Boosting Performance and Reliability by using pgpool-II

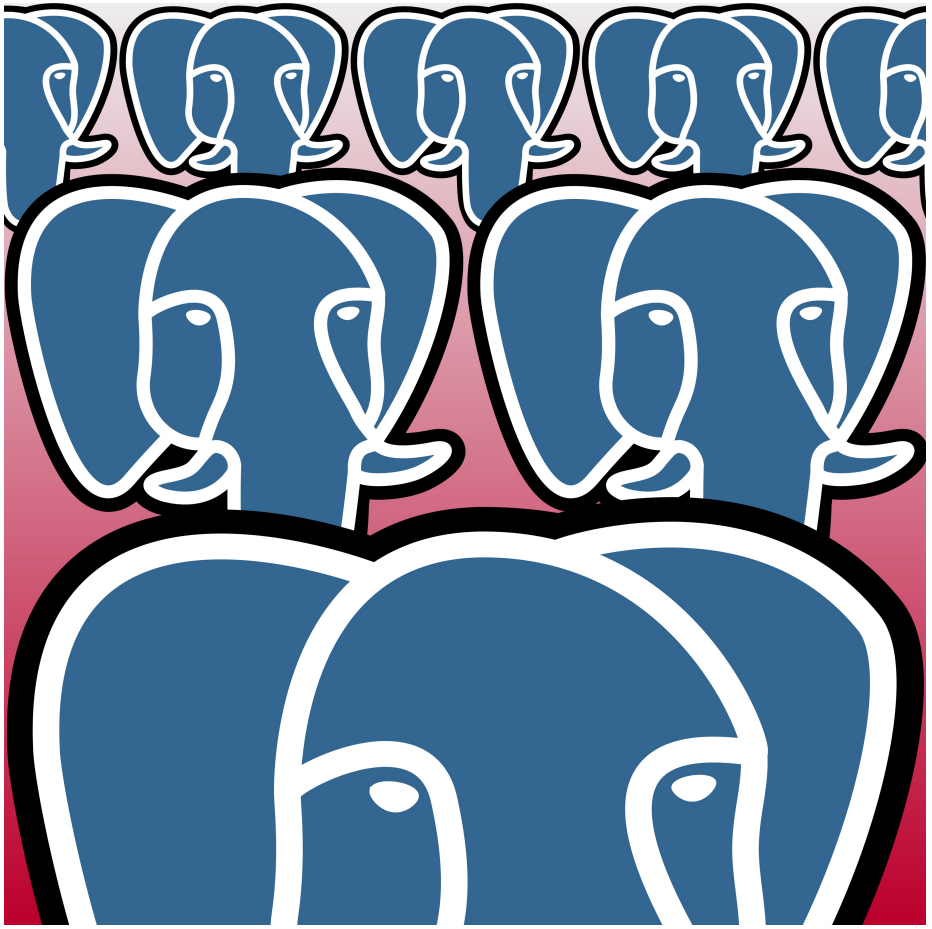
Tatsuo Ishii

SRA OSS, Inc. Japan

SRA OSS, INC.

Introducing myself

- Working on the PostgreSQL project for many years
- Enjoying working for an open source company as a manager/engineer
- Original author of pgpool-II



What is pgpool-II anyway?

- One of open source cluster middle ware for PostgreSQL
- Add functionalities which are currently missing in PostgreSQL
  - Connection pooling
  - Load balancing
  - On memory query cache
  - Fail over
  - Replication(either built in synchronous replication or external replications engines such as streaming replication and Slony-I)
  - Built in HA(watchdog)

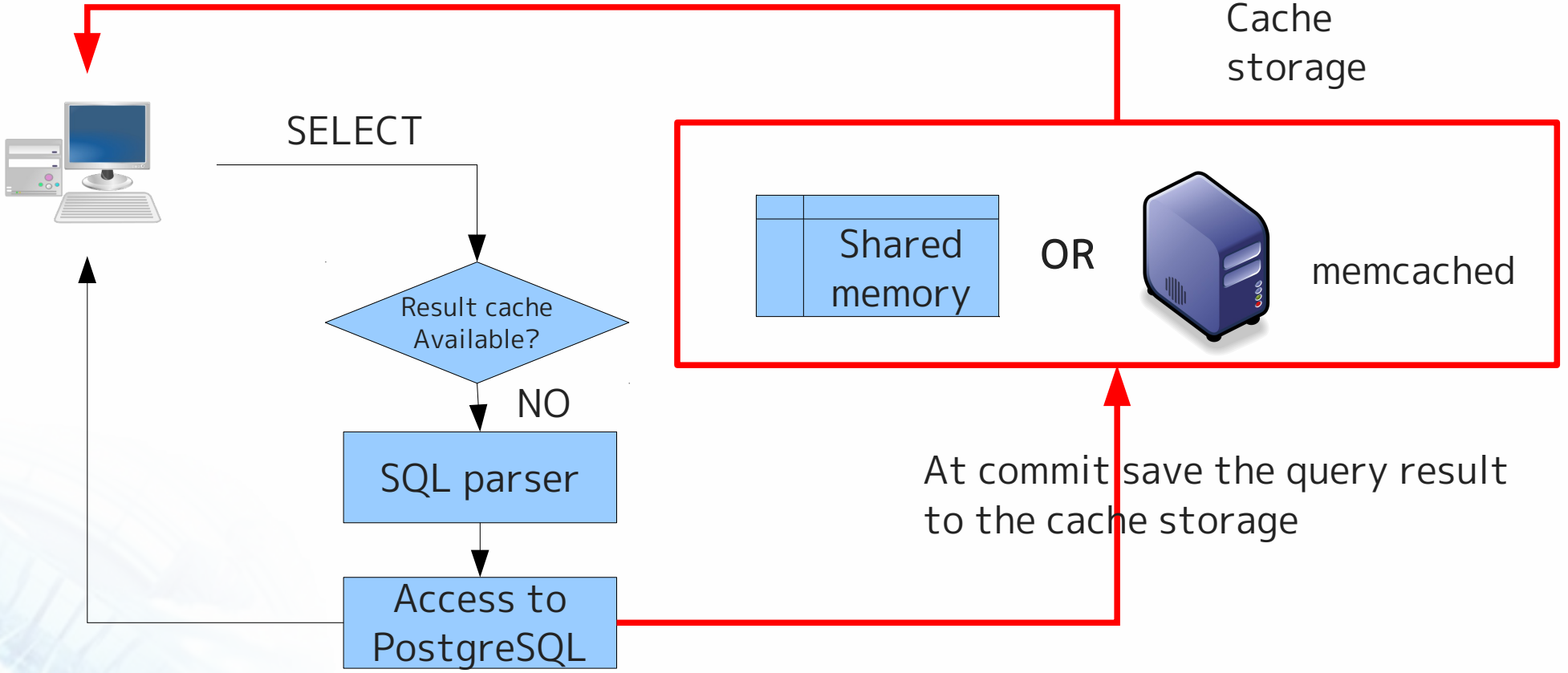
On memory query cache

Major features of “on memory query cache”

- If cache hits, no access to PostgreSQL is involved, thus very fast
- Choices of cache storage
  - Shared memory
    - Fast and easy to use but cache size is limited
  - Memcached
    - Can be used as very large cache storage, but network overhead is involved
- Many of applications do not need to modify to use cache
- Automatic cache invalidation
  - Table updates (including ALTER TABLE)
  - Cache expiration by specified duration
  - There are some cases which pgpool cannot recognize table update
- Can specify which table should be in cache and which is not by using regular expression

Data flow of “on memory query cache”

If there's cache for the query result, returns it directly to client



If there's no cache for the query result, get result from PostgreSQL and returns to client

How to identify cache objects?

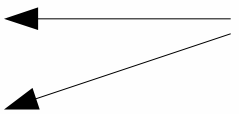
- Cache is essentially identified by query string
- Pgpool does not store query string itself. Rather , stores md5 result to save space for very long query string
- Simple query case
  - Cache id = md5(SQL string + user name + database name)
- Extended protocol case(prepared statements)
  - Cache id = md5(SQL string + bind parameters + user name + database name)

When pgpool does not create cache

- SELECTs including views
  - Pgpool does not know changes of underlying tables
- SELECTs including non immutable functions
  - Consider “SELECT current_timestamp;”
- SELECTs including temp tables, unlogged tables
- SELECT result is too large (memqcache_maxcache)
- SELECT FOR SHARE/UPDATE
- SELECT starting with "/*NO QUERY CACHE*/" comment
- SELECT including system catalogs
- Tables listed in “black_memqcache_table_list”
- Tables listed in “white_memqcache_table_list” will be cached even above conditions are met

SELECT in an explicit transaction

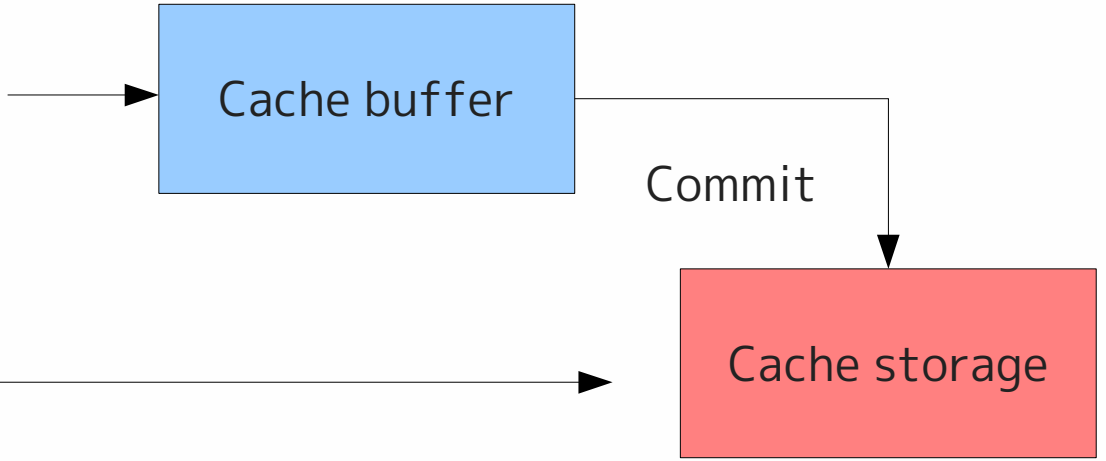
```
BEGIN;
INSERT INTO table1...;
INSERT INTO table2...;
SELECT * FROM table1;
SELECT * FROM table2;
:
:
ROLLBACK;
```



Cannot cache these results because previous INSERTs canceled.

Committing to cache storage when transaction commits

```
BEGIN;
INSERT INTO table1...;
INSERT INTO table2...;
SELECT * FROM table1;
SELECT * FROM table2;
:
:
COMMIT;
```



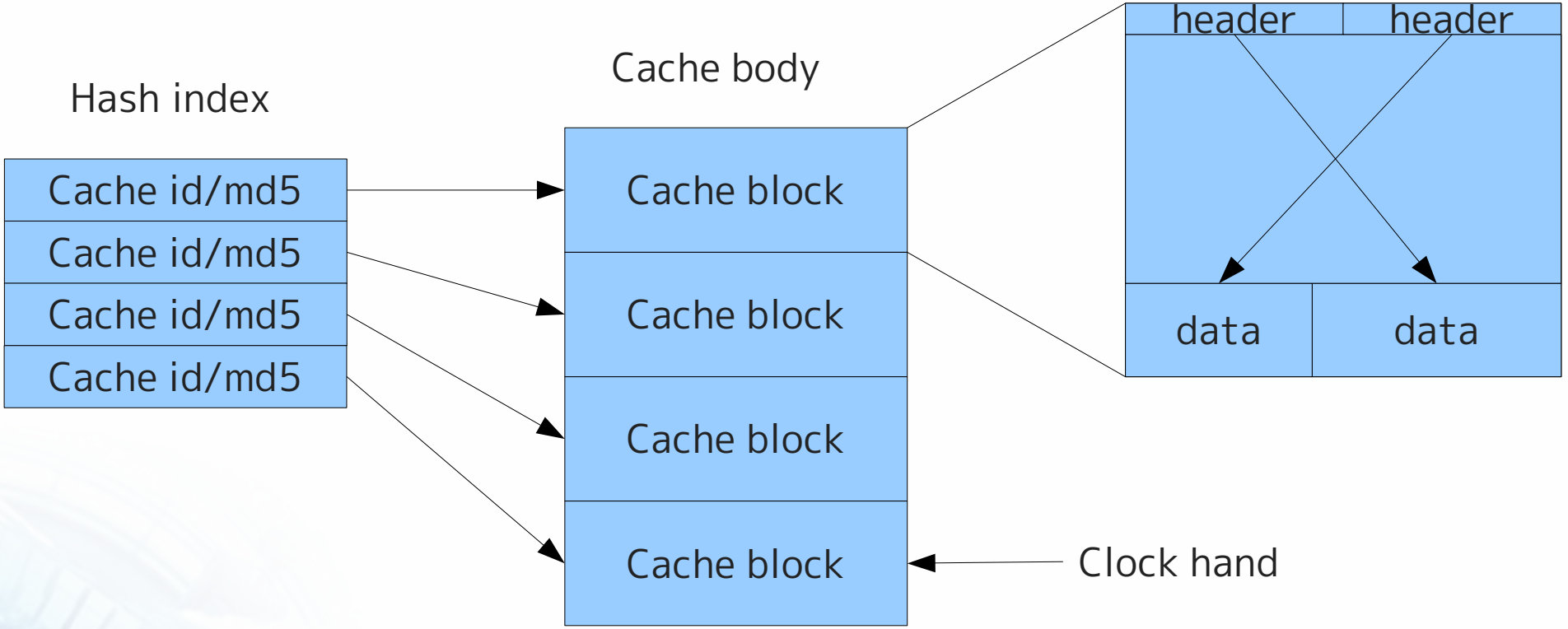
Invalidation/expiration of cache

- Cache is invalidated when corresponding tables are updated
- Cache is expired when *memcache_expire* reached

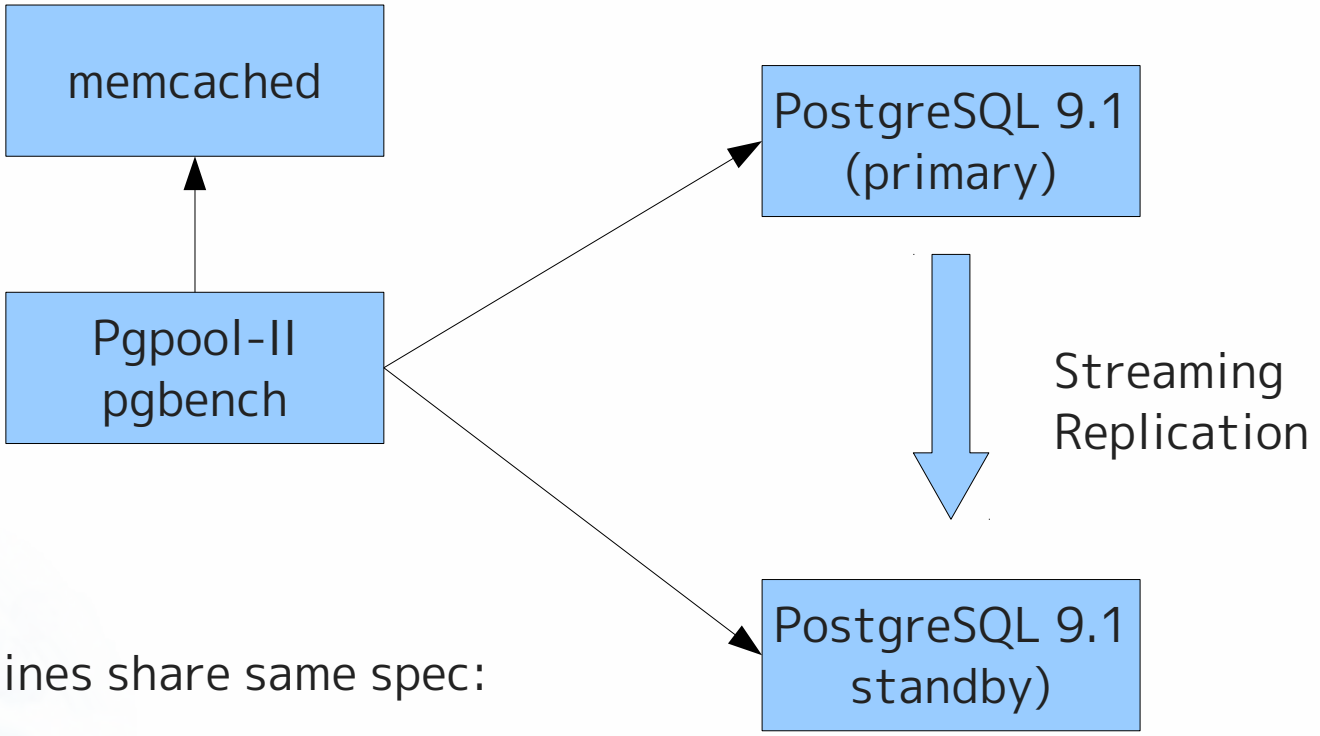
Limitations

- If a table is modified, all caches for the table are invalidated
  - Not recommended for update intensive systems
  - Best for systems whose cache hit ratio > 70%
- Users can only access to cache which was created by the same user(security reason)
- Views are not cached
  - Pgpool-II is unable to know the updates to the underlying tables
- Cascade update to tables are not recognized
- Table change by trigger is not recognized
- Too large query result cannot be cached(~1MB, depending on the configuration)

Shmem cache data structure



Benchmark configuration

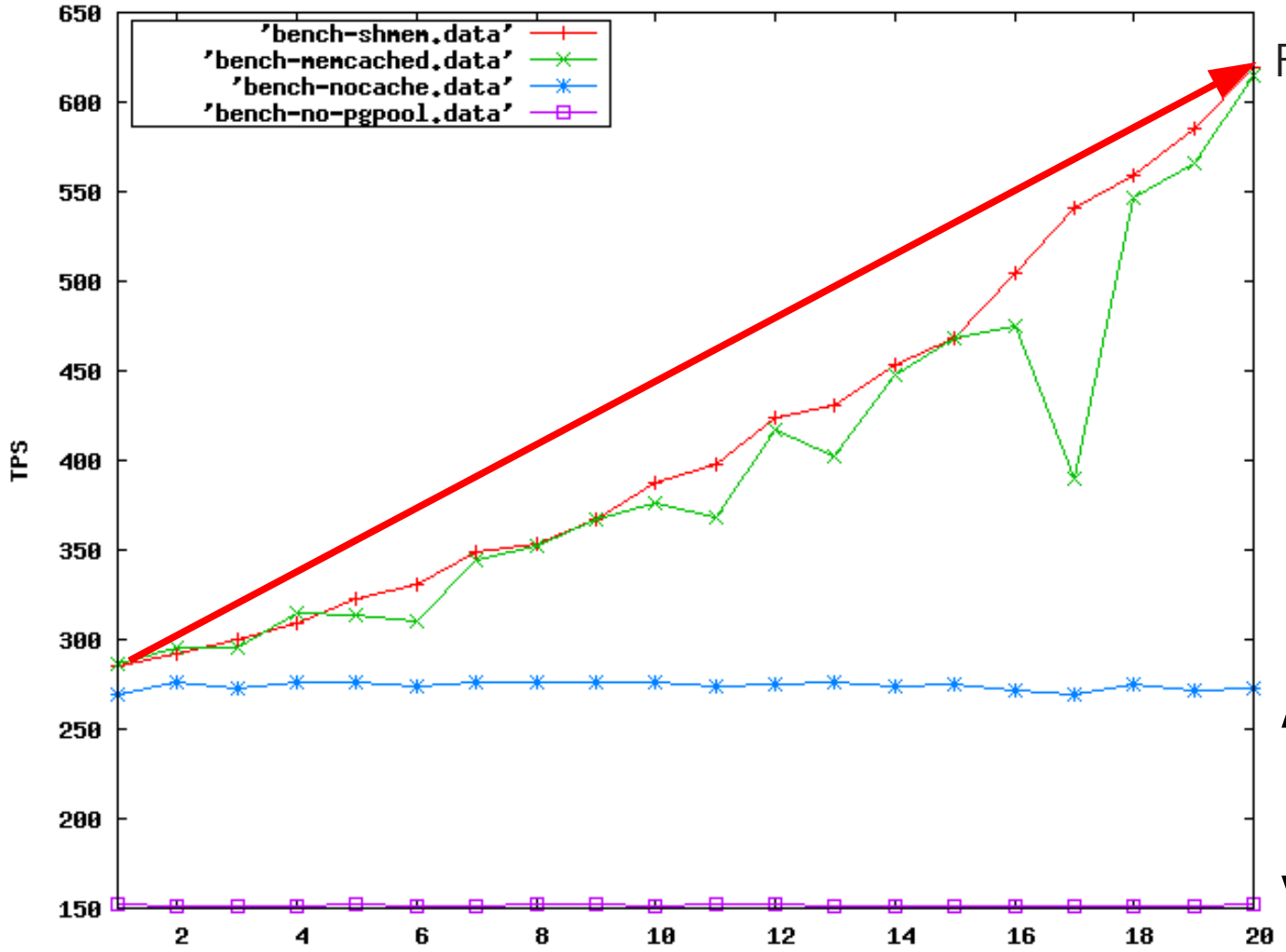


All the machines share same spec:
Cent OS 5.5
Core 2 Duo 2.33GHz
Mem: 2GB

Benchmark configuration

- Scale factor = 1(13MB DB)
- pgbench_accounts primary key is removed to simulate non trivial SELECTs
  - Each SELECT takes about 6.7ms
- SELECT only test
  - Pgbench -S -T 10 (1 run is 10seconds)
- Cache storage is 64MB(for both shmem and memcached)

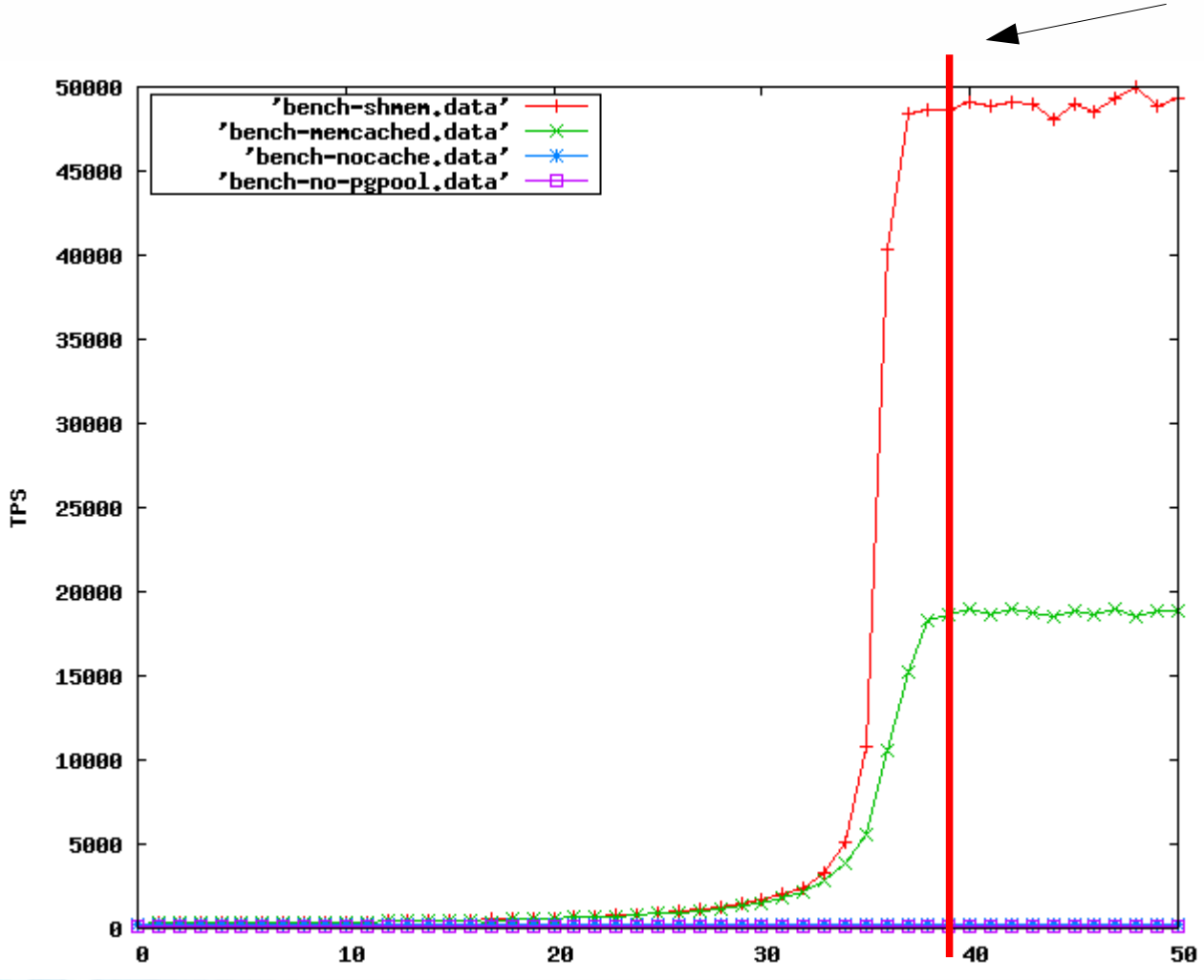
First 20 run of pgbench



Filling up cache

Effect of load balance

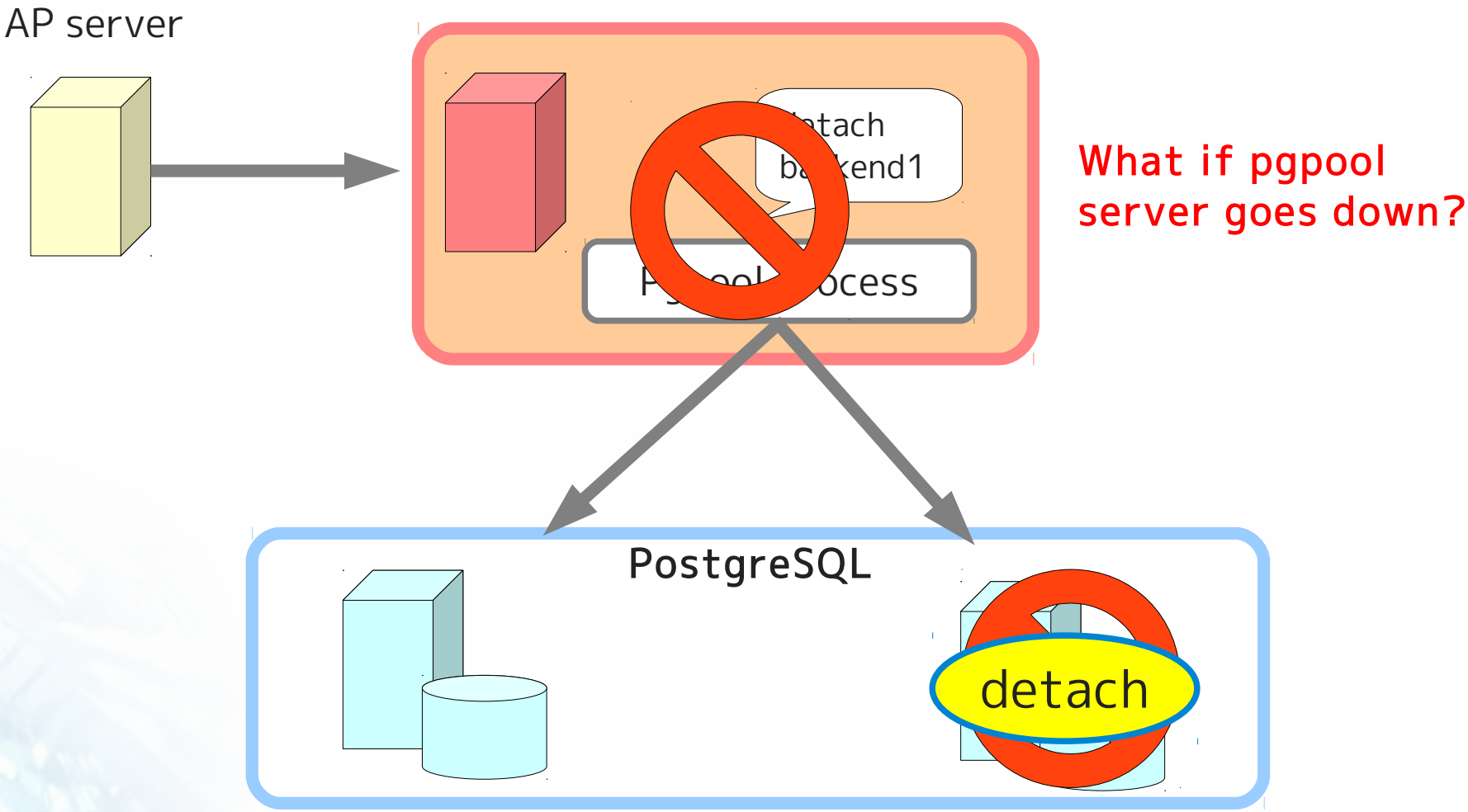
After cache is filled up



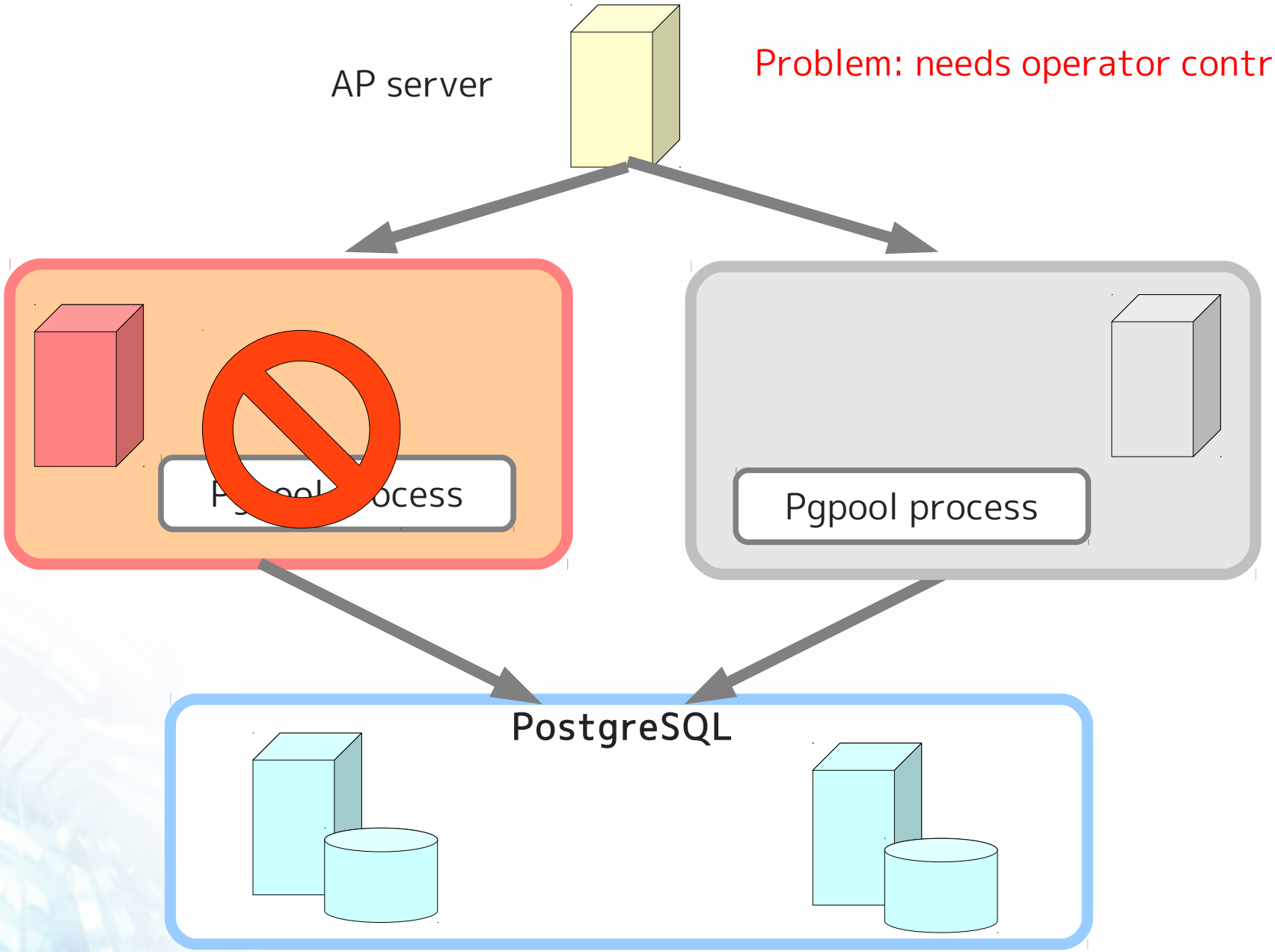
- After cache is filled up both cache method keep high performance.
- Shmem outperforms memcached significantly.

Built in HA: Watchdog

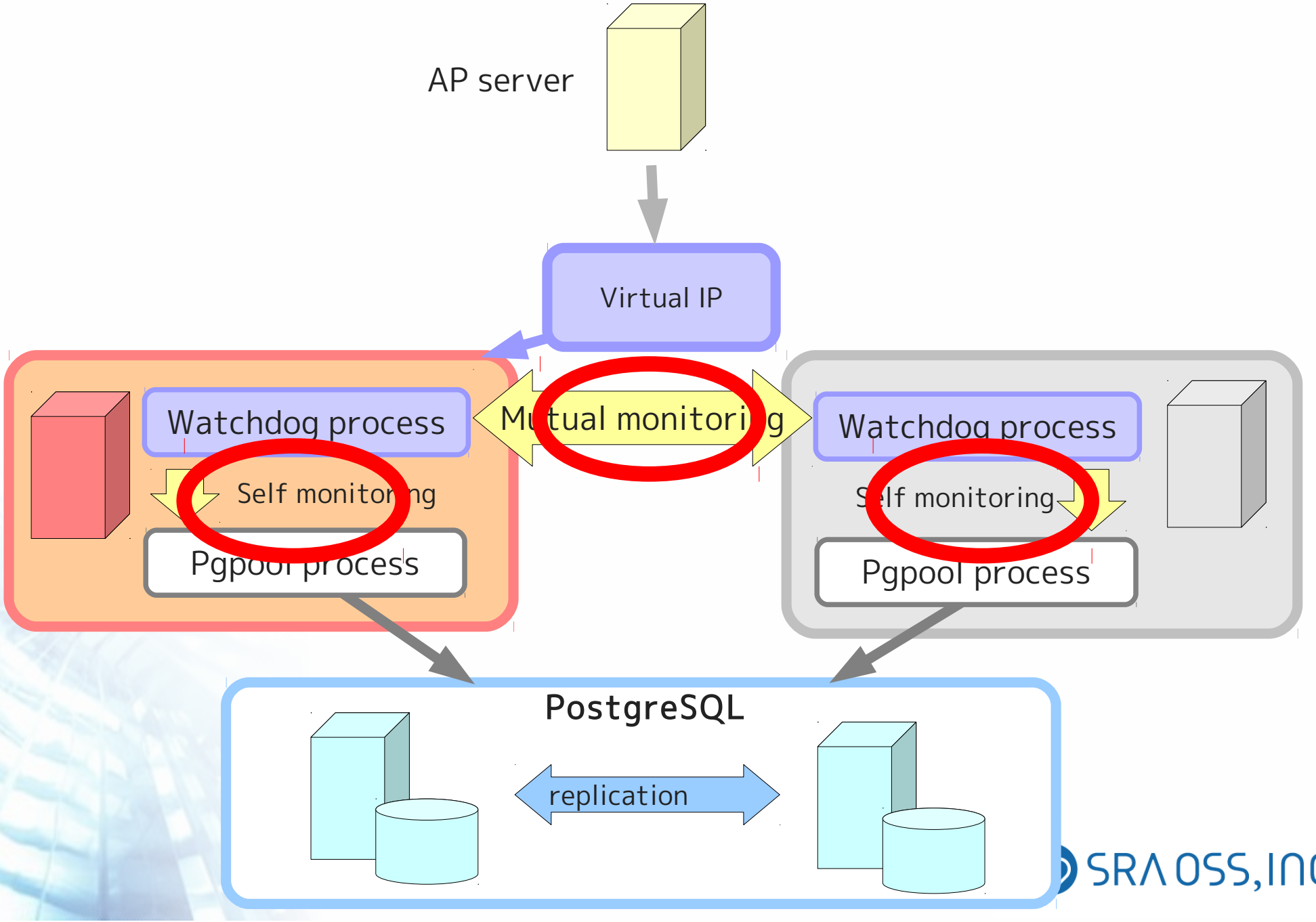
Pgpool-II manages fail over



What about adding standby Pgpool-II?



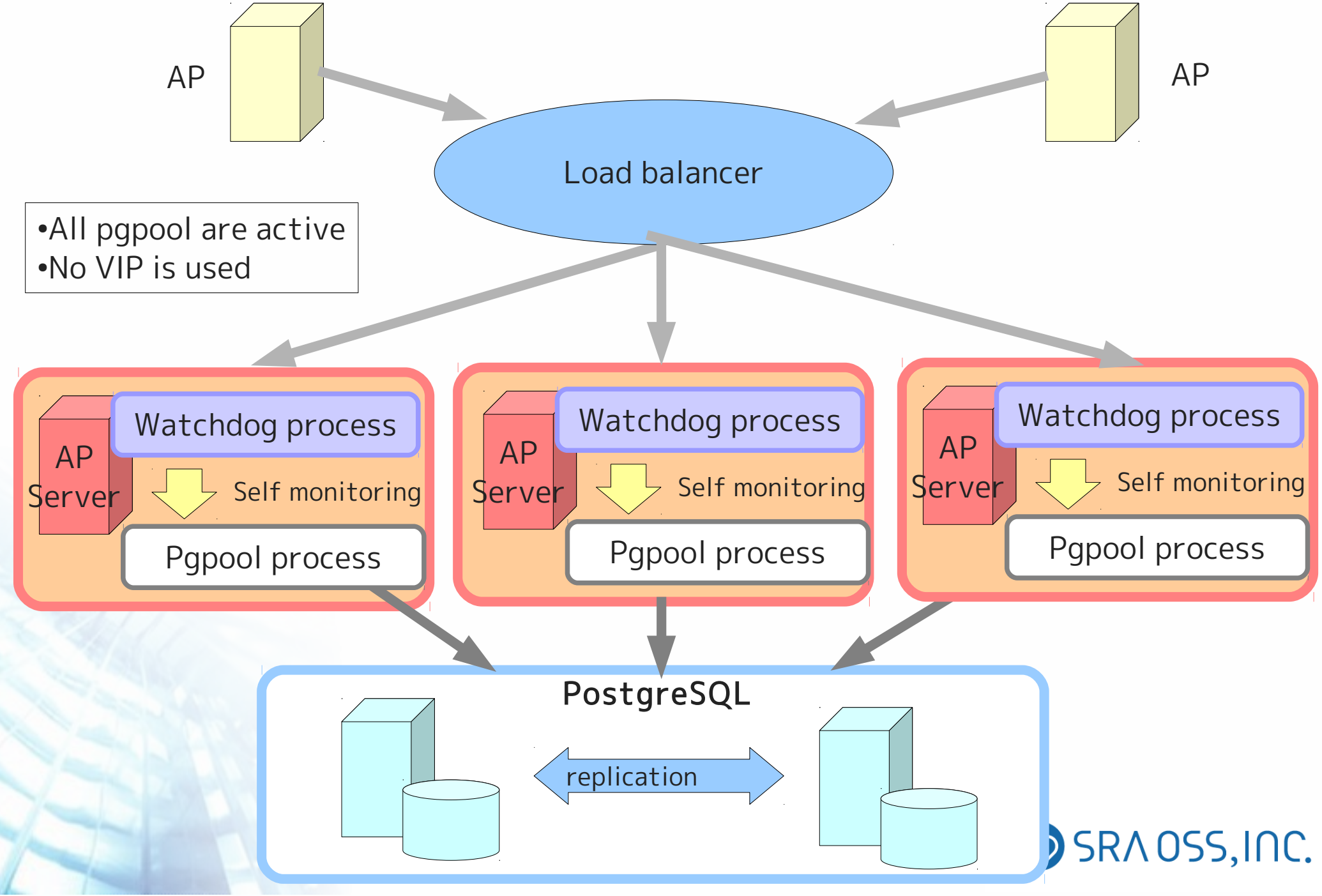
Solution: Watchdog



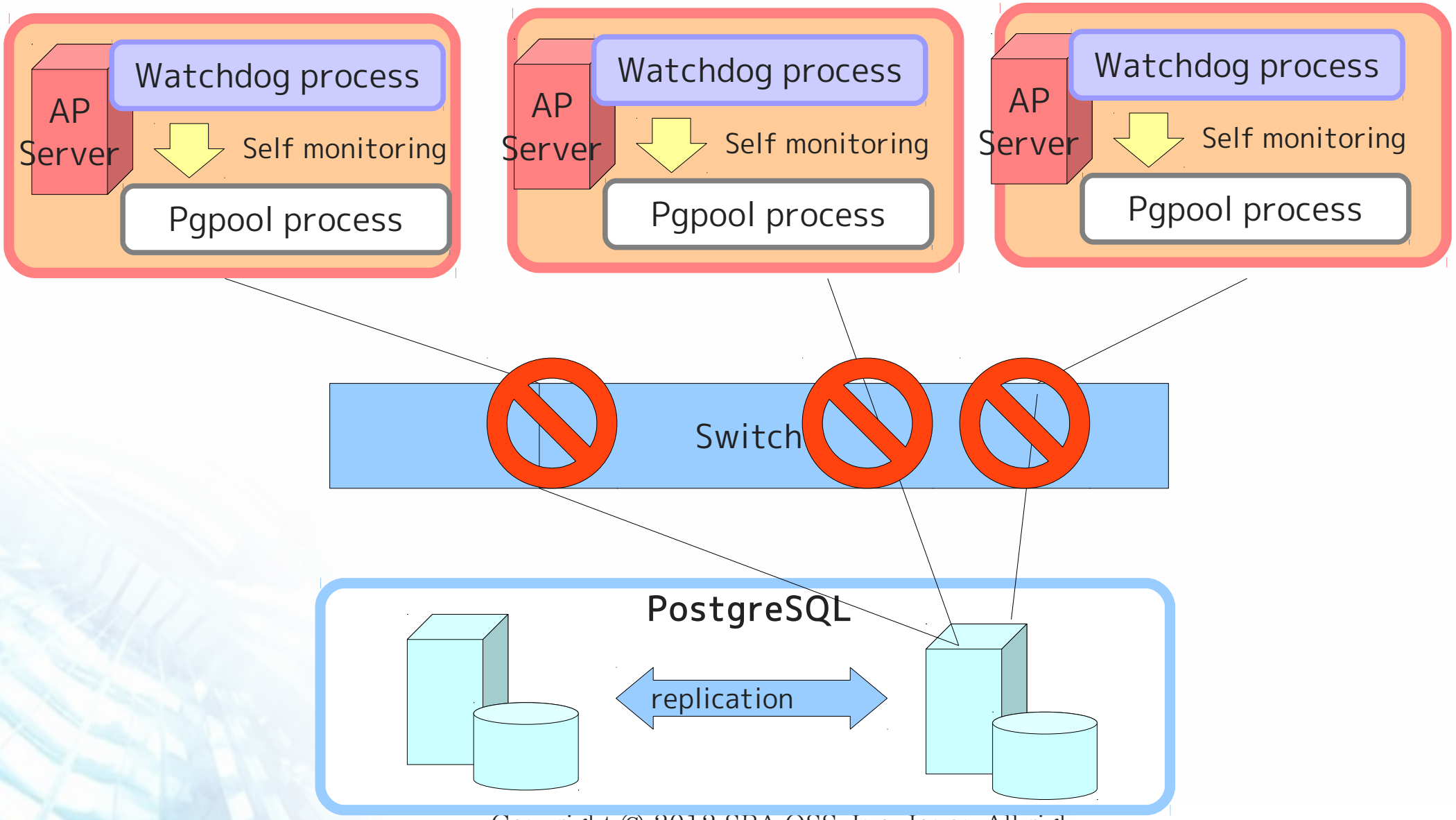
Why we need watch dog?

- We already have pgpool-HA
- Watchdog is a better substitute for pgpool-HA because:
  - Easy to install/manage
  - Can control multiple pgpool-II instances, which is important in multi master configuration

Multi master configuration



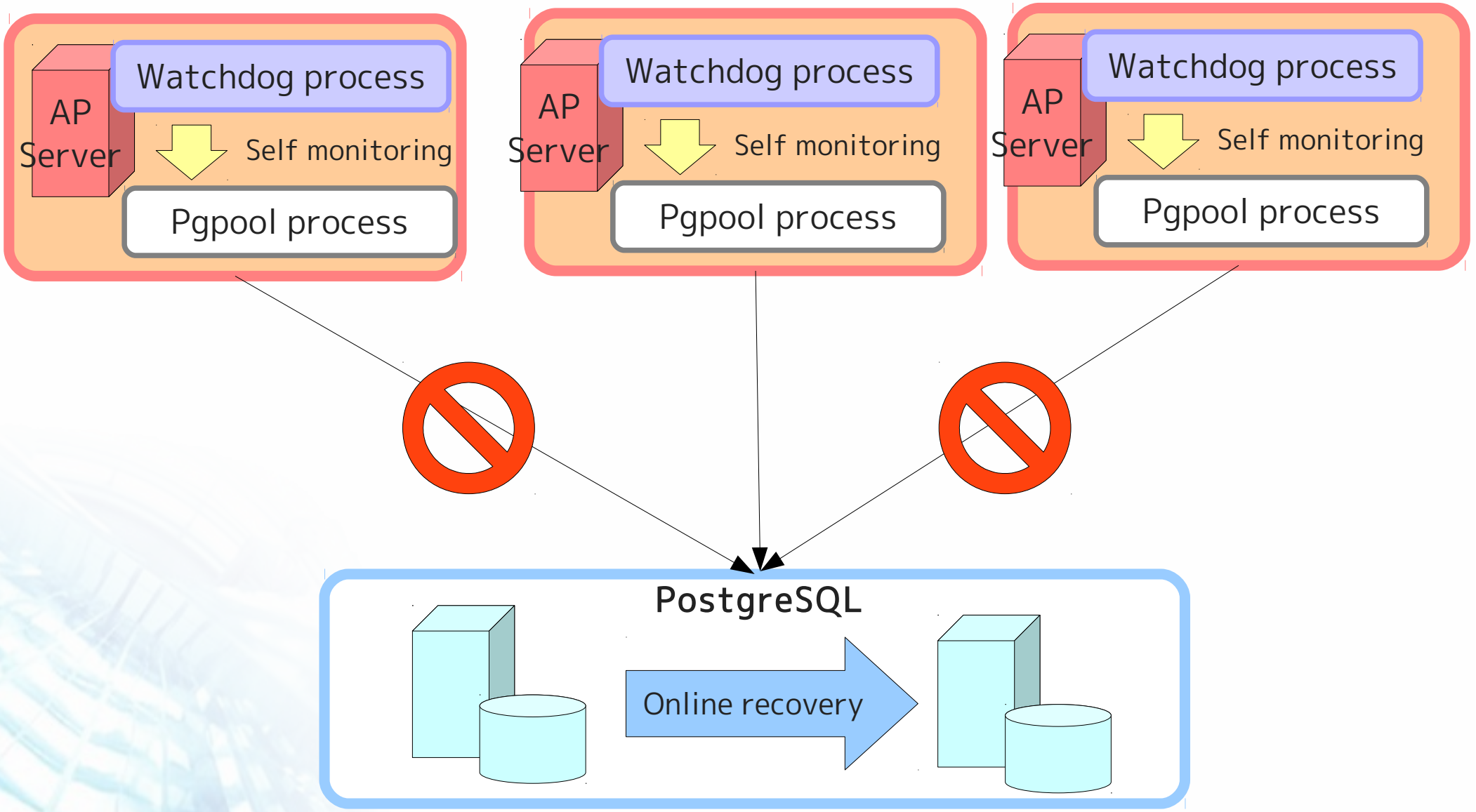
Watchdog synchronizes status



Synchronizing backend status among pgpool instances

- Attaching node
  - pcp_attach_node
- Detaching node
  - Either by pcp_detach_node or fail over event
- Promoting node
  - pcp_promote_node

Watchdog coordinates online recovery



Future plans for watchdog

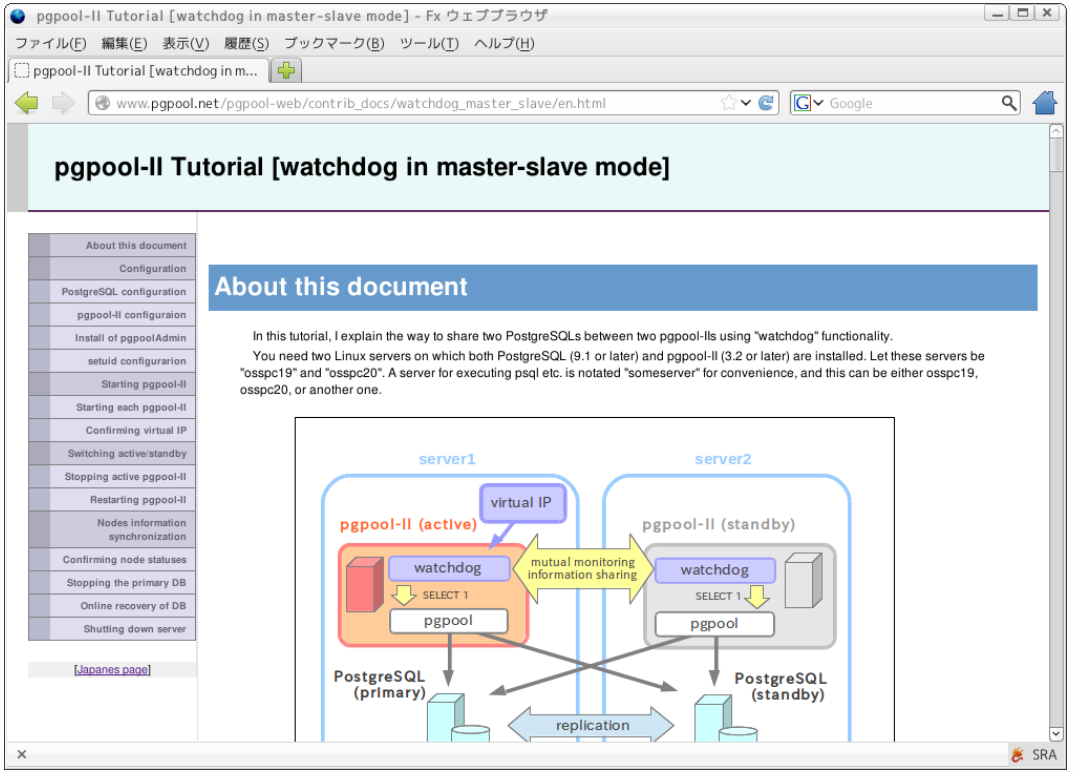
- Secure protocol
  - Currently communication between watchdog is not secure (no authentication required)
- Allow to handle multiple communication path between watchdog

Summary

- Pgpool-II enhances DB performance
  - Connection pooling
  - Load balancing
  - On memory query cache
- Pgpool-II enhances DB availability
  - Replication
  - Fail over
  - Watchdog

References

- Pgpool-II official site
  - http://www.pgpool.net
  - Download
  - Docs
    - There are nice tutorials. Please take a look at!
    - http://www.pgpool.net/mediawiki/index.php/Documentation
  - Mailing lists
  - Bug track



SRA OSS, INC.