Write Ahead Log

Protect Your Data

Oracle
MySQL
PostgreSQL

平安科技数据库技术部
王鹏冲

中国平安
保险·银行·投资

平安科技
PING AN TECHNOLOGY

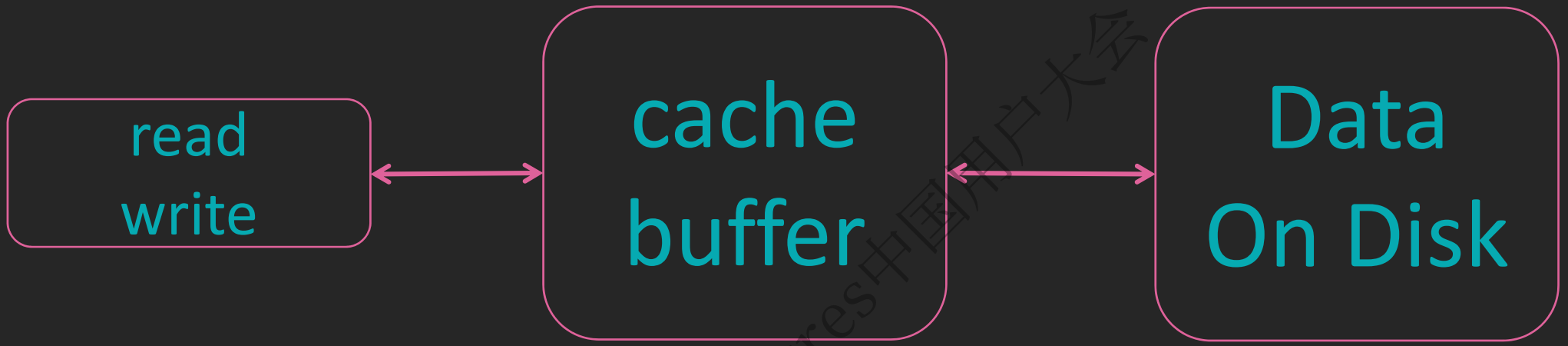
王鹏冲
2007年加入中国平安
十多年数据库从业经验
Oracle 10g OCP
曾带领运维DBA获取公司年度十佳团队
平安科技年度优秀原创课程获得者

微信公众号：数据库技术圈

ACID的D
Durability

雾里看花
如何确保事务提交的数据能被持久化

从缓存读写

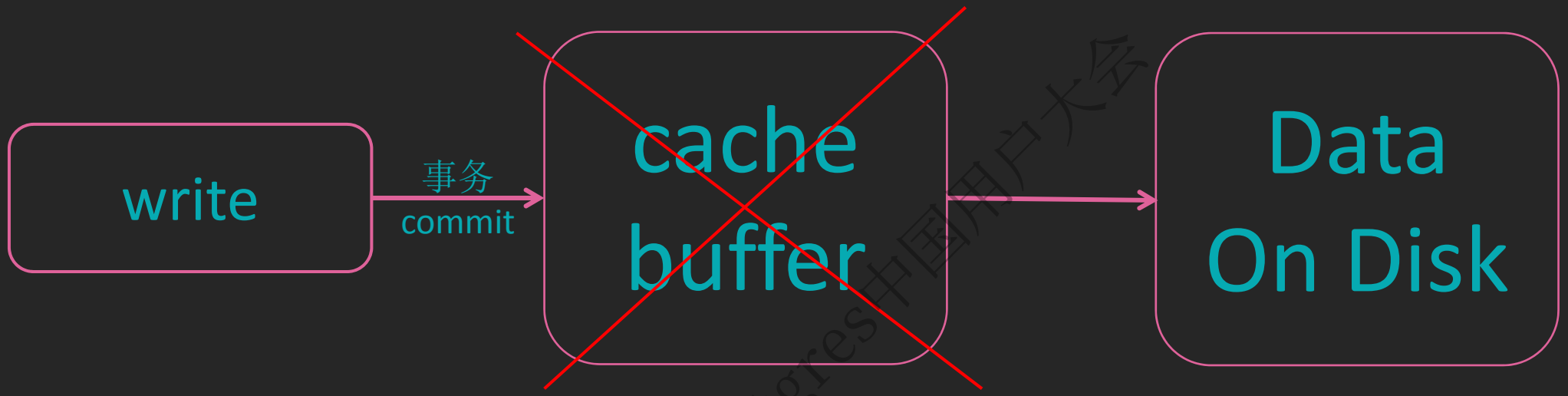


数据库的主要瓶颈是磁盘I/O，为了提高性能，数据库使用缓存、缓冲区管理数据。

读取时先读缓存（逻辑读），缓存没有命中再从磁盘读（物理读）。

写时先修改到缓冲区，然后异步批量刷新到磁盘。

主机崩溃时数据丢失



当事务提交时，若主机崩溃，在缓冲区里面的数据若还没有来得及写入磁盘，这部
数据就会丢失，这就破坏了事务的持久性（ Durability ）

你可以每次提交时都把所有数据都写在磁盘上，但是如果主机崩溃，最终数据可能
有部分写入磁盘，这破坏了事务的原子性（ Atomicity ）。

解决办法

务作出的任何修改必须是或者全部完成，或者全部撤销。

2 个办法解决这个问题：

影子副本/页（Shadow copies/pages）：
个事务创建自己的数据库副本（或部分数据库的副本），并基于这个副本来工作。
错，这个副本就被移除；一旦成功，数据库立即利用文件系统的一个功能，把副
到数据中，然后删掉『旧』数据。

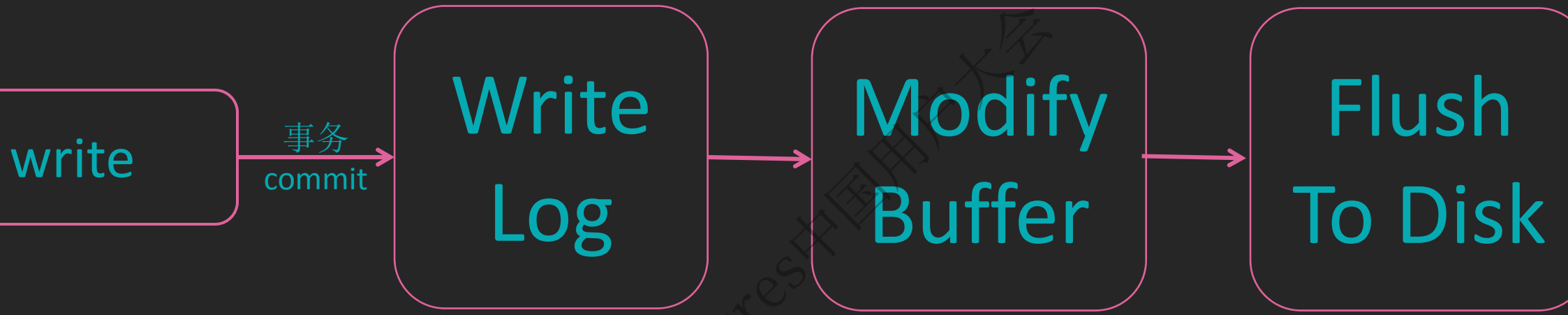
事务日志（Transaction log）：
务日志是一个独立于data的存储空间，在每次写数据到磁盘之前，数据库在事务日
写入一些信息，这样当事务崩溃或回滚，数据库从日志里面就可以知道如何移除（
或继续完成（前滚）尚未完成的事务。

核心问题：
为什么每次commit时不直接刷磁盘？不用写日志只同步刷磁盘不也一样可以达到“D”的目的？

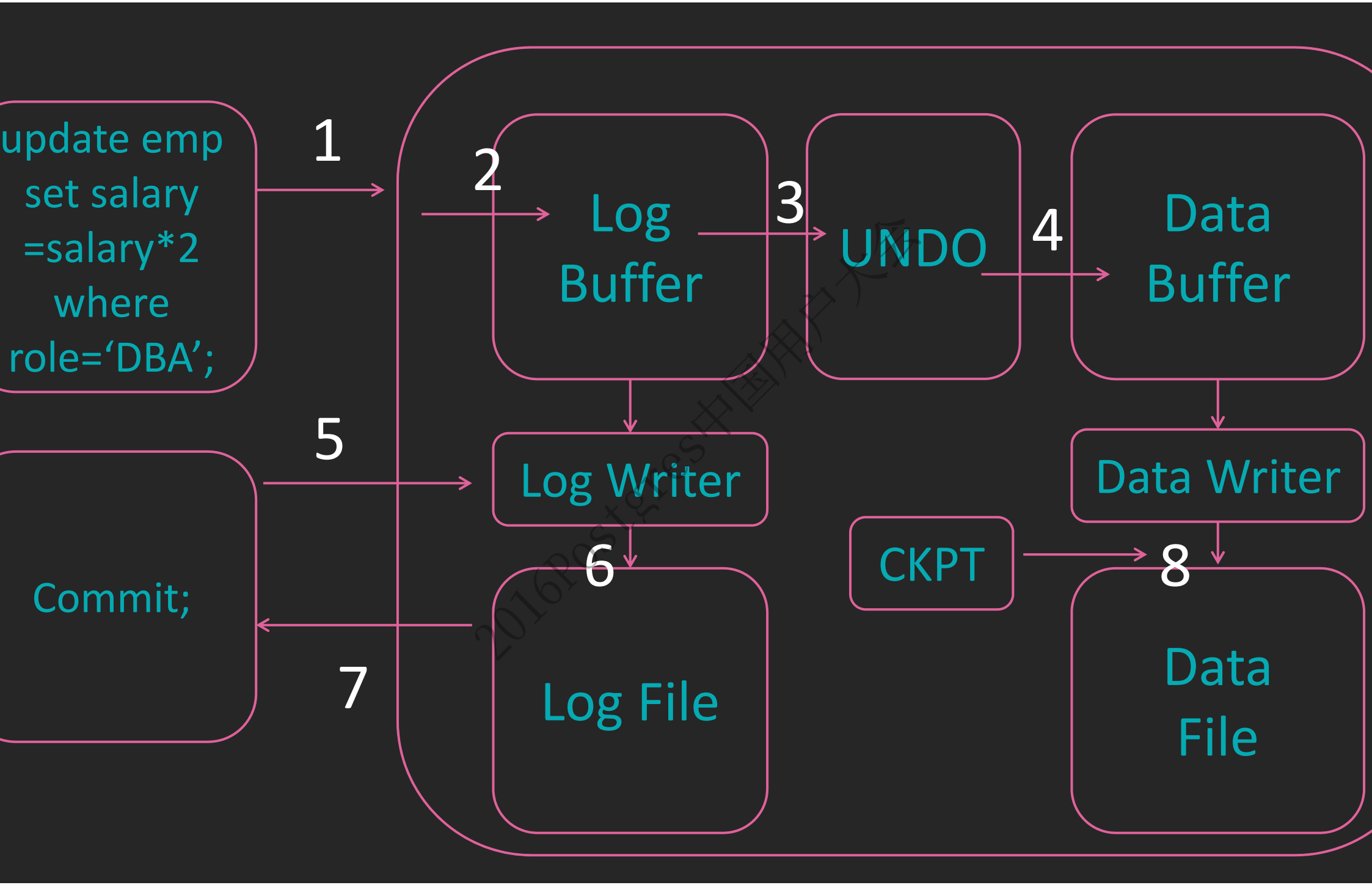
“LGWR writes one contiguous portion of the buffer to the online redo log. By separating the tasks of modifying database buffers, performing scattered writes of dirty buffers to disk, and performing fast sequential writes of redo to disk, the database improves performance.”

Transaction log for Crash recovery、PITR、REDO FILE REUSE。

主流关系型数据库的处理办法



数据库（至少是Oracle， PostgreSQL, MySQL ，Sybase，SQL Server等）使用预
志协议（Write-Ahead Logging protocol ，WAL）来处理事务日志。
协议有 3 个规则：
每个对数据库的修改都产生一条日志记录，在数据写入磁盘之前日志记录必须写入
日志。
日志记录必须按顺序写入；记录 A 发生在记录 B 之前，则 A 必须写在 B 之前。
当一个事务提交时，在事务成功之前，提交指令必须成功写入到事务日志。


update emp
set salary
=salary*2
where
role='DBA';
1
2
Log
Buffer
3
UNDO
4
Data
Buffer
5
Log Writer
Data Writer
Commit;
6
CKPT
8
7
Log File
Data
File
2016Postgres中国用户大会

思考几个问题

若每次事务的commit，都要等待Log Writer将日志从log buffer写入到log file
，如果写log file的IO效率较慢，事务commit将会出现等待。
类数据库是如何考虑这个环节的性能与Duration的权衡的？

在checkpoint唤醒Data Writer刷脏块的时候，如果需要刷的脏块太多，或者刷的
太慢，那么会带来什么问题？
类数据库是怎么处理的？

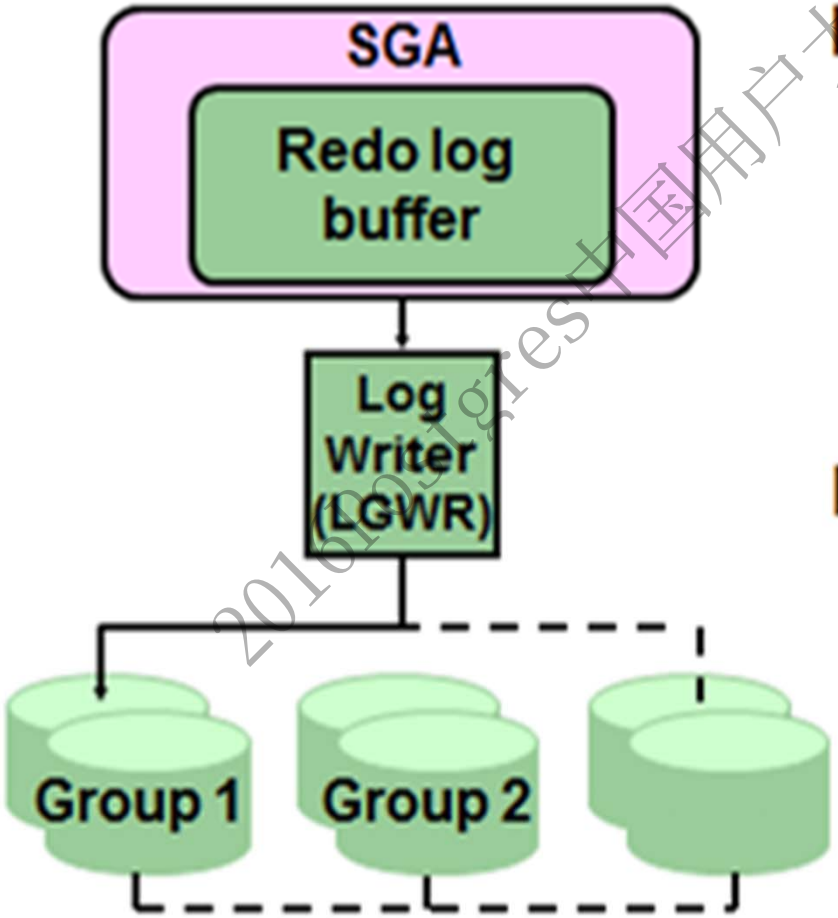
在log writer和data writer 写入操作系统文件时，是如何调用filesystem io
何防止os假写。
c I/O vs Asyn I/O ？ Buffered I/O vs Direct I/O

面我们来对比看看不同数据库对待这些问题时，都有哪些可用的处理手段，从中也
下不同数据库的设计思想。

Oracle

GWR：
mmit、
g switch、
、
3满、
MB、
te-ahead DBWR

Redo Log Files and LogWriter



Redo log files:

- Record changes to the database
- Should be multiplexed t protect against loss

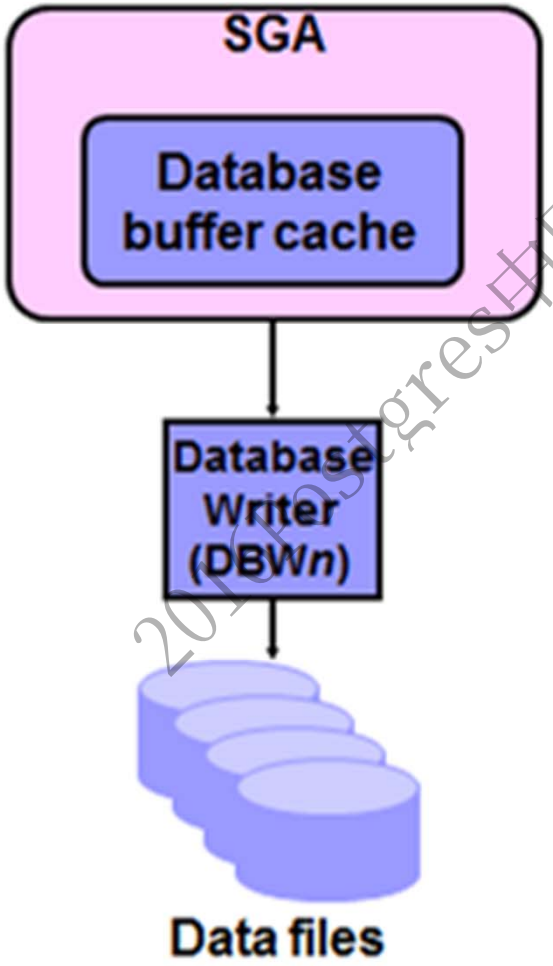
LogWriter writes:

- At commit
- When one-third full
- Every three seconds
- Before DBW*n* writes

Oracle

any cases the blocks
DBWn writes are
tered throughout
disk. Thus, the
es tend to be slower
the sequential
es performed by
VR. DBW performs
tiblock writes when
sible to improve
iency. The number
ocks written in a
tiblock write varies
perating system.

Database Writer (DBW*n*)



Background Information

DBW*n* writes when one of the following events occurs:

- Checkpoint
- Dirty buffers' threshold
- No free buffers
- Timeout
- RAC ping request
- Tablespace `OFFLINE`
- Tablespace `READ ONLY`
- Table `DROP` or `TRUNCATE`
- Tablespace `BEGIN BACKUP`

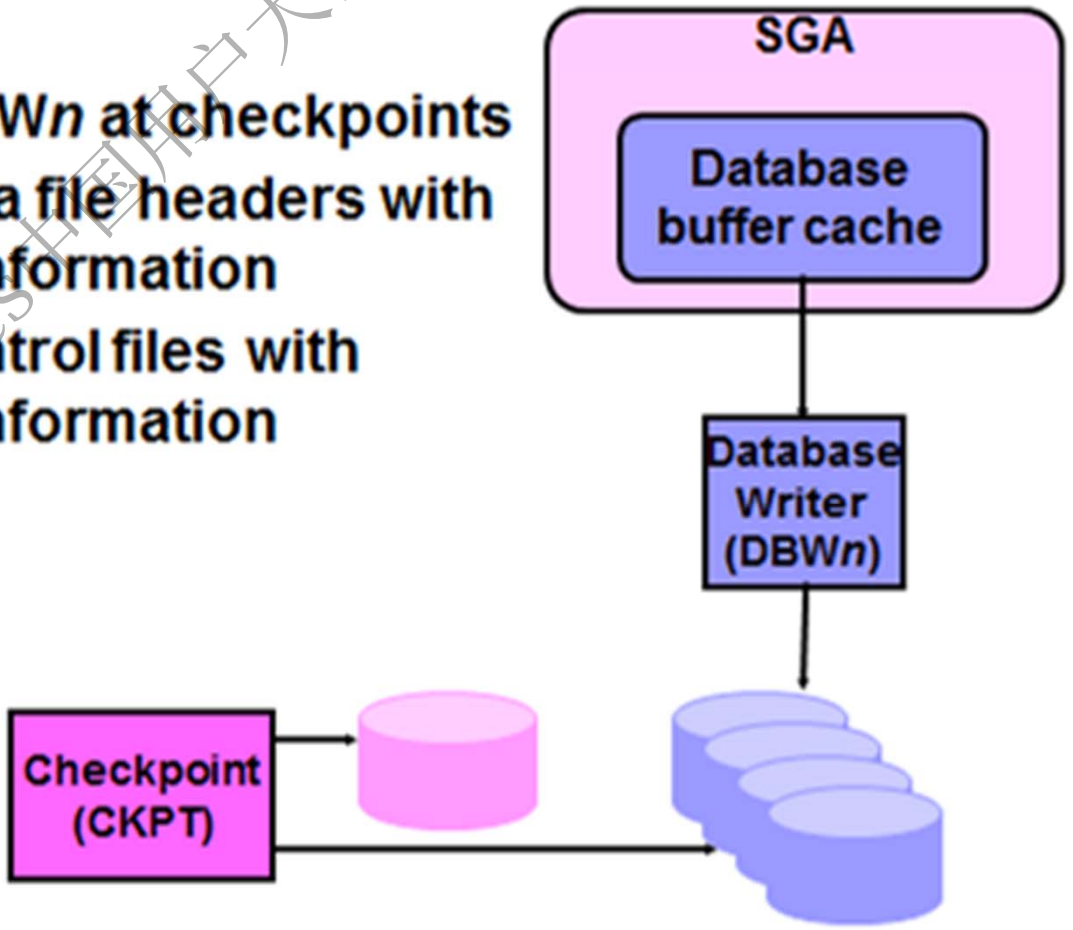
Oracle

checkpoint process
PT) updates the
rol file and data file
ders with checkpoint
rmation and signals
/ to write blocks to
Checkpoint
rmation includes the
ckpoint position, SCN,
tion in online redo
o begin recovery,
so on.

Checkpoint (CKPT)

Responsible for:

- Signaling DBW*n* at checkpoints
- Updating data file headers with checkpoint information
- Updating control files with checkpoint information



Async IO

Asynchronous IO指的是异步IO。
操作系统实现异步IO有两种方式：
一种是下面的Kernelised Asynchronous IO（kaio），
一种是通过多线程模拟的异步IO。
对于同一个线程来讲，IO还是同步的。但是通过启用多个线程并且同时发起同步I
用，一般是read/write/pread/pwrite，达到了异步IO的效果。主线程可以继续其他
这种方式一般是在数据库运行在文件系统上时使用。

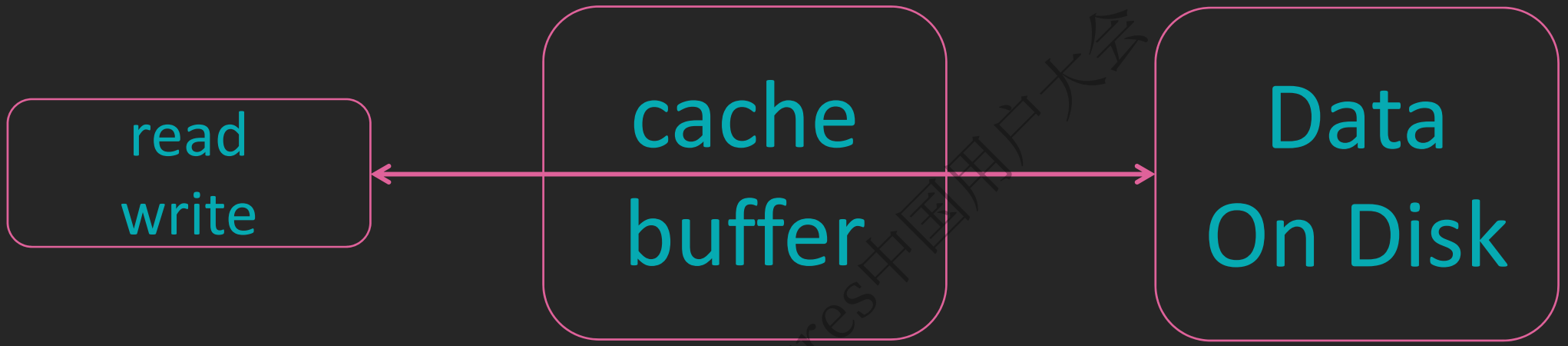
Kernelised Asynchronous IO(kaio)是OS核心支持的异步IO模式，最为高效，没有通过
程模拟异步IO引发的线程调用和通讯开销。但是一般当数据库运行在文件系统上时
OS无法支持kaio的方式，只能以多线程方式来模拟。当然，某些特殊情况下也可以
例如Veritas Quick IO或者ODM都可以支持kaio。

Concurrent IO

Concurrent IO(CIO)是指支持对操作系统文件的并发访问。POSIX接口标准
义要求文件系统对同一个文件的操作要加锁防止并发读写的冲突，但是
显著降低并发。由于数据库自身已经实现了读写的锁机制，所以对于数
库使用的文件系统一般都推荐将Concurrent IO进行enable。

在很多操作系统和文件系统中，当DIO启用后，CIO也会隐式自动启用。
是像AIX、VxFS等，必须显式启用CIO。

Direct IO



由于文件系统缓存对数据库读性能是有提升的，所以如果只是想提升数据库LGWR
性能，在Oracle运行在文件系统的情况下，可以将redo所在的文件卷独立出来并且
卷文件系统同时启用DIO和CIO。但仍然会存在线程调度的scheduling latency。
如果想解决scheduling latency，并且数据库运行在文件系统上，只能启用ODM。Or
对于ODM是支持DIO+CIO+KAIO的，但是要注意同时考虑牺牲读性能的影响。或者增
buffer cache的大小将文件系统缓存转变为buffer cache。
如果Oracle运行在ASM上，那么效果等同于ODM。

Oracle

racle推荐使用的IO模式：
ynchronous I/O
rect I/O
ncurrent I/O(*Disabling RW locking with direct I/O*)

参数控制1：

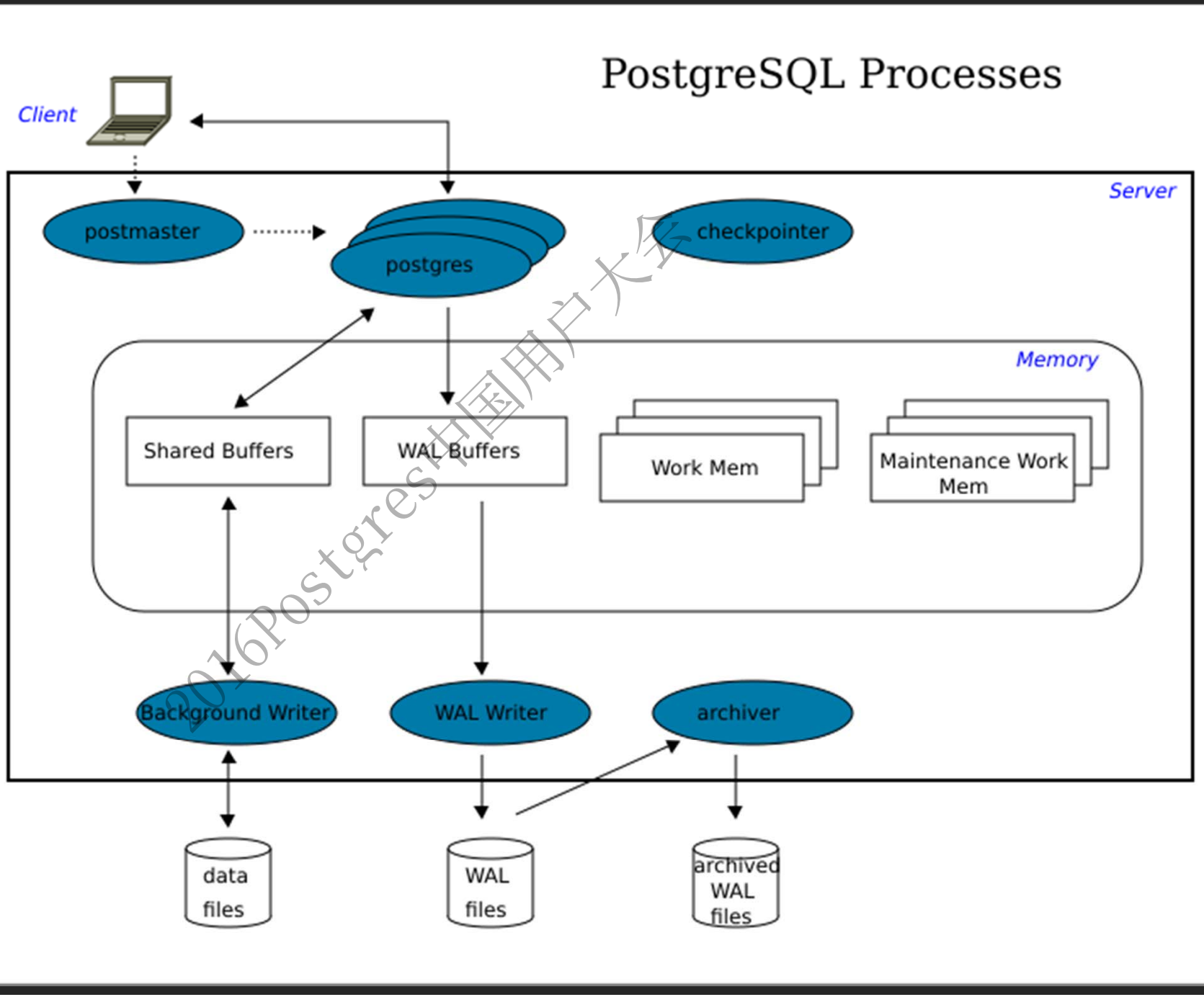
systemio_options	Synchronous I/O	Asynchronous I/O
ffered I/O	none	asynch
ect I/O	directIO	setall

参数控制2：

disk_async_io	Synchronous I/O	Asynchronou
	false	true

The parameter acts as a master switch for asynchronous I/O; when set to false, all I/O is performed synchronously, regardless of the filesystemio_options setting. It is recommen to keep this parameter at the default setting of t

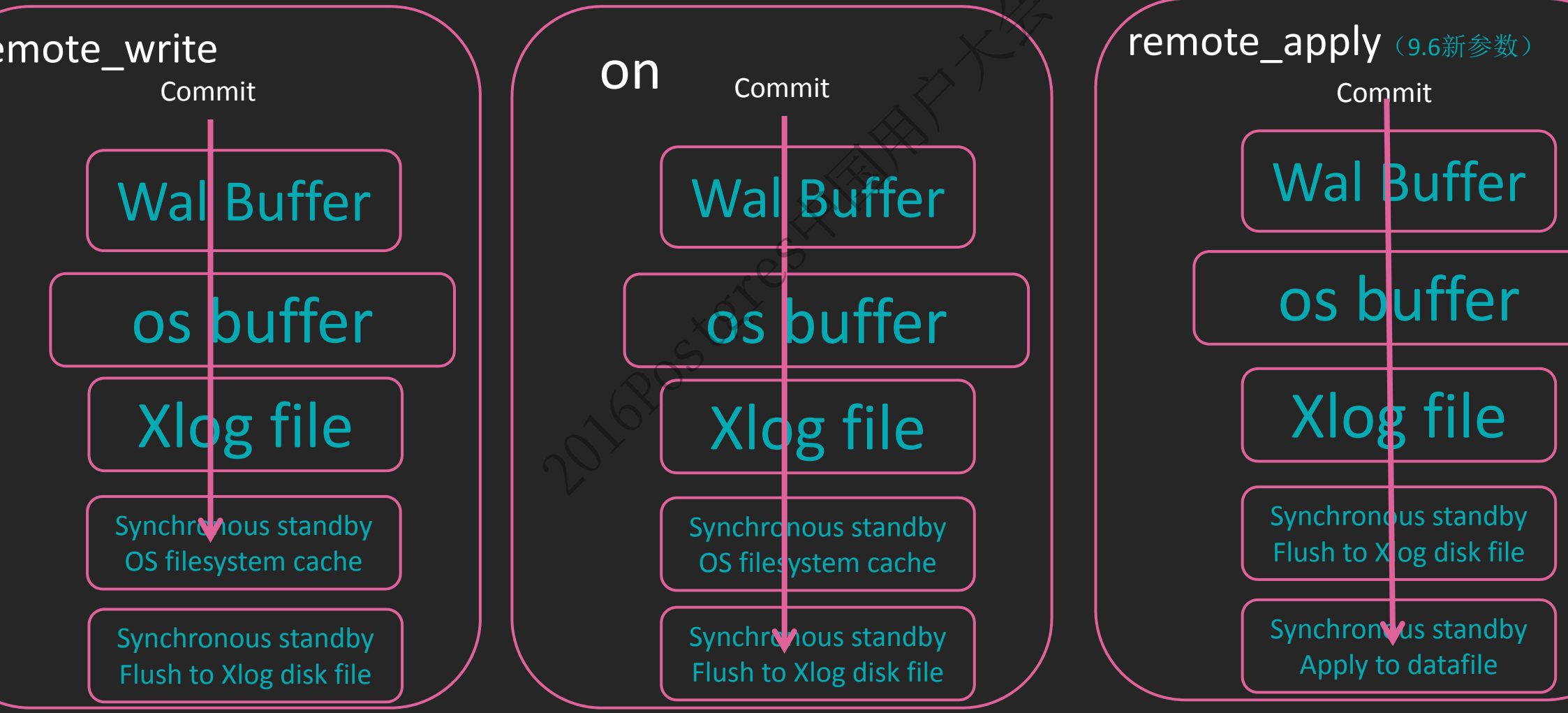
ostgreSQL



PostgreSQL

synchronous_commit
控制事务comit时的WAL log的行为

*假设开启了同步复制的情况下

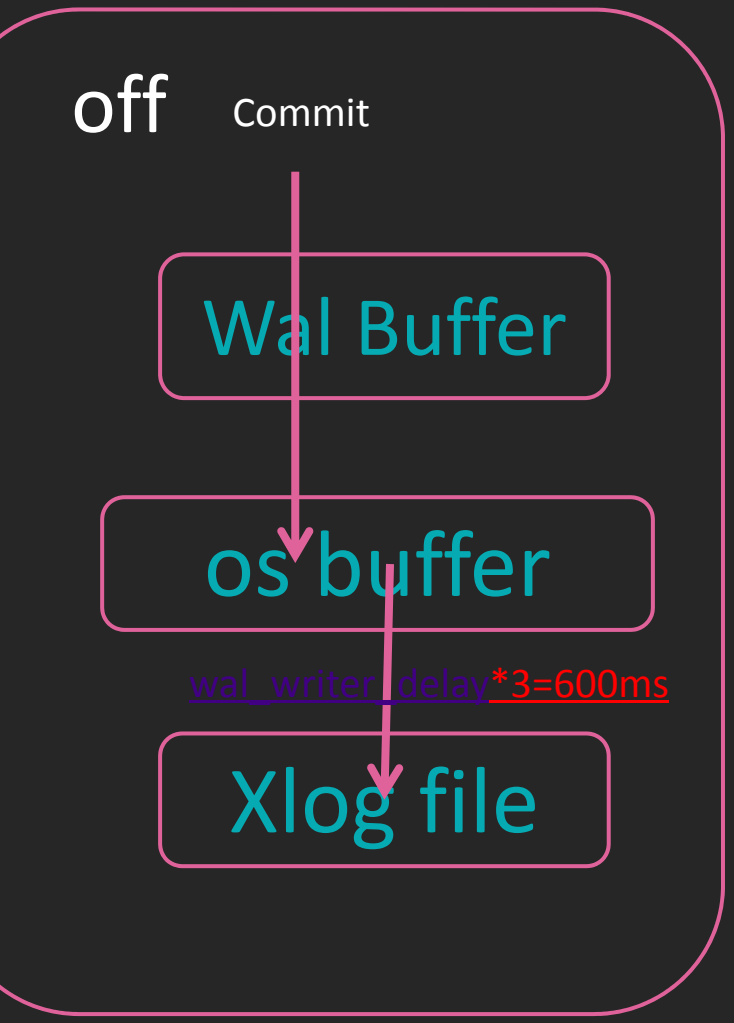


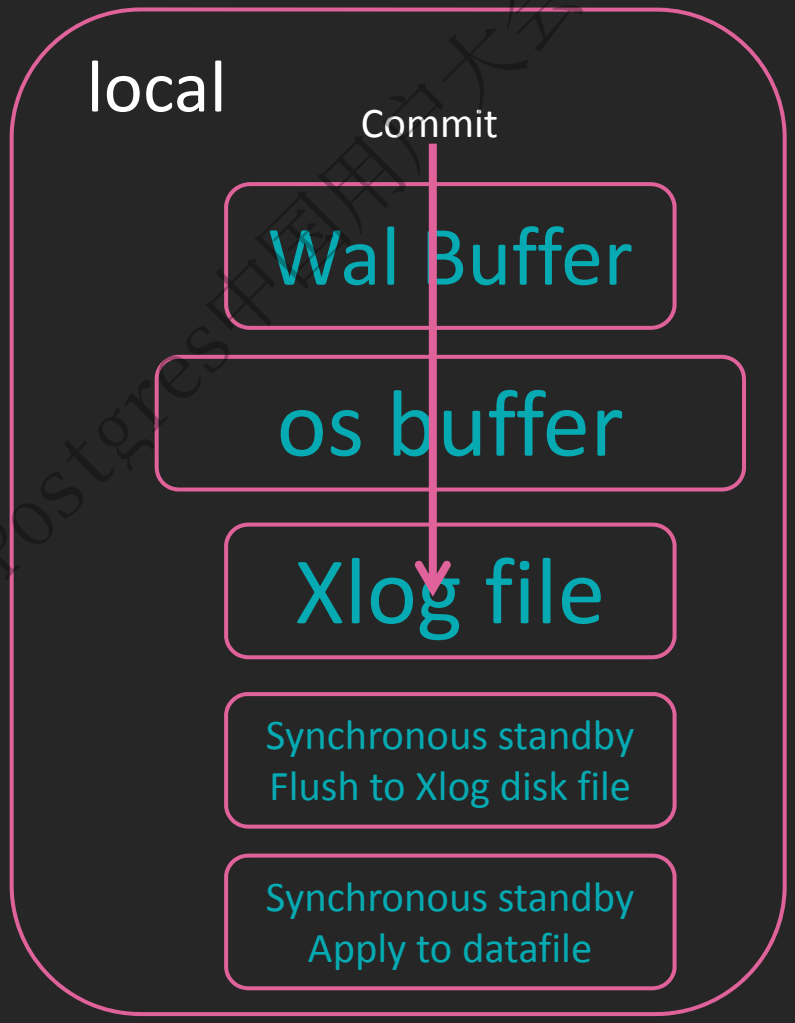
PostgreSQL

synchronous_commit

控制事务commit时的WAL Buffer刷磁盘的行为

*假设开启了同步复制的情况下





*假设未开启同步复
则如下设置都一样:
on, remote_apply,
mote_write and lo
all provide the san
synchronization lev
transaction commi
only wait for local
flush to disk.

PostgreSQL

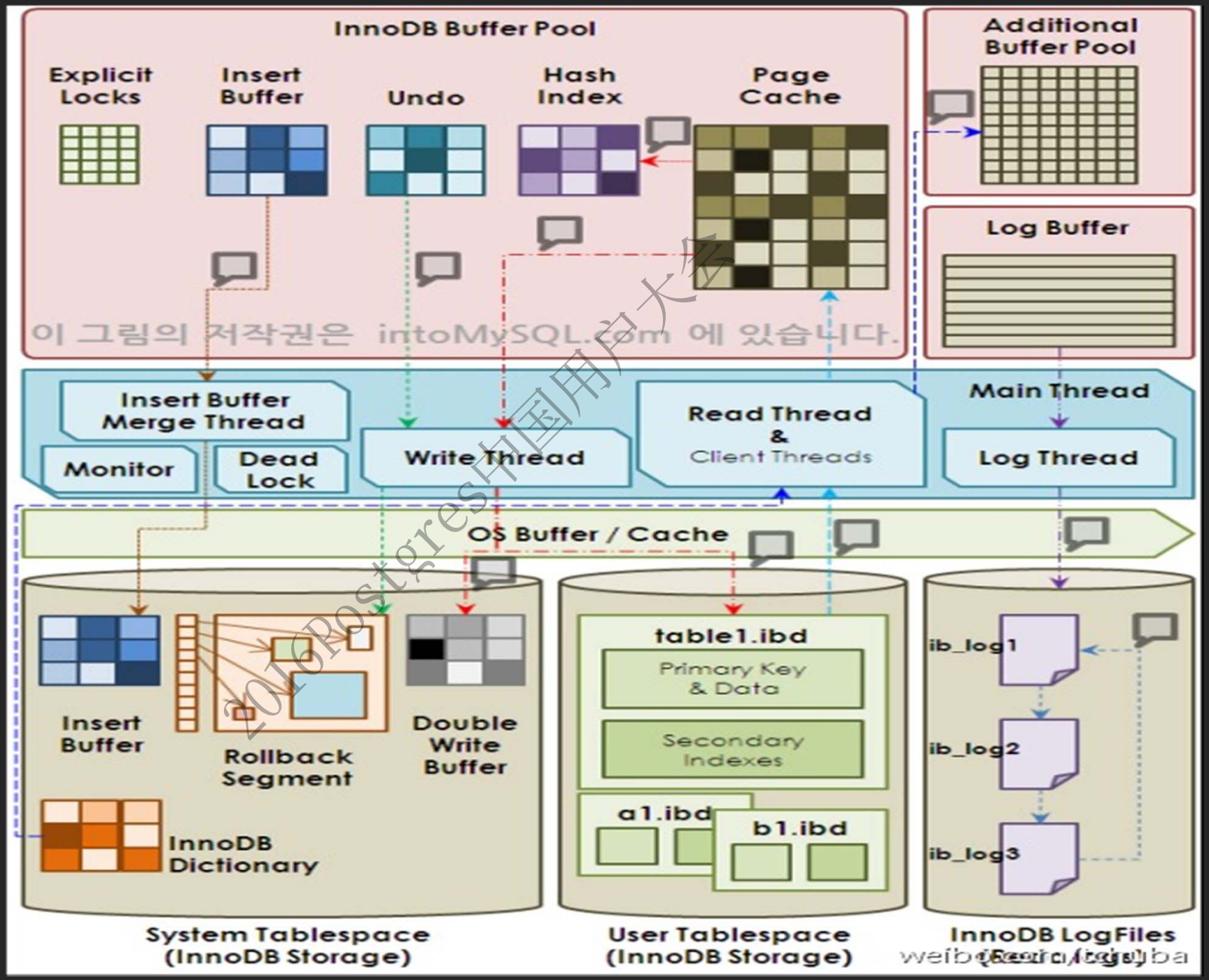
其它相关参数：

fsync： 控制在将更新写入磁盘时是否调用fsync()，确保真实写入，一般都设为true
wal_sync_method： 控制将WAL log强制刷磁盘时所使用的具体方法，在Fsync为ON的时候才有效。可选值为open_datasync/fdatasync/fsync/fsync_writethrough/open_sync
Linux系统默认为fdatasync，有些选项在有些OS平台上面不支持。

wal_writer_delay： 如前所述，控制WAL log从OS cache刷到磁盘的间隔时间。
wal_writer_flush_after： （9.6新参数）跟上面参数一样的功能，控制间隔bytes。

commit_delay： 当有多个active的事务时，让大家等着一起提交、只触发一次WAL log flush，从而提高组提交的吞吐量。（9.3前后的行为不同，具体见doc)
commit_siblings： 控制至少同一时间有多少个并发事务时，才会实施commit delay，默认是5个。

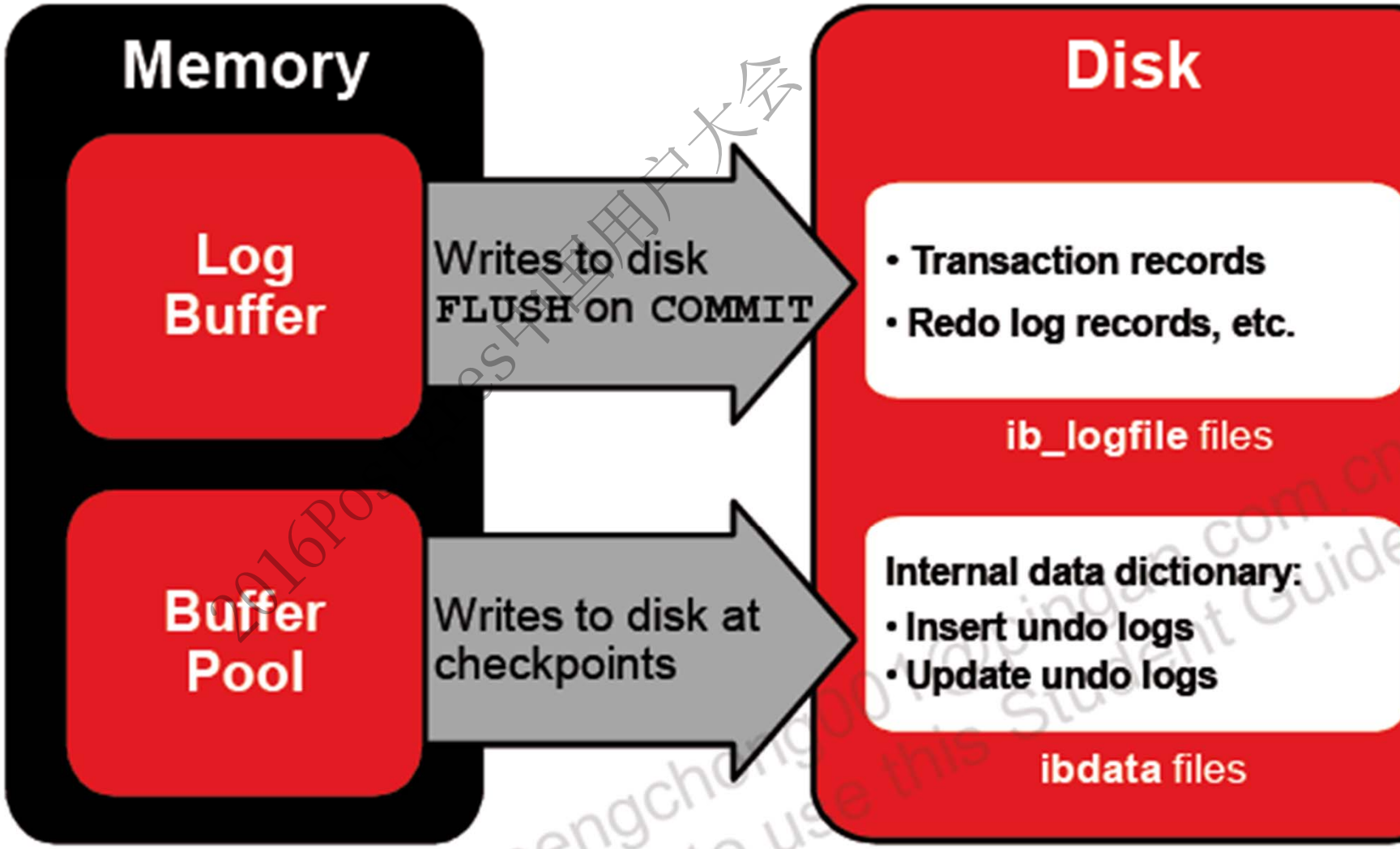
Mysql


InnoDB Buffer Pool
Explicit Locks
Insert Buffer
Undo
Hash Index
Page Cache
Additional Buffer Pool
Log Buffer
이 그림의 저작권은 intoMySQL.com 에 있습니다.
Insert Buffer Merge Thread
Monitor
Dead Lock
Write Thread
Read Thread & Client Threads
Main Thread
Log Thread
OS Buffer / Cache
Insert Buffer
Rollback Segment
Double Write Buffer
InnoDB Dictionary
table1.ibd
Primary Key & Data
Secondary Indexes
a1.ibd
b1.ibd
ib_log1
ib_log2
ib_log3
System Tablespace (InnoDB Storage)
User Tablespace (InnoDB Storage)
InnoDB LogFiles
2016Postgres中国用户大会

MySQL

and flush the logs
N seconds. innodb_f
og_at_timeout was
luced in MySQL 5.6.6.
ws the timeout
d between flushes to
creased in order to
e flushing and avoid
cting performance of
y log group commit.
to MySQL 5.6.6,
ng frequency was
per second. The
lt setting
nodb_flush_log_at_ti
t is also once per
d.

Log Files and Buffers: Diagram

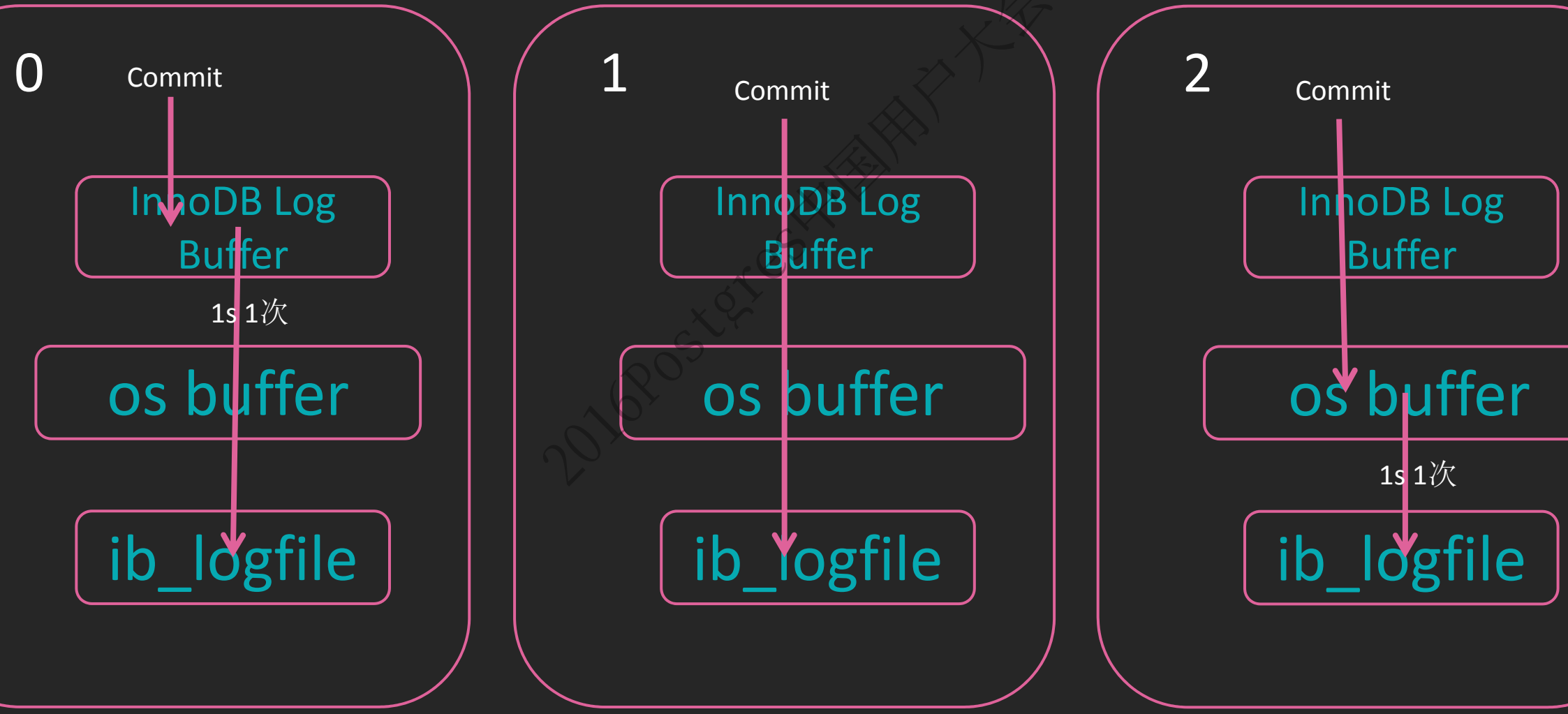


MySQL

Innodb存储引擎：

innodb_flush_log_at_trx_commit

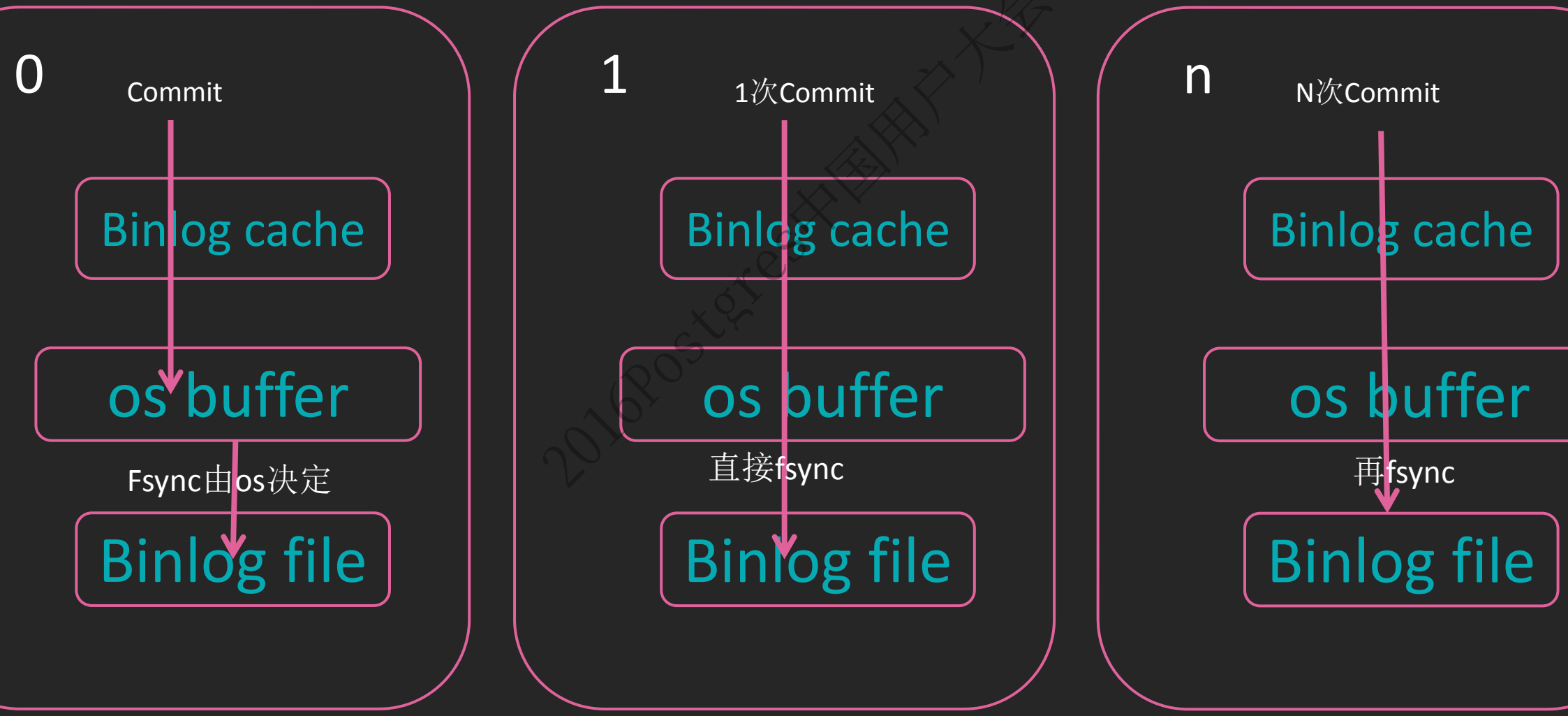
控制事务comit时的日志写到ib logfile行为



MySQL

sync_binlog

控制将二进制日志写入二进制文件的行为



MySQL

Innodb存储引擎：

innodb_flush_method

控制数据写入ib data file和ib log file的行为

Unix-like systems	Open log	Flush log	Open datafile	Flush data
Fsync		fsync()		fsync()
O_DSYNC	O_SYNC	O_SYNC		fsync()
O_DIRECT		fsync()	O_DIRECT or directio() on Solaris	Fsync()

- ✓ fsync被认为是安全的，因为在MySQL总会调用fsync来flush数据。
- ✓ 使用O_DSYNC是有些风险的，有些OS会忽略该参数**O_SYNC**。
- ✓ O_DIRECT和fsync和很类似，但是它会使用**O_DIRECT**来打开数据文件。有数据表明，如果是大量随机写入操作，**O_DIRECT**会提升效率。但是顺序写入和读取效率都会降低。所以使用O_DIRECT需要谨慎。

MySQL 最佳实践

实践证明：

少有把innodb_flush_log_at_trx_commit/sync_binlog设置为0的，再不重要的系统，果问开发和运营人员，他们也不愿意仅仅是实例crash就要丢失数据。

一般情况下推荐设置：

nodb_flush_log_at_trx_commit=1

nc_binlog>=1

业务举行促销活动时，比如平安的财神节，可以考虑临时将nodb_flush_log_at_trx_commit设置为2，sync_binlog继续调高，以加大系统并发，要主机和存储没有故障，其实事务的“D”是一样的。

特别注意：

论怎么设置，请确保将你的ib_logfile和binlog_file放到有带电池保护的写缓存的d卷上。

对比1：维护便利性

若日志文件太大，可能会导致日志组切换耗时过长，instance crash recovery过长，以及从库应用日志（非实时apply模式）所差lag过大。
而日志文件太小，又会导致日志组切换频繁，ckpt过于频繁也会有性能问题。
所以经常我们需要根据实际情况调整日志文件大小。

	Oracle	PostgreSQL	Mysql
Change redo log size	online操作，增减logfile，改变大小	--with-wal-segsize 需要重新编译软件	innodb_log_file_siz 静态参数，重启生效
change Archived log size	随redo log而动	随Wal log而动	max_binlog_size 动态参数，在线修改

问题：

当数据库的日志写量大、效率慢造成拥堵等待的时候，
应用层的现象是什么？
数据库层的现象是什么？
数据库出现连接风暴，到底是因还是果？
为什么同样的堵塞，有些应用影响很大，有些应用的影响很小？

对比2：加速日志写

然Oracle的日志写也可以通过参数控制（COMMIT_WAIT、COMMIT_WRITE），是否必须是
步确保写入磁盘的online redo logfile然后再反馈会话commit完毕，但是一般来讲我们很少
Oracle库修改此参数，也就是说Oracle库为了保证事务的“D”一般最佳实践都是选择默认
项（immediate,wait）日志立刻同步刷新到DISK。所以对于Oracle的日志写的性能优化更多
在如何加速写的效率上。
志写对应于Oracle数据库的等待事件是log file sync，重点提高log file parallel write的效率，
以下几种思路：

- 存储物理能力提升，例如将online redo log file放置到闪存存储上
- 如果使用Solaris+Veritas FS，可以启用ODM（Oracle disk manager），让LGWR写操作穿过
esystem cache，直接写raw device，提高写入效率。(启用direct IO)
- _use_adaptive_log_file_sync=false，禁用log file sync的polling行为。
- 使用Oracle ASM。

对于Mysql和PostgreSQL，除了上述放之四海而皆准的方案1之外，我们通过前面的介绍可
知道，通过调整WAL log&innodb logfile的相关参数，可以达到异步提交到磁盘，进而增加
发事务量的目的。

对比3： Partial Page Write Protection

于Partial Page Write，所有关系型数据库都会遇到这种问题。例如数据库内部一个的最小单位是8K的page（block），而操作系统的一个os block是2k或4K，那么当遇主机突然宕机的时候，OS可能只写入了8k page中的其中4k，这部分数据可能是新数据的混合，而新数据还是不完整的，当主机重启数据库恢复时，仅依靠WAL log面的行级别的改变记录是无法完全恢复该page（block）的。

stgreSQL：

l_page_writes参数默认值为on，在checkpoint后对page的第一次修改时将该整个ge写入到WAL log里，后续对此page的修改则只需要写修改记录到WAL log里面。例crash recovery时会从checkpoint检查点开始恢复，如果有发现某个page存在rtial write的情况,则将上述写入WAL log里面的full page image覆盖此page，并继续cover WAL log即可。

以想象到，如果没有合适调整checkpoint参数，CKPT过于频繁的话，会由于此特性致WAL log的量大幅增加。

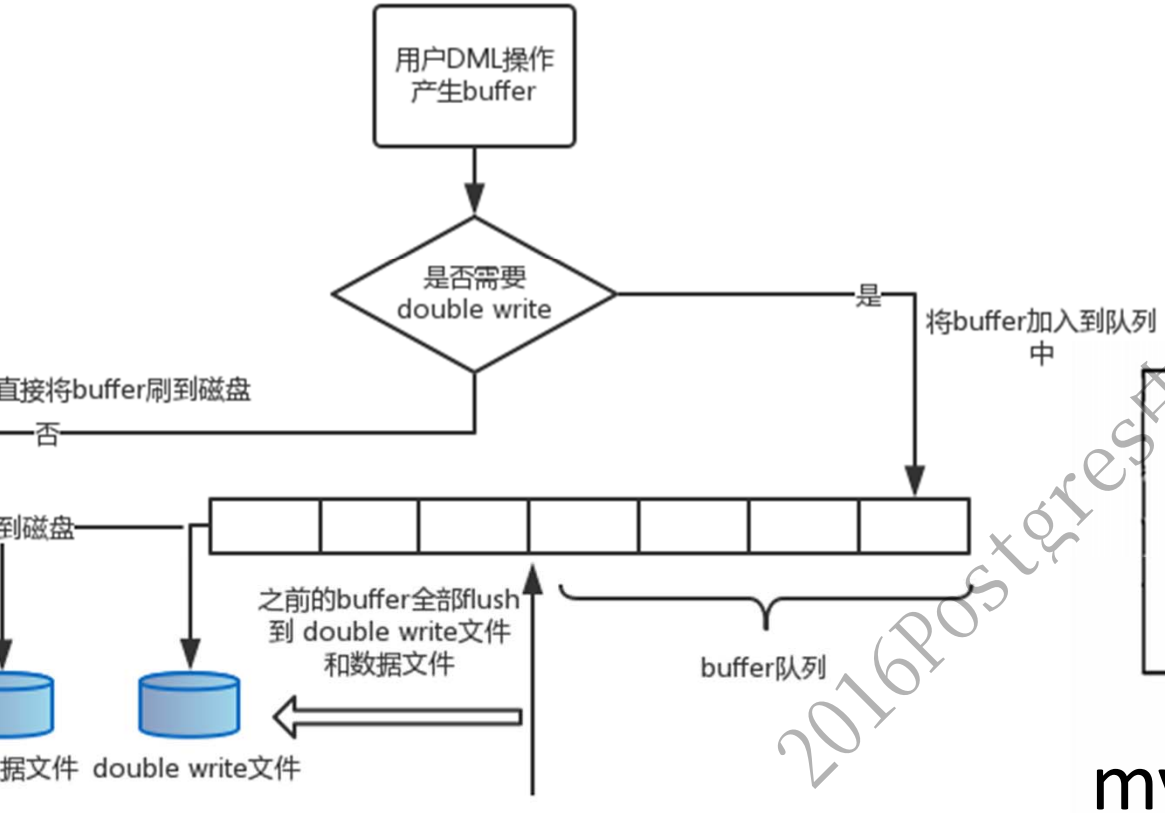
对比3： Partial Page Write Protection

ql对待该问题采用了double write，文档和书籍都有介绍，此处不赘述。

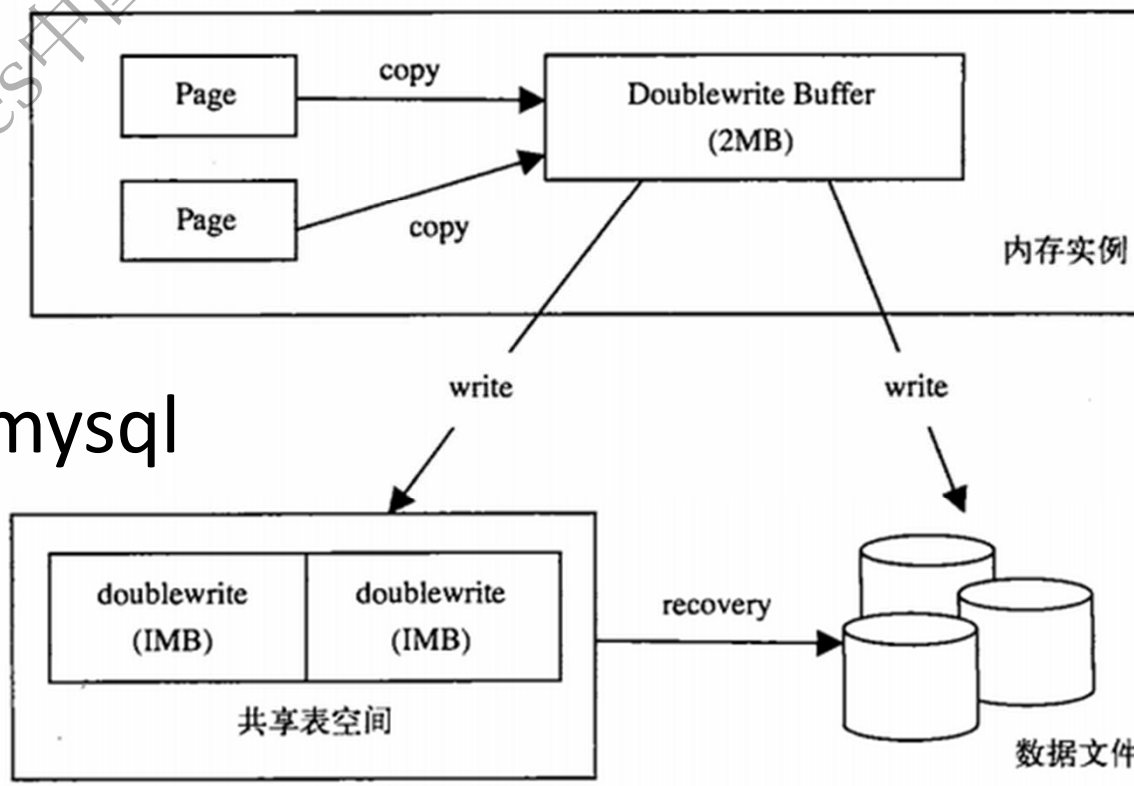
面社区里面一篇文章提到，PostgreSQL为了优化full_page_write，社区提供了一个h，它的主要设计是创建两个共享内存块队列，checkpoint专用buffer队列和double的buffer队列，同时关闭full_page_write。
s://yq.aliyun.com/articles/237

cle是否会存在这个问题呢？
会也存在，如果在crash recover的情况下，某个8k的db block底层只写了4k，Oracle一会出问题。
cle数据库如果有定期备份的话，备份文件本身是不会存在partial page write的问题的，文件加上归档日志文件就能够实现恢复，也就是说media recover可以解决此问题。另外，Oracle如果有Dataguard的话，可以通过DG来进行block的checksum和recovery。
后，Oracle的redo log的block size是与操作系统一致的。

对比3： Partial Page Write Protection



mysql



对比3： Partial Page Write Protection

ouble write的本质是什么？彻底解决问题了吗？

先将脏页复制到内存中的doublewrite buffer里面
然后从doublewrite buffer顺序将脏页写入共享表空间的物理磁盘上，立刻调用
ync同步磁盘。
然后从doublewrite buffer离散将脏页写入各个表空间的文件中。
如果步骤3的过程中主机crash，在recover时，3就可以从2里面取出该页的副本，
制到各个表空间文件中之后，再继续应用redolog进行恢复。

以，大家可以思考问题，如果步骤2也出现partial page write的情况，怎么办？

外，一些底层的手段也有助于此问题的缓解：
些文件系统例如zfs通过日志的方式保证了OS页面的完整性；
电池保护的raid卡也能确保即使os崩溃，存储缓存里面的数据也不会丢失。

对比4：调优checkpoint

	时间	多少	恢复目标	报警
Oracle	log_checkpoint_timeout	log_checkpoint_interval	fast_start_mttr_target	log_checkpoin o_alert
PG	checkpoint_timeout	max_wal_size 、 min_wal_size ~~checkpoint_segments~~	checkpoint_completion _target	log_checkpoi

注意：

PG的这个checkpoint_completion_target与Oracle的fast_start_mttr_target含义不同。

PG的是为了让写数据的IO更加平滑，避免形成密集的IO。

而Oracle是给了一个实例crash recovery的目标时间值，则Oracle自动调整checkpoint的频率，让DBWR工作得更加勤奋。可以看到Oracle设置此参数更加容易跟一个业务目标结合起来。

另外，PG 9.6新增参数：checkpoint_flush_after，定义了CKPT多少SIZE的脏块后需要强制OS去刷一次磁盘。

对比4：调优checkpoint

Mysql的checkpoint：

- **Sharp Checkpoint:**
  Innodb_fast_shutdown

- **Fuzzy Checkpoint:**
  Master Thread CKPT
  FLUSH_LRU_LIST CKPT - (innodb_lru_scan_depth)
  Async/Sync Flush CKPT
  Dirty Page too much CKPT - (innodb_max_dirty_pages_pct)

对比5：同步复制

- ✓ Oracle数据库的Dataguard有三种保护模式：

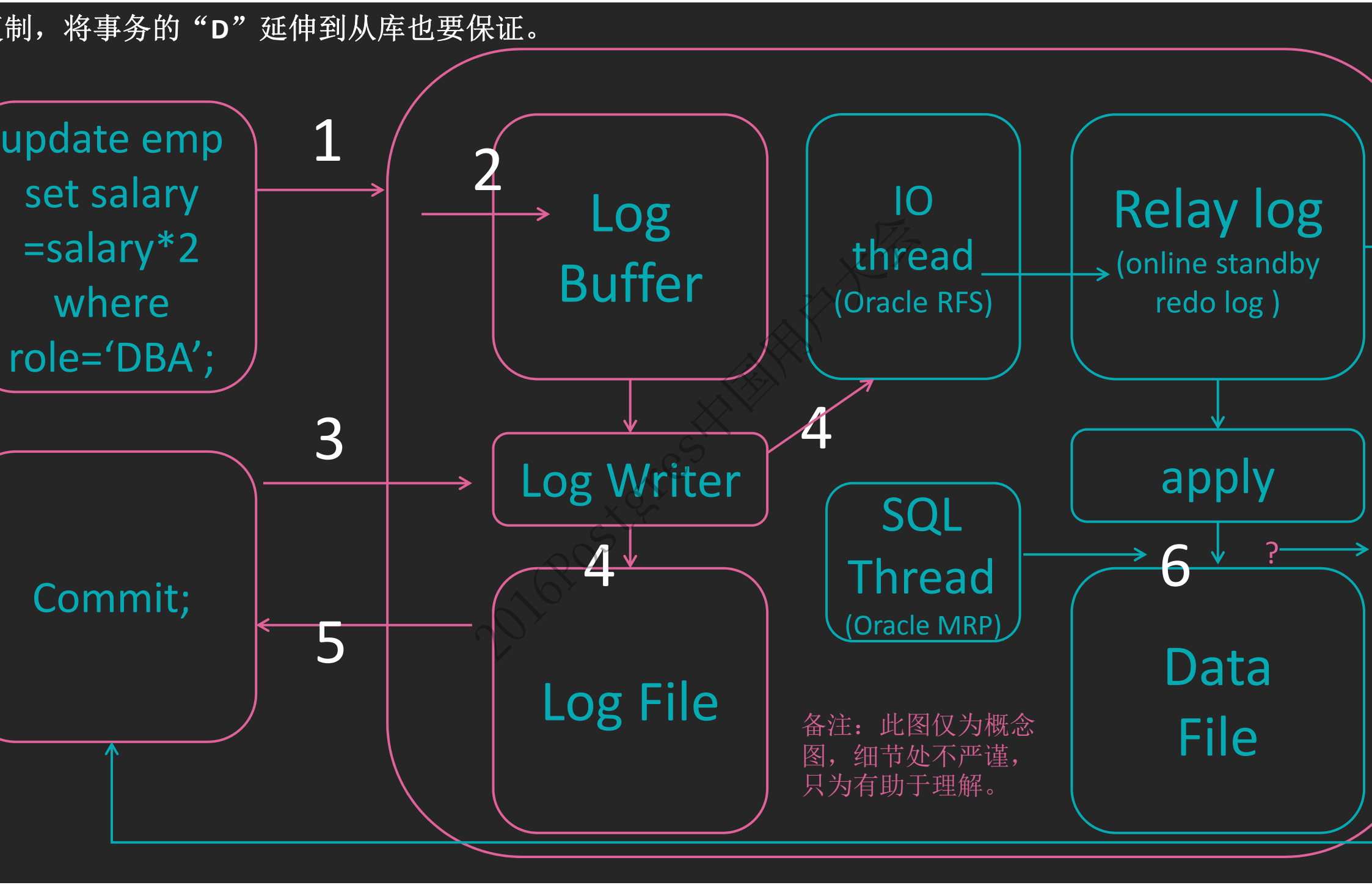
Maximum performance ---类似于异步复制
Maximum availability ---类似于半同步复制
Maximum protection ---类似于同步复制

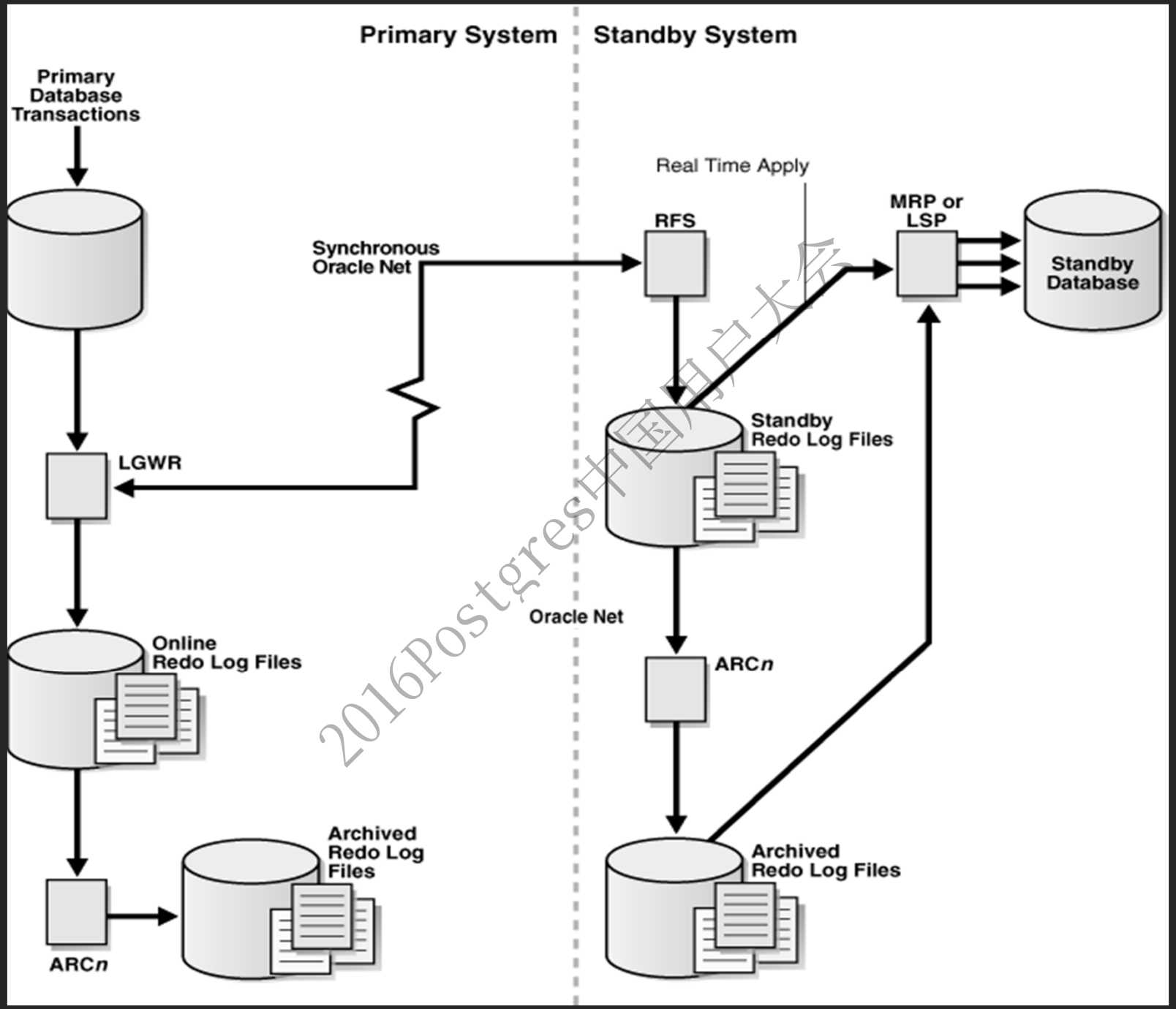
- ✓ PostgreSQL： 异步复制、同步复制

- ✓ Mysql： 异步复制、半同步复制（5.7增强型并行复制）

rpl_semi_sync_master_wait_point: AFTER_SYNC/ AFTER_COMMIT

制，将事务的“D”延伸到从库也要保证。





对比6：redo log（WAL log）的个数

- ✓ 当被redo log保护的脏块被ckpt之后，这些redo log都是可以被重用的。Oracle、Mysql、PG都是如此。
- ✓ 区别在于，Oracle、Mysql只能重用提前定义好的redo log（ib logfile）的个数以及大小。当即将被重用的redo保护的脏块还未被ckpt时，oracle会报：“checkpoint not complete”
- ✓ 而PostgreSQL的WAL log的大小虽然也是固定的，但是其个数不是一个固定值，由多个参数组合决定。当已有的xlog不能重用时，它会生成新的xlog file。(checkpoint_completion_target, checkpoint_segments & wal_keep_segments、9.5之后新增如下2参数：max_wal_size 、 min_wal_size）
- ✓ 当然，优势如果使用不当也会变成劣势，当CKPT等参数配置不当会发现pg_xlog下面会有大量的Xlog生成而无法删除，严重时甚至将磁盘空间耗尽。

对比7：进化脚印

	Mysql 5.5	Mysql 5.7/8.0
Mysql	innodb_log_file_size Defaut: 5MB Min: 1MB Max: (innodb_log_file_size *innodb_log_files_in_group) <4GB	innodb_log_file_size Defaut: 48MB Min: 4MB Max: (innodb_log_file_size *innodb_log_files_in_group) <512
ostgreSQL	PG 8.4	PG 9.6
	Default： --with-wal-segsize=16MB --with-segsize=1GB	Default： --with-wal-segsize=16MB --with-segsize=1GB

对比总结

- ✓ Oracle数据库是一个成熟的商业产品，其新版本研发方向是越来越智能化，减少人为的干预让数据库自动调优，很多控制底层行为的参数都演变成隐含参数或者干脆取消，所有隐参官方说法是：要在Oracle原厂建议下才能使用。反过来讲，Oracle越来越变成黑匣子，可供控制与个性化定制的接口会逐渐被自动化替代。

- ✓ Mysql虽然被Oracle收购，但是依旧保留原有开源的性格，很多参数都可以控制一些底层的行为，而且一些技术比较强的公司都已经开发出来自己的Mysql分支并再次开源。

- ✓ PostgreSQL彻底贯彻了开源精神，不但提供丰富的调整配置参数可以控制各个环节的行为特性、调用机制，你实在不爽的话可以去分析修改源码实现你的愿望。而且PG社区非常乐于听从用户的建议并反馈迅速，版本更新稳定迅速，9.6中大家可以看到很多原本社区期望的功能都已经推出。

一点畅想

- ✓ 软件的设计理念要与业务需求、硬件发展技术相匹配！数据库技术的飞跃离不开操作系统、存储层面的技术革新。
- ✓ 也许有一天我们可以不用care什么随机IO、顺序续IO、异步IO、同步IO等问题！
- ✓ 也许有一天关系型数据库可以抛开当前的设计理念：先写日志、后写数据！

本PPT参考了如下文章，

在此对相关厂商和作者表示感谢：

http://docs.oracle.com/database/121/CNCPT/
https://www.postgresql.org/docs/9.6/static/index.htm
http://dev.mysql.com/doc/refman
http://coding-geek.com/how-databases-work/
http://blog.itpub.net/22664653/viewspace-1063134/
http://www.orczhou.com/index.php/2009/08/innodb_flush_method-file-io/
https://en.wikibooks.org/wiki/PostgreSQL/Architecture
http://www.moeding.net/archives/37-PostgreSQL-WAL-vs.-Oracle-Redo-Log.html

Q & A