The Hierarchical Memory System

The Memory Hierarchy & Cache

- **The impact of real memory on CPU Performance.**
- **Main memory basic properties:**
  - **Memory Types: DRAM vs. SRAM**
- **The Motivation for The Memory Hierarchy:**
  - **CPU/Memory Performance Gap**
  - **The Principle Of Locality**
- **Memory Hierarchy Structure & Operation**
- **Cache Concepts:**
  - **Block placement strategy & Cache Organization:**
    - **Fully Associative, Set Associative, Direct Mapped.**
  - **Cache block identification: Tag Matching**
  - **Block replacement policy**
  - **Cache storage requirements**
  - **Unified vs. Separate Cache**
- **CPU Performance Evaluation with Cache:**
  - **Average Memory Access Time (AMAT)**
  - **Memory Stall cycles**
  - **Memory Access Tree**

Cache exploits memory access locality to:

- **Lower AMAT by hiding long main memory access latency. Thus cache is considered a memory latency-hiding technique.**
- **Lower demands on main memory bandwidth.**

Removing The Ideal Memory Assumption

- **So far we have assumed that ideal memory is used for both instruction and data memory in all CPU designs considered:**
  - **Single Cycle, Multi-cycle, and Pipelined CPUs.**

- **Ideal memory is characterized by a short delay or memory access time (one cycle) comparable to other components in the datapath.**
  - **i.e 2ns which is similar to ALU delays.**

- **Real memory utilizing Dynamic Random Access Memory (DRAM) has a much higher access time than other datapath components (80ns or more).**

Memory Access Time >> 1 CPU Cycle

- **Removing the ideal memory assumption in CPU designs leads to a large increase in clock cycle time and/or CPI greatly reducing CPU performance.**

As seen next →

Ideal Memory Access Time ≤ 1 CPU Cycle
Real Memory Access Time >> 1 CPU cycle

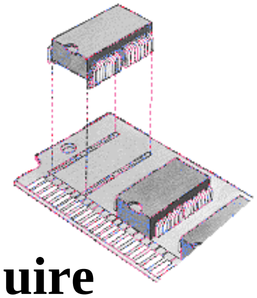
Removing The Ideal Memory Assumption

- **For example if we use real (non-ideal) memory with 80 ns access time (instead of 2ns) in our CPU designs then:**
- **Single Cycle CPU:**
  - **Loads will require 80ns + 1ns + 2ns + 80ns + 1ns = 164ns = C**
  - **The CPU clock cycle time C increases from 8ns to 164ns (125MHz to 6 MHz)**
  - **CPU is 20.5 times slower**
- **Multi Cycle CPU:**
  - **To maintain a CPU cycle of 2ns (500MHz) instruction fetch and data memory now take 80/2 = 40 cycles each resulting in the following CPIs**
    - **Arithmetic Instructions CPI = 40 + 3 = 43 cycles**
    - **Jump/Branch Instructions CPI = 40 + 2 = 42 cycles**
    - **Store Instructions CPI = 80 + 2 = 82 cycles**
    - **Load Instructions CPI = 80 + 3 = 83 cycles**
    - **Depending on instruction mix, CPU is 11-20 times slower**
- **Pipelined CPU:**
  - **To maintain a CPU cycle of 2ns, a pipeline with 83 stages is needed.**
  - **Data/Structural hazards over instruction/data memory access may lead to 40 or 80 stall cycles per instruction.**
  - **Depending on instruction mix CPI increases from 1 to 41-81 and the CPU is 41-81 times slower!**

T = I x CPI x C

Ideal Memory Access Time ≤ 1 CPU Cycle
Real Memory Access Time >> 1 CPU cycle

Main Memory



- Realistic main memory generally utilizes Dynamic RAM (DRAM), which use a single transistor to store a bit, but require a periodic data refresh by reading every row (~every 8 msec).
- DRAM is not ideal memory requiring possibly 80ns or more to access.
- Static RAM (SRAM) may be used as ideal main memory if the added expense, low density, high power consumption, and complexity is feasible (e.g. Cray Vector Supercomputers).
- Main memory performance is affected by:

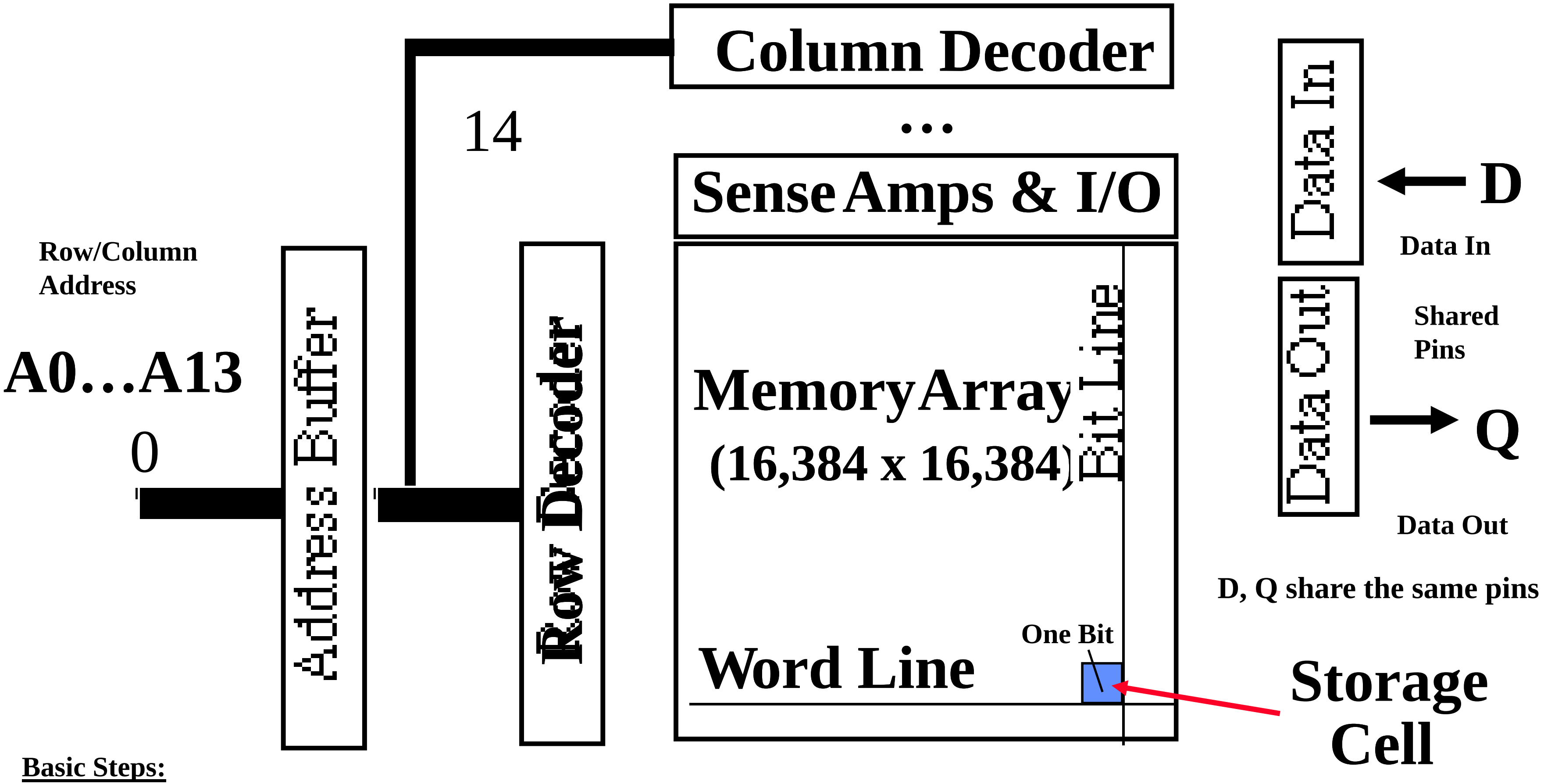
Will be explained later on

  - **Memory latency:** Affects cache miss penalty. Measured by:
    - **Access time:** The time it takes between a memory access request is issued to main memory and the time the requested information is available to cache/CPU.
    - **Cycle time:** The minimum time between requests to memory (greater than access time in DRAM to allow address lines to be stable)
  - **Peak Memory bandwidth:** The maximum sustained data transfer rate between main memory and cache/CPU.

RAM = Random Access Memory

Logical Dynamic RAM (DRAM) Chip Organization (16 Mbit)

Typical DRAM access time = 80 ns or more (non ideal)



D, Q share the same pins

(Single transistor per bit)

<u>Basic Steps:</u>

Control Signals:

1 - Row Access Strobe (RAS): Low to latch row address
2- Column Address Strobe (CAS): Low to latch column address
3- Write Enable (WE) or
Output Enable (OE)
4- Wait for data to be ready

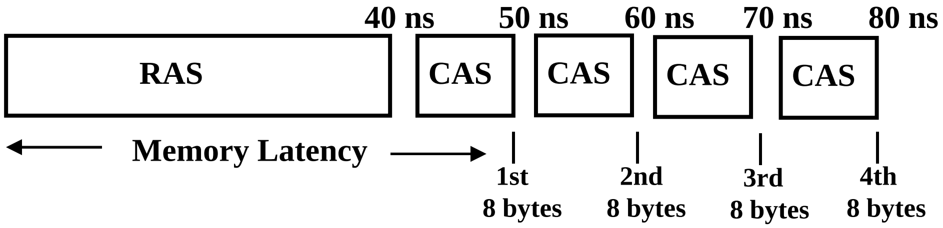
1 - Supply Row Address 2- Supply Column Address 3- Get Data

A periodic data refresh is required by reading every bit

Key DRAM Speed Parameters

- **Row Access Strobe (RAS)Time:**
  - **Minimum time from RAS (Row Access Strobe) line falling to the first valid data output.**
  - **A major component of memory latency and access time.**
  - **Only improves 5% every year.**
- **Column Access Strobe (CAS) Time/data transfer time:**
  - **The minimum time required to read additional data by changing column address while keeping the same row address.**
  - **Along with memory bus width, determines peak memory bandwidth.**

Example: for a memory with 8 bytes wide bus with RAS = 40 ns and CAS = 10 ns and the following simplified memory timing



40 ns 50 ns 60 ns 70 ns 80 ns

RAS CAS CAS CAS CAS

← Memory Latency →

1st 8 bytes

2nd 8 bytes

3rd 8 bytes

4th 8 bytes

Memory Latency = RAS + CAS = 50 ns
(to get first 8 bytes of data)

Peak Memory Bandwidth = Bus width / CAS = $8 \times 100 \times 10^6$ = 800 Mbytes/s

Minimum Miss penalty to fill a cache line with 32 byte block size = 80 ns (miss penalty)

Will be explained later on

DRAM = Dynamic Random Access Memory

DRAM Generations

Year	Size	RAS (ns)	CAS (ns)	DRAM Cycle Time	Memory Type	
1980	64 Kb	150-180	75	250 ns	Page Mode	Asynchronous DRAM
1983	256 Kb	120-150	50	220 ns	Page Mode	
1986	1 Mb	100-120	25	190 ns		
1989	4 Mb	80-100	20	165 ns	Fast Page Mode	
1992	16 Mb	60-80	15	120 ns	EDO	
1996	64 Mb	50-70	12	110 ns	PC66 SDRAM	Synchronous DRAM
1998	128 Mb	50-70	10	100 ns	PC100 SDRAM	
2000	256 Mb	45-65	7	90 ns	PC133 SDRAM	
2002	512 Mb	40-60	5	80 ns	PC2700 DDR SDRAM	
	8000:1 (Capacity)		15:1 (~bandwidth)	3:1 (Latency)	↓	

PC3200 DDR (2003)

↓

DDR2 SDRAM (2004)

↓

DDR3 SDRAM (2007-8)

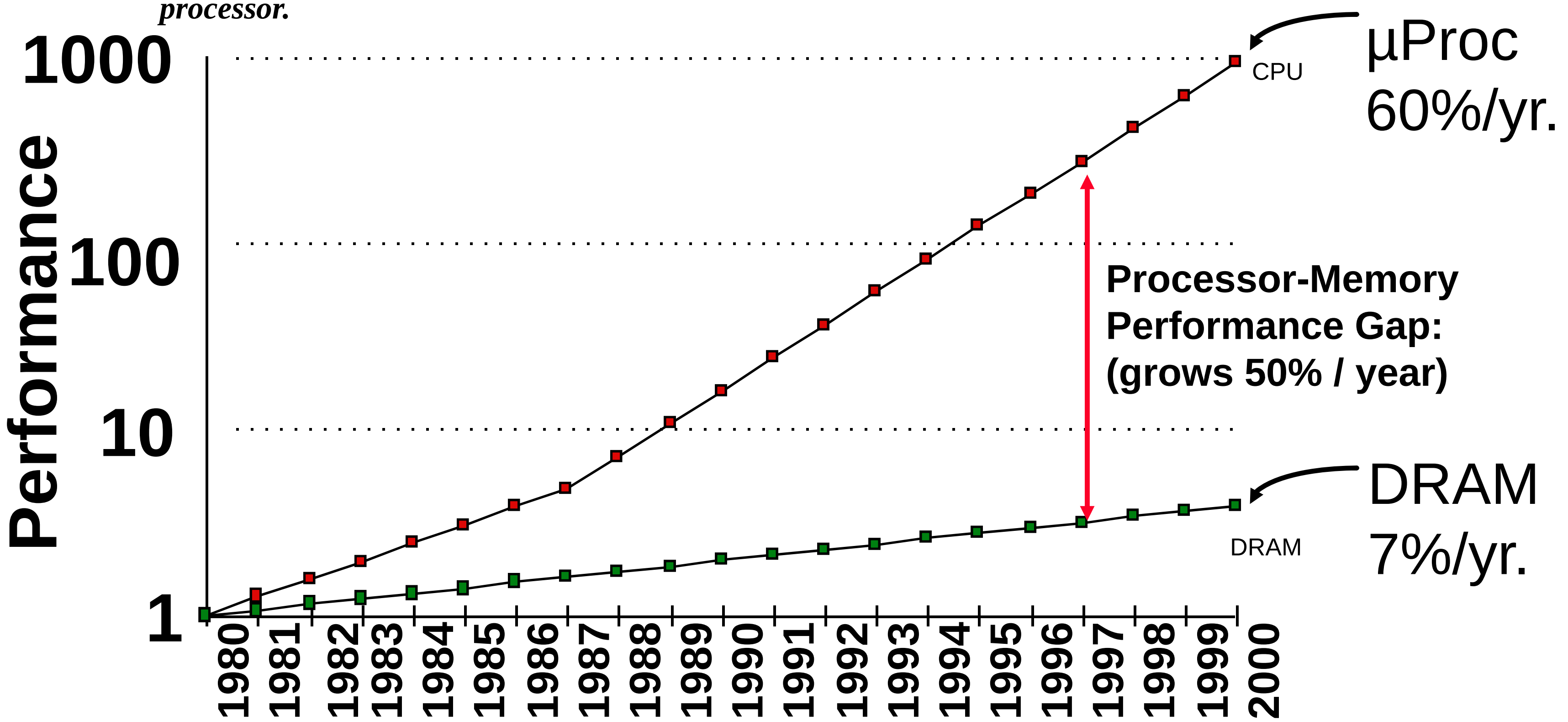
A major factor in cache miss penalty M

Will be explained later on

Memory Hierarchy: Motivation
Processor-Memory (DRAM) Performance Gap

i.e. Gap between memory access time (latency) and CPU cycle time

***<u>Memory Access Latency:</u>* The time between a memory access request is issued by the processor and the time the requested information (instructions or data) is available to the processor.**



Ideal Memory Access Time (latency) = 1 CPU Cycle
Real Memory Access Time (latency) >> 1 CPU cycle

Processor-DRAM Performance Gap: Impact of Real Memory on CPI

- To illustrate the performance impact of using non-ideal memory, we assume a single-issue pipelined RISC CPU with ideal CPI = 1.
- Ignoring other factors, the minimum cost of a full memory access in terms of number of wasted CPU cycles (added to CPI):

Year	CPU speed MHZ	CPU cycle ns	Memory Access ns	Minimum CPU memory stall cycles or instructions wasted
1986:	8	125	190	190/125 - 1 = 0.5
1989:	33	30	165	165/30 -1 = 4.5
1992:	60	16.6	120	120/16.6 -1 = 6.2
1996:	200	5	110	110/5 -1 = 21
1998:	300	3.33	100	100/3.33 -1 = 29
2000:	1000	1	90	90/1 - 1 = 89
2002:	2000	.5	80	80/.5 - 1 = 159
2004:	3000	.333	60	60.333 - 1 = 179

i.e wait cycles added to CPI

Ideal Memory Access Time ≤ 1 CPU Cycle
Real Memory Access Time >> 1 CPU cycle

Memory Hierarchy: Motivation

- The gap between CPU performance and main memory has been widening with higher performance CPUs creating performance bottlenecks for memory access instructions.

For Ideal Memory: Memory Access Time $\leq$ 1 CPU cycle

- The memory hierarchy is organized into several levels of memory with the smaller, faster memory levels closer to the CPU: registers, then primary Cache Level (L_1), then additional secondary cache levels (L_2, L_3…), then main memory, then mass storage (virtual memory).
- Each level of the hierarchy is usually a subset of the level below: data found in a level is also found in the level below (farther from CPU) but at lower speed (longer access time).
- Each level maps addresses from a larger physical memory to a smaller level of physical memory closer to the CPU.
- This concept is greatly aided by the principal of locality both temporal and spatial which indicates that programs tend to reuse data and instructions that they have used recently or those stored in their vicinity leading to working set of a program.

Memory Hierarchy: Motivation

The Principle Of Locality

- **Programs usually access a relatively small portion of their address space (instructions/data) at any instant of time <u>(program working set).</u>**

<u>Thus:</u> Memory Access Locality → Program Working Set

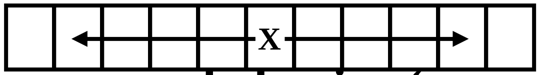
- **<u>Two Types of access locality:</u>**

1 – ***<u>Temporal Locality:</u>*** **If an item (instruction or data) is referenced, it will tend to be referenced again soon.**

  - **e.g. <u>instructions in the body of inner loops</u>**

2 – ***<u>Spatial locality:</u>*** **If an item is referenced, items whose addresses are close will tend to be referenced soon.**

  - **e.g. <u>sequential instruction</u> execution, <u>sequential access</u> to elements of <u>array</u>**



- **The presence of locality in program behavior (memory access patterns), makes it possible to satisfy a large percentage of program memory access needs (both instructions and data) using <u>faster</u> memory levels (<u>cache</u>) <u>with much less capacity</u> than program address space.**

Cache utilizes faster memory (SRAM)

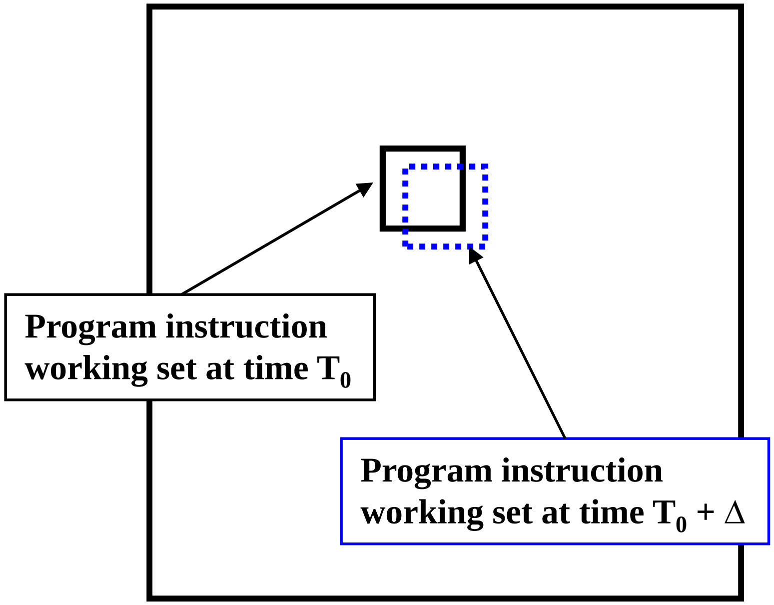
Access Locality & Program Working Set

- **Programs usually access a relatively small portion of their address space (instructions/data) at any instant of time (program working set).**
- **The presence of locality in program behavior and memory access patterns, makes it possible to satisfy a large percentage of program memory access needs using faster memory levels with *much less capacity* than program address space.**

(i.e Cache)

Using Static RAM (SRAM)

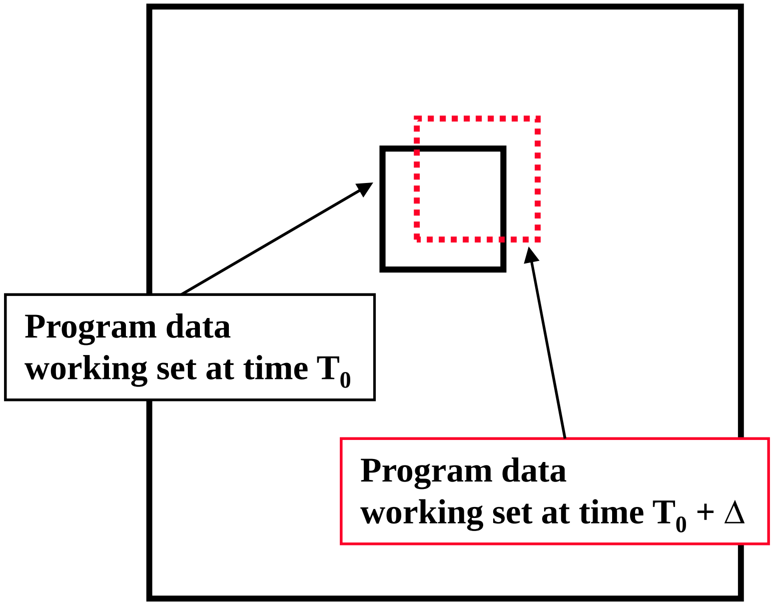
Program Instruction Address Space



Program instruction working set at time T_0

Program instruction working set at time $T_0 + \Delta$

Program Data Address Space

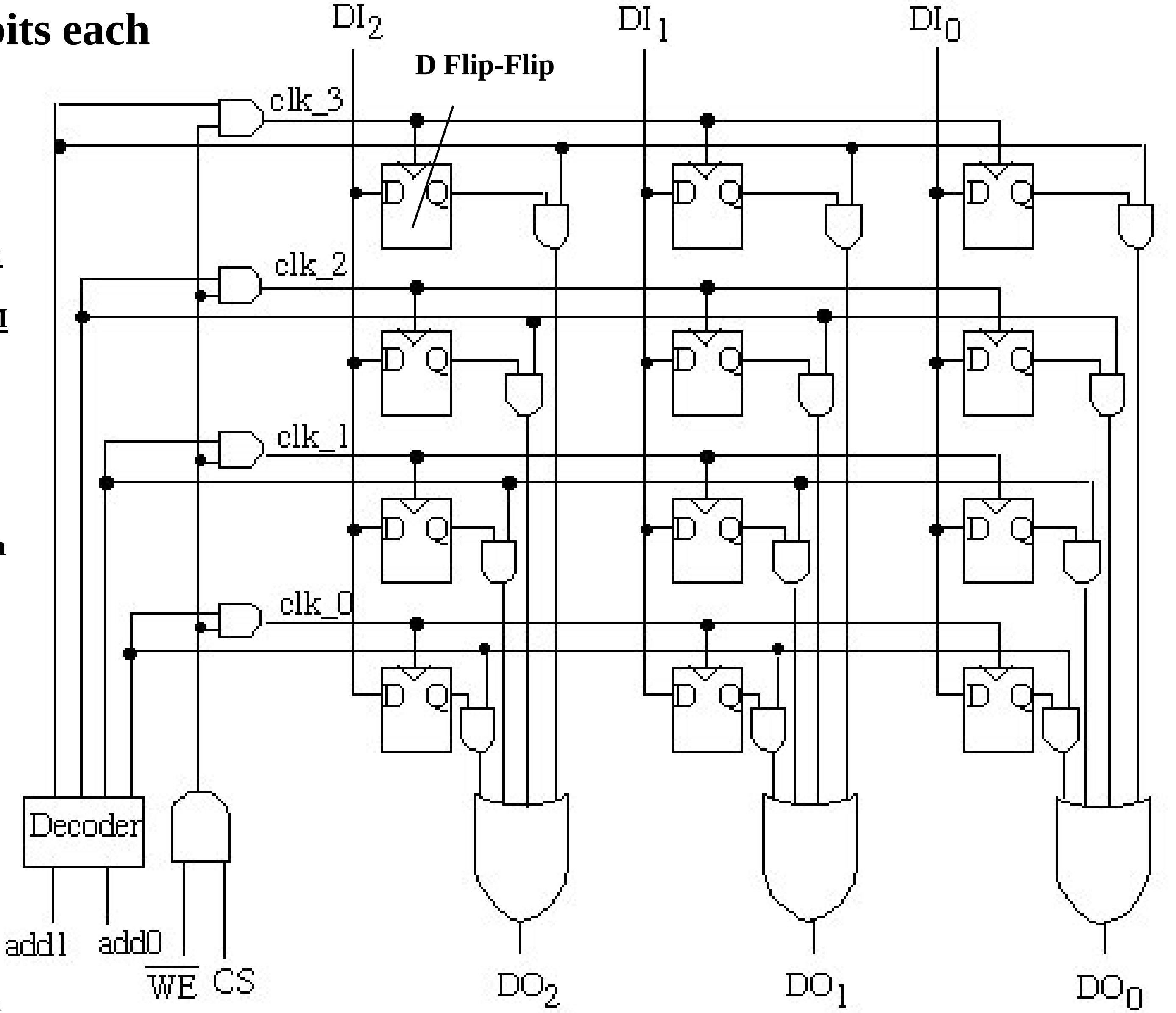


Program data working set at time T_0

Program data working set at time $T_0 + \Delta$

Locality in program memory access → Program Working Set

Static RAM (SRAM) Organization Example
4 words X 3 bits each



Static RAM (SRAM)
Each bit can represented by a D flip-flop

Advantages over DRAM:

Much Faster than DRAM

No refresh needed
(can function as on-chip ideal memory or cache)

Disadvantages:
(reasons not used as main memory)

Much lower density per SRAM chip than DRAM

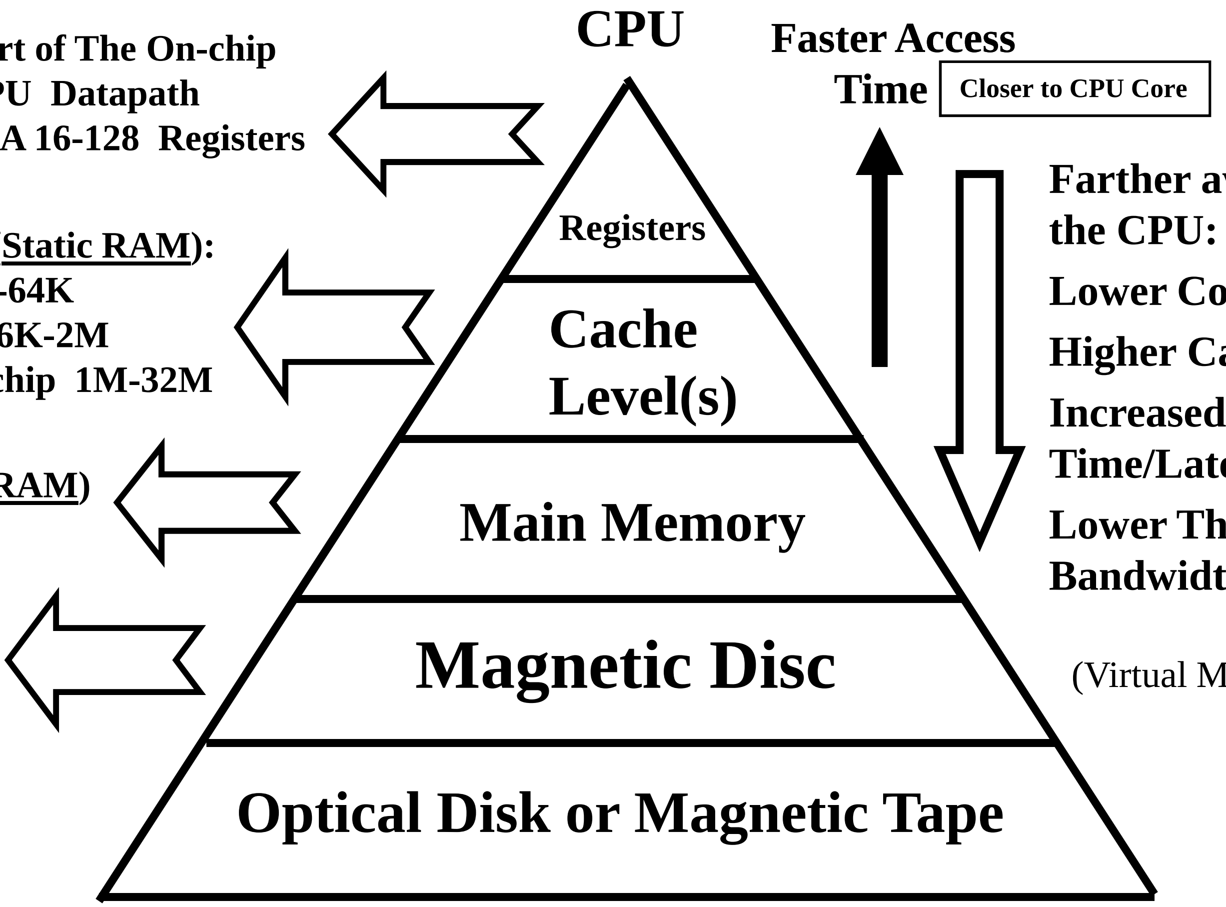
- **DRAM one transistor per bit**
- **SRAM 6-8 transistors per bit**

Higher cost than DRAM

High power consumption

Thus SRAM is not suitable for main system memory but suitable for the faster/smaller cache levels

Levels of The Memory Hierarchy



CPU

Part of The On-chip
CPU Datapath
ISA 16-128 Registers

One or more levels (<u>Static RAM</u>):
Level 1: On-chip 16-64K
Level 2: On-chip 256K-2M
Level 3: On or Off-chip 1M-32M

Dynamic RAM (<u>DRAM</u>)
256M-16G

Interface:
SCSI, RAID,
IDE, 1394
80G-300G

Registers

Cache
Level(s)

Main Memory

Magnetic Disc

Optical Disk or Magnetic Tape

Faster Access
Time

Closer to CPU Core

Farther away from
the CPU:

Lower Cost/Bit

Higher Capacity

Increased Access
Time/Latency

Lower Throughput/
Bandwidth

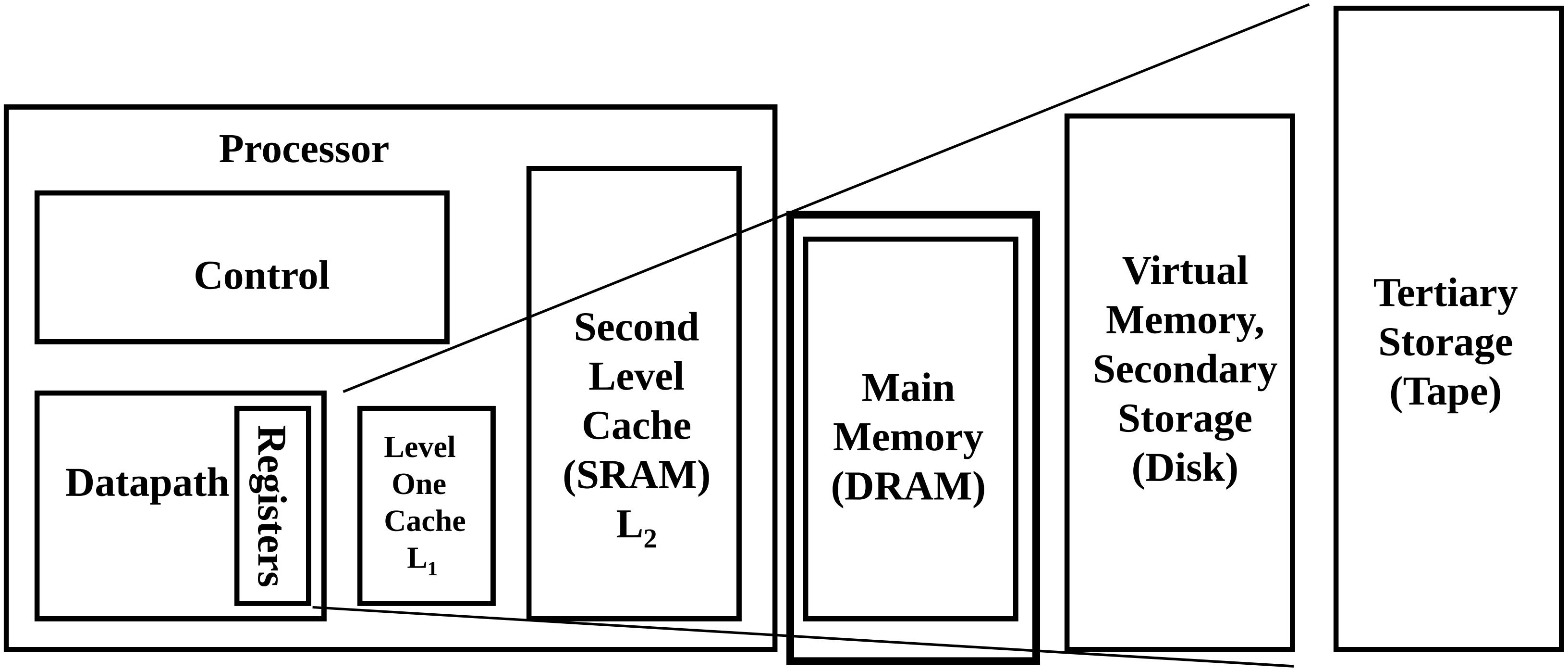
(Virtual Memory)

A Typical Memory Hierarchy

(With Two Levels of Cache)

← Faster

Larger Capacity →



Processor

Control

Datapath

Registers

Level One Cache L_1

Second Level Cache (SRAM) L_2

Main Memory (DRAM)

Virtual Memory, Secondary Storage (Disk)

Tertiary Storage (Tape)

Speed (ns): < 1s 1s 10s 10,000,000s (10s ms) 10,000,000,000s (10s sec)

Size (bytes): 100s Ks Ms Gs Ts

Memory Hierarchy Operation

- **If an instruction or operand is required by the CPU, the levels of the memory hierarchy are searched for the item starting with the level closest to the CPU (Level 1 cache):**

  L_1 Cache

  Hit rate for level one cache = H_1

  - **If the item is found, it's delivered to the CPU resulting in <u>a cache hit</u> without searching lower levels.**

    Hit rate for level one cache = H_1

  Cache Miss

  - **If the item is missing from an upper level, resulting in <u>a cache miss</u>, the level just below is searched.**

    Miss rate for level one cache = 1 – Hit rate = $1 - H_1$

  - **For systems with several levels of cache, the search continues with cache level 2, 3 etc.**
  - **If all levels of cache report a miss then main memory is accessed for the item.**
    - **CPU ↔ cache ↔ memory: <u>Managed by hardware</u>.**
  - **If the item is not found in main memory resulting in a page fault, then disk (virtual memory), is accessed for the item.**
    - **Memory ↔ disk: <u>Managed by the operating system</u> with hardware support**

Memory Hierarchy: Terminology

- **A Block:** The smallest unit of information transferred between two levels.
- **Hit:** Item is found in some block in the upper level (example: Block X)
  - **Hit Rate:** The fraction of memory access found in the upper level.
  - **Hit Time:** Time to access the upper level which consists of

    (S)RAM access time + Time to determine hit/miss
- **Miss:** Item needs to be retrieved from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in the upper level +

    M Time to deliver the missed block to the processor
- Hit Time << Miss Penalty M

e. g. H1

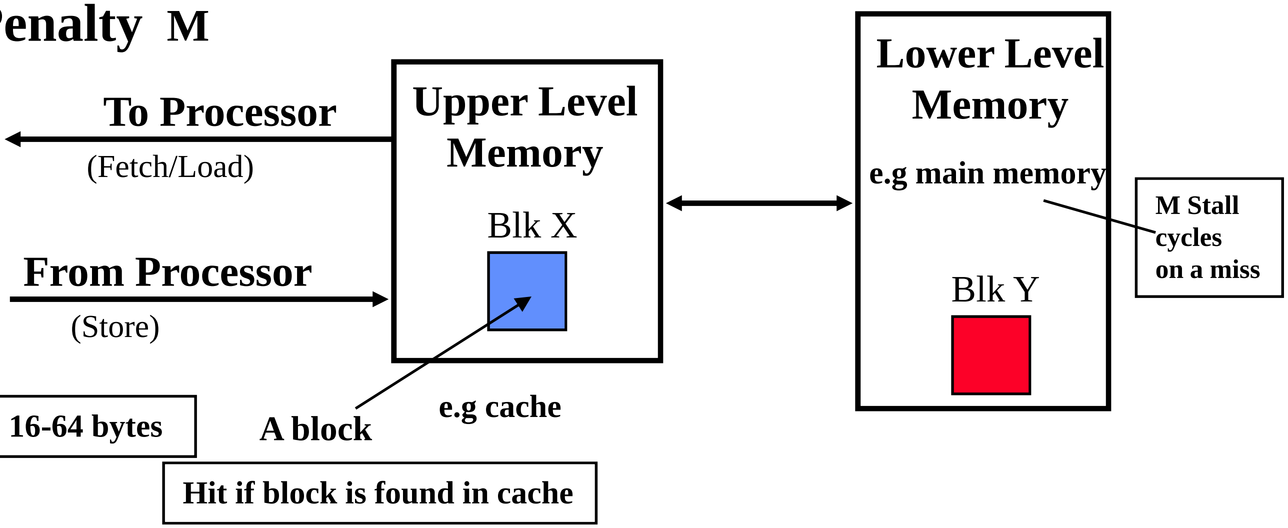
Hit rate for level one cache = H_1

Ideally = 1 Cycle

e. g. 1- H1

Miss rate for level one cache = 1 – Hit rate = 1 - H_1

Ideally = 1 Cycle



Hit if block is found in cache

Typical Cache Block (or line) Size: 16-64 bytes

Basic Cache Concepts

- **Cache is the first level of the memory hierarchy once the address leaves the CPU and is searched first for the requested data.**
- **If the data requested by the CPU is present in the cache, it is retrieved from cache and the data access is a cache hit otherwise a cache miss and data must be read from main memory.**
- **On a cache miss a block of data must be brought in from main memory to cache to possibly *replace* an existing cache block.**
- **The allowed block addresses where blocks can be mapped (placed) into cache from main memory is determined by *cache placement strategy*.**
- **Locating a block of data in cache is handled by cache block identification mechanism (tag checking).**
- **On a cache miss choosing the cache block being removed (replaced) is handled by the *block replacement strategy* in place.**

Cache Design & Operation Issues

Q1: Where can a block be placed cache? **Block placement/mapping**

(Block placement strategy & Cache organization)

- **Fully Associative, Set Associative, Direct Mapped.**

Very complex *Most common* *Simple but suffers from conflict misses*

Q2: How is a block found if it is in cache? **Locating a block**

(Block identification) **Cache Hit/Miss?**

- **Tag/Block.** **Tag Matching**

Q3: Which block should be replaced on a miss?

(Block replacement policy) **Block replacement**

- **Random, Least Recently Used (LRU), FIFO.**

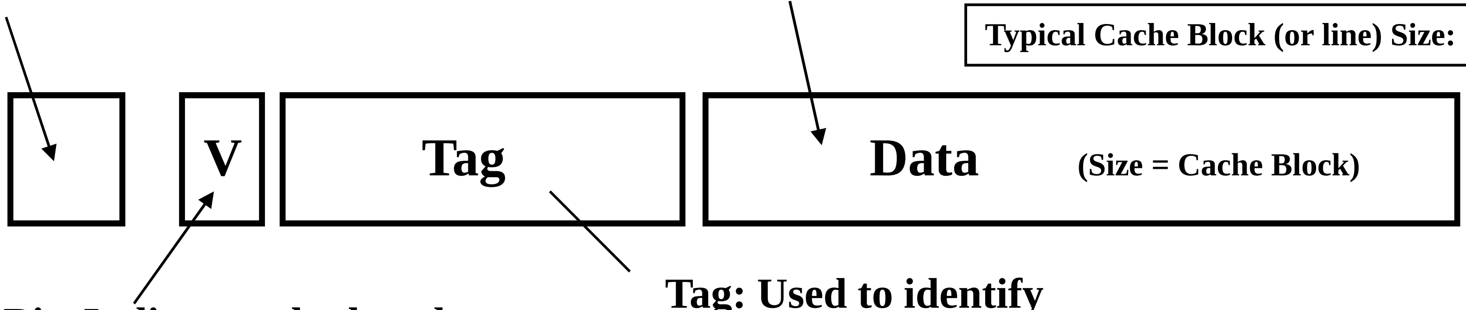
Cache Block Frame

Cache is comprised of a number of cache block frames

Other status/access bits:
(e,g. modified, read/write access bits)

Data Storage: Number of bytes is **the size of a cache block** or cache line size (Cached instructions or data go here)

Typical Cache Block (or line) Size: 16-64 bytes



Valid Bit: Indicates whether the cache block frame contains valid data

Tag: Used to identify if the address supplied matches the address of the data stored

The tag and valid bit are used to determine whether we have a cache hit or miss

Nominal Cache Size

Stated **nominal cache capacity or size** only accounts for space used to store instructions/data and **ignores the storage needed for tags and status bits**:

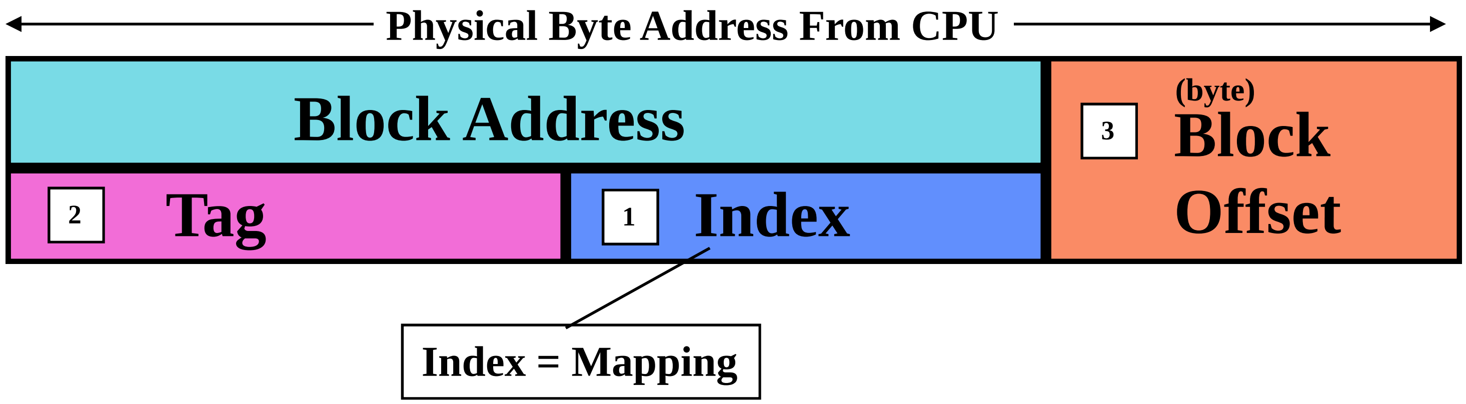
Nominal Cache Capacity = Number of Cache Block Frames x Cache Block Size

e.g For a cache with block size = 16 bytes and 1024 = 2^{10} = 1k cache block frames
Nominal cache capacity = 16 x 1k = 16 Kbytes

Cache utilizes faster memory (SRAM)

Locating A Data Block in Cache

- **Each block frame in cache has an address tag.**
- **The tags of every cache block that might contain the required data are checked or searched in parallel.** **Tag Matching**
- **A valid bit is added to the tag to indicate whether this entry contains a valid address.**
- **The byte address from the CPU to cache is divided into:**
  - **A block address, further divided into:**
    - 1 **An index field to choose/map a block set in cache.**
      (no index field when fully associative).
    - 2 **A tag field to search and match addresses in the selected set.**
  - **A byte block offset to select the data from the block.** 3


Physical Byte Address From CPU

Block Address

2 **Tag**

1 **Index**

3 **(byte) Block Offset**

Index = Mapping

Cache Organization & Placement Strategies

Placement strategies or mapping of a main memory data block onto cache block frame addresses divide cache into three organizations:

1 **Direct mapped cache:** **A block can be placed in only one location (cache block frame), given by the mapping function:**

Least complex to implement
suffers from conflict misses

Mapping Function

index= (Block address) MOD (Number of blocks in cache)

= Frame #

2 **Fully associative cache:** **A block can be placed anywhere in cache. (no mapping function).**

Most complex cache organization to implement

3 **Set associative cache:** **A block can be placed in a restricted set of places, or cache block frames. A set is a group of block frames in the cache. A block is first mapped onto the set and then it can be placed anywhere within the set. The set in this case is chosen by:**

Mapping Function

index = (Block address) MOD (Number of sets in cache)

= Set #

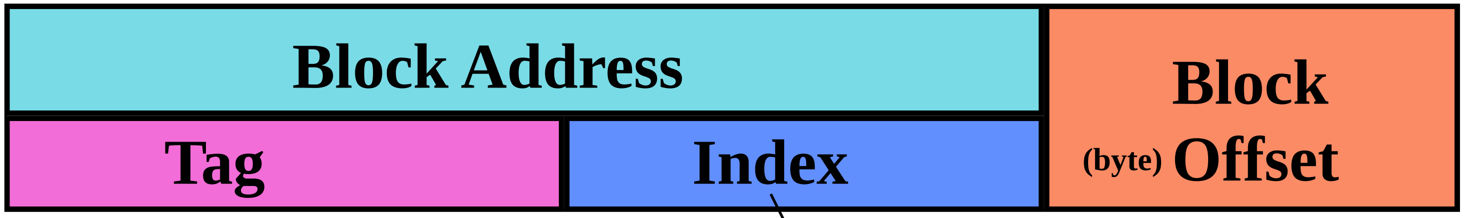
If there are *n* blocks in a set the cache placement is called *n*-way set-associative.

Most common cache organization

Address Field Sizes/Mapping

← Physical Byte Address Generated by CPU →

(The size of this address depends on amount of cacheable physical main memory)



Mapping

Block Byte offset size = $\log_2$(block size)

Index size = $\log_2$(Total number of blocks/associativity)

Number of Sets in cache

Tag size = address size - index size - offset size

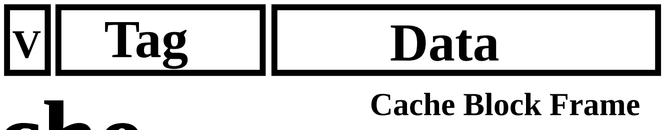
<u>**Mapping function:**</u> (From memory block to cache)

Cache set or block frame number = Index =

= (Block Address) MOD (Number of Sets)

Fully associative cache has no index field or mapping function

Cache Organization:
Direct Mapped Cache

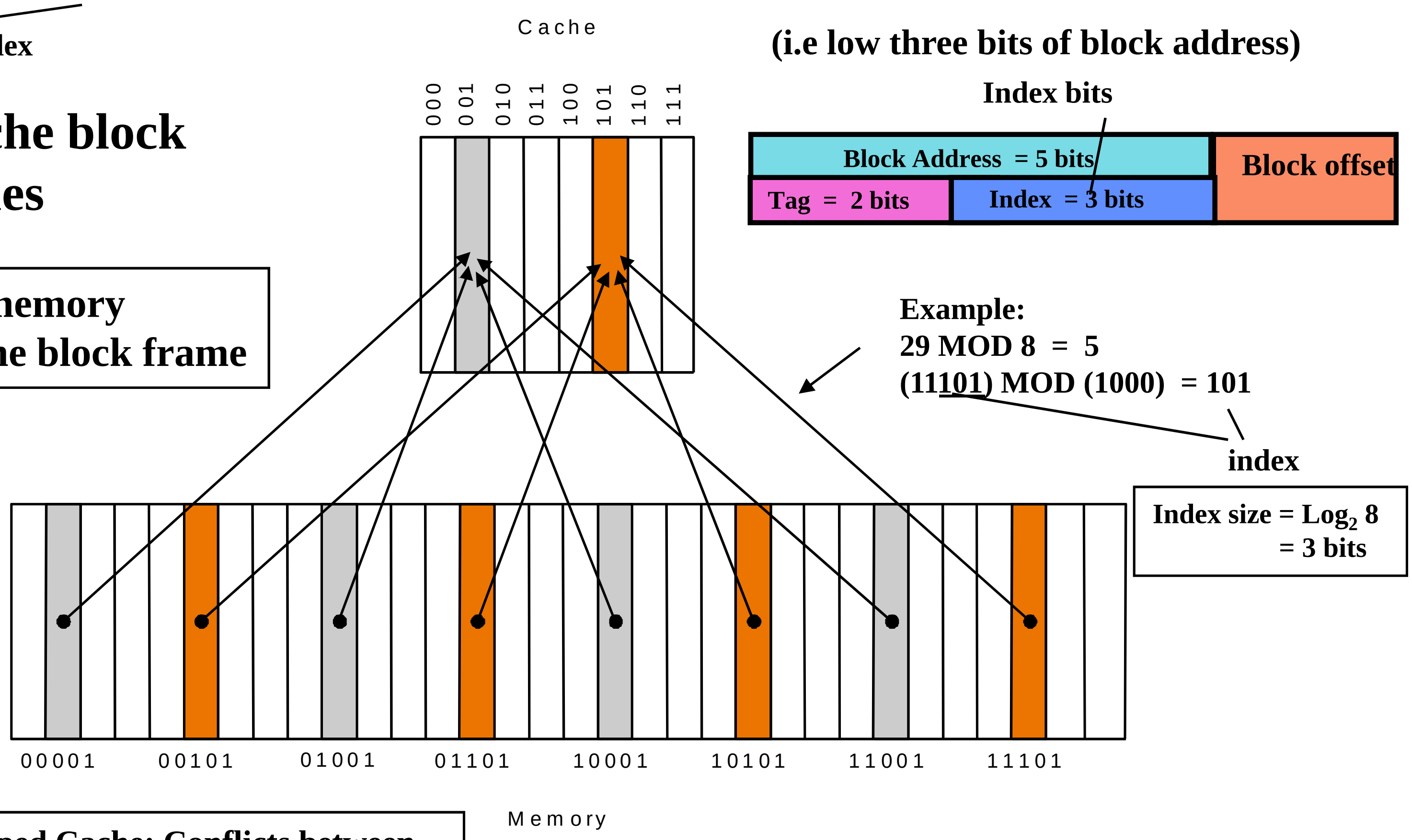


A block in memory can be placed in <u>one location (cache block frame)</u> only, given by: (Block address) MOD (Number of blocks in cache)

In this case, mapping function: (Block address) MOD (8) **= Index**

Index

(i.e low three bits of block address)



8 cache block frames

Here four blocks in memory map to the same cache block frame

32 memory blocks cacheable

Example:
29 MOD 8 = 5
(11101) MOD (1000) = 101

index

Index size = Log_2 8 = 3 bits

<u>Limitation of Direct Mapped Cache:</u> Conflicts between memory blocks that map to the same cache block frame may result in conflict cache misses

4KB Direct Mapped Cache Example

4 Kbytes = Nominal Cache Capacity

1K = 2^{10} = 1024 Blocks

Each block = one word

(4 bytes)

Can cache up to

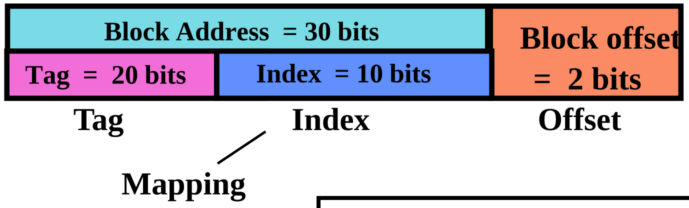
2^{32} bytes = 4 GB

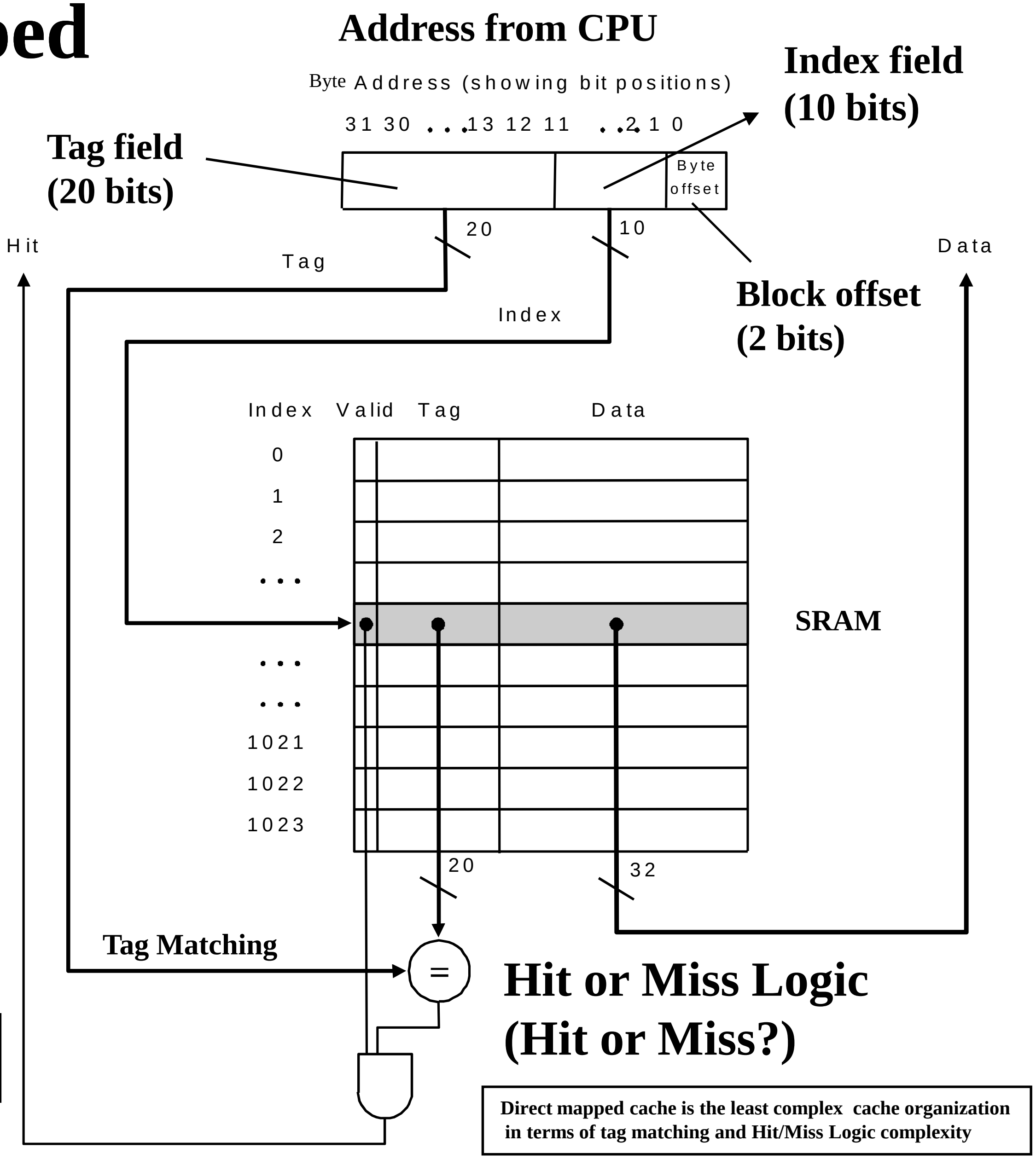
of memory

Mapping function:

Cache Block frame number =
(Block address) MOD (1024)

i.e . Index field or 10 low bits of
block address





Hit or Miss Logic
(Hit or Miss?)

Direct mapped cache is the least complex cache organization in terms of tag matching and Hit/Miss Logic complexity

Hit Access Time = SRAM Delay + Hit/Miss Logic Delay

Direct Mapped Cache Operation Example

Here:
Block Address = Word Address

- Given a series of 16 memory address references given as word addresses:

  1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17.

- Assume a direct mapped cache with 16 one-word blocks that is initially empty, label each reference as a hit or miss and show the final content of cache
- Here: Block Address = Word Address Mapping Function = (Block Address) MOD 16 = Index

Cache Block Frame#		1	4	8	5	20	17	19	56	9	11	4	43	5	6	9	17	
		Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	$\boxed{\text{Hit}}$	Miss	$\boxed{\text{Hit}}$	$\boxed{\text{Hit}}$	Hit/Miss
0																		
1		$\boxed{1}$	1	1	1	1	$\boxed{17}$	17	17	17	17	17	17	17	17	17	$\boxed{17}$	
2																		
3								$\boxed{19}$	19	19	19	19	19	19	19	19	19	
4			$\boxed{4}$	4	4	$\boxed{20}$	20	20	20	20	20	$\boxed{4}$	4	4	4	4	4	
5					$\boxed{5}$	5	5	5	5	5	5	5	5	$\boxed{5}$	5	5	5	
6															$\boxed{6}$	6	6	
7																		
8				$\boxed{8}$	8	8	8	8	$\boxed{56}$	56	56	56	56	56	56	56	56	
9										$\boxed{9}$	9	9	9	9	9	$\boxed{9}$	9	
10																		
11											$\boxed{11}$	11	$\boxed{43}$	43	43	43	43	
12																		
13																		
14																		
15																		

Initial Cache Content (empty)

Cache Content After Each Reference

Final Cache Content

Hit Rate = # of hits / # memory references = 3/16 = 18.75%

Mapping Function = Index = (Block Address) MOD 16
i.e 4 low bits of block address

64KB Direct Mapped Cache Example

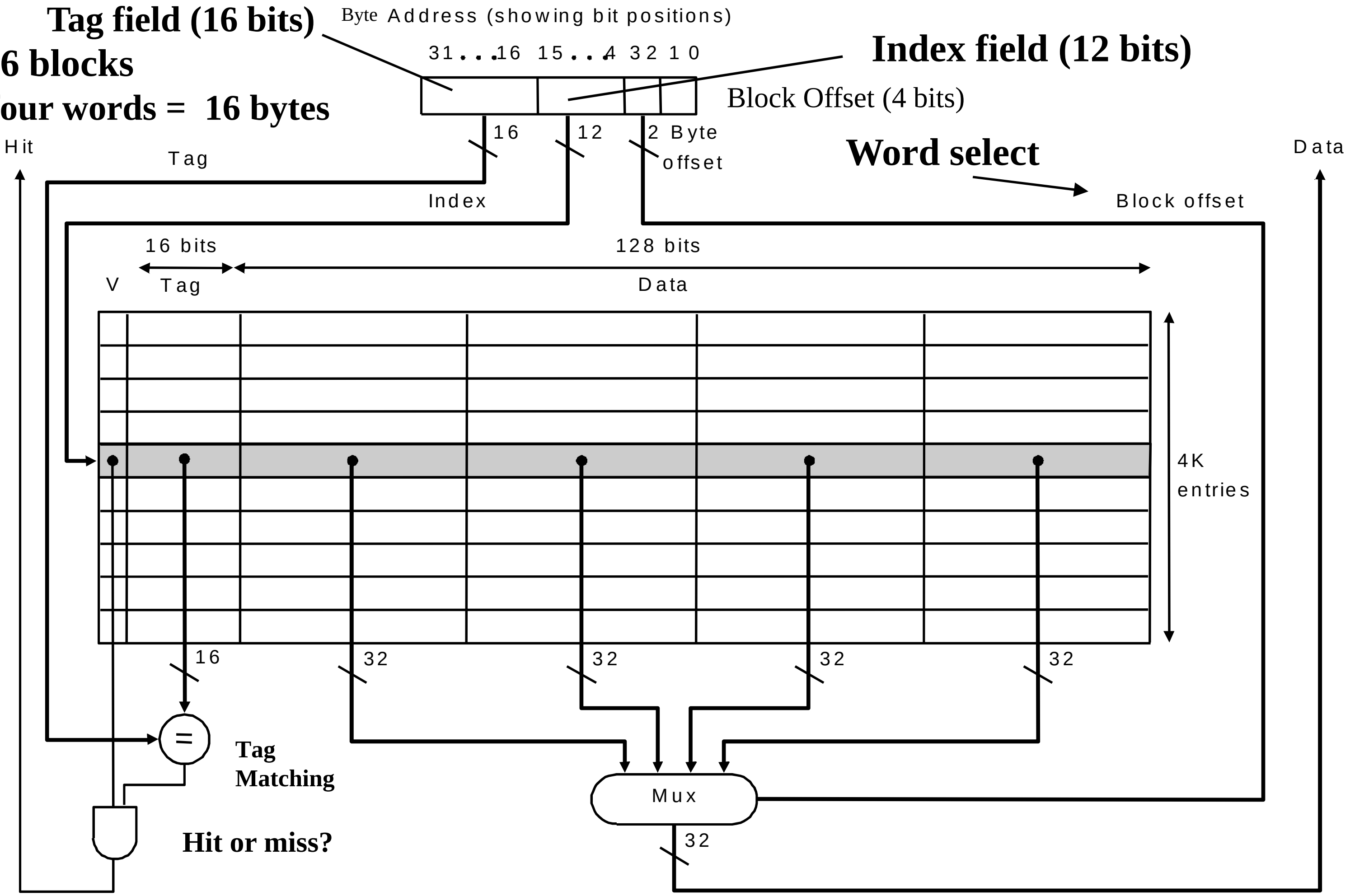
Nominal Capacity

4K= 2^{12} = 4096 blocks

Each block = four words = 16 bytes

Can cache up to 2^{32} bytes = 4 GB of memory

SRAM



Tag field (16 bits)

Index field (12 bits)

Block Offset (4 bits)

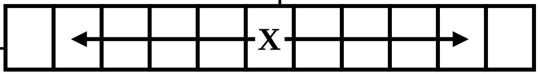
Word select

Tag Matching

Hit or miss?

Typical cache Block or line size: 32-64 bytes

Larger cache blocks take better advantage of spatial locality and thus may result in a lower miss rate



Block Address = 28 bits

Tag = 16 bits **Index = 12 bits**

Block offset = 4 bits

<u>Mapping Function:</u> **Cache Block frame number = (Block address) MOD (4096)**

i.e. index field or 12 low bit of block address

Hit Access Time = SRAM Delay + Hit/Miss Logic Delay

Direct Mapped Cache Operation Example

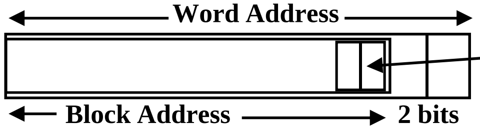
With Larger Cache Block Frames

- **Given the same series of 16 memory address references given as word addresses:**

  1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17.
- **Assume a direct mapped cache with four word blocks and a total of 16 words that is initially empty, label each reference as a hit or miss and show the final content of cache**
- **Cache has 16/4 = 4 cache block frames (each has four words)**
- **Here: Block Address = Integer (Word Address/4)**

i.e We need to find block addresses for mapping

Or



Mapping Function = (Block Address) MOD 4

i.e 2 low bits of block address

Block addresses

Word addresses

		0	1	2	1	5	4	4	14	2	2	1	10	1	1	2	4	
Cache Block Frame#		1	4	8	5	20	17	19	56	9	11	4	43	5	6	9	17	
		Miss	Miss	Miss	$\boxed{\text{Hit}}$	Miss	Miss	$\boxed{\text{Hit}}$	Miss	Miss	$\boxed{\text{Hit}}$	Miss	Miss	$\boxed{\text{Hit}}$	$\boxed{\text{Hit}}$	Miss	$\boxed{\text{Hit}}$	Hit/Miss
0		$\boxed{0}$	0	0	0	0	$\boxed{16}$	16	16	16	16	16	16	16	16	16	$\boxed{16}$	
1			$\boxed{4}$	4	$\boxed{4}$	$\boxed{20}$	20	20	20	20	20	$\boxed{4}$	4	$\boxed{4}$	$\boxed{4}$	4	4	
2				$\boxed{8}$	8	8	8	8	$\boxed{56}$	$\boxed{8}$	$\boxed{8}$	8	$\boxed{40}$	40	40	$\boxed{8}$	8	
3																		

Initial Cache Content (empty)

Starting word address of Cache Frames Content After Each Reference

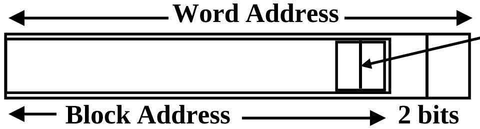
Final Cache Content

Hit Rate = # of hits / # memory references = 6/16 = 37.5%

Here: Block Address ≠ Word Address

Word Addresses vs. Block Addresses and Frame Content for Previous Example

Block size = 4 words



Mapping (index)

i.e low two bits of block address

Given Word address	Block address	Cache Block Frame # (Block address)mod 4	word address range in frame (4 words)
1	0	0	0-3
4	1	1	4-7
8	2	2	8-11
5	1	1	4-7
20	5	1	20-23
17	4	0	16-19
19	4	0	16-19
56	14	2	56-59
9	2	2	8-11
11	2	2	8-11
4	1	1	4-7
43	10	2	40-43
5	1	1	4-7
6	1	1	4-7
9	2	2	8-11
17	4	0	16-19

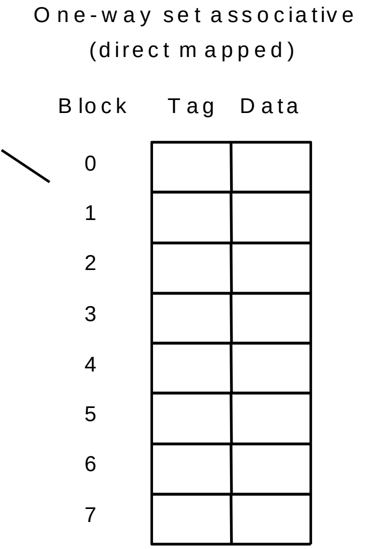
Block Address = Integer (Word Address/4)

Cache Organization: Set Associative Cache

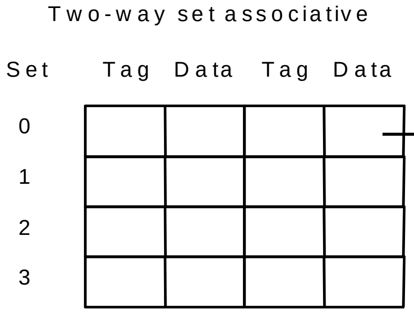
Cache Block Frame

Why set associative?

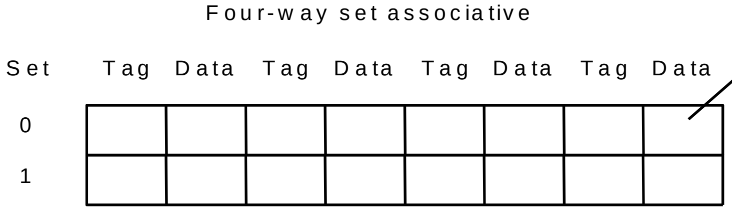
Set associative cache reduces cache misses by <u>reducing conflicts</u> between blocks that would have been mapped to the same cache block frame in the case of direct mapped cache



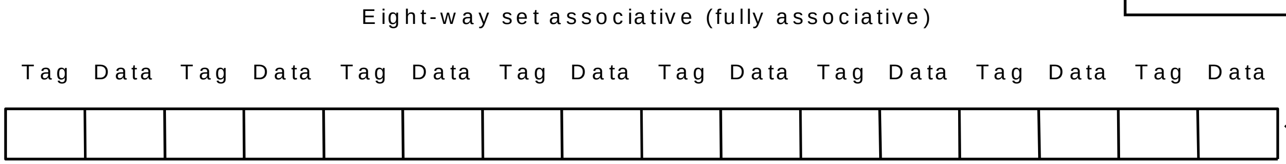
1-way set associative:
(direct mapped)
1 block frame per set



2-way set associative:
2 blocks frames per set



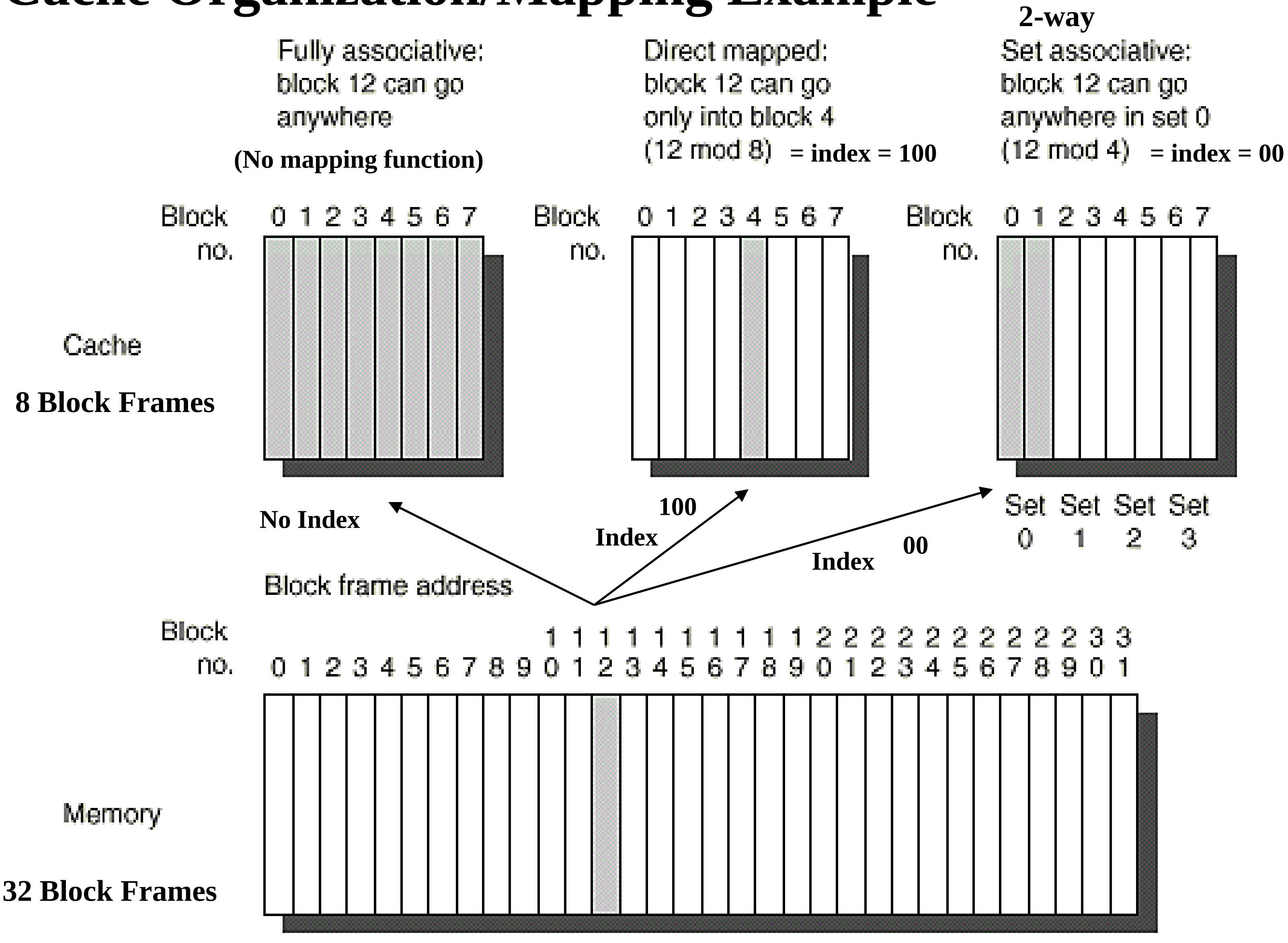
4-way set associative:
4 blocks frames per set



8-way set associative:
8 blocks frames per set
In this case it becomes fully associative since total number of block frames = 8

A cache with a total of 8 cache block frames shown

Cache Organization/Mapping Example



Fully associative:
block 12 can go
anywhere
(No mapping function)

Direct mapped:
block 12 can go
only into block 4
(12 mod 8) **= index = 100**

2-way
Set associative:
block 12 can go
anywhere in set 0
(12 mod 4) **= index = 00**

8 Block Frames

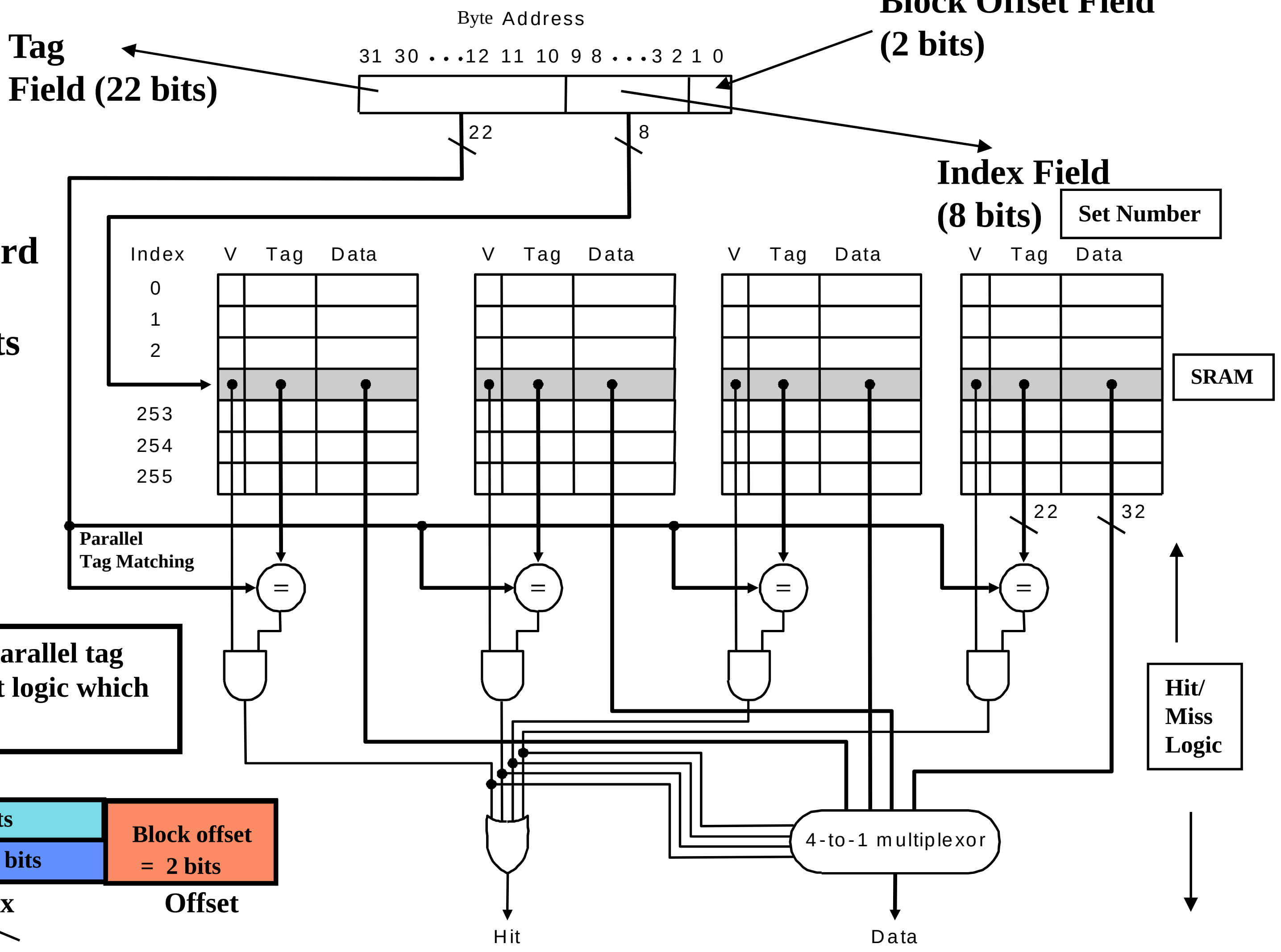
32 Block Frames

This example cache has eight block frames and memory has 32 blocks.

12 = 1100

4K Four-Way Set Associative Cache: MIPS Implementation Example

Nominal Capacity



Tag Field (22 bits)

Block Offset Field (2 bits)

Index Field (8 bits)

Set Number

SRAM

Parallel Tag Matching

Hit/ Miss Logic

1024 block frames
Each block = one word
4-way set associative
1024 / 4= 2^8= 256 sets

Can cache up to 2^{32} bytes = 4 GB of memory

Set associative cache requires parallel tag matching and more complex hit logic which may increase hit time

Block Address = 30 bits		Block offset = 2 bits
Tag = 22 bits	Index = 8 bits	
Tag	Index	Offset

Mapping Function: Cache Set Number = index= (Block address) MOD (256)

Hit Access Time = SRAM Delay + Hit/Miss Logic Delay

Cache Replacement Policy

Which block to replace on a cache miss?

- **When a cache miss occurs the cache controller may have to select a block of cache data to be removed from a cache block frame and replaced with the requested data, such a block is selected by one of three methods:**

  (No cache replacement policy in direct mapped cache) **No choice on which block to replace**

1 – **Random:**
  - **Any block is randomly selected for replacement providing uniform allocation.**
  - **Simple to build in hardware. Most widely used cache replacement strategy.**

2 – **Least-recently used (LRU):**
  - **Accesses to blocks are recorded and and the block replaced is the one that was not used for the longest period of time.**
  - **Full LRU is *expensive* to implement, as the number of blocks to be tracked increases, and is usually approximated by block usage bits that are cleared at regular time intervals.**

3 – **First In, First Out (FIFO):**
  - **Because LRU can be complicated to implement, this approximates LRU by determining the oldest block rather than LRU**

Miss Rates for Caches with Different Size, Associativity & Replacement Algorithm

Sample Data

Nominal

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Lower miss rate is better

Program steady state cache miss rates are given
Initially cache is empty and miss rates ~ 100%

FIFO replacement miss rates (not shown here) is <u>better than random</u> but <u>worse than LRU</u>

For SPEC92

Miss Rate = 1 – Hit Rate = 1 – H1

2-Way Set Associative Cache Operation Example

- **Given the same series of 16 memory address references given as word addresses:**

Here: Block Address = Word Address

1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17. (LRU Replacement)

- **Assume a two-way set associative cache with one word blocks and a total size of 16 words that is initially empty, label each reference as a hit or miss and show the final content of cache**
- **Here: Block Address = Word Address Mapping Function = Set # = (Block Address) MOD 8**

Cache Set #		1	4	8	5	20	17	19	56	9	11	4	43	5	6	9	17	
		Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Miss	$\boxed{\text{Hit}}$	Miss	$\boxed{\text{Hit}}$	Miss	$\boxed{\text{Hit}}$	$\boxed{\text{Hit}}$	Hit/Miss
0				$\boxed{8}$	8	8	8	8	8	8	8	8	8	8	8	8	8	LRU
									$\boxed{56}$	56	56	56	56	56	56	56	56	
1		$\boxed{1}$	1	1	1	1	1	1	1	$\boxed{9}$	9	9	9	9	9	$\boxed{9}$	9	LRU
							$\boxed{17}$	17	17	17	17	17	17	17	17	17	$\boxed{17}$	
2																		
3								$\boxed{19}$	19	19	19	19	$\boxed{43}$	43	43	43	43	
											$\boxed{11}$	11	11	11	11	11	11	LRU
4			$\boxed{4}$	4	4	4	4	4	4	4	4	$\boxed{4}$	4	4	4	4	4	
						$\boxed{20}$	20	20	20	20	20	20	20	20	20	20	20	LRU
5					$\boxed{5}$	5	5	5	5	5	5	5	5	$\boxed{5}$	5	5	5	
6															$\boxed{6}$	6	6	
7																		

Initial Cache Content (empty)

Cache Content After Each Reference

Final Cache Content

Hit Rate = # of hits / # memory references = 4/16 = 25%

Replacement policy: LRU = Least Recently Used

Cache Organization/Addressing Example

- **Given the following:**
  - **A single-level L_1 cache with 128 cache block frames**
    - **Each block frame contains four words (16 bytes)**

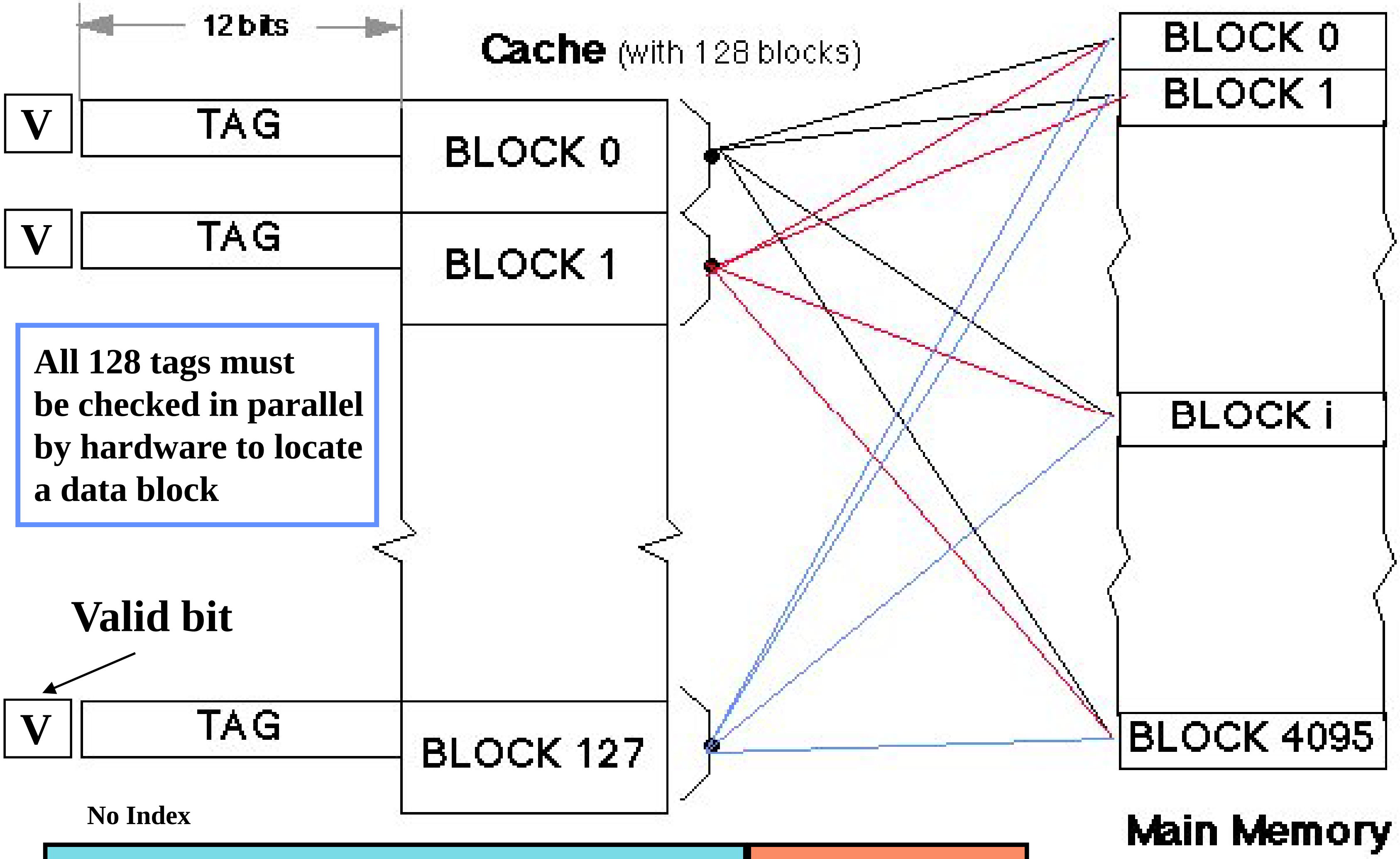
i.e block size = 16 bytes

  - **16-bit memory addresses to be cached (64K bytes main memory or 4096 memory blocks)**

64 K bytes = 2^{16} bytes
Thus byte address size = 16 bits

- **Show the cache organization/mapping and cache address fields for:**
    - **Fully Associative cache.**
    - **Direct mapped cache.**
    - **2-way set-associative cache.**

Cache Example: Fully Associative Case



All 128 tags must be checked in parallel by hardware to locate a data block

Valid bit

No Index

Block Address = 12 bits

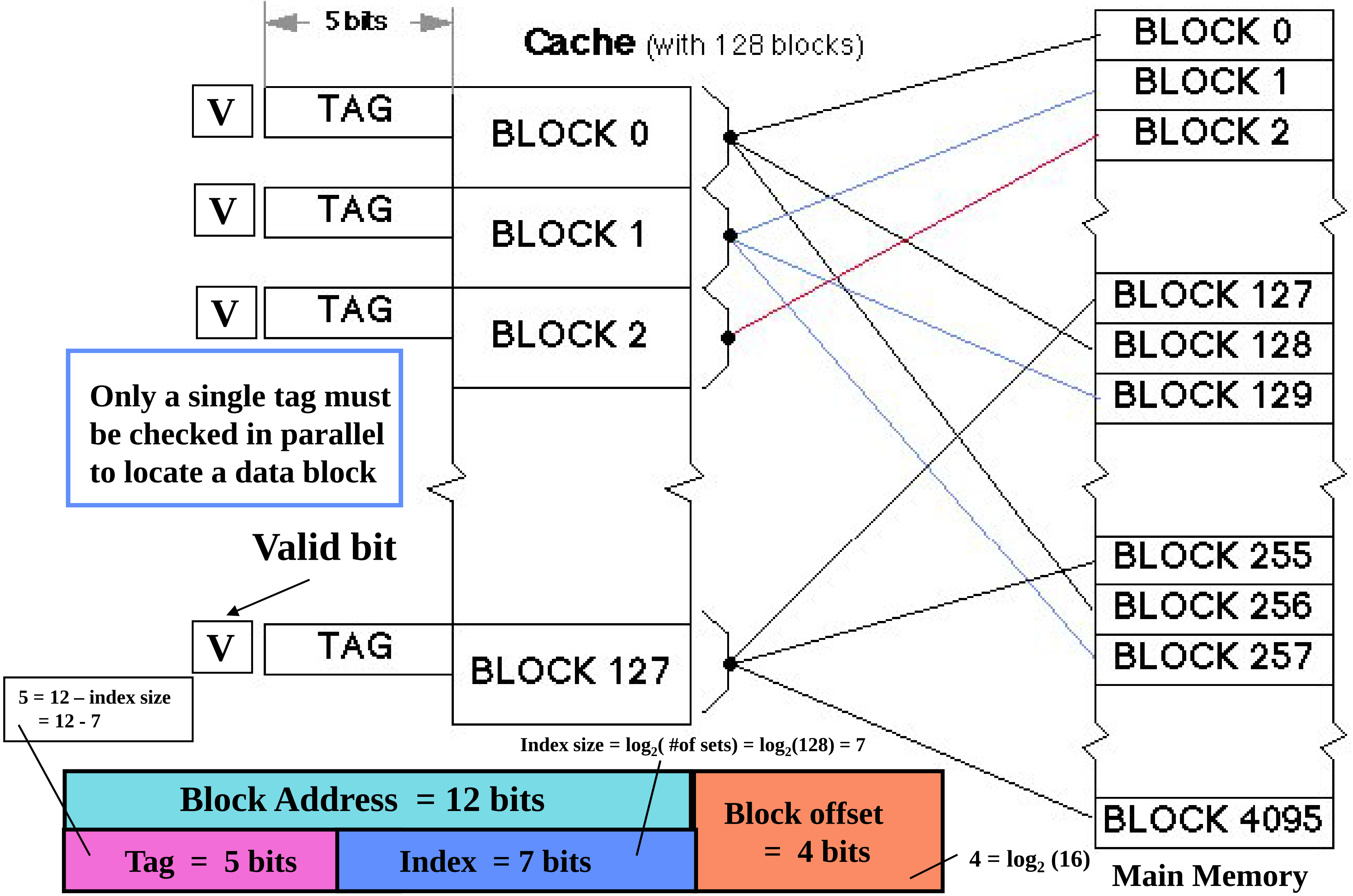
Tag = 12 bits

Block offset = 4 bits

$4 = \log_2(16)$

Mapping Function = none (no index field)
i.e Any block in memory can be mapped to any cache block frame

Cache Example: Direct Mapped Case



Only a single tag must be checked in parallel to locate a data block

Valid bit

5 = 12 – index size
= 12 - 7

Index size = $\log_2$(#of sets) = $\log_2(128) = 7$

Block Address = 12 bits

Tag = 5 bits

Index = 7 bits

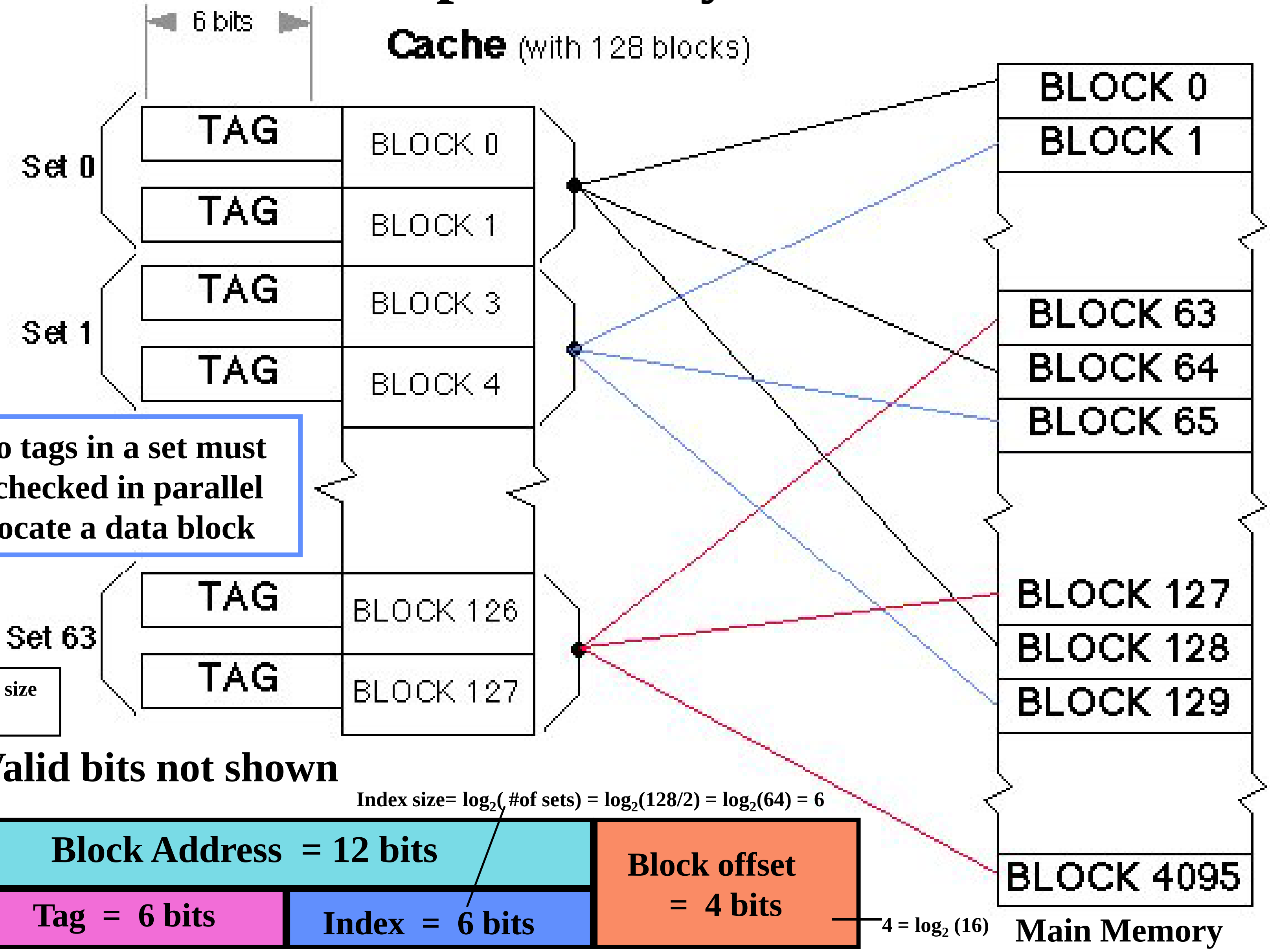
Block offset = 4 bits

$4 = \log_2 (16)$

Mapping Function: Cache Block frame number = Index = (Block address) MOD (128)

2^5 = 32 blocks in memory map onto the same cache block frame

Cache Example: 2-Way Set-Associative



Two tags in a set must be checked in parallel to locate a data block

6 = 12 – index size
= 12 - 6

Valid bits not shown

Index size= $\log_2$(#of sets) = $\log_2(128/2) = \log_2(64) = 6$

Block Address = 12 bits		Block offset = 4 bits
Tag = 6 bits	Index = 6 bits	

4 = $\log_2$ (16)

Mapping Function: Cache Set Number = Index = (Block address) MOD (64)

2^6 = 64 blocks in memory map onto the same cache set

Calculating Number of Cache Bits Needed

Address Fields

Cache Block Frame (or just cache block)

- **How many total bits are needed for a direct- mapped cache with 64 KBytes of data and one word blocks, assuming a 32-bit address?**
  - 64 Kbytes = 16 K words = 2^{14} words = 2^{14} blocks
  - Block size = 4 bytes => offset size = $\log_2(4)$ = 2 bits,
  - #sets = #blocks = 2^{14} => index size = 14 bits
  - Tag size = address size - index size - offset size = 32 - 14 - 2 = 16 bits
  - Bits/block = data bits + tag bits + valid bit = 32 + 16 + 1 = 49
  - Bits in cache = #blocks x bits/block = 2^{14} x 49 = 98 Kbytes

i.e nominal cache
Capacity = 64 KB

Number of cache block frames

Actual number of bits in a cache block frame

- **How many total bits would be needed for a 4-way set associative cache to store the same amount of data?**
  - Block size and #blocks does not change.
  - #sets = #blocks/4 = $(2^{14})/4$ = 2^{12} => index size = 12 bits
  - Tag size = address size - index size - offset = 32 - 12 - 2 = 18 bits
  - Bits/block = data bits + tag bits + valid bit = 32 + 18 + 1 = 51
  - Bits in cache = #blocks x bits/block = 2^{14} x 51 = 102 Kbytes
- **Increase associativity => increase bits in cache**

More bits in tag

Word = 4 bytes

1 k = 1024 = 2^{10}

Calculating Cache Bits Needed

Address Fields

Cache Block Frame (or just cache block)

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and 8 word (32 byte) blocks, assuming a 32-bit address (it can cache 2^{32} bytes in memory)?

  Nominal size

  - 64 Kbytes = 2^{14} words = $(2^{14})/8 = 2^{11}$ blocks

    Number of cache block frames

  - block size = 32 bytes

    => offset size = block offset + byte offset = $\log_2(32) = 5$ bits,

  - #sets = #blocks = 2^{11} => index size = 11 bits
  - tag size = address size - index size - offset size = 32 - 11 - 5 = 16 bits
  - bits/block = data bits + tag bits + valid bit = 8 x 32 + 16 + 1 = 273 bits

    Actual number of bits in a cache block frame

  - bits in cache = #blocks x bits/block = 2^{11} x 273 = 68.25 Kbytes

- Increase block size => decrease bits in cache.

Fewer cache block frames thus fewer tags/valid bits

Word = 4 bytes   1 k = 1024 = 2^{10}

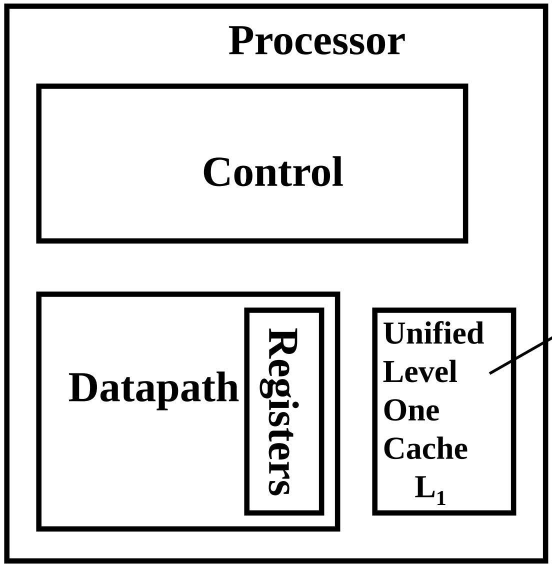
Unified vs. Separate Level 1 Cache

- Unified Level 1 Cache (Princeton Memory Architecture).
  A single level 1 (L_1) cache is used for both instructions and data.

AKA Shared Cache

Or Split

- Separate instruction/data Level 1 caches (Harvard Memory Architecture):
  The level 1 (L_1) cache is split into two caches, one for instructions (instruction cache, L_1 I-cache) and the other for data (data cache, L_1 D-cache).



Processor

Control

Datapath

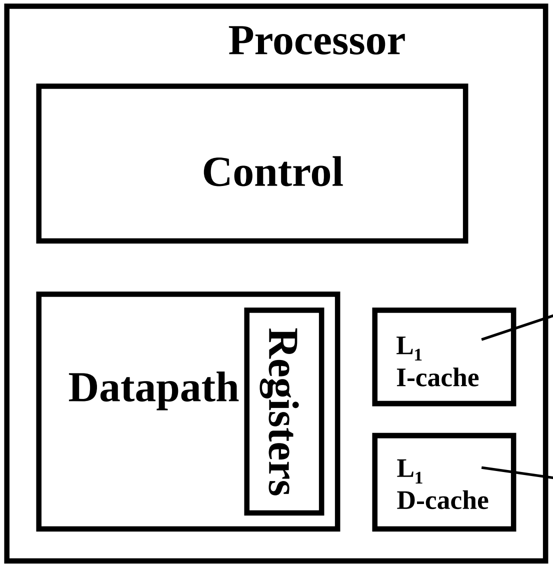
Registers

Unified Level One Cache L_1

Accessed for both instructions And data

AKA shared

Unified Level 1 Cache
(Princeton Memory Architecture)



Processor

Control

Datapath

Registers

L_1 I-cache

L_1 D-cache

Most Common

Instruction Level 1 Cache

Data Level 1 Cache

Separate (Split) Level 1 Caches
(Harvard Memory Architecture)

Why?

Split Level 1 Cache is more preferred in pipelined CPUs to avoid instruction fetch/Data access structural hazards

Memory Hierarchy/Cache Performance:

Average Memory Access Time (AMAT), Memory Stall cycles

- **The Average Memory Access Time (AMAT):** **The number of cycles required to complete an average memory access request by the CPU.**
- **Memory stall cycles per memory access:** **The number of stall cycles added to CPU execution cycles for one memory access.**
- **Memory stall cycles per average memory access = (AMAT -1)**
- **For ideal memory: AMAT = 1 cycle, this results in zero memory stall cycles.**
- **Memory stall cycles per average instruction =**

Number of memory accesses per instruction

x Memory stall cycles per average memory access

Instruction Fetch

= (1 + fraction of loads/stores) x (AMAT -1)

$$\text{Base CPI} = \text{CPI}_{\text{execution}} = \text{CPI with ideal memory}$$

$$\boxed{\text{CPI} = \text{CPI}_{\text{execution}} + \text{Mem Stall cycles per instruction}}$$

cycles = CPU cycles

Cache Performance: Single Level L1 Princeton (Unified) Memory Architecture

CPUtime = Instruction count x CPI x Clock cycle time

$CPI_{execution}$ = CPI with ideal memory

$CPI = CPI_{execution}$ + Mem Stall cycles per instruction

Mem Stall cycles per instruction =

Memory accesses per instruction x Memory stall cycles per access

i.e No hit penalty

Assuming no stall cycles on a cache hit (cache access time = 1 cycle, stall = 0)

Cache Hit Rate = H1 Miss Rate = 1- H1 Miss Penalty = M

Memory stall cycles per memory access = Miss rate x Miss penalty = (1- H1) x M

AMAT = 1 + Miss rate x Miss penalty = 1 + (1- H1) x M

Memory accesses per instruction = (1 + fraction of loads/stores)

Miss Penalty = M = the number of stall cycles resulting from missing in cache

= Main memory access time - 1

Thus for a unified L1 cache with no stalls on a cache hit:

$CPI = CPI_{execution}$ + (1 + fraction of loads/stores) x (1 - H1) x M

AMAT = 1 + (1 - H1) x M

$CPI = CPI_{execution}$ + (1 + fraction of loads and stores) x stall cycles per access

$= CPI_{execution}$ + (1 + fraction of loads and stores) x (AMAT – 1)

Memory Access Tree:
For Unified Level 1 Cache

Probability to be here

CPU Memory Access

H1

100% or 1

(1-H1)

Unified L_1

L1 Hit:

% = Hit Rate = H1

Hit Access Time = 1

Stall cycles per access = 0

Stall= H1 x 0 = 0

(No Stall)

Assuming:
Ideal access on a hit

L1 Miss:

% = (1- Hit rate) = (1-H1)

Access time = M + 1

Stall cycles per access = M

Stall = M x (1-H1)

$$\text{AMAT} = \underbrace{H1}_{\text{Hit Rate}} \times \underbrace{1}_{\text{Hit Time}} + \underbrace{(1 - H1)}_{\text{Miss Rate}} \times \underbrace{(M+1)}_{\text{Miss Time}} = 1 + M \times (1 - H1)$$

Stall Cycles Per Access = AMAT - 1 = M x (1 -H1)

$\text{CPI} = \text{CPI}_{\text{execution}}$ + (1 + fraction of loads/stores) x M x (1 -H1)

M = Miss Penalty = stall cycles per access resulting from missing in cache

M + 1 = Miss Time = Main memory access time

H1 = Level 1 Hit Rate 1- H1 = Level 1 Miss Rate

AMAT = 1 + Stalls per average memory access

Cache Performance Example

- **Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache.**
- **$CPI_{execution}$ = 1.1** **(i.e base CPI with ideal memory)**
- **Instruction mix: 50% arith/logic, 30% load/store, 20% control**
- **Assume a cache miss rate of 1.5% and a miss penalty of M= 50 cycles.**

CPI = $CPI_{execution}$ + mem stalls per instruction

Mem Stalls per instruction =

(1- H1) **M**

Mem accesses per instruction x Miss rate x Miss penalty

Mem accesses per instruction = 1 + .3 = 1.3

Instruction fetch **Load/store**

Mem Stalls per memory access = (1- H1) x M = .015 x 50 = .75 cycles

AMAT = 1 +.75 = 1.75 cycles

Mem Stalls per instruction = 1.3 x .015 x 50 = 0.975

CPI = 1.1 + .975 = 2.075

The ideal memory CPU with no misses is 2.075/1.1 = 1.88 times faster

M = Miss Penalty = stall cycles per access resulting from missing in cache

Cache Performance Example

- **Suppose for the <u>previous example</u> we <u>double the clock rate</u> to 400 MHz, how much faster is this machine, assuming similar miss rate, instruction mix?**
- **Since memory speed is not changed, the miss penalty takes more CPU cycles:**

Miss penalty = M = 50 x 2 = 100 cycles.

CPI = 1.1 + 1.3 x .015 x 100 = 1.1 + 1.95 = 3.05

$$\text{Speedup} = (CPI_{old} \times C_{old}) / (CPI_{new} \times C_{new})$$

$$= 2.075 \times 2 / 3.05 = 1.36$$

The new machine is only 1.36 times faster rather than 2 times faster due to the increased effect of cache misses.

→ ***CPUs with higher clock rate, have more cycles per cache miss and more memory impact on CPI.***

Cache Performance:

Data Level 1 Cache

L_1 D-cache

L_1 I-cache

Instruction Level 1 Cache

__Usually:__ Data Miss Rate >> Instruction Miss Rate

Miss rate = 1 – data H1

Miss rate = 1 – instruction H1

Single Level L1 Harvard (Split) Memory Architecture

For a CPU with separate or __split level one (L1)__ caches for instructions and data (Harvard memory architecture) and __no stalls for cache hits__:

CPUtime = Instruction count x CPI x Clock cycle time

$$\boxed{\text{CPI} = \text{CPI}_{\text{execution}} + \text{Mem Stall cycles per instruction}}$$

This is one method to find stalls per instruction another method is shown in next slide ⟶

Mem Stall cycles per instruction =

Instruction Fetch Miss rate x M +

Data Memory Accesses Per Instruction x Data Miss Rate x M

1- Instruction H1

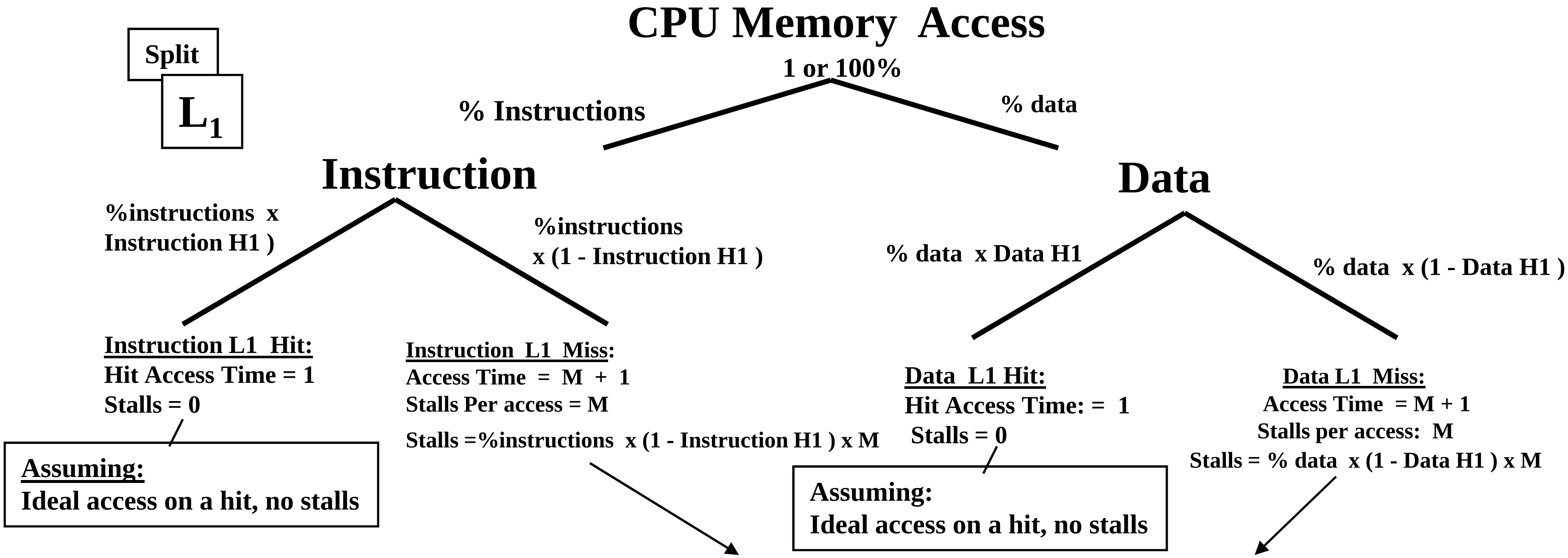
Fraction of Loads and Stores

1- Data H1

M = Miss Penalty = stall cycles per access to main memory resulting from missing in cache

$\text{CPI}_{\text{execution}}$ = base CPI with ideal memory

Memory Access Tree
For Separate Level 1 Caches



Split

L_1

CPU Memory Access

1 or 100%

% Instructions

% data

Instruction

Data

%instructions x Instruction H1)

%instructions x (1 - Instruction H1)

% data x Data H1

% data x (1 - Data H1)

Instruction L1 Hit:
Hit Access Time = 1
Stalls = 0

Assuming:
Ideal access on a hit, no stalls

Instruction L1 Miss:
Access Time = M + 1
Stalls Per access = M
Stalls =%instructions x (1 - Instruction H1) x M

Data L1 Hit:
Hit Access Time: = 1
Stalls = 0

Assuming:
Ideal access on a hit, no stalls

Data L1 Miss:
Access Time = M + 1
Stalls per access: M
Stalls = % data x (1 - Data H1) x M

Stall Cycles Per Access = % Instructions x (1 - Instruction H1) x M + % data x (1 - Data H1) x M

AMAT = 1 + Stall Cycles per access

Stall cycles per instruction = (1 + fraction of loads/stores) x Stall Cycles per access

CPI = $\text{CPI}_{\text{execution}}$ + Stall cycles per instruction
= $\text{CPI}_{\text{execution}}$ + (1 + fraction of loads/stores) x Stall Cycles per access

M = Miss Penalty = stall cycles per access resulting from missing in cache
M + 1 = Miss Time = Main memory access time
Data H1 = Level 1 Data Hit Rate **1- Data H1 = Level 1 Data Miss Rate**
Instruction H1 = Level 1 Instruction Hit Rate **1- Instruction H1 = Level 1 Instruction Miss Rate**
% Instructions = Percentage or fraction of instruction fetches out of all memory accesses
% Data = Percentage or fraction of data accesses out of all memory accesses

Split L1 Cache Performance Example

- **Suppose a CPU uses separate level one (L1) caches for instructions and data (Harvard memory architecture) with different miss rates for instruction and data access:**
  - **$CPI_{execution}$ = 1.1** **(i.e base CPI with ideal memory)**
  - **Instruction mix: 50% arith/logic, 30% load/store, 20% control**
  - **Assume a cache miss rate of 0.5% for instruction fetch and a cache data miss rate of 6%.**
  - **A cache hit incurs no stall cycles while a cache miss incurs 200 stall cycles for both memory reads and writes.** **M**
- **<u>Find the resulting stalls per access, AMAT and CPI using this cache?</u>**

CPI = $CPI_{execution}$ + mem stalls per instruction

Memory Stall cycles per instruction = Instruction Fetch Miss rate x Miss Penalty +
Data Memory Accesses Per Instruction x Data Miss Rate x Miss Penalty

Memory Stall cycles per instruction = 0.5/100 x 200 + 0.3 x 6/100 x 200 = 1 + 3.6 = 4.6 cycles

Stall cycles per average memory access = 4.6/1.3 = 3.54 cycles

AMAT = 1 + Stall cycles per average memory access = 1 + 3.54 = 4.54 cycles

CPI = $CPI_{execution}$ + mem stalls per instruction = 1.1 + 4.6 = 5.7 cycles

- **<u>What is the miss rate of a single level unified cache that has the same performance?</u>**

4.6 = 1.3 x Miss rate x 200 which gives a miss rate of 1.8 % for an equivalent unified cache

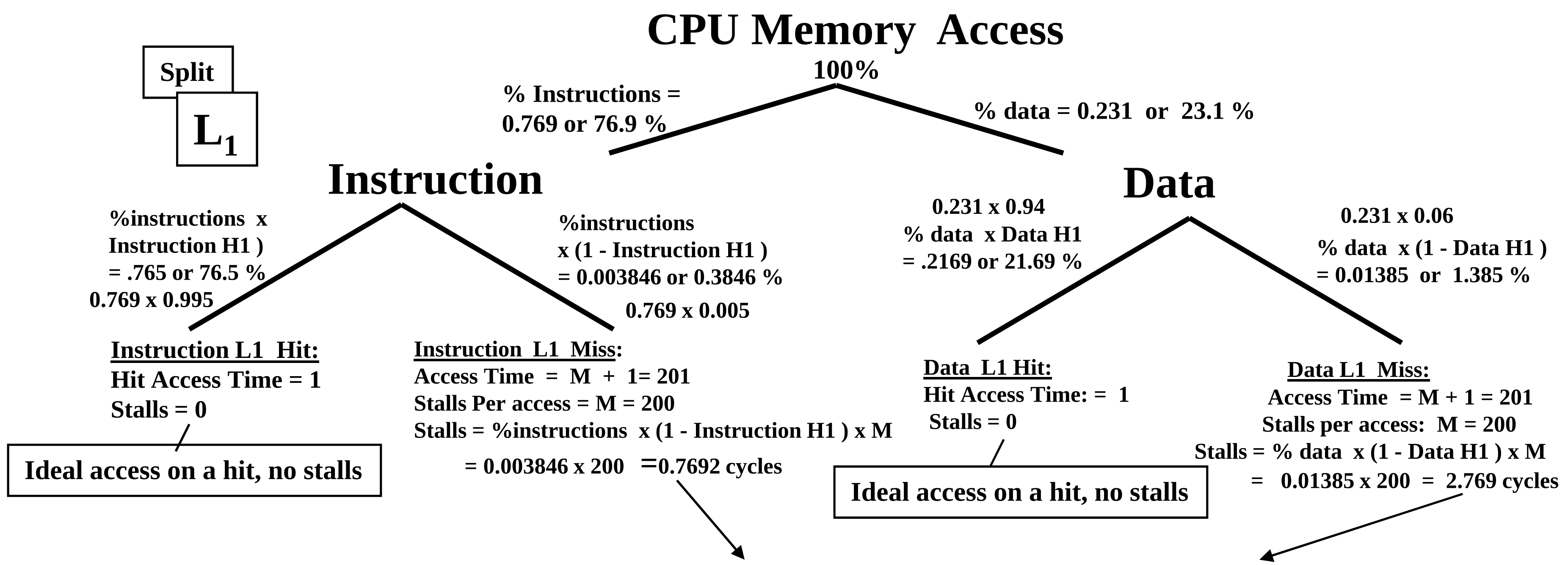
- **<u>How much faster is the CPU with ideal memory?</u>**

The CPU with ideal cache (no misses) is 5.7/1.1 = 5.18 times faster
With no cache at all the CPI would have been = 1.1 + 1.3 X 200 = 261.1 cycles !!

Memory Access Tree For Separate Level 1 Caches Example

For Last Example

30% of all instructions executed are loads/stores, thus:
Fraction of instruction fetches out of all memory accesses = 1/ (1+0.3) = 1/1.3 = 0.769 or 76.9 %
Fraction of data accesses out of all memory accesses = 0.3/ (1+0.3) = 0.3/1.3 = 0.231 or 23.1 %



Split

L_1

CPU Memory Access

100%

% Instructions =
0.769 or 76.9 %

% data = 0.231 or 23.1 %

Instruction

%instructions x
Instruction H1)
= .765 or 76.5 %
0.769 x 0.995

%instructions
x (1 - Instruction H1)
= 0.003846 or 0.3846 %
0.769 x 0.005

Instruction L1 Hit:
Hit Access Time = 1
Stalls = 0

Ideal access on a hit, no stalls

Instruction L1 Miss:
Access Time = M + 1= 201
Stalls Per access = M = 200
Stalls = %instructions x (1 - Instruction H1) x M
= 0.003846 x 200 =0.7692 cycles

Data

0.231 x 0.94
% data x Data H1
= .2169 or 21.69 %

0.231 x 0.06
% data x (1 - Data H1)
= 0.01385 or 1.385 %

Data L1 Hit:
Hit Access Time: = 1
Stalls = 0

Ideal access on a hit, no stalls

Data L1 Miss:
Access Time = M + 1 = 201
Stalls per access: M = 200
Stalls = % data x (1 - Data H1) x M
= 0.01385 x 200 = 2.769 cycles

Stall Cycles Per Access = % Instructions x (1 - Instruction H1) x M + % data x (1 - Data H1) x M
= 0.7692 + 2.769 = 3.54 cycles

AMAT = 1 + Stall Cycles per access = 1 + 3.5 = 4.54 cycles

Stall cycles per instruction = (1 + fraction of loads/stores) x Stall Cycles per access = 1.3 x 3.54 = 4.6 cycles

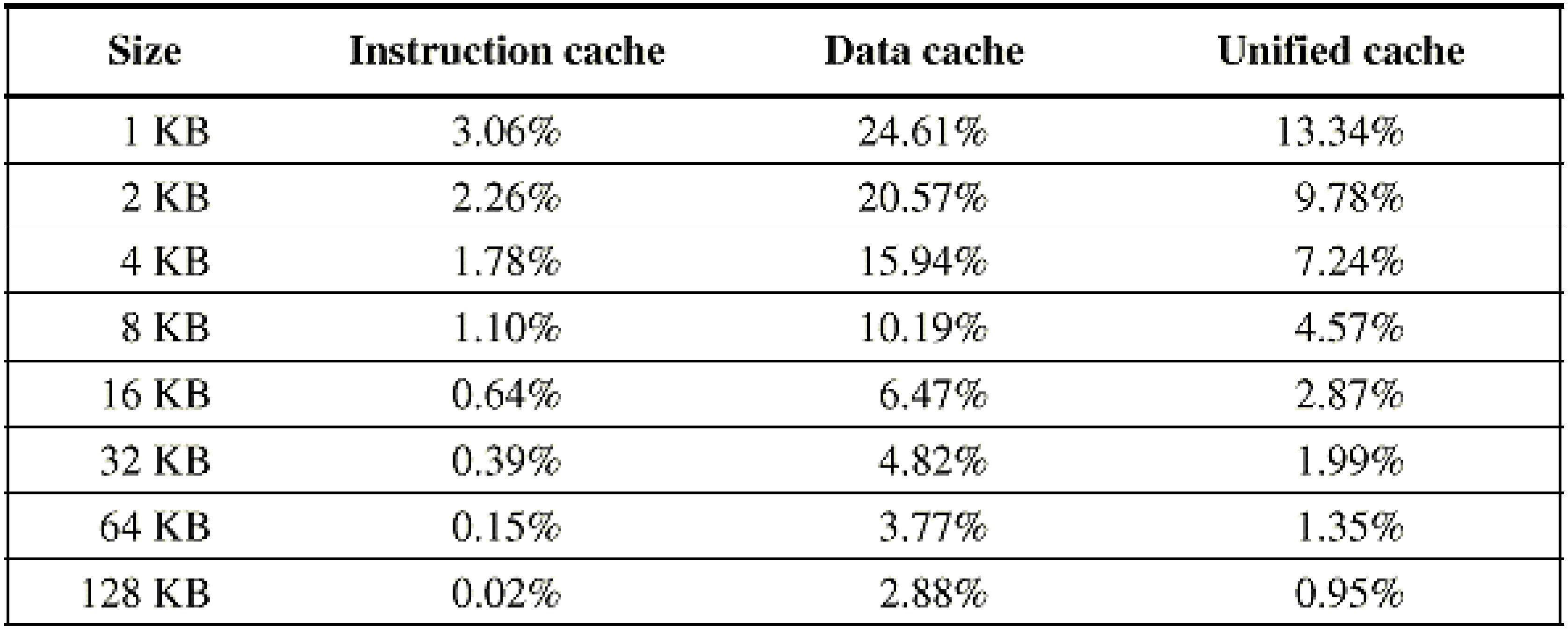
CPI = CPI$_{execution}$ + Stall cycles per instruction = 1.1 + 4.6 = 5.7

Given as 1.1

M = Miss Penalty = stall cycles per access resulting from missing in cache = 200 cycles
M + 1 = Miss Time = Main memory access time = 200+1 =201 cycles L1 access Time = 1 cycle
Data H1 = 0.94 or 94% 1- Data H1 = 0.06 or 6%
Instruction H1 = 0.995 or 99.5% 1- Instruction H1 = 0.005 or 0.5 %
% Instructions = Percentage or fraction of instruction fetches out of all memory accesses = 76.9 %
% Data = Percentage or fraction of data accesses out of all memory accesses = 23.1 %

Typical Cache Performance Data Using SPEC92

Usually: Data Miss Rate >> Instruction Miss Rate (for split cache)



Size	Instruction cache	Data cache	Unified cache
1 KB	3.06%	24.61%	13.34%
2 KB	2.26%	20.57%	9.78%
4 KB	1.78%	15.94%	7.24%
8 KB	1.10%	10.19%	4.57%
16 KB	0.64%	6.47%	2.87%
32 KB	0.39%	4.82%	1.99%
64 KB	0.15%	3.77%	1.35%
128 KB	0.02%	2.88%	0.95%
	1 – Instruction H1	**1 – Data H1**	**1 – H1**

Miss rates for instruction, data, and unified caches of different sizes.

Program steady state cache miss rates are given
Initially cache is empty and miss rates ~ 100%