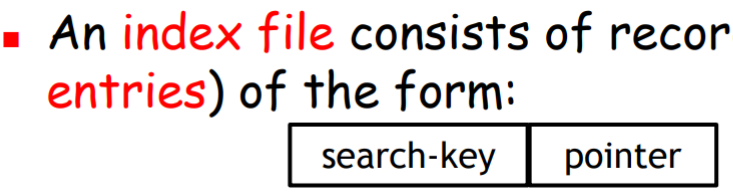
W2

Index Techniques

motivation：更快地查找所需数据

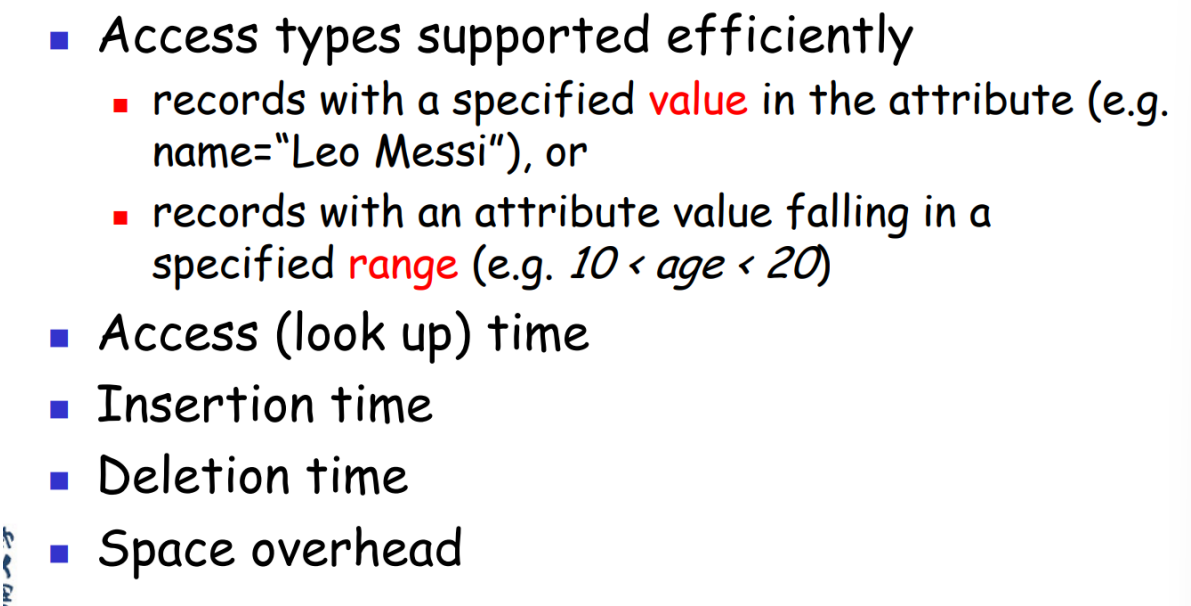
The structure of index

- search key: attributes used to look up records in a file
- data file: collection of blocks holding records on disk
- **index file**：一种更快地查找records数据结构。
  - (key，pointer)的集合



- 比原文件小很多

evaluation



range search：查找一个范围内的records (e.g. 10 < age < 20)

index file的组织方式

- ordered index
- hashing index（不常用，但泛用）

Ordered index

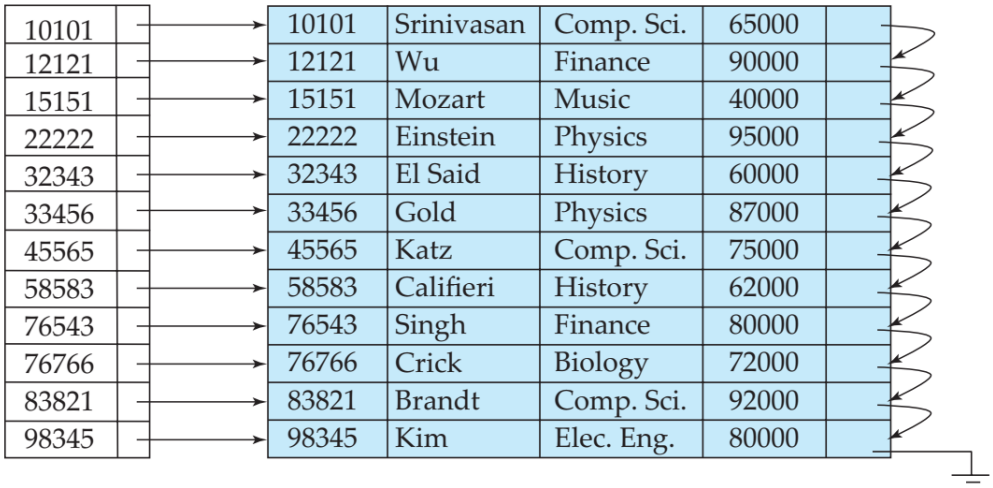
Dense Index 和 Sparse Index是根据索引条目与数据记录的对应关系进行分类的。

Primary Index 和 Secondary Index是根据索引建立在哪些列上以及是否影响数据存储顺序进行分类的。

dense index

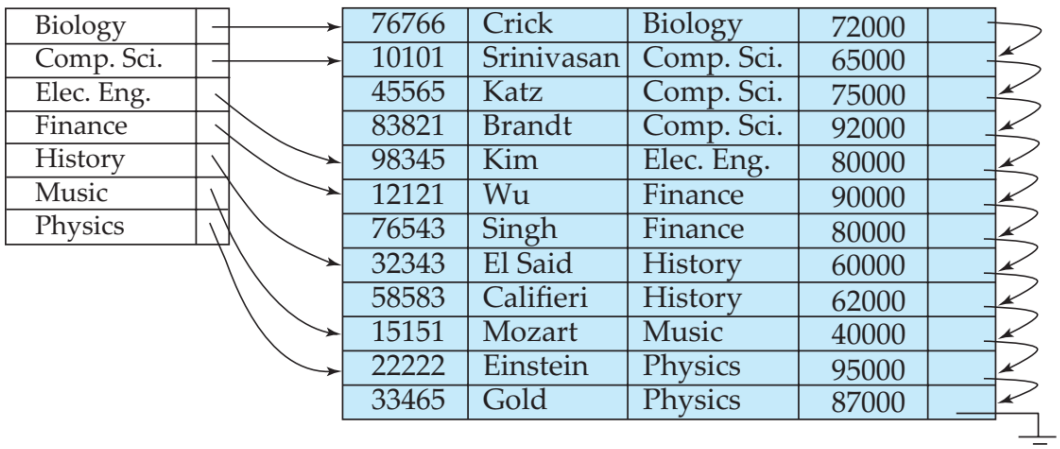
以ID建立索引

- e.g. index on *ID* attribute of instructor relation



以dept_name建立索引

- Dense index on *dept_name*, with *instructor* relation sorted on *dept_name*



每个search key都有对应的index record，且该索引表覆盖了所有记录，因此这是一个**密集索引**

sparse index

只包含部分key value

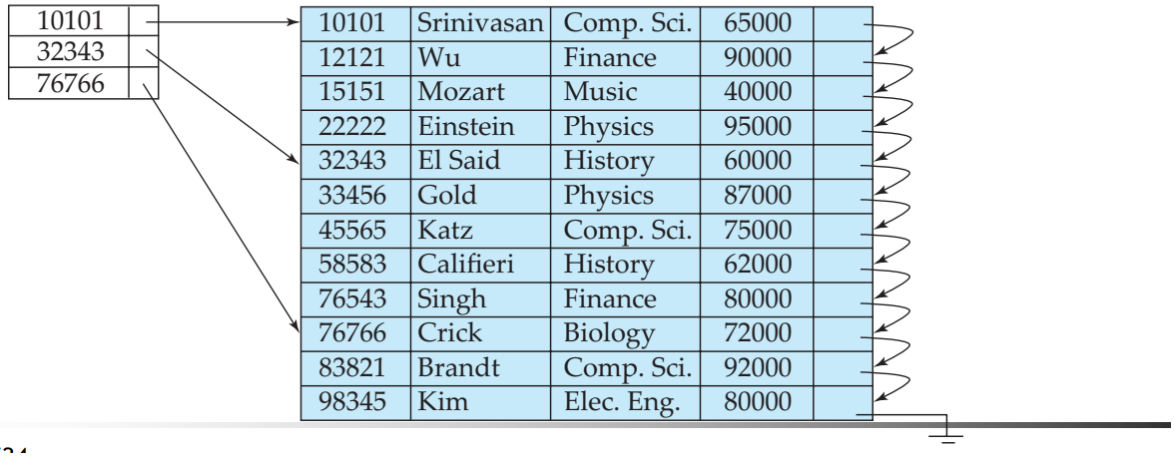
example：**one index entry per block.**

要求：data file必须是顺序文件

查找k时，找到最大的小于k的value，然后顺序查找

To locate a record with search-key value K,

- Find index record with largest search-key $value<K$
- Search file sequentially starting at the record to which the index record points



trade-off：密集索引直接查找快，稀疏省空间，增删更少的变动。用稀疏索引对于每一个block存一个对应块内最小值的index。

Primary index

an index whose search key specifies the sequential order of the file.

- called **clustering** index. 数据的物理存储顺序与索引顺序一致。
- Can be sparse

Secondary index

辅助索引，索引顺序与文件存储顺序无关。

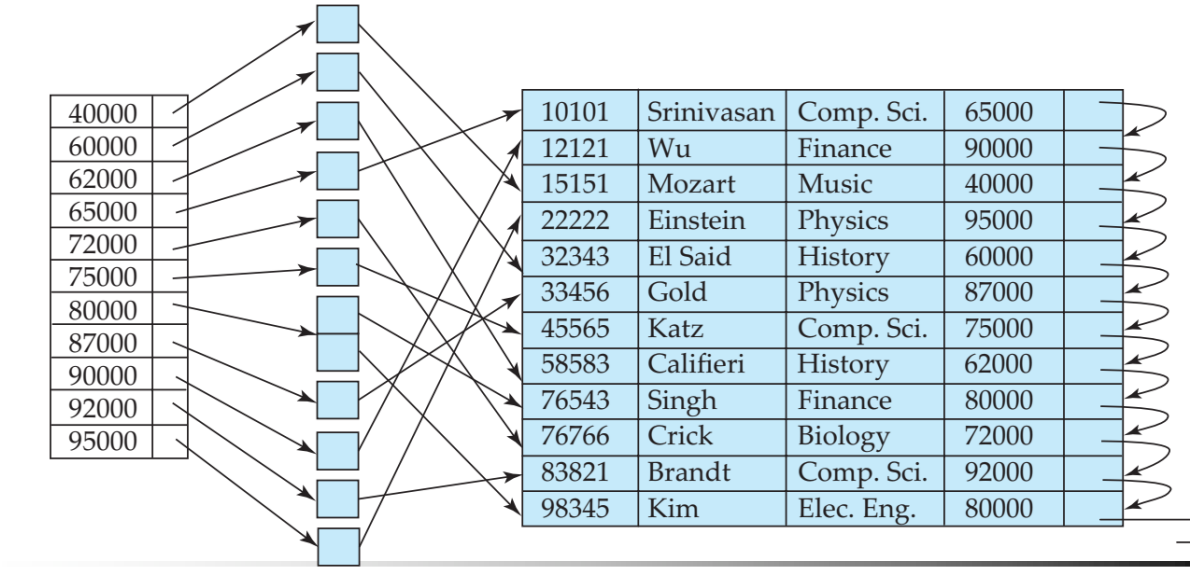
- non-clustering index
- Can not be sparse

Index-sequential file: ordered sequential file with a primary index

Index record points to a **bucket** that contains **pointers** to all the actual records with that particular search-key value

辅助索引必须是密集的

- Example: secondary index on *salary* field of instructor relation



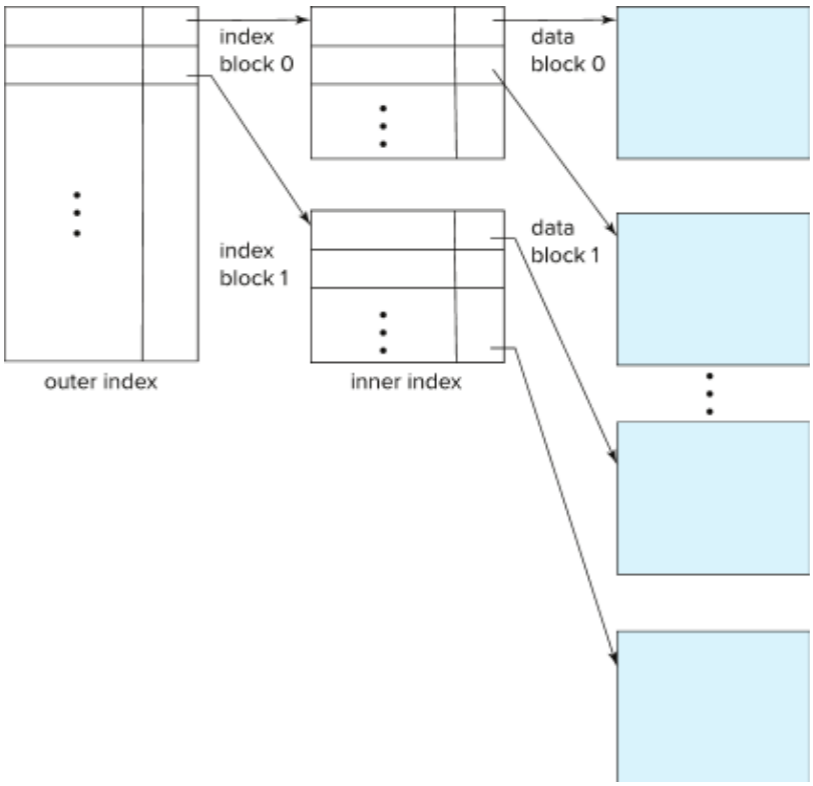
更新record需要修改索引，会增加开销。主索引线性scan很快，辅助索引很慢

Multilevel Index

Index on index

Index on index

- outer index – a sparse index of primary index
- inner index – the primary index file



B+ tree

基本概念

优点是在插入和删除时以较小的改动自平衡（都是对数时间）。额外的时间和空间开销，但瑕不掩瑜。

B+树 is “short” and “Fat”，一个节点包含：

- K_i are the search-key values
- P_i are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes)

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

只有叶节点有record；自下而上构造树

性质

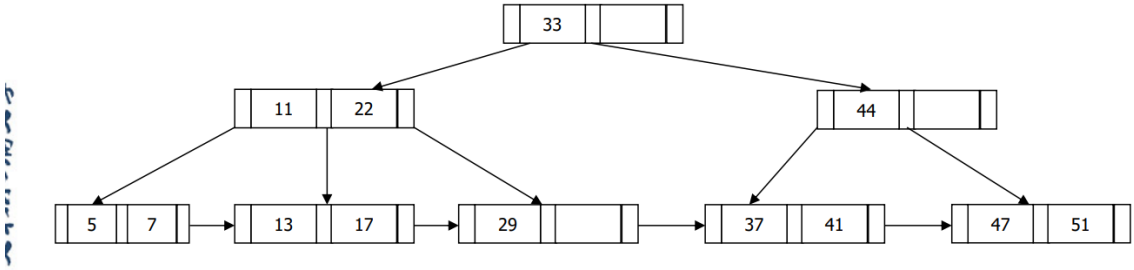
- n表示一个节点内的指针数量
- key按顺序分布（不重复）
- 所有叶节点深度一样
- **root node** 至少有两个 **children**
- **inner node** 至少有 $n/2$ 个 **children**（向上取整）
- **leaf node** 至少 $(n-1)/2$ 个 **key** (向上取整)

主要是后三个对于节点的规定

特殊情况：

如果root 就是 leaf， 数量在0到n-1之间

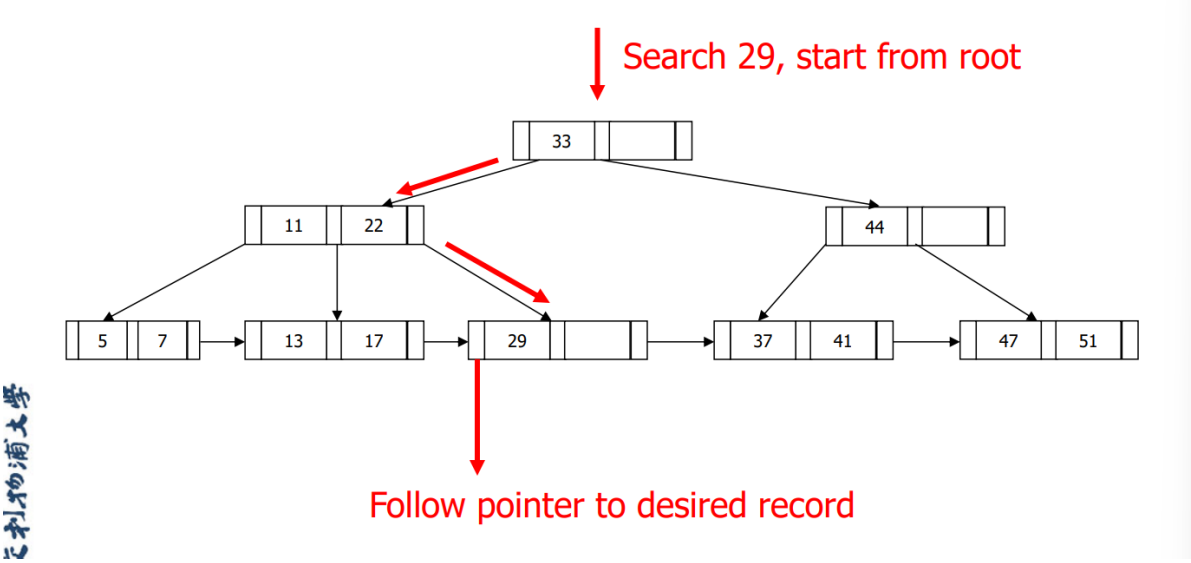
- An Example B+-Tree with n = 3
  - All paths have same length.
  - Root has (at least) two children
  - In each non-leaf node (inter node), at least half (≥ ⌈3/2⌉ =2) pointers are used
  - Each leaf node contains at least ⌈(3-1)/2)⌉ =1 key



inner node level是稀疏索引

最大高度$log(n/2)K$, K代表key数量

查找



典型的n大概是100

插入

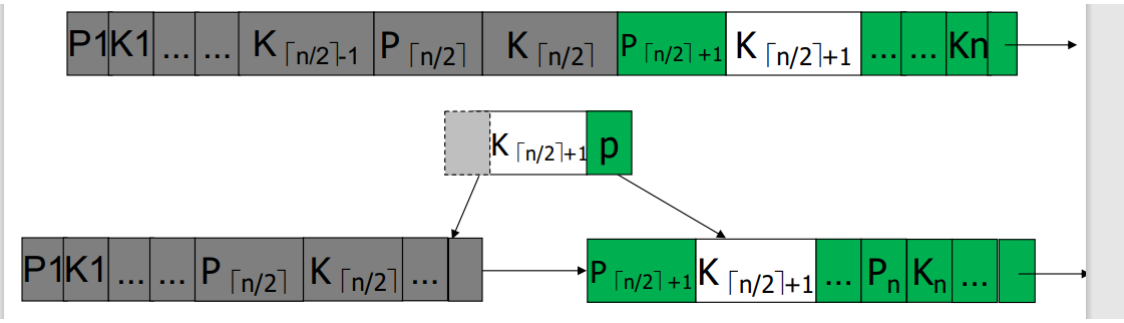
叶节点的插入

1. Find the leaf node in which the search-key value would appear
2. If the search-key value is already present in the leaf node
   - 2.1. Add record to the file
   - 2.2. If necessary add a pointer to the bucket.
3. If the search-key value is not present, then
   - 3.1. add the record to the main file (and create a bucket if necessary)
   - 3.2. If there is room in the leaf node, insert (key-value, pointer) pair in the leaf node
   - 3.3. Otherwise, **split** the node (along with the new (key-value, pointer) entry) as discussed in the next slides.
4. Splitting of nodes proceeds upwards till a node that is not full is found.
   - In the worst case the root node may be split, increasing the height of the tree by 1.

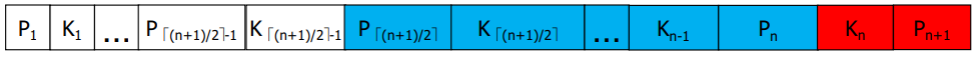
概括地说，节点过载+节点分裂+向上复制/移动

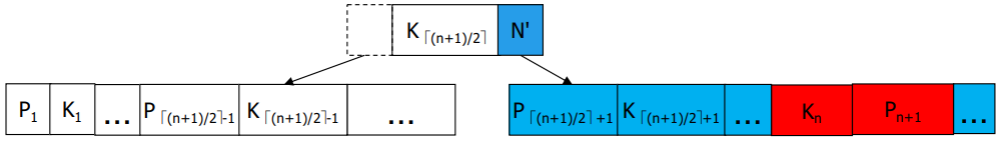
分裂

叶节点：前$\lceil n/2 \rceil$个key保留在原node中，剩下的放到新node中，将新节点的第一个$(\lceil n/2 \rceil + 1)$**复制**到父节点。

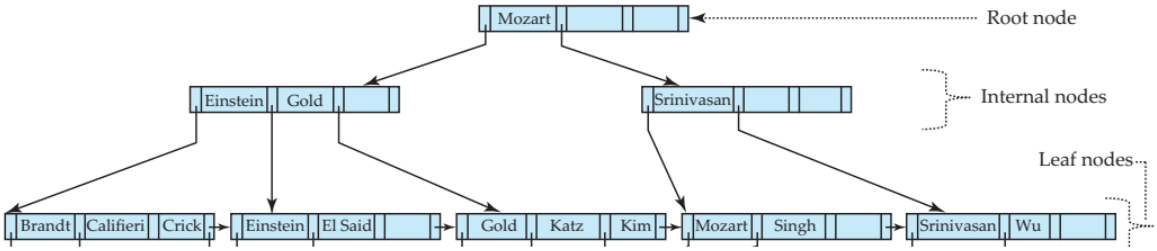


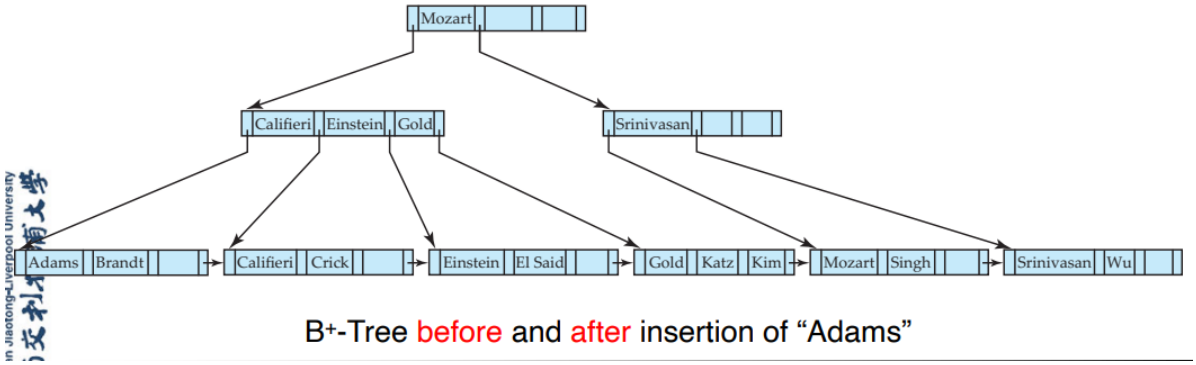
非叶节点：前 $\lceil (n+1)/2 \rceil$ -1 不动，新节点第一个 $\lceil (n+1)/2 \rceil$ 移动到父节点





example：





删除

两种策略，先后执行

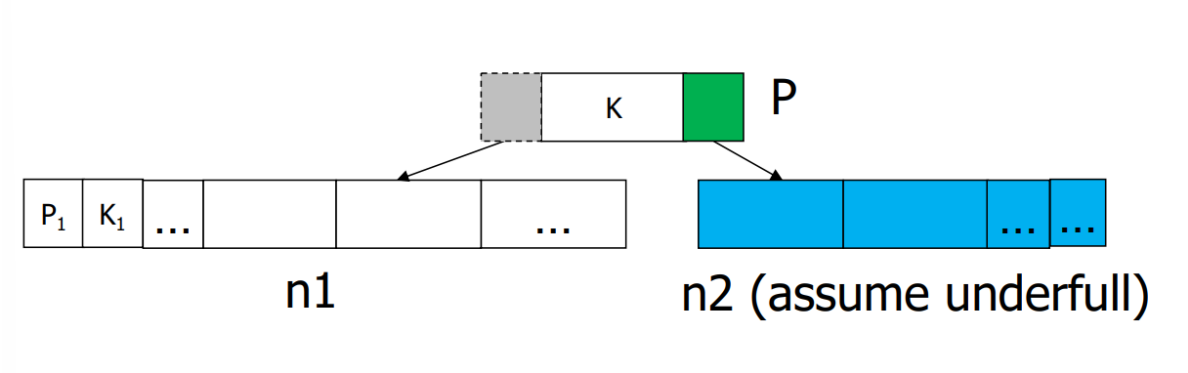
merge

对于叶节点

- merge sibling节点的key （sibling先左后右）
- 删除父节点的key

对于非叶节点

- merge sibling节点和父节点的key
- 删除父节点的key



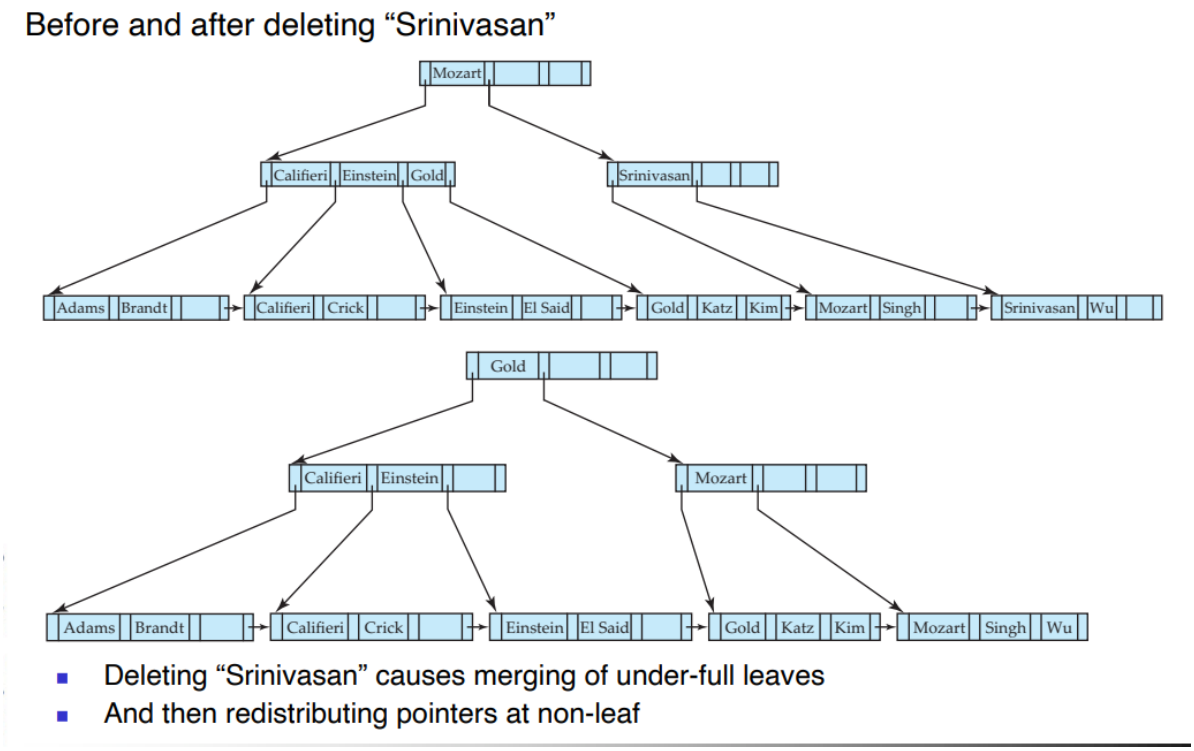
redistribute

叶节点

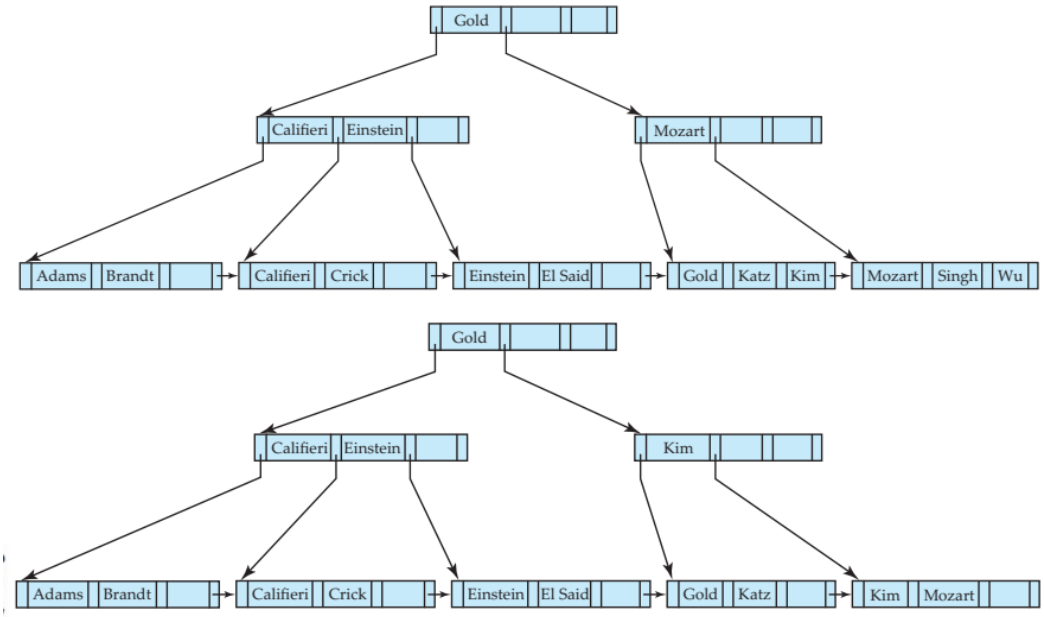
- 把n1最后一个Ki复制到n2
- 删掉n1 ki
- 用Ki 替换掉父节点K

非叶节点

- 把n1最后一个Ki复制到n2
- 用n1 ki 替换K
- 删掉n1 ki

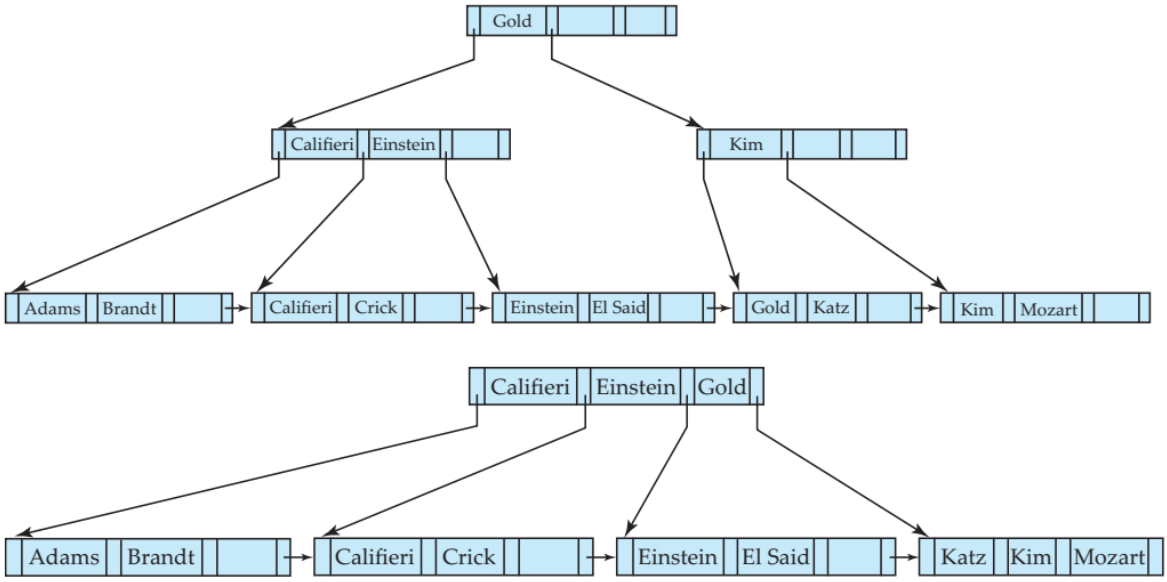


Before and after deleting “Singh and Wu”



- Deleting “Singh” can be done directly
- Deleting “Wu”: redistributing pointers at leaf level

Before and after deleting “Gold”



- Deleting “Gold”: merge leaf nodes first and then merge non-leaf nodes