4小时学会计算机组成与结构
—下卷

Memory

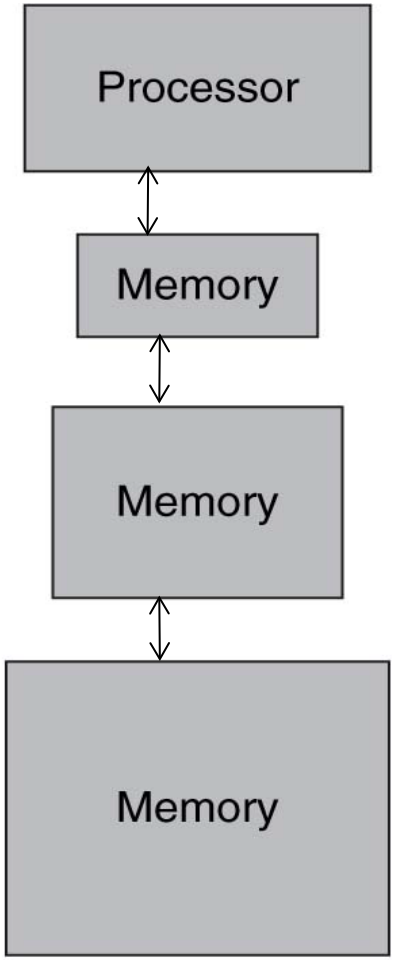
Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $2000 – $5000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $20 – $75 per GB
- Magnetic disk
  - 5ms – 20ms, $0.20 – $2 per GB
- Flash Memory
  - 0.1 – 2 ms, $5 – 50 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - e.g., sequential instruction access, array data

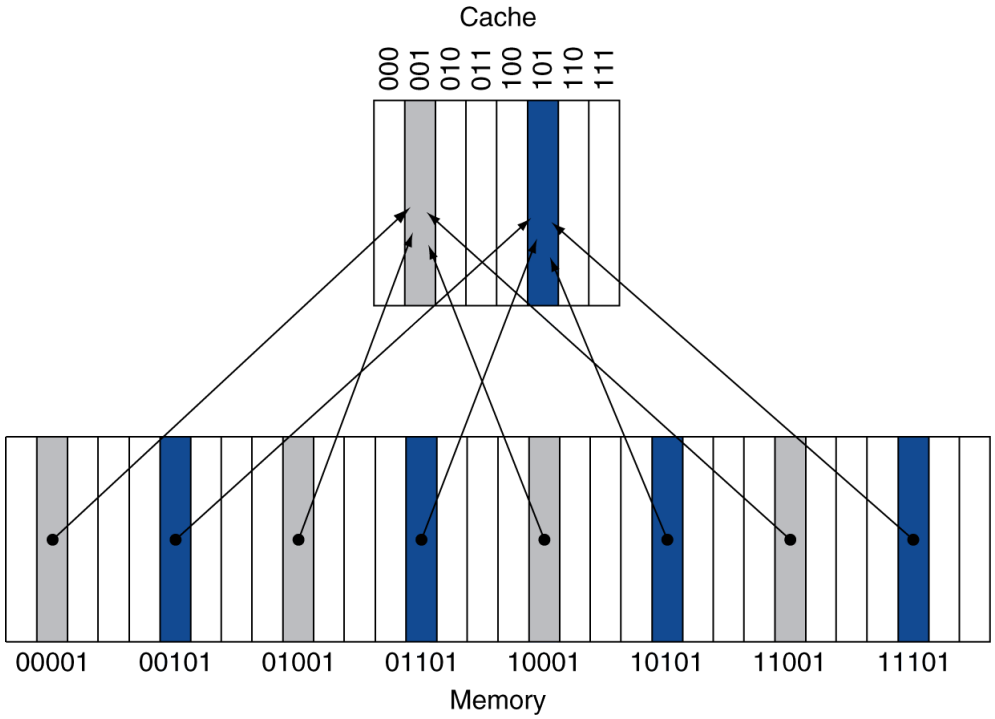
Memory Hierarchy Levels



Speed	Size	Cost ($/bit)	Current technology
Fastest	Smallest	Highest	SRAM
			DRAM
Slowest	Biggest	Lowest	Magnetic disk

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits for block addr

Associative Caches

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - One comparator per cache entry (expensive!)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Spectrum of Associativity

- For a cache with 8 entries

One-way set associative

(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

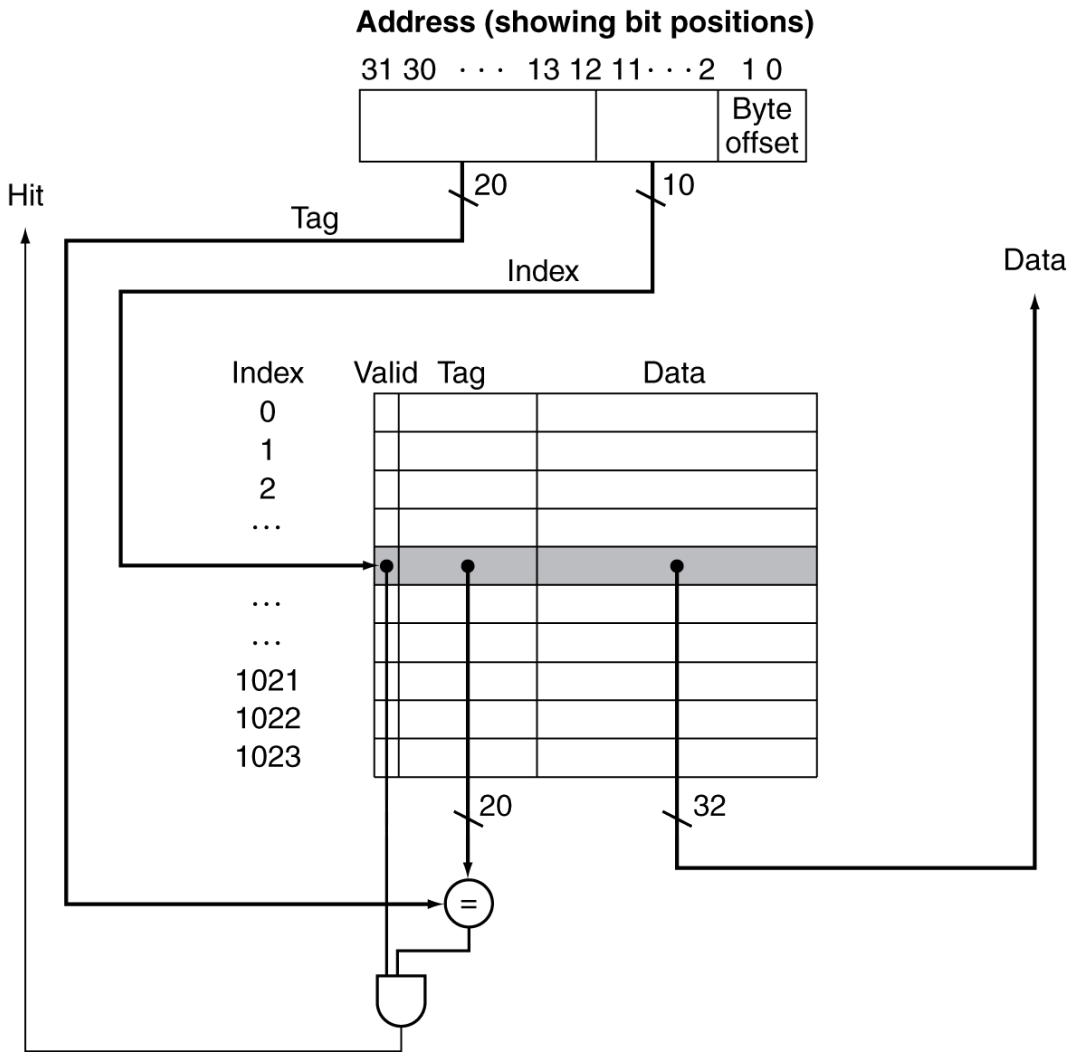
Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

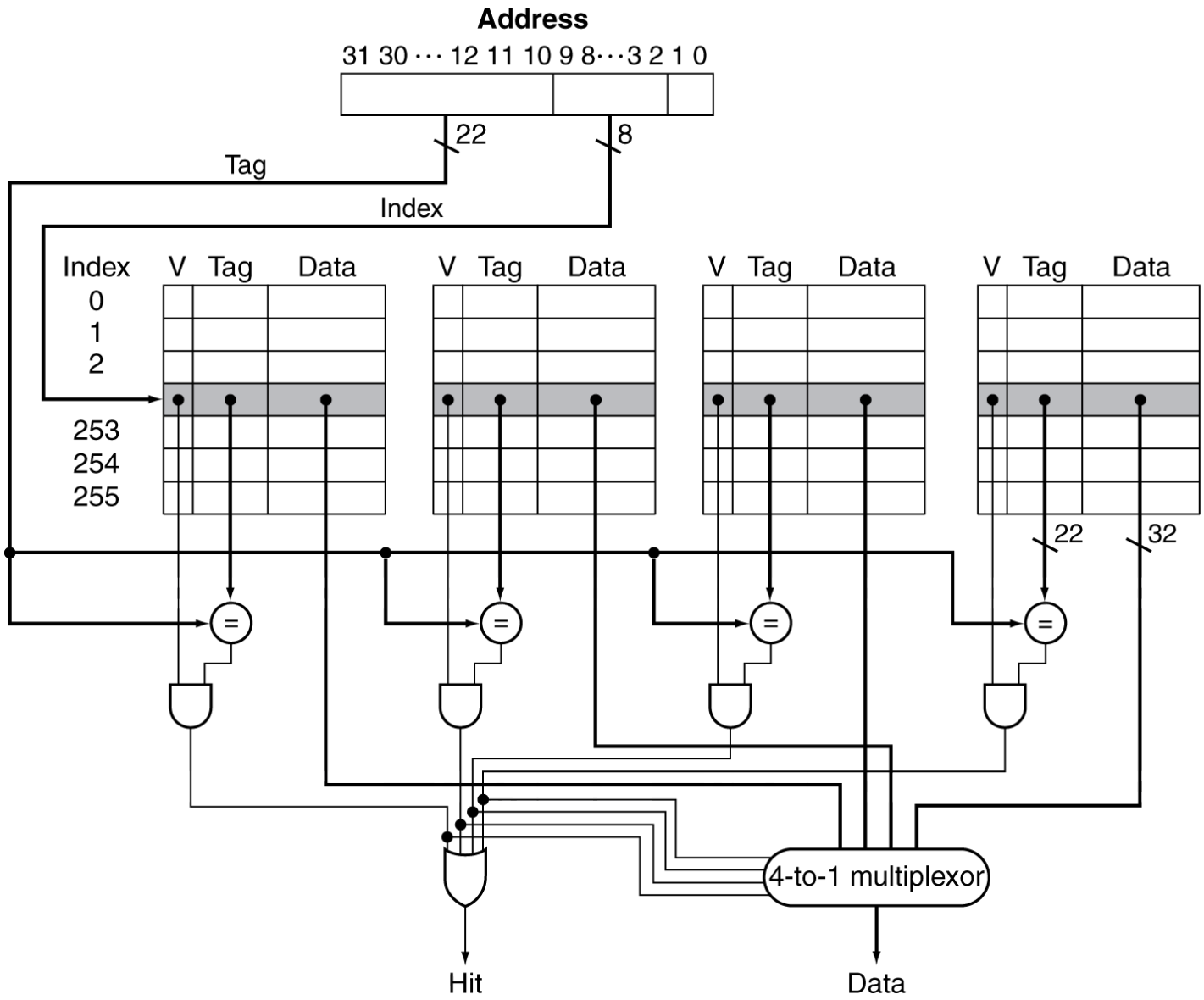
Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Direct Mapped Cache Organization



Set Associative Cache Organization



Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
    - Instead of interrupts which require saving context
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, miss rate/instruction = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:
  - Write buffer stalls negligible

$$\text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Back

- Alternative Solution: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first
    - without waiting for the block to be written into memory first

Replacement Policy

- Direct mapped
  - no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
  - Least-recently used (LRU)
    - Choose the one unused for the longest time
      - Simple for 2-way, manageable for 4-way, too hard beyond that
  - Random
    - Gives approximately the same performance as LRU for high associativity

Multilevel Cache Considerations

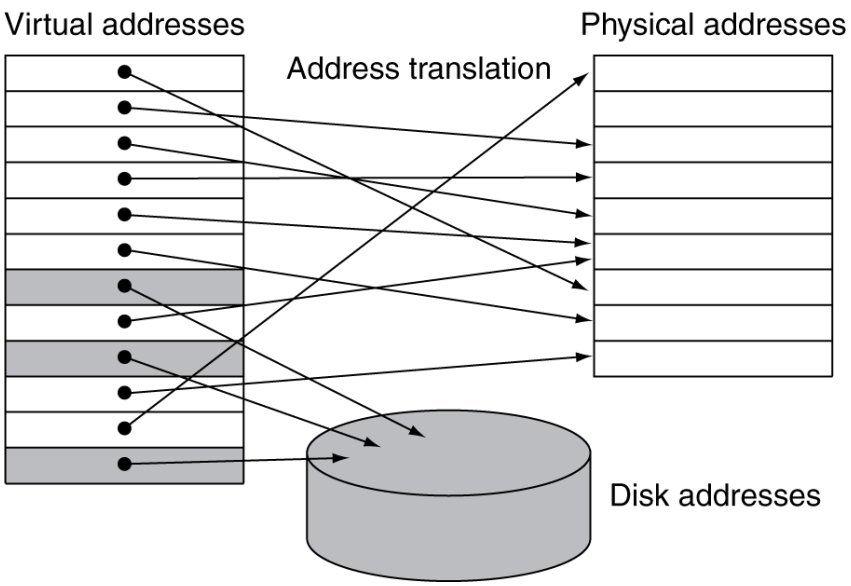
- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

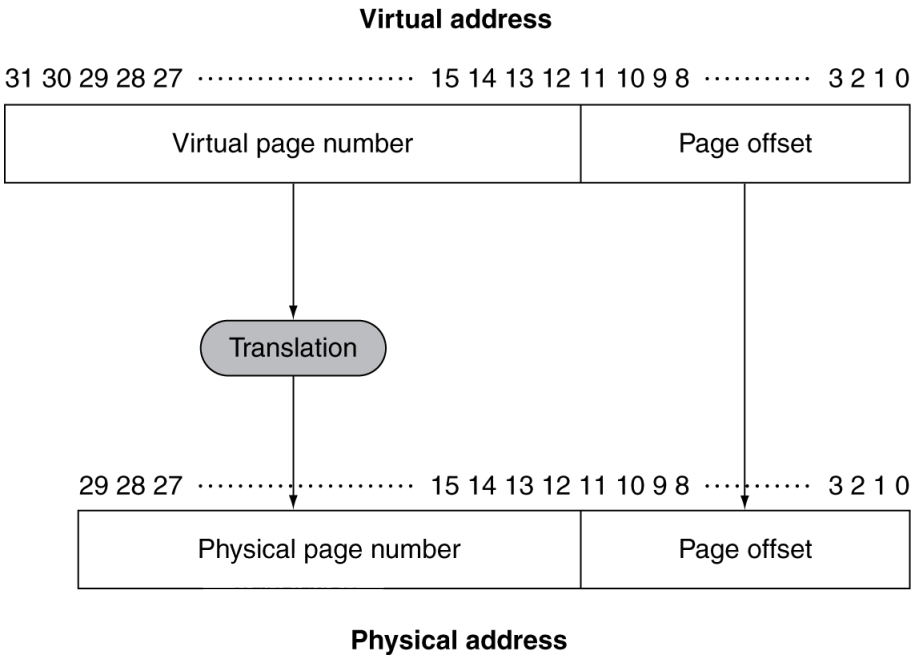
Virtual Memory

- Use main memory as a “cache” for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM “block” is called a page
  - VM translation “miss” is called a page fault

Address Translation

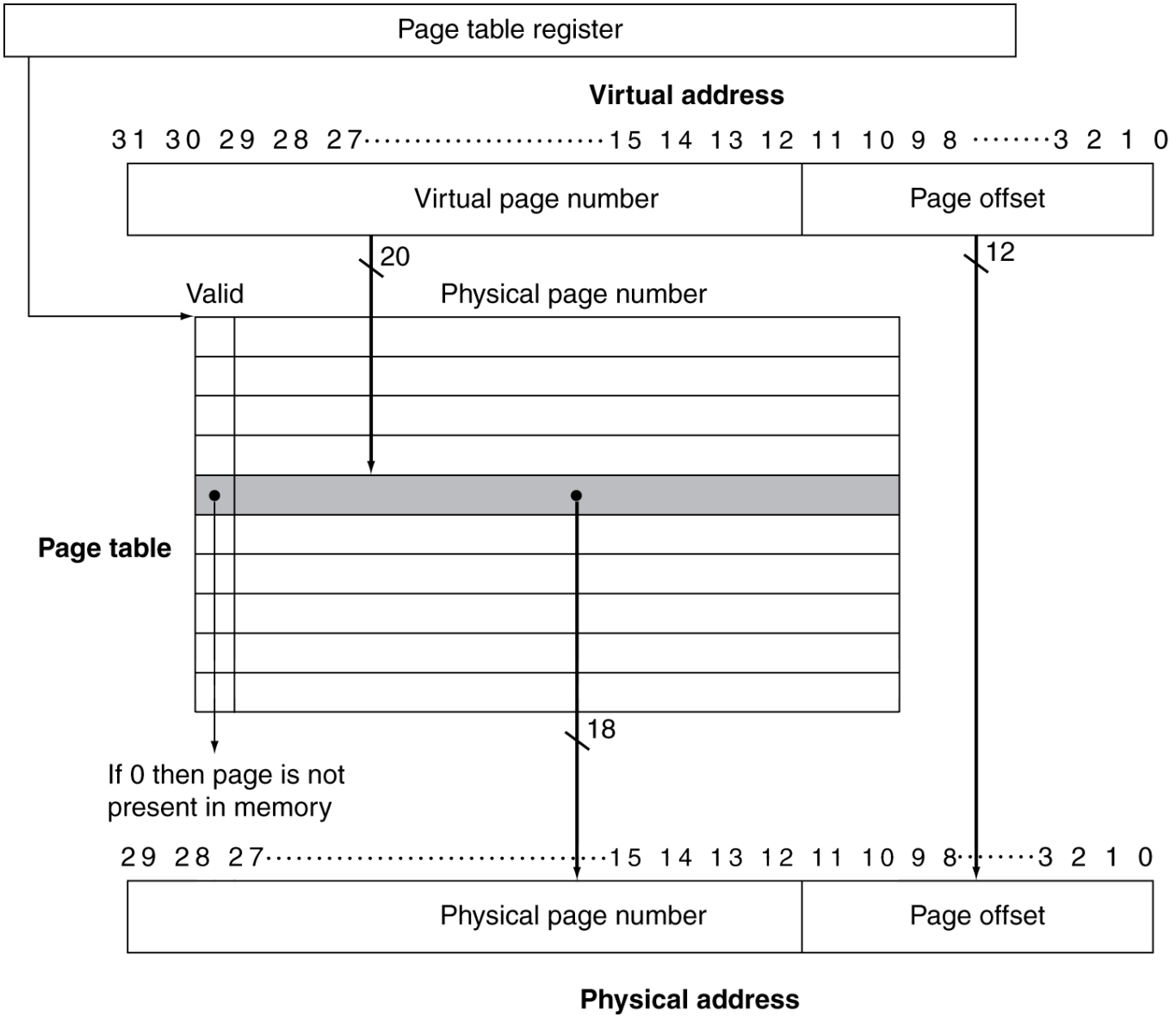
- Fixed-size pages (e.g., 4K)



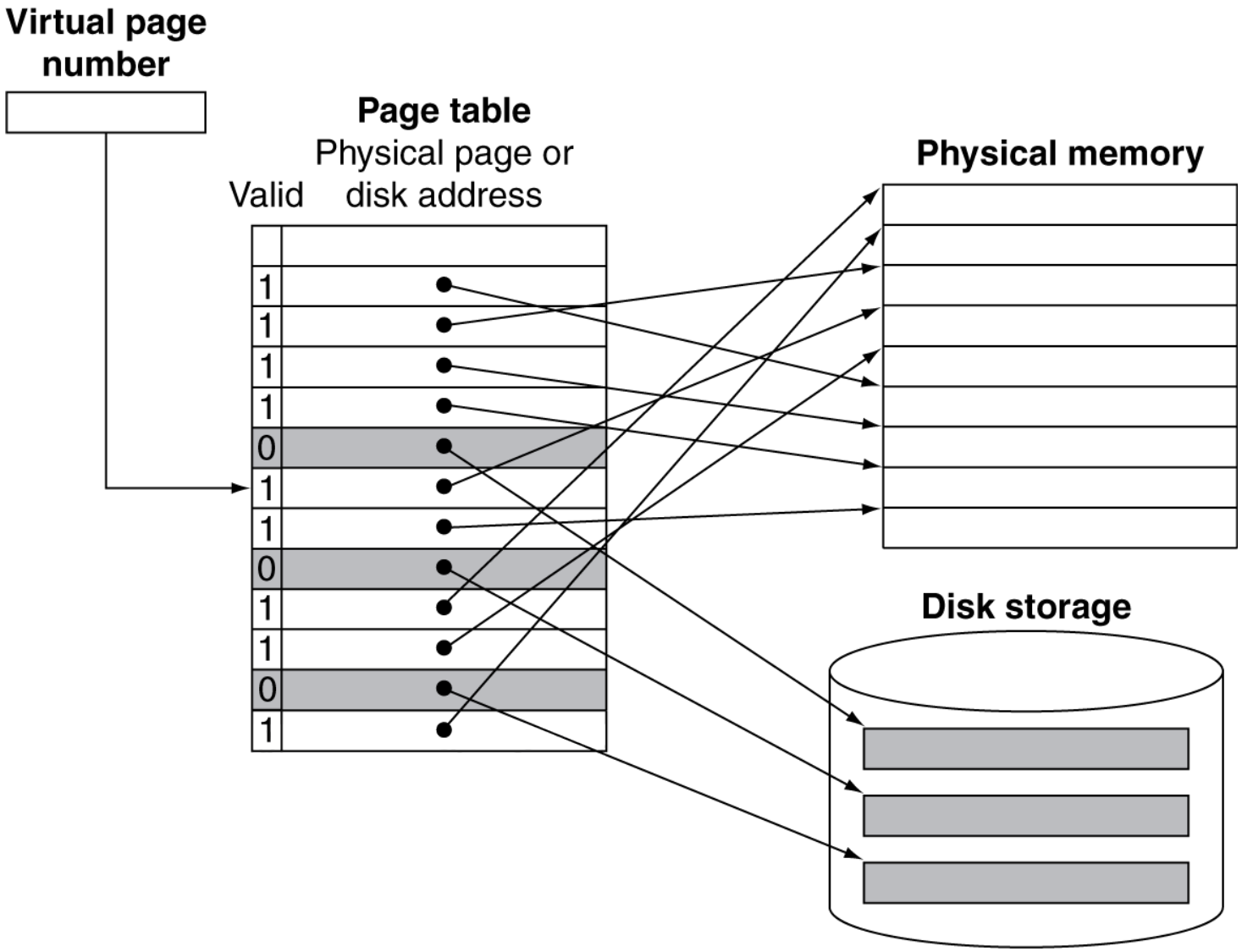


- Main memory can have 1GB while virtual address space is 4 GB

Translation Using a Page Table



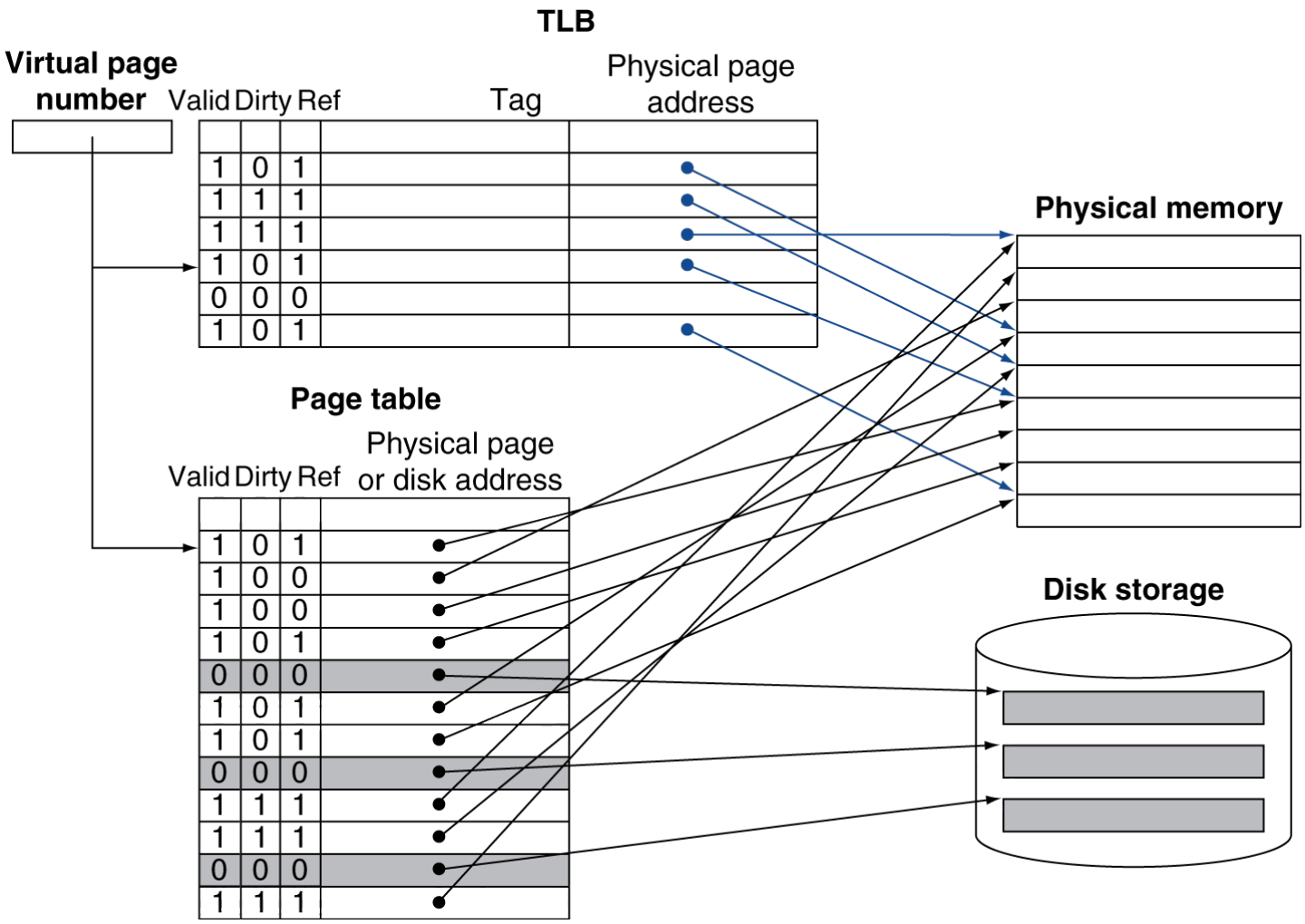
Mapping Pages to Storage



Fast Translation Using a TLB

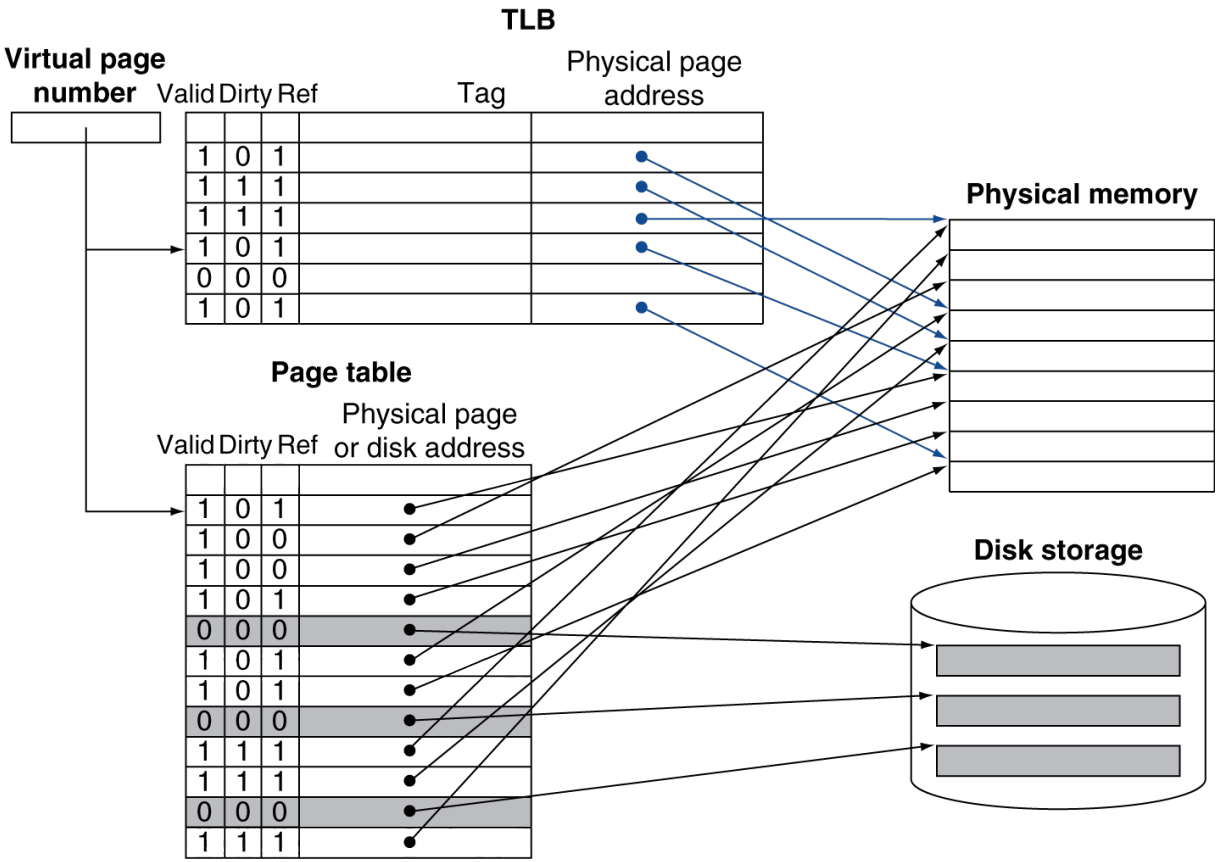
- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Small TLBs are typically fully associative, large TLBs have small associativity (due to cost issues)
  - Misses could be handled by hardware or software
    - Write back scheme for writing reference, dirty bits to PTE

Fast Translation Using a TLB



TLB Misses

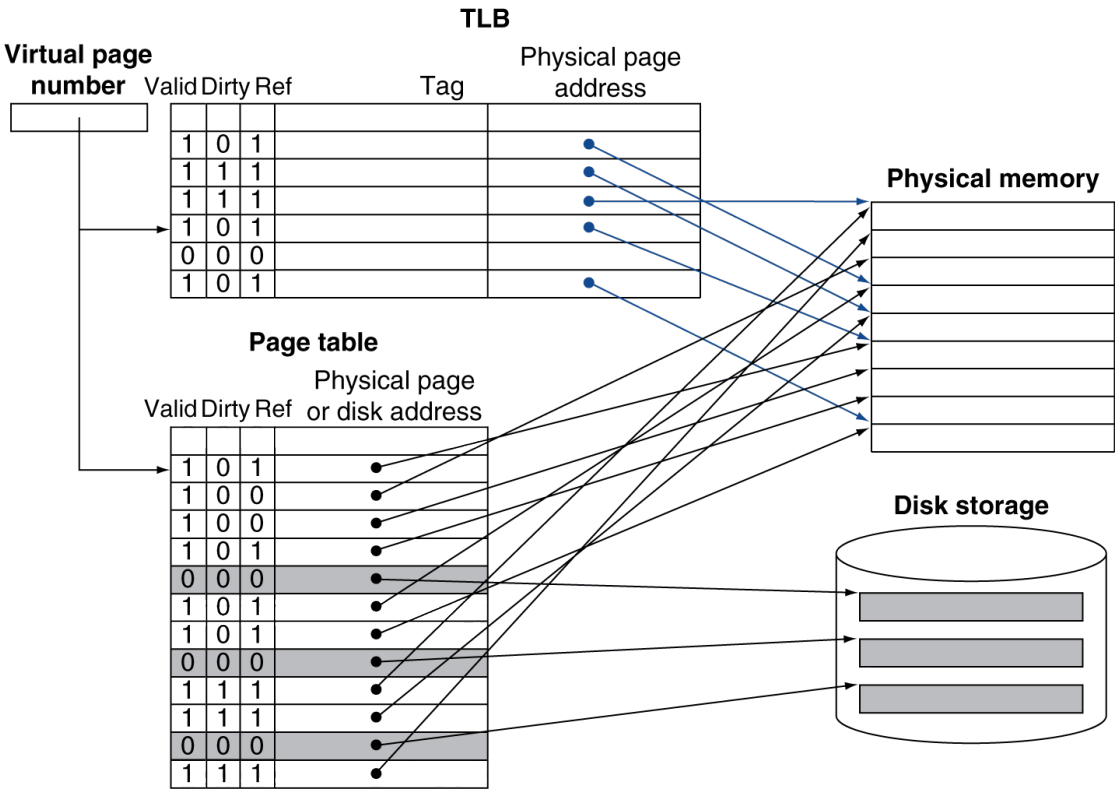
- If page is in memory
  - Load the PTE from memory and retry
  - 10's of cycles



- Could be handled in hardware
  - Can get complex for more complex page table structures
- Or in software
  - Raise a special exception, with optimized handler

TLB Misses

- If page is not in memory (page fault)
  - OS handles fetching the page and updating page table
  - Then restart the faulting instruction
  - 1,000,000's of cycles



TLB misses more frequent than page faults

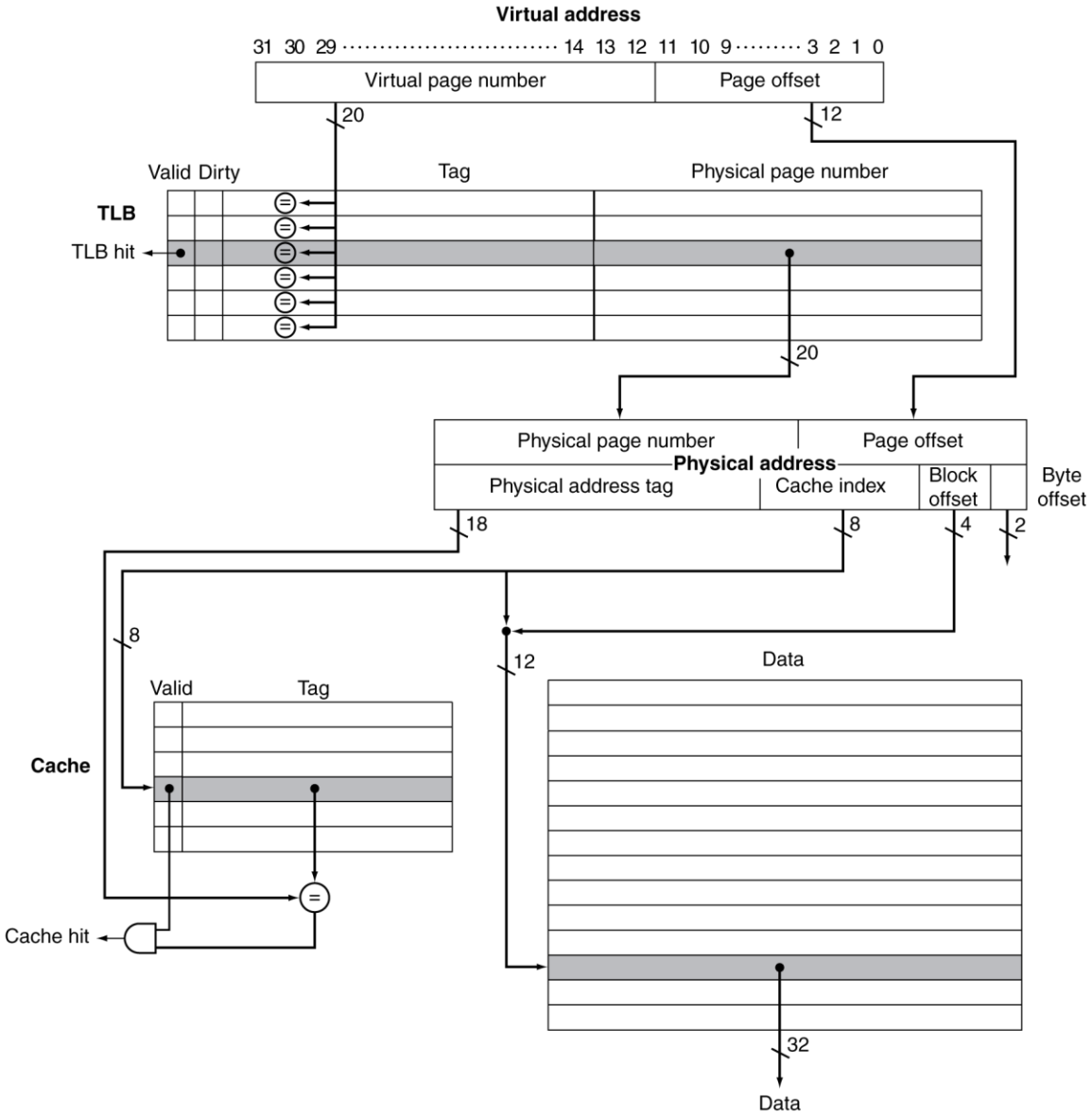
Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction

Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

TLB and Cache Interaction



- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag
  - Complications due to aliasing
    - Different virtual addresses for shared physical address

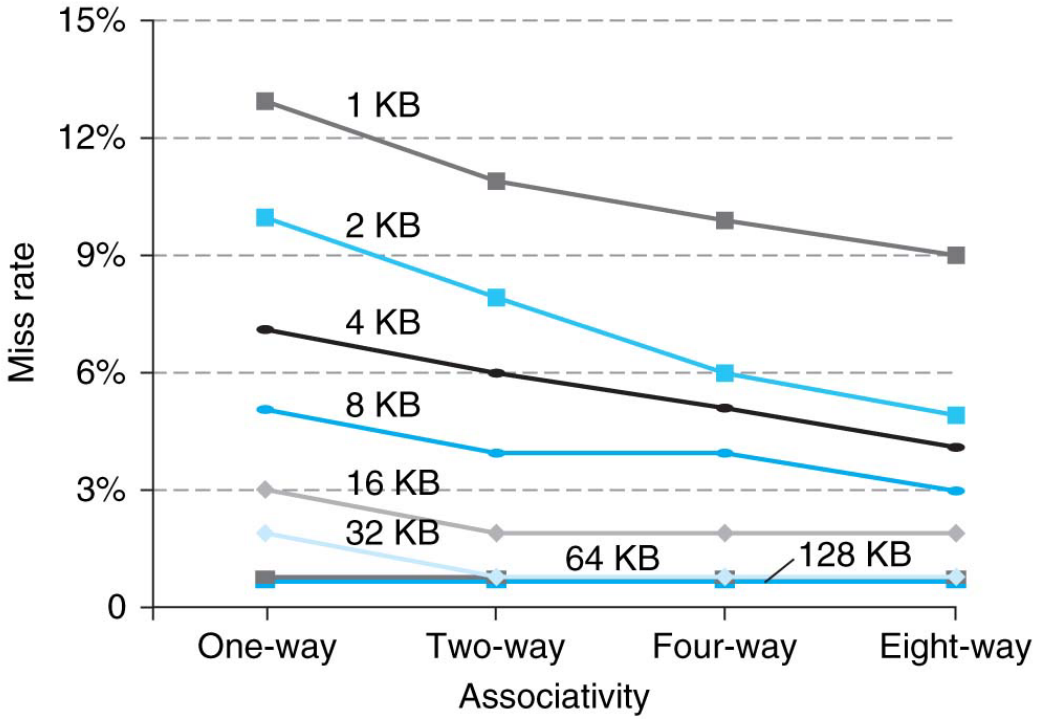
The Memory Hierarchy

The BIG Picture

- Common principles apply at all levels of the memory hierarchy
  - Based on notions of caching
- At each level in the hierarchy
  - Block placement
  - Finding a block
  - Replacement on a miss
  - Write policy

Q1. Block Placement?

- Determined by associativity
  - Direct mapped (1-way associative)
    - One choice for placement
  - n-way set associative
    - n choices within a set
  - Fully associative
    - Any location



- Higher associativity reduces miss rate
  - Increases complexity, cost, and access time

Q2. Finding a Block?

Associativity	Location method	Tag comparisons
Direct mapped	Index	1
n-way set associative	Set index, then search entries within the set	n
Fully associative	Search all entries	#entries

- Hardware caches
  - Reduce comparisons to reduce cost
  - Usually set associative or (less commonly) direct mapped
- Virtual memory systems (with page tables)
  - Almost always fully associative, as misses are very expensive
  - Software can use sophisticated replacement schemes to further reduce miss rate

Q3. Replacement on Miss?

- Choice of entry to replace on a miss
  - Least recently used (LRU)
    - Complex and costly hardware for high associativity
  - Random
    - Close to LRU, easier to implement
- Virtual memory
  - LRU approximation with hardware support

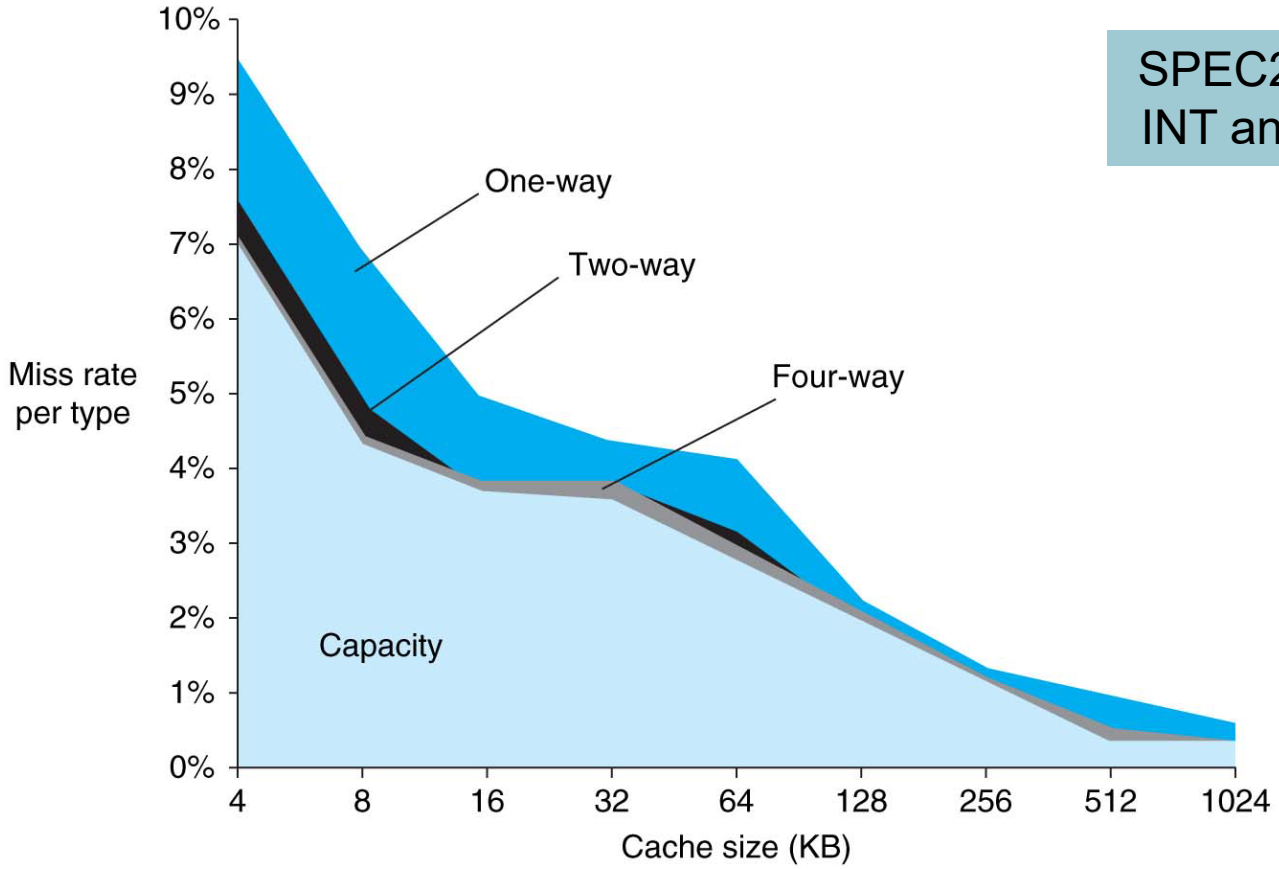
Q4. Write Policy?

- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Virtual memory
  - Only write-back is feasible, given disk write latency

Sources of Cache Misses

- Compulsory (cold start or process migration, first reference):
  - First access to a block, “cold” fact of life, not a whole lot you can do about it. If you are going to run “millions” of instruction, compulsory misses are insignificant
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- Capacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size (may increase access time)
  - Solution 2: increase associativity (may increase access time)

Sources of Cache Misses



- Compulsory misses 0.006% (not shown in graph)
- Capacity misses depend on cache size
- Conflict misses shown change with associativity

Cache Design Trade-offs: Summary

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

Branch Prediction

Reducing Branch Penalty

Branch penalty : wasted cycles due to pipeline flushing on mis-predicted branches

Reduce branch penalty:

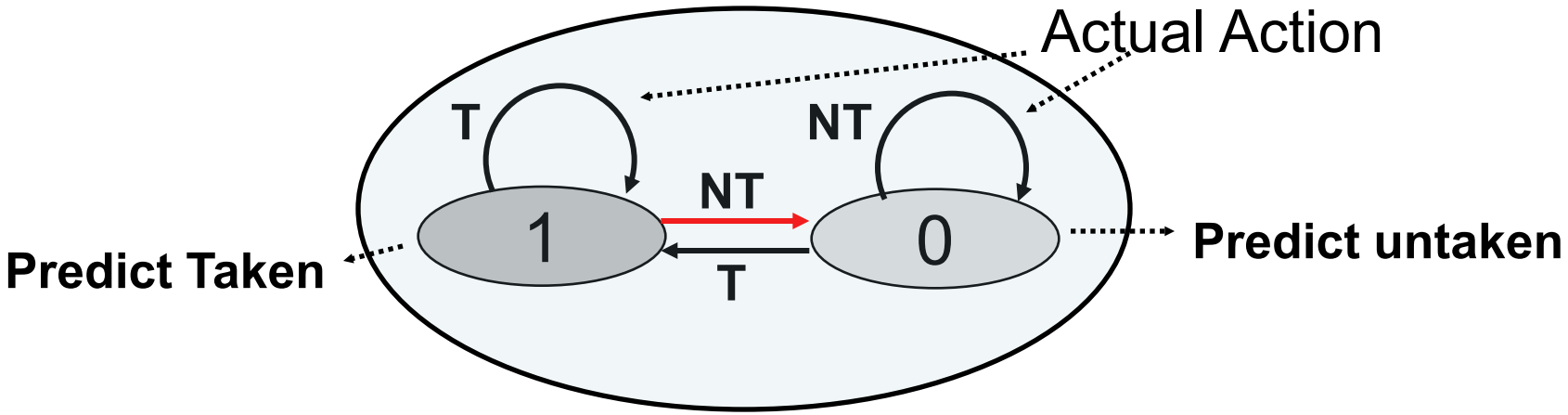
1. Predict branch/jump instructions AND branch direction (taken or not taken)
2. Predict branch/jump target address (for taken branches)
3. Speculatively execute instructions along the predicted path

Branch Prediction Strategies

- Static
  - Decided before runtime
  - Examples:
    - Always-Not Taken
    - Always-Taken
    - Backwards Taken, Forward Not Taken (BTFNT)
    - Profile-driven prediction
- Dynamic
  - Prediction decisions may change during the execution of the program

1-bit Predictor for a Single Branch

1-bit prediction



Branch-History Table (or Branch-Prediction Buffer)

- Implemented as a small memory indexed by a portion (usually some low-significant bits) of the address of the branch instruction.
  - So the size of this table is the number of entries × the number of bit per entry.
  - If unfortunately, the address portions of two branch instructions are identical, they share one entry. Hence, the more entries, the less such conflicts.

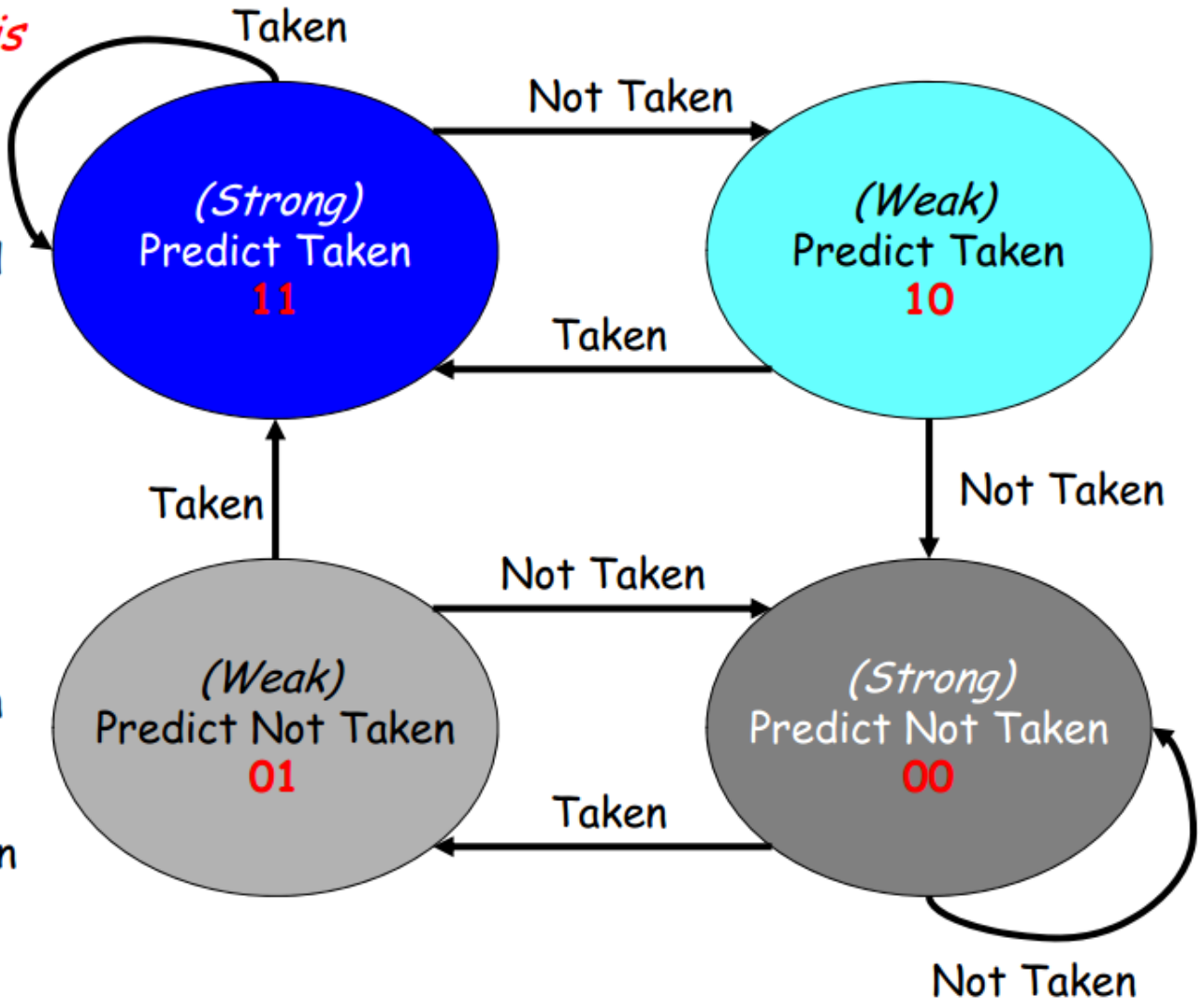
e.g., branch instructions at the following addresses share the BHT entry

- 00F2BC 0101 1100
- 010A5D 1001 1100

Index	Taken?
0000	1
0001	0
0010	1
0011	0
0100	1
0101	0
0110	1
0111	1
1000	1
1001	0
1010	1
1011	1
1100	0
1101	1
1110	1
1111	1

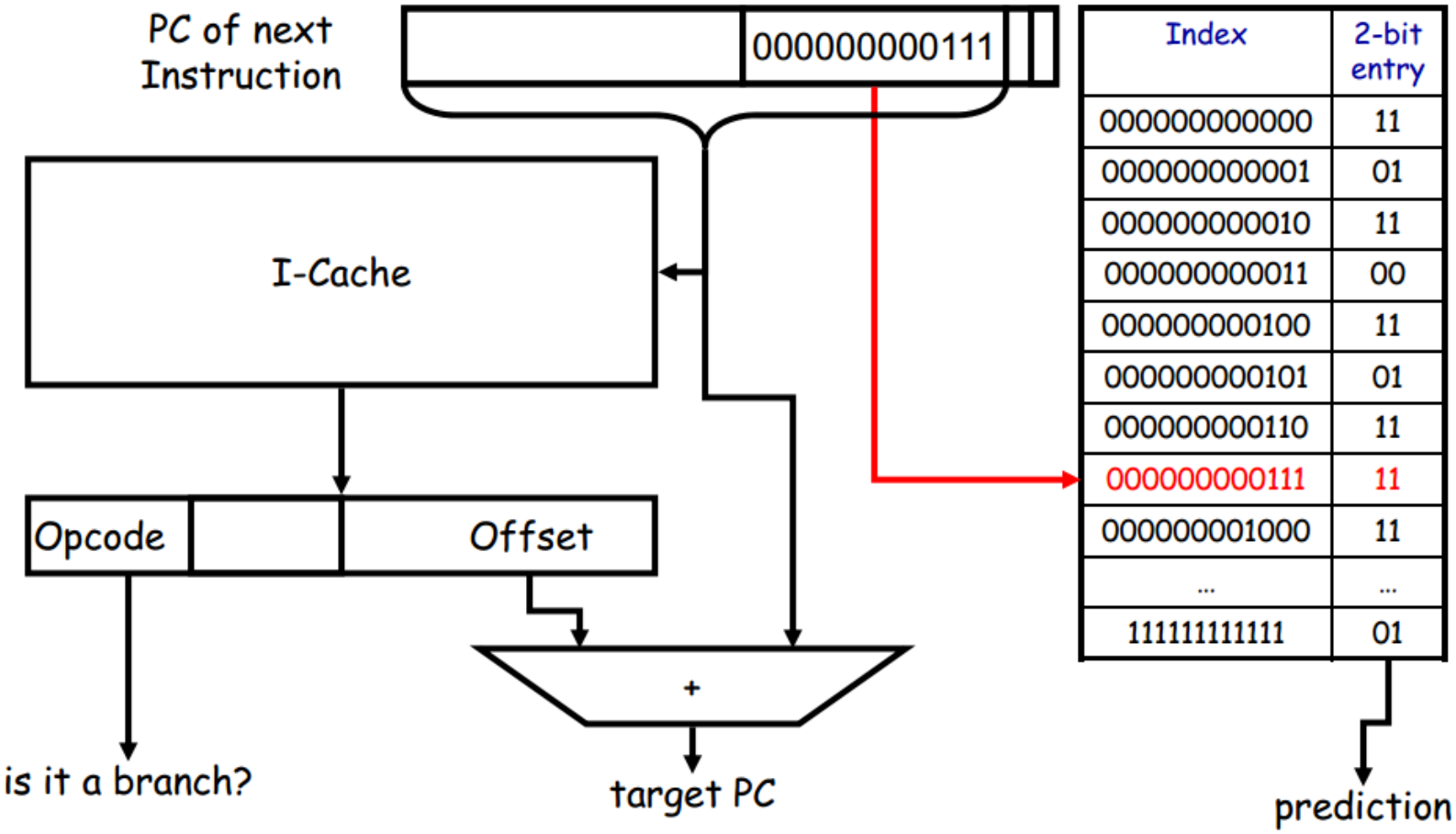
2-Bit Prediction Scheme (a.k.a. Bimodal Predictor)

- Adding *hysteresis* to the prediction scheme
  - a prediction must be missed twice before it is changed
- Implementation
  - a *(2-bit) saturated counter* that is incremented on a taken branch and decremented on an untaken branch



Example

Branch History Table for 4K-Entry 2-Bit Predictor:
4K Entry × 2 bits/Entry = 8K bit

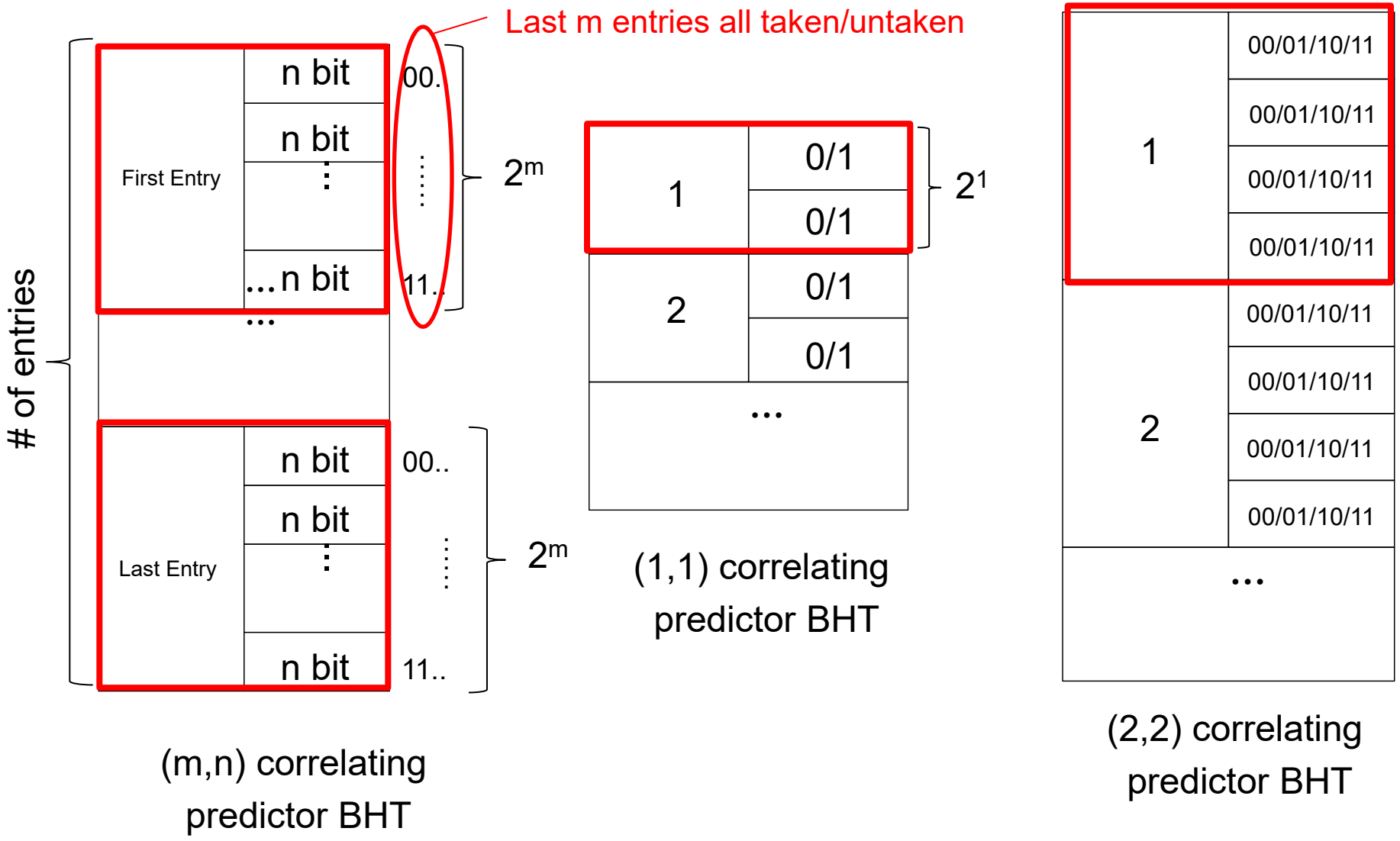


Correlating Branch Predictors

```
if (aa==2)
        aa=0;
if (bb==2)
        bb=0;
if (aa!=bb) {
```

- The limitation of the basic 2-bit predictor:
  - The 2-bit predictor schemes use only the recent behavior of **a single branch** to predict the future behavior of that branch. Increasing to 3-bit or more does not help much!
  - How about looking at the recent behavior of other branches?
  - Look the code: if the first two branches are taken, the 3rd is never taken.
- Correlating predictor / 2-level predictor:
  - Adds information of the most recent branches to decide how to predict a given branch.
  - An (m,n) 2-level predictor uses the behavior of the last m branches to choose from 2^m branch predictors, each of which is an n-bit predictor for a single branch.
  - The size of BHT = # of Entries $\times$ # of bits / Entry = # of Entries $\times$ 2^m $\times$ n

Correlating Branch Predictors



A Simple Example of (1,1) Predictor: Simulation - Cycle 0

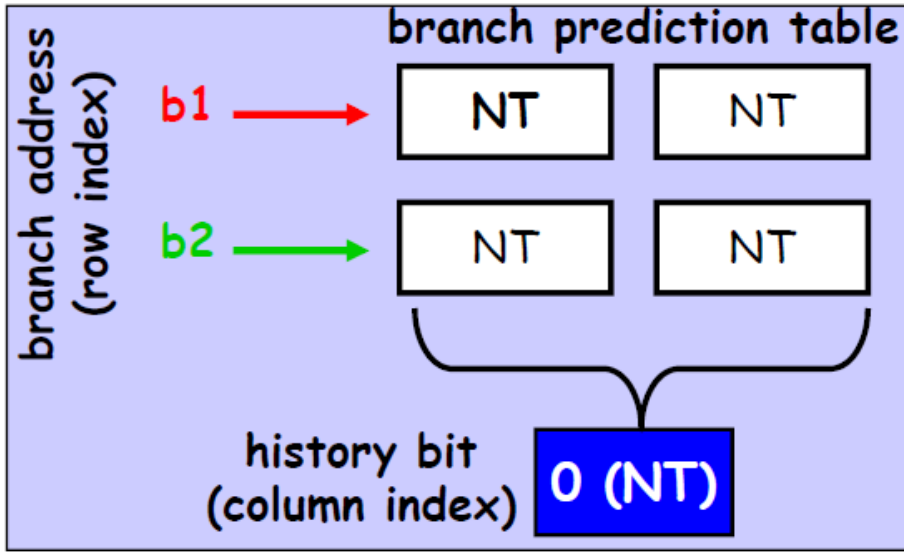
```
     BNEZ R1, L1
     DADDIU R1, R0, #1
L1:  DSUBUI R3, R1, #-1
     BNEZ R3, L2
     …
L2:  …
```

b1

b2

Assumption: d alternates between 2 and 0

X/Y: use X if last branch was not taken, use Y if last branch was taken

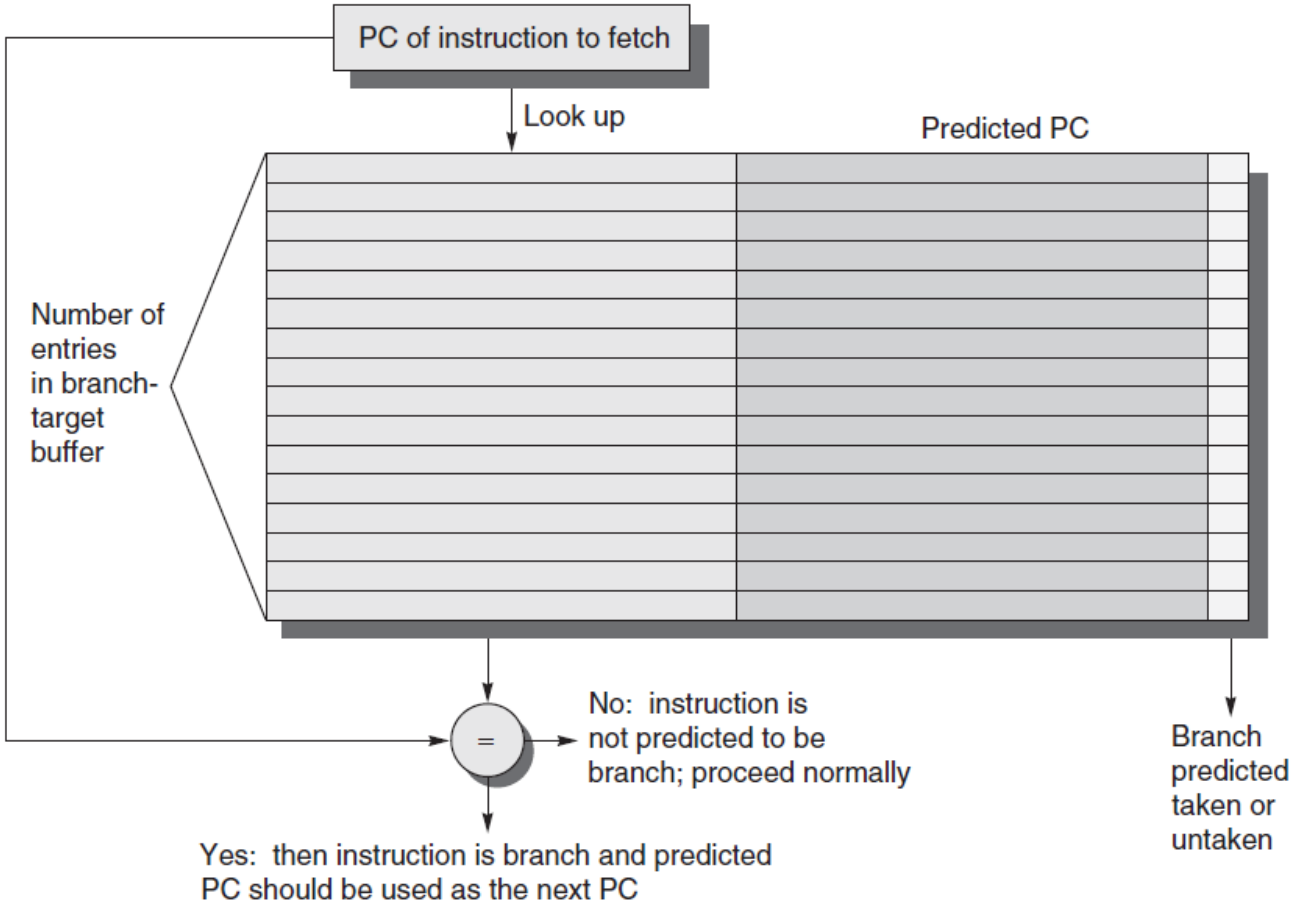


d=?	b1 pred.	b1 action	b1 new pred.	b2 pred.	b2 action	b2 new pred.
2	NT/NT	T		NT/NT	T	
0		NT			NT	
2		T			T	
0		NT			NT	

A (1,1) predictor initialized to NT/NT mispredicts only at the first iter.

Predicting the Instruction Target Address: Branch-Target Buffer (or Branch-Target Cache)

- Only necessary to store the Predicted taken branches since an untaken branch follows the same strategy (to fetch the fall-through instruction) as a non-branch instruction
- But using a 2- bit predictor requires to store information also for untaken branches



I/O

总线的基本概念

(1) 什么是总线

现代计算机系统多采用模块结构，一个模块就是一个功能部件，如主机板、显示适配器、解压卡、声卡、A/D板等。各模块之间进行信息传送的公共通路称为总线。

借助于总线连接，计算机在各功能部件间实现地址、数据和控制信息的交换，并在争用资源的基础上进行工作。

(2) 总线分类

- 按相对于CPU与其他芯片的位置: 片内总线和片外总线。
- 按总线传送信息的类别：地址总线、数据总线和控制总线。
- 按照总线传送信息的方向: 单向总线和双向总线。
- 按总线的层次结构： CPU总线、存储总线、系统总线和外部总线。

一个单处理器系统中的总线，大致分为三类：

内部总线：CPU内部连接各寄存器及运算部件之间的总线。

系统总线：CPU同计算机系统的其他高速功能部件，如存储器、通道等互相连接的总线。

I/O总线： 中、低速I/O设备之间互相连接的总线。

总线结构实例—分层的多总线结构

大多数计算机采用了分层次的多总线结构。

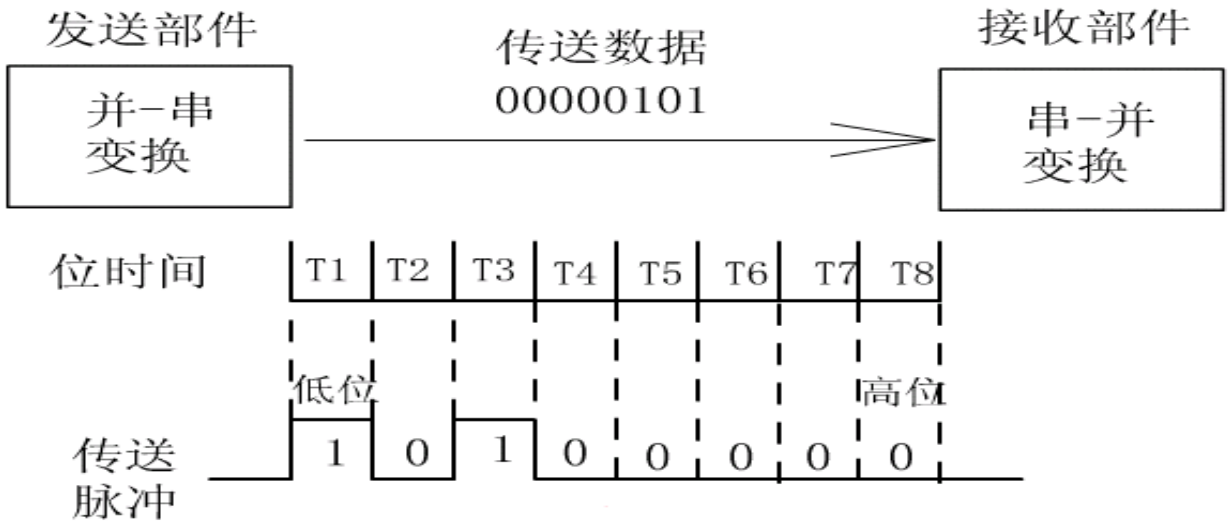
在这种结构中：

速度差异较大的设备模块使用不同速度的总线；

速度相近的设备模块使用同一类总线。

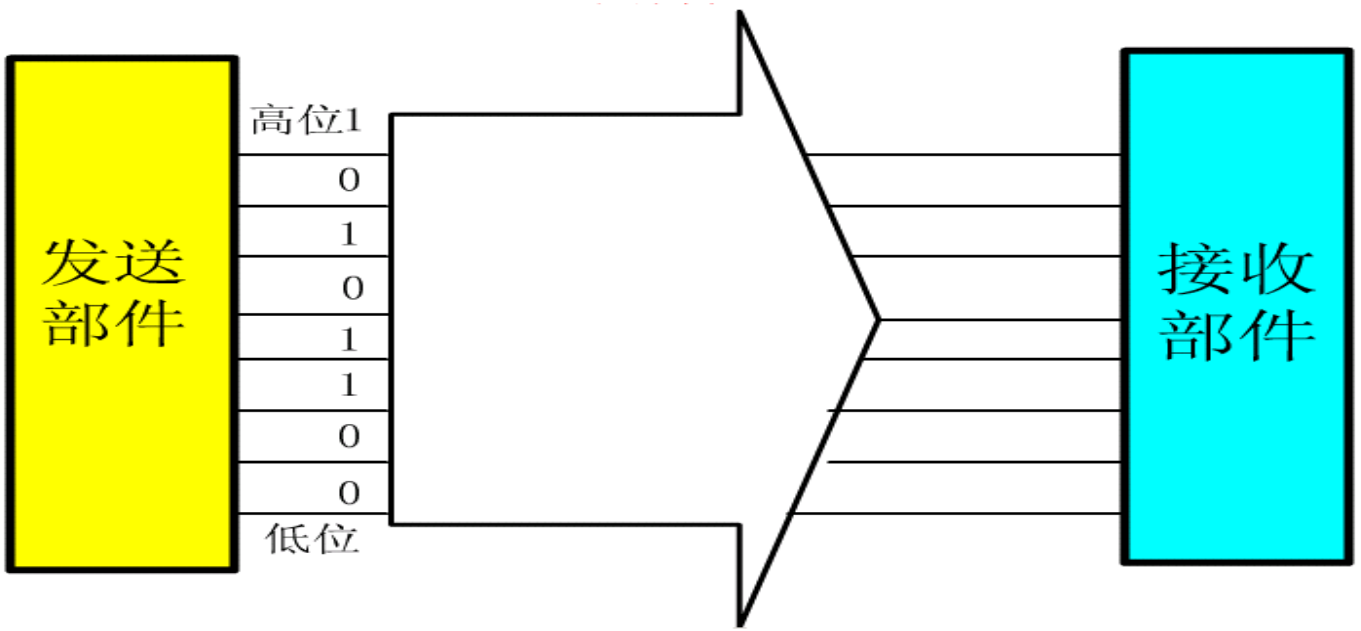
数据传输方式：串行传送

- 当信息以串行方式传送时，只有一条传输线且采用脉冲传送。
- 在串行传送时，按顺序来传送表示一个数码的所有二进制位(bit)的脉冲信号，每次一位，通常以第一个脉冲信号表示数码的最低有效位，最后一个脉冲信号表示数码的最高有效位。
- 传送时低位在前，高位在后。



并行传送

用并行方式传送二进制信息时，对每个数据位都需要单独一条传输线。信息有多少二进制位组成，就需要多少条传输线，从而使得二进制数“0”或“1”在不同的线上同时进行传送。并行传送一般采用电位传送。



分时传送

分时传送有两种概念。

① 总线复用方式

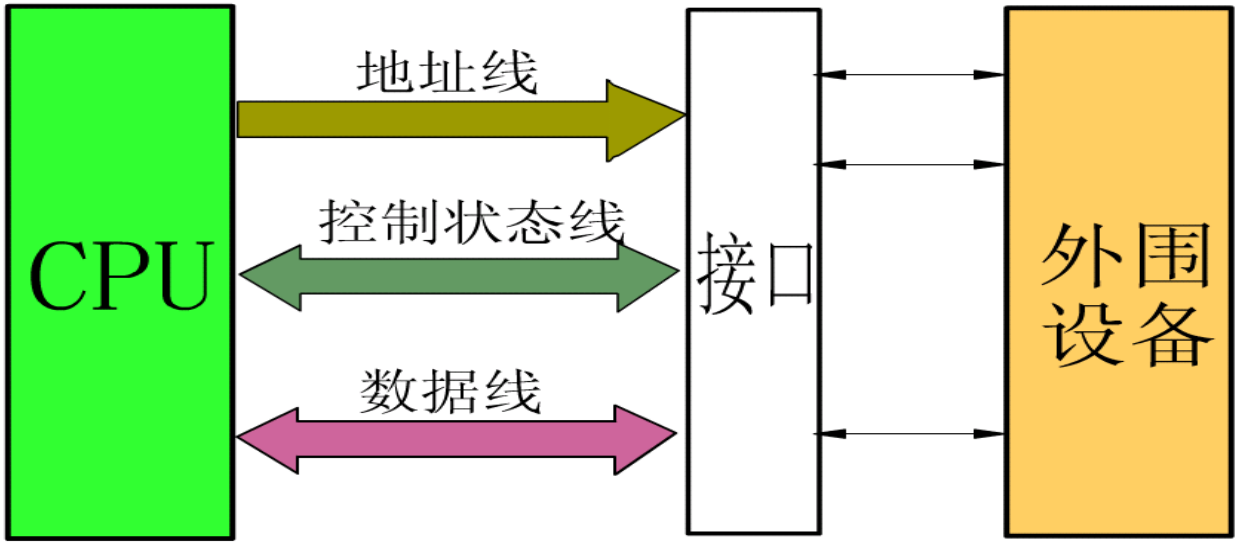
某个传输线上既传送地址信息，又传送数据信息。为此必须划分时间片，以便在不同的时间间隔中完成传送地址和传送数据的任务。

② 共享总线的部件分时使用总线

接口的基本概念

接口定义：

接口指CPU和主存、外围设备之间通过总线进行连接的逻辑部件。



外部设备包括设备控制器

接口部件在它动态连接的两个部件之间起着“转换器”的作用，以便实现彼此之间的信息传送。

接口功能

(1) 控制

接口靠程序的指令信息来控制外围设备的动作，如启动、关闭设备等。

(2) 缓冲

接口在外围设备和计算机系统其他部件之间用作为一个缓冲器，以补偿各种设备在速度上的差异。

(3) 状态

接口监视外围设备的工作状态并保存状态信息。状态信息包括数据“准备就绪”、“忙”、“错误”等等，供CPU询问外围设备时进行分析之用。

(4) 转换

接口可以完成任何要求的数据转换, 例如并－串转换或串－并转换，因此数据能在外围设备和CPU之间正确地进行传送。

(5) 整理

接口可以完成一些特别的功能，例如在需要时可以修改字计数器或当前内存地址寄存器。

(6) 程序中断

每当外围设备向CPU请求某种动作时，接口即发送一个中断请求信号到CPU。

总线仲裁、定时及数据传送模式

1. 总线的仲裁

(1) 为什么要进行总线仲裁

每次总线操作，只能有一个主方占用总线控制权，但可以有多个从方。 除CPU外，I/O模块也可提出总线请求。

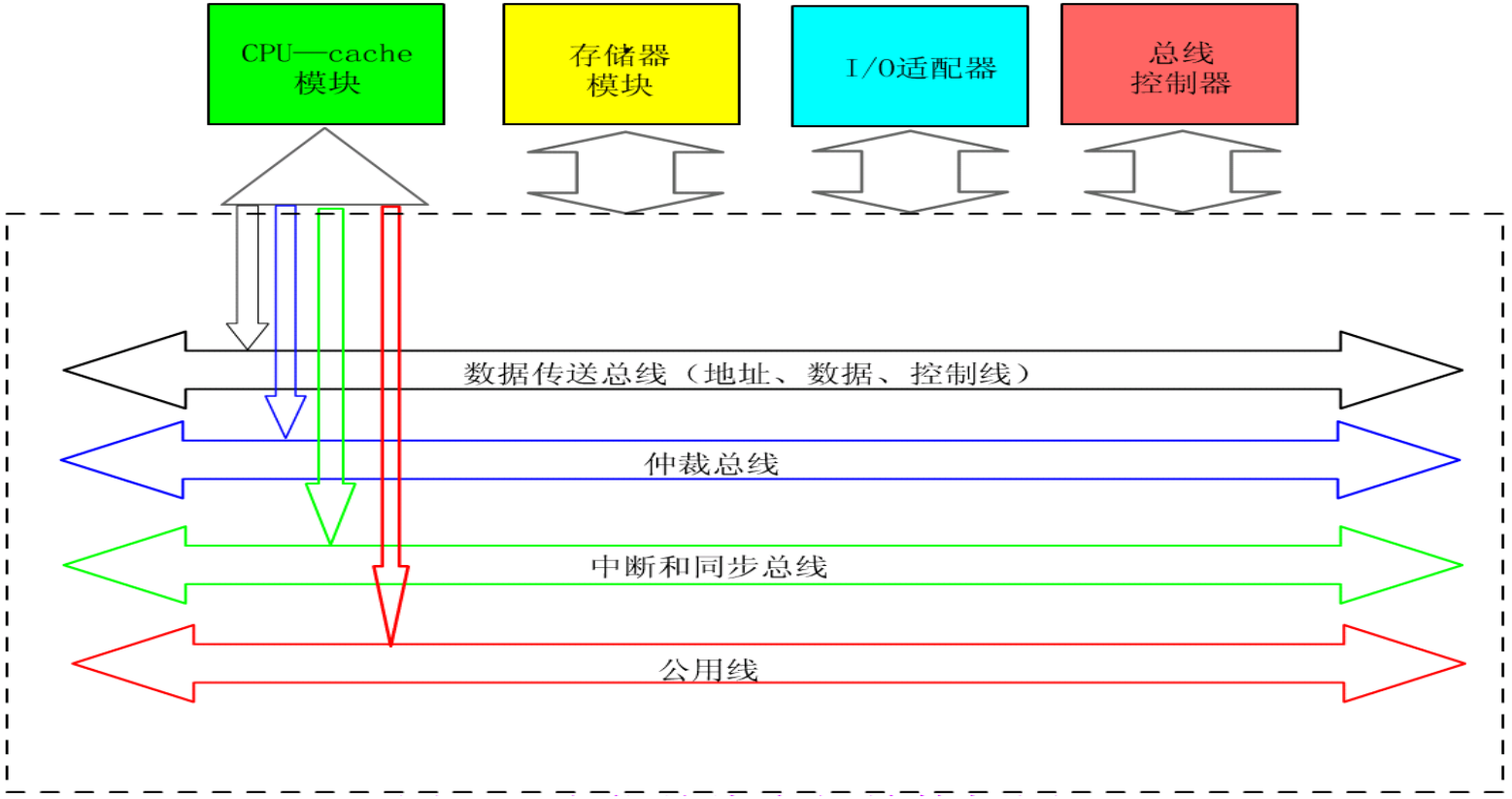
为了解决多个主设备同时竞争总线控制权，必须具有总线仲裁部件，以某种方式选择其中一个主设备作为总线的下一次主方。

(2)总线仲裁方式

仲裁方式分为集中式仲裁和分布式仲裁两类。

① 集中式仲裁

若总线仲裁逻辑集中于一个单元（中央仲裁器又称总线控制器），称为集中式仲裁。

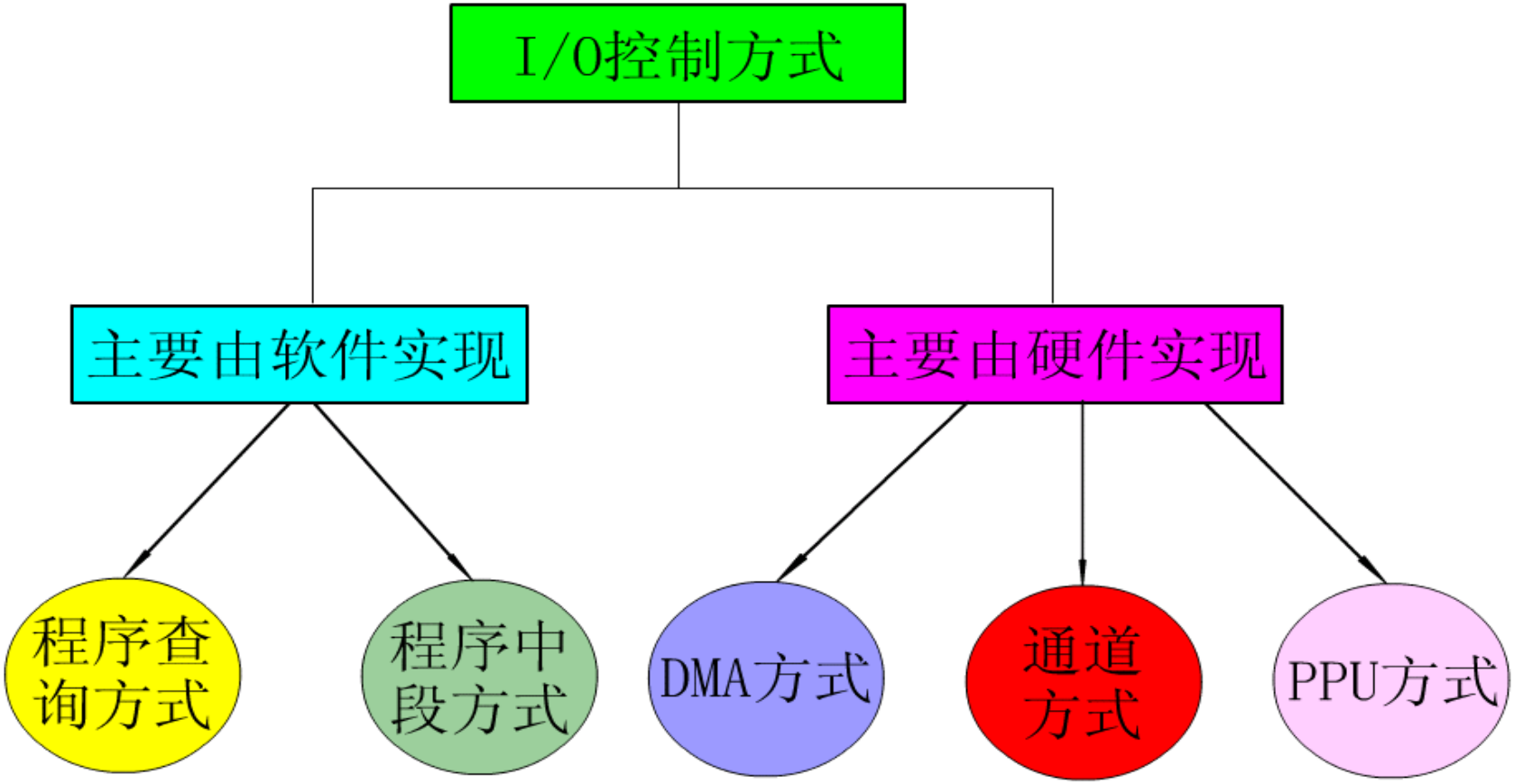


② 分布式仲裁

- 分布式仲裁不需要中央仲裁器，每个潜在的主方功能模块都有自己的仲裁号和仲裁器。
- 当它们有总线请求时，把它们唯一的仲裁号发送到共享的仲裁总线上，每个仲裁器将仲裁总线上得到的号与自己的号进行比较。
- 如果仲裁总线上的号大，则它的总线请求不予响应，并撤消它的仲裁号。
- 最后，获胜者的仲裁号保留在仲裁总线上。

分布式仲裁是以优先级仲裁策略为基础。

信息交换方式



程序查询方式特点：

1、何时对何设备输入/输出操作完全由CPU控制

2、外设与CPU处于异步工作方式

3、外设与CPU处于异步工作方式

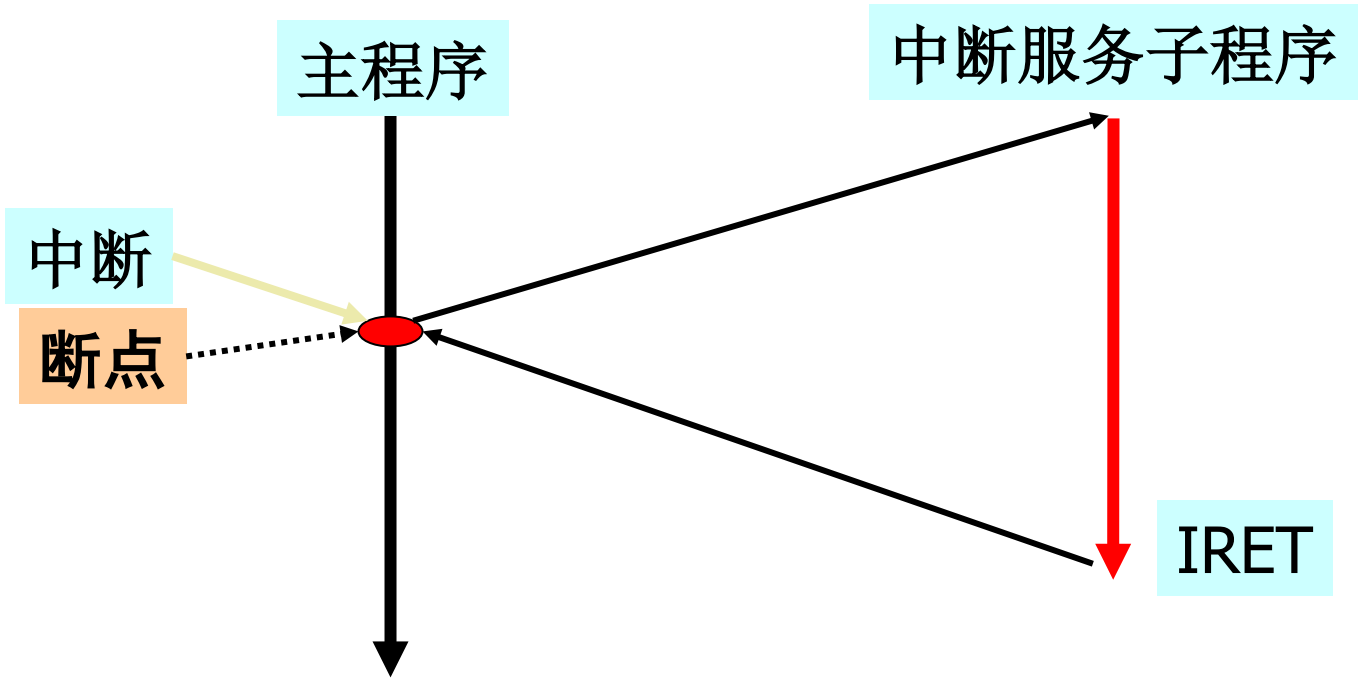
4、数据的输入/输出要经过CPU

5、CPU利用率低，但控制简单

6、实时性差

程序中断方式

当CPU正常运行程序时，由于内部事件或外设请求(随机的)，引起CPU暂时中止正在运行的程序，转去执行发出请求的外设（或内部事件）的服务子程序，待该服务程序执行完毕，再返回被中止的程序，这一过程称为中断。

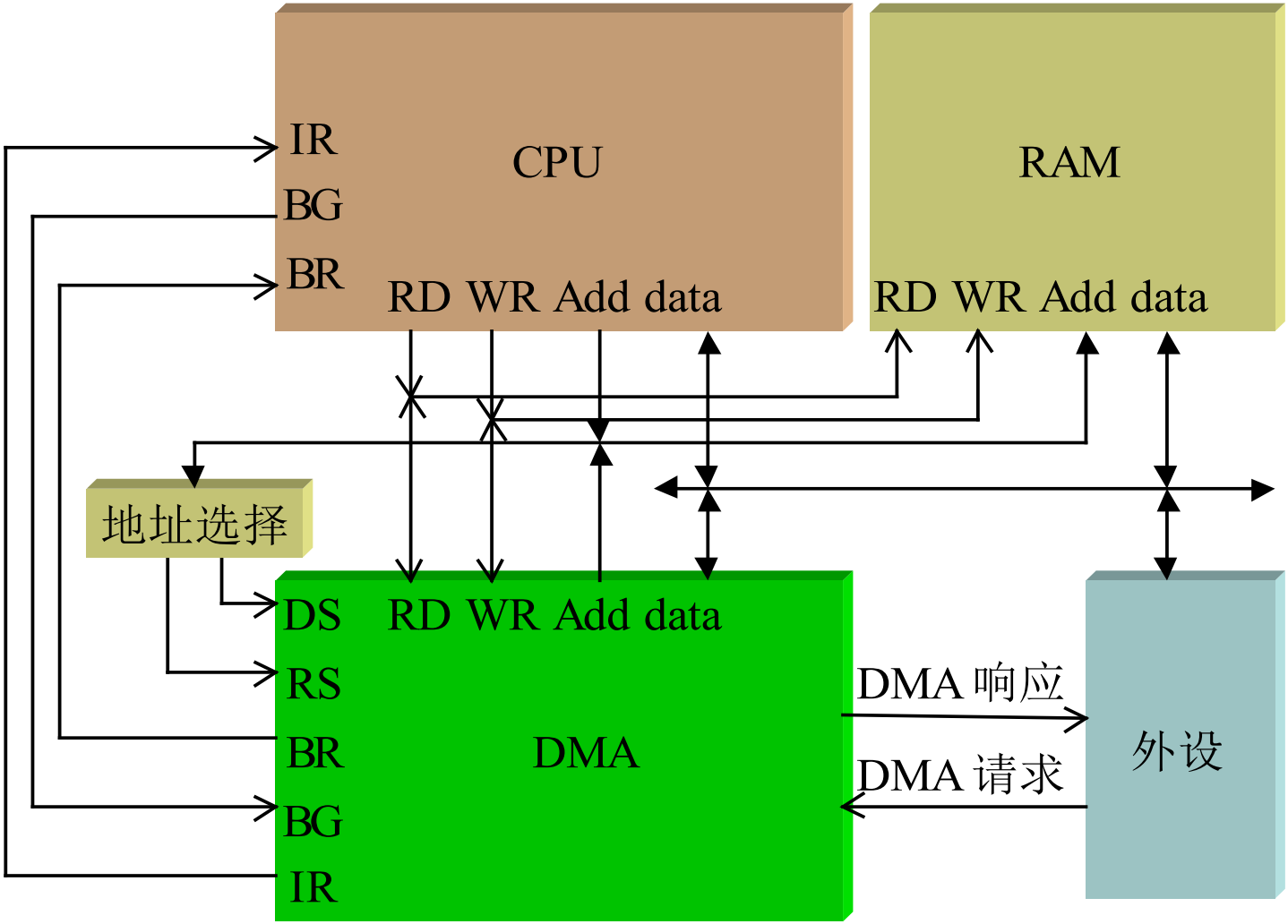


DMA方式

直接内存访问(DMA)是一种完全由硬件执行数据(I/O)交换的工作方式。在这种方式中，DMA控制器从CPU完全接管对总线的控制，数据交换不经过CPU，而直接在内存和I/O设备之间进行 。

DMA方式一般用于高速传送成组数据。DMA控制器将向内存发出地址和控制信号，修改地址，对传送的字的个数计数，并且以中断方式向CPU报告传送操作的结束。

DMA与CPU的连接


IR
BG
BR
CPU
RD WR Add data
RAM
RD WR Add data
地址选择
DS
RS
BR
BG
IR
RD WR Add data
DMA
DMA 响应
DMA 请求
外设

DMA控制器结构