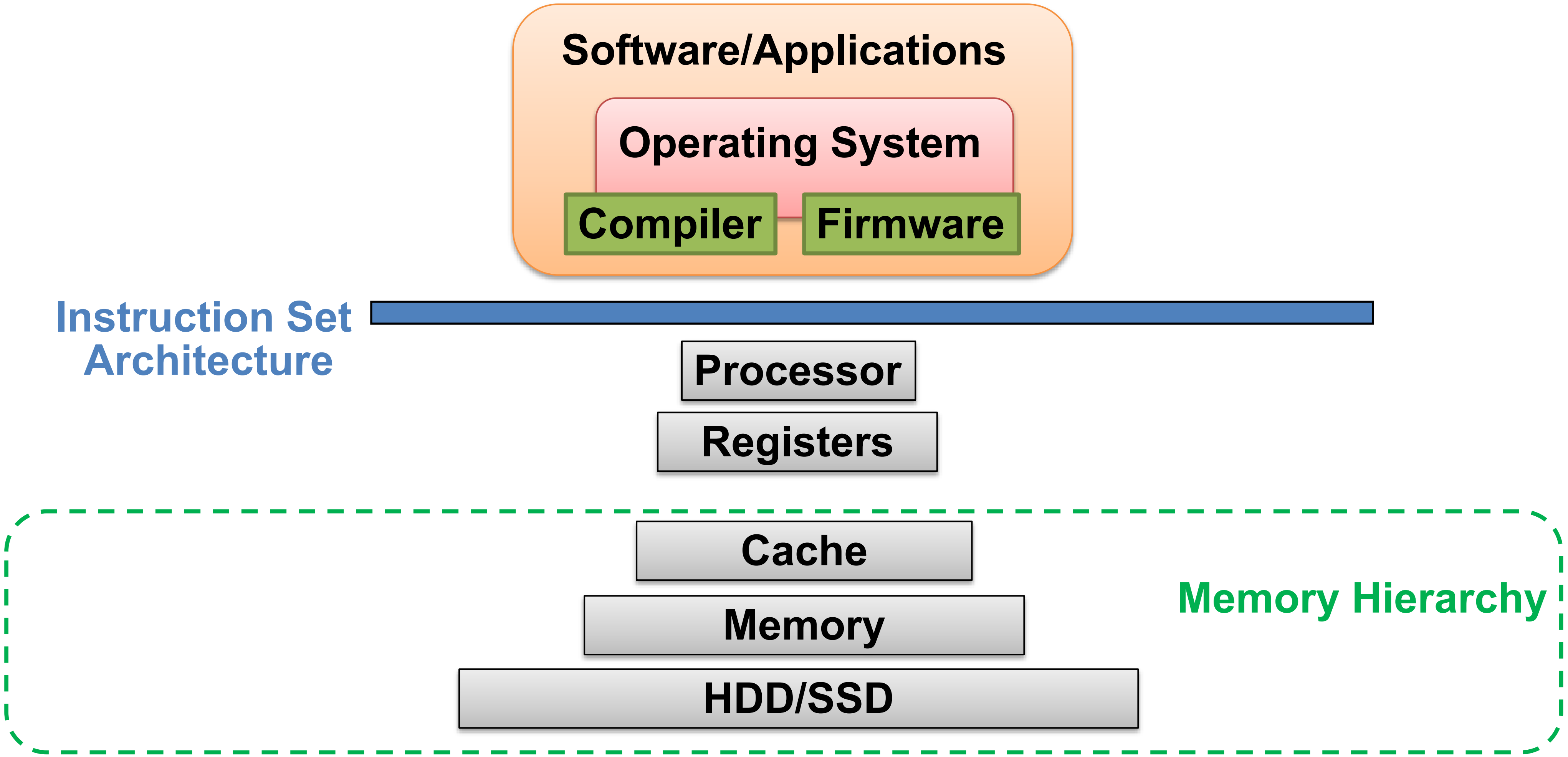
CMPE 200
Computer Architecture & Design

Lecture 4.
Memory Hierarchy

Haonan Wang

SJSU SAN JOSÉ STATE UNIVERSITY

Computer Architecture Overview



Memory Hierarchy

Example: a C program that reads two integer values from “file.txt” file and prints the sum of them.

```
#include <stdio.h>
#include <string.h>

int numbers[2];

void myfunction(void)
{
   FILE *fp;
   int size = 2;
   int sum = 0;

   /* Open file for reading */
   fp = fopen(“mynumbers.txt", “r");

   /* Read and display data */
   fread(numbers, sizeof(int), size, fp);
   fclose(fp);
   sum = numbers[0] + numbers[1];
   printf(“Sum = %d\n”, sum);
}

int main (void) {
   myfunction();
   return(0);
}
```

Processor (CPU)

Understands and executes each line of the code.

Uses fast on-chip memories

Memory (DRAM)

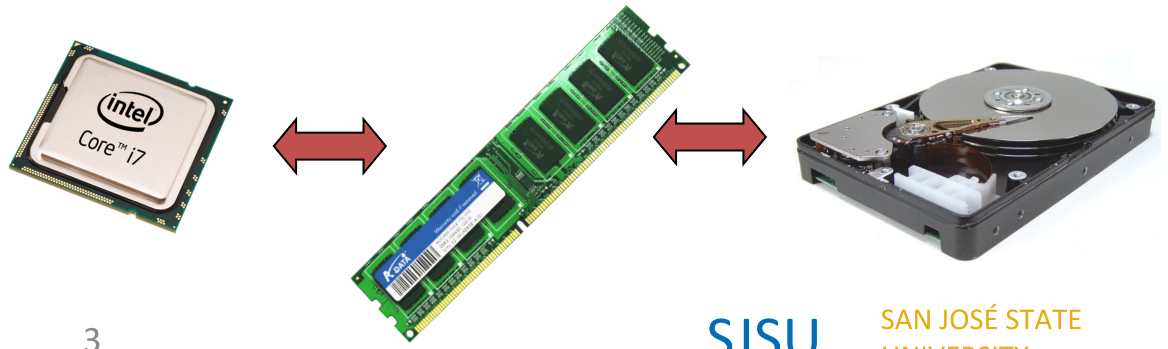
Provides operands to CPU

(*fp, size, sum, numbers[2])

Storage (HDD)

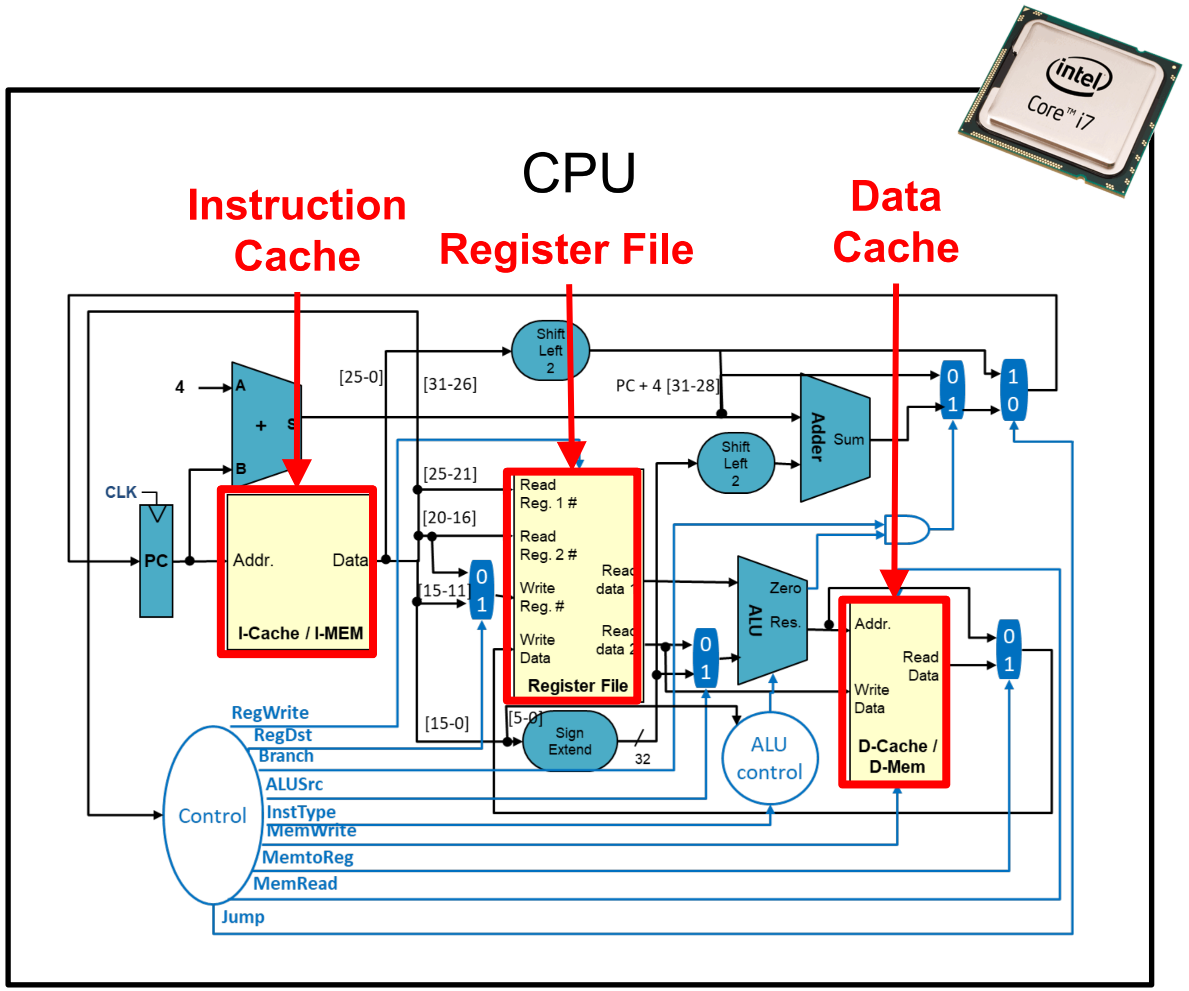
Provides file inputs and program code

(file.txt)

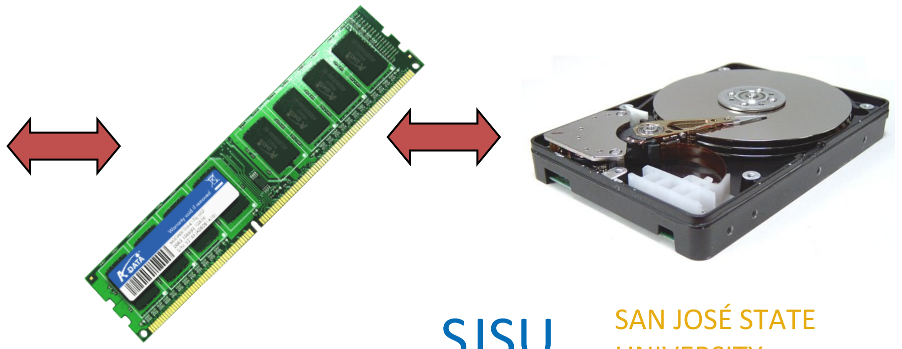


SJSU SAN JOSÉ STATE UNIVERSITY

Memory Hierarchy

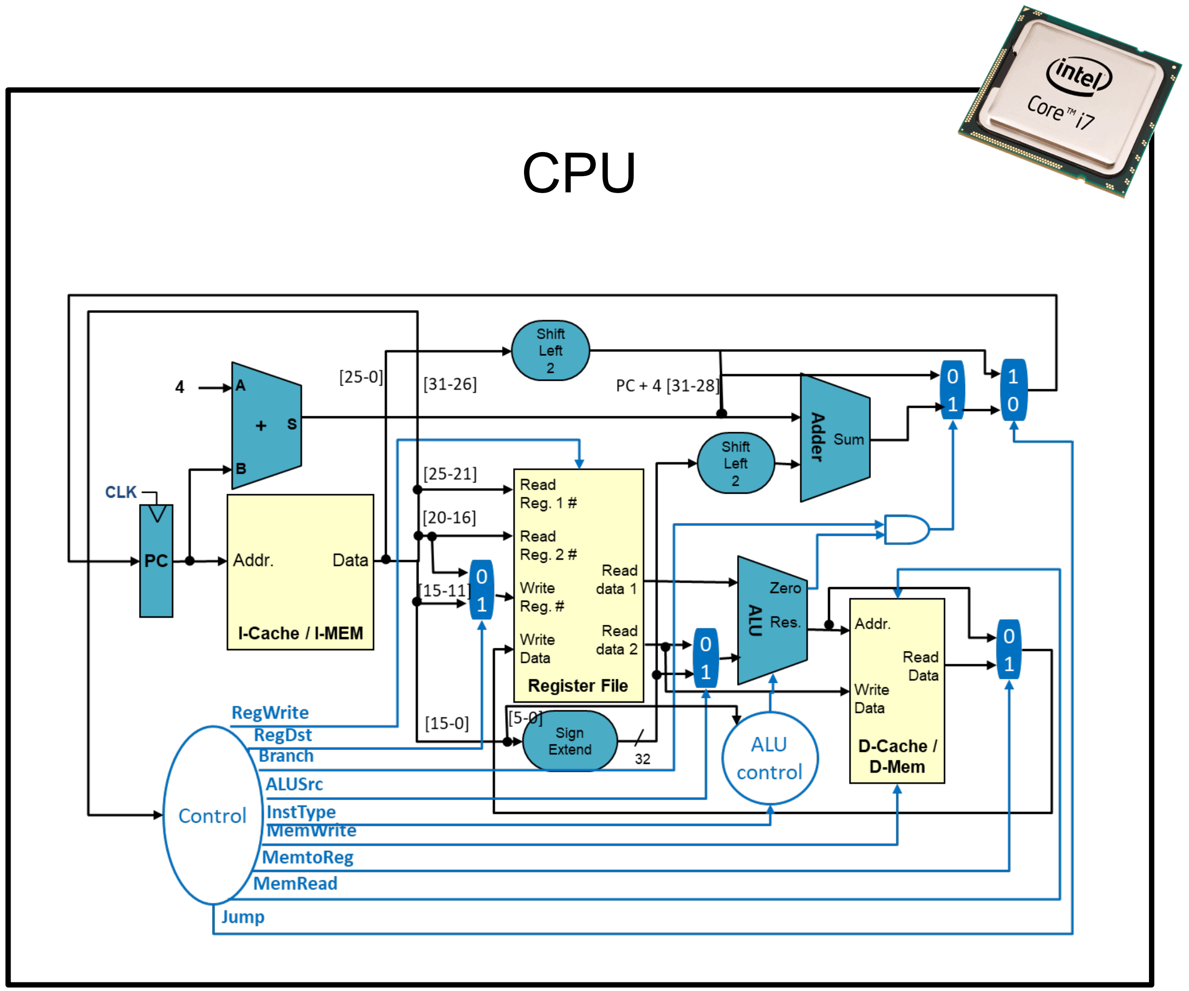


- **On-chip Memories (Memories inside of CPU)**
  - Register file, Caches
  - Small but Fast

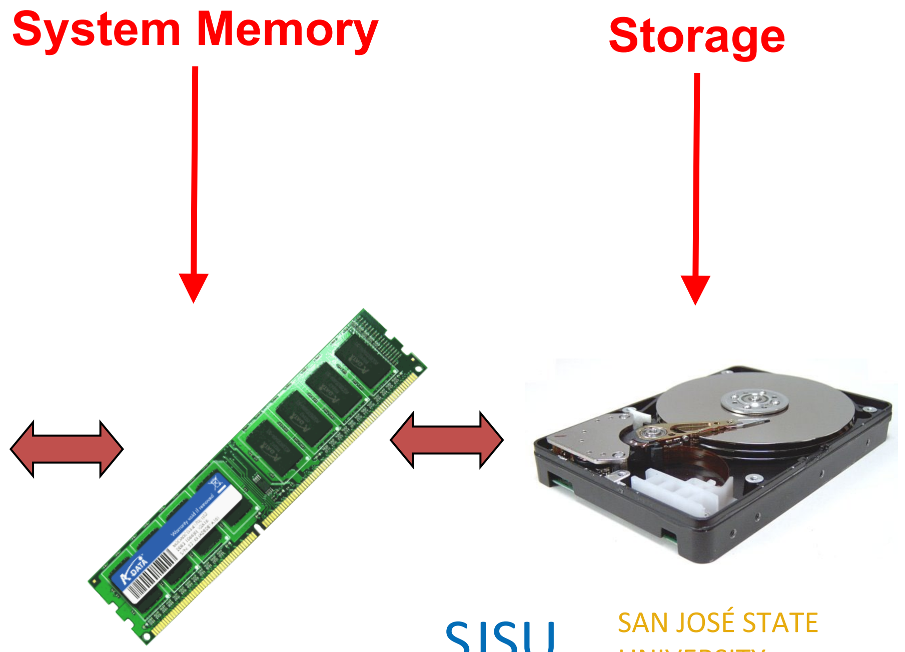


SJSU SAN JOSÉ STATE UNIVERSITY

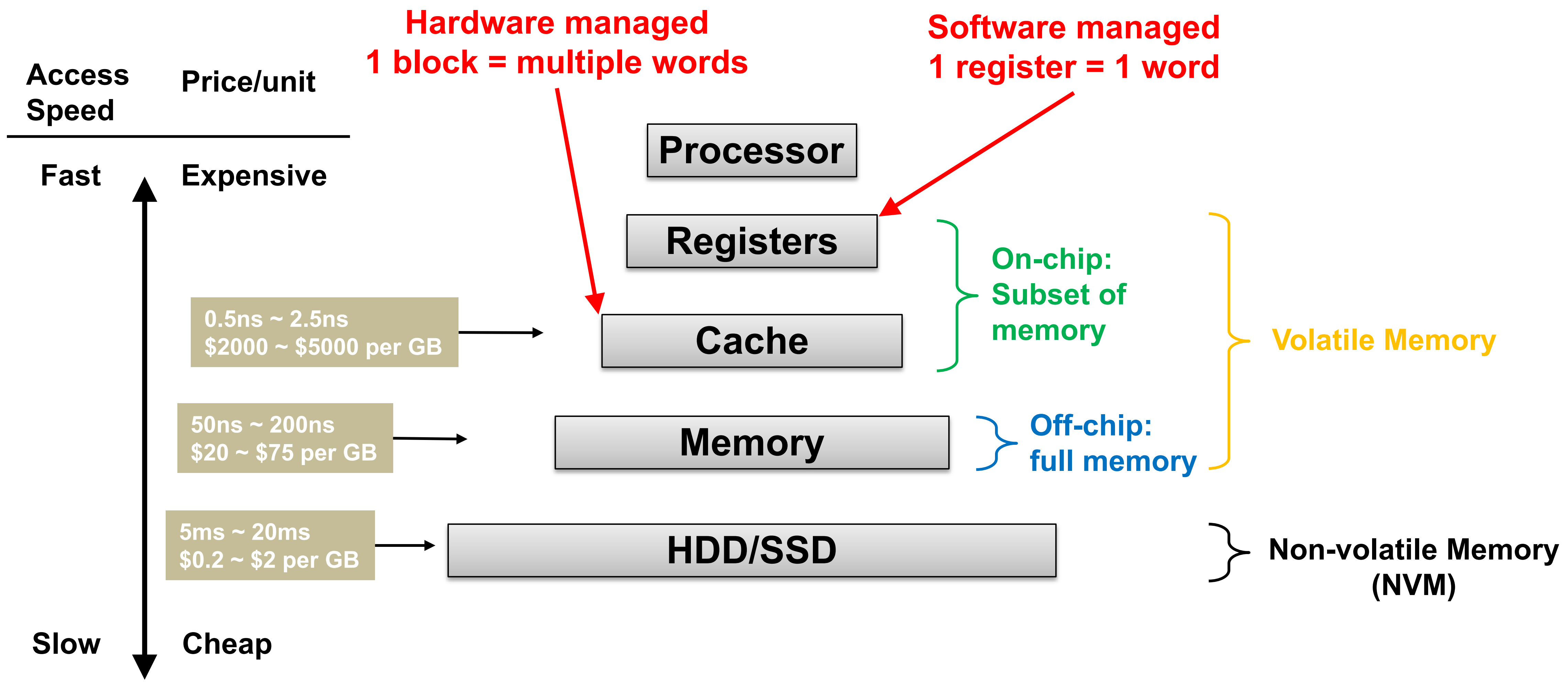
Memory Hierarchy



- **Off-chip Memories (Memories outside CPU)**
  - System Memory, Storage
  - Large but Slow



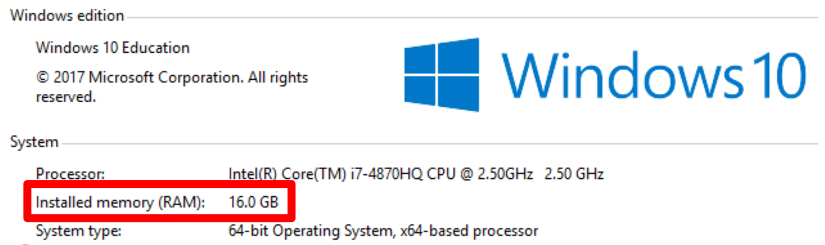
Memory Hierarchy



SJSU SAN JOSÉ STATE UNIVERSITY

Memories in Your PC

- **Windows**
  - This PC → Properties
  - cmd window → wmic
  - 3rd party tool like CPU-Z



Windows edition

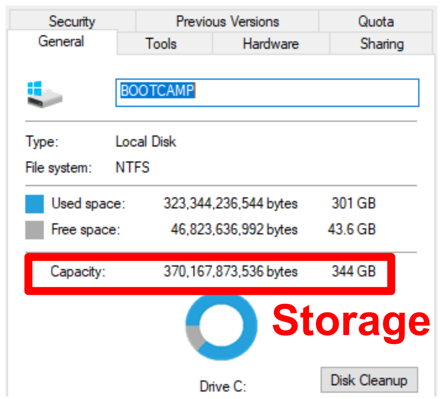
Windows 10 Education

© 2017 Microsoft Corporation. All rights reserved.

System

Processor:	Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz 2.50 GHz
Installed memory (RAM):	16.0 GB
System type:	64-bit Operating System, x64-based processor

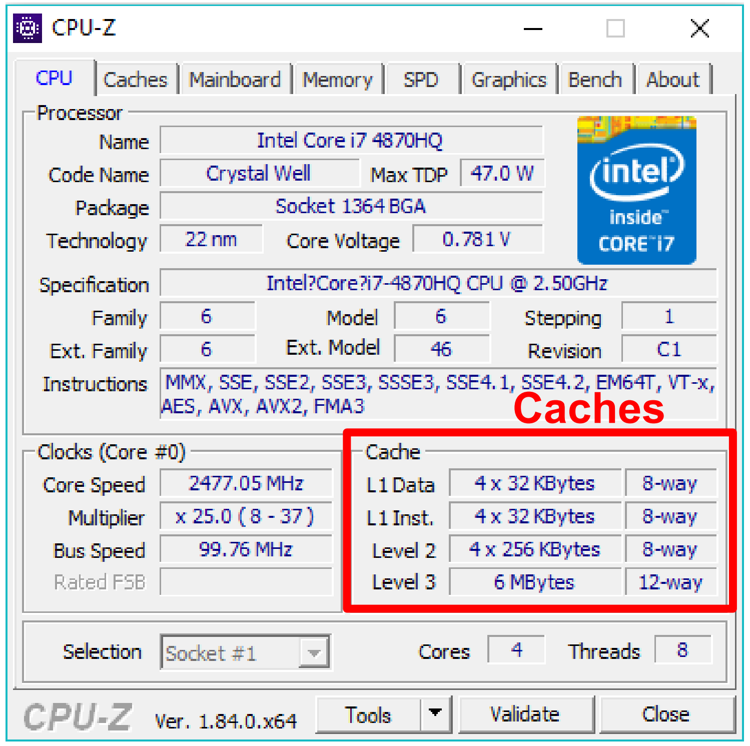
System Memory



BOOTCAMP

Type:	Local Disk	
File system:	NTFS	
Used space:	323,344,236,544 bytes	301 GB
Free space:	46,823,636,992 bytes	43.6 GB
Capacity:	370,167,873,536 bytes	344 GB

Storage



Processor

Name	Intel Core i7 4870HQ				
Code Name	Crystal Well	Max TDP	47.0 W		
Package	Socket 1364 BGA				
Technology	22 nm	Core Voltage	0.781 V		
Specification	Intel?Core?i7-4870HQ CPU @ 2.50GHz				
Family	6	Model	6	Stepping	1
Ext. Family	6	Ext. Model	46	Revision	C1
Instructions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3				

Caches

Clocks (Core #0)

Core Speed	2477.05 MHz
Multiplier	x 25.0 (8 - 37)
Bus Speed	99.76 MHz
Rated FSB	

Cache

L1 Data	4 x 32 KBytes	8-way
L1 Inst.	4 x 32 KBytes	8-way
Level 2	4 x 256 KBytes	8-way
Level 3	6 MBytes	12-way

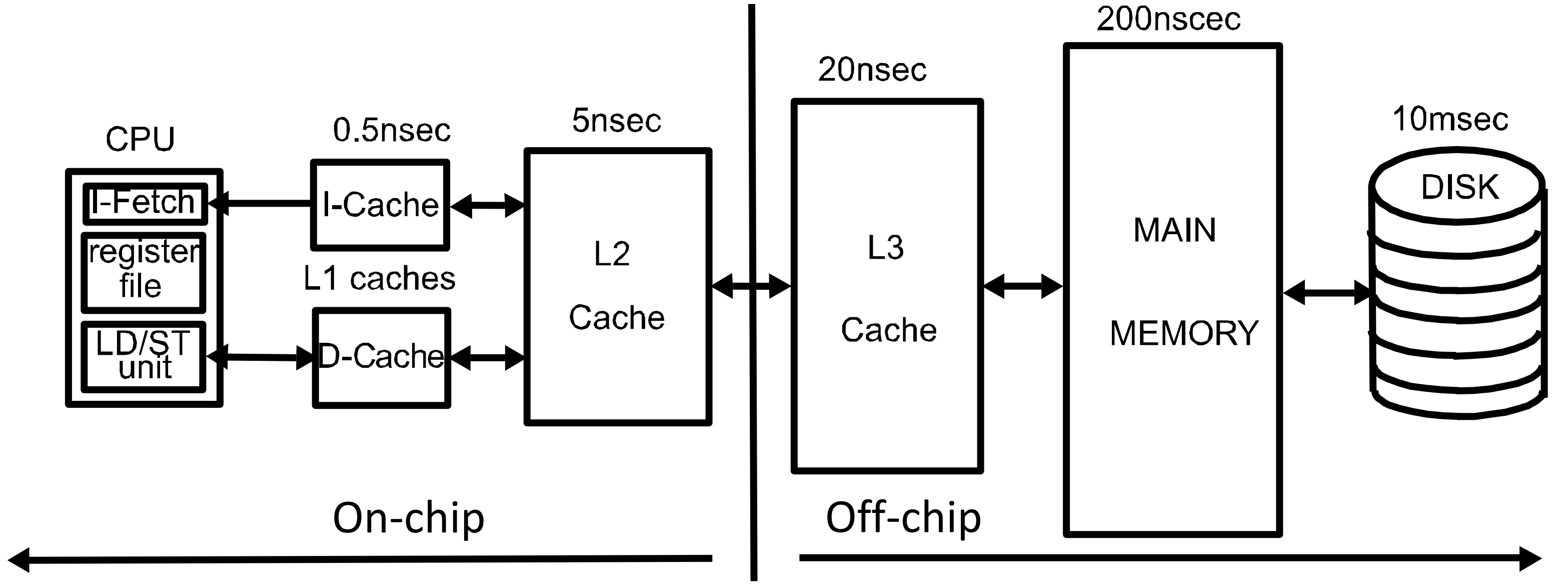
Selection Socket #1 Cores 4 Threads 8

CPU-Z Ver. 1.84.0.x64

- **Linux**
  - lscpu
  - cat /proc/cpuinfo
  - etc.

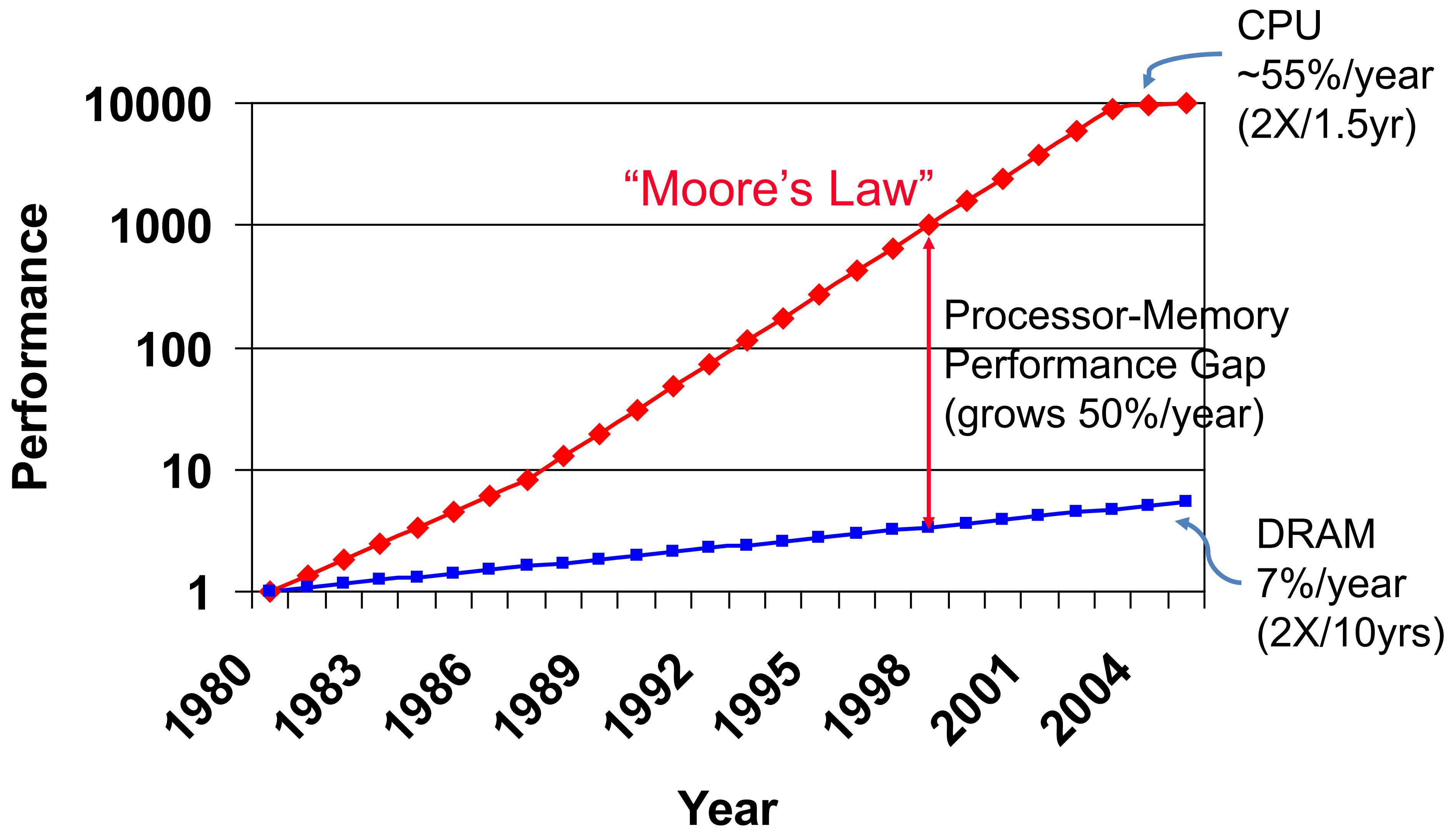
Why?

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



SJSU SAN JOSÉ STATE UNIVERSITY

The “Memory Wall”



SJSU SAN JOSÉ STATE UNIVERSITY

The Memory Hierarchy Goal

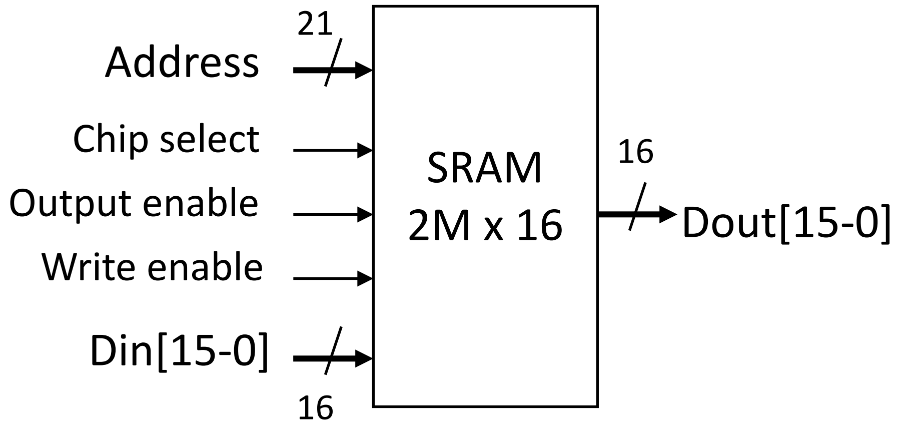
- **How do we create a memory system that gives the illusion of being large, cheap and fast (most of the time)?**
  - With hierarchy
    - try the fast parts first -- most of the time, this works well
    - if not, move the data so it works well the next time
  - With parallelism
    - use multiple identical parts operating simultaneously
    - for large quantities of data, this works well

- **Example – keep a subset of the data in fast memory**
- **Example – 1-byte-wide memory ➔ 4 × 1-byte-wide memory ➔ 4-byte-wide memory**
  - load word takes 4 memory cycles vs. 1 memory cycle

Cache Design

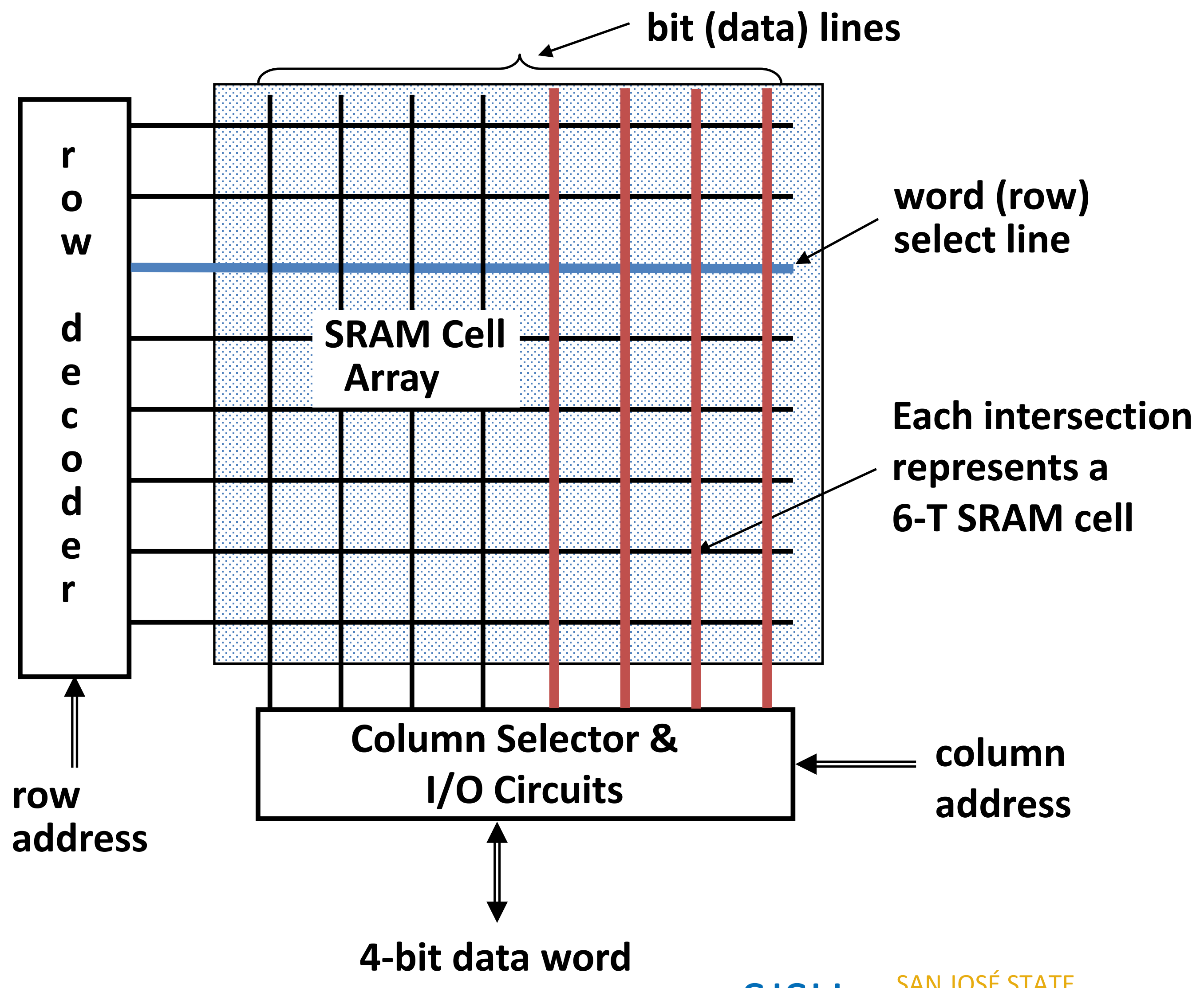
- **Caches use *SRAM* for speed and technology compatibility**
  - Low density (6 transistor cells), high power, expensive, fast
  - Static: content will last "forever" (until power turned off)

- **Example: A (2M X 16 bit) SRAM logic**



SRAM Cache Design

- Each row holds a data block
- Column address selects the requested word from block

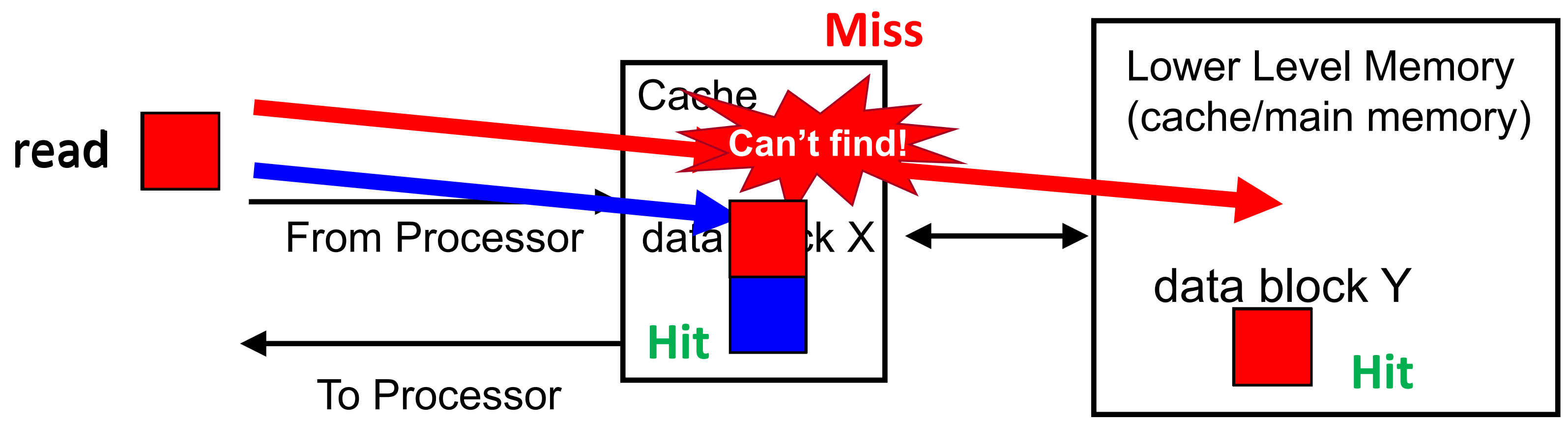


SJSU SAN JOSÉ STATE UNIVERSITY

Cache Hit and Miss

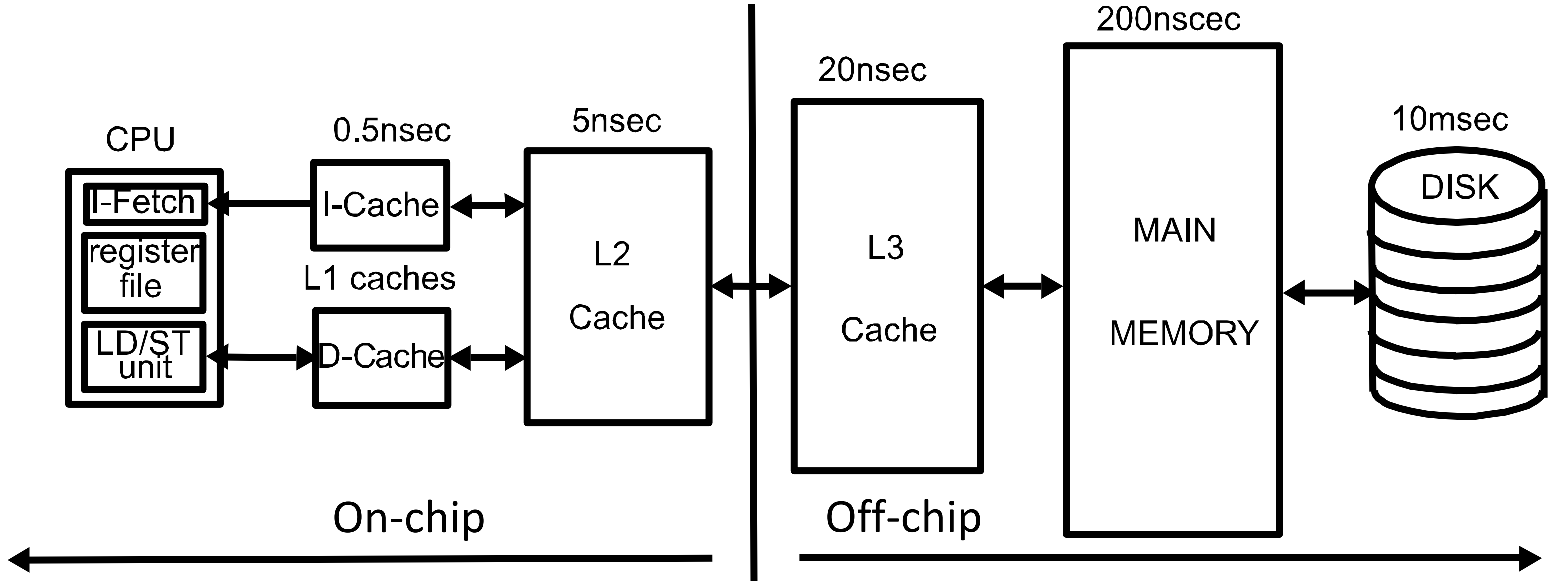
- **Hit:** Data appears in some block of the cache
  - **Hit Rate:** # hits / total accesses on the cache
  - **Hit Time:** Time to access the cache

- **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



Principle of Locality

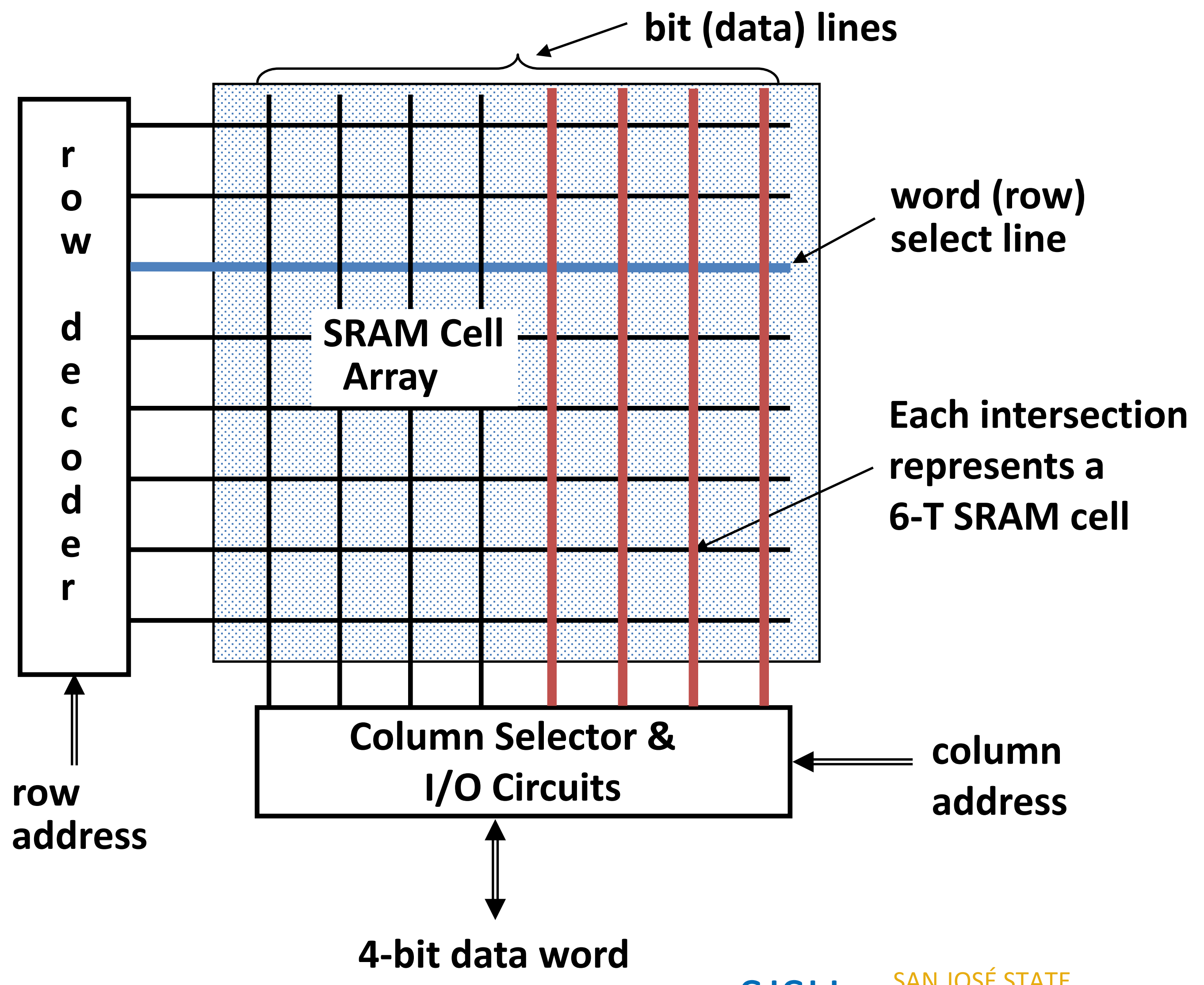
- **Temporal Locality**: If an address is referenced, it tends to be referenced again
- **Spatial Locality**: If an address is referenced, neighboring addresses tend to be referenced
- **How to create a memory system that gives the illusion of being large, cheap and fast?**
  - With hierarchy
  - With parallelism



SJSU SAN JOSÉ STATE UNIVERSITY

SRAM Cache Design

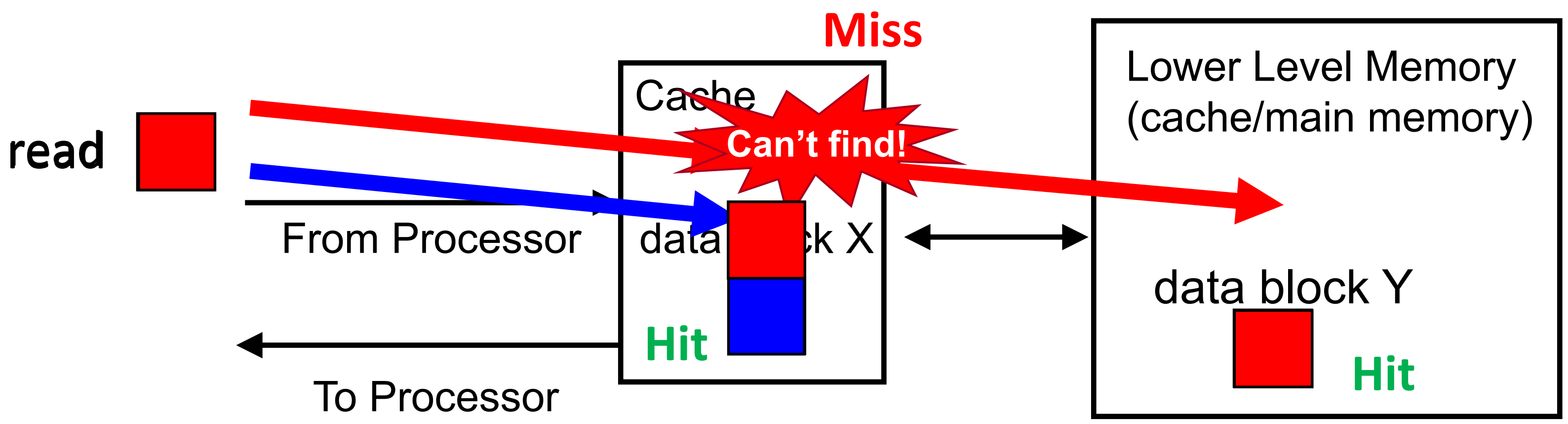
- Each row holds a data block
- Column address selects the requested word from block



Cache Hit and Miss

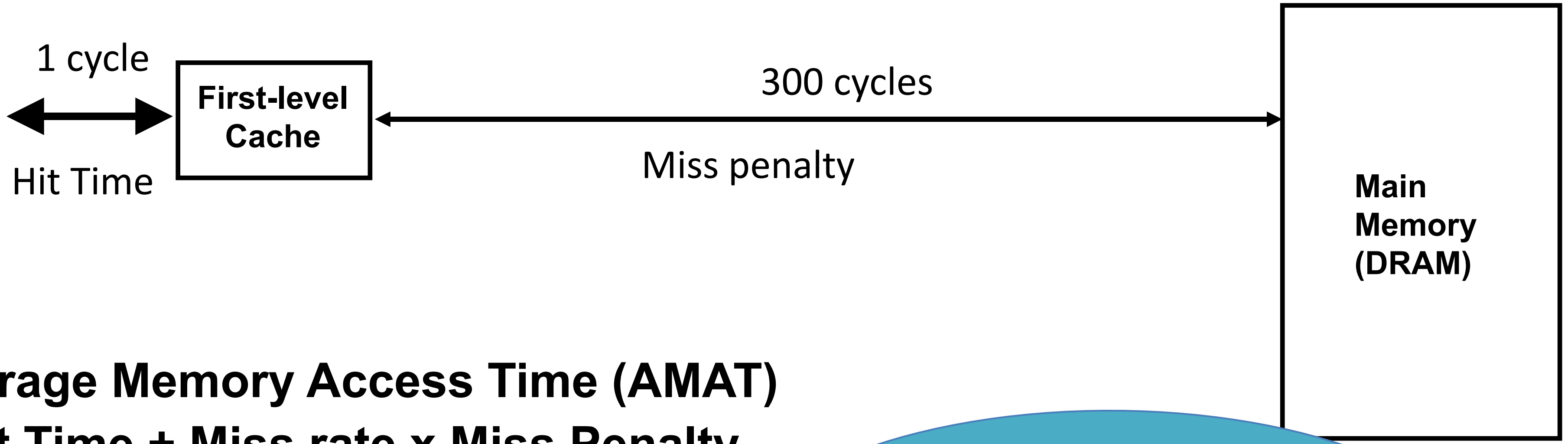
- **Hit:** Data appears in some block of the cache
  - **Hit Rate:** # hits / total accesses on the cache
  - **Hit Time:** Time to access the cache

- **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



SJSU SAN JOSÉ STATE UNIVERSITY

Memory Hierarchy Performance

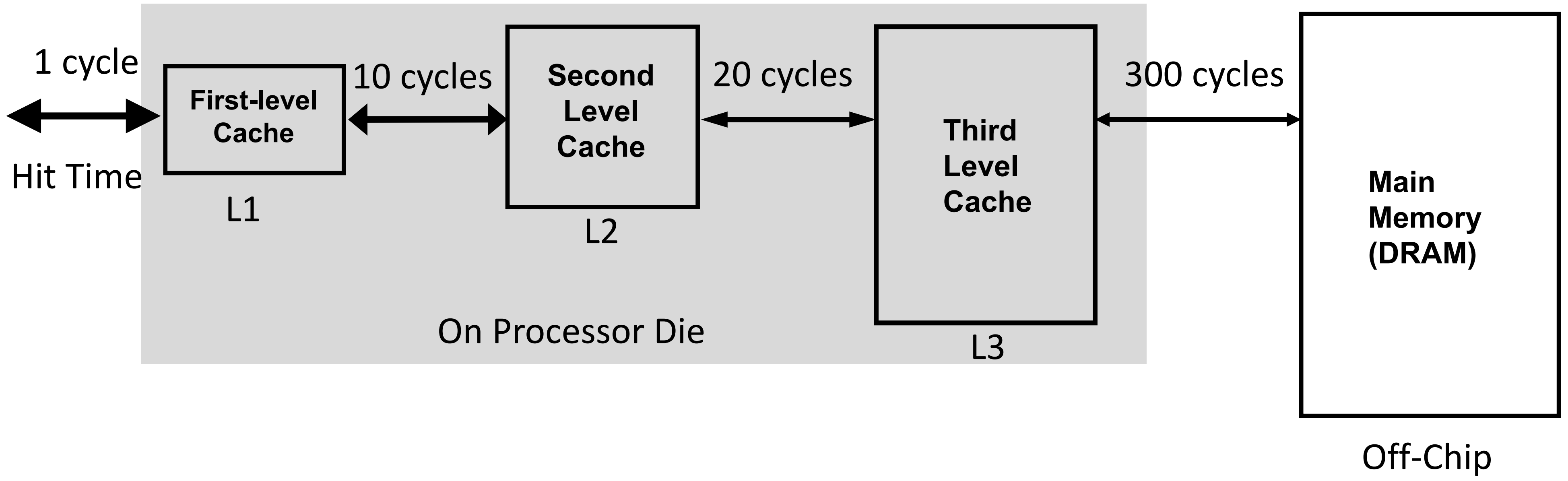


- **Average Memory Access Time (AMAT)**
  = Hit Time + Miss rate x Miss Penalty

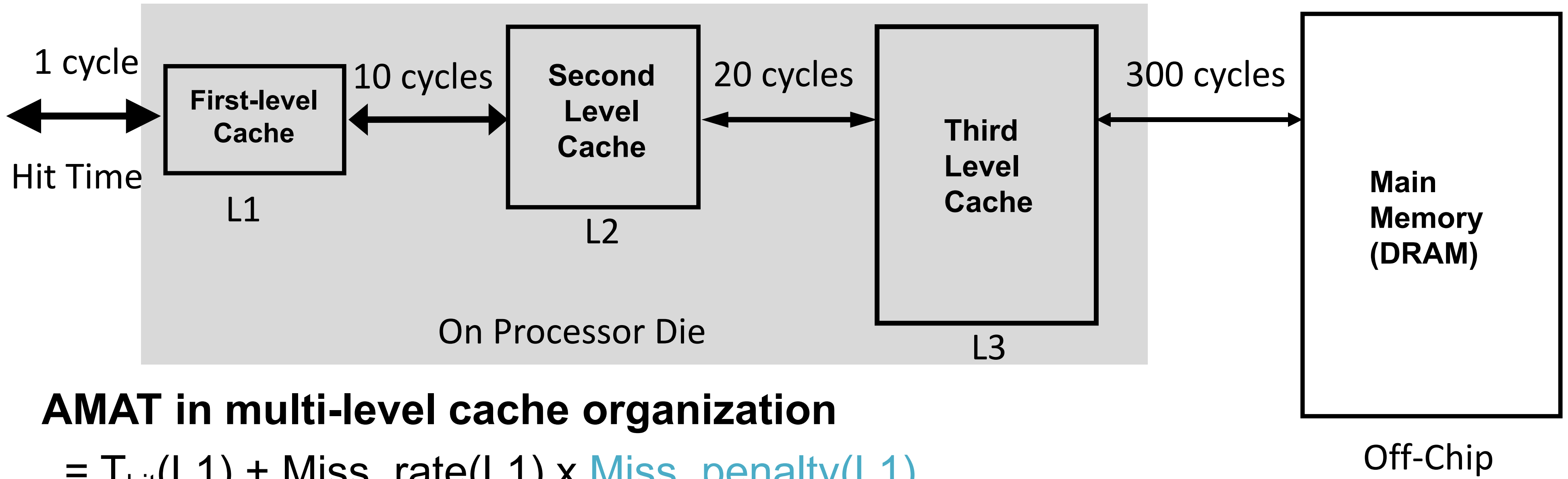
- **Example:**
  - Cache Hit = 1 cycle
  - Miss rate = 10% = 0.1
  - Miss penalty = 300 cycles
  - AMAT = T_{hit}(L1) + Miss_rate(L1) x T(Memory) = 1 + 0.1 x 300 = 31 cycles

Due to long-latency memory, AMAT is 30 cycles longer than cache latency. Can we reduce the overhead?

Reducing Penalty: Multi-Level Cache



Reducing Penalty: Multi-Level Cache

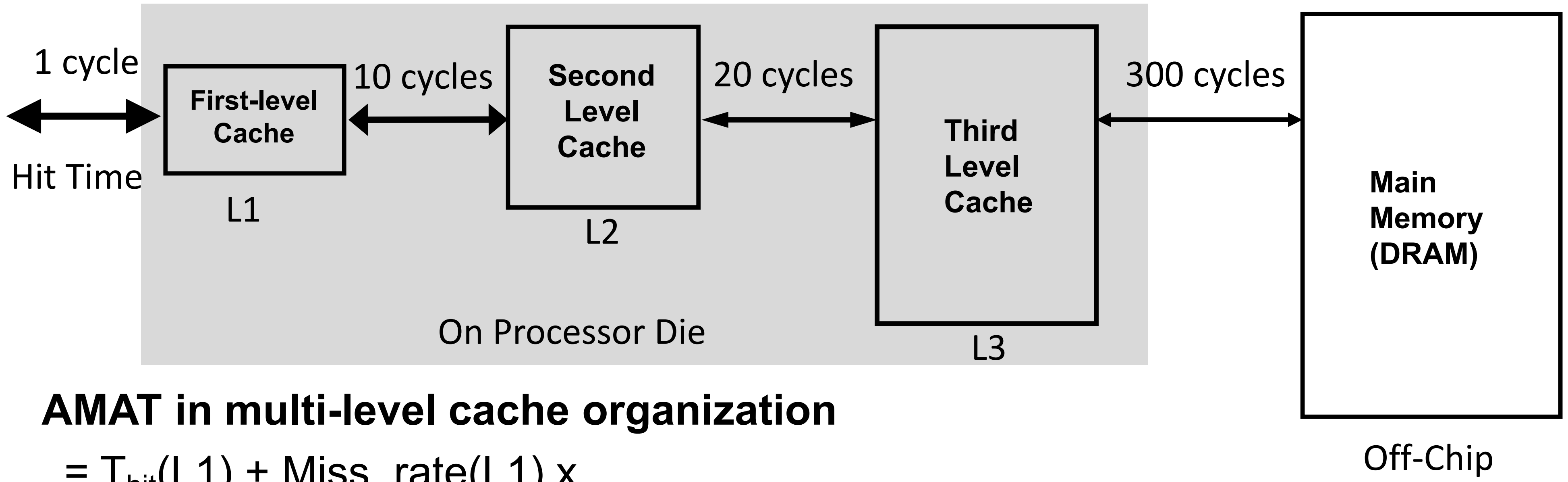


- **AMAT in multi-level cache organization**

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

SJSU SAN JOSÉ STATE UNIVERSITY

Reducing Penalty: Multi-Level Cache



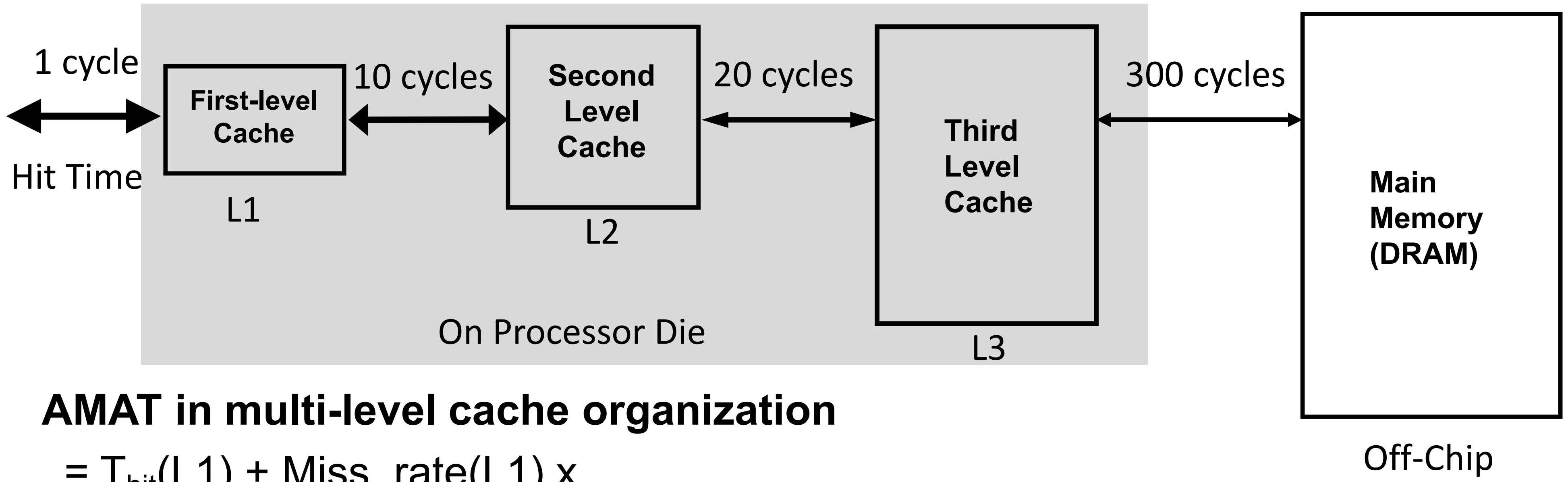
- **AMAT in multi-level cache organization**

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

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

SJSU SAN JOSÉ STATE UNIVERSITY

Reducing Penalty: Multi-Level Cache

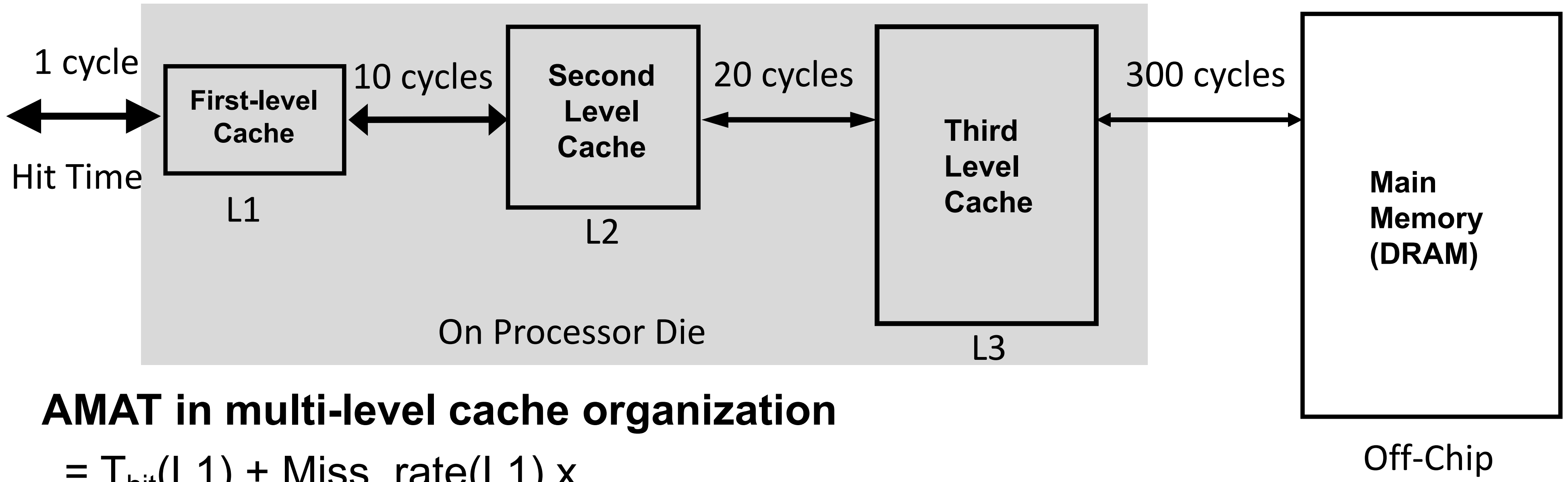


- **AMAT in multi-level cache organization**

  $= T_{hit}(L1) + \text{Miss\_rate}(L1) \times$
  $[\, T_{hit}(L2) + \text{Miss\_rate}(L2) \times$
  $\{\, T_{hit}(L3) + \text{Miss\_rate}(L3) \times \text{Miss\_penalty}(L3) \,\}\,]$

SJSU SAN JOSÉ STATE UNIVERSITY

Reducing Penalty: Multi-Level Cache



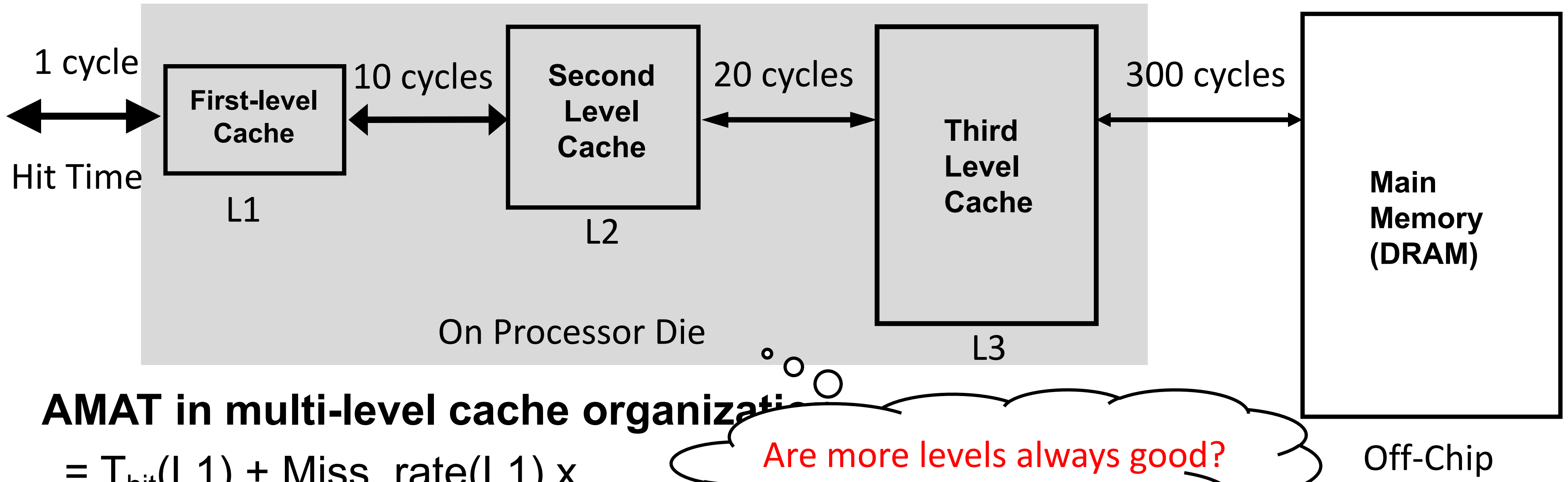
- **AMAT in multi-level cache organization**

  $= T_{hit}(L1) + \text{Miss\_rate}(L1) \times$

  $[\ T_{hit}(L2) + \text{Miss\_rate}(L2) \times$

  $\{\ T_{hit}(L3) + \text{Miss\_rate}(L3) \times T(\text{memory})\ \}\ ]$

Reducing Penalty: Multi-Level Cache



Are more levels always good?

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

  $= T_{hit}(L1) + Miss\_rate(L1) \times$

  $[\ T_{hit}(L2) + Miss\_rate(L2) \times$

  $\{\ T_{hit}(L3) + Miss\_rate(L3) \times T(memory)\ \}\ ]$

- **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!

What to Keep in Caches?

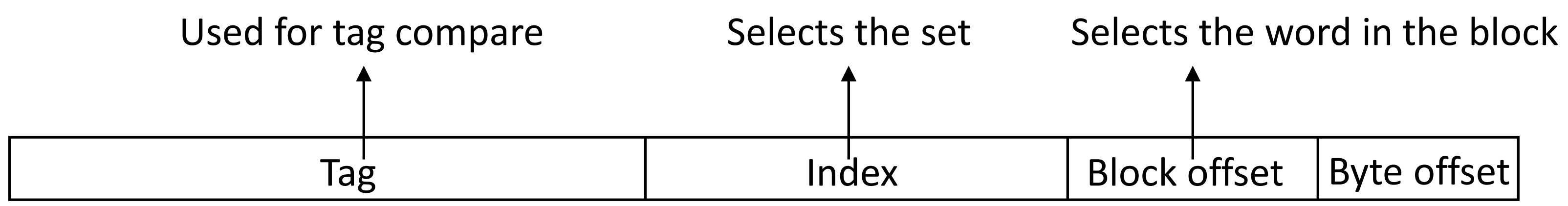
- **It depends on the cache organization and replacing policy**
- **Cache organization:**
  - **Cache line (block):** The basic unit of data replacement. A longer cache line fetches and replaces more data per miss.
  - **Set:** An entry that one cache line is mapped to according to certain bits of its address.
  - **Way:** A slot within a cache set to hold one history cache line.
- **Cache line replacing policy:** which history line should be replaced?
  - In a set entry, each cache line can be identified with a **Tag** (a portion of its address).
  - Least recently used (LRU), First-in first-out (FIFO), Random, etc.

Cache Indexing

Example: a simple 2-set x 2-way cache

	Way 0	Way 1
Set 0	block 0	block 2
Set 1	block 1	block 3

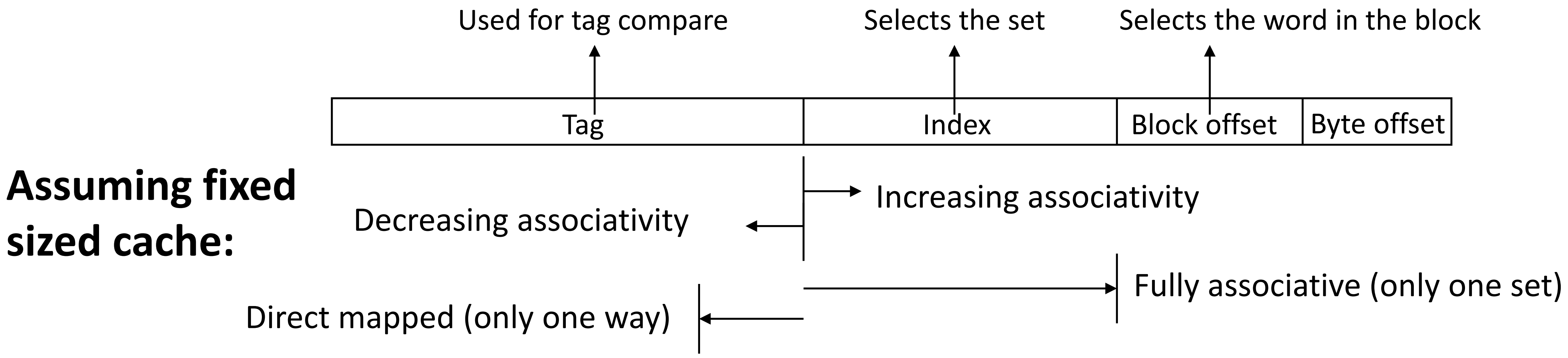
Address decoding:



Cache Types

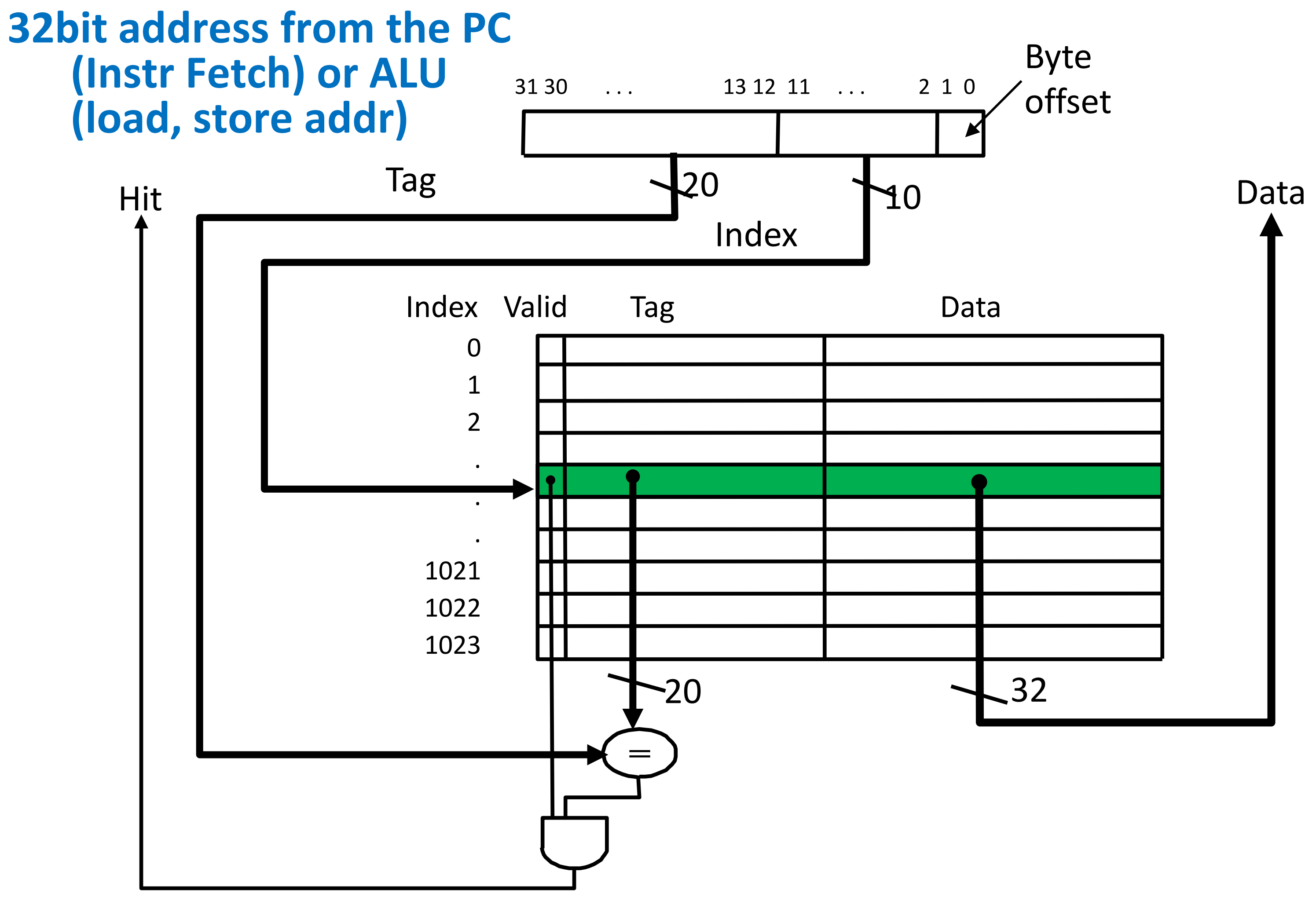
- **N-way Set-Associative:** Number of ways > 1 & Number of sets > 1
  - Slightly complex searching mechanism
- **Direct Mapped:** Number of ways = 1
  - Fast indexing mechanism
- **Fully-Associative:** Number of sets = 1
  - Extensive hardware resources required to search

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


Used for tag compare
Selects the set
Selects the word in the block
Tag
Index
Block offset
Byte offset
Assuming fixed sized cache:
Decreasing associativity
Increasing associativity
Direct mapped (only one way)
Fully associative (only one set)

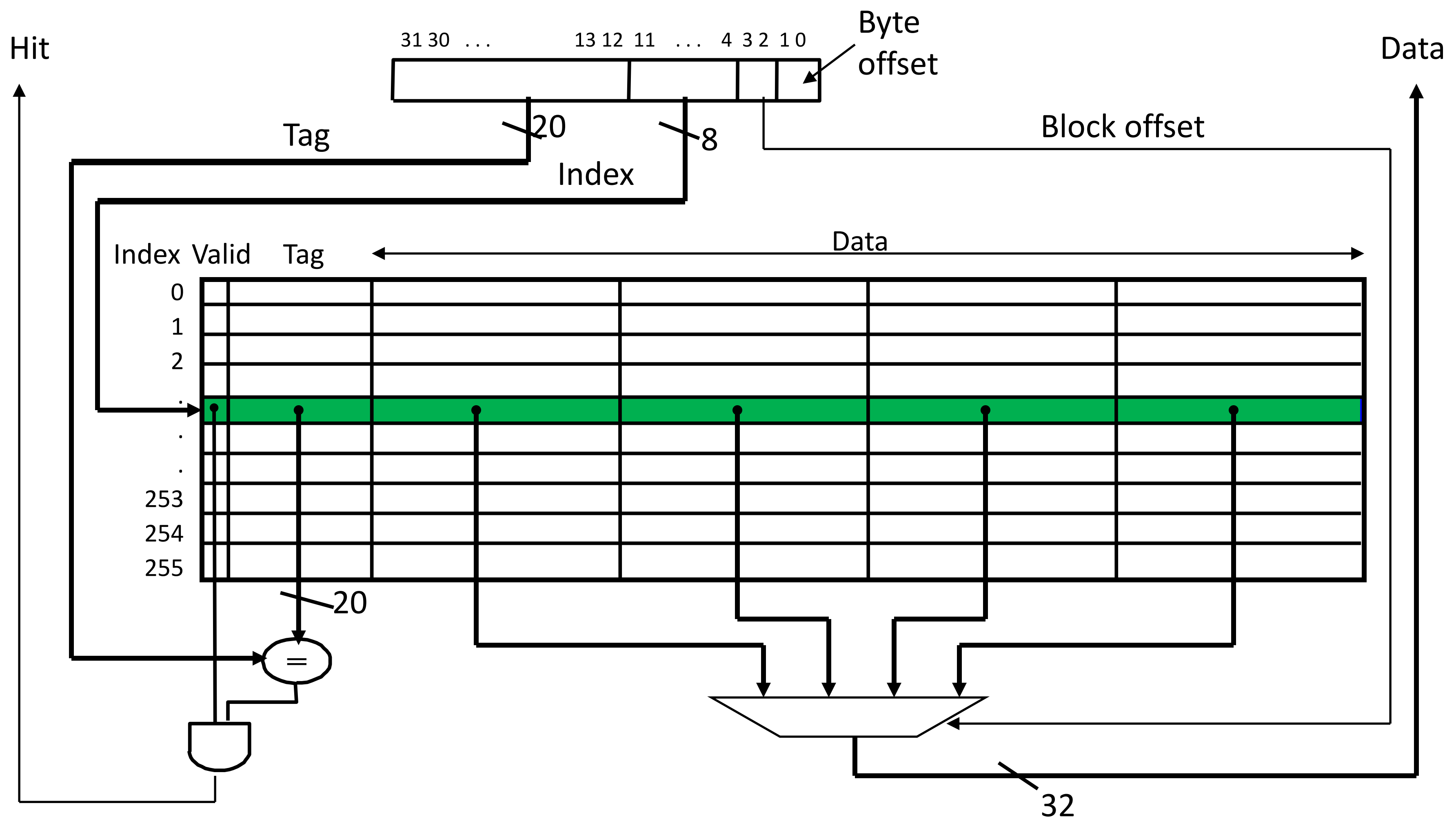
SJSU SAN JOSÉ STATE UNIVERSITY

MIPS Direct Mapped Cache Example



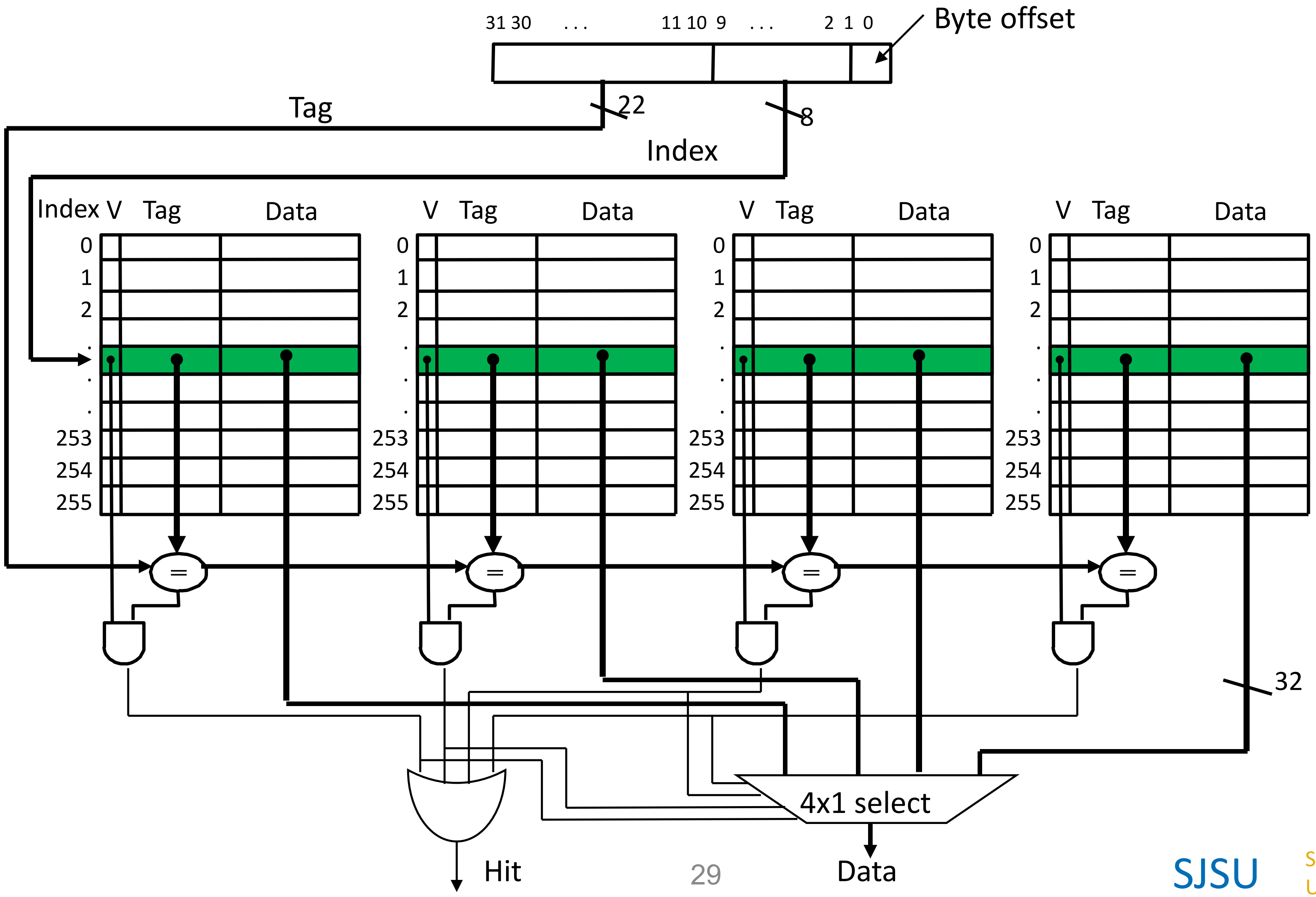
SJSU SAN JOSÉ STATE UNIVERSITY

Multiword Block Direct Mapped Cache



SJSU SAN JOSÉ STATE UNIVERSITY

Four-Way Set Associative Cache



SJSU SAN JOSÉ STATE UNIVERSITY

Costs of Set Associative Caches

- **Must have hardware to keep track of when each way's block was used relative to the other blocks in the set (for replacement policy)**
  - E.g., for 2-way set associative, takes one bit per set → set the bit when a block is referenced (and reset the other way's bit)

- **N-way set associative cache costs**
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data is available after set selection (and Hit/Miss decision). In a direct mapped cache, the cache block is available before the Hit/Miss decision
    - So its not possible to just assume a hit and continue and recover later if it was a miss

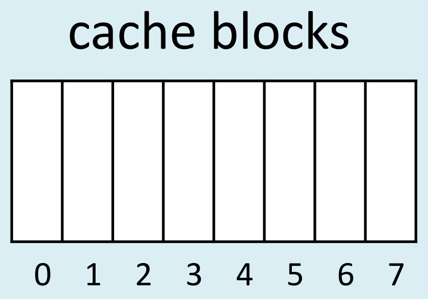
- **Total cache line size = valid field size + tag size + block data size + data for cache policy (e.g., time stamp, modified bit, etc.)**

Cache Miss Classification: The 3 C's

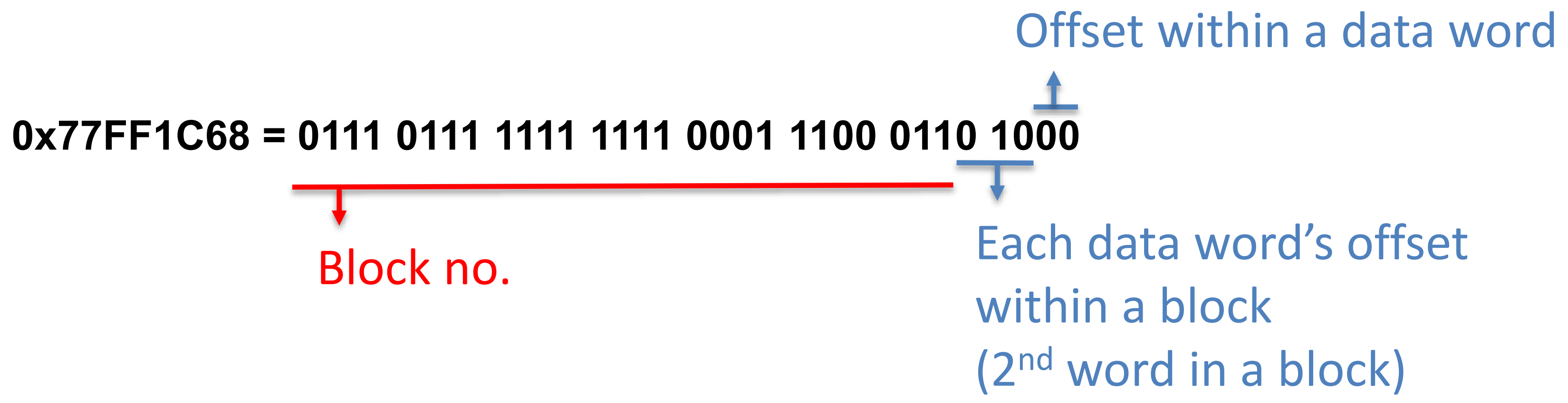
- **Compulsory (cold) Misses**
  - On the 1st reference to a block
  - Related to # blocks accessed by a code, not related to the configuration of a cache
- **Capacity Misses**
  - The program's working set size exceeds the cache capacity
- **Conflict Misses**
  - Multiple memory blocks map to the same set in set-associative caches

More Detailed Mapping Example

- We have a cache that has following configuration
  - Consists of 8 cache blocks (lines)
  - A data word is 4-byte
  - A block is 32-byte (8 words per block)
  - **Direct mapped**
- We load a word from 0x77FF1C68

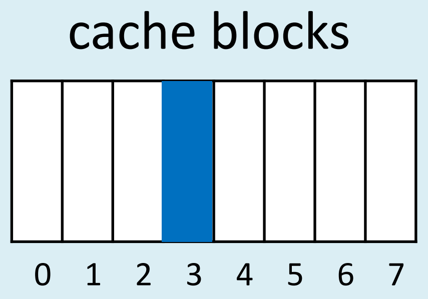


- 1st step: Find data block no. that the word belongs to

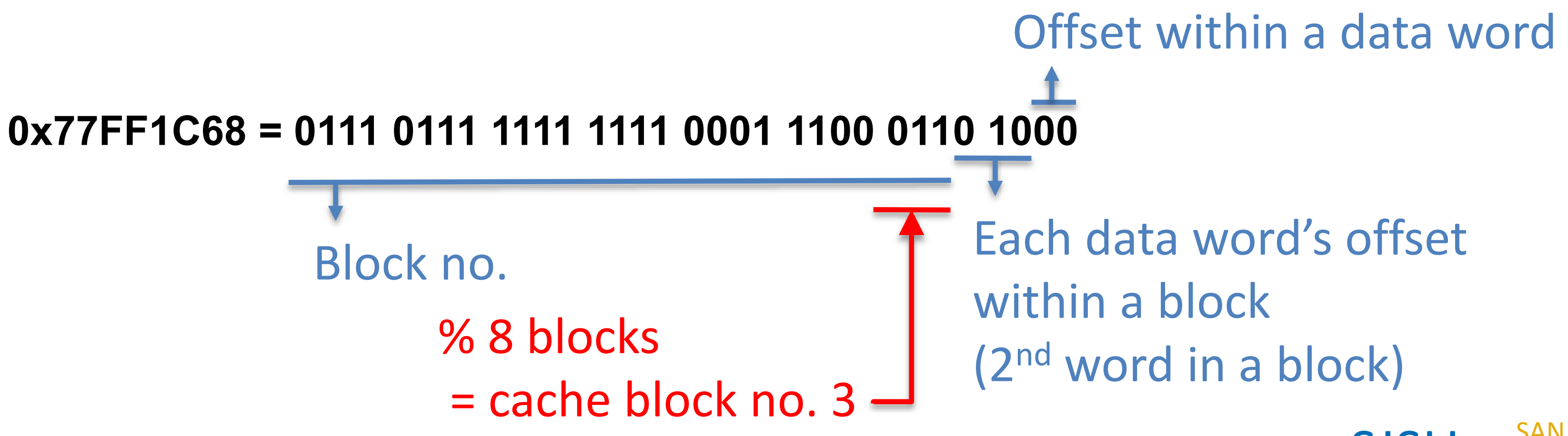

Offset within a data word
0x77FF1C68 = 0111 0111 1111 1111 0001 1100 0110 1000
Block no.
Each data word's offset within a block (2nd word in a block)

More Detailed Mapping Example

- We have a cache that has following configuration
  - Consists of 8 cache blocks (lines)
  - A data word is 4-byte
  - A block is 32-byte (8 words per block)
  - **Direct mapped**
- We load a word from 0x77FF1C68



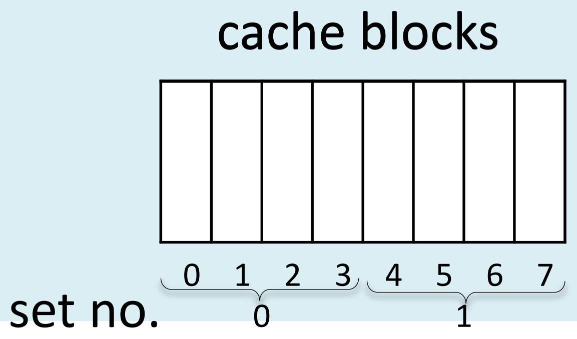
- 2nd step: Map the block no. to cache block no.



Offset within a data word
0x77FF1C68 = 0111 0111 1111 1111 0001 1100 0110 1000
Block no.
% 8 blocks
= cache block no. 3
Each data word's offset within a block (2nd word in a block)

More Detailed Mapping Example

- We have a cache that has following configuration
  - Consists of 8 cache blocks (lines)
  - A data word is 4-byte
  - A block is 32-byte (8 words per block)
  - **4-way Set-Associative**
- We load a word from 0x77FF1C68

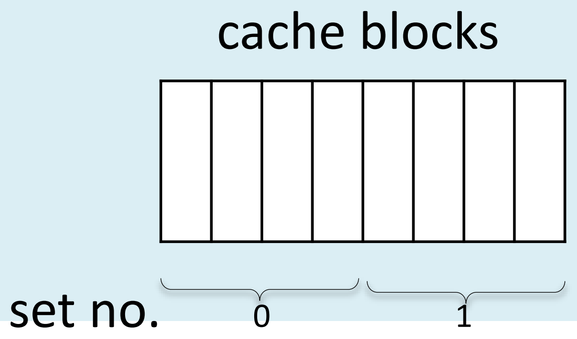


- 1st step: Calculate how many sets exist

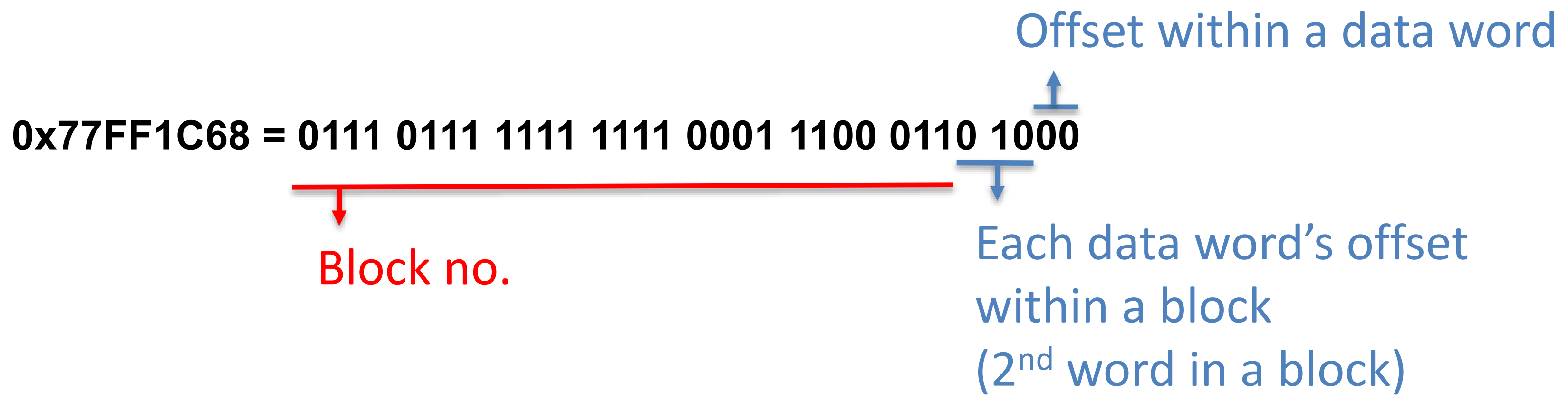
  8 blocks / 4 entries per set = 2 sets

More Detailed Mapping Example

- We have a cache that has following configuration
  - Consists of 8 cache blocks (lines)
  - A data word is 4-byte
  - A block is 32-byte (8 words per block)
  - **4-way Set-Associative**
- We load a word from 0x77FF1C68



- 2nd step: Find data block no. that the word belongs to

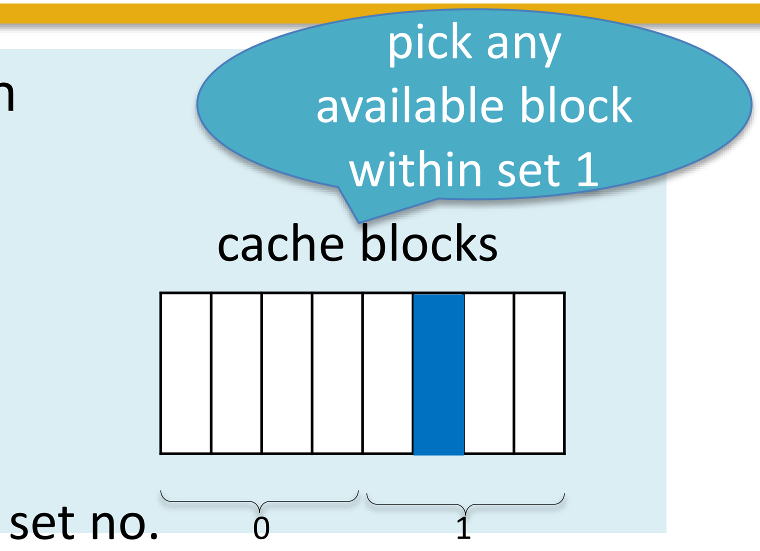

0x77FF1C68 = 0111 0111 1111 1111 0001 1100 0110 1000

Offset within a data word
Block no.
Each data word's offset within a block (2nd word in a block)

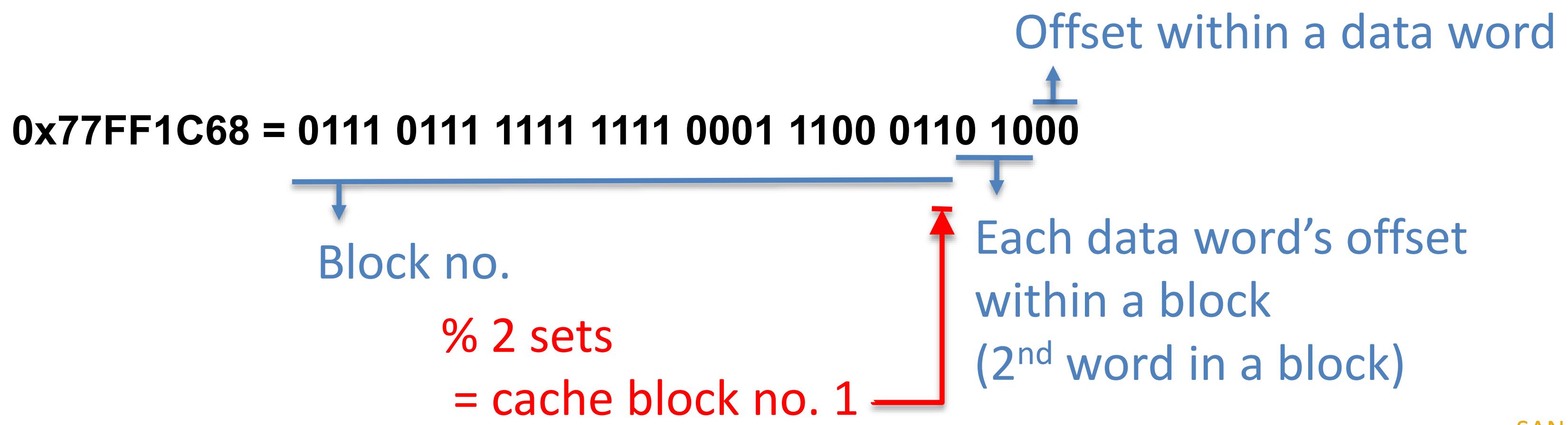
SJSU SAN JOSÉ STATE UNIVERSITY

More Detailed Mapping Example

- We have a cache that has following configuration
  - Consists of 8 cache blocks (lines)
  - A data word is 4-byte
  - A block is 32-byte (8 words per block)
  - **4-way Set-Associative**
- We load a word from 0x77FF1C68



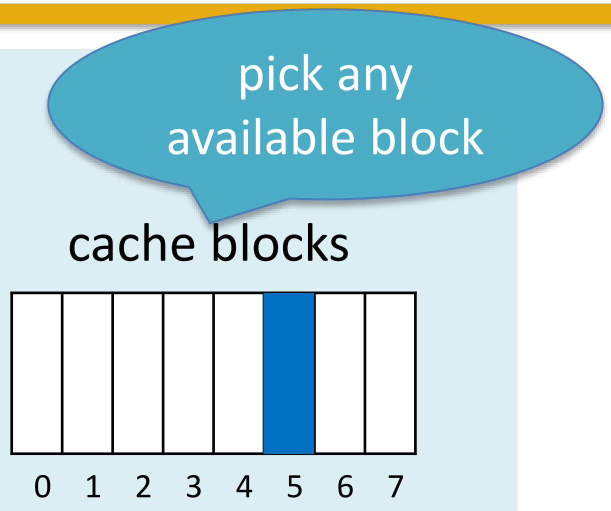
- 3rd step: Map the block no. to cache block no.


Offset within a data word
0x77FF1C68 = 0111 0111 1111 1111 0001 1100 0110 1000
Block no.
% 2 sets
= cache block no. 1
Each data word's offset within a block
(2nd word in a block)

SJSU SAN JOSÉ STATE UNIVERSITY

More Detailed Mapping Example

- We have a cache that has following configuration
  - Consists of 8 cache blocks (lines)
  - A data word is 4-byte
  - A block is 32-byte (8 words per block)
  - **Fully-Associative**
- We load a word from 0x77FF1C68



- 1st and all step: Find any available cache block and map

SJSU SAN JOSÉ STATE UNIVERSITY

Cache Replacement Policy

- **When loading a new block (on miss), if the cache is already full, which block should be replaced (in the set)?**
  - Random: Replace a randomly chosen line
  - FIFO: Replace the oldest line
  - LRU (Least Recently Used): Replace the least recently used line
  - Many others: Round Robin, LIFO, MRU, etc..

LRU Example

- **Cache configuration:**
  - Size: 4 blocks
  - Mapping: fully-associative
  - Replacement: LRU
- **Mem access sequence: (each character indicates a block address)**
  - ABCDCDECG

Accesses		A	B	C	D	C	D	E	C	G
priority order (LRU)	0 (MRU)	A								
	1									
	2									
	3 (LRU)									
cache miss		**m**								

LRU Example

- **Cache configuration:**
  - Size: 4 blocks
  - Mapping: fully-associative
  - Replacement: LRU
- **Mem access sequence: (each character indicates a block address)**
  - ABCDCDECG

Accesses		A	B	C	D	C	D	E	C	G
priority order (LRU)	0 (MRU)	A	B							
	1		A							
	2									
	3 (LRU)									
cache miss		**m**	**m**							

LRU Example

- **Cache configuration:**
  - Size: 4 blocks
  - Mapping: fully-associative
  - Replacement: LRU
- **Mem access sequence: (each character indicates a block address)**
  - ABCDCDECG

Accesses		A	B	C	D	C	D	E	C	G
priority order (LRU)	0 (MRU)	A	B	C						
	1		A	B						
	2			A						
	3 (LRU)									
cache miss		**m**	**m**	**m**						

LRU Example

- **Cache configuration:**
  - Size: 4 blocks
  - Mapping: fully-associative
  - Replacement: LRU
- **Mem access sequence: (each character indicates a block address)**
  - ABCDCDECG

Accesses		A	B	C	D	C	D	E	C	G
priority order (LRU)	0 (MRU)	A	B	C	D					
	1		A	B	C					
	2			A	B					
	3 (LRU)				A					
cache miss		**m**	**m**	**m**	**m**					

LRU Example

- **Cache configuration:**
  - Size: 4 blocks
  - Mapping: fully-associative
  - Replacement: LRU
- **Mem access sequence: (each character indicates a block address)**
  - ABCDCDECG

Accesses		A	B	C	D	C	D	E	C	G
priority order (LRU)	0 (MRU)	A	B	C	D	C				
	1		A	B	C	D				
	2			A	B	B				
	3 (LRU)				A	A				
cache miss		**m**	**m**	**m**	**m**	h				

LRU Example

- **Cache configuration:**
  - Size: 4 blocks
  - Mapping: fully-associative
  - Replacement: LRU
- **Mem access sequence: (each character indicates a block address)**
  - ABCDCDECG

Accesses		A	B	C	D	C	D	E	C	G
priority order (LRU)	0 (MRU)	A	B	C	D	C	D			
	1		A	B	C	D	C			
	2			A	B	B	B			
	3 (LRU)				A	A	A			
cache miss		**m**	**m**	**m**	**m**	h	h			

LRU to be replaced for E

LRU Example

- **Cache configuration:**
  - Size: 4 blocks
  - Mapping: fully-associative
  - Replacement: LRU
- **Mem access sequence: (each character indicates a block address)**
  - ABCDCDECG

Accesses		A	B	C	D	C	D	E	C	G
priority order (LRU)	0 (MRU)	A	B	C	D	C	D	E		
	1		A	B	C	D	C	D		
	2			A	B	B	B	C		
	3 (LRU)				A	A	A	B		
cache miss		**m**	**m**	**m**	**m**	h	h	**m**		

LRU Example

- **Cache configuration:**
  - Size: 4 blocks
  - Mapping: fully-associative
  - Replacement: LRU
- **Mem access sequence: (each character indicates a block address)**
  - ABCDCDECG

Accesses		A	B	C	D	C	D	E	C	G
priority order (LRU)	0 (MRU)	A	B	C	D	C	D	E	C	
	1		A	B	C	D	C	D	E	
	2			A	B	B	B	C	D	
	3 (LRU)				A	A	A	B	B	
cache miss		**m**	**m**	**m**	**m**	h	h	**m**	h	

LRU to be replaced for G

LRU Example

- **Cache configuration:**
  - Size: 4 blocks
  - Mapping: fully-associative
  - Replacement: LRU
- **Mem access sequence: (each character indicates a block address)**
  - ABCDCDECG

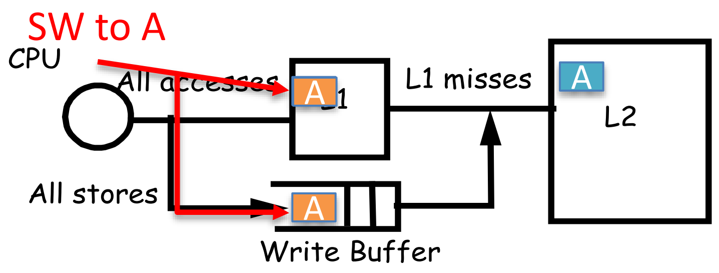
Accesses		A	B	C	D	C	D	E	C	G
priority order (LRU)	0 (MRU)	A	B	C	D	C	D	E	C	G
	1		A	B	C	D	C	D	E	C
	2			A	B	B	B	C	D	E
	3 (LRU)				A	A	A	B	B	D
cache miss		**m**	**m**	**m**	**m**	h	h	**m**	h	**m**

Cache Policies

- **Read policies:**
  - Read Hit: this is what we want
  - Read Miss: replacement policy
- **Write policies (L1D only): consistency issues & performance tradeoffs**
  - Write-through vs. write-back vs. write-evict (no-write)
    - also slightly different between write hit and write miss
  - Write Hit: write policy
  - Write Miss: write allocation policies (2 policies to categorize 3 write policies) + replacement policy

Cache Write Policy

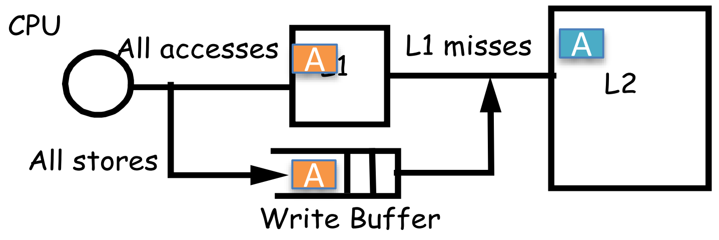
- **Write-through: consistent with lower level**
  - The value is written to both the cache line and the lower-level memory.
    - No need to write a full block
  - Write buffer can be used so cache does not need to wait for the write to lower level
    - stall only if the write buffer is full
  - Example: assume cache block containing word address A is initially in the cache



SJSU SAN JOSÉ STATE UNIVERSITY

Cache Write Policy

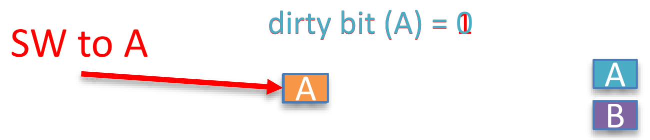
- **Write-through: consistent with lower level**
  - The value is written to both the cache line and the lower-level memory.
    - No need to write a full block
  - Write buffer can be used so cache does not need to wait for the write to lower level
    - stall only if the write buffer is full
  - Example: assume cache block containing word address A is initially in the cache



SJSU SAN JOSÉ STATE UNIVERSITY

Cache Write Policy

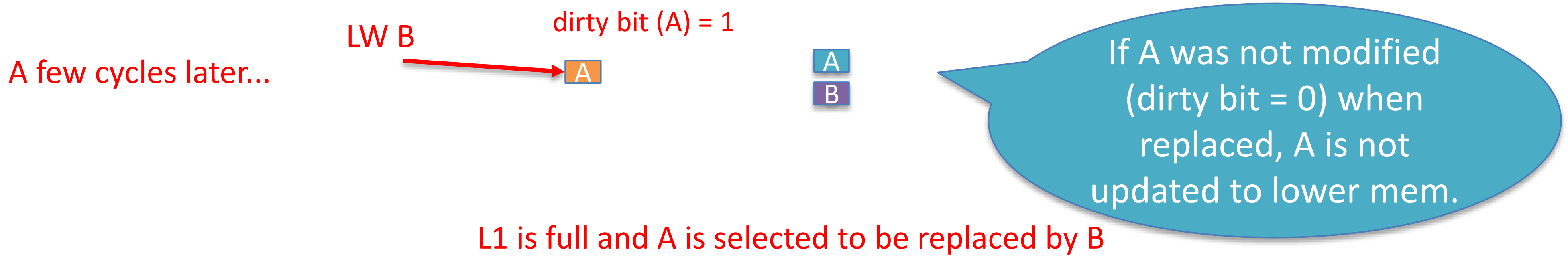
- **Write-back: inconsistent with lower level**
  - The value is written only to the cache line. The modified (dirty) cache line is written to main memory only when it is evicted.
    - 1 dirty bit is needed for each cache line.
  - A write-back buffer to update lower level with evicted dirty blocks
    - Must write a full block at this point since we do not know which word is modified
  - Example: assume cache block containing word address A is initially in the cache



SJSU SAN JOSÉ STATE UNIVERSITY

Cache Write Policy

- **Write-back: inconsistent with lower level**
  - The value is written only to the cache line. The modified (dirty) cache line is written to main memory only when it is evicted.
    - 1 dirty bit is needed for each cache line.
  - A write-back buffer to update lower level with evicted dirty blocks
    - Must write a full block at this point since we do not know which word is modified
  - Example: assume cache block containing word address A is initially in the cache


A few cycles later...
LW B
dirty bit (A) = 1
A
A
B
If A was not modified (dirty bit = 0) when replaced, A is not updated to lower mem.
L1 is full and A is selected to be replaced by B

SJSU SAN JOSÉ STATE UNIVERSITY

Cache Write Policy

- **Write-evict (no-write/bypassing): no consistency issue**
  - The value is written only to the lower level.
  - If the block containing the write address exists in the cache, evict it.
  - The write only need to send the data specified by the store instruction.
  - A write buffer can also be used to reduce the delay.

Cache Policies

- **Read policies:**
  - Read Hit: this is what we want
  - Read Miss: replacement policy + write policy of evicted data
- **Write policies: consistency issues & performance tradeoffs**
  - Write-through vs. write-back vs. write-evict (no-write)
    - also slightly different between write hit and write miss
  - Write Hit: write policy
  - Write Miss: write allocation policy (+ replacement policy) + write policy of evicted data

Cache Write Policy

- **Read miss:**
  - Write-through: find victim block + fetch block from lower level
  - write-back: find 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 allocation policies**
  - Write-allocate:
    - Write-through: find victim block + fetch block from lower level + store value to block and lower level
    - write-back: find victim block + write back evicted block if dirty + fetch block from lower level + store value to block

  - No-write-allocate: bypass the cache, store value to lower level directly

- **Write hit and write miss can use different policies**
  - e.g., write-hit write-evict + write-miss write-allocate-write-back

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

Loop Interchange

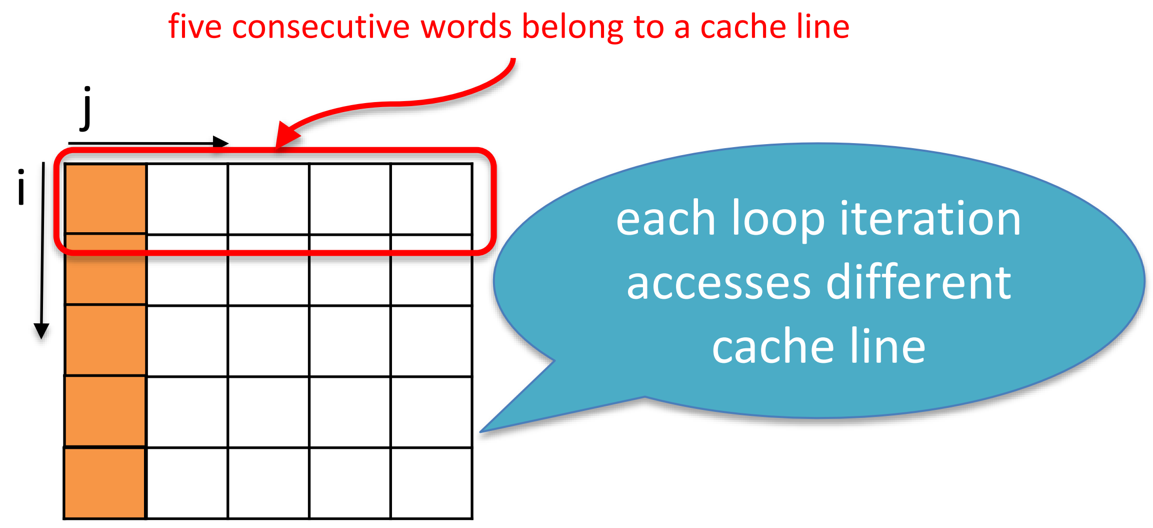
Assume: cache line size = 5 words
What addresses are accessed in each iteration of inner loop?

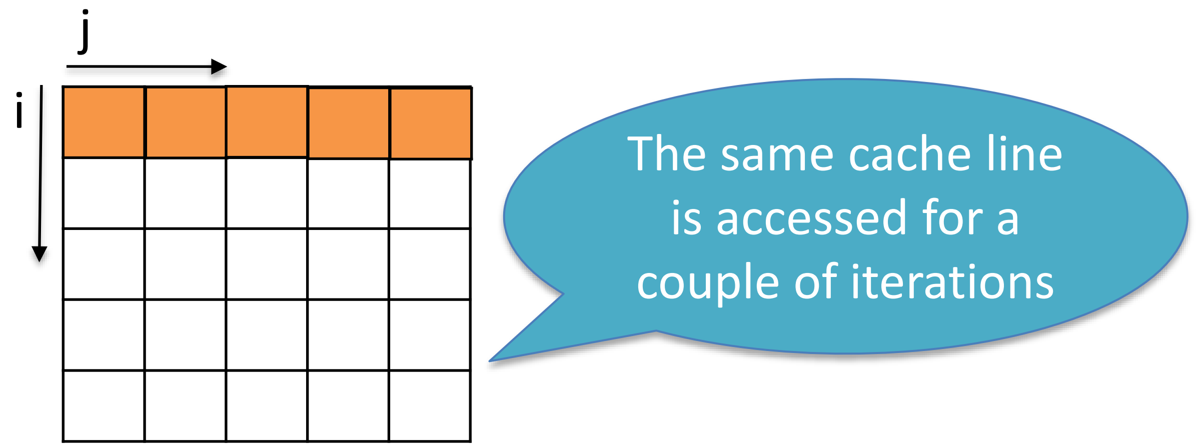
```
/* Before */
for (j=0; j<5; j++)
 for (i=0; i<5; i++)
   x[i][j] = 2*x[i][j]
```

Column-major ordering

Row-major ordering

```
/* After */
for (i=0; i<5; i++)
 for (j=0; j<5; j++)
   x[i][j] = 2*x[i][j]
```


five consecutive words belong to a cache line
j
i
each loop iteration accesses different cache line

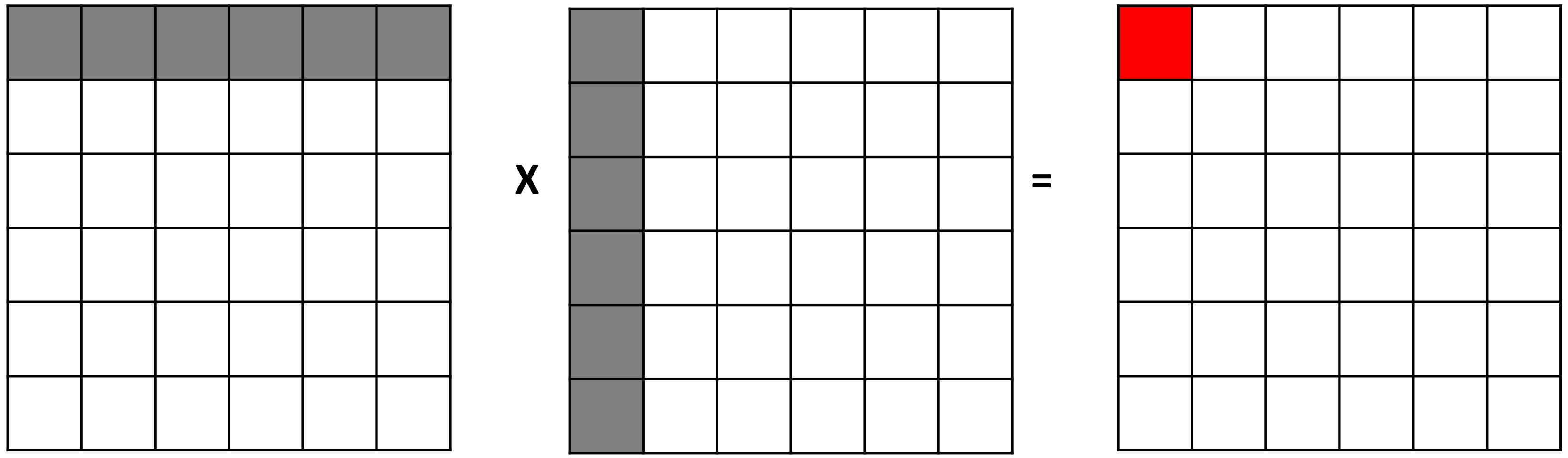


SJSU SAN JOSÉ STATE UNIVERSITY

Loop Blocking

In the traditional matrix multiplication:

```
/* Before */
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

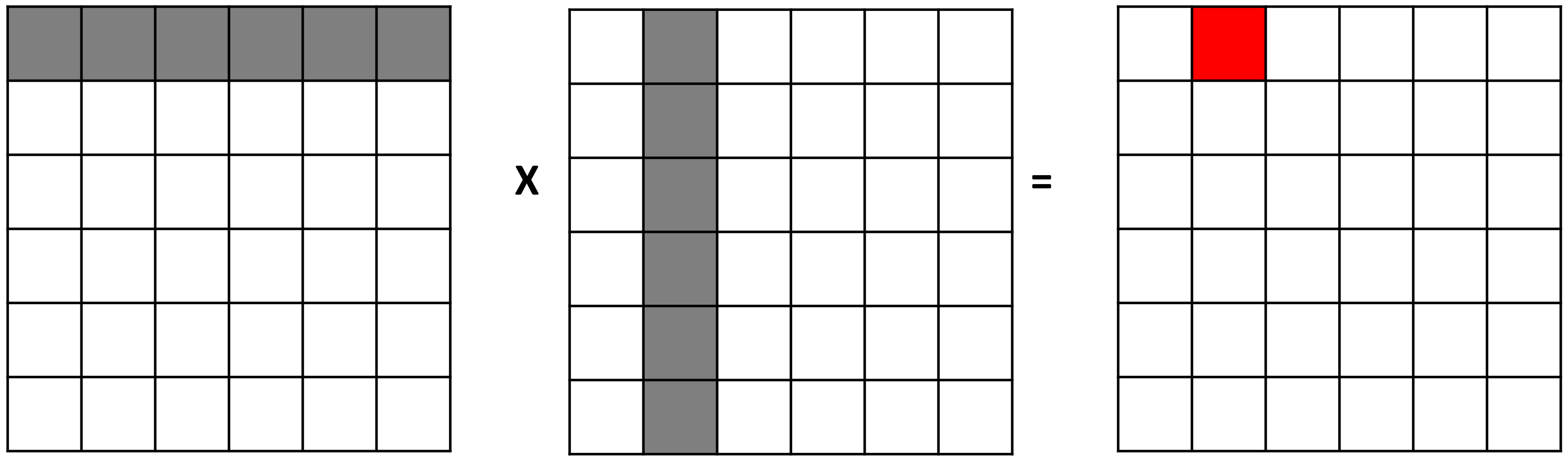


To generate one output element (red color), the 6 elements of each input matrix (gray color) will be accessed.

Loop Blocking

In the traditional matrix multiplication:

```
/* Before */
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

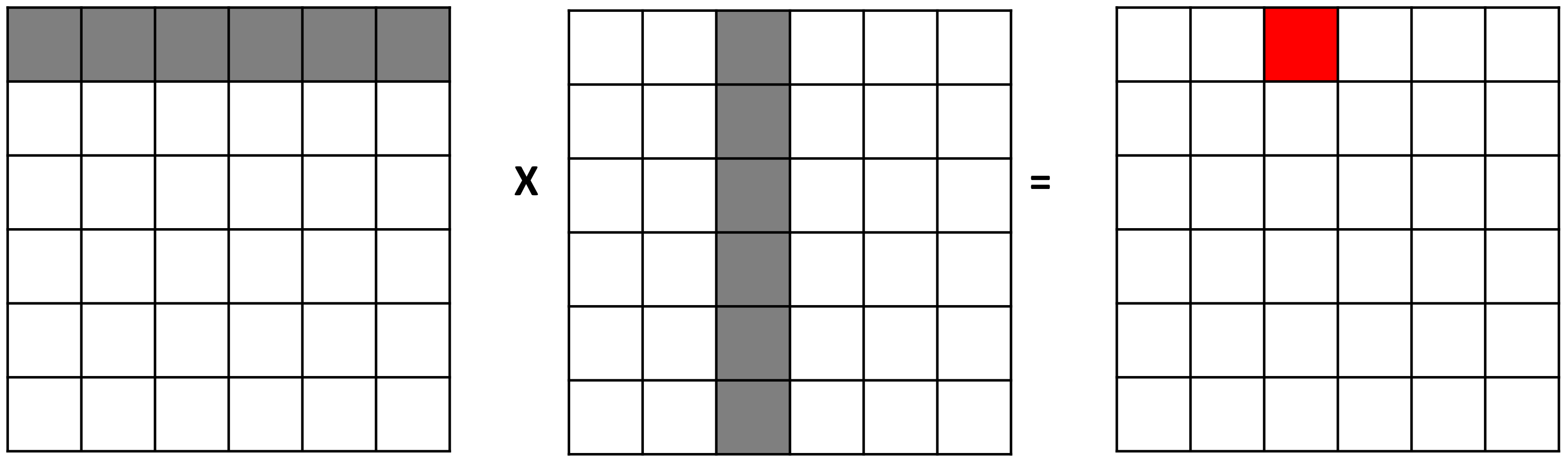


To generate one output element (red color), the 6 elements of each input matrix (gray color) will be accessed.

Loop Blocking

In the traditional matrix multiplication:

```
/* Before */
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j] = C[i][j] + A[i][k] * B[k][j];
```



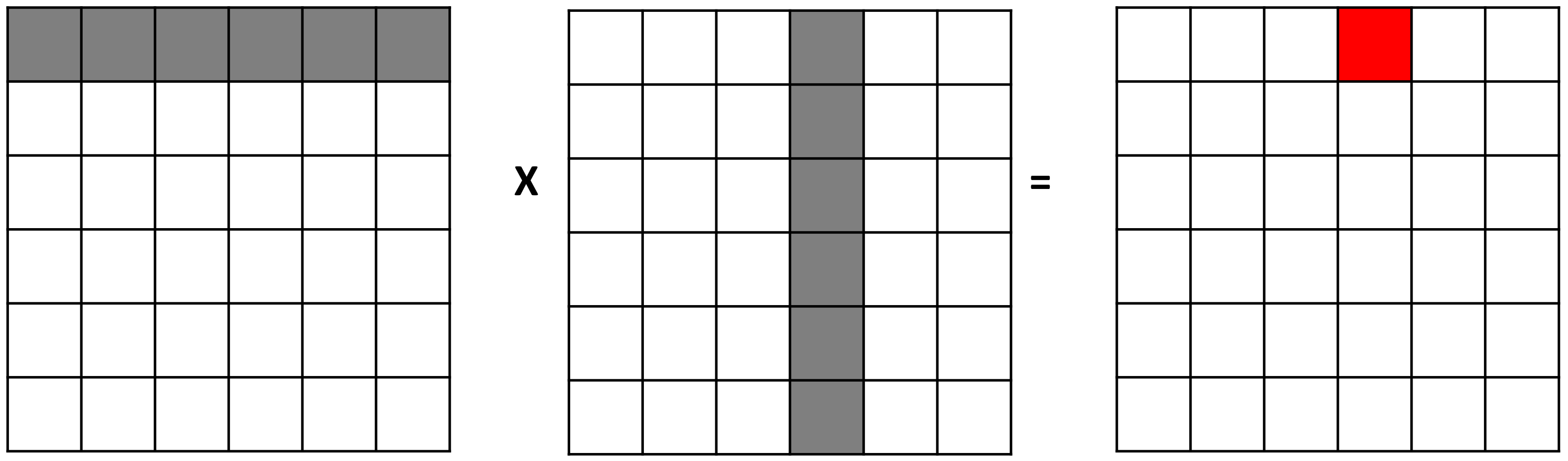
To generate one output element (red color), the 6 elements of each input matrix (gray color) will be accessed.

SJSU SAN JOSÉ STATE UNIVERSITY

Loop Blocking

In the traditional matrix multiplication:

```
/* Before */
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j] = C[i][j] + A[i][k] * B[k][j];
```



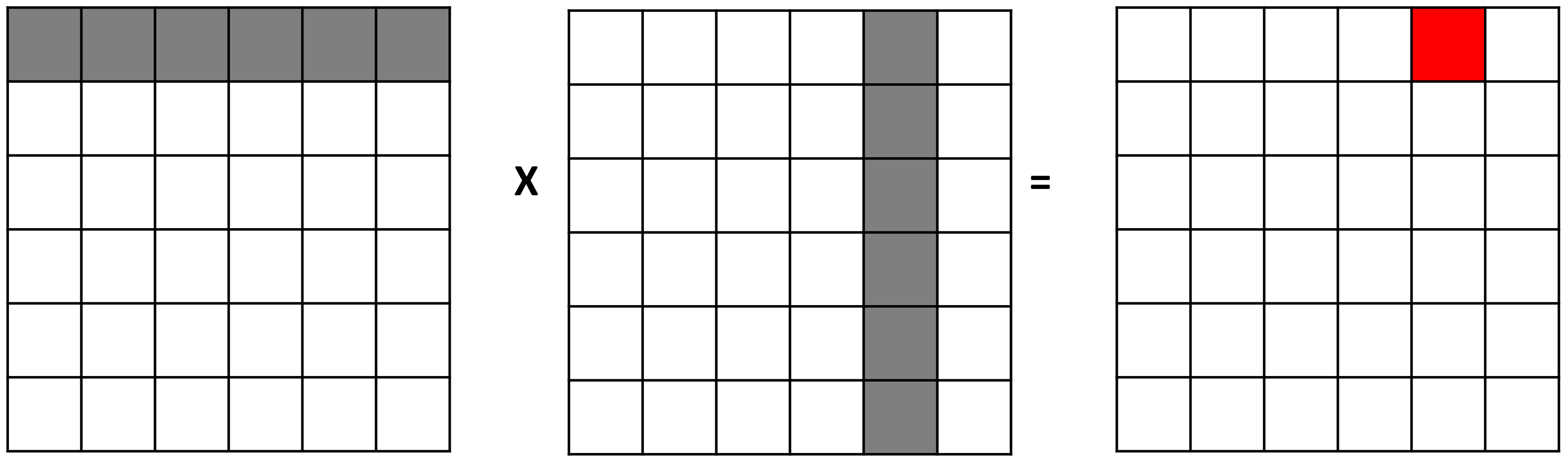
To generate one output element (red color), the 6 elements of each input matrix (gray color) will be accessed.

SJSU SAN JOSÉ STATE UNIVERSITY

Loop Blocking

In the traditional matrix multiplication:

```
/* Before */
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j] = C[i][j] + A[i][k] * B[k][j];
```



To generate one output element (red color), the 6 elements of each input matrix (gray color) will be accessed.

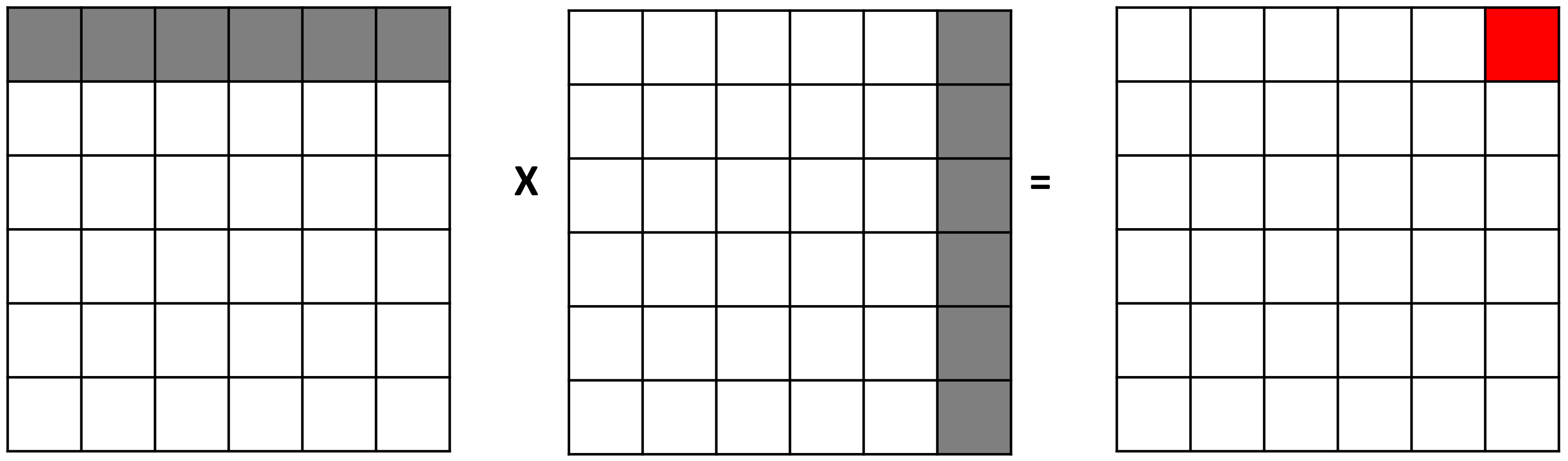
SJSU SAN JOSÉ STATE UNIVERSITY

Loop Blocking

In the traditional matrix multiplication:

```
/* Before */
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

The second matrix may always encounter misses because individual data are from different cache blocks



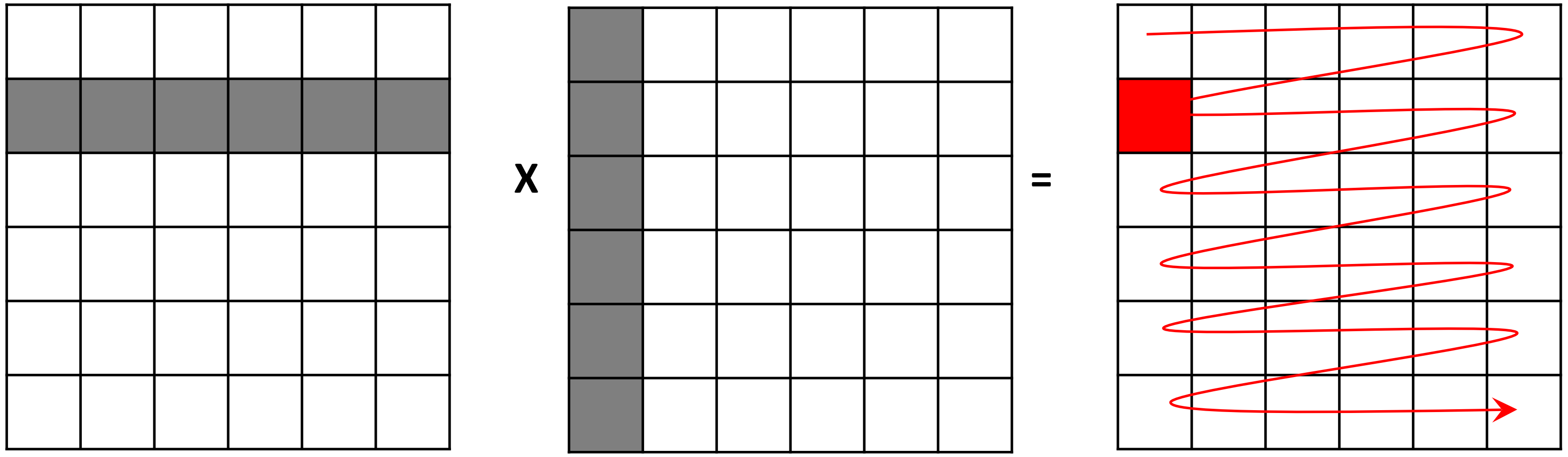
To generate one output element (red color), the 6 elements of each input matrix (gray color) will be accessed.

Loop Blocking

In the traditional matrix multiplication:

```
/* Before */
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

If N is large, the cache line that has been accessed is likely to be replaced before next access



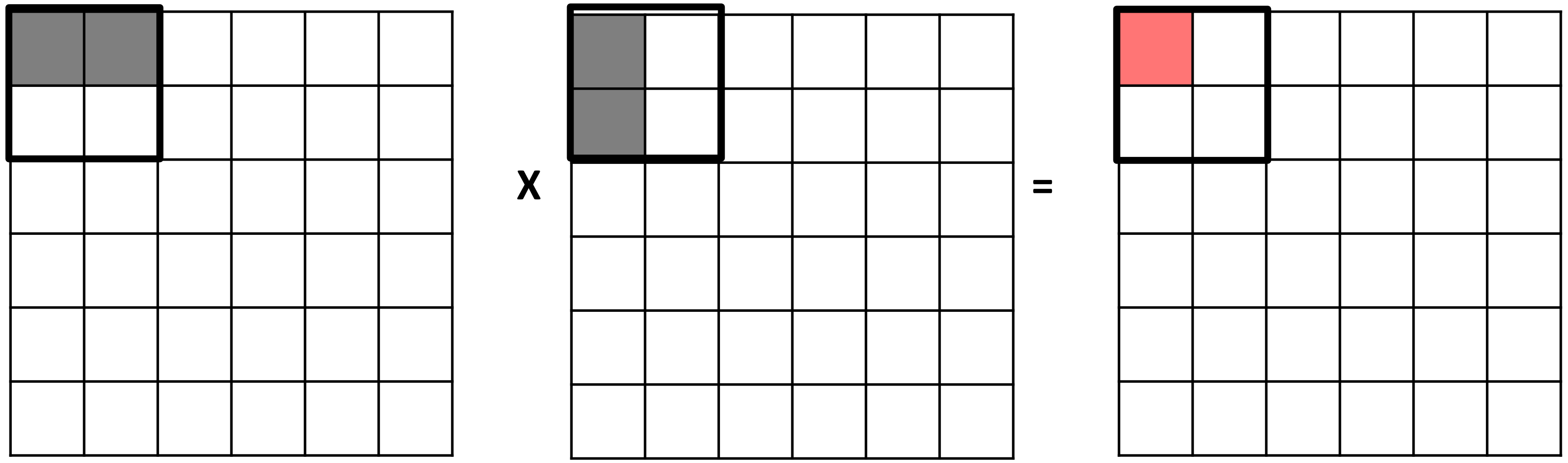
To generate one output element (red color), the 6 elements of each input matrix (gray color) will be accessed.

Loop Blocking

In the block-based matrix multiplication:

```
/* After (block size = 4) */
for (i=0; i<N; i=i+b)
 for (j=0; j<N; j=j+b)
  for (k=0; k<N; k=k+b)
   for (ii=i; ii<i+b; ii++)
        for (jj=j; jj<j+b; jj++)
         for (kk=k; kk<k+b; kk++)
                C[ii][jj] = C[ii][jj] + A[ii][kk] * B[kk][jj];
```

Matrix is computed in a smaller block unit
→ higher locality

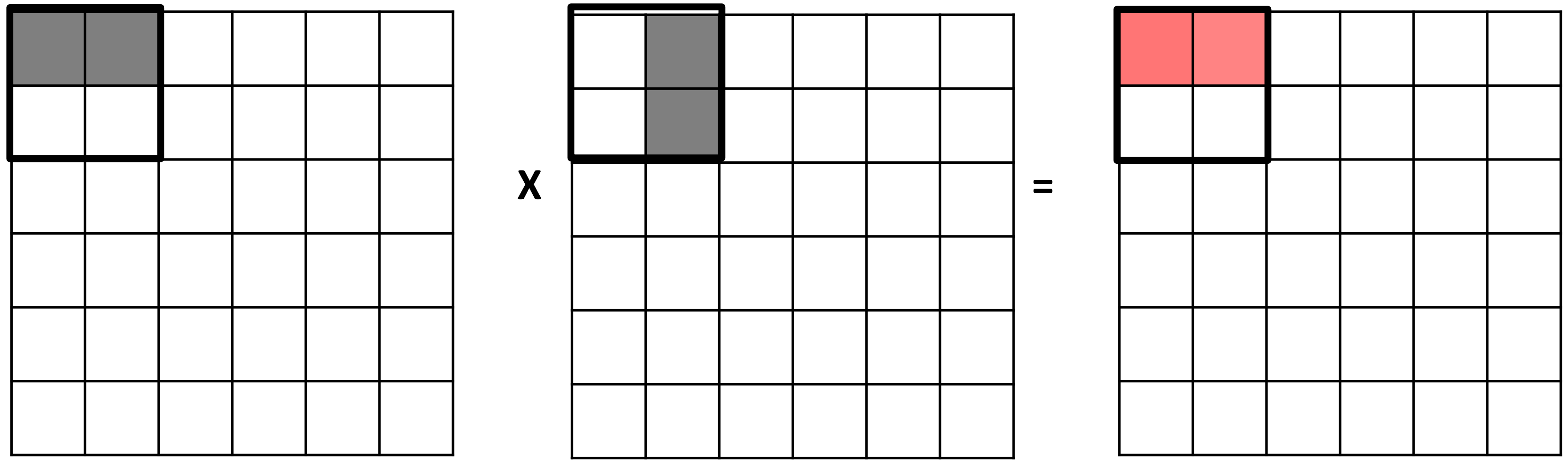


SJSU SAN JOSÉ STATE UNIVERSITY

Loop Blocking

In the block-based matrix multiplication:

```
/* After (block size = 4) */
for (i=0; i<N; i=i+b)
 for (j=0; j<N; j=j+b)
  for (k=0; k<N; k=k+b)
   for (ii=i; ii<i+b; ii++)
        for (jj=j; jj<j+b; jj++)
         for (kk=k; kk<k+b; kk++)
               C[ii][jj] = C[ii][jj] + A[ii][kk] * B[kk][jj];
```

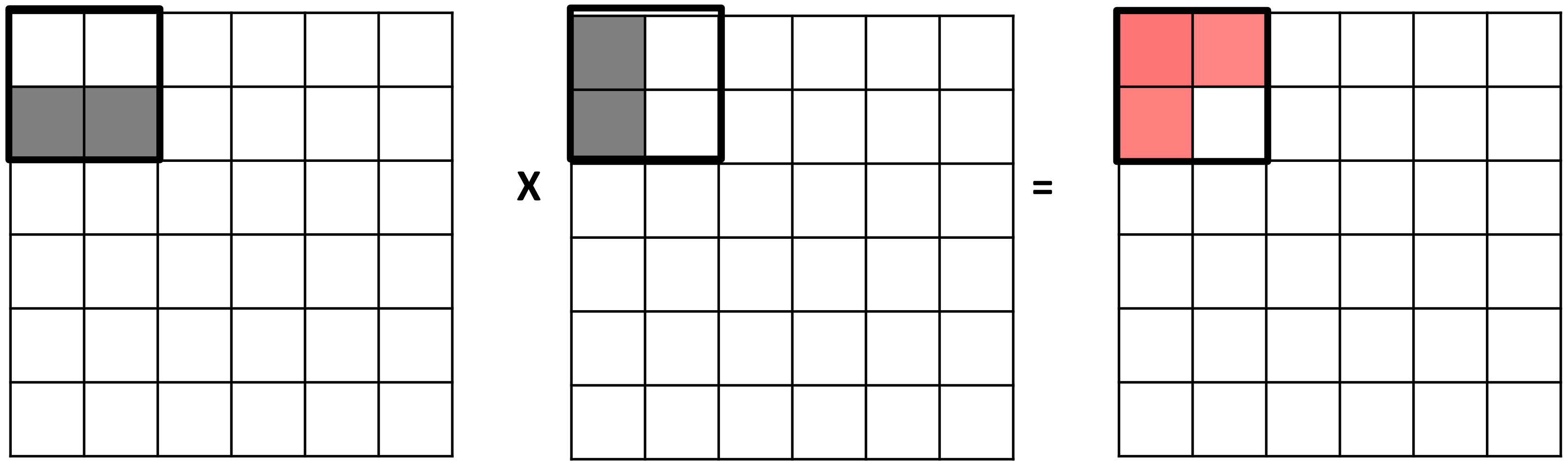


SJSU SAN JOSÉ STATE UNIVERSITY

Loop Blocking

In the block-based matrix multiplication:

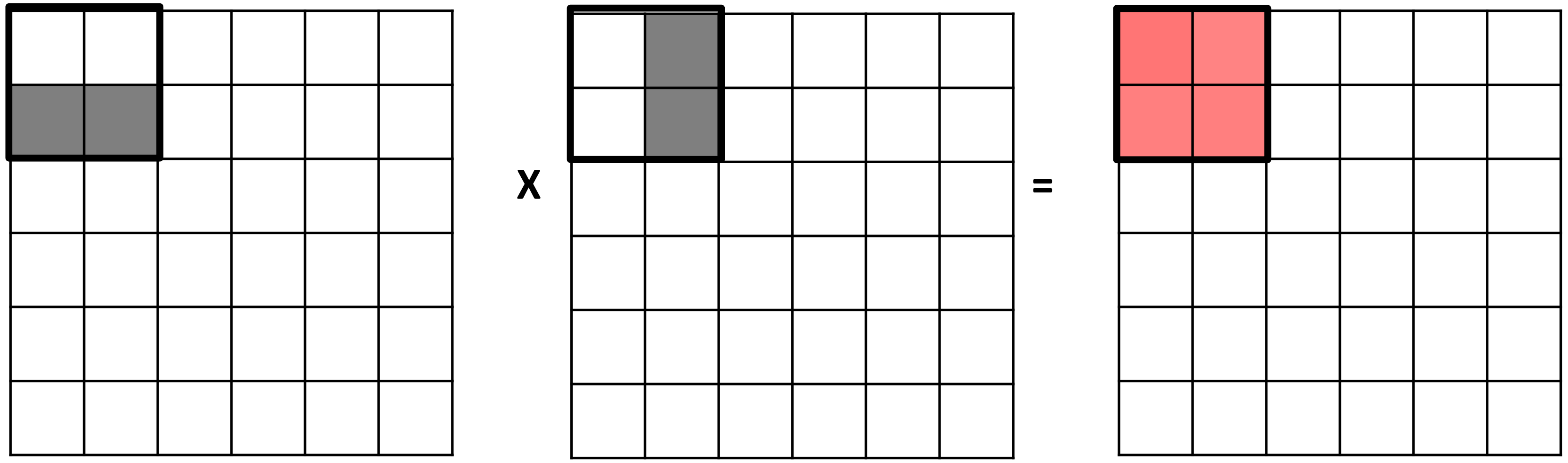
```
/* After (block size = 4) */
for (i=0; i<N; i=i+b)
 for (j=0; j<N; j=j+b)
  for (k=0; k<N; k=k+b)
   for (ii=i; ii<i+b; ii++)
        for (jj=j; jj<j+b; jj++)
         for (kk=k; kk<k+b; kk++)
                C[ii][jj] = C[ii][jj] + A[ii][kk] * B[kk][jj];
```



Loop Blocking

In the block-based matrix multiplication:

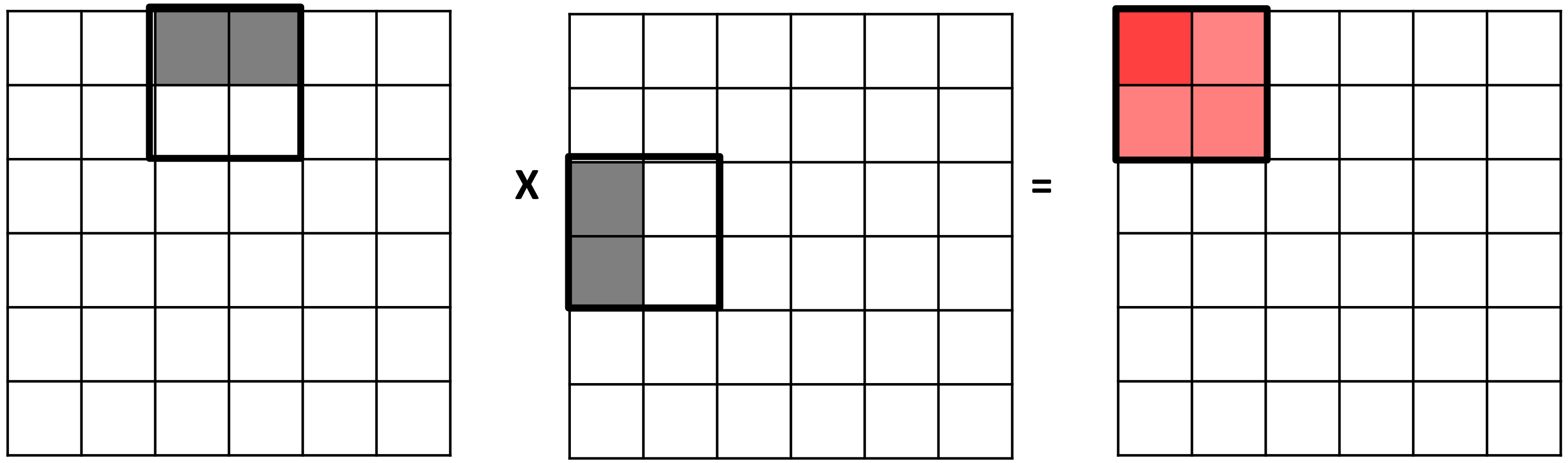
```
/* After (block size = 4) */
for (i=0; i<N; i=i+b)
 for (j=0; j<N; j=j+b)
  for (k=0; k<N; k=k+b)
   for (ii=i; ii<i+b; ii++)
        for (jj=j; jj<j+b; jj++)
         for (kk=k; kk<k+b; kk++)
               C[ii][jj] = C[ii][jj] + A[ii][kk] * B[kk][jj];
```



Loop Blocking

In the block-based matrix multiplication:

```
/* After (block size = 4) */
for (i=0; i<N; i=i+b)
 for (j=0; j<N; j=j+b)
  for (k=0; k<N; k=k+b)
   for (ii=i; ii<i+b; ii++)
        for (jj=j; jj<j+b; jj++)
         for (kk=k; kk<k+b; kk++)
               C[ii][jj] = C[ii][jj] + A[ii][kk] * B[kk][jj];
```

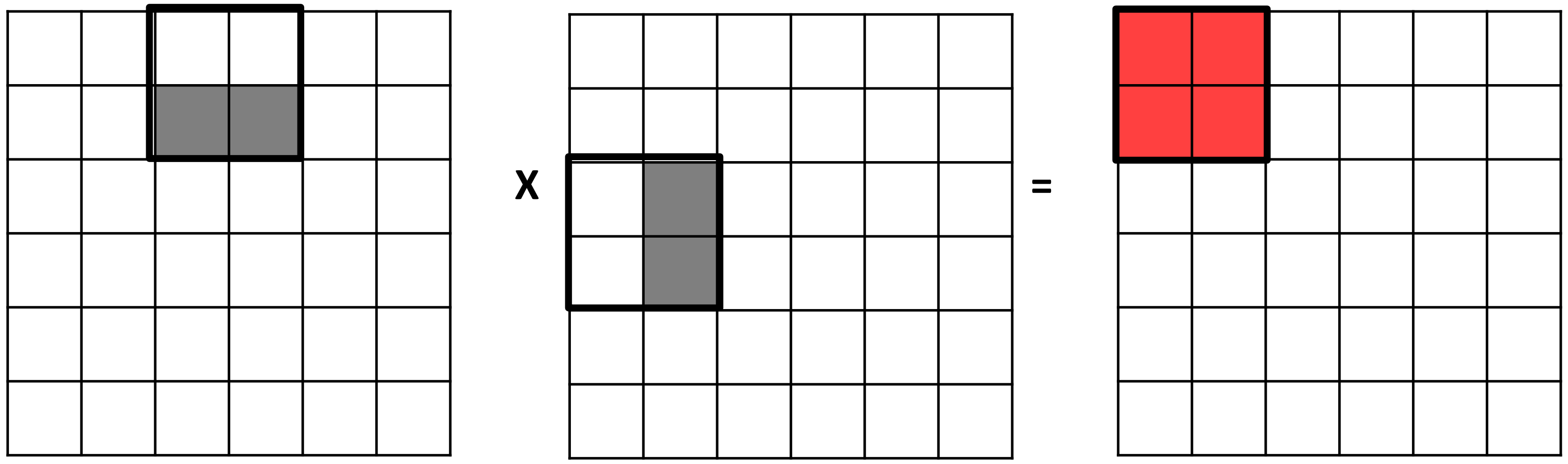


SJSU SAN JOSÉ STATE UNIVERSITY

Loop Blocking

In the block-based matrix multiplication:

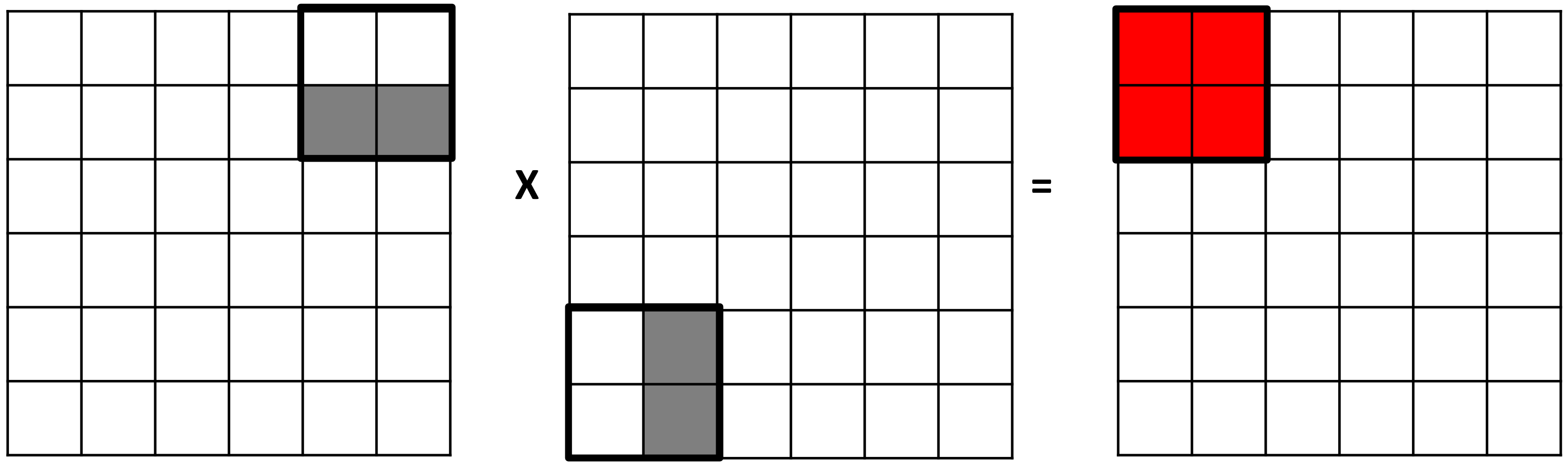
```
/* After (block size = 4) */
for (i=0; i<N; i=i+b)
 for (j=0; j<N; j=j+b)
  for (k=0; k<N; k=k+b)
   for (ii=i; ii<i+b; ii++)
        for (jj=j; jj<j+b; jj++)
         for (kk=k; kk<k+b; kk++)
               C[ii][jj] = C[ii][jj] + A[ii][kk] * B[kk][jj];
```



Loop Blocking

In the block-based matrix multiplication:

```
/* After (block size = 4) */
for (i=0; i<N; i=i+b)
 for (j=0; j<N; j=j+b)
  for (k=0; k<N; k=k+b)
   for (ii=i; ii<i+b; ii++)
        for (jj=j; jj<j+b; jj++)
         for (kk=k; kk<k+b; kk++)
                C[ii][jj] = C[ii][jj] + A[ii][kk] * B[kk][jj];
```

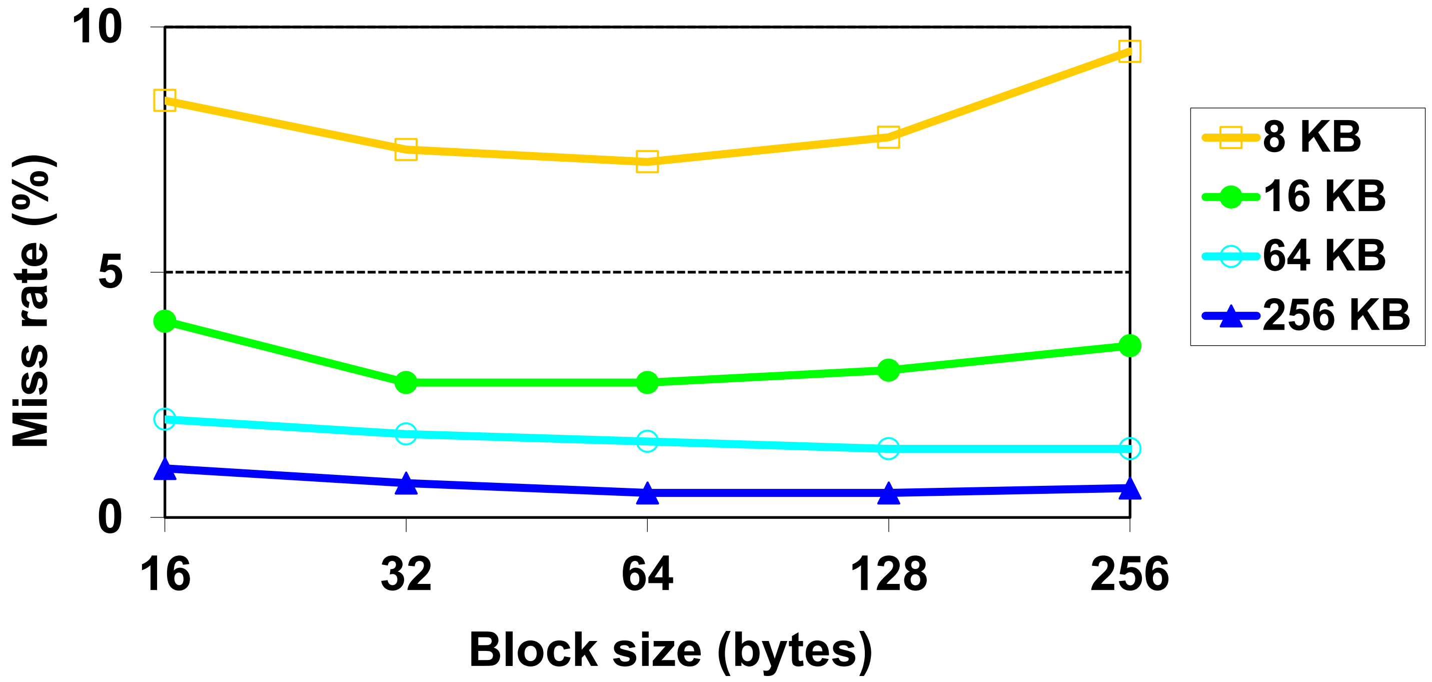


SJSU SAN JOSÉ STATE UNIVERSITY

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

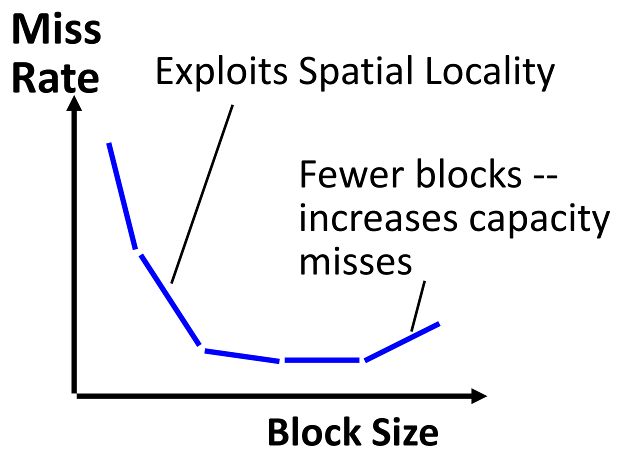
- **Increase Cache Size**
  - Reduce miss for: capacity miss, conflict miss
  - But has many limitations
- **Increase Associativity**
  - Reduce miss for: conflict miss
  - But may increase access latency
- **Increase Cache Block Size**
  - Reduce miss for: compulsory
  - But may increase miss penalty (more data will be evicted and fetched)
  - Very large blocks could increase miss rate

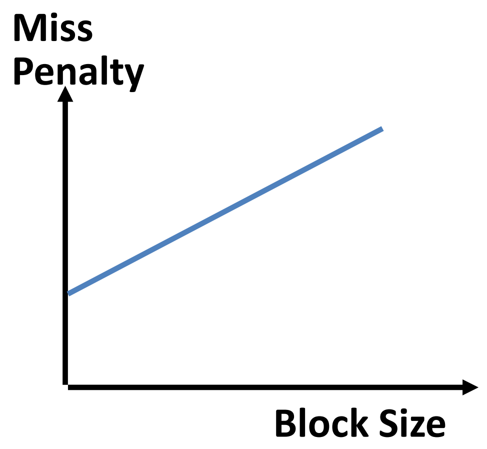
Miss Rate vs Block Size vs Cache Size

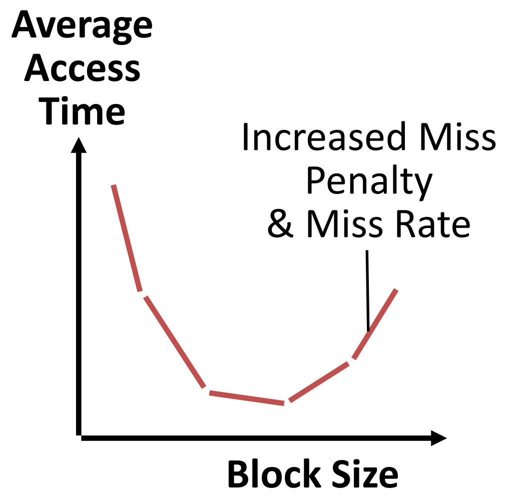


- Cache size – the larger the better
- Block size – tradeoffs

Block Size Tradeoff

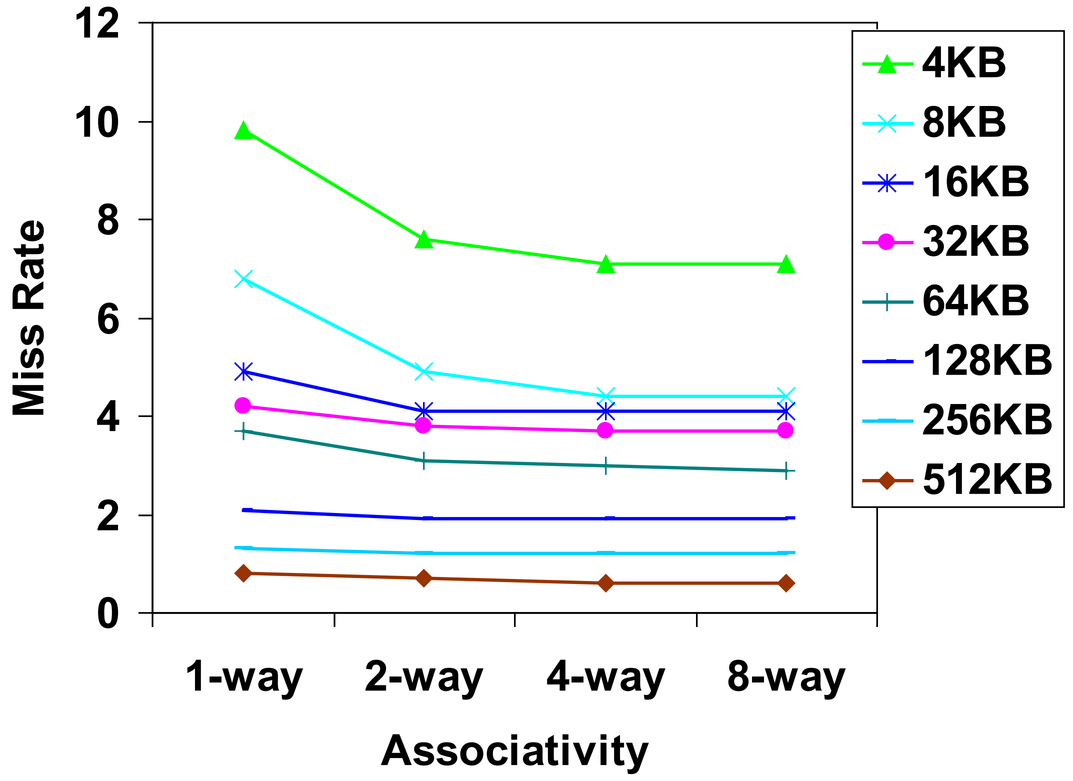






Average Memory Access Time = Hit Time + Miss Penalty x Miss Rate

Benefits of Set Associative Caches



Largest gains when going from direct mapped to 2-way (20%+ reduction in miss rate)

SJSU SAN JOSÉ STATE UNIVERSITY

Data from Hennessy & Patterson, *Computer Architecture*

Reduce Miss Rate (3): Policies

- **Use appropriate policies**
  - LRU + write-back works for most applications
  - Many GPUs use write-hit write-evict + write-miss no-write-allocate
  - Can even dynamically change policy according to profiling
- **Use more advanced policies**
  - E.g., ML based

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**

Split vs. Unified Caches

- **Split L1I$ and L1D$: to optimize t_{hit}**
  - Low capacity/associativity/block size (reduce t_{hit})
  - Minimize structural hazards
  - Can optimize L1I$ for wide output and no writes

- **Unified L2: to optimize hit rate**
  - t_{hit} is less important due to less frequent accesses & long delay already
  - High capacity/associativity/block size (reduce $\%_{miss}$)
    - Unused instr capacity can be used for data (fewer capacity misses)
    - Instr / data conflicts (small to no increase for conflict misses in a large cache)
  - Instr / data structural hazards are rare (would take a simultaneous L1I$ and L1D$ miss)

Reduce Miss Rate (5): Prefetching

- **Hardware prefetching**
  - Fetch blocks into the cache proactively (speculatively)
  - Key is to anticipate the upcoming miss addresses accurately
  - Relies on having unused memory bandwidth available
- **A simple case is to use next block prefetching**
  - Miss on address X → anticipate next reference miss on X + block-size
  - Works well for instr's (sequential execution) and for arrays of data
- **Need to initiate prefetches sufficiently in advance**
- **If prefetched instr/data is not used, the cache is polluted with unnecessary data (possibly evicting useful data)**

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 m… 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 $\text{CPI}_{\text{stall}}$, the cache miss penalty is measured in *processor* clock cycles needed to handle a miss
  - The lower the $\text{CPI}_{\text{ideal}}$, the higher the impact of stalls

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

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

  $\text{CPI}_{\text{stalls}} = 2 + 3.44 = 5.44$

- **What if the $\text{CPI}_{\text{ideal}}$ is reduced to 1? Or the processor clock rate is doubled (doubling the miss penalty)?**

SJSU SAN JOSÉ STATE UNIVERSITY

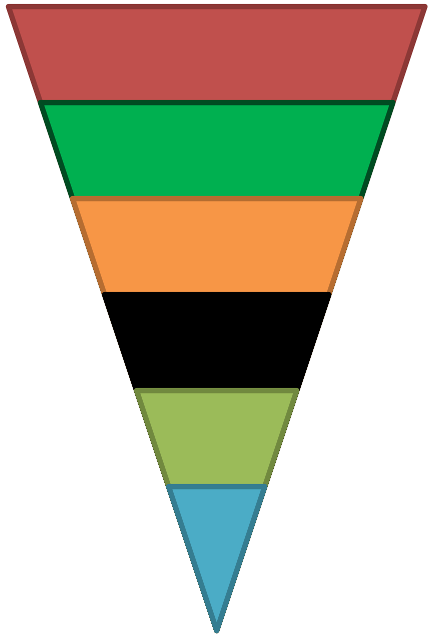
Other than Miss Rates

- **Cache bypassing (e.g., L1D)**
  - Reduce latency if L1 misses
- **Sector cache**
  - Fetch only a portion of the cache line
  - Not the same as reducing cache line size
- **Value prediction**
  - Using history value to predict future values
  - Roll back if predict wrong
  - Or accept a certain quality loss if the application is error tolerant

DRAM Subsystem Organization

Dram: Dynamic RAM

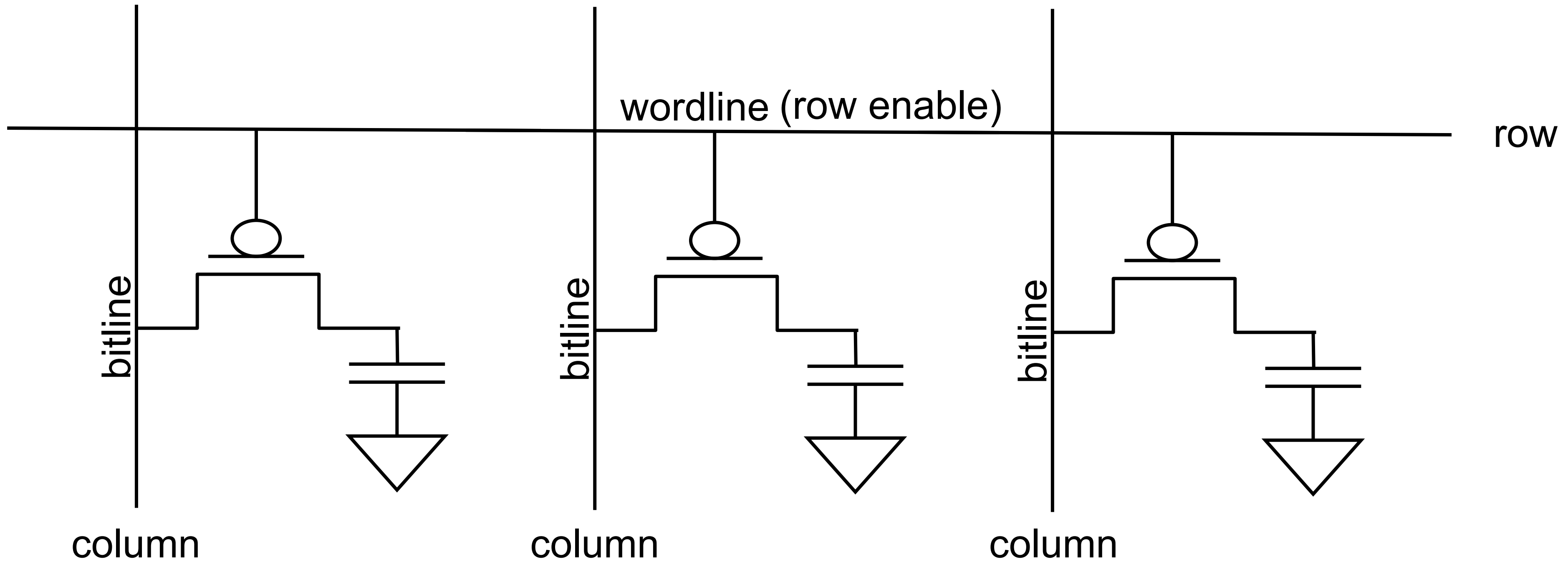
DRAM Organization:



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

Connected to an on-die memory controller

DRAM Cells



- **A DRAM cell consists of a capacitor and an access transistor**
- **It stores data in terms of charge in the capacitor**
- **Cheaper and larger than SRAM**
  - A DRAM chip consists of (10s of 1000s of) rows of such cells

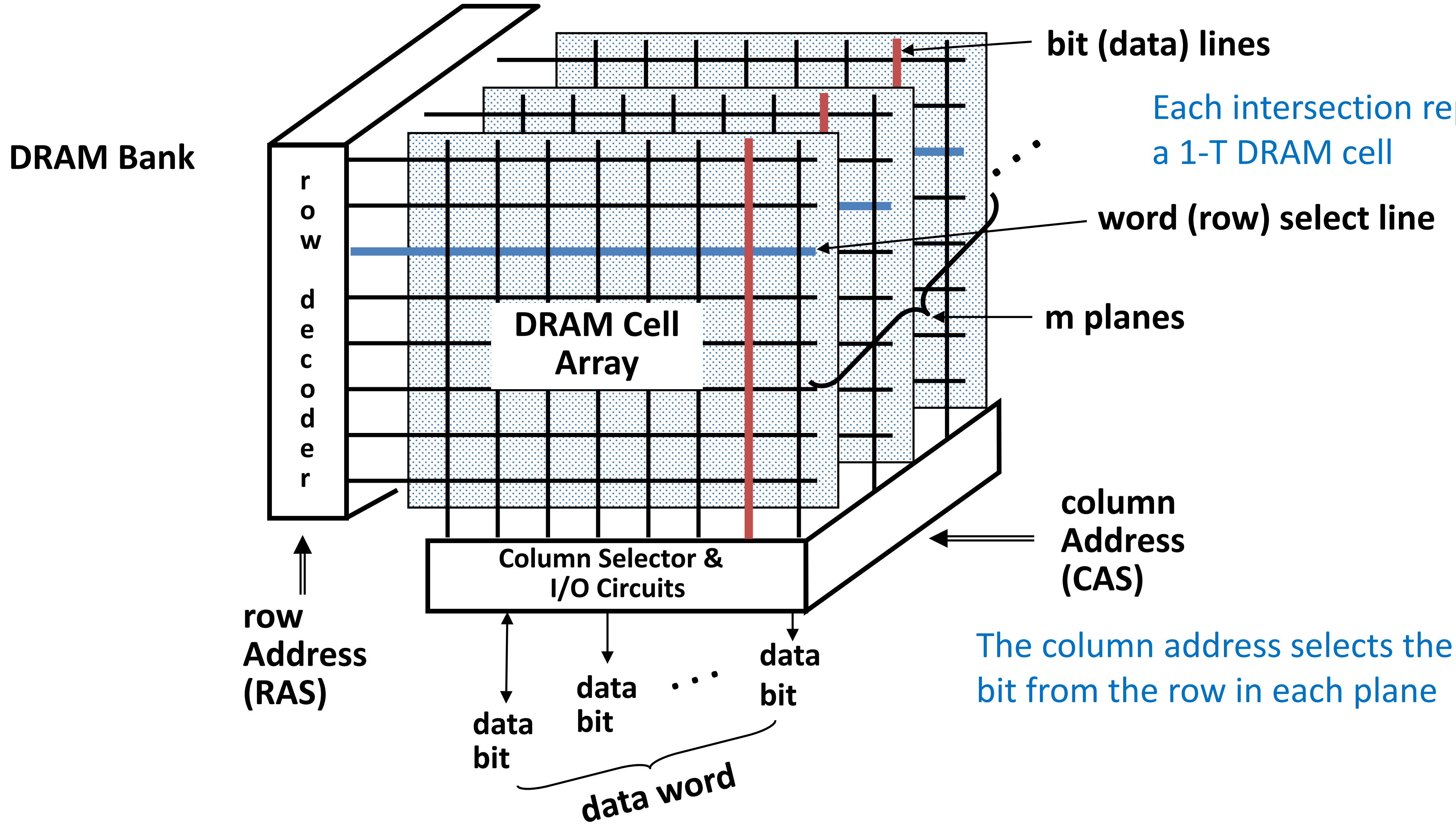
DRAM Refresh

- **DRAM capacitor charge leaks over time**

- **The memory controller needs to refresh each row periodically to restore charge**
  - Dynamic (i.e., never in a stable state)
  - Activate each row every N ms (e.g., typical N = 64 ms)

- **Downsides of refresh**
  - Energy consumption: Each refresh consumes energy
  - Performance degradation: DRAM rank/bank unavailable while refreshed
  - QoS/predictability impact: (Long) pause times during refresh
  - Refresh rate limits DRAM capacity scaling

DRAM Design 1: The Classical



Each intersection represents a 1-T DRAM cell

The column address selects the requested bit from the row in each plane

DRAM Performance Metrics

- **DRAM addresses are divided into 2 halves (row and column)**
  - *RAS* or *Row Access Strobe* that triggers the row decoder
  - *CAS* or *Column Access Strobe* that triggers the column selector

- **Latency: Time to access one word**
  - *Access Time*: time between request and when word is read or written
    - read access and write access times can be different
  - *Cycle Time*: time between successive (read or write) requests
  - Usually, cycle time > access time

- **Bandwidth: How much data can be supplied per unit time**
  - Width of the data channel * channel usage frequency

DRAM Performance Metrics

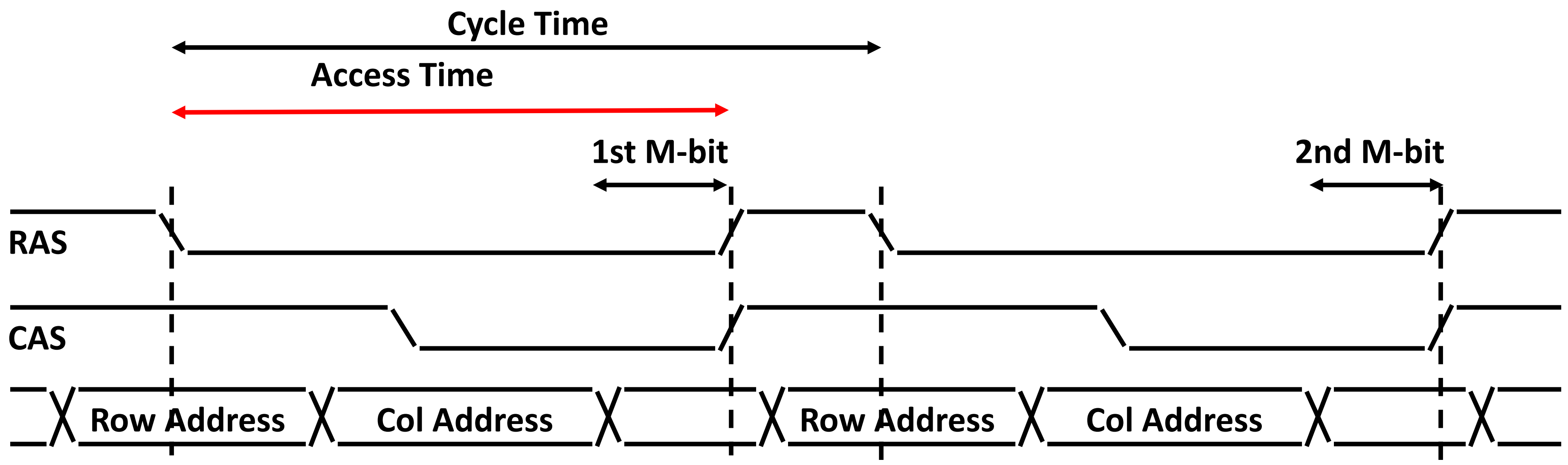
- **DRAM addresses are divided into 2 halves (row and column)**
  - *RAS* or *Row Access Strobe* that triggers the row decoder
  - *CAS* or *Column Access Strobe* that triggers the column selector

- **Latency: Time to access one word**
  - *Access Time*: time between request and when word is read or written
    - read access and write access times can be different
  - *Cycle Time*: time between successive (read or write) requests
  - Usually, cycle time > access time

- **Bandwidth: How much data can be supplied per unit time**
  - Width of the data channel * channel usage frequency

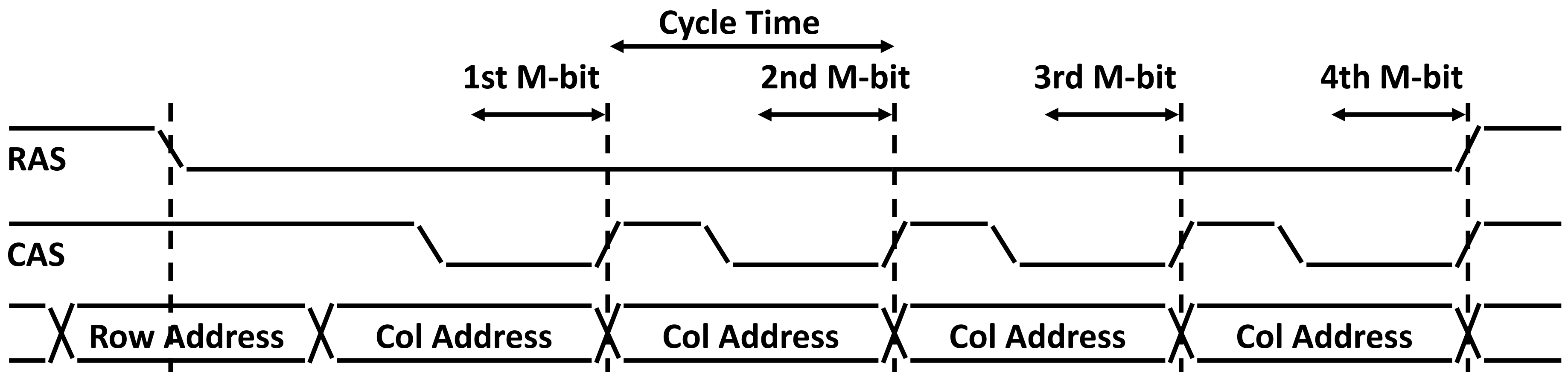
DRAM Design 1: The Classical

- **DRAM Organization:**
  - N rows x N column x M-bit (planes)
  - Reads or Writes M-bit at a time
  - Each M-bit access requires a RAS / CAS cycle



DRAM Design 1: The Classical

- **Page Mode: A row is kept "open" by keeping the RAS asserted**
  - Pulse CAS to access other M-bit blocks on **that** row
  - Successive reads or writes within the row are faster since don't have to precharge and (re)access that row



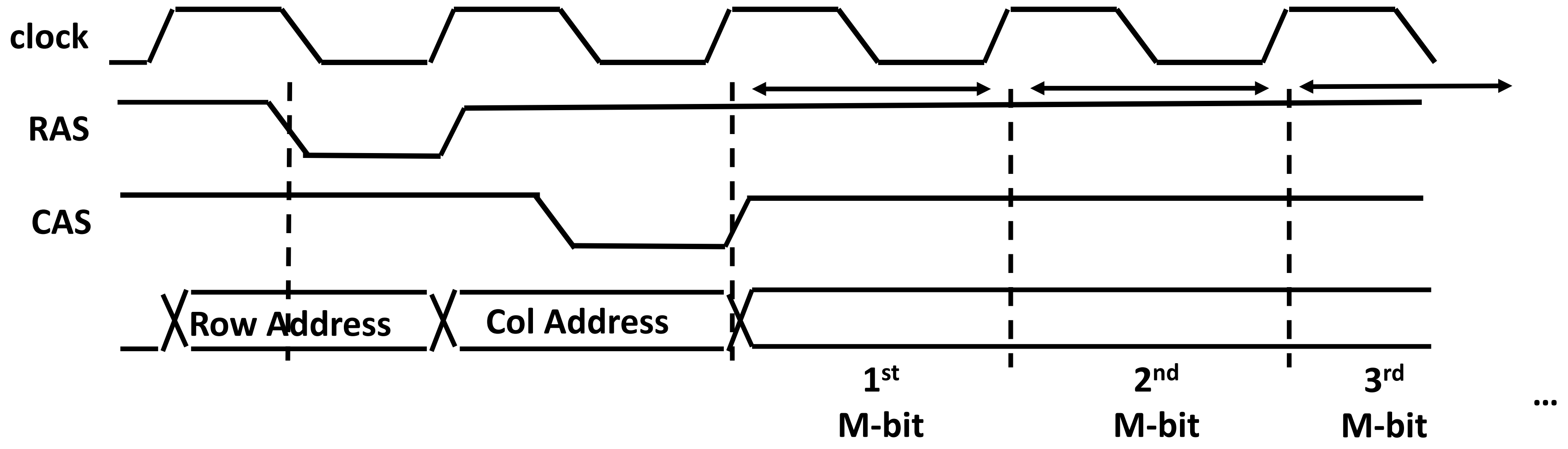
DRAM Design 2: Synchronous DRAMs

- **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**
  - Specify the starting (row+column) address and the burst length (burst must be in the same row). The row is accessed from the DRAM and loaded into a row buffer (SRAM).
  - Data words in the burst are then accessed from that SRAM under control of a clock signal.

Synchronous DRAM (SDRAM) Operation

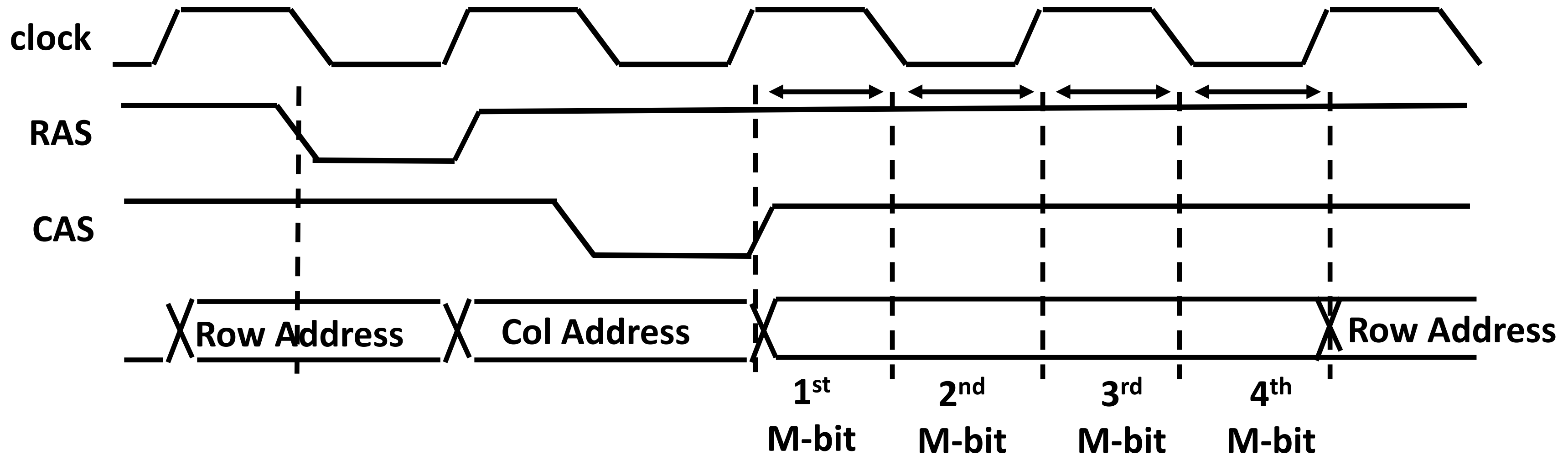
- **After RAS loads a row into the SRAM cache**
  - Input CAS as the starting "burst" address along with a burst length to read a burst of data from a series of sequential addresses within that row on the clock edge



SJSU SAN JOSÉ STATE UNIVERSITY

DDR (Double Data Rate) SDRAMs

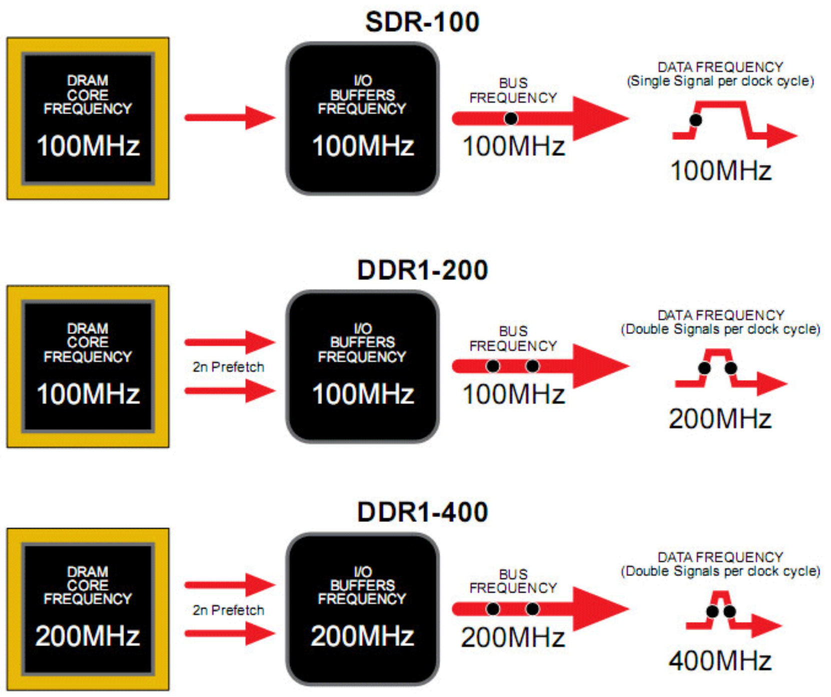
- **Transfers burst data on both the rising and falling edge of the clock (so twice fast)**
  - 2n core prefectch



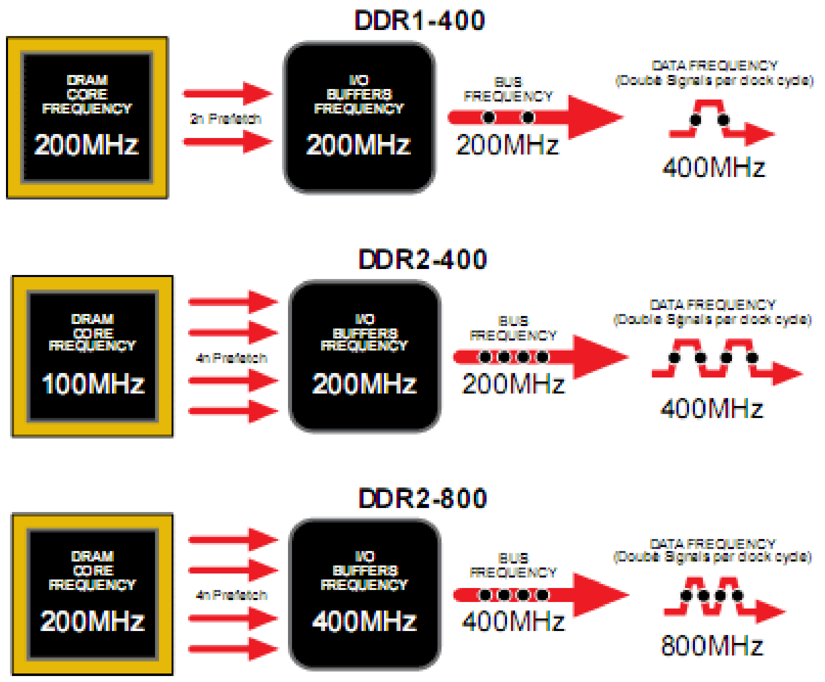
http://en.wikipedia.org/wiki/DDR_SDRAM

DDR (Double Data Rate) SDRAMs

- **DDR1 VS DDR1+:**



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



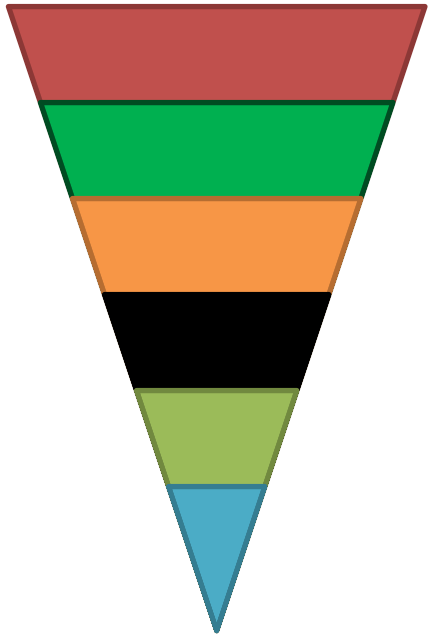
▲ Simplified Comparison between DDR1-400, DDR2-400 and DDR2-800
Illustration: Ryan J. Leng

- **DDR2 vs. QDR**

DRAM Subsystem Organization

Dram: Dynamic RAM

DRAM Organization:



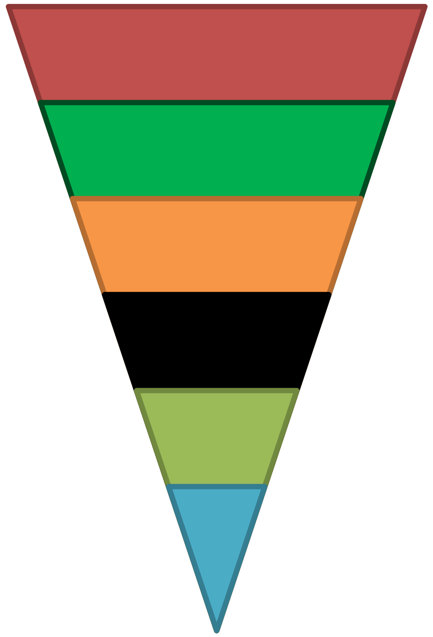
- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column

- **Chip:**
  - Consists of multiple banks (2-16 in Synchronous DRAM)
    - Banks works in parallel to overlap delay
  - Banks share command/address/data buses
  - The chip itself has a narrow interface (4-16 bits per read)

DRAM Subsystem Organization

Dram: Dynamic RAM

DRAM Organization:



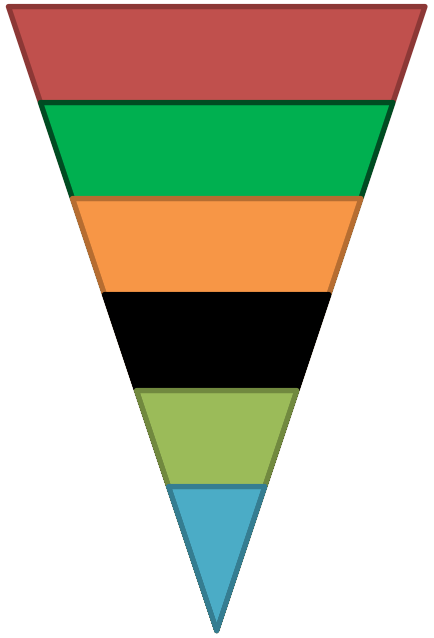
- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column

- **Rank:**
  - Multiple chips operated together to form a wide interface
  - All chips comprising a rank are controlled at the same time
    - Respond to a single command
    - Share address and command buses, but provide different data
  - E.g., Using 8 chips with 8-bit interface to form a 64-bit wide Rank

DRAM Subsystem Organization

Dram: Dynamic RAM

DRAM Organization:

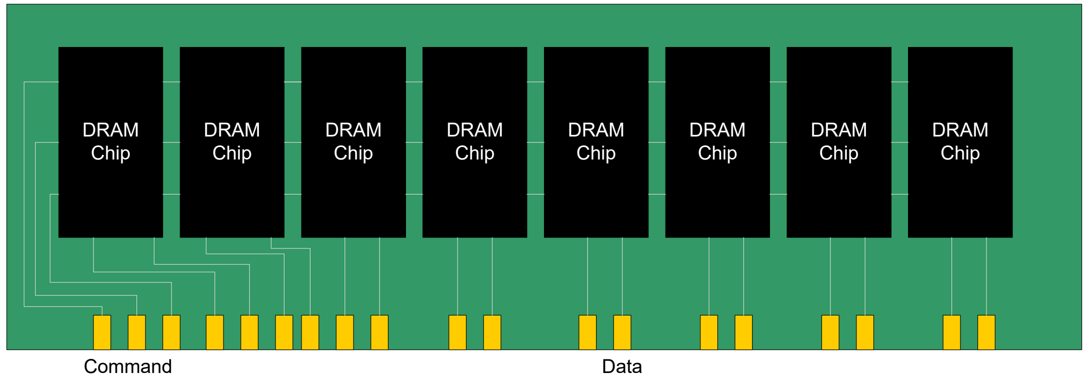


- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column

- **DIMM (dual inline memory module):**
  - A DRAM module consists of one or more ranks
  - This is what you plug into your motherboard

SJSU SAN JOSÉ STATE UNIVERSITY

DIMM (Dual Inline Memory Module)

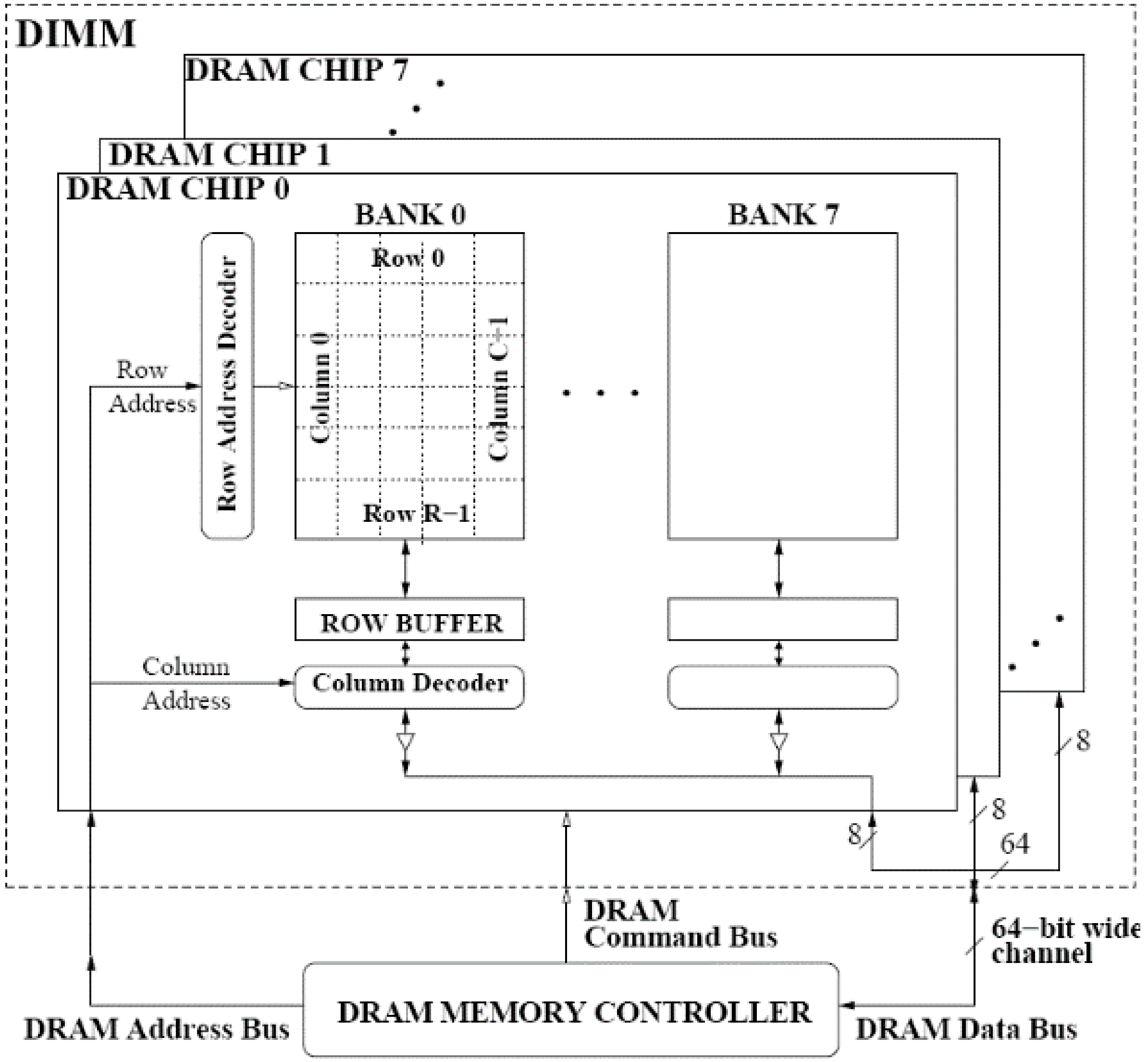


- **Contains DRAM chips each have data widths of x4 or x8**
  - Can be on both sides

- **Can have more than one rank**
  - Increase storage capacity
  - Only one rank accessible at a time

- **SIMM vs DIMM**
  - DIMM has separate contacts on each side of the board to provides twice as much data rate.

A 64-bit Wide DIMM (One Rank)



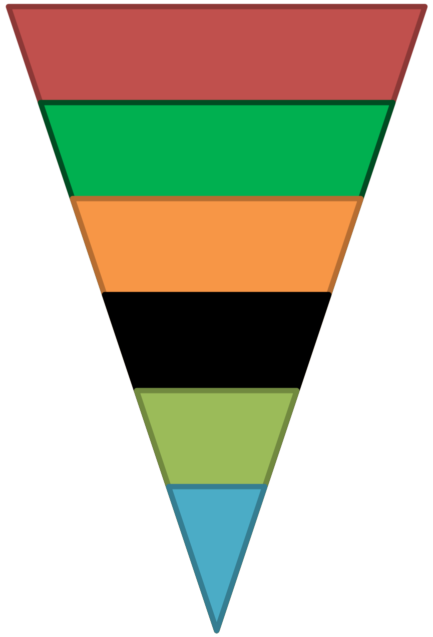
- **Advantages:**
  - Acts like a high-capacity DRAM chip with a wide interface
  - Flexibility: memory controller does not need to deal with individual chips
- **Disadvantages:**
  - Granularity: Accesses cannot be smaller than the interface width

SJSU SAN JOSÉ STATE UNIVERSITY

DRAM Subsystem Organization

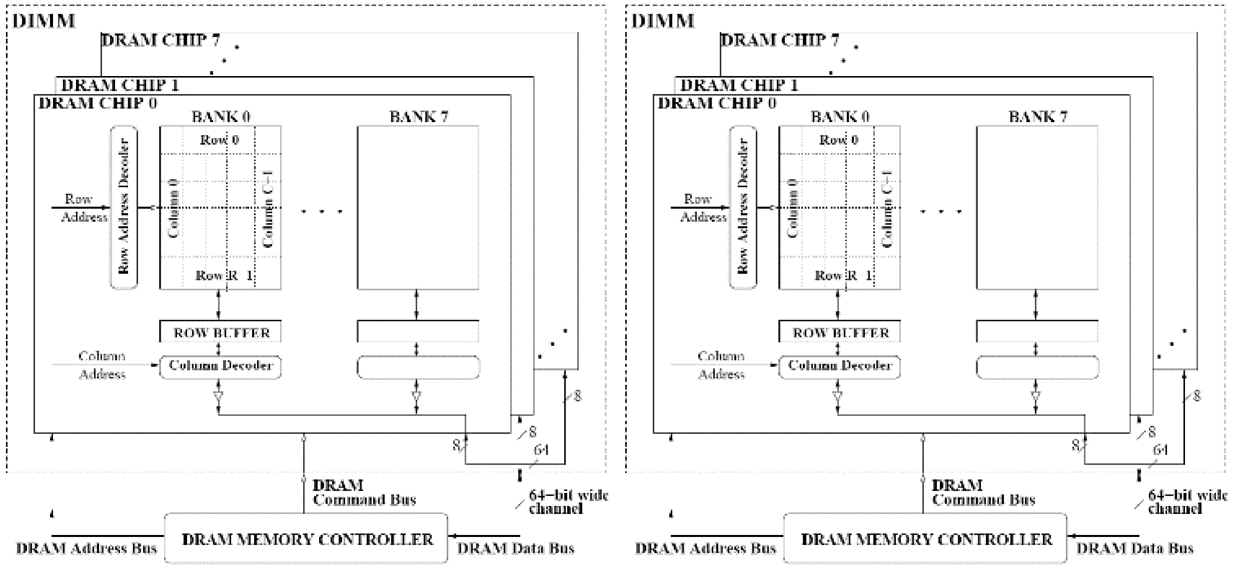
Dram: Dynamic RAM

DRAM Organization:



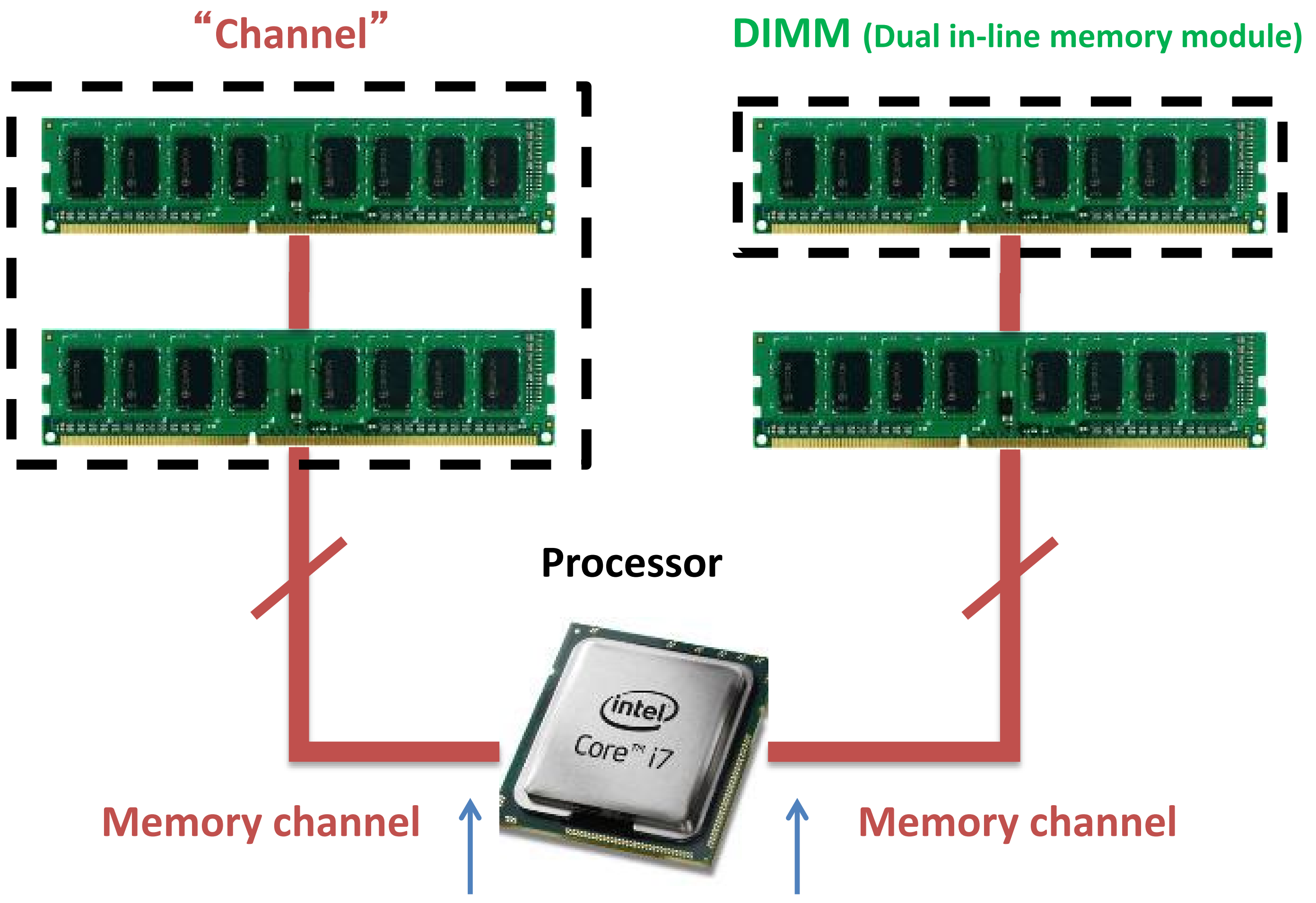
- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column

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



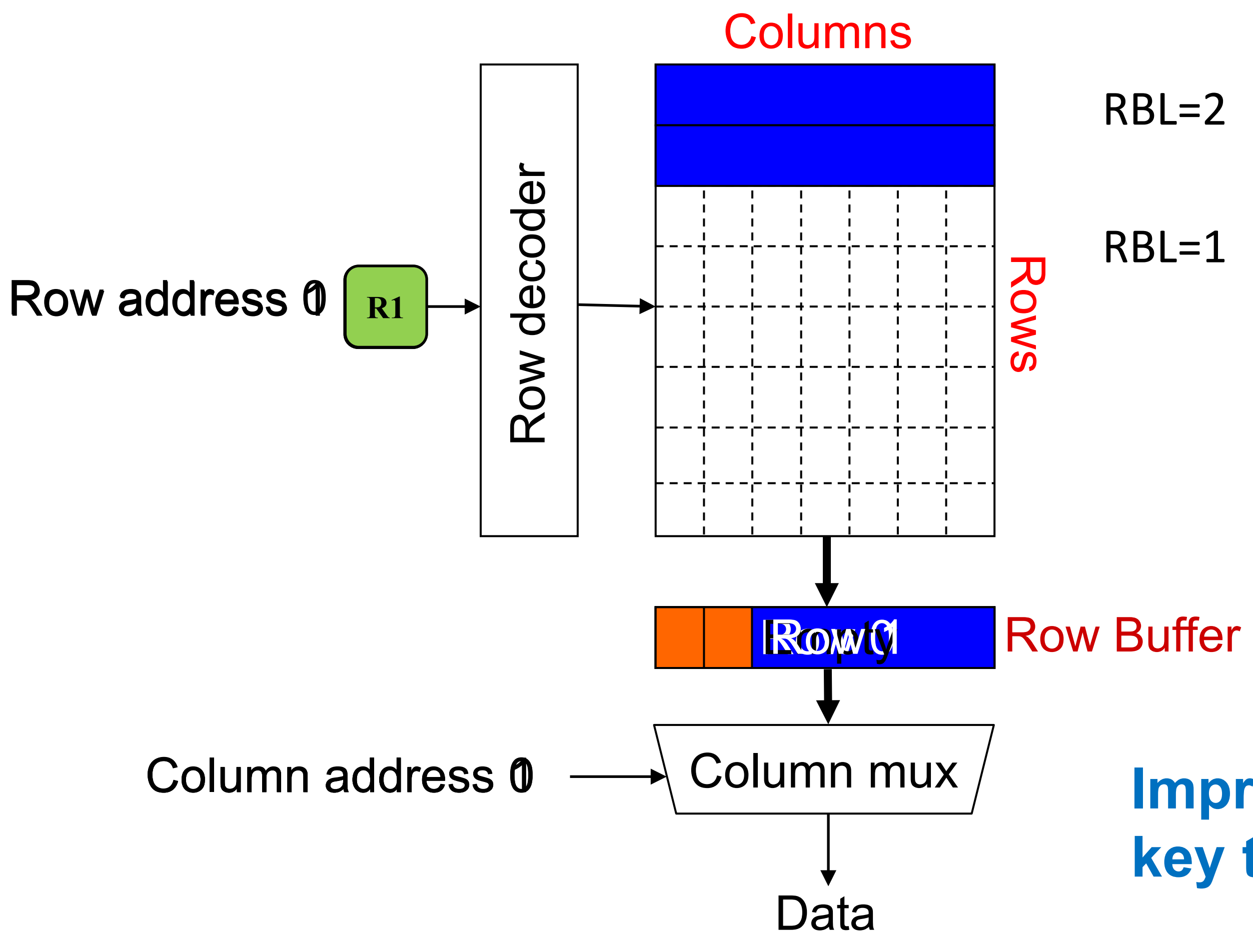
SJSU SAN JOSÉ STATE UNIVERSITY

The DRAM subsystem



Each channel is connected to its own on-die memory controller

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

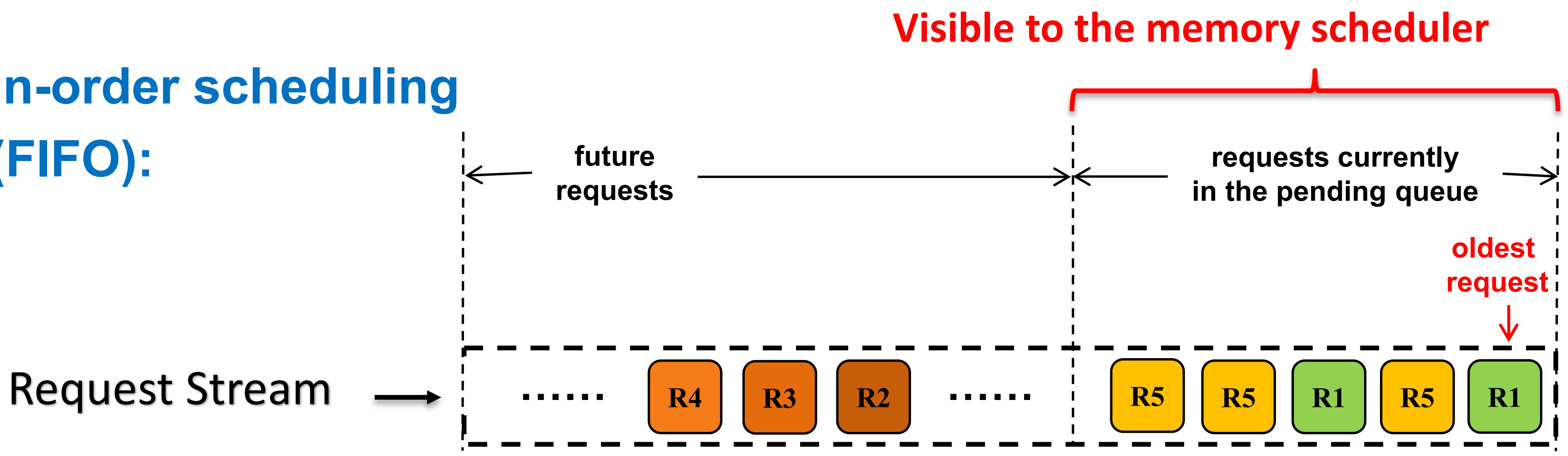
HIT CONFLICT !

Improving Row Buffer Locality (RBL) is the key to reduce DRAM energy consumption

Credit: Onur Mutlu.

RBL & Memory Scheduling Schemes

In-order scheduling (FIFO):

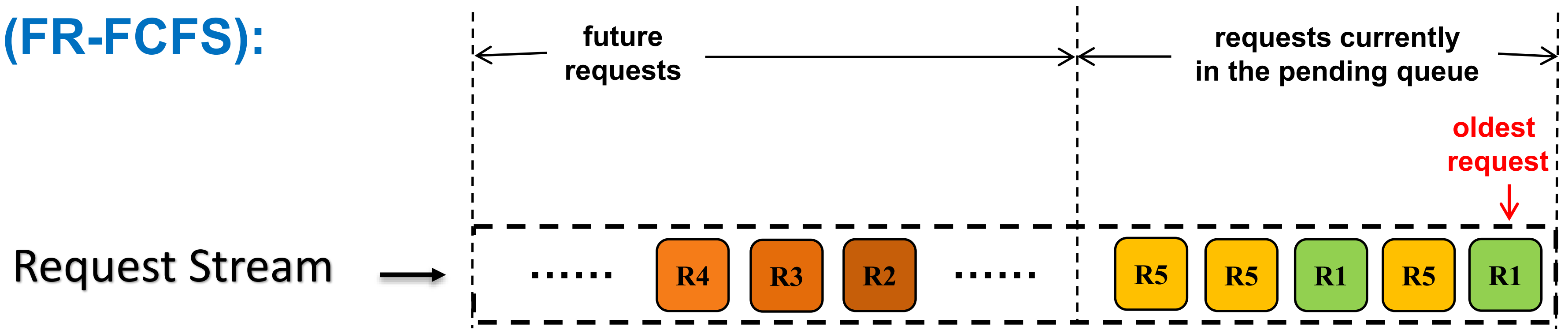


Visible to the memory scheduler
future requests
requests currently in the pending queue
oldest request
Request Stream
……
R4
R3
R2
……
R5
R5
R1
R5
R1

Activation Counter:

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

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



Activation Counter:

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

DRAM Milestones

	DRAM	Page DRAM	Page DRAM	Page DRAM	SDRAM	DDR SDRAM
Module Width	16b	16b	32b	64b	64b	64b
Year	1980	1983	1986	1993	1997	2000
Mb/chip	0.06	0.25	1	16	64	256
Die size (mm^2)	35	45	70	130	170	204
Pins/chip	16	16	18	20	54	66
BWidth (MB/s)	13	40	160	267	640	1600
Latency (nsec)	225	170	125	75	62	52

- **In the time that the memory to processor bandwidth has more than doubled the memory latency has improved by a factor of only 1.2 to 1.4**
- **To deliver such high bandwidth, the internal DRAM has to be organized as interleaved memory banks**

Memory (DRAM) + Caches

- **It is important to match the cache characteristics**
  - Caches want information provided to them one block at a time (and a block is usually more than one word)

- **Memory design considerations:**
  - With the main memory characteristics
    - Use DRAMs that support fast multiple word accesses, preferably ones that **match the block size** of the cache
  - With the memory-bus characteristics
    - Make sure the memory-bus can support the DRAM access rates and patterns
    - With the goal of increasing the Memory-Bus-to-Cache bandwidth

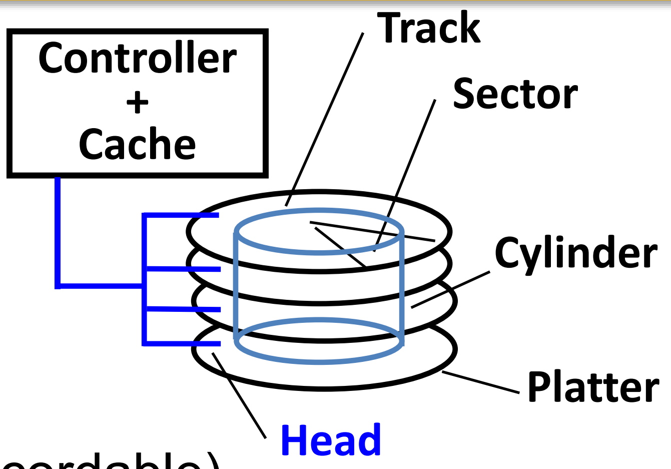
Review: DRAM vs. SRAM

- **DRAM**
  - Slower access (capacitor)
  - Higher density (1T 1C cell)
  - Lower cost
  - Requires refresh (power, performance, circuitry)
  - Manufacturing requires putting capacitor and logic together

- **SRAM**
  - Faster access (no capacitor)
  - Lower density (6T cell)
  - Higher cost
  - No need for refresh
  - Manufacturing compatible with logic process (no capacitor)

Magnetic Disk

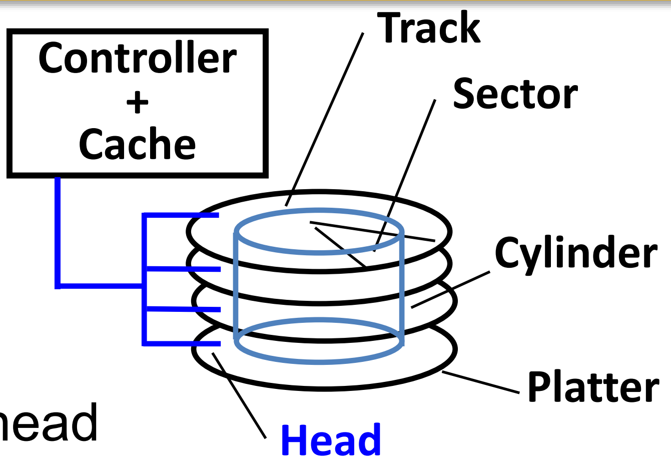
- **Purpose:** Long term, nonvolatile storage
  – Lowest level in the memory hierarchy: large, cheap, slow



- **General structure**
  – 1 to 4 rotating platter coated with a magnetic surface (2 sides recordable)
    - Rotational speeds of 5,400 to 15,000 RPM
  – Moveable read/write head to access the information for each platter
  – 10,000 to 50,000 tracks per surface
    - Cylinder - all the tracks under the head at a given point on all surfaces
  – 100 to 500 sectors per track
    - The smallest unit that can be read/written (typically 512B)
    - Outer tracks hold more sectors than the inner tracks

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

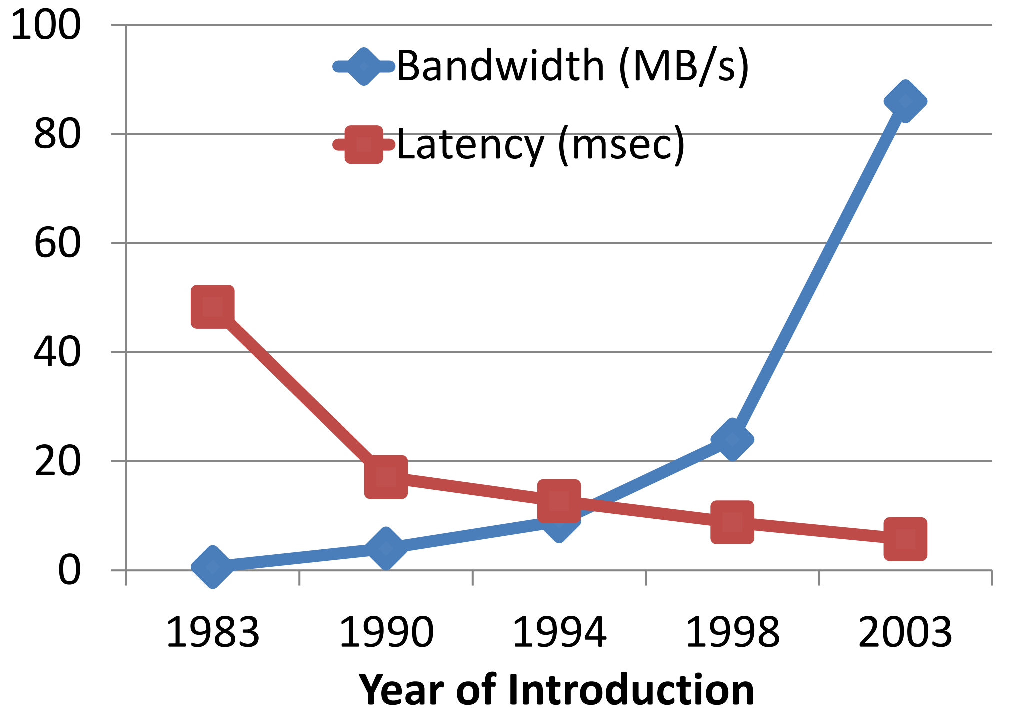
= 4.0 ms + 0.5/(15,000RPM/(60sec/min)) + 0.5KB/(100MB/sec) + 0.2 ms

= 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

Disk Latency & Bandwidth Improvement



- Disk latency = average seek time + rotational latency.
- Disk bandwidth is the peak transfer speed of formatted data from the media (not from the cache).

SJSU SAN JOSÉ STATE UNIVERSITY

Flash Storage

- **Flash memory is the first credible challenger to disks. It is semiconductor memory that is nonvolatile like disks but has latency 100 to 1000 times lower and is smaller, more power efficient, and more shock resistant.**
  - Flash memory bits wear out. But with wear leveling it is unlikely that the write limits of the flash will be exceeded.
  - Example:

Storage	Write throughput	Read throughput	Latency
SSD	2,500 MB/S	3,500 MB/S	0.025 ms
DISK	250 MB/S	250 MB/S	5 ms

Flash Storage Characteristics

- **Flash memory is organized into blocks of pages**
  - page size 512B to 4KB, block size 32 to 128 pages (typical)
  - “read” is page read
  - “write” is block erase (~ 1 ms) followed by page write (and additional copying of other pages in that block)

- **Typically a type of Electrically Erasable PROGRAMMABLE Read-Only Memory (EEPROM)**

- **Comes in two flavors: NOR Flash and NAND Flash**
  - NOR Flash is randomly addressable (Minimum access size 512 bytes)
  - NAND Flash is less expensive (greater storage density) and is not randomly addressable (minimum access size 2048 bytes); so more popular

Dependability, Reliability, Availability

- **Reliability – measured by the mean time to failure (MTTF). Service interruption is measured by mean time to repair (MTTR)**

- **Availability – a measure of service accomplishment**

  Availability = MTTF / (MTTF + MTTR)

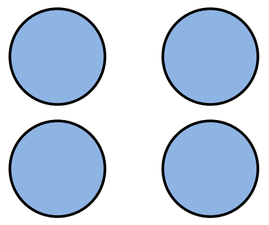
- **To increase MTTF, either improve the quality of components or design the system to continue operating in the presence of faulty components**
  1. Fault avoidance: preventing fault occurrence by construction
  2. Fault tolerance: using redundancy to correct or bypass faulty components (hardware)

RAID: Redundant Array of Independent Disks

- **Arrays of independent physical disks working together as one logical disk**
  - Increase potential throughput by having many disk drives
    - Data can be spread across multiple disk
    - Multiple disk accesses can be made simultaneously for higher throughput

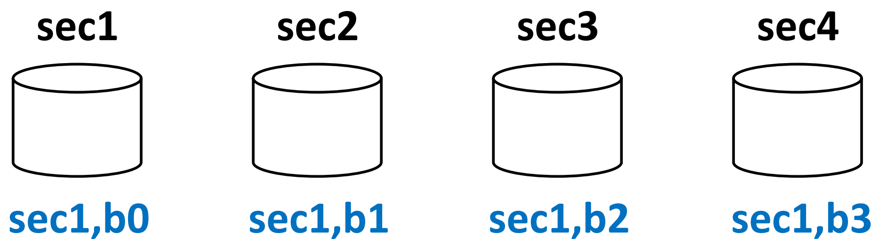
- **Reliability for the array is lower than for a single disk**

- **Availability can be improved by adding redundant disks**
  - Lost information can be reconstructed from redundant information
  - MTTR: mean time to repair is in the order of hours
  - MTTF: mean time to failure of disks is tens of years


RAID Disk Array

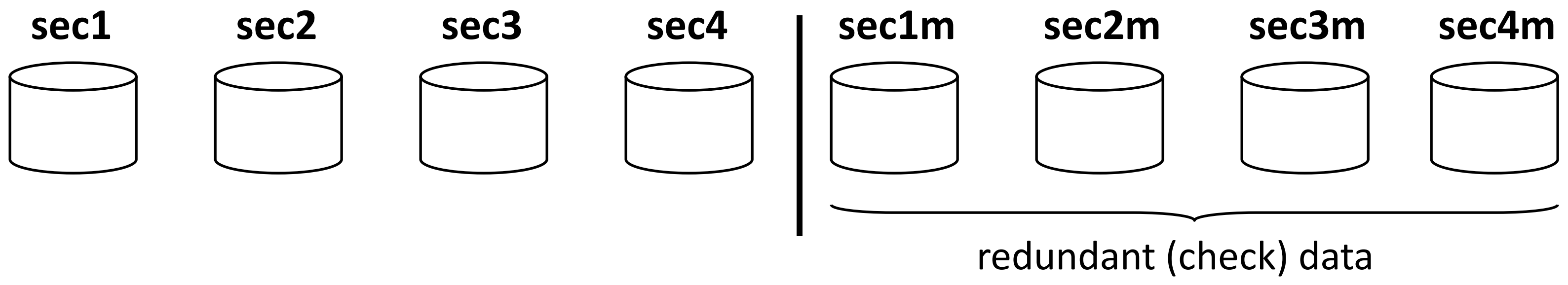
RAID: Level 0 (Striping, No Redundancy)

Assumes one stripe = four blocks



- **Multiple smaller disks as opposed to one big disk**
  - Multiple blocks can be accessed in parallel to increase the performance
  - Works well for large data requests
- **E.g., a 4-disk system gives four times the throughput (R/W) of a 1-disk system**
  - Same cost as one big disk – assuming 4 small disks cost the same as one big disk
- **What if one disk fails?**
  - No redundancy, data is lost
  - More likely to fail as the number of disks increases

RAID: Level 1 (Redundancy via Mirroring)



- **# redundant disks = # of data disks, so always two copies of the data**
  - Twice the cost of one big disk
  - Writes are made to both sets of disks and have no speed improvement
  - Reads can be 2 times faster

- **If a disk fails, the system just goes to the "mirror" for the data**

RAID: Design Choices

Category	Level	Disks	Features
Striping	0	N	No fault tolerance
Mirroring, no striping	1	2N	Expensive
Striping Rarely used	2 3	N + m N + 1	Hamming code Parity disk
Block level Striping Parity	4 5 6	N + 1 N + 1 N + 2	Parity disk Distributed parity Dual distributed parity

- **Parity: used for error detection**
  - E.g., data: 00001111, parity bit: 0
- **Hamming code: limited error correction**
  - Using multiple parity bits to locate 1 error bit

How is the Hierarchy Managed?

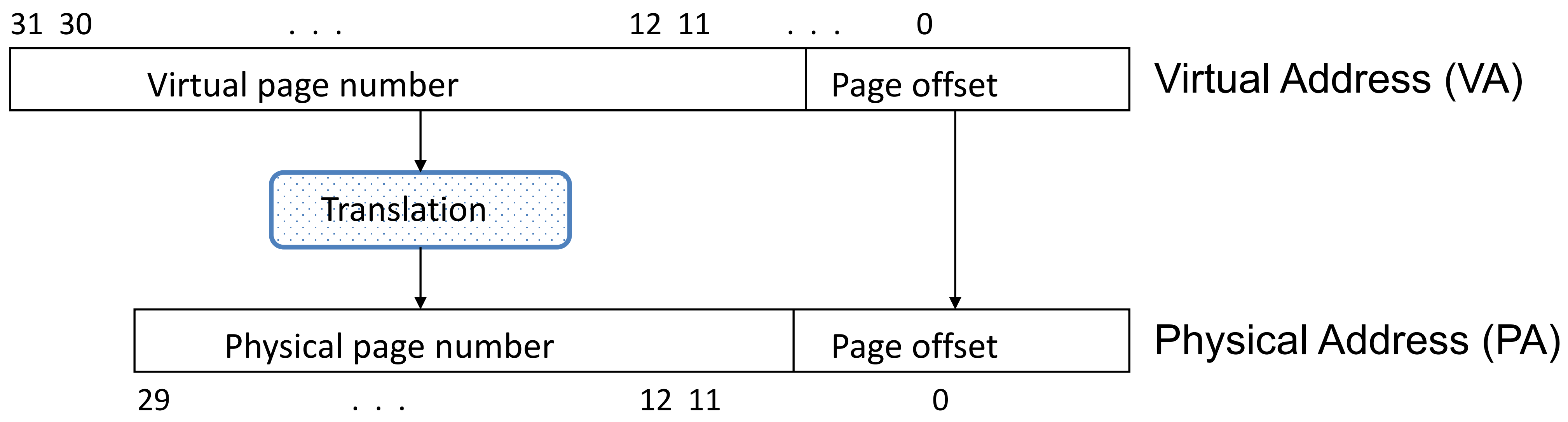
- **registers ↔ memory**
  - by compiler (or programmer)

- **registers ↔ cache ↔ main memory**
  - by the cache & memory controller hardware

- **main memory ↔ external storage (flash, disk)**
  - Static: by the programmer with OS support (files)

  - Dynamic: by the operating system (virtual memory)
    - virtual address to physical address mapping
    - assisted by the hardware (TLB, page tables)

Virtual Memory

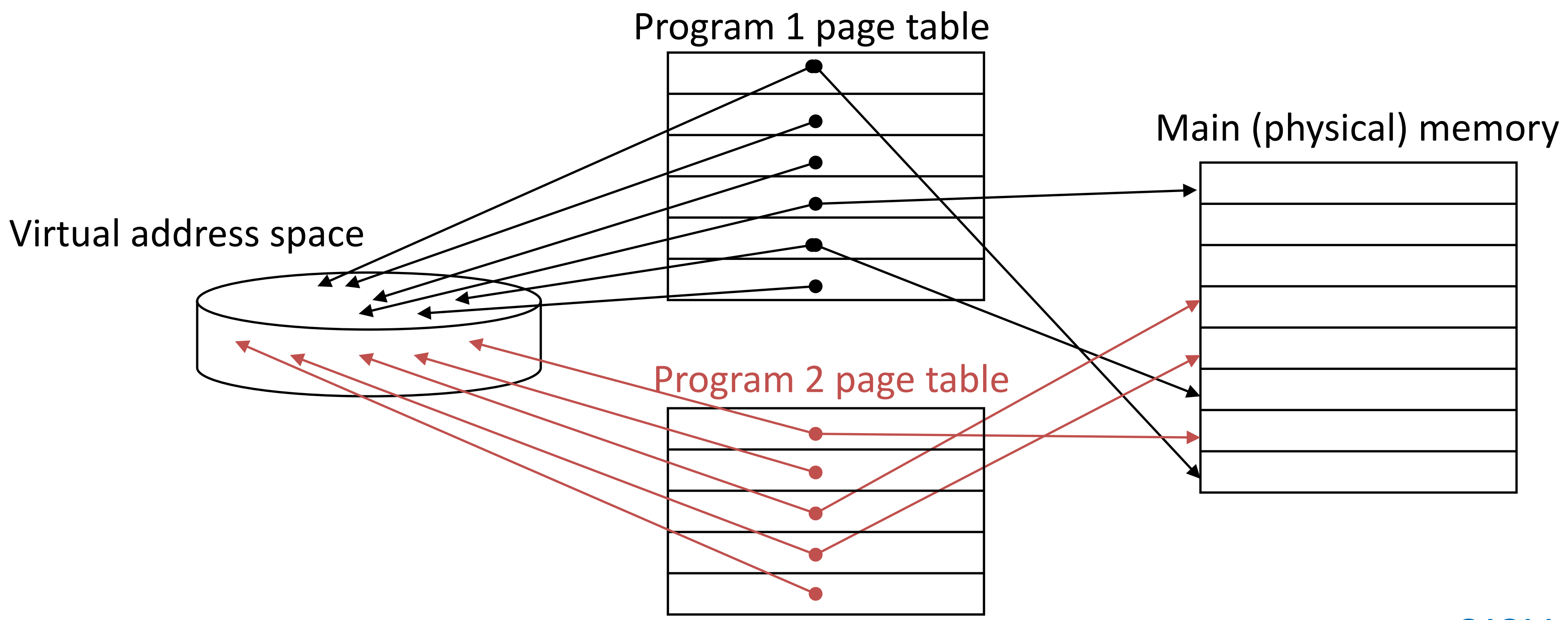


- **A virtual address must first be translated to a physical address to access memory**
  - Basic unit: page (e.g., 1KB to 64KB)
- **Virtual memory size can be larger than physical memory size**
  - Pages can be stored in the secondary storage

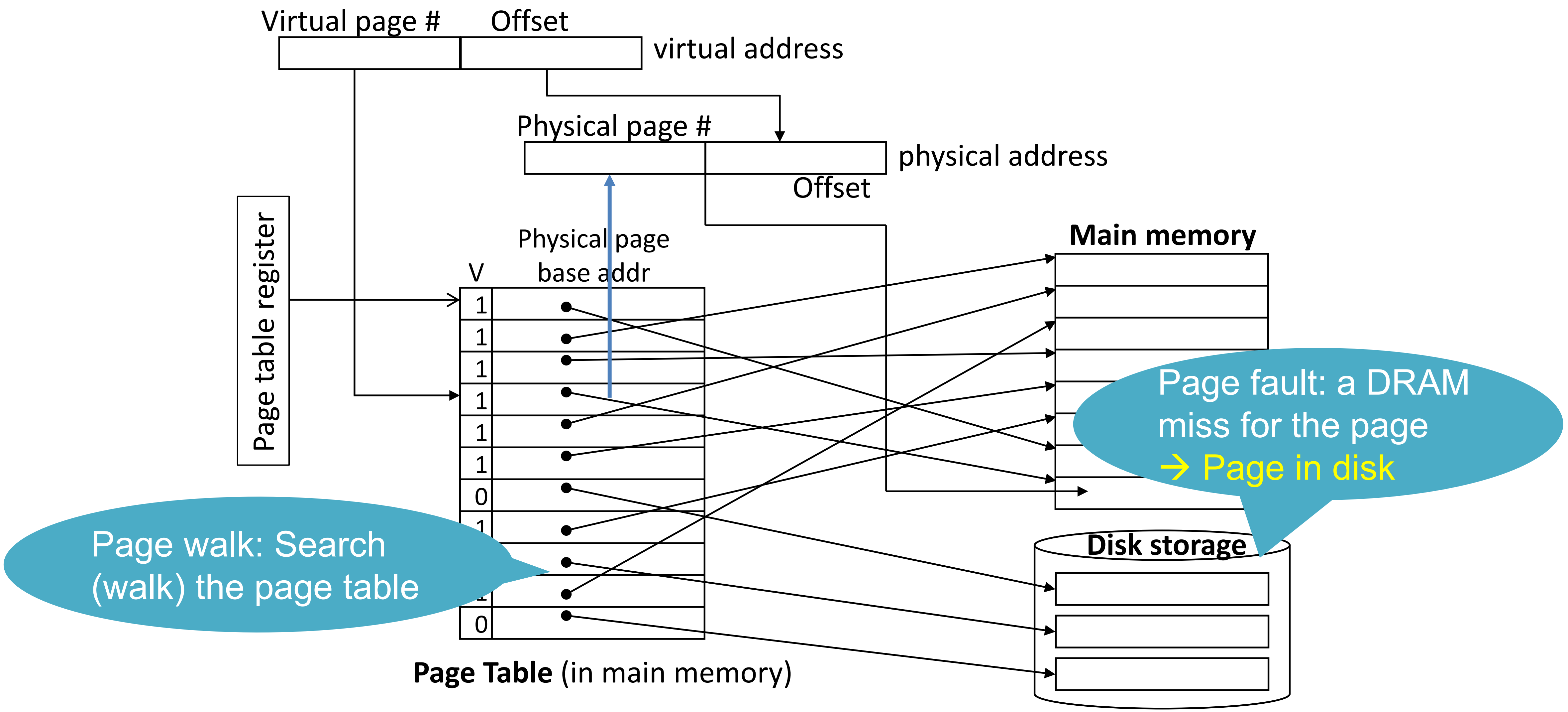
SJSU SAN JOSÉ STATE UNIVERSITY

Address Space Isolation

- **Allows efficient and safe sharing of main memory among multiple processes**
  - The starting location of each page is contained in the program's page table
  - Create the illusion that each program has a large consecutive memory space
  - Improving memory utilization: code can be loaded anywhere the OS can find space



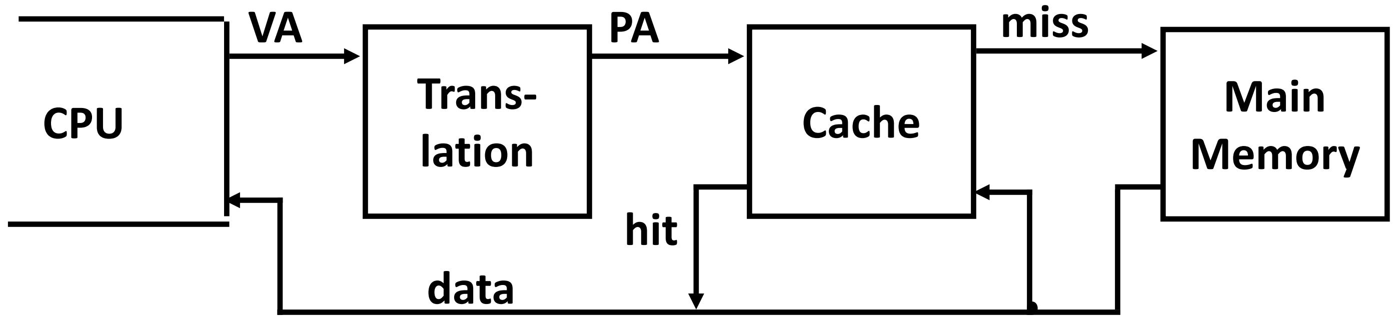
Address Translation Mechanisms



SJSU SAN JOSÉ STATE UNIVERSITY

Virtual Addressing with a Cache

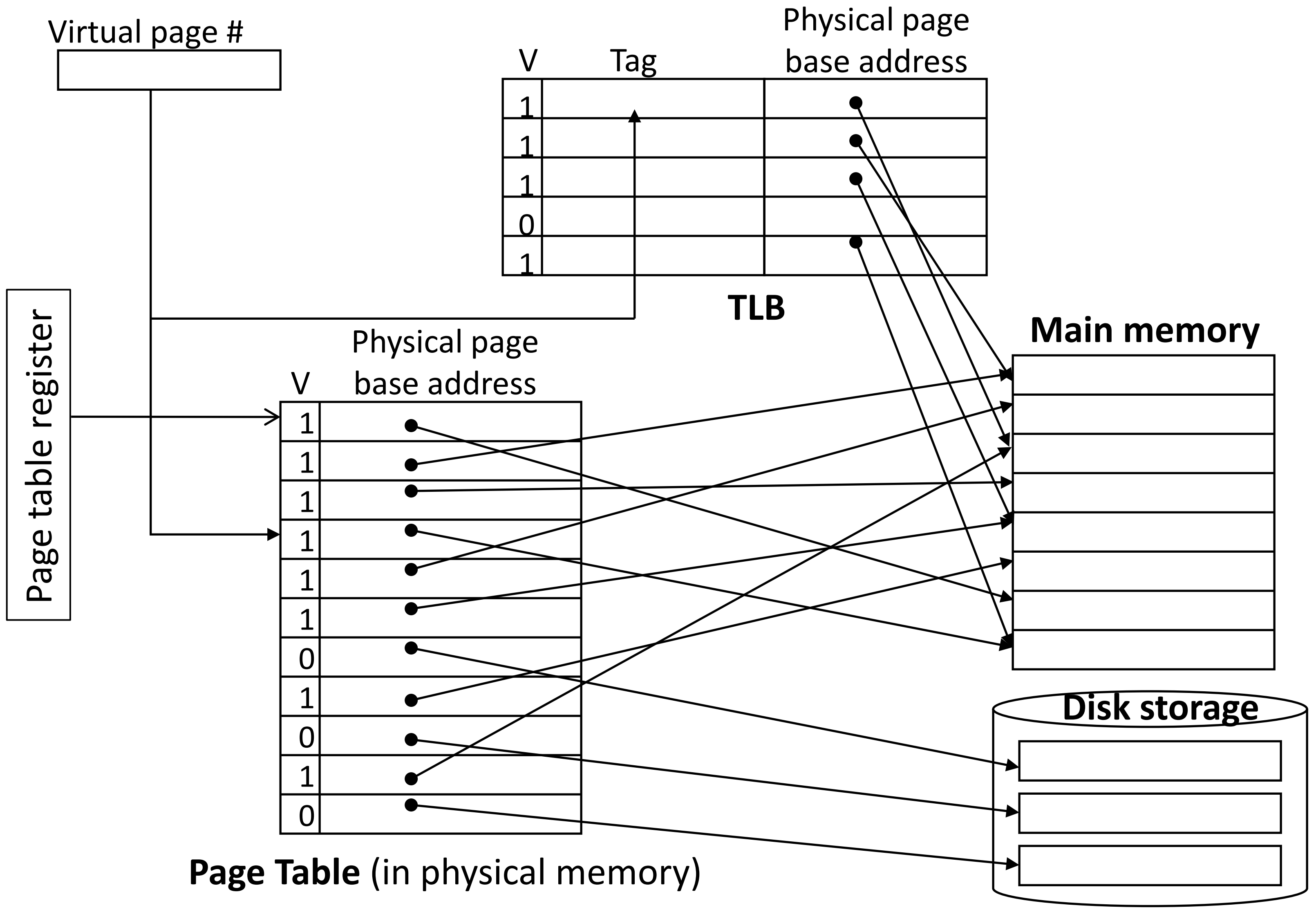
- **It takes an extra memory access to translate a VA to a PA**



- **This makes memory (cache) accesses very expensive**

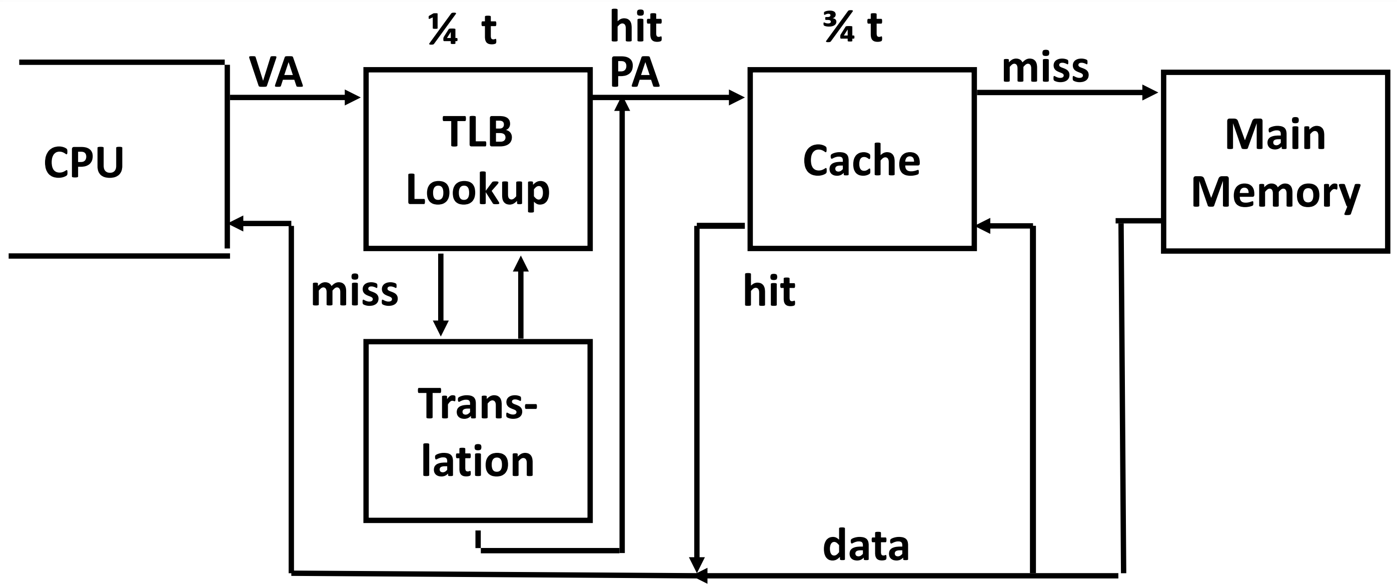
- **The hardware solution is to use a Translation Lookaside Buffer (TLB)**
  - A small cache that keeps track of recently used address mappings to avoid having to do a page table lookup in main memory

Making Address Translation Fast



Page Table (in physical memory)

A TLB in the Memory Hierarchy



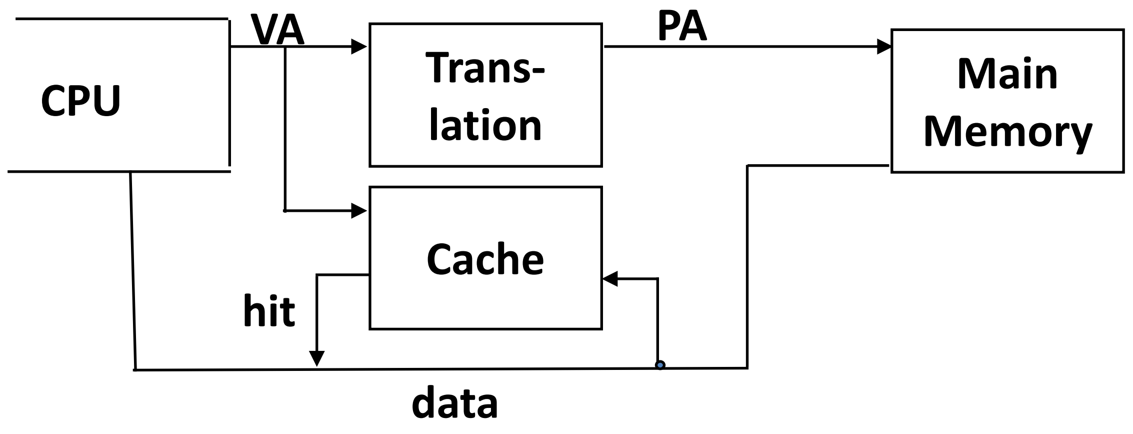
- **A TLB miss – is it a page fault or merely a TLB miss?**
  - If the page is loaded into main memory, then the TLB miss can be handled by loading the translation information from the page table into the TLB (10's of cycles)
  - If the page is not in main memory, then it's a true page fault (1,000,000's)

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

SJSU SAN JOSÉ STATE UNIVERSITY

Why Not a Virtually Addressed Cache?

- **A virtually addressed cache would only require address translation on cache misses**

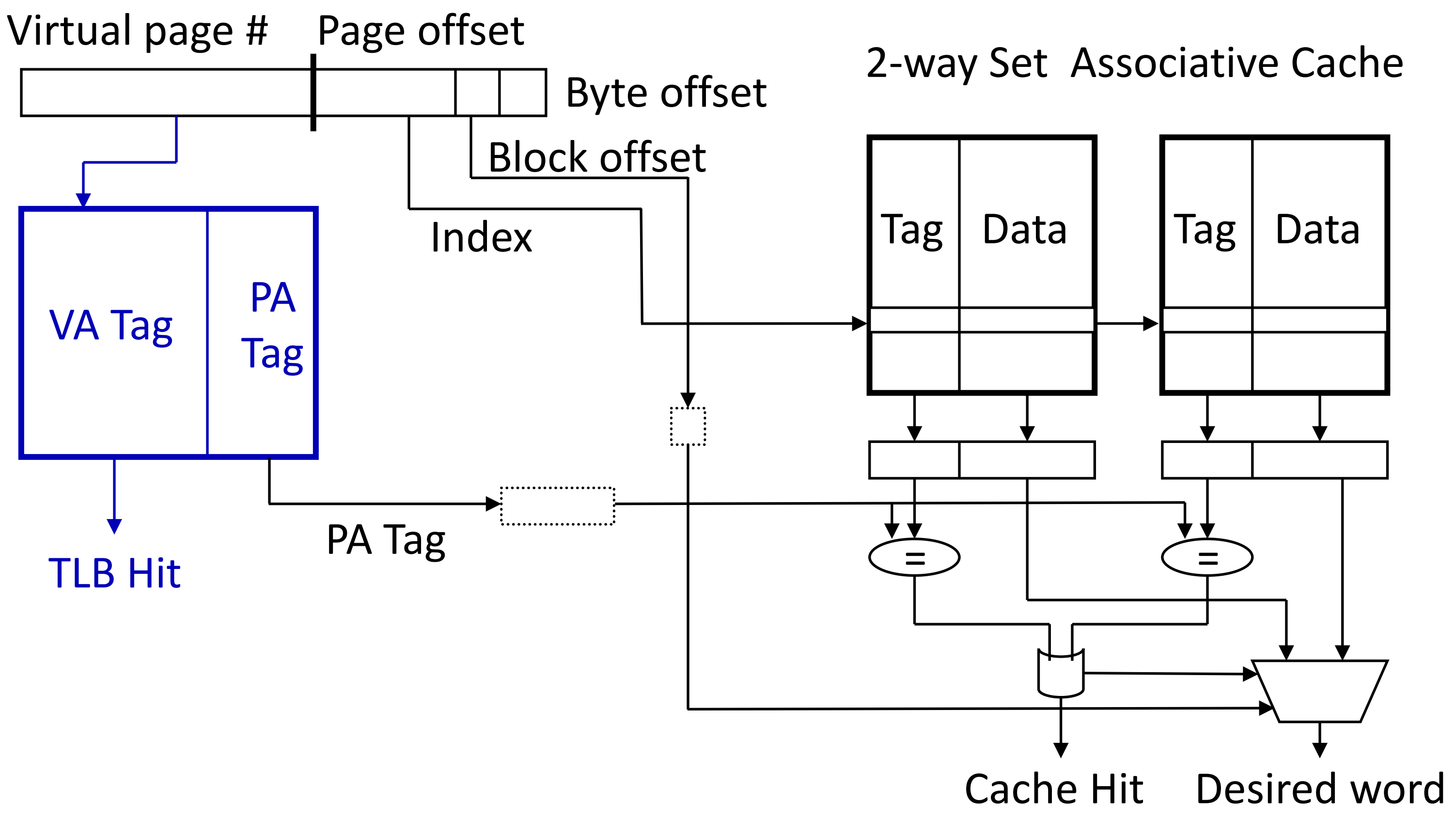


Aliasing issue:

- Two programs which are sharing data will have two different VA for the same PA
  - Two copies of the data in cache and two entries in the TLB → coherence issues
  - Must update all cache entries with the same physical address or the memory becomes inconsistent
- No address space isolation: data can be corrupted

Reducing Translation Time

- **Overlapping the cache access with the TLB access**
  - Higher bits for TLB; lower bits for cache.



SAN JOSÉ STATE UNIVERSITY *powering* SILICON VALLEY