CMPE 200
Computer Architecture & Design

Final Exam Review

Haonan Wang

SJSU SAN JOSÉ STATE UNIVERSITY

Final Exam Composition

- **Multiple-choice Single-answer: 3 points * 10 = 30**
- **Multiple-choice Multiple-answer: 4 points (wrong option -4, missing option -2) * 5 = 20**
- **True or False: 2 points * 9 = 18**
- **Questions & Answers: 8 points (steps 6 + answer 2) * 4 = 32**
- **Bonus question: 10 points * 2 = 20**

Total = 30 + 20 + 18 + 32 + 20 = 100 + 20

- If you get more than 100 points, the excessive points will be used to improved your overall grade.

Exam settings:

- Time: Thursday, Dec. 8th, 12:15 - 14:30 @ CLARK 222 with LockDown Browser
- Closed book (but you can print out the MIPS data card and use it if needed)
- Cheat sheet (handwritten): A4 paper * 2 allowed, pictures of all sides must be submitted before the exam
- Use of calculator and scratch paper allowed
- Double-check your laptop and turn off your cellphone before the exam
- Submit your answers on time (otherwise score will not be counted)!

Processor Interface Mechanisms

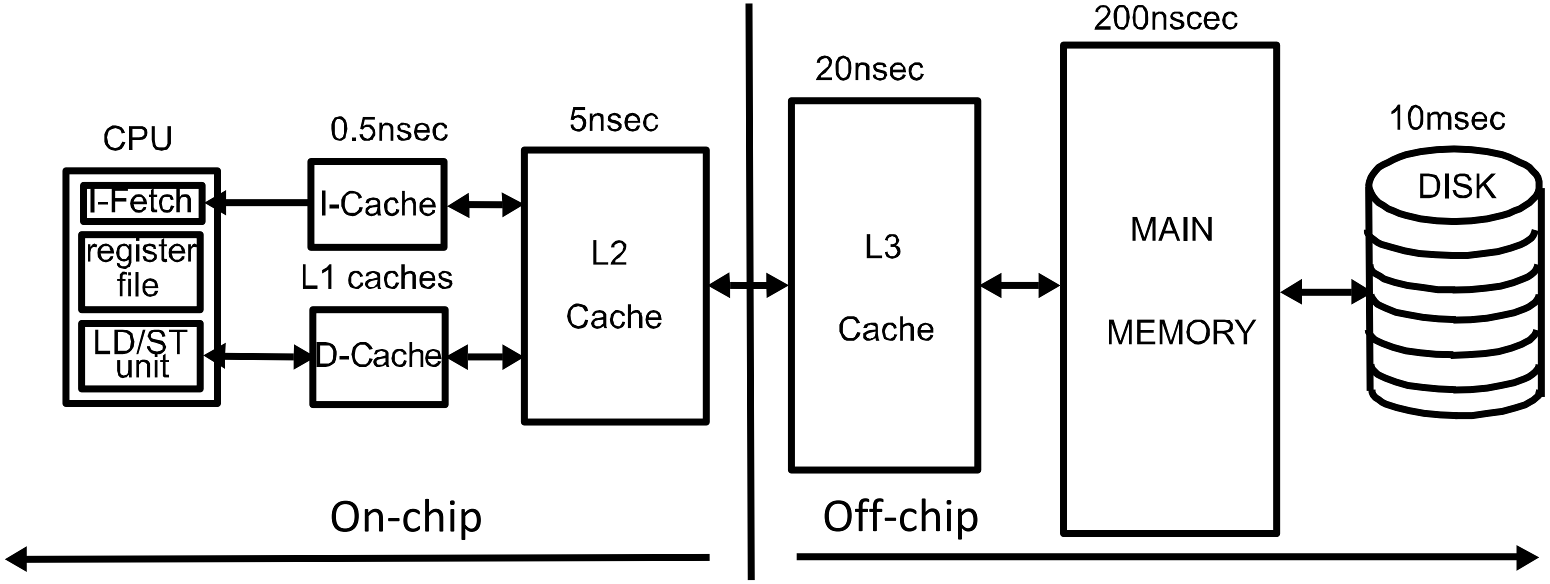
- **Two ways for processors to access peripheral devices:**
  - Port-mapped I/O (PMIO): isolated I/O, special I/O instructions, separate address spaces
  - Memory-mapped I/O (MMIO): one address space
- **MIPS processors use memory-mapped I/O**
  - Use load and store instructions to communicate with peripheral devices
  - For a MIPS processor that only supports lw and sw, all data transfer will be 32 bits

Processor Interface Mechanisms

- **Two ways to communicate with CPU:**
  - Polling: The CPU polls the device periodically
  - Interrupt: The I/O device calls the CPU actively
- **Two ways for data transfer between the memory and I/O:**
  - CPU controlled
  - Direct Memory Access (DMA)

Why?

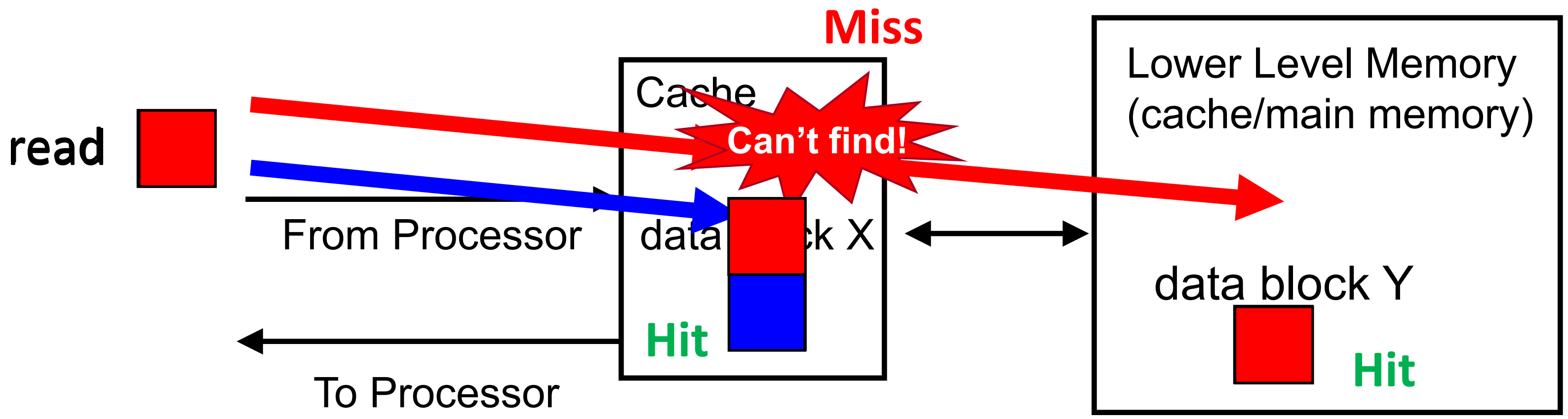
- **Two Types of Locality:**
  - **Temporal Locality** (Locality in Time): If an address is referenced, it tends to be referenced again (e.g., loops, reuse)
  - **Spatial Locality** (Locality in Space): If an address is referenced, neighboring addresses tend to be referenced (e.g., array, stack, etc.)



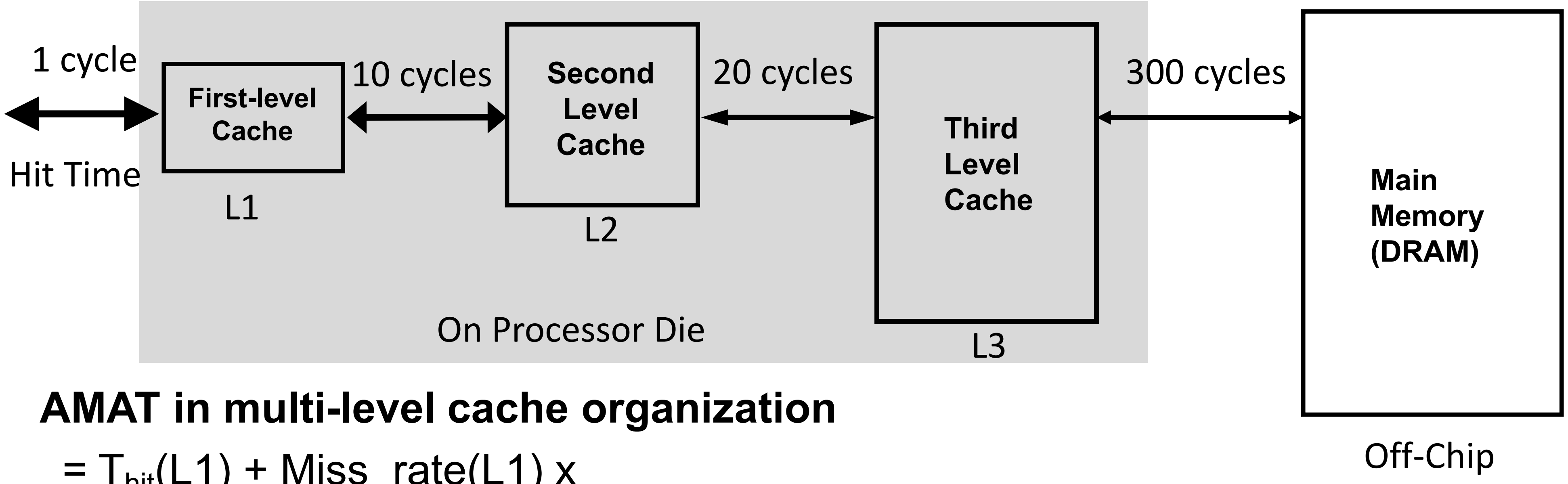
Cache Hit and Miss

- **Hit:** Data appears in some block of the cache
  - **Hit Rate:** # hits / total accesses on the cache
  - **Hit Time:** Time to access the cache

- **Miss:** Data needs to be retrieved from the lower level (and stored in cache)
  - **Miss Rate:** 1 - (Hit Rate)
  - **Miss Penalty:** Average delay in the processor caused by each miss



Reducing Penalty: Multi-Level Cache



- **AMAT in multi-level cache organization**

  = T_{hit}(L1) + Miss_rate(L1) x

  [T_{hit}(L2) + Miss_rate(L2) x

  { T_{hit}(L3) + Miss_rate(L3) x T(memory) }]

- **Example:**
  - Miss rate of L1, L2, L3 = 10%, 5%, 1%, respectively
  - AMAT = 1 + 0.1 x [10 + 0.05 x { 20 + 0.01 x 300 }] = 2.115 cycles

Vs. 31 cycles
14.7x speedup!

Measuring Performance with Caches

- **Assuming cache hit costs are included as part of the normal CPU execution cycle, then**

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{CP}$$

$$= \text{IC} \times (\underbrace{\text{CPI}_{\text{ideal}} + \text{Memory-stall cycles}}_{\text{CPI}_{\text{stall}}}) \times \text{CP}$$

Note: this is miss ratio with regard to all instructions = read ratio * cache miss rate

Memory-stall cycles come from cache misses (a sum of read-stalls and write-stalls)

$$\text{Read-stall cycles} = \text{read miss ratio} \times \text{read miss penalty}$$

$$\text{Write-stall cycles} = \text{write miss ratio} \times \text{write miss penalty} + \text{write buffer stalls}$$

For write-through caches, we can simplify this to

$$\text{Memory-stall cycles} = \text{miss ratio} \times \text{miss penalty}$$

Impacts of Cache Performance

- **Relative cache penalty increases as processor performance improves (faster clock rate and/or lower CPI)**
  - The memory speed is unlikely to improve as fast as processor cycle time. When calculating CPI_{stall}, the cache miss penalty is measured in *processor* clock cycles needed to handle a miss
  - The lower the CPI_{ideal}, the higher the impact of stalls

- **Example: A processor with a CPI_{ideal} of 2, a 100 cycle miss penalty, 36% load/store instructions, and 2% I$ and 4% D$ miss rates**

  Memory-stall cycles = $2\% \times 100 + 36\% \times 4\% \times 100 = 3.44$

  $CPI_{stalls} = 2 + 3.44 = 5.44$

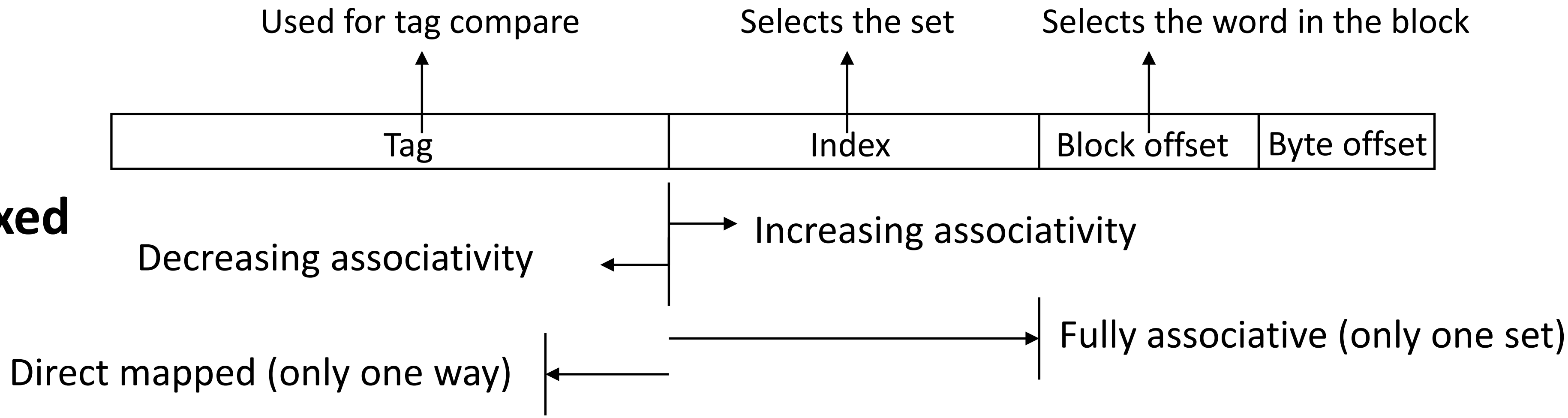
- **What if the CPI_{ideal} is reduced to 1? Or the processor clock rate is doubled (doubling the miss penalty)?**

Cache Types

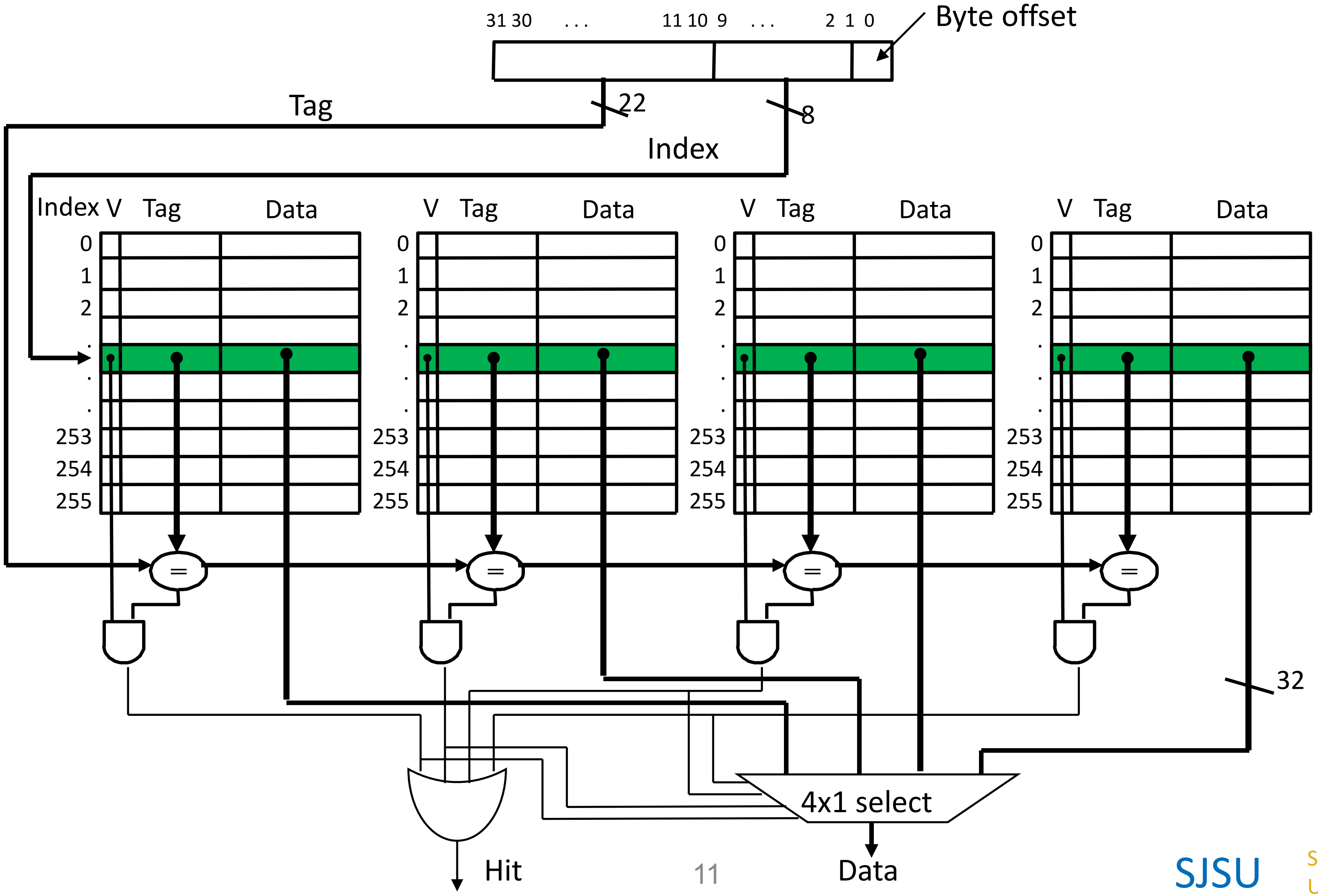
- **N-way Set-Associative:** Number of ways > 1 & Number of sets > 1
  - Slightly complex searching mechanism
- **Direct Mapped:** Number of ways = 1
  - Fast indexing mechanism
- **Fully-Associative:** Number of sets = 1
  - Extensive hardware resources required to search

	Way 0	**Way 1**	…
Set 0	block 0	block 2	
Set 1	block 1	block 3	
⋮			

Assuming fixed sized cache:



Four-Way Set Associative Cache



Cache Miss Classification: The 3 C’s

- **Compulsory (cold) Misses**
  - On the 1st reference to a block
  - Related to # blocks accessed by a code, not related to the configuration of a cache

- **Capacity Misses**
  - The program’s working set size exceeds the cache capacity

- **Conflict Misses**
  - Multiple memory blocks map to the same set in set-associative caches

Cache Policies: Cases - Revisit

- **Allocation policy: do we allocate a block in cache for the missed data?**

What else must be considered?

- **Read policies:**
  - Read Hit: this is what we want. Only one data read from the cache.
  - Read Miss: needs to fetch from lower level, but just write to the register once after that
    - read-allocate (with replacement policy) vs. no-read-allocate (i.e., cache bypassing)

      \+ write policy of evicted data
- **Write policies (only for the data cache): consistency & performance tradeoffs**
  - Write Hit: behavior and number of writes depends on write policy
    - Write-through vs. write-back vs. write-evict
  - Write Miss: needs to first read from lower level, then apply write policies
    - Write-allocate (with replacement policy): Write-through vs. Write-back

      \+ write policy of evicted data
    - No-write-allocate (bypassing): Write-evict

Cache Miss Behavior Analysis

- **Read miss:**
  - **+Write-through**: evict victim block + fetch block from lower level
  - **+Write-back**: evict victim block + write back evicted block if dirty + fetch block from lower level
  - **+No-write**: find victim block + fetch block from lower level
- **Write miss:**
  - Write-allocate:
    - **+Write-through**: evict victim block + fetch block from lower level + store word to block + <u>store word to lower level</u>
    - **+Write-back**: evict victim block + <u>write back evicted block if dirty</u> + fetch block from lower level + store word to block
  - No-write-allocate: **+Write-evict**: store word to lower level directly

Reduce Miss Rate (1): Code Optimization

- **Misses occur if sequentially accessed array elements come from different cache blocks**

- **Code optimizations → No hardware change**
  - Rely on programmers or compilers

- **Examples:**
  - Loop interchange: In nested loops, outer loop becomes inner loop and vice versa
  - Loop blocking: partition large array into smaller blocks, thus fitting the accessed array elements into cache size

Reduce Miss Rate (2): Reduce the 3 C's

- **Increase Cache Size**
  - Reduce miss for: capacity miss, conflict miss
  - But has many limitations

- **Increase Associativity**
  - Reduce miss for: conflict miss
  - But may increase access latency

- **Increase Cache Block Size**
  - Reduce miss for: compulsory
  - But may increase miss penalty (more data will be evicted and fetched)
  - Very large blocks could increase miss rate

Reduce Miss Rate (4): Multi-level Caches

- Having a unified L2 cache (i.e., it holds both instructions and data) and in some cases even a unified L3 cache

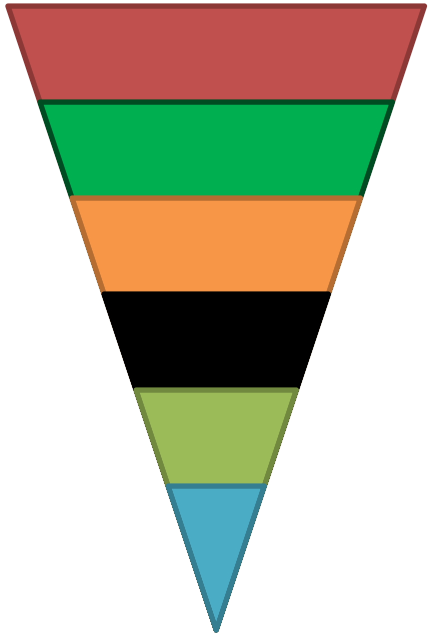
- L1 cache should focus on minimizing hit time in support of a shorter clock cycle

- Secondary cache(s) should focus on reducing miss rate to reduce the penalty of long main memory access times

DRAM Subsystem Organization

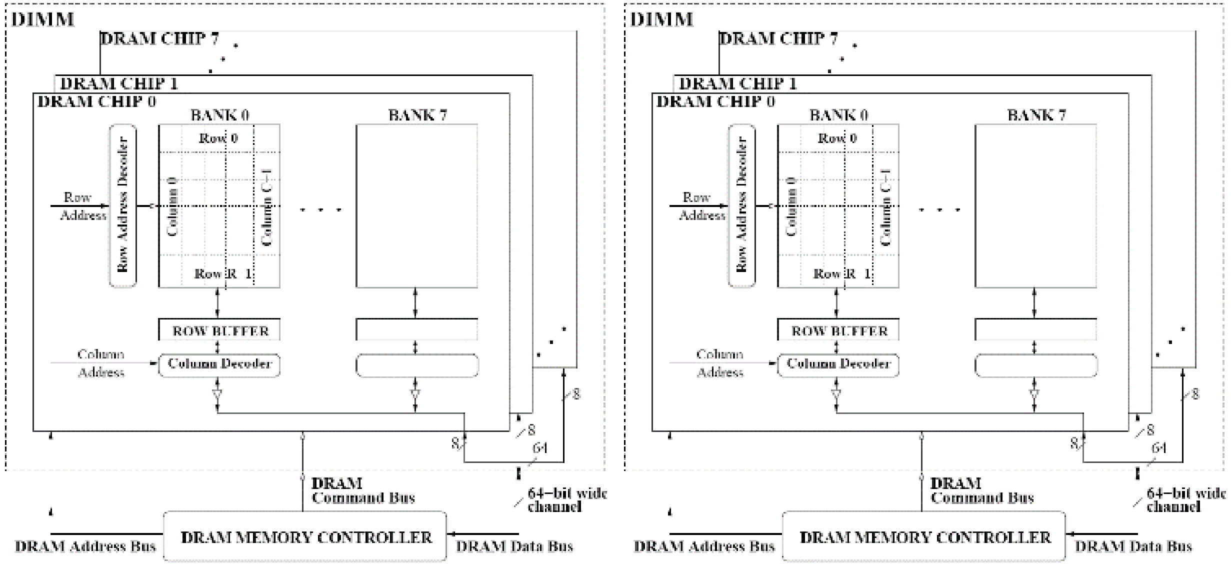
Dram: Dynamic RAM

DRAM Organization:



- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column

- **Channel: Independent memory subsystem**
  - E.g., 2 independent Channels:



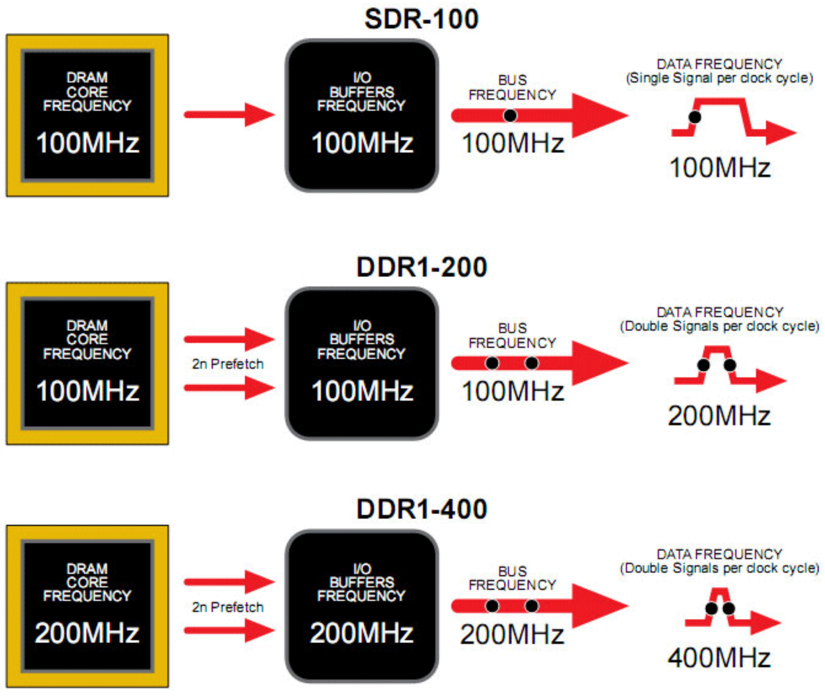
DRAM Design 2: Synchronous DRAMs

- **Like page mode DRAMs, synchronous DRAMs (SDRAMs) can transfer a burst of data from a series of sequential addresses in the same row**

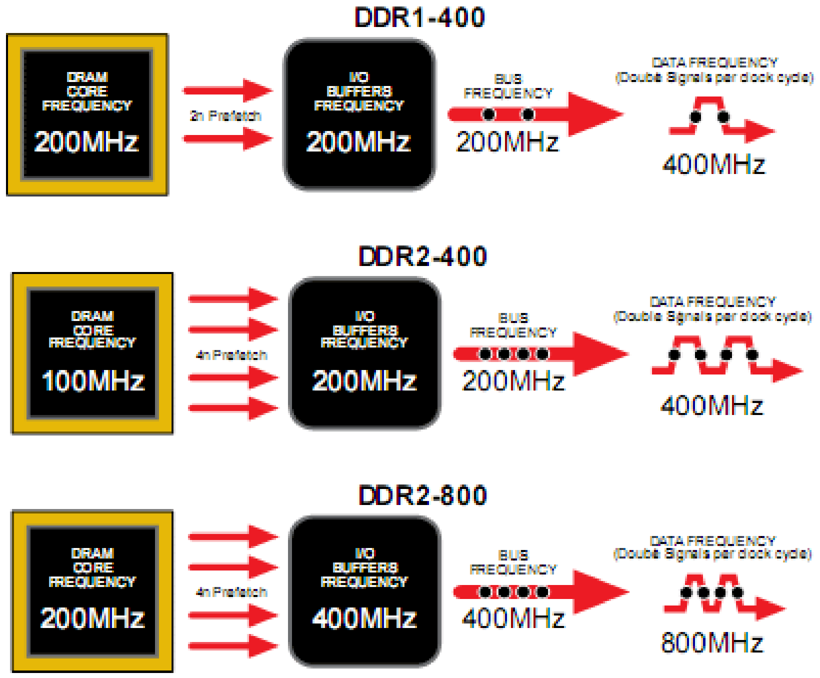
- **For words in the same burst, don't have to provide the complete (row and column) addresses**
  - The entire row is loaded into a row buffer (SRAM).
  - Specify the starting (row+column) address and the burst length (burst must be in the same row).
  - Data words in the burst are then accessed from that SRAM under control of a clock signal.

DDR (Double Data Rate) SDRAMs

- **DDR1 VS DDR1+:**



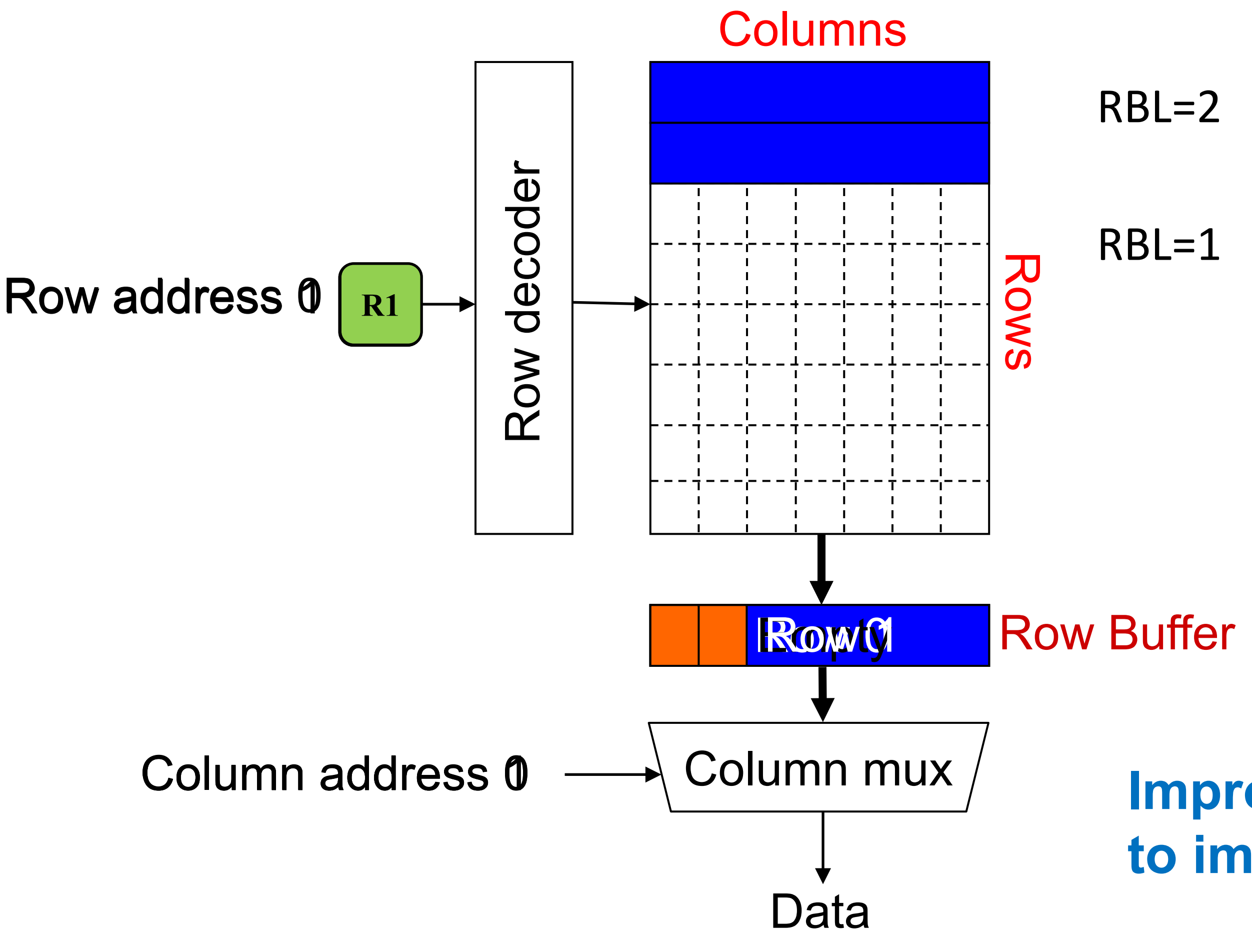
▲ Simplified Comparison between SDR-100, DDR1-200 and DDR1-400
Illustration: Ryan J. Leng



▲ Simplified Comparison between DDR1-400, DDR2-400 and DDR2-800
Illustration: Ryan J. Leng

- **DDR2 vs. QDR**

Row Operations & Row Buffer Locality



	Access Address:	Row Operation:
RBL=2	(Row 0, Column 0)	Activation
	(Row 0, Column 1)	No operation
RBL=1	(Row 1, Column 0)	Restore, Precharge, Activation

CONFLICT !

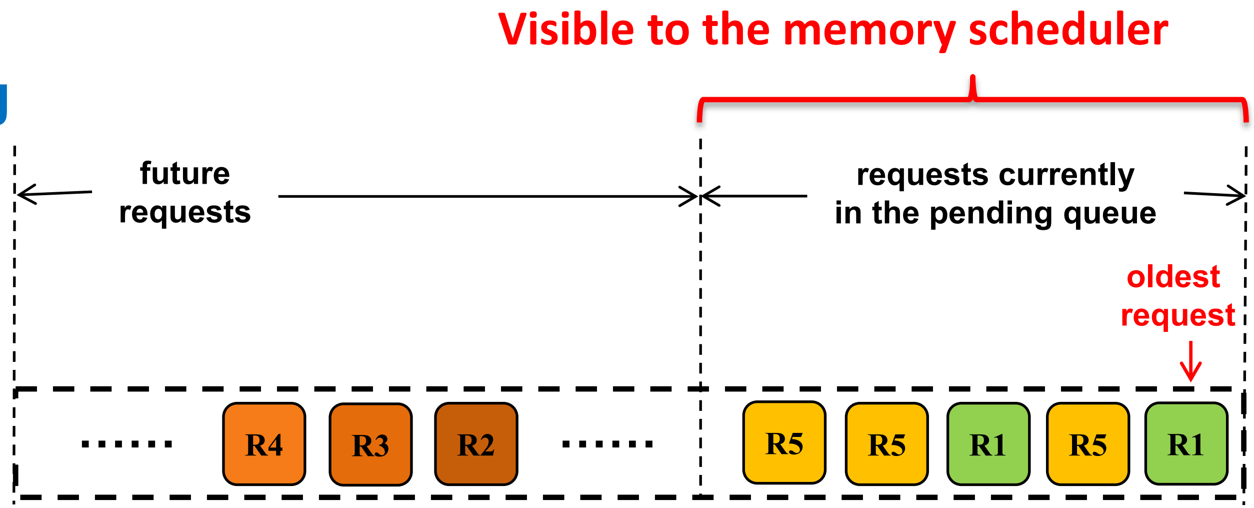
Improving Row Buffer Locality (RBL) is the key to improve DRAM efficiency

Credit: Onur Mutlu.

RBL & Memory Scheduling Schemes

In-order scheduling (FIFO):

Request Stream →

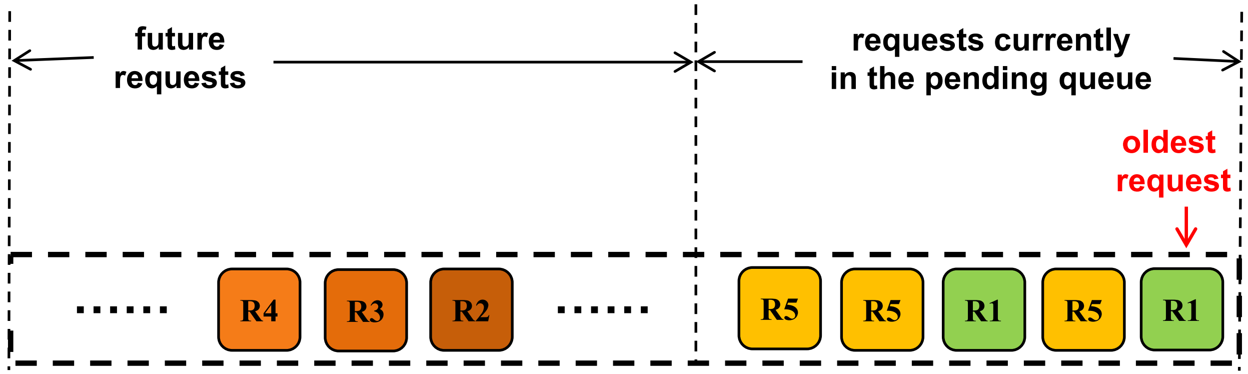


Activation Counter:

R1: Activation = 1
R5: Activation = 2
R1: Activation = 3
R5: Activation = 4 } Same activation
R5: Activation = 4
Avg RBL = 5 / 4 = 1.25

Out-of-order scheduling (FR-FCFS):

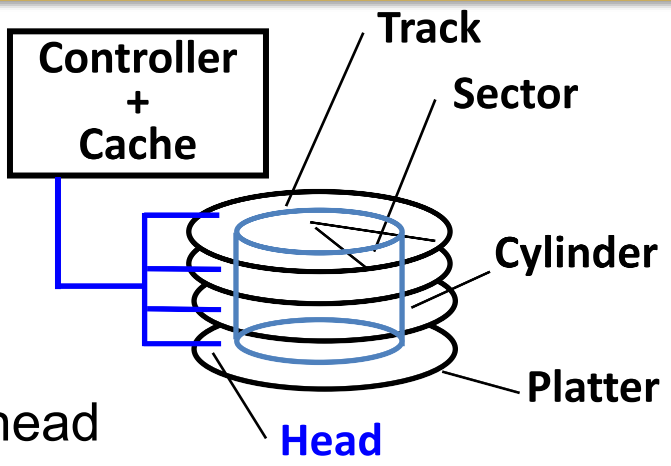
Request Stream →



Activation Counter:

R1: Activation = 1 } Same activation
R1: Activation = 1
R5: Activation = 2 } Same activation
R5: Activation = 2
R5: Activation = 2
Avg RBL = 5 / 2 = 2.5

Magnetic Disk Characteristic



1. Seek time: position the head over the proper track
   - 3 to 12/15 ms on average
   - Due to locality of disk references, the actual average seek time may be only 25% to 33% of the advertised number

2. Rotational latency: wait for the desired sector to rotate under the head
   - ½ of 1/RPM converted to ms: 0.5R/5400RPM = 5.6ms to 0.5R/15000RPM = 2.0ms

3. Transfer time: transfer a block of bits (one or more sectors) under the head to the disk controller's cache (70 to 125 MB/s are typical disk transfer rates)
   - the disk controller's "cache" takes advantage of spatial locality in disk accesses
   - cache transfer rates are much faster (e.g., 375 MB/s)

4. Controller time: the overhead the disk controller imposes in performing a disk I/O access (typically < 0.2 ms)

Typical Disk Access Time

The average time to read or write a 512B sector for a disk rotating at 15,000 RPM with average seek time of 4 ms, a 100MB/sec transfer rate, and a 0.2 ms controller overhead

Avg disk read/write

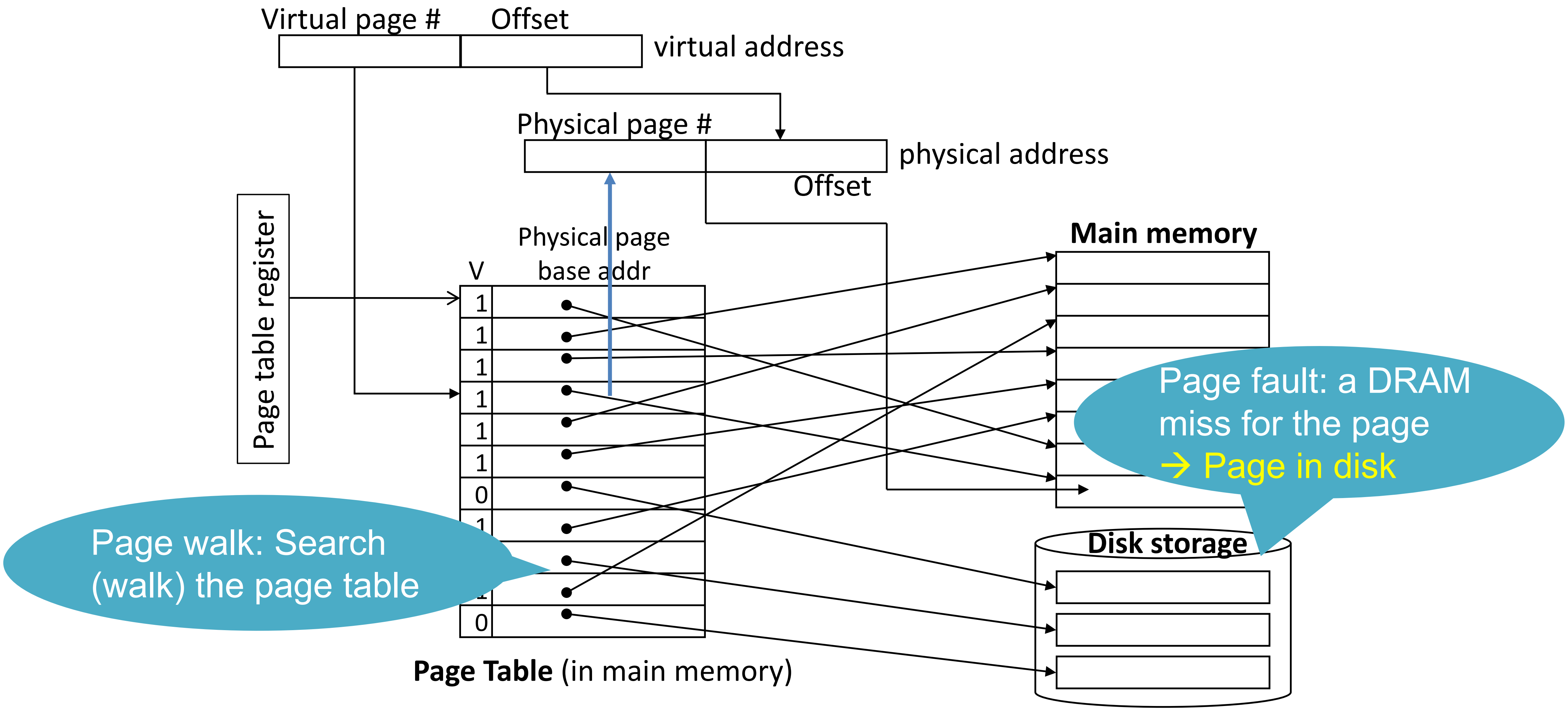
= 4.0 ms + 0.5/(15,000RPM/(60sec/min)) + 0.5KB/(100MB/sec) + 0.2 ms

= 4.0 + 2.0 + 0.005 + 0.2 = 6.2 ms

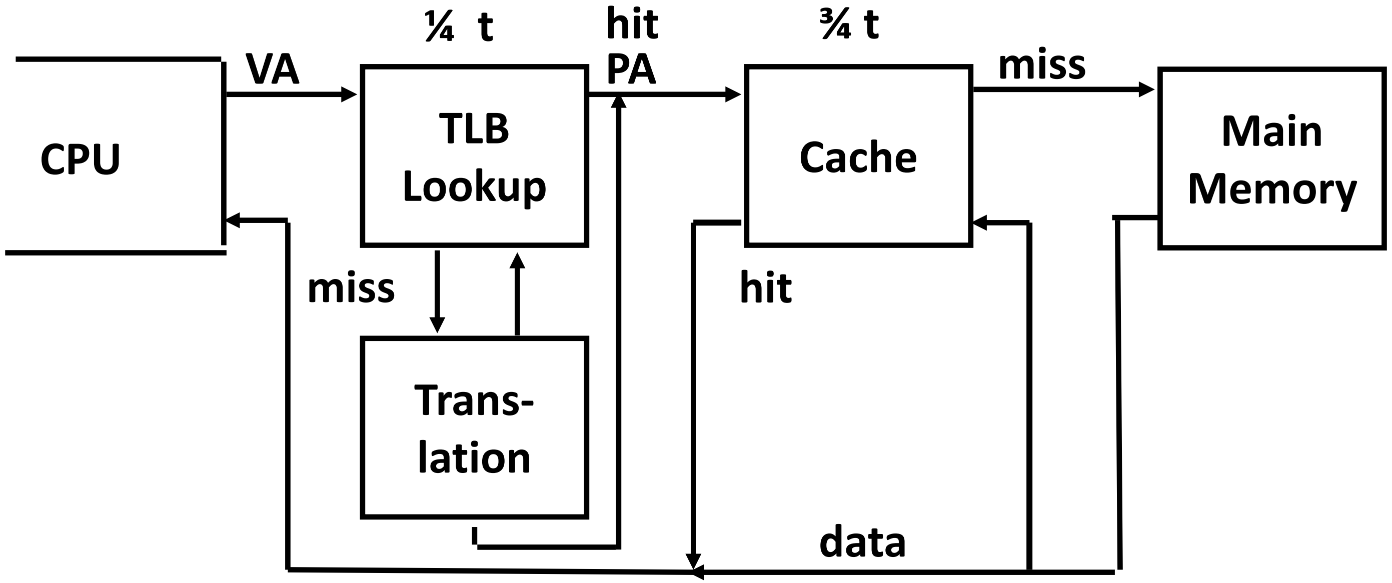
If the measured average seek time is 25% of the advertised average seek time, then

Avg disk read/write = 1.0 + 2.0 + 0.005 + 0.2 = 3.2 ms

Address Translation Mechanisms



A TLB in the Memory Hierarchy



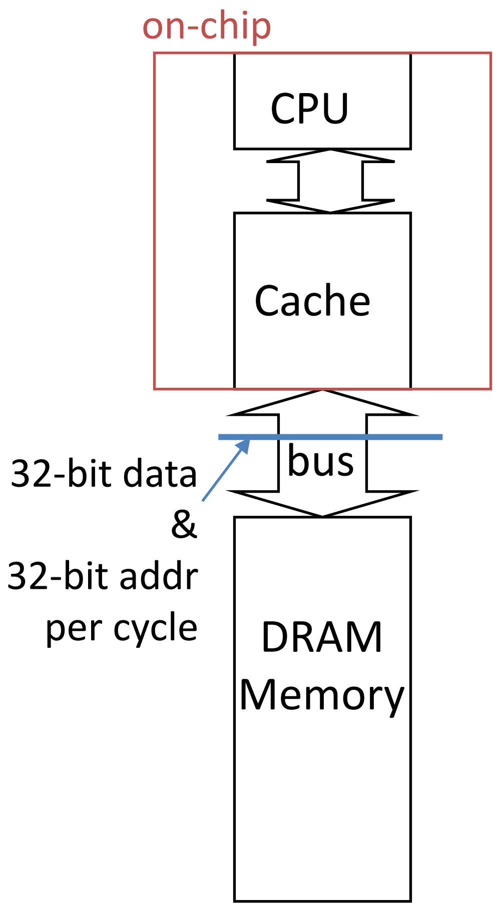
- **A TLB miss – is it a page fault or merely a TLB miss?**
  - If the page is loaded into main memory, then the TLB miss can be handled by loading the translation information from the page table into the TLB (10's of cycles)
  - If the page is not in main memory, then it's a true page fault (1,000,000's)

- **TLB misses are much more frequent than true page faults**

TLB Event Combinations

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – this is what we want!
Hit	Hit	Miss	Yes – although the page table is not checked after the TLB hits
Miss	Hit	Hit	Yes – TLB missed, but PA is in page table and data is in cache
Miss	Hit	Miss	Yes – TLB missed, but PA is in page table, data not in cache
Miss	Miss	Miss	Yes – page fault
Hit	Miss	Miss/ Hit	No – TLB translation is not possible if the page is not present in main memory
Miss	Miss	Hit	No – data is not allowed in the cache if the page is not in memory

Memory Systems that Support Caches



One word wide organization
(one word wide bus & memory)

The off-chip interconnect and memory architecture can affect overall system performance in dramatic ways

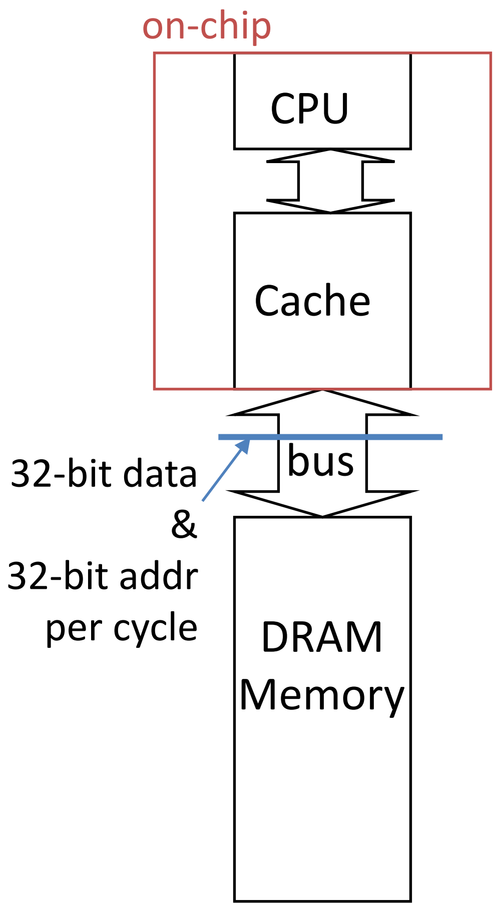
Assume:

- 1 memory bus clock cycle to send the address
- 15 memory bus clock cycles to get the 1st word in the block from DRAM (row cycle time), 5 memory bus clock cycles for 2nd, 3rd, 4th words (column access time)
- 1 memory bus clock cycle to return a word of data

Memory-Bus to Cache bandwidth

- number of bytes accessed from memory and transferred to cache/CPU per memory bus clock cycle

One Word Wide Bus, One Word Blocks



One word wide organization
(one word wide bus & memory)

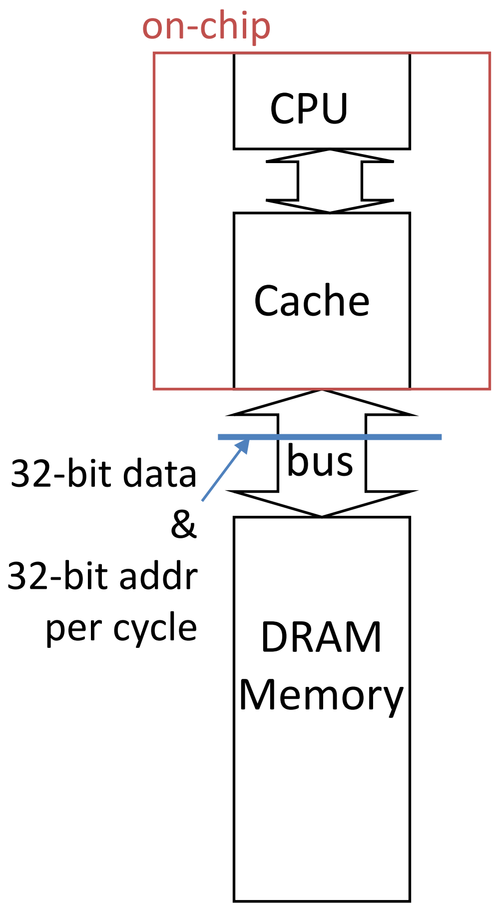
- **If the block size is one word, then for a cache miss, the pipeline will have to stall for:**

  1 memory bus clock cycle to send address
  15 memory bus clock cycles to read DRAM
  1 memory bus clock cycle to return data
  17 total clock cycles miss penalty

- **Number of bytes transferred per clock cycle (bandwidth) for a single miss is**

  4 / 17 = 0.235 bytes per memory bus clock cycle

One Word Wide Bus, Four Word Blocks



One word wide organization
(one word wide bus & memory)

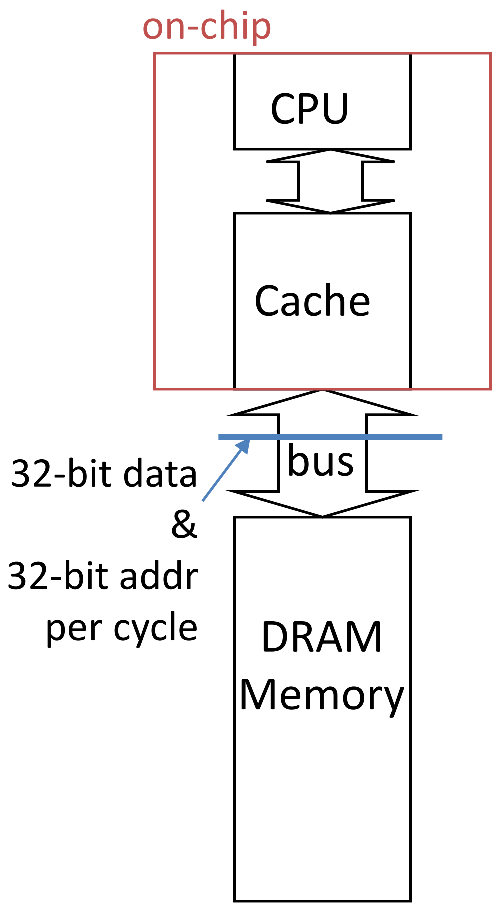
- **What if the block size is four words and each word is in a different DRAM row?**

1	cycle to send 1st address
4 x 15 = 60	cycles to read DRAM
1	cycles to return last data word
62	total clock cycles miss penalty

- **Number of bytes transferred per clock cycle (bandwidth) for a single miss is**

  (4 x 4) / 62 = 0.258 bytes per memory bus clock cycle

One Word Wide Bus, Four Word Blocks



One word wide organization
(one word wide bus & memory)

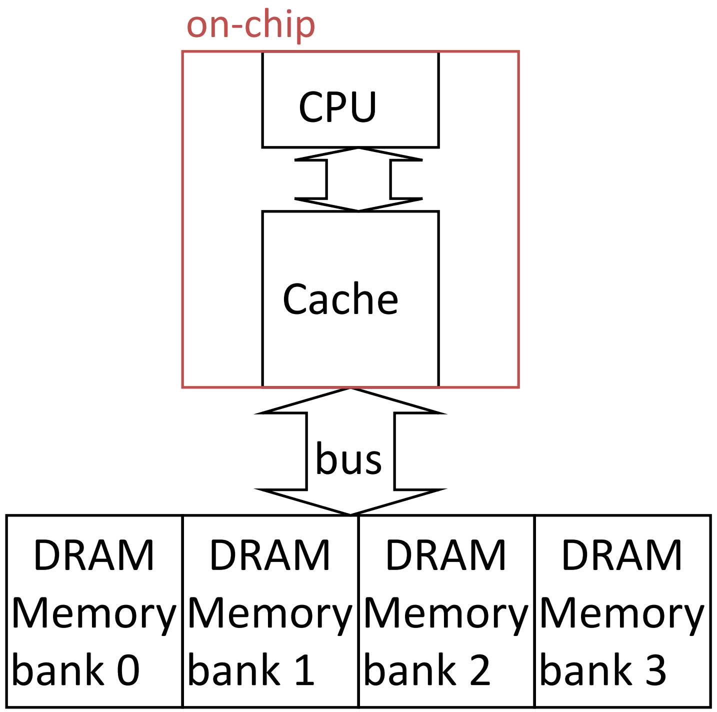
- **What if the block size is four words and all words are in the same DRAM row?**

1	cycle to send 1st address
15 + 3 x 5 = 30	cycles to read DRAM
1	cycles to return last data word
32	total clock cycles miss penalty

- **Number of bytes transferred per clock cycle (bandwidth) for a single miss is**

(4 x 4) / 32 = 0.5 bytes per memory bus clock cycle

Interleaved Memory, One Word Wide Bus



- **For a block size of four words**

1	cycle to send 1st address
15	cycles to read DRAM banks
4 x 1 = 4	cycles to return last data word
20	total clock cycles miss penalty

- **Number of bytes transferred per clock cycle (bandwidth) for a single miss is**

$(4 \times 4) / 20 = 0.8$ bytes per memory bus clock cycle

What about interleaving channels?

Can We Do Better?

- **Latency vs. throughput**

- **Levels of Parallelism:**
  - Instruction-level Parallelism (ILP)
    - Executing independent instructions (in one thread) in parallel
  - Data-level Parallelism (DLP)
    - Executing the same instruction on different data subsets
  - Thread-level Parallelism (or Task-level Parallelism, TLP)
    - Executing independent computing tasks in parallel (on same or different data)

Dynamic Pipeline Scheduling

- **Challenges**
  - All data hazards (memory and registers) must be resolved by hardware

- **Data Dependencies**
  - RAW (Read After Write) : True dependency
    - Dynamic pipeline preserves only this dependency
  - WAR (Write After Read) & WAW (Write After Write) : False dependencies
    - Dynamic pipeline removes false dependencies by using register renaming

Dependencies Among Instructions

Mapping table status

Logical	initial	lw	addu	sub	slti
l1	t0	P1	P1	P1	P4
l2	t1	t1	P2	P2	P2
l3	t2	t2	t2	t2	t2
l4	t3	t3	t3	t3	t3
l5	s2	s2	s2	P3	P3
l6	s4	s4	s4	s4	s4

Free list status

initial	lw	addu	sub	slti
P1	P2	P3	P4	P5
P2	P3	P4	P5	…
P3	P4	P5	…	
P4	P5	…		
P5	…			

```
lw     $t0, 20($s2)
addu   $t1, $t0, $t2
sub    $s2, $s4, $t3
slti   $t0, $s2, 20
```

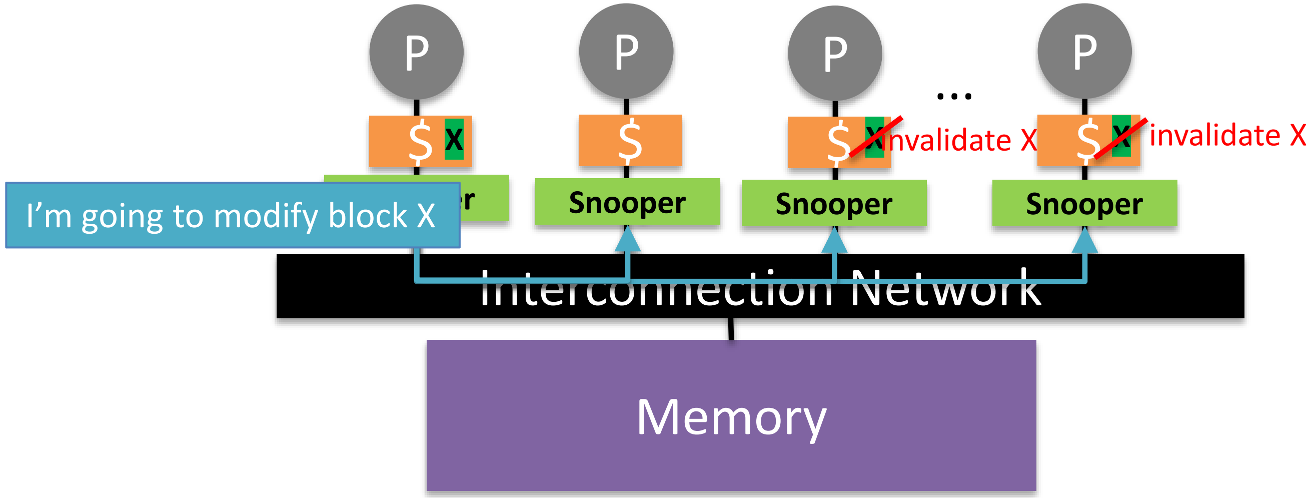
```
lw     P1, 20($s2)
addu   P2, P1, $t2
sub    P3, $s4, $t3
slti   P4, P3, 20
```

How to Share Data?

- **Shared Memory multi-Processor (SMP)**
  - Single address space shared by all cores
  - Cores coordinate/communicate through shared variables (via loads and stores)
    - Need synchronization primitives (e.g., locks)
  - Two styles
    - Uniform memory access (UMA): easier for programming
    - Non-uniform memory access (NUMA): more scalable & lower latency to local memory
- **Message Passing multi-Processors (MPP)**
  - Each core has its own private address space
    - Cores share data by *explicitly* sending and receiving information (message passing)
  - Coordination is built into MP primitives (message send & message receive)

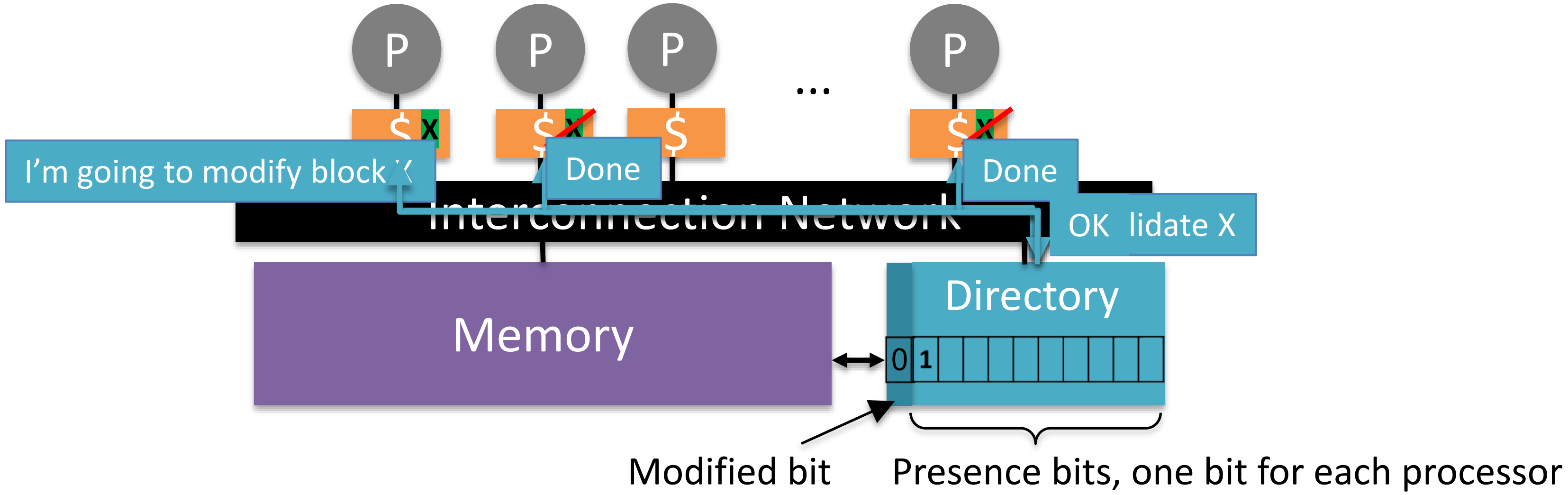
Cache Coherence Methods

- **Snooping-based protocol**
  - Each processor's cache controller constantly snoops on the bus
  - Processors observe other processors' actions
    - E.g., processor A makes "read-exclusive" request for A on bus, processor B sees this and invalidates its own copy of A



Cache Coherence Methods

- **Directory-based protocol**
  - Directory tracks ownership (sharer set) for each block (who has what)
    - A *modified* bit and multiple *presence* bits per cache block
  - Directory coordinates invalidation appropriately
    - E.g., processor A asks directory for “read-exclusive” copy, directory asks processor B to invalidate the requested copy, waits for ACK from processor B, then responds to processor A



Load Balancing

- **Load balancing is another important factor**
  - E.g., a single core with twice the load of the others cuts the speedup almost in half

- **S = serial part, W = parallel part, N cores**
  - Time on 1 processor is S + W

- **Suppose the workload is equally distributed and there is no extra overhead to accumulate partial results**
  - Time on N processors is S + W/N

- **Suppose the workload is almost equally distributed (and still no overhead)**
  - One core does 2(W/N), one does nothing, most do W/N
  - Time is S + max(2(W/N), 0, W/N) = S + 2(W/N) = S + W/(N/2)

Multi-core CPUs, Multiprocessors, and GPUs

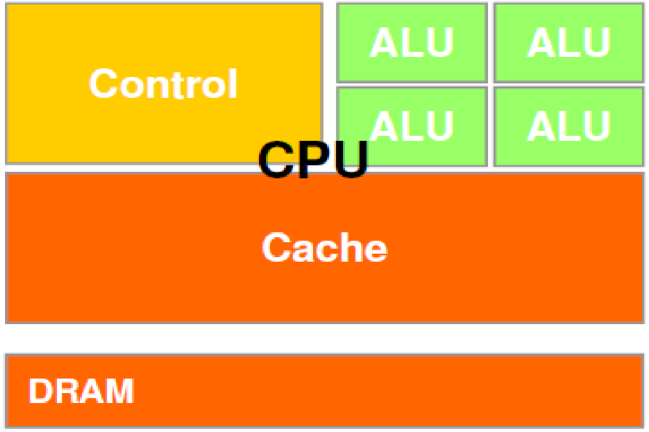
- **Multi-core CPUs, Multiprocessors**
  - Individual cores are fine-tuned for high ILP
  - Good for Task-level parallelism (TLP)
  - However, it is hard to employ hundreds of cores in one system
    - Cache coherence, control, power, cost …

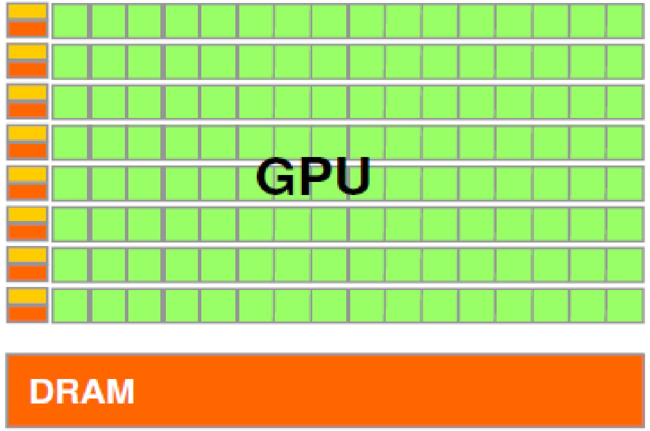
- **For specific tasks, Graphics Processing Unit (GPU) can be an alternative solution**
  - For tasks with high Data-Level parallelism
  - E.g., particle simulation, image/video processing, games, machine learning …
  - The usage of GPU is common now

- **Instruction & Data level parallelism combinations**
  - SISD: Classical Von Neumann machine
  - MISD: NA
  - SIMD: GPU
  - MIMD: multi-core & multiprocessor

Why is GPU so Efficient?

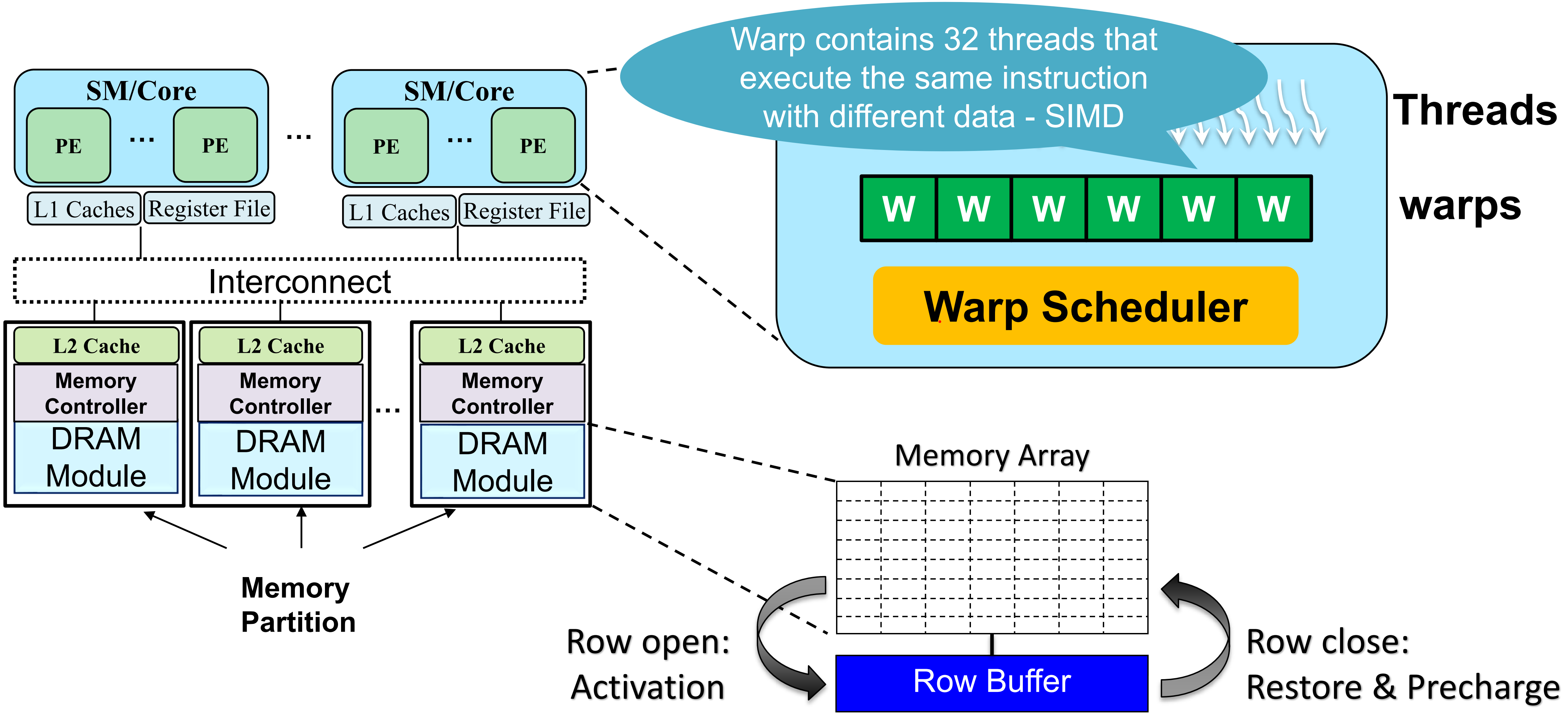
- **GPU is specialized for compute-intensive, highly data parallel computation (owing to its graphics rendering origin)**
  - More transistors can be devoted to data processing rather than caching and control
  - At peak performance GPU uses order of magnitude less energy per operation than CPU
  - Working with suitable applications: high arithmetic intensity (the ratio between arithmetic operations and memory operations), high DLP, not too sensitive to latency





Credit: Dan Negrut, ME964 UW-Madison

Classical GPGPU Architecture (Nvidia)

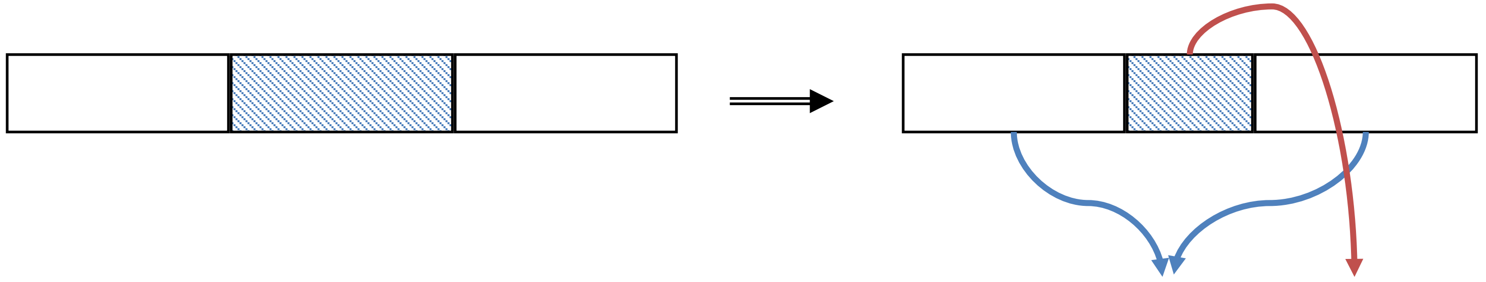


Multicore Performance: Amdahl's Law

- **Speedup due to enhancement E:**

$$\text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}$$

- **Suppose that enhancement E accelerates a fraction F (F <1) of the task by a factor S (S>1) and the remainder of the task is unaffected:**



$$\text{ExTime w/ E} = \text{ExTime w/o E} \times ((1\text{-}F) + F/S)$$

$$\text{Speedup w/ E} = 1 / ((1\text{-}F) + F/S)$$

Example 1: Amdahl's Law

Speedup w/ E = 1 / ((1-F) + F/S)

- **Consider an enhancement that runs 20 times faster but is only usable 25% of the time**

  Speedup w/ E = 1/(.75 + .25/20) = 1.31

- **What if its usable only 15% of the time?**

  Speedup w/ E = 1/(.85 + .15/20) = 1.17

- **Amdahl's Law tells us that to achieve linear speedup with 100 cores, none of the original computation can be scalar!**

- **To get a speedup of 90 from 100 cores, the percentage of the original program that could be scalar would have to be 0.1% or less**

  Speedup w/ E = 1/(.001 + .999/100) = 90.99

Example 2: Amdahl's Law

Speedup w/ E = 1 / ((1-F) + F/S)

- **Consider summing 10 scalar variables and two 10 by 10 matrices on 10 cores**

  Speedup w/ E = 1/(.091 + .909/10) = 1/0.1819 = 5.5

- **What if there are 100 cores?**

  Speedup w/ E = 1/(.091 + .909/100) = 1/0.10009 = 10.0

- **What if the matrices are 100 by 100 (or 10,010 adds in total) on 10 cores?**

  Speedup w/ E = 1/(.001 + .999/10) = 1/0.1009 = 9.9

- **What if there are 100 cores?**

  Speedup w/ E = 1/(.001 + .999/100) = 1/0.01099 = 91

Other Important Points

- **Review the first half with midterm review slides and exam questions!**
- **Processor review quiz**
- **Extra review session (online): 15:00 – 16:15 PM, Sunday, Dec. 4th**
  - Check announcements on Canvas
- **Extra Office hour (online): 15:00 – 16:15 PM, Wednesday, Dec. 7th**
  - Check announcements on Canvas
- **You do not have to know everything in the textbook. However, any content covered in the lecture could appear in the exam. The review slides do not cover everything.**
- **Do not leave it blank!**

Good Luck!

I will be available for questions via Email/Canvas/Slack/Zoom

Interested in GPUs?

- Spring 2023: CMPE214 GPU Architecture and Programming

Grader & Research opportunities

- Yes, you are readily prepared!
- Funded opportunity: RA position for outstanding students
- MS Thesis & MS projects

Course Evaluation: Please participate before Dec 7, 2022

- You should have received an E-mail, or you can find it on Canvas in the SOTE/SOLATE tab

- Feel free to send me feedback directly!

Thank you!

SAN JOSÉ STATE UNIVERSITY *powering* SILICON VALLEY