北京大学
PEKING UNIVERSITY

智能硬件体系结构

第七讲：基于RISC指令集的MIPS架构与编[illegible]

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2次作业将在**10月31日**截止
  - 第3次作业将在**11月1日**放出，**11月15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
  - 第1次编程实验已放出
    - 具体详细信息请参考：https://aiarchpku.github.io/2024Fall//project/
    - **10月22号~11月25号**
  - **下周中安排一次Lab答疑**，请同学们先熟悉和理解项目代码结构
    - 自行搭建简单Module模块级测试验证环境，把verilog代码跑起来
  - 请同学们尽快完成Paper Review VS 考试的问卷

注意事项

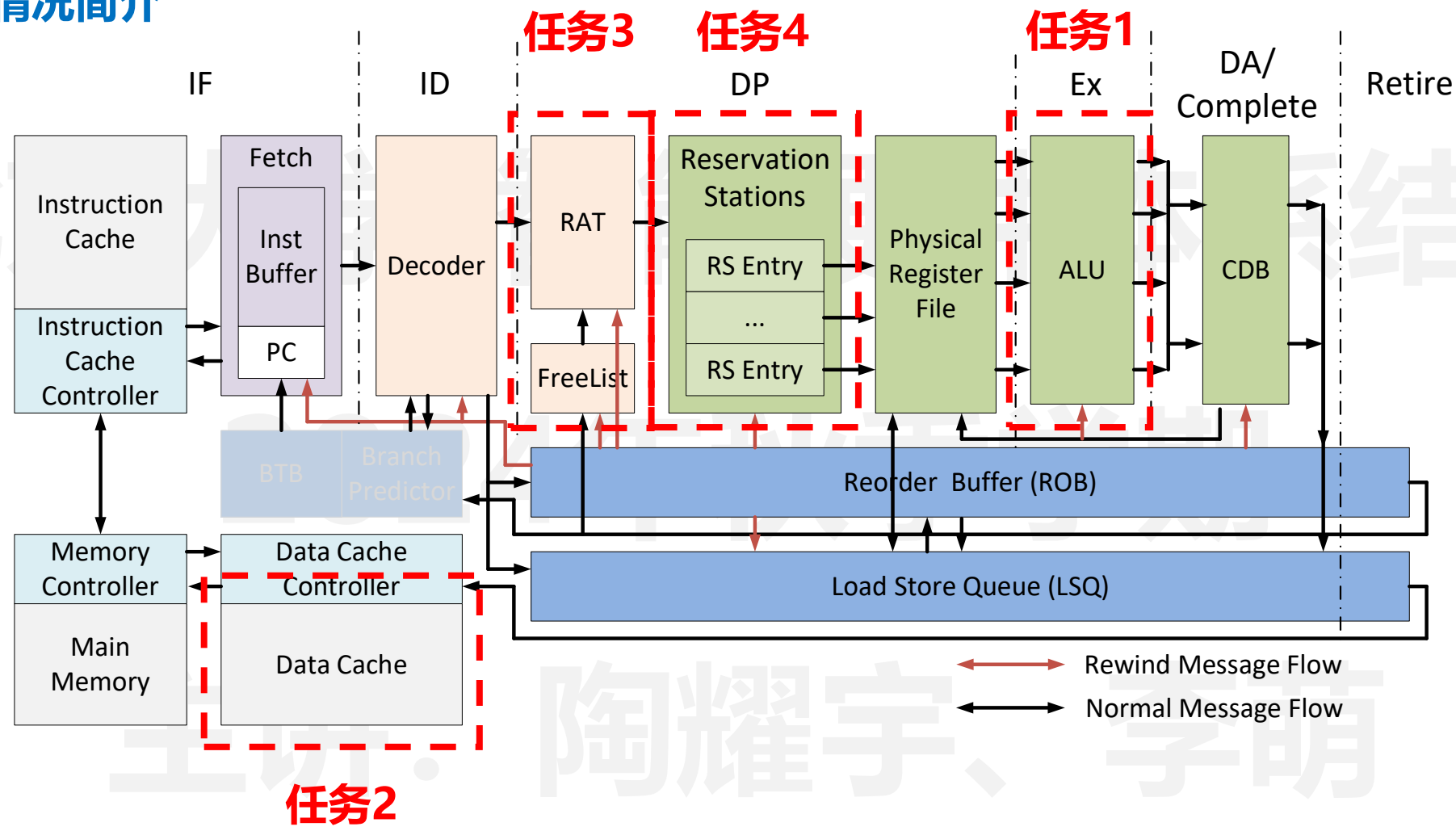
北京大学 PEKING UNIVERSITY

• Lab 1情况简介

- 项目采用CLAB平台：https://clab.pku.edu.cn/
- CLAB平台的具体使用方式请参考： CLAB使用手册
- 实验采用Linux环境进行开发
  - 所需软件环境已为各位同学安装好，无需自己配置环境
  - Linux运行Lab的说明请参考：Linux使用参考信息
- 如有任何问题，请联系授课老师或助教！

注意事项

- Lab 1情况简介



基于开源MIPS RISC-Style指令集的Superscalar Out-of-Order Processor

注意事项

北京大学 PEKING UNIVERSITY

- Lab 1实验任务简介

本项目提供项目代码以及测试系统，同学们需要完成的模块包括以下4个：

任务1：ALU

任务2：DCACHEMEM

任务3：RAT/RATTABLE

任务4：SUPER_RS

请同学们将这4个模块中缺失的代码部分填充完整，缺失代码的起始部分标注如下：

```
// ============ Start ====================
// ==========Descriptions of Functions=========
  /*
  待填充的代码模块
  */
// =============End====================
```

注意事项

北京大学 PEKING UNIVERSITY

- **Lab 1实验任务简介**

1. 在verilog/alu.v源文件中，填补简单ALU模块的功能设计，完成基本算术、逻辑、位移和分支计算功能，模块输入输出解释请参考课程网站Project部分的模块功能介绍，ALUOp需要完成的功能在sys_defs.vh中，可自行构建testbench_alu.v模块进行测试验证，**alu.v中需要填补3个代码块，每一个代码块大约10~20行左右；**
2. 在verilog/dcachemem.v源文件中，填补2-Way Cache的功能设计，完成双路读出、写入功能，可自行构建testbench_dcachemem.v模块进行测试验证，**dcachemem.v中需要填补2个代码块，每一个代码块大约20行左右；**
3. 在**verilog/ratable.v**源文件中，填补完成简单RATTABLE模块的功能设计，支持Architectural Register与Physical Register的映射与回收功能，**需要填补的1个代码块，大约10行左右；**在**verilog/rat.v**源文件中，填补freelist相关逻辑实现对空闲Physical Register的状态检测与更新；可参考testbench/testbench_rat.v构建测试模块进行验证，**需要填补的1个代码块，大约30行左右；**
4. 在**verilog/superrs.v**完成SUPER_RS模块的功能实际，将模块RS（rs.v中）连入SUPER_RS模块以支持超标量=2的指令发射功能，可参考testbench_RS.v构建测试模块进行验证，**需要填补的1个代码块，大约30行左右；**

注意事项

北京大学 PEKING UNIVERSITY

- **Lab 1项目文件结构**

---Makefile	包含所有make command
---Make.*	包含子模块的make command，可新建子模块验证路径单独验证各个子模块 示例：make all　　可自动运行program.mem内的程序 make clean　　删除所有编译产生文件
---debug_out	仿真产生的.out文件，提供简单的可视化观察模块和流水线的状态
---vs-asm	MIPS RISC指令的Assembler编译器，可以将test_progs路径下的指令汇编代码翻译成HEX格式，存入 program.mem文件供testbench文件夹下的tb读取 示例：vs-asm test_progs/fib.s > program.mem
---pipeline_gold	最简单的顺序5级流水线设计，产生用作验证的参考write_back.out和memory.out，与本项目乱序执行超标量结果进行对比
---program.mem	汇编指令代码的HEX格式文件，有tb读取作为icache的输入
---run_tests.sh	运行该脚本可一次性验证test_progs内的所有汇编指令代码运行是否正确，并比较乱序超标量与顺序5级流水线的CPI
---sys_defs.vh	定义架构设计的顶层配置参数，无需改动
---test_progs	包含所有用于测试验证的MIPS RISC-Style汇编代码
---testbench	包含项目所需的所有验证tb文件和可视化支持文件
---verilog	包含项目所需的所有架构设计Verilog源文件，具体解释详见模块说明

目 录

CONTENTS

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1				
mulf f0,f1,f2				
stf f2,Z(r1)				
addi r1,4,r1				
ldf X(r1),f1				
mulf f0,f1,f2				
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	
f2	
r1	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	no						
4	FP1	no						
5	FP2	no						

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

Tomasulo:

Cycle 1

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1			
mulf f0,f1,f2				
stf f2,Z(r1)				
addi r1,4,r1				
ldf X(r1),f1				
mulf f0,f1,f2				
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	
r1	

CDB	
T	V

使用RS序号作为映射值

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	no						
4	FP1	no						
5	FP2	no						

allocate

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

Tomasulo:

Cycle 2

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2		
mulf f0,f1,f2	c2			
stf f2,Z(r1)				
addi r1,4,r1				
ldf X(r1),f1				
mulf f0,f1,f2				
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	no						
4	FP1	yes	mulf	f2	-	RS#2	[f0]	-
5	FP2	no						

allocate

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

Tomasulo:

Cycle 3

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	
mulf f0,f1,f2	c2			
stf f2,Z(r1)	c3			
addi r1,4,r1				
ldf X(r1),f1				
mulf f0,f1,f2				
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	RS#2	[f0]	-
5	FP2	no						

allocate

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

Tomasulo:

Cycle 4

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4		
stf f2,Z(r1)	c3			
addi r1,4,r1	c4			
ldf X(r1),f1				
mulf f0,f1,f2				
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	RS#1

CDB	
T	V
RS#2	[f1]

`ldf` finished (W)
clear `f1` RegStatus
CDB broadcast

Reservation Stations									
T	FU	busy	op	R	T1	T2	V1	V2	
1	ALU	yes	addi	r1	-	-	[r1]	-	allocate
2	LD	no							free
3	ST	yes	stf	-	RS#4	-	-	[r1]	
4	FP1	yes	mulf	f2	-	RS#2	[f0]	CDB.V	RS#2 ready → grab CDB value
5	FP2	no							

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

Tomasulo:

Cycle 5

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5	
stf f2,Z(r1)	c3			
addi r1,4,r1	c4	c5		
ldf X(r1),f1	c5			
mulf f0,f1,f2				
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#4
r1	RS#1

CDB	
T	V

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	yes	addi	r1	-	-	[r1]	-
2	LD	yes	ldf	f1	-	RS#1	-	-
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	-	[f0]	[f1]
5	FP2	no						

allocate

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

假设 multf 需要3个cycle完成

Tomasulo:

Cycle 6

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	
stf f2,Z(r1)	c3			
addi r1,4,r1	c4	c5	c6	
ldf X(r1),f1	c5			
mulf f0,f1,f2	c6			
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	
f2	RS#4RS#5
r1	RS#1

CDB	
T	V

no D stall on WAW: scoreboard would overwrite f2 RegStatus
anyone who needs old f2 tag has it

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	yes	addi	r1	-	-	[r1]	-
2	LD	yes	ldf	f1	-	RS#1	-	-
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	-	[f0]	[f1]
5	FP2	yes	mulf	f2	-	RS#2	[f0]	-

allocate

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

假设 multf 需要3个cycle完成

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	
stf f2,Z(r1)	c3			
addi r1,4,r1	c4	c5	c6	c7
ldf X(r1),f1	c5	c7		
mulf f0,f1,f2	c6			
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#5
r1	RS#1

CDB	
T	V
RS#1	[r1]

Tomasulo:

Cycle 7

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	RS#1	-	CDB.V
3	ST	yes	stf	-	RS#4	-	-	[r1]
4	FP1	yes	mulf	f2	-	-	[f0]	[f1]
5	FP2	yes	mulf	f2	-	RS#2	[f0]	-

no W wait on WAR: scoreboard would anyone who needs old r1 has RS copy

D stall on store RS: structural

addi finished (W)
clear r1 RegStatus
CDB broadcast

RS#1 ready → grab CDB value

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

假设 multf 需要3个cycle完成

Tomasulo:

Cycle 8

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	c8
stf f2,Z(r1)	c3	c8		
addi r1,4,r1	c4	c5	c6	c7
ldf X(r1),f1	c5	c7	c8	
mulf f0,f1,f2	c6			
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#5
r1	

CDB	
T	V
RS#4	[f2]

mulf finished (W)
don't clear f2 RegStatus
already overwritten by 2nd mulf (RS#5)
CDB broadcast

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	yes	ldf	f1	-	-	-	[r1]
3	ST	yes	stf	-	RS#4	-	CDB.V	[r1]
4	FP1	no						
5	FP2	yes	mulf	f2	-	RS#2	[f0]	-

RS#4 ready → grab CDB value

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

假设 multf 需要3个cycle完成

Tomasulo:

Cycle 9

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	c8
stf f2,Z(r1)	c3	c8	c9	
addi r1,4,r1	c4	c5	c6	c7
ldf X(r1),f1	c5	c7	c8	c9
mulf f0,f1,f2	c6	c9		
stf f2,Z(r1)				

Map Table	
Reg	T
f0	
f1	RS#2
f2	RS#5
r1	

CDB	
T	V
RS#2	[f1]

2nd ldf finished (W)
clear f1 RegStatus
CDB broadcast

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	-	-	-	[f2]	[r1]
4	FP1	no						
5	FP2	yes	mulf	f2	-	RS#2	[f0]	CDB.V

RS#2 ready → grab CDB value

Tomasulo动态指令发射算法

- Tomasulo动态指令发射实例

假设 multf 需要3个cycle完成

Tomasulo:

Cycle 10

Insn Status				
Insn	D	S	X	W
ldf X(r1),f1	c1	c2	c3	c4
mulf f0,f1,f2	c2	c4	c5+	c8
stf f2,Z(r1)	c3	c8	c9	c10
addi r1,4,r1	c4	c5	c6	c7
ldf X(r1),f1	c5	c7	c8	c9
mulf f0,f1,f2	c6	c9	c10	
stf f2,Z(r1)	c10			

Map Table	
Reg	T
f0	
f1	
f2	RS#5
r1	

CDB	
T	V

stf finished (W)
no output register → no CDB broadcast

Reservation Stations								
T	FU	busy	op	R	T1	T2	V1	V2
1	ALU	no						
2	LD	no						
3	ST	yes	stf	-	RS#5	-	-	[r1]
4	FP1	no						
5	FP2	yes	mulf	f2	-	-	[f0]	[f1]

free → allocate

超标量+动态指令发射

北京大学 PEKING UNIVERSITY

- **Tomasulo动态指令发射实例**

  - Dynamic scheduling and multiple issue are orthogonal
    - **E.g., Pentium4:** dynamically scheduled 5-way superscalar
    - Two dimensions
      - **N**: superscalar width (number of parallel operations)
      - **W**: window size (number of reservation stations)

  - What do we need for an **N**-by-**W** Tomasulo?
    - RS: **N** tag/value w-ports (D), **N** value r-ports (S), **2N** tag CAMs (W)
    - Select logic: **W**→**N** priority encoder (S)
    - MT: **2N** r-ports (D), **N** w-ports (D)
    - RF: **2N** r-ports (D), **N** w-ports (W)
    - CDB: **N** (W)
    - Which are the expensive pieces?

超标量+动态指令发射

北京大学 PEKING UNIVERSITY

- Tomasulo动态指令发射实例

  - Superscalar select logic: W→N priority encoder
    - – Somewhat complicated (N^2 logW)
    - • Can simplify using different RS designs
  - **Split design**
    - • Divide RS into N banks: 1 per FU?
    - • Implement N separate W/N→1 encoders
    - + Simpler: N * logW/N
    - – Less scheduling flexibility
  - **FIFO design**
    - • Can issue only head of each RS bank
    - + Simpler: no select logic at all
    - – Less scheduling flexibility (but surprisingly not that bad)

目 录

CONTENTS

分支处理机制与地址预测

- Tomasulo动态发射的潜在问题

  - When can Tomasulo go wrong?
    - Branches
      - What if a branch finishes after younger instructions (after the branch) finish?
    - Exceptions!!
      - No way to figure out relative order of instructions in RS
      - **We need a mechanism to predict branch results**
      - **We need a mechanism to ensure finish in order**

分支预测

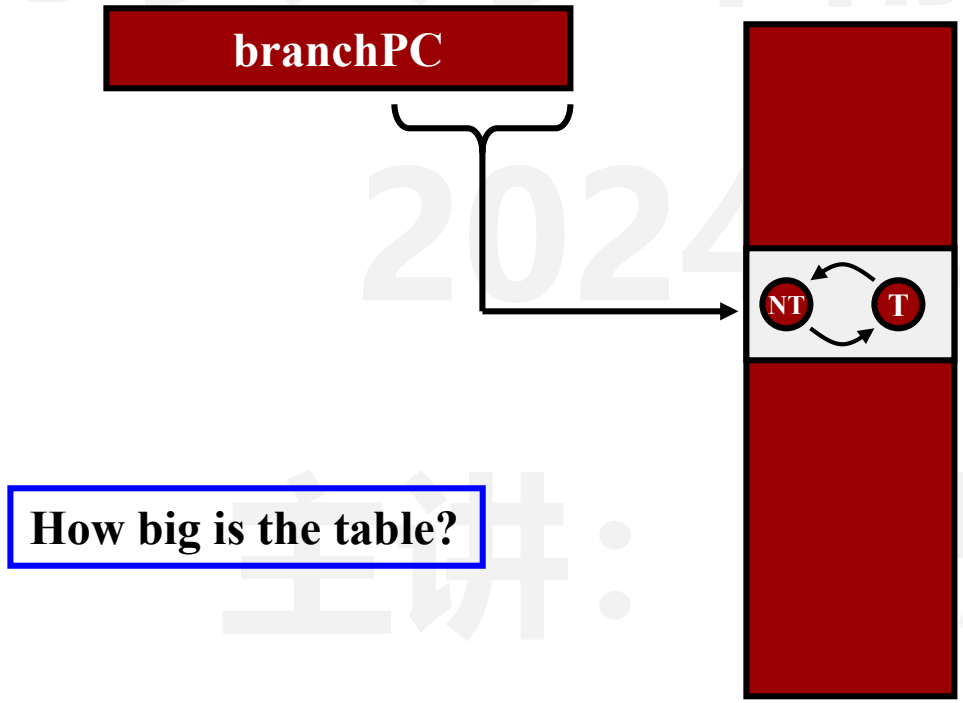
- 包括方向预测、地址预测与恢复机制

- Direction Predictor（方向预测）
  - For conditional branches
    - **Predicts whether the branch will be taken**
  - Examples:
    - Always taken; backwards taken
- Address Predictor（地址预测）
  - Predicts the target address (use if predicted taken)
  - Examples:
    - **BTB; Return Address Stack; Precomputed Branch**
- Recovery logic

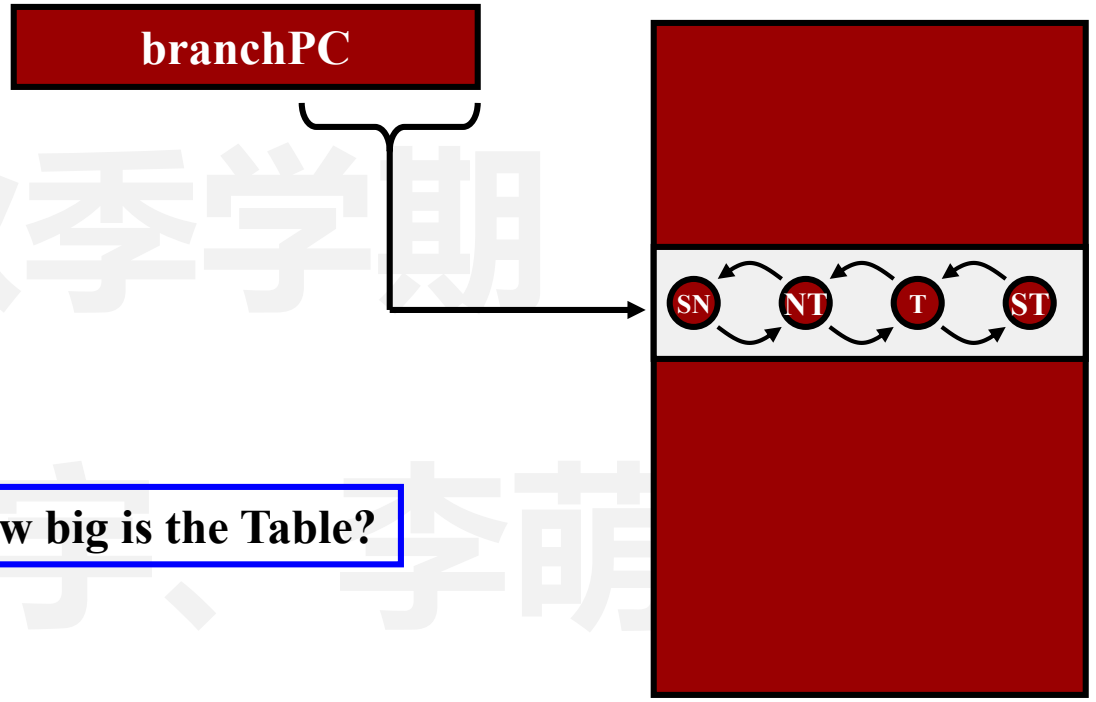
分支预测

- 方向预测 – 基于历史的简单状态机FSM

- 1-bit history (direction predictor)
  - Remember the last direction for a branch



- 2-bit history (direction predictor)



分支预测

- 方向预测 – 基于历史的简单状态机FSM

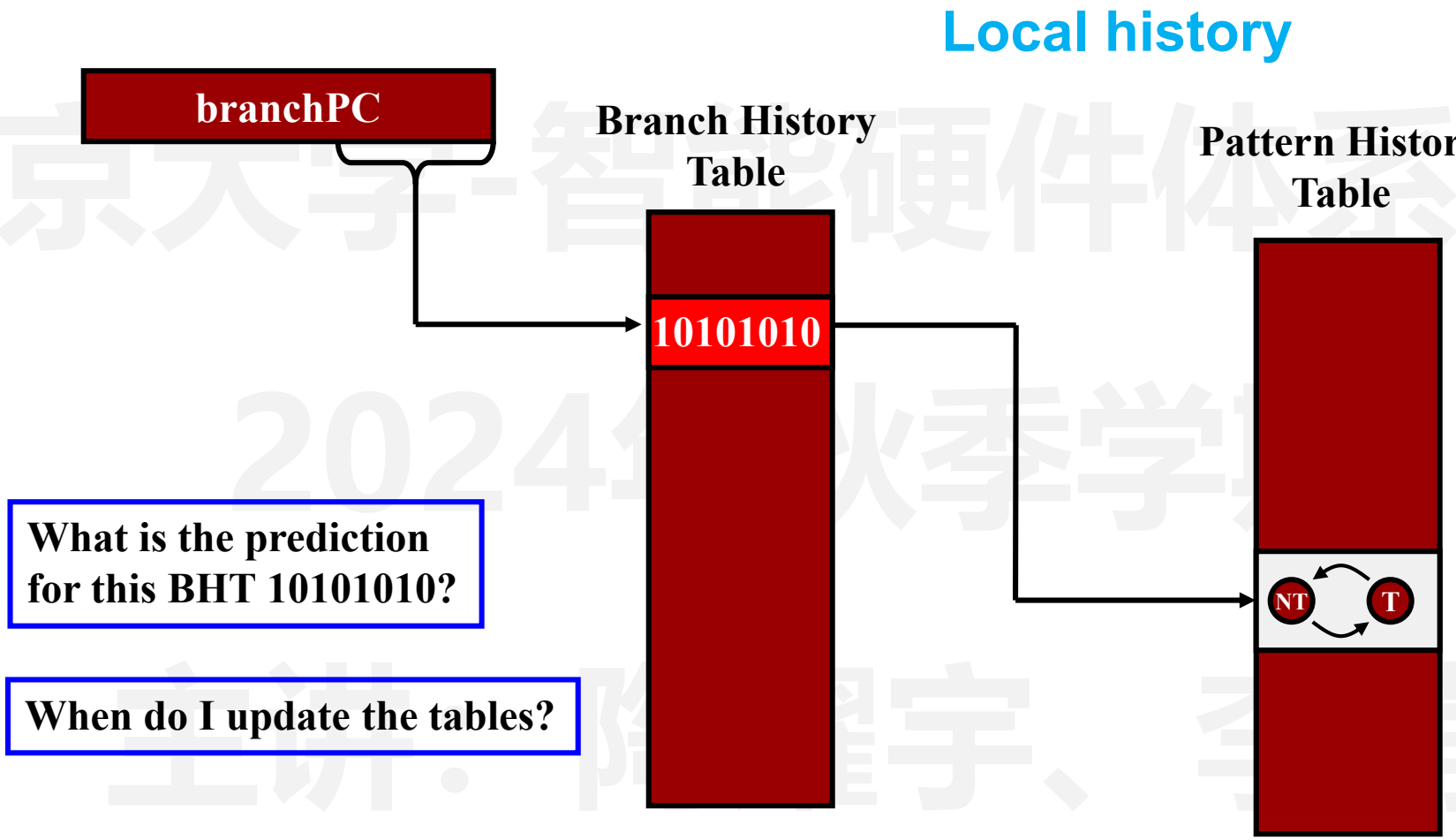
- **~80 percent of branches are either heavily TAKEN or heavily NOT-TAKEN**
- For the other 20%, we need to look at patterns of reference to see if they are predictable using a more complex predictor
- Example: gcc has a branch that flips each time

T(1) NT(0) 10101010101010101010101010101010

Using History Patterns

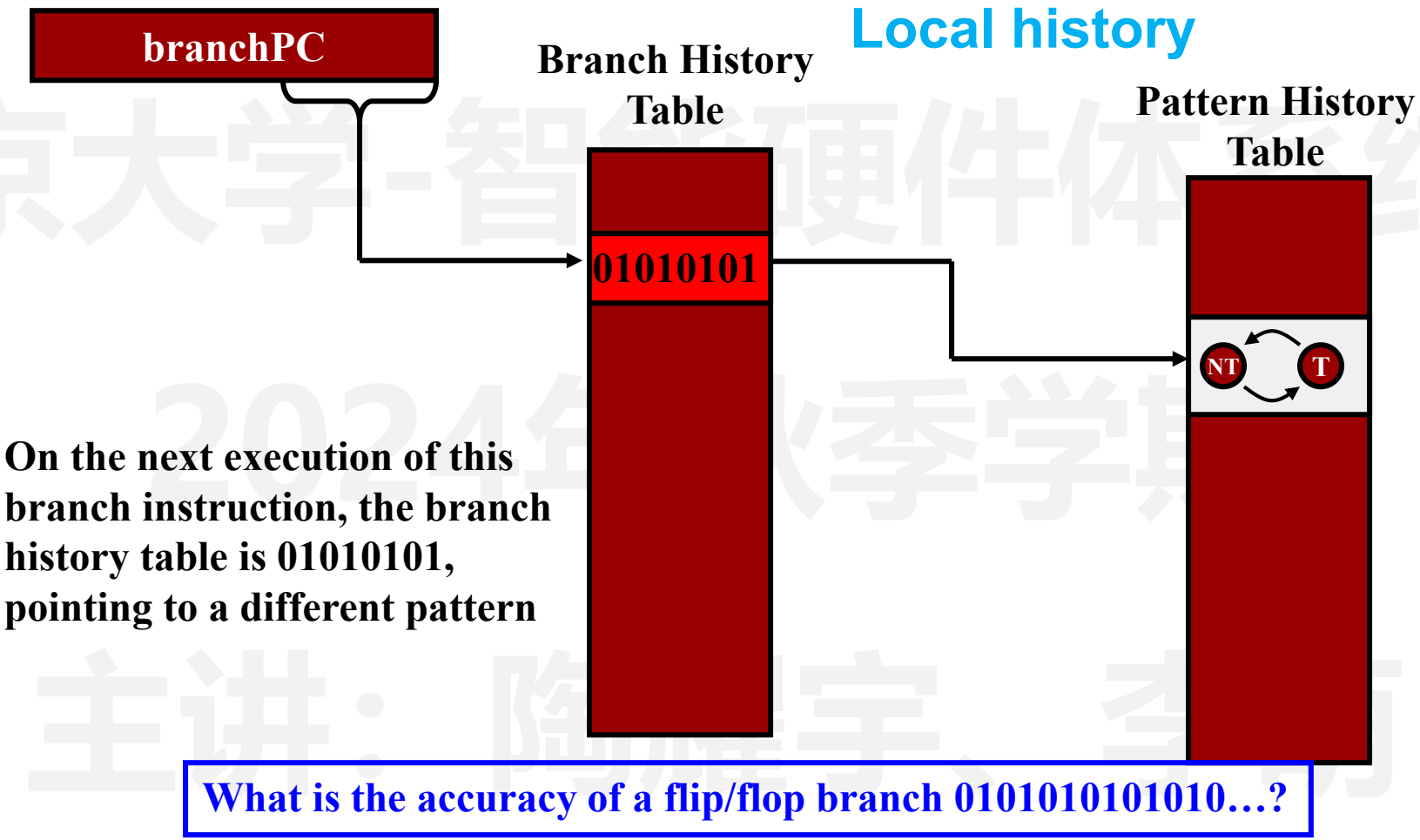
分支预测

- 方向预测 – 基于历史的简单状态机FSM



分支预测

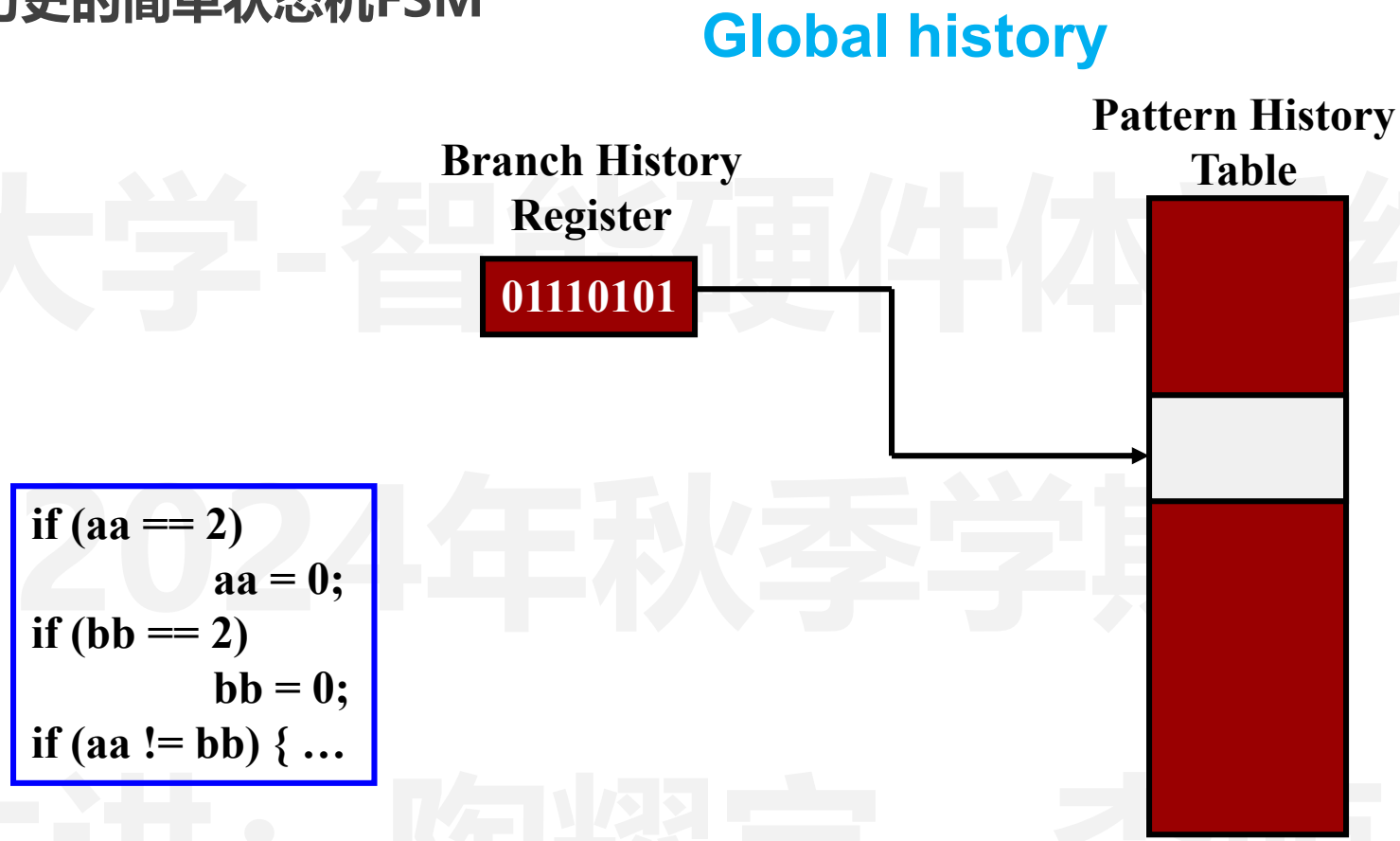
- 方向预测 – 基于历史的简单状态机FSM



Using History Patterns

分支预测

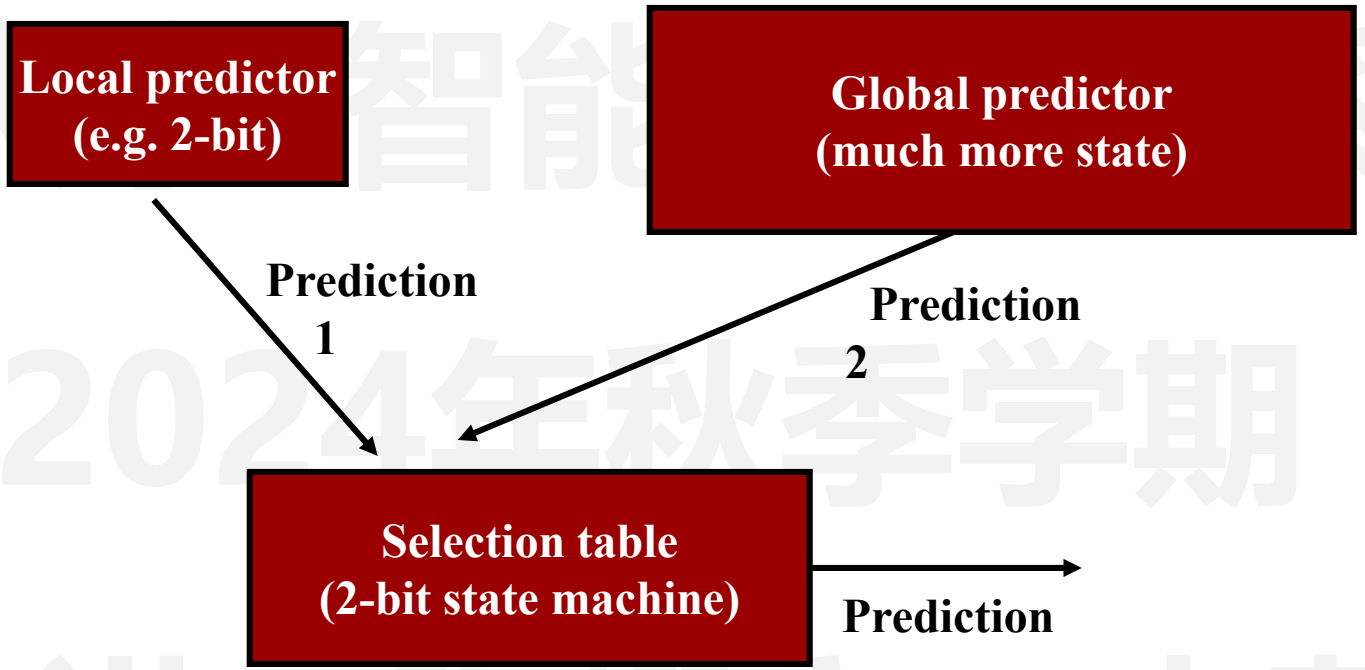
- 方向预测 – 基于历史的简单状态机FSM



Using History Patterns

- 方向预测 – 基于历史的简单状态机FSM

Hybrid predictors



How do you select which predictor to use?
How do you update the various predictor/selector?

Using History Patterns

分支预测

北京大学 PEKING UNIVERSITY

- **地址预测 – Branch Target Buffer**

- BTB indexed by current PC
  - If entry is in BTB fetch target address next

Branch PC	Target address
0x05360AF0	0x05360000
…	…
…	…
…	…
…	…
…	…

分支流水线处理机制

- 对于顺序多级流水线比较简单，对乱序执行需要额外的机制

顺序多级流水线

- Squash and restart fetch with right address
  - Just have to be sure that nothing has "committed" its state yet.
- In our 5-stage pipe, state is only committed during MEM (for stores) and WB (for registers)

Tamosulo

- Recovery seems really hard
  - What if instructions after the branch finish before we find that the branch was wrong?
    - This could happen. Imagine

      R1=MEM[R2+0]

      BEQ R1, R3 DONE ← Predicted not taken

      R4=R5+R6
  - So we have to not speculate on branches or not let anything pass a branch
    - Branches become serializing instructions.
      - Note that can be executing some things before and after the branch once branch resolves.

目 录

CONTENTS

分支处理机制与地址预测

北京大学 PEKING UNIVERSITY

- **Tomasulo动态发射的潜在问题**

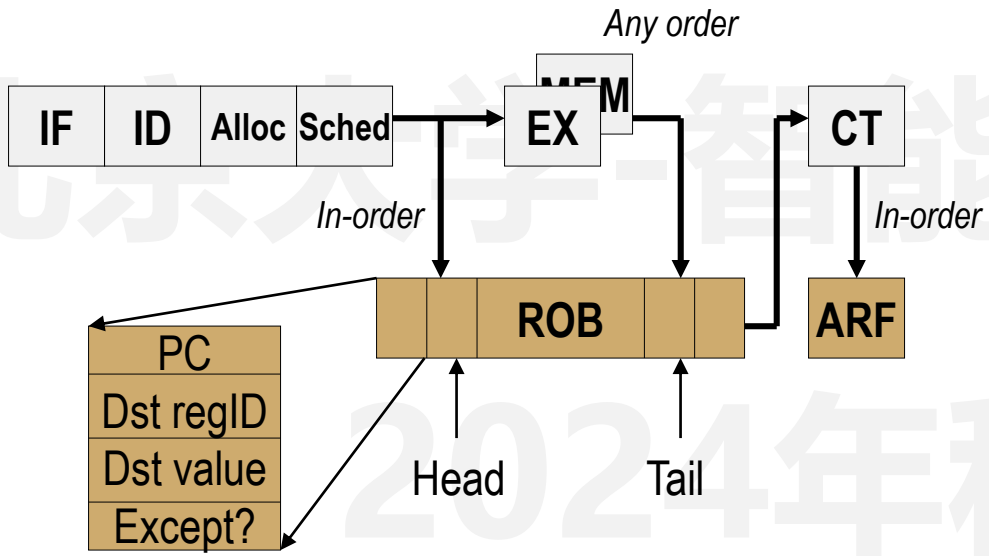
  - When can Tomasulo go wrong?
    - Branches
      - What if a branch finishes after younger instructions (after the branch) finish?
    - Exceptions!!
      - No way to figure out relative order of instructions in RS
      - **We need a mechanism to predict branch results**
      - **We need a mechanism to ensure finish in order**

MIPS R10K：超标量+动态指令发射

- **Adding a Reorder Buffer, aka ROB**
  - Why need Reorder Buffer
    - ROB is an *in-order* queue where instructions are placed.
    - **Instructions *complete* (retire) in-order**
    - **Instructions still *execute* out-of-order**
    - Still use RS
      - **Instructions are issued to RS and ROB at the same time**
      - Rename is to ROB entry, not RS.
      - When *execute* done instruction leaves RS
    - Only when all instructions in before it in program order are done does the instruction retire.

MIPS R10K：超标量+动态指令发射

- Adding a Reorder Buffer, aka ROB

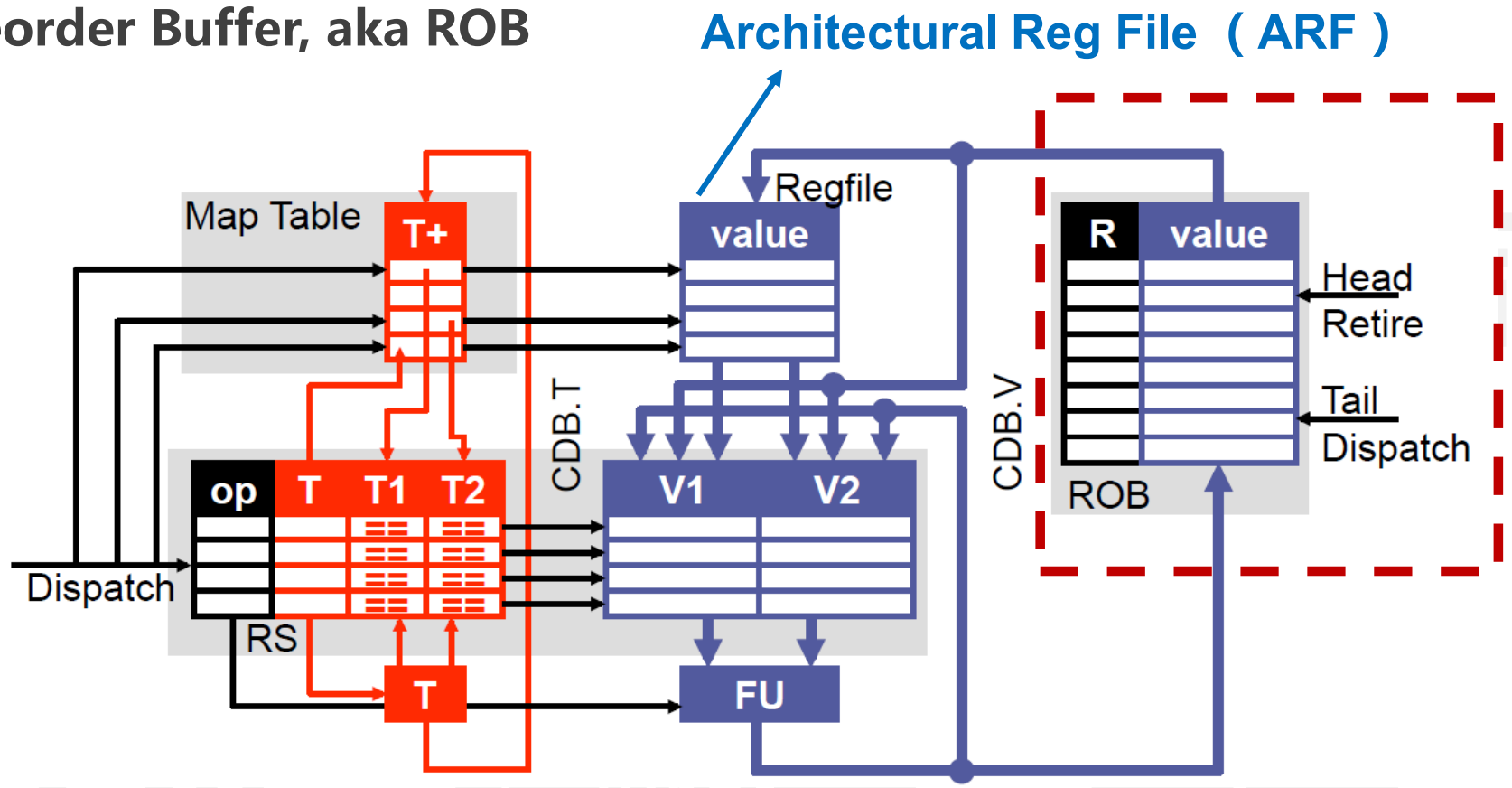


- Reorder Buffer (ROB)
  - Circular queue of spec state
  - May contain multiple definitions of *same* register

- @ **Alloc**
  - Allocate result storage at Tail
- @ **Sched**
  - Get inputs (ROB T-to-H then ARF)
  - Wait until all inputs ready
- @ **WB**
  - Write results/fault to ROB
  - Indicate result is ready
- @ **CT**
  - Wait until inst @ Head is done
  - If fault, initiate handler
  - Else, write results to ARF
  - Deallocate entry from ROB

MIPS：超标量+动态指令发射

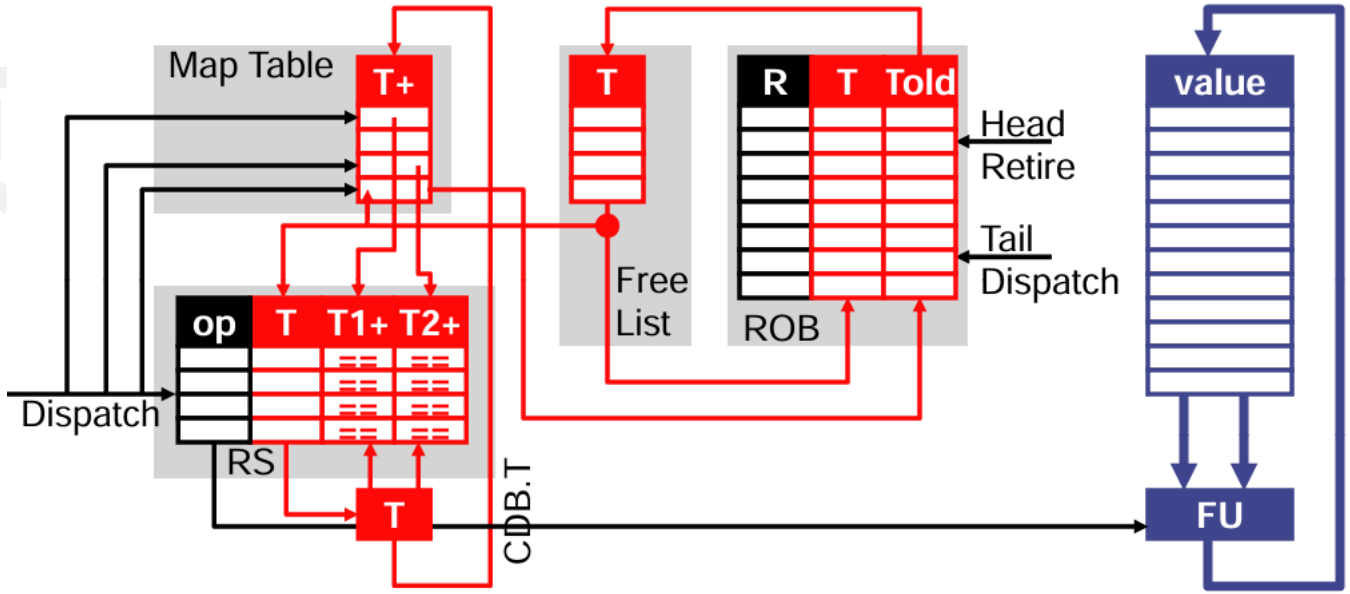
- Adding a Reorder Buffer, aka ROB



- **Simple Tomasulo+ROB**
  - Too much value movement(regfile/ROB→RS→ROB→regfile)
  - Multi input muxes long buses complicate routing and slow clock

MIPS：超标量+动态指令发射

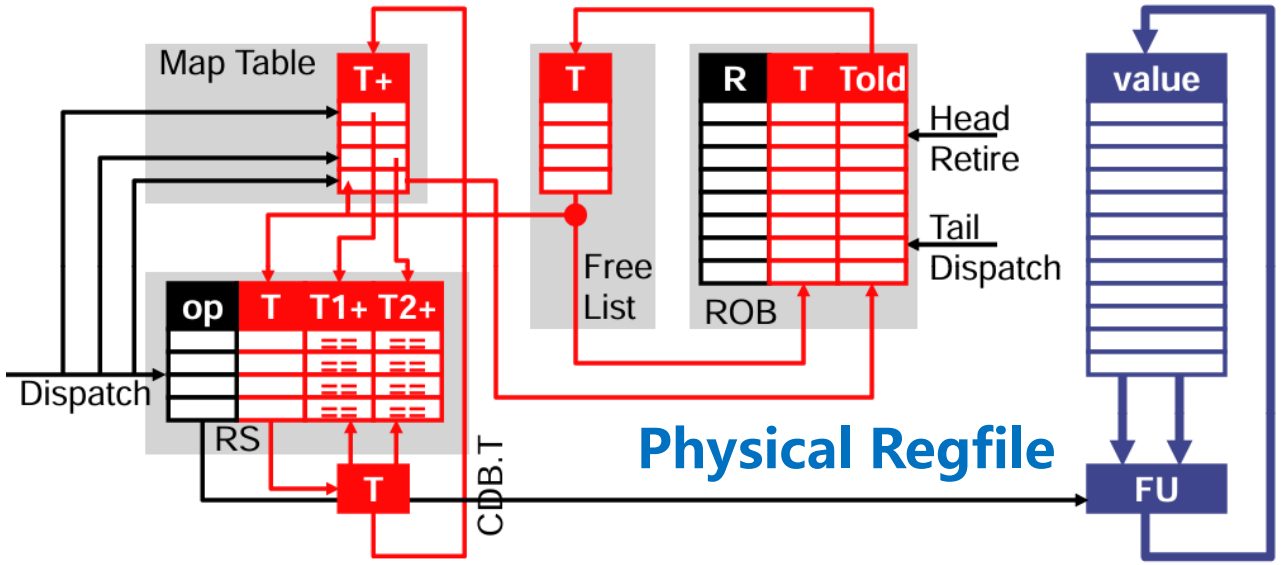
- MIPS：An alternative implementation



- **One big physical register file holds all data - no copies**
  - + Register file close to FUs → small fast data path ROB
  - ROB and RS “on the side” used only for control and tags

MIPS：超标量+动态指令发射

- MIPS：An alternative implementation



- **Architectural register file? Gone**
- **Physical register file holds all values**
  - #physical registers = #architectural registers + #ROB entries
  - Map architectural registers to physical registers
  - Removes data hazards (physical registers replace RS copies)

- Fundamental change to **map table/RAT**
  - Mappings cannot be 0 (there is no architectural register file)
- **Free list** keeps track of unallocated physical registers
  - ROB is responsible for returning physical registers to free list
- Conceptually, this is “true register renaming” without value moving

MIPS：超标量+动态指令发射

- MIPS：An alternative implementation

Parameters

- Names: r1 , r2 , r3
- Locations: p1 , p2 , p3 , p4 , p5 , p6 , p7
- Original mapping: r1→p1, r2→p2, r3→p3, p4–p7 are “free”

MapTable

r1	r2	r3
p1	p2	p3
p4	p2	p3
p4	p2	p5
p6	p2	p5

FreeList

p4 , p5 , p6 , p7
p5 , p6 , p7
p6 , p7
p7

Raw insns	Renamed insns
add r2 , r3 , r1	add p2 , p3 , p4
sub r2 , r1 , r3	sub p2 , p4 , p5
mul r2 , r3 , r1	mul p2 , p5 , p6
div r1 , r3 , r2	div p4 , p5 , p7

Question: how is the insn after div renamed?

- We are out of free locations (physical registers)
- Real question: how/when are physical registers freed?

MIPS：超标量+动态指令发射

- MIPS实例分析

New tags (again)

- Tomasulo+ROB：ROB# → MIPS：PR#

ROB

- **T**: physical register corresponding to insn's logical output
- **Told**: physical register previously mapped to insn's logical output

RS

- **T**, **T1**, **T2**: output, input physical registers

Map Table

- **T+**: PR# (never empty) + "ready" bit

Free List

- **T**: PR#

No values in ROB, RS, or on CDB

MIPS：超标量+动态指令发射

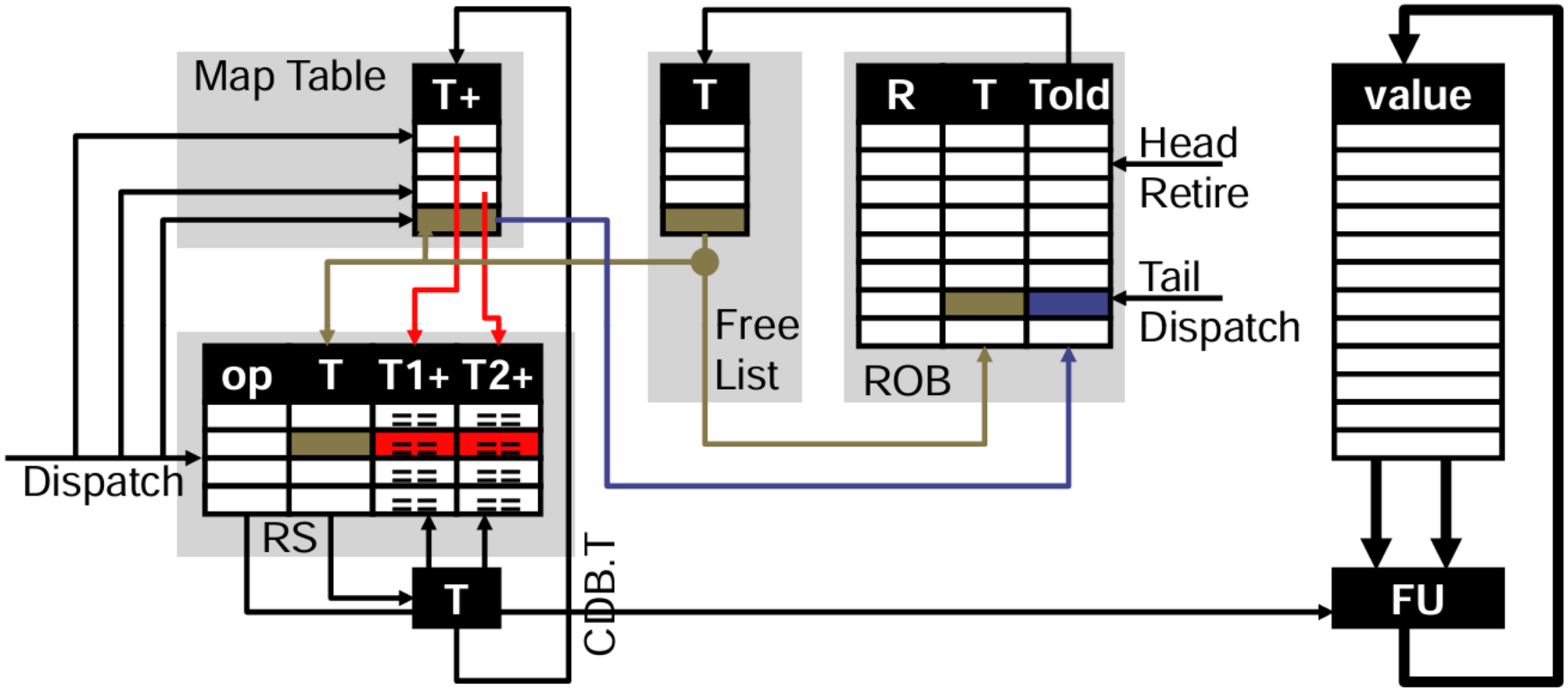
- MIPS实例分析

R10K pipeline structure: F, **D**, S, X, **C**, **R**

- **D (dispatch)**
  - Structural hazard (RS, ROB, LSQ, **physical registers**) ? stall
  - Allocate RS, ROB, LSQ entries and new physical register (T)
  - **Record previously mapped physical register (Told)**
- **C (complete)**
  - Write destination physical register
- **R (retire)**
  - ROB head not complete ? Stall
  - Handle any exceptions
  - Store write LSQ head to D$
  - Free ROB, LSQ entries
  - **Free previous physical register (Told)**

MIPS：超标量+动态指令发射

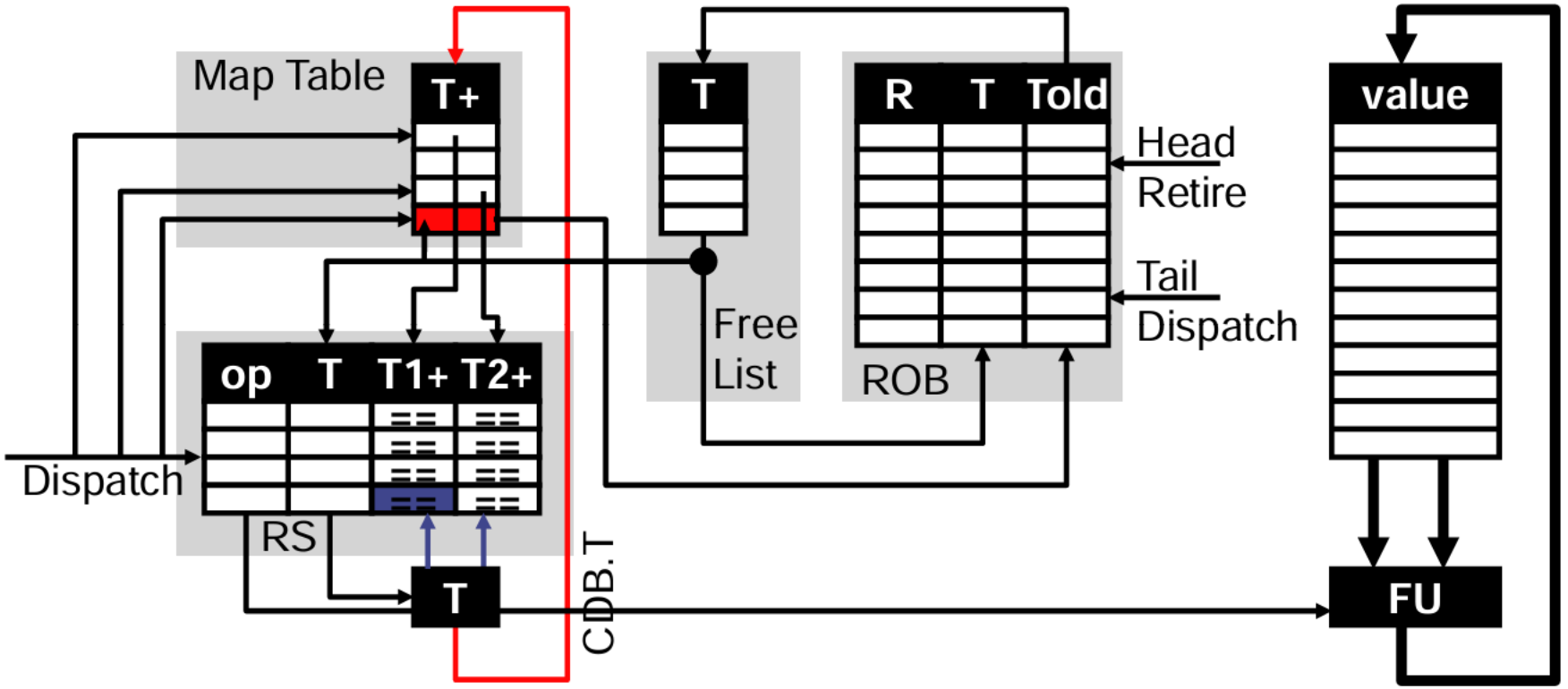
- MIPS R10K Dispatch步骤



- Read preg (physical register) tags for input registers, store in RS
- Read preg tag for output register, store in ROB (Told)
- Allocate new preg (free list) for output register, store in RS, ROB, Map Table

MIPS：超标量+动态指令发射

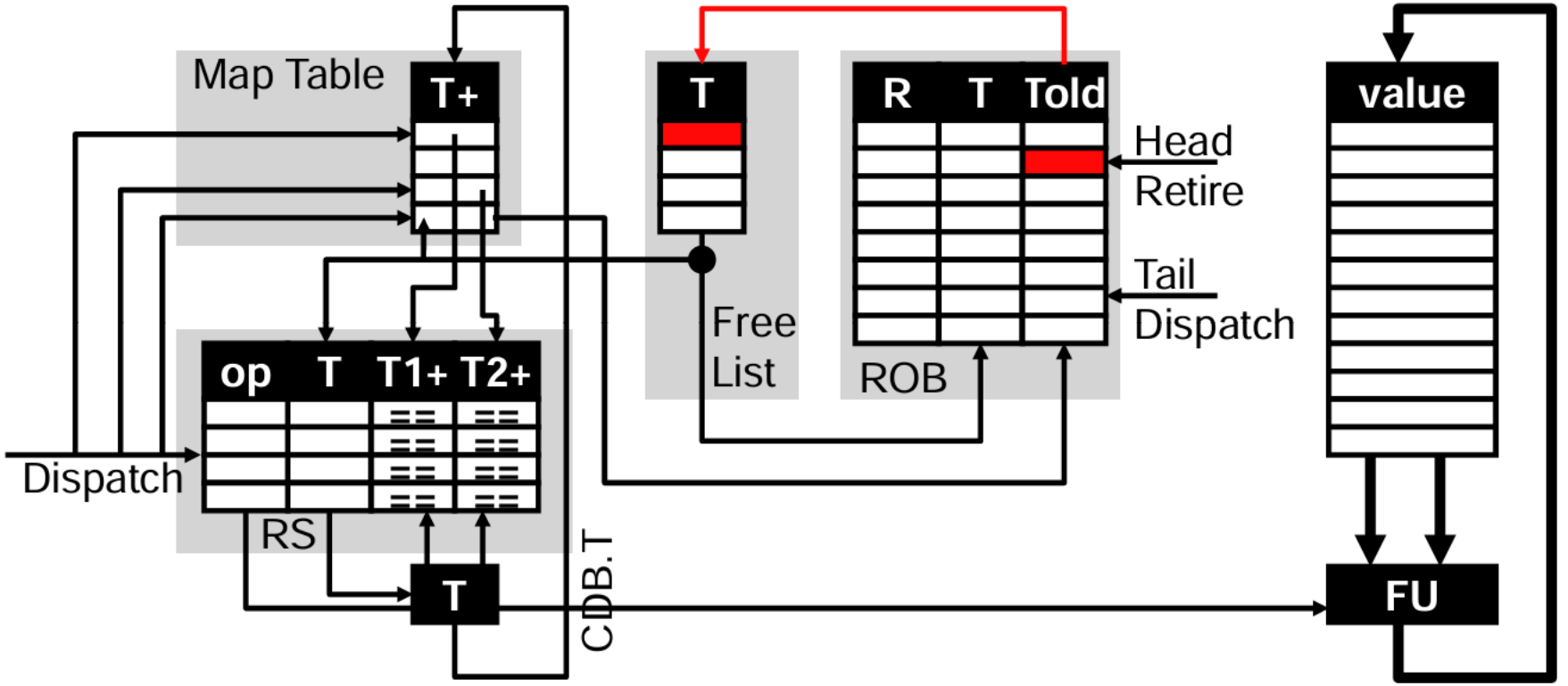
- MIPS R10K Complete步骤



- Set insn's output register ready bit in map table
- Set ready bits for matching input tags in RS

MIPS：超标量+动态指令发射

- MIPS R10K Retire步骤



- Return Told of ROB head to free list

MIPS：超标量+动态指令发射

- MIPS实例分析

ROB							
ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1					
	2	mulf f0,f1,f2					
	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#2+
f2	PR#3+
r1	PR#4+

CDB
T

Free List
PR#5,PR#6, PR#7,PR#8

+：Ready bit

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

Notice I: no values anywhere

Notice II: MapTable is never empty

MIPS：超标量+动态指令发射

- MIPS实例分析

MIPS:

Cycle 1

ROB

ht	#	Insn	T	Told	S	X	C
ht	1	ldf X(r1),f1	PR#5	PR#2			
	2	mulf f0,f1,f2					
	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#3+
r1	PR#4+

CDB

T

Free List

PR#5, PR#6, PR#7, PR#8

Reservation Stations

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

Allocate new preg (PR#5) to f1

Remember old preg mapped to f1 (PR#2) in ROB

MIPS：超标量+动态指令发射

- MIPS实例分析

MIPS:

Cycle 2

ROB							
ht	#	Insn	T	Told	S	X	C
h	1	ldf X(r1),f1	PR#5	PR#2	c2		
t	2	mulf f0,f1,f2	PR#6	PR#3			
	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#6, PR#7, PR#8

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

Allocate new preg (PR#6) to f2

Remember old preg mapped to f3 (PR#3) in ROB

MIPS：超标量+动态指令发射

- MIPS实例分析

MIPS:

Cycle 3

ROB							
ht	#	Insn	T	Told	S	X	C
h	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	
	2	mulf f0,f1,f2	PR#6	PR#3			
t	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#7,PR#8

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

Stores are not allocated pregs

Free

MIPS：超标量+动态指令发射

- MIPS实例分析

MIPS:

Cycle 4

ROB

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

CDB

T
PR#5

Free List

PR#7 , 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	yes	mulf	PR#6	PR#1+	PR#5+
5	FP2	no				

ldf completes
set MapTable ready bit

Match PR#5 tag from CDB & issue

MIPS：超标量+动态指令发射

- MIPS实例分析

MIPS:

Cycle 5

ROB							
ht	#	Insn	T	Told	S	X	C
	1	ldf X(r1),f1	PR#5	PR#2	c2	c3	c4
h	2	mulf 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				

ldf retires
Return PR#2 to free list

Free

MIPS：超标量+动态指令发射

北京大学 PEKING UNIVERSITY

- MIPS实例分析

- 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				

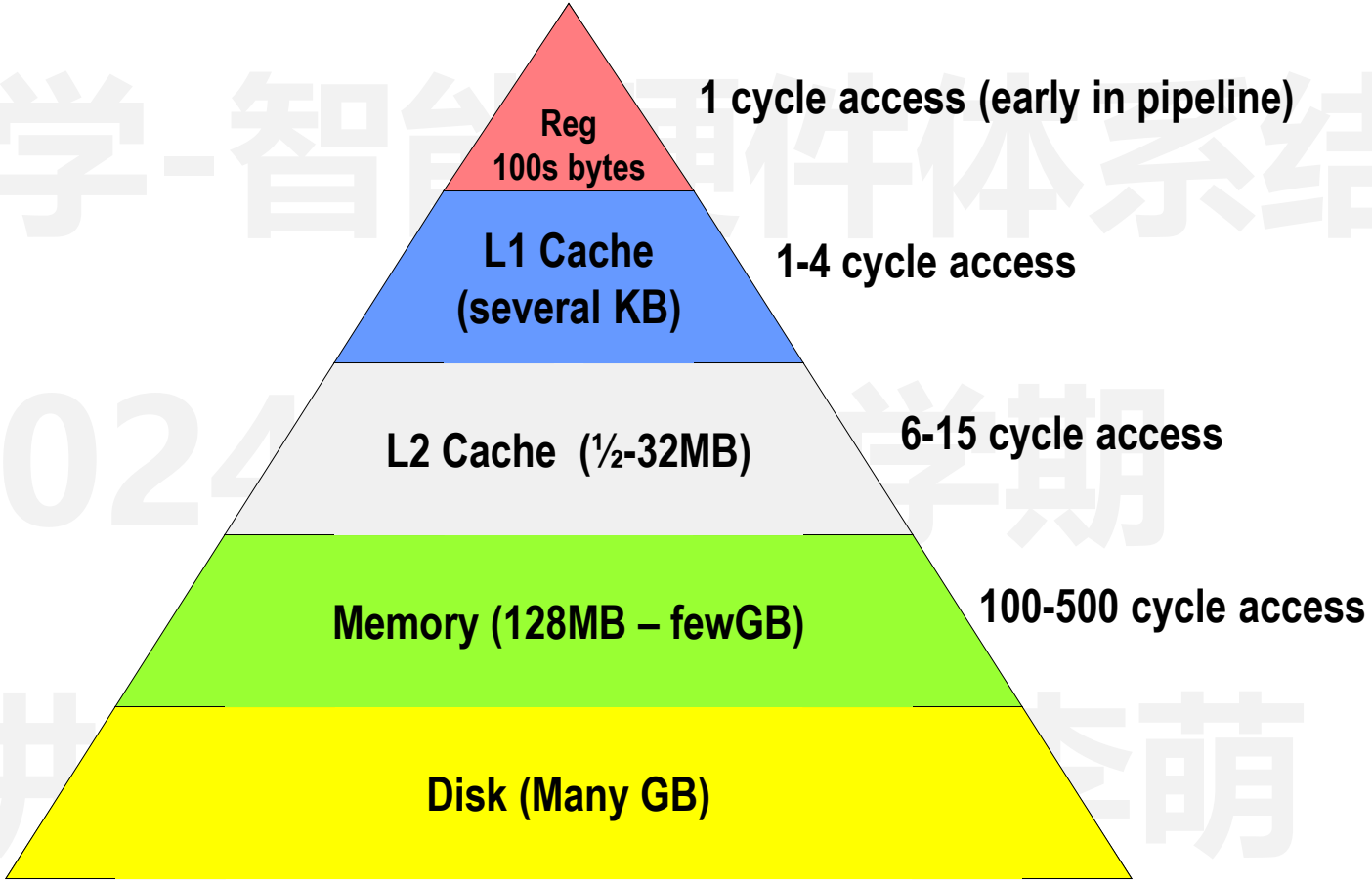
undo stf (ROB#3)
1. free RS
2. free ROB#3
3. no registers to restore/free
4. how is D$ write undone?

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

- 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:

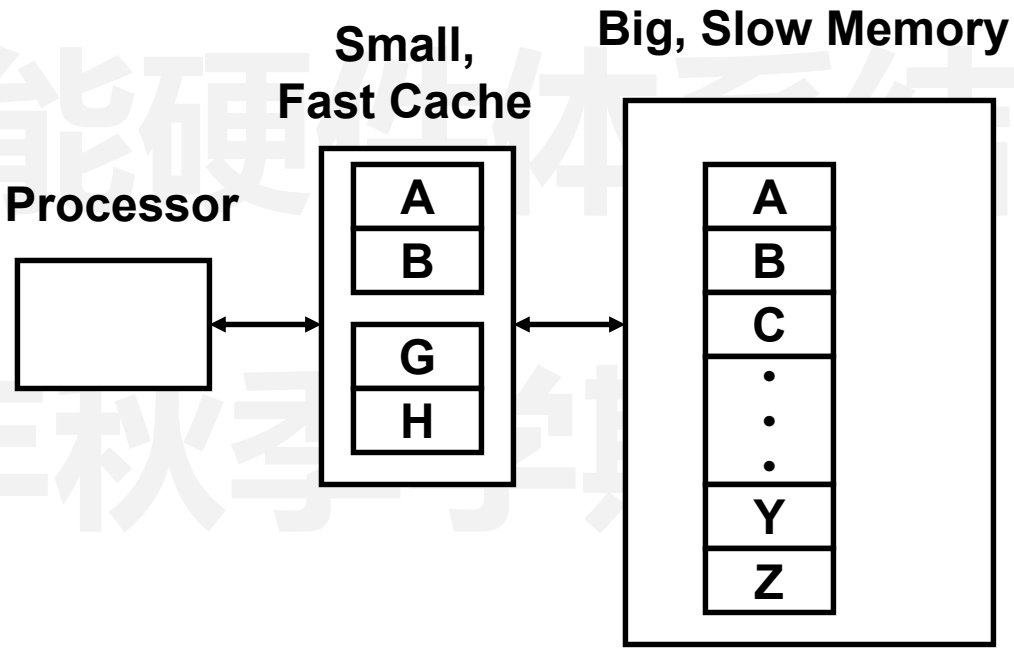
- **Data**
  - Reference array elements in succession (spatial)
- **Instructions**
  - Reference instructions in sequence (spatial)
  - Cycle through loop repeatedly (temporal)

```
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



Cache设计中的参数

- Cache Size、Associativity、Block Size、Number of Set, etc.

- Total cache size
  - (block size × # sets × associativity)
- Associativity (Number of “ways”)
- Block size (bytes per block)
- Number of sets

- **Performance Measures**
  - Miss rate
    - % of memory references which are not found in the cache.
    - A related measure is #misses per 1000 instructions
  - Average memory access time
    - $MR*T_{Miss} + (1-MR)* T_{Hit}$
      - T_{Hit} & T_{Miss} --Access time for a hit or miss
- But what do we want to measure?
  - Impact on program execution time.
- What are some flaws of using
  - Miss Rate?
  - Ave. Memory Access Time?
  - Program execution time?
  - Misses per 1000 instructions?

Cache设计选择的影响

- 总容量：一般来说，Cache总容量越大，Miss Rate越低

- *Total cache size?*
  - Positives:
    - Should decrease miss rate
  - Negatives:
    - May increase hit time
    - Increased area requirements
    - Increased power (mainly static)
      - Interesting paper:
        - » Krisztián Flautner, Nam Sung Kim, Steve Martin, David Blaauw, Trevor N. Mudge: *Drowsy Caches: Simple Techniques for Reducing Leakage Power*. ISCA 2002: 148-157

Cache设计选择的影响

北京大学 PEKING UNIVERSITY

- **Block Size**

  - *Bigger block size?*
    - Positives:
      - Exploit spatial locality ; reduce compulsory misses
      - Reduce tag overhead (bits)
      - Reduce transfer overhead (address, burst data mode)
    - Negatives:
      - Fewer blocks for given size; increase conflict misses
      - Increase miss transfer time (multi-cycle transfers)
      - Wasted bandwidth for non-spatial data

Cache设计选择的影响

- Associativity的选择

  - *Increasing associativity*
    - Positives:
      - Reduces conflict misses
      - Low-associativity cache can have pathological behavior (very high miss)
    - Negatives:
      - Increased hit time
      - More hardware requirements (comparators, muxes, bigger tags)
      - Minimal improvements past 4- or 8- way.

Cache设计选择的影响

- Cache Replacement策略

- Replacement Strategy: (for associative caches)
  - LRU: intuitive; difficult to implement with high assoc; worst case performance can occur (N+1 element array)
  - Random: Pseudo-random easy to implement; performance close to LRU for high associativity
  - Optimal: replace block that has next reference farthest in the future; hard to implement (need to see the future) ☺

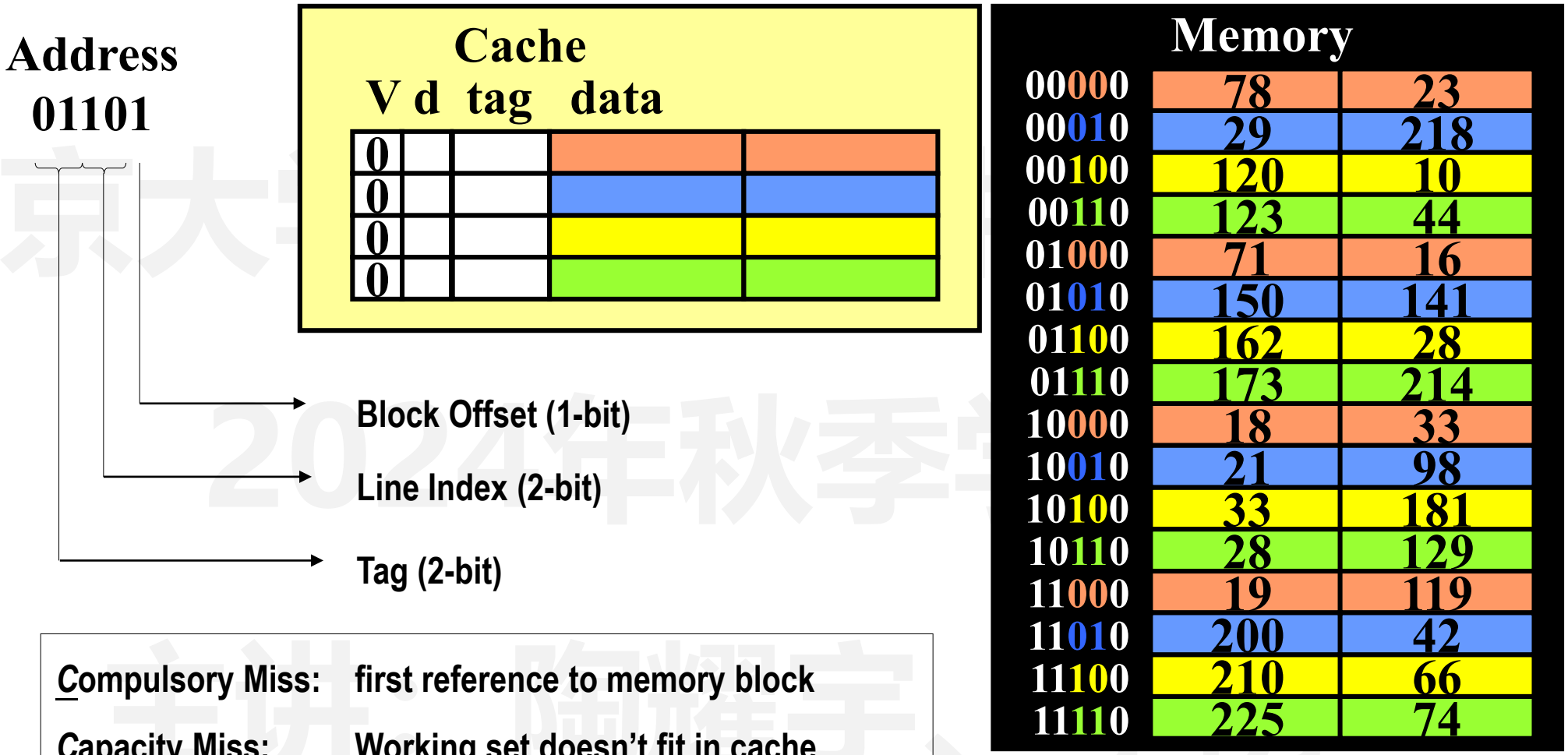
Cache设计选择的影响

北京大学 PEKING UNIVERSITY

- Cache写入和Miss处理策略

  - Write Policy: How to deal with write misses?
    - *Write-through / no-allocate*
      - Total traffic? Read misses × block size + writes
      - Common for L1 caches back by L2 (esp. on-chip)
    - *Write-back / write-allocate*
      - Needs a dirty bit to determine whether cache data differs
      - Total traffic? (read misses + write misses) × block size + dirty-block-evictions × block size
      - Common for L2 caches (memory bandwidth limited)
    - *Variation: Write validate*
      - Write-allocate without fetch-on-write
      - Needs sub-block cache with valid bits for each word/byte

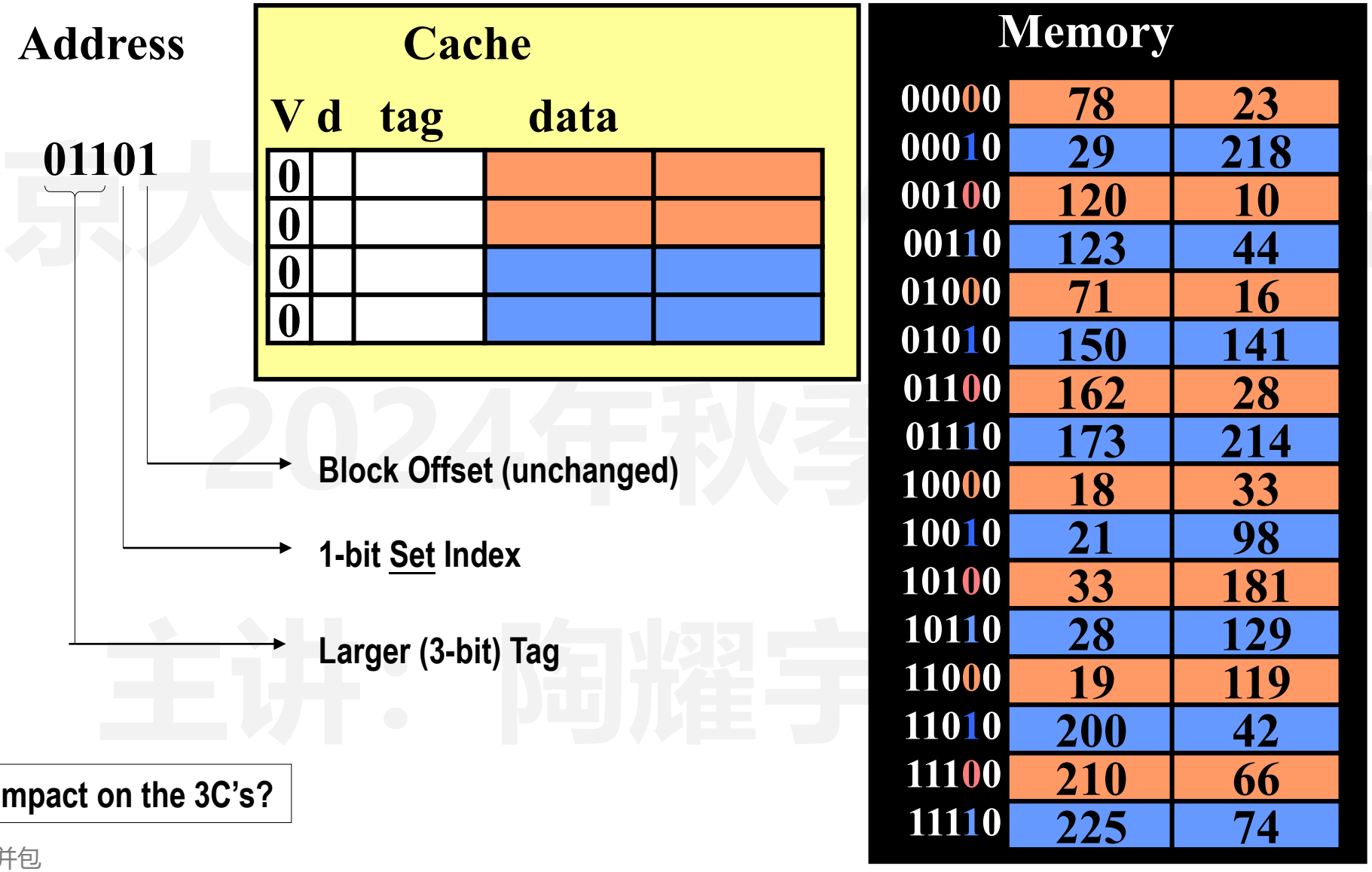
Direct-Mapped Cache设计



3-C's

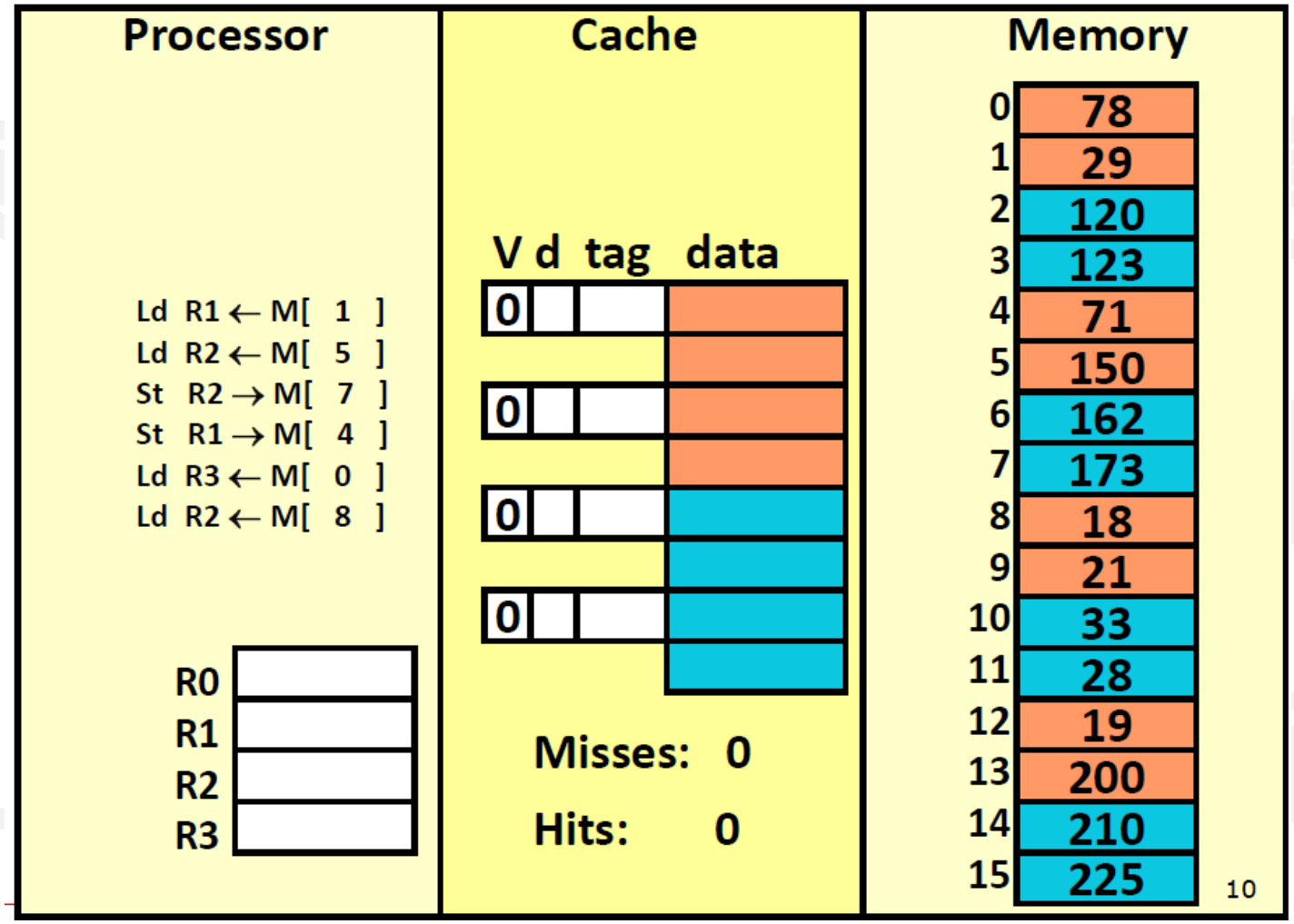
Compulsory Miss:	first reference to memory block
Capacity Miss:	Working set doesn't fit in cache
Conflict Miss:	Working set maps to same cache line

2-Way Set Associative Cache设计



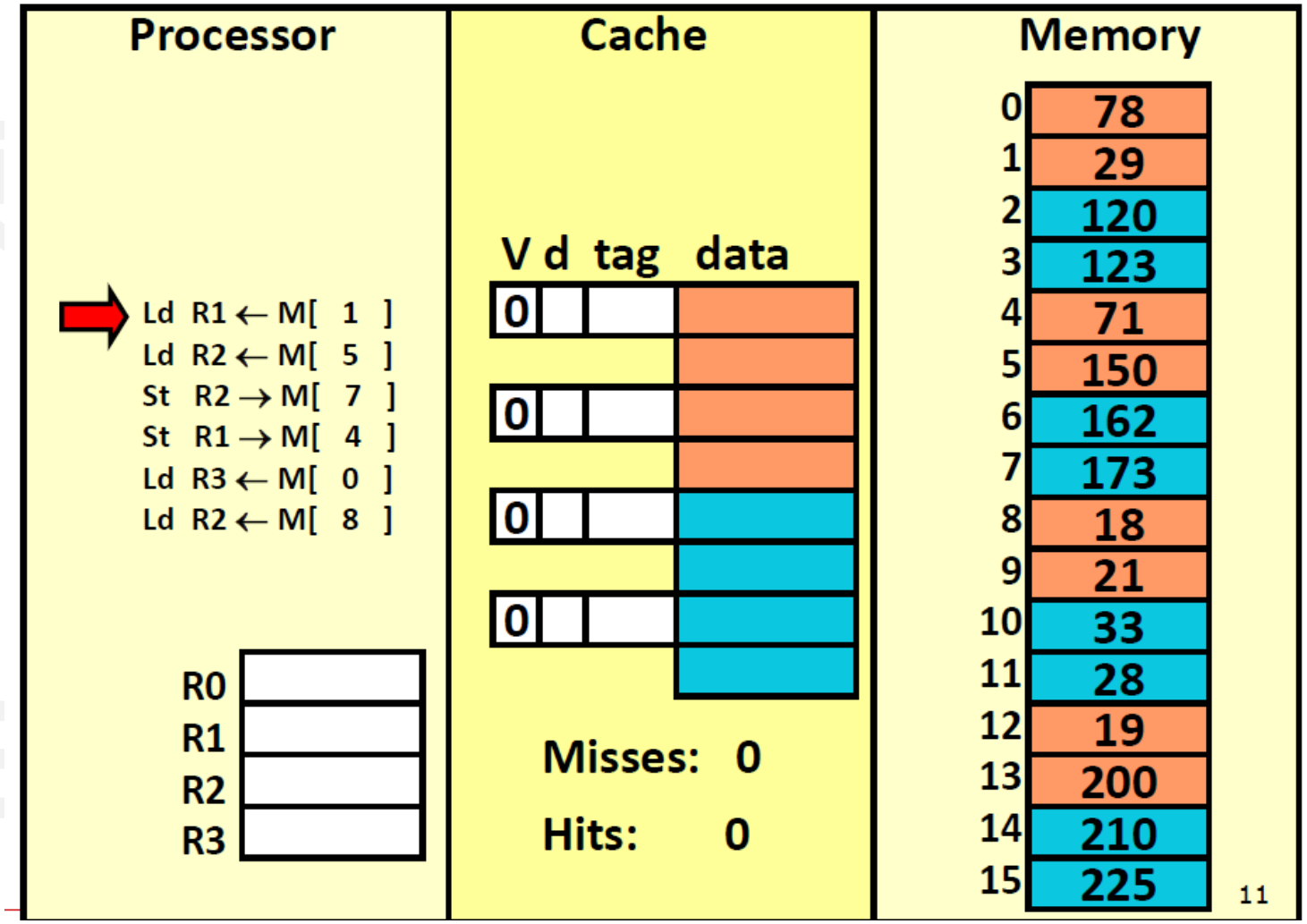
2-Way Set Associative Cache实例

- Write-back & Write-allocate



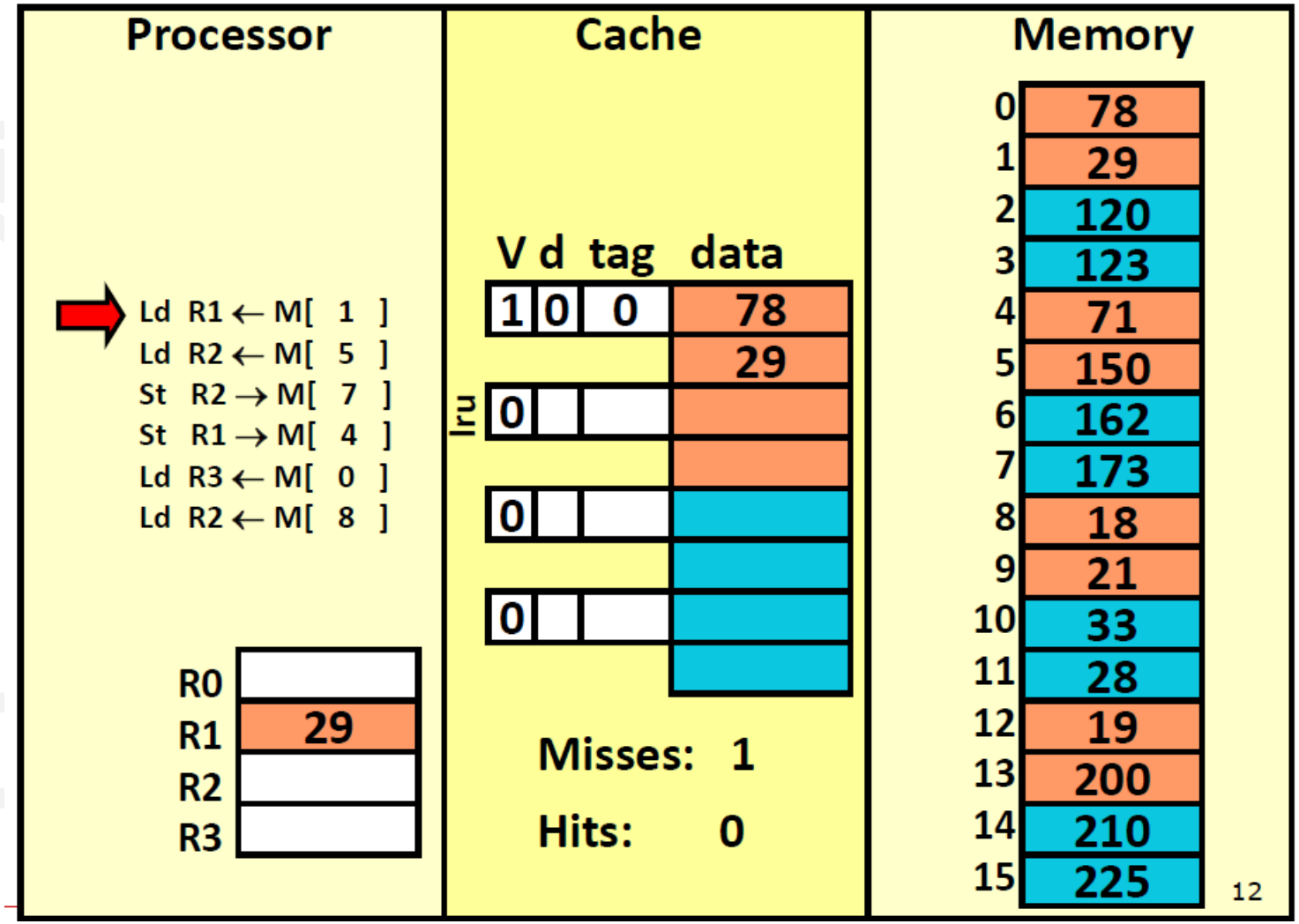
2-Way Set Associative Cache实例

- Write-back & Write-allocate



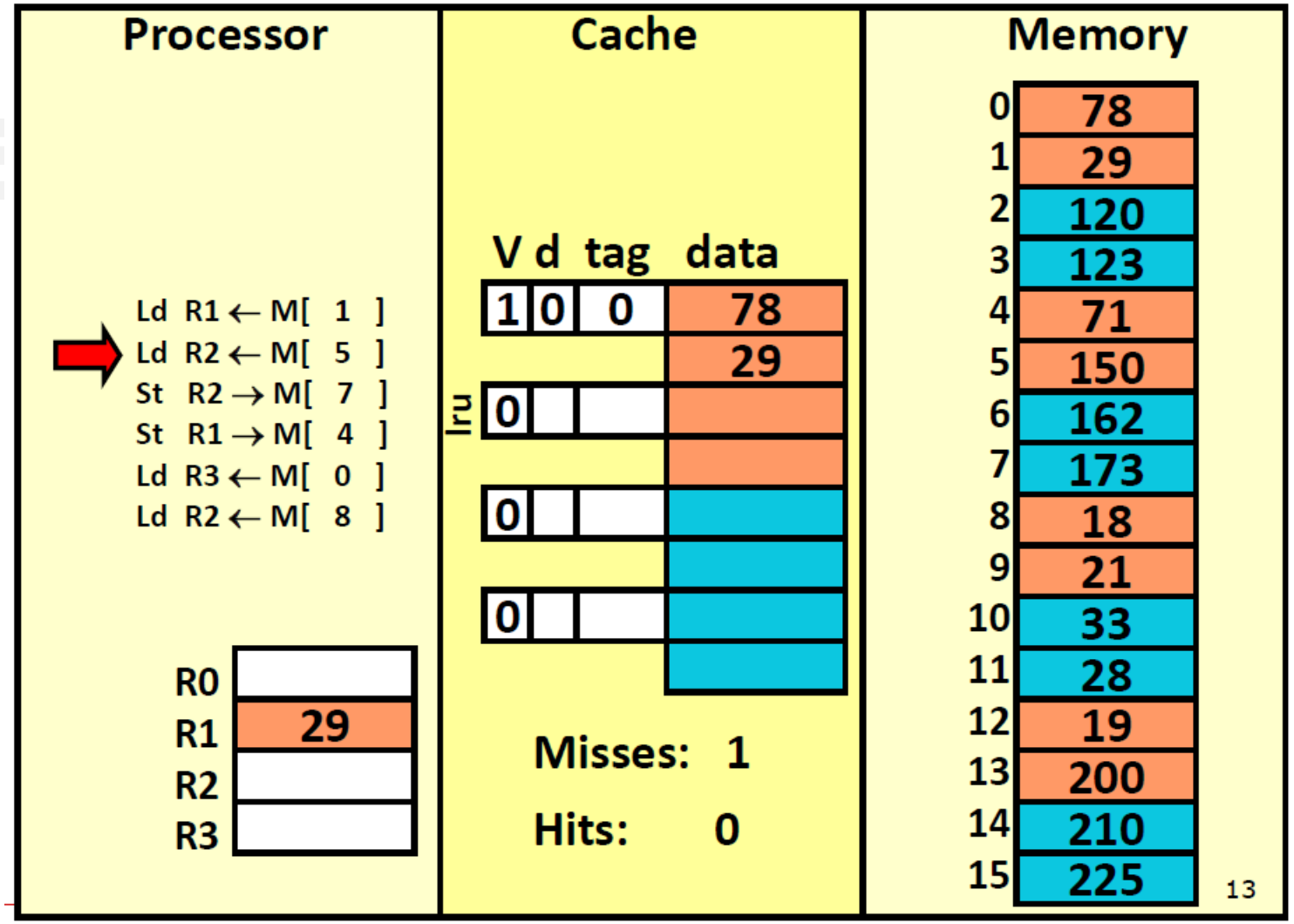
2-Way Set Associative Cache实例

- Write-back & Write-allocate



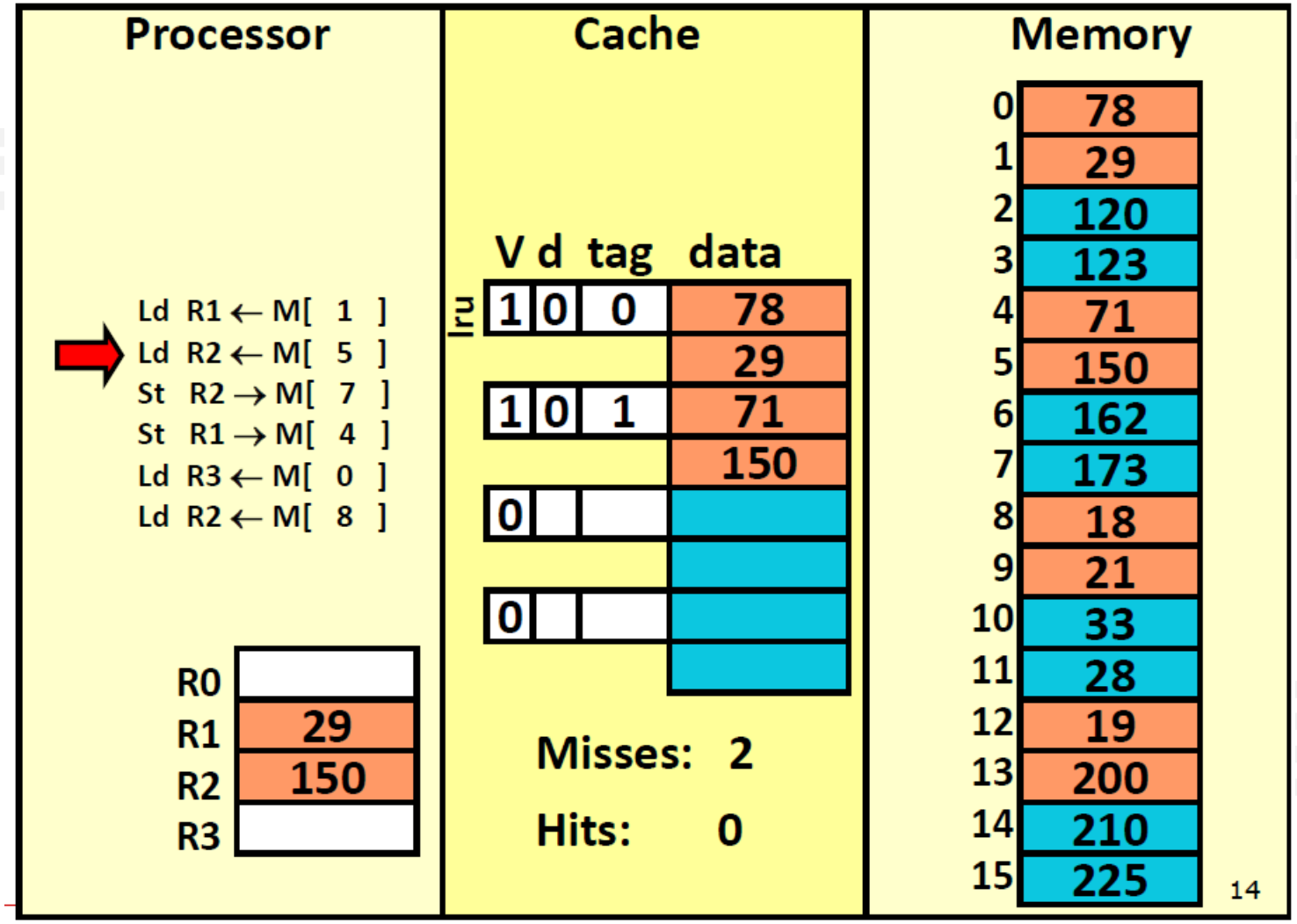
2-Way Set Associative Cache实例

- Write-back & Write-allocate



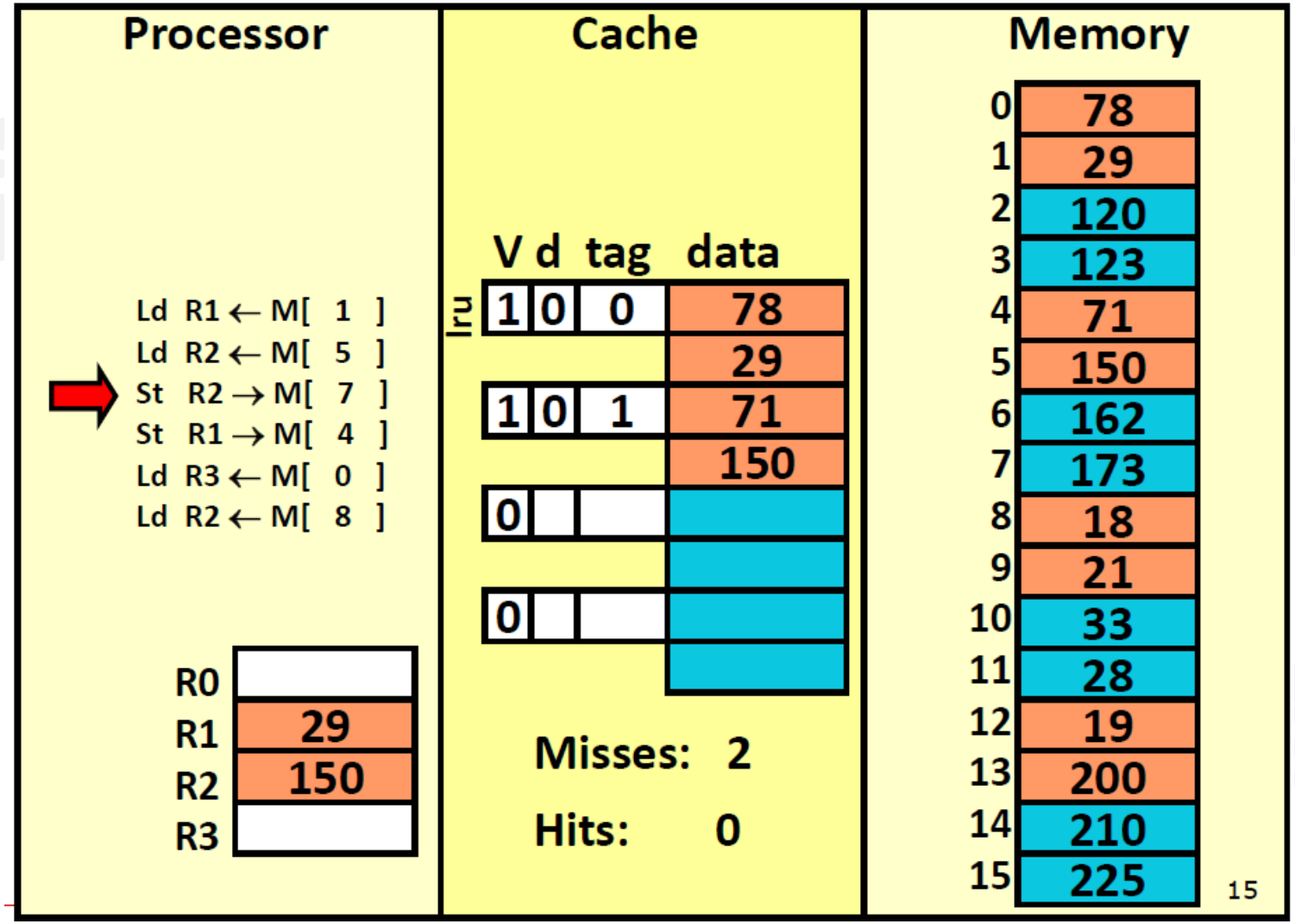
2-Way Set Associative Cache实例

- Write-back & Write-allocate



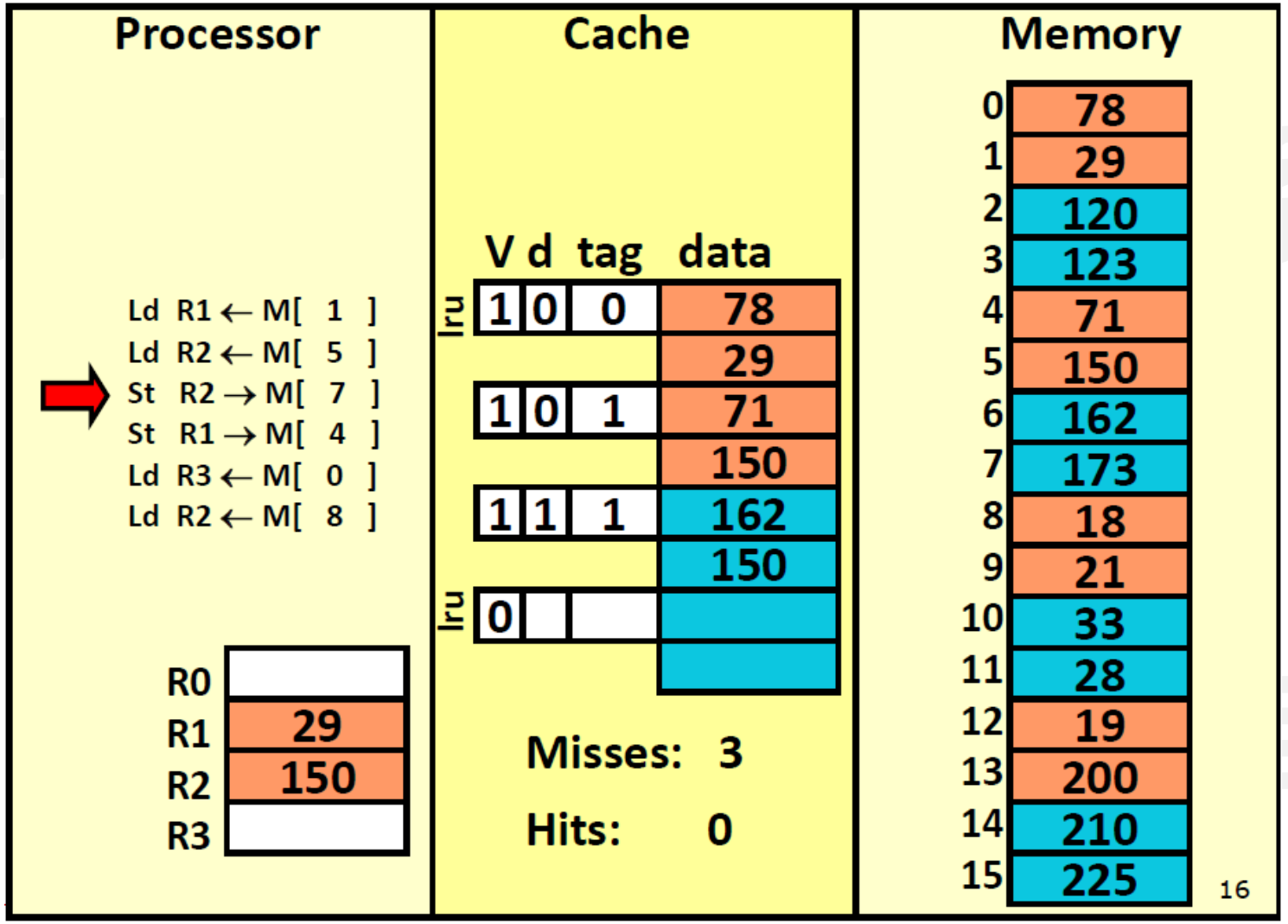
2-Way Set Associative Cache实例

- Write-back & Write-allocate



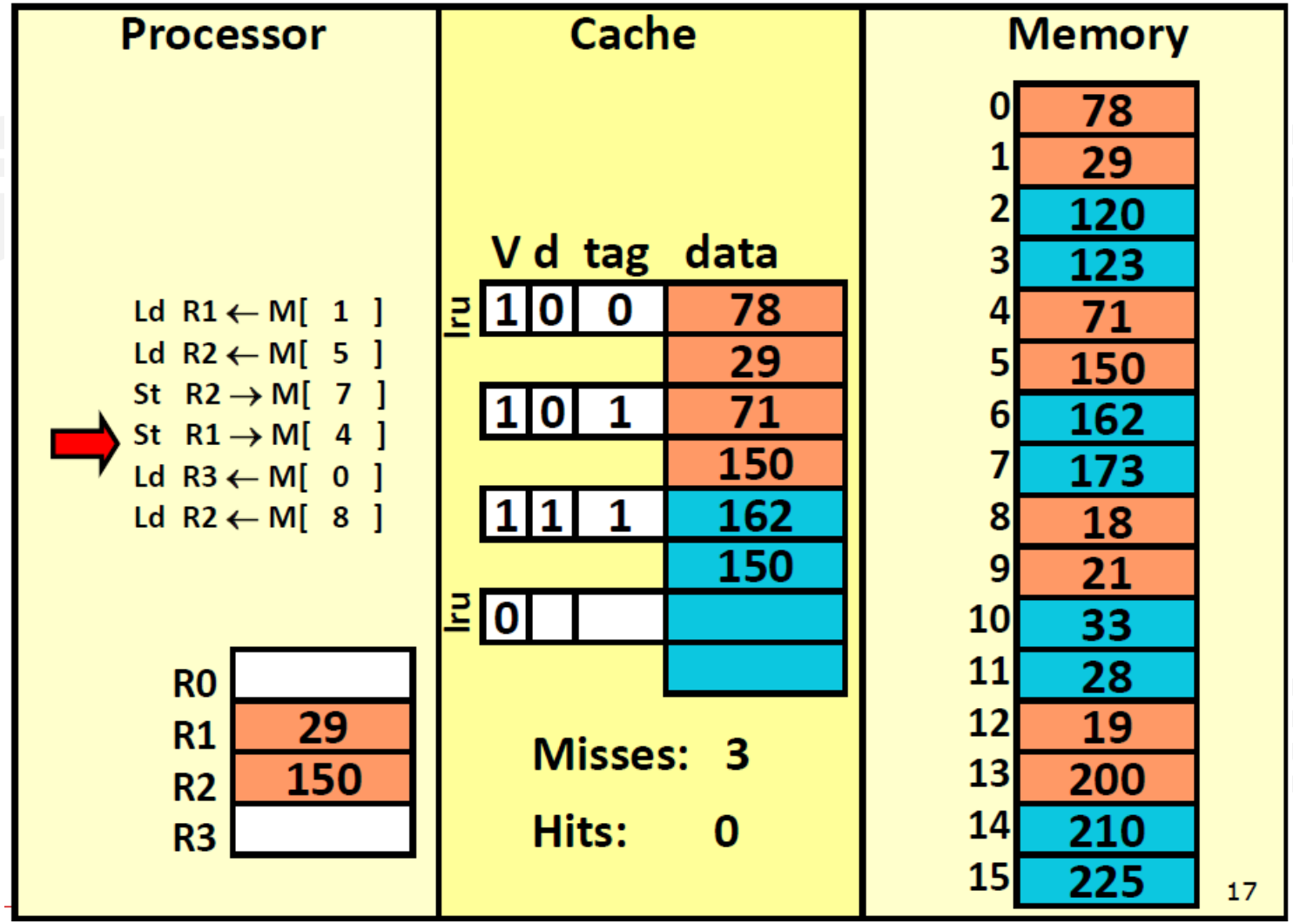
2-Way Set Associative Cache实例

- Write-back & Write-allocate



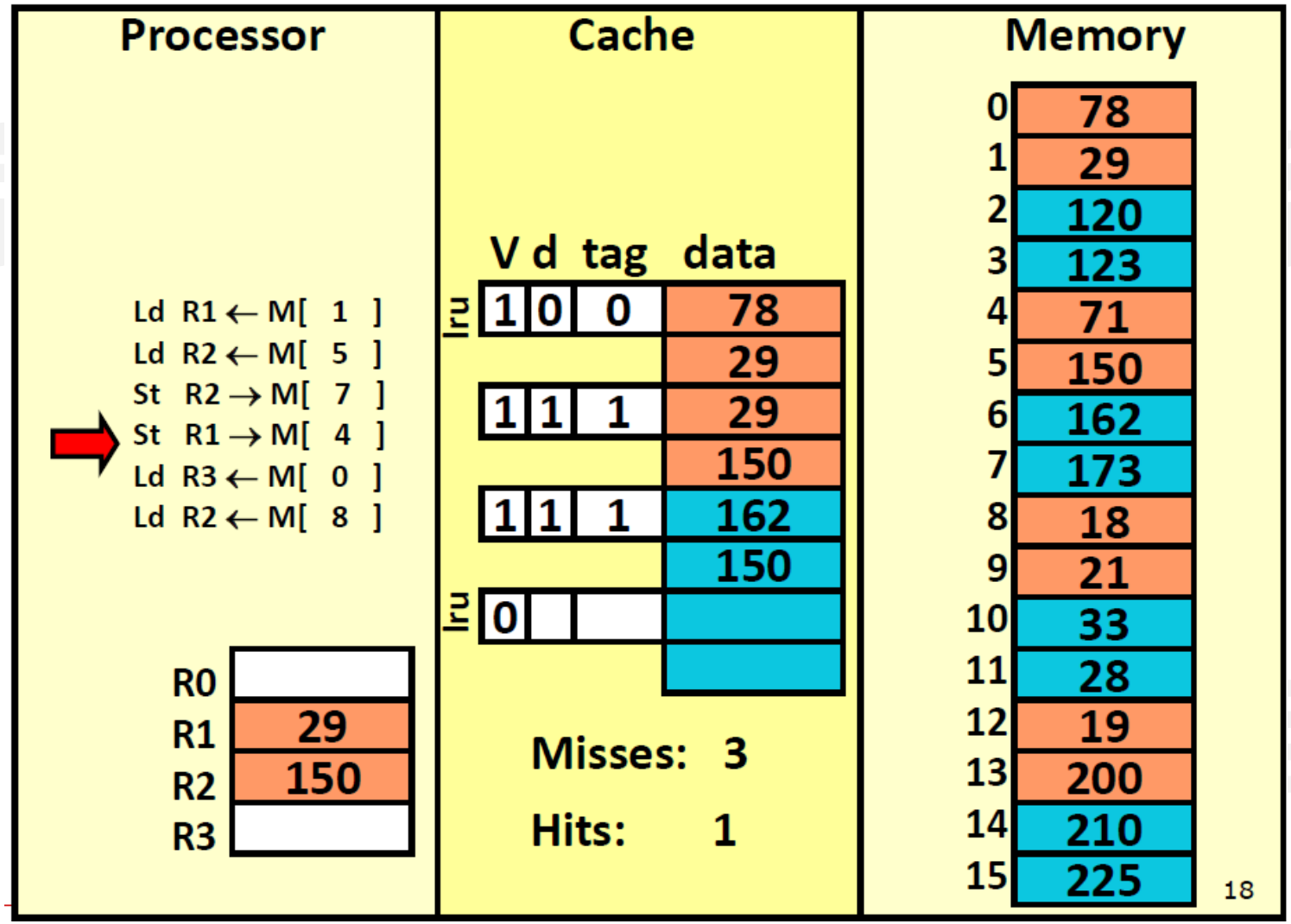
2-Way Set Associative Cache实例

- Write-back & Write-allocate



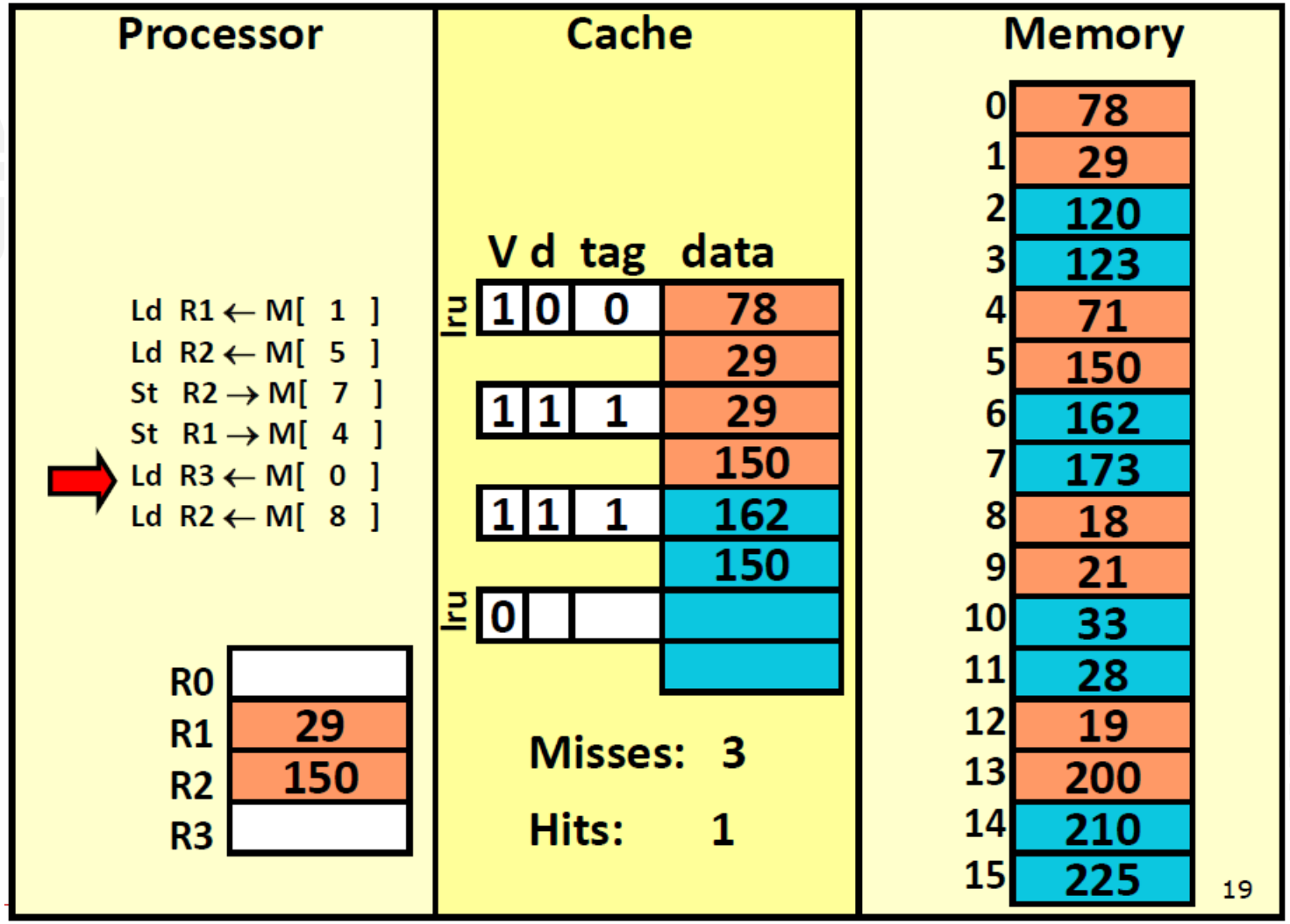
2-Way Set Associative Cache实例

- Write-back & Write-allocate



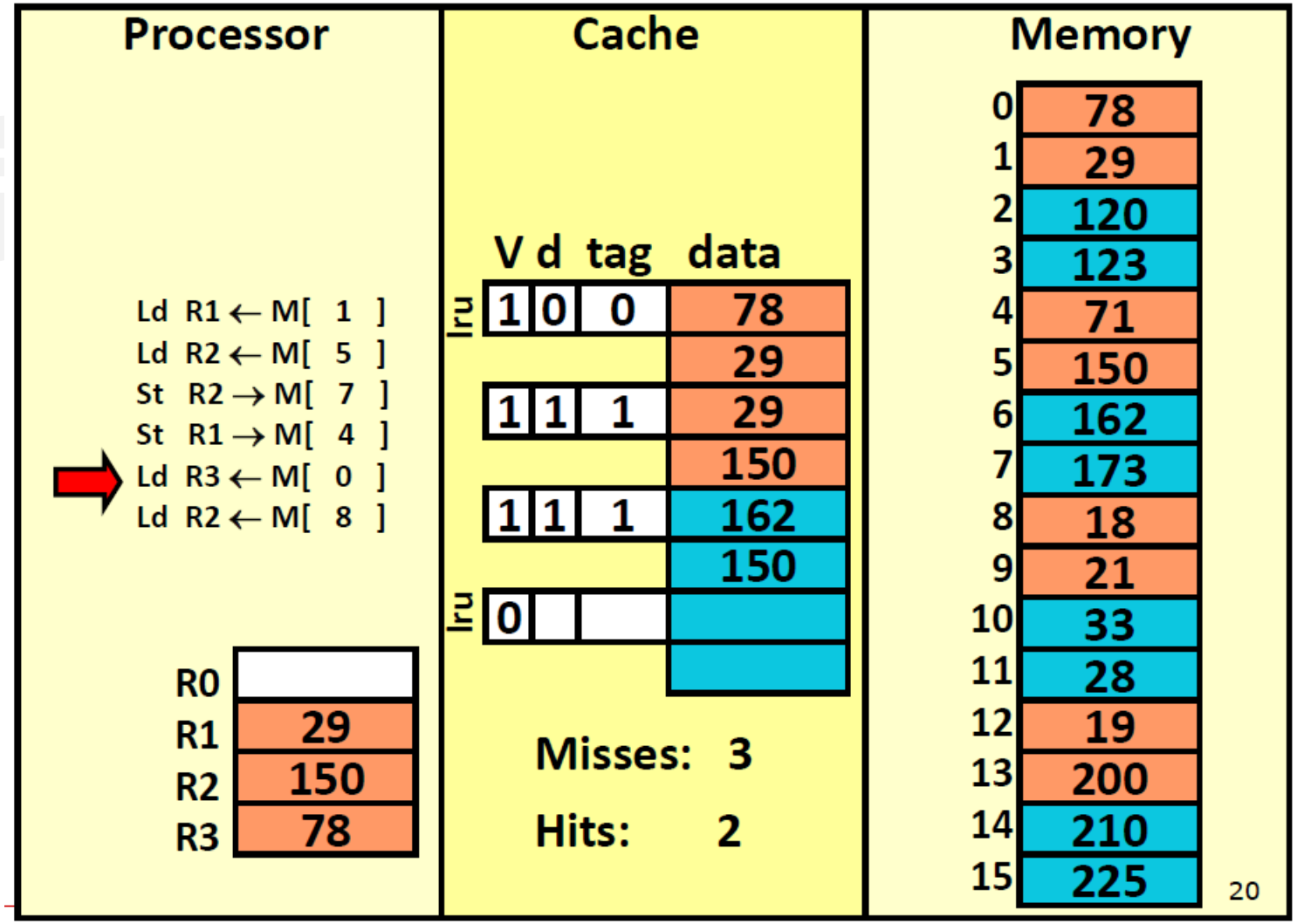
2-Way Set Associative Cache实例

- Write-back & Write-allocate



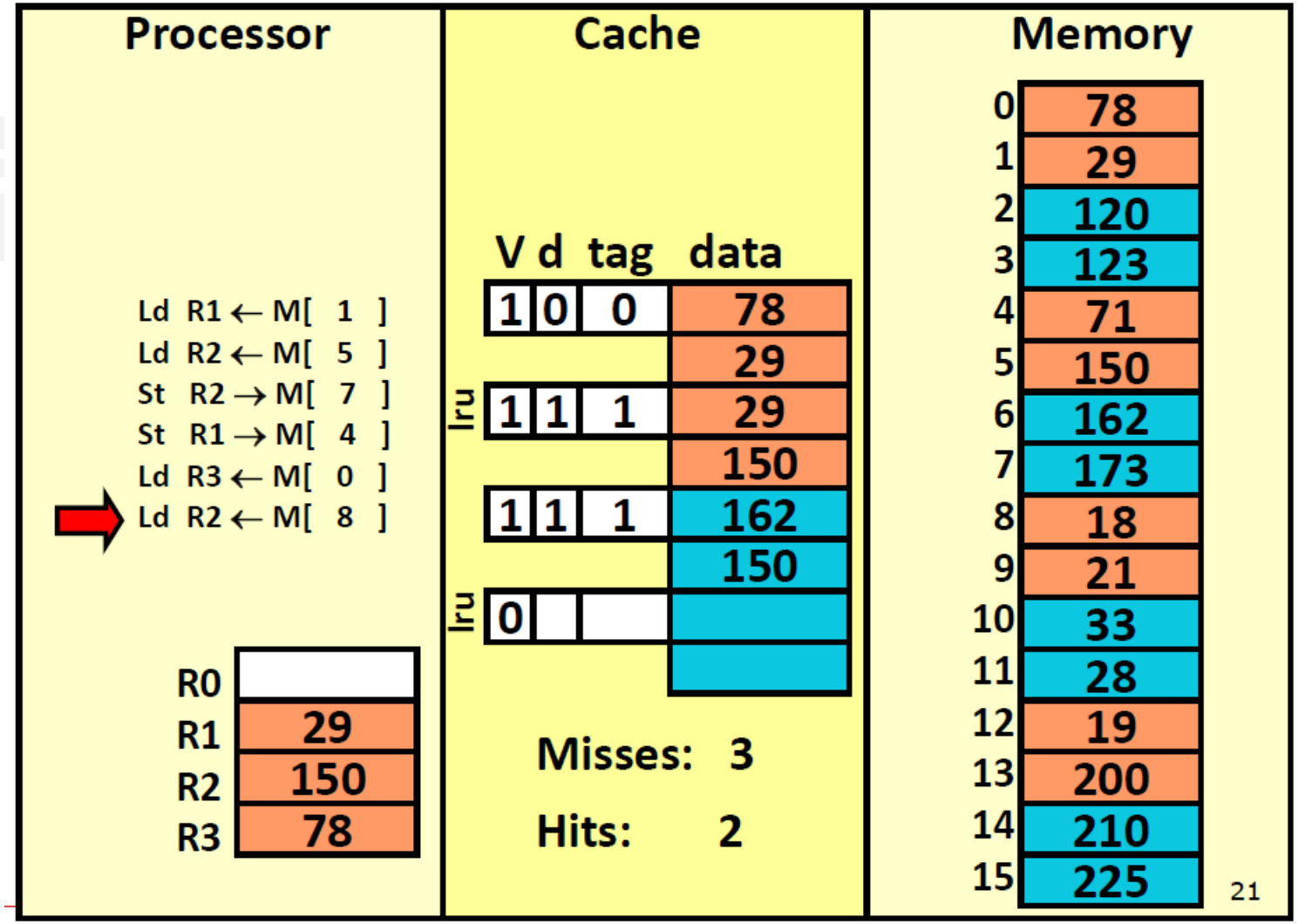
2-Way Set Associative Cache实例

- Write-back & Write-allocate



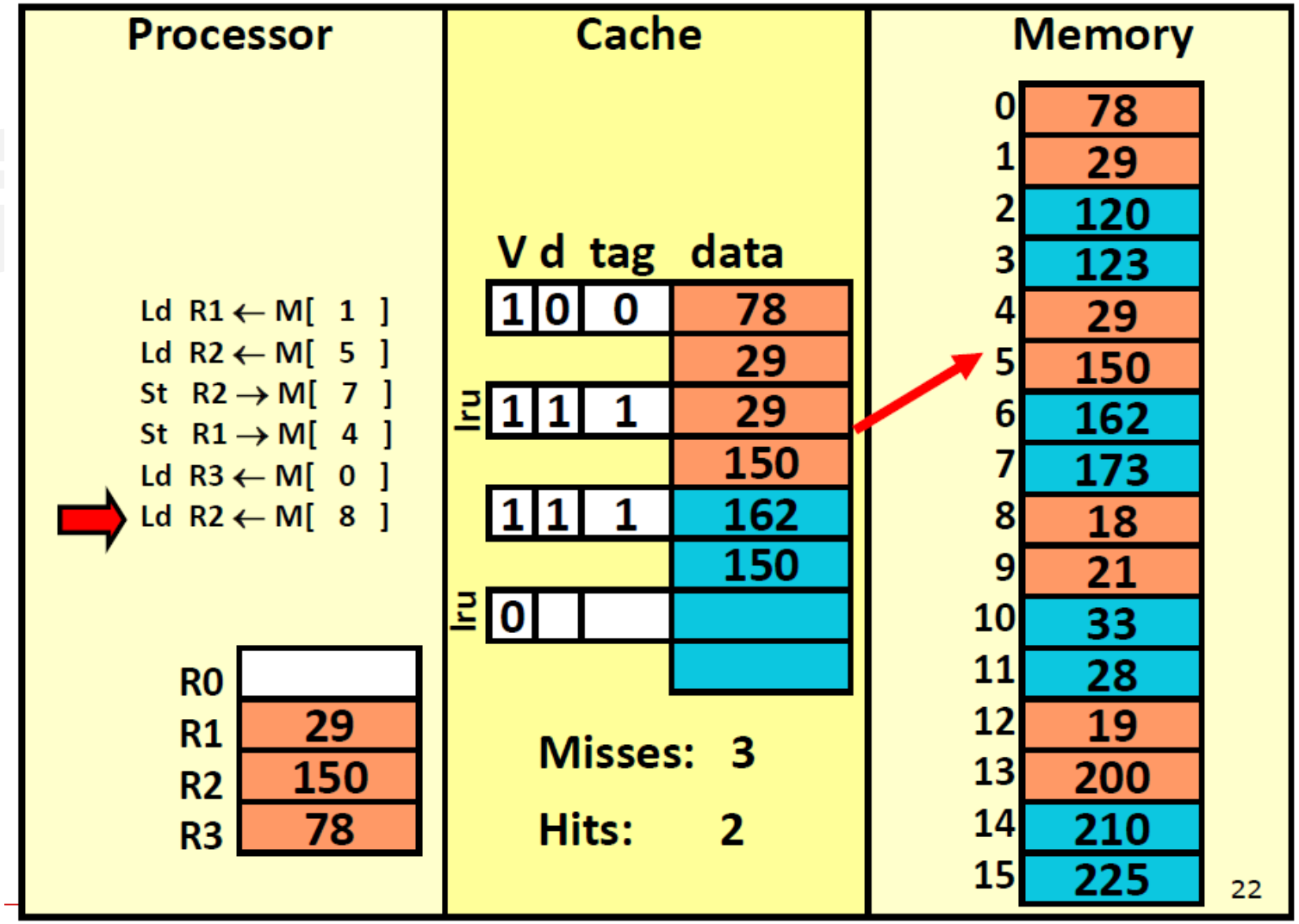
2-Way Set Associative Cache实例

- Write-back & Write-allocate



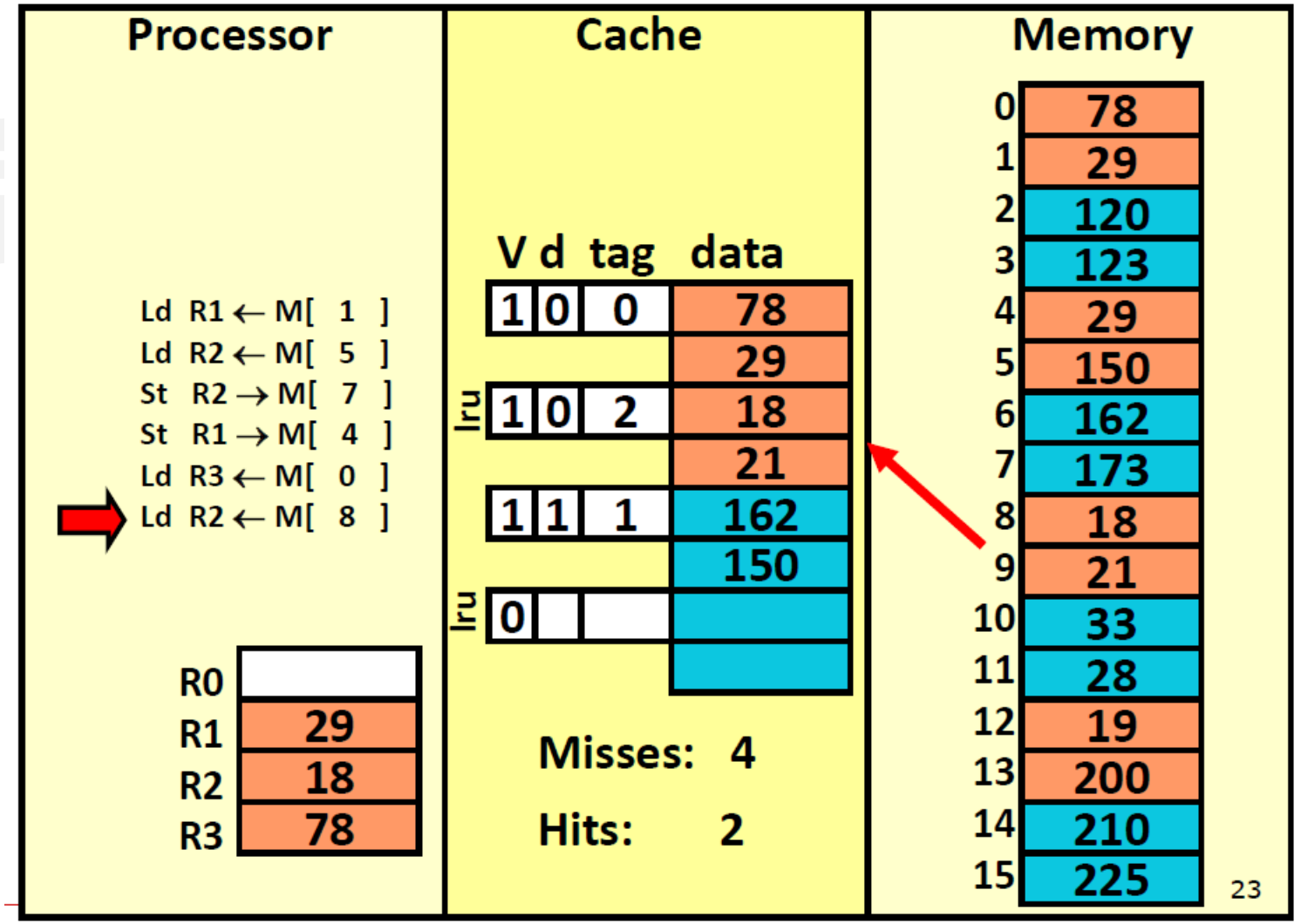
2-Way Set Associative Cache实例

- 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

下一次课 – 第8次课

- 缓存一致性与虚拟内存技术

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