北京大学
PEKING UNIVERSITY

BOOM LSU

BOOMv3 之 访存单元（The Load/Store Unit)

梁书豪

目录

北京大学
PEKING UNIVERSITY

1.
LSU简介

关键词 / 涉及指令

LSU简介

北京大学 PEKING UNIVERSITY

LSU关键词

- **全称：**The Load/Store Unit，访存单元
- **作用：**处理访存相关的指令/微指令
- **约束：**按序执行
  - 对于流水线：访存数据符合程序顺序
  - 对于内存系统：外部可见的操作符合程序顺序（非严格按序，详见指令手册）
- **目标：**最大化吞吐量
  - 尽可能减少miss
  - 尽可能使指令的执行只受数据依赖限制

北京大学 PEKING UNIVERSITY

LSU涉及的指令/微指令

分类	指令（Instruction）	微指令（uop）
整点	L (DW, W, WU, H, HU, B, BU) S (DW, W, H, B)	uopLD uopSTA/uopSTD
浮点	FL (D, W) FS (D, W)	uopLD uopSTA
障碍	FENCE SFENCE.VMA	uopFENCE uopSFENCE
原子	AMO • (ADD, XOR, SWAP, AND, OR, MIN, MINU, MAX, MAXU) • (D, W)	uopAMO_AG
同步	LR (D, W) SC (D, W)	uopLD uopAMO_AG

重点讲这部分

其他的原理相同，都是Load/Store的变体，只是需要清空流水线、需要保存额外信息等

北京大学
PEKING UNIVERSITY

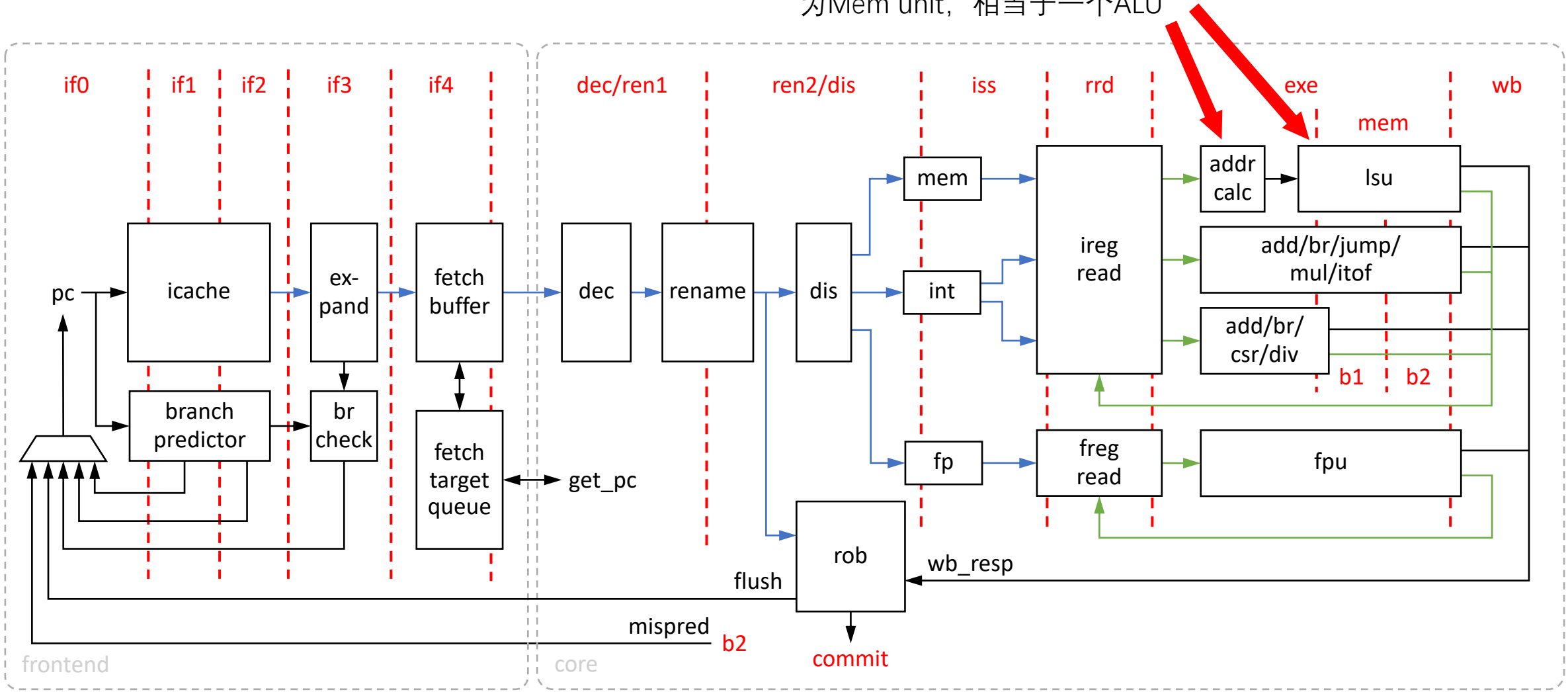
2.
LSU与其他模块的交互

指派 / 发射 / 写回 / 唤醒 / 异常 / 转移预测错误 / 提交

LSU与其他模块的交互

LSU在流水线中的位置

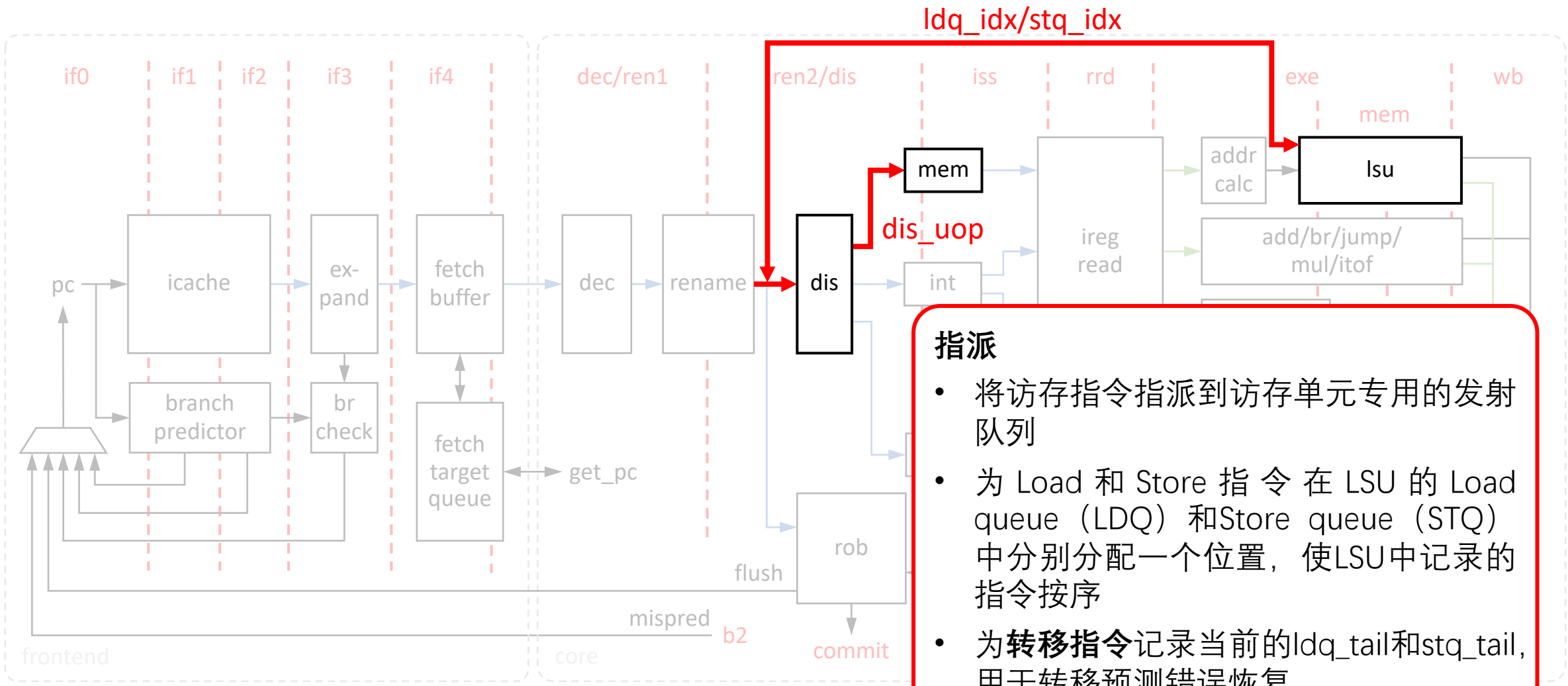
Addr calc unit和LSU的组合又称为Mem unit，相当于一个ALU



instr　data　others

* 流水线配置以MediumBoom为例

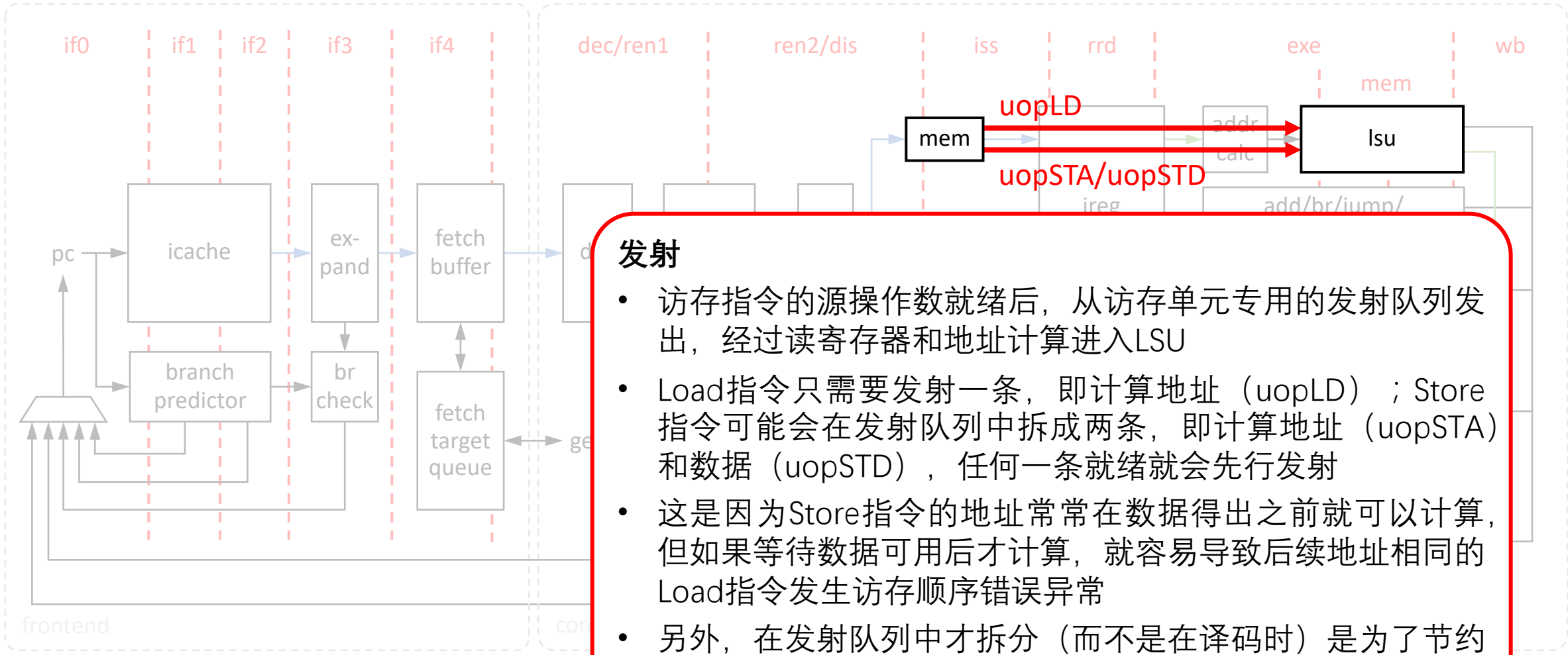
LSU与其他模块的交互



指派

- 将访存指令指派到访存单元专用的发射队列
- 为 Load 和 Store 指令在 LSU 的 Load queue（LDQ）和Store queue（STQ）中分别分配一个位置，使LSU中记录的指令按序
- 为**转移指令**记录当前的ldq_tail和stq_tail，用于转移预测错误恢复

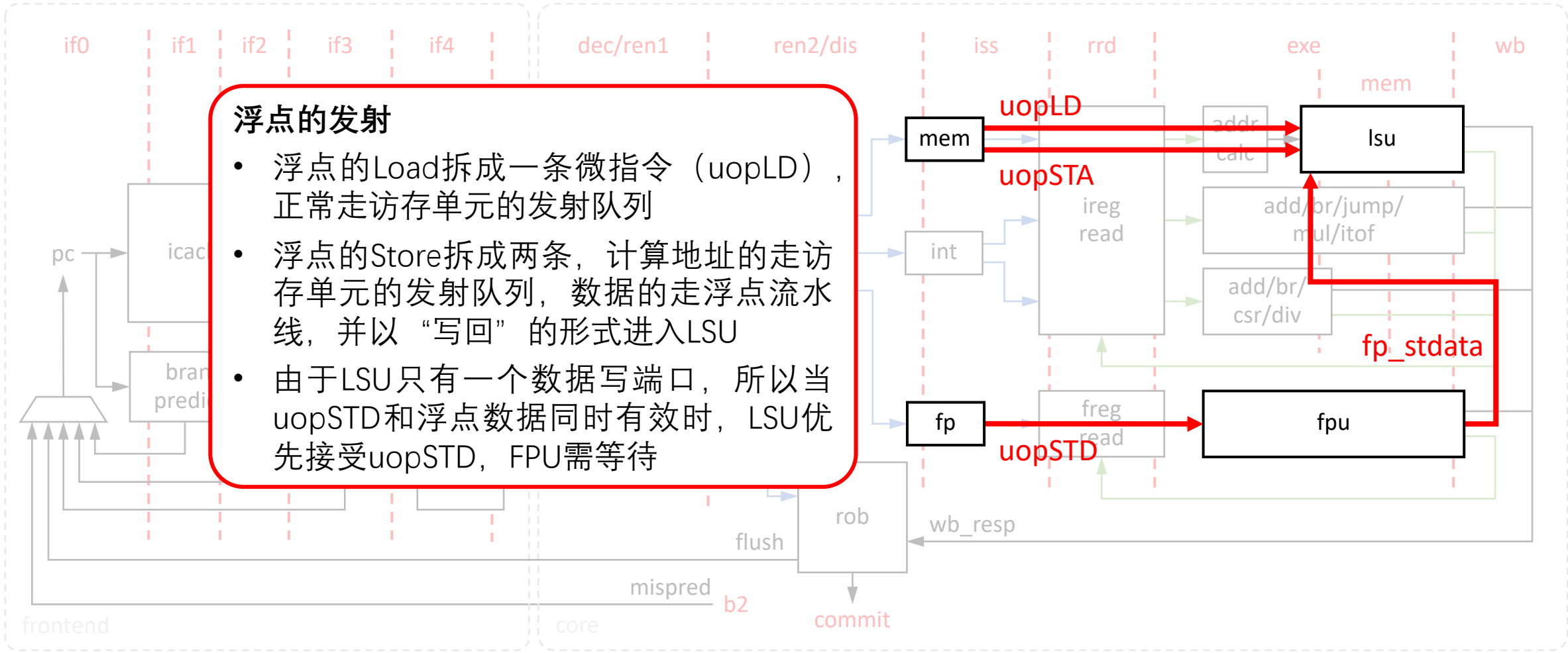
LSU与其他模块的交互



发射

- 访存指令的源操作数就绪后，从访存单元专用的发射队列发出，经过读寄存器和地址计算进入LSU
- Load指令只需要发射一条，即计算地址（uopLD）；Store指令可能会在发射队列中拆成两条，即计算地址（uopSTA）和数据（uopSTD），任何一条就绪就会先行发射
- 这是因为Store指令的地址常常在数据得出之前就可以计算，但如果等待数据可用后才计算，就容易导致后续地址相同的Load指令发生访存顺序错误异常
- 另外，在发射队列中才拆分（而不是在译码时）是为了节约发射队列宝贵的空间

LSU与其他模块的交互

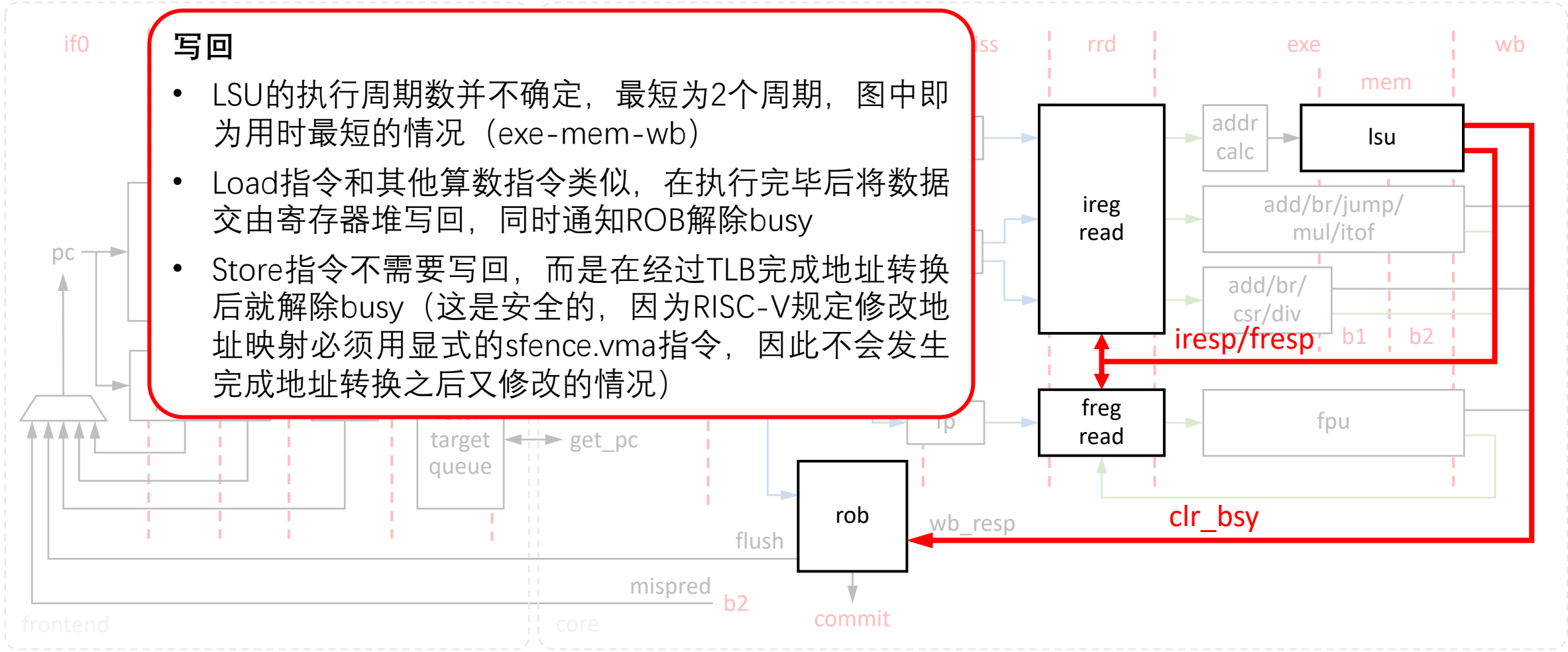


浮点的发射

- 浮点的Load拆成一条微指令（uopLD），正常走访存单元的发射队列
- 浮点的Store拆成两条，计算地址的走访存单元的发射队列，数据的走浮点流水线，并以“写回”的形式进入LSU
- 由于LSU只有一个数据写端口，所以当uopSTD和浮点数据同时有效时，LSU优先接受uopSTD，FPU需等待

instr　data　others

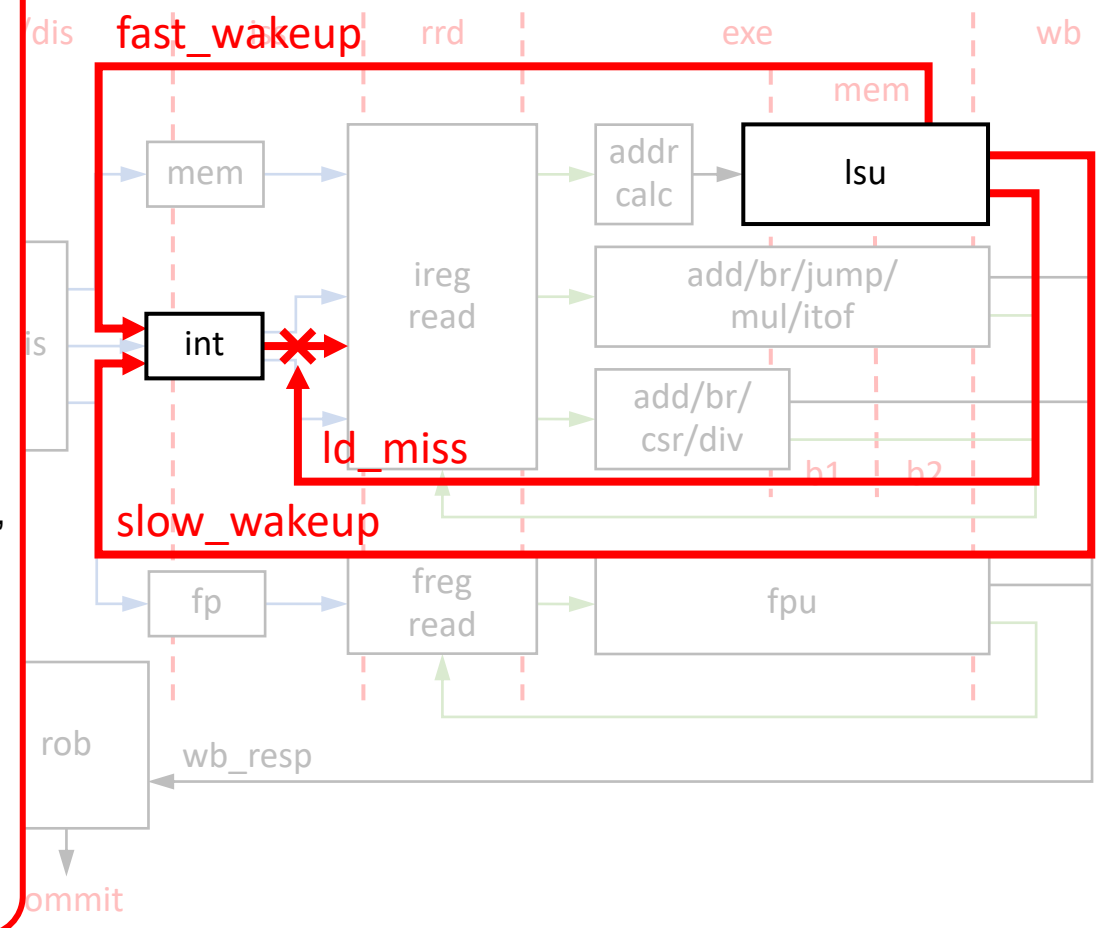
LSU与其他模块的交互



LSU与其他模块的交互

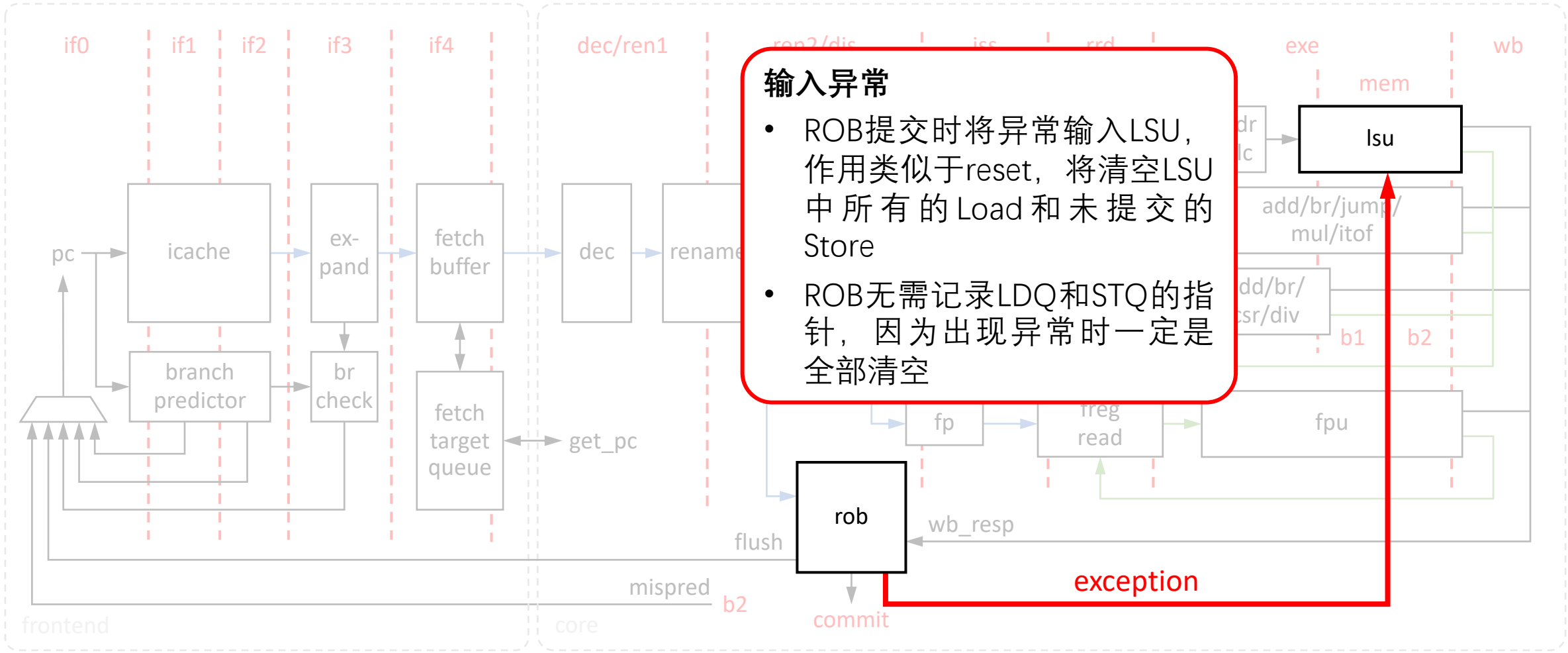
唤醒

- BOOM提供可配置的针对依赖Load的指令的快速唤醒机制（enableFastLoadUse）
- 当不启用该机制时，Load指令在完成时（wb阶段）才唤醒依赖自己的指令
- 启用该机制时，Load在完成前一周期（mem阶段）就推测式唤醒后续指令，节约了一周期；但若推测错误（即DCache未命中），则需要取消被唤醒的指令；发射队列需保留推测唤醒的指令（poisoned），在推测解决后才能清除
- 快速唤醒不涉及数据前递，因为Load到达wb时，被唤醒指令还在iss阶段，到rrd阶段时数据已经写回
- Alpha21264也有该机制，但它更为激进，提前2个周期就唤醒，导致可能会错误唤醒更多的指令，所以取消唤醒的代价更大



instr　data　others

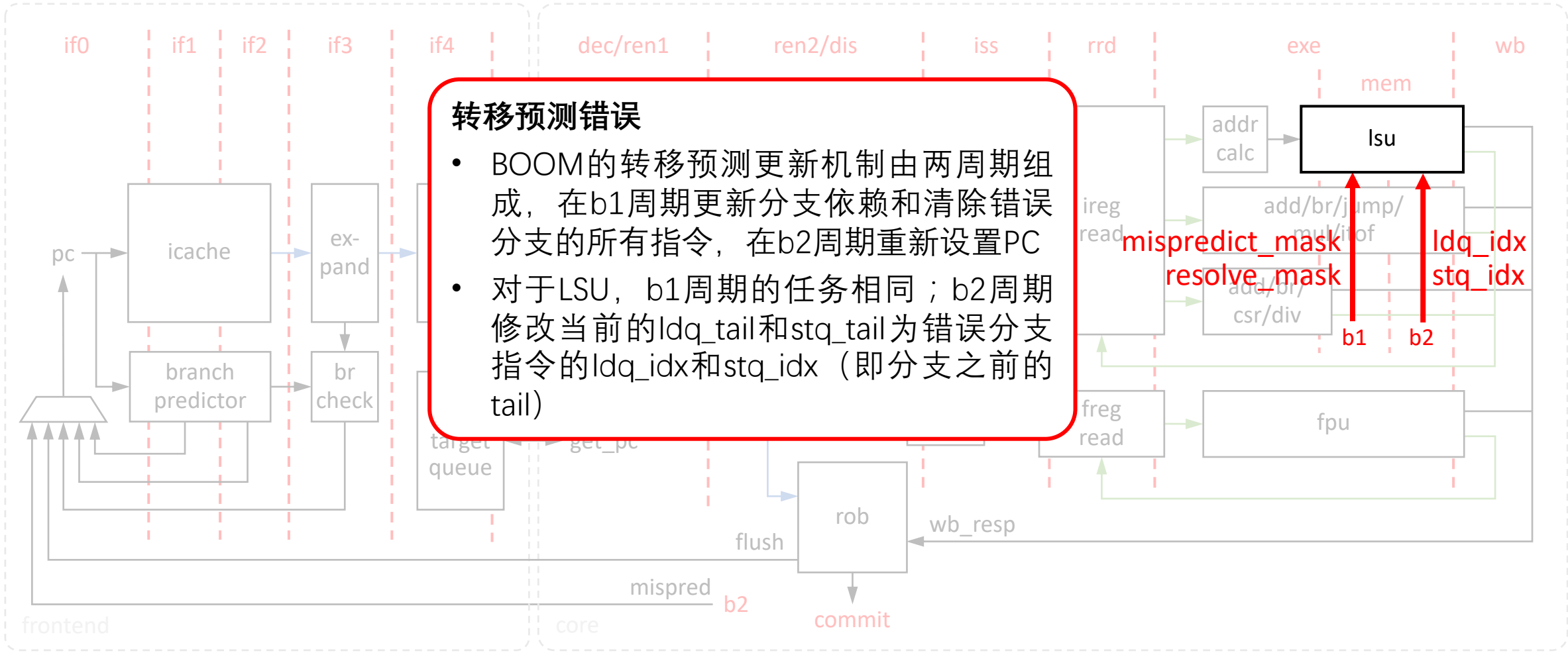
LSU与其他模块的交互



输入异常

- ROB提交时将异常输入LSU，作用类似于reset，将清空LSU中所有的Load和未提交的Store
- ROB无需记录LDQ和STQ的指针，因为出现异常时一定是全部清空

LSU与其他模块的交互

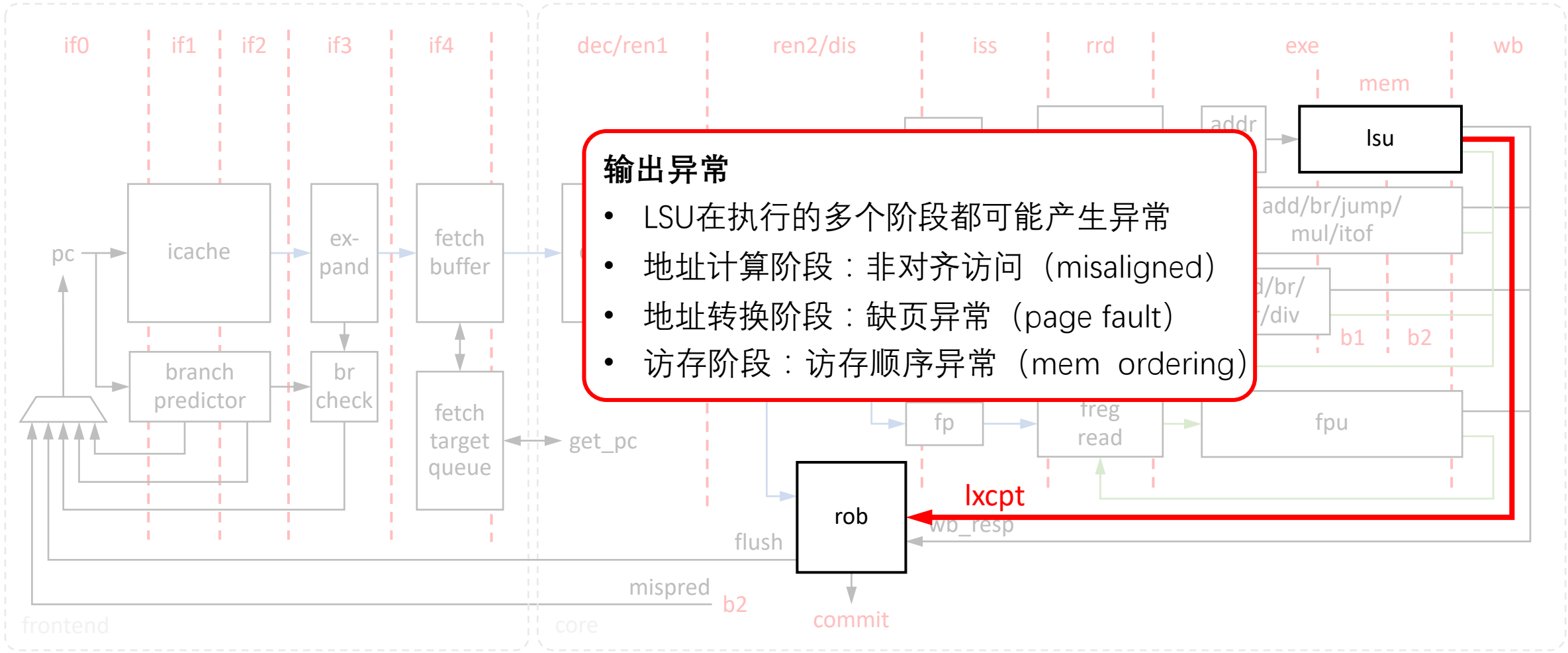


转移预测错误

- BOOM的转移预测更新机制由两周期组成，在b1周期更新分支依赖和清除错误分支的所有指令，在b2周期重新设置PC
- 对于LSU，b1周期的任务相同；b2周期修改当前的ldq_tail和stq_tail为错误分支指令的ldq_idx和stq_idx（即分支之前的tail）

instr　data　others

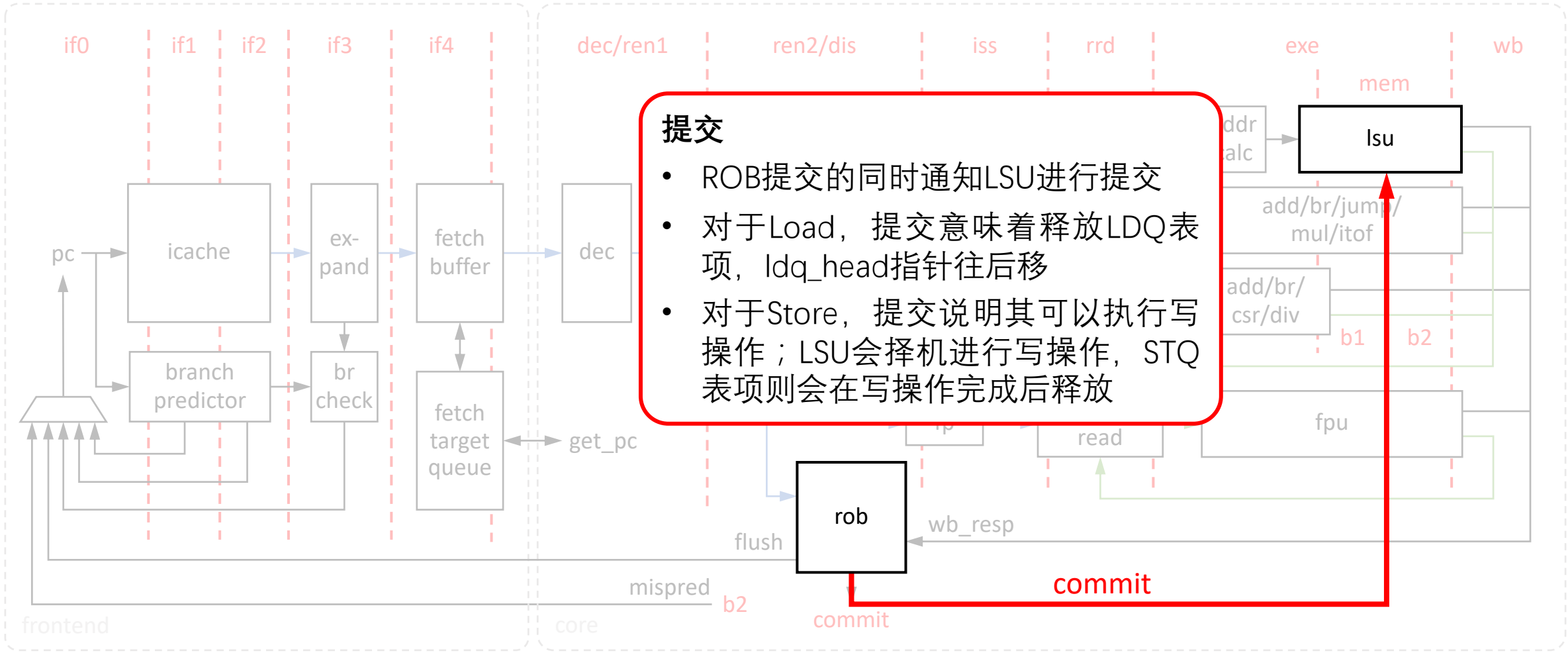
LSU与其他模块的交互



输出异常

- LSU在执行的多个阶段都可能产生异常
- 地址计算阶段：非对齐访问（misaligned）
- 地址转换阶段：缺页异常（page fault）
- 访存阶段：访存顺序异常（mem ordering）

LSU与其他模块的交互



提交

- ROB提交的同时通知LSU进行提交
- 对于Load，提交意味着释放LDQ表项，ldq_head指针往后移
- 对于Store，提交说明其可以执行写操作；LSU会择机进行写操作，STQ表项则会在写操作完成后释放

北京大学
PEKING UNIVERSITY

3.
LSU的内部结构

LSU流水线 / DCache / MSHR

LSU流水线简介

理解LSU流水线的关键——

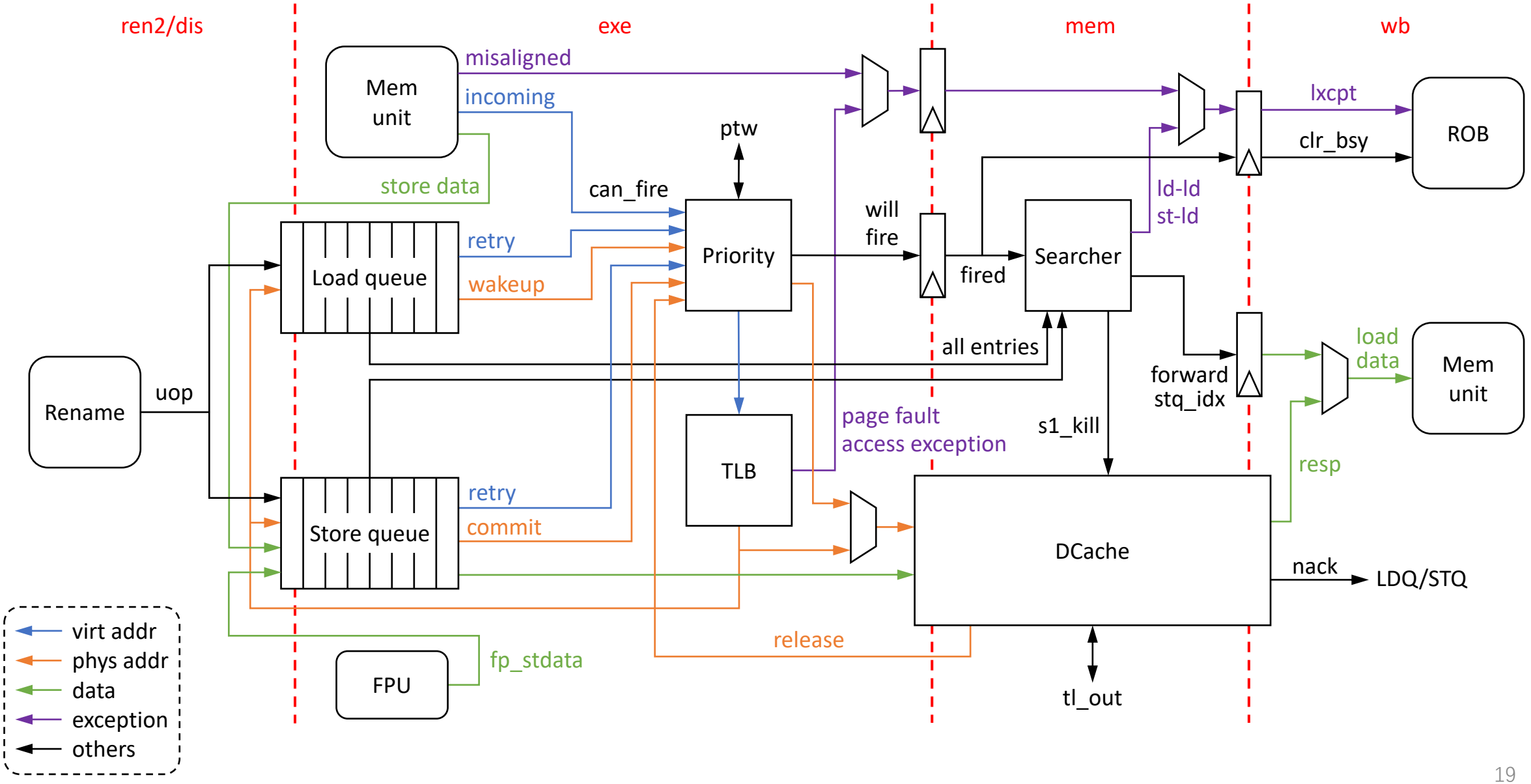
LSU是一个完整的超标量流水线！

- LSU可以自主选择每个周期执行的请求
- LSU可以自行从TLB和DCache miss中恢复
- LSU可以自动发现访存顺序异常

LSU可以做到

- 一定保证访存请求得到响应
- 一定保证访存顺序（否则引发异常）

LSU流水线


ren2/dis
exe
mem
wb
Mem unit
misaligned
incoming
store data
ptw
can_fire
Priority
will fire
fired
Searcher
lxcpt
clr_bsy
ROB
ld-ld
st-ld
Load queue
retry
wakeup
Rename
uop
all entries
forward stq_idx
load data
Mem unit
page fault
access exception
TLB
s1_kill
resp
Store queue
retry
commit
DCache
nack
LDQ/STQ
release
FPU
fp_stdata
tl_out
virt addr
phys addr
data
exception
others

北京大学 PEKING UNIVERSITY

LDQ/STQ表项

Load queue (LDQ)

valid	address	virtual	exec	succ	ord	obs	st_mask	fwd_idx
Bool	*UInt*	*Bool*	*Bool*	*Bool*	*Bool*	*Bool*	*UInt*	*UInt*

Store queue (STQ)

valid	address	virtual	data	commit	succ
Bool	*UInt*	*Bool*	*UInt*	*Bool*	*Bool*

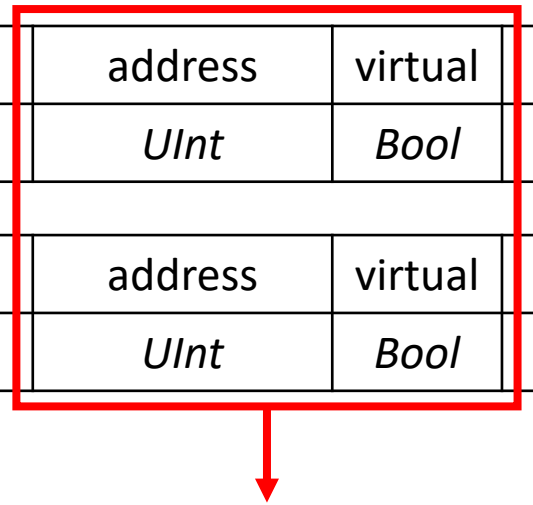
LDQ/STQ表项

Load queue (LDQ)

valid	address	virtual	exec	succ	ord	obs	st_mask	fwd_idx
Bool	*UInt*	*Bool*	*Bool*	*Bool*	*Bool*	*Bool*	*UInt*	*UInt*

Store queue (STQ)

valid	address	virtual	data	commit	succ
Bool	*UInt*	*Bool*	*UInt*	*Bool*	*Bool*



访存地址相关

- LDQ/STQ中的地址可以是物理地址，也可以是虚拟地址，取决于是否已经进行了地址转换
- 该设计相当于复用了address域，只是用一个virtual位区分是否是虚拟地址

LSU流水线

LDQ/STQ表项

Load queue (LDQ)

valid	address	virtual	exec	succ	ord	obs	st_mask	fwd_idx
Bool	*UInt*	*Bool*	*Bool*	*Bool*	*Bool*	*Bool*	*UInt*	*UInt*

Store queue (STQ)

valid	address	virtual	data	commit	succ
Bool	*UInt*	*Bool*	*UInt*	*Bool*	*Bool*

执行相关

- 由于LSU是一个完整的流水线，它需要一些位置来记录指令在流水线中的状态
- exec：Load已送入DCache，尚未得到结果
- commit：Store已提交，可以写入DCache
- succ：对DCache的访问已经命中

北京大学 PEKING UNIVERSITY

LDQ/STQ表项

Load queue (LDQ)

valid	address	virtual	exec	succ	ord	obs	st_mask	fwd_idx
Bool	*UInt*	*Bool*	*Bool*	*Bool*	*Bool*	*Bool*	*UInt*	*UInt*

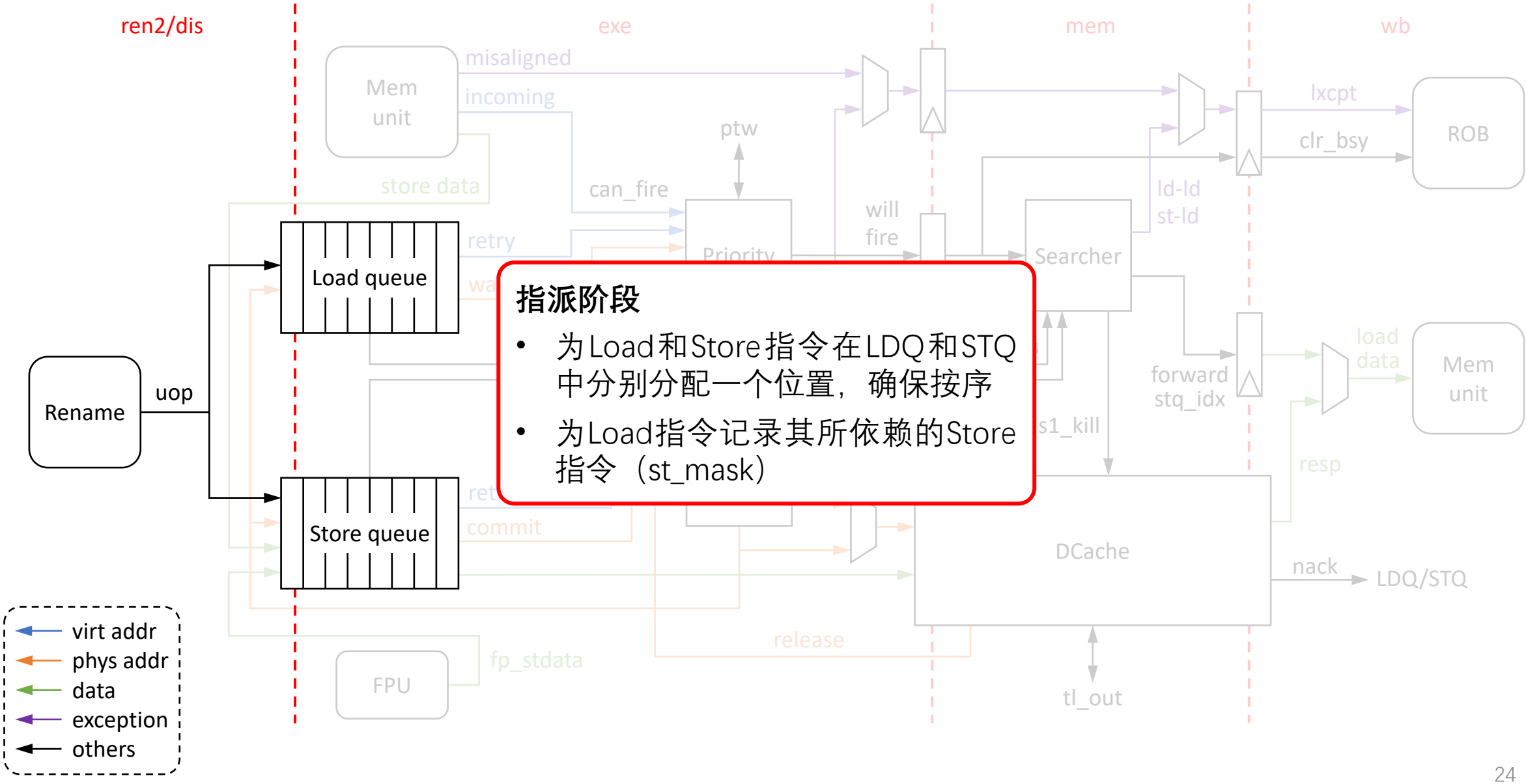
Store queue (STQ)

valid	address	virtual	data	commit	succ
Bool	*UInt*	*Bool*	*UInt*	*Bool*	*Bool*

访存顺序相关

- st_mask：当前Load依赖的所有Store；用于判断前递和st-ld异常
- fwd_idx：当前Load使用了该下标的Store前递的数据；用于判断st-ld异常
- obs：当前Load的数据已释放；用于判断ld-ld异常
- 若出现上述任一异常，设置ord为真

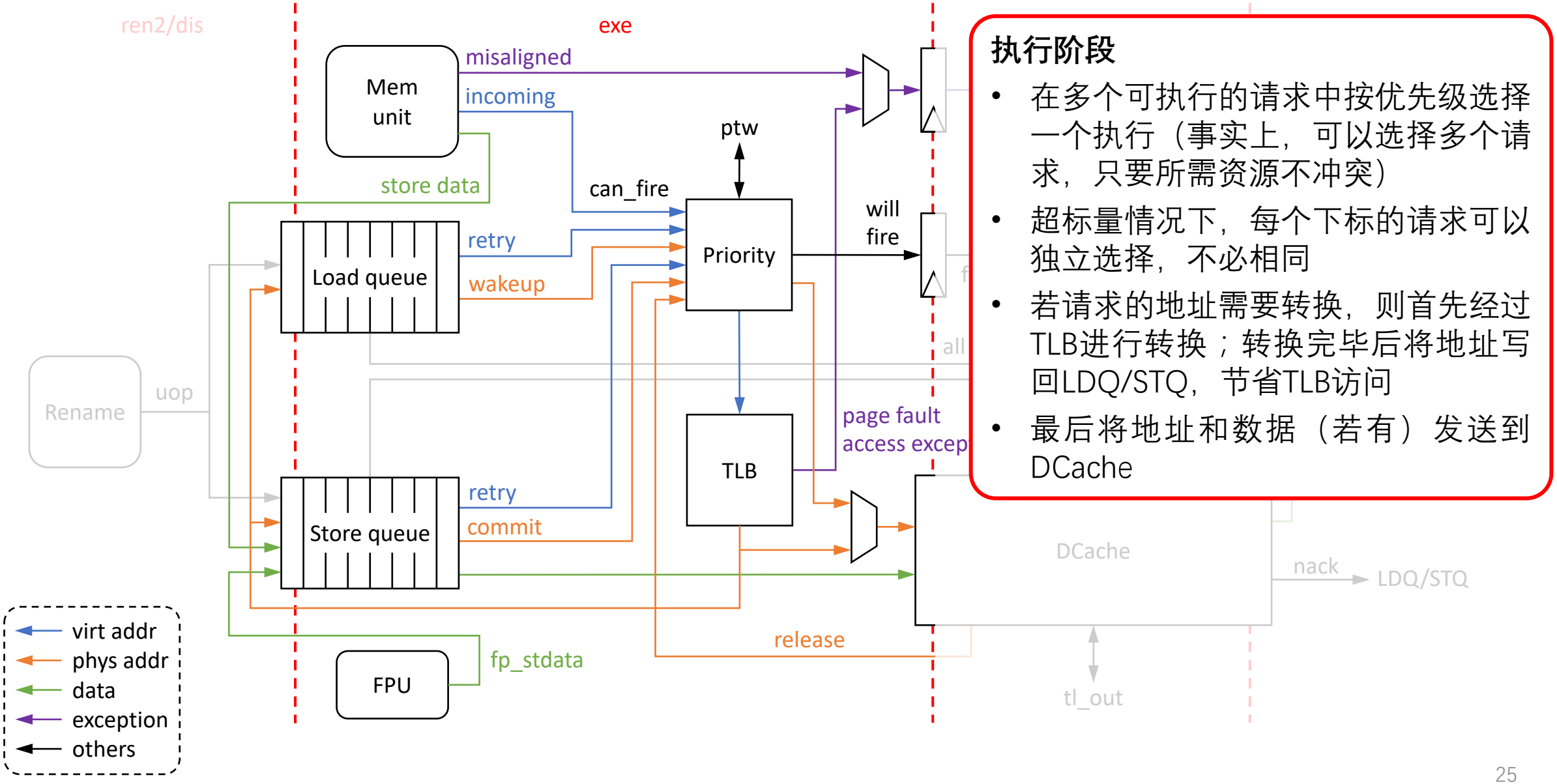
LSU流水线



指派阶段

- 为Load和Store指令在LDQ和STQ中分别分配一个位置，确保按序
- 为Load指令记录其所依赖的Store指令（st_mask）

LSU流水线



执行阶段

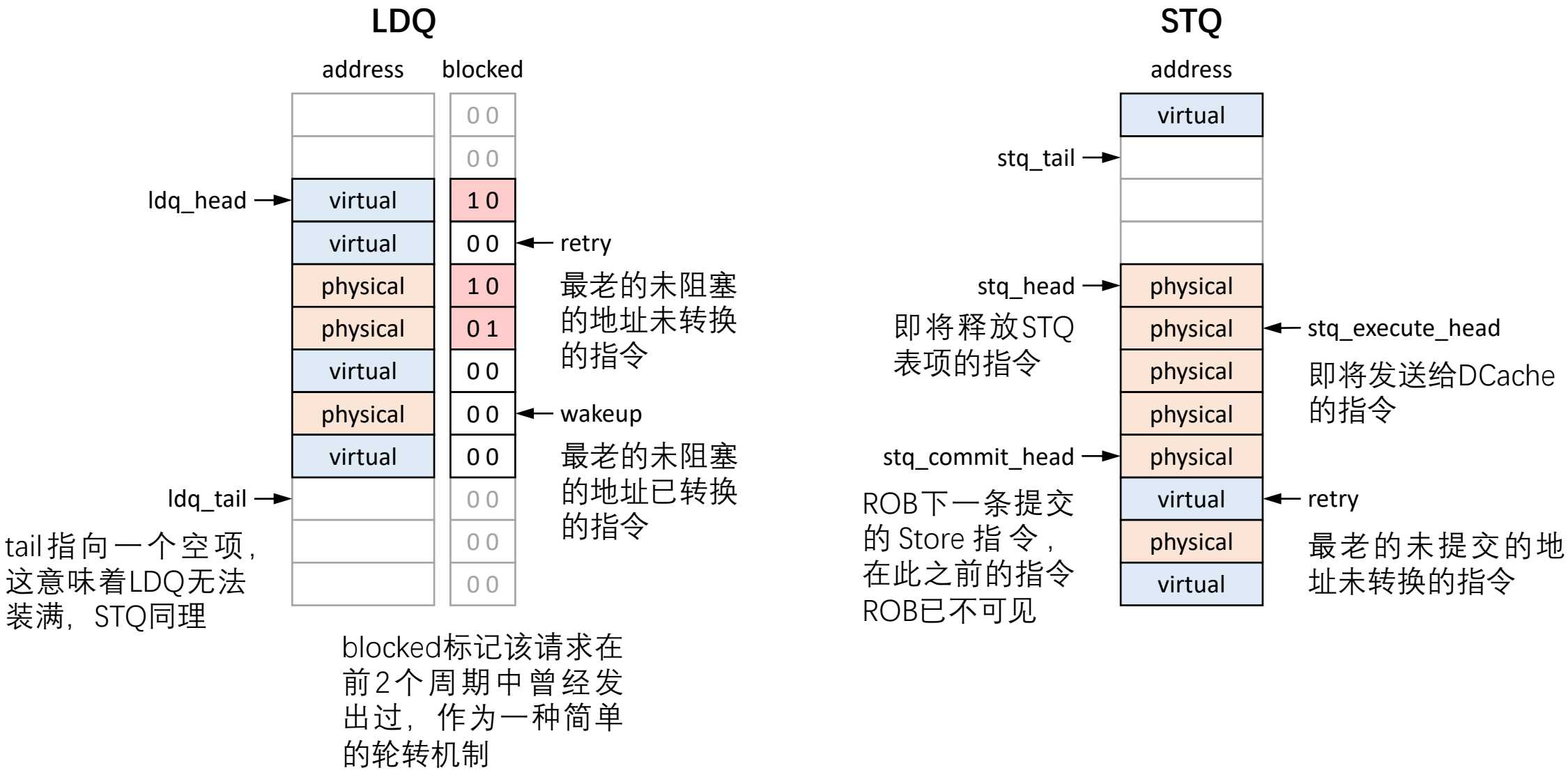
- 在多个可执行的请求中按优先级选择一个执行（事实上，可以选择多个请求，只要所需资源不冲突）
- 超标量情况下，每个下标的请求可以独立选择，不必相同
- 若请求的地址需要转换，则首先经过TLB进行转换；转换完毕后将地址写回LDQ/STQ，节省TLB访问
- 最后将地址和数据（若有）发送到DCache

LSU中的请求（按优先级排列）

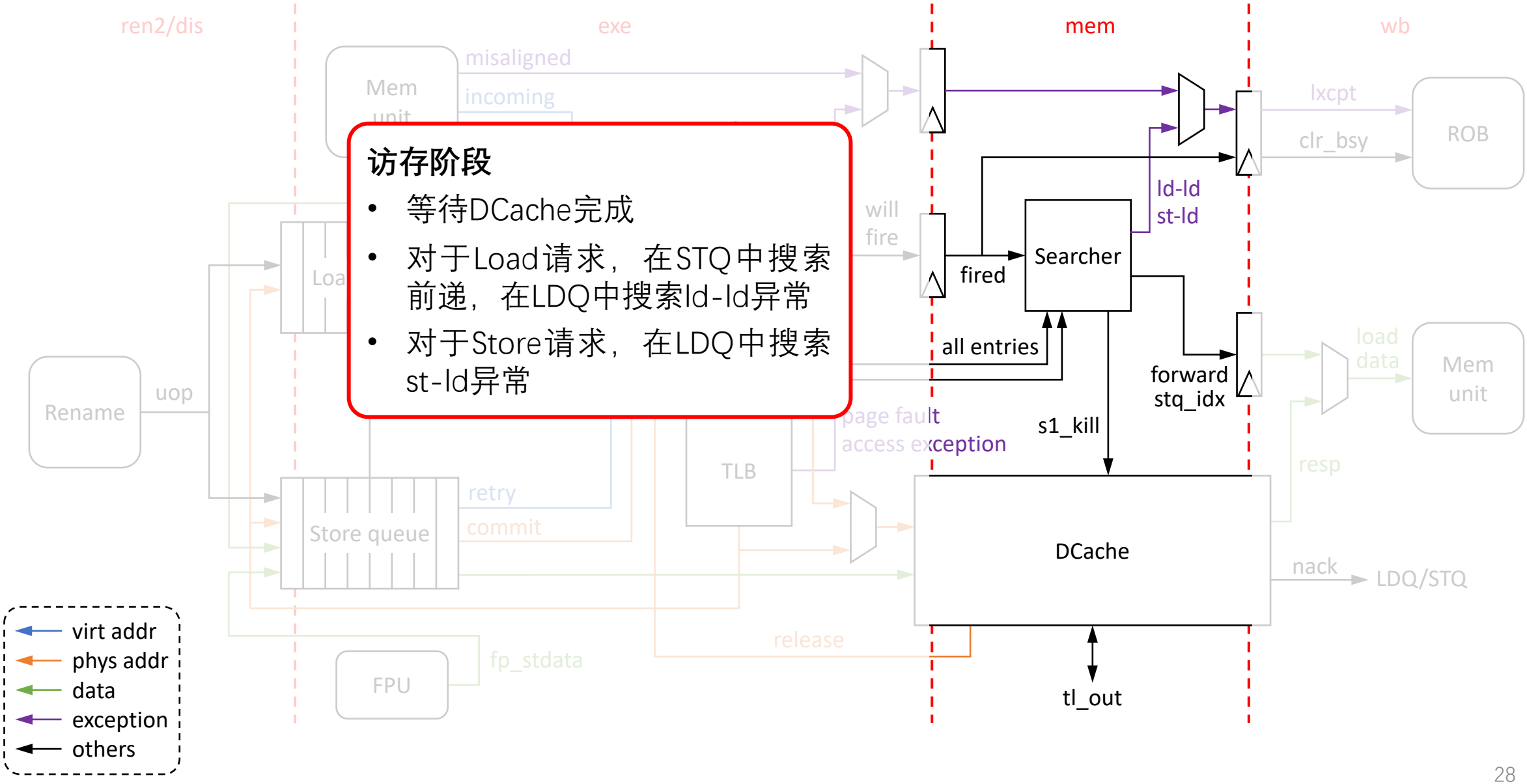
请求	说明	下标
load_incoming	从MemUnit过来的优先级最高，因为不能对MemUnit阻塞；这4条指令的优先级实际上没有区别，因为每个MemUnit一周期只能到其中一条	与所在MemUnit相同
stad_incoming		
sta_incoming		
std_incoming		
sfence	障碍指令，同步TLB	不需要
release	DCache释放了一个块，故下次对这个块的Load会拿到新的内容，可能出现ld-ld异常；LDQ需标记obs位	固定为memWidth-1
hella_incoming	Hella是Rocket的DCache使用的协议，BOOM为了兼容顺序执行的RoCC提供了这个接口	
hella_wakeup		
load_retry	尚未计算出物理地址，需要先经过TLB	
sta_retry		
load_wakeup	已经计算出物理地址，只需访问DCache	
store_commit		固定为0

LSU流水线

关于retry/wakeup/commit



LSU流水线

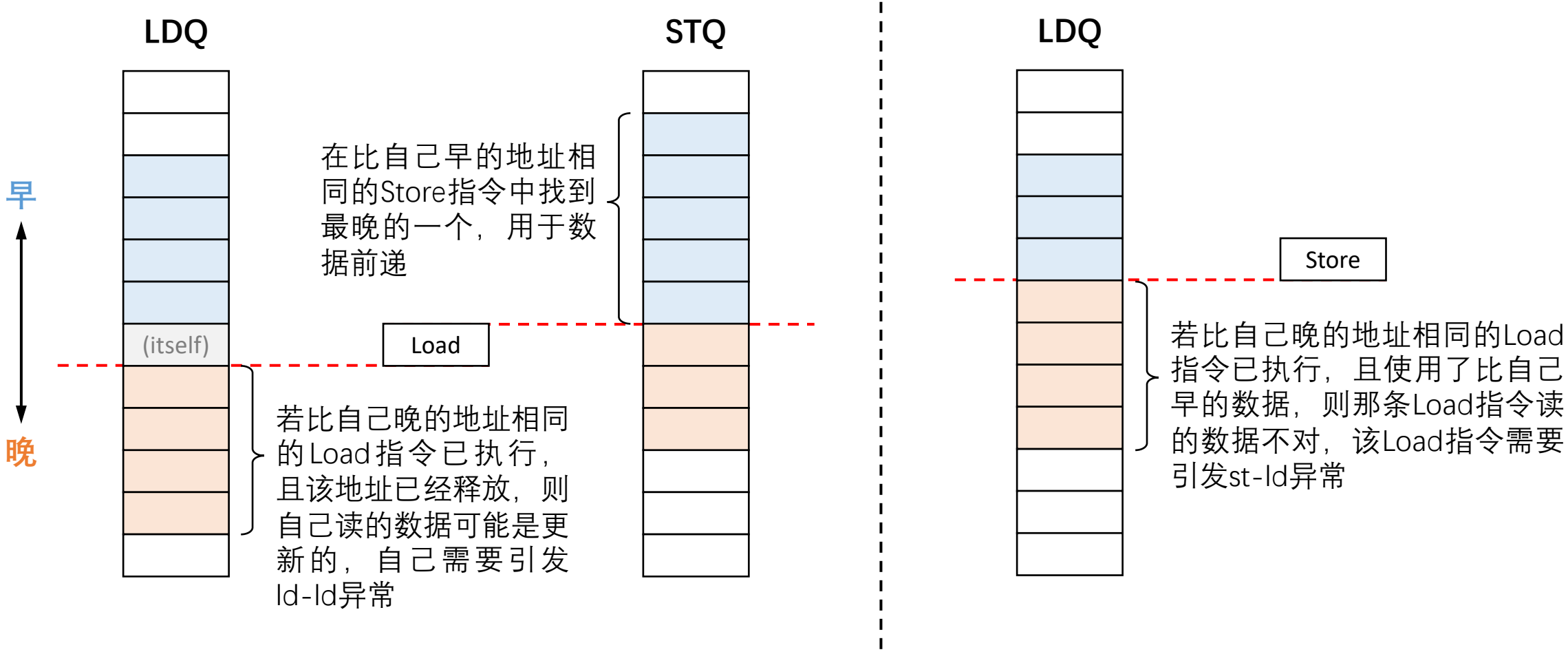


访存阶段

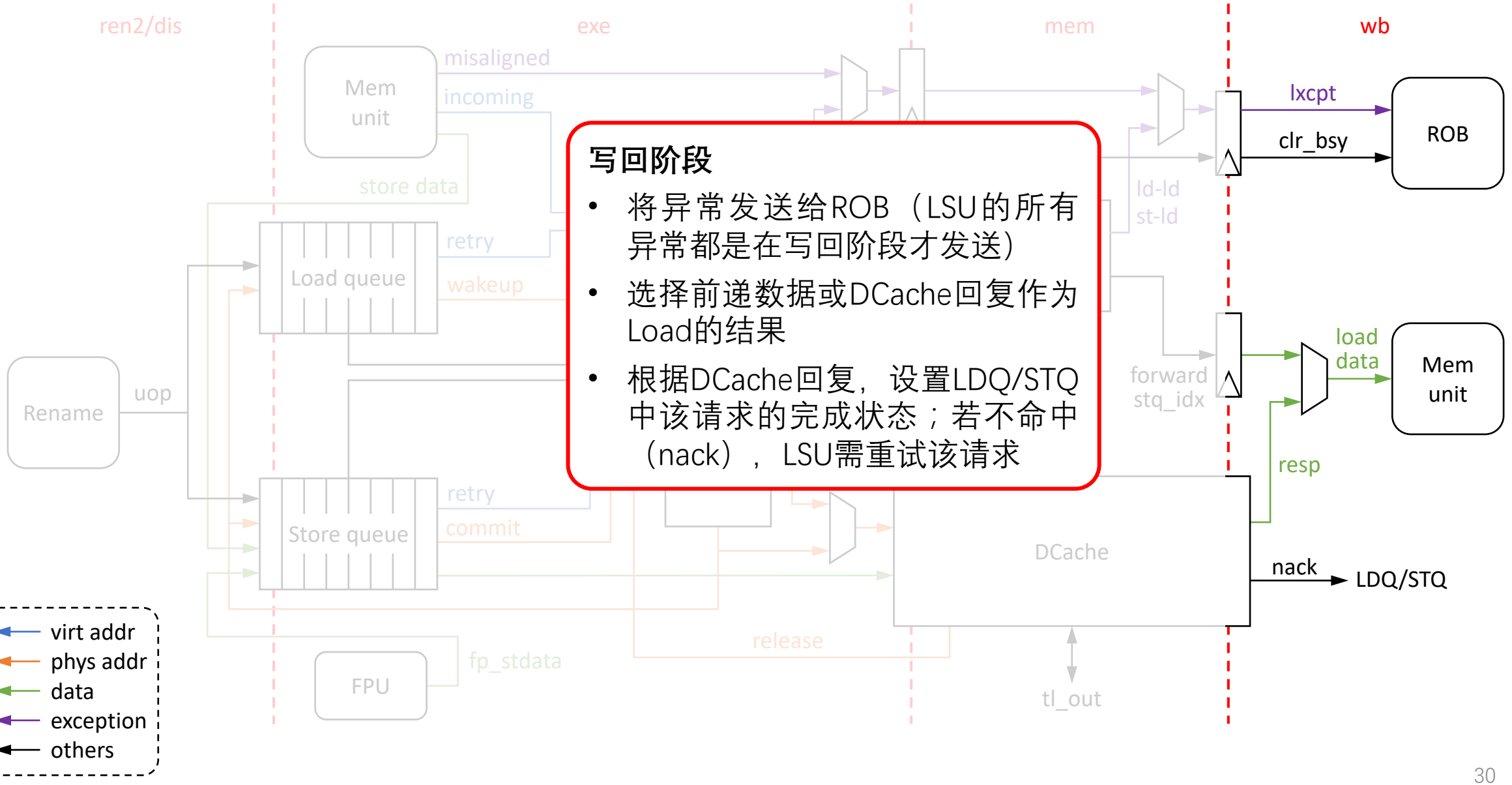
- 等待DCache完成
- 对于Load请求，在STQ中搜索前递，在LDQ中搜索ld-ld异常
- 对于Store请求，在LDQ中搜索st-ld异常

LSU流水线

Searcher逻辑



LSU流水线

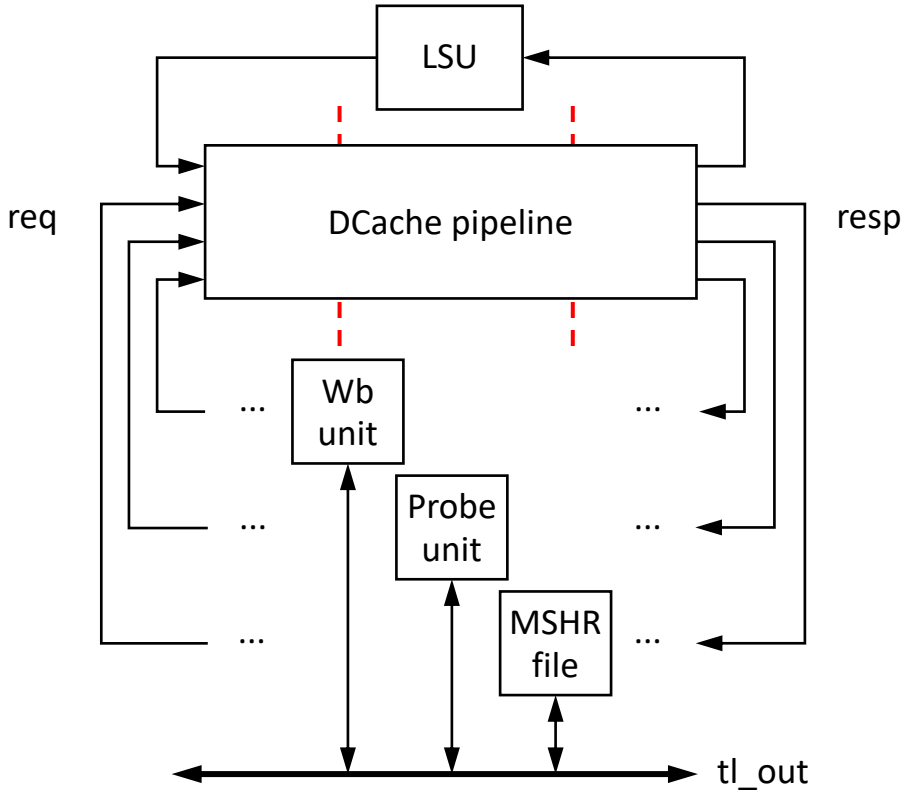


写回阶段

- 将异常发送给ROB（LSU的所有异常都是在写回阶段才发送）
- 选择前递数据或DCache回复作为Load的结果
- 根据DCache回复，设置LDQ/STQ中该请求的完成状态；若不命中（nack），LSU需重试该请求

DCache简介

- **作用：**用于存放缓存数据，可根据输入的物理地址对数据进行访问、修改
- **特点：**是非阻塞（Non-blocking）的，可以同时处理多个miss

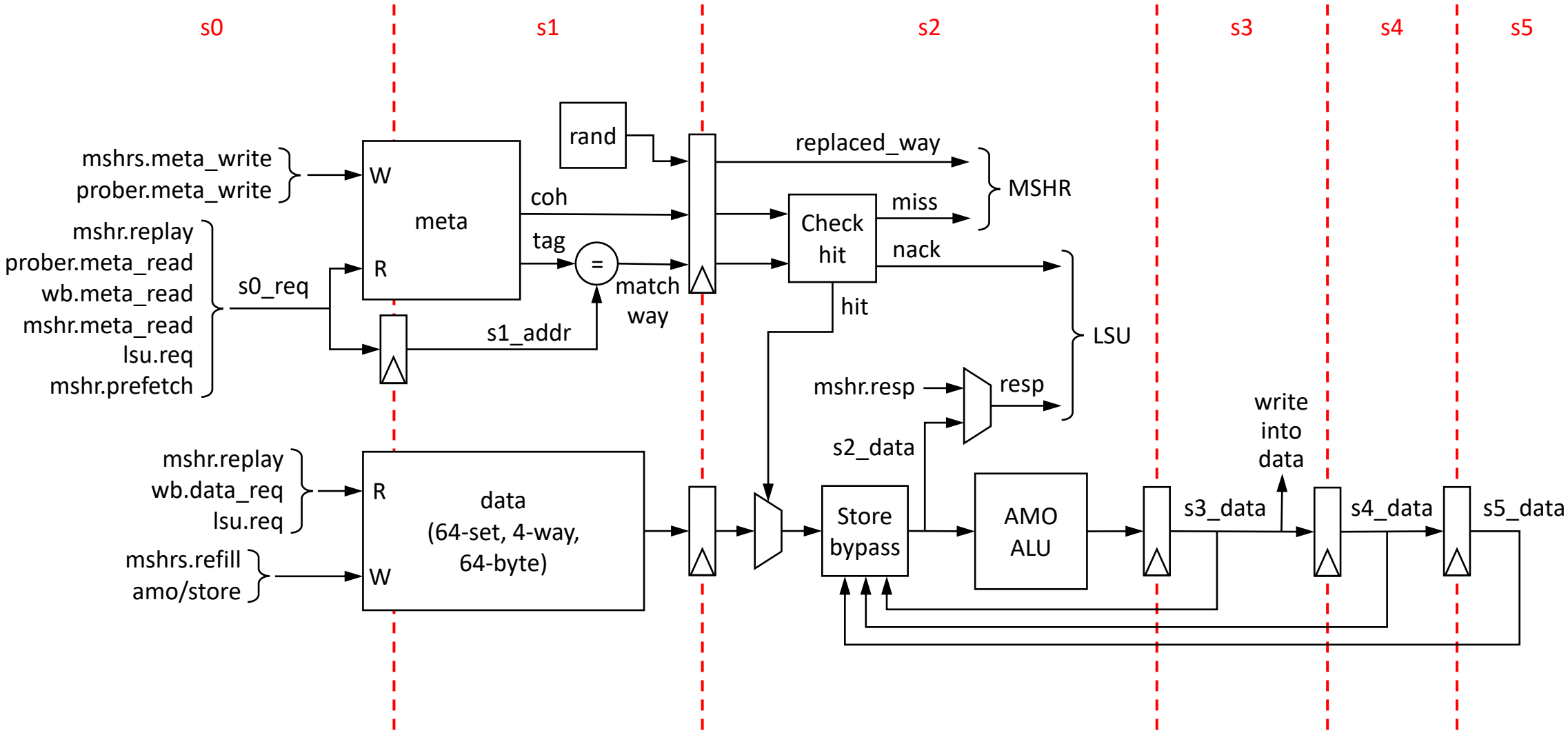


DCache的子模块

- DCache本身并不会处理miss、保证cache一致性和访问下级存储等，它只负责存放数据，其他都交由子模块进行

子模块	作用
Wb unit	处理另外两个模块的写回请求
Probe unit	处理下级存储传来的探测请求
MSHR file	处理LSU产生的miss请求（详见后文）

DCache流水线

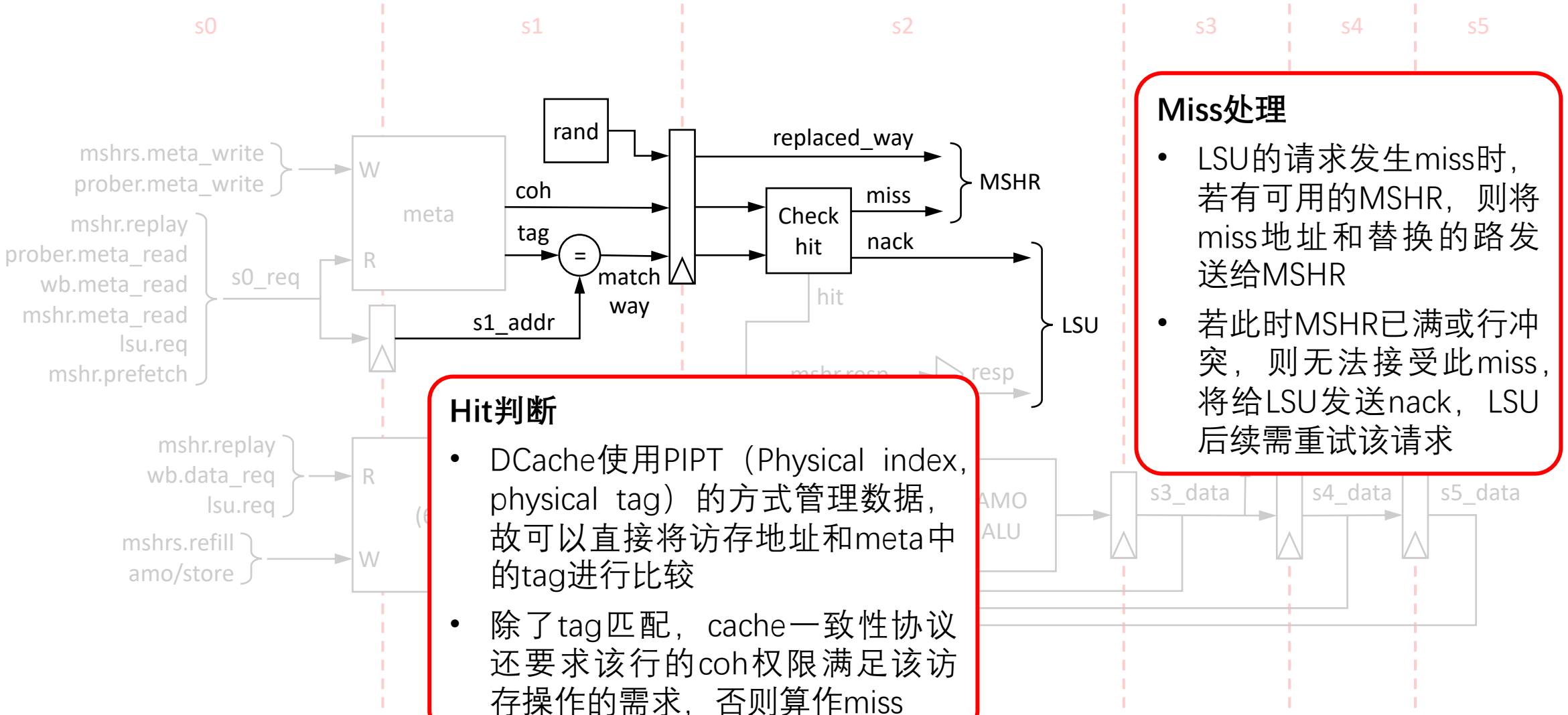

s0
s1
s2
s3
s4
s5
mshrs.meta_write
prober.meta_write
mshr.replay
prober.meta_read
wb.meta_read
mshr.meta_read
lsu.req
mshr.prefetch
s0_req
W
R
meta
coh
tag
=
match
way
s1_addr
rand
replaced_way
miss
nack
MSHR
Check
hit
hit
LSU
mshr.resp
resp
s2_data
write
into
data
mshr.replay
wb.data_req
lsu.req
mshrs.refill
amo/store
R
W
data
(64-set, 4-way,
64-byte)
Store
bypass
AMO
ALU
s3_data
s4_data
s5_data

DCache流水线

请求

- DCache可能同时收到多个模块的请求，此时它会用图示优先级选择其中一个
- 虽然DCache也是超标量的，但它每个周期只能处理**一种**请求
- 事实上，只有LSU的请求可以利用其超标量的能力，其他请求都是单标量的

DCache流水线



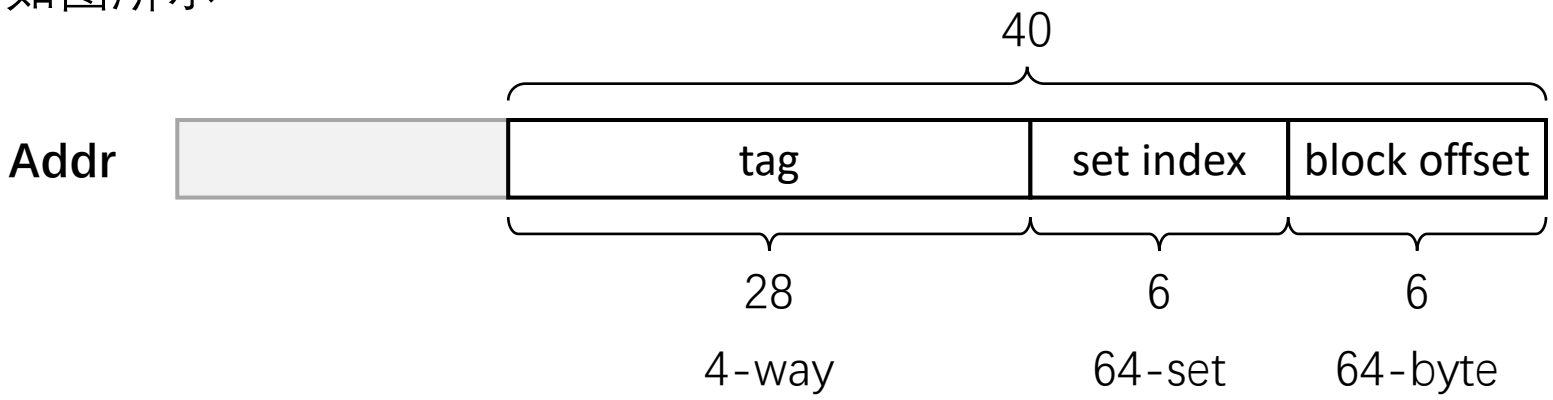
Hit判断

- DCache使用PIPT（Physical index, physical tag）的方式管理数据，故可以直接将访存地址和meta中的tag进行比较
- 除了tag匹配，cache一致性协议还要求该行的coh权限满足该访存操作的需求，否则算作miss

Miss处理

- LSU的请求发生miss时，若有可用的MSHR，则将miss地址和替换的路发送给MSHR
- 若此时MSHR已满或行冲突，则无法接受此miss，将给LSU发送nack，LSU后续需重试该请求

- **tag：**如图所示



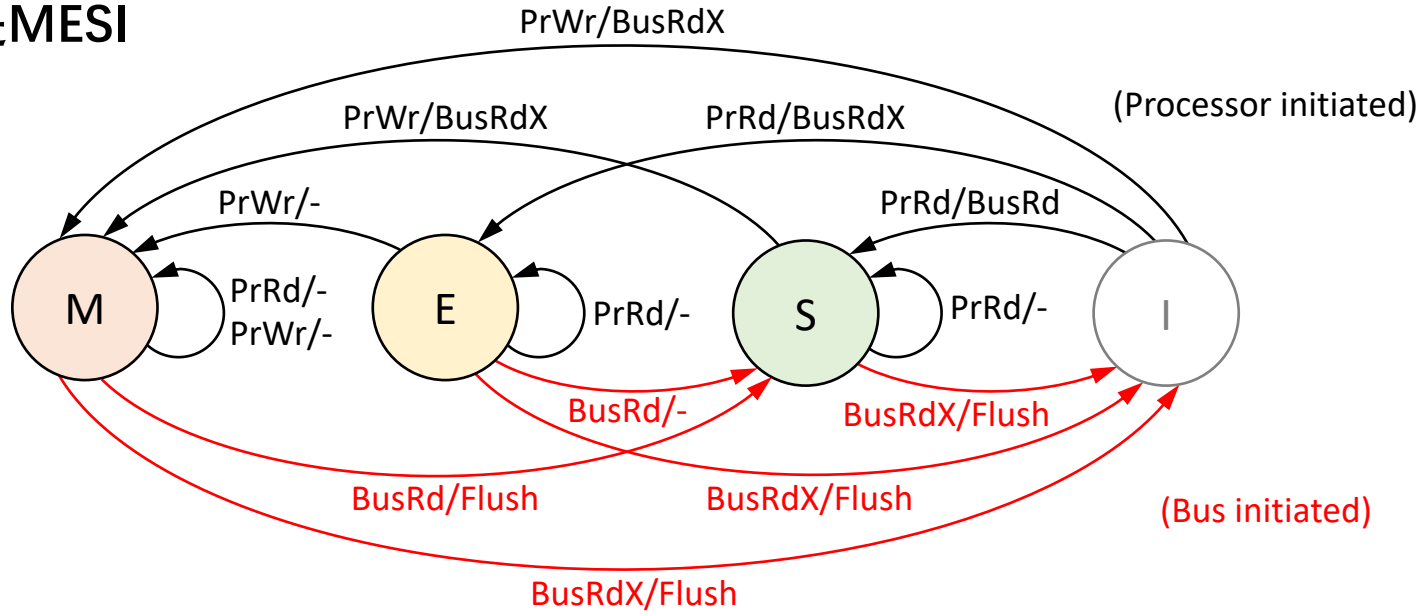
- **coh：**表明该行数据的状态

状态	含义	变化
Nothing	数据无效	读/写操作都需要重新请求数据和升级权限
Branch	只读	读操作不变，写操作（wr/wi）需请求升级权限
Trunk	读写，但尚未写过	读操作不变，写操作（wr）需变为Dirty
Dirty	读写，且已经写过	读/写操作均不变

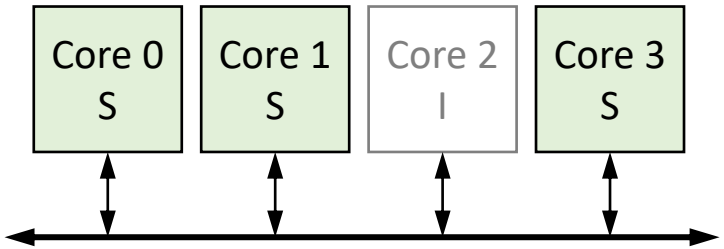
  - 注：此处的读写权限和TLB中的页权限无关，这个是为了内存一致性设置的状态

- **TileLink使用的一致性协议其实就是MESI**

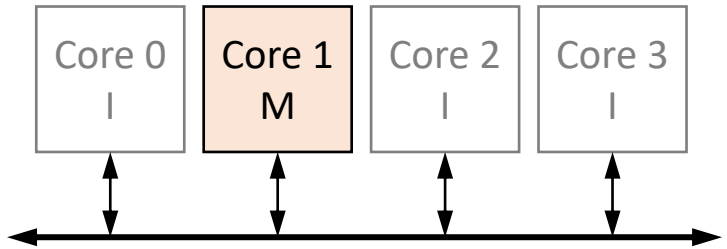
M	Modified
E	Exclusive
S	Shared
I	Invalid



- **简单理解MESI：**

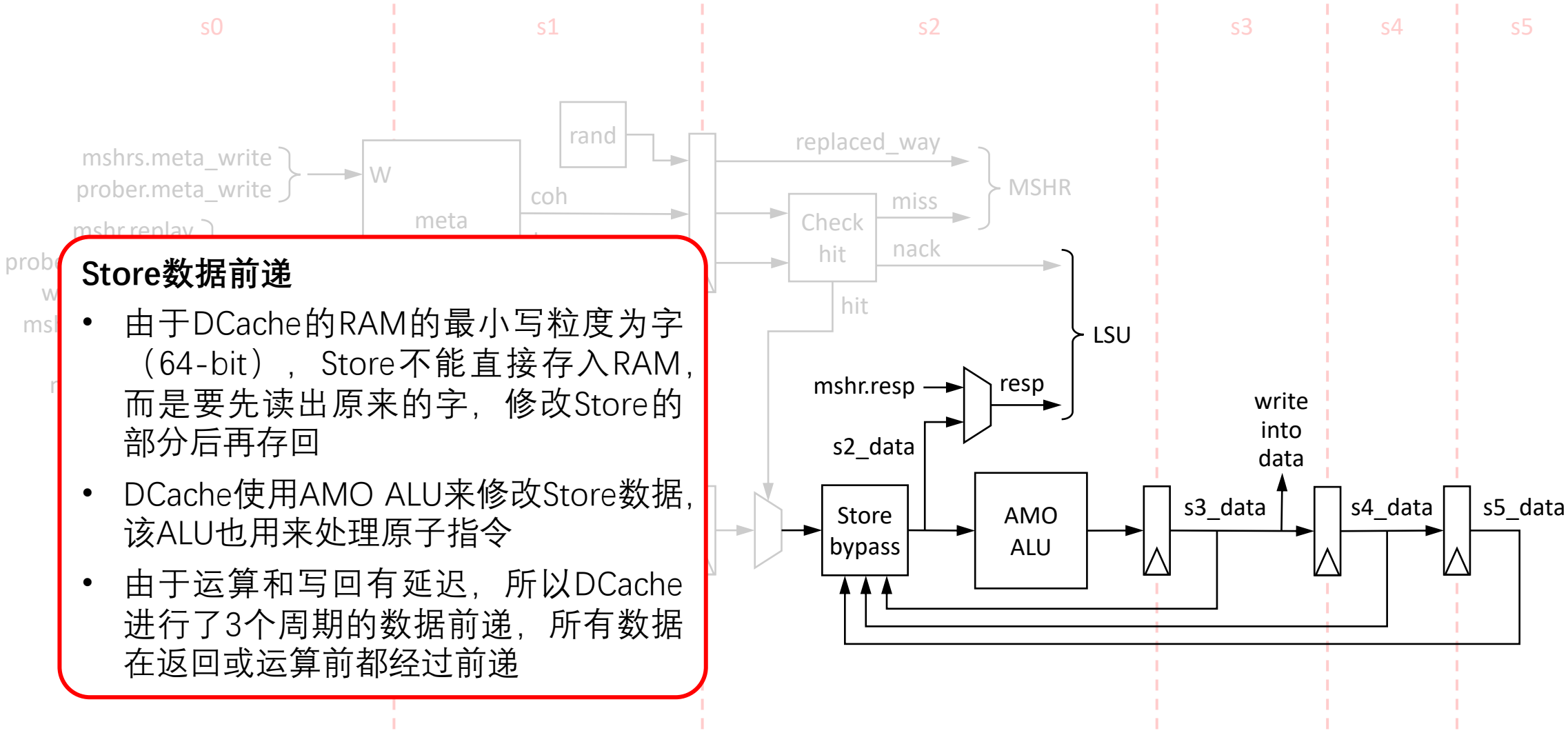


要么大家都有读权限



要么只有一个核有写权限
其他核都无效

DCache流水线



Store数据前递

- 由于DCache的RAM的最小写粒度为字（64-bit），Store不能直接存入RAM，而是要先读出原来的字，修改Store的部分后再存回
- DCache使用AMO ALU来修改Store数据，该ALU也用来处理原子指令
- 由于运算和写回有延迟，所以DCache进行了3个周期的数据前递，所有数据在返回或运算前都经过前递

MSHR

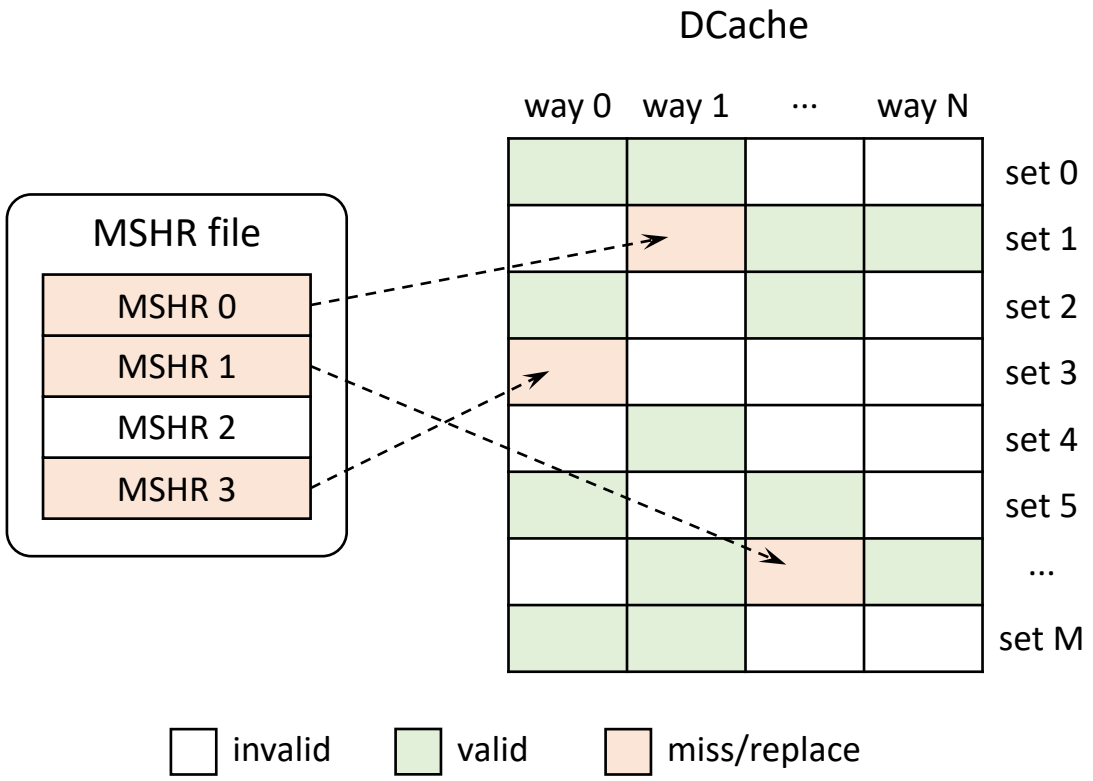
MSHR简介

- **全称：**Miss Status Holding Registers
- **作用：**保存Cache中未解决的miss的状态，并对miss进行处理，是非阻塞Cache的关键
- 多个MSHR组成一个MSHR file

MSHR的设计特点

- **单标量：**MSHR file每周期只能接收一个miss请求；当然，它可以同时保存多个miss请求
- **组互斥：**任意两个MSHR不能同时指向一组，即使两者指向的路不同；这是为了防止两个MSHR同时修改一行

➢ MSHR的策略是**尽力而为**，不保证接受所有的miss，所以LSU需要保留重试功能



MSHR file内部结构

某行首次遇到的miss

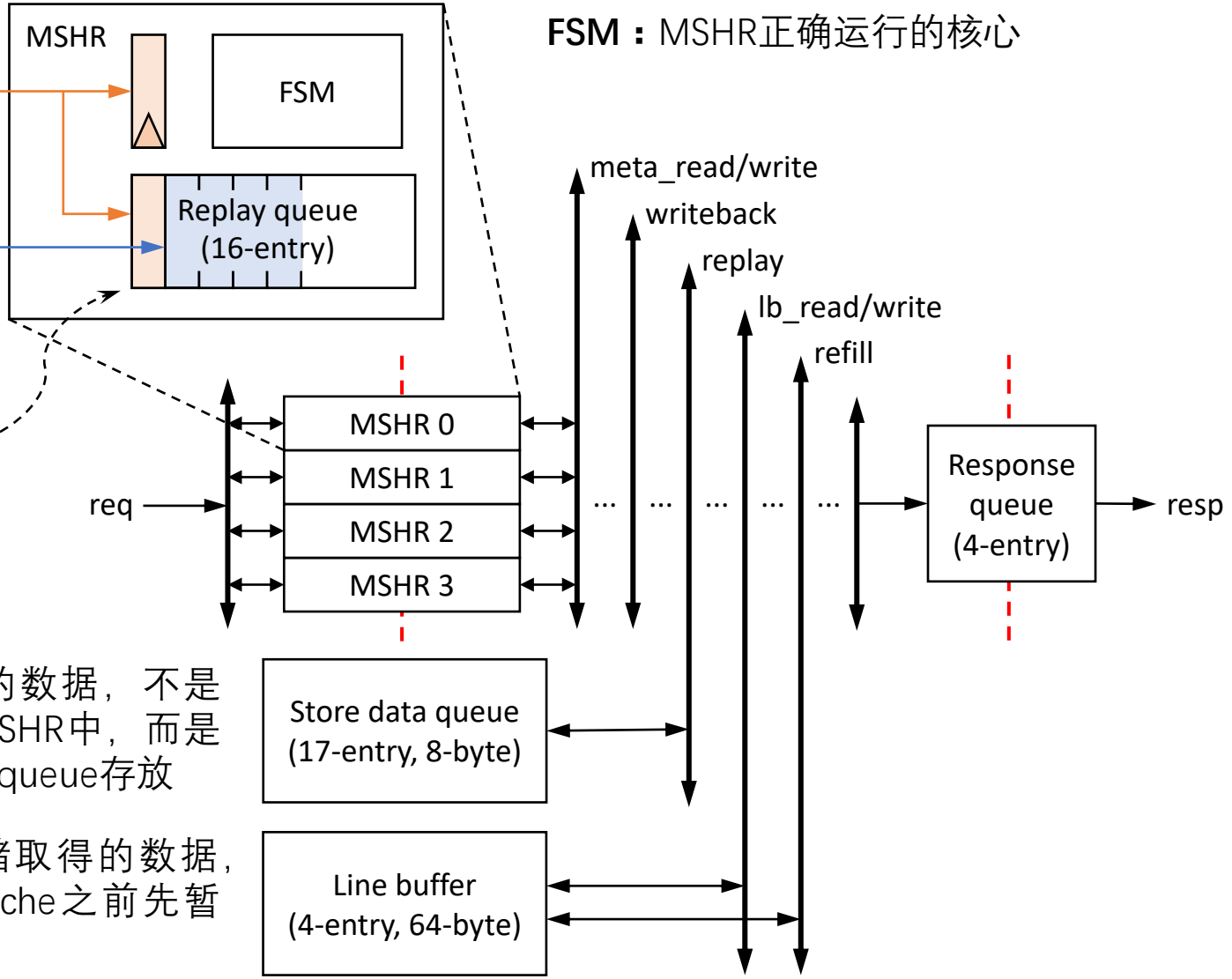
该行在miss处理期间再次遇到的所有miss

存放miss处理期间收到的所有miss请求，待数据取回后，以replay的方式重新访问DCache，并将结果回复给LSU

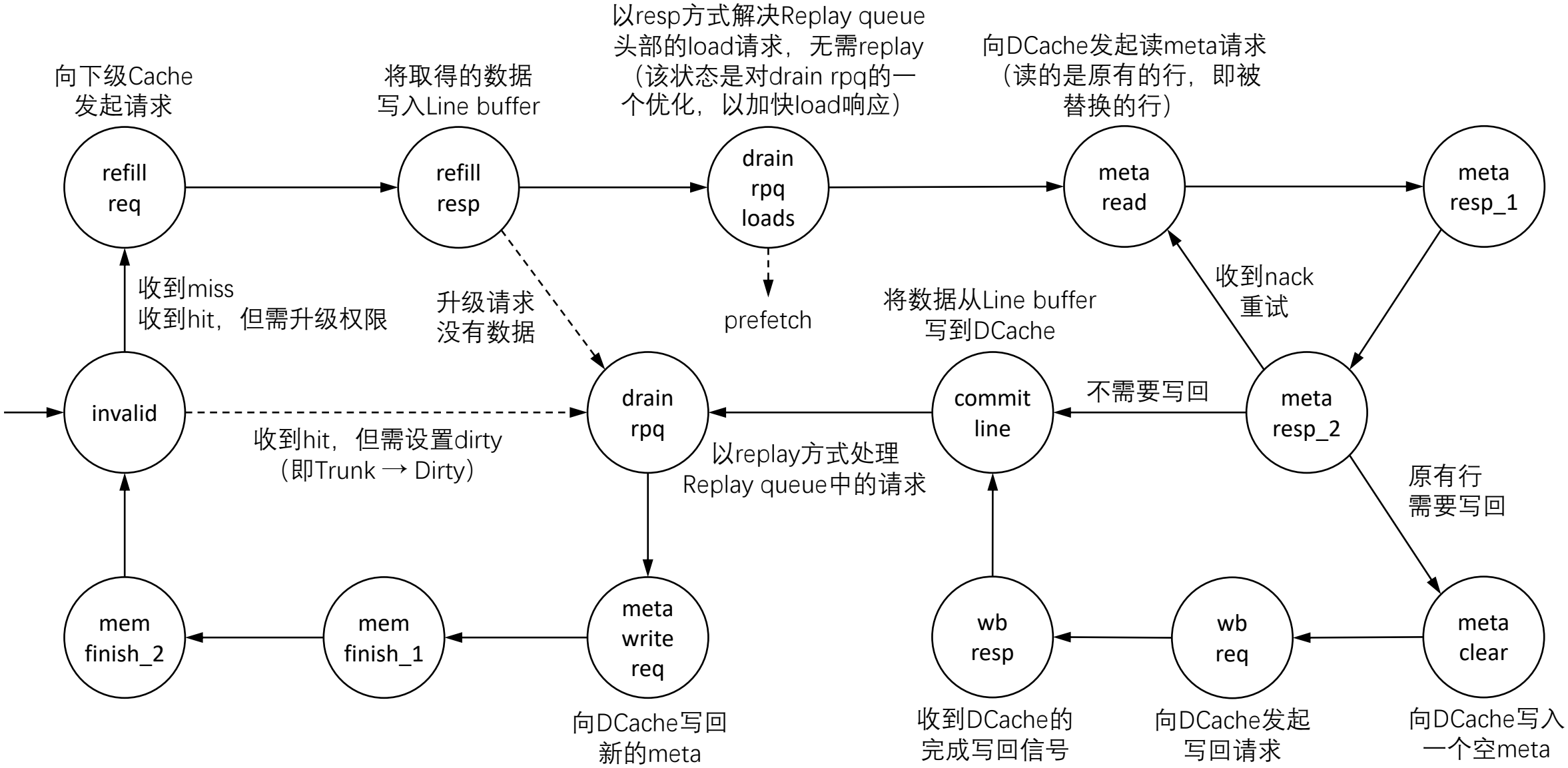
Store请求的数据，不是放在每个MSHR中，而是统一用一个queue存放

从下级存储取得的数据，在写入DCache之前先暂存在这里

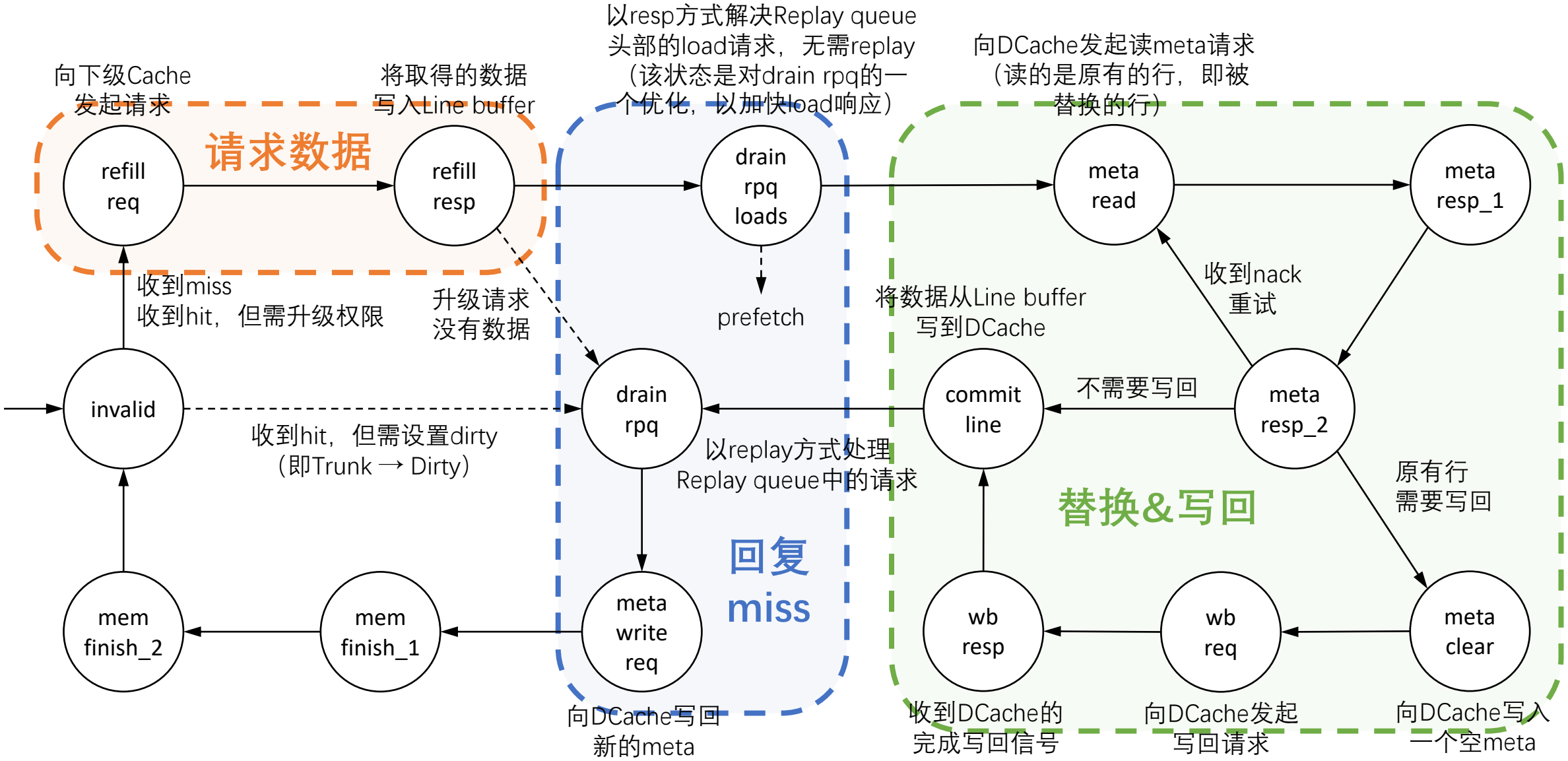
FSM：MSHR正确运行的核心



MSHR有限状态机


向下级Cache
发起请求
将取得的数据
写入Line buffer
以resp方式解决Replay queue
头部的load请求，无需replay
（该状态是对drain rpq的一
个优化，以加快load响应）
向DCache发起读meta请求
（读的是原有的行，即被
替换的行）
refill
req
refill
resp
drain
rpq
loads
meta
read
meta
resp_1
收到miss
收到hit，但需升级权限
升级请求
没有数据
prefetch
将数据从Line buffer
写到DCache
收到nack
重试
invalid
drain
rpq
commit
line
不需要写回
meta
resp_2
收到hit，但需设置dirty
（即Trunk → Dirty）
以replay方式处理
Replay queue中的请求
原有行
需要写回
mem
finish_2
mem
finish_1
meta
write
req
wb
resp
wb
req
meta
clear
向DCache写回
新的meta
收到DCache的
完成写回信号
向DCache发起
写回请求
向DCache写入
一个空meta

MSHR有限状态机


向下级Cache
发起请求
将取得的数据
写入Line buffer
以resp方式解决Replay queue
头部的load请求，无需replay
（该状态是对drain rpq的一
个优化，以加快load响应）
向DCache发起读meta请求
（读的是原有的行，即被
替换的行）
请求数据
refill
req
refill
resp
drain
rpq
loads
meta
read
meta
resp_1
收到miss
收到hit，但需升级权限
升级请求
没有数据
prefetch
将数据从Line buffer
写到DCache
收到nack
重试
invalid
drain
rpq
commit
line
不需要写回
meta
resp_2
收到hit，但需设置dirty
（即Trunk → Dirty）
以replay方式处理
Replay queue中的请求
原有行
需要写回
替换&写回
回复
miss
mem
finish_2
mem
finish_1
meta
write
req
wb
resp
wb
req
meta
clear
向DCache写回
新的meta
收到DCache的
完成写回信号
向DCache发起
写回请求
向DCache写入
一个空meta

北京大学
PEKING UNIVERSITY

4.
改进思路

推测式执行 / 非对齐访问 / VIPT

改进思路

北京大学 PEKING UNIVERSITY

提高LSU中推测式执行的性能

Load什么时候执行

- 错误的执行顺序会导致st-ld和ld-ld异常
- st-ld异常非常常见，在一些benchmark中高达7%的Load会发生该异常
- 在一些benchmark上DCache并非越大越好

➢ **对Load是否执行进行预测？**

依赖Load的指令什么时候发射

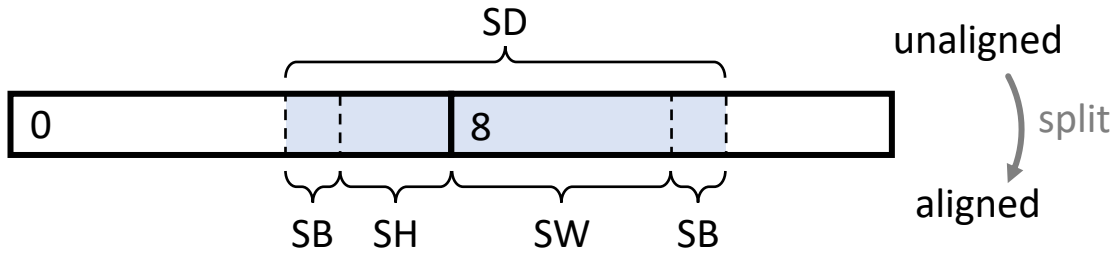
- 目前是Load首次执行时提前1周期唤醒
- 重试时也可唤醒，可提前2周期唤醒

➢ **更激进的唤醒策略？**

改进思路

LSU支持非对齐访问

- 考虑用多次访存来实现非对齐访问
- 根据非对齐跨度可分为：



方案	访存代价	说明
纯多次访存	Load 2次访存 Store 2-4次访存	实现简单，但Store效率低
多次访存 + DCache支持**字**内非对齐访问	Load 2次访存 Store 2次访存	效率较高； 需修改Searcher逻辑和DCache读/写逻辑
多次访存 + DCache支持**行**内非对齐访问	Load 1-2次访存 Store 1-2次访存	效率很高，除跨行外与对齐访问相同； 修改内容并不比字内方案多很多
DCache支持任意非对齐访问	Load 1次访存 Store 1次访存	DCache实现难度很大

- 综合效率与实现难度，个人最推荐方案3

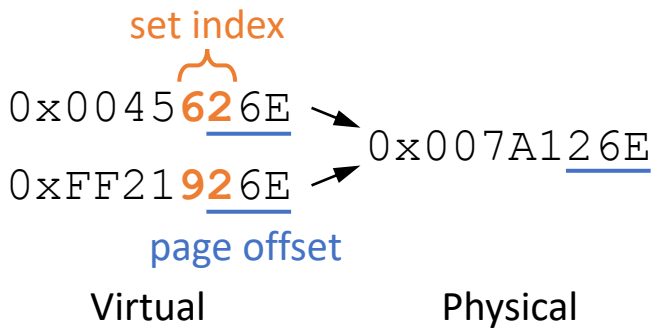
改进思路

LSU改用VIPT

- Virtual index, physical tag，即使用虚拟地址访问DCache
- BOOM的ICache、MIPS R10000的L1都是VIPT

VIPT的问题：Cache aliasing

- index超出页内偏移时，一个物理地址可能出现在两个cache line内
- ICache：只读，可以容忍aliasing
- R10000：使用copy-invalidate方案，禁止L1中出现重复的物理地址



VIPT的意义

- 访问RAM的同时访问TLB，节约TLB的周期
- 但在BOOM的DCache中实现VIPT的意义可能不是很大？
  - 因为目前TLB并没有单独占用一个周期
- 当然，使用VIPT之后TLB可以用一个周期访问，容量可以更大
  - MegaBoom的TLB容量为32，对比R10000为64

谢谢