Database Development in Skype

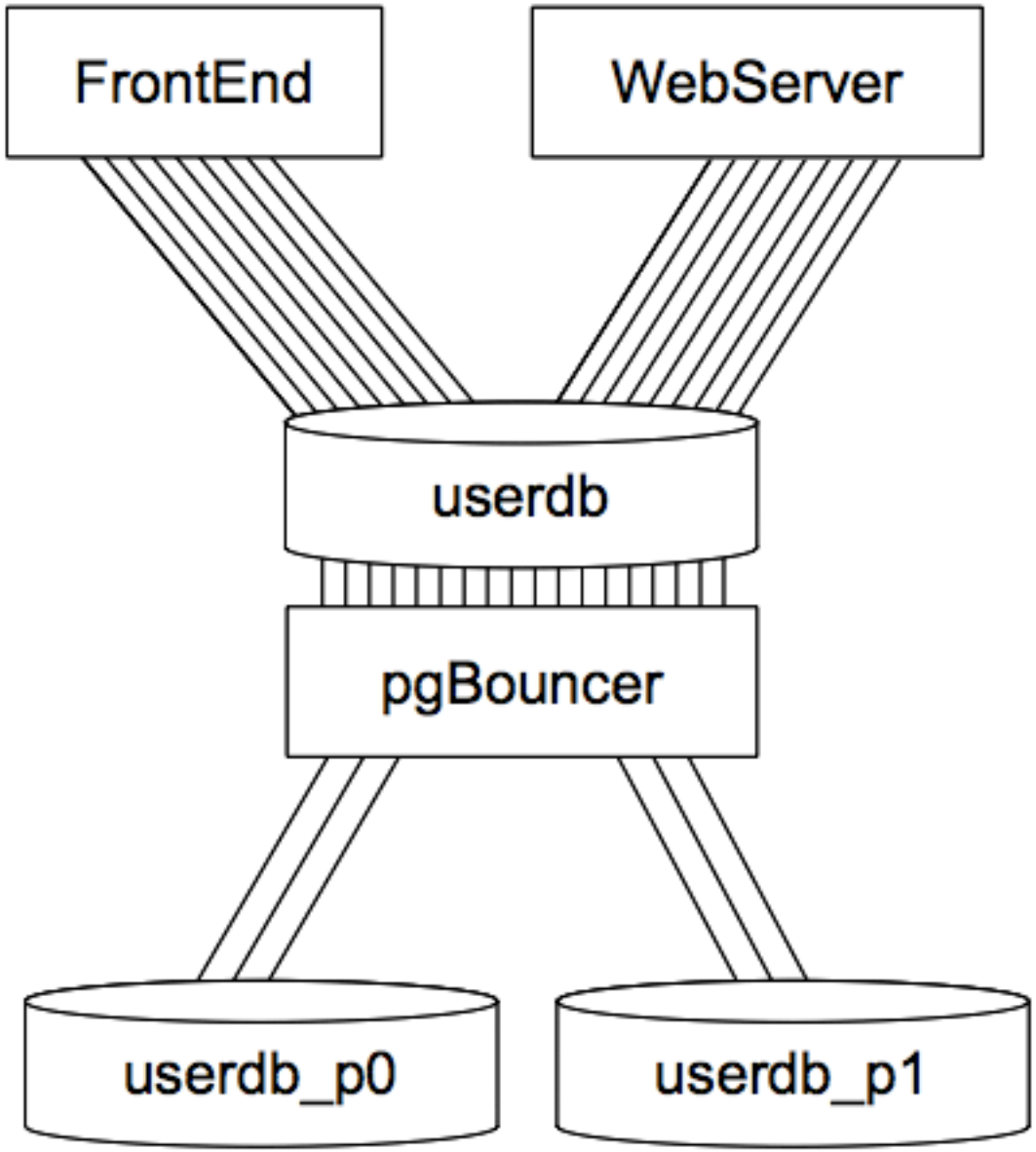
Kristo Kaiv 2010

(hi)

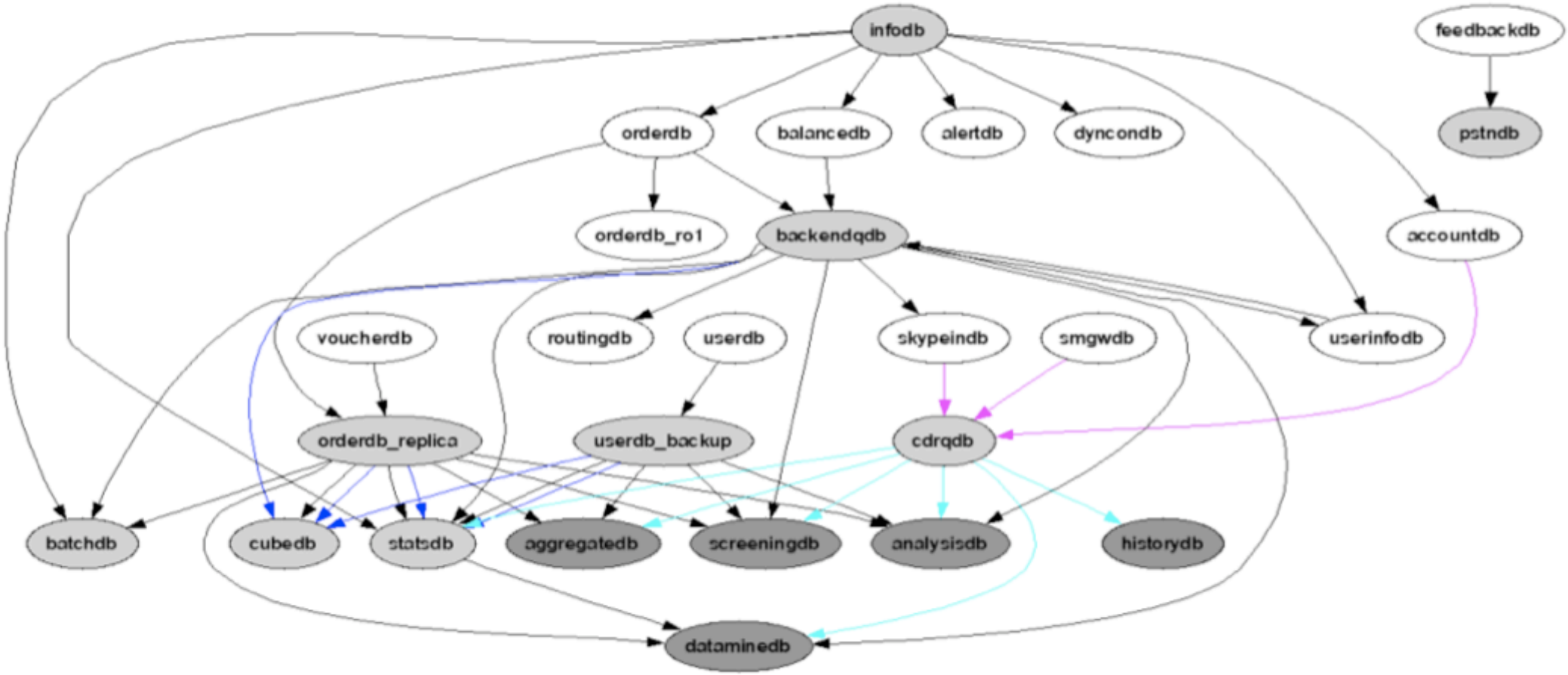
The wise man learns from others mistakes, the fool learns from his own.

What we use

- PostgreSQL except for out-of-the-box apps
- plpgsql and a bit of plpython here and there
- for scripts python, skytools framework
- plproxy
- president



Flying Spaghetti monster



KISS

- data accessed only via function calls
- no dynamic SQL
- everything is autocommit
- access rights controlled per function
- no business logic inside trigger
- A4 rule
- Pareto’s principle applies nicely to scripts
- Why? instead of what?

KISS

- Representing physical structure in code repository is a bad idea
  - do it separately
- Vertical splitting is often a can of worms

Sharding

- for provisioning & billing sharding is easy and cheap to implement
- no cost for hardware and almost none for software development
- networks are hard to shard and in-memory cluster might be the best choice
- full 2-phase commit is too costly
- if things can be done asynchronously then they should
- master write - slave read is more complicated than it sounds

Scalability

- tenfold cost for making existing systems scalable
- dynamic SQL means moving your costs from software to hardware
- replication & failover needs to be considered from the beginning
- overhead from keeping components decoupled

Proxy layer

- decoupling of physical architecture from logical
- physical architecture is “documented” in one place and not scattered around different application configuration files
- plproxy or plpgsql functions
- easy to scale, good for CPU intensive tasks
- stateless, you are not guaranteed to arrive at the same proxy twice
- data can be stored temporarily inside a transaction

Testing

- No unit-tests, only flow tests
- tests are the only documentation that is always up to date
- nose framework, python
- most requirements in a good spec can be covered by a flow test
- extensive regression testing catches what analysts don’t
- parallel to development
- time dimension makes things complex

Deployment

- hundreds of servers, multiple colocations
- release is represented by release index consisting of release items
- release item grants access, deploys function, deploys proxies etc.
- tens of files created on the fly based on deployment hints
- developers know about logical structure, DBA's keep configuration of physical structure

Deployment

-- database: accountdb_partition (plproxy, remote_accessproxydb, log_event_enricher)
-- proxy: balancedb[CLUSTER 'accountdb'; run on hashtext($1);] (***lots of grants here***)
-- proxy: webproxydb[CLUSTER 'balancedb_proxy'; RUN ON ANY;] (webstore)
-- proxy: boproxydb[CLUSTER 'balancedb_proxy'; RUN ON ANY;] (backoffice)
-- proxy: frauddb[CLUSTER 'balancedb_proxy'; RUN ON ANY;] (fraudtool)

Environments

- development, pre-QA x N, integration testing
- commodity hardware is cheap, mirror & backup everything
- tear-down & build-up daily or weekly
  - binary copy from an earlier version (integration testing)
  - deploy all release items in proper order
  - no code will be left out from the release item
  - eliminates many dependency conflicts
  - complicated in case of external partners

jobs@skype.net

subject: DB-DEV-CZ

jobs.skype.com

Thanks for listening