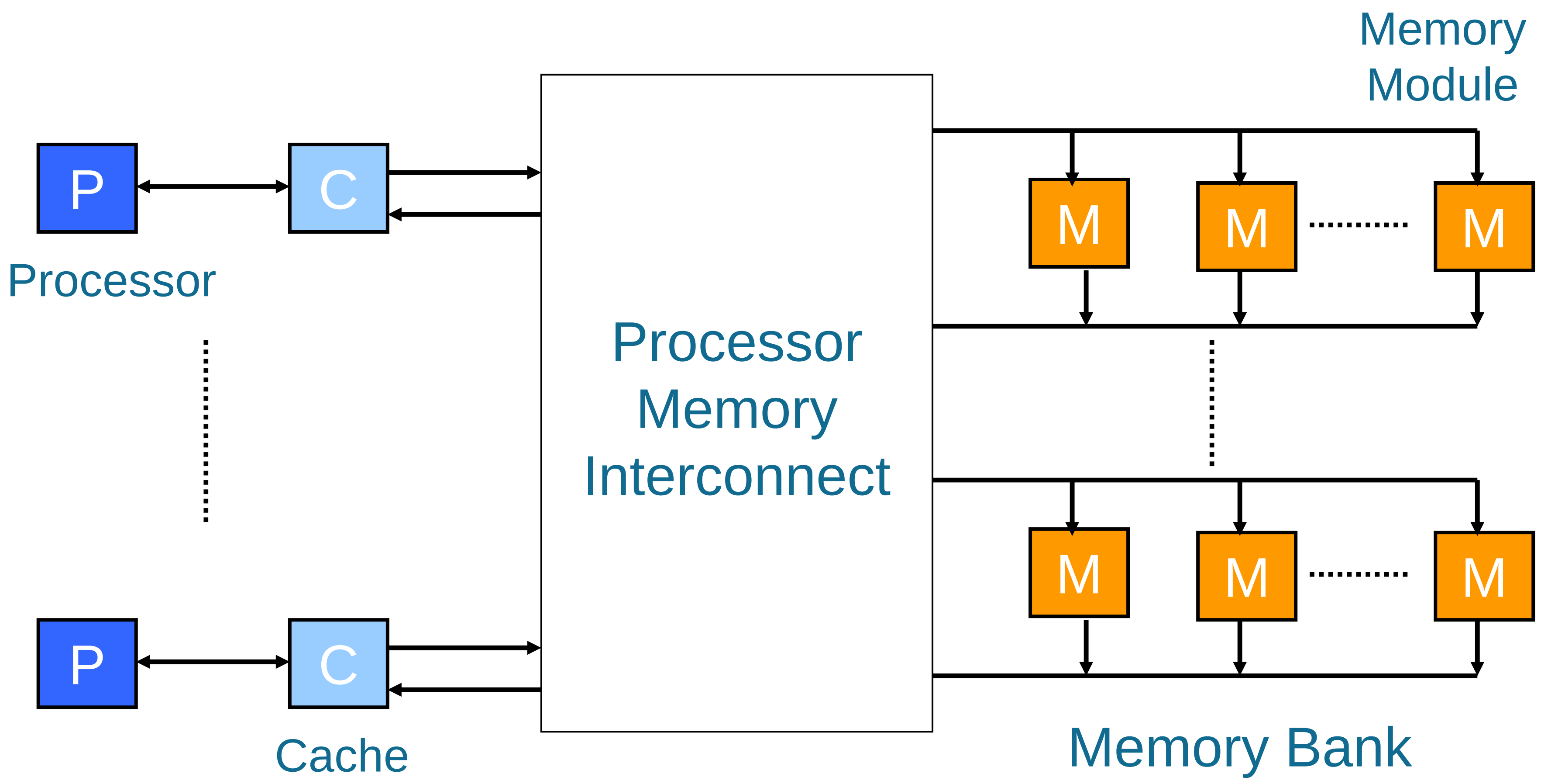
ARM Architecture

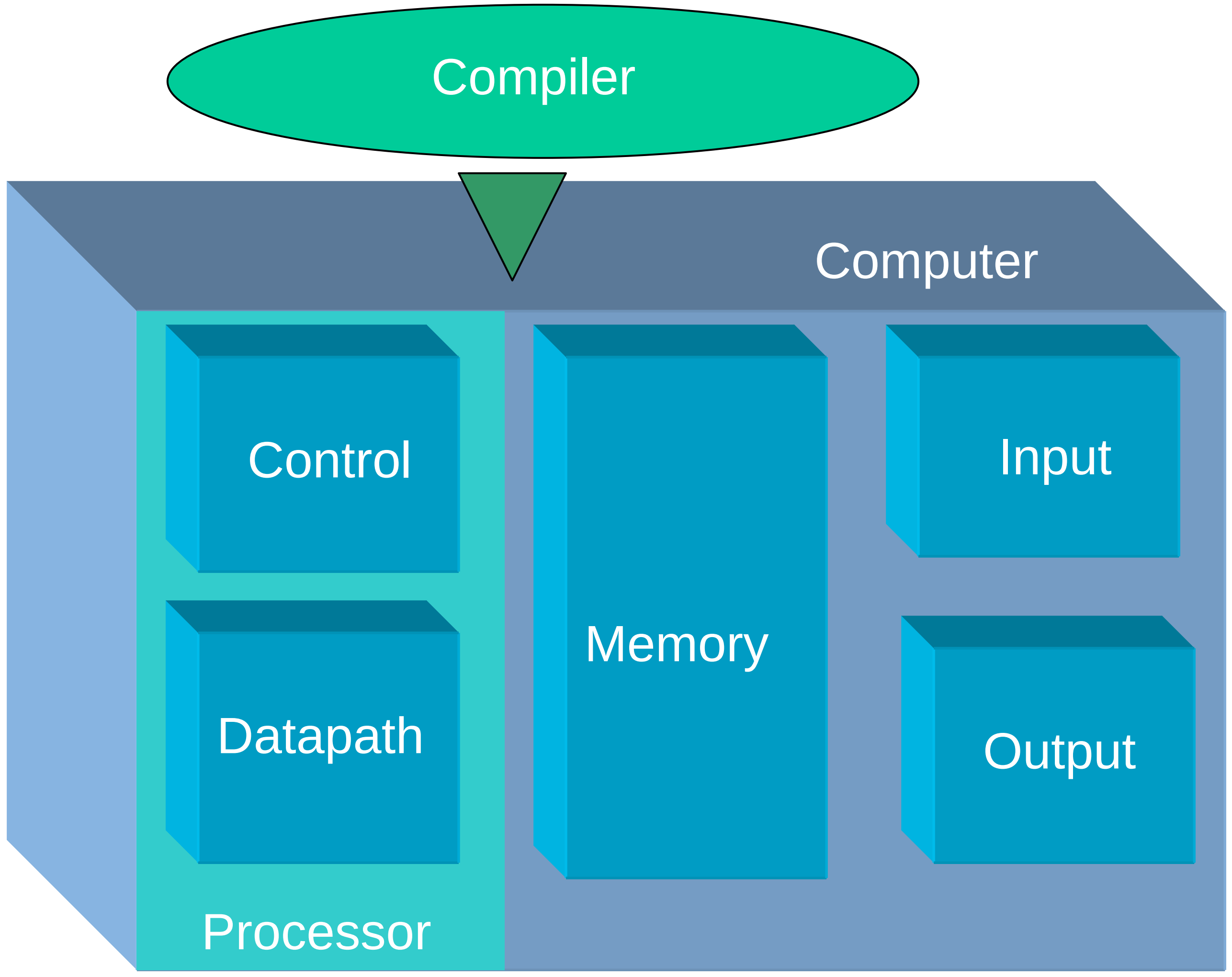
Memory Subsystem

Ashish Kuvelkar
Hardware Technology Development Group
C-DAC, Pune

Canonical Computer System



Components of a Computer



Memory System Performance

- Access time
  - Time elapsed between time request is made and data reaches processor
  - Dependent on bus delay, chip access delay
- Memory Bandwidth
  - Amount of information transferred over a period of time. Depends upon
    - Way physical memory is organised
    - Number of banks
    - Accessing modes within module

Memory Requirements

- Programmers want unlimited amount of fast memory
- Hardware aids programmer by creating an illusion of unlimited fast memory
- Realities
  - Fast memory is costly
  - Programs tend to follow principle of locality
    - Temporal: same location is repeatedly accessed
    - Spatial: Close by locations are accessed soon

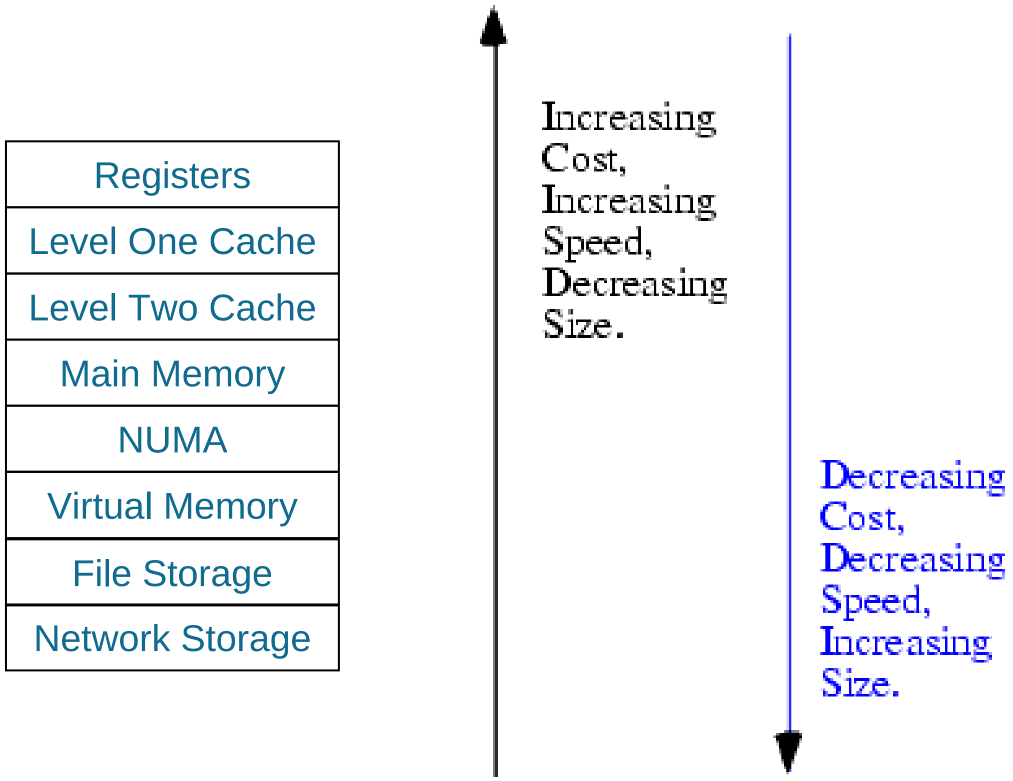
Processor and memory speeds

- When RISC processors were introduced, standard memory parts were faster than contemporary microprocessors
- Over a period
  - Microprocessors have become faster
  - Memories have become little faster but mostly have higher capacity
- The memories are not able to keep up with the speed achieved by microprocessors

Memory Hierarchy

- Takes advantage of principle of locality
- Memory subsystem consists of multiple levels with different speeds and sizes.
- Small fast memory is put close to processor and slower away
- The goal
  - Provide maximum memory which is cheapest
  - Provide access at the speed offered by the fastest
- Data is copied only between two adjacent levels
- If data is found in level closest to CPU, it's a hit
- Hit rate is a measure of the performance

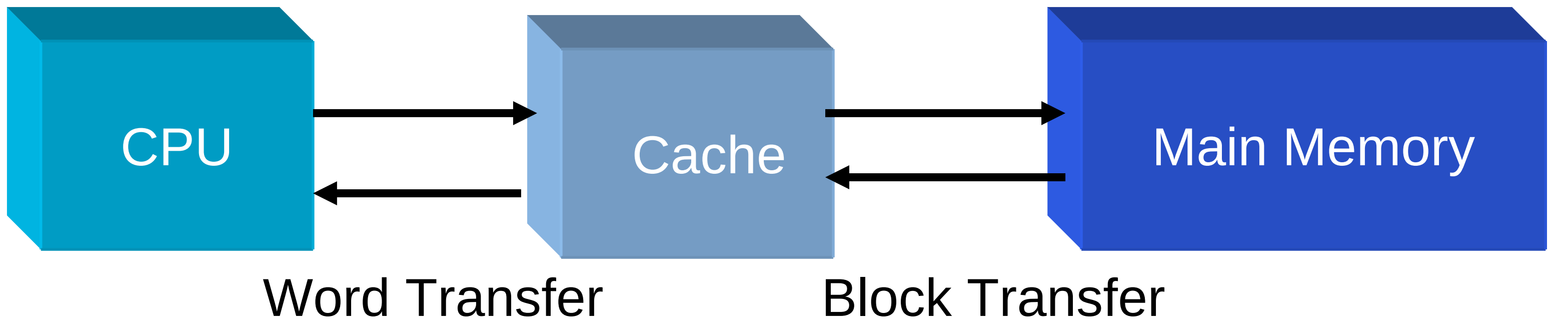
Memory Hierarchy


Registers
Level One Cache
Level Two Cache
Main Memory
NUMA
Virtual Memory
File Storage
Network Storage
Increasing Cost, Increasing Speed, Decreasing Size.
Decreasing Cost, Decreasing Speed, Increasing Size.

Cache

Cache: as safe place for hiding or storing things

-Webster's New world dictionary

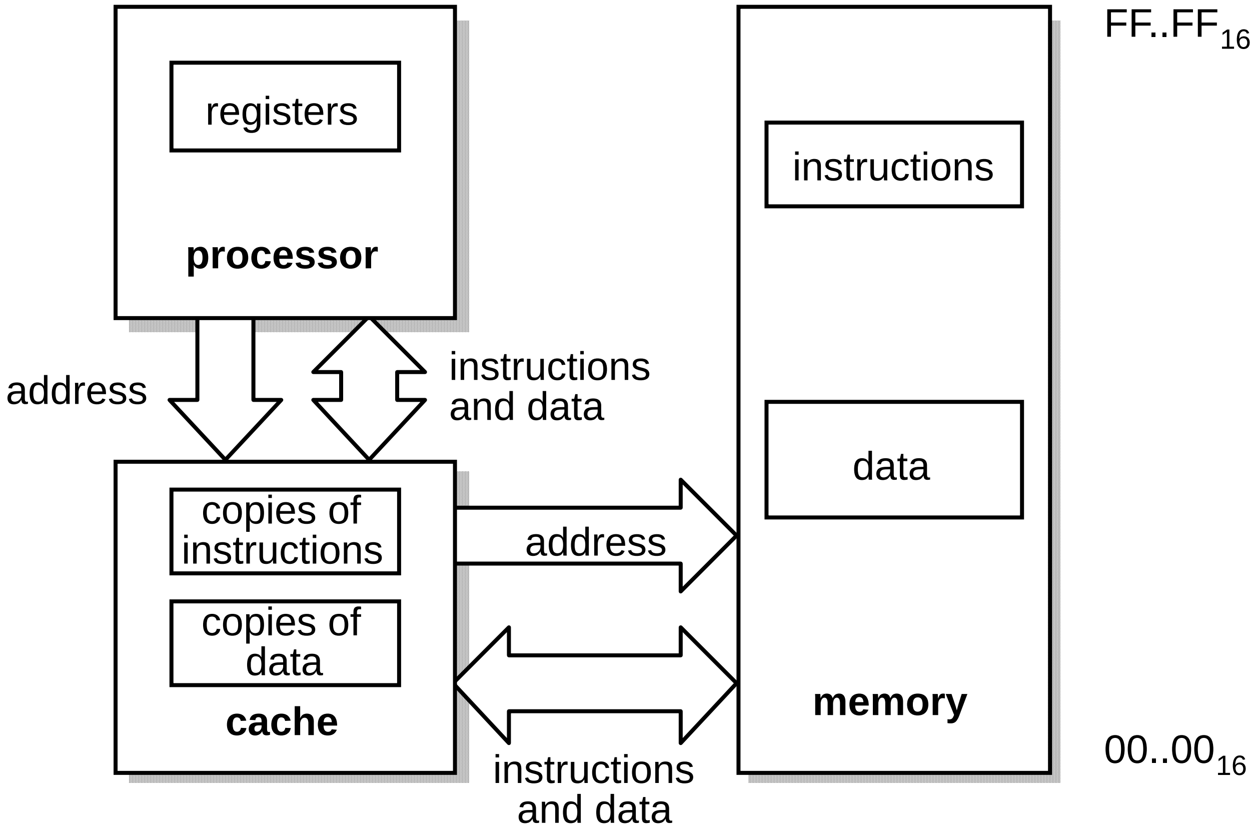


- Cache contains a copy of portions of main memory
- When CPU attempts to read a word, cache is checked first. If found, it is presented, else a line of words is brought from main memory to the cache

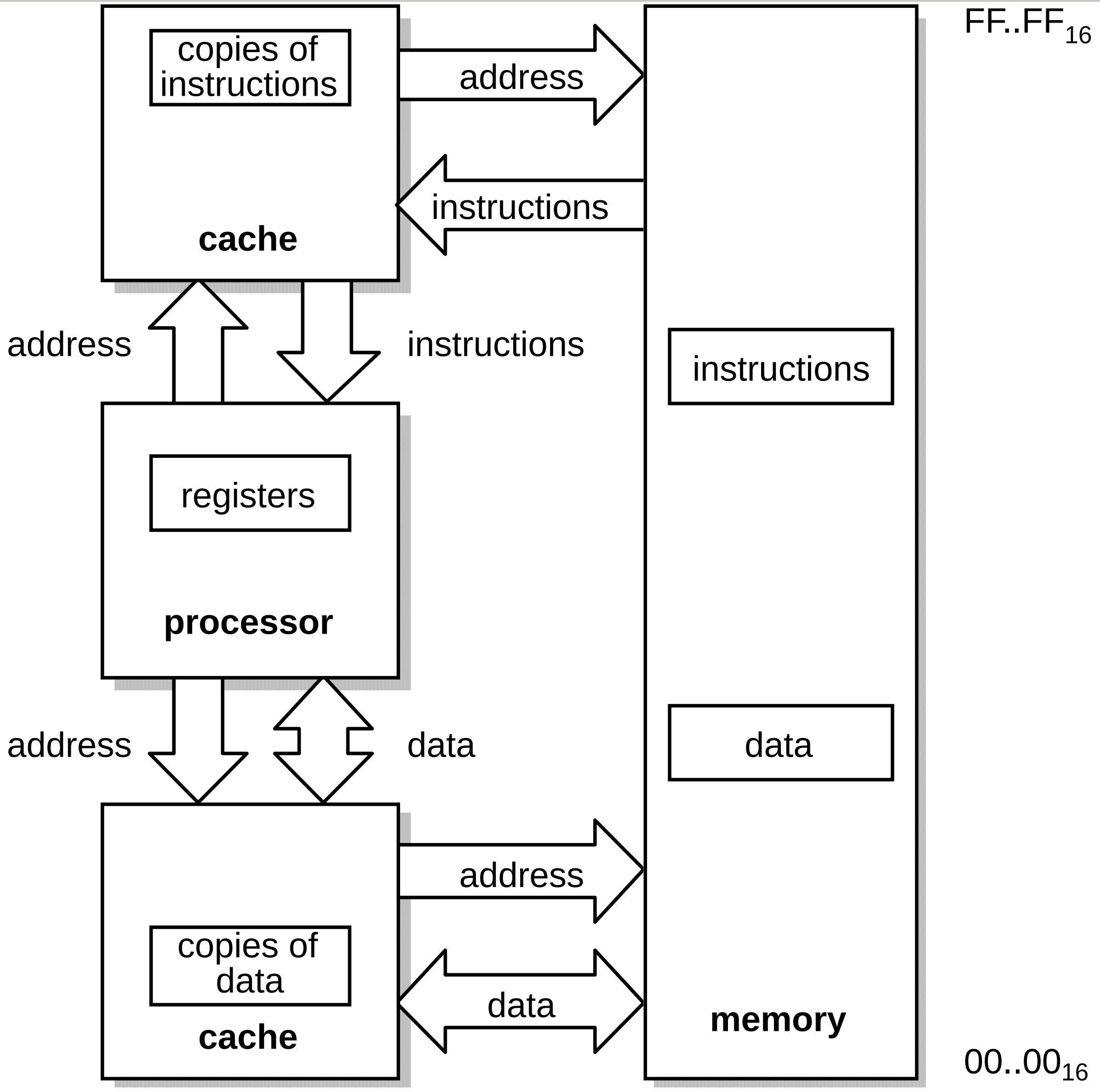
Cache Organisation

- At the highest level the processor can have two ways of organising Cache
  - – Unified Cache
    - Automatically adjust the proportion of cache memory for instructions, according to current programming requirements
  - – Separate instruction and data cache
    - Allows load and store instructions to execute in single clock cycle

Unified Instruction and data cache



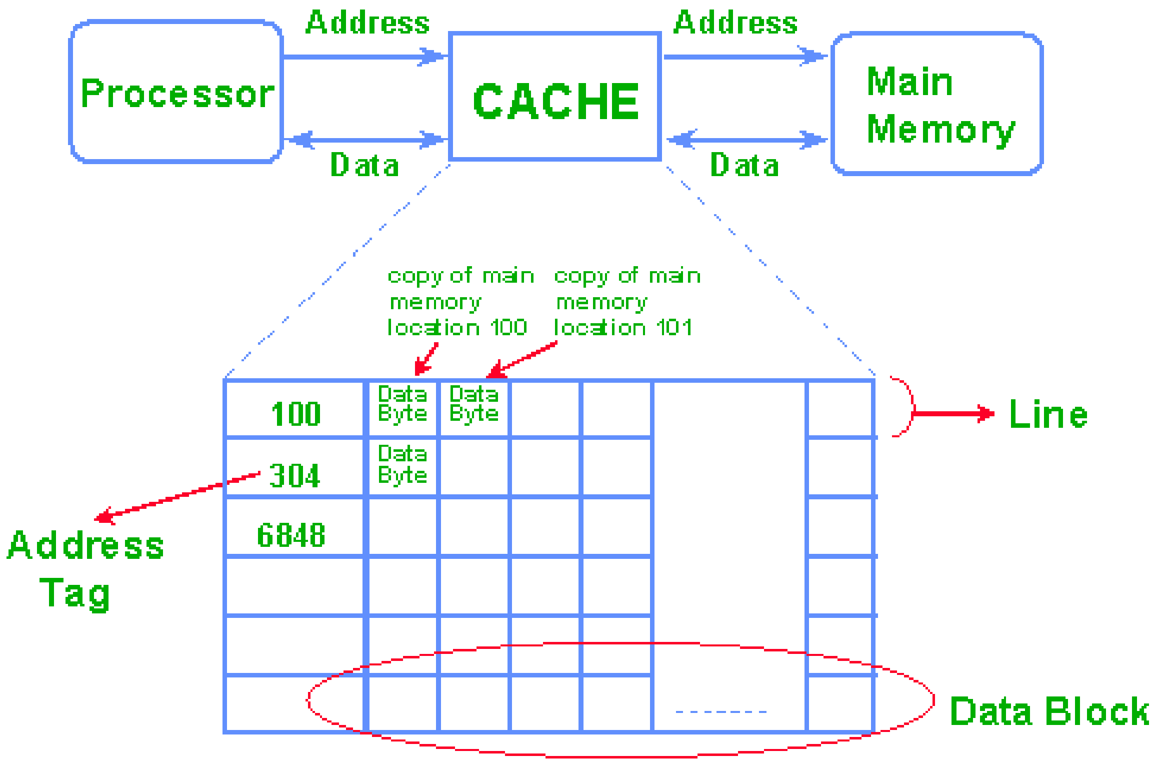
Separate Instruction and data caches



Basic Architecture of Cache

- A cache consists of 3 main parts
  - Directory store
    - The cache must know where the information in cache originated from
    - Each directory entry is called as cache-tag
  - Data section
    - Holds the data copied from main memory
  - Status information
    - Valid bid indicates live data
    - Dirty bit indicates non-coherency

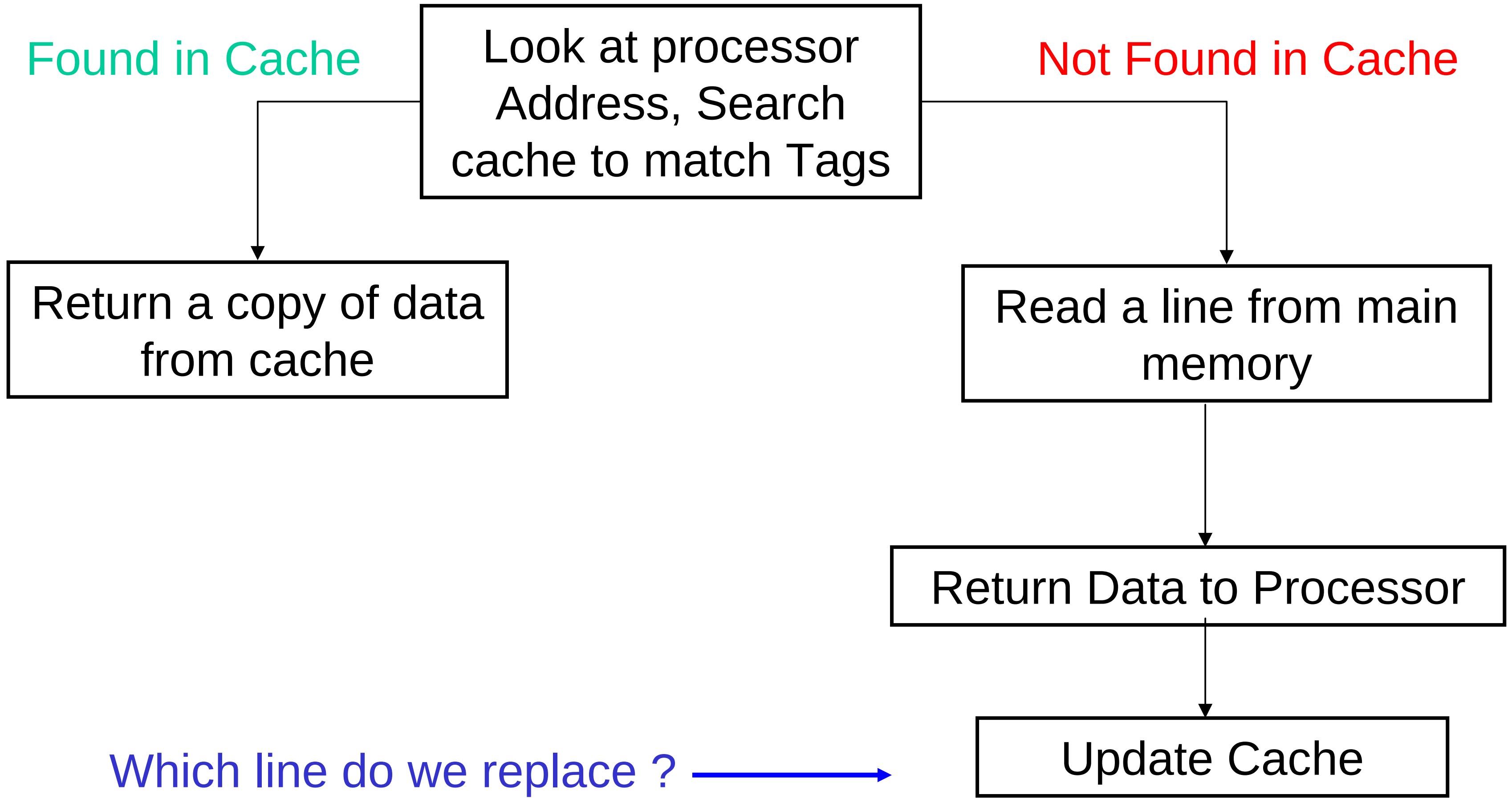
Inside a Cache



Cache Controller

- It’s the hardware that copies code or data from main memory to cache memory automatically.
- The operation is transparent to CPU and the software
- It intercepts reads and writes before passing them on to the memory controller.
- The relationship between main memory and cache is called as mapping

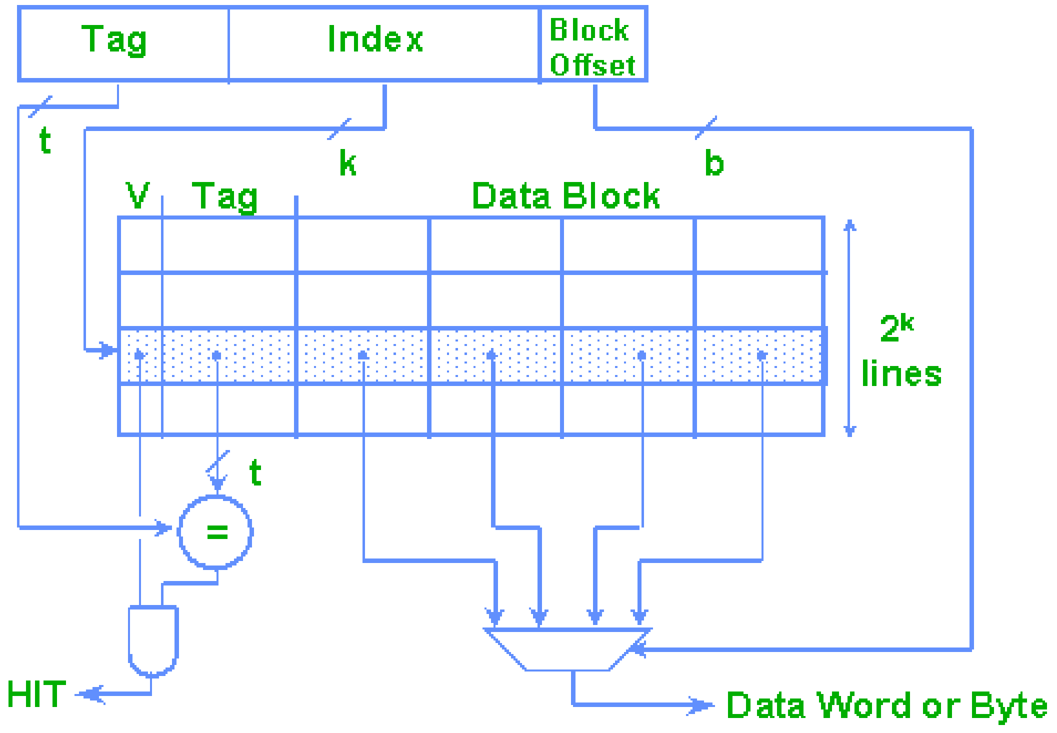
Cache Read Algorithm


Found in Cache
Look at processor Address, Search cache to match Tags
Not Found in Cache
Return a copy of data from cache
Read a line from main memory
Return Data to Processor
Which line do we replace ?
Update Cache

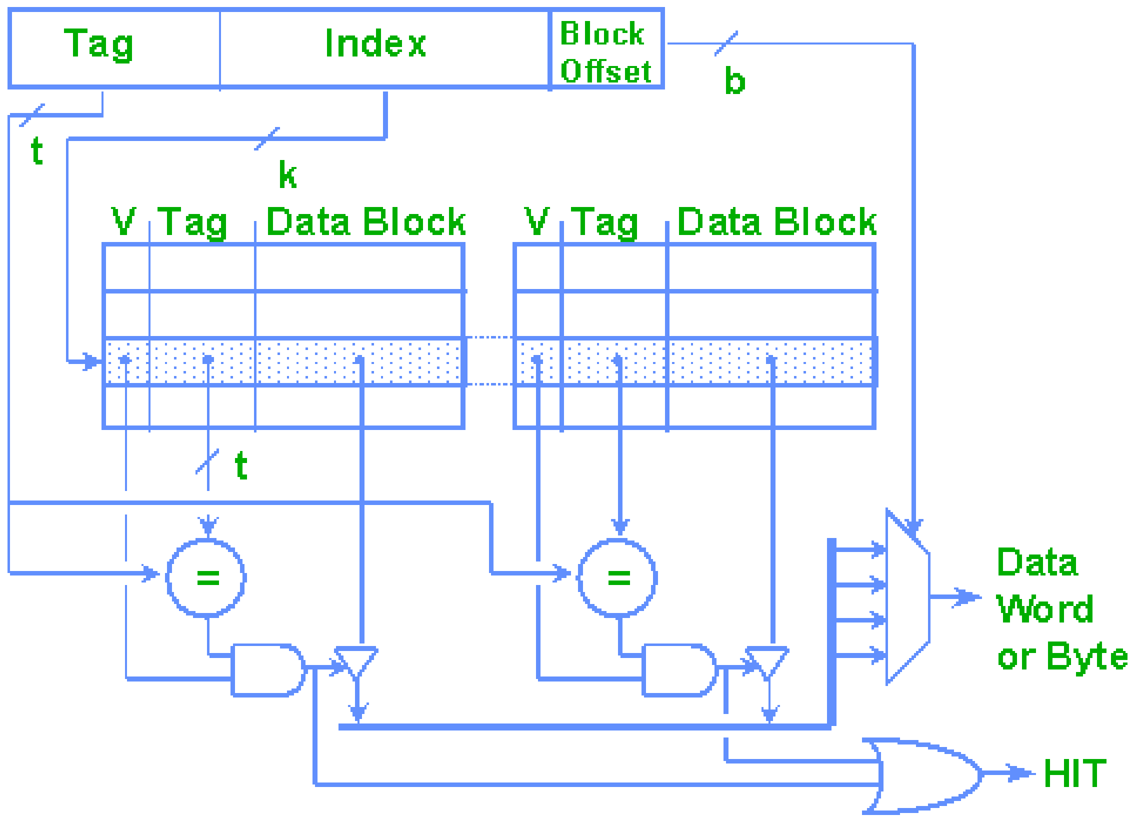
Cache Mapping

- Mapping between main memory locations and cache
  - Direct mapped
  - N-way set-associative (n=2,4)
  - Fully set-associative

Direct mapped



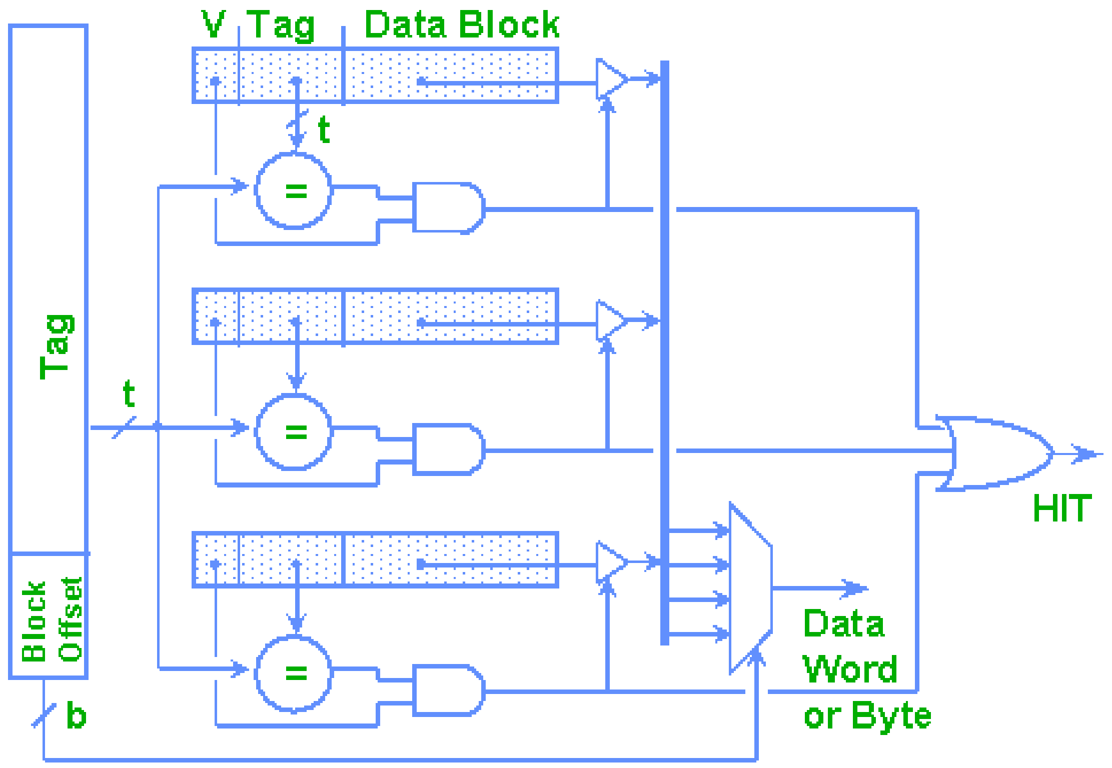
2-way Set Associative Cache



2-way Set Associative Cache

- Improved direct mapped cache
- Each memory line has two destinations in cache
- Extra logic is required for comparing tags
- Replacement policy needs consideration

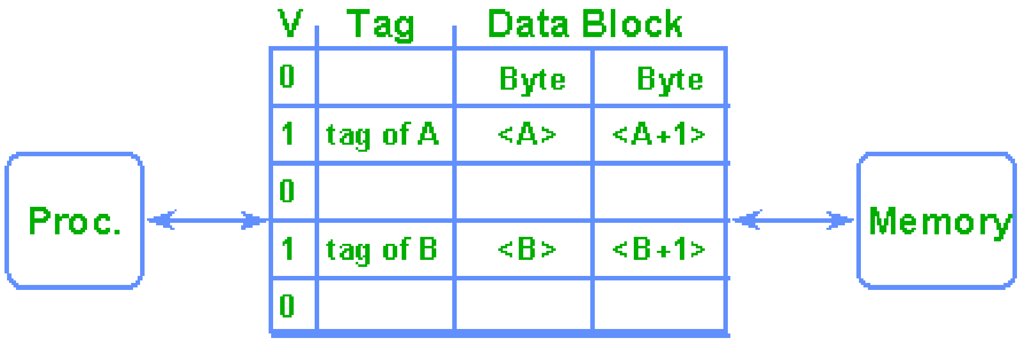
Fully Associative Cache



Fully Associative Cache

- This scheme represents other extreme
  - Any line can be placed in any location in cache
- Gives highest hit ratio
- Search is slowest
- Content addressable Memories (CAM) are used for storing Tags

Valid bits



- Valid bit must be 1 for cache line to HIT.
- At power-up or reset, we set all valid bits to 0.
- Set valid bit to 1 when cache line is first replaced.
- Flush cache by setting all valid bits to 0, under external program control.

Cache Efficiency

- Two terms are use to characterise cache efficiency of a program
  - Cache hit rate
  - Cache miss rate

$$\text{Hit rate} = \frac{\text{cache hits}}{\text{memory requests}} \times 100$$

  - Rates can measure read, writes or both
  - Other performance measurement terms are hit time and miss penalty

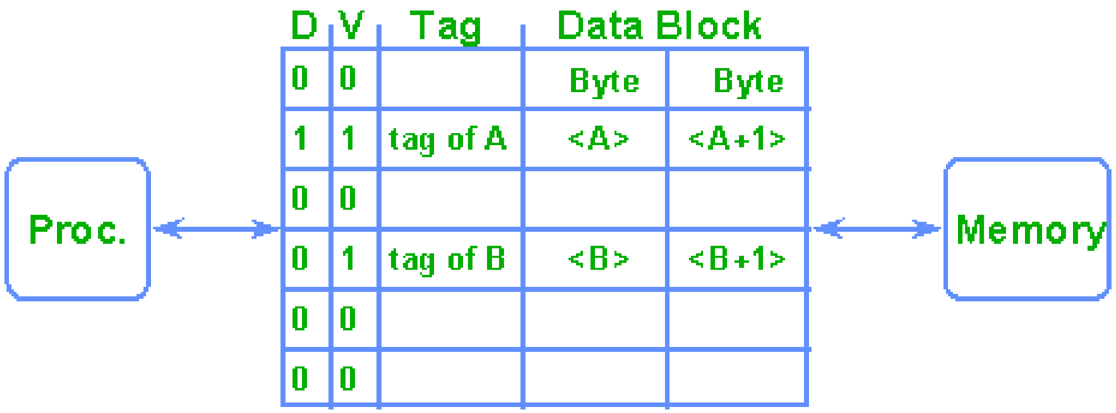
Cache Policy

- Three policies determine cache operation
  - Write policy
    - Determines where data is stored during processor write operation
  - Replacement policy
    - Determines the line that will be used for next line fill during a cache miss
  - Allocation policy
    - Determines when the cache controller allocates a line

Write Policy

- Two alternatives when processor issues a write
  - Update both cache and main memory: write-through
  - Update only the cache: write-back
- Write-through
  - Both memories remain coherent all the time
  - Writes are slower
- Write-back
  - Valid cache lines and memory may not be coherent
  - Dirty bits are used to indicate non-coherency
  - Dirty cache line are updated before eviction

Dirty bits for write-back caches



D	V	Tag	Data Block	
0	0		Byte	Byte
1	1	tag of A	<A>	<A+1>
0	0			
0	1	tag of B	<B>	<B+1>
0	0			
0	0			

When the line corresponding to A is replaced, its data block has to be written to main memory since its dirty bit is set.

Replacement Policy

- After a read miss, which cache line should be replaced with the data line read from main memory
  - Direct mapped cache: unique line
  - Associative cache
    - Least Recently Used(LRU): Replace line in set corresponding to address which has not been read or written for the longest time
    - First-in First Out(FIFO): Select line which has been in the set for the longest time
    - Random: Select line in the set randomly

Allocation Policy

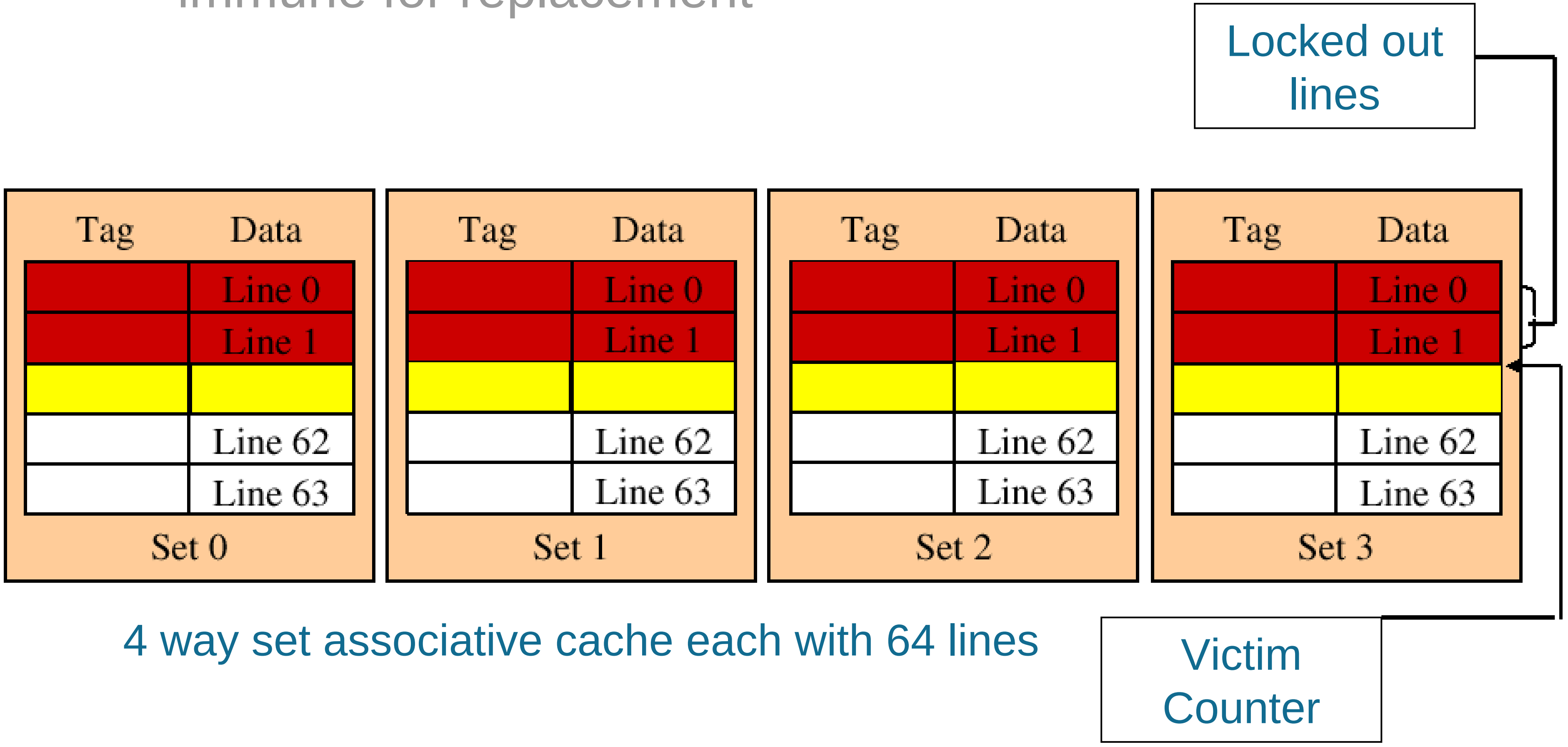
- Two strategies to allocate cache line on a cache miss
  – Read-allocate
    - Allocation is done during read from main memory
    - A write does not update cache unless a line was allocated on previous read
  – Read-write-allocate
    - Allocation is done during either a read or write
    - On write, if cache line is not valid, it does a cache fill before updating the cache.
  – Write policy determines whether main memory should be updated

Cache Lockdown

- A feature that enables a program to load time-critical code and data into cache and exempt it from eviction
- It avoids the problem of unpredictable execution times that result from line replacement
- Cache used for lockdown results in reduction of cache available for other parts of main memory
- Candidates for locking in cache
  - Vector interrupt table
  - Interrupt service routines
  - Critical code used frequently
  - Frequently used global variables

Cache lockdown

- Lines below chosen by victim counter are immune for replacement



4 way set associative cache each with 64 lines

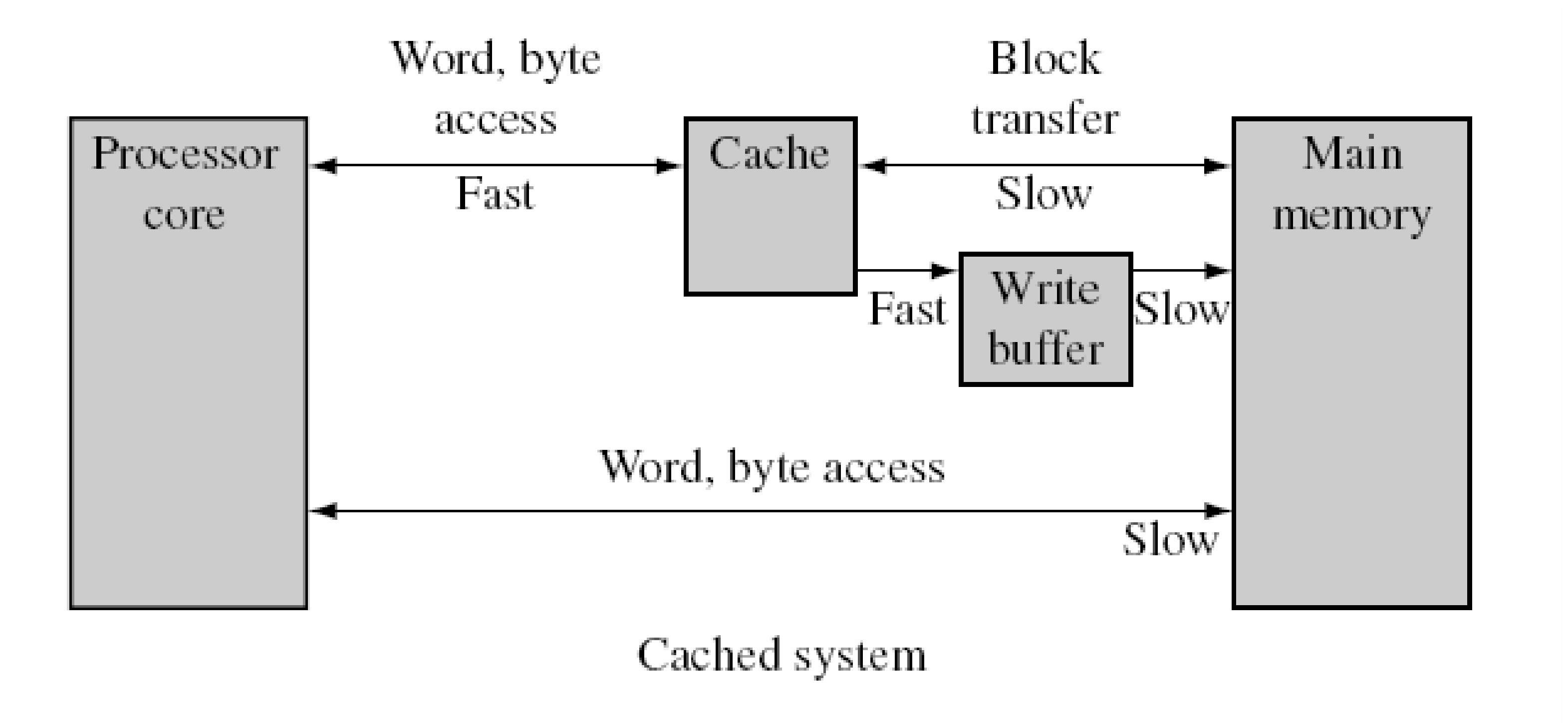
Write Buffers

- Ensures that memory is decoupled from core
  - Data placed in buffer at core speed
  - In parallel data written to memory at bus speed
- Access are regulated to occur in correct order
  - Dedicated write buffer
  - Write buffer is always drained first for non cached reads, cache line fills and non-buffered writes

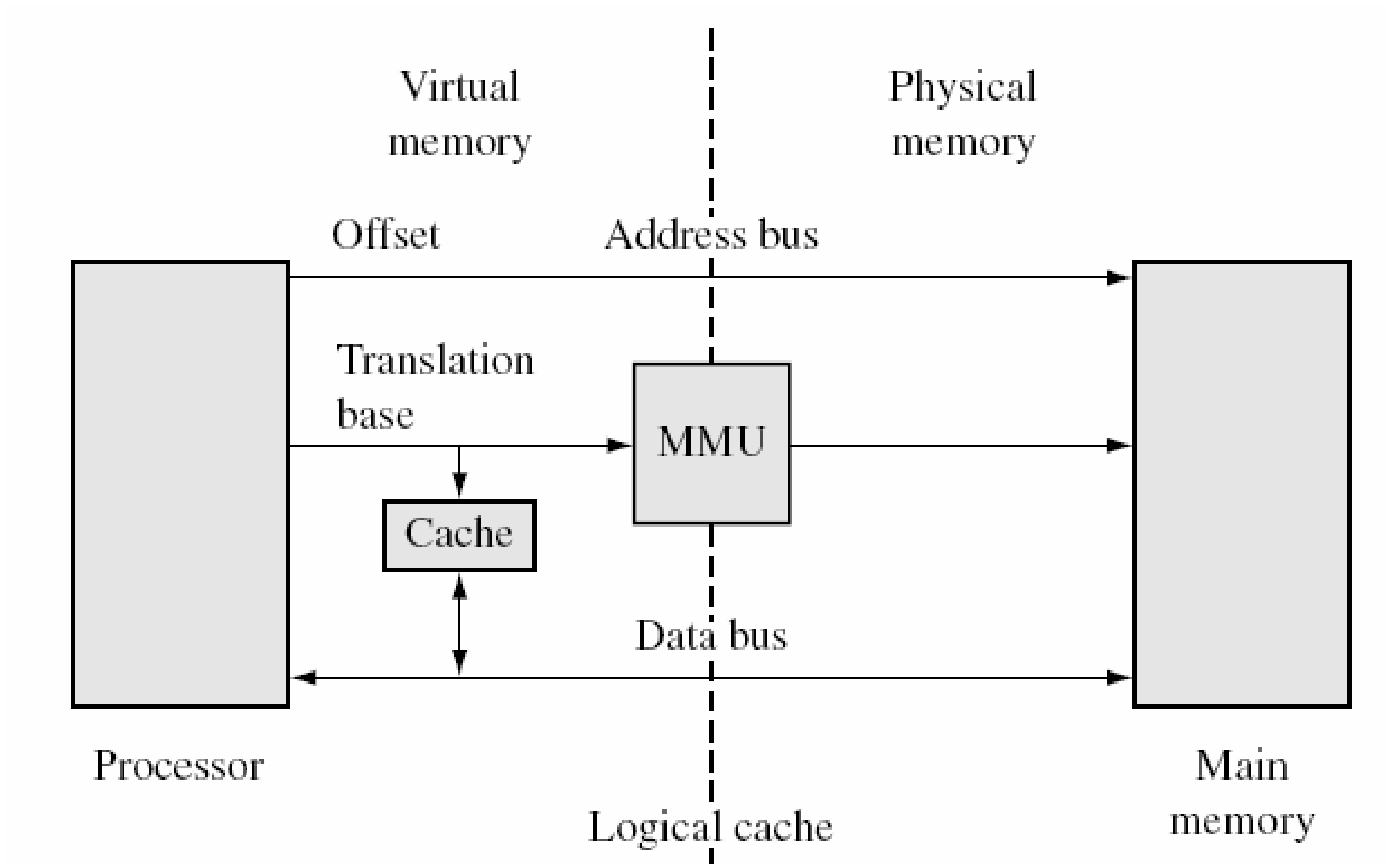
Tightly coupled memories

- Fast memory local to processor
  - High speed performance without access to system bus
  - Real time performance can be predicted
- Alternative to cache
  - Smaller die size and low power
- Appears at fixed locations in memory area
  - Some cores have both TCM and cache, some TCM
  - Different TCMs for instruction and data

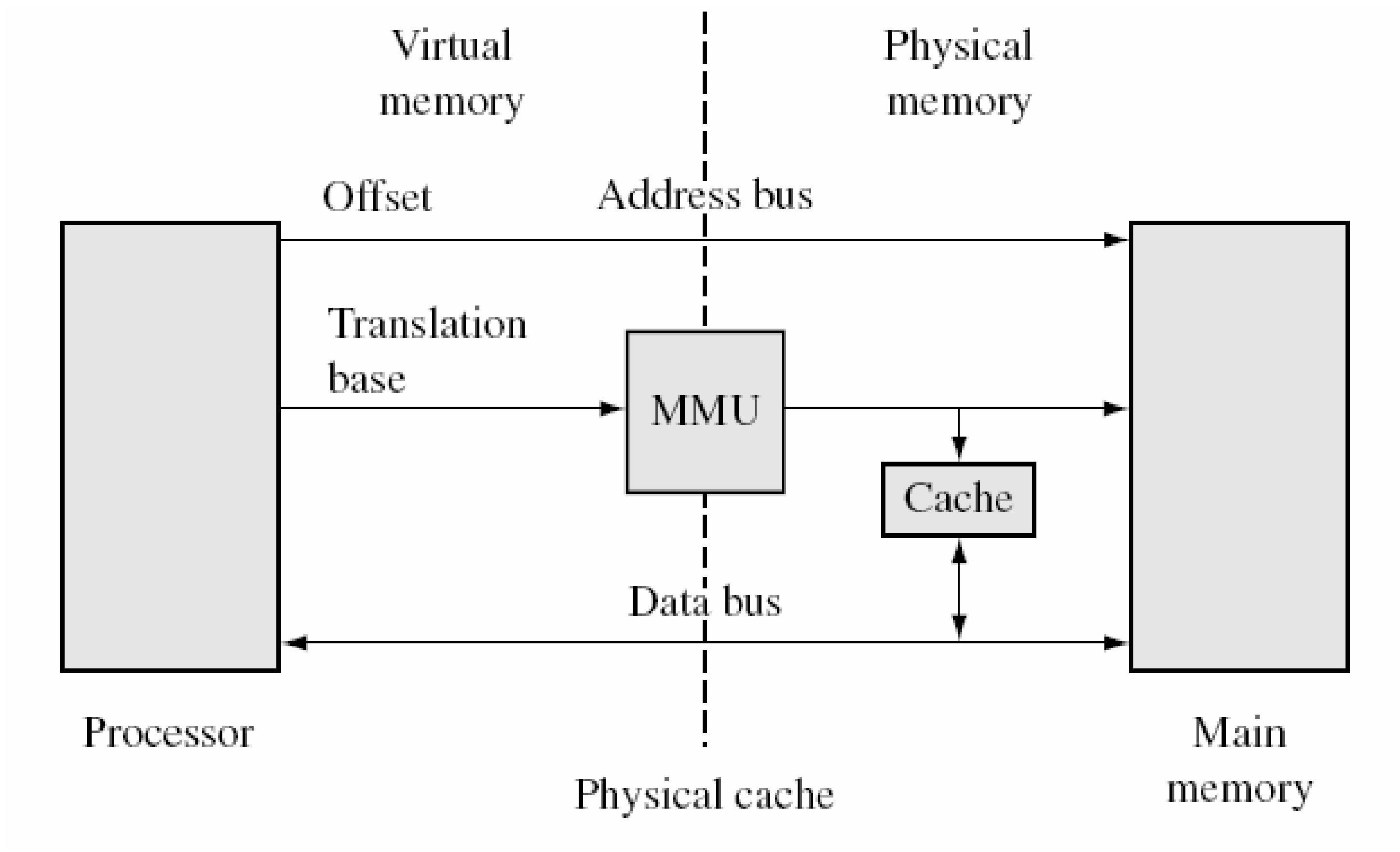
Write Buffers



Logical Cache



Physical Cache

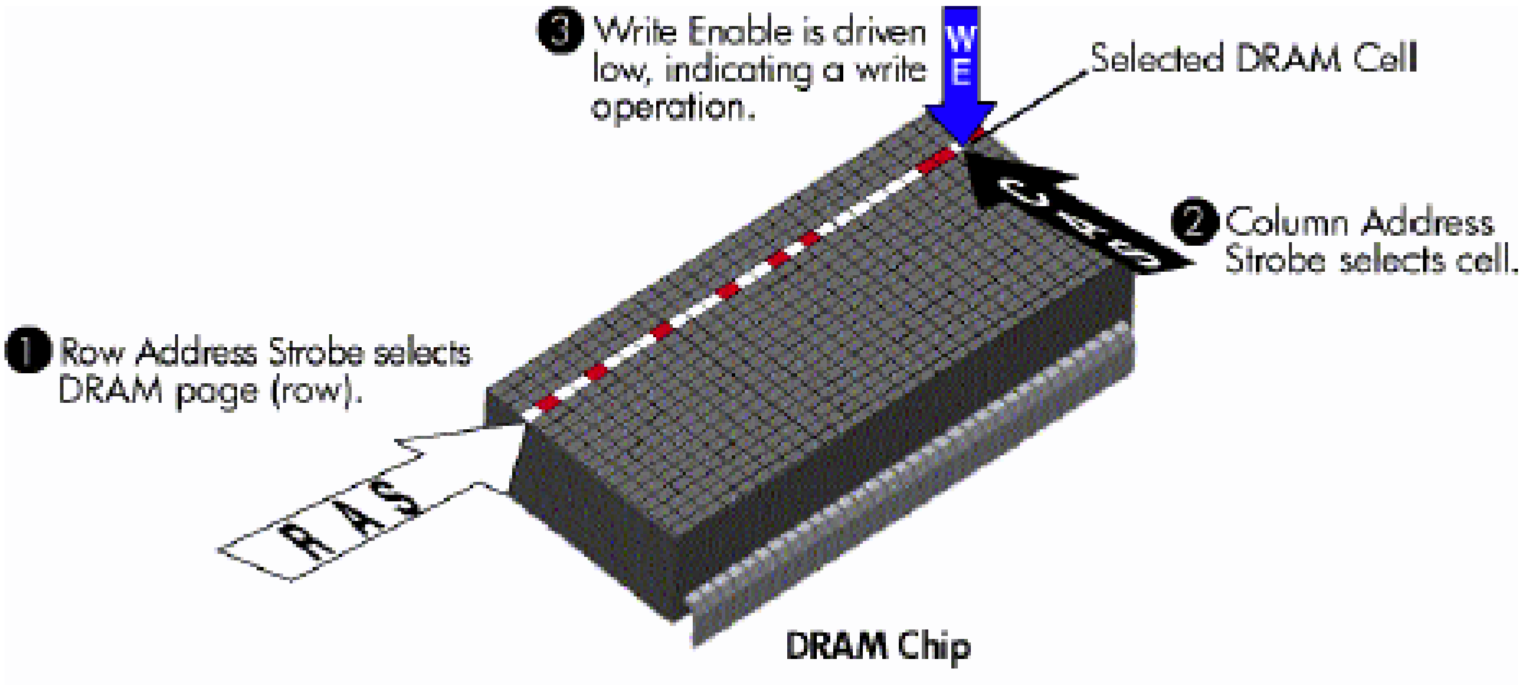


Main Memory

DRAMs

- Over the years evolution of system memory has taken place
  - Asynchronous DRAM technologies,
    - Fast Page Mode (FPM) memory and
    - Extended Data Out (EDO) memory,
  - synchronous DRAM (SDRAM)
    - Registered SDRAM
    - Double Data Rate (DDR) SDRAM
    - DDR – II SDRAM
    - Rambus DRAM (RDRAM)
- Yet, system memory bandwidth has not kept pace with improvements in processor performance

Basic DRAM operation


3 Write Enable is driven low, indicating a write operation.
WE
Selected DRAM Cell
2 Column Address Strobe selects cell.
CAS
1 Row Address Strobe selects DRAM page (row).
RAS
DRAM Chip

Asynchronous DRAMs

- Address is provided in two parts qualified by
  - Row Address Strobe (RAS)
  - Column Address Strobe (CAS)
- Data is available after CAS is asserted
- Memory controller asserts signals to suit DRAM timing specifications
- Latency is longer due to transfer of data betwen asynchronous DRAMs and synchronous system bus

FP and EDO DRAMs

- All locations having same row address is a page
- FP DRAMs allows any location in current page to be access by strobing in only column address with a CAS.
- This reduces latency for subsequent accesses

- EDO DRAMS maintain the output for longer period even when CAS is de-asserted.
- This improves overall system efficiency

Synchronous DRAMs

- Memory bus clock is used to synchronize the input and output signals on the memory chip
  - Simplifies the memory controller
  - Reduces the latency from CPU to memory
- Memory is divide into two to four banks for simultaneous access to more data
- The bandwidth capacity of the memory bus increases with its width (in bits) and its frequency (in MHz).
- For a system transferring 8 bytes (64 bits) at a time and running at 100 MHz, the bandwidth is 800 MB/s,

Memory Management and Protection

Memory Addresses

- The CPU supports a variety of addressing modes
- The address generated by address unit of CPU refers to a logical space of contiguous locations
- The memory presents itself as a contiguous space of physical locations
- A relationship needs to be established between the logical address space of CPU and the physical address space presented by memory.
- In CPUs like 8085, the mapping between the two is one is to one.

Constraints for memory management

- User programs need logical space that is much bigger than the available physical memory
- Assignment and sub-division of physical address space to a number of different user programs
- The protection of the physical address area assigned to a user from possible interference by other user programs

Solutions

- To allow large logical address space, virtual memory used.
  - Address space is mapped onto secondary storage like disks
  - Locality principle helps in keeping frequently used program segments into main memory
- To allow assignment of physical memory to programs dynamic relocation is used
  - Logical addresses are assigned physical address at the time of loading

Solutions

- To prevent unauthorized programs from accessing memory areas
  - Each physical memory area is provided with access attributes
    - Read, write, execute etc

Mapping

- It sets the relationship between logical and physical addresses
- In the simplest of schemes
  - The complete program is treated as single area
  - It is allotted a single contiguous area in physical memory
  - Mapping parameters are base address and size
    - Physical address = base + logical address
  - This method has problems related to compacting and efficient use of physical memory

Dynamic mapping

- Program is treated as number of separate address areas inside which contiguity is observed
- In main memory only those parts of program are maintained which are ‘active’
- This scheme needs description of where each segment of logical address is placed in physical memory
- This is achieved by using two-component address
  - The address area
  - The offset

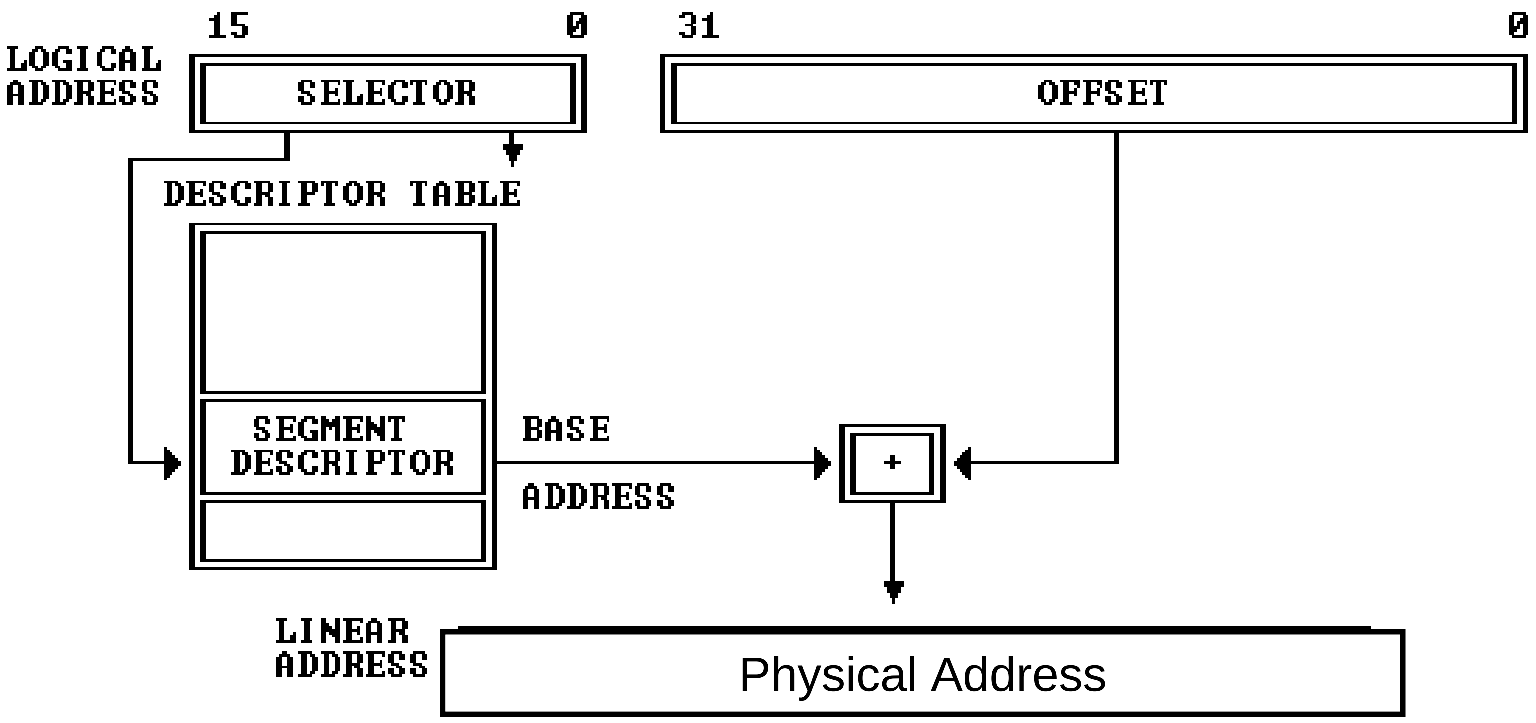
Mapping Methods

- Segmentation
  - The logical address space is broken into units called segments
  - Each segment has base address, Size and attributes
- Paging
  - Memory is divided into pages of constant length
  - Pages are moved between logical and physical address space
- Tables are maintained to achieve the mapping

Segmentation

- To perform translation, the processor uses the following data structures:
  - Segment Descriptors
  - Descriptor tables
  - Selectors
  - Segment Registers

Segment Translation



Segment Descriptors

- The segment descriptor provides the processor with the data it needs to map a logical address into a linear address.
- Descriptors are created by compilers, linkers, loaders, or the operating system
- Descriptors are not created by applications programmers.

Segment Descriptor

Each Segment is represented by a 8 byte segment descriptor

Attributes	Base (16 – 23)	Base (16-31)
Base (0-15)		Limit (0-15)

32 bit Base field : contains linear address of the first byte of the segment

24 bit limit field : denotes segment length

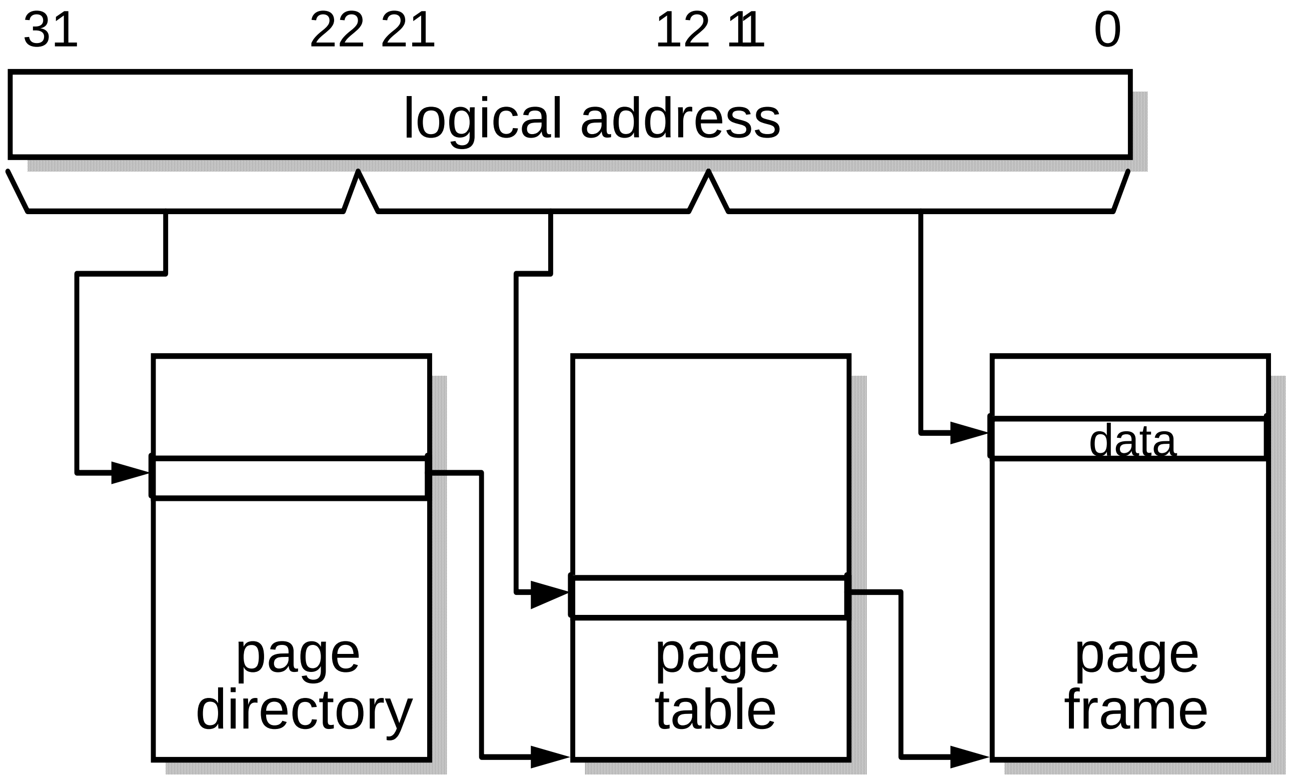
Attributes: Protection, privilege levels etc

Paging Unit

Paging

- Both logical and physical address space is divided into fixed sized components called pages
- Relationship between logical and physical pages is stored in page tables.
  - Single table
    - Requires very large table
  - Two or more levels of page tables

Paging memory management



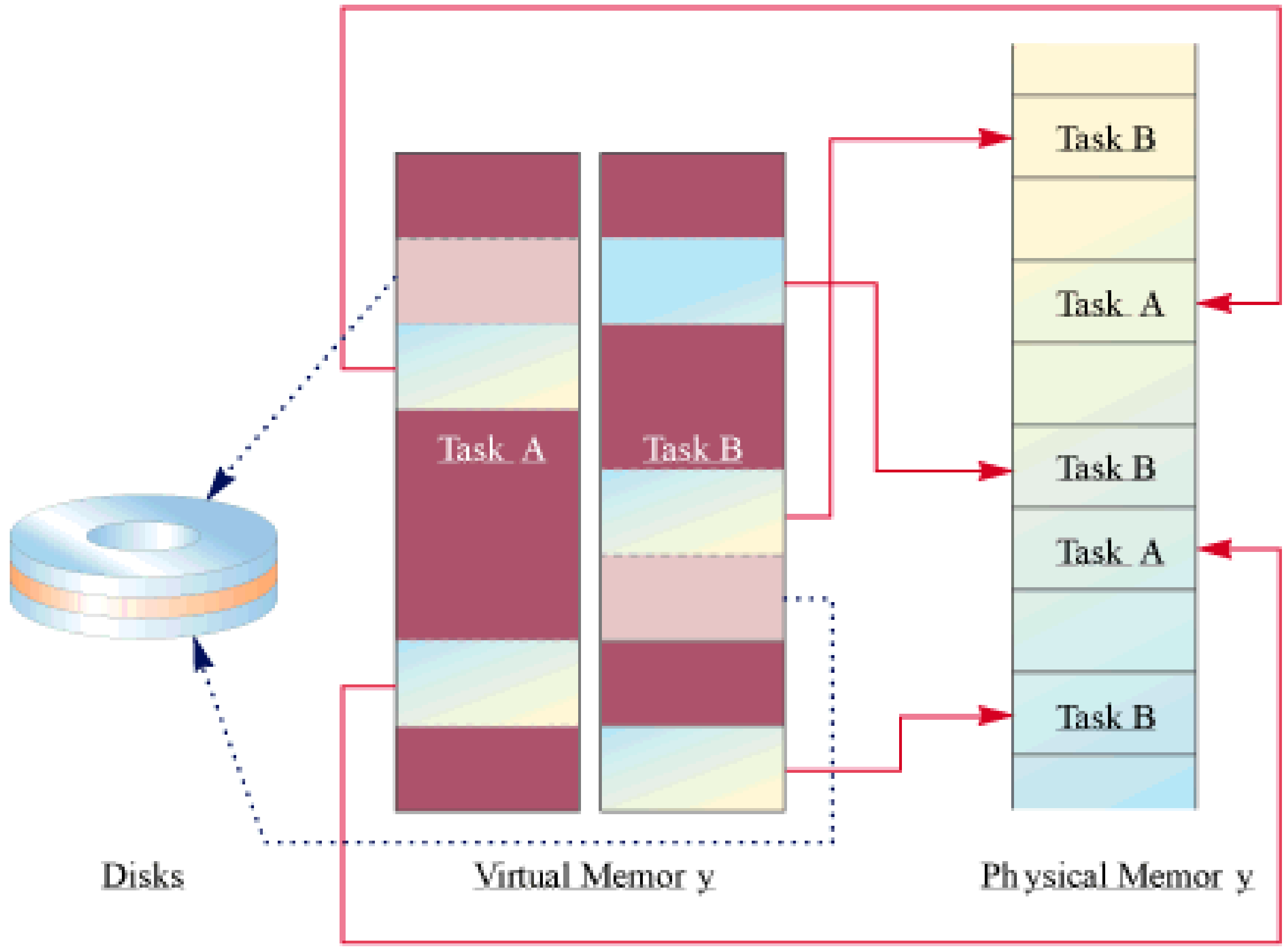
Virtual Memory

- Two motivations
  - To allow efficient sharing of memory among multiple programs
    - A collection of programs are running at a time
    - Total memory requirement of all would be more
    - Fraction of this is being used at a time
  - To remove the programming burden of small, limited amount of main memory
    - Programmers divided large programs into pieces and identified mutually exclusive ones
    - These overlays were loaded and unloaded during execution

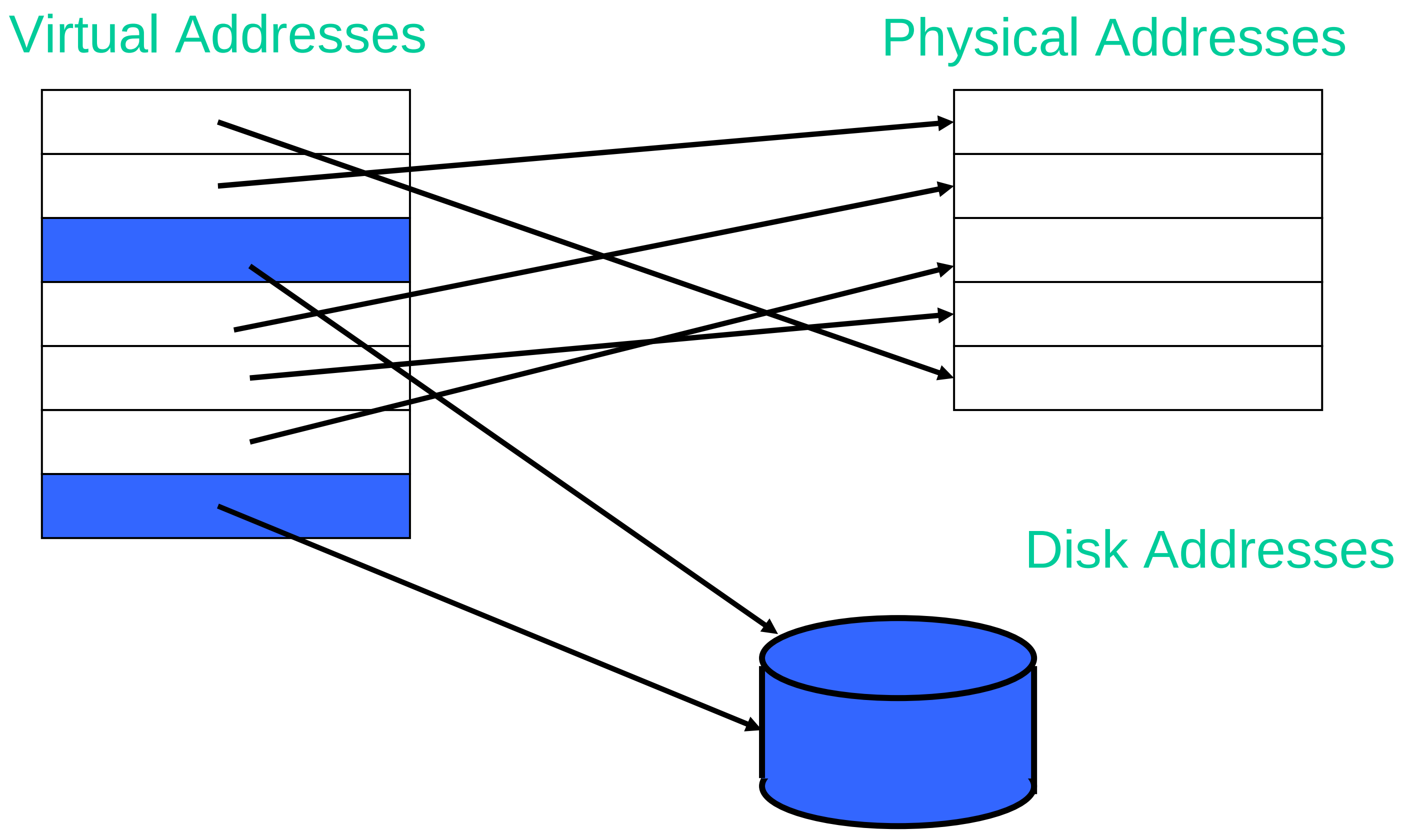
Virtual Memory

- Functions of Virtual Memory subsystem
  - Translation
    - From a program’s own address space to physical memory address
  - Protection
    - Same physical memory is shared between various programs and the OS

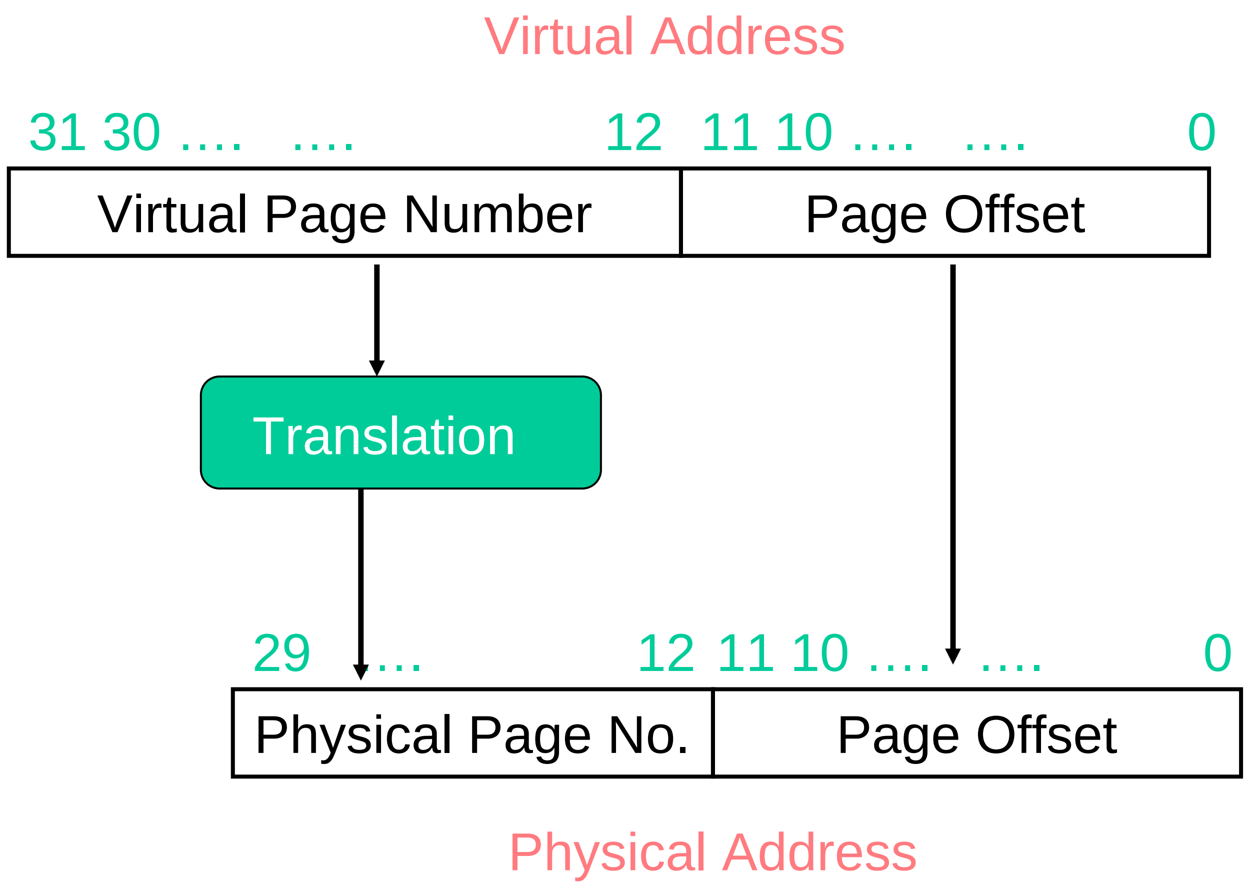
Features


Task A
Task B
Task B
Task A
Task B
Task A
Task B
Disks
Virtual Memory
Physical Memory

Address Translation

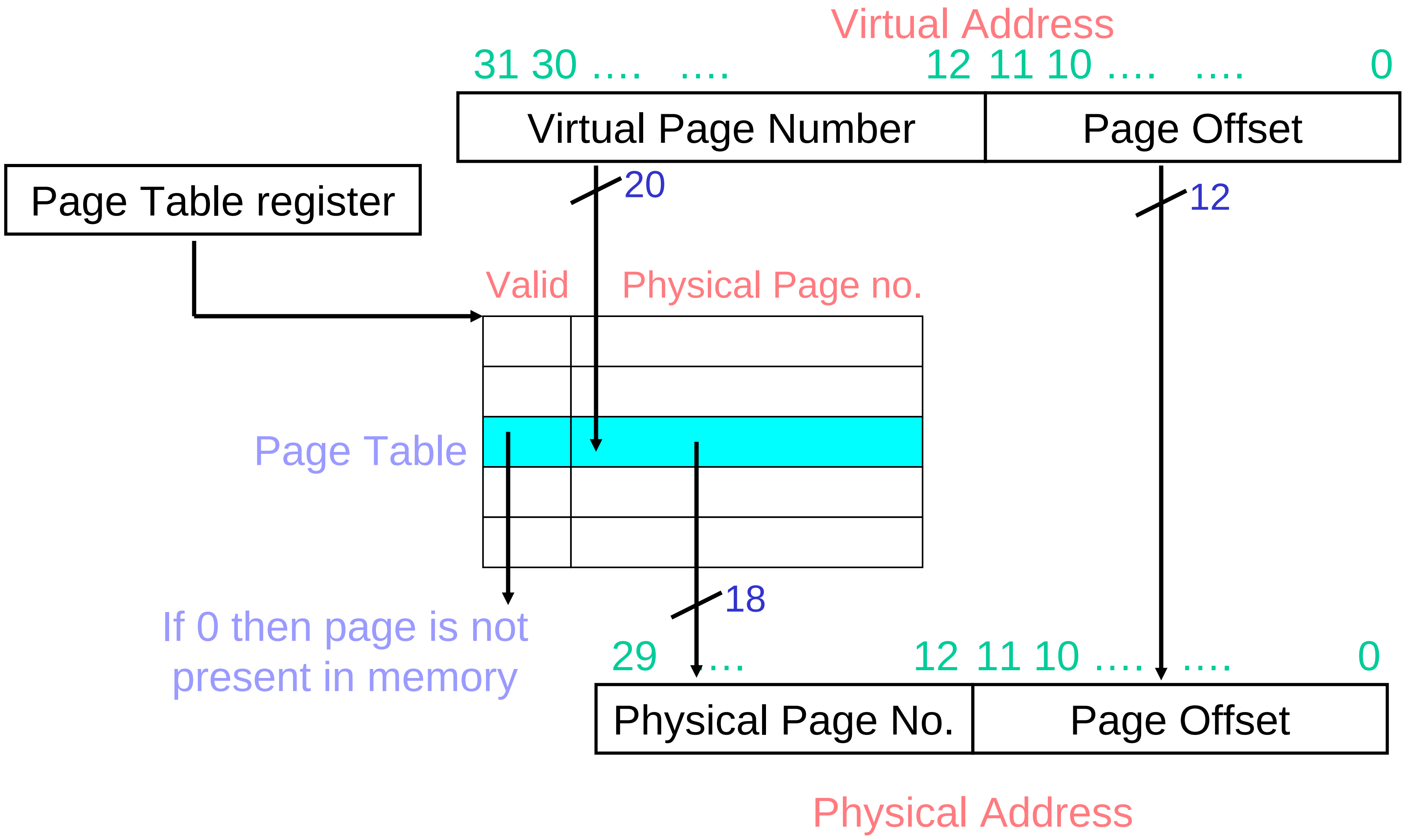


Mapping



- Page size is 4Kb
- Physical pages allowed is 2^18
- Main memory is 1 GB
- Virtual Address space is 4 GB

Mapping


Virtual Address
31 30 …. ….
12 11 10 …. ….
0
Virtual Page Number
Page Offset
Page Table register
20
12
Valid
Physical Page no.
Page Table
If 0 then page is not present in memory
18
29 …
12 11 10 …. ….
0
Physical Page No.
Page Offset
Physical Address

Page Faults

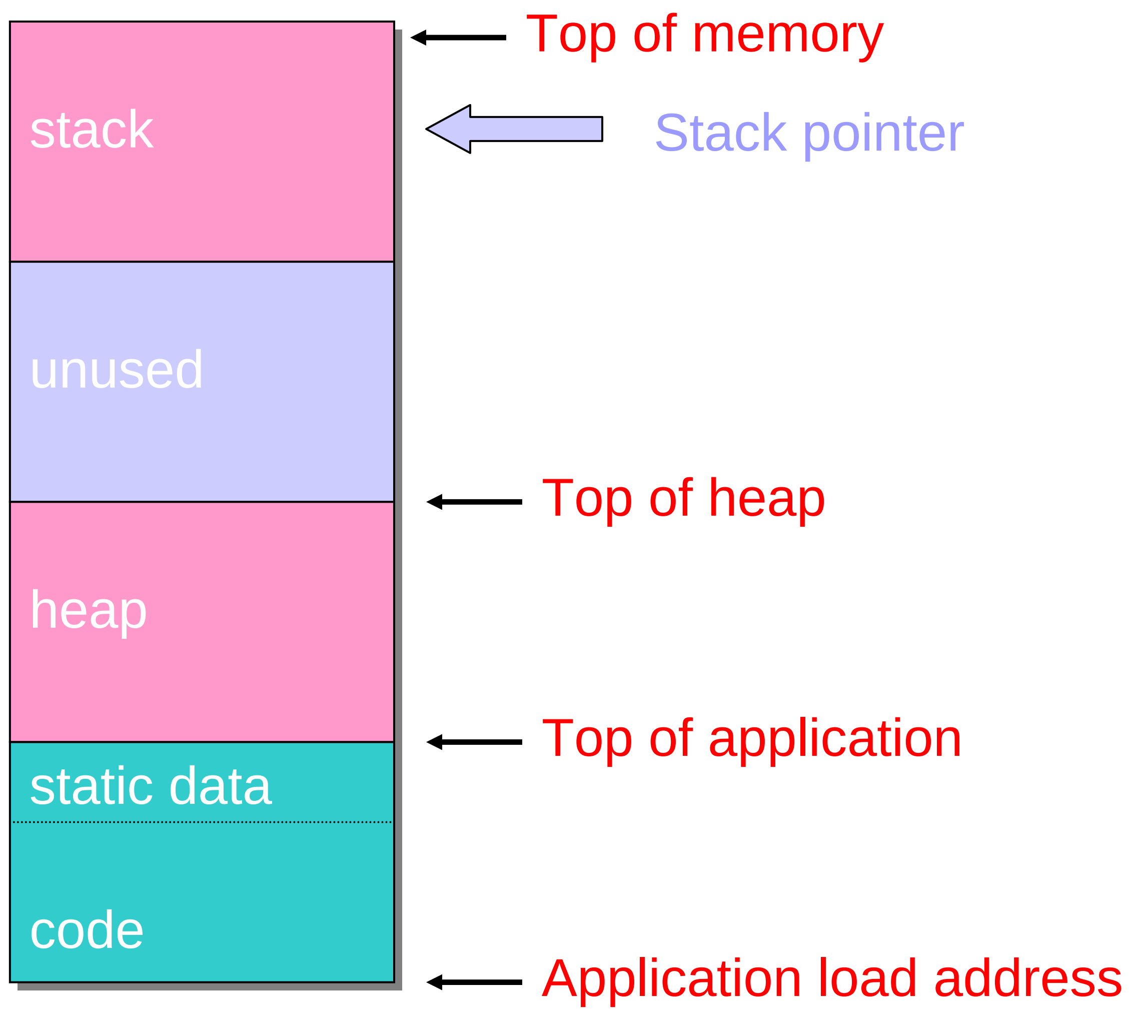
- If the valid bit is off, page fault occurs
- Page faults are very costly, in terms of time
- This results in exception and kernal gets the control
- Kernal finds the page on the disk
- Decides where to put the page in main memory
  - Generally LRU is used
- Updates the page table
- To make translations fast, Translation Look aside Buffer (TLB)is used
- TLB acts like a cache for the page table.

Memory Subsystem of ARM

Memory map

- Arranged as a linear set of logical address
- A ‘C’ program expects to access
  - A fixed area of program memory
  - A dynamically changing data areas
    - Stack: for local variables and function calls
    - Heap: to satisfy program requests for more memory for data structures
- Application is loaded at lowest address
- Unused memory is allocated on demand to heap or stack

ARM address space model



Data Storage

- Basic data types

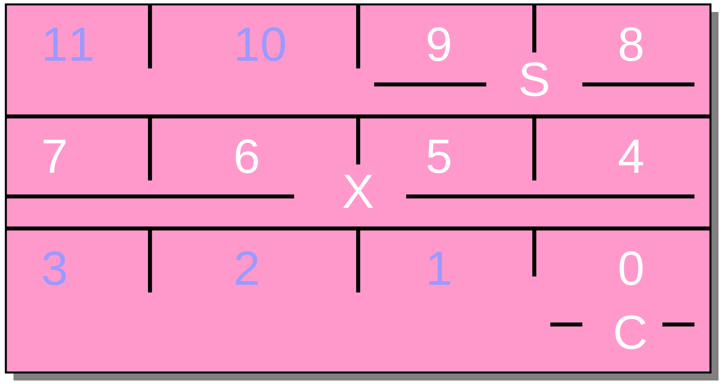
– Char:	byte
– Short int:	half word
– Int, single precision float:	word
– Double precision float:	multiple words

- Derived data types like structs,arrays, unions etc. are defined in terms of basic data types
- ARM instruction set is efficient in loading and storing data items when they are properly aligned in memory
- Storing a word to non-word-aligned memory is very inefficient: can take upto 7 ARM instructions

Data Alignment

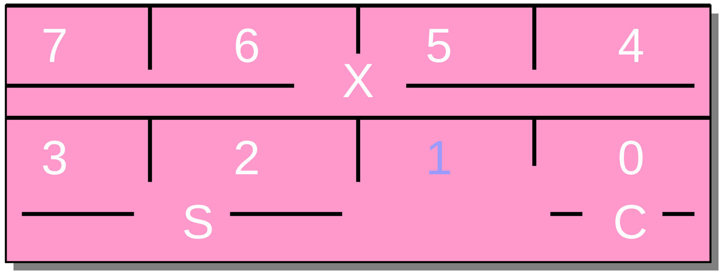
- ARM C compiler generally aligns data on appropriate boundaries
  - Bytes at byte address
  - Half-words at even byte address
  - Words at four-byte boundaries
- When several data items of different types are declared at the same time, compiler introduces padding

Memory Efficiency



Struct S1 (char c; int x; short s;) example1

Occupies 3 words



Struct S1 (char c; short s; int x;) example2

Occupies only 2 words

Memory in ARM based systems

- Wide variety of embedded systems use ARM
  - Memory system requirements vary from
    - Simple memory block with flat address map
    - Using one of more of memory resources
  - Memory resources
    - Multiple types of memories
    - Caches
    - Write buffers
    - Virtual memory and remapping techniques
    - Memory mapped IO

Memory in ARM based systems

- Initialising and controlling memory system
  - Enable cache, configure it for best performance
  - Set up virtual to physical address mapping
  - Restrict access to memory regions
  - Ensure proper access to memory mapped IO
- The standard way to perform memory system control is CP15

Thank You