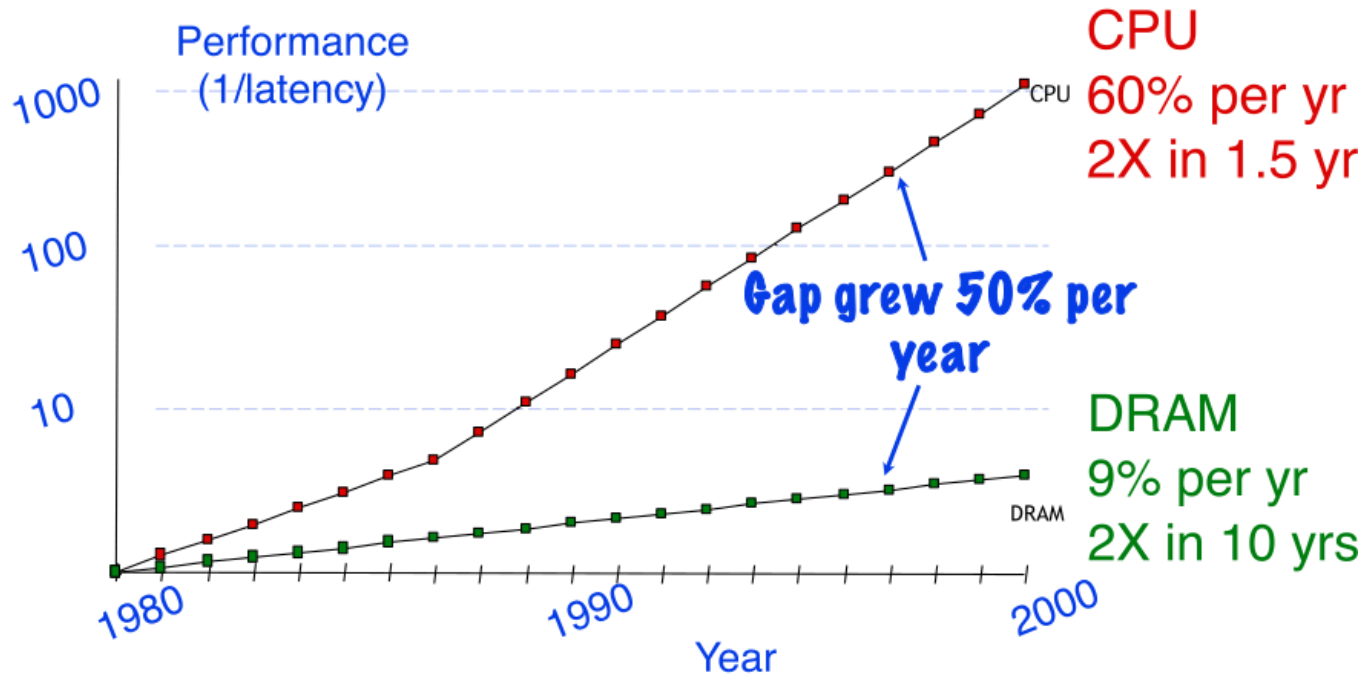
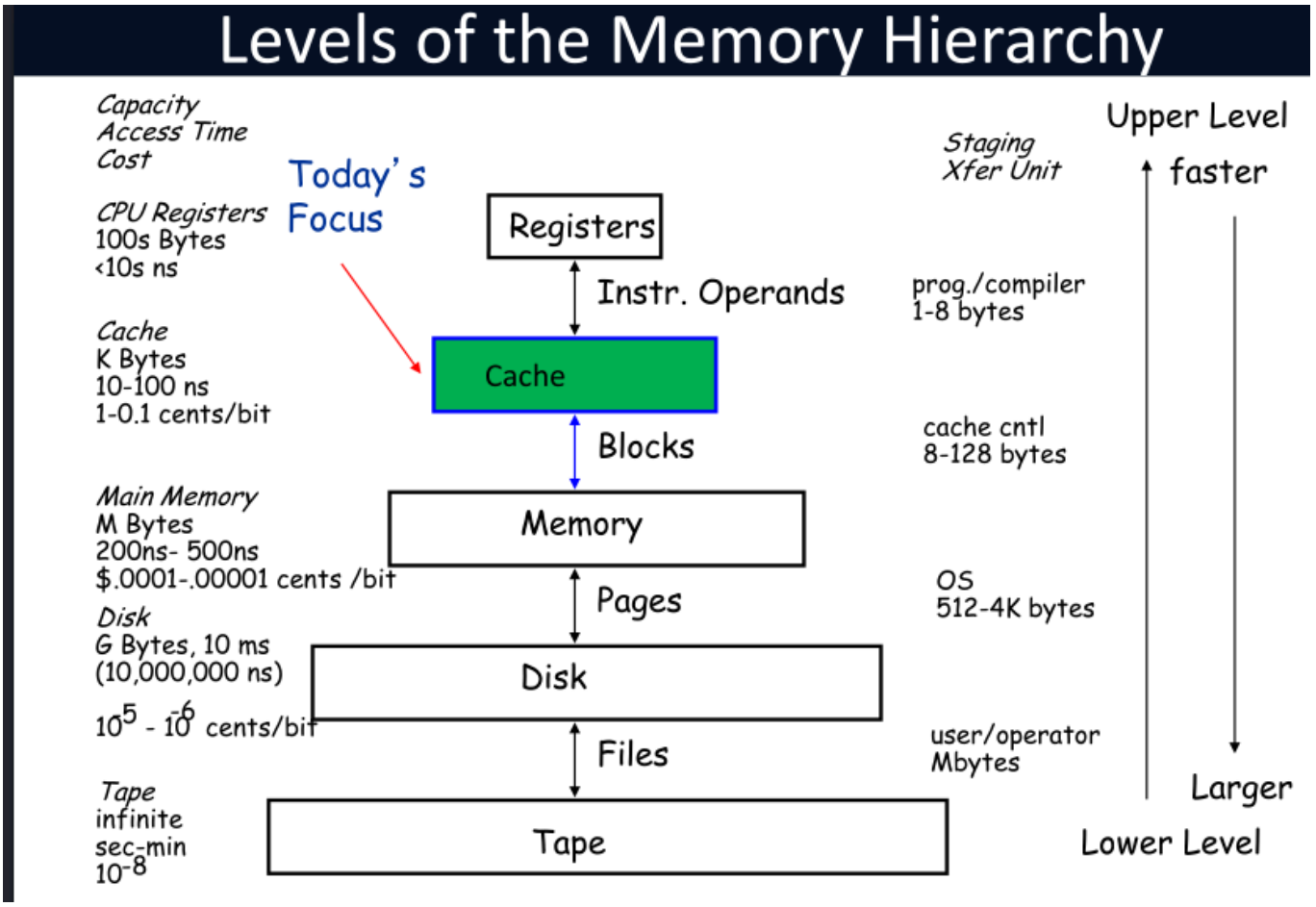
04.Cache

Most important problem in computer architecture is the gap between CPU speed and external memory speed



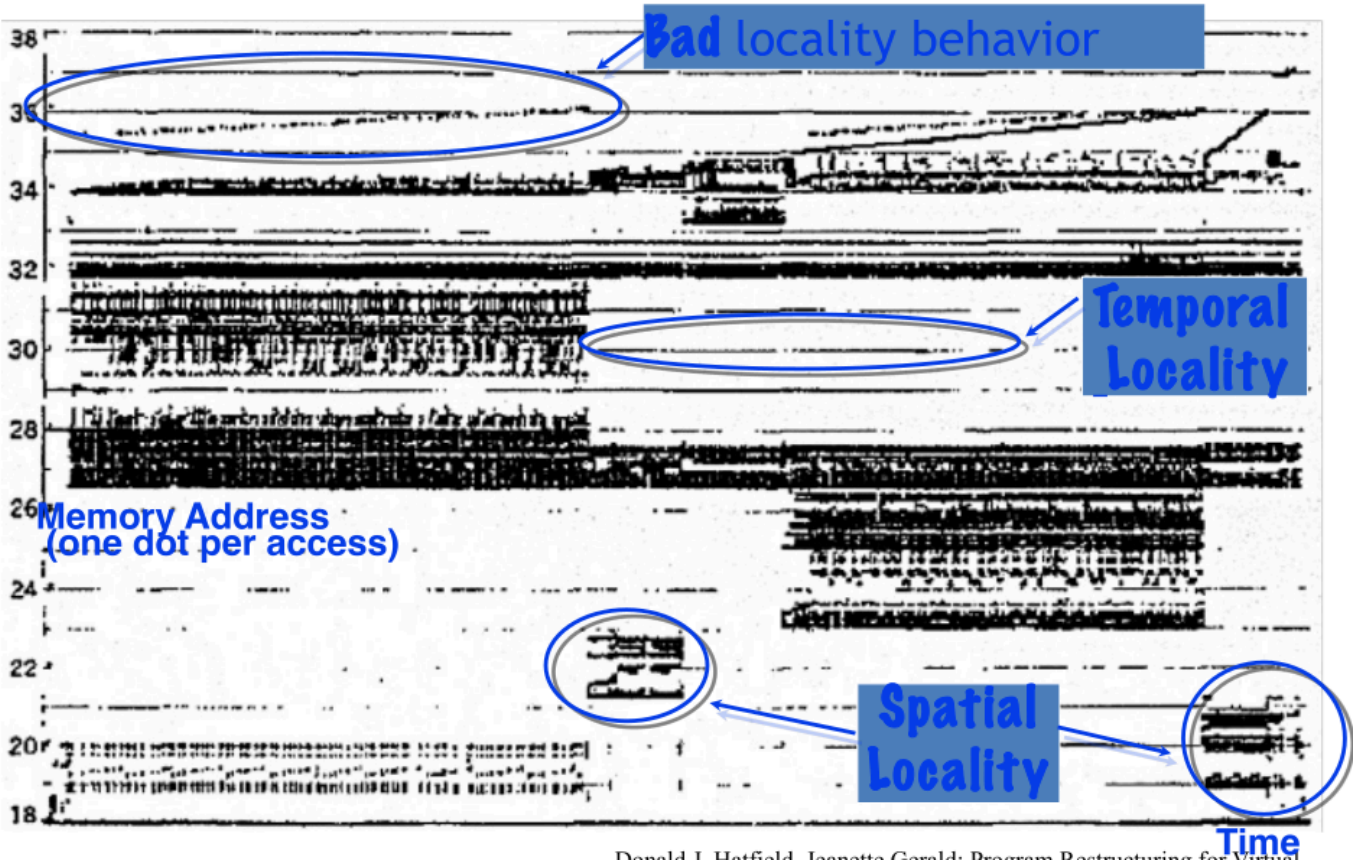
Usually every 1.5 year there is a 2x improvement in CPU speed but there is an 9% improvement in memory speed(almost constant). This could lead to a huge bottleneck. Cannot make DRAM much faster so the solution is hide this latency(Use small fast cache memories between CPU and DRAM).
Can have multiple request per clock-cycle from multi-threading, etc...(CPU arrives at more or less 409 Gb/s DRAM to 25 Gb/s). One possible solution is to optimise the access to DRAM providing multi-port system(request in parallel). Problem from multiple ports is that there could be conflicts, accesses have to be consistent. Can create different spaces but then you have to handle the partition of the data(accesses have to be handled correctly and could request the same data). Single accesses could lead to problem(guarantee WaR). Can add multiple level of cache, hiding the latency of the system, can use a third level shared at chip level. Power consumption goes up and this is a recurrent problem in architecture creation. Needs to handle it. Goal in memory hierarchy is to have the memory to seem large, fast and cheap. Want to officially access data on DRAM level but with the speed access of a register. Memory hierarchy is the combination of different modules with different cost and size and different access mechanism.



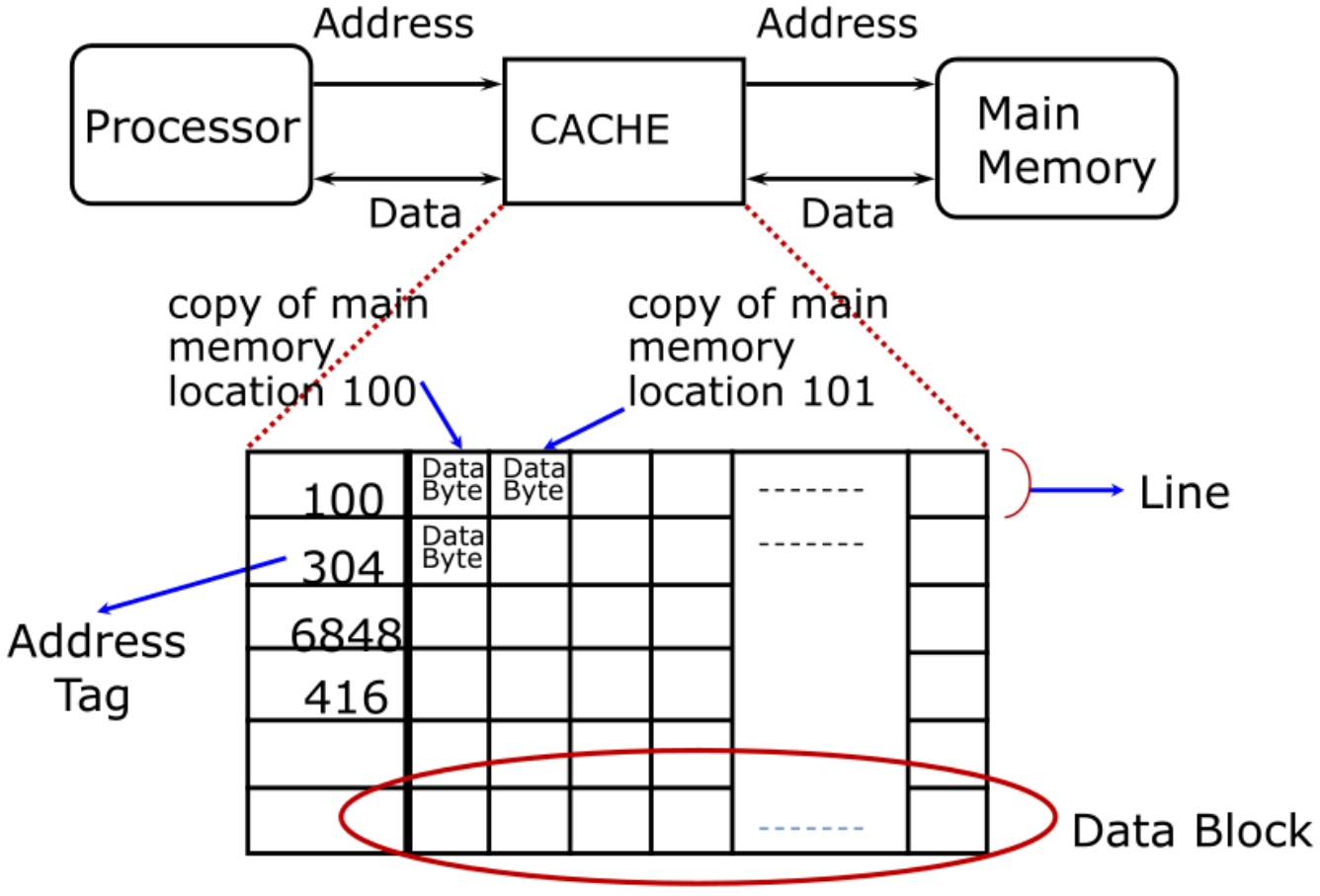
Principle of locality:

Programs access a relatively small portion of the address space at any instant of time. Instead of transferring the single data we transfer the entire block on the cache so the next access to it will be faster. If the cache is faster is true to access to all the consecutive blocks, but need to pay the first access cost.
Two predictable properties of memory references:

- Temporal Locality: If a location is referenced, it is likely to be referenced again in the near future (e.g., loops, reuse).
- Spatial Locality: If a location is referenced it is likely that locations near it will be referenced in the near future (e.g., straightline code, array access). HW based on this for the past 15 years.
  When I move to another block I pay the cost of DRAM. The more an application is data intensive the more these principles are usable. If I know how the application is structured I can organize the data to use the Spatial Locality principle even more.

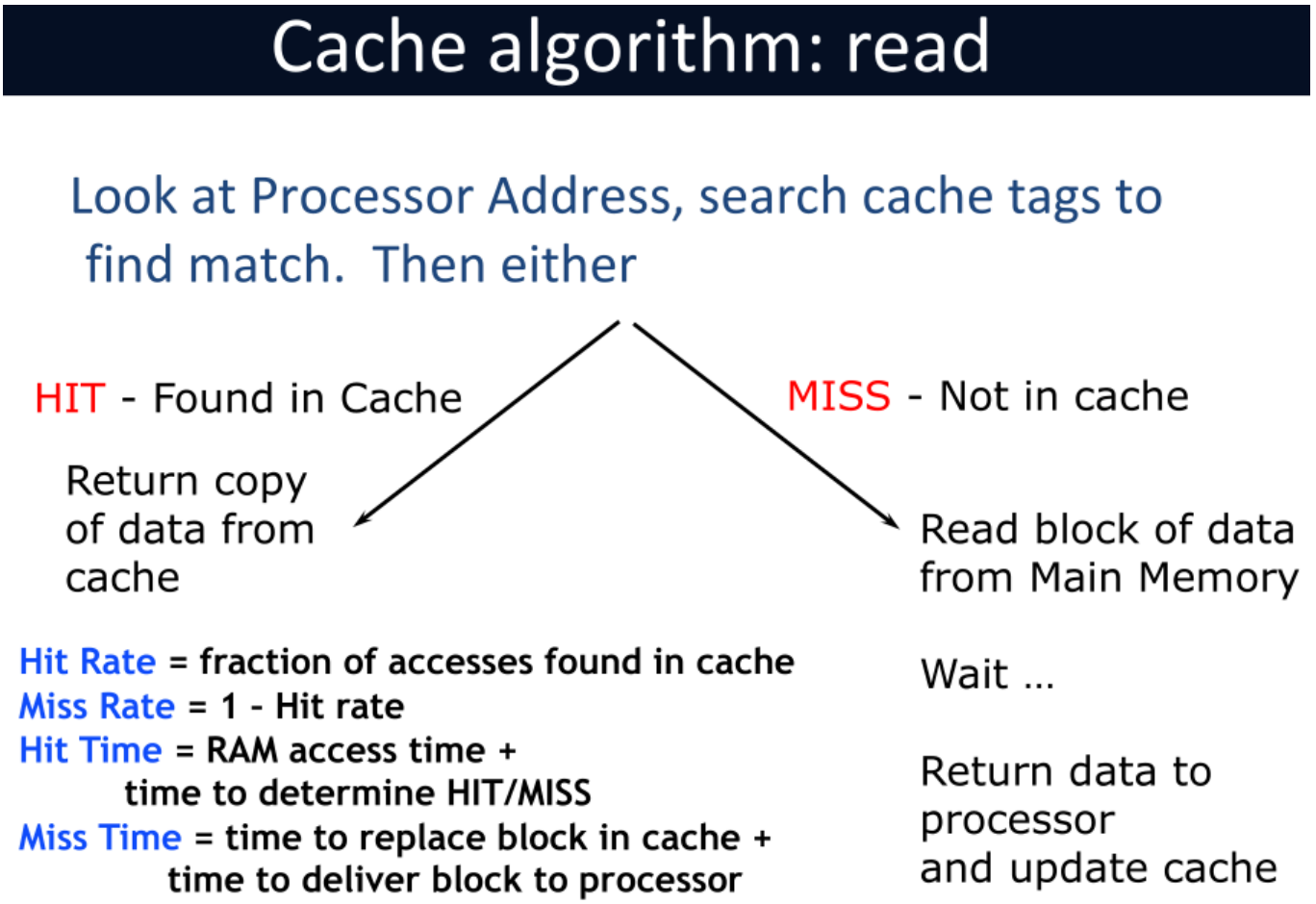


Caches



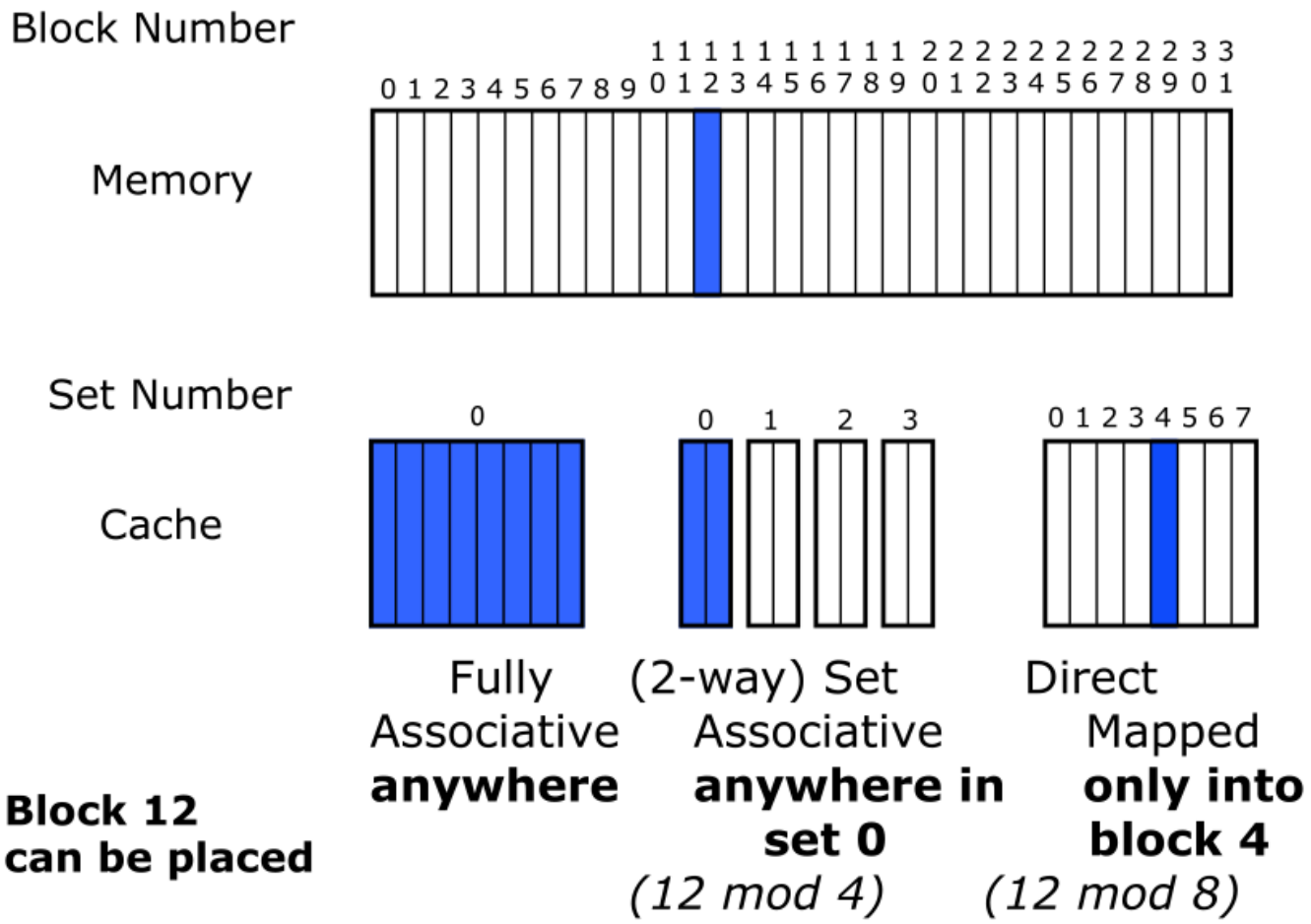
Cache exploits temporal locality and spatial locality fetching the bloc. The idea is we look at the block the process want to access to as it is written in memory. Then we can look in the cache to

see if the data are already inside the cache. If there is an hit we can return the copy in the cache. Otherwise you have to look in the DRAM. IN the beginning you are filling the cache with the miss. After a number of miss the cache will contain the data you are searching and the response will be faster. A perfect system has a high Hit Rate(Hit Rate cannot be 1 as it is needed at least one access to the memory and its related miss). Miss Rate is how much we have miss. We want to increase the Hit Rate as much as possible. Hit time is the time needed to access data in cache. Miss Time is the time needed to transfer the data to the cache after a miss.



Where can a block be placed in the cache?

Need some sort of organization otherwise the checks are on all the data stored in the cache increasing the time needed to check. Depends on the type of memory implemented: memory block that goes to the same block based on the address. Example is giving the block based on the result of module 8(easy system). The handling could result in conflicts. Another solution is the block goes anywhere.
Set Associative: solution in the middle where I divide the cache in groups and a block can goes ONLY in a specific group where it ca occupy any location. This is useful as I then only check only the location in the group corresponding to the block.

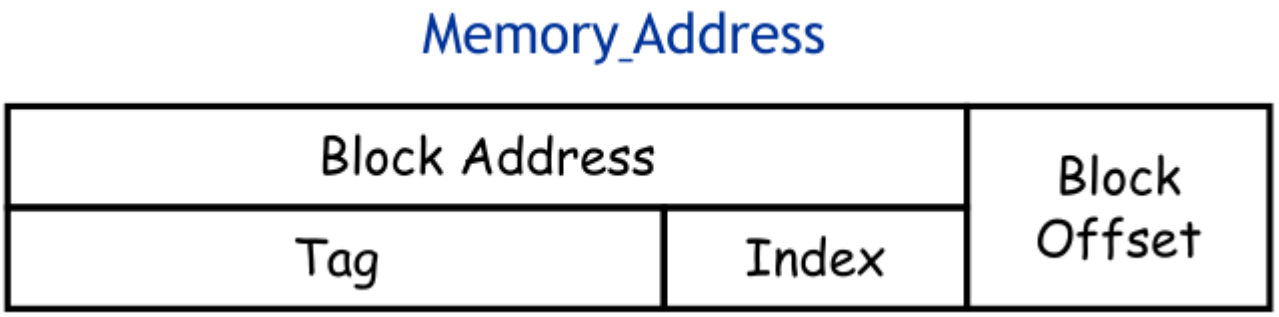


Source of cache misses:

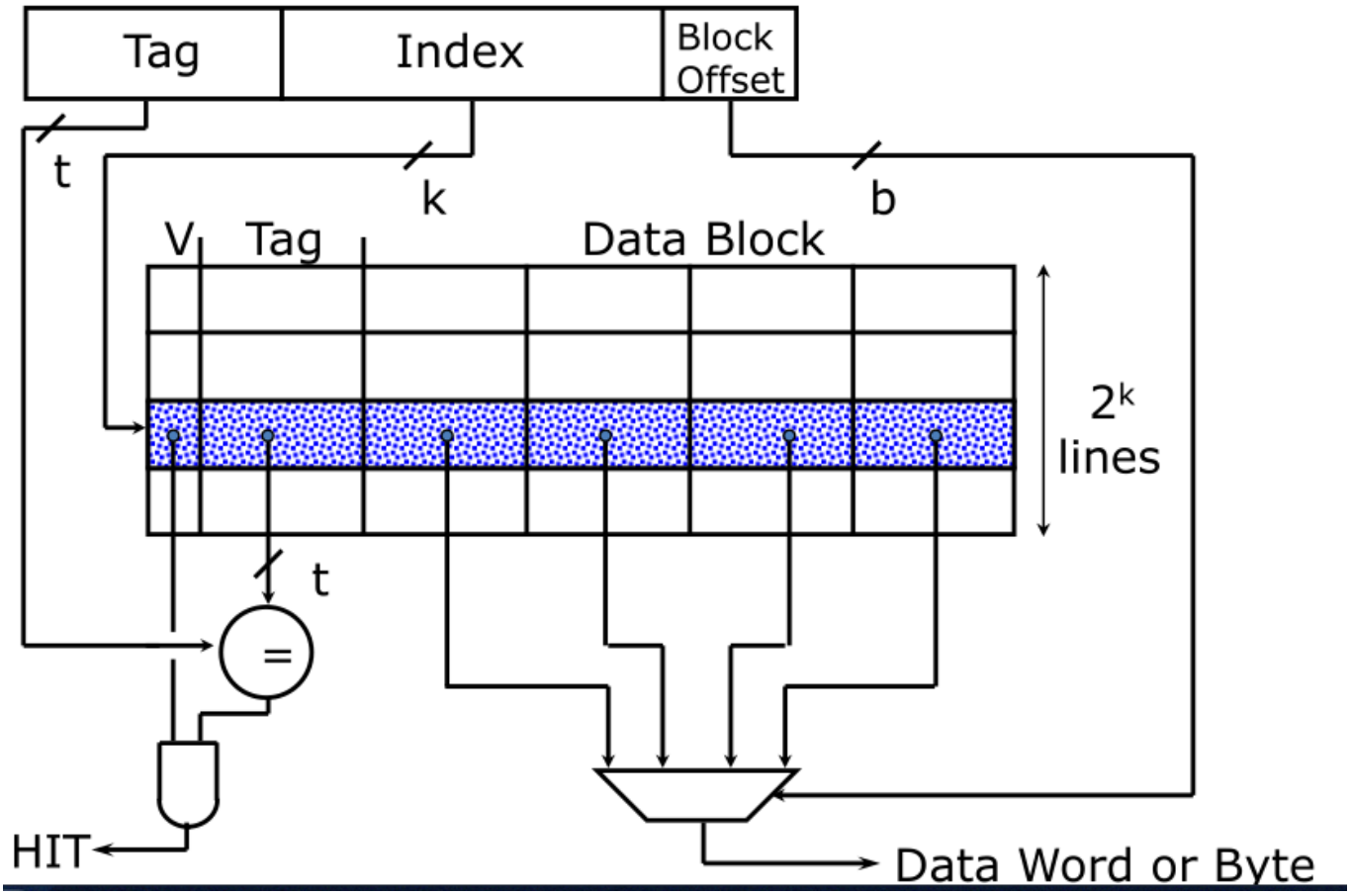
- Compulsory: cold start or process migration, first reference, first access to a block
- Capacity: cache cannot contain all blocks accessed by the program
- Conflict: multiple memory location mapped on the same cache location(Solution is to increase the cache size or increase associativity). Need to update the memory if I replace a block in the cache.
- Coherence: other process updates memory

How is a block found if it is in the cache?

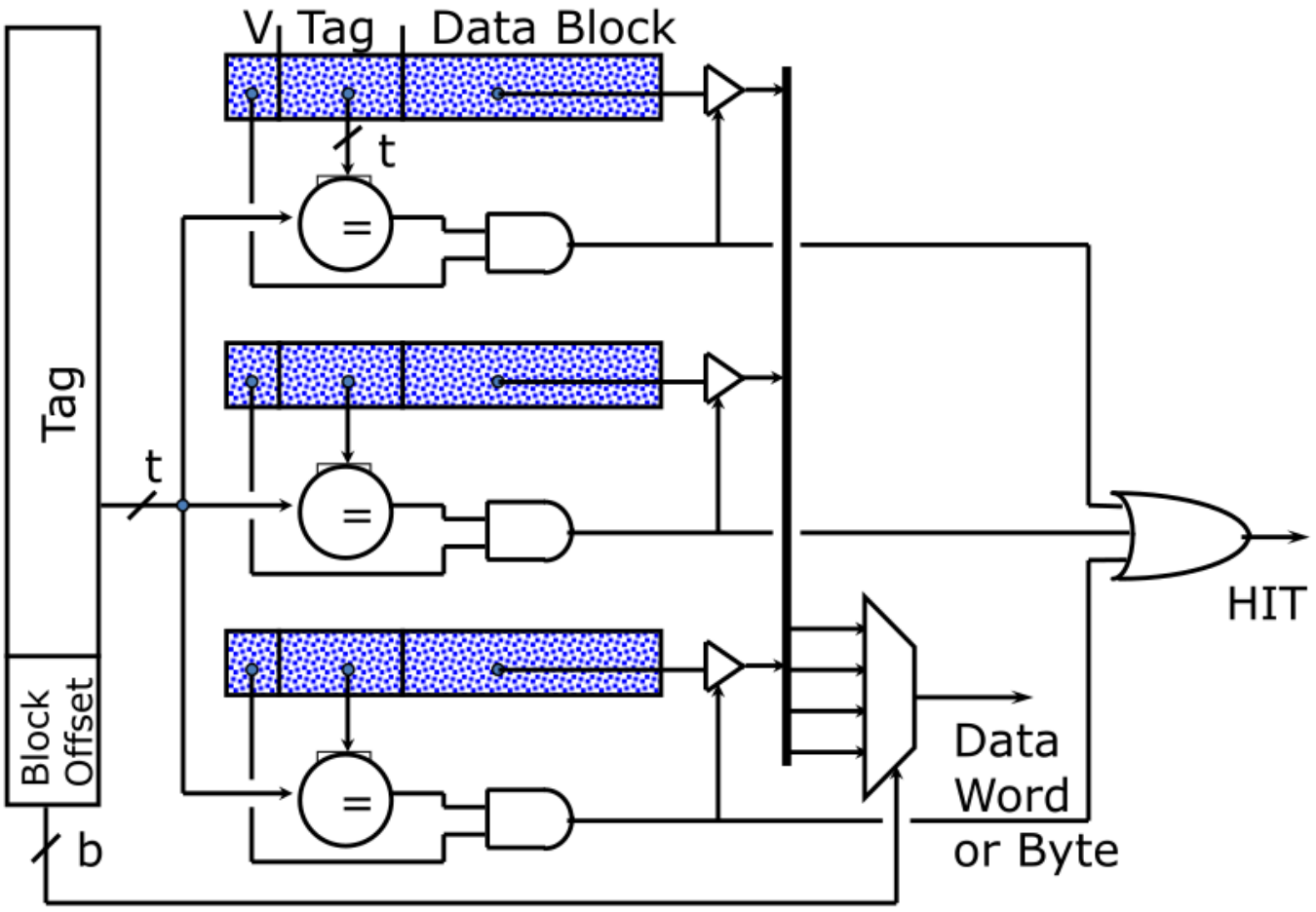
Based on the block placement in the cache.
I can decompose my address in a part related on how to identify the block in the cache and a part related on how to identify the value.



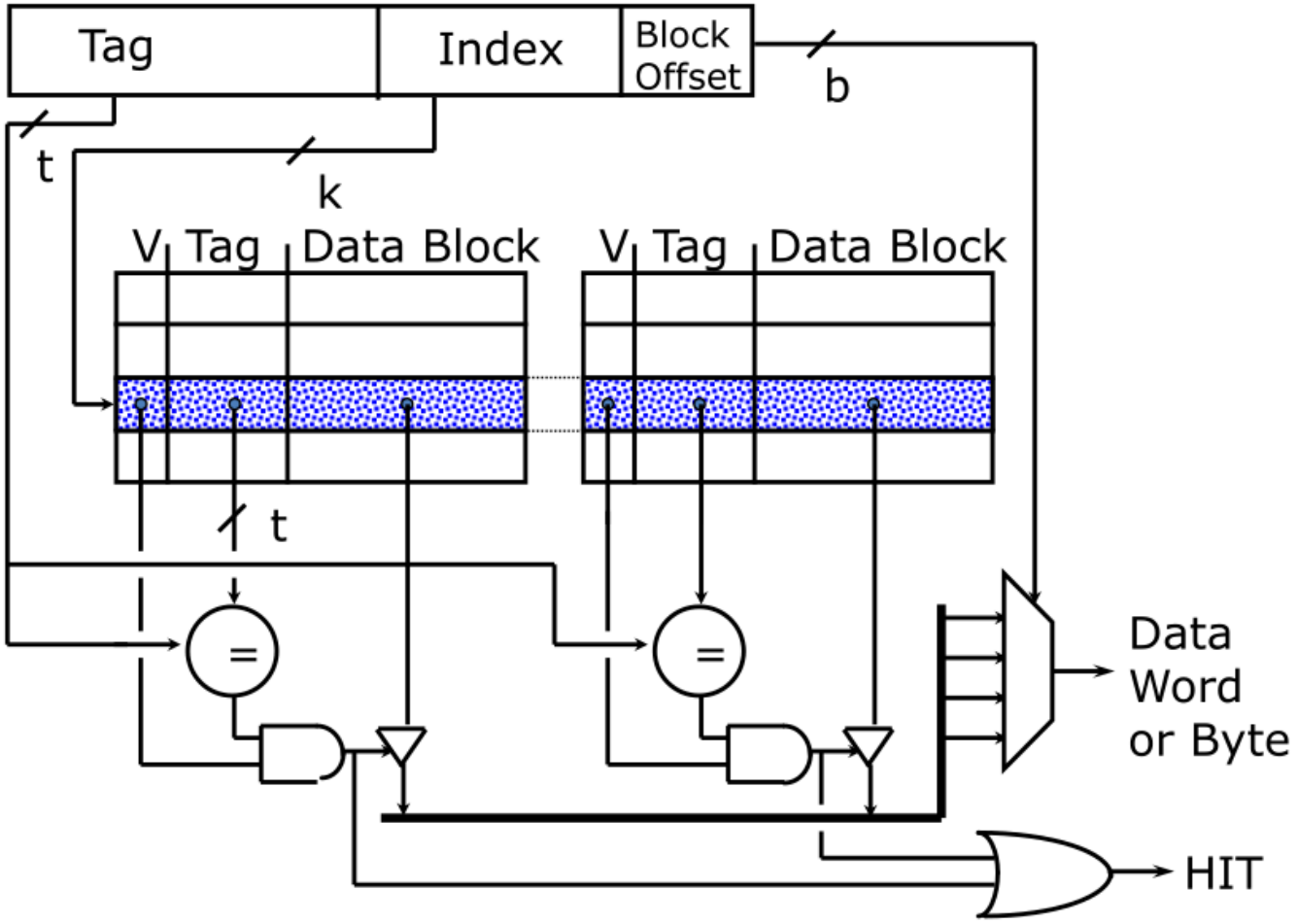
Direct-Mapped Cache



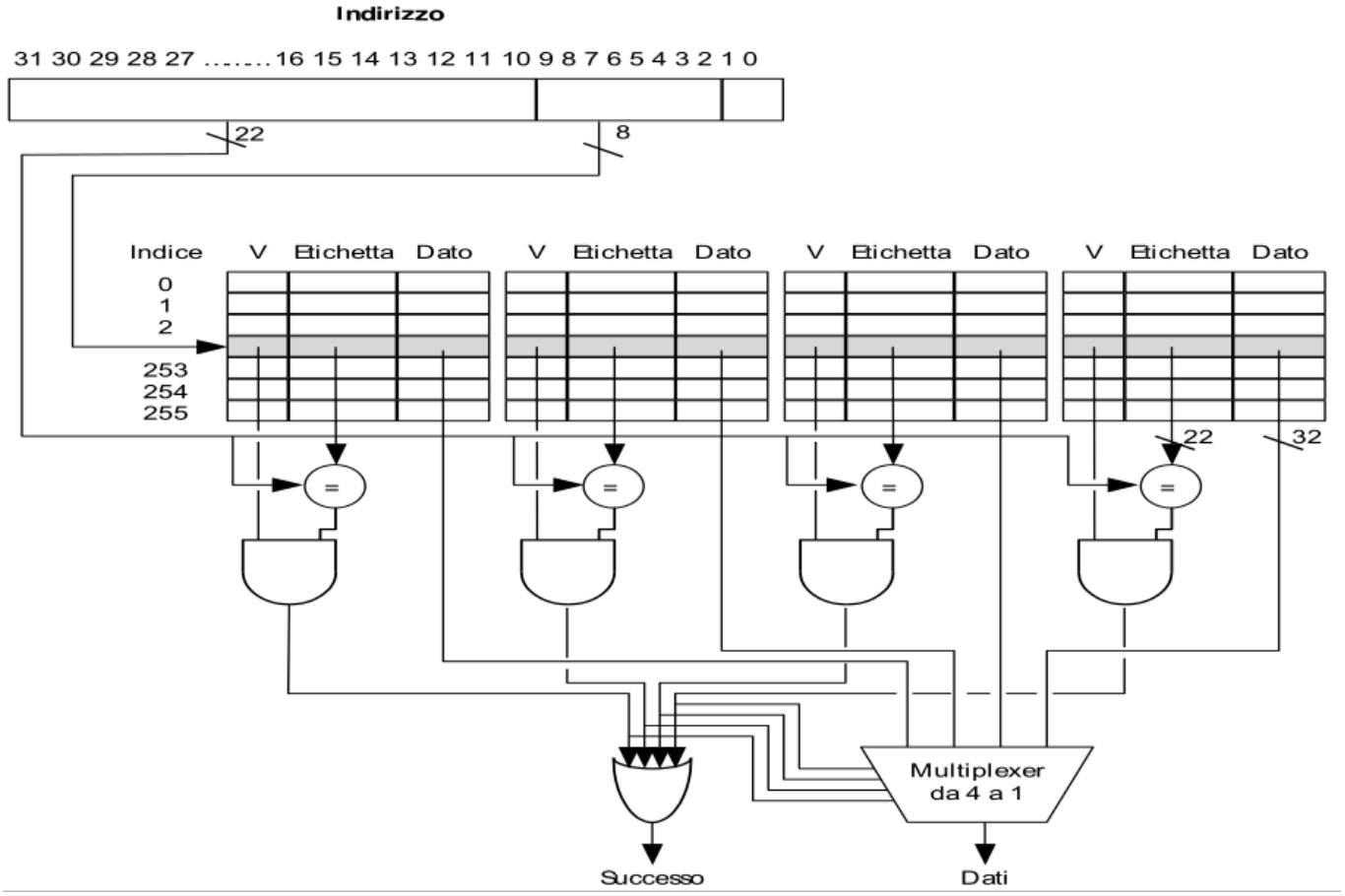
Fully Associative Cache



2-Way Set-Associative Cache

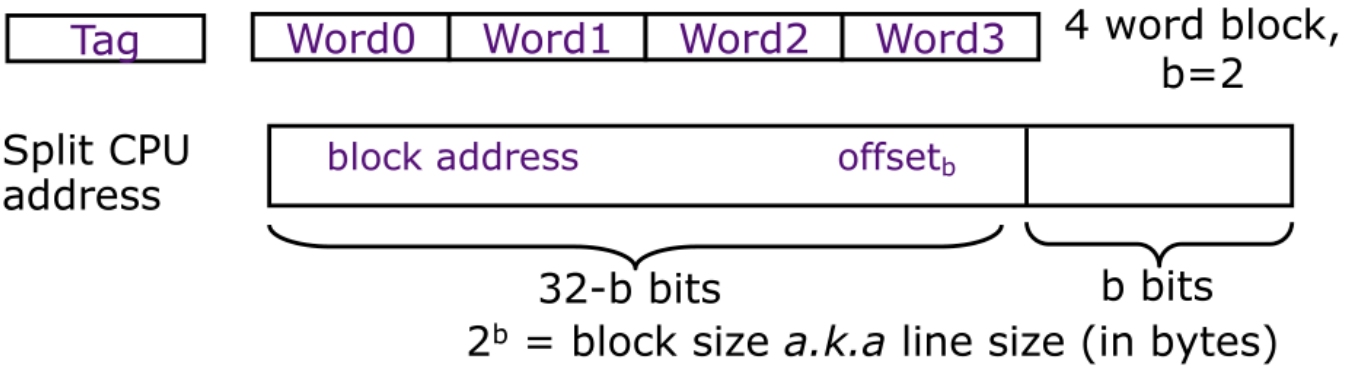


4-Way Set-Associative Cache



Block Size and Spatial Locality

Block is unit of transfer between the cache and memory



Larger block size has distinct hardware advantages

- less tag overhead
- exploit fast burst transfers from DRAM
- exploit fast burst transfers over wide busses

What are the disadvantages of increasing block size?

Fewer blocks => more conflicts. Can waste bandwidth.

Which block should be replaced on a miss?

Easy on direct access

In Set associative or Full Associative I can decide on a random method or I can replace the least recently used for the principle of locality(requires a counter resetted to 0 for every hit on the block increased for every access in memory that doesn't access the block). Can also use a method that replace the oldest block but this could increase the time used as the oldest block could be accessed recently but it doesn't require a counter.

Replacement policy has a second order effect since replacement only happens on misses.

How well random choice works

Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16K	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64K	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256K	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

I need to value if the LRU solution gives me an increase in the time that is so much bigger of the Random solution to mitigate the cost of creating the LRU solution.

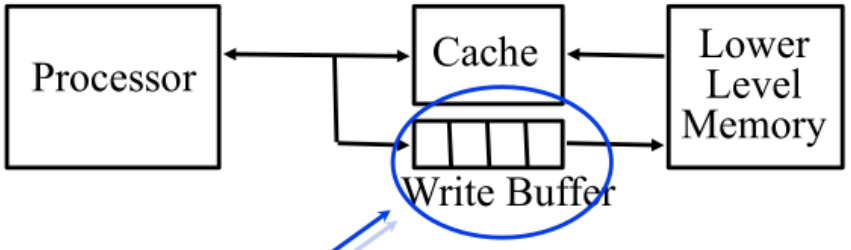
What happens on a write?

It is important because the write has to be saved in cache and in the DRAM. The cache is volatile so if the power goes off the computer use some emergency battery power to propagate the data in the cache to the DRAM. Needs to verify if the block is really valid.
I can decide to write on memory after every write in the cache. This cause an increase in the memory access and traffic(write through). I can replace the block in memory only when the block is replaced in cache(write back), this decrease the access in memory.
I could have a write during cache miss so two cases:

- no write allocate: I only write in the main memory
- write allocate: (fetch on write) I fetch in the cache transferring the block in cache and then doing the write
  Common combination:
- write through and no write allocate
- write back with write allocate

	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Additional option -- let writes to an un-cached address allocate a new cache line ("write-allocate").



Holds data awaiting write-through to lower level memory

Q. Why a write buffer ?

A. So CPU doesn't stall

Q. Why a buffer, why not just one register ?

A. Bursts of writes are common.

Q. Are Read After Write (RAW) hazards an issue for write buffer?

A. Yes! Drain buffer before next read, or check write buffers for match on reads

The buffer can reduce the time for the write as it reduce the time for the write without stalling the CPU. In any case there is a buffer and not a register as there could be multiple write at the same time or there could be RaW. You can check the buffer only where there could be a

potential hazard or don't check the buffer if it is empty.
The cache is faster than memories at lower levels of the hierarchy so the hit time much less than the time requested to access memories at lower levels (main factor of miss penalty).
The time of the penalty depends on the technology and the more the hit rate is high the more hits you have over the penalty(average access time near to hit time).
We can compute the average time as:

- Memory Stall cycles: cycles were you are waiting data for memory access(CPU is stalled).
- The cycle time includes the time necessary to manage a cache hit. So the CPU_execution_time=(CPU_clock_cycles + Memory_stall_cycles) * clock cycle time
- Memory_stall_cycles = Number_of_misses * miss_penalty

Memory_stall_cycles =
IC*(Misses/Instruction)*Miss_penalty =
IC*Reads_per_instruction*Read_miss_rate*Read_miss_penalty +
IC*Writes_per_instruction*Write_miss_rate*Write_miss_penalty

We can simplify by averaging reads and writes
Memory_stall_cycles = IC*(memory_accesses/instruction)*
miss_rate*miss_penalty

A different figure of merit
Misses/instruction = miss_rate (memory_accesses/instruction)*

Independent of the hardware implementation, dependent on the architecture (related to the average number of memory accesses per instructions)

The number of misses is indipentent from the architecture and the implementation.

$$T_A = hit\_rate*hit\_time+miss\_rate*miss\_time$$

$$= hit\_rate*hit\_time + miss\_rate*(hit\_time + miss\_penalty)$$

$$= hit\_time*(hit\_rate+miss\_rate) + miss\_rate*miss\_penalty$$

$$= hit\_time + miss\_rate*miss\_penalty .$$

Architectural choices may reduce the miss rate: ⇨ T_A nearer to *hit_time*.

In general the more you reduce the misses the more you are nearer to the hit time

Basic Cache Optimization

Reducing miss size

We can increase the block size every time we have a miss we take a longer time but htere is a higher probability that the next access would be a hit.
You can have a larger Cache size increasing the cost but reducing the capability misses.
Having a bigger associativity reducing the conflict misses.

Reducing Miss Penalty

Multilevel Caches

Reducing Hit Time

Giving Reads priority to the Writes, especially true when you have a program with many Writes.

Cache design space

There is a research to the cache dimension ongoing from 20 years. You have a very high design spaces:

- cache size

- block size
- associativity
- replacement policy
- write-through vs write-back
  Completely indipendent design policy and space, every choice is a compromise based on hit rate and miss rate. You can predict the application that will run on the architecture. Want to change the design. There is a cycle in trying to optimize the process. Trade off on the characteristic of the access and the characteristic of the space. Prefer a simple solution as has less logic and a less computational time. Latency on access time is a primal problem, then the bandwidth(multiprocessor and I/O solution, the problem is that there will be conflicts, possible only for some application), 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(PAY ATTENTION TO THE TERMINOLOGY DON'T CONFUSE THEM).
  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)
  Amdahl:
- Memory capacity should grow linearly with processor speed so if you have a processor that is faster you can do more operation in the same cycle and with a greater memory can handle these computations.
- 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
  - Prioritise some words over other one
- Wider interfaces
- Double data rate (DDR)
- Multiple banks on each DRAM device(Multiple channels technology)
  When you say I reduce of the access time of the access memory you are reducing all the

access time of the process.

Flash Memory

Type of EEPROM, Must be erased (in blocks) before being overwritten, Non volatile but the memory isn't as large than it could be, Limited number of write cycles the write is expensive(time and material as the memory becomes unusuable after some write cycles), Cheaper than SDRAM, more expensive than disk, Slower than SRAM, faster than disk.

Virtual Memory

Protection via virtual memory: keeps processes in their own memory space, don't need to know where it is going to be executed. Every program has an organization of his accesses that is ideal as he think the processor is for its use only. When you allocate space for a program you calculate the space required for it and the bias for the memory address.
Role of architecture:

- Provide user mode and supervisor mode
- Protect certain aspects of CPU state(If you have a bug in the system or processor there is a risk as there is a way to attack you)
- Provide mechanisms for switching between user mode and supervisor mode(Isolate the processes to have more Writes and not giving these Write to the user gives more protection to the system)
- Provide mechanisms to limit memory accesses(Avoid the filling of the memory)
- Provide TLB to translate addresses

Virtual Machine

Isolation of the entire execution(CPU + Memory) you cannot access a process across more VM. Classic method used in servers as you can have the illusion of having all the system for yourself, partition of all the machines across different user or handling a time multiplexing. Gives the possibility of coexist for more operating systems on the same machine. Shadow pages that are created when there is a VM. These pages handles the virtual memory used for the VM. The translation between the virtual machine and the host machine has to be secure, they are privileged operation and the CPU has to handle them.