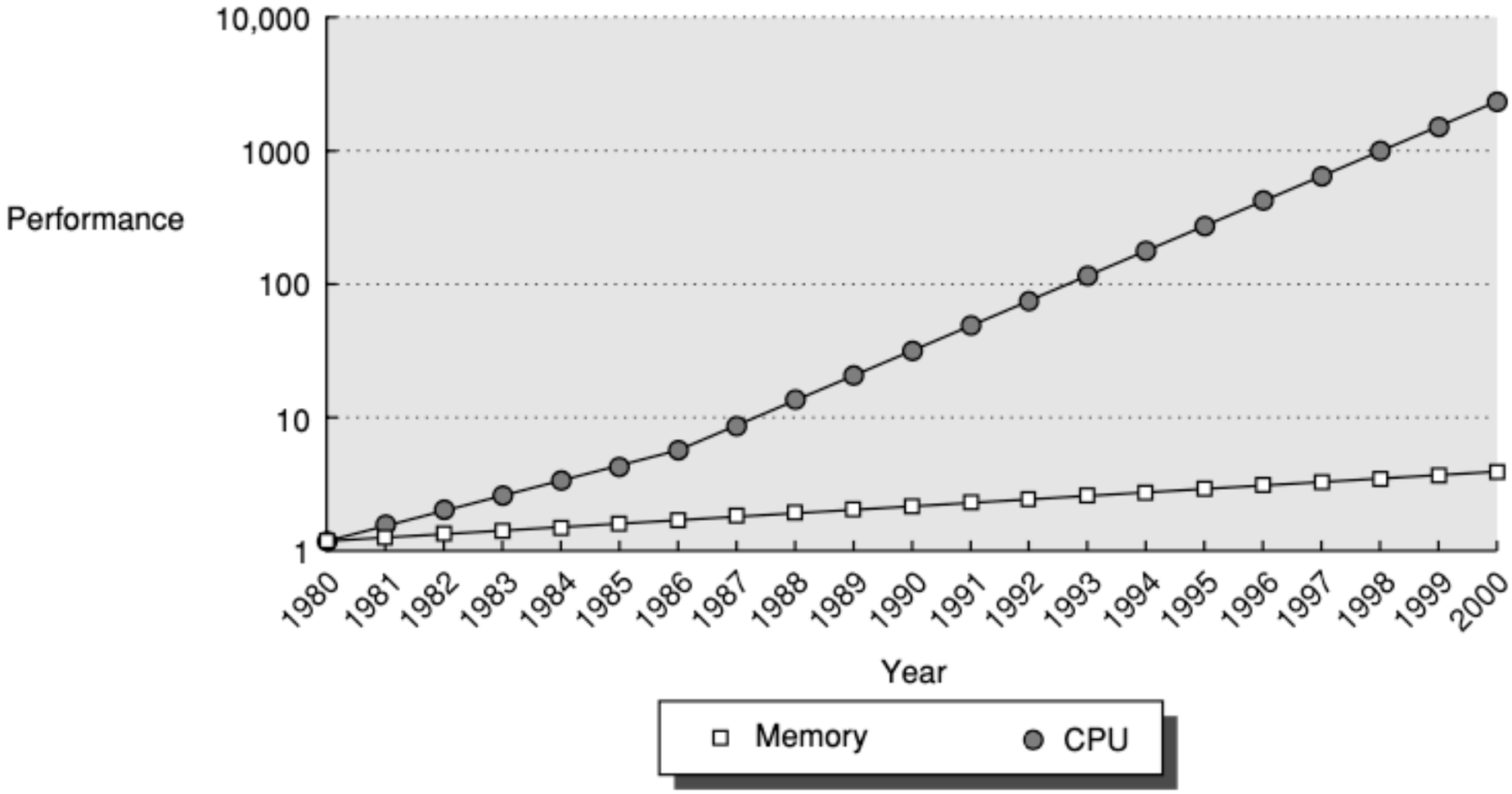
Cache and Memory Hierarchy

CSU34021 - Computer Architecture II

Memory Lagging behind…

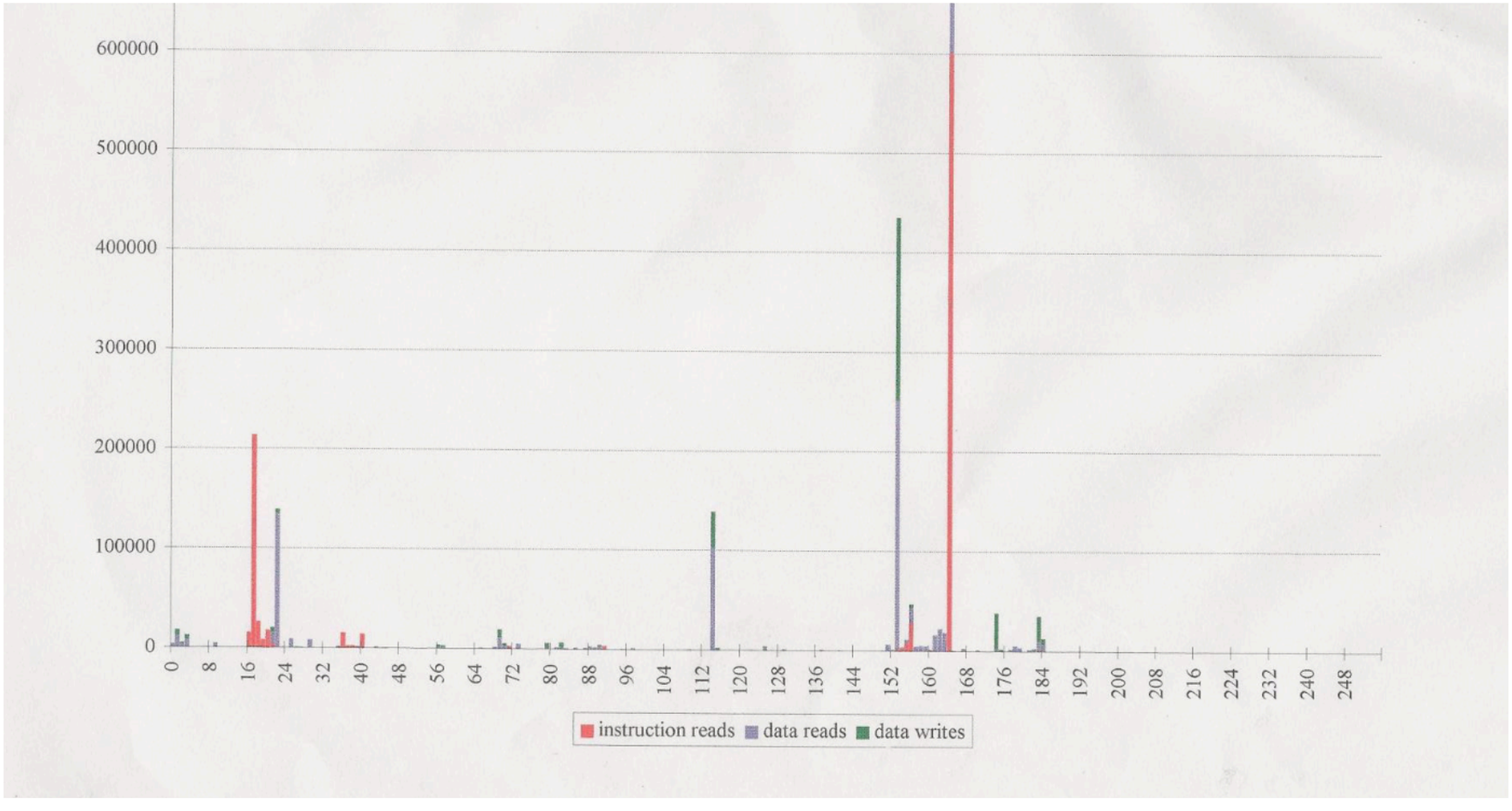


From HP
(2nd edition)

- Fast memory is expensive and cumbersome…
- However Programs require more and more memory as time passes by…

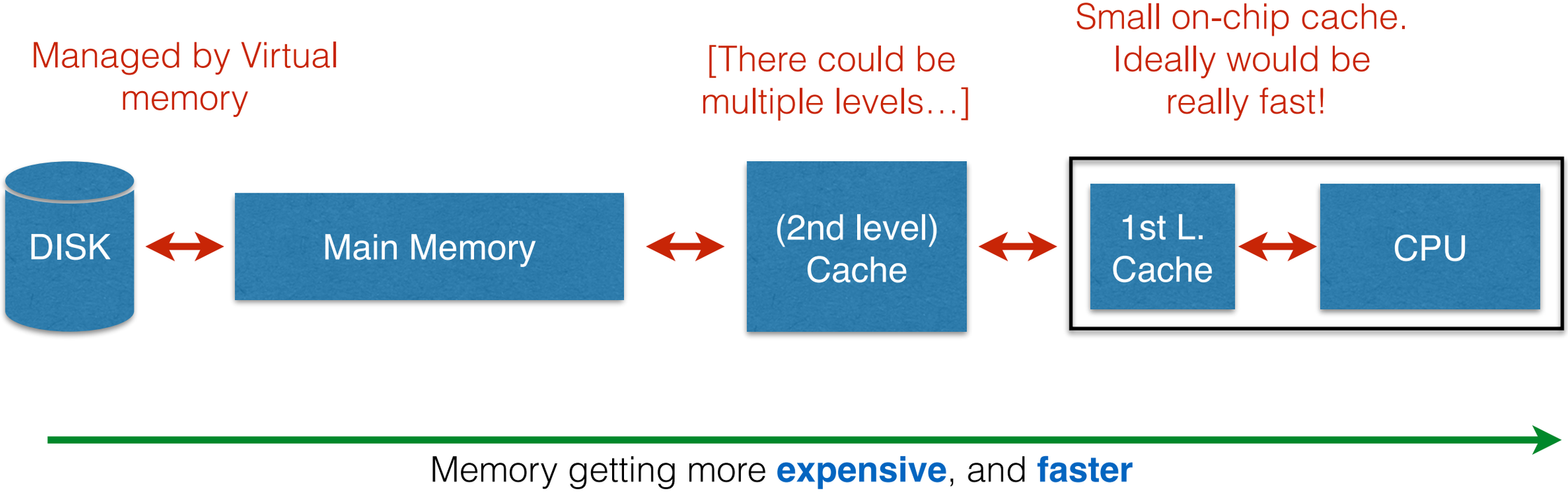
Principle of Locality

- There is hope though… Programs tend to regularly reuse data and instructions they have used recently. This is the principle of locality.
- The rule of thumb is: a program spends 90% of its execution time in only 10% of the code.



Memory Hierarchy

- Exploit the principle of locality to optimise performance/cost trade off.



Cache and Terminology

- The cache is the first level on the memory hierarchy encountered when an address leaves the CPU. It's small, but fast.

- Block: a fixed-size collection of data (analogous to page in VM).

- Cache hit: The requested address is in the cache (hurrah!)

- Cache miss: The requested address is not in the cache. The next level on the hierarchy needs to be accessed…

Quantitative Analysis of Caches

- Assume a simple system with a single cache and main memory. Then effective time for accessing memory will be:

Miss Rate

$$t_{eff} = \boxed{h}\boxed{t_{cache}} + \boxed{(1-h)}\boxed{t_{main}}$$

Hit Rate

Cache Access Time

RAM access time [Only approximation In reality **miss penalty**]

$$t_{main} = 60ns$$
$$t_{cache} = 5ns$$

h	1	0.99	0.98	0.89	0.5	0
t_eff [ns]	5	5.6	6.1	11.1	32.5	60

Caches – four questions

- Where can a block be placed? (Block placement)
- How is a block found? (Block identification)
- Which block should be replaced on a miss? (Block replacement)
- What happens on a write? (Write strategy)

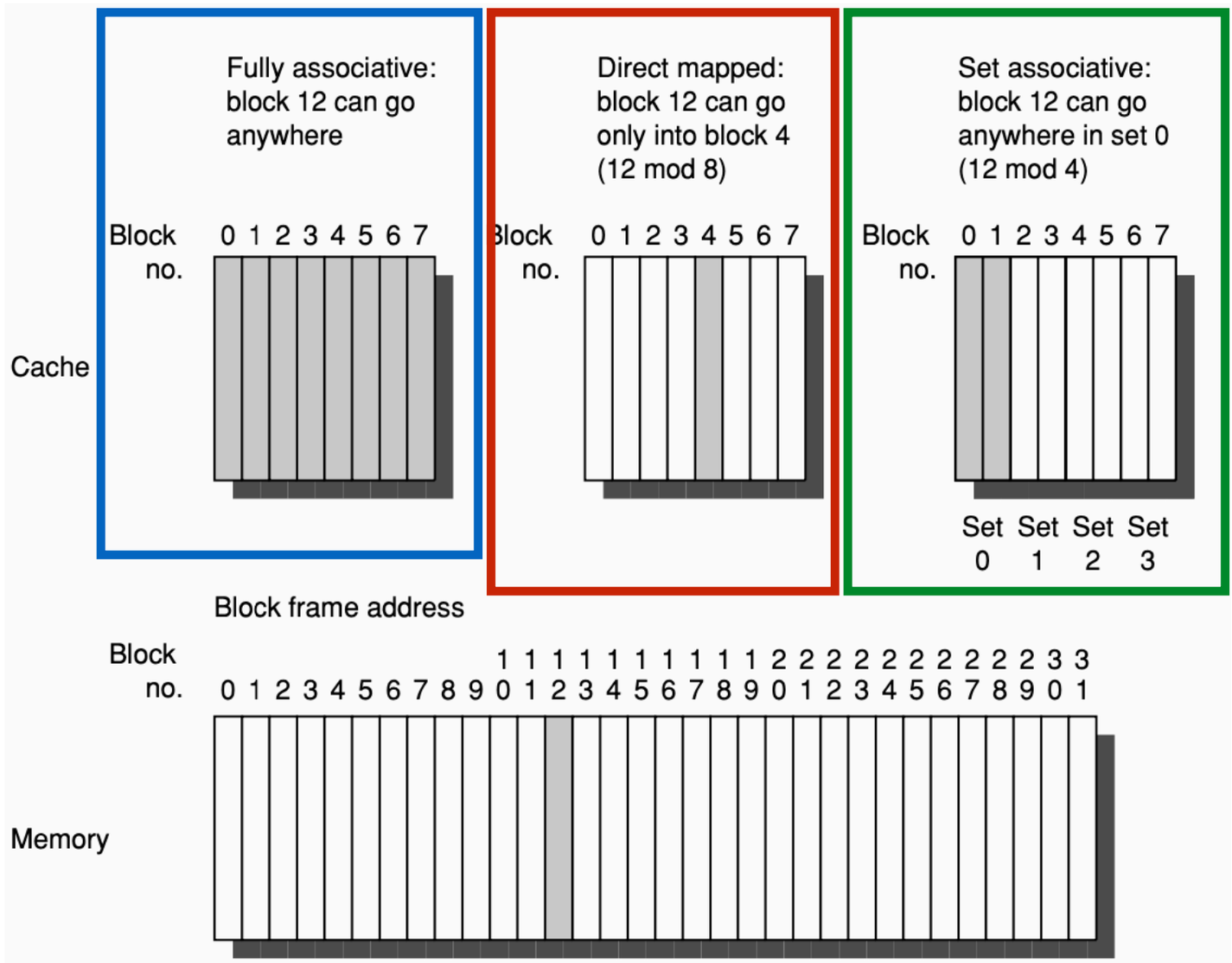
Block placement

Where to place the block from the memory in the cache?

1. Fully Associative

2. Direct mapped

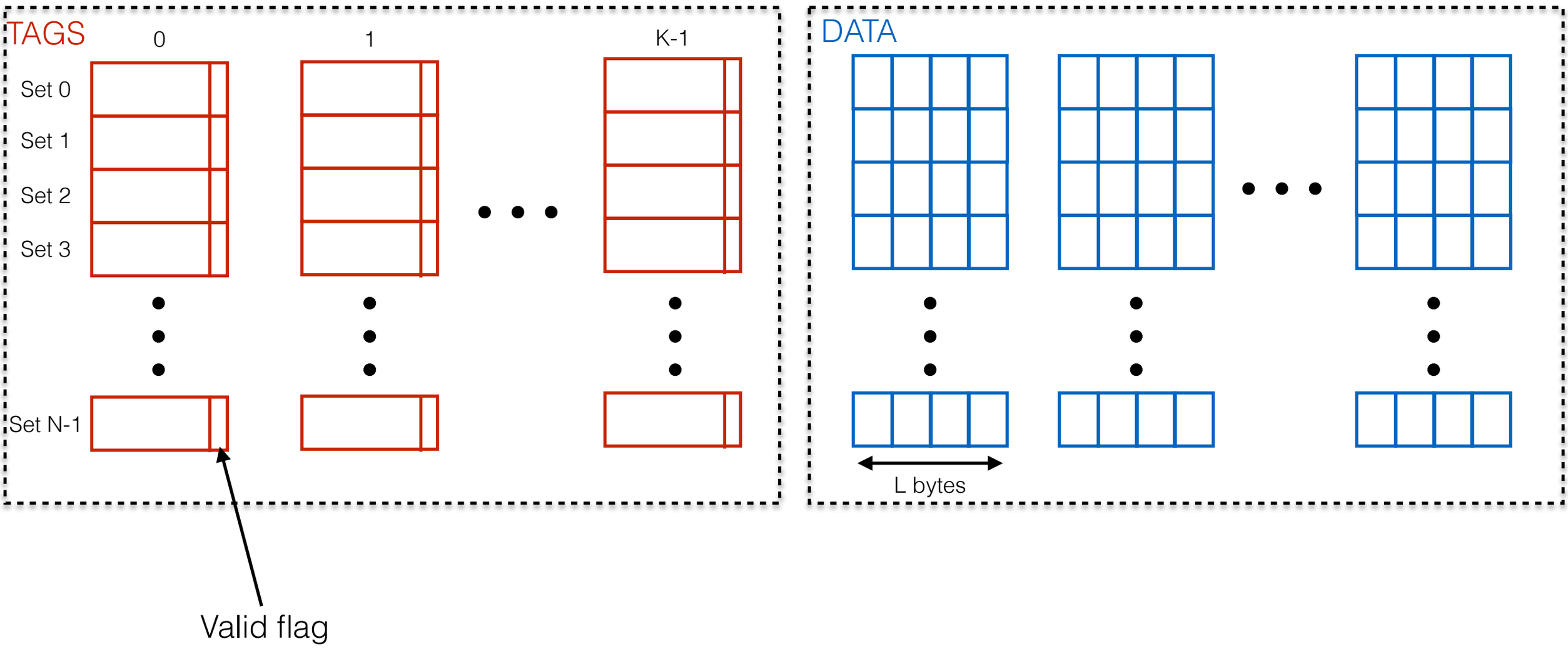
3. Set Associative [1. and 2. are particular cases…]



Majority of high level caches today are either two, four, or at most eight-way set associative… We'll see in a sec why…

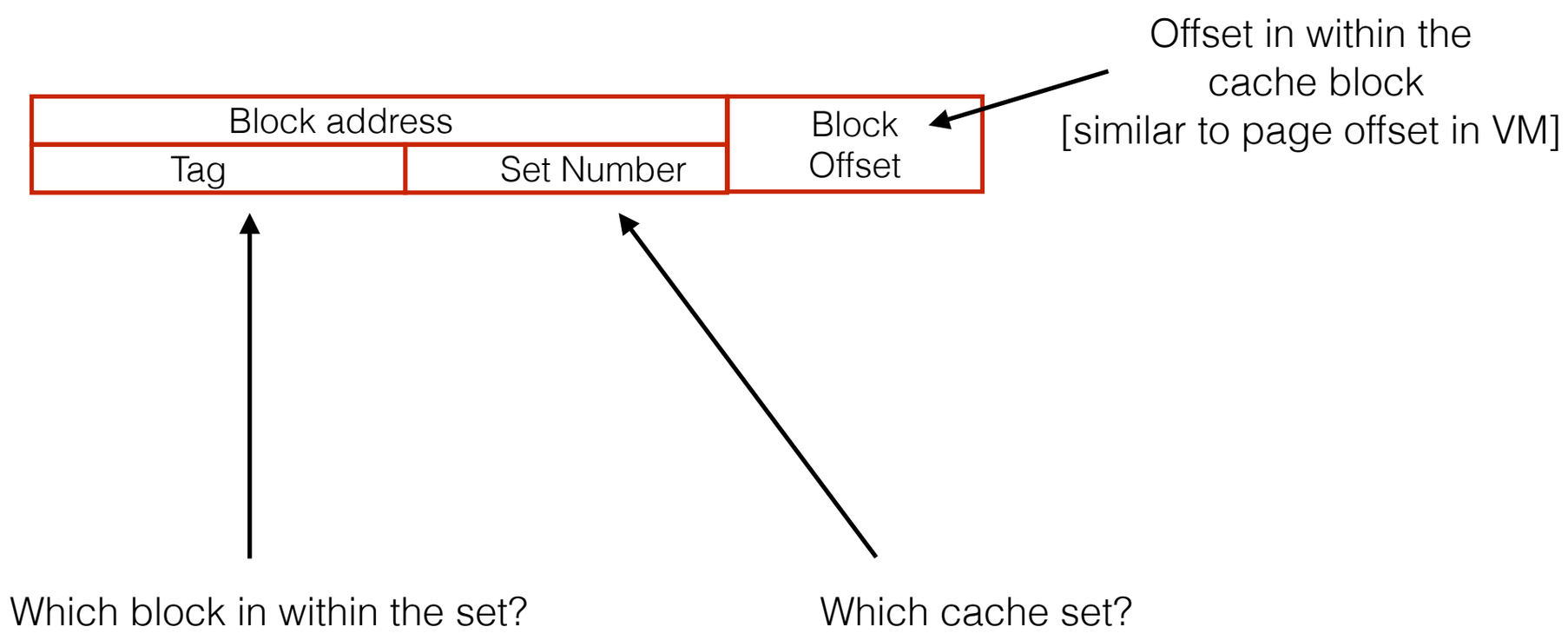
K-way associative flag organisation

Consider an a K-way associative cache, with N sets and L bytes in each block



Block identification

- Each memory address is interpreted by the cache as it follows:



A coming address is compared in parallel against all the addresses currently held in the cache. After selecting the set, only the “tag” part of the address needs to be checked.

Block Replacement

- If cache miss occurs, then data must be read from the lower level in the hierarchy, and the cache needs to be updated with the new data. Where to place the new data?

- In direct mapping, no decision needs to be made – only one block frame is checked for a hit and only one block frame can be replaced…

- On set-associative and fully-associative a decision needs to be made of which block to replace in within the set. Generally LRU, or FIFO, as a simpler approximation.

Write Strategy

- Reading from memory is much more common the writing in memory [every instruction is a memory read, and most instruction don't write in memory…].
- Luckily reading from memory is easier than writing! Why? No harm in reading more than you need! You can start reading in parallel with tag checking – you can start reading before you know how many bytes you have to read…
- But you can't do the same with writing… Two main strategies used:
- **Write-through**
- **Write-back**

Write-through

If write hit:

- update cache block and block in the lower level memory

If write miss, then two options:

- Update only lower-level memory [no-write allocate]
- Or, select/evict a cache block, write on it and on the corresponding lower-level memory block [write allocate]

Write buffer often used for optimisation purposes [data is spilled in the buffer by the CPU so that execution can go on while writing in memory occurs].

Write-back

If write hit:

- update cache block ONLY! The lower level memory is updated only when the cache block is flushed or replaced [dirty bit used for optimisation]

If write miss, then two options:

- Select/evict cache block. Write-back cache block only if dirty. Fill cache block with new data only on the cache itself.

Cache performances

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

- Techniques for cache optimisation focus on either reducing the miss rate, or reducing the miss-penalty…
- We will give a brief overview at some high-level approaches and common optimisations used nowadays…
- We start with miss-penalty…

Miss-penalty reduction: Multi-Level caches

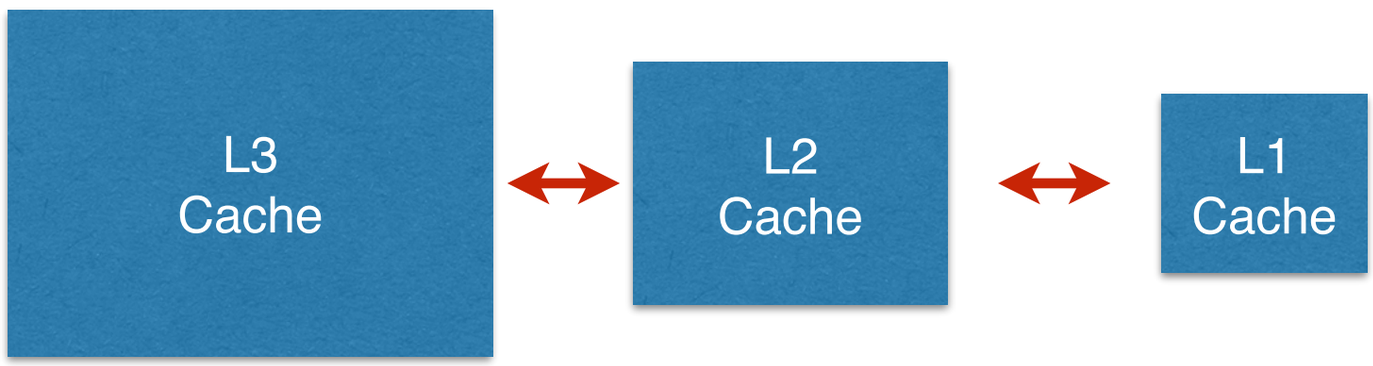
- Idea: having multiple level of caches (L1, L2, L3…). Making L1 small and fast. L2 a bit bigger and a bit slower. L3 even bigger and slower etc.

- In this way, on a cache miss, rather than having to go all the way to main memory, a miss triggers an access to the higher level of memory. Nowadays, L3 caches have sizes of main memory a few decades ago…

- How to organise blocks across the various levels?

- Multilevel-inclusion. L1 data always in L2. L2 data always in L3…

- Multilevel-exclusion. L1 data never in L2. L2 data never in L3… [good for space saving…]



Miss-penalty reduction: Critical World First and Early Restart

Generally the CPU only needs one data word from the memory block. What if we send it as soon as possible without reading the full block?

- **Critical word first**: request the missed word first from memory and send it to the CPU asap. Continue reading filling up the full block while the CPU has restored execution

- **Early restart**: Fetch the words from the block in normal order, but as soon as the requested word arrive, send it to the CPU…

  This is beneficial only with large blocks. However locality means we are very likely to need to rest of the block soon. So improvements can be small….

Miss-penalty reduction: Victim caches

- Remember what was just discarded from the cache, just in case you end up needing it soon again…
- A victim cache is a small, fully-associative cache, interleaved between the cache and its refill path, and that contains data that has recently been evicted because of a miss.
- Victim caches are checked on cache misses. If hit occurs, the victim is swapped with the corresponding block in cache.
- Generally used with a handful of entries… especially useful when the cache does not have high associativity.

A perspective on Miss-Rate

HP classified misses on three categories.

- **Compulsory**: Very first access to a block will typically be a miss.

- **Capacity**: If the cache cannot contain all the blocks needed for program execution, useful blocks will be evicted from it, causing a miss.

- **Conflict**: When cache is direct mapped or set associative, conflicts will arise when multiple useful blocks map to the same set, causing a miss.

Miss rate example

Relative frequency of miss rate types [taken from HP 2nd ed.]

Cache size	Degree associative	Total miss rate	Miss rate components (relative percent) *(Sum = 100% of total miss rate)*					
			Compulsory		Capacity		Conflict	
4 KB	1-way	0.098	0.0001	0.1%	0.070	72%	0.027	28%
4 KB	2-way	0.076	0.0001	0.1%	0.070	93%	0.005	7%
4 KB	4-way	0.071	0.0001	0.1%	0.070	99%	0.001	1%
4 KB	8-way	0.071	0.0001	0.1%	0.070	100%	0.000	0%
8 KB	1-way	0.068	0.0001	0.1%	0.044	65%	0.024	35%
8 KB	2-way	0.049	0.0001	0.1%	0.044	90%	0.005	10%
8 KB	4-way	0.044	0.0001	0.1%	0.044	99%	0.000	1%
8 KB	8-way	0.044	0.0001	0.1%	0.044	100%	0.000	0%
16 KB	1-way	0.049	0.0001	0.1%	0.040	82%	0.009	17%
16 KB	2-way	0.041	0.0001	0.2%	0.040	98%	0.001	2%
16 KB	4-way	0.041	0.0001	0.2%	0.040	99%	0.000	0%
16 KB	8-way	0.041	0.0001	0.2%	0.040	100%	0.000	0%
32 KB	1-way	0.042	0.0001	0.2%	0.037	89%	0.005	11%
32 KB	2-way	0.038	0.0001	0.2%	0.037	99%	0.000	0%
32 KB	4-way	0.037	0.0001	0.2%	0.037	100%	0.000	0%
32 KB	8-way	0.037	0.0001	0.2%	0.037	100%	0.000	0%
64 KB	1-way	0.037	0.0001	0.2%	0.028	77%	0.008	23%
64 KB	2-way	0.031	0.0001	0.2%	0.028	91%	0.003	9%
64 KB	4-way	0.030	0.0001	0.2%	0.028	95%	0.001	4%
64 KB	8-way	0.029	0.0001	0.2%	0.028	97%	0.001	2%
128 KB	1-way	0.021	0.0001	0.3%	0.019	91%	0.002	8%
128 KB	2-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
128 KB	4-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
128 KB	8-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
256 KB	1-way	0.013	0.0001	0.5%	0.012	94%	0.001	6%
256 KB	2-way	0.012	0.0001	0.5%	0.012	99%	0.000	0%
256 KB	4-way	0.012	0.0001	0.5%	0.012	99%	0.000	0%
256 KB	8-way	0.012	0.0001	0.5%	0.012	99%	0.000	0%
512 KB	1-way	0.008	0.0001	0.8%	0.005	66%	0.003	33%
512 KB	2-way	0.007	0.0001	0.9%	0.005	71%	0.002	28%
512 KB	4-way	0.006	0.0001	1.1%	0.005	91%	0.000	8%
512 KB	8-way	0.006	0.0001	1.1%	0.005	95%	0.000	4%

How to reduce miss rate?

- Details can be found in HP. We just give an high level description.
- Compulsory misses can be mitigated by techniques that do *blocks prefetching* of blocks or *way-prediction*.
- There is little that can be done about capacity rather than increasing the cache size, and the compiler performing software optimisation [example later].
- Conflict misses can be mitigated by increasing the associativity of the cache [though there are diminishing returns for higher associativity, and higher associativity increases hit time].
- Another technique that works across the board is increasing the cache block size. But while decreasing miss rate, it increases miss penalty…

Example: Compiler optimisation

- Modern day compilers try to optimise the code in such a way that it might better fit in the cache. One such technique is Loop Interchange.

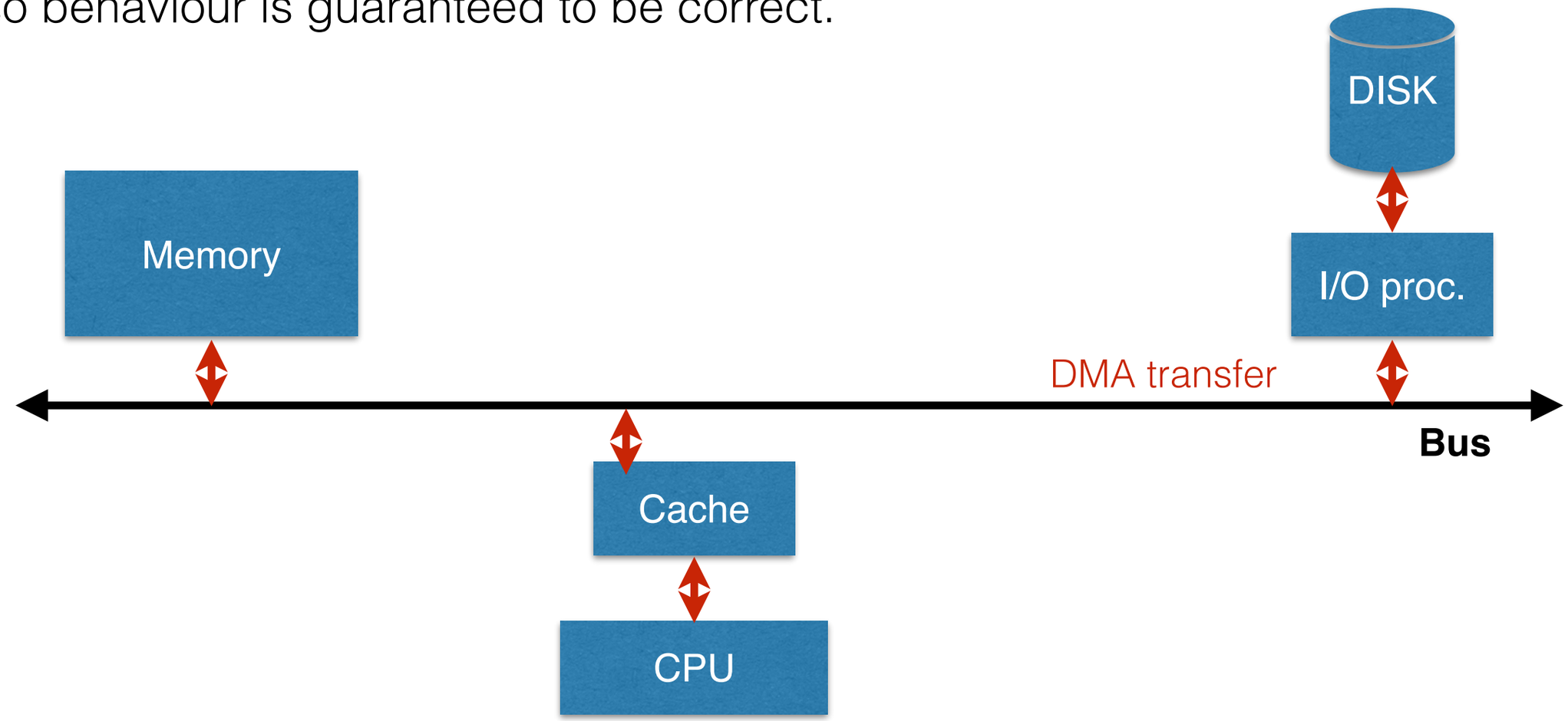
```
/* Before */
for (j = 0; j < 100; j = j+1)
        for (i = 0; i < 5000; i = i+1)
                x[i][j] = 2 * x[i][j];

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

- Another technique is blocking. It exploits temporal locality by dealing with sub-blocks of matrixes/arrays when doing certain operations [e.g. matrix multiplication].

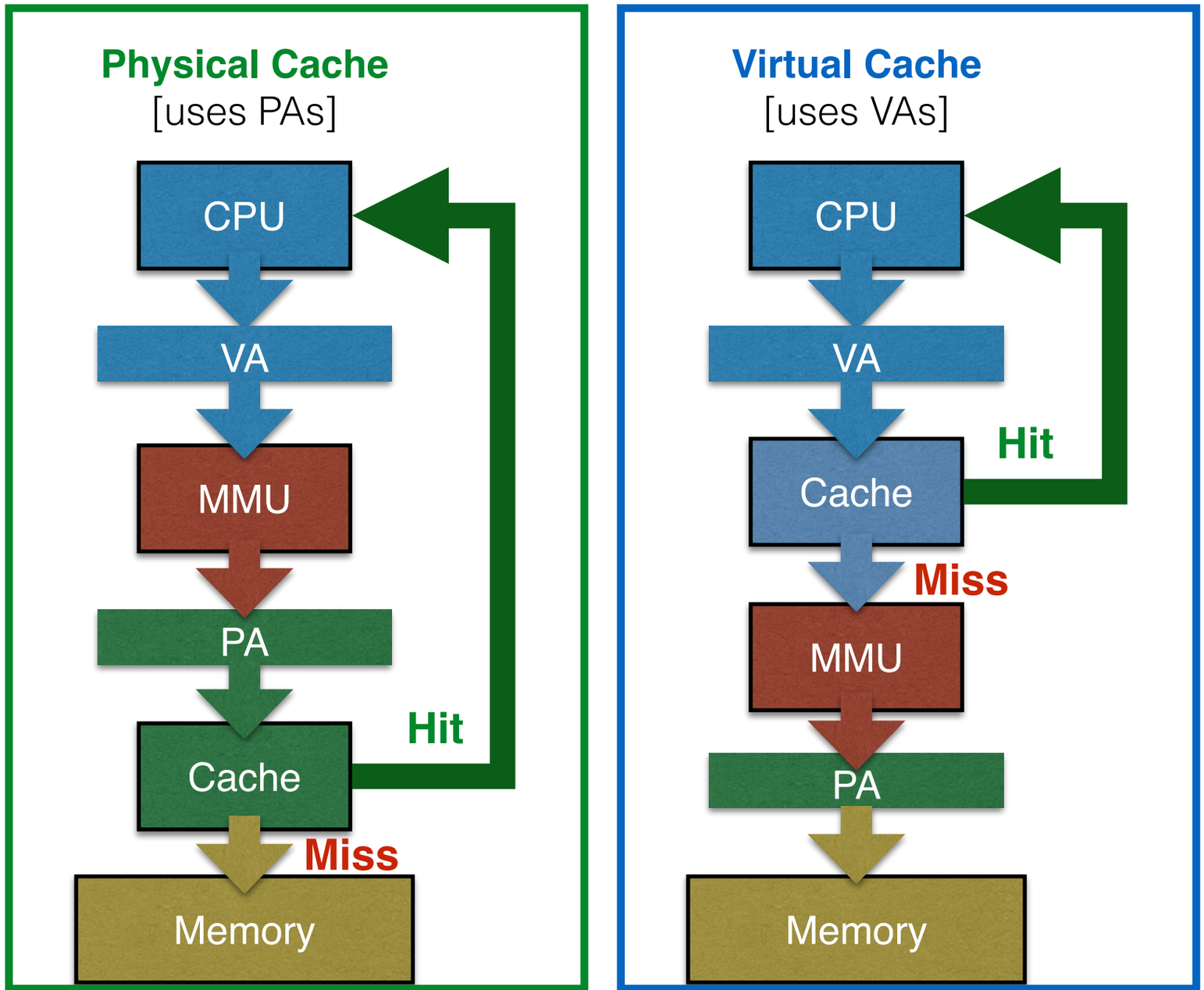
Cache Coherency on I/O

- Consider an I/O process that writes to RAM via a Direct Memory Access (DMA).
- If DMA writes on a location currently in the cache, the cache must keep track of this!
- The cache watches (snoops) the DMA bus. If it notices a write operation at a memory location it currently holds, it will invalidate its entry.
- The next time that location is accessed by the program, the invalid entry will trigger a miss, so behaviour is guaranteed to be correct.



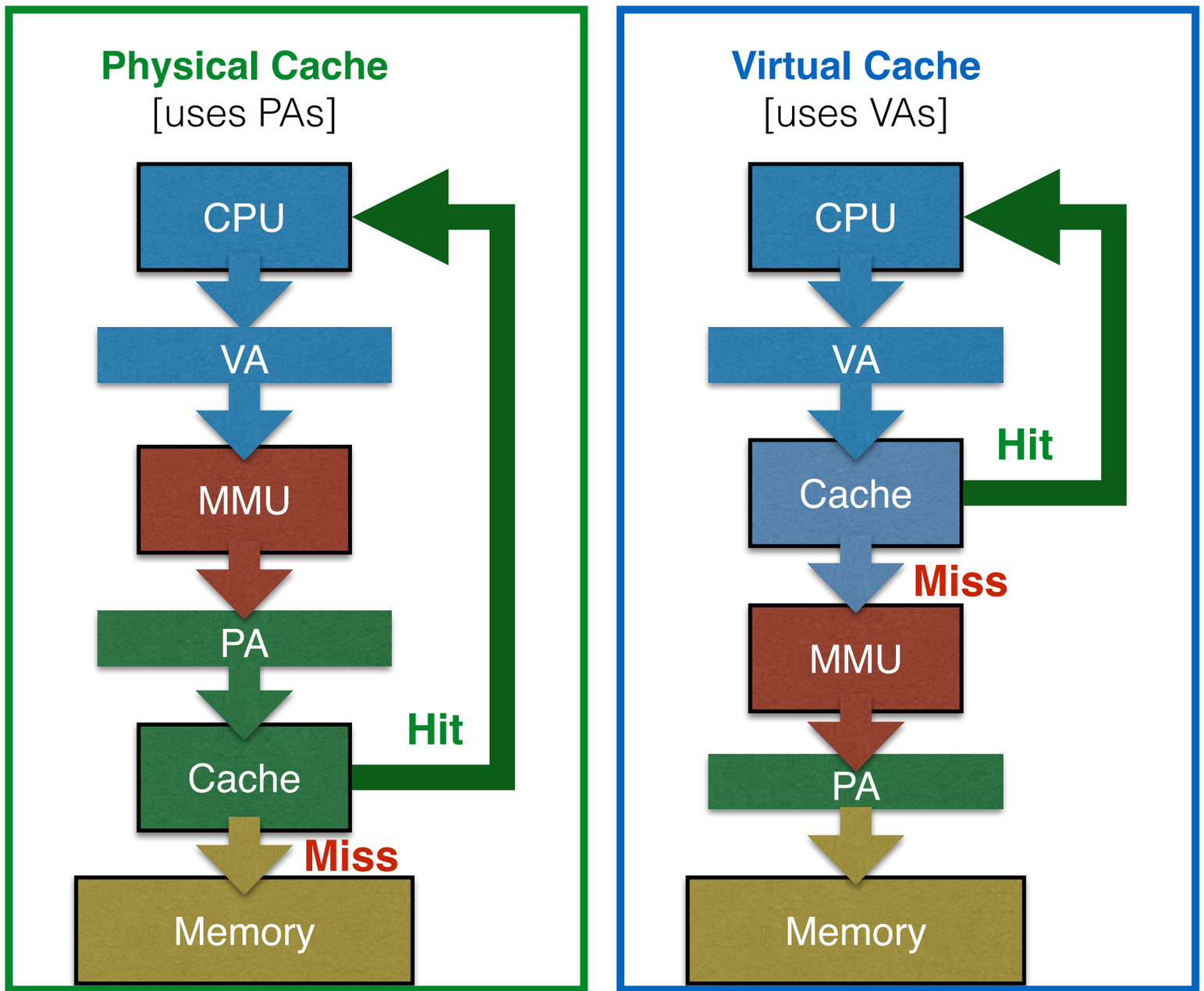
Virtual or Physical Cache?

- Is the cache using VAs or PAs? Let's look at what the two simple alternatives would look like…



Virtual or Physical Cache?

- Is the cache using VAs or PAs? Let's look at what the two simple alternatives would look like…

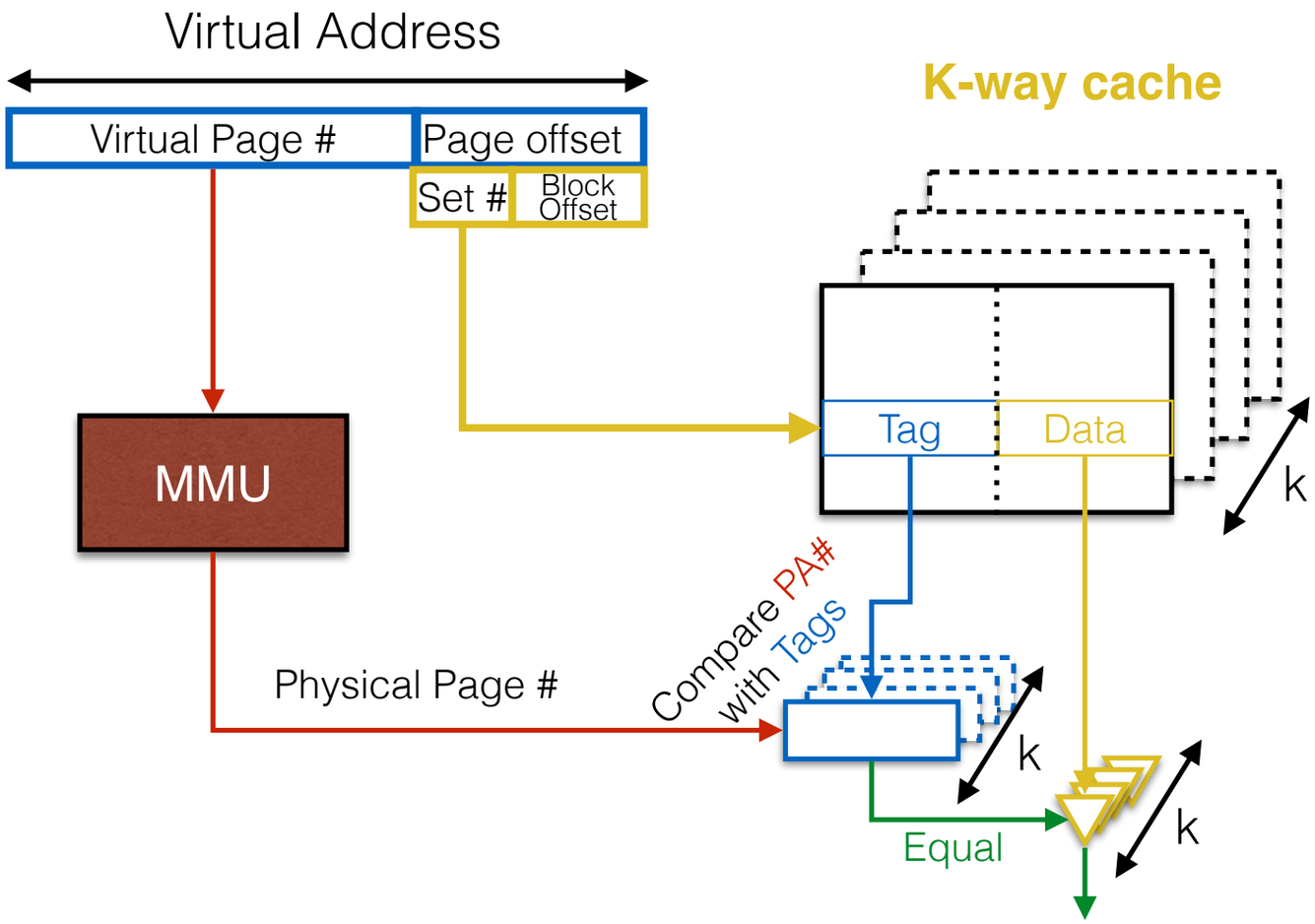


- Physical cache:
  Slow: VA to PA translation needed for each cache access
- Virtual cache:
  Fast: Translation needed only on cache misses
- But How to handle context switch?

Virtually-Indexed Physically-Tagged Cache

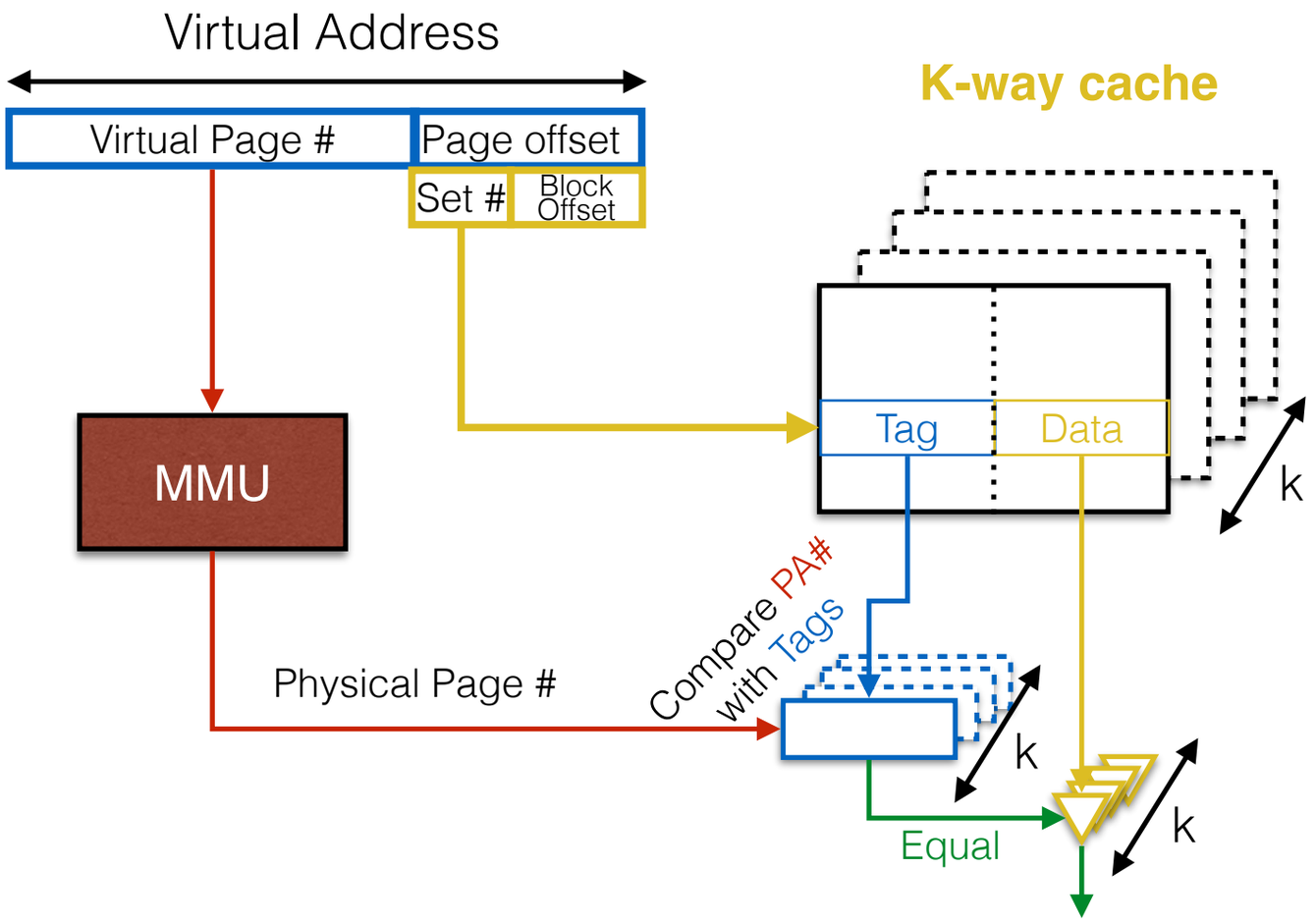
- Is it possible to get the best of both worlds – the speed of virtual cache, with the memory protection of the physical cache?
- The idea behind Virtually-Indexed Physically Tagged (VIPT) cache:
- Look up in the cache using a virtual address, but check for a hit with the physical address.
- In VIPT, cache and MMU are arranged so that they can be accessed concurrently

VIPT Cache



- The cache look-up uses only the page offset (not altered by MMU).
- For a K-way cache, you need to perform k comparisons!

VIPT Cache



- The cache look-up uses only the page offset (not altered by MMU).
- For a K-way cache, you need to perform k comparisons!

- VIPT limits the cache size! Cache size = K x PAGESIZE.
- Most L1 caches today are VIPT (and hence their size…).

Summary

- We have studied what caches are, why they were introduced and how they work.
- We have discussed organisation and operation of caches and ways to measure their performances.
- We have discussed miss penalty, miss rate, and reviewed method to optimise those.
- Finally we have described the interaction between virtual and physical memory and how the cache comes into the picture.