Cache Memory

Gap between the memory and microprocessor speed

- 18 months to double the microprocessor performance
- Several years to double the memory performance (speed/size)

- Memory access time - describes the speed of data transfer between memory and microprocessor
- Memory cycle time - describes how often the memory access can be repeated

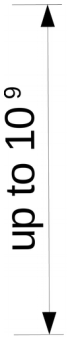
- SRAM – bistable flip-flop or latch, no need to refresh, short access time, more board space, low retain power/ high write power, more heat dissipation, high cost
- DRAM - charge in capacitor, need to refresh, long access time little board space, low power heat, low cost-per-size

Memory - the performance bottleneck

Solutions:

1) Memory fast enough (SRAM) to respond to every memory access request
2) Slow memory system (DRAM) with transfer improvements: wide buses and serial accesses
3) Combination of fast and slow memory systems, arranged so that the fast one is used more often then the slow one

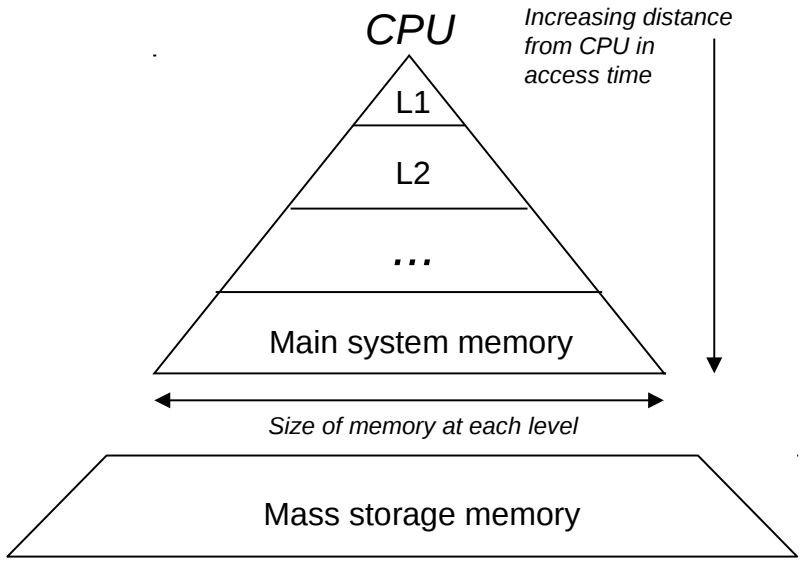
Register	–	< 0.5ns
Cache L1 on-chip	–	0.5ns
Cache L2 on-chip	–	2ns
Cache L3 off-chip	–	10ns
Memory DRAM	–	30-70ns
Flash SSD	–	µs
Magnetic HDD	–	ms



High performance memory system

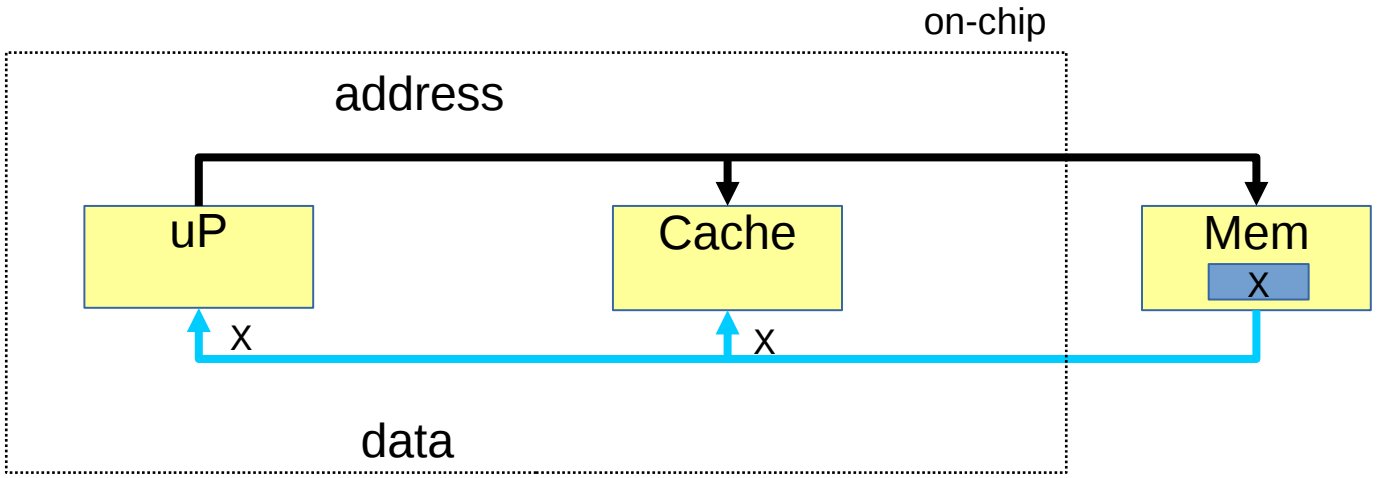
Hierarchical organization:

- Upper level is faster
- Lower level is bigger
- Upper level is subset of lower level
- *Program performance will strongly depend on code structure of program and size of data structures*

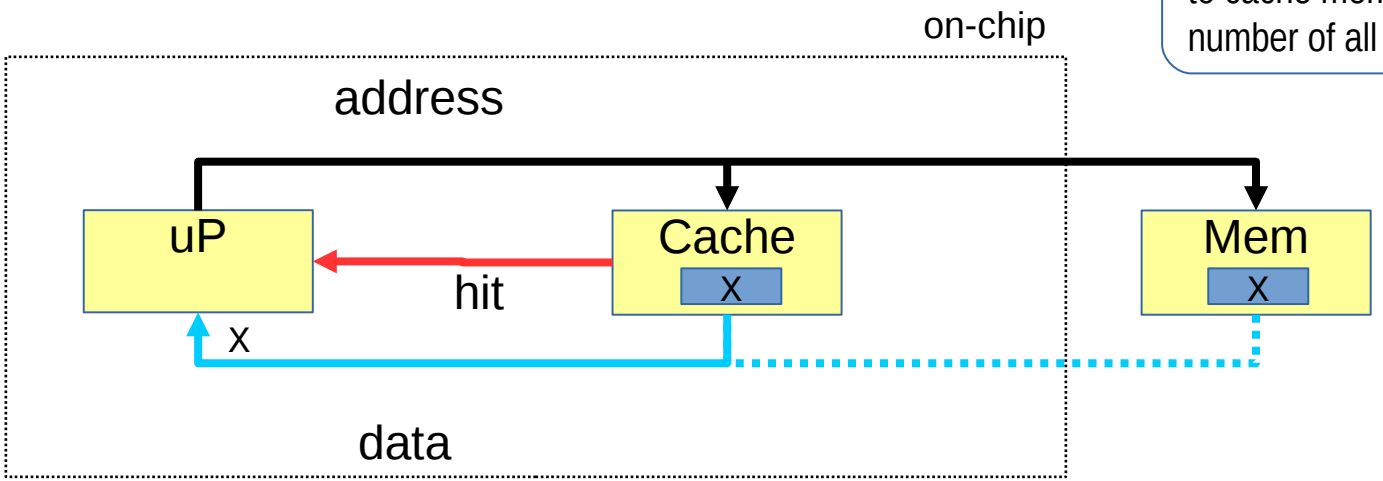


Cache principle – Reading from memory

Data not found in cache → transfer from memory (slow) to both µP and Cache



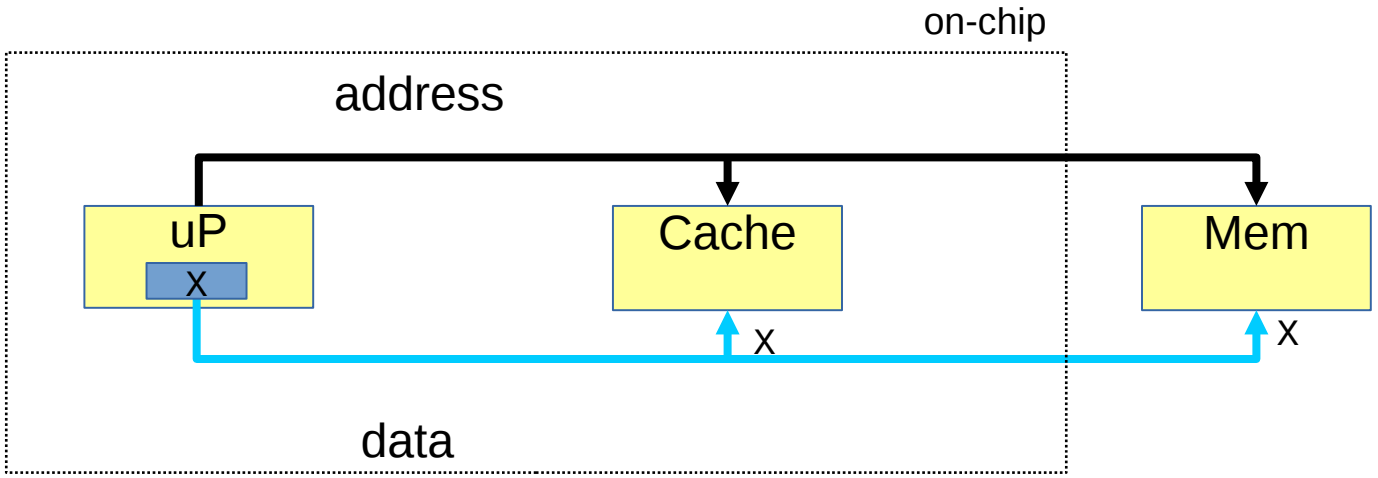
Data found in cache (hit) → transfer from Cache (fast) to µP



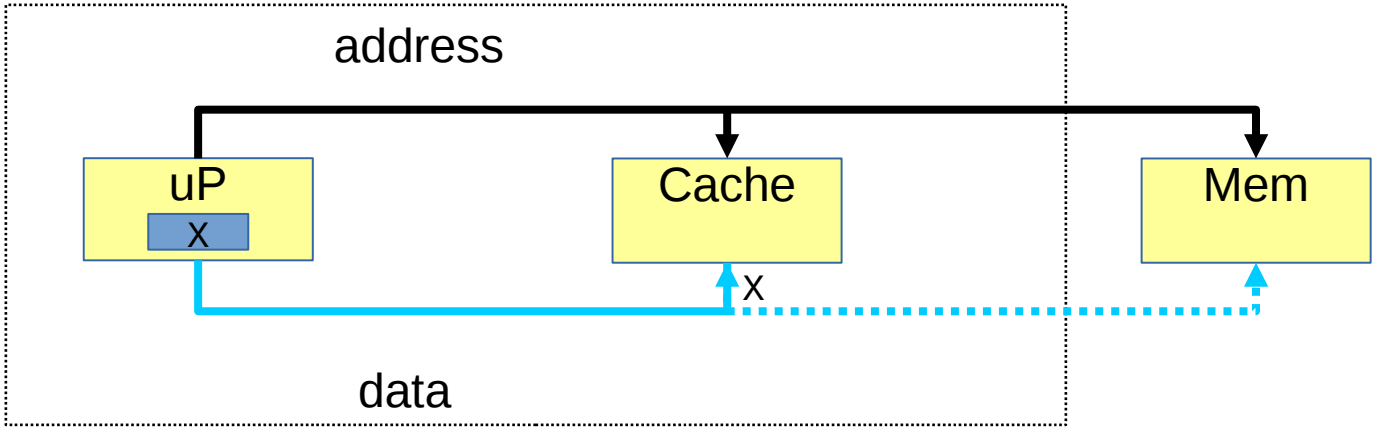
Hit rate - fraction of accesses to cache memory in the total number of all memory accesses

Cache principle – Writing to memory

Write-through → transfer to both: memory (slow) and Cache



Write-back → transfer to Cache only (fast),
the memory will be updated when this cache location is claimed by other data



Principles of locality

Temporal locality (locality in time)

if an item was referenced, it will be referenced again soon

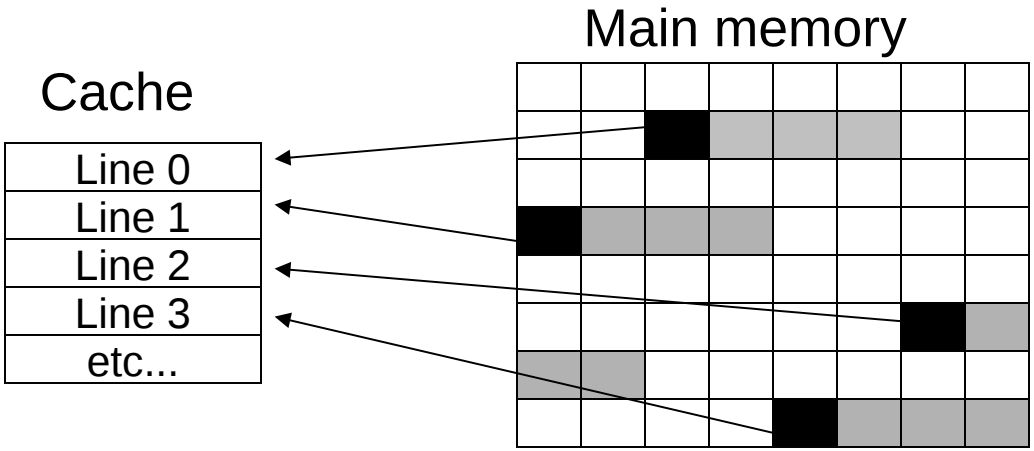
(e.g. cyclical execution in loops)

Spatial locality (locality in space)

if an item was referenced, items close to it will be referenced too

(the very nature of every program - serial stream of instructions)

Cache Organization



✓The principle of locality is valid either for instructions or for data, but there is no locality relation between demand for the both.

✓It is highly recommended to have two independent caches (Harvard Memory Architecture)

Cache and data structures access

Hit rate may depend on the way of accessing the data from memory,
unit-stride access will be preferred for maximal hit-rate

```
for (i = 1; i < 100000; i++)
    sum = sum + A(i);
```

Unit-stride loop

```
for (i = 1; i < 100000; i += 8)
    sum = sum + A(i);
```

Non unit-stride loop

```
double A[row][col];

for (i = 0; i < row; i++)
    for (j = 0; j < col; j++)
        sum = sum + A(i,j);
```

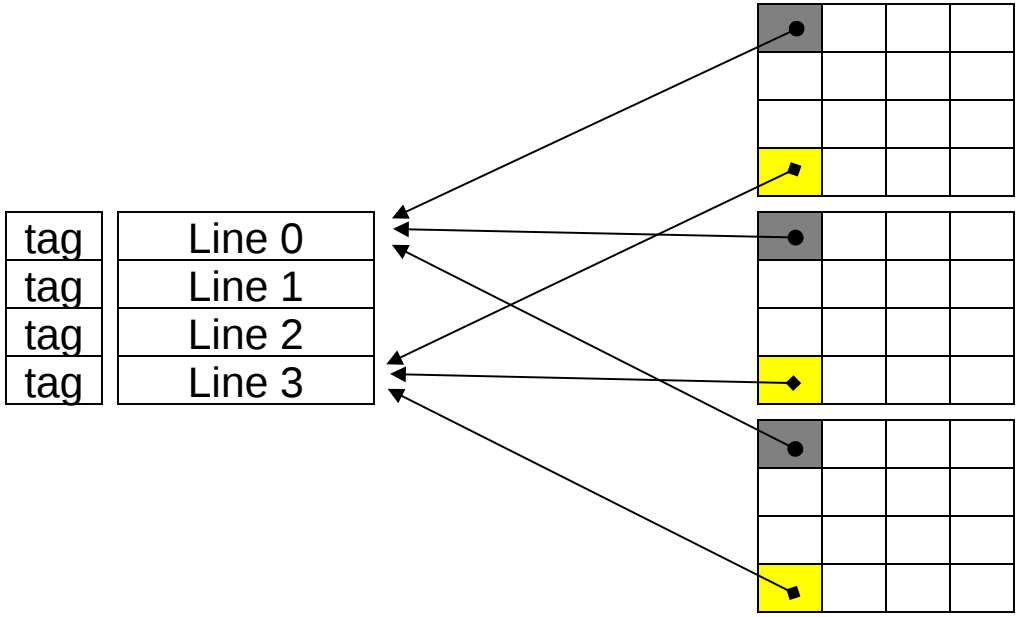
Unit-stride loop

```
double A[row][col];

for (i = 0; i < col; i++)
    for (j = 0; j < raw; j++)
        sum = sum + A(i,j);
```

Non unit-stride loop

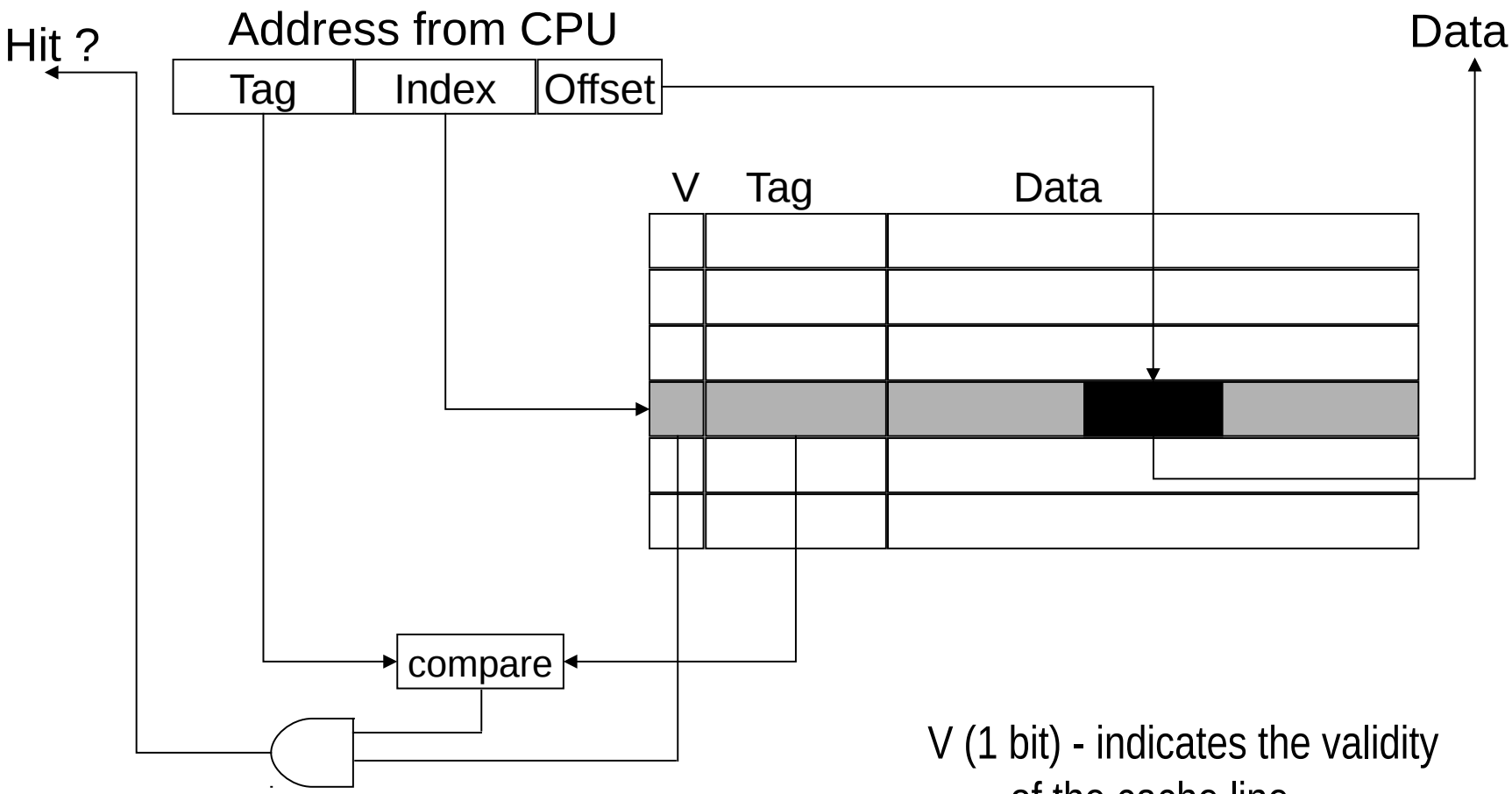
Direct-Mapped Cache



memory address → | tag | index | offset |

Tag – (most sign. part of address) identifies the memory block the data comes from
Index – (mid. part of address) identifies line numbers within cache (and block)
Offset - (least sign. part of address) identifies the byte (word) within a cache line

Direct-Mapped Cache – hit signal



V (1 bit) - indicates the validity of the cache line

Cache thrashing

When alternating memory references point to the same cache line, the cache entry is frequently replaced, lowering the performance.

Direct-Mapped Cache offers no benefits in case of cache thrashing.

Example: 4KB direct-mapped cache

```
float A[1024], B[1024];
…
for (i = 0; i < 1024; i++)
    A(i) = A(i) * B(i);
```

The arrays' size coincide with the cache size. The same elements from A and B will occupy exactly the same cache lines, causing repeated cache misses

Set-Associative Cache

The key to performance increase (and trashing reduction) is the more flexible placement of memory blocks by combining several direct-mapped caches.

One-way set-associative (direct-mapped)

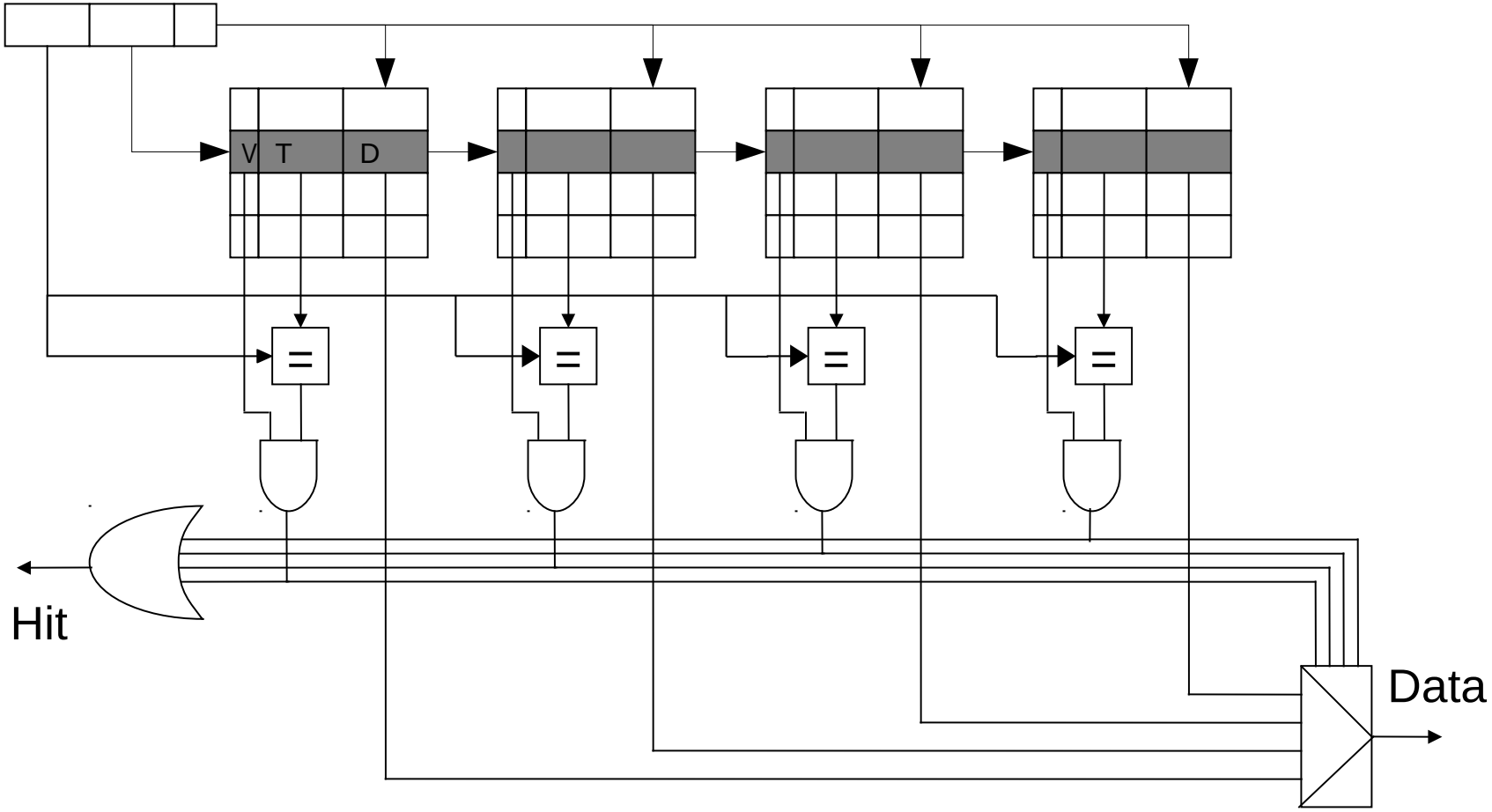
Block	Tag	Data
0		
1		
2		
3		

Two-way set-associative

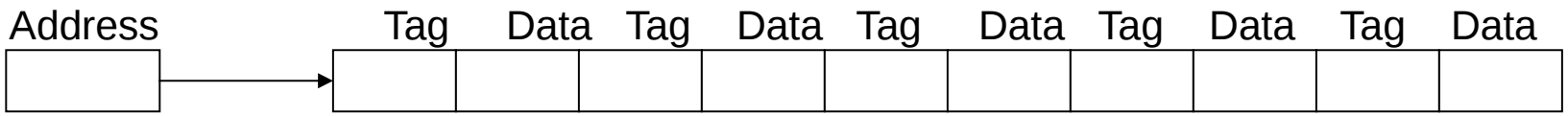
Block	Tag	Data	Tag	Data
0				
1				
2				
3				

The degree of associativity reduces the miss rate, at the cost of increase in the hit time and hardware complexity

Set-Associative Cache : four-way



Fully Associative Cache



- The memory block can be placed in any cache line
  - Slower access - complicated internal circuitry
  - Demand on board space - each cache entry has a comparator
  - Memory needed for tags increases with associativity
- Algorithm to choose which block to replace
  - LRU (Least Recently Used) - requires additional bits for each cache line, updated during each access
  - Random - candidates are selected randomly

Software managed caches

Idea: transfer the data to cache before the processor needs it, so that the cache-fill time will be hidden

Cache-fill time can be hidden and hopefully all memory references will operate at full cache speed.

- *Prefetching* - method of loading cache memory supported by some processors by implementing a new instruction.
- Prefetch instruction operates like any other instruction, except that processor doesn't have to wait for the result
- Compilers can generate *prefetch* instructions when detects data access using a fixed stride

```
for (i = 0; i < n; i +=8 )
{
    PREFETCH( A(i + 8) )
    for (j = 0; j < 8; j++)
        sum = sum + A(i+j);
}
```

Post-RISC effects on memory access

Ability of out-of-order and parallel execution gives the possibility to compensate for slow memory latency

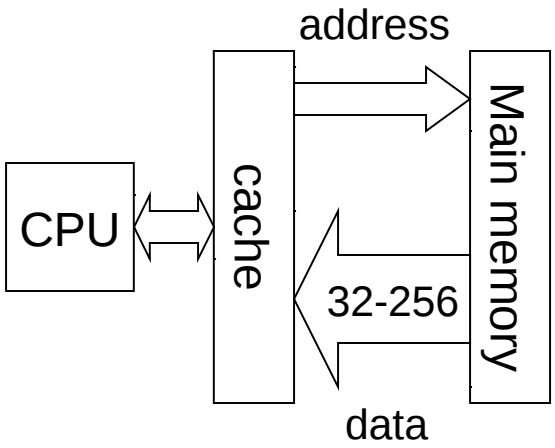
```
        LOADI       R6, 1000        set iterations
        LOADI       R5, 0           set index
LOOP    LOAD        R1, R2(R5)      load from memory
        INCR        R1
        STORE       R1, R3(R5)      save in memory
        INCR        R5
        COMPARE     R5, R6          check termination
        BLT         LOOP            branch if R5<R6
```

Several load/store instructions can be initiated without absolute stalling the program execution

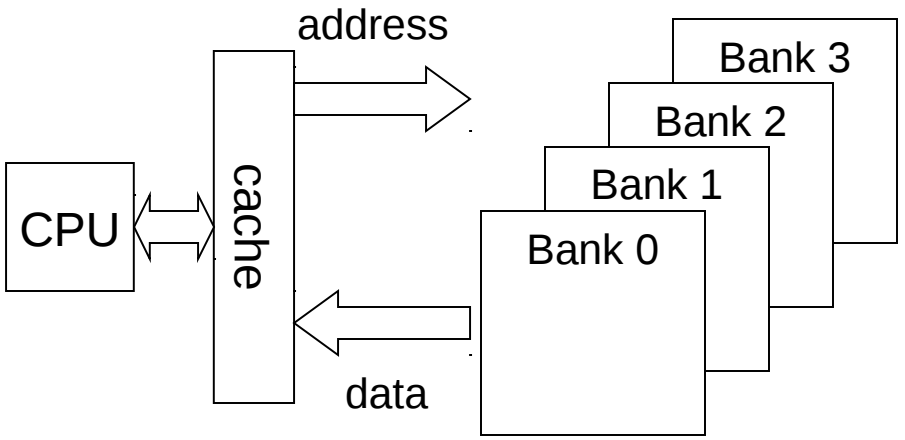
Improving memory performance - overview

Two main obstacles:

- Bandwidth - best possible steady-state transfer rate (usually when running a long unit-stride loop)
- Latency - the worst-case delay during single memory access



Wide memory systems
- high bandwidth



Interleaved memory systems
- lower latency