

Использование PostgreSQL в веб-приложениях

*Проблемы повышения производительности,
масштабирования, надёжности веб-приложений*



Николай Самохвалов
Иван Золотухин
компания Postgresmen

План действий

10 ⁰⁰	Начало
11 ⁰⁰	Перерыв (10 мин)
12 ⁰⁰	Перерыв (10 мин)
13 ⁰⁰	Обед (1 час)
15 ³⁰	Кофе-брейк (30 мин)
16 ⁴⁵	Перерыв (10 мин)
18 ⁰⁰	Завершение программы

План работ

1. Вводная часть
2. Диагностика. Тестирование производительности. Мониторинг
3. Тюнинг (postgresql.conf, параметры ОС)
4. Проблемы проектирования БД
5. Как найти медленные запросы?
6. Оптимизация запросов. EXPLAIN & EXPLAIN ANALYZE
7. Индексы. Специальные типы данных
8. Кэширование и СУБД
9. Отказоустойчивость и HA
10. Балансировка нагрузки. Масштабирование
11. Выбор железа

Что такое PostgreSQL?

PostgreSQL – свободно распространяемая объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

PostgreSQL сегодня

Текущая версия – 8.2 (8.2.4)

8.3 ожидается в сентябре-октябре 2007 :-)

PostgreSQL сегодня

- **Надежность**
ACID, MVCC, WAL, PITR, Slony
- **Безопасность данных**
root, SSL, pg_hba.conf, ROLE
- **Производительность**
B-tree, hash, R-tree, GiST, Gin; geqo; partitioning; Slony, pgpool
- **Расширяемость**
pg_catalog, наследование, GiST, Gin, contribs

PostgreSQL сегодня

- **ISO/ANSI SQL (SQL:200x)**
схемы, представления, триггеры, rules, 2PC...
- **Типы данных**
varlena, массивы, GIS, композитные, GiST
- **Интерфейсы**
C, C++, C#, python, perl, ruby, php, Lisp и т.д.
- **Процедурные языки**
PL/pgSQL, pl/Tcl, PL/Perl и pl/Python; PHP, Java, Ruby, R, shell

Кто использует

- Sony Entertainment (EnterpriseDB)
- Skype
- Cisco
- Fujitsu
- NTT
- Apple
- SUN Microsystems (Solaris, 24x7 support)
- hi5.com
- UNISYS

Кто использует

- SourceForge
- LAMP: Linux/Apache/Middleware(Perl,PHP,Python,Ruby)/PostgreSQL
- New Zealand's Electoral Enrolment Centre
- .ORG, .INFO domain registry
- Многотерабайтные архивы астрономических данных

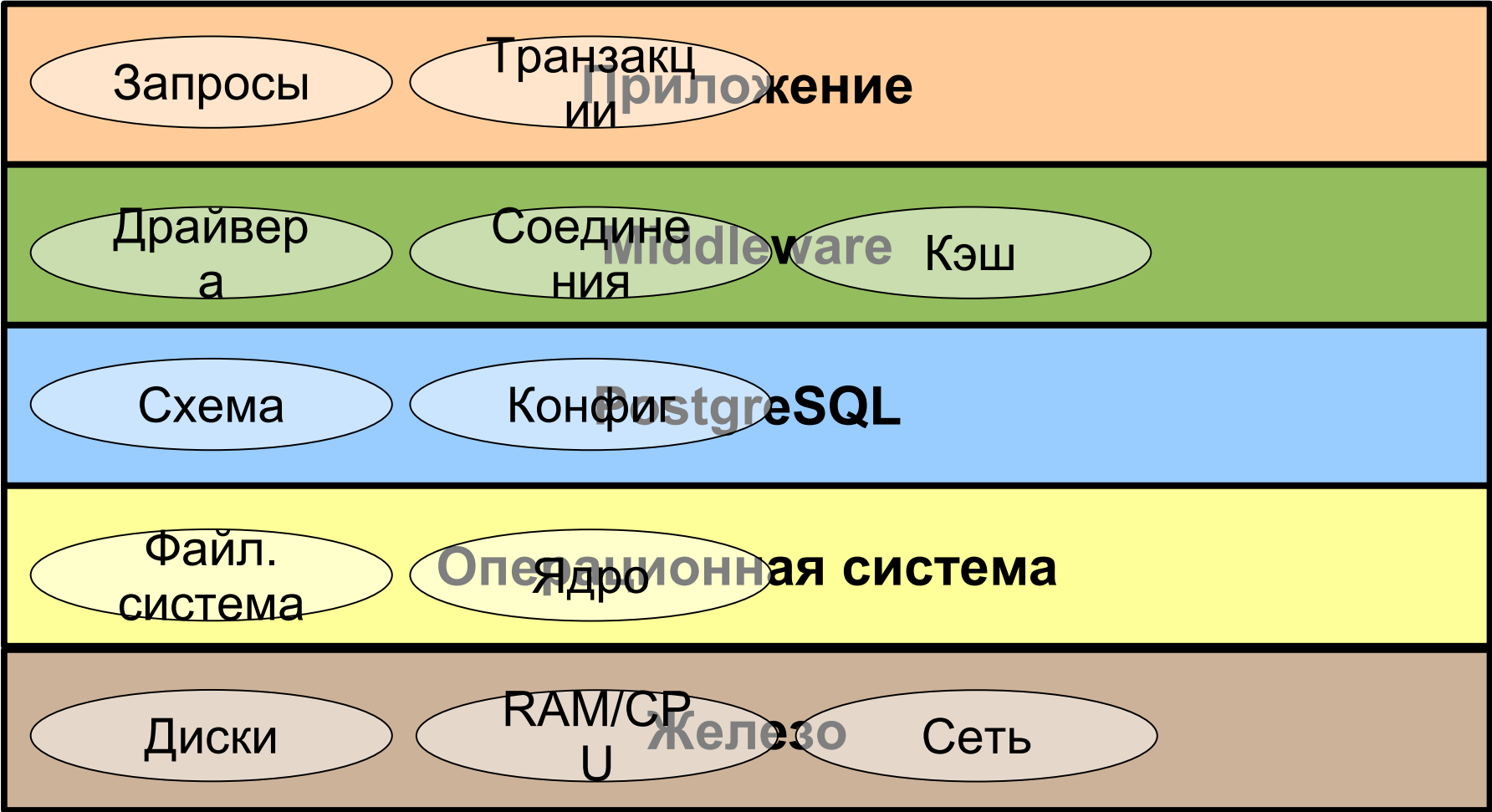
Кто использует: Россия

- Рамблер
- 1С:Предприятие (наряду с MS SQL)
- irr.ru («Из рук в руки»)
- rabota.ru («Работа для вас»)
- price.ru
- moikrug.ru (Яндекс)
- webalta.ru
- РБК
- Beeline
- Мастерхост
- neznamotka.ru

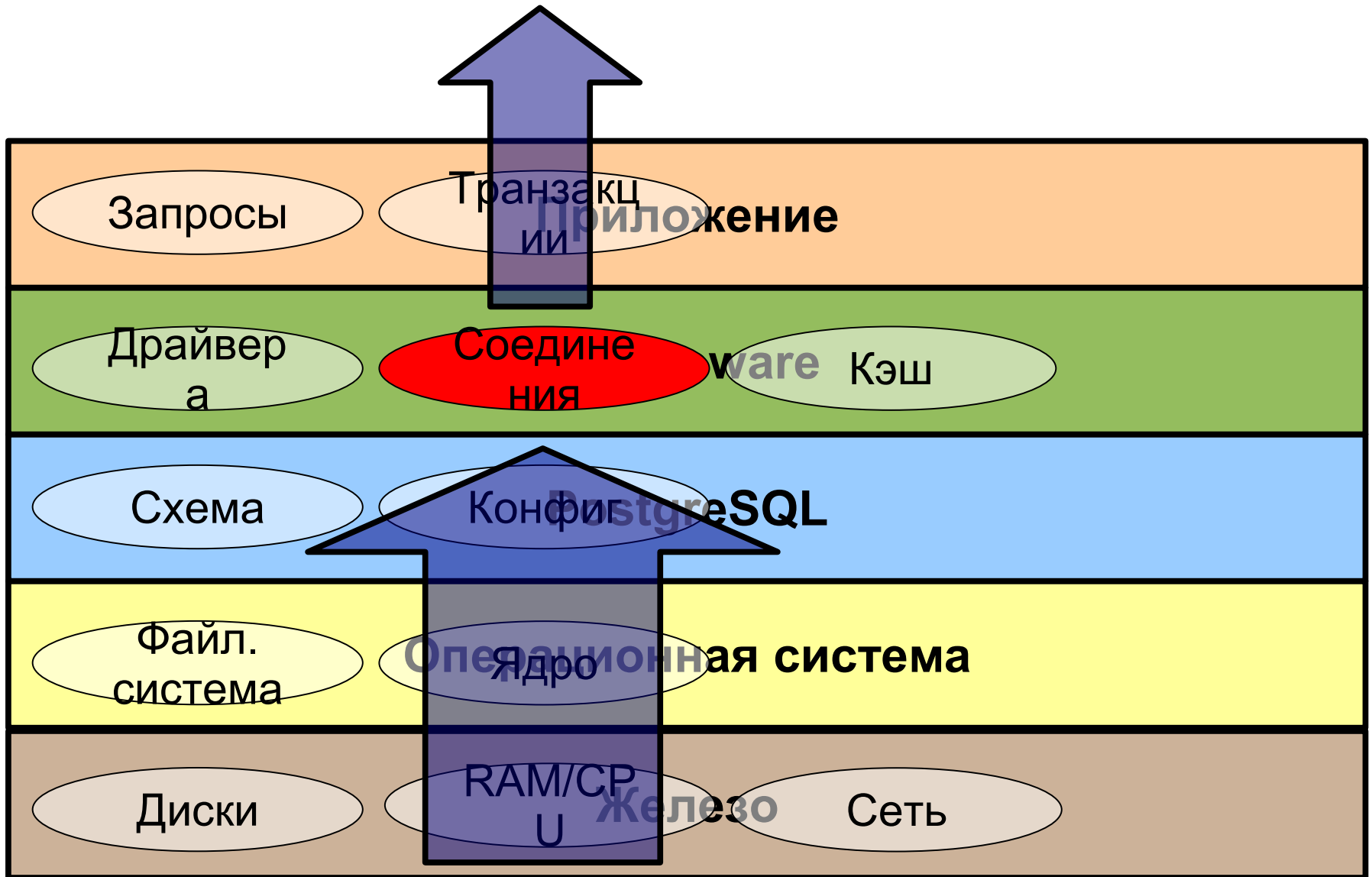
Кто использует: Россия

- udaff.com

Аспекты производительности



Аспекты производительности



Правила оптимизации

- Большинство проблем PostgreSQL на самом деле не являются проблемами PostgreSQL
- 10% проблем порождают 90% ухудшения производительности (the hockey stick)
- Как правило, при оптимизации видна только крупнейшая проблема

The Hockey Stick

Влияние на производительность



Количество проблем

Устройство PostgreSQL

- ACID
- MVCC
- WAL
- Shared Memory

Устройство PostgreSQL

- ACID-совместимая база данных
 - atomicity (атомарность)
 - consistency (непротиворечивость)
 - isolation (изоляция)
 - durability (надежность)

Устройство PostgreSQL

- ACID-совместимая база данных

уровни изоляции транзакций:

- Read Uncommitted
- **Read Committed** (разумный компромисс)
- Repeatable read
- **Serializable** (надежно, но медленно)

Вывод: забудьте про SET TRANSACTION, если вы не в банке

Устройство PostgreSQL

- MVCC: Multiversion Concurrency Control
 - xid – transaction id
 - каждая запись имеет xid_start и xid_end
 - каждая транзакция видит версию базы в момент xid_start
 - записи не удаляются, а просто помечаются xid_end

Устройство PostgreSQL

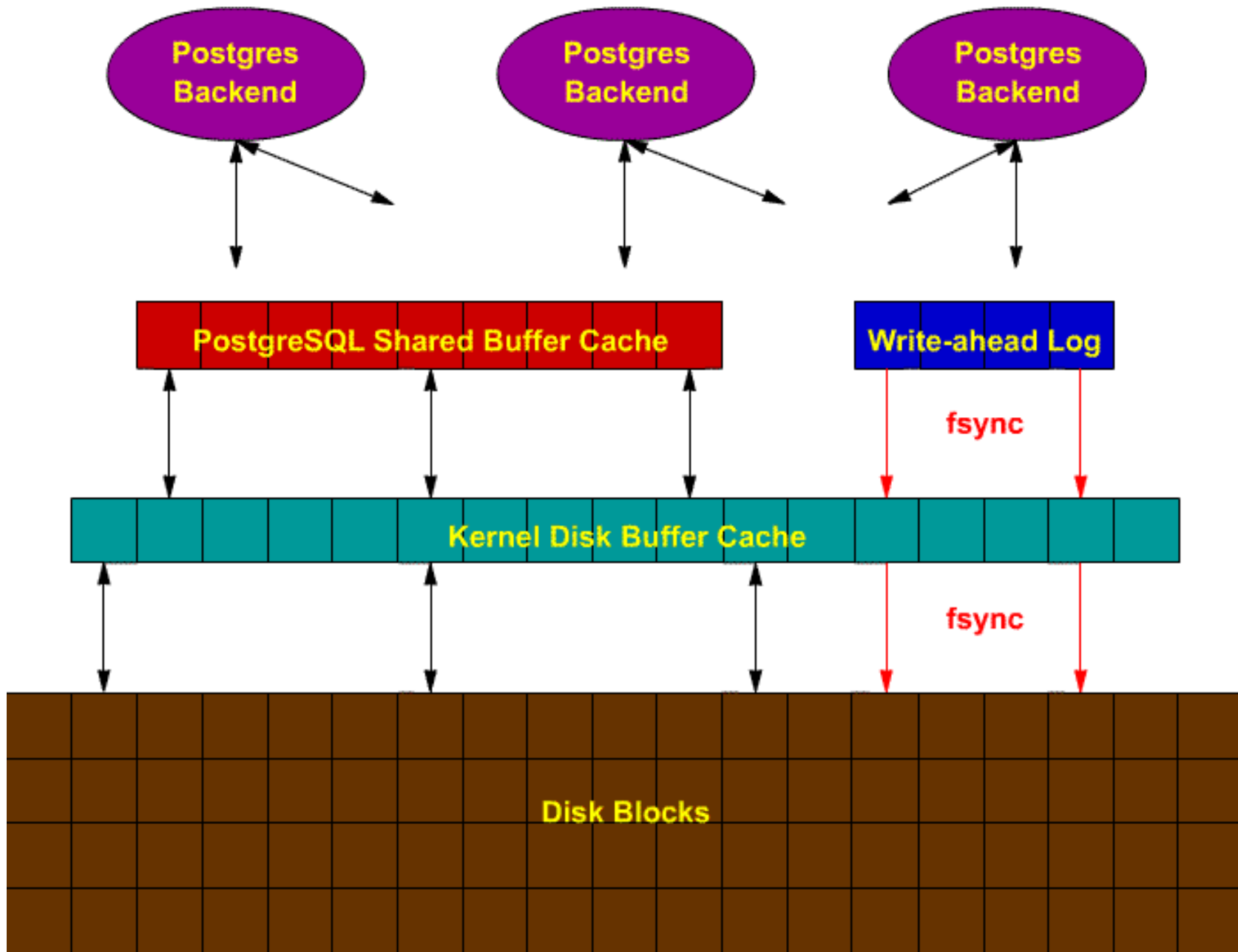
- MVCC – накапливаются старые версии данных
- требует пылесоса – VACUUM
- VACUUM: re-use мертвых данных
- VACUUM FULL: физическое удаление мертвых данных и дефрагментация базы
- Autovacuum – ваш друг

Устройство PostgreSQL

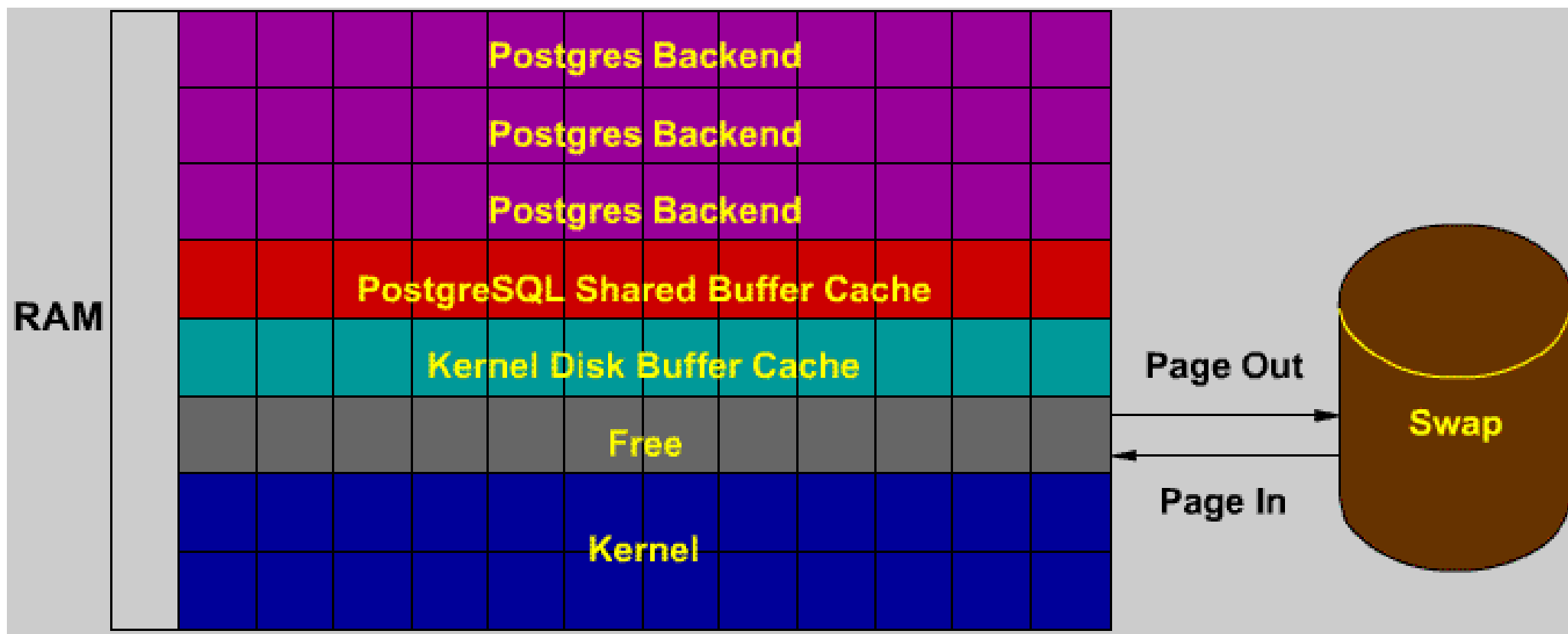
WAL – Write Ahead Log

- механизм протоколирования всех транзакций
- позволяет восстановить систему после возможных сбоев
- **все изменения данных записываются на диск только после их гарантированного журналирования в WAL**
- PITR – Point In Time Recovery

Устройство PostgreSQL



Устройство PostgreSQL



Диагностика проблем

- Средства операционной системы
- Тесты производительности (benchmarks)
- Средства PostgreSQL: логи, статистика
- Системы внешнего мониторинга

Средства операционной системы

- `ps`

Возможность увидеть PostgreSQL-процессы

- Понимание конкурентной работы с CPU и RAM
- Возможность заметить долгие запросы

Средства операционной системы

- `mpstat`

Просмотр активности каждого CPU

- Используются ли все процессоры?
- Является ли CPU узким местом?
- Диагностика context switches

Средства операционной системы

- `vmstat`, `free`

Просмотр использования памяти

- Оценка размеров дискового кэша
- Хватает ли серверу PostgreSQL памяти?
- Не происходит ли `swapping`-а?

Средства операционной системы

- `iostat`

Просмотр дисковой активности

- Сервер достиг предела по I/O?
- Все ли диски выдают ожидаемый I/O?
- Мониторинг checkpoint-ов

Тесты производительности (benchmarks)

- `bonnie++`

Тест дисковой подсистемы

- Измерьте производительность дисков
- Знайте скорость Sequential/Random Read/Write и Random Seek своих дисков

Тесты производительности (benchmarks)

- `contrib/pgbench`

Простейший тест базы данных

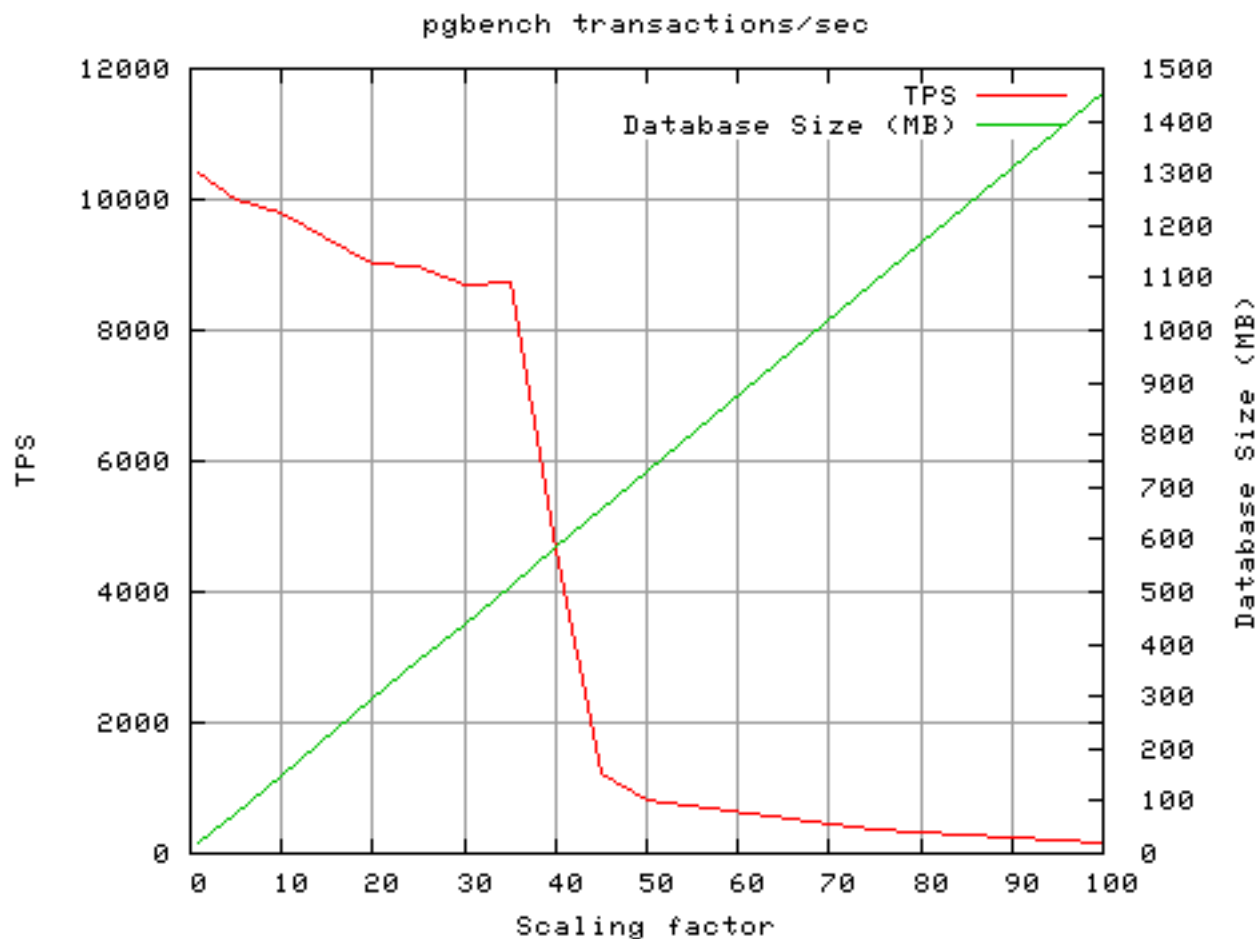
- Тестирует I/O и скорость соединений
- Не тестирует: блокировки, вычислительные задачи, планирование запросов
- Удобен для обнаружения больших HW+OS проблем, мало пригоден для тонкой настройки

Тестируйте систему, подобную вашей

- База в `shared_buffers`
- База в дисковом кэше
- База не помещается в RAM

Тесты производительности (benchmarks)

- contrib/pgbench



Средства PostgreSQL

- `pg_stat_database,`
`pg_database_size()`

Вы должны знать общие параметры БД:

- Количество соединений
- Количество транзакций
- Количество commit-ов и rollback-ов
- Количество hit-ов (попаданий в буфер)
- Размер базы данных (помещается в RAM?)

Средства PostgreSQL

- `pg_tables, pg_relation_size()`

Вы должны знать общие параметры таблиц:

- Количество таблиц
- Размеры таблиц
- Размеры индексов
- Количество триггеров
- «Вздутие» таблиц (bloating)

Средства PostgreSQL

- `pg_stat_activity`, `pg_locks`

Детали конкурентного доступа к таблицам

- Оценка количества idle-соединений
- Runtime-оценка долгих запросов
- Лучше, чем `ps` – больше деталей
- Все ли нормально с блокировками? Нет ли WAITING-бакендов?

Средства PostgreSQL

- `pg_stat[io]_user_tables,`
`pg_stat[io]_user_indexes`

Статистика доступа к таблицам

- Количество SELECT/INSERT/UPDATE/DELETE
- Количество seqscan vs. indexscan
- Есть ли неиспользуемые индексы? Drop it!
- Есть seqscan-ы по большим таблицам? CREATE INDEX

Средства PostgreSQL

- `pg_stat_bgwriter`
 - Можно видеть, как `bgwriter` чистит кэш
 - Помогает при затруднении проходимости checkpoint-ов

Системы внешнего мониторинга

- PgFouine (анализ логов)
- Zabbix
 - Легко расширяется
- Nagios
 - Готовое расширение для PostgreSQL
- pgsnmpd – SNMP-агент для PostgreSQL

Системы внешнего мониторинга

Примеры запросов для мониторинга:

```
select datname,now()-query_start as duration,current_query from  
  pg_stat_activity;
```

```
select datname, case when blks_read = 0 then 0 else blks_hit /  
  blks_read end as ratio from pg_stat_database;
```

```
select relname,seq_scan,idx_scan, case when idx_scan = 0 then 100  
  else seq_scan / idx_scan end as ratio from pg_stat_user_tables  
  order by ratio desc;
```

```
select relname,n_tup_ins,n_tup_upd,n_tup_del from pg_stat_user_tables  
  order by n_tup_upd desc;
```

```
select indexrelname,idx_tup_read,idx_tup_fetch,case when  
  idx_tup_fetch = 0 then 100 else idx_tup_read / idx_tup_fetch end as  
  ratio from pg_stat_user_indexes order by ratio desc;
```

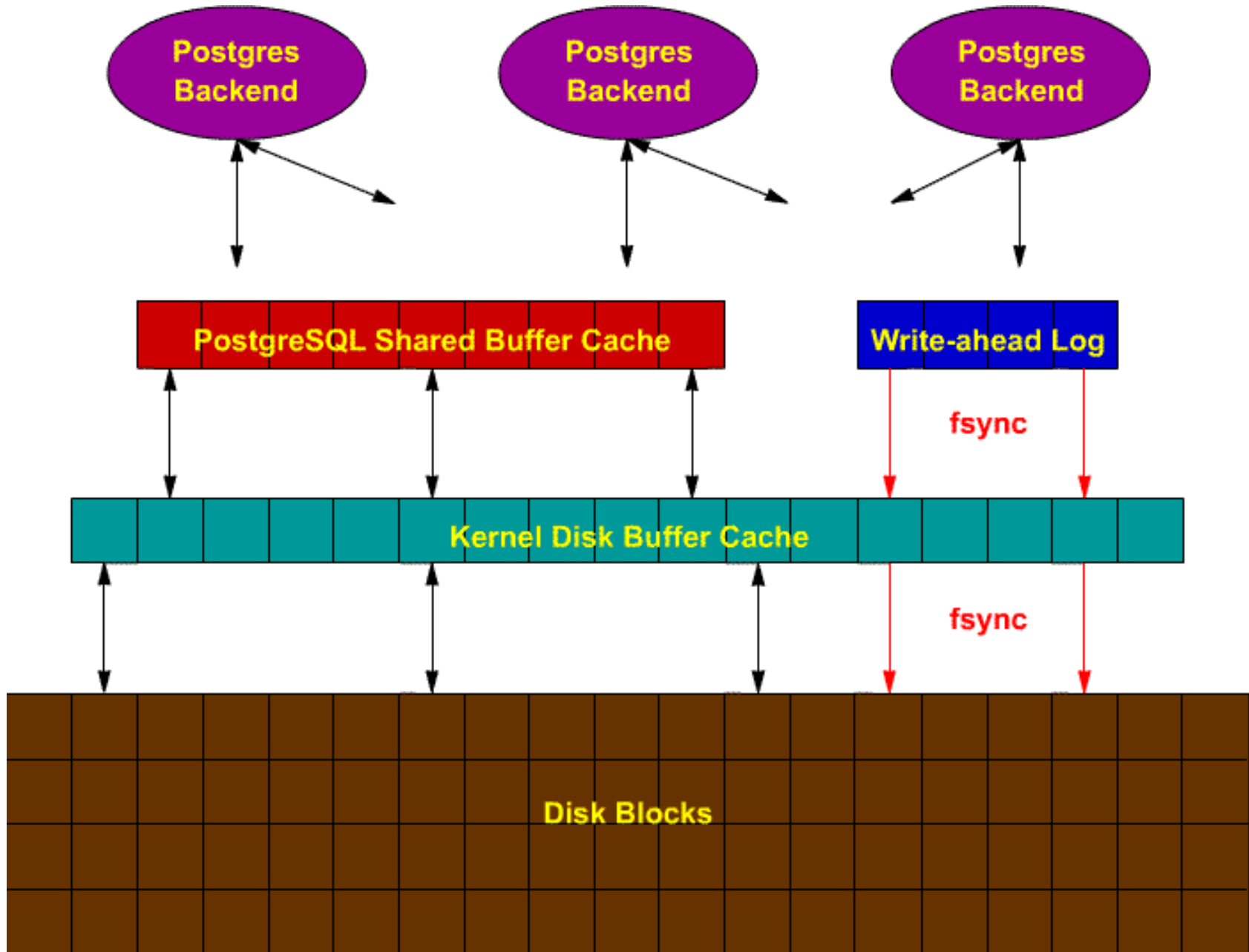
```
select relname,n_tup_ins,n_tup_upd,n_tup_del from pg_stat_user_tables  
  order by n_tup_upd desc;
```

```
select l.mode,d.datname,c.relname,l.granted,l.transactionid from  
  pg_locks as l left join pg_database as d on l.database= d.oid left  
  join pg_class as c on l.relation = c.oid;
```

postgresql.conf

- `shared_buffers = 25 ÷ 35% RAM`
Размер разделяемой памяти
- `work_mem = free / max_nconn * 1 ÷ 3`
Неразделяемая память для сортировок, агрегирования, вычисления хэш-функций.
`malloc` до нескольких раз за соединение
- `effective_cache_size = 2/3 * RAM`
Средний размер дискового кеша
- `checkpoint_timeout = 600 ÷ 900 sec`
Макс. время между WAL checkpoint-ами

Устройство PostgreSQL



postgresql.conf

- `checkpoint_segments = 3 ÷ 10`

Макс. время между WAL checkpoint-ами, в WAL-сегментах размером 16MB

- `commit_delay, commit_siblings`

Задержка и сброс нескольких активных транзакций в WAL одним вызовом `fsync`

- `maintenance_work_mem = 75% от pg_relation_size('the_biggest_table')`

Размер памяти для VACUUM, ANALYZE, CREATE INDEX

postgresql.conf

- `max_fsm_pages = (n измененных рядов)`

Количество дисковых страниц, используемых для Free Space Map. Правильная настройка может отменить необходимость в VACUUM FULL и REINDEX.
- `random_page_cost = 2.0 ÷ 4.0`

Оценка времени случайного доступа (пробег по индексу). 2.0 в случае быстрых дисков, 4.0 в случае, если БД не помещается в RAM
- `wal_buffers = default`

Размер буфера WAL в разделяемой памяти, важен для больших транзакций. Забудьте это.

postgresql.conf

- `log_destination = 'syslog'`
- `redirect_stderr = off`
- `silent_mode = on`
- `log_min_duration_statement = 500`
- `log_duration = off`
- `log_statement = 'none'`

Настройки для логгирования через `syslog` запросов, медленнее 500ms (pgFouine)

postgresql.conf

- `stats_command_string = on`
- `update_process_title = on`
- `stats_start_collector = on`
- `stats_block_level = on`
- `stats_row_level = on`

Настройки для сбора row-level статистики.

Ухудшают производительность на ~5%, но дают возможность работы с вышеуказанными системными представлениями и необходимы для autovacuum

postgresql.conf

- `autovacuum = on`

Автовакуум

- `autovacuum_naptime = 1 ÷ 10 min`

Время между последовательными
автоматическими запусками автовакуума

- `autovacuum_vacuum_threshold,`
`autovacuum_analyze_threshold = 250 ÷`
`1000`

Пороги по количеству измененных рядов для
запуска VACUUM или ANALYZE на конкретную
таблицу

postgresql.conf

- `vacuum_cost_delay = 50 ÷ 250 ms`
Откладывание VACUUM на данное время
- `vacuum_cost_limit = 100 ÷ 200`
Откладывание VACUUM при достижении данной стоимости операций
- `vacuum_page_hit = 1 ÷ 10`
Условная стоимость одной из основных операций VACUUM

Bgwriter & shared buffers

- *flags* (BM_DIRTY)
- *refcount* (is pinned?)
- *usage_count* (BM_MAX_USAGE_COUNT)

- LRU – Least Recently Used
- Round

Bgwriter & shared buffers

- `bgwriter_delay` = 50 ÷ 200 ms

Пауза между циклами bgwriter-a

- `bgwriter_lru_percent` = 1.0 ÷ 5.0

Предел на сканирование LRU-буферов / цикл

- `bgwriter_all_percent` = 0.3 ÷ 1.0

Предел на сканирование всех буферов / цикл

- `bgwriter_lru_maxpages` = 5 ÷ 20

Количество LRU-буферов, записанное на диск /
ЦИКЛ

- `bgwriter_all_maxpages` = 5 ÷ 20

Производительность PostgreSQL

- Диски > RAM > CPU
- Чем больше дисков, тем лучше
- Отделяйте pg_xlog от данных
- RAID 1+0 / 0+1 > RAID 5
- Не ставьте на сервер другие приложения

Проектирование БД

- Разумная денормализация
- «Медленный» `count()`
- Логическое секционирование
- Физическое секционирование
- Суррогатные ключи
- NULLs

Денормализация

- Производные атрибуты
- Одна «широкая» таблица лучше нескольких «узких»
- Как правило, TOAST (varlena types) работает хорошо
- Теперь есть не только GiST, но и GIN!
- Медленный `count()`: хранение «счётчиков»
- Процесс денормализации – компромисс между производительностью и гибкостью

person	
person_id	SERIAL
person_name	VARCHAR(64)
person_email	VARCHAR(255)
person_age	INT2

message	
message_id	SERIAL
message_subject	VARCHAR(32)
message_body	VARCHAR(2000)
message_from_person_id	INT4
message_to_person_id	INT4
message_from_person_name	VARCHAR(64)
message_to_person_name	VARCHAR(64)

?

Медленный count ()

- Хранение «счётчиков» (дополнительные поля, триггеры)
- Оценочные методы

```
CREATE FUNCTION count_estimate(query text) RETURNS integer AS $$
DECLARE
    rec    record;
    rows  integer;
BEGIN
    FOR rec IN EXECUTE 'EXPLAIN ' || query LOOP
        rows := substring(rec."QUERY PLAN" FROM ' rows=([[[:digit:]]+)' );
        EXIT WHEN rows IS NOT NULL;
    END LOOP;
    RETURN rows;
END;
$$ LANGUAGE plpgsql VOLATILE STRICT;
```

```
SELECT count_estimate('SELECT * FROM foo WHERE r < 0.1');
```

Логическое секционирование

- Вертикальное
 - Разделение данных по таблицам, схемам, базам (хостам)
- Горизонтальное

Наследование, constraint exclusion:

```
CREATE TABLE obj (  
    obj_id INTEGER,  
    obj_created TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE TABLE person(  
    obj_id INTEGER PRIMARY KEY,  
    person_name VARCHAR(32) NOT NULL,  
    CHECK(obj_id >= 1000000000 AND obj_id < 2000000000)  
) INHERITS(obj);  
  
CREATE TABLE car(  
    obj_id INTEGER PRIMARY KEY,  
    obj_type_id INT2 NOT NULL DEFAULT 2,  
    car_model VARCHAR(16) NOT NULL,  
    CHECK(obj_id >= 2000000000 AND obj_id < 3000000000)  
) INHERITS(obj);  
  
SET constraint_exclusion TO on;
```

Физическое секционирование

- Вертикальное
 - Разделение по дискам
 - symlinks
 - tablespaces
 - партиционирование таблиц + tablespaces
 - Разделение по хостам
 - после логического секционирования (напр., Logging DB)
 - contrib/dblink

Физическое секционирование

- Горизонтальное

- По дискам: партиционирование таблиц + tablespaces

```
CREATE TABLE obj ...
```

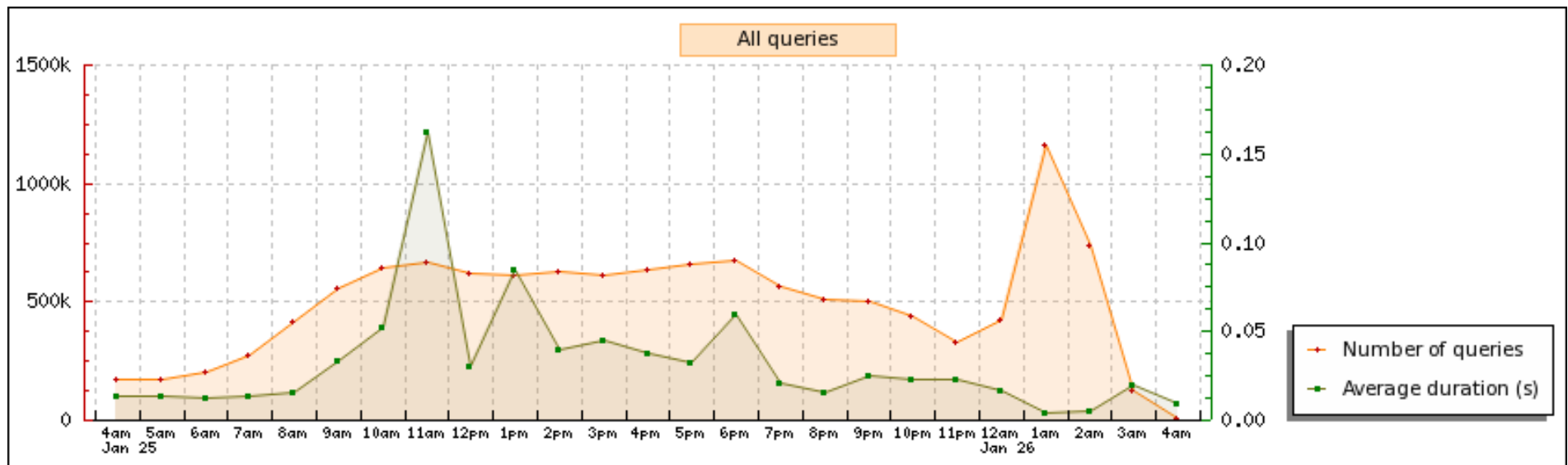
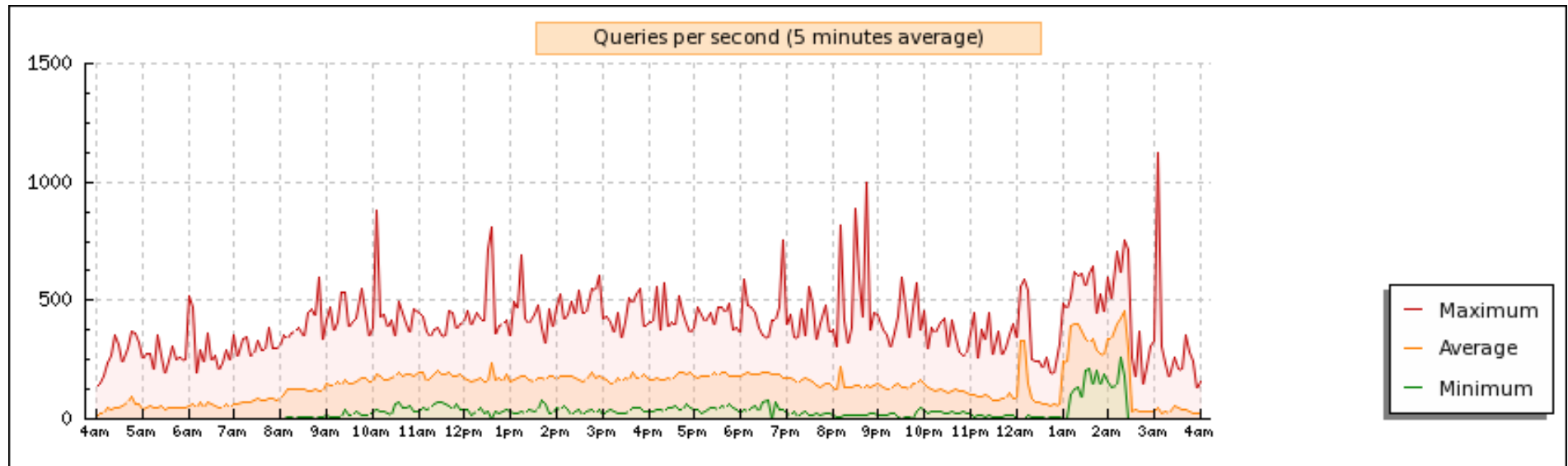
```
CREATE TABLE obj_a(  
    obj_id INTEGER PRIMARY KEY,  
    obj_type_id INT2 NOT NULL DEFAULT 2,  
    CHECK(obj_id >= 2000000000 AND obj_id < 3000000000)  
) INHERITS(obj) TABLESPACE space1;
```

```
CREATE RULE ...  
ON INSERT TO obj WHERE ...  
DO INSTEAD INSERT INTO obj_a ...;
```

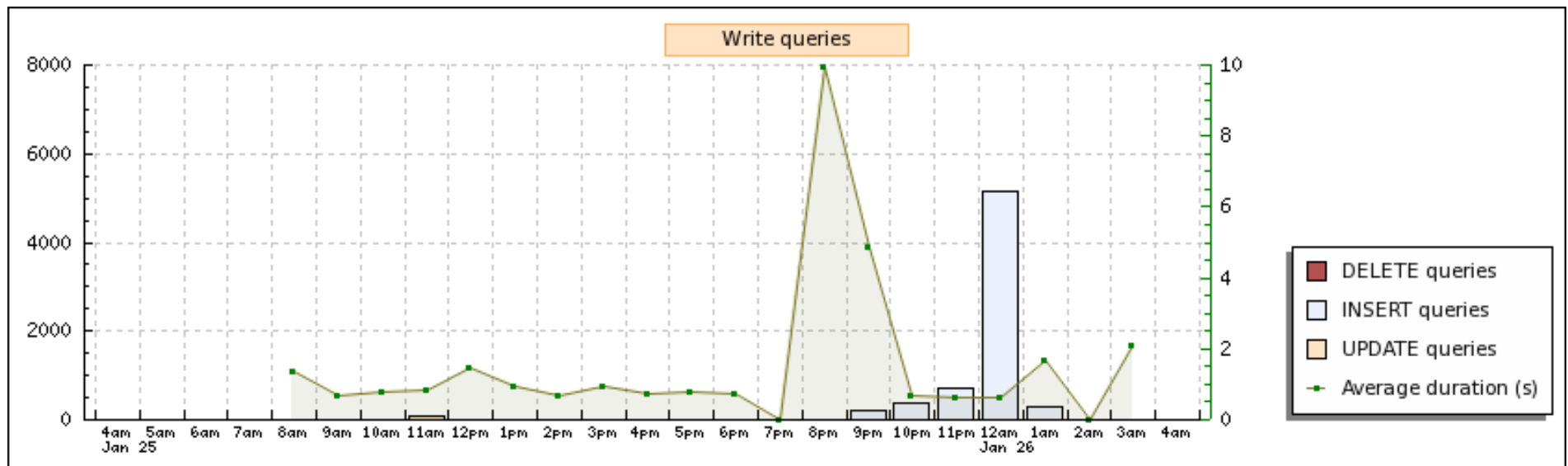
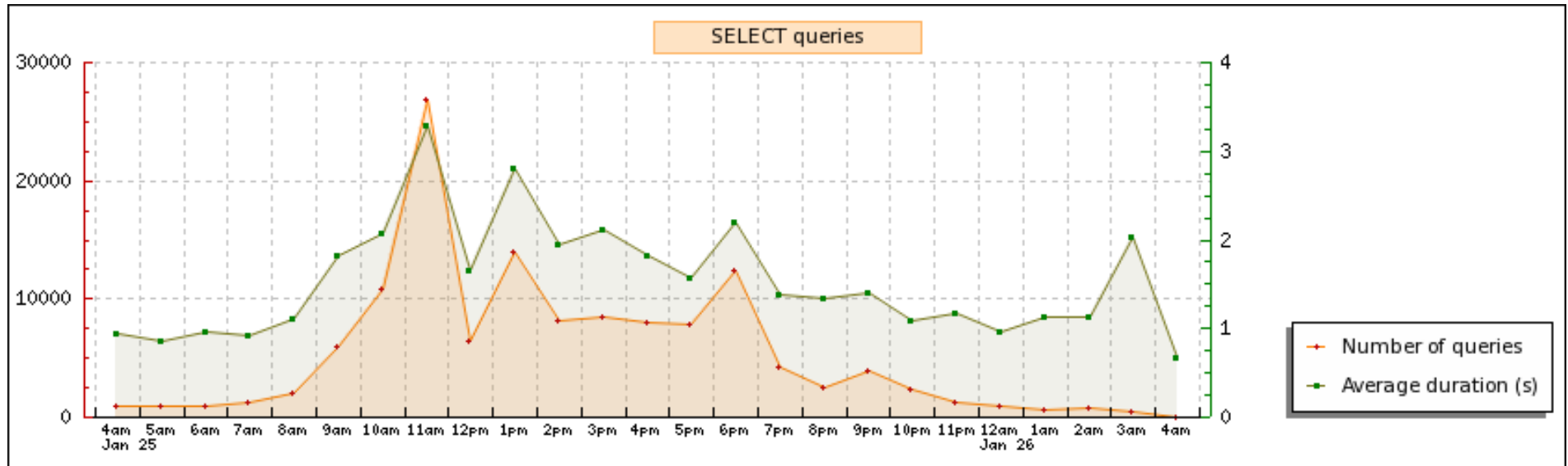
```
SET constraint_exclusion TO on;
```

- По хостам (по базам): pl/proxy

Средства PostgreSQL pgFouine: анализ логов



Средства PostgreSQL pgFouine: анализ логов



Оптимизация запросов

- Минимизация количества запросов (один «большой» SQL-запрос лучше серии «маленьких») – выигрыш до 80%
 - 2-6 мс на каждый запрос => несколько «лишних» секунд для 1000 строк!
- Объединение запросов в одну транзакцию – выигрыш до 50%
 - не всегда лучший выбор для web-приложений!
- Минимизация объёмов данных (включая промежуточные)
 - минимизация проекции (перечисление столбцов)
 - минимизация выборки (WHERE)
 - учёт мощности множеств (кардинальность и селективность каждой из таблиц)

Оптимизация запросов

- Минимизация медленных `count()`
- Минимизация количества операций `DELETE`
 - `UPDATE`
 - `TRUNCATE TABLE`
- Массивные (bulk) операции вместо группы row-level операций
 - **один `UPDATE` вместо цикла `UPDATE`-ов**
 - **`COPY` вместо группы `INSERT`-ов**
- Для массивных операций:
 - `ALTER TABLE ... DISABLE TRIGGER ALL`
 - `DROP INDEX ... / ... / CREATE INDEX ...`
- Подготовленные запросы (`PREPARE ...`)

Обработка запроса в PostgreSQL

- Parser (синтаксический анализатор)
- Planner (выбор оптимального пути)
- Executor (непосредственное выполнение)

SQL – декларативный язык. СУБД решает, как именно будет выполняться запрос.

EXPLAIN

- План запроса – дерево
- Узлы – действия
 - соединения (join)
 - сортировка
 - просмотр таблицы
- Выполнение происходит от листьев к корню
- Оценка «ширины» данных, количества строк и стоимости
- Это только *оценка*, реальную стоимость покажет EXPLAIN ANALYZE

Пример EXPLAIN

```
test=# explain select * from pg_proc order by proname;
```

```
QUERY PLAN
```

```
-----  
Sort   (cost=191.67..197.08 rows=2166 width=299)  
  Sort Key: proname  
    -> Seq Scan on pg_proc   (cost=0.00..71.66 rows=2166  
width=299)  
(3 rows)
```

EXPLAIN: width

```
test=# explain select * from pg_proc order by proname;  
                QUERY PLAN
```

```
-----  
Sort  (cost=191.67..197.08 rows=2166 width=299)  
  Sort Key: proname  
    -> Seq Scan on pg_proc  (cost=0.00..71.66 rows=2166  
width=299)  
    (3 rows)
```

Показывает примерное
среднее количество байт
в одной строке
результата

«Ширины» типов данных

int2	2
int4	4
int8	8
boolean	1
varchar(n) / char(n)	n + 4
timestamp / timestamptz	8

EXPLAIN: rows

```
test=# explain select * from pg_proc order by proname;  
                QUERY PLAN
```

```
-----  
Sort  (cost=191.67..197.08 rows=2166 width=299)  
  Sort Key: proname  
    -> Seq Scan on pg_proc  (cost=0.00..71.66 rows=2166  
width=299)  
(3 rows)
```

- Показывает примерное количество строк результата (в т.ч., промежуточного)
- Большое отклонение от реальности => необходим VACUUM & ANALYZE!
- Можно использовать для вывода *примерного* количества строк результата конечному пользователю

EXPLAIN: cost

```
test=# explain select * from pg_proc order by proname;  
                QUERY PLAN
```

```
-----  
Sort  (cost=191.67..197.08 rows=2166 width=299)  
  Sort Key: proname  
    -> Seq Scan on pg_proc  (cost=0.00..71.66 rows=2166  
width=299)  
(3 rows)
```

- *Абстрактная* величина (ввода-вывода, CPU)
- Именно это даёт возможность планируемому выбрать один план из нескольких
- 2 значения: startup & total
- Это всего лишь *оценка!*

EXPLAIN ANALYZE

```
test=# explain analyze select * from pg_proc order by  
proname;
```

QUERY PLAN

```
-----  
Sort (cost=191.67..197.08 rows=2166 width=299) (actual  
time=30.161..34.990 rows=2166 loops=1)  
  Sort Key: proname  
  Sort Method: quicksort Memory: 639kB  
    -> Seq Scan on pg_proc (cost=0.00..71.66 rows=2166  
width=299) (actual time=2.094..20.232 rows=2166 loops=1)  
  Total runtime: 40.192 ms  
(5 rows)
```

- **Фактическая информация**
- **Время в миллисекундах**
- **Добавляется общее время запроса**
- **loops – количество «проходов» (необходимо умножить время на loops, чтобы получить общее время)**

EXPLAIN ANALYZE

- Способы просмотра таблицы
 - Seq Scan
 - Index Scan
- Способы подготовки данных
 - Sort
 - Hash
- Способы соединения (join)
 - Nested Loop
 - Merge Join
 - Hash Join

Общие рекомендации

- Сбор статистики, мониторинг, анализ логов (pgFouine)
- Итерационное выявление медленных запросов
- Иногда стоит перестроить запрос, а не создать N новых индексов
- Следить за кардинальностью и селективностью промежуточных результатов (`SELECT * FROM a, b`)
- Потенциальные источники проблем: `LEFT JOIN`, `count()`, `UNION` вместо `UNION ALL`, `DISTINCT`, `WHERE ... IN (...)`, соединение 100 таблиц, неожиданное «выпрямление» запросов с подзапросами
- Не забывать об `ANALYZE` после массивных изменений!

Общие рекомендации

- Ещё раз: при решении проблемы, убедитесь в том, что вам не нужны `VACUUM & ANALYZE`
- Запускайте `EXPLAIN ANALYZE` не менее двух раз (не забываем про кэш)
- Читайте план снизу вверх, ищите, где начинаются замедления и/или ошибки планнера
- При возможности, используйте *реальные* данные в тестировании (Slony?) и оборудование, приближенное к production
- Обращайтесь за помощью: `pgsql-performance`, IRC, `sql.ru`, коммерческая поддержка

Управление планнером

- `enable_seqscan`
- `enable_indexscan`
- `enable_sort`
- `enable_nestloop`
- `enable_hashjoin`
- `enable_mergejoin`
- `enable_hashagg`

Как правило, помогает при разработке, но вредит на «боевых» серверах (другие объёмы данных => другая статистика)

Индексы в PostgreSQL

- B-tree
- Hash
- R-tree
- GiST (обобщенное поисковое дерево)
- GIN (обратный индекс)

Индексы в PostgreSQL

- B-tree
- Hash
- R-tree – теперь тоже GiST
- GiST (обобщенное поисковое дерево)
- GIN (обратный индекс)

Фёдор Сигаев, Олег Бартунов

Индексы в PostgreSQL

- Не забываем о таких «приятных» вещах как:
 - Частичные индексы (`CREATE INDEX ... WHERE ...`)
 - Функциональные индексы (`CREATE INDEX ... USING btree (myfunc (a))`)
 - уникальные функциональные индексы
 - индексирование данных экзотических типов (XML, array, ...)
 - Многоколоночные индексы
 - следим за правильным порядком столбцов
 - избегаем ненужных одноколоночных индексов
 - GIN-индексы для массивов
 - `CLUSTER table USING indexname`

Индексы в PostgreSQL

- **Настройка и обслуживание**

- CREATE INDEX ... FILLFACTOR

- CREATE INDEX ... CONCURRENT

- **Необходимость в перестроении индексов (read/fetch ratio):**

```
select
  indexrelname, idx_tup_read, idx_tup_fetch, case
  when idx_tup_fetch = 0 then 100 else
  idx_tup_read / idx_tup_fetch end as ratio from
  pg_stat_user_indexes order by ratio desc;
```

Специализированные типы данных

- intarray
- hstore
- ltree
- tsvector, tsquery
- pg_trgm
- GIS: point/box/..., PostGIS, pgSphere, Q3C
- XML

Специализированные типы данных

- intarray GiST!
- hstore
- ltree
- tsvector, tsquery
- pg_trgm
- GIS: point/box/..., PostGIS, pgSphere, Q3C
- XML

GiST или GIN?

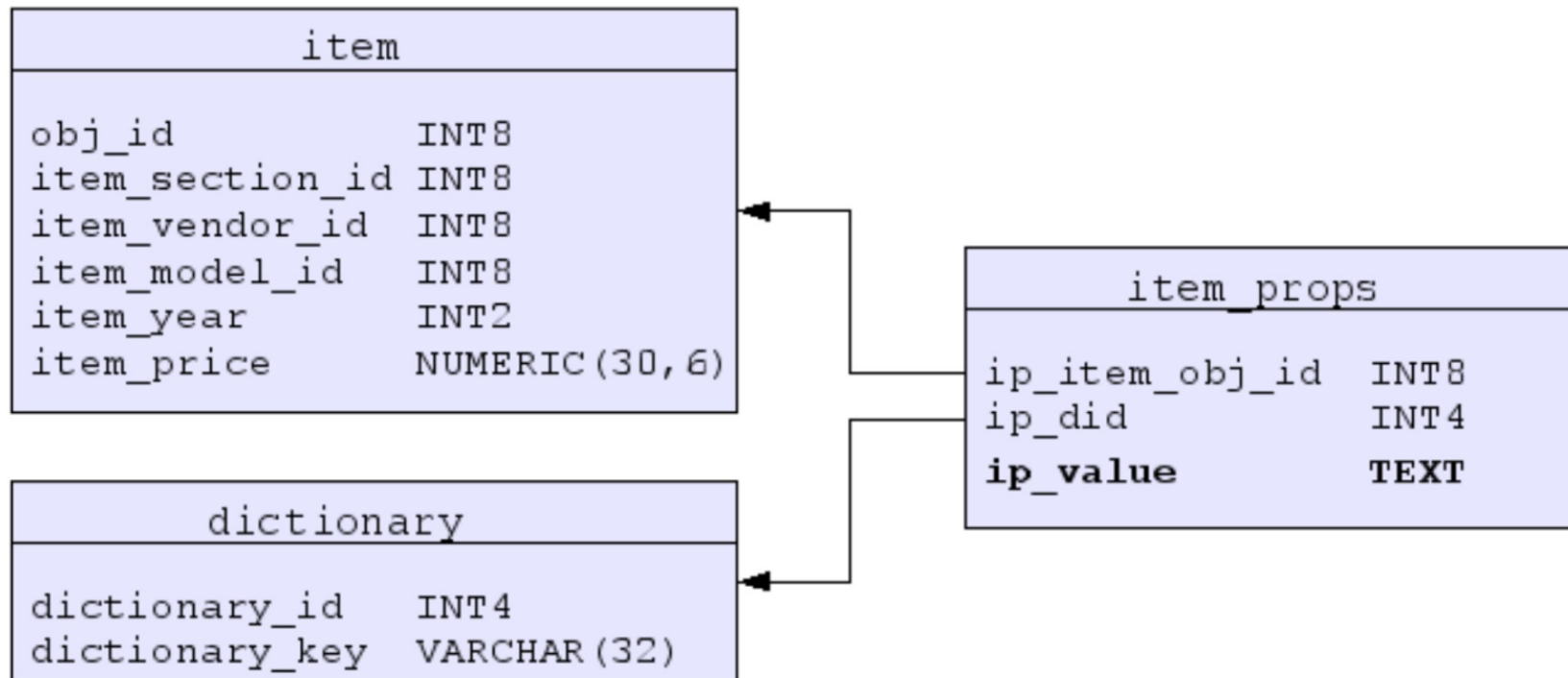
GIN		GiST	
Создание (bulk) (3x)		+	-
Поиск по индексу + (0.3x)		-	
Обновление индекса	+++		- (10x)
Размер индекса (2-3x)		+	-
Масштабирование +++		-	

intarray, hstore (а в будущем и XML) – когда использовать?

item	
obj_id	INT8
item_section_id	INT8
item_vendor_id	INT8
item_model_id	INT8
item_year	INT2
item_price	NUMERIC(30,6)
item_prop1	INT4
item_prop2	INT4
item_prop3	INT4
item_prop4	INT4
...	
item_prop21	TEXT
item_prop22	TEXT
item_prop23	TEXT
...	
item_prop41	BOOLEAN
...	

Каталог разнородных данных: 1-й способ реализации («широкая таблица»)

intarray, hstore (а в будущем и XML) – когда использовать?



Каталог разнородных данных: 2-й способ реализации
(Entity-Attribute-Value, EAV)

intarray, hstore (а в будущем и XML) –
когда использовать?

item	
obj_id	INT8
item_section_id	INT8
item_vendor_id	INT8
item_model_id	INT8
item_year	INT2
item_price	NUMERIC (30, 6)
item_props	XML

Каталог разнородных данных: 3-й способ реализации (hstore)

Каталог разнородных данных: сравнение трёх способов

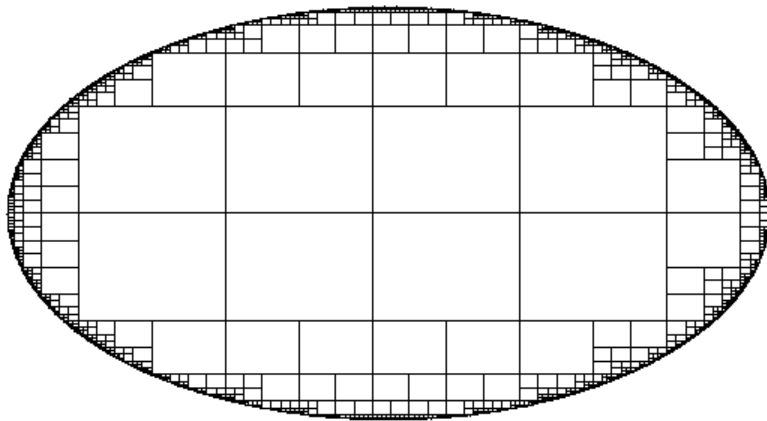
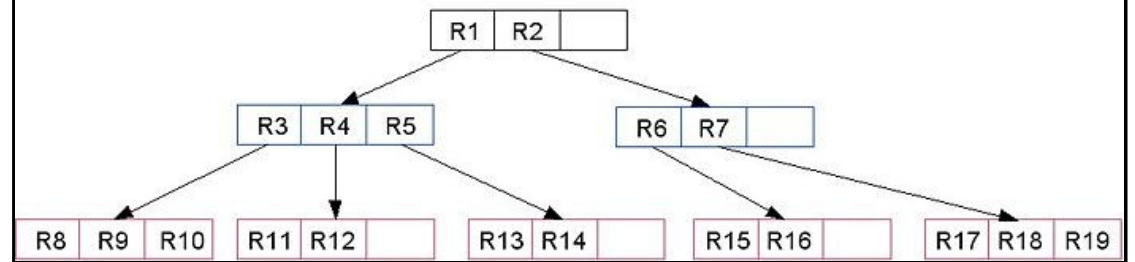
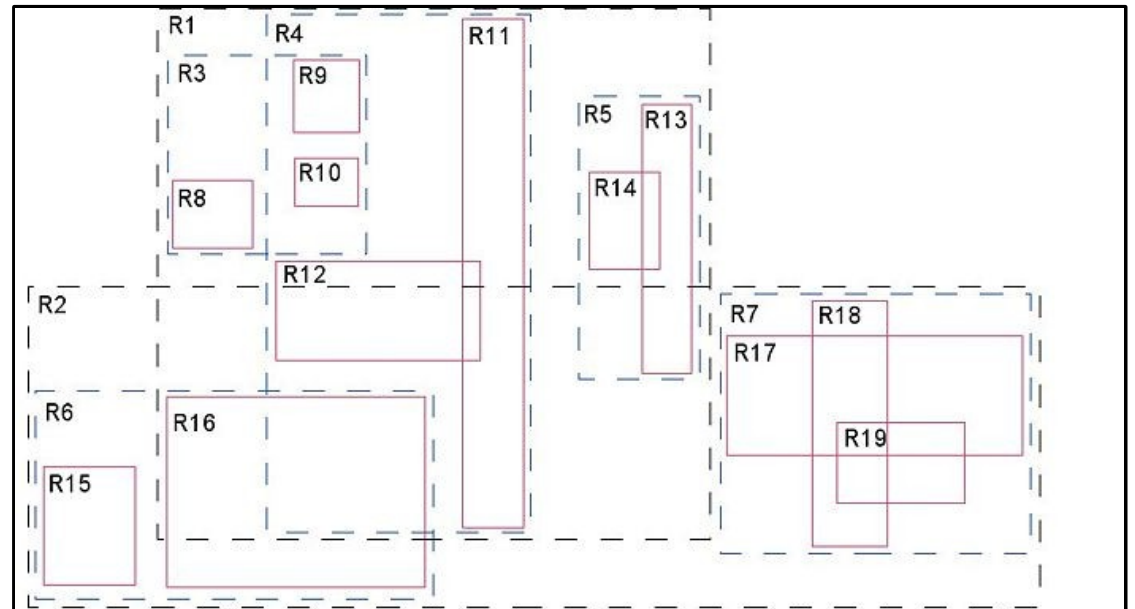
на примере данных <http://domnakarte.ru>, 700 тыс. объектов недвижимости;
ноутбук Pentium-M 1.86GHz, 1GB;
shared_buffers = 250MB
work_mem = 20MB

EAV	hstore (GiST)	«широкая таблица»	
Общий размер таблиц, MB 907		337	1488
Общий размер индексов, MB 48		160+	1255
Простая выборка по условию 410 / ~40000*	, мс	120 / 30000*	690 / ~70000*

*) Что делать? Рецепт для web-приложений: «Поднимаем» файлы в файловый кэш ОС:
dd ... of=/dev/null

Типы данных для GIS

- Встроенные, R-tree (GiST)
- PostGIS (GiST)
- pgSphere (GiST)
- Q3C (btree)



Q3C segmentation

R-tree

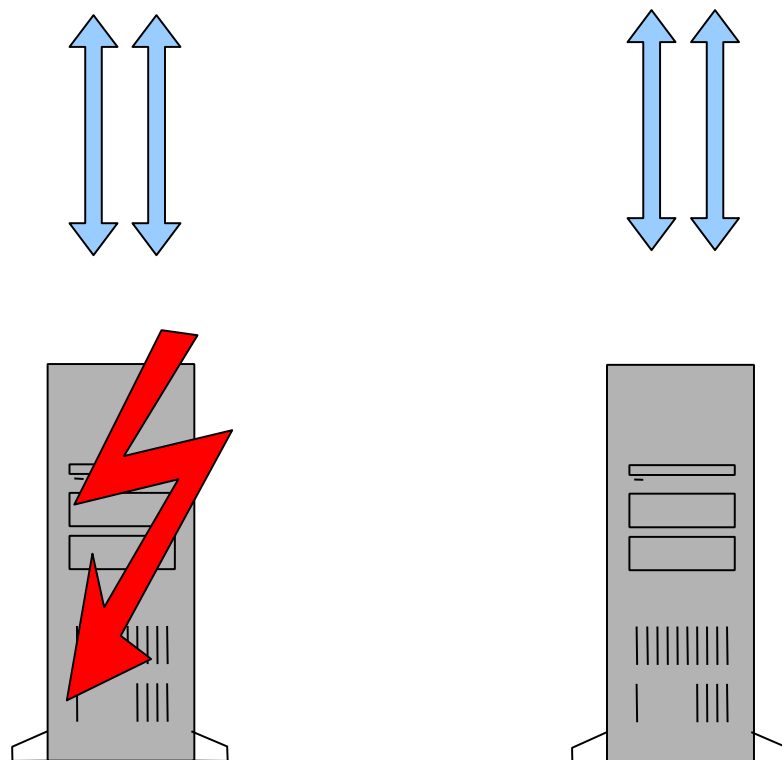
Кэширование и СУБД

- Кэширование на уровне СУБД (и ОС):
 - файловый кэш ОС (!)
 - `shared_buffers`
 - кэширование планов запросов
 - не забываем указывать признак `IMMUTABLE/STABLE/VOLATILE` у функций
 - минимизация количества динамического SQL
- Кэширование на уровне приложения:
 - кэширование результатов запросов (файлы, `memcached`, `shared memory`)
 - основные справочники
 - объекты
 - метаданные базы данных
 - кэширование страниц
 - на сервере (`memcached`, `proxy-server`)
 - на стороне клиента (правильные HTTP-заголовки)

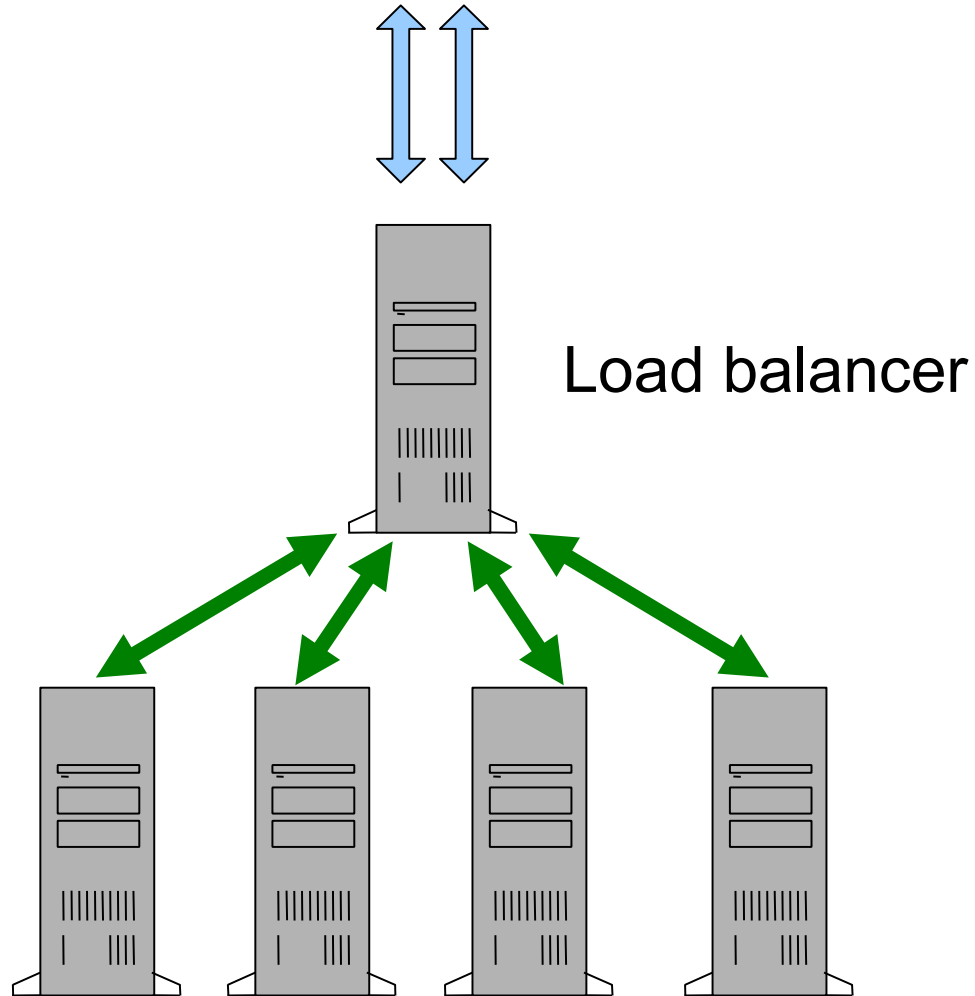
Репликация

- Отказоустойчивость
- Балансировка нагрузки
- Масштабирование

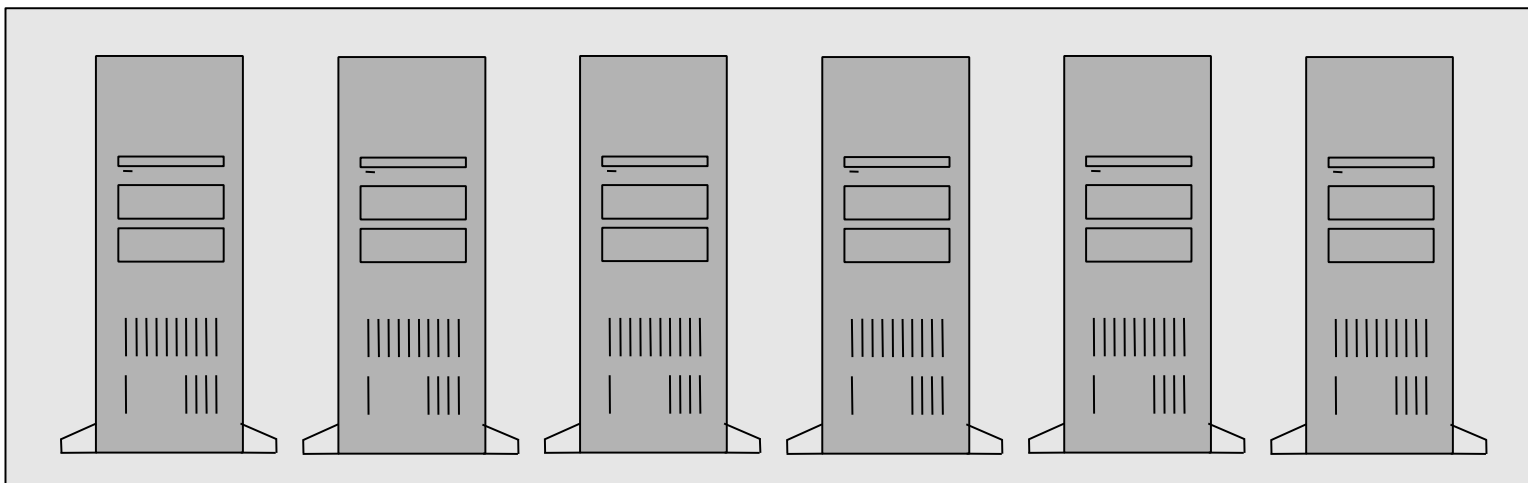
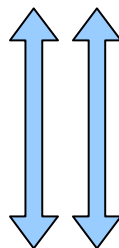
Отказоустойчивость (failover)



Балансировка нагрузки



Масштабирование (scale-out, а не scale-up)

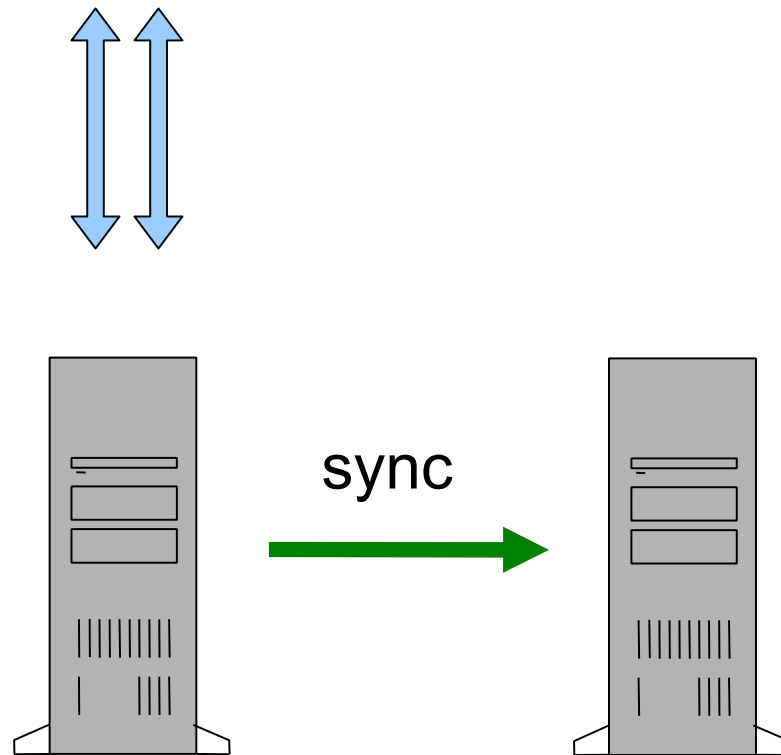


Репликация

- Master/Slave – synchronous
- Master/Slave – **asynchronous**
- Multi-Master – synchronous
- Multi-Master – **asynchronous**

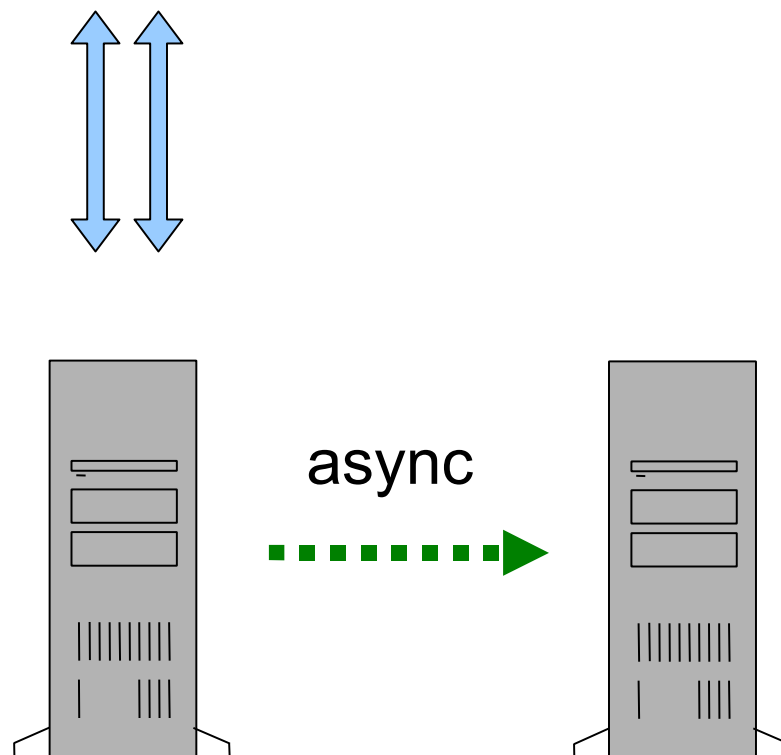
Репликация

Master/Slave – synchronous



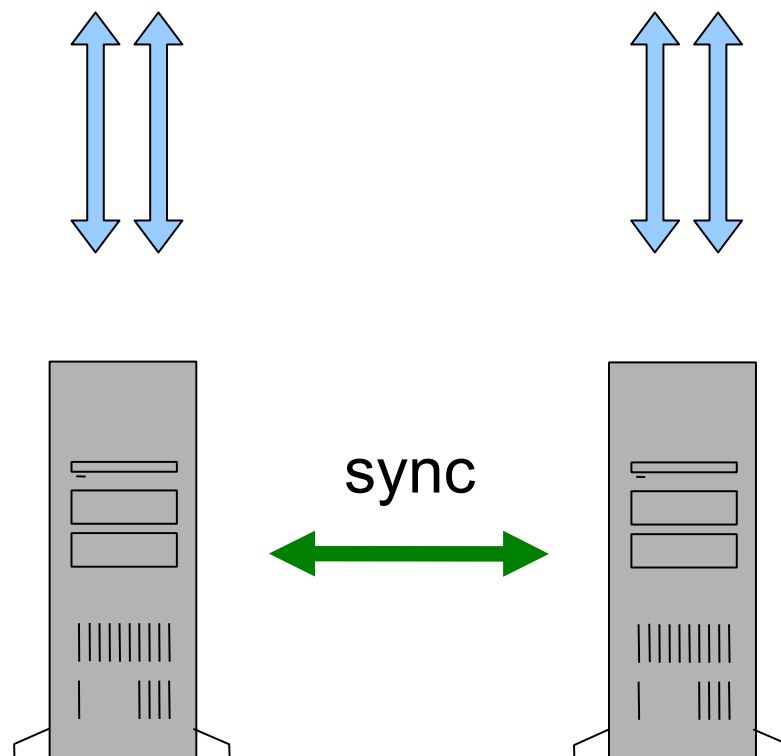
Репликация

Master/Slave – asynchronous



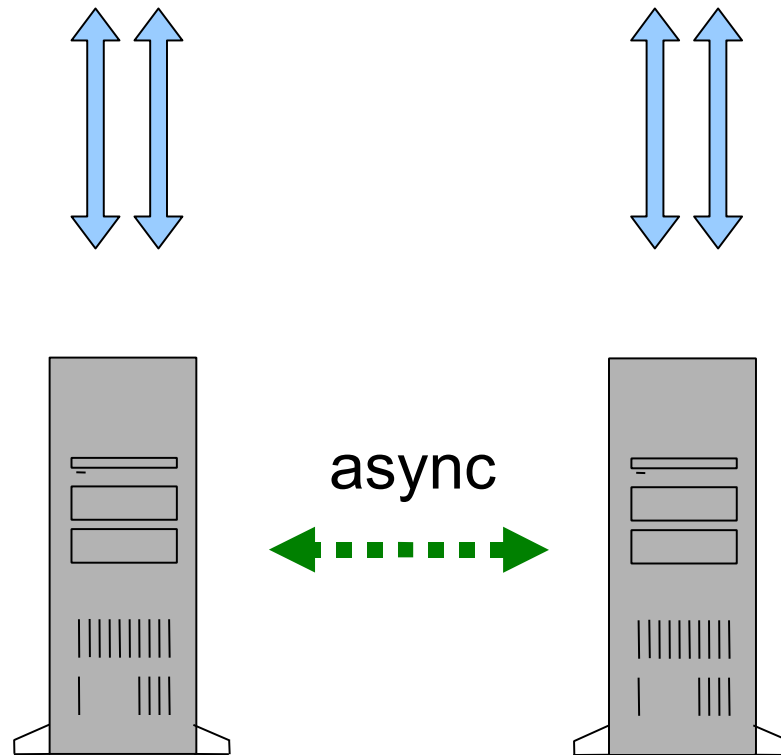
Репликация

Multi-Master – synchronous



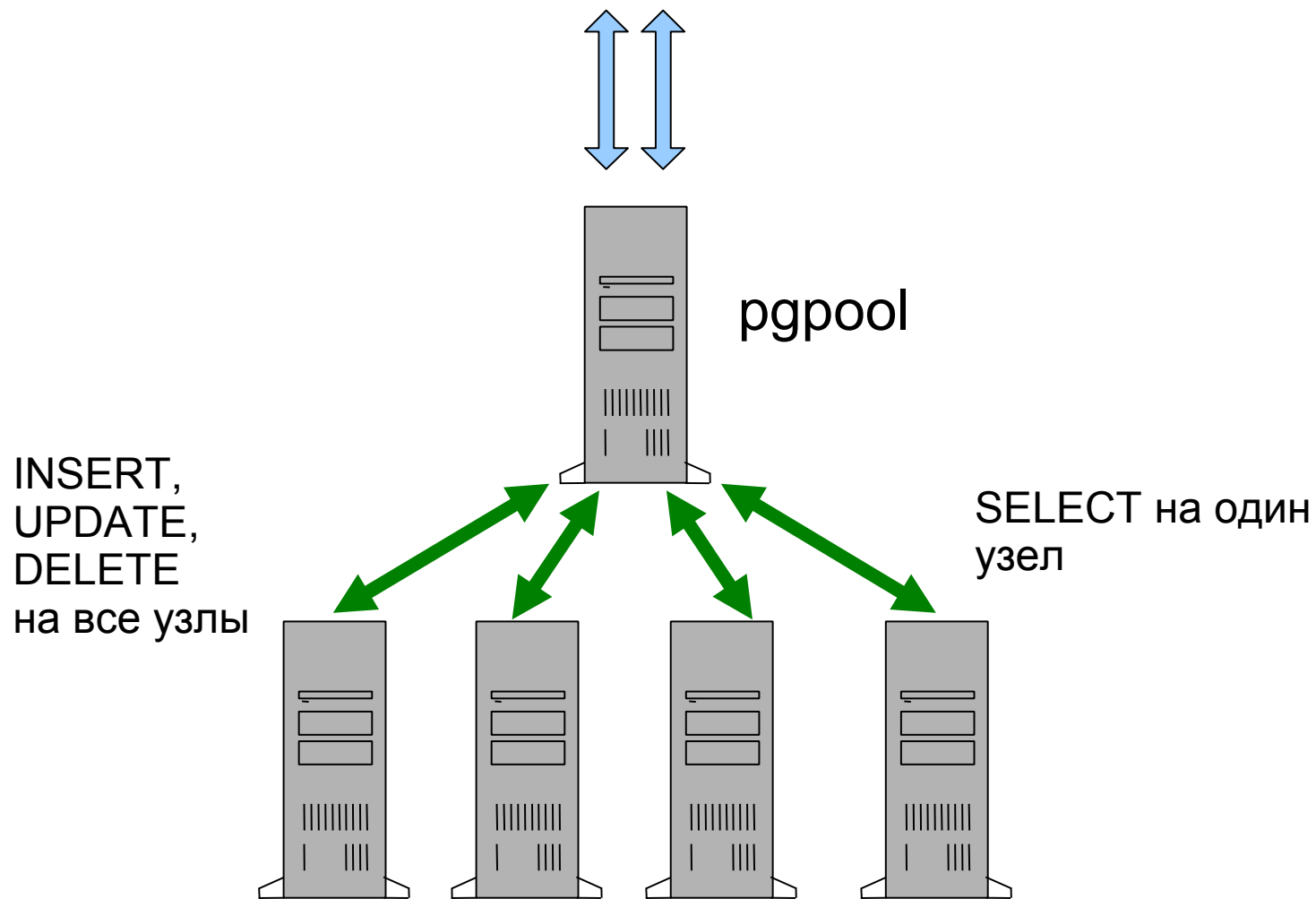
Репликация

Multi-Master – asynchronous



With conflicts resolution!

Pgpool-II



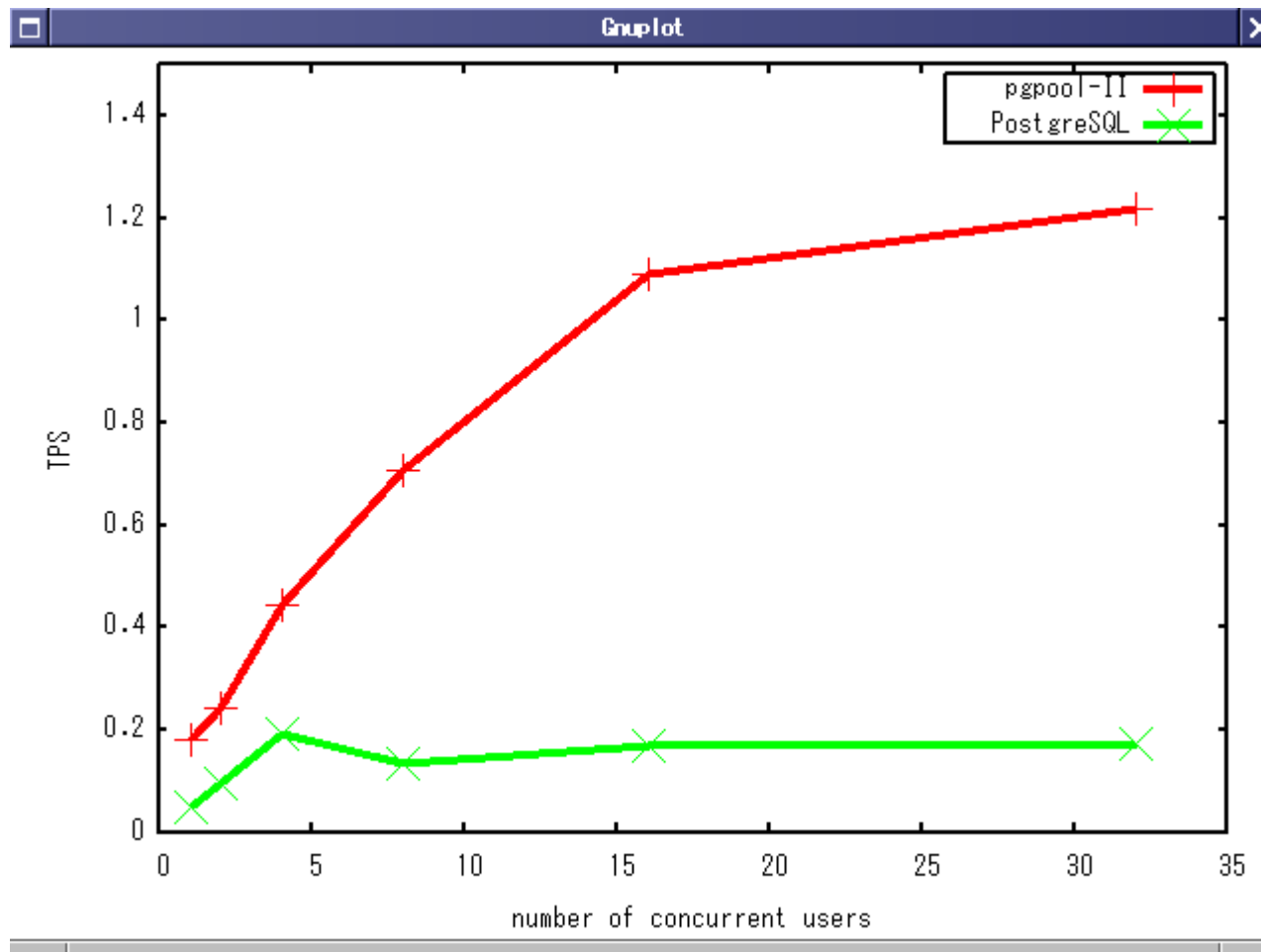
Рdpool-II

- Менеджер соединений (connection pooling + кеш)
- Failover (обнаружение отказа и переключение на резервный сервер)
- Репликация (синхронная, мульти-мастер)

Pgpool-II

- Балансировка нагрузки
- Master/Slave режим -- работа вместе с другой системой репликации: Slony-I, WAL, ...
- Параллельное выполнение запросов
- Нет ограничений на число узлов кластера (как было в pgpool-I)
- Кэш запросов
- GUI
- Линейная масштабируемость на «dblink»-запросах

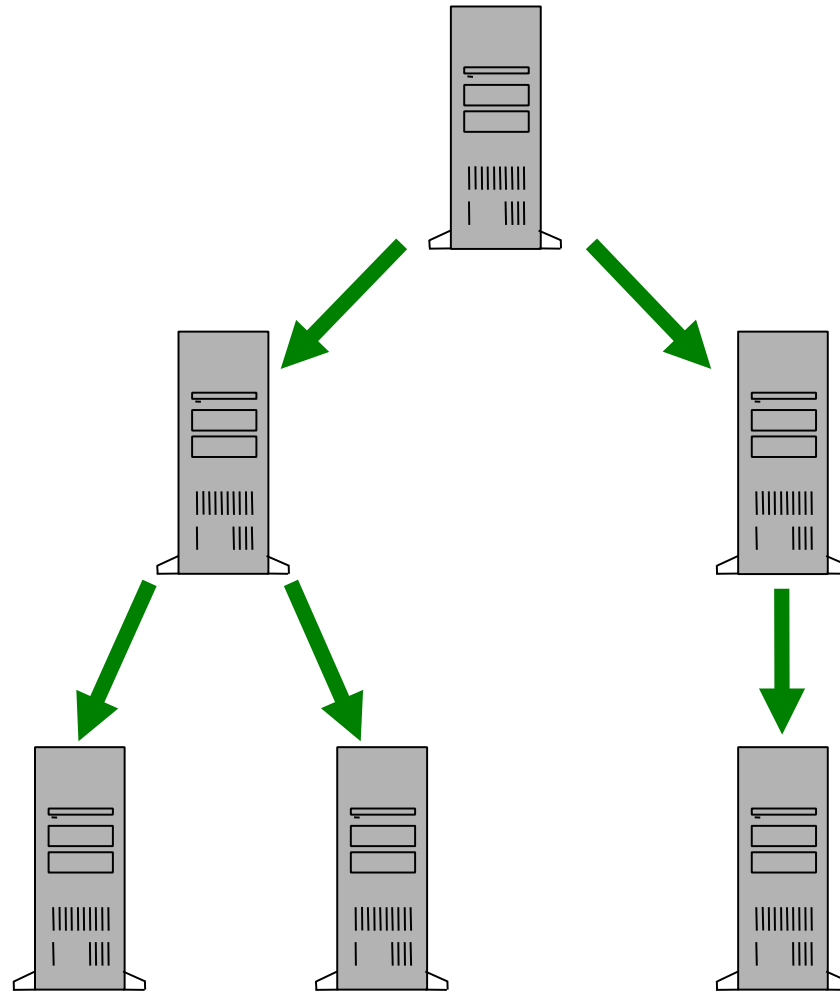
Pgpool-II



Slony-I

- Master to multiple slaves
- Самая известная репликация для PostgreSQL
- Асинхронная система
- Десятки слейвов
- Каскады реплицируемых серверов

Slony-I



PgCluster

- Синхронный мультимастер
- Балансировка нагрузки
- Высокодоступная (HA) система
- Cluster DBs, Load Balancer, Replicator
- Для ленивых: решение от Cybertec, Austria

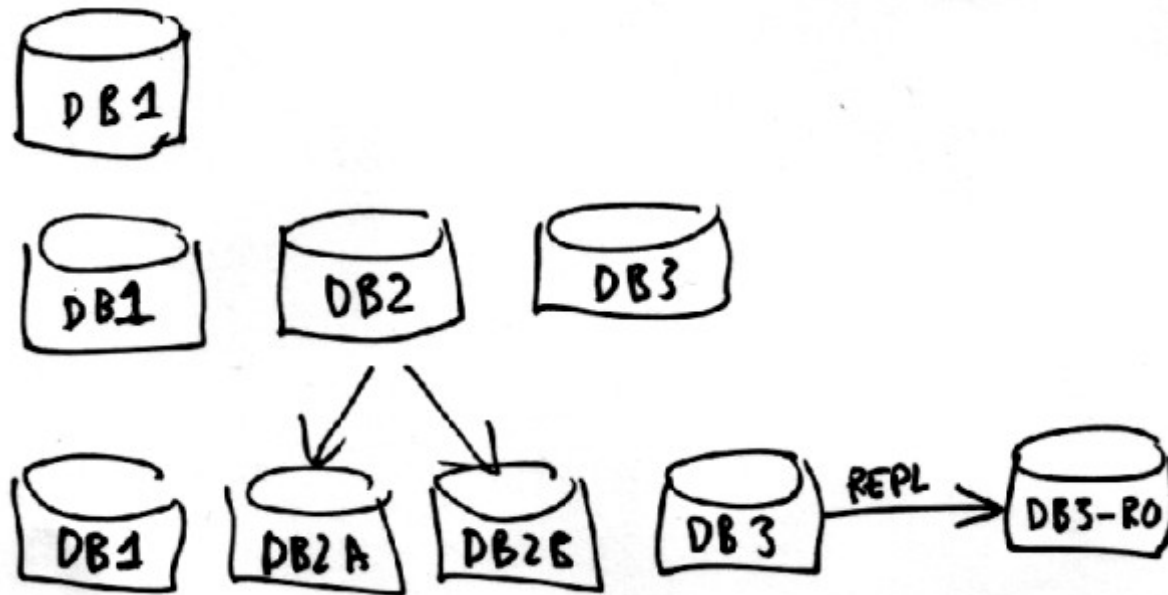
WAL shipping

- Асинхронная система
- Встроена в PostgreSQL
- Hot backup
- Нельзя читать со слейвов
- **walmgr.py**
- **Скоро read queries!**

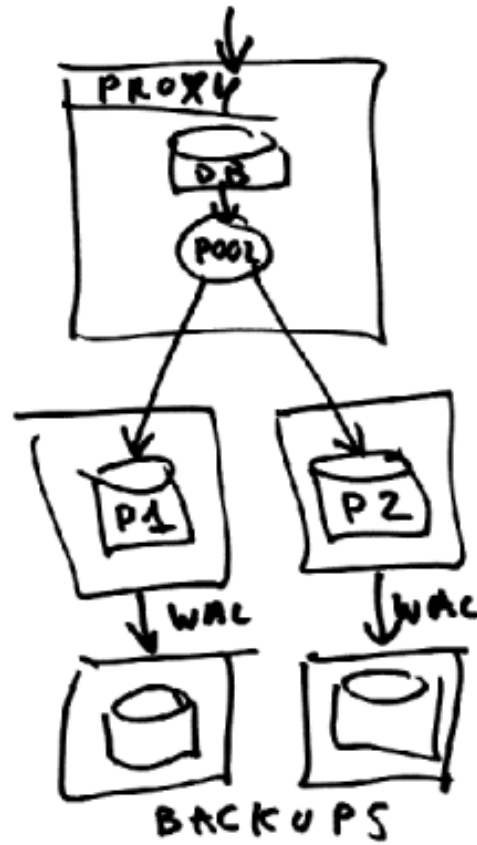
Skype

- PL/Proxy: проксирование запросов
- PgBouncer: легкий менеджер соединений
- SkyTools: управление кластером

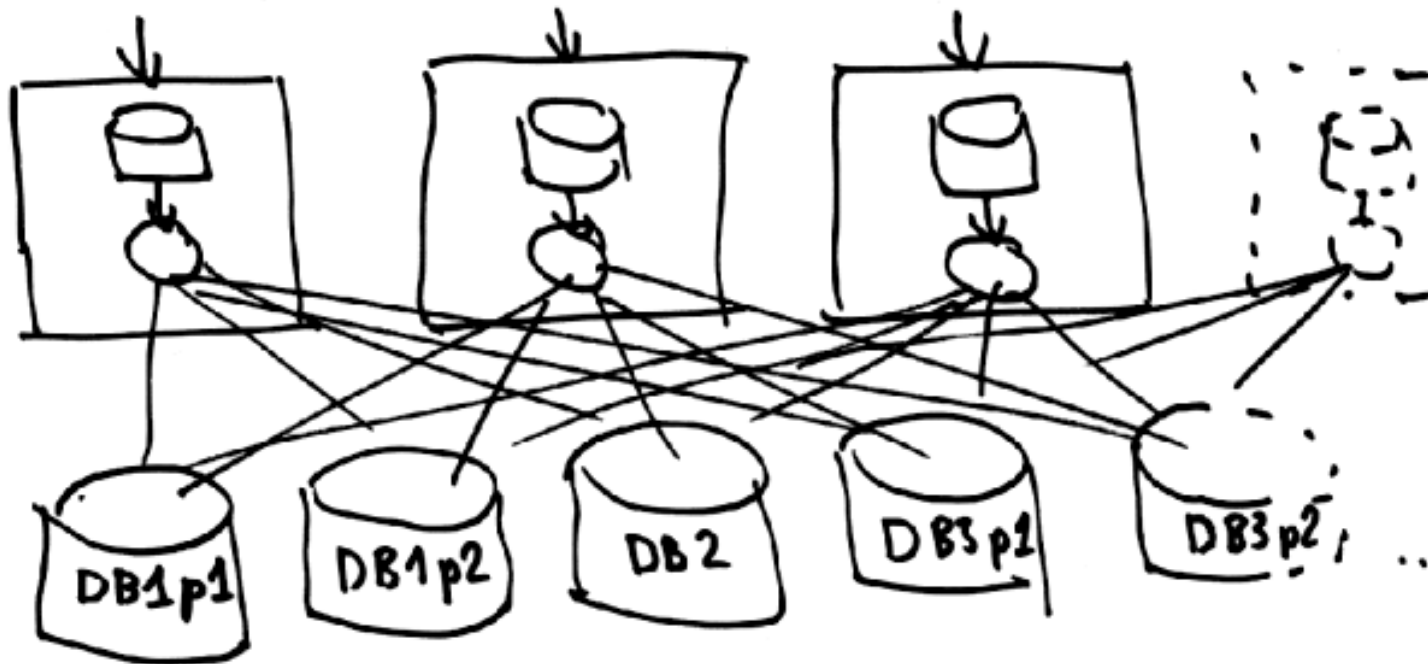
Skype



Skype



Skype



Правильный выбор железа

- Диски
- RAID-контроллер
- Схема RAID
- CPU

Правильный выбор железа

- Диски
 - SCSI лучше, чем SATA
 - Чем больше дисков, тем лучше: 12 SATA лучше 4 SCSI
 - Чем быстрее диски, тем лучше: 15K RPM = 250 оборотов в сек = 250 TPS

Правильный выбор железа

- RAID контроллер
 - **Очень важный элемент**
 - Только не Adaptec!
 - Уж лучше software RAID
 - Хорошие отзывы о MegaRAID, 3ware
 - Включайте write cache только если есть WBC
 - Чем больше памяти на контроллере, тем лучше

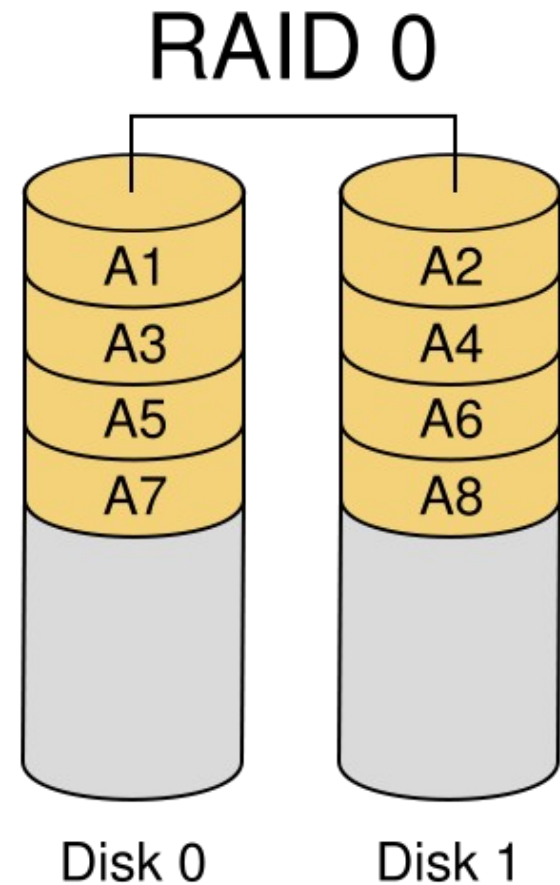
Правильный выбор железа

- RAID 0
- RAID 1
- RAID 5
- RAID 6
- RAID 0+1
- RAID 1+0



Правильный выбор железа

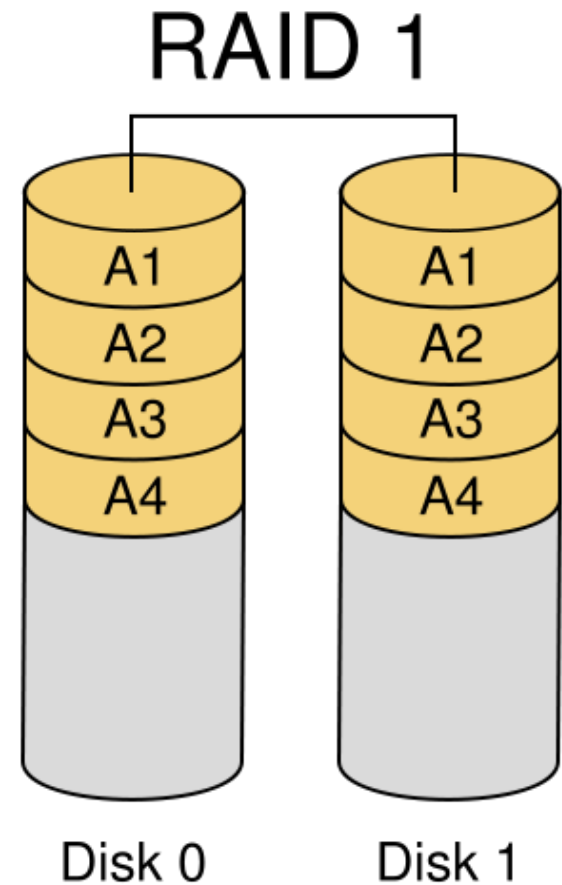
- RAID 0
 - Stripe
 - 2 диска минимум
 - Нет отказоустойчивости
 - Отличный I/O
 - Простое устройство и обслуживание



Правильный выбор железа

- RAID 1

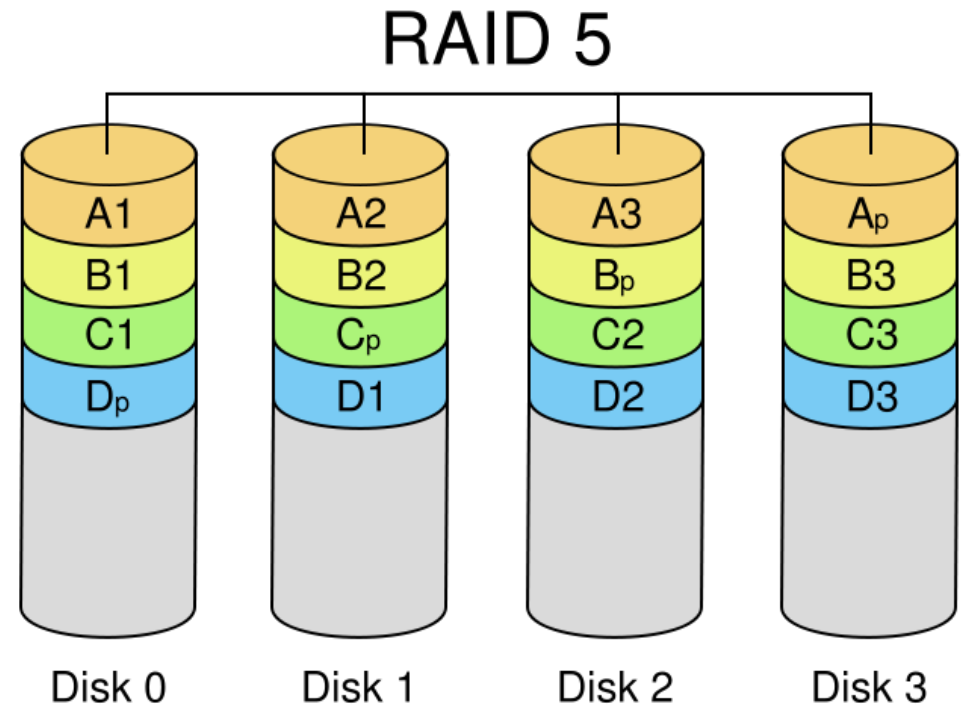
- Mirror
- 2 диска минимум
- Отказоустойчивость
(1 диск)
- Выигрыш в скорости чтения, небольшая деградация в скорости записи



Правильный выбор железа

- RAID 5

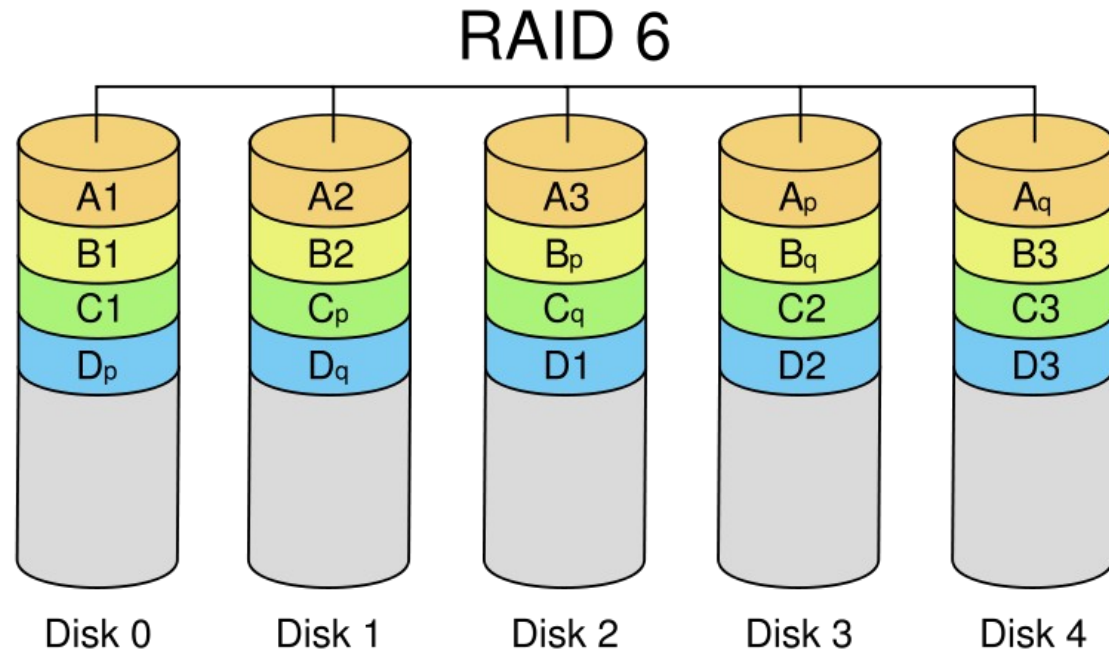
- Striped set with distributed parity
- 3 диска минимум
- Отказоустойчивость (1 диск)
- Медленная запись
- Чтение чуть медленнее RAID 0



Правильный выбор железа

- RAID 6

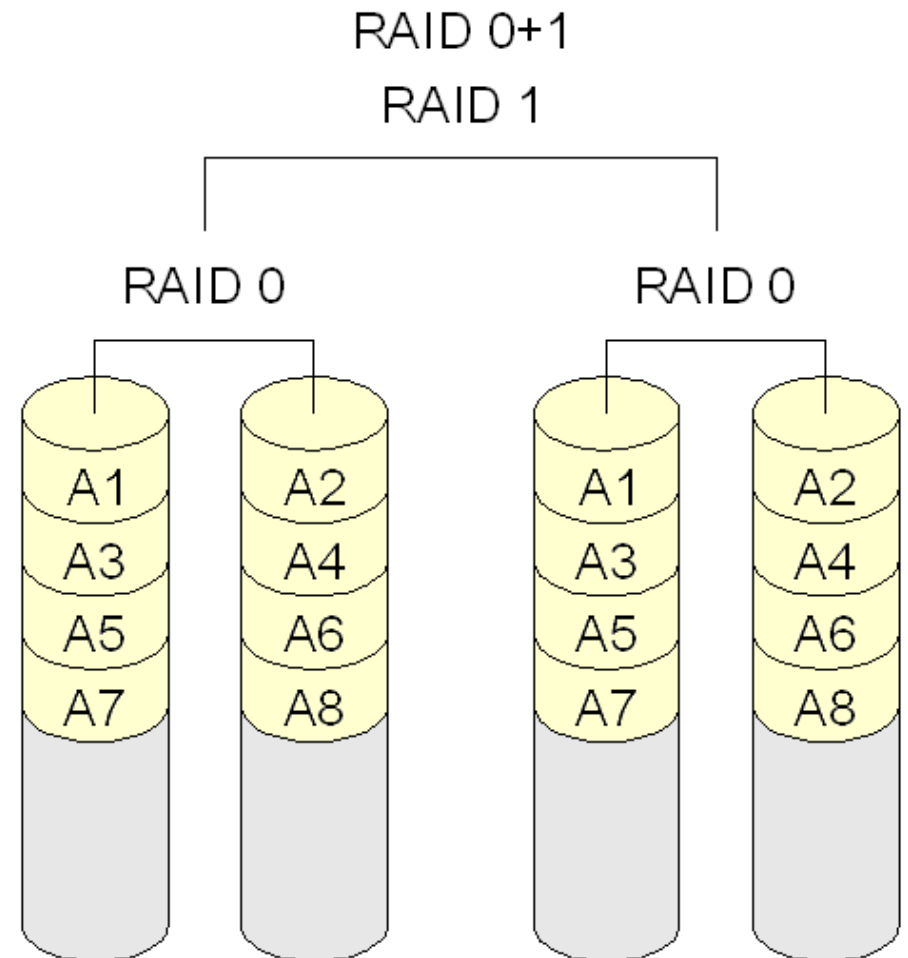
- Striped set with Dual distributed parity
- 4 диска минимум
- Отказоустойчивость (2 диска)
- Медленная запись
- Чтение чуть медленнее RAID 0 на том же количестве дисков



Правильный выбор железа

- RAID 0+1

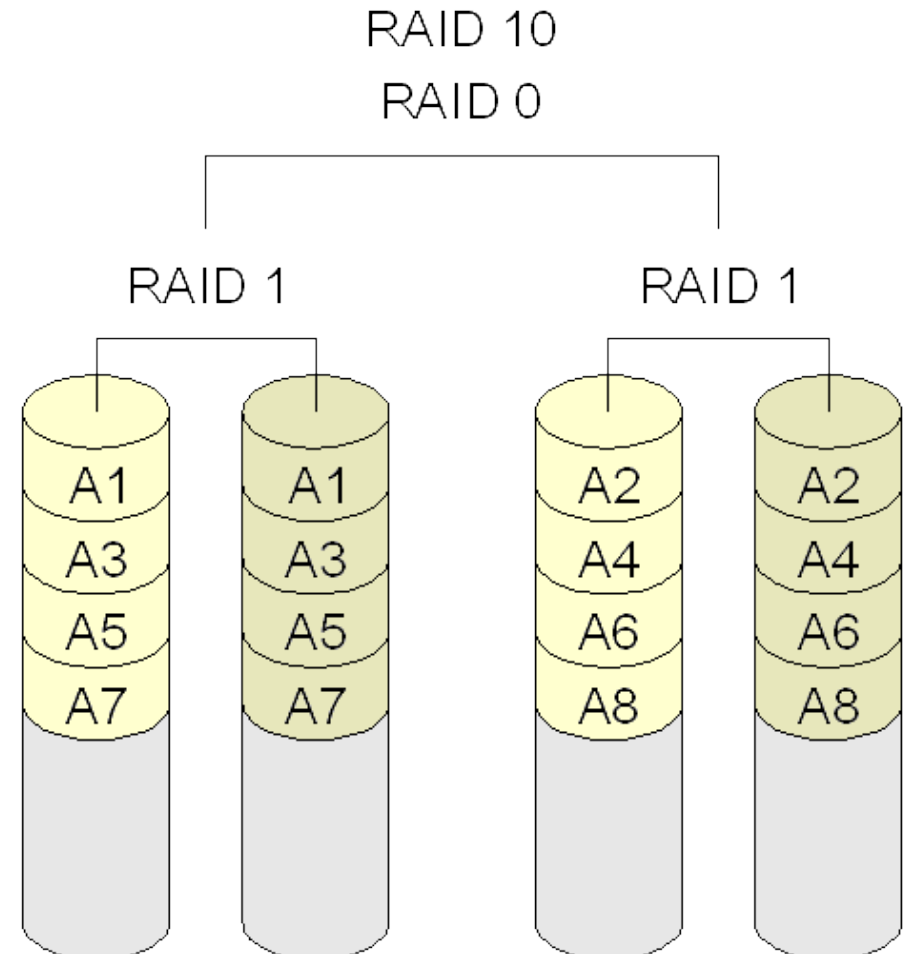
- Отказоустойчивость (выдерживает отказ только 1 диска)
- Требует полного rebuild-а при отказе диска



Правильный выбор железа

- RAID 1+0

- Отказоустойчивость (n*RAID1 дисков может отказать)
- При отказе диска – только rebuild его зеркала
- Оптимальная конфигурация для 4 дисков



Правильный выбор железа

- RAID 0
- RAID 1
- RAID 5
- RAID 6
- RAID 0+1
- RAID 1+0



Правильный выбор железа

- **RAID 0 (2 диска)**
- **RAID 1 (2 диска)**
- **RAID 5 (≥ 6 дисков)**
- RAID 6
- RAID 0+1
- **RAID 1+0 (4 диска)**



Правильный выбор железа

- Всегда тестируйте дисковую подсистему
 - `time "dd if=/dev/zero of=bigfile bs=8k count=RAM*2 && sync"`
 - `time dd if=bigfile of=/dev/null bs=8k`
 - `bonnie++` (de-facto стандарт дисковых тестов)

Обычный 7200 RPM диск:

- 56MB/s чтение
- 42MB/s запись
- Линейный рост по числу дисков в RAID 0

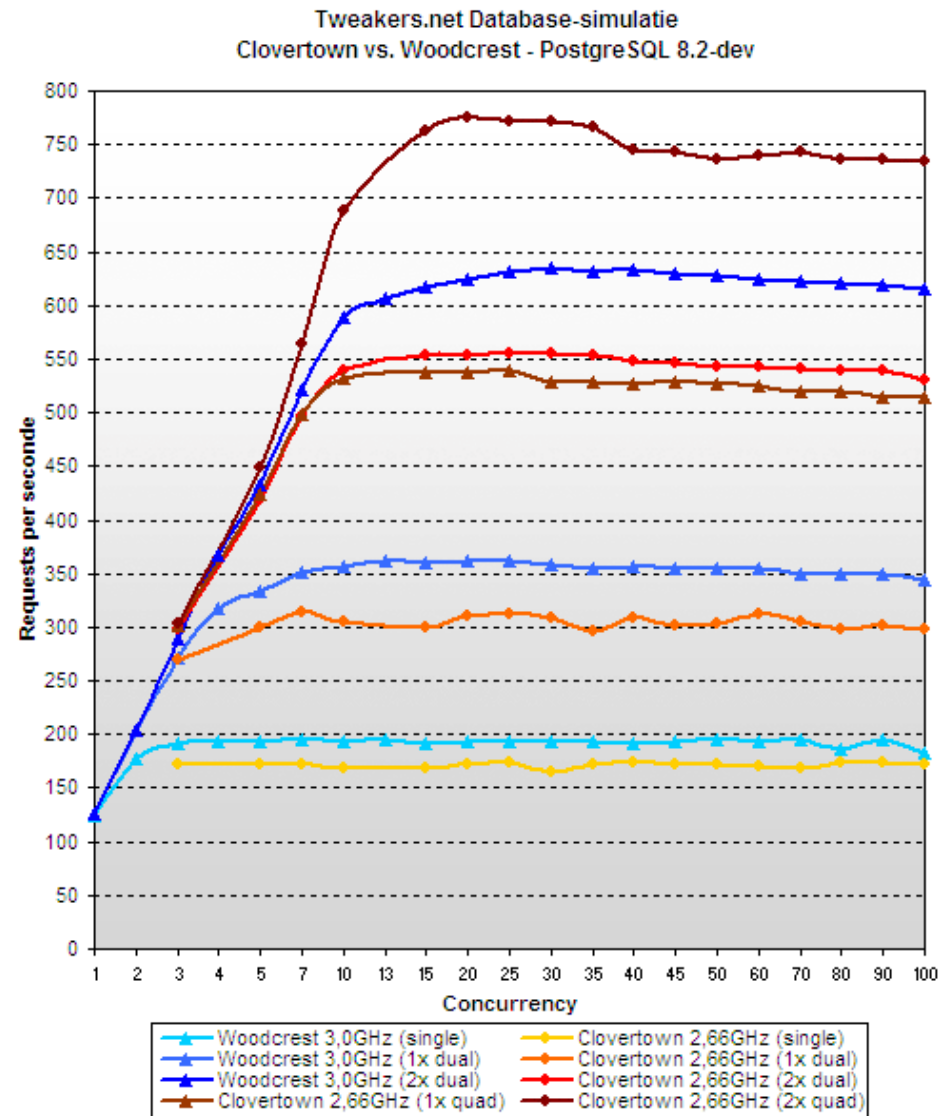
Настройки файловой системы

- Ext3 & AnyOtherJournalingFS
 - `noatime`: не обновлять `last access timestamp`
 - `writeback` на диске с `WAL`: только целостность метаданных, нет коммита самих данных в журнал
- OtherFS
 - Журналируйте в разумных масштабах
 - Компрессия rocks!
 - ZFS rocks!

Правильный выбор железа

- CPU

- Чем больше ядер, тем лучше (Linux)
- Opteron-ы считаются лучше Xeon-ов

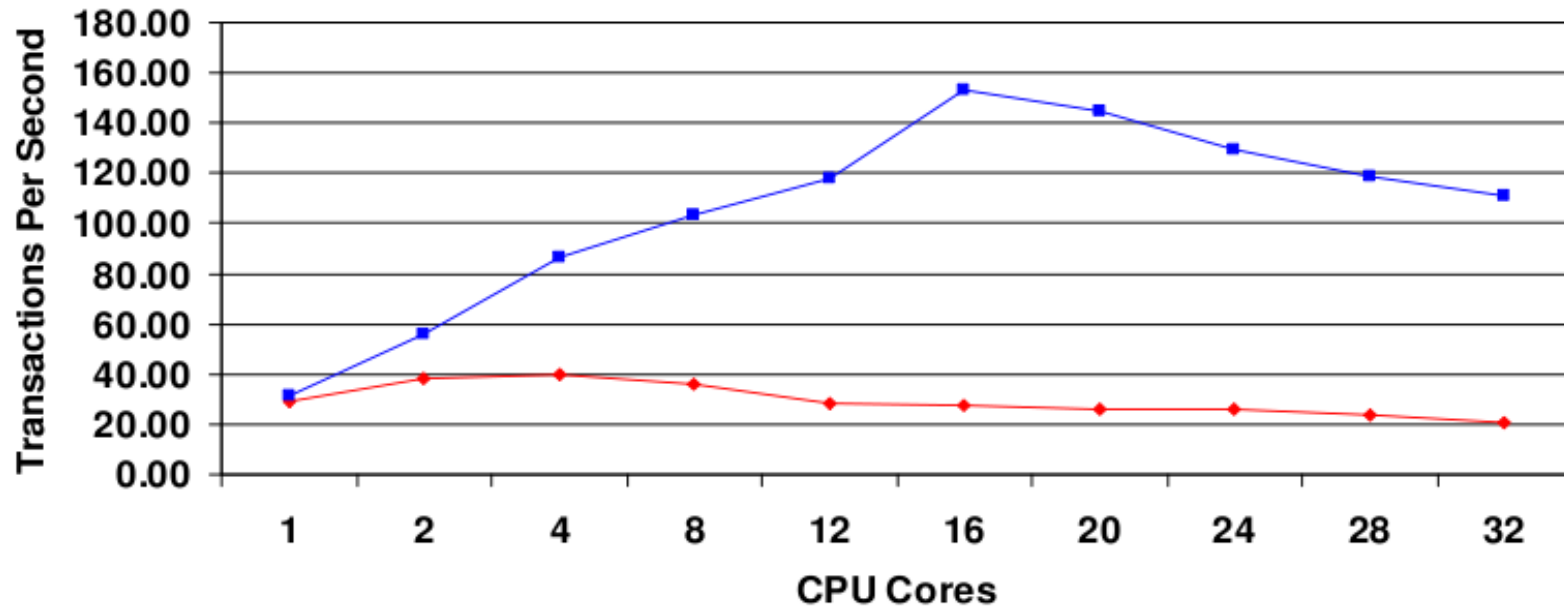


Правильный выбор железа

- CPU

Scaling – Results (2006)

Postgres 8.0.7 vs Postgres 8.1.2

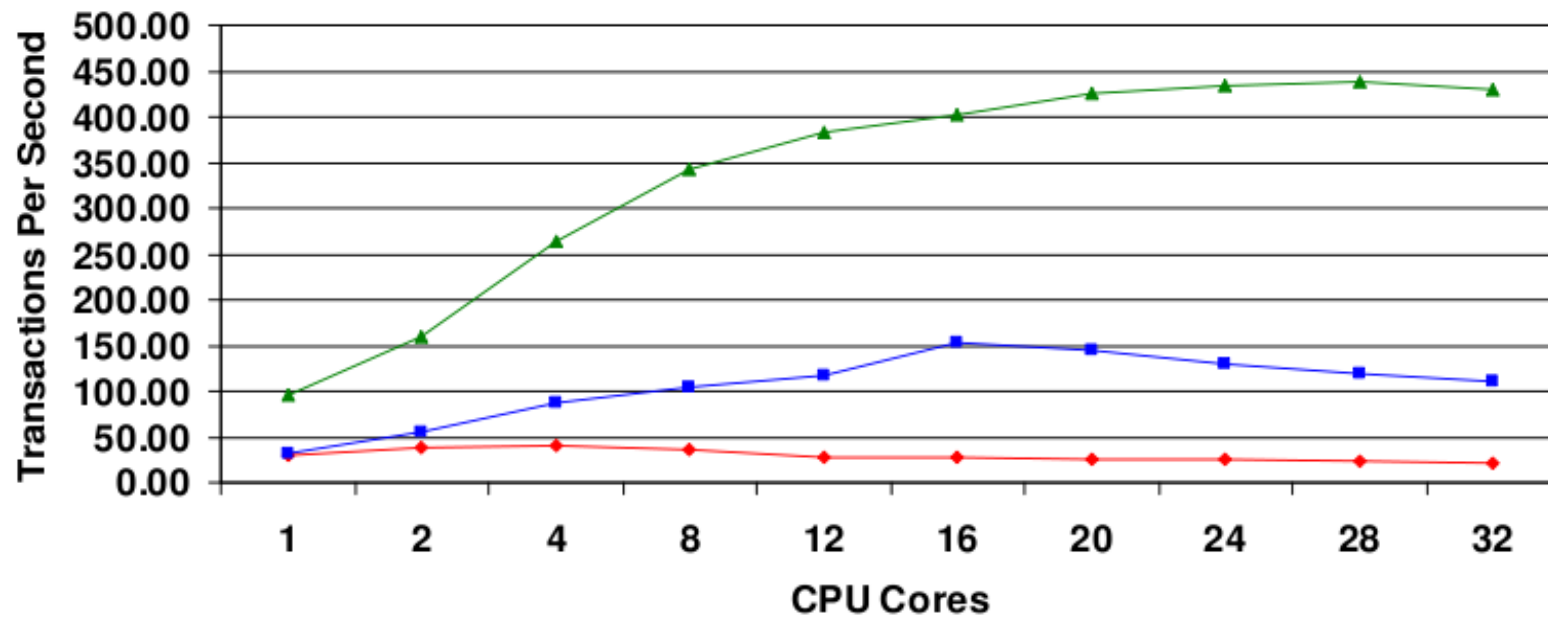


Правильный выбор железа

- CPU

Scaling – Results (2007)

Postgres 8.0.7/8.1.2 vs Postgres 8.2.4



Credits

- Josh Berkus
- Robert Treat
- Greg Smith
- Wikipedia
- Tweakers.net

Спасибо за внимание!