



EECS 318 CAD Computer Aided Design

LECTURE 10: Improving Memory Access: Direct and Spatial caches

Instructor: Francis G. Wolff
wolff@eecs.cwrn.edu

Case Western Reserve University

This presentation uses powerpoint animation: please viewshow

The Art of Memory System Design



*Optimize the memory system organization
to minimize the average memory access time
for typical workloads*

Workload or
Benchmark
programs

Processor

reference stream

$\langle \text{op}, \text{addr} \rangle, \langle \text{op}, \text{addr} \rangle, \langle \text{op}, \text{addr} \rangle, \langle \text{op}, \text{addr} \rangle, \dots$

op: i-fetch, read, write

Memory

\$

MEM

Pipelining and the cache (Designing..., M.J.Quinn, '87)



Instruction Pipelining is the use of pipelining to allow more than one instruction to be in some stage of execution at the same time.

Ferranti ATLAS (1963):

- Pipelining reduced the average time per instruction by 375%
- Memory could not keep up with the CPU, needed a cache.

Cache memory is a small, fast memory unit used as a buffer between a processor and primary memory

Principle of Locality



- **Principle of Locality**

states that programs access a relatively small portion of their address space at any instance of time

- Two types of locality

- Temporal locality (locality in time)

If an item is referenced, then

the same item will tend to be referenced soon

“the tendency to reuse recently accessed data items”

- Spatial locality (locality in space)

If an item is referenced, then

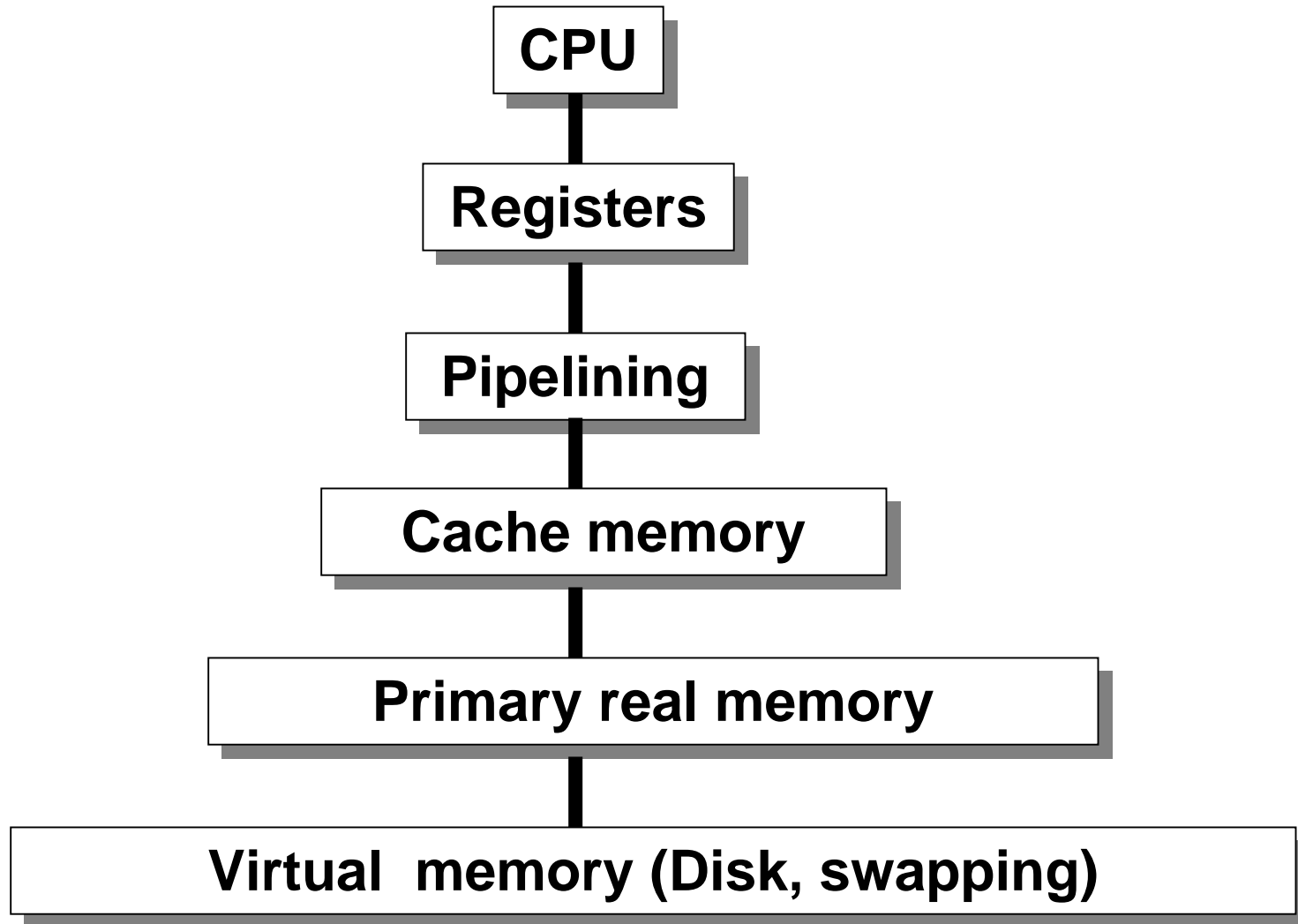
nearby items will be referenced soon

“the tendency to reference nearby data items”

Memory Hierarchy



Faster

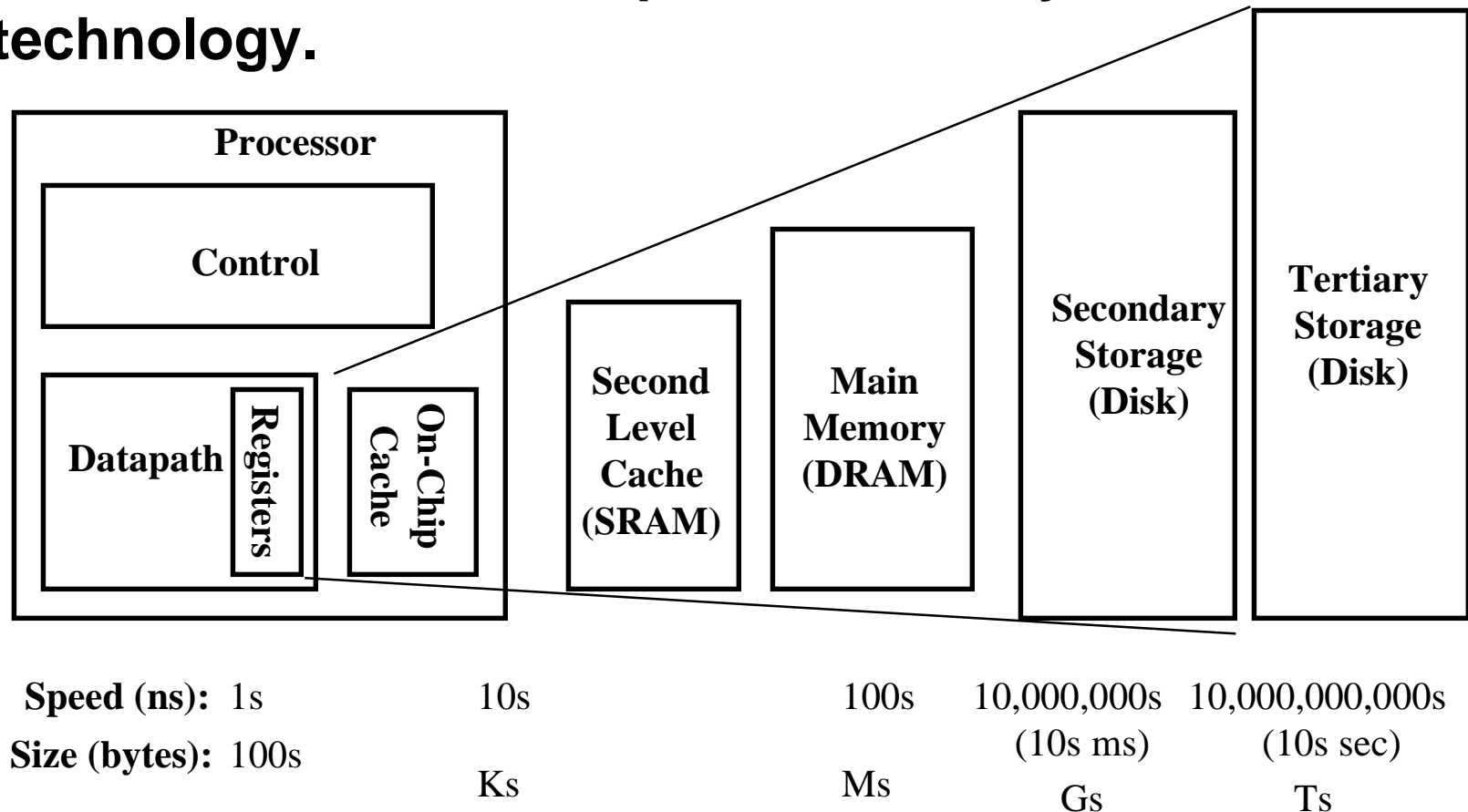


Cheaper Cost \$\$\$

More Capacity

Memory Hierarchy of a Modern Computer System

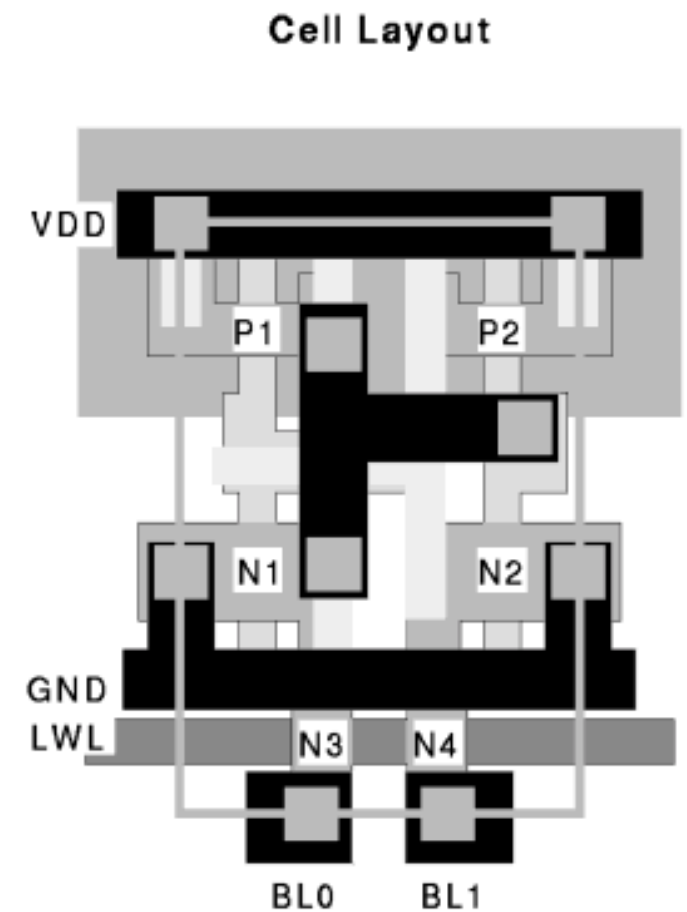
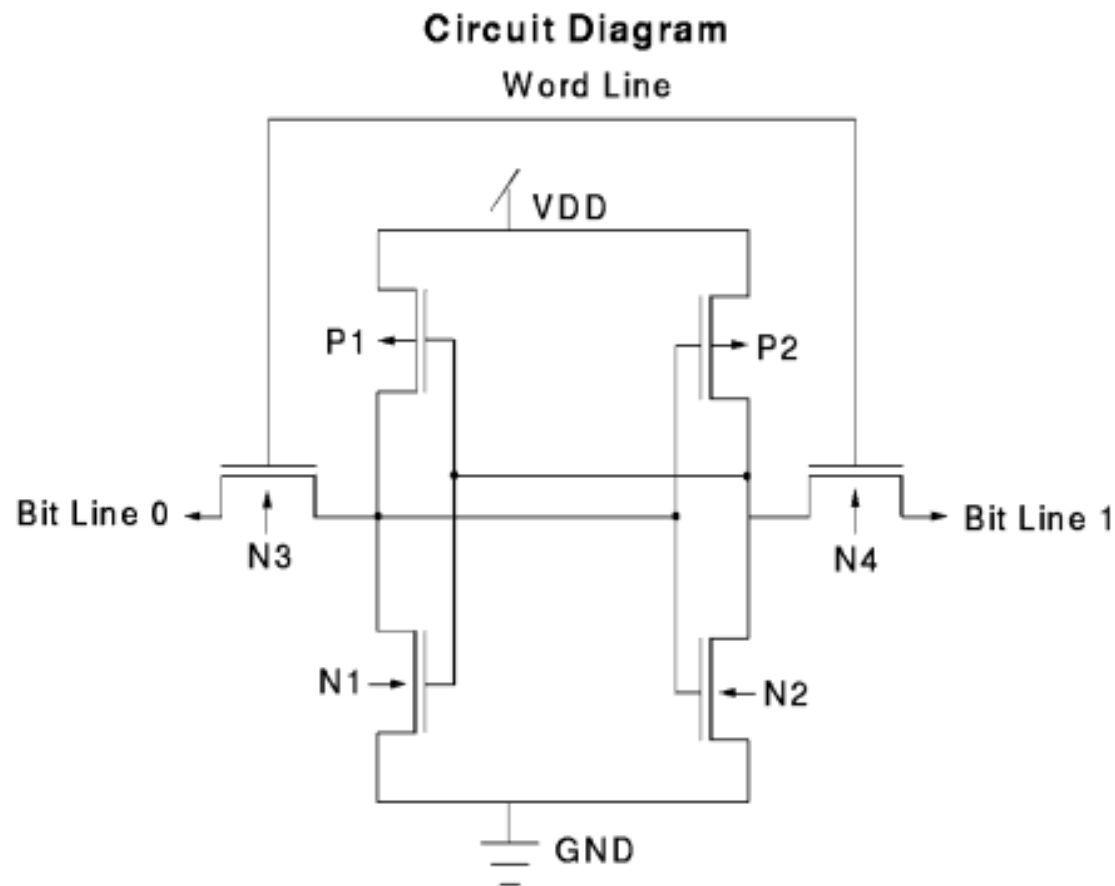
- By taking advantage of the principle of locality:
 - Present the user with as much memory as is available in the cheapest technology.
 - Provide access at the speed offered by the fastest technology.



Cache Memory Technology: SRAM 1 bit cell layout

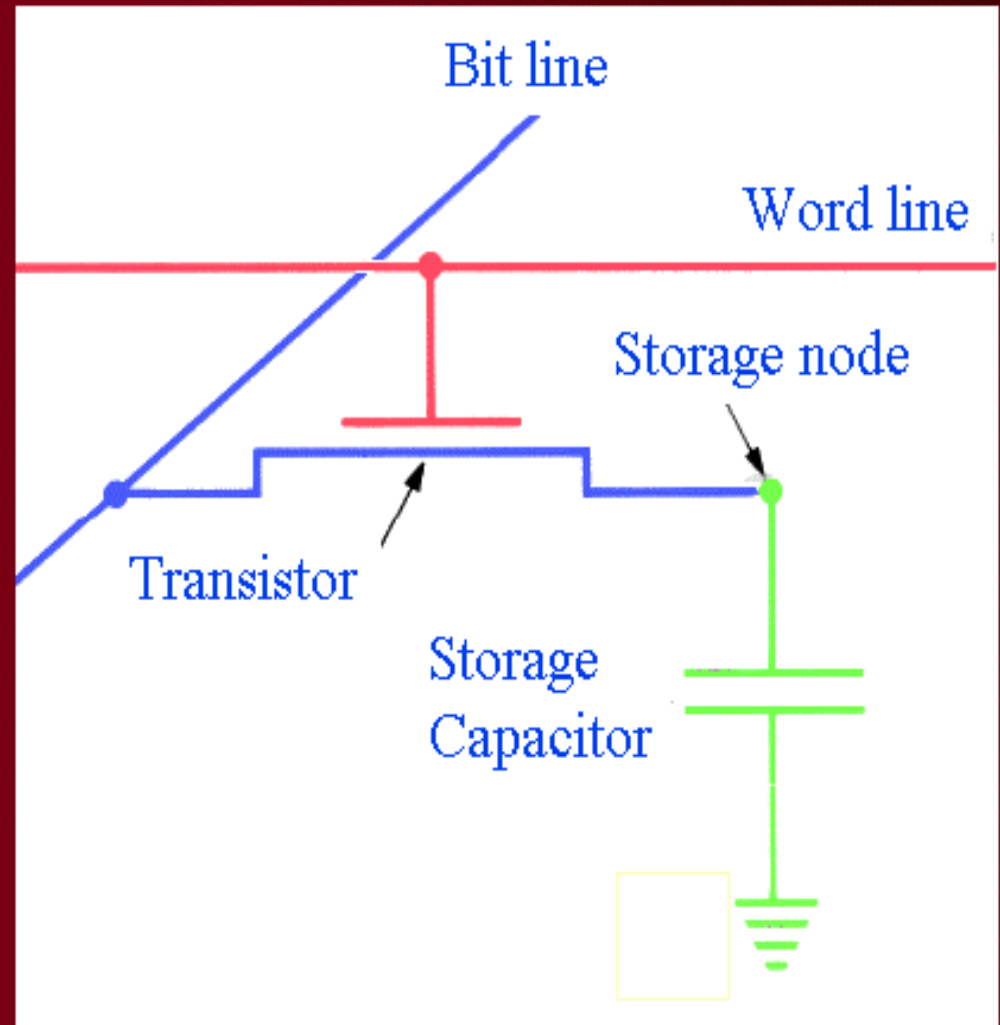


Figure 3. IBM's 6-Transistor Memory Cell



Basic DRAM design

- DRAM replaces all but one transistors of flip-flop with a capacitor
- => smaller!
- Capacitor stores information
- Charge leakage requires periodic refreshment (sense & rewrite)



Memories Technology and Principle of Locality



- **Faster Memories are more expensive per bit**
- **Slower Memories are usually smaller in area size per bit**

Memory Technology	Typical access time	\$ per Mbyte in 1997
SRAM	5-25 ns	\$100-\$250
DRAM	60-120 ns	\$5-\$10
Magnetic Disk	10-20 million ns	\$0.10-\$0.20

Cache Memory Technology: SRAM



- Why use SRAM (Static Random Access Memory)?

- **Speed.**

The primary advantage of an SRAM over DRAM is speed.

The fastest DRAMs on the market still require **5 to 10 processor clock cycles** to access the first bit of data.

SRAMs can operate at processor speeds of 250 MHz and beyond, with access and **cycle times equal to the clock cycle** used by the microprocessor

- **Density.**

when 64 Mb DRAMs are rolling off the production lines, the largest SRAMs are expected to be only 16 Mb.

see reference: <http://www.chips.ibm.com/products/memory/sramoperations/sramop.html>

Cache Memory Technology: SRAM

(con't)



- **Volatility.**

Unlike DRAMs, SRAM cells do not need to be refreshed.
SRAMs are available 100% of the time for reading & writing.

- **Cost.**

If cost is the primary factor in a memory design,
then DRAMs win hands down.

If, on the other hand, performance is a critical factor,
then a well-designed SRAM is an effective cost
performance solution.

Memory Hierarchy of a Modern Computer System



- **By taking advantage of the principle of locality:**
 - Present the user with as much memory as is available in the cheapest technology.
 - Provide access at the speed offered by the fastest technology.
- **DRAM is slow but cheap and dense:**
 - Good choice for presenting the user with a BIG memory system
- **SRAM is fast but expensive and not very dense:**
 - Good choice for providing the user FAST access time.

Cache Terminology



A hit if the data requested by the CPU is in the upper level

Hit rate or **Hit ratio**

is the fraction of accesses found in the upper level

Hit time

is the time required to access data in the upper level

= <detection time for hit or miss> + <hit access time>

A miss if the data is not found in the upper level

Miss rate or **(1 – hit rate)**

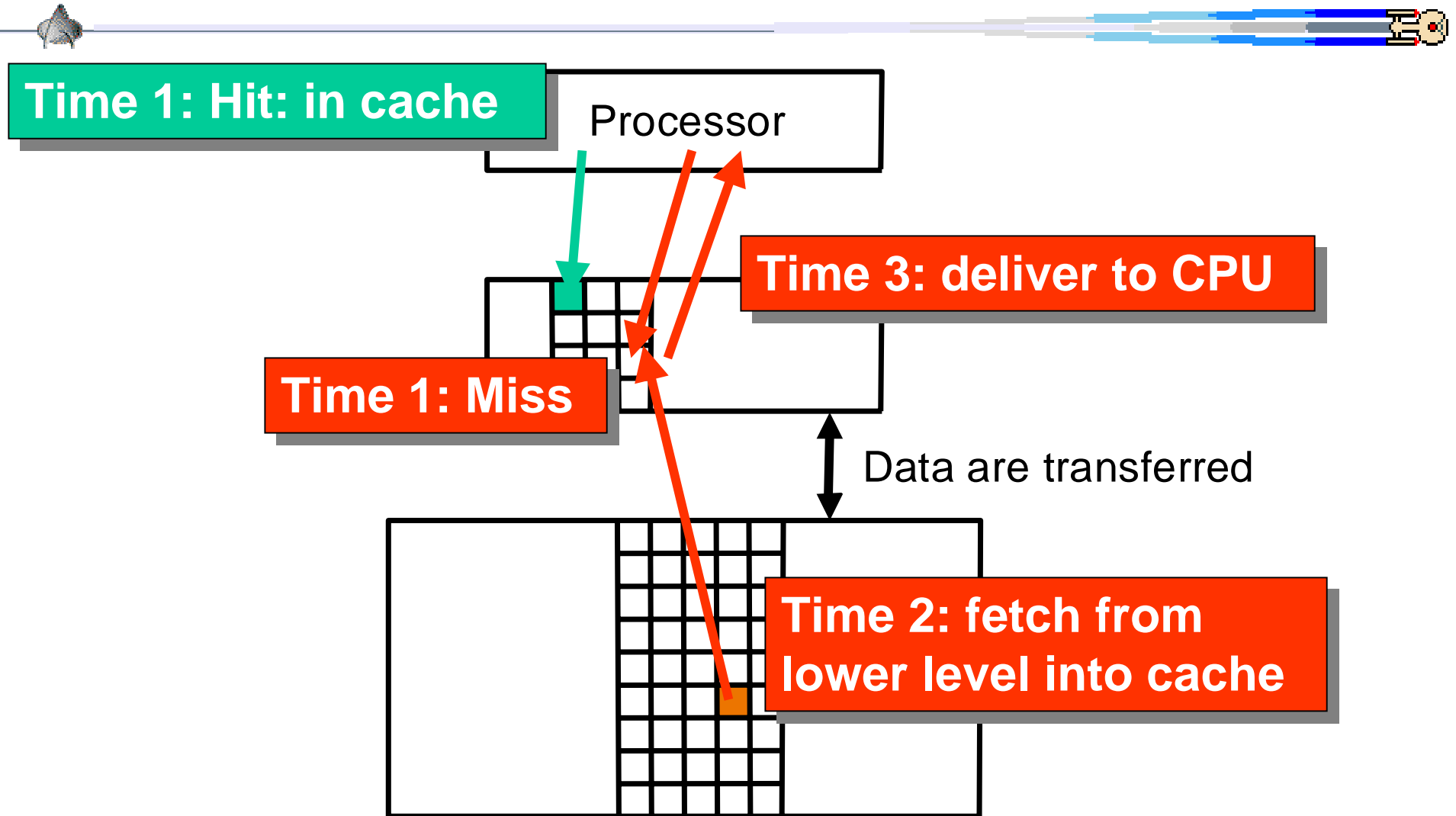
is the fraction of accesses not found in the upper level

Miss penalty

is the time required to access data in the lower level

= <lower access time> + <reload processor time>

Cache Example

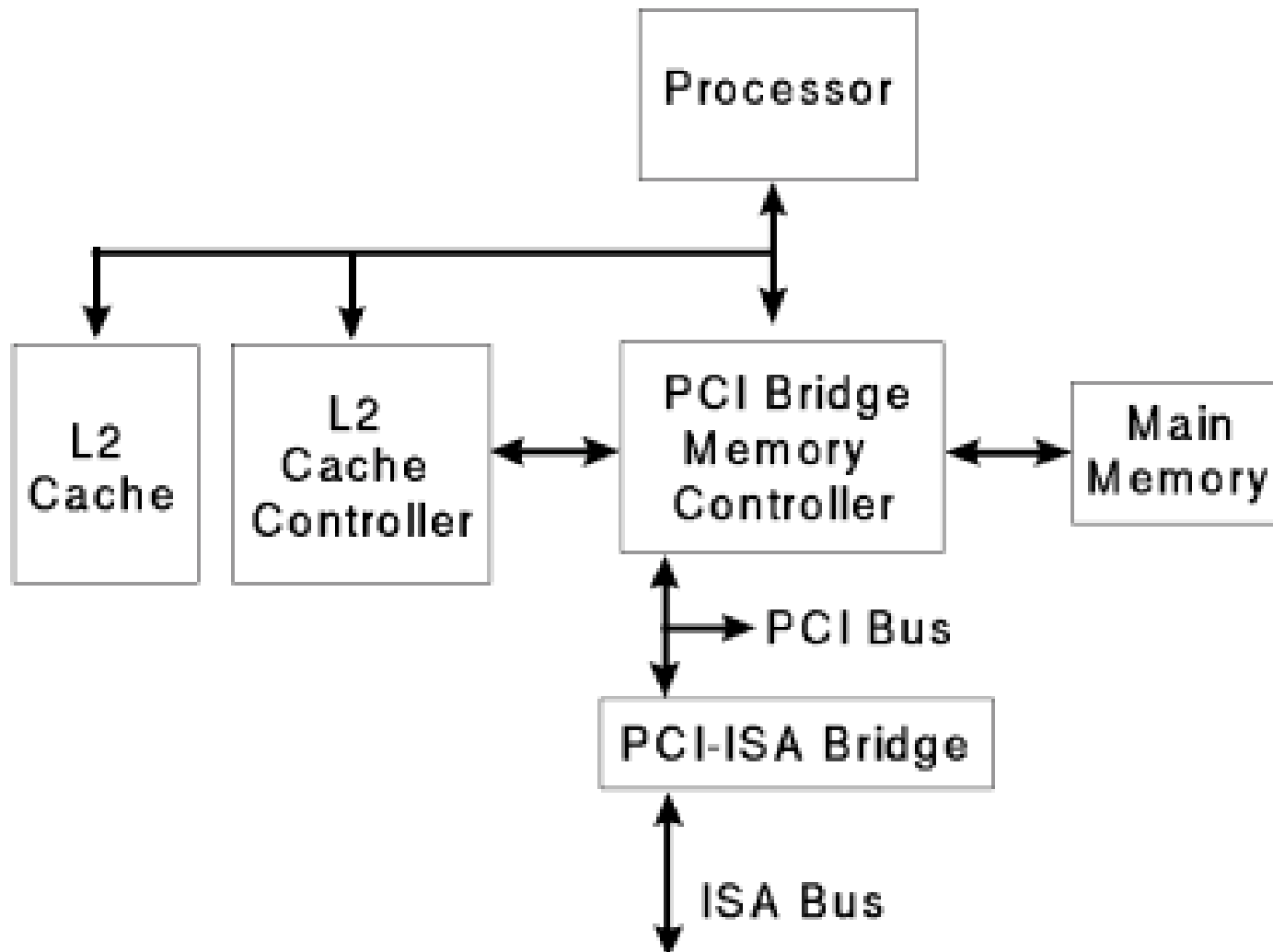


Hit time = Time 1

Miss penalty = Time 2 + Time 3

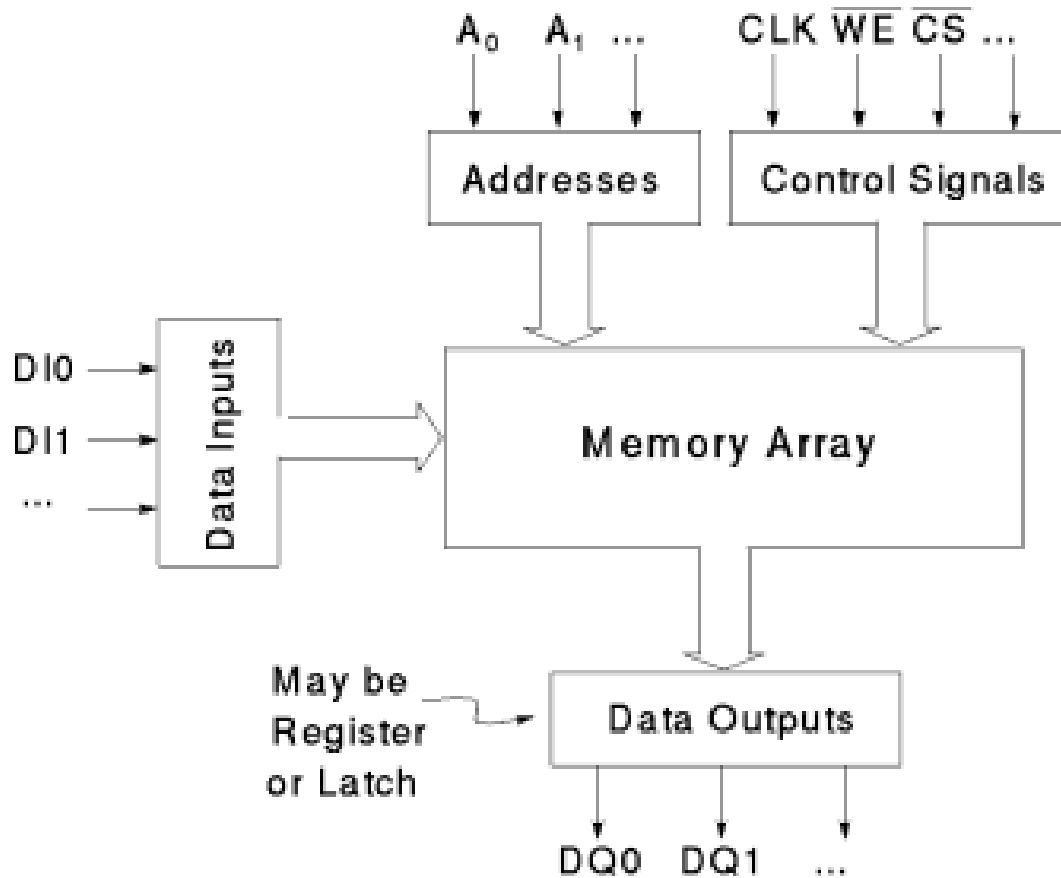
Basic Cache System

Figure 1. Basic Cache System



Cache Memory Technology: SRAM Block diagram

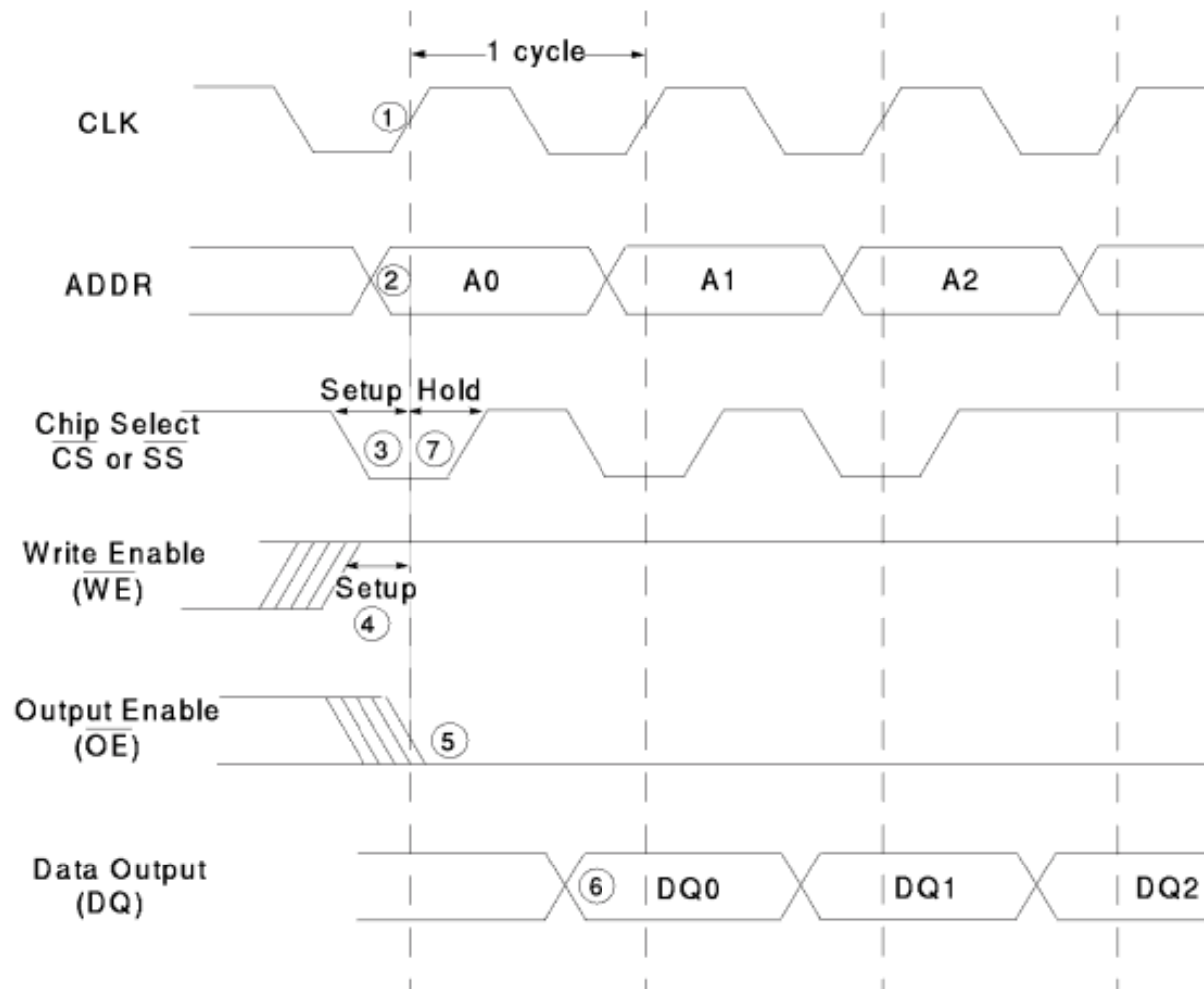
Figure 2. Simplified Block Diagram of a Synchronous SRAM



Cache Memory Technology: SRAM timing diagram



Figure 4. Reading from Memory (Flow Thru mode)

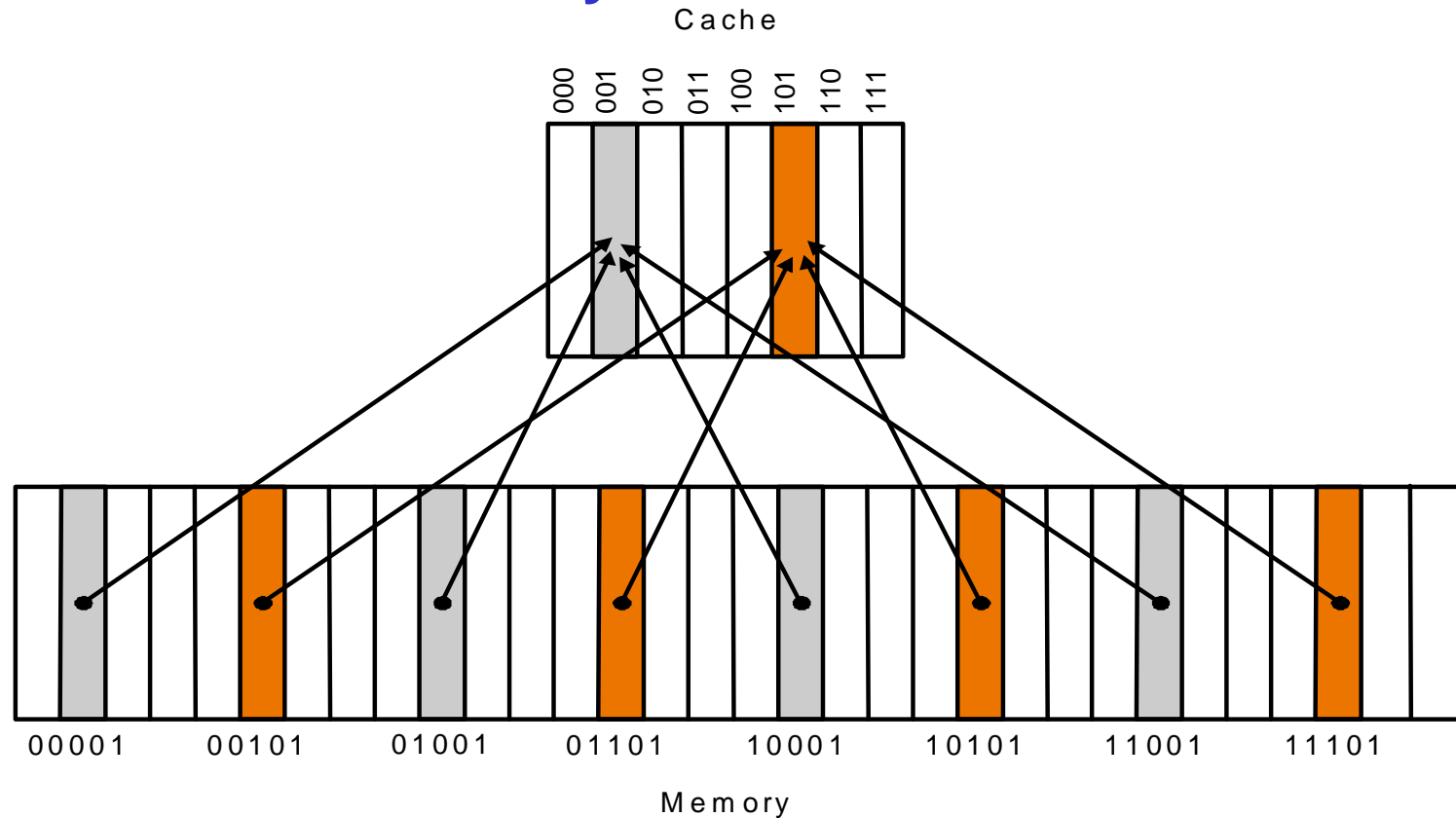


Note: DQ0 is the data associated with Address 0 (A0). DQ1 is the data associated with Address 1 (A1).

Direct Mapped Cache



- **Direct Mapped:** assign the cache location based on the address of the word in memory
- **cache_address = memory_address modulo cache_size;**



Observe there is a **Many-to-1** memory to cache relationship

Direct Mapped Cache: Data Structure



There is a **Many-to-1 relationship** between memory and cache

How do we know whether the data in the cache corresponds to the requested word?

tags

- contain the address information required **to identify** whether a word in the cache corresponds to the requested word.
- tags need only to contain the **upper portion** of the memory address (often referred to as a **page address**)

valid bit

- indicates whether an entry contains a valid address

Direct Mapped Cache: Temporal Example



lw \$1, 10 110 (\$0) Miss: valid
lw \$2, 11 010 (\$0) Miss: valid
lw \$3, 10 110 (\$0) Hit!

lw \$1, 22 (\$0)
lw \$2, 26 (\$0)
lw \$3, 22 (\$0)

Index	Valid	Tag	Data
000	N		
001	N		
010	Y	11	Memory[11010]
011	N		
100	N		
101	N		
110	Y	10	Memory[10110]
111	N		

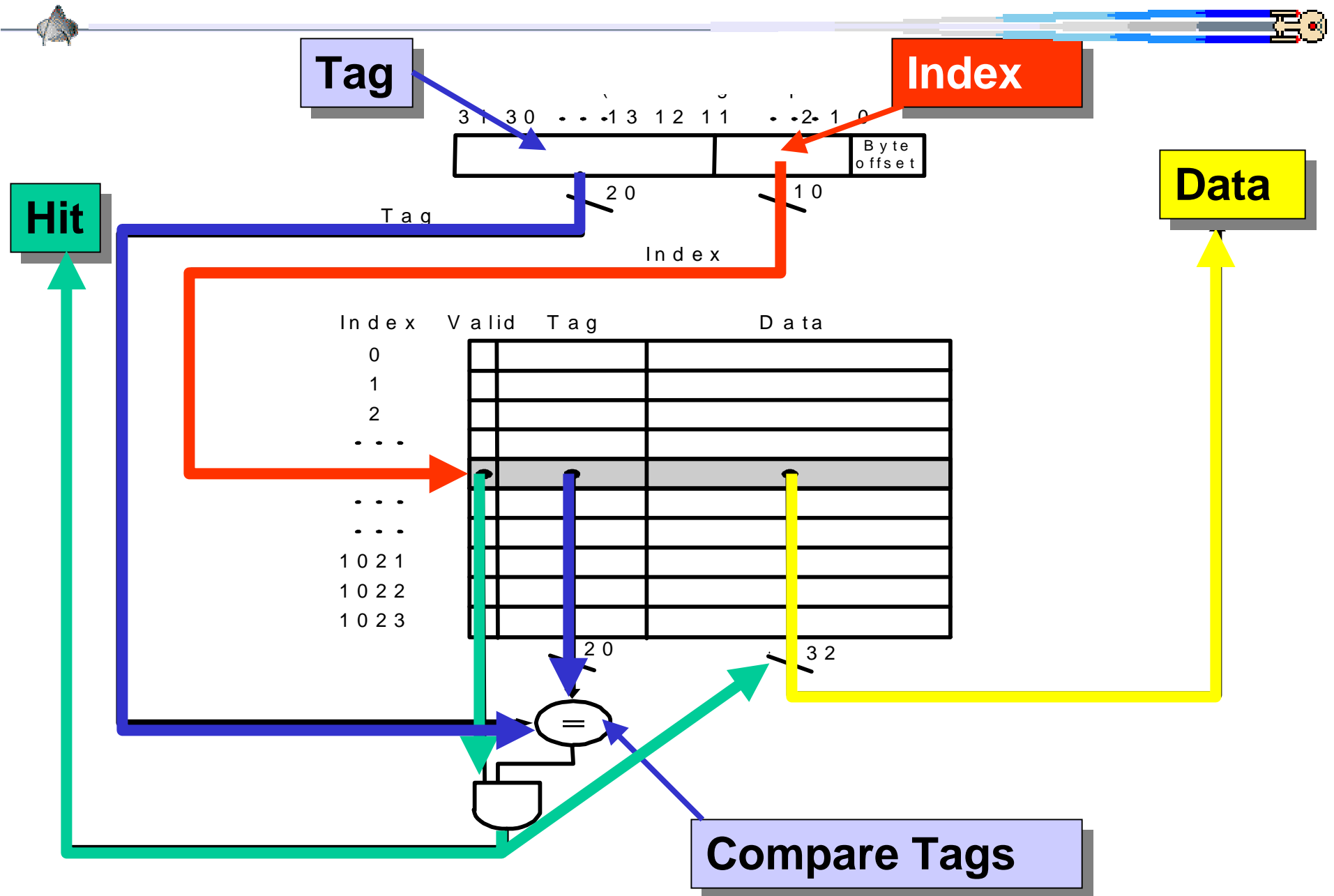
Direct Mapped Cache: Worst case, always miss!




lw \$1,10 110 (\$0) **Miss: valid** lw \$1,22(\$0)
lw \$2,11 110 (\$0) **Miss: tag** lw \$2,30(\$0)
lw \$3,00 110 (\$0) **Miss: tag** lw \$3,6(\$0)

Index	Valid	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	00	Memory[00110]
111	N		

Direct Mapped Cache: Mips Architecture



Bits in a Direct Mapped Cache

 How many total bits are required for a direct mapped cache with 64KB ($= 2^{16}$ KiloBytes) of data and one word ($= 32$ bit) blocks assuming a 32 bit byte memory address?

Cache index width $= \log_2 \text{ words}$
 $= \log_2 2^{16}/4 = \log_2 2^{14} \text{ words} = 14 \text{ bits}$

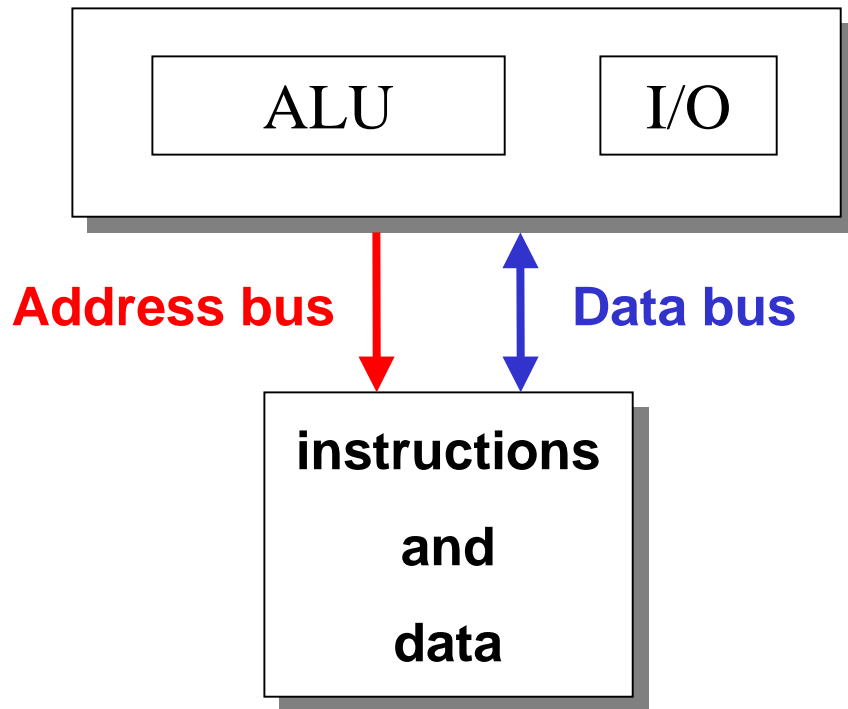
Block address width $= \langle \text{byte address width} \rangle - \log_2 \text{ word}$
 $= 32 - 2 = 30 \text{ bits}$

Tag size $= \langle \text{block address width} \rangle - \langle \text{cache index width} \rangle$
 $= 30 - 14 = 16 \text{ bits}$

Cache block size $= \langle \text{valid size} \rangle + \langle \text{tag size} \rangle + \langle \text{block data size} \rangle$
 $= 1 \text{ bit} + 16 \text{ bits} + 32 \text{ bits} = 49 \text{ bits}$

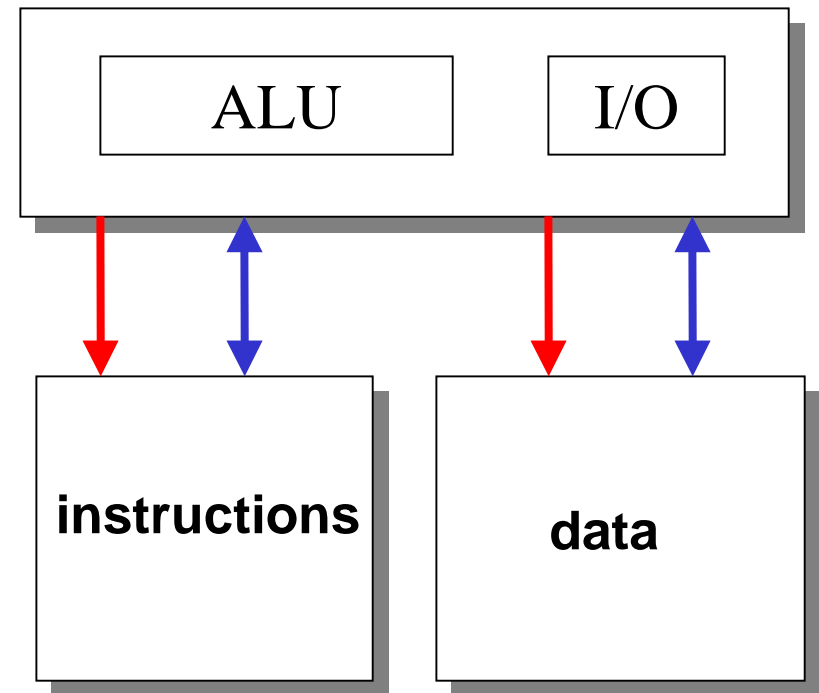
Total size $= \langle \text{Cache word size} \rangle \times \langle \text{Cache block size} \rangle$
 $= 2^{14} \text{ words} \times 49 \text{ bits} = 784 \times 2^{10} = 784 \text{ Kbits} = 98 \text{ KB}$
 $= 98 \text{ KB}/64 \text{ KB} = 1.5 \text{ times overhead}$

Split Cache: Exploiting the Harvard Architectures



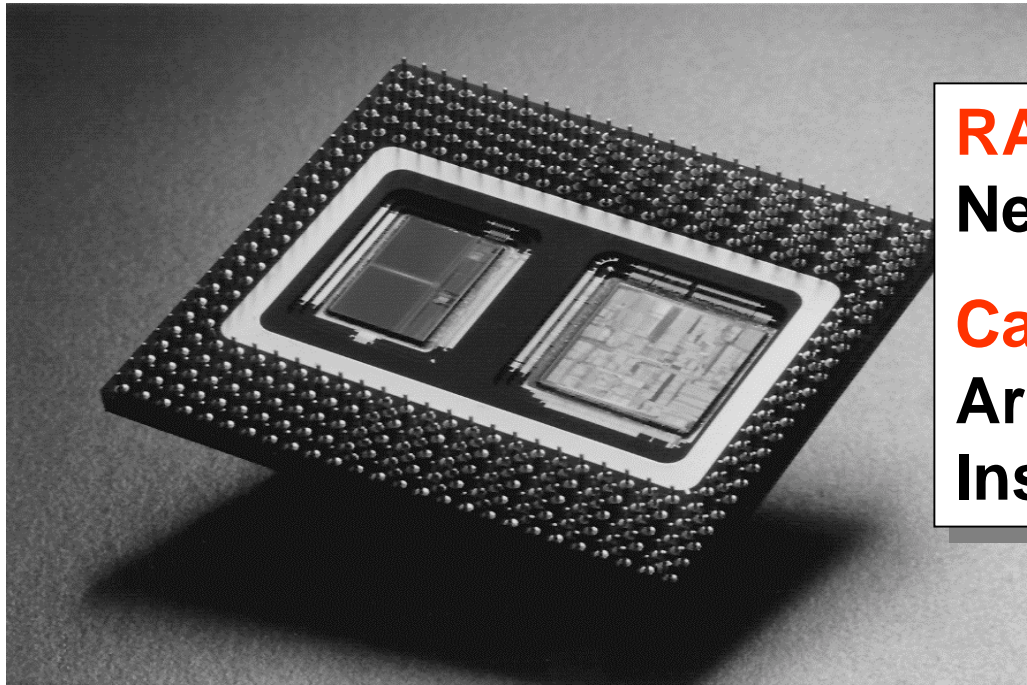
Von Neuman architecture

Area efficient but requires higher bus bandwidth because instructions and data must compete for memory.



Harvard architecture was coined to describe machines with separate memories. **Speed efficient:** Increased parallelism (split cache).

Modern Systems: Pentium Pro and PowerPC



RAM (main memory) : von Neuman Architecture

Cache: uses Harvard Architecture separate Instruction/Data caches

Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split instruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through

Cache schemes



write-through cache

Always write the data into both the cache and memory and then wait for memory.

write buffer

write data into cache and write buffer.
If write buffer full processor must stall.

**No amount of buffering can help
if writes are being generated faster
than the memory system can accept them.**

write-back cache

Write data into the cache block and
only write to memory when block is modified
but complex to implement in hardware.

Chip Area

Speed



Hits vs. Misses



- Read hits
 - this is what we want!
- Read misses
 - stall the CPU, fetch block from memory, deliver to cache, and restart.
- Write hits
 - write-through: can replace data in cache and memory.
 - write-buffer: write data into cache and buffer.
 - write-back: write the data only into the cache.
- Write misses
 - read the entire block into the cache, then write the word.

Example: The DECStation 3100 cache



DECStation uses a [write-through](#) harvard architecture cache

- 128 KB total cache size (=32K words)
 - = 64 KB instruction cache (=16K words)
 - + 64 KB data cache (=16K words)
- 10 processor clock cycles to write to memory

The DECStation 3100 miss rates

- A split instruction and data cache increases the bandwidth

Benchmark Program	gcc	spice
Instruction miss rate	6.1%	1.2%
Data miss rate	2.1%	1.3%
Effective split miss rate	5.4%	1.2%
Combined miss rate	4.8%	

1.2% miss, also means that 98.2% of the time it is in the cache. So using a cache pays off!

Why a lower miss rate?

Numerical programs tend to consist of a lot of small program loops

split cache has slightly worse miss rate

Review: Principle of Locality



- **Principle of Locality**

states that programs access a relatively small portion of their address space at any instance of time

- Two types of locality

- Temporal locality (locality in time)

If an item is referenced, then

the same item will tend to be referenced soon

“the tendency to reuse recently accessed data items”

- Spatial locality (locality in space)

If an item is referenced, then

nearby items will be referenced soon

“the tendency to reference nearby data items”

Spatial Locality



- Temporal only cache

cache block contains only one word (**No spatial locality**).

- Spatial locality

Cache block contains multiple words.

- When a miss occurs, then fetch multiple words.

- Advantage

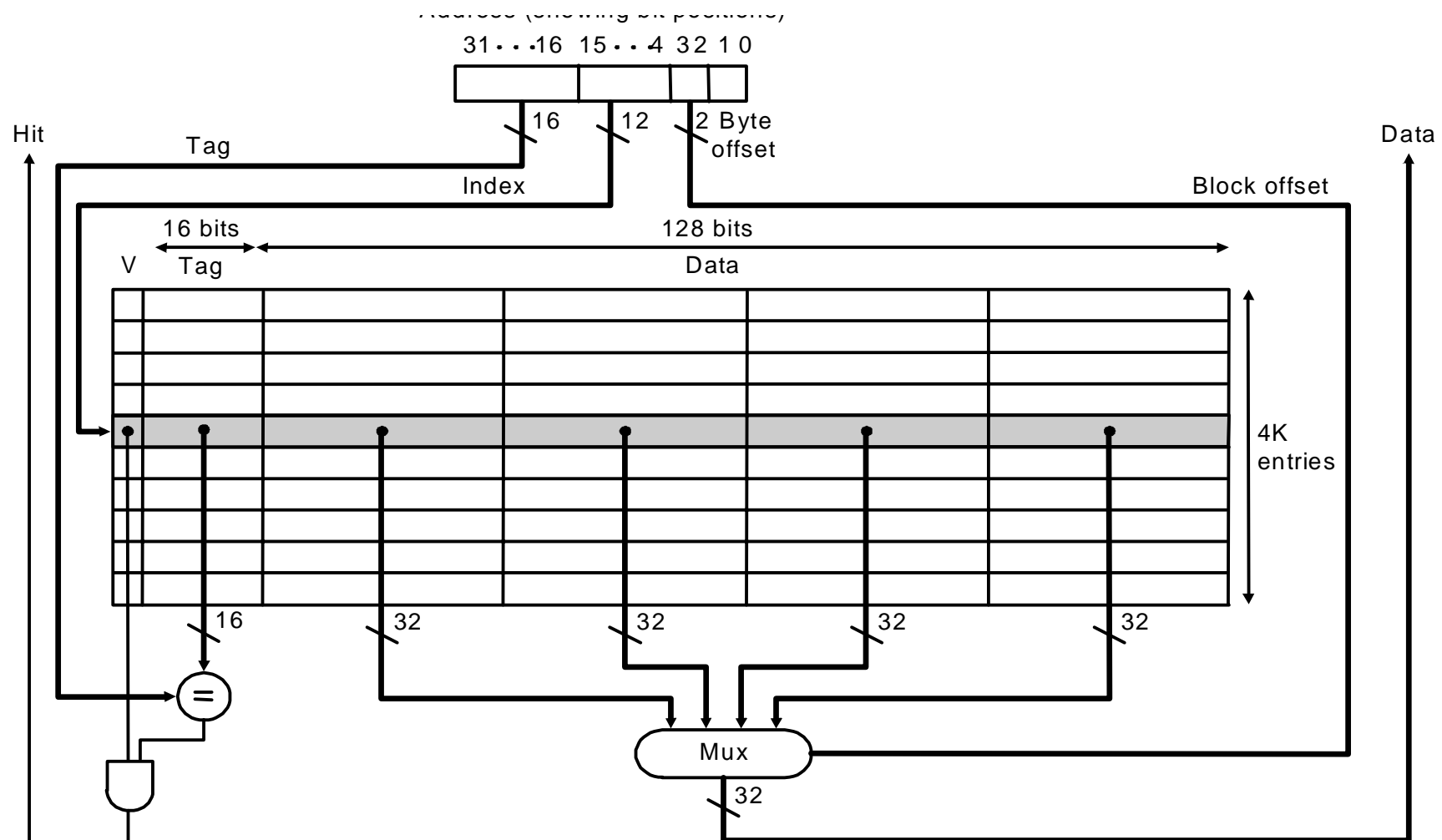
Hit ratio increases because there is a **high probability** that the adjacent words will be needed shortly.

- Disadvantage

Miss penalty increases with block size

Spatial Locality: 64 KB cache, 4 words

- 64KB cache using four-word (16-byte word)
- 16 bit tag, 12 bit index, 2 bit block offset, 2 bit byte offset.



Performance



- Use split caches because there is more spatial locality in code:

Program Block size	gcc =1	gcc =4	spice =1	spice =4
Instruction miss rate	6.1%	2.0%	1.2%	0.3%
Data miss rate	2.1%	1.7%	1.3%	0.6%
Effective split miss rate	5.4%	1.9%	1.2%	0.4%
Combined miss rate	4.8%	4.8%		

**Temporal only split cache:
has slightly worse miss rate**

**Spatial split cache: has
lower miss rate**

Cache Block size Performance



- Increasing the block size tends to decrease miss rate:

