# UNIT 5: MAIN MEMORY

Estructura de Computadores (Computer Organization)
Course 2018 / 2019
ETS Ingeniería Informática
Universitat Politècnica de València

# Unit goals

- To provide a global view of the computer's memory system and the most relevant features of memories
- To measure memory performance
- To justify the need for a hierarchical memory organization
- To describe the most relevant structural, functional and timing properties of synchronous dynamic RAM (SDRAM)
- To understand how memory modules are built
- To define the concept of memory map and to introduce the design of their decoding circuits
- To understand the role played by dynamic memory controllers

# Unit contents

# Bibliography and links


- D. Patterson, J. Hennessy. *Computer organization and design. The hardware/software interface*. 5th edition. 2014. Elsevier
- C. Hamacher, Z. Vranesic, S. Zaky. *Computer Organization*. 5th edition. 2001. McGraw-Hill


- Web links
  - Wikipedia: “computer memory”
  - www.micron.com (memory chips)
  - www.kingston.com (memory modules)
  - www.tomshardware.com

# 1. THE MEMORY SUBSYSTEM

What is memory and where is it located in the computer

Which technologies are more commonly used in computers

The concept of memory hierarchy

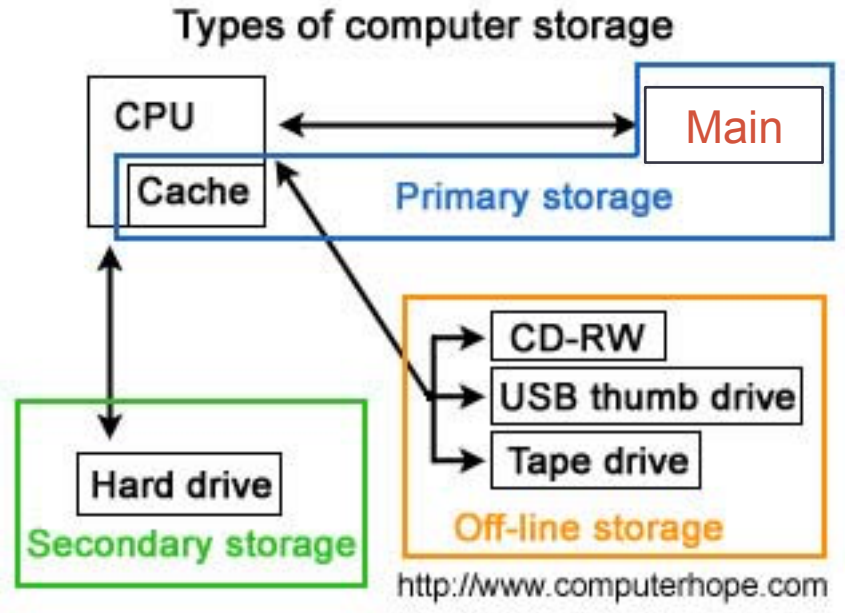
# What is memory in a computer

## Memory:

Physical support capable of retaining information, namely programs and data, either permanently or temporarily

# Where is memory located

- Memory pervades the computer:
  - Primary storage
    - Cache memory
    - Main memory
  - Secondary storage
    - Provides extended capacity and enables multiprogramming through Virtual Memory techniques
  - Off-line storage
    - Backup, portable storage
- Main memory is AKA *RAM*
  - Although this is not precise



# Memory technologies

- Four primary technologies are used in today's computers:
  - Main memory uses synchronous dynamic RAM (SDRAM)
  - Cache memory is based on static RAM (SRAM)
  - Secondary storage is supported by magnetic disks
  - Flash memory has different uses:
    - As a main memory in portable/embedded devices (replacing SDRAM)
    - As high-performance secondary storage (replacing magnetic disks)
- Off-line storage is supported by a variety of devices using different technologies
  - Optical (CD-ROM, DVD…)
  - Flash (pendrives)
  - Magnetic (external disks, tape)
- Each technology provides different features and performance

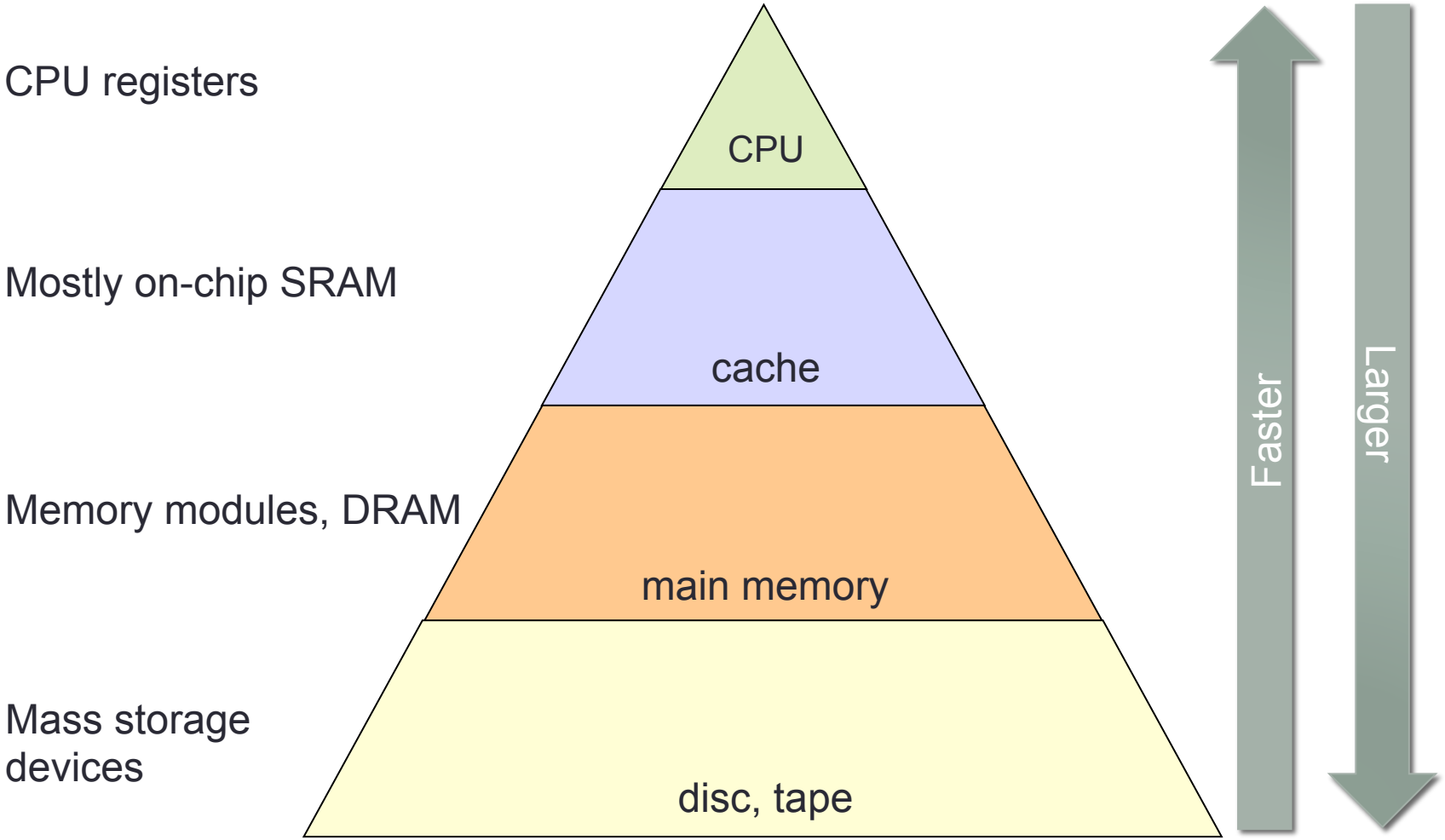
# Memory hierarchy

- Ideally, computer memory should be both large and fast
  - However, the larger the slower; the faster the smaller
- Other features need also be considered
  - Cost per bit, power consumption, reliability…
- No single device is ideal under all requirements

| Technology | Access time (typ.) | Cost/GB (yr. 2012) |
|---|---|---|
| SRAM | 0.5 .. 2.5 ns | $500 .. $1000 |
| DRAM | 50 .. 70 ns | $10 .. $20 |
| Flash | 5,000 .. 50,000 ns | $0.75 .. $1 |
| Magnetic disk | 5,000,000 .. 20,000,000 ns | $0.05 .. $0.10 |

- A hierarchical memory organisation provides the illusion of a large, fast, and affordable memory
  - Each hierarchy level uses the most convenient technology

# Memory hierarchy



# 2. BASIC CHARACTERISTICS OF MEMORY

Access modes
Capacity
Access and cycle times
Bandwidth

# Access modes

- Random access: access time is independent from data location
  - Typical of semiconductor memories. E.g., cache and main memory, ROM memory
- Sequential access: access time is proportional to the location of data in the media
  - E.g., magnetic tape
- Direct access: access comprises two positioning movements of the reading device, one direct and one sequential
  - Typical of magnetic discs
- Associative access: you ask whether some data is in memory
  - AKA contents-addressable memory
  - Access time is independent from data location – but larger than typical *random access memories*, due to the need for the hardware to find the searched data
  - Examples: cache and virtual memory directories (see next unit)

# Accessing memory contents

- Word access
  - Word size is defined by the width of the memory bus (typical sizes range between 1 and 16 bytes, with 4 bytes in MIPS32)
  - Word (or sub-word) access requires a single bus operation
    - Examples: traffic between cache and CPU, or between main memory and CPU (in systems with no cache)
- Block access
  - A block is formed by a set of words
  - Block access may require a number of bus operations
    - Examples:
      - hard disks, optical disks (blocks of 256 B .. 1024 B)
      - transfers between main memory and cache (blocks of 16, 32, 64, 128 B...)
        - Busses for the first cache levels are often block-wide, requiring only one transfer

# Memory capacity

- Information (data or code) is typically stored in bytes (8×bit)
- B = Byte; b = bit
  - Each byte is located at a unique address
- Most memory chips enable word access
  - …but each byte preserves its address: a word takes as many addresses as bytes it contains
- The capacity of a memory device denotes the total number of bytes (sometimes *bits*) it can store
- Prefixes

| Name | value ($2^n$) | value ($10^n$) |
|---|---|---|
| Kilo (K) | $2^{10}$ | $10^3$ |
| Mega (M) | $2^{20}$ | $10^6$ |
| Giga (G) | $2^{30}$ | $10^9$ |
| Tera (T) | $2^{40}$ | $10^{12}$ |
| Peta (P) | $2^{50}$ | $10^{15}$ |

# Memory capacity: examples

- Total capacity, expressed in bytes
  - 1024 Bytes = $2^{10}$ Bytes = 1 KB
- Number and size of words
  - 128 K words of 2 Bytes (16 bits) = 128 K × 2 B = 256 KB
- Total capacity and word size
  - 8 MB in 32-bit words = 8 MB in 4-Byte words = 2 M × 4 B
    = 2 M × 32 b
- Other examples
  - 64 Kb = $2^{16}$ b = $2^{13} \times 2^3$ b = 8 KB
  - 256 Mb = $2^8 \times 2^{20}$ b = $2^5 \times 2^{20} \times 2^3$ b = 32 MB
- Example: calculate the number of 16 b words in a memory of 1 Gb

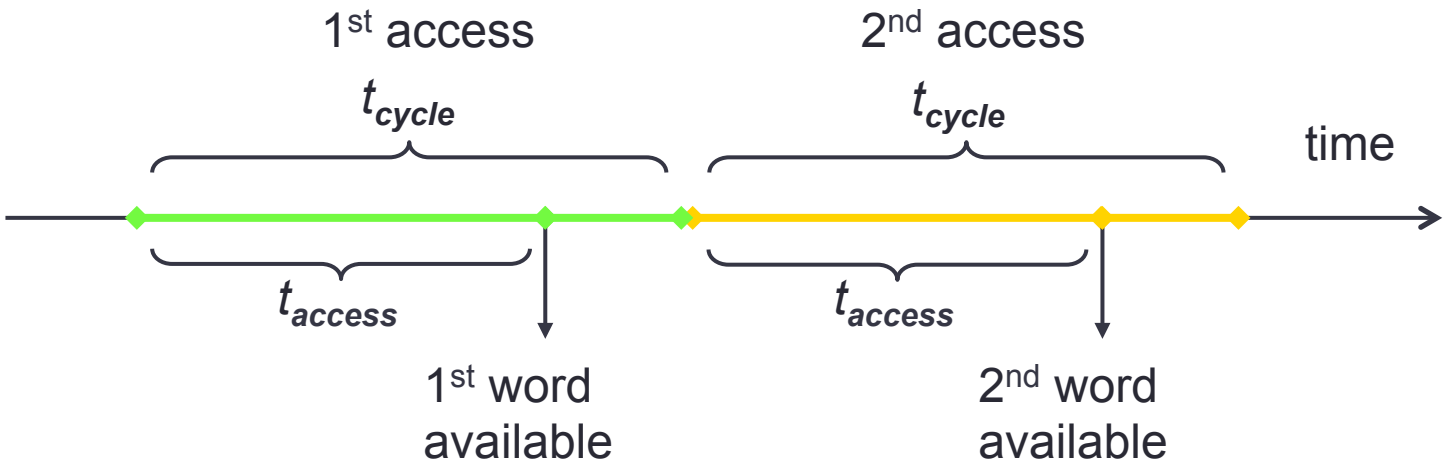
# Timing of memory accesses

- Access time of a memory: is the **maximum** time elapsed between the start of the (read or write) operation and the arrival or storage of:
  - the data word (in word accesses), or
  - the first word of the block (in block accesses)
- Measured in seconds
  - Prefixes:

| Name | Value |
|---|---|
| milli (m) | $10^{-3}$ |
| micro (μ) | $10^{-6}$ |
| nano (n) | $10^{-9}$ |
| pico (p) | $10^{-12}$ |

# Timing of word access

- Cycle time: **minimum** time required by the memory device between two consecutive accesses
- In general, $t_{cycle} \geq t_{access}$
- Example (based on a semiconductor memory)



# Transfer speed: Bandwidth

- Bandwidth: amount of data transferred per time unit

$$B = \frac{D}{t}$$

  - $D$: amount of data transferred, in bytes
  - $t$: time in seconds it takes the memory to transfer those $D$ bytes
  - $B$: bandwidth in B/s
- $B$ is either measured in B/s or b/s (KB/s, MB/s,… Kb/s, Mb/s…)
  - Caution: when time is involved, prefixes denote powers of 10
- Example: find the bandwidth (in MB/s) of a memory chip with:
  - Word size, $W$ = 32 b
  - Cycle time, $t_c$ = 20 ns
  - Access time, $t_a$ = 15 ns

# Transfer speed: Bandwidth

- Another way to calculate bandwidth:

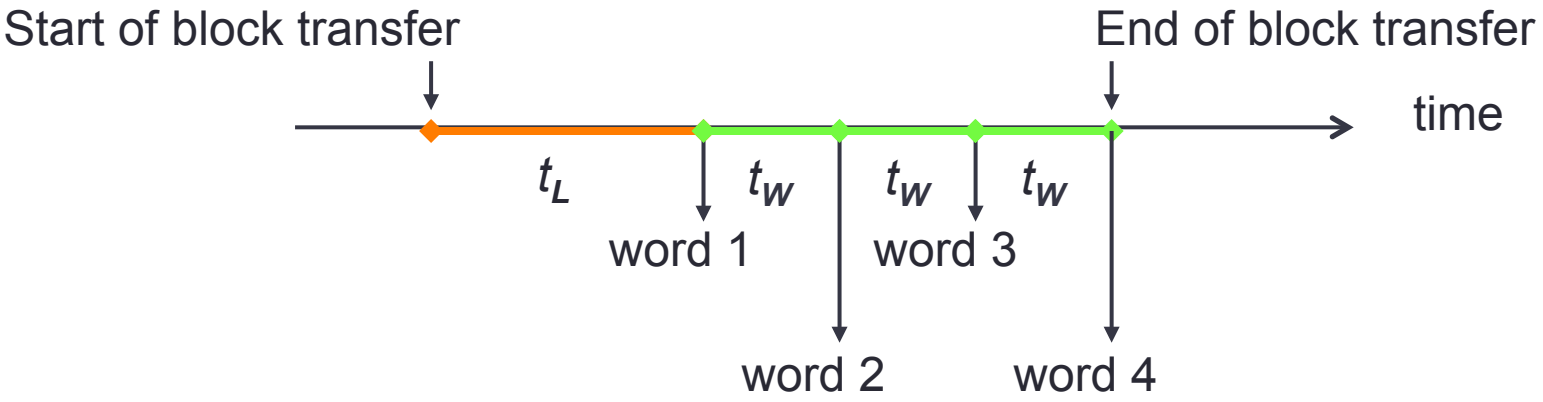
$$B = Bytes\_transferred\_per\_cycle \times Transfer\_frequency$$

  - *Bytes_transferred_per_cycle* depends on data bus width
  - *Transfer_frequency* is the inverse of cycle time
- In our previous example:
  - Bytes_transferred_per_cycle = 4 (word size was 32 b = 4 B)
  - *Transfer_frequency* = 1/20ns = 50 MHz (remember 1 Hz = 1 $s^{-1}$)
  - Hence, 4 B × 50 MHz = 200 MB/s

# Timing of block accesses

- Typical block accesses involve two steps:
  - Access the first word in the block (imposes an initial latency $t_L$)
  - Access subsequent words in block at regular intervals ($t_W$)
- Bandwidth of a block transfer is measured starting at the arrival of the first word in the block ($B$ is asymptotic)

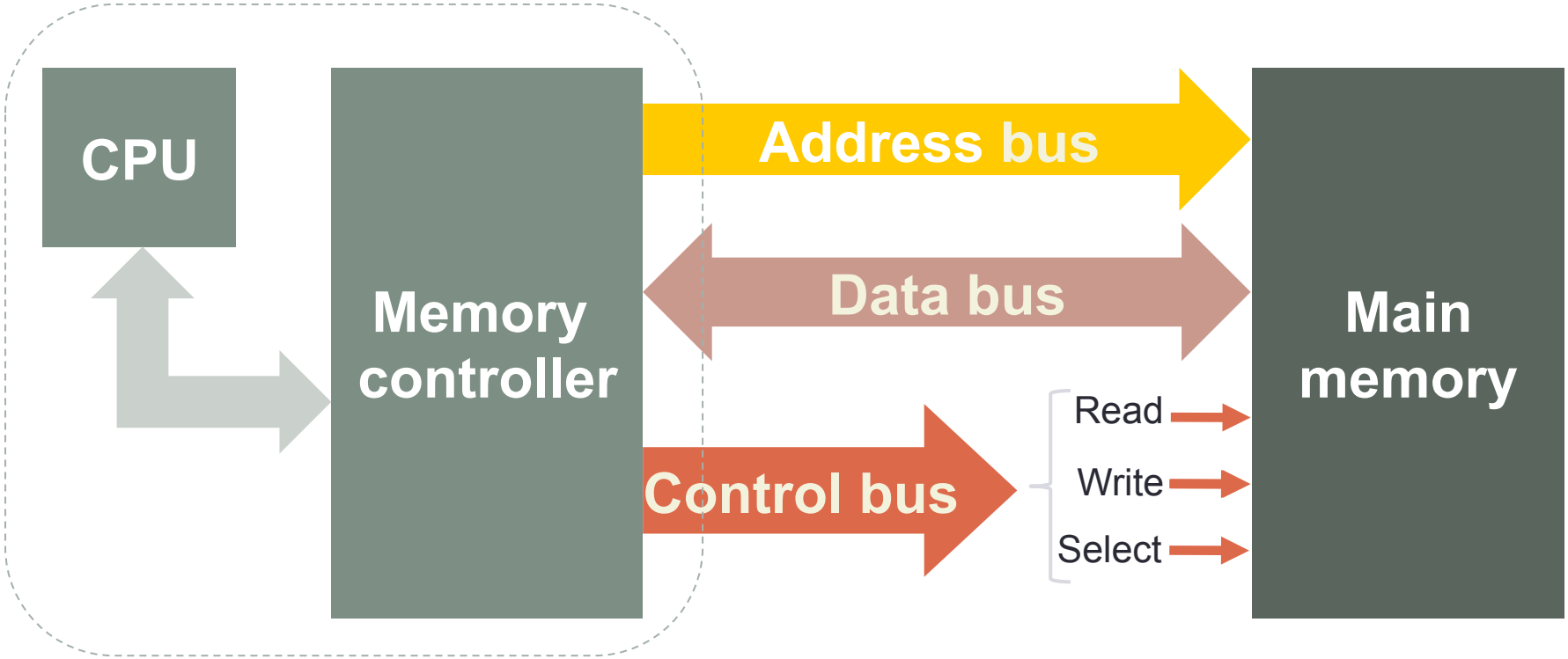
$$B = \frac{WordSize}{t_W}$$



# 3. MAIN MEMORY

External interface
Addressing main memory
Storage arrangement: endianness
Data arrangement in memory (MIPS R2000)
Accessing data

# External interface




- Main memory is supported in today's computers by SDRAM chips
- The memory controller adapts CPU busses to main memory busses
  - Both in terms of signals and timing

# Addressing main memory

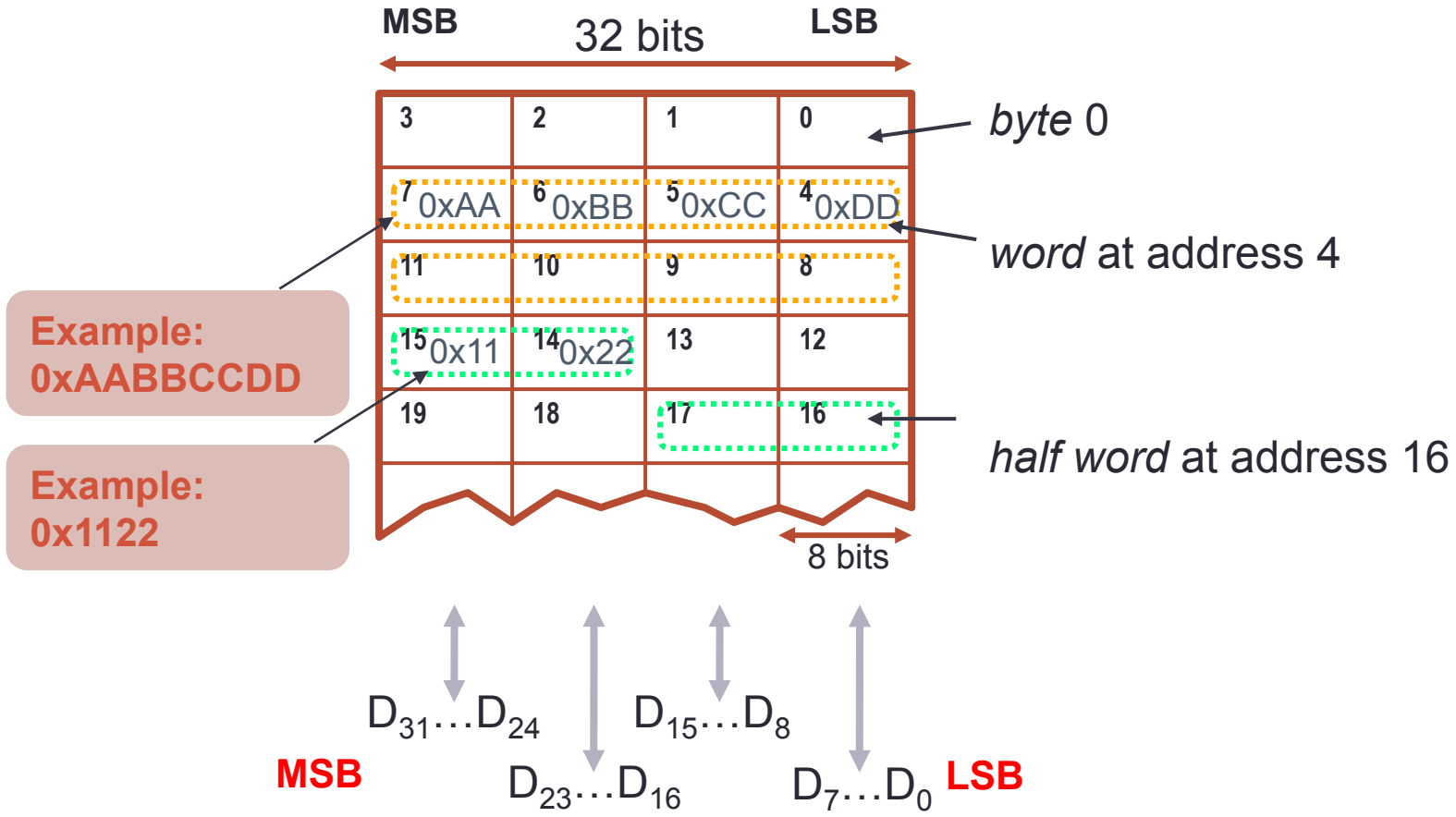
- Addresses: each **byte** in memory has a unique address
  - A byte (not a bit!) is the minimum addressable amount of data
- Parameters of Mem-CPU connection
  - Word width, in bits or bytes
  - Addressing capacity, in number of address bits (n) or in total number of addressable bytes ($2^n$)
    - Word width ≈ data bus size; addressing capacity ≈ address bus size
  - Endianness: how bytes are arranged in memory
    - Little endian or big endian
- Example: MIPS R2000 addressing space
  - Word width of 32 b = 4 B
  - 32 address bits for a total addressing capacity of $2^{32}$ B = 4 GB

# Storage arrangement: endianness

- With multi-byte words, two arrangements are possible
  - Low-order bytes of the word use lower byte addresses: little endian
  - High-order bytes of the word use lower byte addresses: big endian
- Example: in the MIPS R2000 (W = 32 b)
  - Words are aligned to addresses multiple of 4
  - Half words are aligned to even addresses (multiple of 2)
  - Bytes may be located at any address, with no restriction
  - So when you address a word, where are the most and least significant bytes located?

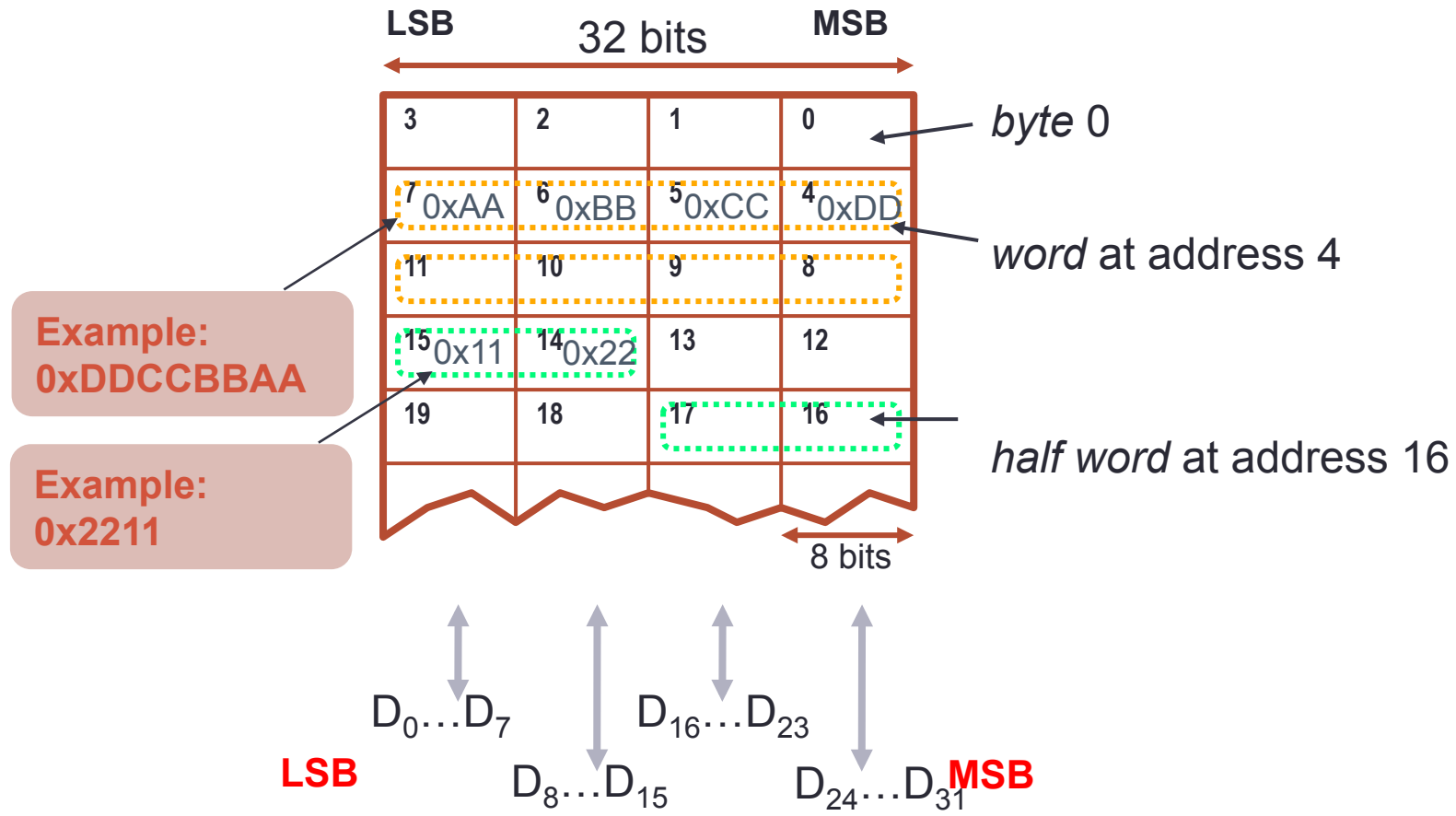
# Data arrangement: endianness

- Little endian example



# Data arrangement: endianness

- Big endian example




Further info: http://support.microsoft.com/kb/102025/en-us Wikipedia: “Endianness”

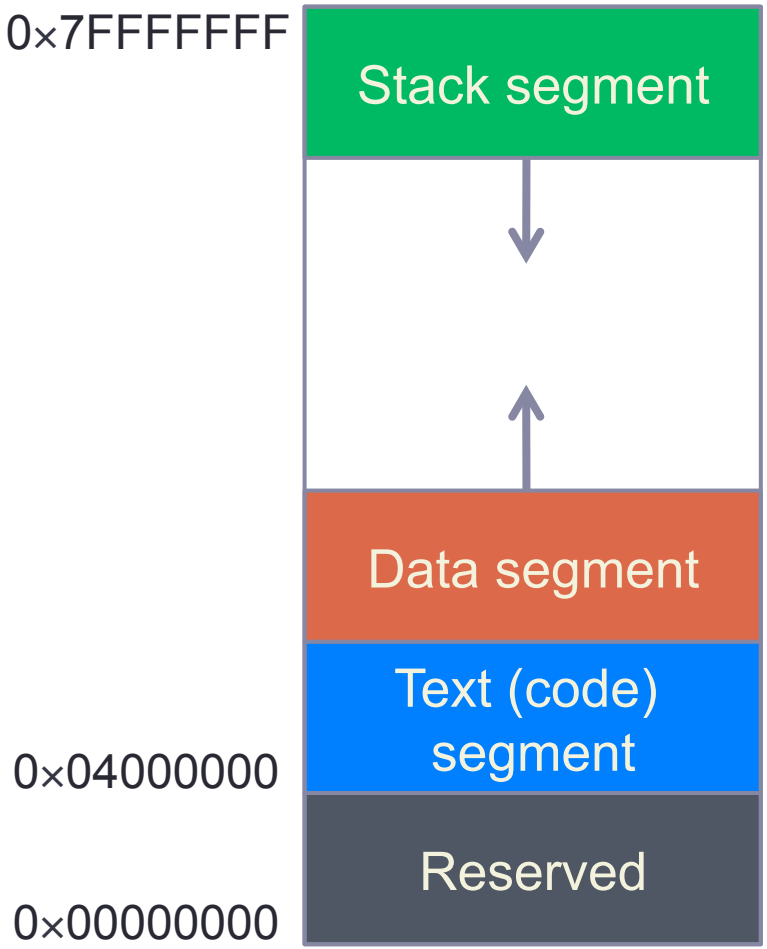
# Data arrangement in assembly

- Assembly directives for data declaration
  - Bytes: `.byte`
  - Half words (2 B): `.half`
  - Words (4 B): `.word`
  - Char strings: `.ascii` and `.asciiz`
- Example

```
            .data 0x1000A000
dni:        .word 0x98534169
strz:       .asciiz "K"
codbar:     .half 0x8733
date:       .byte 0x54
name:       .ascii "Pere"
            .word 0x12345678
            .space 2
```

| | | | | |
|---|---|---|---|---|
| 0x98 | 0x53 | 0x41 | 0x69 | 0x1000A000 |
| 0x87 | 0x33 | 0x00 | 0x4B | 0x1000A004 |
| 0x72 | 0x65 | 0x50 | 0x54 | 0x1000A008 |
| 0x00 | 0x00 | 0x00 | 0x65 | 0x1000A00C |
| 0x12 | 0x34 | 0x56 | 0x78 | 0x1000A010 |
| | | 0x00 | 0x00 | 0x1000A014 |

# Logical memory arrangement (R2000)



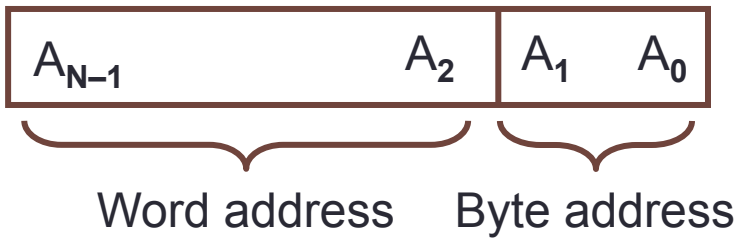

Note that this figure is not drawn to scale

# Accessing data in MIPS R2000

- Possible accesses are read/write and byte/half/word
- Read accesses
  - Byte read: `lb $t0,0($a1)`
  - Half-word read: `lh $t3,10($a0)`
  - Word read: `lw $t1,0($a2)`
- Write accesses
  - Byte write: `sb $t2,27($a3)`
  - Half-word write: `sh $t0,Var($a2)`
  - Word write: `sw $t5,0x40($a1)`
- The type of access is encoded in the instruction
- Accessing wrong addresses is an error!
  - Cases: half-word at odd address; word at address $\neq \dot{4}$
  - Both cases result in an *exception* being raised

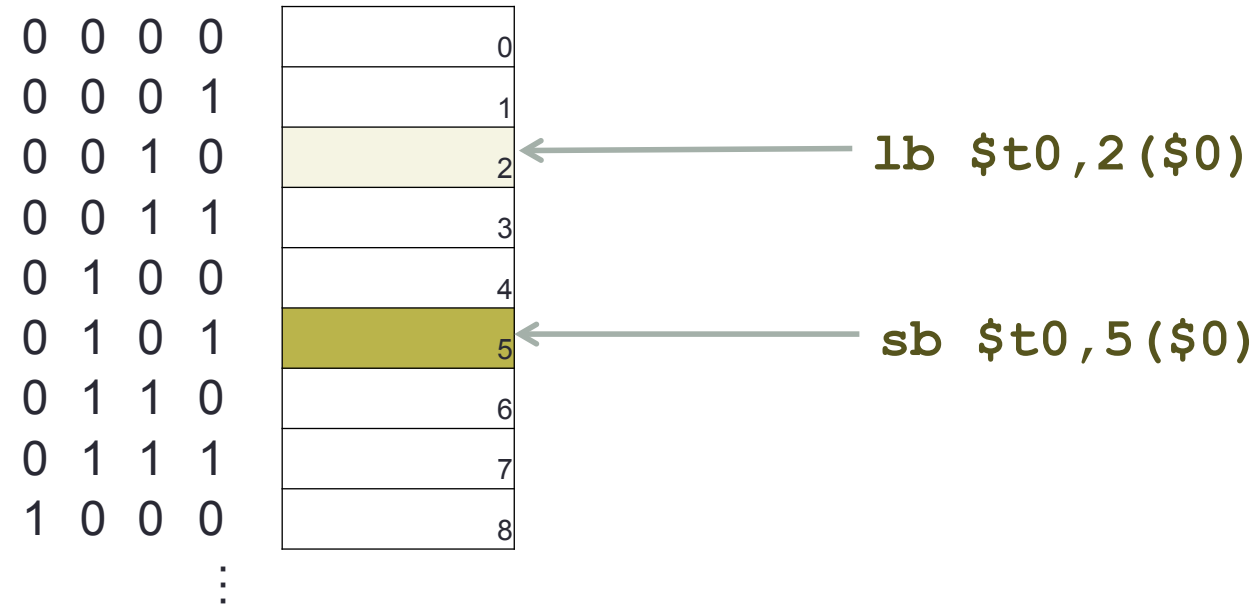
# Accessing memory: hw details

- The programmer uses logical addresses (32 b in R2000)
  - calculated as the sum of the address register and displacement (AKA offset), eg.
    - `lw $t0,28($zero)` → The address is 28, or 0x1C
- Accessing a word implies accessing its four bytes
  - The word's address is the address of its LSB (in little endian)
- Example (W = 32 b): Logical address



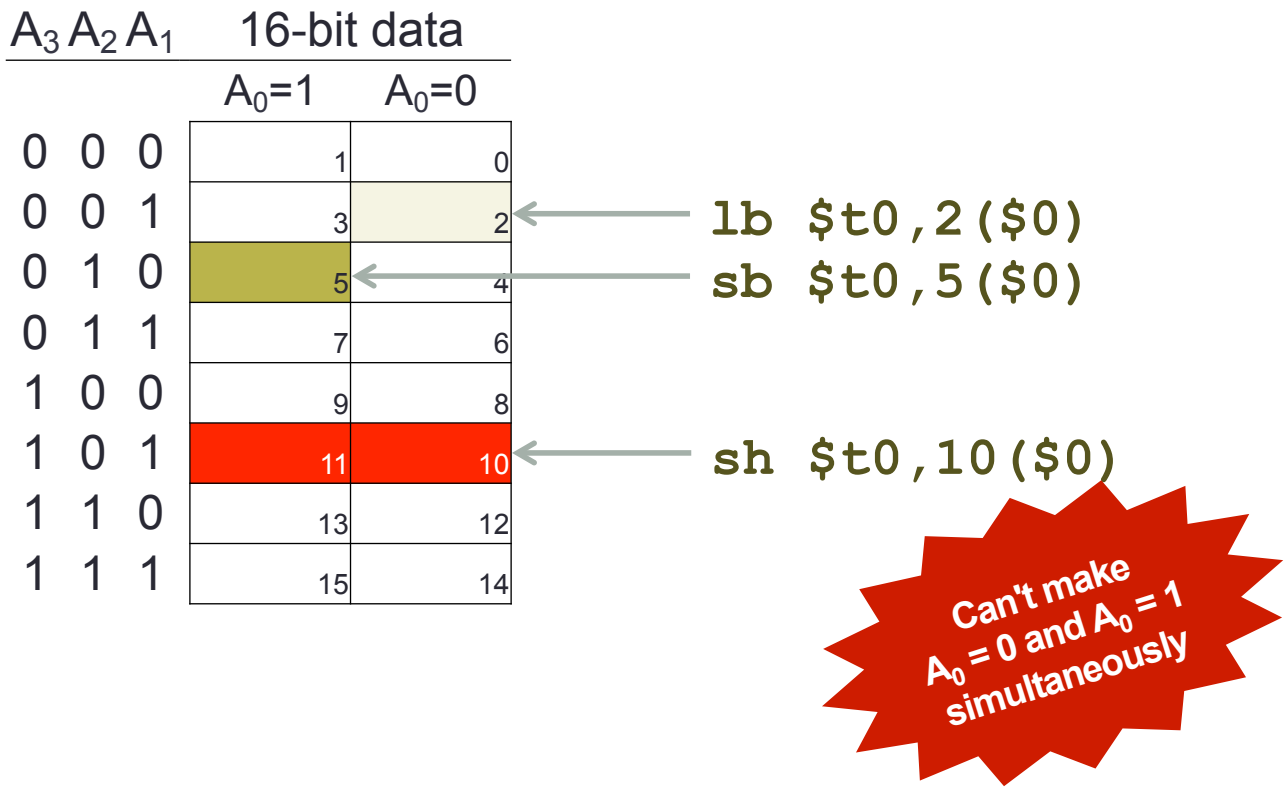
# Accessing memory: hw details

Addressing in 8-bit CPUs



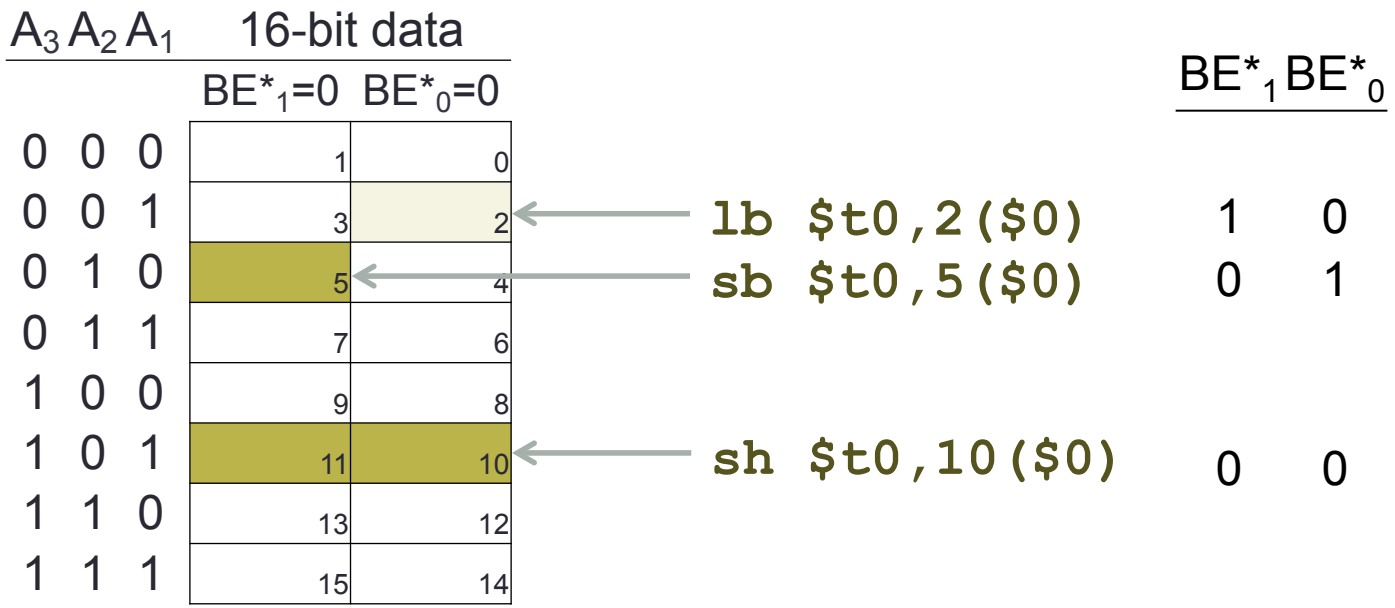
# Accessing memory: hw details

Addressing in 16-bit CPUs



# Accessing memory: hw details

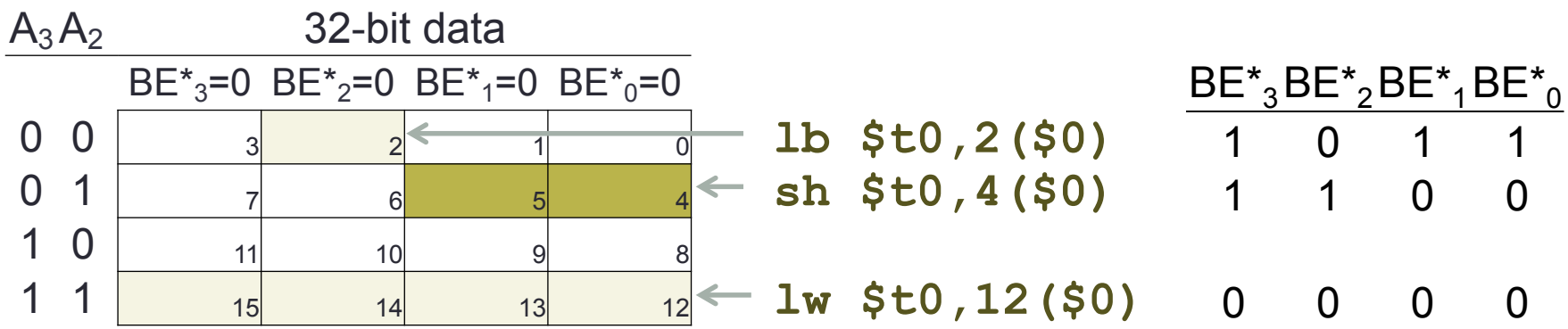
Addressing in 16-bit CPUs: Use of Byte Enable lines ($BE^*_i$)




$BE^*_i$ depend on the address (odd/even) and the type of access (byte/half)
$A_0$ is not needed as an address bus line anymore

# Accessing memory: hw details

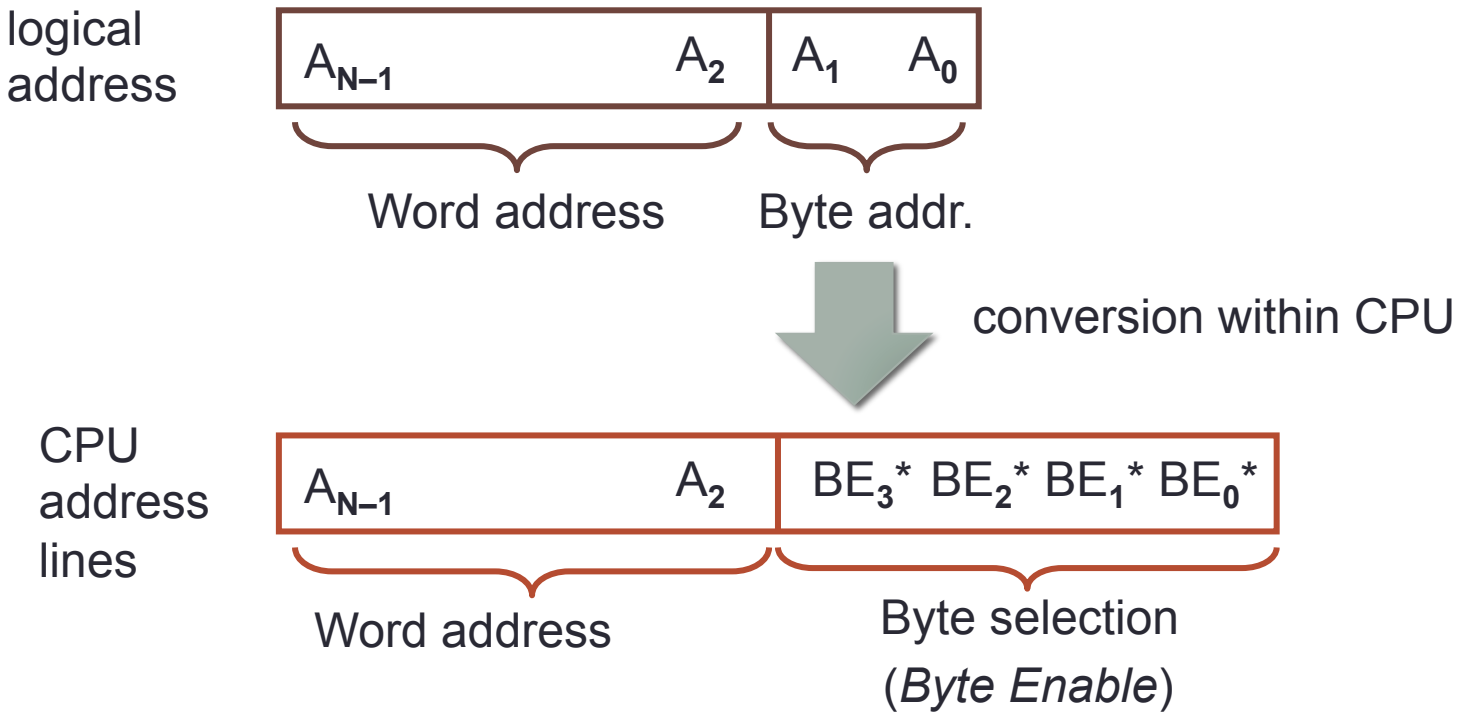
Addressing in 32-bit CPUs: Use of Byte Enable lines ($BE^*_i$)




$BE^*_i$ depend on the address (2 LSbs) and the type of access (byte/half/word)
$A_0$ and $A_1$ are not needed as address bus lines anymore

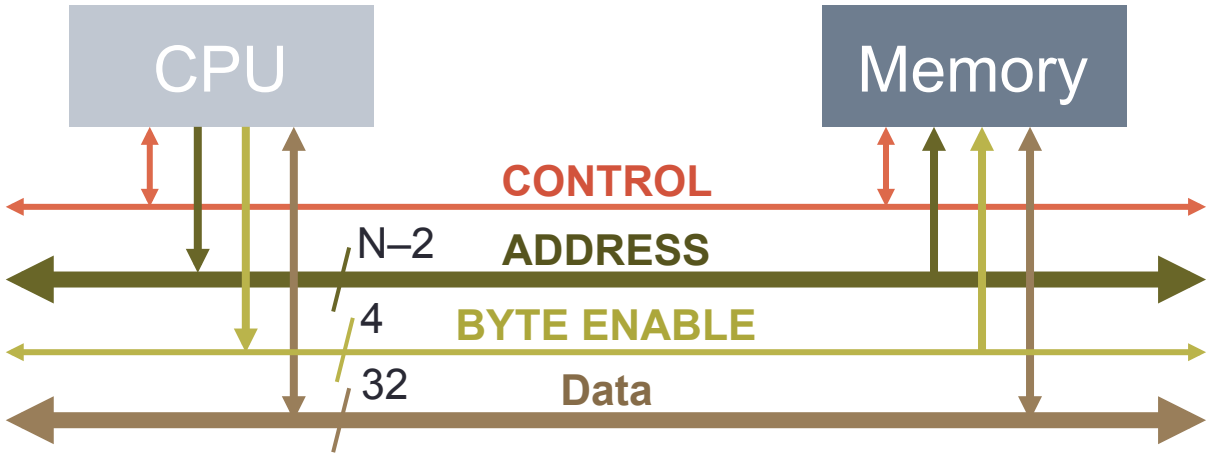
# Accessing memory: hw details

- Address lines decode the byte address bits to allow proper access to sub-word items
  - This conversion considers the type of access (byte/half/word)
  - The least significant bits of the logical address are decoded into *byte selection* (or *byte mask*) lines $BE_i^*$



# Accessing memory: hw details

- Memory bus structure, assuming
  - logical addresses of N bits ($2^N$ addressing capacity)
  - W = 32 b
  - → there will be
    - N – 2 physical address lines, $A_{N-1}$ … $A_2$
    - 4 Byte Enable lines, named $BE_0$*, $BE_1$*, $BE_2$* and $BE_3$*
    - 32 data lines, $D_{31}$ … $D_0$
    - The needed control lines (read, write,…)



# Accessing memory: hw details

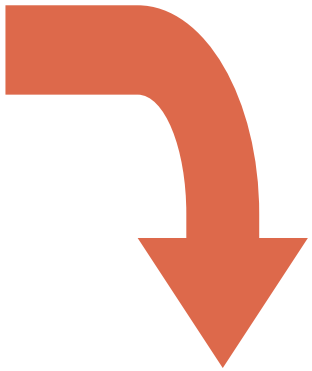
- Generating $BE_i$* signals: example W = 32 b
  - The control unit within the CPU is in charge

| Inputs | | | Outputs | | | |
|---|---|---|---|---|---|---|
| Access type | $A_1$ | $A_0$ | $BE_3$* | $BE_2$* | $BE_1$* | $BE_0$* |
| Byte | 0 | 0 | 1 | 1 | 1 | 0 |
| | 0 | 1 | 1 | 1 | 0 | 1 |
| | 1 | 0 | 1 | 0 | 1 | 1 |
| | 1 | 1 | 0 | 1 | 1 | 1 |
| Half word | 0 | 0 | 1 | 1 | 0 | 0 |
| | 1 | 0 | 0 | 0 | 1 | 1 |
| Word | 0 | 0 | 0 | 0 | 0 | 0 |

# Accessing memory: hw details

- Examples of $BE_i$* activation

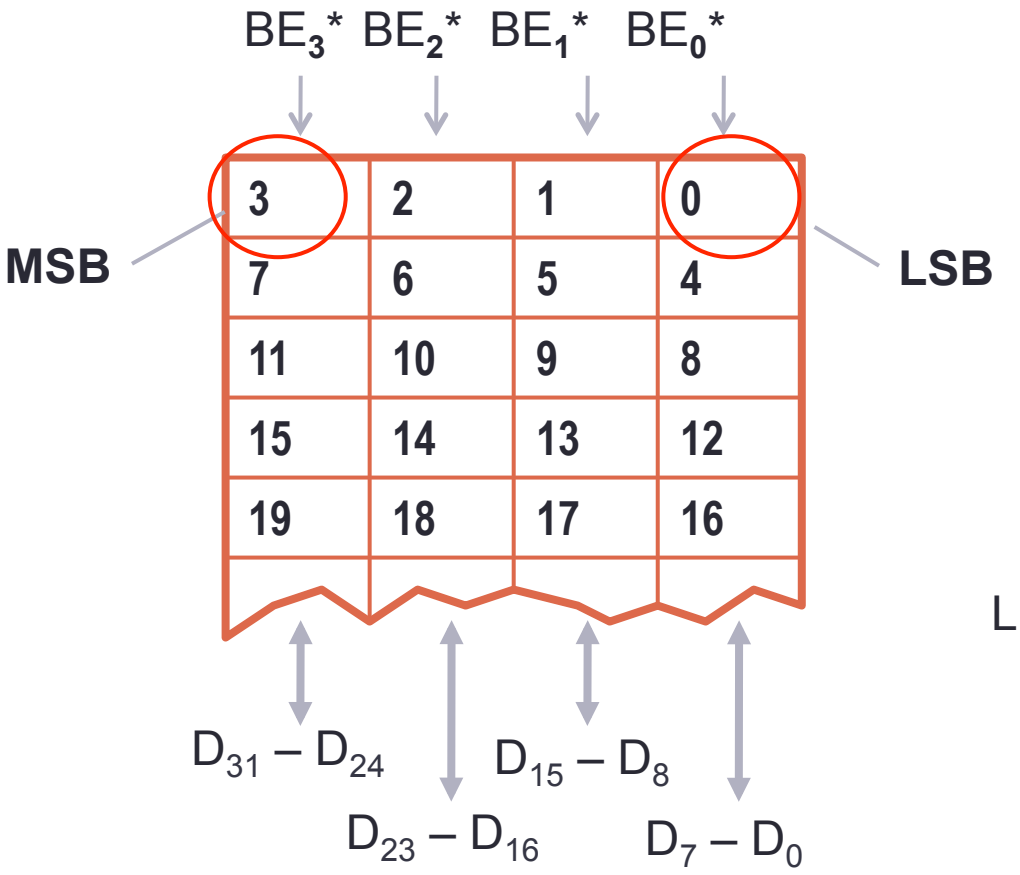
```
la $t0, 0x10000000
lw $a0, 0($t0)
lb $a1, 5($t0)
sh $a2, 6($t0)
sb $a3, 8($t0)
```



| | $A_1$ | $A_0$ | $BE_3$* | $BE_2$* | $BE_1$* | $BE_0$* |
|---|---|---|---|---|---|---|
| **lw** | 0 | 0 | **0** | **0** | **0** | **0** |
| **lb** | 0 | 1 | 1 | 1 | **0** | 1 |
| **sh** | 1 | 0 | **0** | **0** | 1 | 1 |
| **sb** | 0 | 0 | 1 | 1 | 1 | **0** |

# Accessing memory: hw details

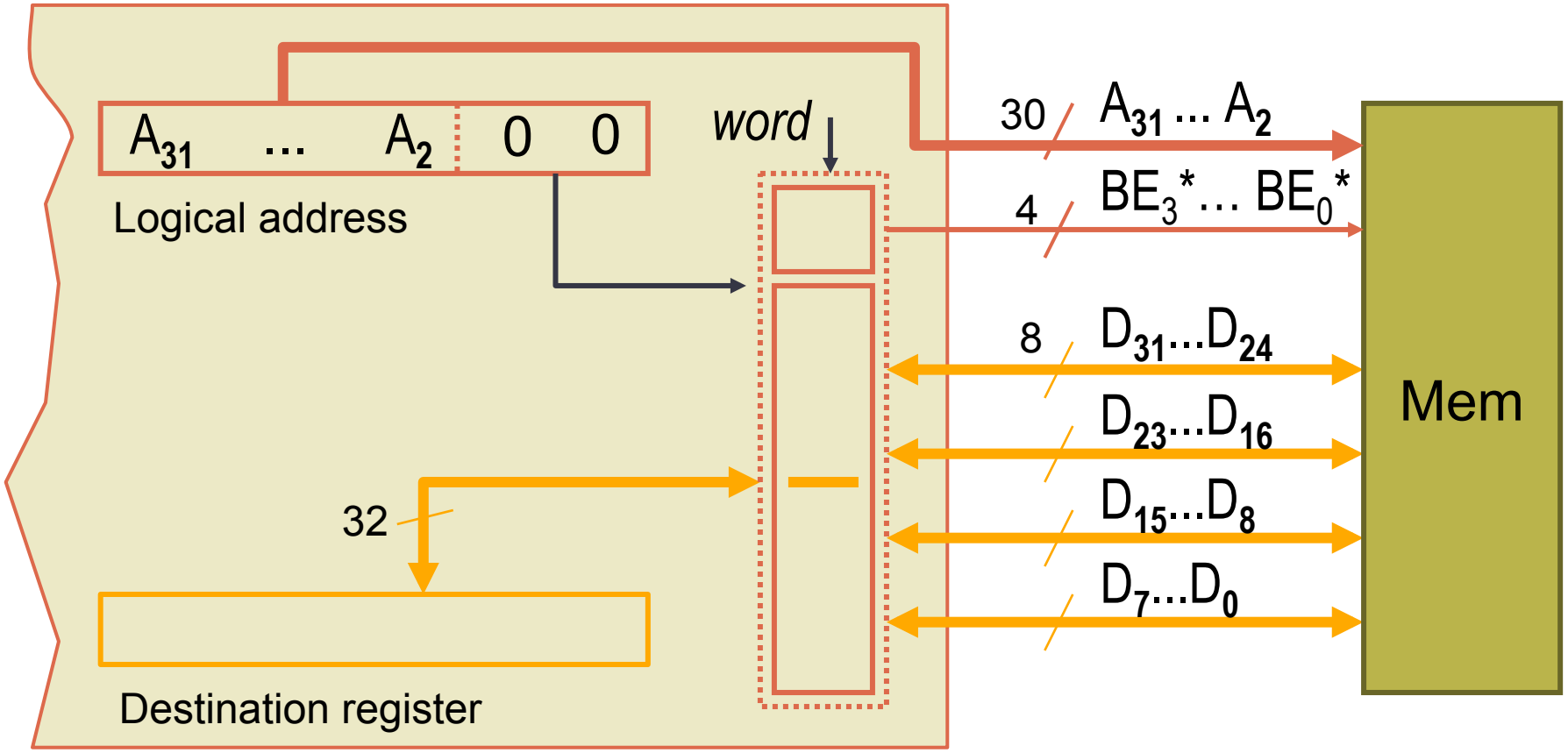
- Endianness dictates the association between $BE_i^*$ lines and data bus bytes




Little endian example

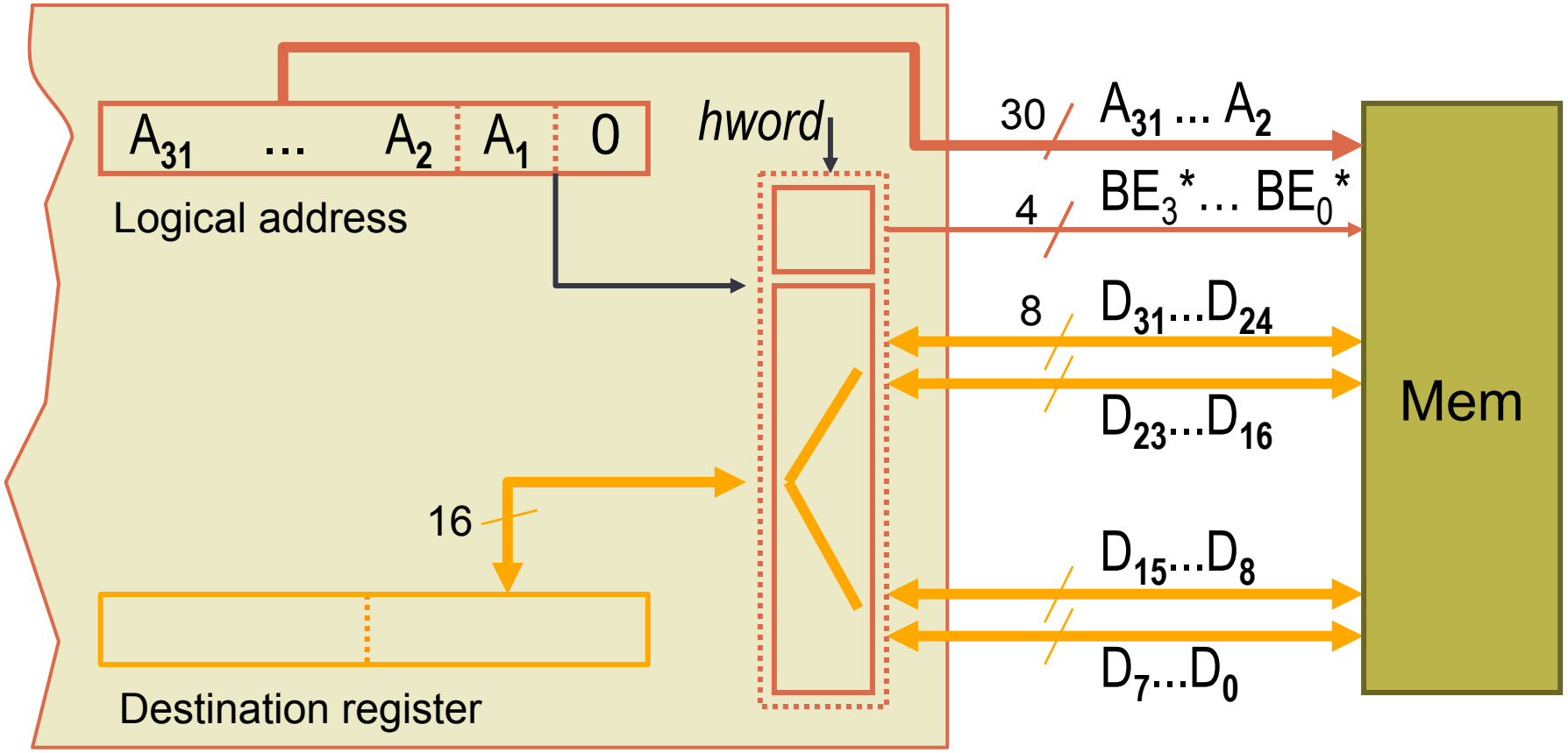
# Accessing memory: hw details

- Accessing a word (32 b)
  - Logical address must be multiple of 4 ($A_0$=$A_1$=0)



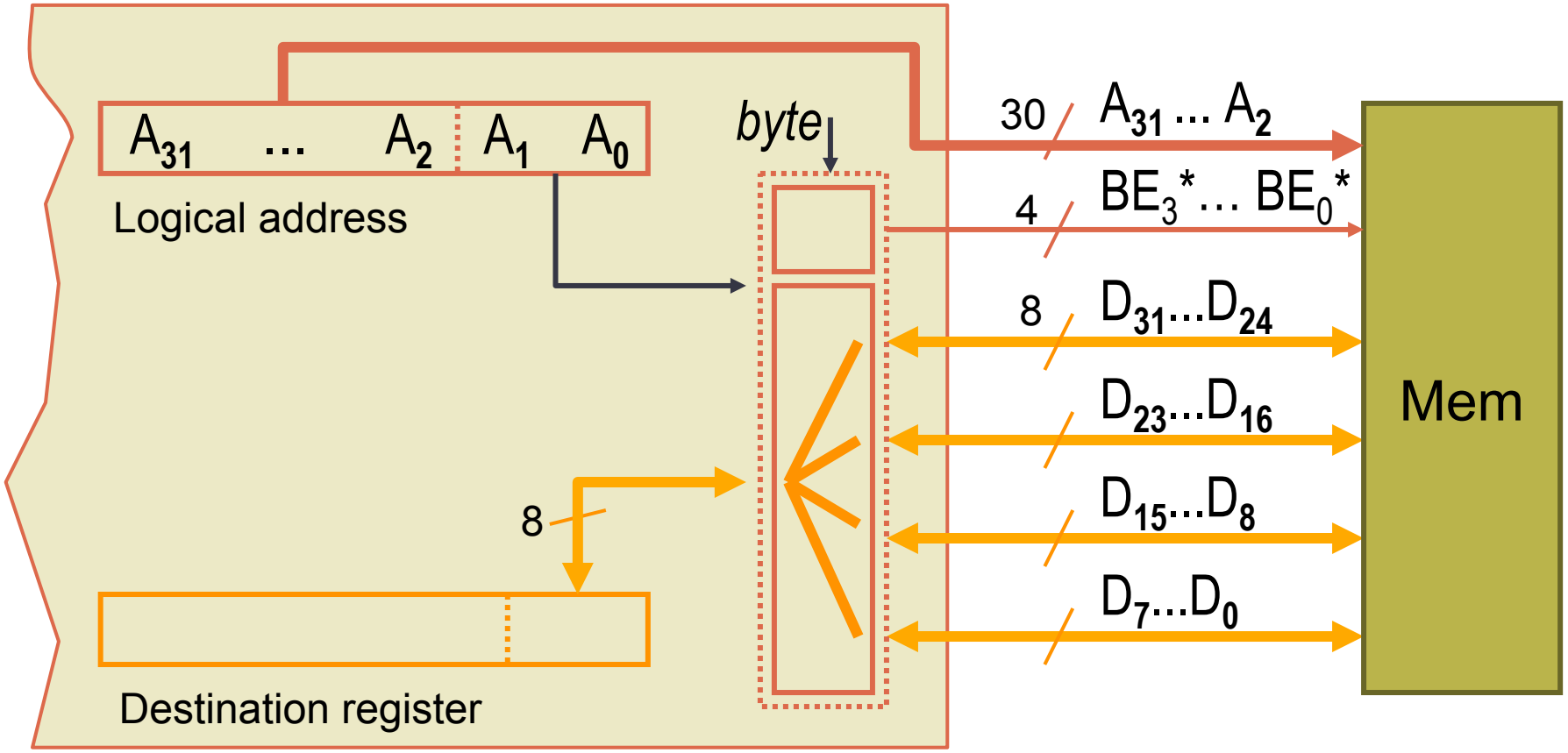
# Accessing memory: hw details

- Accessing a half word (16 b)
  - Logical address must be multiple of 2 ($A_0$=0)



# Accessing memory: hw details

- Accessing a byte (8 b)
  - Logical address may have an arbitrary value

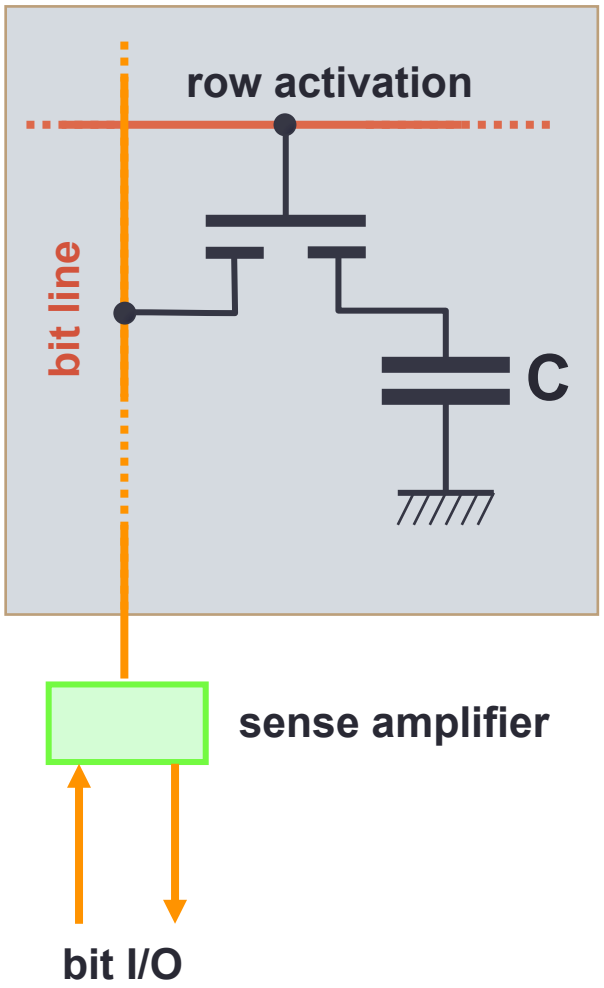


# 4. DYNAMIC RAM MEMORY

Description of DRAM / SDRAM memory
Synchronous DRAM (SDRAM) operation
SDRAM commercial example
Commands and timing
DRAM technology: status and trends

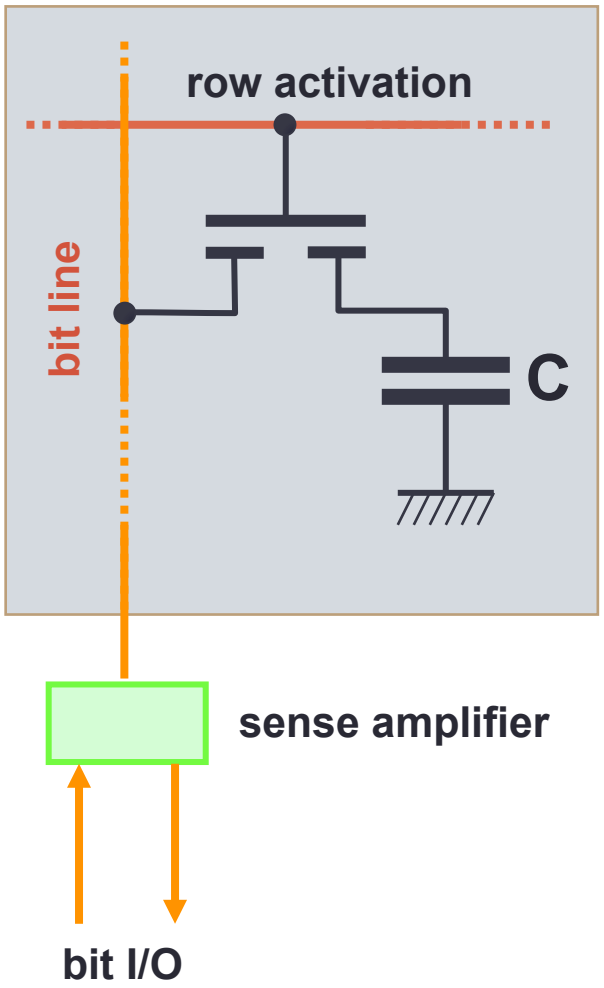
# DRAM memory basics

- DRAM is a R/W semiconductor memory
- Cells are formed by a *pass transistor* and a capacitor – bits stored as capacitor charge
- Row activation enables
  - Cell write: C is charged through the bit line
  - Cell read: C's charge flows through the bit line
- Charge is very small
  - Bit line is *precharged* to an intermediate voltage and upon row activation, the sense amplifier detects and amplifies slight voltage variations around it
    - Bit line has an associated capacitance
- Charge leak imposes the need for refresh
  - Refresh is accomplished by reading the cell and overwriting the same value



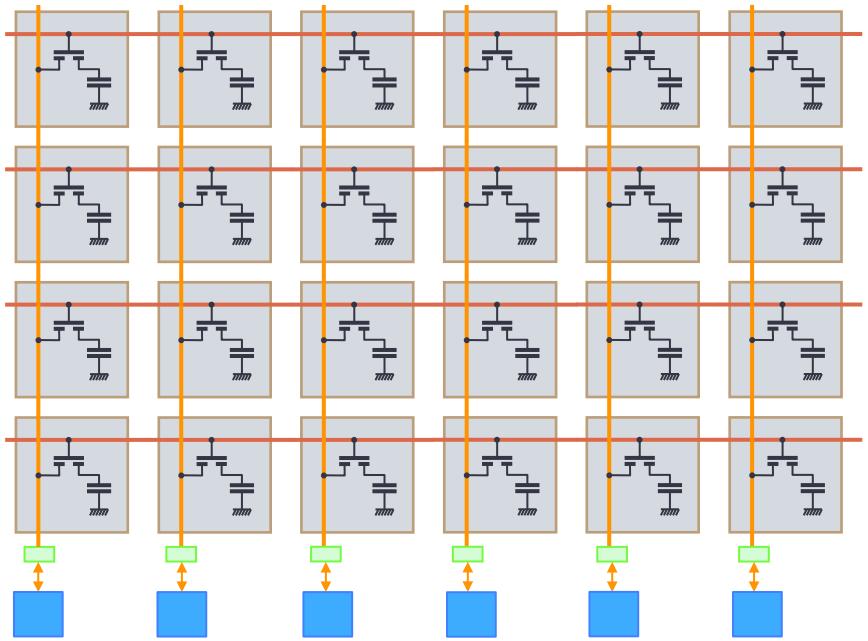
# DRAM memory basics

- DRAM is preferred to Static RAM for main memory because:
  - DRAM requires less area per bit
    - Static RAM cells use flip-flops (~6 transistors), SDRAM uses only one transistor + capacitor
    - Less area implies higher density (more capacity in the same area)
  - DRAM has much lower cost per bit
  - Although DRAM is slower for individual word access, this is mitigated by *block access*
    - Especially convenient for systems with cache



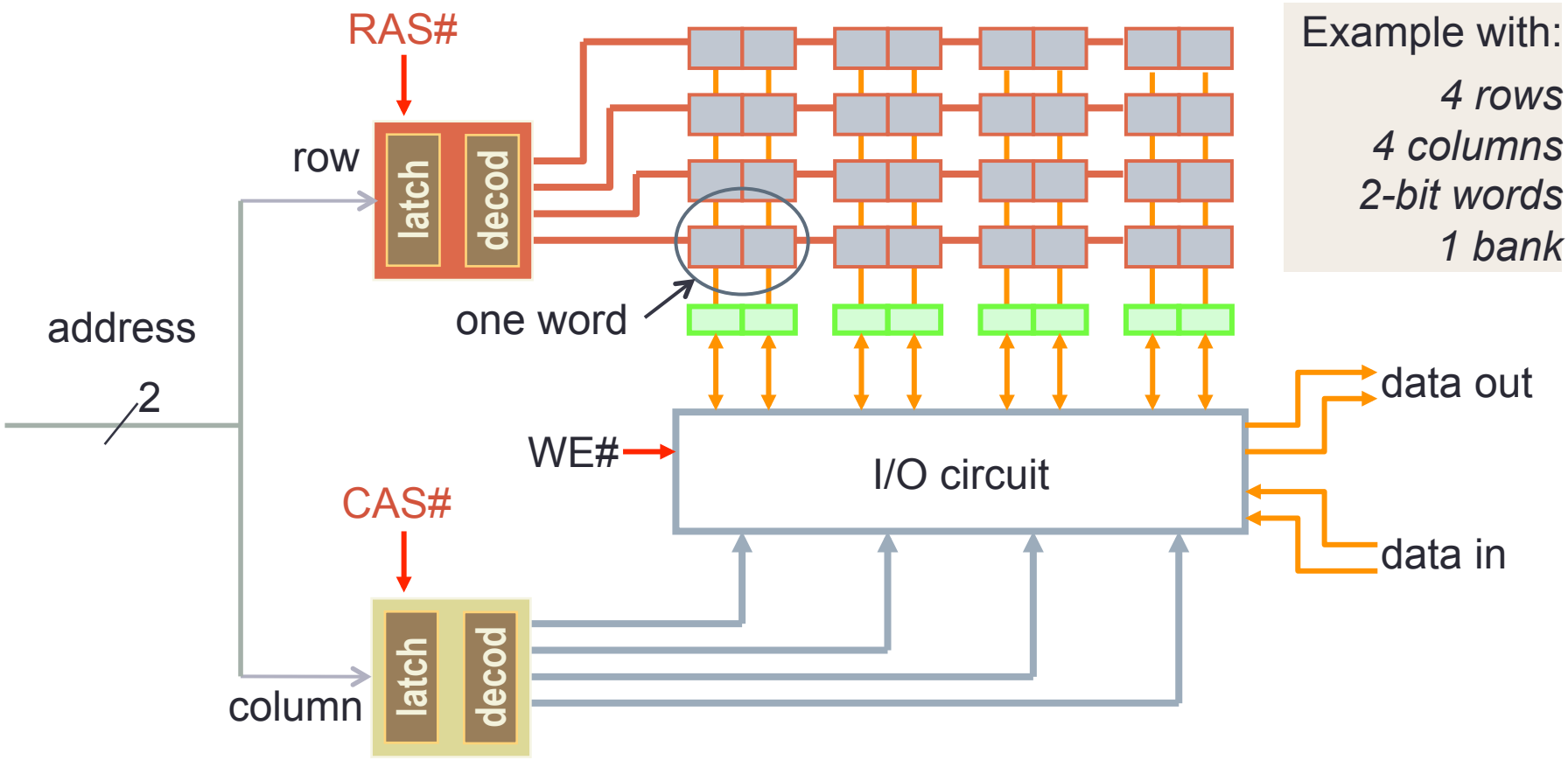
# DRAM memory basics

- Cells are internally arranged as a matrix
  - Accessing a cell is a two-step process: row, then column
- All bits in an *open* row are read and latched
- Subsequent bits from the same row can be read from the latches, without a row opening delay
- Additionally, a single refresh cycle is effective for all cells in a row



# DRAM memory: internal organization

- Each word is stored in the set of cells in the intersection of a row and a column



# DRAM operation: addressing

- Addresses issued to DRAM are structured
  - In a matrix of $2^{r+c}$ words ($2^r$ rows × $2^c$ columns), the $n = r + c$ address bits are organized as:


  - Each *linear* address has two components
    - *row* = *address* **div** $2^c$ *column* = *address* **mod** $2^c$
    - Conversely, *address* = (*row* × $2^c$) + column

row,column form

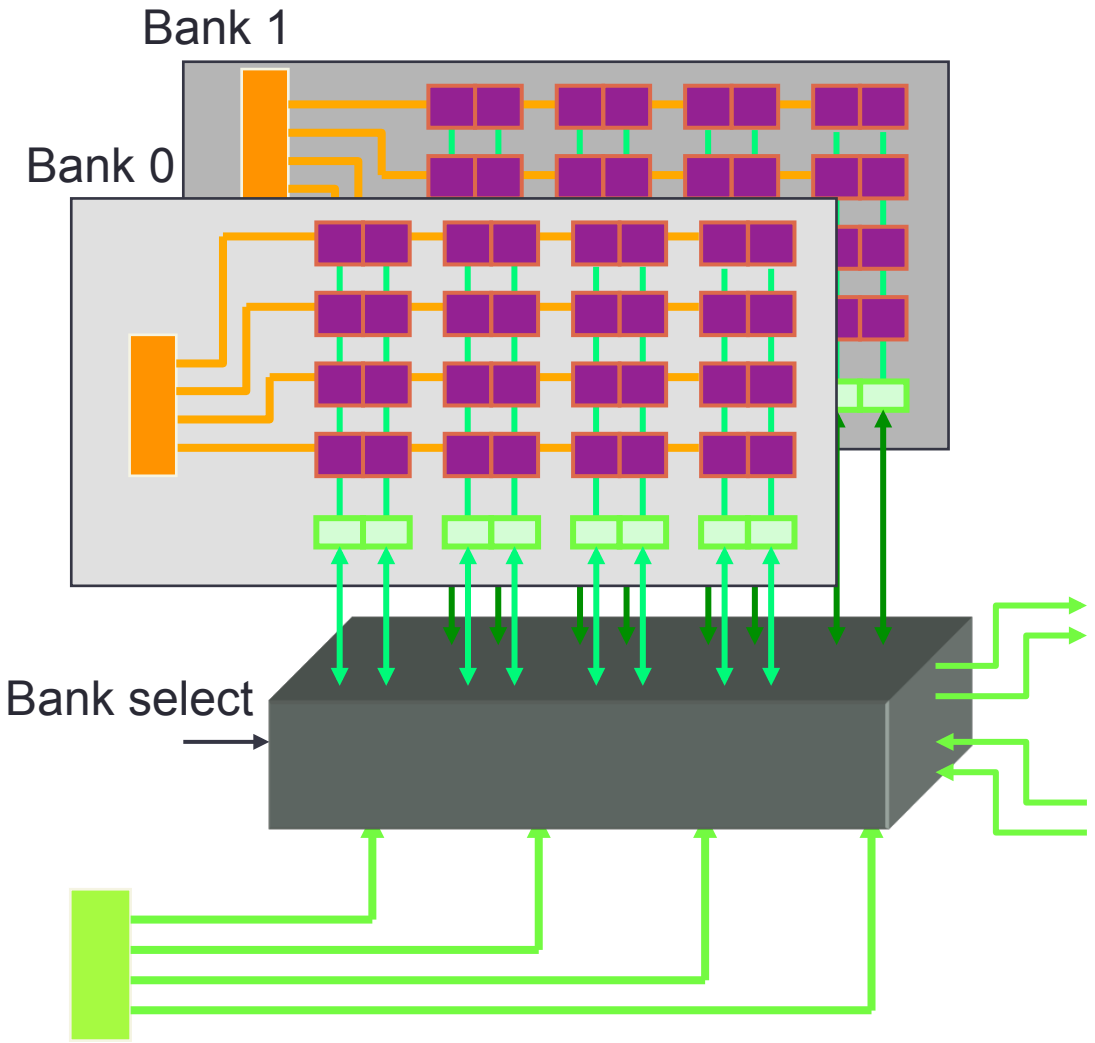
| | | | | |
|---|---|---|---|---|
| 0,0 | 0,1 | 0,2 | ···· | $0,2^c-1$ |
| 1,0 | 1,1 | 1,2 | ···· | $1,2^c-1$ |
| 2,0 | 2,1 | 2,2 | ···· | $2,2^c-1$ |
| ⋮ | ⋮ | ⋮ | | ⋮ |
| $2^r-1,0$ | $2^r-1,1$ | $2^r-1,2$ | ···· | $2^r-1,2^c-1$ |

linear address form

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | ···· | $2^c-1$ |
| $2^c$ | $2^c+1$ | $2^c+2$ | ···· | $2^{c+1}-1$ |
| $2^{c+1}$ | $2^{c+1}+1$ | $2^{c+1}+2$ | ···· | $2^{c+2}-1$ |
| ⋮ | ⋮ | ⋮ | | ⋮ |
| $2^n-2^c$ | $2^n-2^c+1$ | $2^n-2^c+2$ | ···· | $2^n-1$ |

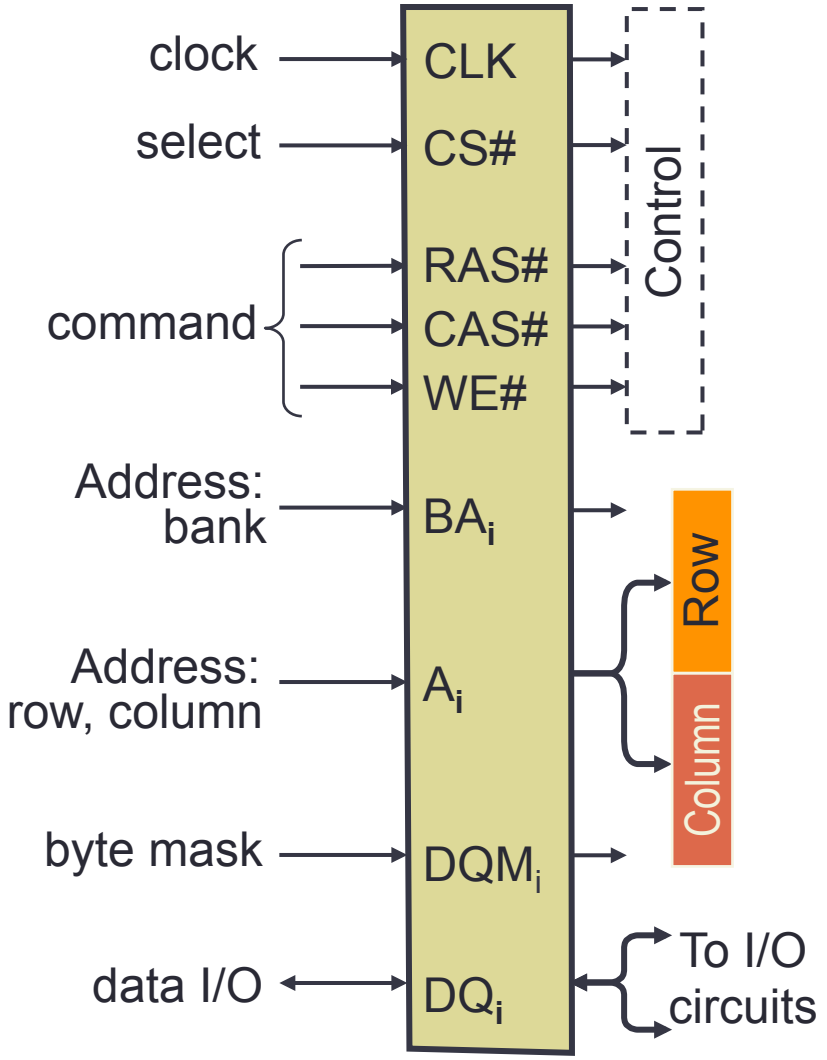
# DRAM memory: internal organization

- DRAM chips are organized in banks
  - Each bank holds a matrix, including amps and row latch
  - Common (shared) I/O circuit
  - Capacity = $B \times 2^r \times 2^c \times w$ bits
    - $r$ row bits, $c$ col bits, $B$ banks
  - Up to $B$ rows may be open at a time
  - There are $b$ bank address lines such that $2^b = B$



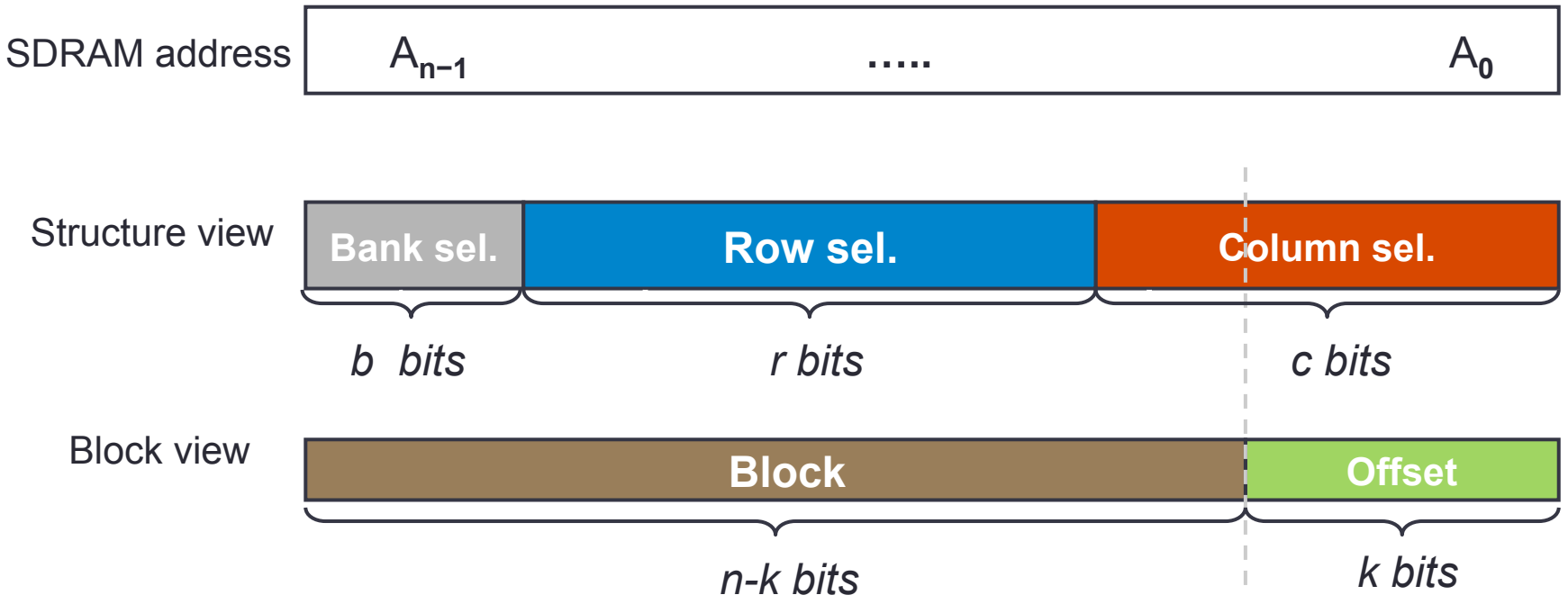
# SDRAM memory: external interface

- Global chip select and clock
  - Clock is needed by Synchronous DRAM (SDRAM)
- Bank and row/column form the address
- Commands are predefined combinations of three signals
- Byte mask lines allow selective byte enable
- Data I/O: as many lines as the chip's word size



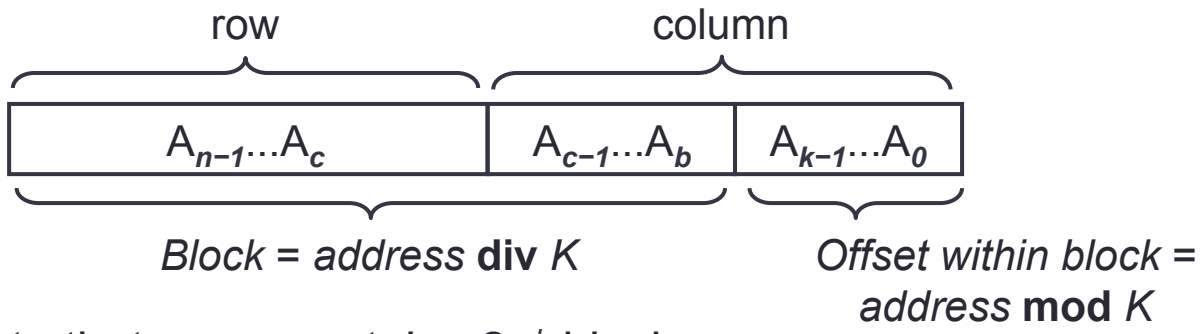
# SDRAM addressing

- DRAM size: $2^n \times w$ bits
- Internal structure: $2^b$ banks × $2^r$ rows × $2^c$ columns
- Address length: n = b + r + c
- Blocks of $2^k$ words ($2^{n-k}$ blocks in SDRAM)



# SDRAM operation: block access

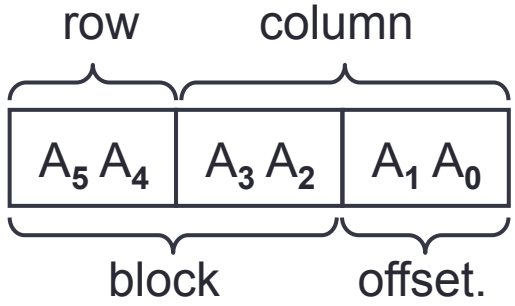
- For efficiency reasons, traffic between cache and main-memory is in the form of blocks
  - Blocks are formed by $K = 2^k$ contiguous words, typically 2, 4 or 8
  - Words in a block use addresses of the form $A_{n-1}\ldots A_k\ x\ldots x$
  - Once programmed at setup *K* remains constant
    - K is typically programmed to fit the *cache line* size (next unit)
  - Blocks always belong in a single row; addresses can thus be further decomposed as:

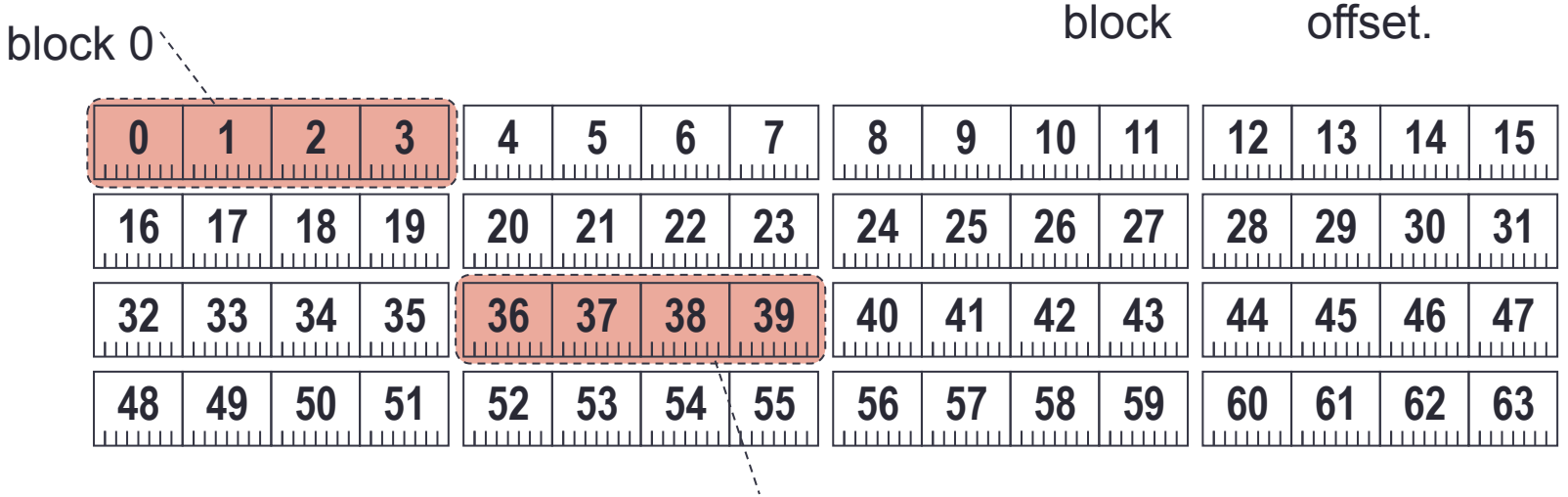



  - Note that a row contains $2^{c-k}$ blocks

# SDRAM operation: block access

- Example
  - Capacity 64x8 bits (W = 8 b)
  - 4 rows of 16 columns (words)
  - K = 4 (blocks of four 8-bit words)




- Memory components:
  - 16 x 8 sense amplifiers and bit latches
  - 2-bit row latch plus decoder 2:4
  - 4-bit column latch plus decoder 4:16

# SDRAM operation: block access

- READ/WRITE commands denote a single word address, but memory operates with the K words of the block containing that word
  - This is also called a burst access
- The addressed word goes first (*critical word first*), followed by the rest of the block's words

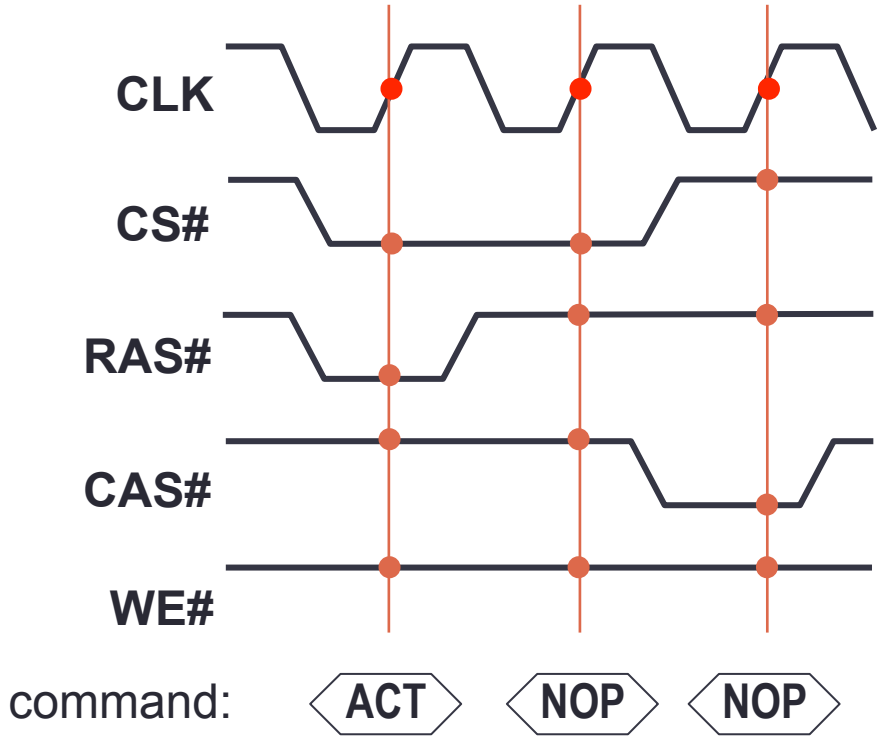
Example: $K = 4$


Block 0

| Address | Sequence |
|---|---|
| 0 | 0-1-2-3 |
| 1 | 1-2-3-0 |
| 2 | 2-3-0-1 |
| 3 | 3-0-1-2 |

# SDRAM operation: Commands

- Commands are accepted at the rising edges of Clock
- When CS# is high, the command is NOP (No OPeration)



# SDRAM operation: Commands

- Commands
  - A combination of CS#, RAS#, CAS#, WE# denote a command
  - The interpretation of address lines depends on the command

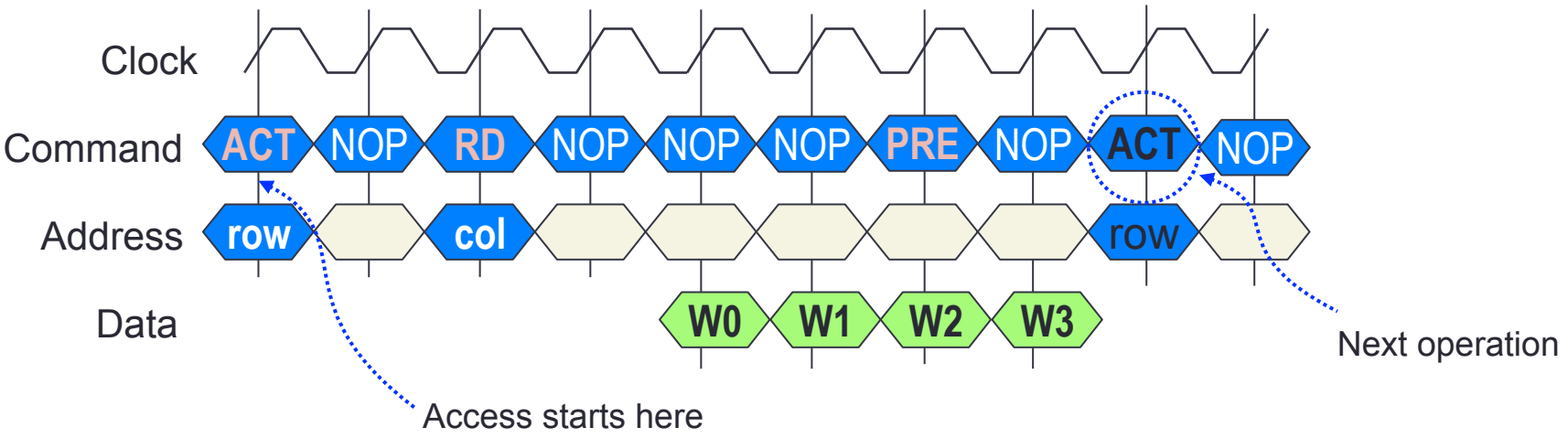
| Command name | RAS# | CAS# | WE# | Addr |
|---|---|---|---|---|
| NO OPERATION (NOP) | H | H | H | X |
| ACTIVATE | L | H | H | Row/Bk. |
| READ | H | L | H | Column/Bk. |
| WRITE | H | L | L | Column |
| PRECHARGE | L | H | L | Code |
| BURST TERMINATE | H | H | L | X |
| AUTO REFRESH | L | L | H | X |
| LOAD MODE REGISTER | L | L | L | Op-Code |

# SDRAM operation: Commands

- ACTIVATE
  - Latch and decode the row address using RAS# and activate the selected row. The row's cells are now connected to the bit lines
    - Sense amps detect and amplify the cells' charge and the whole row contents latched in the row bit latches
    - Access to several columns in the row is possible after a single ACTIVE command
- READ and WRITE commands
  - Latch and decode column address using CAS#. WE# then activates the I/O logic for reading or writing the selected column
- PRECHARGE
  - Prepares bit lines for next access and takes amps to stand-by. No further access is possible until a new ACTIVATE command

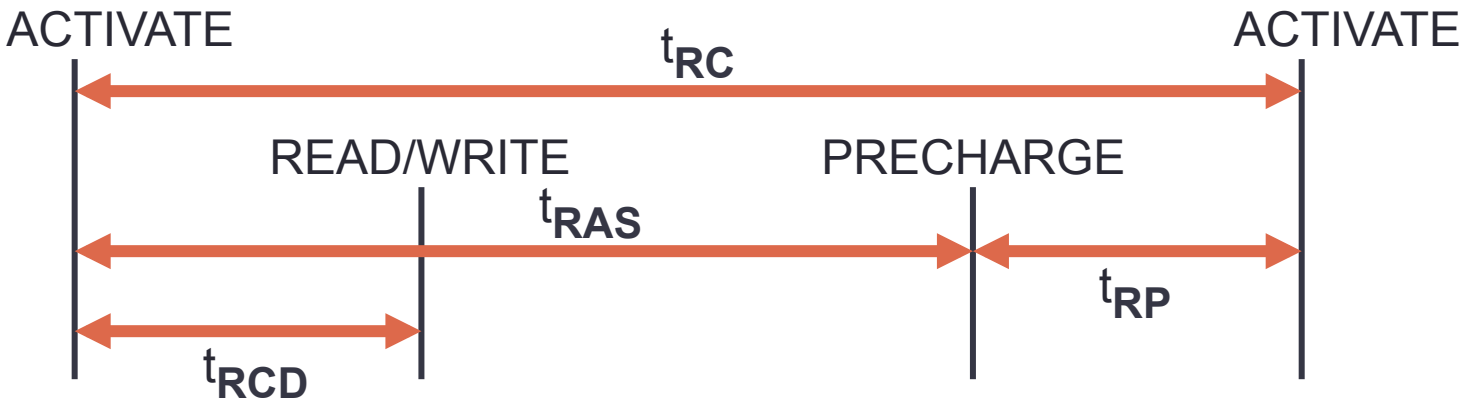
# SDRAM operation: block access

- Accessing a memory block requires a sequence of commands
  - ACTIVATE – READ – PRECHARGE
  - ACTIVATE – WRITE – PRECHARGE
  - with additional NOP commands for proper timing
- Example: read a 4-word block



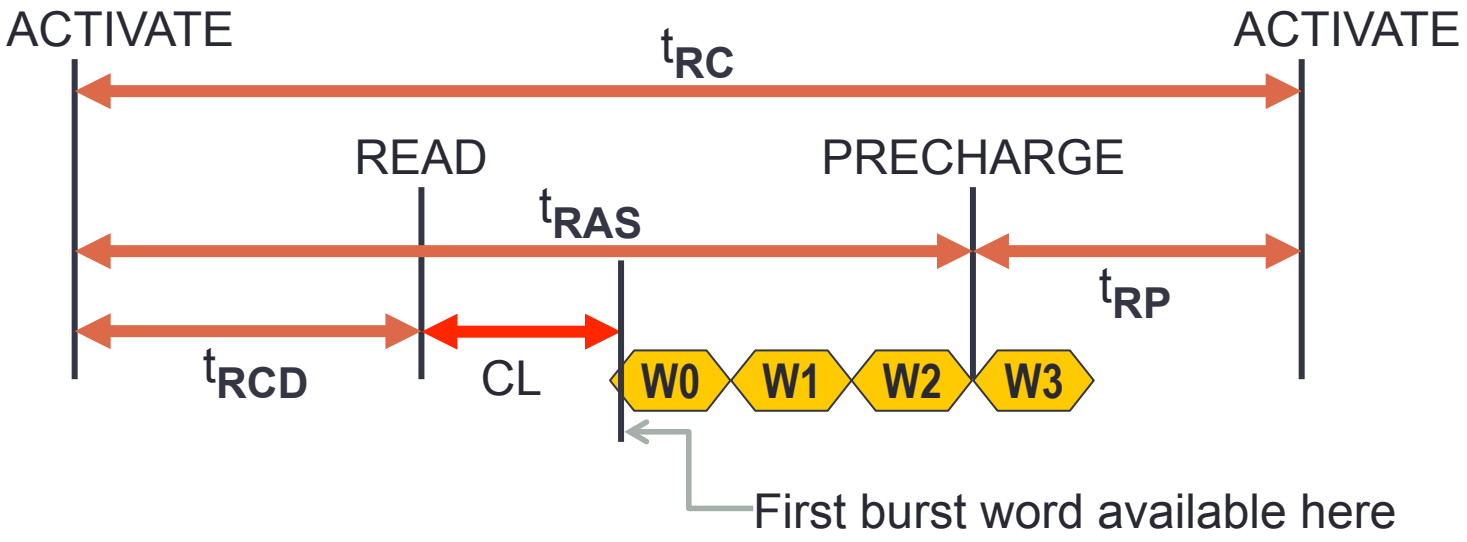
# SDRAM Timing

- Main timing parameters
  - $t_{RC}$: minimum time between two consecutive ACTIVATE commands
    - Equivalent to the memory cycle time
  - $t_{RCD}$: minimum time between ACTIVATE and READ or WRITE
  - $t_{RAS}$: minimum time between ACTIVATE and PRECHARGE
  - $t_{RP}$: minimum time between PRECHARGE and next ACTIVATE



# SDRAM Timing

- CAS Latency (CL)
  - In a READ operation, CL is the number of cycles by which the first burst item appears in the data outputs
    - CL does NOT apply to WRITE operations
  - Access time = $t_{RCD}$ + CL



# SDRAM Timing restrictions

- SDRAM chips specify their maximum working frequency (or minimum clock cycle)
  - The chip can be used at lower frequencies
- A proper number of NOP commands must be inserted to comply with timing requirements ($t_{RC}$, $t_{RAS}$, $t_{RP}$, $t_{RCD}$, CL)
- Example: number of cycles to accommodate timing requirements at different clock speeds ($t_{CK}$ = clock period)

| | 50 MHz ($t_{CK}$ = 20 ns) | 100 MHz ($t_{CK}$ = 10 ns) | 166 MHz ($t_{CK}$ = 6 ns) |
|---|---|---|---|
| $t_{RCD}$ = 18 ns | 1 | 2 | 3 |
| $t_{RAS}$ = 42 ns | 3 | 5 | 7 |
| $t_{RC}$ = 60 ns | 3 | 6 | 10 |
| $t_{RP}$ = 18 ns | 1 | 2 | 3 |

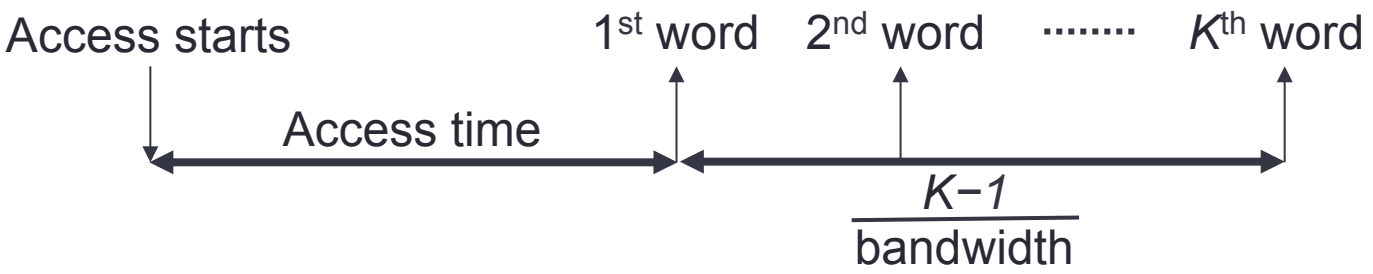
# SDRAM Timing

- Clock frequency and CAS latency
  - CL is programmed at memory startup
  - For a given clock frequency the chip has a minimum CL
    - Conversely, there is a maximum frequency for each CL

| PARAMETER | | SYMBOL | MIN | MAX | UNITS |
|---|---|---|---|---|---|
| Clock cycle time | CL = 3 | $^{t}$CK (3) | 6 | | ns |
| | CL = 2 | $^{t}$CK (2) | 10 | | ns |
| | CL = 1 | $^{t}$CK (1) | 20 | | ns |
| ACTIVE to PRECHARGE command | | $^{t}$RAS | 42 | 120k | ns |
| ACTIVE to ACTIVE command period | | $^{t}$RC | 60 | | ns |
| AUTO REFRESH period | | $^{t}$RFC | 60 | | ns |
| ACTIVE to READ or WRITE delay | | $^{t}$RCD | 18 | | ns |
| Refresh period (4,096 rows) | | $^{t}$REF | | 64 | ms |
| PRECHARGE command period | | $^{t}$RP | 18 | | ns |
| ACTIVE bank a to ACTIVE bank b command | | $^{t}$RRD | 12 | | ns |

# SDRAM Timing

- RD/WR operations always involve the *K* words in a burst
  - Once the row is open, column access is relatively fast
- Time to access a block in a burst has two components:
  - Latency to access the first word (independent of *K*)
  - Time to access subsequent words (depends on *K* and clock rate)
    - The chip's bandwidth ignores the first-word latency

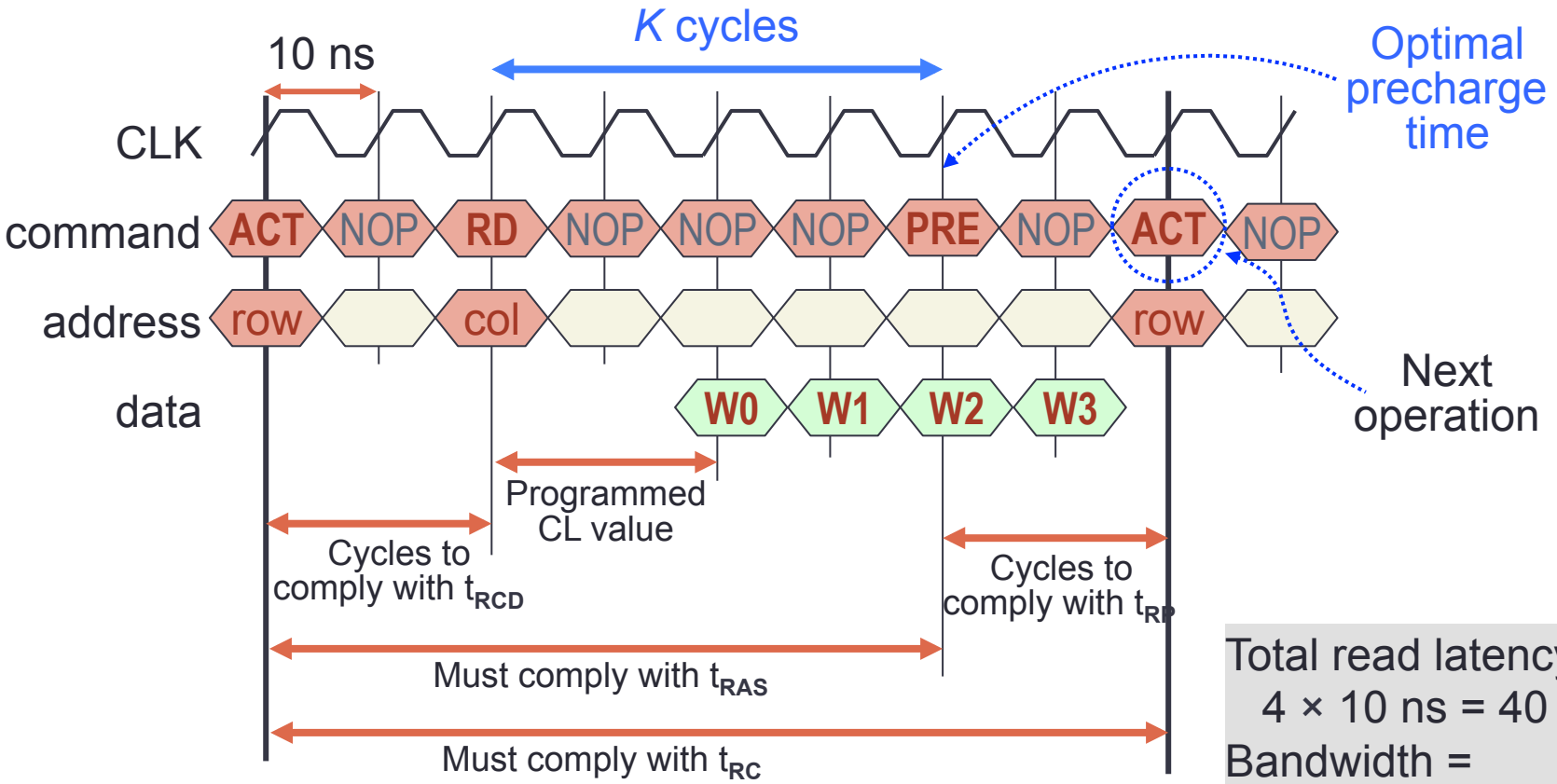


# SDRAM Timing: transfer timing

- First word of a READ appears CL cycles after command
- Write transfers start when WRITE is issued
- The remaining words in the burst are driven by the clock, one word per cycle
- $t_{RAS}$ imposes a minimum delay for issuing PRECHARGE, but it may need to be further delayed when the burst is large
  - A premature PRECHARGE command would abort the burst access
  - PRECHARGE is effective CL cycles after issuing. Optimal time for issuing PRECHARGE is CL – 1 cycles before reading the last word (equivalent to K cycles after READ, for a burst size of K)
- READ and WRITE can be issued in AUTO PRECHARGE mode
  - The chip is then in charge of self-issuing PRECHARGE at the optimum time, based on the programmed burst size, and saves the need for explicit precharge commands

# SDRAM Timing: Chronograms

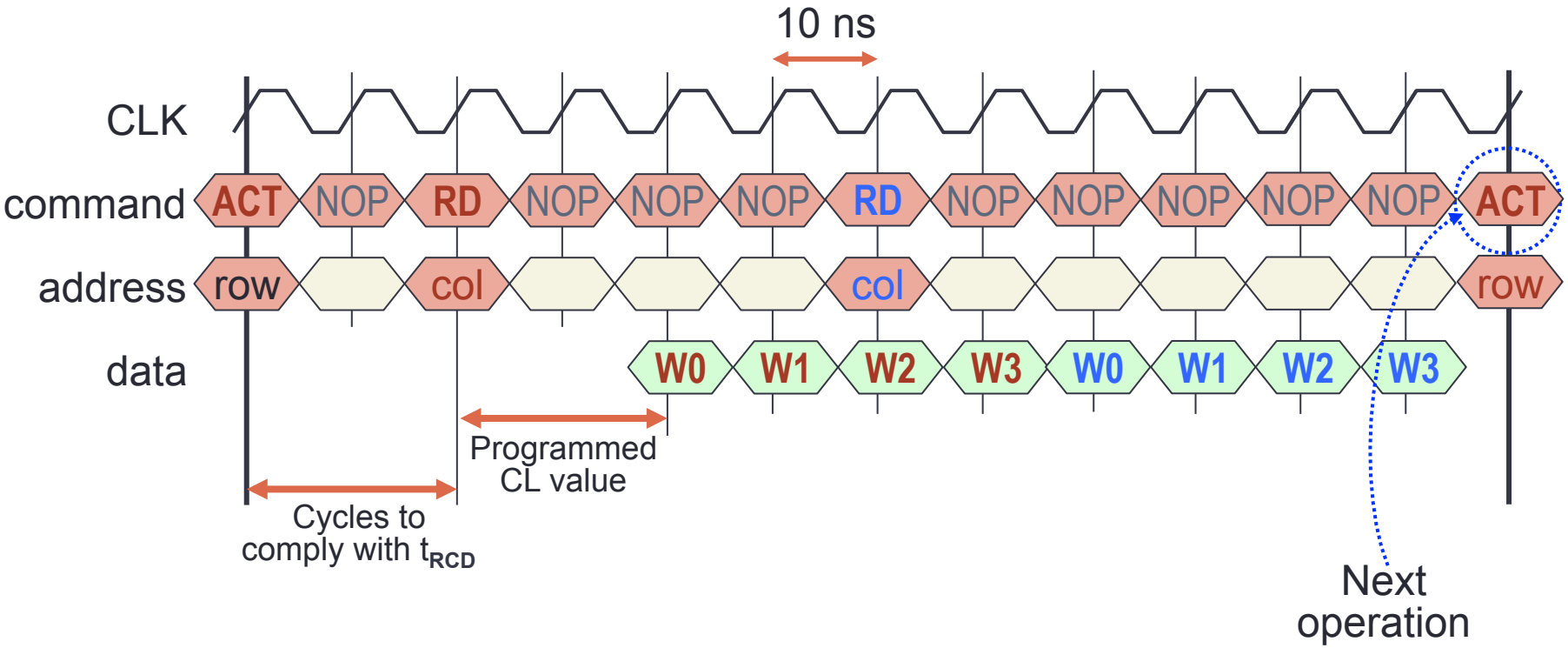
- Example: READ with $f_{CLK}$ = 100 MHz; CL = 2




Total read latency =
4 × 10 ns = 40 ns
Bandwidth =
100×10⁶ × w bits/s

# SDRAM Timing: Chronograms

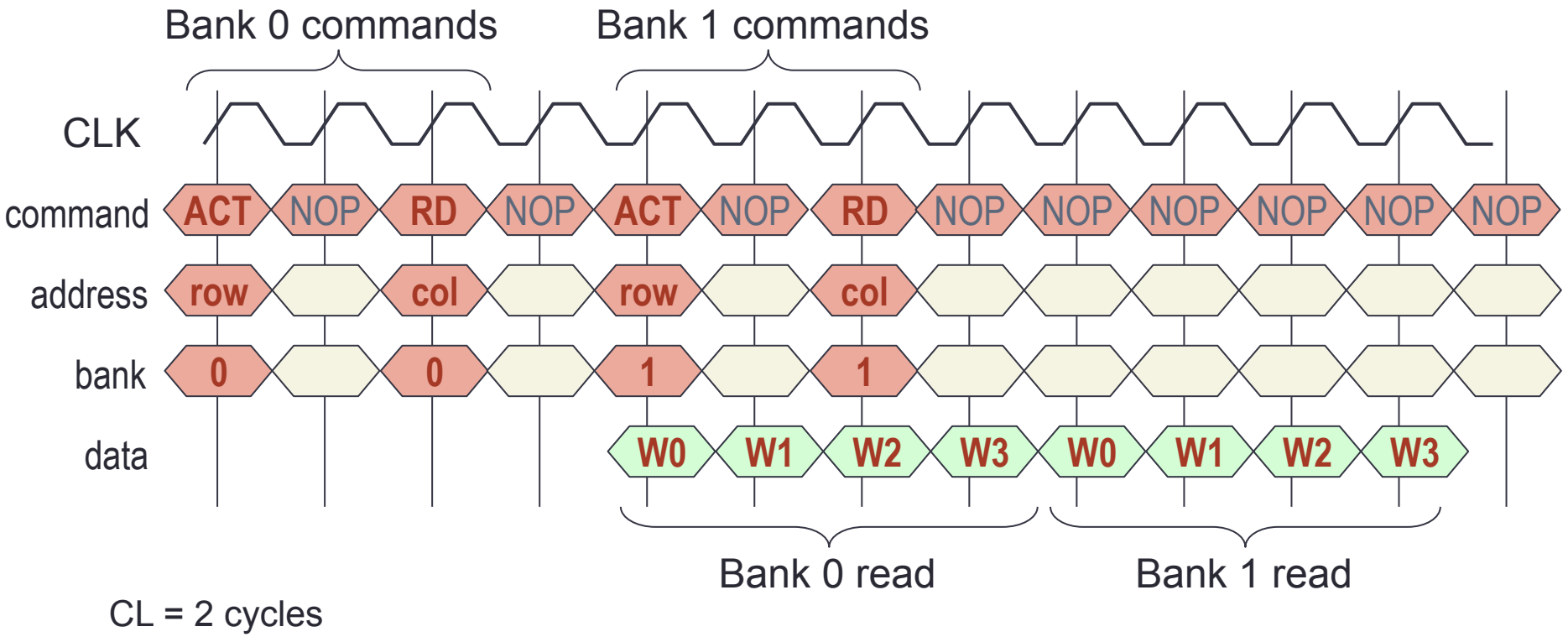
- Chaining two block reads from the same row (with auto-precharge)




Total read latency = 4 × 10 ns = 40 ns
Bandwidth = $100 \times 10^6 \times w$ bits/s

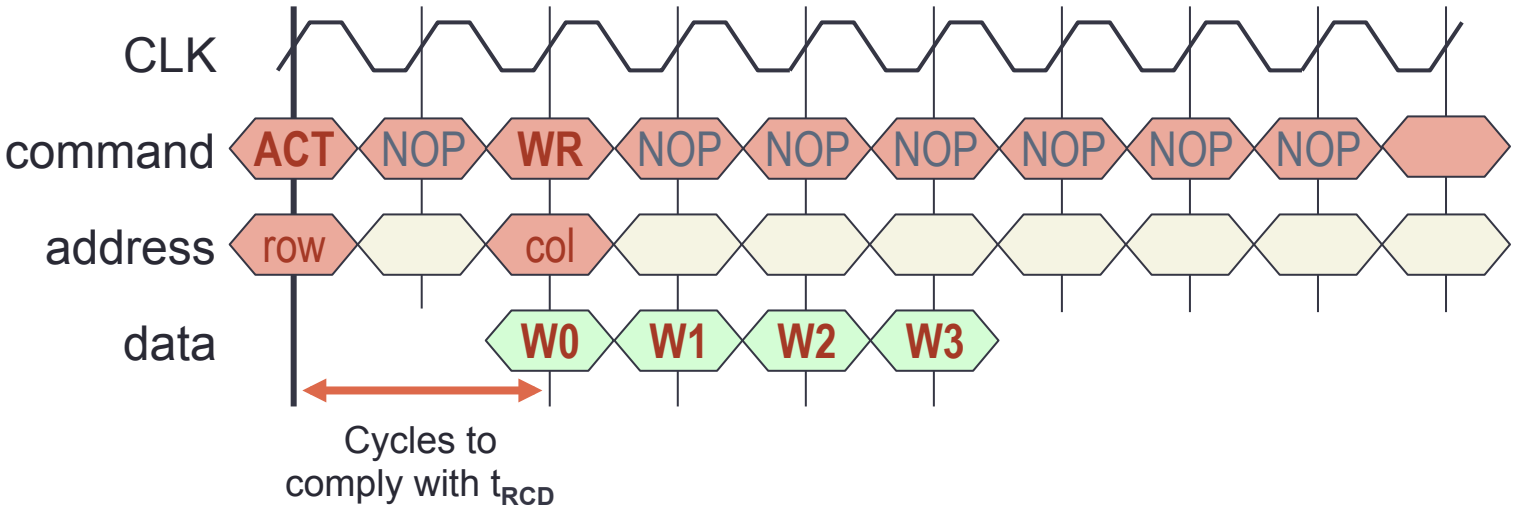
# SDRAM Timing: Chronograms

- Multiple bank READs
  - Independent banks may receive overlapping commands
    - Data are obtained in request order



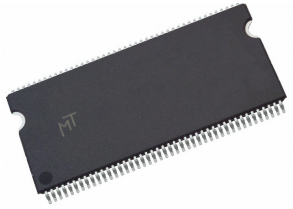
# SDRAM Timing: Chronograms

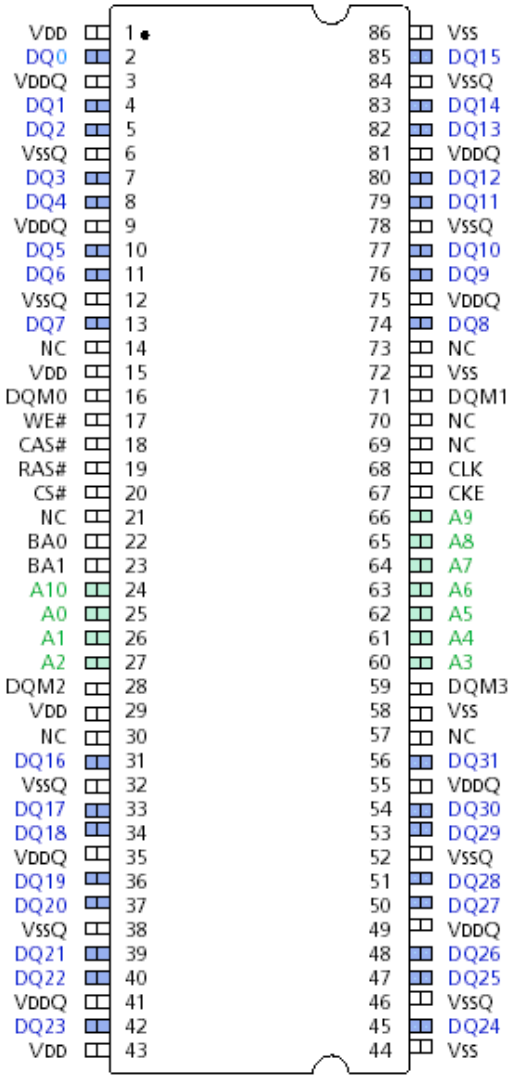
- WRITE cycle with $f_{CLK}$ = 100 MHz, auto-precharge mode



# A commercial example: 2M × 32 b

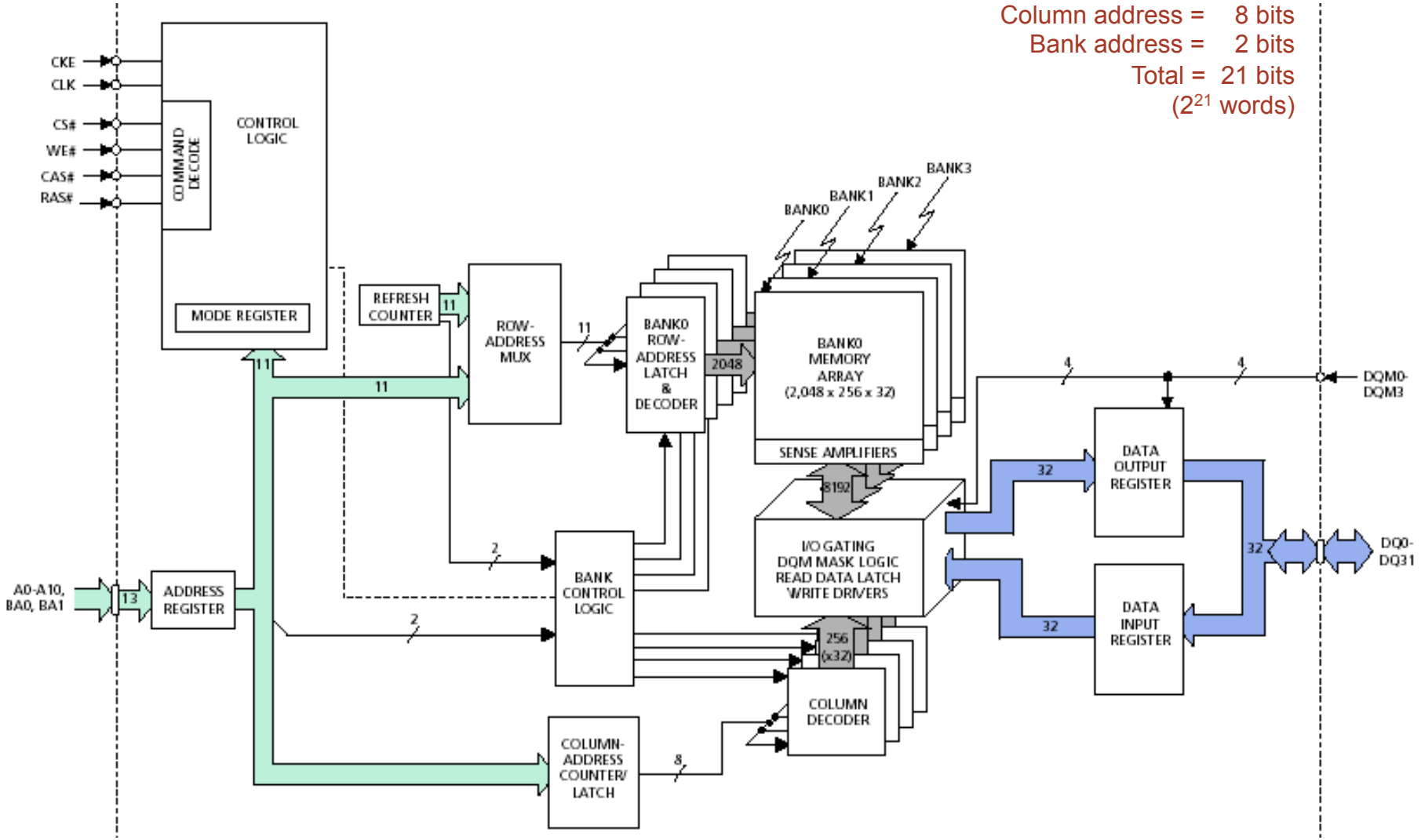
- Model MT49LC2M32B2 (micron)
- 86-pin plastic package
- Clock frequencies and cycle times
  - (143/166/183/200) MHz resp. (7/6/5.5/5) ns
- Programmable CAS latency
  - 143 and 166 MHz → 1, 2 or 3 cycles
  - 183 and 200 MHz → 3 cycles
- Programmable burst length
  - 1, 2, 4, 8 or 256 32-bit words
- Four banks of 2048×256×32 b
  - 2 M×32 b = $2^{26}$ = 4×$2^{24}$ = 4×($2^{11}$×$2^{8}$×32) b
- Refresh period: 64 ms





# Block Diagram

Row address = 11 bits
Column address = 8 bits
Bank address = 2 bits
Total = 21 bits
($2^{21}$ words)



# Improvements to SDRAM technology

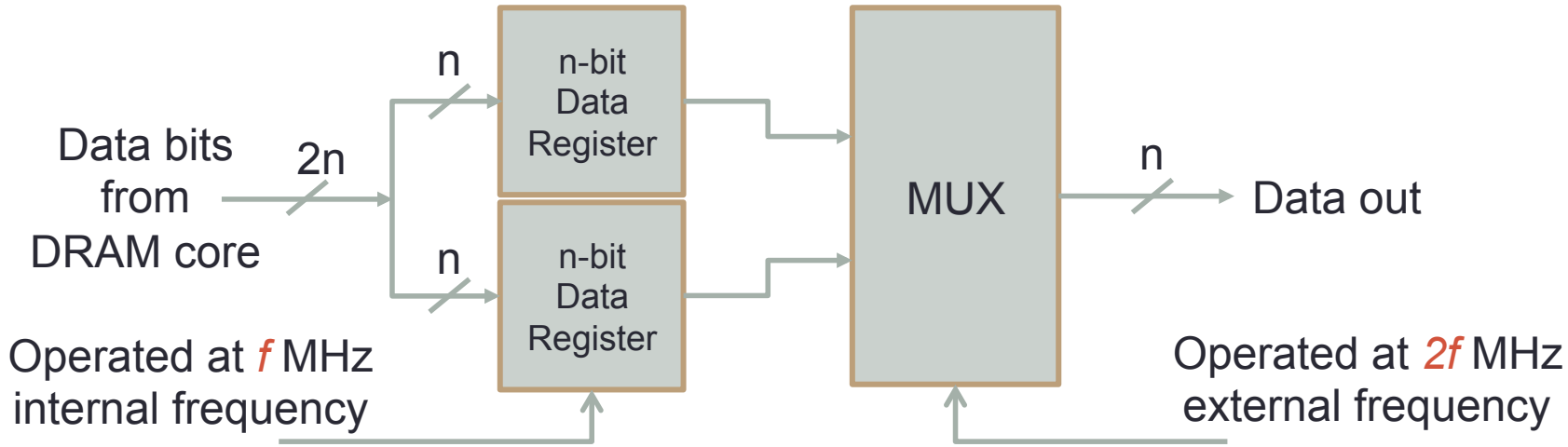
- Memory implementation trends:
  - Higher density
  - Lower power voltage
    - Heat is proportional to $V^2$ and frequency, and it limits density
  - Higher bandwidth based on structural improvements (DDR1/2/3/4)
    - Latencies to access the internal matrix diminish very slowly with time

# Improvements to SDRAM: DDR

- DDR stands for *Dual Data Rate*
  - The JEDEC (Joint Electron Device Engineering Council) maintains standard specification of SDRAM chips
- In DDR, data can be read/written at twice the memory's internal frequency, effectively achieving double bandwidth with respect to SDR SDRAM (Single Data Rate SDRAM)
- DDR is based on the *2n-prefetch* technique
  - A single access to the cell matrix provides 2n bits for an n-bit chip
  - The internal data bus is twice the size of the external
  - Data is externally accessed at a rate of 2 words per cycle
- DDR memories operate with two clocks: one internal for accessing the cell matrix and one external for data transferring

# Improvements to SDRAM: DDR

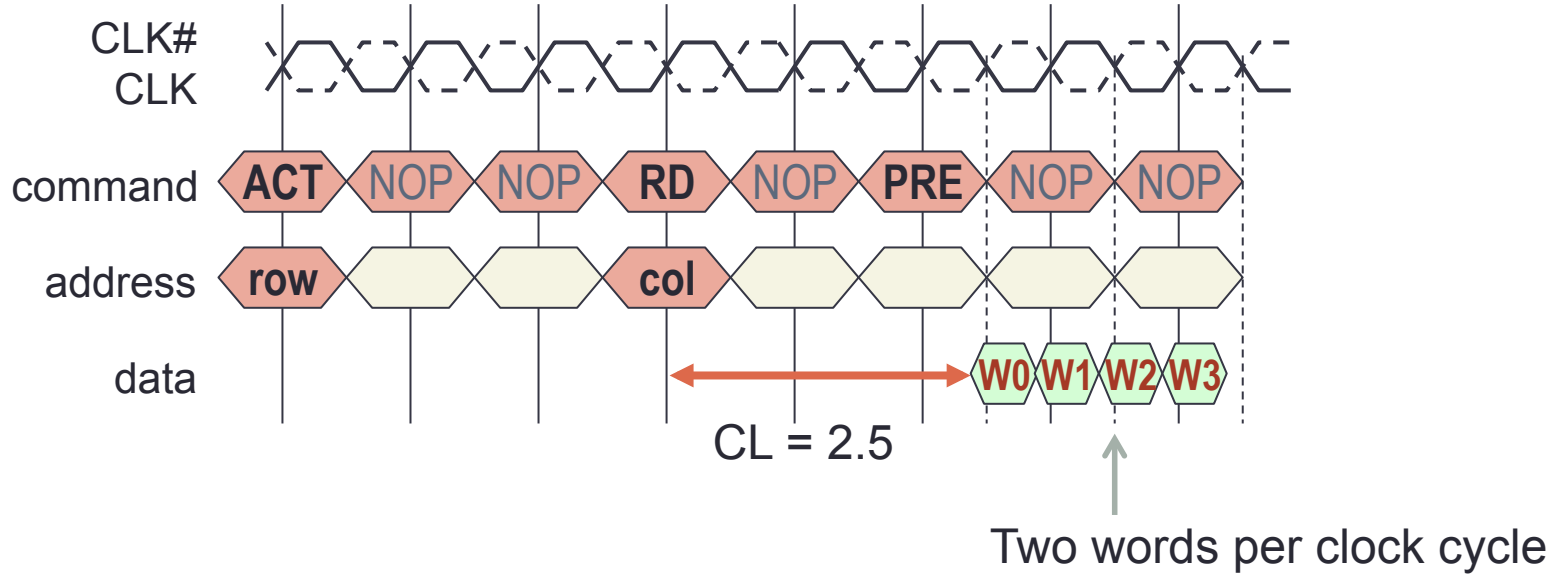
- Simplified block diagram for 2n-prefetch READ




- DDR2 and DDR3 use *4n*- and *8n-prefetch* for accessing larger bursts at higher frequencies, for larger bandwidth
- DDR4 further reduces power consumption and increases data rate by grouping banks, but still using *8n-prefetch*

# Improvements to SDRAM: DDR

- DDR READ chronogram
  - CAS latency is a multiple of 0.5 clock cycles
  - The external interface includes a CLK# input



# SDRAM Standard Naming

- According to the JEDEC standard, SDRAM chips are named after their achievable bandwidth
  - SDR (non-DDR SDRAM): **PC F**
    - F is the chip's clock frequency, hence a direct indication of bandwidth when multiplied by the chip's number of data pins
  - DDR: **DDRX MTps**
    - X can be "", "2", "3" or "4" for DDR, DDR2, DDR3 and DDR4, resp.
    - MTPs (or MT/s) stands for Mega (Millions of) Transactions per Second
    - It is a measure of data pin bandwidth – chip's bandwidth depends on MTps and the number of data pins
  - Examples:
    - Micron's MT46V32M8-5B is a 32M×8b DDR-400
      - With a 200 MHz clock, it can achieve up to 400 MTps
      - Bandwidth of this 8-bit wide chip is thus: 400 MT/s × 1 B/T = 400 MBps
    - Micron's MT41K64M16-125 is a 64M×16b DDR3-1600
      - 800 MHz clock, 1600 MTps, Bandwidth = 1600 MT/s × 2 B/T = 3200 MBps

# SDRAM Standard Naming

- Variations of SDRAM chips

| Name | Internal clock (MHz) | Prefetch (min burst) | External clock (MHz) | Transfer rate (MT/s) | Voltage (V) |
|---|---|---|---|---|---|
| SDR | 66-133 | *1n* | 66-133 | 66-133 | 3.3 |
| DDR | 100-200 | 2n | 100-200 | 200-400 | 2.5-2.6 |
| DDR2 | 100-267 | 4n | 200-533 | 400-1067 | 1.8 |
| DDR3 | 100-267 | 8n | 400-1066 | 800-2133 | 1.35-1.5 |
| DDR4 | 133-267 | 8n | 1066-2133 | 2133-4267 | 1.05-1.2 |

(*) Chips exist that deviate from these specifications and limits

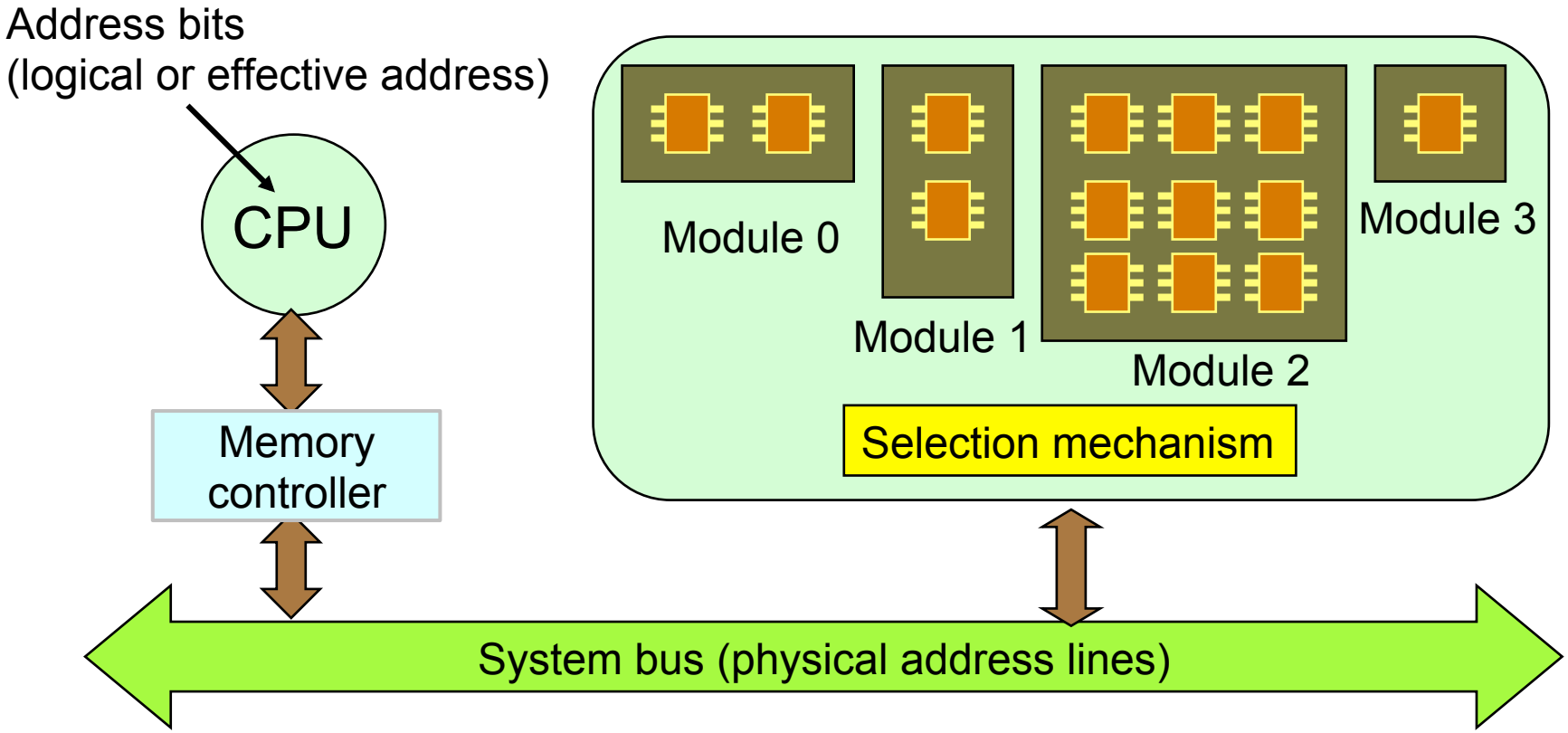
# 5. Memory Modules

- Basic concepts
- Internal organization
- Commercial DRAM modules

# Basic concepts

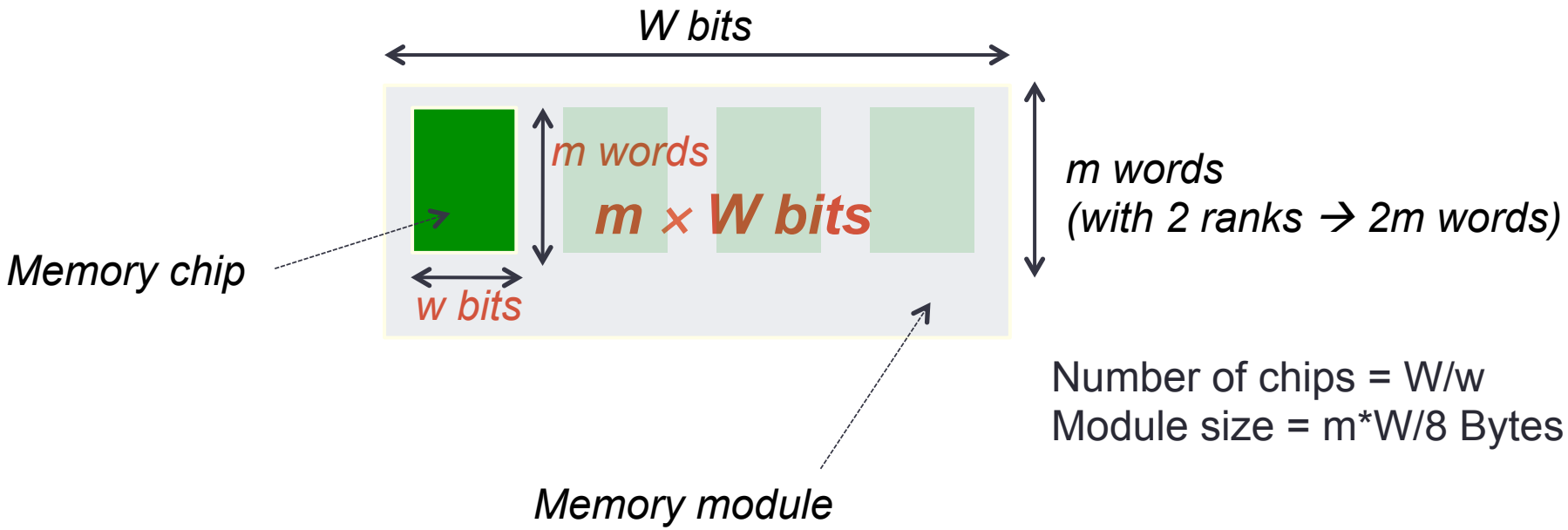
- The main memory system is typically composed of several memory modules
  - A memory module is a circuit composed by memory chips that are grouped to provide an adequate number and size of words
- Each module contains a particular type of memory chips (ROM, SRAM, DRAM, etc.)
- Memory modules tend to comply with a standard specification (see later)
- In principle, only one module may be selected at a time
  - Each module supports a range of addresses
  - A selection mechanism takes care of proper module selection
  - The distribution of addresses per module is called memory map (see later)

# Basic concepts

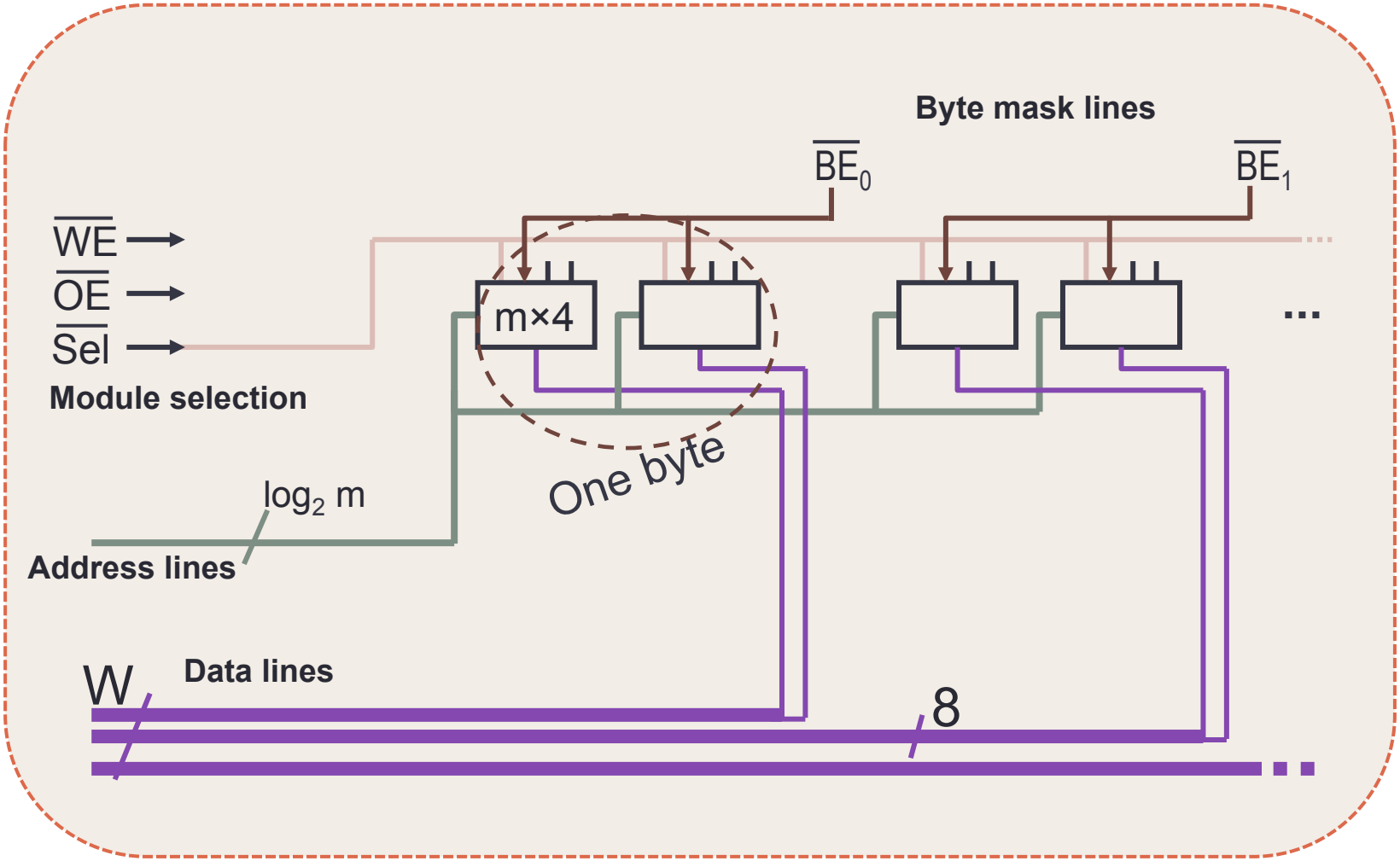


# Internal organisation of modules

- For simplicity, modules are built with identical chips
- Memory chips are typically arranged as one or two rows (called *ranks*)
  - More ranks → more words; more columns → wider words
  - It is convenient to express capacities as *words* × *word_size*

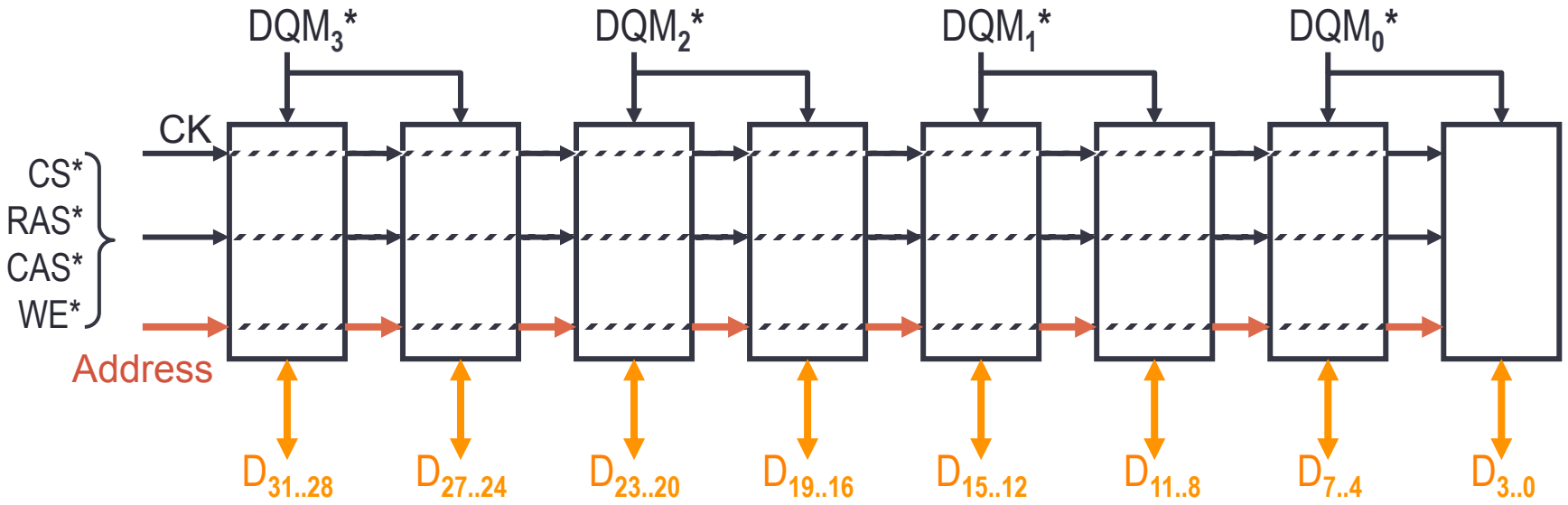


# Internal organisation of modules



# Internal organisation of modules

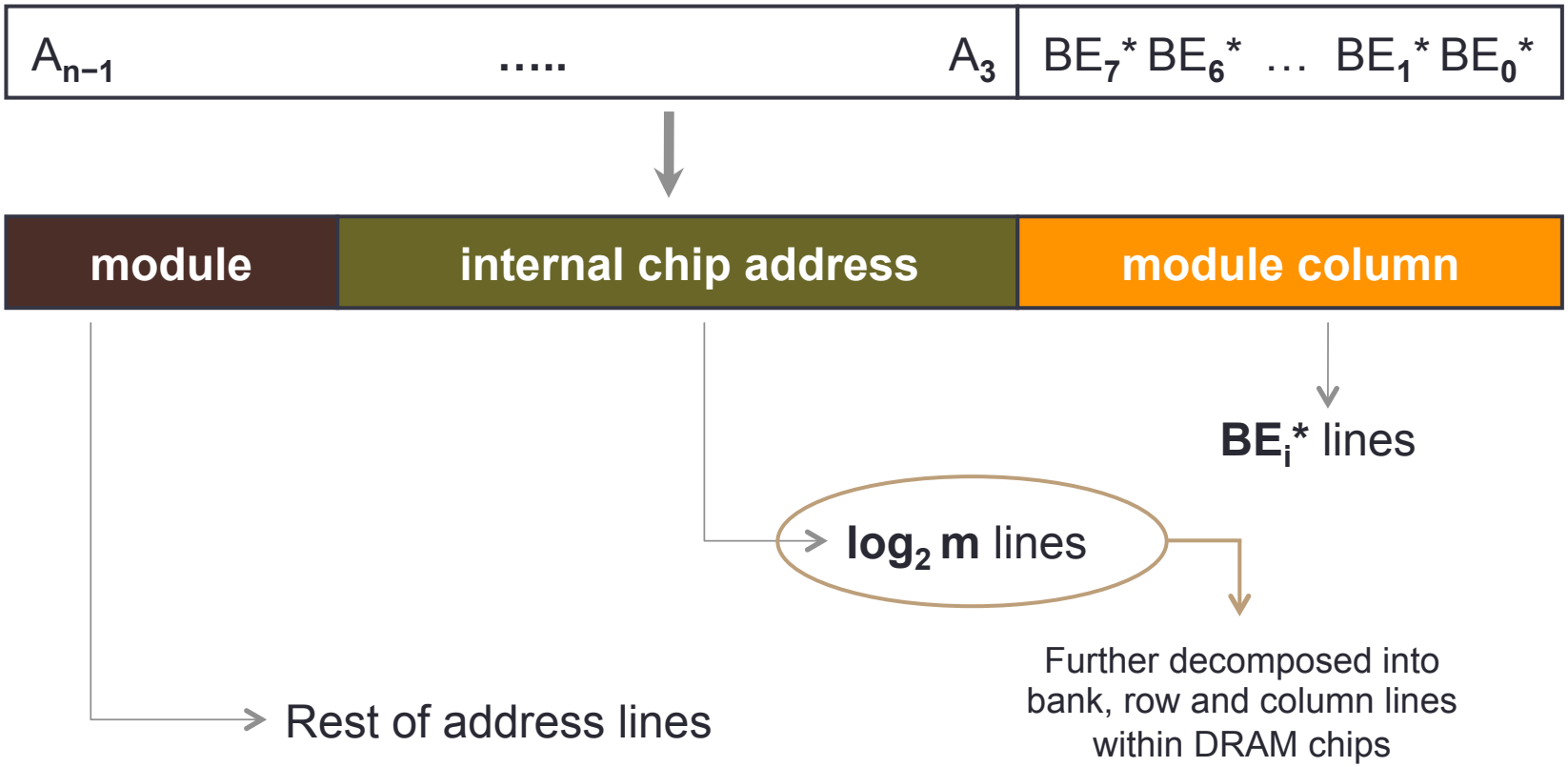
- Connections
  - CK, selection, command and address lines are common to all chips
  - Data mask lines enable selection of individual bytes
  - Total bandwidth equals the sum of chips' bandwidth
- Example: module of m×32 b based on m×4 chips



# Internal organisation of modules

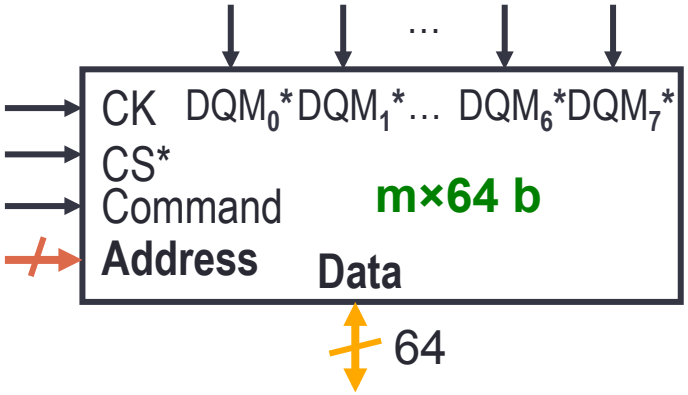
- Address structure with modules

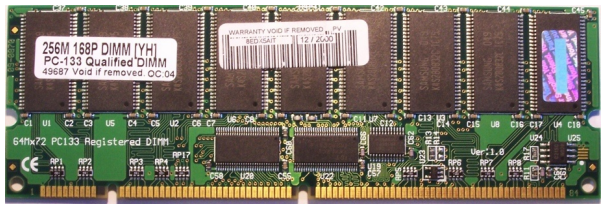
Physical address bus (e.g., 64-bit data bus)



# DRAM Modules

- Modules are inserted into motherboard's sockets
- Some modules include one additional parity bit per byte for error detection
- Example: m×64 b



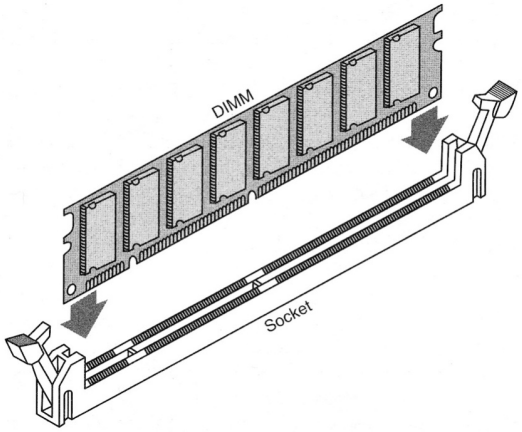



# Standard DRAM Modules

- A standard definition facilitates module interchange
  - Modules easily fit standard slots in the motherboard
  - Optionally, modules include one additional parity bit per byte to enable error detection
  - Examples

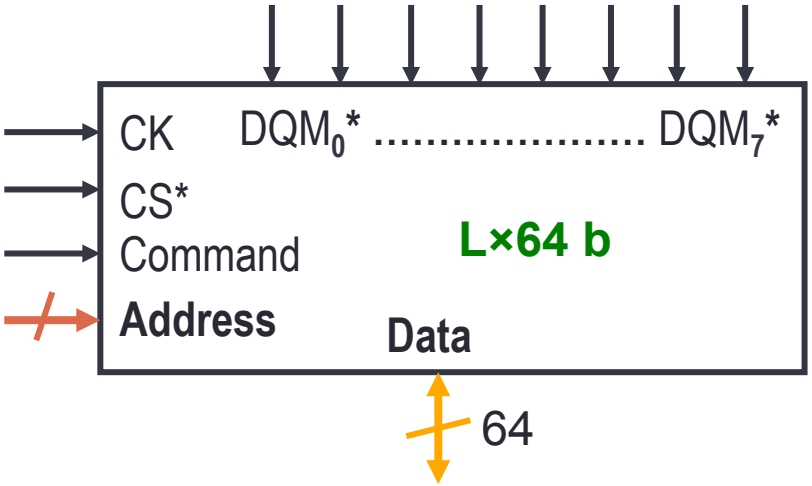
| Format | Word size | Chip type | Capacity |
|---|---|---|---|
| DIMM 168 contacts | 64/72 bits | EDO, SDRAM | 4/.../512 MB |
| SODIMM 144 c. | 64 bits | SDRAM | 8/.../2 GB |

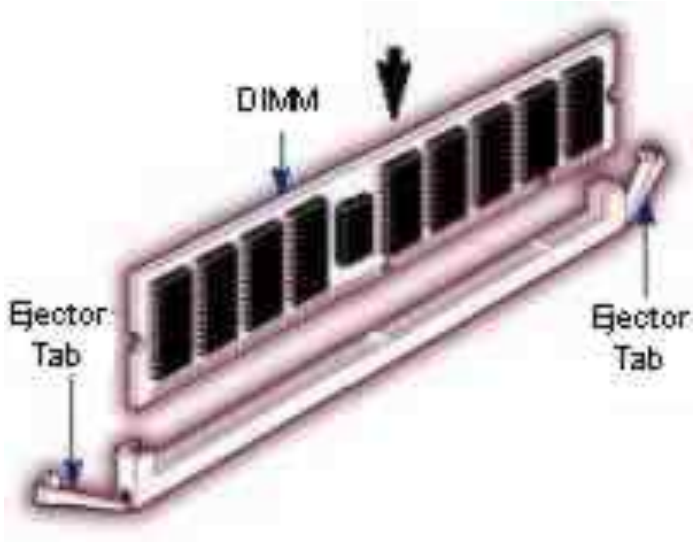
Note: SODIMM = *Small Outline* DIMM (used for portable computers)

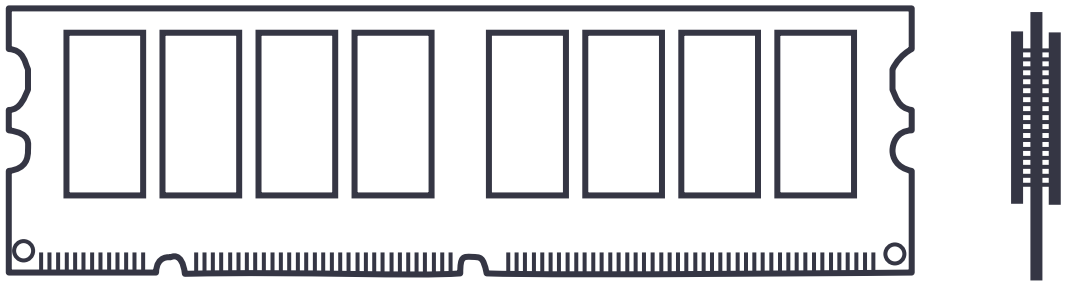


# DIMM Modules

- DIMM (Dual Inline Memory Module)
  - 168/184/240/288 contacts, 64(72) data bits, 5¼" long (13.335 cm)

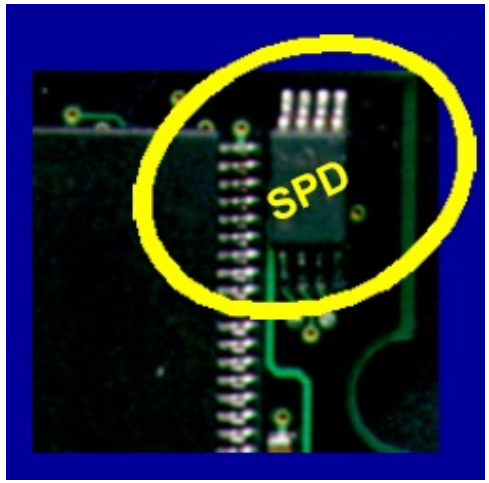


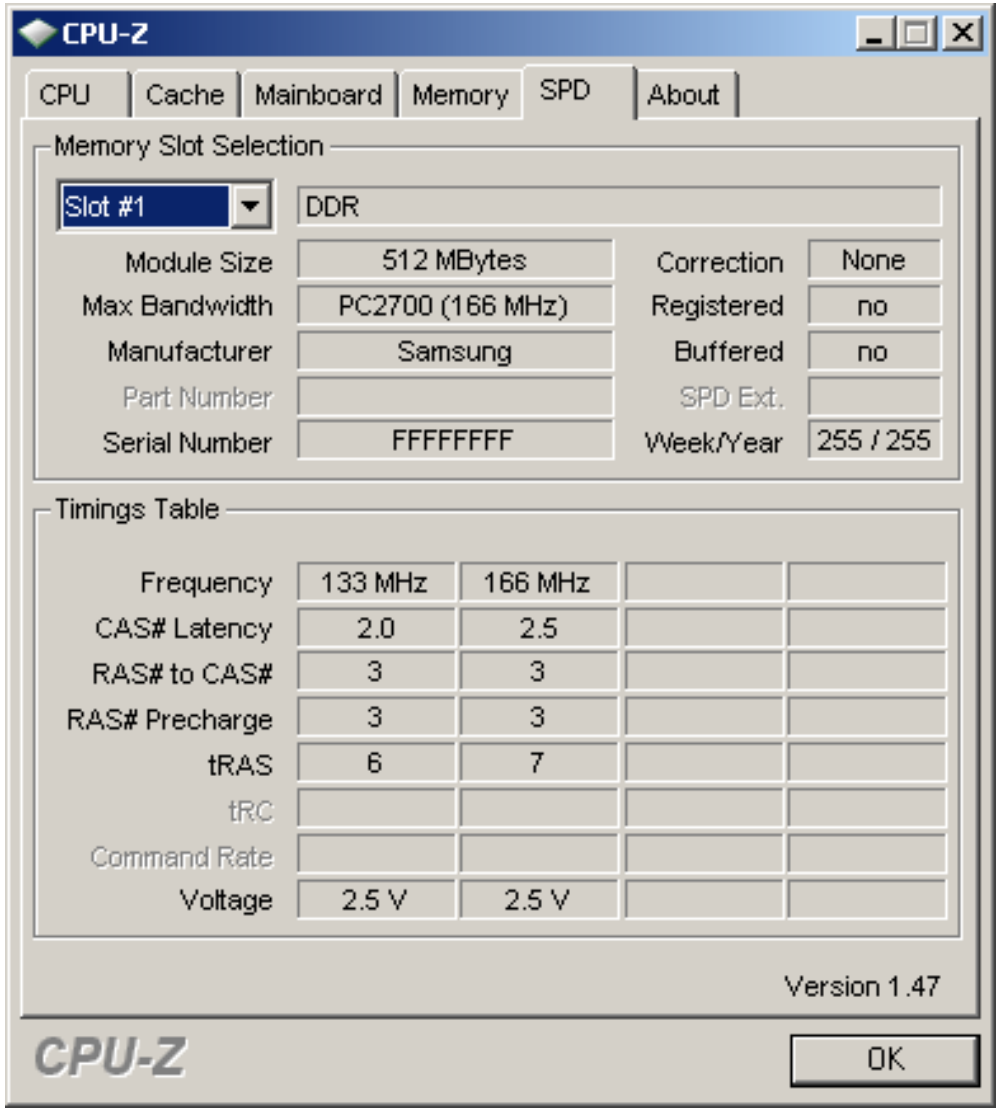



# SPD: Serial Presence Detect

- EEPROM memory chip
- Stores a few bytes of relevant information about the module
  - Timing
  - Capacity
  - Manufacturer
  - Serial number
- Facilitates automatic configuration of the main memory system
- Some programs may access the information contained in the SPD
  - CPU-Z, SiSoftware Sandra, …



# SPD: View from CPU-Z



# Comparative and denominations

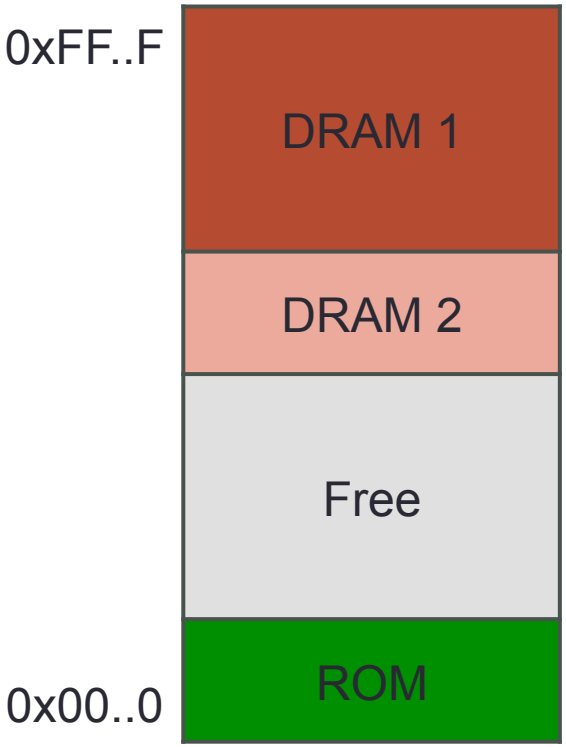
- Packaging
  - SDRAM: DIMM 168 c
  - DDR SDRAM: DIMM 184 c
  - DDR2 SDRAM: DIMM 240 c
  - DDR3 SDRAM: DIMM 240 c
- Bandwidth and naming
  - SDRAM @ 100 MHz → 100 MHz × 8 B = 800 MB/s
    - Denomination: PC100
  - DDR SDRAM @ 100 MHz → 100 MHz × 8 B × 2 = 1600 MB/s
    - Denomination: PC 1600
  - DDR2 SDRAM @ 200 MHz → 200 MHz × 8 B × 2 = 3200 MB/s
    - Denomination: PC2-3200
  - DDR3 SDRAM @ 400 MHz → 400 MHz × 8 B × 2 = 6400 MB/s
    - Denomination: PC3-6400

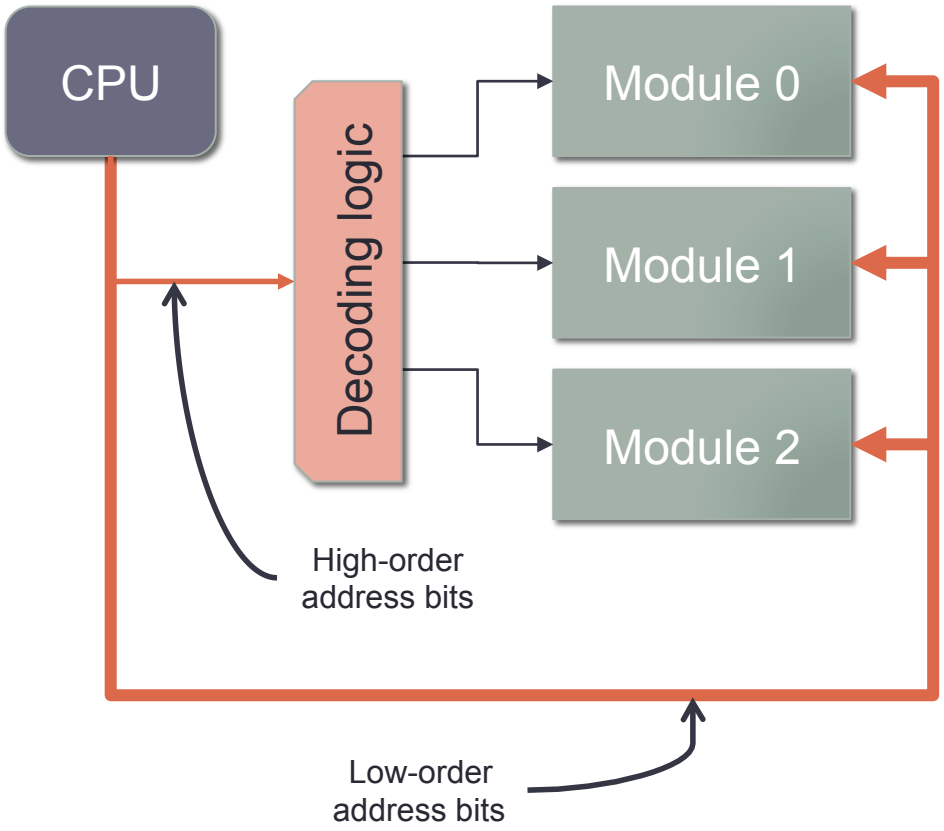
# 6. Memory Map

- Concept
- Map decoding

# Concept of memory map

- The memory map is the distribution of modules across the addressing space



# Map decoding

- To implement the map decoding logic we need to know:
  - CPU addressing space, i.e., nr. of lines in the physical address bus
    - In fact, we only need to know what's the highest address bit
  - Modules' initial addresses
  - Modules' sizes (in bytes)
    - The internal organization of the module doesn't matter
  - Logic (positive or negative) of module select functions

# A simple example

- CPU with addressing space of 256 KB, W = 32 b
- 64 KB DRAM module at address 0x20000
- We are asked to design the logic to implement this map

- First, see what addresses are used by the module

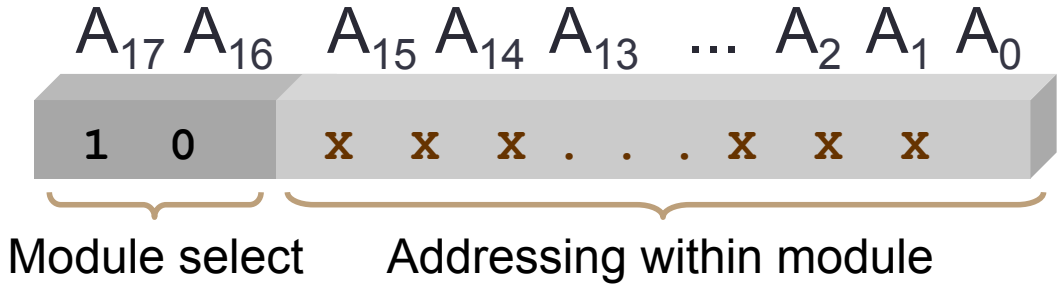
| $A_{17}$ | $A_{16}$ | $A_{15}$ | $A_{14}$ | $A_{13}$ | . . . | $A_2$ | $A_1$ | $A_0$ | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | **0** | 0 | 0 | 0 | . . . | 0 | 0 | 0 | **0x20000** |
| **1** | **0** | 0 | 0 | 0 | . . . | 0 | 0 | 1 | **0x20001** |
| **1** | **0** | 0 | 0 | 0 | . . . | 0 | 1 | 0 | **0x20002** |
| | | | | | . . . | | | | |
| **1** | **0** | 1 | 1 | 1 | . . . | 1 | 1 | 0 | **0x2FFFE** |
| **1** | **0** | 1 | 1 | 1 | . . . | 1 | 1 | 1 | **0x2FFFF** |

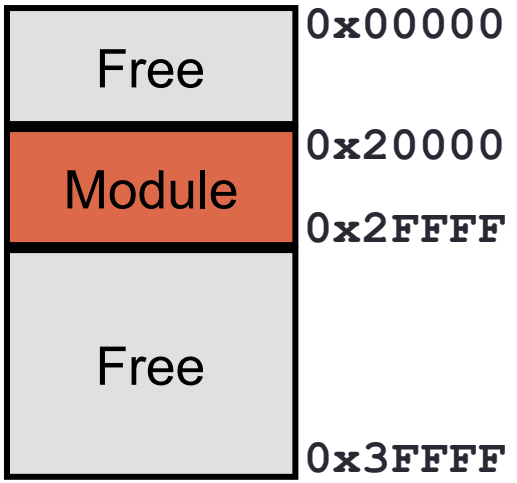
- All addresses in the module have in common $A_{17}$=1 and $A_{16}$=0

# A simple example

- CPU with addressing space of 256 KB, W = 32 b
- 64 KB DRAM module at address 0x20000
- We are asked to design the logic to implement this map

- Second, use the regularity in module's addresses to determine the SEL function
  - All addresses in the module have the form **0x2XXXX**



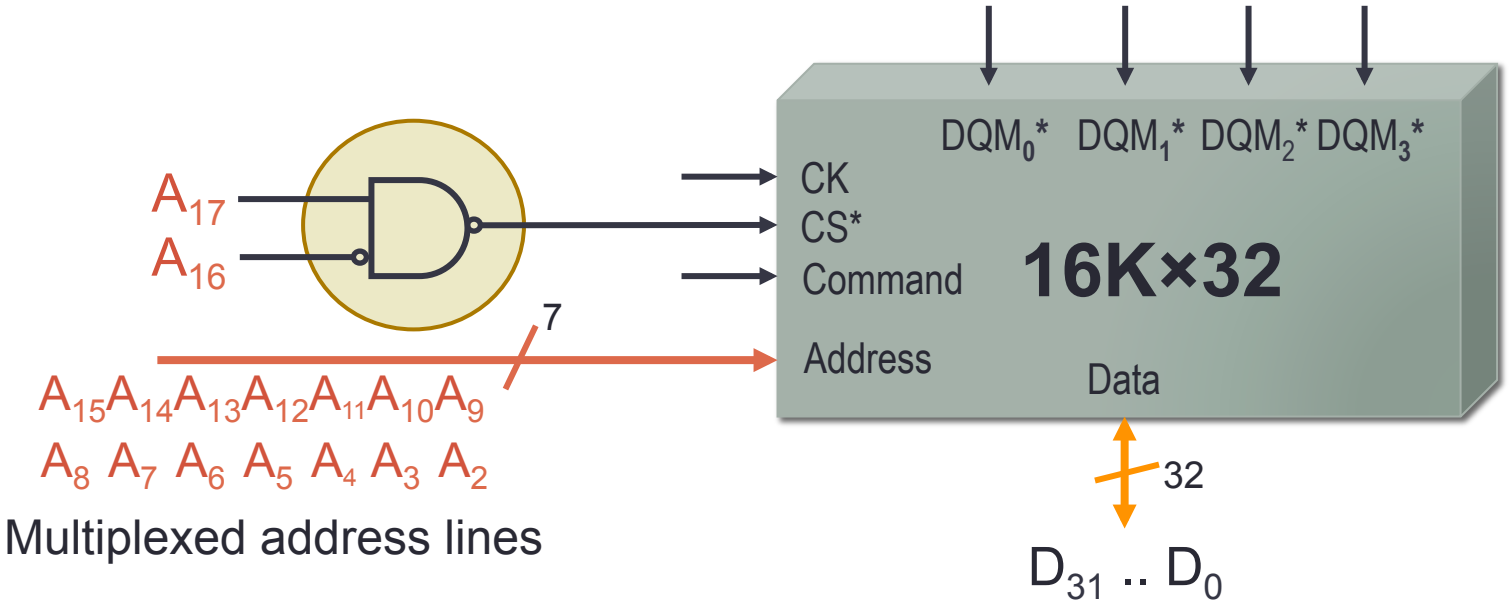

- Third, the functions

$$SEL = A_{17} \cdot A_{16}^*$$
$$SEL^* = A_{17}^* + A_{16}$$

# A simple example

- CPU with addressing space of 256 KB, W = 32 b
- 64 KB DRAM module at address 0x20000
- We are asked to design the logic to implement this map

- Finally, consider the physical lines

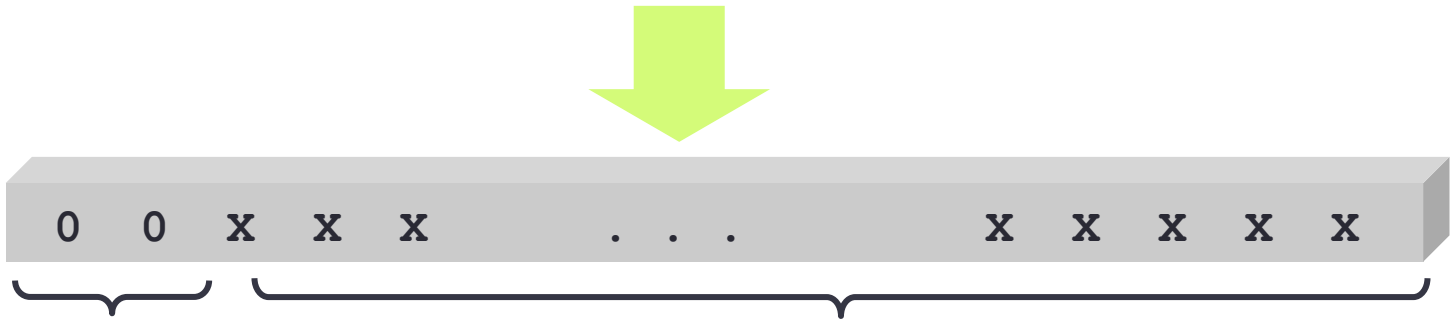


# Example with two modules

- MIPS R2000: 4 GB addressing ($2^{32}$ B)
  - Module RAM1: 1 GB starting at 0x00000000
  - Module RAM2: 2 GB starting at 0x80000000

# Example with two modules: RAM1

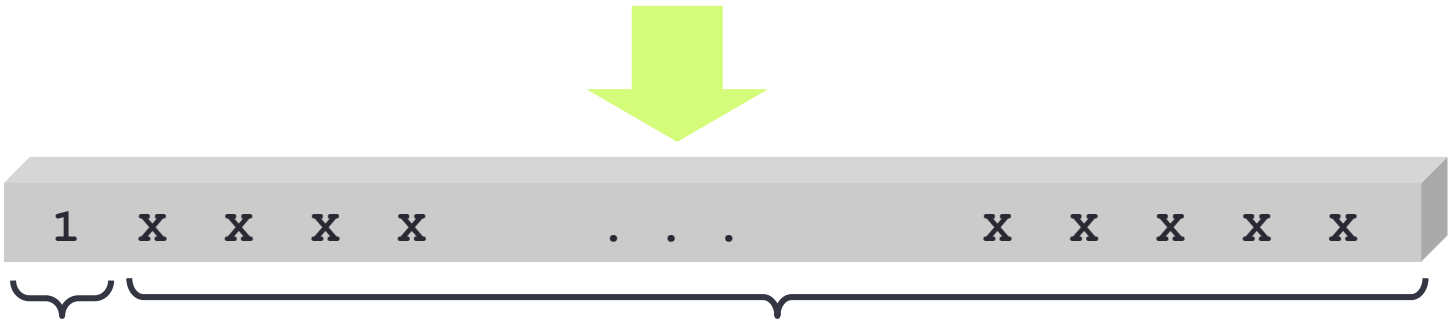
| | $A_{31}$ | $A_{30}$ | $A_{29}$ | $A_{28}$ | $A_{27}$ | . . . | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x00000000 | 0 | 0 | 0 | 0 | 0 | . . . | 0 | 0 | 0 | 0 | 0 |
| 0x00000001 | 0 | 0 | 0 | 0 | 0 | . . . | 0 | 0 | 0 | 0 | 1 |
| 0x00000002 | 0 | 0 | 0 | 0 | 0 | . . . | 0 | 0 | 0 | 1 | 0 |
| • • • | | | | | | | | | | | |
| 0x3FFFFFFE | 0 | 0 | 1 | 1 | 1 | . . . | 1 | 1 | 1 | 1 | 0 |
| 0x3FFFFFFF | 0 | 0 | 1 | 1 | 1 | . . . | 1 | 1 | 1 | 1 | 1 |




$SEL_{RAM1}^{*} = A_{31} + A_{30}$

# Example with two modules: RAM2

| | $A_{31}$ | $A_{30}$ | $A_{29}$ | $A_{28}$ | $A_{27}$ | . . . | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x80000000 | 1 | 0 | 0 | 0 | 0 | . . . | 0 | 0 | 0 | 0 | 0 |
| 0x80000001 | 1 | 0 | 0 | 0 | 0 | . . . | 0 | 0 | 0 | 0 | 1 |
| 0x80000002 | 1 | 0 | 0 | 0 | 0 | . . . | 0 | 0 | 0 | 1 | 0 |
| ••• | | | | | | | | | | | |
| 0xFFFFFFFE | 1 | 1 | 1 | 1 | 1 | . . . | 1 | 1 | 1 | 1 | 0 |
| 0xFFFFFFFF | 1 | 1 | 1 | 1 | 1 | . . . | 1 | 1 | 1 | 1 | 1 |




$SEL_{RAM2}^* = A_{31}^*$

# 7. The Memory Controller

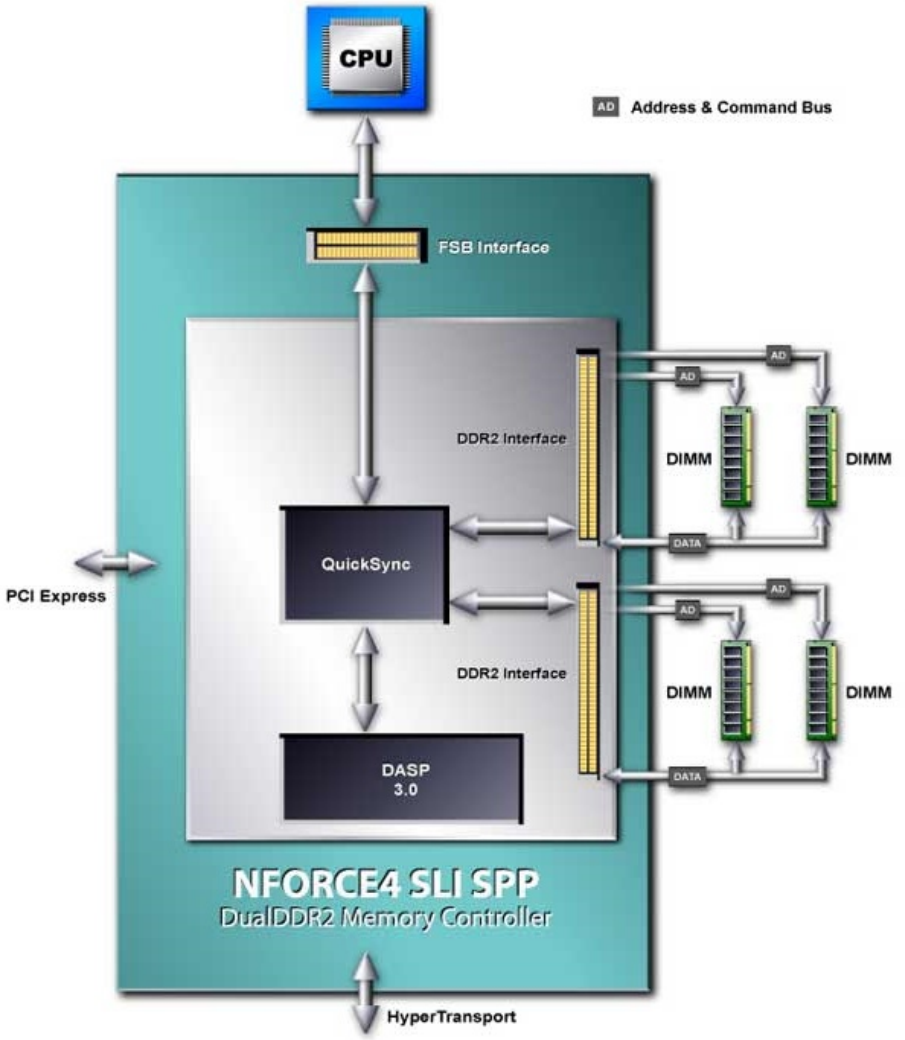
- Description and functions
- Initialization of the memory system

# Description and functions

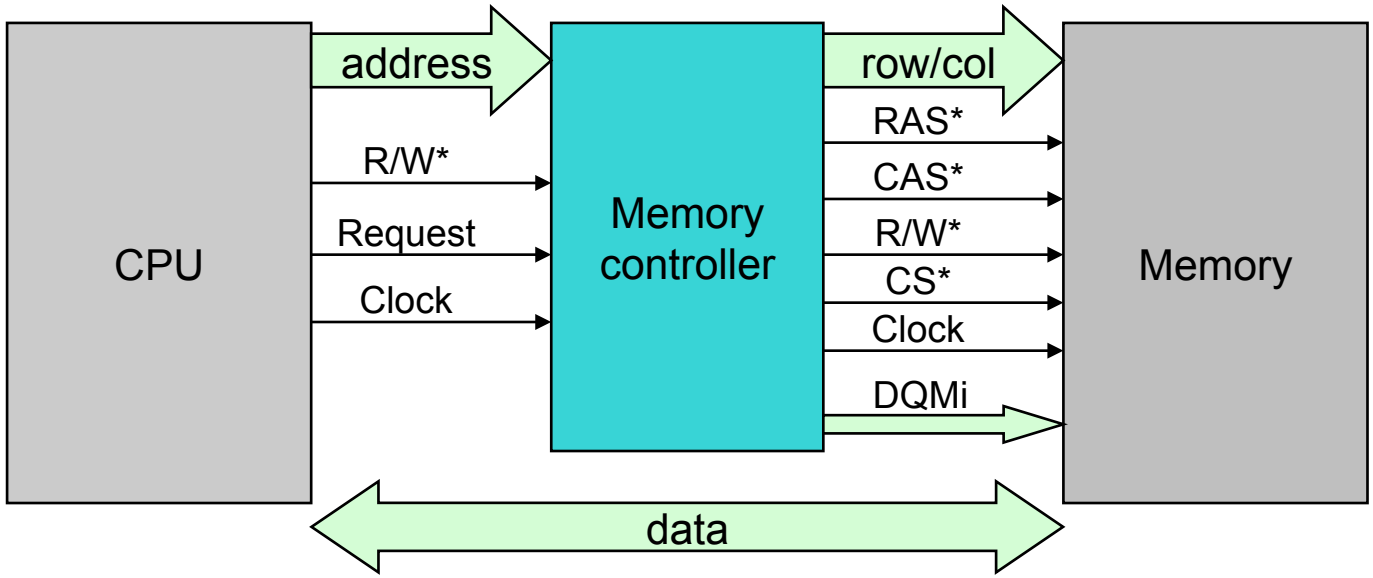
- The memory controller is the circuit in charge of handling the data flow to/from main memory
- It can be a separate chip (for flexibility) or integrated in a microprocessor (for minimum memory latency)
- Its main functions are
  - To identify and properly initialize the memory modules
  - To timely supply addresses and commands
  - To translate CPU addresses to modules' row, col and bank lines
  - To transfer BE* lines to DQM* inputs
  - To ensure DRAM is constantly refreshed within the refresh period
  - To support interfacing to multiple memory modules (map)
  - Ideally, to maximize the throughput of memory transfers

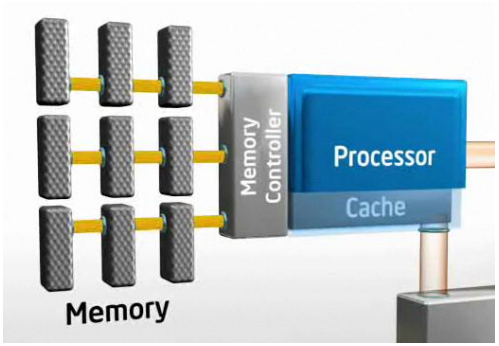
# Description and functions

- Maximizing bandwidth
  - Support for one or more independent channels
    - Total bandwidth is the sum of each channel's bandwidth – when all channels can work in parallel
- The controller may impose restrictions on the distribution of modules across channels
  - You need to know these restrictions before extending the main memory

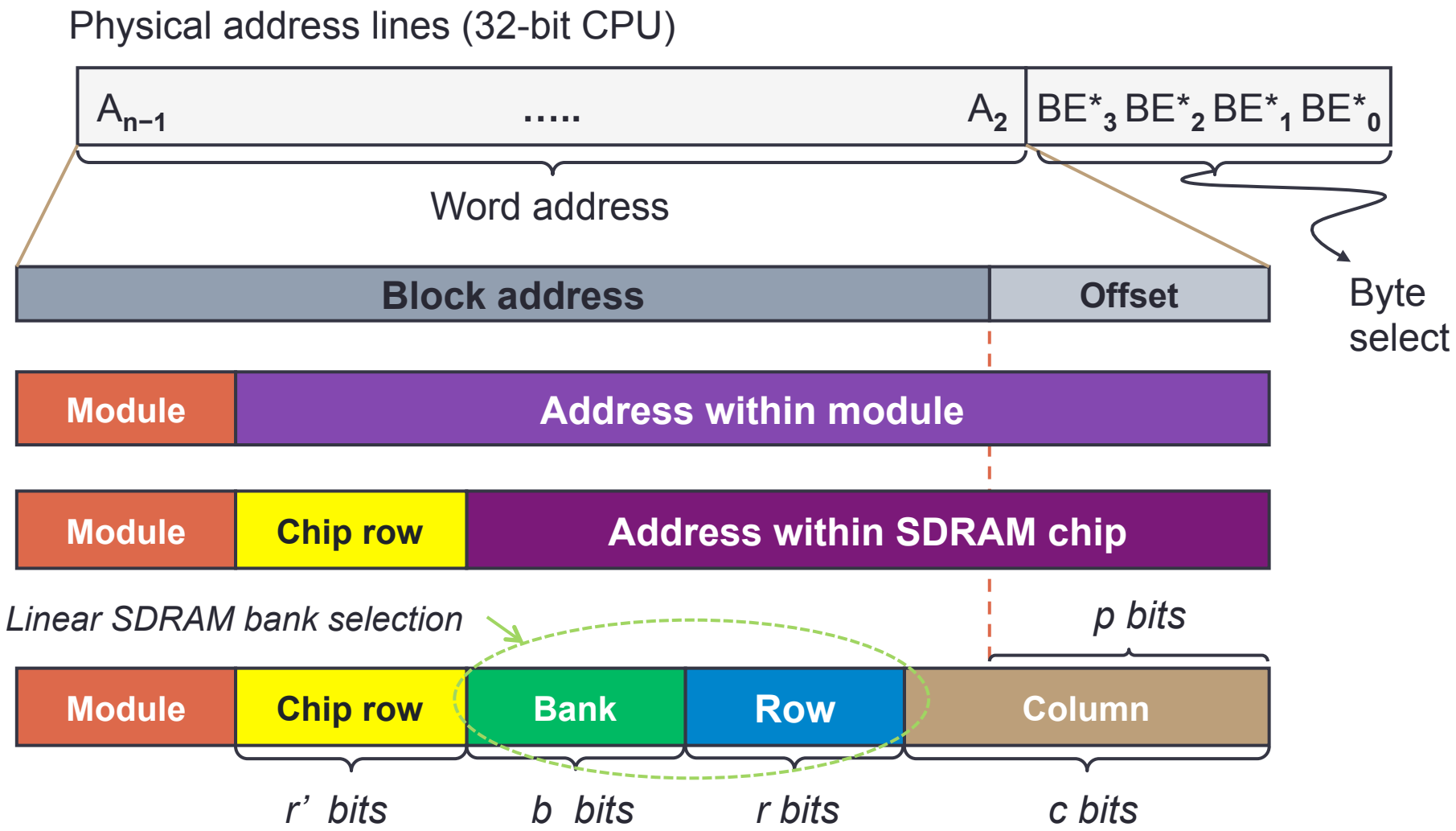


# Description and functions

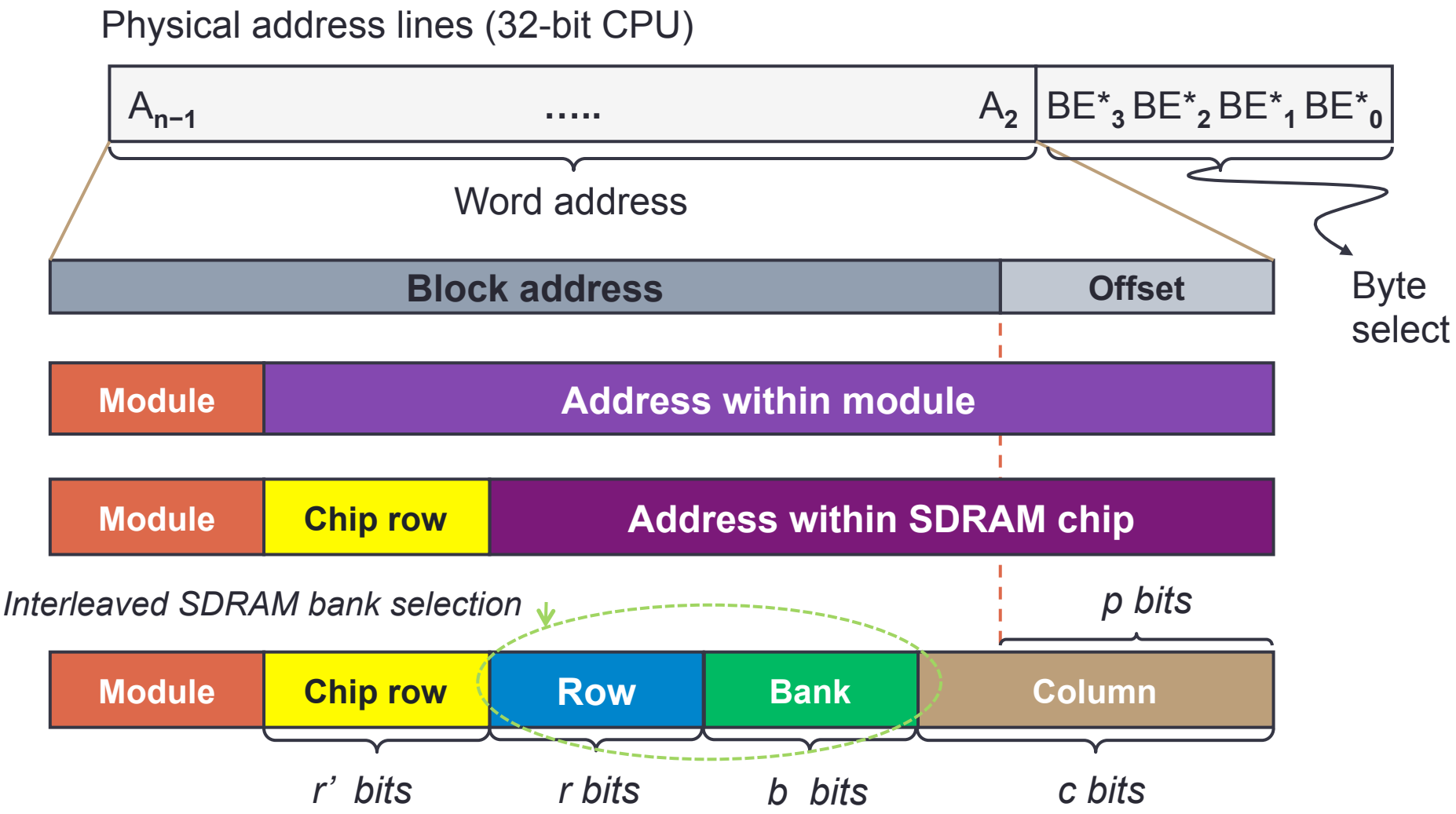


# Memory controller: address handling



# Memory controller: address handling



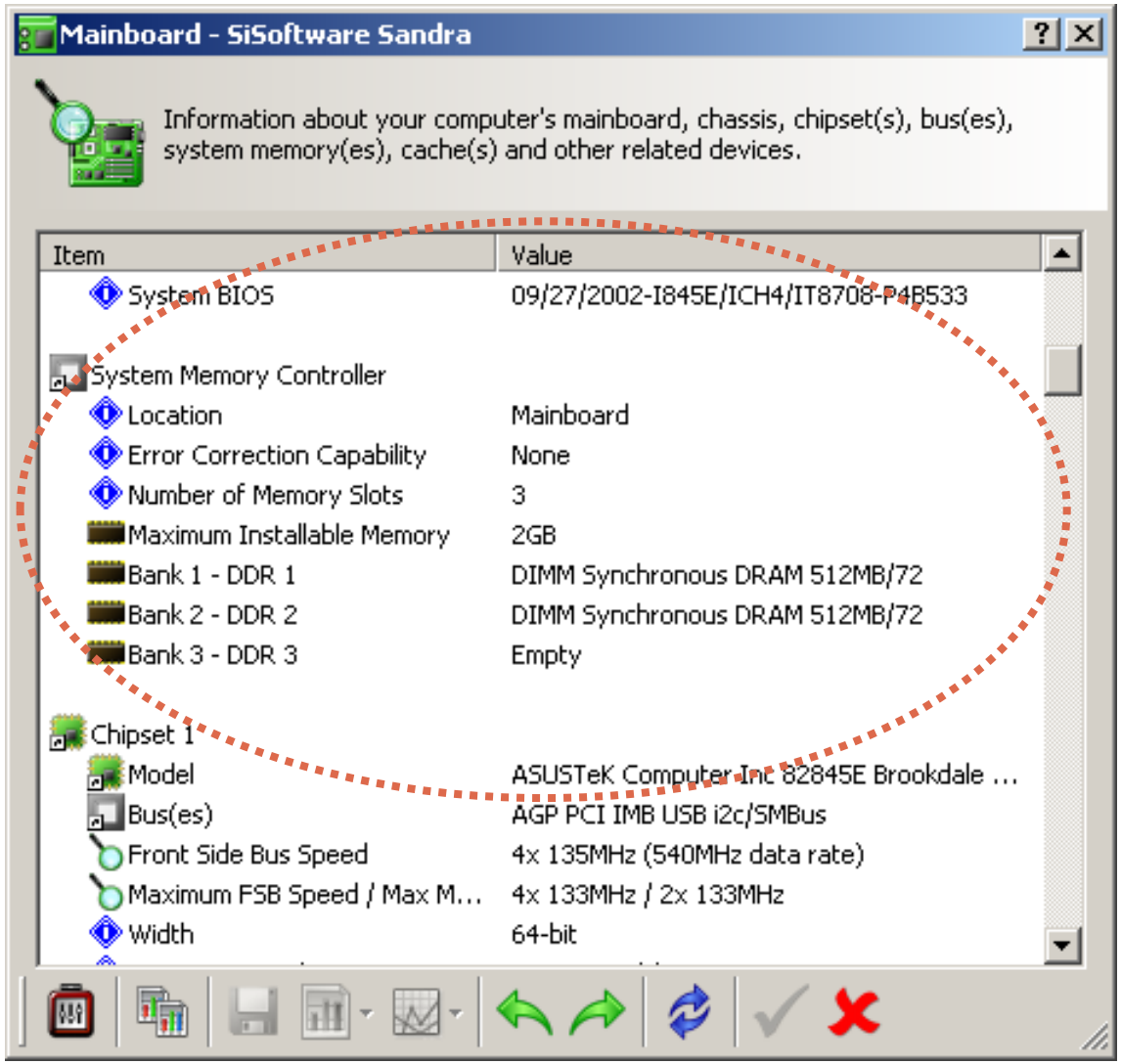
# Memory controller: address handling

- Address structure:
  - *n*: number of address bits
    - Depends on the particular CPU
  - *b:* bank address bits
    - $b = log_2 (banks)$
  - *r:* row address bits
    - $r = log_2 (rows\_per\_chip)$
  - *c:* column address
    - $c = log_2 (columns\_per\_chip)$
  - *r':* address of row of chips in the module
    - $r' = log_2 (chip\_rows\_per\_module)$
  - *p*: number of word within accessed block
    - Depends on the pre-configured block (burst) size

# Initialization of the memory system

- The memory controller is part of the system controller (chipset)
  - The chipset is roughly as complex as a CPU and includes many critical functions
- At startup, the controller
  - checks all the connected modules
  - obtains their characteristics (via the SPD chip)
    - capacity, matrix geometry, timing restrictions, …
  - Initializes the chips in the modules
- The controller also configures
  - the memory map
  - clock frequency
  - proper timing for accessing the modules
  - the frequency of AUTOREFRESH commands

# Memory controller info




- Three memory sockets
- Two of them used by modules of identical capacity