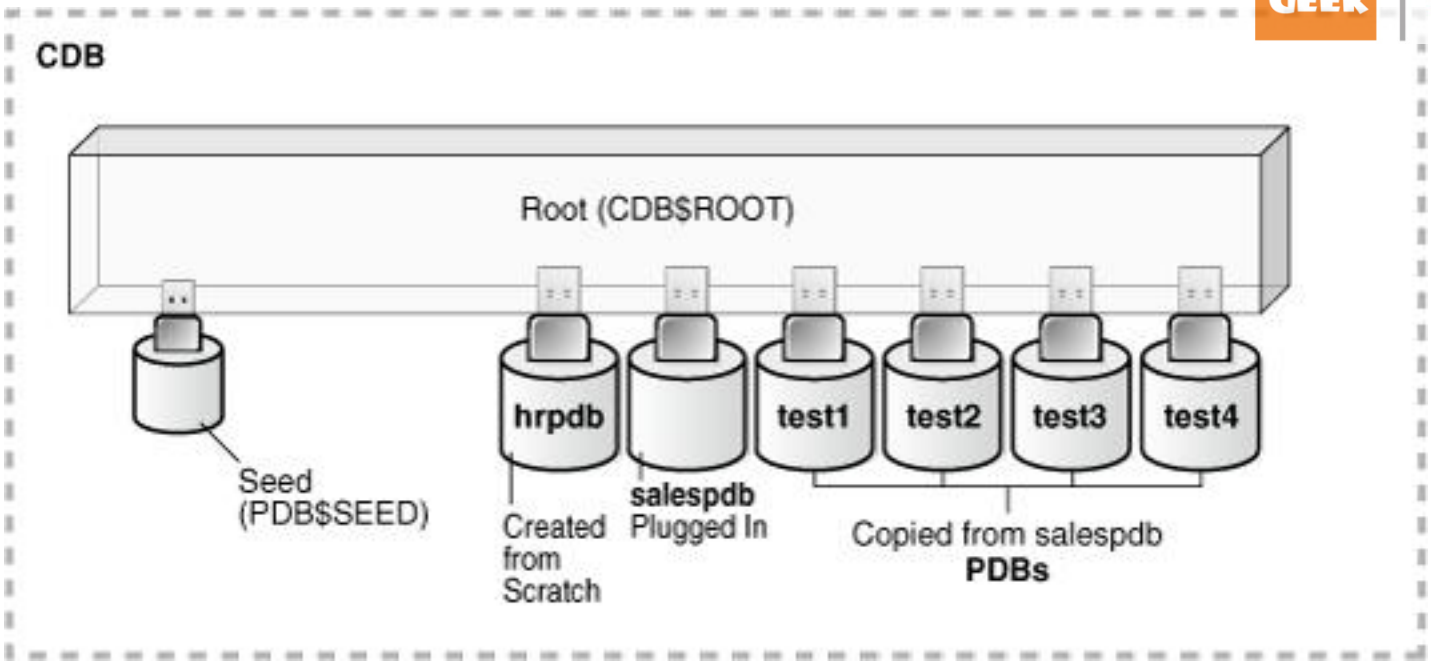
Oracle 最易忽视的特性

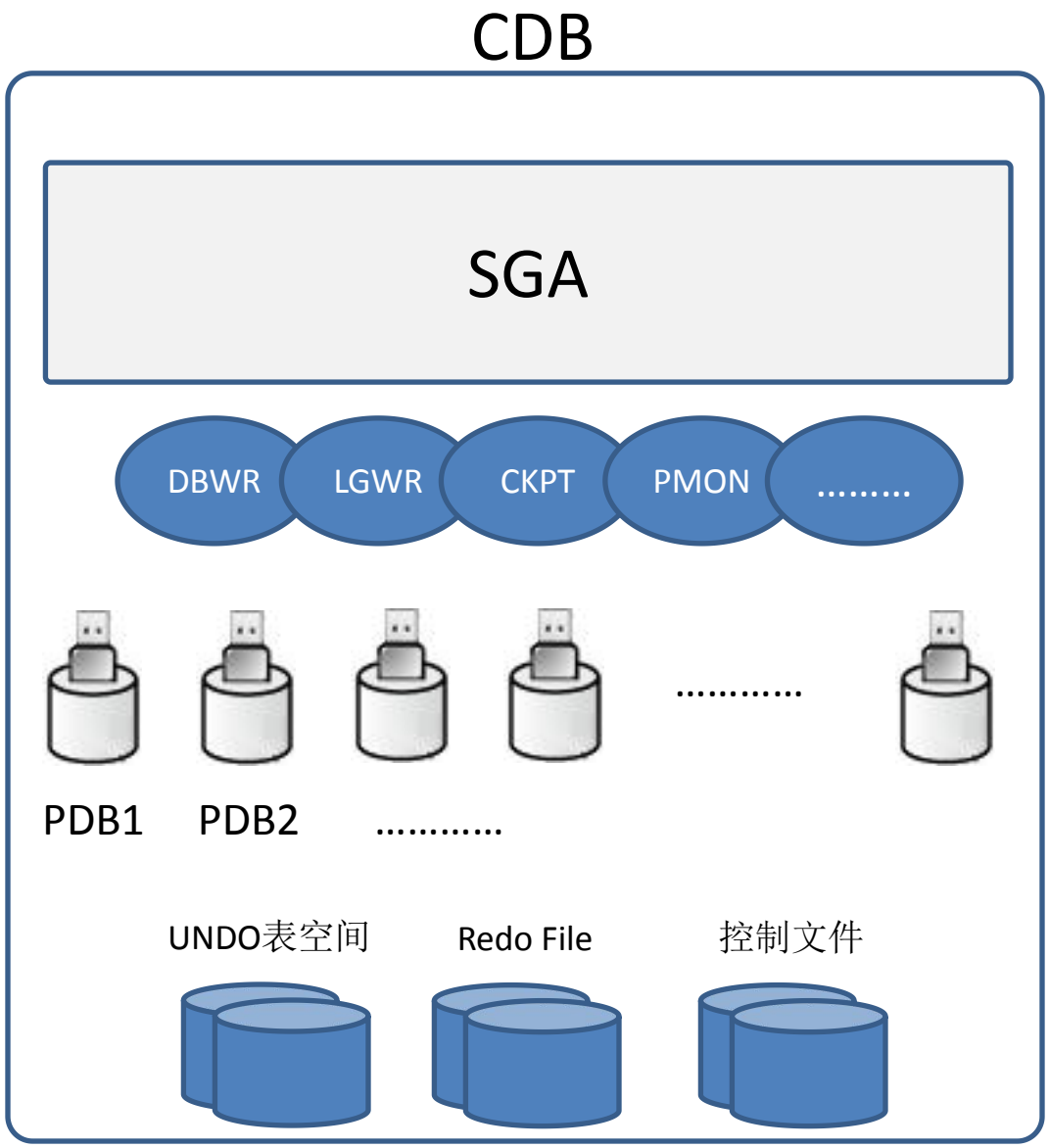
云端的数据库：多租户



Oracle正式的名称是：Multitenant Architecture，多租户。其他的名称：

- 因为CDB中的C是Container，因此又被称为容器数据库。
- 因为PDB的P是Plugged，因此也被称为可插拔数据库。

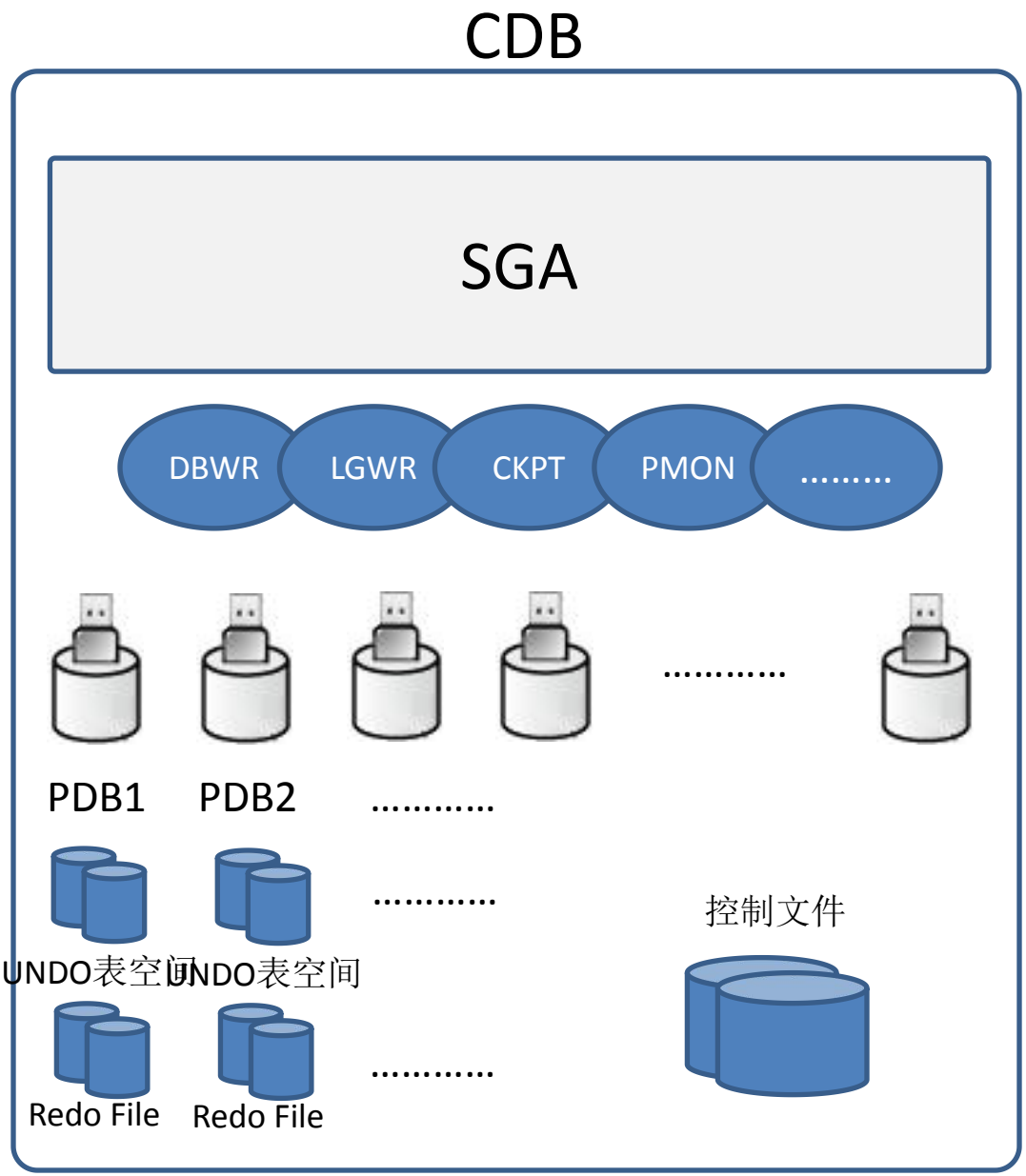
云端的数据库：多租户



简单点说，多租户就是在一个数据库里塞多个数据库。

- 一个CDB可以有N个PDB（至少1个）。
- 所有PDB共享同一个SGA，和一系列的后台进程。
- 每个PDB都是一个相对独立的数据库。有独立的SYSTEM和数据文件。 但共享控制文件和Redo文件和UNDO表空间。
- 当CDB是OPEN状态时，它里面的每个PDB可以是以下三种状态：
  - MOUNT
  - OPEN
  - OPEN Read Only

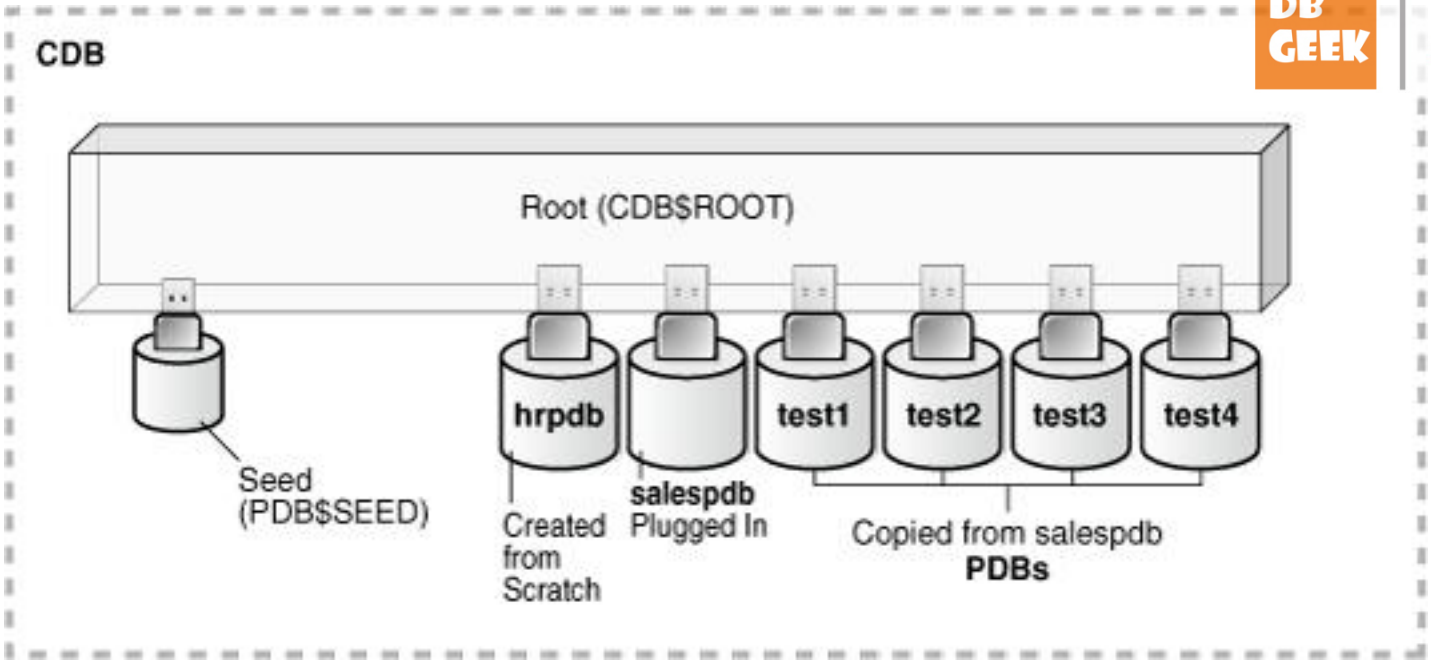
云端的数据库：多租户



简单点说，多租户就是在一个数据库里塞多个数据库。

- 一个CDB可以有N个PDB（至少1个）。
- 所有PDB共享同一个SGA，和一系列的后台进程。
- 每个PDB都是一个相对独立的数据库。有独立的SYSTEM和数据文件。 但共享控制文件和Redo文件和UNDO表空间。
- 当CDB是OPEN状态时，它里面的每个PDB可以是以下三种状态：
  - MOUNT
  - OPEN
  - OPEN Read Only

云端的数据库：多租户

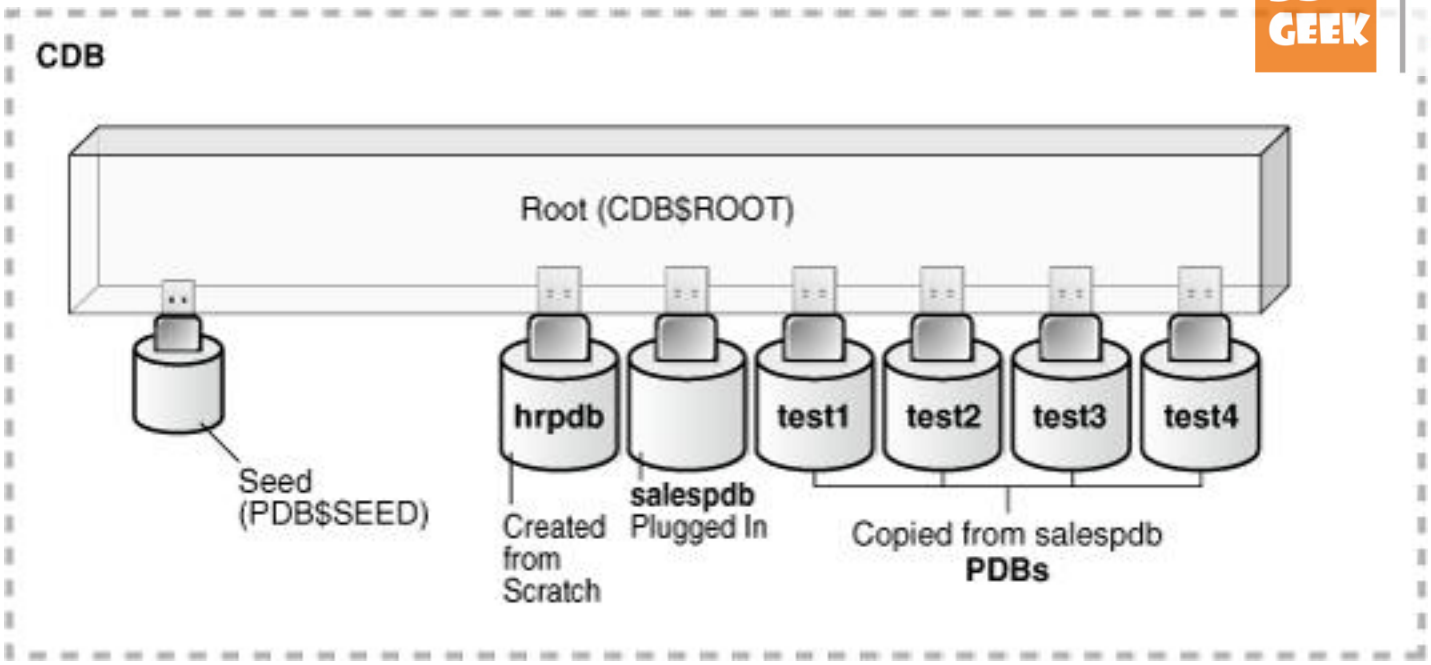


CDB$ROOT是CDB中的根PDB。它也是一个独立数据库。它的数据字典中包含其他PDB的信息。所以拔出某个PDB，可以说是从一个CDB的CDB$ROOT中拔出。而插入到另一个CDB，当然就是插入到另一个CDB中的CDB$ROOT了。CDB$ROOT中的配置，就是所有PDB的默认配置。修改CDB$ROOT中的参数、配置，就是在整个CDB级修改。

保存所有PDB信息的数据字典视图是dba_pdbs：

```
SQL> col pdb_name for a20
SQL> select pdb_id, pdb_name, dbid, con_uid, guid, status, con_id from dba_pdbs;

PDB_ID PDB_NAME            DBID       CON_UID    GUID                             STATUS       CON_ID
---------- -------------------- ---------- ---------- -------------------------------- ------------ ----------
--
         2 PDB$SEED         3752187096 3752187096 257DA1860ED60599E054000C2980BC24 NORMAL            2
         3 PDBTEST4         2804740952 1883059466 258198A7A89F0D75E054000C2980BC24 NEW               3
```

云端的数据库：多租户



PDB$SEED是种子PDB，它只能以只读模式打开，不能手动关闭，只能随CDB$ROOT一起打开或关闭。创建新的PDB时，就是Clone PDB$SEED为一个新的PDB。

它只包含SYSTEM表空间和SYSAUX表空间，而且我们不能增减它的表空间，也不能在它里面创建表、索引等对象。

PDB$SEED只用来作Clone操作时的源，无法对它进行任何操作。

云端的数据库 ---- 多租户：创建CDB

- 使用 DBCA
- 手动创建，步骤同非CDB:
  1. 准备参数文件
  2. Nomount启动实例
  3. 执行建库命令
  4. 运行初始化脚本

云端的数据库 ---- 多租户：创建CDB


```
CREATE DATABASE CDBV4
MAXINSTANCES 2
MAXLOGHISTORY 1
MAXLOGFILES 16
MAXLOGMEMBERS 3
MAXDATAFILES 1024
DATAFILE '/export/home/oradb/oradata/CDBV4/system01.dbf' SIZE 300M REUSE AUTOEXTEND ON NEXT  10240K MAXSIZE UNLIMITED
EXTENT MANAGEMENT LOCAL
SYSAUX DATAFILE '/export/home/oradb/oradata/CDBV4/sysaux01.dbf' SIZE 200M REUSE AUTOEXTEND ON NEXT  10240K MAXSIZE UNLIMITED
SMALLFILE DEFAULT TEMPORARY TABLESPACE TEMP TEMPFILE '/export/home/oradb/oradata/CDBV4/temp01.dbf' SIZE 20M REUSE
AUTOEXTEND ON NEXT  640K MAXSIZE UNLIMITED
SMALLFILE UNDO TABLESPACE "UNDOTBS1" DATAFILE  '/export/home/oradb/oradata/CDBV4/undotbs01.dbf' SIZE 100M REUSE AUTOEXTEND
ON NEXT  5120K MAXSIZE UNLIMITED
CHARACTER SET WE8MSWIN1252
NATIONAL CHARACTER SET AL16UTF16
LOGFILE GROUP 1 ('/export/home/oradb/oradata/CDBV4/redo01.log') SIZE 100M,
GROUP 2 ('/export/home/oradb/oradata/CDBV4/redo02.log') SIZE 100M,
GROUP 3 ('/export/home/oradb/oradata/CDBV4/redo03.log') SIZE 100M
USER SYS IDENTIFIED BY "Oracle1" USER SYSTEM IDENTIFIED BY "Oracle1"
enable pluggable database
seed file_name_convert=('/export/home/oradb/oradata/CDBV4/system01.dbf','/export/home/oradb/oradata/CDBV4/pdbseed/system01.dbf',
'/export/home/oradb/oradata/CDBV4/sysaux01.dbf','/export/home/oradb/oradata/CDBV4/pdbseed/sysaux01.dbf',
‘/export/home/oradb/oradata/CDBV4/temp01.dbf’,‘/export/home/oradb/oradata/CDBV4/pdbseed/temp01.dbf’);
```

注意以下选项：

enable pluggable database：启用CDB。

seed file_name_convert：这是两个选项，SEED和FILE_NAME_CONVERT。建库命令中所创建的SYSTEM、SYSAUX等等表空间、数据文件都属于根PDB：CDB$ROOT，”seed”选项是声明创建PDB$SEED。Oracle会从CDB$ROOT Clone SYSTEM表空间和SYSAUX表空间来创建PDB$SEED，file_name_convert选项指定Clone表空间时新数据文件的位置。此选择的作用和用法类似Standby中的file_name_convert。

云端的数据库 ---- 多租户：创建新的PDB

创建新的PDB：

```
bash-3.2$ sqlplus / as sysdba
SQL> CREATE PLUGGABLE DATABASE pdbtest1
 ADMIN USER dba1 IDENTIFIED BY a
 file_name_convert=
('/export/home/oradb/oradata/CDBV3/pdbseed/',
'/export/home/oradb/oradata/CDBV3/pdbtest1/');
```

或

```
SQL> CREATE PLUGGABLE DATABASE pdbtest2 FROM pdbtest1
file_name_convert=
('/export/home/oradb/oradata/CDBV3/pdbtest1/',
'/export/home/oradb/oradata/CDBV3/pdbtest2/');
```

云端的数据库 ---- 多租户：连接PDB

连接到PDB的两种方式:

- 先进入cdb$root，再alter session set container=....，进入指定PDB（如需要使用sys连接某一个PDB，必须使用此方式）

```
bash-3.2$ sqlplus / as sysdba
SQL> show con_name

CON_NAME
------------------------------
CDB$ROOT
SQL> alter session set container=pdbtest2;

Session altered.

SQL> show con_name

CON_NAME
------------------------------
PDBTEST2
```

- conn .../...@TNS_NAME，使用tns_name直接进入指定的PDB

```
bash-3.2$ sqlplus system/Oracle1@pdbtest2
Connected to:
SQL> show con_name
CON_NAME
------------------------------
PDBTEST2
```

云端的数据库 ---- 多租户：连接PDB

TNS_NAME的配置：

```
pdbtest2 =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = h3)(PORT = 1521))
  (CONNECT_DATA =
   (SERVER = DEDICATED)
   (SERVICE_NAME = pdbtest2)
  )
 )
```

PMON进程会像监听中注册所有已经OPEN的PDB。

监听状态：

```
bash-3.2$ lsnrctl status
LSNRCTL for Solaris: Version 12.1.0.2.0 - Pr...ion on 25-DEC-2015 16:46:10
STATUS of the LISTENER
------------------------
Services Summary...
Service "pdbtest2" has 1 instance(s).
  Instance "CDBV3", status READY, has 1 handler(s) for this service...
Service "pdbtest4" has 1 instance(s).
  Instance "CDBV3", status READY, has 1 handler(s) for this service...
The command completed successfully
```

数据库service_names参数：

```
SQL> show parameter servi

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
service_names                        string      CDBV3
```

云端的数据库 ---- 多租户：数据字典视图

➢ 数据字典

所有元数据存放在CDB$ROOT的SYSTEM中，PDB SYSTEM中的数据字典表只保存一个指针，指向CDB$ROOT中真正的元数据。AWR数据也是在CDB$ROOT中。
比如某个Oracle自身的存储过程，其代码只在CDB$ROOT的SYSTEM中存储一份，其他PDB的SYSTEM中的指针全都指向CDB$ROOT中的存储过程。这样的目的有两个，一是节省空间，二是升级的时候只需要修改CDB$ROOT的SYSTEM中的元数据即可。

➢ 视图的变化：CDB_ DBA_ ALL_ USER_

增加CDB_系统数据字典视图，可以在全局级别查看对象，所以有时候能看到这样的情况：

```
SQL> col owner for a15
SQL> col table_name for a15
SQL> select owner, table_name from cdb_tables where table_name='T1';

OWNER           TABLE_NAME
--------------- ---------------
SYS             T1
SYS             T1
```

重复的对象。其实它们分属不同的PDP。但在CDB视图中，并无专门的列说明对象属于哪个PDB。
只有当PDB OPEN时，才能在cdb_*视图中查到相关PDB的对象。

对于DBA_ ALL_ USER_系列视图，在哪个PDB中发出查询命令，显示结果就是哪个PDB中信息。

云端的数据库 ---- 多租户：打开与关闭PDB

原来的打开数据库方式，只是打开CDB$ROOT和PDB$SEED：

```
bash-3.2$ sqlplus / as sysdba
Connected to an idle instance.

SQL> startup
ORACLE instance started.

Total System Global Area  629145600 bytes
Fixed Size                3006784 bytes
Variable Size             478154432 bytes
Database Buffers          142606336 bytes
Redo Buffers              5378048 bytes
Database mounted.
Database opened.
SQL> select con_id, name, open_mode from v$pdbs;

    CON_ID NAME                           OPEN_MODE
---------- ------------------------------ ----------
         2 PDB$SEED                       READ ONLY
         3 PDBTEST4                       MOUNTED
         4 PDBTEST2                       MOUNTED
```

云端的数据库 ---- 多租户：打开与关闭PDB

- 打开其他PDB的方式：

alter pluggable database PDB_NAME1[,PDB_NAME2,……] open [READ ONLY FORCE] [UPGRADE [RESTRICTED]]
alter pluggable database PDB_NAME1[,PDB_NAME2,……] close [immediate];
//不加immediate必须等待正在连接PDB的用户退出。注意，没有abort选项。因为close只是把某个PDB从open状态转为mount状态，并不是关闭整个CDB，SGA内存并没有释放，因此不存在abort。

其他的打开PDB的命令：
ALTER PLUGGABLE DATABASE ALL OPEN;
ALTER PLUGGABLE DATABASE ALL CLOSE IMMEDIATE;
ALTER PLUGGABLE DATABASE ALL EXCEPT pdb1 OPEN;
ALTER PLUGGABLE DATABASE ALL EXCEPT pdb1 CLOSE IMMEDIATE;

- 另一种打开PDB的方式：

bash-3.2$ sqlplus / as sysdba
Connected to:
SQL> alter session set container=pdbtest2;

Session altered.

SQL> alter database open;

Database altered.

云端的数据库 ---- 多租户：参数管理

整个CDB只有一个SPFILE参数文件，有些参数在各个PDB中可以设置为不同值。spfile中只保存CDB的参数值（或者说只保存CDB$ROOT的值），猜想各PDB的参数值是保存在CDB的数据字典中。
在某一个pdb中create pfile from spfile，结果文件中将只包含此pdb和CDB不同的参数（也就是单独在此pdb中修改了的参数）。

```
SQL> show con_name
CON_NAME
------------------------------
PDBTEST2

SQL> alter system set open_cursors=1000;
System altered.
SQL> show parameter open_cur
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
open_cursors                         integer     1000
```

转到CDB中：

```
SQL> show con_name
CON_NAME
------------------------------
CDB$ROOT
SQL>
SQL> show parameter open_cur
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
open_cursors                         integer     300
```

转到PDBTEST2中：

```
SQL> create pfile='/export/home/oracle/pdbtest2.ora' from spfile;
File created.

bash-3.2$ cat /export/home/oracle/pdbtest2.ora
*.open_cursors=1000
```

云端的数据库 ---- 多租户：参数管理

有些参数不能在PDB级别修改，如内存相关参数：

```
SQL> alter system set memory_target=500m;
alter system set memory_target=500m
*
ERROR at line 1:
ORA-65040: operation not allowed from within a pluggable database
```

所有PDB会共享SGA，无法限制某个PDB的内存使用情况。因此，不必要也不能修改某个PDB的内存参数。

内存参数只能在CDB$ROOT中修改（或者说，只能在CDB级别修改）：

```
SQL> show con_name

CON_NAME
------------------------------
CDB$ROOT
SQL> alter system set memory_target=500m;

System altered.
```

云端的数据库 ---- 多租户：表空间管理

可以在PDB中创建表空间，每个PDB中的数据文件、表空间都是独立的。多个PDB可以使用同一表空间名。查询DBA_视图，只能看到当前PDB的信息：

在根PDB CDB$ROOT：

```
SQL> select tablespace_name, file_name from dba_data_files;

TABLESPACE_NAME                FILE_NAME
------------------------------ ------------------------------------------------------------
SYSTEM                          /export/home/oradb/oradata/CDBV3/system01.dbf
SYSAUX                          /export/home/oradb/oradata/CDBV3/sysaux01.dbf
USERS                          /export/home/oradb/oradata/CDBV3/users01.dbf
UNDOTBS2                        /export/home/oradb/oradata/CDBV3/undotbs2_1.dbf
```

在PDBTEST2：

```
SQL> select tablespace_name, file_name from dba_data_files;
TABLESPACE_NAME                FILE_NAME
------------------------------ ------------------------------------------------------------
SYSTEM                          /export/home/oradb/oradata/CDBV3/pdbtest2/system01.dbf
SYSAUX                          /export/home/oradb/oradata/CDBV3/pdbtest2/sysaux01.dbf
USERS                          /export/home/oradb/oradata/CDBV3/pdbtest2/users01.dbf
```

在PDBTEST4：

```
SQL> select tablespace_name, file_name from dba_data_files;
TABLESPACE_NAME                FILE_NAME
------------------------------ ------------------------------------------------------------
SYSTEM                          /export/home/oradb/oradata/CDBV3/pdbtest4/system01.dbf
SYSAUX                          /export/home/oradb/oradata/CDBV3/pdbtest4/sysaux01.dbf
```

云端的数据库 ---- 多租户：表空间管理

但如果在CDB$ROOT中查询V$视图，可以看到所有PDB中表空间的信息：

```
SQL> select ts#,name,con_id from v$tablespace order by con_id;

     TS# NAME                           CON_ID
---------- ------------------------------ ----------
         0 SYSTEM                          1
         1 SYSAUX                          1
         3 TEMP                            1
         4 USERS                           1
         5 UNDOTBS2                        1
         0 SYSTEM                          2
         1 SYSAUX                          2
         2 TEMP                            2
         1 SYSAUX                          3
         0 SYSTEM                          3
         2 TEMP                            3
         1 SYSAUX                          4
         2 TEMP                            4
         3 USERS                           4
         0 SYSTEM                          4
15 rows selected.
```

因此V$视图中会增加CON_ID列，DBA_视图中则没有此列。

这是因为每个PDB自己的SYSTEM表空间中保存自己的数据字典，因此DBA_视图只有某个PDB的信息。而V$视图中的信息则来自于控制文件，控制文件是所有PDB共享的。

云端的数据库 ---- 多租户：用户管理

用户分两类：Common用户和Local用户。Common用户一旦创建，将在每个PDB中都存在，可以说Common是全局用户。Oracle的内部用户都是Common用户，比如SYS和SYSTEM。DBA自己创建的Common用户，必须以c##或C##开头。Local用户只存在某一个PDB内。

在cdb$root中创建用户，同时以c##开头，将是common用户。在pdb中创建将是Local的，在cdb$root中创建用户，不以c##开头，也是Local的。

Public也是一个Comman用户， 在每个pdb中可以授于Public不同的权限。

云端的数据库 ---- 多租户：角色管理

- 角色也有common和local之分。Local的角色只能在某个PDB中使用。创建角色的规则同User。

- 同一Common role，在不同pdb中可以有不同的权限。

云端的数据库 ---- 多租户：权限与角色

- Common用户在不同PDB中可以有不同的权限。
- 权限相关命令有两个级别：当前PDB和所有PDB，使用选项CONTAINER=current|ALL控制。
  “CONTAINER=all”只能针对common用户使用。
- 命令使用示例：

在CDB$ROOT中创建c##u2用户：

```
SQL> create user c##u2 identified by a;

User created.

SQL> grant create session to c##u2 CONTAINER=all;

Grant succeeded.
```

在其他PDB中创建Common用户将报出错误：

```
SQL> create user c##u3 identified by a;

create user c##u3 identified by a
            *
ERROR at line 1:
ORA-65094: invalid local user or role name
```

云端的数据库 ---- 多租户：PDB的插拔

1). 拔出

alter pluggable database pdbtest1 close;

alter pluggable database pdbtest1 unplug into '/export/home/oracle/pdb/CDBV1.xml';

XML文件中包含此PDB的如下信息：

- 表空间
- 数据文件
- Local用户和Local角色
- 针对此PDB修改过的参数
- 少量性能资料（CPU使用量、软、硬解析次数、DB Block Changes等等）

2). 备分PDB相关文件

rman target /

BACKUP FOR TRANSPORT AS COMPRESSED BACKUPSET PLUGGABLE DATABASE 'PDBTEST1' FORMAT '/export/home/oracle/pdb/PDBTEST_%U.pdb';

3). 拷贝XML文件和RMAN备份集文件到目标库

云端的数据库 ---- 多租户：PDB的插拔

4). 在目标库上还原RAMN备份集：

```
rman target /
RESTORE FOREIGN DATAFILE
  12 format '/export/home/oradb/oradata/CDBV3/pdbtest4/sysaux01.dbf',
  11 format '/export/home/oradb/oradata/CDBV3/pdbtest4/system01.dbf'
 FROM BACKUPSET '/export/home/oracle/pdb/PDBTEST_01qnvmlg_1_1.pdb' ;
```

和之前版本的还原命令不同的是，不要求控制文件中有备份集信息，可以使用选项“FROM BACKUPSET ”指定备份集位置。

在目标库中运行下面的PL/SQL，可以检查XML对应的PDB是否兼容当前CDB：

```
set serveroutput on
DECLARE
  l_result BOOLEAN;
BEGIN
  l_result := DBMS_PDB.check_plug_compatibility(
    pdb_descr_file => '/export/home/oracle/pdb/CDBV3_test1.xml',
    pdb_name       => 'pdbtest1');
  IF l_result THEN
    DBMS_OUTPUT.PUT_LINE('compatible');
  ELSE
    DBMS_OUTPUT.PUT_LINE('incompatible');
  END IF;
END;
/
```

云端的数据库 ---- 多租户：PDB的插拔

5). 将PDB插入到目标数据库：

```
create pluggable database pdbtest3 using '/export/home/oracle/pdb/CDBV3_test1.xml'
source_file_name_convert=
   ('/export/home/oradb/oradata/CDBV3/pdbtest1/','/export/home/oradb/oradata/CDBV3/pdbtest3/')
 file_name_convert=NONE  NOCOPY tempfile reuse ;
```

6). 在目标库打开新的pdb

```
alter pluggable database pdbtest1 open;
```

7). 删除源数据库的PDB（此步不是必须的）

```
drop pluggable database pdbtest1;
```

云端的数据库 ---- 多租户： NoCDB转换为CDB

1) 以受限模式打开数据库：

```
shutdown immediate;
startup restrict mount exclusive;
alter database open read only;
```

2) 生成XML元数据描述文件：

```
begin
dbms_pdb.describe(PDB_DESCR_FILE => '/u02/noncdb/orcl.xml');
end;
/
```

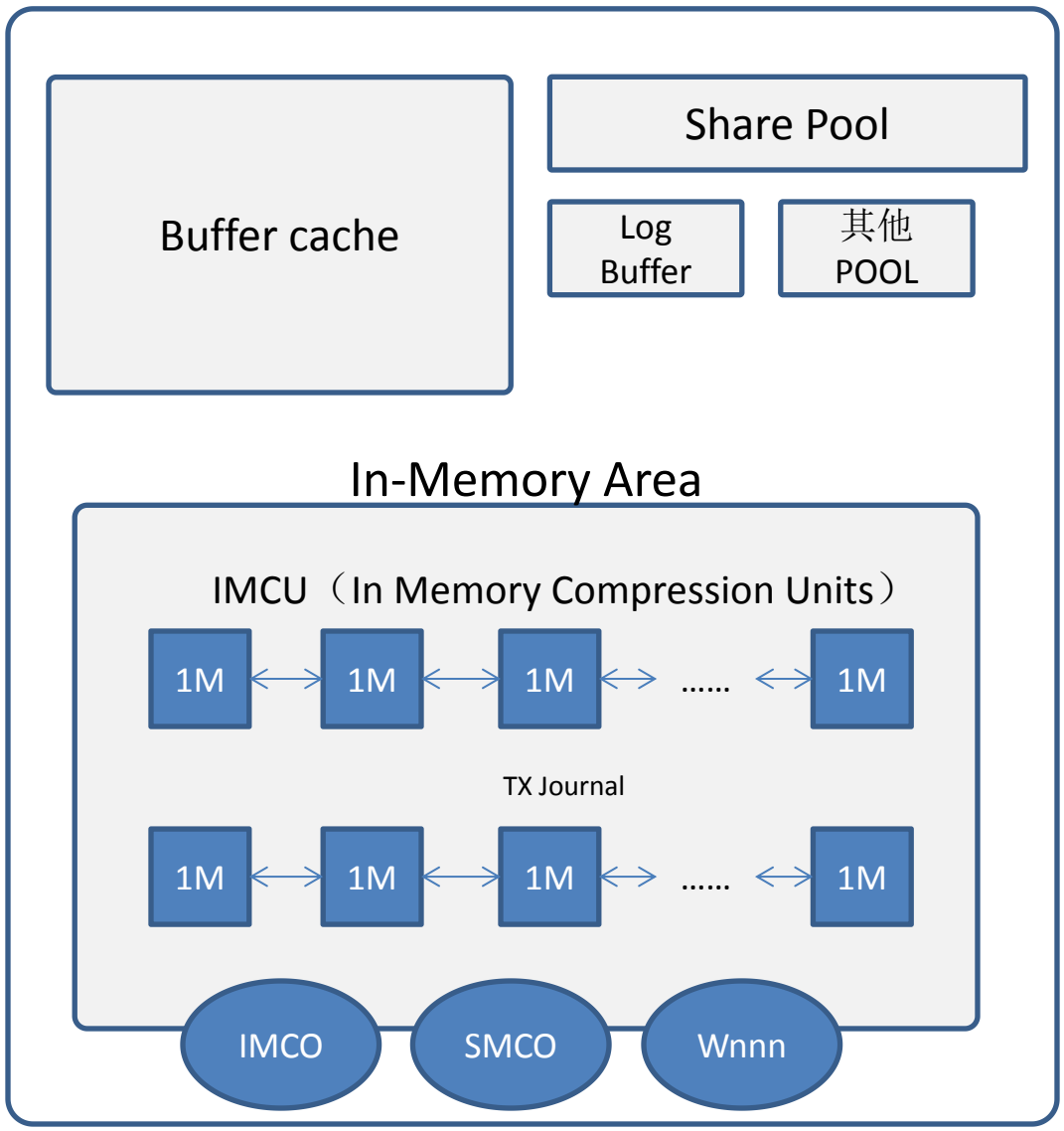
3) 创建PDB

```
create pluggable database orcl using '/u02/noncdb/orcl.xml' copy;
```

4) 执行noncdb_to_pdb脚本

```
alter session set container=orcl;
@?/rdbms/admin/noncdb_to_pdb.sql
```

In-Memory Option：概述



每个IMCU 1M大小左右，具体大小由Oracle自己决定。

表的数据以列为单位保存在IMCU中。一个表至少占一个IMCU。

我们不能指定只保存表的一部分列，Oracle把自动把表的所有列缓存在In-memory Area中。

DML操作将同时针对表相关数据和缓存在IMCU中的数据。Tx Journal中记录IMCU的前映象，以提供未提交事务的一致读。

IMCO：调度进程，负责IM内存数据的加载和重载的计划调度，它每两分钟会触发一次。

SMCO/Wnnn：该两个后台进程是实际去执行IM对象加载和重载工作的进程，

In-Memory Option：开启

- 设置In Memory Area大小：

```
SQL> alter system set inmemory_size=100m scope=spfile;

System altered.
```

- 重启数据库：

```
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.

Total System Global Area  629145600 bytes
Fixed Size                3006784 bytes
Variable Size             452988608 bytes
Database Buffers          62914560 bytes
Redo Buffers              5378048 bytes
In-Memory Area            104857600 bytes
Database mounted.
Database opened.
SQL>
```

In-Memory Option：将表加载到内存中

- 加载表到IM中：

  SQL> alter table t2 inmemory;

  Table altered.

- 取消In-Memeroy Area中的表：

  SQL> alter table t2 no inmemory;

  Table altered.

- 性能资料对比：

In-Memory Option：将表加载到内存中


```
SQL> select count(*) from t2;

Execution Plan
----------------------------------------------------------
Plan hash value: 3321871023

-------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |     1 |   515   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |      |     1 |            |          |
|   2 |   TABLE ACCESS FULL| T2   |   461K|   515   (1)| 00:00:01 |
-------------------------------------------------------------------------

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
       1796  consistent gets
       1788  physical reads
          0  redo size
        542  bytes sent via SQL*Net to client
        551  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

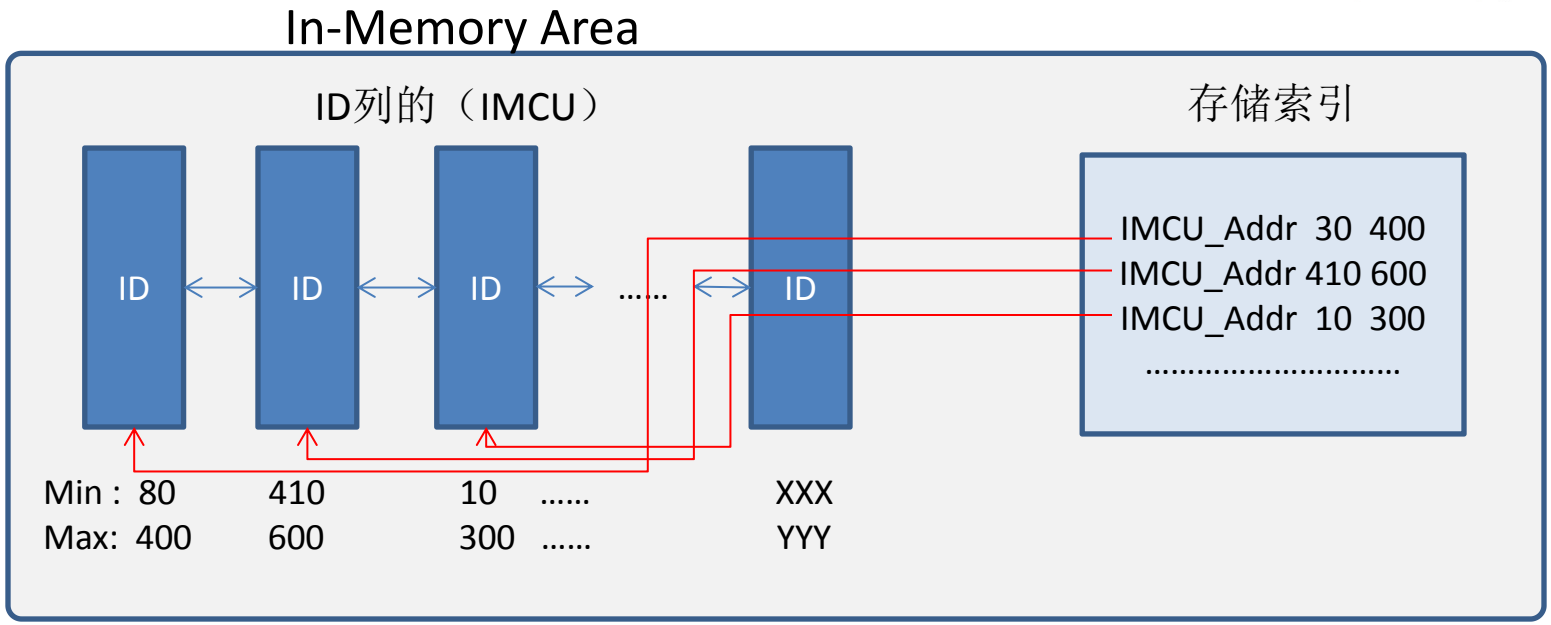
```
SQL> select count(*) from t2;

Execution Plan
----------------------------------------------------------
Plan hash value: 3321871023

-----------------------------------------------------------------------------------
| Id  | Operation                  | Name | Rows  | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |      |     1 |    19   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE            |      |     1 |            |          |
|   2 |   TABLE ACCESS INMEMORY FULL| T2   |   461K|    19   (0)| 00:0 |
-----------------------------------------------------------------------------------

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          8  consistent gets
          0  physical reads
          0  redo size
        542  bytes sent via SQL*Net to client
        551  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

In-Memory Option：性能对比

测试表T2大小15M，ID列上有非唯一索引：

Operation	no memory(微秒)	In-memory(微秒)	No-mem/IM
count(*)	18707（全表扫描）	13009	1.4
Sum(id)	79508（全表扫描）	48697	1.6
Where id=1	781（索引扫描）	6435	0.12

In-Memory Option：存储索引

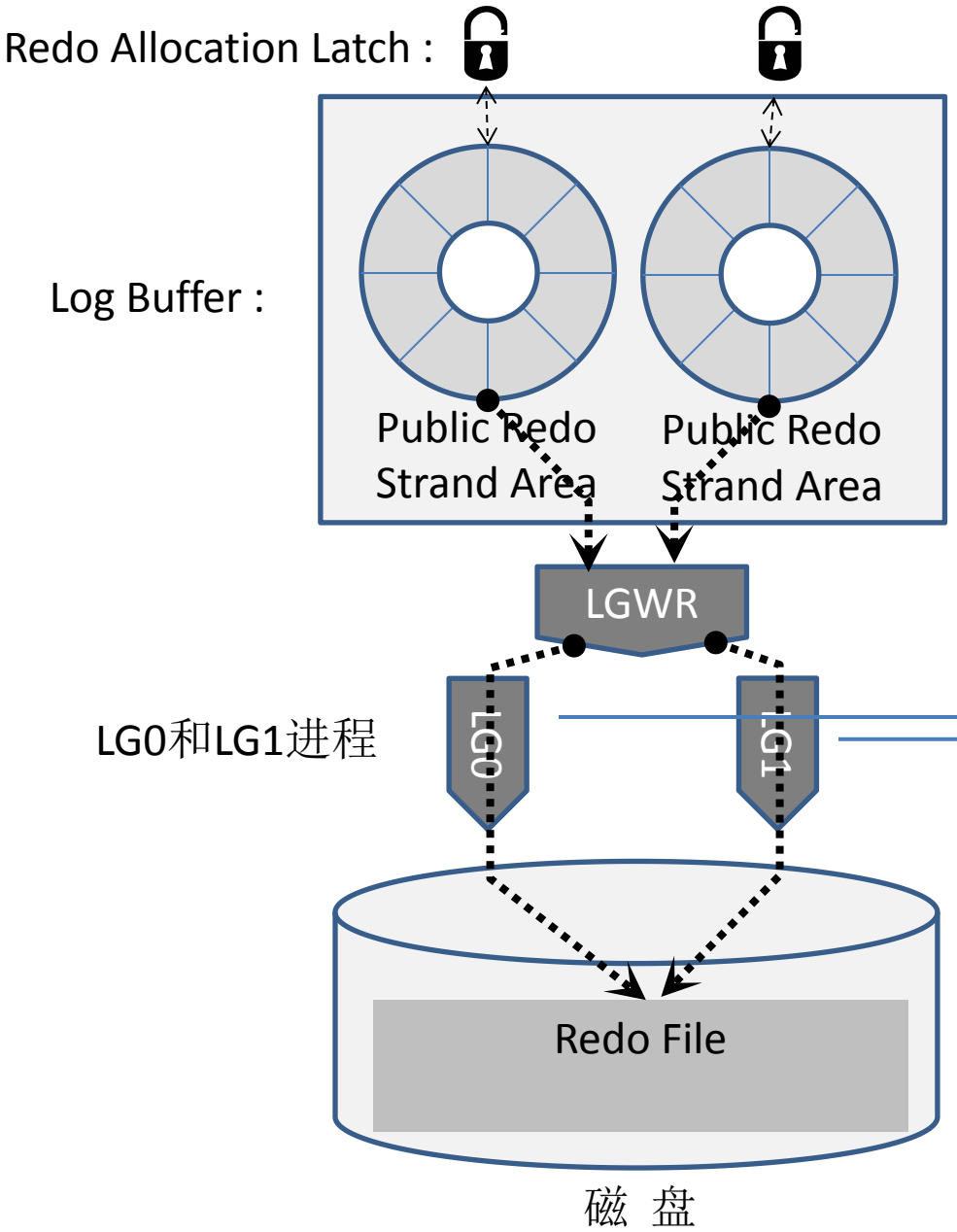


以ID列为例，Oracle将ID列的每个IMCU中最小ID值和最大ID值，保存到一块专门的内存区域，这个内存区就是存储索引。

对于非全扫描类操作，Oracle将先查找存储索引，过滤掉不需要扫描的IMCU，然后再进行扫描操作。比如查询条件是ID>20 and ID<=100的列值，图中第二个IMCU中，ID值的范围是410~600，它就将被过滤掉。

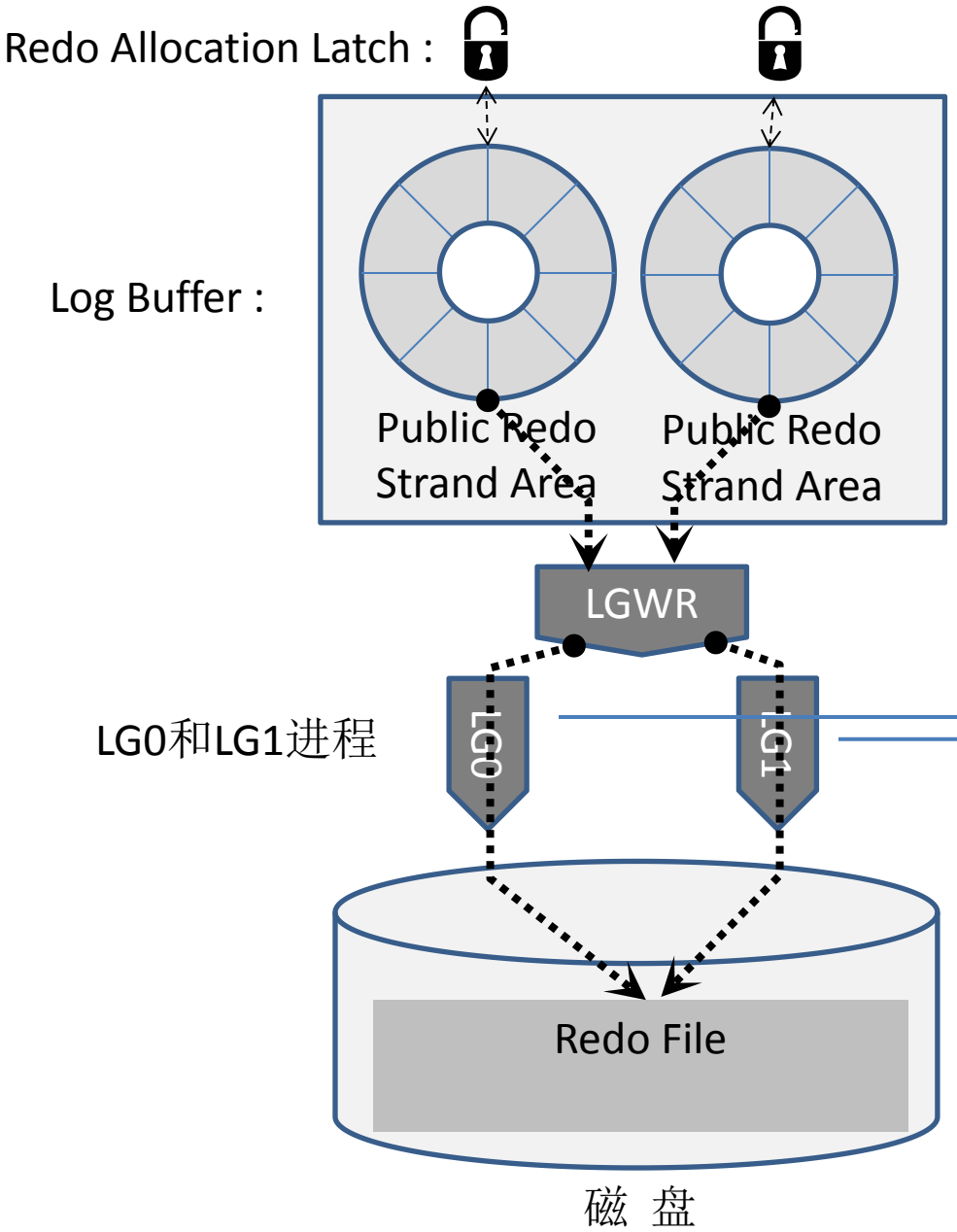
存储索引不同以传统的B树索引，和HASH也不一样，它只是为了帮助跳过一部分IMCU，对于结果集很小的查询操作，它对于性能的帮助比传统索引差很多。

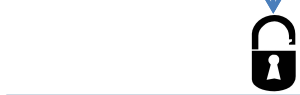
多LGWR



- 提交步骤：
  - （前台进程所占用Log Buffer会记录入自己的PGA。）
  - 前台进程发出Commit请求，将Log Buffer占用记录写入Message Area。
  - 前台进程调用semctl唤醒LGWR，请求完成I/O。然后调用semtimedop转入Sleeping状态，开始等待Log File Sync。
  - LGWR申请Redo Allocation Latch，确定要写的范围。
  - LGWR通知LG0或LG1写Redo。
  - LGn完成I/O后，获得Message Latch，查看Message Area中的消息，通知某个前台进程写Redo完成、Log File Sync结束。

多LGWR





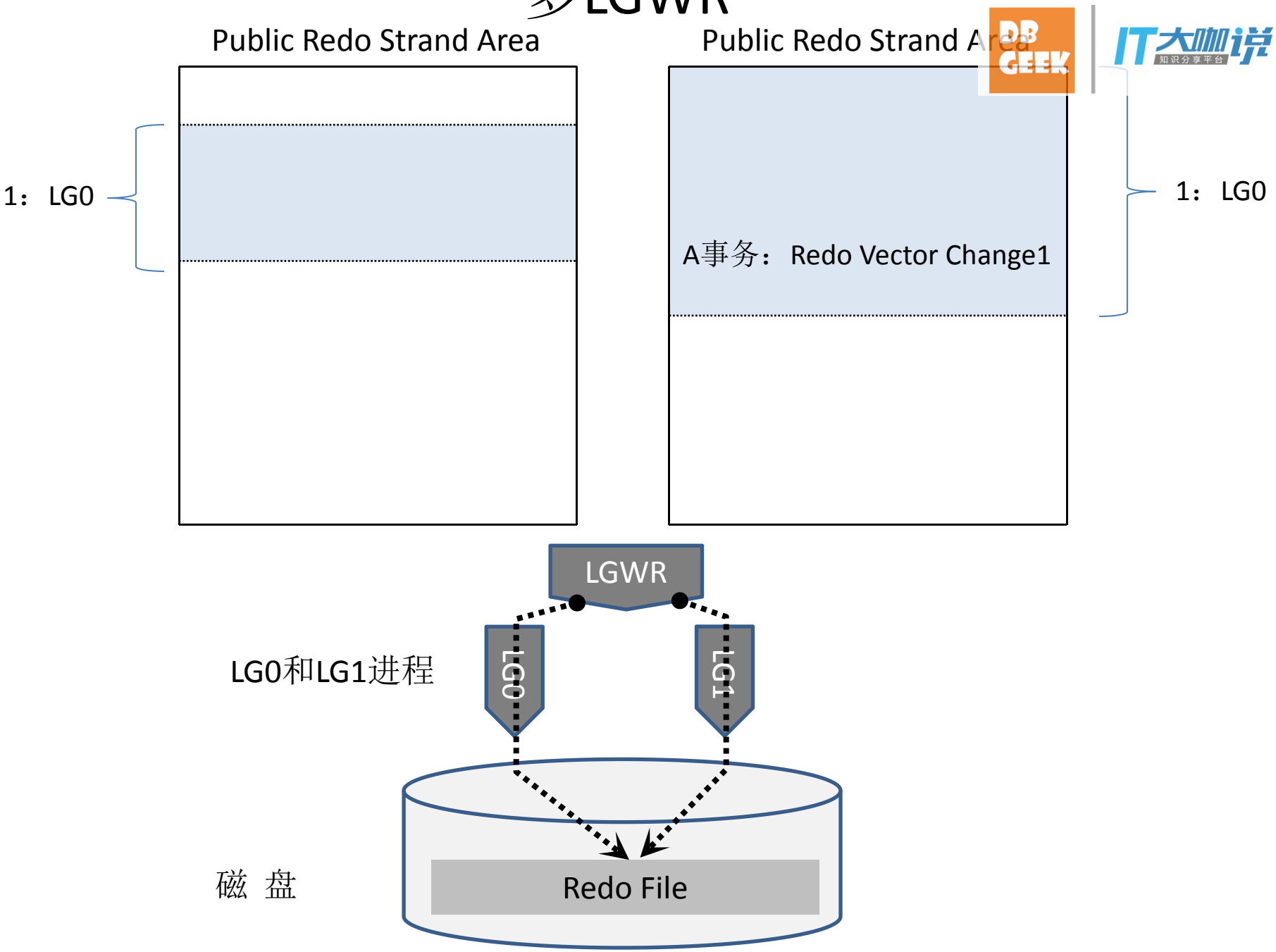
P1进程提交A事务，所占Log Buffer为：第N块，N+2块

P2进程提交B事务，所占Log Buffer为：第N块，N+1块

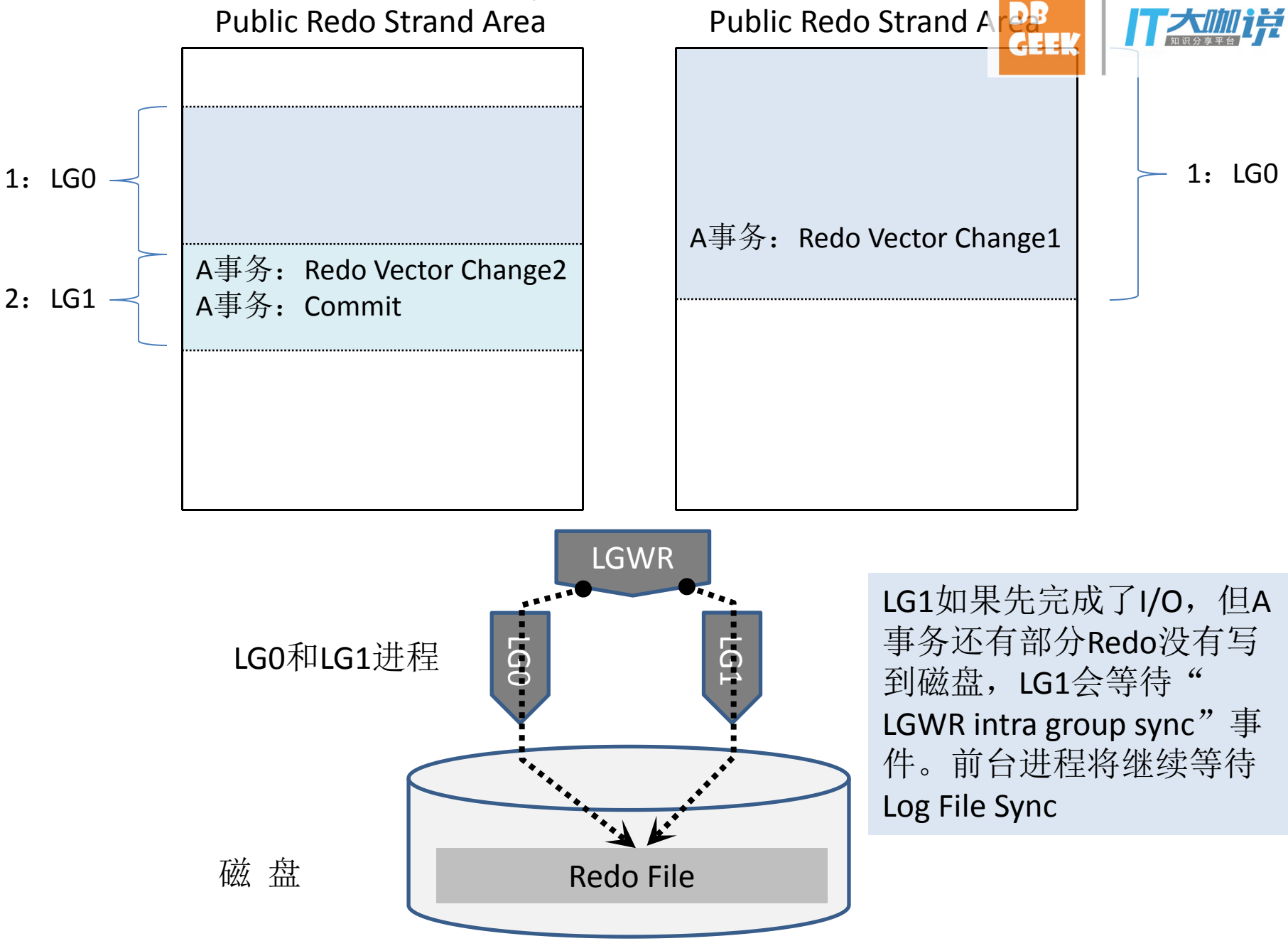
Message Area (在共享池中)

- LGWR通知LGn时的规则：
  - 当Redo量不大时，LGWR每次总是只通知LG0完成I/O。
  - 当Redo量较大时，LGWR才会通知其他的LG进程完成I/O。Redo量较大的主要判断条件：某一个Log Buffer满三分之一。
  - 使用_use_single_log_writer参数控制，默认值ADAPTIVE，将开启多LGWR特性。

多LGWR

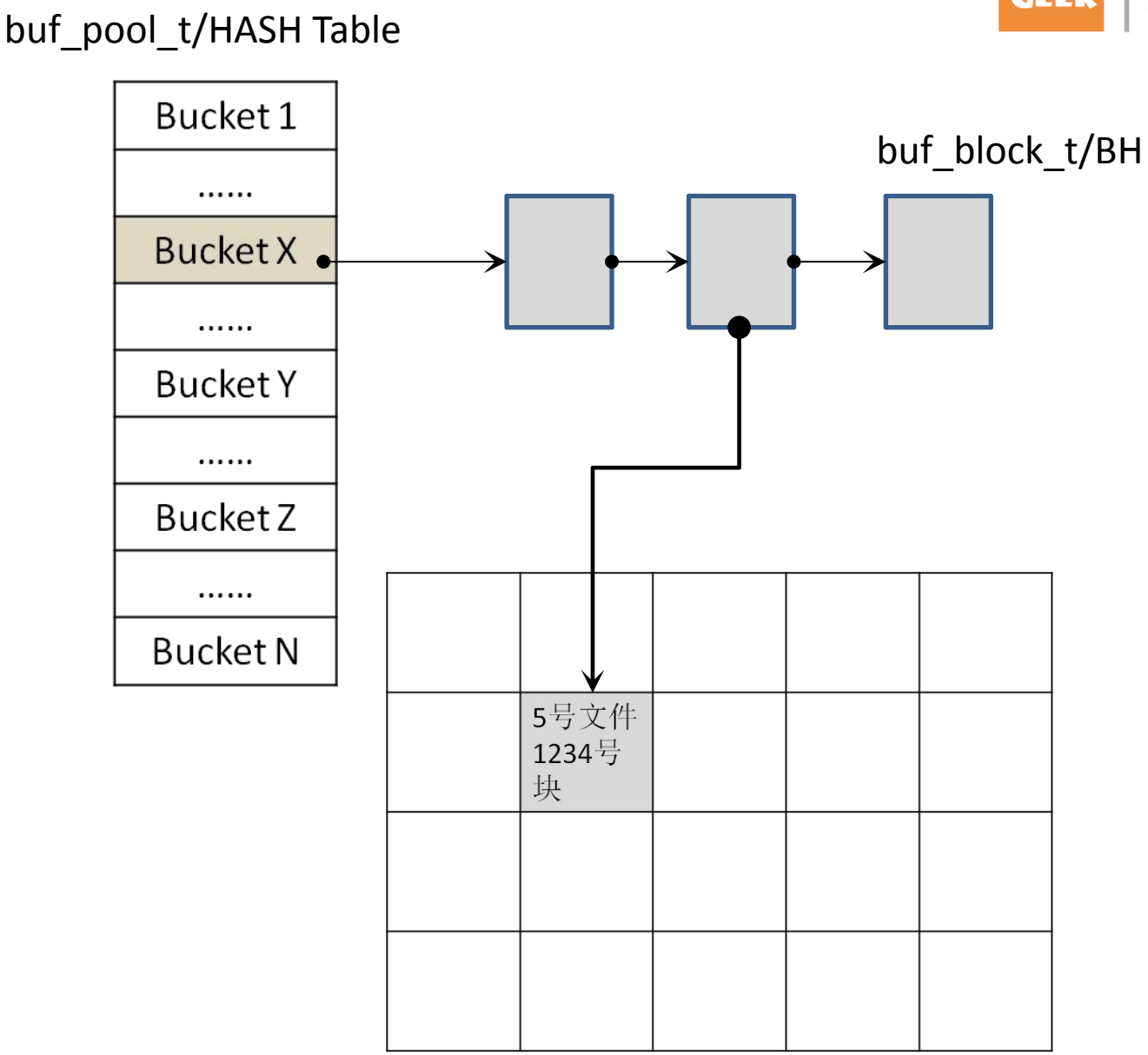

Public Redo Strand Area
Public Redo Strand Area
1：LG0
A事务：Redo Vector Change1
1：LG0
LGWR
LG0和LG1进程
LG0
LG1
磁 盘
Redo File

多LGWR



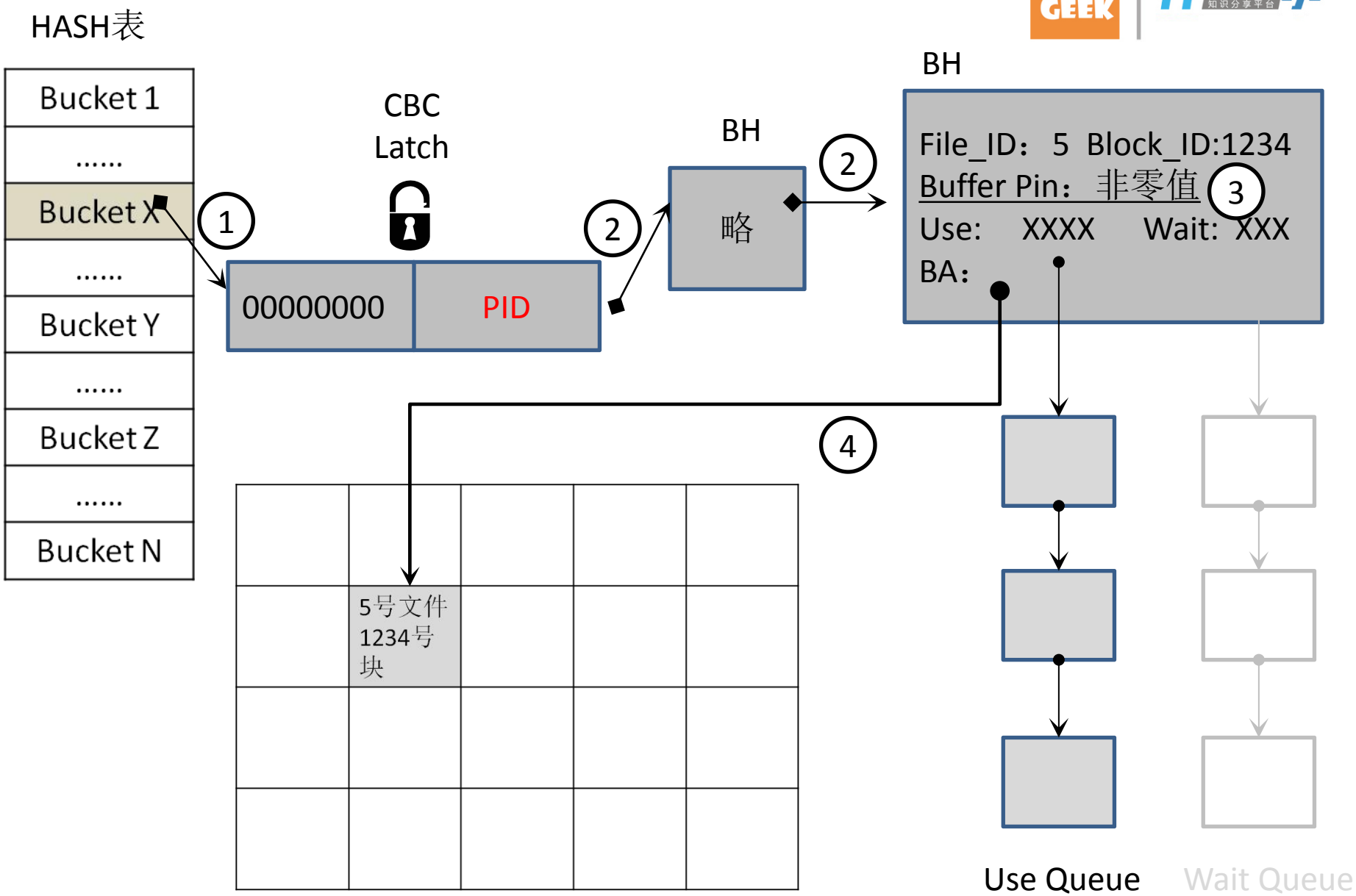
LG1如果先完成了I/O，但A事务还有部分Redo没有写到磁盘，LG1会等待“LGWR intra group sync”事件。前台进程将继续等待Log File Sync

逻辑读时锁的变化：发展历程

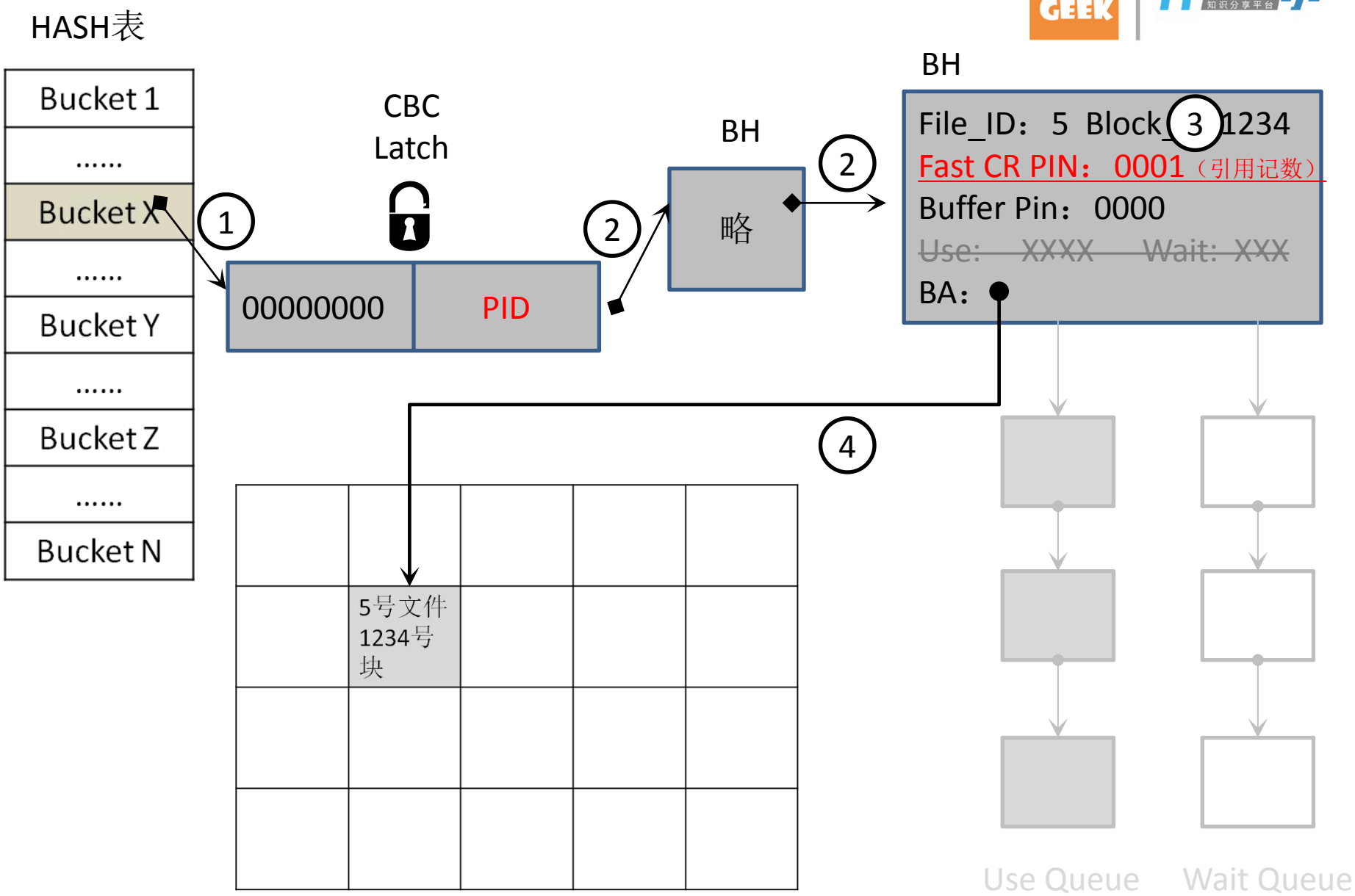


Wait Queue

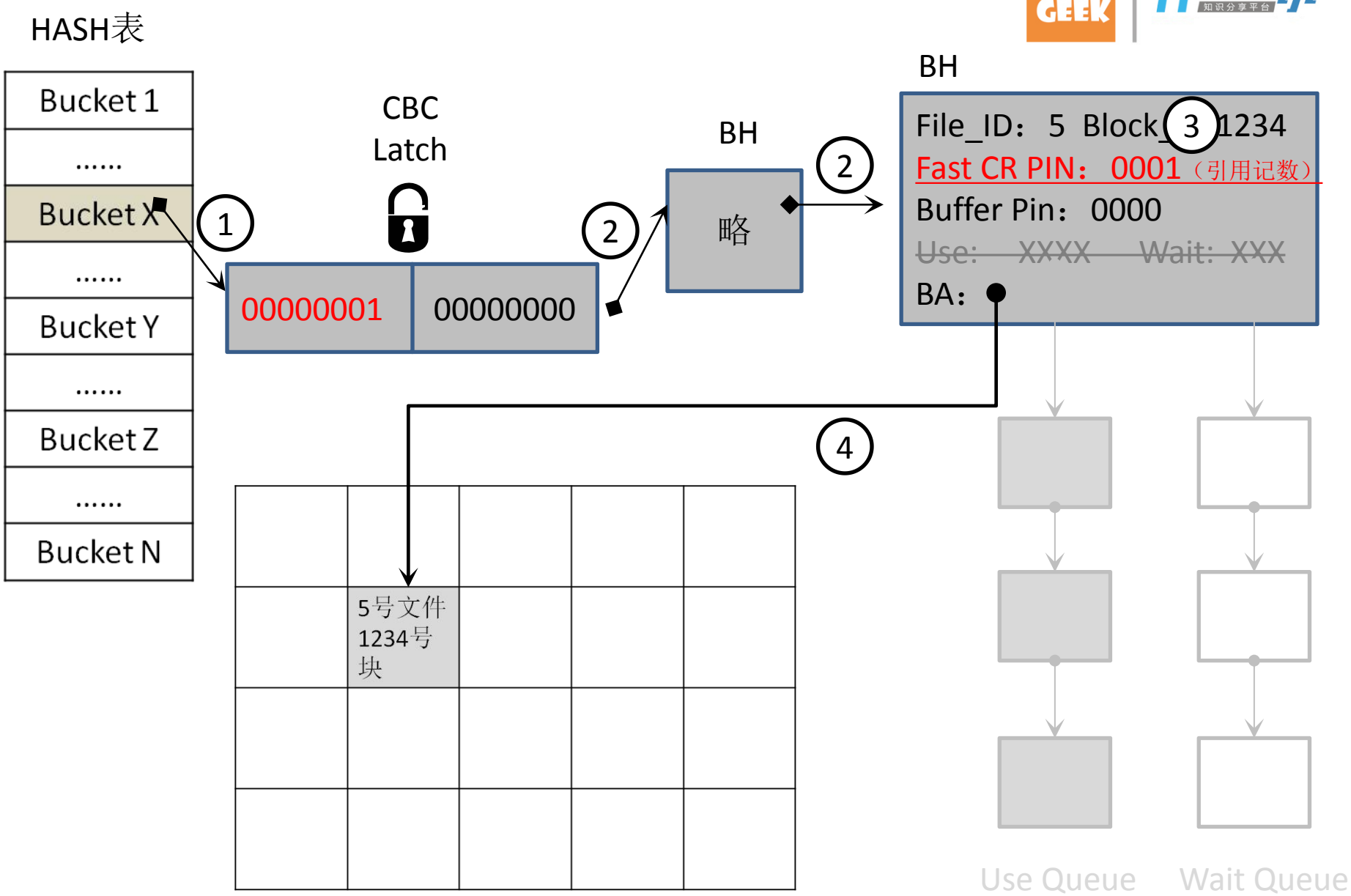
逻辑读时锁的变化：发展历程



逻辑读时锁的变化：发展历程



逻辑读时锁的变化：发展历程



逻辑读时锁的变化：发展历程