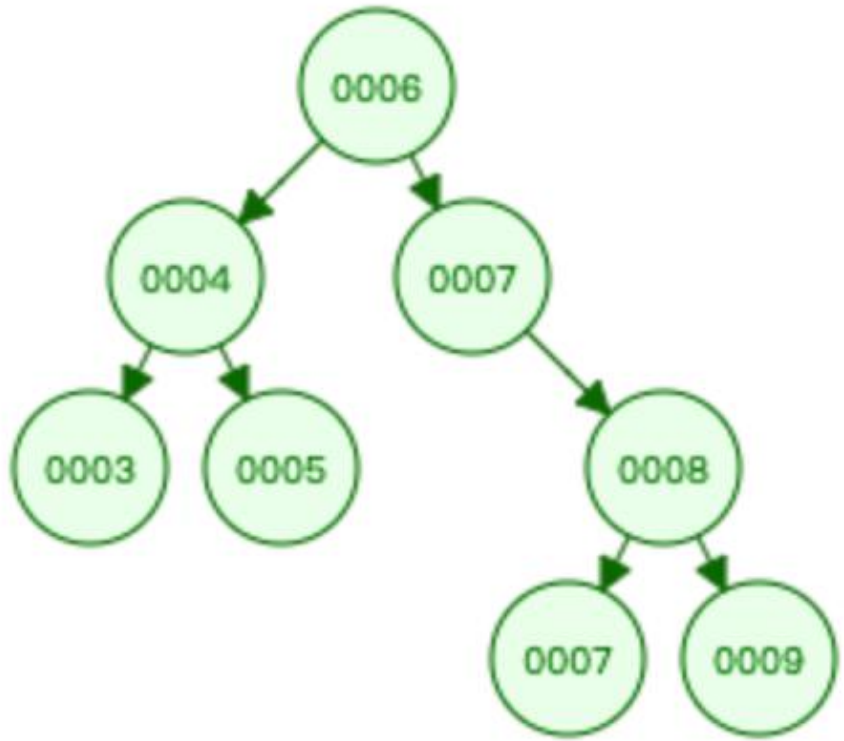
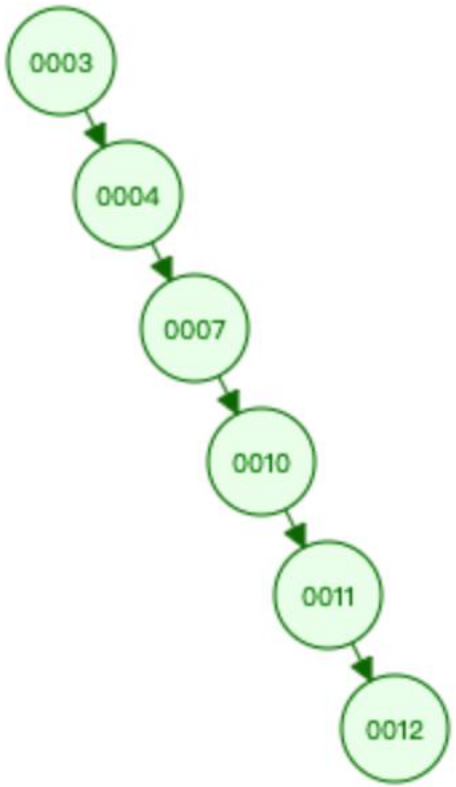
从数组说起

线性查找复杂度 $O(n)$
二分查找复杂度 $O(\log_2 n)$
快速排序复杂度 $O(n\log_2 n)$

二分查找树

时间复杂度：$\log_2(n)$

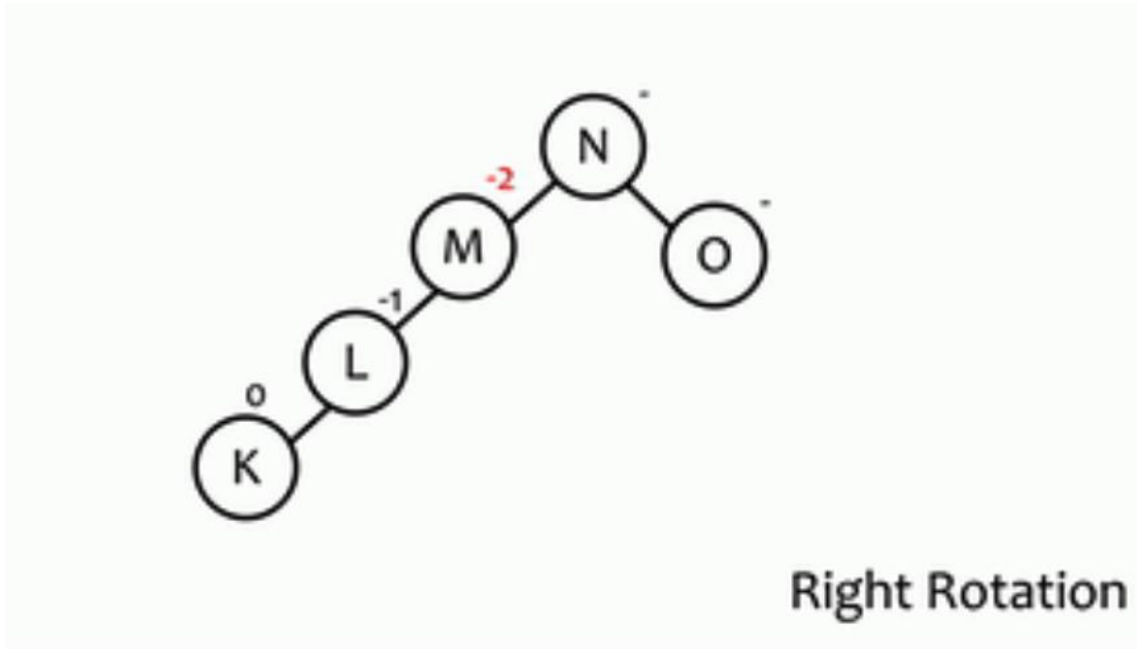

0006
0004
0007
0003
0005
0008
0007
0009

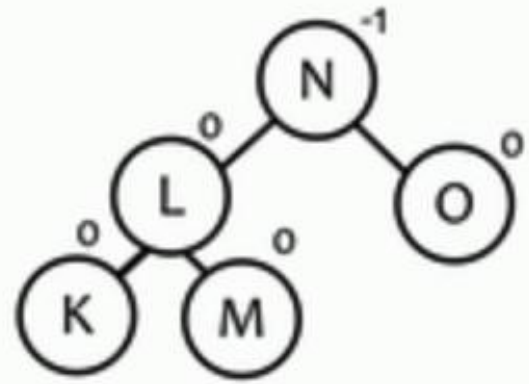


需要平衡

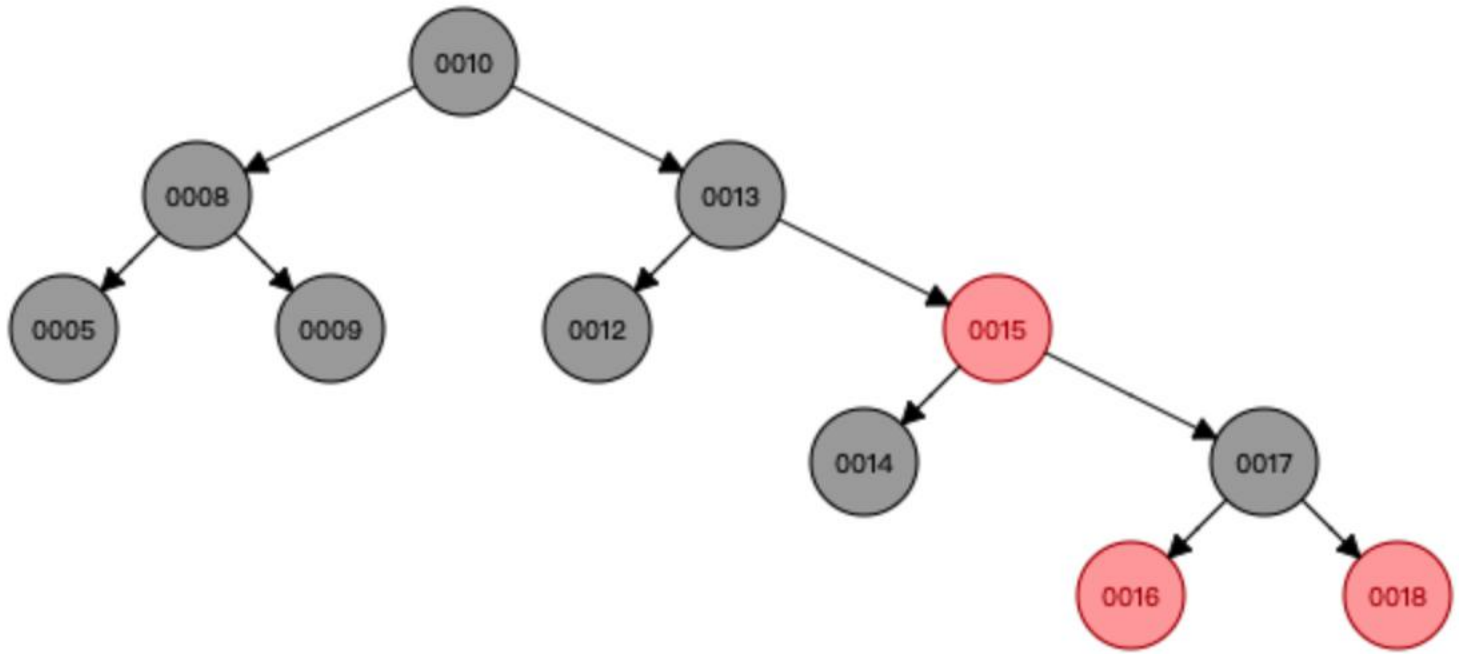
AVL树 vs. RB树

严格平衡 与 大致平衡





Right Rotation


0010
0008
0013
0005
0009
0012
0015
0014
0017
0016
0018

节点数量	树类型	查询时间	插入时间	删除时间	插入旋转	删除旋转	树高度
1,000,000	avl	197	256	61	704626	704619	23
	rbtree	227	251	58	588194	681424	27
	linux rbtree	221	244	51	588194	681424	27
10,000,000	avl	2048	2654	629	7053316	7053496	27
	rbtree	2252	2513	547	5887217	6815235	33
	linux rbtree	2235	2510	520	5887217	6815235	33

从内存到磁盘

硬盘的特点：

1、速度慢：$O(N_{磁盘} >> O(n_{内存})$

2、以扇区块(4KB)为单位读取

3、顺序读写速度>>随机读写速度

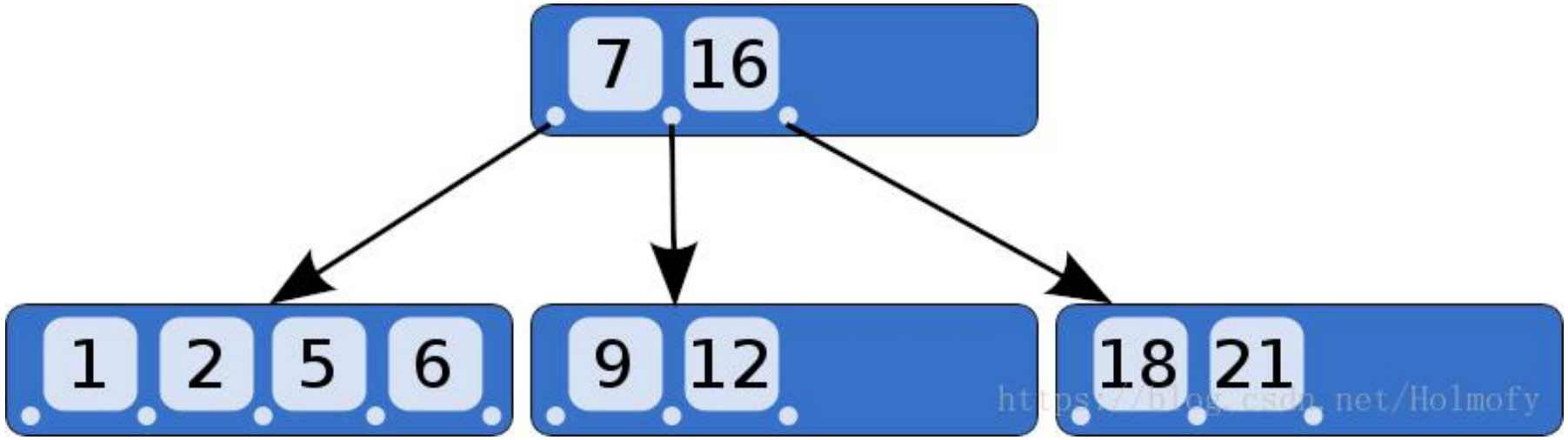
Latency Comparison Numbers (~2012)

• L1 cache reference	0.5 ns		
• Branch mispredict	5 ns		
• L2 cache reference	7 ns		14x L1 cache
• Mutex lock/unlock	25 ns		
• Main memory reference	100 ns		20x L2 cache, 200x L1 cache
• Compress 1K bytes with Zippy	3,000 ns	3 us	
• Send 1K bytes over 1 Gbps network	10,000 ns	10 us	
• Read 4K randomly from SSD*	150,000 ns	150 us	~1GB/sec SSD
• Read 1 MB sequentially from memory	250,000 ns	250 us	
• Round trip within same datacenter	500,000 ns	500 us	
• Read 1 MB sequentially from SSD*	1,000,000 ns	1,000 us	1 ms ~1GB/sec SSD, 4X memory
• Disk seek	10,000,000 ns	10,000 us	10 ms 20x datacenter roundtrip
• Read 1 MB sequentially from disk	20,000,000 ns	20,000 us	20 ms 80x memory, 20X SSD
• Send packet CA->Netherlands->CA	150,000,000 ns	150,000 us	150 ms

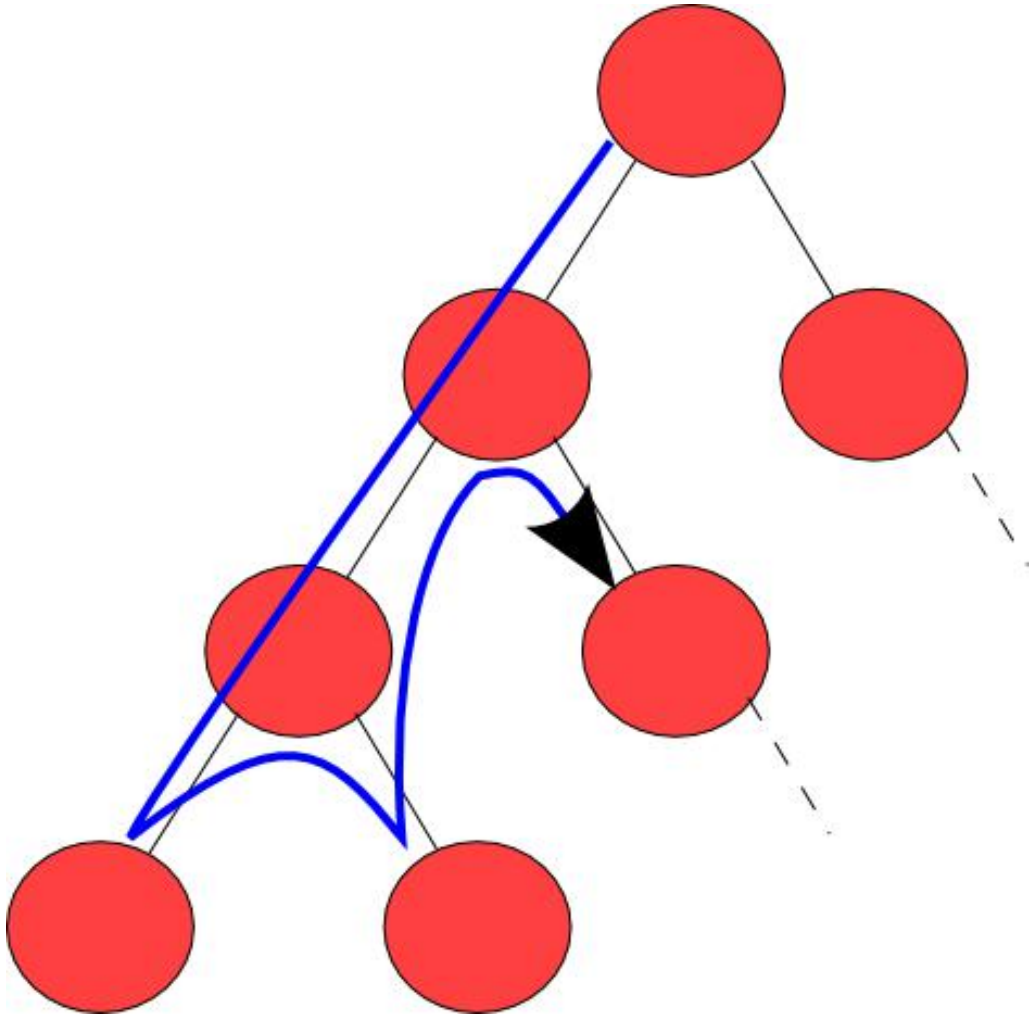
代码优化的再好

一条SQL写的不好一切都白干

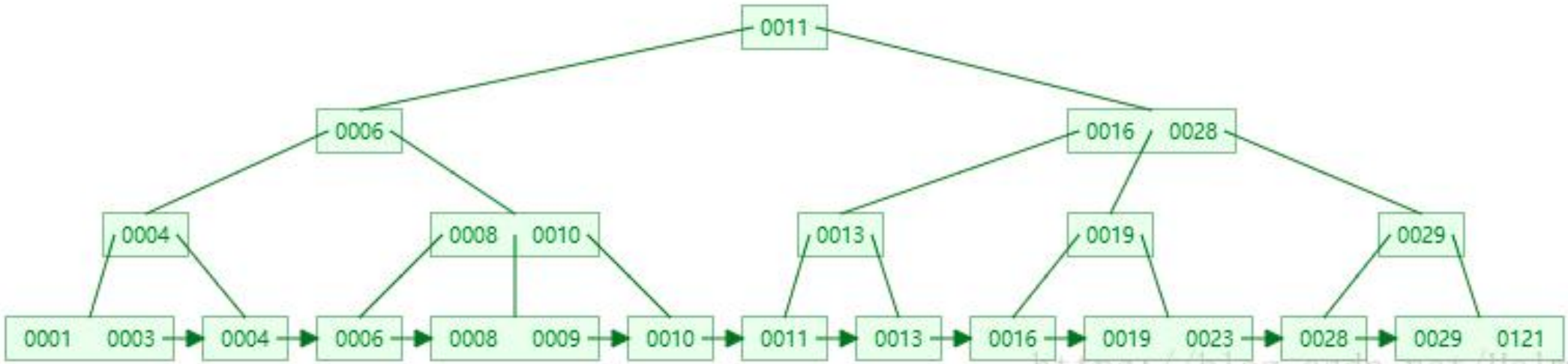
让节点变大以适应磁盘



B树节点回溯问题



把数据链起来



B+树的优点：一切都是为了减少I/O

1、内部的非叶子节点只存储Key，不存Value数据。
完整的数据都保存在叶子节点中。

目的：让非叶子节点可以索引更多数据。

以MySQL为例，InnoDB的页大小为16KB。主键使用8字节的bigint，下级指针也用8字节(4字节32bit只能寻址4GB数据)。相当于：

一页能存1000条左右的数据(不考虑InnoDB预留的1/16空间)。

两层B+树就能索引1000*1000条数据（占用空间16MB）

三层B+树就能索引1000*1000*1000条数据（占用空间16GB）

内两层索引只有16.016MB，完全可以缓存在内存里。

B+树的优点：一切都是为了减少I/O

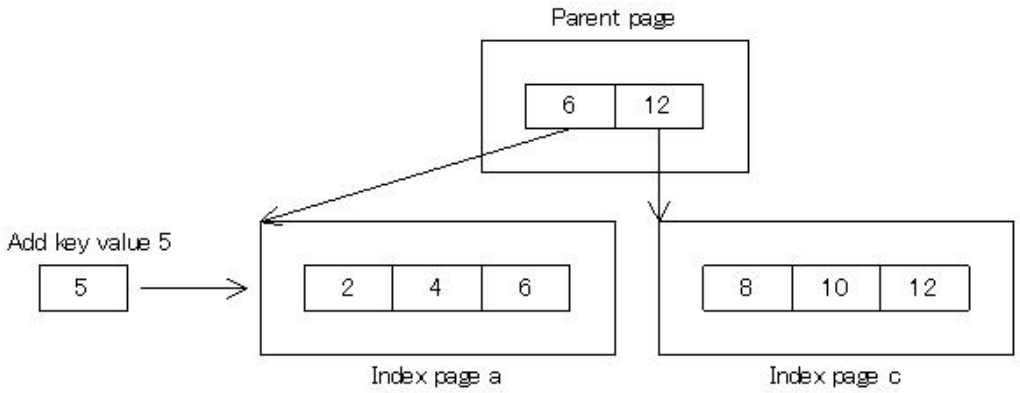
2、叶子节点之间用指针串联起来

范围查询不需要回溯节点，减少磁盘IO次数。

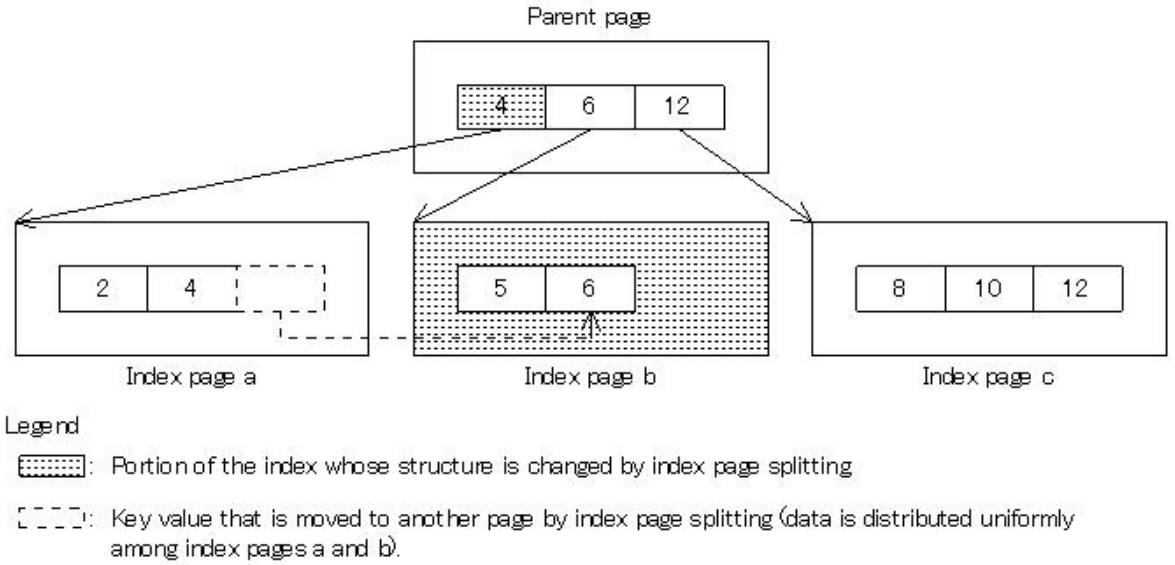
B+树还不够完美

页分裂

1. B-tree structure of index before index page splitting



2. Occurrence of index page splitting



Legend

: Portion of the index whose structure is changed by index page splitting

: Key value that is moved to another page by index page splitting (data is distributed uniformly among index pages a and b).

页分裂需要在父节点新增索引数据。导致原来只需要一个页的I/O操作升级为3个页的I/O操作。

正因为页分裂导致性能差，且页空间浪费严重，所以通常数据库建模时，为了避免随机插入，不推荐用UUID这样的字段作为主键，而应当使用自增的ID为主键。

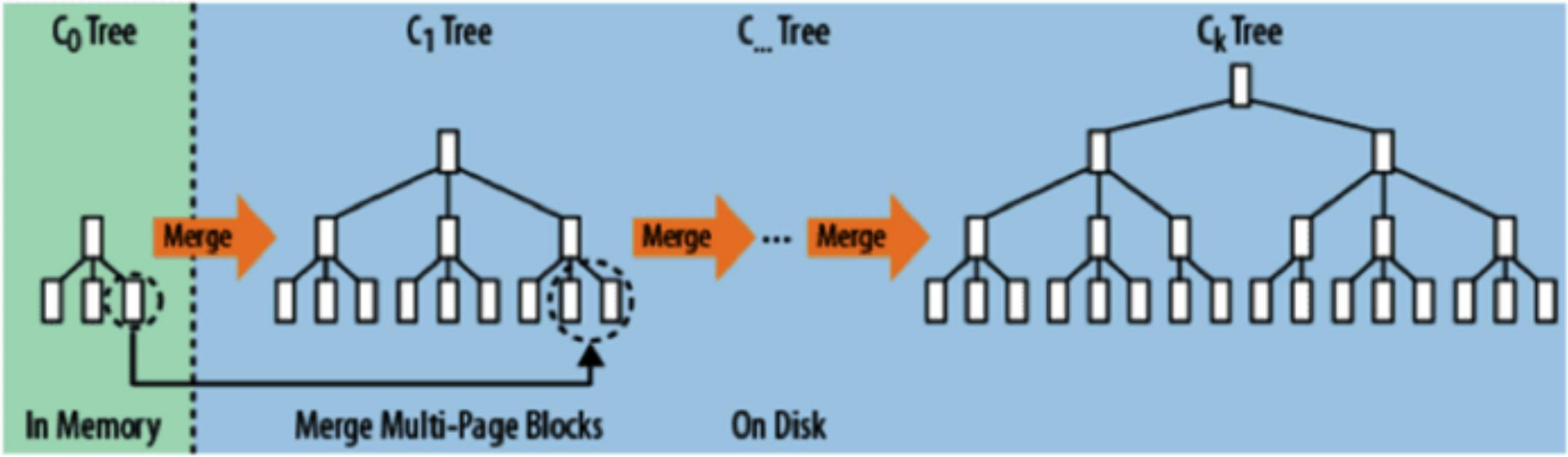
不适合超过百万级的大数据存储

问题:
当数据量超过百万级后，B+树层级过深，I/O次数增多，性能下降。

解决方案:

1、 分库分表：根据表中的某个业务字段，将数据分散到多个数据库。
目前分库分表有三种实现方案:
1）应用端做路由。特点：灵活性高；业务侵入性略强；无法join、sort。
2）中间件代理。shardingsphere等分库中间件提供了hash、range等多种分片路由策略；性能较差。
3）数据库端实现分片。MySQL 5.7开始提供partition的功能。

大数据存储结构：LSM-Tree



写：写入内存里的C_0树，为了防止数据库崩溃保证数据持久性同时也会将数据写入WAL。对于删除，同样也会添加一条墓碑记录，当C_0树达到一定大小，会写入磁盘。当C_1树达到阈值，按照特定的算法负责将小树Merge成大树，同时将修改或删除的数据进行合并。

读：从C_0树开始读，依次从小树读到大树，直到找到数据或者扫描完所有树。

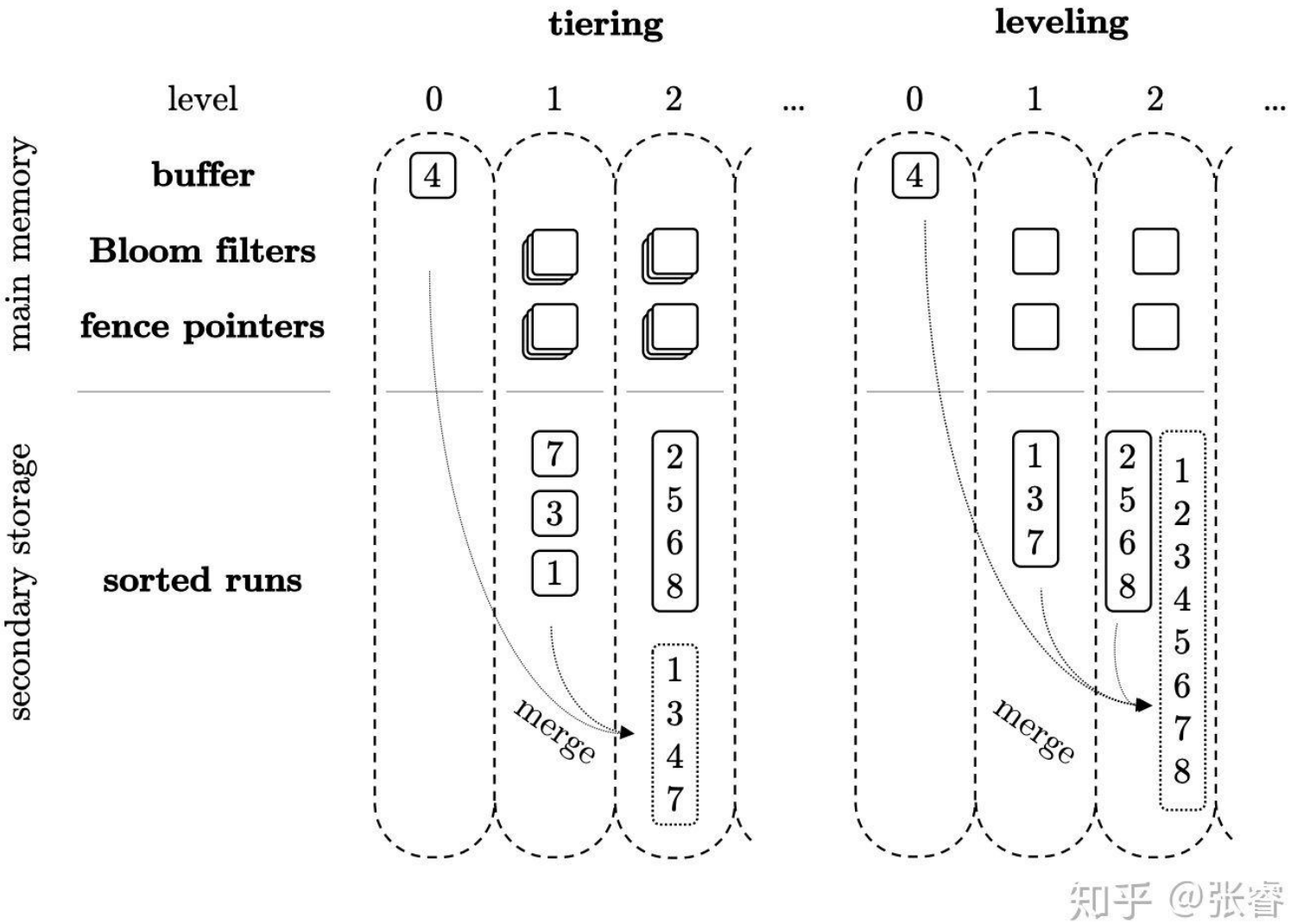
LSM-Tree的缺点

- 1、**读放大**：原来只要读一棵树，现在可能需要读多棵树，而且当数据不存在的时候会扫描所有的树。
  解决方案：使用BloomFilter对不存在的数据进行预判。

- 2、**写放大**：原来一条记录只需要一次I/O写操作，现在因为有Merge操作可能会演化成多次I/O写操作。

- 3、**空间放大**：数据库里可能会有未合并的历史数据占用空间。
  相比于B+树随机写导致的磁盘碎片，这个问题倒不是很严重。

  解决方案：优化Merge算法，这也是目前数据库研究的主要课题。



Merge算法参考波士顿大学研究：https://disc-projects.bu.edu/compactionary/research.html

LSM-Tree应用

- Google BigTable(2006)
- Google LevelDB(2011)
- Apache HBase(2008)
- Facebook RockDB(2013)
- MySQL MyRocks
- MongoDB WireTiger(2008,2014)
- Elasticsearch(2010)
- Yandex Clickhouse(2016)

索引的应用

表 4 student_info 表的数据

学号	姓名	性别	出生日期	家族住址
0001	张青平	男	2000-10-01	衡阳市东风路 77 号
0002	刘东阳	男	1998-12-09	东阳市八一北路 33 号
0003	马晓夏	女	1995-05-12	长岭市五一路 763 号
0004	钱忠理	男	1994-09-23	滨海市洞庭大道 279 号
0005	孙海洋	男	1995-04-03	长岛市解放路 27 号
0006	郭小斌	男	1997-11-10	南山市红旗路 113 号
0007	肖月玲	女	1996-12-07	东方市南京路 11 号
0008	张玲珑	女	1997-12-24	滨江市新建路 97 号

表 5 curriculum 表的数据

课程编号	课程名称	学分
0001	计算机应用基础	2
0002	C 语言程序设计	2
0003	数据库原理及应用	2
0004	英语	4
0005	高等数学	4

1、单列索引

```
select * from t where name=?
idx_name(name)
select * from t where birthday=?
idx_birthday(birthday)
```

2、联合索引

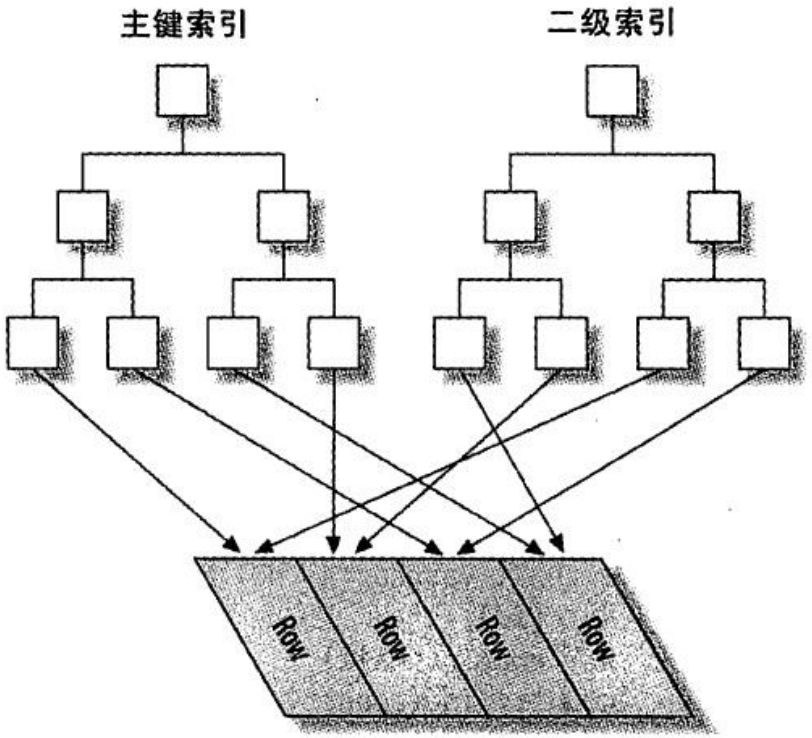
```
select * from t where birthday=? and name=?
idx_birthday_name(birthday, name)
```

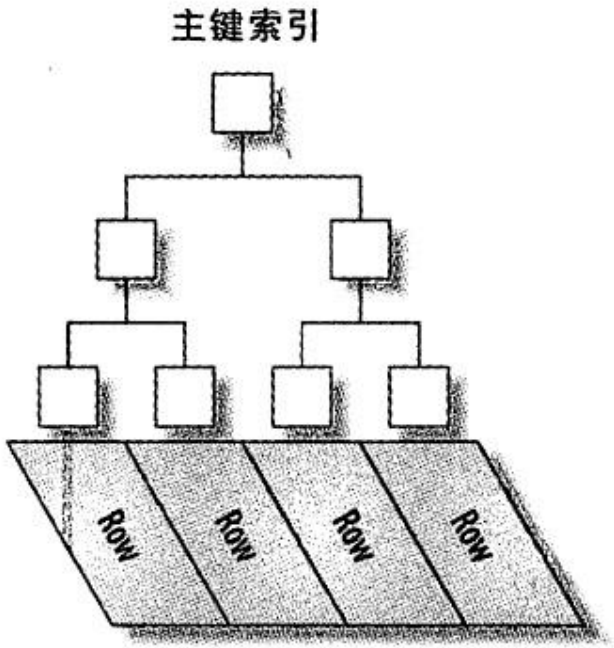
3、前缀匹配

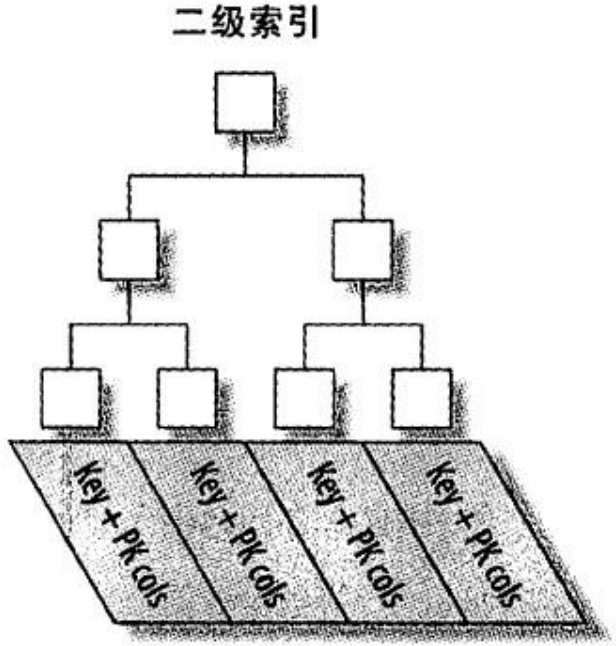
```
select * from t where birthday>? and name=?
idx_name_birthday(name, birthday)

select * from t
where birthday=? and name like '马%'
idx_birthday_name(birthday, name)
```

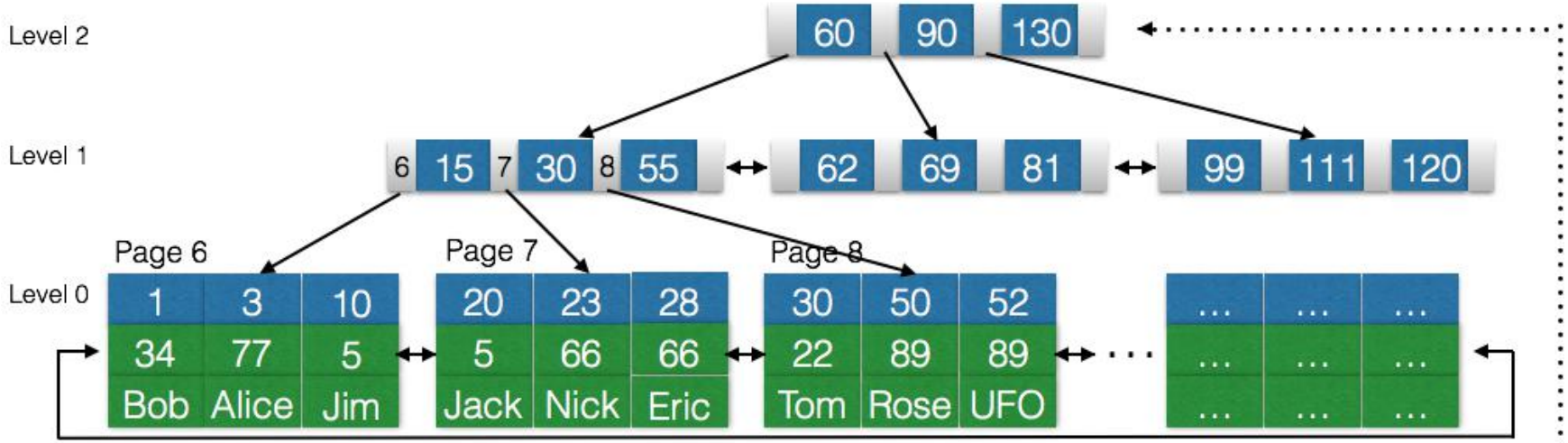
聚簇与非聚簇


主键索引
二级索引
Row
Row
Row
Row

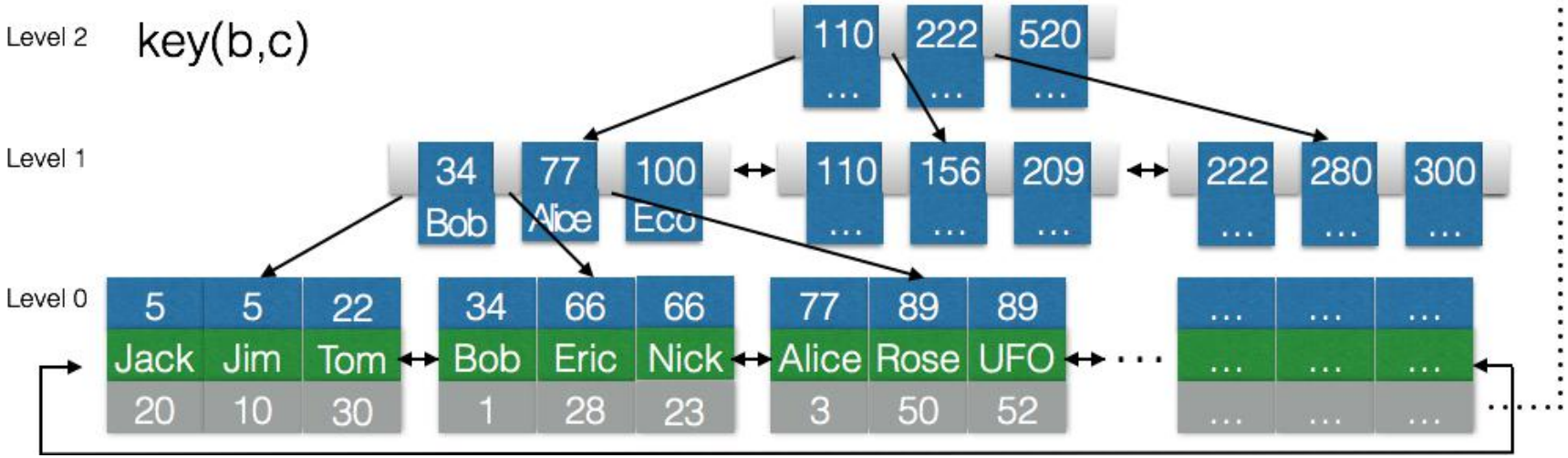

主键索引
Row
Row
Row
Row


二级索引
Key + PK cols
Key + PK cols
Key + PK cols
Key + PK cols

主键索引


Level 2
60
90
130
Level 1
6
15
7
30
8
55
62
69
81
99
111
120
Page 6
Page 7
Page 8
Level 0
1
3
10
34
77
5
Bob
Alice
Jim
20
23
28
5
66
66
Jack
Nick
Eric
30
50
52
22
89
89
Tom
Rose
UFO

二级索引


Level 2
key(b,c)
110
222
520
Level 1
34
Bob
77
Alice
100
Eco
110
156
209
222
280
300
Level 0
5
5
22
Jack
Jim
Tom
20
10
30
34
66
66
Bob
Eric
Nick
1
28
23
77
89
89
Alice
Rose
UFO
3
50
52

索引覆盖

表 4 student_info 表的数据

学号	姓名	性别	出生日期	家族住址
0001	张青平	男	2000-10-01	衡阳市东风路 77 号
0002	刘东阳	男	1998-12-09	东阳市八一北路 33 号
0003	马晓夏	女	1995-05-12	长岭市五一路 763 号
0004	钱忠理	男	1994-09-23	滨海市洞庭大道 279 号
0005	孙海洋	男	1995-04-03	长岛市解放路 27 号
0006	郭小斌	男	1997-11-10	南山市红旗路 113 号
0007	肖月玲	女	1996-12-07	东方市南京路 11 号
0008	张玲珑	女	1997-12-24	滨江市新建路 97 号

表 5 curriculum 表的数据

课程编号	课程名称	学分
0001	计算机应用基础	2
0002	C 语言程序设计	2
0003	数据库原理及应用	2
0004	英语	4
0005	高等数学	4


```
select name from t
group by name
order by count(*) desc;
```

idx_name(name)

```
select name, birthday from t
order birthday limit 10;
```

idx_birthday_name(birthday, name)

排序与分组

sort file | uniq -c | sort -nr

c1	c2	c3	c4
12	25	6	23
18	11	11	17
4	17	3	1
16	26	8	11
7	24	8	26
3	6	12	10
7	12	5	11
4	8	3	12
8	3	9	19
11	24	3	18
4	23	9	15
2	14	23	5
23	23	1	19
13	15	19	20
13	7	6	19
24	5	10	11

sort

group by c3

c1	c2	c3	c4
23	23	1	19
4	17	3	1
4	8	3	12
11	24	3	18
7	12	5	11
12	25	6	23
13	7	6	19
16	26	8	11
7	24	8	26
8	3	9	19
4	23	9	15
24	5	10	11
18	11	11	17
3	6	12	10
13	15	19	20
2	14	23	5

group

c1	c2	c3	c4
23	23	1	19

c1	c2	c3	c4
4	17	3	1
4	8	3	12
11	24	3	18

c1	c2	c3	c4
7	12	5	11

c1	c2	c3	c4
12	25	6	23
13	7	6	19

c1	c2	c3	c4
16	26	8	11
7	24	8	26

c1	c2	c3	c4
8	3	9	19

c1	c2	c3	c4
24	5	10	11

c1	c2	c3	c4
18	11	11	17

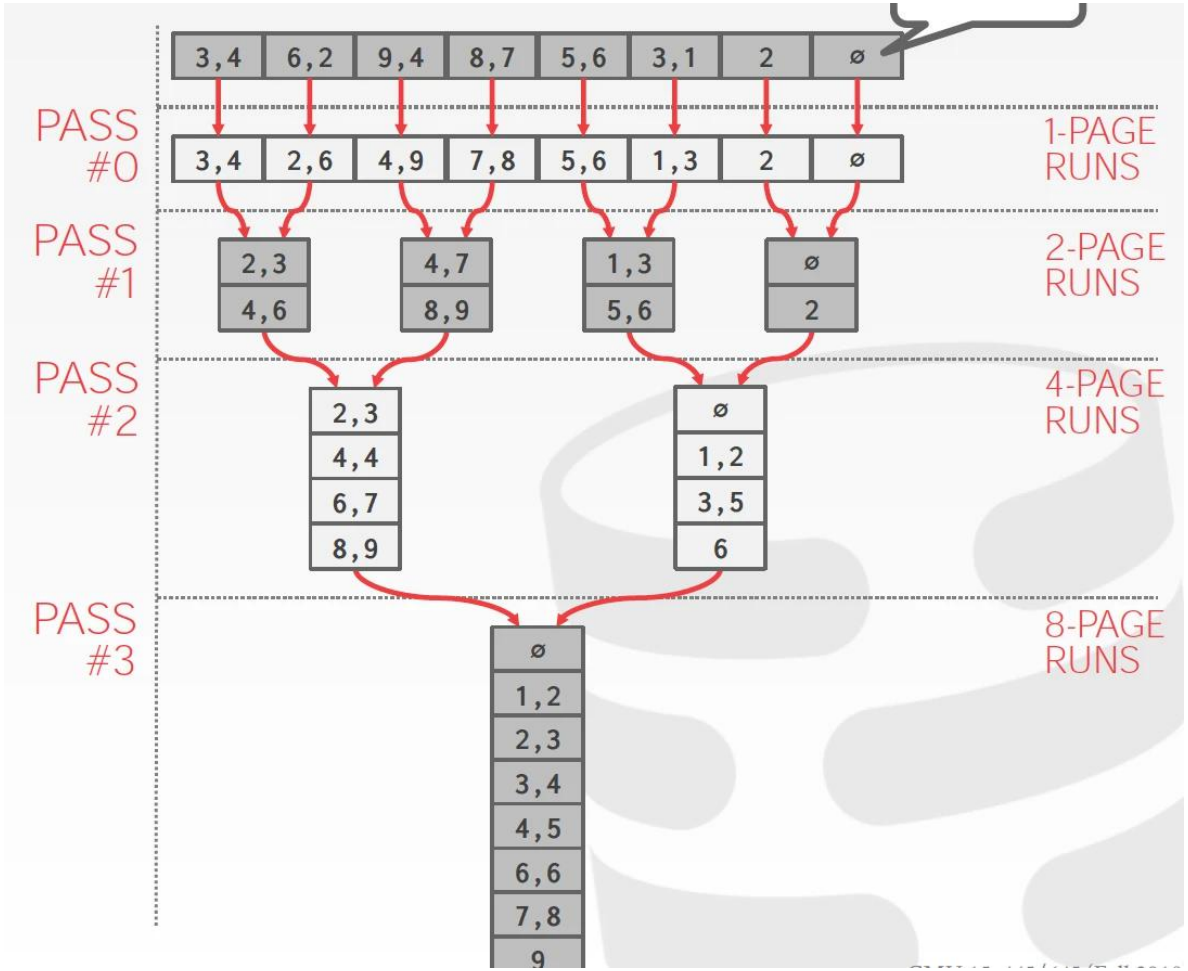
c1	c2	c3	c4
3	6	12	10

c1	c2	c3	c4
13	15	19	20

c1	c2	c3	c4
2	14	23	5

排序与分组

复杂度：$N*log_2(N)$



Join操作

1、NestedLoopJoin

mysql(<5.6)

BlockNestedLoopJoin

mysql(>=5.6)

2、SortMergeJoin

3、HashJoin

mysql(>= 8.0.18)

1、https://dev.mysql.com/doc/refman/8.0/en/nested-loop-joins.html

2、https://dev.mysql.com/doc/refman/8.0/en/bnl-bka-optimization.html

3、https://dev.mysql.com/doc/refman/8.0/en/hash-joins.html

NestedLoopJoin

```
def NestedLoopJoin(table_a, table_b)
    for row_a in table_a.rows():
        for row_b in table_b.rows():
            if condition(row_a, row_b):
                process_tuple(row_a, row_b);
```

Block-based NestedLoopJoin

```
def BlockNestedLoopJoin(table_a, table_b)
    for row_a in table_a.blocks():        // M block
        for row_b in table_b.blocks():    // N block
            for row_a in table_a.rows():
                for row_b in table_b.rows():
                    if condition(row_a, row_b):
                        process_tuple(row_a, row_b);
```

时间复杂度：M+M*N

把小表放在外层能减少IO次数，小表也被叫做driverd表

NestedLoopIndexJoin

```
def NestedLoopIndexJoin(table_a, table_b)
    for row_a in table_a.blocks():        // M block
        for row_b in table_b.index():     // n block
            for row_a in table_a.rows():
                    if condition(row_a, b_join_column):
                        process_tuple(row_a, row_b);
```

时间复杂度：M+M*n

SortMergeJoin

```
def SortMergeJoin(table_a, table_b)
    (table_a_join_key,table_a_row) = next(table_a);
    (table_b_join_key,table_b_row) = next(table_b);
    while table_a_join_key and table_b_join_key:
        if equal(table_a_join_key, table_b_join_key):
            process(table_a_row, table_b_row);
            (table_a_join_key,table_a_row) = next(table_a);
            (table_b_join_key,table_b_row) = next(table_b);
        elseif gt(table_a_join_key, table_b_join_key):
            (table_b_join_key,table_b_row) = next(table_b);
        else:
            (table_a_join_key,table_a_row) = next(table_a);
```

没有索引复杂度：$M + N + 2M * (\log_2 M) + 2N * (\log_2 N)$

有索引复杂度：m+n

HashJoin

```
def HashJoin(table_a, table_b):
    hashtable_a = build_hashtable(table_a);
    for row_b in table_b.rows():
        if match(rowb, hashtable_a):
            process_row();
```

较小的表 A 建立的哈希表能够全部放进内存的话，时间复杂度只有M+N。

如果表 A 比较大，不能全部放进内存的话，我们又要借助外部哈希表的算法来分而治之：用一个哈希函数把表 A 和表 B 分别划分到 x 个 bucket，这个阶段我们叫它partition phase，复杂度是2 * (M + N)；第二阶段就是对于每个 bucket，分别读取表 A 和表 B 然后建立哈希表做 JOIN，称之为probing phase，复杂度是M+N，整体复杂度是3*(M+N)。

Join操作

类别	Nested Loop	Hash Join	Merge Join
使用条件	任何条件	等值连接（=）	等值或非等值连接(>，<，=，>=，<=)，‘<>’除外
相关资源	CPU、磁盘I/O	内存、临时空间	内存、临时空间
特点	当有高选择性索引(NestedLoopIndexJoin)或进行限制性搜索时效率比较高，能够快速返回第一次的搜索结果。	当缺乏索引或者索引条件模糊时，Hash Join比Nested Loop有效。通常比Merge Join快。在数据仓库环境下，如果表的记录数多，效率高。	当缺乏索引或者索引条件模糊时，Merge Join比Nested Loop有效。 非等值连接时，Merge Join比Hash Join更有效
缺点	当索引丢失或者查询条件限制不够时，效率很低；当表的记录数多时，效率低。	为建立哈希表，需要大量内存。第一次的结果返回较慢。	所有的表都需要排序。它为最优化的吞吐量而设计，并且在结果没有全部找到前不返回数据。

开源优化器：https://github.com/greenplum-db/gporca

索引不是越多越好

1、索引过多，因为要更新索引，会降低插入、修改、删除的性能。

2、频繁更新的列尽量不要建索引。

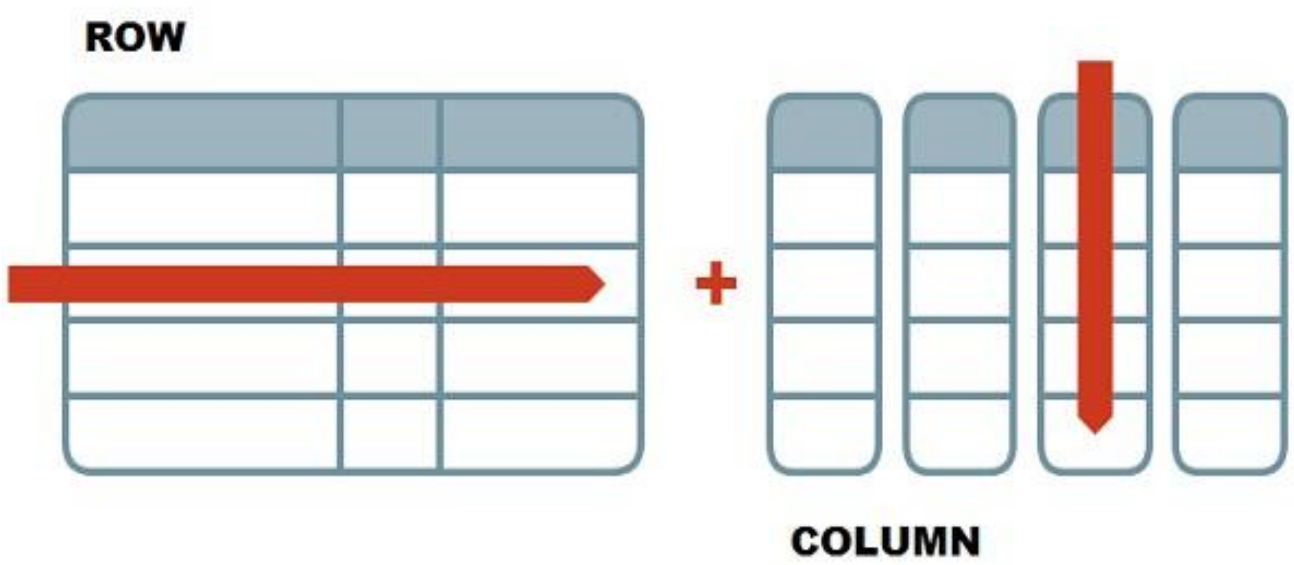
3、只对查询条件中的列和排序的列建索引。

4、复合索引，将选择性高的、查询频繁的列放前面。

行式存储 vs. 列式存储

OLTP
事务型

原子性
一致性
隔离性
持久性



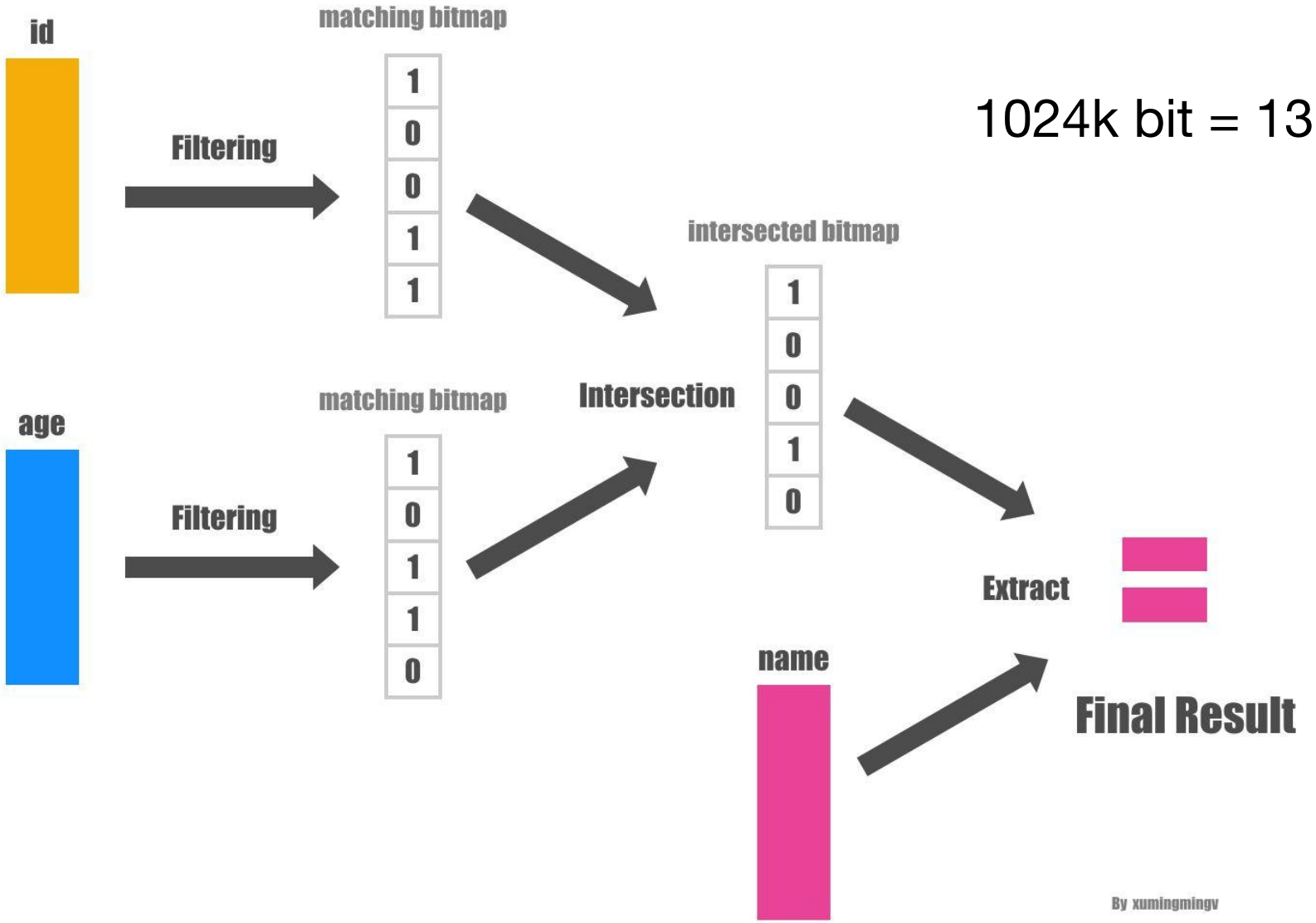
以行为单位存储
便于事务操作

以列为单位存储
便于压缩、读取
部分行、表的列
数没有限制。

OLAP
分析型

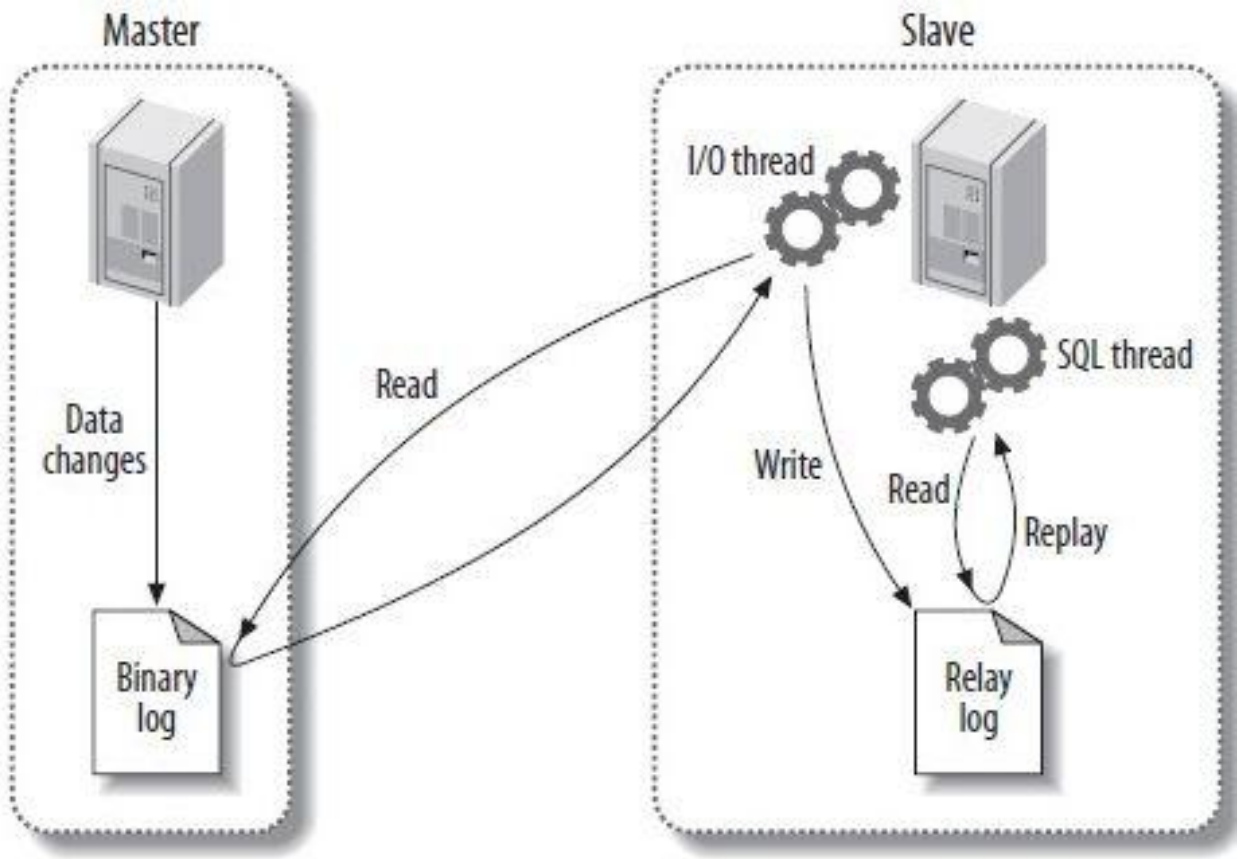
聚合
排序
计算

SELECT name FROM person WHERE id > 10 and age > 20



1024k bit = 131KB

数据复制：横向扩展与高可用



同步复制 vs. 异步复制

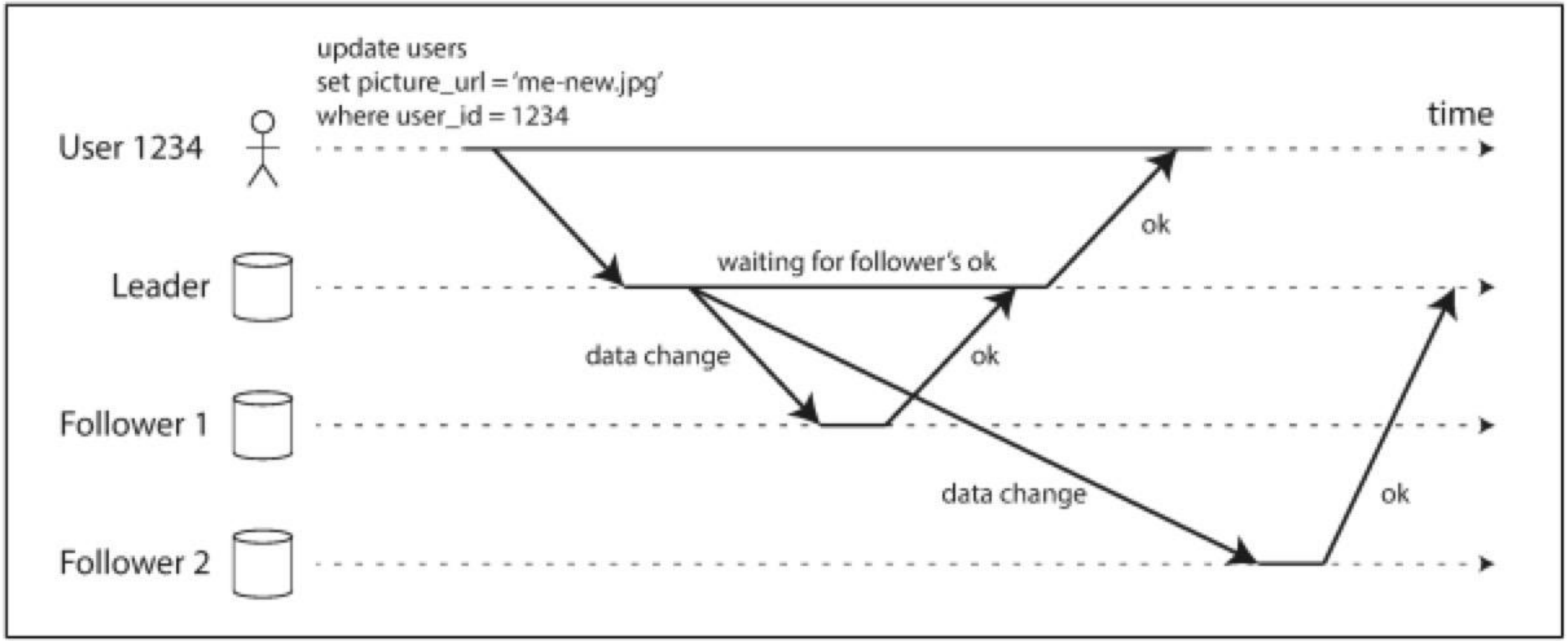
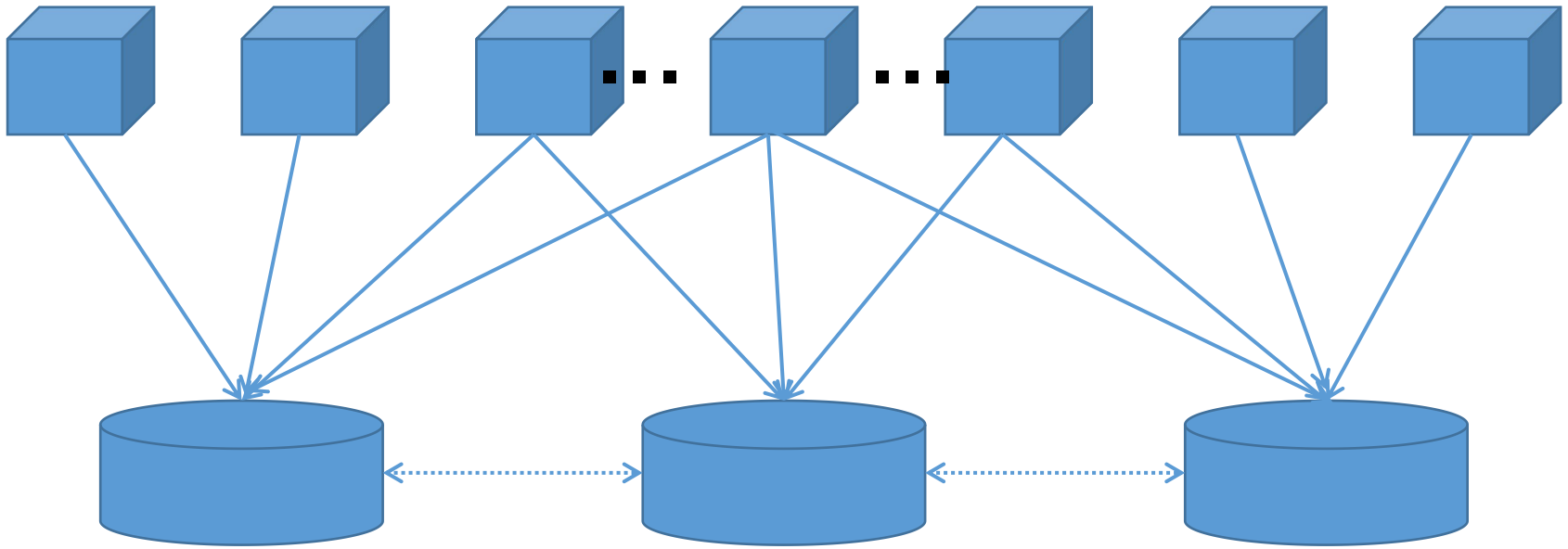


Figure 5-2. Leader-based replication with one synchronous and one asynchronous follower.

存储与计算分离

App容易横向扩展，但数据库因为需要数据复制不容易横向扩展。
对于OLTP型查询，尽量不要把复杂的操作放在数据库层。



大数据仓库与分布式存储

1、传统MySQL等RDBMS单表百万级以上的大数据量存储性能急剧下降
2、分库分表方案，join、排序、分组实现困难，无法做统计
3、PB级数据单机无法存储、更别说直接做统计查询。

Google系列论文

云产品

BigQuery | Cloud DataFlow | GAE | Cloud Bigtable | Cloud Spanner

OLAP/OLTP

计算

存储/数据库

Ali-NPU | TPU 2017

TensorFlow | PAI | TensorFlow 2016

UDAP | Goods 2016 | Shasta 2016

Drill(Parquet) Impala Presto | ODPS-AliORC | Dremel 2010

ODPS-Lighting | Mesa 2014 | Kudu

ODPS-R

PowerDrill 2012 | ADS | Druid

Mahout | ODPS-Xlib

MillWheel 2013 | Blink | Flink Storm

Spark | ODPS-Spark

F1 2013 | TiDB

OceanBase | CockroachDB

Hama/ Giraph | ODPS-Graph Geabase | Pregel 2010

FlumeJava 2010 | Percolator 2011 | Spanner 2012 | TiKV

Hive | ODPS-SQL | Tenzing 2011

Pig/Dryad | Sawzall 2003 | MegaStore 2011 | Lindorm

ODPS-MR

Hadoop Yarn | Fuxi | MapReduce 2004 | Bigtable 2006 | OTS | HBase

ODPS-DAG

Mesos

K8S | Sigma | Borg 2015 | Omega 2013 | GFS 2003 | Pangu | HDFS | Chubby 2006 | Nuwa | Zookeeper

CPI2 2013 | GWP 2010 | The Datacenter as a Computer 2009 | GoogleCluster 2003 | Dapper 2010 | EagleEye | Zipkin

NetO | B4 2013

Google Web Search Engine 1998

谢谢

- 更多参考:
- https://blog.hufeifei.cn/2018/04/DB/mysql/01-b-tree-hash-index/
- https://blog.hufeifei.cn/2021/06/DB/index/
- https://blog.hufeifei.cn/2021/06/DB/how-datebases-work/
- https://www.infoq.cn/theme/46
- https://db-engines.com/en/ranking