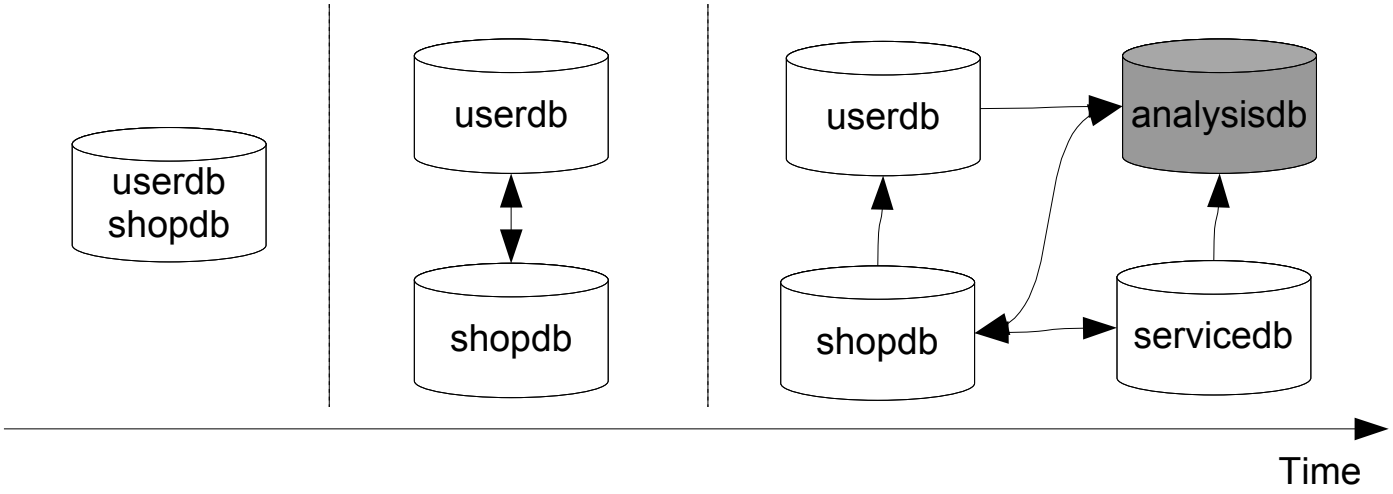
plProxy, pgBouncer, pgBalancer

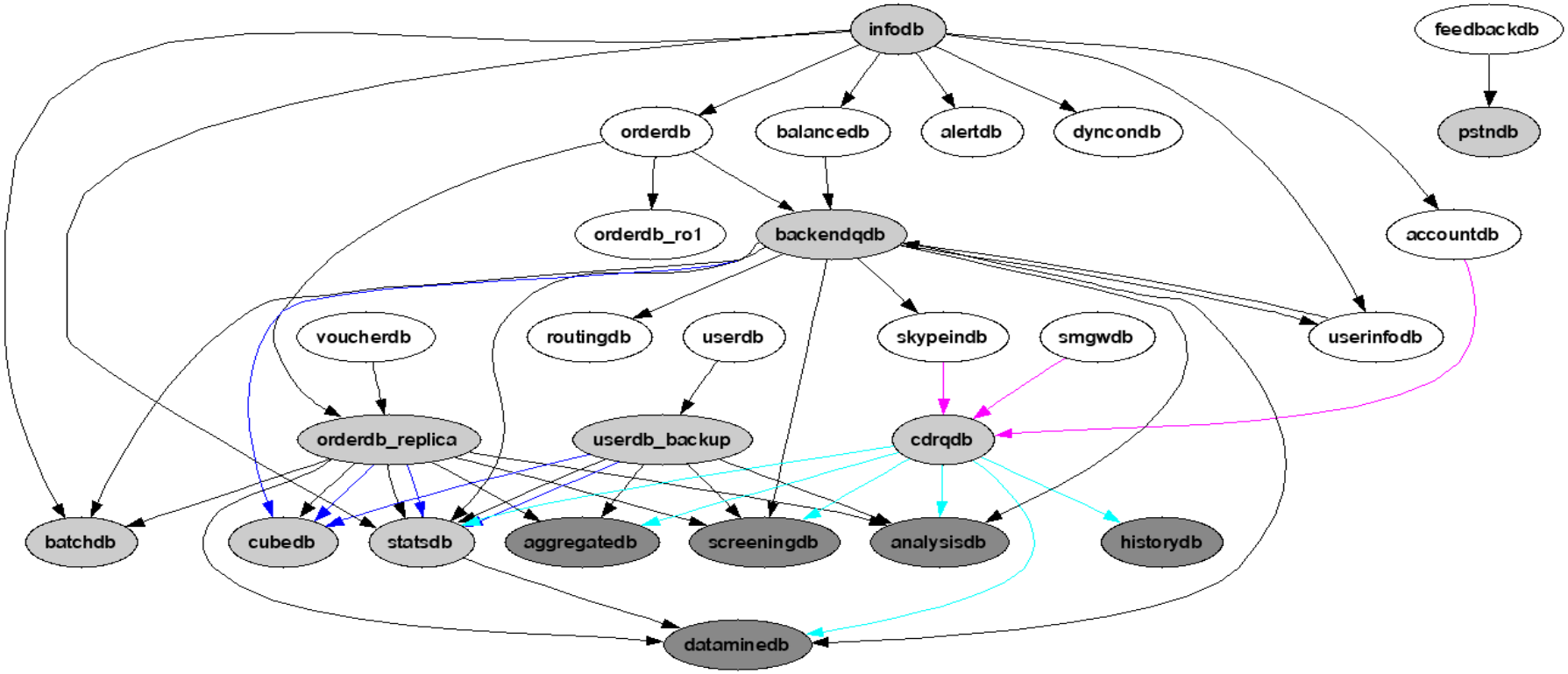
Asko Oja

Vertical Split

- All database access through functions requirement from the start
- Moved out functionality into separate servers and databases as load increased.
- Slony for replication, plpython for remote calls.
- As number of databases grew we ran into problems
  - Slony replication became unmanageable because of listen/notify architecture and locking.
- We started looking for alternatives that ended up creating SkyTools PgQ and Londiste.

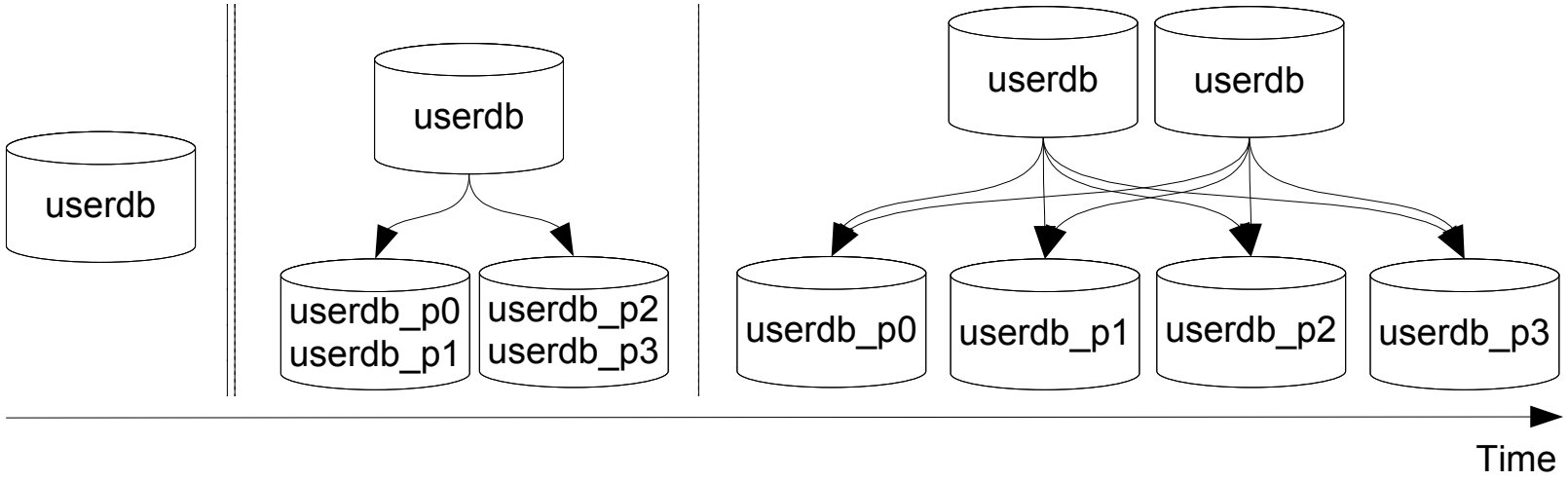


Vertical Split Picture



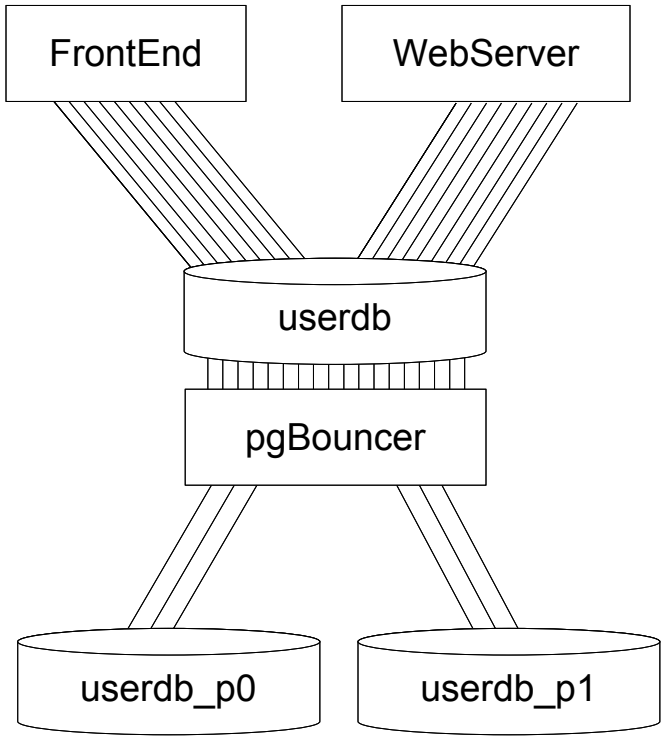
Horizontal Split

- One server could not service one table anymore
- We did first split with plpython remote calls
- Replaced it with plProxy language but it had still several problems
  - complex configuration database
  - internal pooler and complex internal structure
  - scaleability issues



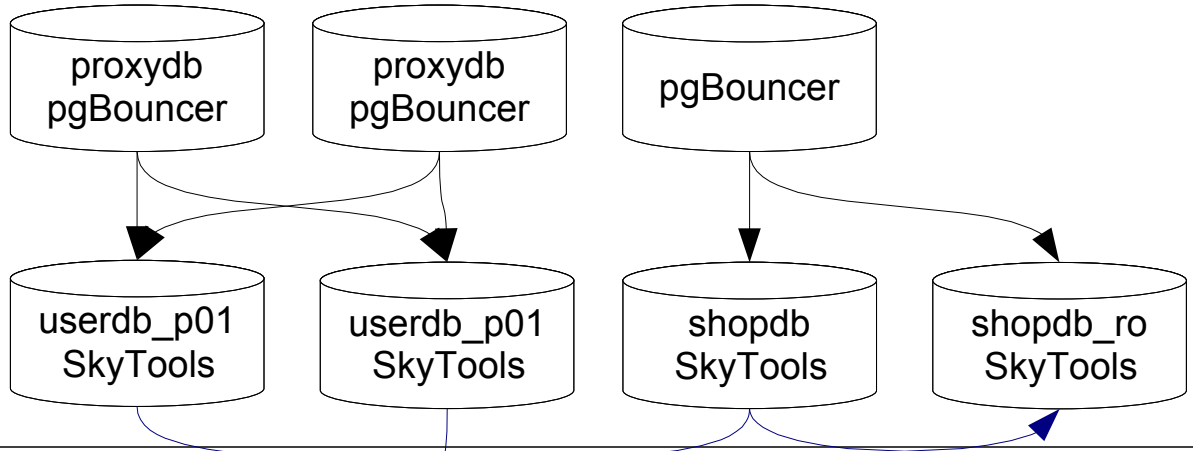
plProxy Version 2

- plProxy second version
  - just remote call language
  - simplified internal structure
  - added flexibility
  - added features
  - configuration and management improved
- Connection pooling separated into pgBouncer.
- Resulting architecture is
  - Scaleable
  - Maintainable
  - Beautiful in it's simplicity :)

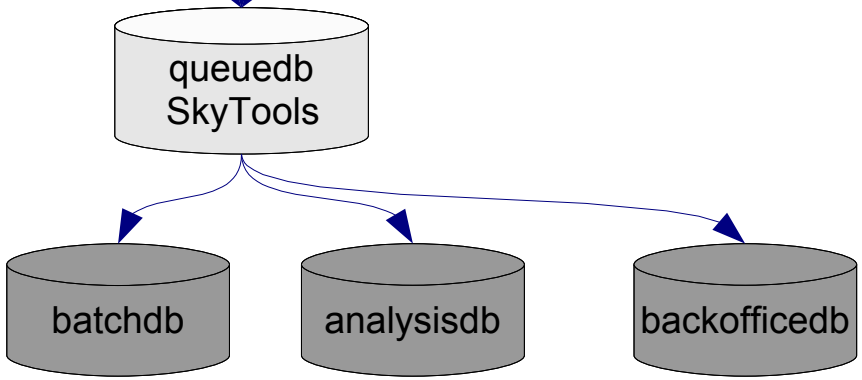


Overall picture

Online databases
- Proxy db's
- pgBouncers
- OLTP db's
- read only db's



Support databases
- Queue db's
- Datamining
- Batchjobs
- Backoffice
- Greenplum



plProxy: Installation

- Download PL/Proxy from http://pgfoundry.org/projects/plproxy and extract.
- Build PL/Proxy by running make and make install inside of the plproxy directory. If your having problems make sure that pg_config from the postgresql bin directory is in your path.
- To install PL/Proxy in a database execute the commands in the plproxy.sql file. For example psql -f $SHAREDIR/contrib/plproxy.sql mydatabase
- Steps 1 and 2 can be skipped if your installed pl/proxy from a packaging system such as RPM.
- Create a test function to validate that plProxy is working as expected.

```
CREATE FUNCTION public.get_user_email(text) RETURNS text AS
$_$ connect 'dbname=userdb'; $_$
LANGUAGE plproxy SECURITY DEFINER;
```

plProxy Language

- The language is similar to plpgsql - string quoting, comments, semicolon at the statements end.It contains only 4 statements: CONNECT, CLUSTER, RUN and SELECT.
- Each function needs to have either CONNECT or pair of CLUSTER + RUN statements to specify where to run the function.
- **CONNECT 'libpq connstr';** -- Specifies exact location where to connect and execute the query. If several functions have same connstr, they will use same connection.
- **CLUSTER 'cluster_name';** -- Specifies exact cluster name to be run on. The cluster name will be passed to plproxy.get_cluster_* functions.
- **CLUSTER cluster_func(..);** -- Cluster name can be dynamically decided upon proxy function arguments. cluster_func should return text value of final cluster name.

plProxy Language RUN ON ...

- **RUN ON ALL;** -- Query will be run on all partitions in cluster in parallel.
- **RUN ON ANY;** -- Query will be run on random partition.
- **RUN ON <NR>;** -- Run on partition number <NR>.
- **RUN ON partition_func(..);** -- Run partition_func() which should return one or more hash values. (int4) Query will be run on tagged partitions. If more than one partition was tagged, query will be sent in parallel to them.

```
CREATE FUNCTION public.get_user_email(text) RETURNS text AS
$_$
    cluster 'userdb';
    run on public.get_hash($1);
$_$
LANGUAGE plproxy SECURITY DEFINER;
```

plProxy Configuration

- Schema **`plproxy`** and 3 functions are needed for plProxy
- **`plproxy.get_cluster_partitions(cluster_name text)`** – initializes plProxy connect strings to remote databases
- **`plproxy.get_cluster_version(cluster_name text)`** – used by plProxy to determine if configuration has changed and should be read again. Should be as fast as possible because it is called for every function call that goes through plProxy.
- **`plproxy.get_cluster_config(in cluster_name text, out key text, out val text)`** – can be used to change plProxy parameters like connection lifetime.

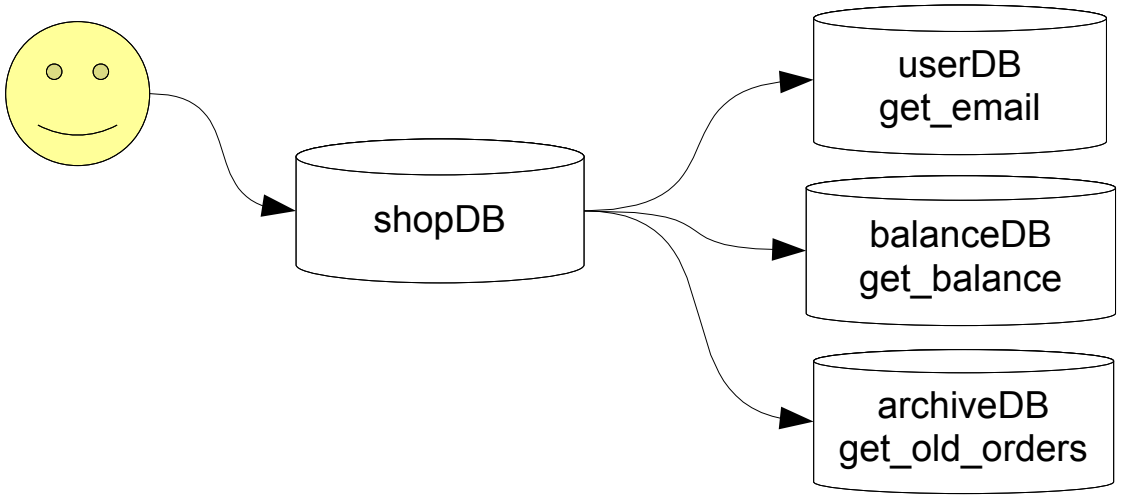
```
CREATE FUNCTION plproxy.get_cluster_version(i_cluster text)
RETURNS integer AS $$
    SELECT 1;
$$ LANGUAGE sql SECURITY DEFINER;

CREATE FUNCTION plproxy.get_cluster_config(
    cluster_name text, OUT "key" text, OUT val text)
RETURNS SETOF record AS $$
    SELECT 'connection_lifetime'::text as key, text( 30*60 ) as val;
$$ LANGUAGE sql;
```

plProxy: Get Cluster Partitions

```
CREATE FUNCTION plproxy.get_cluster_partitions(cluster_name text)
RETURNS SETOF text AS $$
begin
    if cluster_name = 'userdb' then
        return next 'port=9000 dbname=userdb_p00 user=proxy';
        return next 'port=9000 dbname=userdb_p01 user=proxy';
        return;
    end if;
    raise exception 'no such cluster: %', cluster_name;
end; $$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE FUNCTION plproxy.get_cluster_partitions(i_cluster_name text)
RETURNS SETOF text AS $$
declare
    r record;
begin
    for r in
        select connect_string
        from plproxy.conf
        where cluster_name = i_cluster_name
    loop
        return next r.connect_string;
    end loop;
    if not found then
        raise exception 'no such cluster: %', i_cluster_name;
    end if;
    return;
end; $$ LANGUAGE plpgsql SECURITY DEFINER;
```

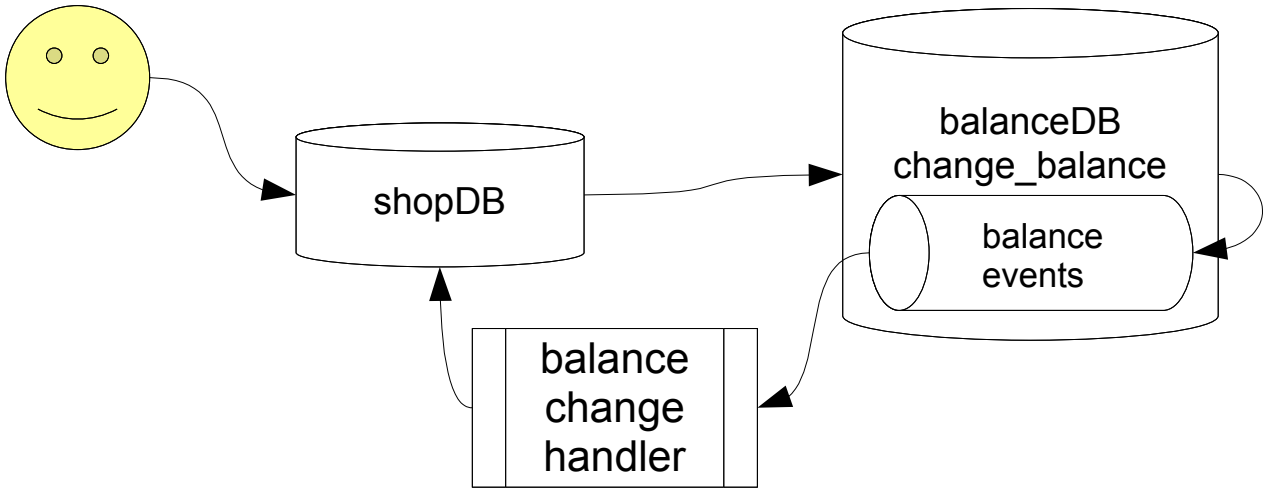
plProxy: Remote Calls

- We use remote calls mostly for read only queries in cases where it is not reasonable to replicate data needed to calling database.
- For example balance data is changing very often but whenever doing decisions based on balance we must use the latest balance so we use remote call to get user balance.
- Another use case when occasionally archived data is needed together with online data.



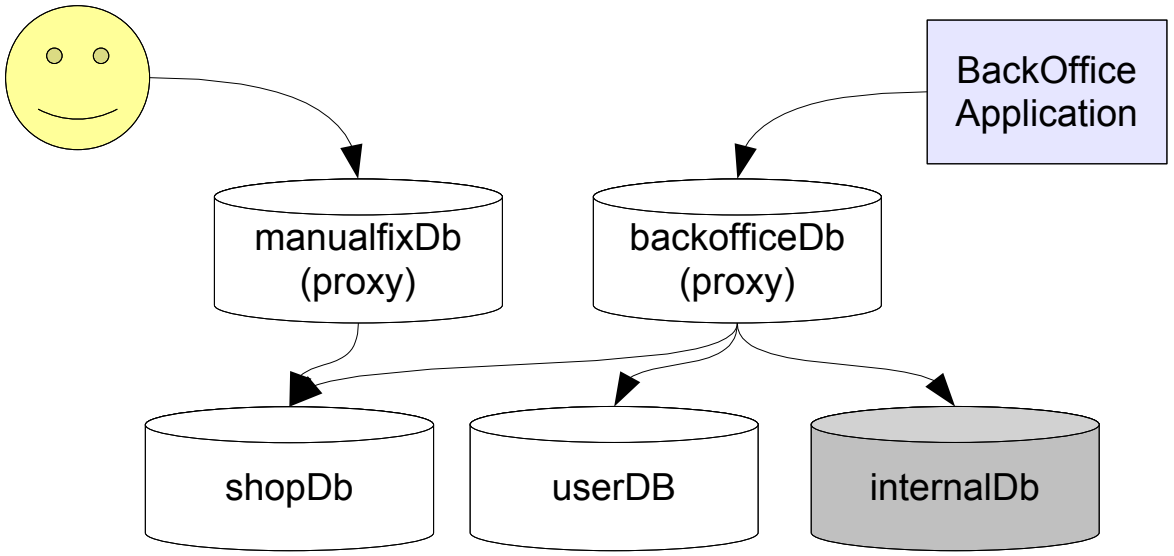
plProxy: Remote Calls (update)

- plProxy remote calls inside transactions that change data in remote database should have special handling
- no 2 phase commit
- some mechanism should be used to handle possible problems like inserting events into PgQ queue and let consumer validate that transaction was committed or rolled back and act accordingly.



plProxy: Proxy Databases

- Additional layer between application and databases.
- Keep applications database connectivity simpler giving DBA's and developer's more flexibility for moving data and functionality around.
- Security layer. By giving access to proxy database DBA's can be sure that user has no way of accessing tables by accident or by any other means as only functions published in proxy database are visible to user.



plProxy: Run On All

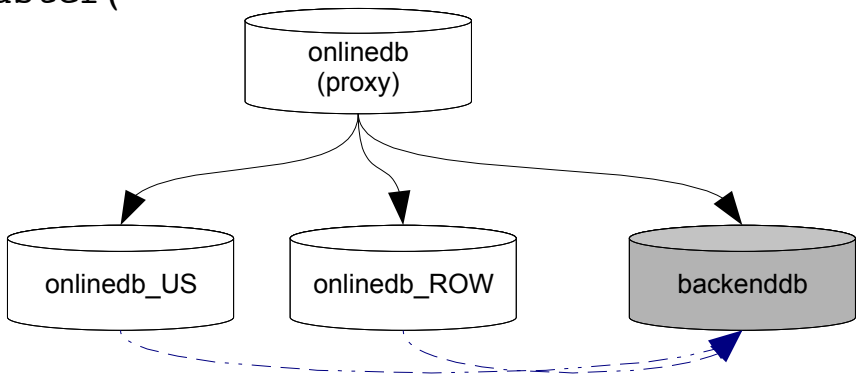
- Run on all executes query on all partitions in cluster once. Partitions are identified by connect strings.
- Useful for gathering stats from several databases or database partitions.
- Also usable when exact partition where data resides is not known. Then function may be run on all partitions and only the one that has data does something.

```
CREATE FUNCTION stats._get_stats(
    OUT stat_name text,
    OUT stat_value bigint
    ) RETURNS SETOF record AS
$_$
    cluster 'userdb';
    run on all;
$_$
    LANGUAGE plproxy SECURITY DEFINER;

CREATE FUNCTION stats.get_stats(
    OUT stat_name text,
    OUT stat_value bigint
    ) RETURNS SETOF record AS
$_$
    select stat_name
    , (sum(stat_value))::bigint
    from stats._get_stats()
    group by stat_name
    order by stat_name;
$_$
    LANGUAGE sql SECURITY DEFINER;
```

plProxy: Geographical

- plProxy can be used to split database into partitions based on country code. Example database is split into 'us' and 'row' (rest of the world)
- Each function call caused by online users has country code as one of the parameters
- All data is replicated into internal database for use by internal applications and batch jobs. That also reduces number of indexes needed in online databases.

```
CREATE FUNCTION public.get_cluster(
    i_key_cc text
    ) RETURNS text AS
$_$
BEGIN
    IF i_key_cc = 'us' THEN
        RETURN 'oltp_us';
    ELSE
        RETURN 'oltp_row';
    END IF;
END;
$_$ LANGUAGE plpgsql;
```



plProxy: Partitioning Proxy Functions

- We have partitioned most of our database by username using PostgreSQL hashtext function to get equal distribution between partitions.
- When splitting databases we usually prepare new partitions in other servers and then switch all traffic at once to keep our life pleasant.
- Multiple exact copies of proxy database are in use for scaleability and availability considerations.

```
CREATE FUNCTION public.get_user_email(text) RETURNS text AS
$_$
    cluster 'userdb';
    run on public.get_hash($1);
$_$ LANGUAGE plproxy SECURITY DEFINER;

CREATE FUNCTION public.get_hash(i_user text) RETURNS integer AS
$_$
BEGIN
    return hashtext(lower(i_user));
END;
$_$ LANGUAGE plpgsql SECURITY DEFINER;
```

plProxy: Partitioning Partition Functions

- Couple of functions in `partconf` schema added to each partition:
  - `partconf.global_id()` - gives globally unique keys
  - `partconf.check_hash()` - checks that function call is in right partition
  - `partconf.valid_hash()` - used as trigger function

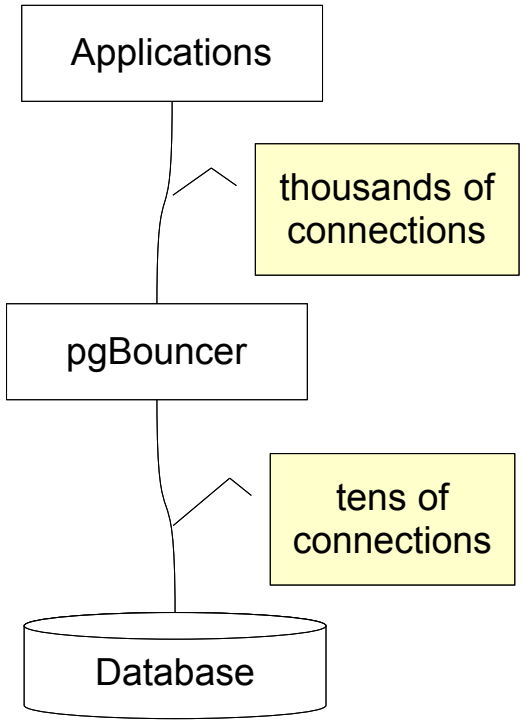
```
CREATE FUNCTION public.get_user_email(i_username text) RETURNS text AS
$_$
DECLARE
    retval text;
BEGIN
    PERFORM partconf.check_hash(lower(i_username));

    SELECT email
    FROM users
    WHERE username = lower(i_username)
    INTO retval;

    RETURN retval;
END;
$_$ LANGUAGE plpgsql SECURITY DEFINER;
```

plProxy: Summary

- plProxy adds 1-10ms overhead when used together with pgBouncer.
- Quote from Gavin M. Roy's blog “After closely watching machine stats for the first 30 minutes of production, it appears that plProxy has very little if any impact on machine resources in our infrastructure.”
- On the other hand plProxy adds complexity to development and maintenance so it must be used with care but that is true for most everything.
- Our largest cluster is currently running in 16 partitions on 16 servers.

pgBouncer

- pgBouncer is lightweight and robust connection pooler for Postgres.
- Low memory requirements (2k per connection by default). This is due to the fact that PgBouncer does not need to see full packet at once.
- It is not tied to one backend server, the destination databases can reside on different hosts.
- Supports pausing activity on all or only selected databases.
- Supports online reconfiguration for most of the settings.
- Supports online restart/upgrade without dropping client connections.
- Supports protocol V3 only, so backend version must be >= 7.4.
- Does not parse SQL so is very fast and uses little CPU time.

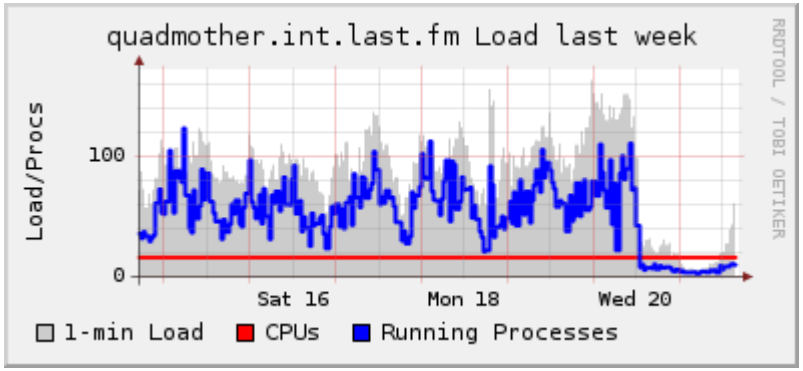


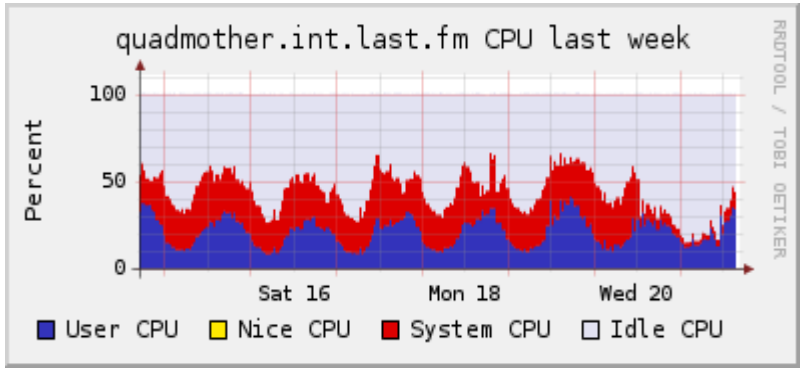
pgBouncer Pooling Modes

- **Session pooling** - Most polite method. When client connects, a server connection will be assigned to it for the whole duration it stays connected. When client disconnects, the server connection will be put back into pool. Should be used with legacy applications that won't work with more efficient pooling modes.
- **Transaction pooling** - Server connection is assigned to client only during a transaction. When PgBouncer notices that transaction is over, the server will be put back into pool. This is a hack as it breaks application expectations of backend connection. You can use it only when application cooperates with such usage by not using features that can break.
- **Statement pooling** - Most aggressive method. This is transaction pooling with a twist - multi-statement transactions are disallowed. This is meant to enforce "autocommit" mode on client, mostly targeted for PL/Proxy.

pgBouncer Pictures

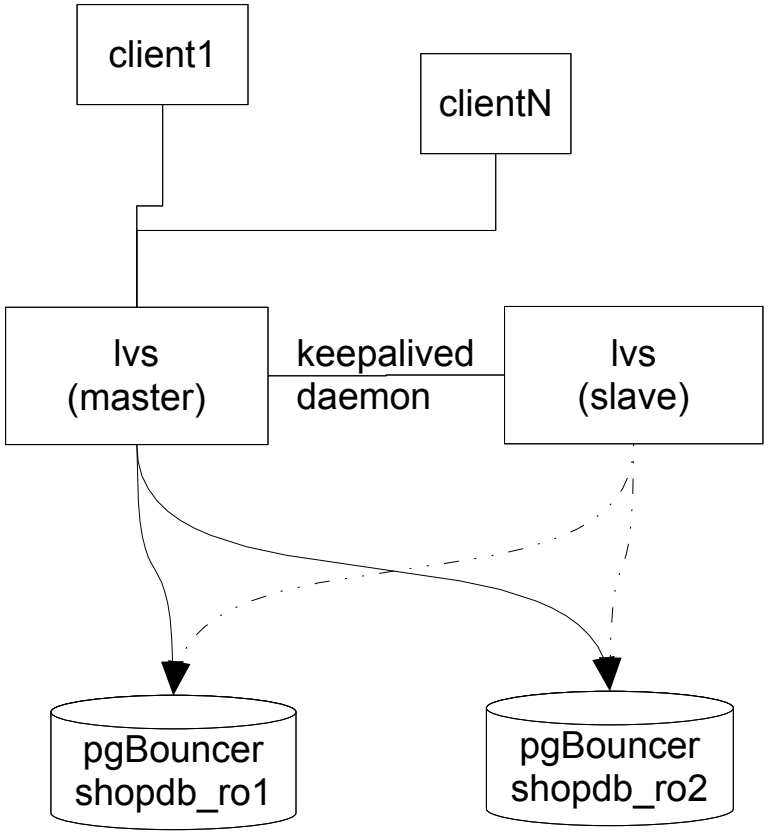
- http://www.last.fm/user/Russ/journal/2008/02/21/654597/
- Nice blog about pgBouncer





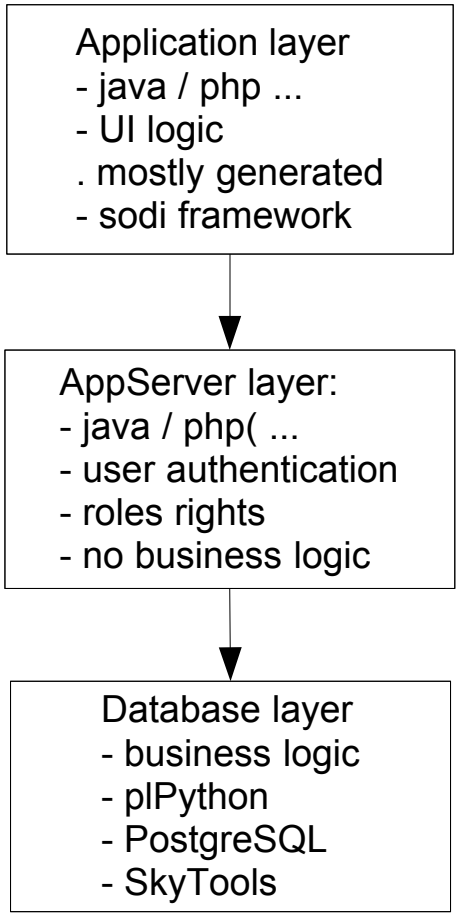
pgBalancer

- Keep pgBouncer small stable and focused
- pgBalancer is experiment with existing software to implement statement level load balancing.
- So we created sample configuration using existing software that shows how load could be divided on several read only databases.
- LVS – Linux Virtual Server is used (http://www.linuxvirtualserver.org/)
  - Different load balancing algorithms
  - Weights on resources.

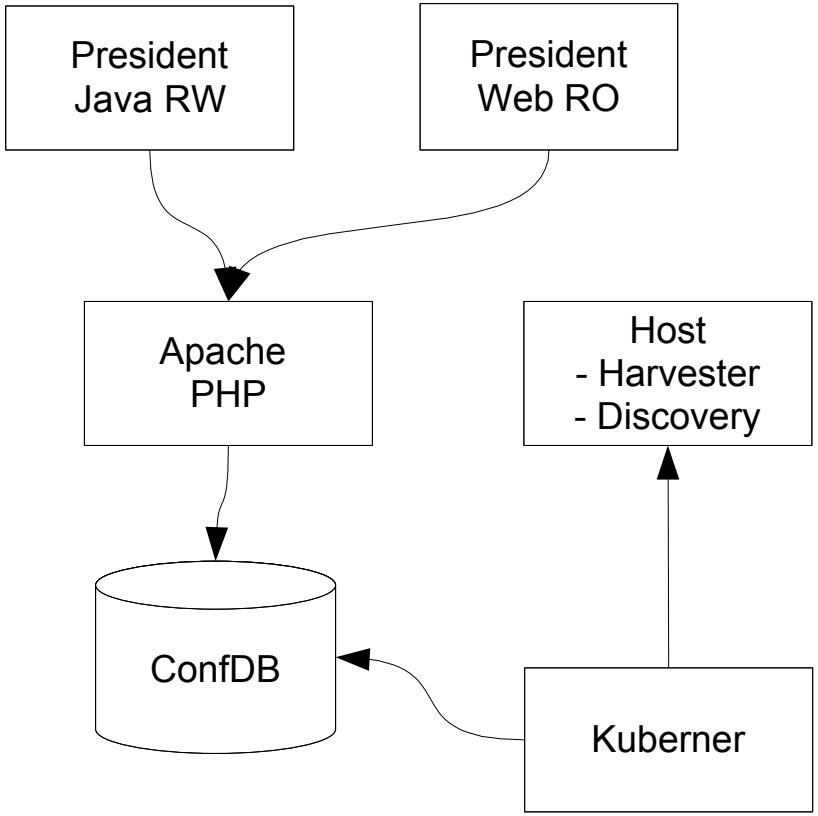


SODI Framework

- Rapid application development
- Cheap changes
- Most of the design is in metadata.
- Application design stays in sync with application over time.
- Minimizes number of database roundtrips.
- Can send multiple recordsets to one function call.
- Can receive several recordsets from function call.



President



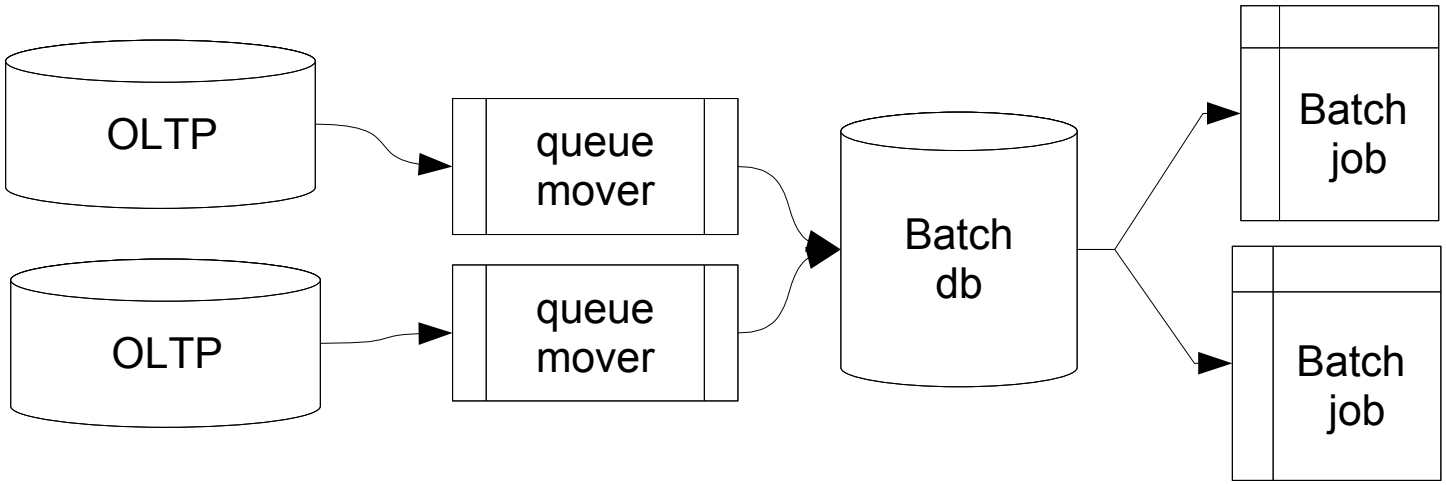
- President configuration management application
- Java Client allows updates Web client view only access. All outside access to system over HTTPS and personal certificates can be generated if needed.
- Kuberner polls confdb periodically for commands and executes received commands.
- Harvester runs inside each host and writes machine hardware information into file for Kuberner to read
- Discovery plugin collects specific configuration files and stores them into ConfDB
- Kuberner: File upload plugin uploads new configuration files into hosts.

SkyTools

- SkyTools - Python scripting framework and collection of useful database scripts. Most of our internal tools are based on this framework.
  - (centralized) logging and exception handling
  - database connection handling
  - configuration management
  - starting and stopping scripts
- Some of scripts provided by SkyTools
  - londiste – nice and simple replication tool
  - walmgr - wal standby management script
  - serial consumer – Script for executing functions based on data in queue
  - queue mover – Move events from one queue into another
  - queue splitter – Move events from one queue into several queues
  - table dispatcher – writes data from queue into partitioned table
  - cube dispatcher - writes data from queue into daily tables

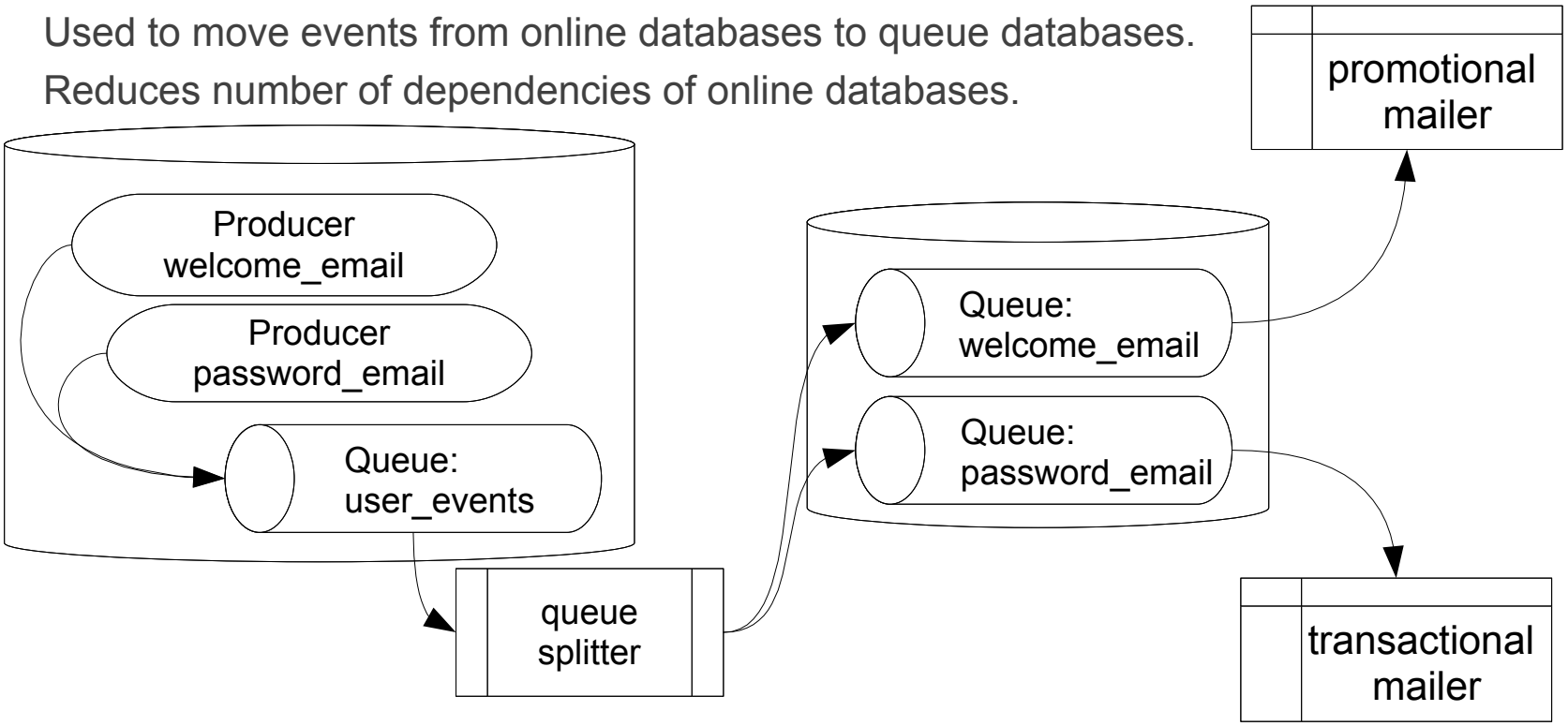
SkyTools: Queue Mover

- Moves data from source queue in one database to another queue in other database.
- Used to move events from online databases to queue databases.
- We don't need to keep events in online database in case some consumer fails to process them.
- Consolidates events if there are several producers as in case of partitioned databases.



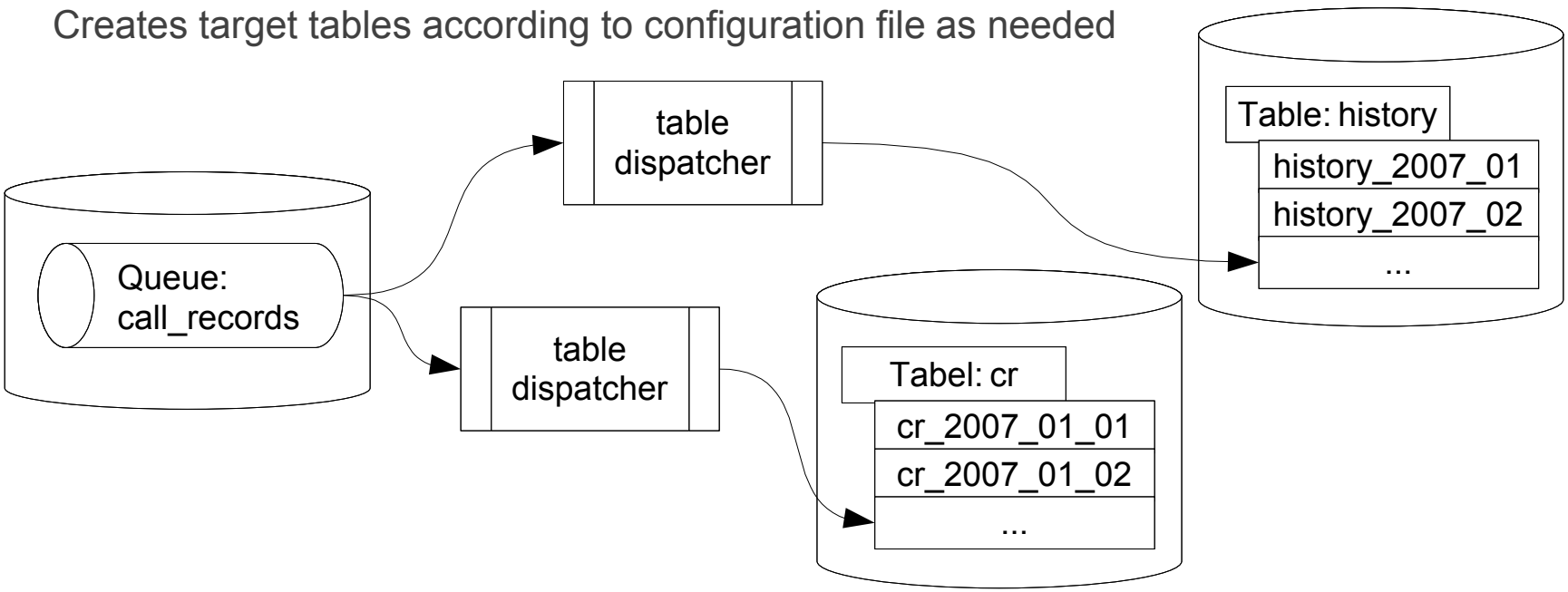
SkyTools: Queue Splitter

- Moves data from source queue in one database to one or more queue's in target database based on producer. That is another version of queue_mover but it has it's own benefits.
- Used to move events from online databases to queue databases.
- Reduces number of dependencies of online databases.



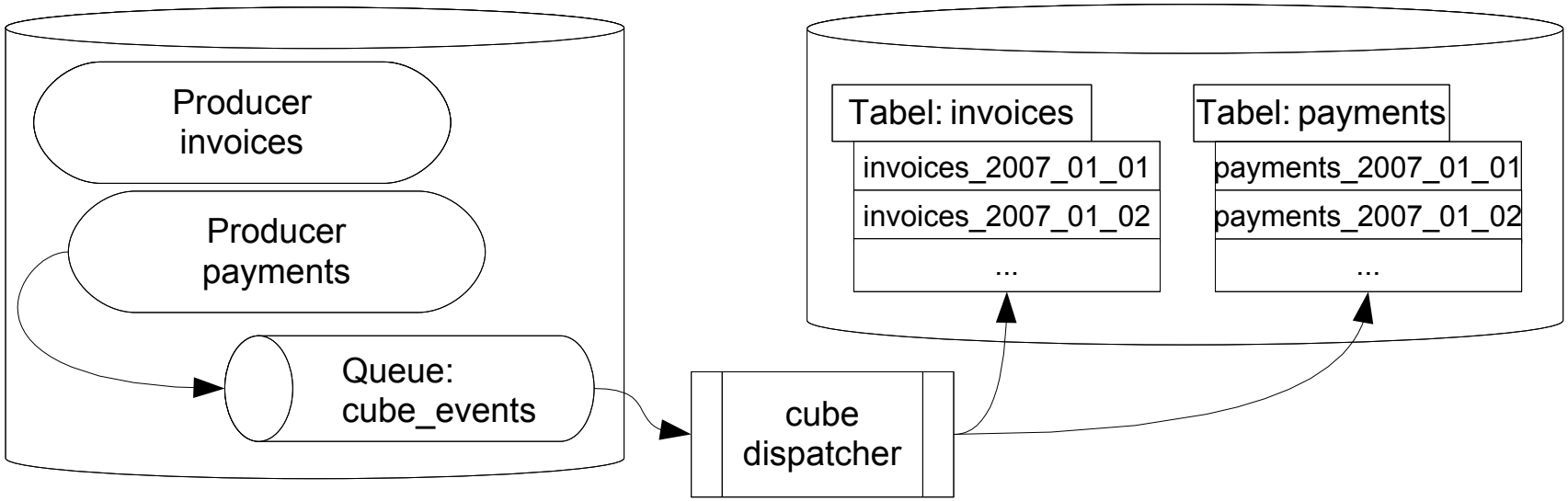
SkyTools: Table Dispatcher

- Has url encoded events as data source and writes them into table on target database.
- Used to partiton data. For example change log's that need to kept online only shortly can be written to daily tables and then dropped as they become irrelevant.
- Also allows to select which columns have to be written into target database
- Creates target tables according to configuration file as needed



SkyTools: Cube Dispatcher

- Has url encoded events as data source and writes them into partitoned tables in target database. Logutriga is used to create events.
- Used to provide batches of data for business intelligence and data cubes.
- Only one instance of each record is stored. For example if record is created and then updated twice only latest version of record stays in that days table.
- Does not support deletes.



Questions, Problems?

- Use SkyTools, plProxy and pgBouncer mailing list at PgFoundry
- skype me: askoja