5. 虚拟化 (virtualization)

本章主要参考文献

- Bugnion E, Devine S, Govil K, et al. Disco: Running commodity operating systems on scalable multiprocessors[J]. ACM Transactions on Computer Systems (TOCS), 1997, 15(4): 412-447.
- Waldspurger C A. Memory resource management in VMware ESX server[J]. ACM SIGOPS Operating Systems Review, 2002, 36(SI): 181-194.
- Adams K, Agesen O. A comparison of software and hardware techniques for x86 virtualization[J]. ACM Sigplan Notices, 2006, 41(11): 2-13.

回顾

- 用语言特性实现模块化
  - 软模块化
  - 收益：源代码的模块化构造
  - 问题：错误在模块之间传播

回顾

- 用C/S+通信链路实现模块化
  - 强制模块化，限制除通信之外的所有交互
  - 收益
    - 防止程序错误传播 ← 隔离
    - 安全 ← 隔离
    - 容错/容灾/可靠性 ← 分布
  - 开销
    - 需要很多物理或虚拟设备，资源配置不灵活
      - 如果需要很多，能否解决？

√

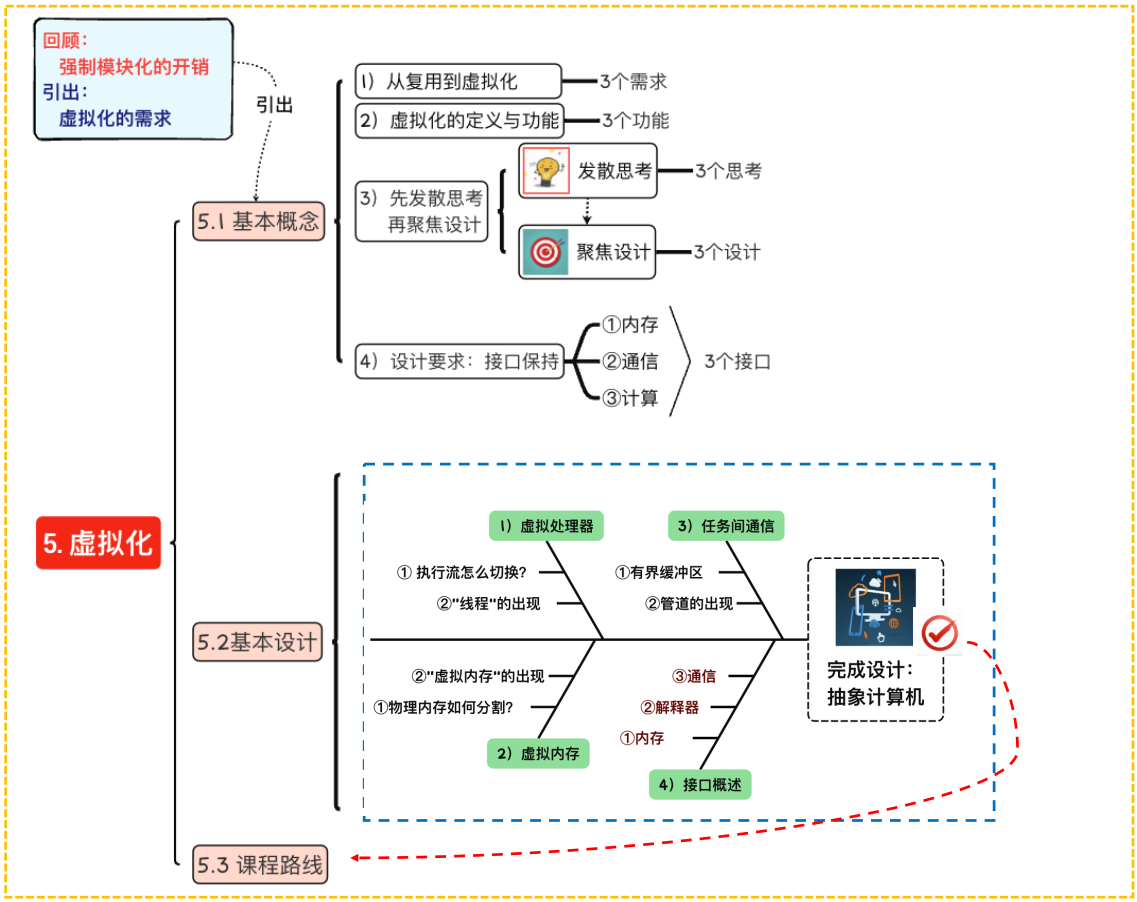
?

虚拟化将1台设备变多台！

一则旧闻

2014级本科生张焱宇，（大四）在GeekPwn'18成功演示VMwareESXi虚拟机逃逸，获奖金40万元。

本次演示为全球首例，VMware公司发布公开致谢。



基本概念


引出
4.1 基本概念
1）从复用到虚拟化
3个需求
2）虚拟化的定义与功能
3个功能
3）先发散思考
再聚焦设计
发散思考
3个思考
聚焦设计
3个设计
4）设计要求：接口保持
①内存
②通信
③计算
3个接口

1）从资源灵活使用到虚拟化

- 资源的灵活使用
  - 时、空两个维度
  - 延伸出三个需求：1变多，多合1，（甚至）A变B

- 计算机系统
  - 我想用Office Word编辑文件、同时用QQ聊天
  - 它们交替使用1个CPU，同时使用一份内存
  - 我也想用多个硬盘模拟一个巨大的硬盘

1）从资源灵活使用到**虚拟化**

- 可以通过人工分配，进行朴素的管理，但会带来：
  - 复杂的操作接口
  - 困难的资源协调
- 计算机系统重要的两个方法：增加**间接**层，用**抽象**设计接口。
  - 既间接化处理，又保持了接口 = **虚拟化**

2）虚拟化的定义和功能

- 定义：
  - 虚拟化是指如下过程：将计算机的各种实体或抽象资源，间接处理后呈现出保持原有接口的更高抽象层的资源。
- 功能
  - 复用：1个物理资源，多个虚拟资源
  - 聚合：多个物理资源，1个虚拟资源
  - 模拟：物理资源为A，虚拟资源为B
- 实施对象
  - 回忆：抽象计算机的划分？

计算	存储	通信

- 类比现实生活

复用：2台餐桌共用1名服务员

聚合：1台餐桌使用2名服务员

模拟：餐厅经理替代服务员

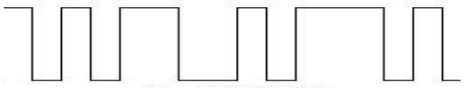
复用 + 聚合 = 伸缩

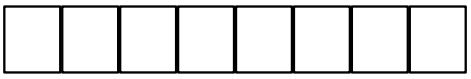
3）先发散思考，再聚焦设计

- “复用”是有条件的吗?
  - 资源：要在空间上分割
  - 任务：使用资源的时候在空间上有伸缩性
  - 资源：要在时间上分割
  - 任务：使用资源的时候能保存状态、能恢复状态
- “聚合”是有条件的吗?
  - 任务：可以分为多个并行的子任务
- “模拟”是有条件的吗?
  - 资源要能被模拟，就必须完全置于控制之下!

聚焦设计：先分割资源

- 通信
  - 属性：时间、频率
  - 分割：时分、频分
- 存储
  - 属性：空间、0和1
  - 分割：空间划分
- 计算
  - 属性：时间、指令
  - 分割：时钟周期





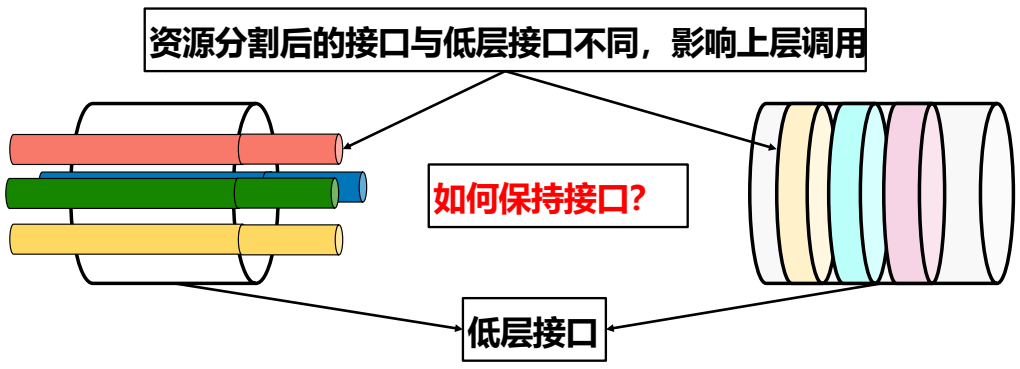
聚焦设计：再分配给任务

资源的分割 → **任务的分配**

- 通信资源
  - 频分 → 通信任务：使用不同频率同时进行
  - 时分 → 通信任务：分配到间隔开的时间片
- 存储
  - 空间划分 → 存储：分配到一部分的空间
- 计算
  - 时钟周期 → 计算：分配到一部分的时钟周期

4）接口设计：接口保持

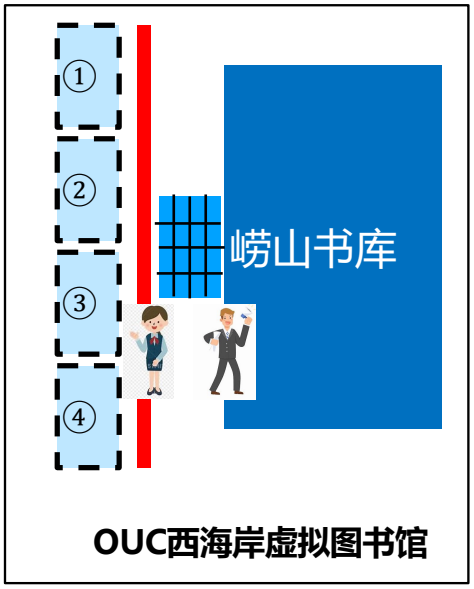
- 如何保持接口?
  - 先抽象出接口，再考虑如何通过间接层隐藏复杂调度，保持接口



目标：既满足资源灵活使用的功能！又保持接口的不变！

① 内存的接口保持

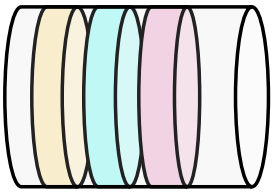
- 内存的访问接口（读）：
  - 输入：连续地址中的一个地址
  - 输出：该地址的值
  - 要求：满足写读连贯性
- 设计思路：
  - 地址映射：虚拟接口的地址是新地址
  - 保存关系：新地址到实际地址的映射关系
- 关键：间接层！



全新的地址　　更大的空间　　透明的访问

② 通信的接口保持

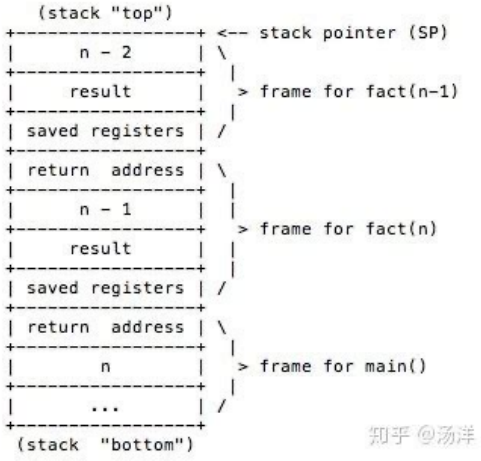
- 通信的接口：
  - send (数据, 目的地址) → { }
  - receive (源地址) → 数据
- 设计思路：
  - 设计间接层，接收输入 → 保持接口
  - （复用）将输入数据分成多个分组，分别发出去
- 关键：
  - 不要引入新的问题：数据差错，及时送达，分组次序



差错控制　　时延处理　　序列号

③ 计算的接口保持

- 计算的接口
  - 给定：指令引用、环境引用、指令集
  - CPU就能开启计算之旅
- 设计思路
  - 将这些状态保存起来，可封存
- 关键
  - 对应每个任务，保存状态就能重启
  - 提示：函数调用时是如何保存调用者状态的?



计算机系统中常见的虚拟化

	虚拟化方法	低层资源	虚拟资源
1	复用	服务器	网站
2	复用	处理器	线程
3	复用	物理内存	虚拟内存
4	复用	物理线路	虚电路
5	聚合	物理线路	链路捆绑
6	聚合	硬盘	磁盘阵列
7	模拟	硬盘	交换空间
8	模拟	苹果电脑	x86 PC机

虚拟化还有更多的收益

- 健壮性
  - 互相隔离逻辑错误
  - 但不能隔离物理错误
- 安全性
  - 一定程度上隔离攻击
  - 隔离效果低于物理隔离

小测验

1.C/S架构的要点是什么？写出交互中必要的操作步骤，和交互时重点要考虑的事项。

2.应对复杂性的四种一般方法：模块化、抽象、层次结构和分层。虚拟内存可能会主要使用哪种方法?

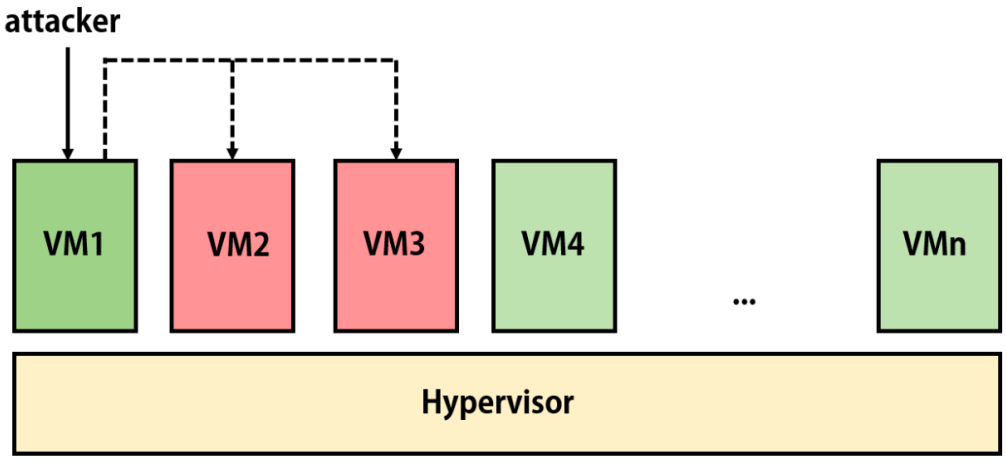
认真思考，抓住要点，不啰嗦、不笼统。1个问题不超过5句话。

主观题 10分 设置

请答题（任选一题作答）

作答

虚拟机跳跃、拒绝服务与逃逸



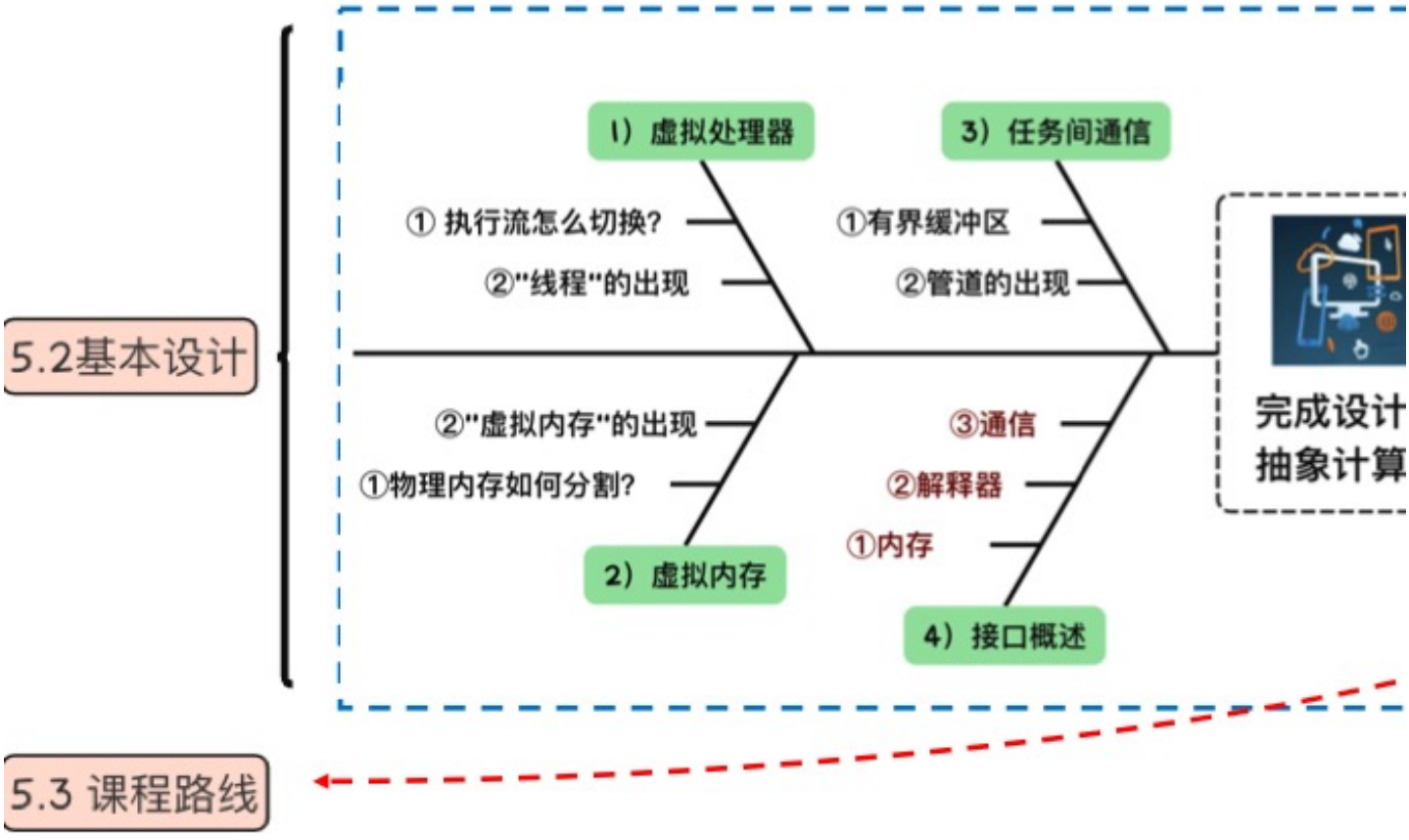
接下来……

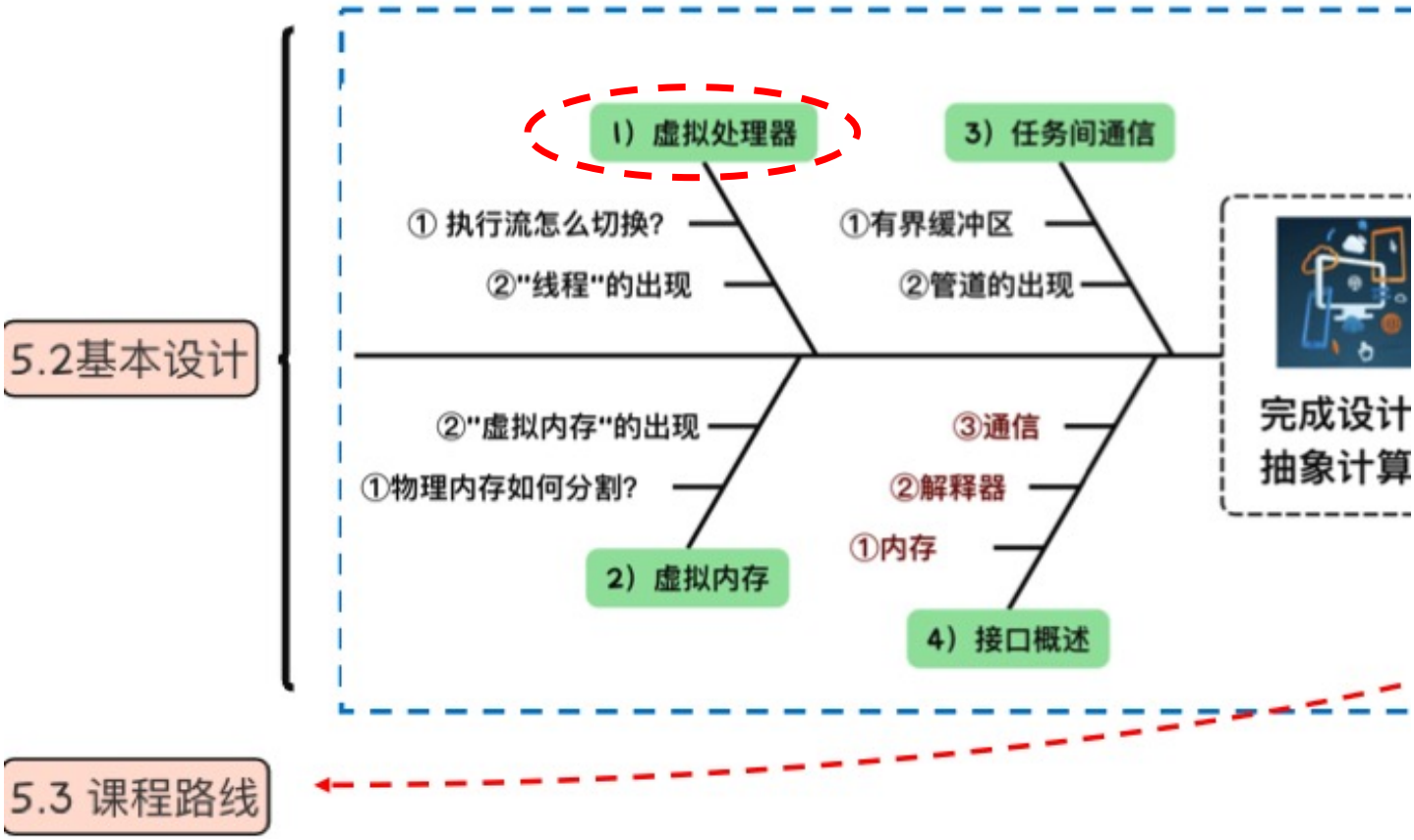
1. 目标：实现计算机资源的虚拟化
   - 物理资源：处理器、内存、通信链路

2. 挑战：
   - 如何建立可虚拟化（virtualizable）的计算机抽象
   - 并用正确的间接层实施虚拟化

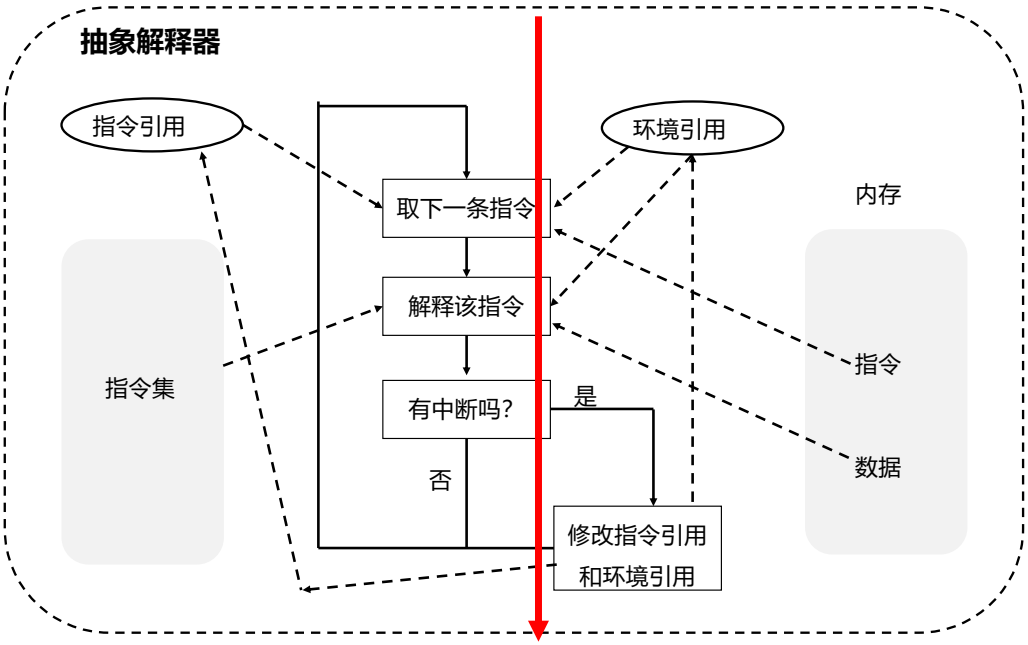
3. 设计成果：线程、虚拟内存、虚拟链路（任务间通信）

4. 最终成果：我们设计了一个操作系统内核！

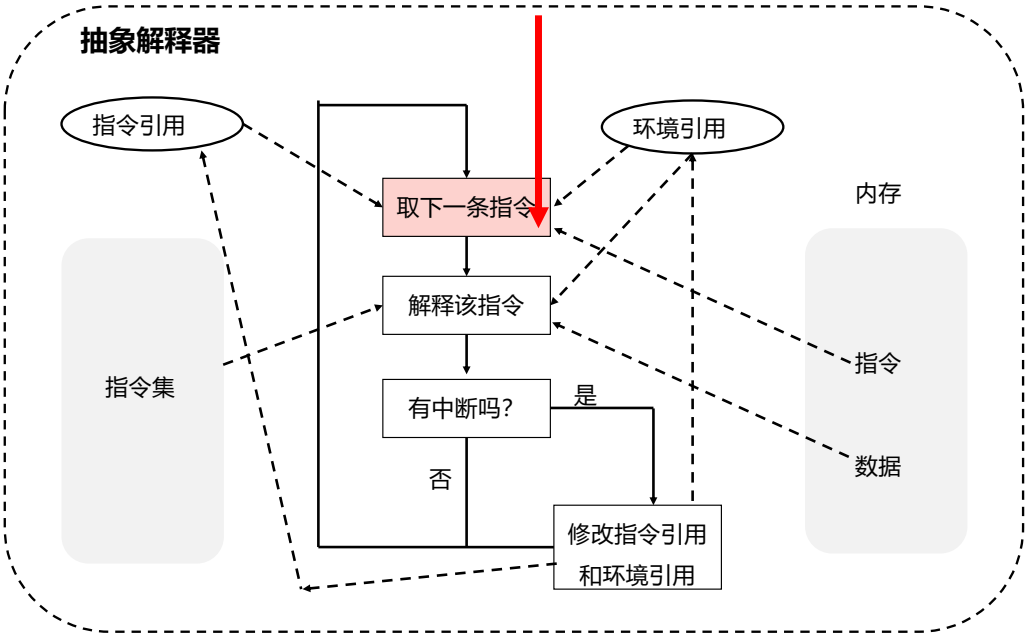



5.2基本设计
1）虚拟处理器
① 执行流怎么切换?
②"线程"的出现
3）任务间通信
①有界缓冲区
②管道的出现
②"虚拟内存"的出现
①物理内存如何分割?
2）虚拟内存
③通信
②解释器
①内存
4）接口概述
完成设计
抽象计算
5.3 课程路线

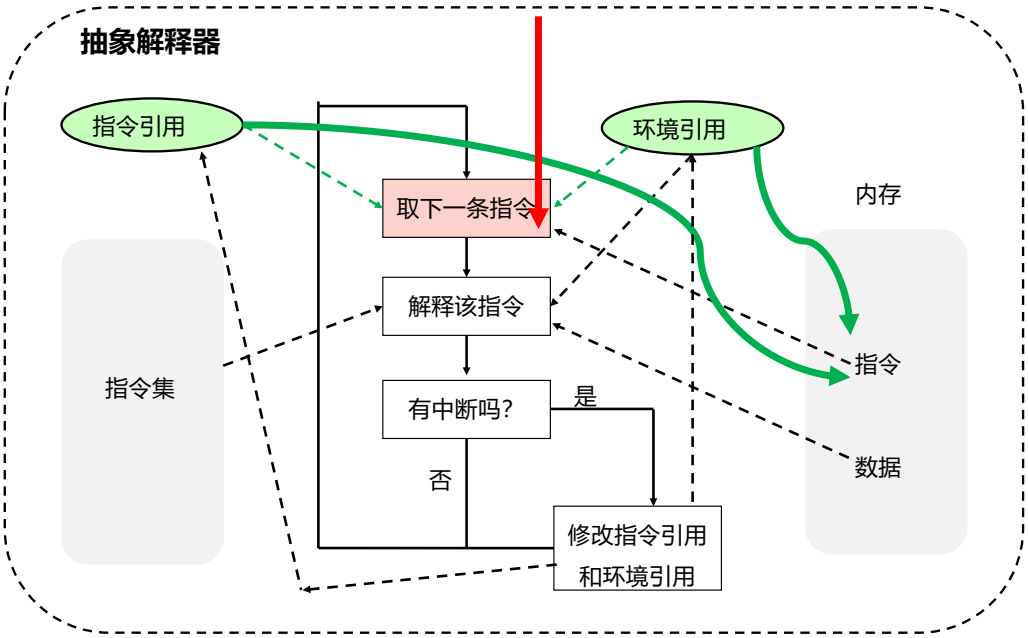
1）虚拟处理器


抽象解释器
指令引用
环境引用
取下一条指令
解释该指令
有中断吗?
是
否
修改指令引用
和环境引用
指令集
内存
指令
数据

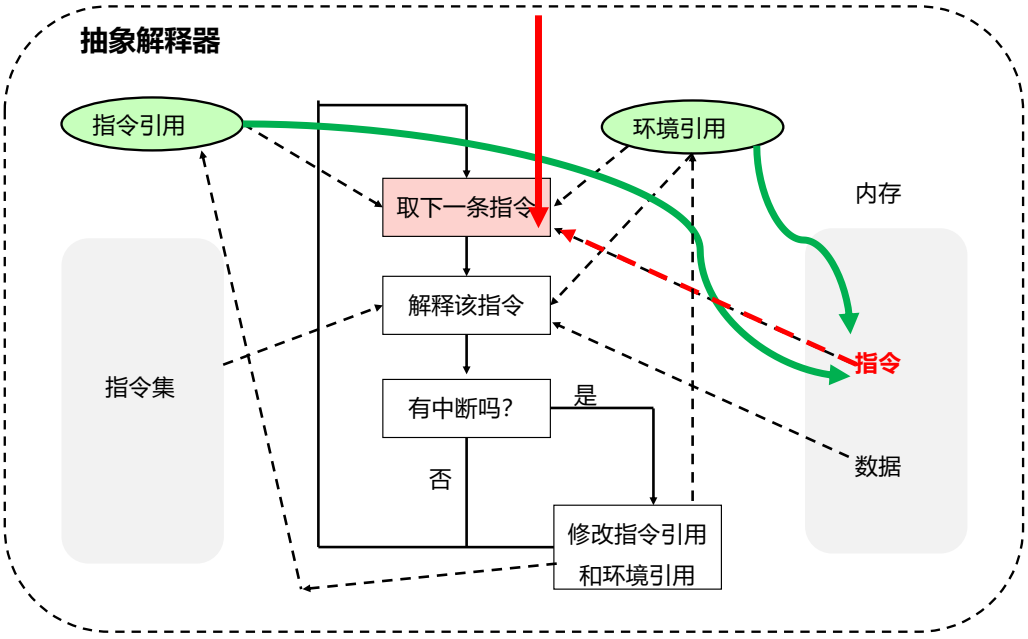
1）虚拟处理器



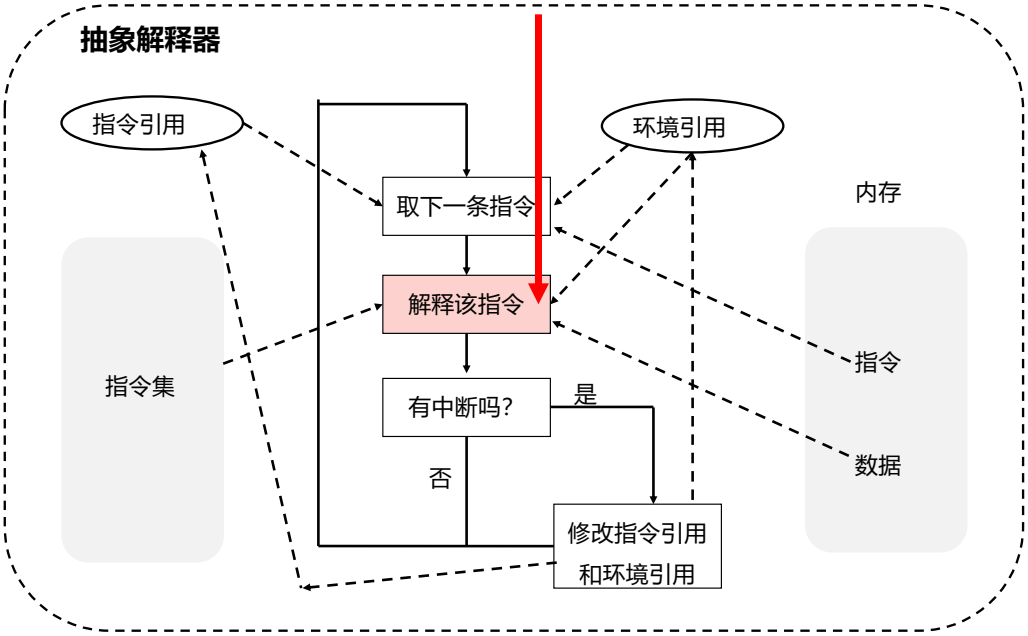
1）虚拟处理器



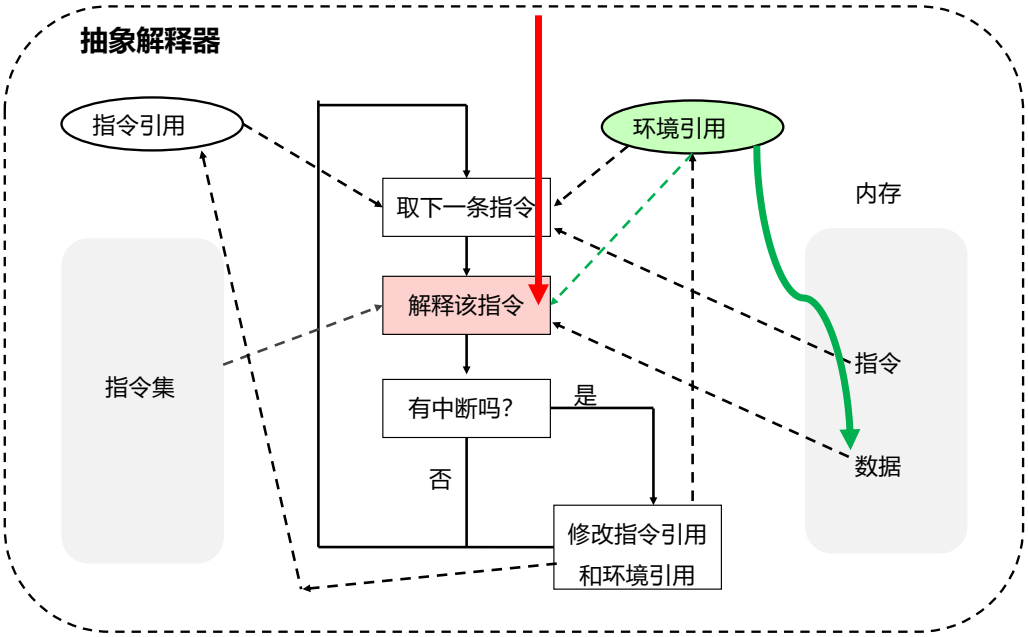
1）虚拟处理器



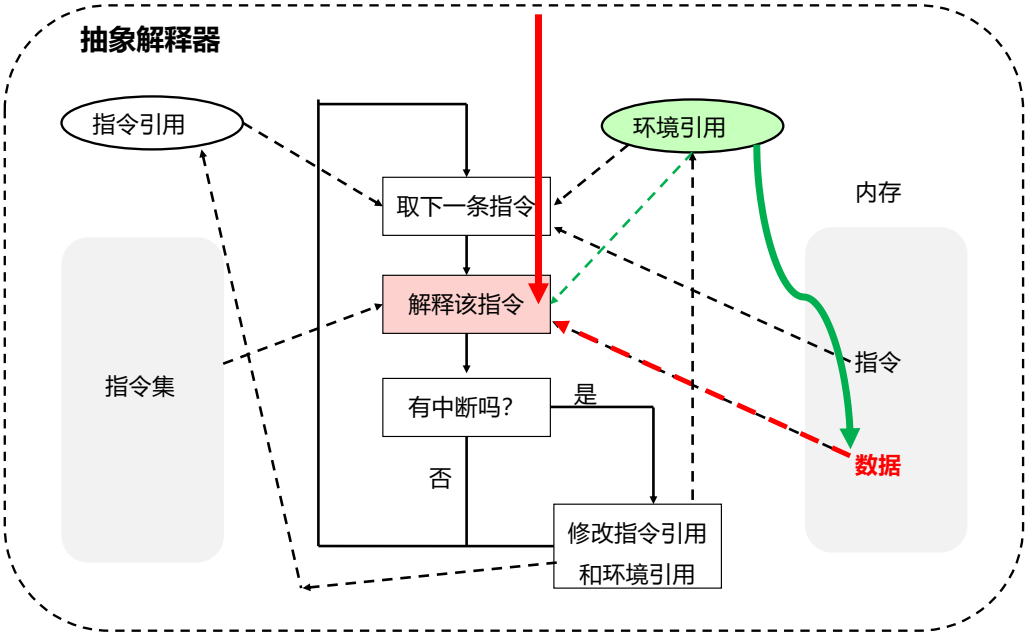
1）虚拟处理器



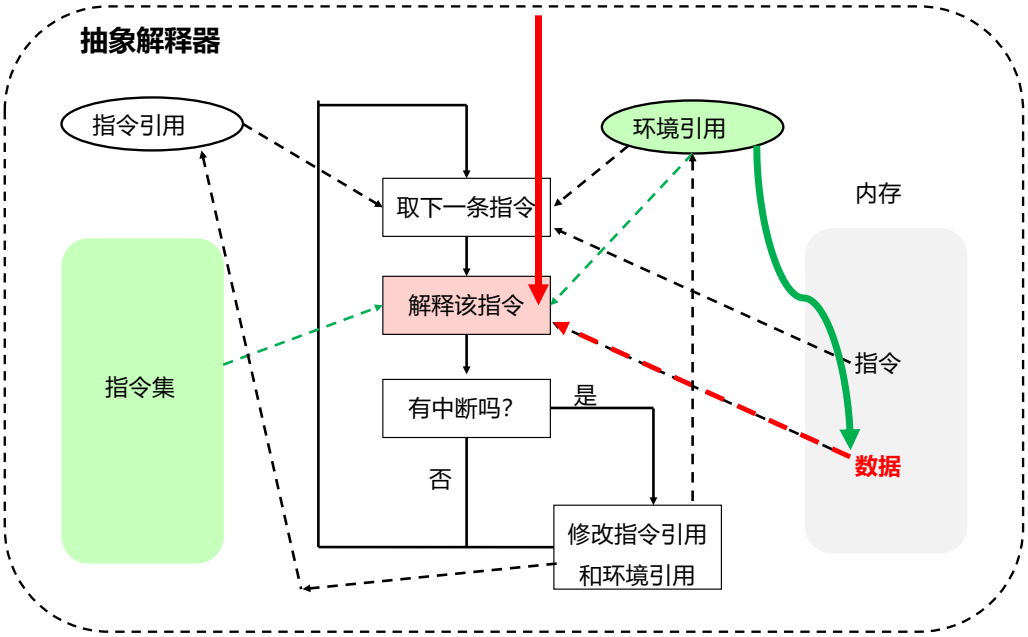
1）虚拟处理器



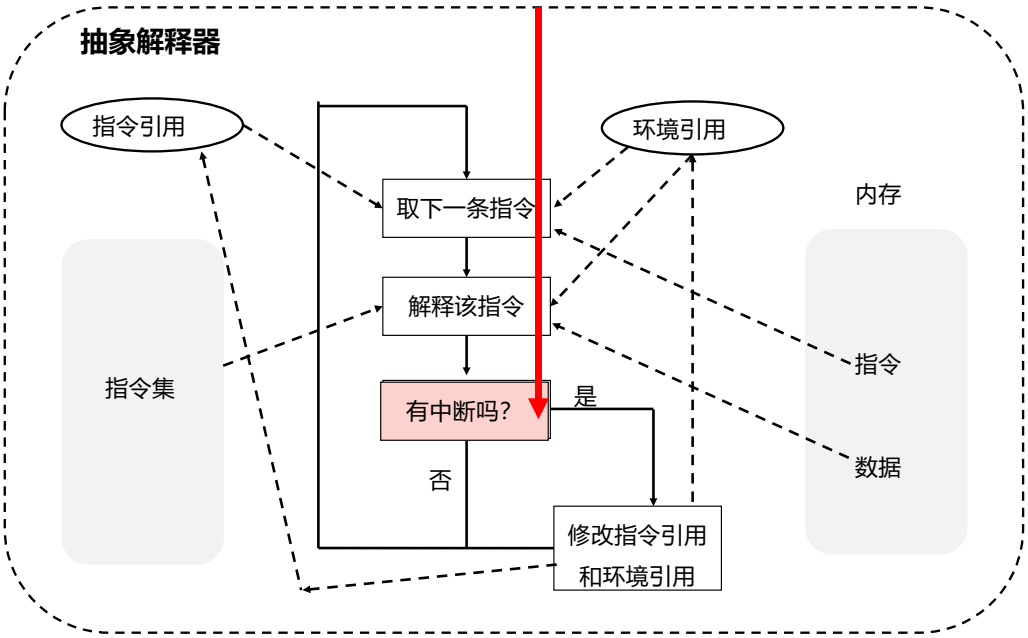
1）虚拟处理器



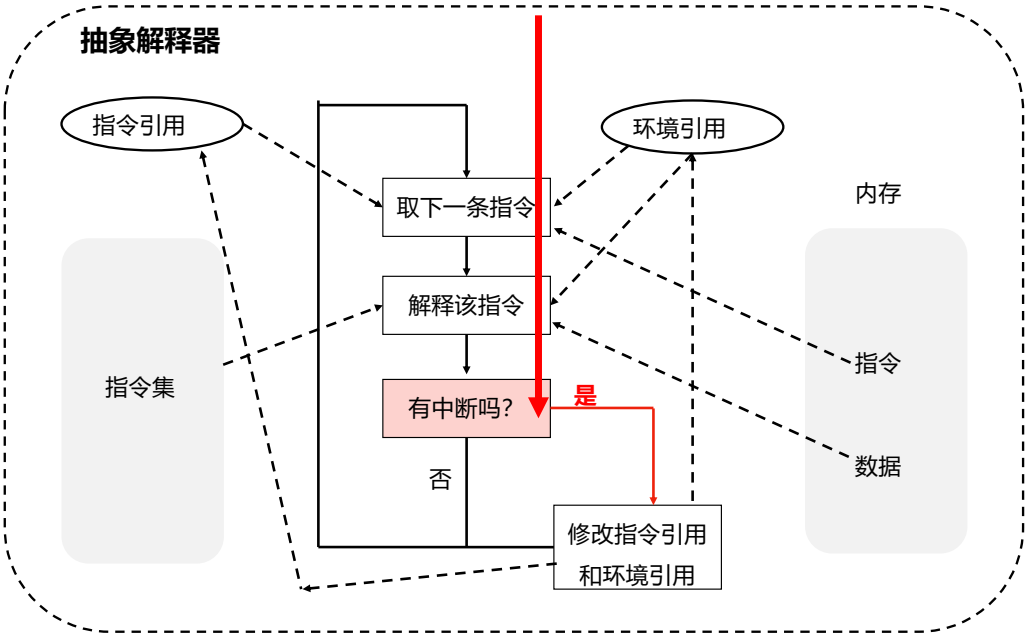
1）虚拟处理器



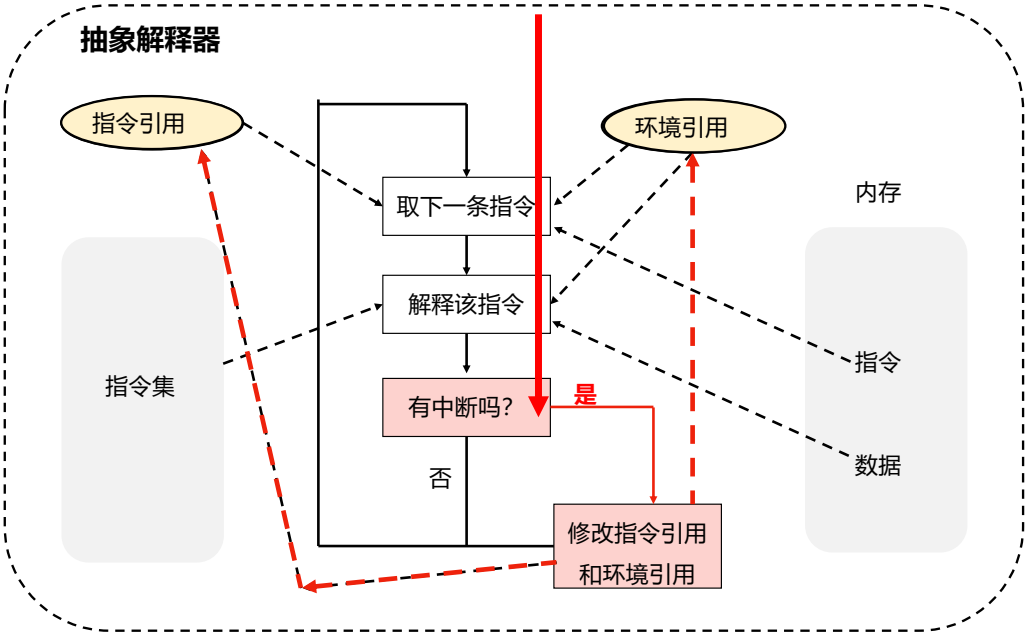
1）虚拟处理器



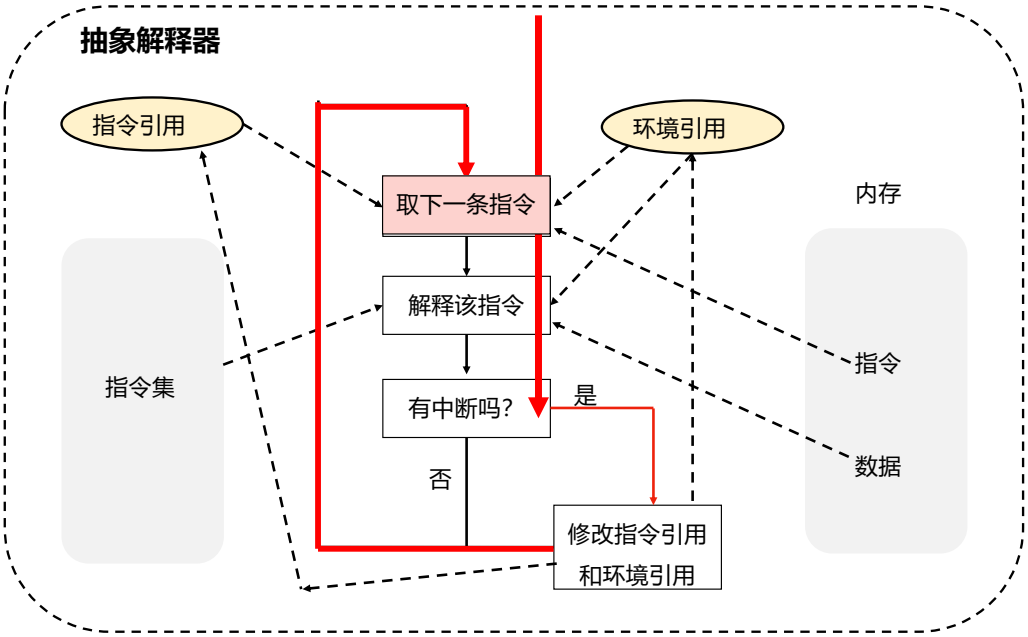
1）虚拟处理器



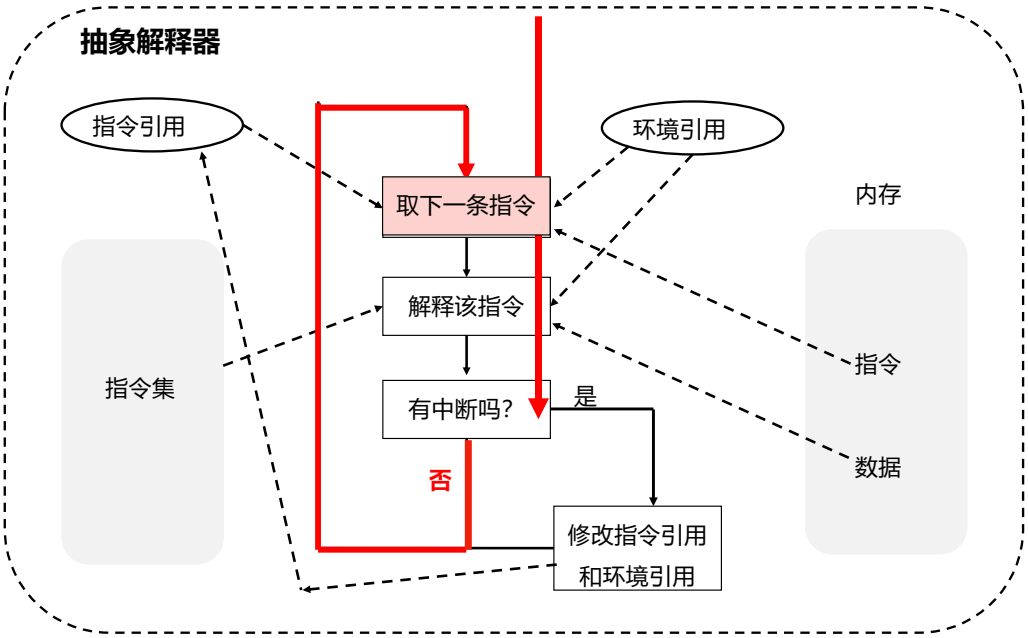
1）虚拟处理器



1）虚拟处理器

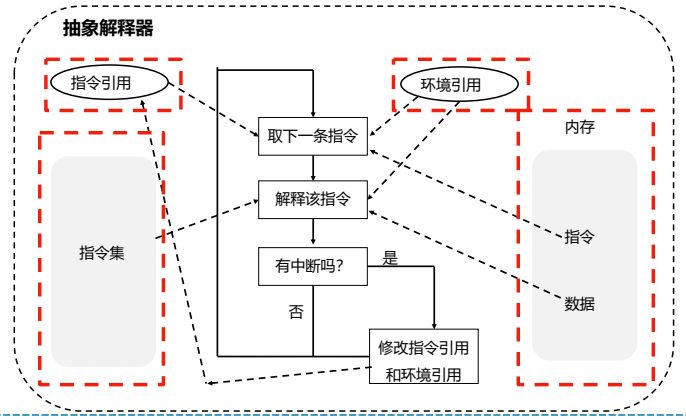


1）虚拟处理器



① 执行流如何切换?

- 物理处理器：是依次执行的指令时间流
  - 解释器：使用上下文[1]，执行指令[2]，访问总线[3]
  - 执行流状态由（指令引用、环境引用)+(内存、指令集）唯一确定
  - 可将执行流抽象为4元组：<指令引用，环境引用，内存，指令集>



① 执行流如何切换?

- 切换执行流时发生什么?
  - 4元组中：指令集不切换，内存不切换，**引用会切换**

- 继续抽象：
  - 用环境和指令引用作为执行流的抽象
    - x86：eip、寄存器、程序段……

- 能否将指令引用存于环境引用中?
  - 用环境引用作为执行流抽象
    - x86：sp

② 线程的出现

- 抽象后的执行流称为线程（thread of execution 简称 thread）
  - 如果抽象正确：
    - 随时可以移出处理器、停止
    - 随时可以移入处理器、恢复
    - 用1个sp就能访问
  - 1个线程的容器 = 1个虚拟处理器
  - 线程该怎么运行、怎么用?

线程是怎么运行起来的?

1.生成一个空的线程(容器)，拥有1个空的栈

2.为了能进行管理，第1个栈帧的返回地址应该是?
- 管理线程的某个专门函数的入口地址

3.为了让程序能在线程中运行，接下来应该做什么?
（假设程序已经加载到内存）
- 第2个栈帧是初始化函数的栈帧
- sp指向第3个栈帧，调用返回指令

重要结论：线程的关键并不在于内存中的代码和数据，而在于栈和栈帧的设立。

线程该怎么用?

- 程序员编写程序，朴素的思维是一个模块包含一个线程
  - 准确讲：一个顺序的代码序列
  - 符合人的思维习惯，分析更容易
    - 不违反[principle of least astonishment]最小惊吓原则

线程该怎么用?

- 但程序员更想用一个模块包含多个线程
  - 意想不到的惊喜和惊吓
  - 例：
    - 为每个设备启动一个线程（方便管理）
    - 多线程，交叠隐藏延迟（比如等待硬盘读写）
    - 多线程 对 多处理器，并行（并行计算）
    - 多线程，并发执行任务（用户体验）
    - 但人的思维并不能很好地处理并发序列（惊吓）

这是设计原本的面貌：
1.从朴素直白的想法，
2.到朴素直白的实现。
3.再到问题，再到改善。
而不是一眼就看到成品。

最大的惊吓：程序员必须处理数据的竞争使用。

管理线程

- 实现的难点
  - 任务：n个线程 复用 m个处理器
    - 隔离：互不影响（都保存有自己的状态）
    - 分时：并发运行（停/启 均能持续计算）
    - 关键：记录足够用的线程状态
  - 问题：如何分时？（如何让占据CPU的线程退出）
    - 自觉出让
    - 时钟中断：定时，CPU执行线程管理代码，按规则换入/换出

不难

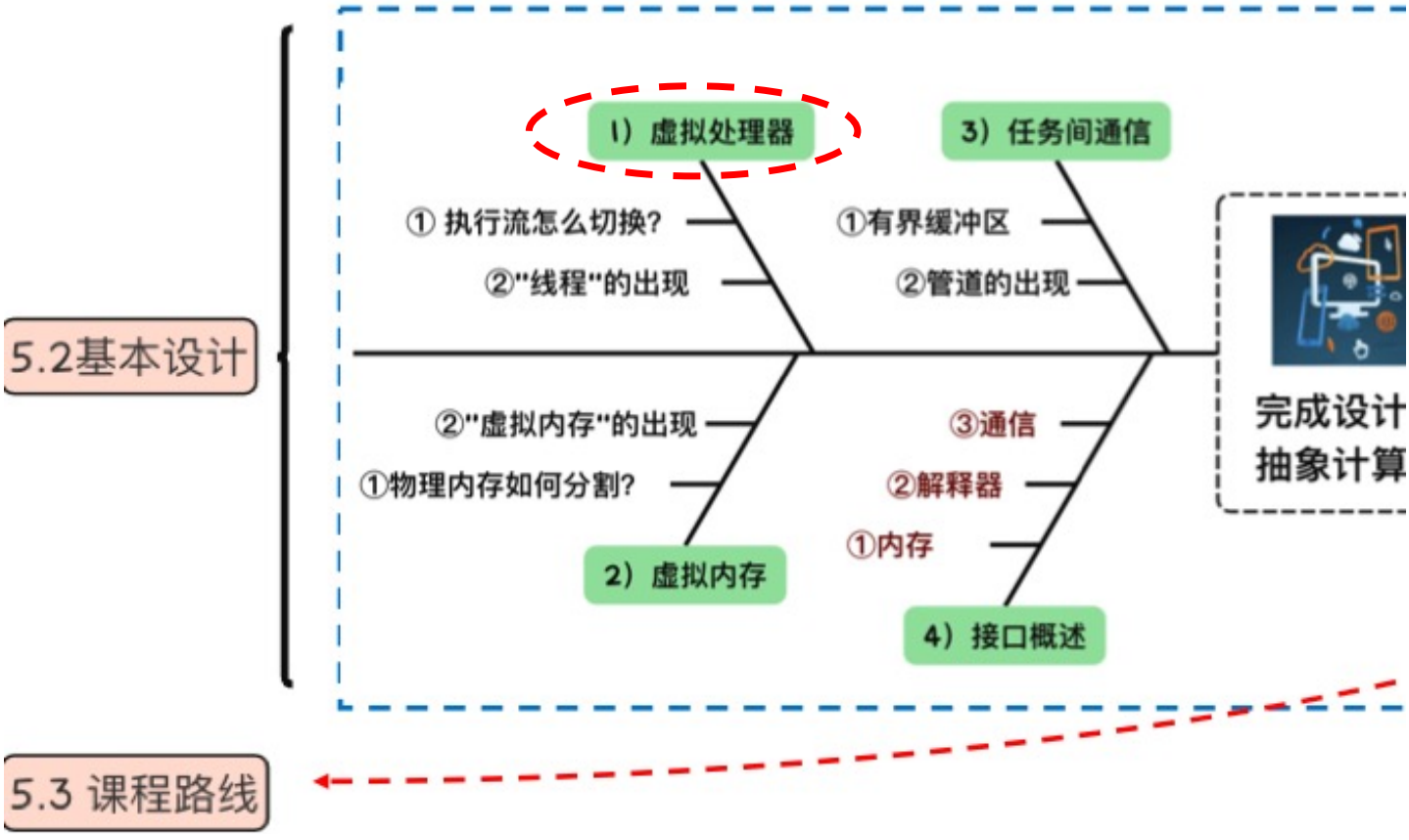
有难度

回顾中断与异常

- 中断（interrupt）
  - 外部事件引发的执行流异常变化
    - 一般外设引发，如：时钟定期引发时钟中断
  - CPU收到中断信号，就保存现场，进入中断处理程序
  - [自学]多处理器与中断嵌套
- 异常（exception）
  - 当前线程产生的执行流非正常变化，称为异常
    - 操作不能完成时发生，如：除以0
  - 过程同上，但中断处理器使用当前线程的上下文

中断与异常

- 程序设计语言经常使用异常机制
  - "goto有害论"之后，程序内部的异常机制用的更多了
- 上下文的差异，使中断/异常 处理程序的安全考虑有所不同
  - 大家思考，后面将展开



2）虚拟内存

- 线程共用（1份）内存的缺点
  - 错误传播
  - 不安全
  - 隔离依赖“软模块化”（代码约束）

- 虚拟内存

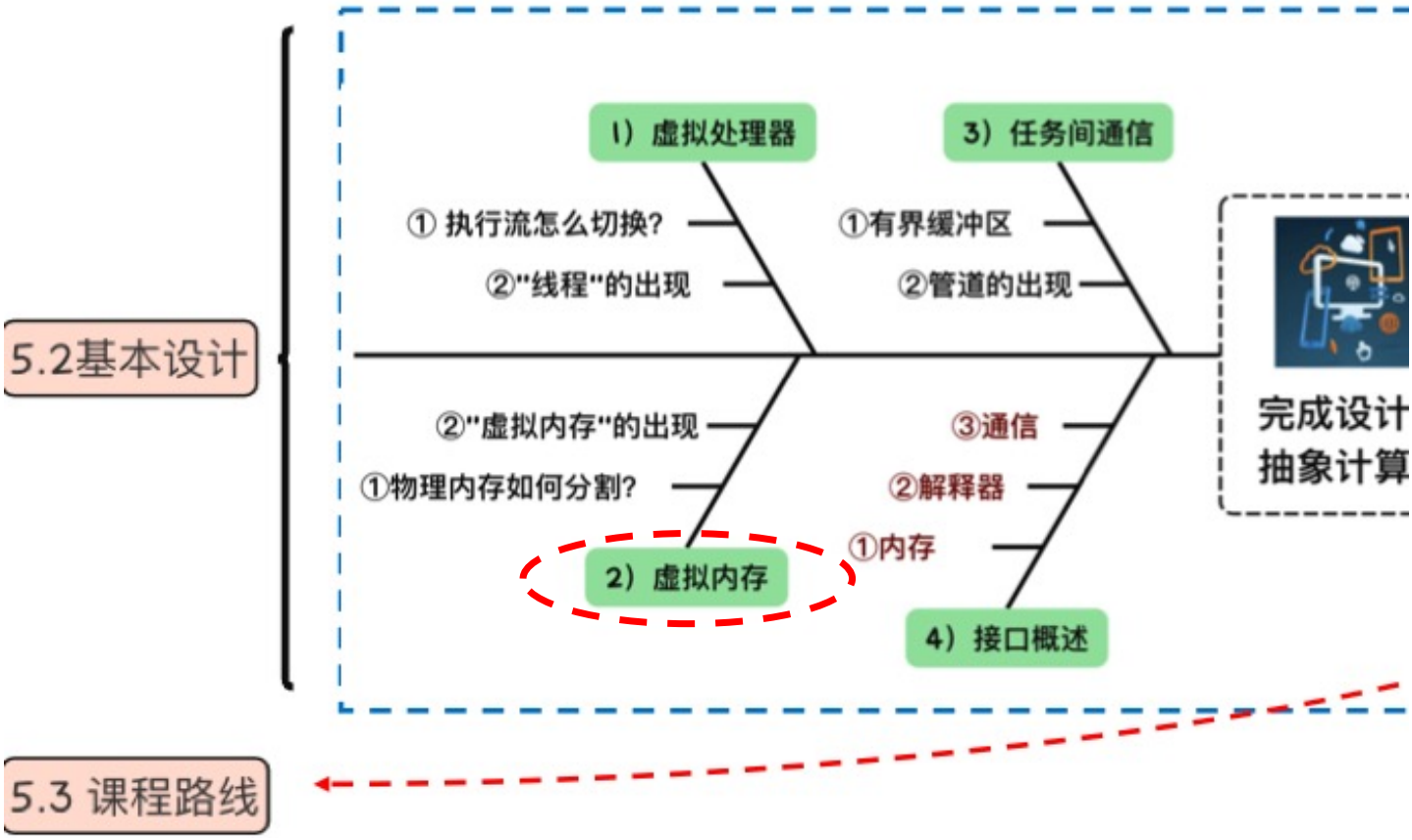
2）虚拟内存

- 虚拟内存
  - 阶段1：实施管理器，内存互相隔离
    - 依赖权限管理
  - 阶段2：增加间接层，增加名称解析
    - 虚拟地址以线程做上下文，都使用**相同**地址
    - 虚拟地址可延迟绑定，从而具备**很大**的虚拟地址空间
    - 由于解析使用了局部上下文，虚拟内存间自然**互相隔离**

便利-降低复杂性
性能-提高容量
安全-隔离资源

问题-性能-时延高
性能-吞吐率低
性能

- 有硬件支持更好：虚拟内存管理器(硬件MMU)



3）线程间通信：有界缓冲区

问题：线程之间需要交换数据/交互 怎么办?

- 方案：有界缓冲区
  - 一段固定大小的内存，首尾两个指针
  - 发送者将数据写入
  - 标记数据首部的指针+1

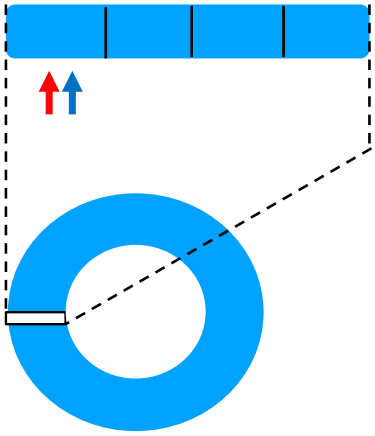
头-尾=N
则等待

  - 接收者读取数据，并删除
  - 标记数据的尾部的指针+1

头 = 尾
则等待

0	1	10	11
100	101	110	111
1000	尾	√	√
头	1101	1110	1111

mod 100



——《数据结构》循环队列

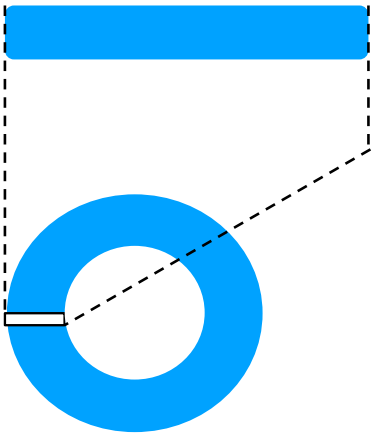
该（模拟）设计再加上复用，会有问题吗?

3）线程间通信：有界缓冲区

问题：线程之间需要交换数据/交互 怎么办?

- 原语设计：
  - SEND：插入数据，满则等待
  - RECEIVE：读取数据，空则等待

- 该抽象链路上，可以进一步实现RPC

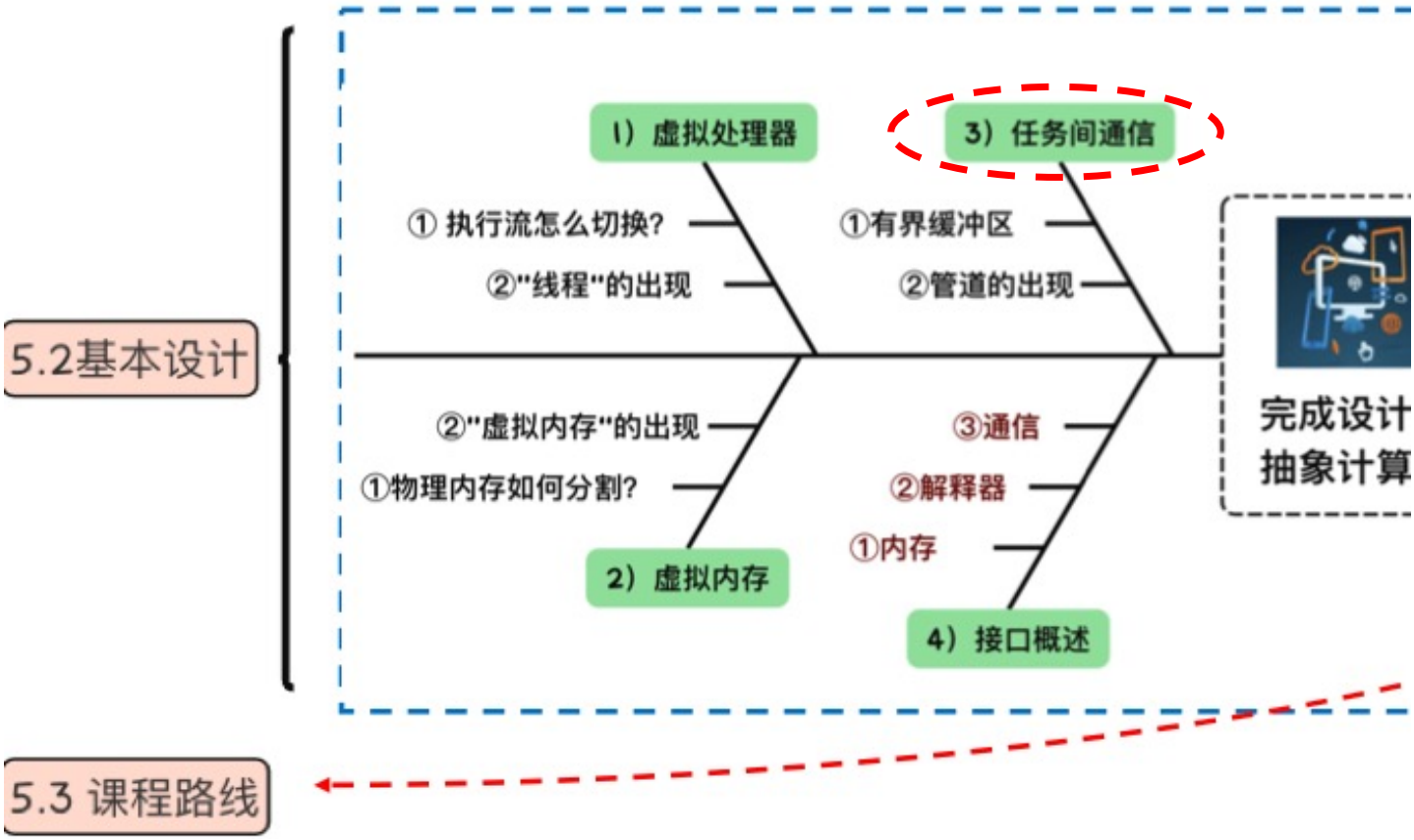


——《数据结构》环形队列

管道

- 思考：当我们使用管道的时候，发生了什么?

```
~ % ls -la | grep cse
```



4）接口概述

抽象模块	接口
内存	CREATE_ADDRESS_SPACE
	DELETE_ADDRESS_SPACE
	ALLOCATE_BLOCK
	FREE_BLOCK
	MAP
	UNMAP
通信	ALLOCATE_BOUNDED_BUFFER
	DEALLOCATE_BOUNDED_BUFFER
	SEND
	RECEIVE

抽象模块	接口
解释器	ALLOCATE_THREAD
	EXIT_THREAD
	DESTROY_THREAD
	YIELD
	AWAIT
	ADVANCE
	TICKET
	ACQUIRE
	RELEASE

4）接口概述

如何学习和研究这些接口（又称为抽象）？

- 小型概念OS
  - 原理：揭示真实OS中的某些机制
  - 简单：具体OS无关
- 具体OS
  - 性能优化：不清晰、不简洁
  - 可靠性保证：充满各类细节

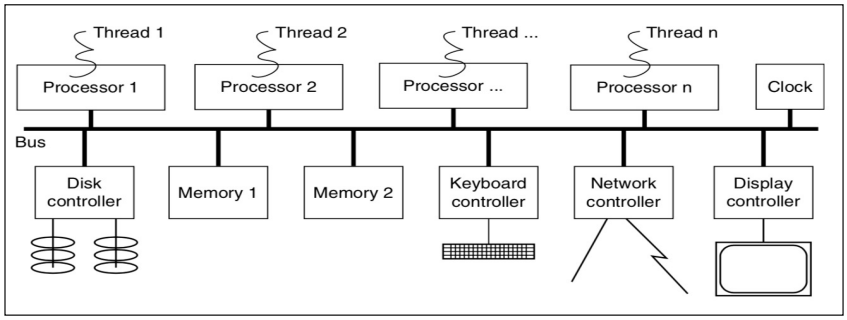
  - 具体说明不一定要用具体系统，也可用具体概念的系统

虚拟计算机

- 线程+虚拟内存 = 虚拟计算机
- 虚拟计算机 + 虚拟链路 → 为强制模块化提供架构基础
- 接口定义：清晰地分析多层抽象的系统架构
- 第6-8章将对三者分别进行设计

5.3 课程路线：如何逐步实现虚拟化

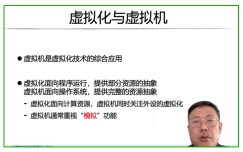
- 从弱假设开始：
  - 处理器多于线程，1 中断处理程序也有自己的专用处理器 2
  - 允许内存共用，3 线程的内存访问不会出错 4
  - 线程间无需通信 5
  - 例：
    - 5个程序模块，共享内存，共同控制



5.3 课程路线：如何逐步实现虚拟化

- 逐步去掉假设：
  - 如何建立有效的虚拟通信链路，构成C/S模式？
  - 如何解决共享内存破坏模块化的问题？
  - 如何建立相互隔离的内存？
  - 如何解决处理器不够的情况？
  - 如何结合计算任务，有条件出让处理器？如何唤醒？
- 实例分析：x86历史，逐步支持模块化

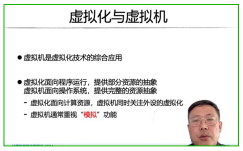
虚拟化与虚拟机（在线资源）



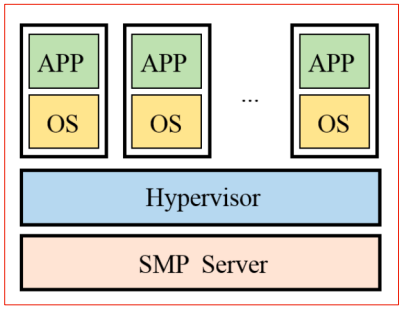
- 虚拟机是虚拟化技术的综合应用

- 虚拟化面向程序运行，提供部分资源的抽象
  虚拟机面向操作系统，提供完整的资源抽象
  - 虚拟化面向计算资源，虚拟机同时关注外设的虚拟化
  - 虚拟机通常重视“模拟”功能

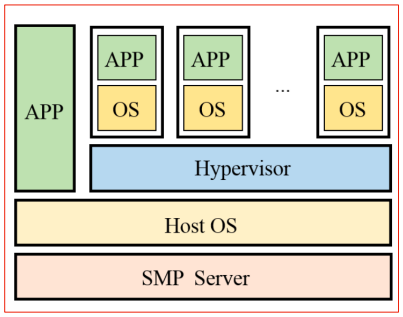
虚拟机分类（在线资源）



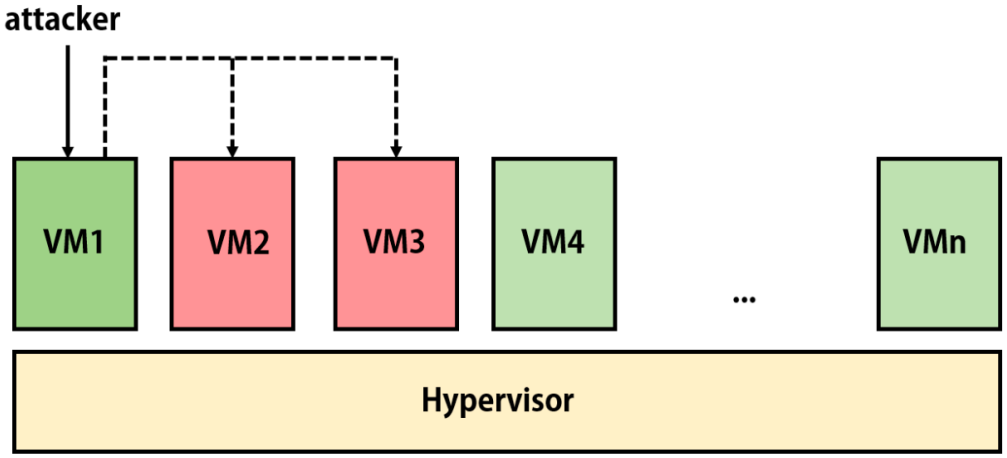
- 第一类虚拟机



- 第二类虚拟机



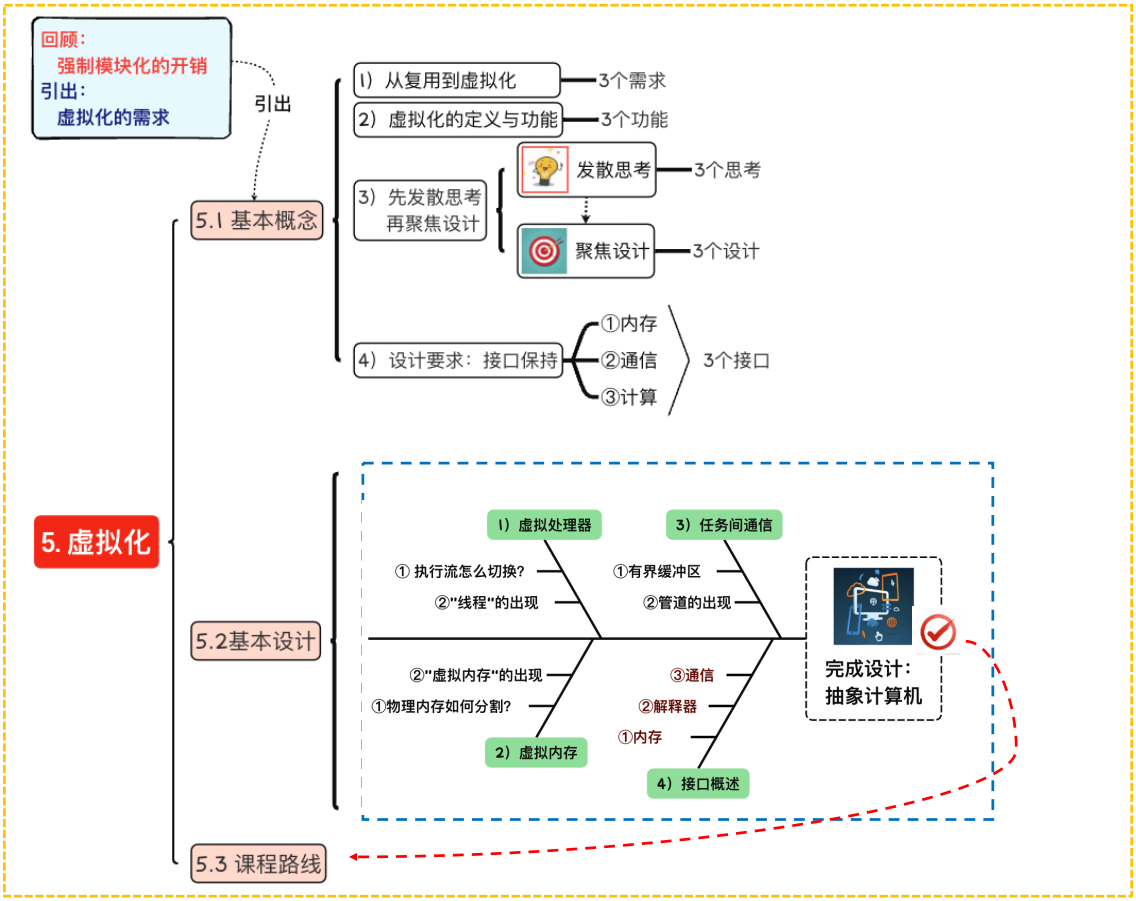
虚拟机跳跃、拒绝服务与逃逸



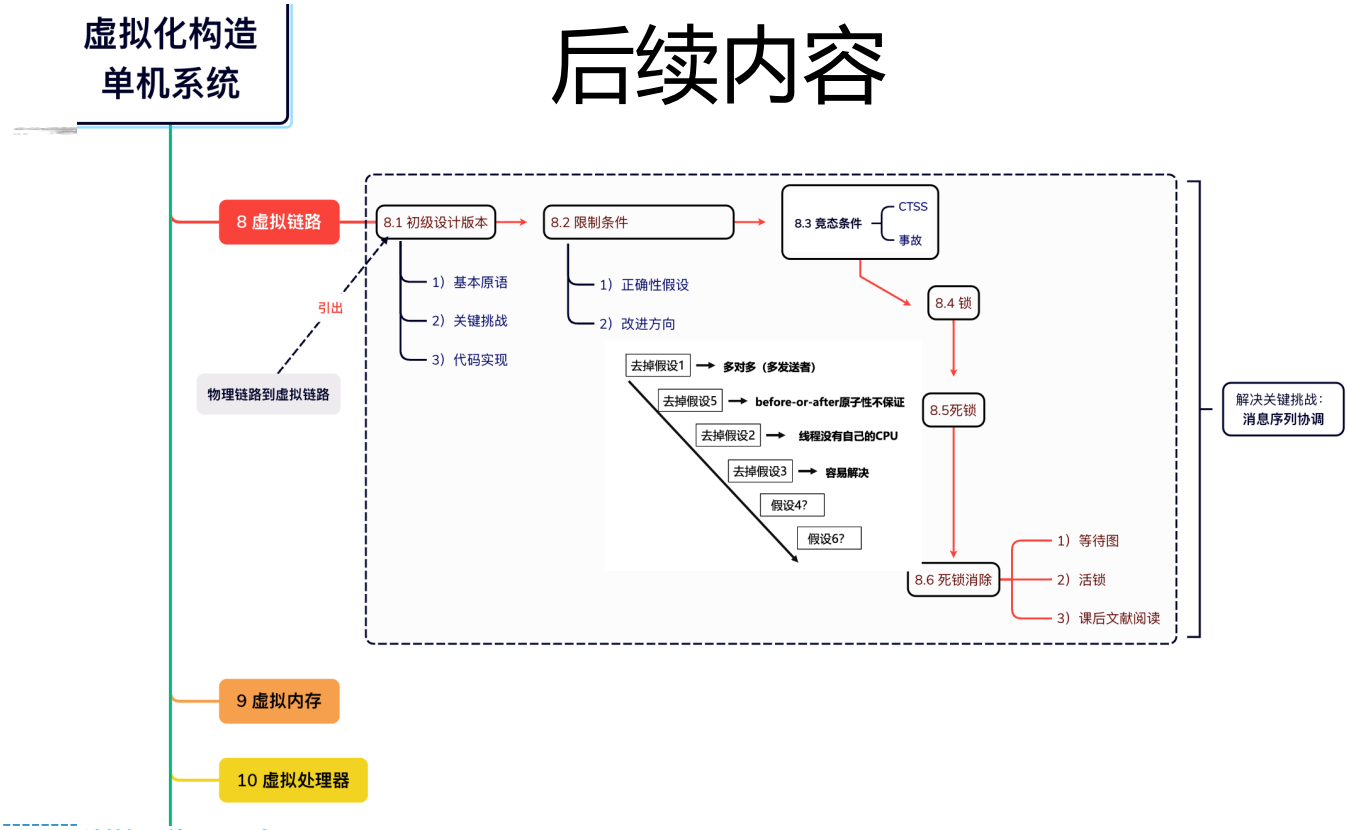
模拟器有什么用?

- 在新的硬件上使用旧的软件，低成本迁移（模拟过去的硬件）
  - 苹果在PowerPC上模拟Motorola，在x86上模拟PowerPC
- 提前开发（模拟未来的硬件）

总结


回顾：
强制模块化的开销
引出：
虚拟化的需求
引出
5.1 基本概念
1）从复用到虚拟化
3个需求
2）虚拟化的定义与功能
3个功能
3）先发散思考
再聚焦设计
发散思考
3个思考
聚焦设计
3个设计
4）设计要求：接口保持
①内存
②通信
③计算
3个接口
5. 虚拟化
5.2基本设计
1）虚拟处理器
① 执行流怎么切换?
②"线程"的出现
②"虚拟内存"的出现
①物理内存如何分割?
2）虚拟内存
3）任务间通信
①有界缓冲区
②管道的出现
③通信
②解释器
①内存
4）接口概述
完成设计：
抽象计算机
5.3 课程路线

后续内容


虚拟化构造
单机系统
8 虚拟链路
8.1 初级设计版本
1）基本原语
2）关键挑战
3）代码实现
8.2 限制条件
1）正确性假设
2）改进方向
8.3 竞态条件
CTSS
事故
8.4 锁
8.5死锁
去掉假设1 → 多对多（多发送者）
去掉假设5 → before-or-after原子性不保证
去掉假设2 → 线程没有自己的CPU
去掉假设3 → 容易解决
假设4?
假设6?
8.6 死锁消除
1）等待图
2）活锁
3）课后文献阅读
解决关键挑战：
消息序列协调
引出
物理链路到虚拟链路
9 虚拟内存
10 虚拟处理器

重要术语中英文对照

虚拟化：virtualization

复用：multiplexing

聚合：aggregating

模拟：emulation

线程：thread

虚拟内存：virtual memory

中断：interrupt

异常： exception

虚拟机： virtual machine

虚拟机管理器：VMM（ virtual machine monitor）

本章主要参考文献

- Bugnion E, Devine S, Govil K, et al. Disco: Running commodity operating systems on scalable multiprocessors[J]. ACM Transactions on Computer Systems (TOCS), 1997, 15(4): 412-447.
- Waldspurger C A. Memory resource management in VMware ESX server[J]. ACM SIGOPS Operating Systems Review, 2002, 36(SI): 181-194.
- Adams K, Agesen O. A comparison of software and hardware techniques for x86 virtualization[J]. ACM Sigplan Notices, 2006, 41(11): 2-13.


第5章 结束