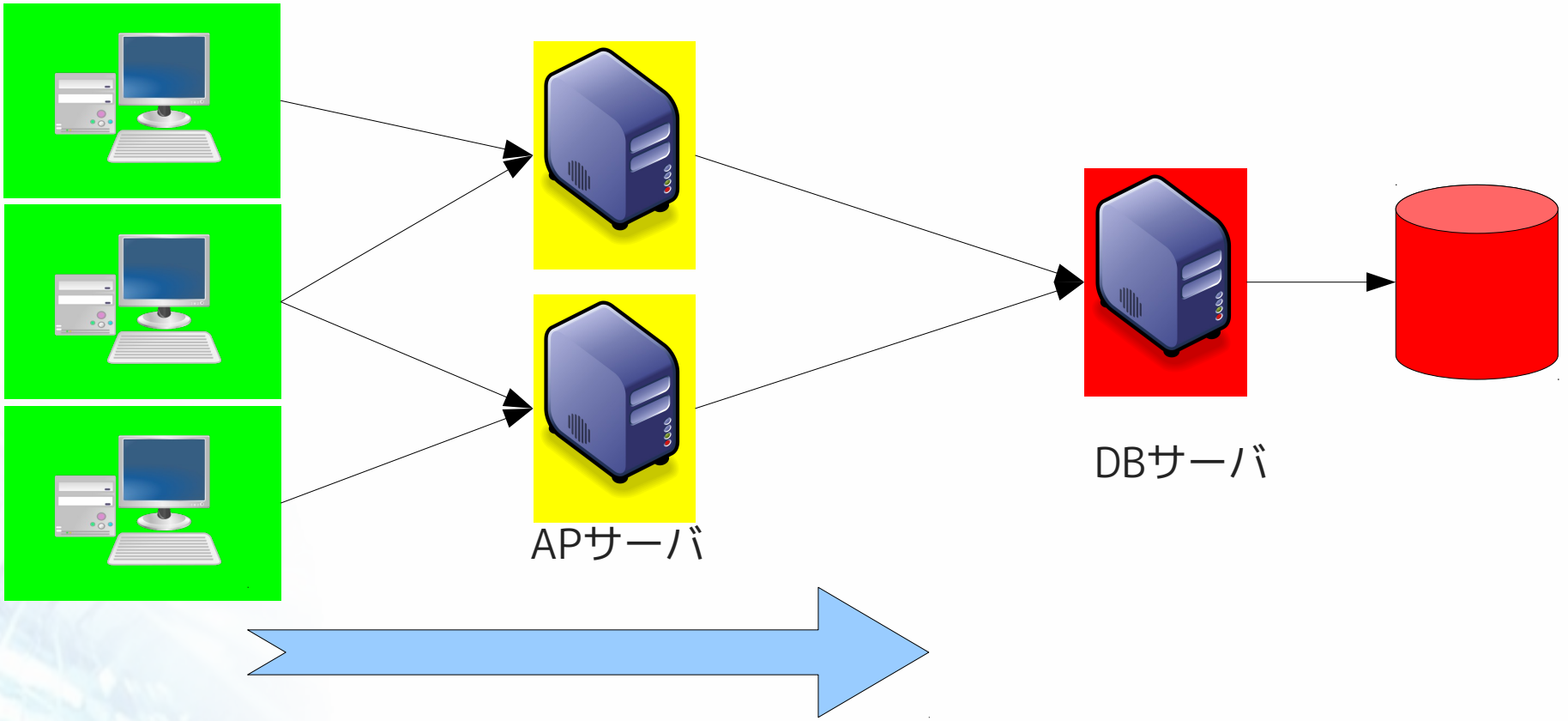
# pgpool-II最新情報<br>開発中のメモリキャッシュ機能<br>について

SRA OSS, Inc. 日本支社
石井　達夫

SRA OSS,INC.

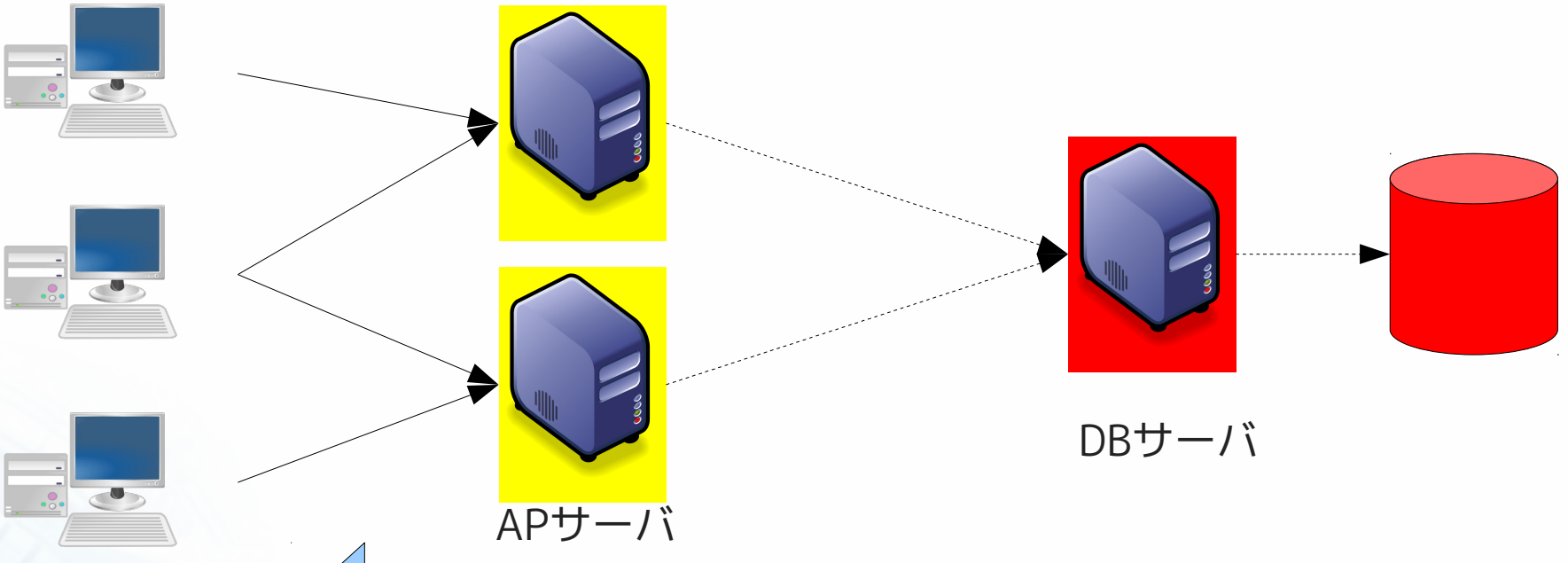
# Web環境におけるレイヤー別負荷の違い






後ろのレイヤーほど負荷が高く、ボトルネックになりやすい

# キャッシュを活用して負荷を軽減




APサーバで結果をキャッシュして返す

DBサーバで結果をキャッシュして返す

# キャッシュの実装例(1)

- アプリケーションサーバ／httpdサーバレベルでのキャッシュ実装
  - APC(Alternative PHP cache)
    - PHPが動いているサーバ上の共有メモリにキャッシュを作る
  - Apache2のコンテントキャッシュ
    - mod_cache, mod_file_cache
  - RailsやHibernateなどのORマッパーでのキャッシュ
  - SquidやVarnishなどのリバースプロキシを使う
  - memcachedなどのKVS(Key Value Store)を使ったキャッシュ
- APIやフレームワーク固有の方法でのキャッシュなのでそれぞれ使い方や特性が違う->使用するフレームワークが変わると実装し直し

# キャッシュの実装例(2)

- DBMSでのキャッシュ
  - MySQLの実装が有名。DB2にもある？
  - キャッシュにアクセスが高速化するが、DBMSへのアクセスがなくなるわけではないので、DBMSがボトルネックになることもある
- MySQLのクエリキャッシュ
  - オンメモリキャッシュ
  - クエリ文字列が一致したらキャッシュヒット
  - Prepared queryには対応していない
  - テーブルが更新されたら該当キャッシュは全クリアされる
  - MySQLを再起動したらキャッシュの内容はなくなる

# APサーバのキャッシュと DBサーバのキャッシュの比較

| | APサーバで キャッシュ | DBサーバでキャッシュ |
|---|---|---|
| キャッシュの効果 | ◎ | ○ |
| DBサーバへの負荷 | ○ | △ |
| アプリケーション透過性 | × | ○ |
| スケーラビリティ | ○ | × |

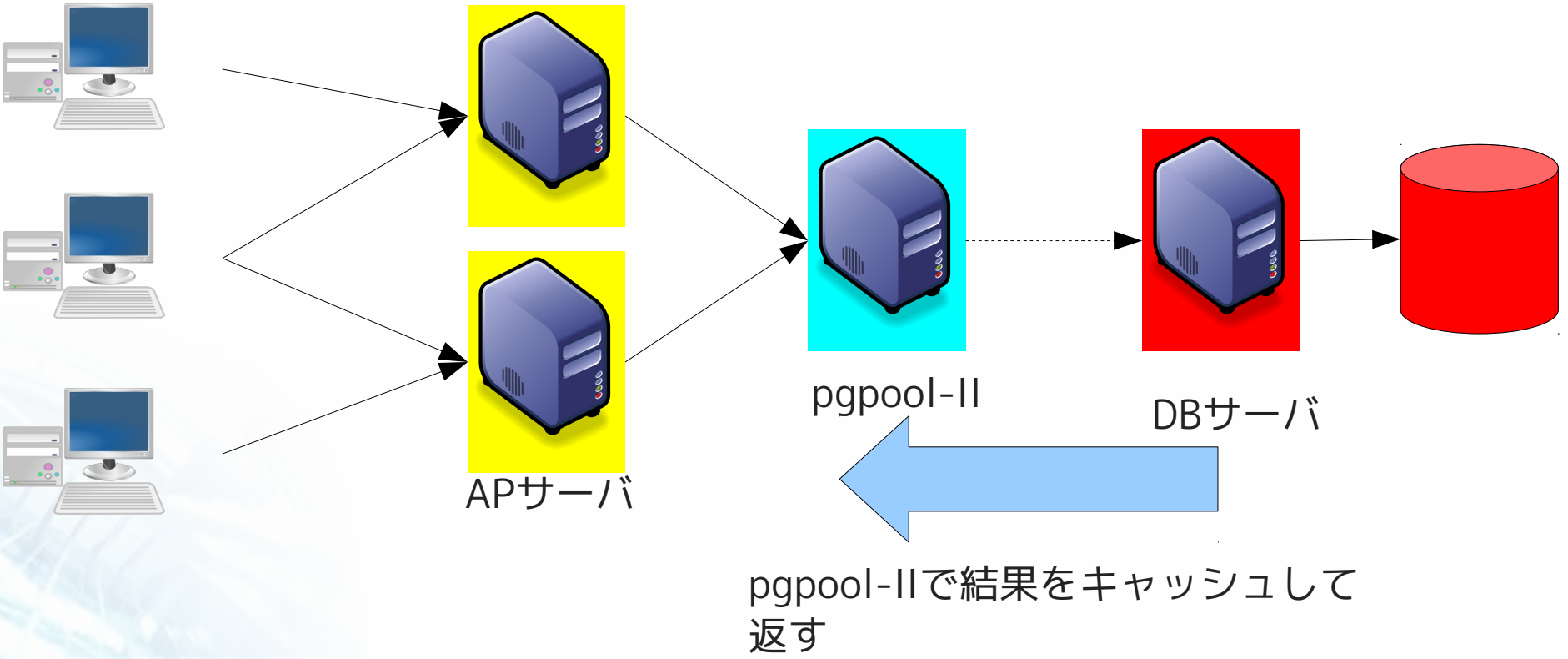
# 実はPgpool-IIにもすでにクエリキャッシュがあります

- しかしいろいろ問題が...
  - キャッシュストレージがDBなので遅い
  - DBを更新しても自動ではキャッシュが更新されない
  - 拡張問い合わせに対応していない
- そこで、新しく実装しなおすことにしました
  - Google Summer of Codeとしてプロトタイプを実装
  - 現在バグ取り、未実装部分の実装を行っています

# Pgpool-IIオンメモリクエリキャッシュの主な機能

- キャッシュストレージとして、共有メモリとmemcachedを選択できる
- セッション、ユーザをまたがってキャッシュが再利用できる
  - memcachedを使う場合は、pgpool-II、PostgreSQLを再起動した後もキャッシュを再利用することも可能
- キャッシュの更新は自動的に行われる
  - 更新問い合わせが来たらキャッシュをクリア
  - 一定時間が過ぎたキャッシュを自動クリアすることも可能
- 拡張問い合わせでもキャッシュが使える

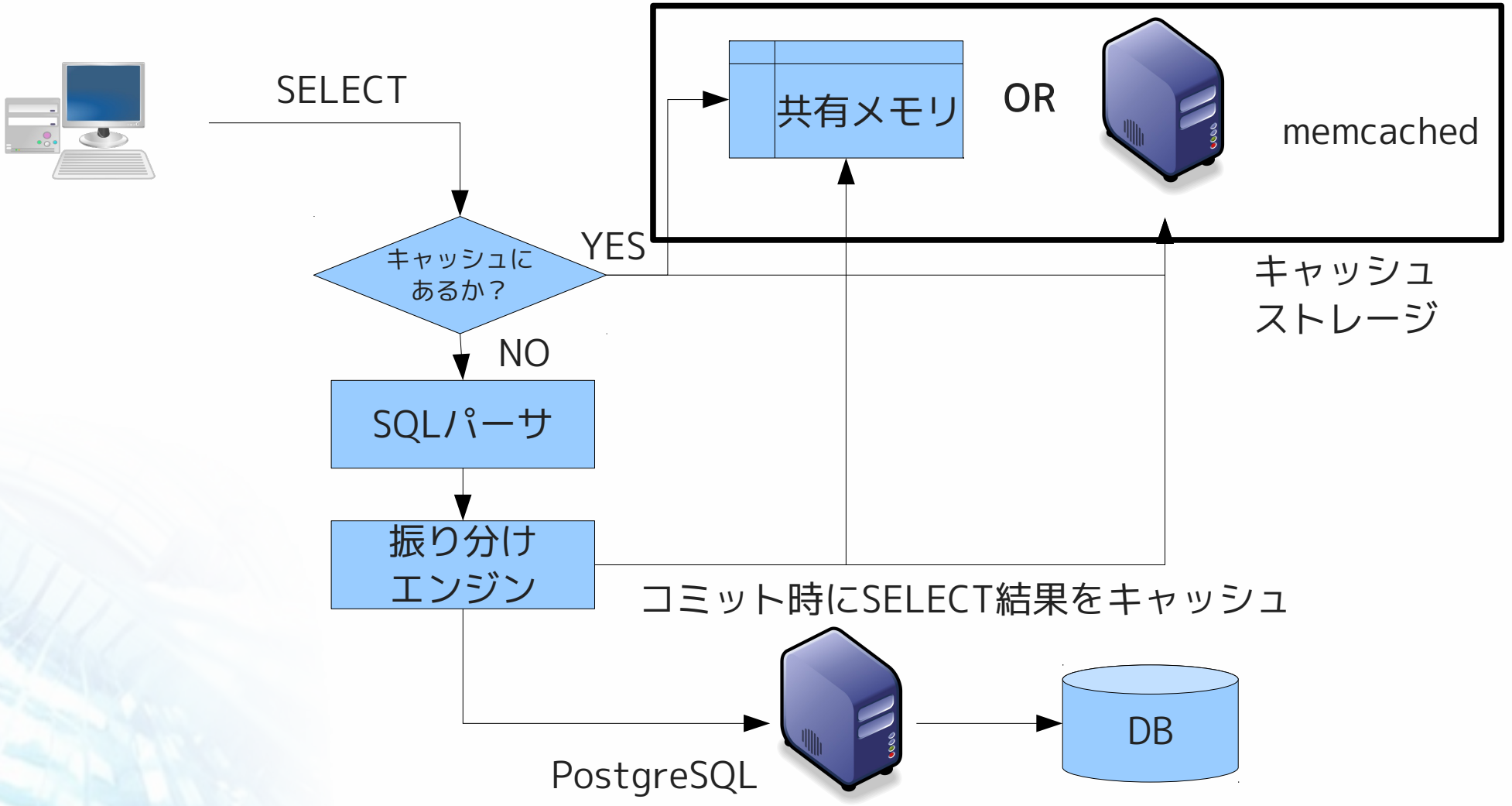
# Pgpool-IIのメモリキャッシュの利用イメージ



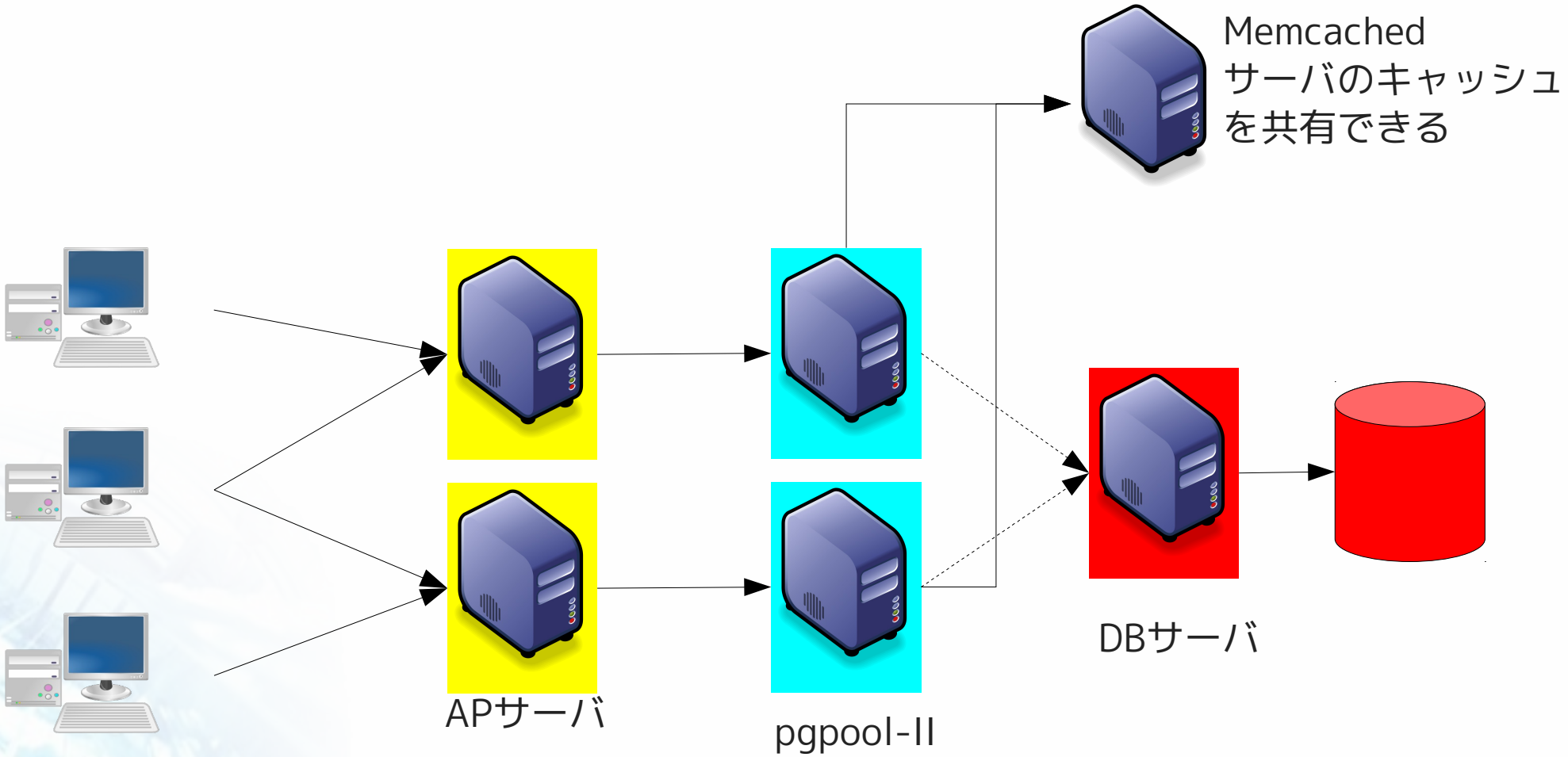
# APサーバ、DBサーバ、pgpool-IIのキャッシュ機能の比較

| | APサーバでキャッシュ | DBサーバでキャッシュ | pgpool-IIでキャッシュ |
|---|---|---|---|
| キャッシュの効果 | ◎ | ○ | ○ |
| DBサーバへの負荷 | ○ | △ | ○ |
| アプリケーション透過性 | × | ○ | ○ |
| スケーラビリティ | ○ | × | ○ |

# Pgpool-IIオンメモリ クエリキャッシュの仕組み



# キャッシュサーバの スケールアウト



# キャッシュの「ヒット」の定義

- クエリ文字列+データベース名+テーブル名+ユーザ名をMD5ハッシュしたものをキーにして一致を判断し、ハッシュ値が一致したらキャッシュがヒットしたものと見なす
- 直接SQL文を比較しないのは、長大なSQL文を保存する必要性を避けるため
- ユーザ名がキーに含まれているのは、参照権限がないテーブルのデータをキャッシュを経由して他のユーザが参照できないようにするため

# キャッシュ対象とならないSELECT

- Immutableでない関数への呼び出しを含むSELECT
  - SELECT CURRENT_TIMESTAMP;
- SELECT INTO, SELECT FOR UPDATE, SELECT FOR SHARE
- SELECTの結果データが大きいもの(> memqcache_maxcache)
  - 巨大なデータでキャッシュが使い尽くされてしまうのを防ぐため
- 成功しなかったSELECT
- ロールバックされたトランザクション内のSELECT結果
  - BEGIN;
  - INSERT INTO t1 VALUES(1);
  - SELECT * FROM t1;
  - ROLLBACK;
    - もしSELECT * FROM t1の結果をキャッシュしてしまうと、次のSELECT * FROM t1で存在しないはずの「1」が返ってしまう
- 一時テーブルを含んでいるSELECT
  - セッションが終了するとテーブルが消えてしまうため
- Unloggedテーブルを含んでいるSELECT
  - PostgreSQLクラッシュ後の再起動でデータが消えてしまうため

# キャッシュの更新/無効化ポリシー (invalidation)

- キャッシュ対象のテーブルが一部でも更新されたら、そのテーブルを参照しているキャッシュをすべて自動的に削除する
  - 更新クエリ：INSERT/UPDATE/DELETE/TRUNCATE
  - そのために、キャッシュを登録する際に参照しているテーブルのOIDを調べ、ファイルに登録しておく
  - 更新クエリが実行されたらそのファイルを調べて、関連するキャッシュを削除する
- データベースやテーブルが削除された場合も同様
- テーブル構造が変わるようなコマンド(ALTER TABLE)が実行された場合も同様
- 更新が多いDBには向かない
- 有効時間を過ぎたキャッシュは無効になる(無効にしない設定も可能)
- 手動でキャッシュを削除可能(計画中)
- キャッシュストレージがmemcachedの場合はちょっと悩ましい
  - pgpool-IIやPostgreSQLが再起動しても、memcachedが動いていれば前回のキャッシュを有効に使える？

# 制限事項

- VIEWもキャッシュされるが、VIEWが参照しているテーブルが更新されてもキャッシュは有効
- スキーマが異なっていても、DB名、テーブル名が同じならば同じテーブルと見なされる
- トリガによって暗黙的に更新されるテーブルが認識できない
- 外部キーが指定されていて、ON DELETE CASCADEなどで他のテーブルの行が暗黙的に更新されたことが認識できない
- DROP TABLE CASCADEで暗黙的に削除されたテーブルが認識できない
- TRUCATE TABLE CASCADEで暗黙的に内容が削除されたテーブルが認識できない

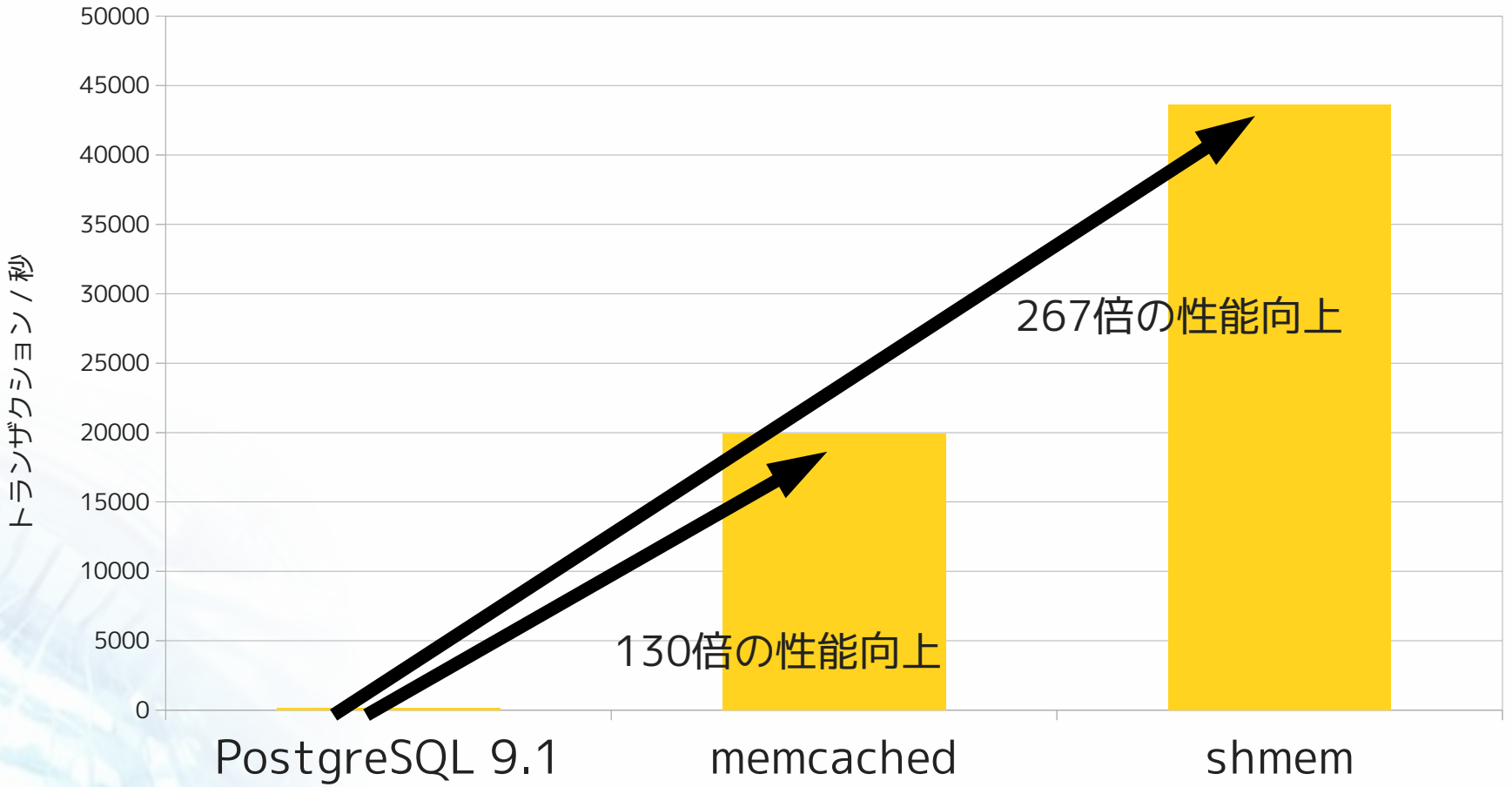
# オンメモリクエリキャッシュの設定項目

- memory_cache_enabled = false
  - メモリキャッシュの有効／無効
- memqcache_method = 'shmem'
  - キャッシュストレージの選択。shmem(共有メモリ) or memcachedが選択可能
- memcachedを選択した場合の設定項目
  - memqcache_memcached_host = 'localhost'
  - memqcache_memcached_port=11211
- memqcache_total_size=134217728
  - トータルキャッシュサイズ
- memqcache_expire=60
  - キャッシュの有効時間
- memqcache_maxcache=1024
  - 格納できる最大のSELECT結果サイズ
- memqcache_cache_block_size=1048576
  - キャッシュブロックのサイズ(共有メモリのときのみ)
- memqcache_oiddir = '/var/log/pgpool/oiddir'
  - テーブルOIDを格納する領域

# ベンチマーク！

- SELECT count(*)という、PostgreSQLでは全スキャンになる遅いクエリを約100%のキャッシュヒット率でアクセスしたケース(ベストケース)
  - SSDを使い、しかもテーブルがキャッシュに乗った状態なので、実環境ではもっと差が開く可能性がある
- ハードウェア
  - ノートPC(dual core i5-2540M CPU @ 2.60GHz、Hyper threading有効、メモリ8GB、SSD(Intel 320 MLC)
- ソフトウェア
  - Pgpool, PostgreSQL, memcachedを同じマシンで動かしている
    - PostgreSQL 9.1
    - Shared_buffers = 32MB
  - Pgbench
    - SQL: SELECT count(*) FROM pgbench_accounts;
      - 1000回実行
      - 10万件、テーブルサイズ13MB
      - I/Oネックにならない

# ベンチマーク結果



# まとめ

- 各レイヤにおけるキャッシュソリューションを比較
  - アプリケーションサーバでのキャッシュは効果が高いが、アプリケーションの改造が必要になることが多い
  - DBMSでのキャッシュは、アプリケーションの改造が要らないメリットがある。また、DBMSの負担を軽減する。更新の多い用途には向かない。スケールアウトができない
  - pgpool-IIのオンメモリキャッシュは、アプリケーションの改造が不要、DBMSの負担を軽減し、さらにスケールアウトすることができる。ただし、更新の多い用途には向かない
- Pgpool-IIのようなミドルウェアにおけるキャッシュは、用途によって大きなメリットがある

# 参考URL

- pgpool-IIのダウンロードはこちら
  - http://pgfoundry.org/projects/pgpool/
  - 開発ML(英語)もこちらから登録できます
- メインWebサイト
  - http://pgpool.projects.postgresql.org/
- SRA OSS's website
  - http://pgpool.srasoss.jp
  - 日本語ML(pgpool-general-jp)登録もこちらから
- Twitter
  - @pgpool2