



# 虚存和地址转换

---

中国科学院大学计算机与控制学院  
中国科学院计算技术研究所

2019-11-11





# 内容提要

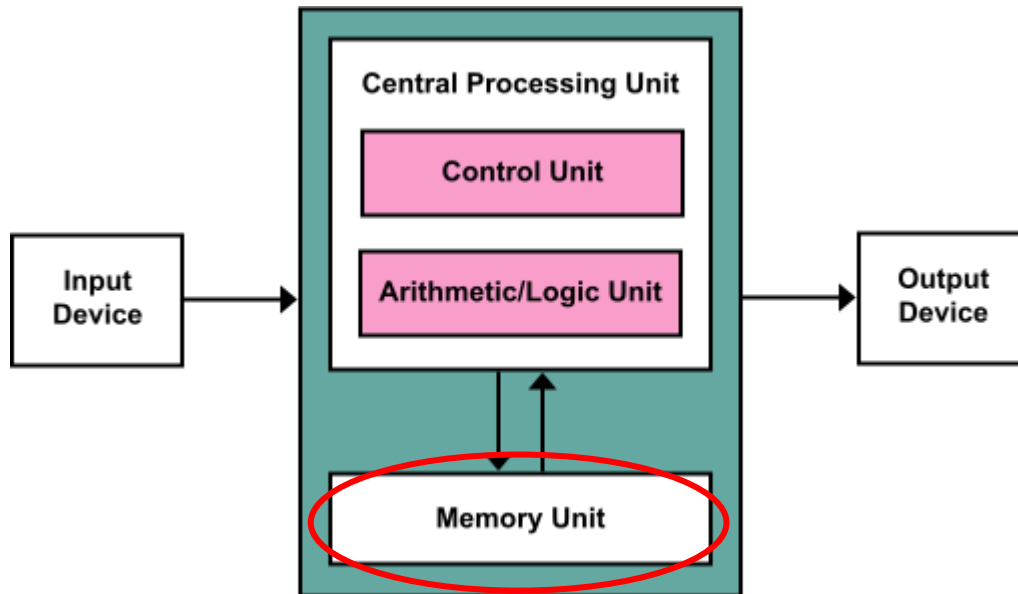
---

- 计算机存储体系结构
- 虚存
  - 虚拟化
  - 保护
- 地址映射
  - 基址+长度
  - 分段
  - 分页
- TLB



# 现有计算机体系结构

- 冯·诺伊曼结构



《天才的拓荒者——冯·诺依曼传》  
[美] 麦克雷，上海科技出版社  
2008年12月

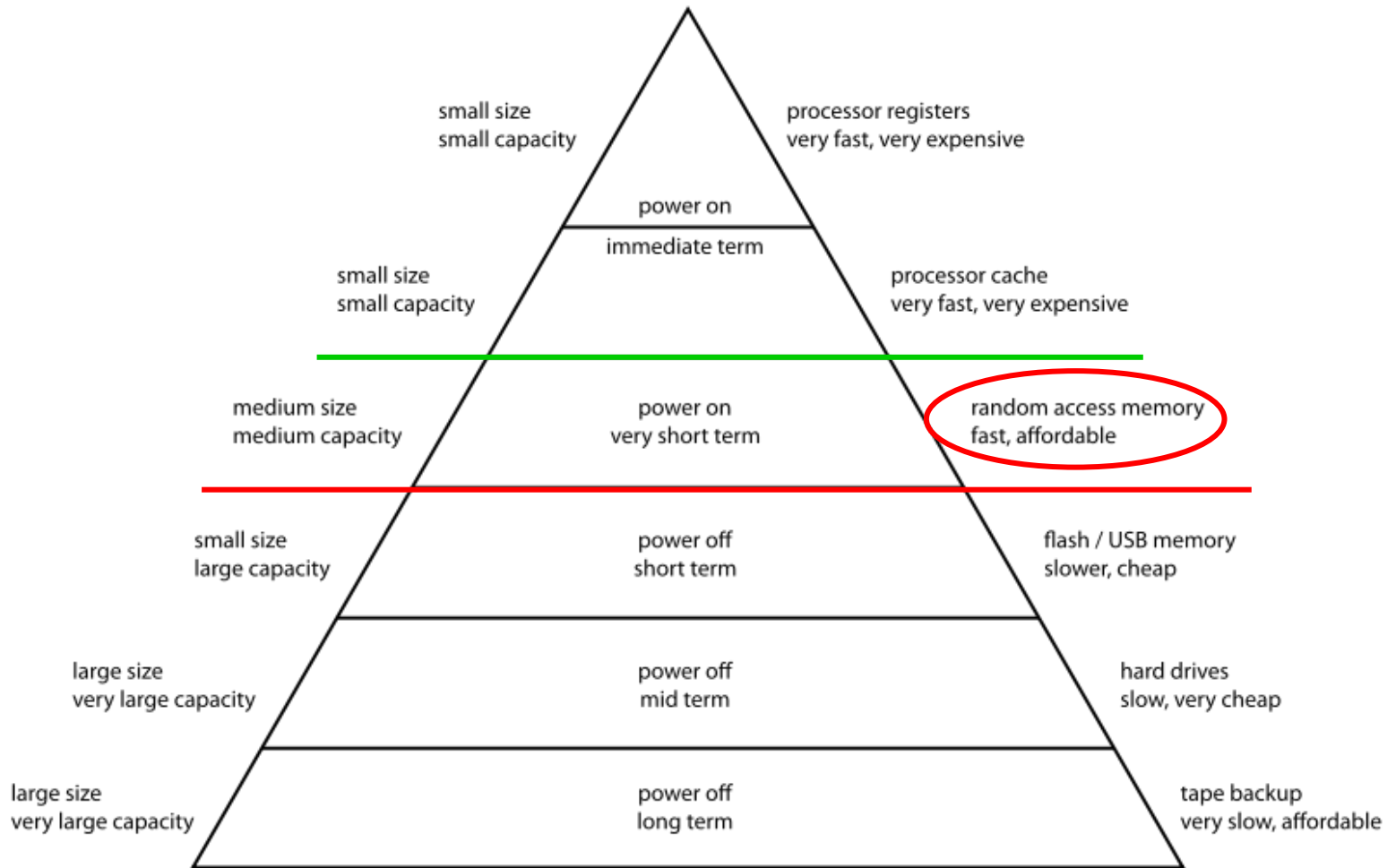
Source: [https://en.wikipedia.org/wiki/Von\\_Neumann\\_architecture](https://en.wikipedia.org/wiki/Von_Neumann_architecture)

Jonh von Neumann, **First Draft of a Report on the EDVAC**, 1945



# 现有计算机体系结构

- 层次化存储结构 Computer Memory Hierarchy



图片来源: [https://en.wikipedia.org/wiki/Memory\\_hierarchy](https://en.wikipedia.org/wiki/Memory_hierarchy)



# 现有计算机体系结构

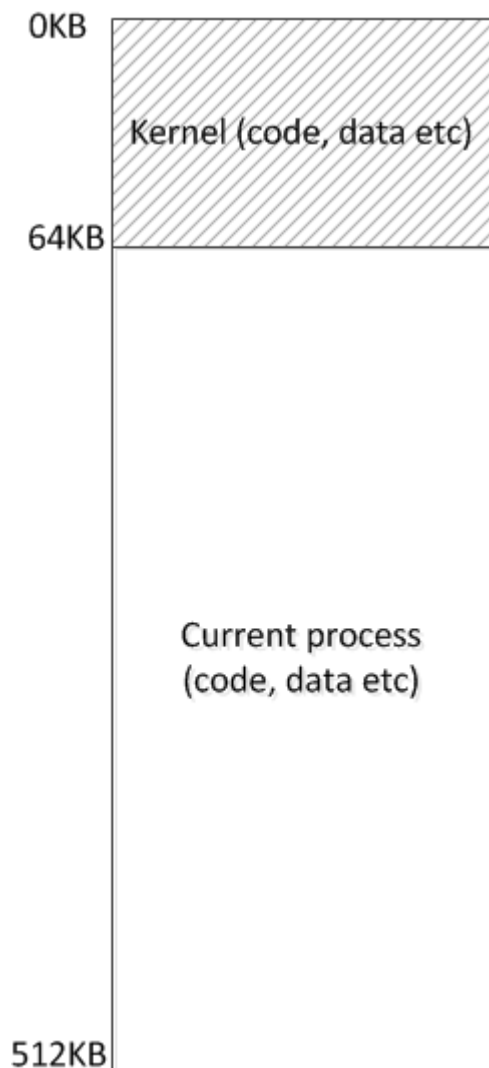
- 内存DRAM：快，但贵，容量小，易失性
- 外存磁盘：持久化，便宜，容量大，但慢

现有计算机系统存储层次	延迟	容量
Register	0.4~1ns	2K
L1 cache	1~4ns	16~32K
L2 cache	5~10ns	64~256K
L3 cache	20~40ns	1~32M
Memory	100ns	4~16G
Flash-based SSD	20~100us	1~4T
Hard disk	3~5ms	1~4T

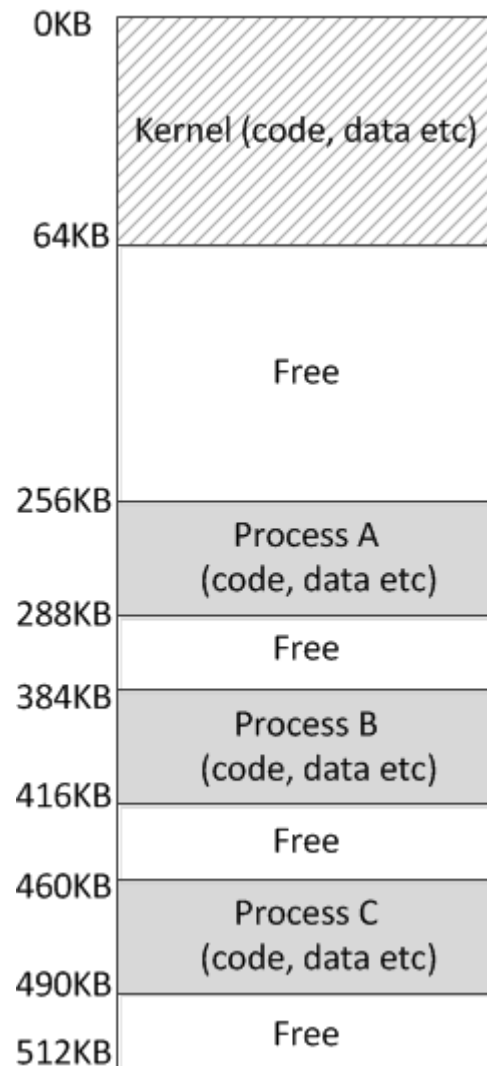


# 问题

- 如何高效地使用内存空间？



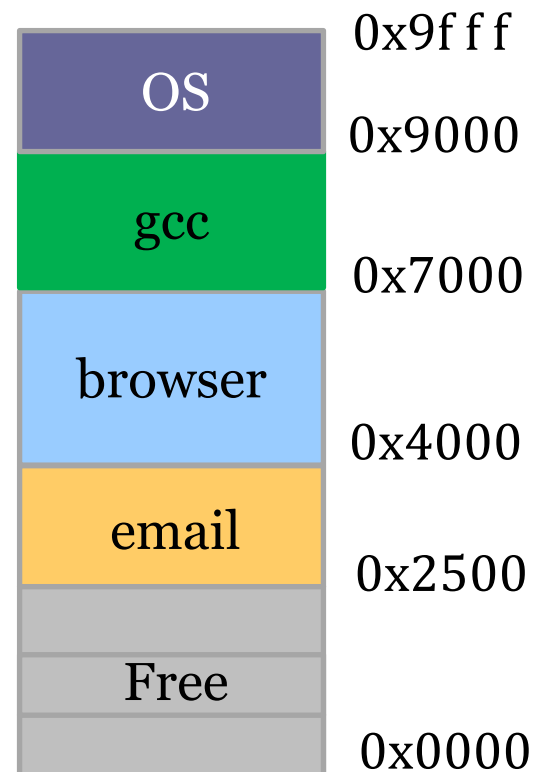
## Multiprogramming & Time sharing





# 最简单的系统

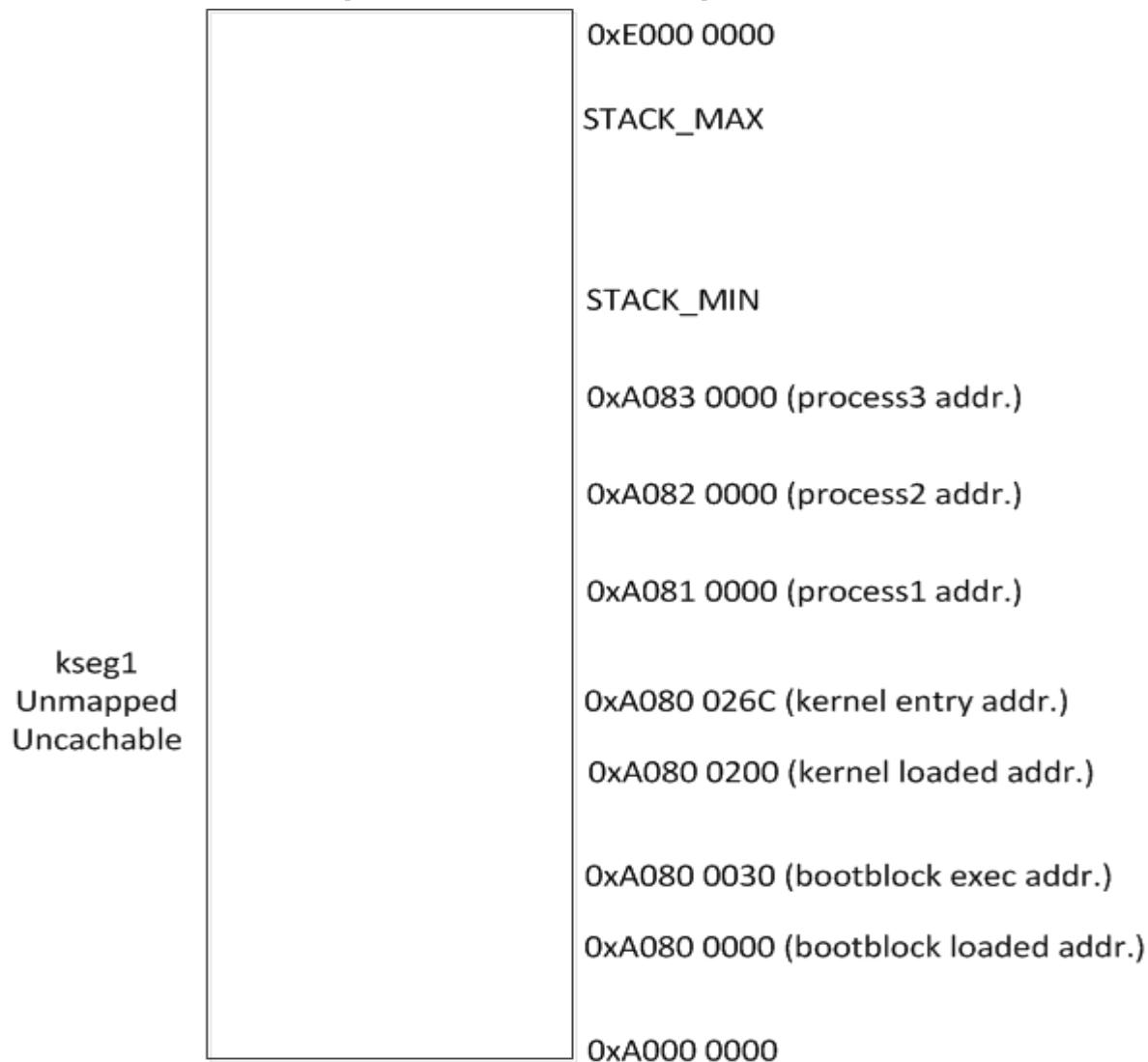
- 只有物理内存
  - 应用程序直接使用物理内存
- 例子：运行3个进程
  - Email , browser, gcc
- 可能会发生以下这些情况
  - email发生地址错误
  - browser对地址0x7050写
  - gcc需要扩展内存
  - browser需要比物理内存更多的内存





# 最简单的系统

- 物理内存静态划分（研讨课实现）

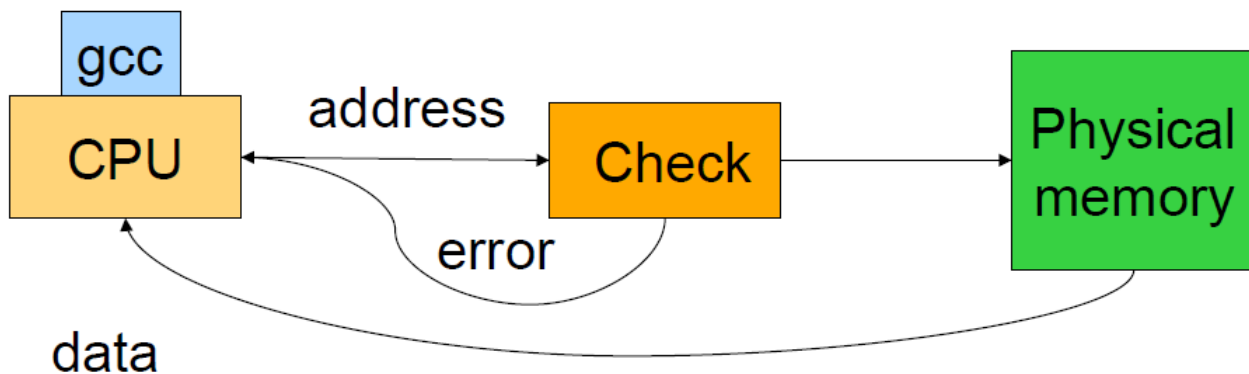






# 需求一：进程保护

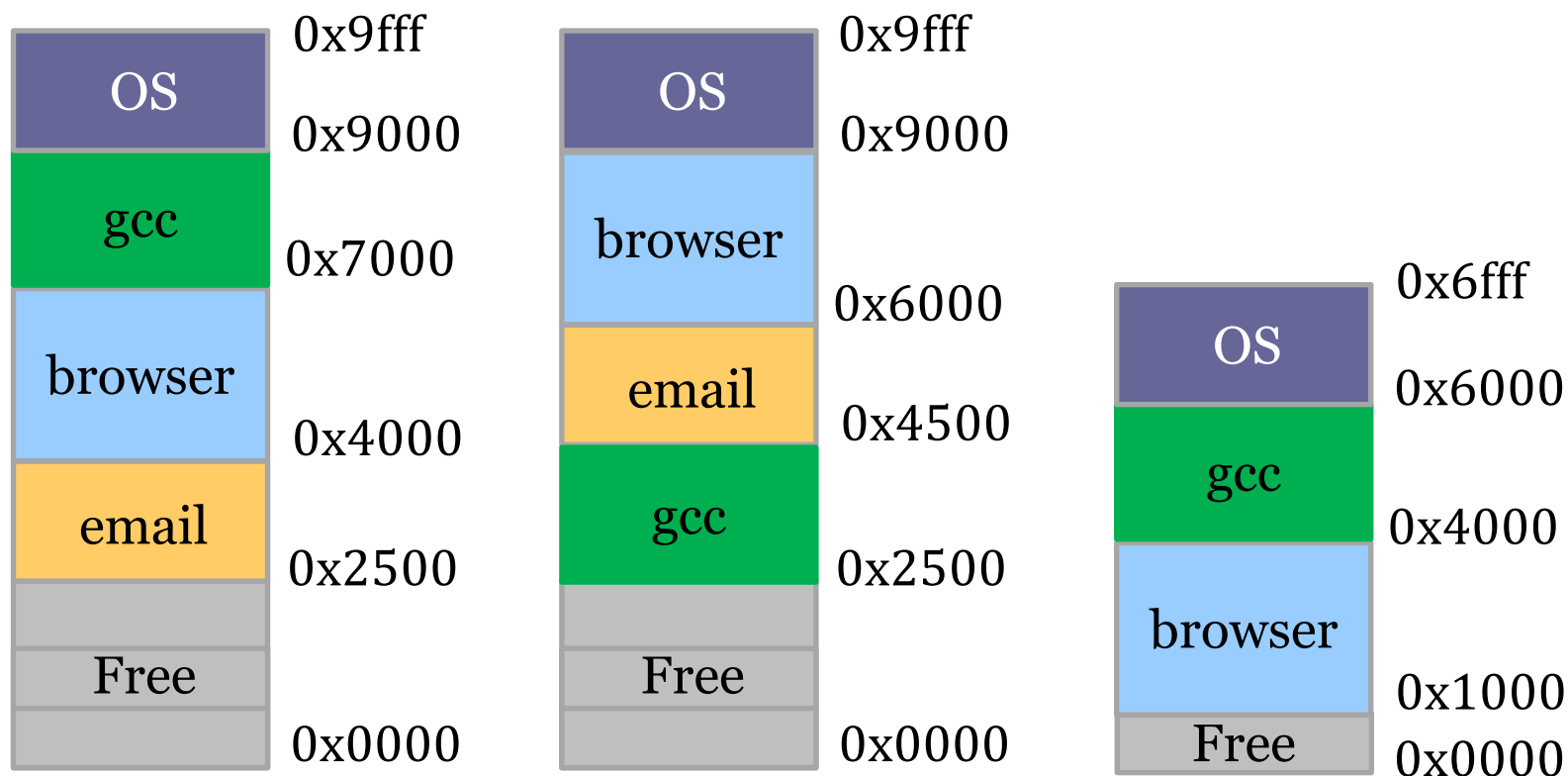
- 一个进程出错不能影响其它进程
- 对每次内存访问都进行检查，只允许合法的内存访问
  - 每个进程的每个load和store指令、隐式访存指令





## 需求二：扩展内存和应用透明

- 一个进程必须能运行在不同的物理内存区域上
- 一个进程必须能运行在不同的物理内存大小上





# 问题

- 如何高效地使用内存空间？
  - 目标1：同时运行多个进程
    - 系统运行的进程越多越好
  - 目标2：地址空间足够大
    - 一个大进程，其大小超过物理内存
    - 很多小进程，但它们的总大小超过物理内存
  - 目标3：保护
    - 一个用户进程不能读取、更不能修改另一个用户进程的内存
    - 用户进程不能破坏内核的内存



# 解决方案

- 基本内存抽象
  - 地址空间：进程的内存视图 → 虚拟内存
  - 透明使用，高效访问，安全保护
- 虚拟内存 vs 虚拟CPU
  - 虚拟CPU
    - 进程不与CPU绑定，可以迁移到任一CPU
    - 进程执行时，以为“独占”CPU
    - 进程CPU状态与切换：上下文
  - 虚拟内存
    - 进程数据不与内存绑定，可以迁移到任意的内存/磁盘位置
    - 进程执行时，以为“独占”内存
    - 进程内存状态：？



# 内容提要

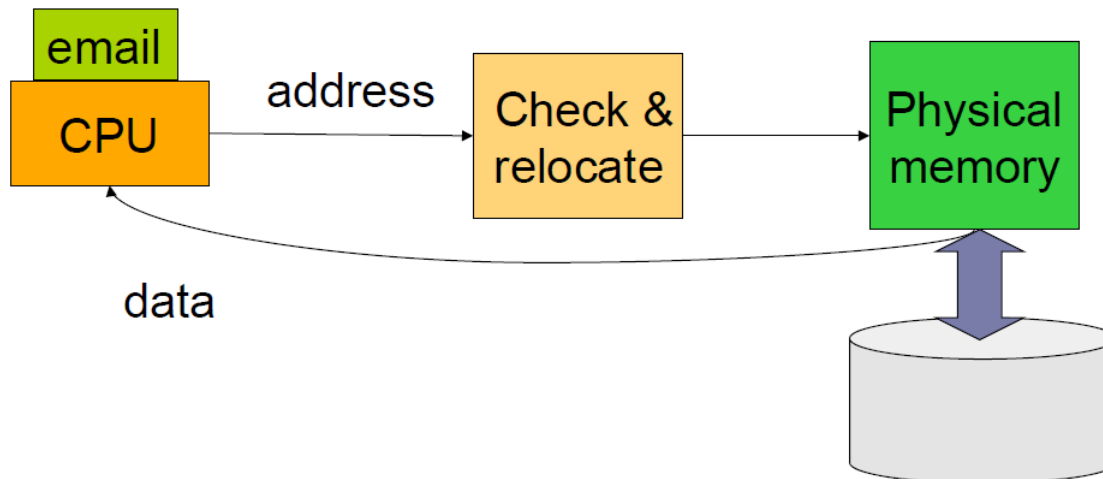
---

- 计算机存储体系结构
- 虚存
  - 虚拟化
  - 保护
- 地址映射
  - 基址+长度
  - 分段
  - 分页
- TLB



# 虚拟内存 ( Virtual Memory )

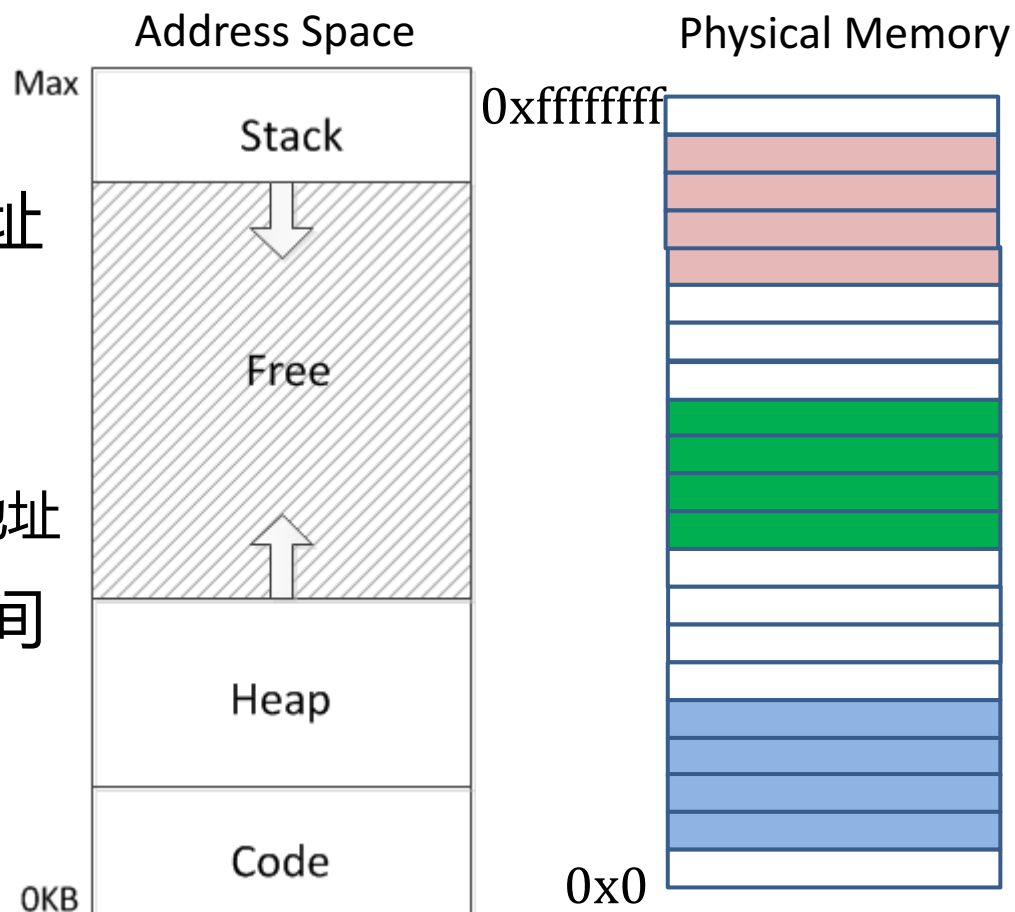
- 独立的 ( 进程 ) 地址空间
  - 给每个进程提供一个很大的、静态的 “虚拟” 地址空间
- 虚实地址转换
  - 进程运行时，每次访存通过地址转换 ( relocate ) 获得实际的物理内存地址
- 磁盘作为内存的延展 ( **磁盘交换区** )
  - 按需加载：只装载部分地址空间至内存





# 地址空间

- 独立的进程地址空间  $[0, \text{max}-1]$ 
  - 程序员看到的是虚地址
- 运行时装载部分地址空间
- 每次访存：虚地址  $\rightarrow$  物理地址
  - CPU看到的是虚地址
  - 进程看到的是虚地址
  - 内存与I/O设备看到的是物理地址
- 如果访问到未装载的地址空间
  - 通知OS将它加载进内存





# 虚存的好处

---

- 灵活
  - 进程在执行时才放进内存，一部分在内存 & 另一部分在磁盘
- 简单
  - 进程的内存访问变得非常简单
- 高效
  - 20/80原则：20%的地址空间承担80%的访问
  - 将20%地址空间放进物理内存
- 安全
  - 虚实地址转换时进行安全检查，防止非法访问





# 虚存设计

---

- 设计问题
  - 如何进行地址转换？
  - 如何划分内存？
  - 如何实施保护？



# 内容提要

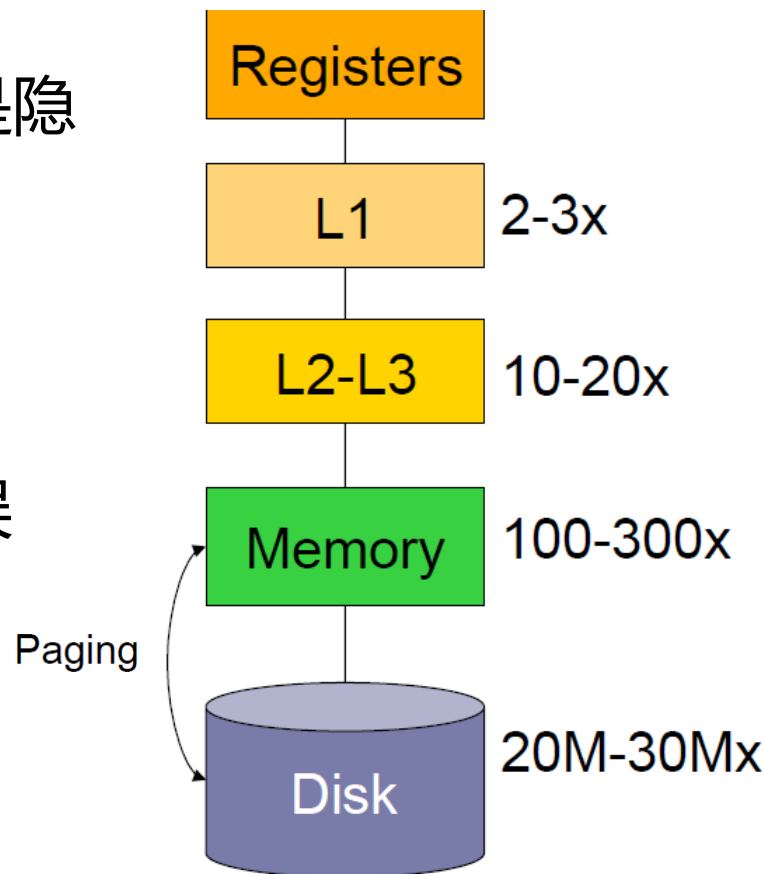
---

- 计算机存储体系结构
- 虚存
  - 虚拟化
  - 保护
- 地址映射
  - 基址+长度
  - 分段
  - 分页
- TLB



# 地址转换

- 目标
  - 隐式：对每个内存访问，转换是隐式的
  - 快速：命中内存时必须非常快
  - 例外：不命中时触发一个例外
  - 保护：能够隔离用户进程的错误





# 地址映射和粒度

- 需要某种“映射”机制
  - 把虚地址空间（大）的内容放进物理内存空间（小）
- 映射必须有合适的粒度
  - 粒度决定灵活性
  - 大粒度映射可能造成内存浪费
  - 细粒度映射需要更多的映射信息（trade-off）
- 极端情况
  - 字节粒度映射：映射表过大
  - 进程粒度映射：内存浪费



# 基址+长度 (Cray-1采用的方法)

- 连续分配

- 为每个进程分配地址连续的内存
- 用一个二元组来限定其内存区域： $\langle \text{base}, \text{bound} \rangle$

- 保护

- 一个进程只能访问  $[\text{base}, \text{base} + \text{bound}]$  区间的内存

- 上下文切换

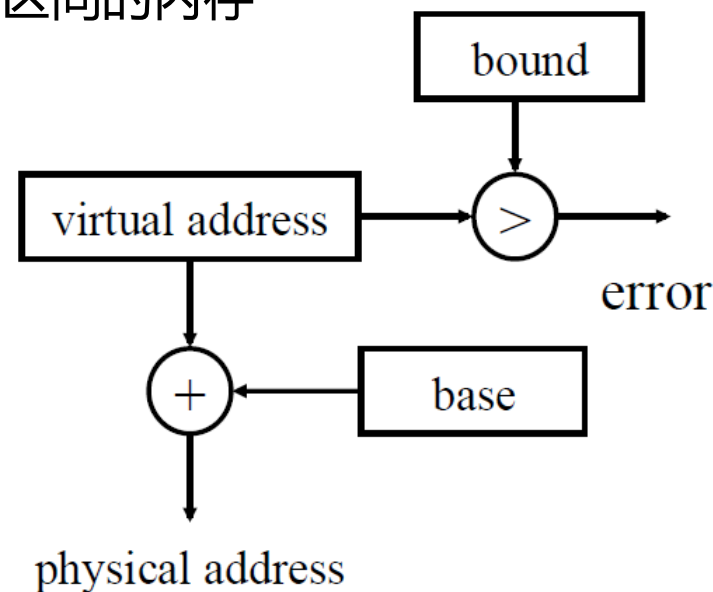
- 保存/恢复基址寄存器及上限寄存器

- 好处

- 简单：映射时将虚地址与基址相加
- 支持换出(swapping)：多进程并发执行

- 坏处

- 外部碎片（进程间的碎片）
- 难以支持进程增大
- 难以共享内存

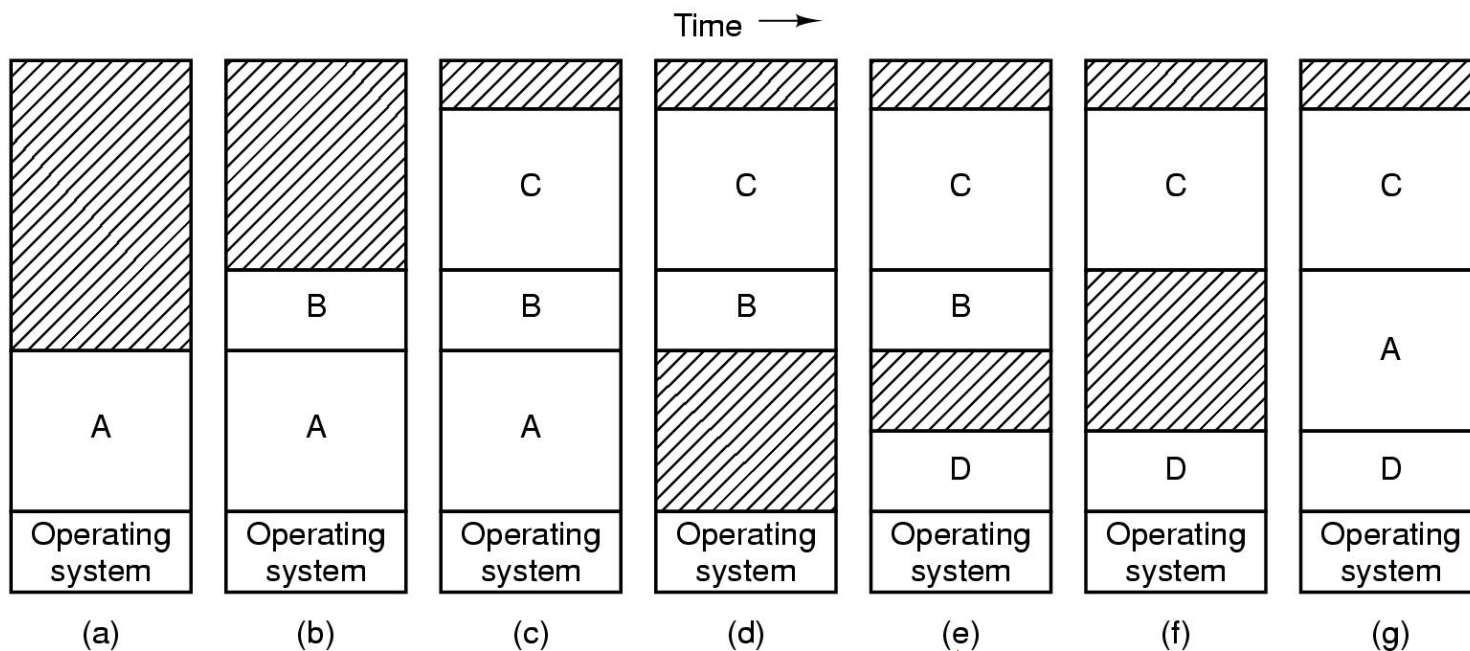




# 内存碎片

- 碎片问题

- 随着进程的换入与换出，内存产生很多空洞（未使用的小区域）



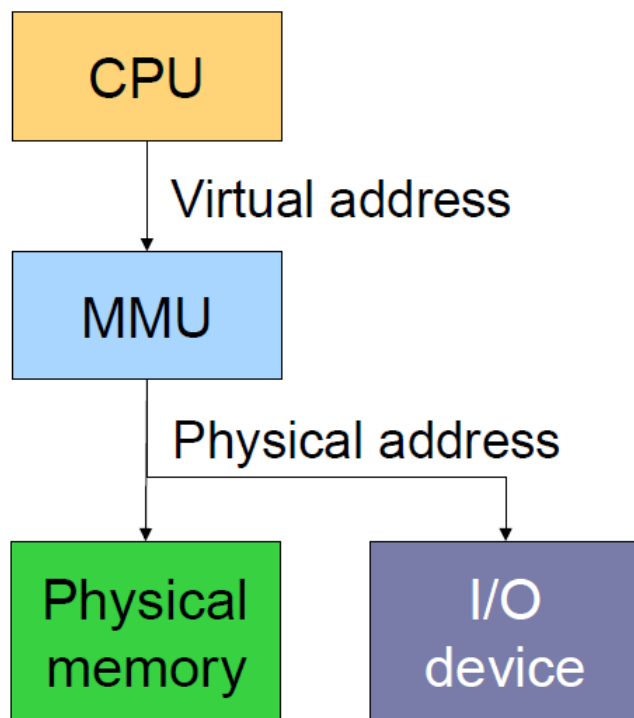
如果此时A程序要运行

- 要么把另一个进程换出，如(f)(g)所示
- 要么进行碎片聚合（memory compaction）



# 地址转换实现

- 地址类别
  - CPU发出的是虚地址
  - 内存和I/O设备接收的是物理地址
- MMU职责
  - Memory Management Unit，负责虚地址到物理地址转换的硬件单元
  - 通常在片内实现，每个CPU有一个base寄存器和一个register寄存器
  - 虚存地址转换为物理地址，每条load和store指令都需要地址转换
  - 内存保护，检查地址是否有效
  - 特殊指令操作base和bound寄存器





# 地址转换实现

- 操作系统职责
  - 内存管理
    - 新进程分配空间，结束的进程回收空间
  - 进程切换时base-bound管理
    - 保存当前进程的base-bound值，设置即将运行进程的base-bound值
  - 异常处理
    - 内存越界访问、无效地址（例如base-bound不存在）等



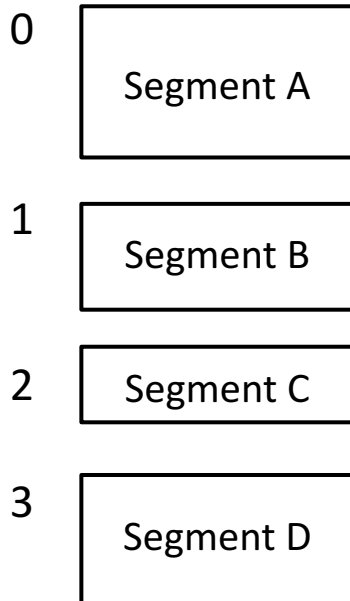


# 分段

## • 不连续分配

- 把程序逻辑上划分为若干段：代码、全局变量、栈、...
- 每个段分配连续内存，段间不必连续
- 每个进程有一张段表：(seg, size)
- 每个段采用基址+长度

### Virtual Memory



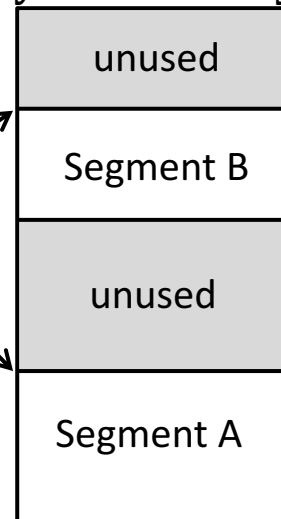
### Segment Table

valid	size	base
1	10KB	_____
1	6KB	_____
0	5KB	_____
0	8KB	_____

### Virtual address

seg #	offset
-------	--------

### Physical Memory

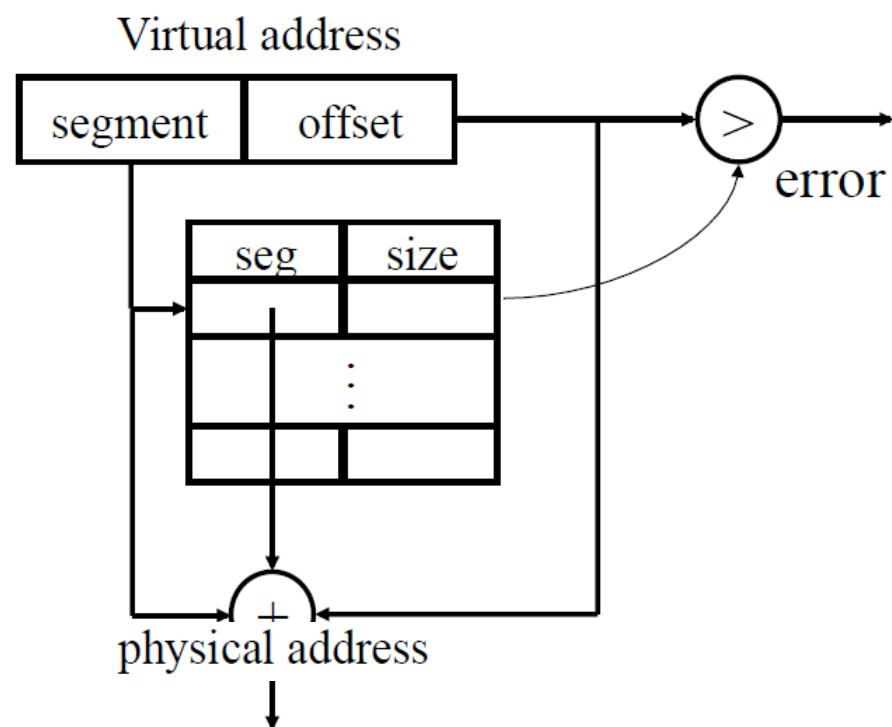


如果访问地址的offset >= 段size，则memory violation  
如果访问地址所在seg#的valid为0，则segment fault



# 分段

- 保护
  - 每个段有(nil, read, write, exec)
- 上下文切换
  - 保存/恢复 段表和指向段表的内核指针
- 好处
  - 相比基址+长度，资源使用更高效
  - 易共享
- 不足
  - 管理复杂
  - 外部碎片（段间碎片）

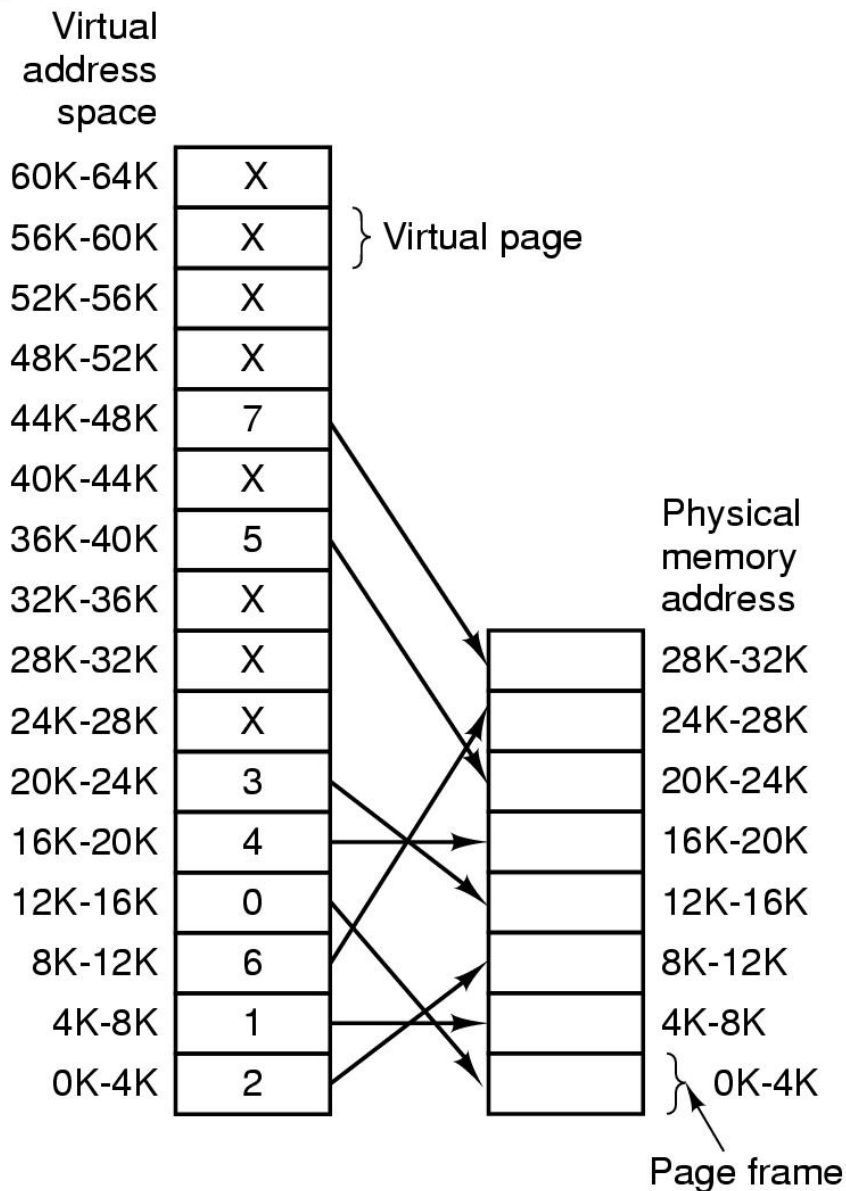




# 分页

- 分页机制

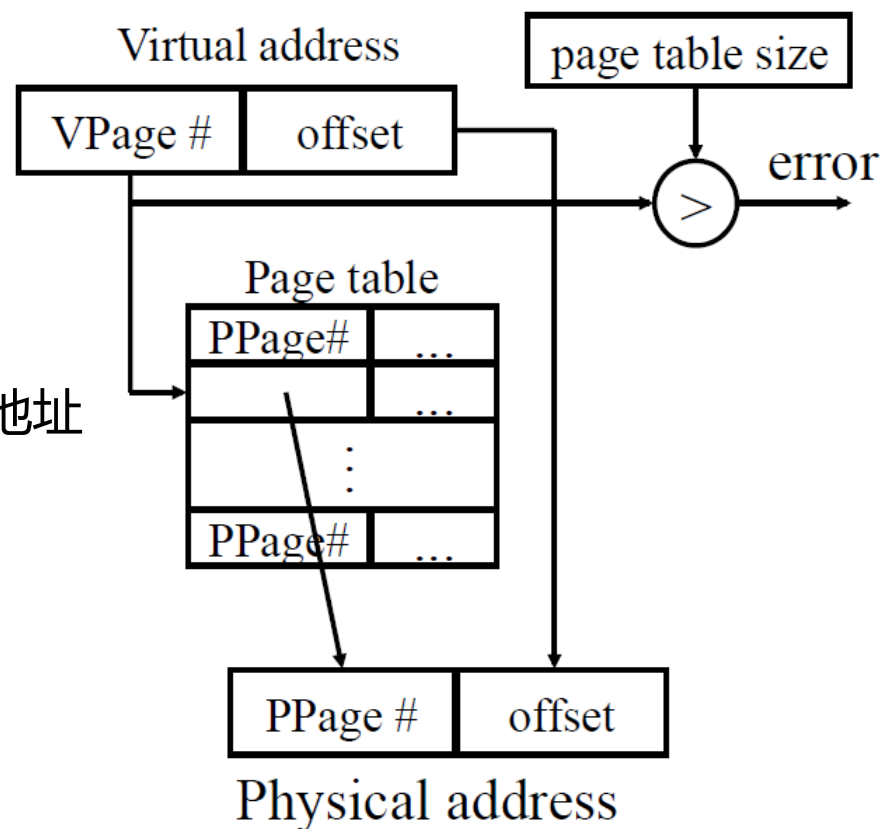
- 使用固定大小的映射单元
- 把虚存划分成固定大小的单元（称为页，page）
- 把物理内存划分成同样大小的单元（称为页框，page frame）
- 按需加载





# 分页

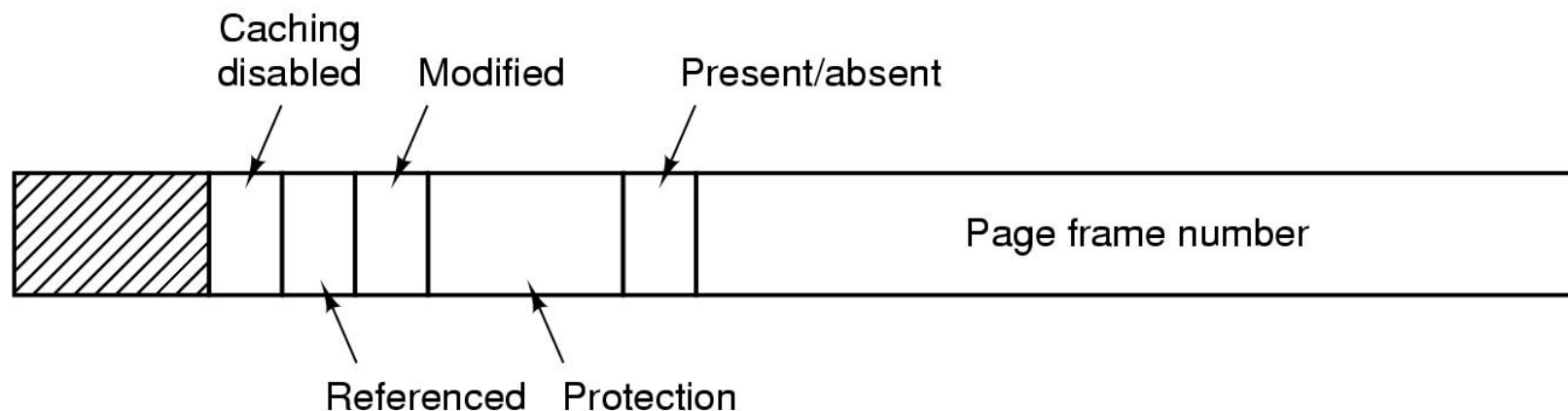
- 用**页表**来记录映射
  - 虚页 → 物理页
- 每个表项有若干控制位
  - 按页保护 ( read, write, exe)
- 上下文切换
  - 与分段类似：保存/恢复页表及其地址
- 好处
  - 分配简单
  - 易共享
- 坏处
  - 页表很大
  - 进程地址空间有很多空洞：对应的页表项无用





# 页表项

- 表达一个映射关系 ( Page Table Entry , PTE ) : 虚页号→物理页号



- PP#
- 控制位
  - 有效位(Valid) : 1位 , 标识该页在内存(Present)或不在内存(absent) , V位
  - 保护位(Protection) : 1~3位 , read, write, exe
  - 修改位(Modified) : 1位 , M位
  - 访问位(Referenced) : 1位 , R位
  - 缓存位(Cacheable) : 1位 ( 讲设备时会用到 )