Computer Architecture & Real-Time Operating System

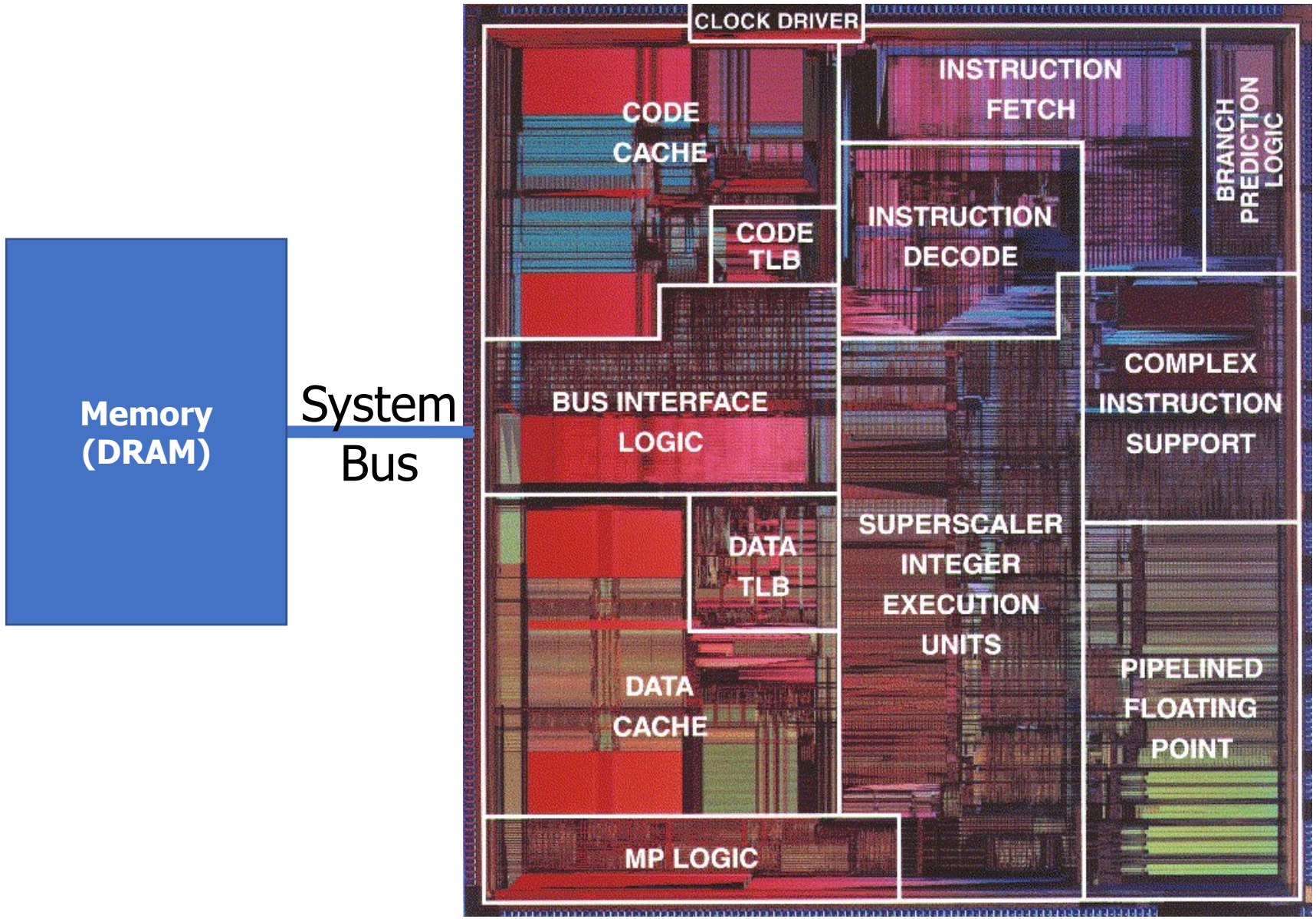
9. Memory Subsystem (2/2)

Prof. Jong-Chan Kim

Dept. Automobile and IT Convergence

KMU 국민대학교 KOOKMIN UNIVERSITY

What's covered so far and what's to be

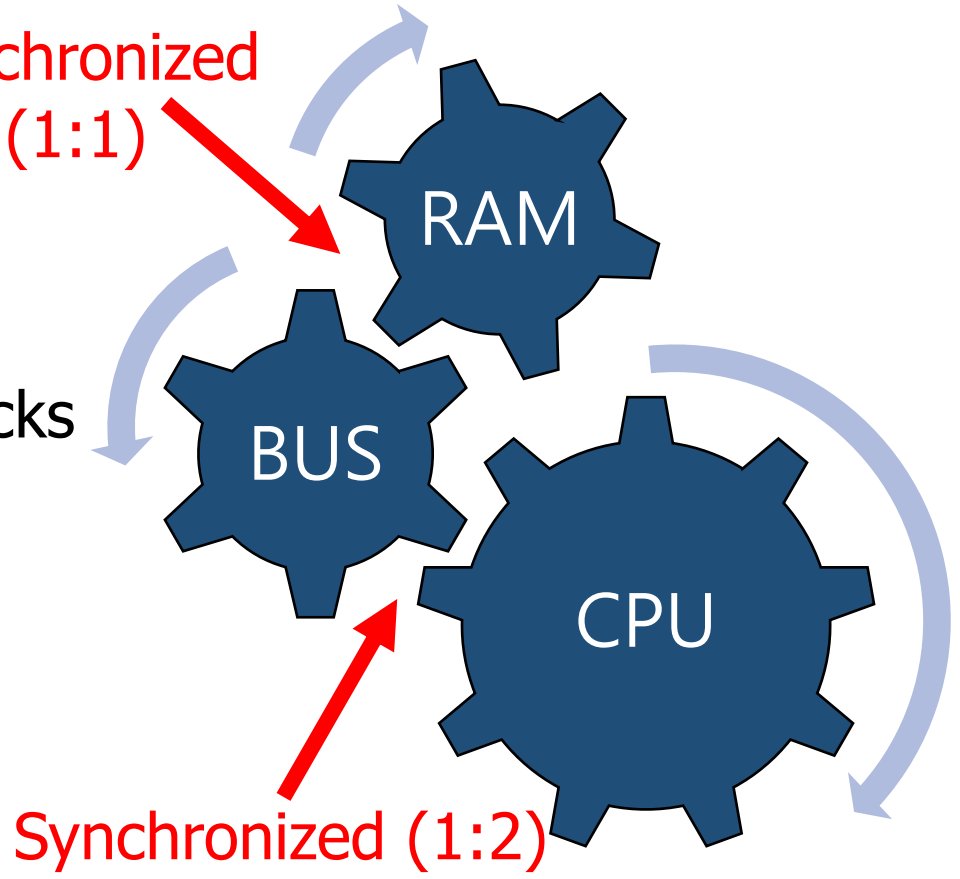


Timing between CPU and Memory (RAM)

- Asynchronous RAM
  - CPU and RAM run independently
  - No longer used
- Synchronous RAM
  - CPU and RAM run synchronized by system clocks
  - They know each other's timing

Synchronous Dynamic RAM

8GB DDR3 SDRAM running at 1600MHz



Speeds of CPU and RAM

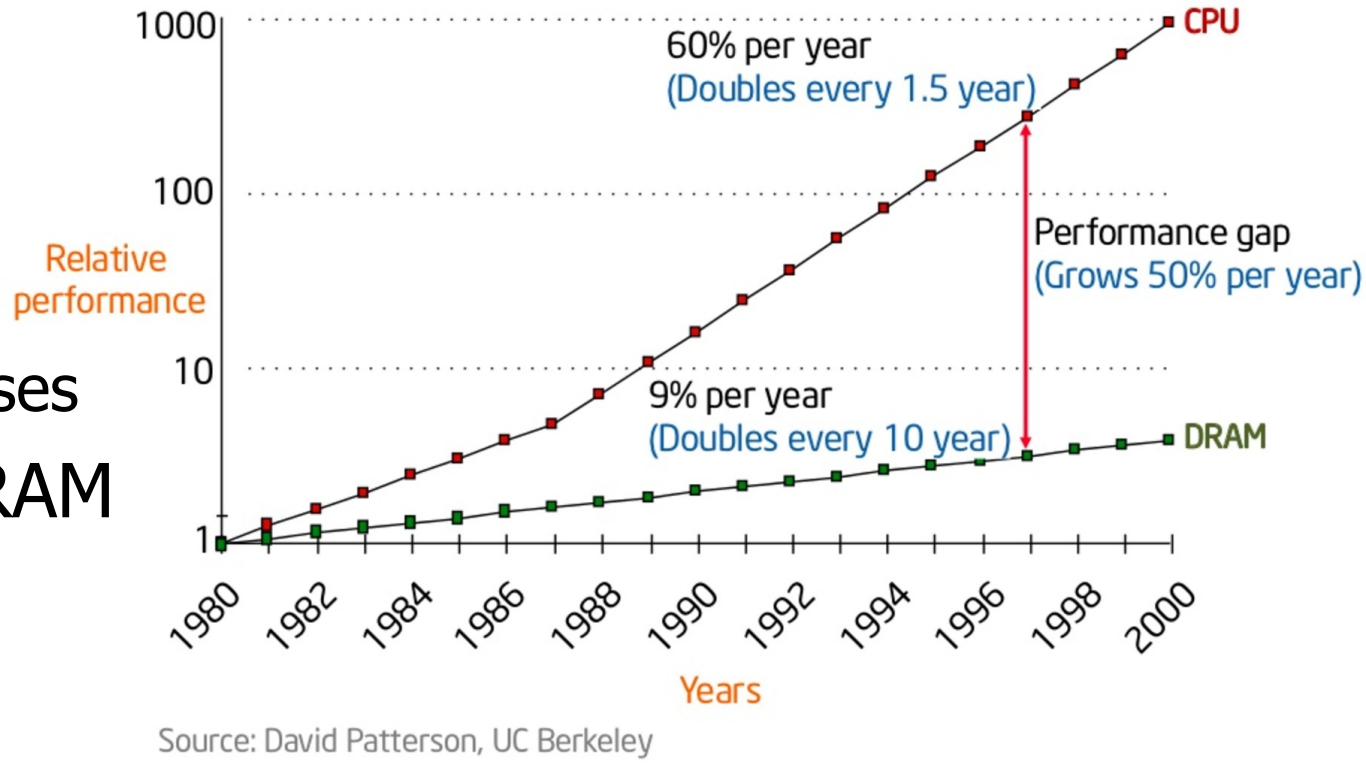
execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

It's about latency, not bandwidth

Source: http://norvig.com/21-days.html

CPU-RAM Performance Gap

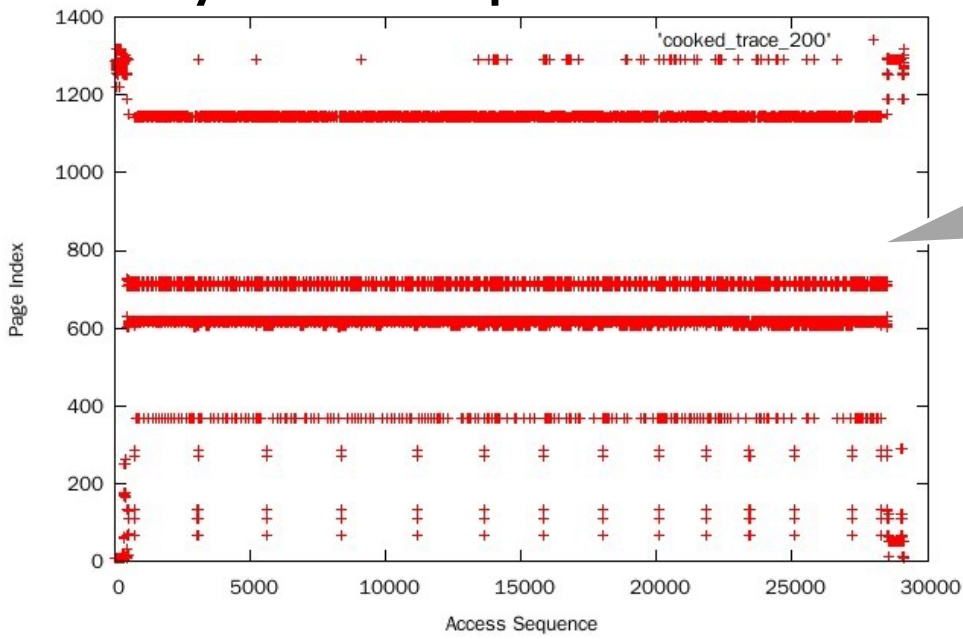
- RAM latency becomes a serious performance bottleneck
  - Instruction fetch
  - Load instruction
  - Store instruction
- CPI (Cycles Per Instruction)
  - Will be dominated by RAM accesses
- CPU executes at the speed of RAM
  - X100 slower



Source: David Patterson, UC Berkeley

Locality of Memory Accesses

- Typical program's memory access pattern



Working set: a small number of actually accessed memory locations during a given period

- Temporal Locality
  - Recently accessed locations tend to be accessed again in the near future (e.g., index variables in loops)
- Spatial Locality
  - If a location is accessed, its neighbors tend be accessed in the near future (e.g., array items and sequential instructions)

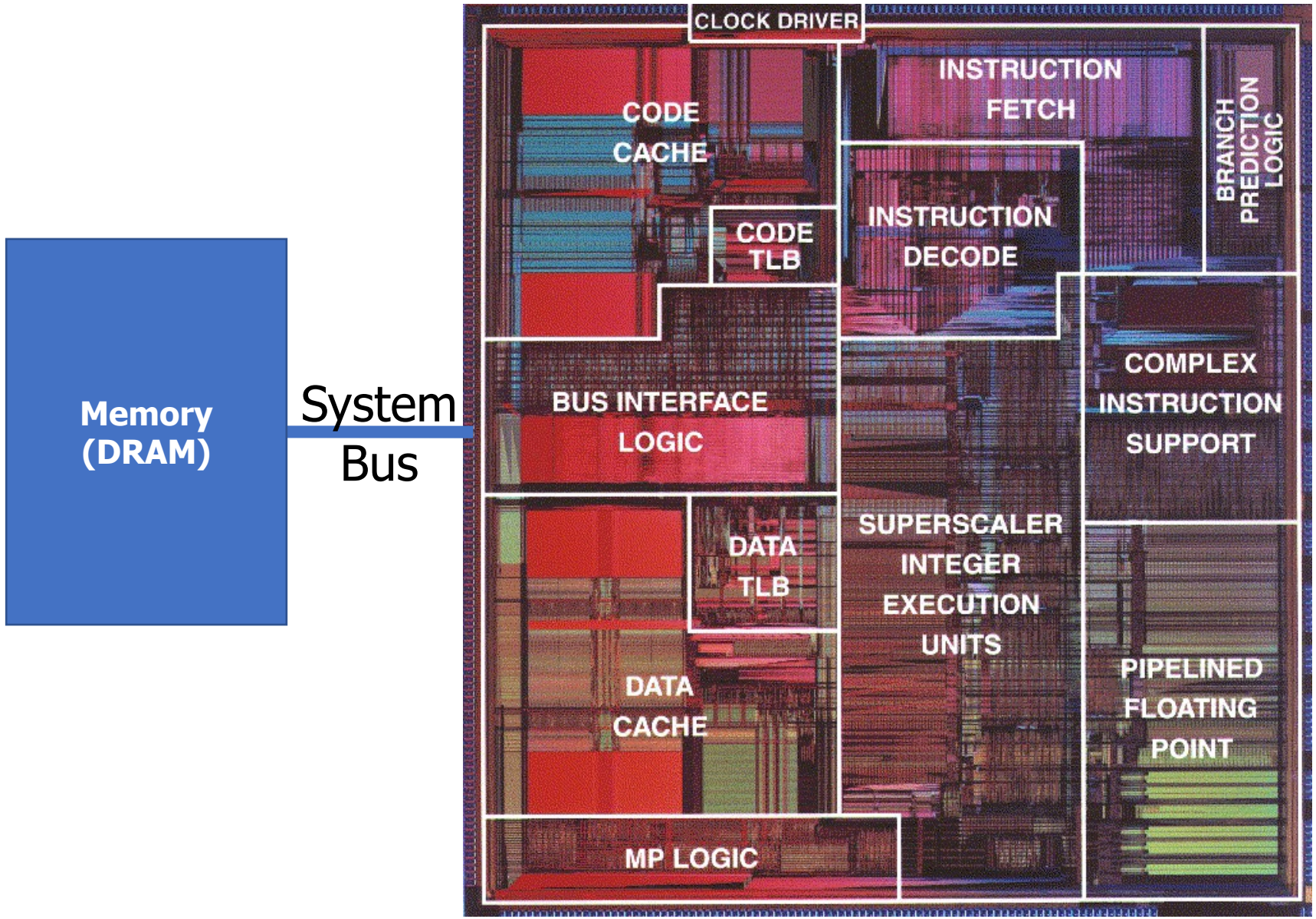
Quiz

- Which of the following codes has better locality?

```
int sum(int a [M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

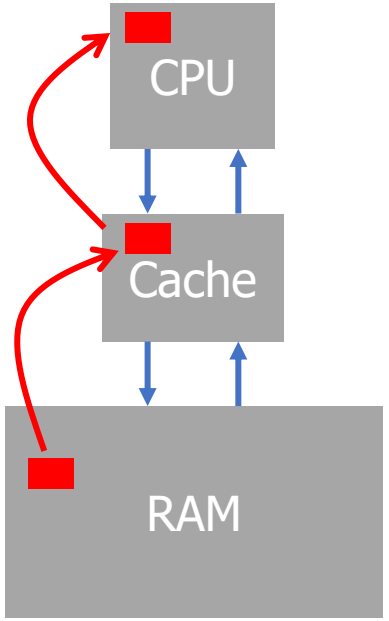
```
int sum(int a [M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Cache


Memory
(DRAM)
System
Bus
CLOCK DRIVER
CODE
CACHE
CODE
TLB
INSTRUCTION
FETCH
BRANCH
PREDICTION
LOGIC
INSTRUCTION
DECODE
BUS INTERFACE
LOGIC
COMPLEX
INSTRUCTION
SUPPORT
DATA
TLB
SUPERSCALER
INTEGER
EXECUTION
UNITS
DATA
CACHE
PIPELINED
FLOATING
POINT
MP LOGIC

Library Analogy

"Cache is like a desk, whereas RAM is like the whole library full of books"



Speeds of Cache

execute typical instruction	1/1,000,000,000 sec = 1 nanosec
fetch from L1 cache memory	0.5 nanosec
branch misprediction	5 nanosec
fetch from L2 cache memory	7 nanosec
Mutex lock/unlock	25 nanosec
fetch from main memory	100 nanosec
send 2K bytes over 1Gbps network	20,000 nanosec
read 1MB sequentially from memory	250,000 nanosec
fetch from new disk location (seek)	8,000,000 nanosec
read 1MB sequentially from disk	20,000,000 nanosec
send packet US to Europe and back	150 milliseconds = 150,000,000 nanosec

Source: http://norvig.com/21-days.html

Basic Cache Operations

- CPU tries to read a word in RAM
  - If it is in cache (Hit), just read it
  - If it is not in cache (Miss), copy it from RAM first and read it from cache

- Performance of cache memory
  - Hit Ratio: Probability of cache hit (>95% in modern computer systems)
  - Miss Ratio = 1 – Hit Ratio

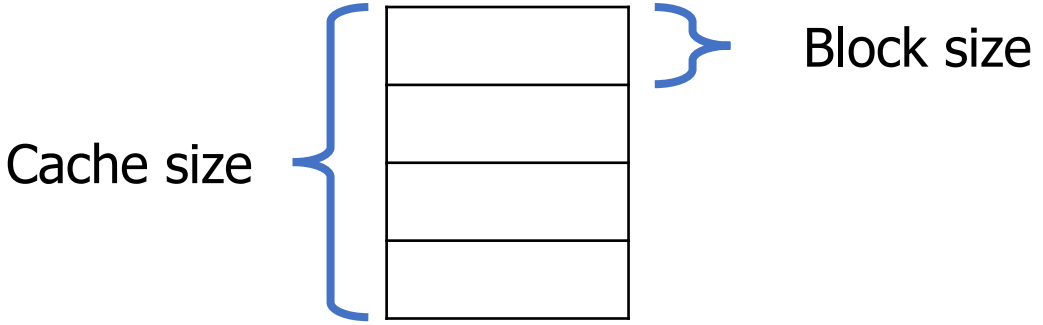
- How can we improve the cache performance?
  - A: Size and contents

Cost issue

Needs a good cache management algorithm

Cache Management Policies

- Cache size
- Block size: unit of cache management

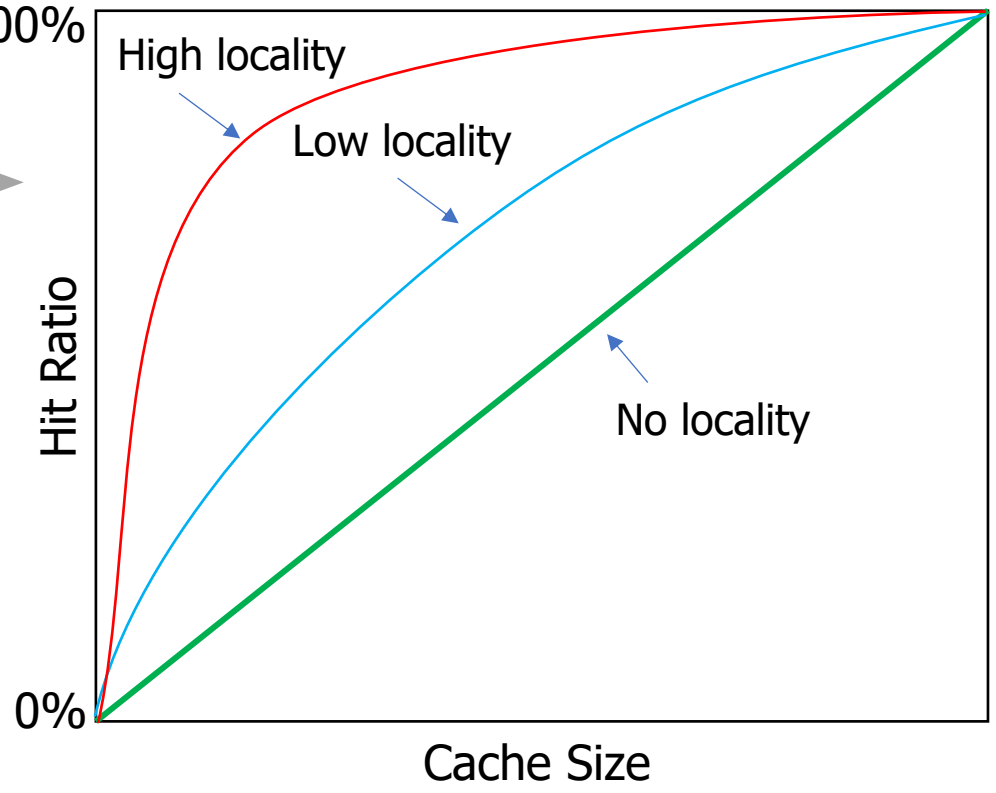


- Placement
  - Direct Mapped
  - Full Associative
  - Set Associative
- Replacement
  - FIFO (First In First Out)
  - LRU (Least Recently Used)
  - LFU (Least Frequently Used)
- Organization
  - Unified
  - Separated
- Write Policy
  - Write-back
  - Write-through

Cache Size & Block Size

- Cache size
  - Larger cache size ➔ higher hit ratio & more cost
  - Smaller cache size ➔ lower hit ratio & less cost

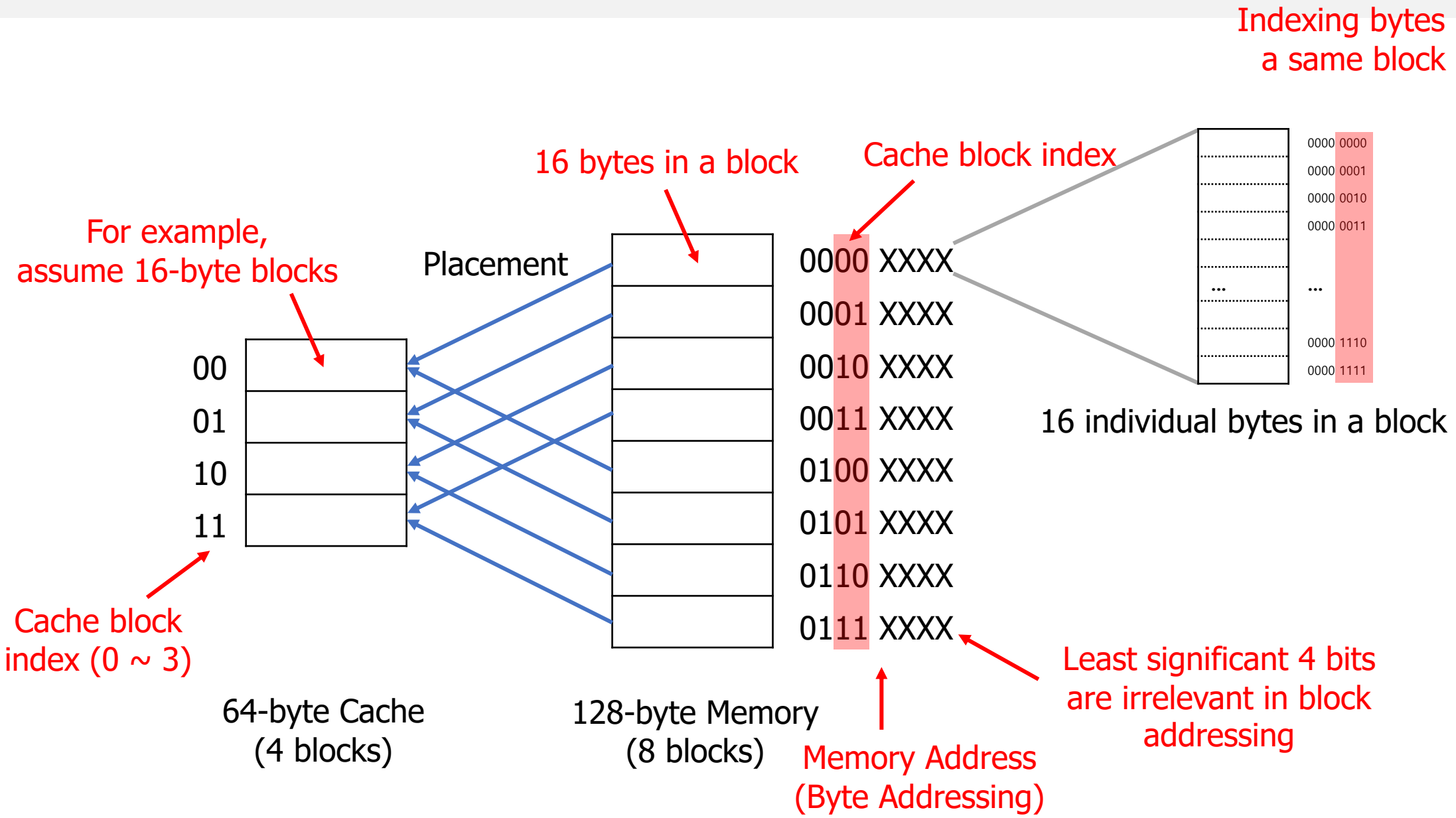
We have to decide an optimal point between cost and performance (i.e., hit ratio)



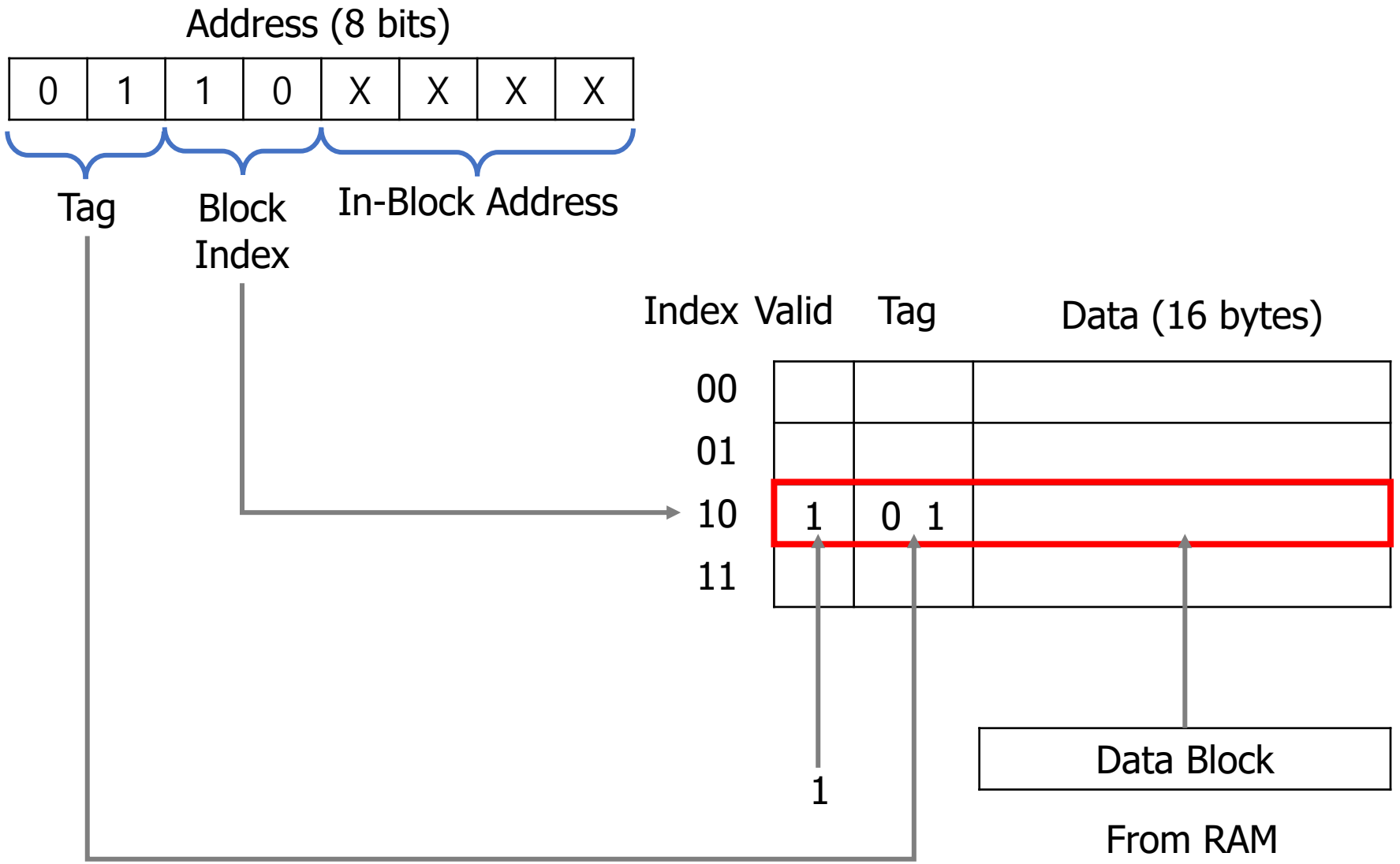
- Block size (or Line size)
  - Too large: useless data are cached together
  - To small: cannot utilize spatial locality

http://slideplayer.com/slide/10992960/

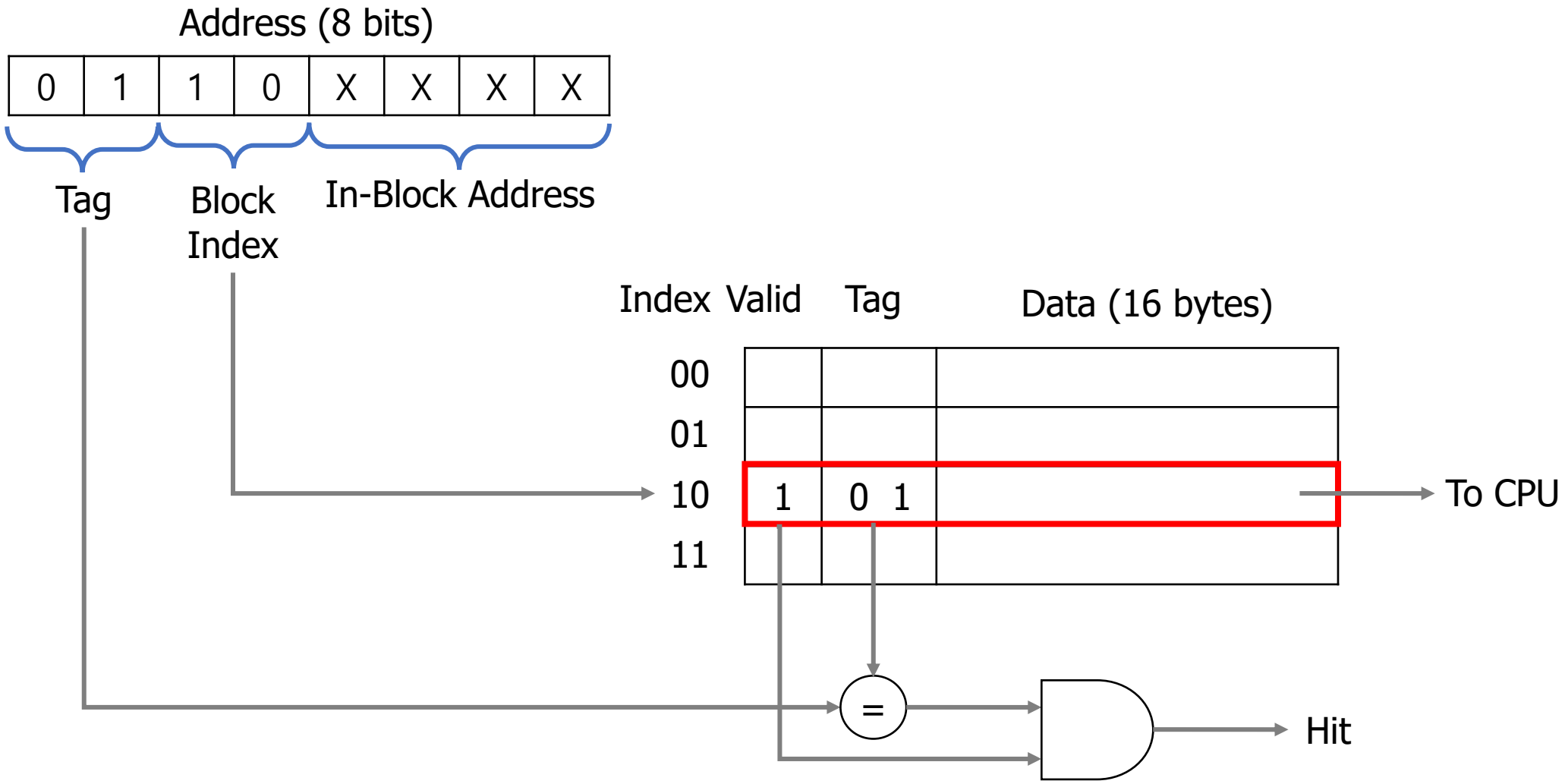
Basic Cache Organization



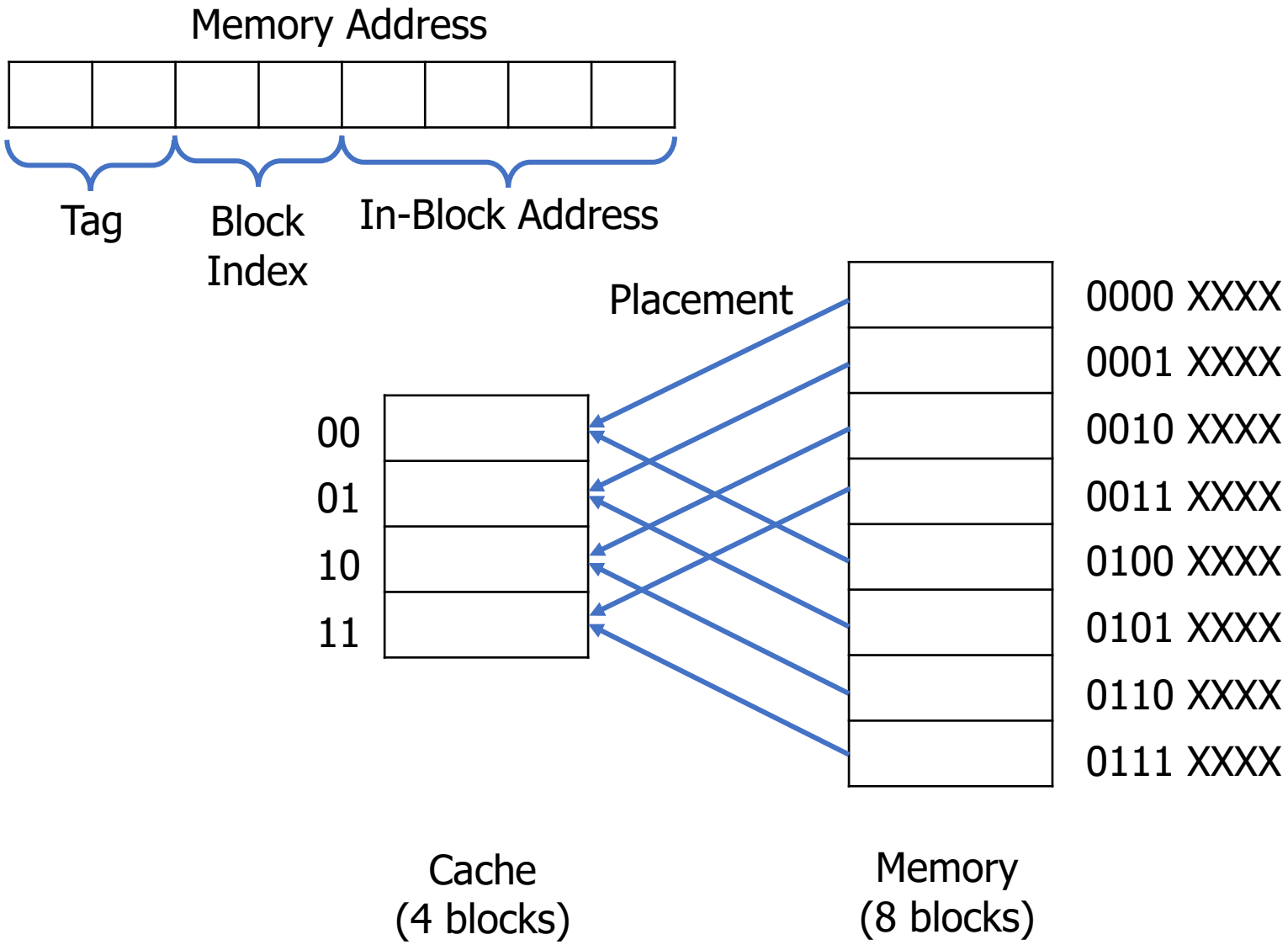
Cache Controller: Data Load to Cache



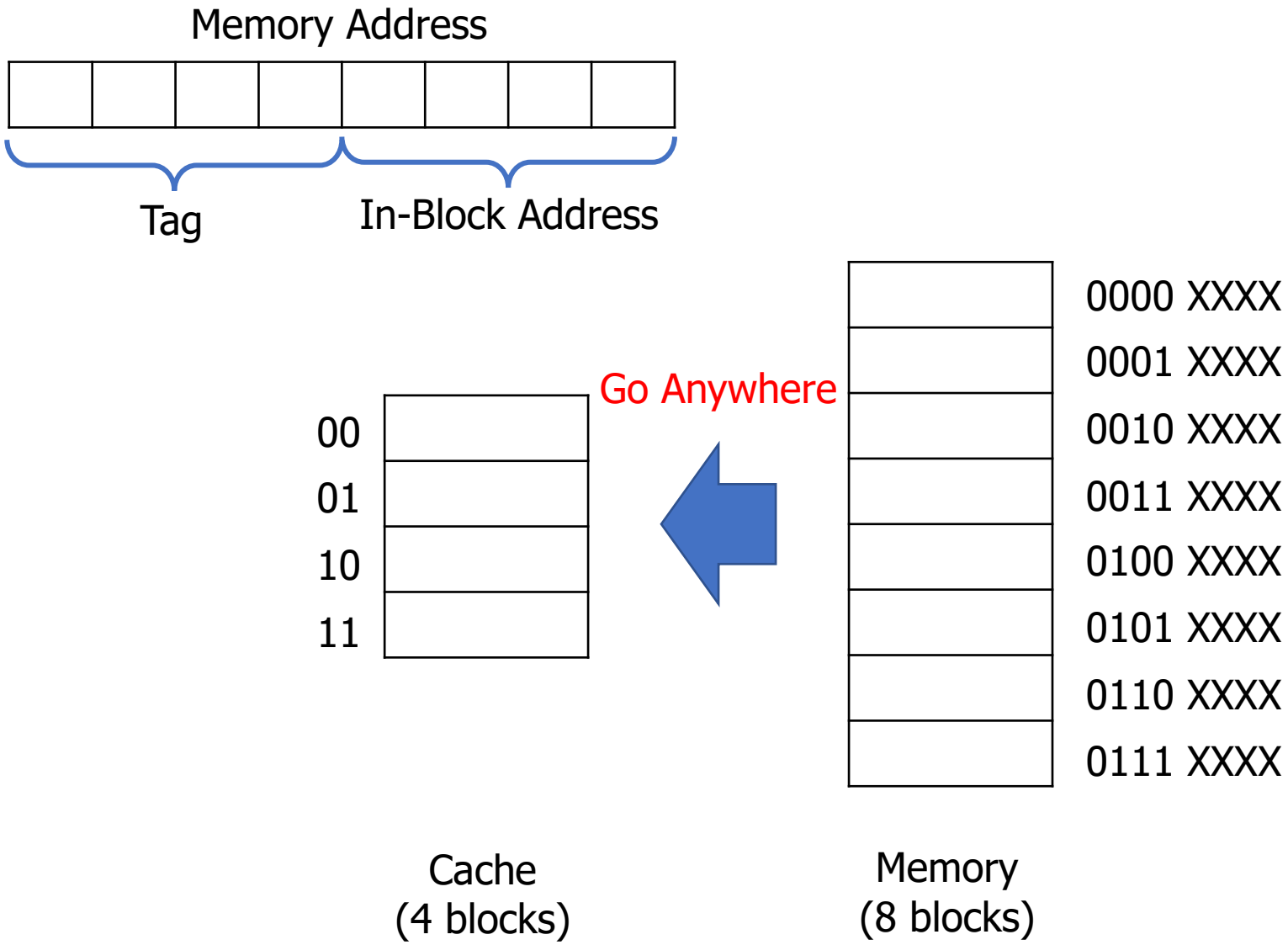
Cache Controller: Hit Check



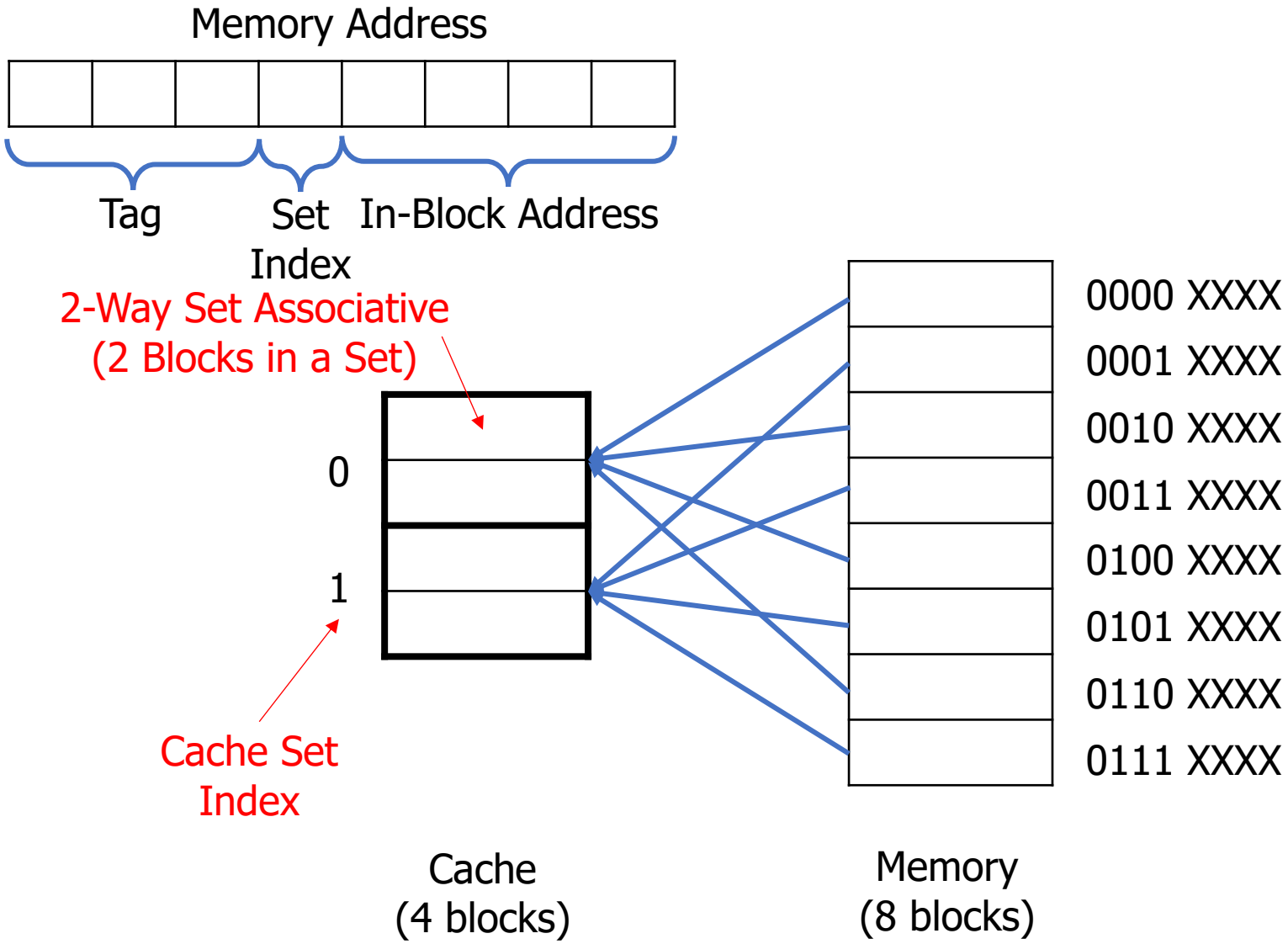
Placement: Direct Mapped



Placement: Fully Associative



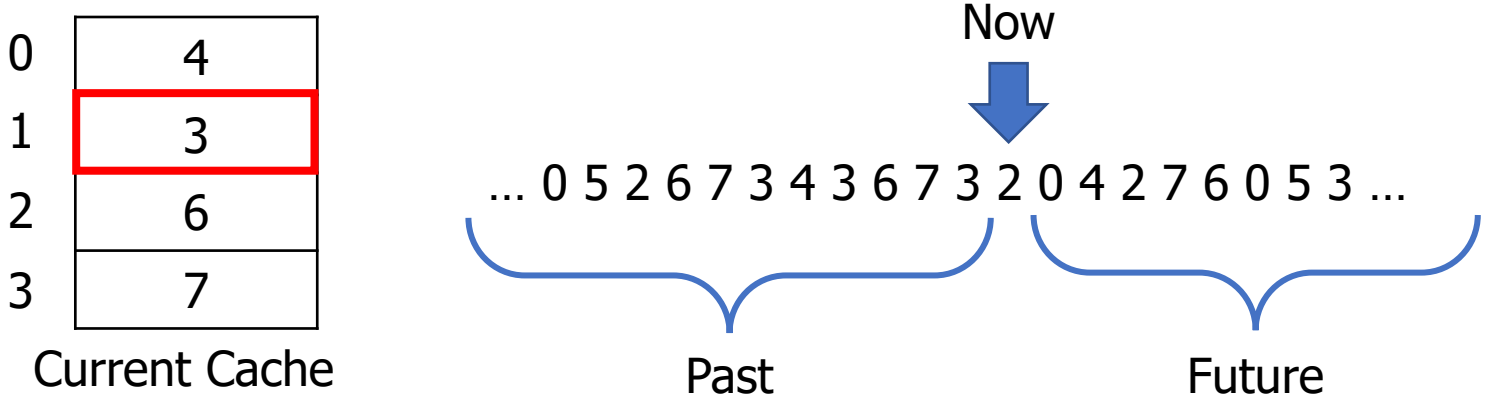
Placement: Set Associative



Replacement Policy

- Direct mapped cache
  - Victim block (to be evicted) is automatically decided by the placement policy

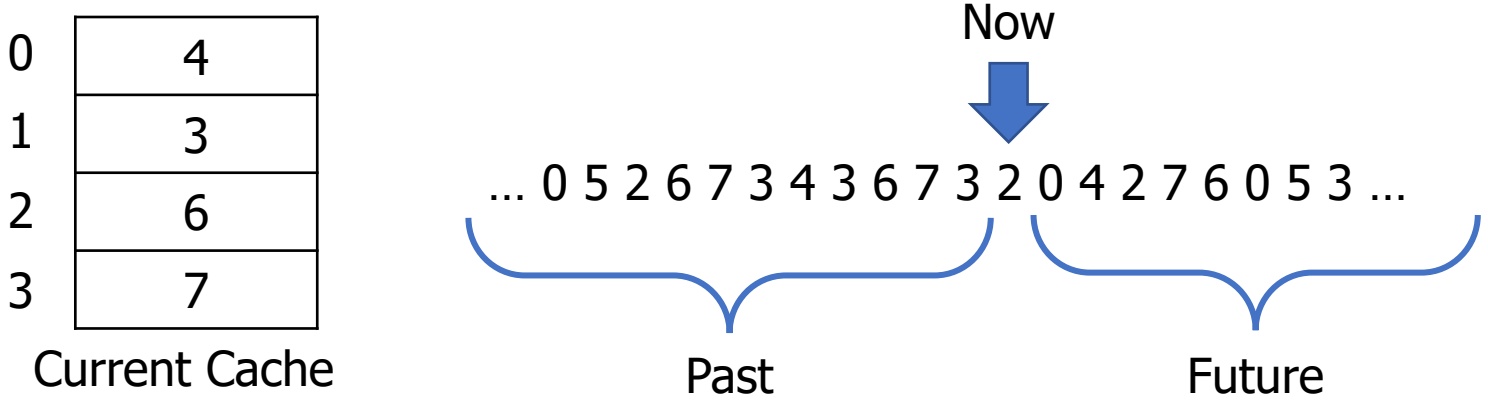
- Fully or set-associative cache
  - A new block has freedom to choose a victim
  - How can we find the most desirable victim?
    - In case we know the future, what is the **optimal replacement** algorithm?
    - In case we only know the past?



History-based Replacement Algorithms

- Random
- FIFO (First In First Out)
- LRU (Least Recently Used)
- LFU (Least Frequently Used)
- LRFU (Least Recently/Frequently Used)
- …

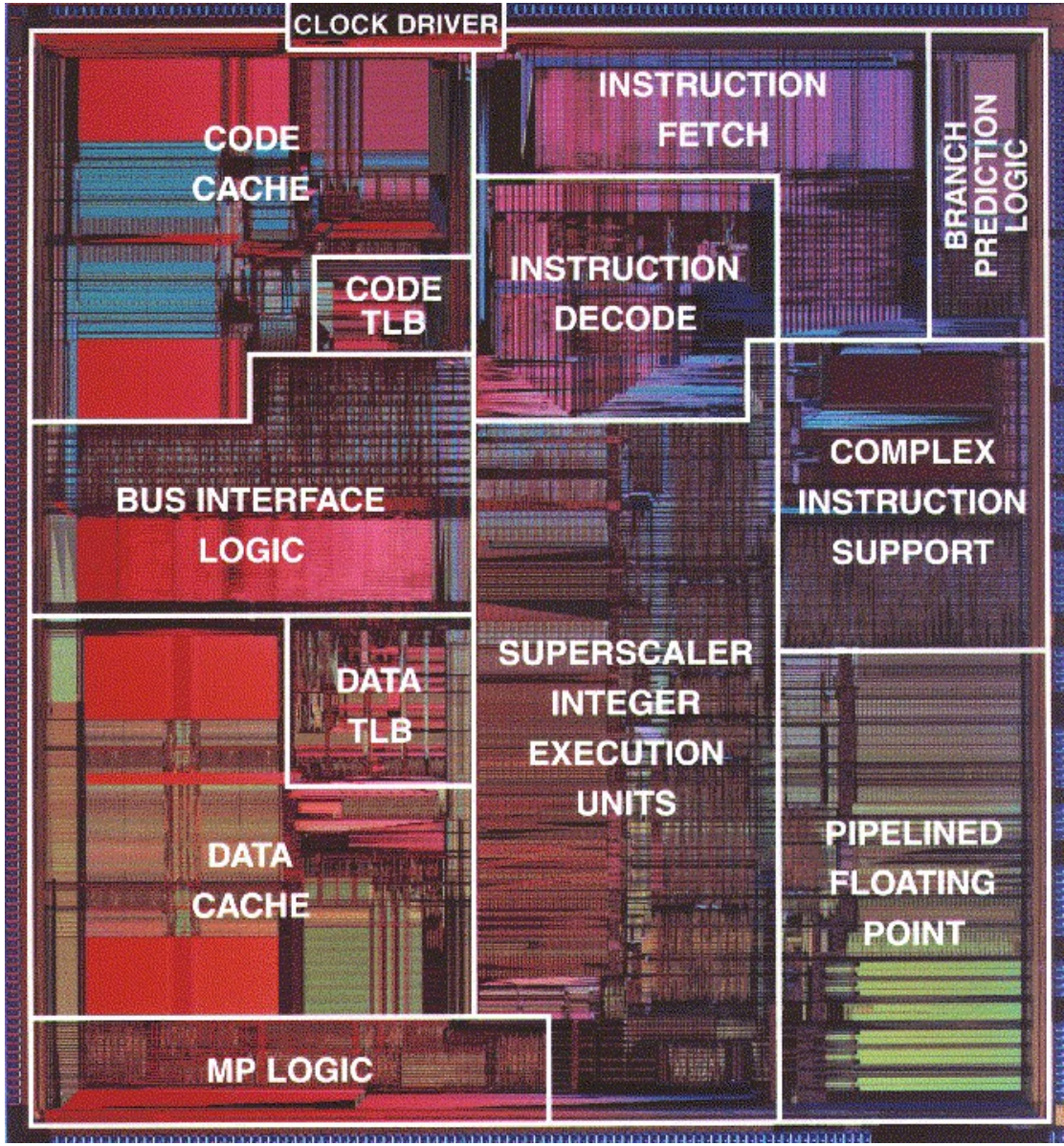
What if we apply LRU and LFU in the following example?



Unified or Separated Cache

- Unified architecture
  - Data and instructions in the same cache

- Separated architecture
  - Separated caches for data and instructions
  - More popular these days
  - Advantages
    - Good for pipelining
  - Disadvantages
    - Less flexible (separation is rigid)
    - More complex
  - Emulates the Harvard architecture
    - Separated paths to instruction memory and data memory



Write Policy

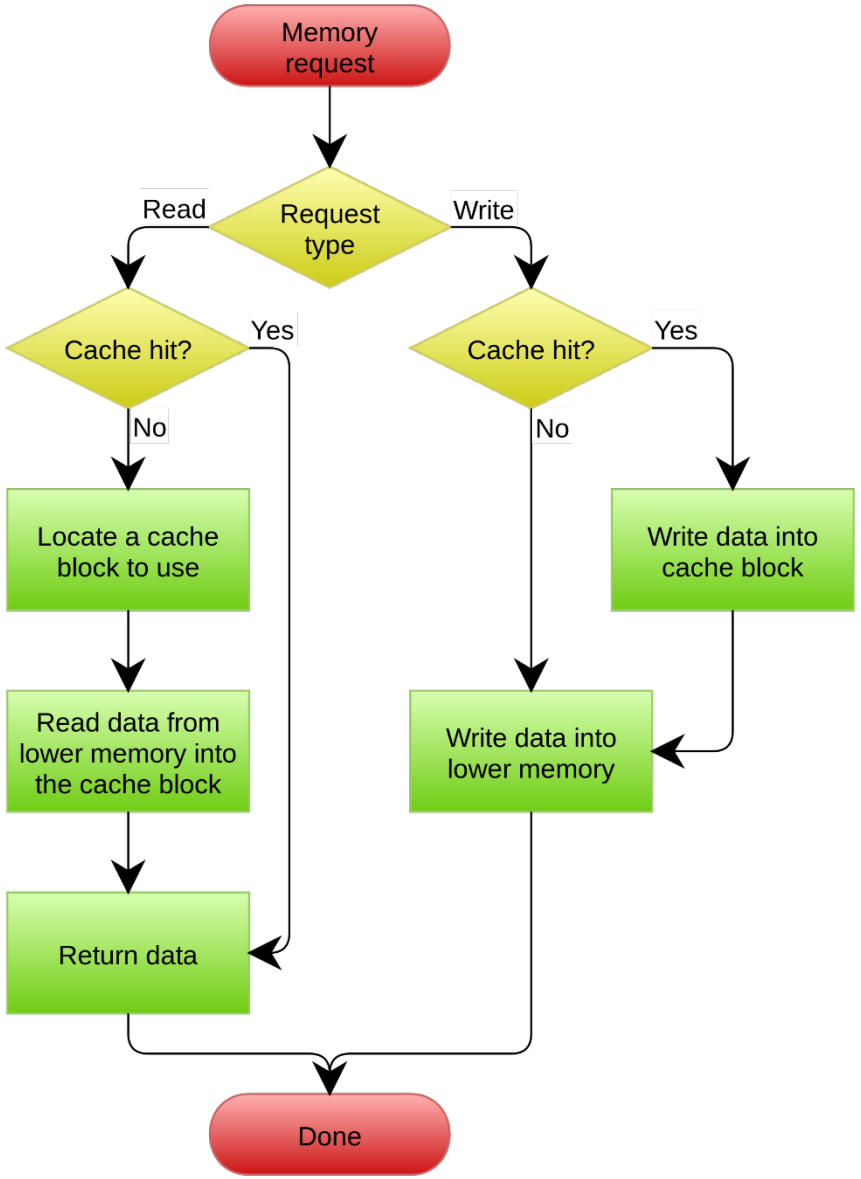
- When cache write hits
  - Write-through: update cache and memory at the same time
  - Write-back: update only cache and write memory later when its' evicted
- When cache write misses
  - Write allocate: allocate cache for the missed block
  - No-write allocate: just forward the write to RAM
- Write-back and write allocate go together
  - Hoping more cache writes in near future
- Write-through and no-write allocate go together
  - More cache writes incur the same number of memory writes
  - No need to maintain the cache

More complex, but higher expected hit ratio

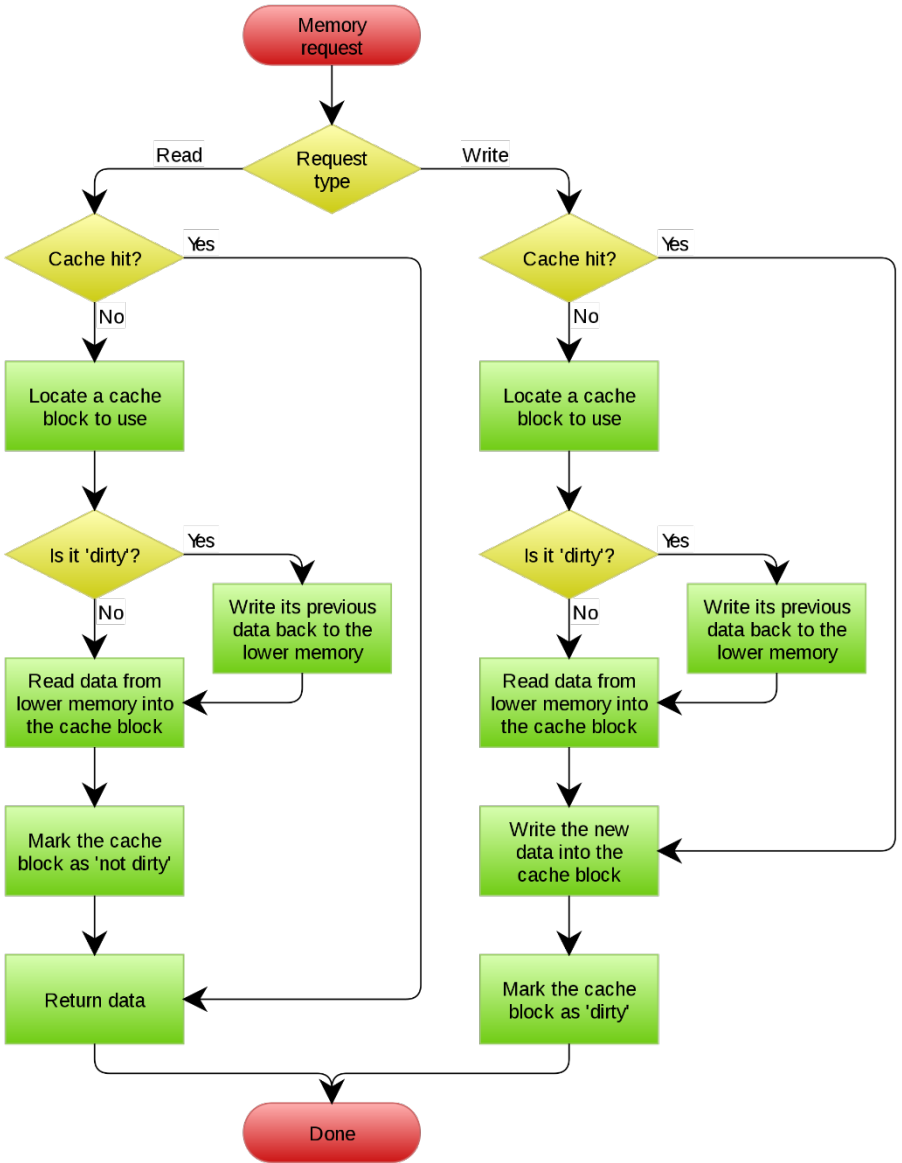
Simpler, but lower expected hit ratio

Write Strategies

Write-through & no-write allocate

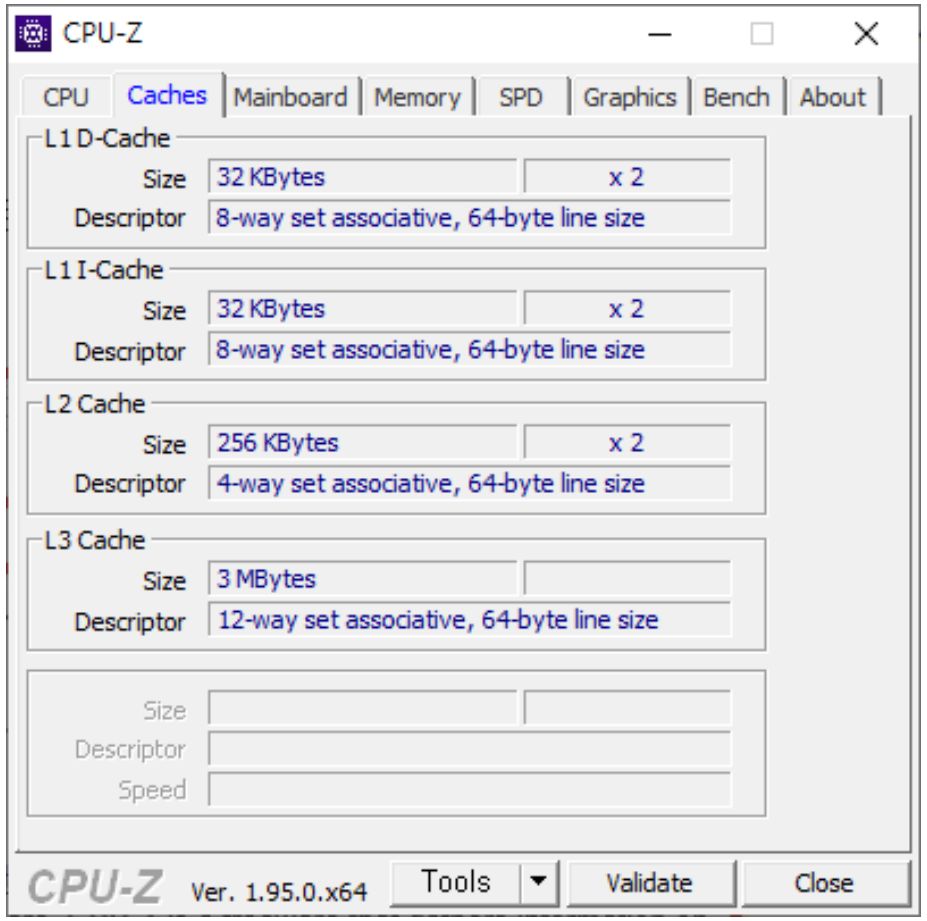


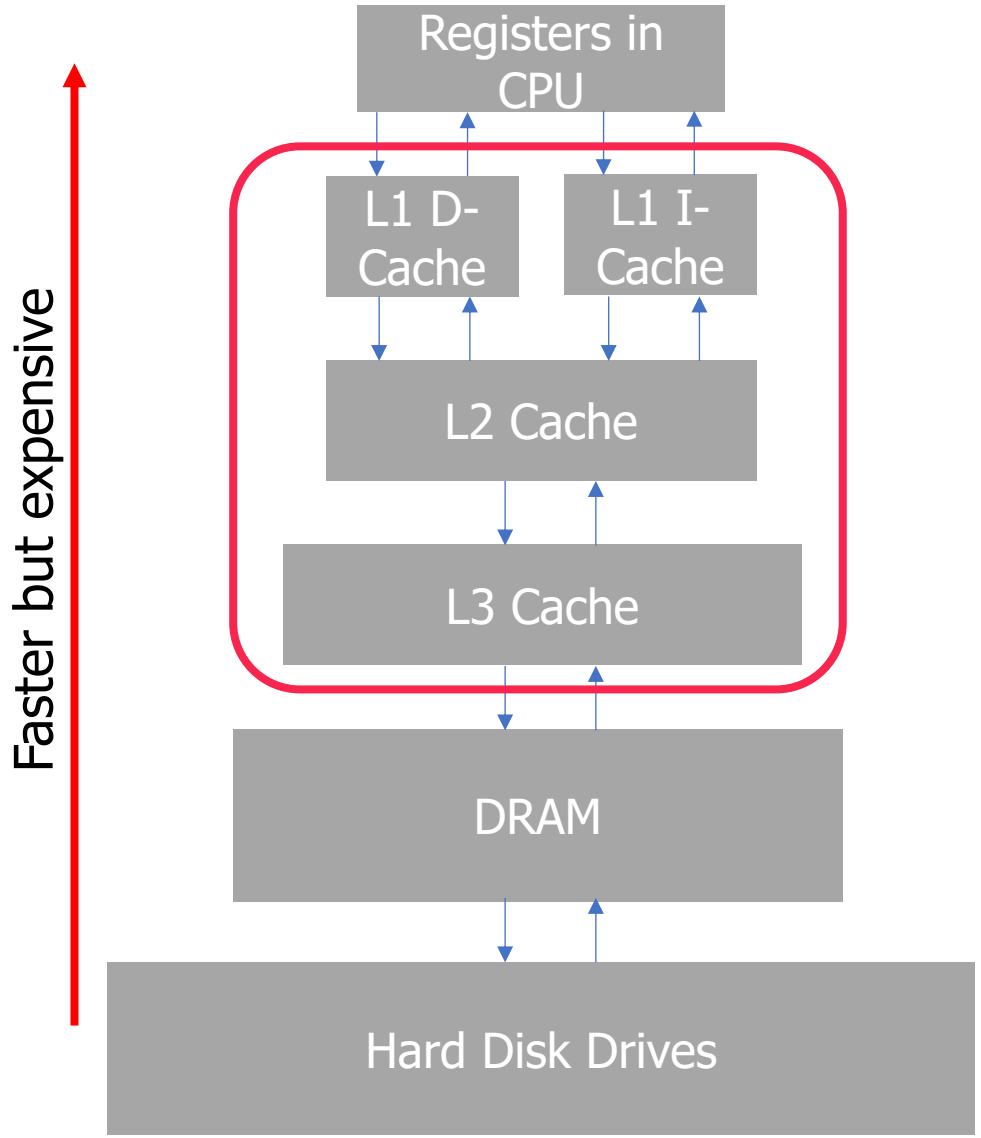
Write-back & write allocate



Source: Wikipedia

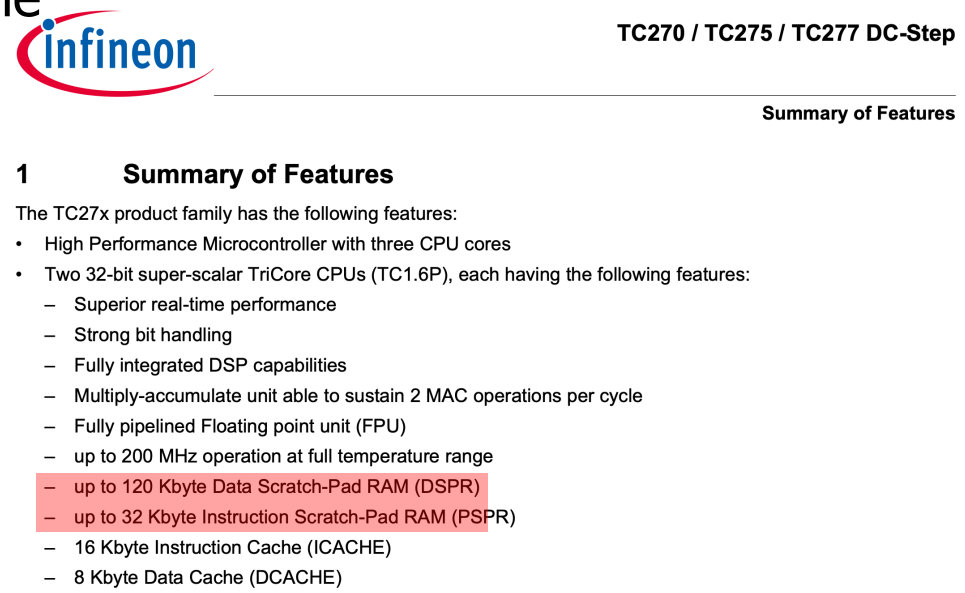
Multi-level Caches and Memory Hierarchy





Cache Transparency

- Cache operations are completely transparent to software
- There isn't any single line of code that manipulates cache in programs
- It is automatically done purely in CPU hardware
- Programmable on-chip SRAM is called "Scratchpad Memory" or "SPM"
  - Programmers can put some variables and functions to faster memory locations
  - Sometimes used in safety-critical embedded systems. Why?
    - To minimize execution time variations caused by cache



Infineon

TC270 / TC275 / TC277 DC-Step

Summary of Features

1 Summary of Features

The TC27x product family has the following features:

- High Performance Microcontroller with three CPU cores
- Two 32-bit super-scalar TriCore CPUs (TC1.6P), each having the following features:
  - Superior real-time performance
  - Strong bit handling
  - Fully integrated DSP capabilities
  - Multiply-accumulate unit able to sustain 2 MAC operations per cycle
  - Fully pipelined Floating point unit (FPU)
  - up to 200 MHz operation at full temperature range
  - up to 120 Kbyte Data Scratch-Pad RAM (DSPR)
  - up to 32 Kbyte Instruction Scratch-Pad RAM (PSPR)
  - 16 Kbyte Instruction Cache (ICACHE)
  - 8 Kbyte Data Cache (DCACHE)

Summary

- Performance gap between CPU and RAM
- Locality of programs
- Cache architecture