中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

1924-2024 中山大學 世纪华诞 100th ANNIVERSARY SUN YAT-SEN UNIVERSITY

网络协议栈与系统实现II

SSE202/204: 操作系统原理

苏玉鑫

suyx35@mail.sysu.edu.cn

助教：龙玉丹 单诗雯 毛晨希 沈志轩 郑灿峰 胡伟峰

版权信息

- 部分内容来自：上海交通大学并行与分布式系统研究所操作系统课件
  - https://ipads.se.sjtu.edu.cn/courses/os/
- 其它参考资料：
  - 清华大学操作系统公开课
    - https://open.163.com/newview/movie/courseintro?newurl=ME1NSA351
    - 介绍标准内容，适合考研
  - 南京大学计算机软件研究所
    - http://jyywiki.cn/OS/2025/
    - https://space.bilibili.com/202224425/channel/collectiondetail?sid=192498
    - 比较有趣

大纲

- 越俎代庖部分
  - 网络协议的分层模型
  - 套接字模型
- 网络驱动模型
- Linux系统收包过程
  - 函数视角
  - 数据视角
- Linux系统发包过程
- 网络处理性能优化
  - 挑战
  - 数据面控制面分离
- Intel DPDK 软件优化方案
  - 总体框架
  - 无锁环
  - 内存池
  - 其它模块
  - 扩展框架
- 硬件优化方案

Linux网络协议栈的问题

- 网络设备的速度越来越高
  - 100Mbps
  - 1Gbps
  - 10Gbps
  - 40Gbps
  - 200Gbps
  - 当前的上限：800G/1.6T
- CPU朝着多核方向发展
- 内核协议栈逐渐成为高性能网络的性能瓶颈

Linux网络协议栈的问题

- 中断处理
  - 大量网络包到来 -> 频繁的硬件中断请求
  - 中断频繁打断较低优先级的软中断或者系统调用的执行过程 -> 网络处理缓慢

- 上下文切换
  - 线程间的调度产生频繁上下文切换开销
  - 锁竞争的开销严重：共享缓存行

- 内存拷贝
  - 数据从网卡通过DMA的方式传到内核缓冲区，再从内核空间拷贝到用户空间
  - 占据Linux 内核协议栈数据包整个处理流程的 57.1%

Linux网络协议栈的问题

- 内存管理
  - 内存页大小为4K，对TLB要求更高

- 局部性失效
  - 数据包处理可能跨多核：导致CPU 缓存失效，空间局部性差
  - NUMA 架构：存在跨 NUMA内存访问，性能影响很大

大纲

- 越俎代庖部分
  - 网络协议的分层模型
  - 套接字模型
- 网络驱动模型
- Linux系统收包过程
  - 函数视角
  - 数据视角
- Linux系统发包过程
- 网络处理性能优化
  - 挑战
  - 数据面控制面分离
- Intel DPDK 软件优化方案
  - 总体框架
  - 无锁环
  - 内存池
  - 其它模块
  - 扩展框架
- 硬件优化方案

单选题 1分

设置

此题未设置答案，请点击右侧设置按钮

哪边的速度更快些?

```
for(int i = 0; i < 10; i++) {
    a[i] = func(a[i]);
}
```

```
a[0] = func(a[0]);
a[1] = func(a[1]);
a[2] = func(a[2]);
a[3] = func(a[3]);
a[4] = func(a[4]);
a[5] = func(a[5]);
a[6] = func(a[6]);
a[7] = func(a[7]);
a[8] = func(a[8]);
a[9] = func(a[9]);
i += 10;
i < 10;
// ...
```

左边快　　差不多　　右边快

提交

数据面(Data Plane)与控制面(Control Plane)分离

- 什么叫数据面?
  - 负责数据传输、处理的部分
- 什么叫控制面?
  - 负责控制流程的部分

数据面(Data Plane)与控制面(Control Plane)分离

- 什么叫数据面？
  - 负责数据传输、处理的部分
- 什么叫控制面？
  - 负责控制流程的部分

```
for(int i = 0; i < 10; i++) {
    a[i] = func(a[i]);
}
```

```
a[0] = func(a[0]);
i++;
i < 10;
a[1] = func(a[1]);
i++;
i < 10;
a[2] = func(a[2]);
i++;
i < 10;
a[3] = func(a[3]);
i++;
i < 10;
a[4] = func(a[4]);
i++;
i < 10;
//...
```

数据面(Data Plane)与控制面(Control Plane)分离

- 什么叫数据面？
  - 负责数据传输、处理的部分
- 什么叫控制面？
  - 负责控制流程的部分

```
for(int i = 0; i < 10; i++) {
    a[i] = func(a[i]);
}
```

```
a[0] = func(a[0]);
i++;
i < 10;
a[1] = func(a[1]);
i++;
i < 10;
a[2] = func(a[2]);
i++;
i < 10;
a[3] = func(a[3]);
i++;
i < 10;
a[4] = func(a[4]);
i++;
i < 10;
//...
```

和数据相关

和控制相关

数据面(Data Plane)与控制面(Control Plane)分离

- 什么叫数据面？
  - 负责数据传输、处理的部分
- 什么叫控制面？
  - 负责控制流程的部分

```
for(int i = 0; i < 10; i++) {
    a[i] = func(a[i]);
}
```

```
a[0] = func(a[0]);
i++;
i < 10;
a[1] = func(a[1]);
i++;
i < 10;
a[2] = func(a[2]);
i++;
i < 10;
a[3] = func(a[3]);
i++;
i < 10;
a[4] = func(a[4]);
i++;
i < 10;
//...
```

i 如果不再小于10呢

- 控制指令没出结果前，流水线不能真正有效地执行
  - 不知道要读哪段指令
- 控制部分和数据部分混合在一起，影响运行效率

数据面(Data Plane)与控制面(Control Plane)分离

- 什么叫数据面？
  - 负责数据传输、处理的部分
- 什么叫控制面？
  - 负责控制流程的部分

```
for(int i = 0; i < 10; i++) {
    a[i] = func(a[i]);
}
```

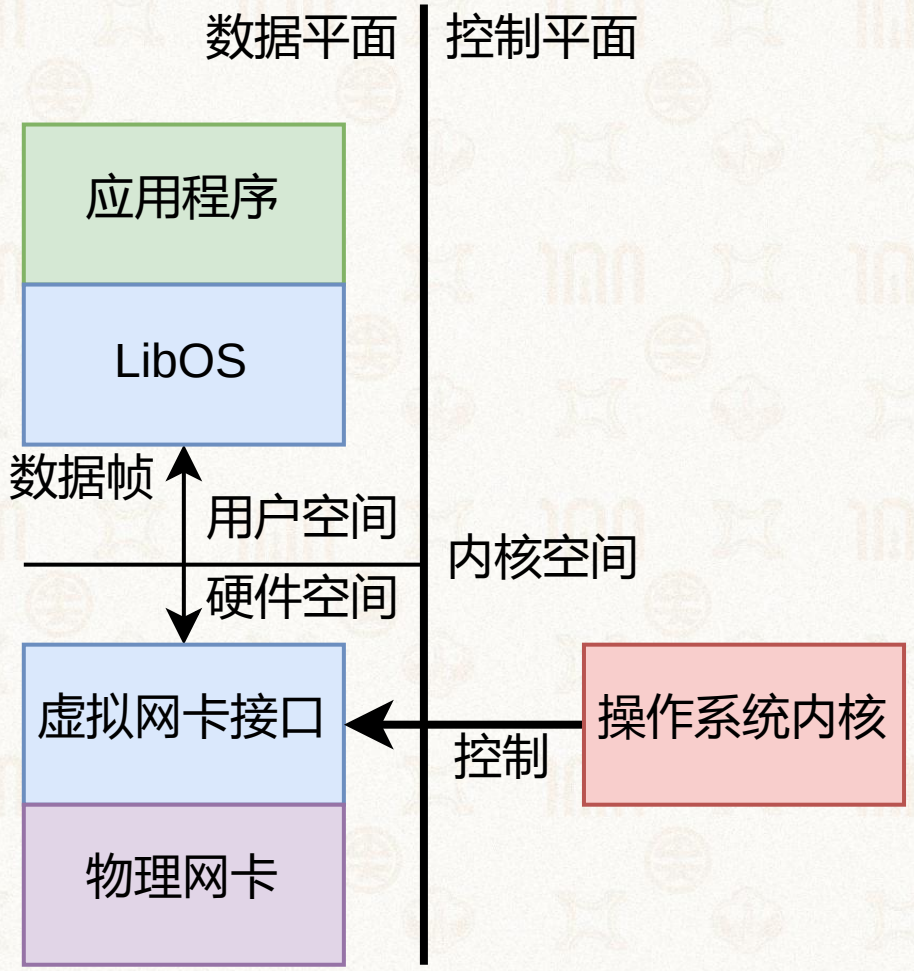
```
a[0] = func(a[0]);
a[1] = func(a[1]);
a[2] = func(a[2]);
a[3] = func(a[3]);
a[4] = func(a[4]);
a[5] = func(a[5]);
a[6] = func(a[6]);
a[7] = func(a[7]);
a[8] = func(a[8]);
a[9] = func(a[9]);
i += 10;
i < 10;
// ...
```

数据部分

- 去掉控制部分，数据部分可以全速运转

数据面(Data Plane)与控制面(Control Plane)分离

- 什么叫数据面?
  - 负责数据传输、处理的部分
- 什么叫控制面?
  - 负责控制流程的部分
- 通用的思想：控制面和数据面分离
  - 利于提高数据面的处理效率



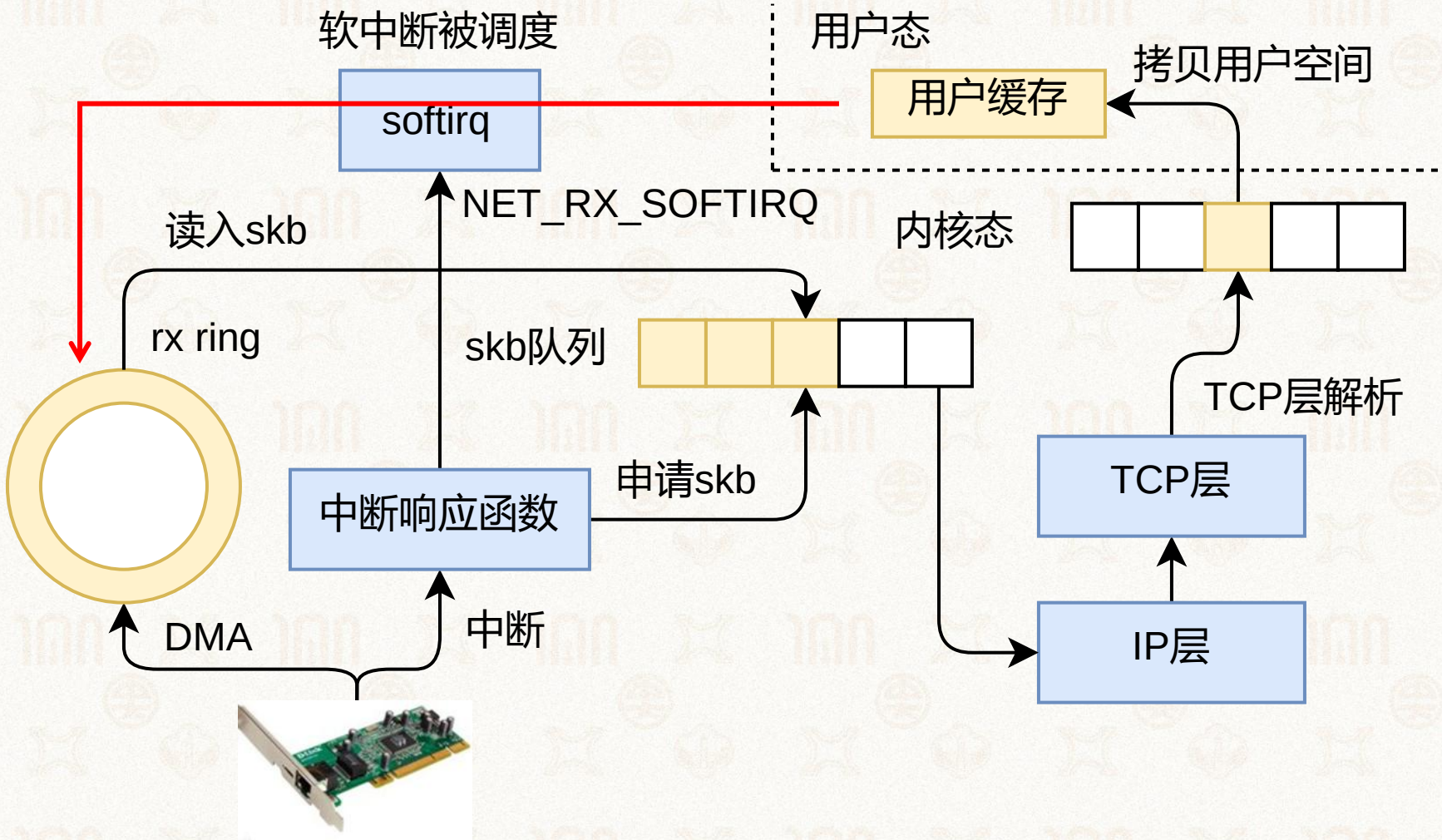
大纲

- 越俎代庖部分
  - 网络协议的分层模型
  - 套接字模型
- 网络驱动模型
- Linux系统收包过程
  - 函数视角
  - 数据视角
- Linux系统发包过程
- 网络处理性能优化
  - 挑战
  - 数据面控制面分离
- Intel DPDK 软件优化方案
  - 总体框架
  - 无锁环
  - 内存池
  - 其它模块
  - 扩展框架

  （无锁环、内存池）和操作系统课程最紧密的模块
- 硬件优化方案

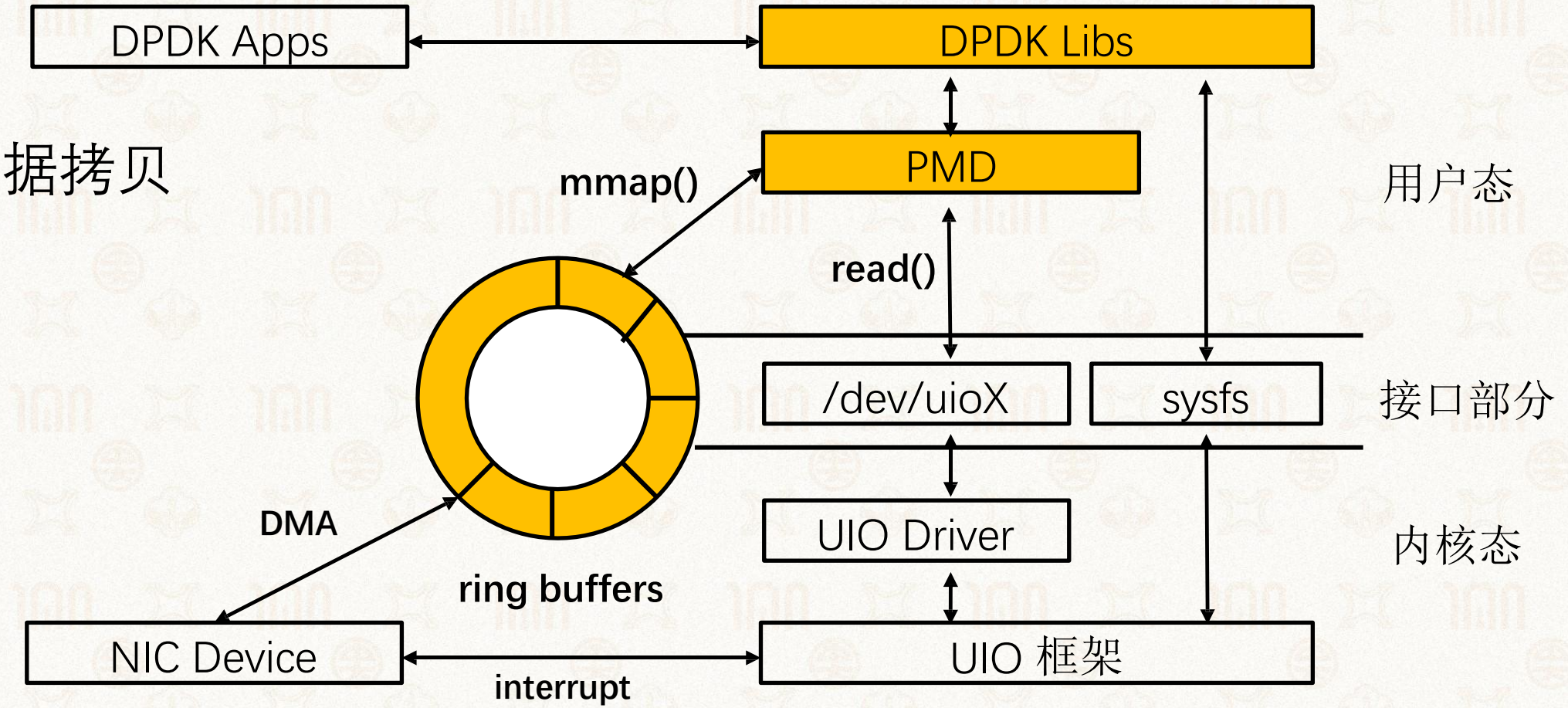
时下最流行的软件加速方案：Intel DPDK

- Data Plane Development Kit
- 绕过Linux内核协议栈（bypass kernel）
  - 直接在用户空间实现数据包的收发和处理
- Linux User I/O
  - 在用户态访问 MMIO（和设备交互）与DMA ring buffers
- 不用中断，改轮询：Poll Mode Driver (PMD)



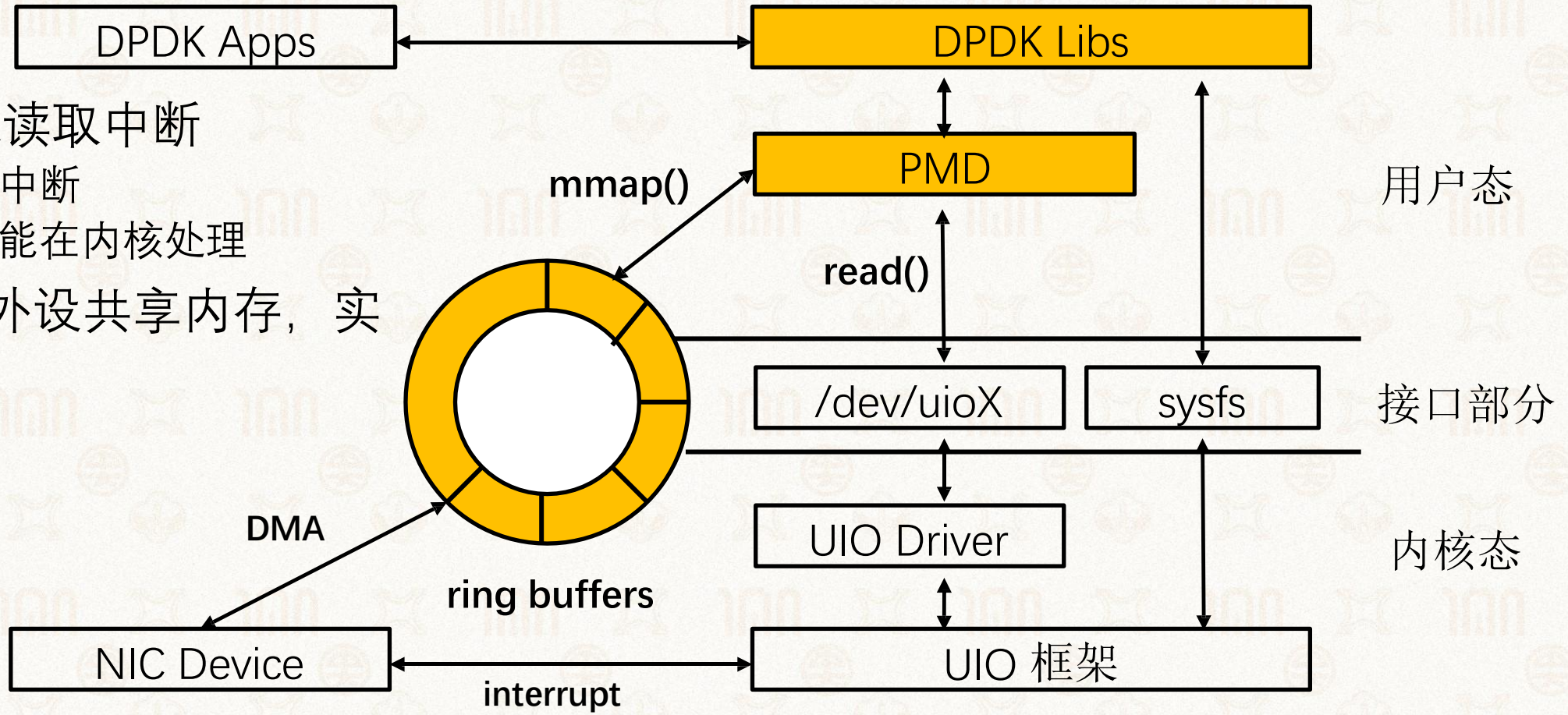
DPDK架构图

- 减少一次数据拷贝



Userspace I/O, 用户态I/O

- 通过/dev/uioX读取中断
  - 通过read感知中断
  - 因为硬中断只能在内核处理
- 通过mmap和外设共享内存，实现零拷贝



DPDK核心思想：穷尽一切手段优化性能

- 用户态模式下的PMD Driver
  - 去除了中断影响，减少了操作系统内核的开销，消除了IO吞吐瓶颈
  - 避免了内核态和用户态的报文拷贝
  - 用户态下软件崩溃，不会影响系统的稳定性
  - Intel提供的PMD驱动，充分利用指令和网卡的性能

- Huge Page和m_buf管理
  - 提供2M和1G的大页，减少了TLB Miss，TLB Miss严重影响报文转发性能；
  - 高效的m_buf管理，能够灵活的组织报文，包括多buffer接收，分片/重组，都能够轻松应对

- 向量指令
  - 明显的降低内存等待开销，提升CPU的流水线效率

DPDK核心思想：穷尽一切手段优化性能

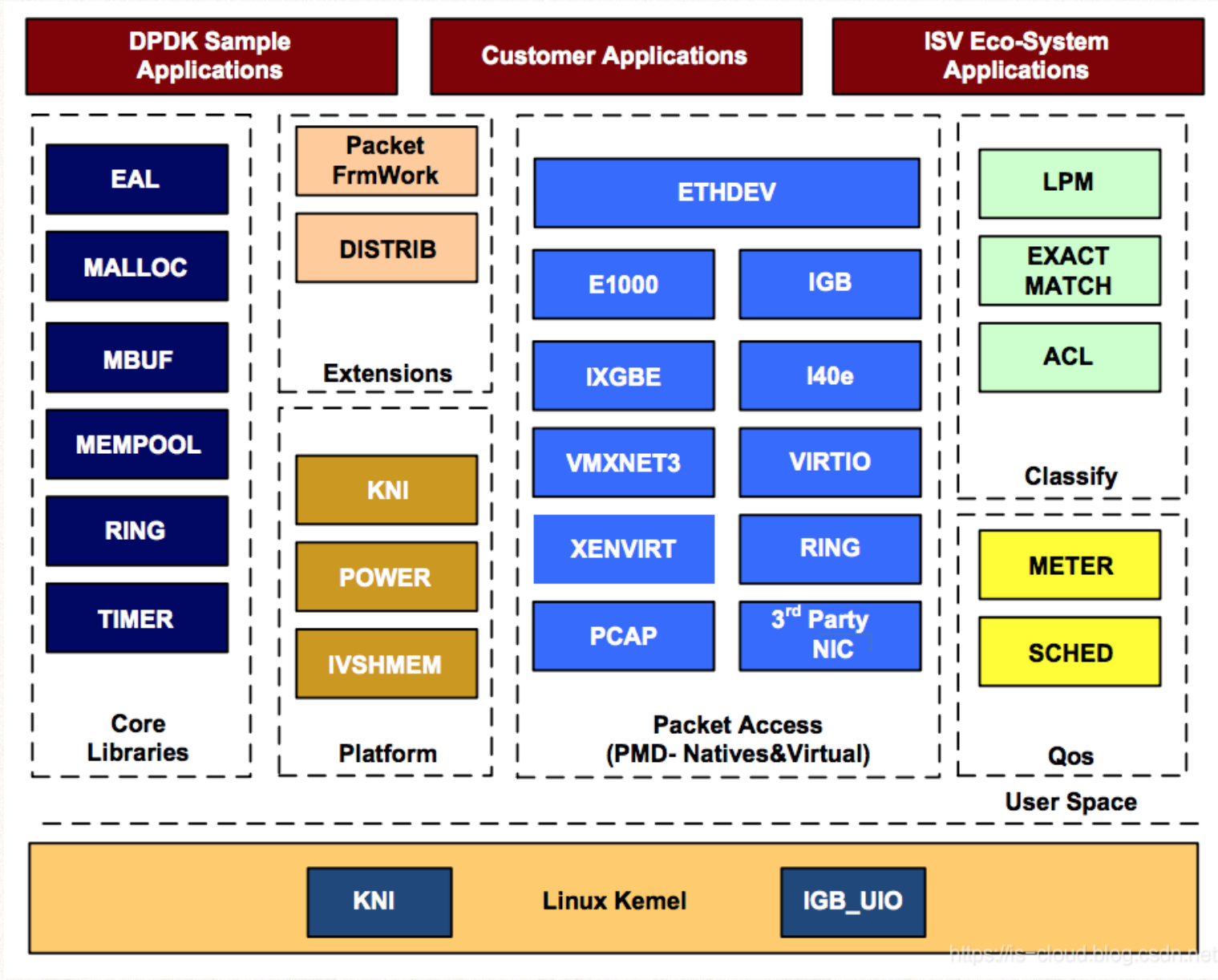
- 避免缓存不命中
- 避免分支预测错误

- 控制平面与数据平面相分离：
  - 内核态负责“控制平面”：内核仅负责控制指令的处理
  - 用户态负责“数据平面”：将数据包处理、内存管理、处理器调 度等任务转移到用户空间去完成，避免繁重的模式切换
- 用多核编程代替多线程技术
  - 绑核：设置 CPU 亲和性，减少彼此间调度切换
  - 无锁环形队列：解决资源竞争问题
- CPU 核尽量使用所在 NUMA 节点的内存
  - 避免跨NUMA内存访问

DPDK实现框架

- 核心库
  - 包括系统抽象内存管理、无锁环、缓存池
- 流分类(Packet Classification)
  - 支持精确匹配、最长匹配和通配符匹配，提供常用的包处理表操作
- 软件加速库(Accelerated SW Libraries)
  - 一些常用的包处理软件库集合，比如IP分片、报文重组、排序等
- QoS (Quality of Service)
  - 提供网络服务质量相关组件，比如限速和调度
- 数据包分组架构
  - 提供了搭建复杂的多核流水线模型的基础组件

DPDK模块汇总


DPDK Sample Applications
Customer Applications
ISV Eco-System Applications
EAL
MALLOC
MBUF
MEMPOOL
RING
TIMER
Core Libraries
Packet FrmWork
DISTRIB
Extensions
KNI
POWER
IVSHMEM
Platform
ETHDEV
E1000
IGB
IXGBE
I40e
VMXNET3
VIRTIO
XENVIRT
RING
PCAP
3rd Party NIC
Packet Access (PMD- Natives&Virtual)
LPM
EXACT MATCH
ACL
Classify
METER
SCHED
Qos
User Space
KNI
Linux Kemel
IGB_UIO

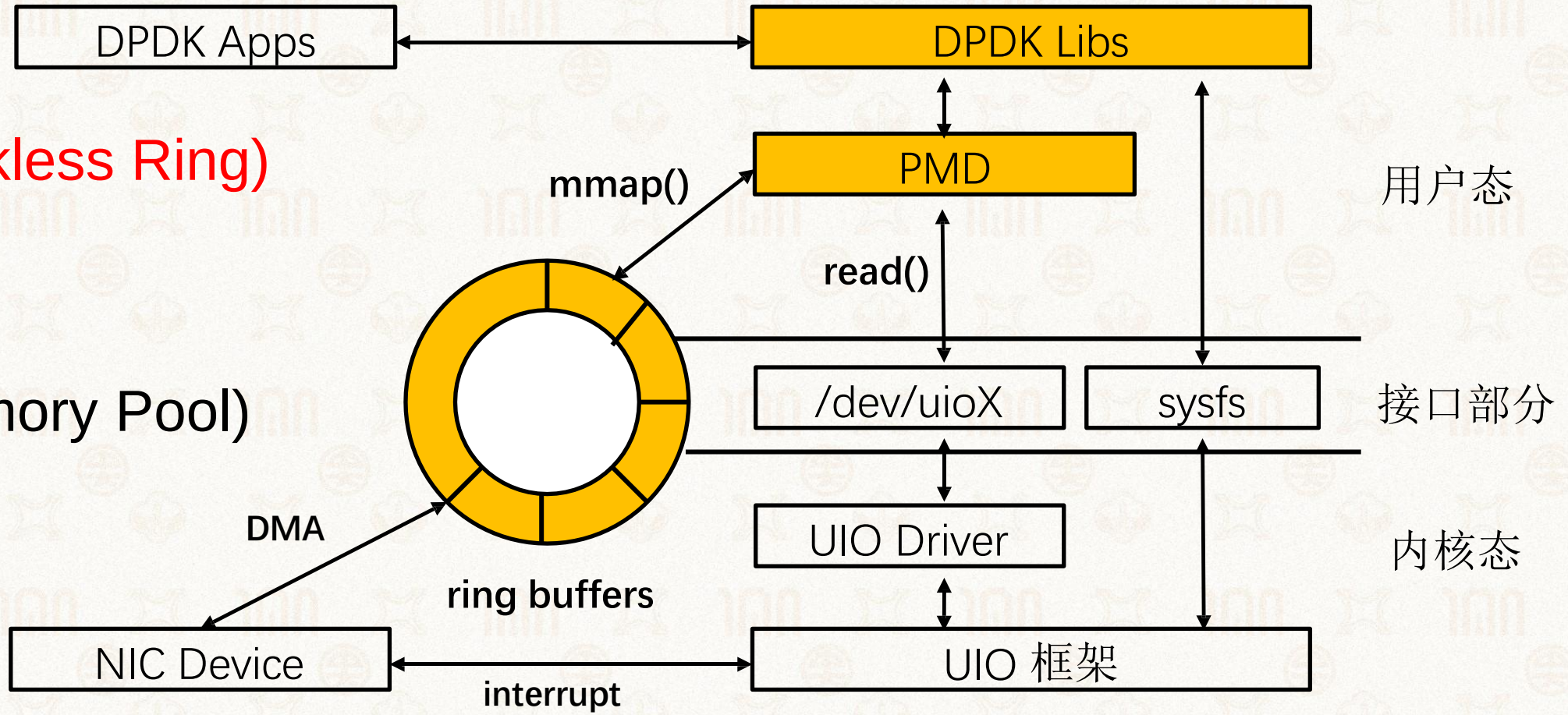
大纲

- 越俎代庖部分
  - 网络协议的分层模型
  - 套接字模型
- 网络驱动模型
- Linux系统收包过程
  - 函数视角
  - 数据视角
- Linux系统发包过程
- 网络处理性能优化
  - 挑战
  - 数据面控制面分离
- Intel DPDK 软件优化方案
  - 总体框架
  - 无锁环
  - 内存池
  - 其它模块
  - 扩展框架

  （无锁环、内存池）和操作系统课程最紧密的模块
- 硬件优化方案

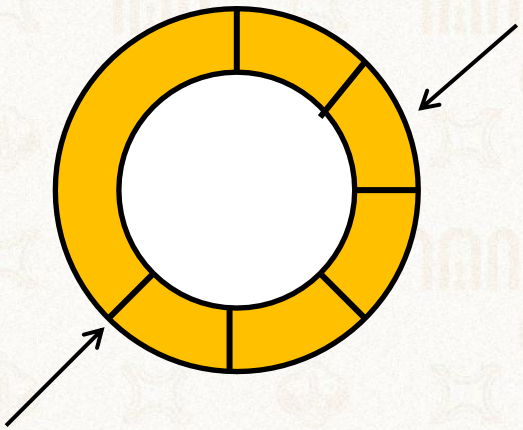
深入学习DPDK性能优化思想

- 无锁环(Lockless Ring)
- 内存池(Memory Pool)



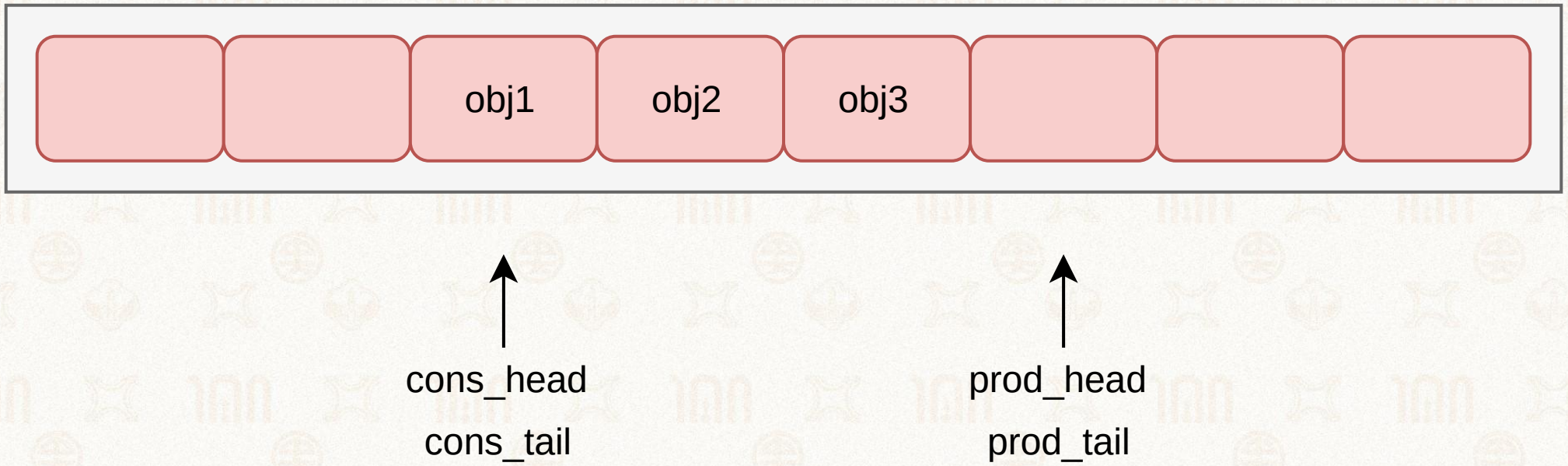
无锁环

- 首尾相连的数组，充当缓存
- 针对多生产者、多消费者的队列
- 不加锁，并行读写性能非常高
- 特殊点：
  - 元素定长：4字节或4字节的整数倍



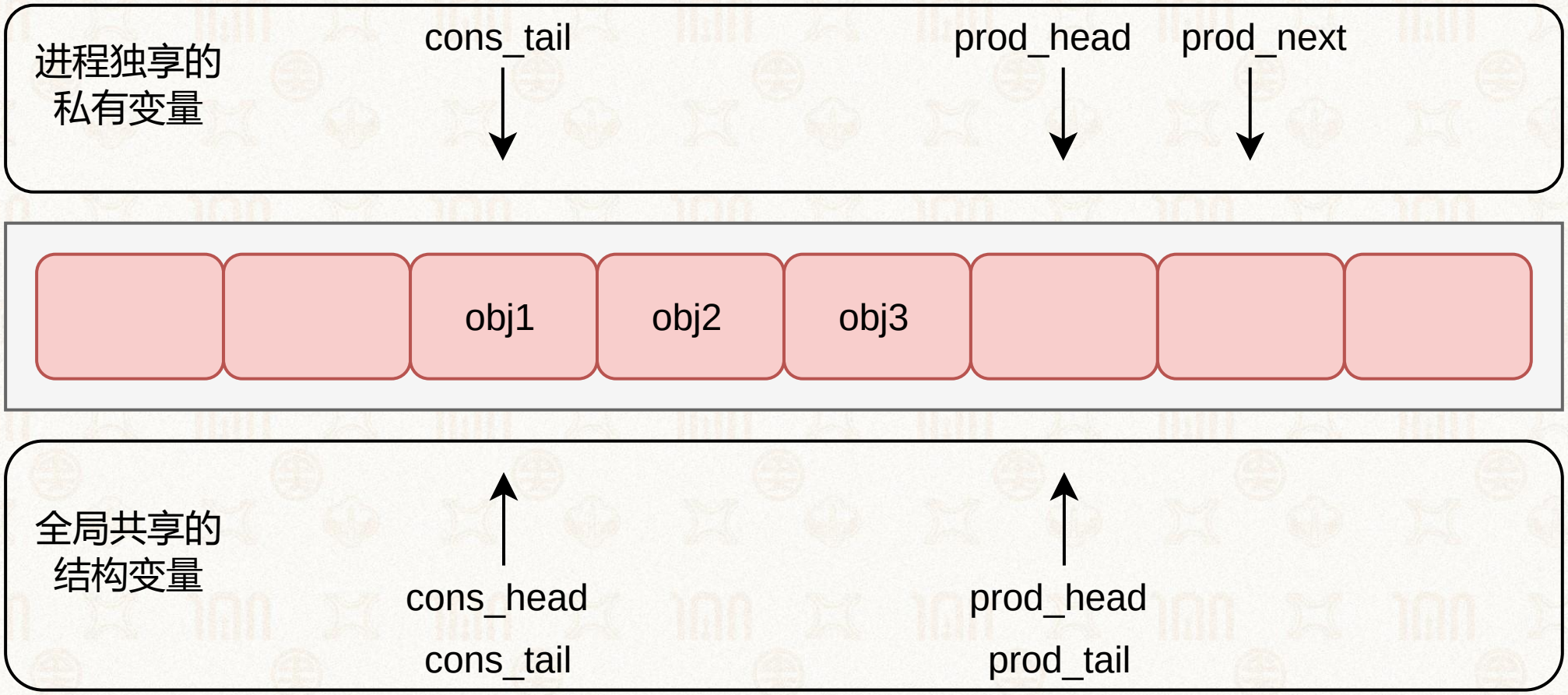
无锁环

- 生产者(producer):
  - 队头：prod_head
  - 队尾：prod_tail
- 消费者(consumer):
  - 队头：cons_head
  - 队尾：cons_tail
- 队头指向第一个元素
- 队尾指向最后一个元素的下一个位置



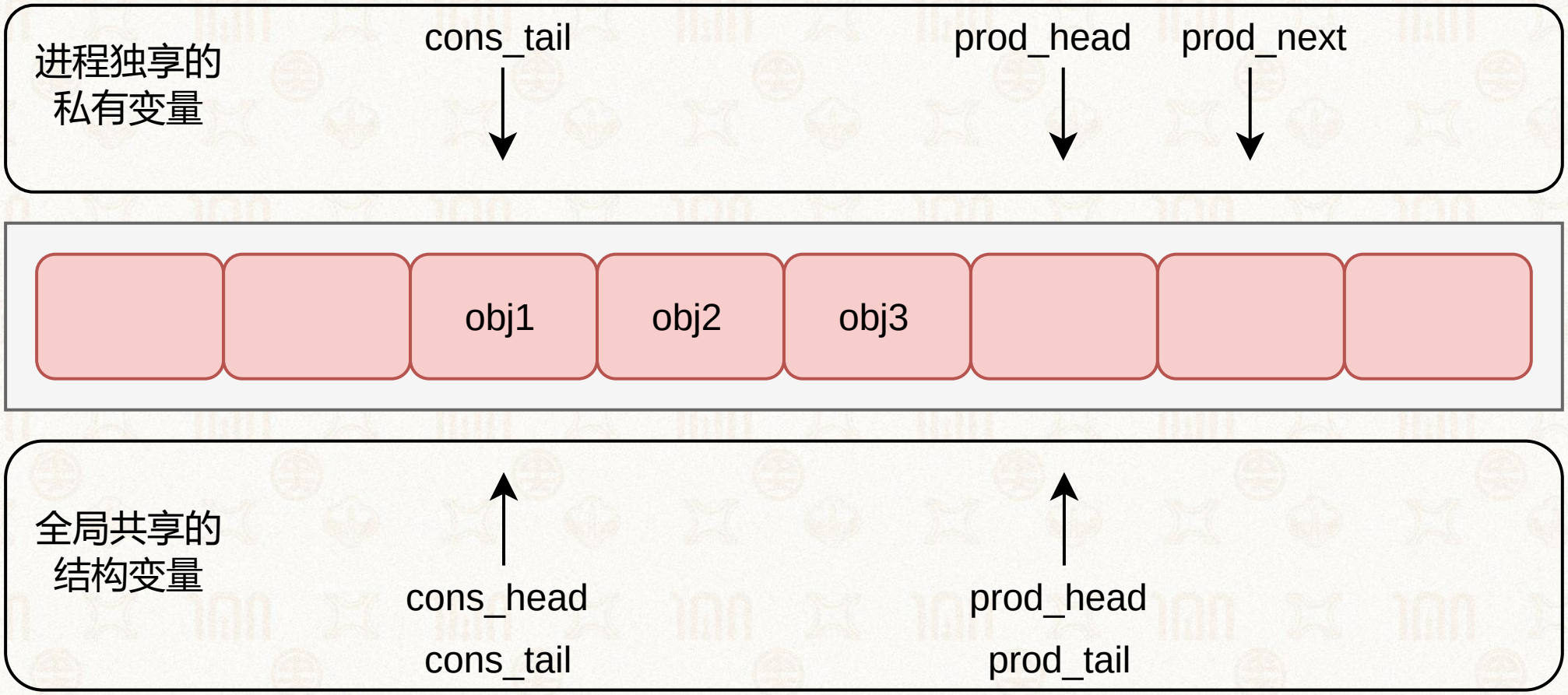
无锁环

- 指针分为共享共享与私有变量
- 执行具体操作之前，先从共享变量处复制一份私有变量



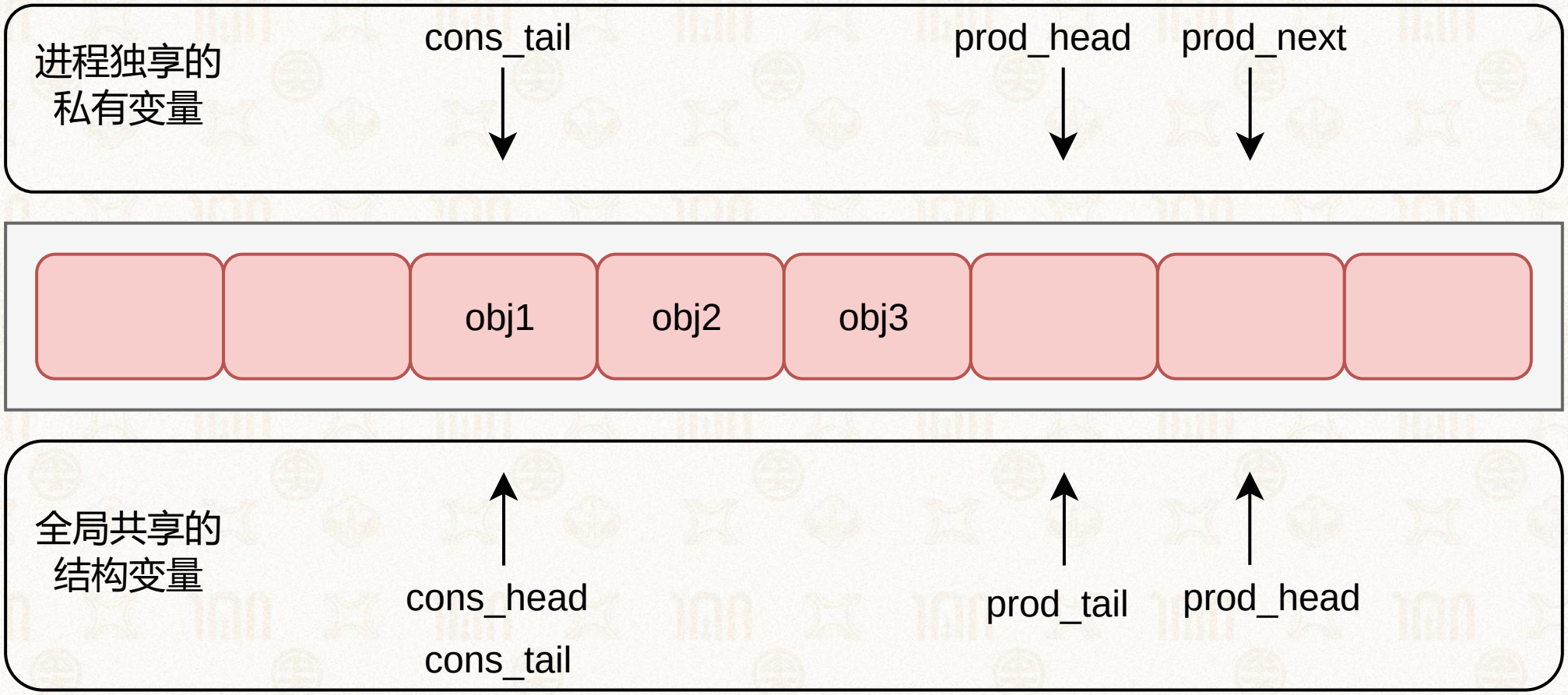
无锁环：生产者入队

- 先检查prod_head和cons_tail的关系判断队列是否已满
  - 若满(prod_head == cons_tail)直接返回错误



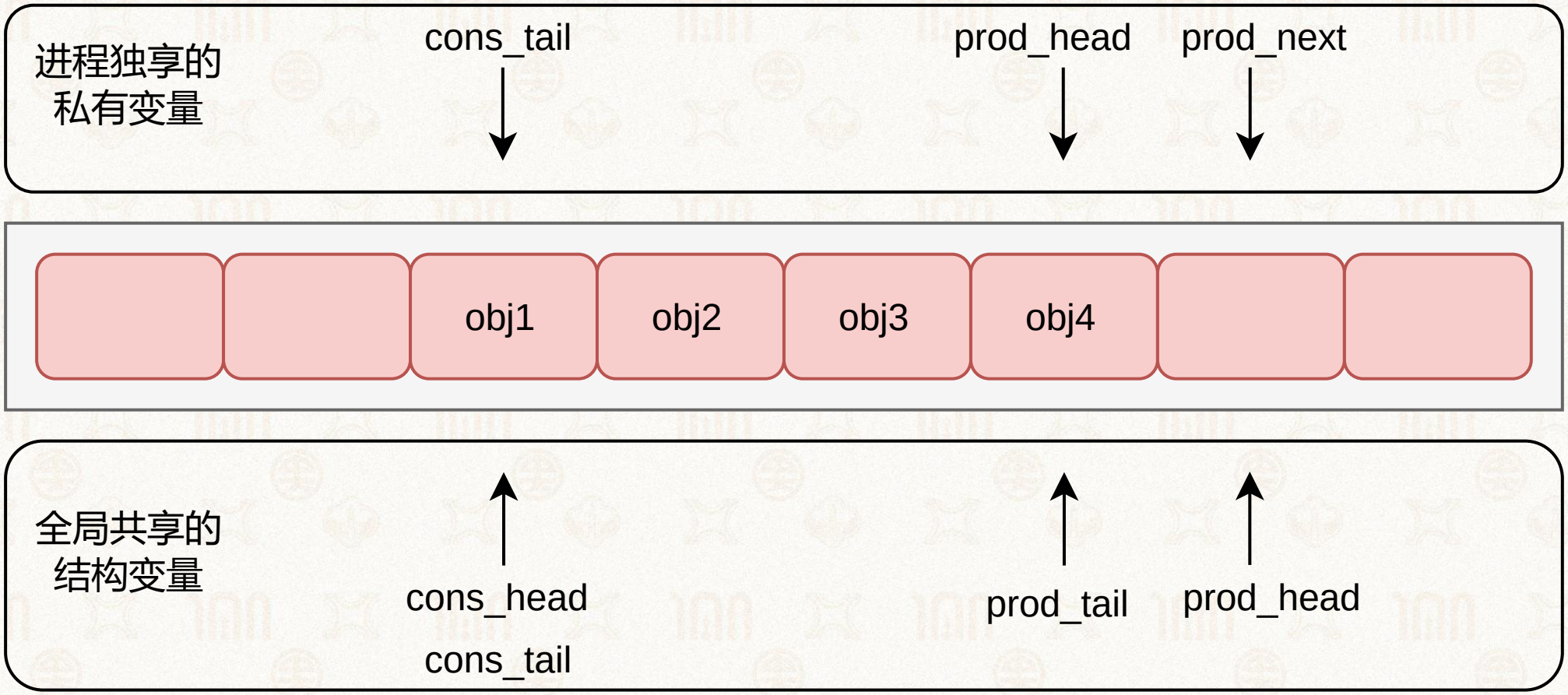
无锁环：生产者入队

- 若没满，将共享变量prod_head移向prod_next



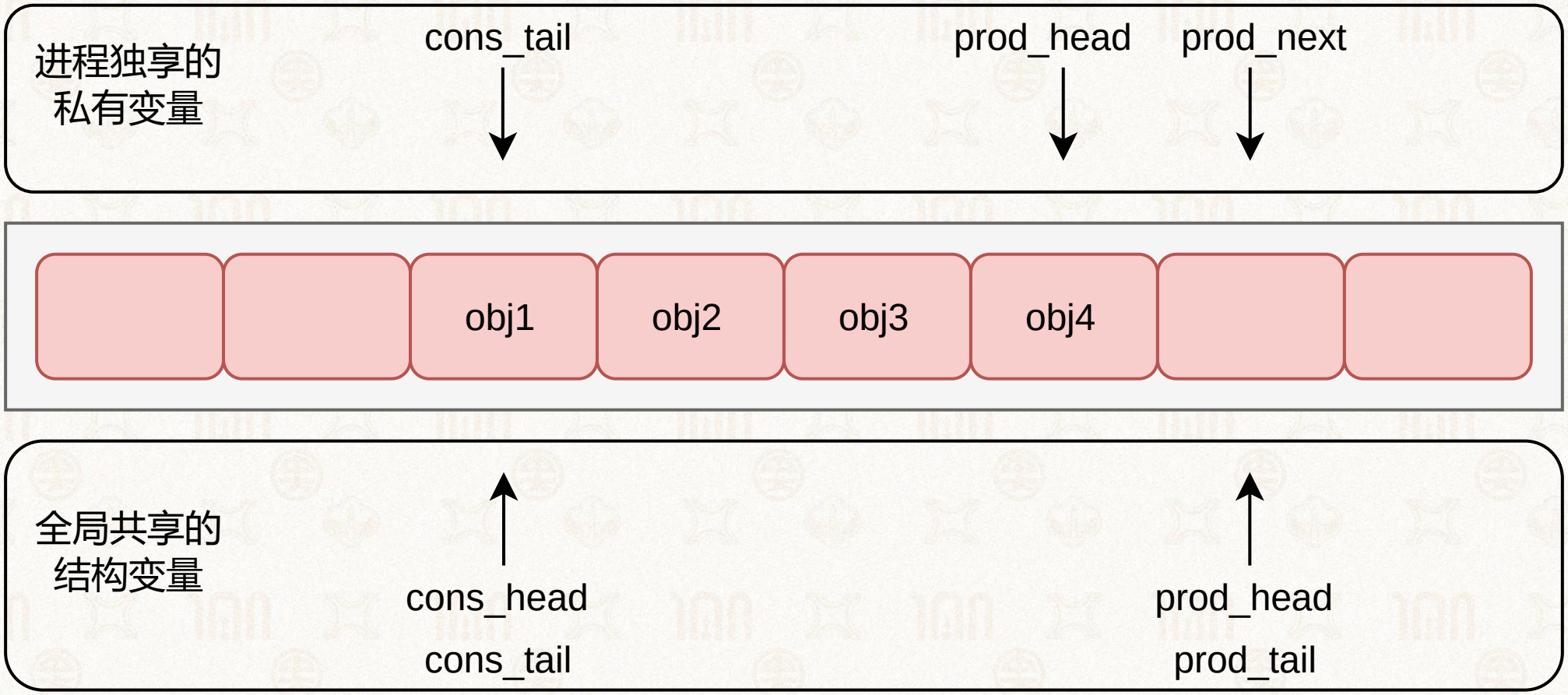
无锁环：生产者入队

- 向私有的prod_head写入新的数据



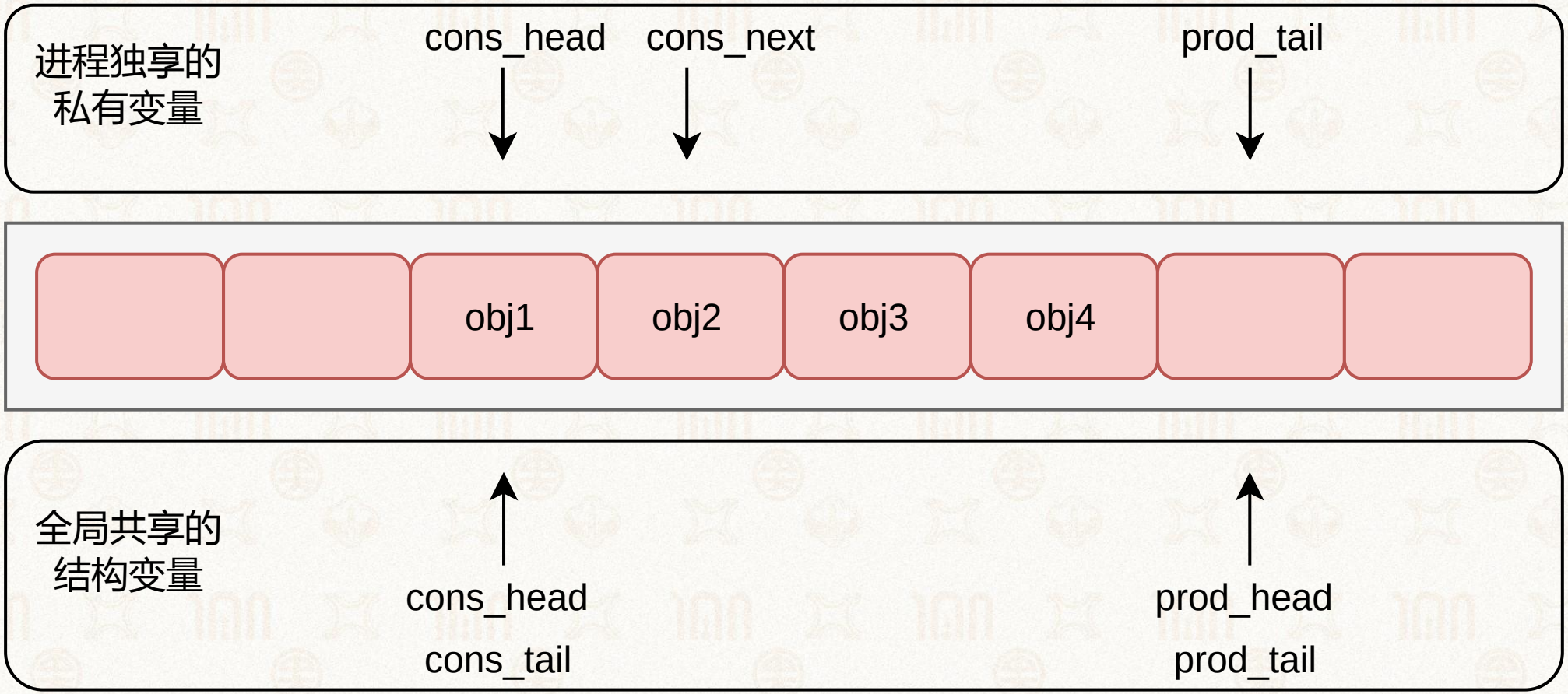
无锁环：生产者入队

- 最后将共享变量prod_tail移向共享变量prod_head



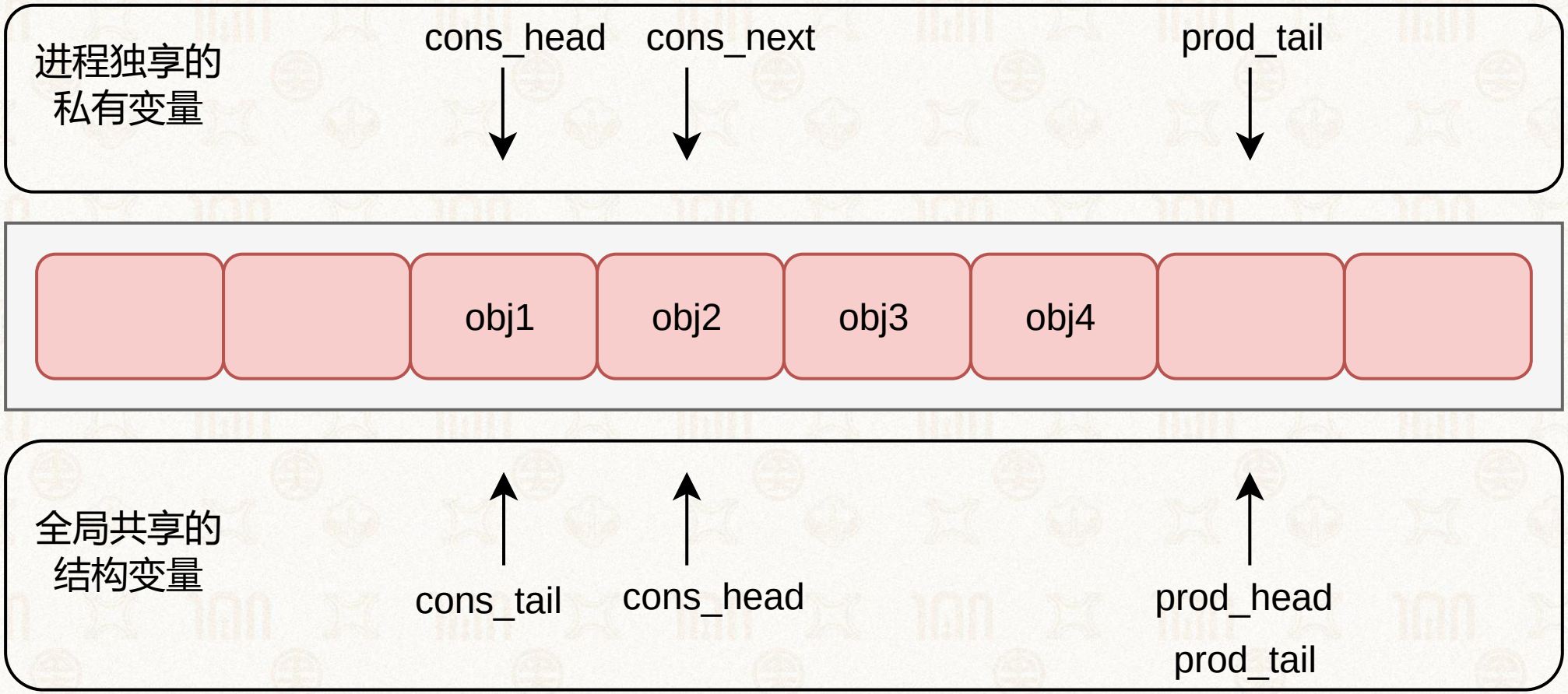
无锁环：消费者出队

- 先复制私有变量，再检查是否有元素可以出队
  - 比较cons_head和prod_tail



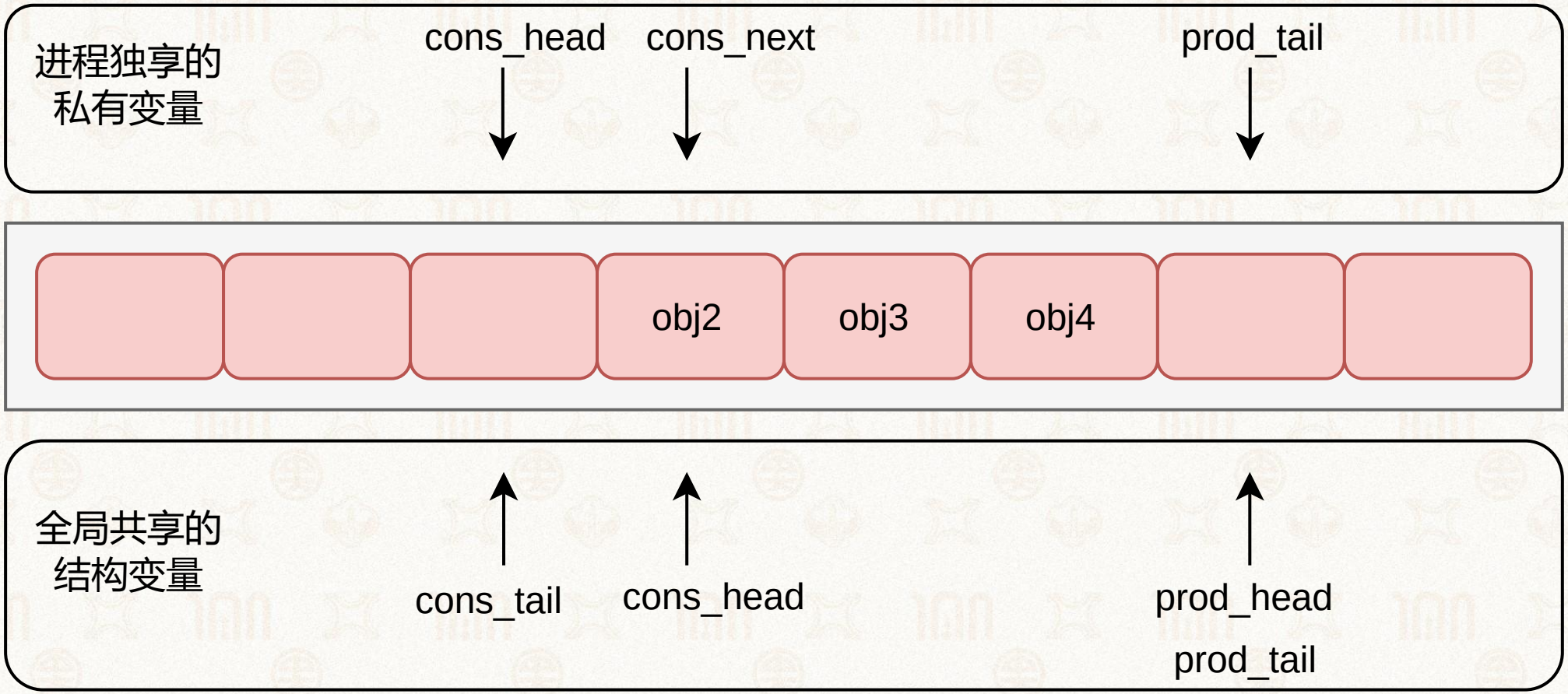
无锁环：消费者出队

- 将共享变量cons_head移向cons_next



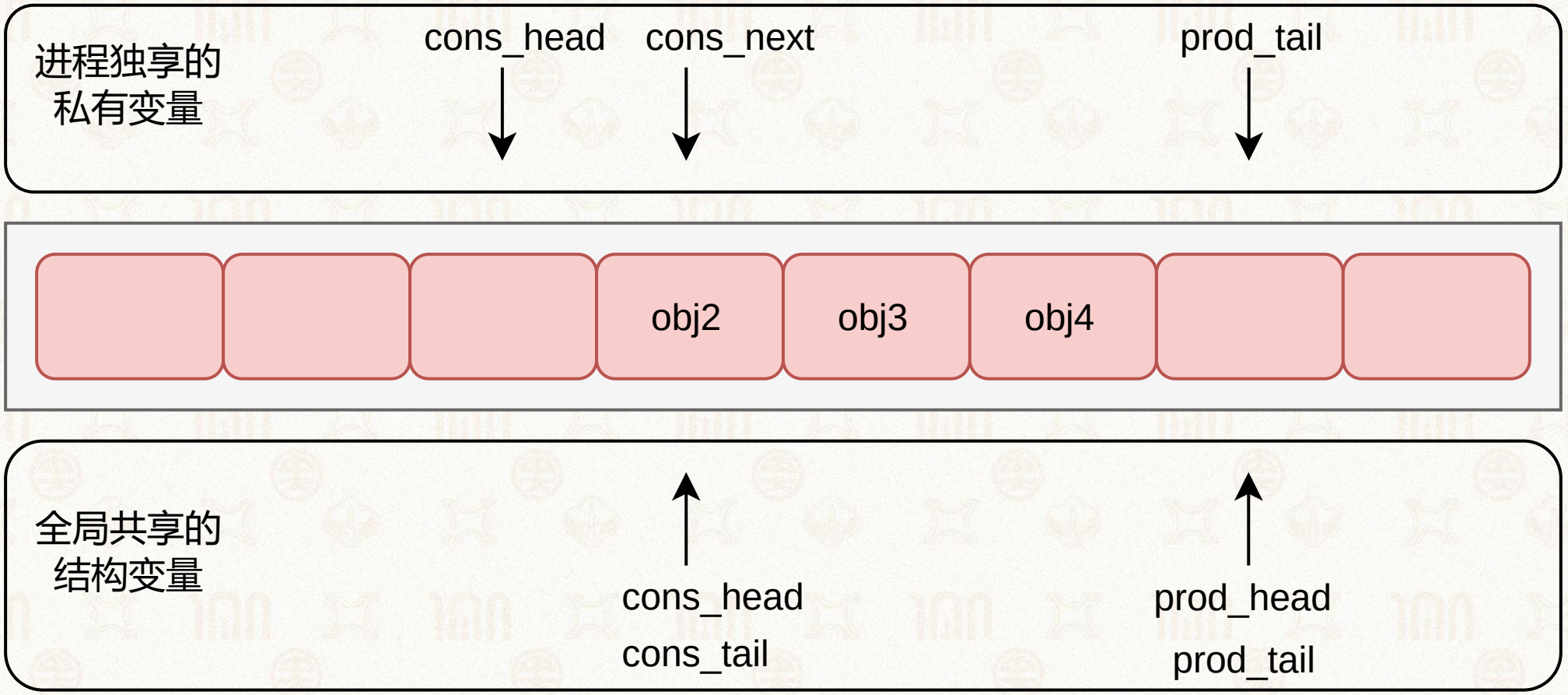
无锁环：消费者出队

- 从私有变量cons_head处移出数据



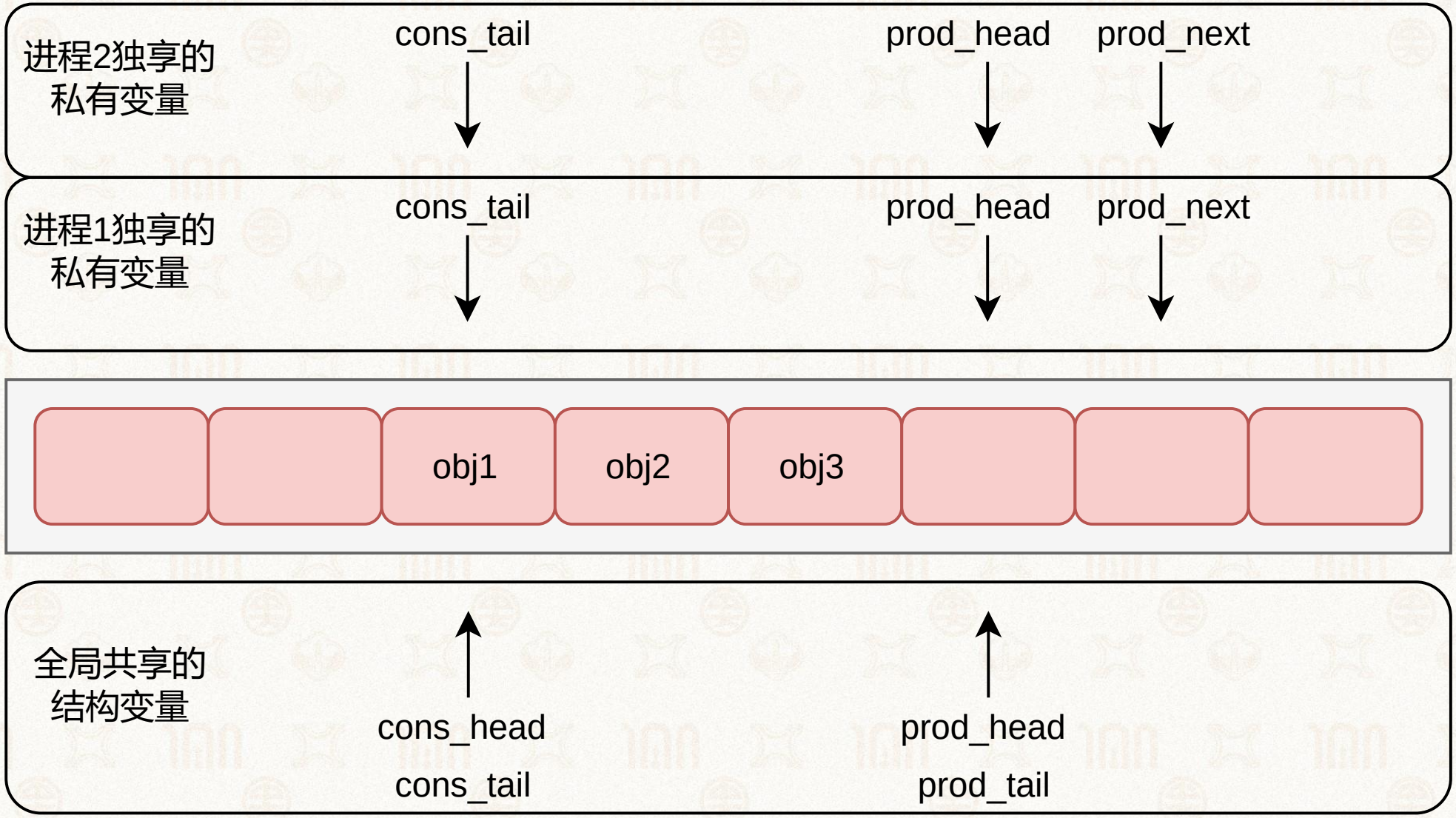
无锁环：消费者出队

- 最后，共享变量cons_tail移向共享变量cons_head



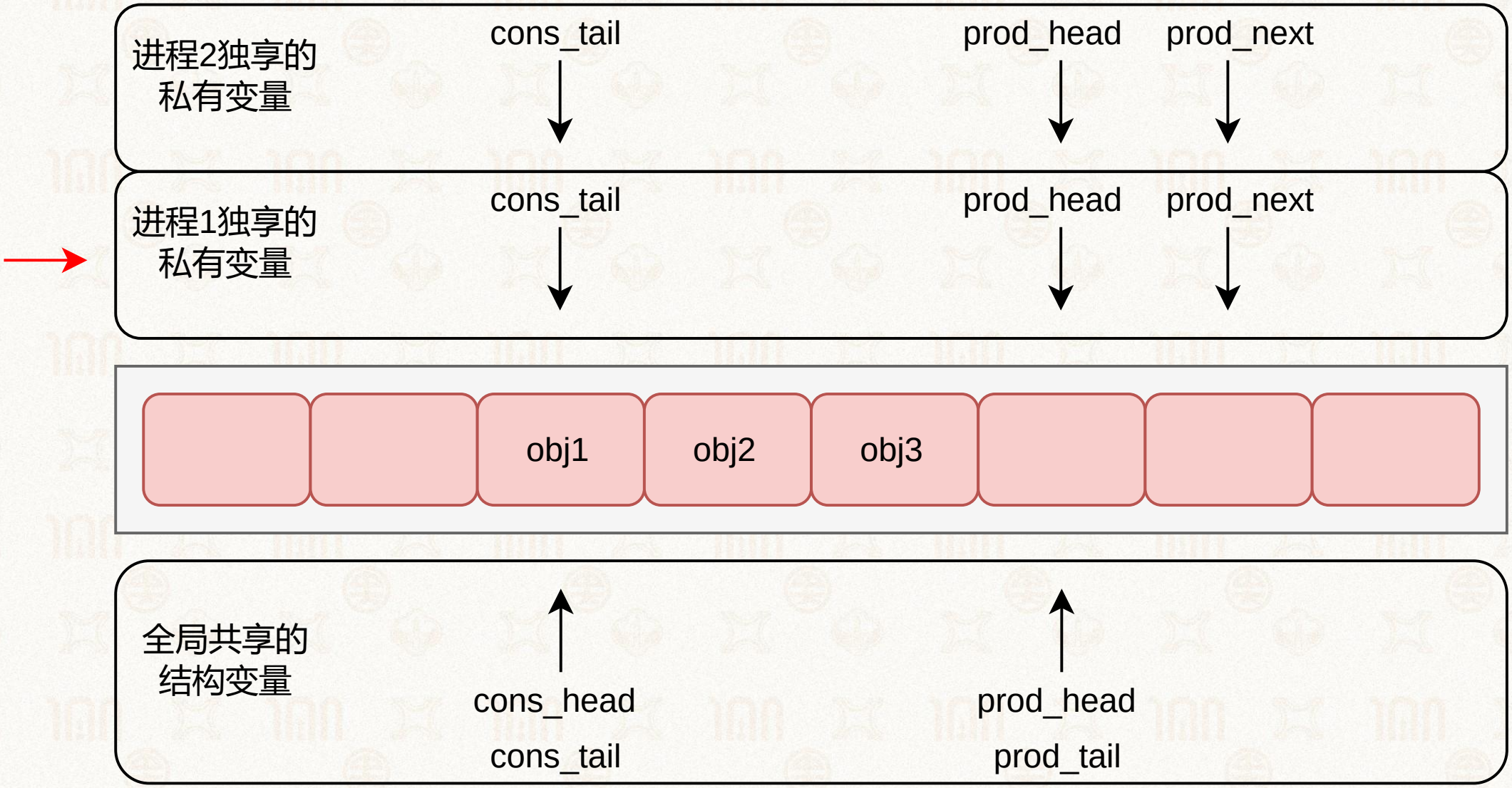
无锁环：多生产者入队

- 第一步：复制私有变量，再检查是否有多余空间



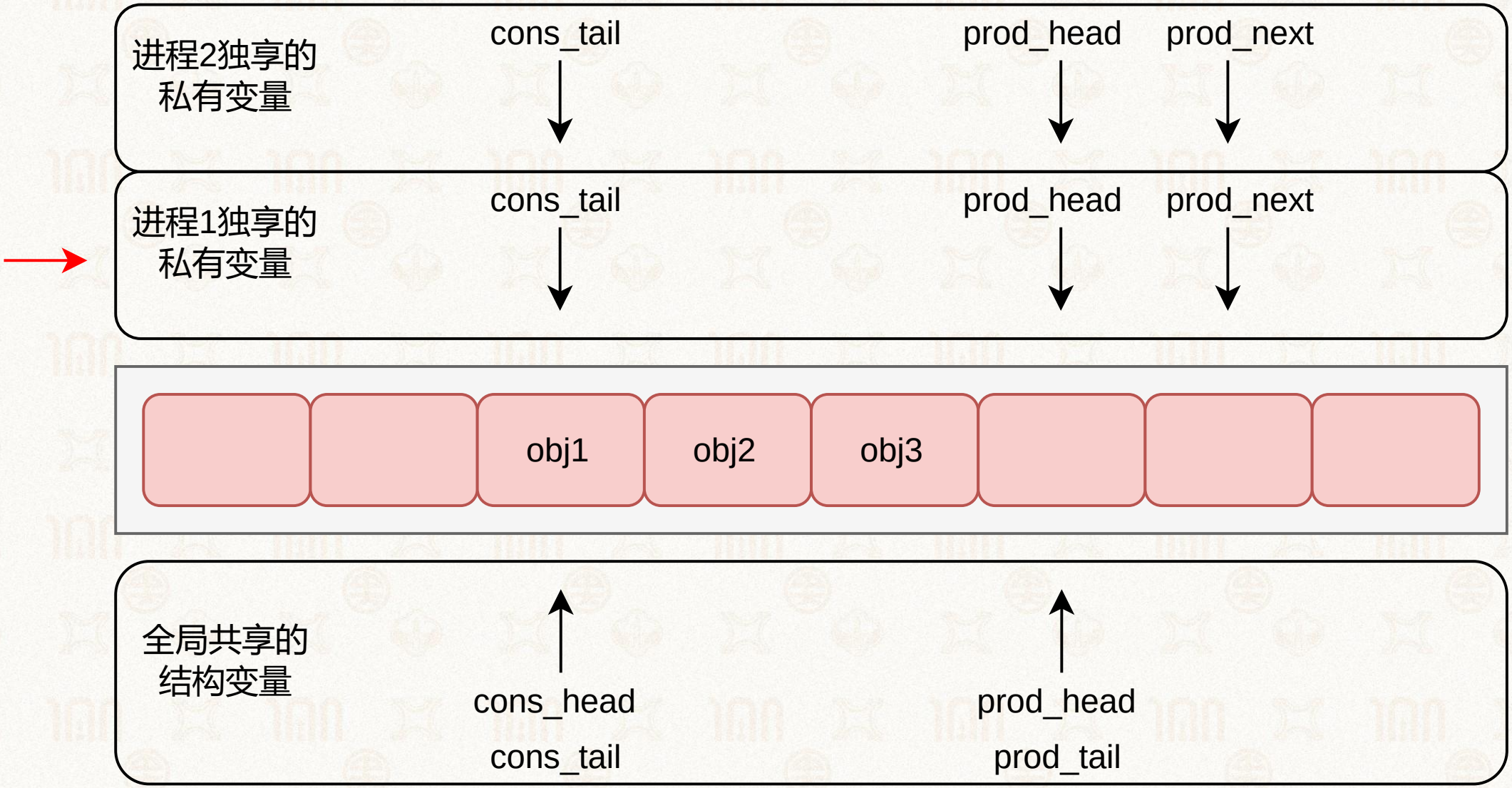
无锁环：多生产者入队

- 用CAS原子操作移动共享变量prod_head到私有变量prod_next



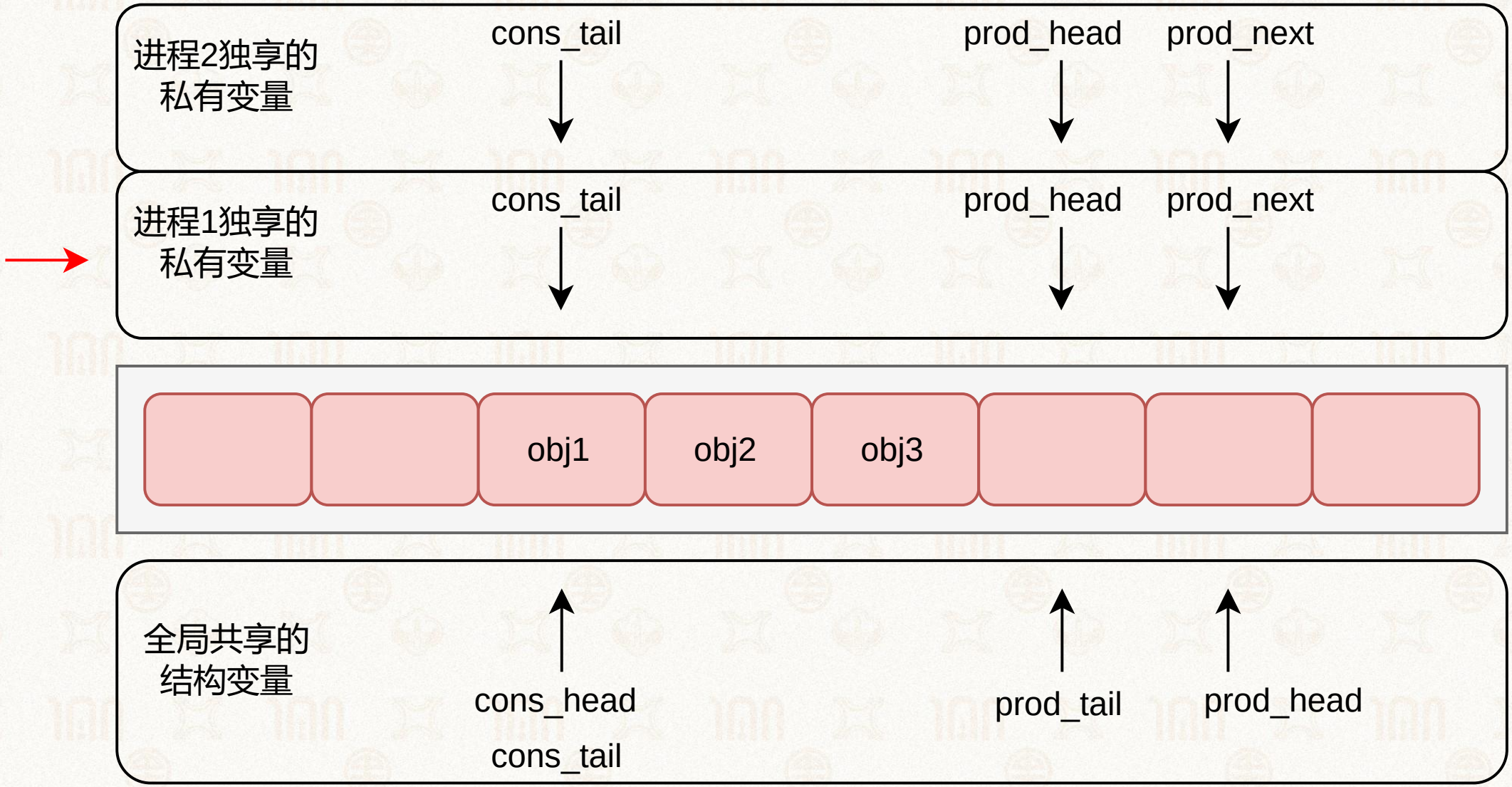
无锁环：多生产者入队

- CAS:如果共享prod_head不等于私有prod_head，失败，从第一步开始



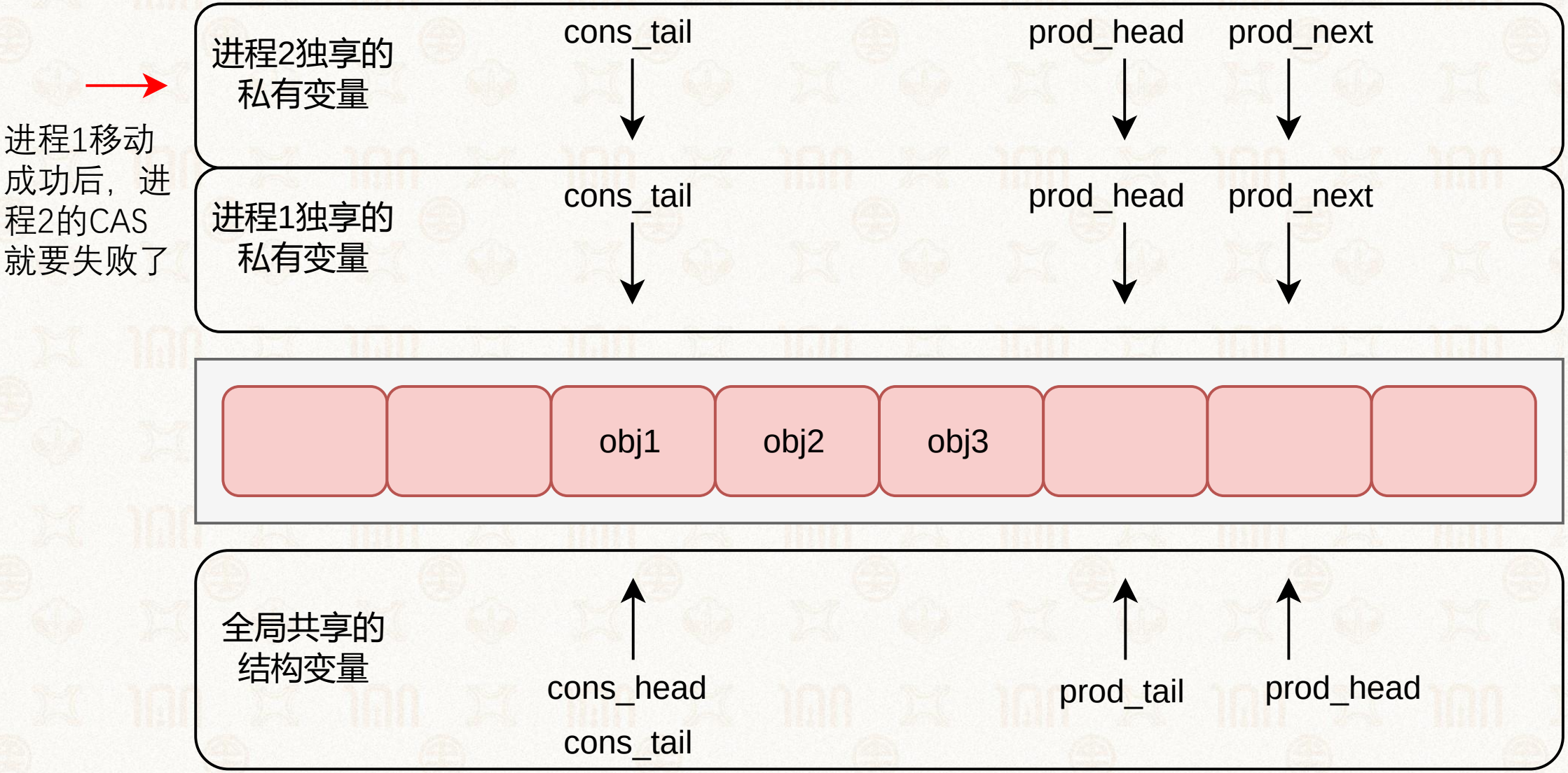
无锁环：多生产者入队

- CAS:如果共享prod_head等于私有prod_head，共享prod_head移到私有prod_next



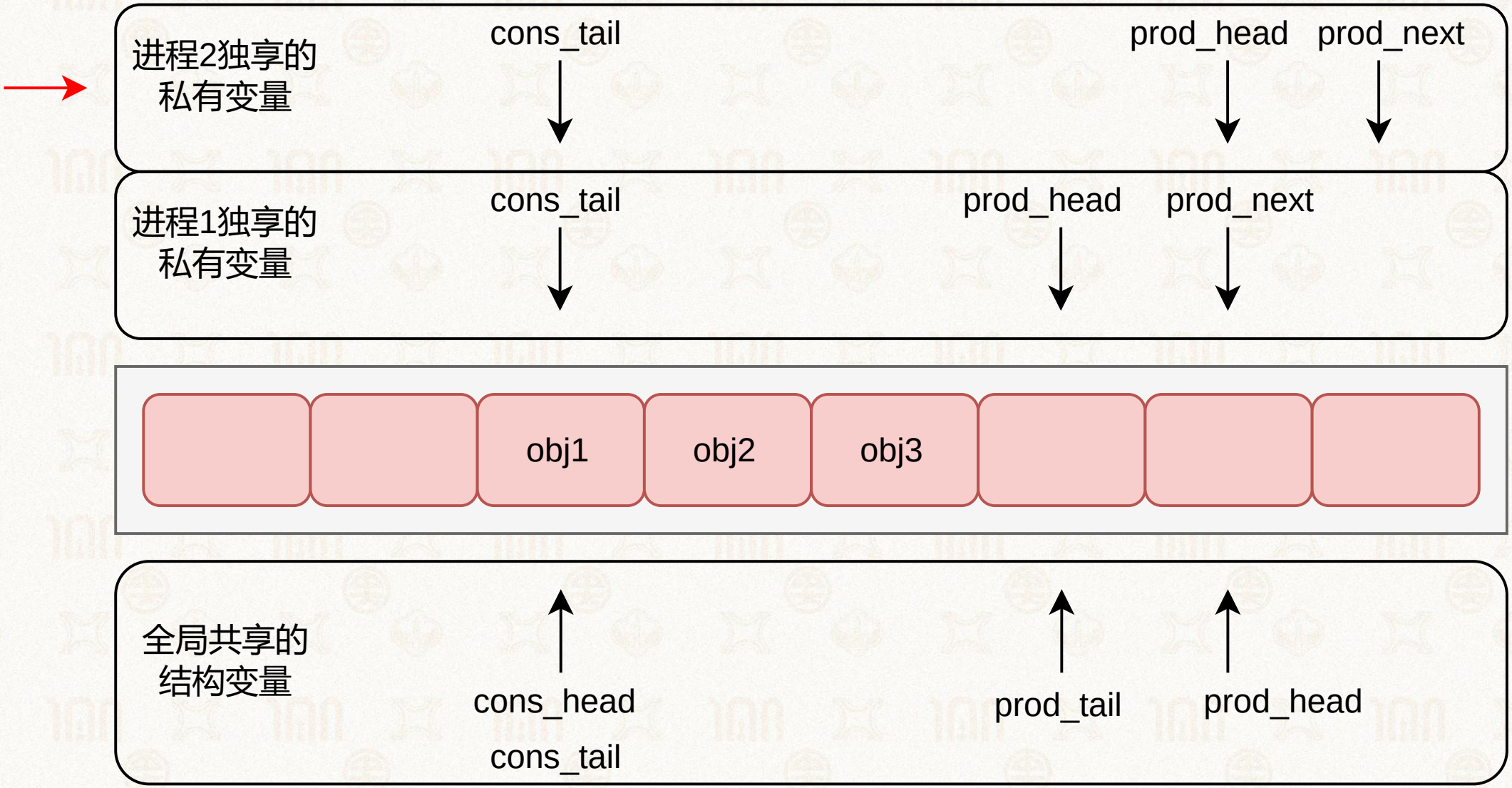
无锁环：多生产者入队

- CAS:如果共享prod_head不等于私有prod_head，失败，从第一步开始



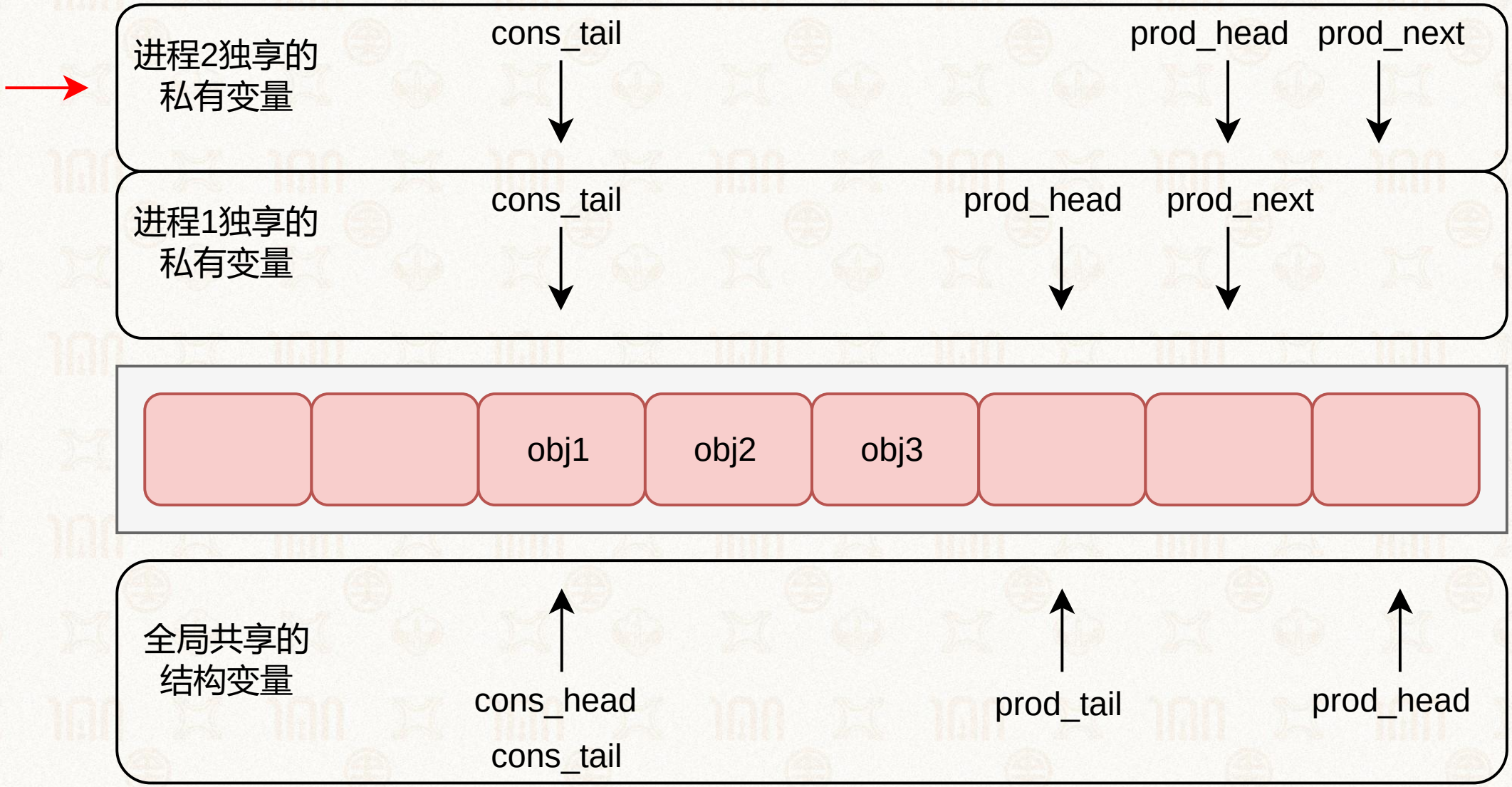
无锁环：多生产者入队

- 进程2从第一步开始：复制私有变量



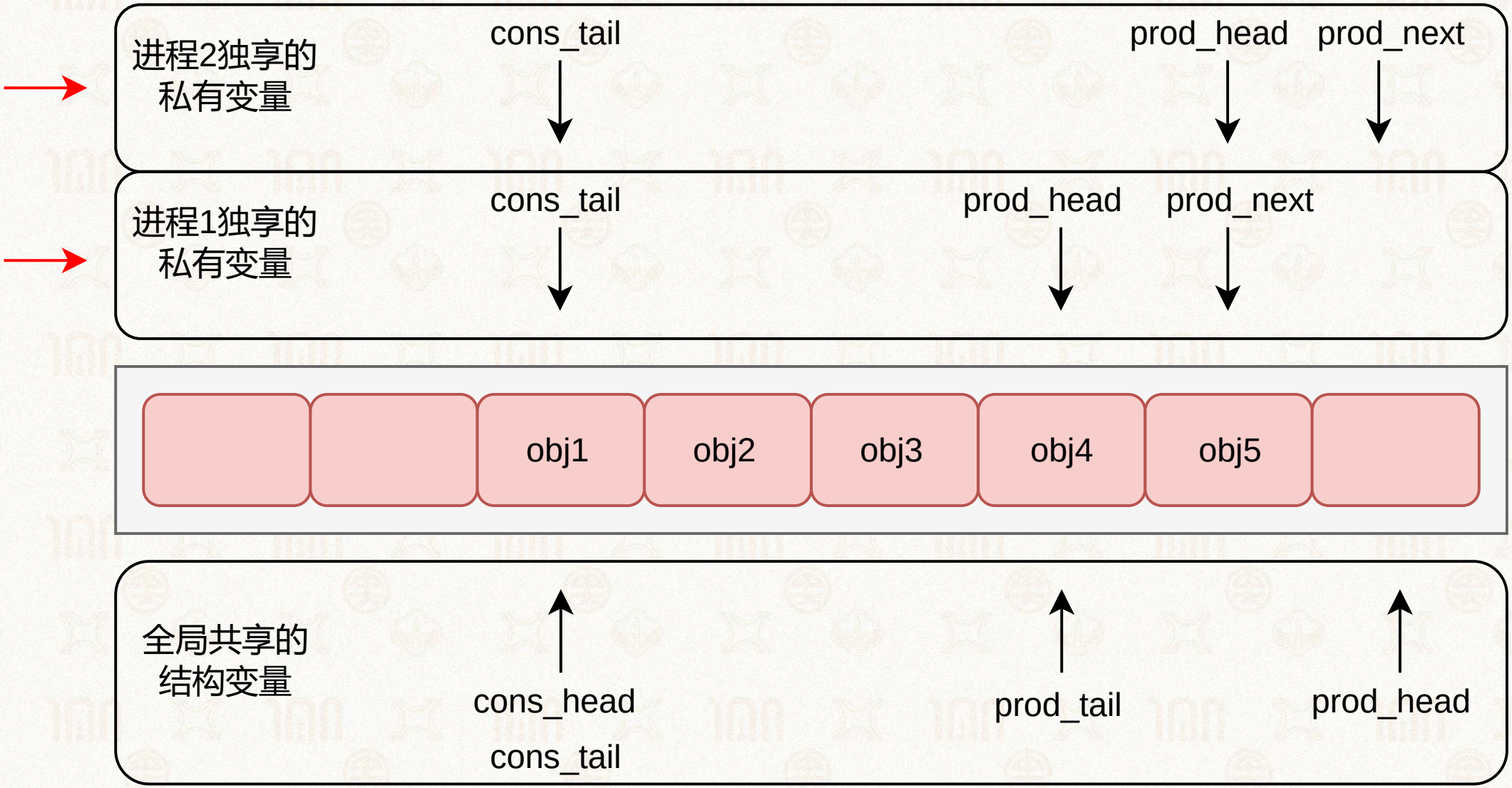
无锁环：多生产者入队

- CAS:如果共享prod_head等于私有prod_head，共享prod_head移到私有prod_next



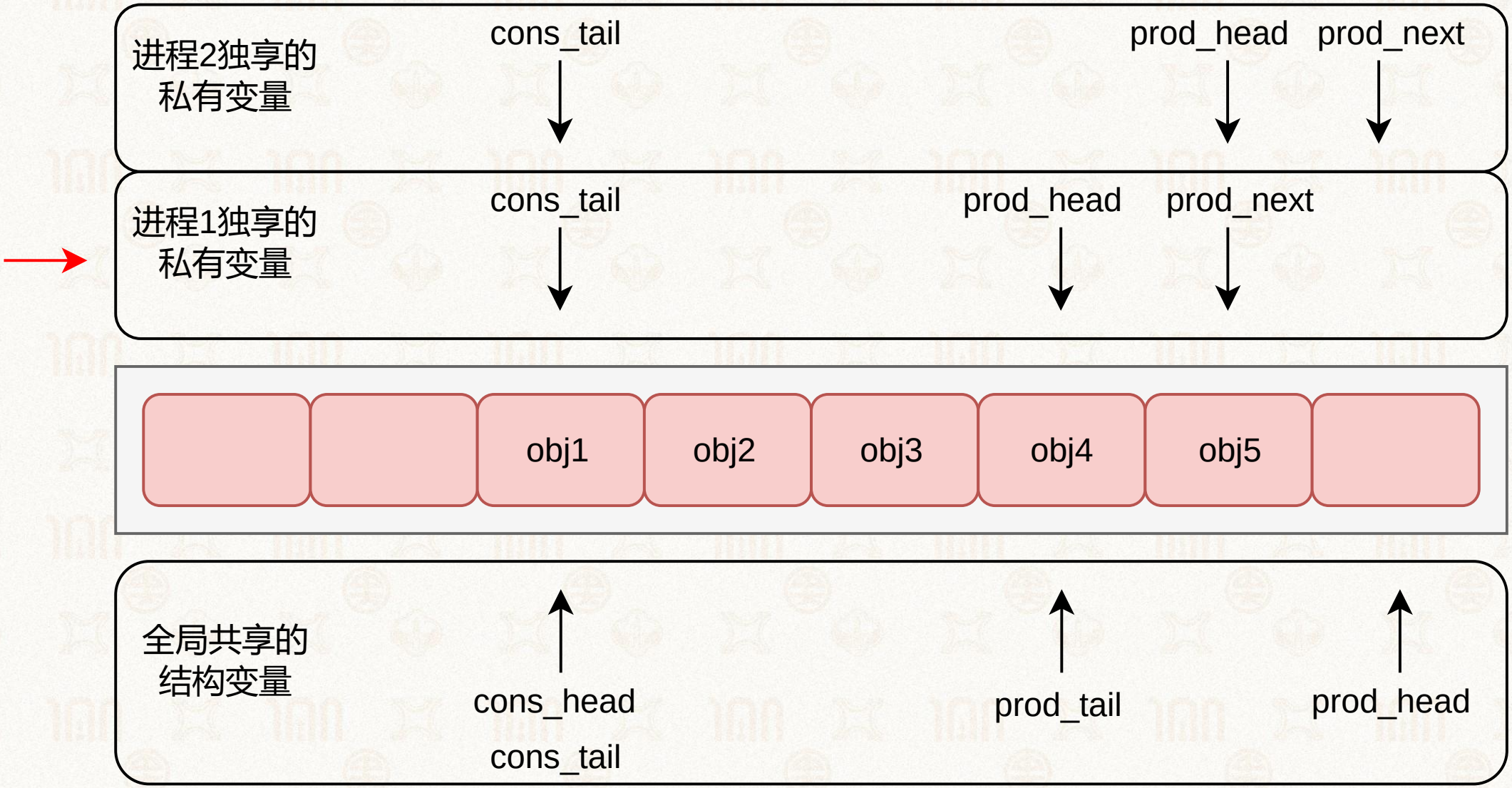
无锁环：多生产者入队

- 此时进程1&2都可以安全地向私有prod_head处写入数据



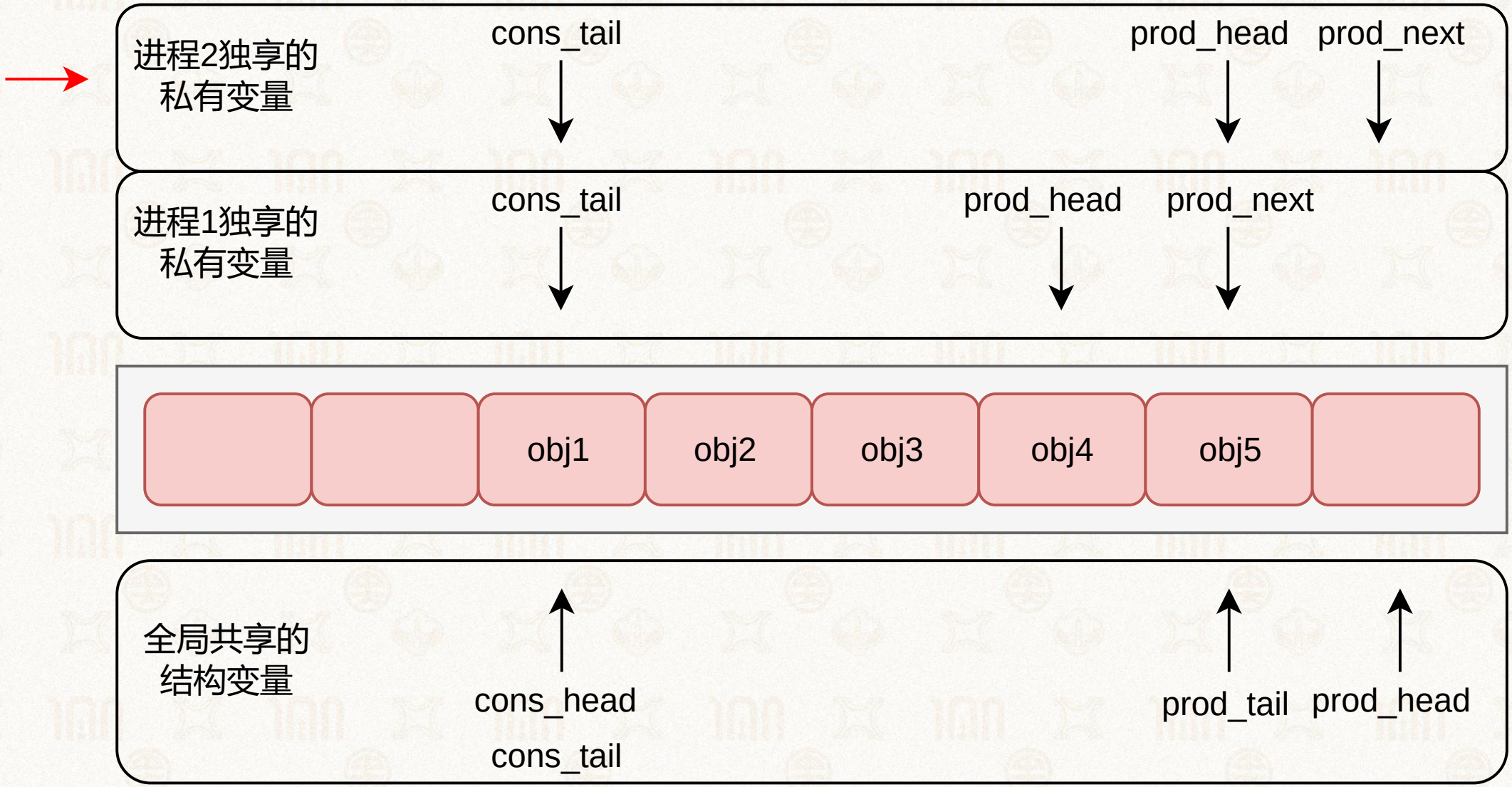
无锁环：多生产者入队

- 最后：只有共享prod_tail等于私有prod_head时才可以更新共享prod_tail



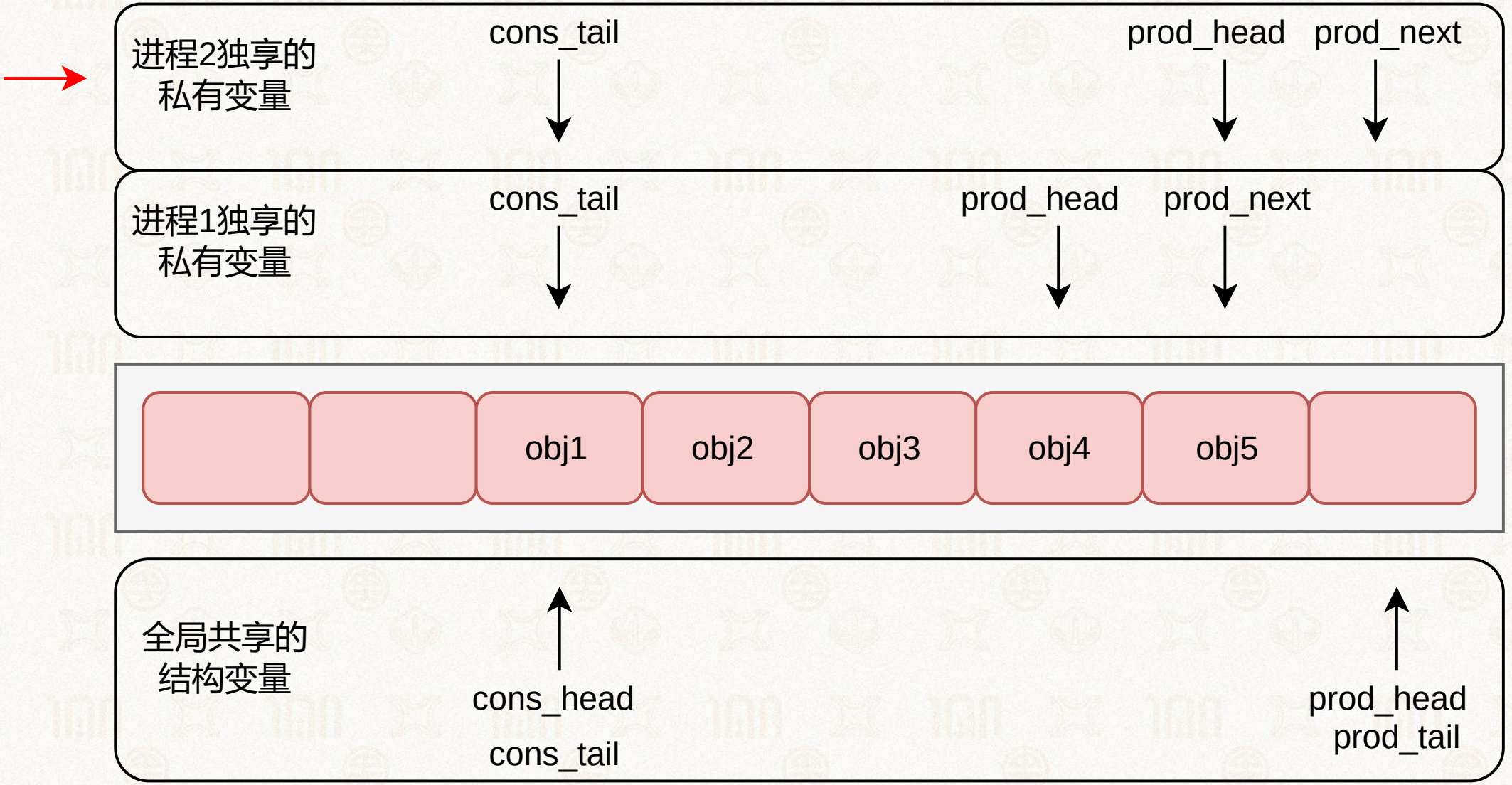
无锁环：多生产者入队

- 最后：进程1成功更新之后，进程2也有机会更新



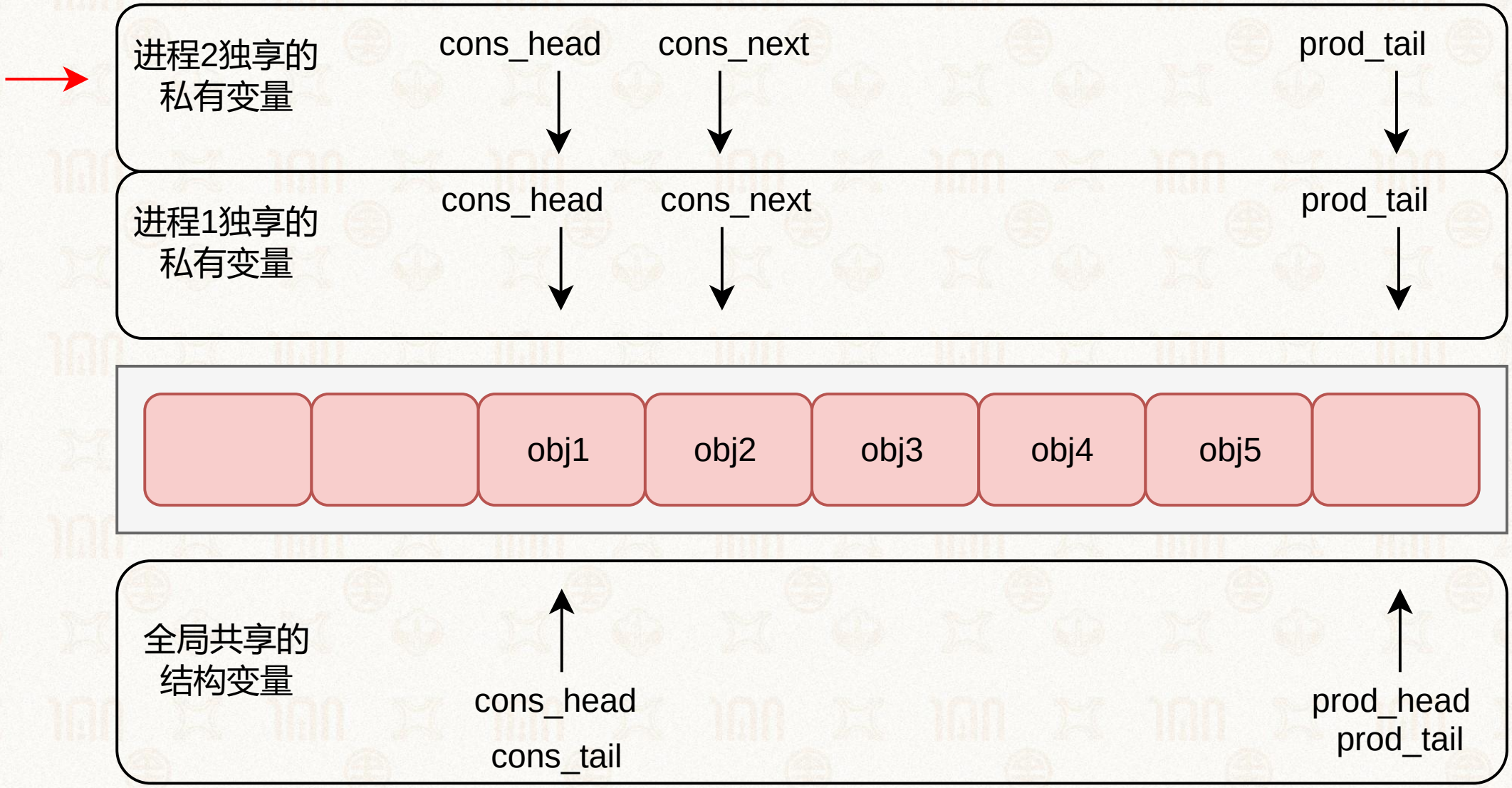
无锁环：多生产者入队

- 最后：进程2也有机会更新



无锁环：多消费者出队

- 与多消费者入队类似，不再重复。关键靠原子操作更新指针以实现无锁

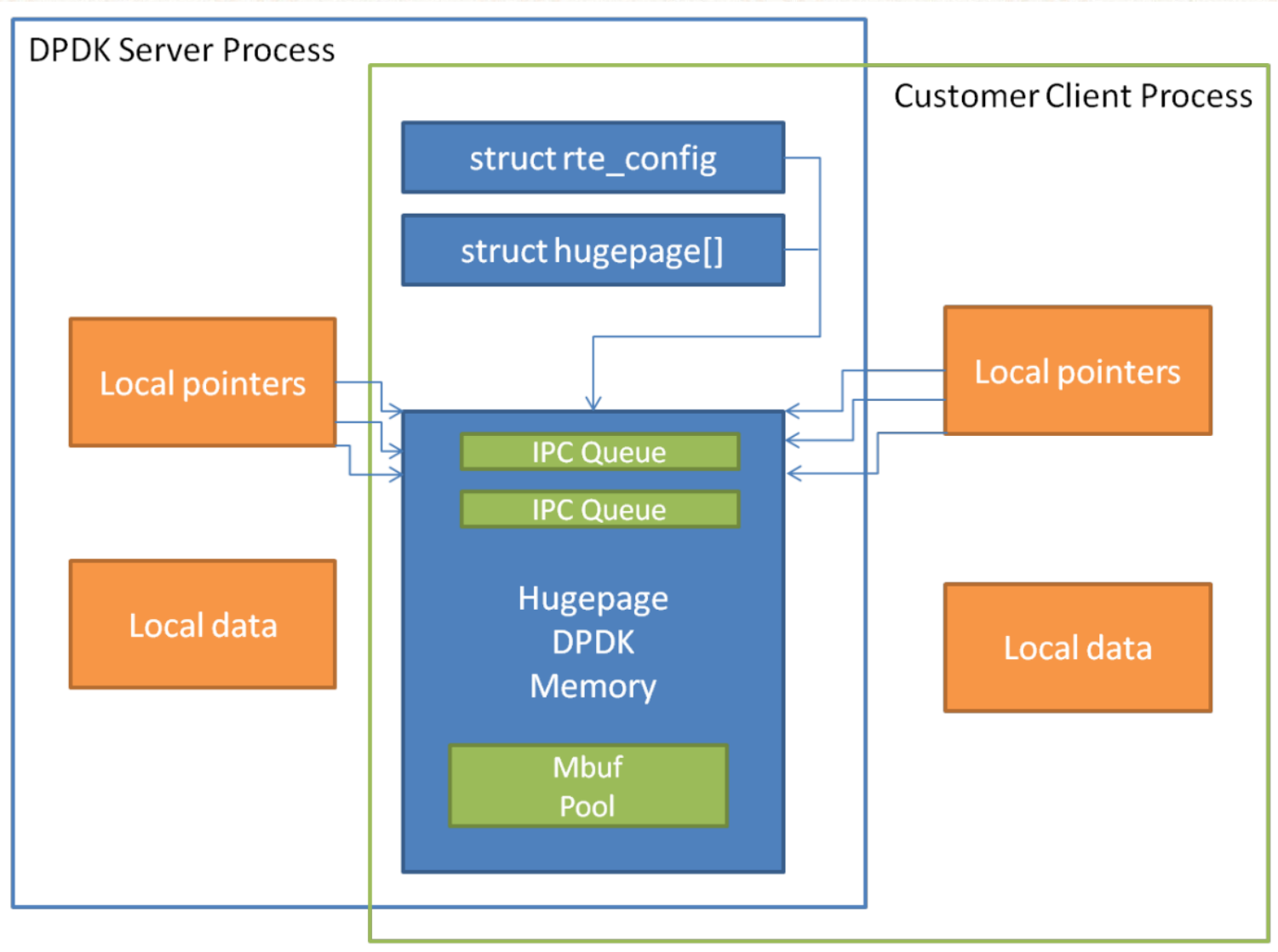


无锁环

- 参考实现：
  - buf_ring.h (https://svnweb.freebsd.org/base/release/8.0.0/sys/sys/buf_ring.h?revision=199625&view=markup)
  - subr_bufring.c (https://svnweb.freebsd.org/base/release/8.0.0/sys/kern/subr_bufring.c?revision=199625&view=markup)
  - Linux内核的实现思路
    - https://lwn.net/Articles/340400/

多进程环境下的内存共享

- 私有指针指向资源，并不拥有资源
- 还记得死锁预防里的“代理执行”么？



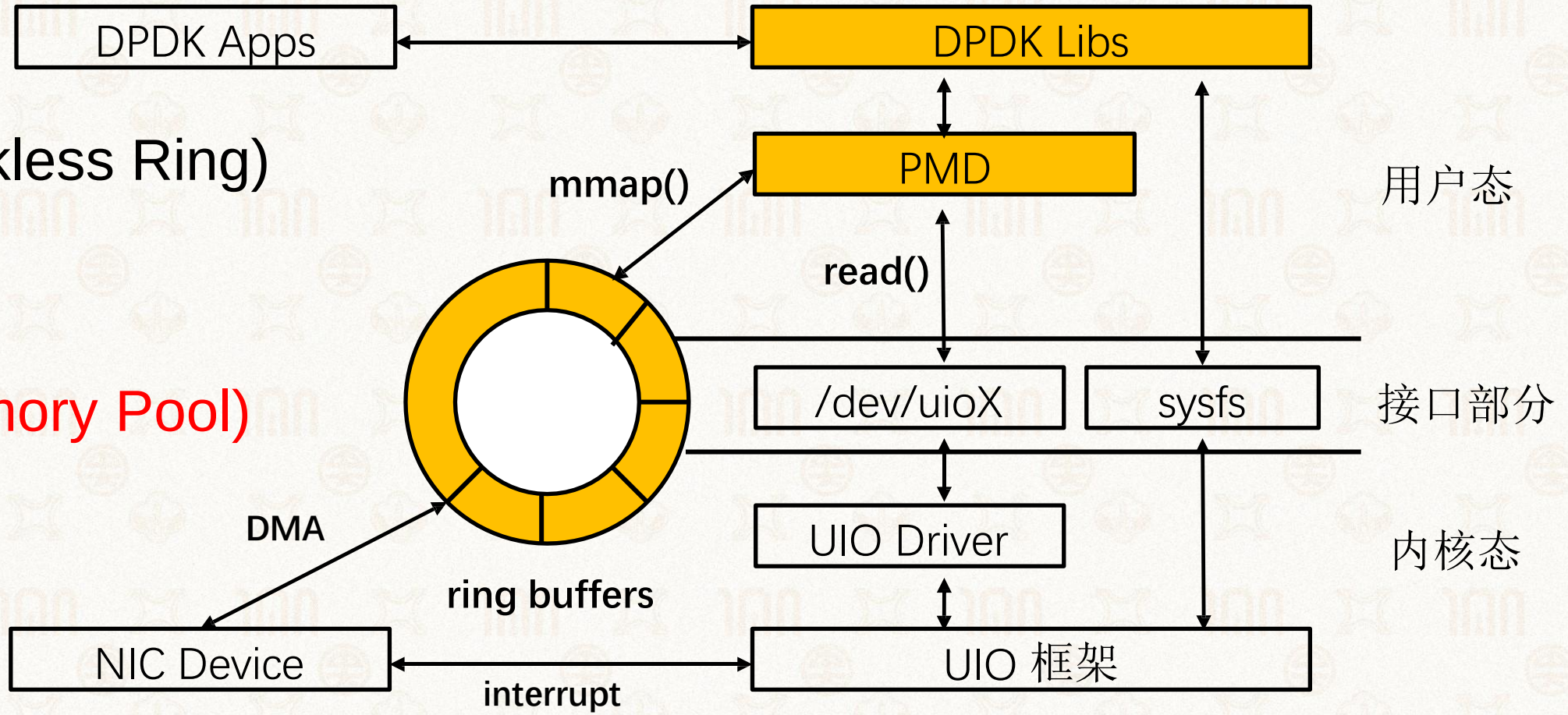
大纲

- 越俎代庖部分
  - 网络协议的分层模型
  - 套接字模型
- 网络驱动模型
- Linux系统收包过程
  - 函数视角
  - 数据视角
- Linux系统发包过程
- 网络处理性能优化
  - 挑战
  - 数据面控制面分离
- Intel DPDK 软件优化方案
  - 总体框架
  - 无锁环
  - 内存池
  - 其它模块
  - 扩展框架

  （无锁环、内存池）和操作系统课程最紧密的模块
- 硬件优化方案

深入学习DPDK性能优化思想

- 无锁环(Lockless Ring)
- 内存池(Memory Pool)



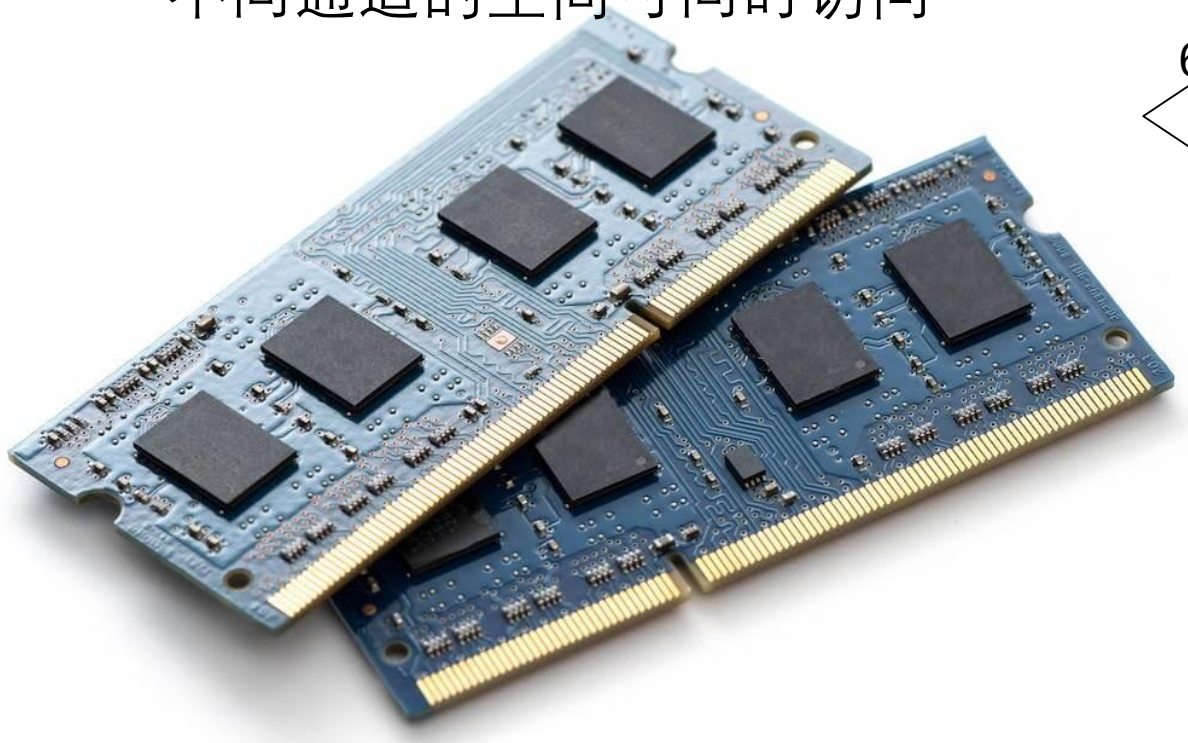
内存池(MemPool)

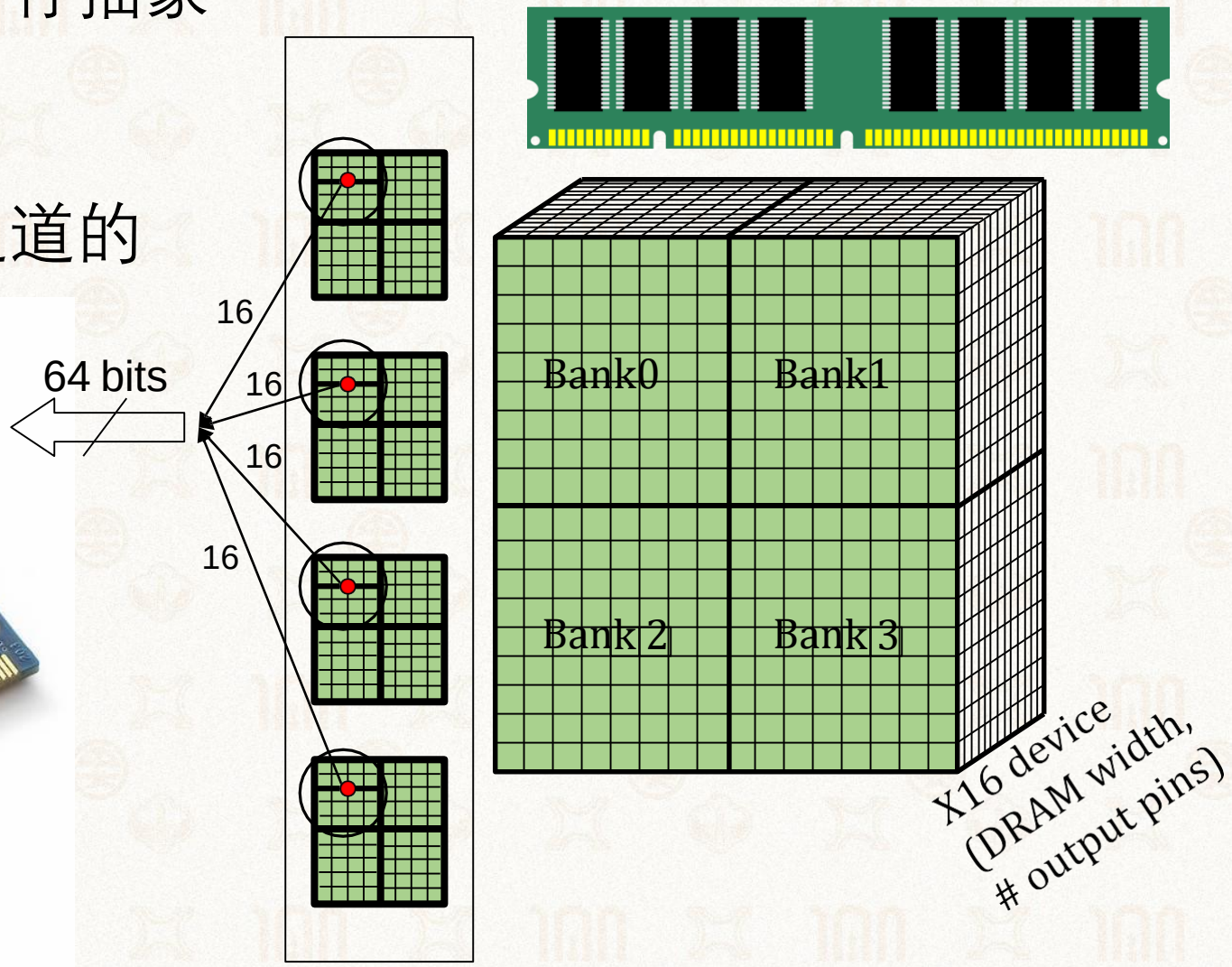
- 用于分配对象在内存中的存储空间
- 由名字与无锁环共同代表对象存储空间
- 为提高读写速率，使用了
  - 内存对齐
  - 私有缓存

http://doc.dpdk.org/guides/prog_guide/mempool_lib.html#

内存控制器(Memory Controller)

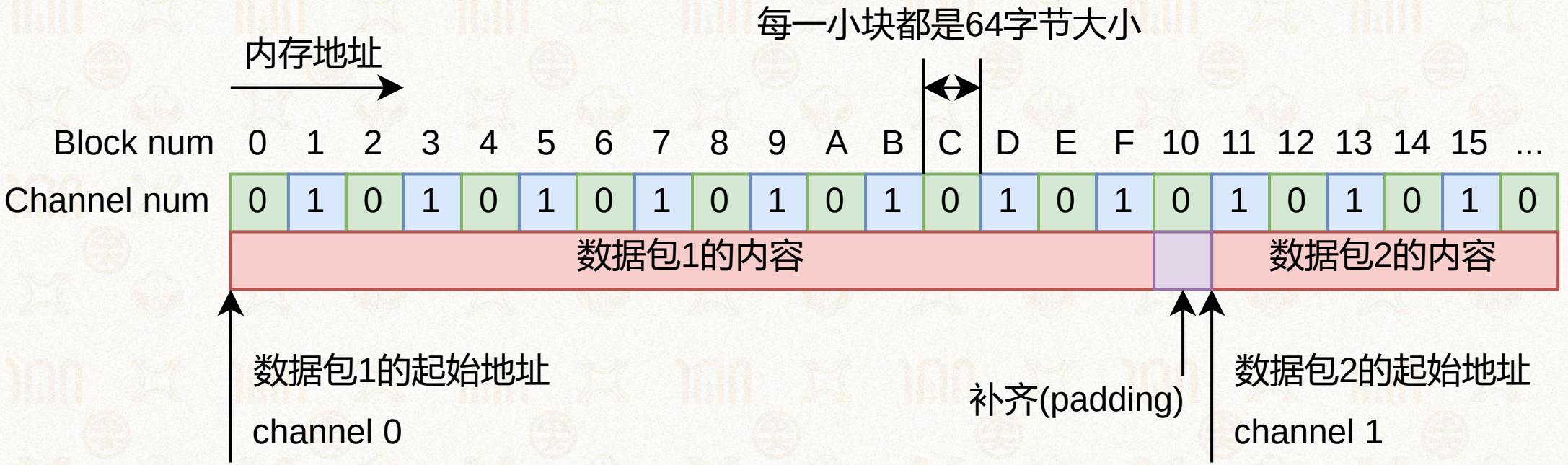
- 为操作系统提供了易用的物理内存抽象
  - 逐字节可寻址的“大数组”
  - 屏蔽了硬件细节
- 细节不能忽视：内存条是分多通道的
  - 不同通道的空间可同时访问





内存池(MemPool)：内存对齐

- 内存空间也需要对齐
  - 同一个通道(Channel)内的数据不能被同时获取
  - 因为连接着相同的数据总线



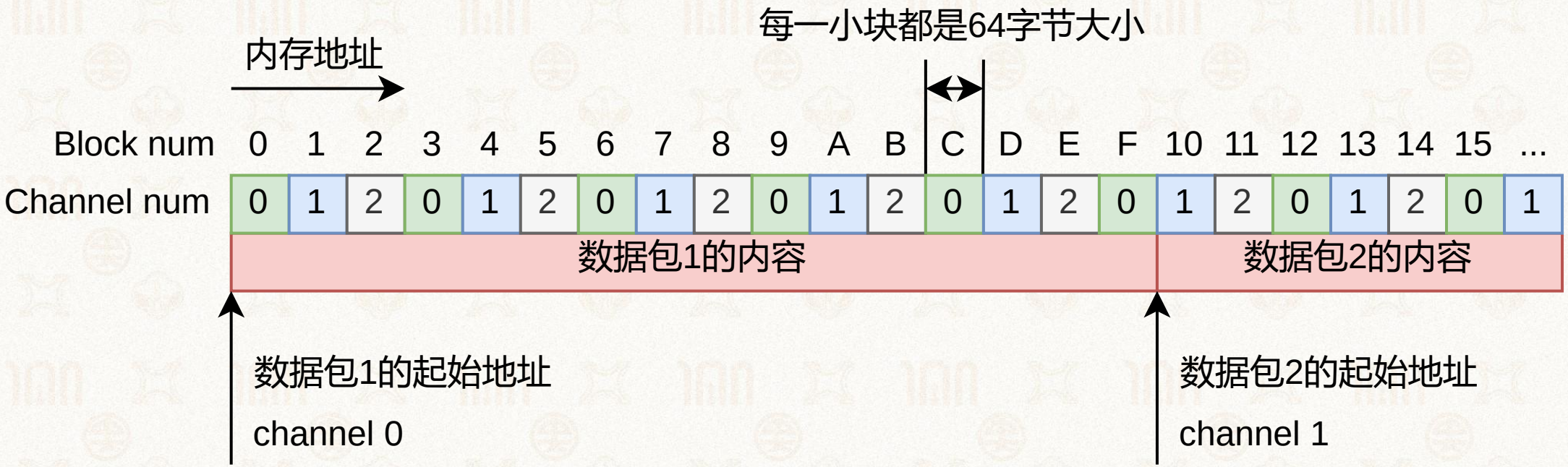
这是双通道的内存条布局

http://doc.dpdk.org/guides/prog_guide/mempool_lib.html

p.s. 家用电脑的内存条一般都是双通道

内存池(MemPool)： 内存对齐

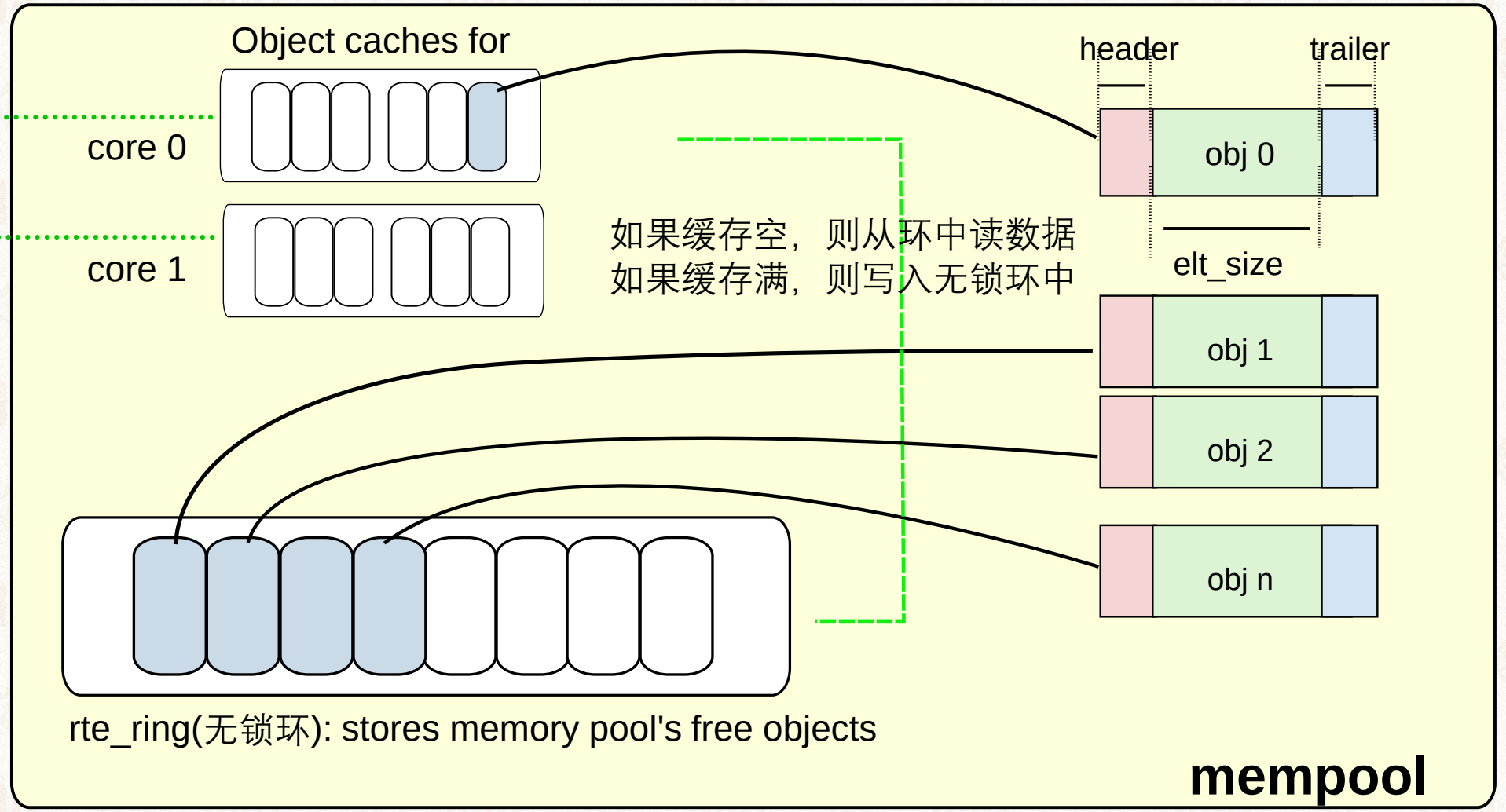
- 内存空间也需要对齐
  - 同一个通道(Channel)内的数据不能被同时获取
  - 因为连接着相同的数据总线



这是三通道的内存条布局，不需要补齐

http://doc.dpdk.org/guides/prog_guide/mempool_lib.html

内存池(MemPool)：私有缓存



减少原子操作的执行次数，提升多核的运行效率

毕竟原子操作会阻塞总线

http://doc.dpdk.org/guides/prog_guide/mempool_lib.html

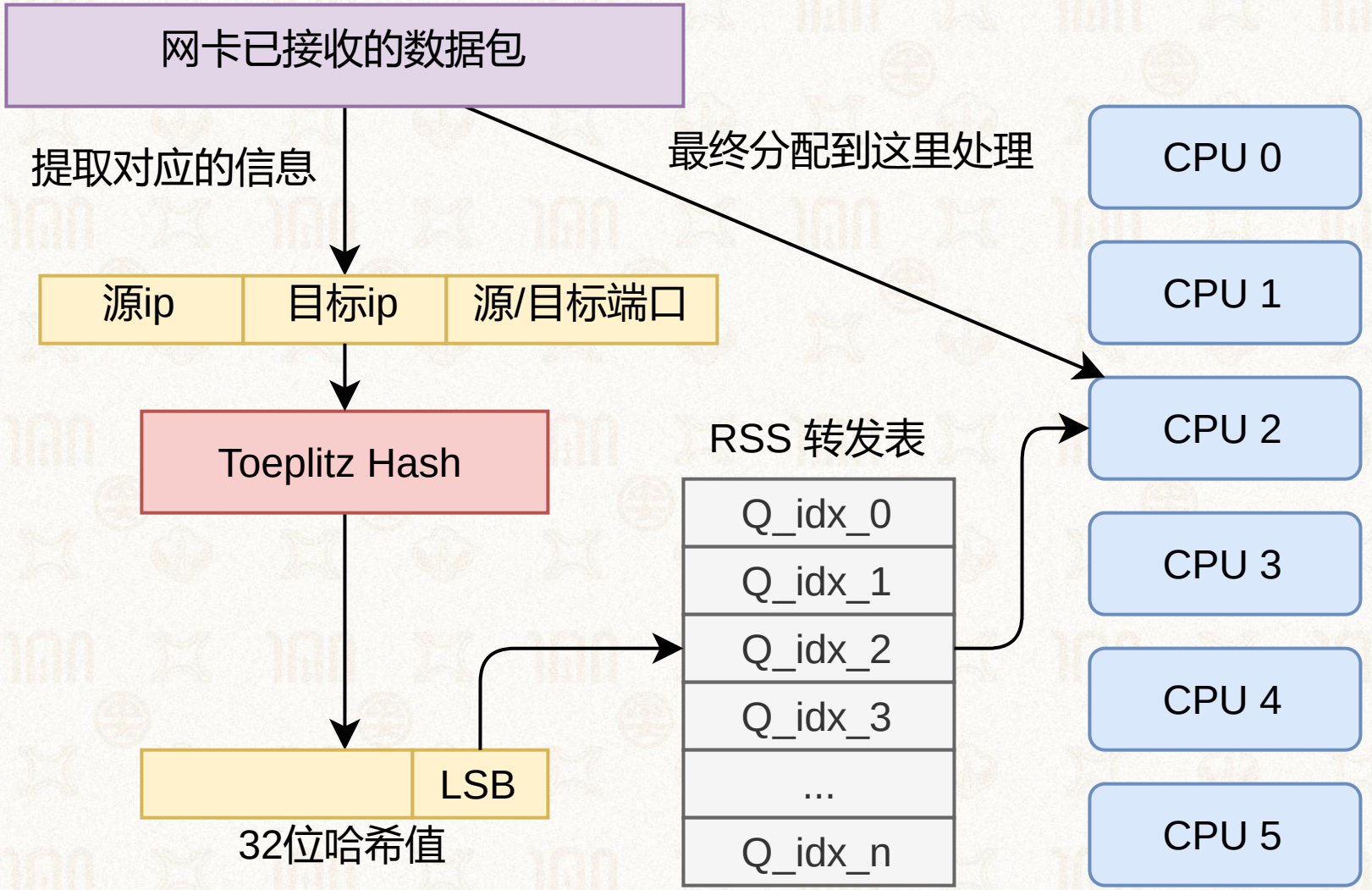
大纲

- 越俎代庖部分
  - 网络协议的分层模型
  - 套接字模型
- 网络驱动模型
- Linux系统收包过程
  - 函数视角
  - 数据视角
- Linux系统发包过程
- 网络处理性能优化
  - 挑战
  - 数据面控制面分离
- Intel DPDK 软件优化方案
  - 总体框架
  - 无锁环
  - 内存池
  - 其它模块
  - 扩展框架

  （无锁环、内存池）和操作系统课程最紧密的模块
- 硬件优化方案

DPDK里其它宝藏：托普利兹哈希(Toeplitz Hash)

- 用于接收端数据包处理的负载均衡



http://doc.dpdk.org/guides/prog_guide/toeplitz_hash_lib.html

DPDK里其它宝藏

- 时钟管理(Timer)
  - 提供简洁精确的时间信息，以精准地时间间隔执行异步函数调用
- 报文转发
  - 精确匹配算法
  - 最长前缀匹配算法
  - ACL算法
- 流分类/过滤算法
  - 杜鹃(Cuckoo，布谷鸟)哈希

大纲

- 越俎代庖部分
  - 网络协议的分层模型
  - 套接字模型
- 网络驱动模型
- Linux系统收包过程
  - 函数视角
  - 数据视角
- Linux系统发包过程
- 网络处理性能优化
  - 挑战
  - 数据面控制面分离
- Intel DPDK 软件优化方案
  - 总体框架
  - 无锁环
  - 内存池
  - 其它模块
  - 扩展框架

  （无锁环、内存池）和操作系统课程最紧密的模块
- 硬件优化方案

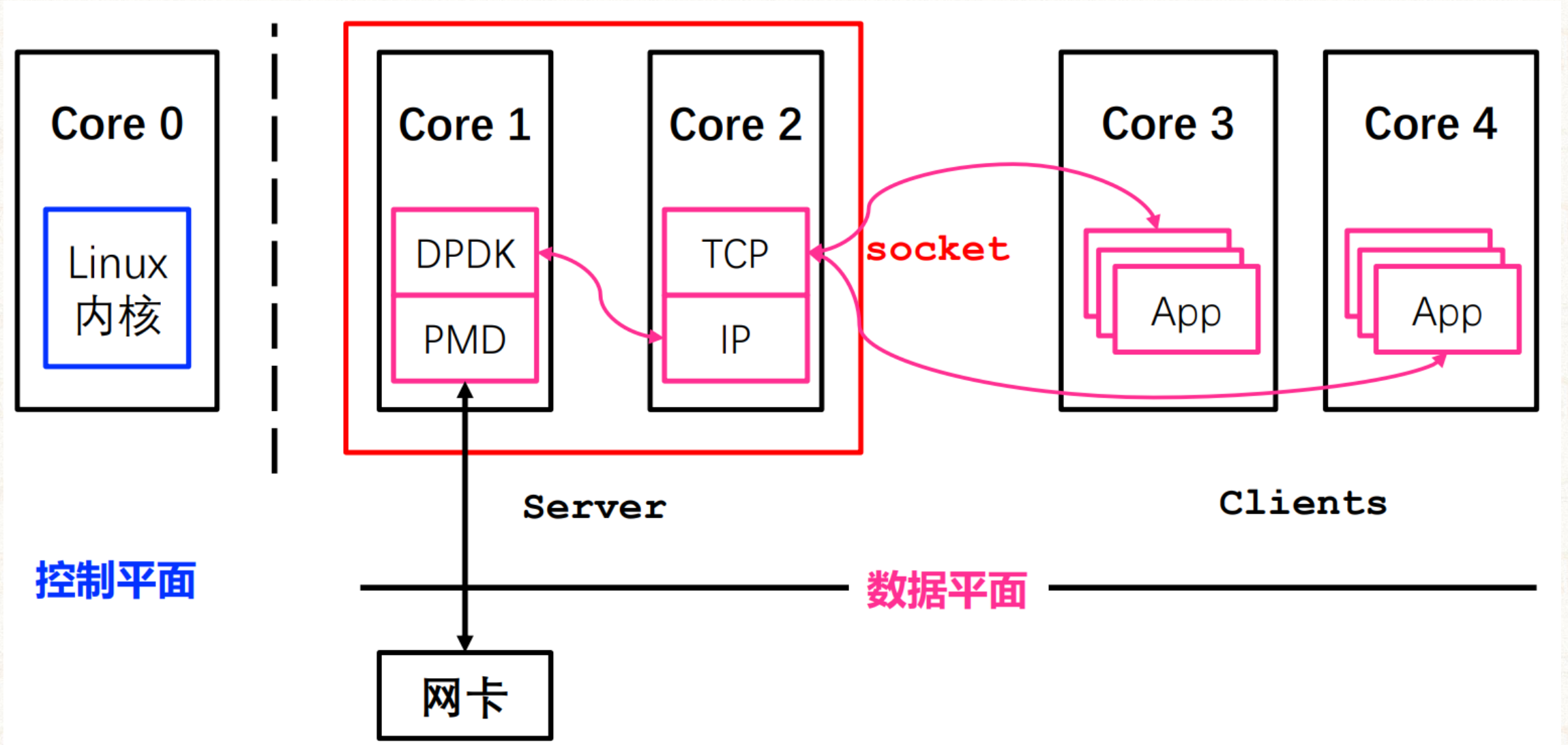
DPDK的使用特点

- DPDK本身只是2层协议，不提供socket接口
  - 适合于软路由场景

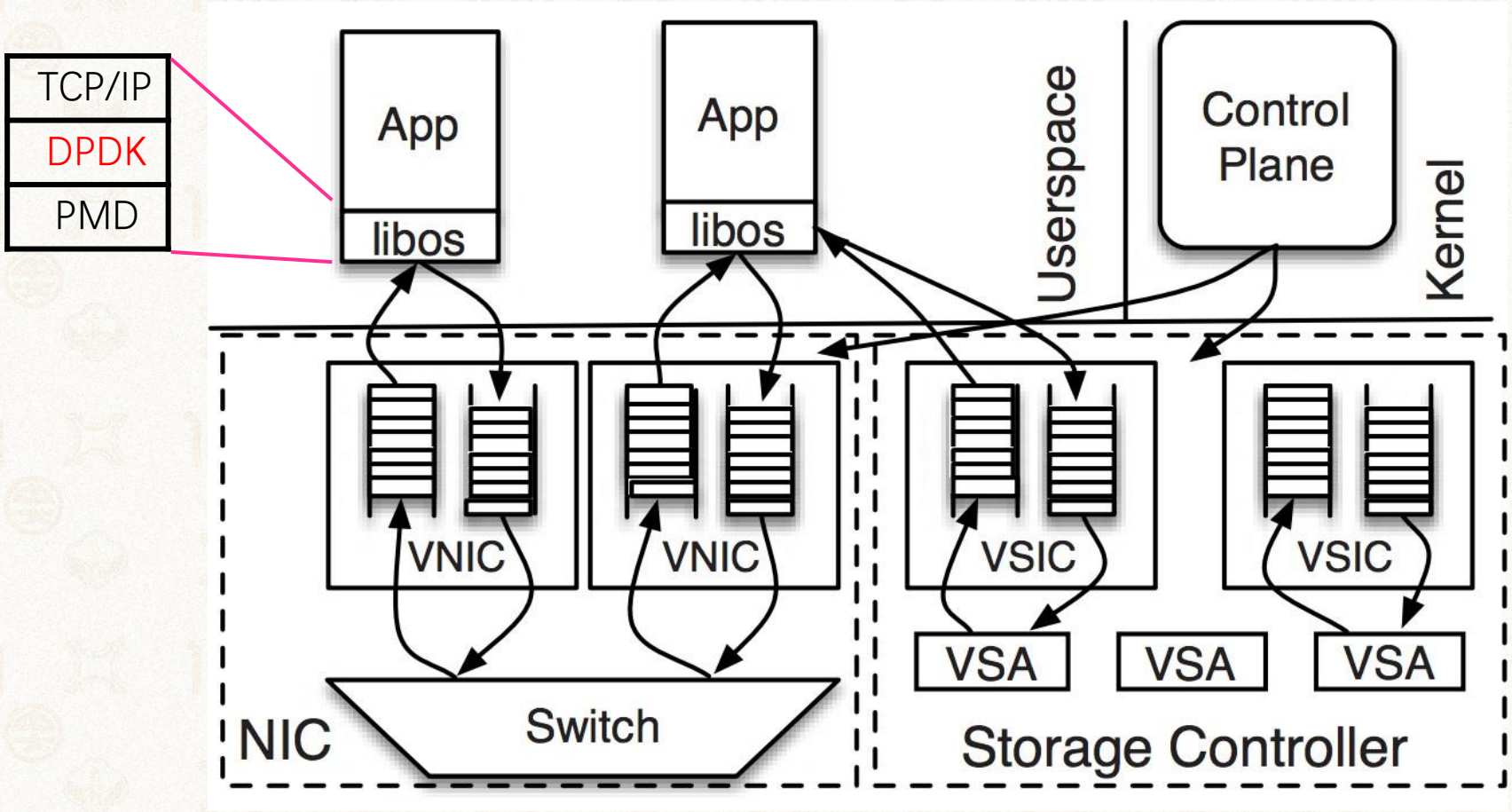
- 如果要用于应用程序，提供socket抽象
  - 类微内核方案：将DPDK+TCP/IP协议栈作为一个服务器，为每个应用单独维护 socket fd 和进程间的对应关系
  - LibOS方案：需要逐个应用添加协议栈支持

- 有更多基于DPDK开发的完整协议栈框架
  - 腾讯云的F-stack (http://f-stack.org/)
  - Linux基金会的Vector Packet Processor (https://s3-docs.fd.io/vpp/22.06/#)
  - mTCP (https://github.com/mtcp-stack/mtcp)

类微内核方案


Core 0
Linux
内核
Core 1
DPDK
PMD
Core 2
TCP
IP
socket
Core 3
App
Core 4
App
Server
Clients
控制平面
数据平面
网卡

LibOS方案



Arrakis: The Operating System Is the Control Plane. ACM Trans. Comput. Syst. 33, 4, Article 11 (January 2016), 30 pages. https://doi.org/10.1145/2812806

大纲

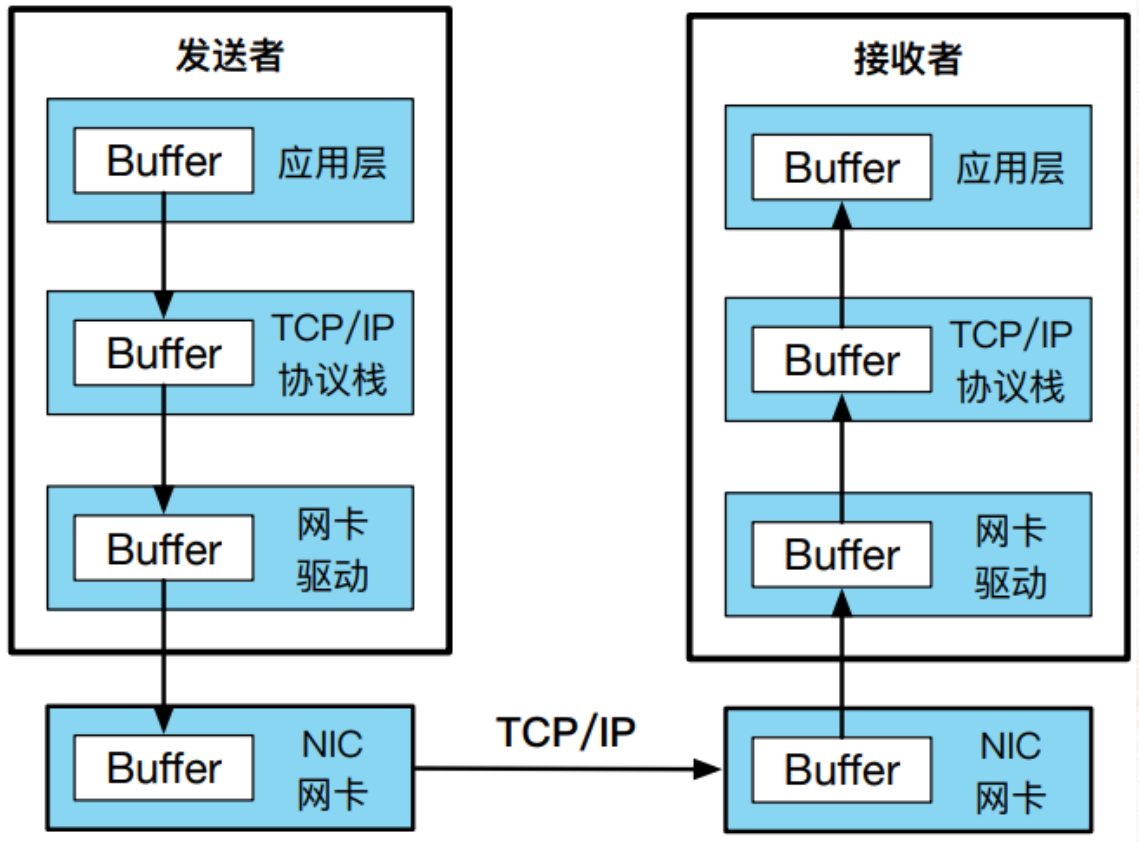
- 越俎代庖部分
  - 网络协议的分层模型
  - 套接字模型
- 网络驱动模型
- Linux系统收包过程
  - 函数视角
  - 数据视角
- Linux系统发包过程
- 网络处理性能优化
  - 挑战
  - 数据面控制面分离
- Intel DPDK 软件优化方案
  - 总体框架
  - 无锁环
  - 内存池
  - 其它模块
  - 扩展框架

（无锁环、内存池）} 和操作系统课程最紧密的模块

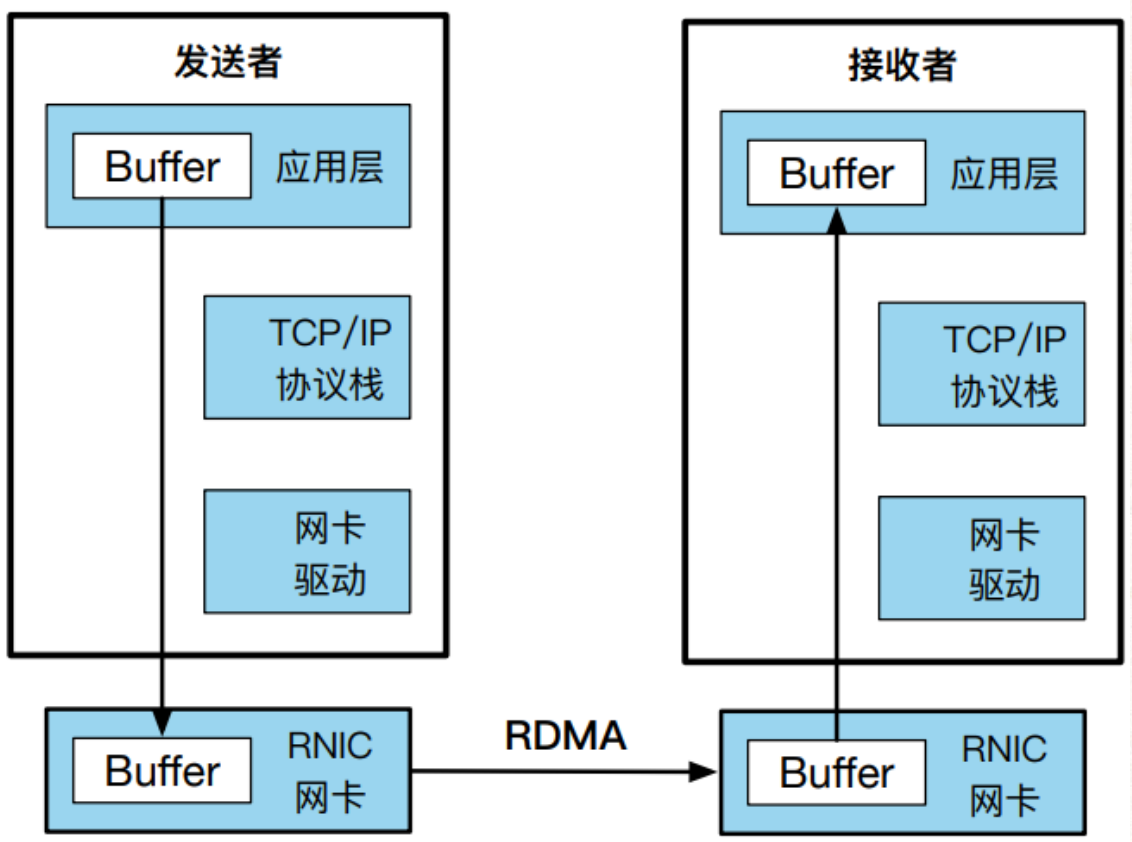
- 硬件优化方案

硬件加速方案：RDMA

- Remote Direct Memory Access (RDMA)
- 在远端无感的情况下，直接读取远端内存数据，绕开内核协议栈



传统网络数据传输方式



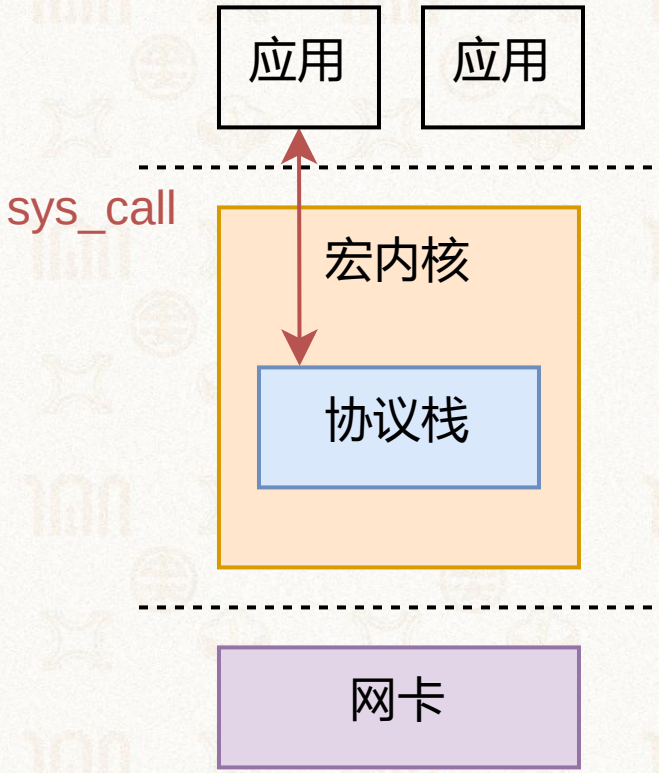
RDMA网络数据传输方式

大纲

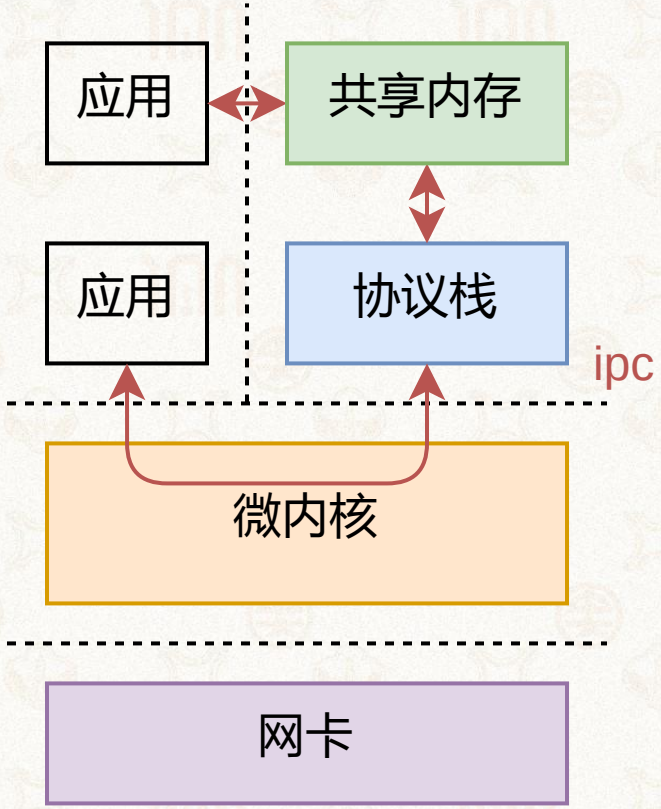
- 越俎代庖部分
  - 网络协议的分层模型
  - 套接字模型
- 网络驱动模型
- Linux系统收包过程
  - 函数视角
  - 数据视角
- Linux系统发包过程
- 网络处理性能优化
  - 挑战
  - 数据面控制面分离
- Intel DPDK 软件优化方案
  - 总体框架
  - 无锁环
  - 内存池
  - 其它模块
  - 扩展框架

  （无锁环、内存池）和操作系统课程最紧密的模块
- 硬件优化方案

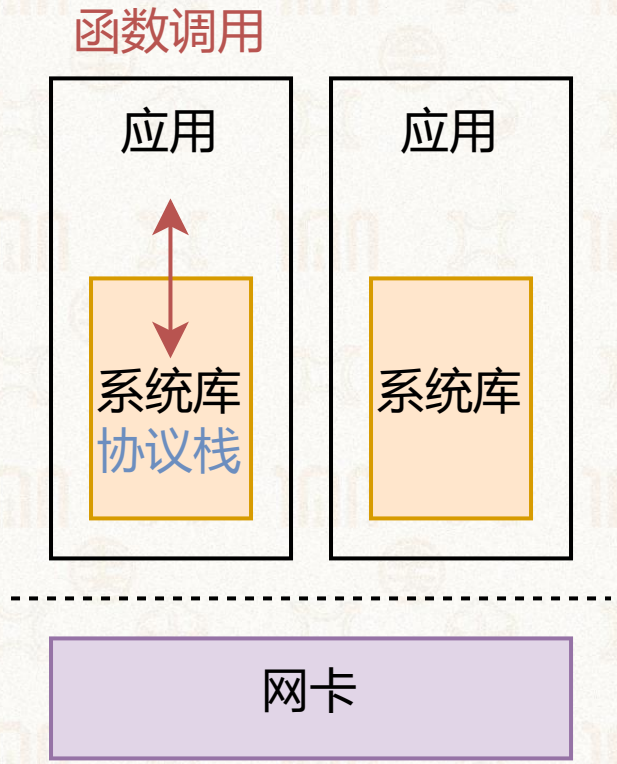
架构对比



宏内核系统



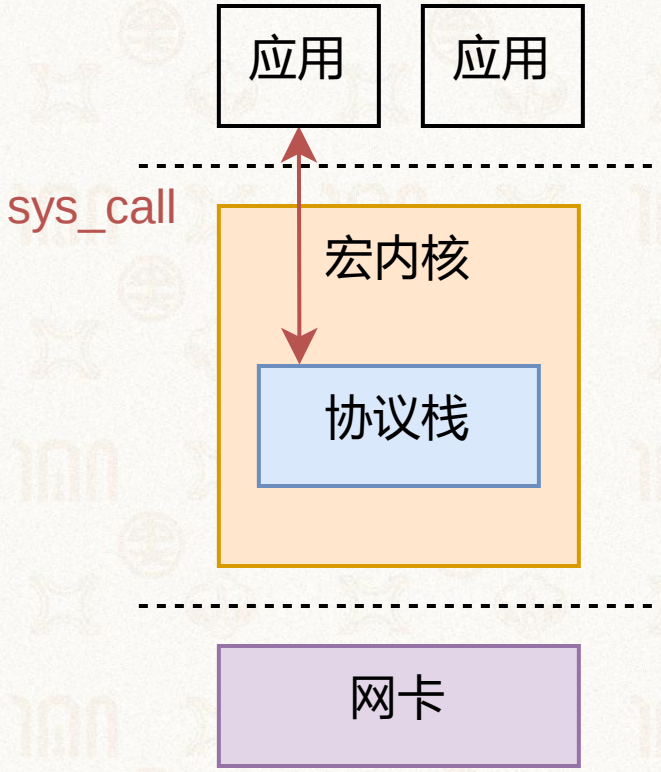
微内核系统



LibOS系统

架构对比：宏内核系统网络模块

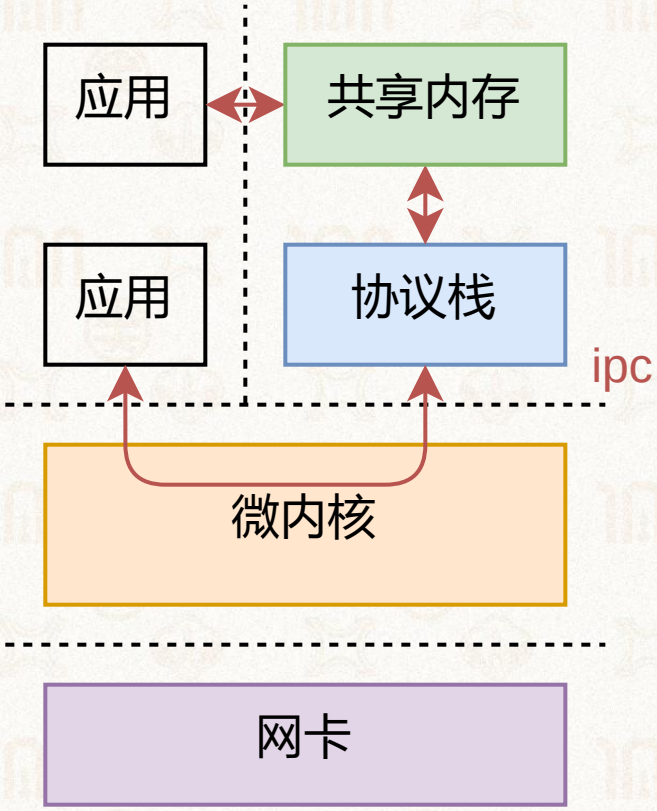
- 控制平面和数据平面都经过内核
- 驱动和协议栈处在同一地址空间，没有模式切换
- 驱动程序的安全问题会波及协议栈



宏内核系统

架构对比：微内核系统网络模块

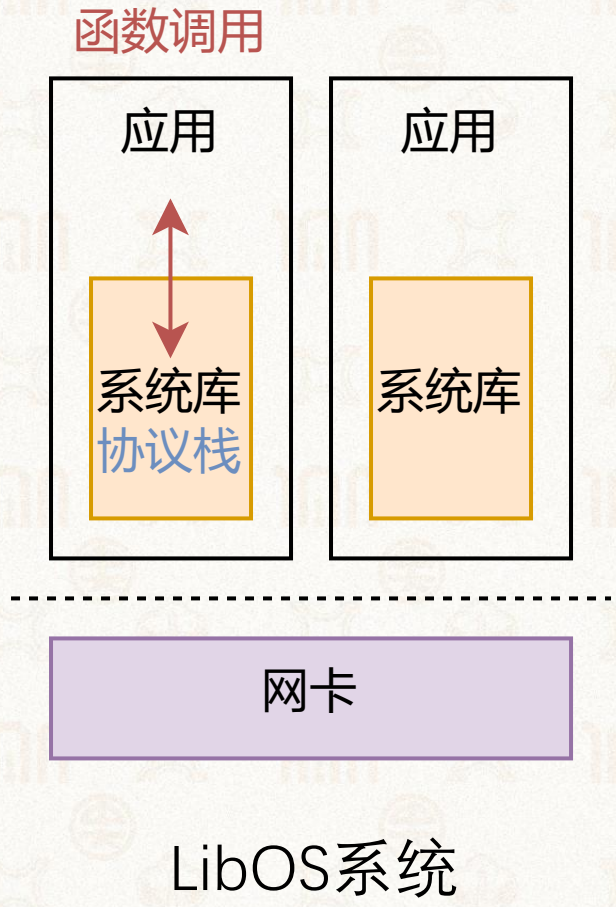
- 用户态驱动+协议栈，安全性好
- 只有一个协议栈，运维成本低
- 控制平面通信需要借助IPC完成，有一定开销



微内核系统

架构对比：库操作系统(LibOS)网络模块

- 用户态或non-root模式下协议栈，鲁棒性好
- 每个实例都有自己的协议栈
- 一旦需要更新，可维护性成本高
- 多实例轮询的情况下会导致浪费CPU



LibOS系统

谢谢

微信：suyuxin
钉钉：苏玉鑫
B站：https://space.bilibili.com/502854403
软工集市课程专区：https://ssemarket.cn/new/course
匿名提问箱：https://suask.me/ask-teacher/106/苏玉鑫