نموذج رقم (2)

COURSE TITLE

Course Code	:30102315
Credit Hours	:3
Prerequisite	:30102214

Instructor Information

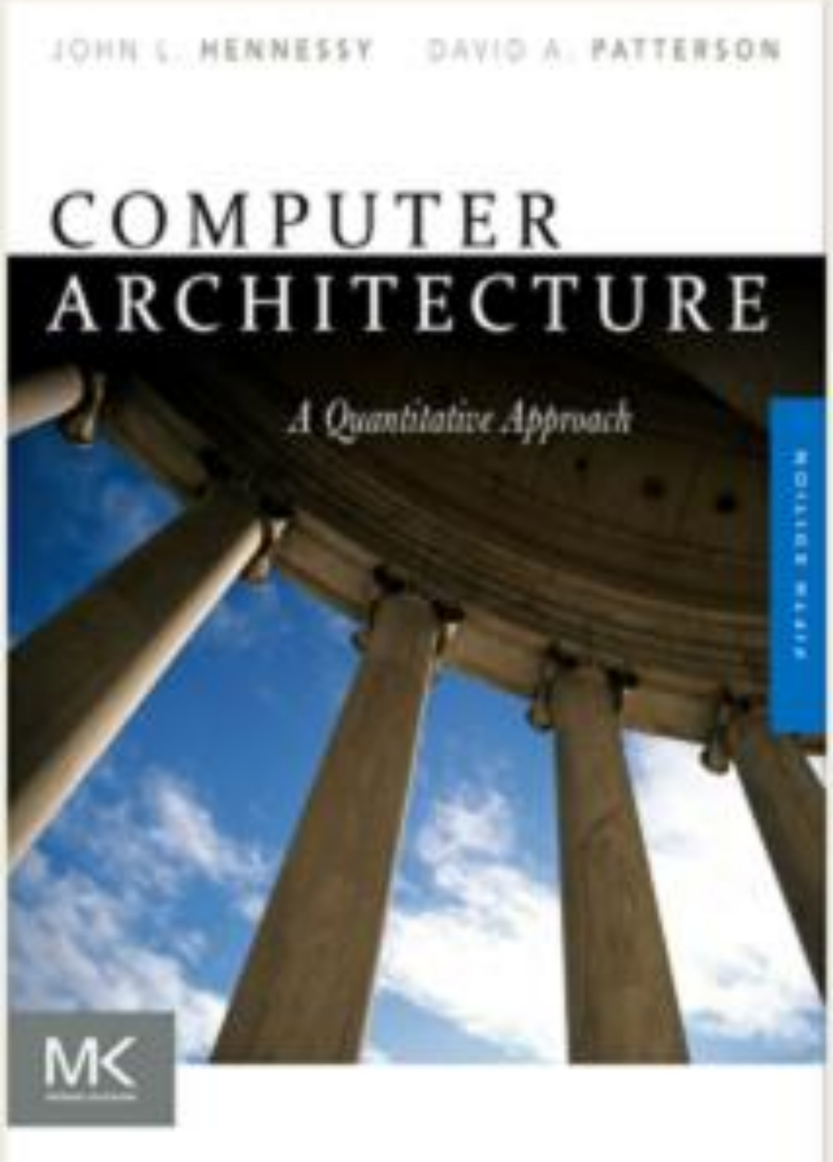
Name	: Dr. Rushdi Saleem Abu Zneit				
Office No.	**18, building 17 floor 3**				
Tel (Ext)	**Contact Telephone(Department Telephone)**				
E-mail	dr.rushdizneit@bau.edu.jo				
Office Hours	**Online**				
Class Times	Building	Day	Start Time	End Time	Room No.
		Sun,Tue, Thu	10:00	11:00	**Online**

Course description {From course plan}

Course Title; Computer Architecture
Credit Hour(3)

[Pre-req. 30102214

Textbook: Textbook Title

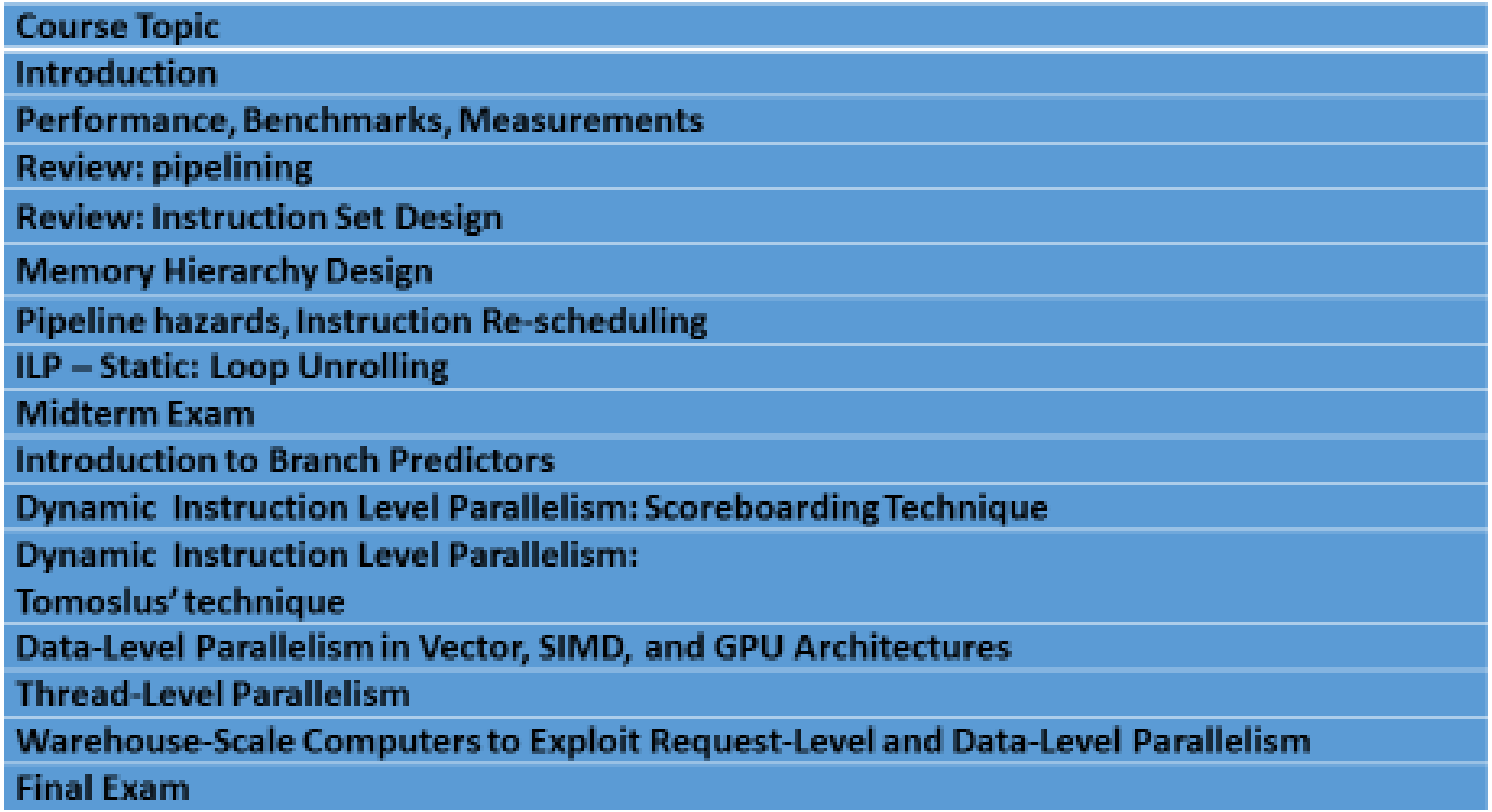


COURSE OBJECTIVES

The module aims to provide students with a fundamental knowledge of computer hardware and computer systems, with an emphasis on system design and performance. The module concentrates on the principles underlying systems organization, issues in computer system design, and contrasting implementations of modern systems. The module is central to the aims of the Computing Systems degree course, for which it is core.

COURSE SYLLABUS

Week	Course Topic
Week 1	Course overview & Introduction
Week 2	Performance, Benchmarks, Measurements
Week 3	Pipelining Review
Week 4	Review: Instruction Set Design & Memory Hierarchy Design
Week 5	Memory Hierarchy Design
Week 6	Pipeline hazards, Instruction Re-scheduling
Week 7	ILP – Static: Loop Unrolling
Week 8	Midterm Exam
Week 9	Introduction to Branch Predictors
Week 10	Dynamic Instruction Level Parallelism: Scoreboarding Technique
Week 11	Dynamic Instruction Level Parallelism: Tomoslus' technique
Week 12	Data-Level Parallelism in Vector, SIMD, and GPU Architectures
Week 13	Data-Level Parallelism in Vector, SIMD, and GPU Architectures
Week 14	Thread-Level Parallelism
Week 15	Warehouse-Scale Computers to Exploit Request-Level and Data-Level Parallelism
Week 16	Final Exam



Course Topic
Introduction
Performance, Benchmarks, Measurements
Review: pipelining
Review: Instruction Set Design
Memory Hierarchy Design
Pipeline hazards, Instruction Re-scheduling
ILP – Static: Loop Unrolling
Midterm Exam
Introduction to Branch Predictors
Dynamic Instruction Level Parallelism: Scoreboarding Technique
Dynamic Instruction Level Parallelism: Tomoslus’ technique
Data-Level Parallelism in Vector, SIMD, and GPU Architectures
Thread-Level Parallelism
Warehouse-Scale Computers to Exploit Request-Level and Data-Level Parallelism
Final Exam

Week 6

Advanced Optimizations

2) Way Prediction

- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - “Way selection”
  - Increases mis-prediction penalty

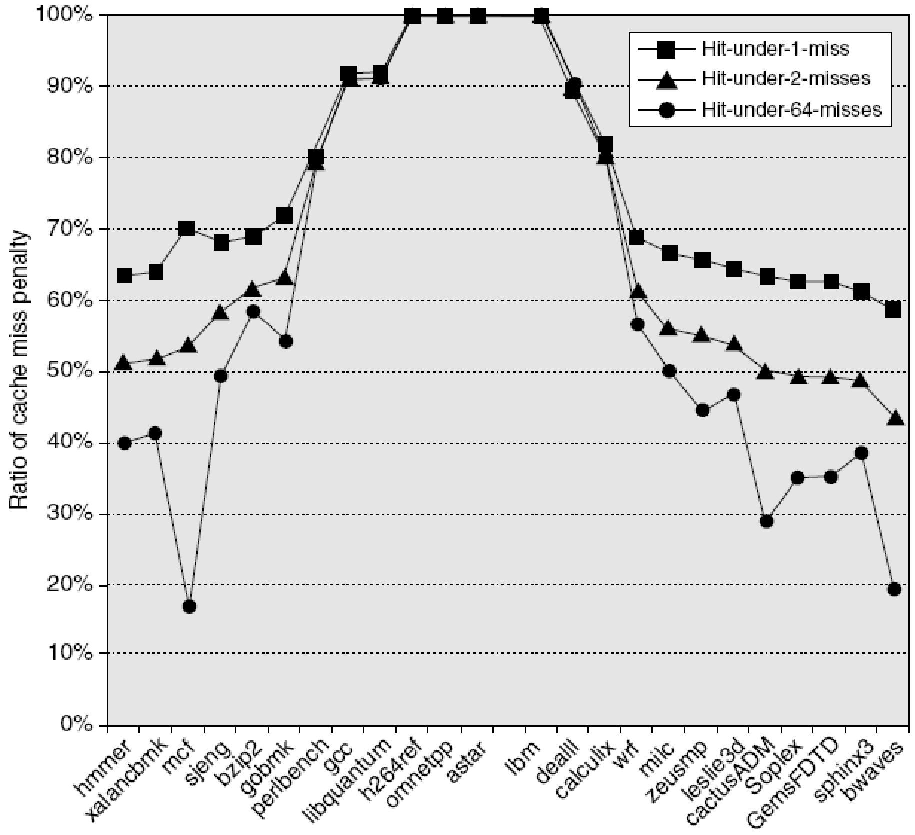
3) Pipelining Cache

- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles

- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

4) Nonblocking Caches

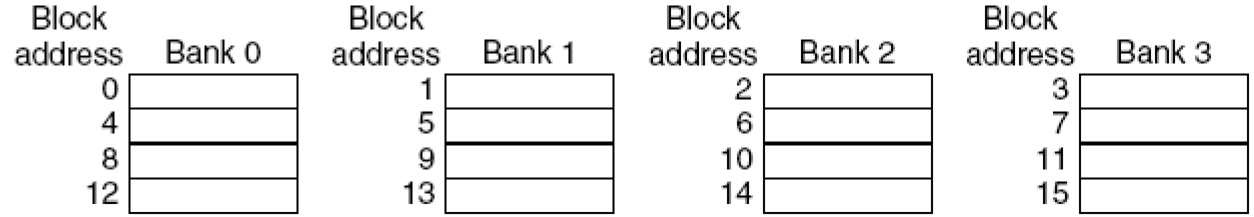
- Allow hits before previous misses complete
  - “Hit under miss”
  - “Hit under multiple miss”
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty



5) Multibanked Caches

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2

- Interleave banks according to block address



Block address	Bank 0	Block address	Bank 1	Block address	Bank 2	Block address	Bank 3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	

Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

6) Critical Word First, Early Restart

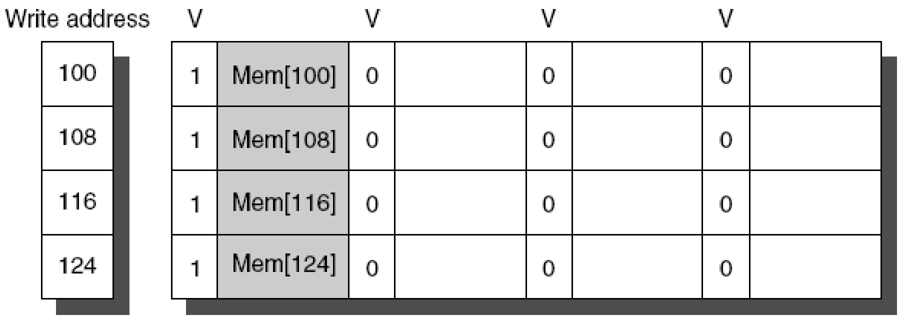
- Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
- Early restart
  - Request words in normal order
  - Send missed word to the processor as soon as it arrives

- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

Advanced Optimizations

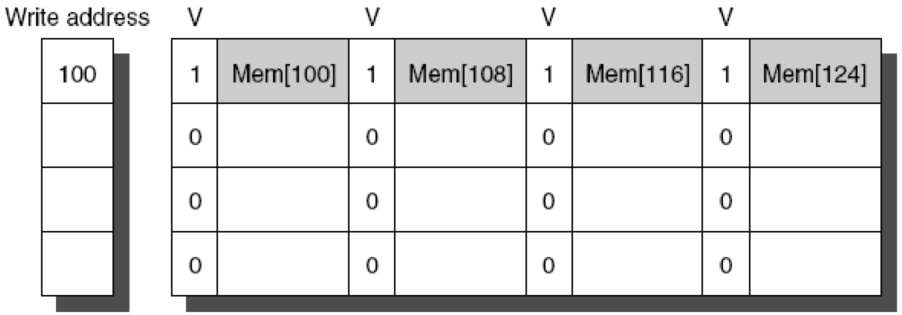
7) Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses



Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

No write buffering

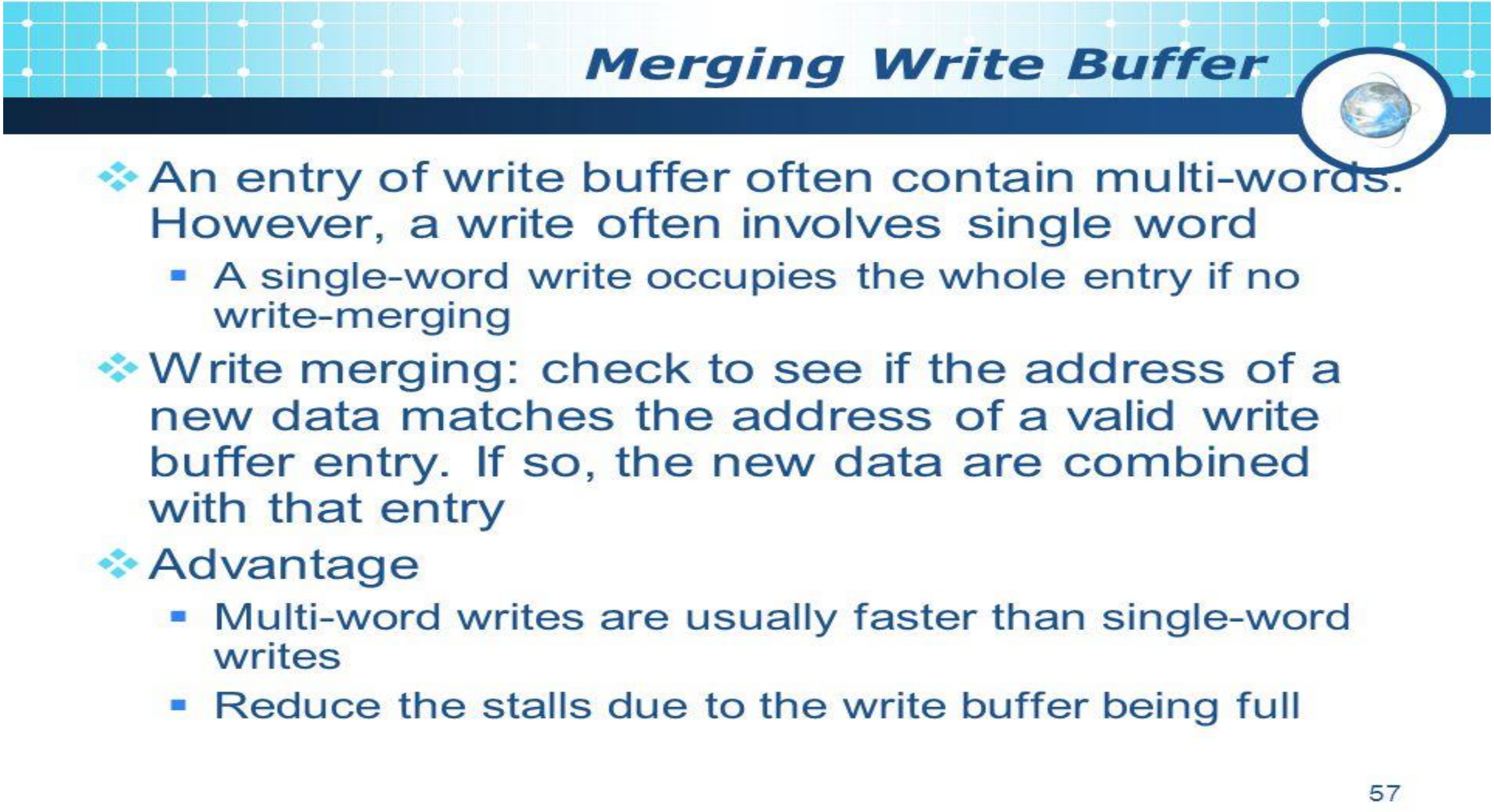


Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

Write buffering

In a CPU cache, a write buffer can be used to hold data being written from the cache to main memory or to the next cache in the memory hierarchy. This is a variation of write-through caching called buffered write-through.

Use of a write buffer in this manner frees the cache to service read requests while the write is taking place. It is especially useful for very slow main memory in that subsequent reads are able to proceed without waiting for long main memory latency. When the write buffer is full (i.e. all buffer entries are occupied), subsequent writes still have to wait until slots are freed. Subsequent reads could be served from the write buffer.



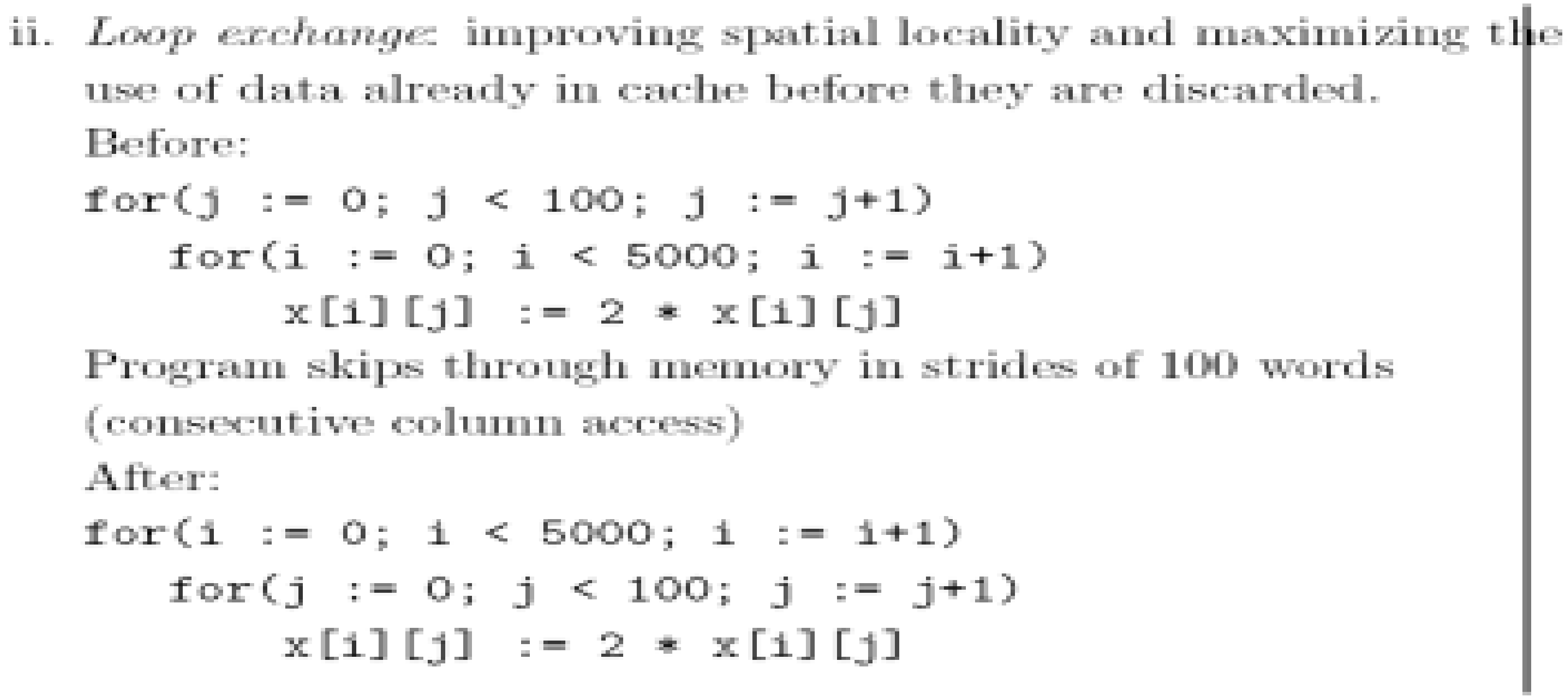
Merging Write Buffer

- An entry of write buffer often contain multi-words. However, a write often involves single word
  - A single-word write occupies the whole entry if no write-merging
- Write merging: check to see if the address of a new data matches the address of a valid write buffer entry. If so, the new data are combined with that entry
- Advantage
  - Multi-word writes are usually faster than single-word writes
  - Reduce the stalls due to the write buffer being full

8) Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order
- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

Reducing Cache Misses:
5. Compiler Optimizations



ii. *Loop exchange:* improving spatial locality and maximizing the use of data already in cache before they are discarded.

Before:

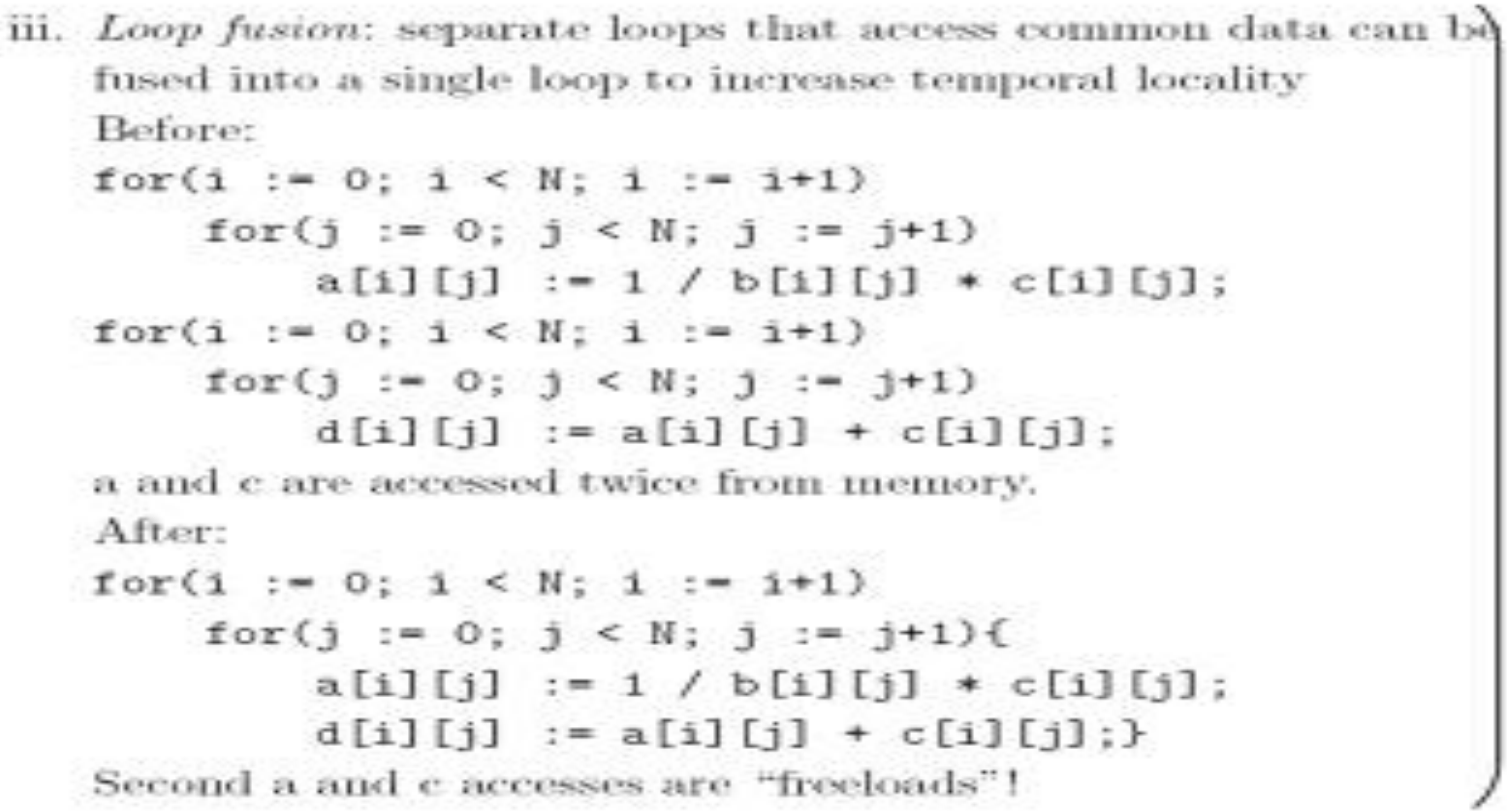
```
for(j := 0; j < 100; j := j+1)
   for(i := 0; i < 5000; i := i+1)
       x[i][j] := 2 * x[i][j]
```

Program skips through memory in strides of 100 words (consecutive column access)

After:

```
for(i := 0; i < 5000; i := i+1)
   for(j := 0; j < 100; j := j+1)
       x[i][j] := 2 * x[i][j]
```

Reducing Cache Misses:
5. Compiler Optimizations



iii. *Loop fusion*: separate loops that access common data can be fused into a single loop to increase temporal locality

Before:

```
for(i := 0; i < N; i := i+1)
    for(j := 0; j < N; j := j+1)
        a[i][j] := 1 / b[i][j] * c[i][j];
for(i := 0; i < N; i := i+1)
    for(j := 0; j < N; j := j+1)
        d[i][j] := a[i][j] + c[i][j];
```

a and c are accessed twice from memory.

After:

```
for(i := 0; i < N; i := i+1)
    for(j := 0; j < N; j := j+1){
        a[i][j] := 1 / b[i][j] * c[i][j];
        d[i][j] := a[i][j] + c[i][j];}
```

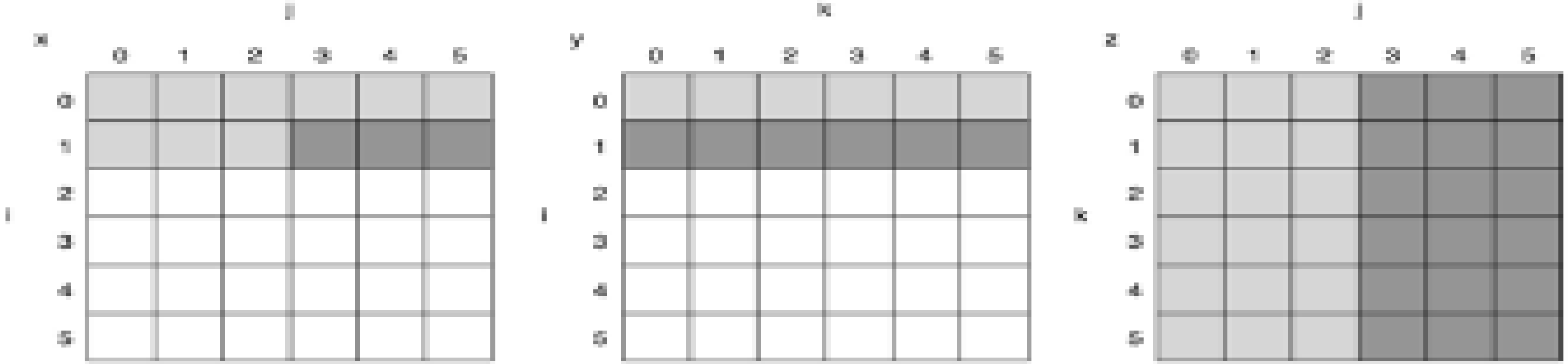
Second a and c accesses are "freeloads"!

Reducing Cache Misses:
5. Compiler Optimizations

- *Blocking*: improve temporal and spatial locality
  - a) multiple arrays are accessed in both ways (i.e., row-major and column-major), namely, orthogonal accesses that can not be helped by earlier methods
  - b) concentrate on submatrices, or blocks

```
for(i := 0; i < N; i := i+1)
    for(j := 0; j < N; j := j+1){
        r := 0;
        for(k := 0; k < N; k := k+1)
            r := r + y[i][k] * z[k][j];
        x[i][j] := r;}
```

  - c) All *N***N* elements of Y and Z are accessed *N* times and each element of X is accessed once. Thus, there are N^3 operations and $2N^3 + N^2$ reads! Capacity misses are a function of *N* and cache size in this case.



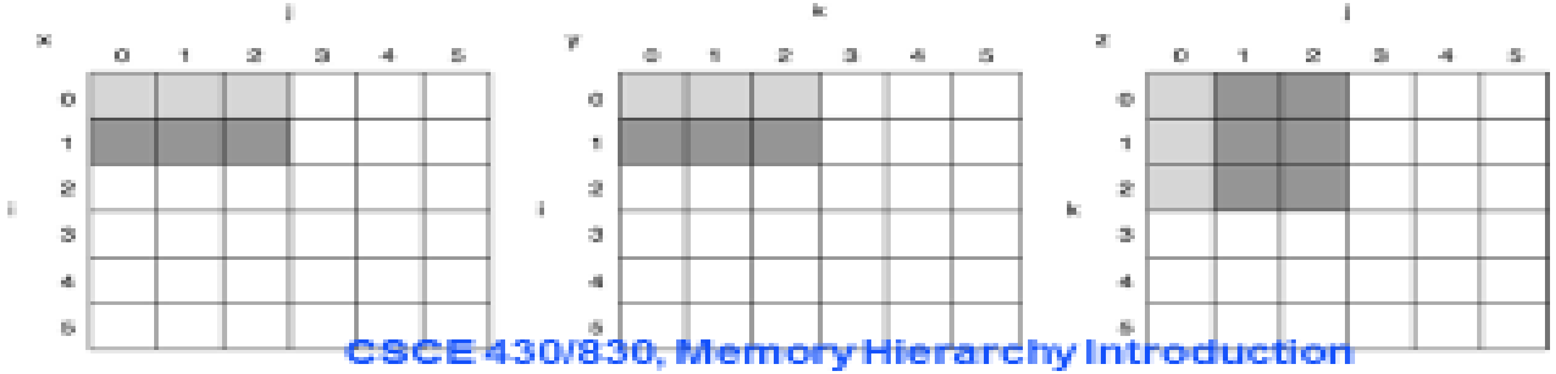
Reducing Cache Misses:

5. Compiler Optimizations (cont'd)

- *Blocking*: improve temporal and spatial locality
  a) To ensure that elements being accessed can fit in the cache, the original code is changed to compute a submatrix of size *B*B*, where *B* is called the *blocking factor*.
  b) To total number of memory words accessed is $2N^3/B + N^2$
  c) Blocking exploits a combination of spatial (Y) and temporal (Z) locality.

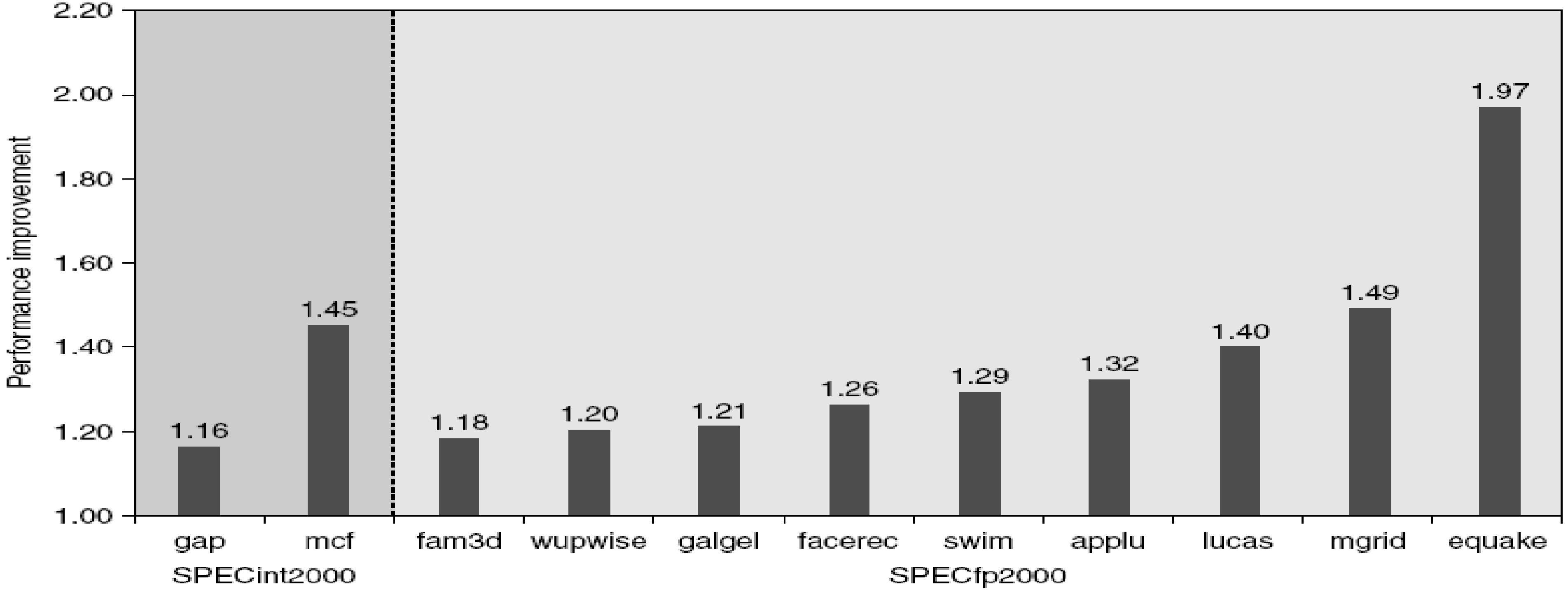
After:

```
for(jj := 0; jj < N; jj := jj+B)
for(kk := 0; kk < N; kk := kk+B)
for(i := 0; i < N; i := i+1)
    for(j := jj; j < min(jj+B-1,N); j := j+1){
        r := 0;
        for(k := kk; k < min(kk+B-1,N); k := k+1)
            r := r + y[i][k] * z[k][j];
```



9) Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)



Pentium 4 Pre-fetching

10) Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn’t cause exceptions

- Register prefetch
  - Loads data into register
- Cache prefetch
  - Loads data into cache

- Combine with loop unrolling and software pipelining

Advanced Optimizations

Reducing Cache Miss Penalty:
Compiler-Controlled Prefetching

- Compiler inserts prefetch instructions
- An Example

```
for(i:=0; i<3; i:=i+1)
    for(j:=0; j<100; j:=j+1)
        a[i][j] := b[j][0] * b[j+1][0]
```

- 16-byte blocks, 8KB cache, 1-way write back, 8-byte elements; What kind of locality, if any, exists for a and b?
  a. 3 100-element rows (100 columns) visited; spatial locality: even-indexed elements miss and odd-indexed elements hit, leading to 3*100/2 = 150 misses
  b. 101 rows and 3 columns visited; no spatial locality, but there is temporal locality: same element is used in *i*th and (i + 1)st iterations and the same element is access in each *i* iteration (outer loop). 100 misses for b[j+1][0] when i = 0 and 1 miss for j = 0 for a total of 101 misses
- Assuming large penalty (100 cycles and at least 7 iterations must be prefetched). Splitting the loop into two, we have

Reducing Cache Miss Penalty:
3. Compiler-Controlled Prefetching

- Assuming that each iteration of the pre-split loop consumes 7 cycles and no conflict and capacity misses, then it consumes a total of 7*300 iteration cycles + 251*100 cache miss cycles = 27,200 cycles;

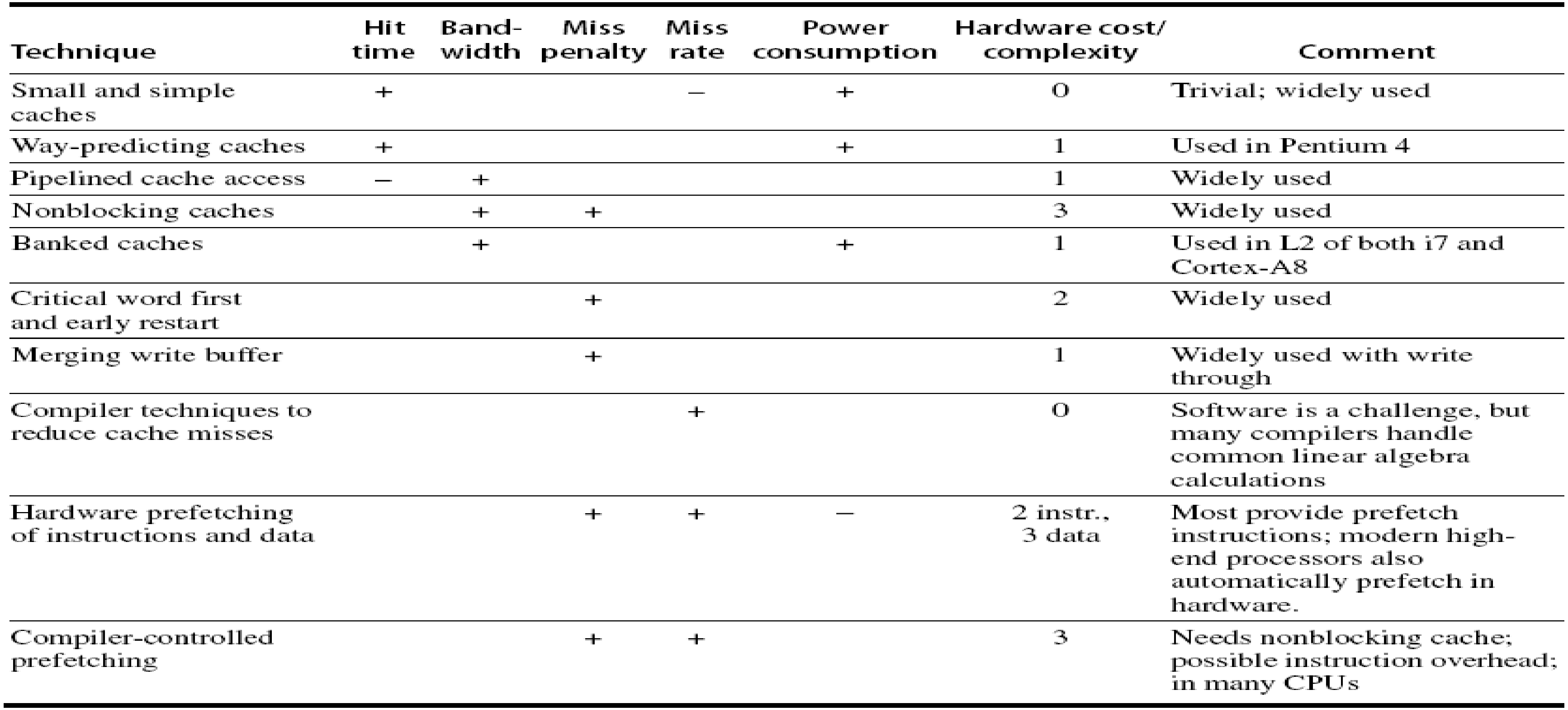
- **With prefetching instructions inserted:**

```
for(j:=0; j<100; j:=j+1){
        prefetch(b[j+7][0];
        prefetch(a[0][j+7];
        a[0][j] := b[j][0] * b[j+1][0];};
for(i:=1; i<3; i:=i+1)
        for(j:=0; j<100; j:=j+1){
                prefetch(a[i][j+7];
                a[i][j] := b[j][0] * b[j+1][0]}
```

Reducing Cache Miss Penalty:
3. Compiler-Controlled Prefetching (cont'd)

- An Example (continued)
  - the first loop consumes 9 cycles per iteration (due to the two prefetch instruction) and iterates 100 times for a total of 900 cycles,
  - the second loop consumes 8 cycles per iteration (due to the single prefetch instruction) and iterates 200 times for a total of 1,600 cycles,
  - during the first 7 iterations of the first loop array a incurs 4 cache misses, array b incurs 7 cache misses, for a total of (4+7)*100=1,100 cache miss cycles,
  - during the first 7 iterations of the second loop for *i = 1* and *i = 2* array a incurs 4 cache misses each, for total of (4+4)*100=800 cache miss cycles; array b does not incur any cache miss in the second split!
  - Total cycles consumed: 900+1600+1100+800= 4400
  - Prefetching improves performance: 27200/4400=6.2 folds!

Summary



Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.

Memory Technology

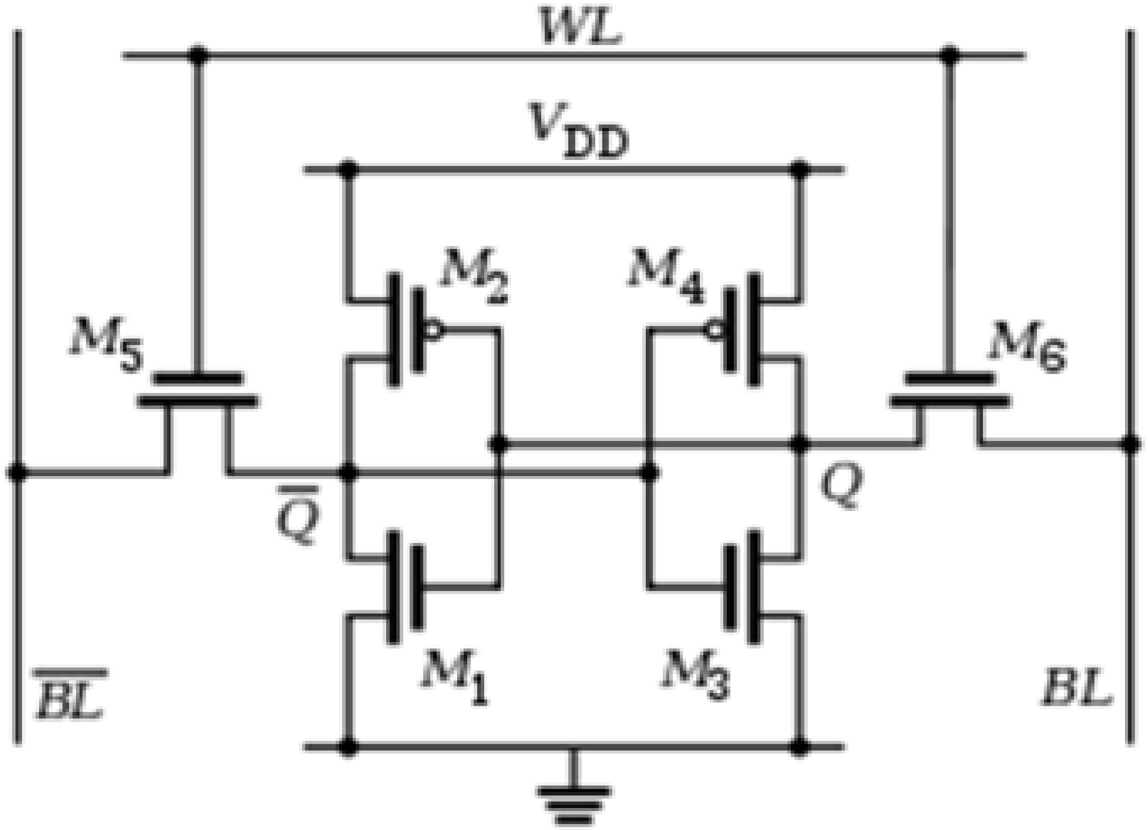
Memory Technology

- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory
- DRAM used for main memory, SRAM used for cache

Memory Technology

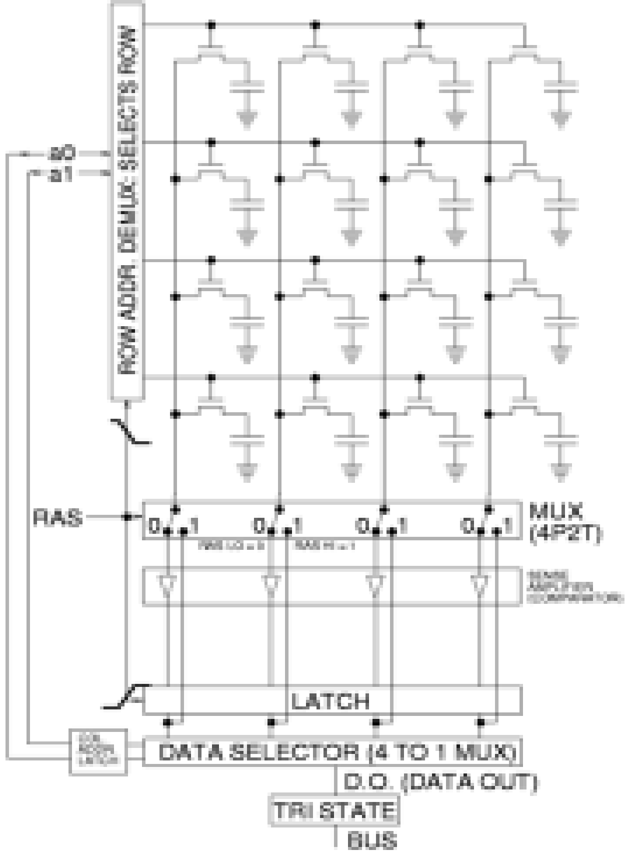
- SRAM
  - Requires low power to retain bit
  - Requires 6 transistors/bit
- DRAM
  - Must be re-written after being read
  - Must also be periodically refreshed
    - Every ~ 8 ms
    - Each row can be refreshed simultaneously
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

A SRAM Example



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

A DRAM Example



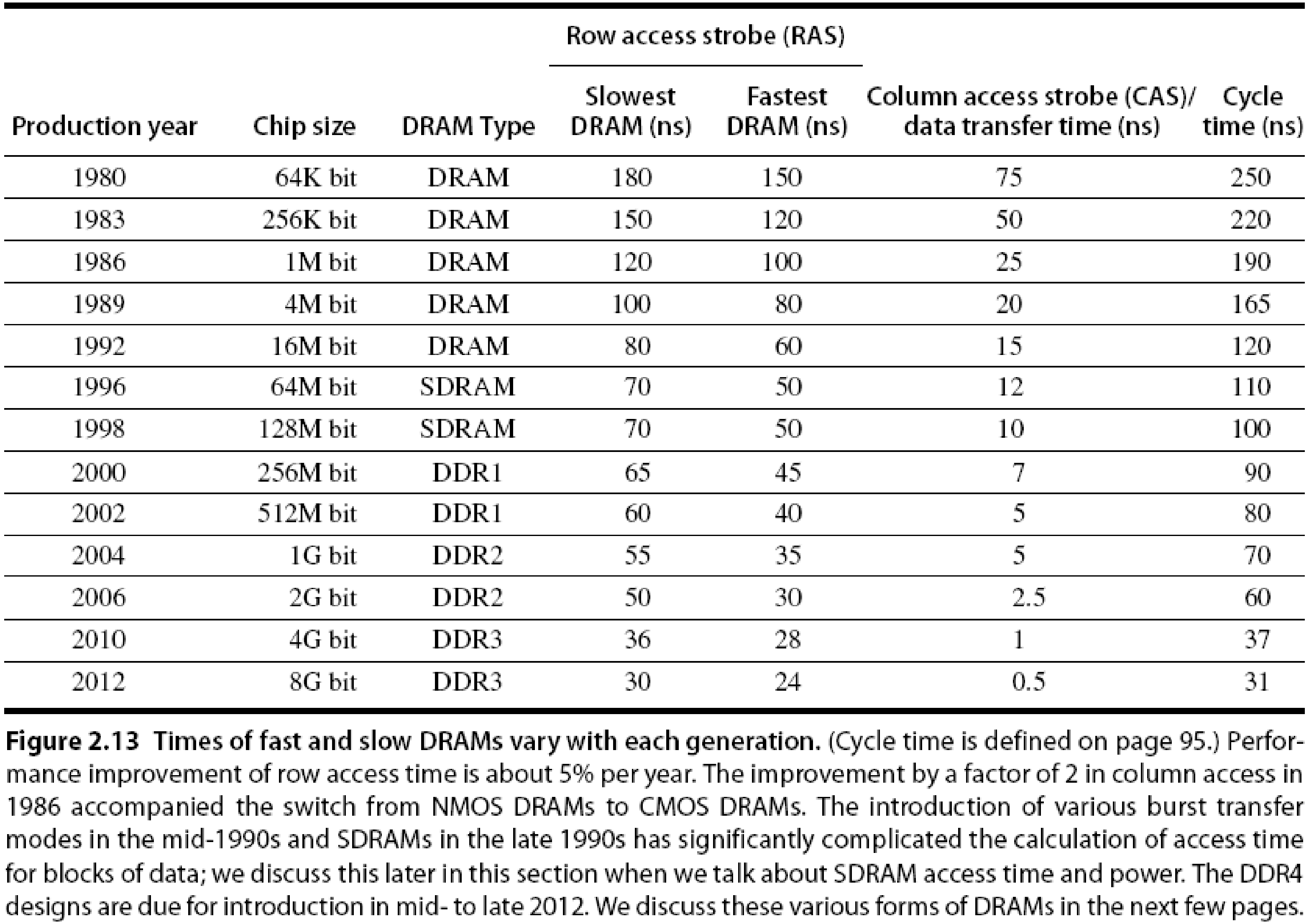
http://en.wikipedia.org/wiki/Dynamic_random-access_memory

Memory Technology

- Amdahl:
  - Memory capacity should grow linearly with processor speed
  - Unfortunately, memory capacity and speed has not kept pace with processors
- Some optimizations:
  - Multiple accesses to same row
  - Synchronous DRAM
    - Added clock to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces
  - Double data rate (DDR)
  - Multiple banks on each DRAM device

Memory Optimizations

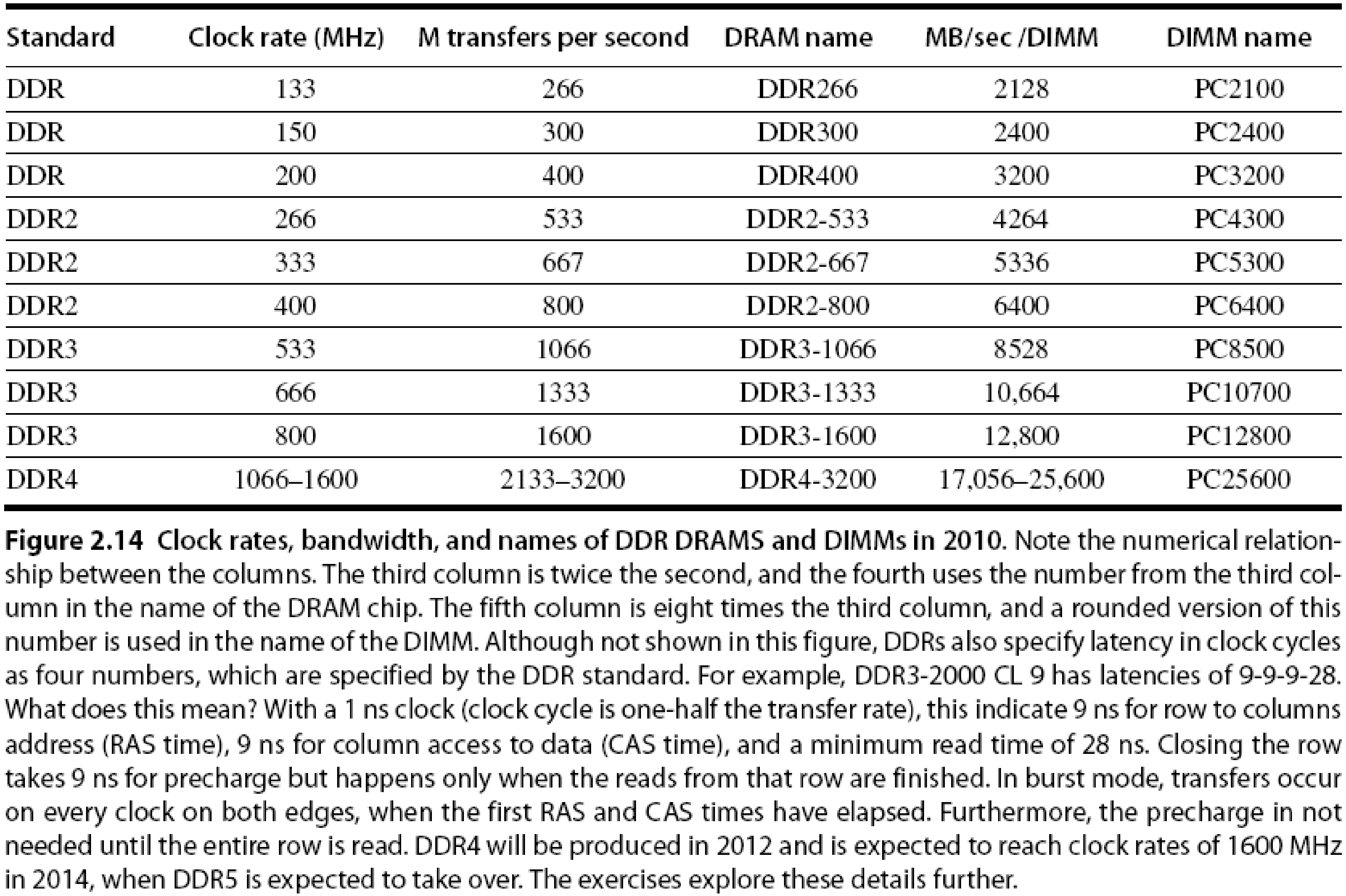
Memory Technology



Production year	Chip size	DRAM Type	Row access strobe (RAS) Slowest DRAM (ns)	Row access strobe (RAS) Fastest DRAM (ns)	Column access strobe (CAS)/ data transfer time (ns)	Cycle time (ns)
1980	64K bit	DRAM	180	150	75	250
1983	256K bit	DRAM	150	120	50	220
1986	1M bit	DRAM	120	100	25	190
1989	4M bit	DRAM	100	80	20	165
1992	16M bit	DRAM	80	60	15	120
1996	64M bit	SDRAM	70	50	12	110
1998	128M bit	SDRAM	70	50	10	100
2000	256M bit	DDR1	65	45	7	90
2002	512M bit	DDR1	60	40	5	80
2004	1G bit	DDR2	55	35	5	70
2006	2G bit	DDR2	50	30	2.5	60
2010	4G bit	DDR3	36	28	1	37
2012	8G bit	DDR3	30	24	0.5	31

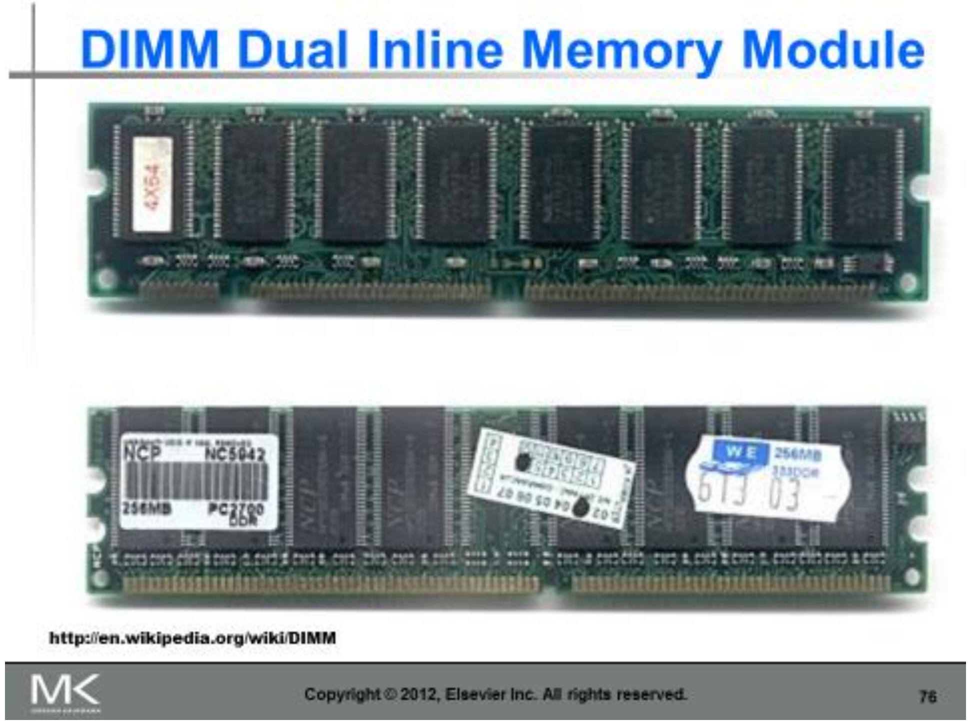
Figure 2.13 Times of fast and slow DRAMs vary with each generation. (Cycle time is defined on page 95.) Performance improvement of row access time is about 5% per year. The improvement by a factor of 2 in column access in 1986 accompanied the switch from NMOS DRAMs to CMOS DRAMs. The introduction of various burst transfer modes in the mid-1990s and SDRAMs in the late 1990s has significantly complicated the calculation of access time for blocks of data; we discuss this later in this section when we talk about SDRAM access time and power. The DDR4 designs are due for introduction in mid- to late 2012. We discuss these various forms of DRAMs in the next few pages.

Memory Optimizations



Standard	Clock rate (MHz)	M transfers per second	DRAM name	MB/sec /DIMM	DIMM name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1066–1600	2133–3200	DDR4-3200	17,056–25,600	PC25600

Figure 2.14 Clock rates, bandwidth, and names of DDR DRAMS and DIMMs in 2010. Note the numerical relationship between the columns. The third column is twice the second, and the fourth uses the number from the third column in the name of the DRAM chip. The fifth column is eight times the third column, and a rounded version of this number is used in the name of the DIMM. Although not shown in this figure, DDRs also specify latency in clock cycles as four numbers, which are specified by the DDR standard. For example, DDR3-2000 CL 9 has latencies of 9-9-9-28. What does this mean? With a 1 ns clock (clock cycle is one-half the transfer rate), this indicate 9 ns for row to columns address (RAS time), 9 ns for column access to data (CAS time), and a minimum read time of 28 ns. Closing the row takes 9 ns for precharge but happens only when the reads from that row are finished. In burst mode, transfers occur on every clock on both edges, when the first RAS and CAS times have elapsed. Furthermore, the precharge in not needed until the entire row is read. DDR4 will be produced in 2012 and is expected to reach clock rates of 1600 MHz in 2014, when DDR5 is expected to take over. The exercises explore these details further.



DIMM Dual Inline Memory Module

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

Memory Optimizations

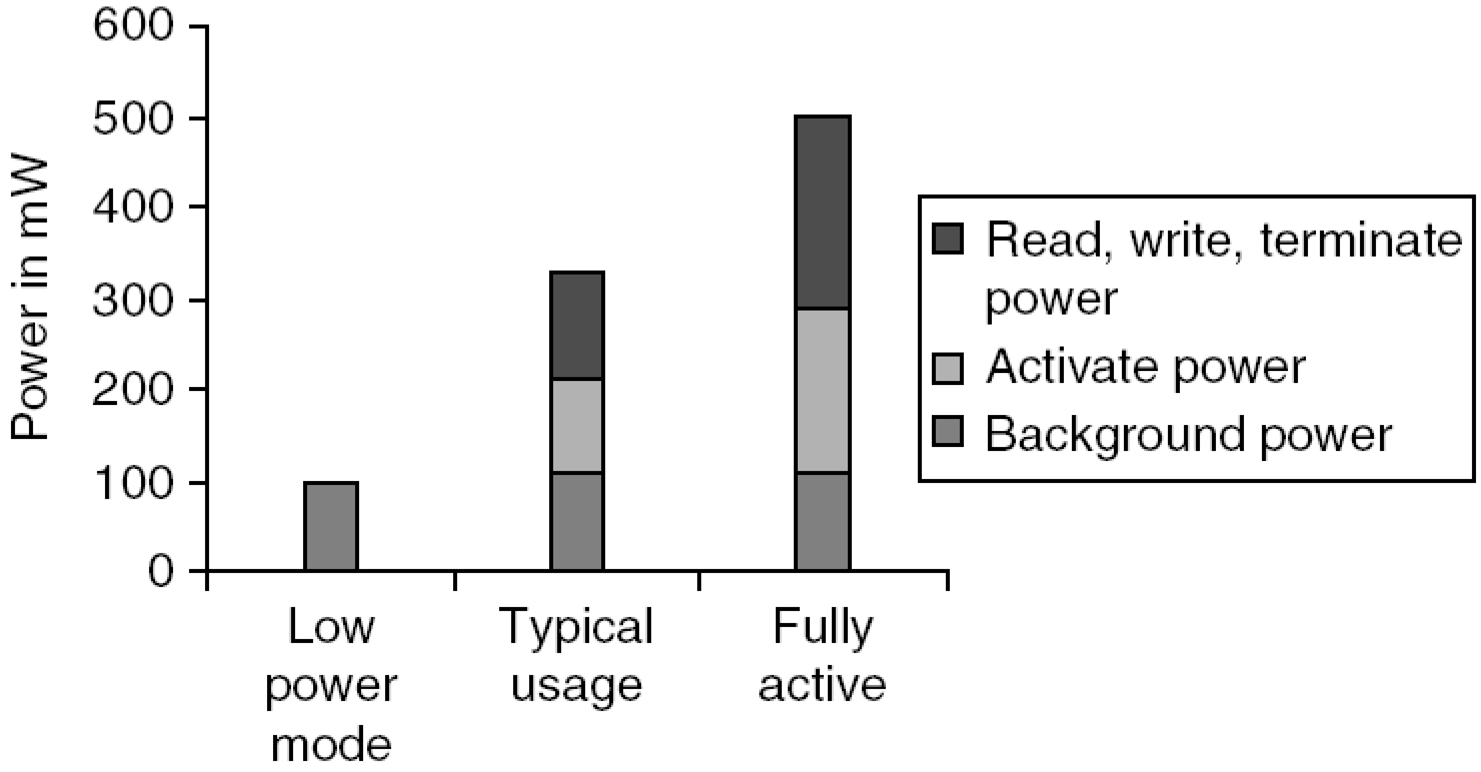
- DDR:
  - DDR2
    - Lower power (2.5 V -> 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1600 MHz
- GDDR5 is graphics memory based on DDR3

Memory Optimizations

- Graphics memory:
  - Achieve 2-5 X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketted DIMM modules

- Reducing power in SDRAMs:
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)

Memory Power Consumption



Flash Memory

- Type of EEPROM
- Must be erased (in blocks) before being overwritten
- Non volatile
- Limited number of write cycles
- Cheaper than SDRAM, more expensive than disk
- Slower than SRAM, faster than disk

Memory Dependability

- Memory is susceptible to cosmic rays
- *Soft errors*: dynamic errors
  - Detected and fixed by error correcting codes (ECC)
- *Hard errors*: permanent errors
  - Use sparse rows to replace defective rows

- Chipkill: a RAID-like error recovery technique
  - (Hamming codes)

Virtual Memory

- Protection via virtual memory
  - Keeps processes in their own memory space

- Role of architecture:
  - Provide user mode and supervisor mode
  - Protect certain aspects of CPU state
  - Provide mechanisms for switching between user mode and supervisor mode
  - Provide mechanisms to limit memory accesses
  - Provide TLB to translate addresses

Virtual Memory and Virtual Machines

Virtual Machines

- Supports isolation and security
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making the overhead more acceptable

- Allows different ISAs and operating systems to be presented to user programs
  - “System Virtual Machines”
  - SVM software is called “virtual machine monitor” or “hypervisor”
  - Individual virtual machines run under the monitor are called “guest VMs”

Virtual Memory and Virtual Machines

Impact of VMs on Virtual Memory

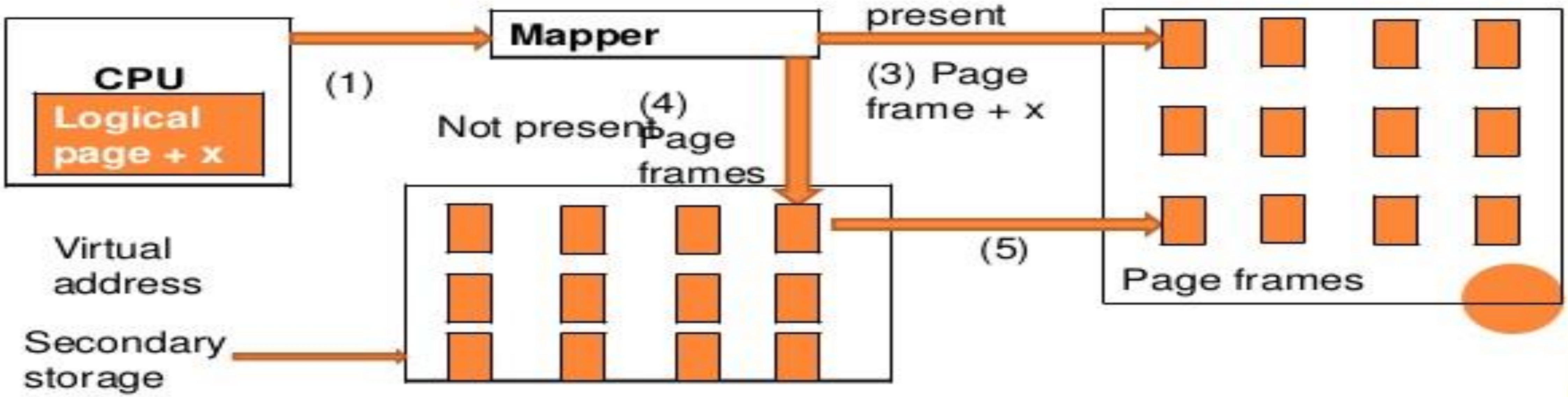
- Each guest OS maintains its own set of page tables
  - VMM adds a level of memory between physical and virtual memory called “real memory”
  - VMM maintains shadow page table that maps guest virtual addresses to physical addresses
    - Requires VMM to detect guest’s changes to its own page table
    - Occurs naturally if accessing the page table pointer is a privileged operation

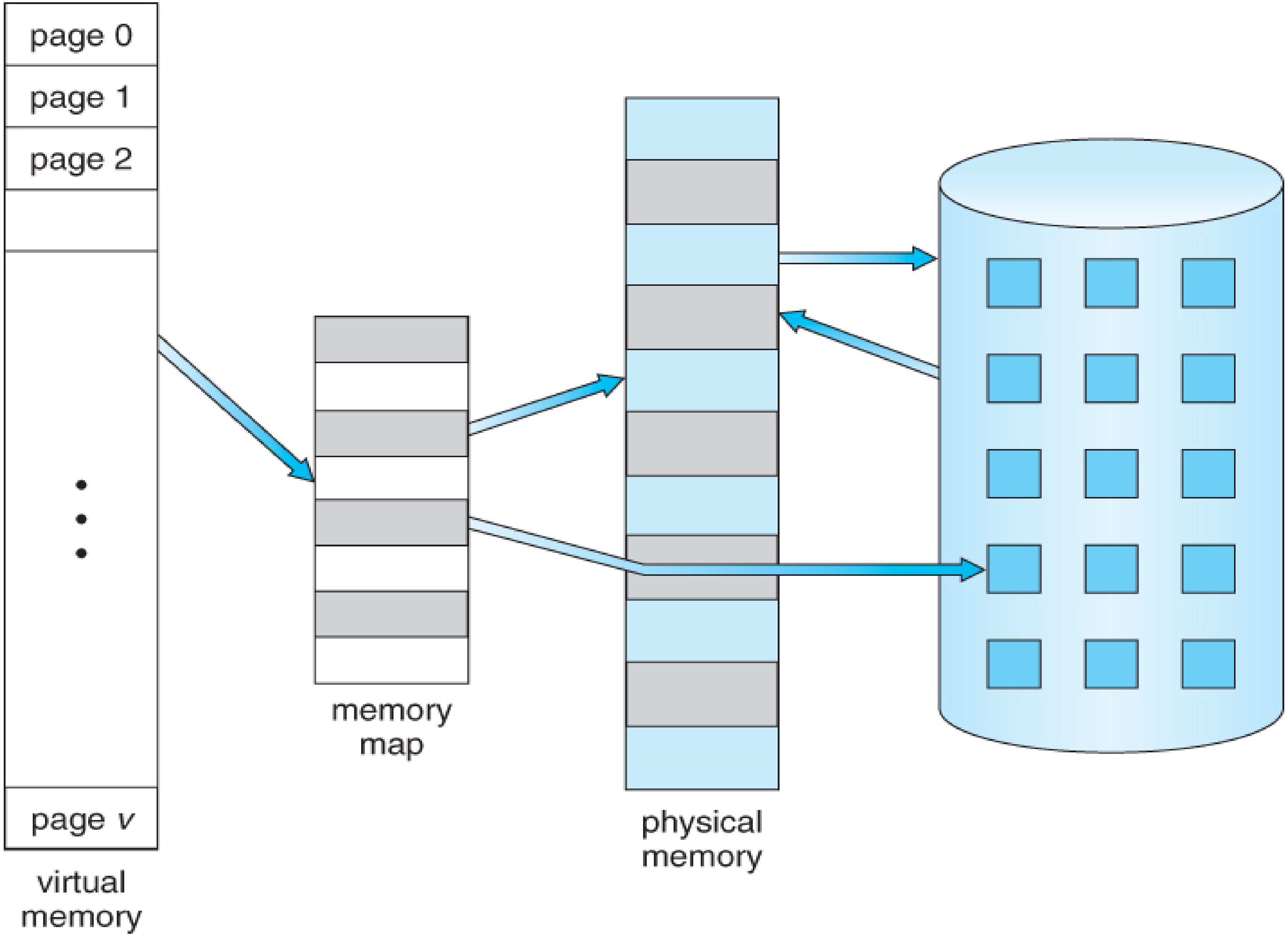
VMM: Virtual Machine Monitor

Virtual Memory and Virtual Machines

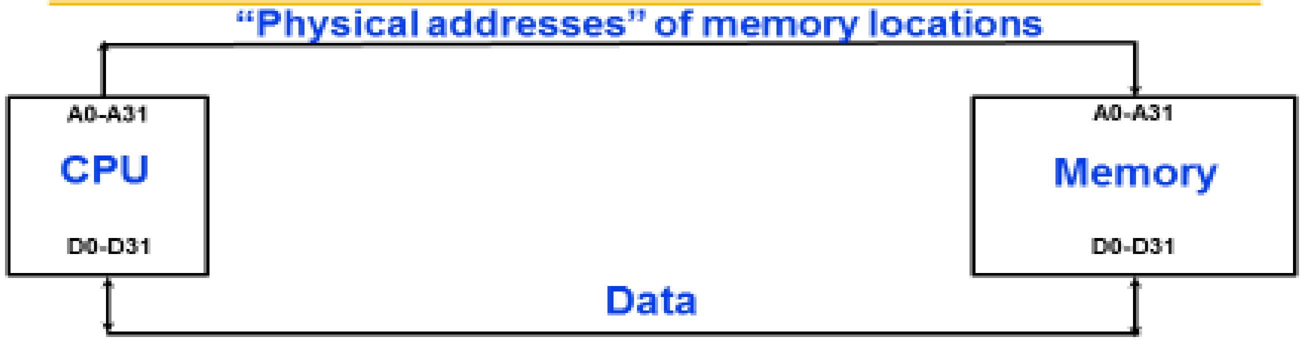
Virtual memory

- Virtual memory is a common part of operating system on desktop computers.
- The term virtual memory refers to something which appears to be present but actually it is not.
- The virtual memory technique allows users to use more memory for a program than the real memory of a computer.



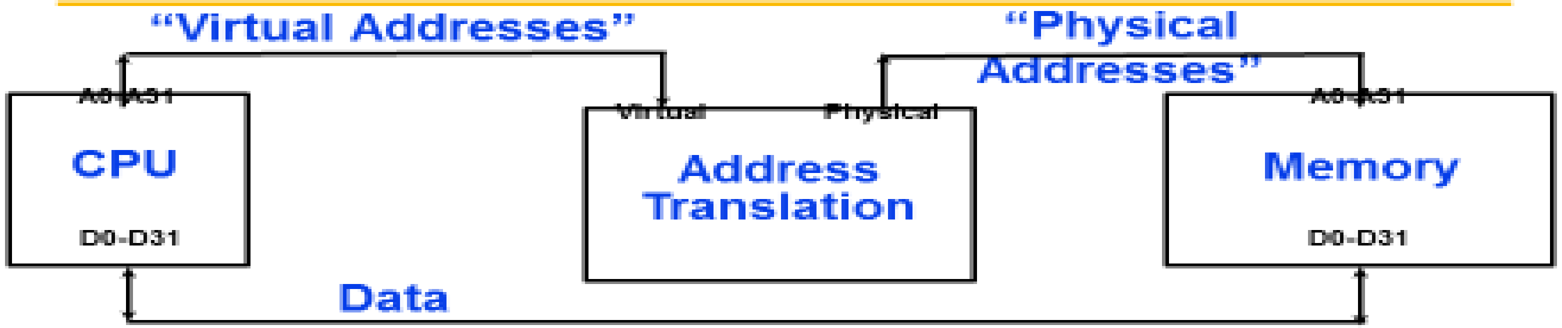

page 0
page 1
page 2
page v
virtual memory
memory map
physical memory

The Limits of Physical Addressing

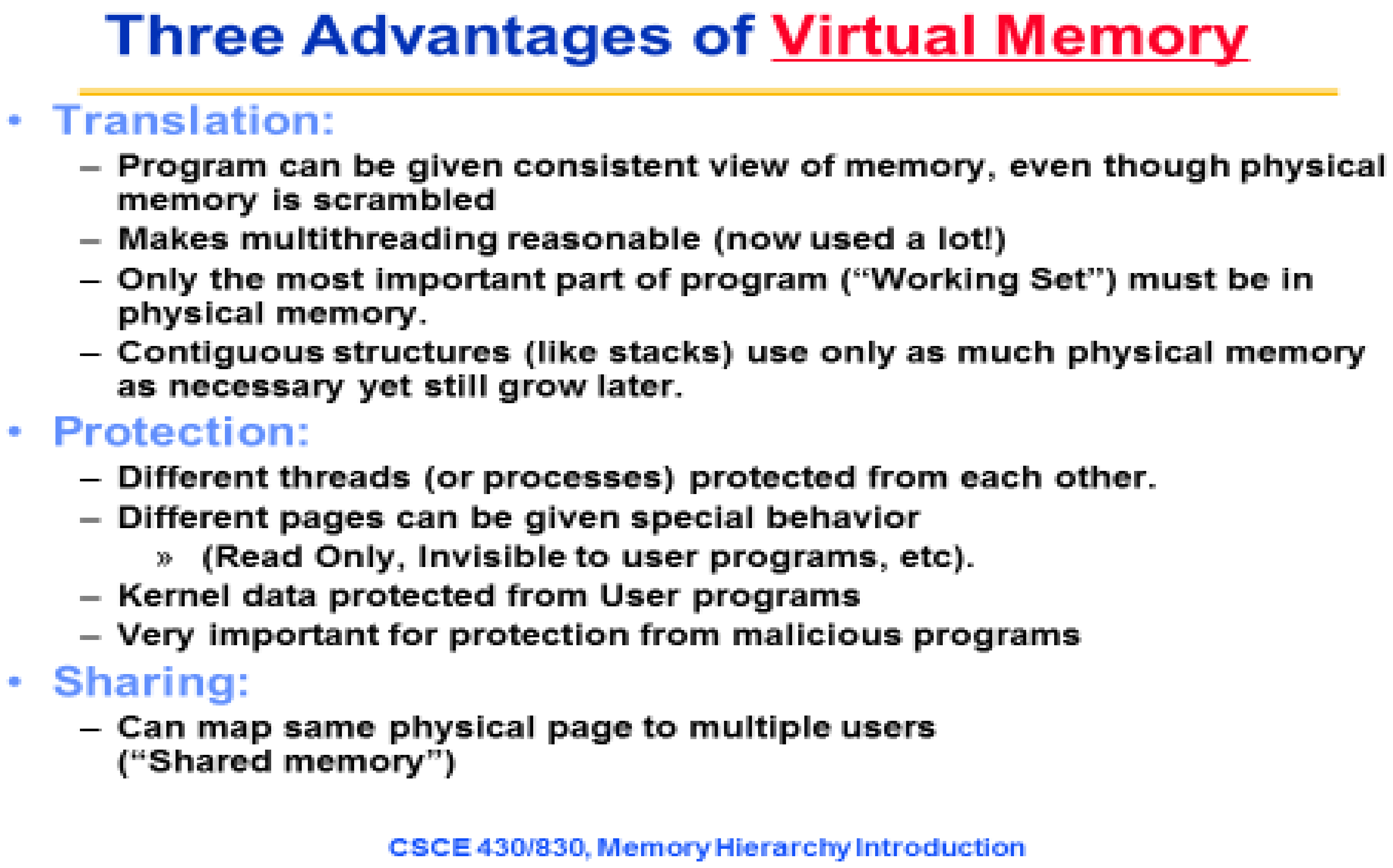


- ○All programs share one address space: The physical address space
- ○Machine language programs must be aware of the machine organization
- ○No way to prevent a program from accessing any machine resource

Solution: Add a Layer of Indirection



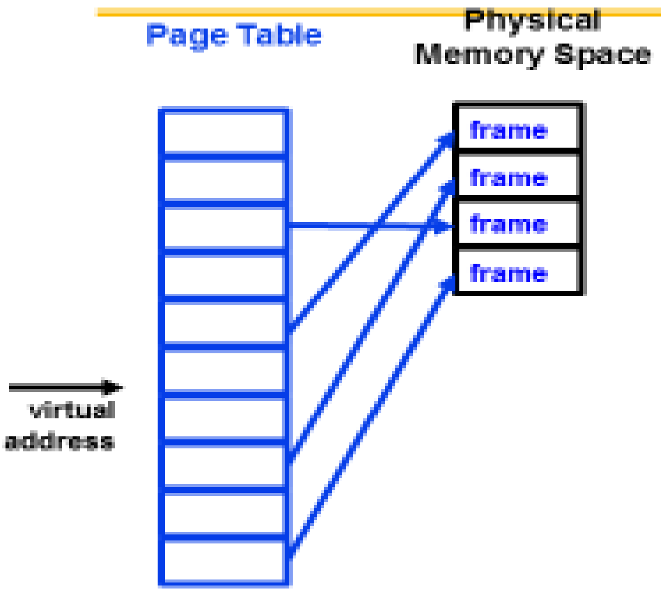
- User programs run in a standardized virtual address space
- Address Translation hardware, managed by the operating system (OS), maps virtual address to physical memory
- Hardware supports "modern" OS features: Protection, Translation, Sharing



Three Advantages of Virtual Memory

- Translation:
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program (“Working Set”) must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
- Protection:
  - Different threads (or processes) protected from each other.
  - Different pages can be given special behavior
    - » (Read Only, Invisible to user programs, etc).
  - Kernel data protected from User programs
  - Very important for protection from malicious programs
- Sharing:
  - Can map same physical page to multiple users (“Shared memory”)

Page tables encode virtual address spaces



A virtual address space is divided into blocks of memory called pages

A machine usually supports pages of a few sizes (MIPS R4000):

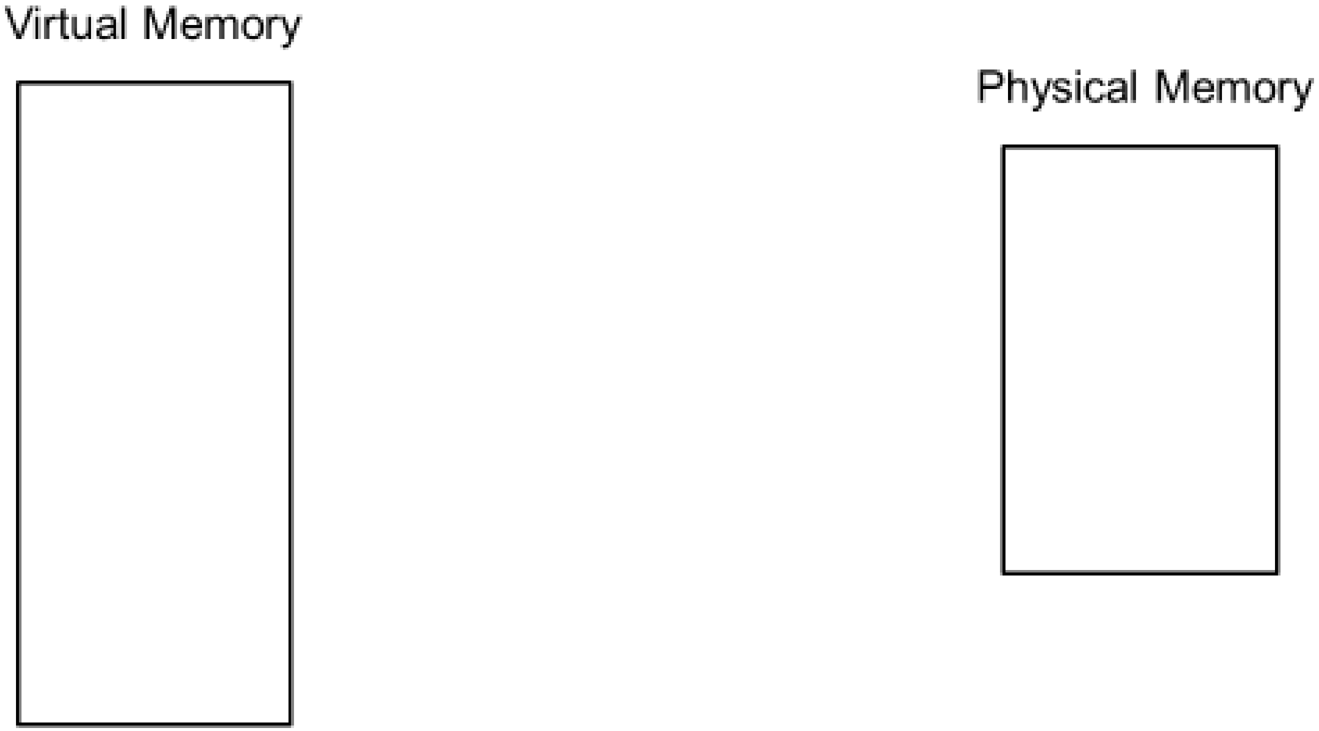
Page Size
4 Kbytes
16 Kbytes
64 Kbytes
256 Kbytes
1 Mbyte
4 Mbytes
16 Mbytes

A page table is indexed by a virtual address

OS manages the page table for each ASID (Addr. Space ID)

A valid page table entry codes physical memory "frame" address for the page

An Example of Page Table



Virtual Memory

Physical Memory

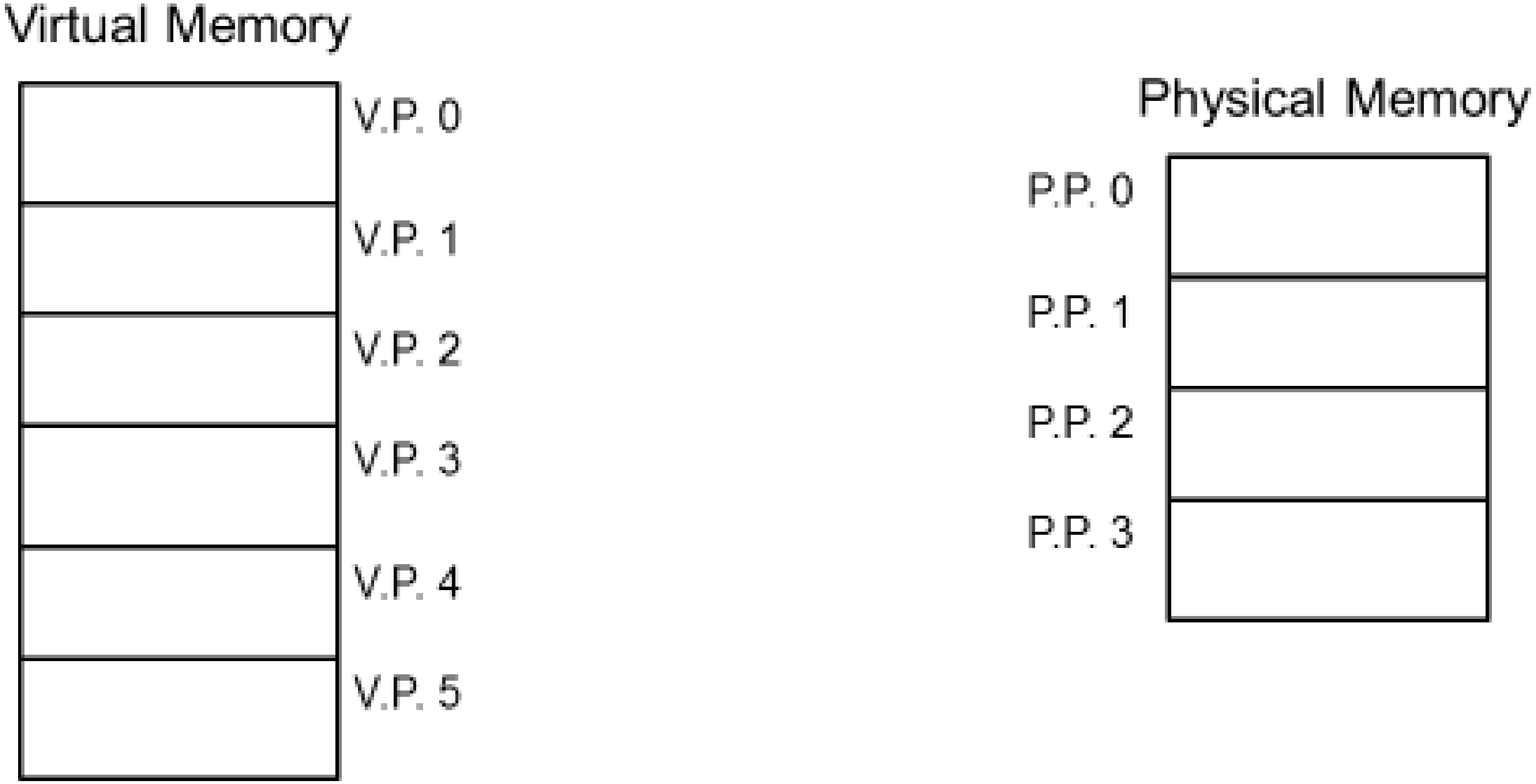
Dividing the address space by a page size

Virtual Memory

Physical Memory

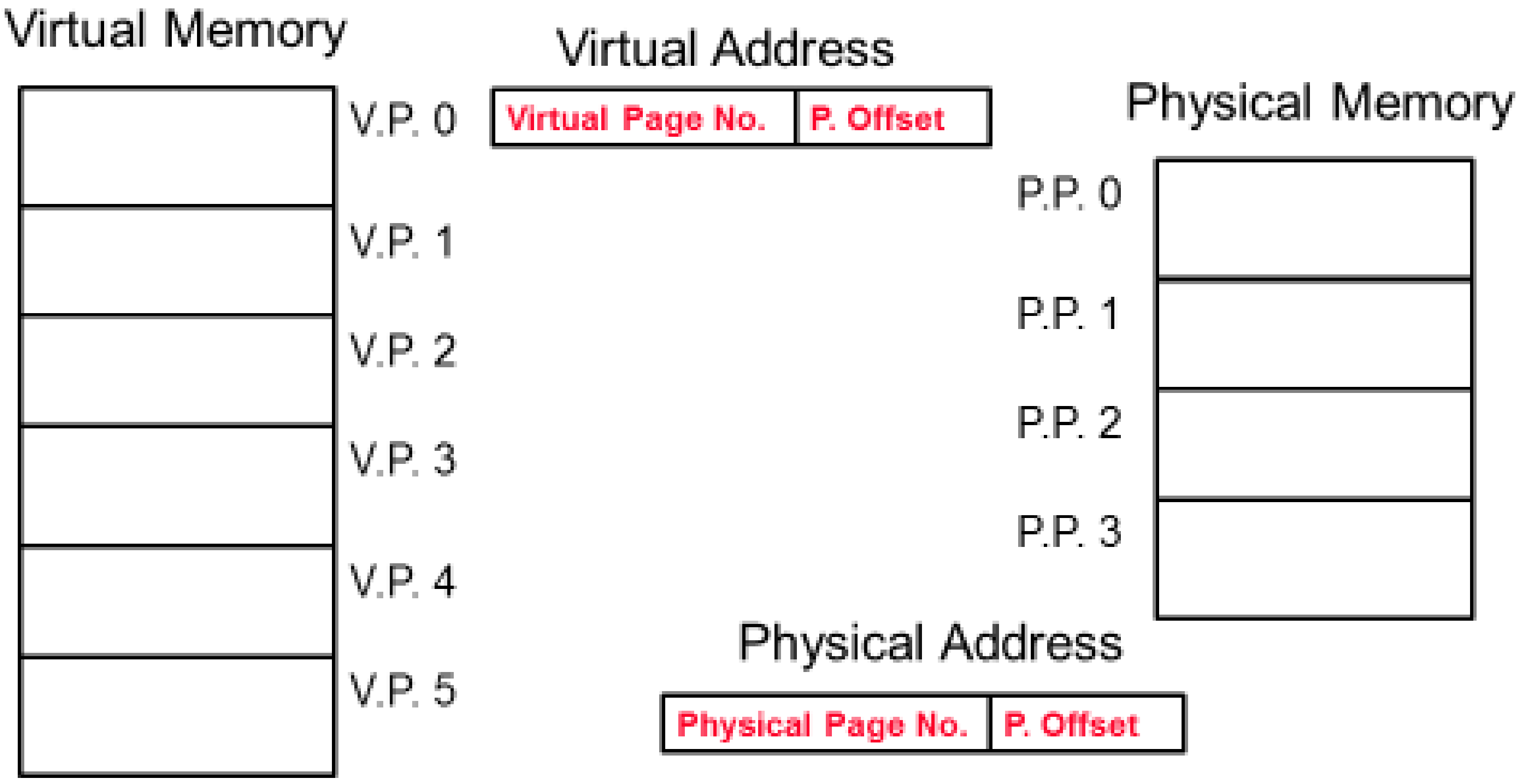
Page size:4KB

Virtual Page & Physical Page



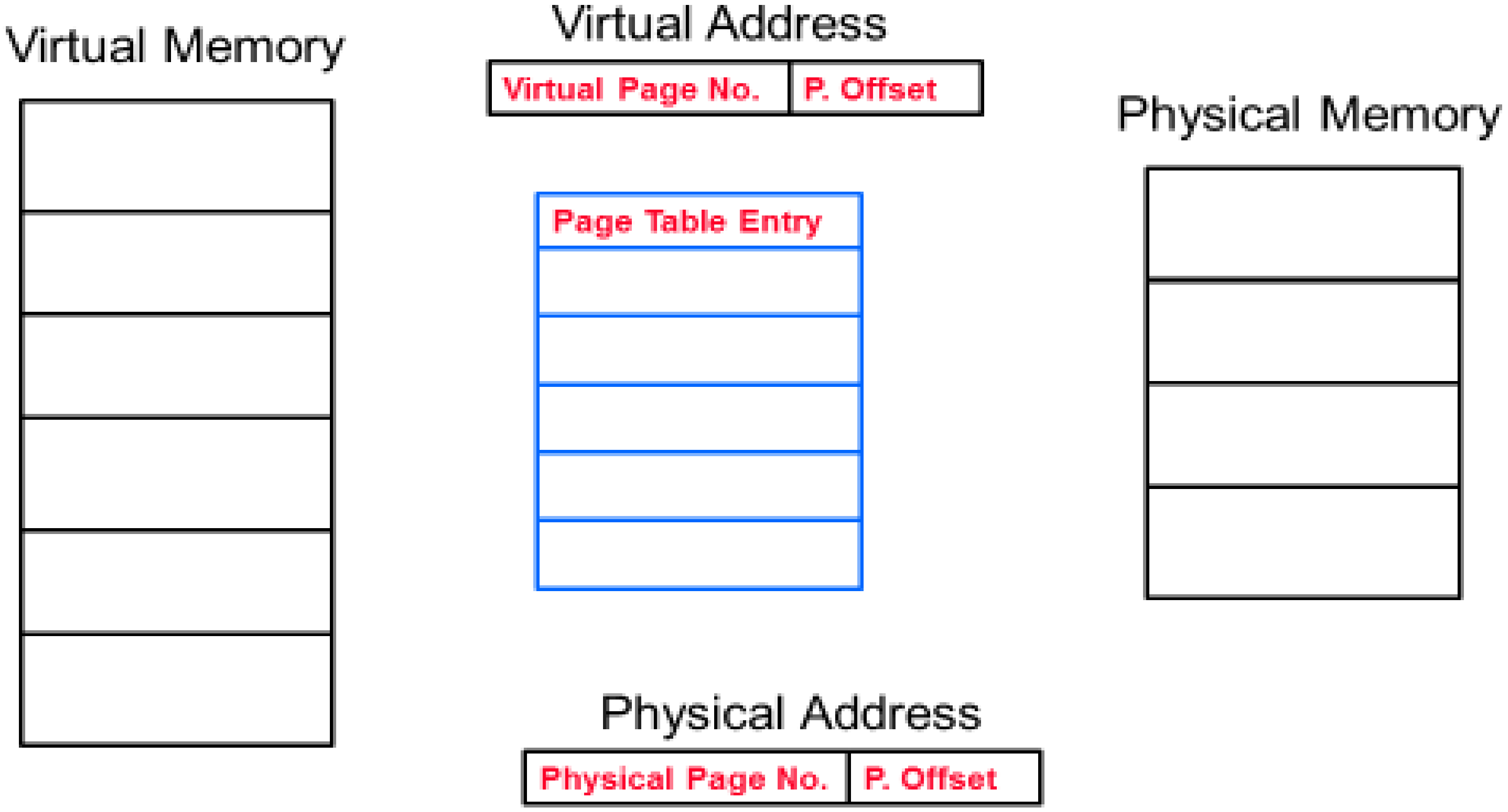
Page size:4KB

Addressing



Page size:4KB

Addressing



Virtual Memory

Virtual Address

Virtual Page No. | P. Offset

Page Table Entry

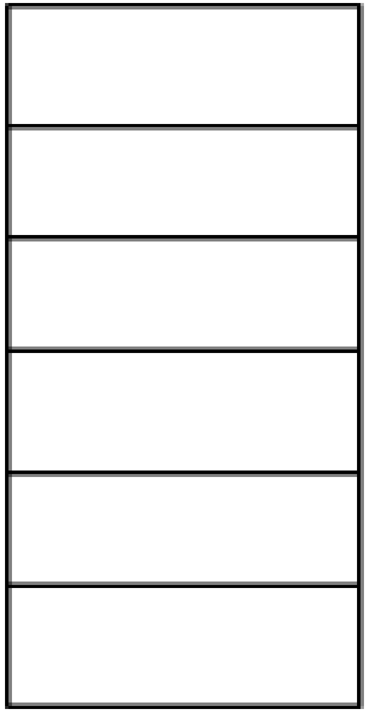
Physical Memory

Physical Address

Physical Page No. | P. Offset

Addressing

Virtual Memory



Virtual Address

Virtual Page No.	P. Offset

Page Table Entry

V	P	R	D	Physical Page No.

Valid/Present Bit

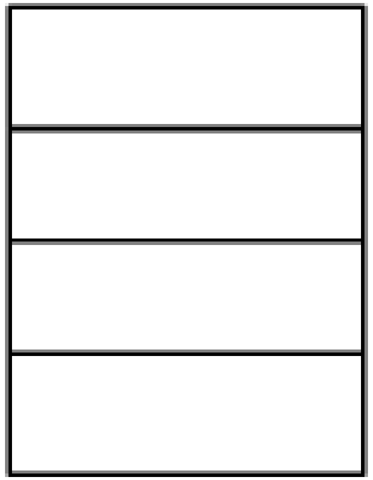
If set, page being pointed is resident in memory

Otherwise, on disk or not allocated

Physical Address

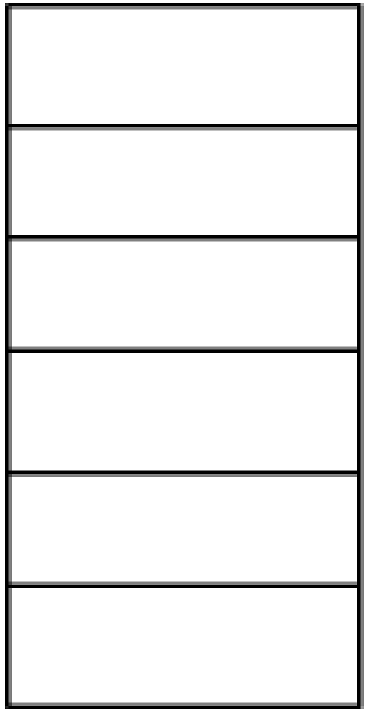
Physical Page No.	P. Offset

Physical Memory



Addressing

Virtual Memory



Virtual Address

Virtual Page No.	P. Offset

Page Table Entry

V	P	R	D	Physical Page No.

Valid/Present Bit

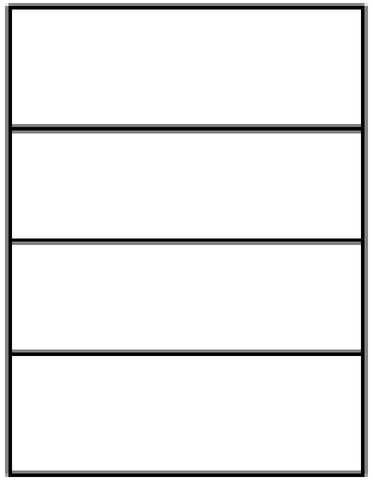
If set, page being pointed is resident in memory

Otherwise, on disk or not allocated

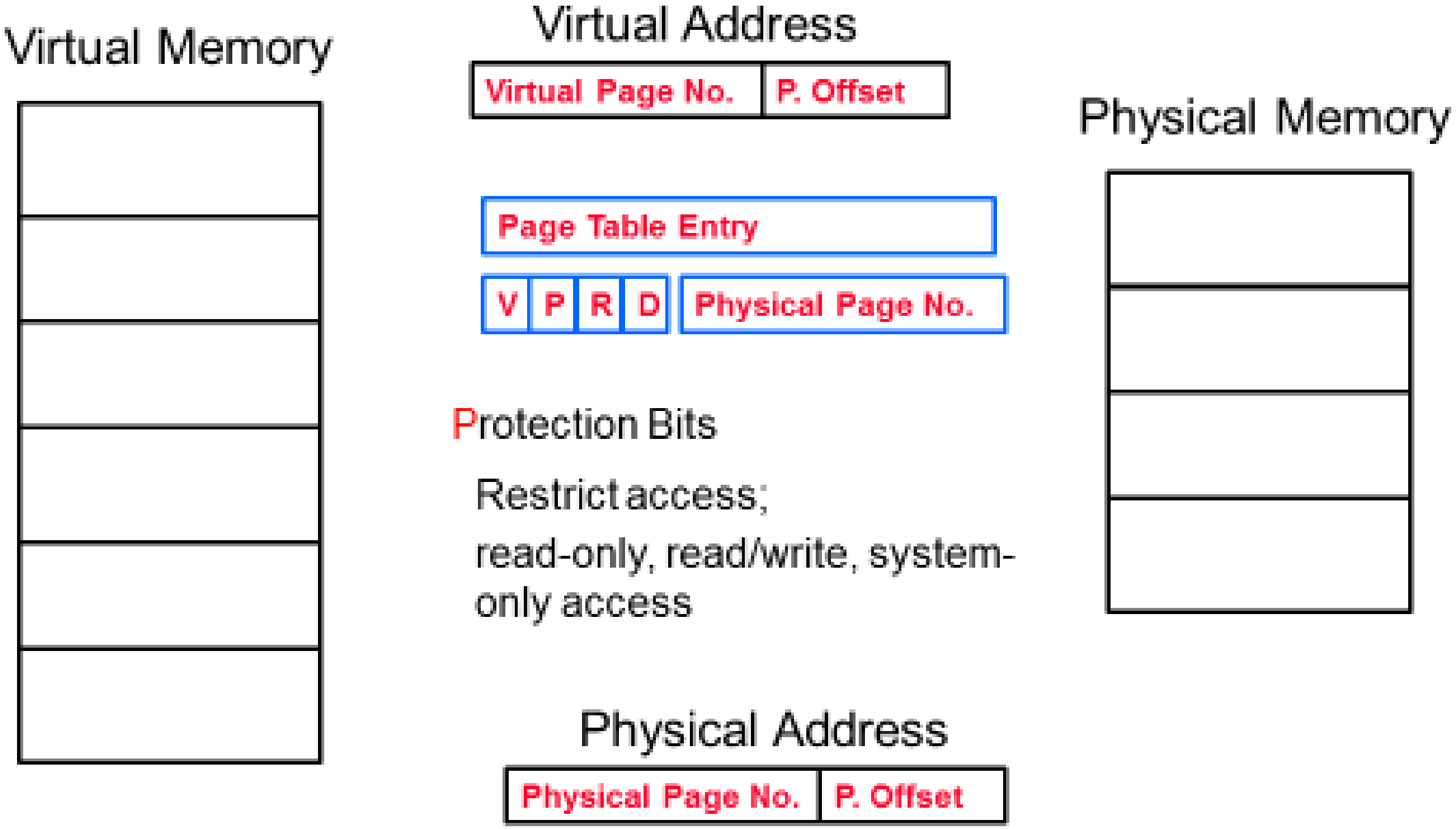
Physical Address

Physical Page No.	P. Offset

Physical Memory



Addressing



Virtual Memory

Virtual Address

Virtual Page No.	P. Offset

Page Table Entry

V	P	R	D	Physical Page No.

Physical Memory

Protection Bits

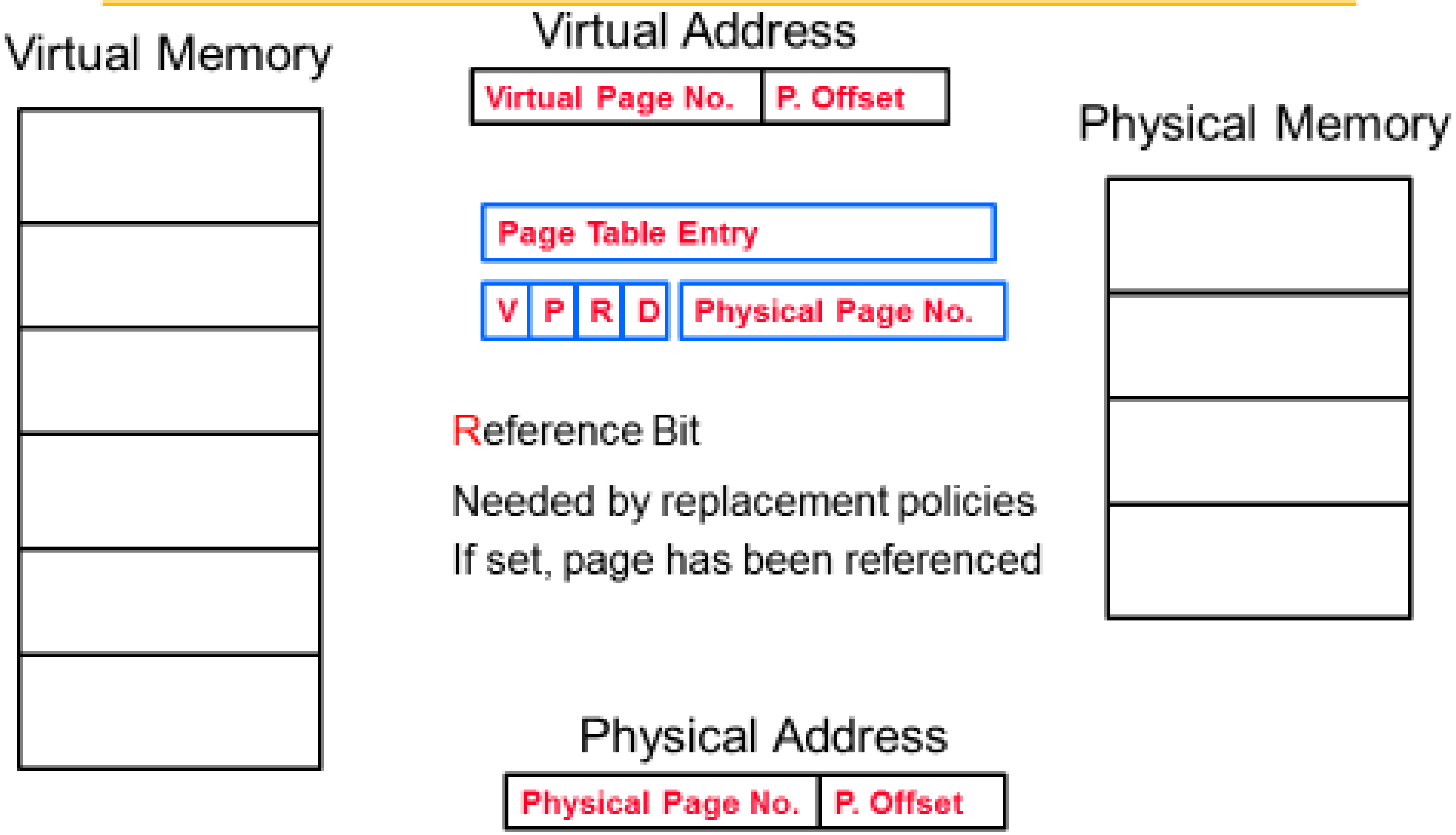
Restrict access;

read-only, read/write, system-only access

Physical Address

Physical Page No.	P. Offset

Addressing



Virtual Memory

Virtual Address

Virtual Page No.	P. Offset

Physical Memory

Page Table Entry

V	P	R	D	Physical Page No.

Reference Bit

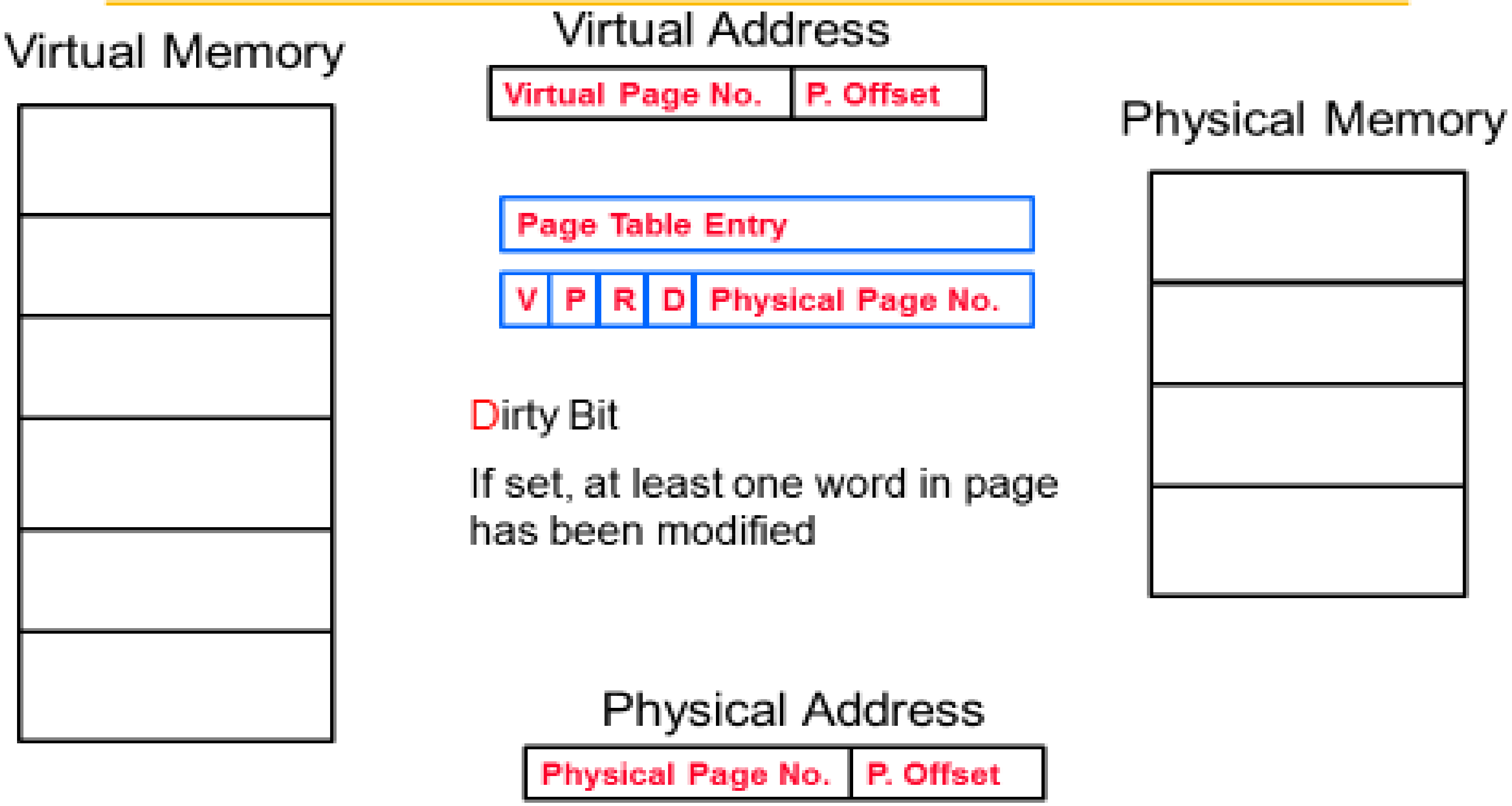
Needed by replacement policies

If set, page has been referenced

Physical Address

Physical Page No.	P. Offset

Page Table Entry



Virtual Memory

Virtual Address

Virtual Page No.	P. Offset

Physical Memory

Page Table Entry

V	P	R	D	Physical Page No.

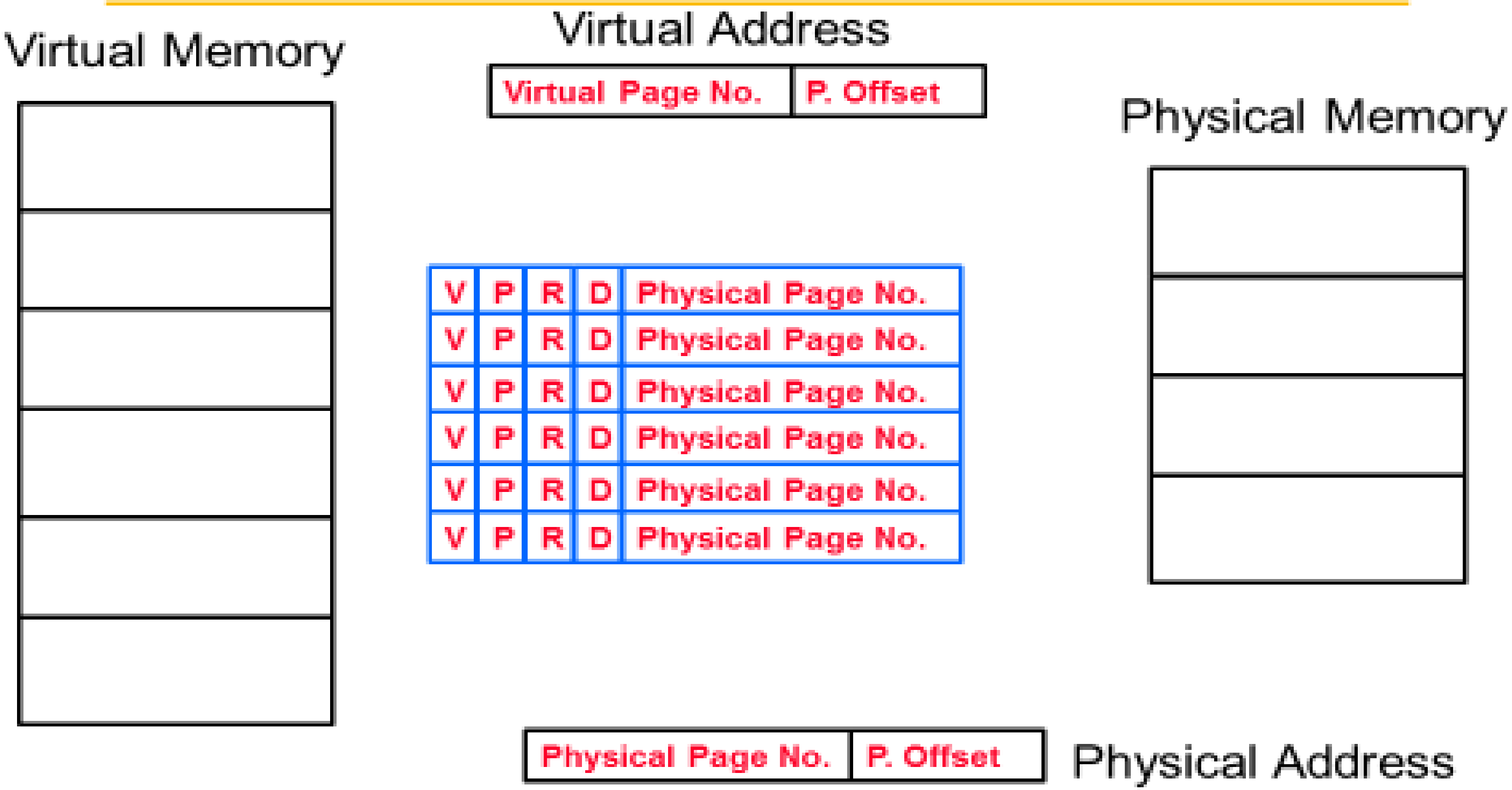
Dirty Bit

If set, at least one word in page has been modified

Physical Address

Physical Page No.	P. Offset

Page Table Entry



Virtual Memory

Virtual Address

Virtual Page No.	P. Offset

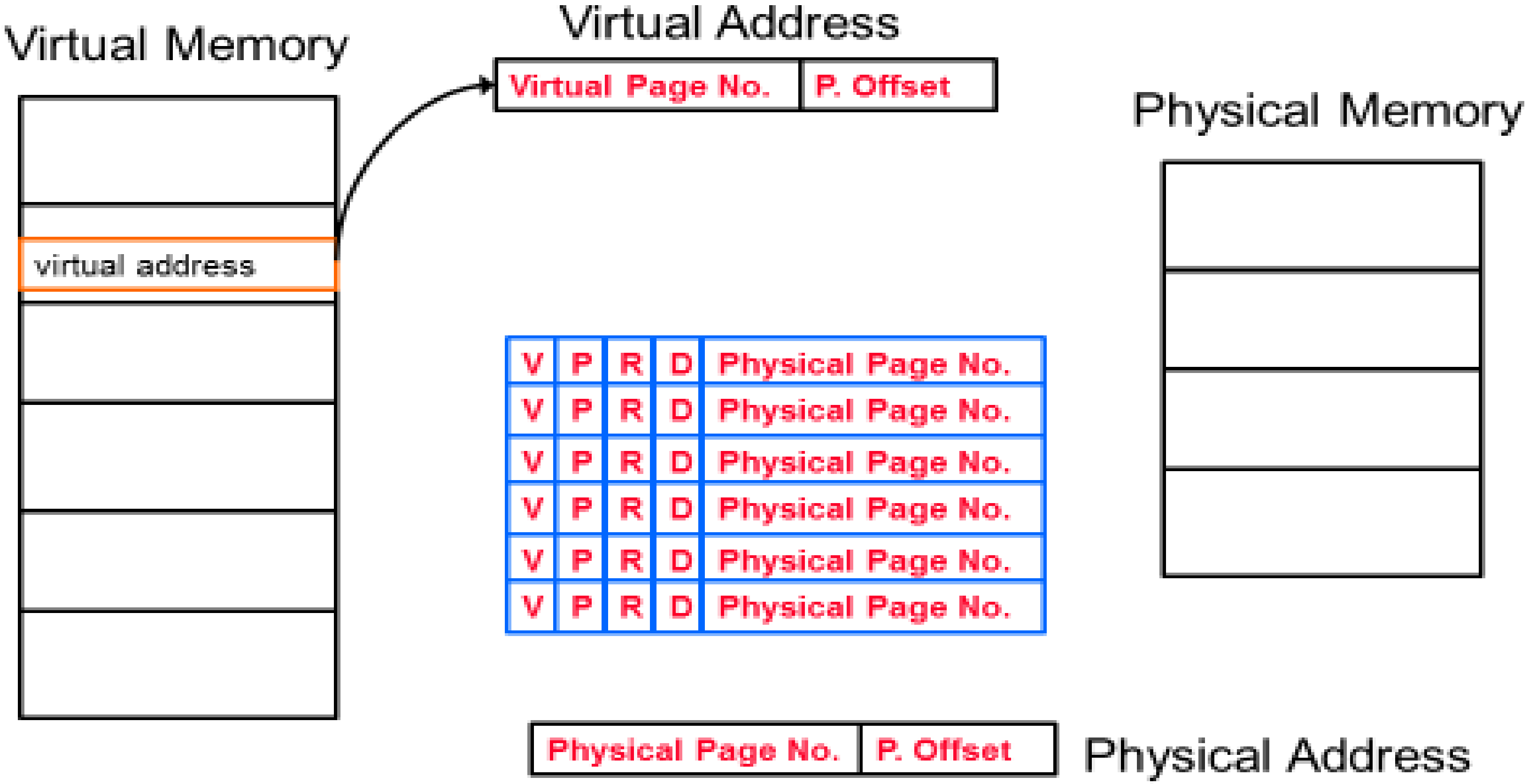
Physical Memory

V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.

Physical Page No.	P. Offset

Physical Address

Page Table Lookup



Virtual Memory

virtual address

Virtual Address

Virtual Page No.	P. Offset

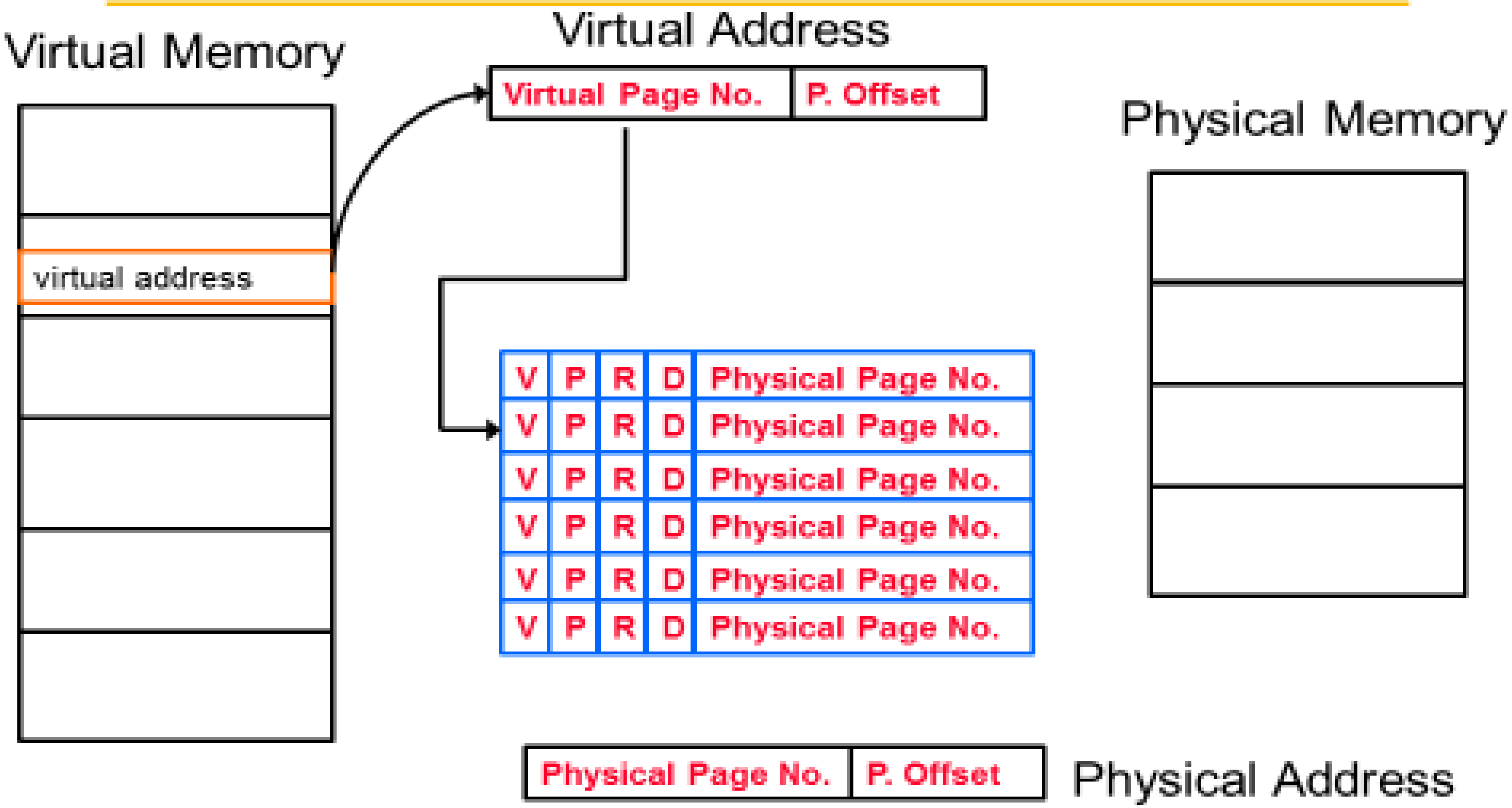
Physical Memory

V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.

Physical Page No.	P. Offset

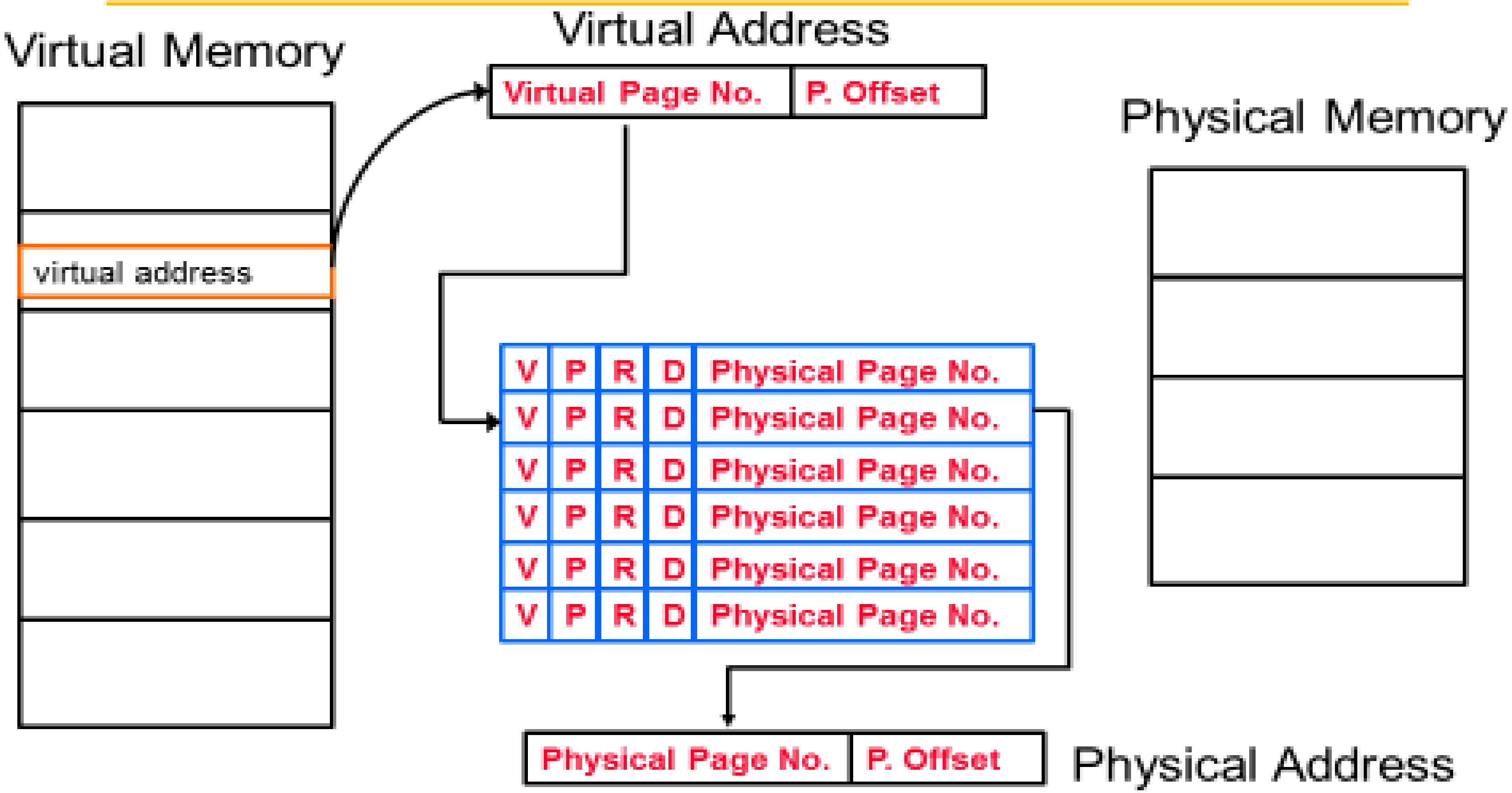
Physical Address

Page Table Lookup



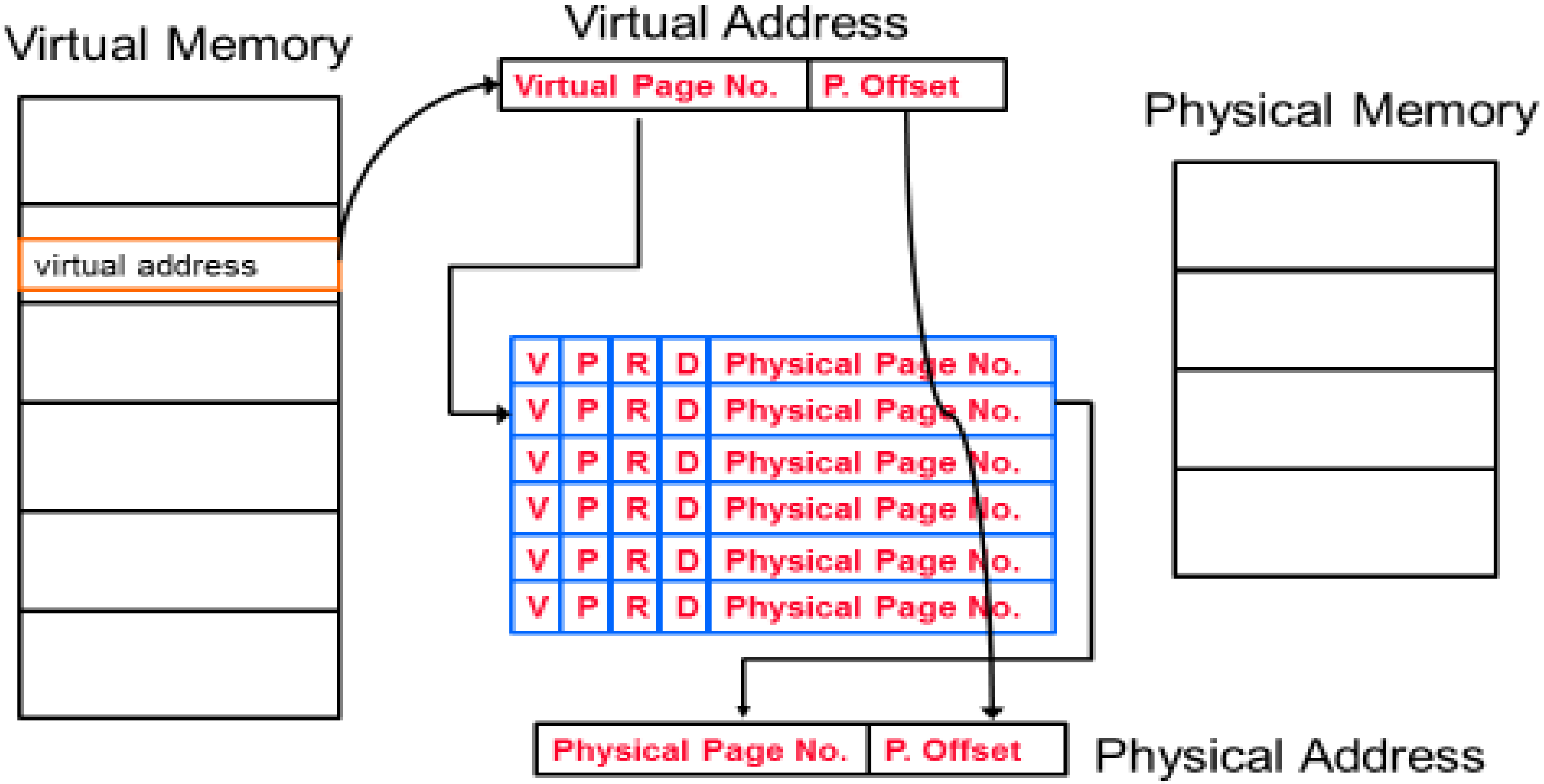
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.

Page Table Lookup

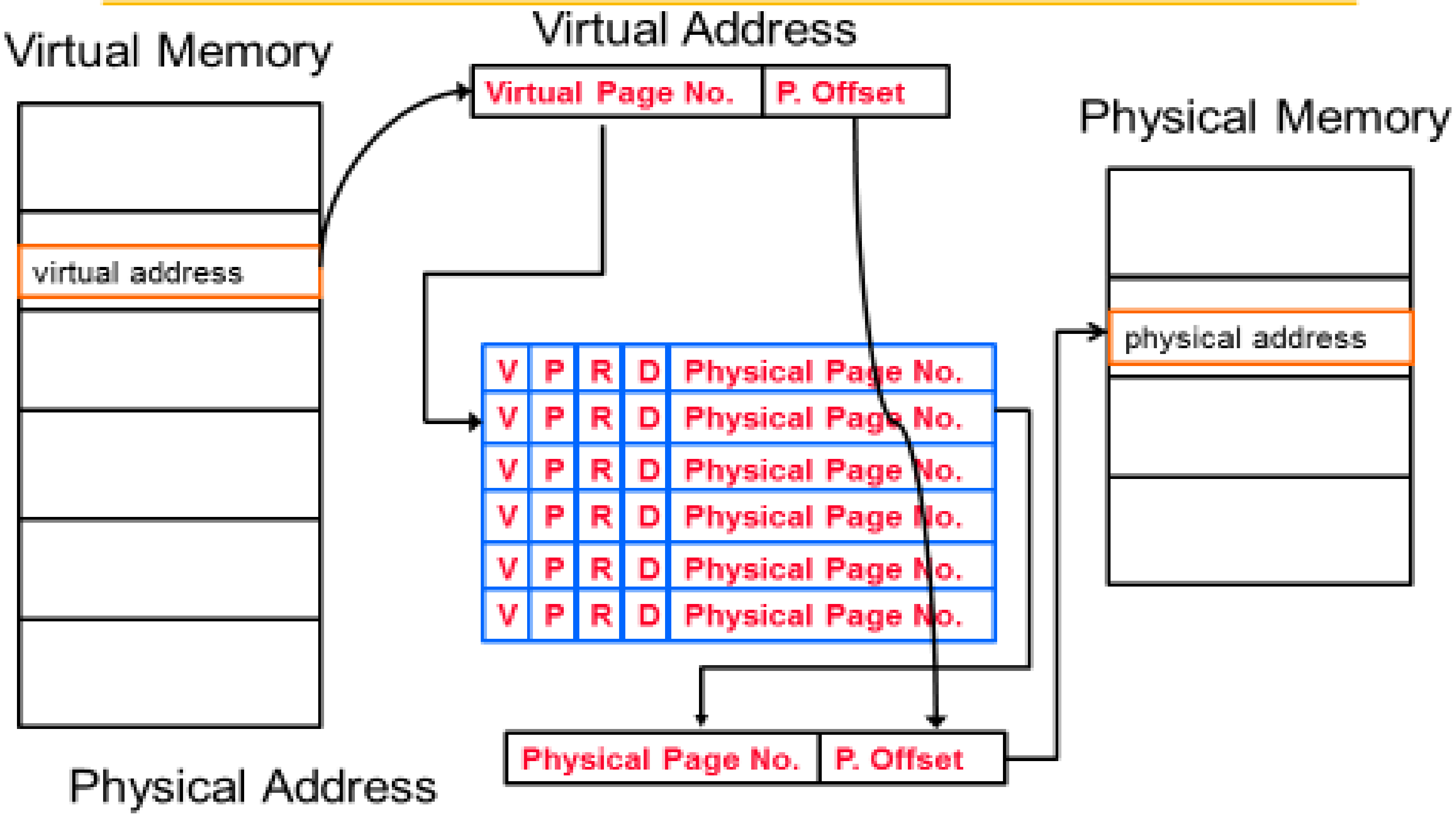


V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.
V	P	R	D	Physical Page No.

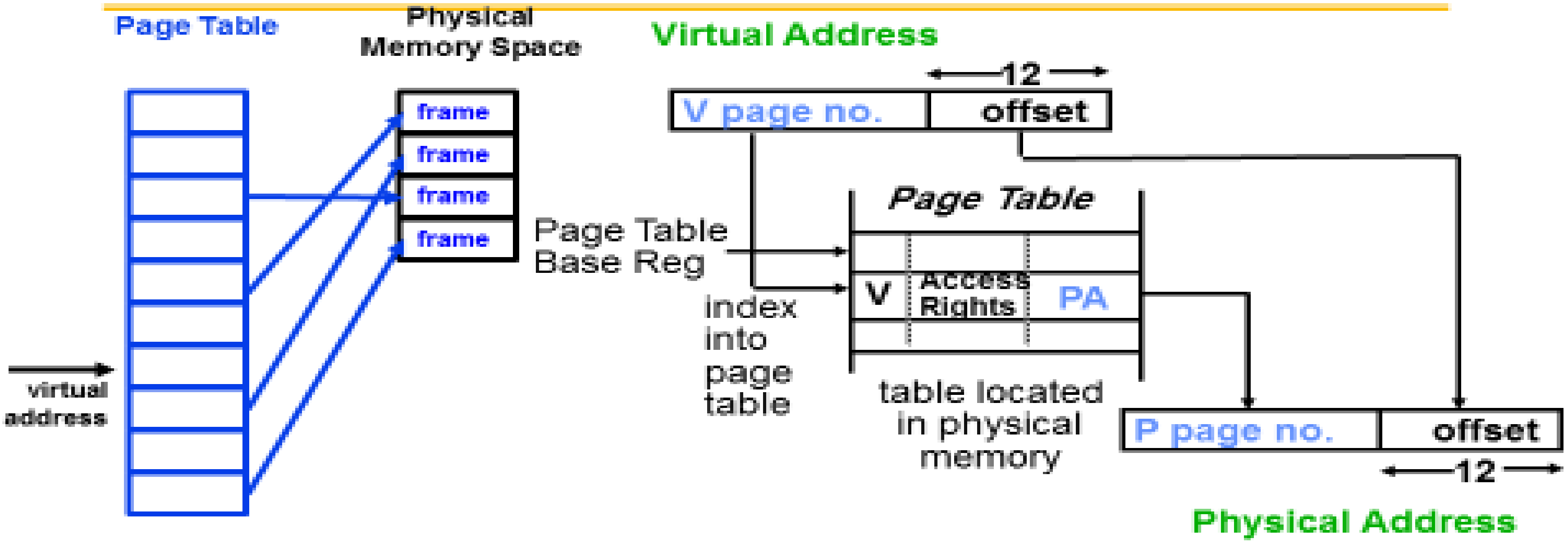
Page Table Lookup



Page Table Lookup



Details of Page Table



- Page table maps virtual page numbers to physical frames (“PTE” = Page Table Entry)
- Virtual memory => treat memory ≈ cache for disk
- 4 fundamental questions: placement, identification, replacement, and write policy?

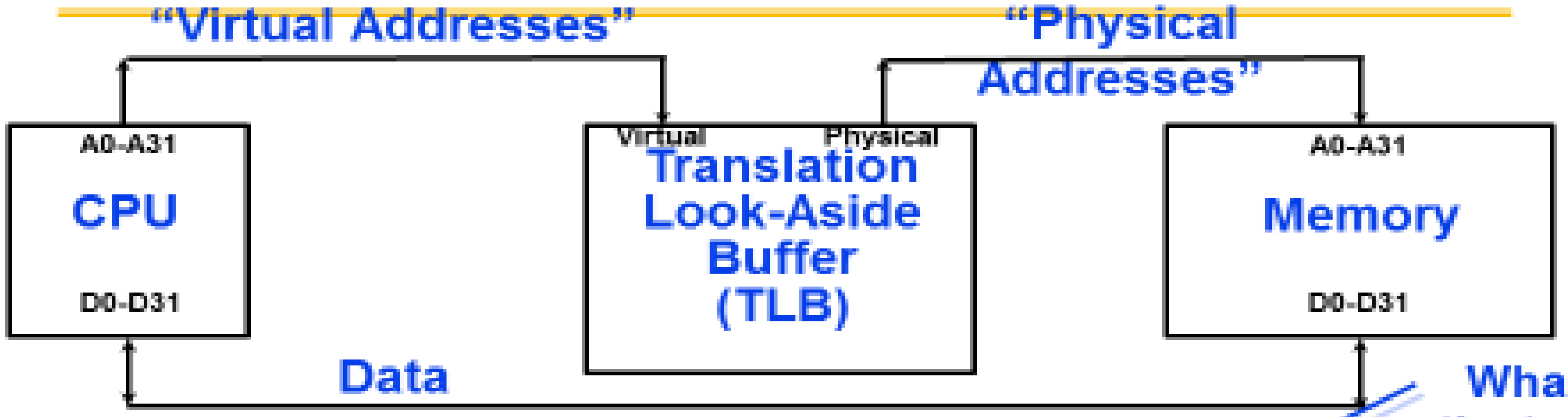
4 Fundamental Questions

- Placement
  - Operating systems allow blocks to be placed anywhere in main memory
- Identification
  - Page Table, Inverted Page Table
- Replacement
  - Almost all operating systems try to use LRU
- Write Policies
  - Always write back

Latency

- Since Page Table is located in main memory, it takes one memory access latency to finish an address translation;
- As a result, a load/store operation from/to main memory needs two memory access latency in total;
- Considering the expensive memory access latency, the overhead of page table lookup should be optimized;
- How?
  - Principle of Locality
  - Caching

MIPS Address Translation: How does it work?



Translation Look-Aside Buffer (TLB)

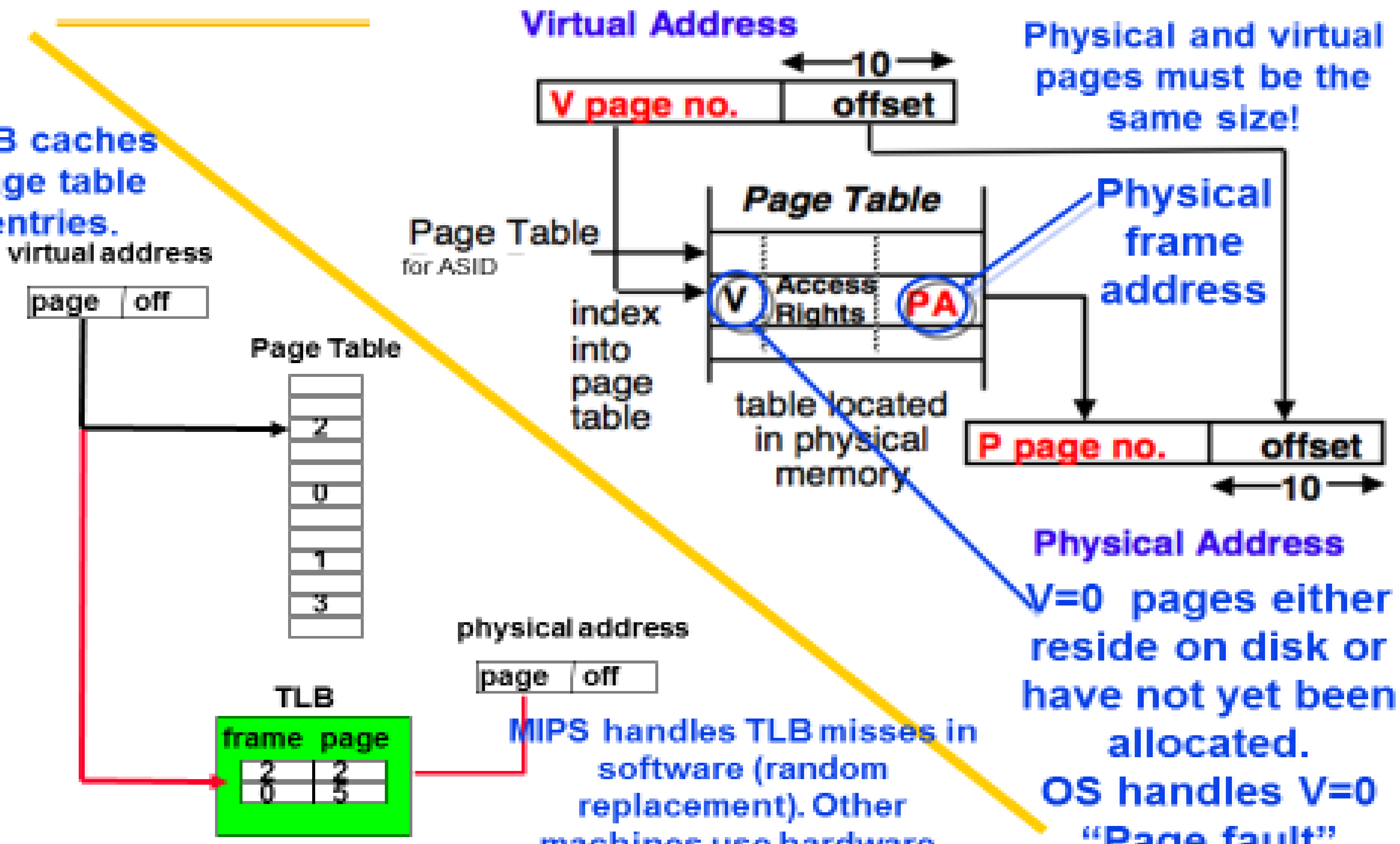
A small fully-associative cache of mappings from virtual to physical addresses

What is the table of mappings that it caches?

TLB also contains protection bits for virtual address

Fast common case: Virtual address is in TLB, process has permission to read/write it.

The TLB caches page table entries



TLB caches page table entries.

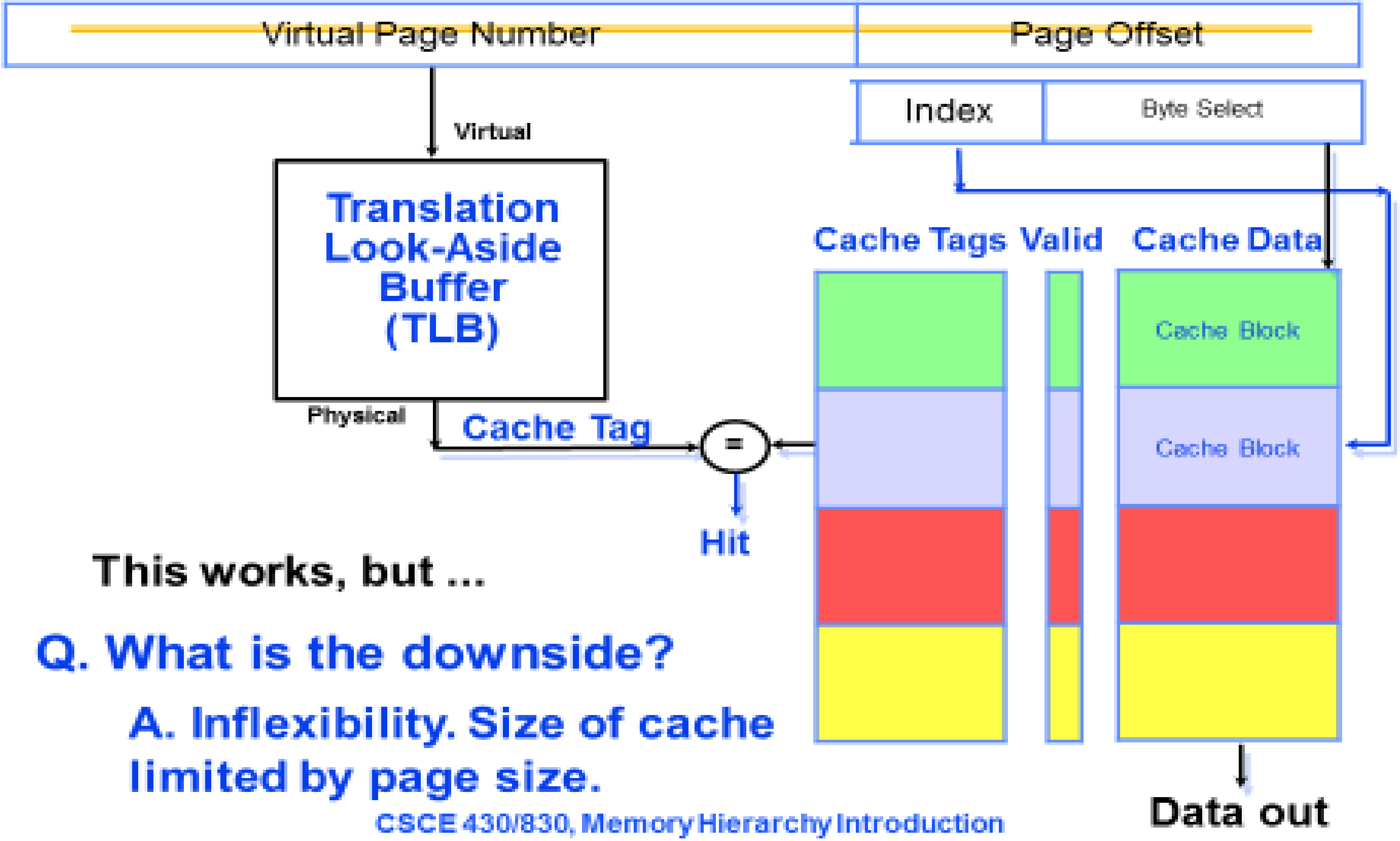
MIPS handles TLB misses in software (random replacement). Other machines use hardware.

Physical and virtual pages must be the same size!

Physical frame address

V=0 pages either reside on disk or have not yet been allocated.
OS handles V=0 "Page fault"

Can TLB and caching be overlapped?



This works, but ...

Q. What is the downside?

A. Inflexibility. Size of cache limited by page size.

VA: 64bits

PA: 40bits

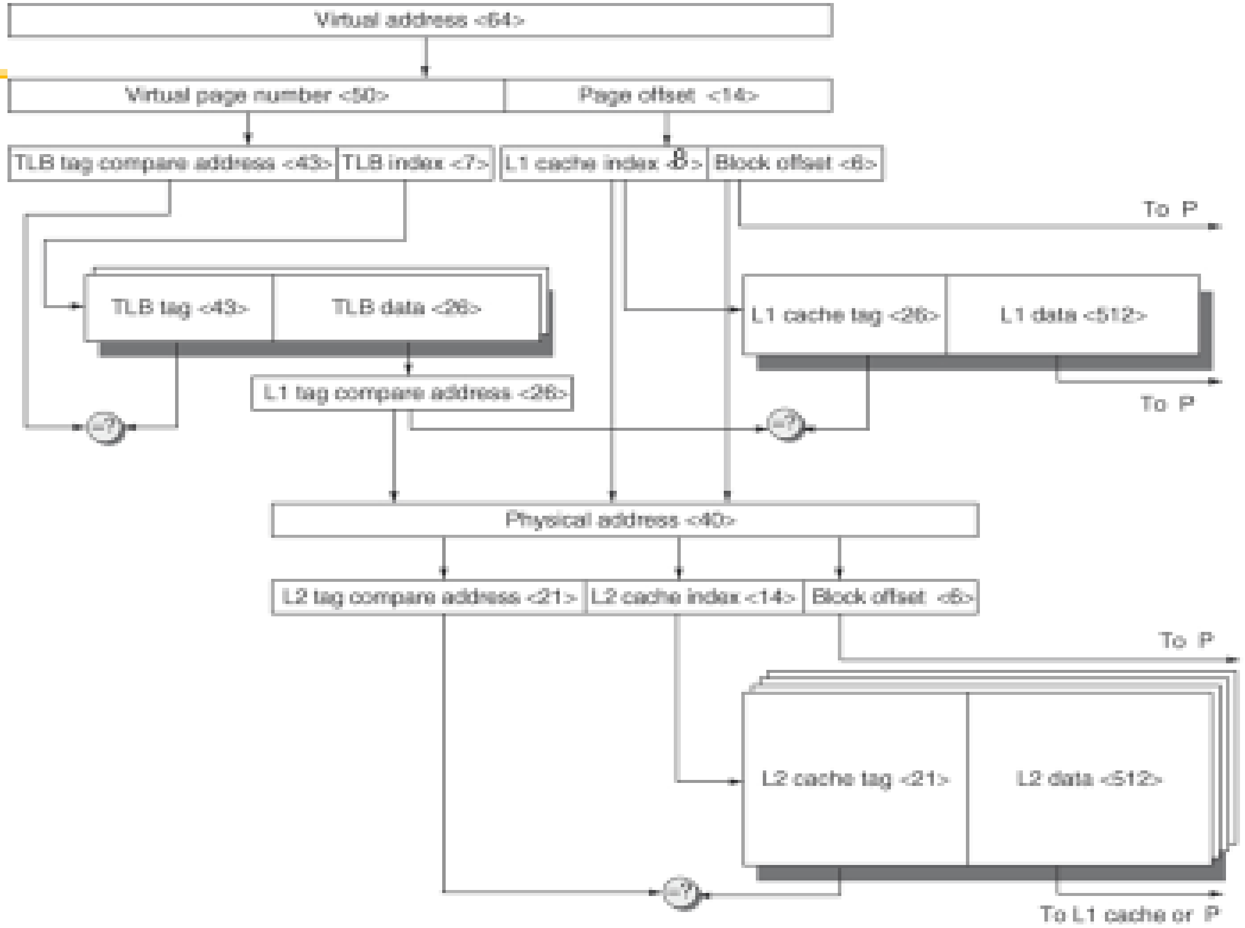
Page size: 16KB

TLB: 2-way set associative, 256 entries

Cache block: 64B

L1: direct-mapping, 16KB

L2: 4-way set associative, 4MB


Virtual address <64>
Virtual page number <50>
Page offset <14>
TLB tag compare address <43>
TLB index <7>
L1 cache index <8>
Block offset <6>
To P
TLB tag <43>
TLB data <26>
L1 cache tag <26>
L1 data <512>
L1 tag compare address <26>
=?
=?
To P
Physical address <40>
L2 tag compare address <21>
L2 cache index <14>
Block offset <6>
To P
L2 cache tag <21>
L2 data <512>
=?
To L1 cache or P