计算机领域本科教育教学改革试点
工作计划（“101计划”）研究成果

数据结构

授课教师：张羽丰

湖南大学　信息科学与工程学院

第 14 章

索引

提纲

问题的引入：索引和查询

- **背景**
  - “大数据”存放在大型数据库或大型文件系统中
    - 需要支持高效插入、删除、更新和查找等操作
  - **索引**是**高效查找**的基础，而查找又是其它操作的基础
- **索引**
  - 书籍的索引，方便读者快速查找名词的定义、图表的位置
  - 字典中的索引，可以按拼音、笔画、偏旁等快速查找到需要的字词
- **查找**
  - 在数据元素集合中，通过一定的方法找出与给定的关键码相同的数据元素的过程

14.2 索引的定义及基本概念

- 输入顺序文件（Entry-Sequenced File）
  - 按照记录进入系统的顺序存储记录
  - 相当于一个磁盘中未排序的线性表，因此不支持高效的检索
- 主码（Primary Key）
  - 数据库中区别每条记录的唯一标识
    - 例如：公司职员信息的记录的主码可以是职员的身份证号码
- 辅码（Secondary Key）
  - 数据库中可以出现重复值的码
  - 辅码索引把一个辅码值与具有该辅码值的记录的主码值关联起来

基本概念

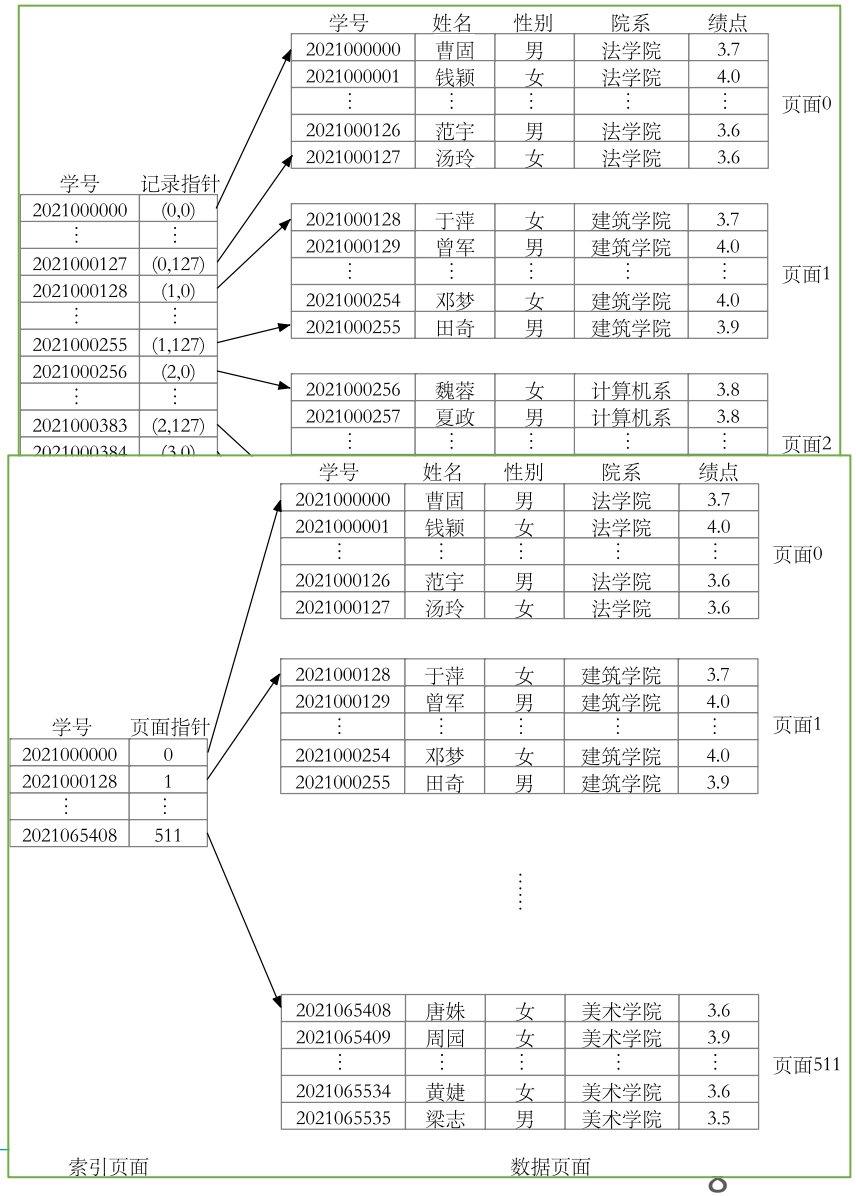
- 索引（Indexing）
  - 把一个关键码与它对应的数据记录的位置相关联的过程
    - （关键码，指针）对，即（key, pointer）
    - 指针指向数据文件中的完整记录
- 索引文件（Index File）
  - 用于记录这种联系的文件组织结构
  - 一个主文件可以有多个索引文件
    - 每个索引文件往往基于不同的关键码字段
    - 不需要重新排列主文件

基本概念

- 索引技术是组织大型数据库的一种重要技术，可以通过索引文件高效访问记录中该关键码值
  - 支持高效的检索
  - 插入、更新、删除

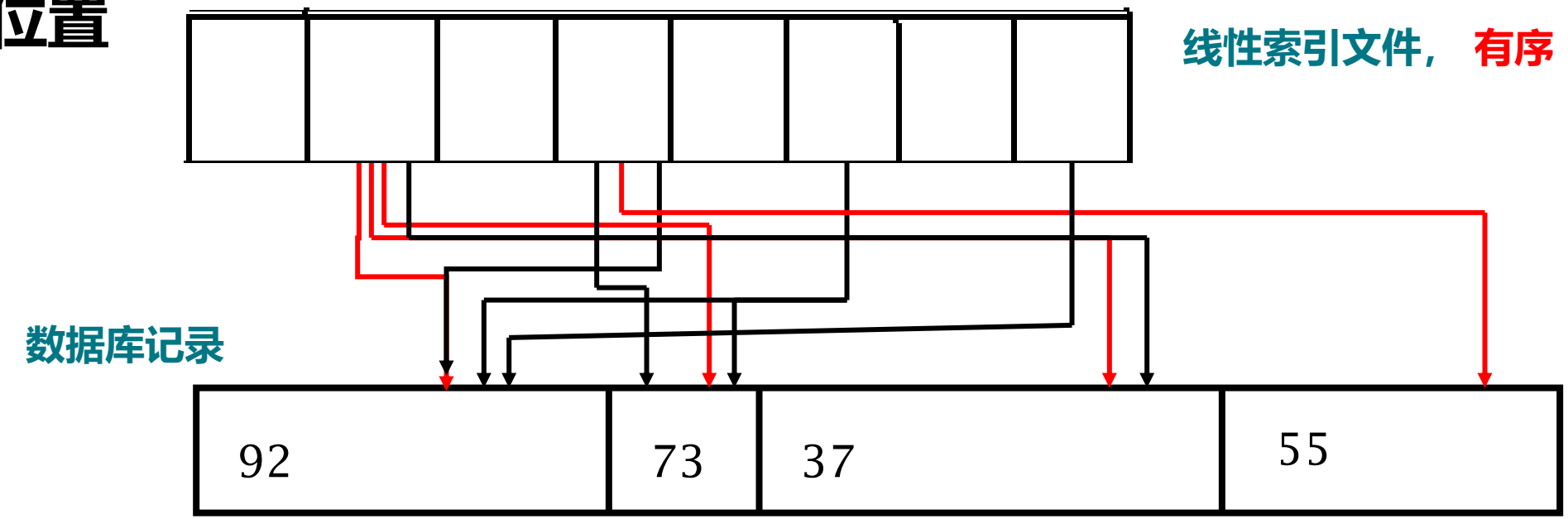
稠密索引 VS 稀疏索引

- **稠密索引**
  - 对每个记录建立一个索引项
  - 主文件可以不按照关键码的顺序排列
- **稀疏索引**
  - 对一组记录建立一个索引项
  - 记录需按照关键码的顺序存放
  - 可以把记录分成多个组（块）
  - 索引指针指向这一组记录在磁盘中的起始位置



14.3 线性索引与静态索引

- **线性索引：按照关键码顺序进行排序**
  - **可以基于二分法进行快速检索**
  - **指针指向存储在磁盘上的文件记录起始位置或者主索引中主码的起始位置**

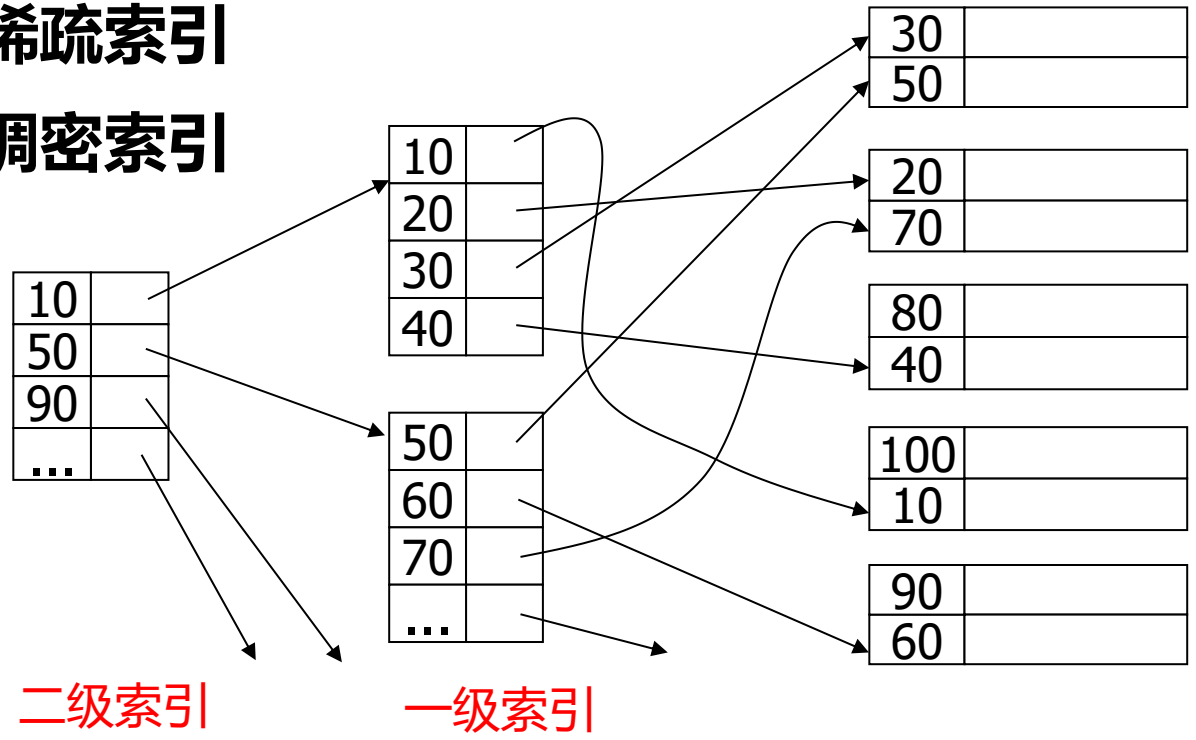


线性索引

- **问题**
  - 线性索引太大，存储在磁盘的多个块中，影响检索效率
    - 一次检索过程可能多次访问几个磁盘块
    - 使用**二级线性索引**
  - 更新线性索引需要整体移动后面的索引记录，效率低
    - 适用于数据静态不变的场景，索引构建后，基本不需要更新
    - 所以也称为**静态索引**

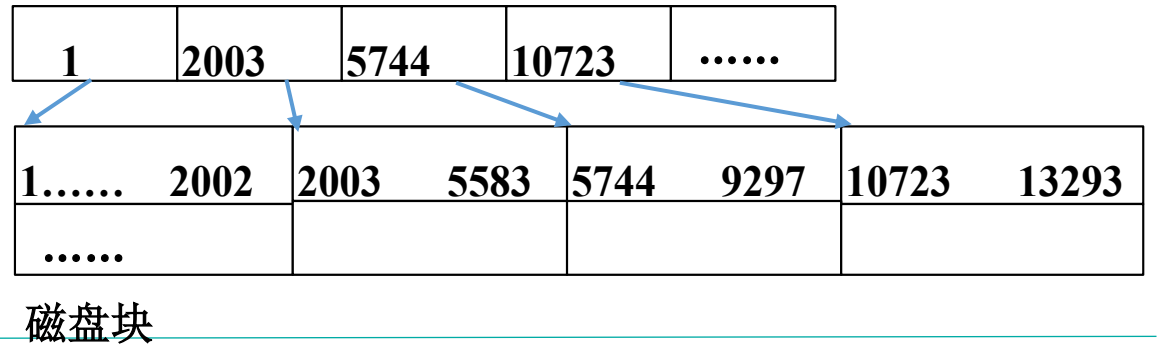
二级线性索引

- **二级线性索引：在一级索引上构建的索引**
  - **二级线性索引肯定是稀疏索引**
  - **一级线性索引可以是稀疏索引，也可以是稠密索引**
    - **数据文件有序，则可以采用稀疏索引**
    - **数据文件无序，则应当采用稠密索引**

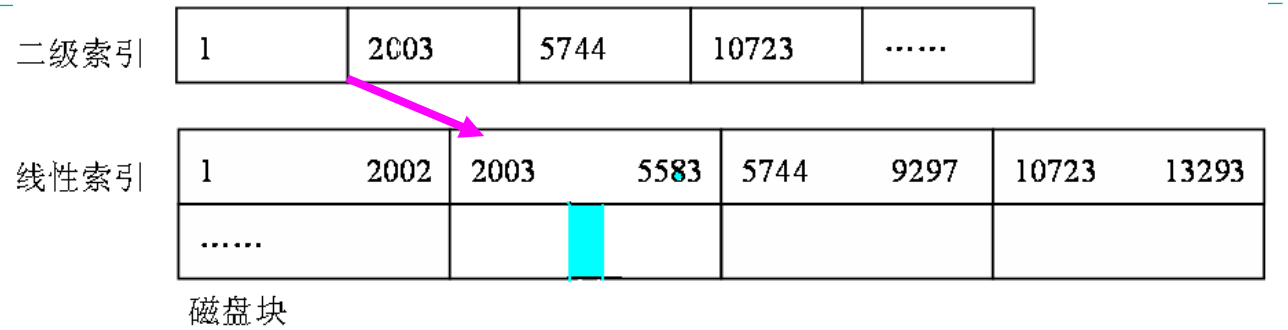


二级线性索引的例子

- 磁盘块大小1kB，每对（关键码，指针）索引对需要8字节
  - 每个磁盘块可以存储 1024 / 8 = 128条这样的索引对
- 假设数据文件包含10K条记录
  - 稠密一级线性索引中包含10k条记录
  - 一级线性索引占用10K/128=80个磁盘块
  - 二级线性索引文件中只有80个索引对
    - 关键码值与相应磁盘块中的第一条记录的关键码值相同
    - 指针指向相应磁盘块的起始位置
    - 可以存放在一个磁盘块



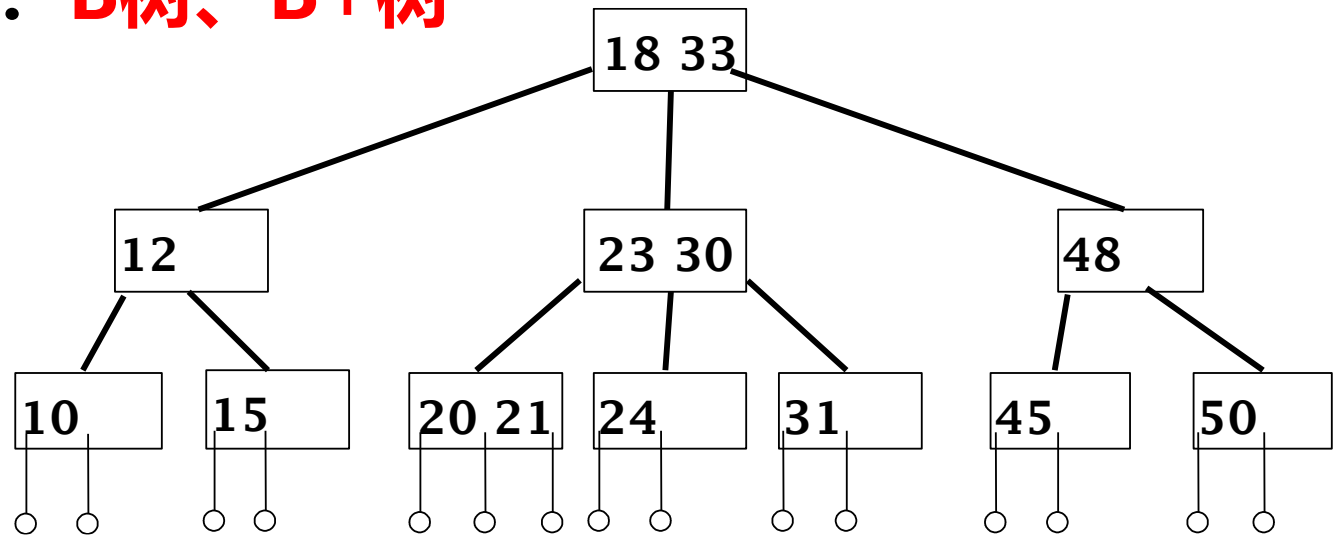
二级线性索引的例子



- **检索关键码为2555的记录**
  - **二级线性索引读入内存**
  - **二分法找关键码的值小于等于2555的最大关键码所在一级索引磁盘块地址—— 关键码为2003的记录**
  - **根据记录2003中的地址指针找到其对应的一级线性索引文件的磁盘块，并把该块读入内存**
  - **按照二分法对该块进行检索，找到所需记录在磁盘上的位置**
  - **最后把所需记录读入，完成检索操作**

14.5 动态索引

- **动态索引结构**
  - 系统运行过程中插入、删除、更新记录时，索引结构本身也可能发生改变
  - **目的**：保持较高的检索效率，降低索引动态维护的开销
  - 常见的有：**B树、B+树**



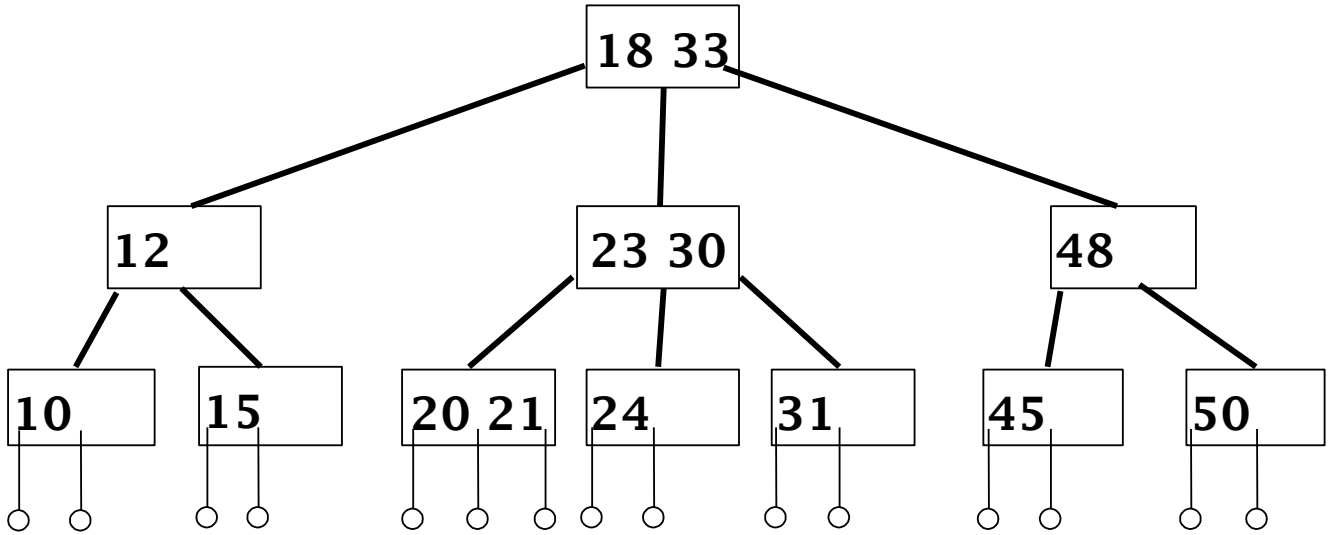
动态索引

- **动态索引：希望保持较高的检索效率，较低的动态维护开销**
  - **AVL树，红黑树是否可以?**
    - **查询、插入、删除的时间复杂度均为logn**
  - **索引的应用场景：海量数据场景**
    - **AVL树、红黑树的高度会比较高，比较次数较多；对于数据存在外存中的情况而言，需要读取磁盘的次数较多**
      - **如1T的数据，树高为40**

14.5.1 B树

- B树：一种平衡的多叉树（Balanced Tree）
  - 平衡
    - 树的检索依赖树的高度，平衡树可以避免退化成链表，提高检索效率
  - 多叉树
    - 降低树的高度，减少读取磁盘的次数，提高检索效率
      - 每个结点一个磁盘块
- m阶B树
  - B树的最大子子节点数，称为阶

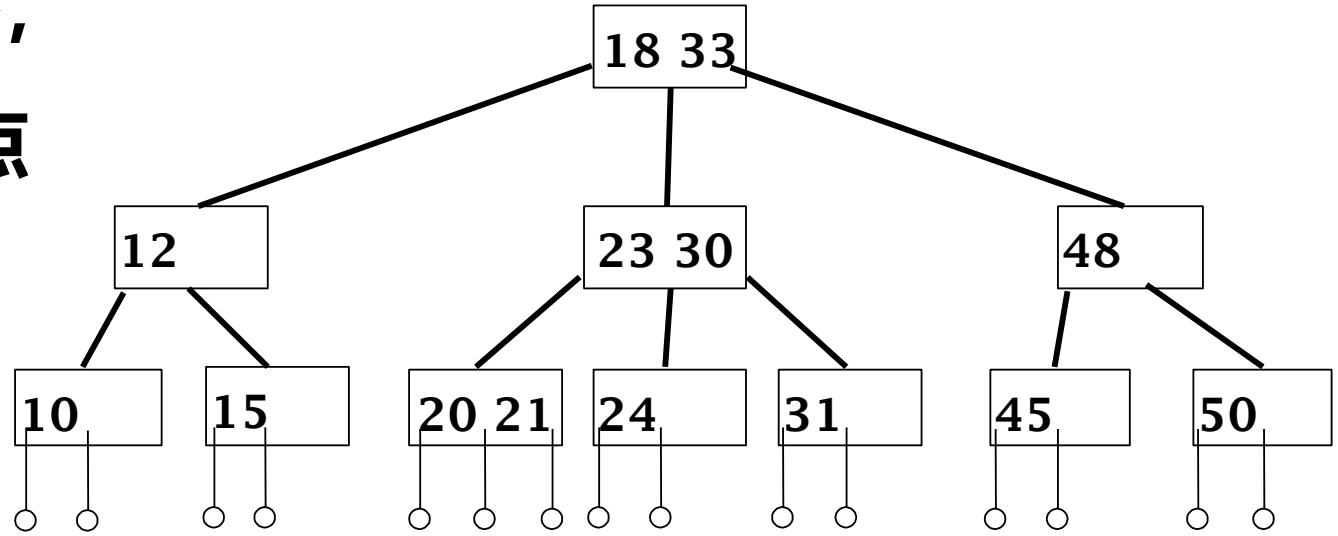
3 阶 B 树



B树

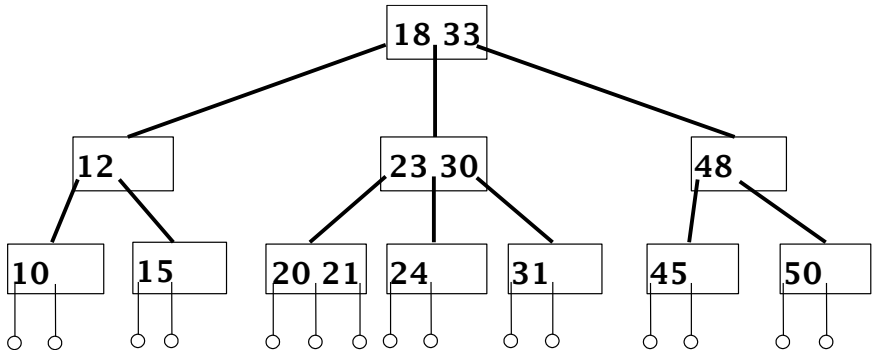
- **m阶B树的结构定义**
  - **m阶**：每个结点至多有m个子结点
  - **树**：k个关键码恰好包含k+1个子结点
  - **平衡性**：所有叶结点在同一层
  - **半满**：除根结点和叶结点外，每个结点至少有$\lceil \frac{m}{2} \rceil$个子节点
  - 根结点至少有两个子节点
    - 例外：空树 或 独根

3 阶 B 树



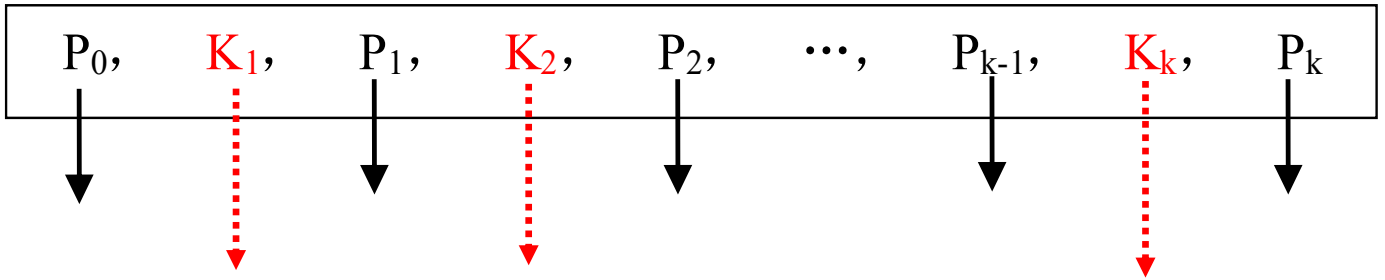
B树的性质

- 平衡性：所有叶结点在同一层
- 关键码没有重复
  - 父节点中的关键码是其子结点的分界
  - 子节点不会重复存储父节点中的关键码
- B树把（值相近）相关记录放在同一个磁盘页中
  - 利用了访问局部性原理
- B树保证树中至少有一定比例的结点是满的
  - 改进空间的利用率
  - 减少检索和更新操作的磁盘读取数目



B树的结点结构

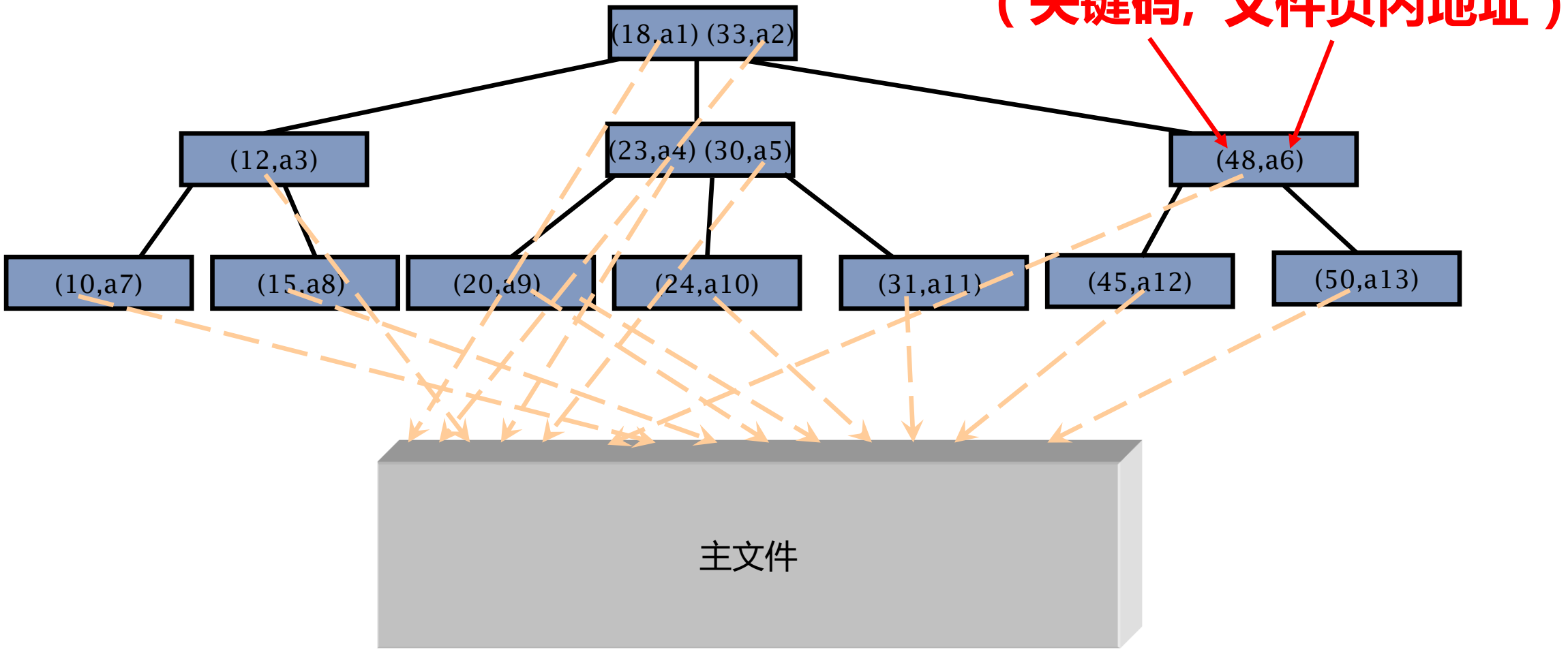
- B树的一个包含k个关键码，k+1个孩子结点的一般形式



  - 其中K_i是关键码的值，$K_1 < K_2 < \ldots < K_j$
  - P_i是指向子树的指针，其中的关键码的值处于K_i到K_{i+1}之间
  - 每个关键码还保存一个指向数据所在文件页内地址的指针
  - 既包含了指向孩子结点的指针，也包含了指向数据的指针
    - 2k+1个指针

B树隐含指针

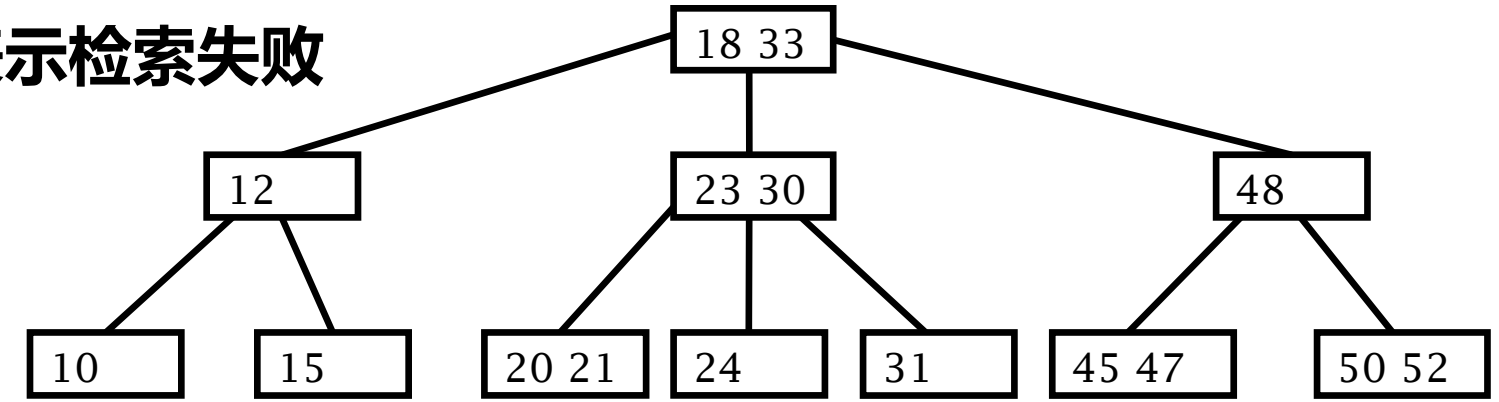
- 数据所在文件页内地址的指针常忽略不画



B树的查找

- 与二叉查找树类似，交替的两步过程
  - 1. 读取根结点，在根结点包含的关键码K_1，K_2，…，K_j中查找给定的关键码
    - 找到，则检索成功
  - 2. 否则，确定要查的关键码值是在某个$[K_i, K_{i+1}]$之间，取P_i所指向的结点继续查找
    - 如果Pi指向空结点，表示检索失败

查找关键码=24的记录

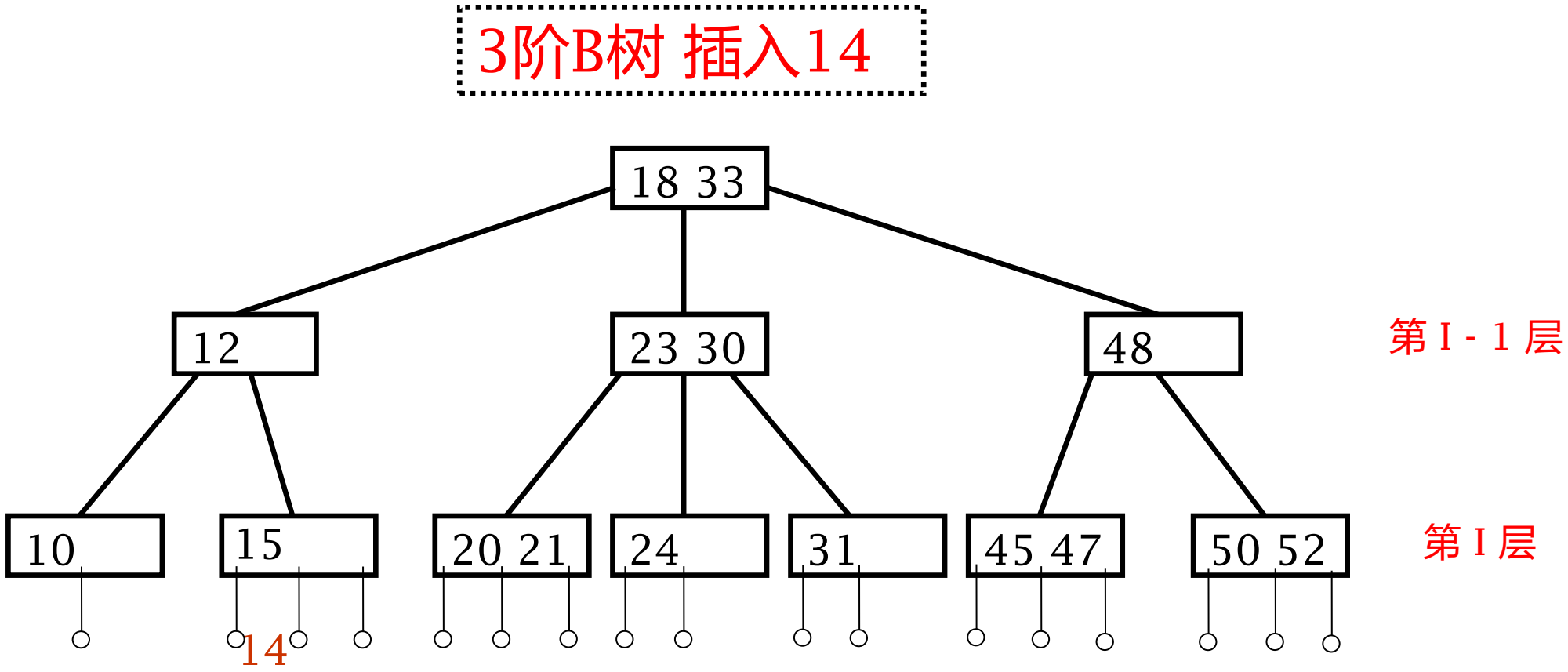


B树的插入

- 类似于二叉查找树，首先找到插入的位置
  - 插入到最底层叶子结点
- 检查阶和等高的限制
  - 若溢出（大于m-1个关键字），则结点分裂为两个结点
    - 中间关键码连同新节点的指针插入父结点（在孩子结点中不保留该关键字）
  - 若父结点也溢出，则继续分裂
    - 分裂过程可能传达到根结点，则新增一个根结点

B树的插入

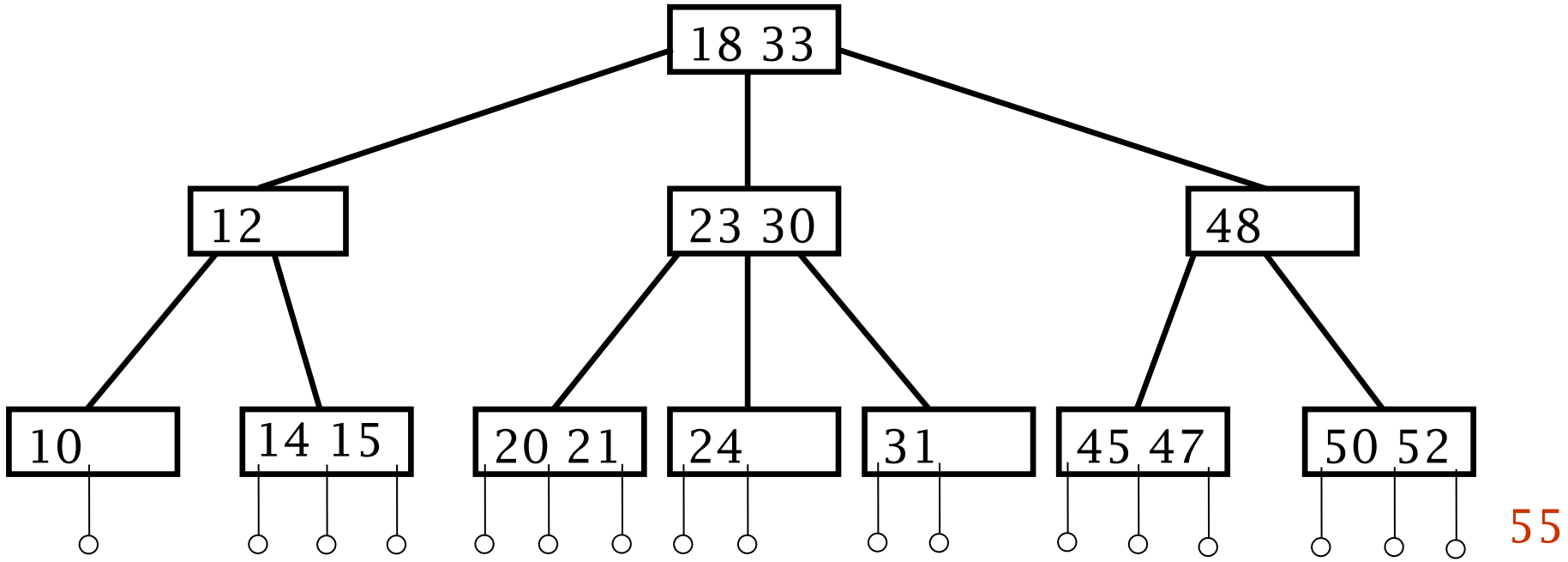
- 插入在最底层叶子结点



B树的插入

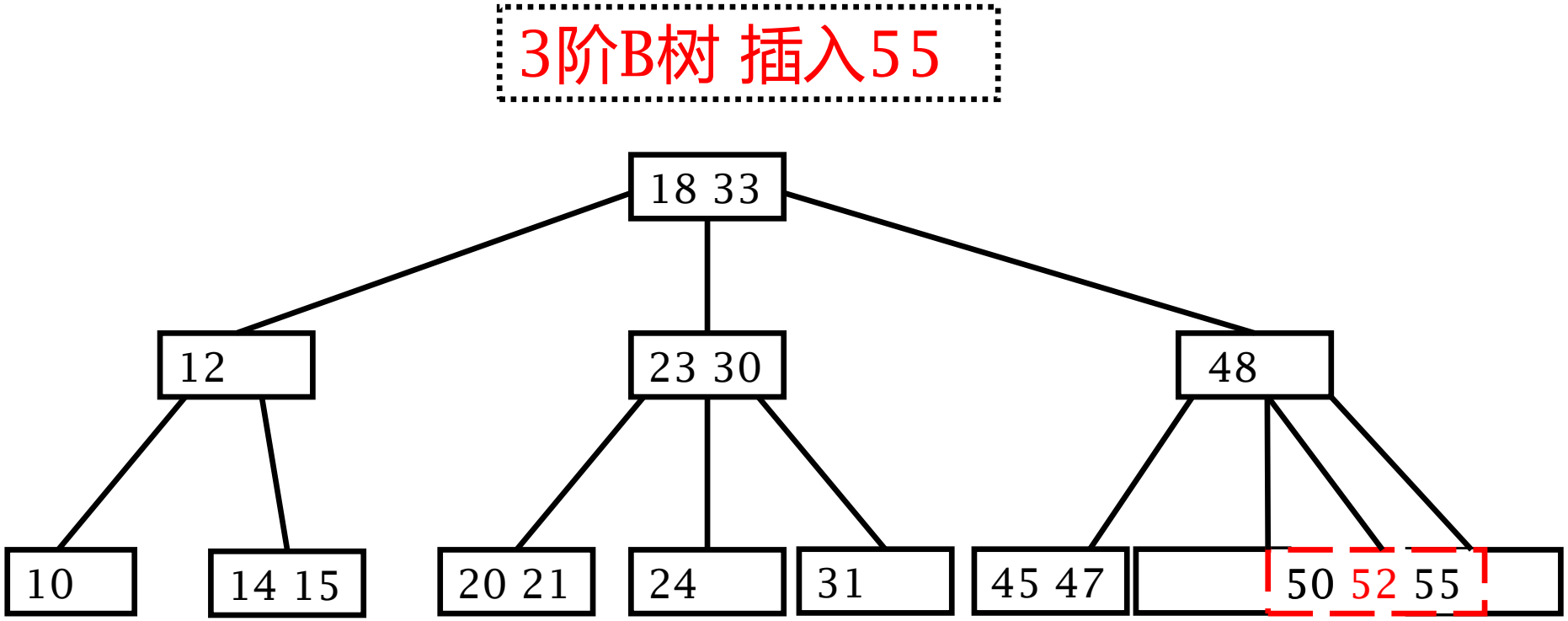
- 结点溢出，从中间关键码处分裂成两个结点
  - 中间关键码提升到父节点

3阶B树 插入55



B树的插入

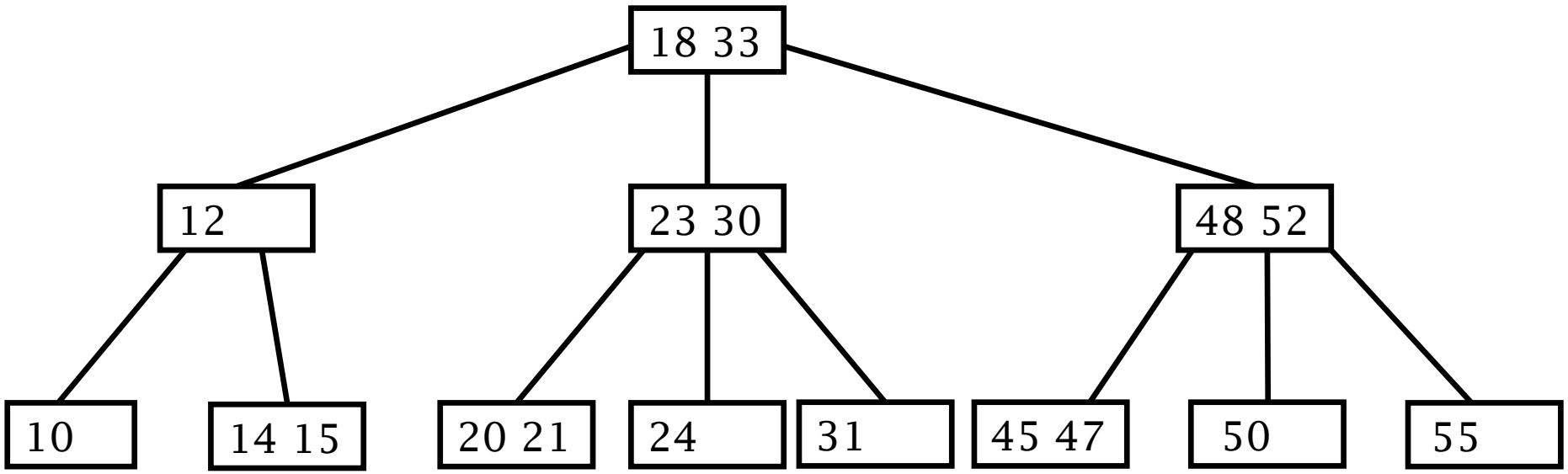
- 结点溢出，从中间关键码处分裂成两个结点
  - 中间关键码提升到父节点



B树的插入

- 结点溢出，从中间关键码处分裂成两个结点
  - 中间关键码提升到父节点

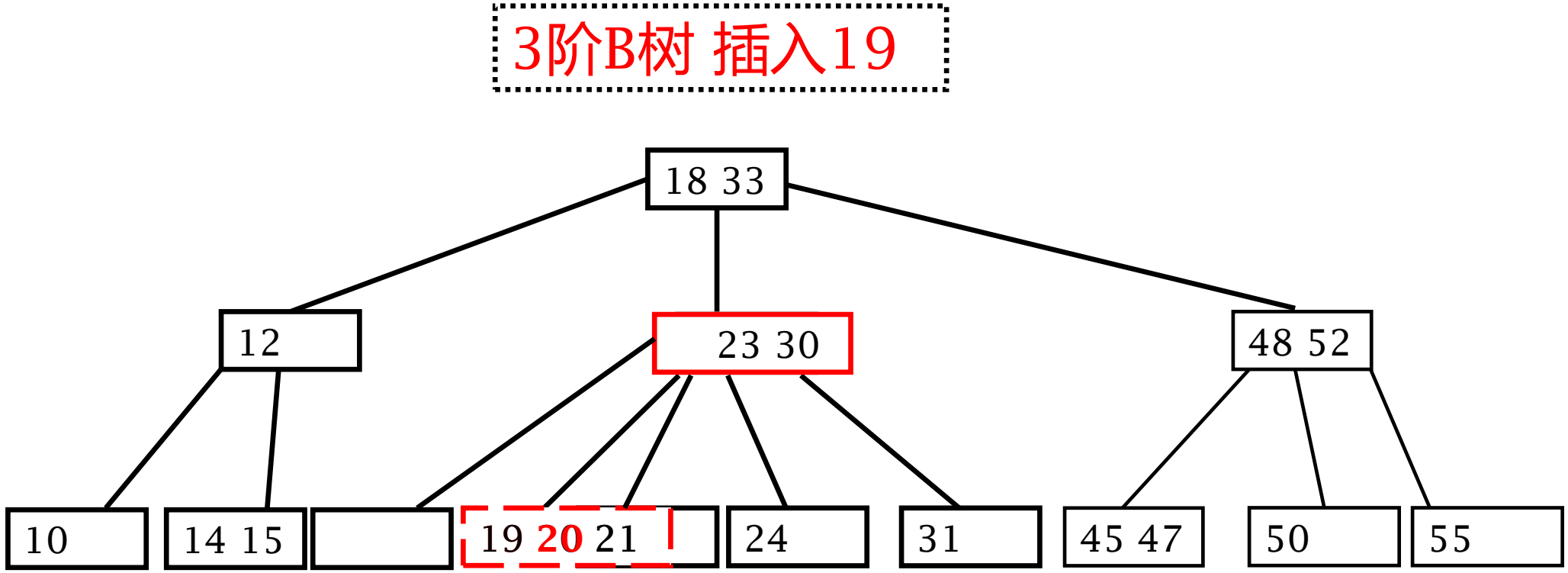
3阶B树 插入19



19

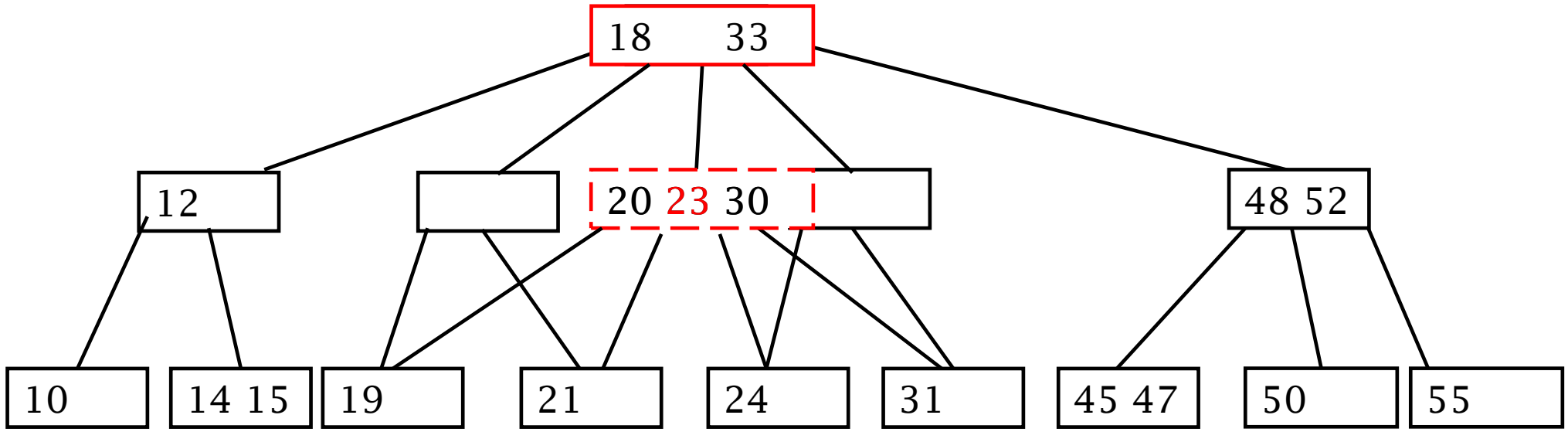
B树的插入

- 页结点分裂



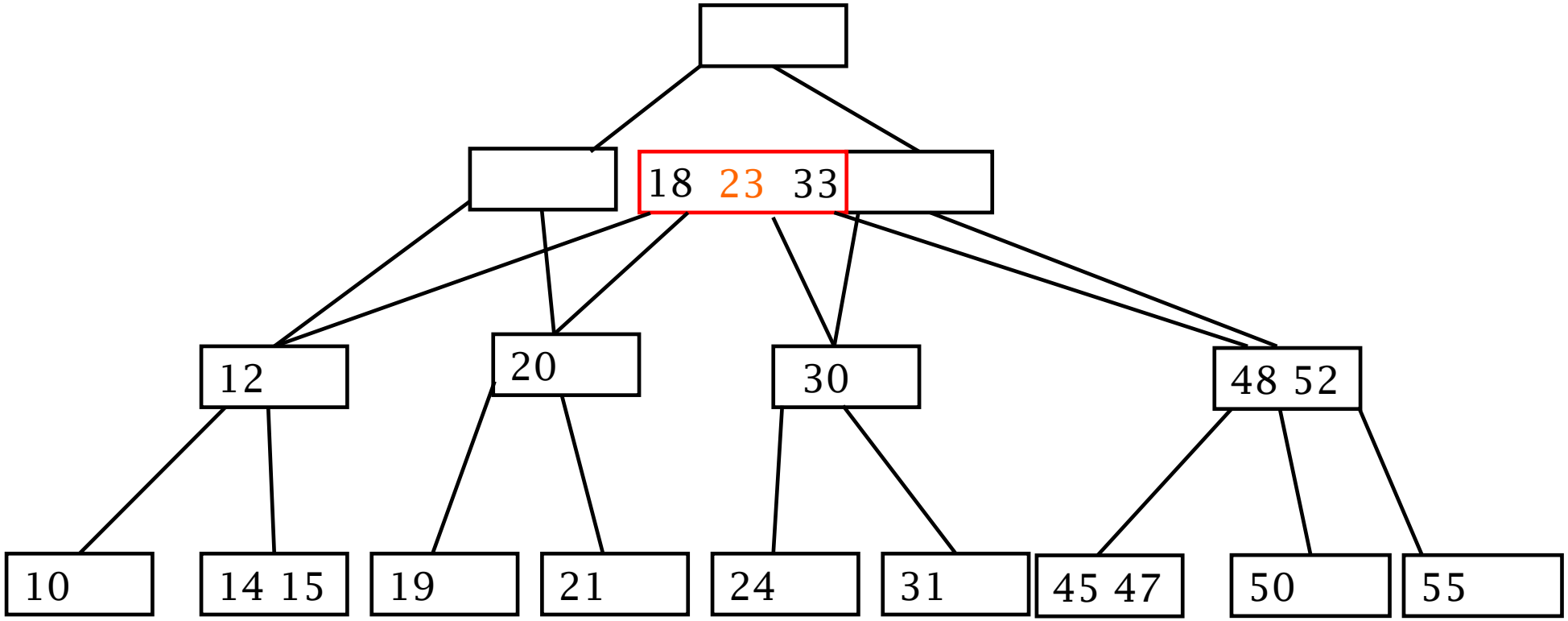
B树的插入

- 第二层结点分裂



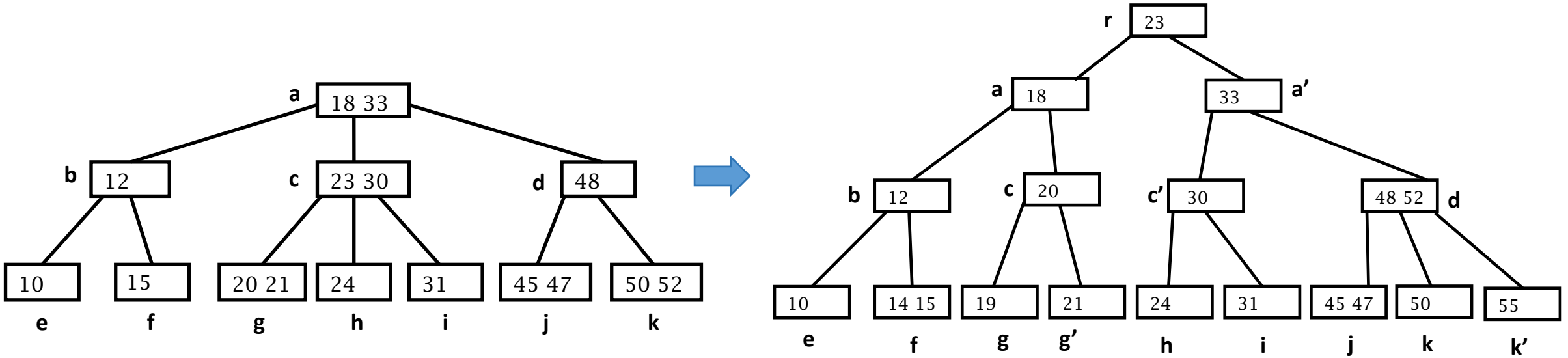
B树的插入

- 根结点分裂



B树操作的访问外存次数

- 连续插入14、55、19，假设访问过的磁盘块都缓存
  - 读盘7次（a, b, f; d, k; c, g)
  - 写盘11次（f; k, k’，d; g, g’，c, c’，a, a’，r)



B树操作的访问外存次数

- 假设内存工作区足够大，向下检索时读入的结点在插入后向上分裂时不必再从磁盘读入
  - 读盘次数与查找相同：树的高度h
  - 写盘次数
    - 依赖于分裂的次数n
      - 每次分裂，需写回分裂后的两个结点
      - 最后一次分裂，还需写回父节点
      - 总的写盘次数为2n+1
    - 最少1次：不分裂，只需将该结点写出磁盘
    - 最多2h + 1次：每层都需要分裂（包括根节点）

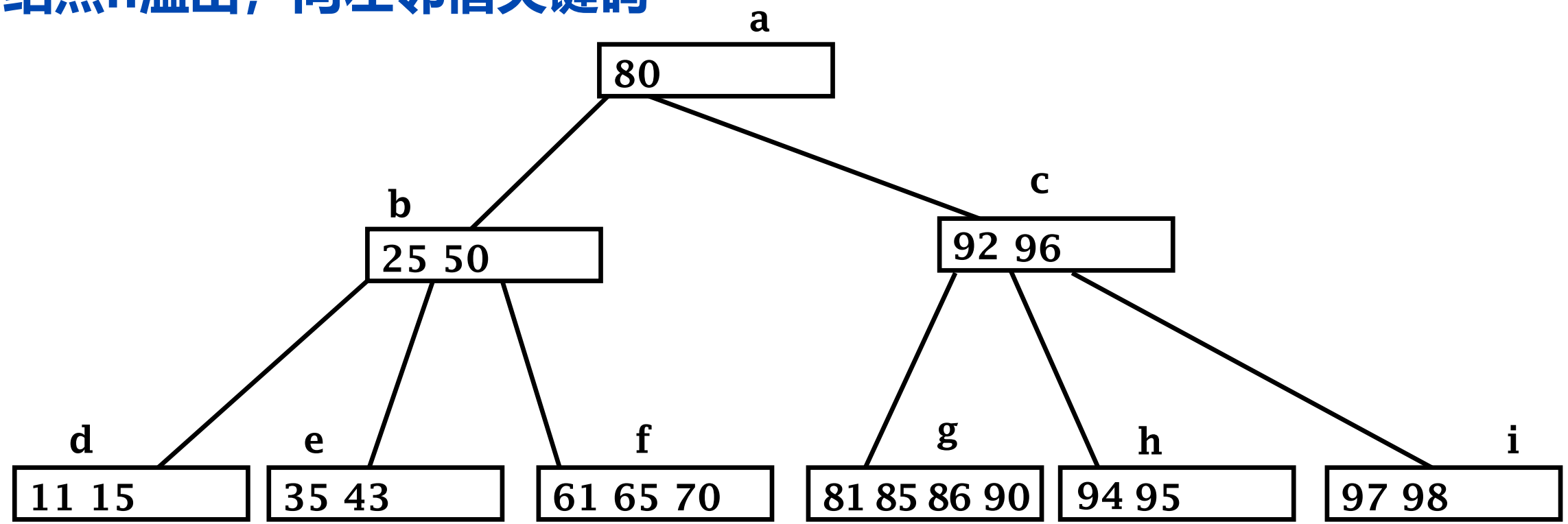
读写盘次数最多为3h+1次

B树的删除

- 查找关键码所在的结点
  - 若为叶结点
    - 结点的关键码个数≥ $\lceil \frac{m}{2} \rceil$（保持半满），直接删除
    - 否则
      - 兄弟结点的关键码个数≥ $\lceil \frac{m}{2} \rceil$，从兄弟结点中移1个关键码到该结点来（保持半满）
        - 父节点中的一个关键码需要做相应变化
      - 如果兄弟结点的关键码个数< $\lceil \frac{m}{2} \rceil$，则进行合并
  - 若为内部结点
    - 用关键码K_i对应的孩子结点P_i中的第一个关键码替换该关键码，删除孩子结点中的关键码

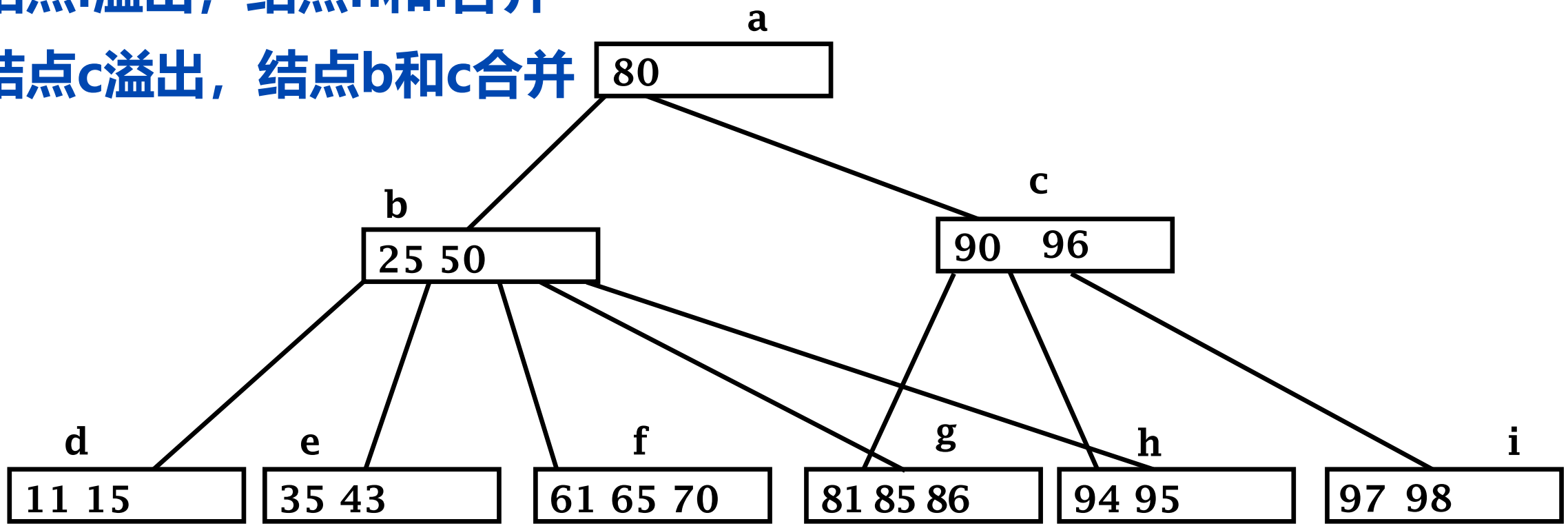
5阶B树的删除示例

- 删除92，为内部结点，递归的与后继交换
- 结点h溢出，向左邻借关键码



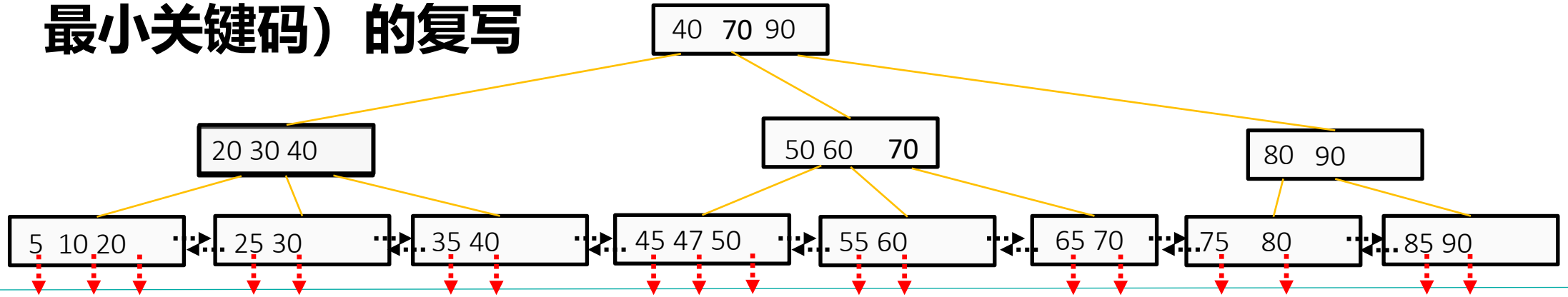
5阶B树的删除示例

- 继续删除96，为内部结点，递归的与后继交换
- 结点i溢出，结点h和i合并
- 结点c溢出，结点b和c合并



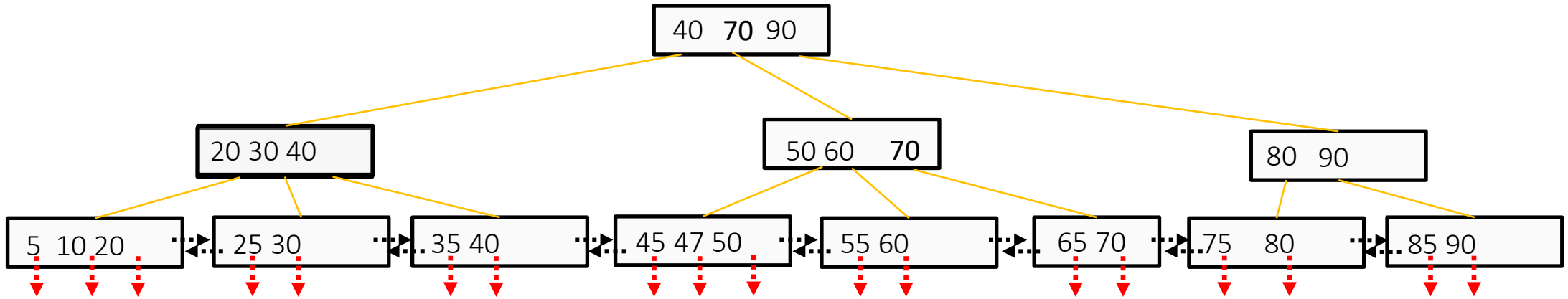
14.5.2 B+树

- **B+树是B树的一种变形**
  - **内部结点不存储数据指针（指向文件页内地址的指针）**
    - **只保存指向孩子结点的指针**
  - **只有叶子结点存储数据指针，所有的关键码均出现在叶子结点上**
    - **无指向孩子结点的指针**
  - **内部各层结点中的关键码均是下一层相应结点中最大关键码（或最小关键码）的复写**



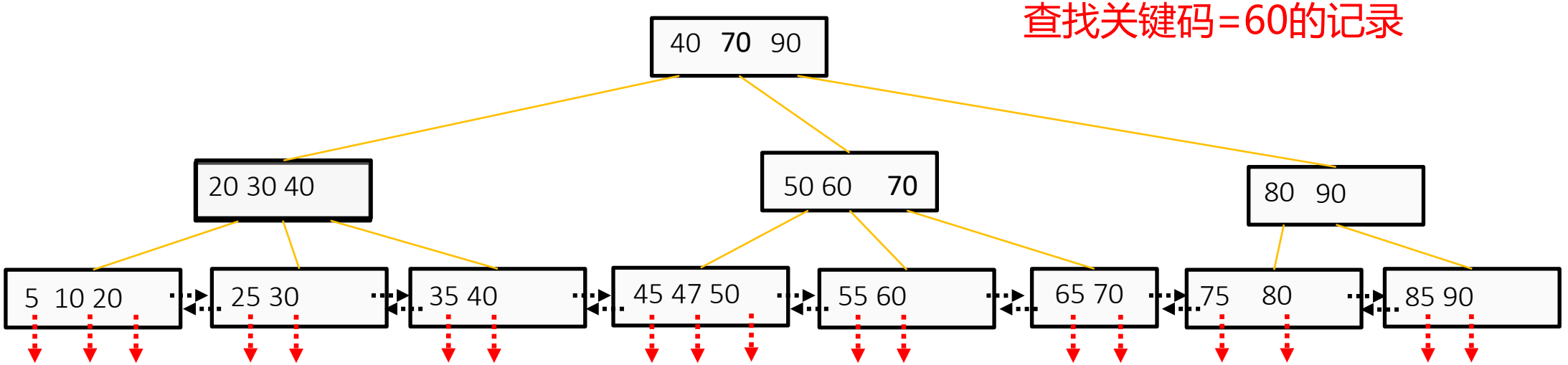
B+树的结构定义

- m阶B+树的结构定义与m阶B树的类似
  - 每个结点至多m个子结点
  - 每个结点（除根结点外），至少有 $\lceil m/2 \rceil$ 个子结点
  - 根结点至少有两个子结点
  - 有K个子结点的结点必有k个关键码



B+树的查找

- 与B树的查找类似
  - 但是一直要找到叶子结点
  - 内部结点即使找到待查的关键码也不能停止

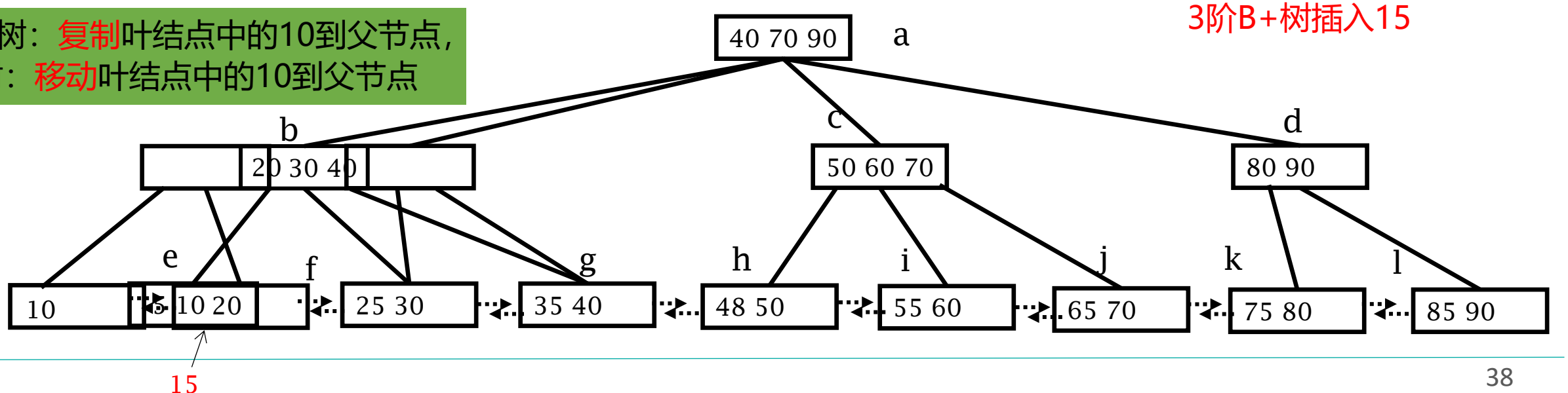


B+树的插入

- 与B树类似
  - 查找关键码应该插入的位置
    - 关键码的插入只在叶结点进行
  - 若结点的关键码的个数>m，则产生分裂
    - 分裂时，需要在父结点中插入关键码

B+树：复制叶结点中的10到父节点，
B树：移动叶结点中的10到父节点

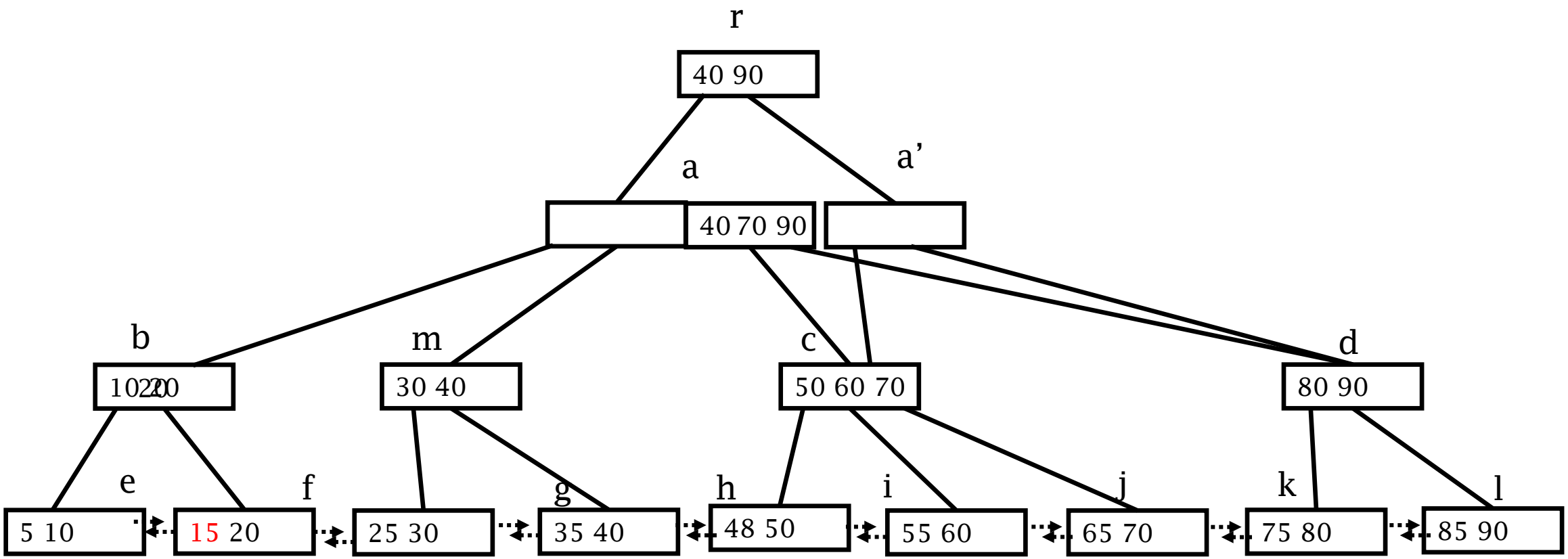
3阶B+树插入15



B+树的插入

- 根节点溢出，分裂，树增高一层

3阶B+树插入15



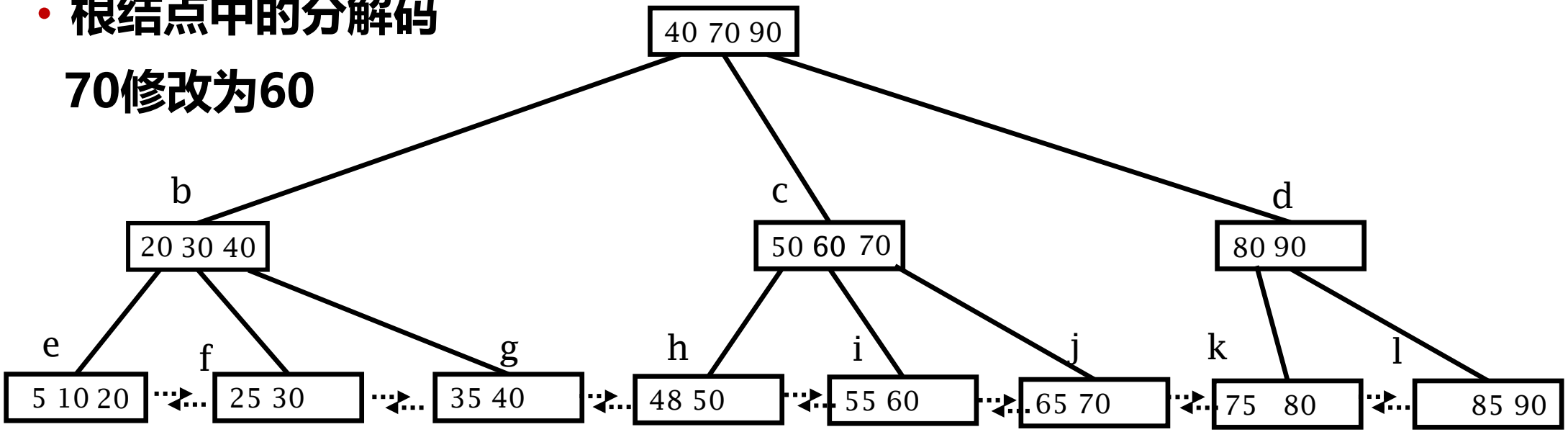
B+树的删除

- 与B树的删除类似
  - 查找关键码所在的叶子结点
  - 当关键码所在结点溢出时，向左或右兄弟借关键码，若兄弟结点无关键码可借，则合并兄弟结点
  - 关键码在叶结点删除后，其在上层的副本可以保留，作为一个“分界关键码”存在
    - 也可替换为新的最大关键码（或最小关键码）

B+树的删除

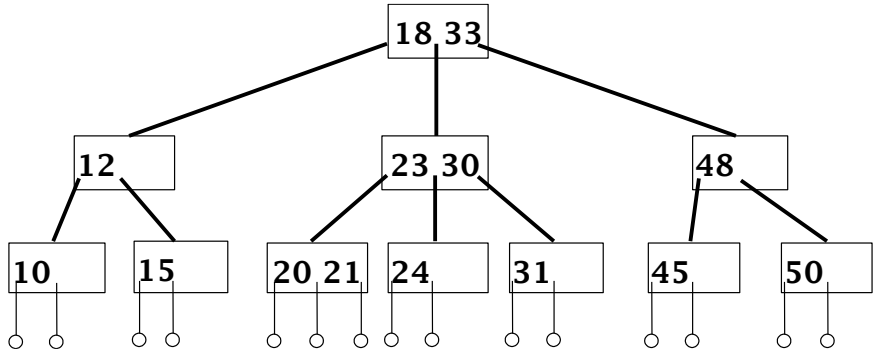
- 在3阶B+树中删除75
  - 结点k溢出，合并结点k与l
  - 结点d溢出，借左邻的关键码
    - 根结点中的分解码

      70修改为60



14.3 B树的性能分析

- **给定关键码的个数N，求B树的最高高度k**
  - **设B树包含N个关键字，则有N+1个扩展的外部空结点。或者说N个关键字，把区间分成了N+1份，分别对应外部节点**
    - **叶子结点数 = 内部结点数 + 1**
  - **各层的结点数**
    - **第0层为根，第1层至少两个结点**
    - **第2层至少$2 \cdot \lceil m/2 \rceil$个结点**
    - **第k层至少$2 \cdot \lceil m/2 \rceil^{k-1}$个结点**



$$N+1 \geq 2 \cdot \lceil m/2 \rceil^{k-1}, \qquad k \leq 1+\log_{\lceil m/2 \rceil}\left(\frac{N+1}{2}\right)$$

  - **保证了B树检索的高效率**

示例

- **关键码个数N=1,999,998，B树的阶m=199**

$$k \le 1 + \log_{\lceil m/2 \rceil}(\frac{N+1}{2})$$

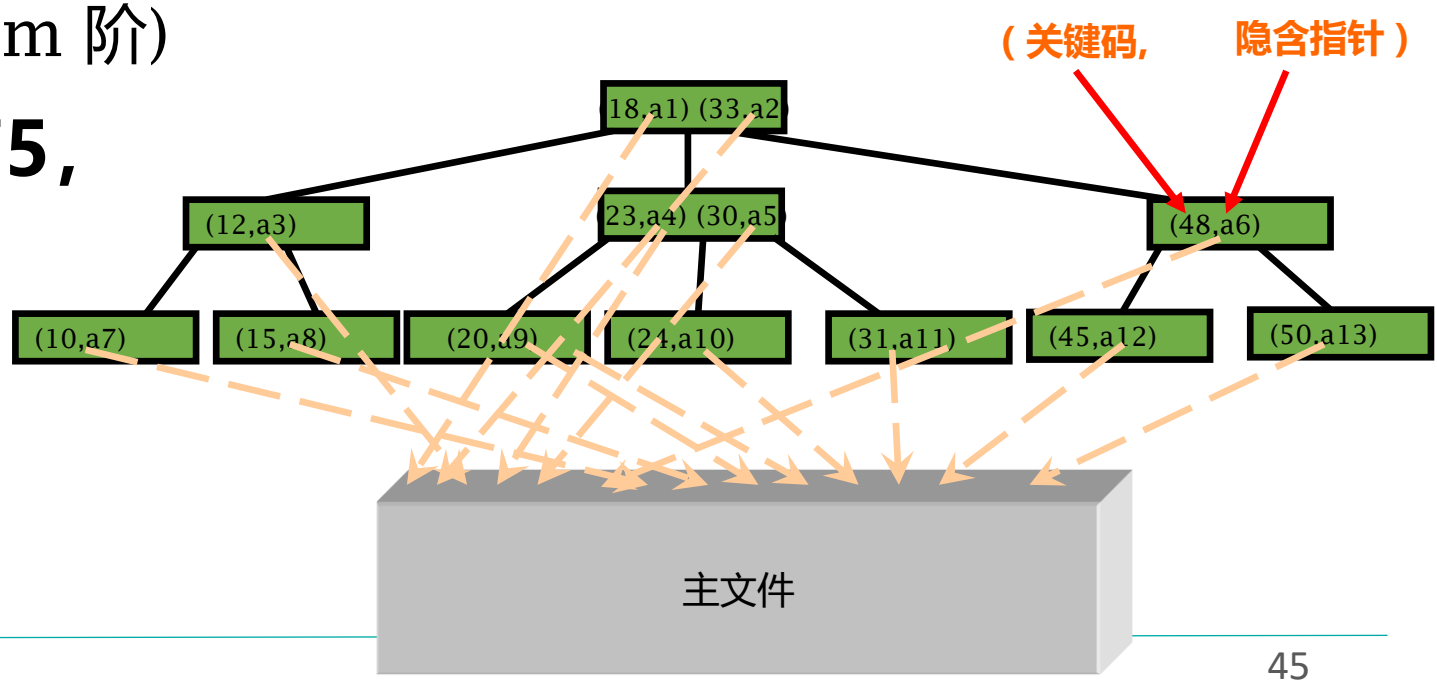
  - **k=4**
  - **一次检索最多4层**

B/B+树的索引效率

- 索引效率取决于树的高度
- 假设一个主文件有N个记录，一个页块可以存m个（关键码，子节点页块地址）二元对
- B+树
  - 假设平均每个结点有0.75m个子结点
    - 每个结点至少半满，平均（1+0.5）/2=0.75
    - 0.75也称为充盈度
  - 因此B+树的高度为 $\lceil \log_{0.75m} N \rceil$

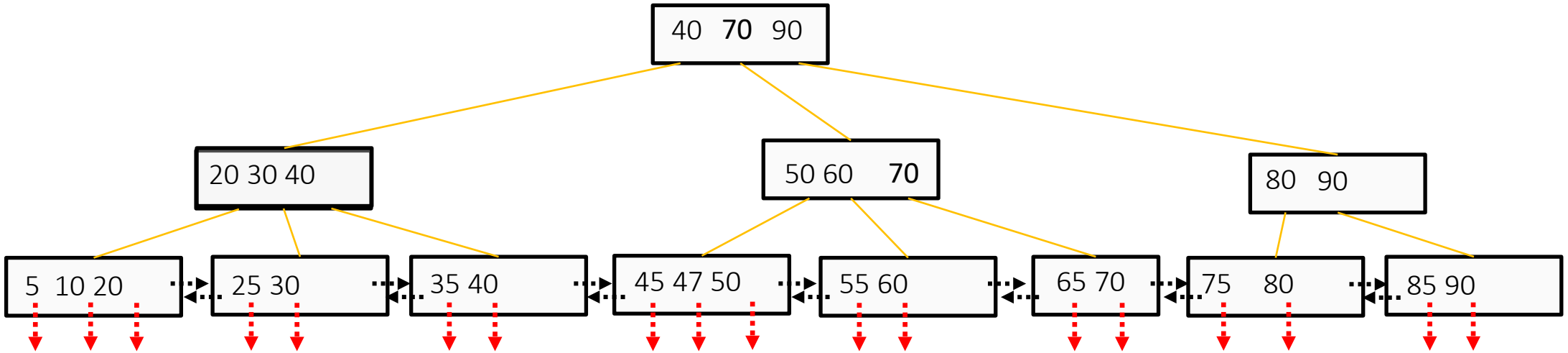
B/B+树的索引效率

- **B树**
  - **相对于B+树，B树的结点需要存储额外的数据指针**
  - **假设关键码所占字节数与指针相同**
    - 可以容纳 B 树的(关键码，隐含指针，子结点页块指针)最多为 2m/3（B 树为 0.67m 阶)
  - **假设B数的充盈度也是0.75，**
    - **B树结点有0.5m个子节点**
  - **B树的高度为** $\lceil \log_{0.5m} N \rceil$



B/B+树的应用

- B+树的索引效率更高，检索层次更少（树较矮）
- 因此，B+树应用更为广泛
  - 数据库系统的主码索引



14.5.4 动态索引和静态索引的性能比较

- **动态索引的优点**
  - 能保持较高的检索效率，较低的动态维护成本
  - 动态地分配和释放存储，可以保持平均75%的存储利用率
- **动态索引存在的问题**
  - 要考虑并行策略
  - 辅助索引维护困难
  - 索引层数多

小结

- 索引
  - 索引是把关键码与其主文件中的数据记录位置关联的过程
- 线性索引
  - 组织成简单的“关键码-地址指针”的序列，按照关键码的顺序进行排序
  - 二级线性索引
- B/B+树
  - 动态索引结构，本质为平衡多叉树，每个结点对应一个磁盘块