

# 计算机组成和体系结构

## 第八讲

四川大学网络空间安全学院

2020年4月20日

封面来自microsemi.com

# 版权声明

---

课件中所使用的图片、视频等资源版权归原作者所有。

课件原创内容采用 [创作共用署名—非商业使用—相同方式共享4.0国际版许可证\(Creative Commons BY-NC-SA 4.0 International License\)](#) 授权使用。

Copyright@四川大学网络空间安全学院计算机组成与体系结构课程组，2020



# 上期内容回顾

---

- 掌握常见的存储器类型及特点
- 掌握存储器层次结构
  - 层次化存储系统的性能指标
- 高速缓存
  - 缓存的映射策略：直接映射、全相联映射、组相联映射
  - 缓存的替换策略

# 本期学习目标

---

- 高速缓存
  - 缓存的写策略
  - 指令和数据缓存、多级缓存
- 虚拟内存
  - 分页内存管理：页表，地址转换，TLB
  - 内存碎片和分段内存管理
- 存储器实例：Pentium处理器

# 中英文缩写对照表

---

英文缩写	英文全称	中文全称
EAT	Effective Access Time	有效访问时间
TLB	Translation Lookaside Buffer	转换后备缓冲器
MMU	Memory Management Unit	内存管理单元



# 存储器2: 高速缓存

# 缓存的写策略

---

缓存的写策略体现在两类事件：

# 缓存的写策略

---

缓存的写策略体现在两类事件：

- 脏块(dirty block)：载入到缓存中的数据块被修改，导致缓存中的块与主存中的块数据不一致

# 缓存的写策略

---

缓存的写策略体现在两类事件：

- 脏块(dirty block)：载入到缓存中的数据块被修改，导致缓存中的块与主存中的块数据不一致
- 写缺失(write miss)：写入的内存块不在缓存中

# 缓存的写策略

---

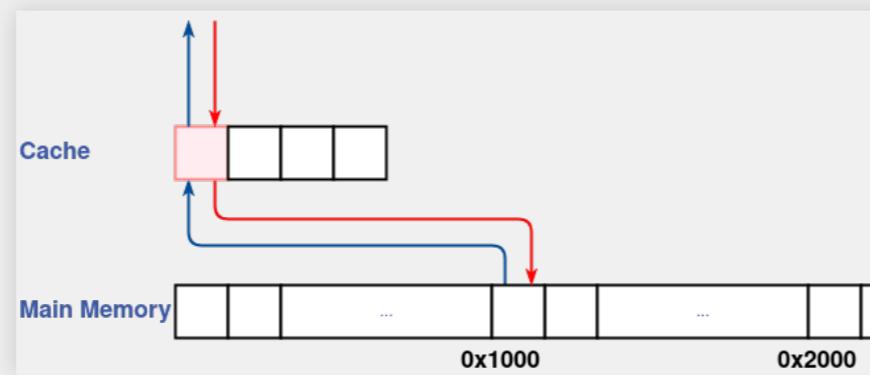
缓存的写策略体现在两类事件：

- 脏块(dirty block)：载入到缓存中的数据块被修改，导致缓存中的块与主存中的块数据不一致
- 写缺失(write miss)：写入的内存块不在缓存中

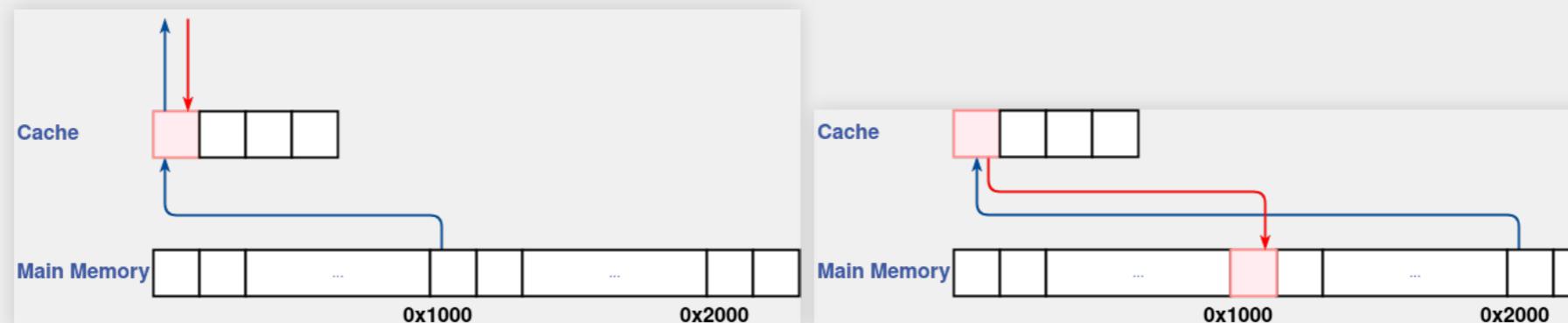
```
LOAD r0, 0x1000
STORE r0, 0x1001 ;; 脏块
STORE r0, 0x2000 ;; 写缺失
```

# 脏块写策略

- 直写策略(Write-through): 写入缓存的同时将内容写入主存

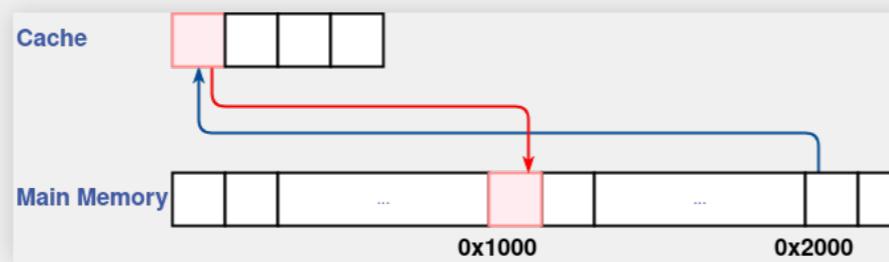


- 写回策略(Write-back): 只写入缓存块，当缓存块替换的时候写回主存

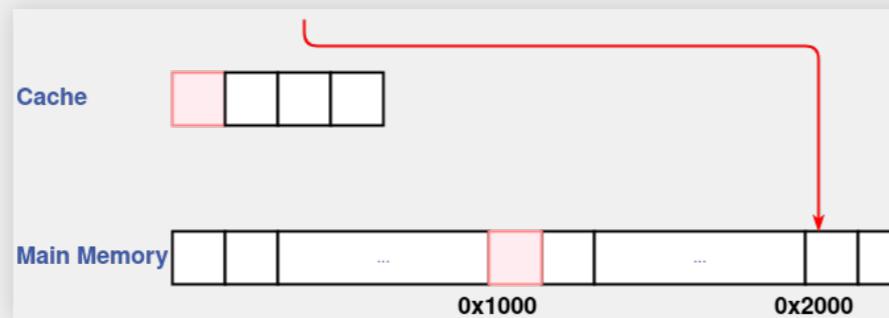


# 写缺失策略

- 写分配策略(Write-allocate): 写入之前先载入对应的块，然后采用脏块处理的策略进行写入



- 写不分配策略(No-write-Allocate): 直接写入主存



# 指令和数据缓存、多级缓存

- 哈佛缓存架构：指令和数据分开存储
- 多级缓存：
  - 包容缓存：层次较高的缓存中的内容一定在层次较低的缓存中
  - 独占缓存：同一份数据只存在一个缓存层次中
  - 非阻塞缓存：允许缓存在等待内存数据拷贝的过程中继续读取

Vendor ID:	GenuineIntel
CPU family:	6
Model:	78
Model name:	Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz
Stepping:	3
CPU MHz:	800.057
CPU max MHz:	3400.0000
CPU min MHz:	400.0000
BogoMIPS:	5602.18
Virtualization:	VT-x
L1d cache:	64 KiB
L1i cache:	64 KiB
L2 cache:	512 KiB
L3 cache:	4 MiB



# 存储器3: 虚拟内存

# 虚拟存储器

---

为什么需要虚拟存储器：

# 虚拟存储器

---

为什么需要虚拟存储器：

- 程序的可寻址地址空间大于主存的实际容量

# 虚拟存储器

---

为什么需要虚拟存储器：

- 程序的可寻址地址空间大于主存的实际容量
  - 以64位计算机为例，理论寻址空间为 $2^{64} = 16EB = 17,179,869,184GB \gg 128GB$

# 虚拟存储器

---

为什么需要虚拟存储器：

- 程序的可寻址地址空间大于主存的实际容量
  - 以64位计算机为例，理论寻址空间为 $2^{64} = 16EB = 17,179,869,184GB \gg 128GB$
- 实际使用的内存空间大于主存的实际容量

# 虚拟存储器

---

为什么需要虚拟存储器：

- 程序的可寻址地址空间大于主存的实际容量
  - 以64位计算机为例，理论寻址空间为 $2^{64} = 16EB = 17,179,869,184GB \gg 128GB$
- 实际使用的内存空间大于主存的实际容量
  - 例如多任务系统上执行了多个消耗大内存的应用，如图形处理、游戏等

# 虚拟存储器

---

为什么需要虚拟存储器：

- 程序的可寻址地址空间大于主存的实际容量
  - 以64位计算机为例，理论寻址空间为 $2^{64} = 16EB = 17,179,869,184GB \gg 128GB$
- 实际使用的内存空间大于主存的实际容量
  - 例如多任务系统上执行了多个消耗大内存的应用，如图形处理、游戏等
- 多个进程的地址空间重叠

# 虚拟存储器

---

为什么需要虚拟存储器：

- 程序的可寻址地址空间大于主存的实际容量
  - 以64位计算机为例，理论寻址空间为 $2^{64} = 16EB = 17,179,869,184GB \gg 128GB$
- 实际使用的内存空间大于主存的实际容量
  - 例如多任务系统上执行了多个消耗大内存的应用，如图形处理、游戏等
- 多个进程的地址空间重叠
  - 每个进程理论上都可以使用全部内存地址空间

# 虚拟存储器

---

为什么需要虚拟存储器：

- 程序的可寻址地址空间大于主存的实际容量
  - 以64位计算机为例，理论寻址空间为 $2^{64} = 16EB = 17,179,869,184GB \gg 128GB$
- 实际使用的内存空间大于主存的实际容量
  - 例如多任务系统上执行了多个消耗大内存的应用，如图形处理、游戏等
- 多个进程的地址空间重叠
  - 每个进程理论上都可以使用全部内存地址空间
- 虚拟存储器的核心思想：将一部分数据存储在硬盘上，提高内存的存储能力

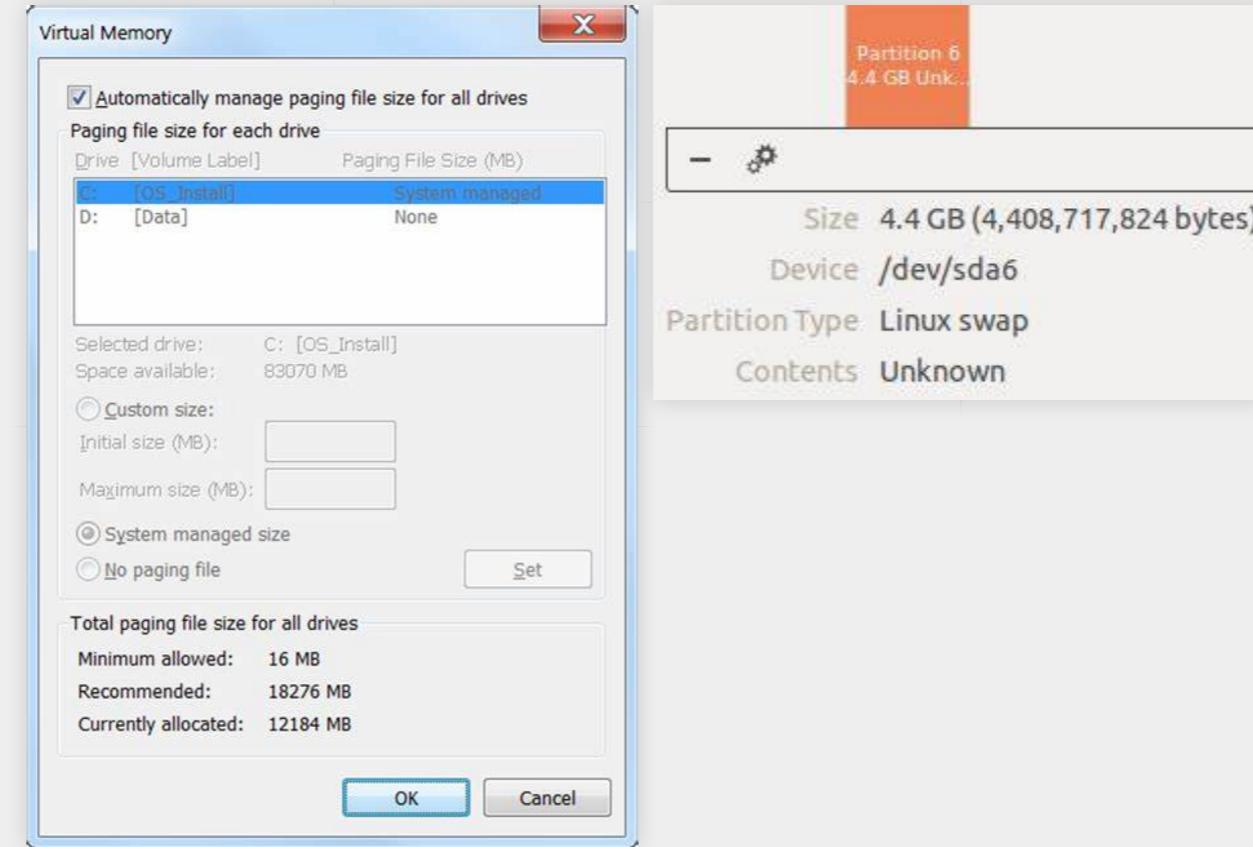
# 虚拟存储器

---

为什么需要虚拟存储器：

- 程序的可寻址地址空间大于主存的实际容量
  - 以64位计算机为例，理论寻址空间为 $2^{64} = 16EB = 17,179,869,184GB \gg 128GB$
- 实际使用的内存空间大于主存的实际容量
  - 例如多任务系统上执行了多个消耗大内存的应用，如图形处理、游戏等
- 多个进程的地址空间重叠
  - 每个进程理论上都可以使用全部内存地址空间
- 虚拟存储器的核心思想：将一部分数据存储在硬盘上，提高内存的存储能力
  - 存储在硬盘上的部分称为**页面文件**(page file)

# 不同操作系统上的页面文件



图片来源：<https://www.howtogeek.com/126430/htg-explains-what-is-the-windows-page-file-and-should-you-disable-it/>, <http://askubuntu.com/questions/485661/swap-not-available>

# 虚拟存储器的层次化存储结构

---

高速缓存		虚拟存储器
上层存储器	高速缓存	主存
低层存储器	主存	硬盘
存储单元	块	
上层地址	块号+块内偏移	
低层地址	内存地址	
映射机制	内存地址转为缓存中的地址	
失效	缓存失效	
上层已满	缓存块替换	

# 虚拟存储器的层次化存储结构

---

	高速缓存	虚拟存储器
上层存储器	高速缓存	主存
低层存储器	主存	硬盘
存储单元	块	页
上层地址	块号+块内偏移	
低层地址	内存地址	
映射机制	内存地址转为缓存中的地址	
失效	缓存失效	
上层已满	缓存块替换	

# 虚拟存储器的层次化存储结构

---

	高速缓存	虚拟存储器
上层存储器	高速缓存	主存
低层存储器	主存	硬盘
存储单元	块	页
上层地址	块号+块内偏移	物理地址
低层地址	内存地址	
映射机制	内存地址转为缓存中的地址	
失效	缓存失效	
上层已满	缓存块替换	

# 虚拟存储器的层次化存储结构

---

	高速缓存	虚拟存储器
上层存储器	高速缓存	主存
低层存储器	主存	硬盘
存储单元	块	页
上层地址	块号+块内偏移	物理地址
低层地址	内存地址	虚拟地址
映射机制	内存地址转为缓存中的地址	
失效	缓存失效	
上层已满	缓存块替换	

# 虚拟存储器的层次化存储结构

高速缓存		虚拟存储器
上层存储器	高速缓存	主存
低层存储器	主存	硬盘
存储单元	块	页
上层地址	块号+块内偏移	物理地址
低层地址	内存地址	虚拟地址
映射机制	内存地址转为缓存中的地址	虚拟地址转换为物理地址
失效	缓存失效	
上层已满	缓存块替换	

# 虚拟存储器的层次化存储结构

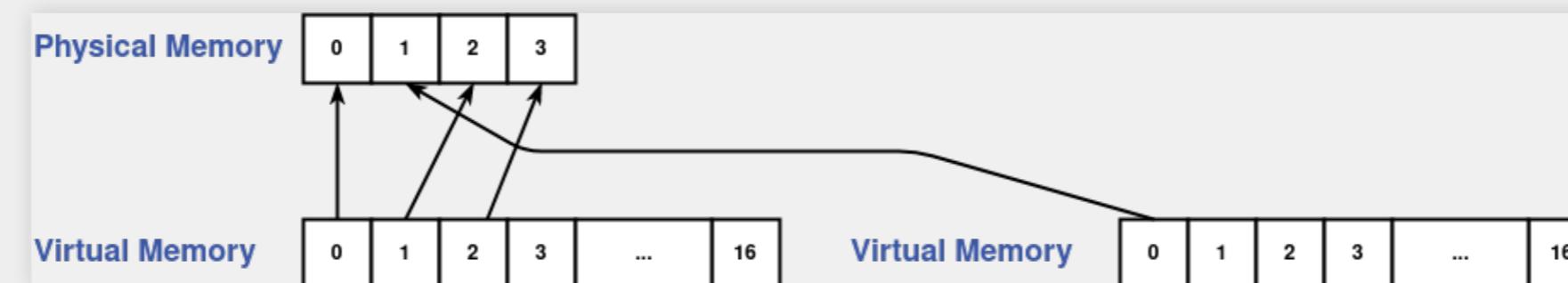
高速缓存		虚拟存储器
上层存储器	高速缓存	主存
低层存储器	主存	硬盘
存储单元	块	页
上层地址	块号+块内偏移	物理地址
低层地址	内存地址	虚拟地址
映射机制	内存地址转为缓存中的地址	虚拟地址转换为物理地址
失效	缓存失效	页面故障
上层已满	缓存块替换	

# 虚拟存储器的层次化存储结构

高速缓存		虚拟存储器
上层存储器	高速缓存	主存
低层存储器	主存	硬盘
存储单元	块	页
上层地址	块号+块内偏移	物理地址
低层地址	内存地址	虚拟地址
映射机制	内存地址转为缓存中的地址	虚拟地址转换为物理地址
失效	缓存失效	页面故障
上层已满	缓存块替换	页替换

# 分页

- 将每个进程的虚拟内存和系统物理内存划分为固定大小的数据单元，称为**页**(Page)
- 在物理内存中仅保存一部分进程的一部分页
- 虚拟内存页(简称虚页)需要经过地址转换为物理内存中的页
- 物理内存中的页称为页面(Frame/Page Frame)



# 页表

- **页表**保存了每个进程的虚拟内存页与物理内存页的地址转换关系
  - 每个进程有独立的页表
  - 有效位(valid bit): 0表示虚页不在主存中, 1表示在主存中
  - 脏位(dirty bit)/修改位(modified bit): 1表示该页在内存中被修改, 0表示未修改
  - 使用位(reference bit): 1表示该页最近被访问过, 0表示未访问过
- 页表通常保存在内存中



# 地址转换

---

- 虚拟地址分为两个字段
  - 页编号：该页在虚拟内存中的编号
  - 页内偏移值：该地址在虚拟内存中对应页内的偏移值
- 物理地址分为两个字段
  - 页面编号：该页在物理内存中对应的页面编号
  - 页内偏移值：该地址在对应页面中的偏移值

# 地址转换

---

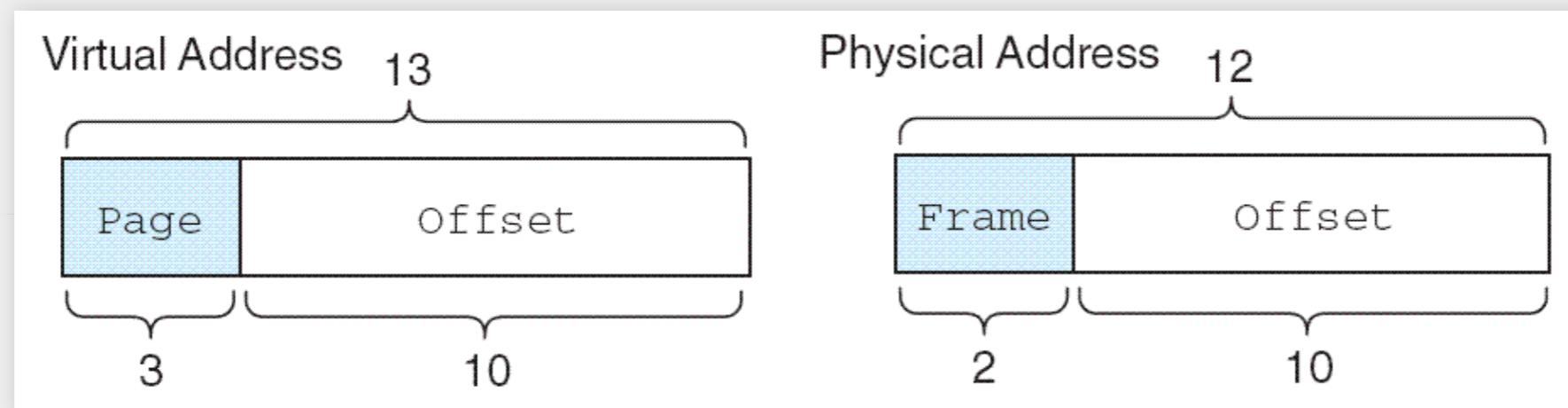
- 虚拟地址分为两个字段
  - 页编号：该页在虚拟内存中的编号
  - 页内偏移值：该地址在虚拟内存中对应页内的偏移值
- 物理地址分为两个字段
  - 页面编号：该页在物理内存中对应的页面编号
  - 页内偏移值：该地址在对应页面中的偏移值

如何确定各字段的大小？

# 地址转换示例1

例：假设一个按字节寻址的系统支持13位虚拟地址，物理内存采用12位地址，页面大小为1024B

- 虚拟地址共13位
  - 页内偏移值长度： $\log_2(1024) = 10$ 位
  - 页编号长度： $13 - 10 = 3$ 位
- 物理地址共12位
  - 页内偏移值长度： $\log_2(1024) = 10$ 位
  - 页面编号长度： $12 - 10 = 2$ 位



# 地址转换示例2

例：假设一个按字节寻址的13位系统，物理内存大小为4KB，页面大小为1KB，且进程1的页表如下，请问下列虚拟地址对应的物理内存地址分别是多少：

- 0x1553
- 0x0802
- 0x1004

Page	Page Table		Addresses		
	Frame	Valid Bit	Page	Base 10	Base 16
0	-	0	0 :	0 - 1023	0 - 3FF
1	3	1	1 :	1024 - 2047	400 - 7FF
2	0	1	2 :	2048 - 3071	800 - BFF
3	-	0	3 :	3072 - 4095	C00 - FFF
4	-	0	4 :	4096 - 5119	1000 - 13FF
5	1	1	5 :	5120 - 6143	1400 - 17FF
6	2	1	6 :	6144 - 7167	1800 - 1BFF
7	-	0	7 :	7168 - 8191	1C00 - 1FFF

# 地址转换示例2

例：假设一个按字节寻址的13位系统，物理内存大小为4KB，页面大小为1KB，且进程1的页表如下，请问下列虚拟地址对应的物理内存地址分别是多少：

- $0x1553 \rightarrow 0x0553$
- $0x0802$
- $0x1004$

Page	Page Table		Addresses		
	Frame	Valid Bit	Page	Base 10	Base 16
0	-	0	0 :	0 - 1023	0 - 3FF
1	3	1	1 :	1024 - 2047	400 - 7FF
2	0	1	2 :	2048 - 3071	800 - BFF
3	-	0	3 :	3072 - 4095	C00 - FFF
4	-	0	4 :	4096 - 5119	1000 - 13FF
5	1	1	5 :	5120 - 6143	1400 - 17FF
6	2	1	6 :	6144 - 7167	1800 - 1BFF
7	-	0	7 :	7168 - 8191	1C00 - 1FFF

- 虚拟地址 $0x1553 = (1\ 0101\ 0101\ 0011) = (101\mid 01\ 0101\ 0011)$
- 页表中(101)=5对应表项有效位为1,且页面编号为1
- 物理地址 $(01\mid 01\ 0101\ 0011) = (0101\ 0101\ 0011) = 0x0553$

# 地址转换示例2

例：假设一个按字节寻址的13位系统，物理内存大小为4KB，页面大小为1KB，且进程1的页表如下，请问下列虚拟地址对应的物理内存地址分别是多少：

- $0x1553 \rightarrow 0x0553$
- $0x0802 \rightarrow 0x0002$
- $0x1004$

Page	Page Table		Addresses		
	Frame	Valid Bit	Page	Base 10	Base 16
0	-	0	0 :	0 - 1023	0 - 3FF
1	3	1	1 :	1024 - 2047	400 - 7FF
2	0	1	2 :	2048 - 3071	800 - BFF
3	-	0	3 :	3072 - 4095	C00 - FFF
4	-	0	4 :	4096 - 5119	1000 - 13FF
5	1	1	5 :	5120 - 6143	1400 - 17FF
6	2	1	6 :	6144 - 7167	1800 - 1BFF
7	-	0	7 :	7168 - 8191	1C00 - 1FFF

- 虚拟地址 $0x0802 = (0\ 1000\ 0000\ 0010) = (010\mid 00\ 0000\ 0010)$
- 页表中(010)=2对应表项有效位为1,且页面编号为0
- 物理地址 $(00\mid 00\ 0000\ 0010) = (0000\ 0000\ 0010) = 0x0002$

# 地址转换示例2

例：假设一个按字节寻址的13位系统，物理内存大小为4KB，页面大小为1KB，且进程1的页表如下，请问下列虚拟地址对应的物理内存地址分别是多少：

- $0x1553 \rightarrow 0x0553$
- $0x0802 \rightarrow 0x0002$
- $0x1004$  发生页面故障(Page Fault)

Page	Page Table		Addresses		
	Frame	Valid Bit	Page	Base 10	Base 16
0	-	0	0 :	0 - 1023	0 - 3FF
1	3	1	1 :	1024 - 2047	400 - 7FF
2	0	1	2 :	2048 - 3071	800 - BFF
3	-	0	3 :	3072 - 4095	C00 - FFF
4	-	0	4 :	4096 - 5119	1000 - 13FF
5	1	1	5 :	5120 - 6143	1400 - 17FF
6	2	1	6 :	6144 - 7167	1800 - 1BFF
7	-	0	7 :	7168 - 8191	1C00 - 1FFF

- 虚拟地址 $0x1004 = (1\ 0000\ 0000\ 0100) = (100 | 00\ 0000\ 0100)$
- 页表中(100)=4对应表项有效位为0, 页不在内存中, 发生页面故障

# 分页内存管理的有效访问时间

---

- 访问一次虚拟内存需要**两次**内存访问：第一次访问页表，第二次访问物理地址
- 假设访问主存时间为200ns，发生页面失效的处理时间为10ms
- 当页面失效率为1%时

$$ETA = 0.99 \times (200\text{ns} + 200\text{ns}) + 0.01 \times 10\text{ms} = 100,396\text{ns}$$

- 当页面失效率为0%时

$$ETA = 0.99 \times (200\text{ns} + 200\text{ns}) = 400\text{ns}$$

# 分页内存管理的有效访问时间

---

- 访问一次虚拟内存需要**两次**内存访问：第一次访问页表，第二次访问物理地址
- 假设访问主存时间为200ns，发生页面失效的处理时间为10ms
- 当页面失效率为1%时

$$ETA = 0.99 \times (200\text{ns} + 200\text{ns}) + 0.01 \times 10\text{ms} = 100,396\text{ns}$$

- 当页面失效率为0%时

$$ETA = 0.99 \times (200\text{ns} + 200\text{ns}) = 400\text{ns}$$

即使在理想状态下，加入了虚拟内存之后数据的访问时间增加了一倍

# 转换后备缓冲器

---

- 为了减少访问内存的时间：使用高速缓存
- 为了减少访问页表的开销：

# 转换后备缓冲器

---

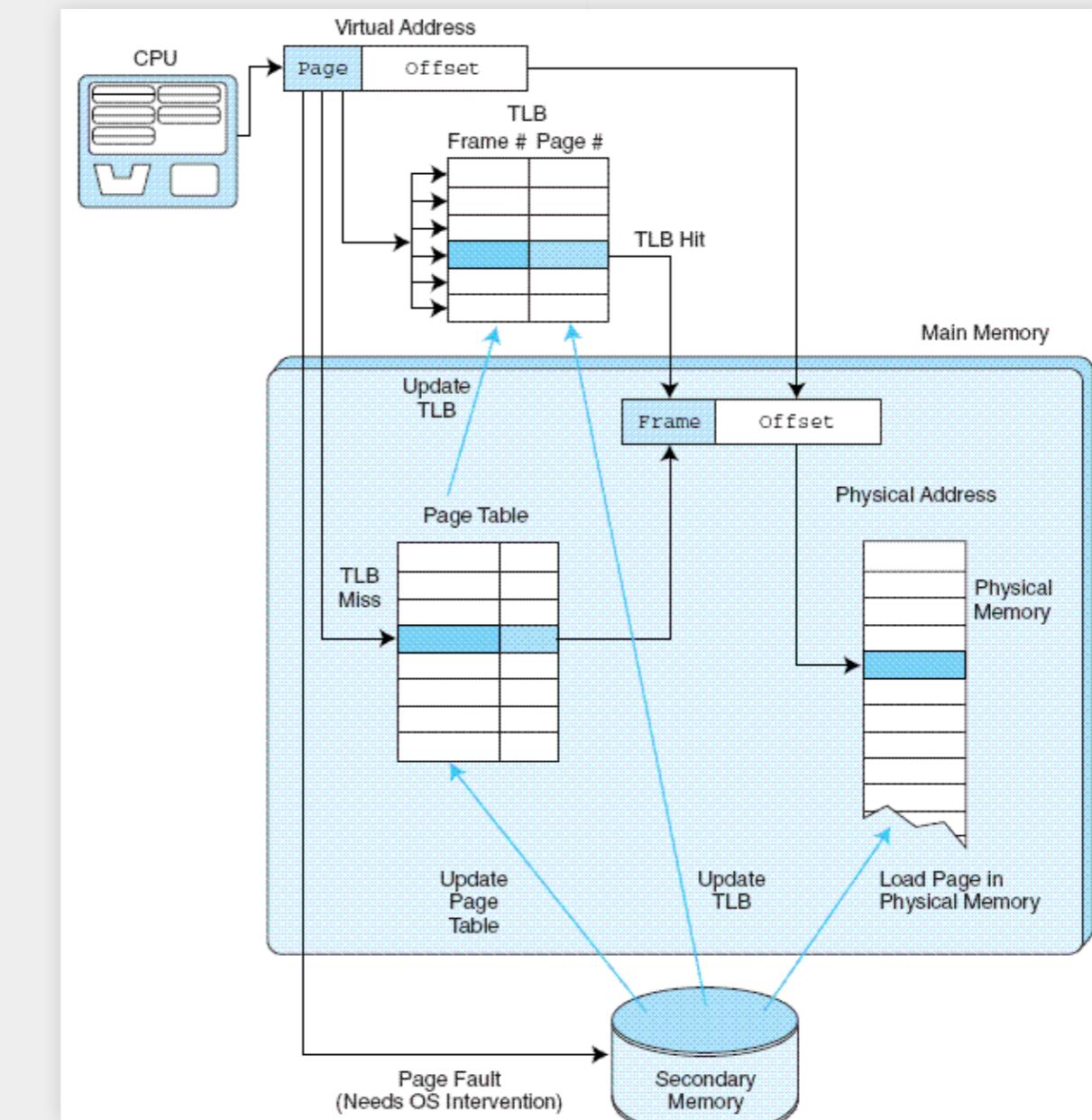
- 为了减少访问内存的时间：使用高速缓存
- 为了减少访问页表的开销：使用转换后备缓冲器(Translation Lookaside Buffer, TLB)

# 转换后备缓冲器

---

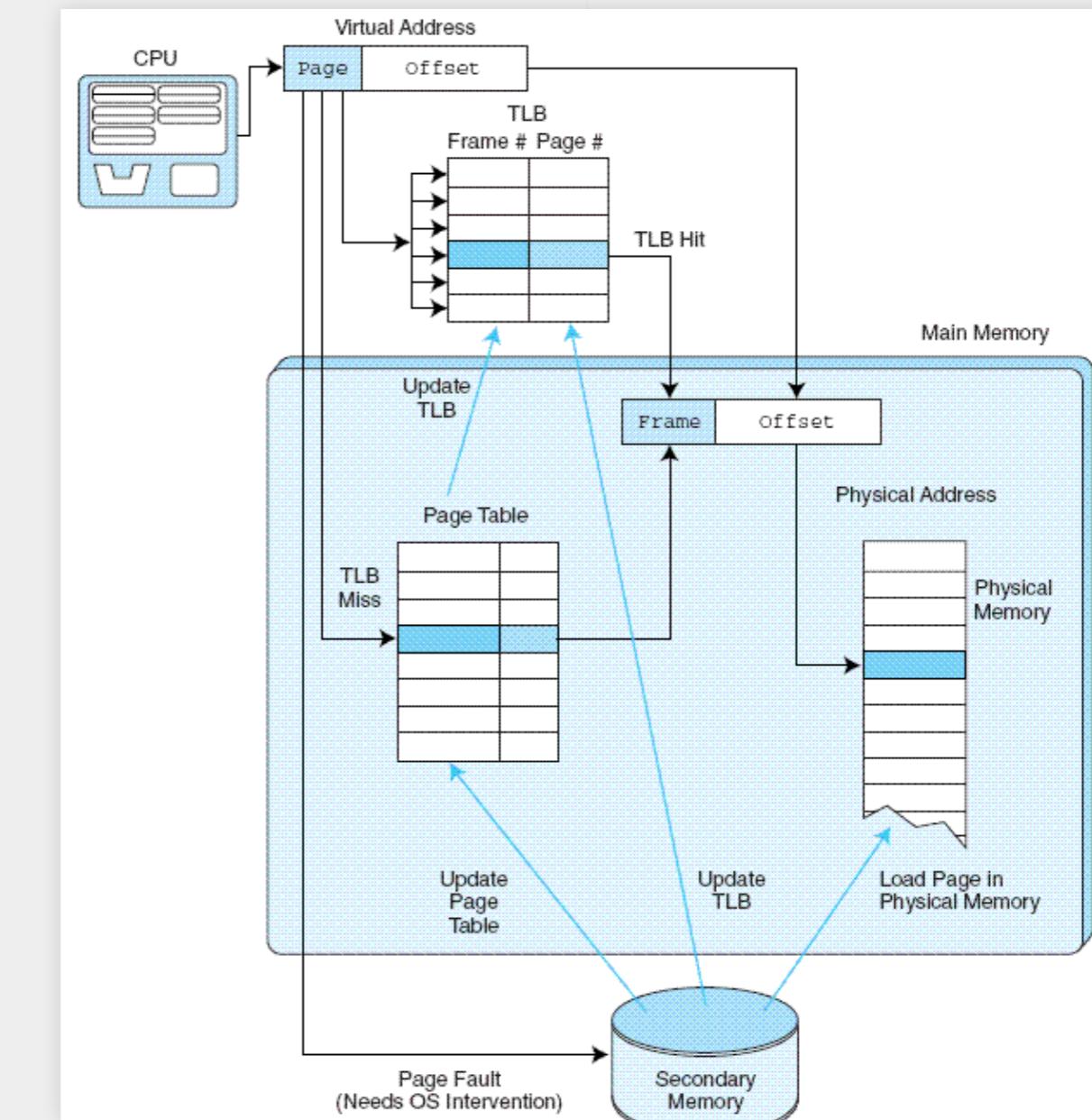
- 为了减少访问内存的时间：使用高速缓存
- 为了减少访问页表的开销：使用转换后备缓冲器(Translation Lookaside Buffer, TLB)
- TLB的本质就是将**当前进程**的页表放在缓存中，从而通过层次化存储结构提高访问页表的速度
- TLB是一类特殊的缓存：保存的是虚拟页编号到物理页编号的映射
- TLB通常采用相联缓存实现

# 使用TLB的页面访问过程



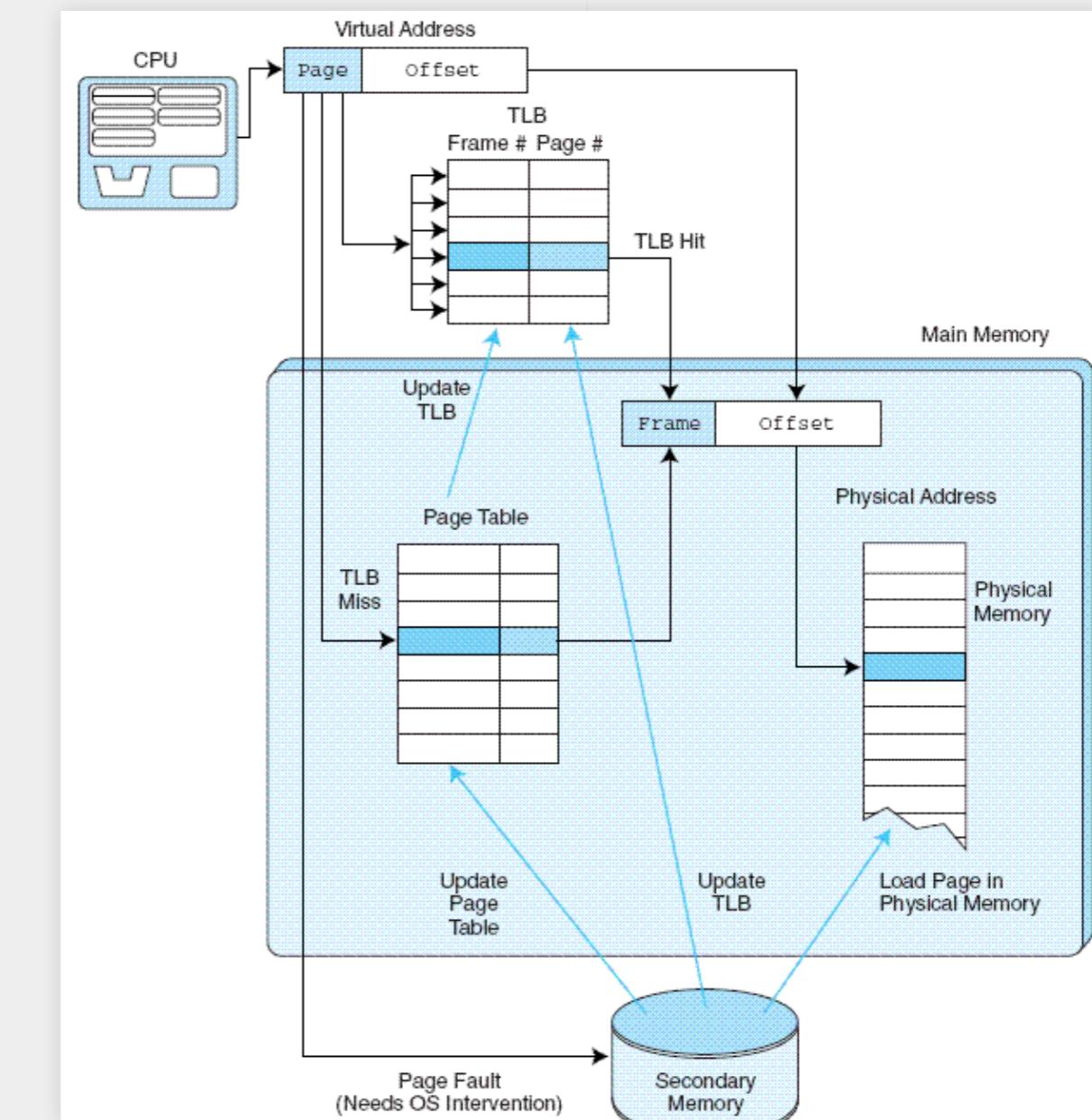
# 使用TLB的页面访问过程

- 从虚拟地址中提取出页号和偏移值



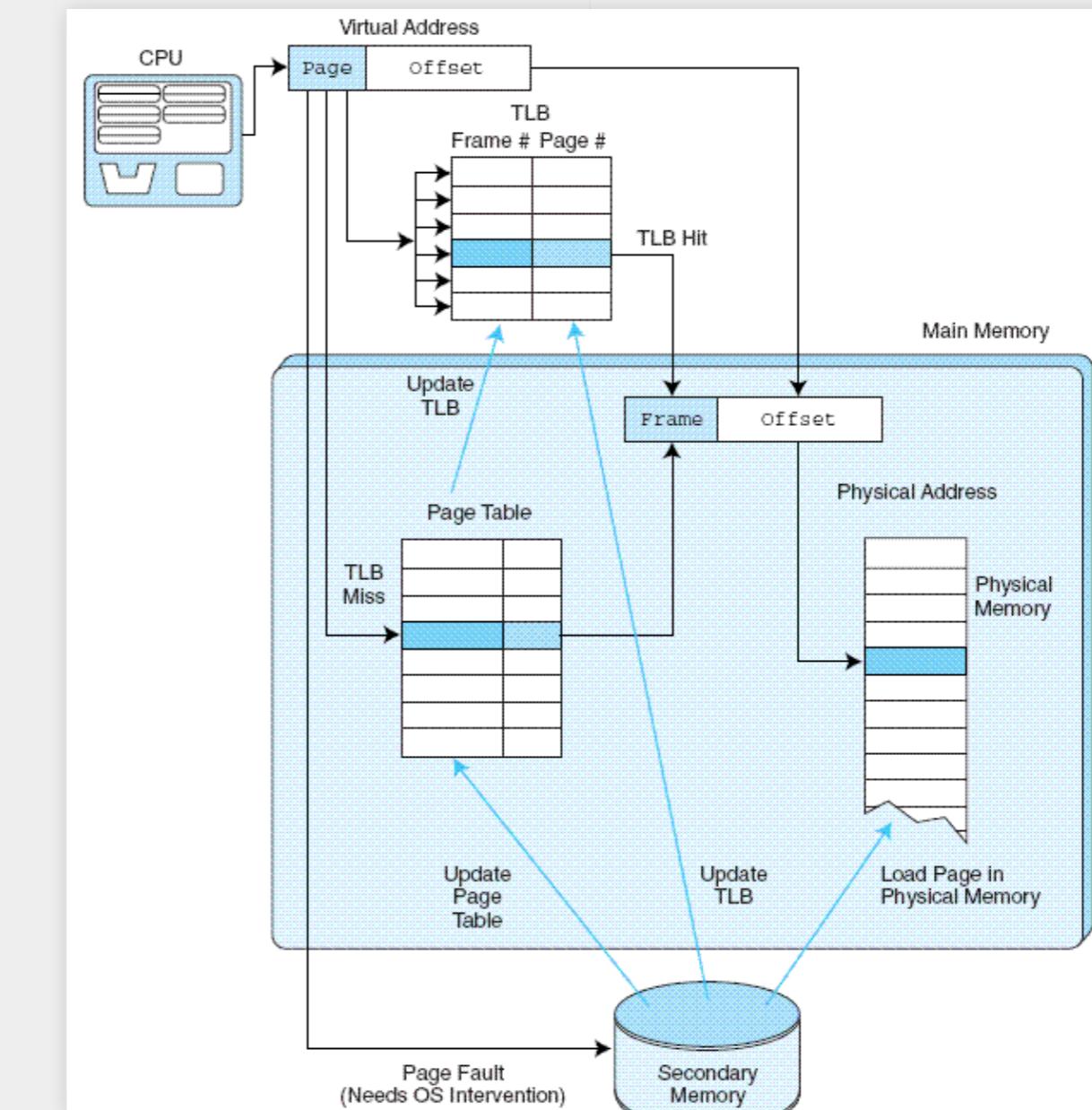
# 使用TLB的页面访问过程

- 从虚拟地址中提取出页号和偏移值
- 在TLB中查找虚拟页号对应的表项



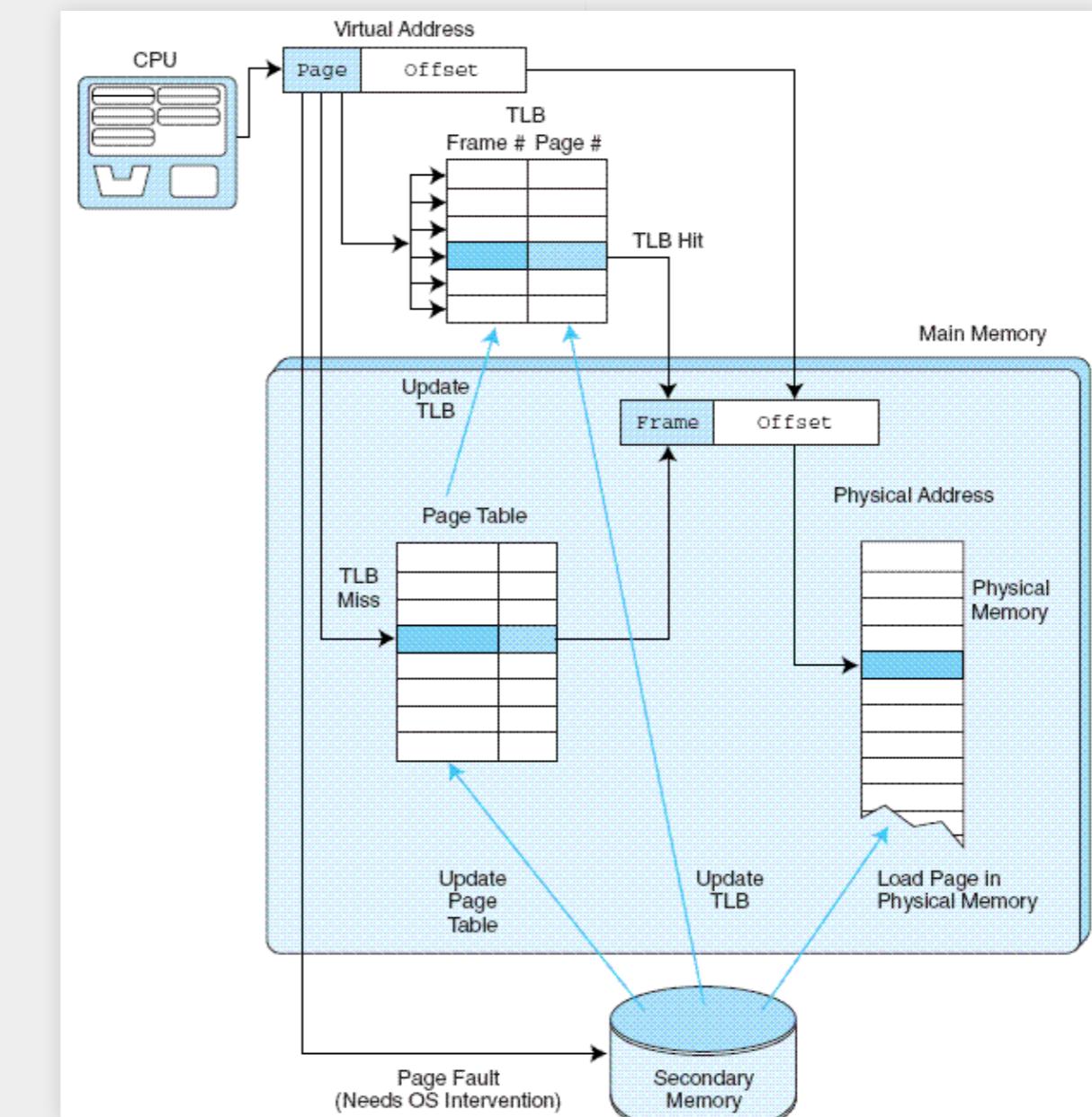
# 使用TLB的页面访问过程

- 从虚拟地址中提取出页号和偏移值
- 在TLB中查找虚拟页号对应的表项
- 如果找到，则用对应的物理页号和偏移值生成物理地址



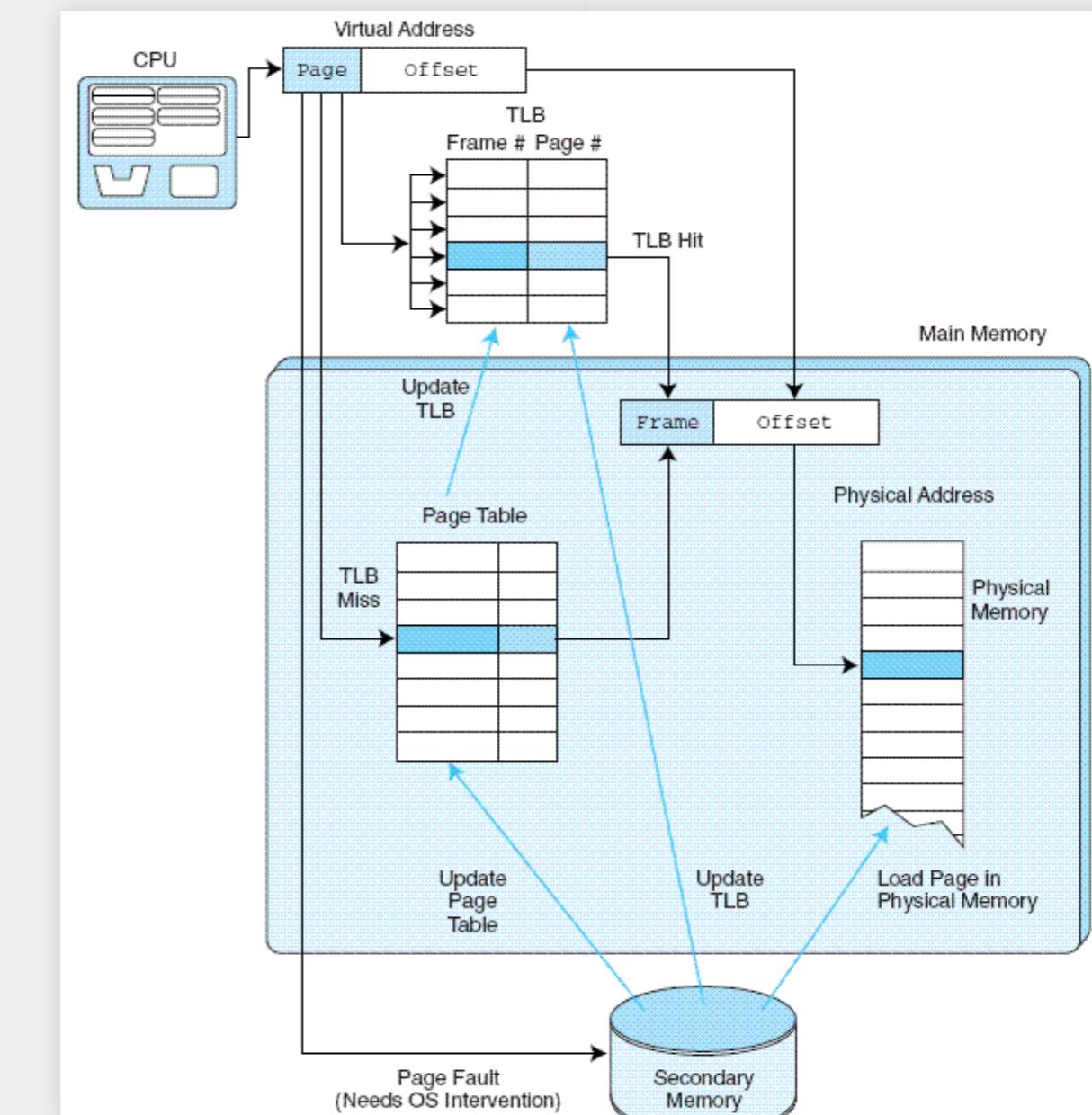
# 使用TLB的页面访问过程

- 从虚拟地址中提取出页号和偏移值
- 在TLB中查找虚拟页号对应的表项
- 如果找到，则用对应的物理页号和偏移值生成物理地址
- 如果未找到，需要访问内存中的页表



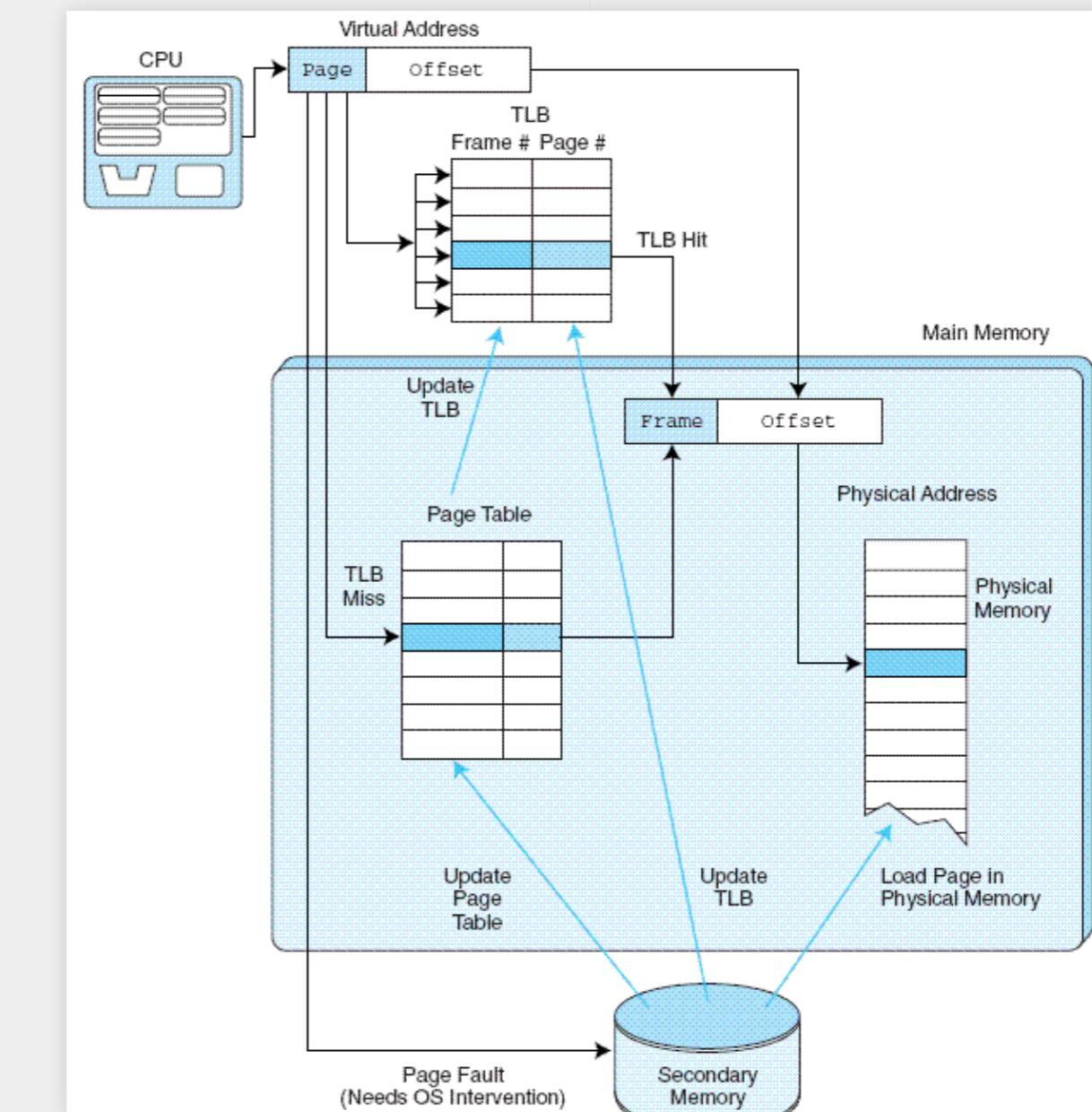
# 使用TLB的页面访问过程

- 从虚拟地址中提取出页号和偏移值
- 在TLB中查找虚拟页号对应的表项
- 如果找到，则用对应的物理页号和偏移值生成物理地址
- 如果未找到，需要访问内存中的页表
  - 如果该页在主存中，根据页表中的页面编号和偏移值生成物理地址，并更新TLB

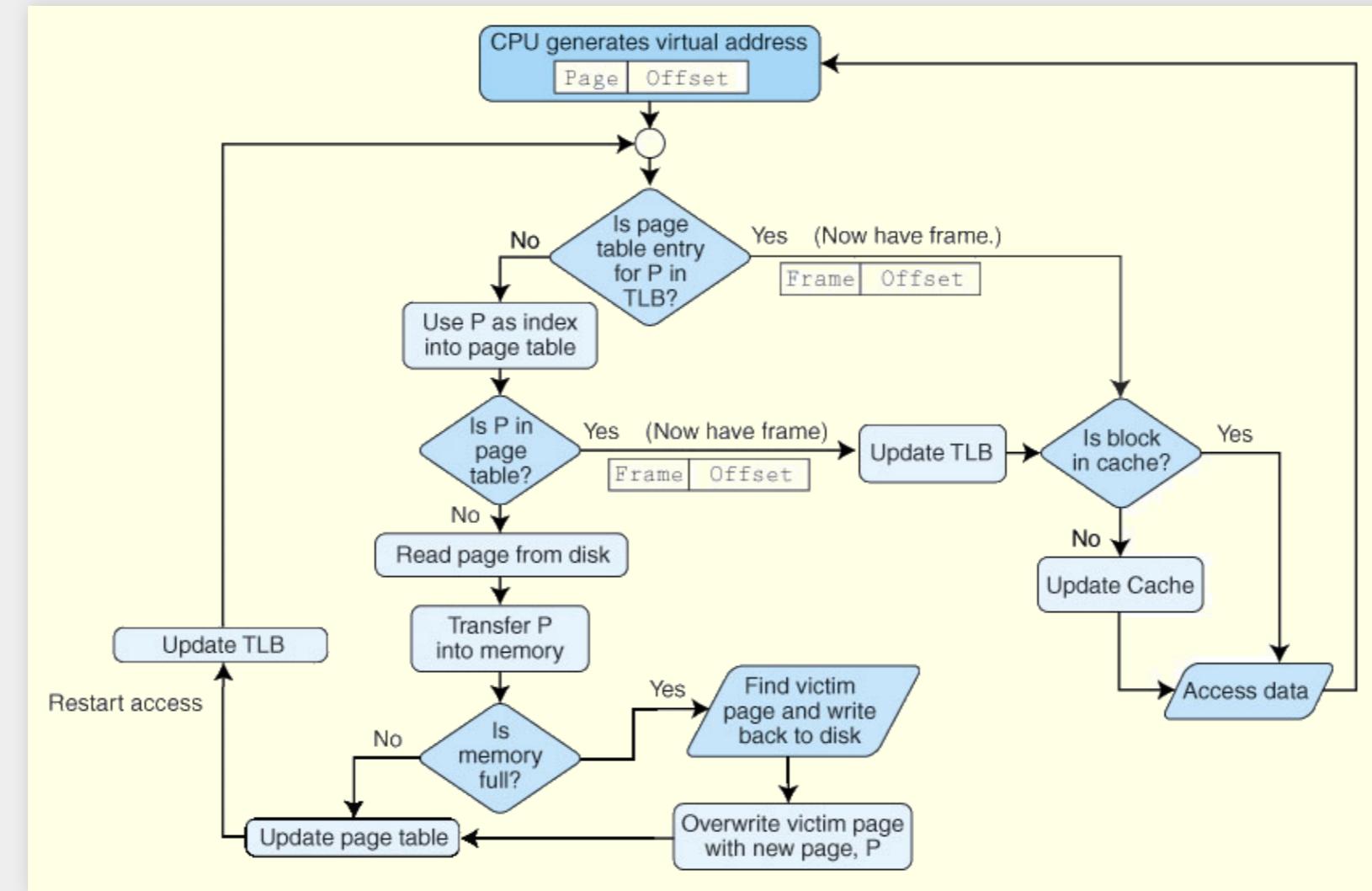


# 使用TLB的页面访问过程

- 从虚拟地址中提取出页号和偏移值
- 在TLB中查找虚拟页号对应的表项
- 如果找到，则用对应的物理页号和偏移值生成物理地址
- 如果未找到，需要访问内存中的页表
  - 如果该页在主存中，根据页表中的页面编号和偏移值生成物理地址，并更新TLB
  - 否则发生页面故障，触发中断，通常中断程序需要将页面加载到内存中，更新页表和TLB

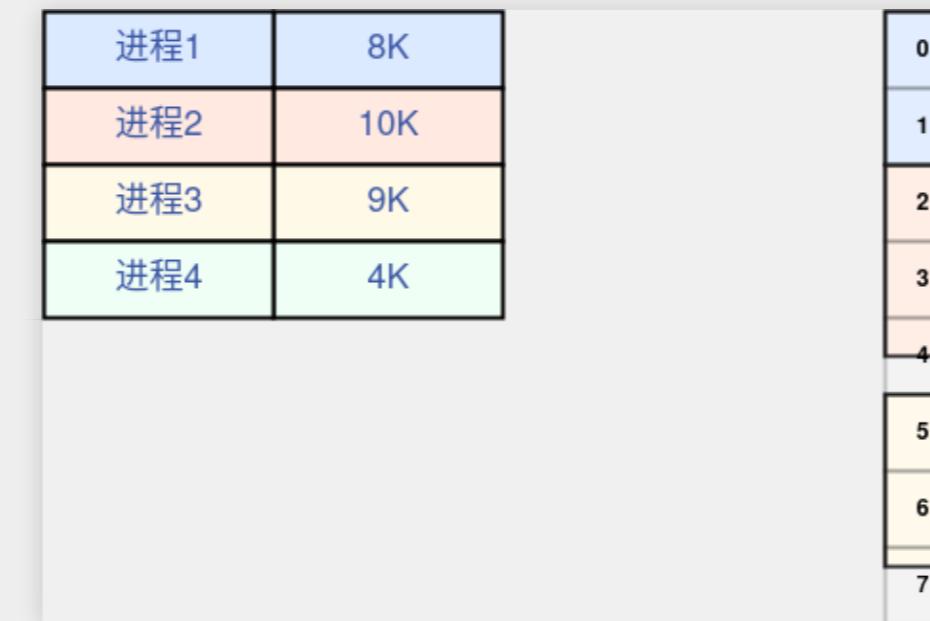


# 使用高速缓存、虚拟内存和TLB的数据访问过程



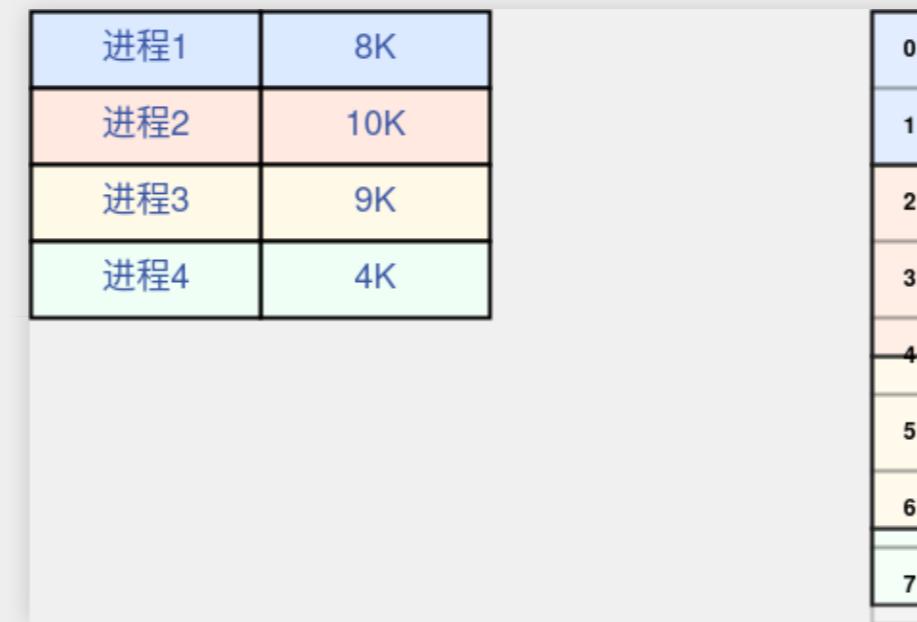
# 分页内存管理的内存碎片

- 当进程实际使用的数据没有与页大小对齐时
- 最后一页**存在一部分内存空间没有保存有意义的数据
- 载入内存后，这部分空间无法被其它进程使用，称为**内部碎片**



# 分段管理

- 分段管理允许一个进程拥有多个段(segment)
- 每个段的数据以一段连续的、长度可变的内存保存



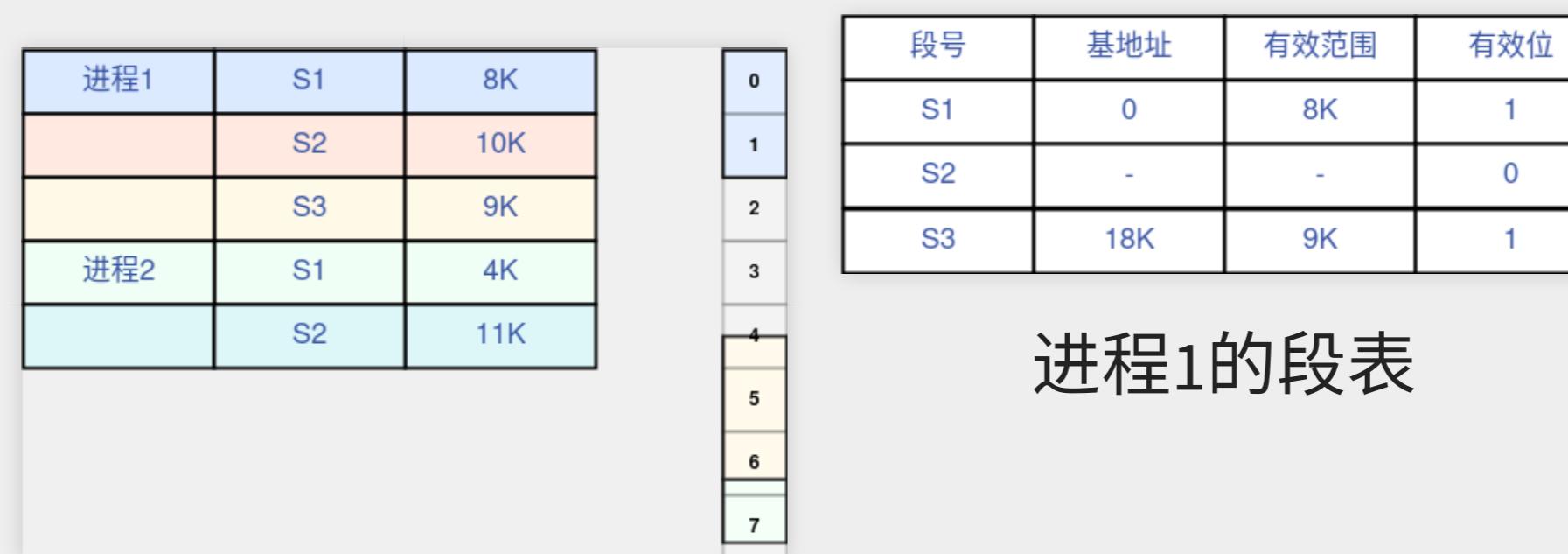
# 分段管理的地址转换

---

- 虚拟地址分为两个字段：段编号和段内偏移值
- 地址转换通过**段表**实现，段表中保存了每个段对应的物理内存基地址(base)和有效范围(limit)，以及状态位
- 对应的物理地址等于基地址加上偏移值

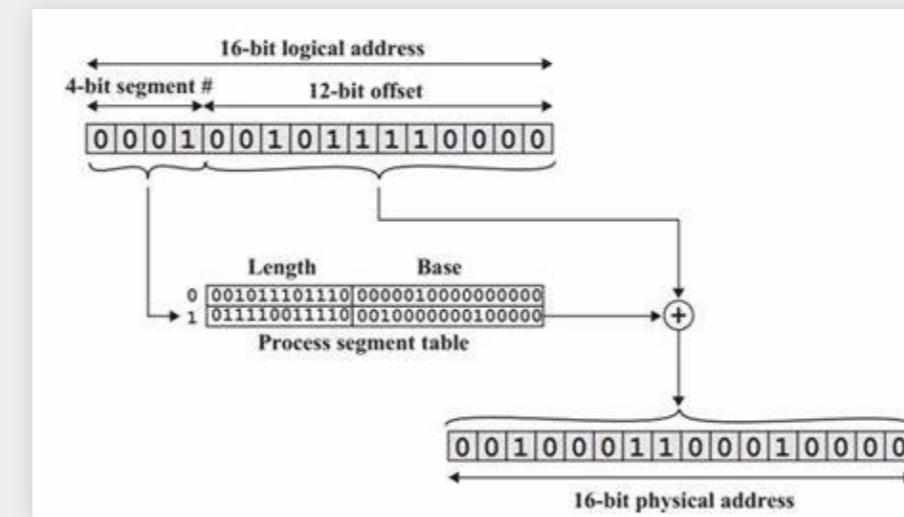
# 分段管理的地址转换

- 虚拟地址分为两个字段：段编号和段内偏移值
- 地址转换通过**段表**实现，段表中保存了每个段对应的物理内存基地址(base)和有效范围(limit)，以及状态位
- 对应的物理地址等于基地址加上偏移值



# 分段管理的地址转换

- 虚拟地址分为两个字段：段编号和段内偏移值
- 地址转换通过**段表**实现，段表中保存了每个段对应的物理内存基地址(base)和有效范围(limit)，以及状态位
- 对应的物理地址等于基地址加上偏移值



图片来源：<https://www.quora.com/What-is-the-difference-between-paging-and-segment-in-memory-management>

# 地址转换示例

假设一个按字节寻址的系统采用13位虚拟地址，最多支持8个段，其中一个进程的段表如下，请问下列虚拟地址对应的物理内存地址分别是多少：

- 0x1553
- 0x0852

段号	基地址	有效范围	有效位
0	0000 0000 0000	100	1
1	-	-	0
2	0100 0000 0000	64	1
3	-	-	0
4	-	-	0
5	0101 0000 0000	512	1
6	-	-	0
7	-	-	0

# 地址转换示例

假设一个按字节寻址的系统采用13位虚拟地址，最多支持8个段，其中一个进程的段表如下，请问下列虚拟地址对应的物理内存地址分别是多少：

- 0x1553
- 0x0852
- $0x1553 = (1\ 0101\ 0101\ 0011) = (101\mid 01\ 0101\ 0011)$ , 段号为(101)=5,偏移值为(01 0101 0011)=0x153=339
- 段表中5对应的有效位为1, 对应的基地址为(01 10 0000 0000), 有效范围为512 > 339
- 基地址加上偏移值后得到(01 11 0101 0011)=0x0753

段号	基地址	有效范围	有效位
0	0000 0000 0000	100	1
1	-	-	0
2	0100 0000 0000	64	1
3	-	-	0
4	-	-	0
5	0101 0000 0000	512	1
6	-	-	0
7	-	-	0

# 地址转换示例

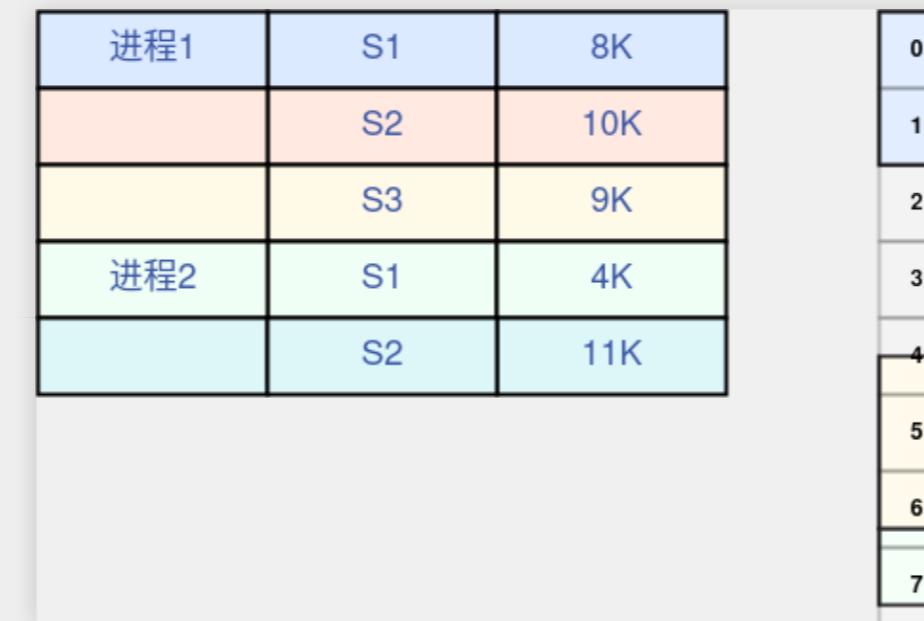
假设一个按字节寻址的系统采用13位虚拟地址，最多支持8个段，其中一个进程的段表如下，请问下列虚拟地址对应的物理内存地址分别是多少：

- 0x1553
- 0x0852
- $0x0852 = (0\ 1000\ 0101\ 0010) = (010\mid 00\ 0101\ 0011)$ , 段号为(010)=2,偏移值为(00 0101 0010)=0x052=82
- 段表中2对应的有效位为1, 对应的基地址为(01 00 0000 0000), 有效范围为64 < 82, 虚拟地址越界

段号	基地址	有效范围	有效位
0	0000 0000 0000	100	1
1	-	-	0
2	0100 0000 0000	64	1
3	-	-	0
4	-	-	0
5	0101 0000 0000	512	1
6	-	-	0
7	-	-	0

# 分段内存管理的碎片

- 由于一个段的数据必须是连续的，当较小的段换出之后，较大的段可能无法找到合适的位置
- 这类碎片称为**外部碎片**



进程2的S2无法直接载入内存

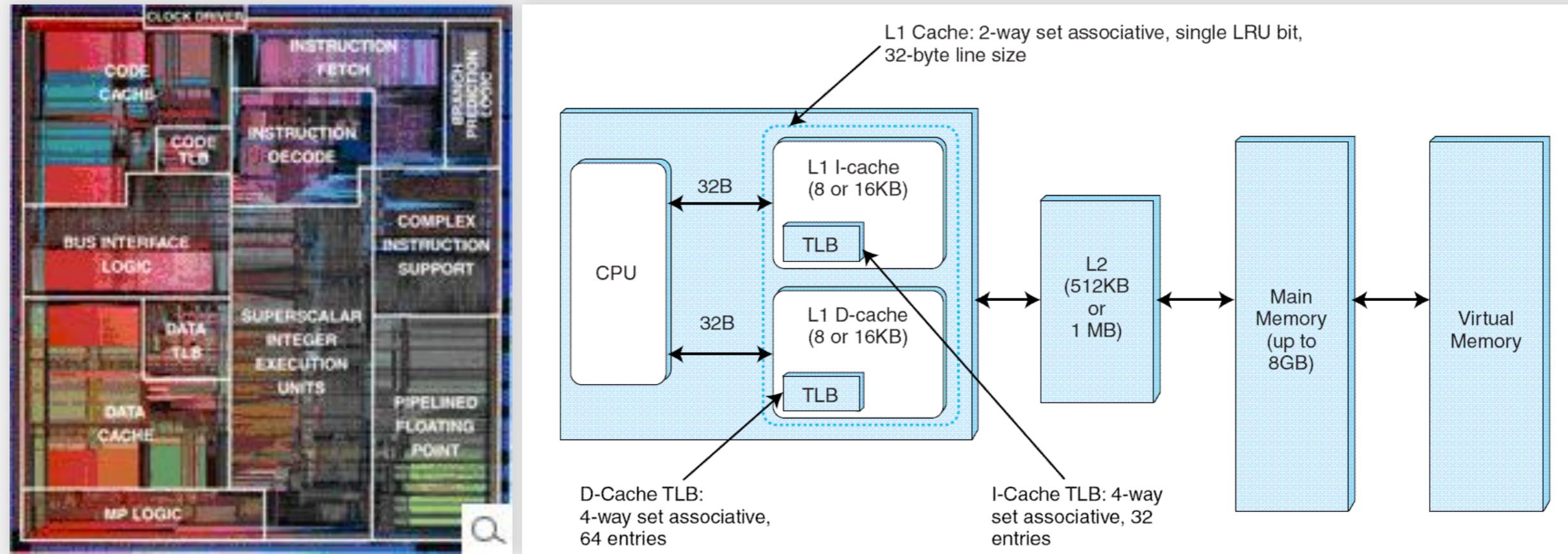
# 分页管理和分段管理的组合

---

	分页管理	分段管理
数据块大小	固定大小，较小	可变大小，一般较大
碎片	可消除外部碎片	可消除内部碎片
特点	易于系统管理调度	易于编程管理

在实际中，采用分段和分页结合的内存管理方法，称为**段页式内存管理**(操作系统课程中会进一步学习)

# 存储器管理实例



图片来源: <http://mistic.web.cs.unibo.it/files/images/cronologia/pent.jpg>



第七讲结束

# 本期内容总结

---

- 高速缓存
  - 缓存的写策略
  - 指令和数据缓存、多级缓存
- 虚拟内存
  - 分页内存管理：页表，地址转换，TLB
  - 内存碎片和分段内存管理
- 存储器实例：Pentium处理器



# Q & A