GREENPLUM DATABASE

阿里云 aliyun.com

# Greenplum内核揭秘之架构解读

钉钉直播 | 4月17日 16：00-17：00

GREENPLUM DATABASE

# 课程大纲

- 数据库管理系统
- Greenplum整体架构
- 存储管理
- 索引
- 查询执行
- 事务与日志
- 准备工作

# 数据库管理系统

# 基于文件的存储

GREENPLUM DATABASE

**Bars (name, address, city)**

```
('Toronado', '547 Haight St', 'San Francisco')
('The Beer Hall', '1 Polk St', 'San Francisco')
('Sunset Beer', '1498 Sunset Blvd', 'Los Angles')
```

**Sells (bar, price, beer, date)**

```
('Toronado', $10, 'Corona', '2018-07-21')
('Sunset Beer', $5, 'Skol', '2014-10-13')
```

统计每个店的啤酒销售额度

```
for bar_record in Bars
for sell_record in Sells
    if sell_record[0] == bar_record[0]
        bars_sell_count[bar_record[0]]+=
            sell_record[1]
```

问题？

- 如何保证数据数据的一致性
  - Bars文件
  - Sells文件
- 如何修改记录？
  - 特别是变长记录
- 每次查询都需要写遍历程序，冗余，复杂并且低效
- ……

# 数据库和数据库管理系统

GREENPLUM DATABASE

数据库

- 数据库是一组内部之间有相互关联关系，并且有效组织的数据的集合

数据库管理系统

- 用于管理数据库的软件系统，用于高效创建和管理大量数据，使得数据能够安全长期保存

关系型数据库

- 基于关系模型做为底层的数据模型的数据库管理系统

# 关系模型

- 数据定义和存储（DDL）
  - 存储数据的格式和内容
  - 变长数据和修改
  - ...

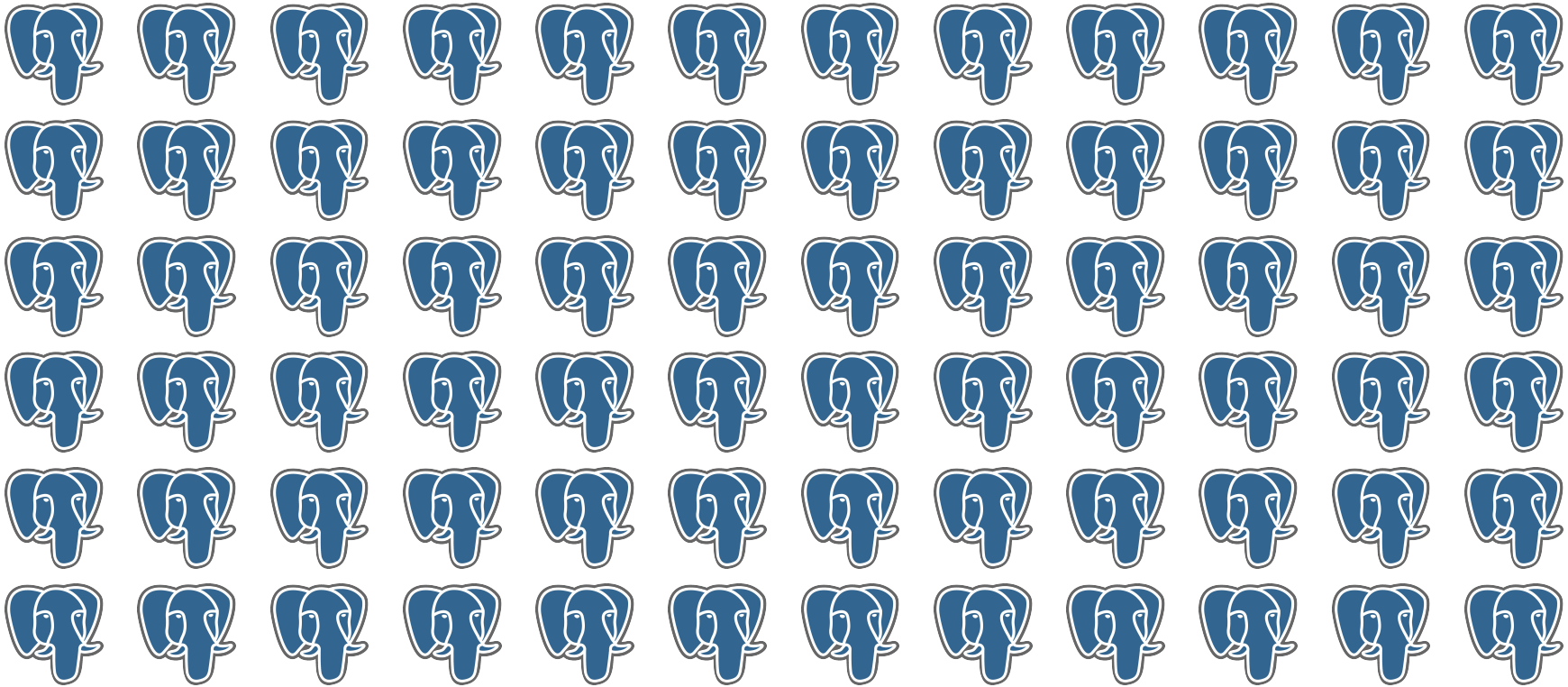
- 数据完整性（ACID）
  - 数据满足要求的约束条件
  - 事务和并发控制
  - ....

- 数据操作（DML）
  - SQL结构化查询语言：访问和修改数据库内容
  - SELECT, INSERT, UPDATE
  - 扫描，过滤，连接，聚合，投影，排序，集合操作
  - ...

# Greenplum整体架构

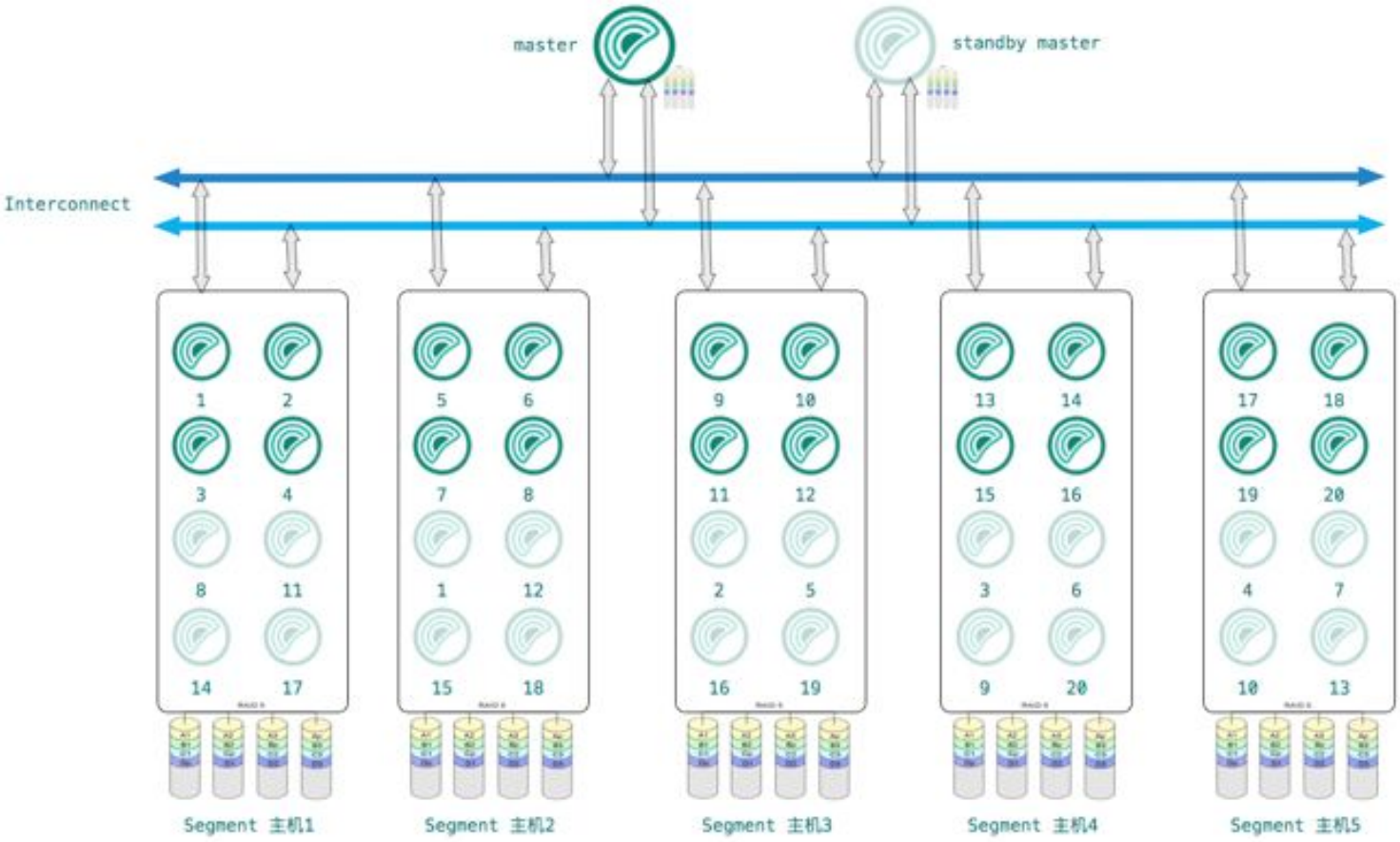
# 本质上是单机Postgres组成的数据库集群

GREENPLUM DATABASE



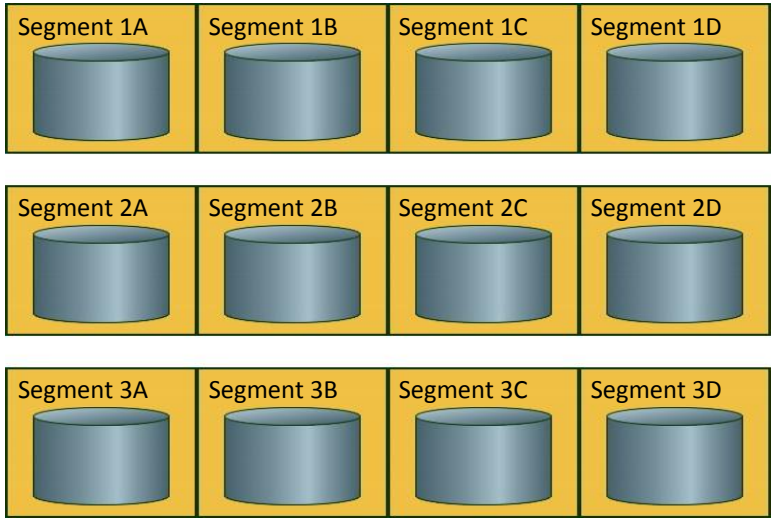
# 对外提供统一的数据库接口，并针对集群处理做了大量优化

# 基于MPP无共享拓扑


master
standby master
Interconnect
1 2 3 4 8 11 14 17
5 6 7 8 1 12 15 18
9 10 11 12 2 5 16 19
13 14 15 16 3 6 9 20
17 18 19 20 4 7 10 13
Segment 主机1
Segment 主机2
Segment 主机3
Segment 主机4
Segment 主机5

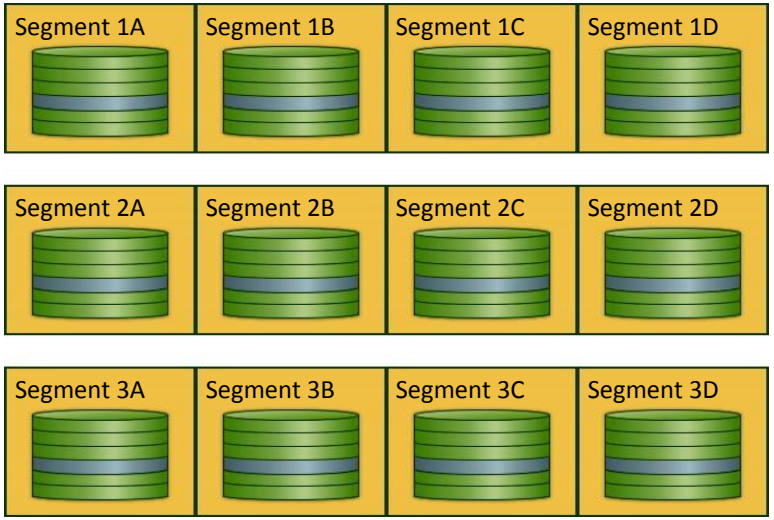
# 数据分布与分区

```
SELECT COUNT(*)
  FROM Sells
 WHERE date >= '2017-10-20'
   AND date <  '2019-10-20'
```




Sells 表数据按照bar均匀分布于各个节点

&

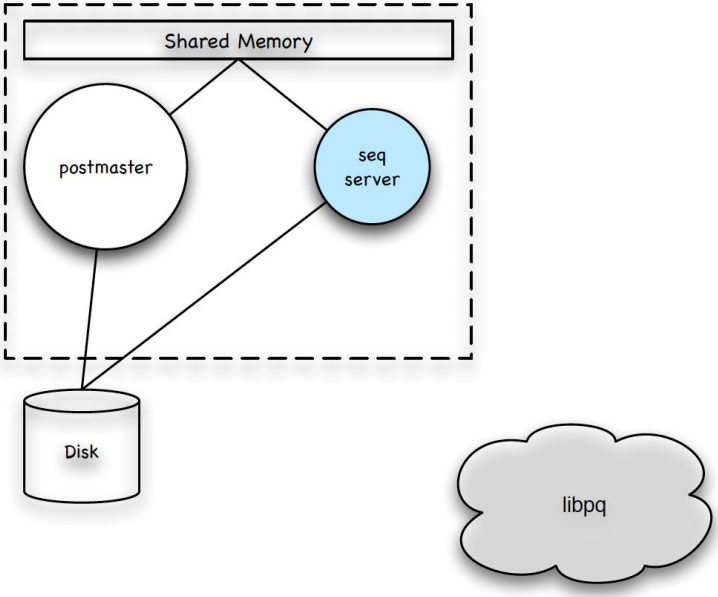


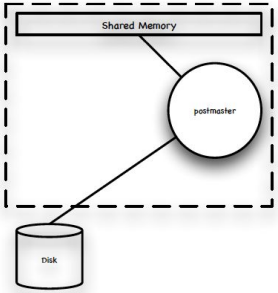

仅仅扫描 Sells 表相关的对应于[‘2011-10-20’, ‘2019-10-20’)的分区子表

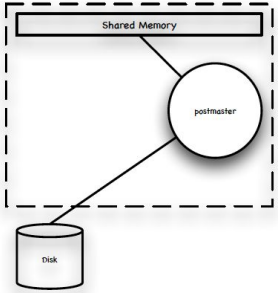
# 系统空闲状态

- 1个master，2个segment
- postmaster 是数据库主进程，监听用户请求
- 此时系统空闲，没有任何运行查询

0. The system at rest.



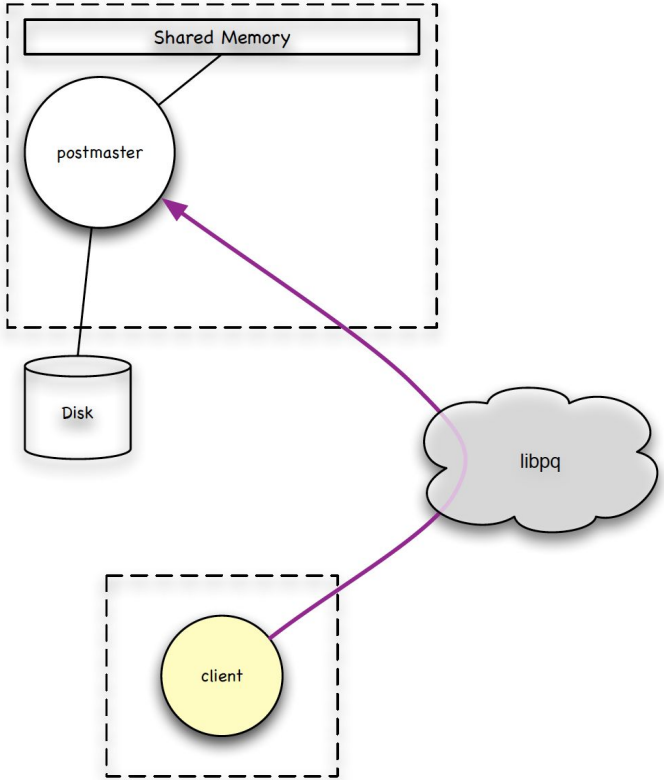



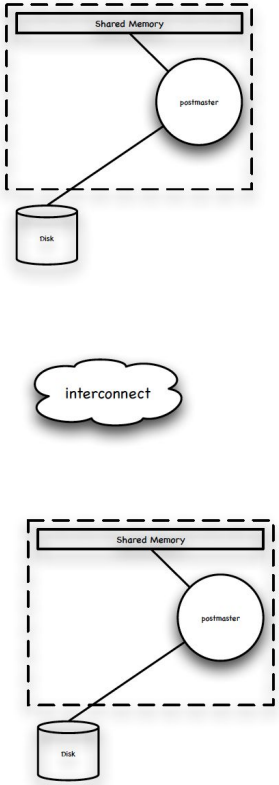


# 客户端建立会话连接

1. Client connects via the entry postmaster.

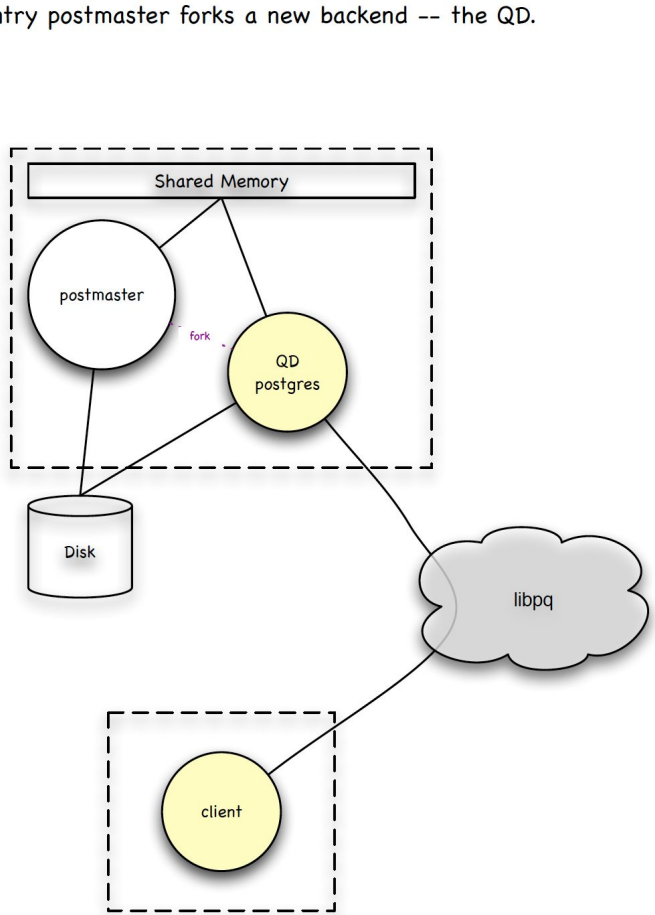
- 客户端通过 libpq 协议发送连接请求给 Greenplum master 节点
- Master 节点上 postmaster 进程会监听到连接请求，并处理

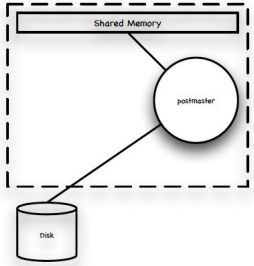


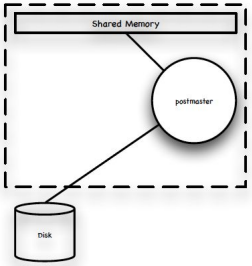



# Master fork 一个进程处理客户端请求

- Master 上的postmaster 进程监听到连接请求后，fork 一个子进程用于处理该客户端的所有查询请求
- 子进程在Postgres中称为 backend；在 Greenplum 中该进程被称为 QD

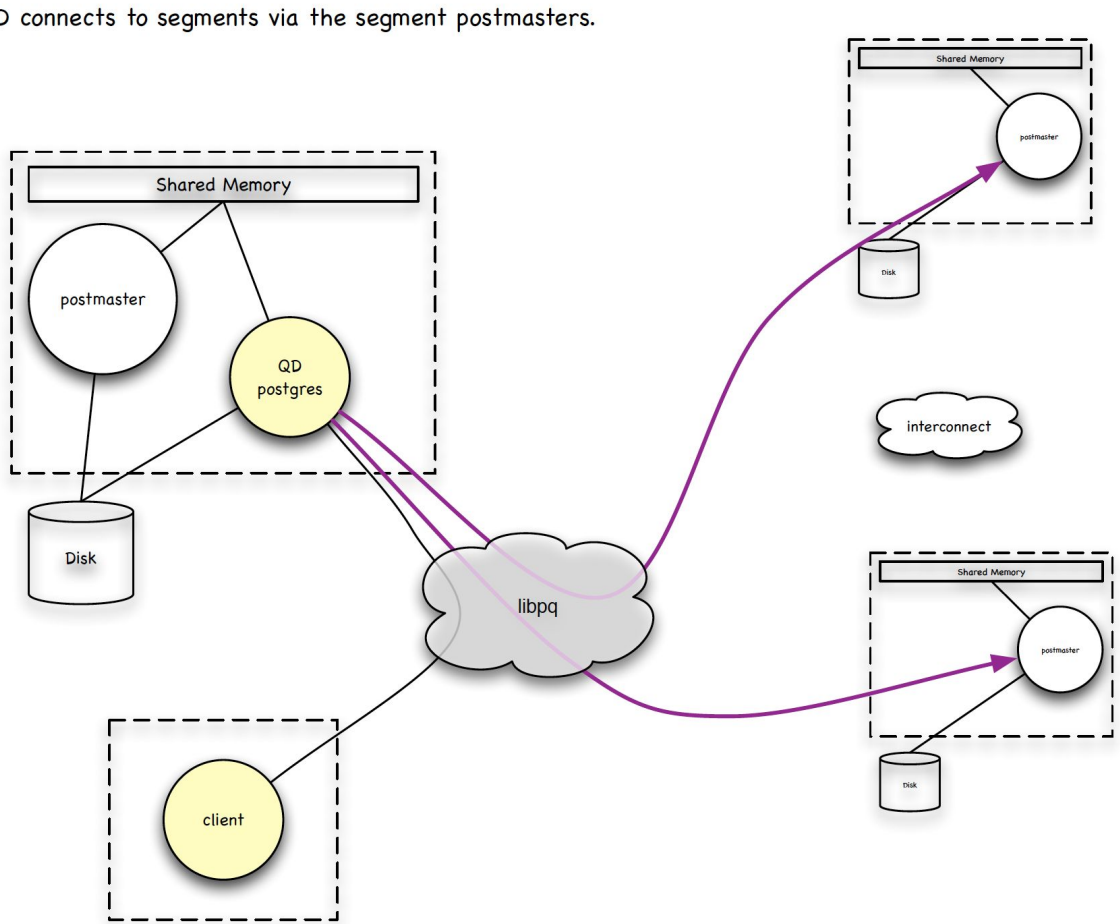




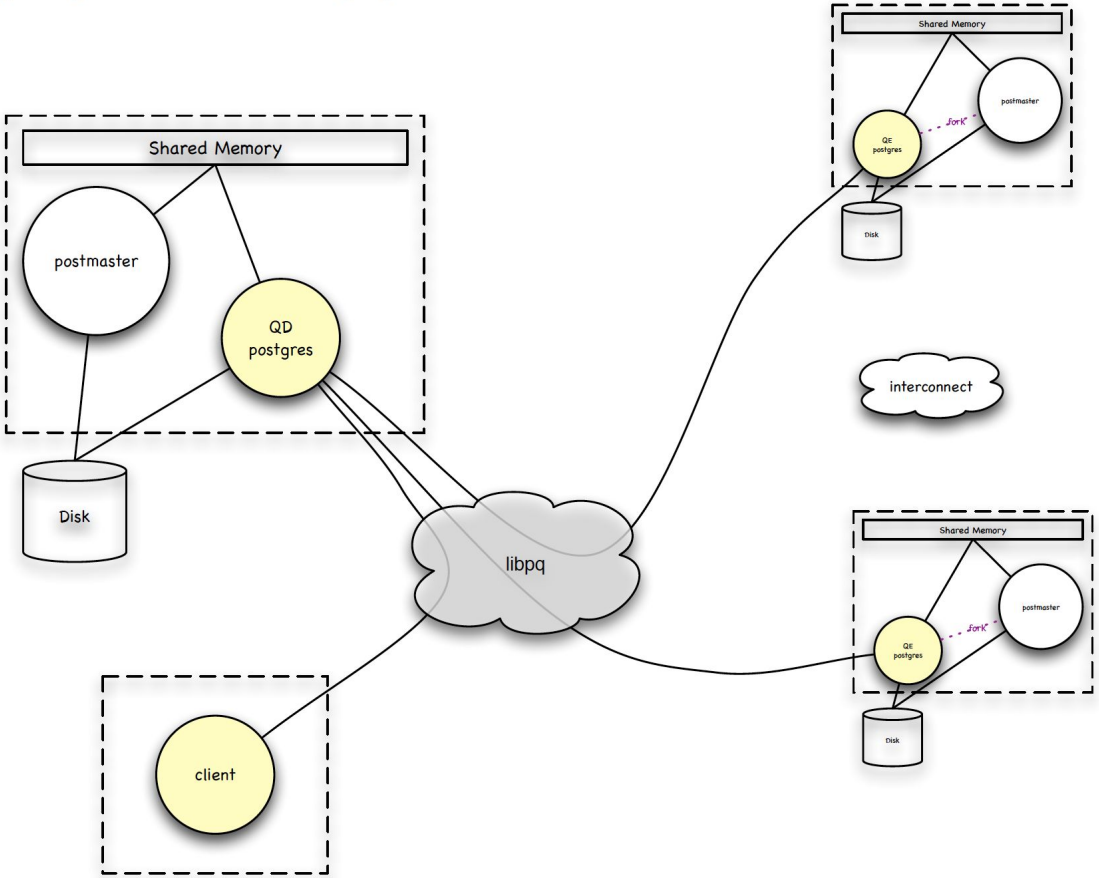
# QD 建立和 Segment 的连接

- QD 进程使用libpq协议和每个segment建立连接请求
- Segment 上的 postmaster 进程监听到 QD 的连接请求，并进行处理
- 对于 segment 而言，QD 是它们的客户端
- 仅有在需要时QD才会建立和Segment的连接



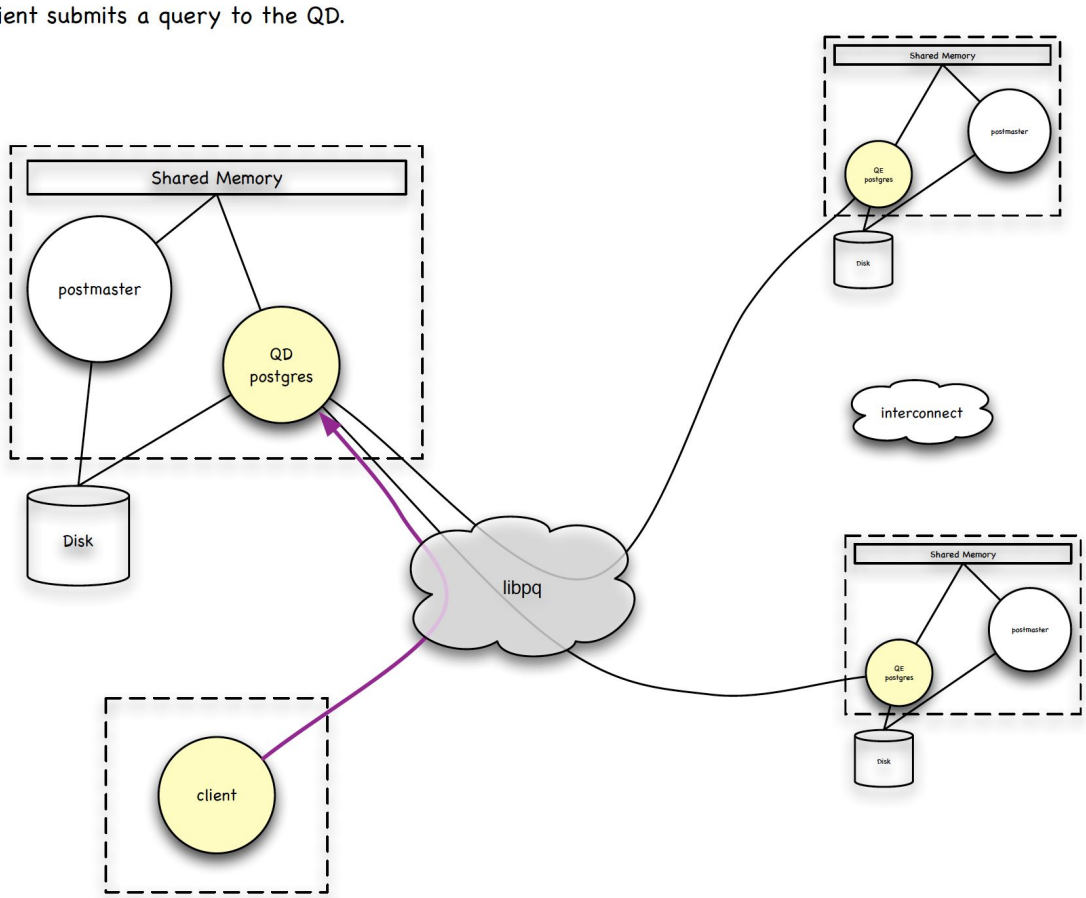
# Segment fork 一个子进程处理 QD 的连接请求

- Segment 上的 postmaster 进程监听到 QD 的连接请求后，创建一个子进程以处理后续查询请求
- Segment 上创建的子进程成为 QE



# 查询请求通过 QD发送给QE进程

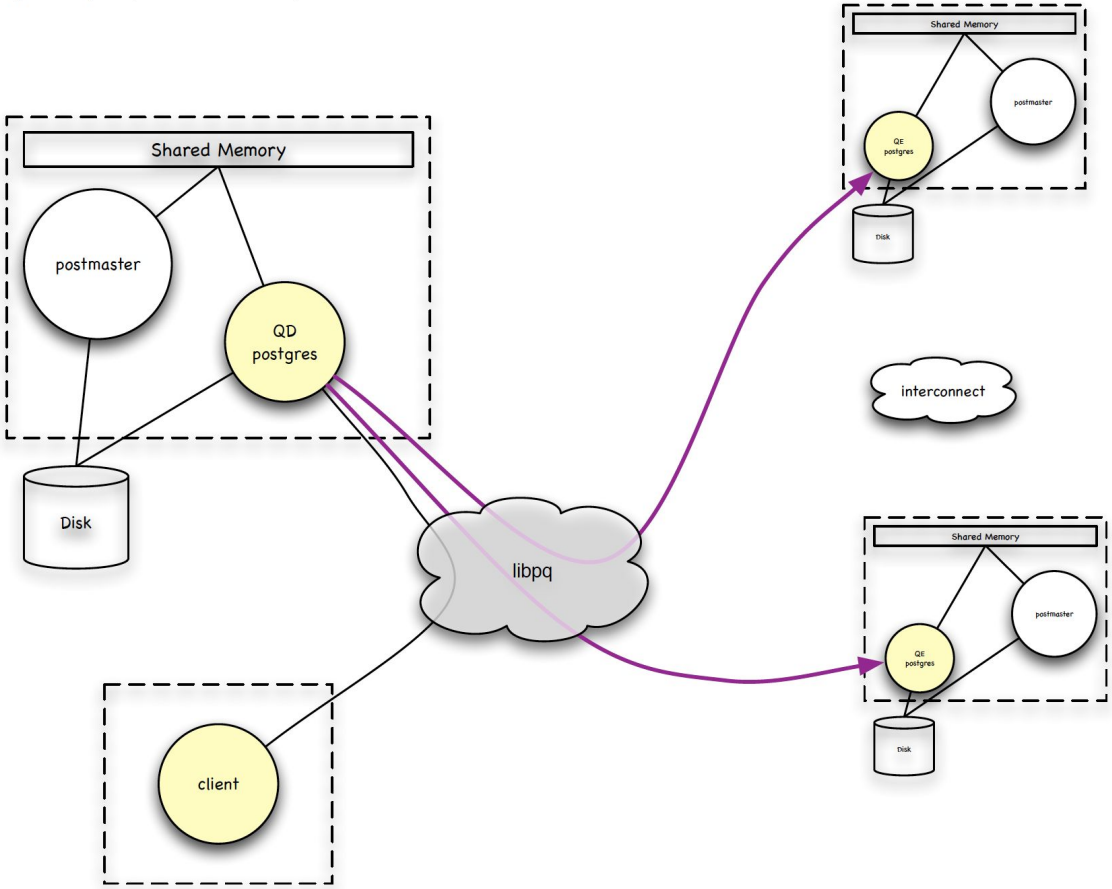
- 客户端使用 libpq 协议发送查询请求给 master 上的 QD 进程
- QD 进程对收到的查询进行处理，包括解析、优化生成分布式查询计划等



# QD 发送任务给 QE

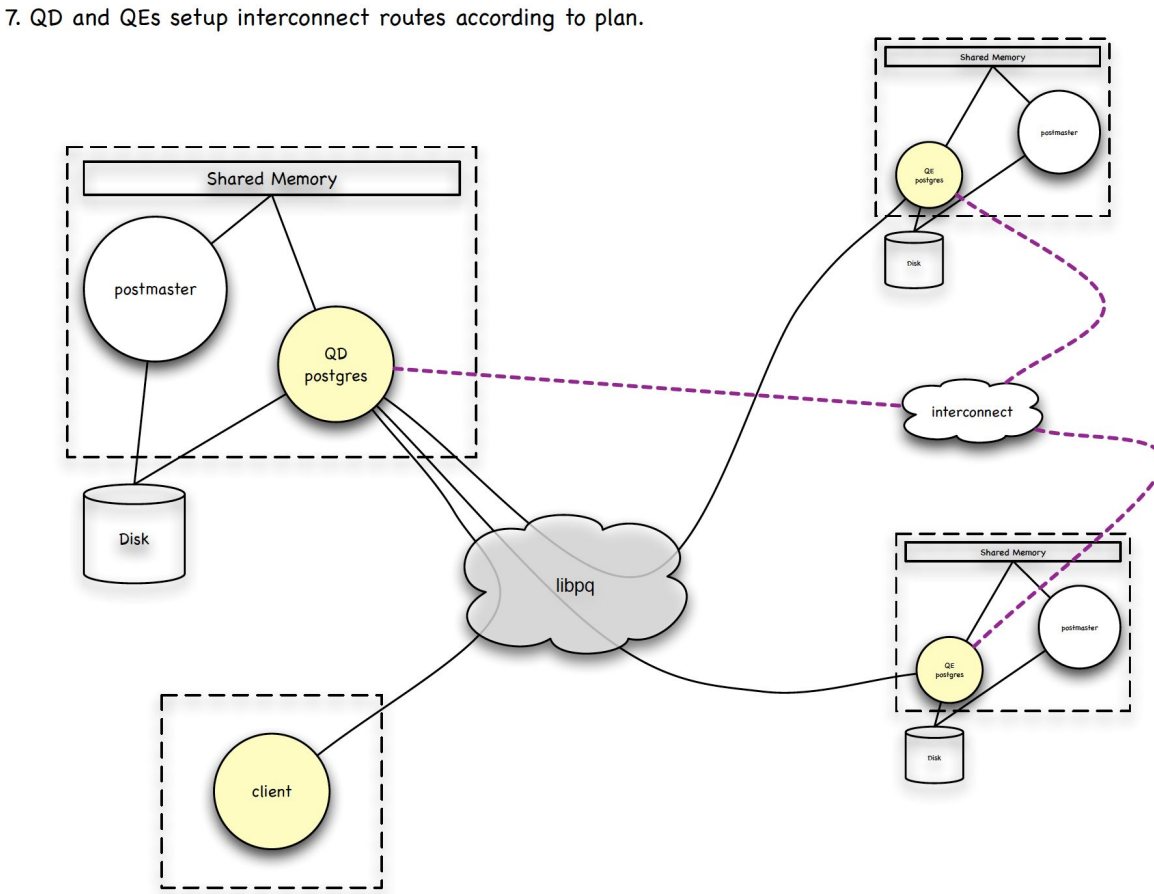
- QD 生成分布式查询计划后，通过libpq协议发送给各个 segment 上的 QE 进程

6. QD plans query and submits plans to QEs.



# QD和QEs建立数据通信通道

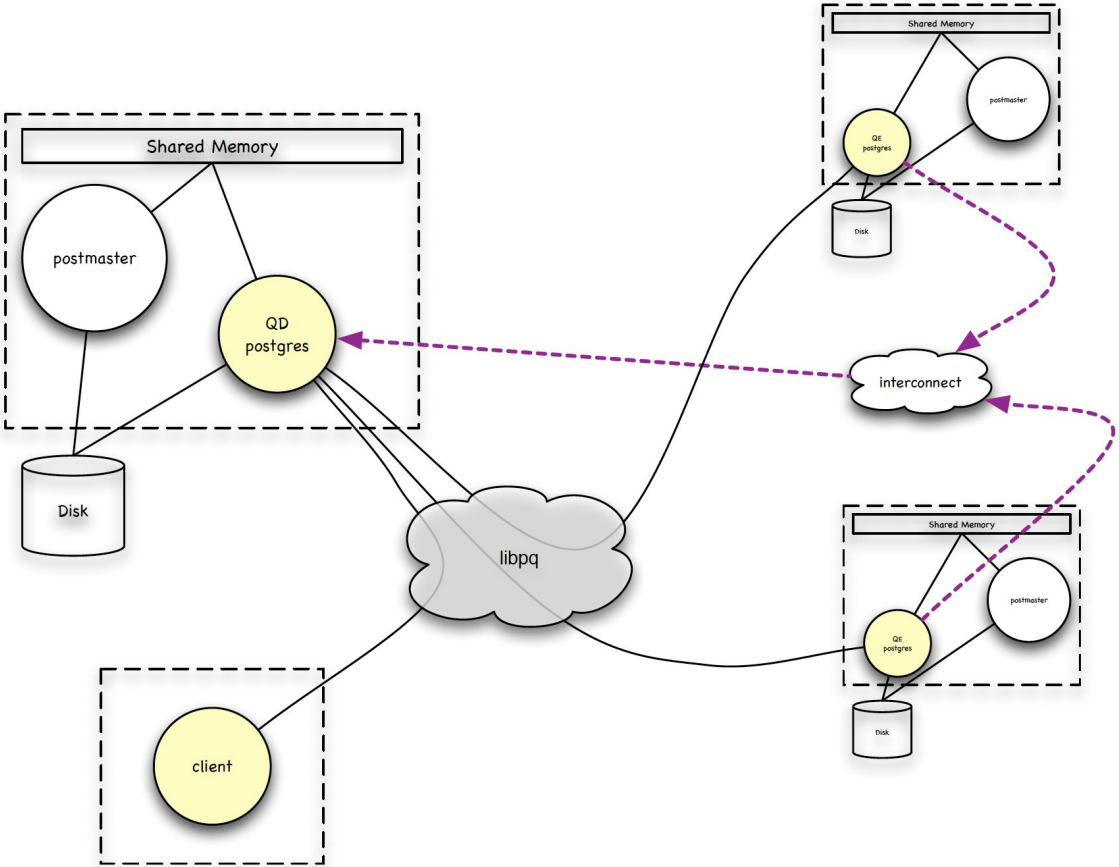
- QD和每个segment上的QEs根据查询计划里面的信息，建立interconnect连接
- Interconnect 用于内部数据通信
- Libpq 用于控制命令和结果返回



# QE 各司其职

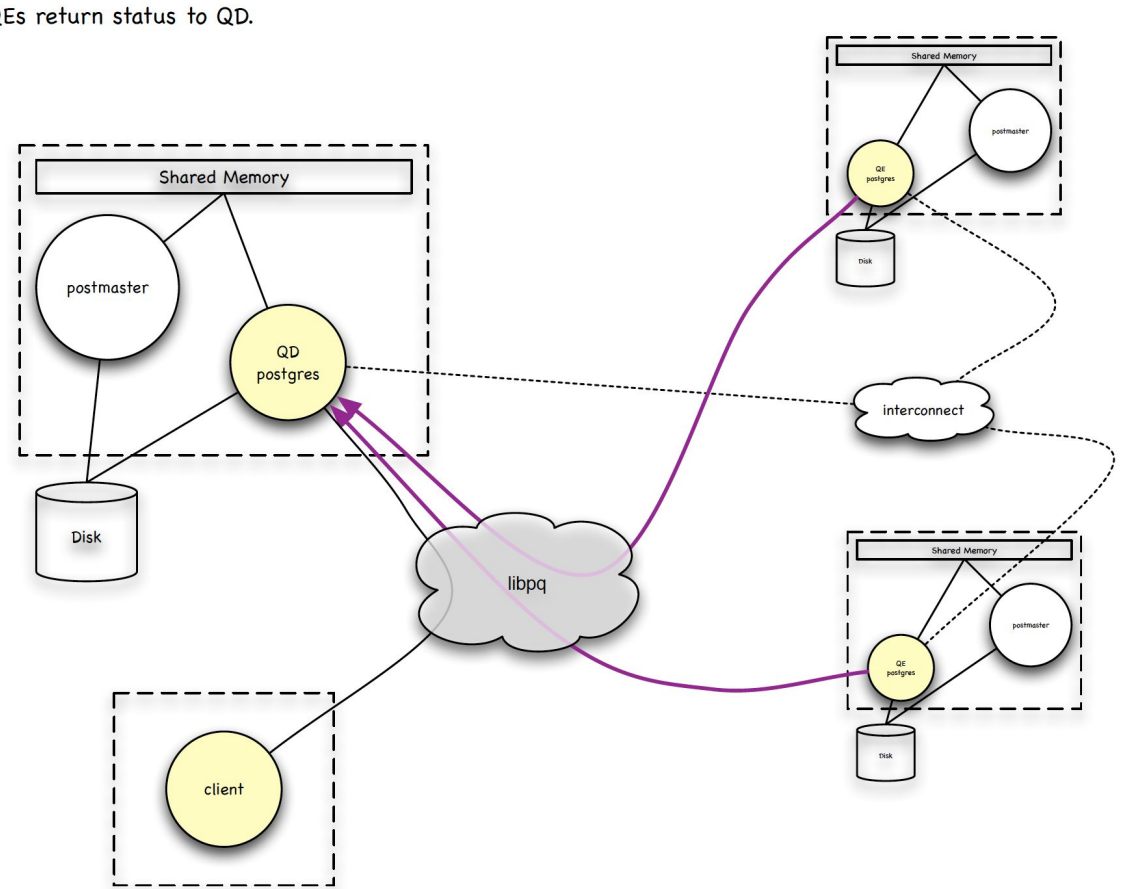
- 每个 QE 执行分配给它的任务
- QE之间的通过interconnect交互数据

8. QD and QEs execute their slices sending tuples up the slice tree.



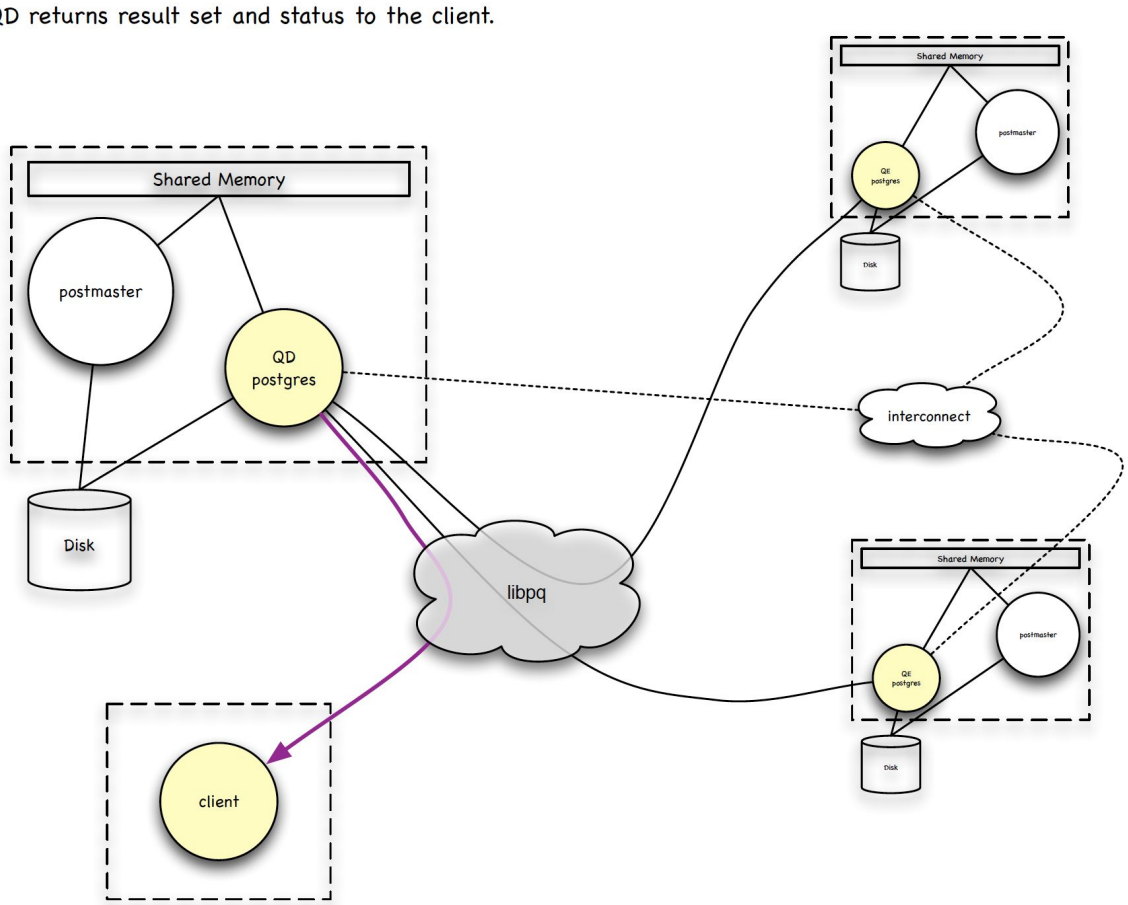
# QE 状态管理

- QE和QD之间通过libpq协议进行状态更新和管理，包括错误处理等
- QE之间没有 libpq 连接



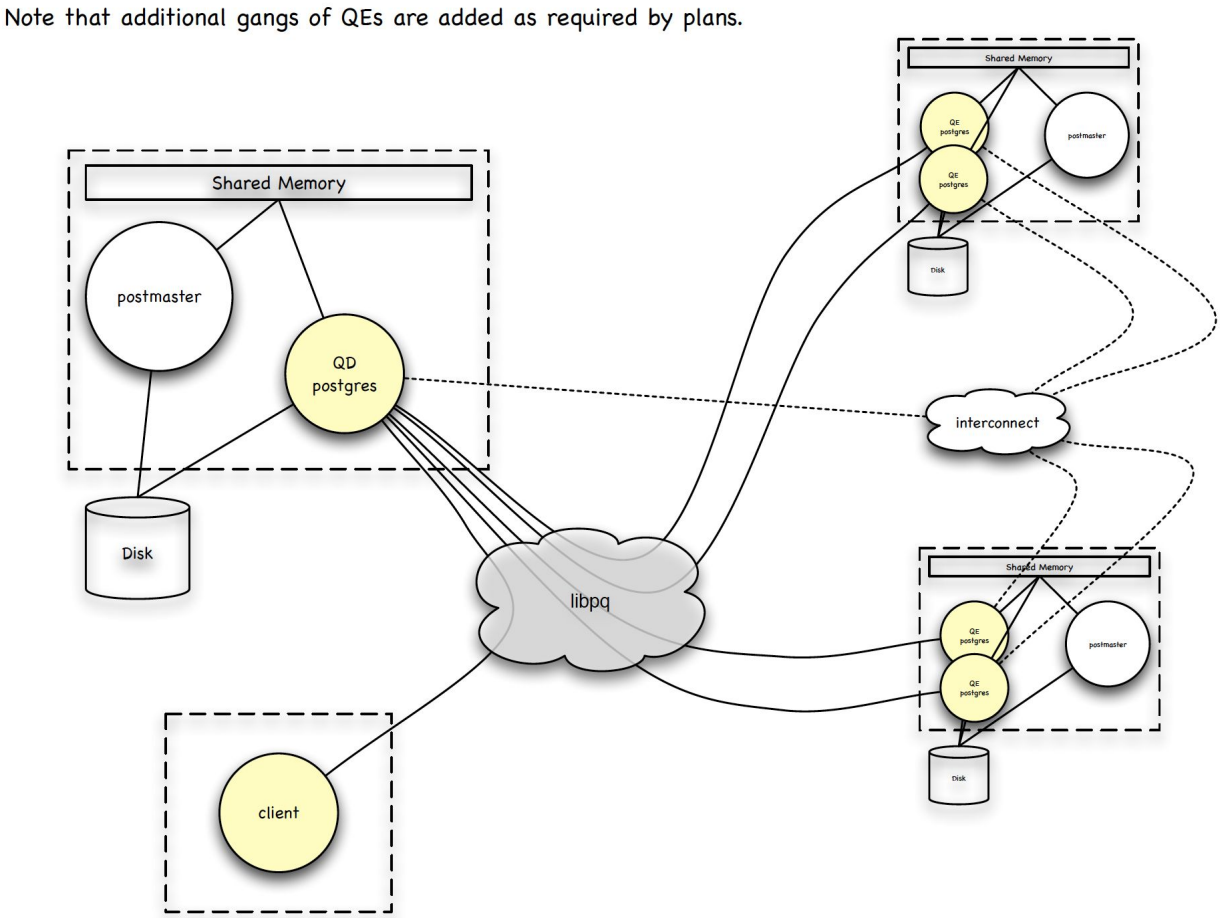
# QD 返回查询结果给客户端

- 最终 QD 将查询的结果返回给客户端



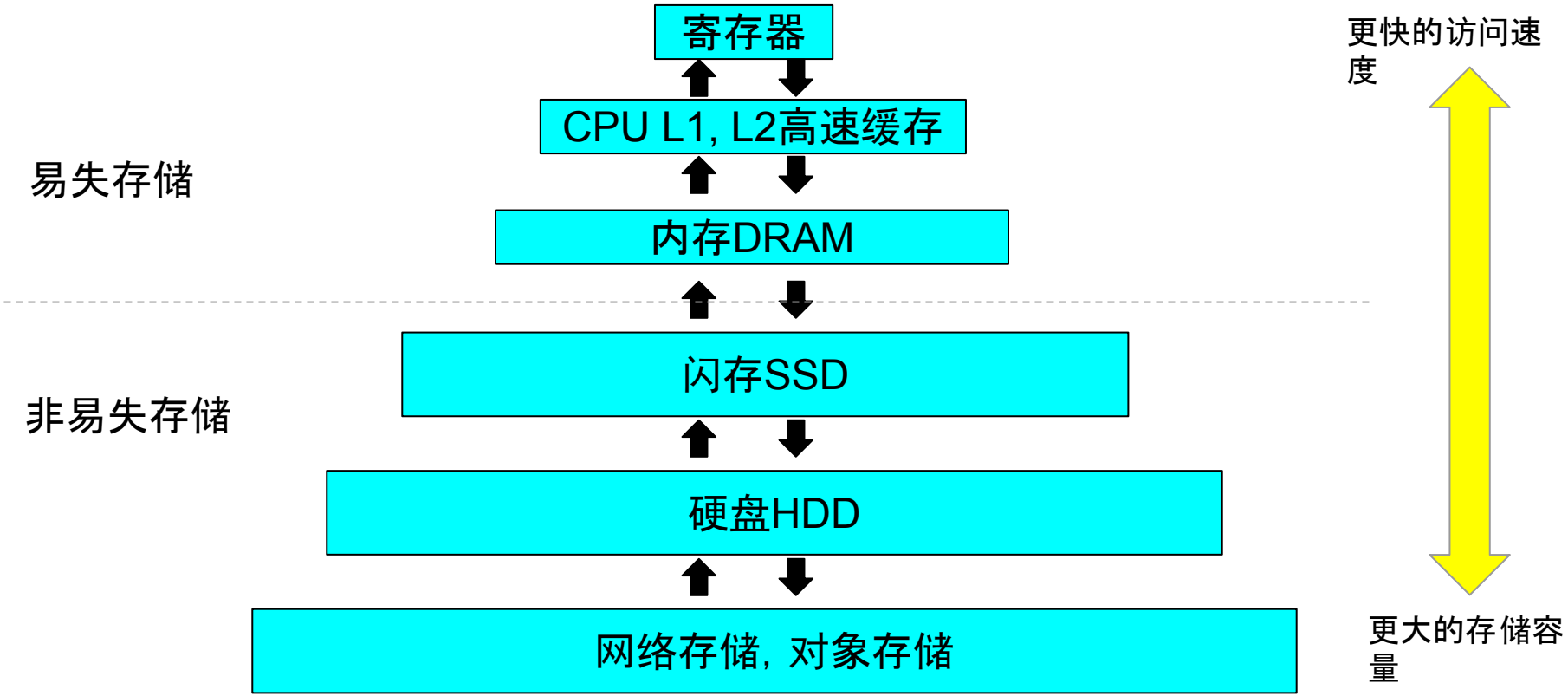
# 算子并行化

- PostgreSQL 是进程模型，为每个查询创建单个进程进行处理为了提高CPU利用率，Greenplum实现了Segment内算子之间查询并行化
- 用户查询在单个 Segment 上可以有多个 QE，QE之间通过 interconnect 交互数据

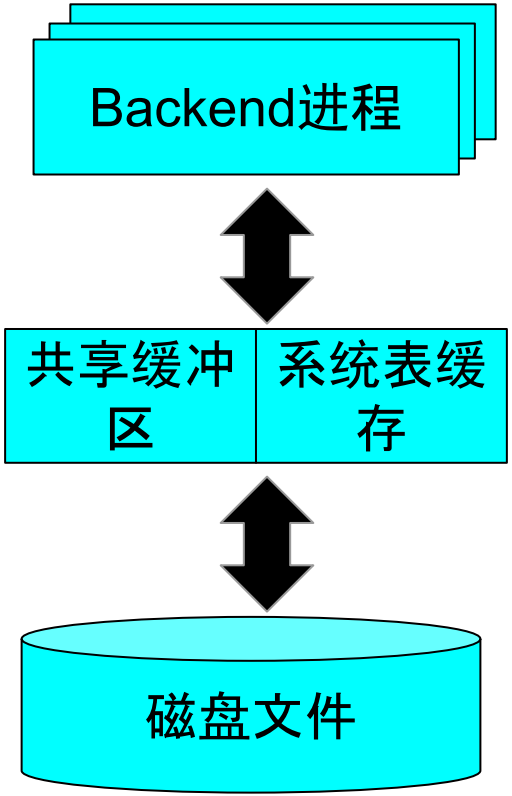


# 存储管理

# 存储介质

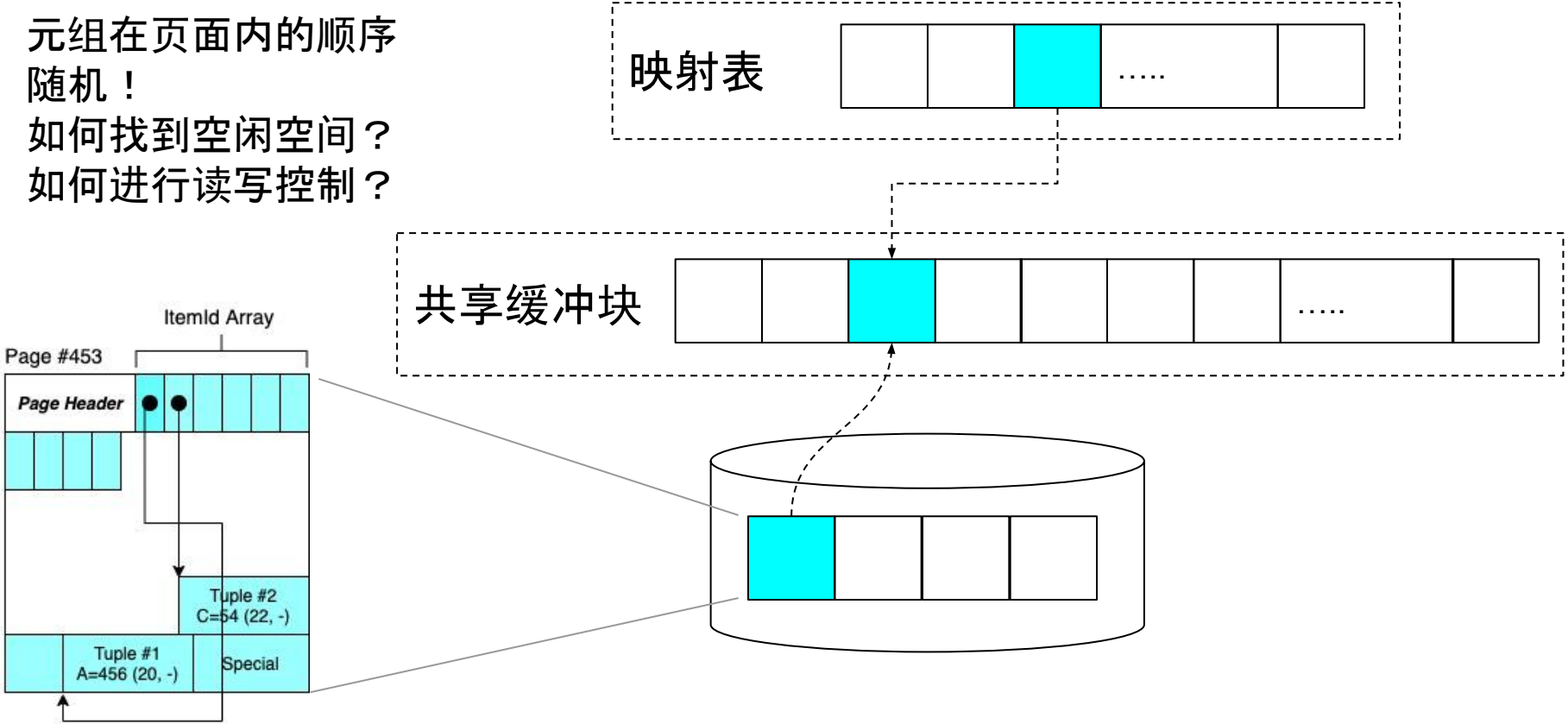

寄存器
CPU L1, L2高速缓存
内存DRAM
闪存SSD
硬盘HDD
网络存储，对象存储
易失存储
非易失存储
更快的访问速度
更大的存储容量

# Greenplum存储层次



# Greenplum共享缓冲区

元组在页面内的顺序随机！
如何找到空闲空间？
如何进行读写控制？



# Greenplum多版本控制MVCC

| Version | Value | t_xmin | t_xmax |
|---|---|---|---|
| A0 | 123 | 10 | — |

a) 事务 10 INSERT A=123

| Version | Value | t_xmin | t_xmax |
|---|---|---|---|
| A0 | 123 | 10 | — |
| B0 | 231 | 14 | — |

b) 事务 14 INSERT B=231

| Version | Value | t_xmin | t_xmax |
|---|---|---|---|
| A0 | 123 | 10 | — |
| B0 | 231 | 14 | 18 |

c) 事务 18 DELETE B

| Version | Value | t_xmin | t_xmax |
|---|---|---|---|
| A0 | 123 | 10 | 20 |
| B0 | 231 | 14 | 18 |
| A1 | 456 | 20 | — |

d) 事务 20 UPDATE A to 456

| Version | Value | t_xmin | t_xmax |
|---|---|---|---|
| A1 | 456 | 20 | — |

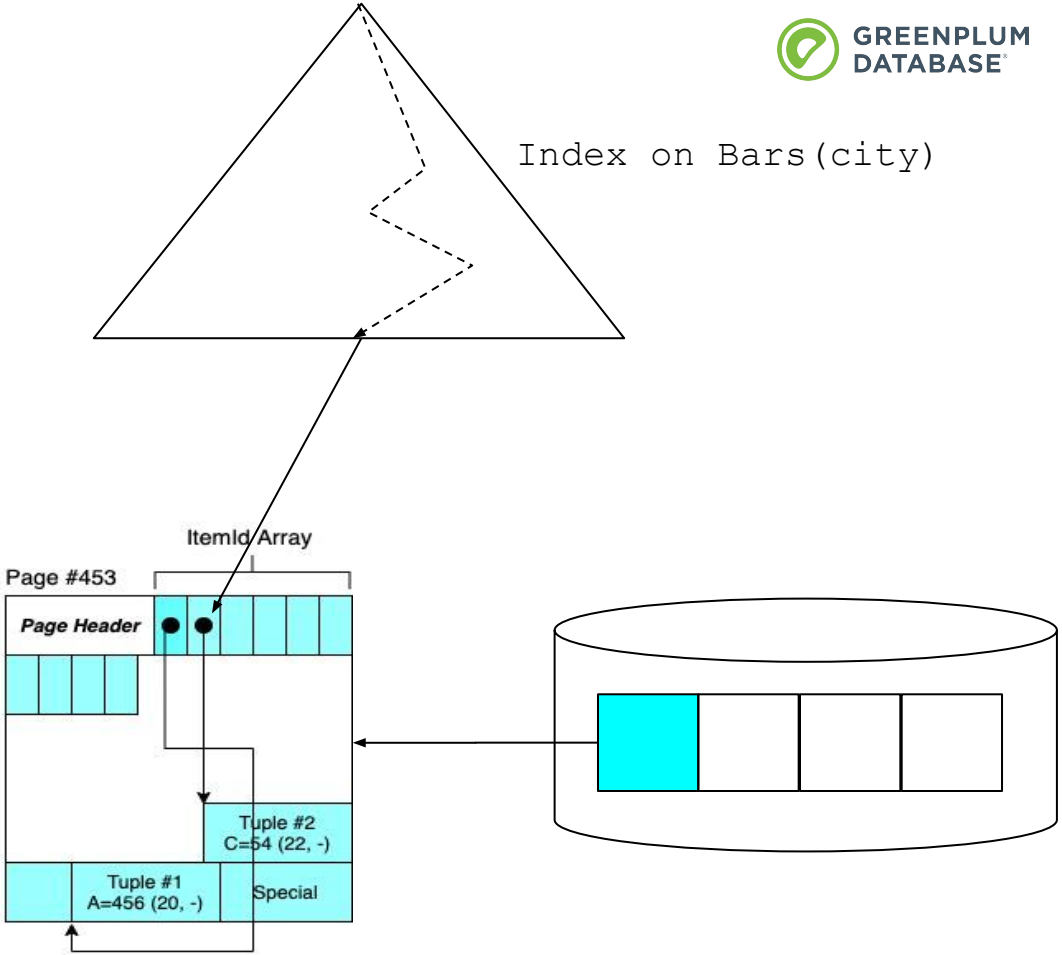
e) VACUUM

- 读和读可以同时进行
- 读和写可以同时进行
- 写和写同一张表同时进行，但是如果涉及同一元组，根据隔离级别：
  - 等待
  - 报错

# 索引

# 为什么需要索引?

```
SELECT COUNT(*)
FROM Bars
WHERE
city = 'San Francisco';
```




索引结构怎么维护？
索引包含哪些类型？
索引怎么并发控制？
索引，扫描，磁盘随机访问？

# 查询执行

# 查询执行流程

SQL
语句
Parser
Query解
析树
Planner
Plan
执行计划
Executor
Results
查询结果

# 查询计划与执行:迭代器方式执行

```
SELECT s.beer, s.price
FROM Bars b, Sells s
WHERE b.name = s.bar
AND b.city = 'San Francisco'
```

- Bars随机分布
- Sells表按照bar分布

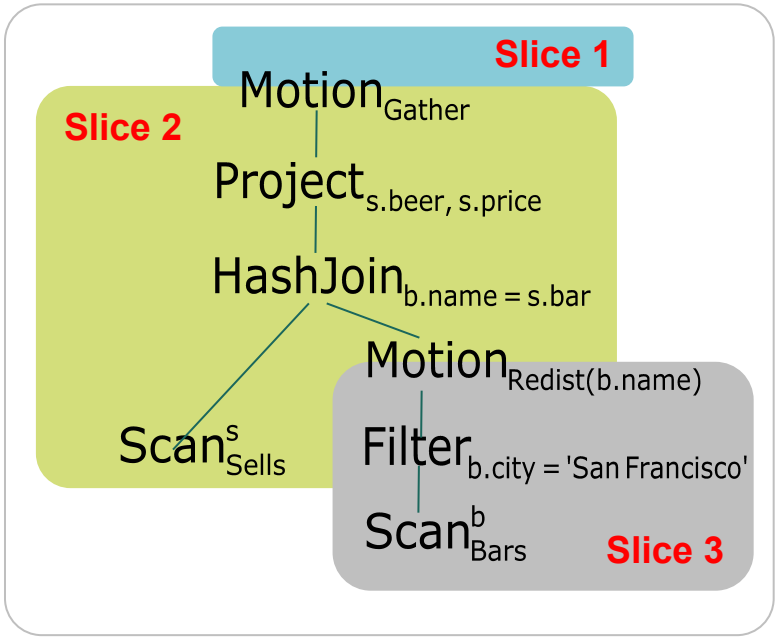
查询优化器会足够聪明选择最小代价的执行计划！

查询优化器如何选择查询计划？

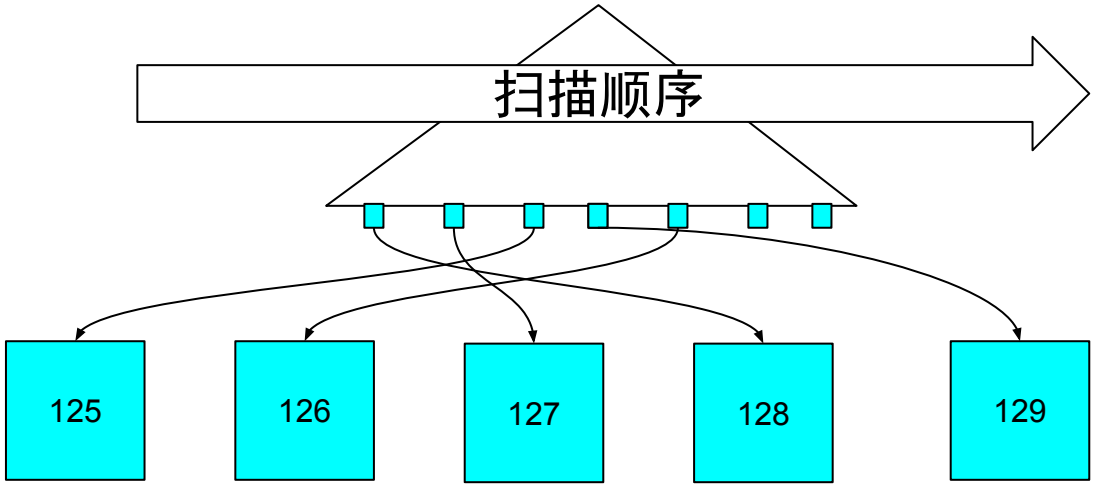
查询计划如何改写

- 好的DBA？
- 优化器？
- 消除外连接？

列存？

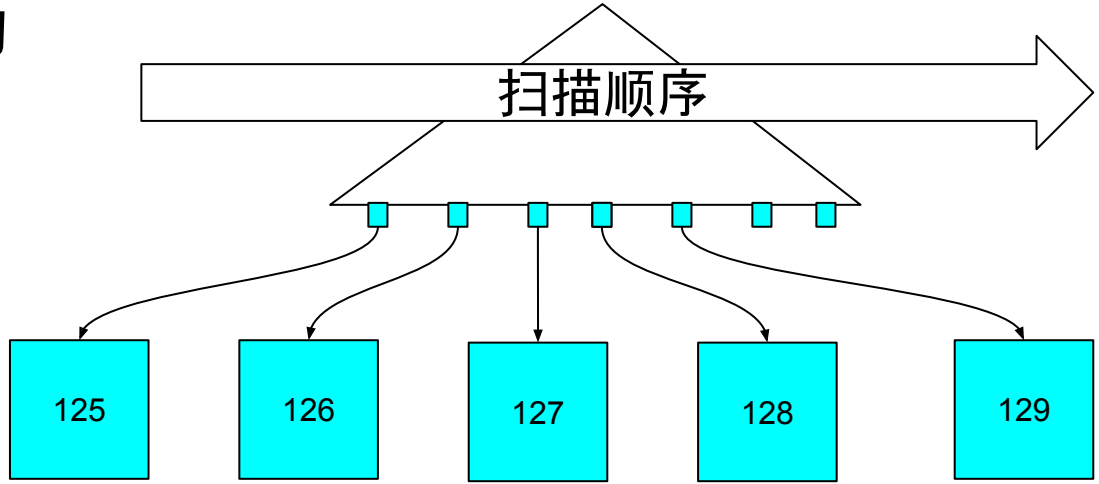


# 索引与磁盘的随机访问？



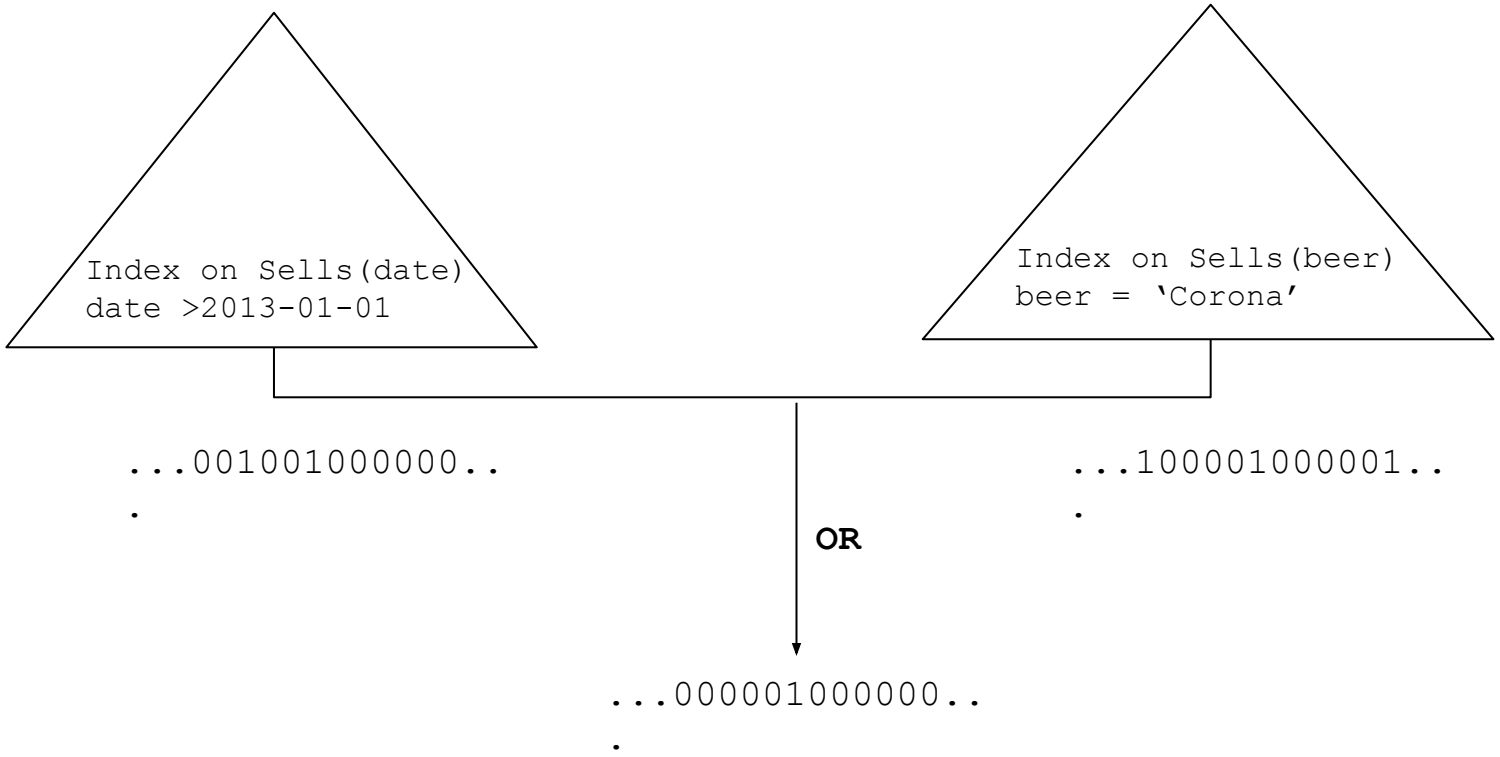
# Greenplum CLUSTER操作

利用OS的prefetch！



# 另一种思路:多条件基于位图扫描

```
SELECT * FROM Sells WHERE date > '2013-01-01' OR beer =
'Corona';
```



# Nested Loop Join

GREENPLUM DATABASE

**Bars**

| name | address | city |
|---|---|---|
| Toronado | 547 Haight St | SF |
| Sunset Beer | 1498 Sunset Blvd | LA |
| The Beer Hall | 1 Pork St | SF |

**Sells**

| bar | price | beer | date |
|---|---|---|---|
| Toronado | 10 | Corona | 2018-07-21 |
| Sunset Beer | 5 | Skol | 2014-10-13 |

基于扫描

```
foreach bar in Bars:
    foreach sell in Sells:
        if bar.name == sell.bar:
            return tuple
```

基于索引

```
foreach sell in Sells:
    if lookup(sell.bar,
              index(bar, name)):
        return tuple
```

# Merge Join：第一阶段排序

**Bars**

| name | address | city |
|---|---|---|
| Toronado | 547 Haight St | SF |
| Sunset Beer | 1498 Sunset Blvd | LA |
| The Beer Hall | 1 Pork St | SF |

排序！

**Sells**

| bar | price | beer | date |
|---|---|---|---|
| Toronado | 10 | Corona | 2018-07-21 |
| Sunset Beer | 5 | Skol | 2014-10-13 |

排序！

内存空间不够？

- 需要使用外存空间存放中间结果
- 基于外排序
  - 如何减少文件扫描次数？
- 如何加快比较速度

# Merge Join：第二阶段归并

**Bars**

| name | address | city |
|---|---|---|
| Sunset Beer | 1498 Sunset Blvd | LA |
| The Beer Hall | 1 Pork St | SF |
| Toronado | 547 Haight St | SF |

**Sells**

| bar | price | beer | date |
|---|---|---|---|
| Sunset Beer | 5 | Skol | 2014-10-13 |
| Toronado | 10 | Corona | 2018-07-21 |

```
p_bar <- frist_element(Sorted(Bars)by
id)
p_sell <- first_element(Sorted(Sells)
by bar)

while p_bar && p_sell:
    if p_bar->name == p_sell->bar:
        process_equal_condition
    else if bar.id < sell.bar:
       p_bar++
    else
       p_sell++
```

# Hash Join

**Bars**

| id | name | address | city |
|---|---|---|---|
| 1 | Toronado | 547 Haight St | SF |
| 2 | The Beer Hall | 1 Pork St | SF |
| 3 | Sunset Beer | 1498 Sunset Blvd | LA |

**Sells**

| bar | price | beer | date |
|---|---|---|---|
| 1 | 10 | 'Corona' | 2018-07-21 |
| 3 | 5 | 'Skol' | 2014-10-13 |

```
hasht<- Hash(Sells)by bar

for each bar in Bars:
    if bar.id in hasht:
        return tuple
```

Hash表过大不能放到内存？需要将哪些元组存放到外存？

Outter待匹配的元组发现需要匹配的Hash表的元组在外存怎么处理？

# 事务与日志

# 原子性：经典问题从A账号向B账户转账100元

```
BEGIN;
A = A - 100;
B = B + 100;
COMMIT;
```

Greenplum通过事务保证操作的原子性

- 事务的状态不会因为宕机而丢失
- 元组的可见性通过事务状态检查
- 分布式事务与各节点Postgres本地事务间的对应关系

# 事务隔离性

T1

```
BEGIN;
A = A - 100;
B = B + 100;
COMMIT;
```

T2

```
BEGIN;
A = A * 1.02;
B = B * 1.02;
COMMIT;
```

# 事务隔离性:错误执行序列

| T1 | T2 |
| --- | --- |
| BEGIN; | |
| A = A - 100; | |
| | BEGIN; |
| | A = A * 1.02; |
| | B = B * 1.02; |
| | COMMIT; |
| B = B + 100; | |
| COMMIT; | |

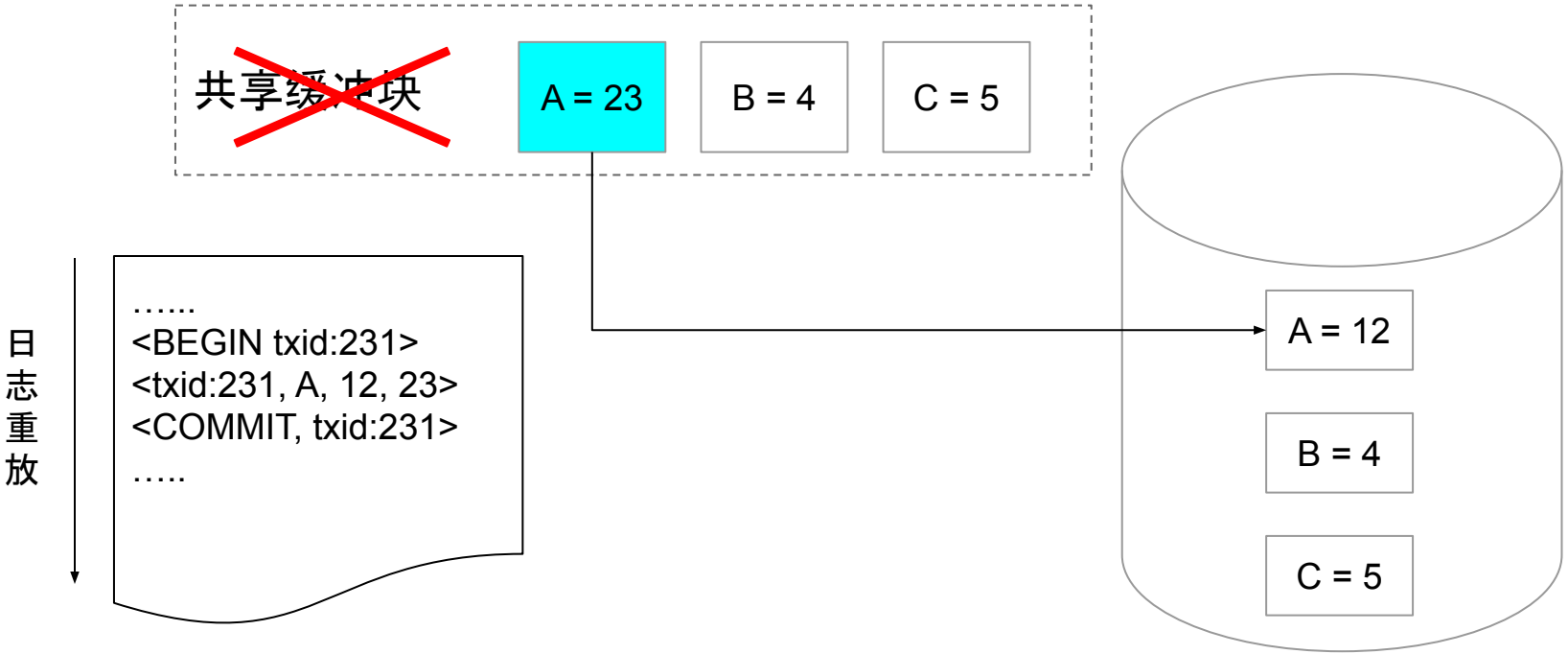
A初始值1000
B初始值1000
A + B = 2000

A = 918
B = 1120
A + B = 2038?
期望结果
2000 * 1.02 = 2040

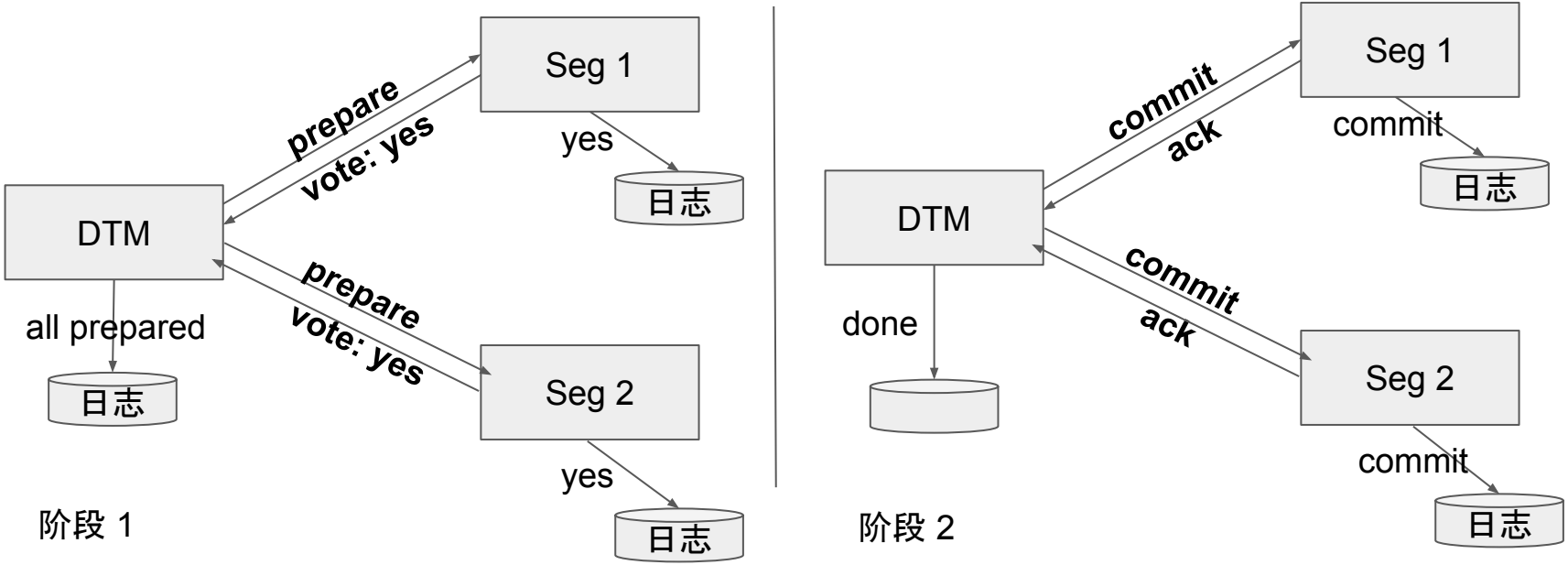
为什么少了2块钱？

Greenplum通过MVCC加并发控制实现隔离级别

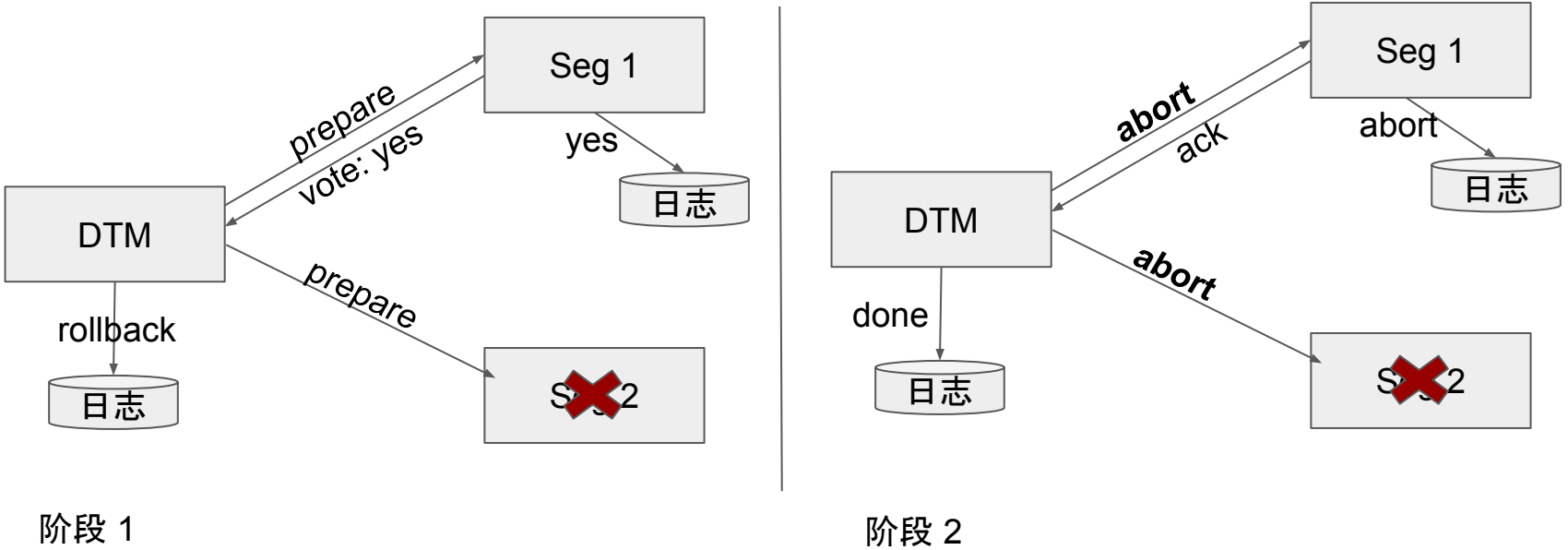
# 事务持久性


共享缓冲块
A = 23
B = 4
C = 5
日志重放
……
<BEGIN txid:231>
<txid:231, A, 12, 23>
<COMMIT, txid:231>
…..
A = 12
B = 4
C = 5

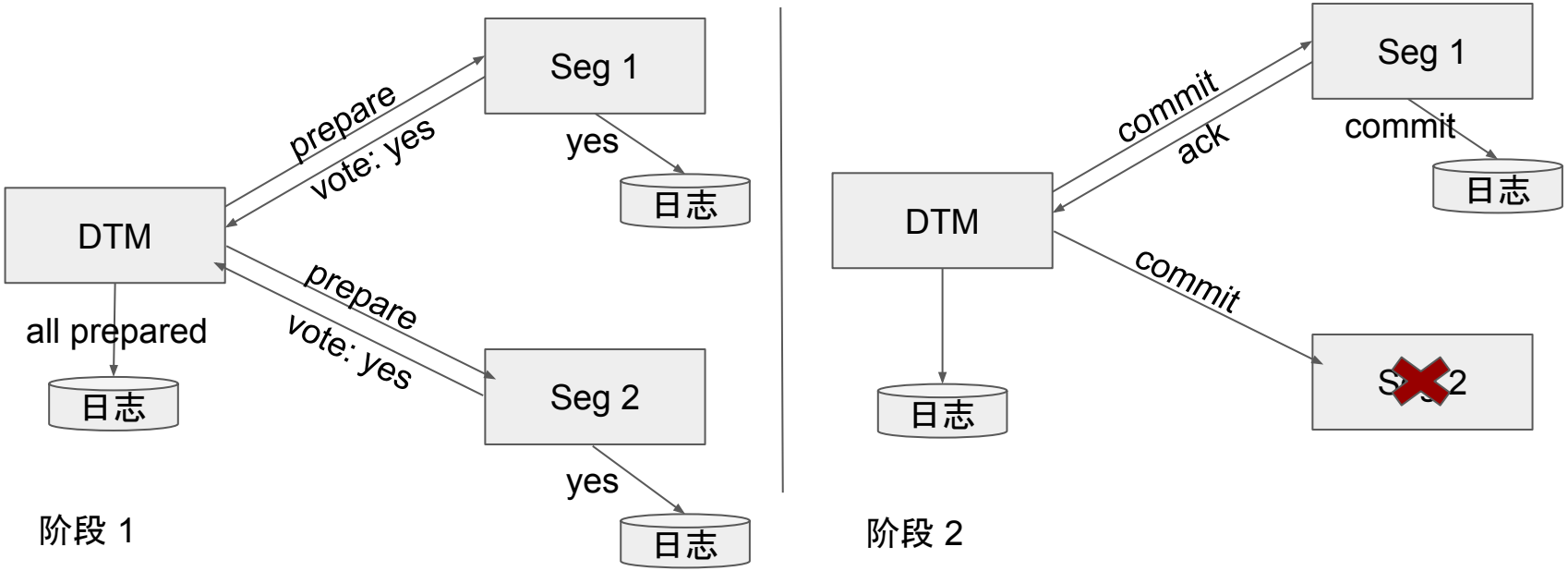
# 跨节点事务:两阶段提交


DTM
Seg 1
Seg 2
prepare
vote: yes
prepare
vote: yes
yes
yes
all prepared
日志
日志
日志
阶段 1
DTM
Seg 1
Seg 2
commit
ack
commit
ack
commit
commit
done
日志
日志
阶段 2

# 两阶段提交:阶段1失败

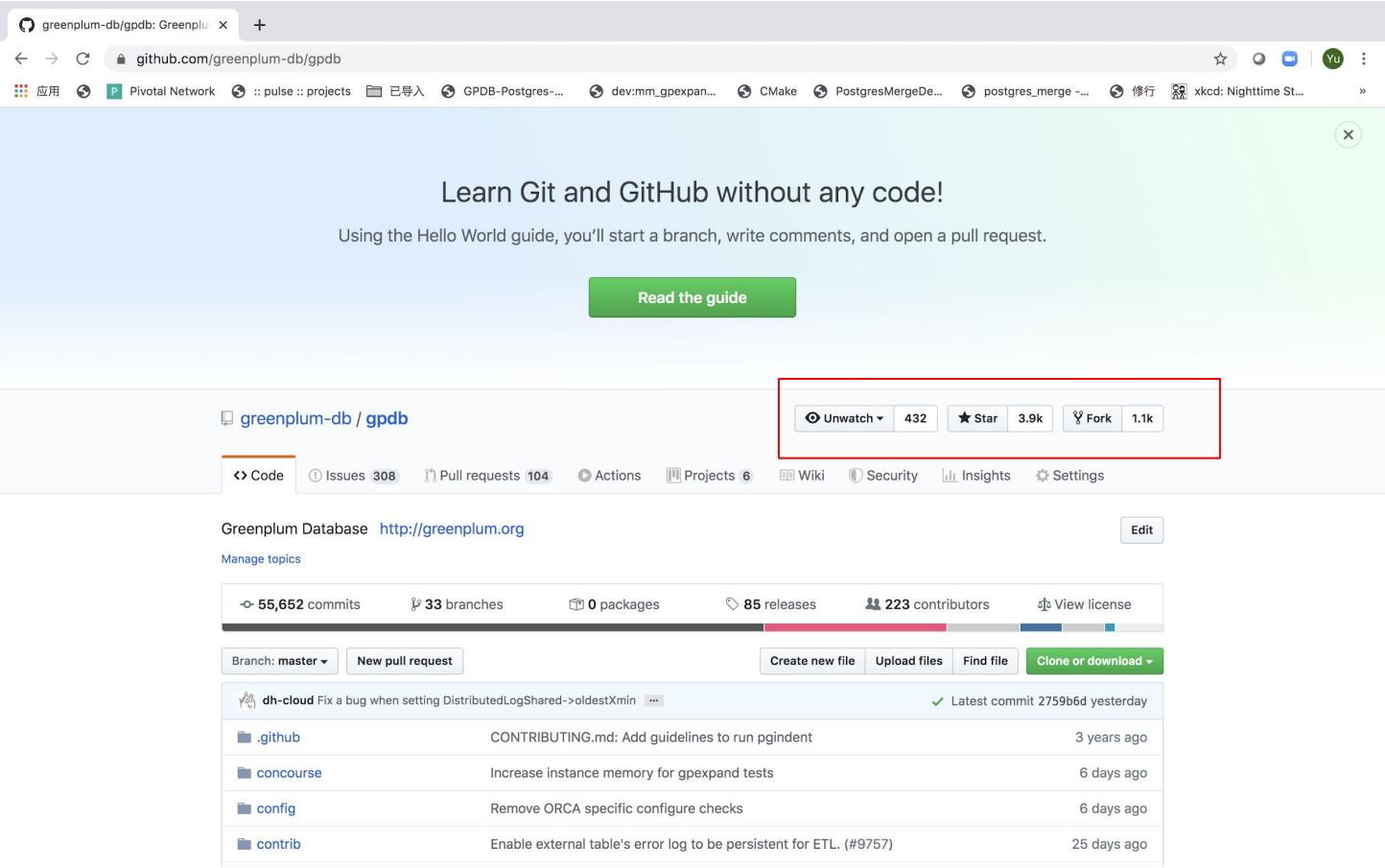

Seg 1
prepare
vote: yes
yes
日志
DTM
prepare
rollback
日志
Seg 2
阶段 1
Seg 1
abort
ack
abort
日志
DTM
abort
done
日志
Seg 2
阶段 2

# 两阶段提交:阶段2失败


DTM
prepare
vote: yes
Seg 1
yes
日志
prepare
vote: yes
Seg 2
yes
日志
all prepared
日志
阶段 1
DTM
commit
ack
Seg 1
commit
日志
commit
日志
阶段 2

# 准备工作

# 从源代码开始：下载编译Greenplum源代码



# 源代码路径

```
|____access
|____bootstrap
|____catalog
|____cdb
|____executor
|____foreign
|____fts
|____libpq
|____main
|____nodes
|____optimizer
|____parser
|____postmaster
|____replication
|____rewrite
|____utils
```

```
cscope -RI .
```

GREENPLUM DATABASE®

## 扫码加入Greenplum技术讨论群

微信群：gp_assistant　　钉钉群：https://dwz.cn/23XPHVOD　　QQ群：99194625

欢迎访问Greenplum问答平台 ask.greenplum.cn