Computer Organization & Design

Hardware/Software interface

Lou Xueqing

玉泉校区曹光彪东楼507室

Website:10.214.47.99/index.php
Email: xqlou@zju.edu.cn
hzlou@163.com

浙江大学计算机学院&软件学院

Chapter 7: Memory Hierarchy

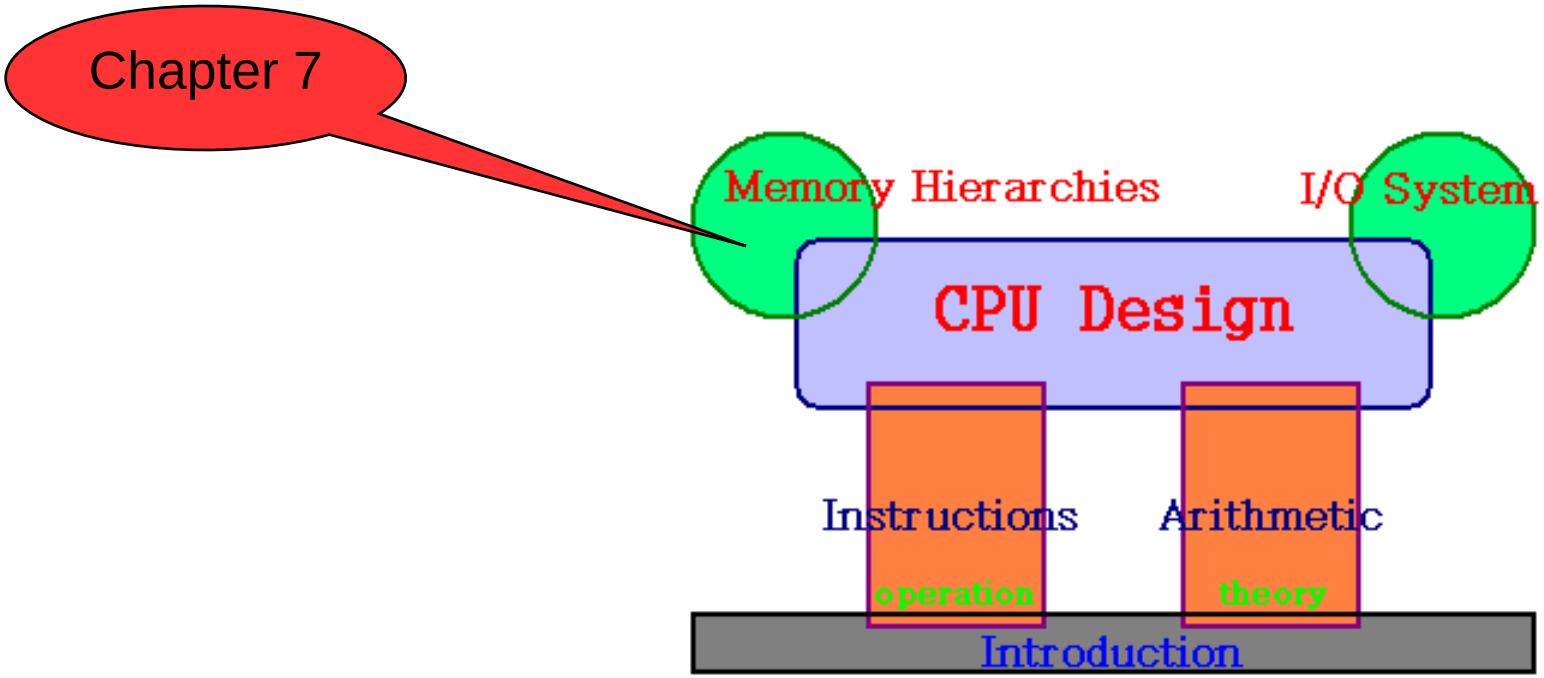
Lou Xueqing

Computer Organization & Design

浙江大学计算机学院&软件学院

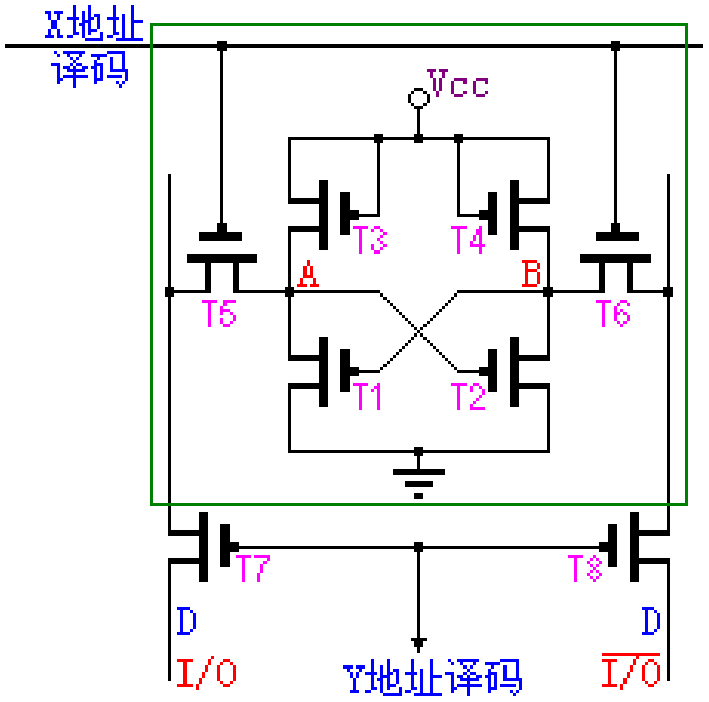
Chapter 7

- Topics: Memory Hierarchy



Memories: Review

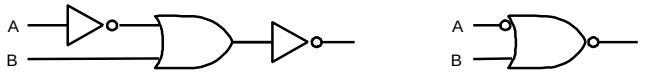
- SRAM:
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)

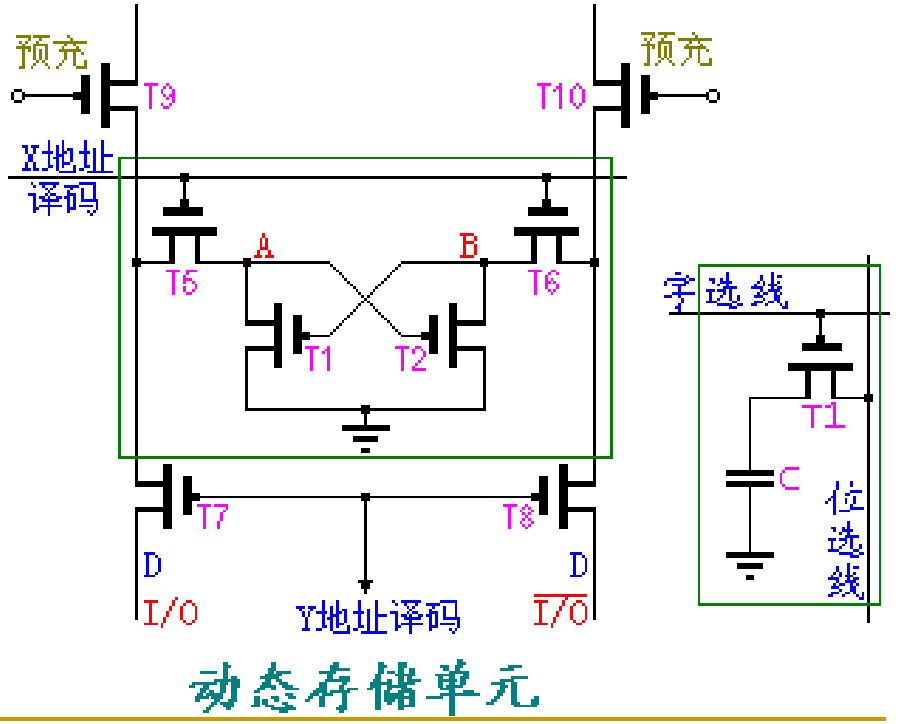


六管静态存储单元

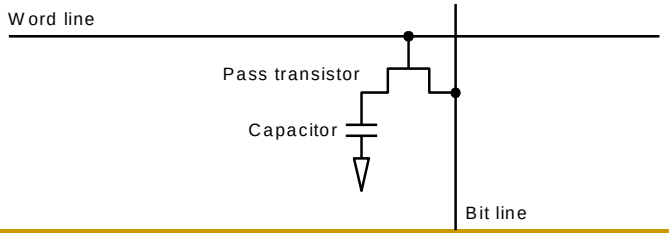
Memories: Review

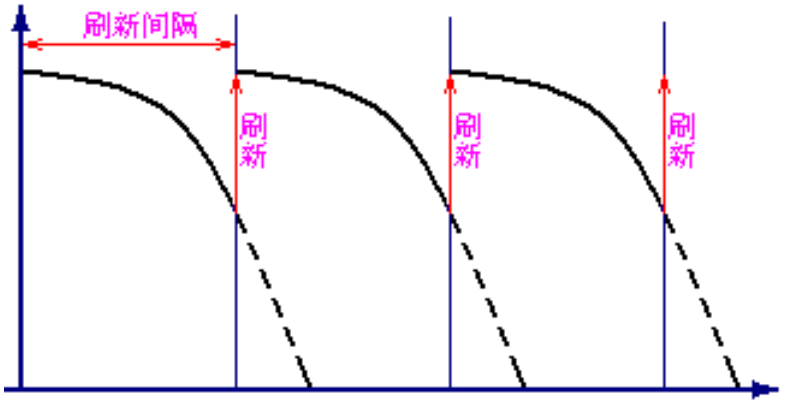
- DRAM:
  - value is stored as a charge on capacitor (must be refreshed)
  - very small but slower than SRAM (factor of 5 to 10)

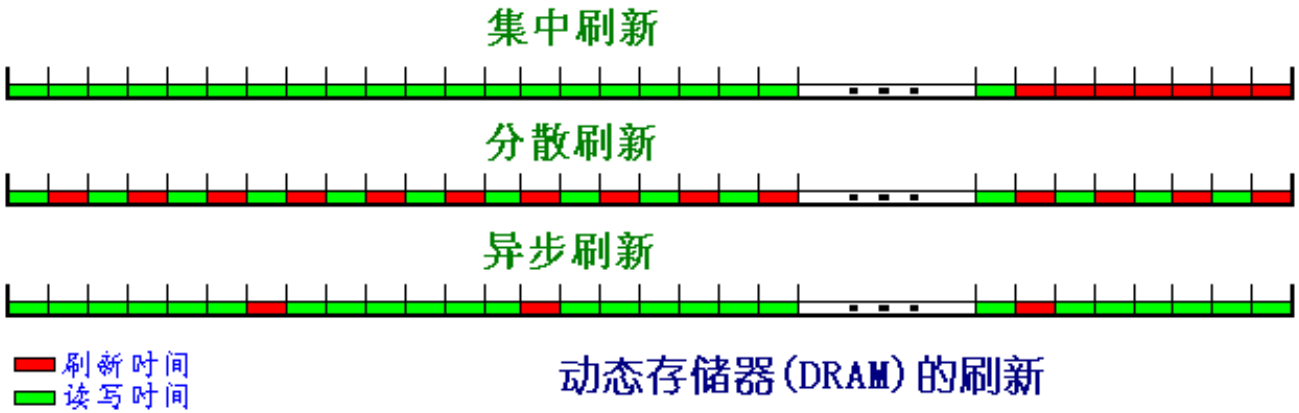




动态存储单元

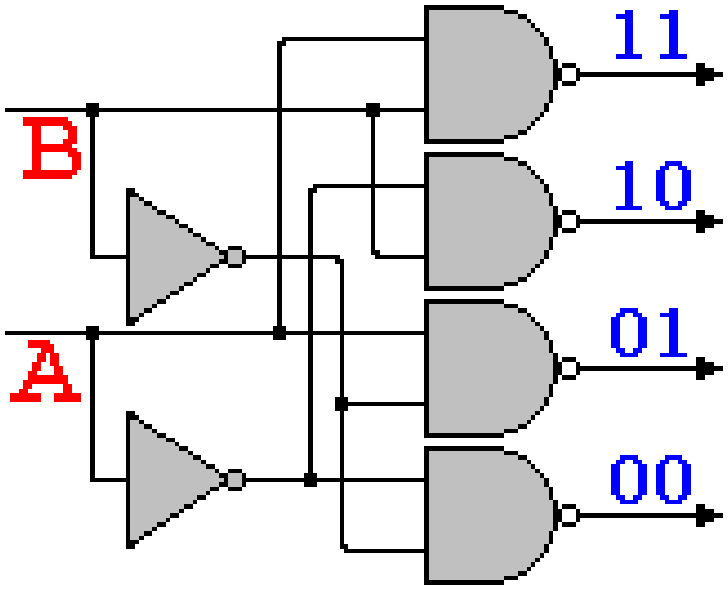


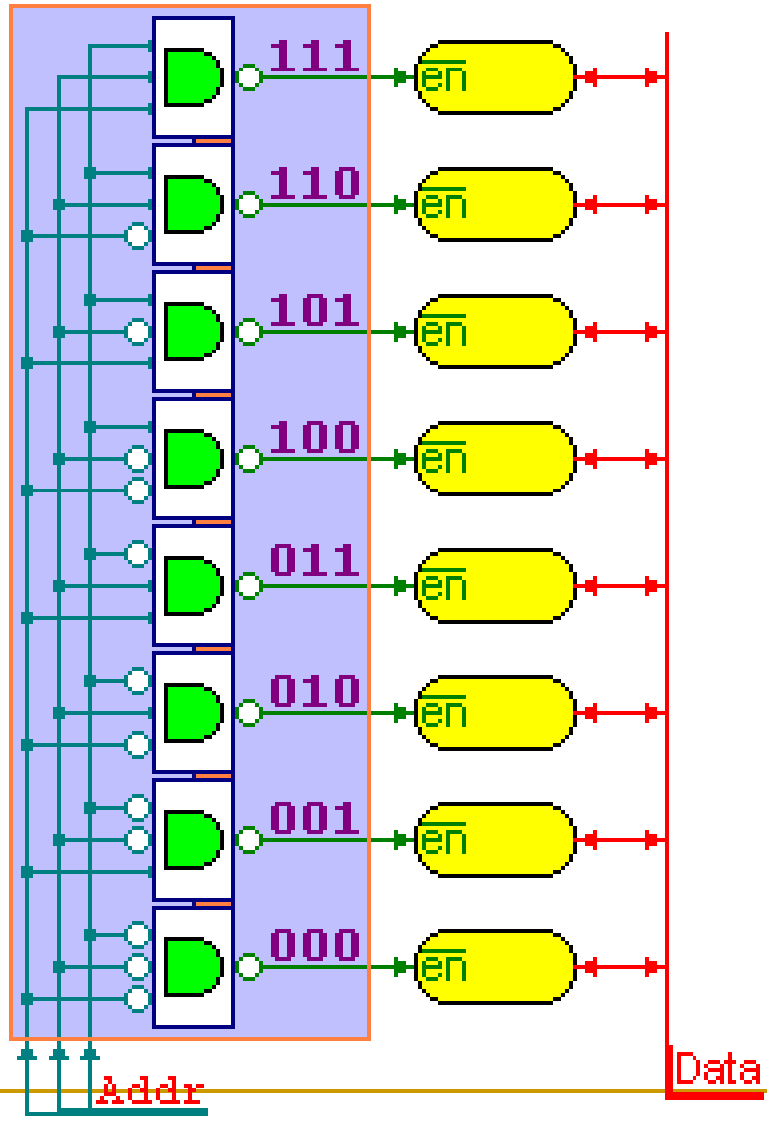


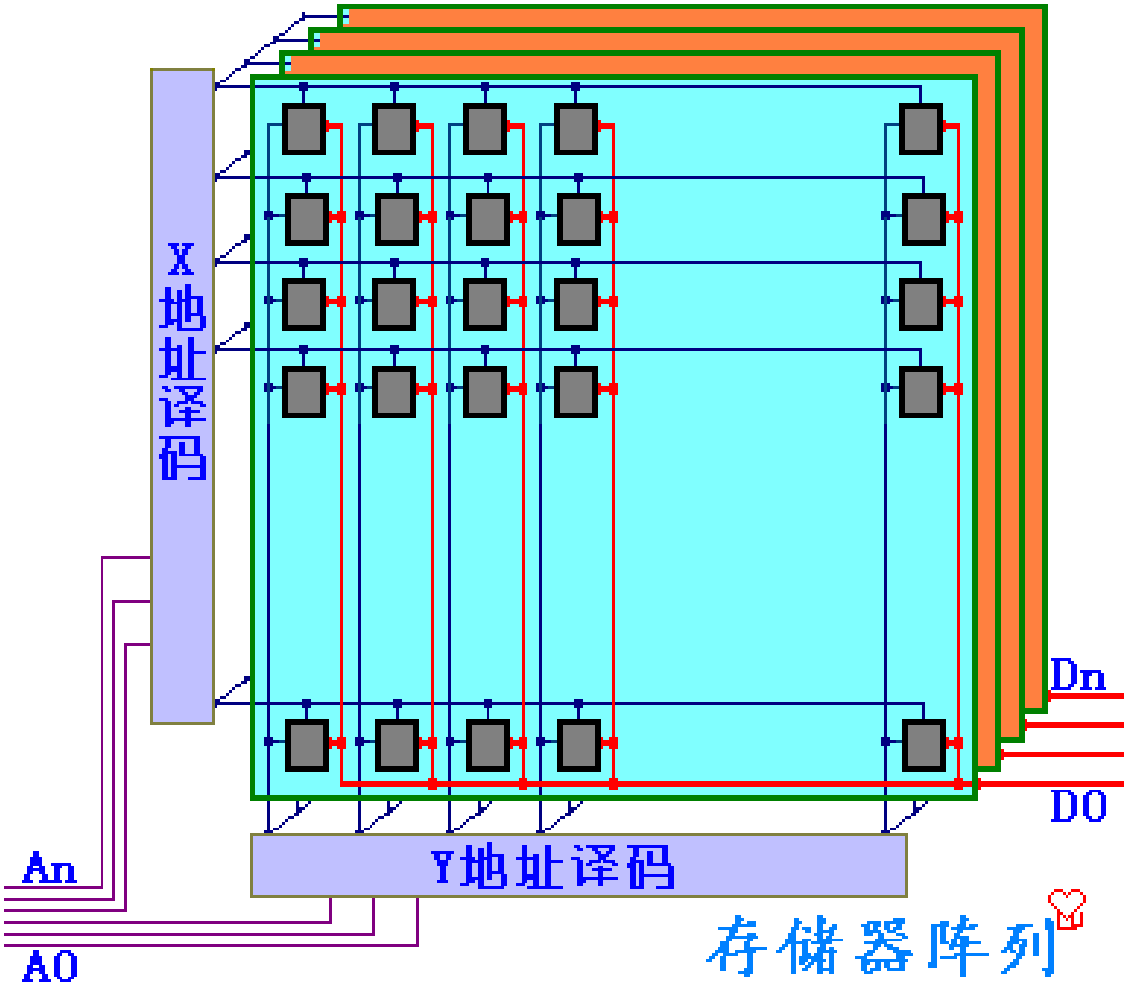


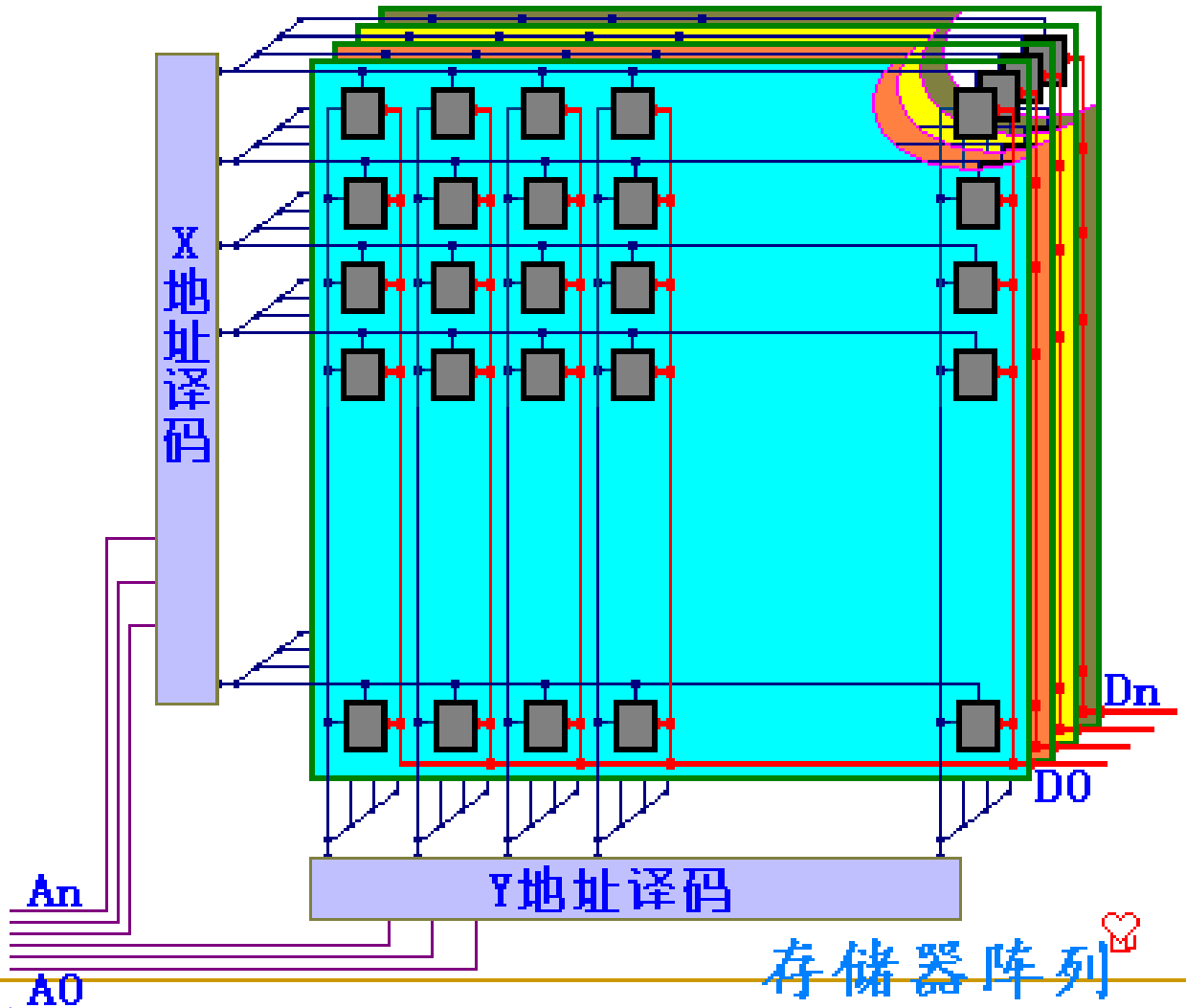
动态存储器(DRAM)的刷新

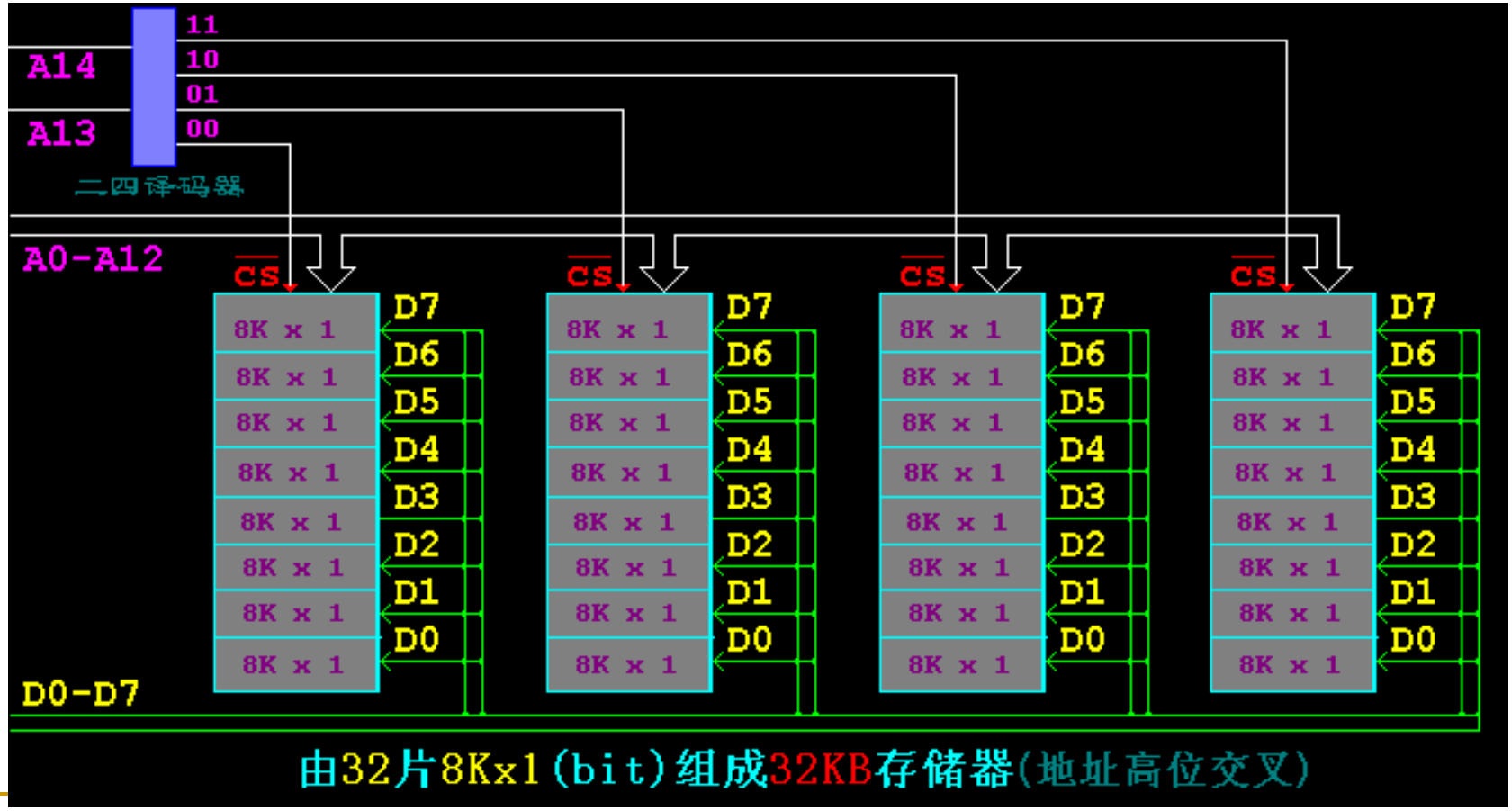
Decode: Addressing

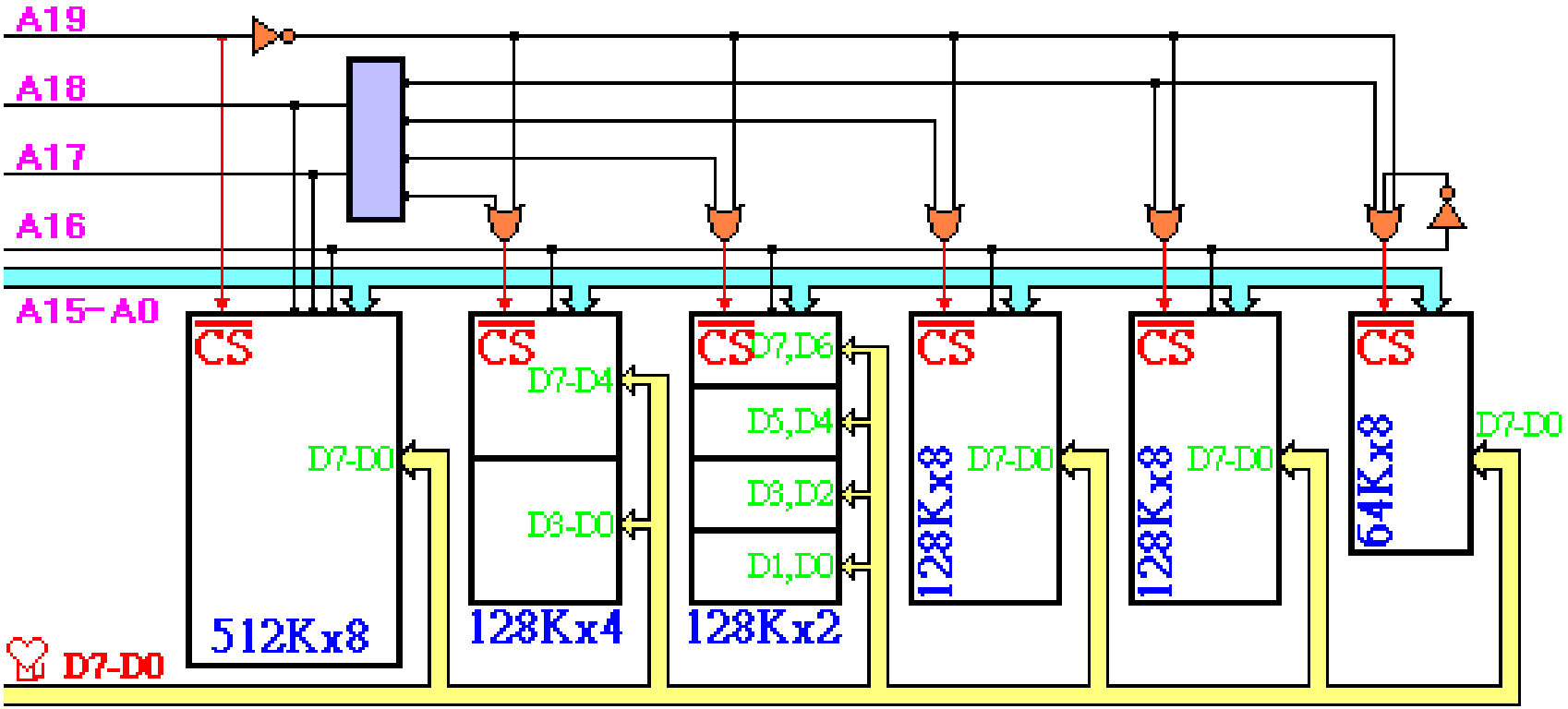





X
地
址
译
码
Dn
D0
An
A0
Y地址译码
存储器阵列


X
地
址
译
码
Dn
D0
An
A0
Y地址译码
存储器阵列


A14
A13
11
10
01
00
二四译码器
A0-A12
CS
8K x 1
D7
D6
D5
D4
D3
D2
D1
D0
D0-D7
由32片8Kx1(bit)组成32KB存储器(地址高位交叉)


A19
A18
A17
A16
A15-A0
CS
D7-D0
512Kx8
D7-D4
D3-D0
128Kx4
D7,D6
D5,D4
D3,D2
D1,D0
128Kx2
128Kx8
128Kx8
64Kx8
D7-D0

Exploiting Memory Hierarchy

- Users want large and fast memories!

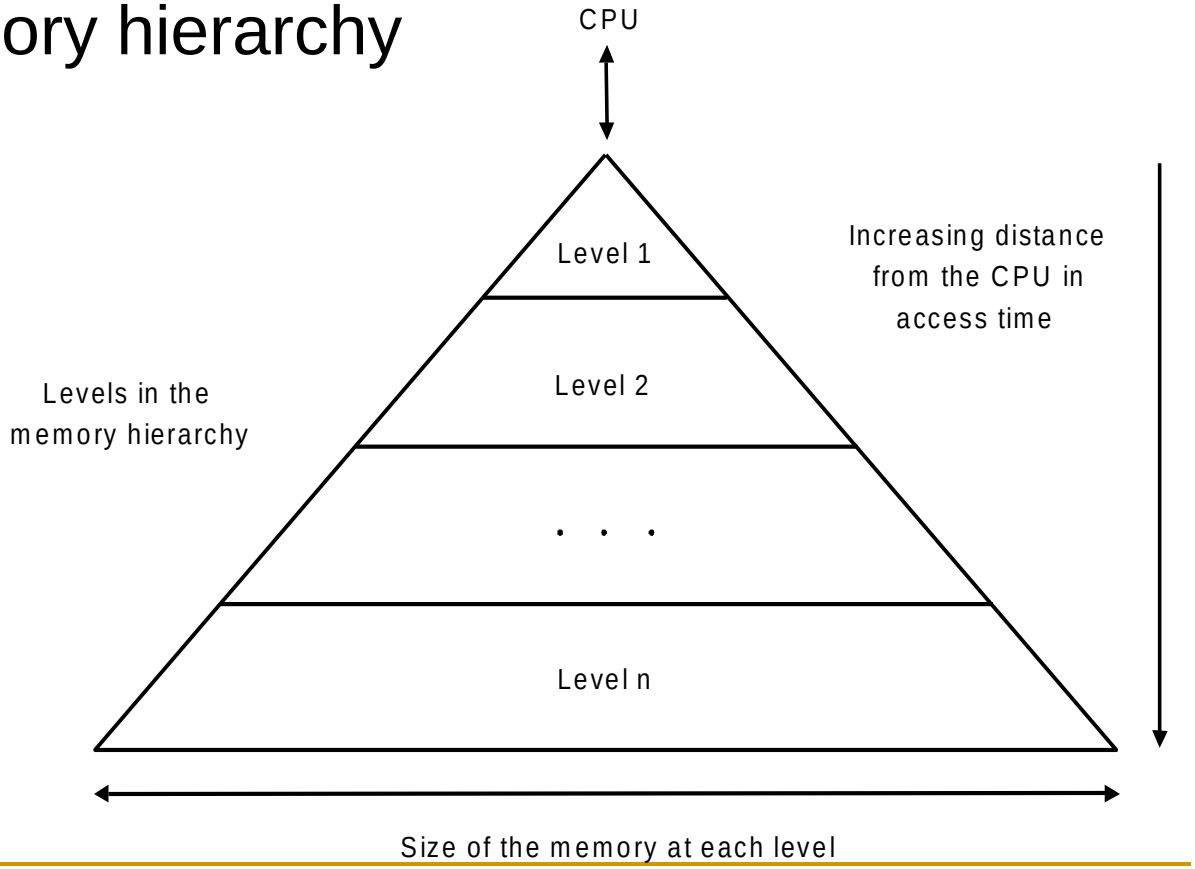
	Access Time	Dollars/MB(1997)
SRAM	2 - 25ns	$100 to $250
DRAM	60-120ns	$5 to $10
Disk	10 to 20 ms	$.10 to $.20

1997

EPROM

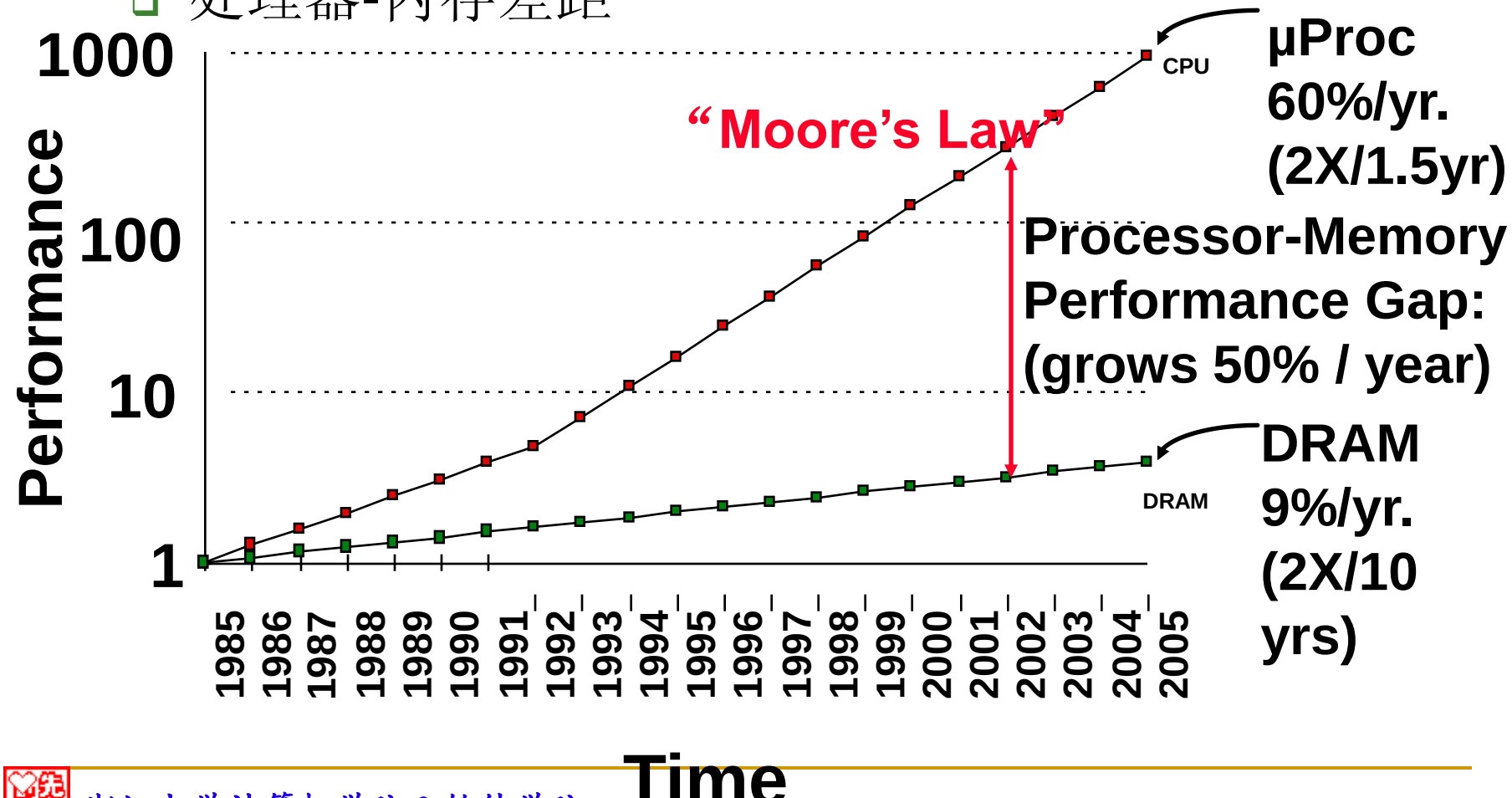
Exploiting Memory Hierarchy

- Try and give it to them anyway
  - build a memory hierarchy



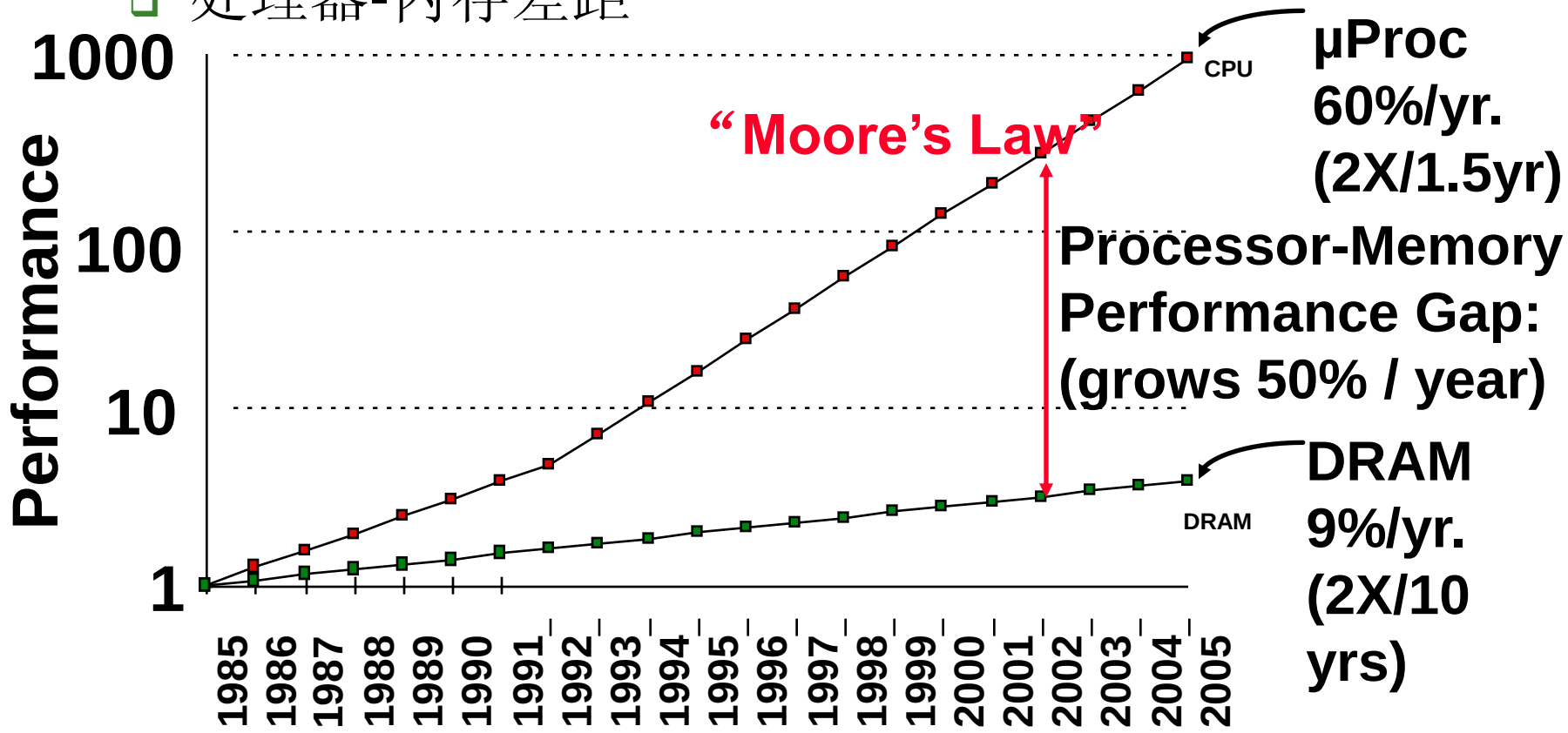
2. 高性能计算的发展与现状（26）

- 处理器-内存差距



处理器-内存差距

- 处理器-内存差距



Locality

- A principle that makes having a memory hierarchy a good idea
- If an item is referenced,
  - temporal locality: it will tend to be referenced again soon
  - spatial locality: nearby items will tend to be referenced soon.

Locality

- Our initial focus: two levels (upper, lower)
  - block: minimum unit of data
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level
- *Why does code have locality?*

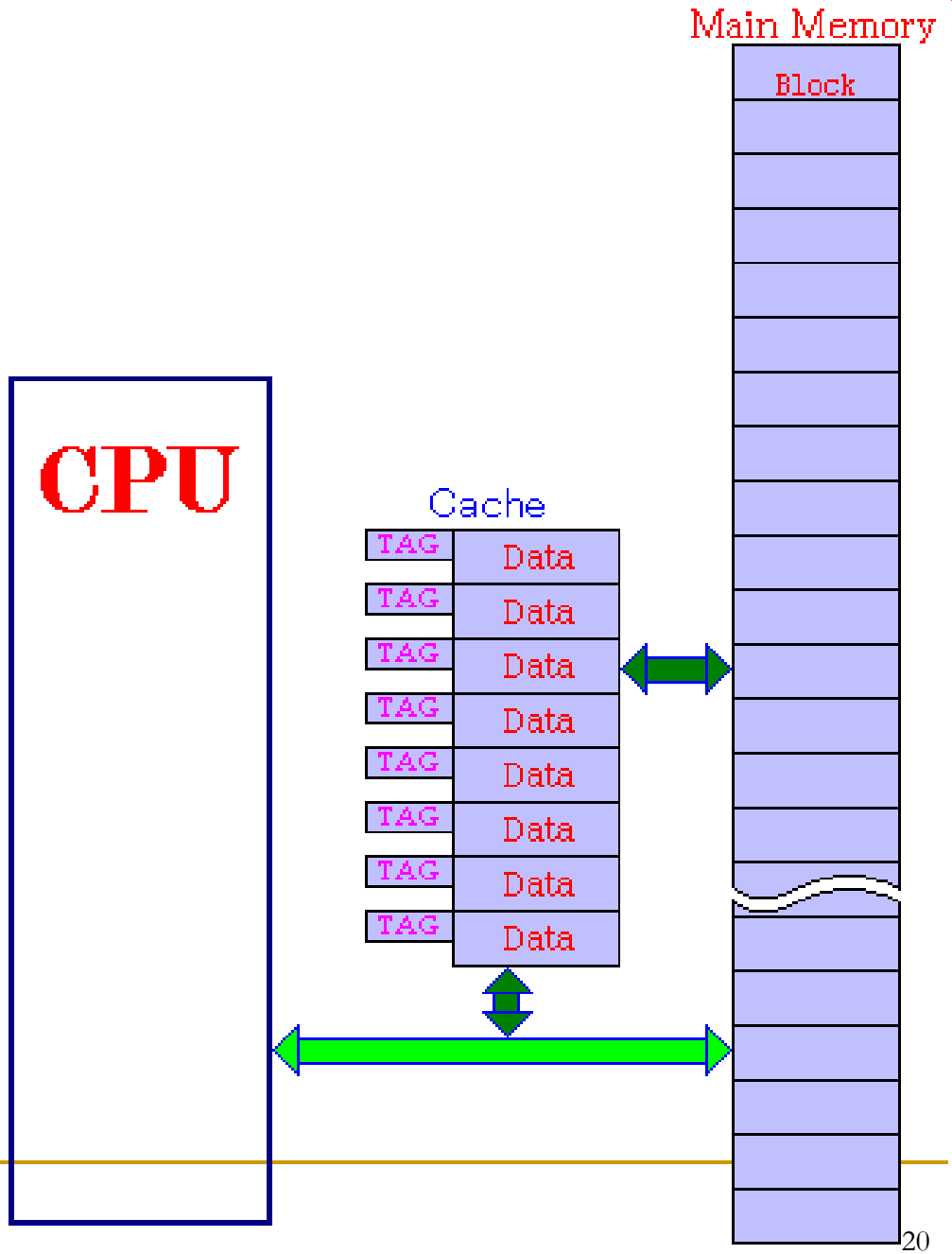
Cache

- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"

For each item of data at the lower level,
there is exactly one location in the cache where it might be.

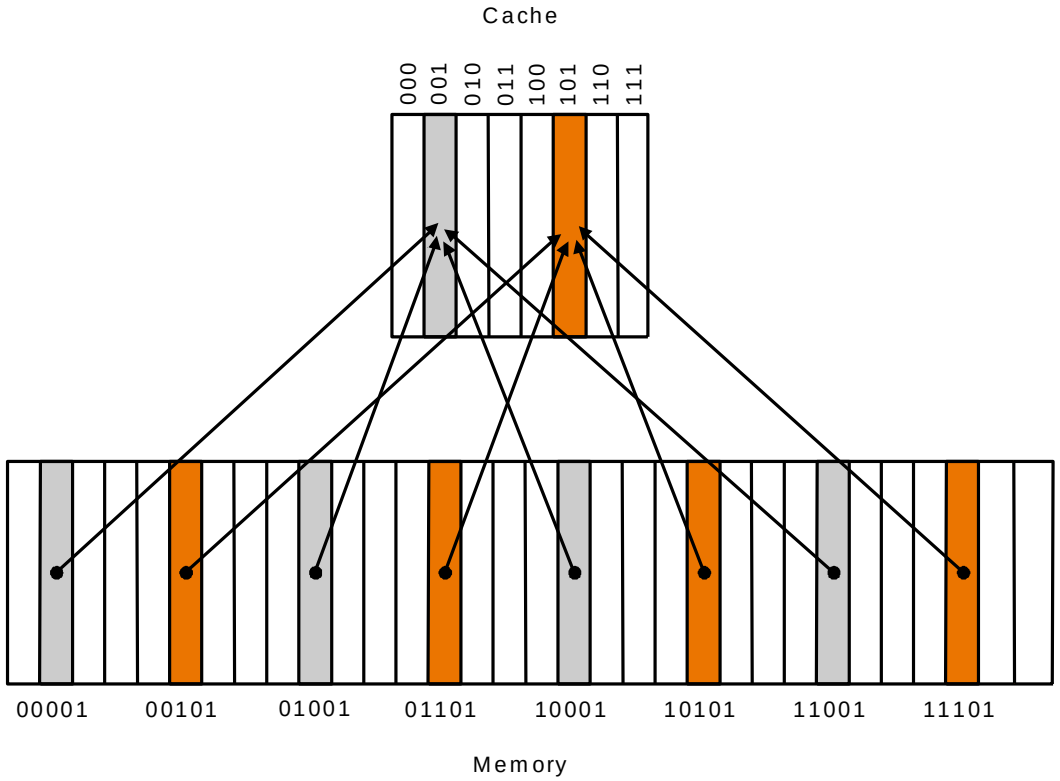
e.g., lots of items at the lower level share locations in the upper level

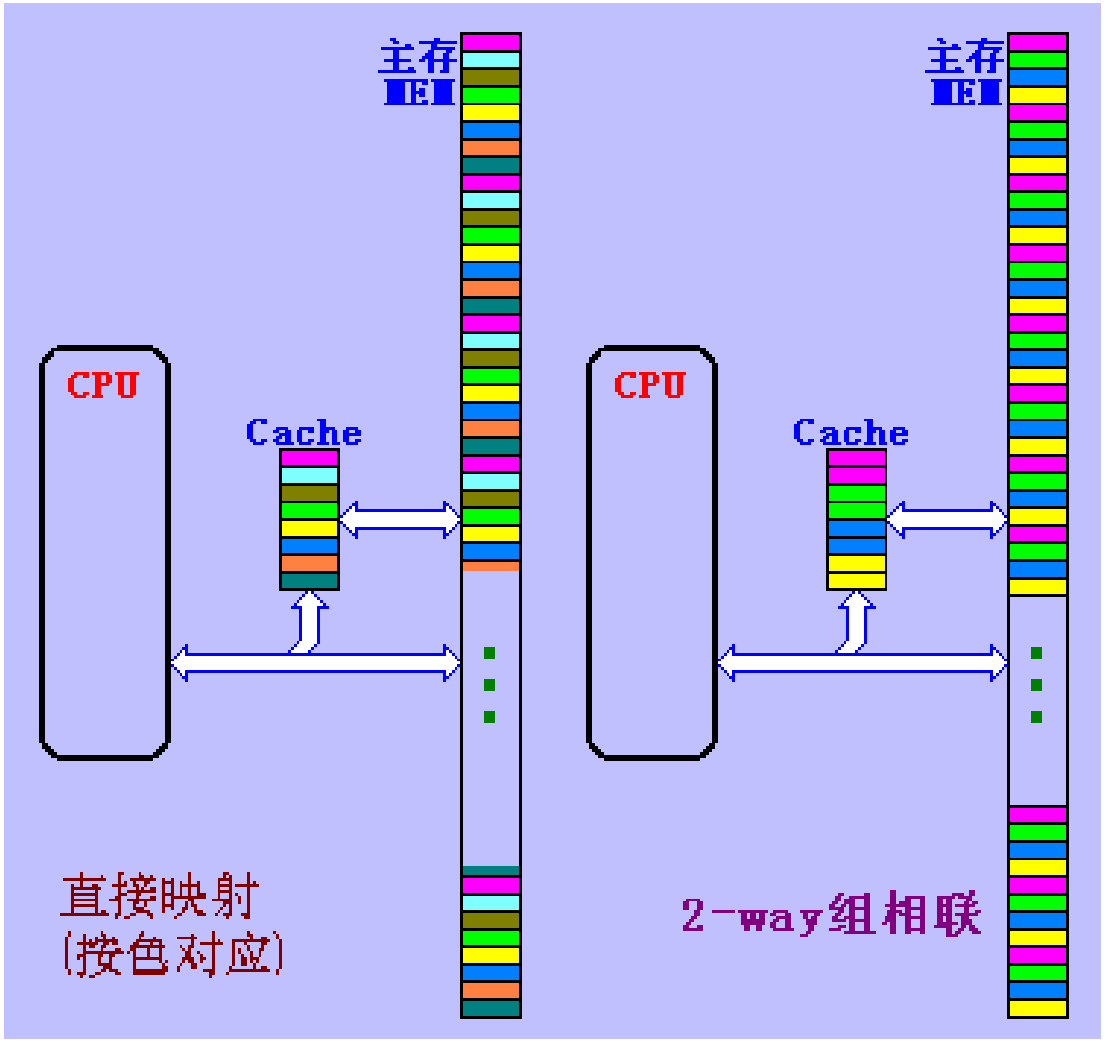
Cache


Main Memory
Block
CPU
Cache
TAG
Data
TAG
Data
TAG
Data
TAG
Data
TAG
Data
TAG
Data
TAG
Data
TAG
Data

Direct Mapped Cache

- Mapping: address is modulo the number of blocks in the cache




主存
CPU
Cache
直接映射
(按色对应)
主存
CPU
Cache
2-way组相联

Organization

- Full-Associative:

  cache line mapped to any cache slot - associative search on all tag fields
- Direct Mapped:

  cache line mapped to only one cache slot - tag fields compared
- set-assocative:

  cache line mapped to only one set of cache slots - associative seach on tag fields for set

Replacement policy

- LRU - least recently used
  (time stamp last access)
- FIFO - first in first out
  (time stamp when loaded)
- LFU - least frequently used
  (count number of accesses)
- Random
- Oracle (Optimal)
  predict next access time for all lines

Cache Read/Write Policies

- Read/Cache Miss:

  load through (load to cache and CPU in parallel) vs non-load through (load cache only and read again)

- Write/Cache Hit:

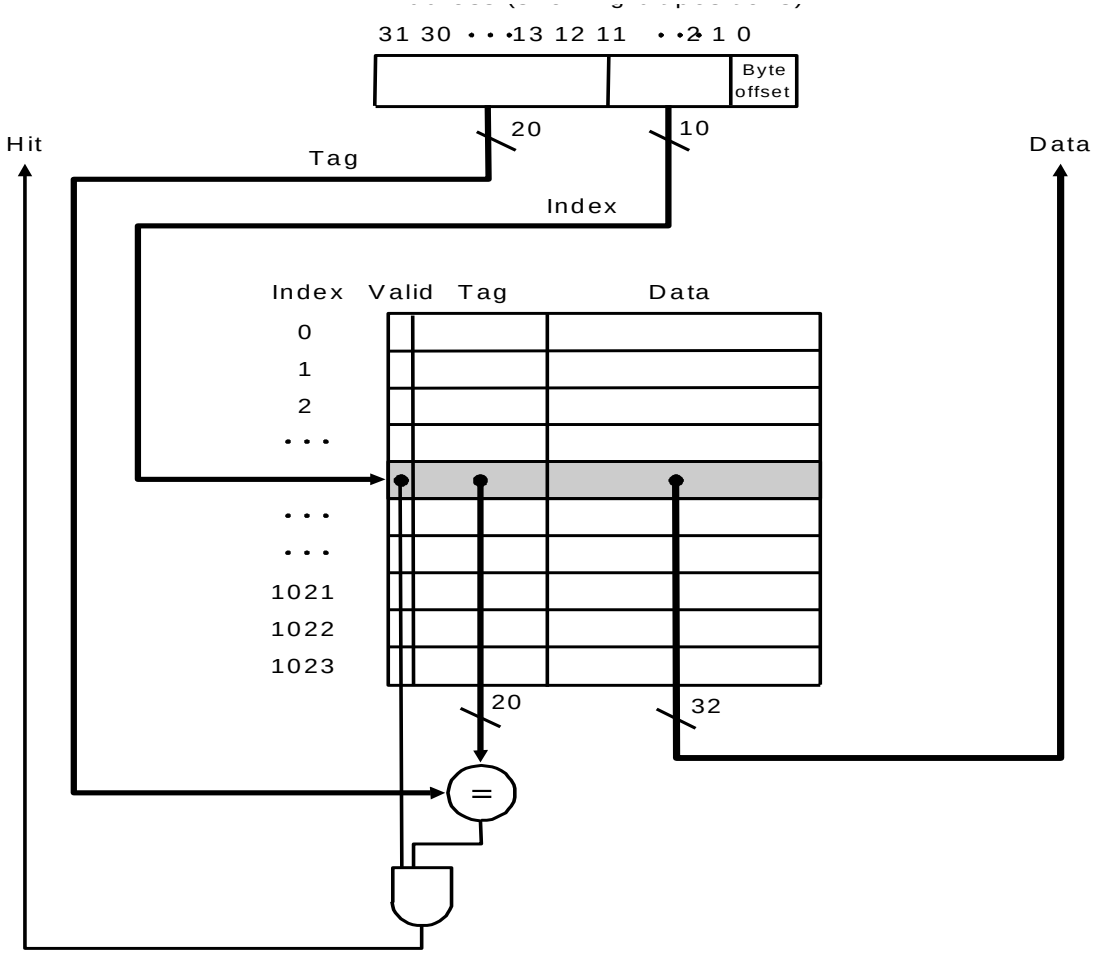
  write through (write to cache and main memory in parallel) vs **write back** (write to cache only; defer main memory update until cache line flushed)

- Write/Cache Miss:

  write allocate (load into cache and update) vs non-write allocate (write to main memory only)

Direct Mapped Cache

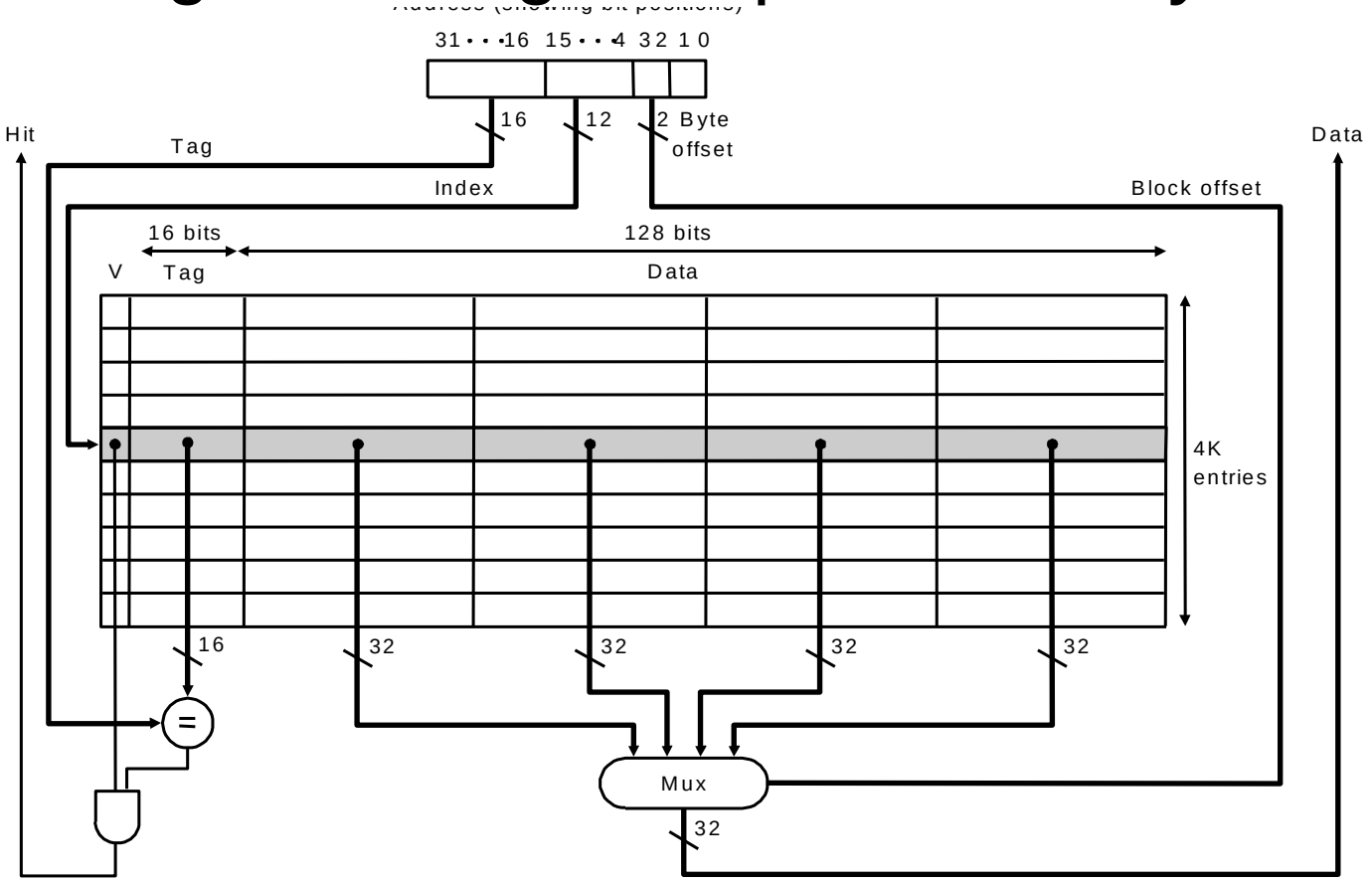
- For MIPS:



- *What kind of locality are we taking advantage of?*

Direct Mapped Cache

- Taking advantage of spatial locality:



Hits vs. Misses: Read

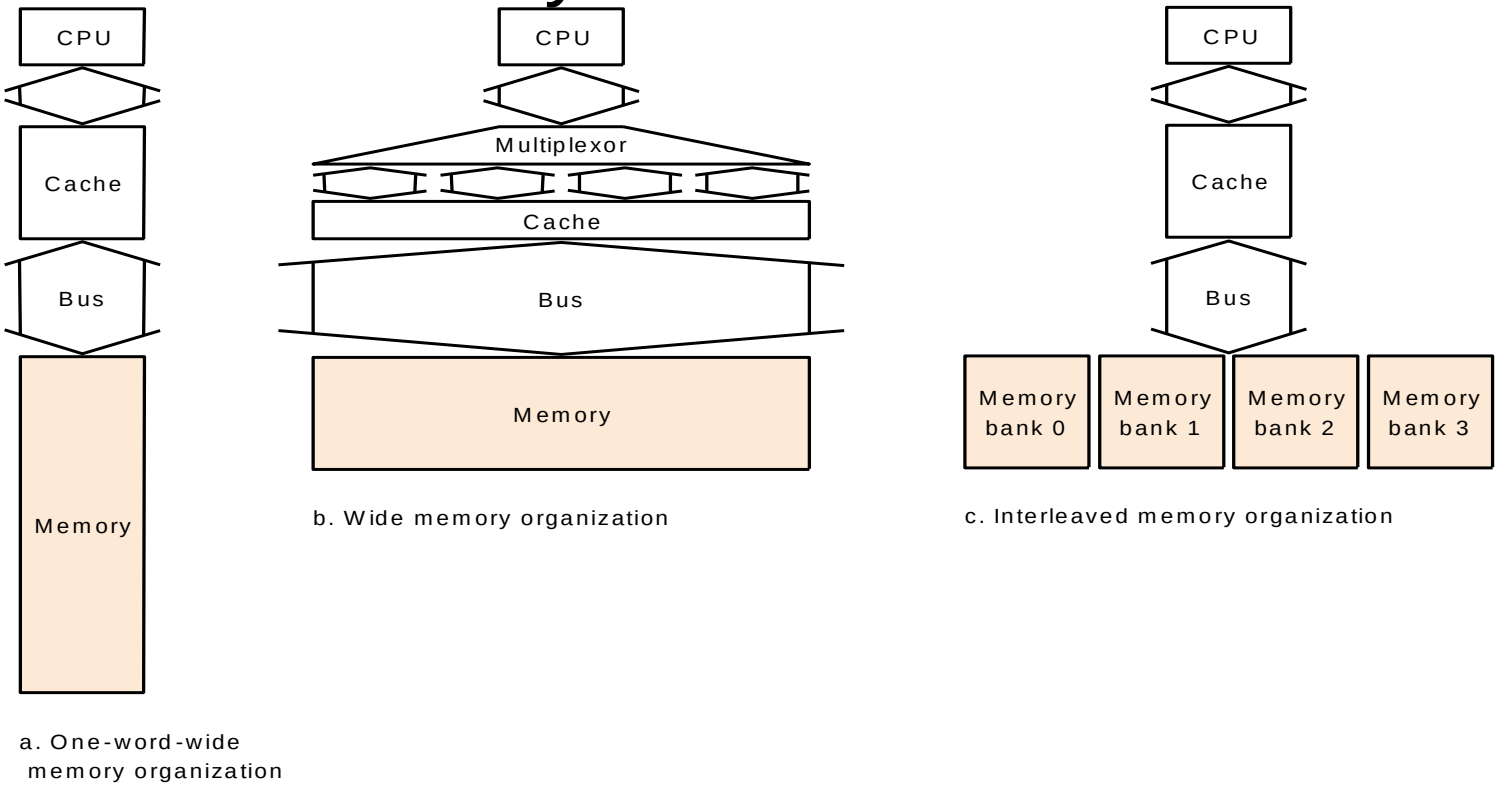
- Read hits
  - this is what we want!
- Read misses
  - stall the CPU, fetch block from memory, deliver to cache, restart

Hits vs. Misses: Write

- Write hits:
  - can replace data in cache and memory (write-through)
  - write the data only into the cache (write-back the cache later)
- Write misses:
  - read the entire block into the cache, then write the word

Hardware Issues

- Make reading multiple words easier by using banks of memory

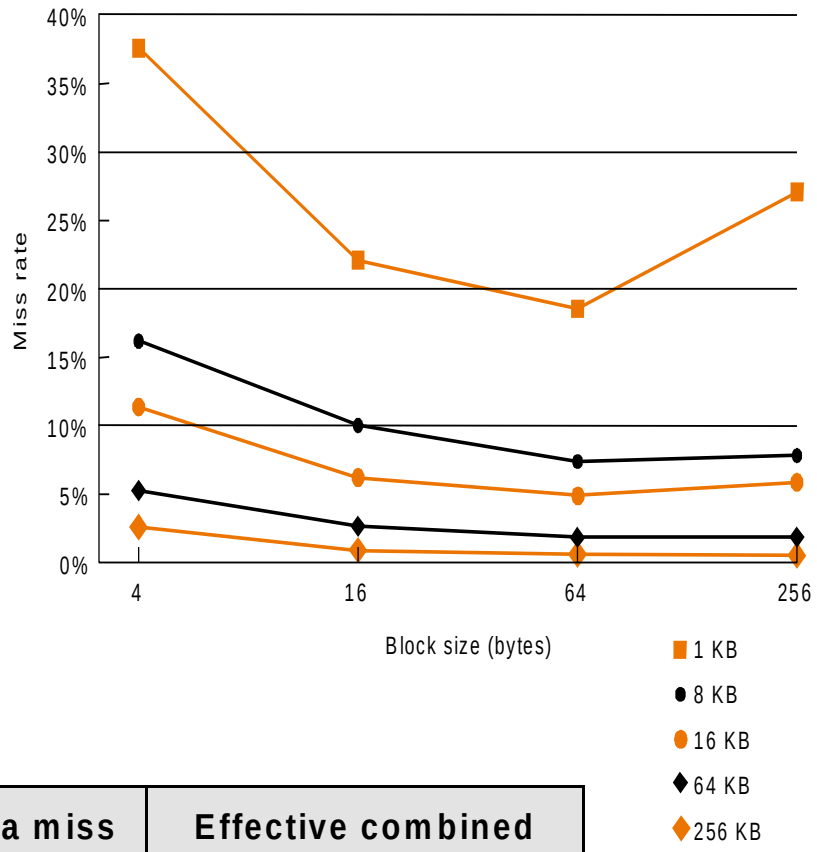


- It can get a lot more complicated...

浙江大学计算机学院&软件学院

Performance

- Increasing the block size tends to decrease miss rate:
- Use split caches because there is more spatial locality in code:



Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

Performance

- Simplified model:

  execution time = (execution cycles + stall cycles) * cycle time

  stall cycles = # of instructions * miss ratio * miss penalty

- Two ways of improving performance:
  - decreasing the miss ratio
  - decreasing the miss penalty

What happens if we increase block size?

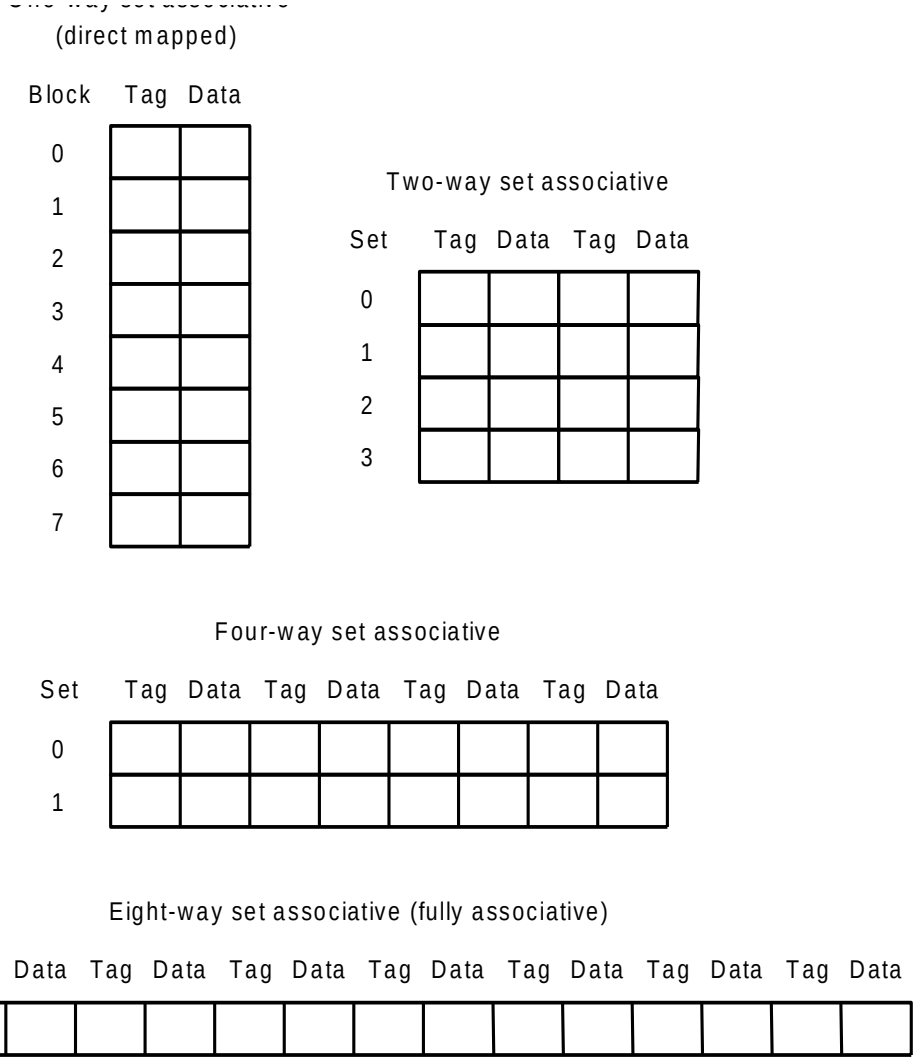
Decreasing miss ratio with associativity

Compared to direct mapped, give a series of references that:

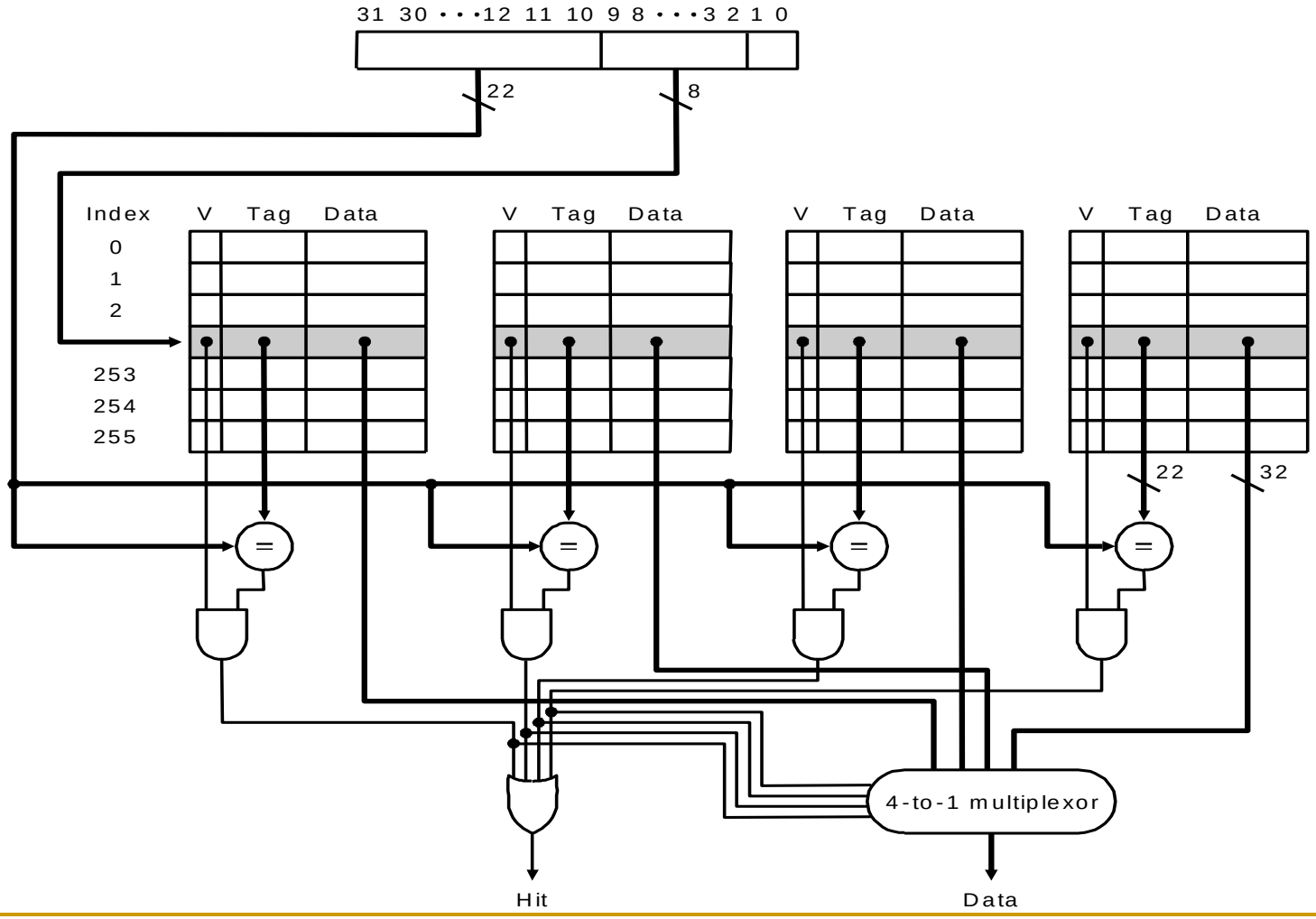
- *results in a lower miss ratio using a 2-way set associative cache*
- *results in a higher miss ratio using a 2-way set associative cache*

assuming we use the "least recently used" replacement strategy

Decreasing miss ratio with associativity



An implementation



Address mapping

8、设有一个 16 字的 Cache，就用三种地址映射方法：

A：直接地址映射，分 8 组，每组一块 2 个字；

B：组相联地址映射，分四组，每组 4 路(4-Way)，每块一字；

C：全相联地址相联，每块一字。

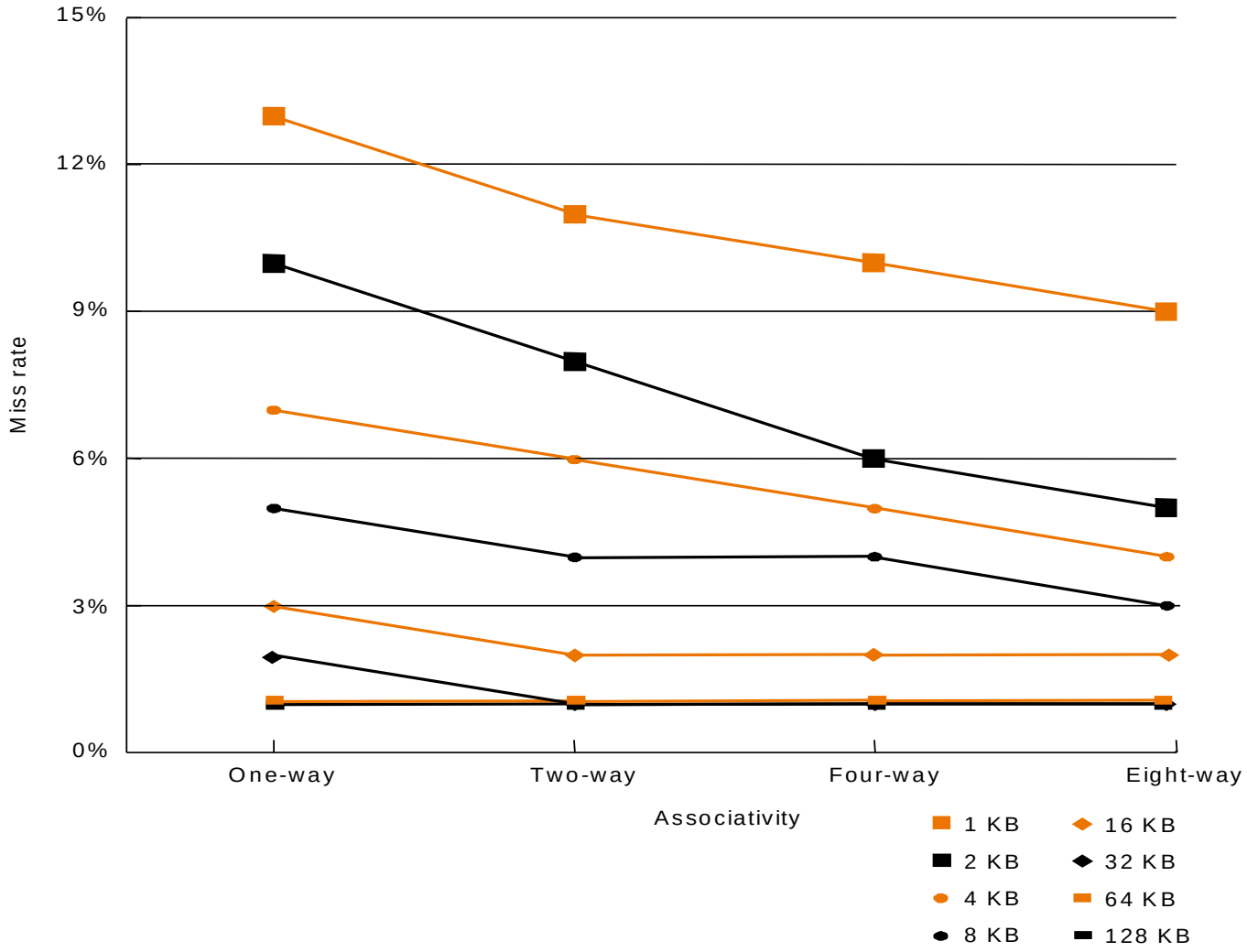
问：主存地址 35 号，在三种方式下，分别映射到 Cache 的哪个单元。

解：∵ $35_{10}=100011_2$

∴

35	**1**	**0**	**0**	**0**	**1**	**1**	描述(字组均从 0 开始)
A	Tag		Index(3 位：8 组)			Block	第 001 组，第 1 字
B	Tag				Index(2 位:4 组)		第 11 组(二进制)任一路
C	Tag						任一字块

Performance



3C

- Compulsory misses (cold-start misses)
  - can be reduced by increasing the block size.
- Capacity misses
  - multilevel caches
- Conflict misses (collision misses)
  - set-assocative → Full-assocative

Decreasing miss penalty with multilevel caches

- Add a second level cache:
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache

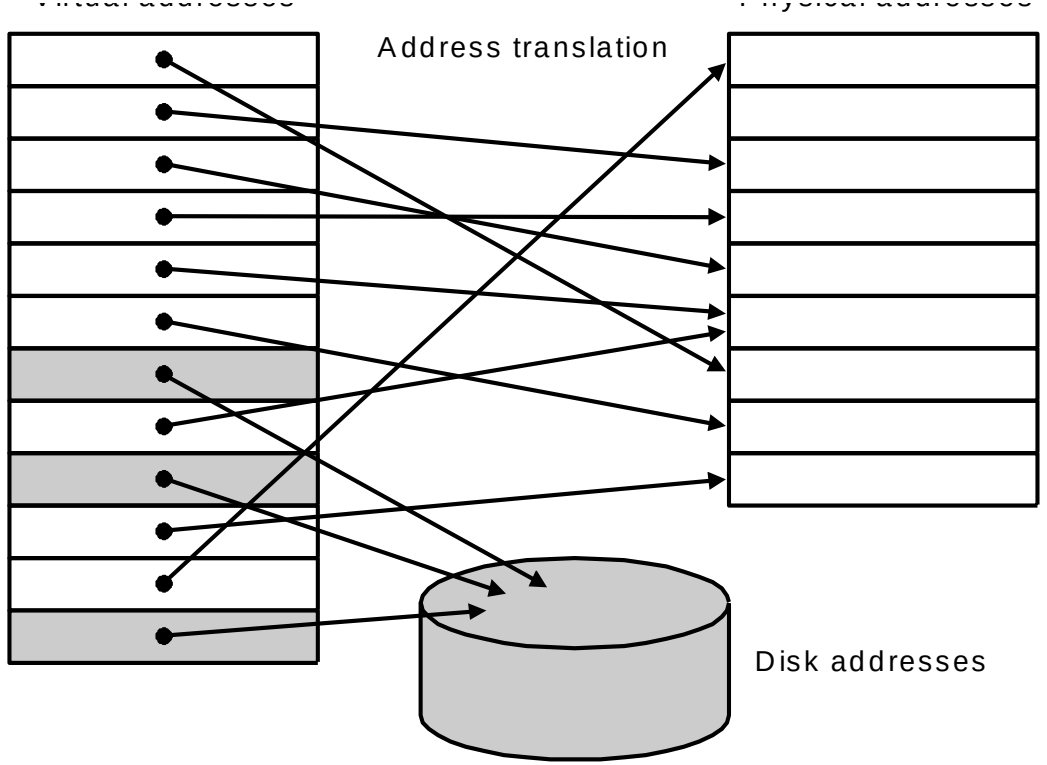
Decreasing miss penalty with multilevel caches

- Example:
  - CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
  - Adding 2nd level cache with 20ns access time decreases miss rate to 2%

- Using multilevel caches:
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache

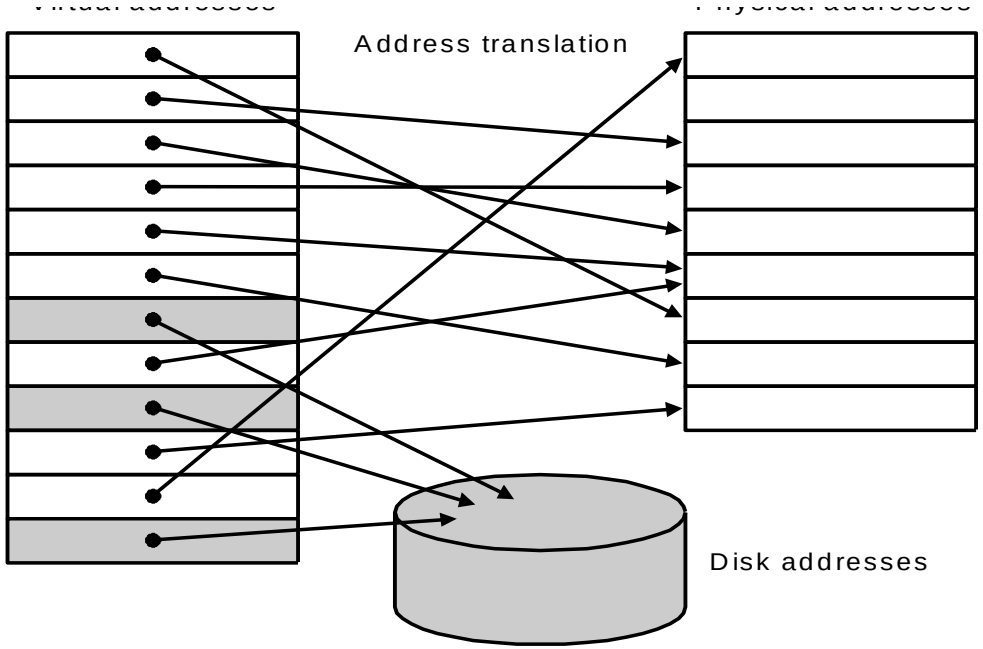
Virtual Memory

- Main memory can act as a cache for the secondary storage (disk)



Virtual Memory

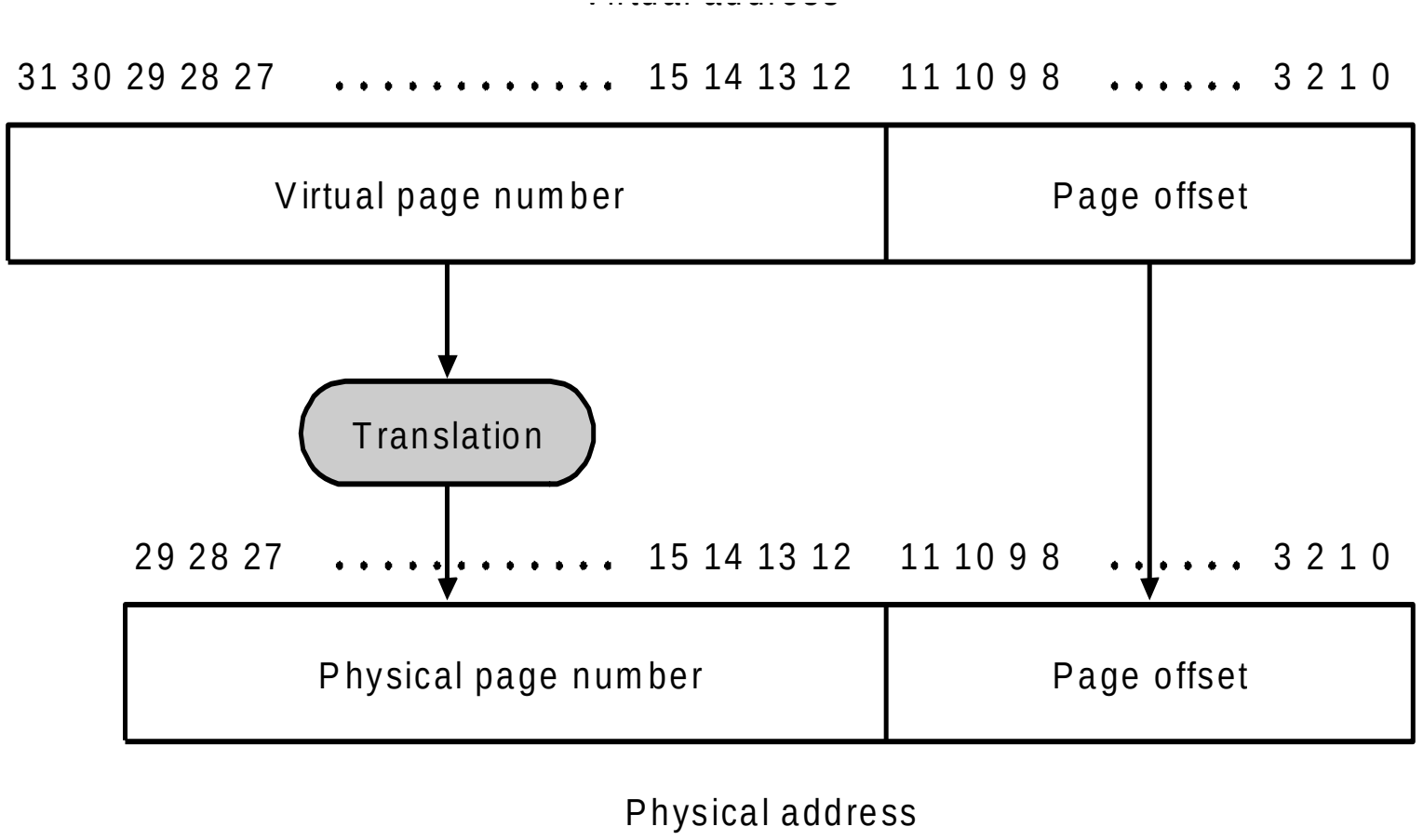
- Advantages:
  - illusion of having more physical memory
  - program relocation
  - protection



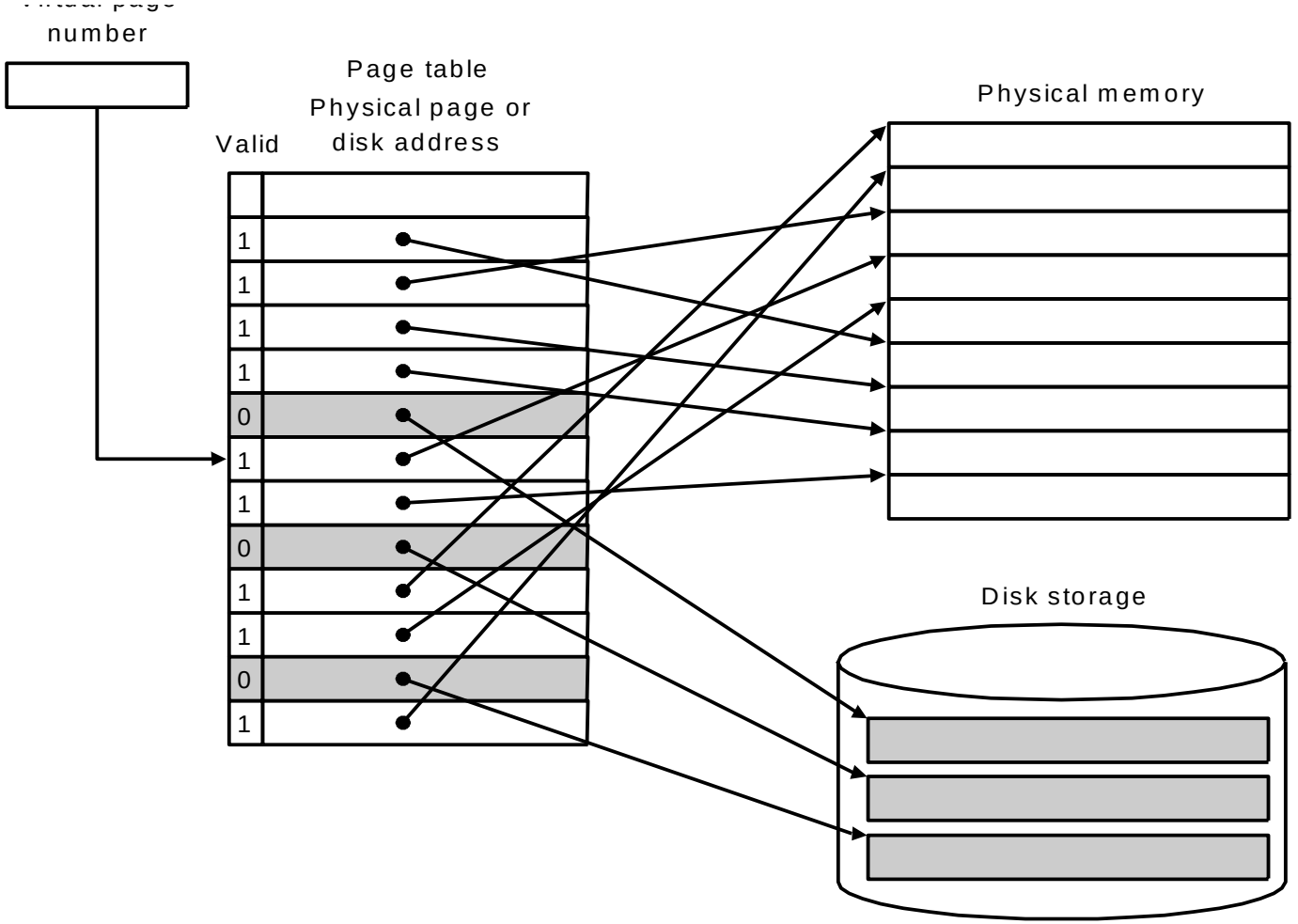
Pages: virtual memory blocks

- Page faults: the data is not in memory, retrieve it from disk
  - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
  - reducing page faults is important (LRU is worth the price)
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we use write-back

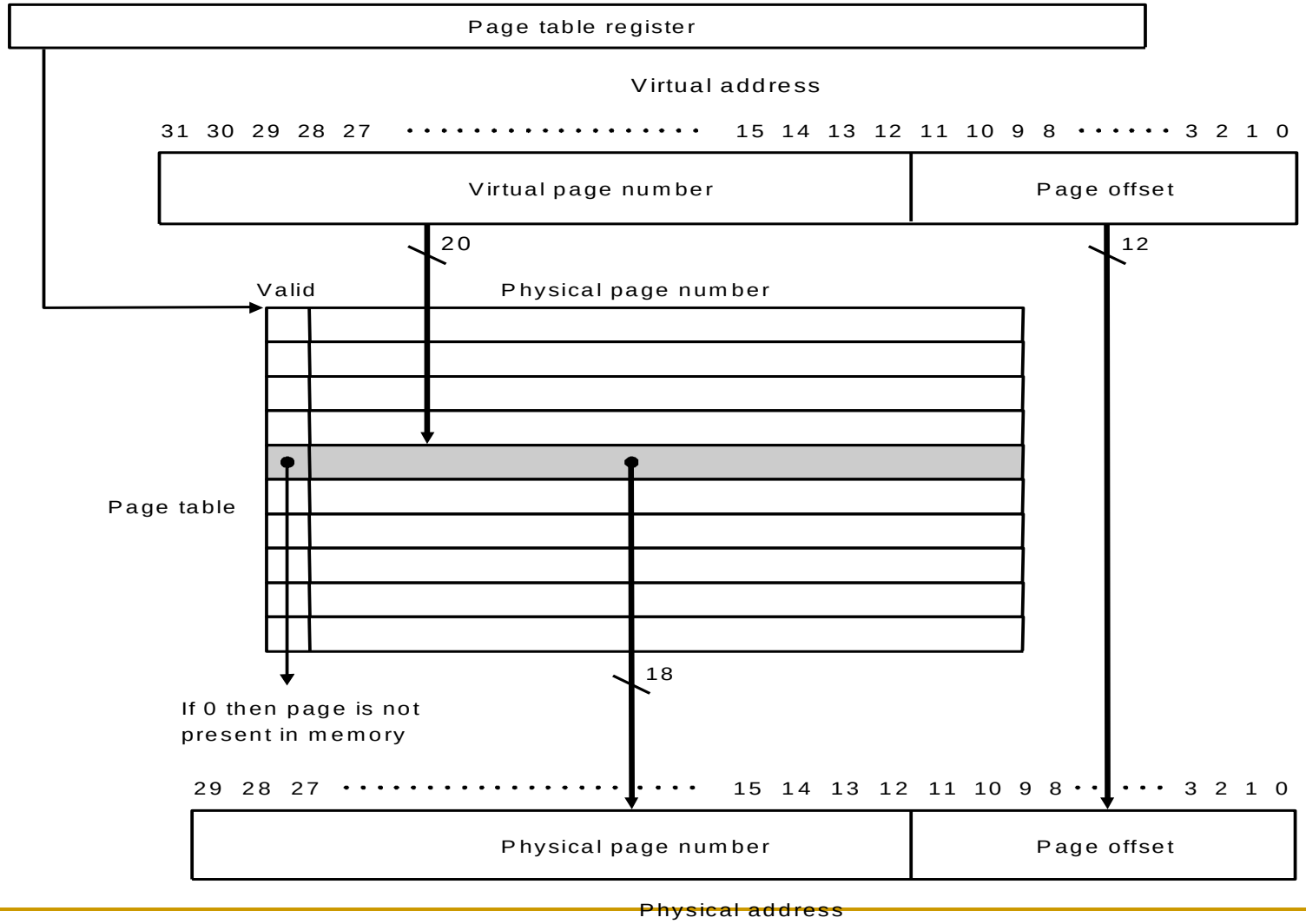
Address: virtual memory blocks



Page Tables

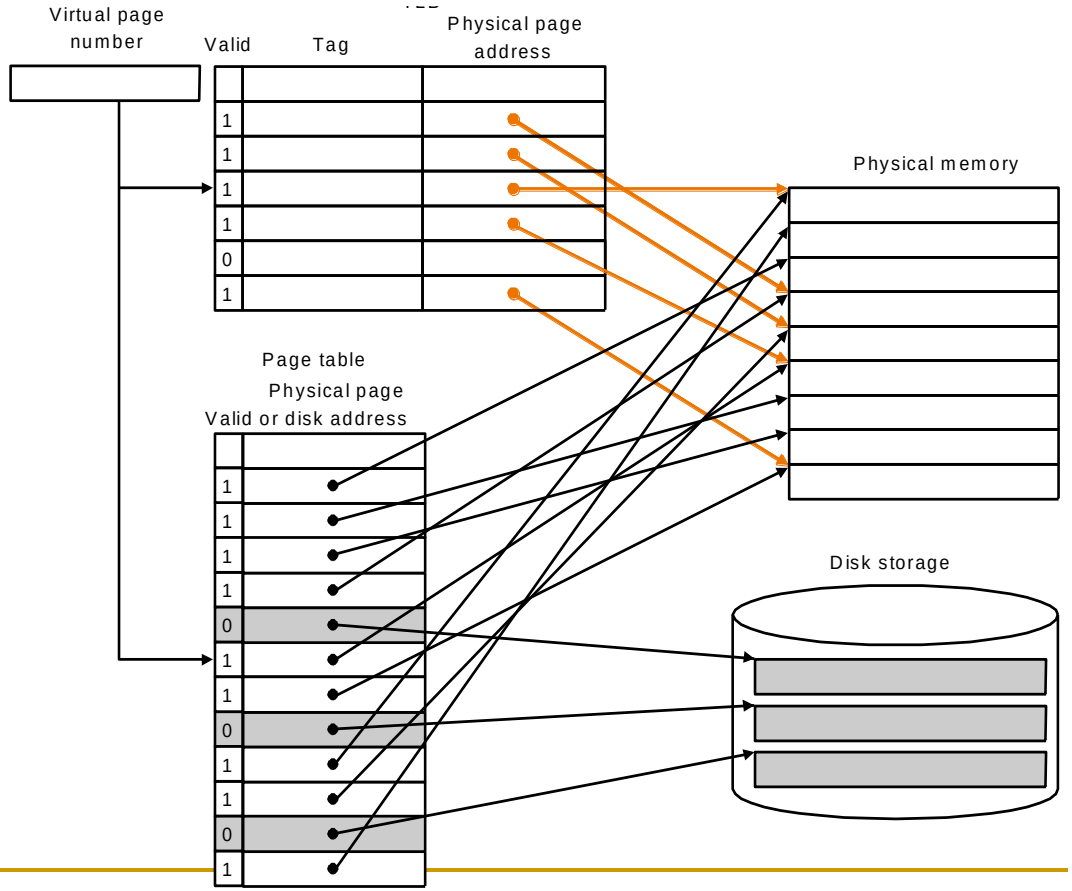

number
Page table
Physical page or
disk address
Valid
1
1
1
1
0
1
1
0
1
1
0
1
Physical memory
Disk storage

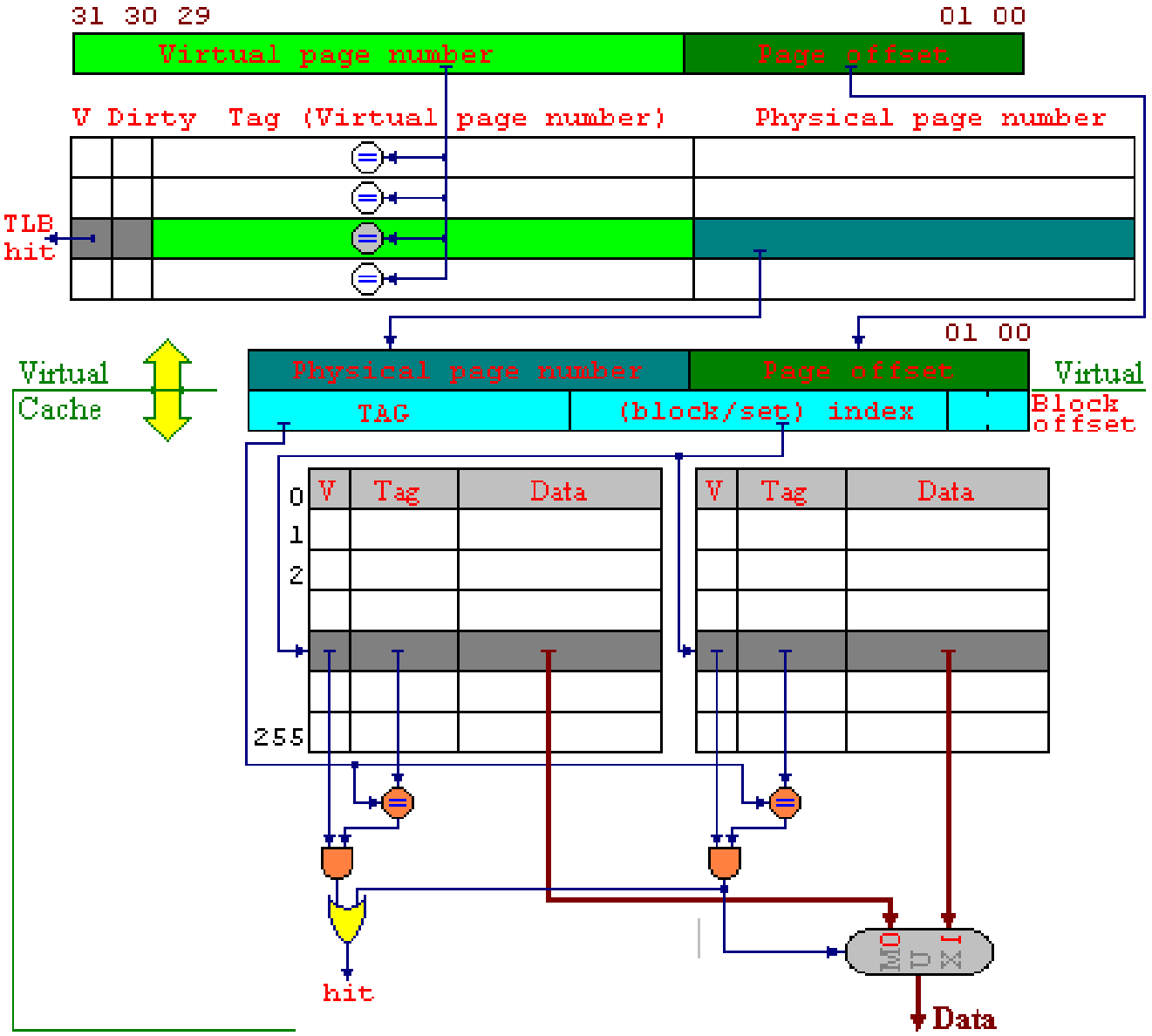
Page Tables


Page table register
Virtual address
31 30 29 28 27 ..... 15 14 13 12 11 10 9 8 ..... 3 2 1 0
Virtual page number
Page offset
20
12
Valid
Physical page number
Page table
18
If 0 then page is not present in memory
29 28 27 ..... 15 14 13 12 11 10 9 8 ..... 3 2 1 0
Physical page number
Page offset
Physical address

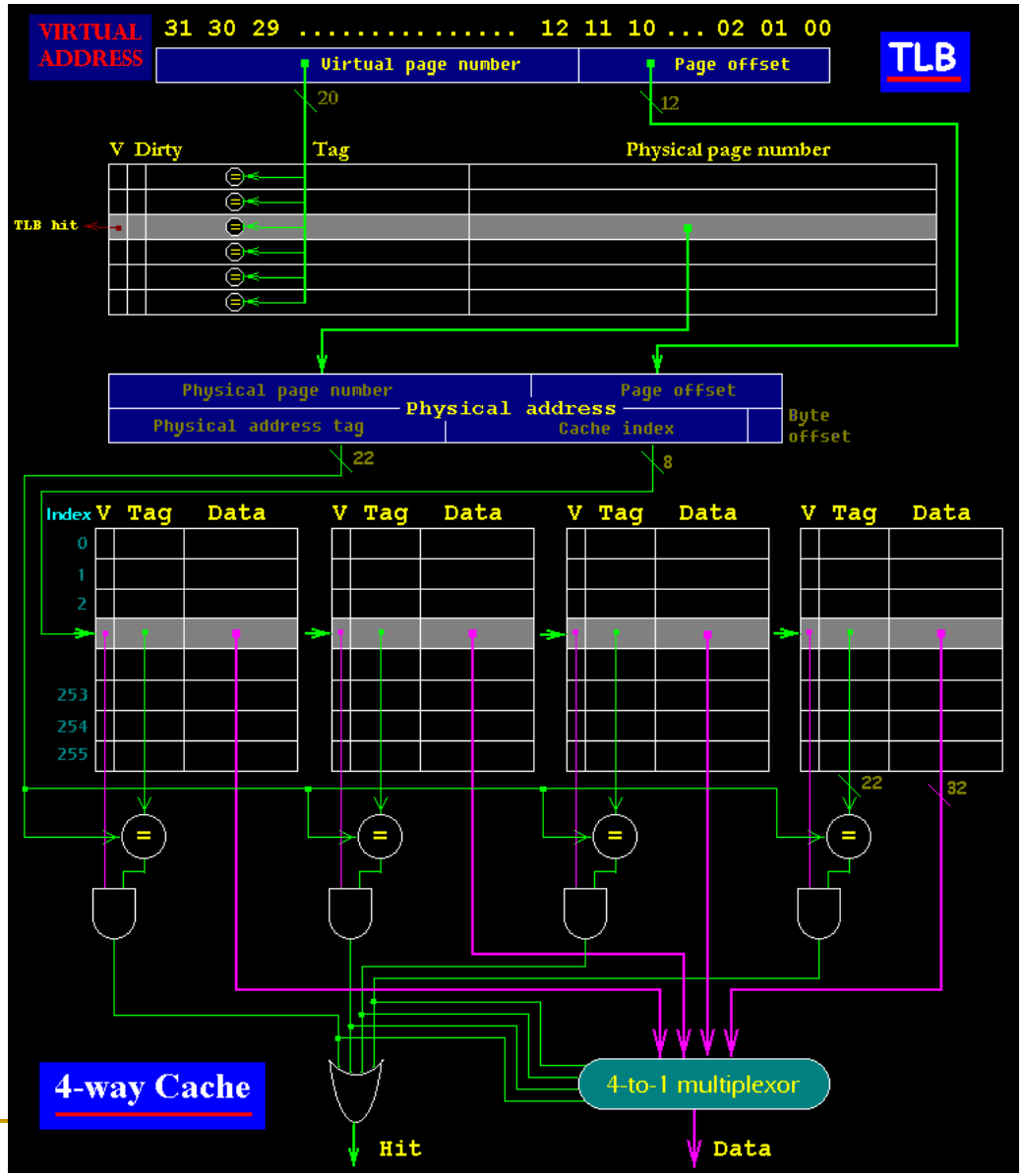
Making Address Translation Fast

- A cache for address translations: translation lookaside buffer

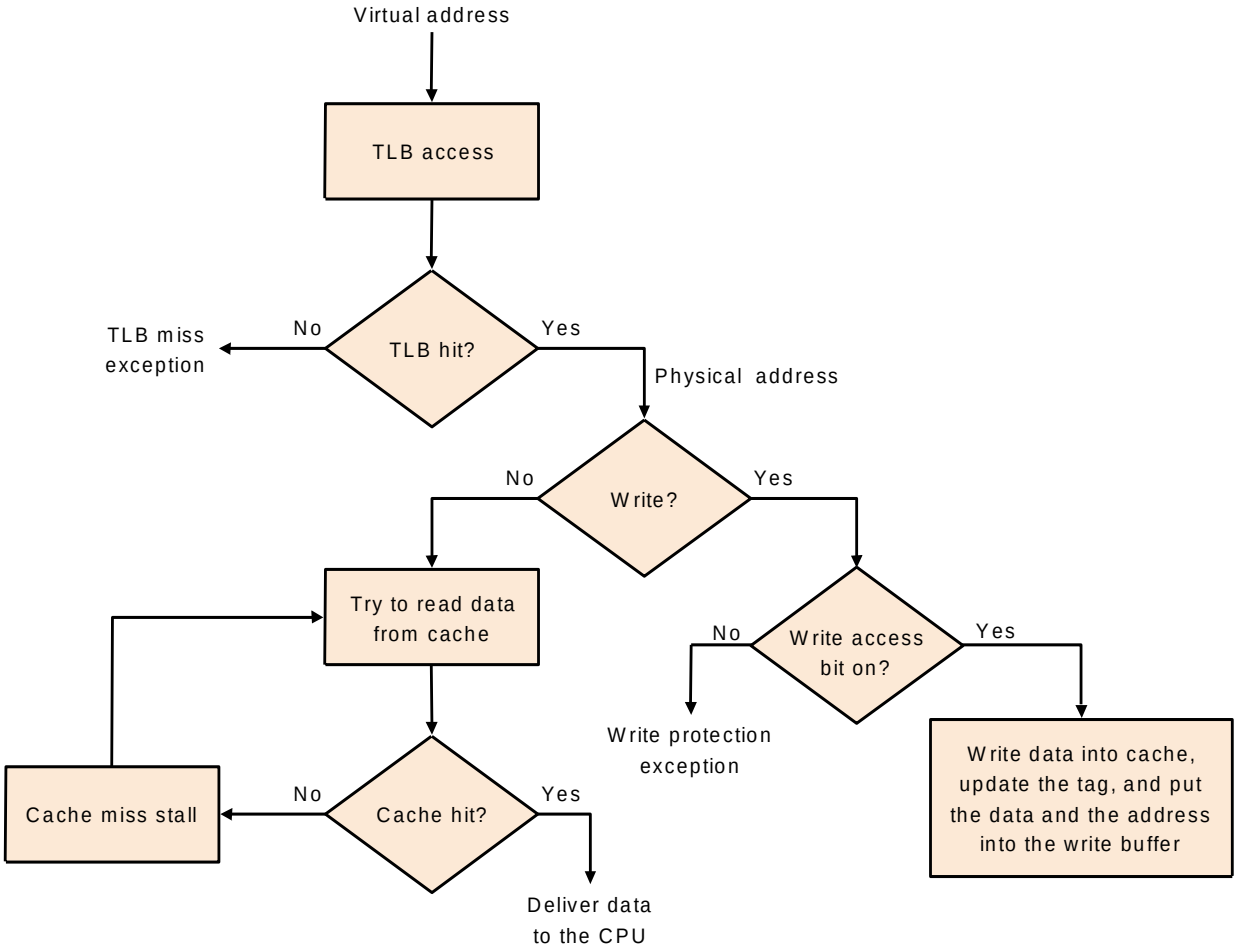



31 30 29
01 00
Virtual page number
Page offset
V Dirty Tag (Virtual page number)
Physical page number
TLB hit
01 00
Physical page number
Page offset
Virtual
Cache
TAG
(block/set) index
Virtual Block offset
0
1
2
255
V
Tag
Data
V
Tag
Data
MUX
hit
Data

Cache & Virtual



TLBs and caches

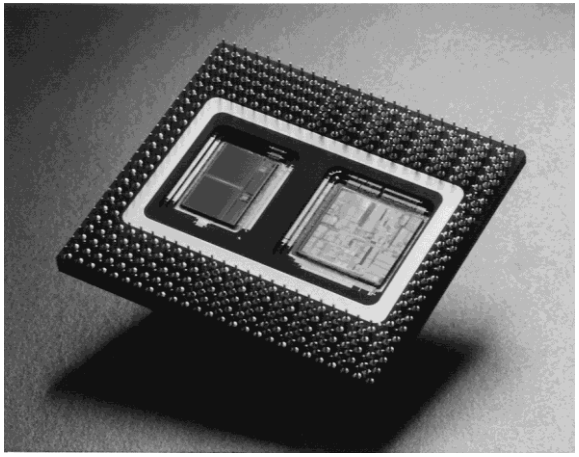


Modern Systems

- Very complicated memory systems:

Characteristic	Intel Pentium Pro	PowerPC 604
Virtual address	32 bits	52 bits
Physical address	32 bits	32 bits
Page size	4 KB, 4 MB	4 KB, selectable, and 256 MB
TLB organization	A TLB for instructions and a TLB for data Both four-way set associative Pseudo-LRU replacement Instruction TLB: 32 entries Data TLB: 64 entries TLB misses handled in hardware	A TLB for instructions and a TLB for data Both two-way set associative LRU replacement Instruction TLB: 128 entries Data TLB: 128 entries TLB misses handled in hardware

Modern Systems



Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split intruction 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

Some Issues

- Processor speeds continue to increase very fast
  — much faster than either DRAM or disk access times
- Design challenge: dealing with this growing disparity

Some Issues

- Trends:
  - synchronous SRAMs (provide a burst of data)
  - redesign DRAM chips to provide higher bandwidth or processing
  - restructure code to increase locality
  - use prefetching (make cache visible to ISA)