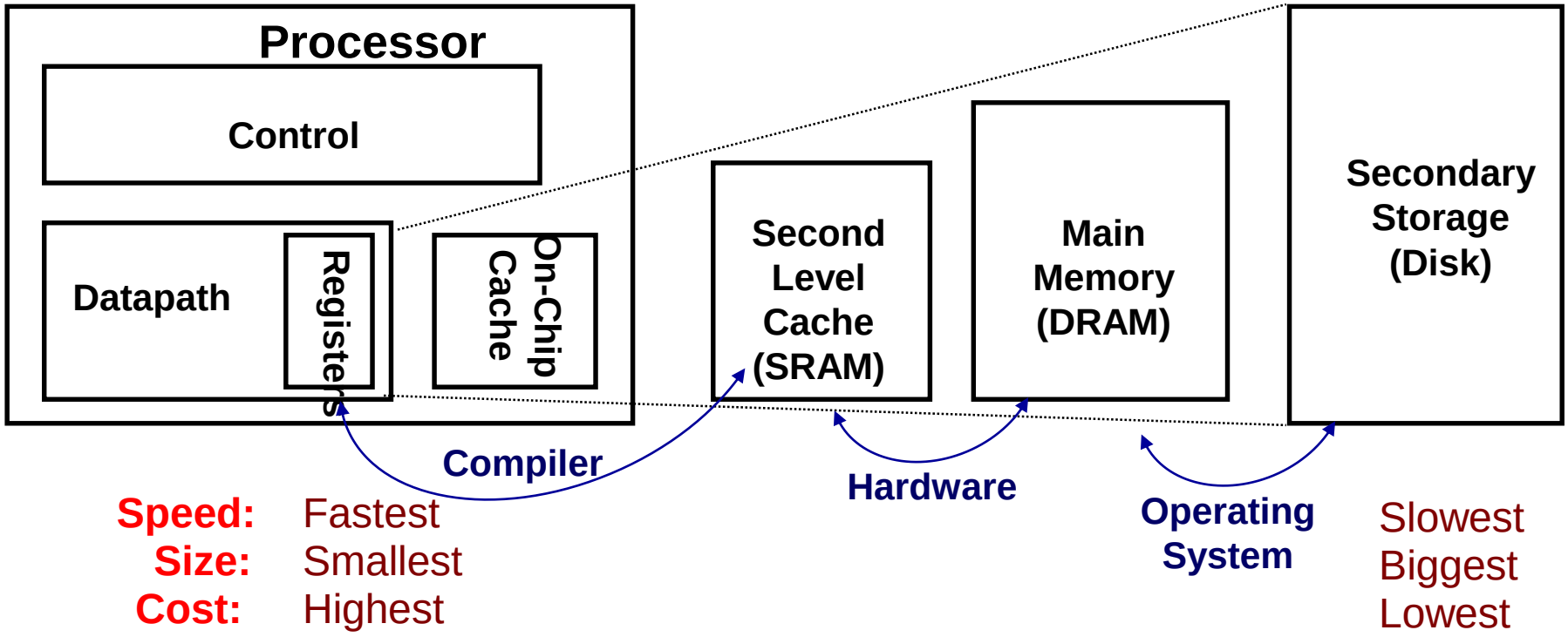
Lec13
Memory Technology

Memory Hierarchy

- The goal: To provide a memory system with cost most almost as low as the cheapest level of memory and speed almost as fast as the fastest level.



Main Memory Performance

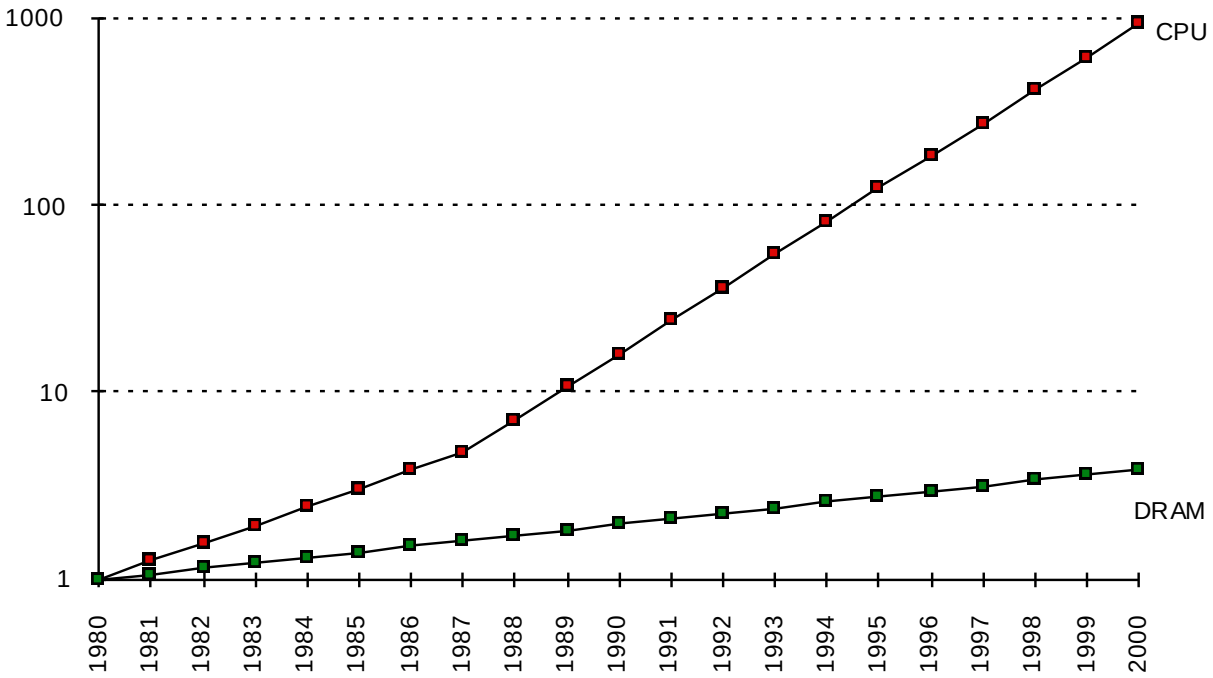
- Main memory---- the next level down in the hierarchy.
  - Main memory is usually made from DRAM while caches use SRAM.
- Performance of main memory
  - *Latency*
    - harder to reduce latency ; *Important for caches.*
  - *Bandwidth*
    - easier to improve bandwidth with new organizations
    - *Important for I/O.*
    - *Also for cache with second-level and larger block sizes.*
- The previous sections describe cache organization to reduce CPU-DRAM performance gap
- This section we analyze techniques for organizing memory to improve bandwidth.

Why do I freaking care?

- By it's nature, DRAM isn't built for speed
  - Reponse times dependent on capacitive circuit properties which get worse as density increases
- DRAM process isn't easy to integrate into CMOS process
  - DRAM is off chip
  - Connectors, wires, etc introduce slowness
  - IRAM(Internal RAM) efforts looking to integrating the two
- Memory Architectures are designed to minimize impact of DRAM latency
  - Low Level: Memory chips
  - High Level memory designs.
  - You will pay $$$$$$ and then some $$$ for a good memory system.

Why do I freaking care?

- 1960-1985: Speed = *f*(no. operations)
- 1990
  - Pipelined Execution & Fast Clock Rate
  - Out-of-Order execution
  - Superscalar Instruction Issue



- 1998: Speed = *f*(non-cached memory accesses)
- Superscalar, Out-of-Order machines hide L1 data cache miss (5 clocks) but not L2 cache miss (50 clocks)?

basic memory organization

First-level caches are often organized with a physical width of 1 word because most CPU accesses are that size

Assume

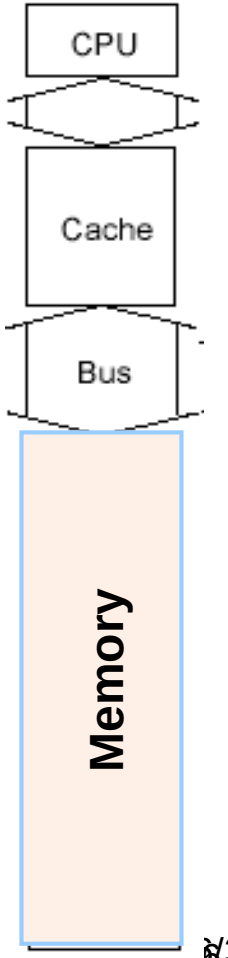
4 clock cycles to send the address

56 clock cycles for the access time per word

4 clock cycles to send a word of data

Block size is 4 words

Every word is 8 bytes



The miss penalty:

$4 \times (4+56+4) = 256$ CLKs

Bandwidth :

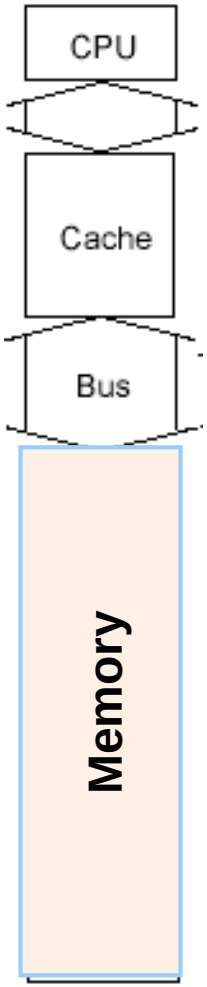
$$\frac{4 \times 8}{256} = \frac{1}{8}$$

1st Technique for Higher Bandwidth

Wider Main Memory

- Amdahl'law suggested that memory capacity should grow linearly with CPU speed.
- Retrospecting
  - Memory capacity grows four-fold every three years to supply this demand.
  - The CPU-DRAM performance gap is a problem, however, DRAM performance improvement is only about 5% per year.
- Wider Main Memory
  - Doubling or quadrupling the width of the cache and the memory
  - It will therefore double or quadruple the memory bandwidth.

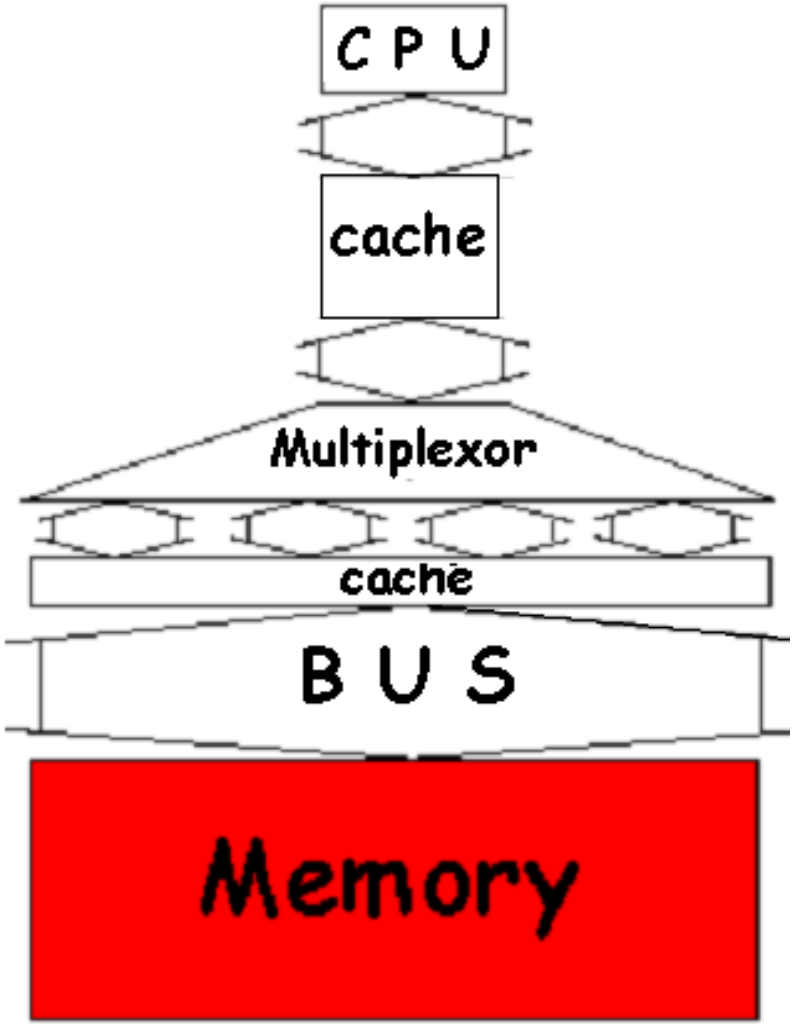
Many Words in Single bank



Widening memory:
Doubles /quadruples
Memory and Bandwidth

Disadvantages: bus

- MUX required on critical path to allow word access
- Increase minimum memory increment purchased by customer.
- Complicates error correction



Perf. in the Technique

- With a main memory width of 2 words

  The miss penalty: 4words/Block

  $2 \times (4+56+4) = 128$ CLKs

  Bandwidth :

  $$\frac{4 \times 8}{128} = \frac{1}{4}$$

- With a main memory width of 4 words

  The miss penalty: 4words/Block

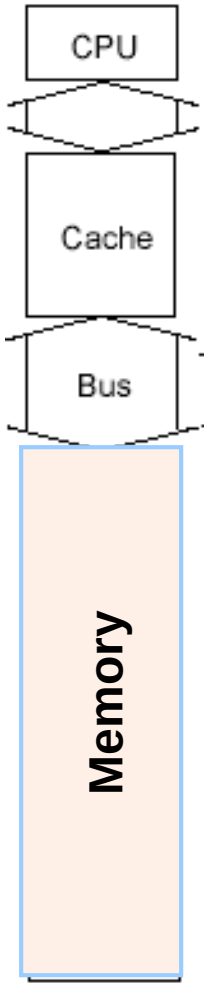
  $1 \times (4+56+4) = 64$ CLKs

  Bandwidth :

  $$\frac{4 \times 8}{64} = \frac{1}{2}$$

2nd Technique for Higher Bandwidth

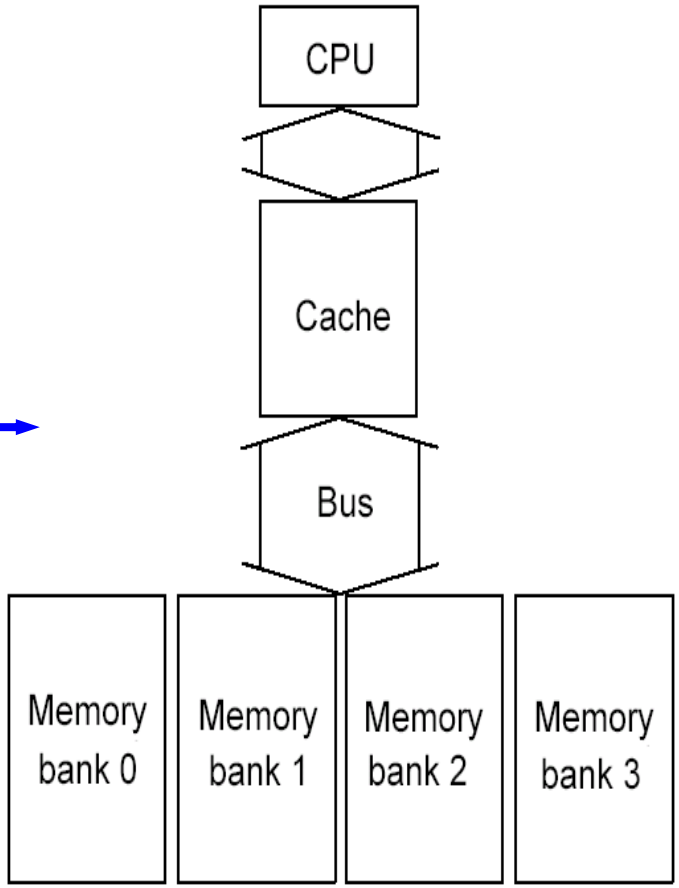
simple Interleaved Memory



Advantages:

Take advantage of the potential parallelism of having many chips memory system.

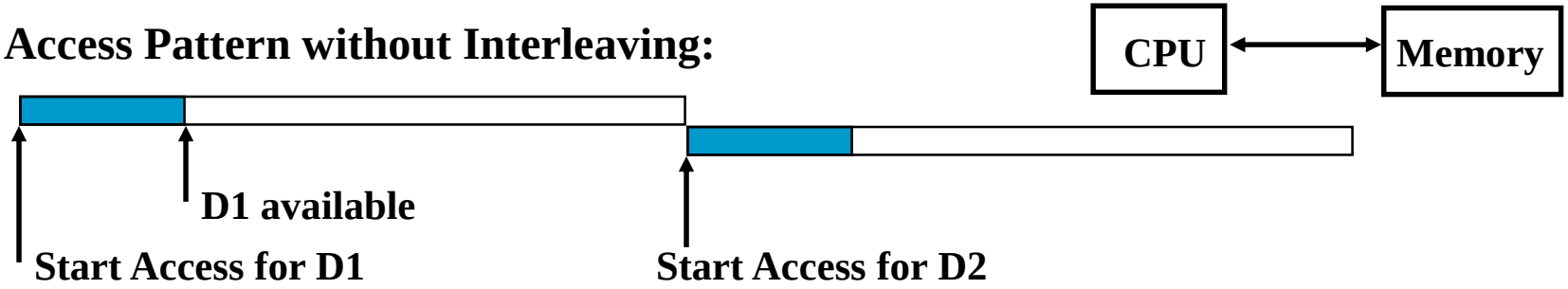
Such a memory organization optimi: sequential memory accesses, whi ideally matched with cache read misses.



- Memory chips can be organized in banks to read or write multiple words at a time
- The banks are often 1 word wide so that width of the bus and cache need ***NOT*** change.
- Sending addresses to several banks permits them all to read simultaneously.

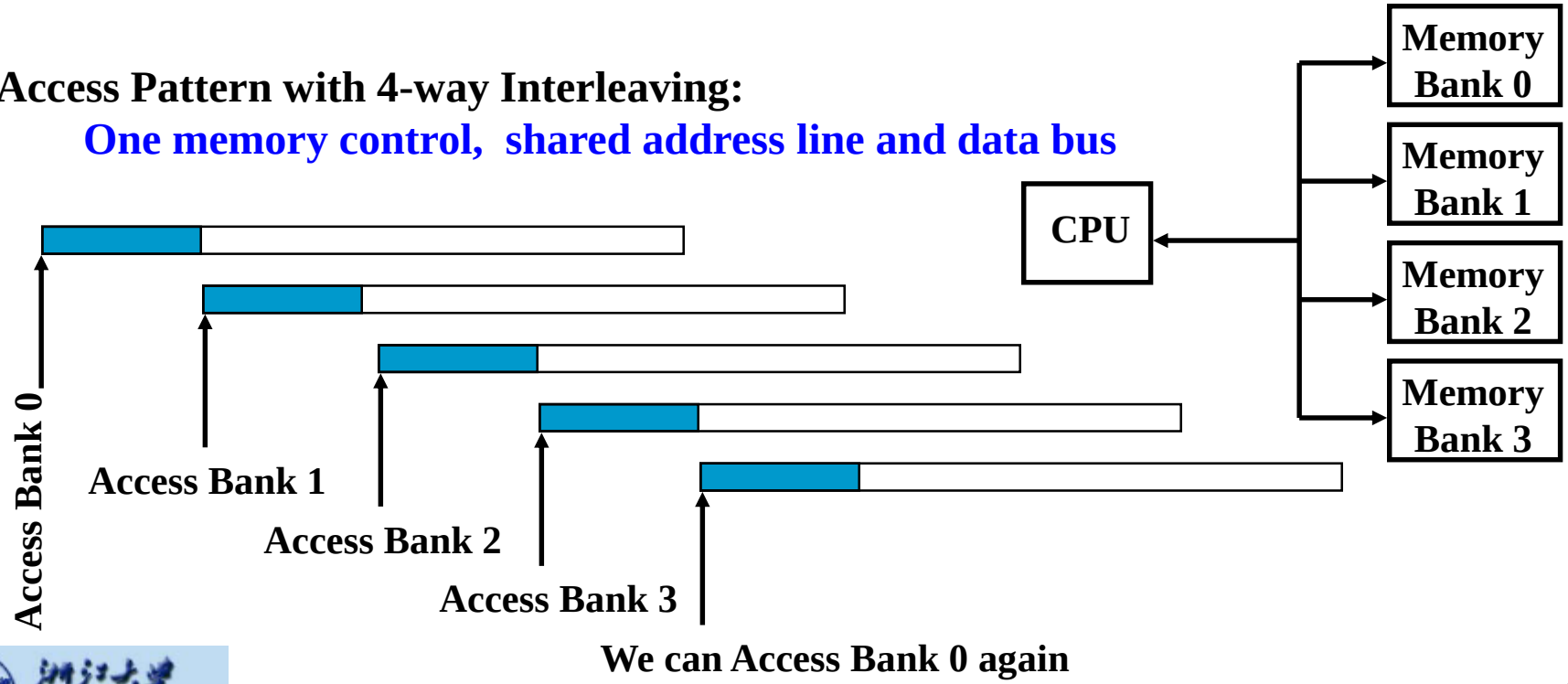
How Access Banks

Access Pattern without Interleaving:



Access Pattern with 4-way Interleaving:

One memory control, shared address line and data bus



Performance of 4-way interleaved memory

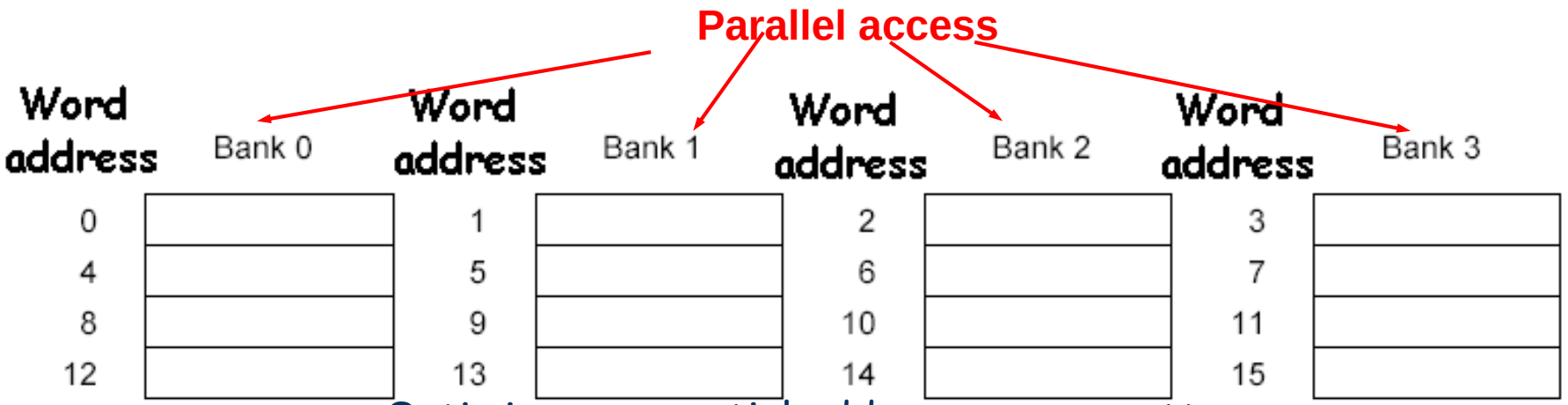
- With 4 banks Interleaved Memory
  The miss penalty: 4words/Block

$$4+56+(4 \times 4)=76$$

  Bandwidth :

$$\frac{4\times 8}{76}=0.4$$

- Four-way interleaved memory



Optimizes sequential address access patterns

3rd Technique for Higher Bandwidth: Independent Memory Banks

How many banks should be included? One metric, used in vector computers, is as follows:

Number of banks ≥ Number of clock cycles to access word in bank

- Independent memory banks
  - The interleaved memory concept can be extended to remove all restrictions on memory access.
    - Independent memory controller was present for every bank.
    - Using separate address line and data bus
    - This allowed the interleaving of sequential access patterns.

Avoiding Memory Bank Conflicts

Assume: Memory banks 128, interleaved on a word basis, and execute following code:

```
int x[256][512];
  for (j = 0; j < 512; j = j+1)
      for (i = 0; i < 256; i = i+1)
          x[i][j] = 2 * x[i][j];
```

How are Memory Bank Conflicts and does it solve ?

4th Technique for Higher Bandwidth: Avoiding Memory Bank Conflicts

Conflicts Since the 512 is an even multiple of 128, all the elements of a **column** will be in the **same memory bank** and code will **stall** on data cache misses no matter how sophisticated a CPU or memory system.

Solutions: There are both software and hardware solutions

The compiler :

- Loop interchange optimization to avoid accessing the same bank.
- For the programmer or the compiler to **expand the size of the array** so that it is **not a power of 2**, thereby forcing the addresses above to go to different banks.

The hardware:

- **Using a prime number of memory banks 2^n-1 or 2^n+1**

Innovations into DRAM chips

- Fewer chips in the same-sized memory system
- 2GB main memory
  - 256 chips * (16M * 4 bits)
  - 16 chips * (256Mbit * 4 bits)

- Shrinking number of chips in a standard configuration shrinks the importance of innovations at the board level.

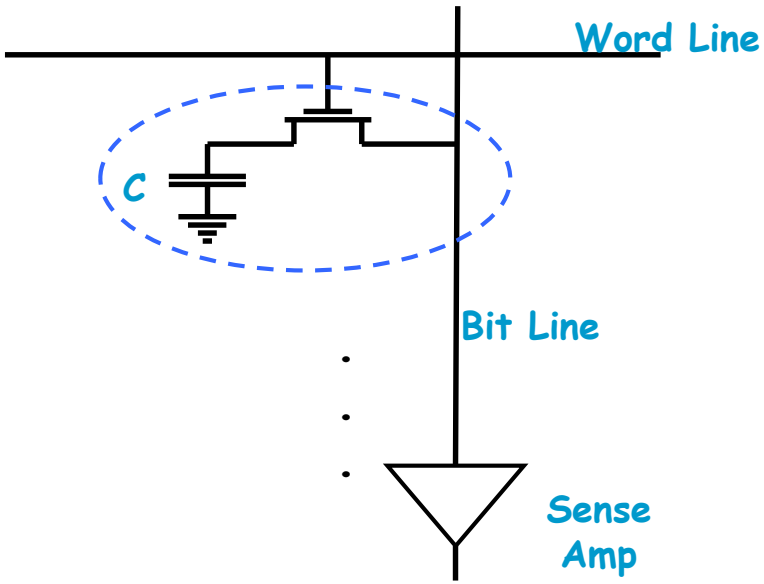
Memory Technologies

- Random Access Memories
  - **DRAM**: *Dynamic* Random Access Memory
    - High density, low power, cheap, slow
    - Dynamic: needs to be "refreshed" regularly
  - **SRAM**: *Static* Random Access Memory
    - Low density, high power, expensive, fast
    - Static: content will last "forever"(until lose power)
- **Two measures—access time and cycle time.**
  - Access time ----- time between when a read is requested and when the desired word arrives,
  - Cycle time ----- minimum time between requests to memory.
  - One reason that cycle time is greater than access time is that the memory needs the address lines to be stable between accesses.

DRAM & SRAM

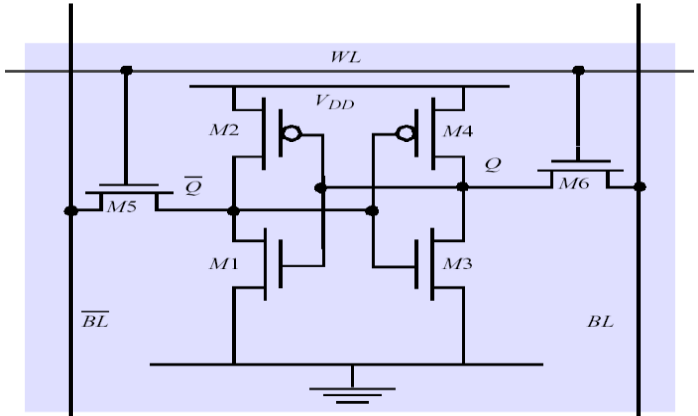
Dynamic RAM

- simple transistor/capacitor pairs in high density form



Static RAM

- Provides regular AND inverted outputs
- Implemented in CMOS process



Single Port 6-T SRAM Cell

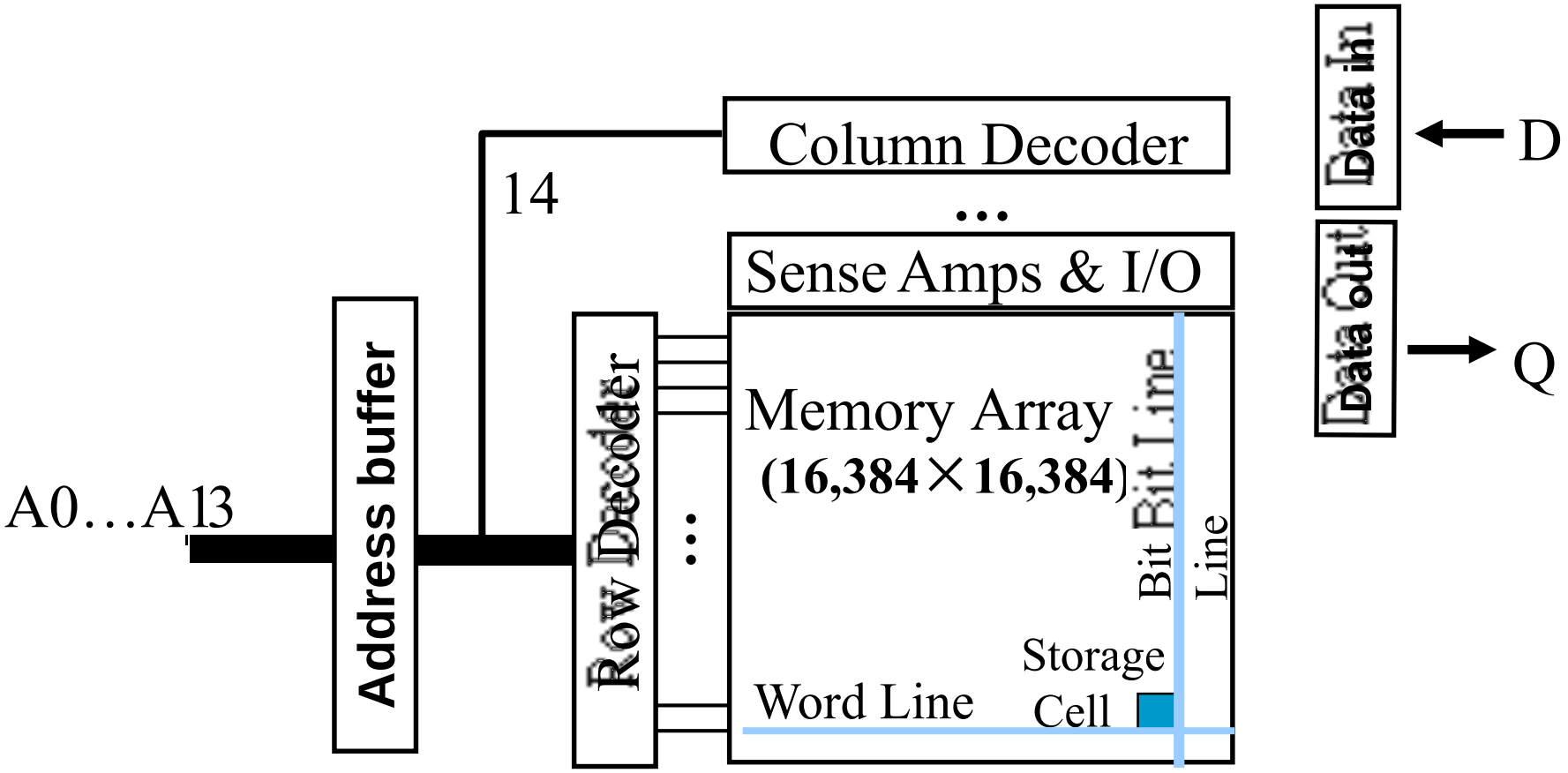
Main Memory Background

- What gets used where?
  - Main memory is **DRAM**: you need it big, so you need it cheap
  - CPU cache memory is **SRAM**: you need it fast, so it's more expensive, so it's smaller than you would usually want due to resource limitations

- Relative performance
  - Size: DRAM/SRAM: 4-8x bigger for DRAM
  - Cost/Cycle time: SRAM/DRAM: 8-16x faster, 8-16x expensive

- Main Memory is *DRAM*: Dynamic Random Access Memory
  - Dynamic since needs to be refreshed periodically (8 ms, 1% time)
  - Addresses divided into 2 halves (Memory as a 2D matrix):
    - *RAS* or *Row Access Strobe*
    - *CAS* or *Column Access Strobe*

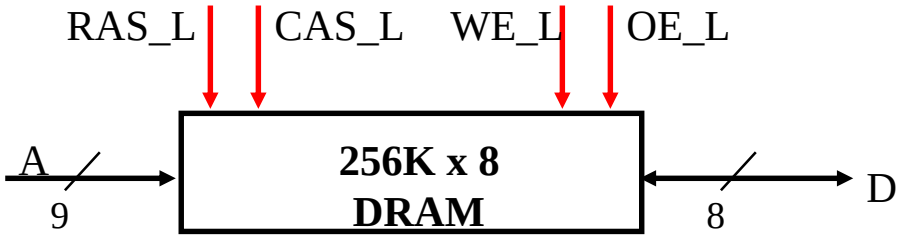
DRAM logical organization (64 Mbit)

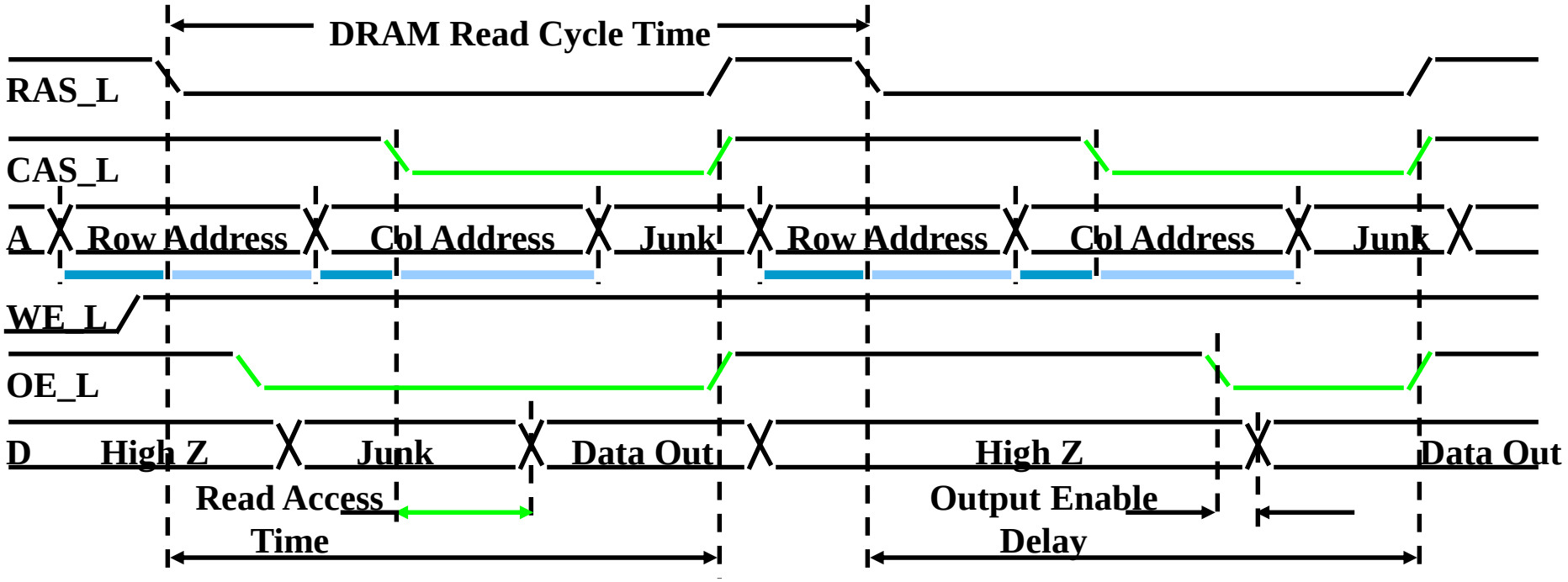


- Square root of bits per RAS/CAS

DRAM Read Timing

- Every DRAM access begins at:
  - The assertion of the RAS_L
  - 2 ways to read:
    early or late v. CAS





Early Read Cycle: OE_L asserted before CAS_L

Late Read Cycle: OE_L asserted after CAS_L

Times of fast and slow DRAMs with each generation.

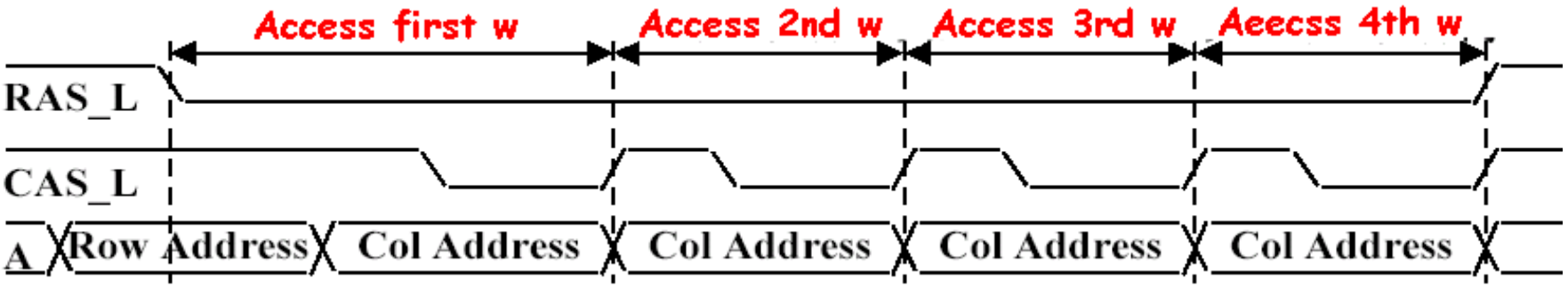
Year of introduction	Chip size	Row access strobe (RAS)		Column access strobe (CAS) / Data Transfer Time	Cycle time
		Slowest DRAM	Fastest DRAM		
1980	64 Kbit	180 ns	150 ns	75 ns	250 ns
1983	256 Kbit	150 ns	120 ns	50 ns	220 ns
1986	1 Mbit	120 ns	100 ns	25 ns	190 ns
1989	4 Mbit	100 ns	80 ns	20 ns	165 ns
1992	16 Mbit	80 ns	60 ns	15 ns	120 ns
1996	64 Mbit	70 ns	50 ns	12 ns	110 ns
1998	128 Mbit	70 ns	50 ns	10 ns	100 ns
2000	256 Mbit	65 ns	45 ns	7 ns	90 ns
2002	512 Mbit	60 ns	40 ns	5 ns	80 ns

Times of fast and slow DRAMs with each generation.

Year of introduction	Chip size	Row access strobe (RAS)		Column access strobe (CAS) / Data Transfer Time	Cycle time
		Slowest DRAM	Fastest DRAM		
1980	64 Kbit	180 ns	150 ns	75 ns	250 ns
1983	256 Kbit	150 ns	120 ns	50 ns	220 ns
1986	1 Mbit	120 ns	100 ns	25 ns	190 ns
1989	4 Mbit	100 ns	80 ns	20 ns	165 ns
1992	16 Mbit	80 ns	60 ns	15 ns	120 ns
1996	64 Mbit	70 ns	50 ns	12 ns	110 ns
1998	128 Mbit	70 ns	50 ns	10 ns	100 ns
2000	256 Mbit	65 ns	45 ns	7 ns	90 ns
2002	512 Mbit	60 ns	40 ns	5 ns	80 ns

1st Improving DRAM Performance
Fast Page Mode DRAM (FPM)

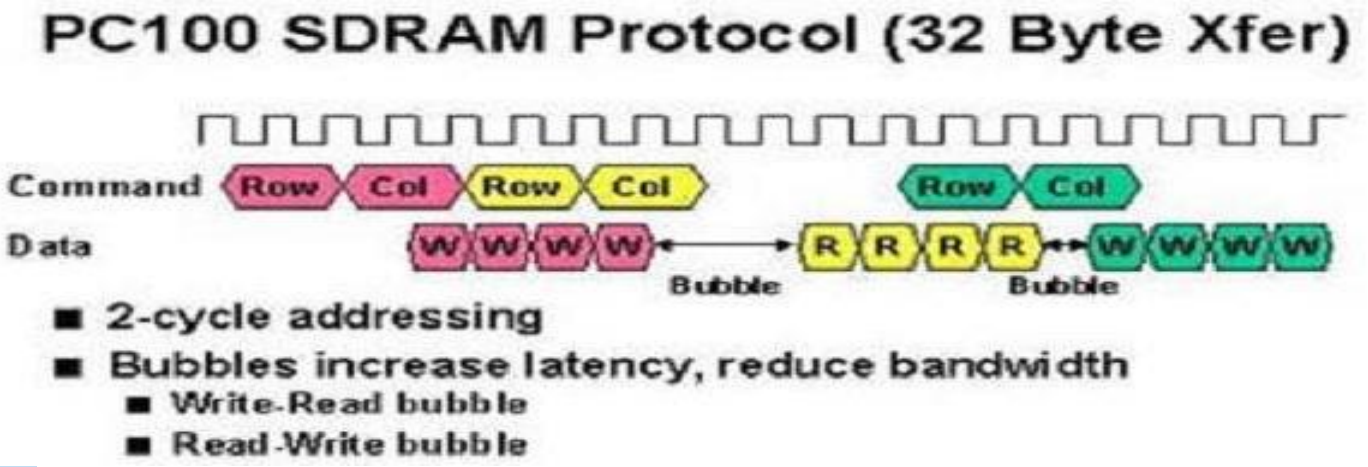
- Timing signals that allow repeated accesses to the row buffer (page) without another row access time
- Such a buffer comes naturally, as each array will buffer 1024 to 2048 bits for each access.
- Page: All bits on the same ROW (Spatial Locality)
  - Don't need to wait for wordline to recharge
  - Toggle CAS with new column address



2nd Improving DRAM Performance

Synchronous DRAM

- conventional DRAMs have an asynchronous interface to the memory controller, and hence every transfer involves overhead to synchronize with the controller.
- The solution was to add a clock signal to the DRAM interface, so that the repeated transfers would not bear that overhead.
  - Data output is in bursts w/ each element clocked



3rd Improving DRAM Performance

DDR--Double data rate

- **On both the** rising edge and falling edge **of the DRAM clock signal, DRAM innovation to increase bandwidth is to transfer data,**
  - **thereby doubling the peak data rate.**

	Standard	Clock rate (MHz)	M transfers per second	DRAM name	MB/sec /DIMM	DIMM name
2.5V	DDR	133	266	DDR266	2128	PC2100
	DDR	150	300	DDR300	2400	PC2400
	DDR	200	400	DDR400	3200	PC3200
1.8v	DDR2	266	533	DDR2-533	4264	PC4300
	DDR2	333	667	DDR2-667	5336	PC5300
	DDR2	400	800	DDR2-800	6400	PC6400
1.5v	DDR3	533	1066	DDR3-1066	8528	PC8500
	DDR3	666	1333	DDR3-1333	10,664	PC10700
	DDR3	800	1600	DDR3-1600	12,800	PC12800

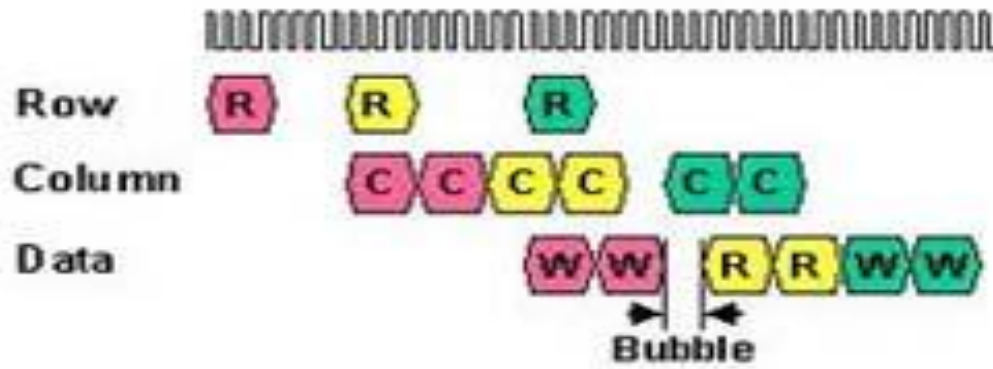
4rd Improving DRAM Performance

New DRAM Interface: RAMBUS (RDRAM)

- a type of synchronous dynamic RAM, designed by the Rambus Corporation.
- Each chip has interleaved memory and a high speed interface.
- Protocol based RAM w/ narrow (16-bit) bus
  - High clock rate (400 Mhz), but long latency
  - Pipelined operation
- Multiple arrays w/ data transferred on both edges of clock
- The first generation RAMBUS interface dropped RAS/CAS, replacing it with a bus that allows other accesses over the bus between the sending of the address and return of the data. It is typically called *RDRAM*.
- The second generation RAMBUS interface include a separate row- and column-command buses instead of the conventional multiplexing; and a much more sophisticated controller on chip. Because of the separation of data, row, and column buses, three transactions can be performed simultaneously. called *Direct RDRAM or DRDRAM*

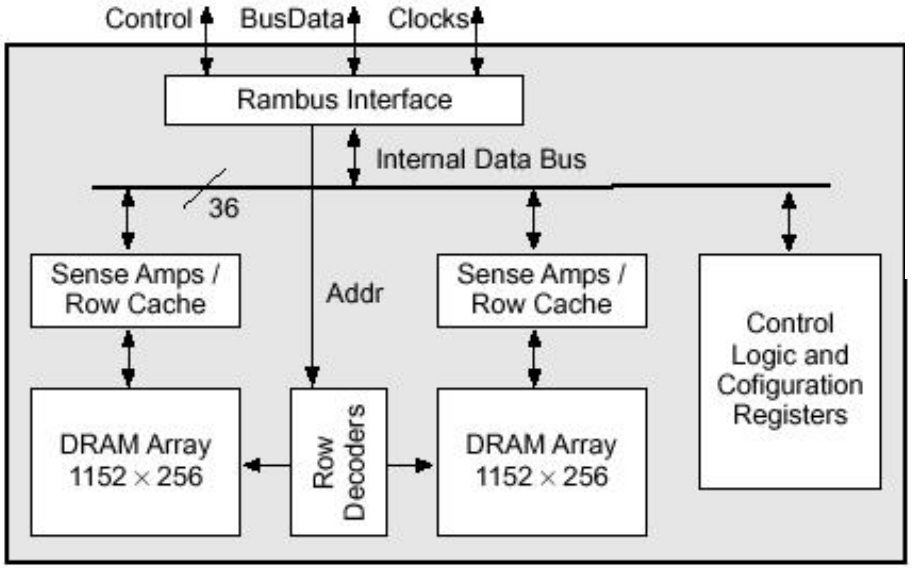
RDRAM Timing

Direct RDRAM Protocol (32 byte Xfer)

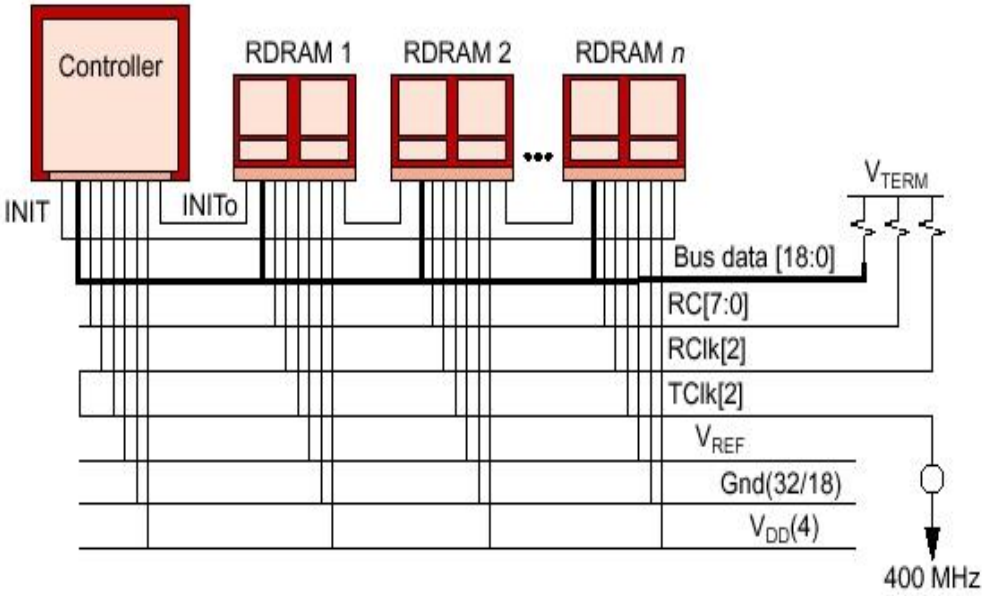


- Separate Row and Column control
  - Enables pipelining, enhances performance
- Smaller Write-Read bubble
  - Increases bandwidth
- Bank conflicts still possible
  - High bank counts reduce probability of conflicts

RAMBUS (RDRAM)



RAMBUS Bank



RDRAM Memory System

Comparing RAMBUS and DDR SDRAM

- Since the most computers use memory in DIMM packages, which are typically at least 64-bits wide, the DIMM memory bandwidth is closer to what RAMBUS provides than you might expect when just comparing DRAM chips.
- Caution that performance of cache are based in part on latency to the first byte and in part on the bandwidth to deliver the rest of the bytes in the block.
  - Although these innovations help with the latter case, none help with latency.
  - Amdahl's Law reminds us of the limits of accelerating one piece of the problem while ignoring another part.

Summery

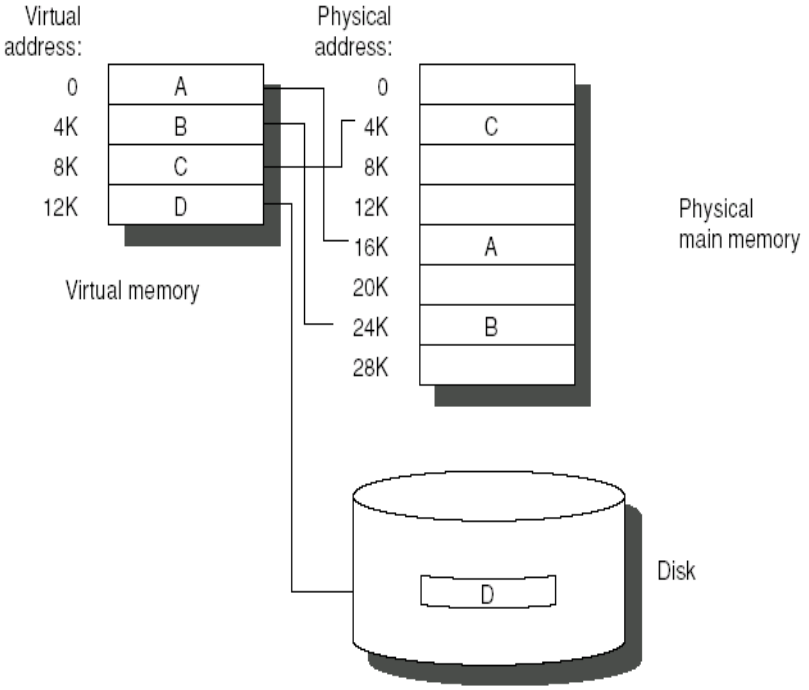
- Memory organization
  - Wider memory
  - Simple interleaved memory
  - Independent memory banks
  - Avoiding Memory Bank Conflicts
- Memory chip
  - Fast Page Mode DRAM
  - Synchronize DRAM
  - Double Date Rate
  - RDRAM

5.10 virtual Memory

- What is virtual memory?
  - Technique that allows execution of a program that
    - can reside in discontiguous memory locations
    - does not have to completely reside in memory
  - Allows the computer to "fake" a program into believing that its
    - memory is contiguous
    - memory space is larger than physical memory, Provides *illusion* of very large memory
- Why is VM important?
  - Cheap - no longer have to buy lots of RAM
  - Removes burden of memory resource management from the programmer
  - Enables multiprogramming, time-sharing, protection

Advantages

- Main memory (physical memory) can act as a *cache* for the secondary storage (disk)
- illusion of having more and contiguous physical memory
- program relocation by "pages" or "segment"
- protection in multiprogramming



Virtual Address : address used by the programmer

Virtual Address Space: collection of such addresses

Memory Address: address of word in physi memory also known as "physical address" "real address"

How Does VM Work

Two memory "spaces"

- Virtual memory space - what the program "sees"
- Physical memory space - what the program runs in (size of RAM)

On program startup

- OS copies program into RAM
- If there is not enough RAM, OS stops copying program & starts running the program with some portion of the program loaded in RAM
- When the program touches a part of the program not in physical memory, OS copies that part of the program from disk into RAM
- In order to copy some of the program from disk to RAM, OS must evict parts of the program already in RAM
  - OS copies the evicted parts of the program back to disk if the pages are dirty (ie, if they have been written into, and changed)

Memory Hierarchy Parameters for Virtual Memory.

Terms are different

Block ----*Page* or *segment*

Miss ----*page fault* or *address fault*

Memory mapping or *address translation* ----**With virtual memory, the CPU produces *virtual addresses* that are translated by a combination of *hardware and software* to *physical addresses*, which access main memory.------ Operating system translate**

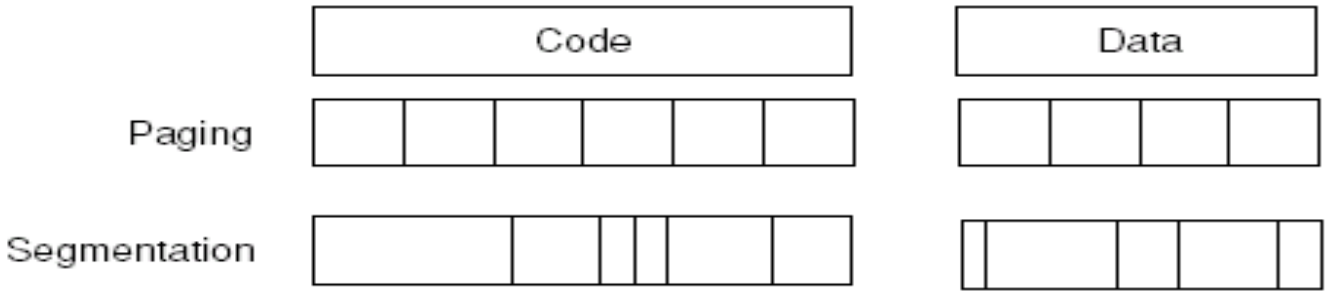
The two memory-hierarchy levels controlled by virtual memory are *DRAMs and magnetic disks*.

Cache vs. VM

Parameter	First-level cache	Virtual memory
Block (page) size	16-128 bytes	4096-65,536 bytes
Hit time	1-3 clock cycles	50-150 clock cycles
Miss penalty	8-150 clock cycles	1,000,000-10,000,000 clock cycles
(Access time)	(6-130 clock cycles)	(800,000-8,000,000 clock cycles)
(Transfer time)	(2-20 clock cycles)	(200,000-2,000,000 clock cycles)
Miss rate	0.1-10%	0.00001- 0.001%
Address mapping	25- 45 bit physical address to 14- 20 bit cache address	32-64 bit virtual address to 25-45 bit physical address

Paging vs. Segmentation

	Page	Segment
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Trivial (all blocks are the same size)	Hard (must find contiguous, variable-size, unused portion of main memory)
Memory use inefficiency	Internal fragmentation (unused portion of page)	External fragmentation (unused pieces of main memory)
Efficient disk traffic	Yes (adjust page size to balance access time and transfer time)	Not always (small segments may transfer just a few bytes)



Four Memory Hierarchy Questions Revisited

Q1: Where can a block be placed in main memory?

- **The high miss penalty**
  - Quite high
    - access to a rotating magnetic storage device
- **Must be lower miss rates**
  - choosing a simpler placement algorithm
  - operating systems designers normally pick lower miss rates because of the exorbitant miss penalty.
- **Fully associative strategy.**
  - Thus, operating systems allow blocks to be placed anywhere in main memory.

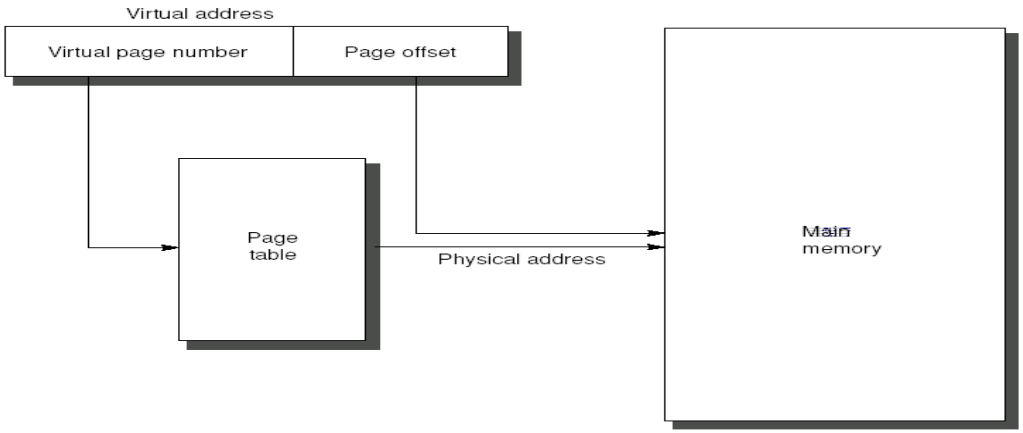
Q2: How is a block found if it is in main memory?

Both paging and segmentation data structure table

- The data structure contains the physical address of the block.
- That is indexed by the page or segment number

- For segmentation:
  - The offset is added to the segment's physical address to obtain the final physical address.
- For paging:
  - the offset is simply concatenated to this physical page address
- The size of the table
  - The number of pages in the virtual address space.
  - Assume: 32-bit virtual address, 4-KB pages, and 4 bytes per page table entry; the size of the page table:

How to find a block is in memory



$$\frac{2^{32}}{2^{12}} \times 2^2 = 2^{22}\text{B} = 4\ \text{MB}$$

To reduce the size table

-Apply a hashing function to the virtual address.

-The hash allows the data structure to be the length of the number of *physical pages* in main memory. Such a structure is called an *inverted page table*.

To reduce address translation time

-Computers use a cache dedicated to these address translations, called a translation look-aside buffer, or simply translation buffer.

Q3: Which block should be replaced on a virtual memory miss?

For minimizing page faults

- Almost all operating systems try to replace the least-recently used (LRU) block
- because if the past predicts the future, that is the one less likely to be needed.

Mechanism

- many processors provide a *use bit or reference bit*
  - which is logically set whenever a page is accessed.
  - The operating system periodically clears the use bits and later records them
  - By keeping track in this way, the operating system can select a page that is among the least-recently referenced.

Q4: What happens on a write?

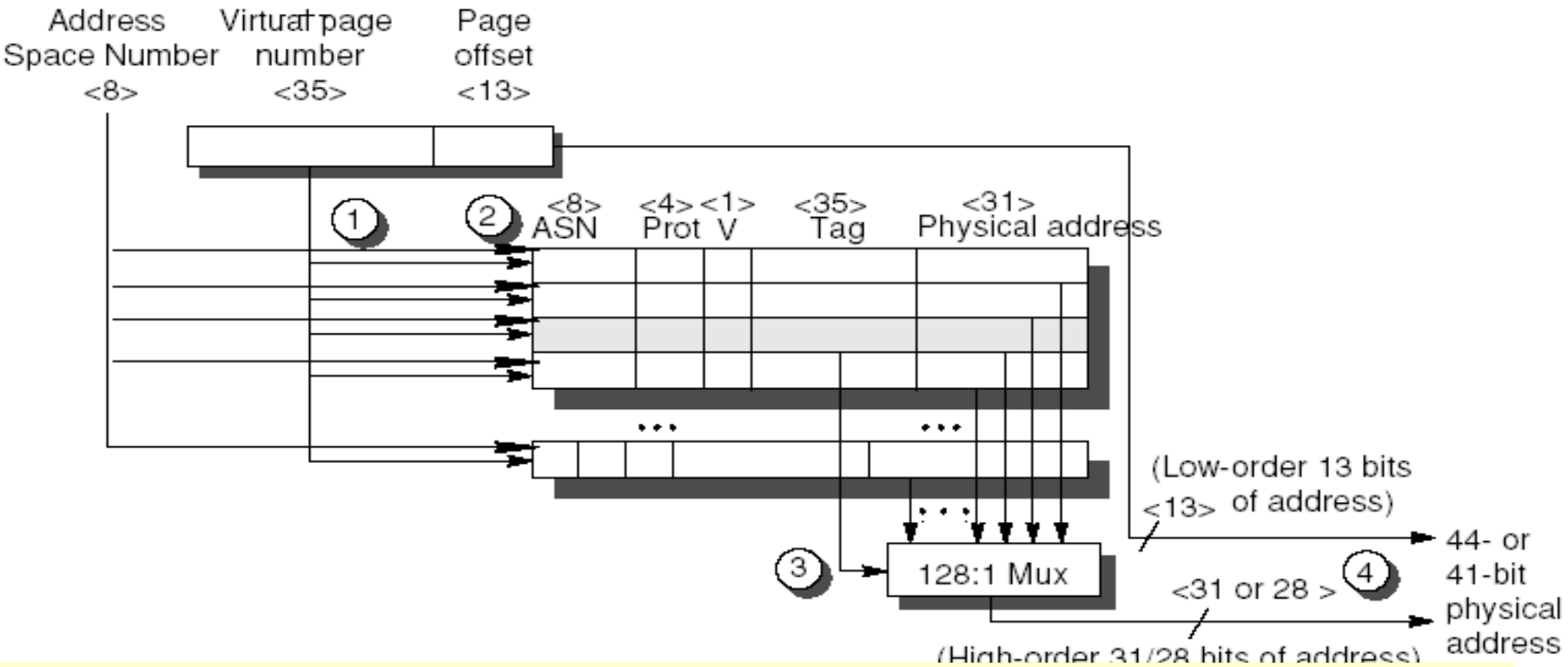
write strategy

- The level below main memory contains rotating magnetic disks that take millions of clock cycles to access.
  - Thus, the write strategy is always write back.
- Dirty bit
  - cost of an unnecessary access Disk is so high, it allows blocks to be written to disk only if they have been altered since being read from the disk.

Techniques for Fast Address Translation

- Access frequently with obtaining one data This cost is far too dear
  - Page tables are usually so large that they are stored in main memory, and sometimes paged themselves.
  - Paging means that every memory access logically takes at least twice as long, with one memory access to obtain the physical address and a second access to get the data.
- translation look-aside buffer----TLB
  - if the accesses have locality, then the address translations for the accesses must also have locality.
  - By keeping these address translations in a special cache, a memory access rarely requires a second access to translate the data.
  - This special address translation cache is referred to as a translation look-aside buffer or TLB, also called a translation buffer or TB.

Operation of the Alpha 21264 data TLB during address translation



TLB each entry has an 8-bit Address Space Number (ASN)

- If the context switching returns to the process with the ***same*** ASN, it can still match the TLB.
- Thus, the process ASN and the *page table entry* (PTE) ASN must also match for a valid tag (same role as a Process PID number)

Steps for address translation

steps 1 and 2:

- The translation begins by sending the virtual address to all tags. Of course, the tag must be marked valid to allow a match.
- At the same time, the type of memory access is checked for a violation (in step 2) against protection information in the TLB.

Sept 3: The matching tag sends the corresponding physical address(page) through effectively a 128:1 multiplexor

Sept 4: The page offset is then combined with the physical page frame to form a full

physical address The address size is 44 or 41 bits depending on a physical address mode bit

Balancing smaller and larger

Balancing larger page size versus smaller size.

- The size of the page table is inversely proportional to the page size;
  - memory (or other resources used for the memory map) can therefore be saved by making the pages bigger.
- As mentioned on page 433 in section 5.7, a larger page size can allow larger caches with fast cache hit times.
- Transferring larger pages to or from secondary storage, possibly over a network, *is more efficient than transferring smaller pages.*
- The number of TLB entries are restricted, so a larger page size means that more memory can be mapped efficiently, thereby reducing the number of TLB misses.

Selecting a Page Size

Smaller page size

- Be conserving storage.
- A small page size will result in less wasted storage when a contiguous region of virtual memory is not equal in size to a multiple of the page size.
- *Internal fragmentation*----Unused memory in a page

larger page size

- The page sizes become very large (more than 32 KB), lots of storage (both main and secondary) may be wasted
- As well as I/O bandwidth.
- A final concern is process start-up time
  - many processes are small, so a large page size would lengthen the time to invoke a process

Multiple page size

To support multiple page sizes

It is ***for the number of TLB entries*** that recent microprocessors have decided to support multiple page sizes

For some programs, TLB misses can be as significant on CPI as the cache misses.

Summary of Virtual Memory and Caches

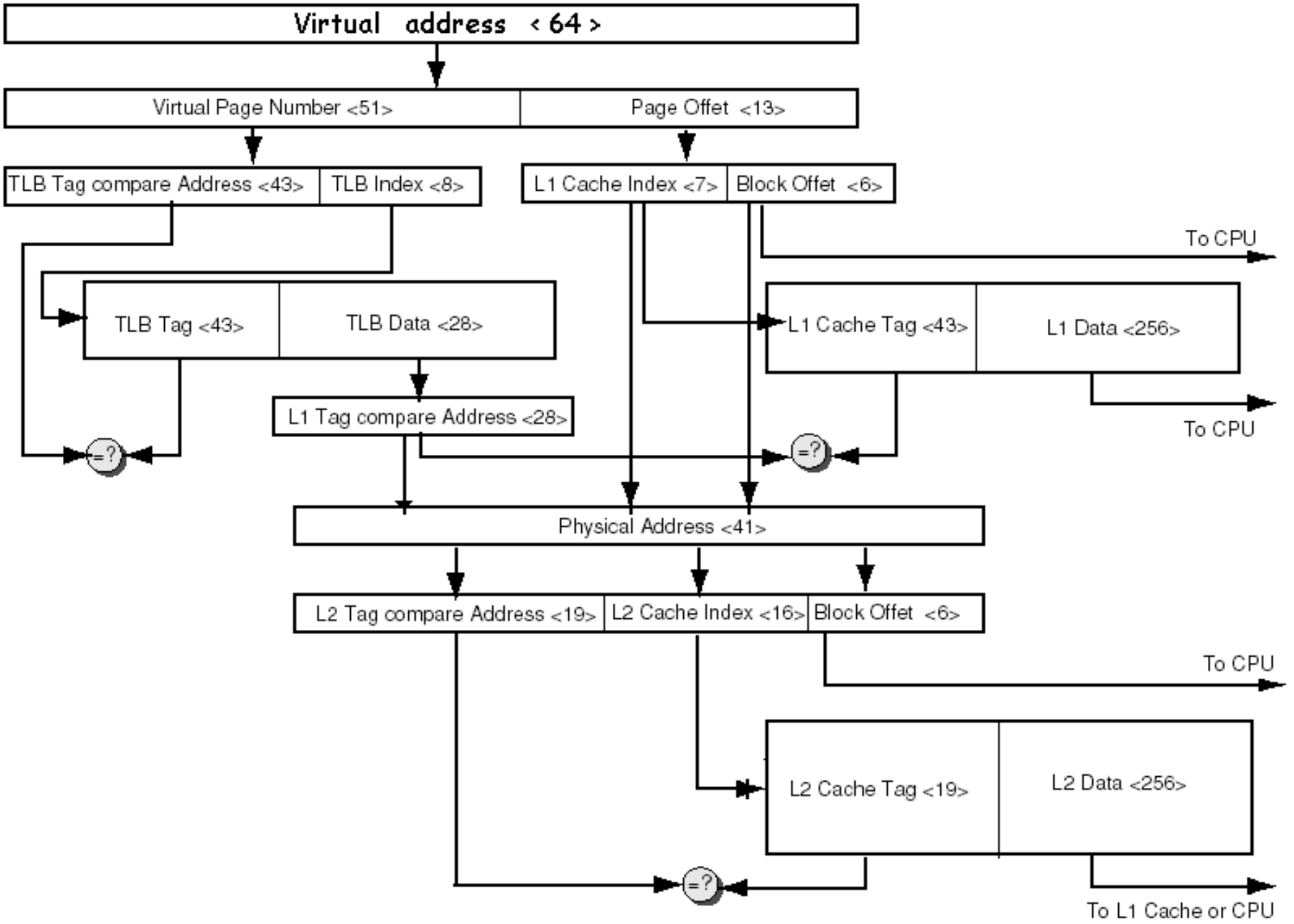
Hypothetical parameter

Virtual address: 64-bit

Physical address: 41 bit

Memory organization: with two levels of cache

- This L1 cache is virtually indexed, physically tagged since both the cache size and the page size are 8 KB.
- The L2 cache is 4 MB. The block size for both is 64 bytes.


Virtual address <64>
Virtual Page Number <51>
Page Offet <13>
TLB Tag compare Address <43>
TLB Index <8>
L1 Cache Index <7>
Block Offet <6>
To CPU
TLB Tag <43>
TLB Data <28>
L1 Cache Tag <43>
L1 Data <256>
L1 Tag compare Address <28>
To CPU
=?
=?
Physical Address <41>
L2 Tag compare Address <19>
L2 Cache Index <16>
Block Offet <6>
To CPU
L2 Cache Tag <19>
L2 Data <256>
=?
To L1 Cache or CPU

Protection and Examples of Virtual Memory

Multiprogramming and *process*.

- computer would be shared by several programs running concurrently, which led to new demands for protection and sharing among programs.

process switch or context switch.

- Giving the illusion that all users have their own computers.
  - Time-sharing is a variation of multiprogramming that shares the CPU and memory with several interactive users at the same time
- **Thus, at any instant it must be possible to switch from one process to another.**

Protection

- *VM is often used to protect one program from others in the system.*
- *Protection mechanisms must have hardware and software support.*
  - The computer designer must ensure that the CPU portion of the process state can be saved and restored.
  - The operating system designer must guarantee that processes do not interfere with each others' computations.

Protecting Processes

Protecting with Base & bounds

- A pair of registers with address range that checks every address that accesses memory
- Each reference must fall between two addresses, given by the base & bound registers.

Base and Bound ----Address limits

- An address is valid if Base ≤ Address ≤ Bound

  Or (Base + Address) ≤ Bound

Change registers values

- User processes cannot be allowed to change these registers
- but the OS must be able to do so on a process switch.

Three more Responsibilities

Computer designer has three more responsibilities in helping the operating system designer protect processes from each other:

1. **Provide at least two modes**
   - indicating whether the running process is a user process
   - or an operating system process. Called a *kernel process*, a *supervisor process*, or an *executive process*.
2. **Provide a portion of the CPU state**
   - that a user process can use but not write.
   - This state includes
     - The base/bound registers,
     - a user/supervisor mode bit(s),
     - And the exception enable/disable bit.
     - Users are prevented from writing this state

Three more Responsibilities

3. Provide mechanisms whereby the CPU can go from user mode to supervisor mode and vice versa.
   - The first direction ----typically accomplished by a *system call*, implemented as a special instruction that transfers control to a dedicated location in supervisor code space.
     - The PC is saved from the point of the system call
     - The CPU is placed in supervisor mode.
   - The second direction ---- return to user mode
     - like a subroutine return that restores the previous user/supervisor mode.

Protecting with Another Idea

- Fine-grained Protecting
- **Virtual memory** offers a more fine-grained protection
  - The CPU address must be mapped from virtual to physical address.
  - This mapping provides the opportunity for the hardware to *check further for errors in the program* or to protect processes from each other.
- How:
  - **add permission flags** *to each page or segment*.
  - For example, since few programs today intentionally modify their own code, an operating system can detect accidental writes to code by offering read-only protection to pages.
  - This page-level protection can be extended by adding user/kernel protection to prevent a user program from trying to access pages that belong to the kernel.

Protecting with Another Idea-2

Page tables Protecting

- Processes are thus protected from one another by having their own page tables, each pointing to distinct pages of memory.
- Obviously, user programs must be prevented from modifying their page tables or protection would be circumvented.

Rings Protecting

- Rings added to the CPU protection structure expand memory access protection from two levels (user and kernel) to many more.
  - Top secret→ secret → confidential → unclassified
  - Concentric ***rings*** of security levels allow the most trusted let the second most trusted to access everything except the innermost level, and so on.
  - The "civilian" programs are the least trusted and, hence, have the most limited range of accesses.

Protecting with Another Idea-3

Key Protecting

- Restricting the freedom given a program in the inner sanctum requires a new classification system.
- Keys and Locks
  - A program can't unlock access to the data unless it has the key.
  - For these keys, or capabilities, to be useful, the hardware and operating system must be able to explicitly pass them from one program to another without

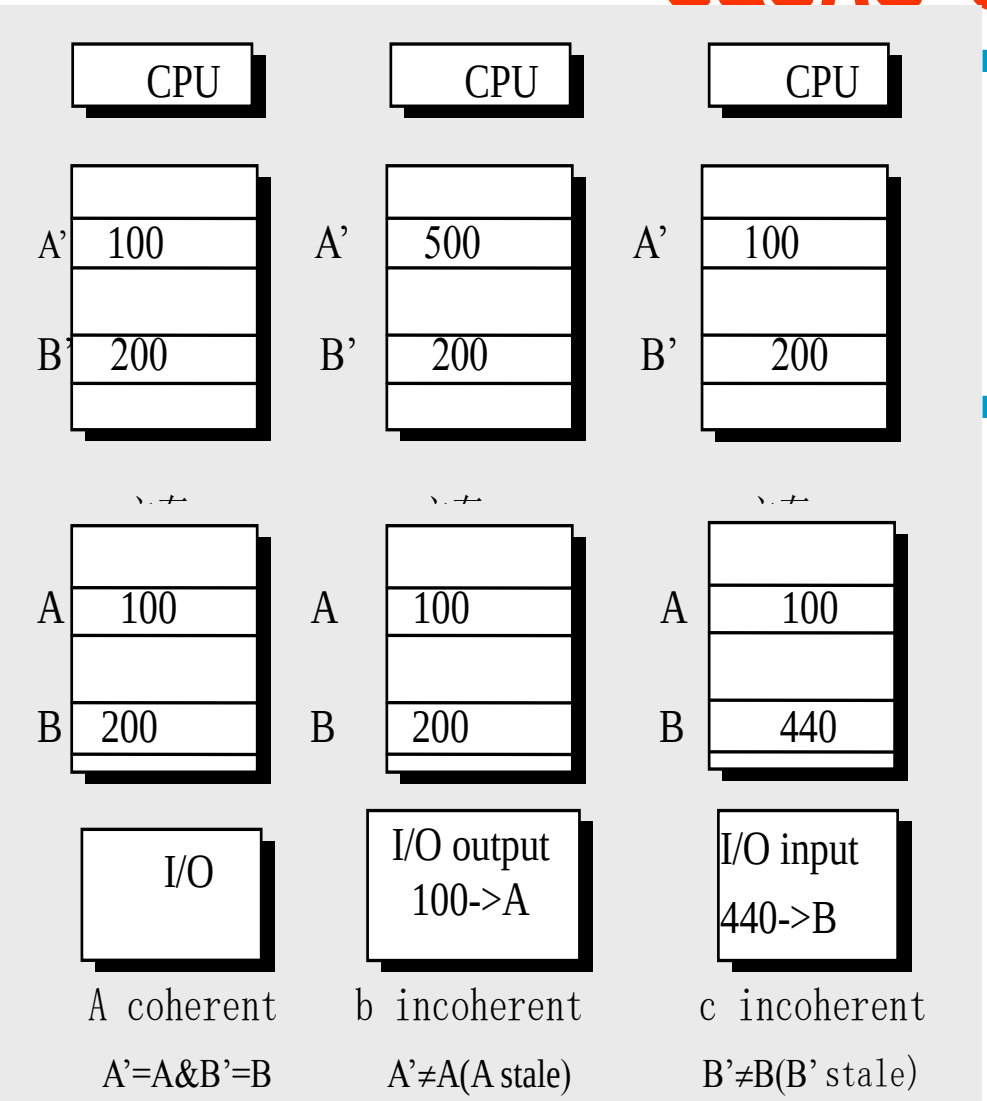
Segmented VM example : P476
Protection in the intel Pentium

- Double the traditional two-level protection model
  - Innemost (0) to outermost (3)
- Divides the address space, allowing both the OS and user access to the full space.
  - Global address space: shared by all processes
  - Local address space: unique to each process
- Descriptor table (item: Segment descriptor)
  - Present bit ~ valid bit
  - Base field ~ page frame address
  - Access bit ~ reference bit
  - Attributes field ~ valid operation and protection level
  - Special segment descriptor: Call gate

5.12 Crosscutting Issues: the design of memory hierarchy

- Superscalar CPU →
  - Cache should provide peak bandwidth that matches CPU demand
  - the memory hierarchy must also be nonblocking
- Speculative Execution→
  - Memory system must identify speculatively executed instructions and conditionally executed instructions and suppress corresponding exception
  - Nonb locking caches
- Combine Instruction cache with IF and ID
  - Trace cache, branch prediction with IF,
- Embedded Computer Cache and Real-time performance
  - A portion of the cache can be "locked down" under program control.
- Embedded computer Caches and power
  - Way prediction to only power half of the address-checking hardware for two-way set-associate cache.

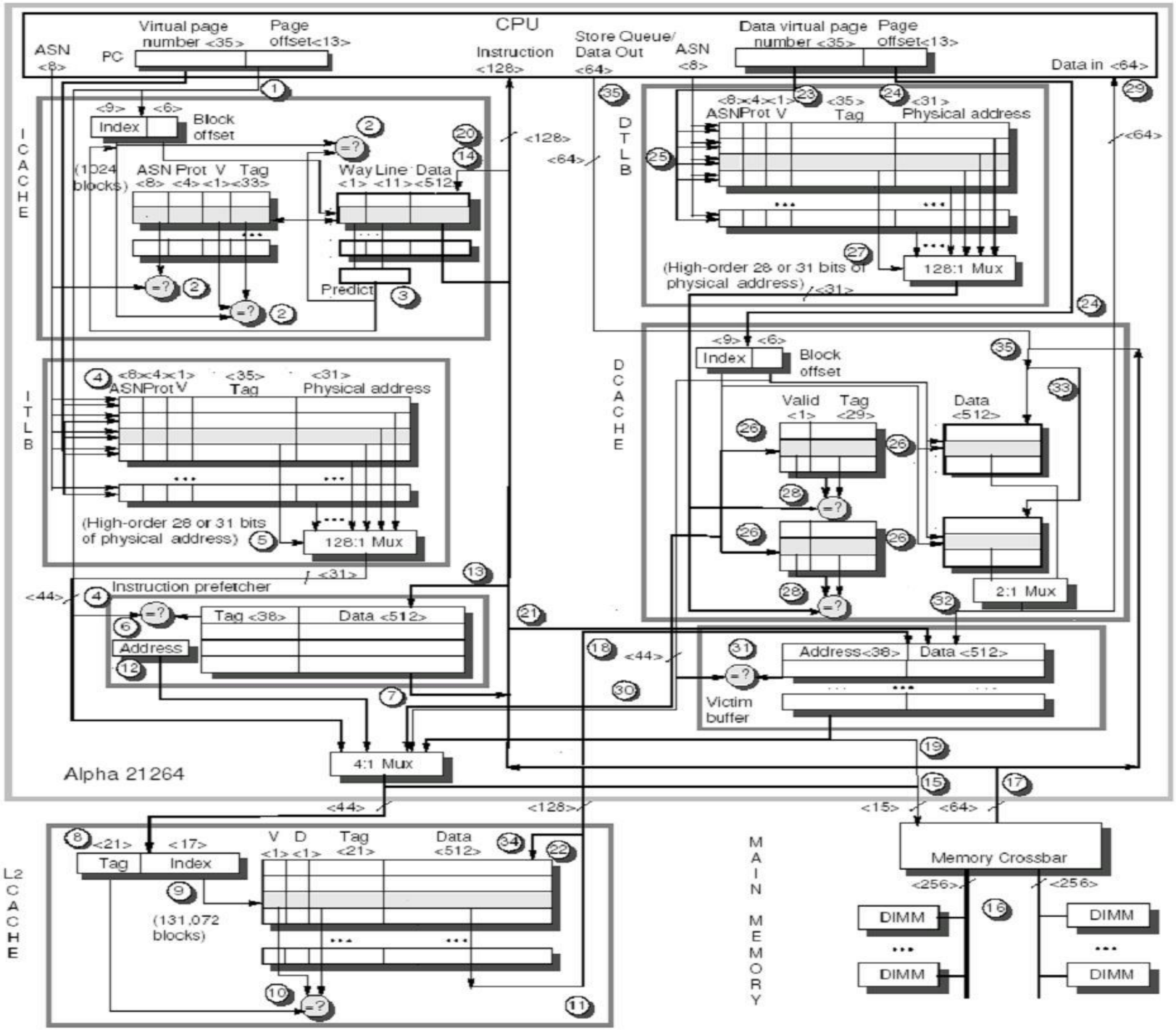
I/O and consistency of cache data



- Solution of Situation a
  - Write through to avoid incoherent of b
  - Write through is usually found in 1-level cache backed by a write –back 2-level cache.
- Solution of Situation b
  - **Software:** Guarantee no blocks of the I/O buffer are in the cache
    - Mark buffered blocks as **noncachable**
    - **Flushes** the buffer addresses from the cache before the input occur
  - **Hardware:** check the I/O address on input to see if they are in the cache.
  - A duplicate set of tags may be used

Example: Alpha 21264 Memory Hierarchy

- **The 21264 is an out-of-order execution processor**
  - fetches up to 4 instructions per clock cycle and executes up to 6 instructions per clock cycle.
- **virtual address**
  - 48-bit virtual address and a 44-bit physical address
    / 43-bit virtual address and 41-bit physical;
  - In either case, Alpha halves the physical address space, with the lower half for memory addresses and the upper half for I/O addresses.
- **when the Alpha is turned on.**
  - Hardware on the chip loads the instruction cache serially from an external PROM. (16K instructions)
  - The same serial interface (and PROM) also loads configuration information that specifies L2 cache speed/timing, system port speed/timing, and much other information necessary


CPU
ASN
PC
Virtual page number <35>
Page offset<13>
Instruction <128>
Store Queue/ Data Out <64>
Data virtual page number <35>
Data in <64>
ICACHE
Index
Block offset
(1024 blocks)
ASN Prot V Tag
Way Line Data
Predict
DTLB
ASN Prot V Tag
Physical address
(High-order 28 or 31 bits of physical address)
128:1 Mux
ITLB
Instruction prefetcher
Tag <38>
Data <512>
Address
DCACHE
Valid Tag
Data
2:1 Mux
Victim buffer
Address<38>
4:1 Mux
Alpha 21264
L2 CACHE
V D Tag
(131,072 blocks)
MAIN MEMORY
Memory Crossbar
DIMM