

PostgreSQL DBA培训

2013/11 (重庆)

张文升 vincent

13684090848

wensheng.zhang@postgres.cn

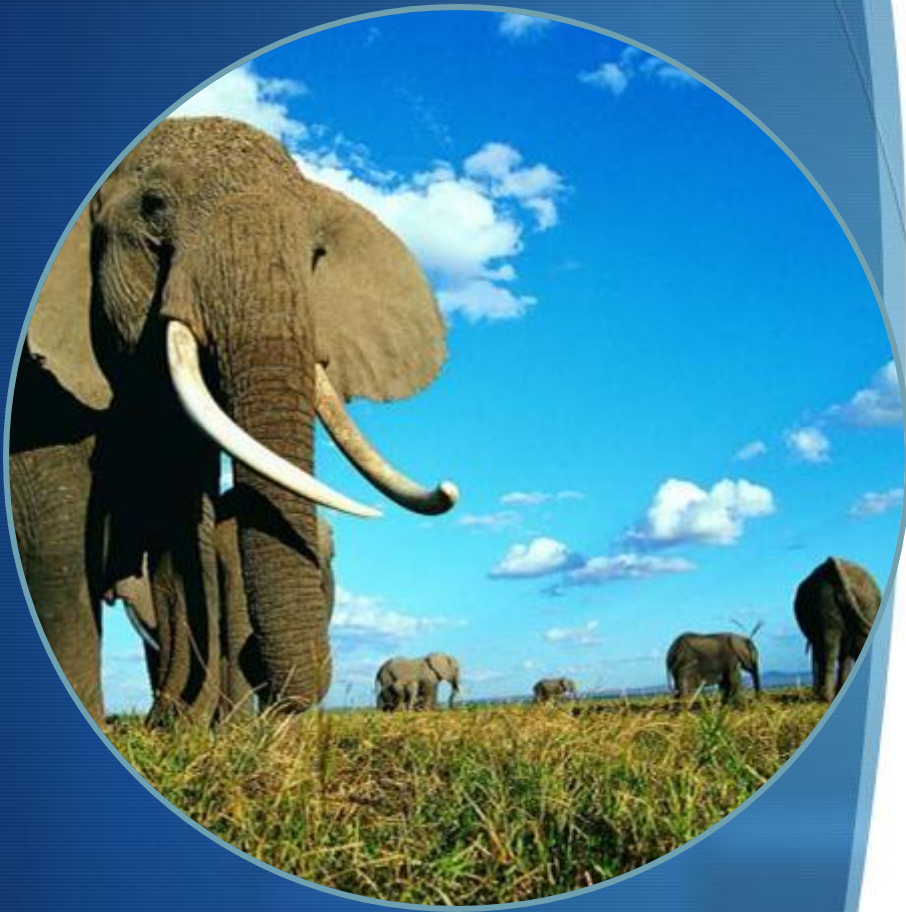
讲师个人简介



张文升

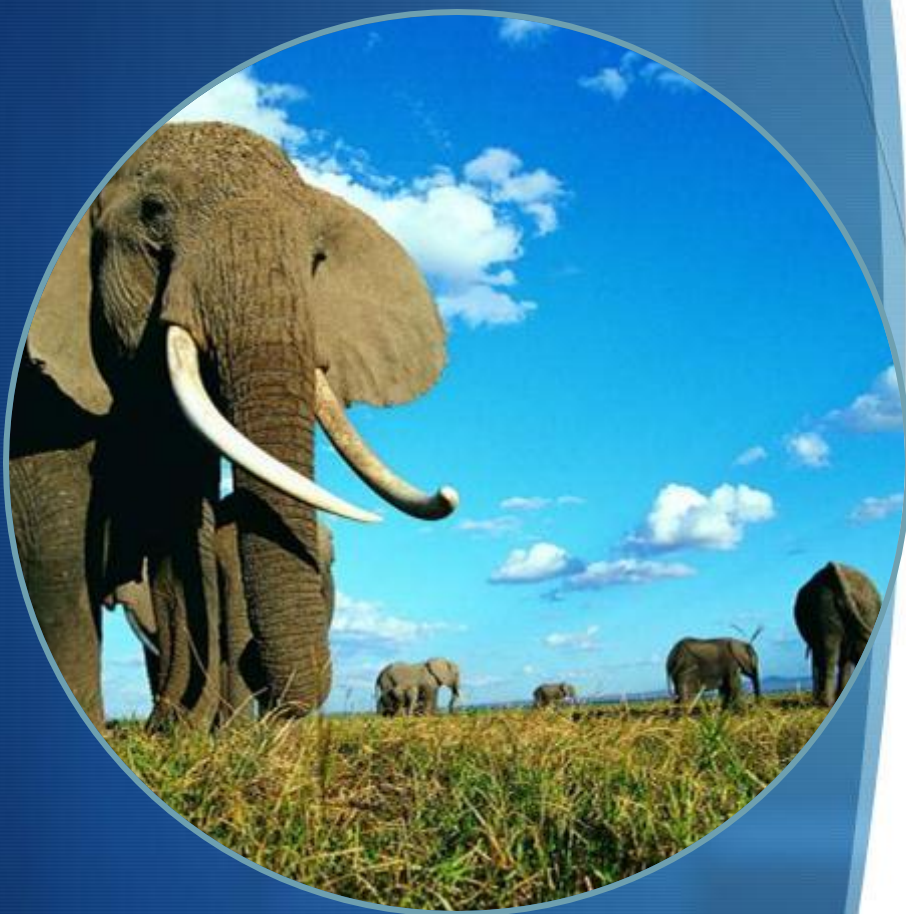
Wensheng.zhang@postgres.cn

- ◆ PostgreSQL中国社区成员。
- ◆ 10年IT从业经验，曾为武汉、云南、西安等地政府及企业开发政府行政系统、水利环保系统、油田管理系统等。
- ◆ 长期使用PostgreSQL作为业务系统的主要数据库，自2010年起参加PostgreSQL社区，推动PostgreSQL在中国地区的发展。
- ◆ 2012年10月获得：EnterpriseDB认证Postgres数据库专家



PostgreSQL 培训-Part1

- 逻辑结构基础
- 安装及文件结构
- 配置基础
- 数据库逻辑结构
- 数据备份及恢复
- 数据库权限及安全管理

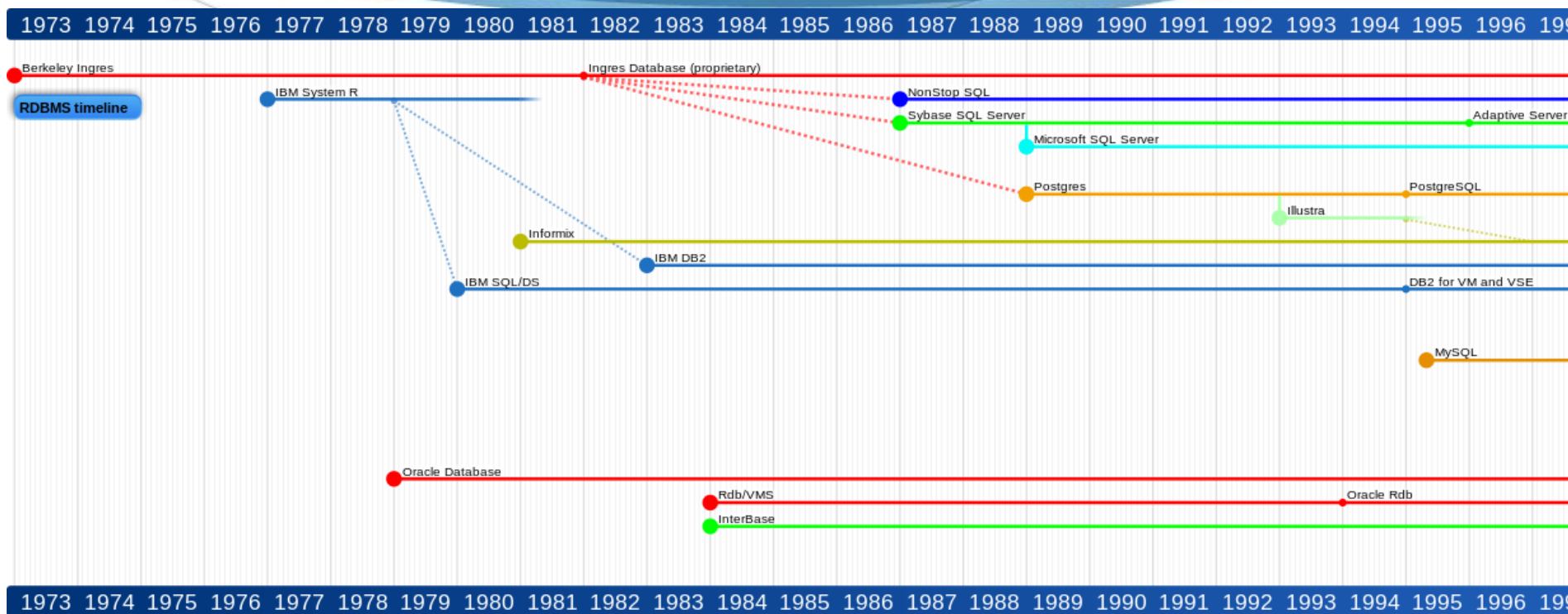


PostgreSQL 培训-Part2

- 基于时间点的数据恢复
 - 运行时维护管理
 - 数据类型
 - 存储过程及调试
 - 并发控制、JDBC、libPQ
 - 监控排错基础
- Appendixes-----
- Hot-Standby and Stream Replication简介
 - 高可用性集群架构简介
 - 读写分离架构简介
 - 集中管理及性能优化工具

PostgreSQL介绍

关系型数据库历史

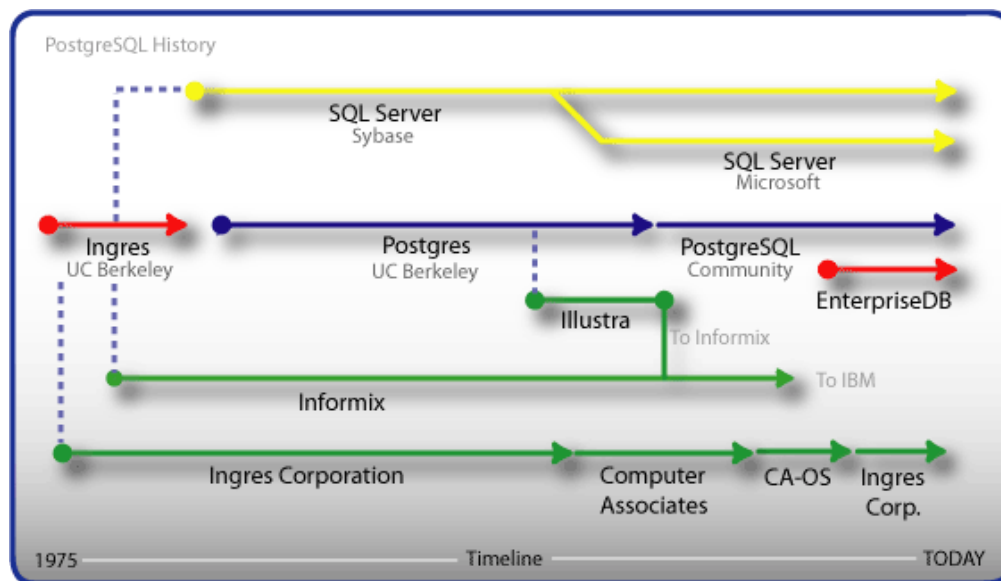


来自: http://en.wikipedia.org/wiki/File:RDBMS_timeline-2.svg

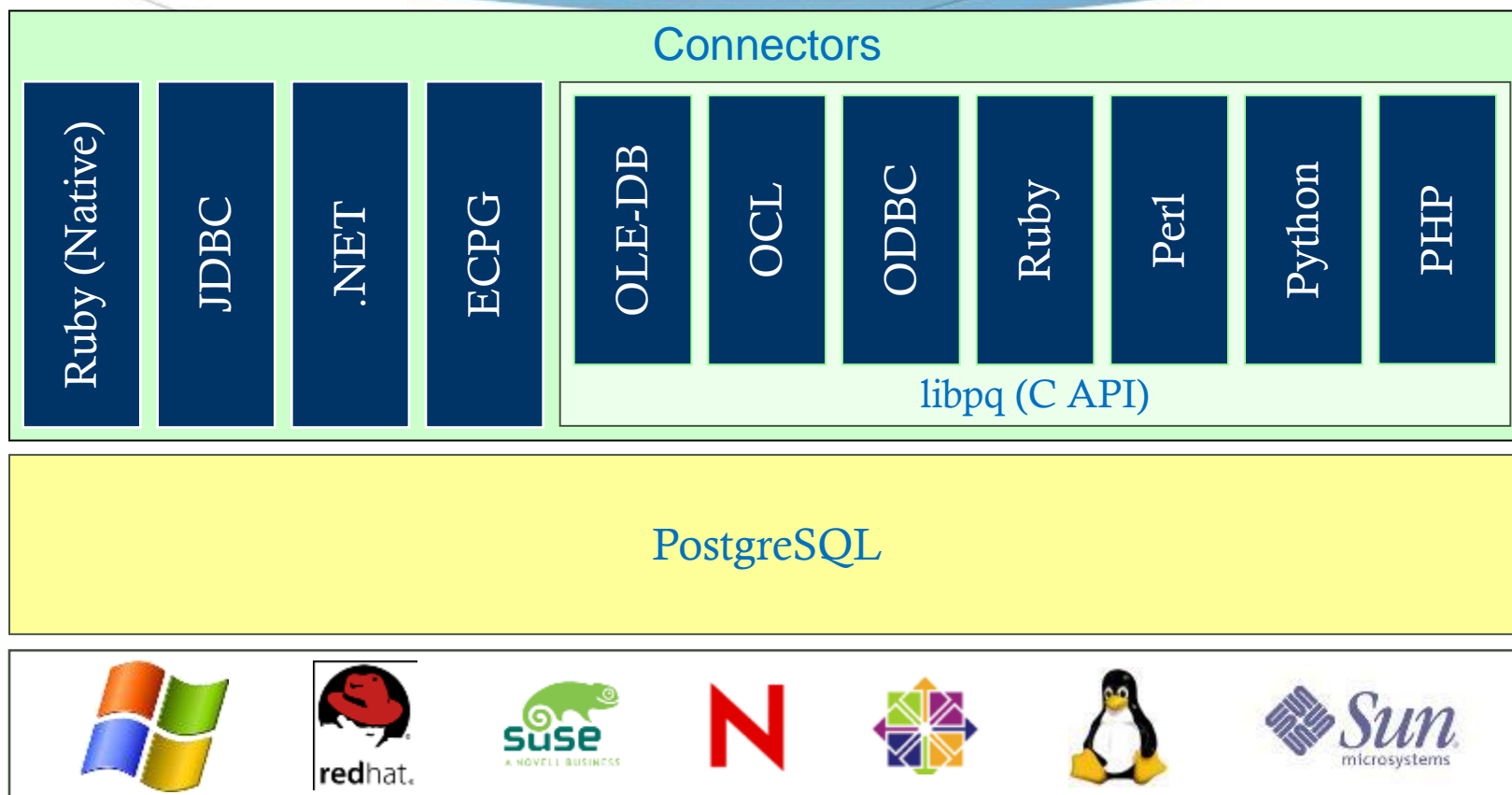
PostgreSQL数据库历史

- 发源史
- Postgres系列数据库与DB2，Oracle是同一时代的产品
- 衍生出Informix，Sybase，SQL Server等数据库，都在良性发展

- 发展史
- 1977 Ingres项目
- 1985 Post-Ingres项目
- 1995 将SQL引擎改为当时最流行的SQL92
- 之后由于社区的推动加入了很多优秀的特性，使PostgreSQL被誉为“最先进的开源数据库”
- 2004 EnterpriseDB成立商业化了PostgreSQL



PostgreSQL应用概貌



数据库限制

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows / Table	Unlimited
Maximum Columns / Table	250-1600
Maximum Indexes / Table	Unlimited

PostgreSQL术语

- 通用数据库对象名称

Industry Term	Postgres Term
Table or Index	Relation
Row	Tuple
Column	Attribute

- 存储对象名称

Industry Term	Postgres Term
Data Block	Page (when block is on disk)
Page	Buffer (when block is in memory)

谁在用PostgreSQL

NATIONAL SECURITY AGENCY



- » 经过了20多年的社区发展
- » 拥有丰富的特性并且非常稳定
- » 拥有模块化，可扩展性，高性能的软件架构
- » 适合于大型的，混合负载的应用
- » 开源社区蓬勃发展，拥有众多活跃的项目



重庆机场集团有限公司
Chongqing Airport Group Co., Ltd.

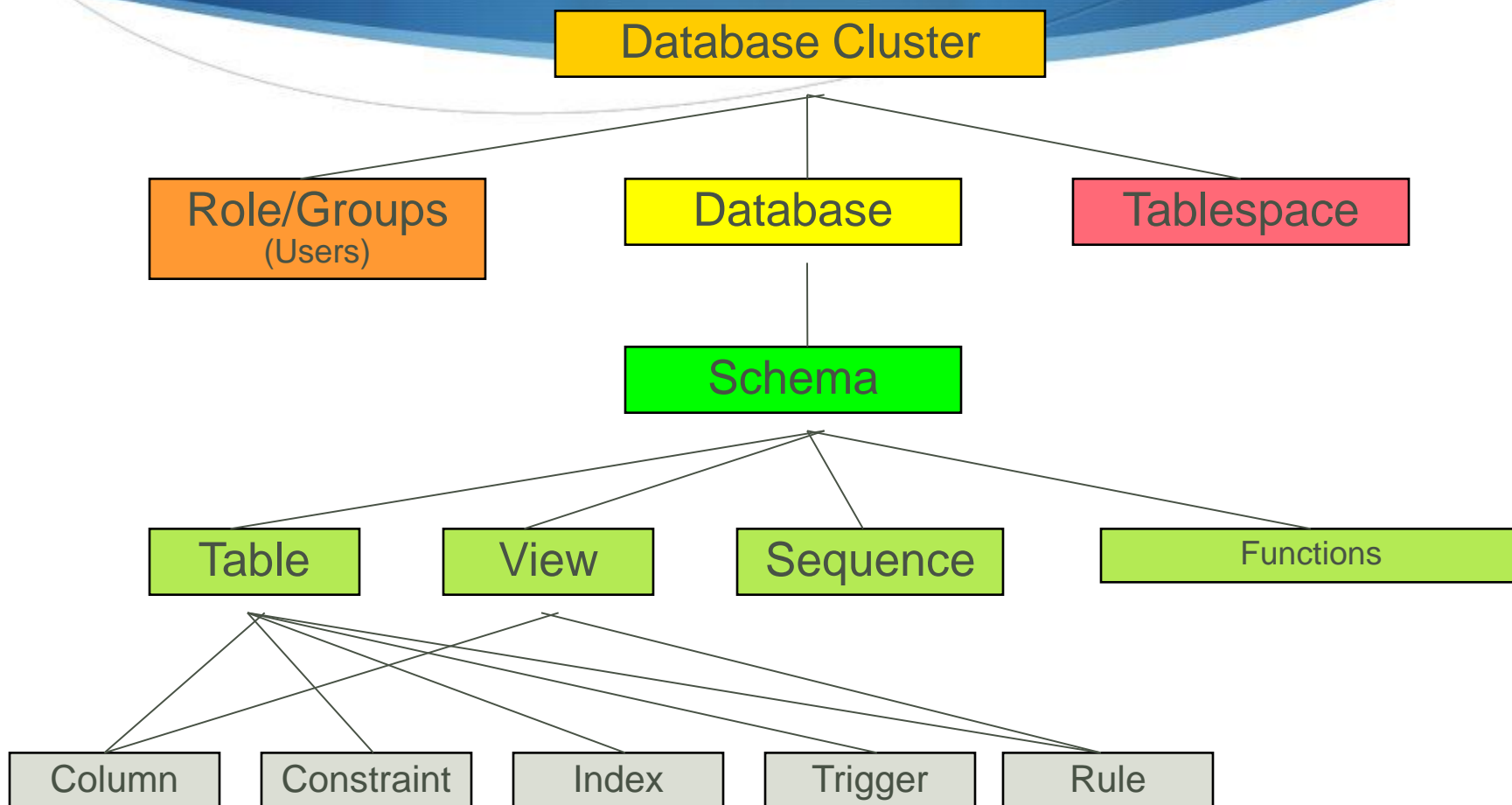


PostgreSQL逻辑存储结构

数据库&实例&集群

- ◆ 数据库（Database）
 - ◆ 特指存储数据和相关对象的物理文件
- ◆ 实例（Instance）
 - ◆ 一系列操作系统进程和由这些进程管理的内存区
- ◆ 集群（Cluster）
 - ◆ 一系列数据库的集合，集群中的这些数据库通过单个数据库服务器的实例管理。

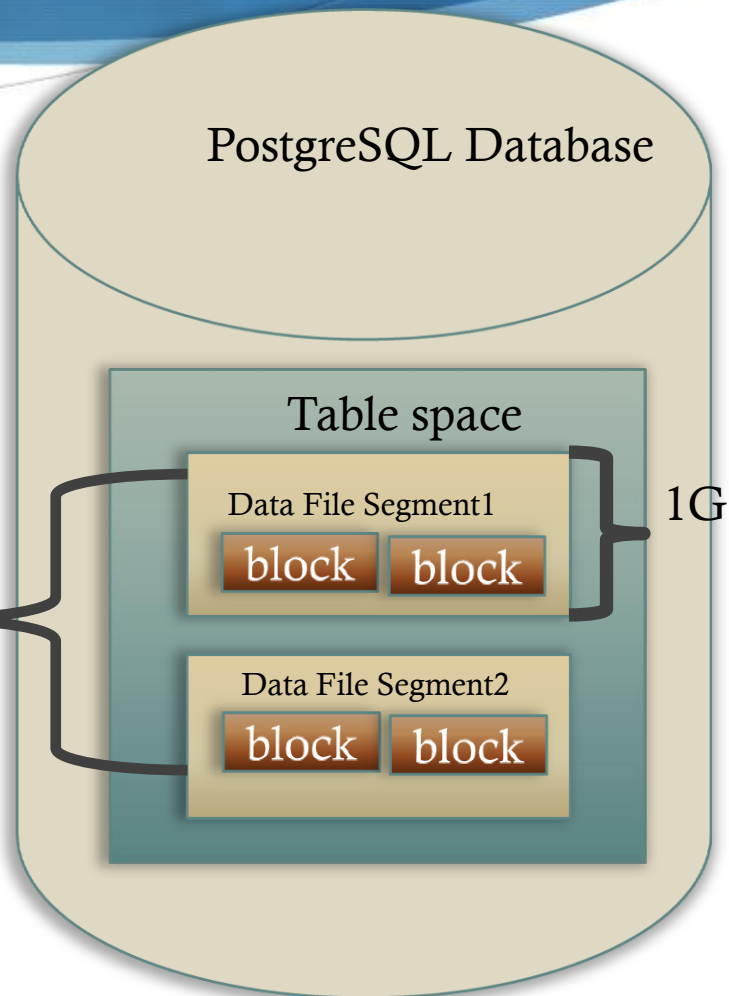
数据库对象层级结构



存储结构

- 表空间 (**Table space**)
 - 保存数据文件
- 数据文件 (**Data file**)
 - 表存放在一个单独的文件
 - 如果文件超过1G则存放为多个文件
- 块 (**Block**)
 - 文件被划分为多个块 (8k)

table



安装及文件结构

操作系统用户及权限

- ◆ PostgreSQL 在Unix / Linux上作为一个后台进程 (daemon) ，在Windows上作为一个服务 (Service)
- ◆ 所有的PostgreSQL 的进程以及数据文件必须属于一个OS的用户
 - ◆ OS 用户和数据库用户账户没有关系 (un-related)
 - ◆ 出于安全的考虑，OS 用户不能是root或具有操作系统管理权限的帐号
 - ◆ 在 Windows 上只支持NTFS(只有NTFS下才支持建立表空间) 。

安装方式

- ◆ 跨平台图形化二进制安装包
 - ◆ <http://www.postgresql.org/download/linux/#oneclick>
- ◆ RPM & DEB二进制安装包(不支持Windows)
 - ◆ <http://www.postgresql.org/download/linux/#multibinary>
- ◆ 操作系统自带安装包
- ◆ 源代码编译安装





Stack Builder 3.1.0



Welcome to Stack Builder!

This wizard will help you install additional software to complement your PostgreSQL or EnterpriseDB Postgres Plus installation.


To begin, please select the installation you are installing software for from the list below. Your computer must be connected to the Internet before proceeding.

PostgreSQL 9.1 on port 5 ▾

Proxy servers

< Back

Next >

 Cancel



Stack Builder 3.1.0



Please select the applications you would like to install.

- ☒ Spatial Extensions
 - ☒ PostGIS 1.5 for PostgreSQL 9.1 v1.5.3-1
- ▼ ☒ Web Applications
 - ☐ Drupal 6 v6.19-1
 - ☒ Drupal 7 v7.7-1 (installed)
 - ☒ mediaWiki v1.17.0-1 (installed)
 - ☒ phpBB v3.0.9-1
- ▼ ☒ Web Development
 - ☒ Apache/PHP v2.2.20-5.3.8-1 (installed)
 - ☒ phpPgAdmin v5.0.2-1

PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems (GIS). PostGIS follows the OpenGIS

< Back

Next >

Cancel

PostgreSQL目录结构

/opt/PostgreSQL/9.1

| _____bin

| _____data

| | _____base

| | _____global

| | _____pg_clog

| | _____pg_xlog

| | _____pg_tblspc

| | _____postgresql.conf pg_hba.conf

| _____doc

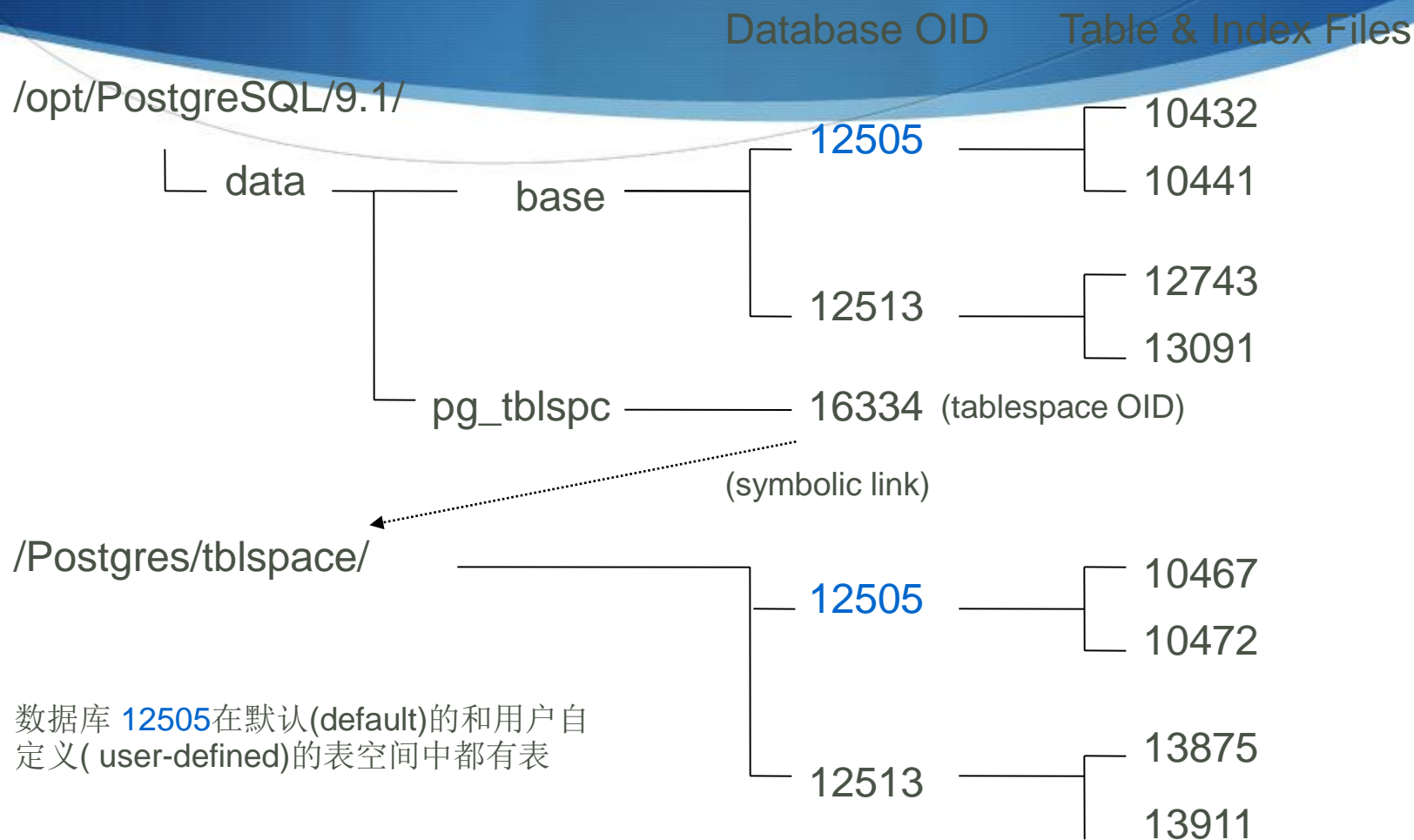
| _____include

| _____installer

| _____lib

| _____share

PostgreSQL数据目录结构



数据文件存储方式

- 每一个表和索引都存放到单独的数据文件中
 - 文件名是表或索引的文件结点(filenum)编号
 - 文件结点(filenum)编号可以在pg_class.relfilenum中查看
- 段(Segments)
 - 如果表或索引超过 1 GB 就会被分割为 gigabyte-sized 大小的段
 - 第一个段以文件结点(filenum)编号命名
 - 第二个以及之后的段以如下形式命名：
filenum.1, filenum.2, 等等.

设置环境变量

- ◆ 设置环境变量将减少许多在数据库服务器启动和关闭中的问题.
 - ◆ PATH - 必须指向正确的 bin 目录.
 - ◆ PGDATA - 必须指向正确的数据集合实例目录.
 - ◆ PGPORT - 必须指向正确的数据集合实例运行的端口.
 - ◆ 修改 .bash_profile 文件, 设置变量
 - ◆ 在 Windows 上, 通过我的电脑的属性页设置这些环境变量.

数据集合实例

- ◆ 每一个 PostgreSQL 的实例和一个”数据集合实例”关联
- ◆ 由数据目录组成，目录中包含了所有的数据文件和配置文件
- ◆ 通过两种方式引用：
 - ◆ 数据目录的位置
 - ◆ 端口号
- ◆ 一个服务器可以管理多个数据集合实例

手动创建数据集合实例

- 利用initdb 创建数据集合实例。必须作为OS用户运行，这个用户也将运行时间就看实例

```
initdb -D <data directory>  
-D <data directory> - Database cluster directory  
-U <super user> - Select the database super user name  
-E <encoding> - Specify the database encoding  
-n - No clean (do not clean up files in case of failure)
```

- 创建了一个新的数据集合实例之后，修改 postgresql.conf 和 pg_hba.conf 文件
- 确保在postgresql.conf文件中为这个数据集合实例分配了一个唯一的端口号(port #)

PostgreSQL启动及停止(1)

💧 Linux:

- 💧 `/etc/init.d/postgresql-9.1`
`<start|stop|restart|status|reload>`
- 💧 `service postgresql-9.1`
`<start|stop|restart|status|reload>`

💧 Windows:

- 💧 通过“开始->控制面板->管理工具->服务”

💧 通用方式:

- 💧 `$PGHOME/bin/pg_ctl <start|stop|restart|status|reload>`
`[options]`

PostgreSQL启动及停止(2)

◆ 常用的 pg_ctl 参数

- ◆ -D <data directory> - 数据集合实例目录
- ◆ -l <log file> - 将日志输出到指定的文件中
- ◆ -m smart - 等待客户端断开连接（默认）
- ◆ -m fast - 未完成的 tx 回滚，断开客户端连接
- ◆ -m immediate - 强行终止进程，数据库没有干净的关闭

使用psql连接到数据库

`psql [OPTIONS]... [DBNAME [USERNAME]]`

被连接的数据库也可以通过指定 `-d DBNAME` 选项设置。连接的用户可以指定 `-U` 参数设置。这些在命令行下将覆盖 `[DBNAME [USERNAME]]`。

```
pgdba2000:/opt/PostgreSQL/9.1/bin # ./psql -U postgres template1
Password for user postgres:
psql.bin (9.1.3)
Type "help" for help.
```

```
template1=#
```

在一个 `psql` 会话中，连接将通过如下命令改变 “`\c[onnect] [DBNAME [USERNAME]]`”

```
template1=# \c postgres test
You are now connected to database "postgres" as user "test".
postgres=>
```

`DBNAME` 和 `USERNAME` 默认是操作系统用户

题目

- 1. 安装一个Postgres，要求如下：
 - Data安装在/edb1_data目录下
 - 要求默认最佳性能，用于OLAP操作
- 2. 在本机中添加一个“数据集(Database Cluster)”，要求如下：
 - 可以与默认安装的“数据集”同时运行
 - Data安装在/edb2_data目录下

配置文件基础

配置文件及参数

- ◆ 主要的配置文件 `postgresql.conf`
- ◆ `postgresql.conf`通常放在数据目录中
- ◆ 每个数据集合实例都有一个`postgresql.conf`文件
- ◆ 部份参数只有进行`restart`才生效

在Session调整系统参数

- 部份参数可通过SET命令在session中进行设置
 - postgres=# SET work_mem = 8192;
- SHOW命令可以用于查看当前环境中某个参数的值
 - postgres=# SHOW work_mem;
- 通过pg_settings数据字典可以列出所有参数的信息
 - postgres=# SELECT * FROM pg_settings;

连接设置

- 🍯 postgresql.conf 文件中如下信息决定了连接设置：
 - 🍯 listen_addresses (string)
 - 🍯 设置服务器监听的用于连接的TCP/IP 地址
 - 🍯 port (integer)
 - 🍯 设置服务器监听的TCP 端口；默认是5432.
 - 🍯 max_connections (integer)
 - 🍯 决定了连接到数据库服务器的最大并发连接数
 - 🍯 superuser_reserved_connections (integer)
 - 🍯 决定了为超级用户连接而保留的连接“槽位”
 - 🍯 unix_socket_directory (string)
 - 🍯 设置服务器监听连接的Unix域套接字(Unix-domain socket)的目录。默认通常是 /tmp
 - 🍯 unix_socket_group (string)
 - 🍯 设置Unix域套接字(Unix-domain socket)的所属组.
 - 🍯 unix_socket_permissions (integer)
 - 🍯 设置Unix域套接字(Unix-domain socket)的访问权限.

安全和认证设置

- ◆ postgresql.conf 文件中如下信息决定了安全和认证设置
 - ◆ authentication_timeout (integer)
 - ◆ 设置完成客户端认证的最长时间，以秒为单位。
 - ◆ ssl (boolean)
 - ◆ 开启 SSL 连接。
 - ◆ ssl_ciphers (string)
 - ◆ 设置一个 SSL 一系列在安全连接中允许被使用的密码。
 - ◆ password_encryption (boolean)
 - ◆ 当使用命令CREATE USER 或 ALTER USER 设置密码而没有写 ENCRYPTED 或 UNENCRYPTED是, 这个参数将决定密码是否要加密. 默认是on (对密码加密)。

内存设置

- ◆ postgresql.conf 文件中如下信息决定了内存设置:
 - ◆ shared_buffers (integer)
 - ◆ 设置数据库服务器使用的作为shared memory buffers的内存大小.
 - ◆ temp_buffers (integer)
 - ◆ 设置每个数据库会话使用的temporary buffers 的最大值.
 - ◆ max_prepared_transactions (integer)
 - ◆ 设置可以同时的“准备(prepared)” 状态的最大事务数
 - ◆ work_mem (integer)
 - ◆ 设置内部排序操作和 Hash 表在开始使用临时磁盘文件之前使用的内存大小
 - ◆ maintenance_work_mem (integer)
 - ◆ 设置在维护性操作时使用的最大内存数, 例如 VACUUM, CREATE INDEX, 和 ALTER TABLE ADD FOREIGN KEY.

内部资源设置

- 💧 postgresql.conf 文件中如下信息决定了内部资源设置：
 - 💧 shared_preload_libraries (string)
 - 这个变量声明一个或者多个在服务器启动的时候预先装载的共享库。
 - 💧 **Note:**通过预先装载一个共享库，我们就可以避免第一次使用这个库的加载时间。不过，启动每个服务器进程的时间可能会增加，即使进程从来没有使用过这些库也这样。因此我们只是建议对那些将被大多数会话使用的库才使用这个选项。

日志配置(1)

`log_destination: stderr (default), syslog, eventlog(Windows)`

`redirect_stderr: true/false` - 捕获并将stderr 发送到log文件中

`log_directory`: 当`redirect_stderr` 开启的时候, 它设置日志文件的位置

`log_filename`: 当`redirect_stderr` 开启的时候, 它设置日志文件的名称

`log_rotation_age`: log的生命周期(以分钟为单位), 当超过此时间就会创建一个新的log文件

`log_rotation_size`: log的最大大小(以KB为单位), 当超过此大小就会创建一个新的log文件

`log_truncate_on_rotation`: 在基于时间的日志循环中, 新的日志信息覆盖原日志, 而不是追加

日志配置(2)

`client_min_messages`:控制什么样的信息级(message levels)别需要发送到客户端。
默认是NOTICE。

`log_min_messages`:控制什么样的信息级(message levels)别需要发送到服务器端。
默认是NOTICE。

`log_error_verbosity`: TERSE, DEFAULT or VERBOSE

`log_min_error_statement`: 控制sql语句导致的错误是否记录到服务器的log中，默认是ERROR.

`log_min_duration_statement`:确定当超过多少milliseconds 就对语句进行log记录。

日志配置(3)

`log_connections`: 每一次成功的连接都被记录到服务器日志中

`log_disconnections`: 每一次断开连接都被记录到服务器日志中

`log_duration`: 设置时间间隔, 当语句的执行时间超过此时间间隔就会被记录.

`log_line_prefix`: 允许你在每个日志行的开始输出其他的一些变量, 如用户名, 进程id, 时间戳等.

`log_statement`: 控制什么样的语句将会被记录。合法的值包括ddl, mod 或者 all.

后台写进程设置

- ◆ 有一个独立的服务器进程叫做后台写进程(background writer)。它的功能是实现对共享区的“脏(dirty)”数据的写操作。
- ◆ 设置后台写进程的参数：
 - ◆ bgwriter_delay (integer)
 - ◆ 设置后台写进程活跃轮回之间的延迟。默认值是 200 milliseconds
 - ◆ bgwriter_lru_maxpages (integer)
 - ◆ 在每个轮回里，不超过这么多个缓冲区将被后台写进程写入磁盘。将此值设置为0将关闭后台写(检查点触发除外)缺省值是 100个缓冲区(page)。

VACUUM开销设置

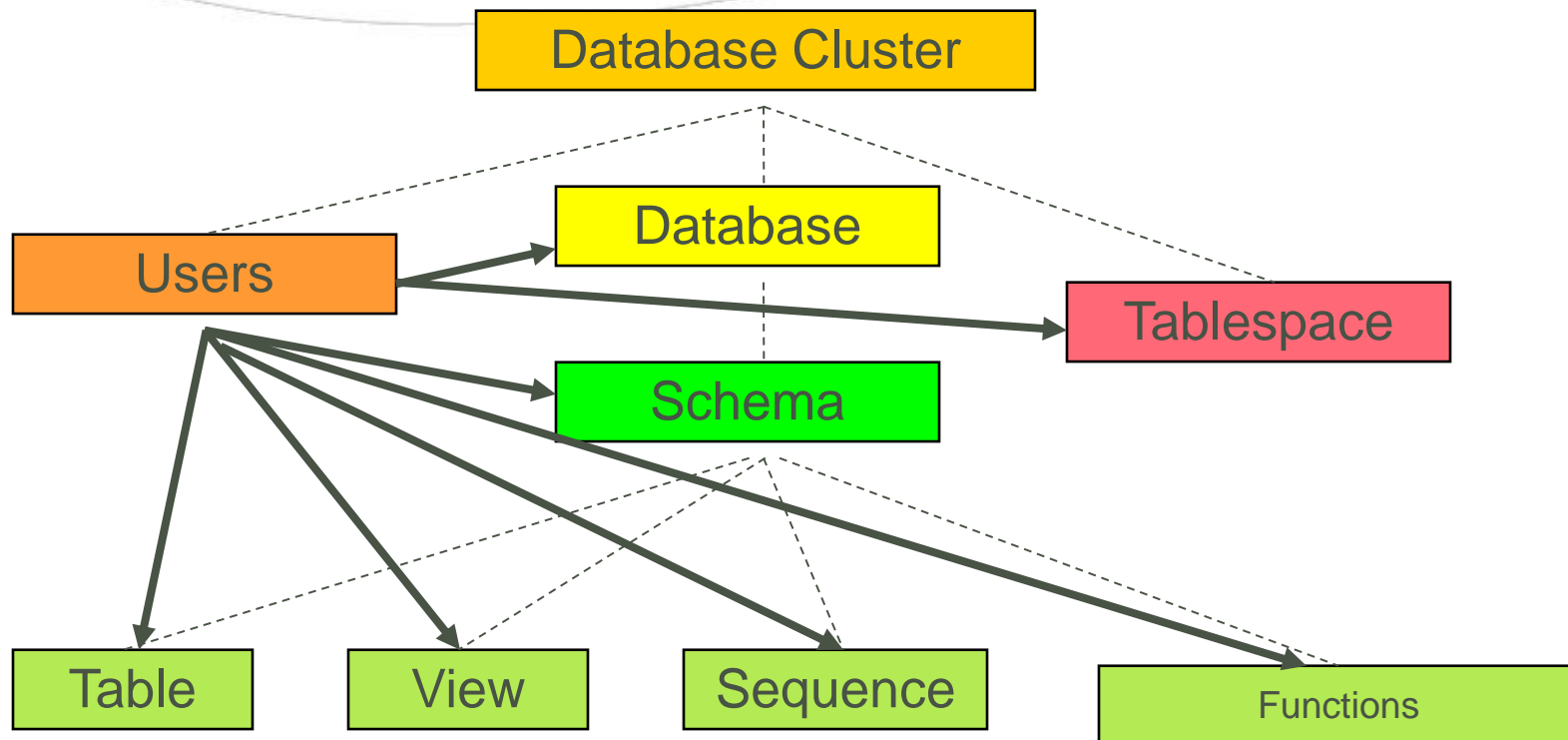
- 在执行 VACUUM 和 ANALYZE 命令时，如下参数允许管理员减少 I/O对并发数据库的影响：
 - `vacuum_cost_delay (integer)`
 - 以毫秒(milliseconds)为单位，如果超过了开销限制，那么进程将睡眠一会儿。
 - `vacuum_cost_page_hit (integer)`
 - 清理一个在共享缓存里找到的缓冲区的预计开销。
 - `vacuum_cost_page_miss (integer)`
 - 清理一个要从磁盘上读取的缓冲区的预计开销。
 - `vacuum_cost_page_dirty (integer)`
 - 清理修改一个原先是干净的块的预计开销。
 - `vacuum_cost_limit (integer)`
 - 导致清理进程休眠的积累开销。

AutoVACUUM设置

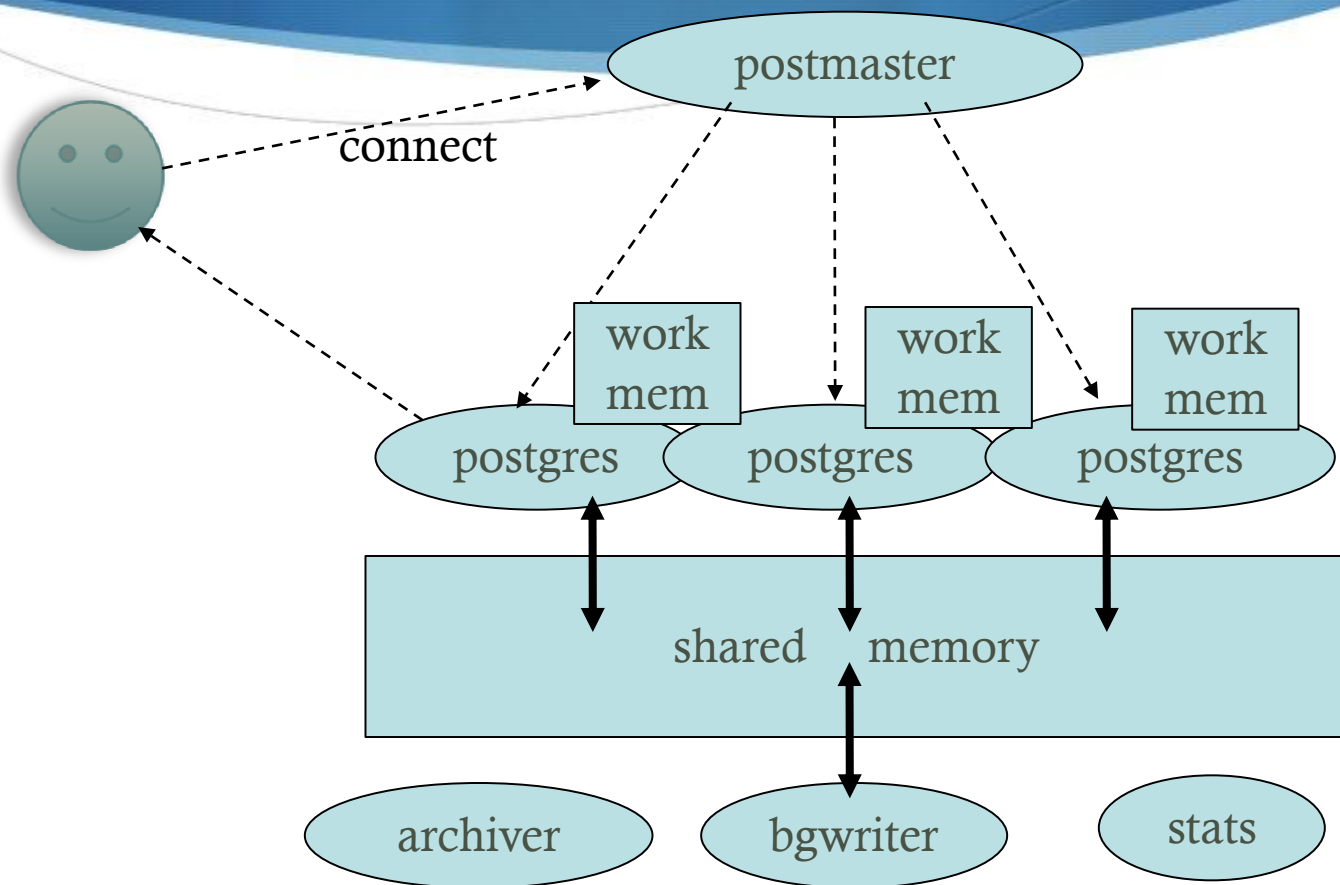
- ◆ 如下设置控制了autovacuum 特性的行为:
 - ◆ autovacuum (boolean)
 - ◆ 控制服务器是否需要运行自动清理守护进程 (autovacuum launcher daemon)。默认是on
 - ◆ log_autovacuum_min_duration (integer)
 - ◆ 当自动清理 (autovacuum) 执行超过一定时间 (milliseconds) 之后，需要对每一个自动清理执行的操作进行日志记录。当设置此值为0时，将记录所有的自动清理操作 (autovacuum actions)
 - ◆ autovacuum_max_workers (integer)
 - ◆ 设置在任何一个时间点允许存在的自动清理进程 (而不是自动清理守护进程 (autovacuum launcher)) 默认是3.

数据库核心架构详解

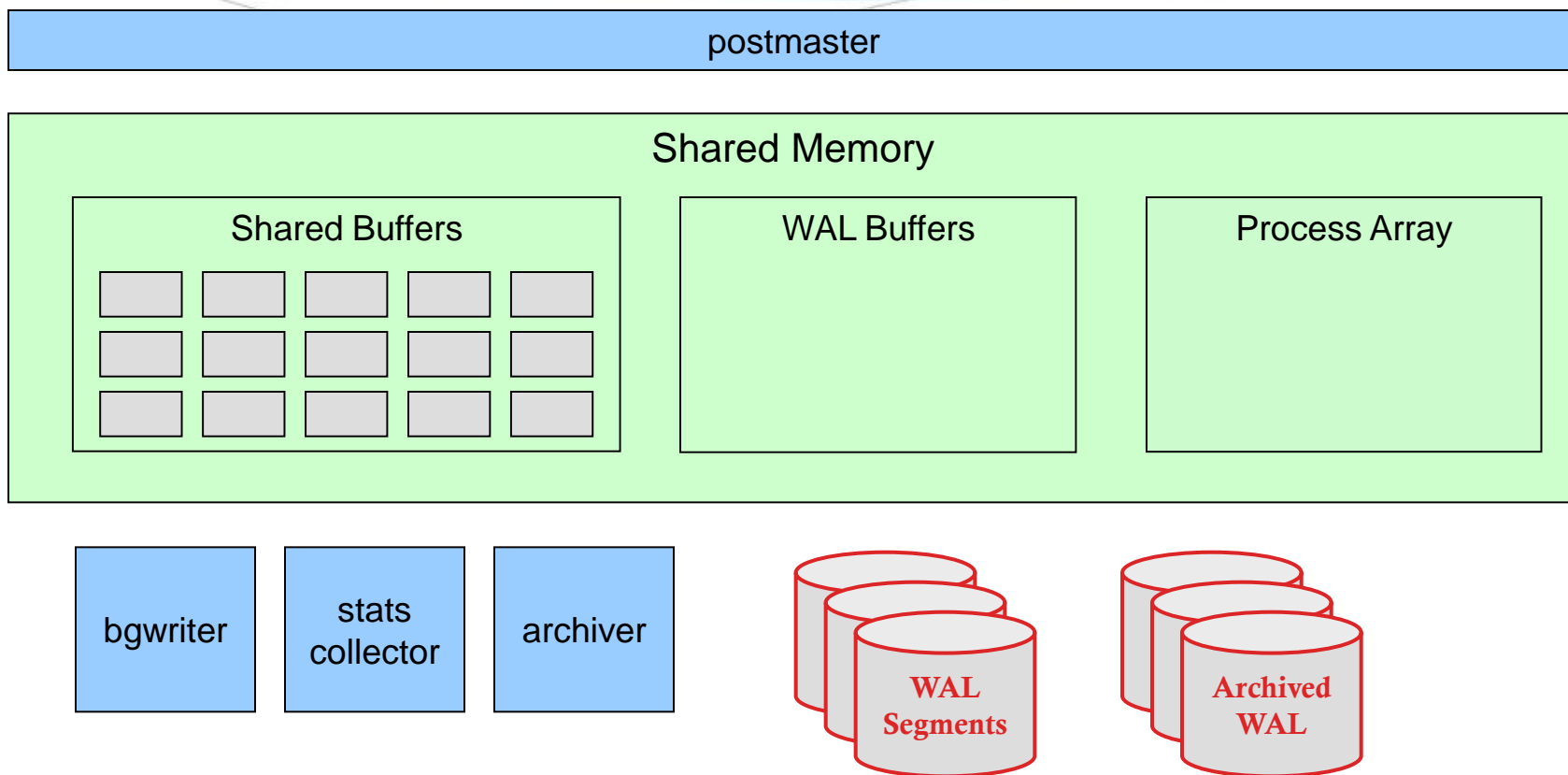
各对象的主属关关系



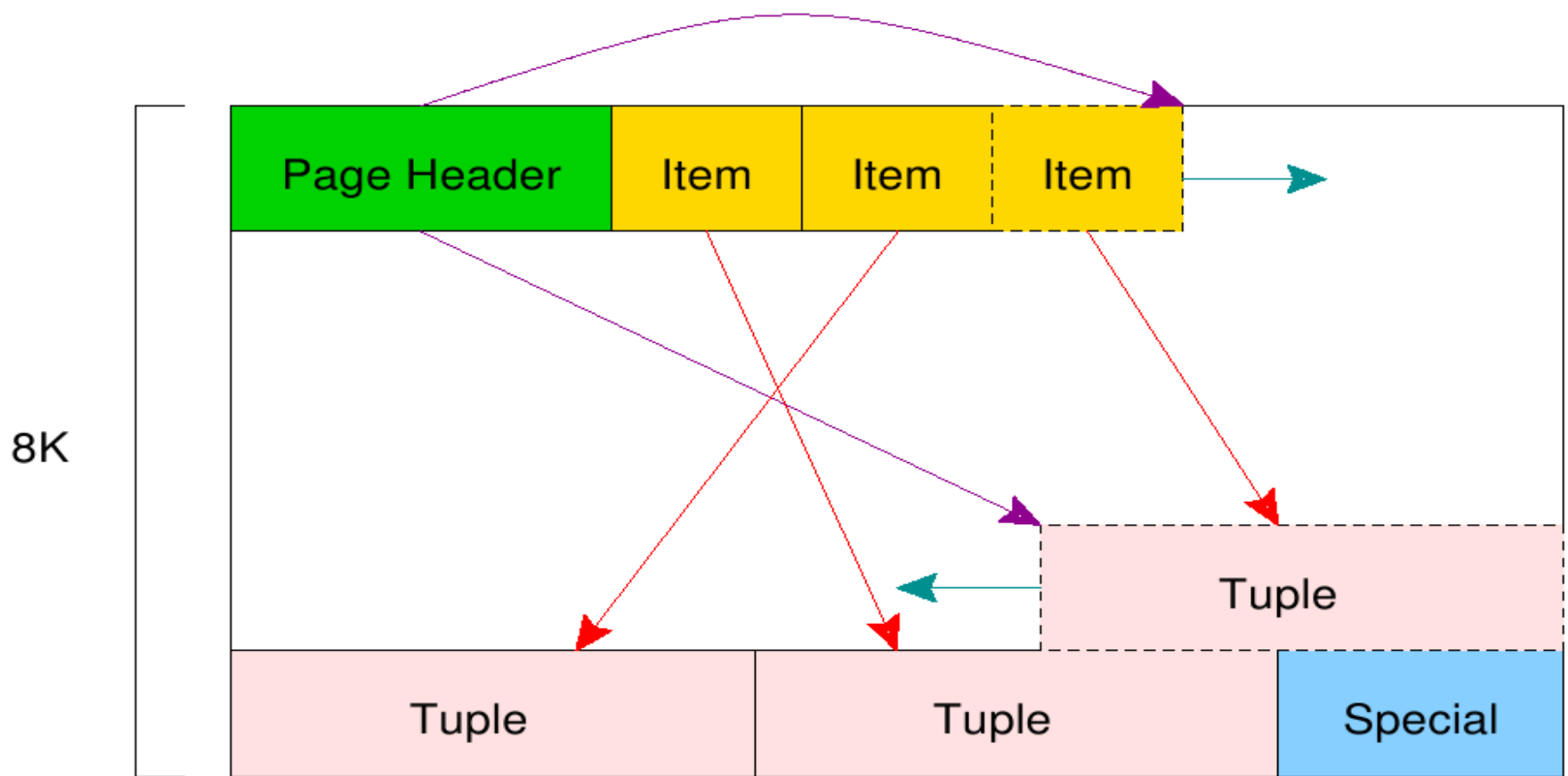
用户连接及进程关系



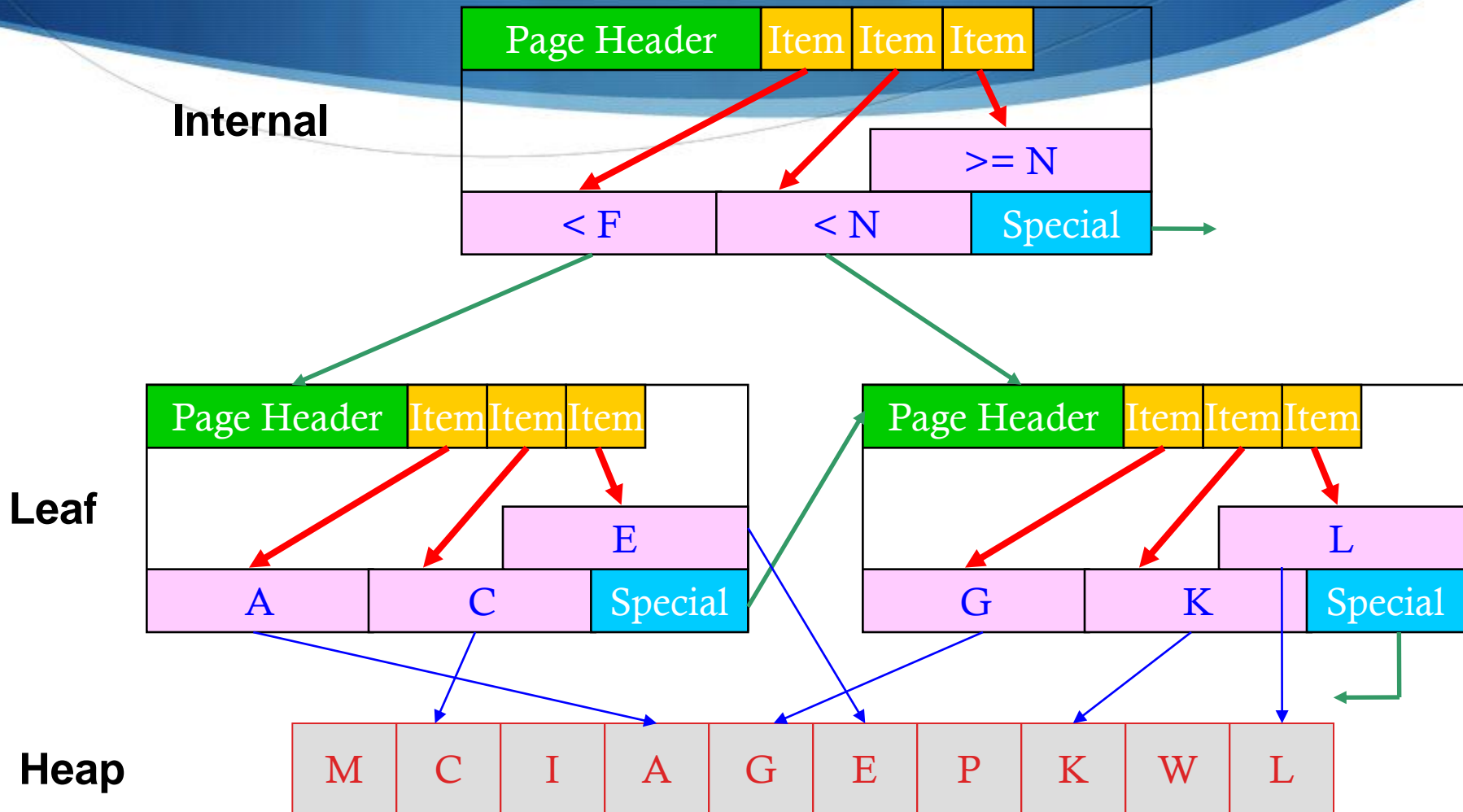
进程及内存



PostgreSQL页结构



索引页结构

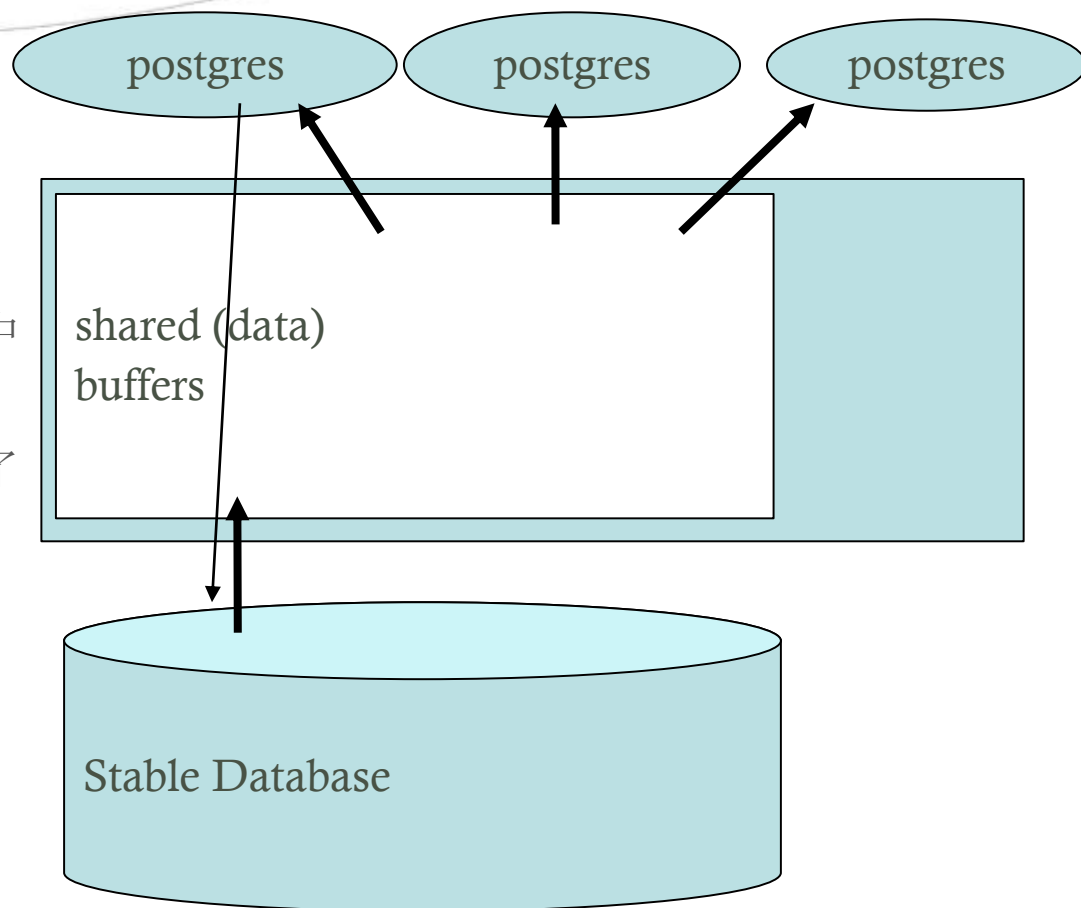


磁盘读入缓存

- postmaster

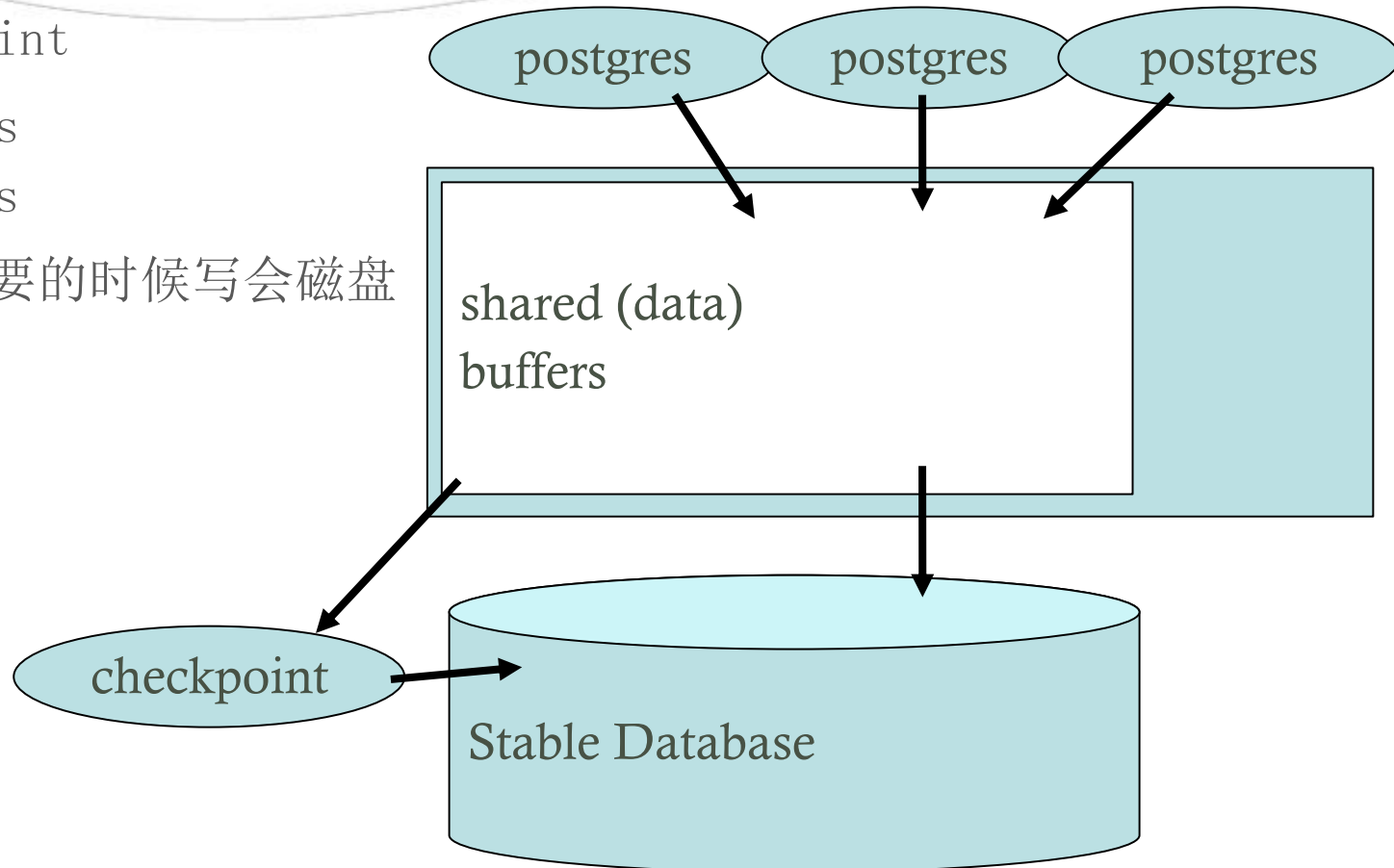
- postgres backends

- 一个 read I/O
 - 许多逻辑读(logical reads)是从缓冲区(buffer cache)中获得
 - cache 管理在 8.0版本中有了很大的提升



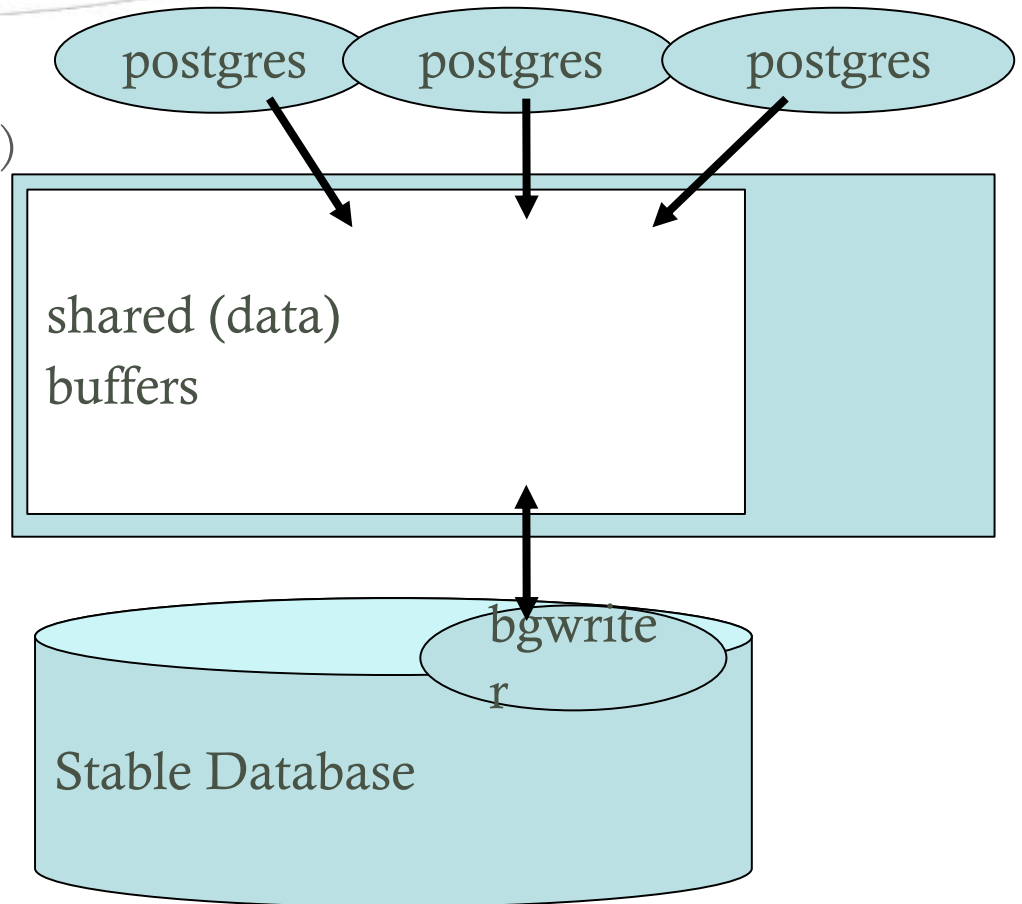
写缓存

- postmaster
 - checkpoint
 - postgres backends
 - 当需要的时候写会磁盘



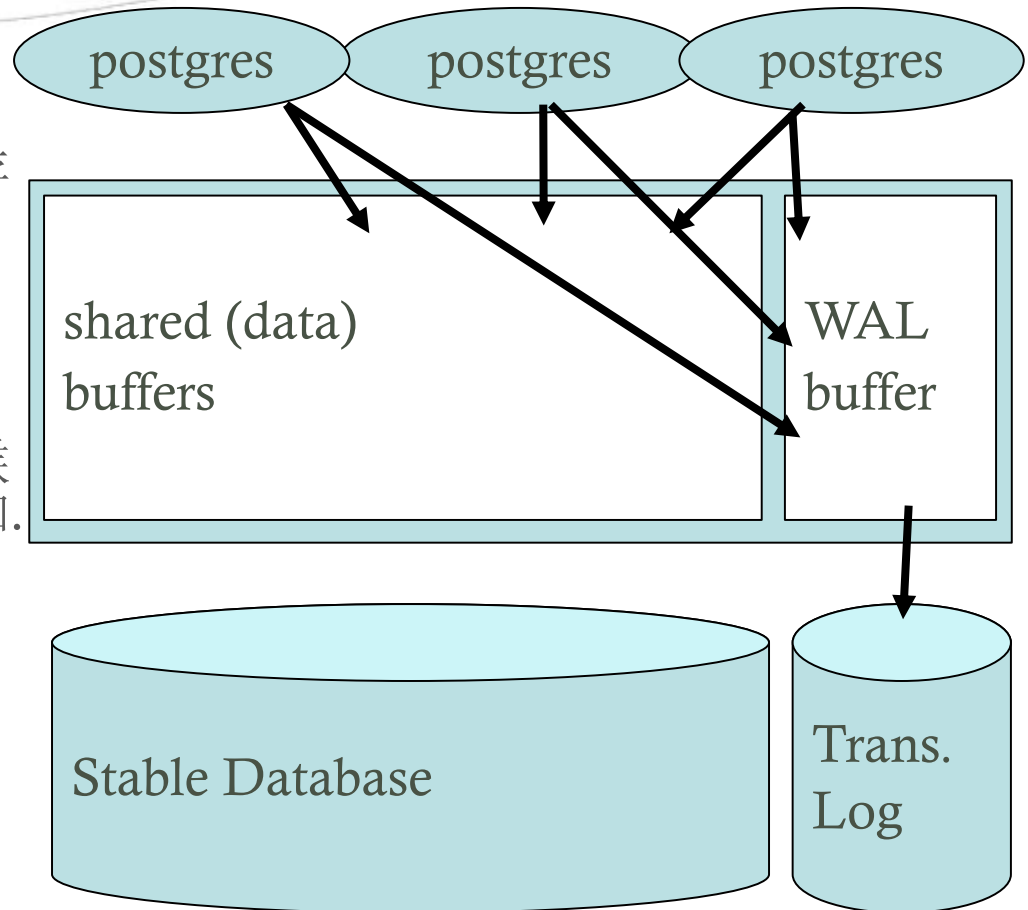
写缓存

- postmaster
 - bgwriter
 - 清除脏块(dirty block)
 - 检查点(checkpoint)



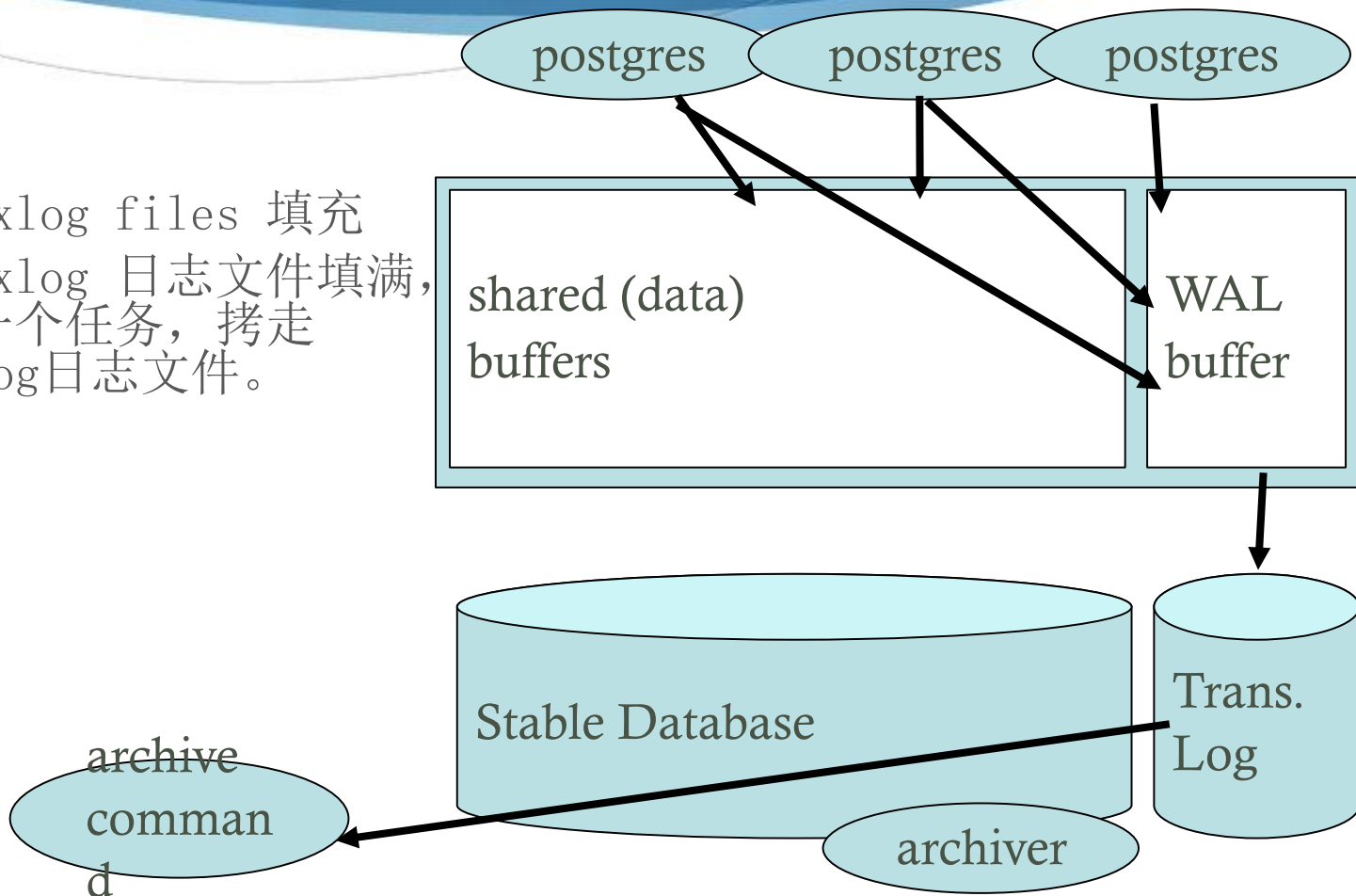
预写日志Write Ahead Logging (WAL)

- postmaster
 - postgres backends
 - 将数据写入数据库缓存 (database buffer) 中
 - 将 txn 日志数据写入日志缓存 (wal buffer) 中
 - 在提交 (commit) 的时候刷新 wal buffer (比如. 写入到磁盘的 pg_xlog 目录中)
 - ... 或者缓冲区 (cache) 满了
 - 组 (group) 提交



事物日志归档(Transaction Log Archiving)

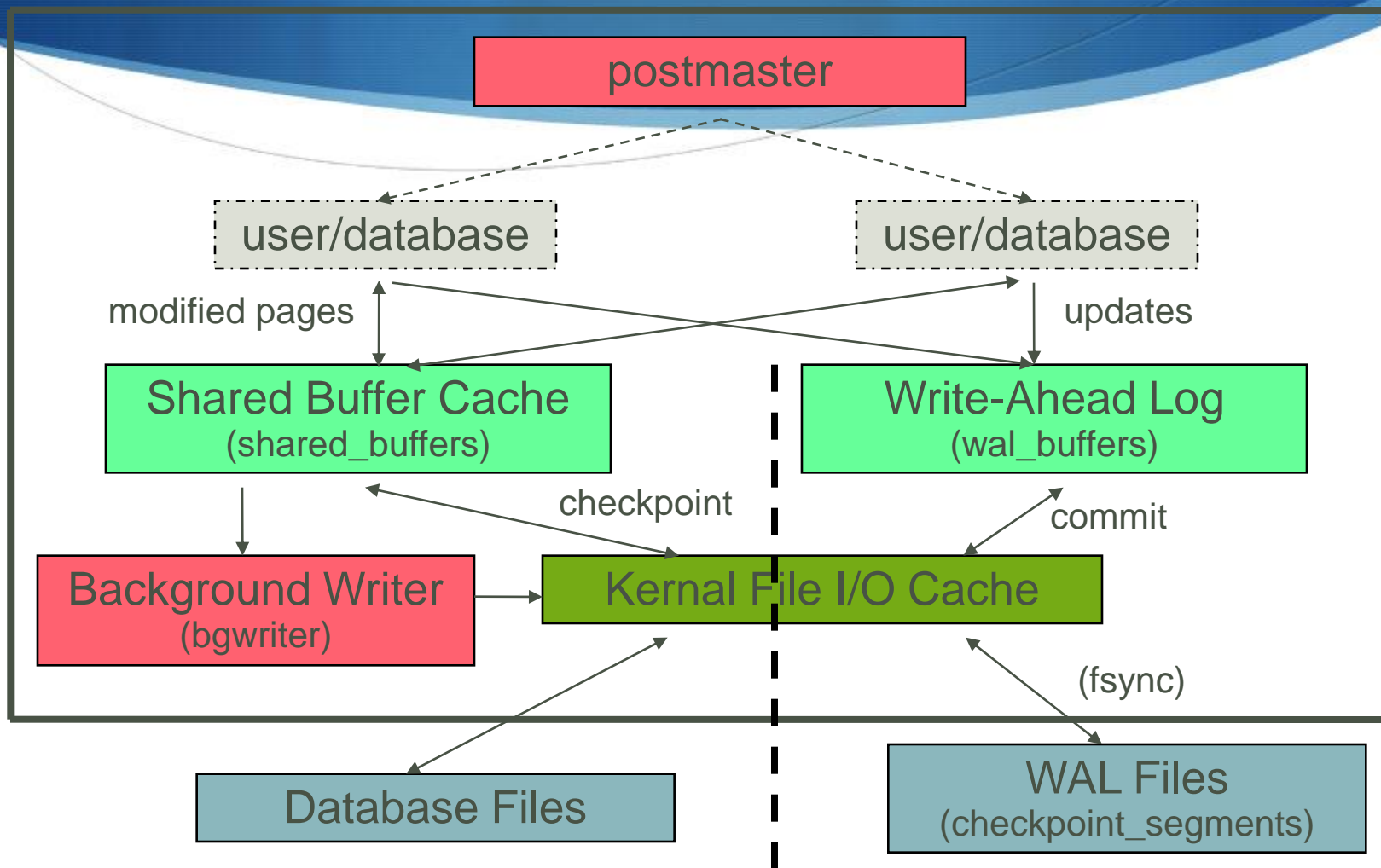
- postmaster
 - archiver
 - 等待 xlog files 填充
 - 当pg_xlog 日志文件填满, 发起一个任务, 拷走pg_xlog日志文件。



提交及检查点

- 提交(Commit)前
 - 没有提交的更新在内存中
- 提交(Commit)后
 - 将提交的更新从共享内存(shared memory)写入磁盘中 (write-ahead log 文件)
- 检查点(Checkpoint)之后
 - 将修改的数据页从共享内存(shared memory)写入到数据文件中

共享缓存及预写日志



模式查找路径(1)

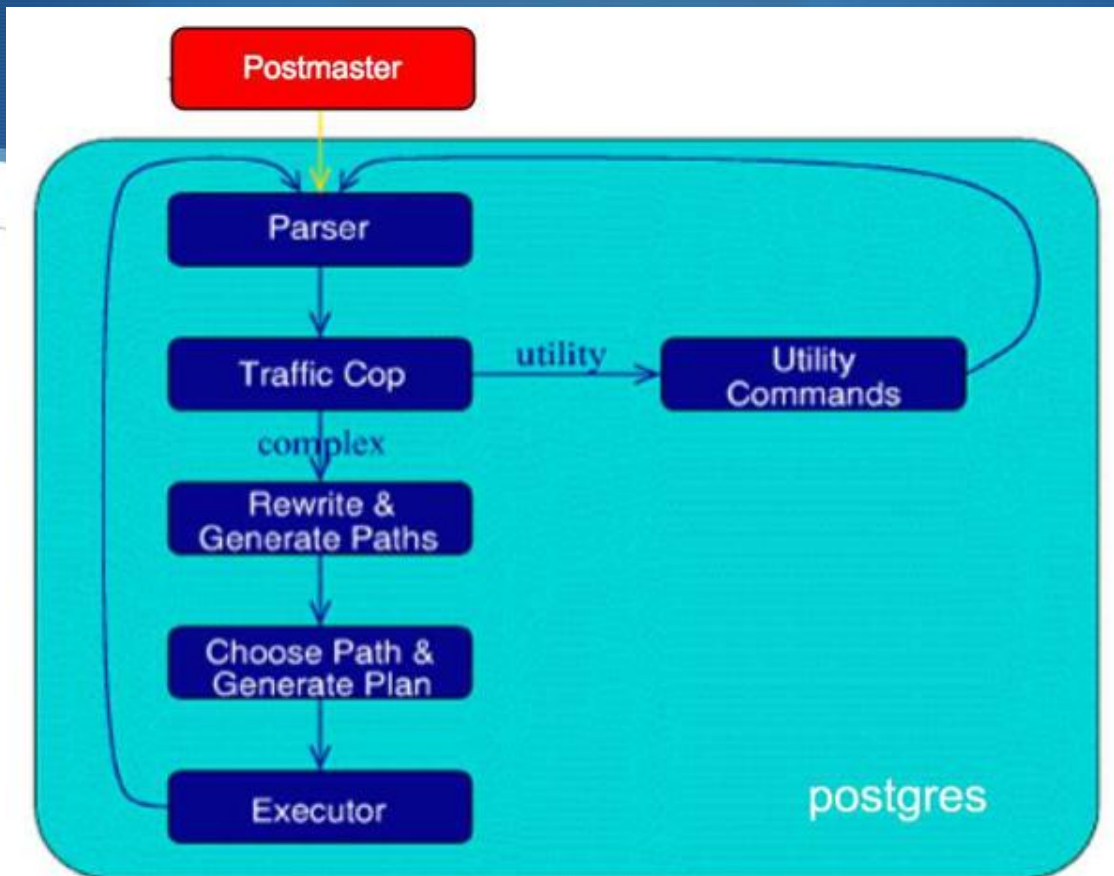
- ◆ 一个完整的名字写起来太冗长，我们一般在查询时直接写表名。
- ◆ 在查找表的时候如果前面没有明确给出模式名，那么将利用模式查找路径(Schema Search Path)决定要访问的模式名。
- ◆ 例如：
 - ◆ `SELECT * FROM employee`
 - ◆ 这条语句将首先查看在查找路径(search path)中列出的第一个模式的employee表

模式查找路径(2)

- ◆ 在查找路径中首先列出的模式叫做当前模式(current schema)。它除了是被首先搜索的模式，还是在CREATE TABLE命令中新表创建时首先被放入的模式（如果没有指明模式名）。
- ◆ 要查看查找路径可通过如下命令：
 - ◆ SHOW search_path;
 - ◆ 在默认的设置中它将返回：
search_path

"\$user", public
 - ◆ 第一个值声明了与当前用户同名的模式将首先被搜索。如果没有这个模式，这个值将被忽略。第二个值表明之后将搜索public模式
 - ◆ 将我们自定义的新模式放到路径中，可以：
 - ◆ SET search_path TO myschema, public;

SQL执行过程



更多信息可以参考<http://momjian.us/presentations>中的
PostgreSQL Internals Through Pictures

事务隔离级别

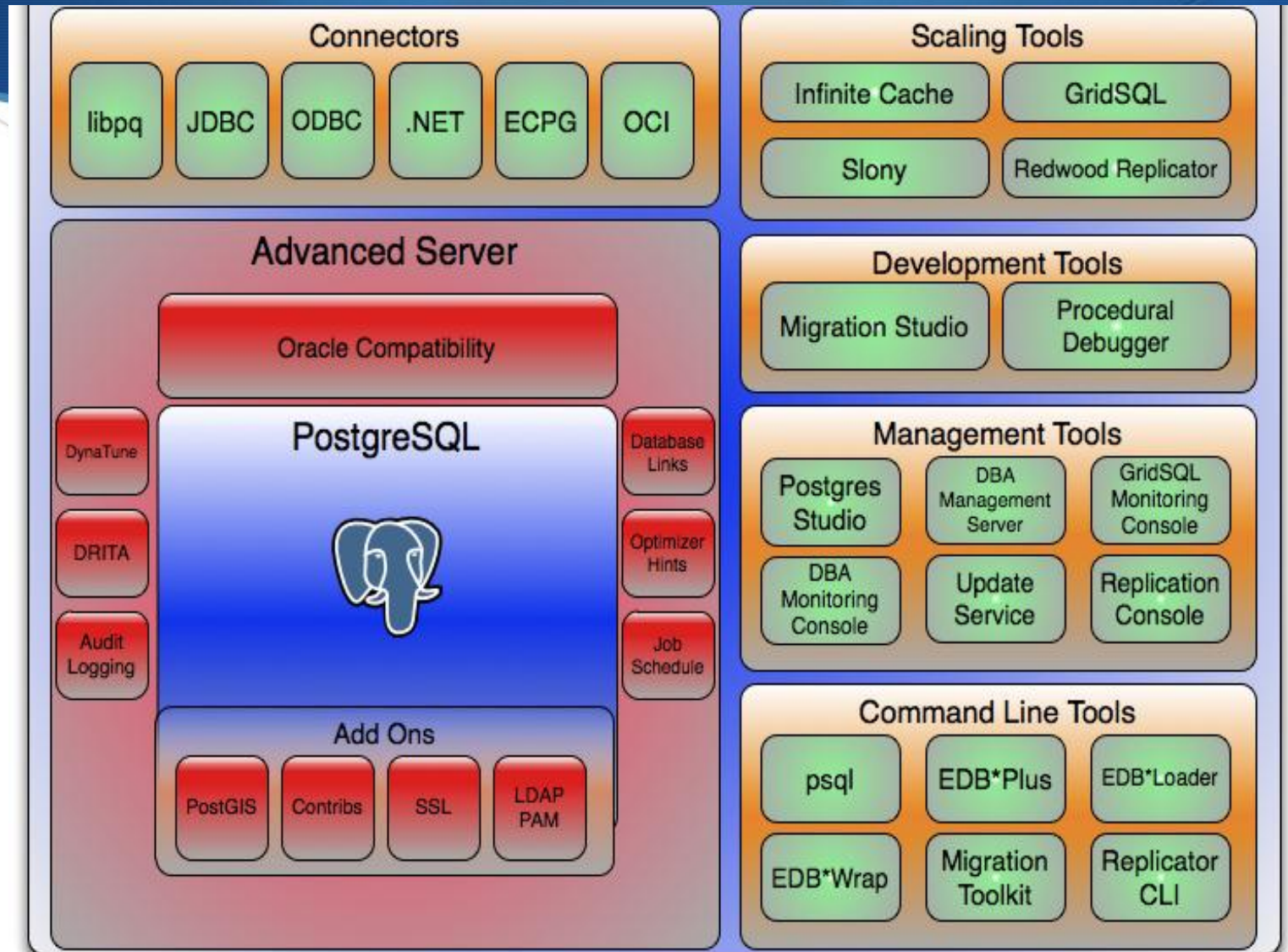
Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not Possible	Not Possible	Not Possible

MVCC多版本控制

Transaction State	DeptNo	DName	Loc
Committed	10	Accounting	New York
Deleted	20	Research	Dallas
Special Case	30	Sales	Chicago
Committed	40	Operations	Boston
Committed	20	Research	San Diego
Committed	50	Telesales	Salt Lake City
Deleted	999	Test	Tokyo
Tx1 (Insert)	60	Pacific Sales	Seattle
Tx2 (Update)	30	HQ Sales	Chicago

Write-locks don't block reads and read-locks don't block writes.

Postgres Plus Advanced Server (了解)



权限及安全管理

认证 (Authentication) 和授权 (Authorization)

- 安全包括两个过程：
 - 认证 (Authentication) - 保证一个用户是他/她声明的用户
 - 授权 (Authorization) - 保证一个已认证的用户只能访问他/她被授权的数据。
- 让我们分别解释这两个过程

安全级别 (Levels of Security)

Server & Application Security
pg_hba.conf



Database Object Security
Schemas & Users



Table Security
Grant & Revoke

建立数据库用户

- ◆ 数据库用户和操作系统用户不同
- ◆ 用户可以通过SQL命令 “CREATE USER” 或 “createuser” 工具创建

例如:

```
CREATE USER scott PASSWORD 'tiger';  
CREATE USER scott_dba CREATEDB CREATEUSER;  
CREATE USER scott_edb VALID UNTIL '2013-03-01';  
DROP USER scott CASCADE;
```

操作权限赋予

对表的访问权限通过 GRANT 和 REVOKE SQL 命令赋予和回收。

例如：

```
GRANT UPDATE DELETE ON emp TO scott_temp;
```

```
GRANT UPDATE chgdesc ON jobhist TO scott;
```

```
GRANT ALL ON dept TO GROUP scott_edb;
```

```
REVOKE UPDATE DELETE ON emp FROM scott_edb;
```

```
GRANT USAGE ON SCHEMA postgres TO scott;
```

应用程序访问

- 控制应用程序的访问可以通过设置 `postgresql.conf` 和 `pg_hba.conf` 两个文件中的部分参数实现
- 在 `postgresql.conf` 中设置如下参数：
 - `listen_address`
 - `max_connections`
 - `superuser_reserved_connections`
 - `port`
 - `unix_socket_directory`
 - `unix_socket_group`
 - `unix_socket_permissions`

pg_hba.conf 访问控制

- 本地访问控制(Host based access)
 - 控制 (通过 ip / subnet) 认证机制(authentication mechanism)
 - 通过 ip / subnet添加新的的认证控制
 - 修改后需要对postgresql进行reload

```
# TYPE      DATABASE      USER      CIDR-ADDRESS      METHOD

# "local" is for Unix domain socket connections only
local      all             all              md5

# IPv4 local connections:
host       customer     readonly      62.124.15.9       md5
host       all           all           127.0.0.1/32      md5
host       all           all           0.0.0.0/0         reject

# IPv6 local connections:
host       all           all           ::1/128           reject
```

自动登陆

- ◆ 可以通过配置pgpass文件实现数据库免密码自动登陆
- ◆ Linux: `<HOME>/.pgpass`, 要求文件权限0600
- ◆ Windows: `%APPDATA%\postgresql\pgpass.conf`
- ◆ 格式
 - ◆ `hostname:port:database:username:password`
- ◆ 实例
 - ◆ `localhost:5432:postgres:scott:postgres`

目 录

1. 在edb这个数据库下建立一个用户user1；并在public这个schema下建立一个名为newuser_t1的table；只有1个名为id，类型为int的column
2. 建立一个与user1同名的schema，并在此schema中建立名为newuser_t1的table；只有1个名为username，类型为varchar的column
3. 登陆user1，测试user1执行select * from newuser_t1的结果
4. 建立并登陆user2，测试user2执行select * from newuser_t1的结果

数据备份及恢复

备份

- 💧 Postgres 和其他数据库一样，需要定时的备份。
- 💧 有三种基本的类型备份Postgres 数据：
 - 💧 SQL转储 (SQL dump)
 - 💧 文件系统级别的备份
(File system level backup)
 - 💧 在线备份 (On-line backup)

备份 – SQL Dump

- 💧 创建一个带有SQL命令的text文档
- 💧 PostgreSQL 提供工具 `pg_dump` 来完成.
- 💧 `pg_dump` 不阻碍块的读写.
- 💧 `pg_dump` 操作不需要有特殊的权限。但需要你至少能够读所有你要备份的表，因此，在练习中你几乎总是要作为数据库超级管理员来运行它。
- 💧 通过`pg_dump`创建出来的转储文件(Dumps)是内部一致的，也就是说，dump代表了一个数据库在`pg_dump`命令开始执行时的快照(snapshot)。

语法：

```
pg_dump [options] [dbname]
```

pg_dump命令

pg_dump 选项

- ◆ -a - Data only. Do not dump the data definitions (schema)
- ◆ -b - Include large objects in dump
- ◆ -s - Data definitions (schema) only. Do not dump the data
- ◆ -n <schema> - Dump from the specified schema only
- ◆ -t <table> - Dump specified table only
- ◆ -f <path/file name.backup> - Send dump to specified file
- ◆ -Fp - Dump in plain-text SQL script (default)
- ◆ -Ft - Dump in tar format
- ◆ -Fc - Dump in compressed, custom format
- ◆ --inserts - dump data as INSERT commands, rather than COPY
- ◆ -o use oids

pg_dump备份实例(1)

- ◆ `pg_dump -U postgres -s -Fp -f /tmp/postgres_schema.sql postgres`
通过postgres用户登陆到postgres库，并将所有数据结构备份到
`postgres_schema.sql`
- ◆ `pg_dump -U postgres -a -b --inserts -Fp -f /tmp/postgres_data.sql postgres`
通过postgres用户登陆到postgres库，并将所有数据通过insert命令备份到
`postgres_schema.sql`
- ◆ `pg_dump -U postgres -b -Fp -f /tmp/postgres_all.sql postgres`
通过postgres用户登陆到postgres库，并将所有表结构及数据通过备份到
`postgres_schema.sql`，当中数据导出成COPY语句

恢复 – SQL Dump

- 通过pg_dump命令创建的text文件一般是为 psql 程序作为读入的。通常用命令行形式去恢复一个dump如下
 - `psql dbname < infile`

其中infile是pg_dump 命令的输出文件。那个叫 dbname 的数据库不会通过这个命令自动创建，你需要手动创建。

- pg_restore 用来恢复由pg_dump命令备份出的归档形式的文件 - 比如, 一个非文本形式
- 归档的文件在不同的硬件体系之间是可移植的。
语法：
`pg_restore [options...] [filename.backup]`

恢复 - SQL Dump

pg_restore 选项

- ◆ `-d <database name>` - Connect to the specified database. Also restores to this database if `-C` option is omitted
- ◆ `-C` - Create the database named in the dump file & restore directly into it
- ◆ `-a` - Restore the data only, not the data definitions (schema)
- ◆ `-s` - Restore the data definitions (schema) only, not the data
- ◆ `-n <schema>` - Restore only objects from specified schema
- ◆ `-t <table>` - Restore only specified table
- ◆ `-v` - Verbose option

pg_dump恢复实例(1)

- ◆ `psql -U postgres -f /tmp/postgres_schema.sql postgres`
通过postgres用户登陆到postgres库，并将/tmp/postgres_schema.sql恢复到postgres库(也可通过pgAdmin3打开SQL直接执行)
- ◆ `psql -U postgres -f /tmp/postgres_data.sql postgres`
通过postgres用户登陆到postgres库，并将/tmp/postgres_data.sql恢复到postgres库(也可通过pgAdmin3打开SQL直接执行)
- ◆ `psql -U postgres -f /tmp/postgres_all.sql postgres`
通过postgres用户登陆到postgres库，并将/tmp/postgres_all.sql恢复到postgres库(不可通过pgAdmin3打开SQL直接执行，COPY必须通过stdin进行)

pg_dump备份实例(2)

pg_restore恢复

- ◆ 备份: `pg_dump -U postgres -t emp -t dept -Fc -f /tmp/postgres_all.backup postgres`

通过postgres用户登陆到postgres库，并将所有表结构及数据通过数据压缩后备份到postgres_schema.backup

- ◆ 恢复: `pg_restore -U postgres -f /tmp/postgres_all.backup postgres`

通过postgres用户登陆到postgres库，并将所有表结构及数据通过数据从postgres_schema.backup恢复到数据库中(由于此文件备份时使用-Fc参数，因此不能通过psql或pgAdmin3的SQL编辑器打开倒入，但可通过pgAdmin3的Restore菜单进行数据恢复操作)

备份 – SQL Dump

pg_dumpall 选项

- 💧 -a - Data only. Do not dump the data definitions (schema)
- 💧 -s - Data definitions (schema) only. Do not dump the data
- 💧 -g - Dump global objects only - i.e., users and groups
- 💧 -v - Verbose option

恢复 - SQL Dump

语法:

```
psql -d template1 < filename.backup
```

or

```
psql -d template1 -f filename.backup
```

数据集合实例中的任何数据库都可以作为初始连接(initial connection) - 不必要是 template1

备份 – File system level backup

- 💧 另一种备份方式是直接拷贝PostgreSQL 中用来存储数据的文件。
- 💧 你可以用任何方式来进行通常的系统文件备份，比如：
 - 💧 `tar -cf backup.tar /usr/local/pgsql/data`
- 💧 为了得到一个可用的备份，数据库服务器必须关闭。
- 💧 文件系统级别的备份只为完全备份，并且恢复整个数据集实例。

pg_dumpall备份实例

- pg_dumpall -U postgres -f /tmp/pgdump_all.sql
Password:
Password:
Password:
Password:
Password:

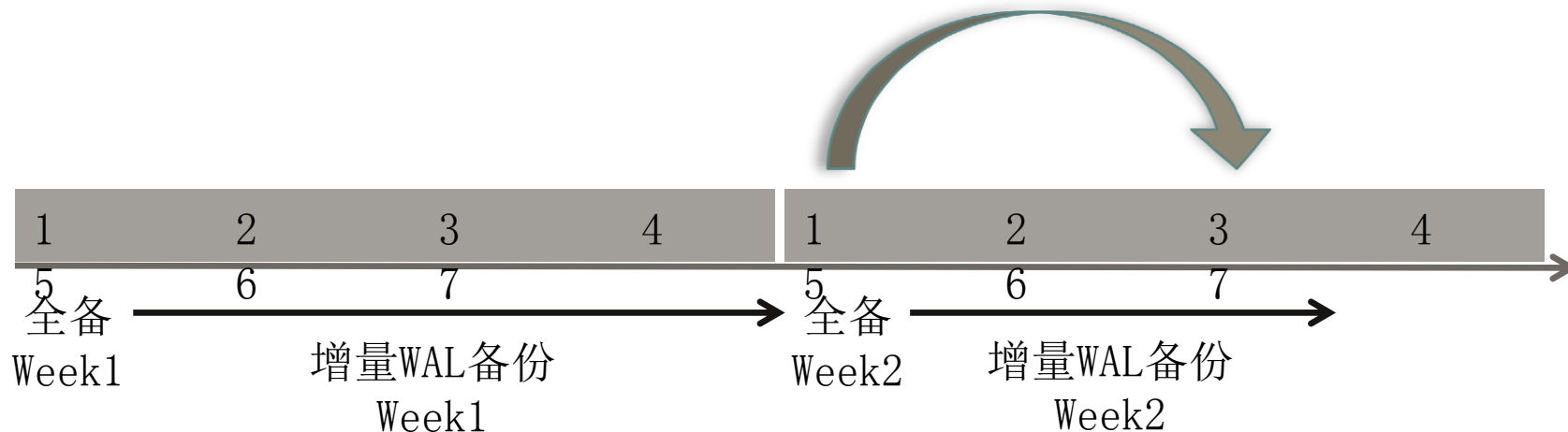
- 此处会出现多个Password，这是由于pgdump_all会对整个数据集合实例备份，每备份一个数据库时都要进行重新的连接认证。
可通过设置pgpass文件避免进行多次密码填写
- 恢复时使用psql的-f参数进行数据导入

基于时间点的数据恢复

实现数据增量备份

Point-In-Time-Recovery (PITR)
基于时间点的数据恢复

通过Week2的归档WAL
进行数据恢复



配置PITR

- 建立目录/opt/PostgreSQL/9.1/archive及/opt/PostgreSQL/9.1/full_backup并使它们属于postgres用户及postgres组
- 修改postgresql.conf的以下参数

```
wal_level = archive
archive_mode = on
archive_command = 'cp %p /opt/PostgreSQL/9.1/archive/%f'
```
- Restart服务器

进行一次全备份

- ◆ `SELECT pg_start_backup('FullBackup');`
- ◆ `tar zcvf`
`/opt/PostgreSQL/9.1/full_backup/week1.tar.gz`
`/opt/PostgreSQL/9.1/data`
- ◆ `SELECT pg_stop_backup();`

进行数据恢复

- 删除/opt/PostgreSQL/9.1/data模拟数据丢失(或将此目录改名)
- `tar zxvf /opt/PostgreSQL/9.1/full_backup/week1.tar.gz -C /`
- 建立/opt/PostgreSQL/9.1/data/recovery.conf加入以下内容
`restore_command = 'cp /opt/PostgreSQL/9.1/archive/%f %p'`
`recovery_target_time = '2012-03-18 09:40:00'`
注意: 此时间必须在`SELECT pg_start_backup('FullBackup');`时间之后

目 录

1. 搭建自己的PITR，并尝试进行指定时间点的恢复

运行时维护管理

VACUUM (1)

- ◆ 对一行的 UPDATE 或 DELETE 操作并不会立即删除此行的旧的版本(old version)。这种做法对于得到多版本控制(Multi-Version Concurrency Control)的好处是必要的。
- ◆ 但是最后，一个过时的或已删除的行的旧版本对于系统是没用的，这些行所占的空间最终要被回收以供新行使用，来满足对磁盘空间的不断增长的需求。
- ◆ 通过 VACUUM来满足此需求。

VACUUM (2)

- ◆ 由于以下几个原因，必须周期性运行PostgreSQL 的 VACUUM 命令：
 - ◆ 1. 恢复或重用那些已更新或已删除的行所占的空间。
 - ◆ 2. 更新PostgreSQL查询规划器使用的统计数据。
 - ◆ 3. 避免因为事务 ID 重叠 (transaction ID wraparound)造成的老旧数据的丢失。
- ◆ VACUUM 的标准形式可以和生产数据库的操作并行运行。像 SELECT, INSERT, UPDATE, 和 DELETE 将照常运行。

VACUUM (3)

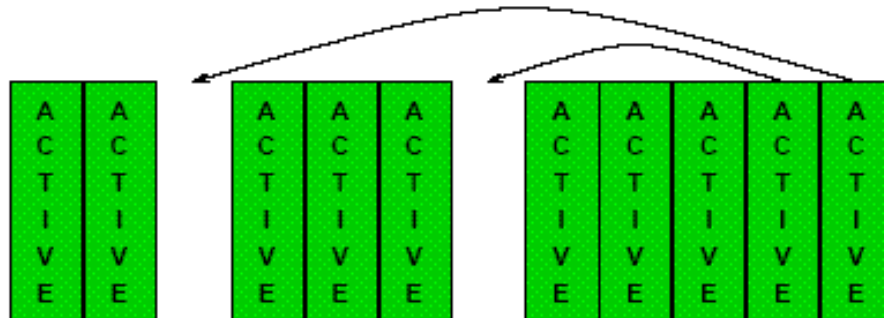
- 💧 VACUUM 命令有两种形式.
- 💧 第一种形式是VACUUM命令:
 - 💧 对表和索引中没用数据 (dead data) 做一个标记, 以便将来使用。
 - 💧 VACUUM它不会试图去回收没用的数据 (dead data) 所占的空间, 除非这些没用的空间在表的末尾并且非常容易获得表上的排它锁。
- 💧 第二种形式是VACUUM FULL命令:
 - 💧 VACUUM FULL 采用了一种更积极的算法来回收没用的数据。
 - 💧 被VACUUM FULL释放的空间将立即返回给操作系统。
 - 💧 在VACUUM FULL执行的时候, 需要对所操作的表加一个排它锁。
- 💧 语法: `VACUUM [FULL | FREEZE] [VERBOSE] ANALYZE [table [(column [, ...])]]`

VACUUM FULL

Original Heap
With Expired
Rows Identified

A	A	E	A	A	A	E	A	A	A	A	A
C	C	X	C	C	C	X	C	C	C	C	C
T	T	P	T	T	T	P	T	T	T	T	T
I	I	I	I	I	I	I	I	I	I	I	I
V	V	R	V	V	V	R	V	V	V	V	V
E	E	E	E	E	E	E	E	E	E	E	E

Move Trailing
Rows Into Expired
Slots



Truncate File

A	A	A	A	A	A	A	A	A	A		
C	C	C	C	C	C	C	C	C	C		
T	T	T	T	T	T	T	T	T	T		
I	I	I	I	I	I	I	I	I	I		
V	V	V	V	V	V	V	V	V	V		
E	E	E	E	E	E	E	E	E	E		

来源: http://amevor.de/wp-content/uploads/2008/05/vacuum_full.GIF

REINDEX

- 💧 在某些情况下，我们值得用REINDEX 命令周期性地重建索引。
- 💧 在PostgreSQL 7.4 及之后的版本中，那些完全为空的索引页将会被回收以重复利用。膨胀的可能性不是无穷的 — 最差的情况是每个页面至少还有一个键字 — 但是对这样的使用模式，我们可能仍然值得安排周期性的重新索引(reindexing)。
- 💧 REINDEX { TABLE | DATABASE | INDEX } name [FORCE]

注意：REINDEX会造成操作对象的EXCLUSIVE LOCK，从而导致write操作堵塞，但对于reader操作是不会堵塞的。

ANALYZE

- PostgreSQL 查询规划器依赖于ANALYZE命令所收集的统计信息。 ANALYZE可以自行触发，或作为VACUUM的一个可选步骤。
- 可以通过ANALYZE命令在某个表上，甚至在表上的某个列上运行。所以，如果你的应用需要，你可以灵活地选择更新某些统计数据，而不是全部。
- 推荐你最好规划一个数据库范围内的ANALYZE，然后每天在系统不太繁忙的时候运行一次。
- ```
ANALYZE [VERBOSE] [table [(column [, ...])]]
VACUUM [FULL | FREEZE] [VERBOSE] ANALYZE [table [(column [, ...])]]
```

# 避免事务ID重复

## Transaction ID Wraparound

- ◆ PostgreSQL的 MVCC 事务语义依赖于比较事务 ID(XID)的数值：一条带有大于当前事务的 XID 的插入 XID 的行版本是“属于未来的(in the future)”，并且不应为当前事务可见。但是因为事务 ID 的大小有限（在我们写这些的时候是 32 位），如果一次数据集实例运行的时间很长（大于 40亿次事务），那么将遇到 *事务 ID 重叠* 的问题。
- ◆ 为了避免这种情况，数据库的每个表在达到十亿事物量时都需要 VACUUM 一次。

# 存储过程及调试



# 过程语言概述

- ◆ PostgreSQL 允许用户自定义的函数，它可以用各种过程语言编写。数据库服务器是没有任何内建的知识获知如何解析该函数的源文本的。实际上这些任务都传递给一个知道如何处理这些细节的句柄(handler)处理。
- ◆ PostgreSQL 当前支持多个标准的过程语言：
  - ◆ PL/pgSQL
  - ◆ PL/Tcl
  - ◆ PL/Perl
  - ◆ PL/Python
  - ◆ PL/Java
  - ◆ PL/Ruby
  - ◆ 其他语言可以由用户自定义

# PL/pgSQL结构(1)

- PL/pgSQL是一种块结构(block-structured)的语言。 函数定义的所有文本都必须是一个块。 一个块用下面的方法定义：

```
[<<label>>]
[DECLARE
 declarations]
BEGIN
 statements
END [label];
```

- 块中的每个声明和每条语句都是用一个分号终止的， 如果一个子块在另外一个块里， 那么 END 后面必须有个分号， 如上所述； 不过结束函数体的最后的 END 可以不要这个分号。

# PL/pgSQL结构(2)

- ◆ 所有的关键字和标识符(identifiers)可以大小写混用。如果没有双引号，标识符默认转换为小写。
- ◆ 在 PL/pgSQL中有两种注释
  - ◆ -- 单行注释
  - ◆ /\* 多行注释 \*/
- ◆ 在块前面的位于声明部分的变量会在每次块载入的时候(不是每次函数被调用的时候)被初始化为默认值。

# PL/pgSQL 实例

```
CREATE FUNCTION make_it_double(v_input integer) RETURNS
 integer AS $$
DECLARE
 v_double integer := 2;
BEGIN
 -- This is a DEMO!
 RAISE NOTICE 'Your input is %', v_input;
 v_double := v_double * v_input;
 RAISE NOTICE 'Our return is %', v_double;
 RETURN v_double;
END;
$$ LANGUAGE plpgsql;
```

# 运行PL/pgSQL所建立的函数

```
postgres=# SELECT make_it_double(40);
```

```
NOTICE: Your input is 40
```

```
NOTICE: Our return is 80
```

```
make_it_double
```

```

```

```
80
```

```
(1 row)
```

# Triggers触发器(1)

- ◆ 用 CREATE FUNCTION命令创建
- ◆ 系统会在顶层的声明段里自动创建几个特殊变量。
  - ◆ NEW
    - ◆ 数据类型是 RECORD; 该变量为INSERT/UPDATE 操作在行级(row-level)触发器中保存一个新的数据库行。这个变量在语句级(statement-level)触发器中为NULL。
  - ◆ OLD
    - ◆ 数据类型是 RECORD; 该变量为INSERT/UPDATE 操作在行级(row-level)触发器中保存一个旧的数据库行。这个变量在语句级(statement-level)触发器中为NULL。

# Triggers触发器(2)

## 💧 TG\_WHEN

- 💧 数据类型是 text; 是一个由触发器定义决定的字符串, 要么是 BEFORE 要么是 AFTER。

## 💧 TG\_LEVEL

- 💧 数据类型是 text; 是一个由触发器定义决定的字符串, 要么是 ROW 要么是 STATEMENT。

## 💧 TG\_OP

- 💧 数据类型是 text; 是一个说明触发器的操作的字符串, 可以是 INSERT, UPDATE 或者 DELETE。

## 💧 TG\_RELNAME

- 💧 数据类型是 name; 是激活触发器调用的表的名称。

## 💧 TG\_NARGS

- 💧 数据类型是 integer; 是在CREATE TRIGGER 语句里面赋予触发器过程的参数的个数。



```
CREATE OR REPLACE FUNCTION emp_sal_trig_function() RETURNS trigger AS $$
DECLARE
```

```
 sal_diff numeric(7,2);
```

```
BEGIN
```

```
 IF TG_OP = 'INSERT' THEN
```

```
 RAISE NOTICE 'Inserting employee %', NEW.empno;
```

```
 RAISE NOTICE '..New salary: %', NEW.sal;
```

```
 END IF;
```

```
 IF TG_OP = 'UPDATE' THEN
```

```
 sal_diff := NEW.sal - OLD.sal;
```

```
 RAISE NOTICE 'Updating employee %', OLD.empno;
```

```
 RAISE NOTICE '..Old salary: %', OLD.sal;
```

```
 RAISE NOTICE '..New salary: %', NEW.sal;
```

```
 RAISE NOTICE '..Raise : %', sal_diff;
```

```
 END IF;
```

```
 IF TG_OP = 'DELETE' THEN
```

```
 RAISE NOTICE 'Deleting employee %', OLD.empno;
```

```
 RAISE NOTICE '..Old salary: %', OLD.sal;
```

```
 END IF;
```

```
 RETURN NEW; --如果返回NULL修改将被回滚
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER emp_sal_trig BEFORE INSERT OR UPDATE OR DELETE ON emp FOR
EACH ROW EXECUTE PROCEDURE emp_sal_trig_function();
```

# 配置pgAdmin3启用Debugger

- 编辑postgresql.conf

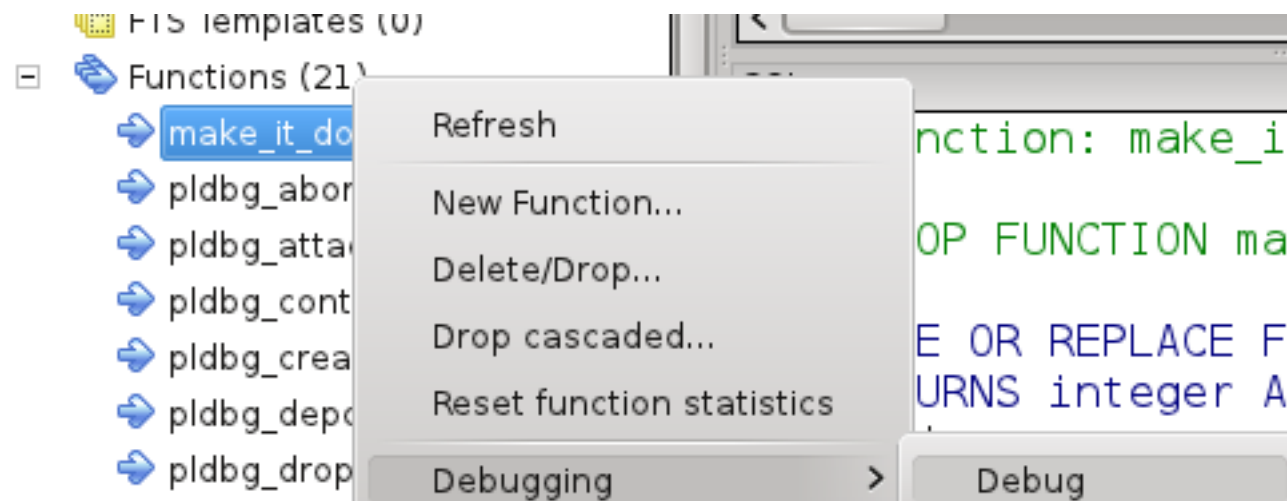
```
shared_preload_libraries =
'$libdir/plugins/plugin_debugger.so'
```

- 重新启动数据库restart

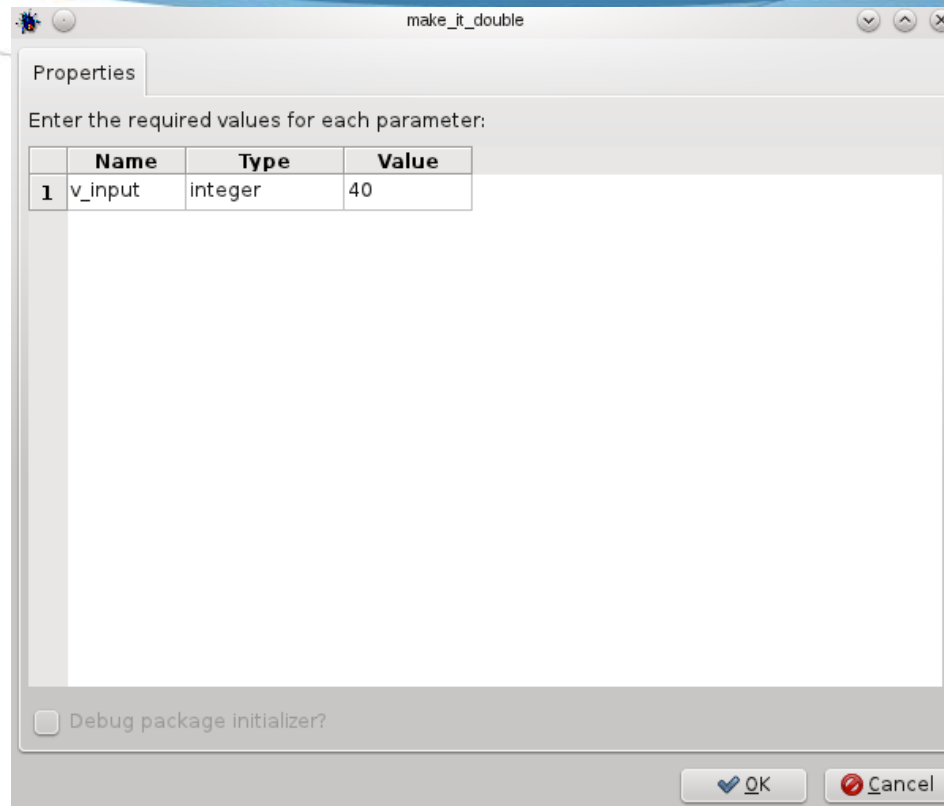
- 执行

```
/opt/PostgreSQL/9.1/share/postgresql/contrib/pldbga
pi.sql
```

# Function Debugging(1)



# Function Debugging(2)



# Function Debugging(3)

Debugger - make\_it\_double

File Debug View Help

DECLARE  
v\_double integer := 2;  
BEGIN  
-- This is a DEMO!  
RAISE NOTICE 'Your input is %',  
v\_double := v\_double \* v\_input;  
▶ RAISE NOTICE 'Quantity here is %',  
RETURN v\_double;  
END;

Stack pane

make\_it\_double(v\_input=40)@7

Output pane

| Name     | Type    | Value |
|----------|---------|-------|
| v_double | integer | 2     |

Parameters Local Variables DBMS Messages

Paused at line 7 Ln 7 Col 1 Ch 140

# 监控排错基础

# 数据库日志

- ◆ 默认路径是 `.../data/pg_log`

- ◆ 用来

通用信息(General log information)

**LOG:** `database system is ready`

**NOTICE:** `CREATE TABLE / PRIMARY KEY will create  
implicit index "dept_pk" for table "dept"`

错误(Errors)

**FATAL:** `user "scott" does not exist`

**ERROR:** `source database "edb" is being accessed by  
other users`

提示(Hints)

**LOG:** `checkpoints are occurring too frequently (11  
seconds apart)`

**HINT:** `Consider increasing the configuration parameter  
"checkpoint_segments".`



# 操作系统进程监控 (Linux)

## Linux

💧 ps - 当前进程的信息

```
ps waux | grep postgres (for Linux and OS/X)
ps -ef | grep postgres (other Unix)
```

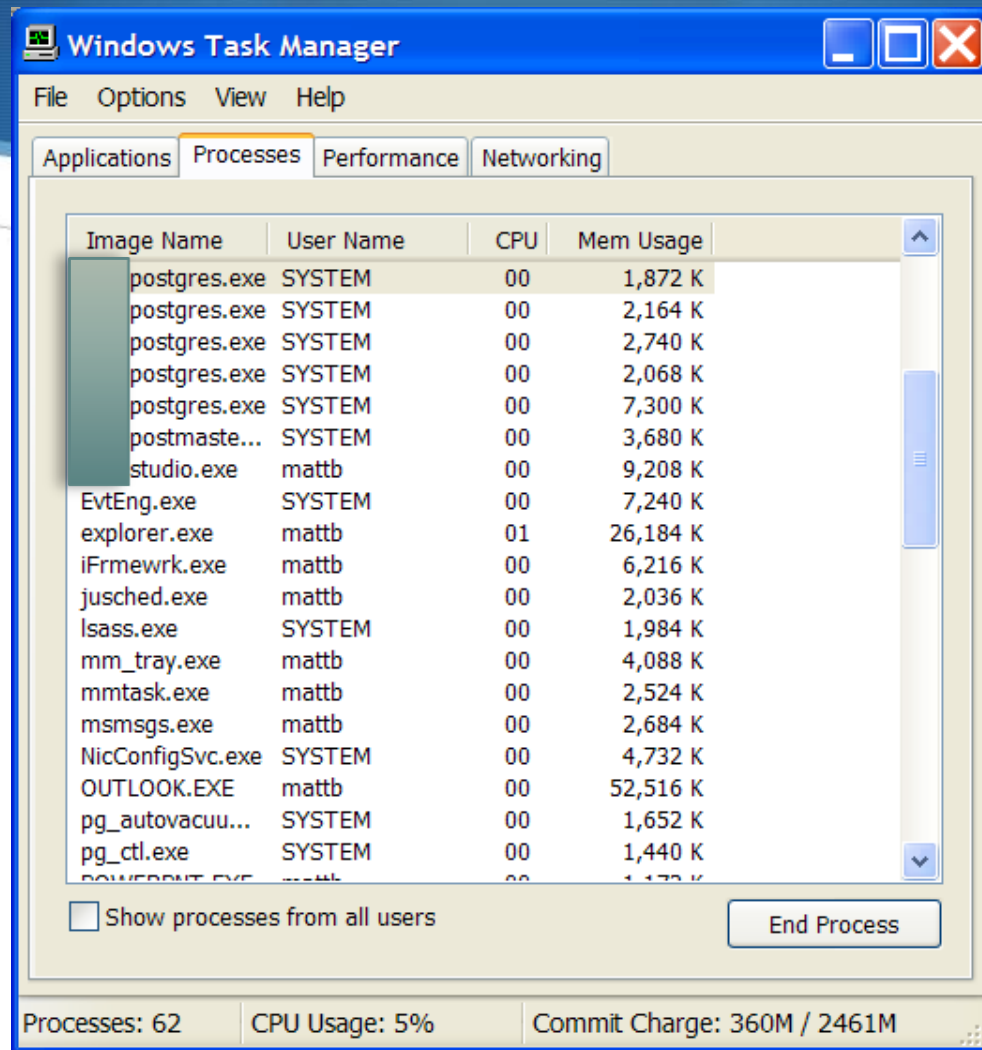
💧 netstat - 当前网络连接的信息

```
netstat -an | grep LISTEN
```

# 操作系统进程监控 (Linux)

```
pgdba2000:/opt/PostgreSQL/9.1 # ps waux | grep postgres
postgres 5818 0.0 0.8 51968 6684 pts/4 S 11:49 0:00
/opt/PostgreSQL/9.1/bin/postgres -D /opt/PostgreSQL/9.1/data
postgres 5819 0.0 0.1 13548 792 ? Ss 11:49 0:00
postgres: logger process
postgres 5822 0.0 0.2 51968 1908 ? Ss 11:49 0:00
postgres: writer process
postgres 5823 0.0 0.1 51968 1148 ? Ss 11:49 0:00
postgres: wal writer process
postgres 5824 0.0 0.2 52512 2092 ? Ss 11:49 0:00
postgres: autovacuum launcher process
postgres 5825 0.0 0.1 13676 1168 ? Ss 11:49 0:00
postgres: stats collector process
postgres 5927 0.0 1.0 53904 7644 ? Ss 11:52 0:00
postgres: postgres postgres ::1(39459) idle
postgres 5980 0.0 0.7 53232 5420 ? Ss 11:55 0:00
postgres: postgres postgres 127.0.0.1(59832) SELECT
postgres 6053 0.0 0.6 52984 4876 ? Ss 11:59 0:00
postgres: postgres postgres 127.0.0.1(59833) idle
root 6202 0.0 0.1 3900 764 pts/4 S+ 12:07 0:00 grep
postgres
```

# Windows进程监控



# 通过pg\_stat\_activity获得信息

显示当前执行的查询的 process id, database, user, query string, 和 start time 信息。

- 💧 修改 postgresql.conf

```
stats_command_string = true
```

- 💧 运行如下的 SQL 命令:

```
SELECT * FROM pg_stat_activity;
```

# 通过pgAdmin3进行锁监控

Tools -> Server

Status - PostgreSQL 9.1 (localhost:5432)

Current log Rotate 10 seconds

Prepared Transactions

| XID | Global ID | Time |
|-----|-----------|------|
|-----|-----------|------|

Logfile

| Timestamp               | Level |
|-------------------------|-------|
| 2012-03-16 11:49:47 EDT | LOG   |
| 2012-03-16 11:49:47 EDT | LOG   |

Activity

| TX start                               | Blocked by | Query                                                                                      |
|----------------------------------------|------------|--------------------------------------------------------------------------------------------|
| 73969+08 2012-02-19 21:50:00.273969+08 | 1123       | begin for value in 50000001..60000000 loop insert into test.test(aa, bb, cc) values(value, |
| 96071+08 2012-02-19 21:50:03.496071+08 | 1123       | delete from test.test where aa between 1 and 10000000;                                     |
| 99131+08 2012-02-19 21:50:05.899131+08 | 1123       | UPDATE test.test SET bb=5000 WHERE aa between 40000001 and 50000000;                       |
| 79426+08 2012-02-19 21:49:54.679426+08 |            | VACUUM FULL VERBOSE test.test                                                              |
|                                        |            | <IDLE>                                                                                     |
|                                        |            | <IDLE>                                                                                     |
|                                        |            | <IDLE>                                                                                     |
|                                        |            | <IDLE>                                                                                     |
|                                        |            | <IDLE>                                                                                     |
| 28924+08 2012-02-19 21:49:57.428924+08 | 1123       | select * from test.test;                                                                   |

Locks

| PID | Database | Relation | User |
|-----|----------|----------|------|
|-----|----------|----------|------|

Activity

| PID  | Application name      | Database | User     | Client          | Cli |
|------|-----------------------|----------|----------|-----------------|-----|
| 5980 |                       | postgres | postgres | 127.0.0.1:59832 | 20  |
| 6302 | psql.bin              | postgres | postgres | local pipe      | 20  |
| 6410 | pgAdmin III - Browser | postgres | postgres | :::1:49981      | 20  |

Done.

# log\_min\_duration\_statement

- ◆ 此参数在postgresql.conf中
  - ◆ 设置最少执行时间(以milliseconds为单位) , 超过此时间语句执行将被日志记录。
  - ◆ 将它设置为0将打印所有的查询以及他们的执行时间。
  - ◆ -1 (默认) 关闭此特性。
  - ◆ 开启这个选项对于跟踪你系统中没有优化的查询将是非常有利的。

# 高级议题

(选学)

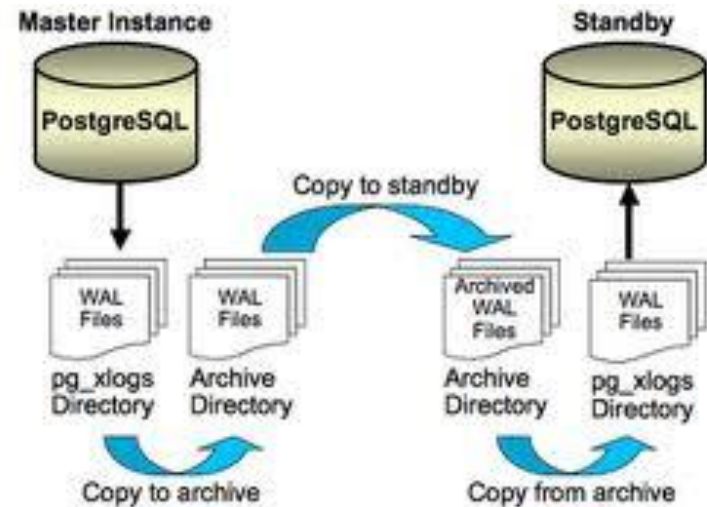


# Hot-Standby & Stream Replication简介及容灾备份 份

Appendixes A

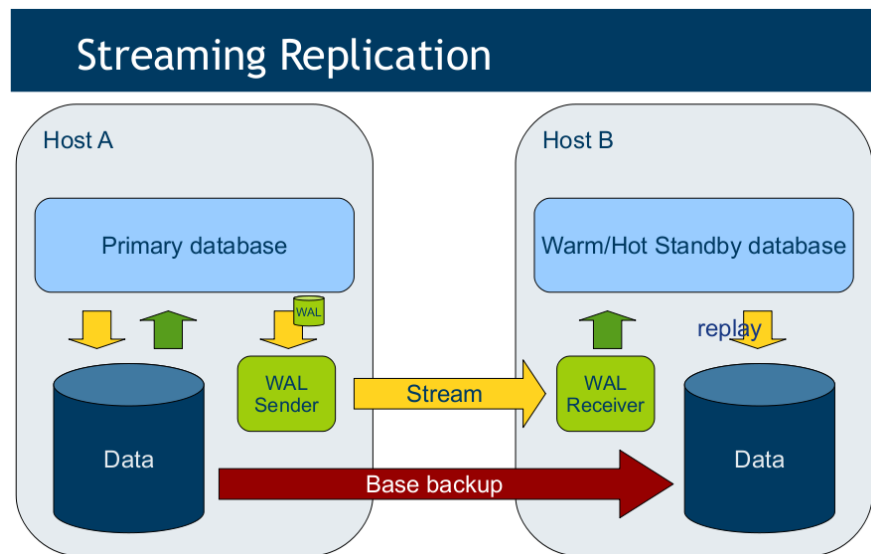
# PostgreSQL 8.X Replication

- 以每个WAL为单位进行复制
- 如果Master宕机，可能丢失数据
- 如果Master的文件系统可打开，有机会恢复所有数据
- Standby不可进行任何操作

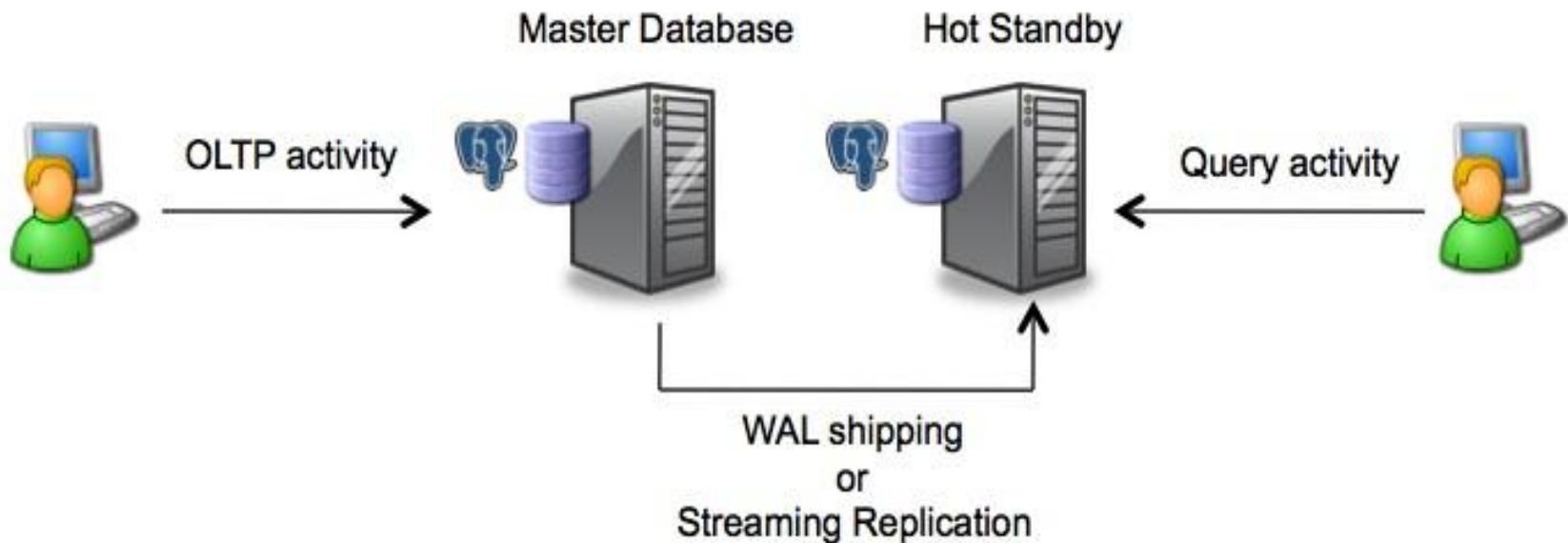


# PostgreSQL 9.X Replication

- 对WAL中的record进行复制
- 支持异步及同步模式
- Standby可进行只读操作
- 异步操作相对性能更好
- 适用于读写分离方案

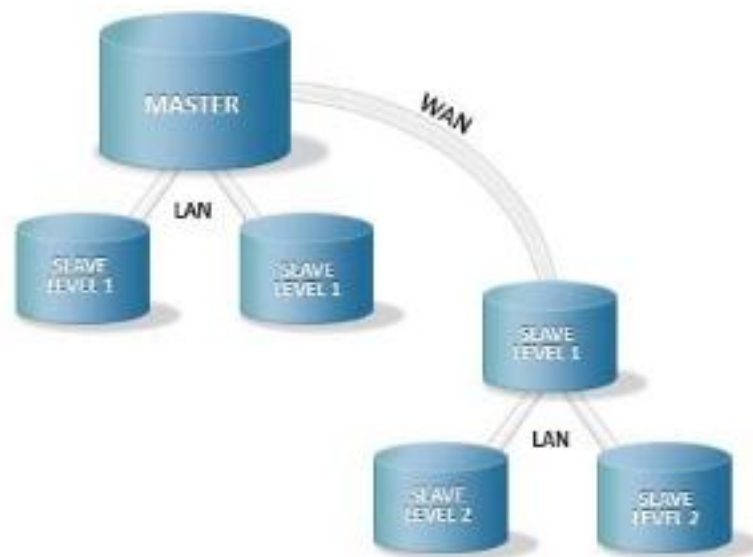


# PostgreSQL 9.X Hot Standby

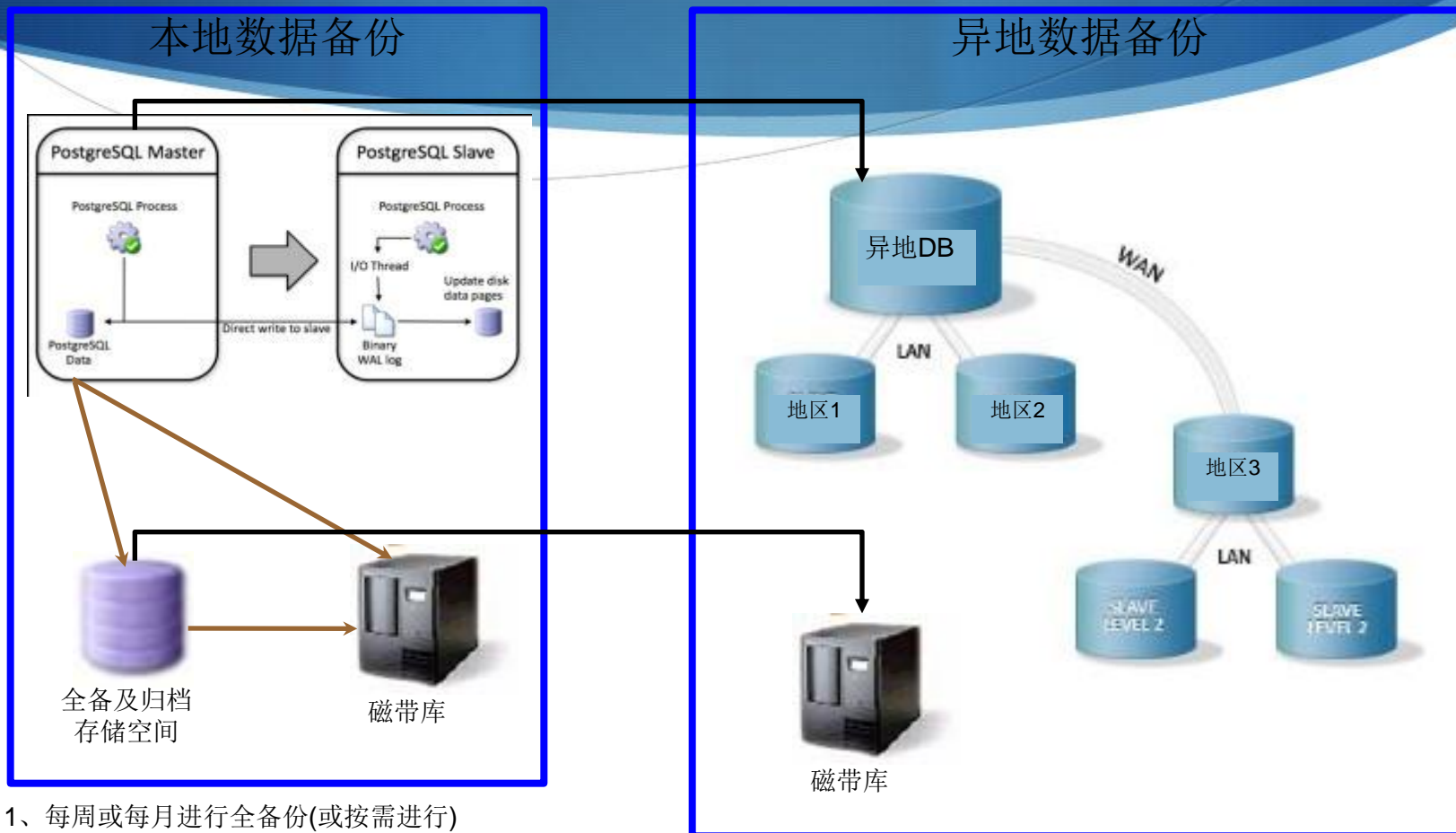


# 其它复制方案：Slony-II

- 💧 适用说异地复制
- 💧 可定义复制间隔
- 💧 支持数据过滤，而非全实例复制
- 💧 Slave服务器支持只读访问
- 💧 支持Windows、Linux及不同版本的复制



# 综合容灾解决方案



- 1、每周或每月进行全备份(或按需进行)
- 2、归档每次全备后的归档作为增量数据
- 3、可结合支持Disk to Disk或Disk to Tape功能的第三方备份系统实现自动管理



# 读写分离集群架构简介

Appendixes B

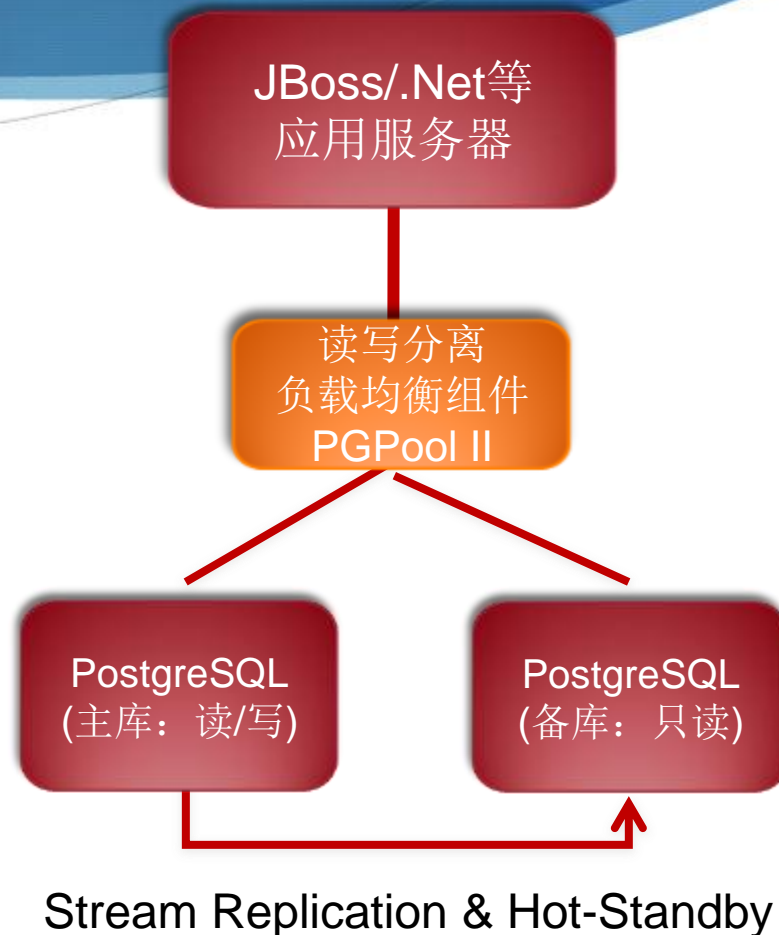


# 如何进行读写分离

- 💧 硬代码
- 💧 数据库中间件：PGPool-II
- 💧 将数据缓存进内存

# PGPool-II

- 自动分离数据库读、写操作
- 智能负载均衡实现并行访问
- 多分数据冗余提高数据安全性
- 高性能异步数据复制
- 独立只读服务器提高查询性能
- 无须独立存储支持节省成本



# 将Data主动缓冲到内存(1)

## 💧 PGFincore

- 💧 数据库启动的时候，数据是冷却的，在执行SQL时需要到磁盘搜索BLOCK并载入到BUFFER，这个时候的SQL响应速度比命中情况下的响应速度一般要慢10倍以上，这种一般被称为未命中的查询。
- 💧 如果数据库启动的时候刚好遇到SQL执行高峰，可能应为SQL响应速度过慢导致应用被堵塞死。造成长时间的应用堵塞，恶性循环。
- 💧 pgfincore, 这个项目是利用posix\_fadvise来对PostgreSQL数据库对象文件进行缓存，放到OS级别的缓存中（注意不是数据库buffer）。所以数据库去文件系统取BLOCK的时候实际上是在操作系统的缓存就拿到的。比直接从磁盘取要快很多。

# 将Data主动缓冲到内存(2)

优化前:

statement latencies in milliseconds:

```
0.004577 \setrandom userid 1 20000000
12.963789 select userid, engname, cnname, occupation, birthday, signname, email, qq from
user_info where userid=:userid;
5.540750 insert into user_login_rec (userid, login_time, ip) values
(:userid, now(), inet_client_addr());
4.457834 update user_session set logintime=now(), login_count=login_count+1 where
userid=:userid;
```

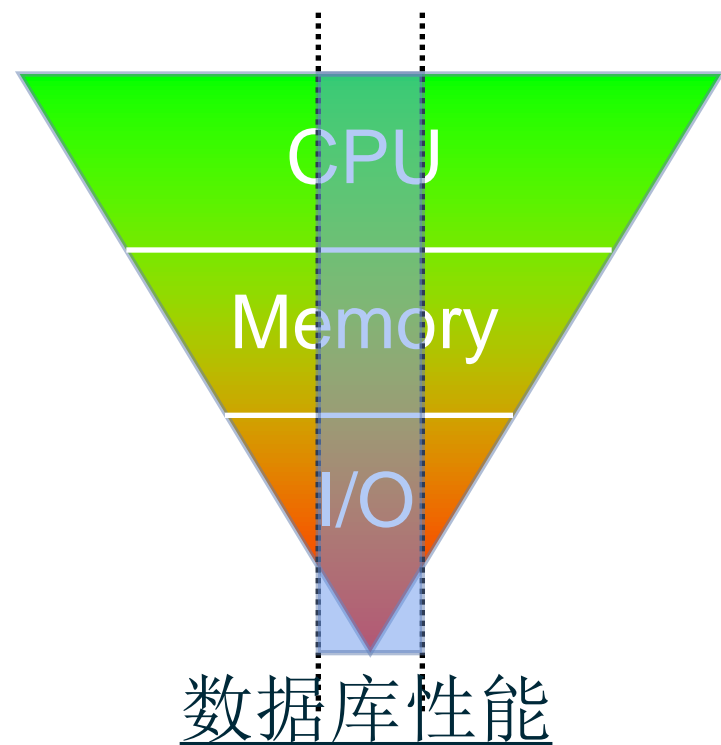
优化后:

statement latencies in milliseconds:

```
0.004226 \setrandom userid 1 20000000
0.459824 select userid, engname, cnname, occupation, birthday, signname, email, qq from
user_info where userid=:userid;
2.457797 insert into user_login_rec (userid, login_time, ip) values
(:userid, now(), inet_client_addr());
2.501684 update user_session set logintime=now(), login_count=login_count+1 where
userid=:userid;
```

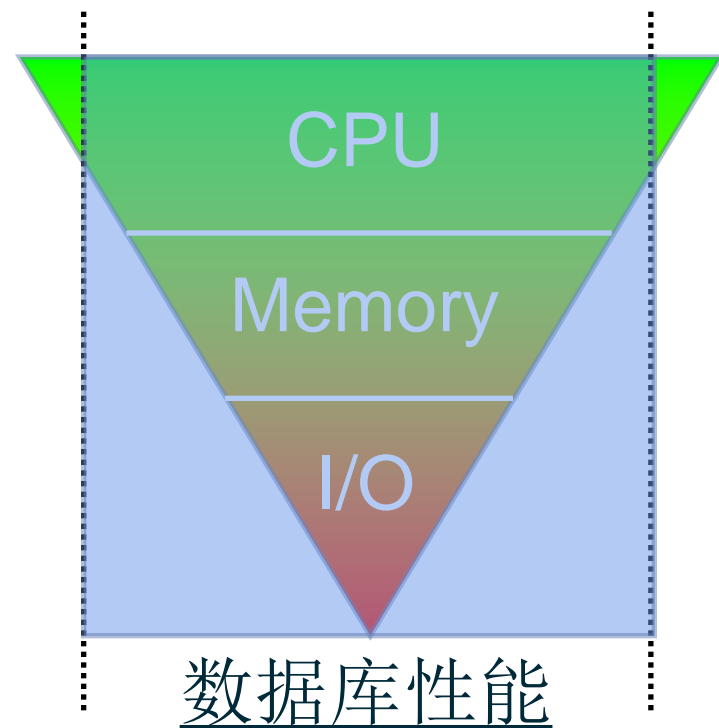
# 商业方案: Infinite Cache

- 解决: CPU使用率低
- 解决数据库占用内存有限
- 解决: 磁盘读写混合操作I/O压力大



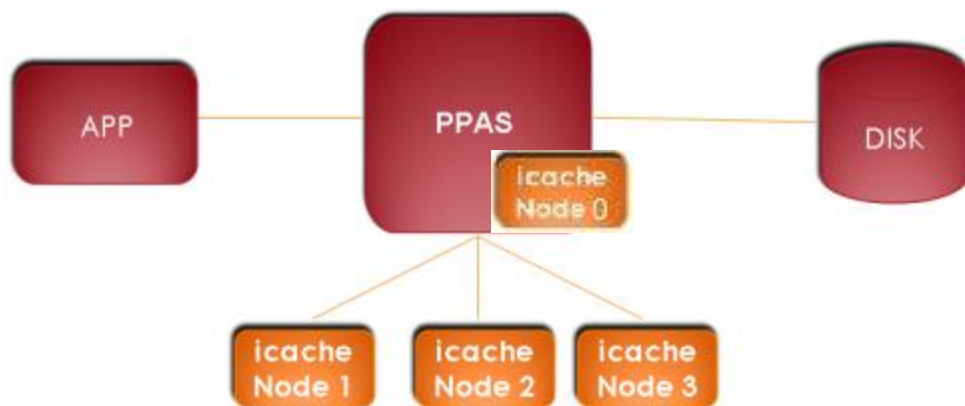
# 商业方案: Infinite Cache

- 解决: CPU使用率低  
常用数据压缩到基于NoSQL的内存
- 解决: 数据库占用内存有限  
通过压缩数据扩展内存空间  
通过网络无限扩展服务器
- 解决: 磁盘读写混合操作I/O压力大  
常用数据在内存中取出减少I/O操作  
在单服务器中实现最大限度读写分离

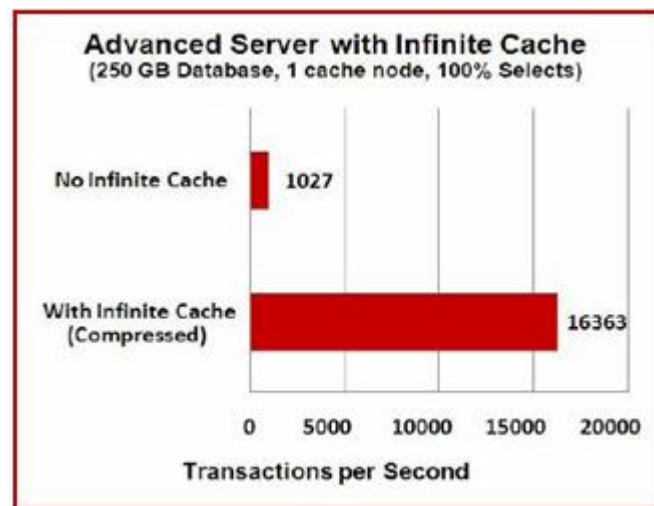


# 商业方案: Infinite Cache

- 💧 利用内存打破磁盘I/O瓶颈
- 💧 透明操作颠覆内存数据库概念
- 💧 支持无限横向扩展
- 💧 高达12倍内存压缩保存更多数据
- 💧 实现高达16倍的性能提升



有无**Infinite Cache**的性能差别





# 高可用性集群架构简介

Appendixes C

# 为何要实现高可用(HA)集群

- 🟢 监控：硬件、操作系统、应用软件
- 🟢 自动的Failover操作，避免单点故障
- 🟢 通过HA组件或自定义脚本进行标准化作业
- 🟢 如何评价HA集群？

|     |    |     |      |     |   |
|-----|----|-----|------|-----|---|
| 9   | 9. | 9   | 9    | 9   | % |
| 35天 | 4天 | 8小时 | 50分钟 | 5分钟 |   |

系统可用率 = ( 总评估时间 - 总失效时间 ) / 总评估时间

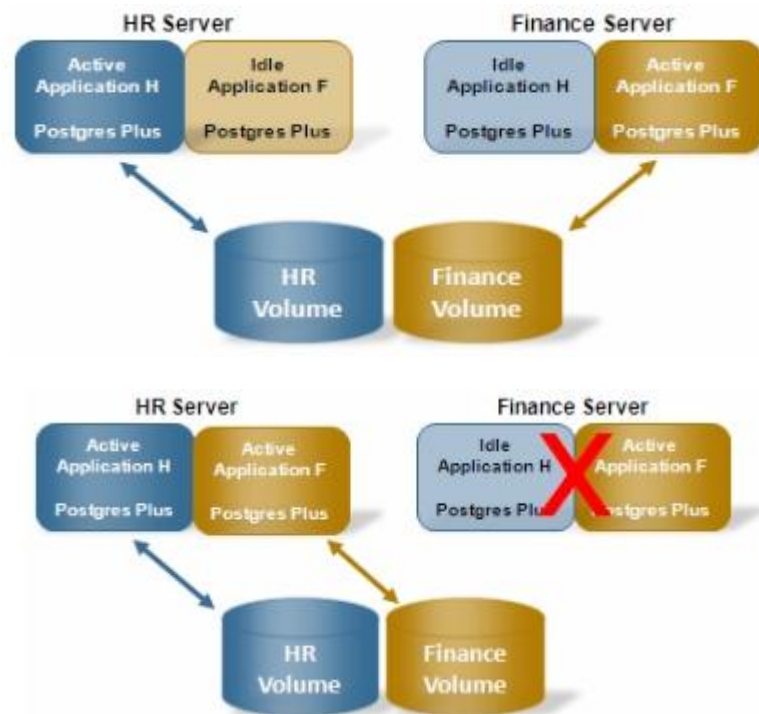
当中：总失效时间 = 计划内停机时间 + 计划外停机时间

# 规划您的高可用集群

- 💧 成本！
- 💧 恢复时间标准：要求多高的系统可用率？
- 💧 恢复数据标准：能接受丢失多少数据？
- 💧 备用数据库访问
- 💧 1、活动（RW）                      2、活动（RO）                      3、备用
- 💧 对主数据库的性能影响

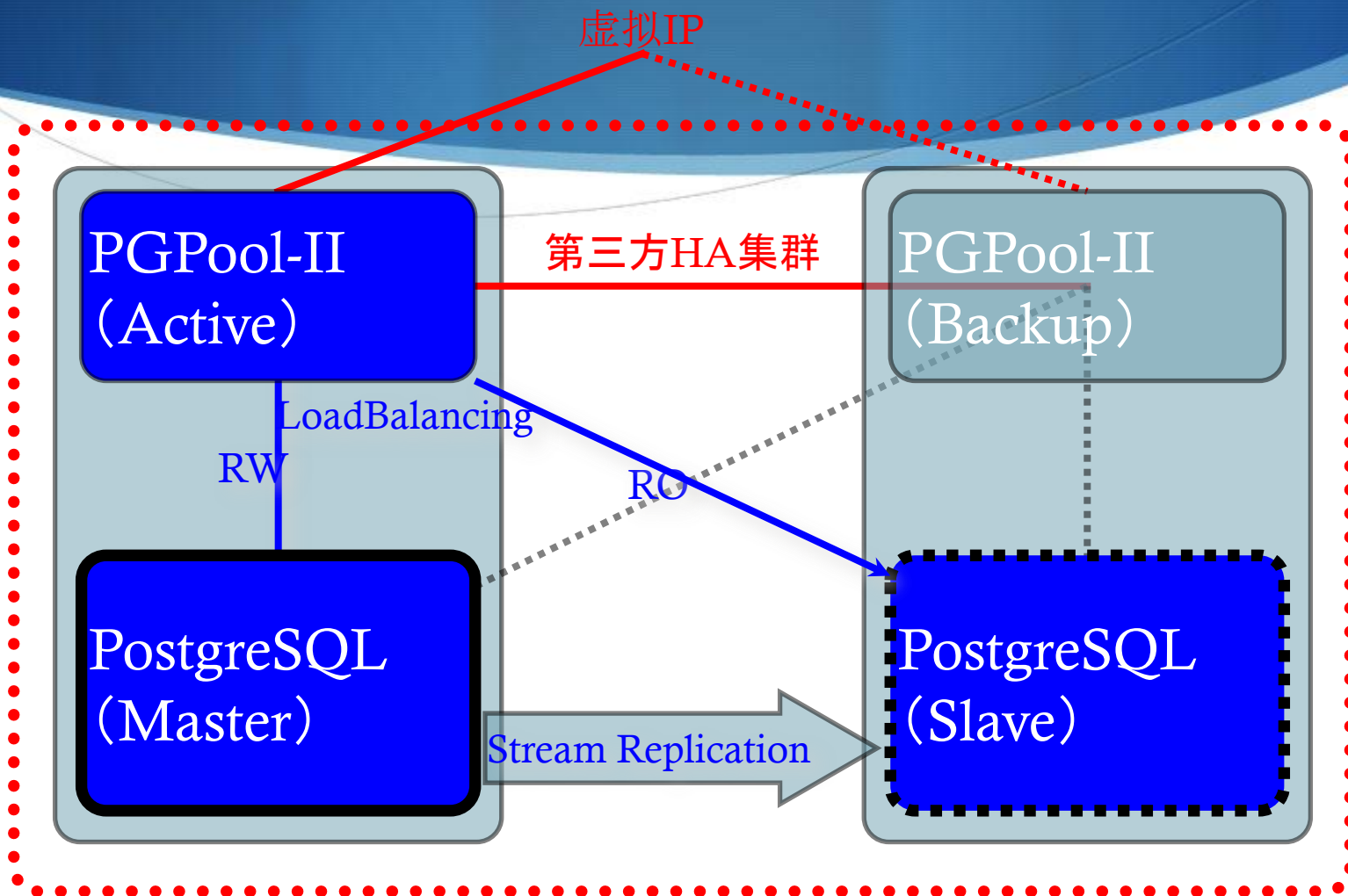
# 基于共享存储的高可用集群

- ◆ 多台服务器组成集群
- ◆ 对外提供统一访问接口
- ◆ 避免系统单点故障
- ◆ 全自动监控及切换
- ◆ 支持多程集群平台
  - ✓ Red Hat Cluster Suite
  - ✓ LifeKeeper
  - ✓ Sun Solaris Cluster Suite
  - ✓ Microsoft MSCS
  - ✓ 等等.....

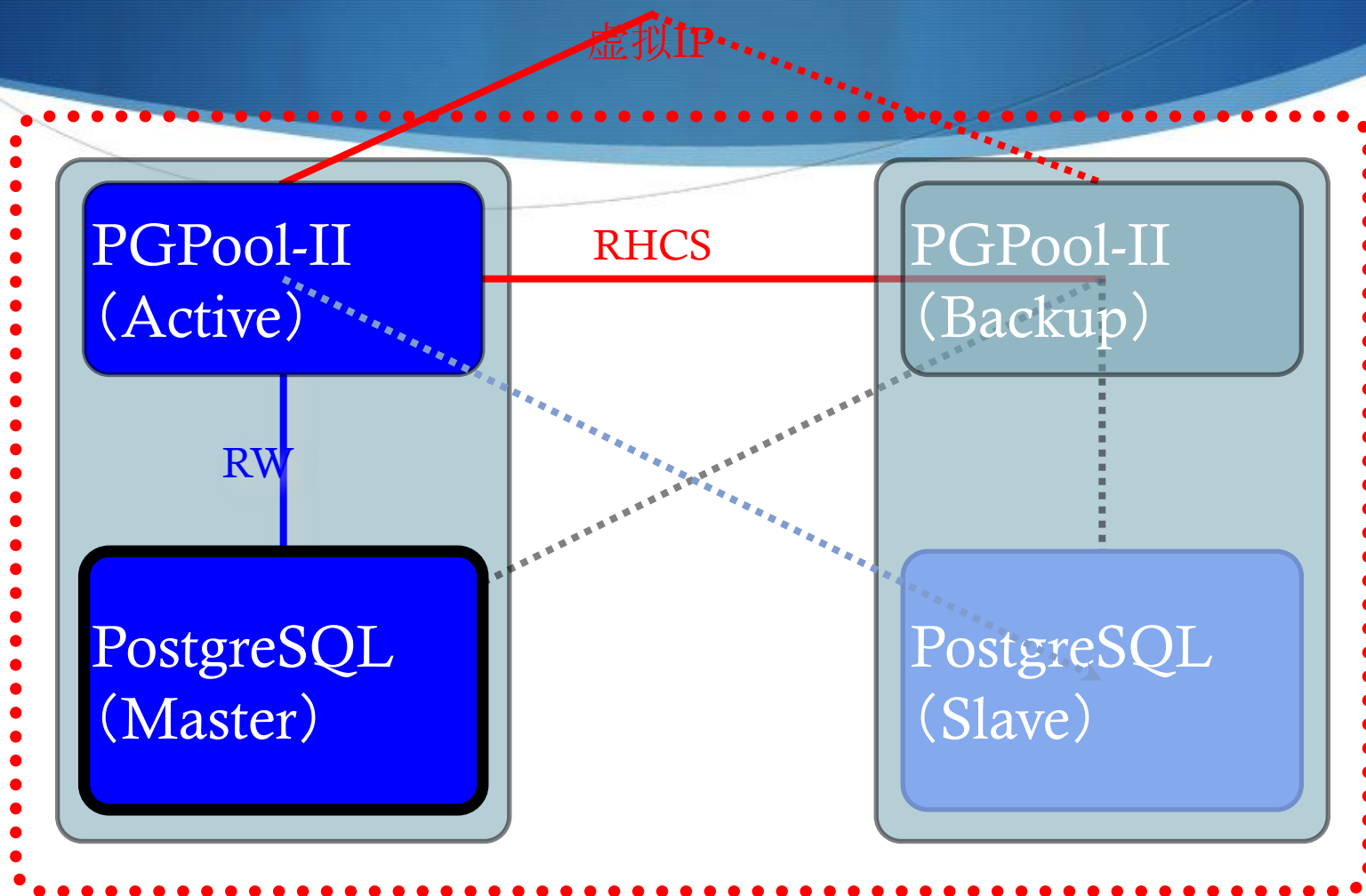


最高可达**99.99%**的可用性

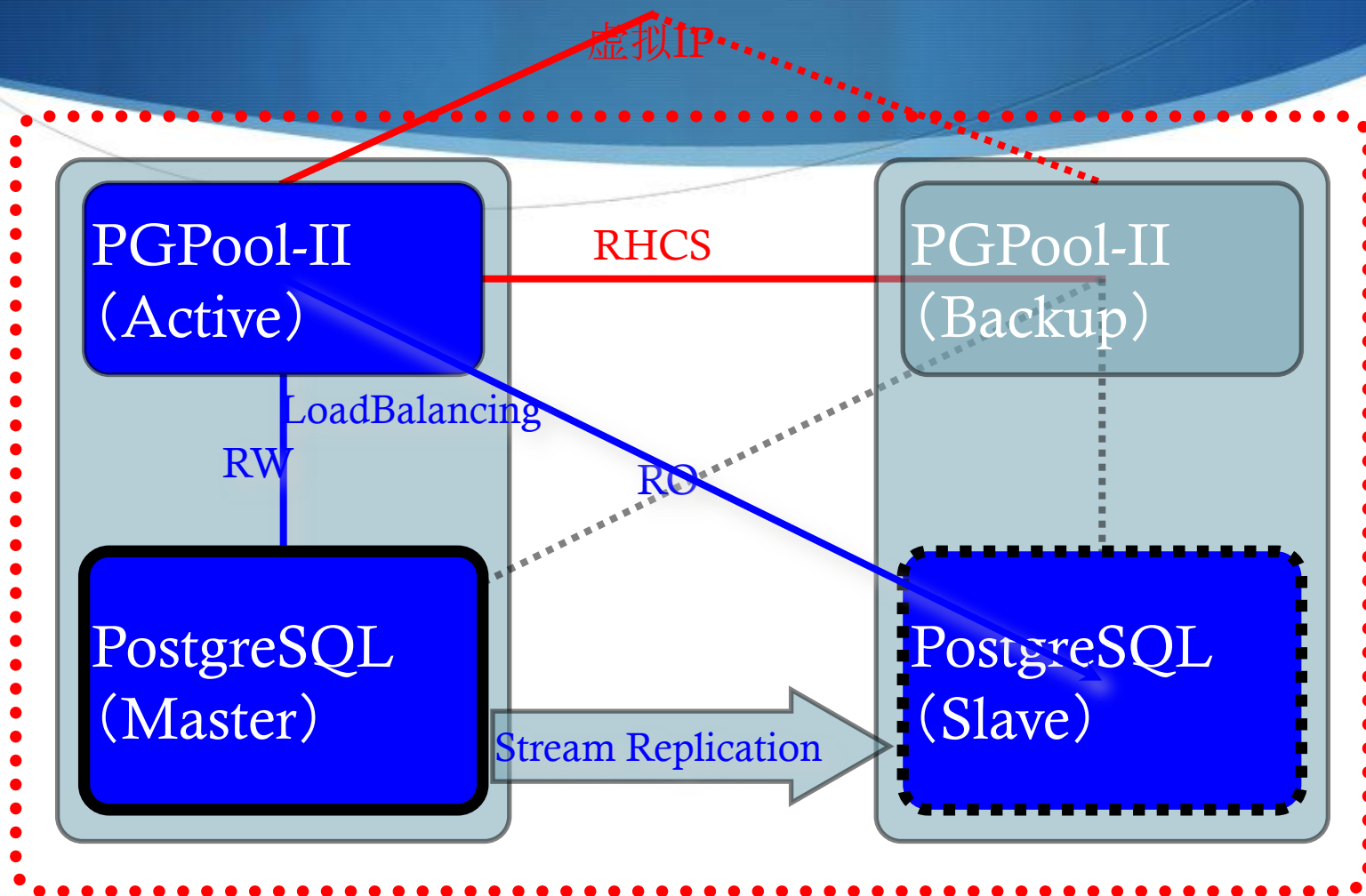
# 基于PGPool-II的高可用集群



# 启动2台服务器并启动主库

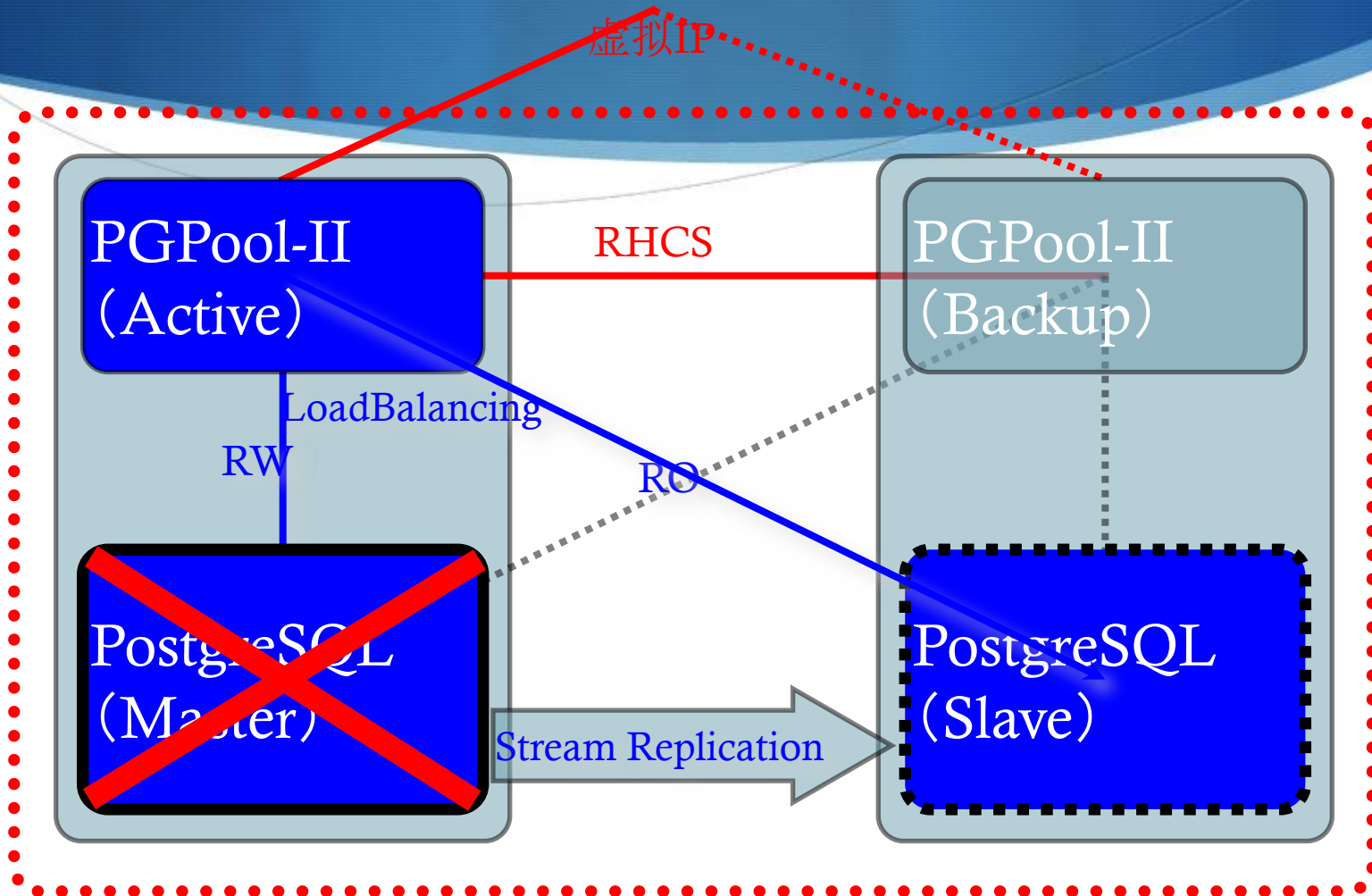


# 备库启动为Slave模式

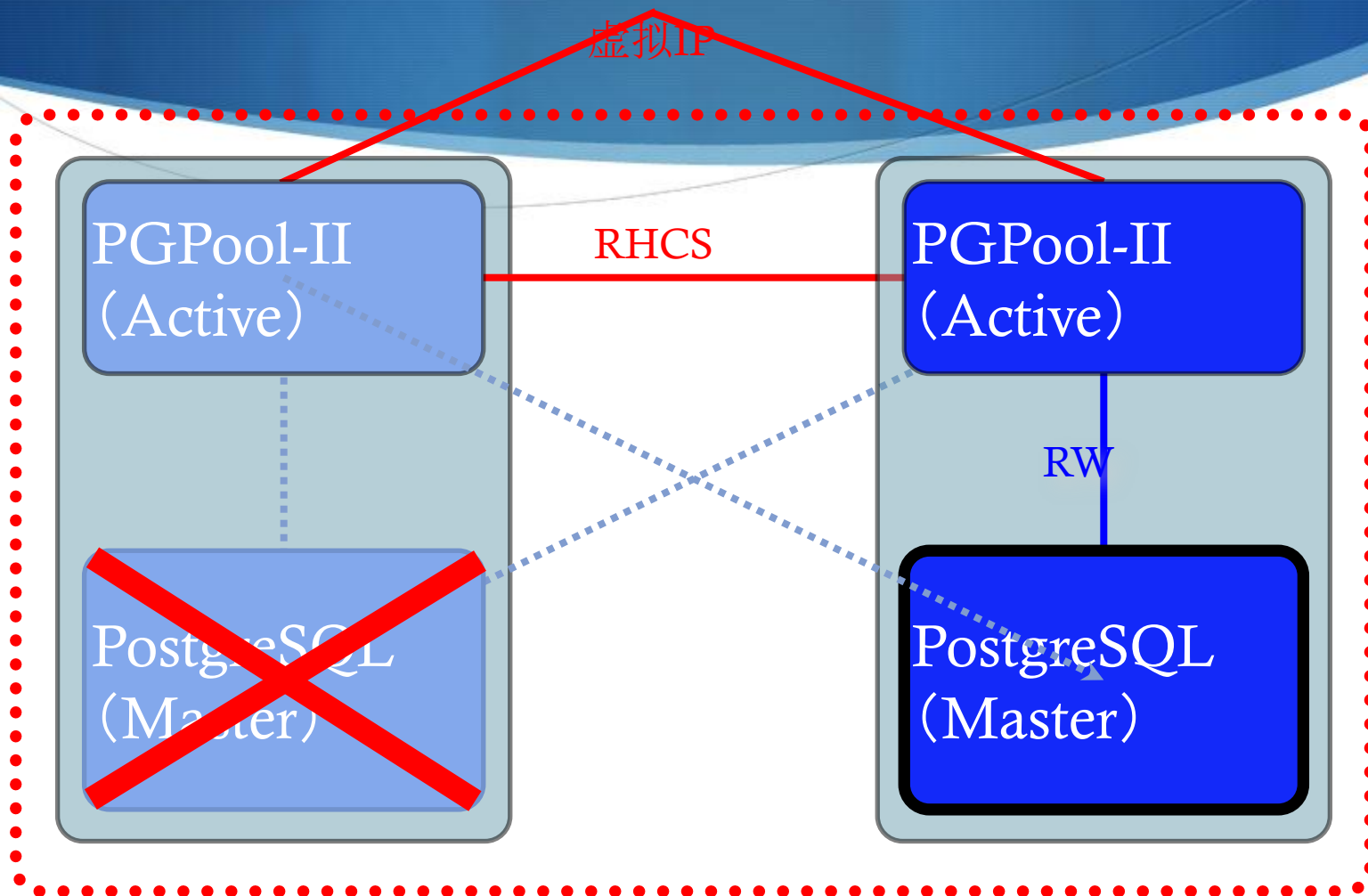




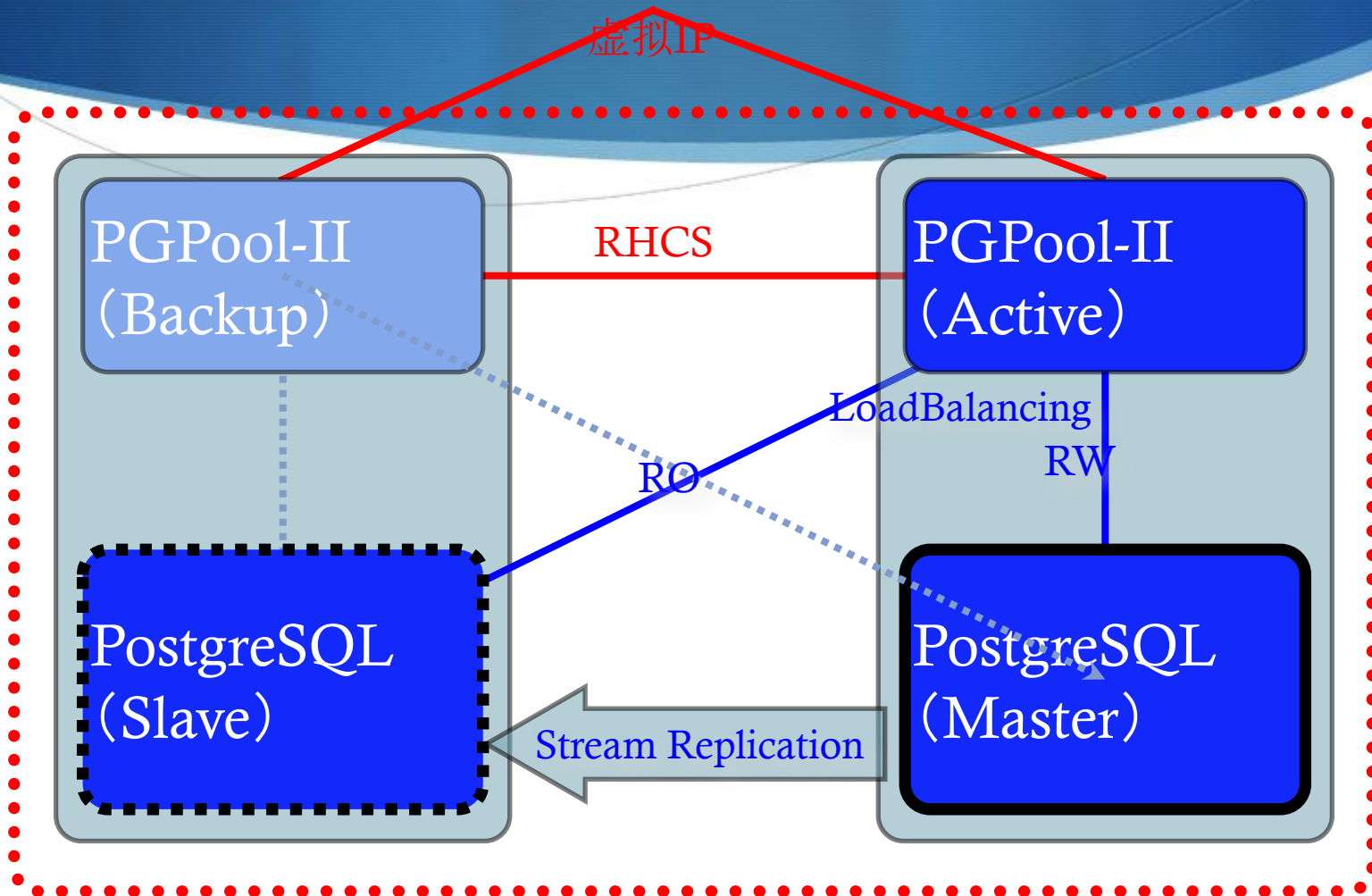
# 模拟主库崩溃



# 模拟主库崩溃



# 模拟主库崩溃



# 集中管理及性能监控工具

Appendixes D

# 管理及监控工具(开源或免费1)

- 💧 <http://www.hyperic.com/products/postgresql-monitoring>
- 💧 <http://demo.munin-monitoring.org/munin-monitoring.org/demo.munin-monitoring.org/index.html>
- 💧 [http://www.cybertec.at/en/postgresql\\_products/pgwatch-cybertec-enterprise-postgresql-monitor](http://www.cybertec.at/en/postgresql_products/pgwatch-cybertec-enterprise-postgresql-monitor)

# 管理及监控工具(开源或免费2)

💧 <http://www.dbwrench.com/>

💧 <http://mogwai.sourceforge.net/erdesignerng.html>

# 可管理及监控工具(商业)

- ◆ <http://www.datanamic.com/dezign/index.html>
- ◆ <http://www.sqlmaestro.com/products/postgresql/maestro/>
- ◆ [http://www.navicat.com/en/products/navicat\\_pgsql/pgsql\\_overview.html](http://www.navicat.com/en/products/navicat_pgsql/pgsql_overview.html)
- ◆ <http://enterprisedb.com/products-services-training/products/postgres-enterprise-manager/capacity-planning-forecasting>



# 感谢参加 PostgreSQL DBA培训

张文升 vincent

13684090848

wensheng.zhang@postgres.cn