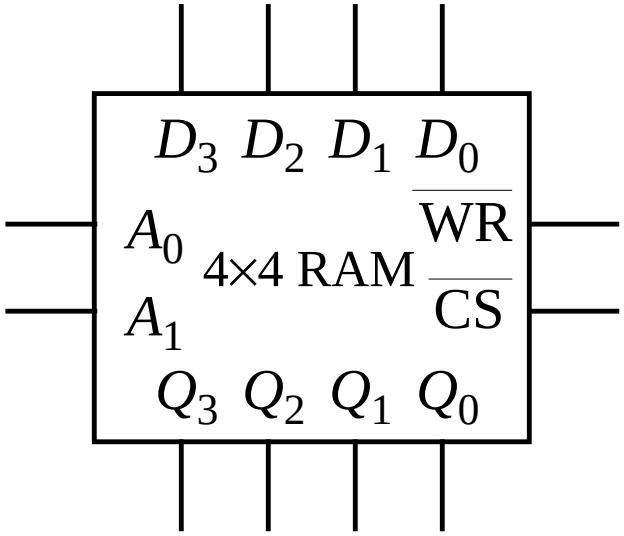
Simplified RAM Chip Pinout



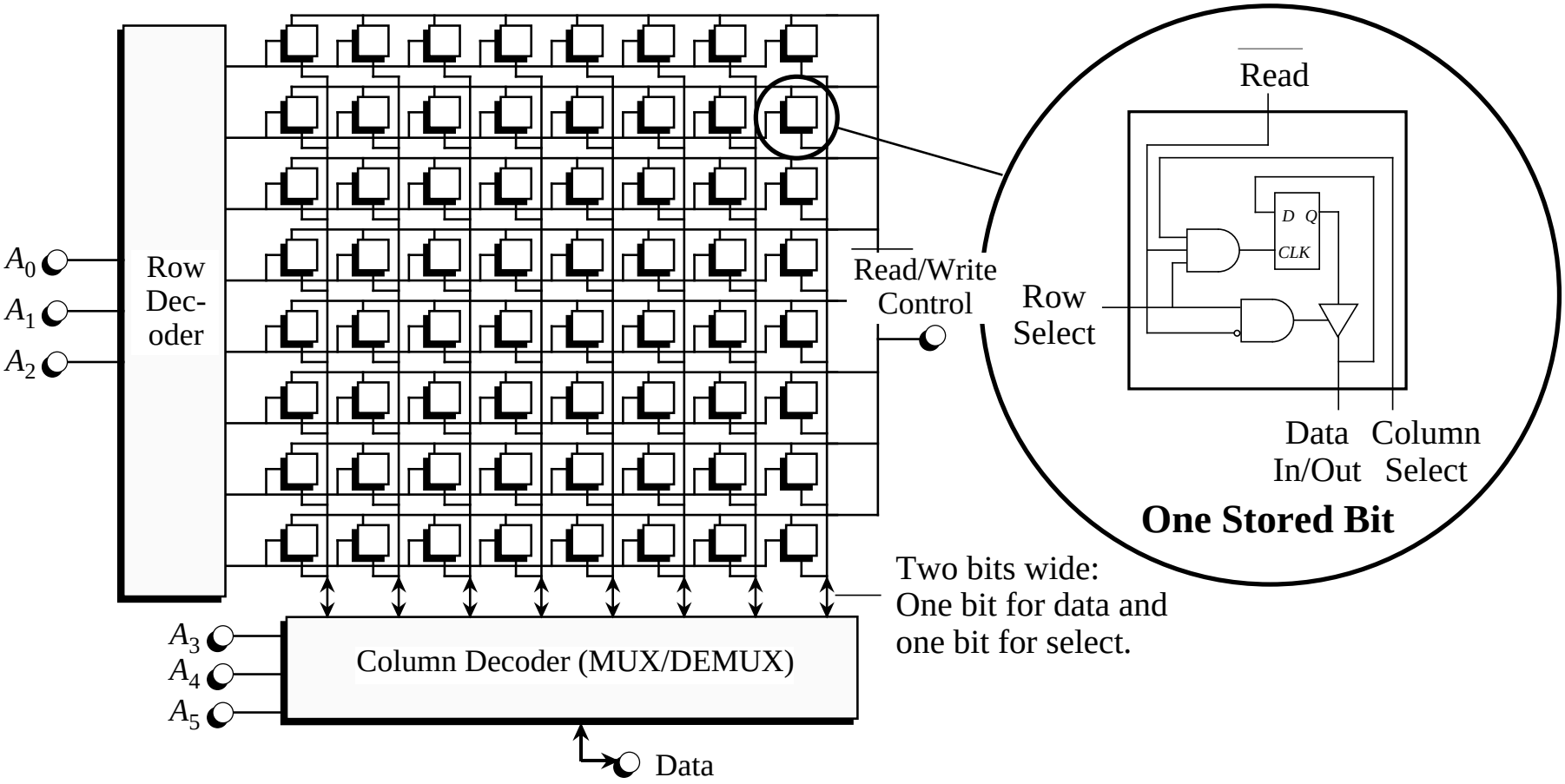
A Four-Word Memory with Four Bits per Word in a 2D Organization



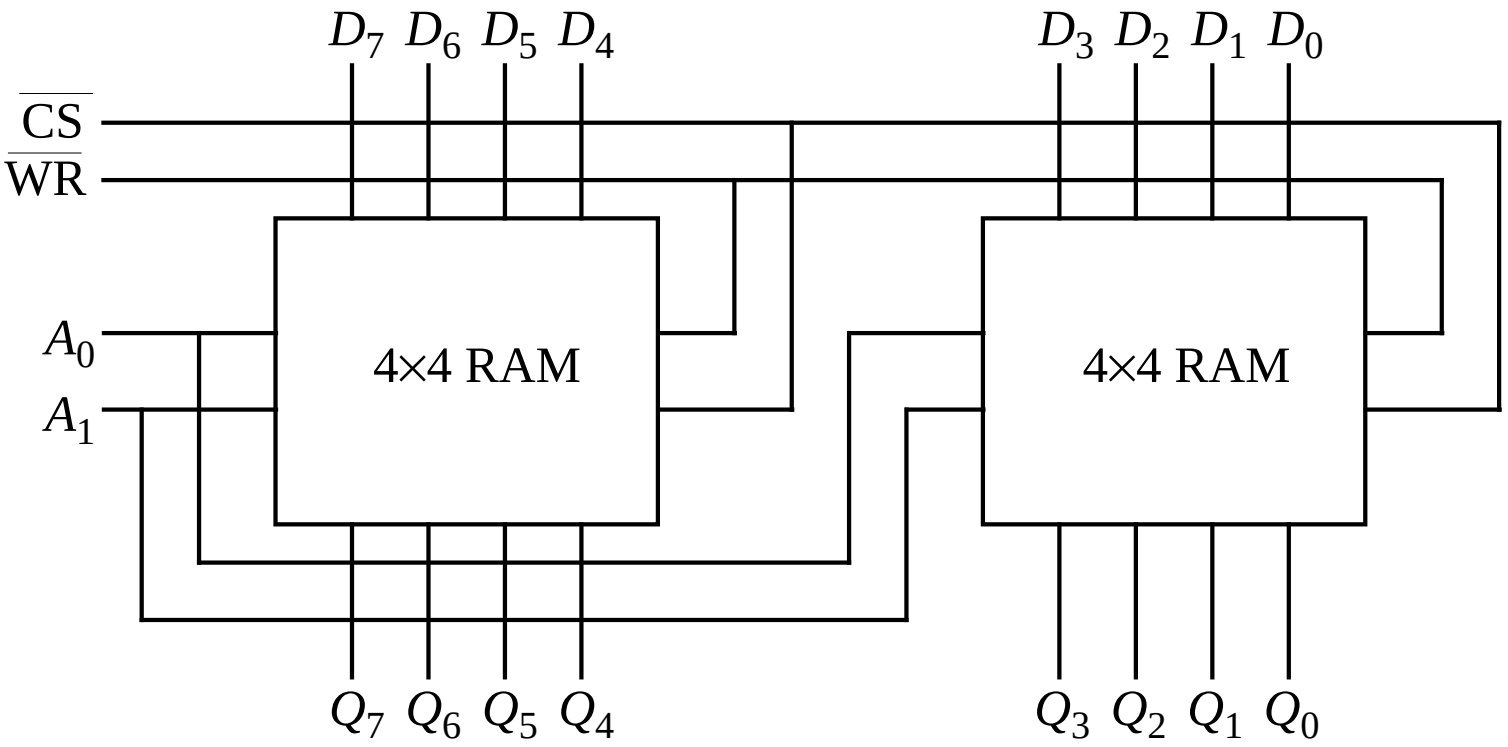
A Simplified Representation of the Four-Word by Four-Bit RAM



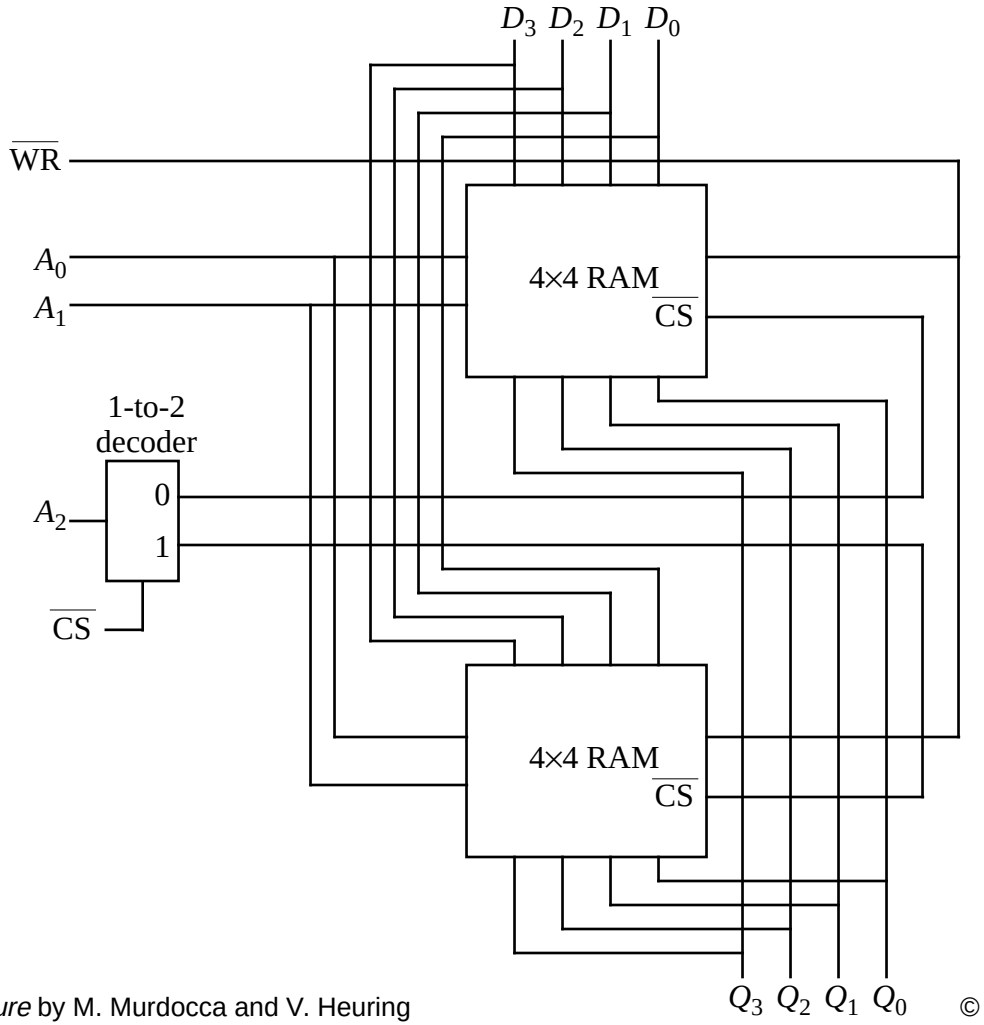
2-1/2D Organization of a 64-Word by One-Bit RAM



Two Four-Word by Four-Bit RAMs are Used in Creating a Four-Word by Eight-Bit RAM



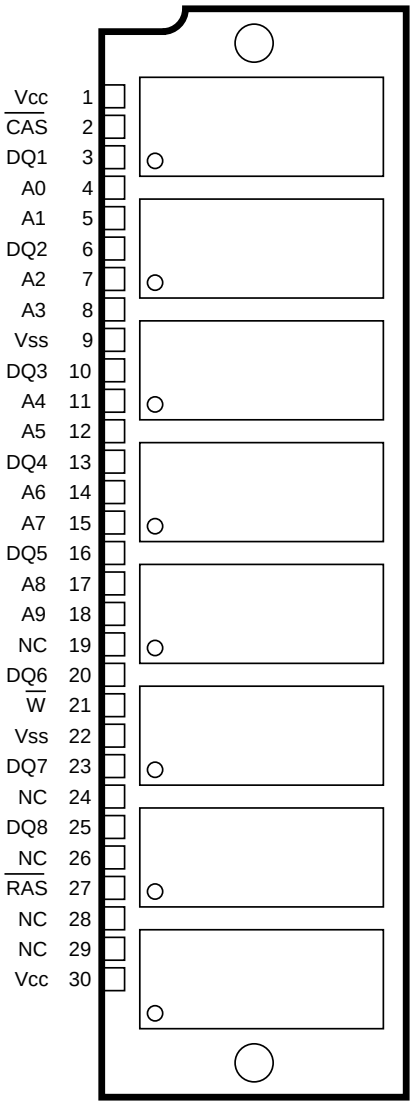
Two Four-Word by Four-Bit RAMs Make up an Eight-Word by Four-Bit RAM



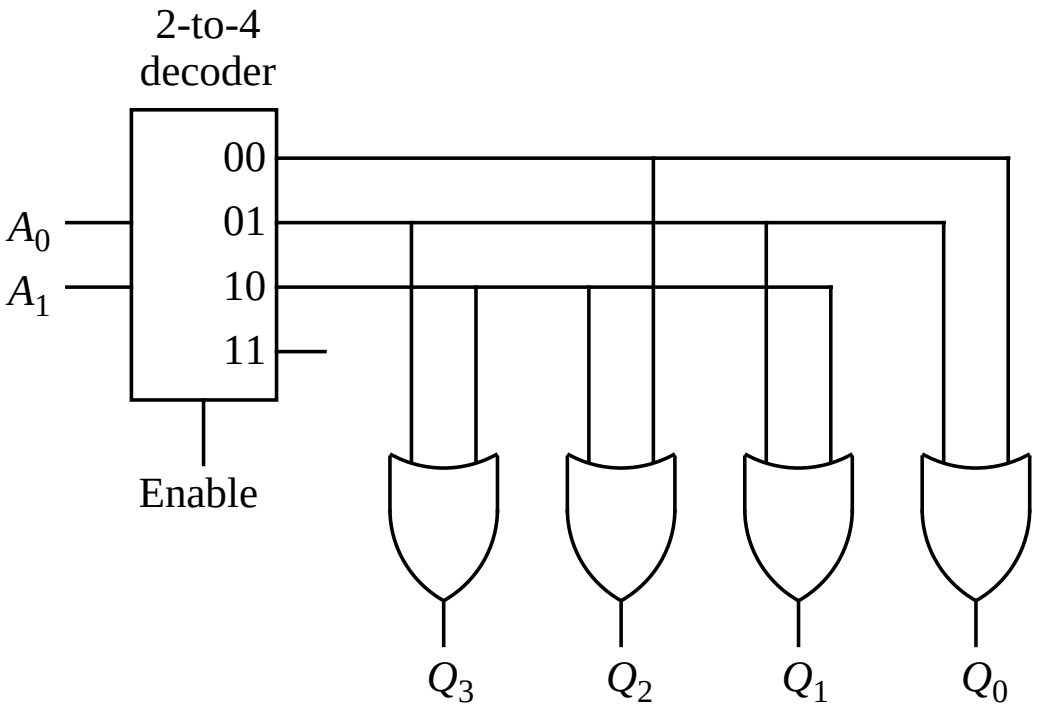
Single-In-Line Memory Module

- **Adapted from(Texas Instruments, *MOS Memory: Commercial and Military Specifications Data Book*, Texas Instruments, Literature Response Center, P.O. Box 172228, Denver, Colorado, 1991.)**

PIN NOMENCLATURE	
A0-A9	Address Inputs
$\overline{CAS}$	Column-Address Strobe
DQ1-DQ8	Data In/Data Out
NC	No Connection
$\overline{RAS}$	Row-Address Strobe
V_{cc}	5-V Supply
V_{ss}	Ground
$\overline{W}$	Write Enable

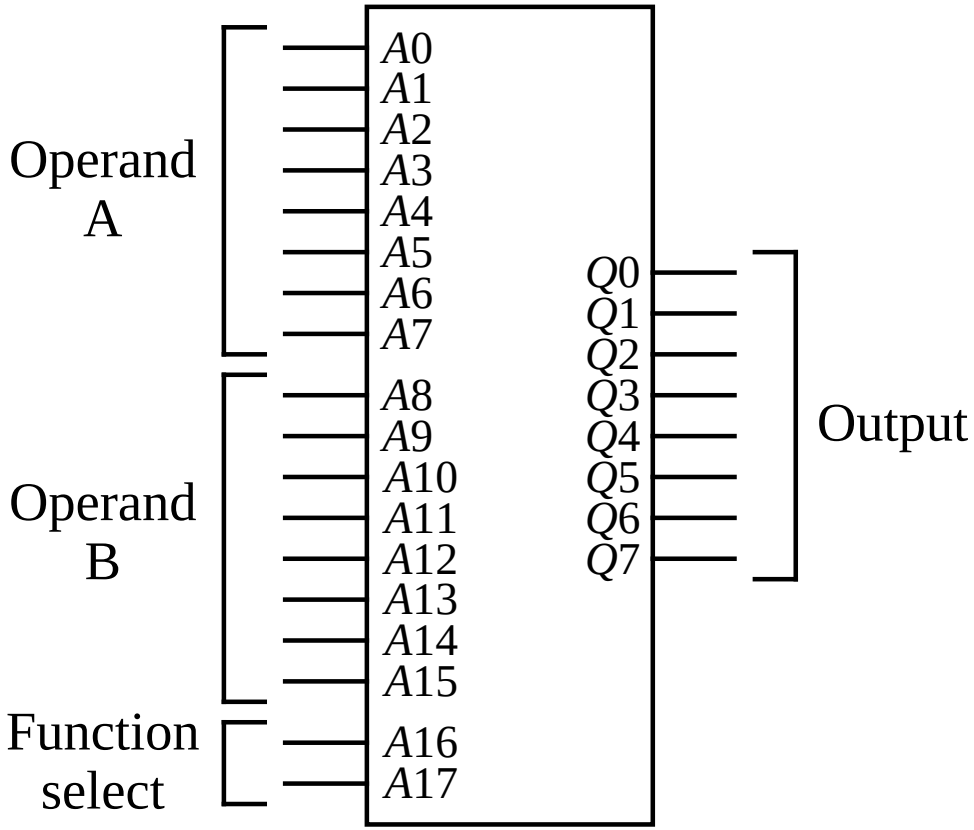


A ROM Stores Four Four-Bit Words



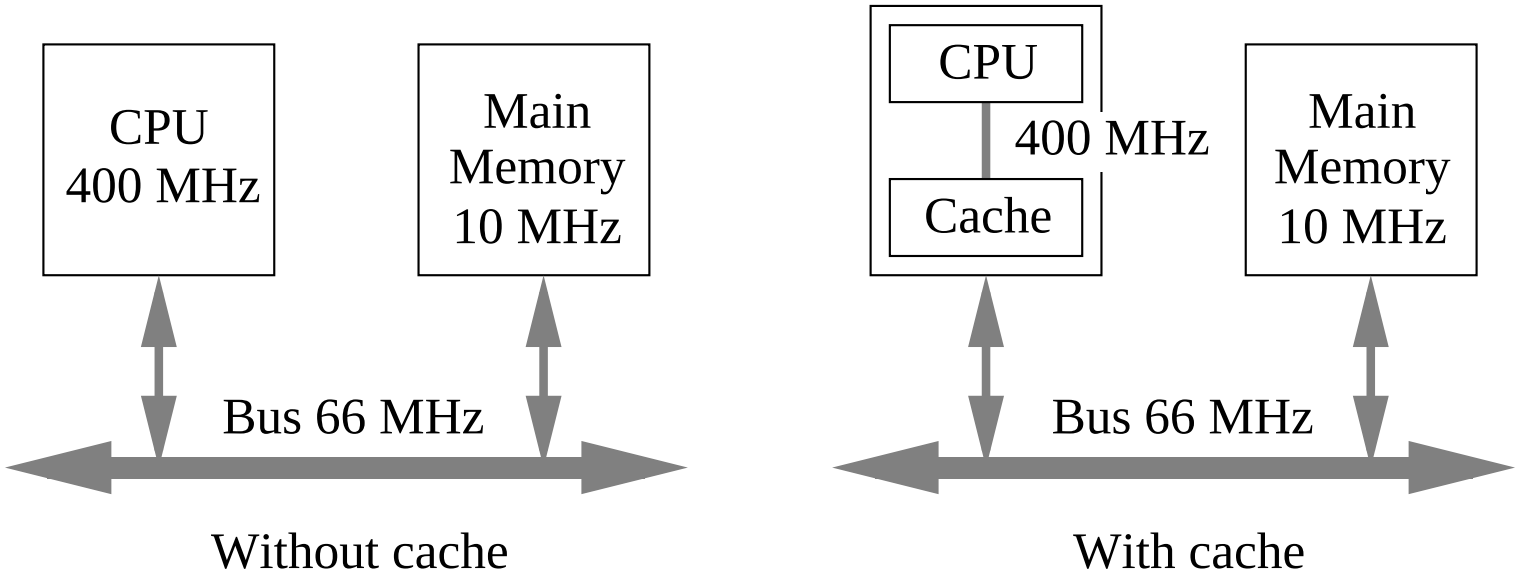
Location	Stored word
00	0101
01	1011
10	1110
11	0000

A Lookup Table (LUT) Implements an Eight-Bit ALU



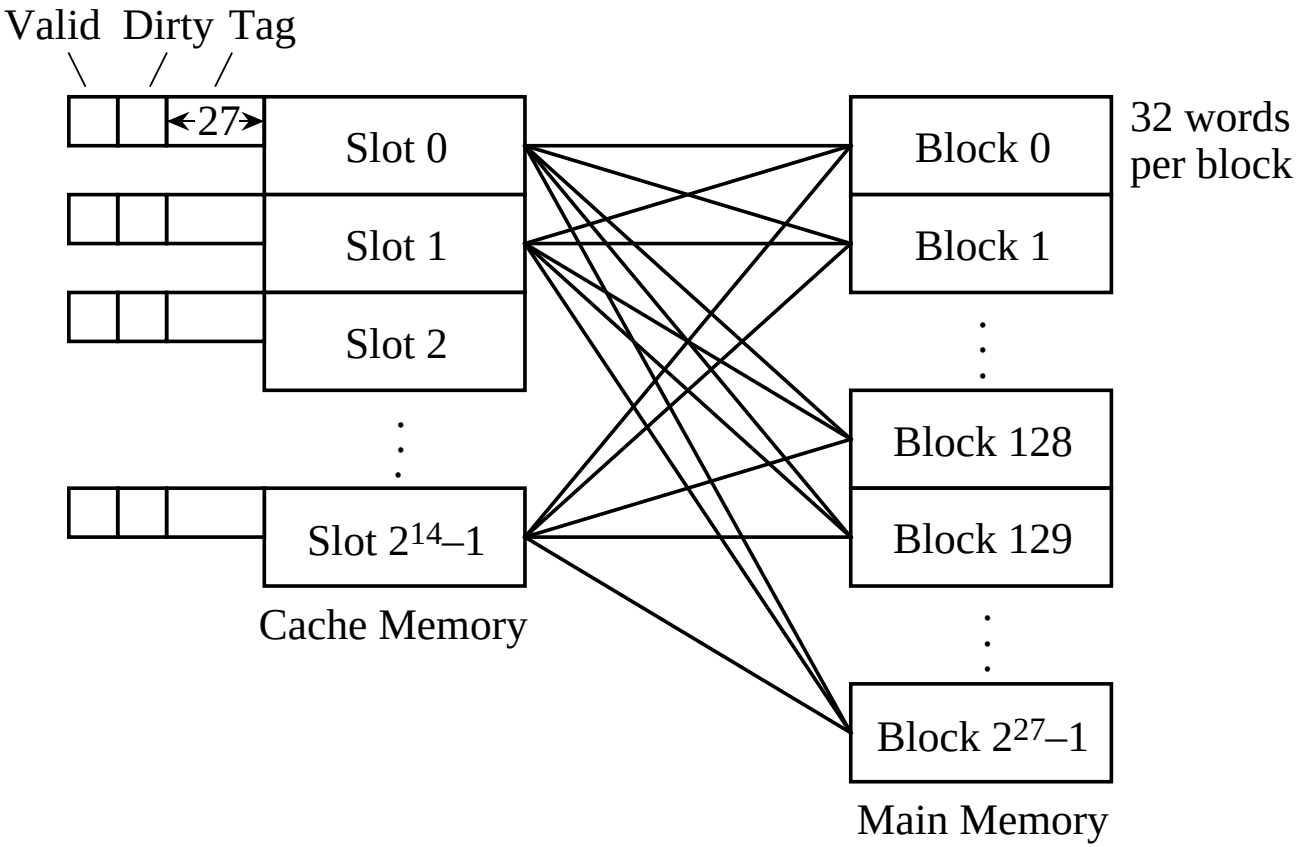
A17	A16	Function
0	0	Add
0	1	Subtract
1	0	Multiply
1	1	Divide

Placement of Cache in a Computer System



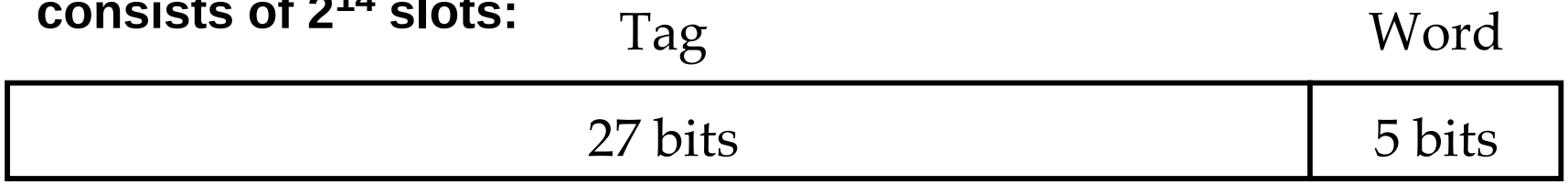
- **The *locality principle*: a recently referenced memory location is likely to be referenced again (*temporal locality*); a neighbor of a recently referenced memory location is likely to be referenced (*spatial locality*).**

An Associative Mapping Scheme for a Cache Memory

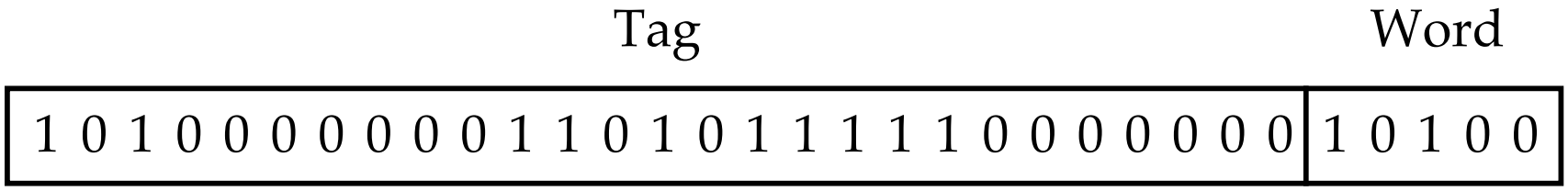


Associative Mapping Example

- Consider how an access to memory location $(A035F014)_{16}$ is mapped to the cache for a 2^{32} word memory. The memory is divided into 2^{27} blocks of $2^5 = 32$ words per block, and the cache consists of 2^{14} slots:



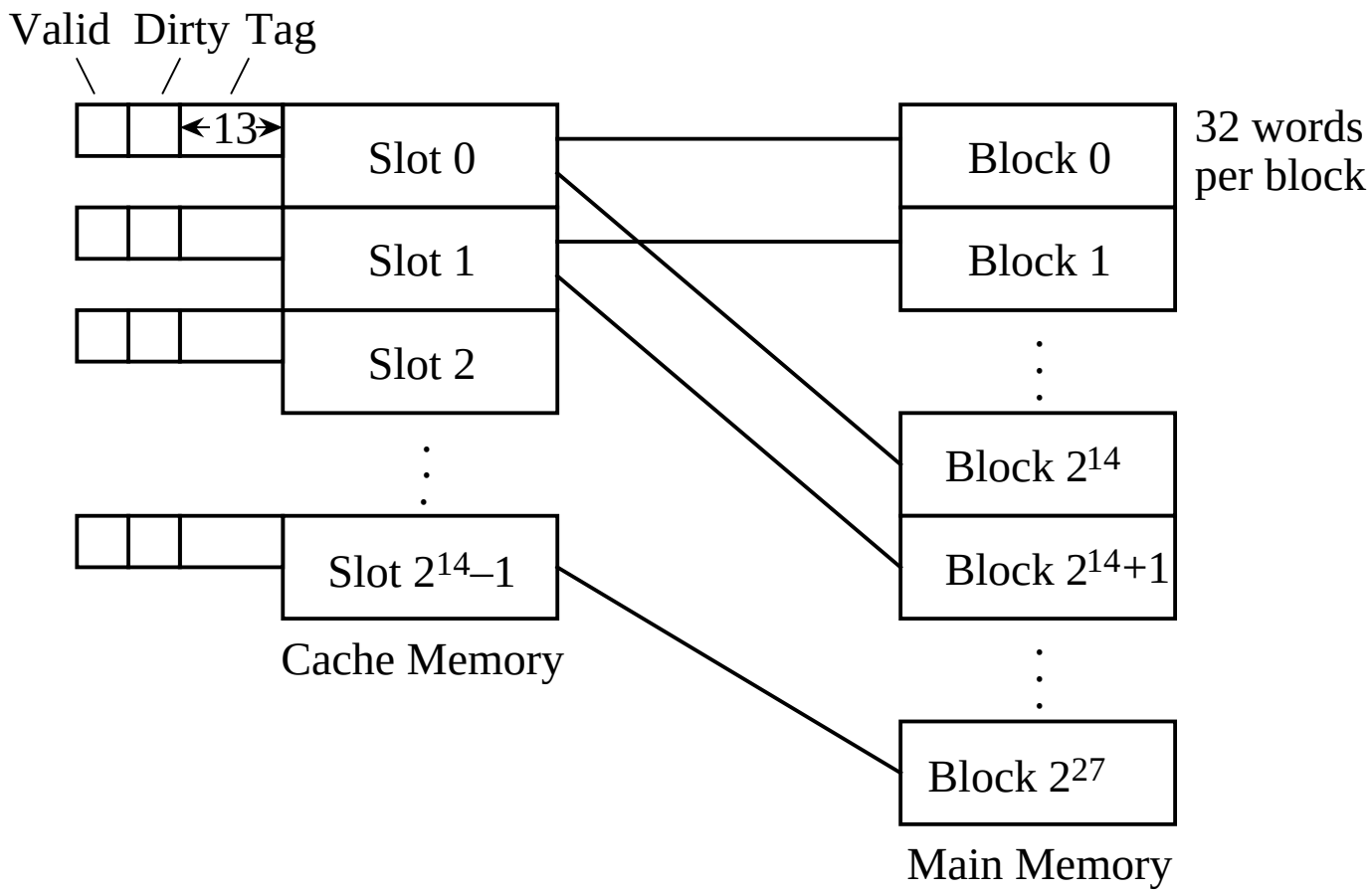
- If the addressed word is in the cache, it will be found in word $(14)_{16}$ of a slot that has tag $(501AF80)_{16}$, which is made up of the 27 most significant bits of the address. If the addressed word is not in the cache, then the block corresponding to tag field $(501AF80)_{16}$ is brought into an available slot in the cache from the main memory, and the memory reference is then satisfied from the cache.



Replacement Policies

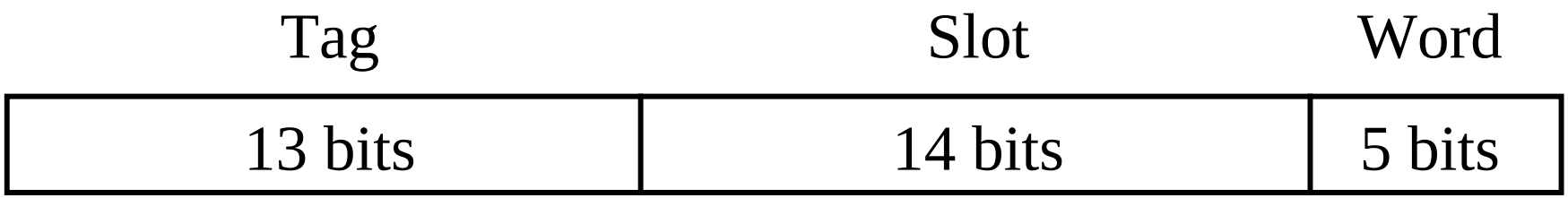
- When there are no available slots in which to place a block, a *replacement policy* is implemented. The replacement policy governs the choice of which slot is freed up for the new block.
- Replacement policies are used for associative and set-associative mapping schemes, and also for virtual memory.
- Least recently used (LRU)
- First-in/first-out (FIFO)
- Least frequently used (LFU)
- Random
- Optimal (used for analysis only – look backward in time and reverse-engineer the best possible strategy for a particular sequence of memory references.)

A Direct Mapping Scheme for Cache Memory

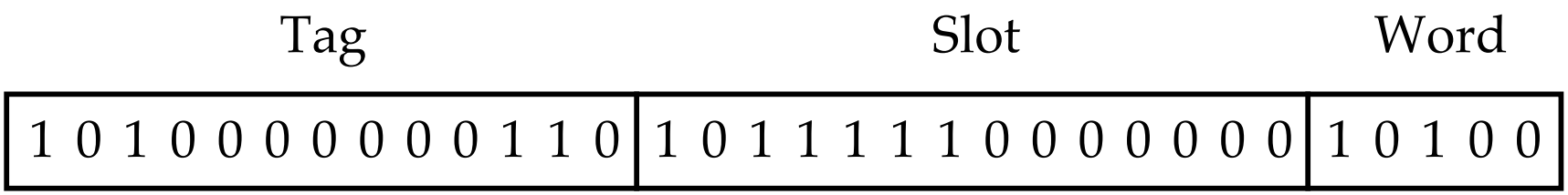


Direct Mapping Example

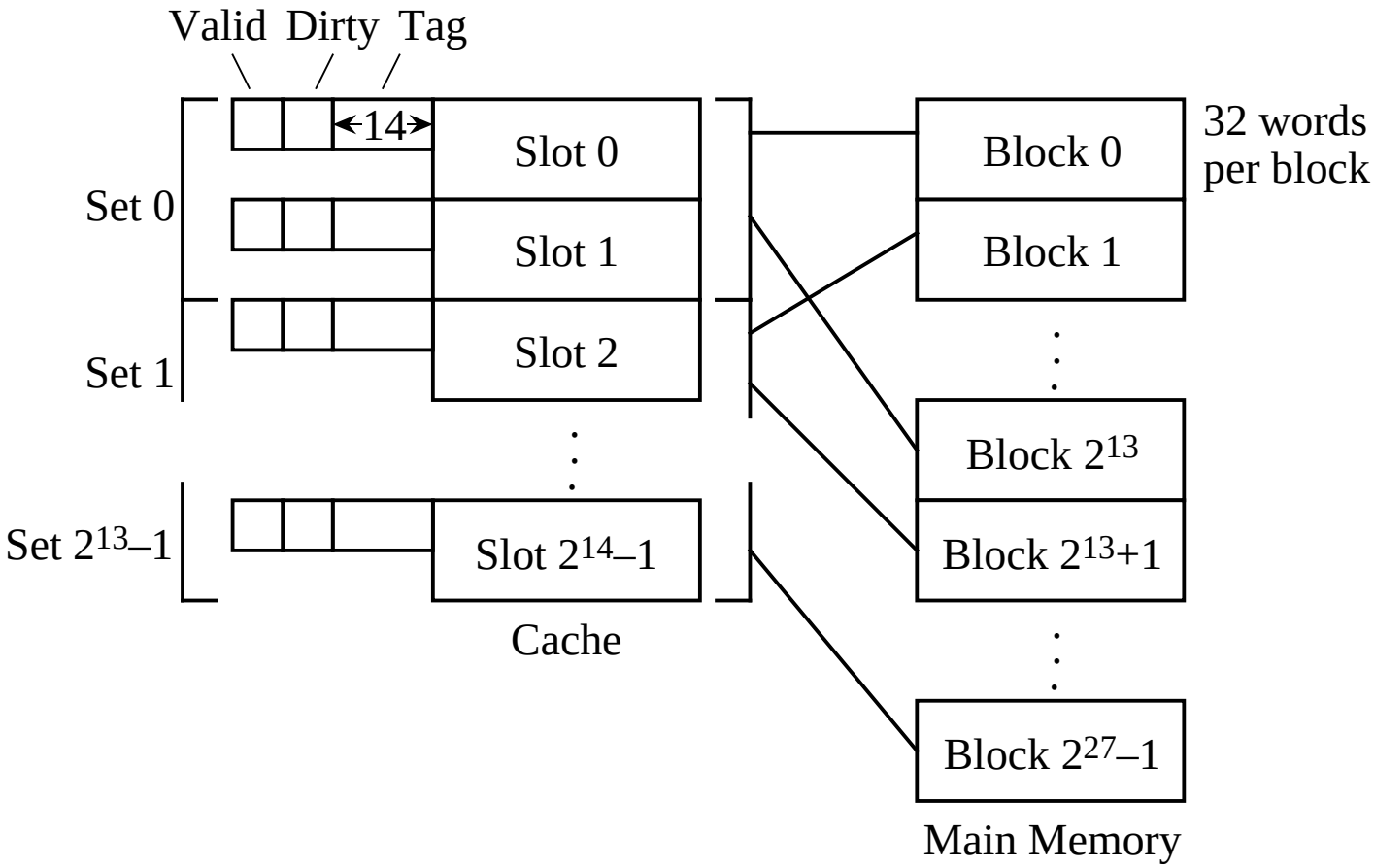
- **For a direct mapped cache, each main memory block can be mapped to only one slot, but each slot can receive more than one block. Consider how an access to memory location $(A035F014)_{16}$ is mapped to the cache for a 2^{32} word memory. The memory is divided into 2^{27} blocks of $2^5 = 32$ words per block, and the cache consists of 2^{14} slots:**



- **If the addressed word is in the cache, it will be found in word $(14)_{16}$ of slot $(2F80)_{16}$, which will have a tag of $(1406)_{16}$.**

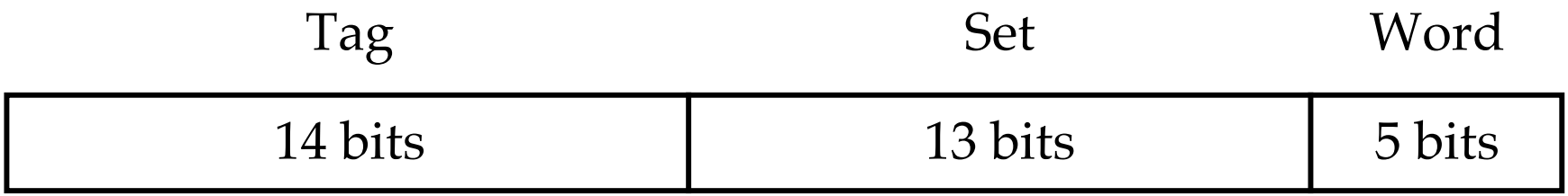


A Set Associative Mapping Scheme for a Cache Memory

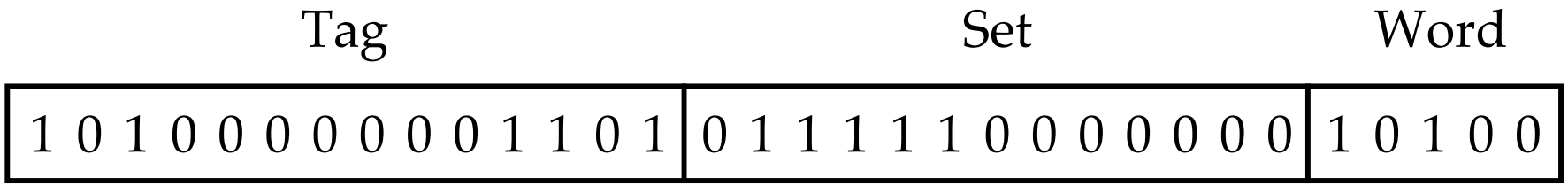


Set-Associative Mapping Example

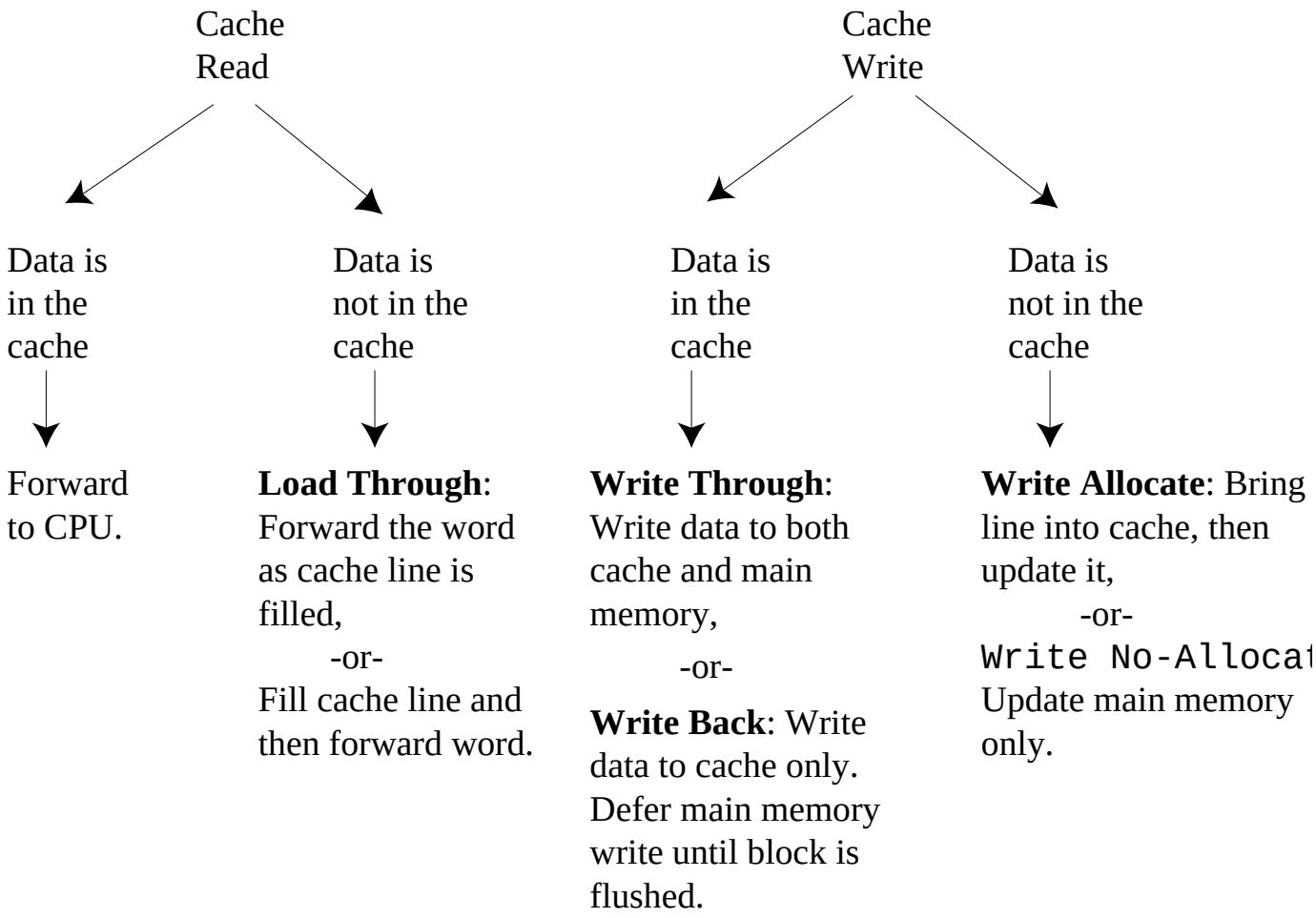
- **Consider how an access to memory location $(A035F014)_{16}$ is mapped to the cache for a 2^{32} word memory. The memory is divided into 2^{27} blocks of $2^5 = 32$ words per block, there are two blocks per set, and the cache consists of 2^{14} slots:**



- **The leftmost 14 bits form the tag field, followed by 13 bits for the set field, followed by five bits for the word field:**



Cache Read and Write Policies



Hit Ratios and Effective Access Times

- **Hit ratio and effective access time for single level cache:**

$$Hit\ ratio = \frac{No.\ times\ referenced\ words\ are\ in\ cache}{Total\ number\ of\ memory\ accesses}$$

$$Eff.\ access\ time = \frac{(\#\ hits)(Time\ per\ hit) + (\#\ misses)(Time\ per\ miss)}{Total\ number\ of\ memory\ access}$$

- **Hit ratios and effective access time for multi-level cache:**

$$H_1 = \frac{No.\ times\ accessed\ word\ is\ in\ on\text{-}chip\ cache}{Total\ number\ of\ memory\ accesses}$$

$$H_2 = \frac{No.\ times\ accessed\ word\ is\ in\ off\text{-}chip\ cache}{No.\ times\ accessed\ word\ is\ not\ in\ on\text{-}chip\ cache}$$

$$T_{EFF} = (No.\ on\text{-}chip\ cache\ hits)(On\text{-}chip\ cache\ hit\ time) +$$
$$(No.\ off\text{-}chip\ cache\ hits)(Off\text{-}chip\ cache\ hit\ time) +$$
$$(No.\ off\text{-}chip\ cache\ misses)(Off\text{-}chip\ cache\ miss\ time)$$
$$/\ Total\ number\ of\ memory\ accesses$$

Direct Mapped Cache Example

- **Compute hit ratio and effective access time for a program that executes from memory locations 48 to 95, and then loops 10 times from 15 to 31.**

- **The direct mapped cache has four 16-word slots, a hit time of 80 ns, and a miss time of 2500 ns. Load-through is used. The cache is initially empty.**

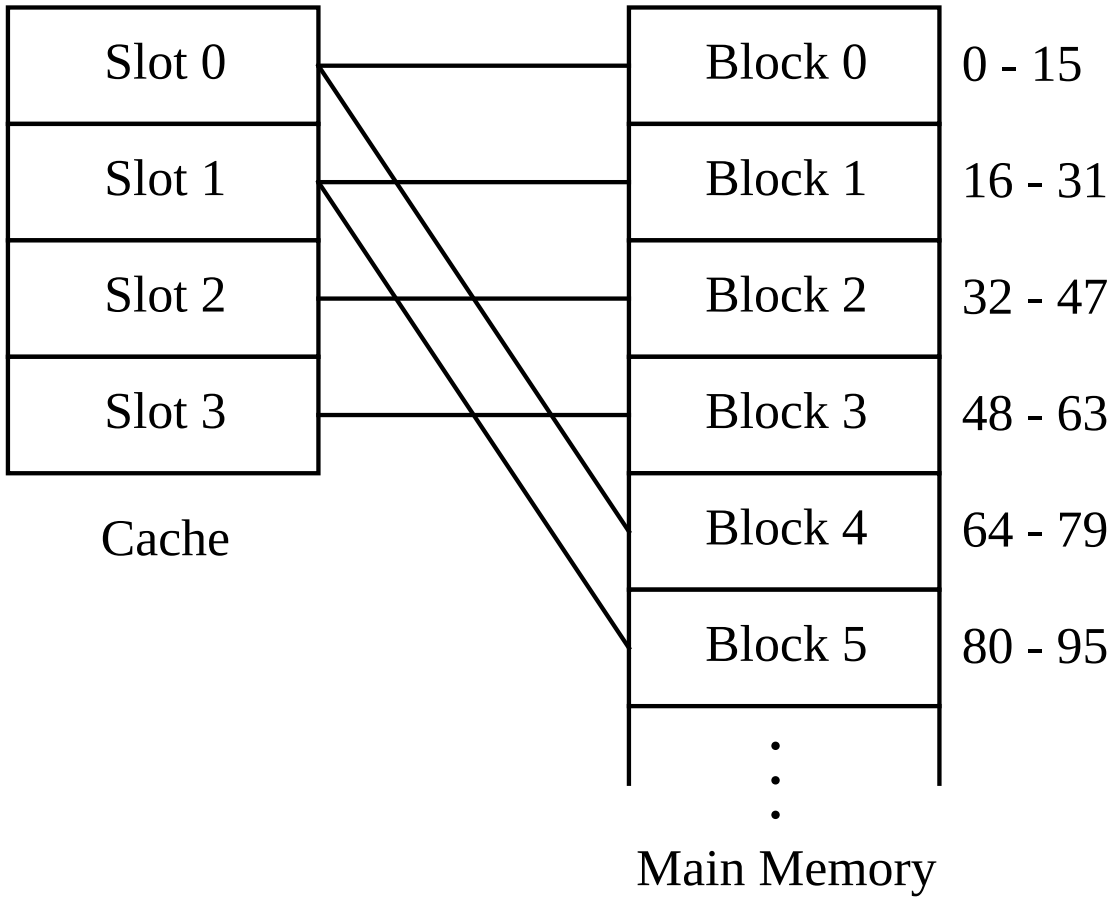


Table of Events for Example Program

Event	Location	Time	Comment
1 miss	48	2500ns	Memory block 3 to cache slot 3
15 hits	49-63	80ns×15=1200ns	
1 miss	64	2500ns	Memory block 4 to cache slot 0
15 hits	65-79	80ns×15=1200ns	
1 miss	80	2500ns	Memory block 5 to cache slot 1
15 hits	81-95	80ns×15=1200ns	
1 miss	15	2500ns	Memory block 0 to cache slot 0
1 miss	16	2500ns	Memory block 1 to cache slot 1
15 hits	17-31	80ns×15=1200ns	
9 hits	15	80ns×9=720ns	Last nine iterations of loop
144 hits	16-31	80ns×144=12,240ns	Last nine iterations of loop
Total hits = 213		Total misses = 5	

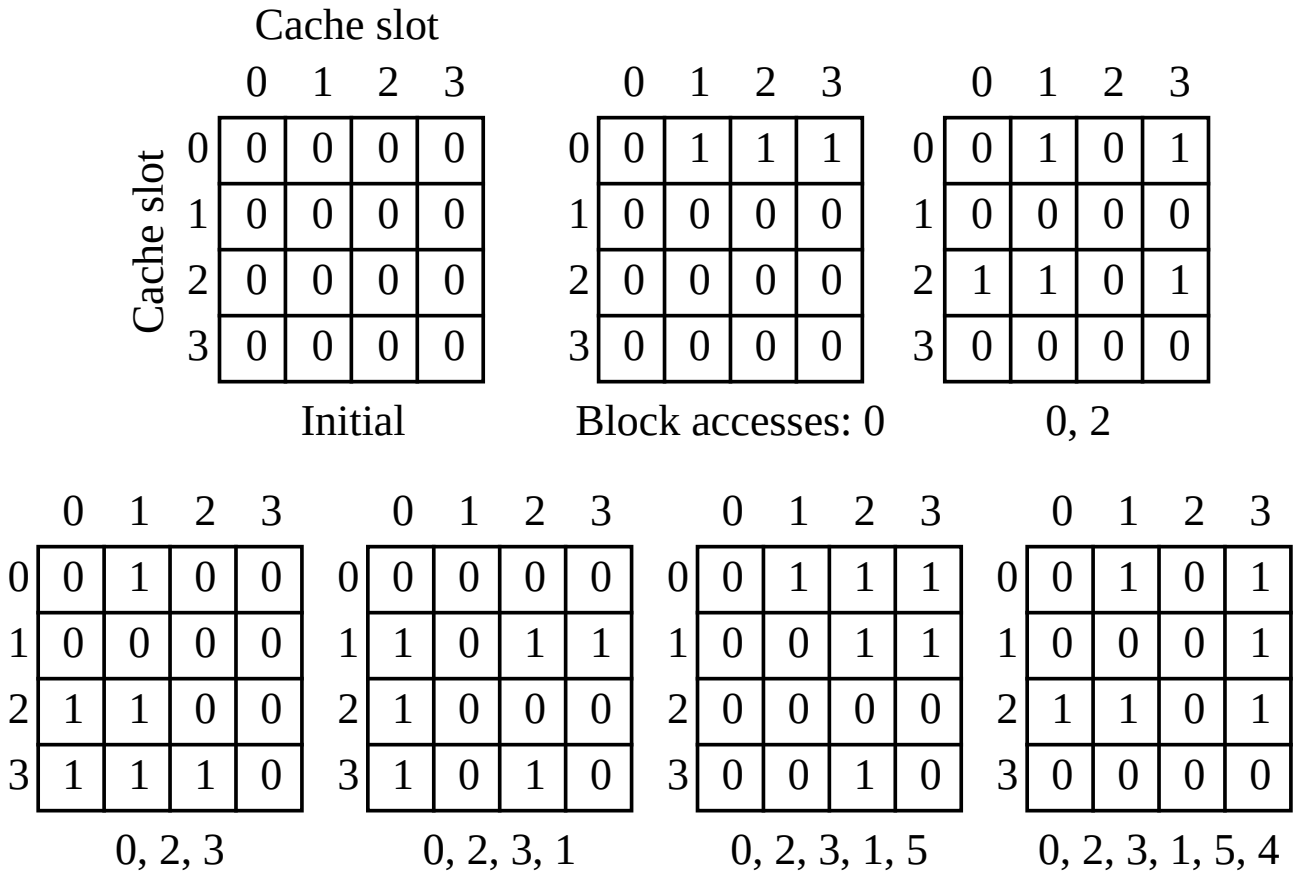
Calculation of Hit Ratio and Effective Access Time for Example Program

$$Hit\ ratio = \frac{213}{218} = 97.7\%$$

$$Effective Access Time = \frac{(213)(80ns) + (5)(2500ns)}{218} = 136ns$$

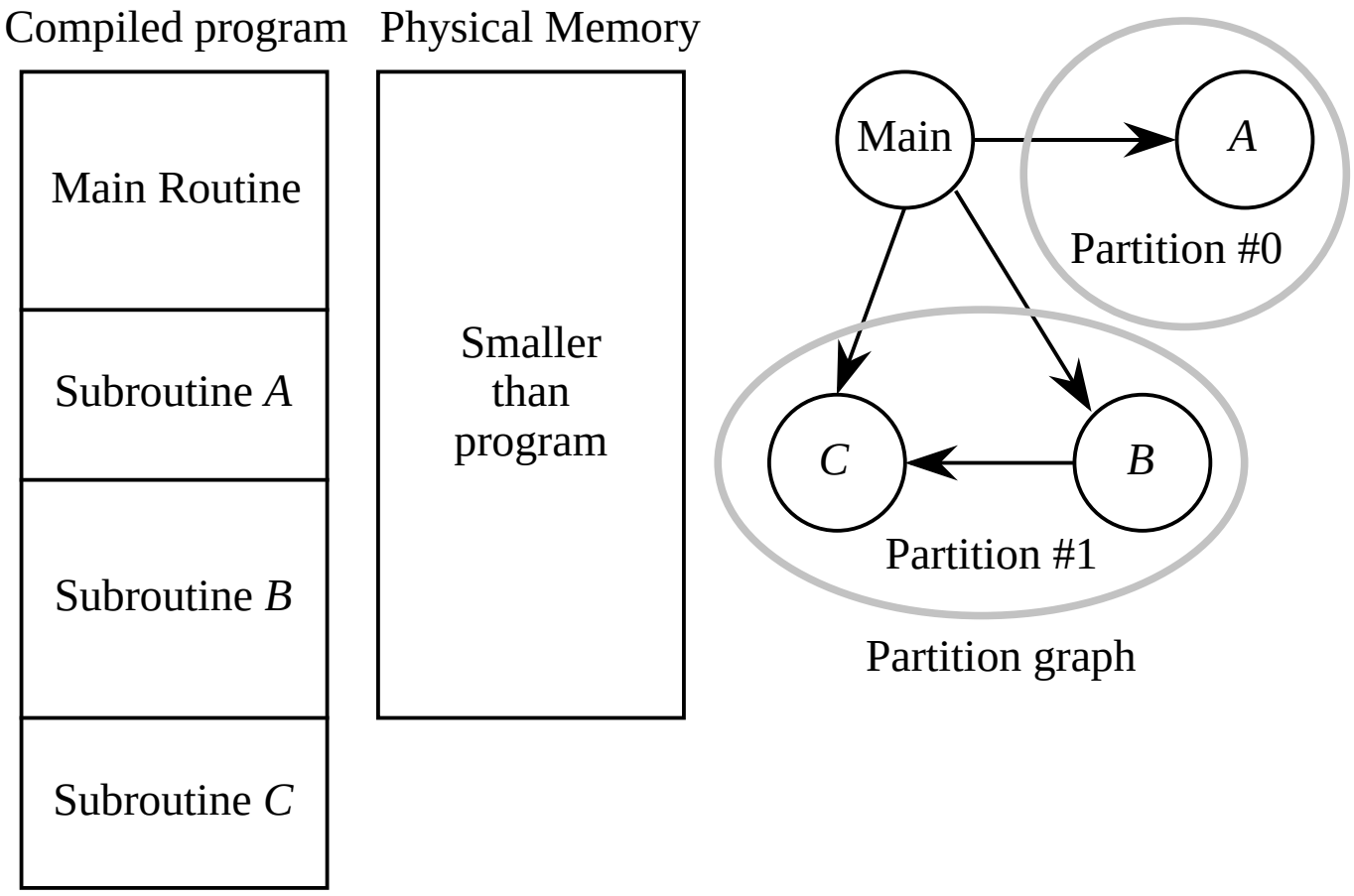
Neat Little LRU Algorithm

- **A sequence is shown for the Neat Little LRU Algorithm for a cache with four slots. Main memory blocks are accessed in the sequence: 0, 2, 3, 1, 5, 4.**



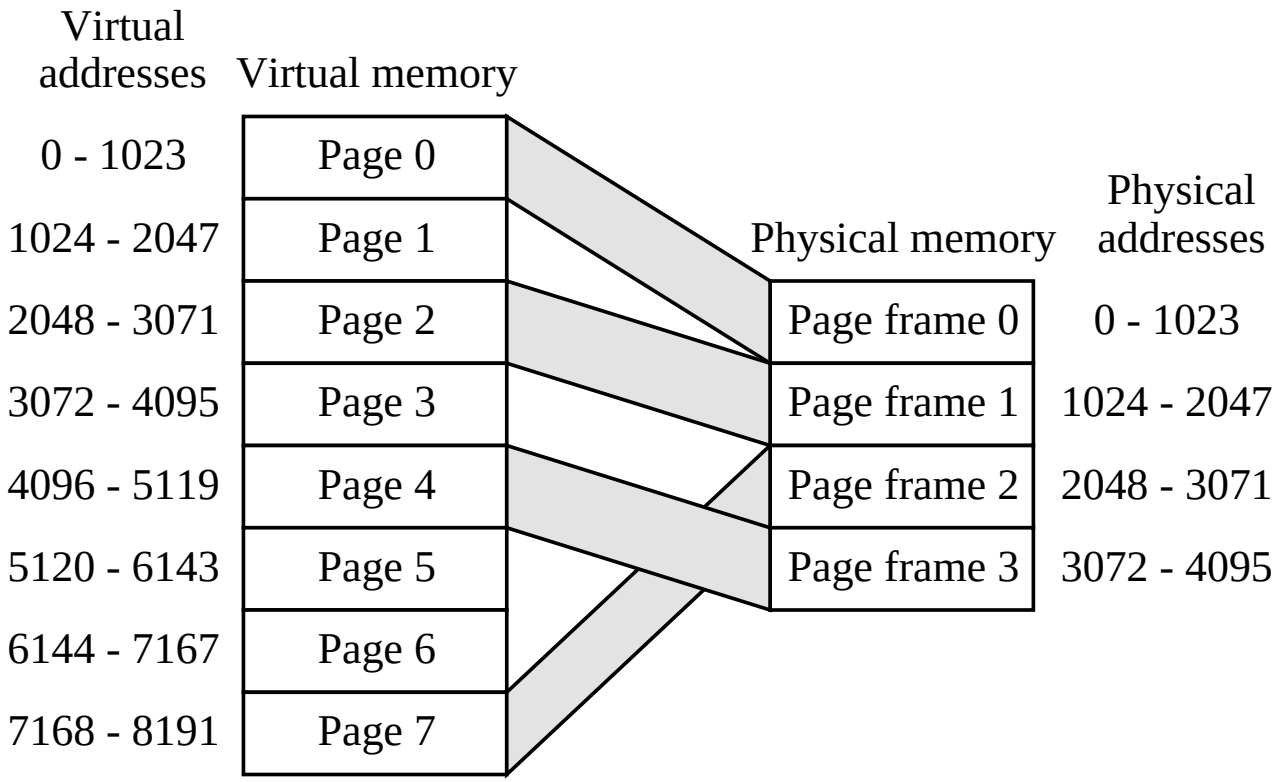
Overlays

- **A partition graph for a program with a main routine and three sub-routines:**



Virtual Memory

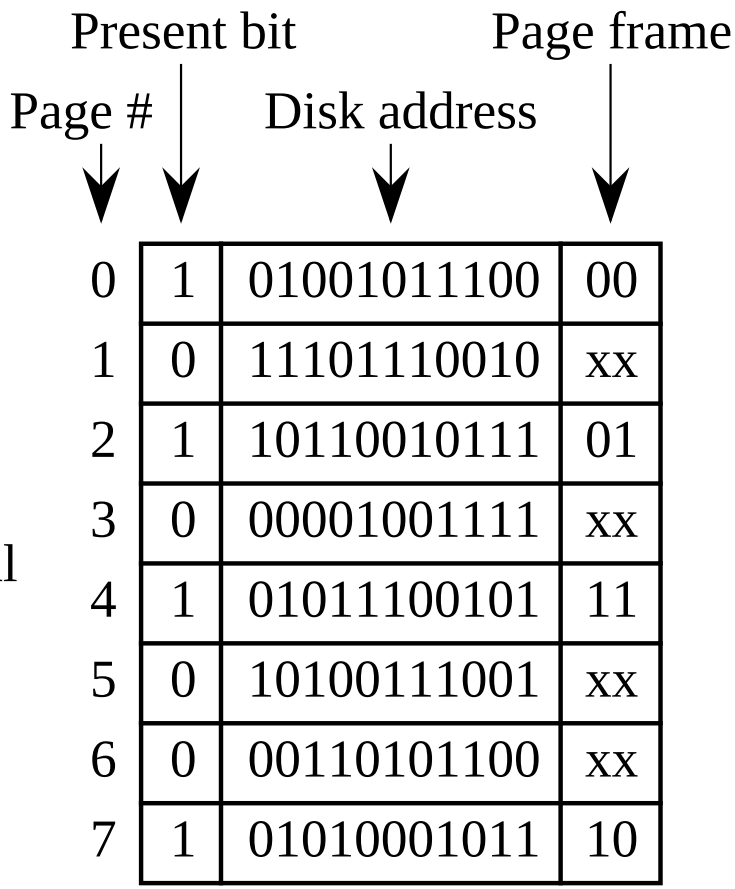
- **Virtual memory is stored in a hard disk image. The physical memory holds a small number of virtual *pages* in physical *page frames*.**
- **A mapping between a virtual and a physical memory:**



Page Table

- **The page table maps between virtual memory and physical memory.**

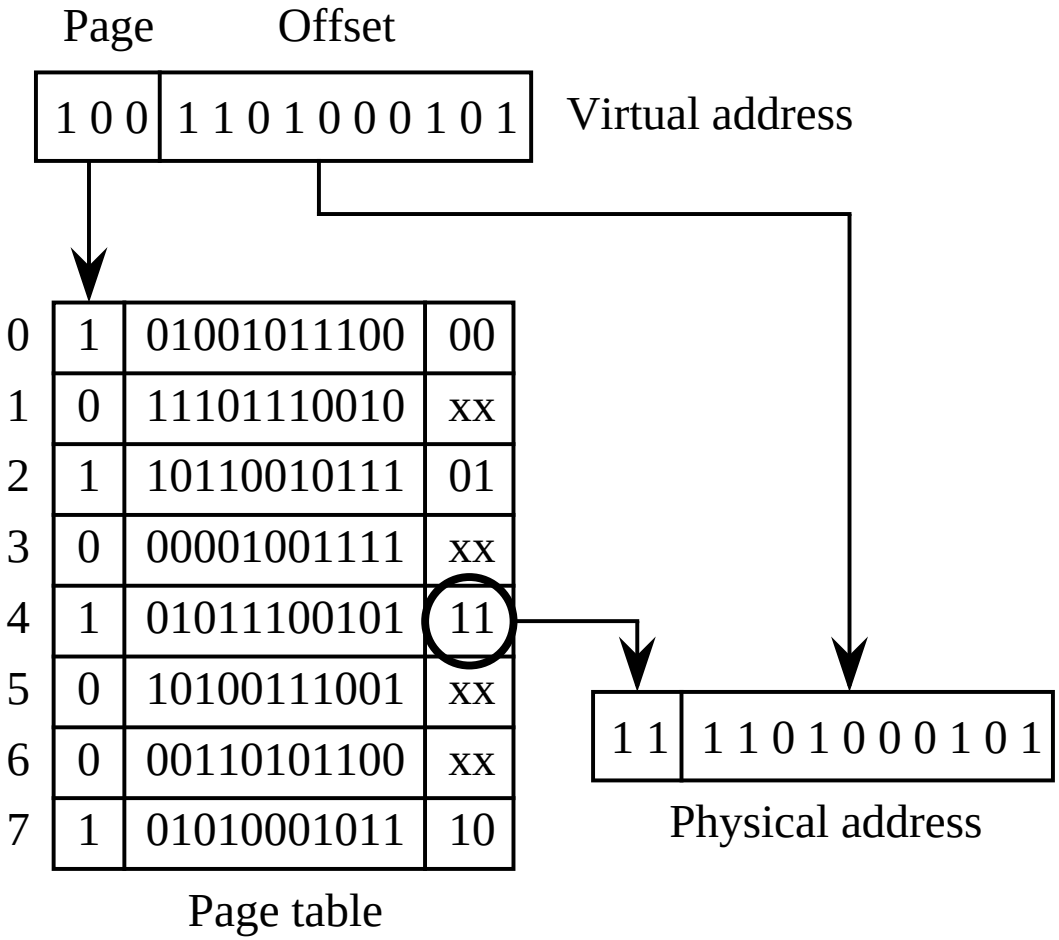
Present bit:
0: Page is not in physical memory
1: Page is in physical memory



Page #	Present bit	Disk address	Page frame
0	1	01001011100	00
1	0	11101110010	xx
2	1	10110010111	01
3	0	00001001111	xx
4	1	01011100101	11
5	0	10100111001	xx
6	0	00110101100	xx
7	1	01010001011	10

Using the Page Table

- **A virtual address is translated into a physical address:**



Using the Page Table (cont')

- The configuration of a page table changes as a program executes.
- Initially, the page table is empty. In the final configuration, four pages are in physical memory.

0	0	01001011100	xx
1	1	11101110010	00
2	0	10110010111	xx
3	0	00001001111	xx
4	0	01011100101	xx
5	0	10100111001	xx
6	0	00110101100	xx
7	0	01010001011	xx

After fault on page #1

0	0	01001011100	xx
1	1	11101110010	00
2	1	10110010111	01
3	0	00001001111	xx
4	0	01011100101	xx
5	0	10100111001	xx
6	0	00110101100	xx
7	0	01010001011	xx

After fault on page #2

0	0	01001011100	xx
1	1	11101110010	00
2	1	10110010111	01
3	1	00001001111	10
4	0	01011100101	xx
5	0	10100111001	xx
6	0	00110101100	xx
7	0	01010001011	xx

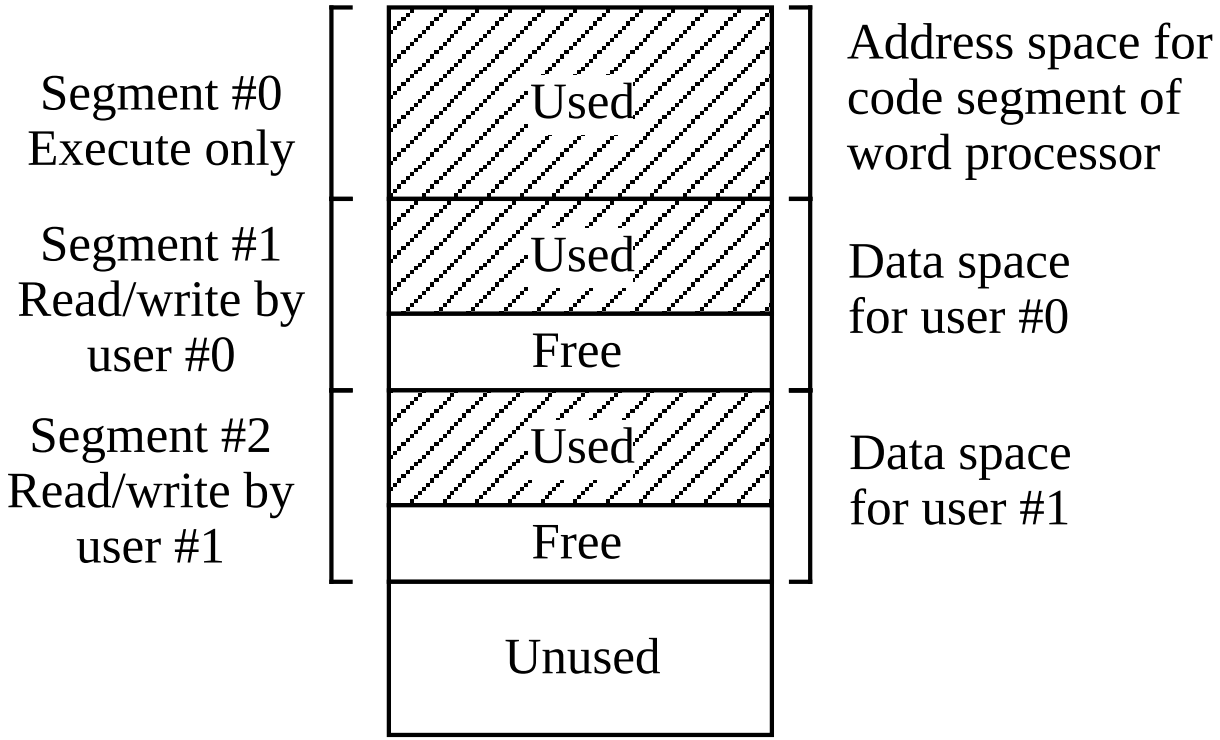
After fault on page #3

0	0	01001011100	xx
1	0	11101110010	xx
2	1	10110010111	01
3	1	00001001111	10
4	1	01011100101	11
5	1	10100111001	00
6	0	00110101100	xx
7	0	01010001011	xx

Final

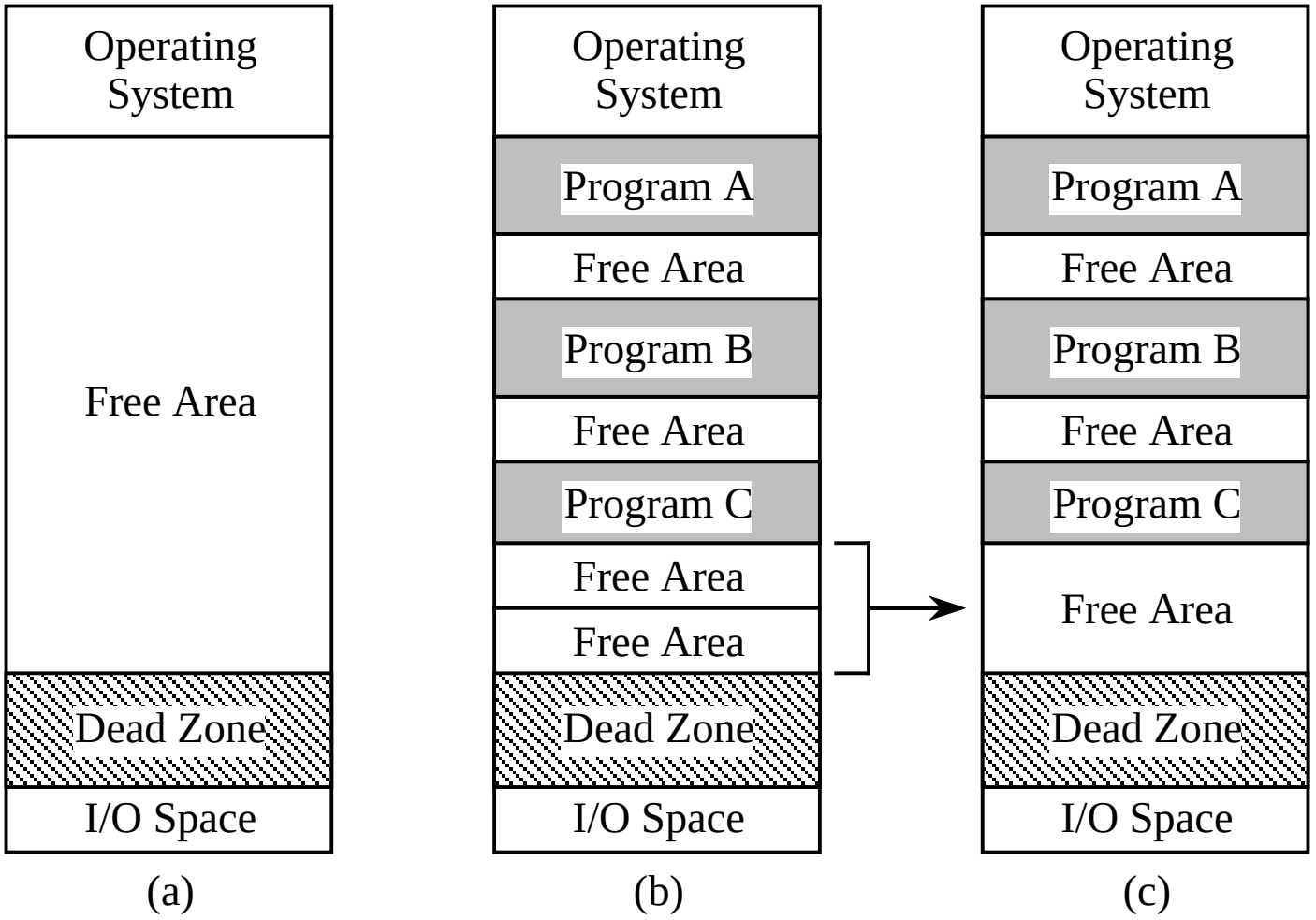
Segmentation

- **A segmented memory allows two users to share the same word processor code, with different data spaces:**



Fragmentation

- **(a) Free area of memory after initial-ization; (b) after frag-mentation; (c) after coa-lescing.**



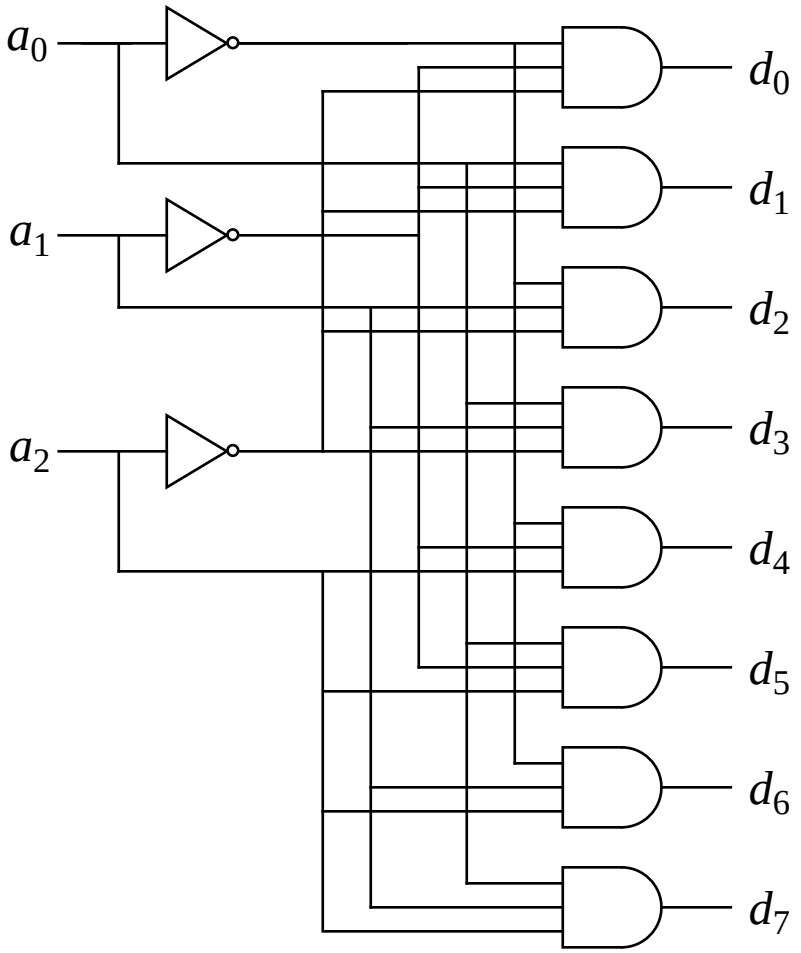
Translation Lookaside Buffer

- **An example TLB holds 8 entries for a system with 32 virtual pages and 16 page frames.**

Valid	Virtual page number	Physical page number
1	0 1 0 0 1	1 1 0 0
1	1 0 1 1 1	1 0 0 1
0	- - - - -	- - - -
0	- - - - -	- - - -
1	0 1 1 1 0	0 0 0 0
0	- - - - -	- - - -
1	0 0 1 1 0	0 1 1 1
0	- - - - -	- - - -

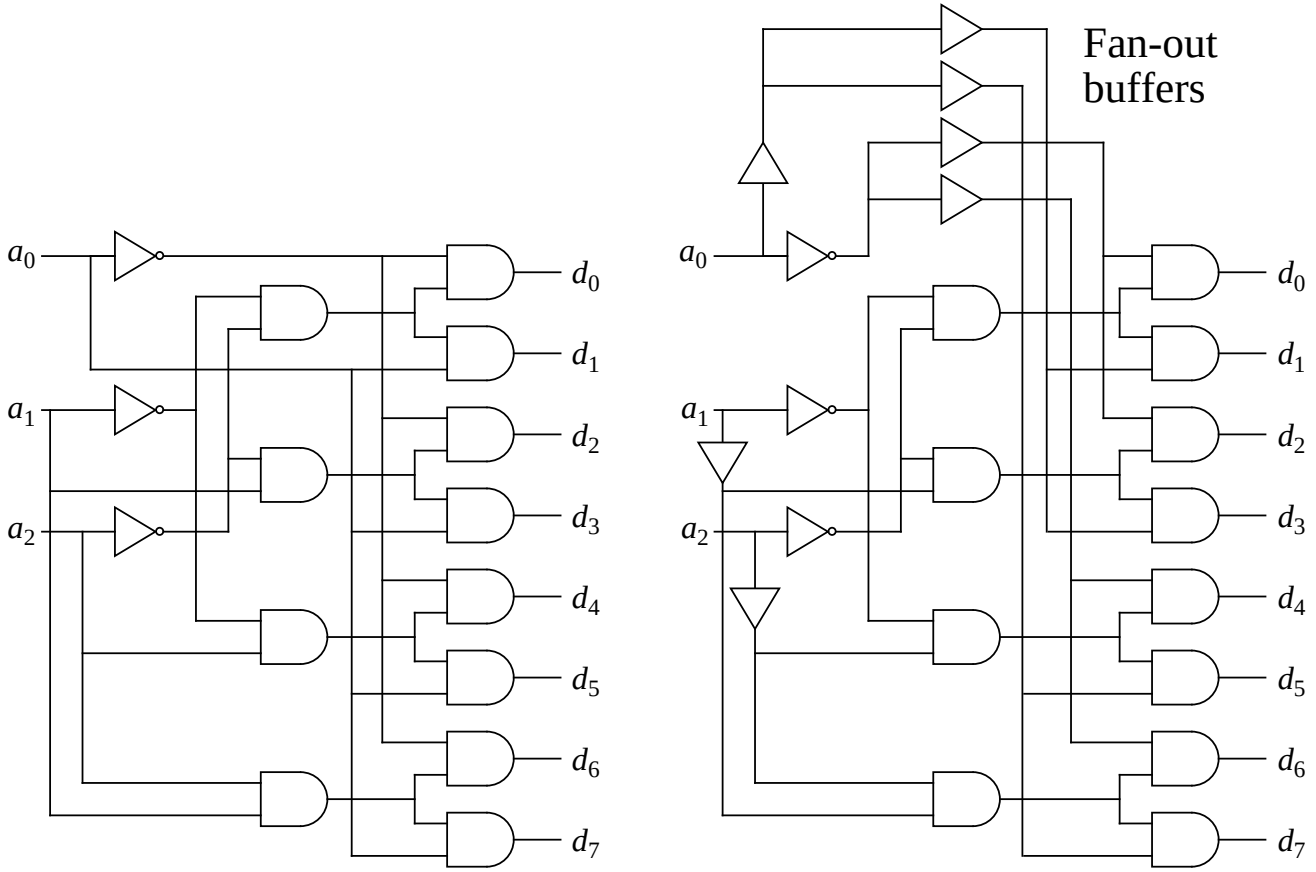
3-Variable Decoder

- **A conventional decoder is not extensible to large sizes because each address line drives twice as many logic gates for each added address line.**



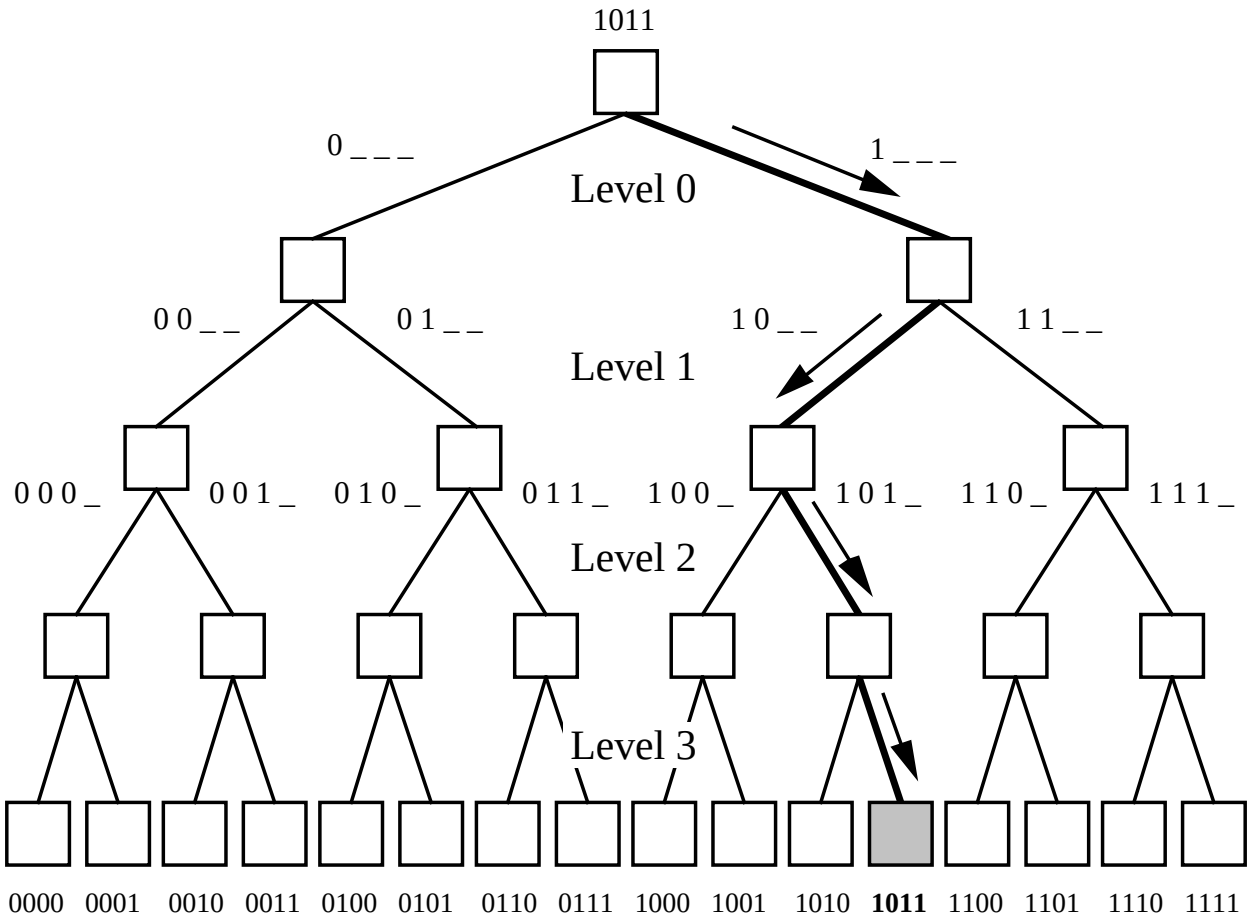
Tree Decoder - 3 Variables

- **A tree decoder is more easily extended to large sizes because fan-in and fan-out are managed by adding deeper levels.**



Tree Decoding – One Level at a Time

- **A decoding tree for a 16-word random access memory:**



Content Addressable Memory – Addressing

- **Relationships between random access memory and content addressable memory:**

Address	Value
0000A000	0F0F0000
0000A004	186734F1
0000A008	0F000000
0000A00C	FE681022
0000A010	3152467C
0000A014	C3450917
0000A018	00392B11
0000A01C	10034561

32 bits 32 bits

Random access memory

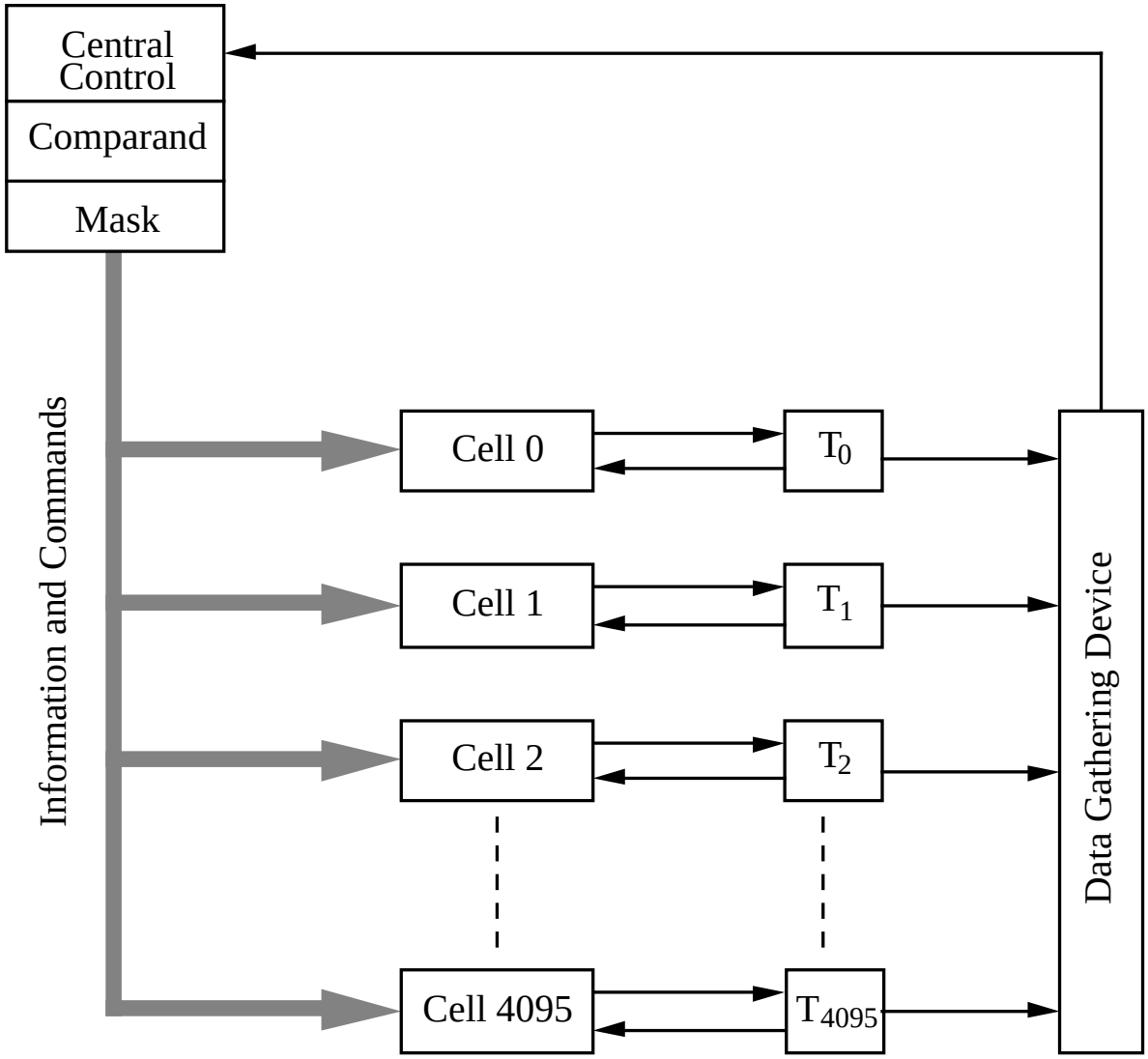
Field1	Field2	Field3
000	A	9E
011	0	F0
149	7	01
091	4	00
000	E	FE
749	C	6E
000	0	50
575	1	84

12 bits 4 bits 8 bits

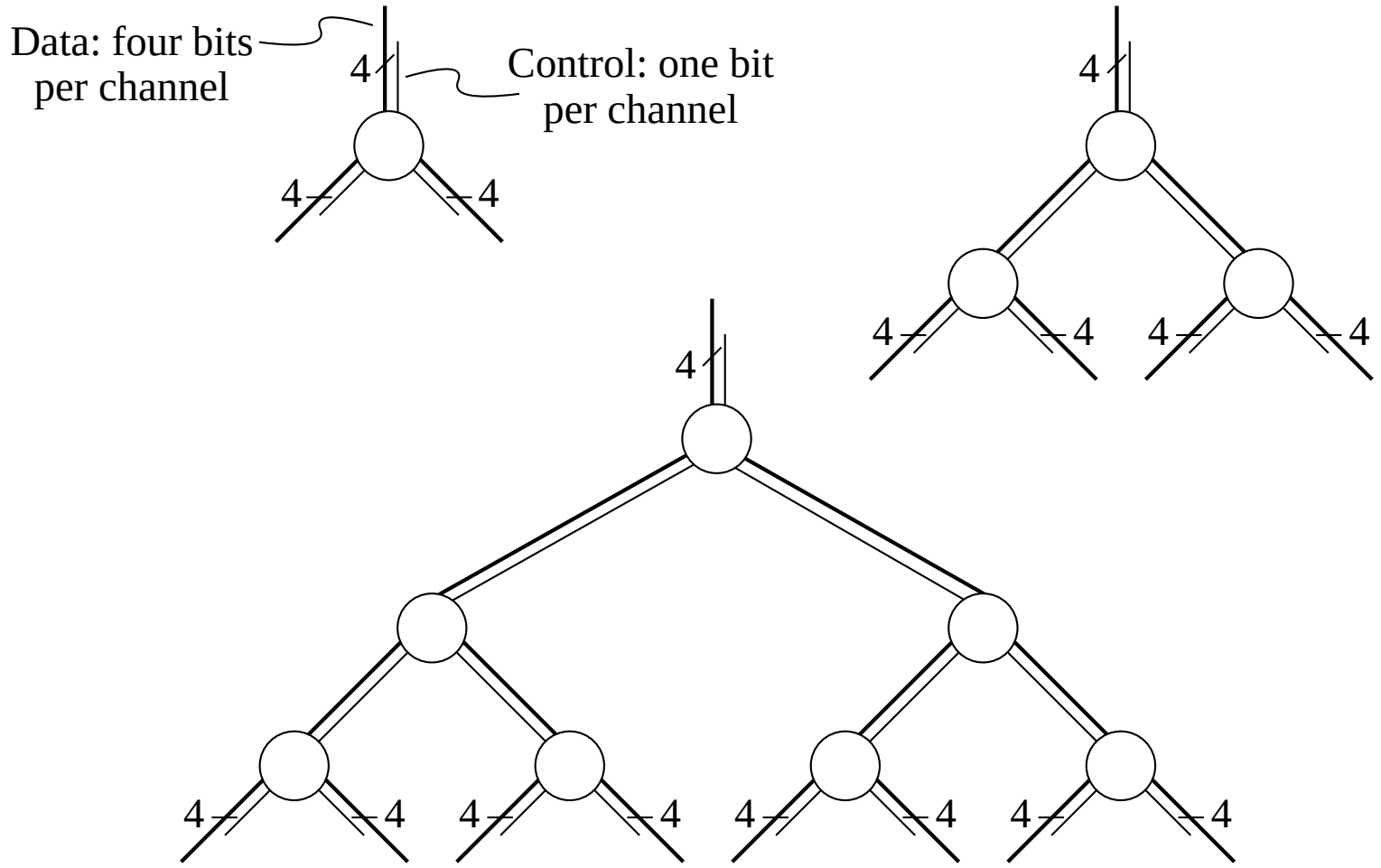
Content addressable memory

Overview of CAM

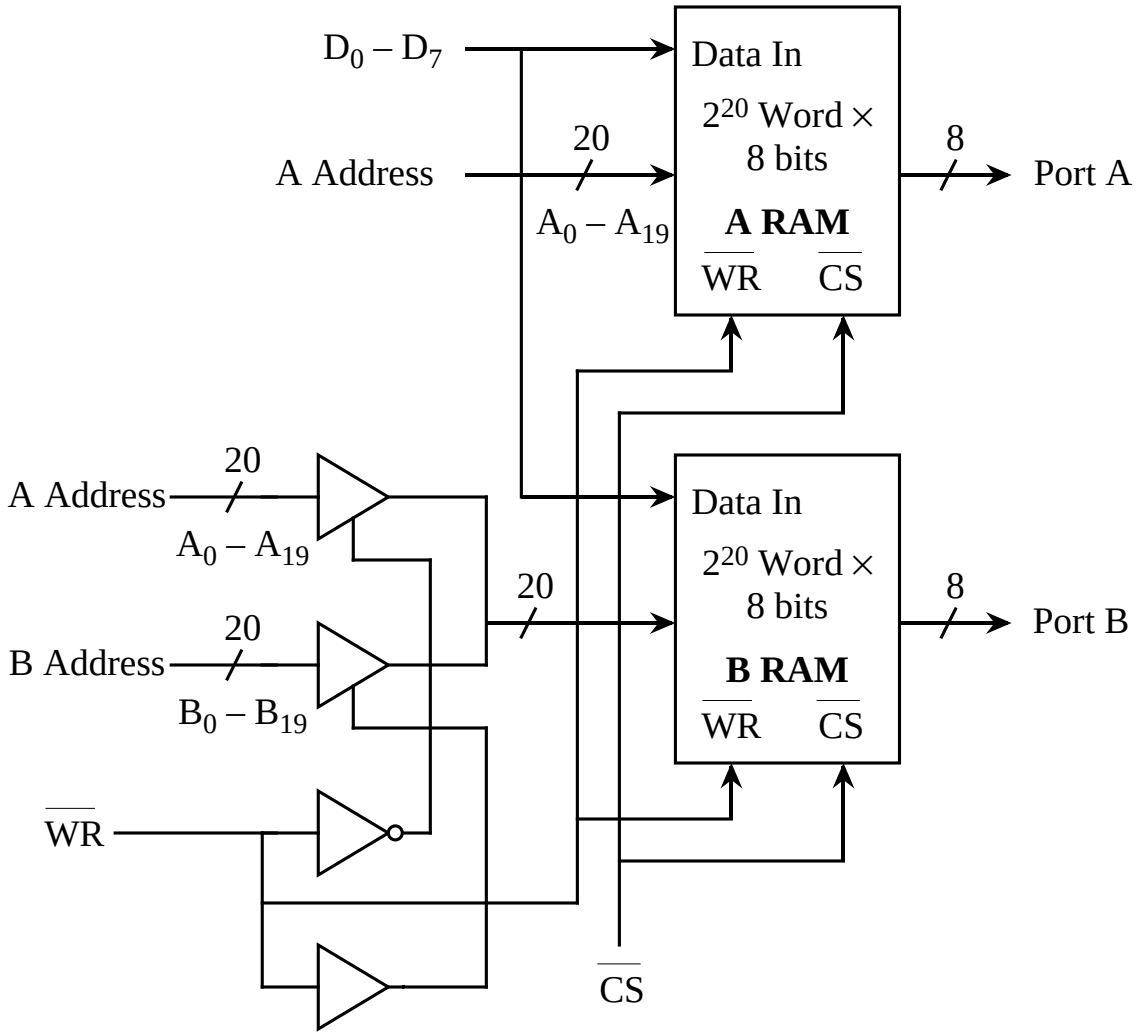
- **Source: (Foster, C. C., *Content Addressable Parallel Processors*, Van Nostrand Reinhold Company, 1976.)**



Addressing Subtrees for a CAM

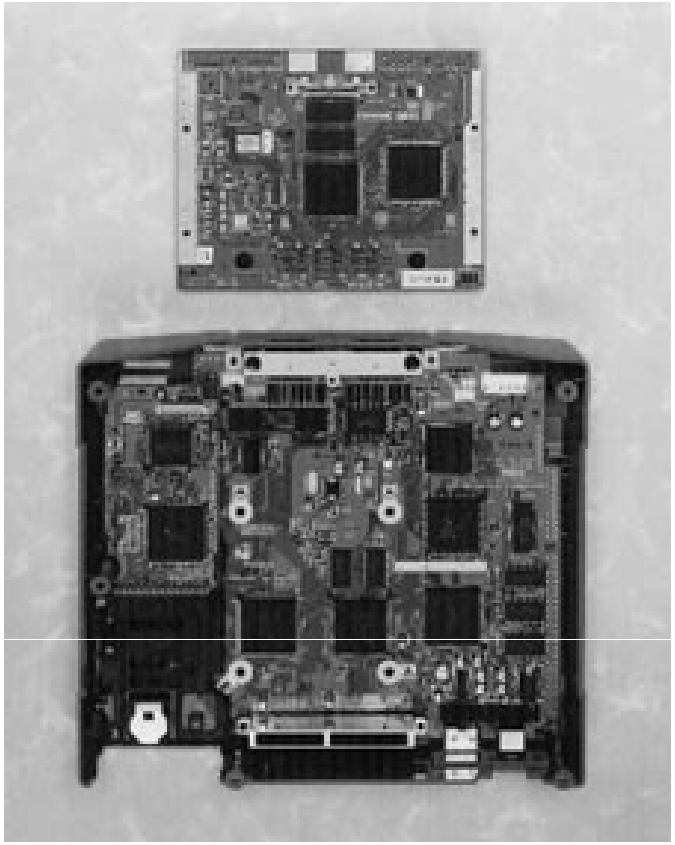


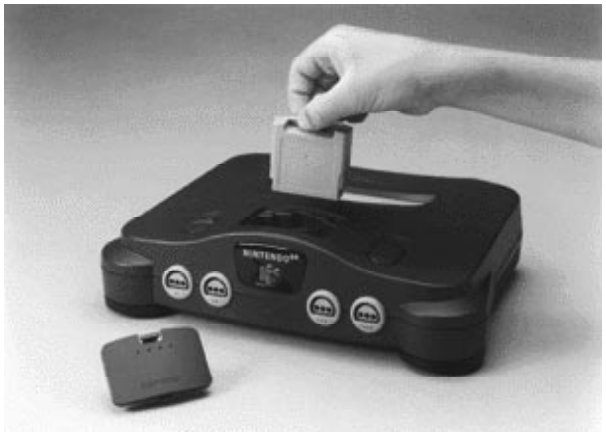
Block Diagram of Dual-Read RAM



Rambus Memory

- **Rambus technology on the Nintendo 64 motherboard (top left and bottom right) enables cost savings over the conventional Sega Saturn motherboard design (bottom left). (Photo source: Rambus, Inc.)**





The Intel Pentium Memory System