北京大学
PEKING UNIVERSITY

智能硬件体系结构

第八讲：多级缓存与缓存一致性

主讲：陶耀宇、李萌

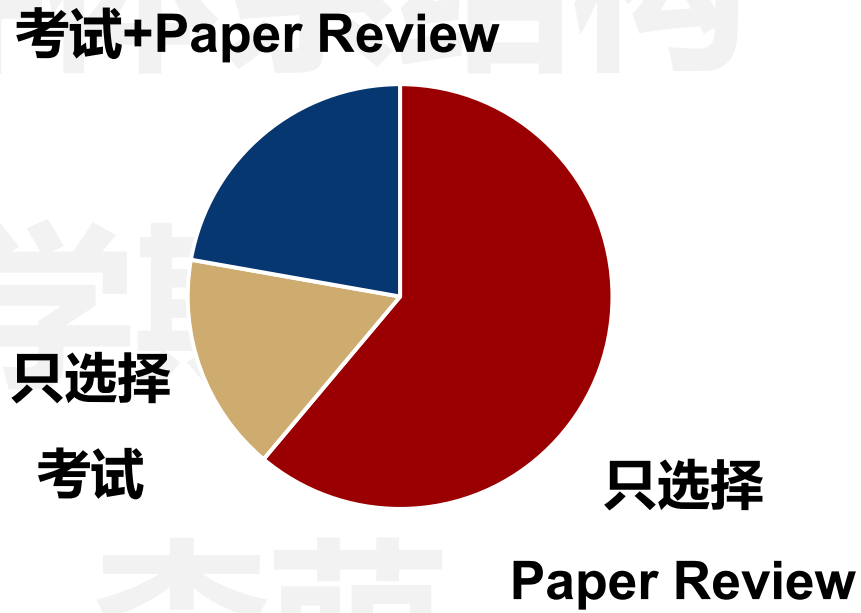
2024年秋季

注意事项

北京大学 PEKING UNIVERSITY

• 课程作业情况

- 第1次作业成绩已上传到教学网
- 第2次作业答案将在本周放出
- 第3次作业已在**11月2日**放出，**11月16日**截止（OoO等）
- 未来**3次作业（11.15-11.30、12.1-12.15、12.15-12.30）**
  - Cache设计、AI芯片架构、软硬加协同优化
- **周五2-3点安排一次Lab答疑**
- **Take Home考试**
  - **11月25中午12:00 – 11月26日凌晨11:59**
  - 每迟交一天扣10分
- **Paper Review**
  - 请自行阅读近5年内的体系结构相关论文1-2篇
  - 做10页左右的PPT



目 录

CONTENTS

MIPS：超标量+动态指令发射

- 为什么需要rewind机制

- Problem with R10K design? Precise state is more difficult
  - Physical registers are written out-of-order (at C)
  - That's OK, there is no architectural register file
  - We can "free" written registers and "restore" old ones
  - Do this by manipulating the Map Table and Free List, not regfile

- Two ways of restoring Map Table and Free List
  - Option I: serial rollback using T, T_{old} ROB fields
    ± Slow, but simple
  - Option II: single-cycle restoration from some checkpoint
    ± Fast, but checkpoints are expensive
  - Modern processor compromise: **make common case fast**
    - Checkpoint only (low-confidence) branches (frequent rollbacks)
    - Serial recovery for page-faults and interrupts (rare rollbacks)

MIPS：超标量+动态指令发射

- MIPS实例分析

Replace with a taken branch

MIPS:

Cycle 5

Precise

State

ROB

ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
h	2	jmp f0 f1 f2	PR#6	PR#3	c4	c5	
	3	stf f2,Z(r1)					
	4	addi r1,4,r1	PR#7	PR#4	c5		
t	5	ldf X(r1),f1	PR#8	PR#5			
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table

Reg	T+
f0	PR#1+
f1	PR#8
f2	PR#6
r1	PR#7

CDB

T

Free List

PR#8, PR#2

Reservation Stations

#	FU	busy	op	T	T1	T2
1	ALU	yes	addi	PR#7	PR#4+	
2	LD	yes	ldf	PR#8		PR#7
3	ST	yes	stf		PR#6	PR#4+
4	FP1	no				
5	FP2	no				

Undo insns 3-5 using roll back if insn 2 is a taken branch

MIPS：超标量+动态指令发射

- MIPS实例分析

MIPS:

Cycle 6

ROB							
ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
h	2	jmp f0 f1 f2	PR#6	PR#3	c4	c5	
	3	stf f2,Z(r1)					
t	4	addi r1,4,r1	PR#7	PR#4	c5		
	5	ldf X(r1),f1	~~PR#8~~	~~PR#5~~			
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#5+
f2	PR#6
r1	PR#7

CDB
T

Free List
PR#2 , PR#8

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	yes	addi	PR#7	PR#4+	
2	LD	no				
3	ST	yes	stf		PR#6	PR#4+
4	FP1	no				
5	FP2	no				

undo ldf (ROB#5)
1. free RS
2. free T (PR#8), return to FreeList
3. restore MT[f1] to Told (PR#5)
4. free ROB#5

insns may execute during rollback (not shown)

MIPS：超标量+动态指令发射

- MIPS实例分析

MIPS:

Cycle 7

ROB

ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
h	2	jmp f0 f1 f2	PR#6	PR#3	c4	c5	
t	3	stf f2,Z(r1)					
	4	addi r1,4,r1	PR#7	PR#4	c5		
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table

Reg	T+
f0	PR#1+
f1	PR#5+
f2	PR#6
r1	PR#4+

CDB

T

Free List

PR#2, PR#8, PR#7

Reservation Stations

#	FU	busy	op	T	T1	T2
1	ALU	no				
2	LD	no				
3	ST	yes	stf		PR#6	PR#4+
4	FP1	no				
5	FP2	no				

undo addi (ROB#4)
1. free RS
2. free T (PR#7), return to FreeList
3. restore MT[r1] to Told (PR#4)
4. free ROB#4

MIPS：超标量+动态指令发射

- MIPS实例分析

MIPS:

Cycle 8

ROB							
ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
ht	2	jmp f0 f1 f2	PR#6	PR#3	c4	c5	
	3	stf f2,Z(r1)					
	4	addi r1,4,r1					
	5	ldf X(r1),f1					
	6	mulf f0,f1,f2					
	7	stf f2,Z(r1)					

Map Table	
Reg	T+
f0	PR#1+
f1	PR#5+
f2	PR#6
r1	PR#4+

CDB
T

Free List
PR#2, PR#8, PR#7

Reservation Stations						
#	FU	busy	op	T	T1	T2
1	ALU	no				
2	LD	no				
3	ST	no				
4	FP1	no				
5	FP2	no				

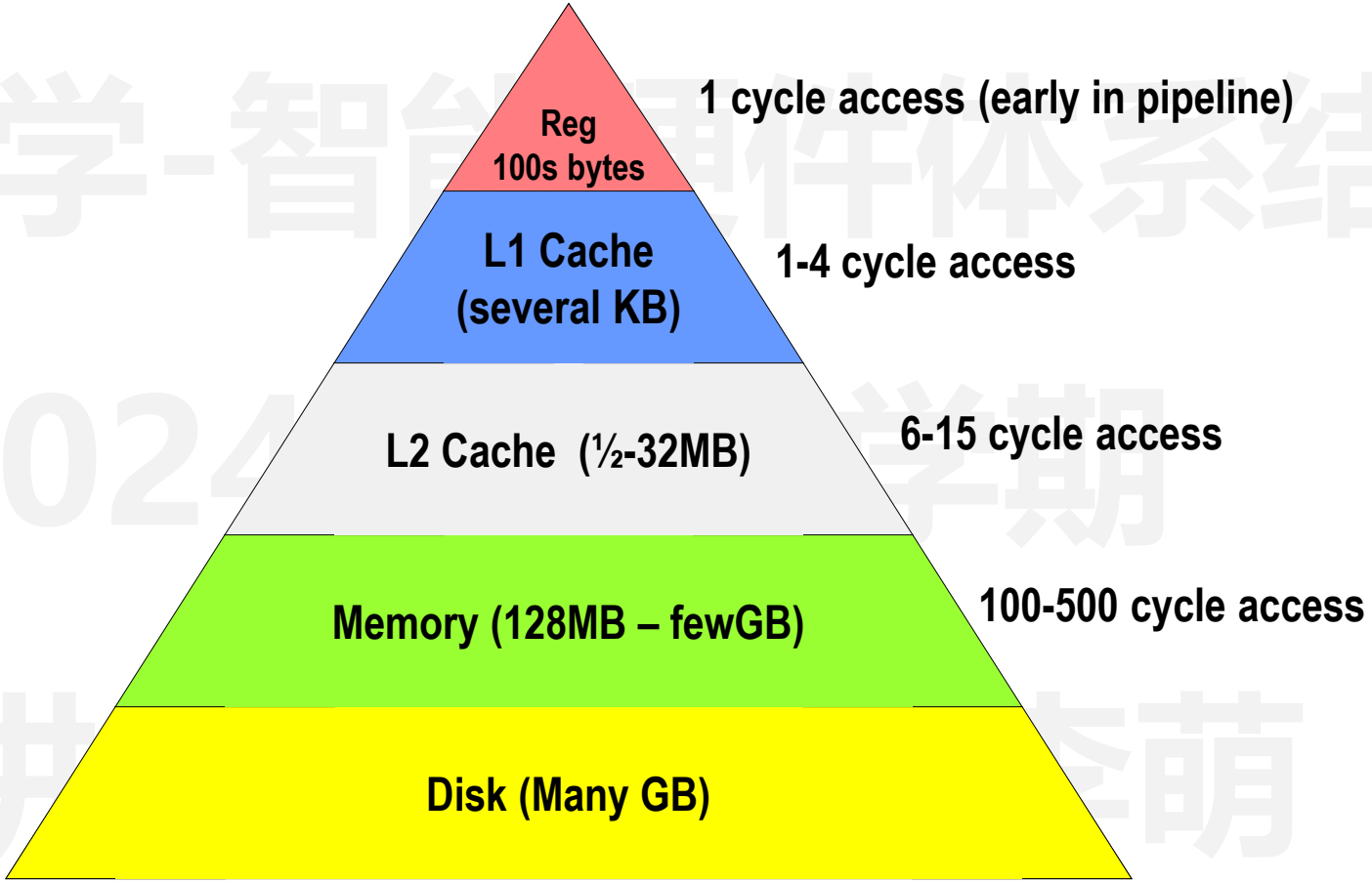
undo stf (ROB#3)
1. free RS
2. free ROB#3
3. no registers to restore/free
4. how is D$ write undone?

目 录

CONTENTS

缓存Cache设计基础

- 当前芯片架构中的缓存层级



缓存Cache设计中的局部性

北京大学 PEKING UNIVERSITY

- **局部性原理 – 时间局部性与空间局部性**

- Principle of Locality:
  - Programs tend to reuse data and instructions near those they have used recently.
  - *Temporal locality:* recently referenced items are likely to be referenced in the near future.
  - *Spatial locality:* items with nearby addresses tend to be referenced close together in time.

Locality in Example:

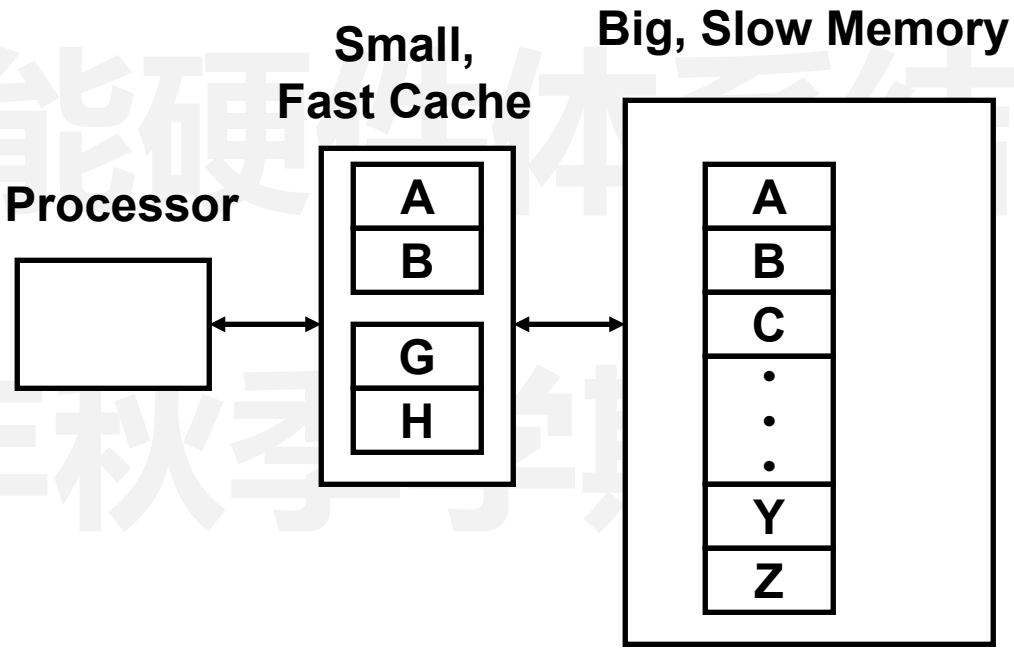
- **Data**
  - Reference array elements in succession (spatial)
- **Instructions**
  - Reference instructions in sequence (spatial)
  - Cycle through loop repeatedly (temporal)

```
sum = 0;
for (i = 0; i < n; i++)
        sum += a[i];
*v = sum;
```

缓存Cache的基本思路

- 加快访存速度

- **Main Memory**
  - Stores words
    A–Z in example
- **Cache**
  - Stores subset of the words
    4 in example
  - Organized in lines
    - Multiple words
    - To exploit spatial locality
- **Access**
  - Word must be in cache for processor to access

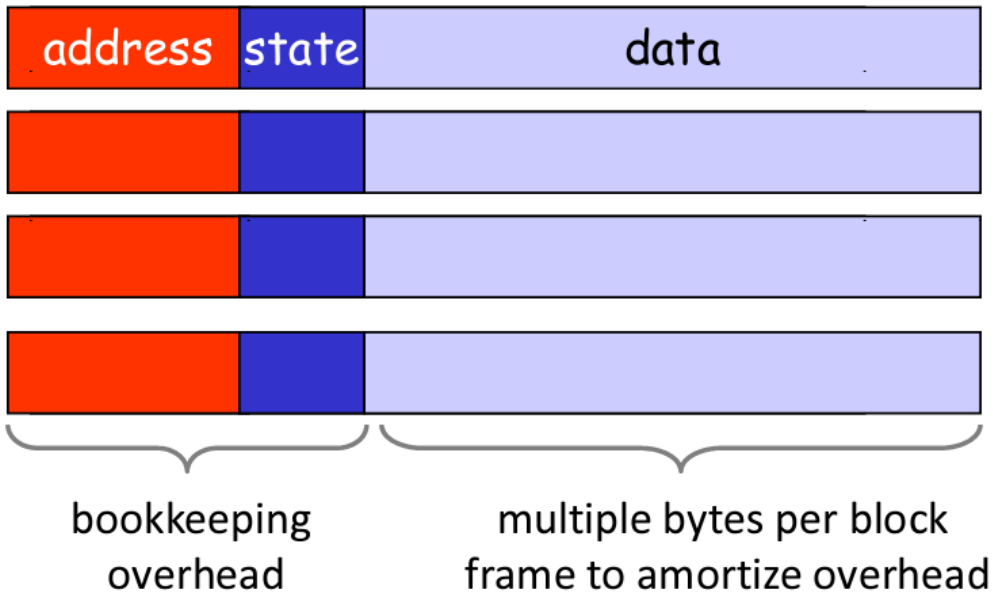


缓存Cache的基本思路

- Cache的抽象模型

Keep recently accessed block in “block frame”

- state (e.g., valid)
- address tag
- data



On memory read

- if incoming address corresponds to on e of the stored address tag then
  - **HIT**
  - return data
- else
  - **MISS**
  - Choose and displace a current block in use
  - fetch new (referenced) block from memory into frame
  - return data

缓存Cache的基本思路

北京大学 PEKING UNIVERSITY

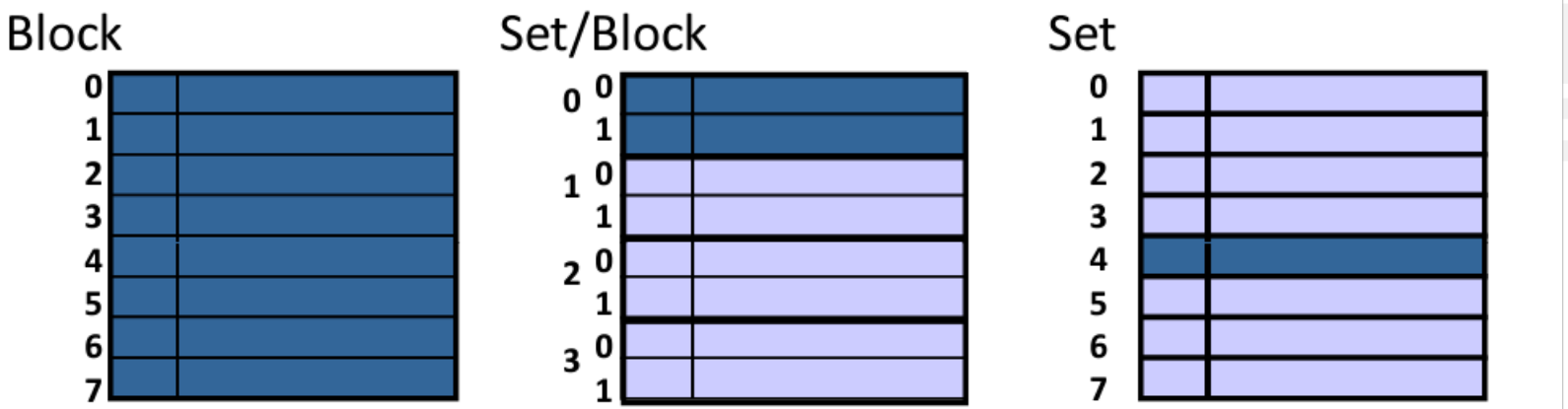
- Cache使用术语

  - **block (cache line)** — minimum unit that may be present
  - **hit** — block is found in the cache
  - **miss** —block is not found in the cache
  - **miss ratio** — fraction of references that miss
  - **hit time** —time to access the cache miss penalty
  - **miss penalty**
    - time to replace block in the cache + deliver to upper level
    - access time — time to get first word
    - transfer time — time for remaining words

缓存Cache的基本思路

- **Cache Block Placement**

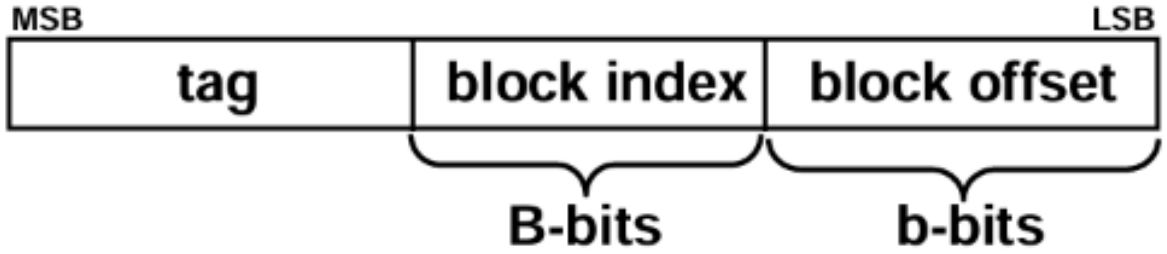
Where does block 12 (b'1100) go?



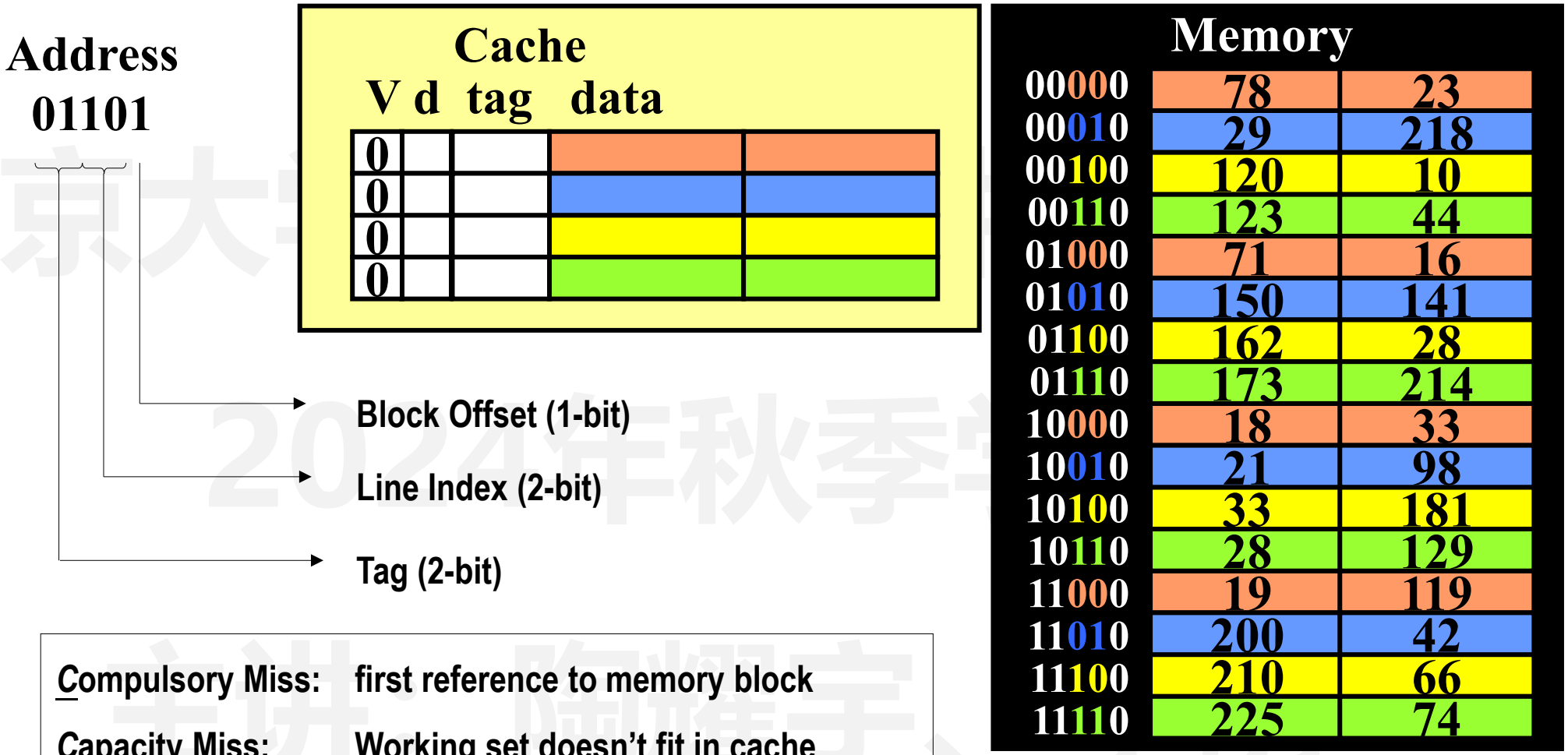
Fully-associative	Set-associative	Direct-mapped
block goes in any frame	a block goes in any frame in exactly one set	block goes in exactly one frame
(think all frames in 1 set)	(frames grouped into sets)	(think 1 frame per set)

Cache Block Size的概念

- Each cache block frame or (cache line) has only one tag but can hold multiple “chunks” of data
  - **Reduce tag storage overhead**
    - In 32-bit addressing, an 1-MB direct-mapped cache has 12 bits of tags
      - 4-byte cache block ⇒ 256K blocks ⇒ ~384KB of tag
      - 128-byte cache block ⇒ 8K blocks ⇒ ~12KB of tag
  - **The entire cache block is transferred to and from memory all at once**
    - good for spatial locality because if you access address i you will probably want i+1 as well (prefetching effect)
- Block size = 2^b; Direct Mapped Cache Size = 2^(B+b)



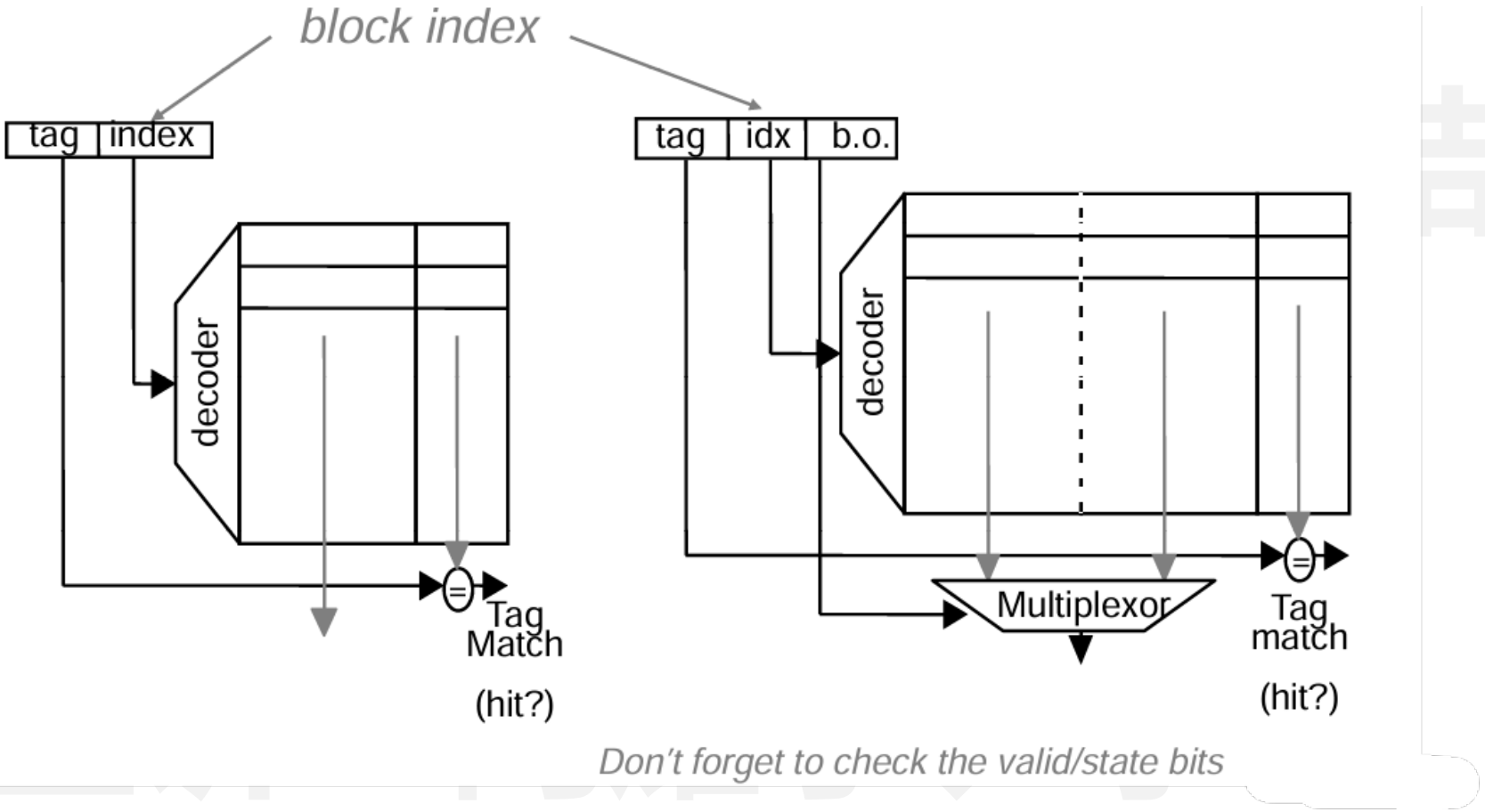
Direct-Mapped Cache设计



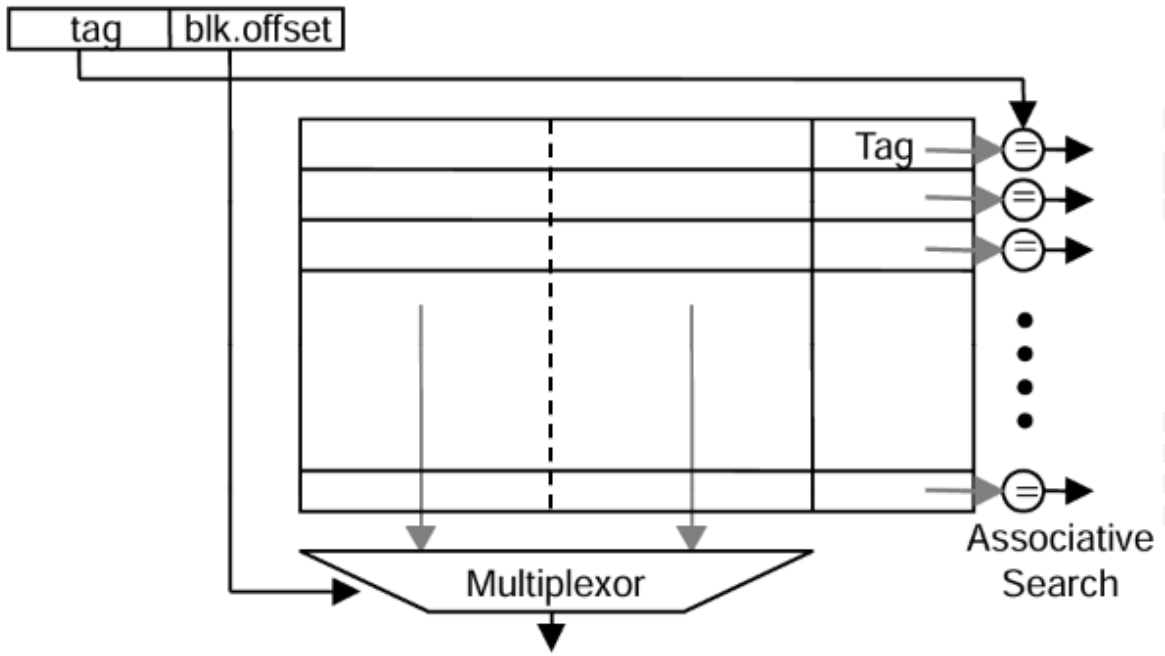
3-C's

Compulsory Miss:	first reference to memory block
Capacity Miss:	Working set doesn't fit in cache
Conflict Miss:	Working set maps to same cache line

Direct-Mapped Cache设计

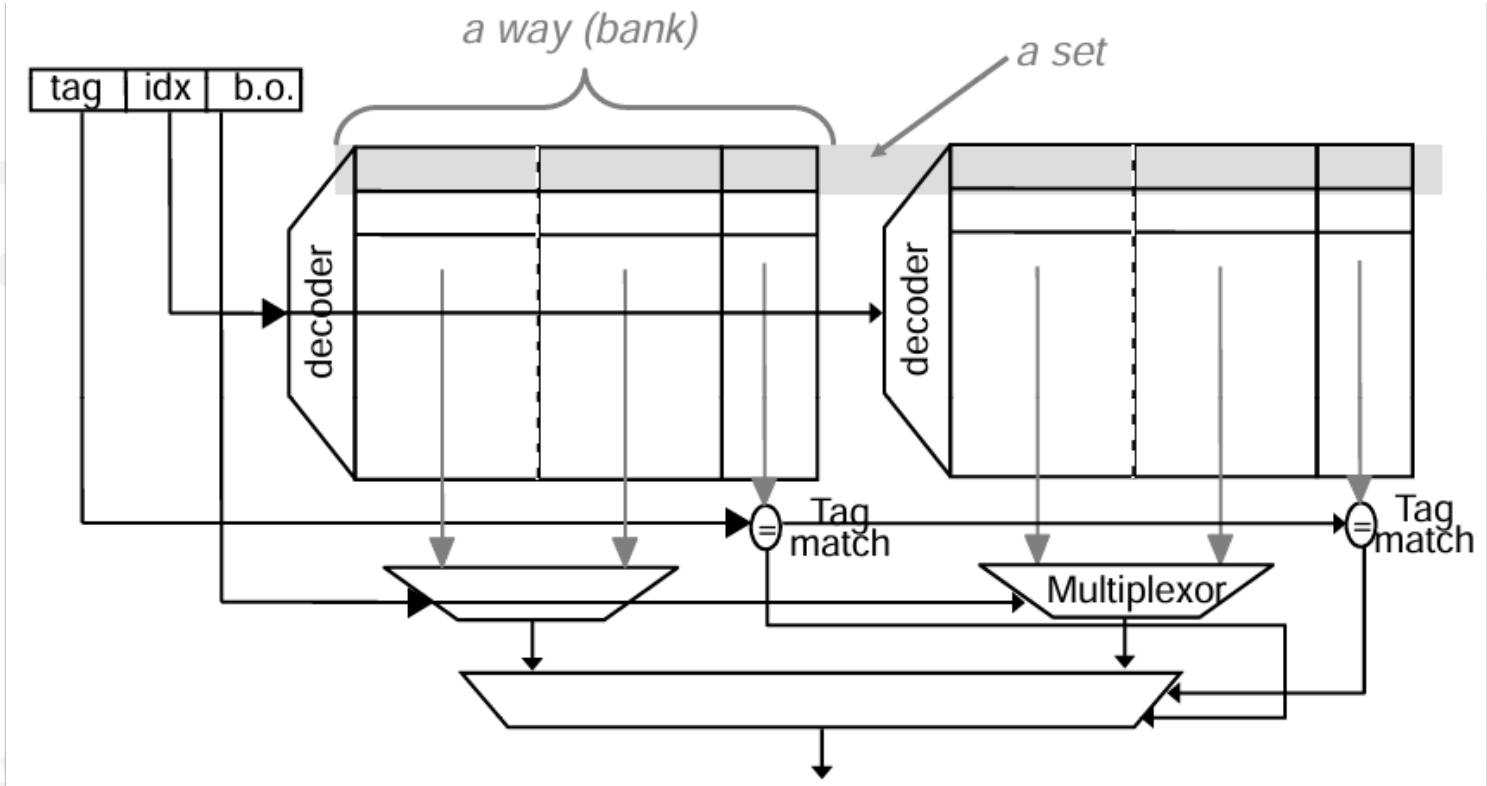


Fully-Associative Cache设计



没有block index

N-Way Set Associative Cache设计



N-Way Set Associative Cache设计

北京大学 PEKING UNIVERSITY

- **Associative Block Replacement**
  - Which block in a set to replace on a miss?
    - Ideally — replace the block that "will" be accessed the furthest in the future
      - How do you implement it?
    - Approximations:
      - **Least recently used — LRU**
        - optimized (assume) for temporal locality (expensive for more than 2-way)
      - **Not most recently used — NMRU**
        - track MRU, random select from others, good compromise
      - **Random**
        - **nearly as good as LRU, simpler (usually pseudo-random)**
        - How much can block replacement policy matter?

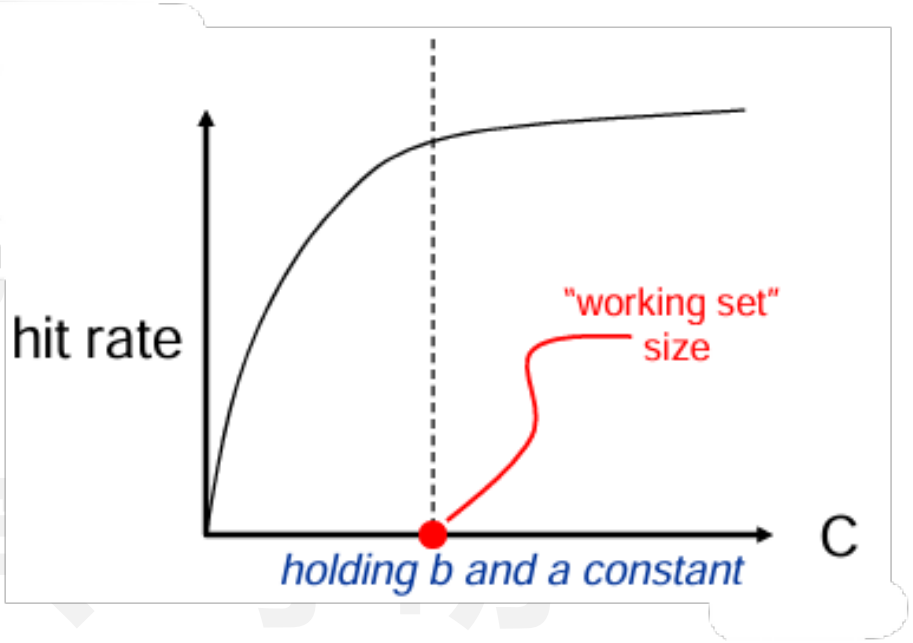
Cache Miss种类

- Miss Classification (3+1 C’s)
  - **Compulsory Miss**
    - “cold miss” on first access to a block
      - —defined as: miss in infinite cache
  - **Capacity Miss**
    - misses occur because cache not large enough — defined as: miss in fully-associative cache
  - **Conflict Miss**
    - misses occur because of restrictive mapping strategy
    - only in set-associative or direct-mapped cache
      - —defined as: not attributable to compulsory or capacity
  - **Coherence Miss**
    - misses occur because of sharing among multiprocessors

Cache参数的选择

- Cache Size（C）

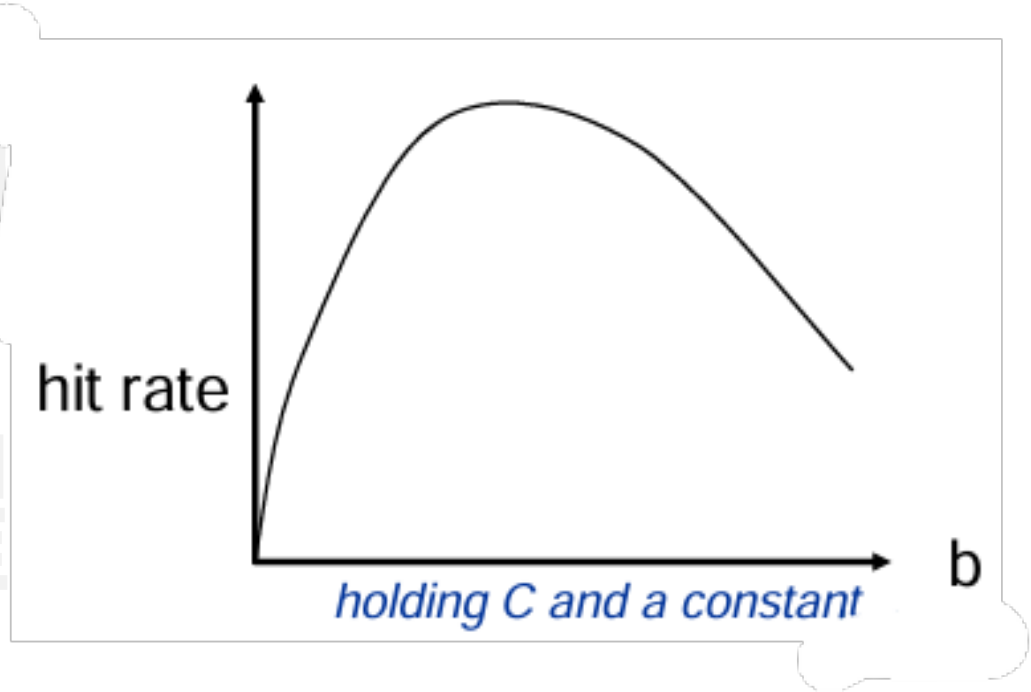
- **Cache size is the total data (not including tag) capacity**
  - bigger can exploit temporal locality better
  - not ALWAYS better
- **Too large a cache**
  - smaller is faster => bigger is slower
  - access time may degrade critical path
- **Too small a cache**
  - don’t exploit temporal locality well
  - useful data constantly replaced



Cache参数的选择

- Block Size（b）

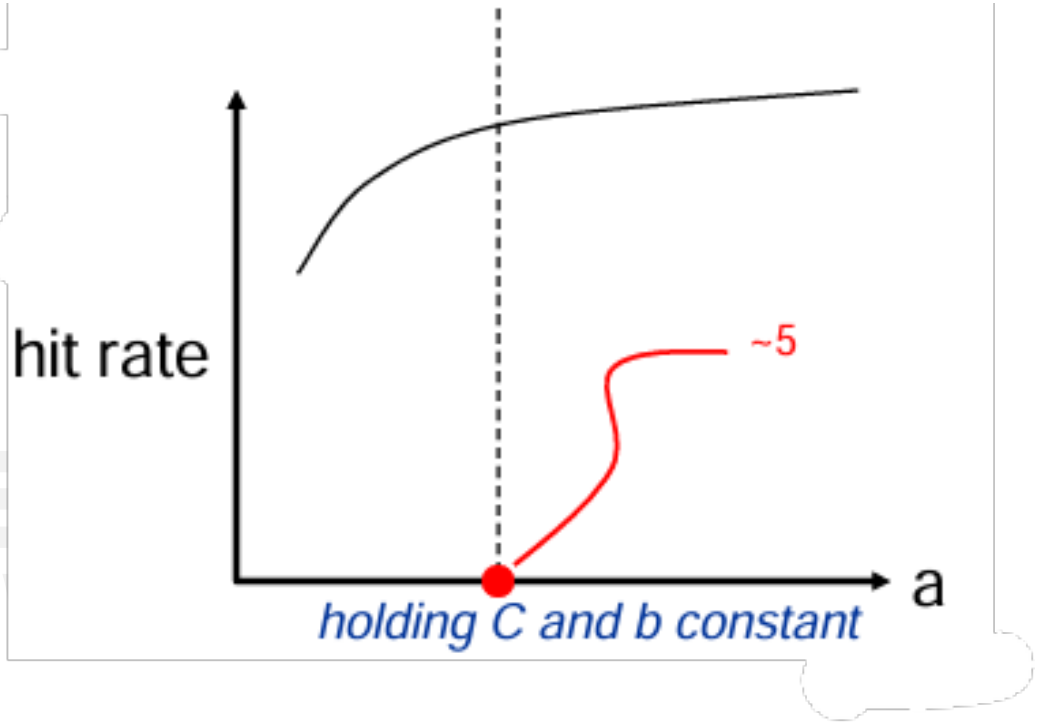
- **Block size is the data that is**
  - associated with an address tag
  - not necessarily the unit of transfer between hierarchies (remember sub-blocking)
- **Too small blocks**
  - don’t exploit spatial locality well
  - have inordinate tag overhead
- **Too large blocks**
  - useless data transferred
  - useful data permanently replaced
  - —too few total # blocks



Cache参数的选择

- Associativity（a）

  - **Partition cache frames into**
    - equivalence classes of frames called sets
  - **Typical values for associativity**
    - 1, 2-, 4-, 8-way associative
  - **Larger associativity**
    - lower miss rate less variation among programs
    - only important for small “C/b”
  - **Smaller associativity**
    - lower cost, faster hit time

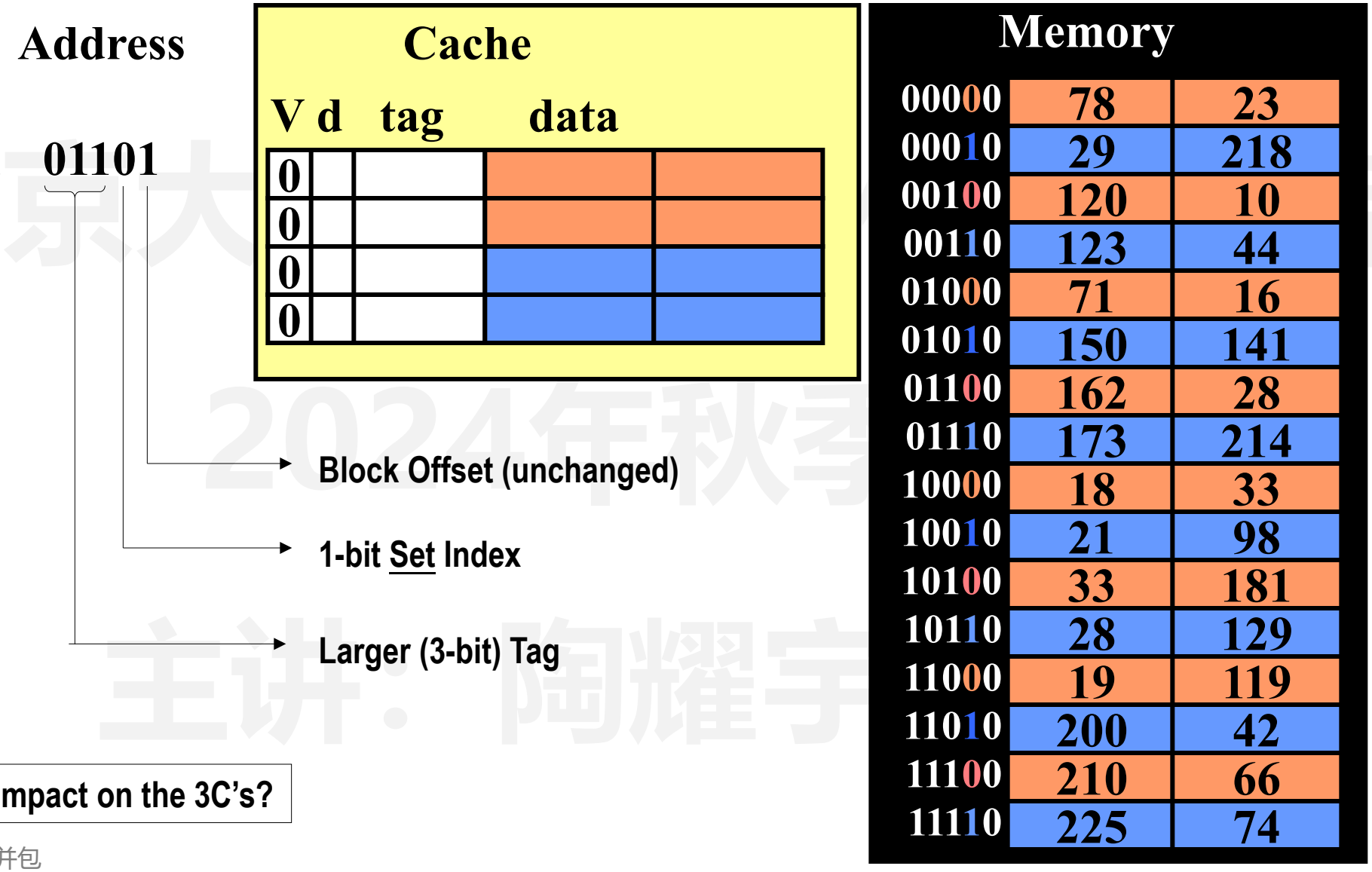


Cache设计选择的影响

- Cache写入和Miss处理策略

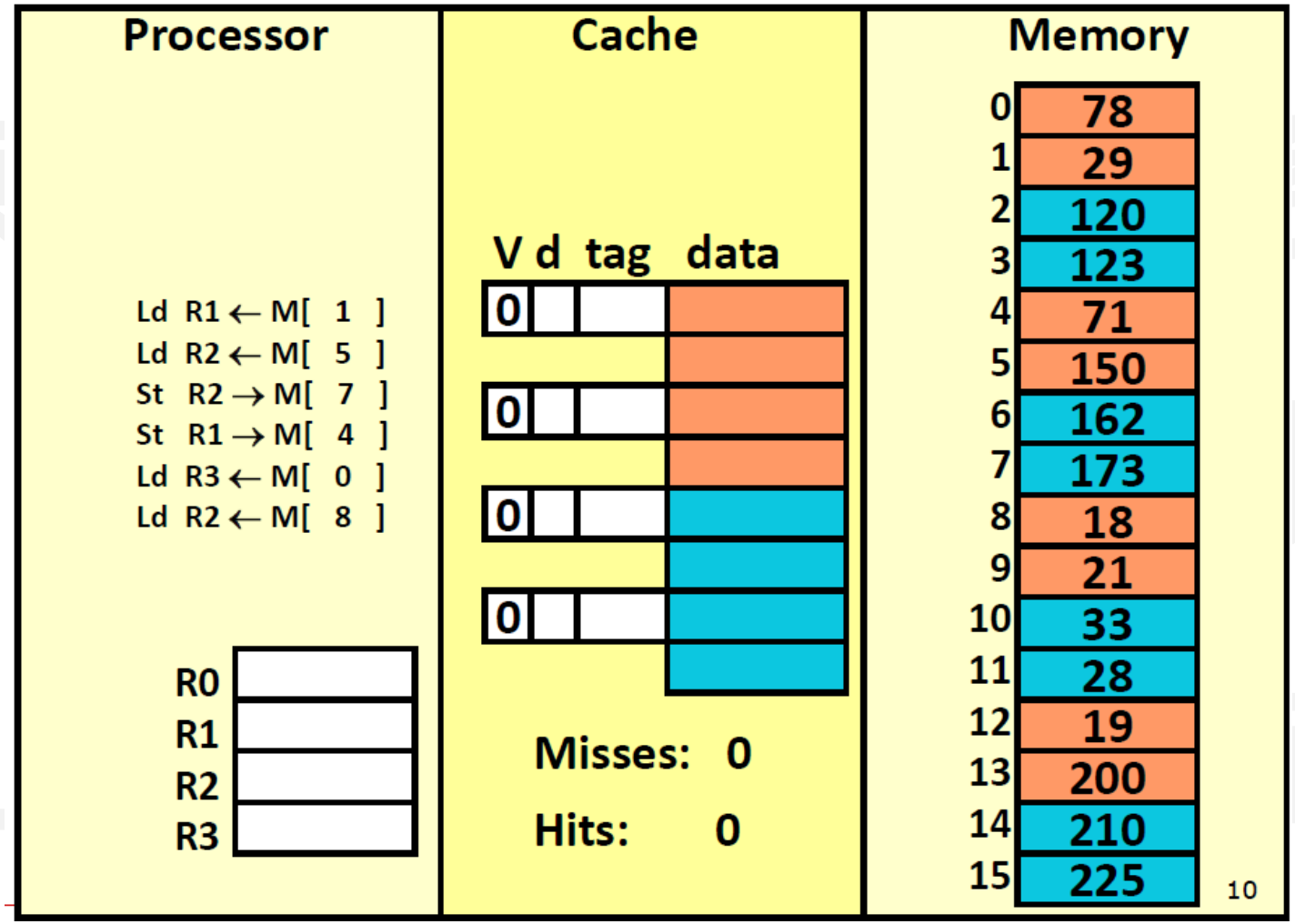
- Write Policy: How to deal with write misses?
  - ***Write-through / no-allocate***
    - update memory on each write
    - keeps memory up-to-date
  - ***Write-back / write-allocate***
    - update memory only on block replacement
    - Many cache lines are only read and never written to
    - add “dirty” bit to status word
      - originally cleared after replacement
      - set when a block frame is written to
      - only write back a dirty block, and “drop” clean blocks w/o memory update

2-Way Set Associative Cache设计



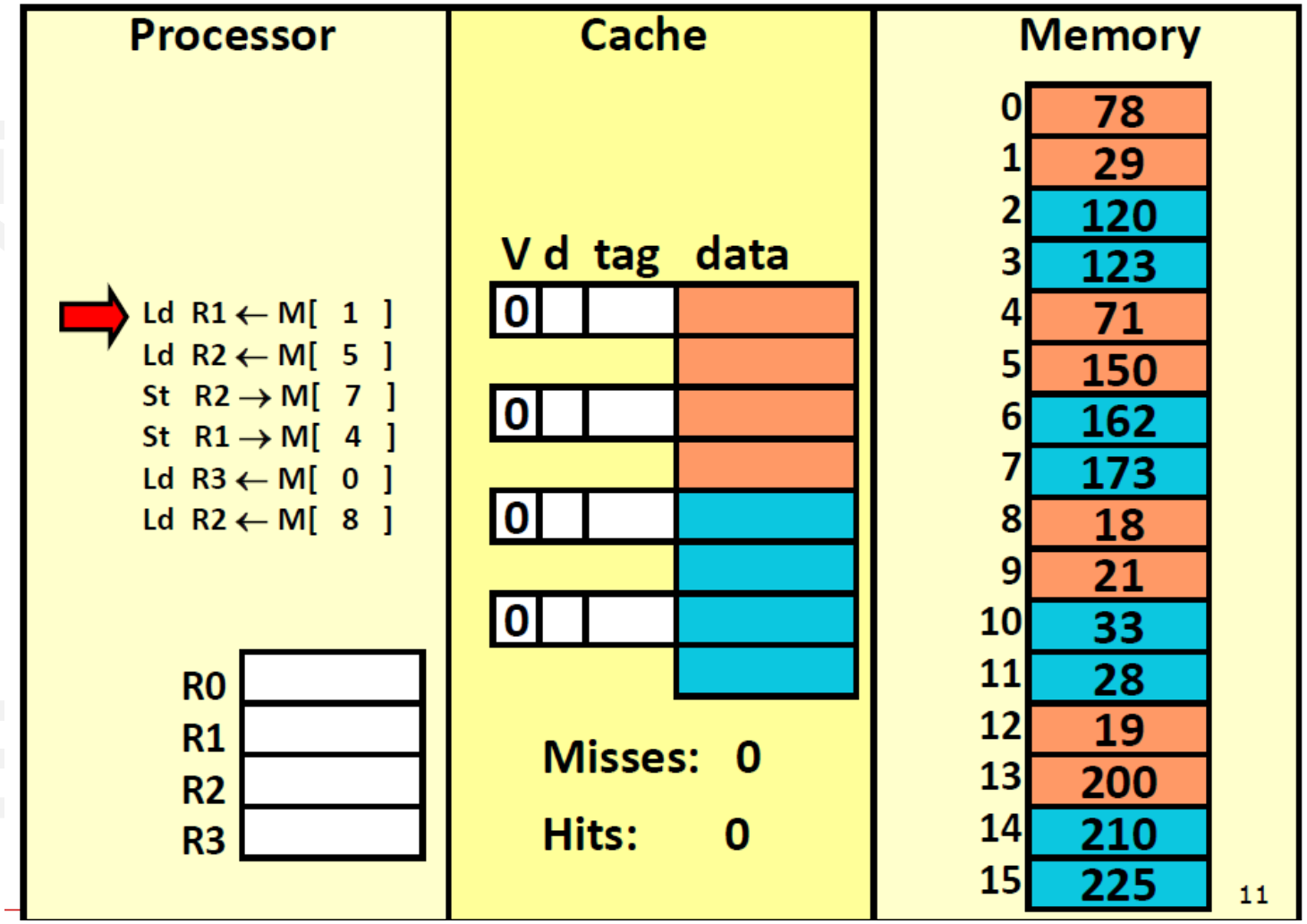
2-Way Set Associative Cache实例

- Write-back & Write-allocate



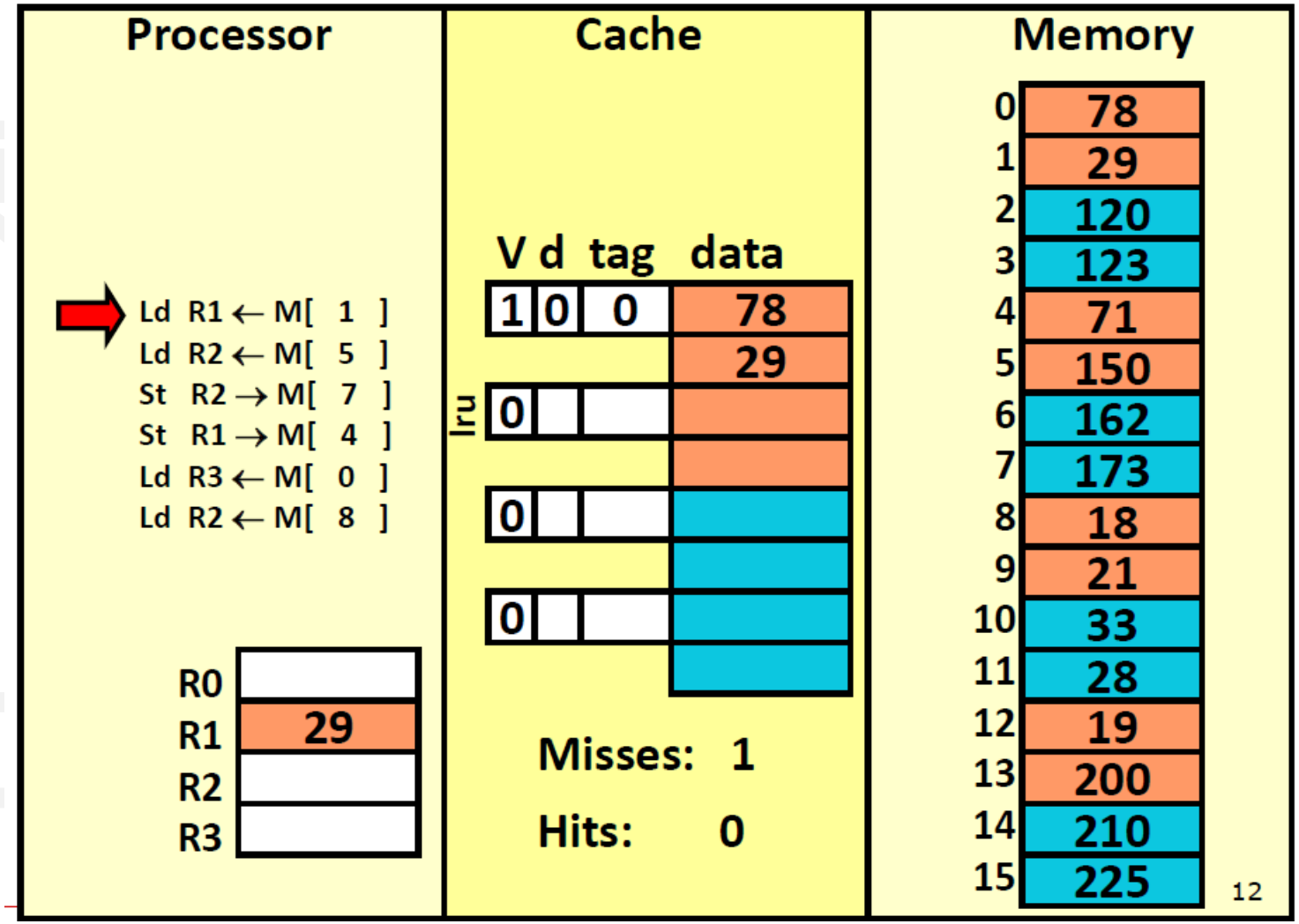
2-Way Set Associative Cache实例

- Write-back & Write-allocate



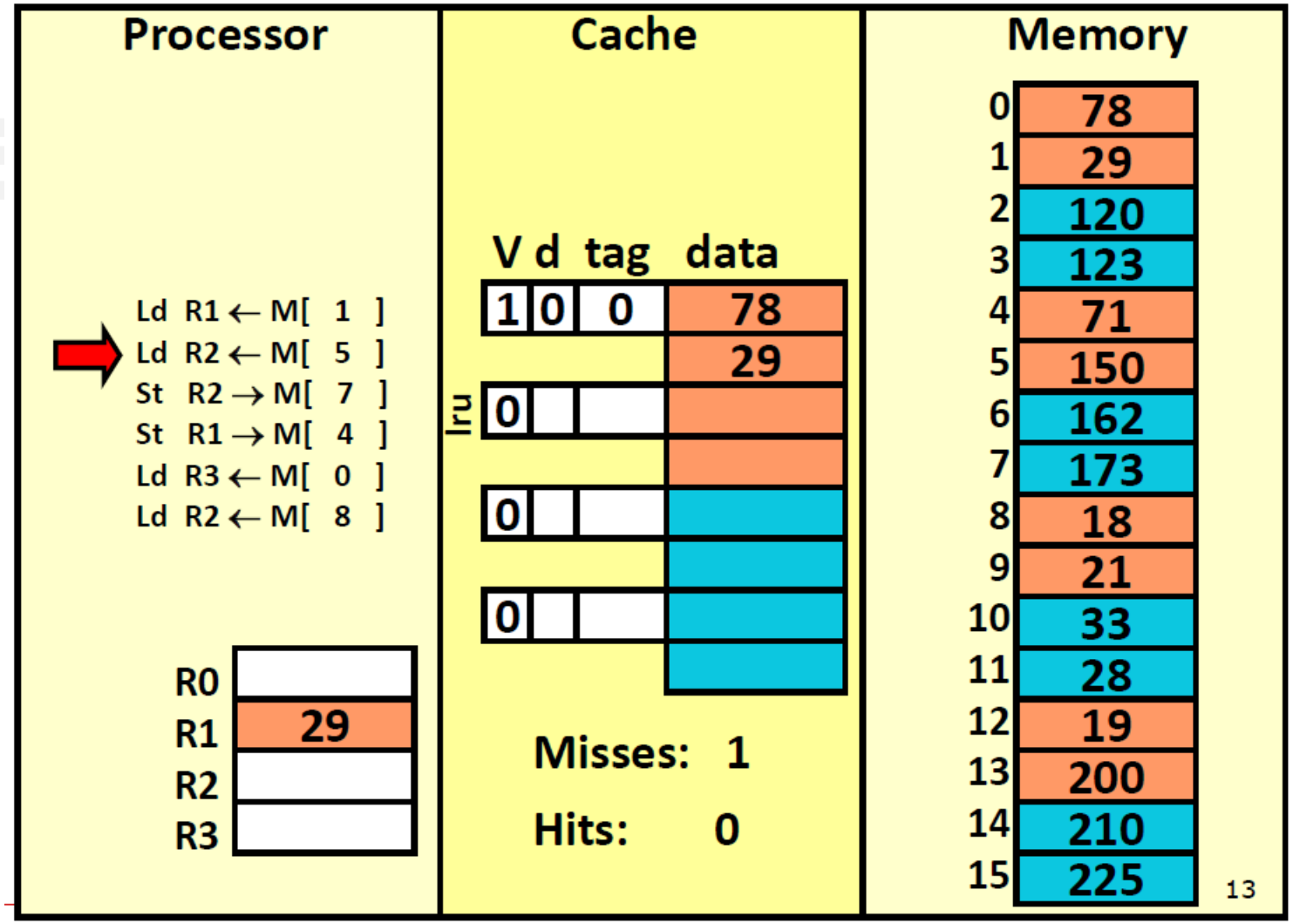
2-Way Set Associative Cache实例

- Write-back & Write-allocate



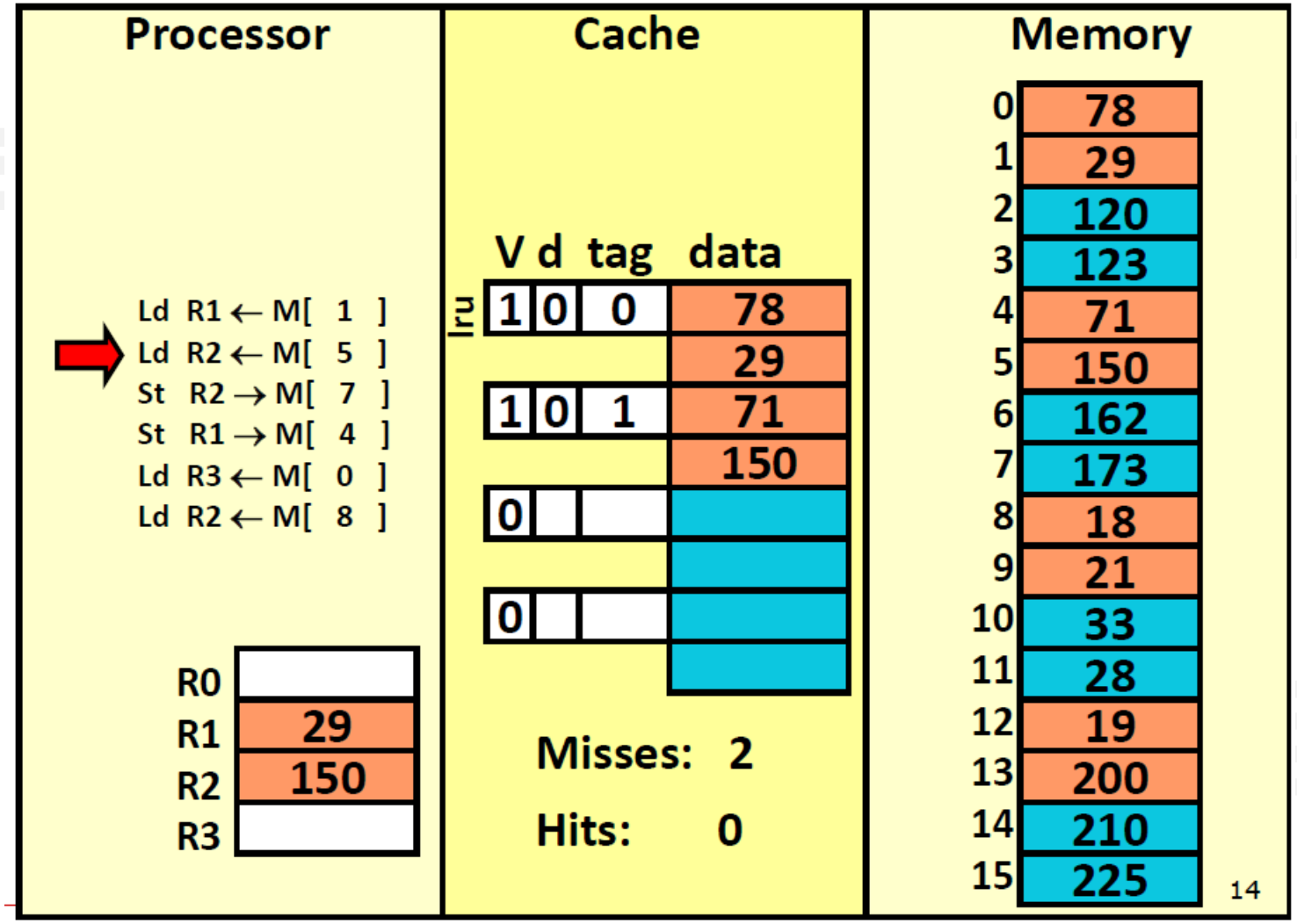
2-Way Set Associative Cache实例

- Write-back & Write-allocate



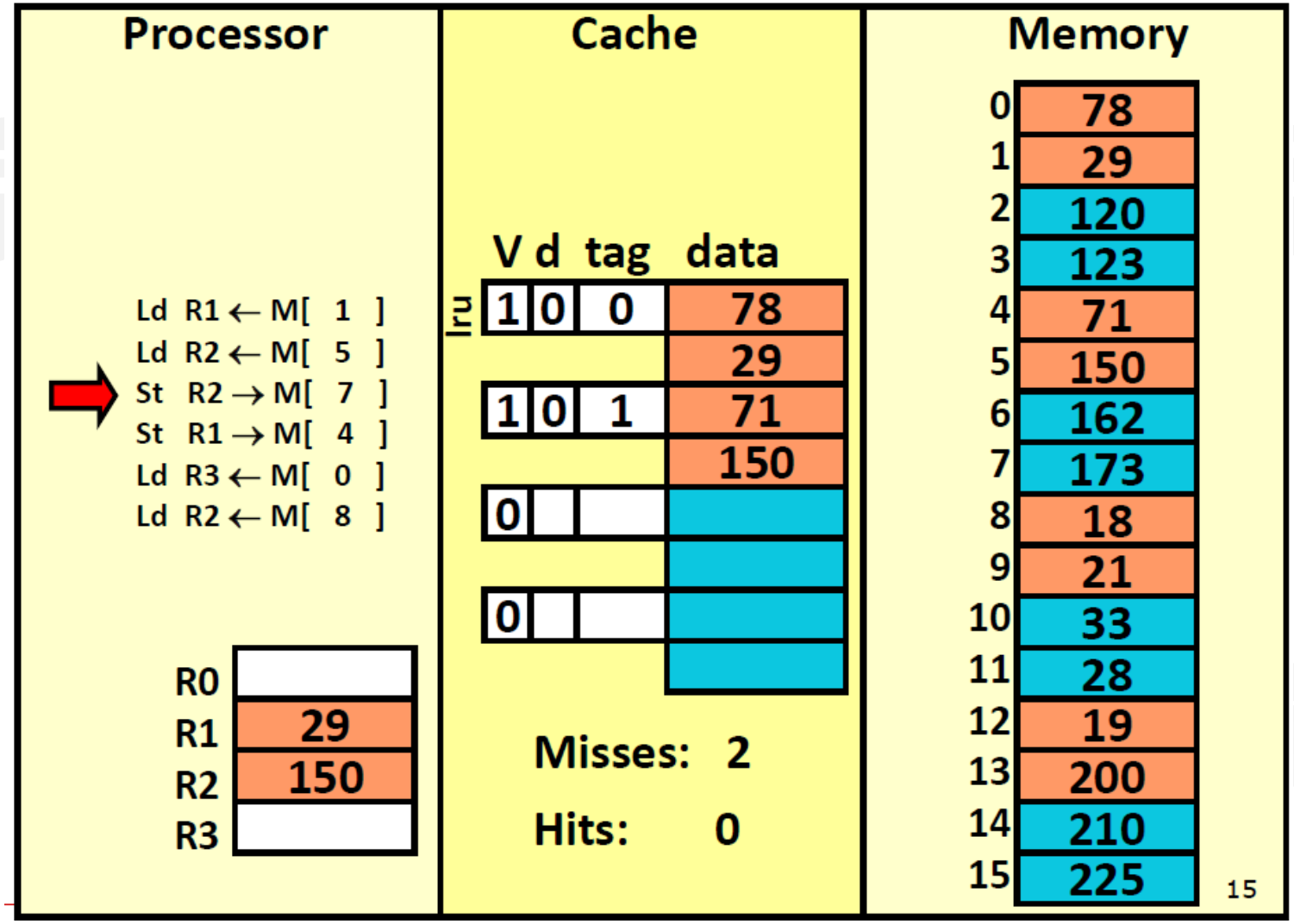
2-Way Set Associative Cache实例

- Write-back & Write-allocate



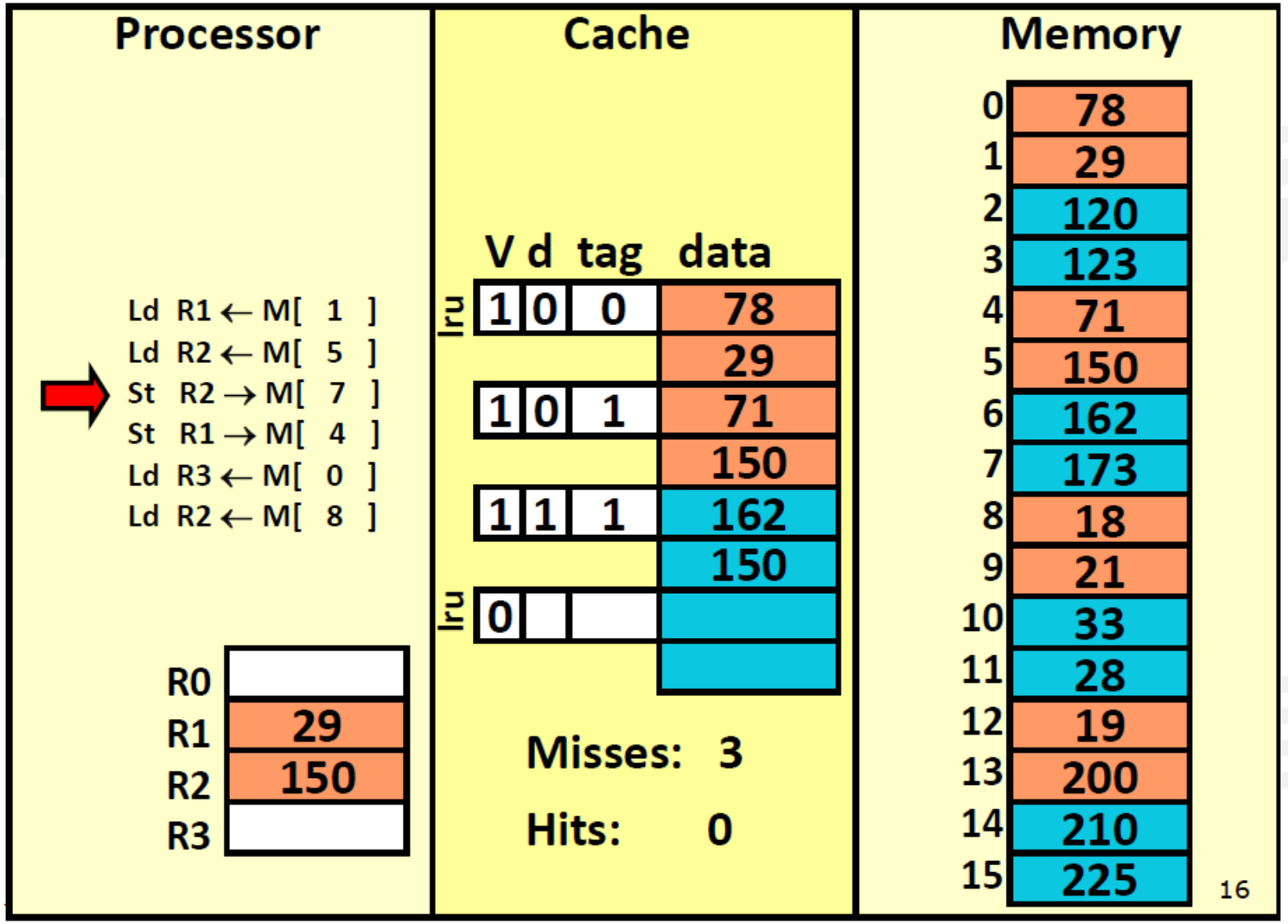
2-Way Set Associative Cache实例

- Write-back & Write-allocate



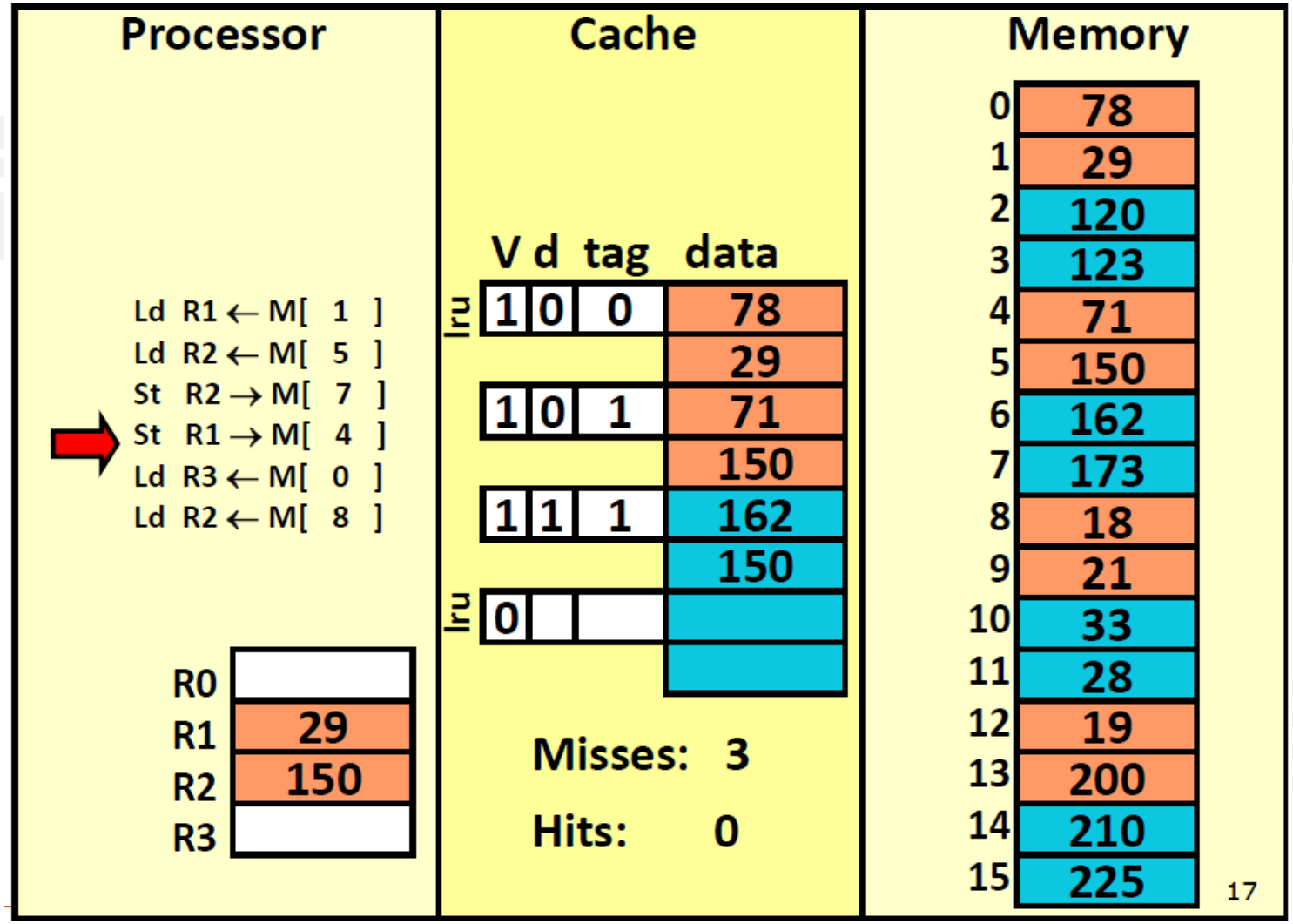
2-Way Set Associative Cache实例

- Write-back & Write-allocate



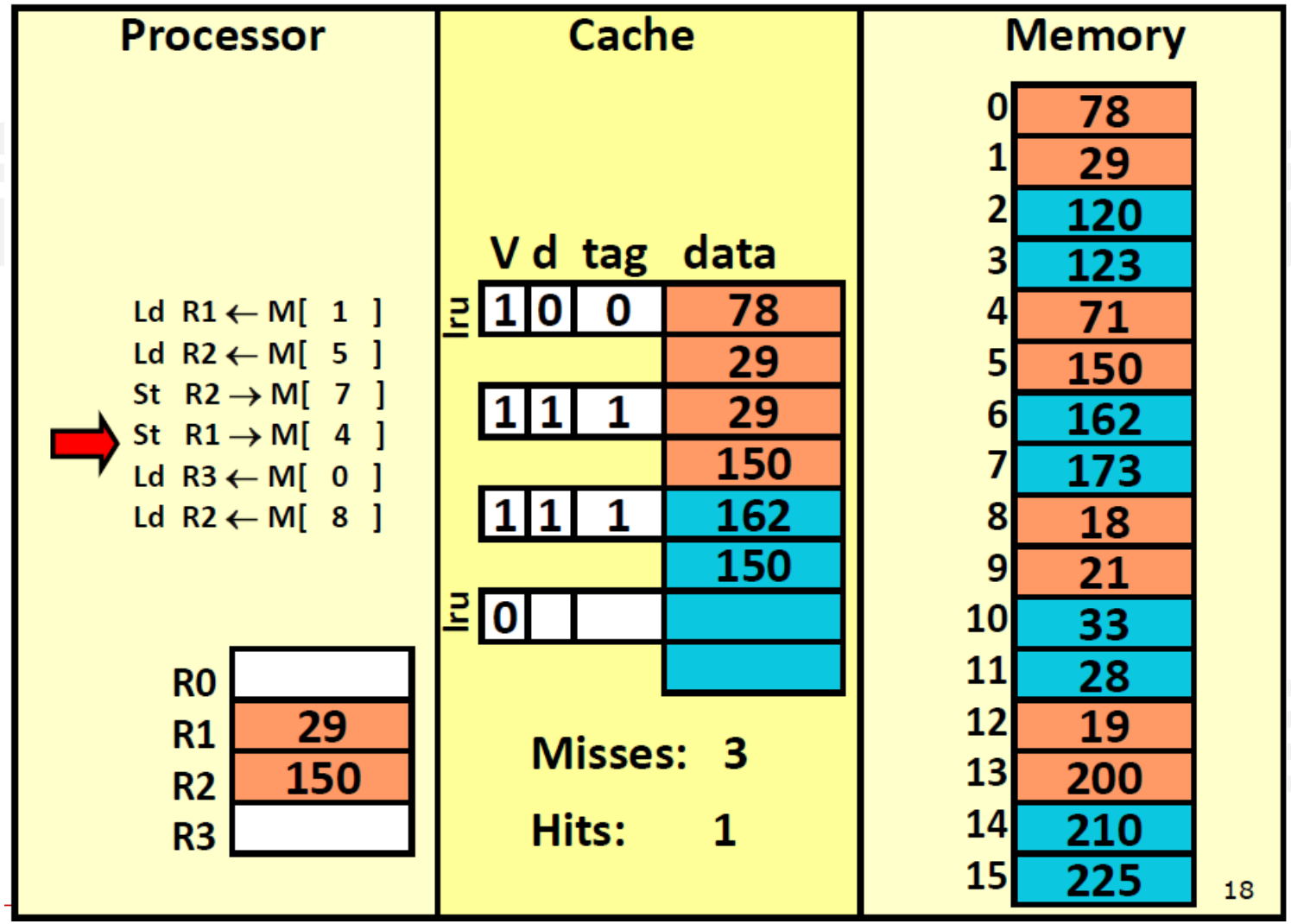
2-Way Set Associative Cache实例

- Write-back & Write-allocate



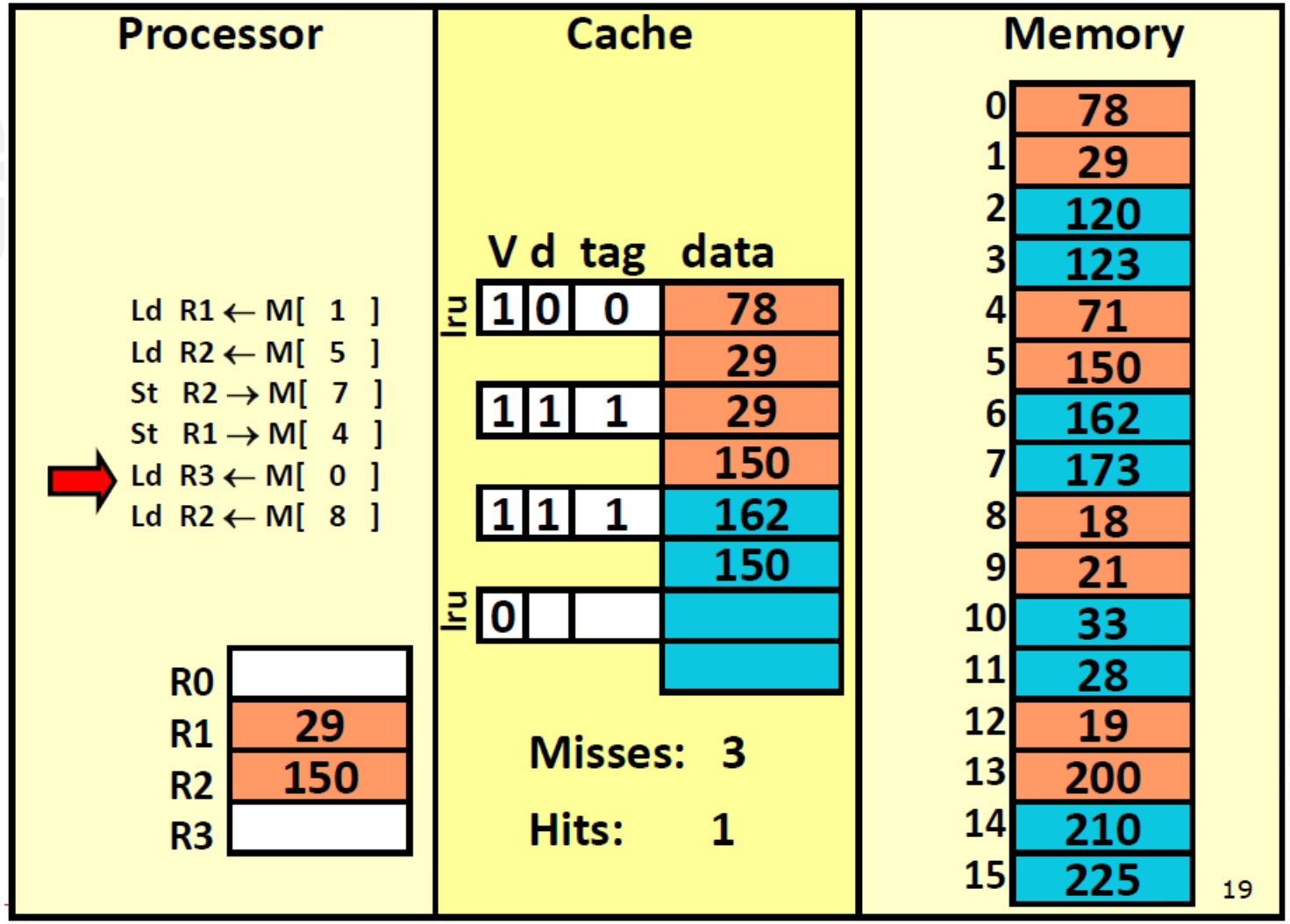
2-Way Set Associative Cache实例

- Write-back & Write-allocate



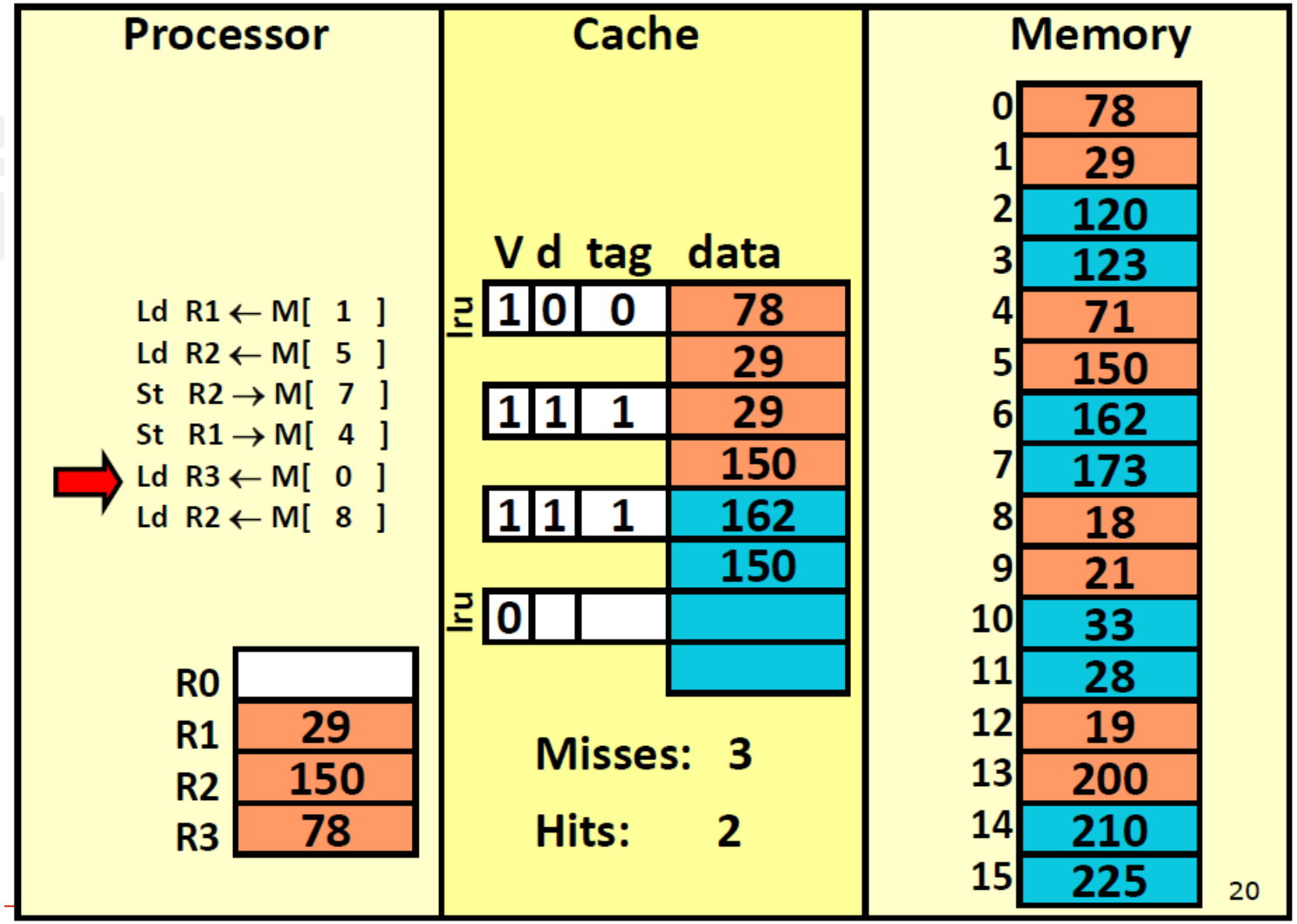
2-Way Set Associative Cache实例

- Write-back & Write-allocate



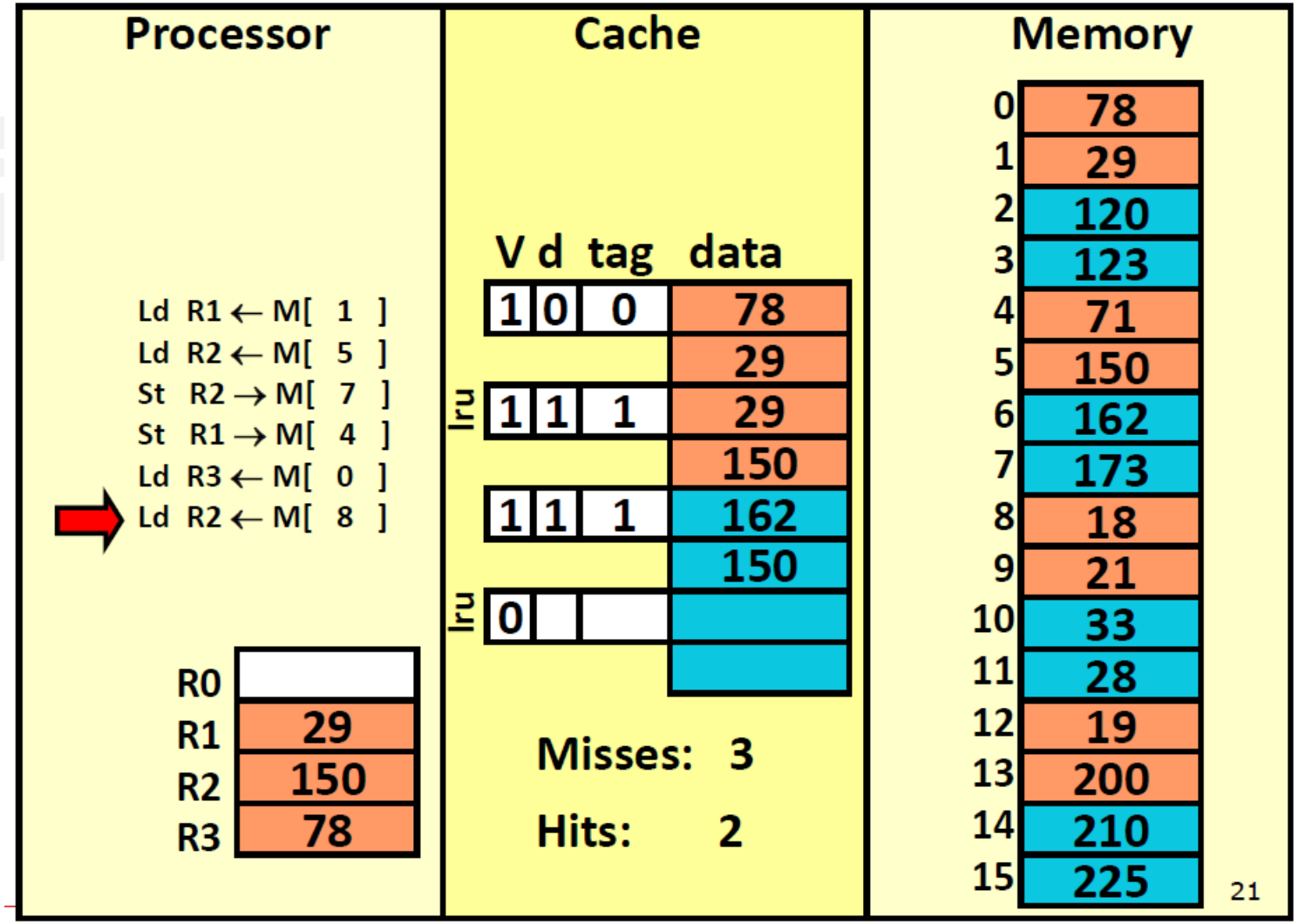
2-Way Set Associative Cache实例

- Write-back & Write-allocate



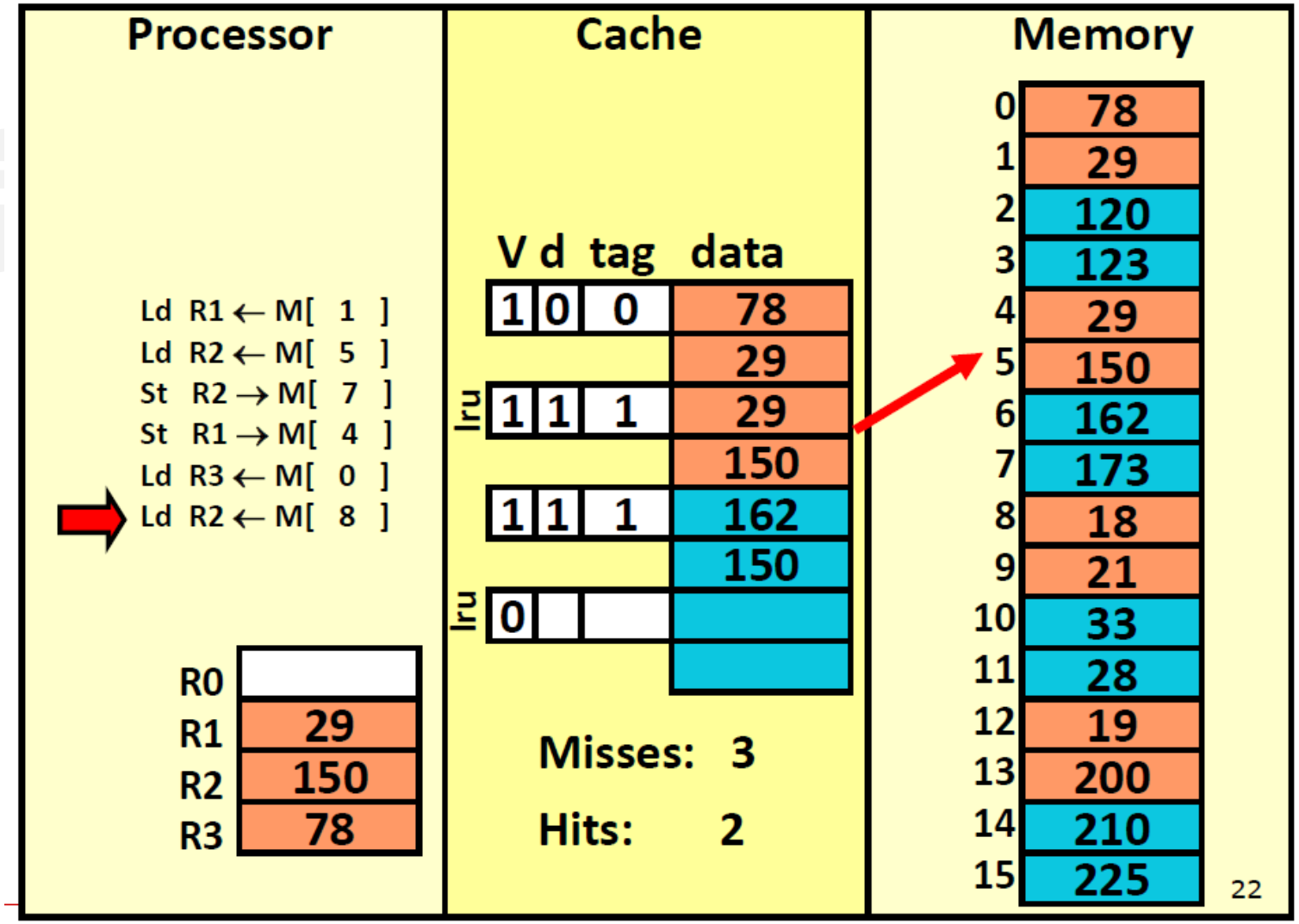
2-Way Set Associative Cache实例

- Write-back & Write-allocate



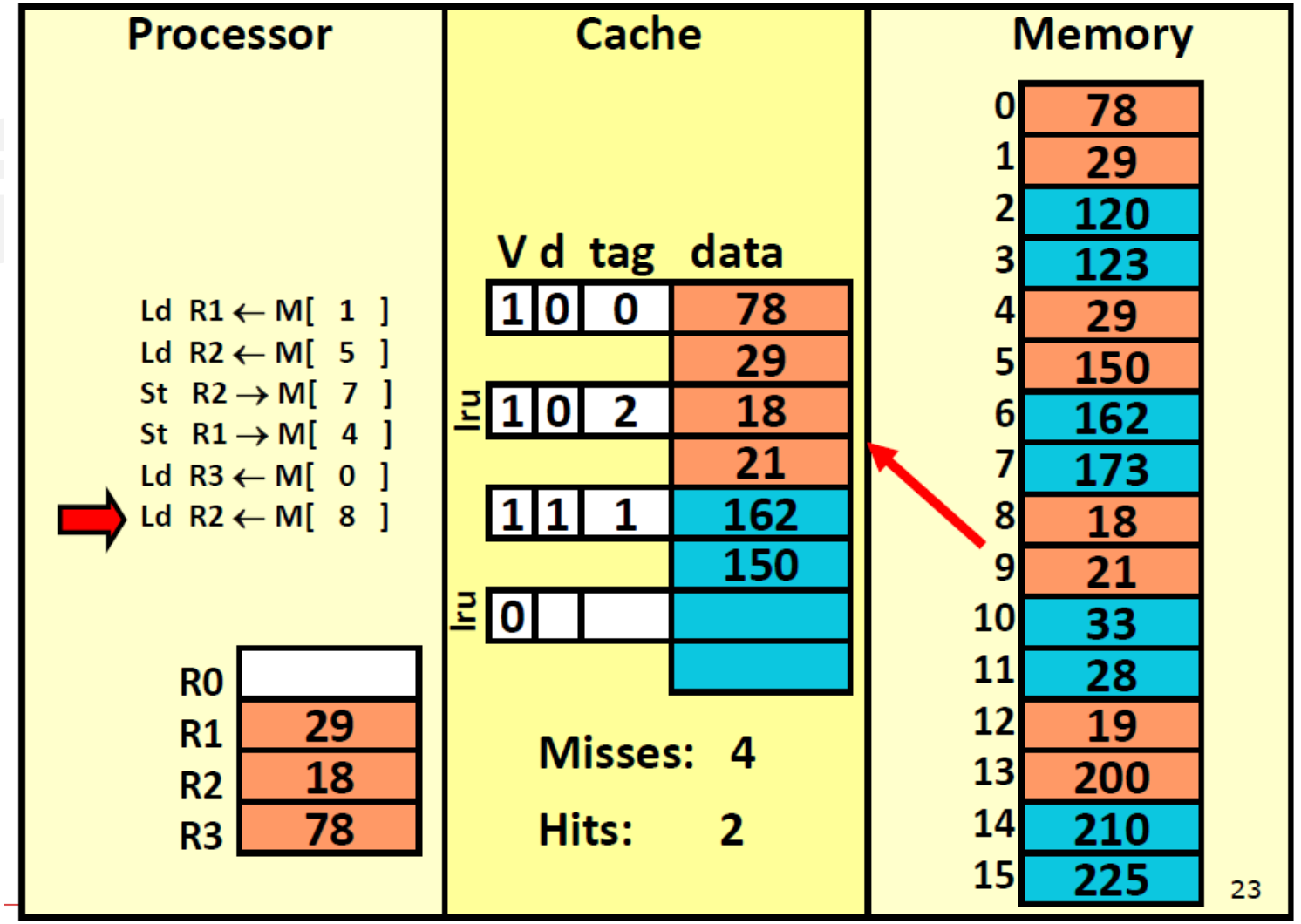
2-Way Set Associative Cache实例

- Write-back & Write-allocate



2-Way Set Associative Cache实例

- Write-back & Write-allocate



案例分析

- Cache地址的比特分区映射

For a 32-bit address and 16KB cache with 64-byte blocks, show the breakdown of the address for the following cache configuration:

A) fully associative cache

Block Offset = 6 bits
Tag = 32 - 6 = 26 bits

B) 4-way set associative cache

Block Offset = 6 bits
#sets = #lines / ways = 64
Set Index = 6 bits
Tag = 32 - 6 - 6 = 20 bits

C) Direct-mapped cache

Block Offset = 6 bits
#lines = 256 Line Index = 8 bits
Tag = 32 – 6 – 8 = 18 bits

案例分析

- Cache Access时间分析

  - **T_avg = T_hit + miss_ratio x T_miss**
    - comparable DM and SA caches with same T_miss
    - but, associativity that minimizes T_avg often smaller than associativity t hat minimizes miss_ratio

$$diff(t_{cache}) = t_{cache}(SA) - t_{cache}(DM) \geq 0$$

$$diff(miss) = miss(SA) - miss\ (DM) \leq 0$$

e.g.,

assuming $diff(t_{cache}) = 0$ => SA better

assuming $diff(miss) = -1\%$, $t_{miss} = 20$

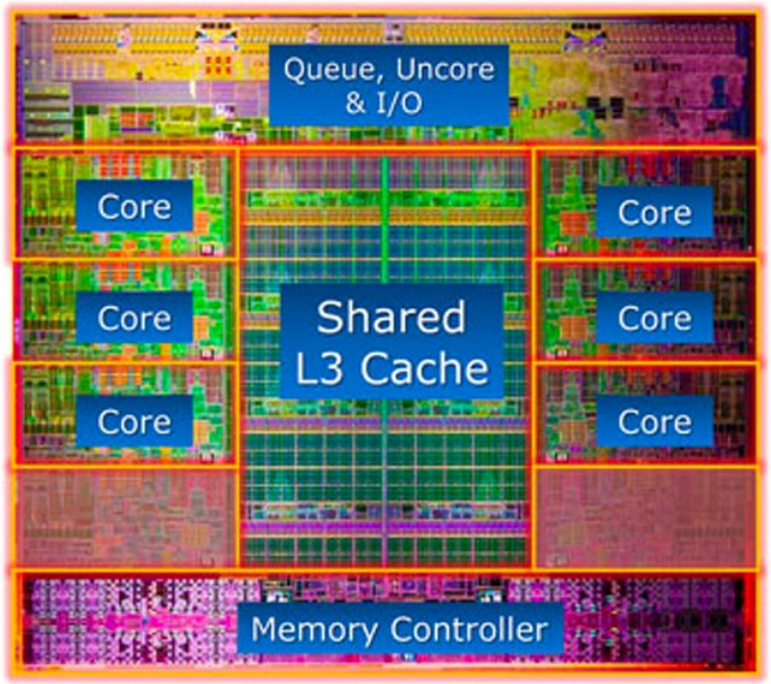
$\Rightarrow$ if $diff(t_{cache}) > 0.2$ cycle then SA loses

目 录

CONTENTS

缓存一致性的来源

- 多核共享cache



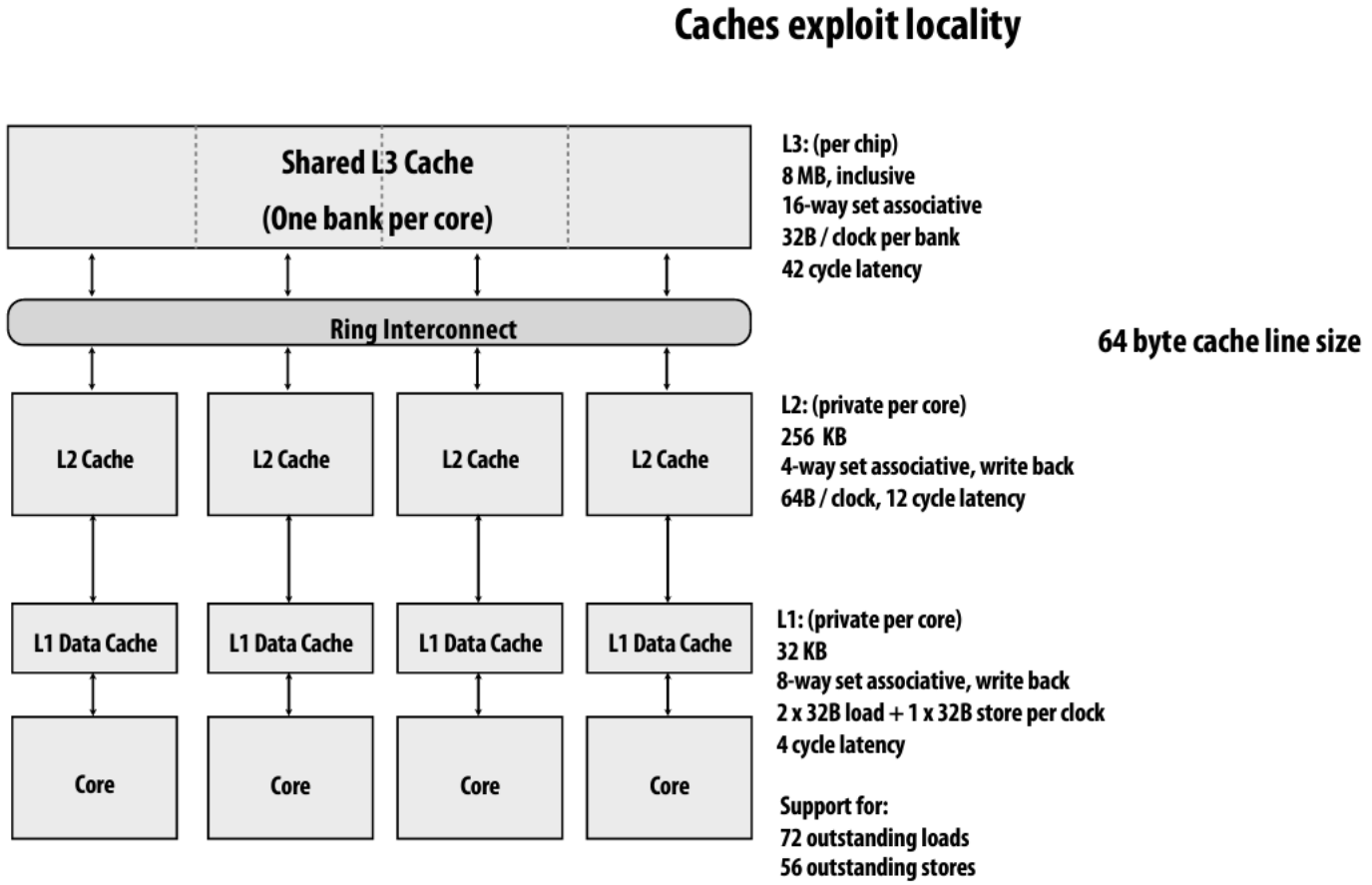
- 30% of the die area is cache

缓存一致性的来源

- 多核共享cache

3 Cs cache miss model

- Cold
- Capacity
- Conflict



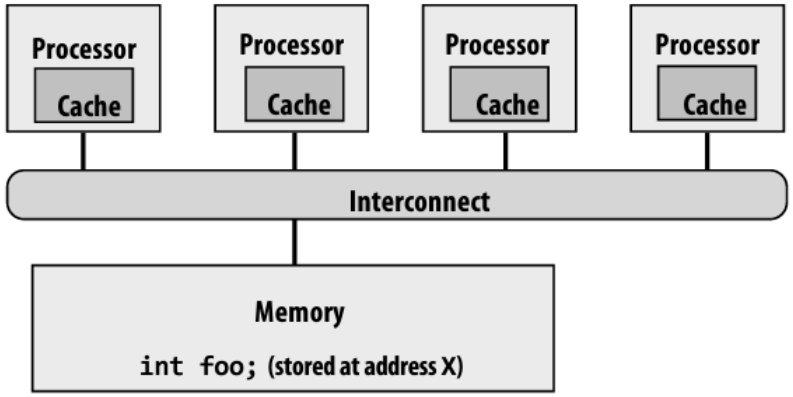
Source: Intel 64 and IA-32 Architectures Optimization Reference Manual (June 2016)

缓存一致性的来源

- 多核共享cache

Modern processors replicate contents of memory in local caches

Problem: processors can observe different values for the same memory location



The chart at right shows the value of variable `foo` (stored at address X) in main memory and in each processor's cache

Assume the initial value stored at address X is 0

Assume write-back cache behavior

Action	P1 $	P2 $	P3 $	P4 $	mem[X]
					0
P1 load X	0 miss				0
P2 load X	0	0 miss			0
P1 store X	1	0			0
P3 load X	1	0	0 miss		0
P3 store X	1	0	2		0
P2 load X	1	0 hit	2		0
P1 load Y (assume this load causes eviction of X)		0	2		1

缓存一致性的来源

北京大学 PEKING UNIVERSITY

- 缓存一致性协议

- **The logic we are about to describe is performed by each processor’s cache controller in response to:**
  - - Loads and stores by the local processor
  - - Messages from other caches on the bus
- **If all cache controllers operate according to this described protocol, then coherence will be maintained**
  - - The caches “cooperate” to ensure coherence is maintained

缓存一致性的来源

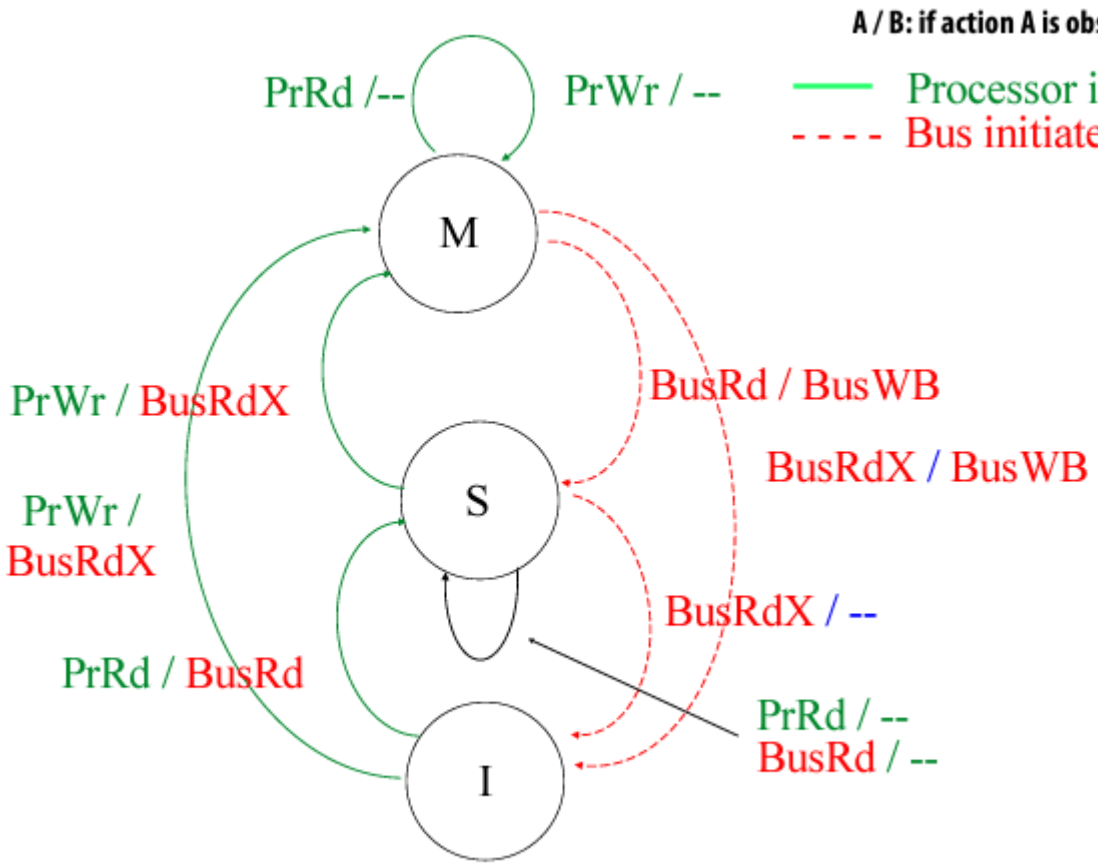
- **Invalidation-based write-back protocol**
  - **Key ideas:**
    - A line in the “modified” state can be modified without notifying the other caches
  - **Processor can only write to lines in the modified state**
    - Need a way to tell other caches that processor wants exclusive access to the line
    - We accomplish this by sending all the other caches messages
  - **When cache controller sees a request for modified access to a line it contains**
    - It must invalidate the line in its cache

缓存一致性

- MSI write-back invalidation protocol
  - **Key tasks of protocol**
    - Ensuring processor obtains exclusive access for a write
    - Locating most recent copy of cache line’s data on cache miss
  - **Three cache line states**
    - Invalid (I): same as meaning of invalid in uniprocessor cache
    - Shared (S): line valid in one or more caches, memory is up to date
    - Modified (M): line valid in exactly one cache (a.k.a. “dirty” or “exclusive” state)
  - **Two processor operations (triggered by local CPU)**
    - PrRd(read)
    - PrWr(write)
  - **Three coherence-related bus transactions (from remote caches)**
    - BusRd: obtain copy of line with no intent to modify
    - BusRdX: obtain copy of line with intent to modify
    - BusWB: write dirty line out to memory

缓存一致性

- Cache Coherence Protocol: MSI State Diagram



Abbreviation	Action
PrRd	Processor Read
PrWr	Processor Write
BusRd	Bus Read
BusRdX	Bus Read Exclusive
BusWB	Bus Writeback

缓存一致性

- Cache Coherence Protocol: MSI State Diagram

Proc Action	P1 State	P2 state	P3 state	Bus Act	Data from
1. P1 read x	S	--	--	BusRd	Memory
2. P3 read x	S	--	S	BusRd	Memory
3. P3 write x	I	--	M	BusRdX	Memory
4. P1 read x	S	--	S	BusRd	P3's cache
5. P2 read x	S	S	S	BusRd	Memory
6. P2 write x	I	M	I	BusRdX	Memory

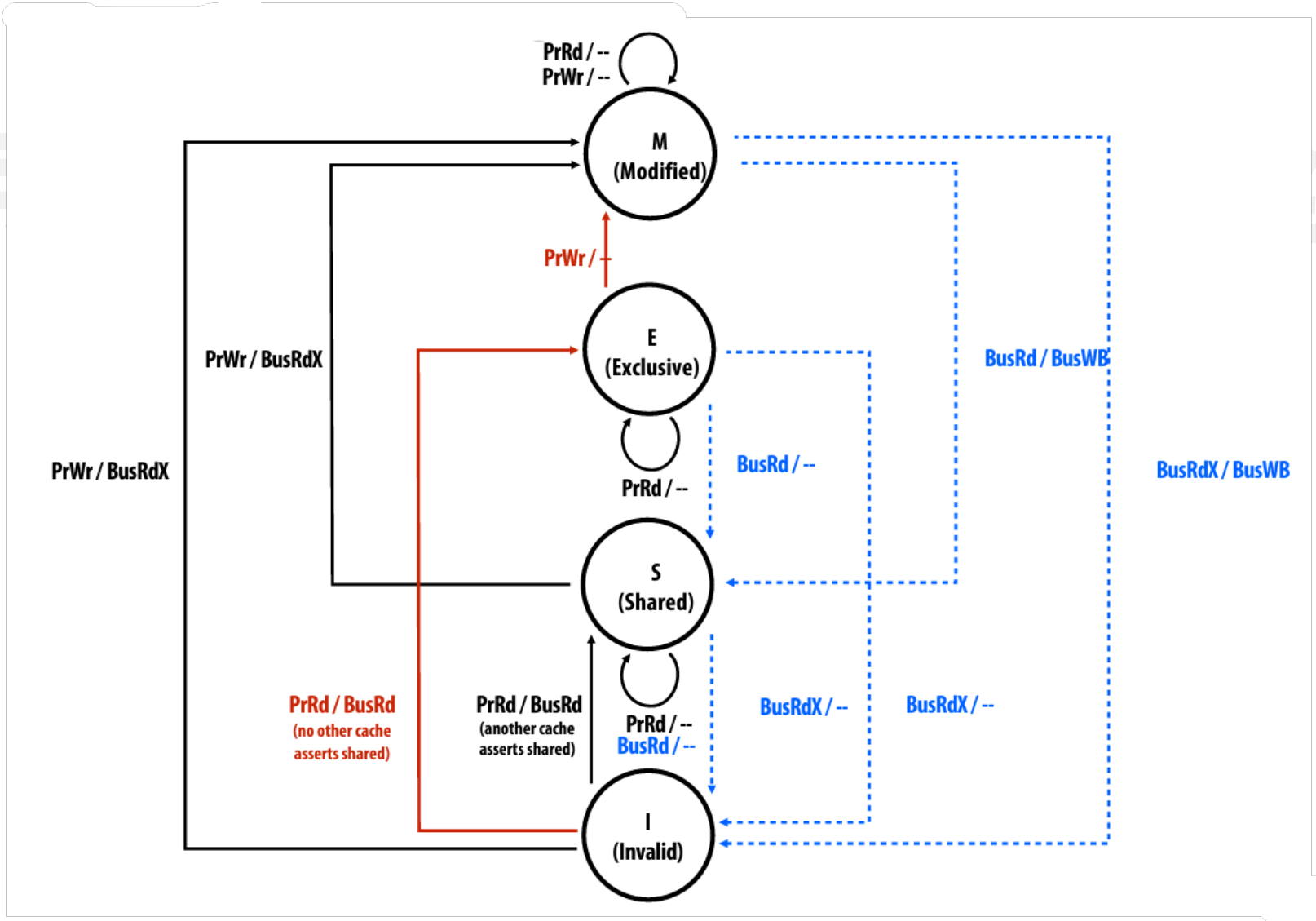
缓存一致性

- Cache Coherence Protocol: MESI invalidation protocol

- **MSI requires two interconnect transactions for the common case of reading an address, then writing to it**
  - Transaction 1: BusRdto move from I to S state
  - Transaction 2: BusRdXto move from S to M state
- **This inefficiency exists even if application has no sharing at all**
- **Solution: add additional state E (“exclusive clean”)**
  - Line has not been modified, but only this cache has a copy of the line
  - Decouples exclusivity from line ownership (line not dirty, so copy in memory is valid copy of data)
  - Upgrade from E to M does not require an bustransaction

缓存一致性

- Cache Coherence Protocol: MESI invalidation protocol



缓存一致性

- 下节课继续 – 缓存一致性与缓存优化

北京大学-智能硬件体系结构

2024年秋季学期

主讲：陶耀宇、李萌