MYSQL 数据库索引技术原理初探

管宜尧

github.com/mylxsw

目录

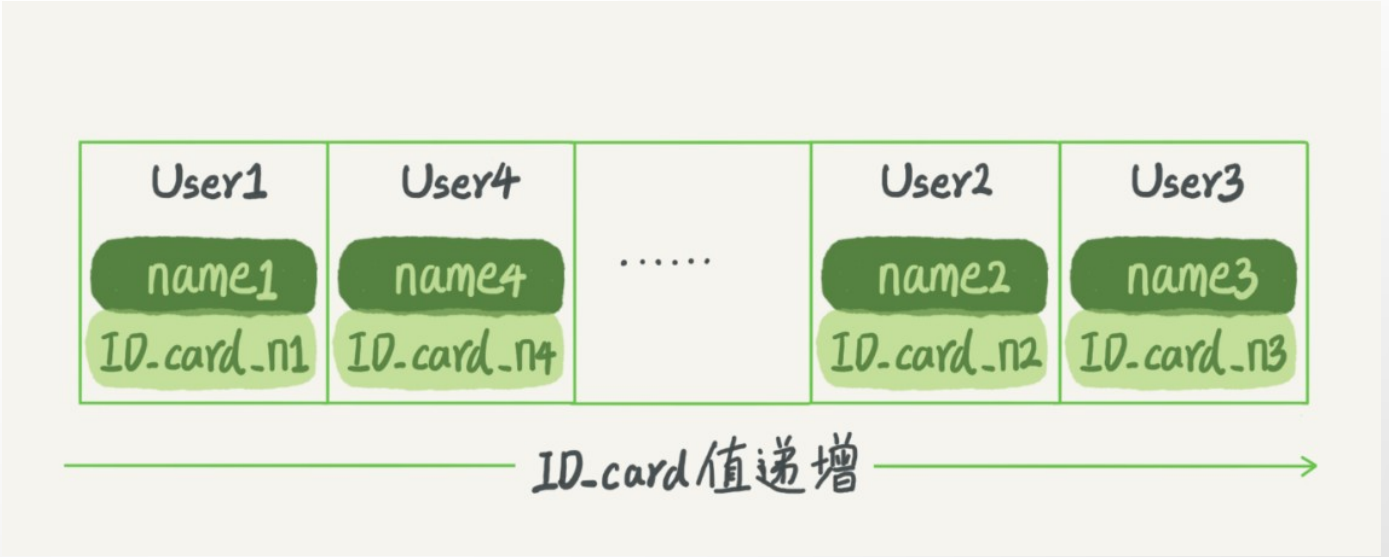
- 什么是索引
- 有序数组
- 哈希表
- B-TREE
- B+TREE

什么是索引

- 一本书 500 页的书，如果没有目录，直接去找某个知识点，可能需要找一会儿，但是借助前面的目录，就可以快速找到对应知识点在书的哪一页。这里的目录就是索引。

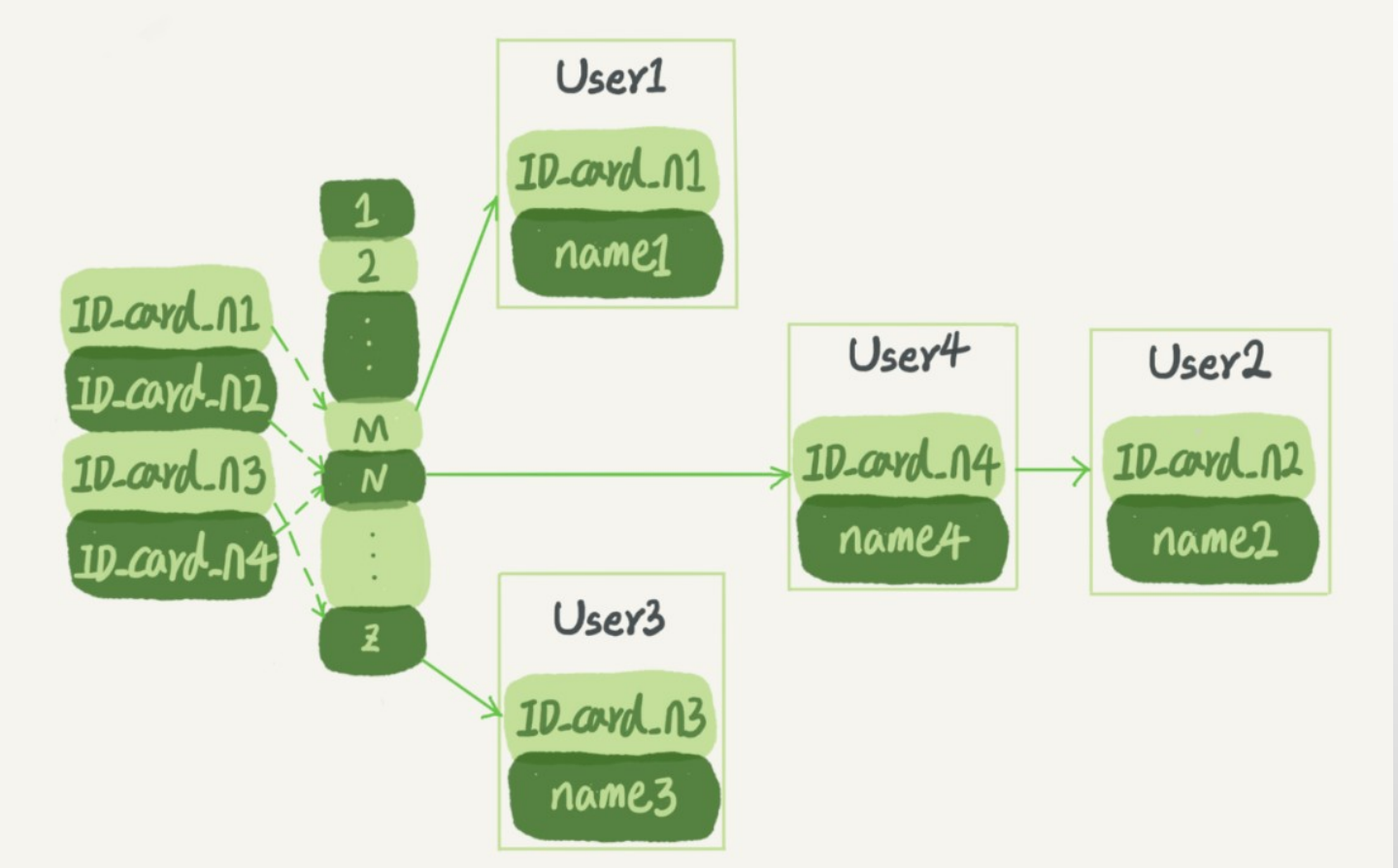
- 所以，为什么会有索引？为了提高数据查询效率。

有序数组



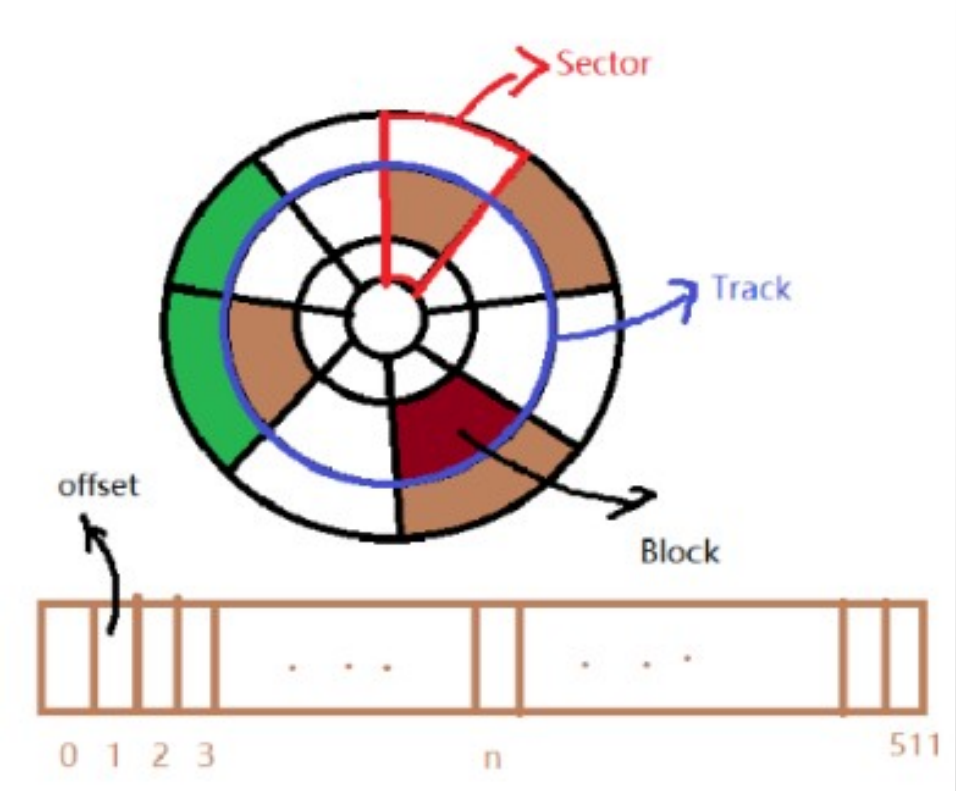
- 使用二分法快速查找到需要的数据（log N）
- 数组元素有序排列
- 处理等值查询和范围查询时，性能优异
- 插入性能很差（N），每次往中间插入一条记录，就必须挪动后面所有的记录，这个成本太高了

哈希表



- KV 形式的数据结构，使用哈希函数计算 K 存储的位置
- 等值查询效率非常高
- Redis/Memcached 都用了哈希表作，Java 中的 HashMap 也是个哈希表
- 范围查询支持性极差，数据无序，必须全表遍历

机械硬盘如何找到一条数据



（Track, Sector） -> Block -> Offset

要访问磁盘上的某个条数据：

- 我们需要通过磁道，扇区来确定数据所在的 Block
- 通过 Offset 就可以定位到磁盘上的任意一个字节

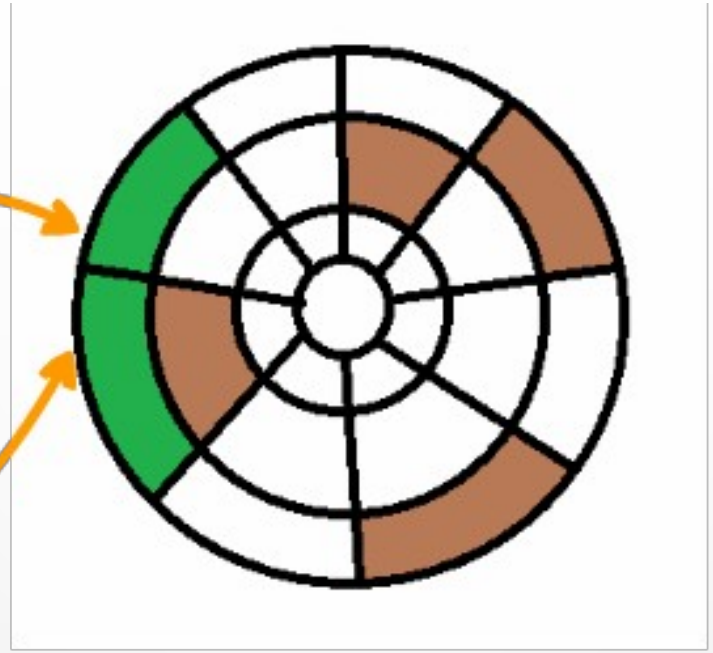
从磁盘上读取数据时，都是以 Block 的形式读取的。

这里我们可以看到，一个 Block 的大小是 512 Bytes

无索引查询

10000条

id	name	dept...
1	张三	...
2	李四	...
3	王五	...
4	赵六	...
5	夏洛克	...
6	李逍遥	...
7	史塔克	...
8	巴里·艾伦	...
9	奥利弗·奎恩	...
10	迪格尔	...
11	莎拉·兰斯	...
12	....	...

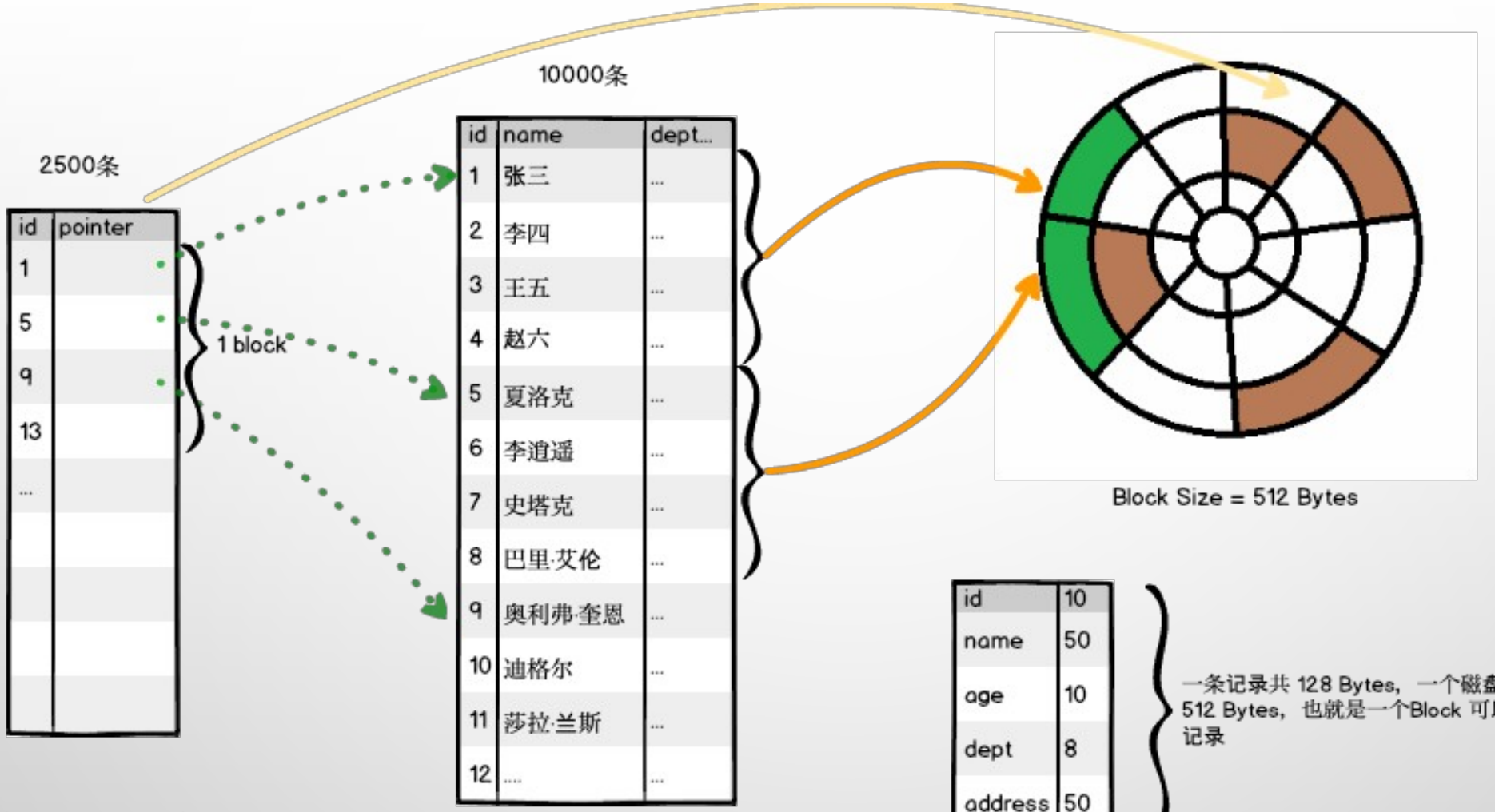


Block Size = 512 Bytes

id	10
name	50
age	10
dept	8
address	50

一条记录共 128 Bytes，一个磁盘 Block 共 512 Bytes，也就是一个Block 可以存储 4 条记录

存储所有记录的 Block 数量等于总记录数量 / 4，也就是 10000 条记录，需要 2500 个 Block



10000 条记录的 Blocks 共需要 2500 个 Index 记录，需要读取
79（索引 Block）+ 1（数据 Block）= 80 个 Blocks

存储所有记录的 Block 数量等于总记录数量 / 4，也就是 10000 条记录，需要 2500 个 Block

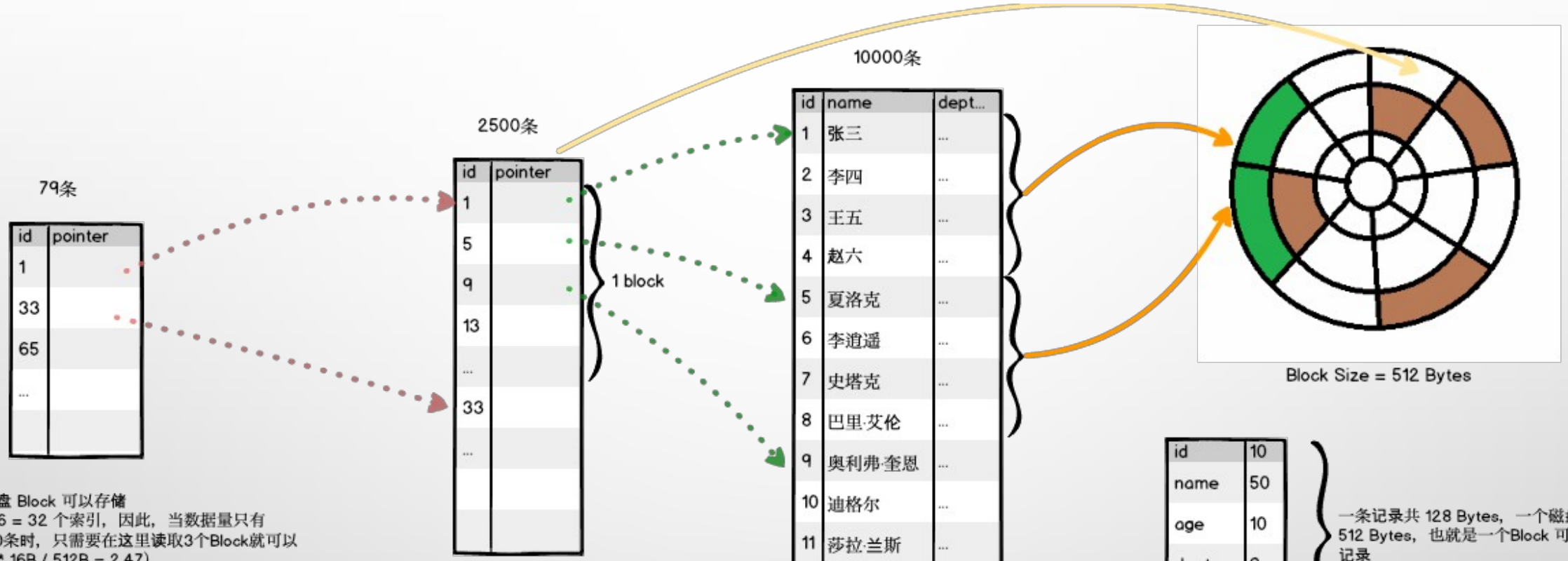
id	10
name	50
age	10
dept	8
address	50

一条记录共 128 Bytes，一个磁盘 Block 共 512 Bytes，也就是一个Block 可以存储 4 条记录

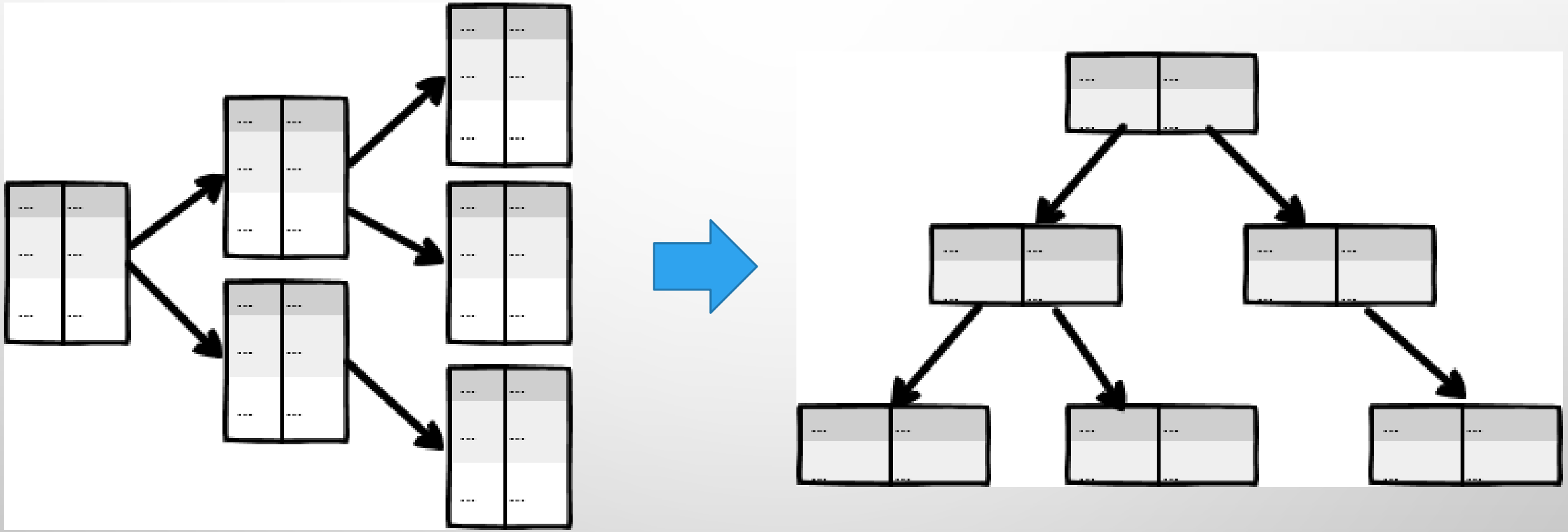
id	10
pointer	6

一条索引记录假设占用空间为 16 Bytes，一个磁盘 Block 可以存储 32 条索引记录，当10000 条记录时，需要读取 313 个Block

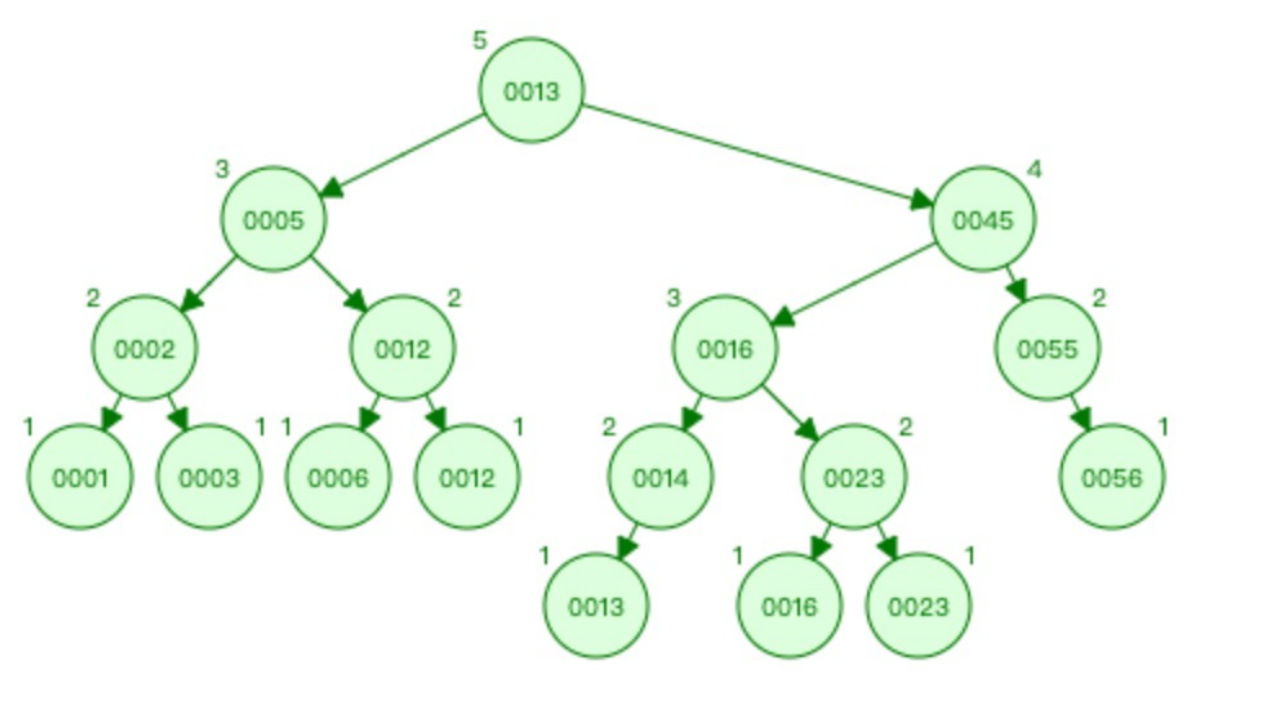
指向数据的索引


79条
id
pointer
1
33
65
...
2500条
id
pointer
1
5
9
13
...
33
...
1 block
10000条
id
name
dept...
1
张三
2
李四
3
王五
4
赵六
5
夏洛克
6
李逍遥
7
史塔克
8
巴里·艾伦
9
奥利弗·奎恩
10
迪格尔
11
莎拉·兰斯
12
....
Block Size = 512 Bytes
1个磁盘 Block 可以存储
512/16 = 32 个索引，因此，当数据量只有
10000条时，只需要在这里读取3个Block就可以
（79 * 16B / 512B = 2.47）
到这里，10000 条数据查询一条，读取的
磁盘 Block 数量就变成了 3+ 1 + 1 = 5 了
10000 条记录的 Blocks 共需要 2500 个 Index 记
录，需要读取
79（索引 Block）+ 1（数据 Block）= 80 个
Blocks
存储所有记录的 Block 数量等于总记录数量 / 4，
也就是 10000 条记录，需要 2500 个 Block
id
10
name
50
age
10
dept
8
address
50
一条记录共 128 Bytes，一个磁盘 Block 共
512 Bytes，也就是一个Block 可以存储 4 条
记录
id
10
pointer
6
一条索引记录假设占用空间为 16 Bytes，一个磁盘 Block 可以存储
32 条索引记录，当10000 条记录时，需要读取 313 个Block

指向索引的索引



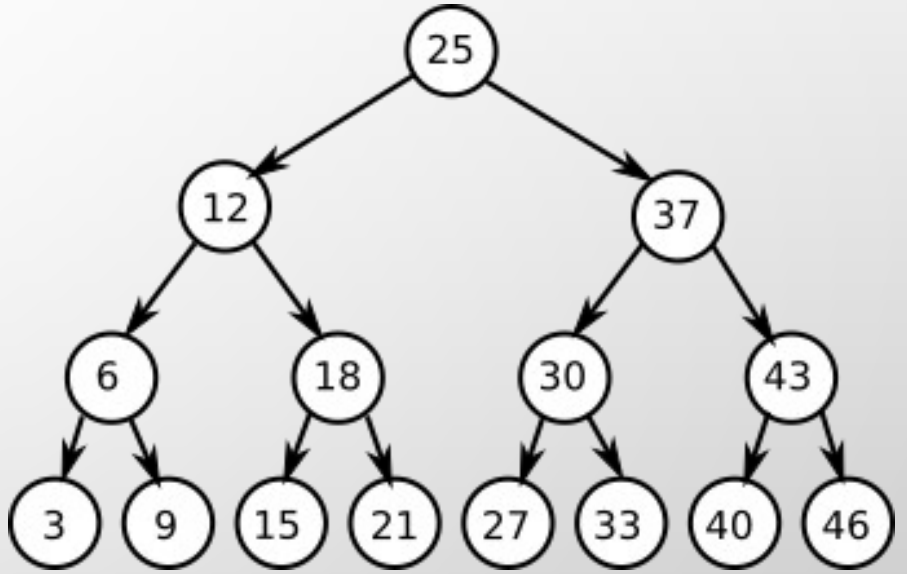
二叉树



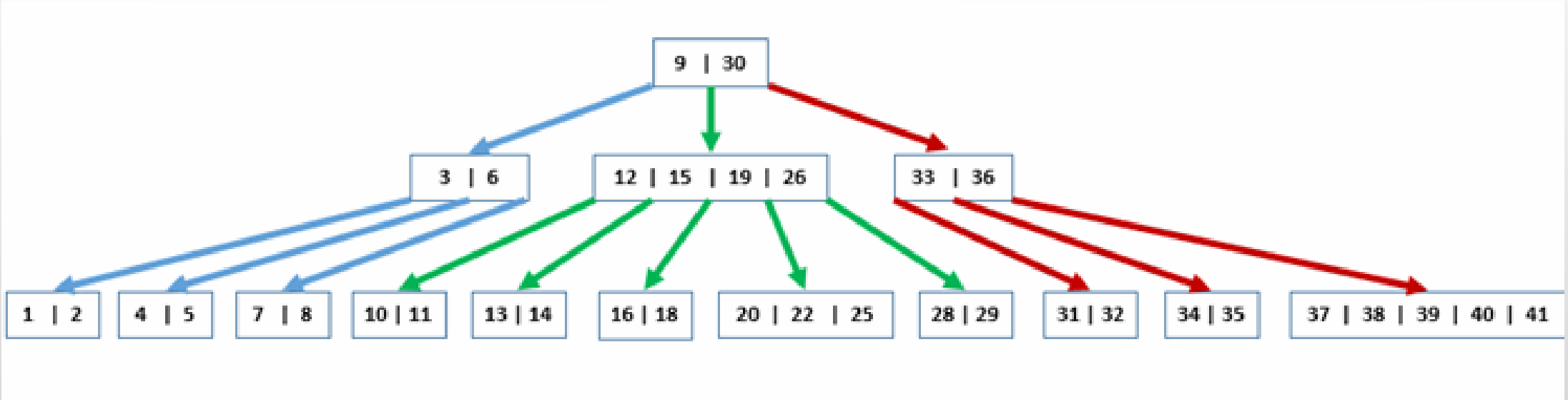
- 每个节点只有两个子节点
- 有序，左子树小于父节点，右子树大于父节点
- 搜索和插入效率都很高（log N）

为什么不用二叉树作为数据库的索引结构？

- 因为我们的数据是存储在磁盘上的，程序运行过程中要使用数据，必须从磁盘把数据加载到内存才行。
- 二叉树随着节点的增多，树的高度也越来越高，对应到磁盘访问上，我们就需要访问更多的数据块。
- 当我们的数据存储在机械硬盘的时候，从磁盘随机读取一个数据块就需要 10 毫秒左右的寻址时间，也就是说，如果我们扫描一个 100 万行的表（树高 20 ， 2^20=1048576），单独访问一行就可能需要 20 个 10 毫秒，可想可知这个查询会有多慢！

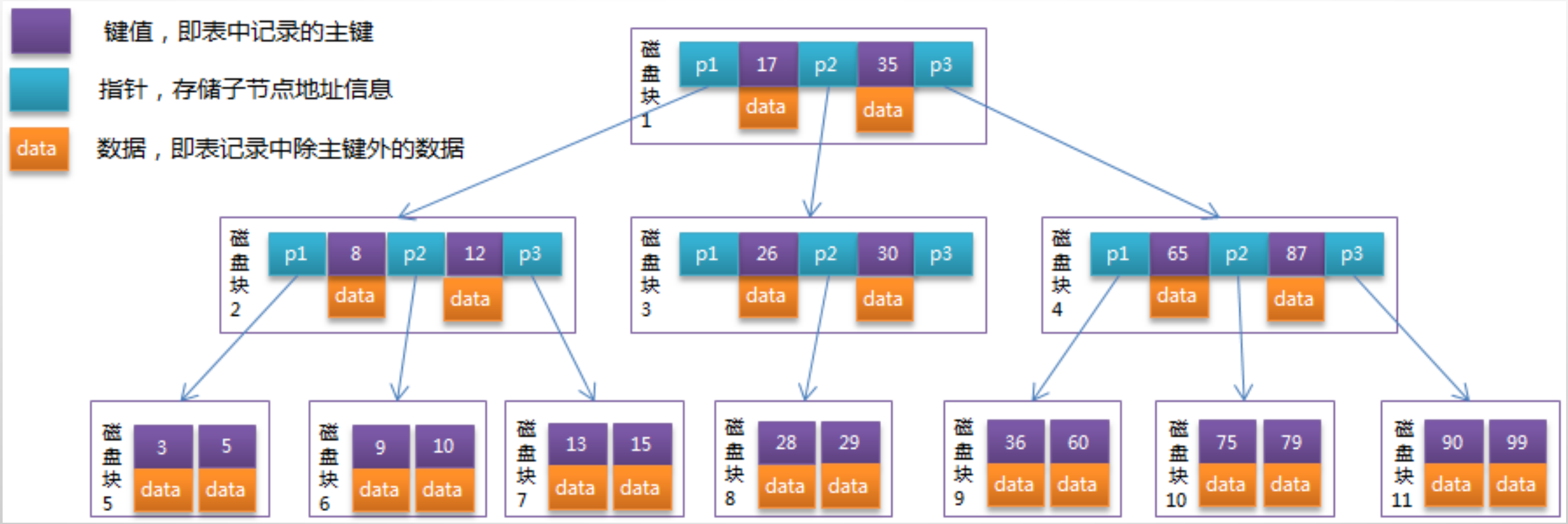


多叉树



- 每个节点都有多个分支，我们把这种树叫做多叉树（N 叉树）
- 每个节点的子节点越多，树的高度就越低

B-TREE



- B-Tree 是一棵 N 叉树，每个节点都会有很多分叉，因此树的高度会很低
- B-Tree 的每个节点都存储数据，很有可能不需要到叶子节点就已经拿到数据了
- B-Tree 很好的利用了“局部性原理”

局部性原理

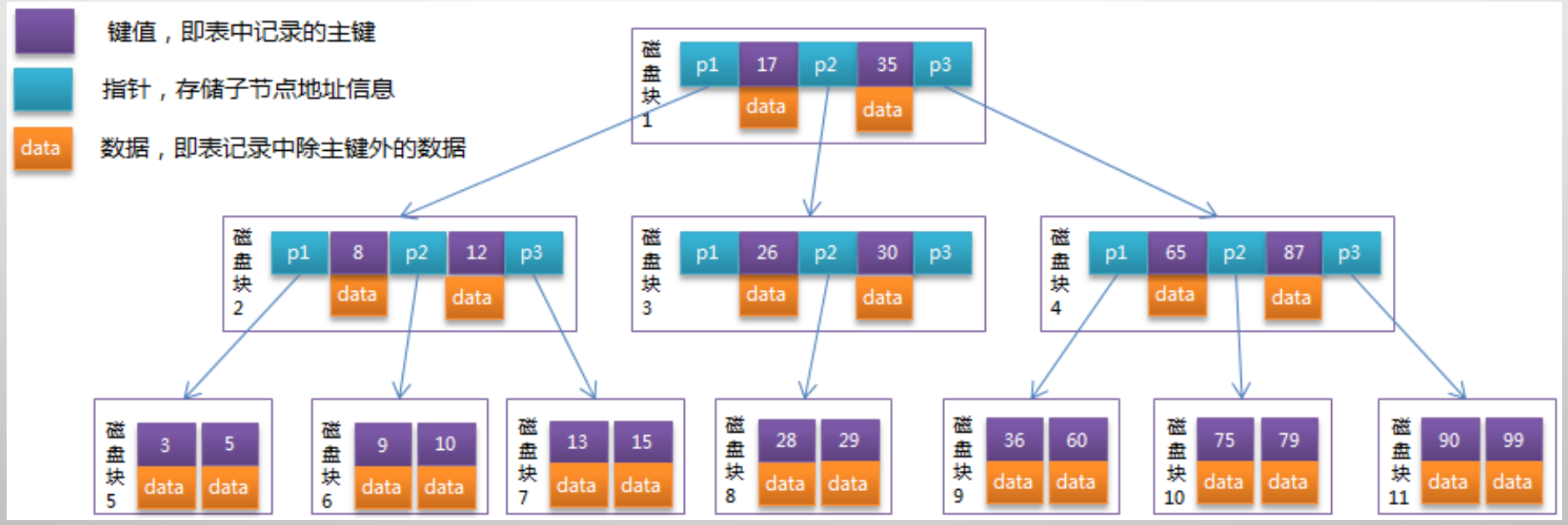
在一段时间内，整个程序的执行仅限于程序的某一部分，相应的，执行所访问的存储空间也局限于某个内存区域。局部性原理分为时间局部性和空间局部性。

- 时间局部性：被引用过一次的存储器位置在未来会被多次引用（通常在循环中）
- 空间局部性：如果一个存储器的位置被引用，那么将来它附近的位置也会被引用

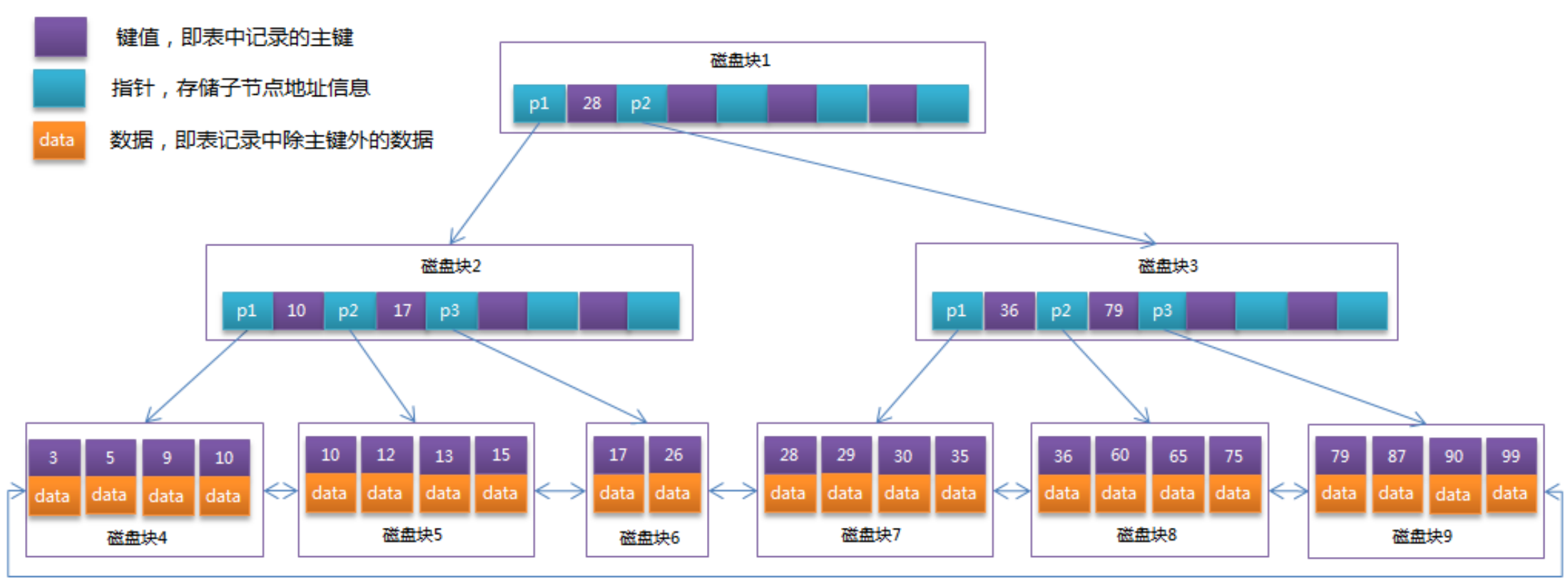
利用局部性原理可以实现磁盘预读，在 InnoDB 中一次是读取一页的数据（16K），也就是说，每次我们实际加载的数据比我们需要的可能会多一些，这些数据可以缓存在内存中，未来我们需要读取的时候，就可以避免磁盘 IO 了。

B-TREE 的缺陷

- 每个节点都存储数据，因此索引会变得很大，每个 BLOCK 能够容纳的索引数就会变少，我们也就需要访问更多次的磁盘
- 对范围查询支持不是很好，需要中序遍历



B+TREE



- 非叶子节点不存储数据，可以实现查询加速
- 范围查询更加优秀，可以顺着叶子节点的链表直接查询出某一个范围内的数据

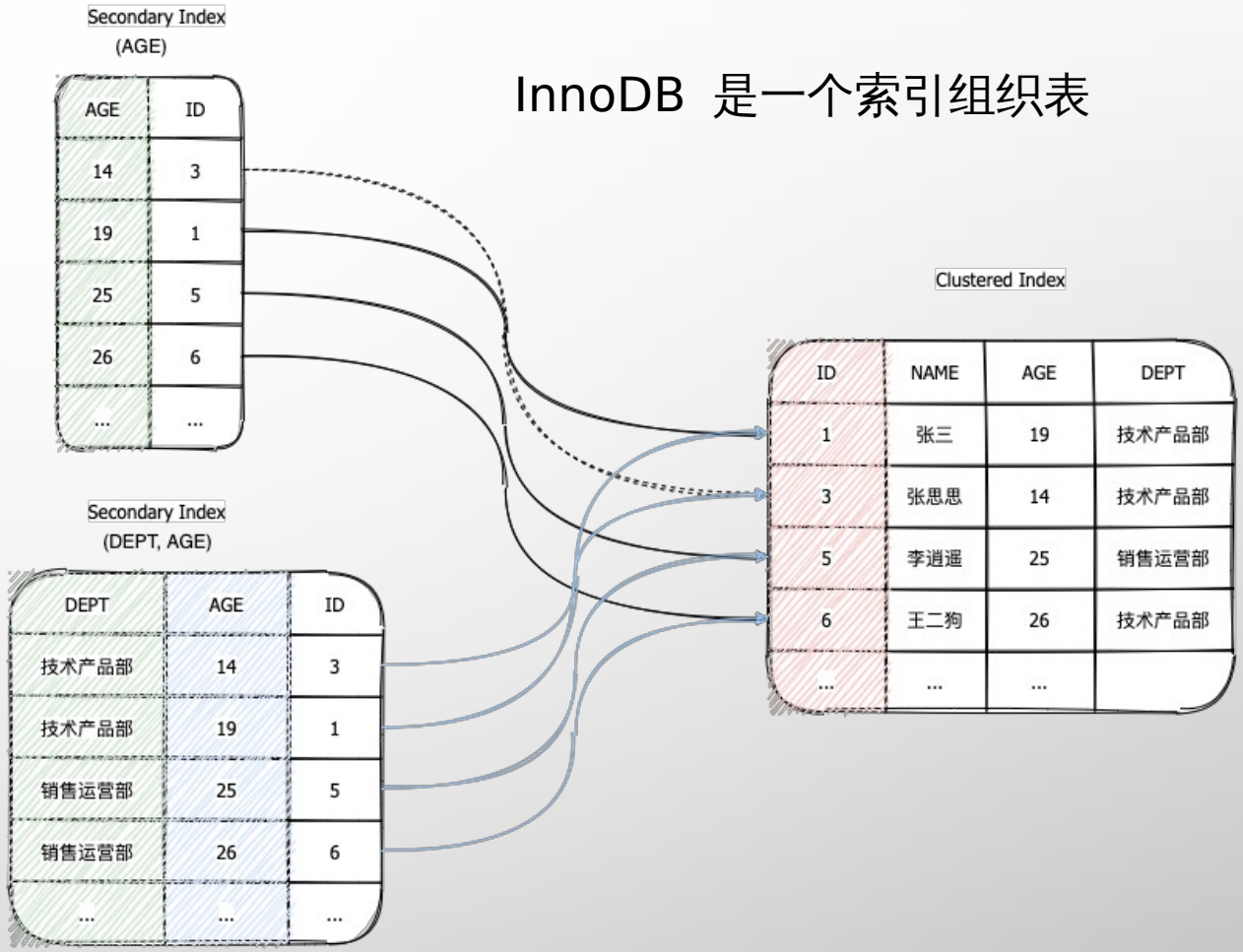
B+TREE

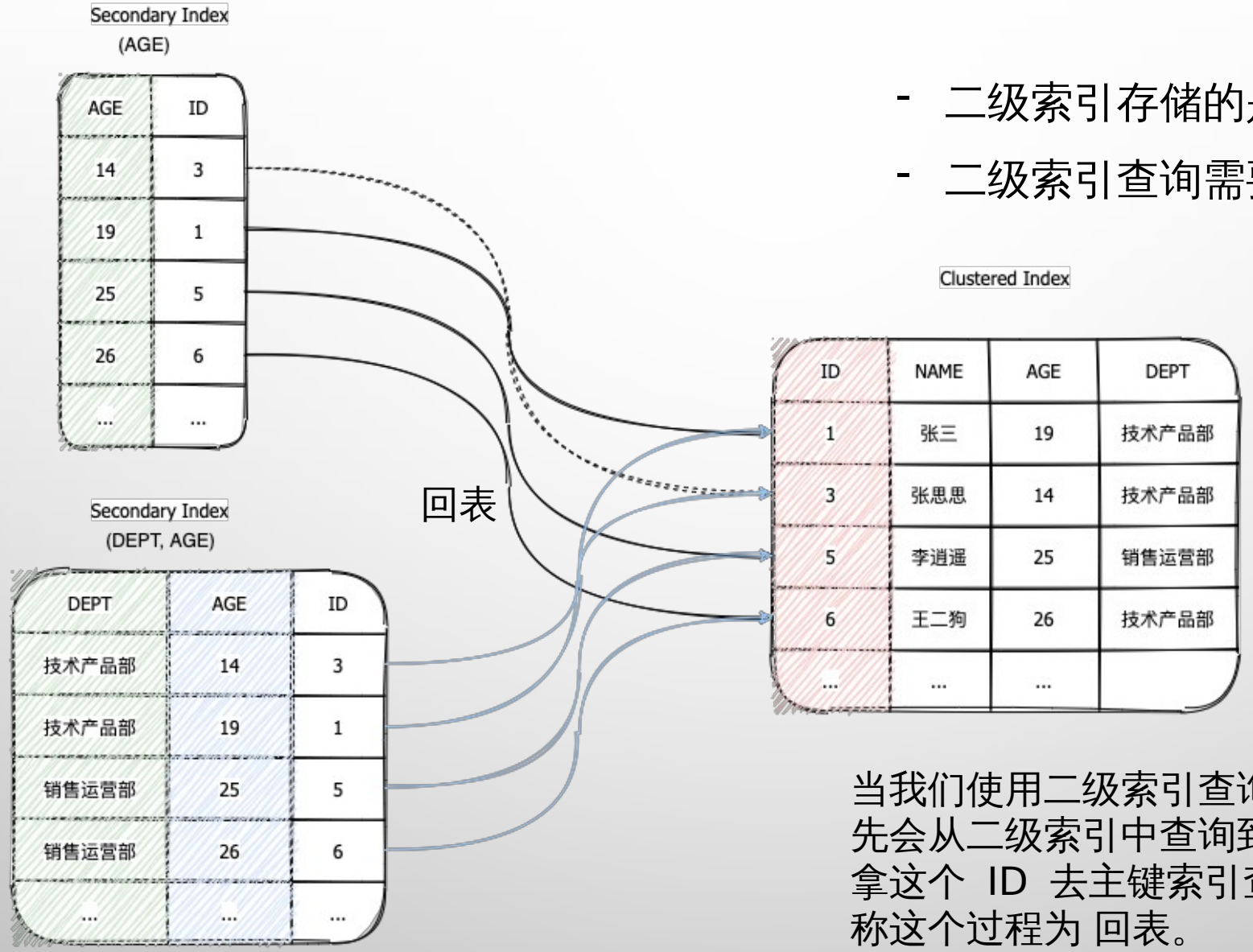
- MySQL 的 InnoDB 存储引擎就是使用 了 B+ 树作为默认的索引算法，B+ 树是一棵 N 叉树，其中 N 的大小取决于索引字段的大小。
- 以整数字段索引为例， N≈1200 ，当树高为 4 的时候，就是 1200 的 3 次方， 17 亿。一个 10 亿行的表上一个整数字段索引，查找一个值最多只需要访问 3 次磁盘（树根一般在内存中）。

INNODB 索引种类

- 主键索引，也成为聚簇索引（Clustered index），在叶子节点存储的是整行数据

- 非主键索引，也成为二级索引（Secondary index），叶子节点存储的是主键的值





- 二级索引存储的是数据的主键
- 二级索引查询需要回表

当我们使用二级索引查询一条数据的时候，首先会从二级索引中查询到这条记录的 ID ，然后拿这个 ID 去主键索引查询真正的数据，我们称这个过程为 回表。

为什么 INNODB 数据表要有自增整型主键

- 因为二级索引存储的是主键的 ID ，因此通常我们会选择 INTEGER 或者 BIGINT 等整型类型作为主键，这样做的目的是可以减少二级索引占用空间的大小。如果用字符串作为主键，可想可知二级索引会有多大！
- 除了上面这个外，通常要求主键一定是要自增的，这样做是为了保证主键的有序，每次插入数据都是追加到 B+ 树，避免页分裂（如果数据页满了，则需要申请新的数据页，然后挪动部分数据过去，这个过程叫做 页分裂）的产生，提高数据写入性能。

优化索引的小技巧

- 索引应该尽可能小，这样一次磁盘读取可以返回尽可能多的索引数据，在查询数据时就可以减少磁盘 IO
- 大表查询尽可能的使用索引，不使用索引就会造成全表扫描，想想一个查询，需要遍历几百万数据，读取成千上百次磁盘会有多慢
- 如果可能，尽量使用主键索引进行查询，使用主键索引可以直接触达数据，不需要执行回表，减少磁盘 IO
- 如果索引中包含了我们要查询的所有字段，那就不需要在进行回表，可以减少磁盘 IO ，显著提升查询性能，我们把这种查询数据都在索引里面的情况叫做 **覆盖索引**

引用资料

- https://blog.csdn.net/yin767833376/article/details/81511377
- https://time.geekbang.org/column/article/69236
- https://www.codeproject.com/articles/1158559/b-tree-another-implementation-by-java

THANKS