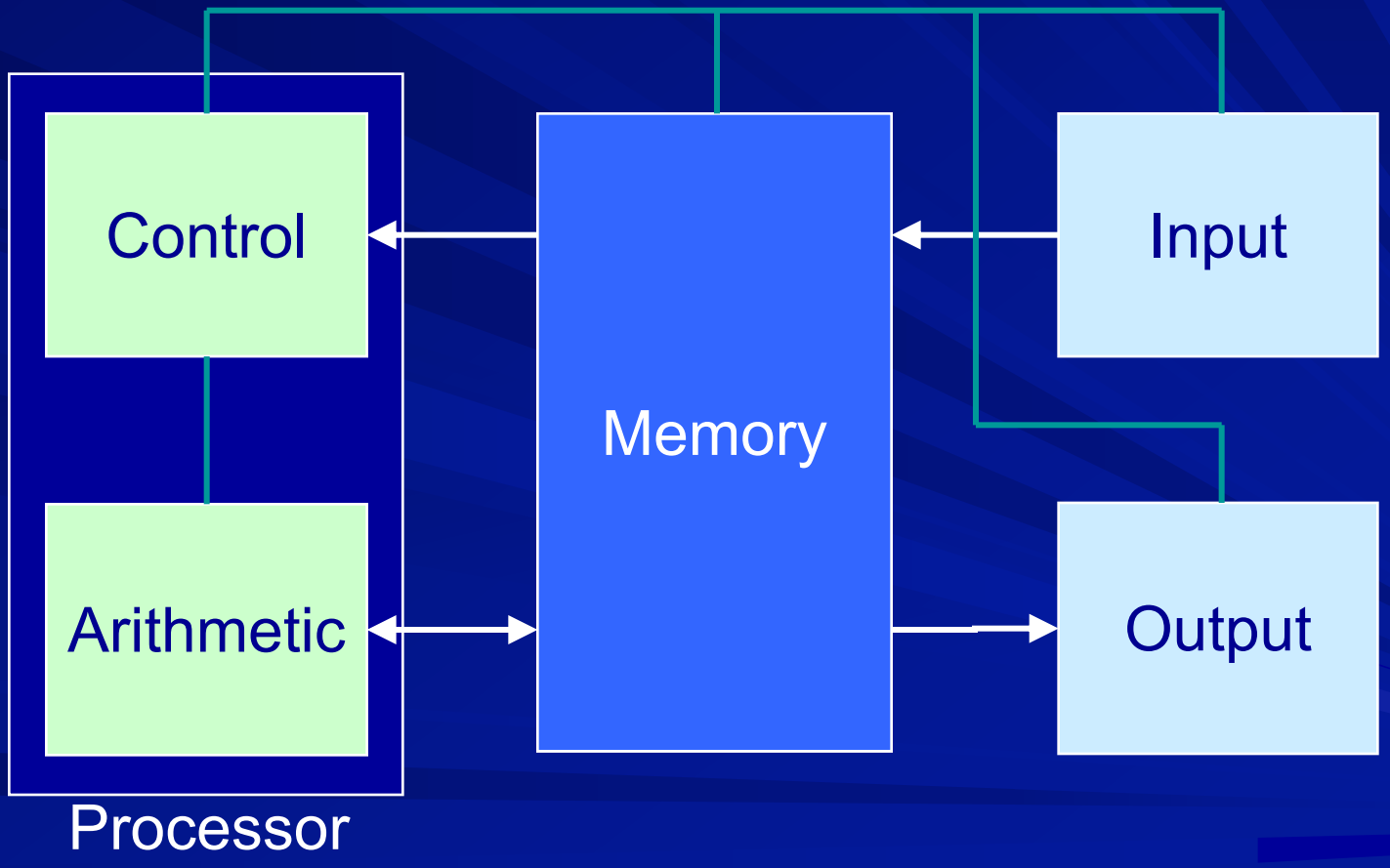
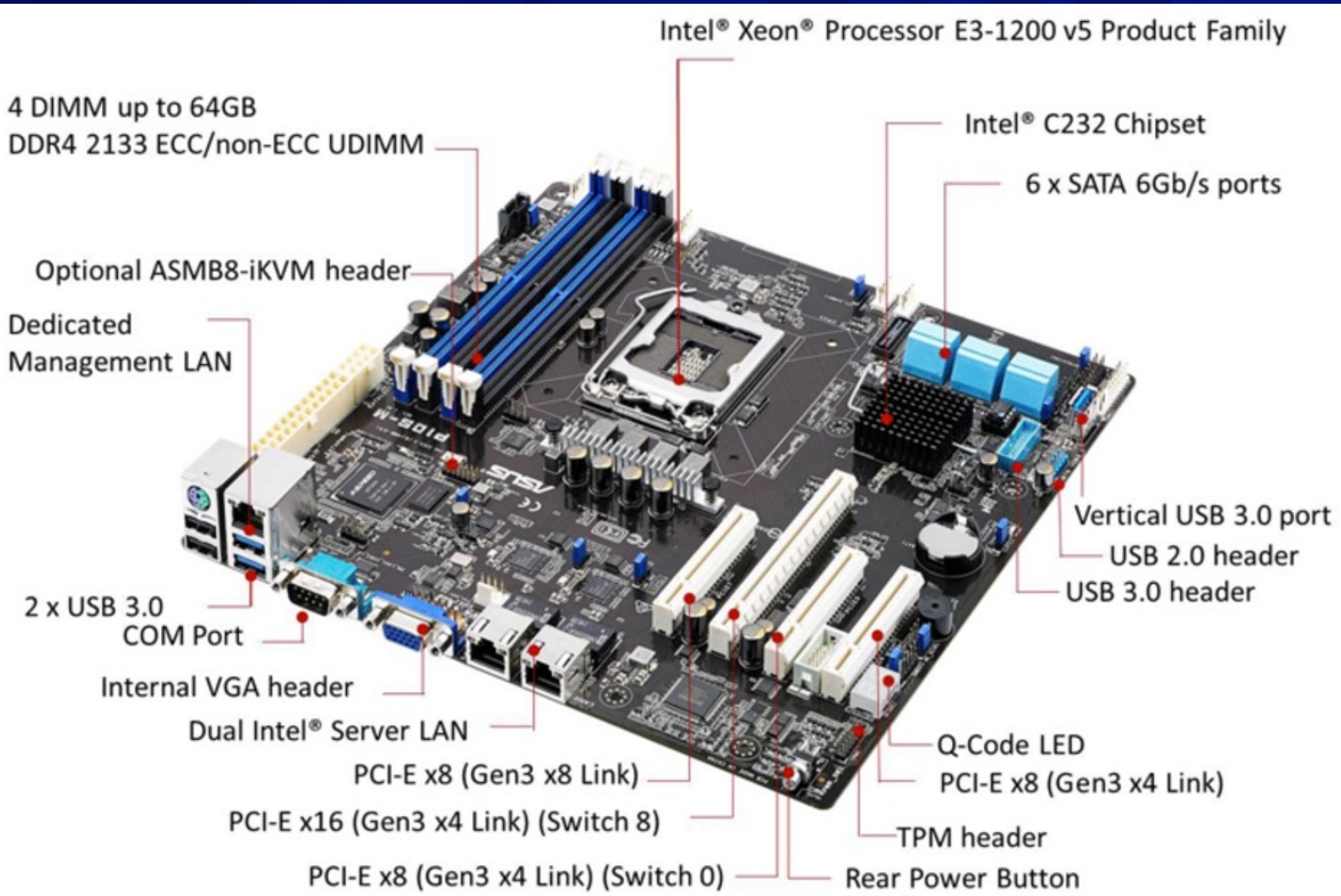
COL216
Computer Architecture

Memory Organization:
Hierarchy
7th March 2022

Computer System: Typical Block Diagram

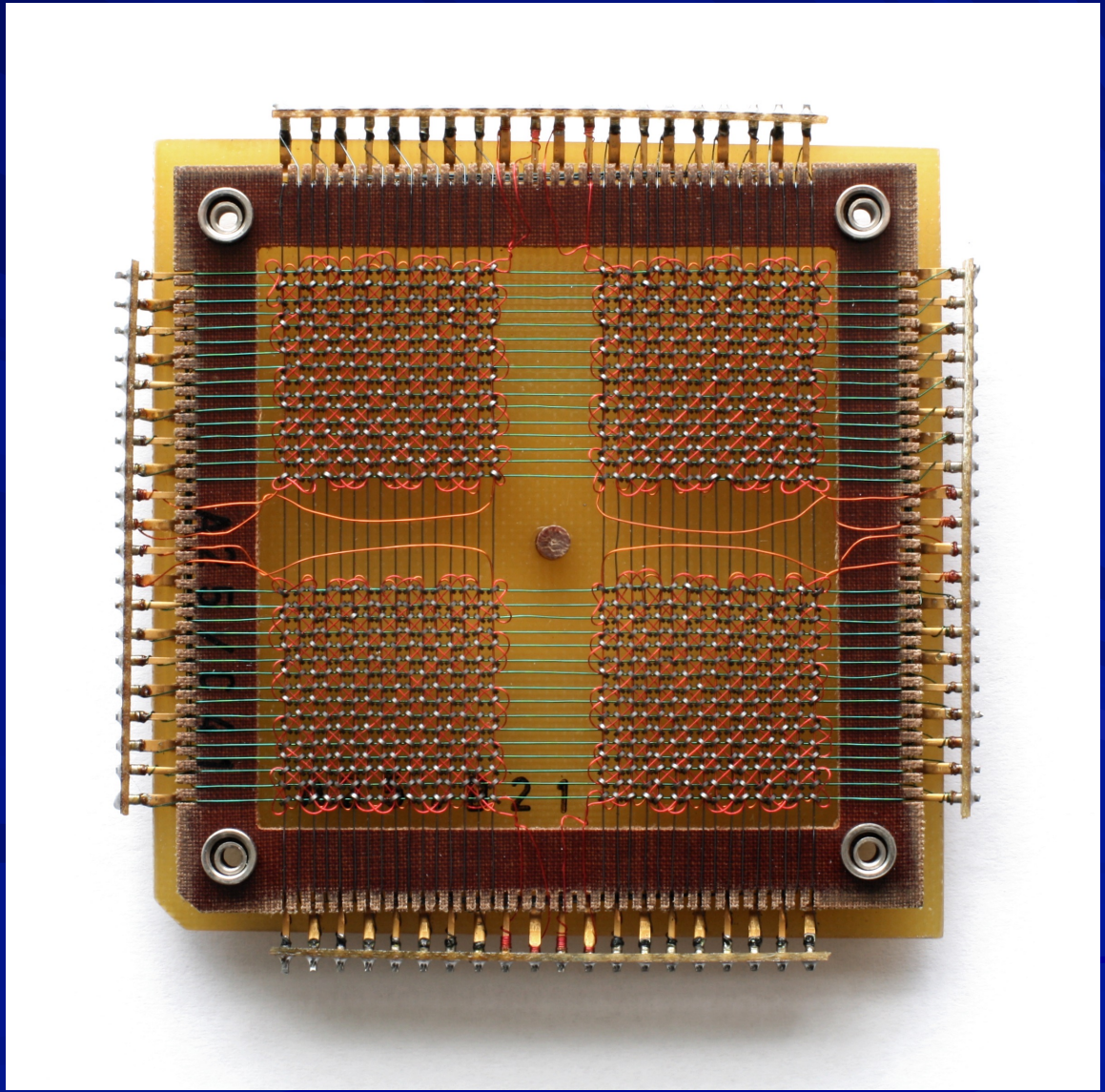


A server board

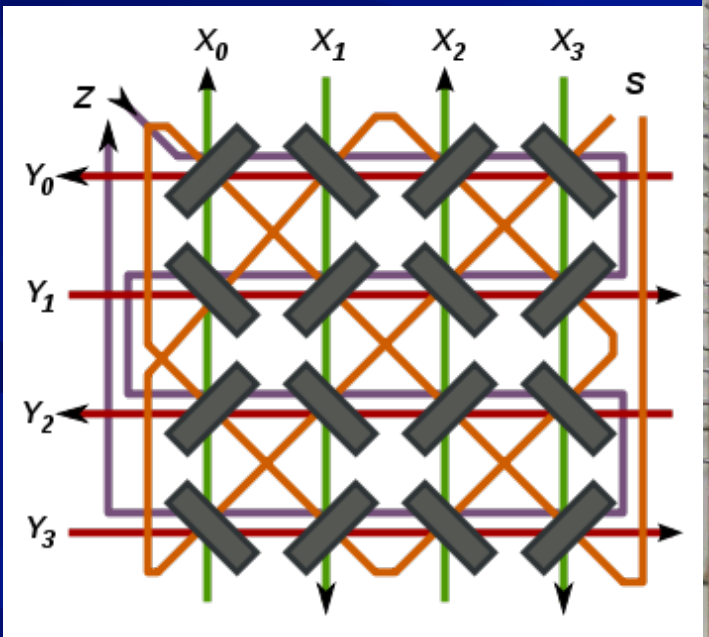


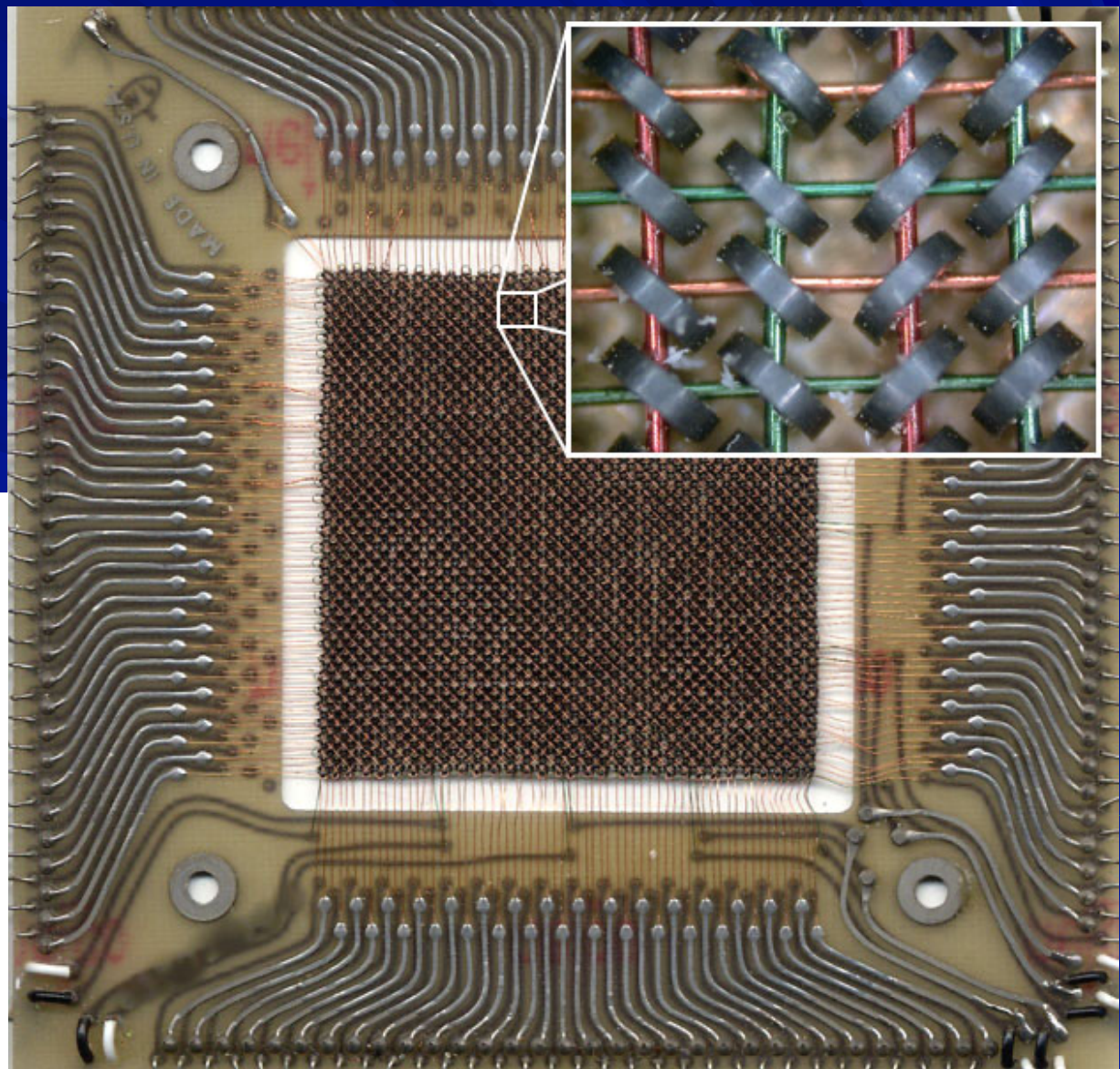
Memory Modules

Ferrite core memory



Ferrite core memory





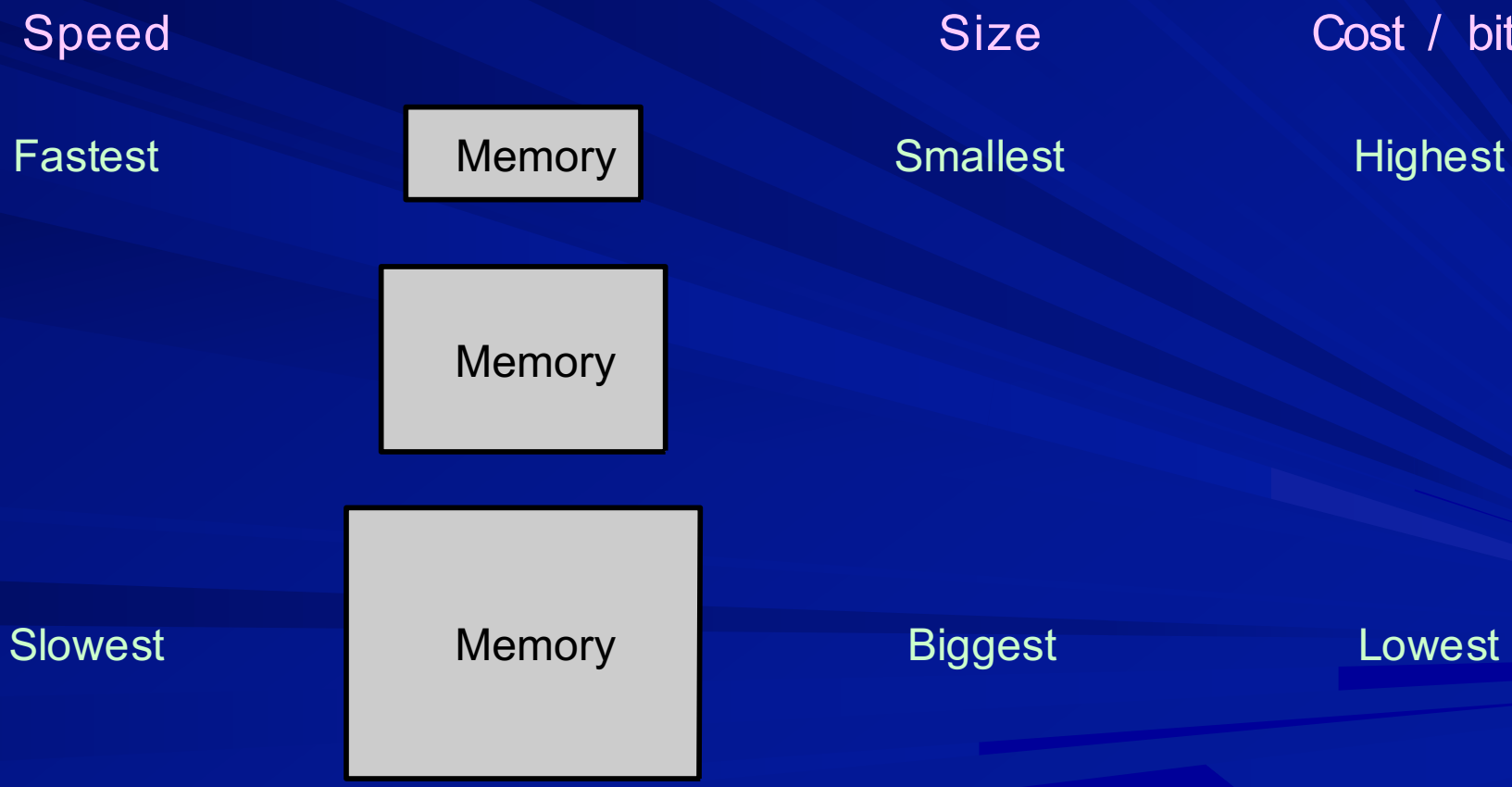
Memory Technologies

- Semiconductor
  - Registers
  - SRAM
  - DRAM
  - FLASH
- Magnetic
  - FDD
  - HDD
- Optical
  - CD
  - DVD
  - Blu Ray

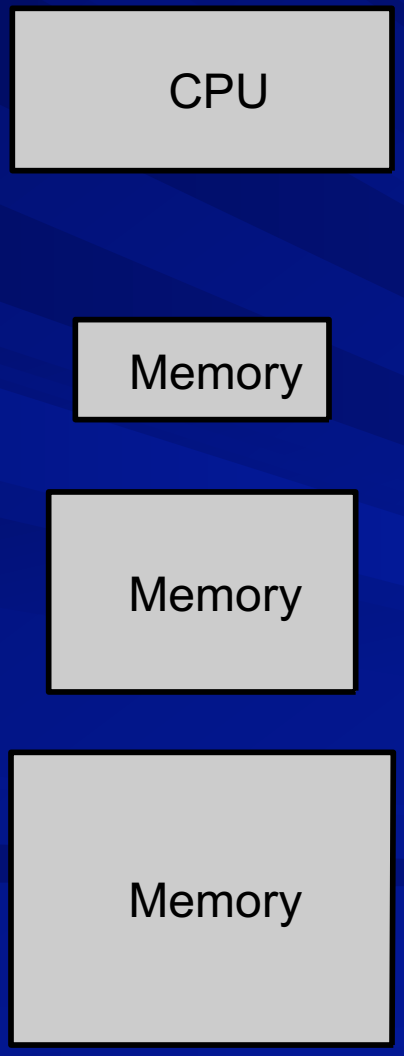
Random Access

Random + Sequential

Speed vs size|cost



Hierarchical Organization



Speed		Size	Cost / bit
Fastest	Cache memory	Smallest	Highest
	Primary memory		
Slowest	Secondary memory	Biggest	Lowest

Primary Memory

- Also called Main memory
- Made using DRAM
- Volatile
- Separate from processor chip (off-chip)

Cache Memory

- 1 to 3 levels
- Made using SRAM
- Volatile
- On processor chip (on-chip)

Cache Memory

- 1 to 3 levels
  - L1 : split L2 : unified L3 : unified
  - L1 : private L2 : priv/sh L3 : shared
- Made using SRAM
- Volatile
- On processor chip (on-chip)

Cache example

Intel Xeon E7 8870

- Level 1 cache
  - 64 KB (split I + D)
- Level 2 cache
  - 256 KB
- Level 3 cache
  - 30 MB (shared by 10 cores)
- External address space
  - 16 TB

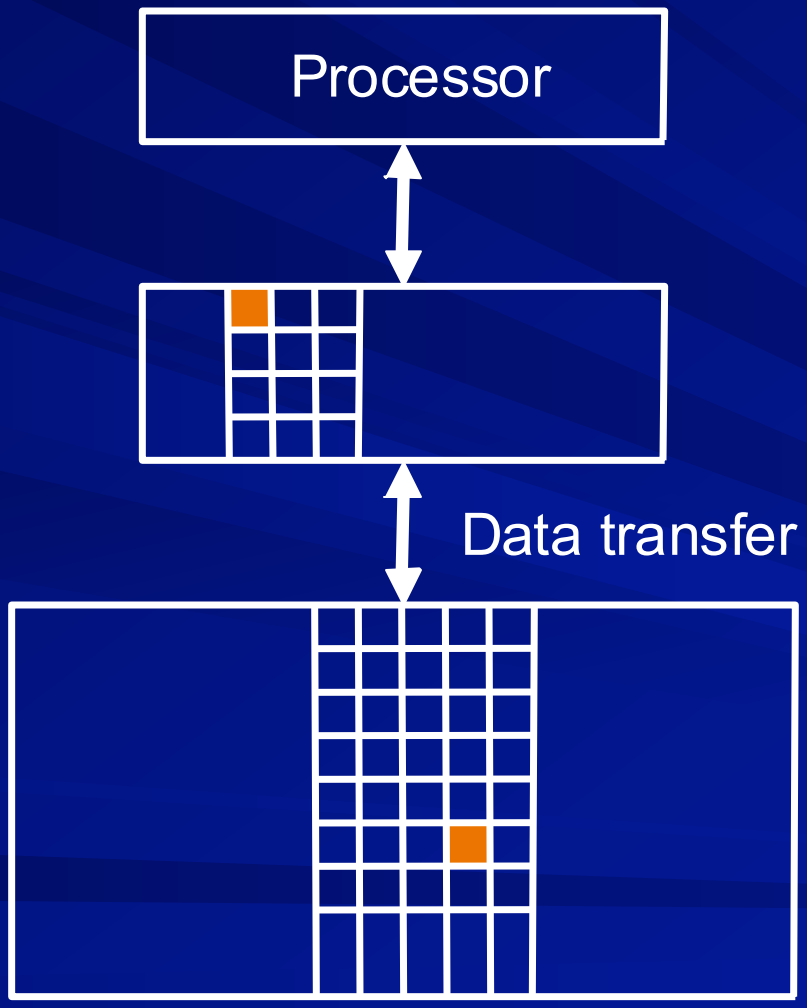
Secondary Memory

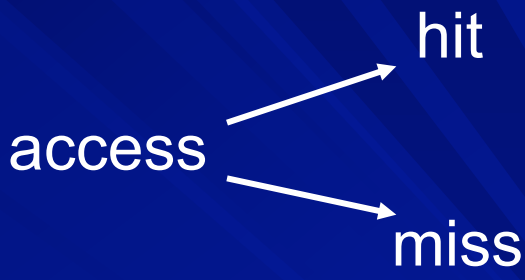
- Flash or Disc drive
- Not random access
- Non-volatile
- Like a peripheral device

Back-up Memory

- Used for archival
- Optical Discs, HDD, Cloud
- External, Removable

How CPU accesses hierarchy?





unit of transfer = block

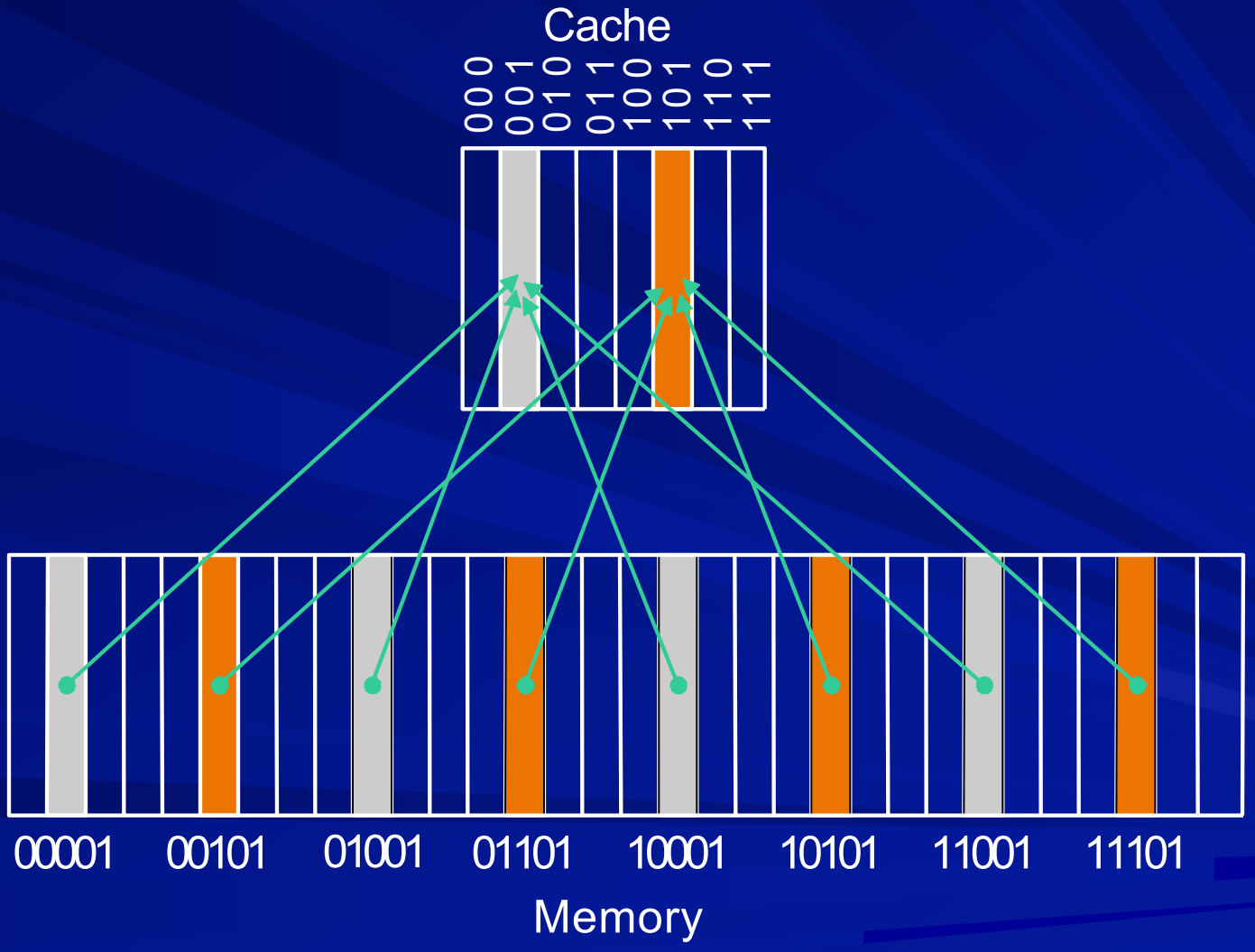
Why does the hierarchy work?

- Temporal Locality
- Spatial Locality

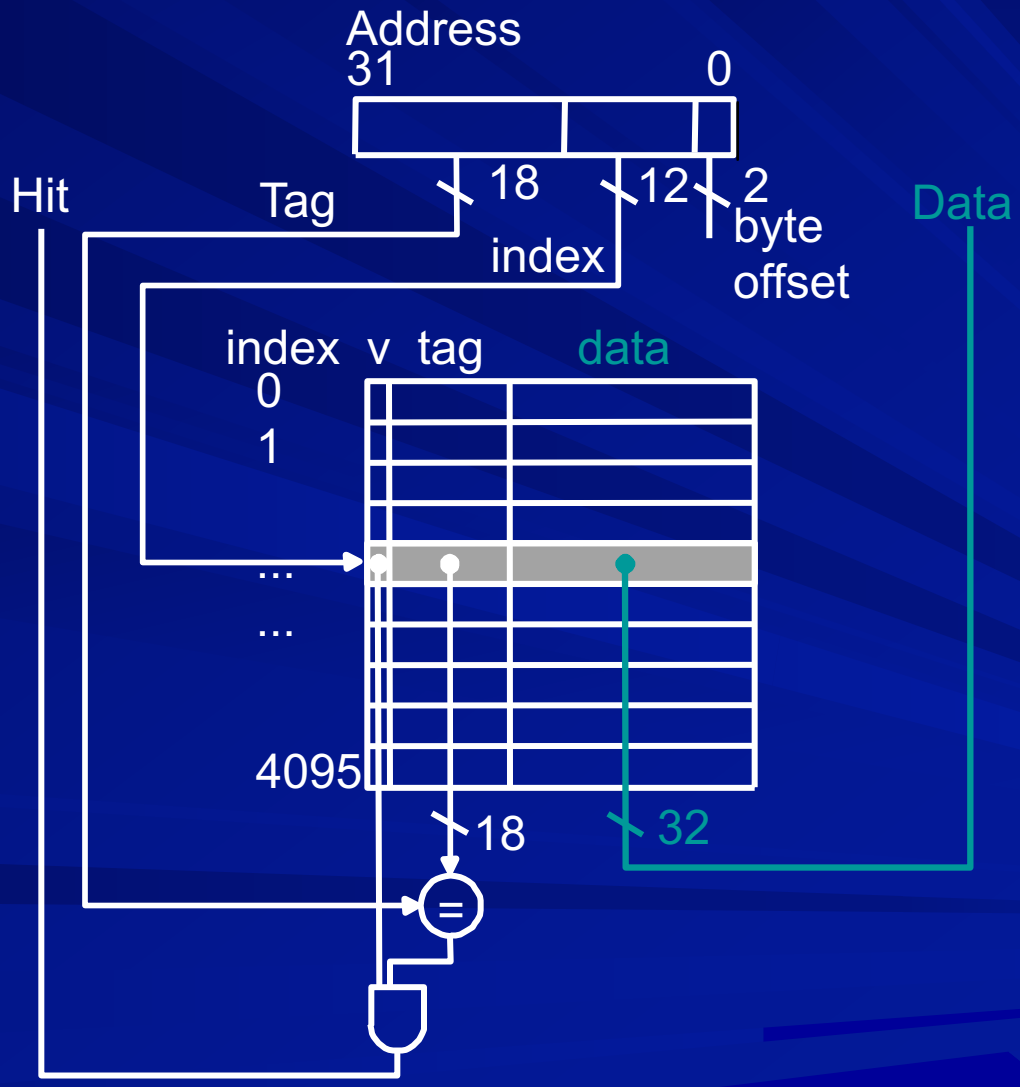
Why does the hierarchy work?

- Temporal Locality
  - references repeated in time
- Spatial Locality
  - references repeated in space
  - Special case: Sequential Locality

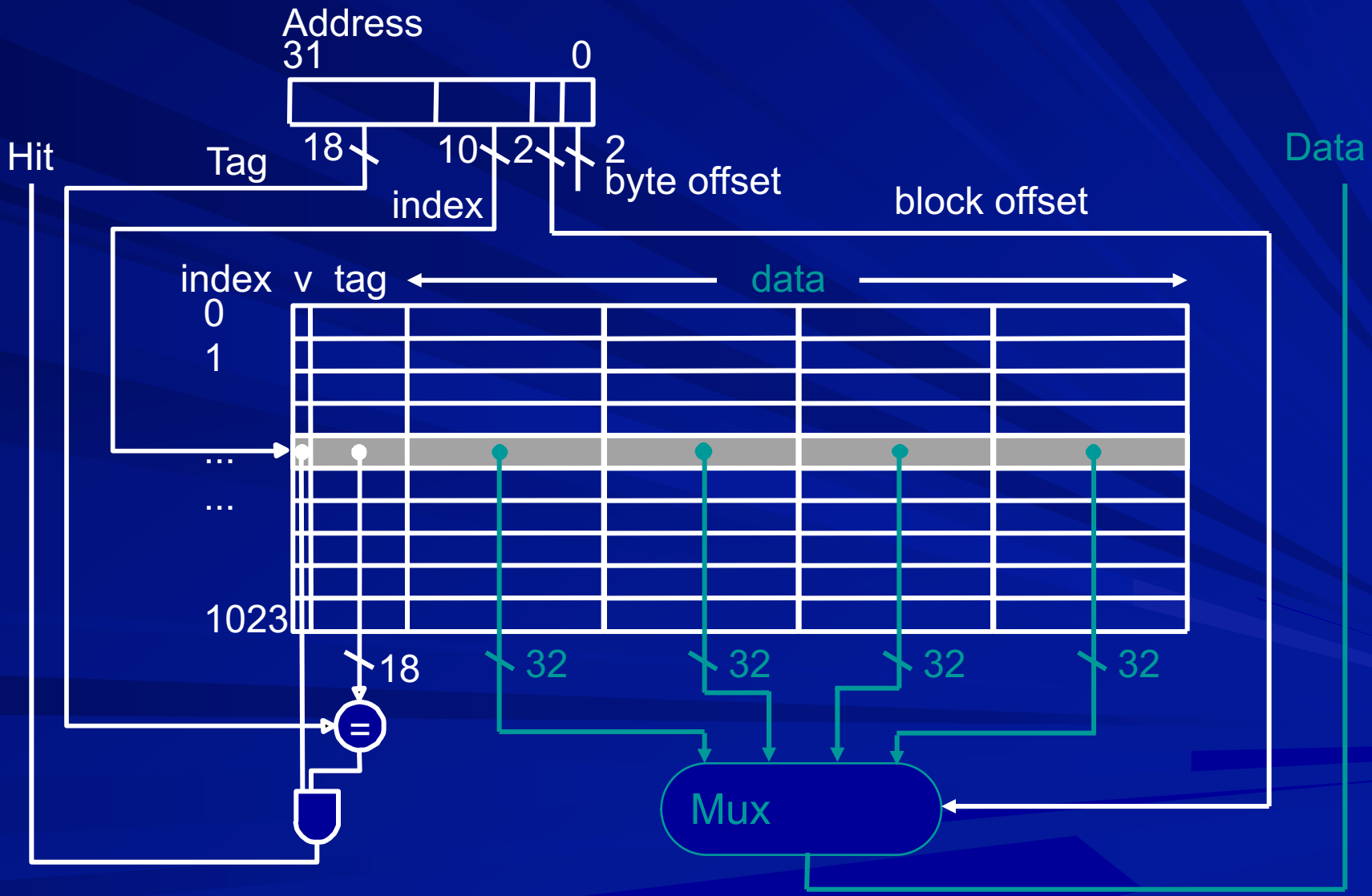
Direct mapped cache



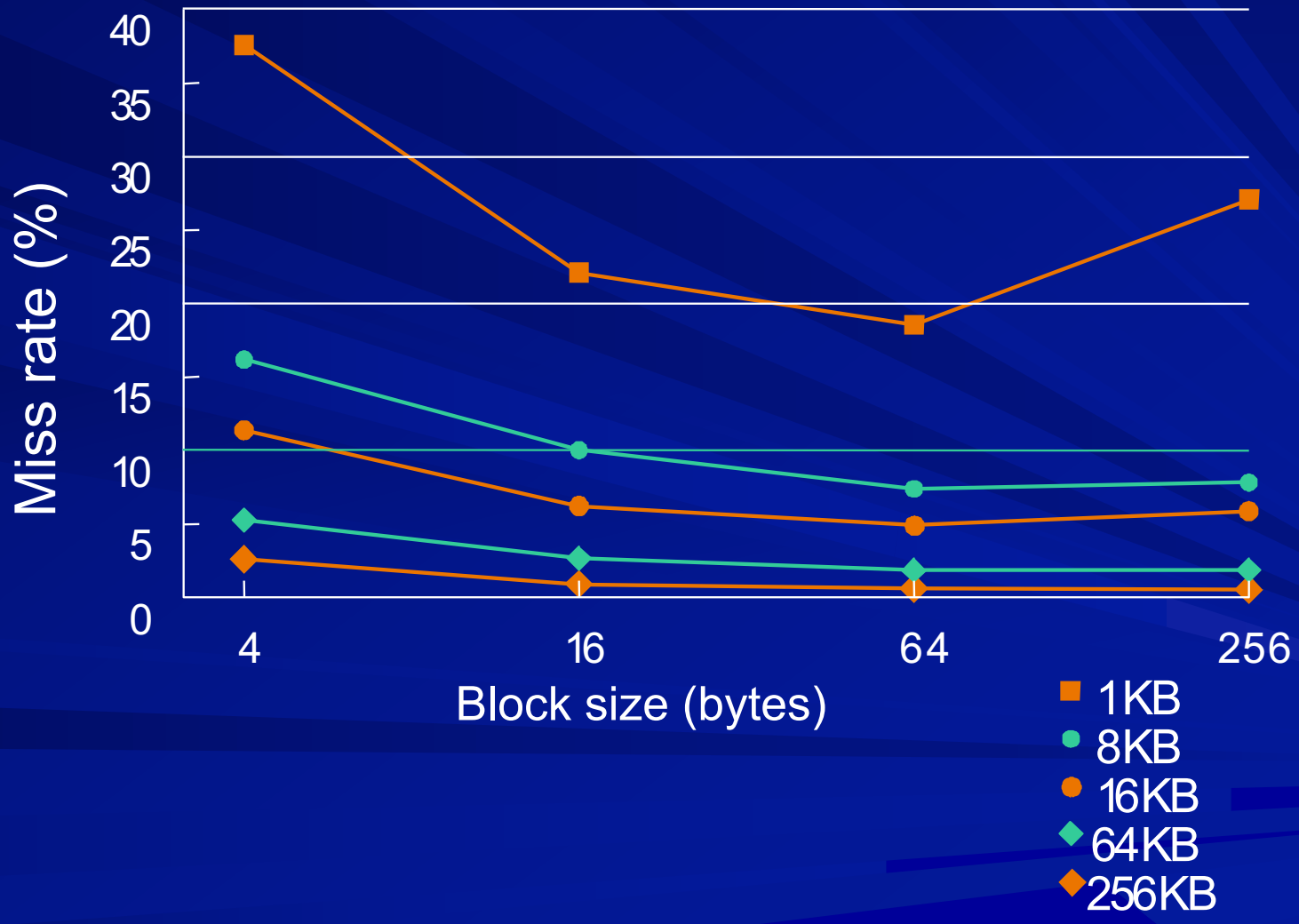
Cache access mechanism



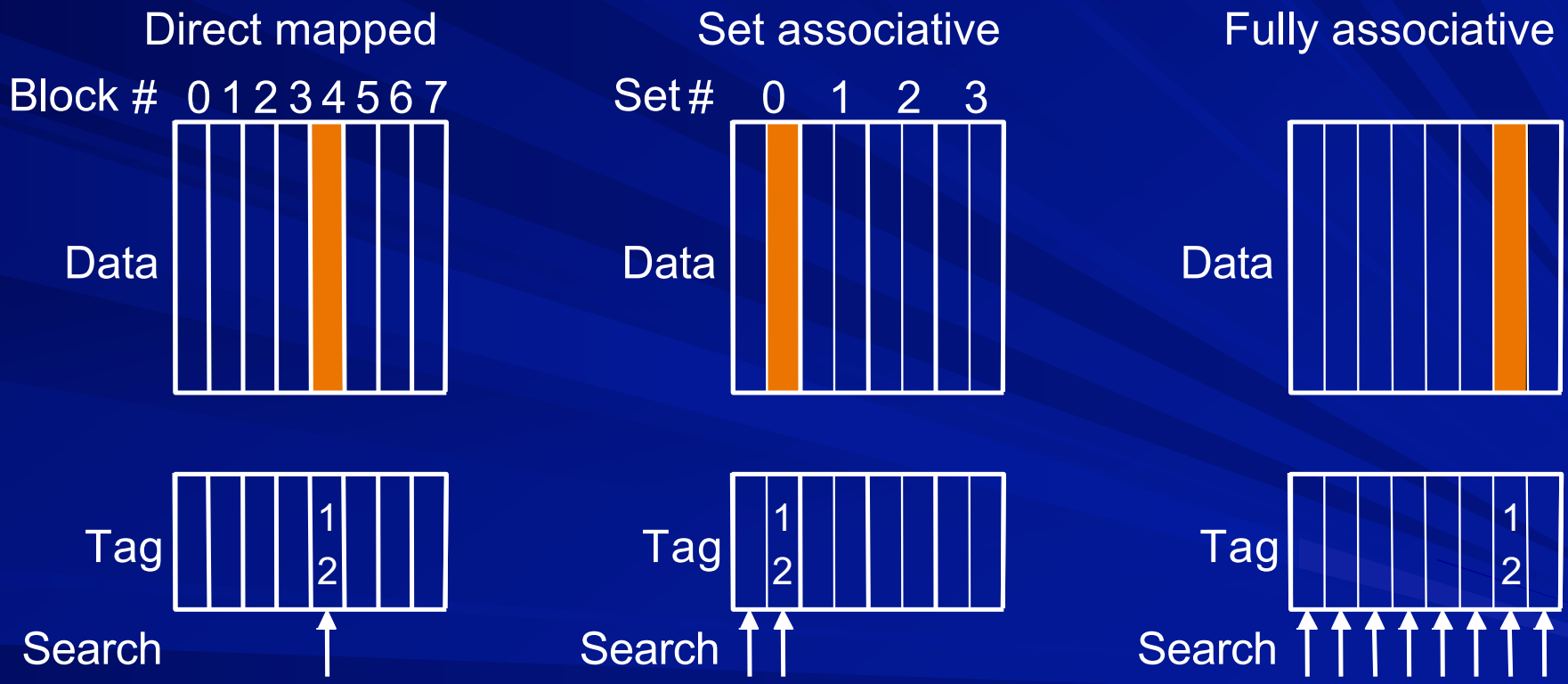
Cache with 4 word blocks



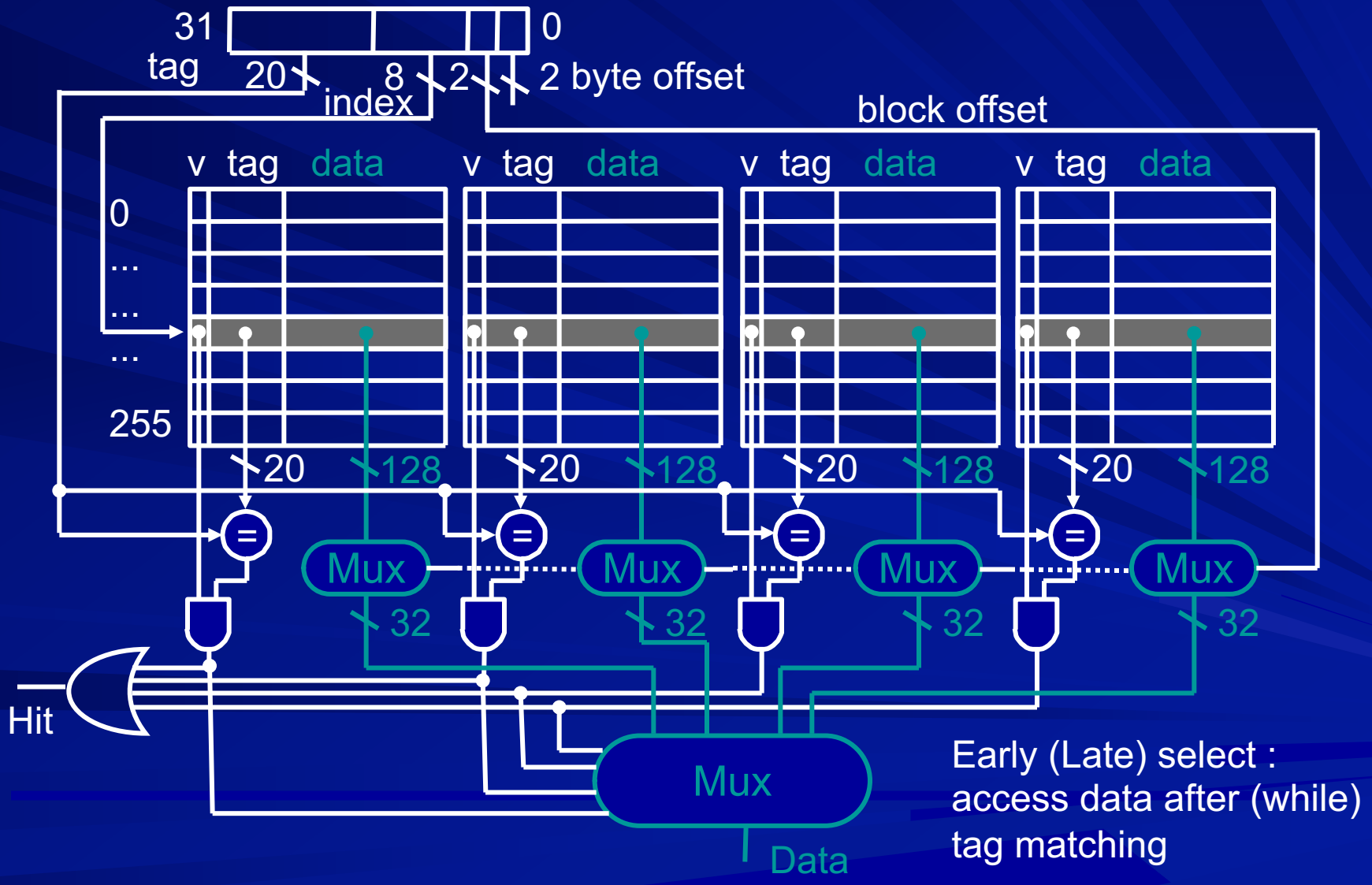
Miss rate vs block size



More flexible block placement



4-way set associative cache



Sizes and bits (Direct mapped cache)

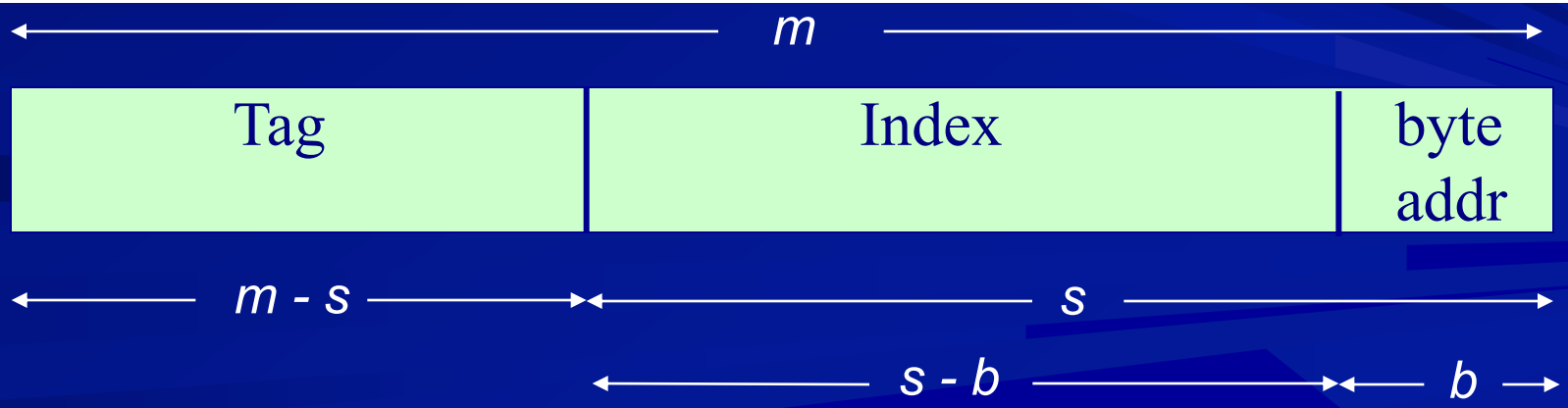
Memory = $M = 2^m$ bytes

Cache = $S = 2^s$ bytes

Block = $B = 2^b$ bytes No. of cache blocks = $\frac{S}{B} = 2^{s-b}$

No. of possible tag values = No. of mem blocks that map to same cache block = $\frac{M}{S} = 2^{m-s}$

No. of tag bits per block = $m - s$



Sizes and bits (Set assoc. cache)

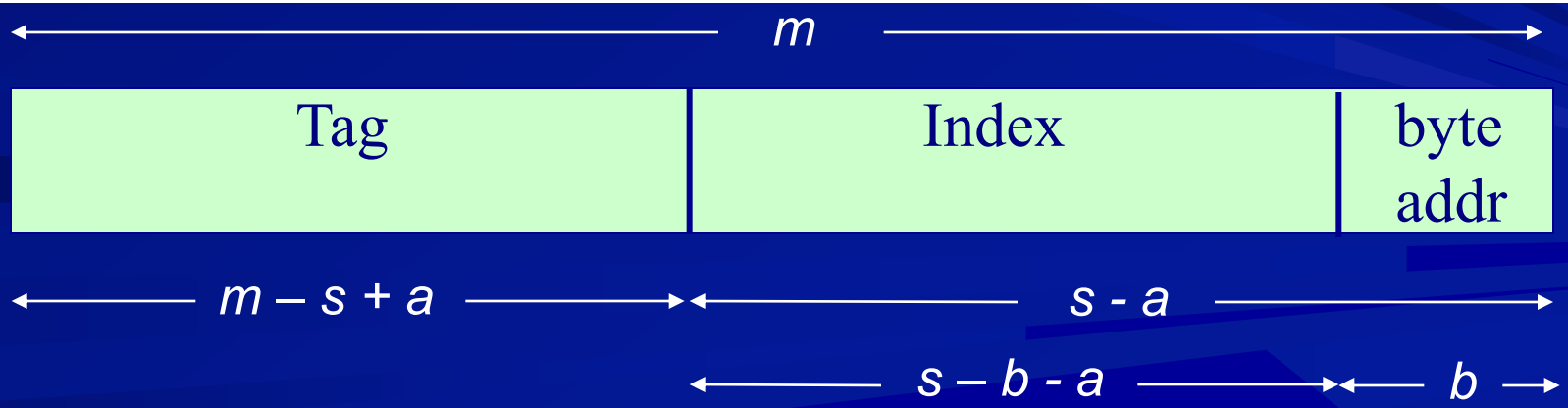
Degree of S.A. = $A = 2^a$

No. of sets = $\frac{S}{A \cdot B}$

Index bits = $\log_2(\frac{S}{A \cdot B}) = s - a - b$

No. of possible tag values = No. of mem blocks that map to same cache set $= \frac{M}{S} A = 2^{m-s+a}$

Tag size = $m - s + a$ Total tag bits = $(m - s + a) \times \frac{S}{B}$



Cache Policies

- Placement — what gets placed where?
- Read — when? from where?
- Load — order of bytes/words?
- Replacement — which one?
- Fetch — when to fetch new block?
- Write — when? to where?

When to initiate memory access?

- Sequential
  - initiate memory access only after detecting a miss
- Concurrent
  - initiate memory access along with cache access in anticipation of a miss

Where CPU gets data from?

- Without data forwarding
  - give data to CPU after filling the missing block in cache
- With forwarding
  - forward data to CPU as it gets filled in cache

Order of bytes/words in a block

- Block in cache may be loaded starting from the missing word, in a wrap around manner
  - useful when data is forwarded to CPU as it gets filled in cache

Which block to replace?

- Least Recently Used (LRU)
- Least Frequently Used (LFU)
- First In First Out (FIFO)
- Random

Cache Policies

- Placement — direct / associative / set assoc
- Read — sequential / concurrent, with/without forwarding
- Load — with/without wrap around
- Replacement — LRU / LFU / FIFO / Random
- Fetch — ??
- Write — ??

Fetch Policies

- Demand fetching
  - fetch on a miss
- Prefetch
  - fetch in anticipation
- Prefetch approaches
  - hardware driven prefetch
  - software driven prefetch

Write Hit

- Write in cache ?
  - Obviously yes
- Write in main memory ?
  - May be
    - Writing in memory has an overhead
    - Writing will keep cache and memory consistent
    - Can we handle inconsistent data?

Write Miss

- Write in cache ?
  - May be
    - Where to write?
- Write in main memory ?
  - Definitely yes, if answer above is no
    - Writing in memory has an overhead
    - Writing will keep cache and memory consistent
    - Can we handle inconsistent data?

Thanks