COL216
Computer Architecture

Performance

3rd March, 2022

Comparing Computers

- Functionality
- Performance/speed
- Power/energy consumption
- Cost
- Reliability
- Size/weight

Different Perspectives

- Individual user's perspective
- Service provider's perspective
- CPU designer's perspective
- Embedded (system designer)'s perspective

Multiple parameters may be important

What are implications of architectural choices on these parameters?

Questions related to performance

- Can one system be better for one user whereas other system be better for another user ?
- Is the system performance only dependent on hardware? What about software?
- How does the machine's instruction set affect performance?

Defining Performance

- How do you define performance?
- Is there a unique way of defining it?

Two notions of "performance"

- Time (response time, execution time, latency)
  - How long does it take for my job to run?
  - How long must I wait for my query?

- Throughput (tasks per unit time)
  - How many jobs can the machine run at once?
  - What is the average execution rate?

Latency vs. Throughput

- Latency and throughput may often go together, but some time they may be in opposition - why?
- When is throughput more important than execution time? tasks / time
- When is execution time more important than throughput?

Understanding Performance

- How are the following likely to effect response time and throughput?
  - Upgrade a machine with a new processor with faster clock. both
  - Increasing the number of jobs (e.g., having a single computer service multiple users). time worse(waiting time), throughput increase
  - Adding a new machine in the lab. higher throughput
  - Increasing the number of processors (e.g., in a network of ATM machines). higher throughput, response time better

Performance Definitions

- Performance is in units of things-per-second
  - bigger is better
- If we are primarily concerned with response time

$$\text{performance} = \frac{1}{\text{execution\_time}}$$

Relative Performance

" X is n times faster than Y" means

$$n = \frac{\text{performance}(X)}{\text{performance}(Y)} = \frac{\text{execution\_time}(Y)}{\text{execution\_time}(X)}$$

Comparing Performance

only one parameter - others include cost, power consumption

- If an Intel processor runs a program in 8 seconds and an ARM processor runs the same program in 10 seconds, how many times faster is the Intel processor?
- n = 10 / 8 = 1.25 times faster (or 25% faster)
- Why might someone choose to buy the ARM processor in this case?

Definitions of Time

- Defined in different ways:
  - Response time : total time = CPU exec time + disk access time + waiting time.
  - CPU execution time : total time spent by CPU
  - User CPU time : time spent in user program
  - System CPU execution time : time spent by OS.
- For example, a program may have a system CPU time of 22 sec., a user CPU time of 90 sec., a CPU execution time of 112 sec., and a response time of 162 sec..

Computing CPU time

- The time to execute a given program can be computed as

  CPU time = CPU clock cycles x clock cycle time or

  CPU time = CPU clock cycles / clock rate

- CPU clock cycles = (instr/program) x (clock cycles/instruction)

  = Instruction count x CPI

  which gives

  loop - count all instructions and cpi is avg cpi

  user

  CPU time = Instruction count x CPI x clock cycle time

  CPU time = Instruction count x CPI / clock rate

- The units for this are

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{clock cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}$$

Example of computing CPU time

- If clock rate = 500 MHz, find execute time for a program that executes 1,000 instructions, if the CPI for the program = 3.5?

CPU time = instruction count x CPI / clock rate

CPU time = 1000 x 3.5 / (500 x 10^6) sec = 7 μs

- If clock rate increases from 500 MHz to 600 MHz and the other factors remain the same, how many times faster will the computer be?

$$\frac{\text{CPU time old}}{\text{CPU time new}} = \frac{\text{clock rate new}}{\text{clock rate old}} = \frac{600\ \text{MHz}}{500\ \text{MHz}} = 1.2$$

Computing CPI

multicycle processor
in pipelined, ideal cpi is 1, account for branch delays, what % of instructions are getting delayed due to hazards and then add those cycles to cpi

- CPI = avg. no. of cycles per instruction.

$$CPI = \sum_{i=1}^{n} CPI_i \times F_i$$

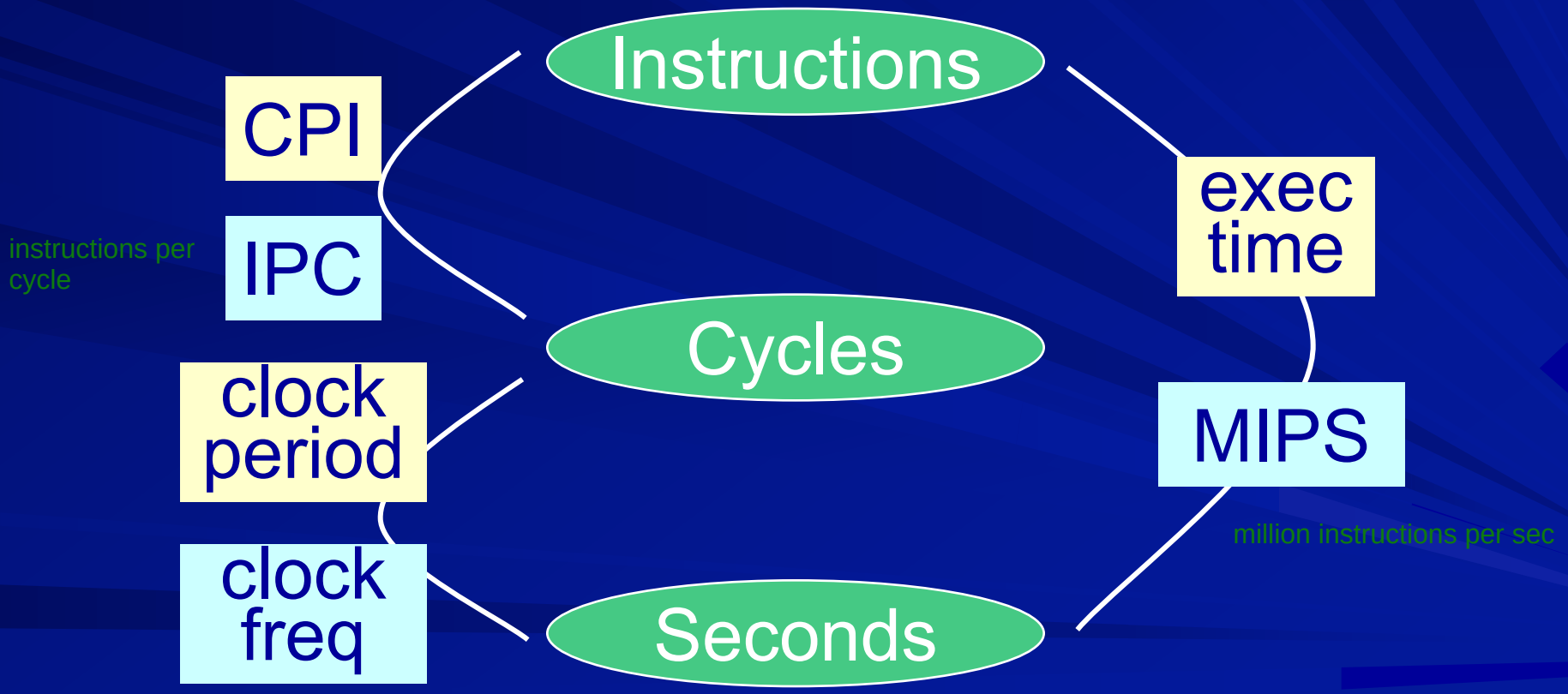
Example:

Op	F_i	CPI_i	CPI_i x F_i	% Time
DP	50%	4	2.0	50%
Load	20%	5	1.0	25%
Store	10%	4	0.4	10%
Branch	20%	3	0.6	15%
Total	100%		4.0	100%

Different numbers of cycles for different instructions

- Multiplication takes longer than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers
- Changing cycle time often changes number of cycles required for various instructions

Instructions↔Cycles↔Seconds



Performance

- Performance is determined by execution time
- Common pitfall: Use one of these variables as indicator of performance
  - # of cycles to execute program? clock freq
  - # of instructions in program? cpi needed
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?

Example

- Computer A, (400 MHz clock), a program runs in 10 sec.
- Target: build a new machine B, the program runs in 6 sec.
- New (more expensive!) technology available to substantially increase the clock rate.
- This change will affect the rest of the CPU design, causing B to require 1.2 times as many clock cycles as A for the same program.
- What clock rate will make it possible for the designer to achieve the target?

Solution

- Two implementations (A and B) of the same instruction set architecture (ISA).

- If two machines have the same ISA which of our quantities (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?

Solution continued

A has clock cycle time = 2.5 ns
B has clock cycle time = *T* ns

- For given program,
  A has CPI = *c* , execution time = 10 s
  B has CPI = 1.2 *c* , execution time = 6 s

No. of instructions executed = *N* 10^9 for both

10 = *N* * *c* * 2.5 6 = *N* * 1.2 * *c* * *T*

T^{-1} = 1.2 *10 / (2.5 * 6) = .8 GHz = 800 MHz

Compiler factor

Instructions r generated by compiler so we will speak of compilers here

clock period is same

A compiler designer is to decide between two code sequences.

- First code sequence: cpi = 2
  - 2 of A, 1 of B, 2 of C
- Second code sequence: cpi = 1.5
  - 4 of A, 1 of B, 1 of C

3 classes of instructions:

- A : 1 cycle
- B : 2 cycles
- C : 3 cycles

- Which sequence will be faster?
  - How much?
  - CPI for each?

Compiler and MIPS

- Two compilers being tested for 100 MHz machine with 3 classes of instructions:
  - A (1 cycle), B (2 cycles), and C (3 cycles)
  - Compiler 1: 5M A, 1M B, 1M C instructions
  - Compiler 2: 10M A, 1M B, 1M C instructions
- Which sequence has higher MIPS?
- Which sequence has lower execution time?

compiler 2 : 15M cycles
compiler 1 : 10M cycles

2 is slower

12M instr in 15 cycles
0.4 MIPS
7M instr in 10 cycles
0.7 MIPS

Programs to test performance

- Performance best determined by running a real application
  - Use programs typical of expected workload, or
  - Typical of expected class of applications e.g., compilers/editors, scientific applications, graphics, etc.

Computer Benchmarks

- Benchmark = program (s) used to evaluate computer performance.
- Allow a user to make performance comparisons
- Benchmarks should
  - Be representative of the type of applications
  - Not be overly dependent on one or two features
- Benchmarks can vary greatly in terms of their complexity and their usefulness.

Sources of Benchmarks

- Synthetic benchmarks
  - small benchmarks can be easily written
  - nice for architects and designers
  - easy to use, but can be abused
- SPEC (**S**tandard **P**erformance **E**valuation **C**orporation)
  http://www.spec.org

  "An ounce of honest data is worth a pound of marketing hype"
  - benchmarks derived from real programs

SPEC Benchmark Categories

- Cloud
- CPU
- Graphics/Workstations
- ACCEL/MPI/OMP
- Java Client/Server
- Mail Servers
- Storage
- Power
- Virtualization
- Web Servers

SPEC CPU Benchmarks

- SPEC CPU 92 => SPEC CPU 95 => SPEC CPU 2000 => SPEC CPU 2006 => SPEC CPU 2017
- SPEC CPU 2017: 43 benchmarks organized into 4 suites
  - SPECspeed 2017 Integer
  - SPECspeed 2017 Floating Point
  - SPECrate 2017 Integer
  - SPECrate 2017 Floating Point

Integer benchmarks

- Perl interpreter, GNU C compiler
- Route planning
- Network simulation
- Video compression, Data compression
- Tree search in games like chess
- Recursive solution generator (Sudoku)

Floating point benchmarks

- Modeling of physical phenomena like explosion, molecular dynamics, ocean, atmosphere etc.
- Optical tomography
- Ray tracing, 3-d rendering, animation
- Computational electromagnetics
- Image manupulation

Amdahl's Law

$$\text{Speedup} = \frac{\text{ExTime old}}{\text{ExTime new}} = \frac{\text{Performance new}}{\text{Performance old}}$$

Suppose that an enhancement accelerates a fraction $\text{Fraction}_{\text{enhanced}}$ of the task by a factor $\text{Speedup}_{\text{enhanced}}$

Execution Time After Improvement =
time unaffected + time affected / improvement

speeding up some part of the program

Amdahl's Law

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times \left[(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right]$$

$$\text{Speedup} = \frac{\text{ExTime}_{old}}{\text{ExTime}_{new}} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

Example:

A program runs in 100 seconds on a machine, with multiply responsible for 80%. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?

How about making it 5 times faster?

0.25 =0.2 + 0.8/s => 0.05 = 0.8/s => s=>16

Example

- Suppose we enhance a machine making all floating-point instructions run five times faster.
- Let the execution time of some benchmark before the floating-point enhancement be 10 seconds.
- What will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

$Speedup_{enhanced} = 5$

$ExTime_{old} = 10$

$Fraction_{enhanced} = 0.5$

$ExTime_{new} =$

$10\ (1-.5\ +.5/5) = 6$

$Speedup = 10/6 = 1.67$

Example continued

- We are looking for a benchmark to show off the new FP unit.
- Want the overall benchmark to show a speedup of 3.
- Benchmark under consideration runs for 100 seconds with the old floating-point hardware.
- How much of the execution time would FP instructions have to account for in this program in order to yield our desired speedup on this benchmark?

$Speedup_{enhanced} = 5$

$Speedup = 3$

$ExTime_{old} = 100$

$ExTime_{new} = 33.3$

$Fraction_{enhanced} = f$

$33.3 = 100(1-f + f/5)$

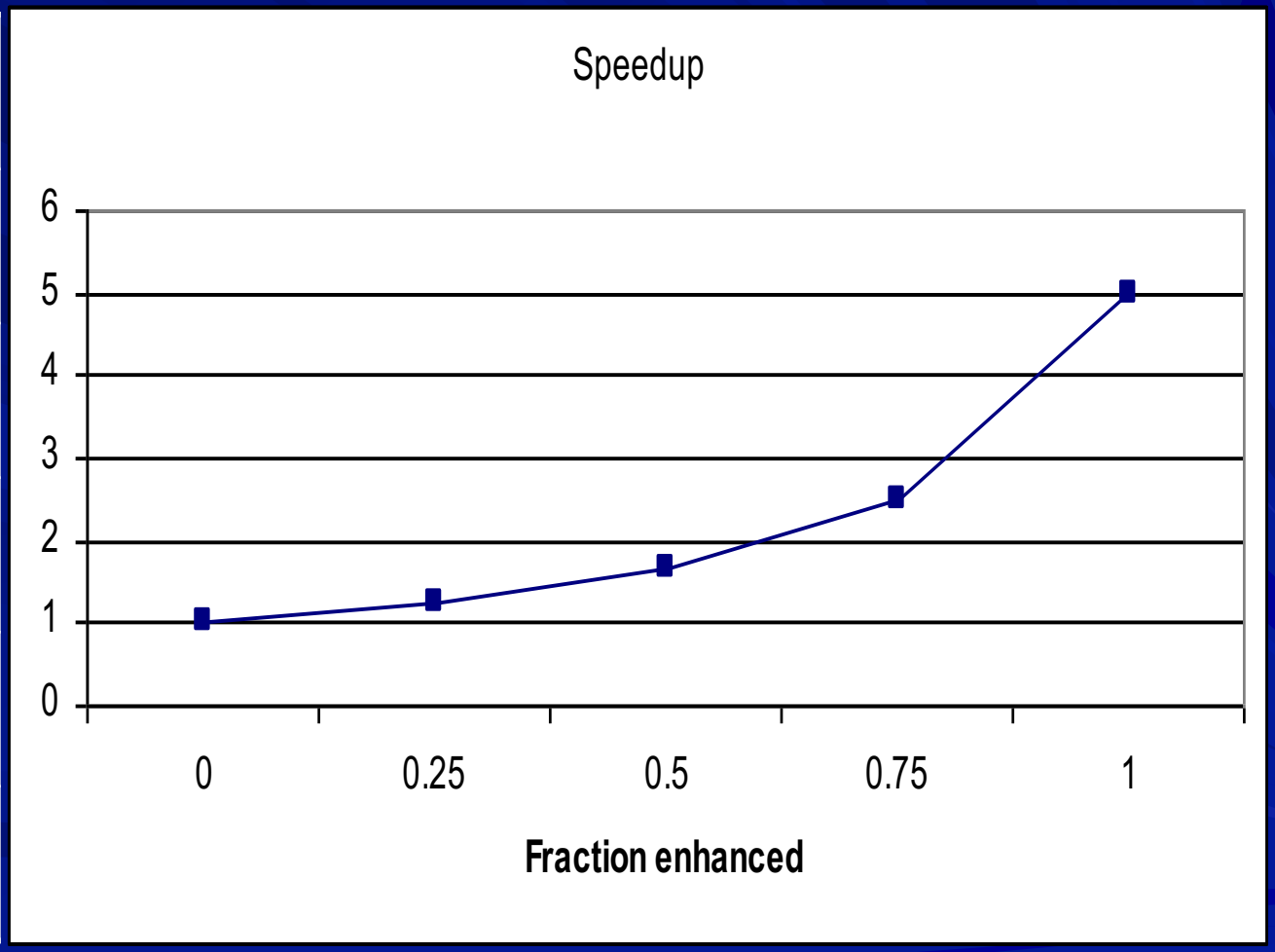
$1 = 3 (1 – 4 f / 5)$

$f = 5 / 6$

$100 * 5/6 = 83.3$

Example continued

$Frac_{enh}$	Speedup
0.00	1.00
0.25	1.25
0.50	1.67
0.75	2.50
1.00	5.00



Example: FP h/w vs s/w

C Clock freq 1000 MHz

	Instr mix in prog P	Cycles in MFP	Int instr in MNFP
FP multiply	10%	6	30
FP add	15%	4	20
FP divide	5%	20	50
Int instructions	70%	2	
CPI			

10% * 6 + 15% * 4 + 5% * 20 + 70% * 2
= 0.6 + 0.6 + 1.0 + 1.4 = 3.6

Example: FP h/w vs s/w

C Clock freq 1000 MHz

	Instr mix in prog P	Cycles in MFP	Int instr in MNFP
FP multiply	10%	6	30
FP add	15%	4	20
FP divide	5%	20	50
Int instructions	70%	2	
CPI		3.6	2.0
MIPS = C/CPI		277.8	500.0
N		300M	

300M(10% * 30 + 15% * 20 + 5% * 50 + 70% * 1)
= 300M(3 + 3 + 2.5 + 0.7) = 300M * 9.2 = 2760M

Example: FP h/w vs s/w

C Clock freq 1000 MHz

	Instr mix in prog P	Cycles in MFP	Int instr in MNFP
FP multiply	10%	6	30
FP add	15%	4	20
FP divide	5%	20	50
Int instructions	70%	2	
CPI		3.6	2.0
MIPS = C/CPI		277.8	500.0
N		300M	2760M
ExTime = N*CPI/C		1.08	5.52
MFLOPS =	30%*300M/ExTime	90/1.08	90/5.52

number of floating point instructions

mega flops

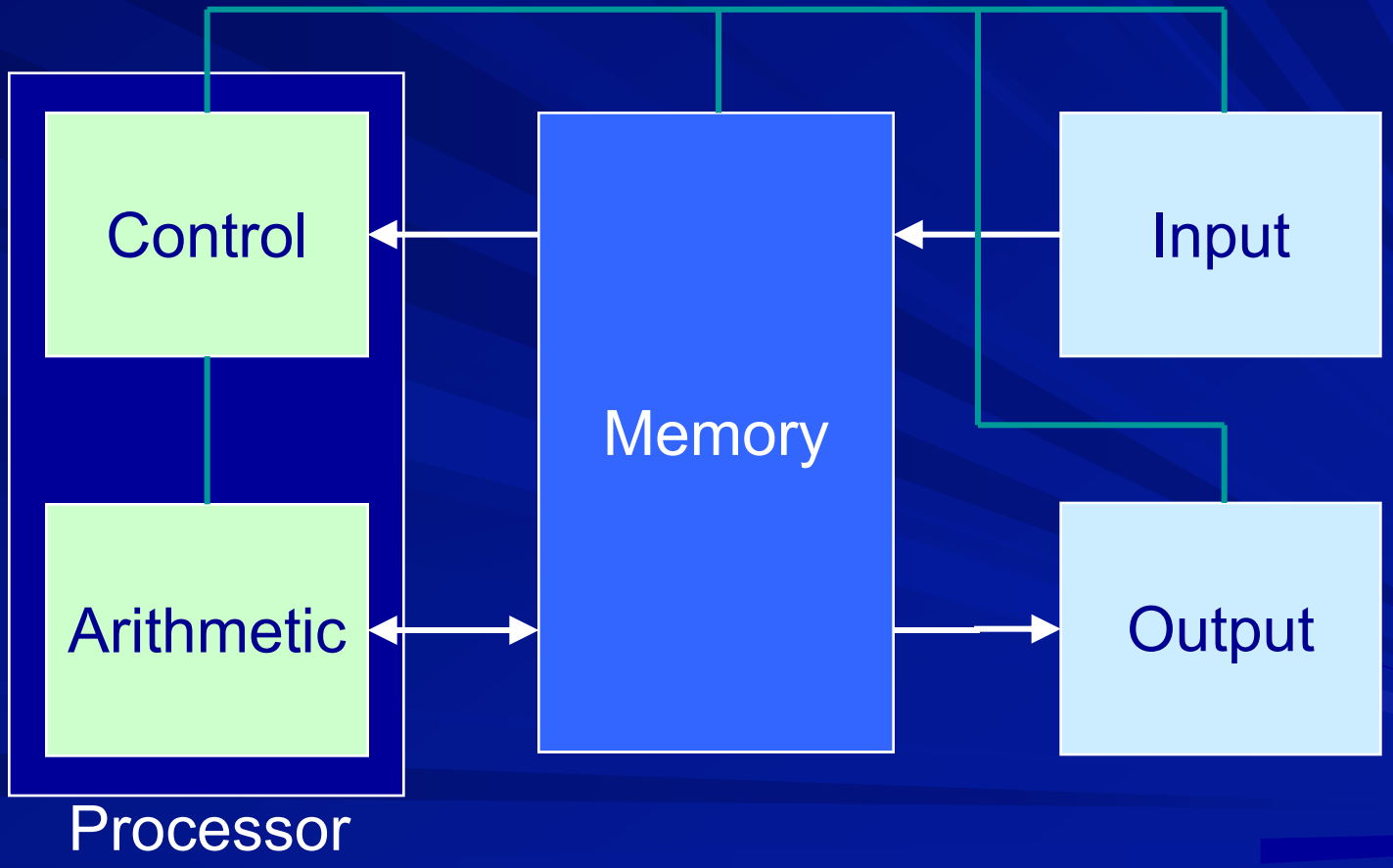
overall, what instruction needs in one case is different from what it needs in the other case, hence actual measure should always be execution time which is N*CPI/Cf and not only CPI

Thanks

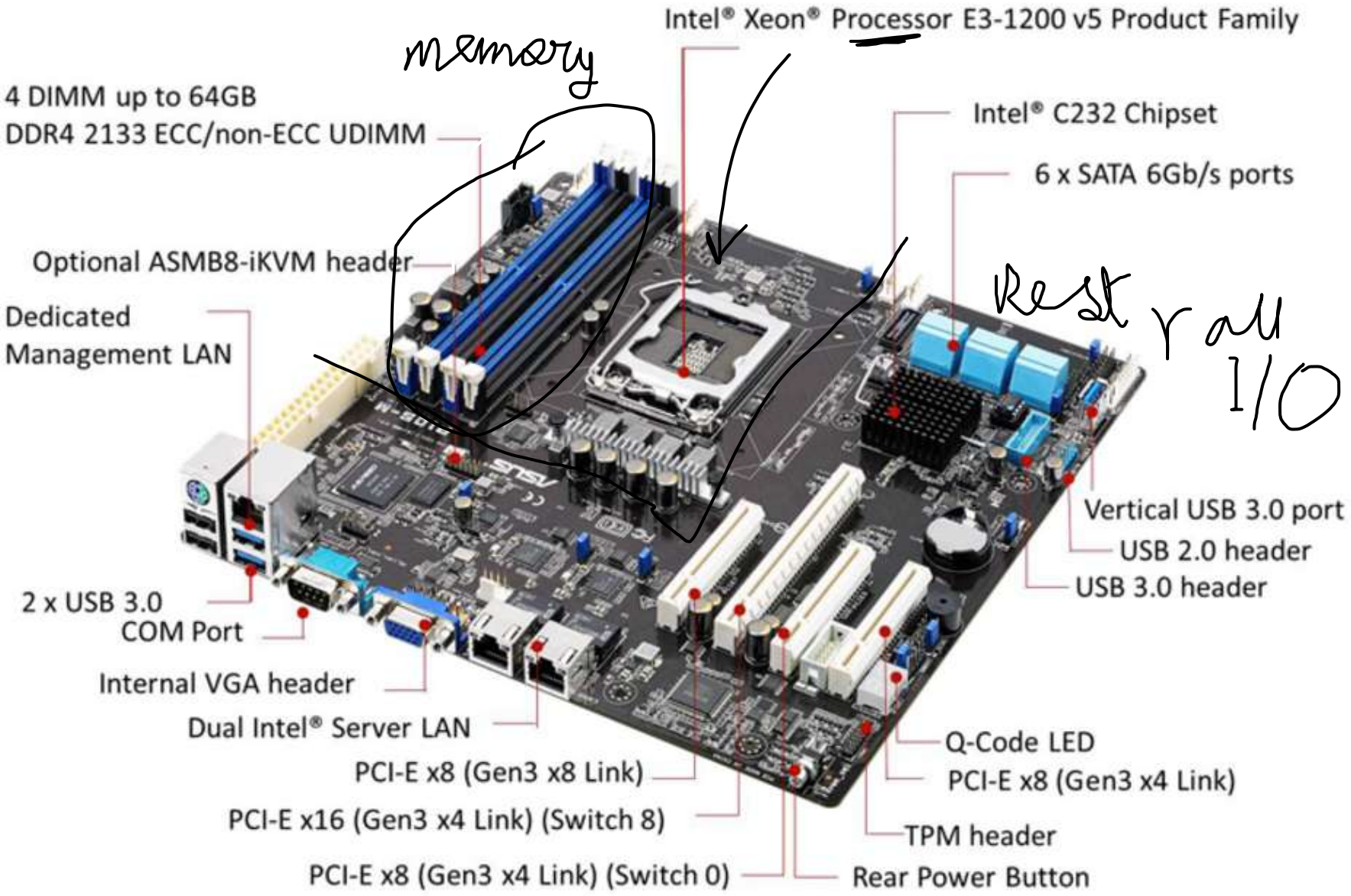
COL216
Computer Architecture

Memory Organization:
Hierarchy
7th March 2022

Computer System: Typical Block Diagram

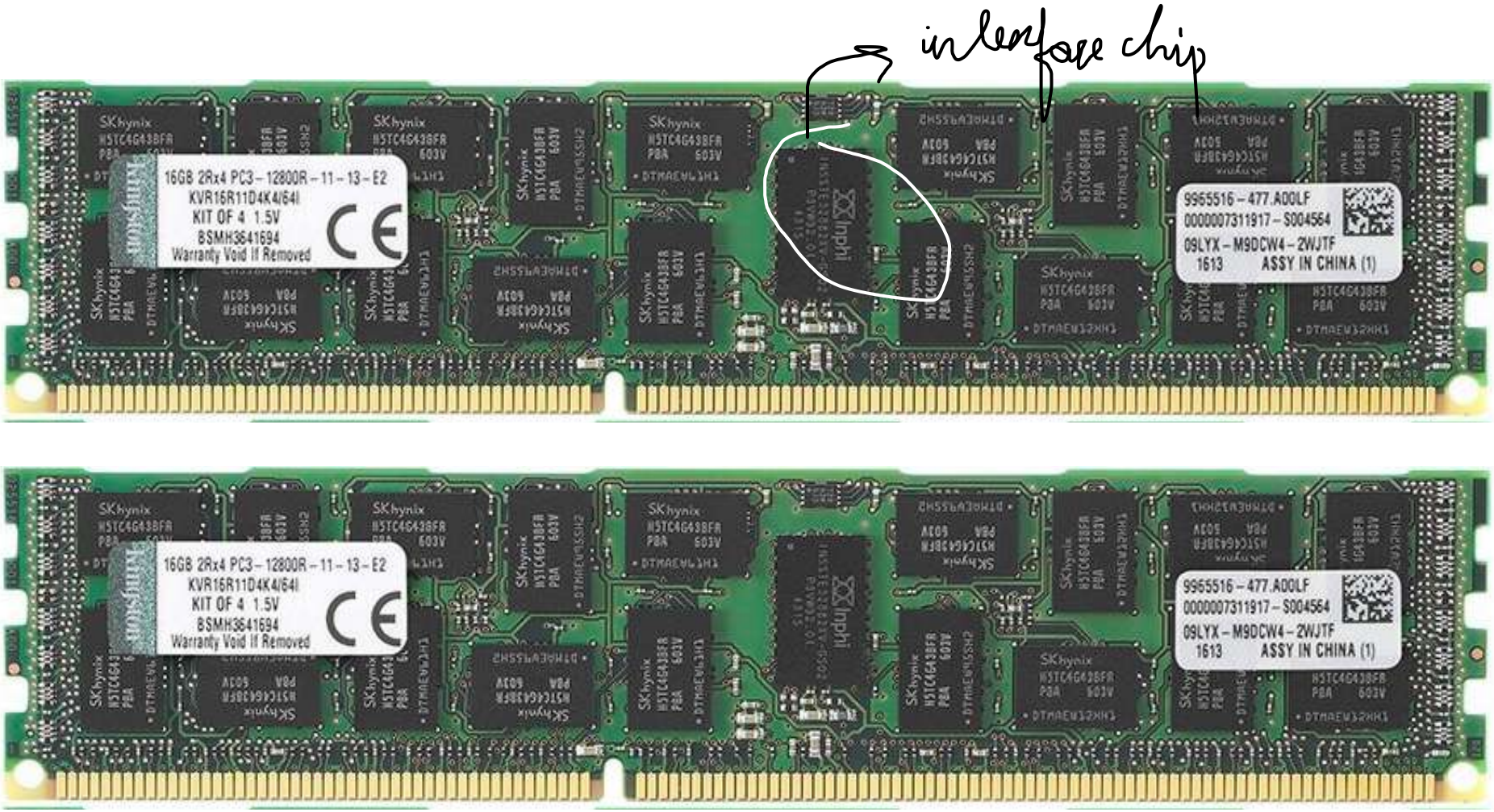


A server board

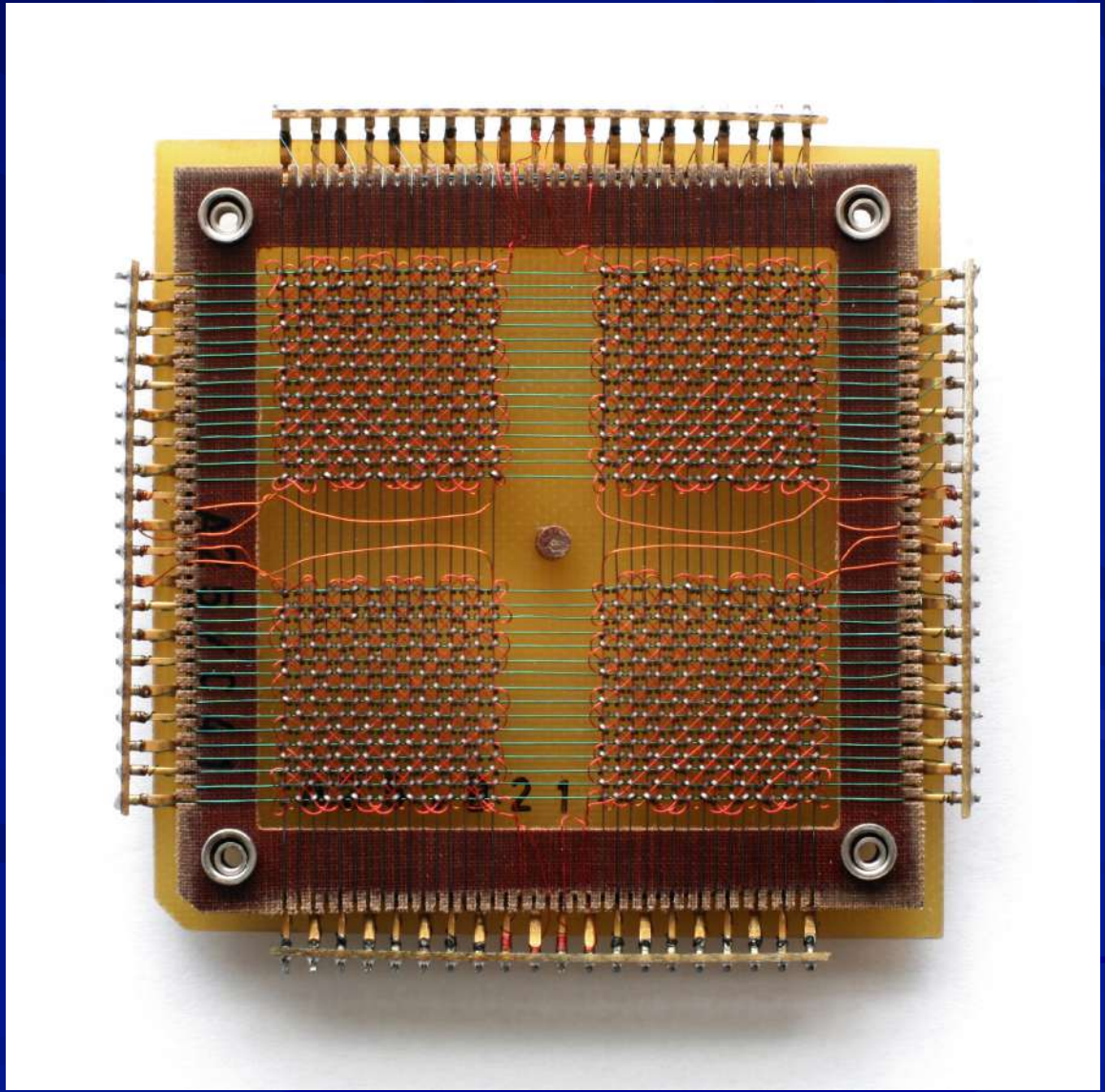


Memory Modules

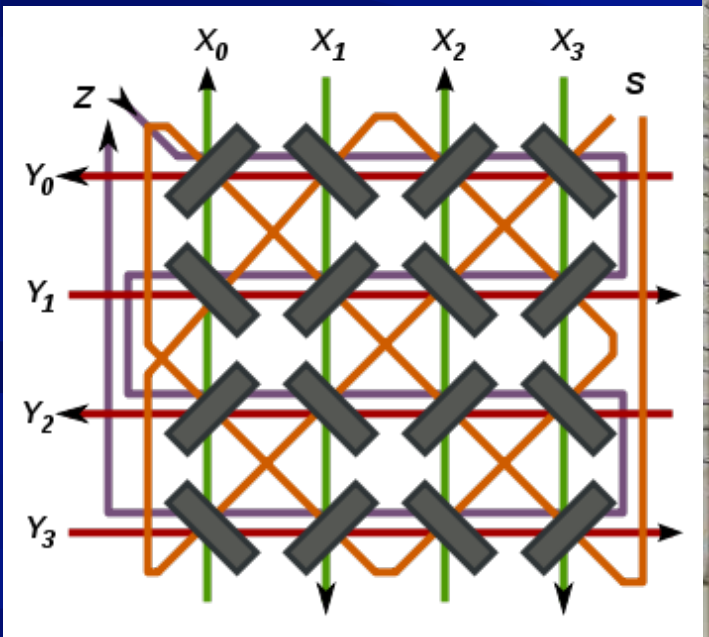
8 mem chips 8 bits + 1 parity bit

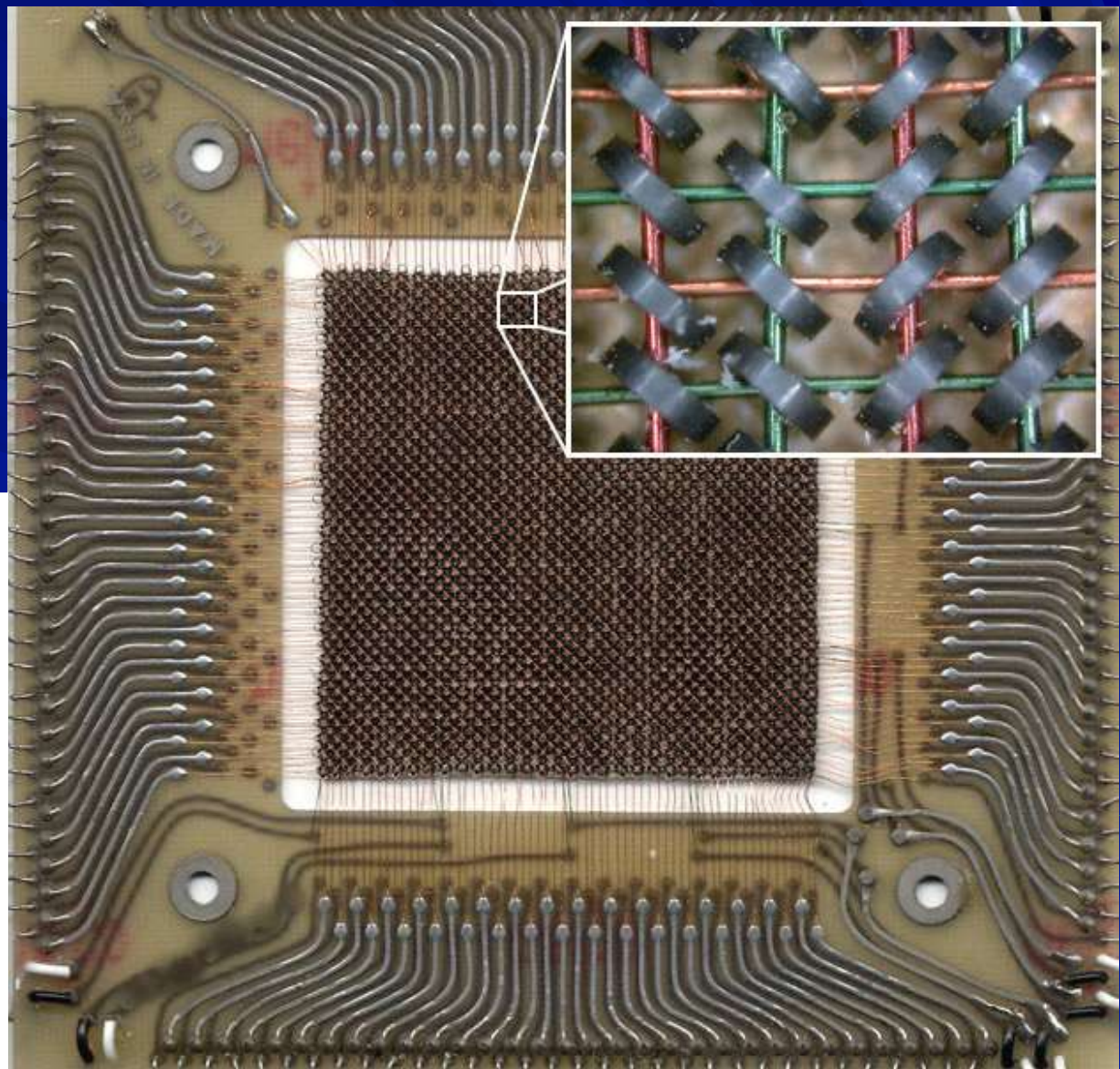


Ferrite core memory



Ferrite core memory





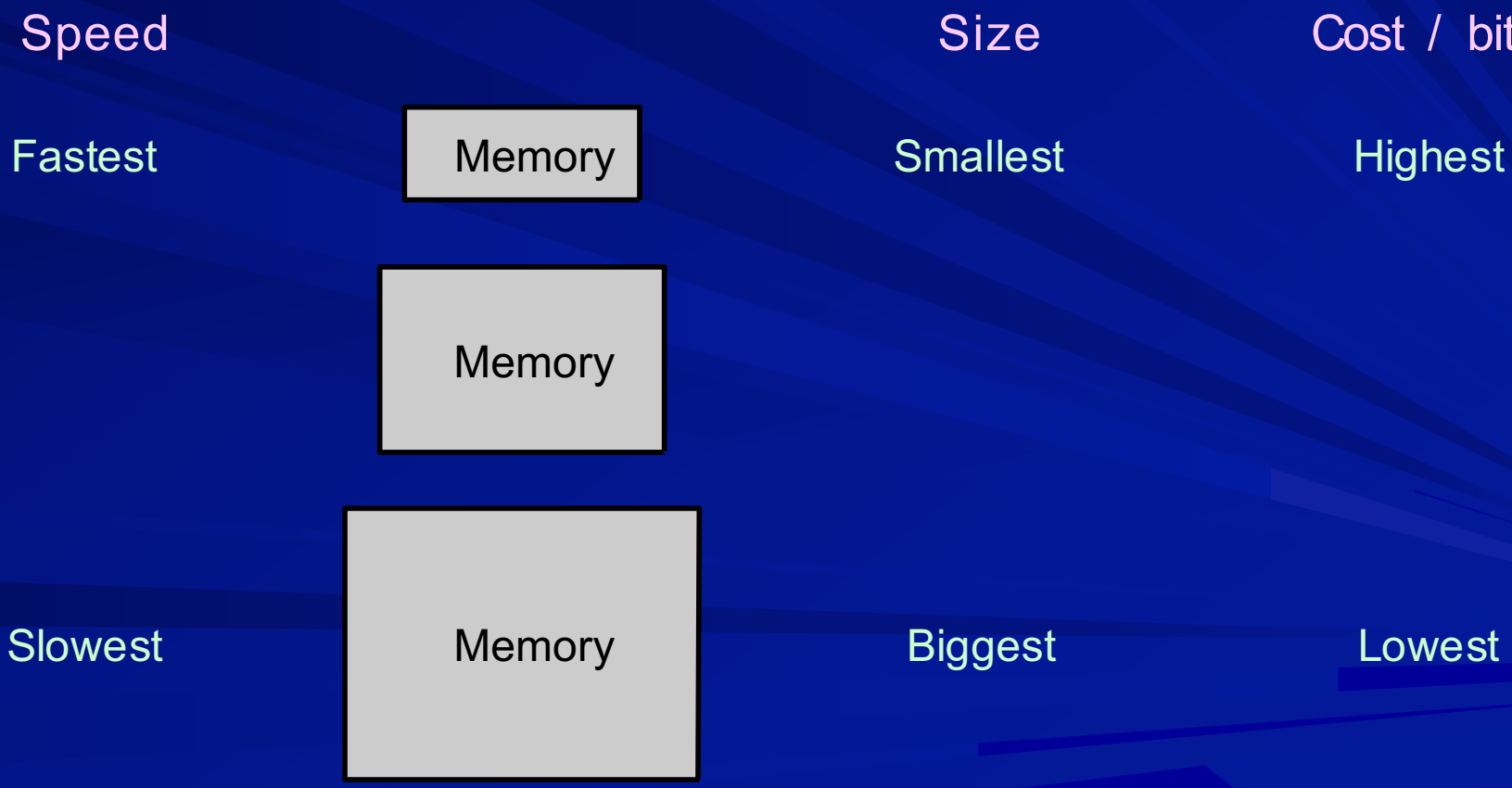
Memory Technologies

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

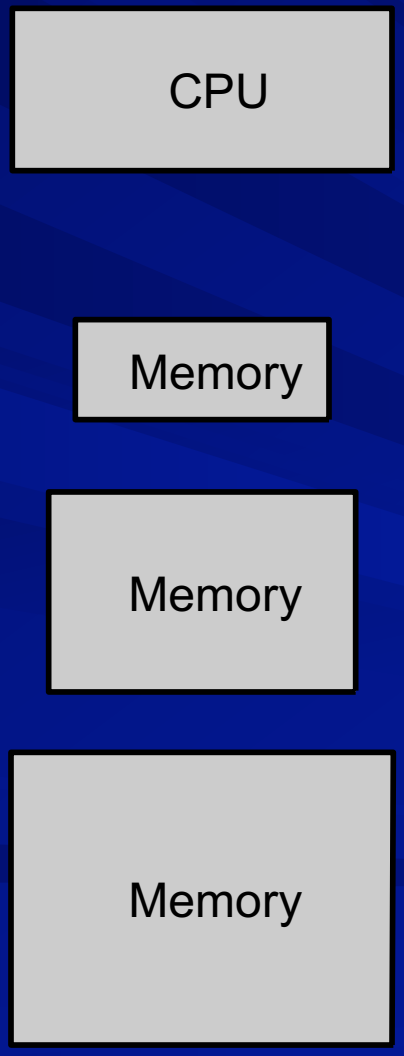
Random Access

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 when turned off, info is lost
- Separate from processor chip (off-chip)

Cache Memory

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

Cache Memory

- 1 to 3 levels
  - L1 : split (instruction and data mem are separate) L2 : unified L3 : unified
  - L1 : private L2 : priv/sh L3 : shared

each of the 8 cores would have separate L1 cache, whereas L3 is always 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

1 TB = 10^9 KB

Secondary Memory

- Flash or Disc drive

SSD is solid state device which is semiconductor while HDD is magnetic

- Not random access
- Non-volatile

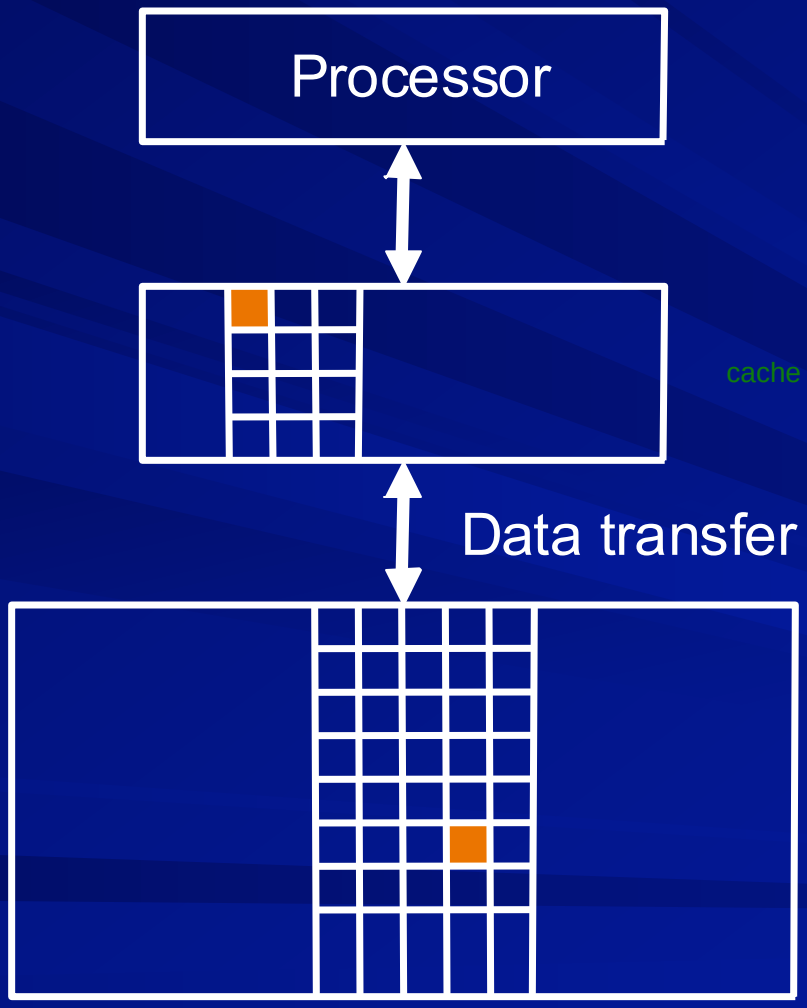
input/output

- Like a peripheral device

Back-up Memory

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

How CPU accesses hierarchy?



access → hit

if info is there in cache then hit else miss

access → miss

unit of transfer = block

block of data is transferred from main mem to cache

Why does the hierarchy work?

- Temporal Locality
- Spatial Locality

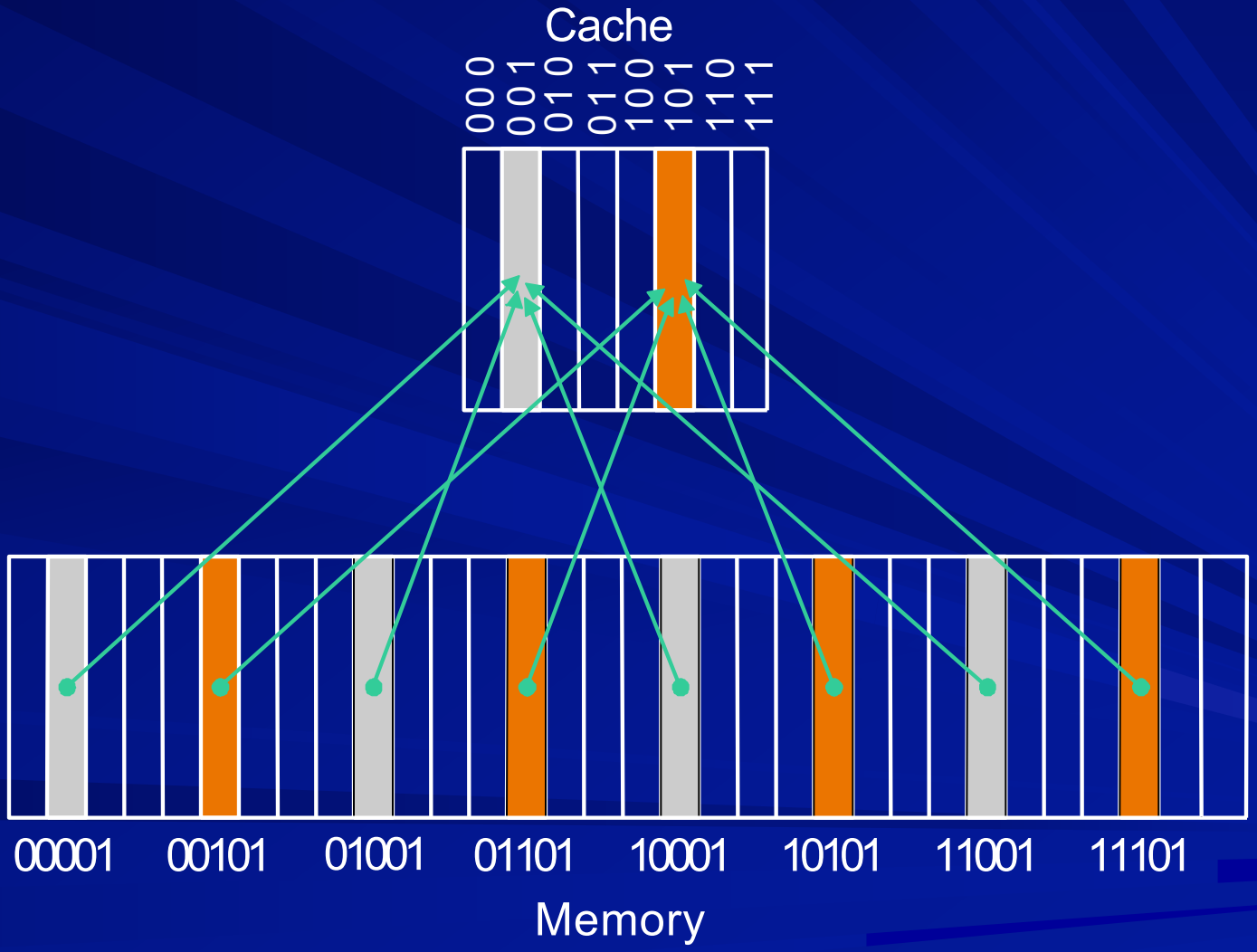
Why does the hierarchy work?

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

nearby data

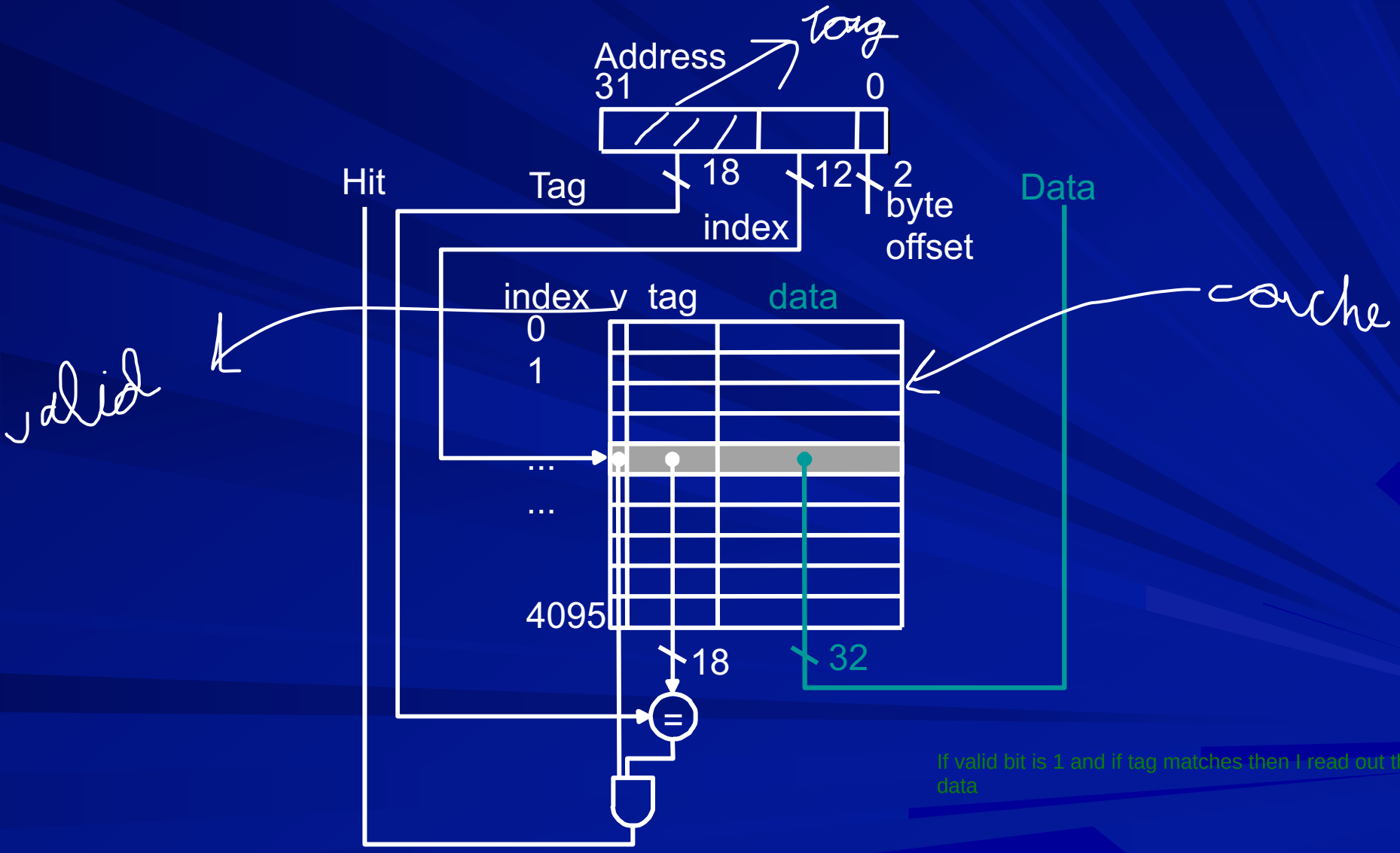
Direct mapped cache

001 can have any of the 4 data in main memory ending with 001



when we start our computer, then we hv all information in our main memory so then we hv all valid bits as 0

Cache access mechanism



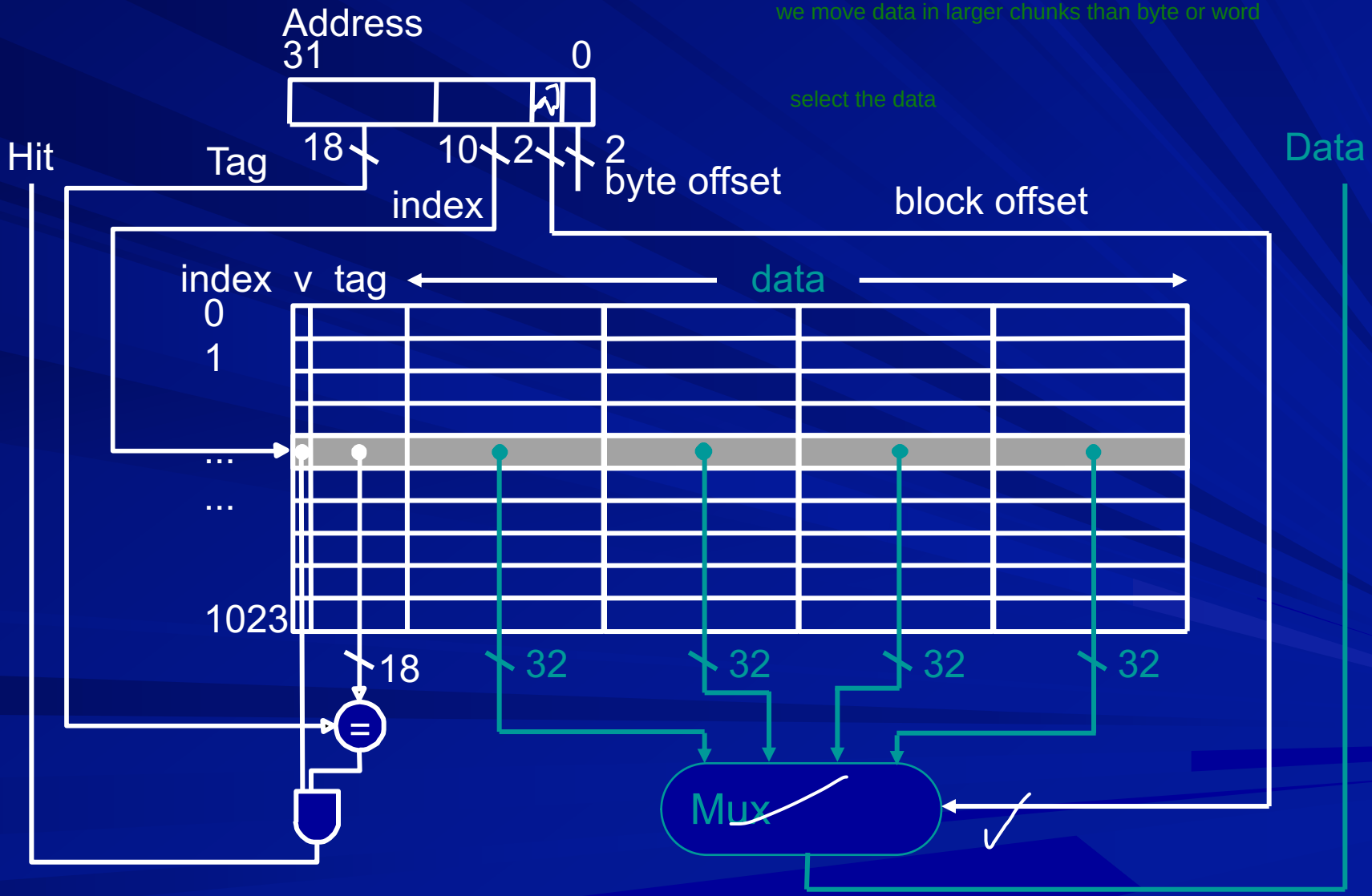
If valid bit is 1 and if tag matches then I read out the data

direct mapped

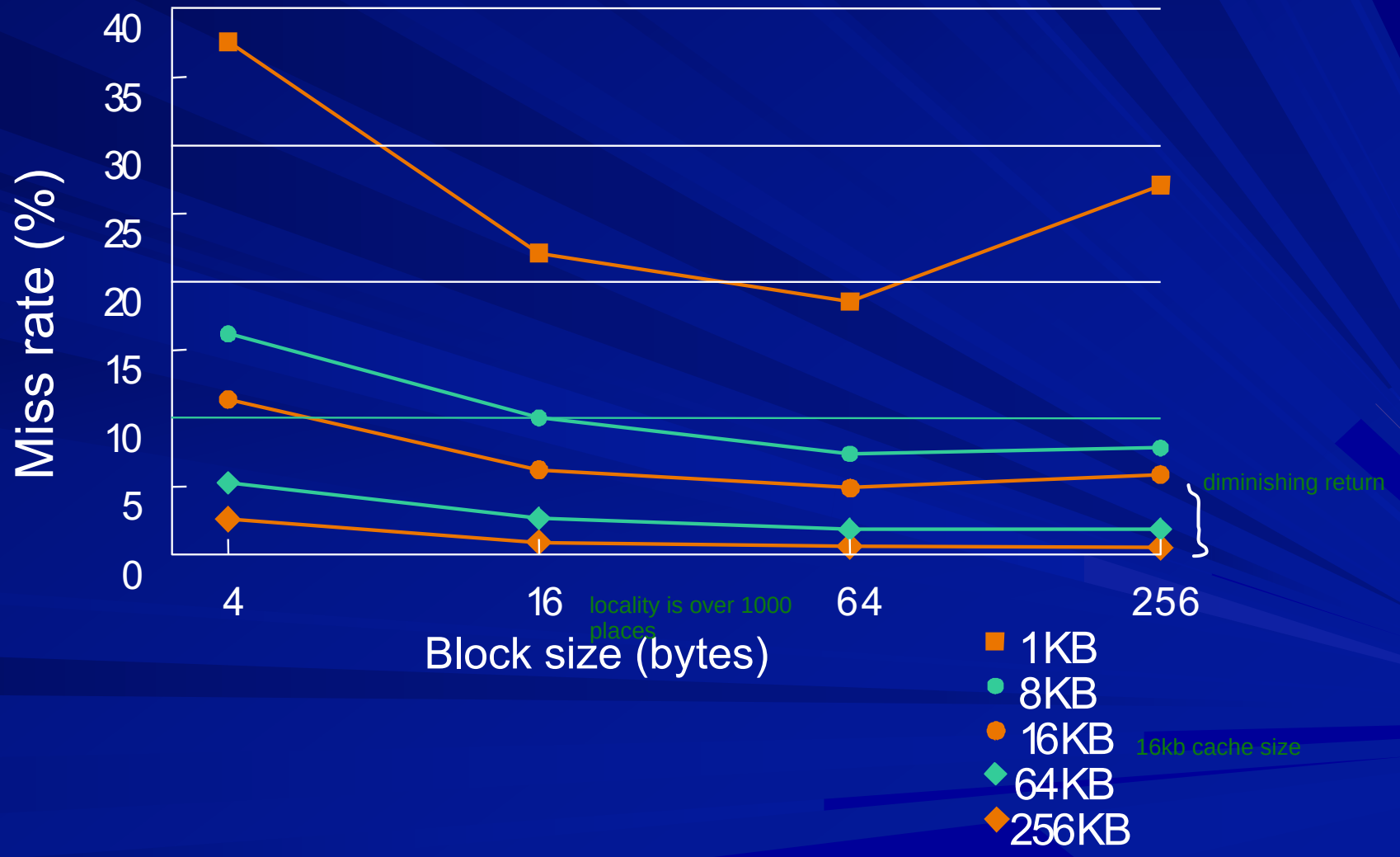
Cache with 4 word blocks

we move data in larger chunks than byte or word

select the data

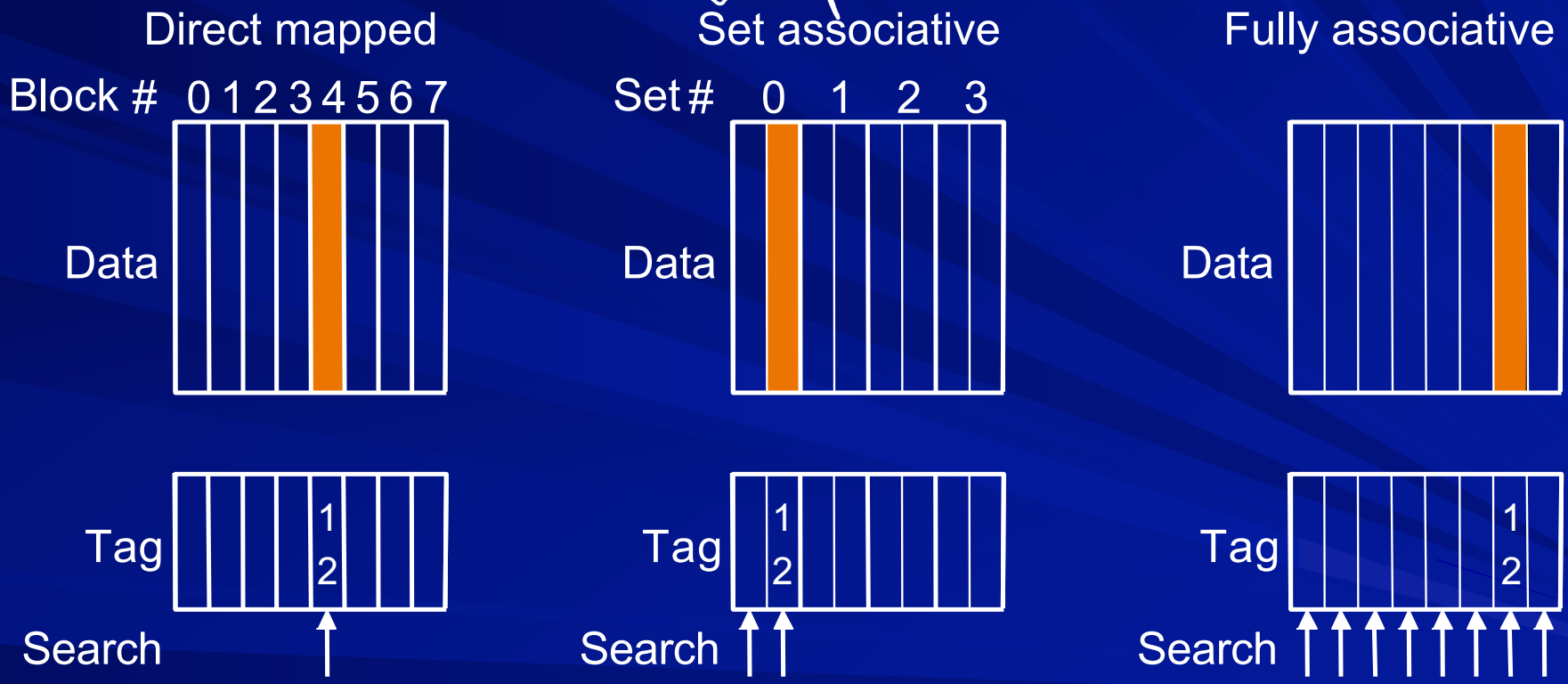


Miss rate vs block size



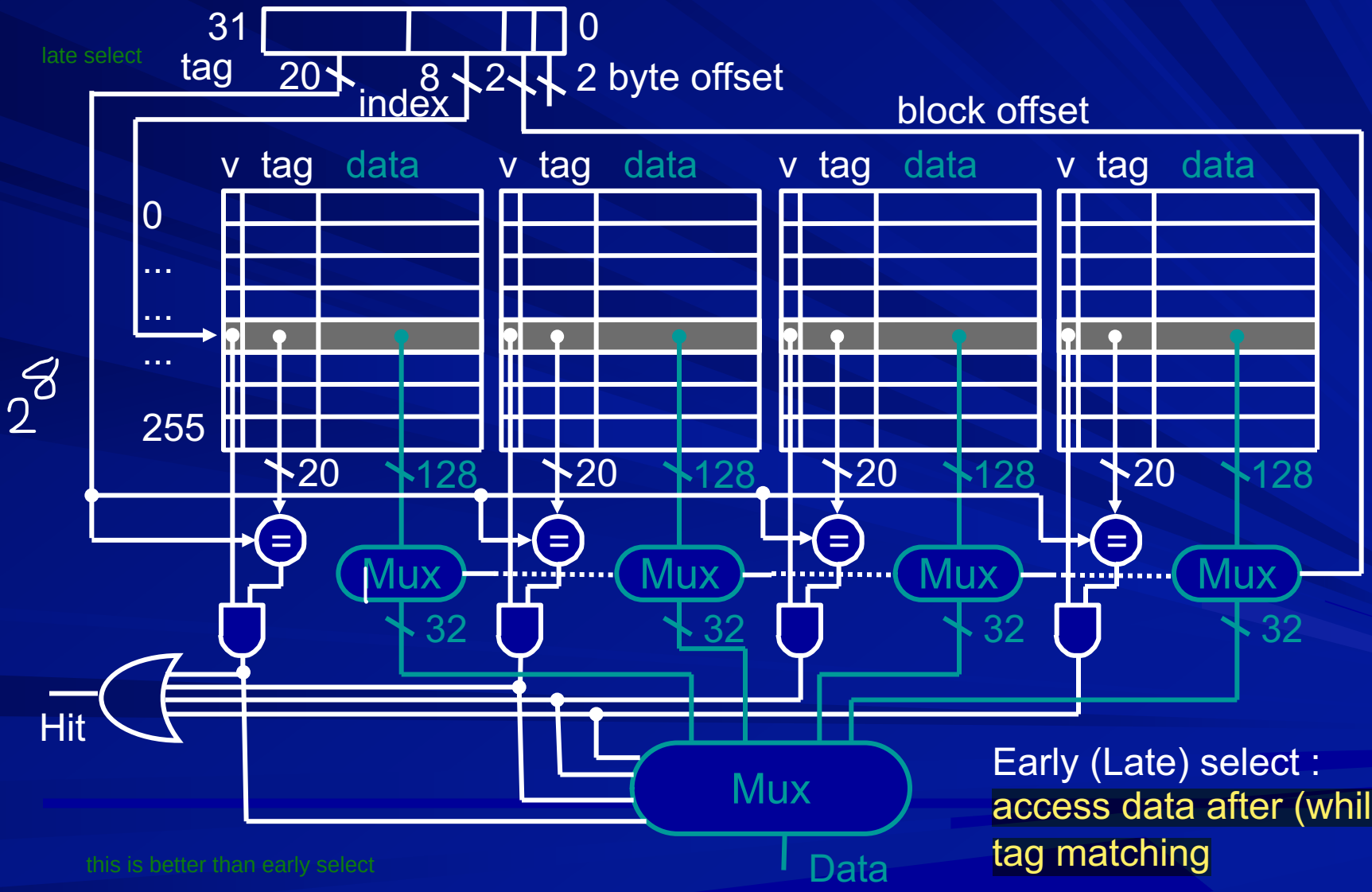
More flexible block placement

practical



in one go, we check all the places otherwise if sequential then purpose of cache is lost

4-way set associative cache



Early (Late) select :
access data after (while) tag matching

Sizes and bits (Direct mapped cache)

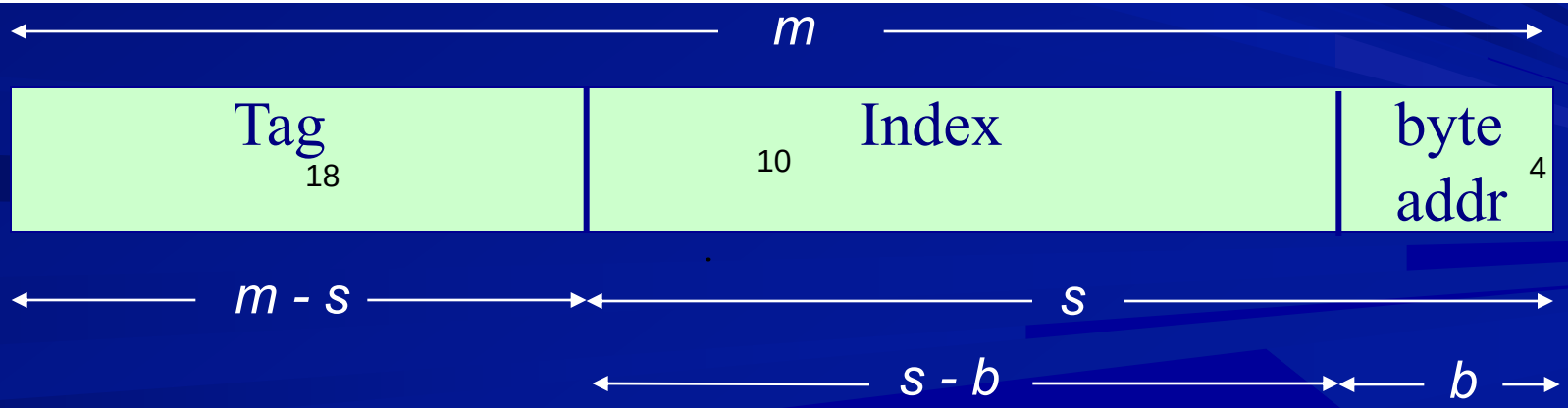
Memory = $M = 2^m$ bytes m = 32

Cache = $S = 2^s$ bytes s = 32-18 = 14

Block = $B = 2^b$ bytes b=4 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. = $A = 2^a$

here number of index bits will reduce a=4 (in our example)

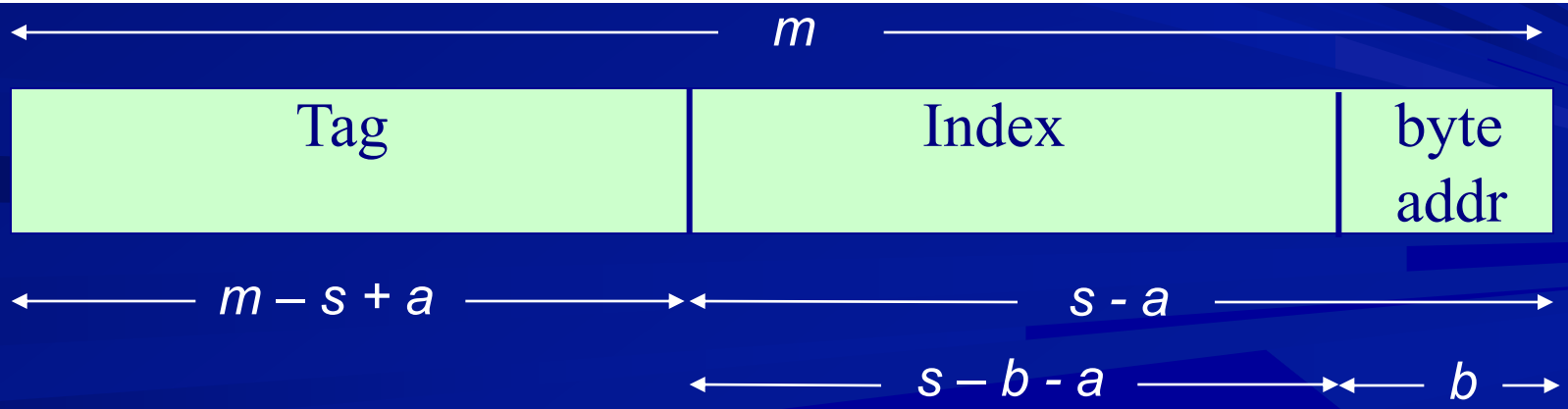
No. of sets = $\frac{S}{A \cdot B}$

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 = $(m - s + a) \times \frac{S}{B}$

overhead in terms of tag bits



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

suppose block size is 16 words and we see 6th word is missing then we start fetching from memory starting from missing word and in a wrap around manner

Which block to replace?

when cache is full, we may need to replace some block so 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

fetch a block when there is a miss

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

There may be a branch and that might lead to a wastage of the prefetched block

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

COL216
Computer Architecture

Memory Organization:

Cache performance

10th March 2022

Cache Policies

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

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?

Write Policies

- Write Back
  - Do not write in memory immediately
  - Write a block in memory when it is getting evicted from cache
- Write Through
  - Write word in memory immediately
  - Two choices in case of a write miss
    - Write Allocate
    - No Write Allocate

Write Policies - Comparison

rd hit - policy nothing

	WB	WTWA	WTNWA
wr hit	- (write to cache, no change in main mem)	word write (into mem)	word write
wr miss	block rd* (when it is a write miss, we read the entire block from mem b/c spatial locality)	block rd word write	word write
rd miss	block rd*	block rd	block rd
Miss rate	lower	lower	higher

* Write back displaced block if "dirty bit" set

dirty - inconsistent with main mem
if dirty block is getting displaced then we need to update main mem

Write Policies : Timings

- Write time depends upon
  - number of blocks transferred
  - block transfer time
  - number of words transferred
  - word transfer time
- Timings as seen by the CPU vs timings as seen by BUS
  - CPU time can be minimized by using a write buffer

Write Buffers

- Processor writes into a buffer and proceeds without waiting
- Transfer from buffer to memory takes place in background
- Possible with WT as well as WB
- What happens if a read comes up for data still in buffer?

Cache Performance

Average memory access time =

Hit time + Miss rate * Miss penalty

Mem stalls / Instr =

Miss rate * Miss Penalty * Mem accesses / Instr

Performance Improvement

- Reducing miss penalty
- Reducing miss rate
- Reducing hit time

Performance Improvement

- Reducing miss penalty
- Reducing miss rate
- Reducing hit time

Miss rate depends on

- cache size, block size, associativity, policies

larger the associativity, lesser is the miss rate

Miss penalty depends on

- Memory access time

higher block size => higher miss penalty

- cache – memory interface

Types of misses

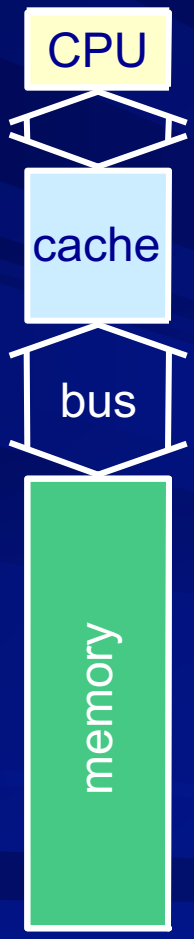
- compulsory miss — empty cache - unavoidable miss
- capacity miss
- conflict miss — data needed might have got displaced

for an individual, we cannot say capacity or conflict miss

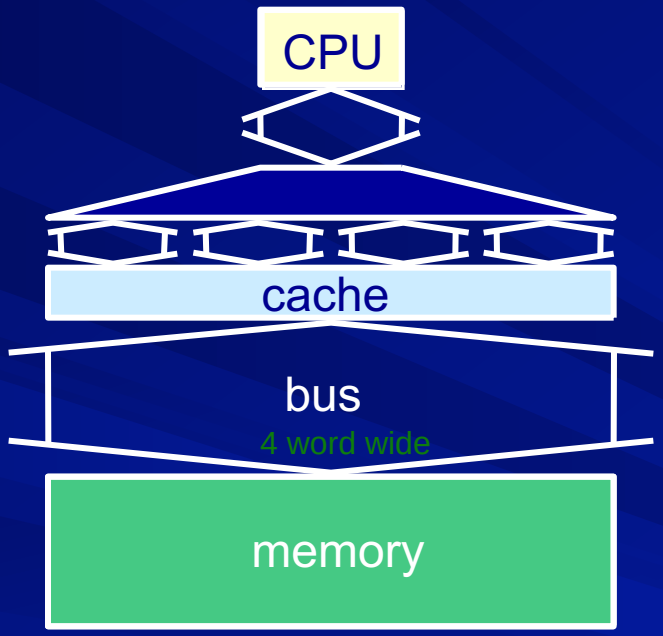
overall, say we hv 90 misses in fully associative cache and 100 in set associative cache then 90 are capacity misses

fully associative we can place the block from main mem anywhere while in set associative we place the block anywhere in a set, which set is decided by the last 4 bits of the address.

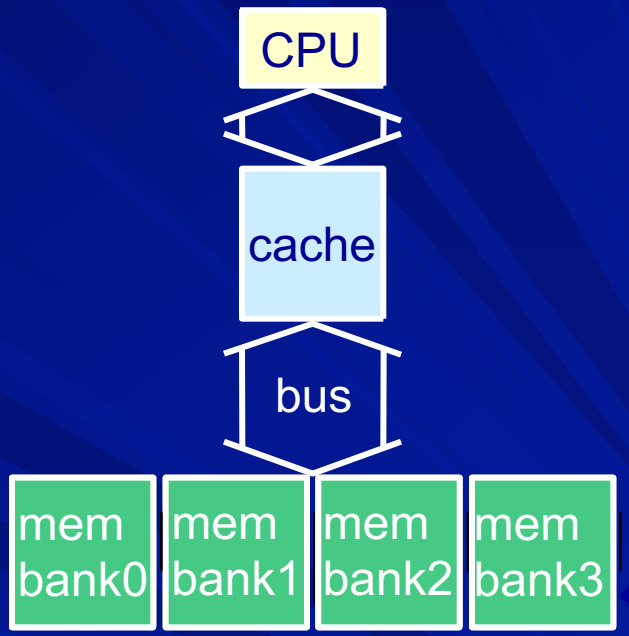
Transferring blocks to/from memory



a. one word wide memory

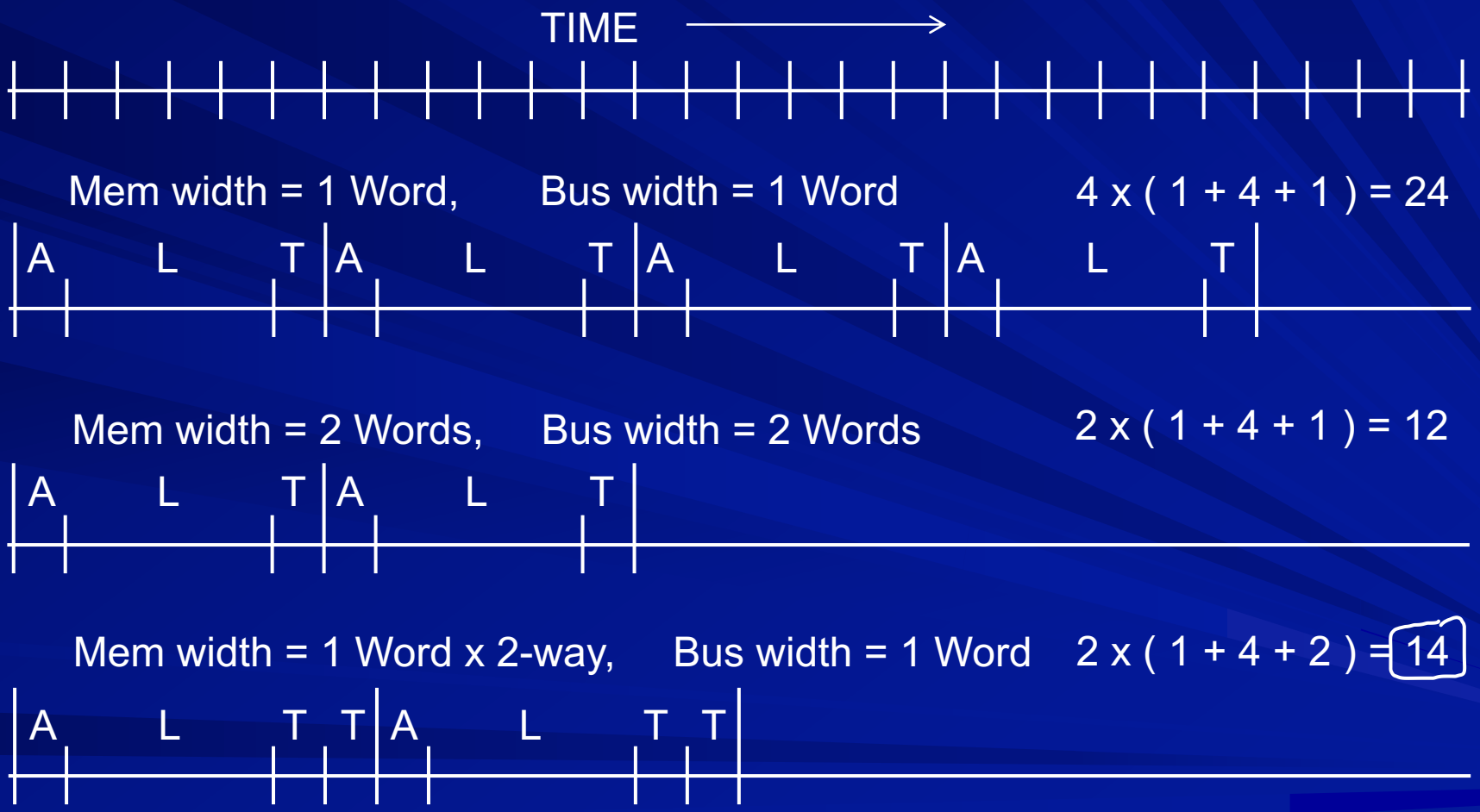


b. four word wide memory



c. interleaved memory

Miss penalty



bus width is 1 word so transfer of 2 words have to be done hence 2 (1 + 4 + 2*1) = 14

having a wide bus is difficult so best to have a memory that is 2-way and bus that is 1 word wide

memory is 4 word wide - 4 way link

Miss penalty example

- 1 clock cycle to send address
- 15 cycles for RAM access
- 1 cycle for sending data
- block size = 4 words

Compute miss penalty

Answer:

local arrangement to update address

case (a): 4 (1 + 15 + 1) = 68 or 1 + 4 (15 + 1) = 65

case (b): 1 + (15 + 1) = 17

case (c): 1 + 15 + 4 x 1 = 20

reading a row entirely is called a page mode

DRAM with page mode

- Memory cells are organized as a 2-D structure
- Entire row is accessed at a time internally and kept in a buffer
- Reading multiple bits from a row can be done very fast
  - sequentially, without giving address again
  - randomly, giving only the column addresses

Performance analysis example

CPI_{eff} = CPI + Miss rate * Miss Penalty * Mem accesses / Instr

CPI = 1.2 Miss rate = 0.5%

Block size = 16 w

memory is 4 words wide and block size of cache is 16 words, hence 4 memory reads need to be done,

Miss penalty??

Assume mem access / Instr = 1

Miss penalty calculation

Data / address transfer time = 1 cycle

Memory latency = 10 cycles

a) Miss penalty = 16*(1+10+1) = 192

b) Miss penalty = 4*(1+10+1) = 48

c) Miss penalty = 4*(1+10+4*1) = 60

a) memory is 1 word wide, 10 cycles to read, bus is 1 word hence total 16 ( ) = 192
b) memory is 4 word wide, 10 cycles, bus is 4 word wide, hence 4 ()
c) 4(4*1 for bus is 1 word wide)

overall
cpi_eff = cpi + miss rate * miss penalty * mem accesses/instr

Back to CPI calculation

CPI_{eff} = 1.2 + .005 * miss penalty * 1.0

a) 1.2 + .005 * 192 * 1.0 = 1.2 + .96
= 2.16

b) 1.2 + .005 * 48 * 1.0 = 1.2 + .24
= 1.44

c) 1.2 + .005 * 60 * 1.0 = 1.2 + .30
= 1.50

What makes cache work?

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

Locality Example

Code 1

```
for ( i = 0; i < 8000; i++)
    for ( j = 0; j < 8; j++)
        A [ i ] [ j ] = B [ j ] [ 0 ] + A [ j ] [ i ];
```

here B has temporal locality since we access B[j][0] multiple times and here gaps are larger so larger chances of miss

Code 2

```
for ( j = 0; j < 8; j++)
    for ( i = 0; i < 8000; i++)
        A [ i ] [ j ] = B [ j ] [ 0 ] + A [ j ] [ i ];
```

Arrays are stored row-wise in C, column-wise in Matlab

here A[j][i] has spatial locality

here A[i][j] there is no spatial locality when considering language C because we access wrt columns while in C arrays are stored wrt rows

here B has temporal locality since we access B[j][0] multiple(8000) times

Identify spatial locality and temporal locality

Is cache managed by HW or SW?

Check for hit or miss

Hit: Read/write cache

Miss: Possible actions

1. write back a block in memory
2. read a block from memory
3. wite a word in memory

Are these actions performed by hardware or by software?

Hit time, miss rate, miss penalty

How do these parameters vary with

1. cache size? higher cache size means lower miss rate but hit time is more
   improves capacity and conflict miss
2. degree of associativity? hit time could be more while miss rate goes down
   capacity and conflict miss
3. block size? large block size means large miss penalty, no effect on hit time
   improves compulsory misses, since with every miss, we get a whole block
4. memory access time?
5. degree of memory interleaving?

Indicate the type of miss that is affected in each case.

Cache comparison example

data misses -

Cache	mapping	block size	I-miss	D-miss	CPI
1	direct	1 word	4%	8%	2.0
2	direct	4 word	2%	5%	??
3	2-way s.a.	4 word	2%	4%	??

Miss penalty = 6 + block size block size is number of words in the block

50% instructions have a data reference 50% r DT

Solution:

Stall cycles: cache1: 7*(.04+.08*.5)=.56

cache2: 10*(.02+.05*.5) = .45

cache3: 10*(.02+.04*.5) = .40

Solution continued

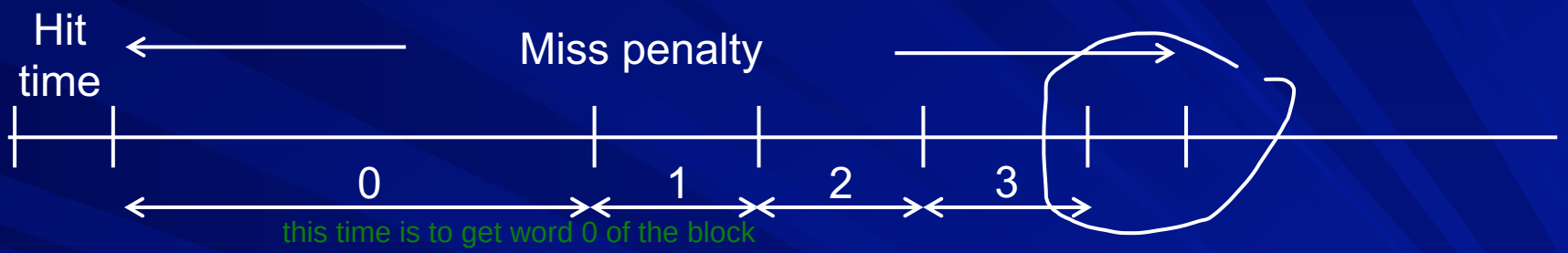
Cache	CPI	clock period	time/instr
1	2.0	2.0	4.0
2	2.0 - .56 + .45 = 1.89	2.0	3.78
3	2.0 - .56 + .40 = 1.84	2.4	4.416

Hit time, miss rate, miss penalty

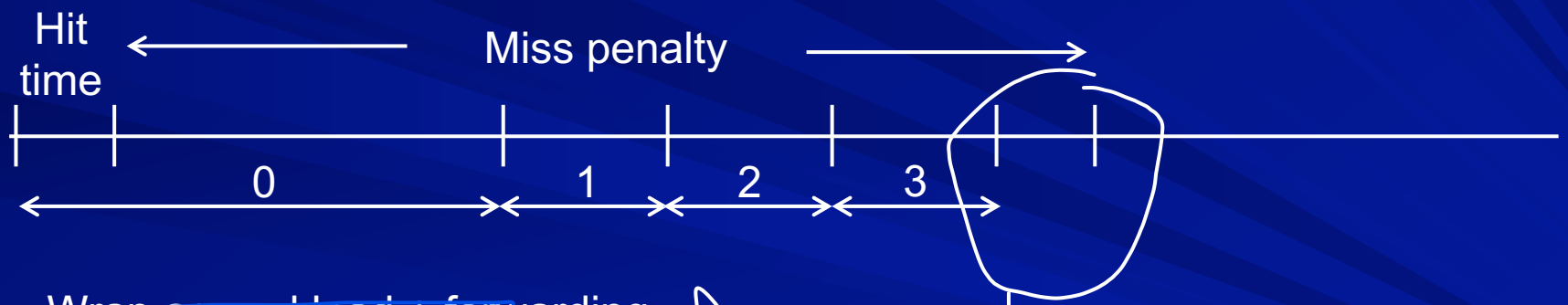
How do these parameters vary with

1. read policy?
2. load policy?
3. replacement policy?
4. fetch policy?
5. write policy?

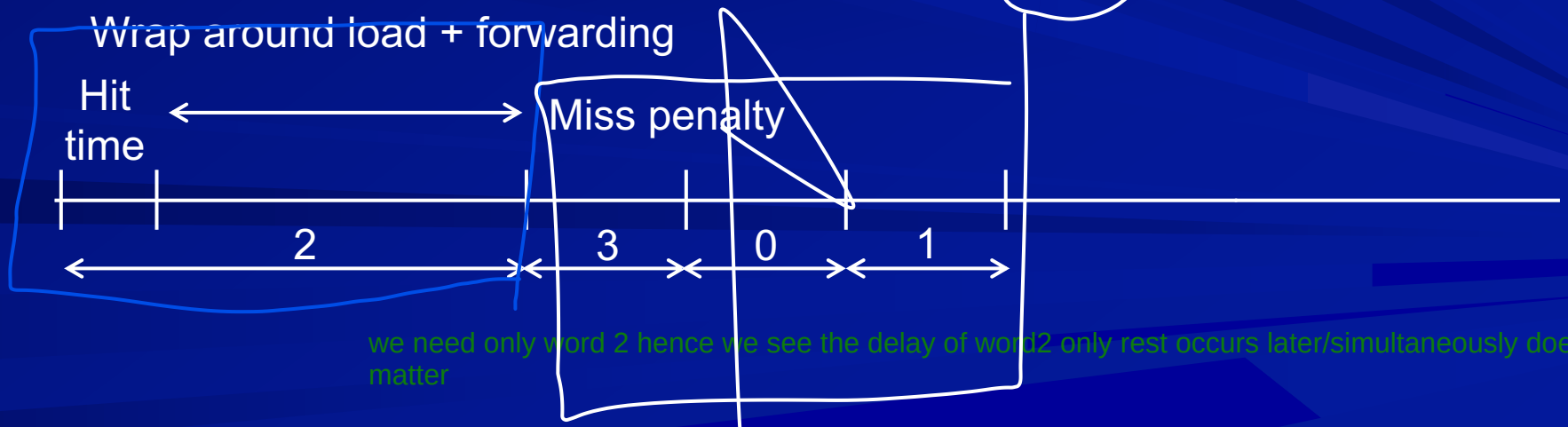
Miss penalty



Concurrent read



Wrap around load + forwarding



Cache plus various buffers

- Buffer for block(s)/word(s) waiting to be written into memory
  - write buffer
- Buffer for block(s) fetched in anticipation
  - prefetch buffer
- Buffer for displaced blocks waiting for re-admission or permanent disposal
  - victim cache

when new victim cache comes in, we dispose off old victim cache

Which write policy works better?

- Clustered and frequent writes
  - get the block into cache
  - do many writes into it
  - write back
- Sparse writes
  - just write into main memory
- Sparse writes followed by reads
  - get the block
  - write into both

during iteration of an array

Improved buffer for write through

in general, in write through cache we write a word into memory instead of a block but in case of improved buffer, we write block to memory i.e. from buffer to memory but we write word from cpu to buffer, so this helps.

- Effective for Write Through cache when there are multiple writes to same block
- Buffer size = 1 block, rather than 1 word
- Writes belonging to same block are collected in buffer
- Write from buffer to memory when
  - Buffer gets full or
  - Write for a different block arrives

Split cache vs. Unified, cache

instr vs data cache

Compare

1. access conflicts — split better
2. space utilization — unified better
3. cost. — unified better

On-chip cache vs. off-chip cache

having cache on the same chip as processor

Compare

1. access time on-chip better
2. capacity. off-chip better

till this, we had only 1 level cache performance now we go to 2-level cache

miss in 1st level, search in 2nd level

2-Level Cache Performance

Average memory access time =

Hit time + Miss rate * Miss penalty

Mem stalls / Instr =

Miss rate * Miss Penalty * Mem accesses / Instr

Average memory access time = Hit time +

miss penalty1 is much lower than miss penalty2

Miss rate1 * Miss penalty1 +

Miss rate2 * Miss penalty2

miss rate2 < miss rate1

Mem stalls / Instr = (Miss rate1 * Miss penalty1 +

Miss rate2 * Miss penalty2) *

Mem accesses / Instr

Miss rates in 2 level cache

Consider 2 level cache

L1 miss rate = no. of L1 misses/ no. of requests at L1

L2 miss rate = no. of L2 misses/ no. of requests at L2?

- Global Miss rate
  - no. of misses / no. of requests, on the whole
- Solo Miss rate
  - miss rate if only L2 cache was present
- Relationship among these?

global miss rate = solo miss rate when L2 is superset of L1

Inclusive & Exclusive caches

- Inclusive:

  Every information that is there in L1 cache is also in L2 cache should not displace anything from L2 that is in L1

  Not vice versa (of course)

- Exclusive:

  No information is duplicated

- Neither inclusive, nor exclusive

What is required to ensure the above properties?

2-level cache example

CPI with no miss = 1.0 Clock = 500 MHz

Main mem access time = 200 ns

cycles to be found in case of miss penalty

Miss rate = 5%

Adding L2 cache (20 ns) reduces miss to 2%. Find performance improvement.

Solution:

Miss penalty (mem) = 200/2 = 100 cycles

Effective CPI with L1 = 1+5%*100 = 6

Solution continued

Miss penalty (L2) = 20/2 = 10 cycles

Total CPI = Base CPI + stalls due to L1 miss
+ stalls due to L2 miss

= 1.0 + 5% * 10 + 2% * 100

= 1.0 + 0.5 + 2.0 = 3.5

Performance ratio = 6.0/3.5 = 1.7

Question

Why many modern processors have 3 levels of cache, while a decade ago it was common to have only one level?