

# 课程内容

- 数据库系统基本概念（数据模型，体系结构）

- 关系数据库

- 关系数据库标准语言SQL

- 数据库保护

- 关系数据理论

- 数据库设计

基础理论

设计理论

- 存储管理与存取方法

- 查询处理和查询优化

- 事务处理技术

实现技术

- 数据库技术新发展

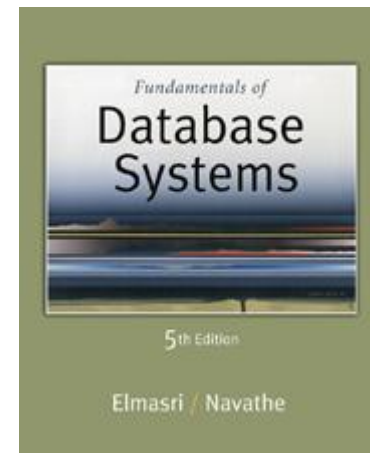
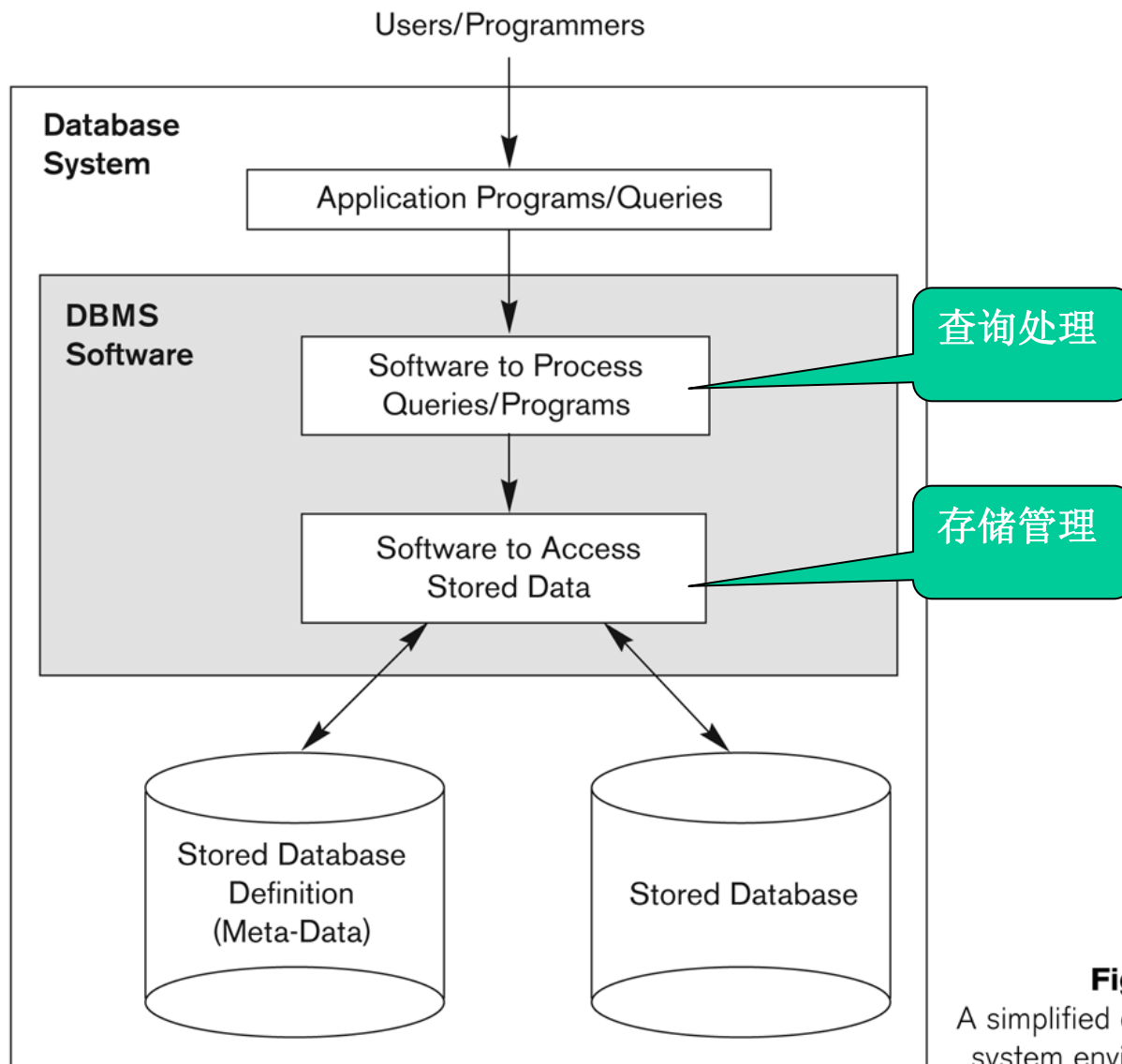
新技术

# 第三部分 DBMS实现技术

---

- 概述
- 存储管理与存取方法
- 查询处理和查询优化
- 事务处理技术

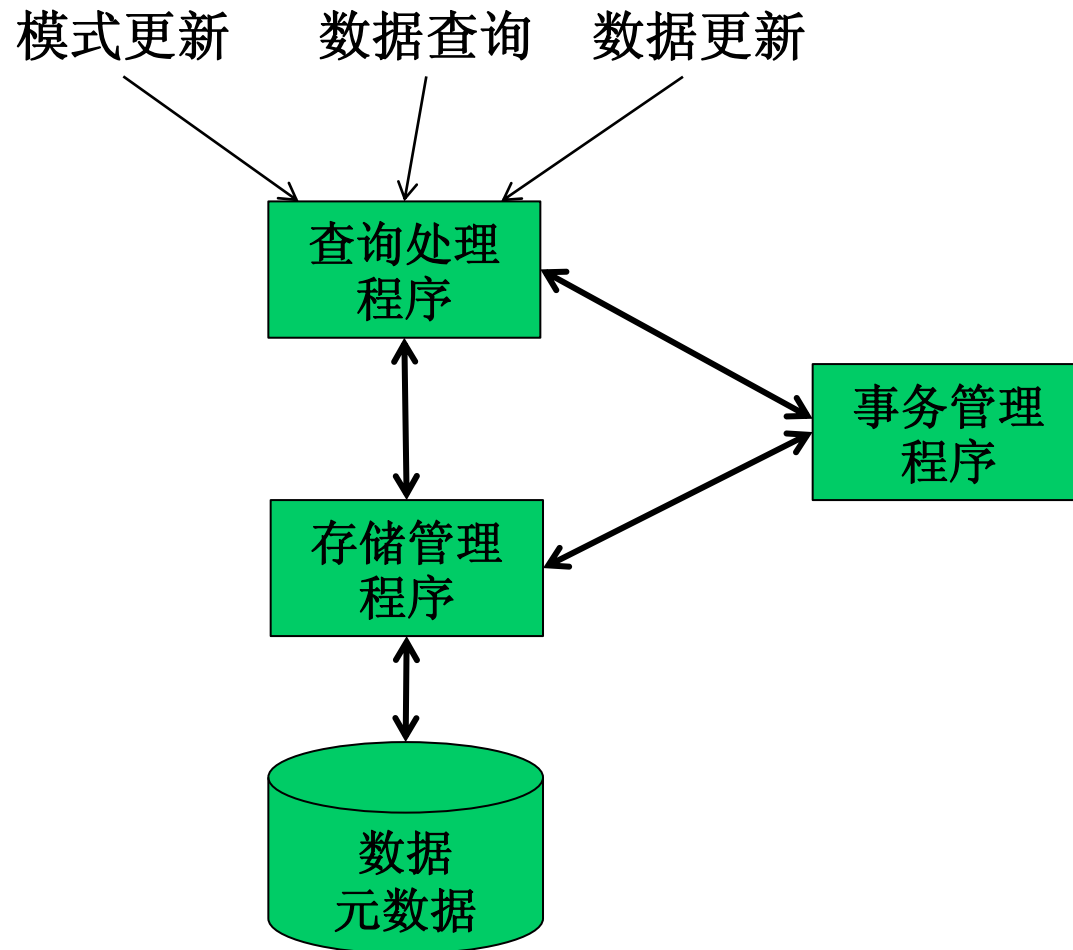
# DBMS体系结构



**Figure 1.1**  
A simplified database system environment.

# DBMS 主要组成部分

---



# DBMS查询 执行过程

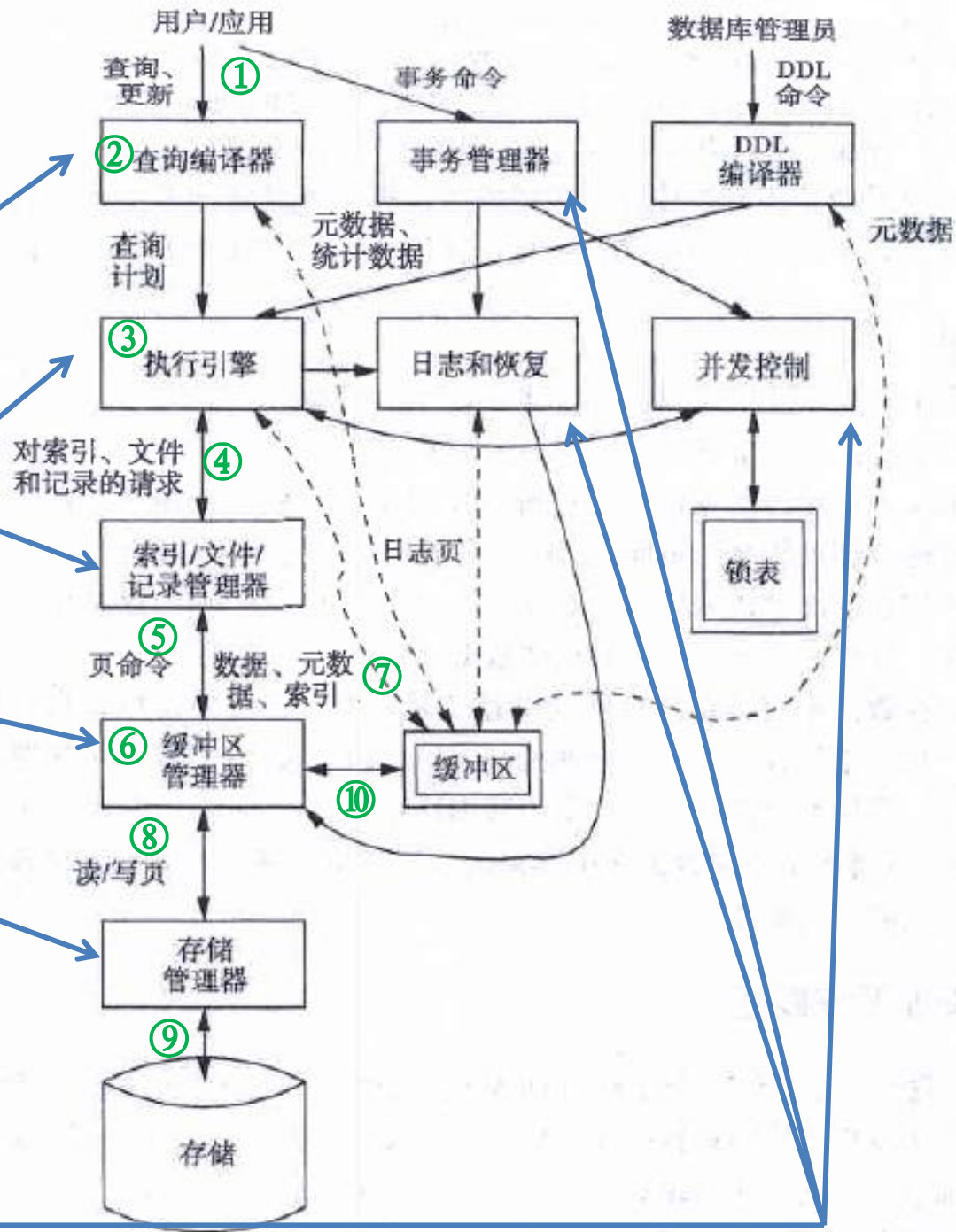
查询处理与优化

查询执行/数据存取

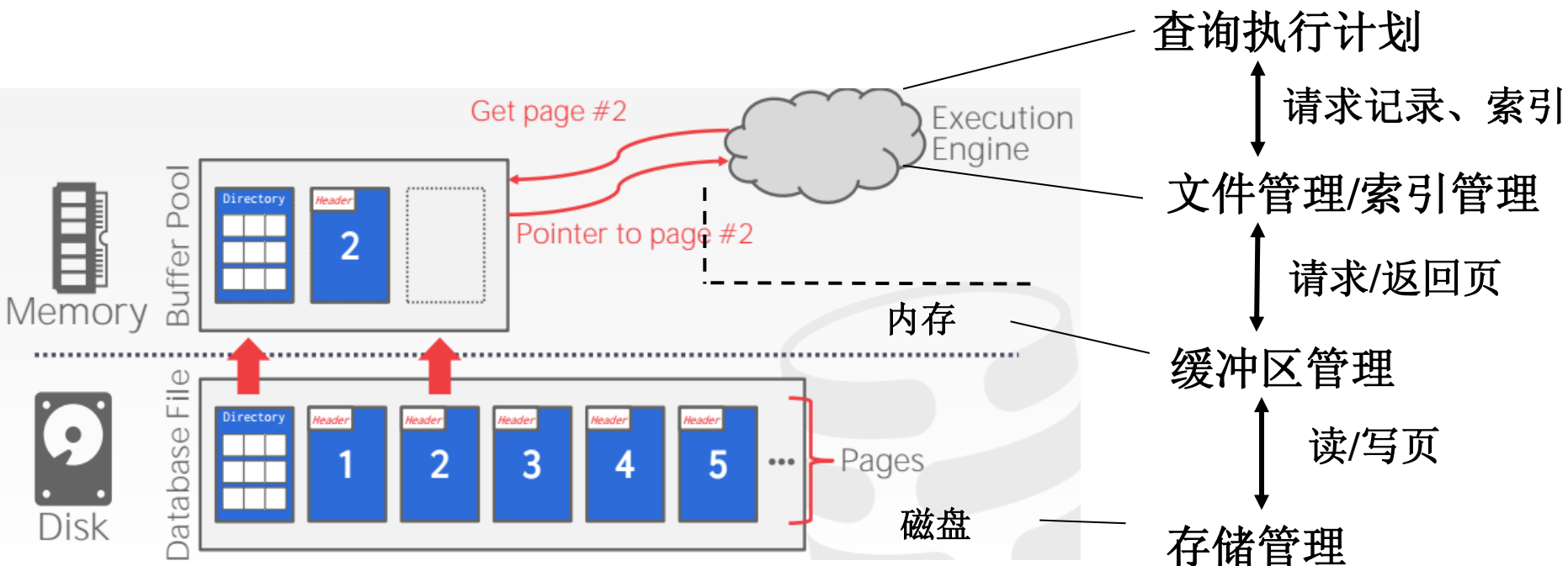
缓冲区管理

存储管理

事务处理

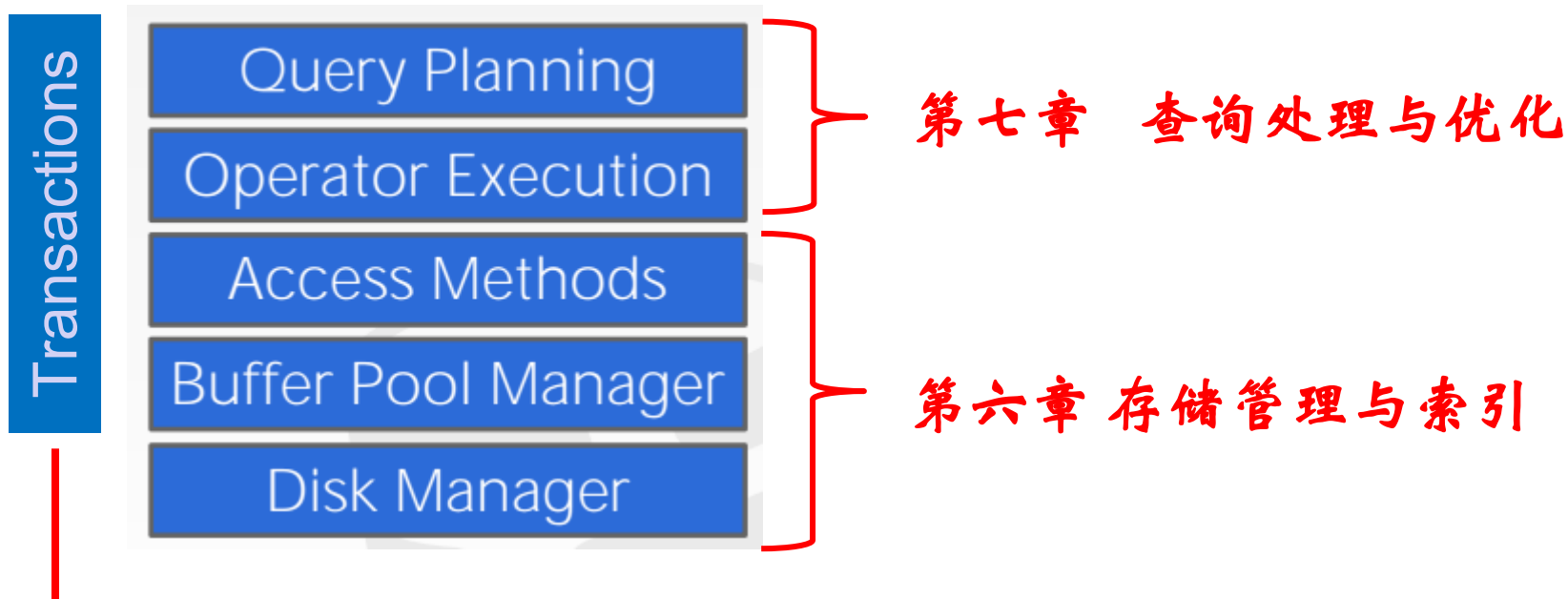


# DBMS数据存储与查询实现基本框架



# DBMS实现中的关键技术

---



第八章 事务处理

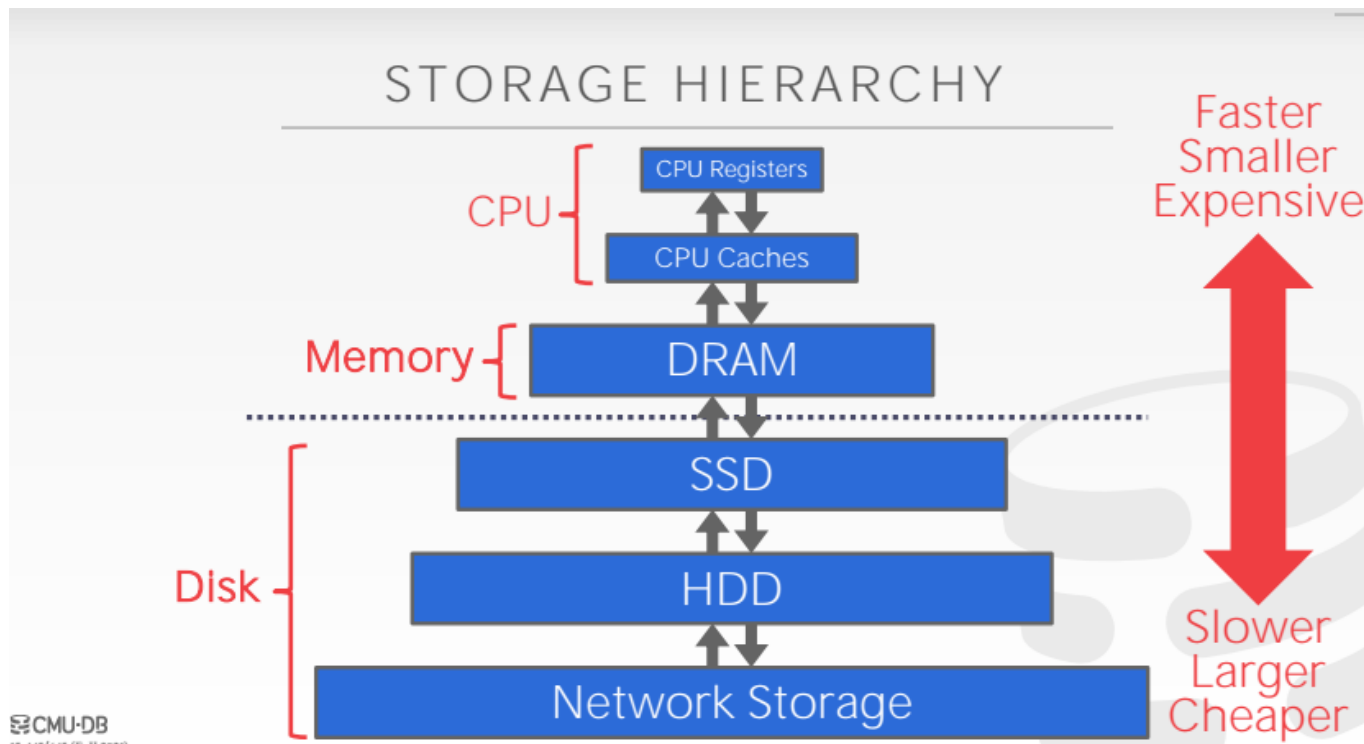
# 第六章 存储管理和索引

---

- 物理存储系统
- 数据存储结构/物理结构
- 缓冲区管理
- 索引



# 存储体系结构



- 数据只有放入内存才能被处理
- DBMS设定数据库的基本存储是在磁盘上，DBMS的组件管理内存与外存数据的交换

# 不同存储访问时间

## ACCESS TIMES

0.5 ns L1 Cache Ref

7 ns L2 Cache Ref

100 ns DRAM

150,000 ns SSD

10,000,000 ns HDD

~30,000,000 ns Network Storage

1,000,000,000 ns Tape Archives

## ACCESS TIMES

0.5 ns L1 Cache Ref

7 ns L2 Cache Ref

100 ns DRAM

150,000 ns SSD

10,000,000 ns HDD

~30,000,000 ns Network Storage

1,000,000,000 ns Tape Archives

← 0.5 sec

← 7 sec

← 100 sec

← 1.7 days

← 16.5 weeks

← 11.4 months

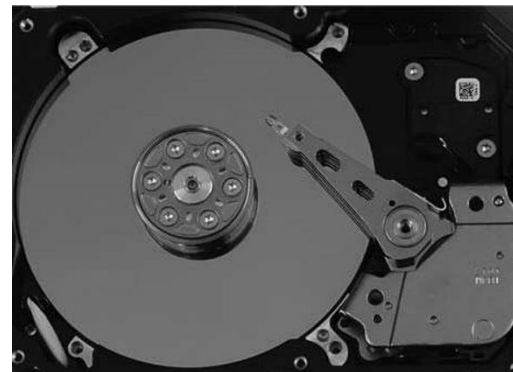
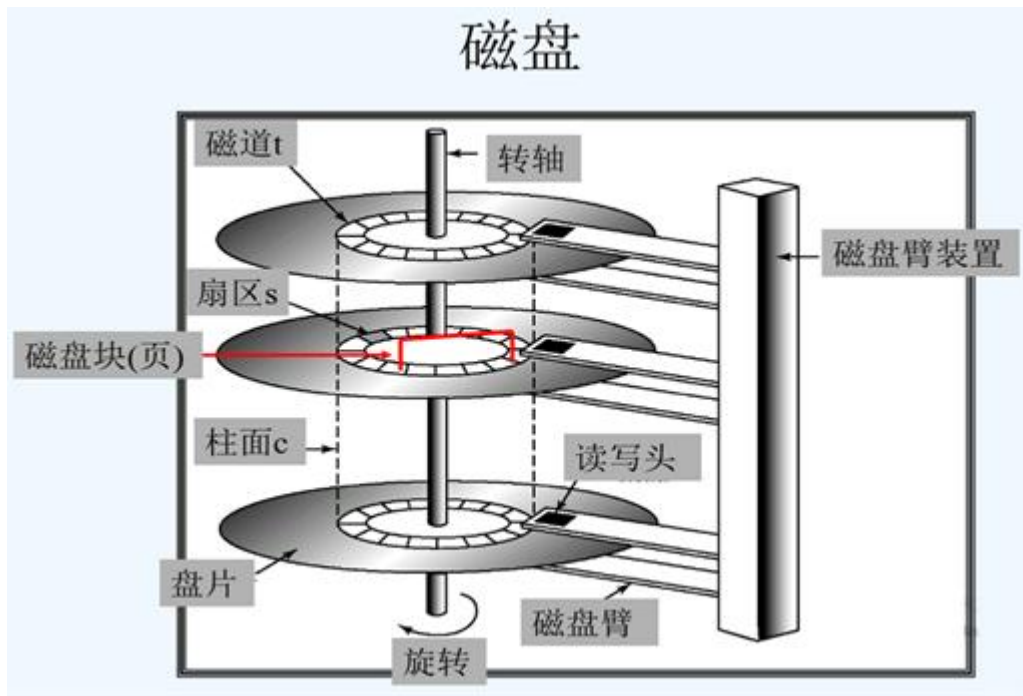
← 31.7 years

## • DBMS存储管理的目标

- 最小化磁盘和主存间传输存储块的数量，即最小化磁盘存取次数；实现手段是在主存中保持尽量多的块，使得上层要访问一个块时，它的主存中的概率最大。

# 磁盘装置

- 磁盘结构

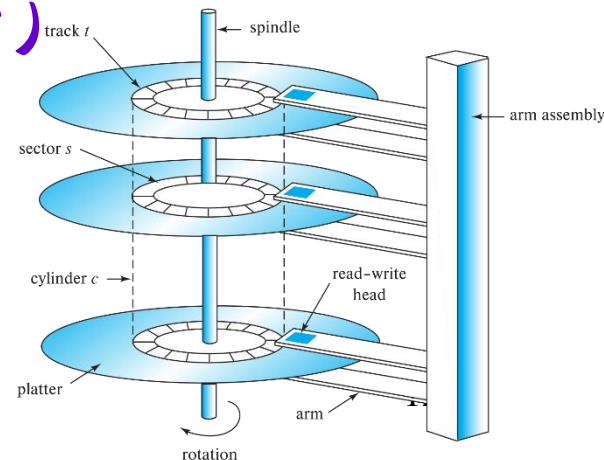


- 磁盘块

- 由若干个连续的扇区构成，是存储分配和检索的逻辑单元，大小一般在4K-16K之间，数据以块为单位在磁盘和主存之间传输。页面 (page) 通常指块

# 磁盘访问时间

- 磁盘访问时间：从发出读写请求到数据开始传输
  - 寻道时间
    - 磁盘臂定位时间，2-20ms
  - 旋转时间
    - 等待被访问的扇区出现在读写头下方的时间，4-11ms
  - 传输时间
    - 从磁盘读取数据或向磁盘存储数据的时间，每秒50M-200M
    - 磁盘块一般在4KB-16KB，最小传输时间在0.2ms
- 示例——读一个磁盘块（4个扇区16384字节）
  - 时间=寻道时间+旋转时间+传输时间
  - 平均时间 在10ms左右



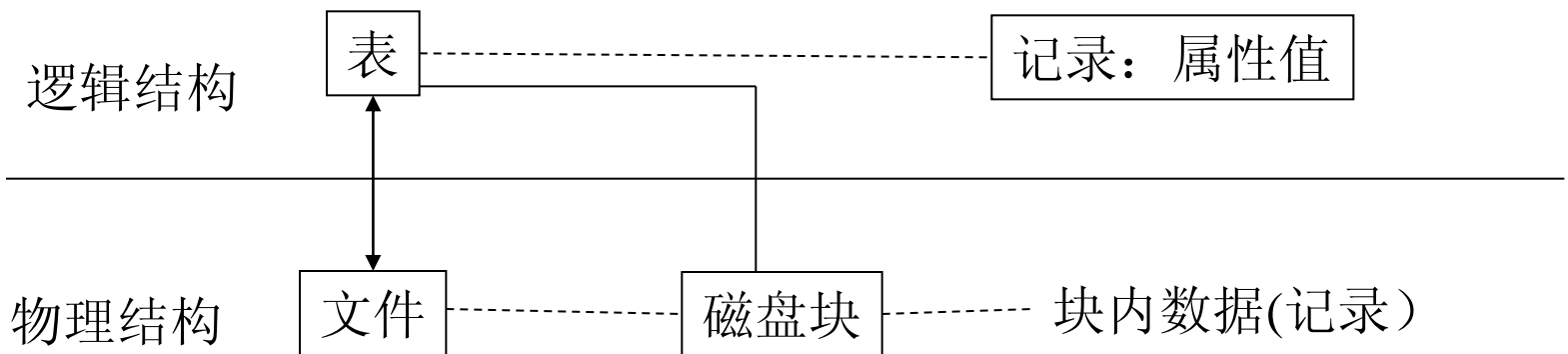
# 存储管理系统

---

- 数据库-文件-块/页
  - 数据库：由若干文件组成，这些文件采用专有的格式。操作系统不能获取这些文件内容的任何信息。
  - 文件：由若干个定长的存储单元/存储块/页构成。
  - 页：存储分配和数据传输的单位。
- DBMS中的存储管理器负责维护这些数据库文件，将文件组织为块/页的集合，并且
  - 跟踪页的数据读取/写入
  - 跟踪可用的空间

# 数据库的物理结构

- 数据库的表被映射为底层存储中的文件
- 一个文件在逻辑上被组织为记录的序列，记录被映射到磁盘块上
- 文件在存储中由若干磁盘块构成，块是存储分配和数据传输的单位
- 一个块可以包含几个记录，每条记录被完全包含在单个块中。



块号—盘面: 柱面: 扇区

# 数据库的物理结构

---

- 表所占磁盘块的分配方法
  - 连续分配--数据块被分配到连续的磁盘块上
  - 链接分配-数据块中包含指向下个数据块的指针
  - 按簇分配——簇是连续的几个磁盘块，簇之间指针连接
  - 索引分配——索引块中存放指向数据块的指针

# 数据库页/磁盘块

---

- 页是固定大小的数据块
  - 可以包含元组/记录，元数据，索引，log记录等
  - 每个页有唯一标识符（ID），DBMS将页ID映射为页的物理位置



# 数据库页/磁盘块结构

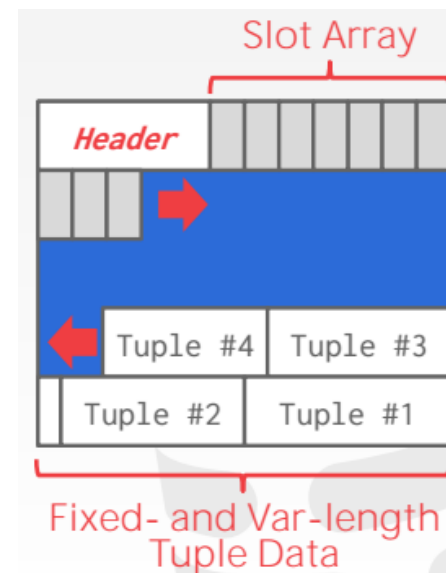
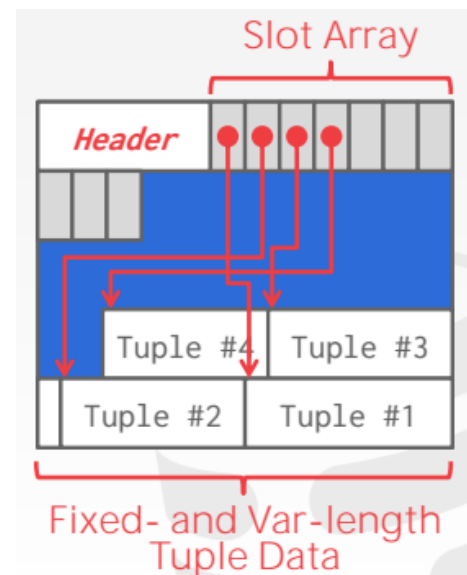
---

- 每个页由头部header和数据构成。
- Header 包含了页中数据的元数据，例如：
  - 页大小
  - Checksum
  - DBMS 版本



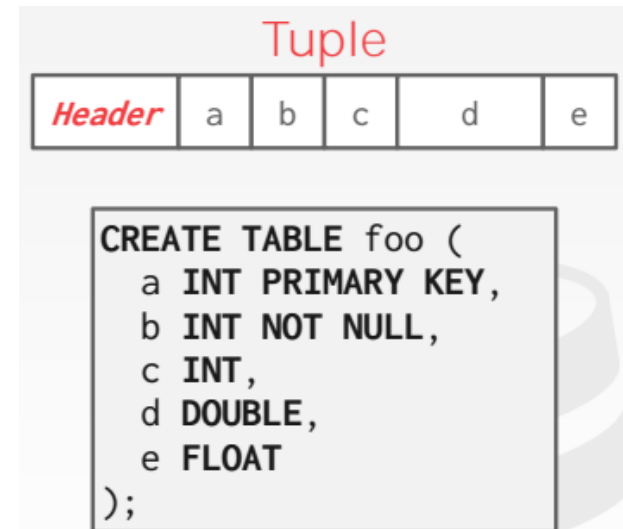
# 数据库页/磁盘块结构

- 最常用的结构是分槽(slot)页结构
  - Header记录了已使用的槽数，以及最后一个被用槽的起始位置偏移量，以及一个槽数组
  - 槽数组保存了每个元组的起始位置偏移量；
  - 增加记录时，槽数组从开始到尾部的方向增长，而记录数据则从数据区的尾部到开始的方向增长。当槽数组与元组数据连接到一起时，认为页满
  - 便于存储变长记录



# 记录的结构

- 是字节序列，DBMS负责将该序列解释为属性类型和值
  - 记录头部：包含元组的元数据，例如加锁信息等。
  - 记录数据：属性的实际数据。属性一般按表定义中的顺序存储。多数DBMS不允许一个记录大小超过一个页
  - 唯一标识符ID
    - 每个记录被分配了一个ID；
    - 最常见的形式：页ID+（offset或槽）
    - 应用程序不能依赖该ID进行唯一性标识



# 文件的记录组织

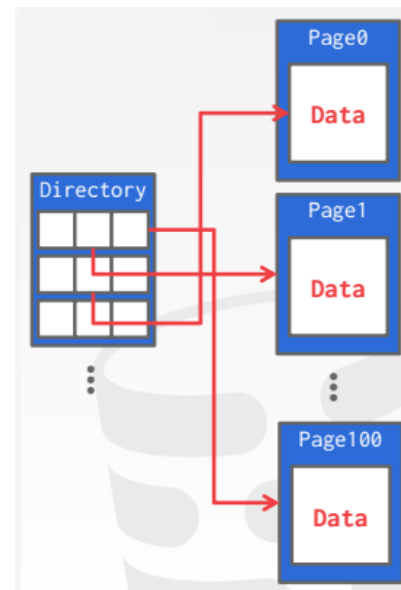
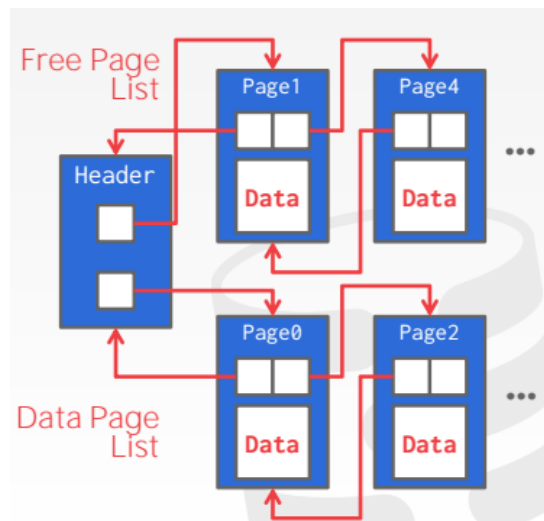
---

- 文件中记录的组织方式
  - 堆(Heap): 记录可以存放在文件空间中的任何位置
  - 顺序(Sequential): 基于每个记录的搜索码值顺序排列
  - 索引(Indexing): 按某种顺序有序存储
  - 散列(Hashing): 在搜索码上的hash函数, 计算出记录在文件中存放的块
  - 聚集(clustering): 将有联系的记录存储在同一个块上, 以最小化I/O次数

# 文件中的记录组织-堆

- 堆文件组织

- 链表方式：在文件开始维护一个header页，该页存储了空白页链表头指针和数据页链表头指针，每个页记录了当前包含的空槽数
- 页目录方式：DBMS维护特殊页保存文件中的数据页的位置，并记录每个页中空槽数

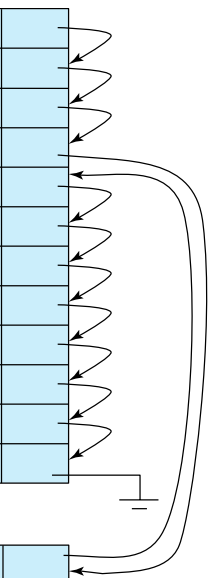


# 文件的记录组织-顺序

- 顺序文件组织

- 文件中的记录按搜索码排序。搜索码可以是任意属性或属性集合
- 通过指针把记录链接起来，每个记录的指针指向按搜索码排列的下一条记录
- 可以高效按某个搜索码处理记录

ID	name	dept_name	salary	
10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	
32222	Verdi	Music	48000	



插入记录

# 文件的记录组织-聚集

- 具有相同或相似属性值的记录存储于连续的磁盘块中
- **聚集码**是一种属性，它定义了哪些记录被存储在一起
- 多表聚集：将多个关系存储于一个文件中，在每个块中存储两个或更多关系的相关记录.可以加快特定的连接查询，但会使单个表的访问变慢

*department*

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Physics	Watson	70000

Select dept\_name, building, budget, ID, name salary  
From department, instructor  
Where department.dept\_name = instructor.dept\_name;

*instructor*

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

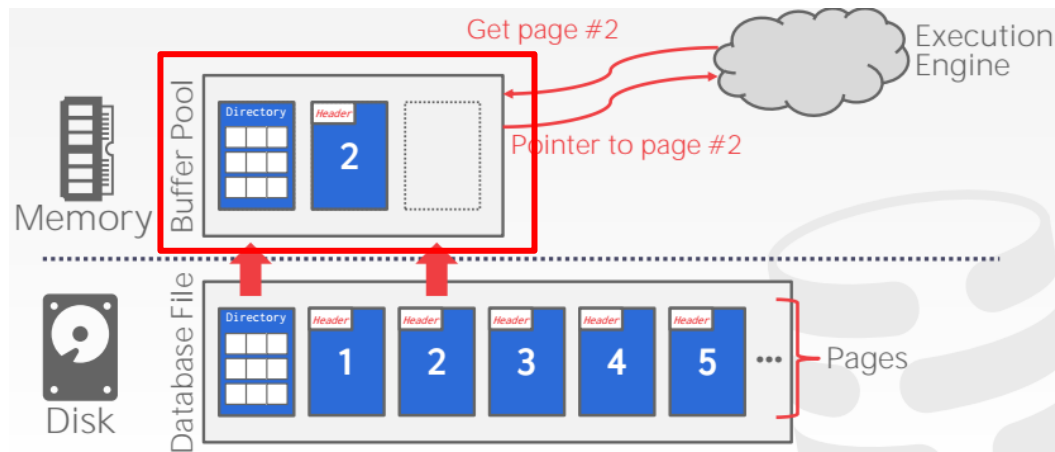
**聚集文件结构**

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000



# 缓存管理系统

- 块/页是存储分配和数据交换的单位
- 管理目标：最小化磁盘和主存间传输存储块的数量，即最小化磁盘存取次数；实现手段是在主存中保持尽量多的块
- 缓冲区：是主存中可以存储磁盘块副本的区域
- 缓存管理器：负责缓存空间分配，内外存交换





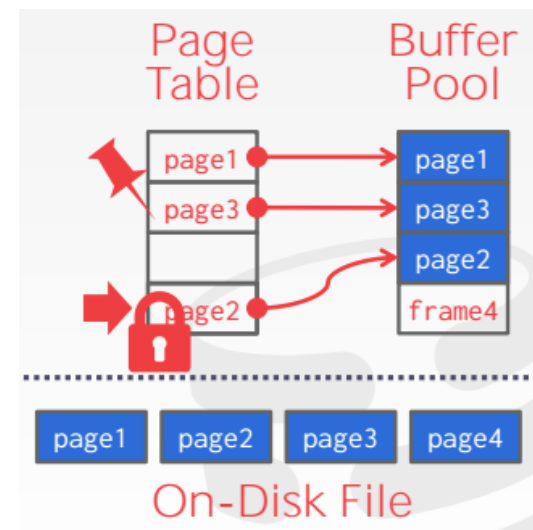
# 缓冲区管理系统

---

- 执行引擎需要操作某磁盘块时，通过资源管理器调用缓冲区管理
  - 如果该块在缓冲区中，则缓冲区管理器返回该块在内存的地址
  - 如果不在缓冲区，则缓冲区管理器为该块在缓冲区中分配空间（这可能替换某个块，如果被替换的块修改过，需要将其写回磁盘），将块从磁盘读进缓冲区，并将内存地址返回给调用者

# 缓冲区组织与管理

- 缓冲区被组织为一个固定大小**页面数组**，每个元素称为帧，存放磁盘上的一个页/块
- 缓冲区元数据—页表（page table），跟踪当前内存中的所有页，并保存了每个页的元数据，包括
  - Dirty Flag: 由修改页的线程设置，通知存储管理器该页必须写回磁盘
  - Pin/Reference 计数器: 在一页被进程读写操作前要钉住（pin），防止该页被移出，操作结束后解除钉（计数器减1），只有计数器=0时，才能被移出或写回磁盘



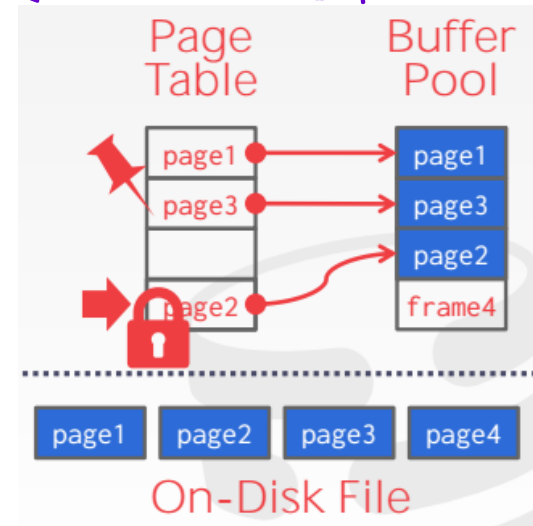
# 缓冲区组织与管理

- 缓冲区中的共享锁与排它锁

- 缓存区管理器提供封锁系统，允许数据库进程以共享或排他模式封锁页，在完成操作后释放封锁
- 实现并发控制，读操作加共享锁，更新操作加排他锁
- 加锁规则：一次只能由一个进程获得排它锁，共享锁与排它锁不能同时加，多个进程可以同时持有共享锁

- 缓冲区替换策略

- 最近最少使用（Least Recently Used）策略及其改进算法



# 索引的基本概念

---

- 索引文件构成

- 索引记录/索引项，是索引文件的记录，包括两个域：

- 索引域（搜索码）：存储数据文件中一个或一组域（属性）
    - 指针：指向索引域值为K的记录所在磁盘块的地址。

索引域/搜索码	指针
---------	----

- 索引将表中的部分属性进行组织或排序，使得利用这些属性能够快速有效进行表的访问
- DBMS负责在执行查询时使用最恰当的索引

# 索引的分类

---

- 分类——两种基本类型
  - 排序索引：索引项是排序的
  - 哈希索引：索引项使用索引域上的hash函数确定位置
- 聚集索引与非聚集索引
  - 聚集索引：索引项值排列顺序与记录在文件中的排列顺序一致，也称为主索引
  - 非聚集索引：索引项指定的次序与文件中记录的排列顺序不同，也称为辅助索引
- 稠密索引与稀疏索引
  - 稠密索引
  - 稀疏索引

# 稠密索引

- 稠密索引：对于文件中的每个搜索码值都有一个索引项，例如，*instructor* 表的索引按ID建立稠密索引：

ID		ID	name	dept_name	salary	
10101	→	10101	Srinivasan	Comp. Sci.	65000	↙
12121	→	12121	Wu	Finance	90000	↙
15151	→	15151	Mozart	Music	40000	↙
22222	→	22222	Einstein	Physics	95000	↙
32343	→	32343	El Said	History	60000	↙
33456	→	33456	Gold	Physics	87000	↙
45565	→	45565	Katz	Comp. Sci.	75000	↙
58583	→	58583	Califieri	History	62000	↙
76543	→	76543	Singh	Finance	80000	↙
76766	→	76766	Crick	Biology	72000	↙
83821	→	83821	Brandt	Comp. Sci.	92000	↙
98345	→	98345	Kim	Elec. Eng.	80000	↙

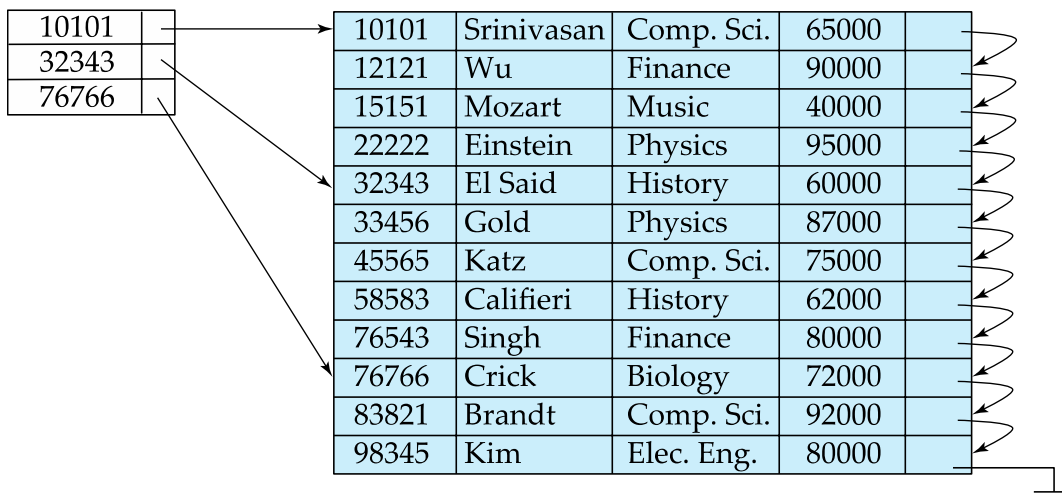
按ID建立稠密索引

Biology	→	76766	Crick	Biology	72000	↙
Comp. Sci.	→	10101	Srinivasan	Comp. Sci.	65000	↙
Elec. Eng.	→	45565	Katz	Comp. Sci.	75000	↙
Finance	→	83821	Brandt	Comp. Sci.	92000	↙
History	→	98345	Kim	Elec. Eng.	80000	↙
Music	→	12121	Wu	Finance	90000	↙
Physics	→	76543	Singh	Finance	80000	↙
	→	32343	El Said	History	60000	↙
	→	58583	Califieri	History	62000	↙
	→	15151	Mozart	Music	40000	↙
	→	22222	Einstein	Physics	95000	↙
	→	33465	Gold	Physics	87000	↙

按dept\_name  
建立稠密索引

# 稀疏索引

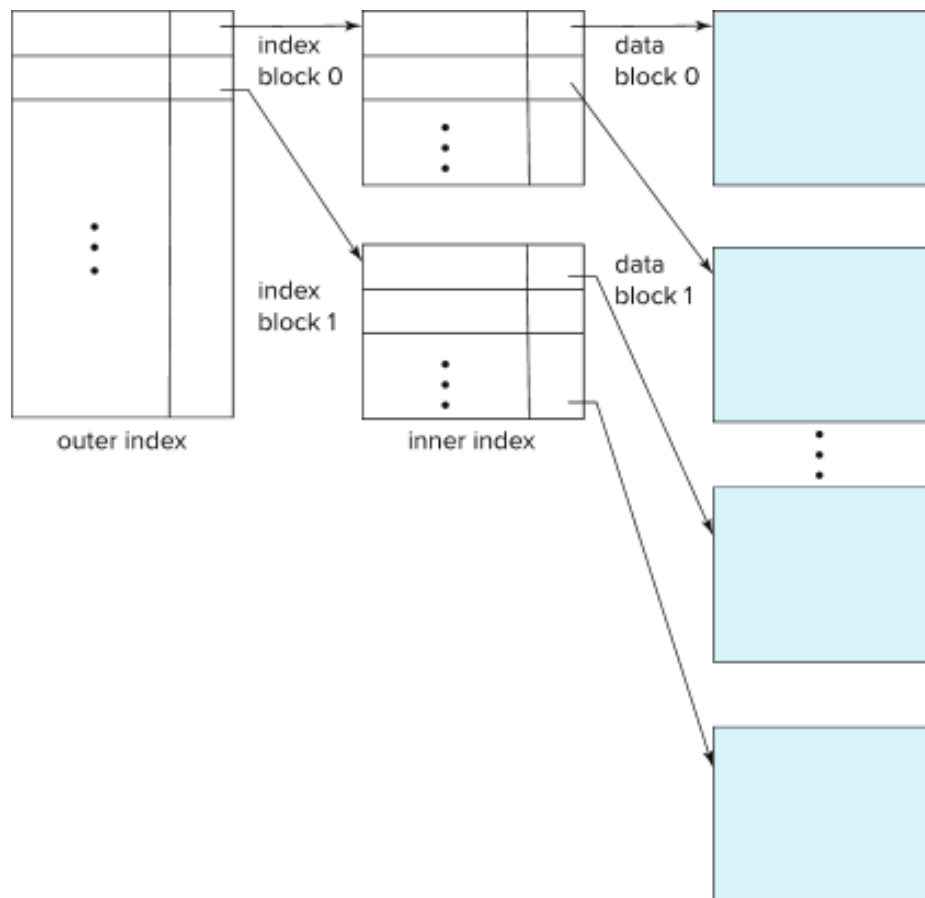
- 稀疏索引：只有部分索引域值有索引记录。当文件记录以索引域排序时，可以采用。



- 利用稀疏索引查找索引域为k的记录：在索引中定位小于k的最大索引域值，从该索引域指针指向的记录开始顺序查找
- 相比稠密索引，占空间小并且维护代价低，但定位记录慢。非聚集索引都是稠密索引

# 多级索引

- 索引规模大，无法全部放入内存
- 对索引文件建立稀疏索引
  - 外层索引：基本索引的稀疏索引
  - 内层索引：基本索引文件





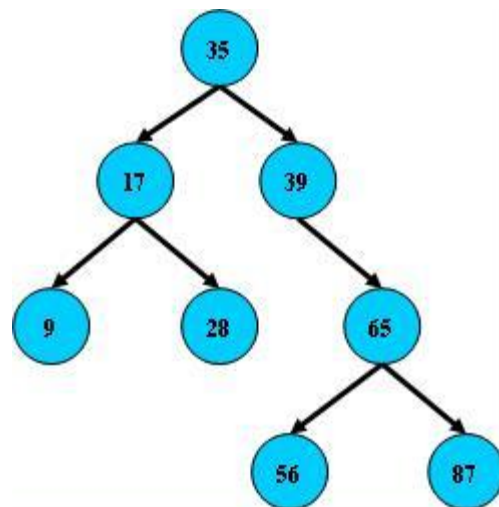
# 多级索引

- 二叉树索引

- 每个节点只有一个关键字，有两个指针，分别指向关键字值小于或大于当前节点值的节点；

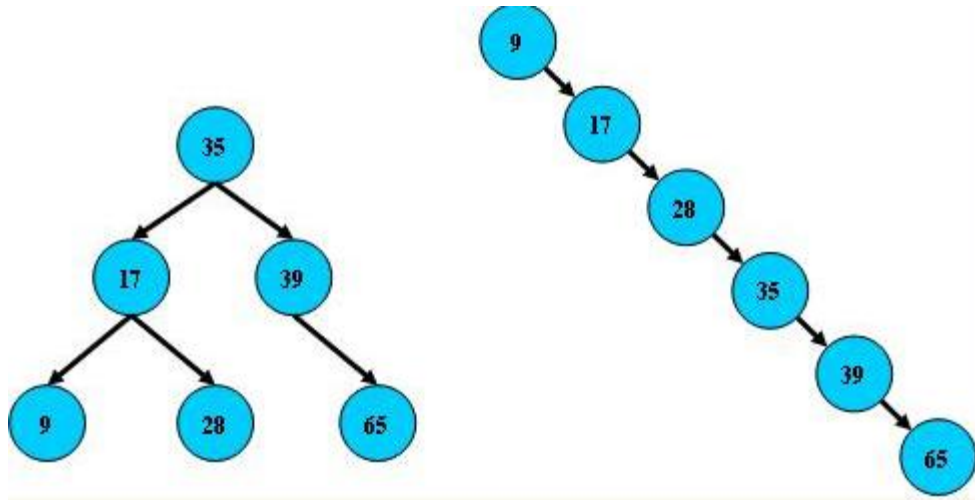
- 多枝树索引

- 每个节点有 $D$  ( $D \geq 2$ )个关键字值，则有 $D+1$ 个指针，每两个指针之间对应相邻关键字之间取值的区间。

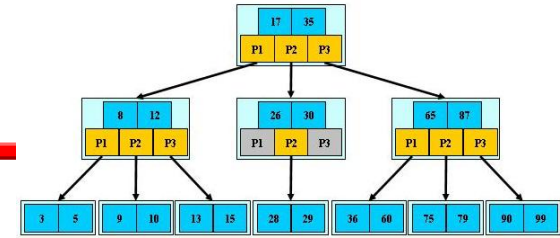


# 多级索引

- 二叉树索引的问题

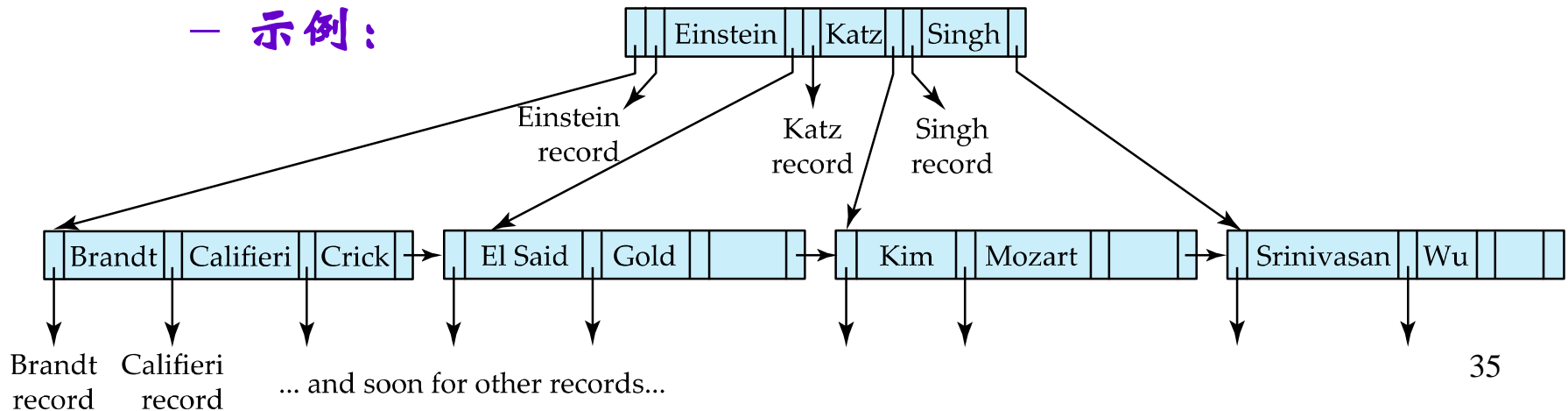


# B树



## • B树（平衡树）索引

- 是附加限制条件的索引树。限制了每个节点放置关键字与指针的最小和最大个数：根节点有 $[2, n]$ 个子节点，中间节点有 $[n/2, n]$ 个子节点，叶节点有 $[n/2, n-1]$ 个记录指针， $n$ 值对特定树是固定的
- 从树根到叶节点每条路径的长度都相同，因此所有的叶节点都在同一层上。
- B树的关键字是散布在各层上。
- 示例：



# B<sup>+</sup>树

- 是B树的改进。把树中所有关键字都按递增次序从左到右安排在叶节点上，并且链接起来。B<sup>+</sup>树能同时进行随机查找和顺序查找。
- B<sup>+</sup>树节点结构

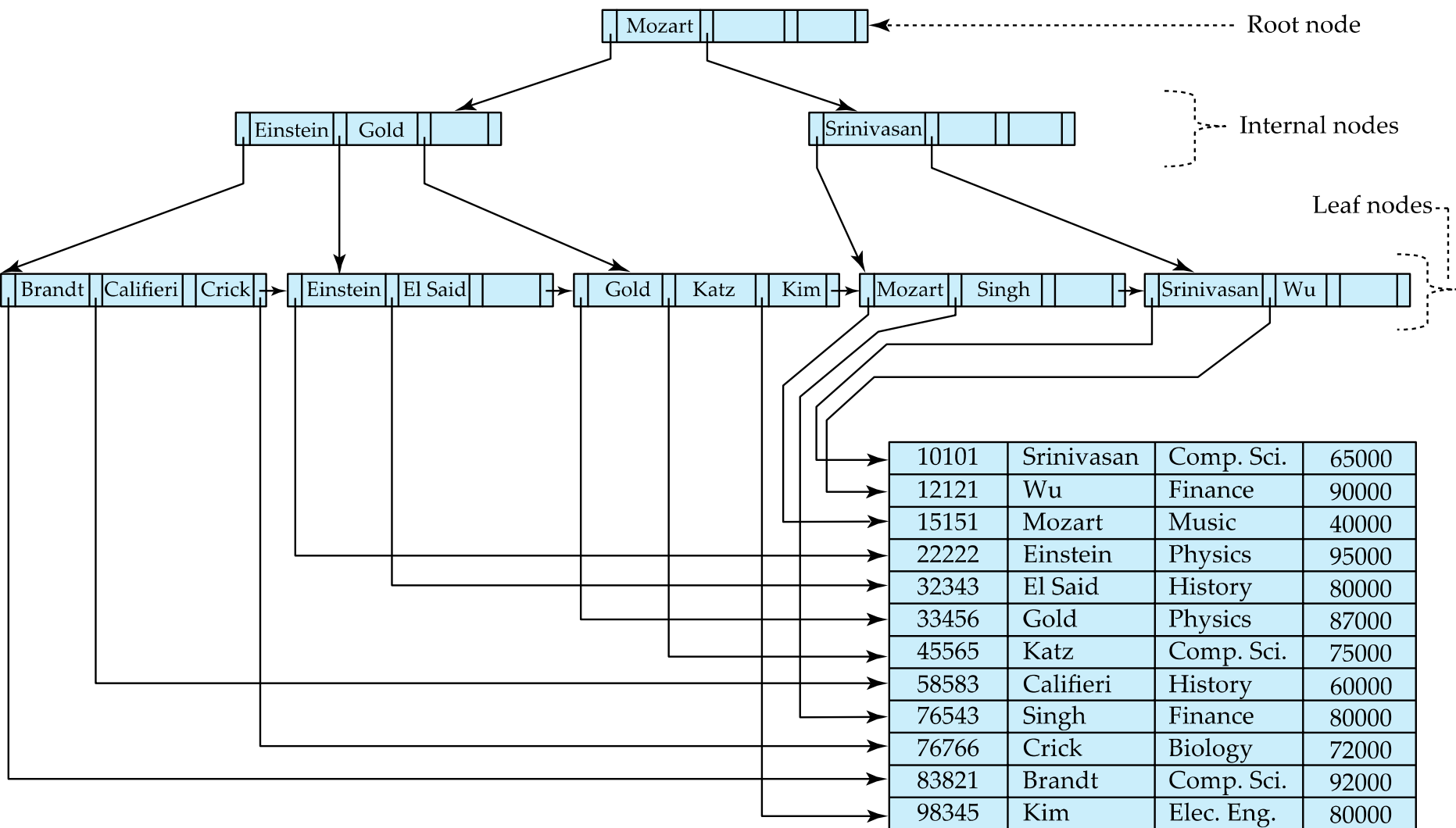
每个节点最多包含n-1个搜索码/索引码值  $K_1, K_2, \dots, K_{n-1}$ ，以及n个指针  $P_1, P_2, \dots, P_n$ 。

$P_1$	$K_1$	$P_2$	...	$P_{n-1}$	$K_{n-1}$	$P_n$
-------	-------	-------	-----	-----------	-----------	-------

- $K_i$  是索引码的值
- $P_i$  对于非叶子节点是指向子节点的指针；对于叶子节点  $P_1, P_2, \dots, P_{n-1}$  是指向记录或记录桶的指针， $P_n$  指向下个叶子节点
- 节点中各索引码的值满足：

$$K_1 < K_2 < K_3 < \dots < K_{n-1}$$

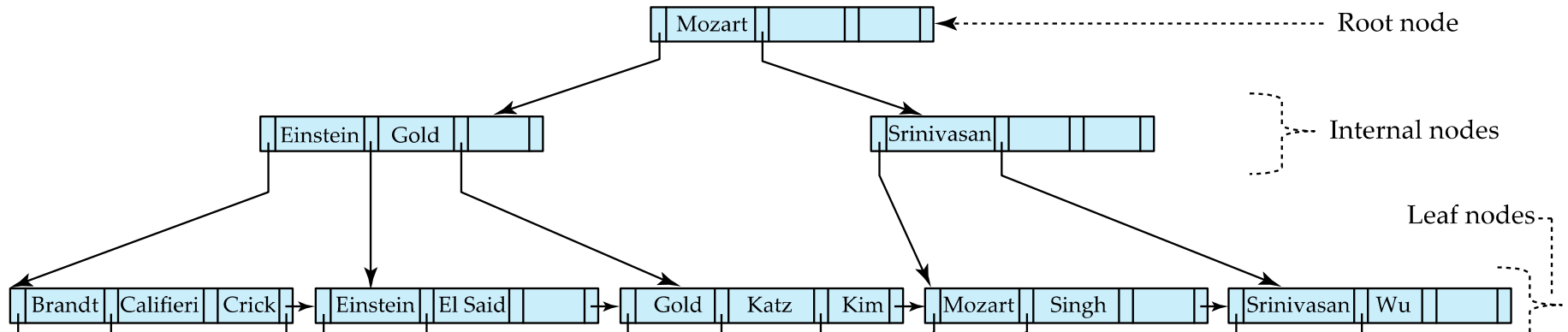
# B<sup>+</sup>树示例



# B<sup>+</sup>树的查询

- 查询

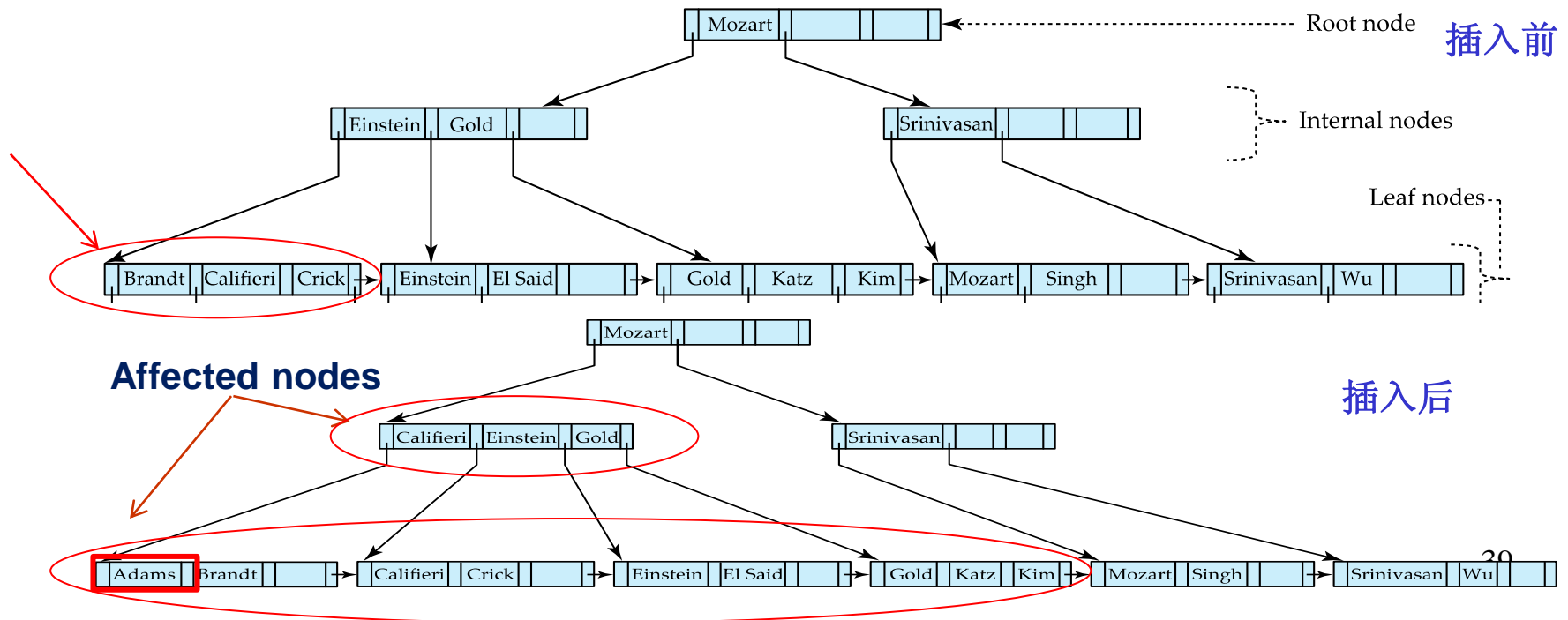
- 从树的根节点开始，通过比较查询码值 $v$ 与节点的 $k_i$ 值，向下遍历树，直到到达包含指定值的叶节点为止。如果在叶子节点中找到 $k_i=v$ ，则返回目标记录指针 $p_i$ ；否则关系中不存在该值的记录



# B<sup>+</sup>树的更新

## • 插入

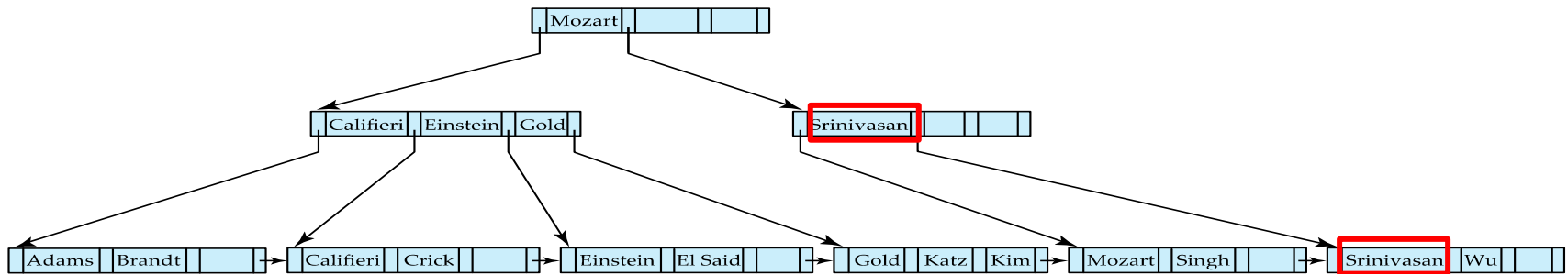
- 采用查询算法定位插入的叶节点 $l$ ，如果有空间则插入，否则需要拆分
- 拆分：将节点 $l$ 的 $n$ 个码值的前 $\lceil n/2 \rceil$ 放在 $l$ 中，剩下值放在新节点中；将新节点插入 $l$ 的父节点中。如此自底向上递归处理，直到插入不再产生拆分或建立了一个新根节点为止
- 示例：name值为Adams的记录



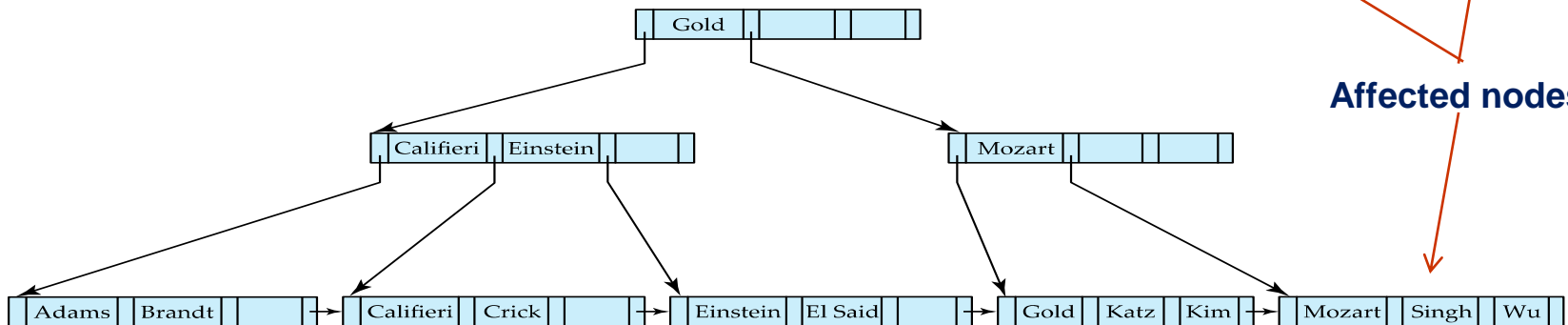
# B<sup>+</sup>树的更新

## • 删除

- 采用查询算法定位删除的叶节点*l*进行删除，如果*l*的码值个数低于下限，可能导致节点合并或重新分配
- 合并或重新分配：如果删除后节点*l*太小，将其与兄弟节点合并，并从父节点中删除，如此自底向上递归处理，直到根节点。如果在合并时，新节点码值数超过上限，则需要该节点与其兄弟节点之间重新分配指针



Before and after deleting "Srinivasan"

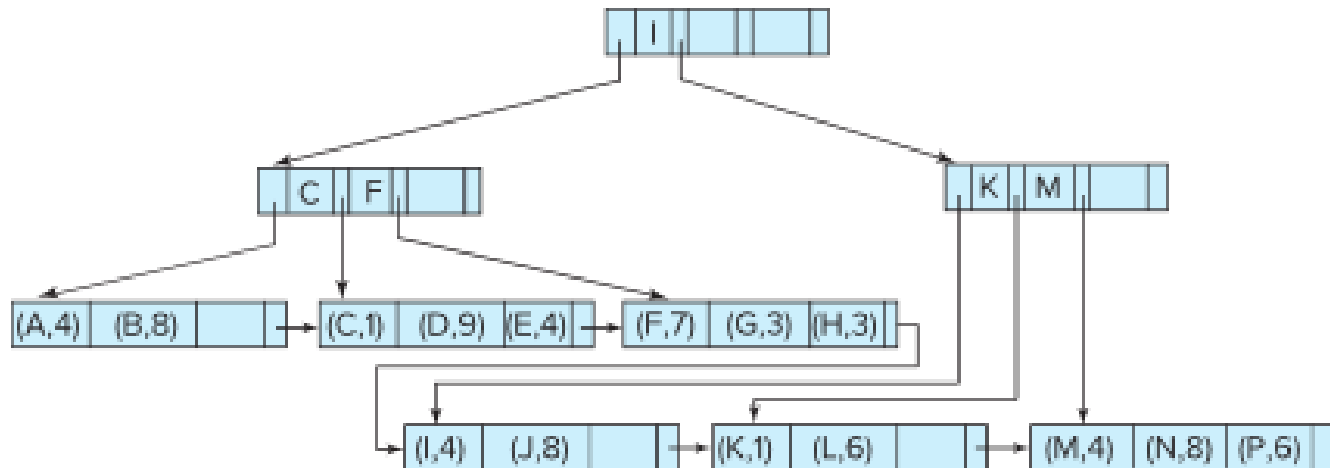


Affected nodes



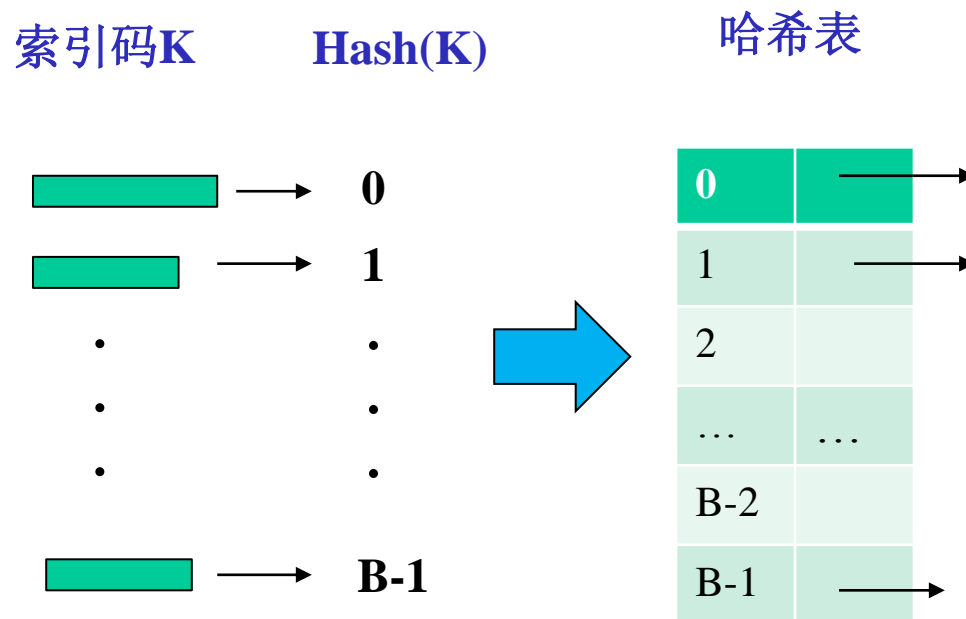
# B<sup>+</sup>树文件组织

- 在B<sup>+</sup>树文件组织中，叶节点存储的是记录而不是记录的指针
- 用B<sup>+</sup>树索引解决索引顺序文件组织的这种性能下降问题
- 由于记录通常比指针大，所以要求叶节点半满，而最大记录数要小于中间节点的指针数
- 插入和删除操作算法与B<sup>+</sup>树索引相同
- 示例：



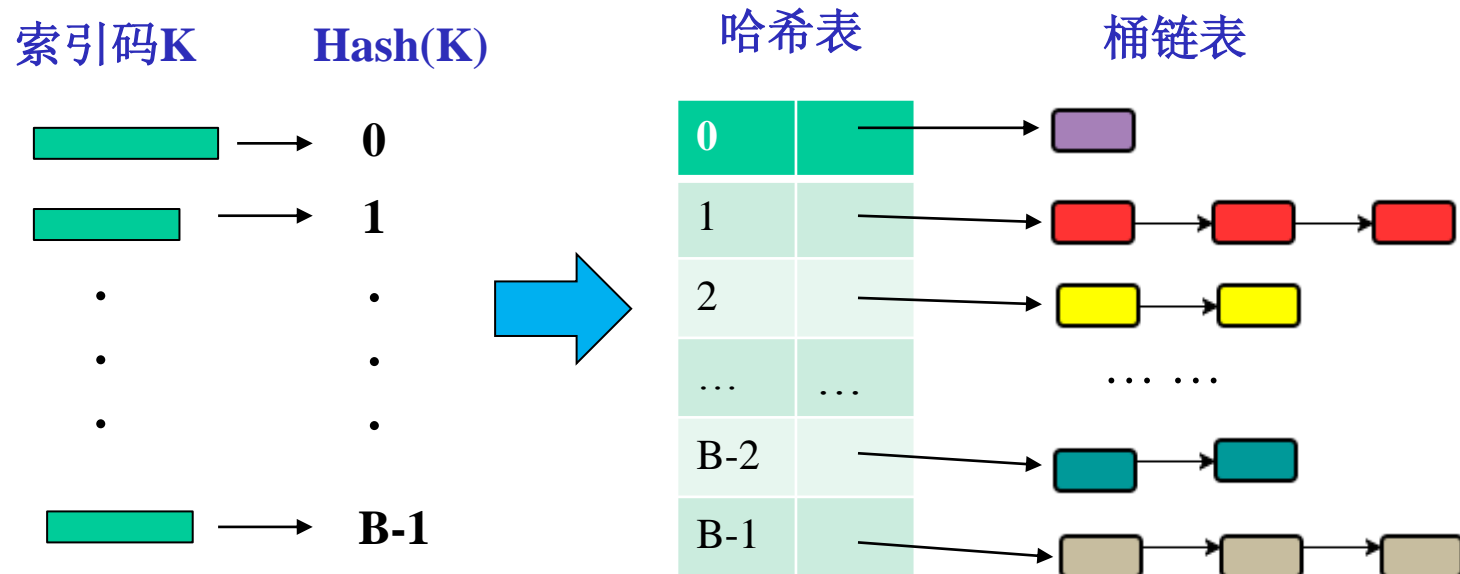
# Hash索引

- 基于哈希表 (Hash table) 实现
- 哈希表实现key到value的映射。通过键值映射到表中一个位置来访问记录，这个映射函数叫做 Hash 函数，存放记录的数组叫做哈希表。



# Hash方法

- 哈希表的实现包括两部分
  - 哈希函数：将很大的key空间映射到比较小的域，用于计算桶/槽数组的元素序号；非用于加密算法的哈希函数；计算速度快且碰撞率低
  - 哈希方案（scheme）：解决一个哈希值对应多条记录。最常使用溢出链接(Chaining)法



# Hash方法

---

- 静态哈希：哈希表的大小是固定的
  - 文件增大时，太多的溢出桶将降低访问性能
  - 数据规模缩小时，会造成空间浪费
- 动态哈希：允许哈希表的大小动态修改
  - 定期重哈希：创建新的大的哈希表，把原表上的key重新哈希到新表上
  - 线性哈希：以一种递增的方式重新哈希

# Hash索引

---

- 周期性重组的开销大
- 适用于检索哈希码具有特定值的记录检索。不适用于区间值的检索，以及部分匹配检索
- 有很多重复值的列，不适于做key

# 小结

---

- 物理存储系统
- 数据存储结构/物理结构
- 缓存管理
- 索引