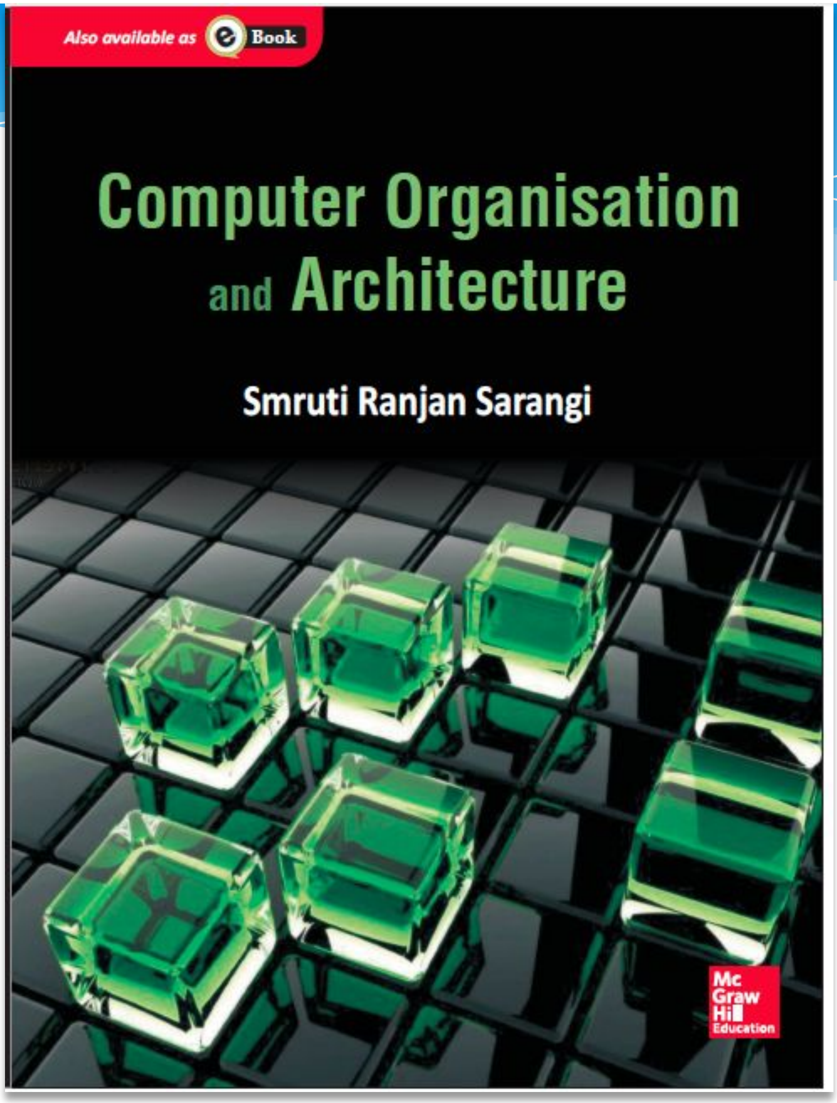
PowerPoint Slides

Computer Organisation and Architecture

Smruti Ranjan Sarangi,
IIT Delhi

Chapter 10 The Memory System



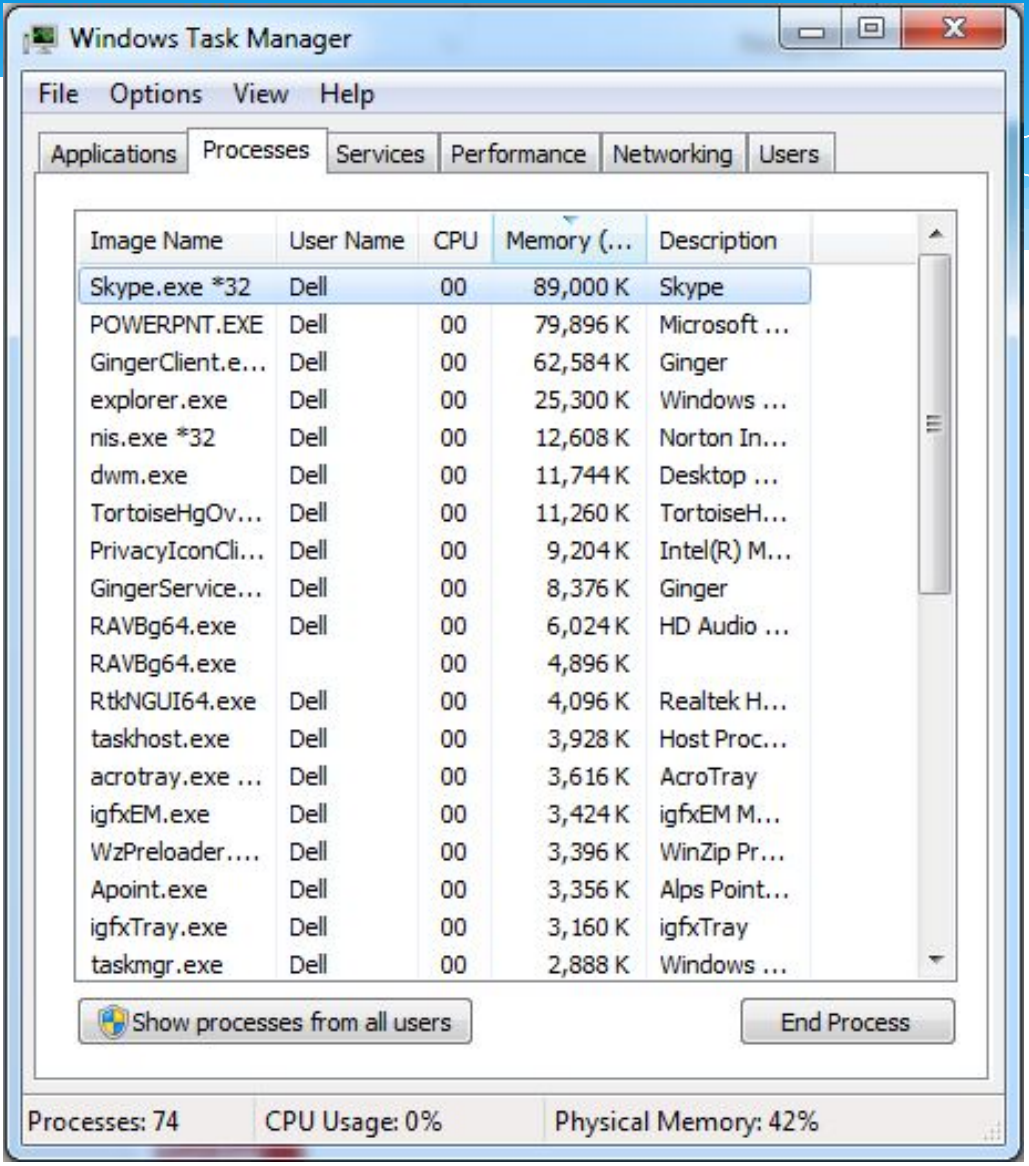
These slides are meant to be used along with the book: Computer Organisation and Architecture, Smruti Ranjan Sarangi, McGrawHill 2015
Visit: http://www.cse.iitd.ernet.in/~srsarangi/archbooksoft.html

Outline

* Overview of the Memory System
* Caches
* Details of the Memory System
* Virtual Memory

Need for a Fast Memory System

- We have up till now assumed that the memory is one large array of bytes
  - Starts a 0, and ends at $(2^{32} - 1)$
  - Takes 1 cycle to access memory (read/write)
- All programs share the memory
  - We somehow magically avoid overlaps between programs running on the same processor
  - All our programs require less than 4 GB of space



Windows Task Manager

File Options View Help

Applications | Processes | Services | Performance | Networking | Users

Image Name	User Name	CPU	Memory (...	Description
Skype.exe *32	Dell	00	89,000 K	Skype
POWERPNT.EXE	Dell	00	79,896 K	Microsoft ...
GingerClient.e...	Dell	00	62,584 K	Ginger
explorer.exe	Dell	00	25,300 K	Windows ...
nis.exe *32	Dell	00	12,608 K	Norton In...
dwm.exe	Dell	00	11,744 K	Desktop ...
TortoiseHgOv...	Dell	00	11,260 K	TortoiseH...
PrivacyIconCli...	Dell	00	9,204 K	Intel(R) M...
GingerService...	Dell	00	8,376 K	Ginger
RAVBg64.exe	Dell	00	6,024 K	HD Audio ...
RAVBg64.exe		00	4,896 K	
RtkNGUI64.exe	Dell	00	4,096 K	Realtek H...
taskhost.exe	Dell	00	3,928 K	Host Proc...
acrotray.exe ...	Dell	00	3,616 K	AcroTray
igfxEM.exe	Dell	00	3,424 K	igfxEM M...
WzPreloader....	Dell	00	3,396 K	WinZip Pr...
Apoint.exe	Dell	00	3,356 K	Alps Point...
igfxTray.exe	Dell	00	3,160 K	igfxTray
taskmgr.exe	Dell	00	2,888 K	Windows ...

Show processes from all users | End Process

Processes: 74 | CPU Usage: 0% | Physical Memory: 42%

All the programs running on my machine. The CPU of course runs one program at a time. Switches between programs periodically.

What are the different storage elements we have come across so far?

What are the different storage elements we have come across so far?

- Register File
- Main memory
- Latches

Regarding all the memory being homogeneous $\longrightarrow$ NOT TRUE

Cell Type	Area	Typical Latency
Master Slave D flip flop	0.8 μm^2	Fraction of a cycle
SRAM cell in an array	0.08 μm^2	1-5 cycles
DRAM cell in an array	0.005 μm^2	50-200 cycles

Typical Values

* More area means more power consumed

Regarding all the memory being homogeneous → NOT TRUE

Cell Type	Area	Typical Latency
Master Slave D flip flop	0.8 μm^2	Fraction of a cycle
SRAM cell in an array	0.08 μm^2	1-5 cycles
DRAM cell in an array	0.005 μm^2	50-200 cycles

Typical Values

* Should we make our memory using only flip-flops ?
  * 10X the area of a memory with SRAM cells
  * 160X the area of a memory with DRAM cells
  * Significantly more power !!!

$CPI = CPI_{ideal}$ + fraction_of_loads * memory_latency

Processor - Memory Gap

Can we do something to overcome this gap and improve the CPI?

Tradeoffs

* Tradeoffs
  * Area, Power, and Latency
  * Increase Area → Reduce latency, increase power
  * Reduce latency → increase area, increase power
  * Reduce power → reduce area, increase latency
* We cannot have

the best of all worlds

What do we do ?

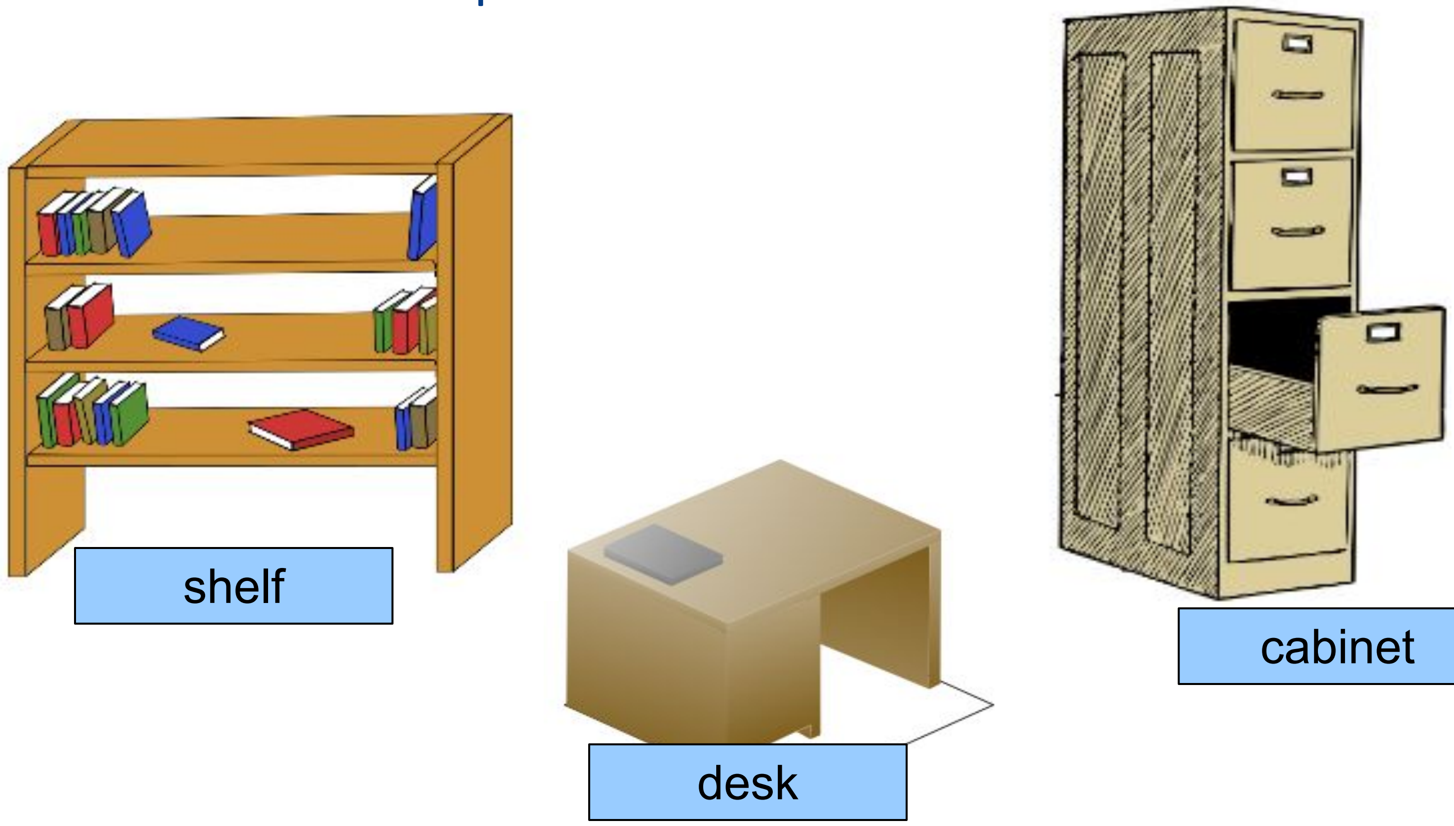
- We cannot create a memory of just flip flops
  - We will hardly be able to store anything
- We cannot create a memory of just SRAM cells
  - We need more storage, and we will not have a 1 cycle latency
- We cannot create a memory of DRAM cells
  - We cannot afford 50+ cycles per access

Memory Access Latency

* What does memory access latency depend on ?
  * Size of the memory → larger is the size, slower it is
  * Number of ports → More are the ports (parallel accesses/cycle), slower is the memory
  * Technology used → SRAM, DRAM, flip-flops

Solution : Leverage Patterns

- Look at an example in real life
  - Sofia's workplace



A Protocol with Books

* Sofia keeps the
  * most frequently accessed books on her desk
  * slightly less frequently accessed books on the shelf
  * rarely accessed books in the cabinet
* Why ?
  * She tends to read the same set of books over and over again, in the same window of time
    → **Temporal Locality**

Protocol – II

- If Sofia takes a computer architecture course
  - She has comp. architecture books on her desk
- After the course is over
  - The architecture books go back to the shelf
  - And, vacation planning books come to the desk
  - **Idea** : Bring all the vacation planning books in one go. If she requires one, in high likelihood she might require similar books in the near future.

Temporal and Spatial Locality

Temporal Locality

It is a concept that states that if a resource is accessed at some point of time, then most likely it will be accessed again in a short period of time.

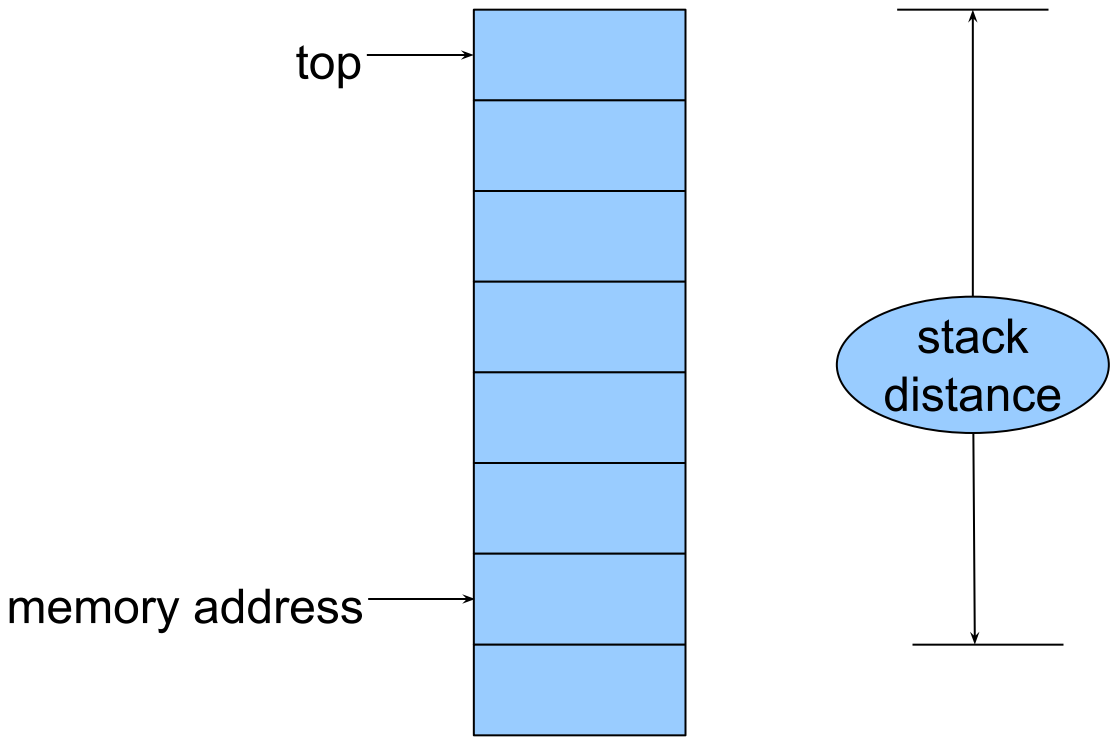
Spatial Locality

It is a concept that states that if a resource is accessed at some point of time, then most likely similar resources will be accessed again in the near future.

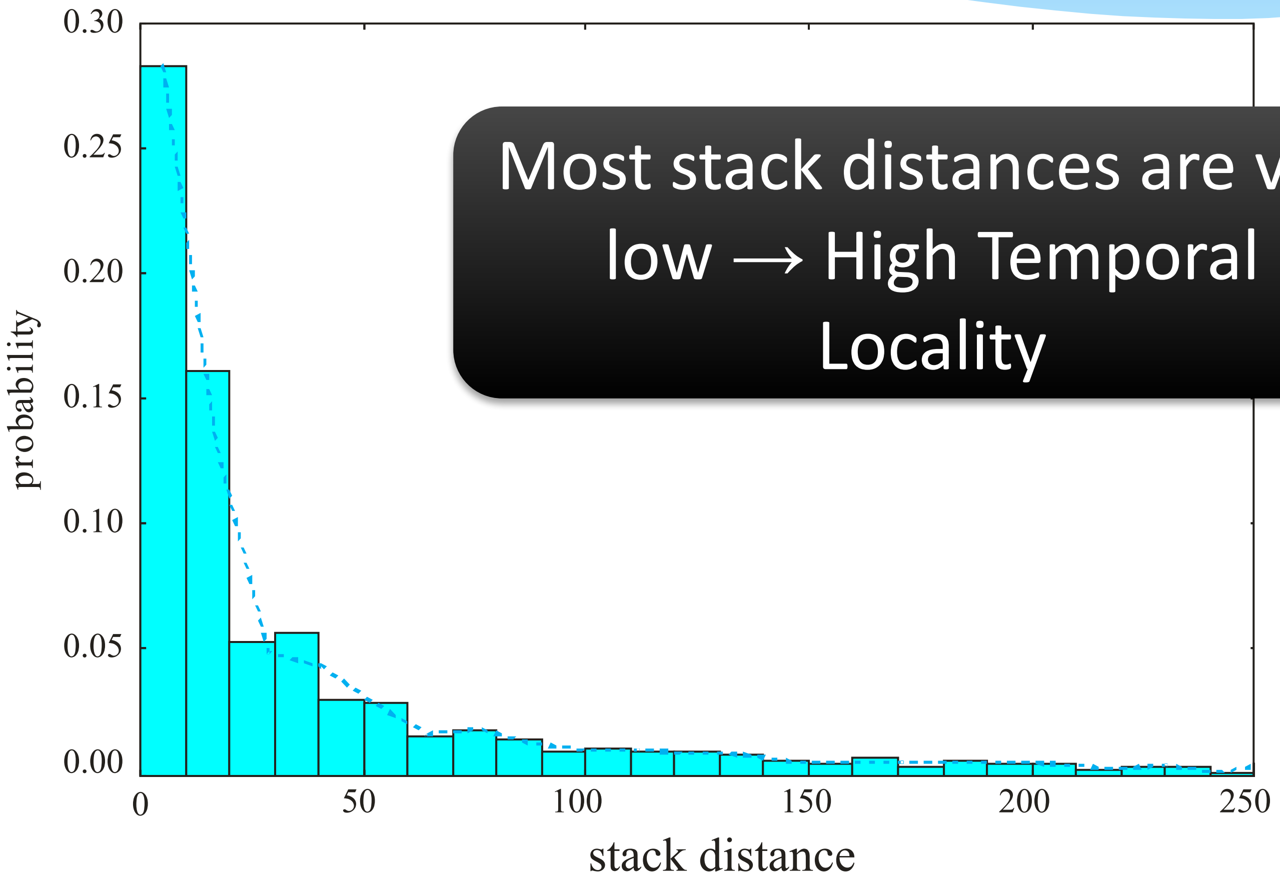
Temporal Locality in Programs

* Let us verify if programs have temporal locality
* Stack distance
  * Have a **stack** to store memory addresses.
  * Whenever, we access an address $\longrightarrow$ we bring it to the top of the stack
  * Stack distance $\longrightarrow$ Distance from the top of the stack to where the element was found
  * Quantifies reuse of addresses

Stack Distance



Stack Distance Distribution



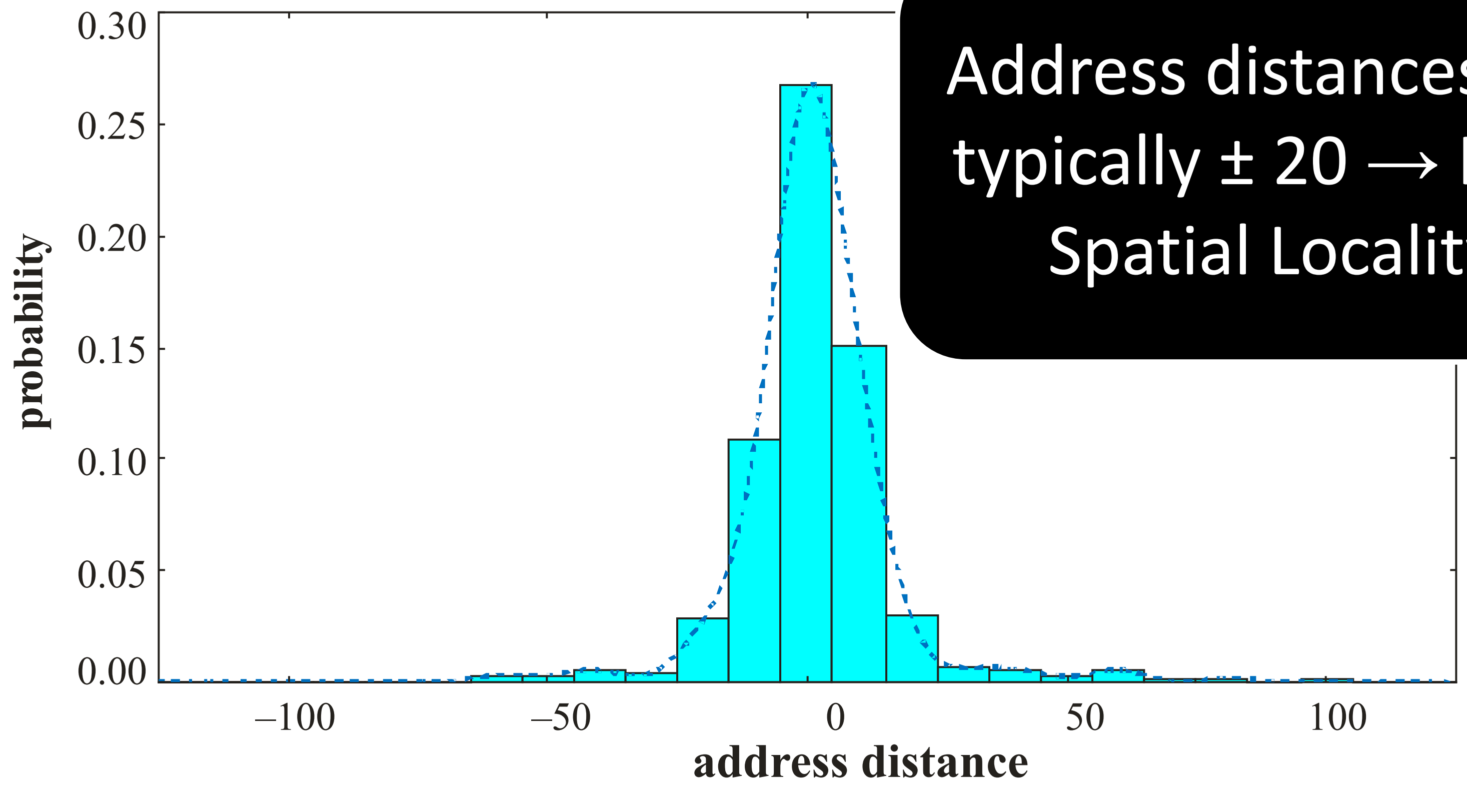
Most stack distances are very low → High Temporal Locality

* Benchmark : Set of perl programs

Address Distance

- Maintain a sliding window of the last *K* memory accesses
- Address distance :
  - The ith address distance is the difference in the memory addresses of the ith memory access, and the closest address in the set of last K memory accesses.
  - Shows the **similarity** in addresses

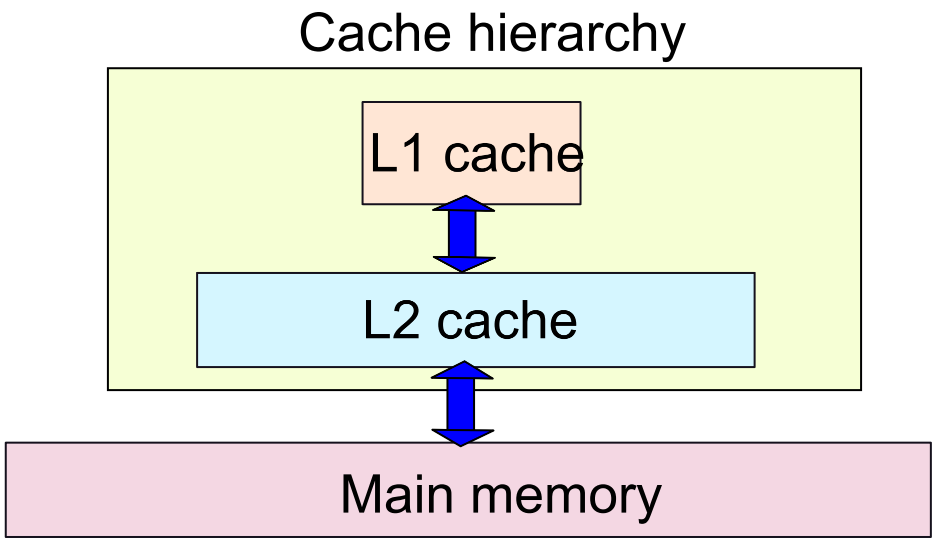
Address Distance Distribution



Address distances are typically ± 20 ⟶ High Spatial Locality

K=10, benchmark consisting of *perl* programs

Exploiting Temporal Locality



* Use a hierarchical memory system
* L1 (SRAM cells), L2 (SRAM cells), Main Memory (DRAM cells)

The Caches

* The L1 cache is a small memory (8-64 KB) composed of SRAM cells
* The L2 cache is larger and slower (128 KB – 4 MB) (SRAM cells)
* The main memory is even larger (1 – 64 GB) (DRAM cells)
* Cache hierarchy
  * The main memory contains all the memory locations
  * The caches contain a subset of memory locations

Access Protocol

* Inclusive Cache Hierarcy
  * addresses(L1) ⊏ addresses(L2) ⊏ addresses(main memory)
* Protocol
  * First access the L1 cache. If the memory location is present, we have a **cache hit**.
    * Perform the access (read/write)
  * Otherwise, we have a **cache miss**.
    * Fetch the value from the lower levels of the memory system, and populate the cache.
    * Follow this protocol recursively

Advantage

- Typical Hit Rates, Latencies
  - L1 : 95 %, 1 cycle
  - L2 : 60 %, 10 cycles
  - Main Memory : 100 %, 300 cycles
- Result :
  - 95 % of the memory accesses take a single cycle
  - 3 % take, 10 cycles
  - 2 % take, 300 cycles

Exploiting Spatial Locality

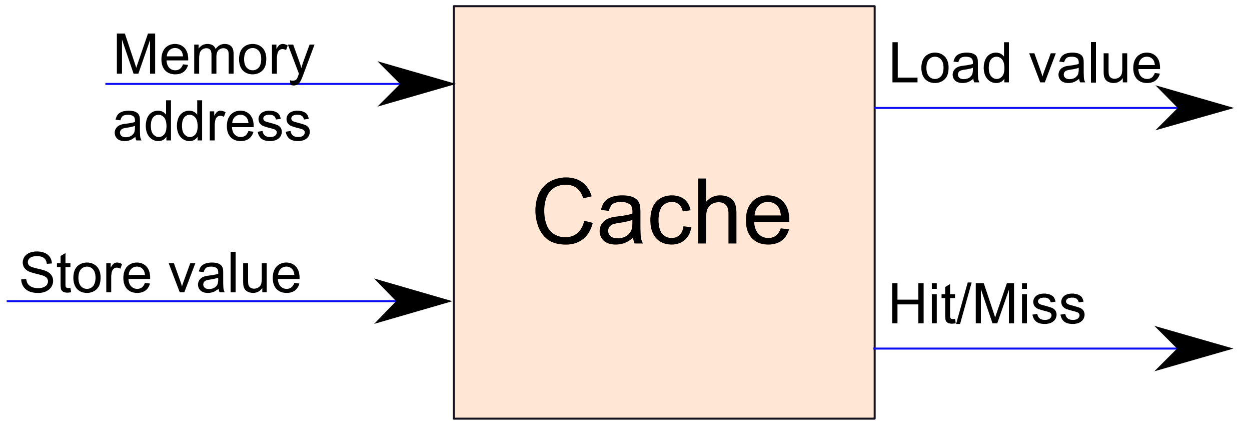
* Conclusion from the address locality plot
  * Most of the addresses are within +/- 25 bytes
* Idea :
  * Group memory addresses into sets of *n* bytes
  * Each group is known as a cache line or cache block
  * A cache block is typically 32, 64, or 128 bytes

Reason: Once we fetch a block of 32/64 bytes. A lot of accesses in a short time interval will find their data in the block.

Outline

* Overview of the Memory System
* Caches
* Details of the Memory System
* Virtual Memory

Overview of a Basic Cache



* Saves a subset of memory values
    * We can either have hit or miss
    * The load/store is successful if we have a hit

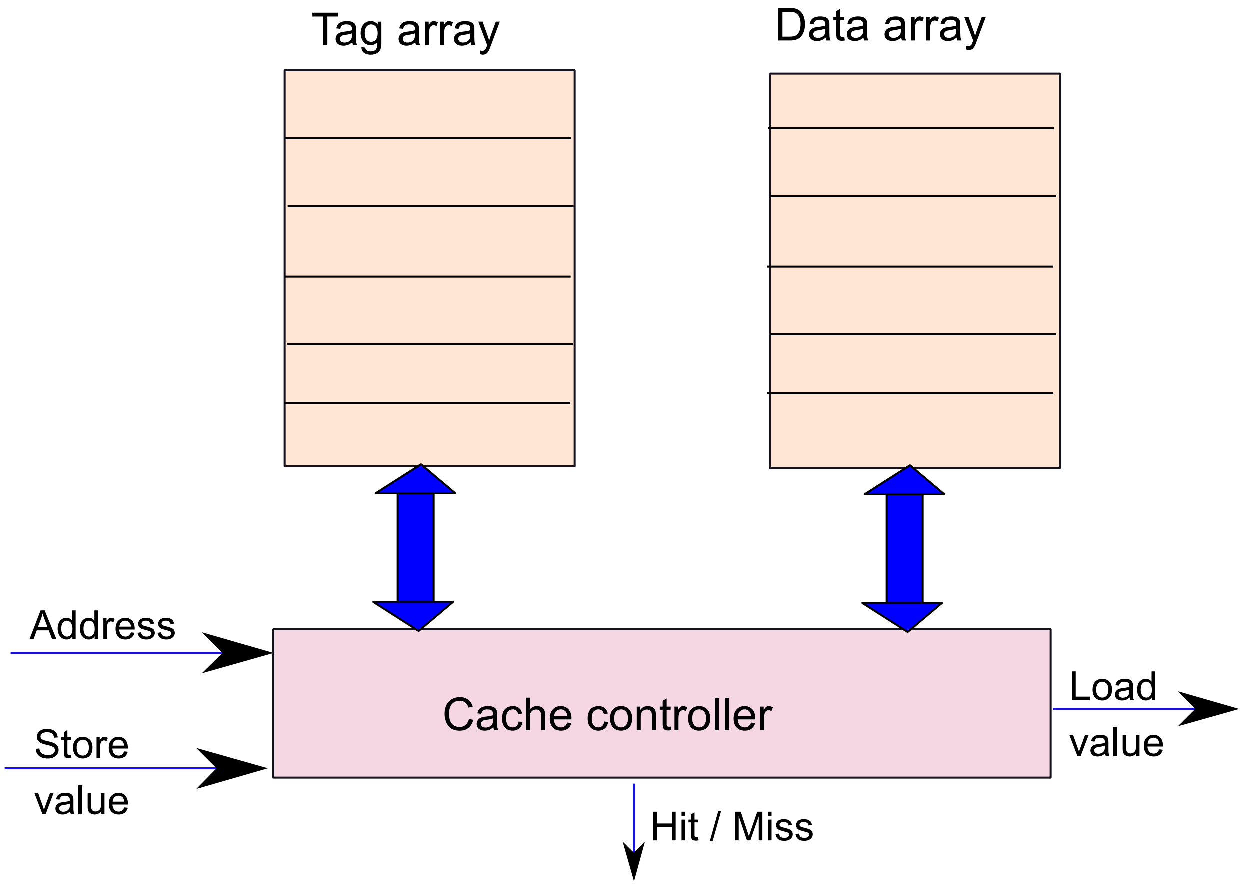
Basic Cache Operations

* lookup $\longrightarrow$ Check if the memory location is present
* data read $\longrightarrow$ read data from the cache
* data write $\longrightarrow$ write data to the cache
* insert $\longrightarrow$ insert a block into a cache
* replace $\longrightarrow$ find a candidate for replacement
* evict $\longrightarrow$ throw a block out of the cache

Cache Lookup

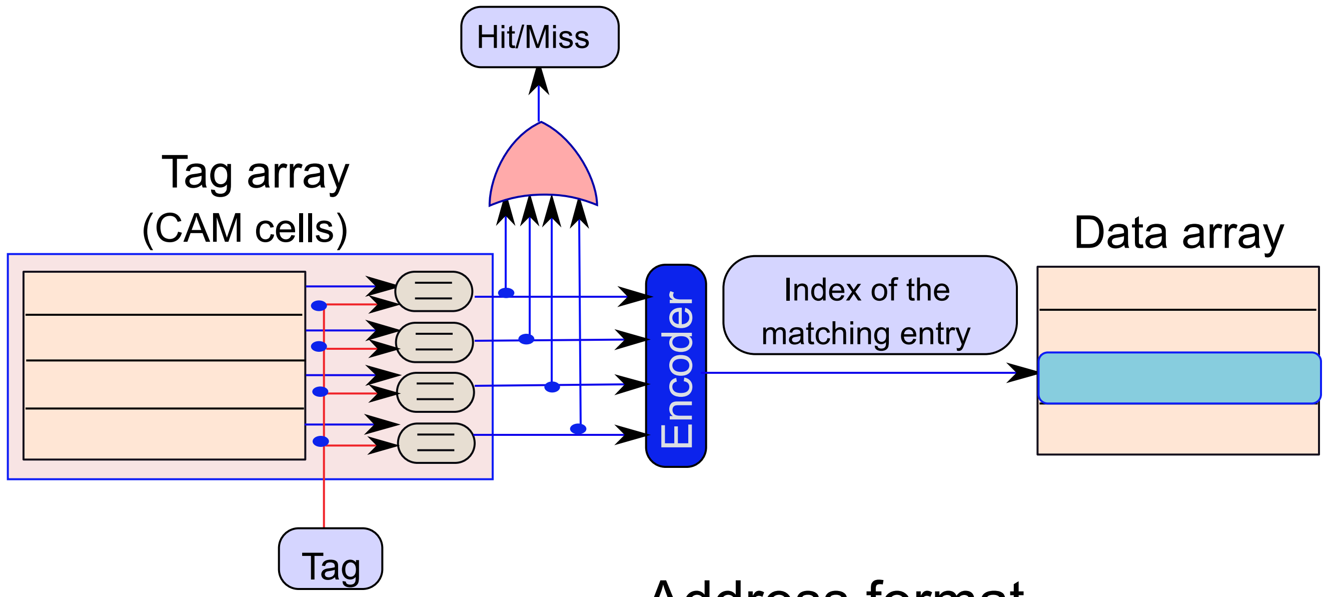
* Running example : 8 KB Cache, block size of 64 bytes, 32 bit memory system
* Let us have two SRAM arrays
    * tag array → Saves a part of the block address such that the block can be uniquely identified
    * block array → Saves the contents of the block
    * Both the arrays have the same number of entries

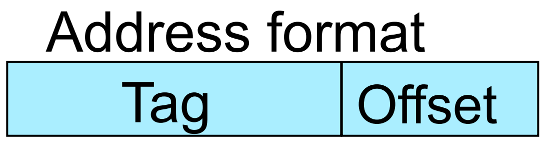
Structure of a Cache



Fully Associative Cache

* We have $2^{13} / 2^{6} = 128$ entries
* A block can be saved in any entry
* 26 bit tag, and 6 bit offset



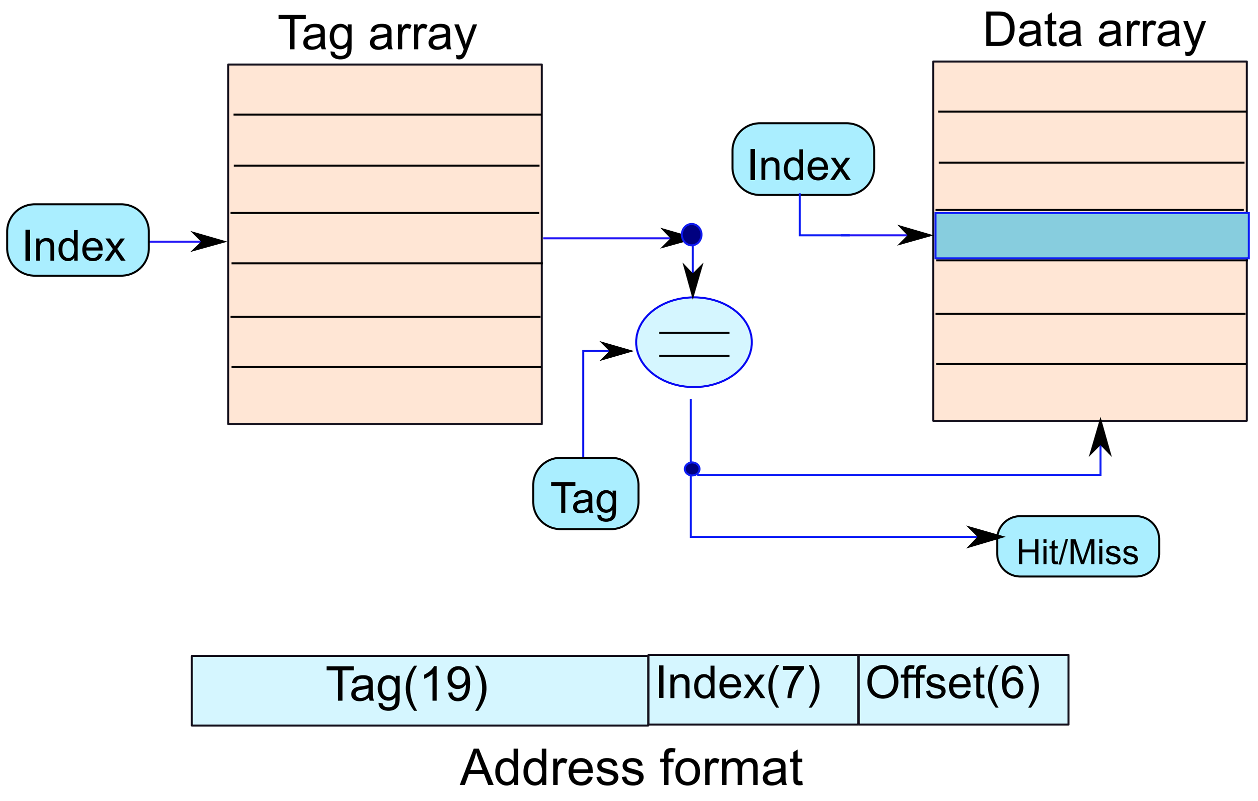


Implementation of the FA Cache

* We use an array of CAM cells for the tag array
* Each entry compares its contents with the tag
* Sets the match line to 1
* The OR gate computes a hit or miss
* The encoder computes the index of the matching entry.
* We then read the contents of the matching entry from the block array

Refer to Chapter 6: Digital Logic

Direct Mapped Cache

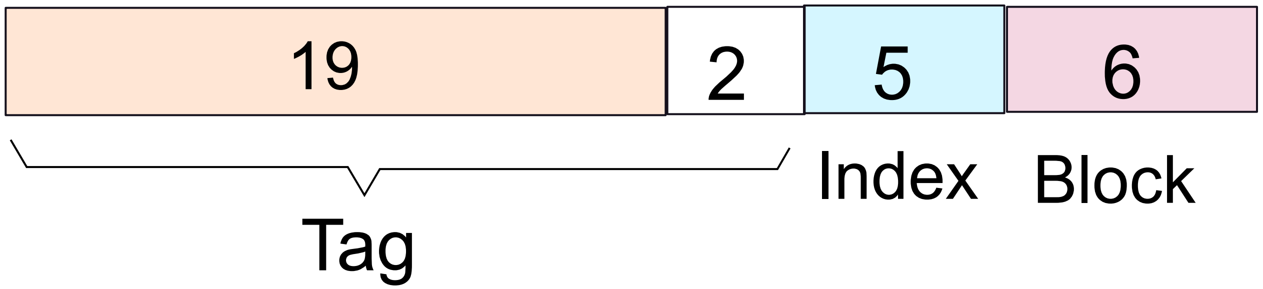


* Each block can be mapped to only 1 entry

Direct Mapped Cache

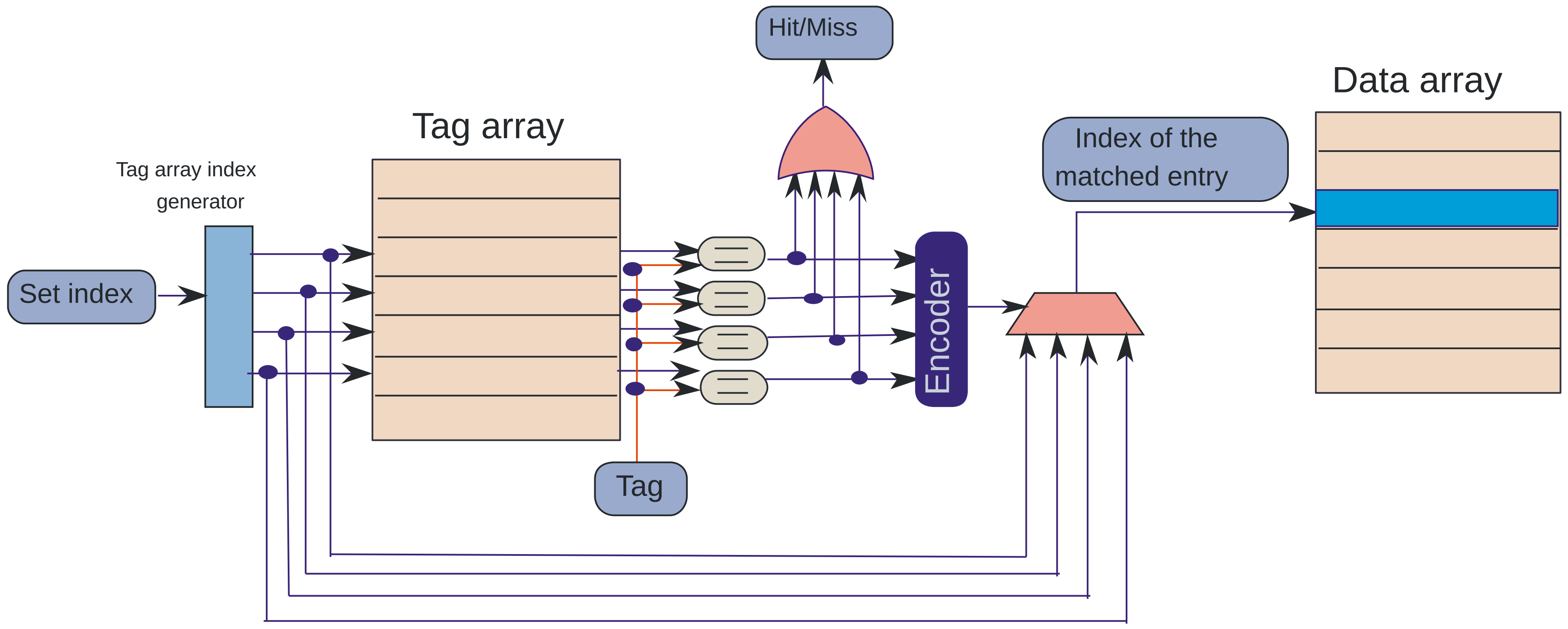
* We have 128 entries in our cache.
* We compute the index as idx = *block address* % 128
* We access entry, idx, in the tag array and compare the contents of the tag (19 msb bits of the address)
* If there is a match → **hit**
  * else → **miss**
* Need a solution that is in the middle of the spectrum

Set Associative Cache



- Let us assume that an address can reside in 4 locations
  - Access all 4 locations, and see if there is a hit
- Thus, we have 128/4 = 32 indices
- Each index points to a *set* of 4 entries
- → We now use a 21 bit tag, 5 bit index

Set Associative Cache



Set Associative Cache

- Let the index be *i* , and the number of elements in a set be *k*
  - We access indices, i*k, i*k+1 ,.., i*k + (k-1)
  - Read all the tags in the set
  - Compare the tags with the tag obtained from the address
  - Use an OR gate to compute a hit/ miss
  - Use an encoder to find the index of the matched entry

Set Associative Cache – II

* Read the corresponding entry from the block array
* Each entry in a set is known as a way
* A cache with k blocks in a set is known as a k-way associative cache

Data read operation

- This is a regular SRAM access.
- Note that the data read and lookup can be overlapped for a load access
  - We can issue a parallel data read to all the ways in the cache
  - Once, we compute the index of the matching tag, we can choose the correct result with a multiplexer.

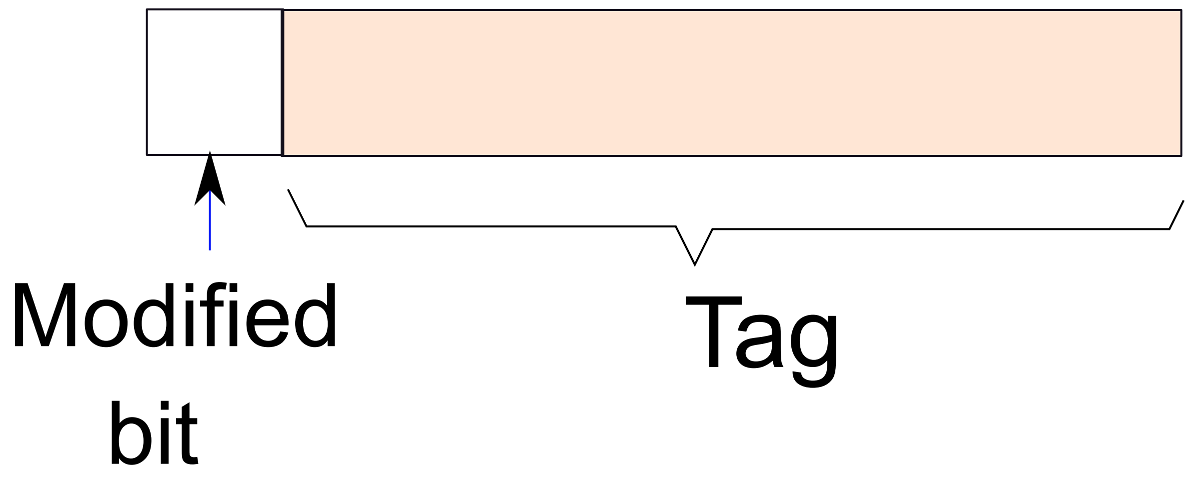
Data write operation

* Before we write a value
    * We need to ensure that the block is present in the cache
* Why ?
    * Otherwise, we have to maintain the indices of the bytes that were written to
    * We treat a block as an atomic unit
    * Hence, on a miss, we fetch the entire block first
* Once a block is there in the cache
    * Go ahead and write to it ....

Mc Graw Hill Education

Modified bit

- Maintain a modified bit in the tag array.
- If a block has been written to, after it was fetched, set it to 1.



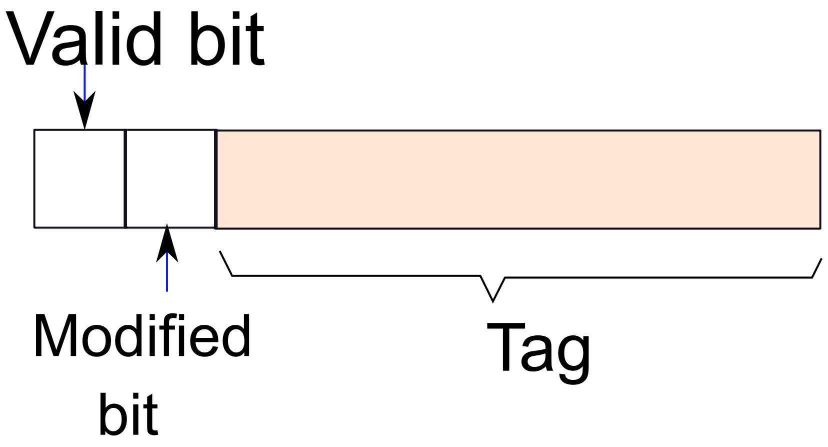
Write Policies

- Write through $\longrightarrow$ Whenever we write to a cache, we also write to its lower level
  - Advantage : Can seamlessly evict data from the cache
- Write back $\longrightarrow$ We do not write to the lower level. Whenever we write, we set the modified bit.
  - At the time of eviction of the line, we check the value of the modified bit

insert operation

If we don't find a block in a cache. We fetch it from the lower level. Then we insert the block in the cache ⟶ insert operation

* Let us add a valid bit to a tag
  * If the line is non-empty, valid bit is 1
  * Else it is 0
* Structure of a tag



insert operation - II

* Check if any way in a set has an invalid line
    * If there is one, then write the fetched line to that location, set the valid bit to 1.
* Otherwise,
    * find a candidate for replacement

McGraw Hill Education

The replace operation

- A cache replacement scheme or replacement policy is a method to replace an entry in the set by a new entry
- Replacement Schemes
  - Random replacement scheme
  - FIFO replacement scheme
    - When we fetch a block, assign it a counter value equal to 0
    - Increment the counters of the rest of the ways

Replacement Schemes

* FIFO
    * For replacement, choose the way with the highest counter (oldest).
* Problems :
    * Can violate the principle of temporal locality
    * A line fetched early might be accessed very frequently.

LRU (least recently used)

* Replace the block that has been accessed the least in the recent past
  * Most likely we will not access it in the near future
    * Directly follows from the definition of stack distance
    * Sadly, we need to do more **work** per access
    * Proved to be **optimal** in some restrictive scenarios
* True LRU requires saving a hefty timestamp with every way
    * Let us implement pseudo-LRU

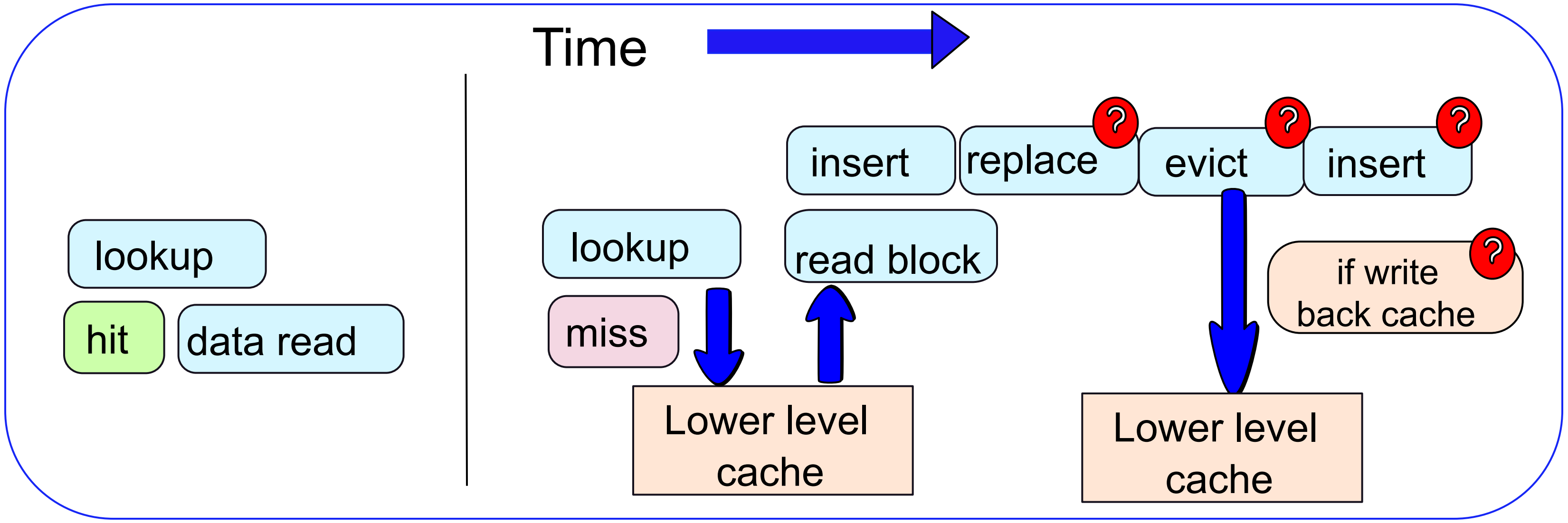
Psuedo-LRU

* Let us try to mark the most recently used (MRU) elements.
* Let us associate a 3 bit counter with every way.
  * Whenever we access a line, we increment the counter.
  * We stop incrementing beyond 7.
  * We periodically decrement all the counters in a set by 1.
  * Set the counter to 7 for a newly fetched block
  * For replacement, choose the block with the smallest counter.

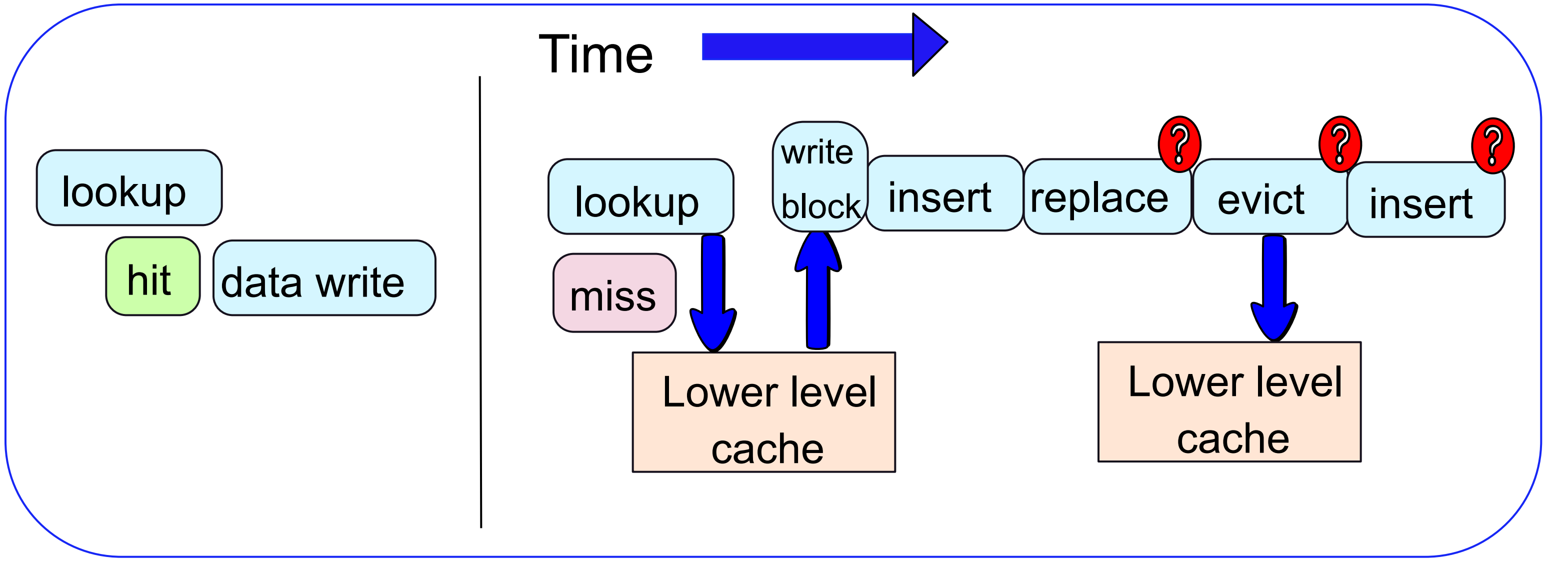
evict Operation

- If the cache is write-through
  - Nothing needs to be done
- If the cache is write-back
  - AND the modified bit is 1
  - Write the line to the lower level

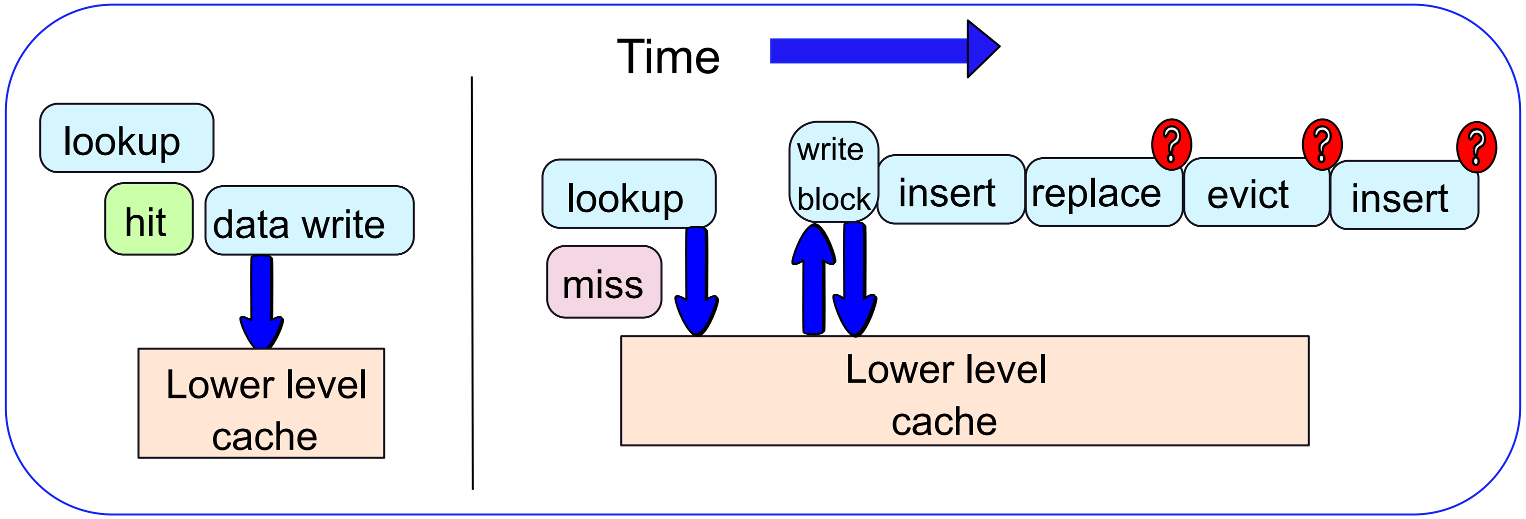
The read(load) Operation



Write operation in a write back cache



Write operation in a write through cache



Outline

* Overview of the Memory System
* Caches
* Details of the Memory System
* Virtual Memory

Mathematical Model of the Memory System

$$\begin{aligned} CPI &= CPI_{ideal} + stall_{rate} * stall_{cycles} \\ &= CPI_{ideal} + f_{mem} * (AMAT - 1) \end{aligned}$$

- AMAT ⟶ Average Memory Access Time
- f_{mem} ⟶ Fraction of memory instructions
- **CPI_{ideal}** ⟶ ideal CPI assuming a perfect 1 cycle memory system

Mc Graw Hill Education

Equation for AMAT

$$AMAT = L1_{hit\ time} + L1_{miss\ rate} * L1_{miss\ penalty}$$

$$= L1_{hit\ time} + L1_{miss\ rate} * (L2_{hit\ time} + L2_{miss\ rate} * L2_{miss\ penalty})$$

- Irrespective of an hit or a miss, we need to spend some time (hit time)
- This is the hit time in the L1 cache ($L1_{\text{hit time}}$)
- This time should be discarded while calculating the stall penalty due to L1 misses
- **stall penalty** = AMAT - $L1_{\text{hit time}}$

Mc Graw Hill Education

n-Level Memory System

$$AMAT = L1_{hit\ time} + L1_{miss\ rate} * L1_{miss\ penalty}$$

$$L1_{miss\ penalty} = L2_{hit\ time} + L2_{miss\ rate} * L2_{miss\ penalty}$$

$$L2_{miss\ penalty} = L3_{hit\ time} + L3_{miss\ rate} * L3_{miss\ penalty}$$

$$\ldots. = \ldots.$$

$$L(n-1)_{miss\ penalty} = Ln_{hit\ time}$$

Definition: Local and Global Miss Rates, Working Set

local miss rate	It is equal to the number of misses in a cache at level *i* divided by the total number of accesses at level *i*.
global miss rate	It is equal to the number of misses in a cache at level *i* divided by the total number of memory accesses.
working set	The amount of memory, a given program requires in a time interval.

Types of Misses

* Compulsory Misses
    * Misses that happen when we read in a piece of data for the first time.
* Conflict Misses
    * Misses that occur due to the limited amount of associativity in a set associative or direct mapped cache. Example: Assume that 5 blocks (accessed by the program) map to the same set in a 4-way associative cache. Only 4 out of 5 can be accommodated.
* Capacity Misses
    * Misses that occur due to the limited size of a cache. Example: Assume the working set of a program is 10 KB, and the cache size is 8 KB.

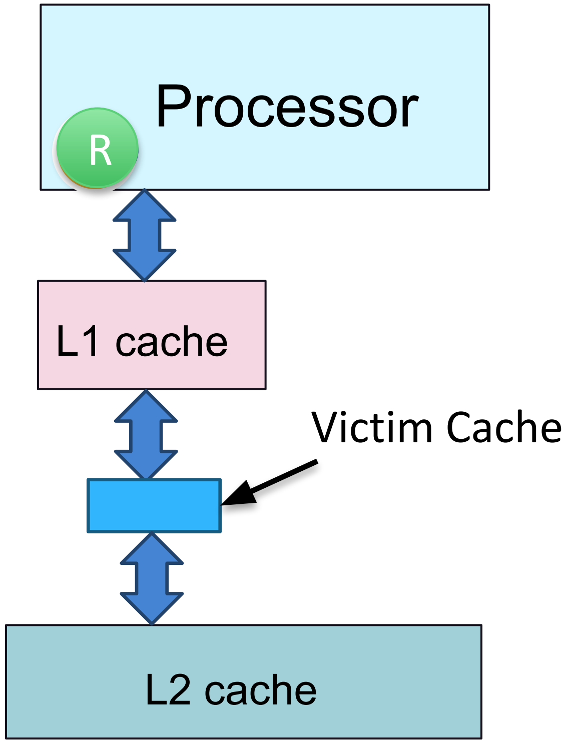
Schemes to Mitigate Misses

* Compulsory Misses
  * Increase the block size. We can bring in more data in one go, and due to **spatial locality** the number of misses might go down.
  * Try to guess the memory locations that will be accessed in the near future. **Prefetch** (fetch in advance) those locations. We can do this for example in the case of array accesses.

Schemes to Mitigate Misses - II

* Conflict Misses
  * Increase the associativity of the cache (at the cost of **latency** and **power**)
  * We can use a smaller fully associative cache called the ***victim cache*** . Any line that gets displaced from the main cache can be put in the victim cache. The processor needs to check both the L1 and victim cache, before proceeding to the L2 cache.
  * Write programs in a **cache friendly** way.

Victim Cache



Schemes to Mitigate Misses - III

- Capacity Misses
  - Increase the size of the cache
  - Use better prefetching techniques.

Some Thumb Rules

$$miss\ rate \propto \frac{1}{\sqrt{cache\ size}} \qquad [Square\ Root\ Rule]$$

* **Associativity Rule** $\longrightarrow$ Doubling the associativity is almost the same as doubling the cache size with the original associativity
  * 64 KB, 4 way $\longleftarrow\!\longrightarrow$ 128 KB, 2 way

Software Prefetching

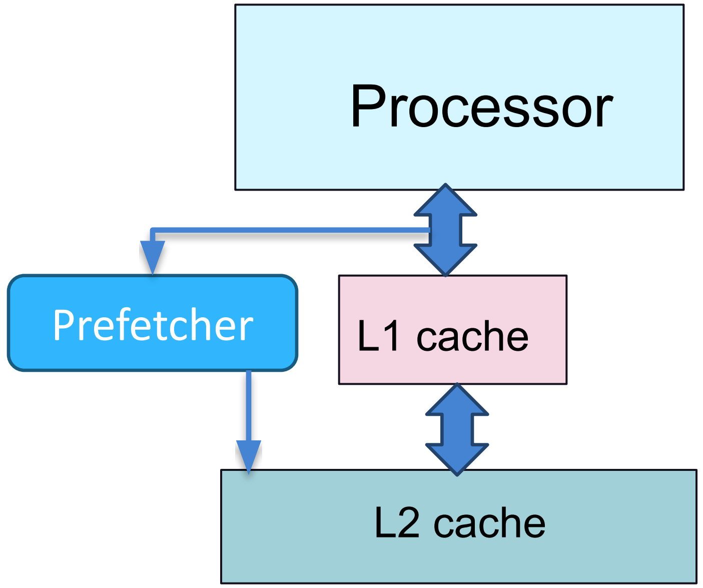
Original Code

```
int addAll(int data[], int vals[]) {
        int i, sum = 0;
        for (i=0; i < N; i++)
                sum += data[vals[i]];
        return sum;
}
```

Modified Code with Prefetching

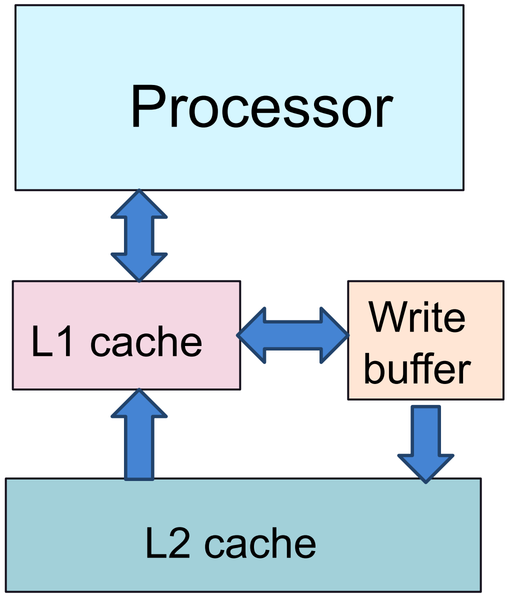
```
int addAllP(int data[], int vals[]) {
        int i, sum = 0;
        for (i=0; i < N; i++) {
                __builtin_prefetch(& data[vals[i+100]]  );
                sum += data[vals[i]];
        }
        return sum;
}
```

Hardware Prefetching



Reduction of Hit Time and Miss Penalty

- For reducing the hit time, we need to use smaller and simpler caches
- For reducing the miss penalty :
  - **Write misses**
    - Send the writes to a **fully associative** write buffer on an L1 miss.
    - Once the block comes from the **L2 cache**, merge the write
    - **Insight**: We need not send separate writes to the L2 for each write request in a block.



Reduction of the Miss Penalty

- Read Miss
  - **Critical Word First :** The memory word that cause the read/write miss is fetched first from the lower level. The rest of the block follows.
  - **Early Restart** : Send the **critical word** to the processor, and make it restart its execution.

Technique	Application	Disadvantages
large block size	compulsory misses	reduces the number of blocks in the cache
prefetching	compulsory misses, capacity misses	extra complexity and the risk of displacing useful data from the cache
large cache size	capacity misses	high latency, high power, more area
increased associativity	conflict misses	high latency, high power
victim cache	conflict misses	extra complexity
compiler based techniques	all types of misses	not very generic
small and simple cache	hit time	high miss rate
write buffer	miss penalty	extra complexity
critical word first	miss penalty	extra complexity and state
early restart	miss penalty	extra complexity

Outline

- Overview of the Memory System
- Caches
- Details of the Memory System
- Virtual Memory

Need for Virtual Memory

* Up till now we have assumed that a program perceives the entire memory system to be its own

* Furthermore, every program on a 32 bit machine assumes that it owns 4 GB of memory space, and it can access any location at will

* We now need to take multiple **programs** into account. The CPU runs program *A* for some time, then switches to program *B*, and then to program *C*. Do they corrupt each other's data?

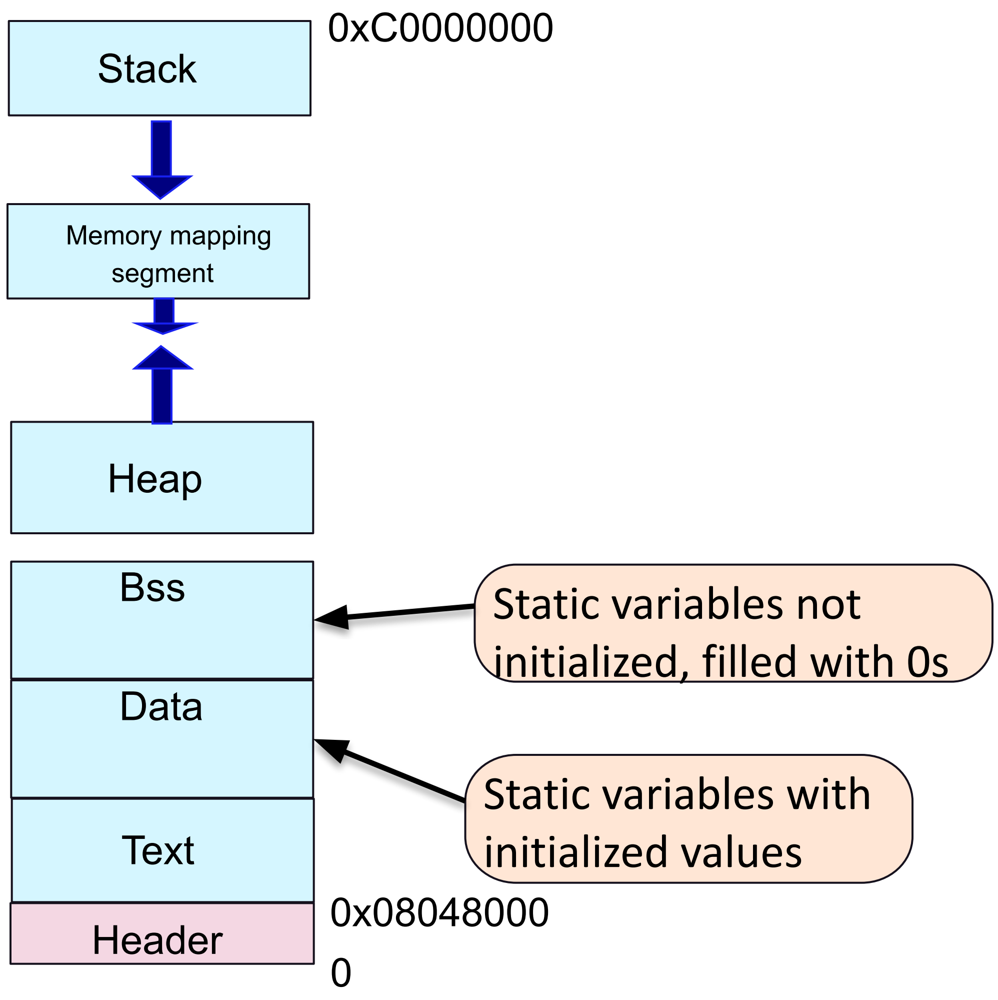
* Secondly, we need to design memory systems that have less than 4 GB of memory (for a 32 bit memory address)

Mc Graw Hill Education

Let us thus **define** two concepts …

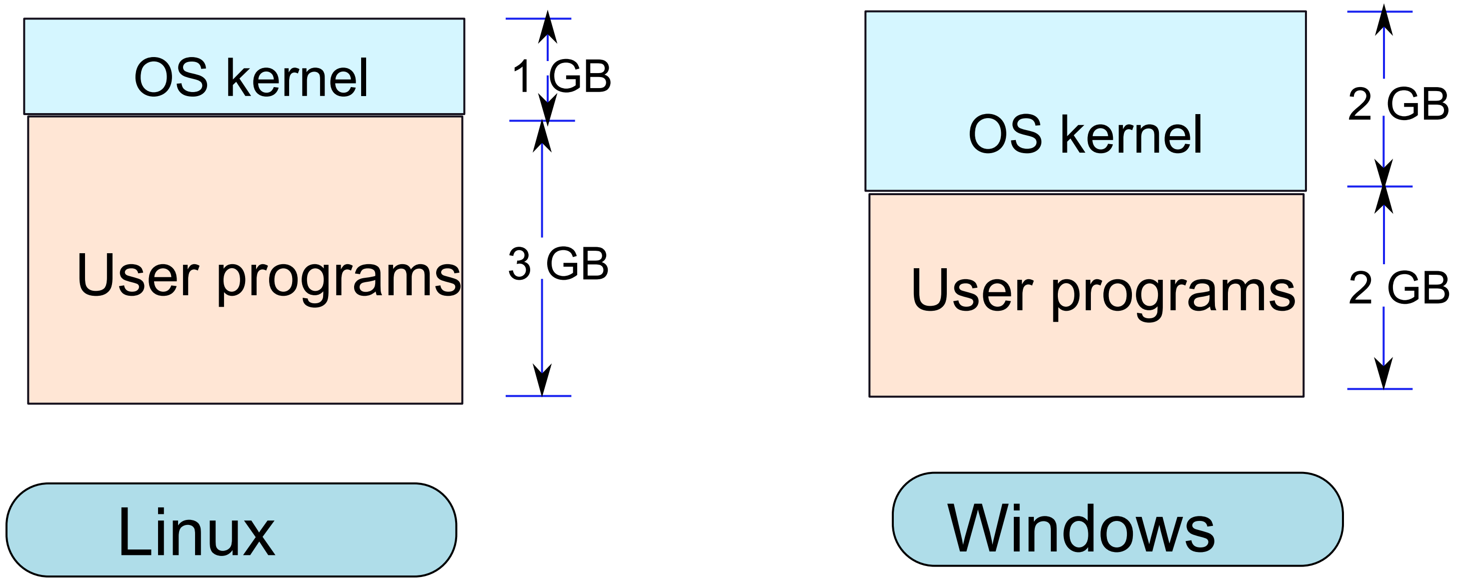
- **Physical Memory**
  - Refers to the actual set of physical memory locations contained in the main memory, and the caches.
- **Virtual Memory**
  - The memory space **assumed** by a program.
  - Contiguous, without limits.

Virtual Memory Map of a Process (in Linux)

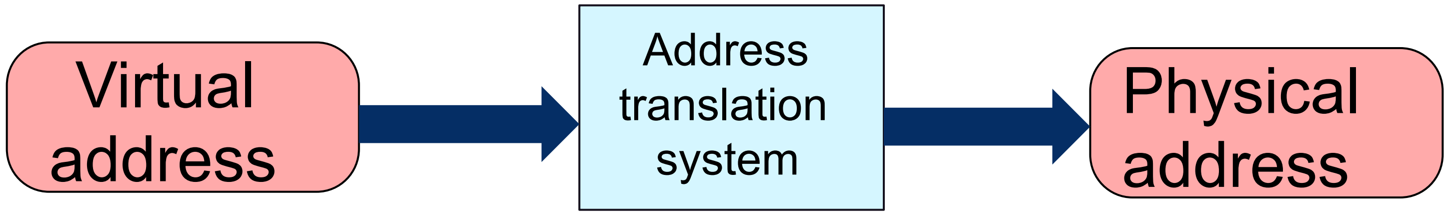


0xC0000000
Stack
Memory mapping segment
Heap
Bss
Data
Text
Header
0x08048000
0
Static variables not initialized, filled with 0s
Static variables with initialized values

Memory Maps Across Operating Systems



Address Translation

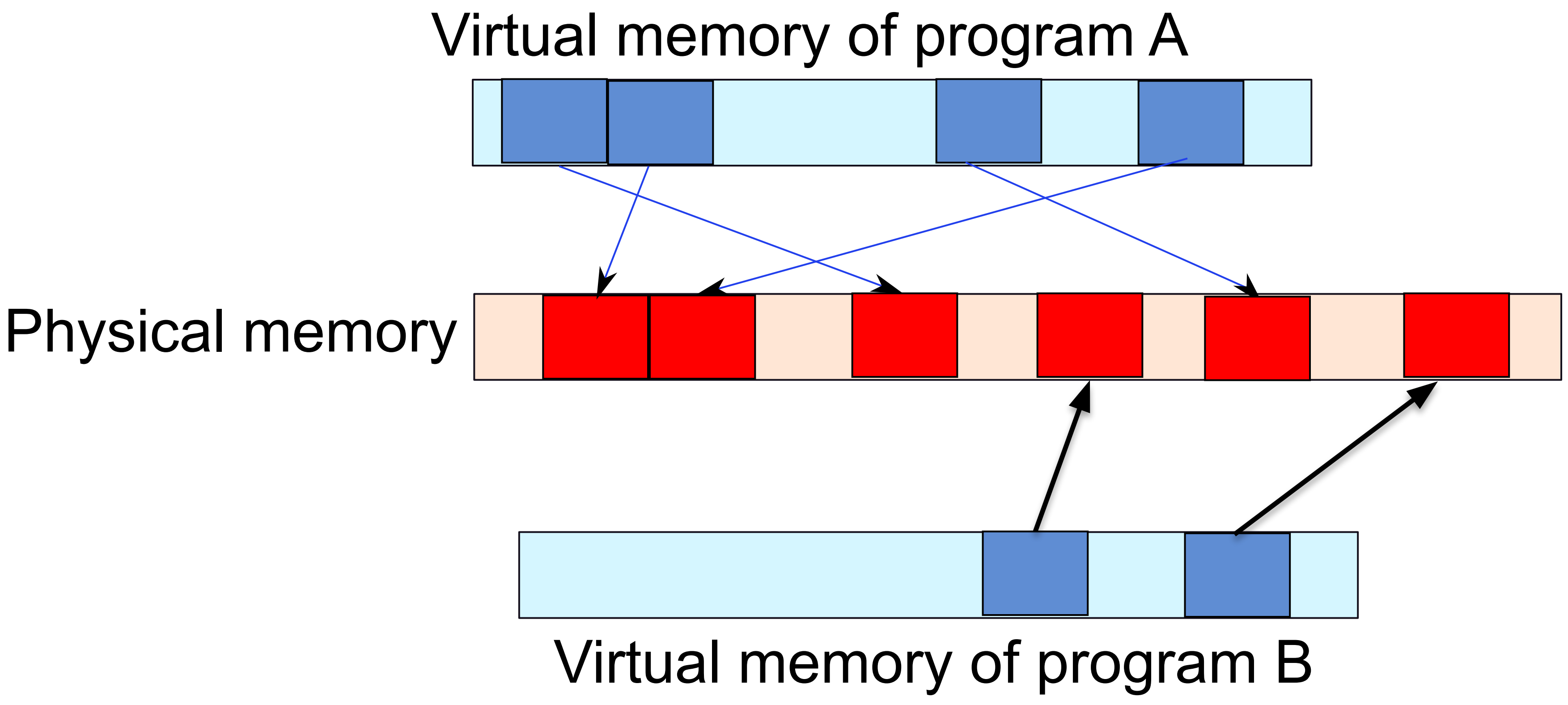


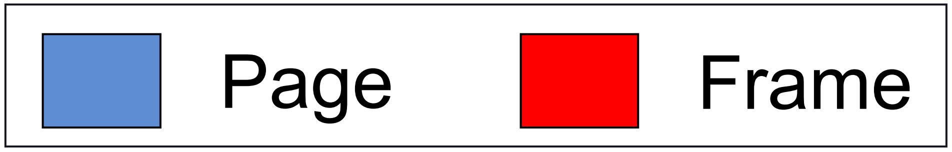
- Convert a virtual address to a physical address to satisfy all the aims of the virtual memory system

Pages and Frames

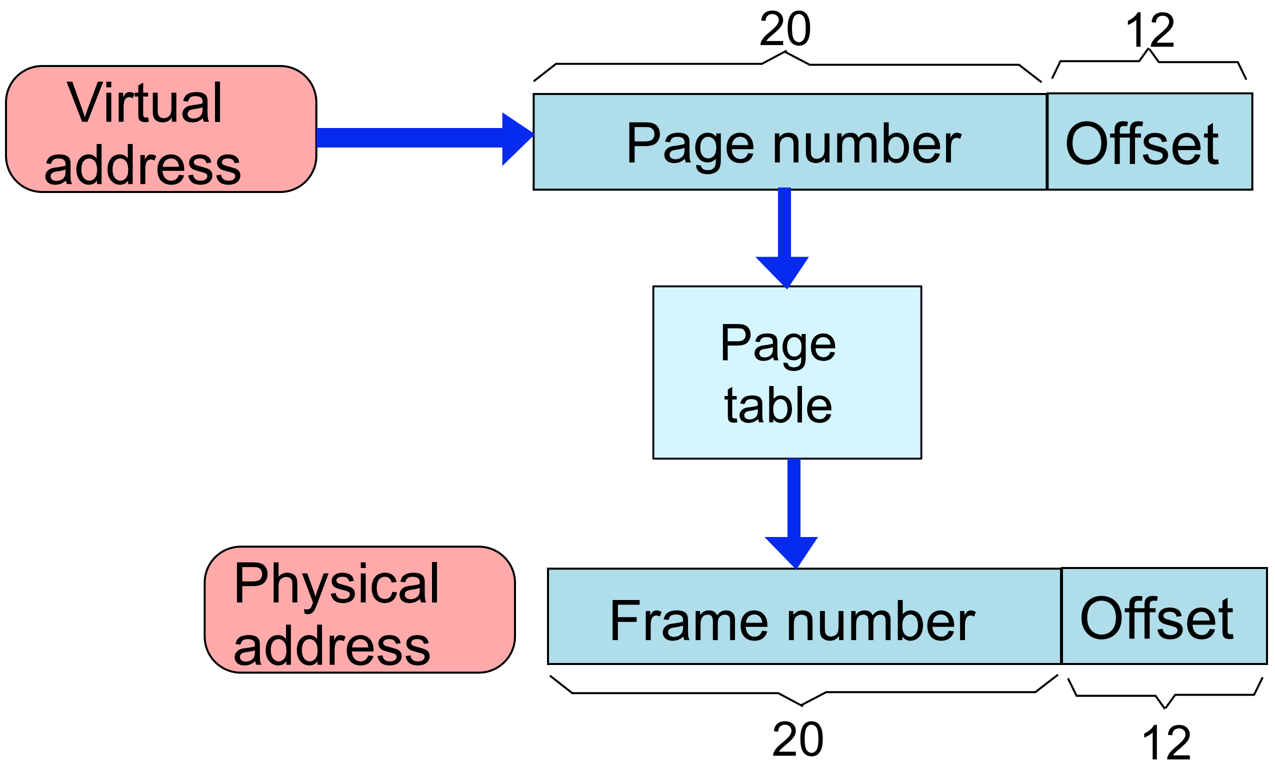
* Divide the virtual address space into chunks of 4 kB ⟶ page
* Divide the physical address space into chunks of 4 kB ⟶ frame
* **Map** pages to frames
    * Insight: If a page/frame size is large, most of it may remain unused
    * If the page/frame size is very small, the overhead of mapping will be very high

Map Pages to Frames

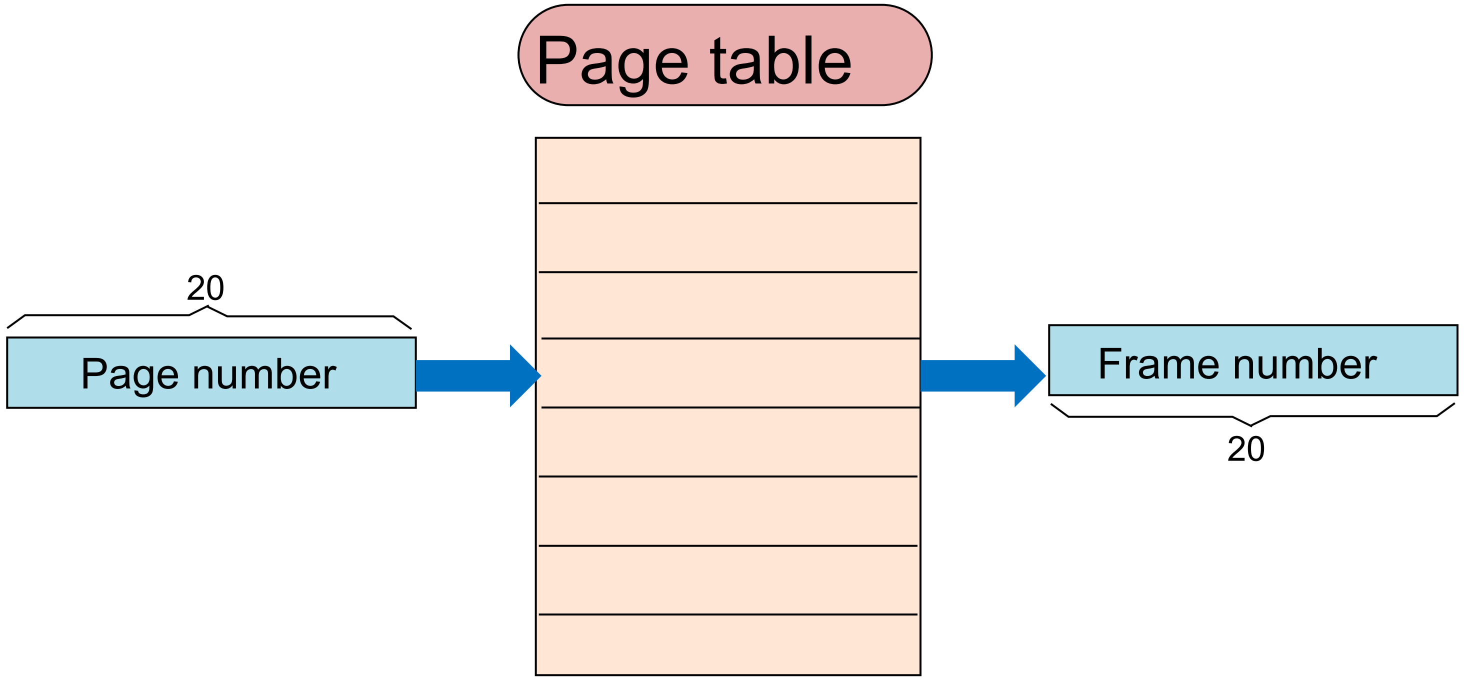




Example of Page Translation



Single Level Page Table



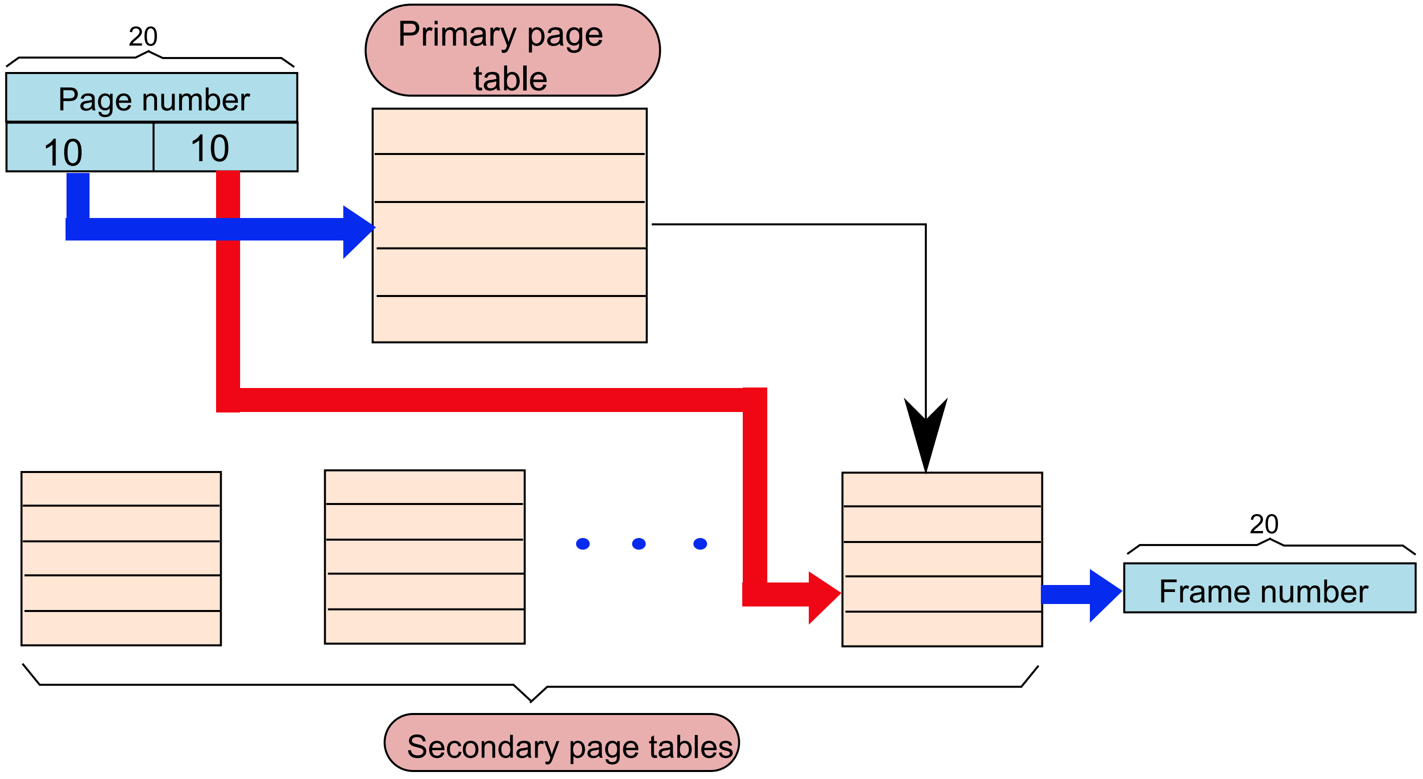
Issues with the Single Level Page Table

- Size of the **single level page table**
  - Size of an entry (20 bits = 2.5 bytes) *
  - Number of entries (2^{20} = 1 million)
  - Total → 2.5 MB
- For 200 processes (running instances of programs)
  - We spend 500 MB in saving page tables (not acceptable)

Insight : Most of the **virtual address space** is **empty**

- Most programs do not require that much of memory
- They require maybe 100 MBs or 200 MBs (most of the time)

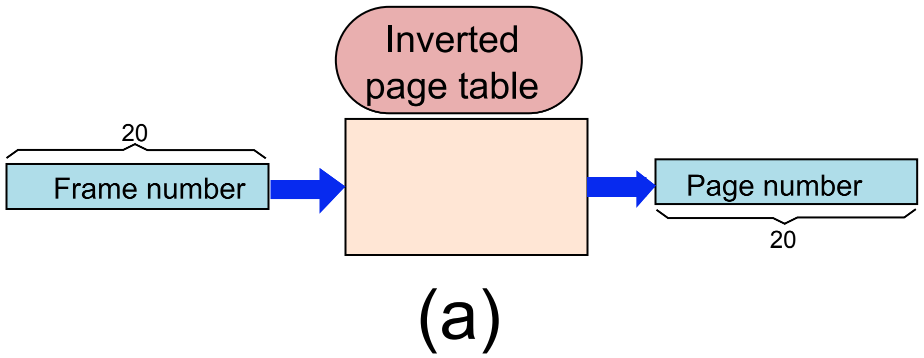
Two Level Page Table

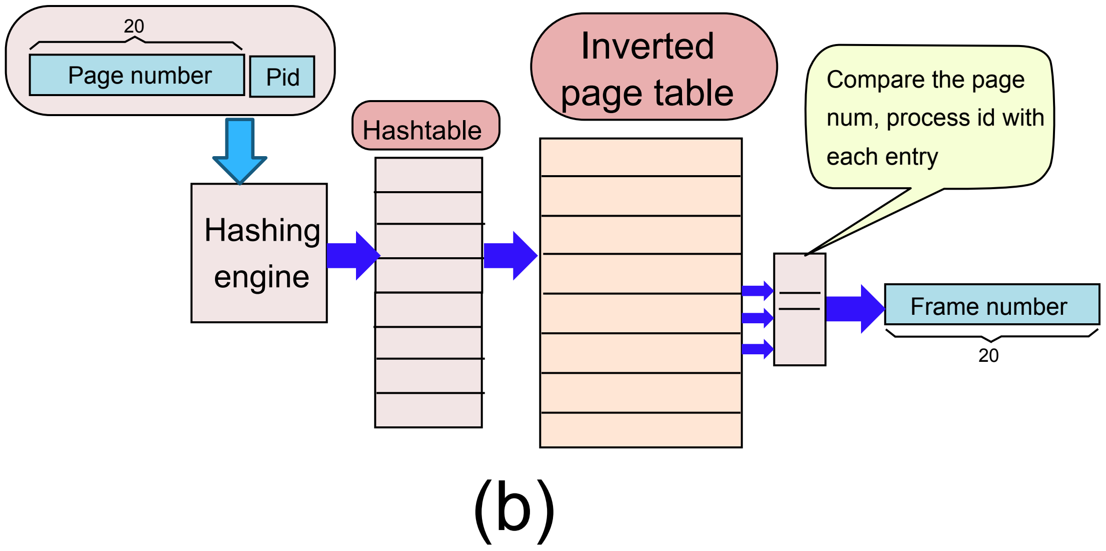


Two Level Page Tables - II

- We have a two level set of page tables
  - Primary and secondary page tables
- Not all the entries of the primary page table point to valid secondary page tables
- Each secondary page table → 1024 * 2.5 B = 2.5 KB
  - Maps 4MB of virtual memory
- **Insight**: Allocate only those many secondary page tables as required.
  - We do not need many secondary page tables due to **spatial locality** in programs
  - **Example**: If a program uses 100 MB of virtual memory and needs 25 secondary page tables, we need a total of 2.5KB * 25 = 62.5 KB of space for saving secondary page tables (minimal).

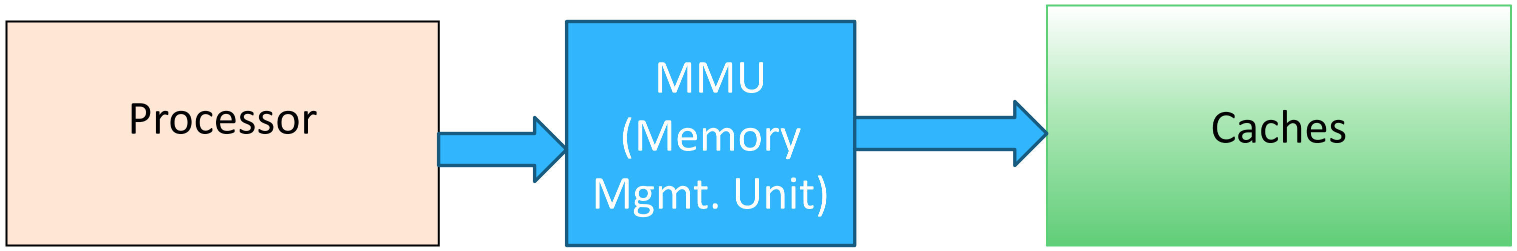
Inverted Page Table





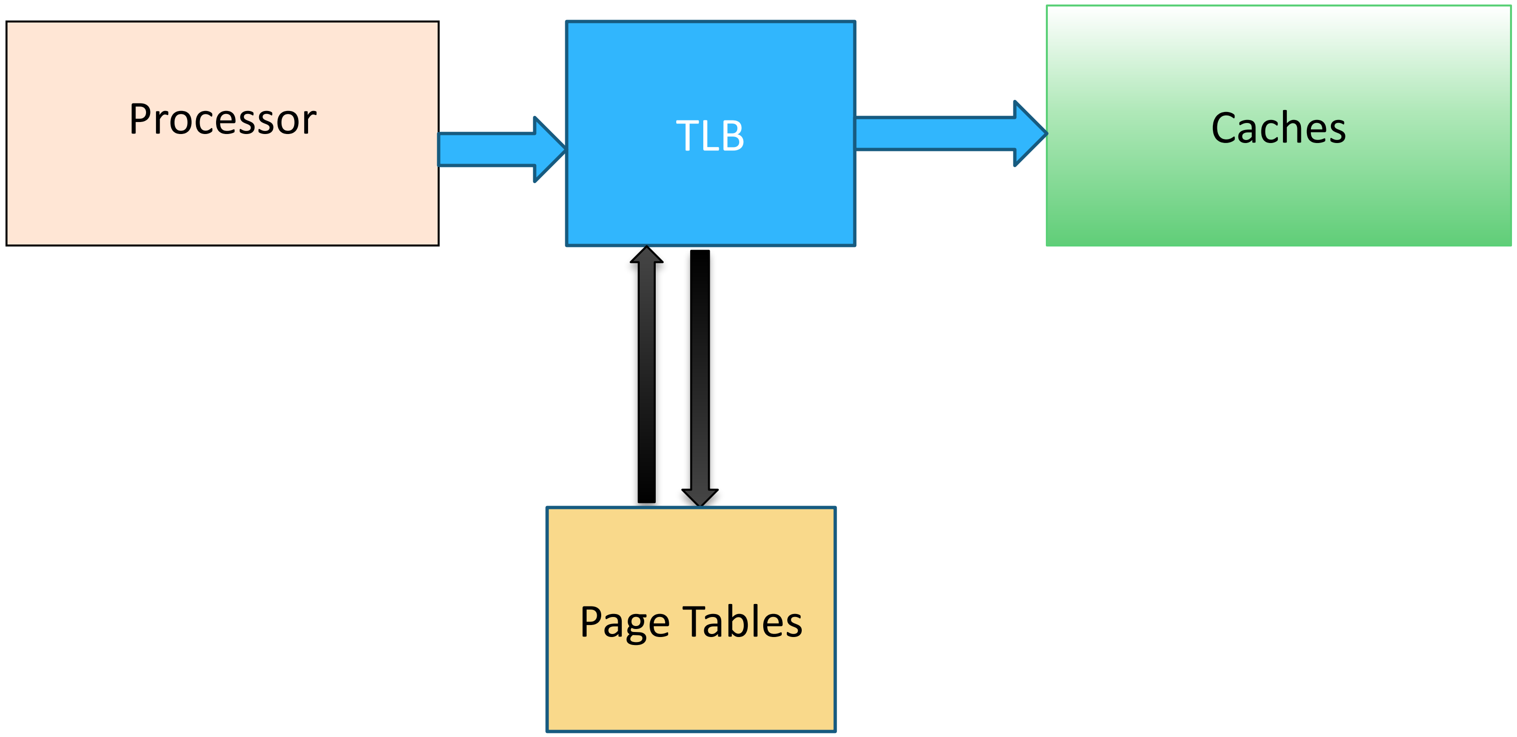
Advantage: One page table for the entire system

Memory Access



* Every access needs to go through the MMU (memory management unit)
* It will access the page tables, which themselves are stored in memory (very slow)
* Fast mechanism → Cache *N* recent mappings. Due to temporal and spatial locality, we should observe a very high hit rate. We need not access the page tables for every access.

Memory Access with a TLB



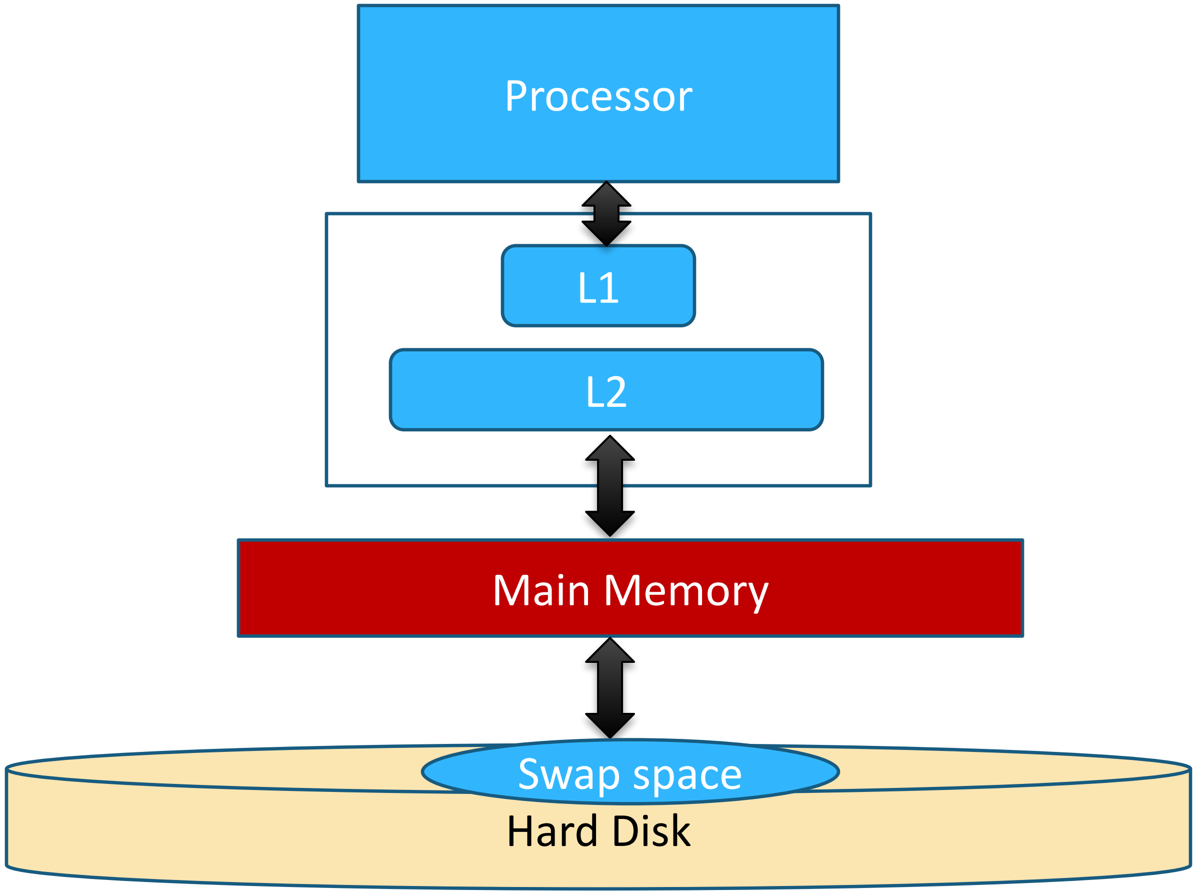
TLB

- **TLB** (Translation Lookaside Buffer)
  - A fully associative cache
  - Each entry contains a page ⟶ frame (mapping)
  - Typically contains 64 entries
  - Very few accesses go to the page table.
- Accesses that go to the **page table**
  - If there is no mapping, we have a **page fault**
  - On a page fault, create a mapping, and allocate an empty frame in memory. Update the list of empty frames.

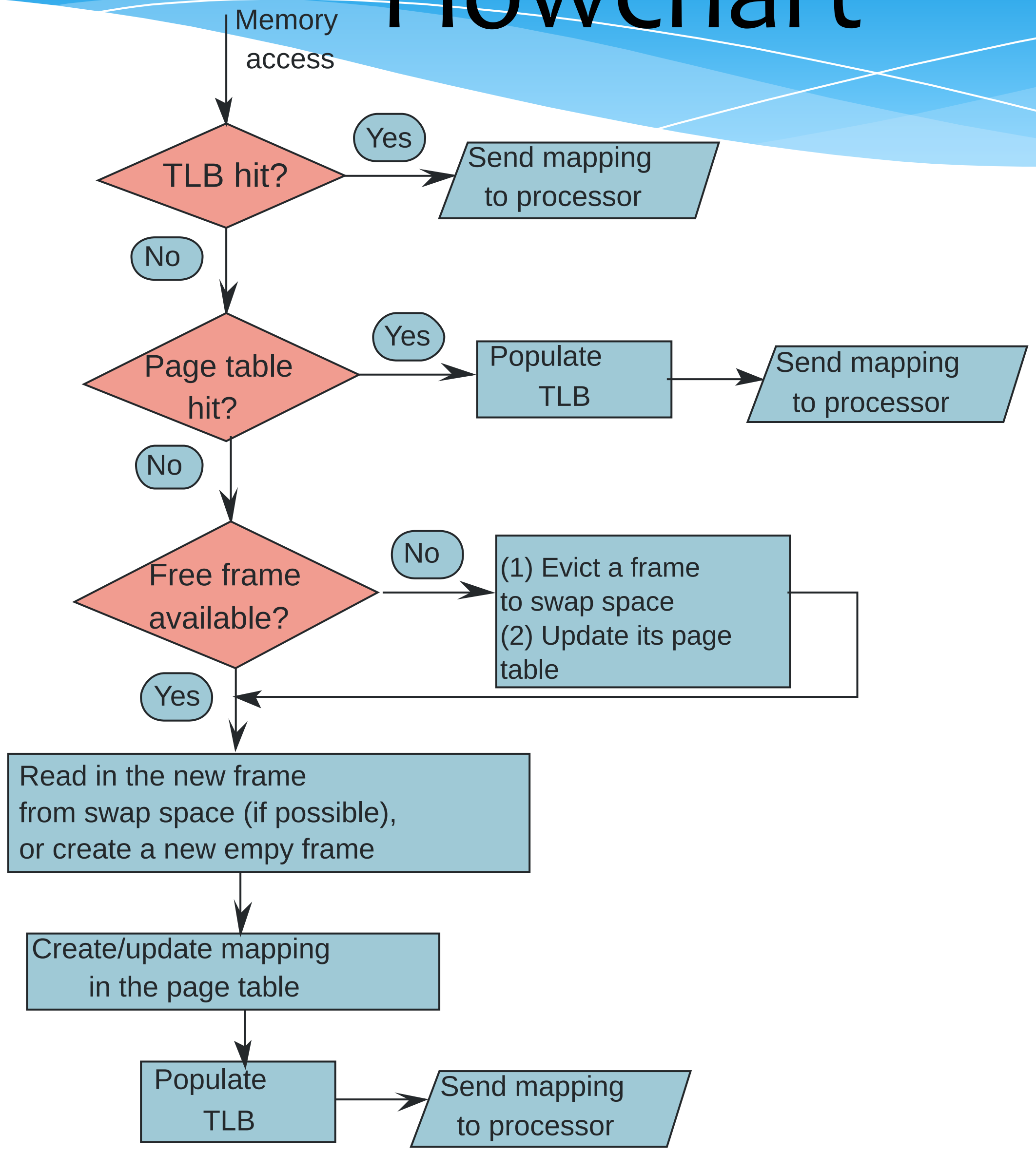
Swap Space

* Consider a system with 500 MB of main memory.
  * Can we run a program that requires 1 GB of main memory ?
  * **YES**
* Add an additional **entry** in the page table.
  * bit $\longrightarrow$ Is the frame found in main memory, or **somewhere else (???)**
  * **Hard disk** (studied later) contains a dedicated area to save frames that do not fit in main memory. This area is known as the **swap space**.

System with a Hard Disk



Flowchart


Memory access
TLB hit?
Yes
Send mapping to processor
No
Page table hit?
Yes
Populate TLB
Send mapping to processor
No
Free frame available?
No
(1) Evict a frame to swap space (2) Update its page table
Yes
Read in the new frame from swap space (if possible), or create a new empy frame
Create/update mapping in the page table
Populate TLB
Send mapping to processor

Advanced Features

- Shared Memory ⟶ Sometimes it is necessary for two processes to share data. We can map two pages in each virtual address space to the same physical frame.

- Protection ⟶ The pages in the text section are marked as read-only. The program thus cannot be modified.

THE END