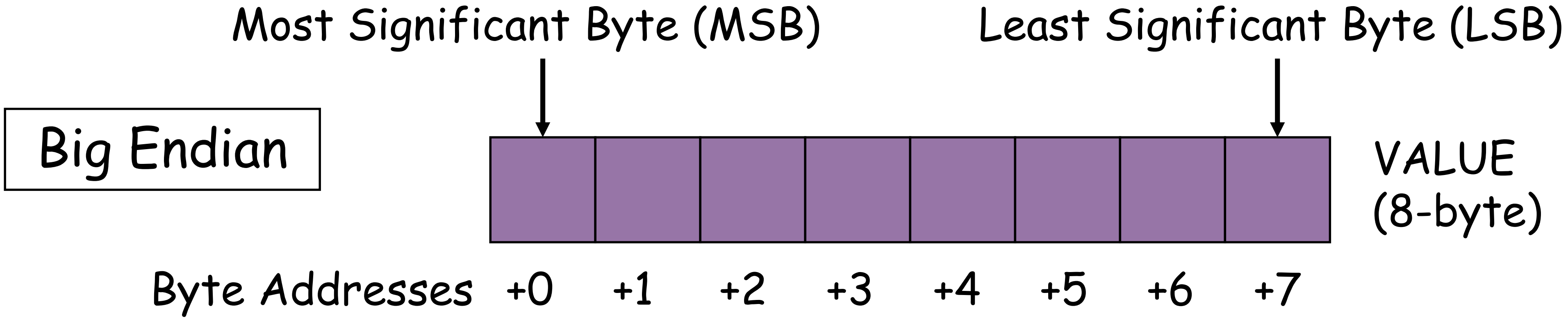
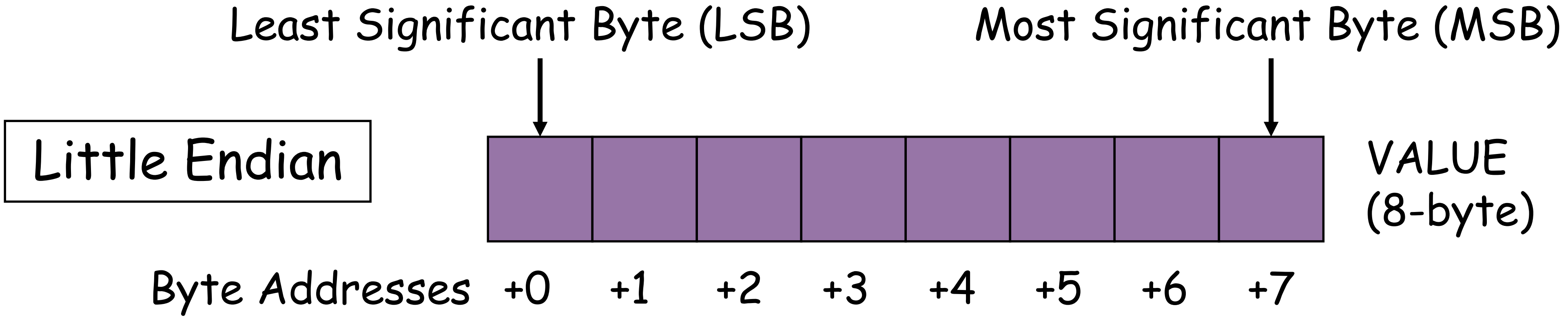
Byte Ordering of **Multibyte** Data Items

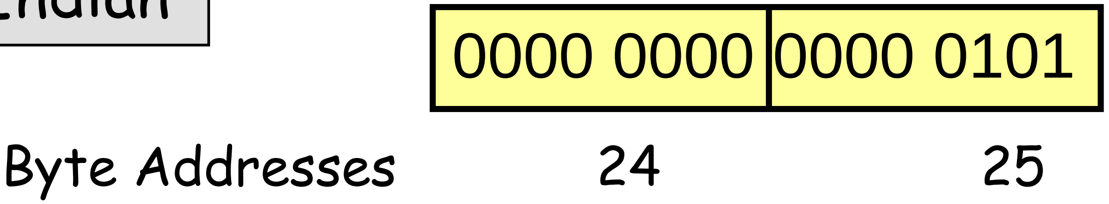




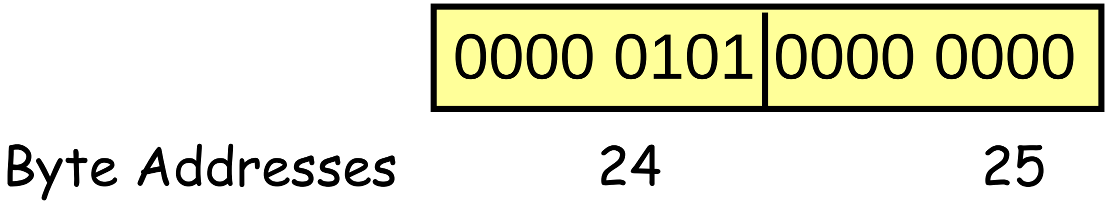
Example 1: 16-bit integer (View 1)

- 16-bit (2's Complement) integer 5 stored at memory address 24.

Big Endian



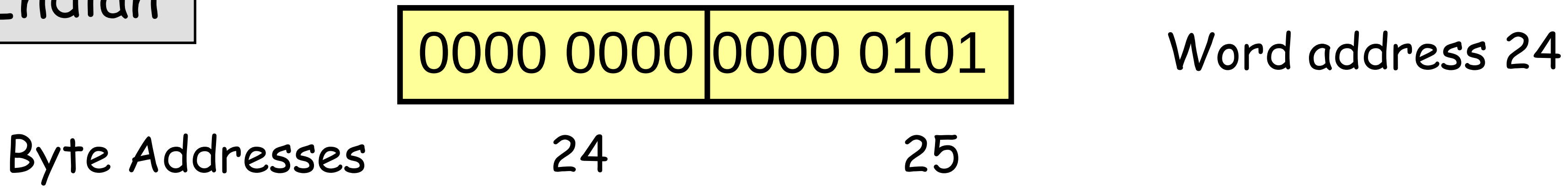
Little Endian



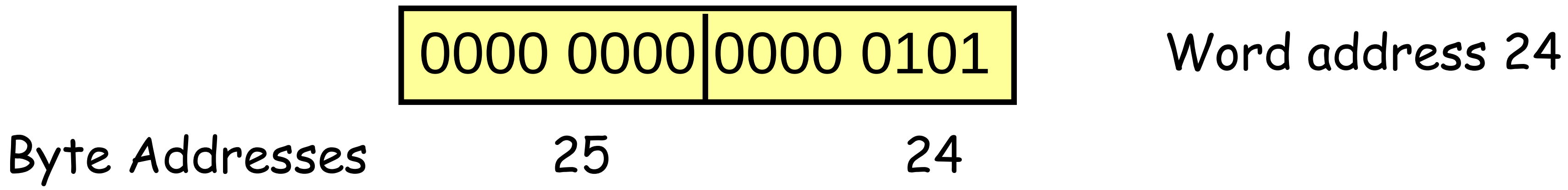
Example 1: 16-bit integer (View 2)

- 16-bit (2's Complement) integer 5 stored at memory address 24.

Big Endian

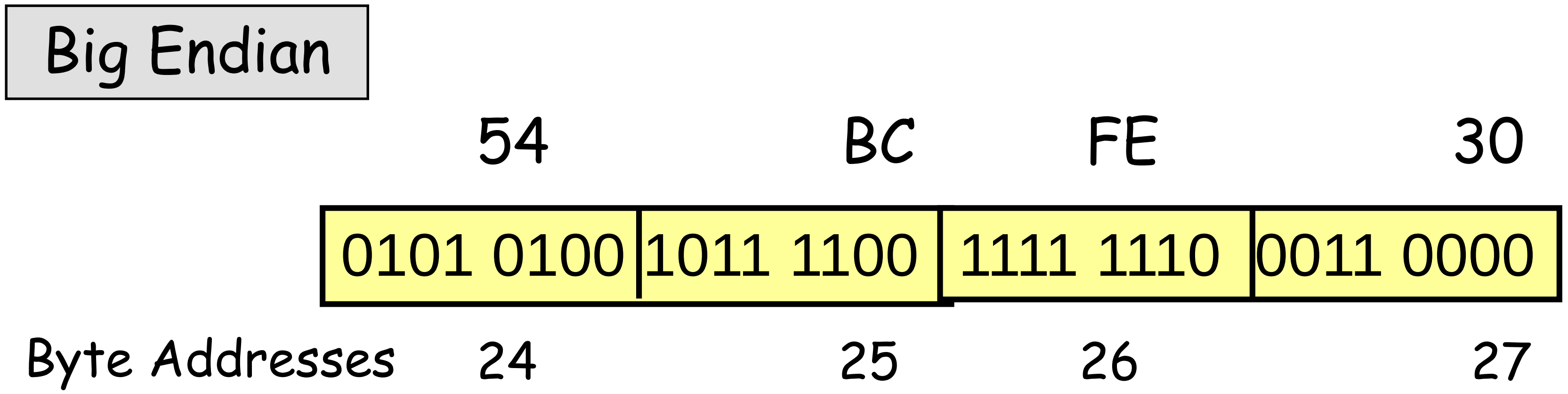


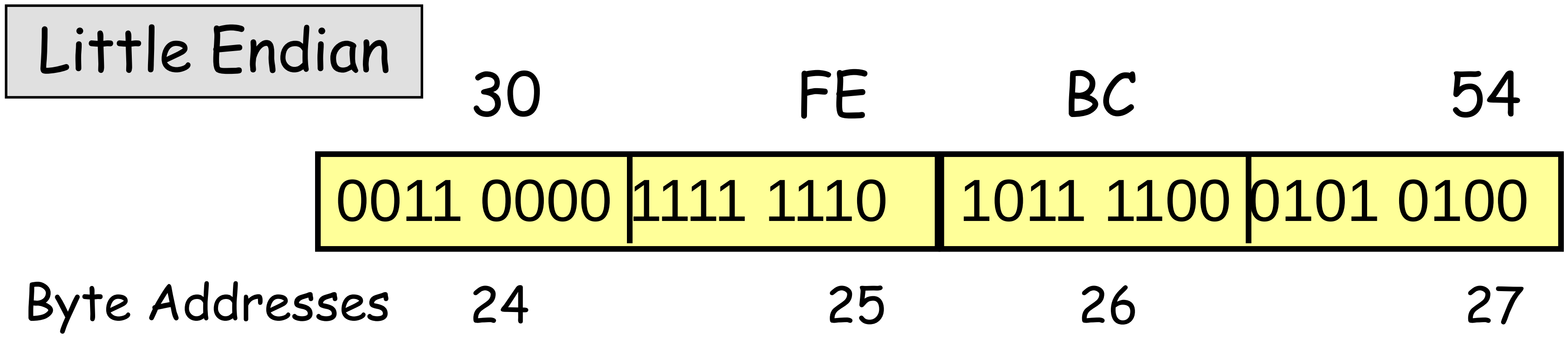
Little Endian



Example 2: 32-bit value (View 1)

- 32-bit hex value 54 BC FE 30 stored at memory address 24.

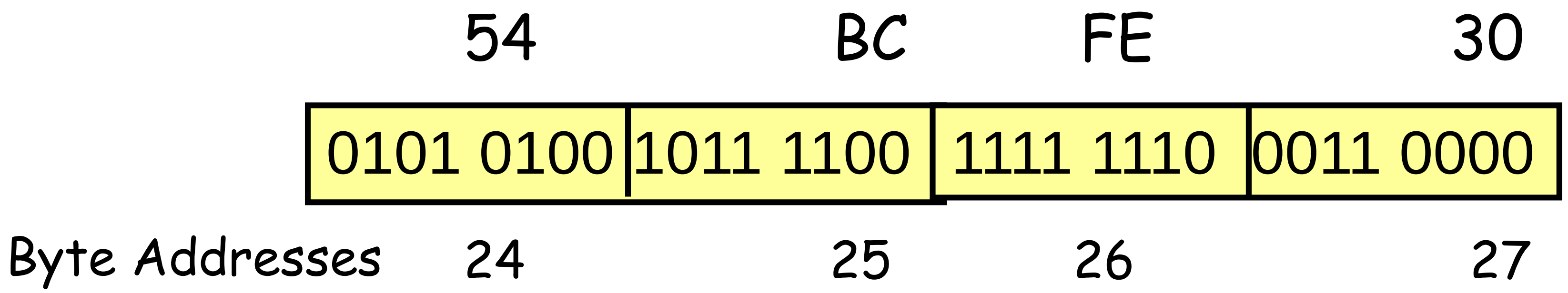




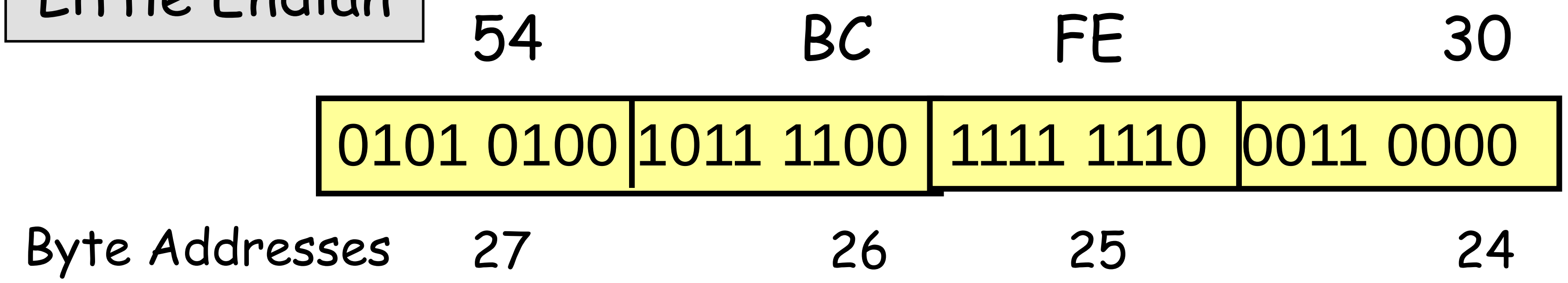
Example 2: 32-bit value (View 2)

- 32-bit hex value 54 BC FE 30 stored at memory address 24.

Big Endian



Little Endian



Example 3: ASCII string (View 1)

- String "JIM BLOGGS" stored at memory address 24
- Treat a string as an array of (ASCII) bytes. Each byte is considered individually.

Big Endian

	J	I	M		B	L	O	G	G	S
Byte Addresses	24	25	26	27	28	29	30	31	32	33

Little Endian

	J	I	M		B	L	O	G	G	S
Byte Addresses	24	25	26	27	28	29	30	31	32	33

Example 3: ASCII string (View 2)

- String “JIM BLOGGS” stored at memory address 24
- Treat a string as an array of (ASCII) bytes. Each byte is considered individually.

Big Endian

	J	I	M		B	L	O	G	G	S
Byte Addresses	24	25	26	27	28	29	30	31	32	33

Little Endian

	S	G	G	O	L	B		M	I	J
Byte Addresses	33	32	31	30	29	28	27	26	25	24

Data Transfer

- How do we **transfer a multi-byte value** (e.g. a 32-bit two’s complement integer) from a Big-Endian memory to a Little-Endian memory?

- How do we transfer an ASCII **string** value (e.g. “JIM BLOGGS”) from a Big-Endian memory to a Little-Endian memory?

- How do we transfer an **object** which holds both types of values above?

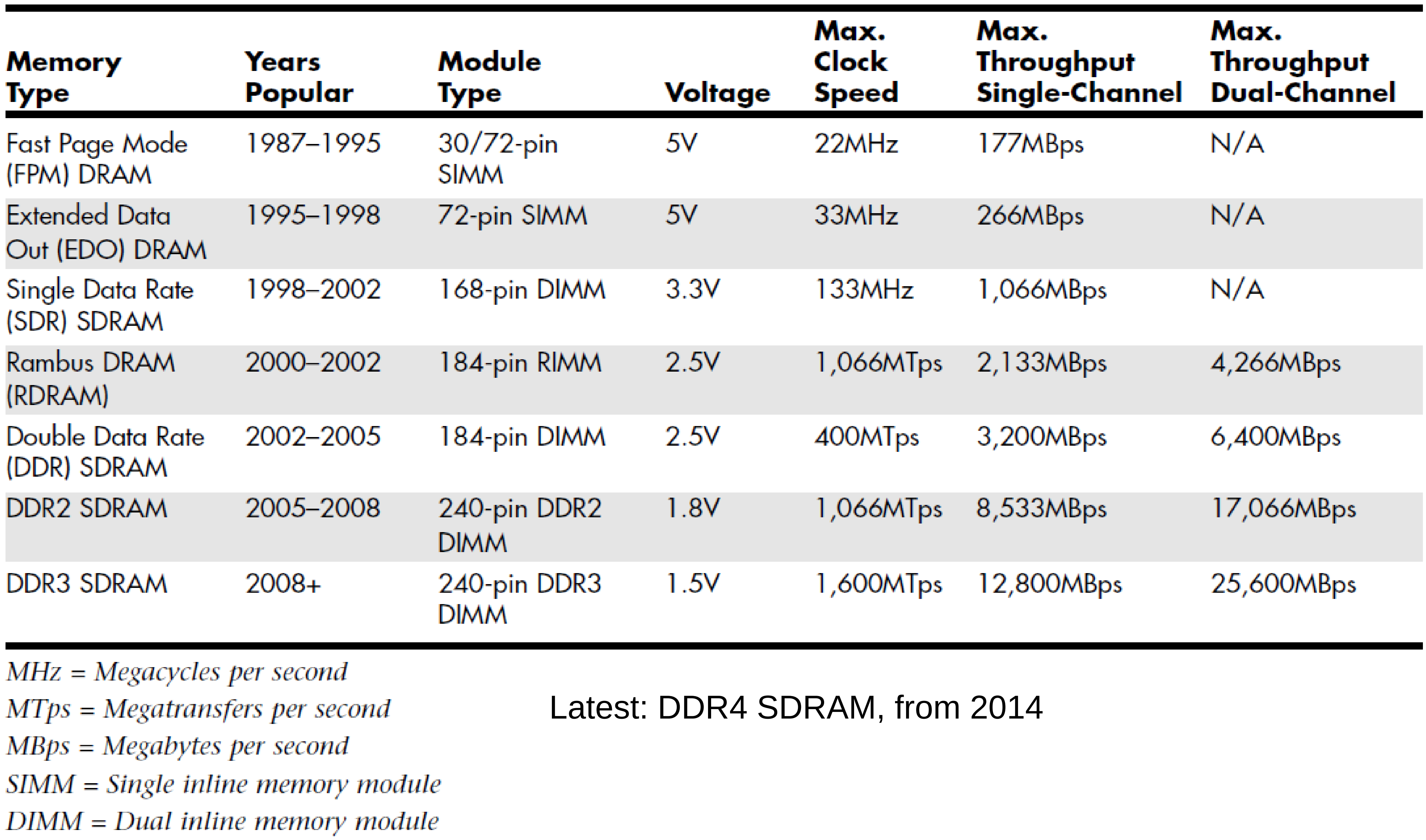
(Un) Aligned Memory Accesses

Main Memory (Big Endian)

0	0110 1101	1010 1101
2	**1010 1001**	**1010 0001**
4	0000 0000	0000 0000
6	1111 1111	**1111 0000**
8	**0010 0001**	0000 0000
10	1001 1010	1010 0010
12	0000 0000	0000 0000
14	1111 1111	1111 1110

- The 16-bit hex value (A9A1) at address 2 is memory word **aligned**.
- The 16-bit hex value (F021) at address 7 is **unaligned**.
- Some architectures prohibit unaligned accesses. Why?
- How many memory accesses are required to read a 32-bit value from memory address 11?
- How many memory accesses are required to write a 32-bit value to memory address 11?

Types and Performance Levels of Memories



Memory Type	Years Popular	Module Type	Voltage	Max. Clock Speed	Max. Throughput Single-Channel	Max. Throughput Dual-Channel
Fast Page Mode (FPM) DRAM	1987–1995	30/72-pin SIMM	5V	22MHz	177MBps	N/A
Extended Data Out (EDO) DRAM	1995–1998	72-pin SIMM	5V	33MHz	266MBps	N/A
Single Data Rate (SDR) SDRAM	1998–2002	168-pin DIMM	3.3V	133MHz	1,066MBps	N/A
Rambus DRAM (RDRAM)	2000–2002	184-pin RIMM	2.5V	1,066MTps	2,133MBps	4,266MBps
Double Data Rate (DDR) SDRAM	2002–2005	184-pin DIMM	2.5V	400MTps	3,200MBps	6,400MBps
DDR2 SDRAM	2005–2008	240-pin DDR2 DIMM	1.8V	1,066MTps	8,533MBps	17,066MBps
DDR3 SDRAM	2008+	240-pin DDR3 DIMM	1.5V	1,600MTps	12,800MBps	25,600MBps

MHz = Megacycles per second

MTps = Megatransfers per second

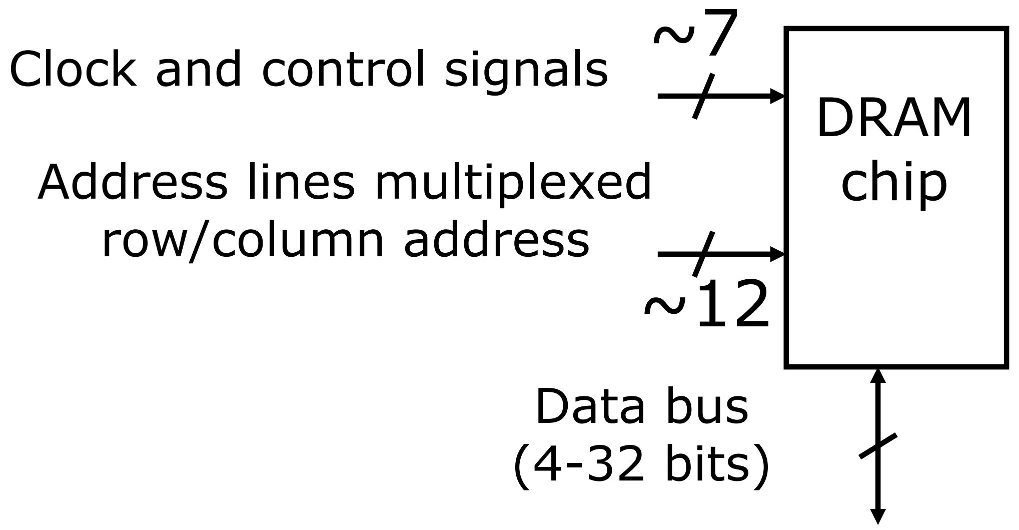
MBps = Megabytes per second

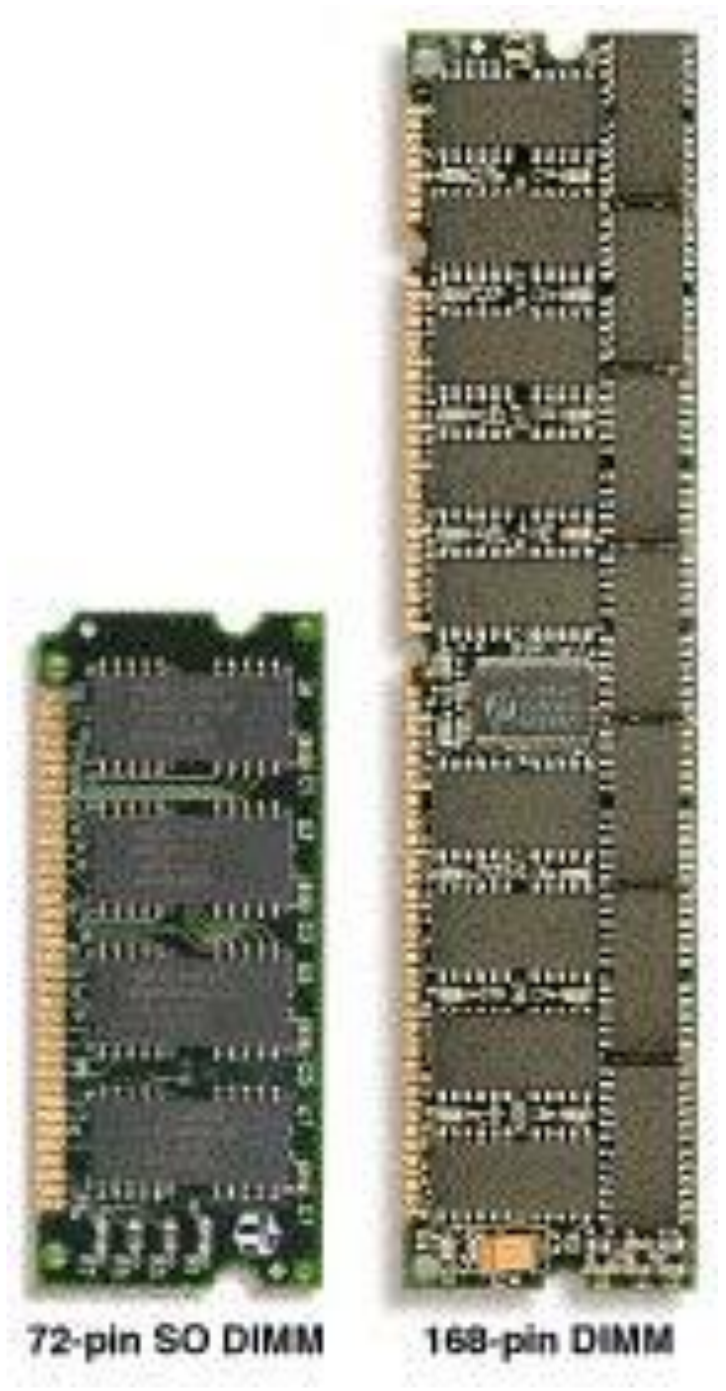
SIMM = Single inline memory module

DIMM = Dual inline memory module

Latest: DDR4 SDRAM, from 2014

Memory Modules and Chips


~7
Clock and control signals
Address lines multiplexed row/column address
~12
DRAM chip
Data bus (4-32 bits)

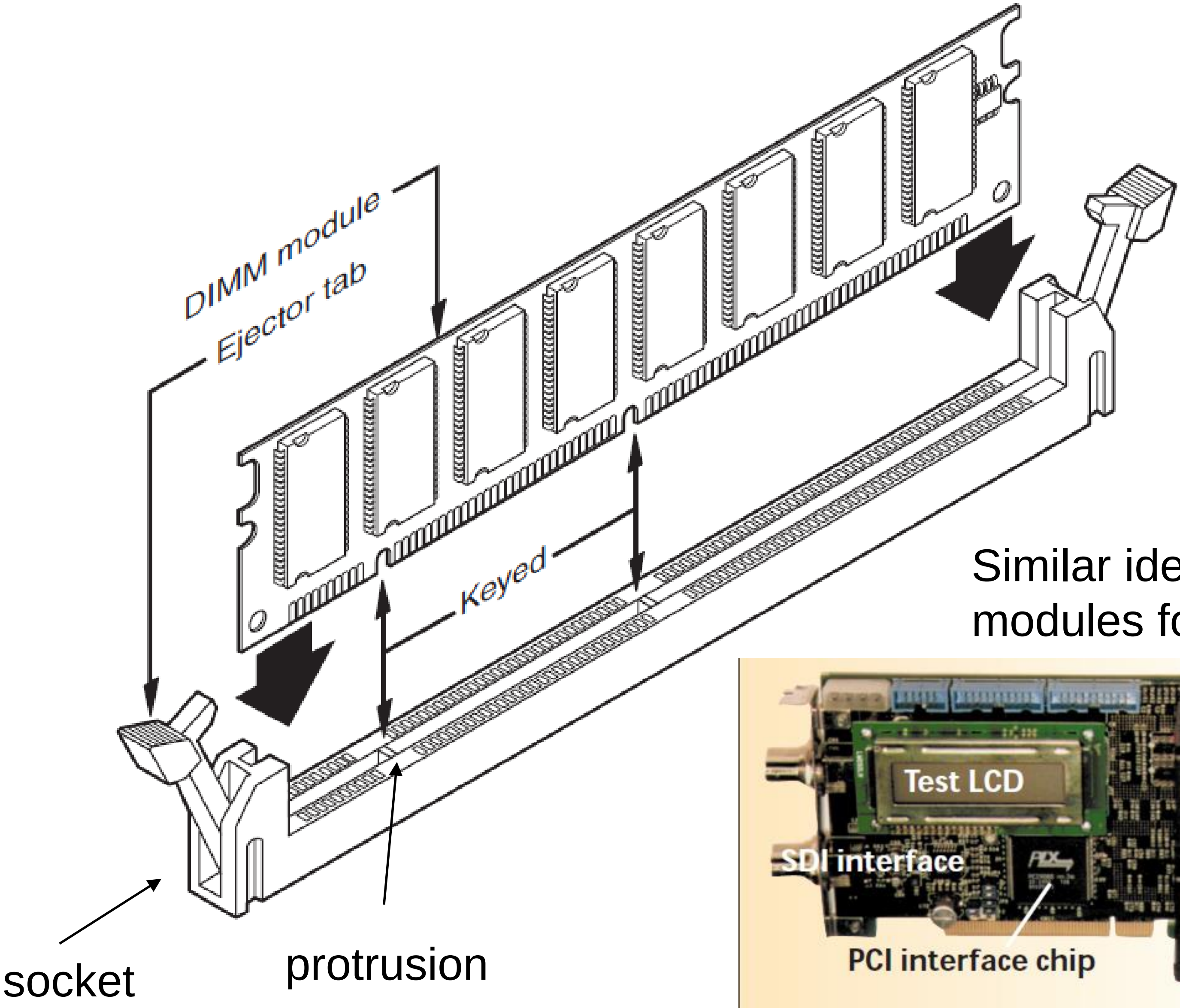


- DIMM: Dual Inline Memory Module
  - multiple chips
  - clock/control/address signals in parallel
  - may need buffers: drive signals to all chips
- data pins: work together to return wide word
- DDR4 SDRAM: read/write 8 consecutive words

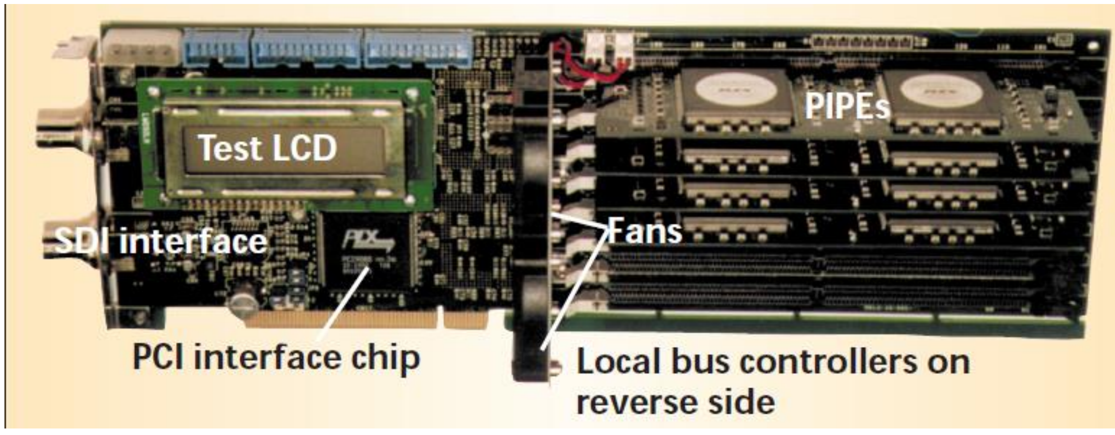
Source: UC Berkeley

SO = Small Outline

Inserting a DIMM Module

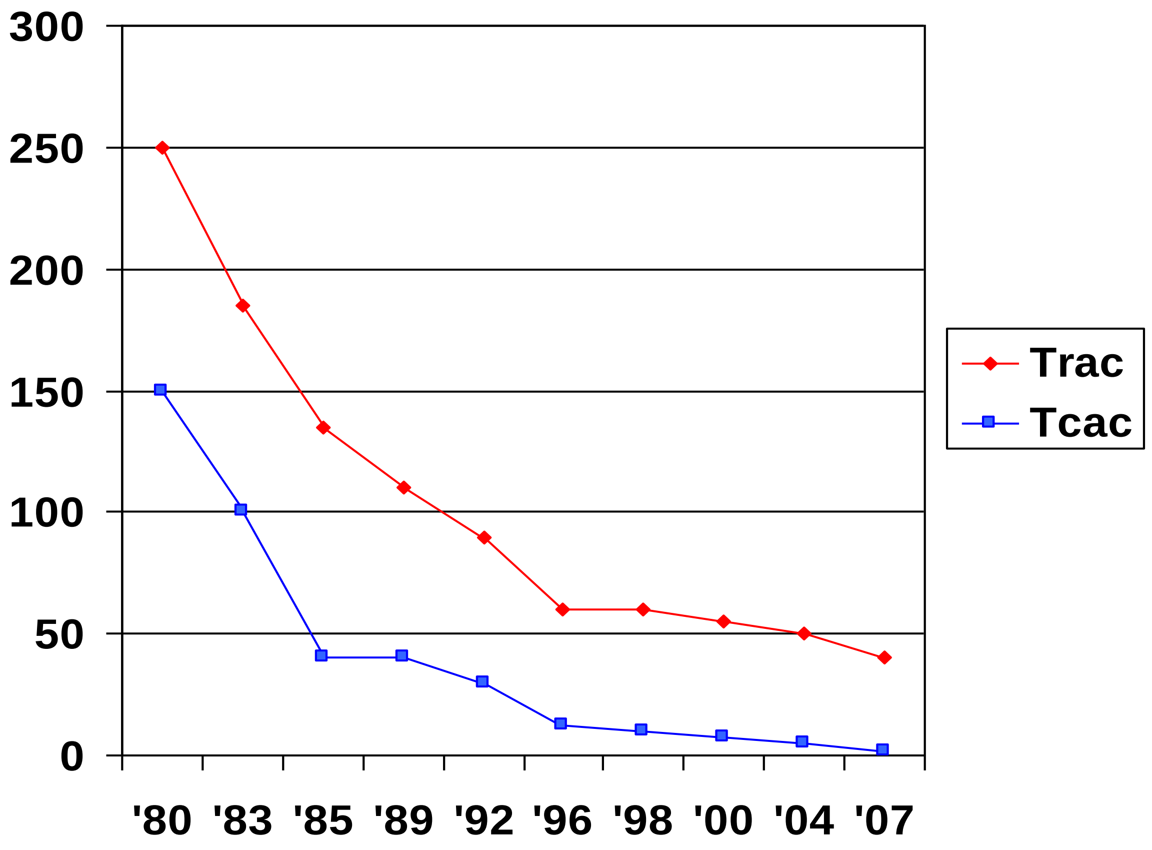


Similar idea used in FPGA PIPE modules for our SONIC system



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

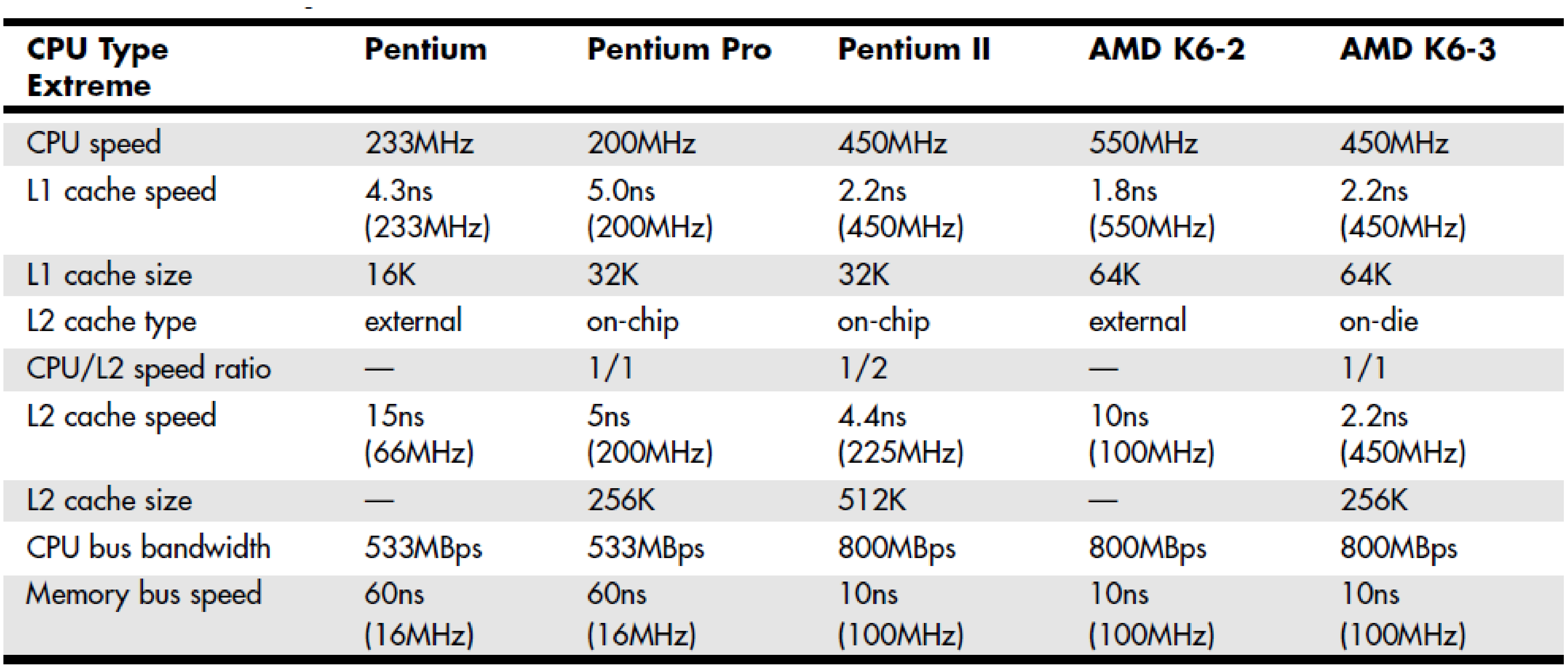


Trac: Total access time to a new row/column

Tcac: Column access time to existing row

Source: MKP

Evolution of memory hierarchy with CPUs



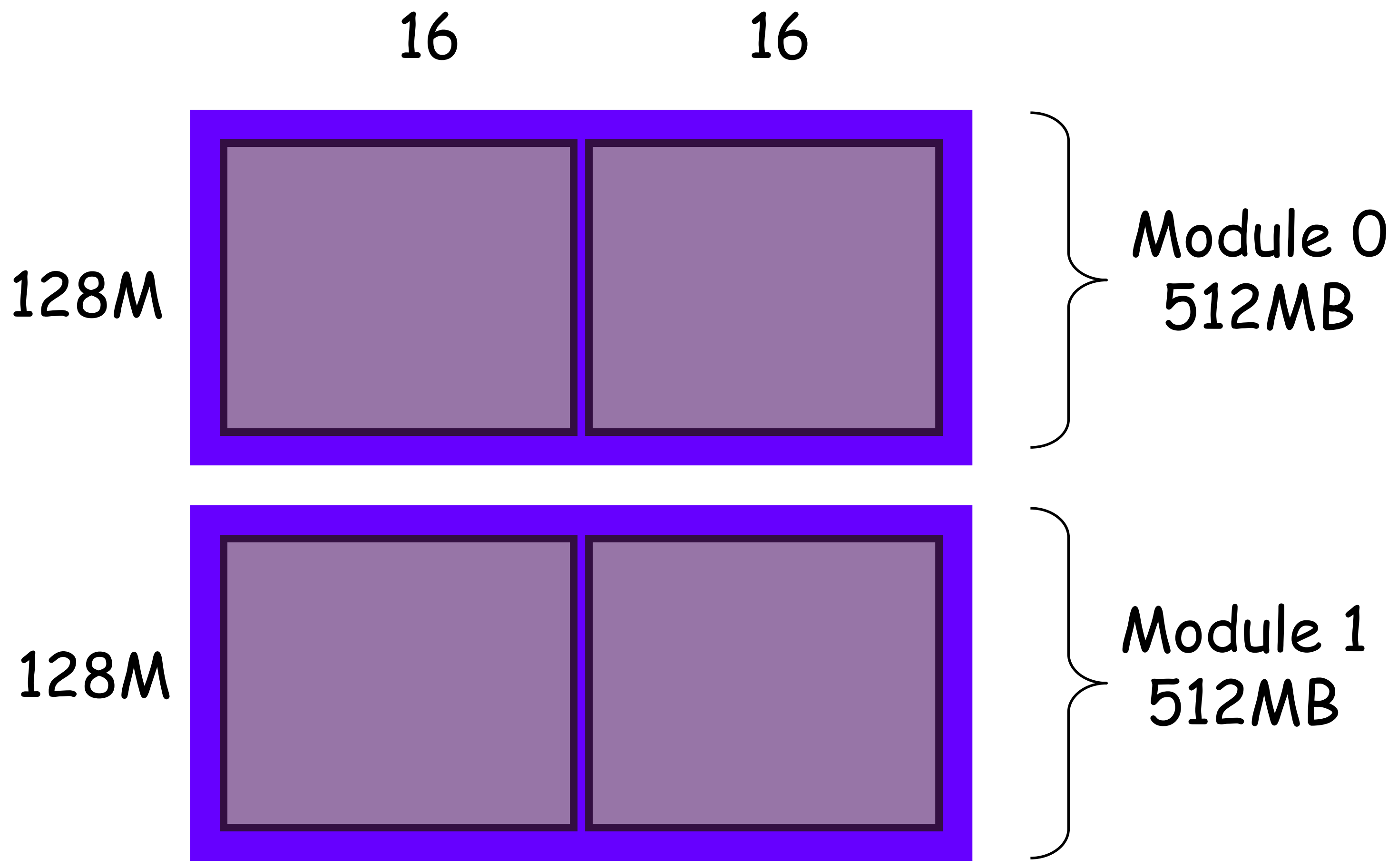
CPU Type Extreme	Pentium	Pentium Pro	Pentium II	AMD K6-2	AMD K6-3
CPU speed	233MHz	200MHz	450MHz	550MHz	450MHz
L1 cache speed	4.3ns (233MHz)	5.0ns (200MHz)	2.2ns (450MHz)	1.8ns (550MHz)	2.2ns (450MHz)
L1 cache size	16K	32K	32K	64K	64K
L2 cache type	external	on-chip	on-chip	external	on-die
CPU/L2 speed ratio	—	1/1	1/2	—	1/1
L2 cache speed	15ns (66MHz)	5ns (200MHz)	4.4ns (225MHz)	10ns (100MHz)	2.2ns (450MHz)
L2 cache size	—	256K	512K	—	256K
CPU bus bandwidth	533MBps	533MBps	800MBps	800MBps	800MBps
Memory bus speed	60ns (16MHz)	60ns (16MHz)	10ns (100MHz)	10ns (100MHz)	10ns (100MHz)

Evolution of memory hierarchy with CPUs

CPU Type Extreme	Athlon	Athlon XP	Pentium 4	Athlon 64 X2	Core 2 Duo	Core 2
CPU speed	1.4GHz	2.2GHz	3.8GHz	3GHz	2.66GHz	2.93GHz
L1 cache speed	0.71ns (1.4GHz)	0.45ns (2.2GHz)	0.26ns (3.8GHz)	0.33ns (3GHz)	0.37ns (2.66GHz)	0.34ns (2.93GHz)
L1 cache size	128K	128K	20K	256K	64K	64K
L2 cache type	on-die	on-die	on-die	on-die	On-die	on-die
CPU/L2 speed ratio	1/1	1/1	1/1	1/1	1/1	1/1
L2 cache speed	0.71ns (1.4GHz)	0.45ns (2.2GHz)	0.26ns (3.8GHz)	0.33ns (3GHz)	0.37ns (2.66GHz)	0.34ns (2.93GHz)
L2 cache size	256K	512K	2M	2M	4M[1]	8M[2]
CPU bus bandwidth	2,133MBps	3,200MBps	6,400MBps	4,000MBps	8,533MBps	8,533MBps
Memory bus speed	3.8ns (266MHz)	2.5ns (400MHz)	1.25ns (800MHz)	2.5ns (400MHz)	0.94ns (1066MHz)	0.94ns (1066MHz)

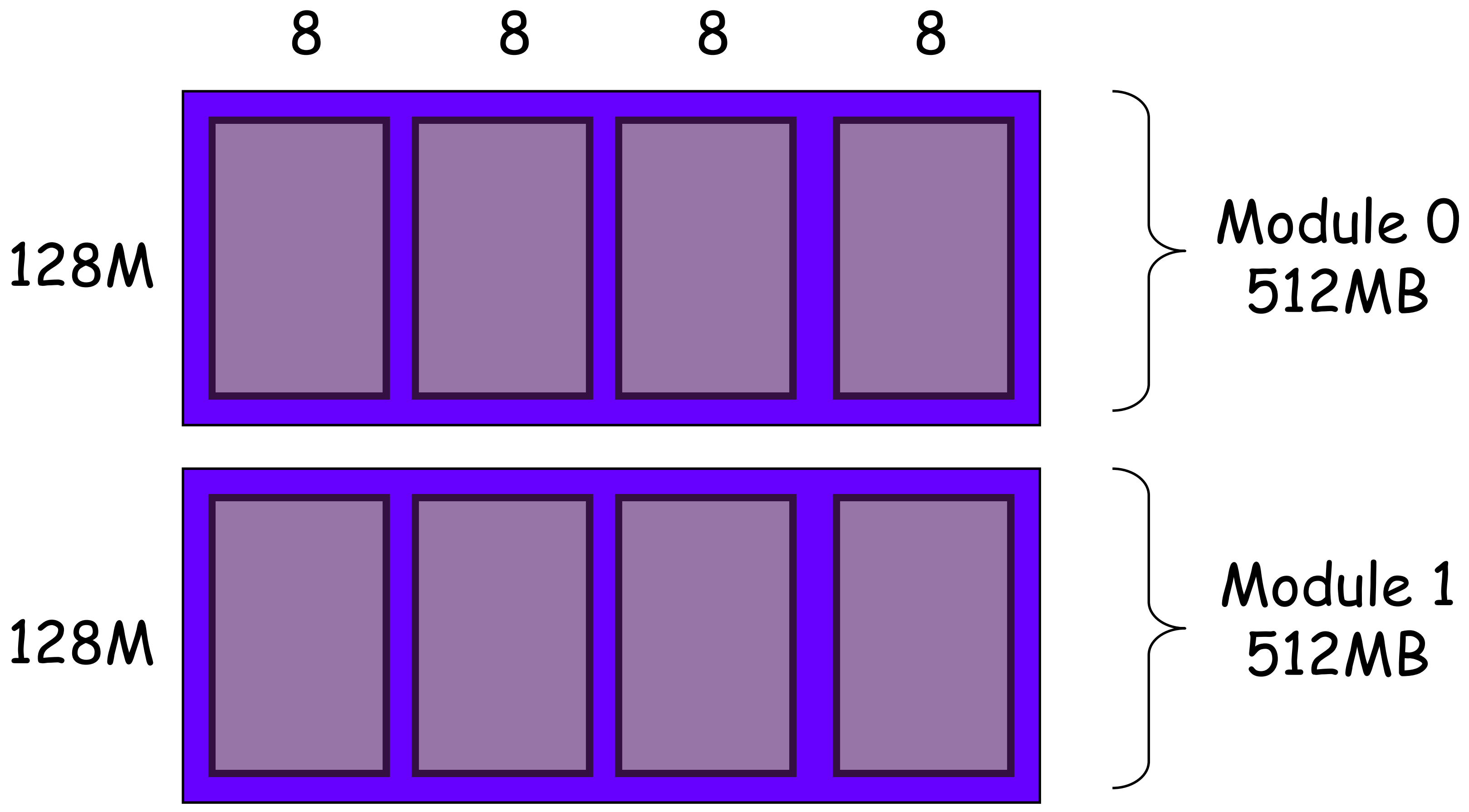
1GB (256Mx32bit) Memory

- Two 512MB memory modules. Each module has two 128M x 16-bit RAM Chips



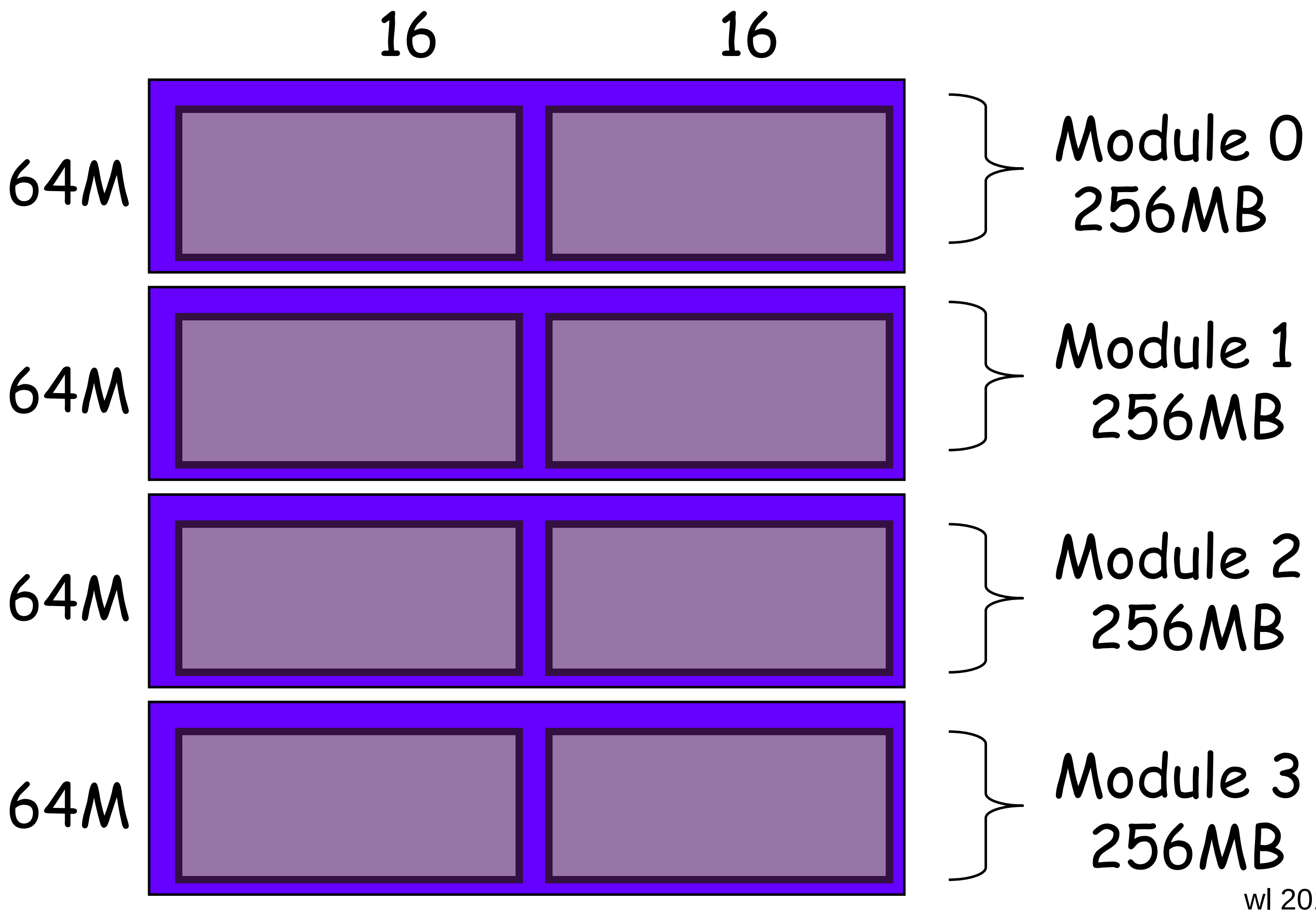
1GB (256Mx32bit) Memory

- Two 512MB memory modules. Each module has four 128M x 8-bit RAM Chips



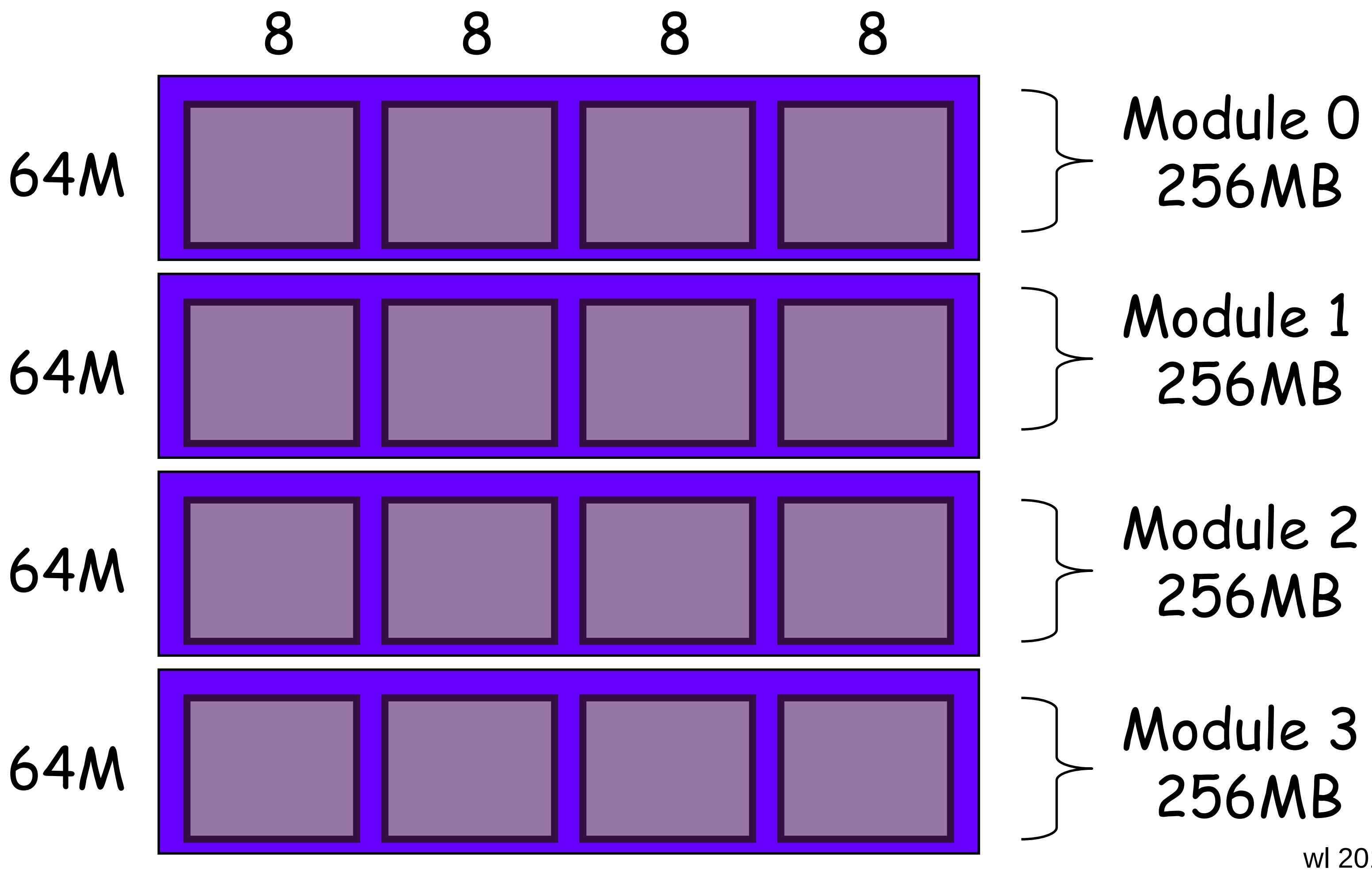
1GB (256Mx32bit) Memory

- Four 256MB memory modules. Each module has two 64M x 16-bit RAM Chips



1GB (256Mx32bit) Memory

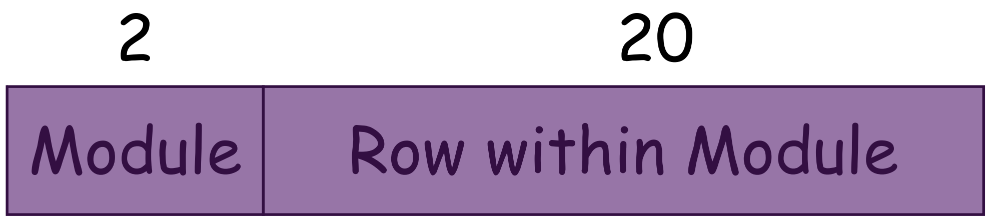
- Four 256MB memory modules. Each module has four 64M x 8-bit RAM Chips



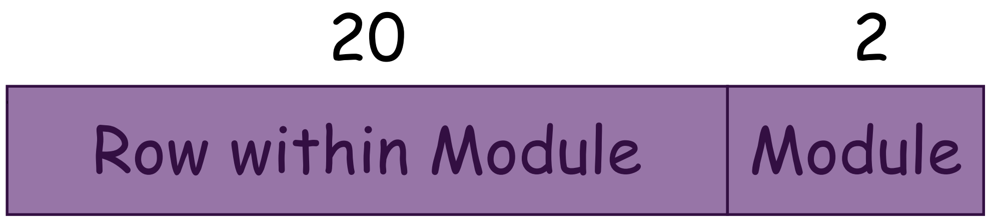
Memory Interleaving

- **Example**. Memory=4M words. Word Addressed. Each word = 32-bits. Built with 4 x 1Mx32-bit memory modules.
- For 4M words we need 22 bits for an address.

  22 bits = 2 bits (to select Modules) + 20 bits (to select row within Module)



High-Order Interleave



Low-Order Interleave

High-Order Interleave

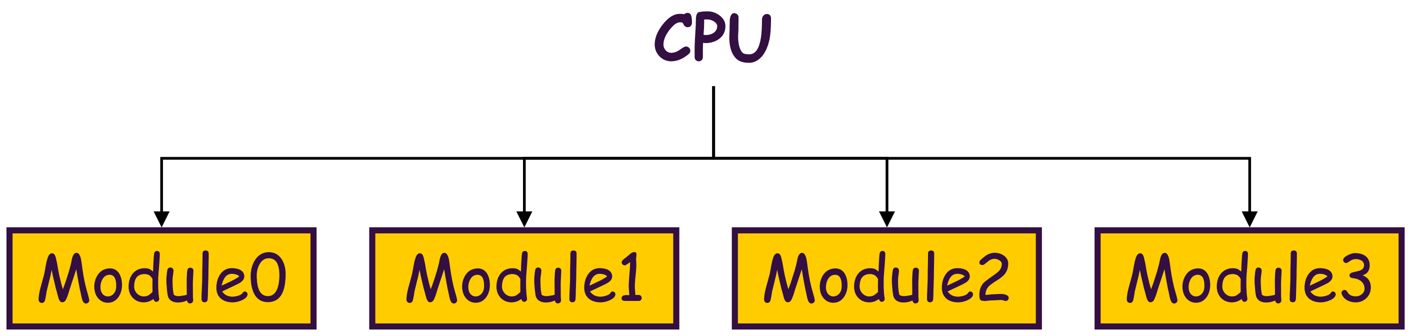
Address Decimal	Address Binary							
0	00	0000	0000	0000	0000	0000	Module=0	Row=0
1	00	0000	0000	0000	0000	0001	Module=0	Row=1
2	00	0000	0000	0000	0000	0010	Module=0	Row=2
3	00	0000	0000	0000	0000	0011	Module=0	Row=3
4	00	0000	0000	0000	0000	0100	Module=0	Row=4
5	00	0000	0000	0000	0000	0101	Module=0	Row=5
...								
2^20-1	00	1111	1111	1111	1111	1111	Module=0	Row=2^20-1
2^20	01	0000	0000	0000	0000	0000	Module=1	Row=0
2^20+1	01	0000	0000	0000	0000	0001	Module=1	Row=1

Low-Order Interleave

Address Decimal	Address Binary		
0	00 0000 0000 0000 0000 0000	Module=0	Row=0
1	00 0000 0000 0000 0000 0001	Module=1	Row=0
2	00 0000 0000 0000 0000 0010	Module=2	Row=0
3	00 0000 0000 0000 0000 0011	Module=3	Row=0
4	00 0000 0000 0000 0000 0100	Module=0	Row=1
5	00 0000 0000 0000 0000 0101	Module=1	Row=1
...			
2^20-1	00 1111 1111 1111 1111 1111	Module=3	Row=2^18-1
2^20	01 0000 0000 0000 0000 0000	Module=0	Row=2^18
2^20+1	01 0000 0000 0000 0000 0001	Module=1	Row=2^18+1
...			

Low-Order Interleave

- good if CPU can request multiple adjacent memory locations



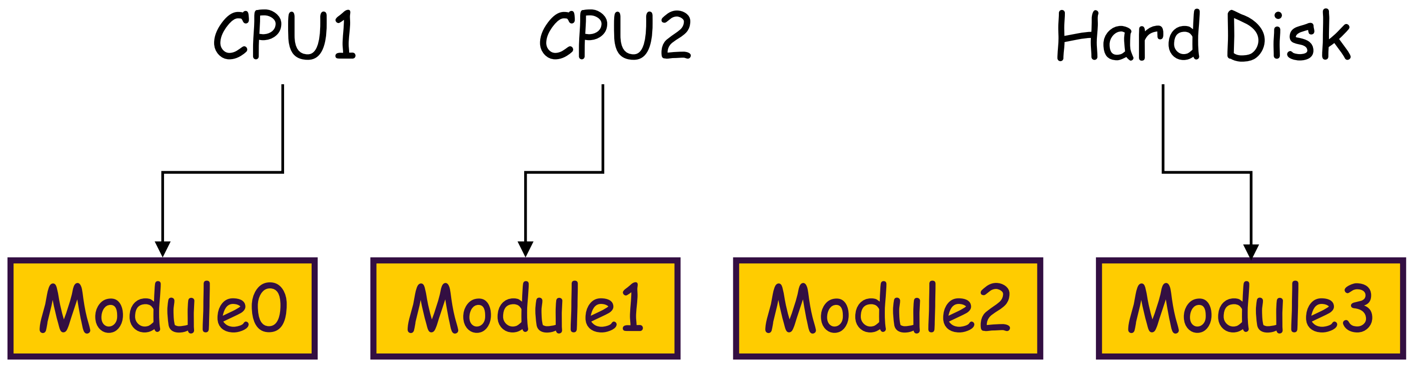
- adjacent memory locations in different modules:
  so opportunity for parallel access; useful for situations like

  i) elements in an array, e.g. Array[N], Array[N+1], Array[N+2], ...
  ii) instructions in a program, Instruction N, Instruction N+1,...

- CPU can pre-fetch adjacent memory locations in parallel
  => higher performance

High-Order Interleave

- good if modules can be accessed independently by different units
- e.g. by the CPU and a Hard Disk (or a second CPU) and the units use different modules
- parallel operation => higher performance


CPU1
CPU2
Hard Disk
Module0
Module1
Module2
Module3