IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

内存管理

陈海波 / 夏虞斌

上海交通大学并行与分布式系统研究所

https://ipads.se.sjtu.edu.cn

版权声明

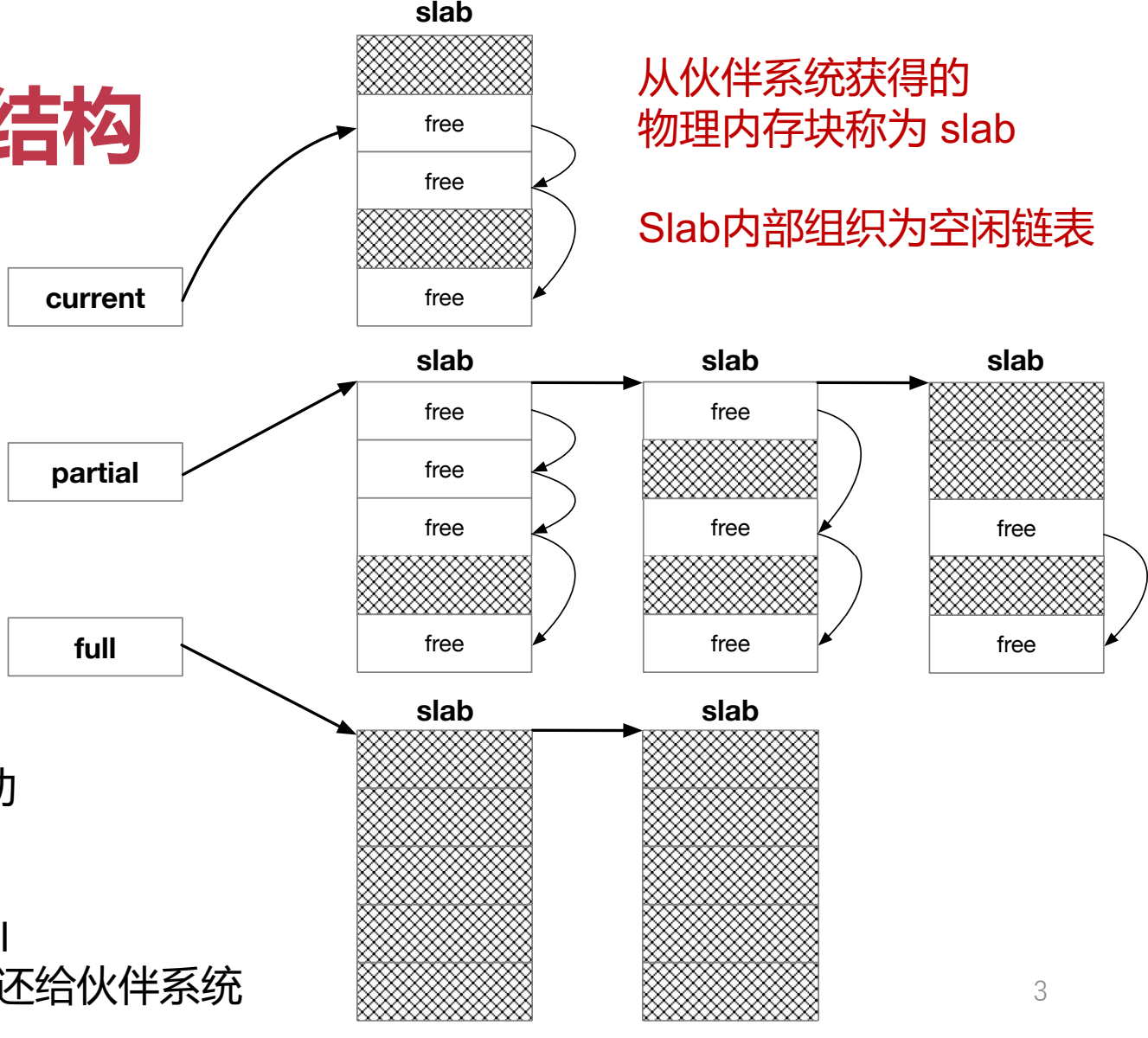
- 本内容版权归**上海交通大学并行与分布式系统研究所**所有
- 使用者可以将全部或部分本内容免费用于非商业用途
- 使用者在使用全部或部分本内容时请注明来源
  - 内容来自：上海交通大学并行与分布式系统研究所+材料名字
- 对于不遵守此声明或者其他违法使用本内容者，将依法保留追究权
- 本内容的发布采用 Creative Commons Attribution 4.0 License
  - 完整文本：https://creativecommons.org/licenses/by/4.0/legalcode

SLUB数据结构

从伙伴系统获得的
物理内存块称为 slab

Slab内部组织为空闲链表

三个指针
- current仅指向一个 slab
- partial指向未满slab链表
- full指向全满slab链表



分配使用current slab
- 若满发生两个移动

释放到对应的slab
- 移动 full 到 partial
- 若partial全free则还给伙伴系统

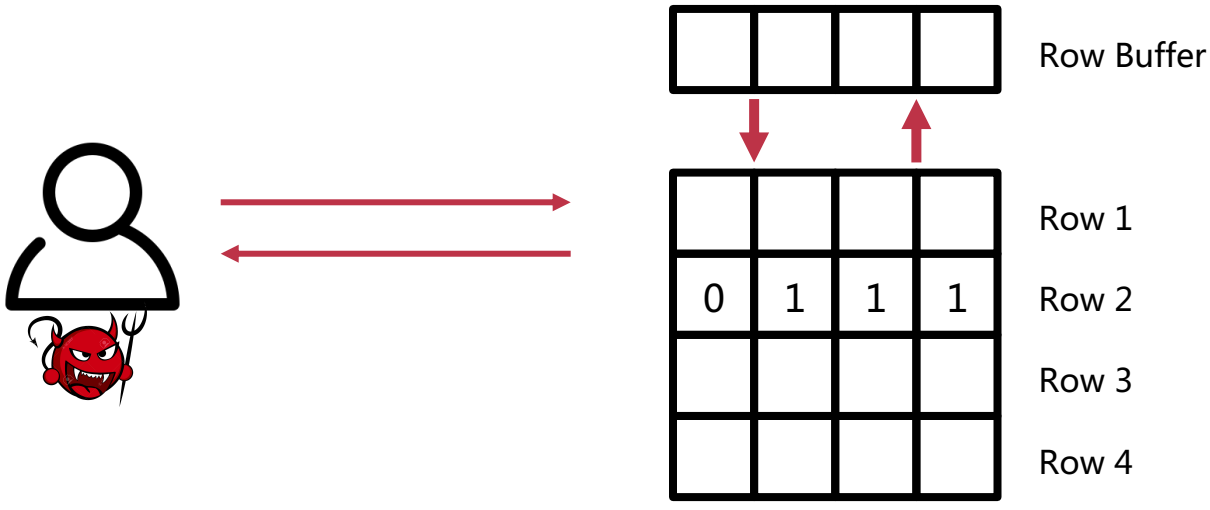
物理内存管理的其它问题

- **安全问题**
  - Rowhammer
  - Cache Side Channel

- **性能问题**
  - 性能隔离、QoS等

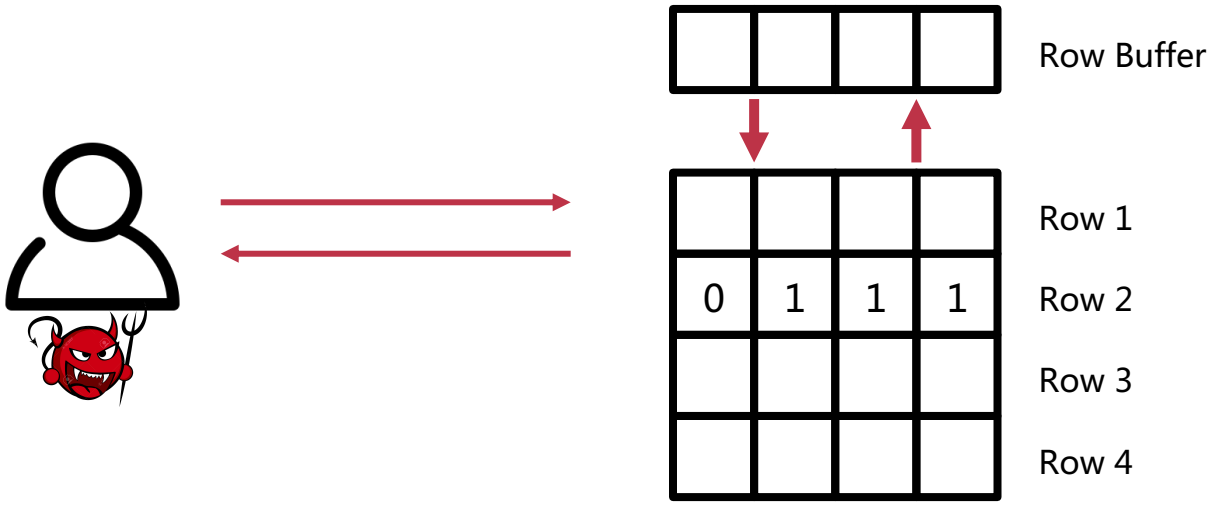
Rowhammer攻击

Recap：尽管Memory controller屏蔽了物理内存细节，
但是真实访问依然会用到Row等物理结构



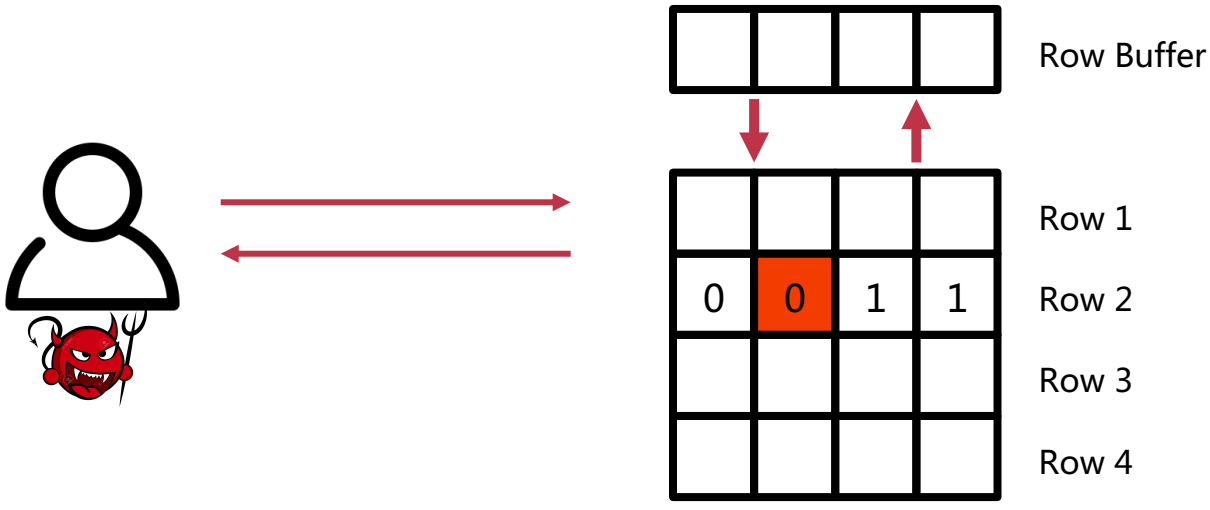
Rowhammer攻击

攻击者利用物理内存缺陷，极频繁访问某一行，其相邻行某些位会发生翻转



Rowhammer攻击

巧妙地利用位翻转，可以实施包括提权在内的多种攻击



安全防御

- 为抵御Rowhammer攻击，实际上操作系统需要知道部分硬件细节，从而能够在物理内存分配时主动加入一些Guard Page

- 为抵御cache Side Channel攻击，操作系统需要知道同样cache映射细节

性能考虑

- **Cache miss的开销很高**
  - 物理内存分配同样需要考虑到降低cache冲突几率
  - 典型机制：cache coloring

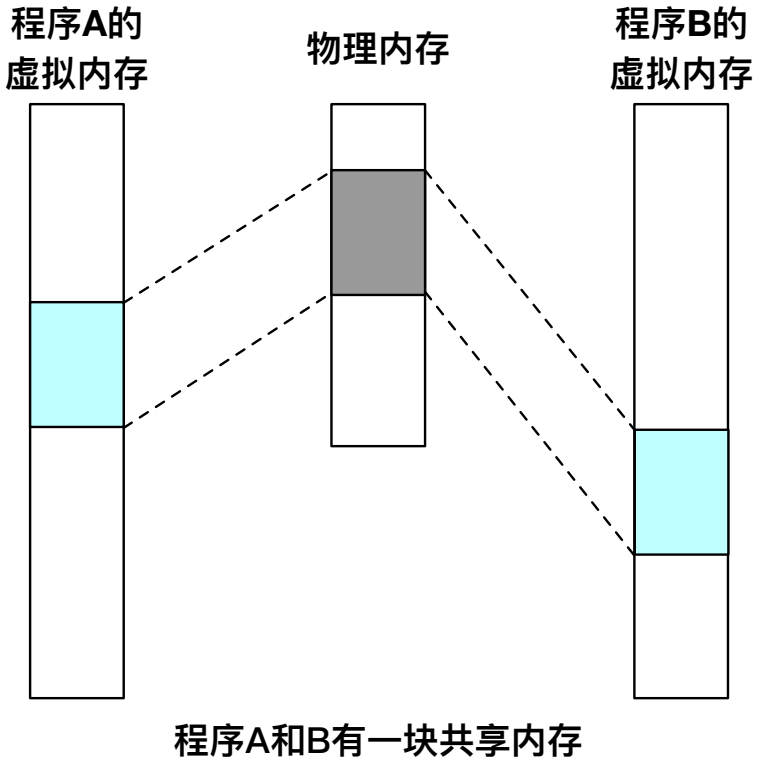
- **保证性能隔离是QoS的必要前提**
  - Intel CAT
  - AARCH64 MPAM

- ☑ 虚拟内存抽象
- ☑ 物理内存分配

操作系统内存管理的功能

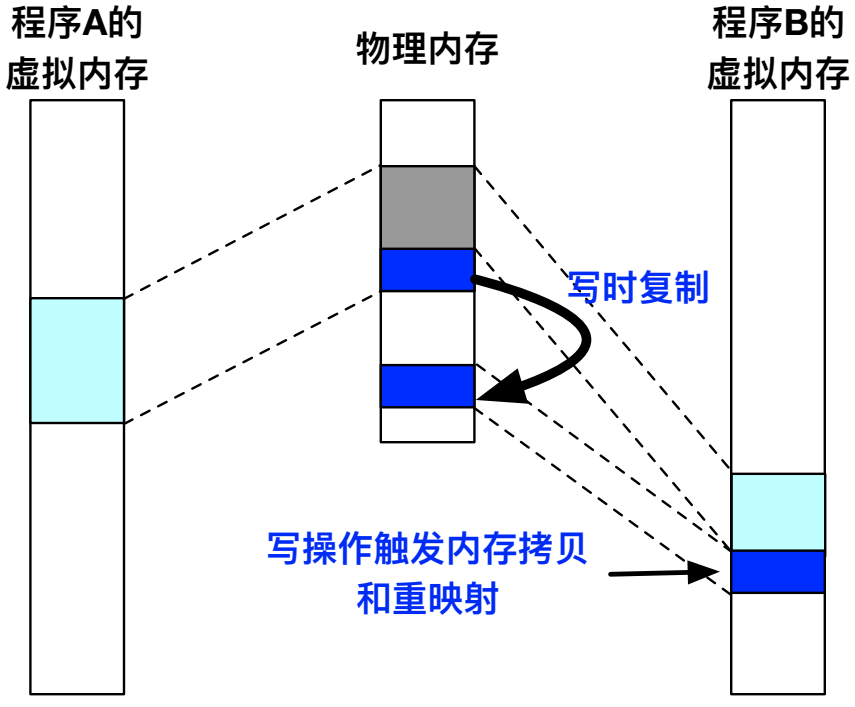
共享内存

- **基本功能**
  - 节约内存，如共享库
  - 进程通信，传递数据



写时拷贝（copy-on-write）

- **实现**
  - 修改页表项权限
  - 在缺页时拷贝、恢复
- **典型场景fork**
  - 节约物理内存
  - 性能加速



以写时拷贝的方式共享内存

内存去重

- **memory deduplication**
  - 基于写时拷贝机制
  - 在内存中扫描发现具有相同内容的物理页面
  - 执行去重
  - 操作系统发起，对用户态透明
- **典型案例：Linux KSM**
  - kernel same-page merging

内存去重潜在安全隐患

- **导致新的side channel**
  - 访问被合并的页会导致访问延迟明显
- **潜在攻击**
  - 攻击者可以确认目标进程中含有构造数据

- **思考：如何平衡这个tradeoff？**

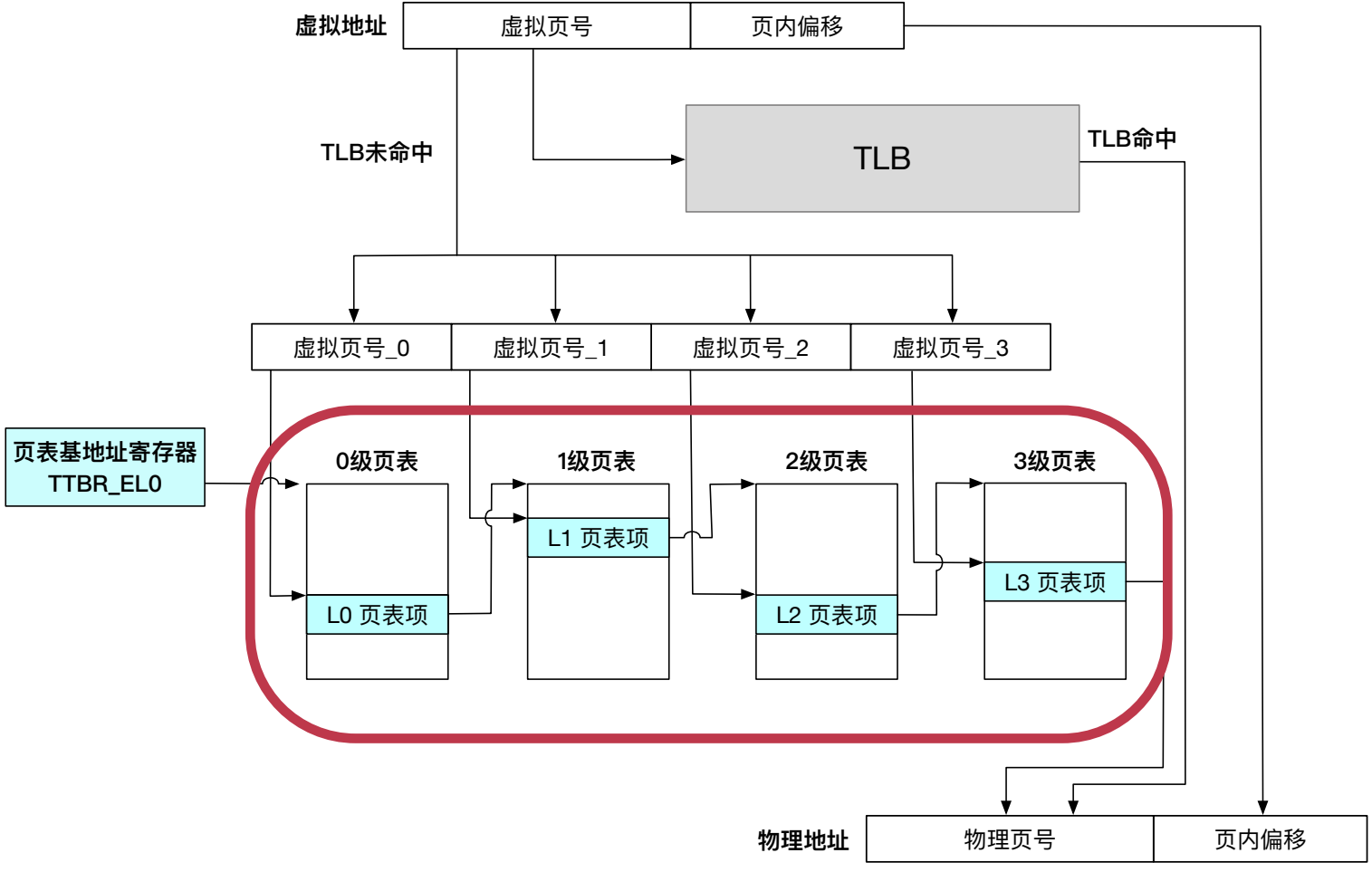
内存压缩

- **基本思想**
  - 当内存资源不充足的时候， 选择将一些“最近不太会使用”的内存页进行数据压缩，从而释放出空闲内存

内存压缩案例

- **Windows 10**
  - 压缩后的数据仍然存放在内存中
  - 当访问被压缩的数据时，操作系统将其解压即可
  - 思考：对比交换内存页到磁盘？
- **Linux**
  - zswap：换页过程中磁盘的缓存
  - 将准备换出的数据压缩并先写入 zswap 区域（内存）
  - 好处：减少甚至避免磁盘I/O；增加设备寿命

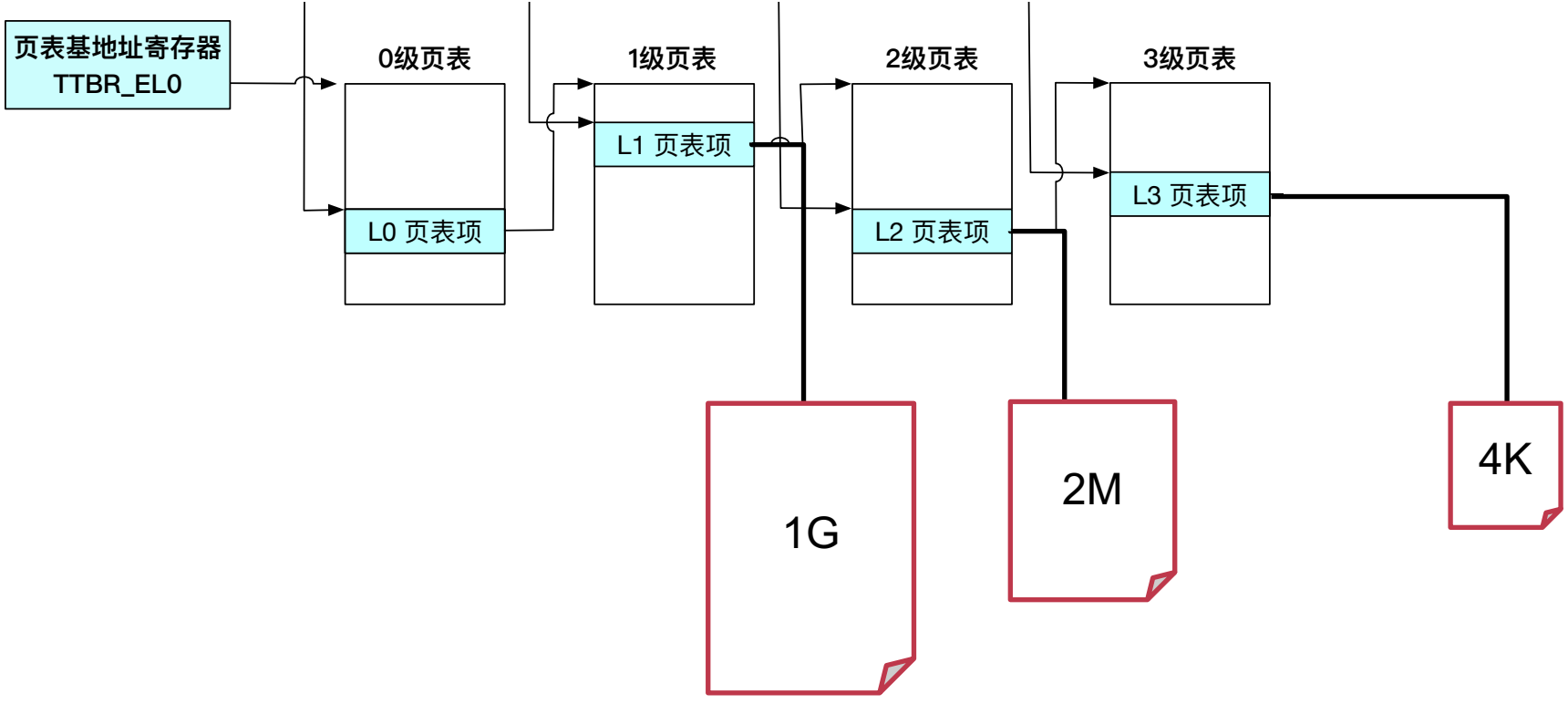
大页：再次回顾4级页表


虚拟地址
虚拟页号
页内偏移
TLB未命中
TLB
TLB命中
虚拟页号_0
虚拟页号_1
虚拟页号_2
虚拟页号_3
页表基地址寄存器
TTBR_EL0
0级页表
1级页表
2级页表
3级页表
L0 页表项
L1 页表项
L2 页表项
L3 页表项
物理地址
物理页号
页内偏移

大页

- **在4级页表中，某些页表项只保留两级或三级页表**
- **L2页表项的第1位**
  - 标识着该页表项中存储的物理地址（页号）是指向 L3 页表页（该位是 1）还是指向一个 2M 的物理页（该位是 0）
- **L1页表项的第1位**
  - 类似地，可以指向一个 1G 的物理页

大页


页表基地址寄存器
TTBR_EL0
0级页表
L0 页表项
1级页表
L1 页表项
2级页表
L2 页表项
3级页表
L3 页表项
1G
2M
4K

大页的利弊

- **好处**
  - 减少TLB缓存项的使用，提高 TLB 命中率
  - 减少页表的级数，提升遍历页表的效率
- **案例**
  - 提供API允许应用程序进行显示的大页分配
  - 透明大页（Transparent Huge Pages) 机制
- **弊端**
  - 未使用整个大页而造成物理内存资源浪费
  - 增加管理内存的复杂度

AARCH64支持多种最小页面大小

- **x86_64：4K**
- **AARCH64**
  - TCR_EL1可以配置3种：4K、16K、64K
  - 4K + 大页：2M/1G
  - 16K + 大页：32M（思考为什么是32M？）
    - 只有L2页表项支持大页
  - 64K + 大页：512M
    - 只有L2页表项支持大页（ARMv8.2之前）

思考

- 什么页/什么情况适合使用大页？
- 安卓关于大页使用的讨论

Linux上有趣的内存管理API

- **mmap**
  - void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset)

- **轶事：IPADS学生凭借mmap夺得世界冠军**
  - 文件操作，相比于read/write的优势
  - 避免特权级切换

Linux上有趣的内存管理API

- **madvise**
  - int madvise(void *addr, size_t length, int advice)

- **使用**
  - 用户态的语义信息告诉内核，便于优化
  - 例如：将madvise和mmap搭配起来使用，在使用数据前告诉内核这一段数据需要使用，从而减少缺页异常

思考题

- **在物理内存足够大的今天，虚拟内存是否还有存在的必要？**
  - 如果不使用虚拟内存抽象，恢复到只用物理内存寻址，会带来哪些改变？

- **如果不依靠 MMU，是否有可以替换虚拟内存的方法？**
  - 基于高级语言实现多个同一个地址空间内运行实例的隔离
  - 基于编译器插桩实现多个运行实例的隔离
    - 参考 Software Fault Isolation

下节课

- 进程