How WhatsApp Moved 1.5B Users Across Data Centers

Igors Istocniks

Code BEAM SF 2019

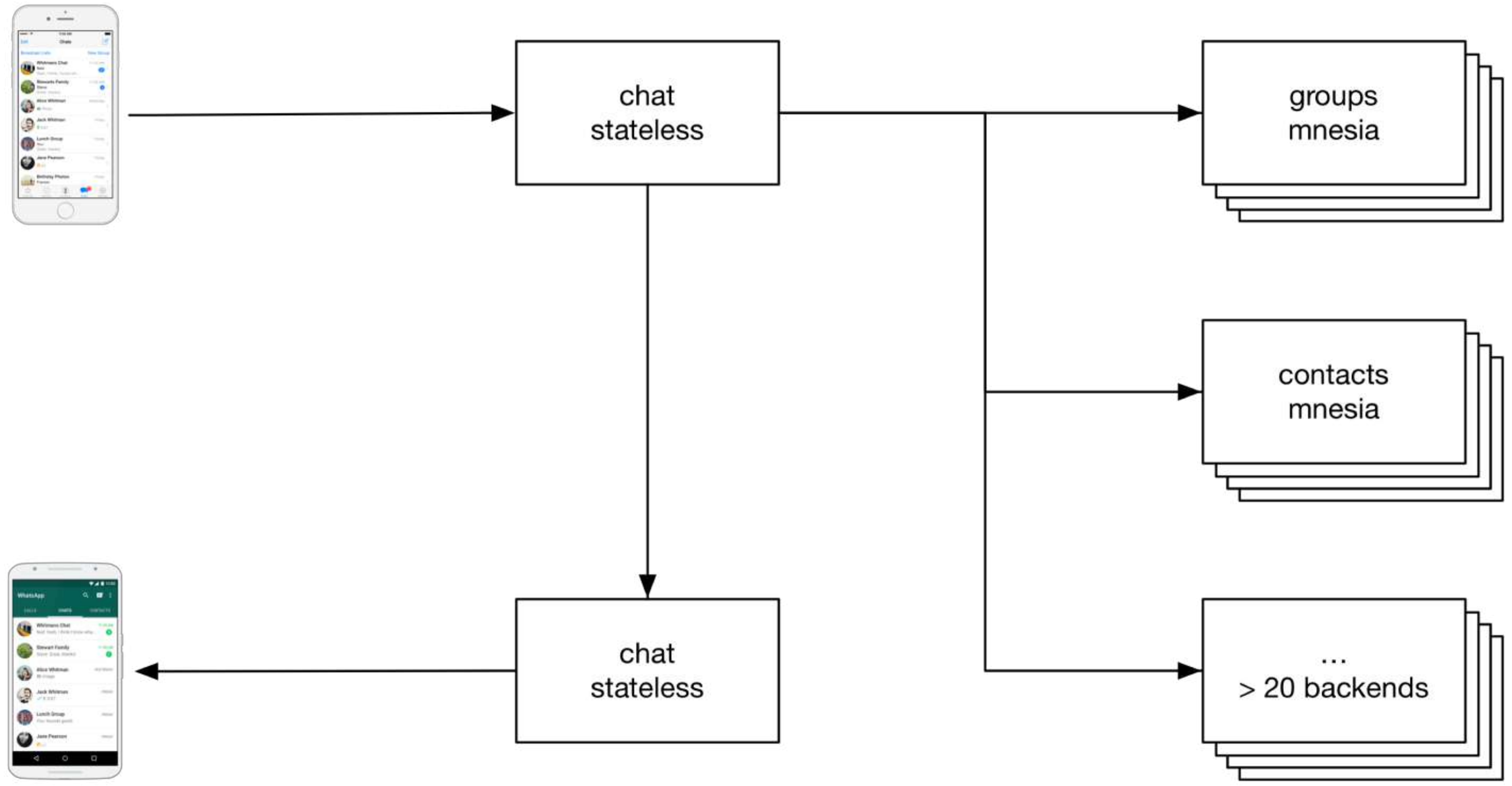
WhatsApp Migration to FB Infra

- FreeBSD → Linux
- Erlang R16 → R21
- bare metal → containers
- manual ops → automation
- local storage → hosted DB services
- …

WhatsApp Migration to FB Infra

- FreeBSD → Linux
- Erlang R16 → R21
- bare metal → containers
- manual ops → automation
- <u>local storage → hosted DB services</u>
- …

Server Architecture



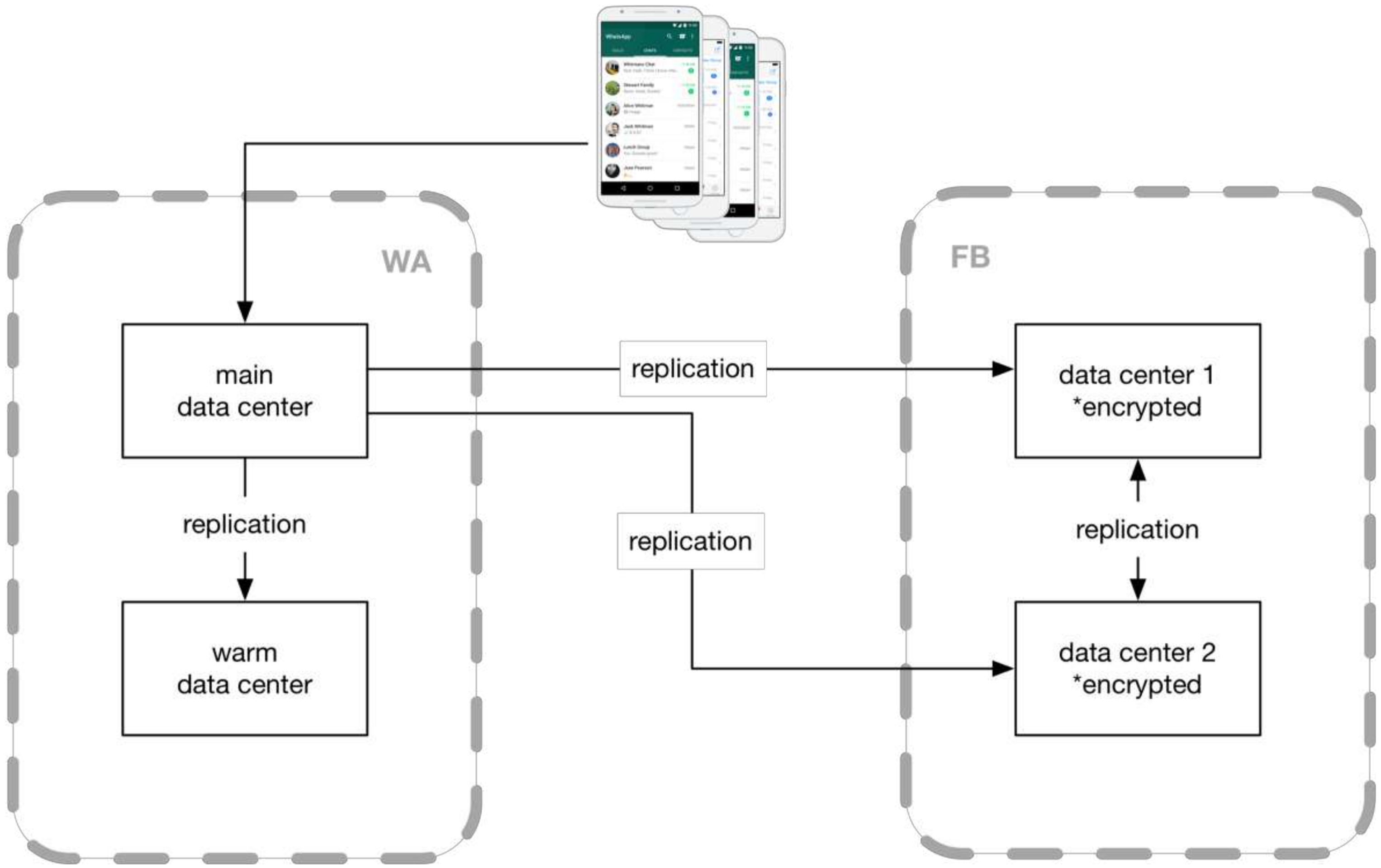
Server Architecture

- 20+ backends
  - relationship: groups, blocklist
  - key-value: device platform, profile picture
  - miscellaneous: public encryption keys
- stateful
  - mnesia
  - custom storage
- single data center
  - warm copy

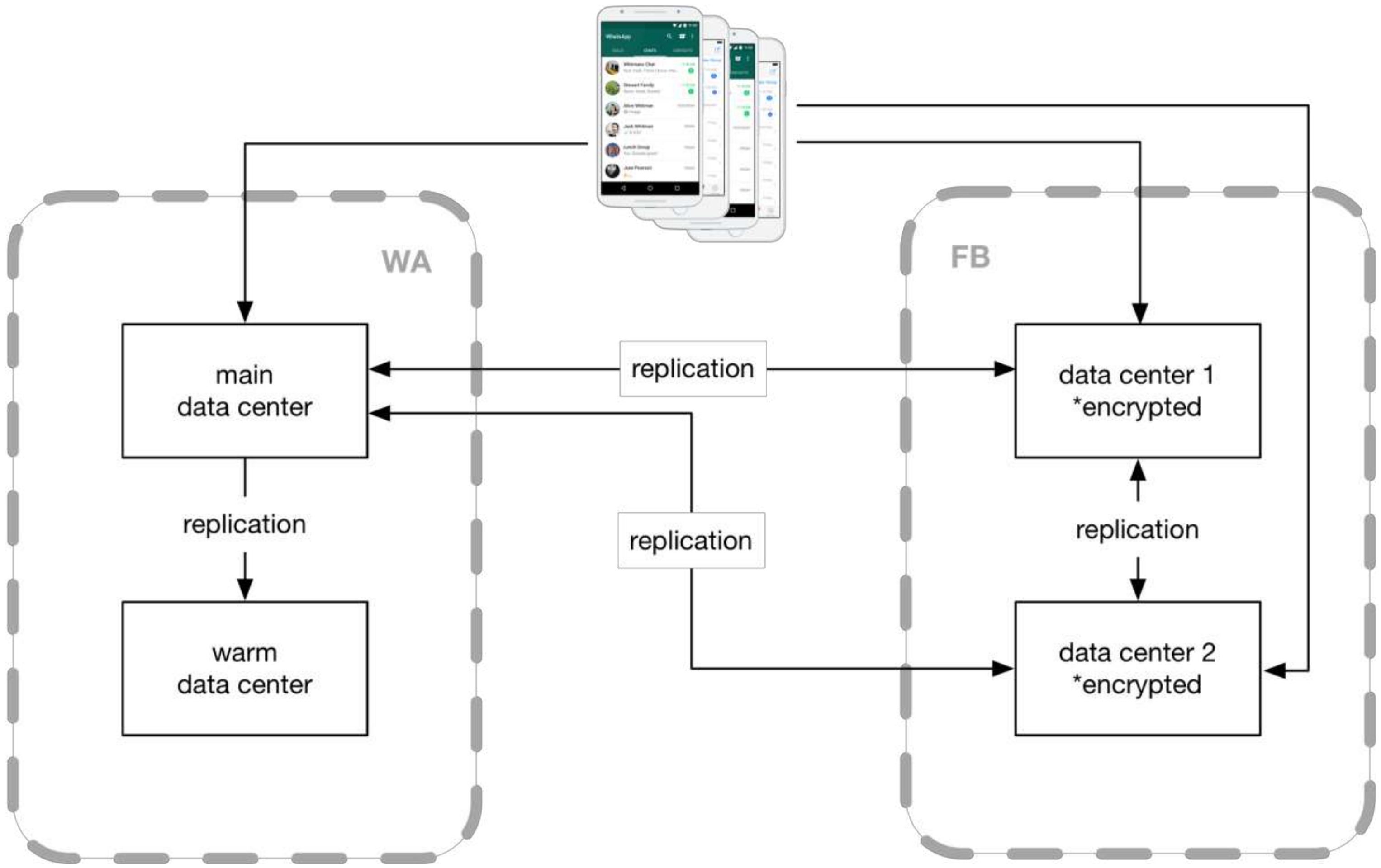
Reasons to Migrate

- better integration with FB infra
  - deploy, testing, monitoring, automation
  - leveraging internal FB systems
  - knowledge/technology sharing
- hot-hot multi data center
  - WhatsApp
    - ForgETS: drop-in Mnesia replacement (Code BEAM STO 2018)
  - Facebook
    - TAO: graph (Data @Scale 2013)
    - ZippyDB: key-value (Data @Scale 2015)

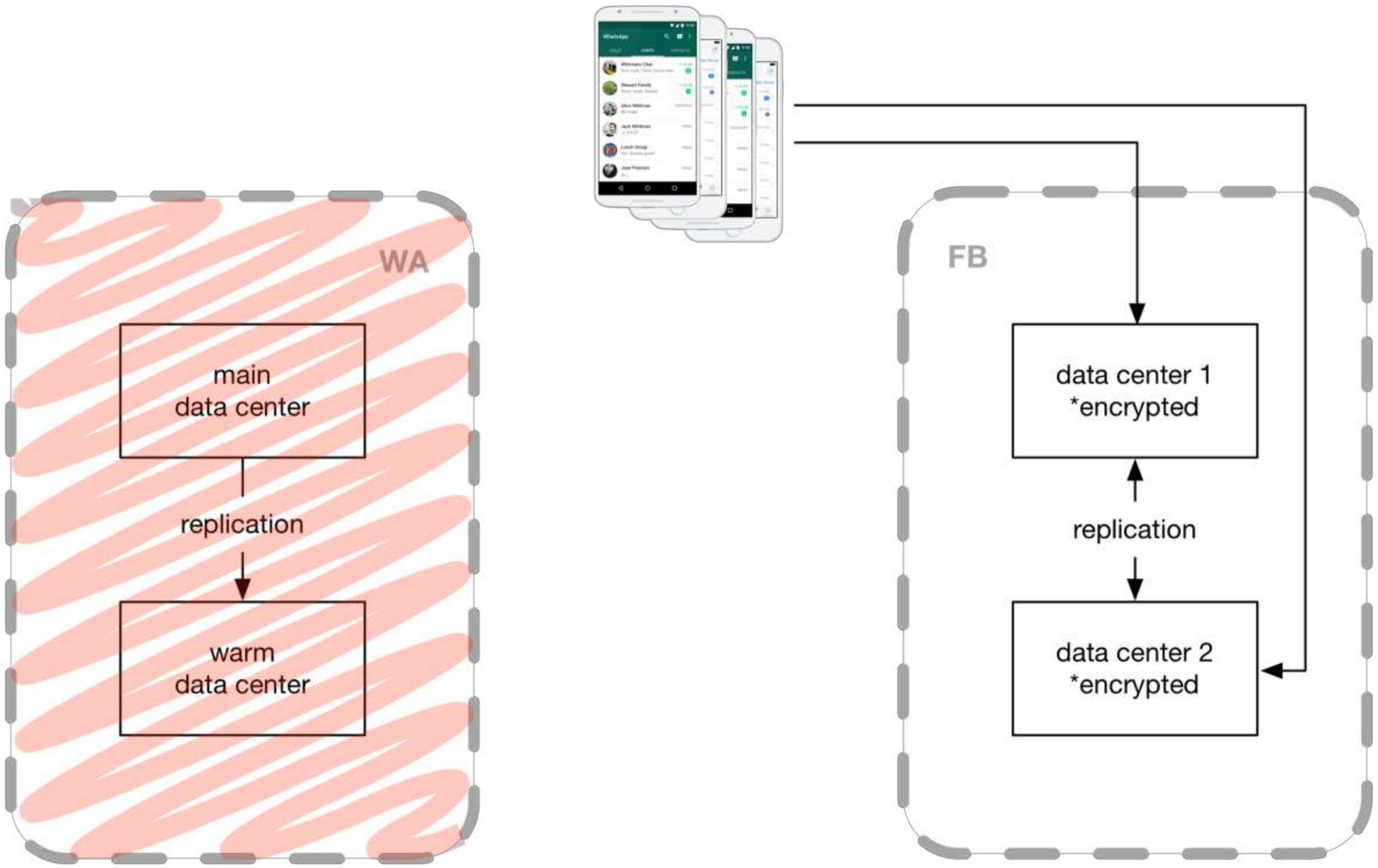
Migration Plan



Migration Plan

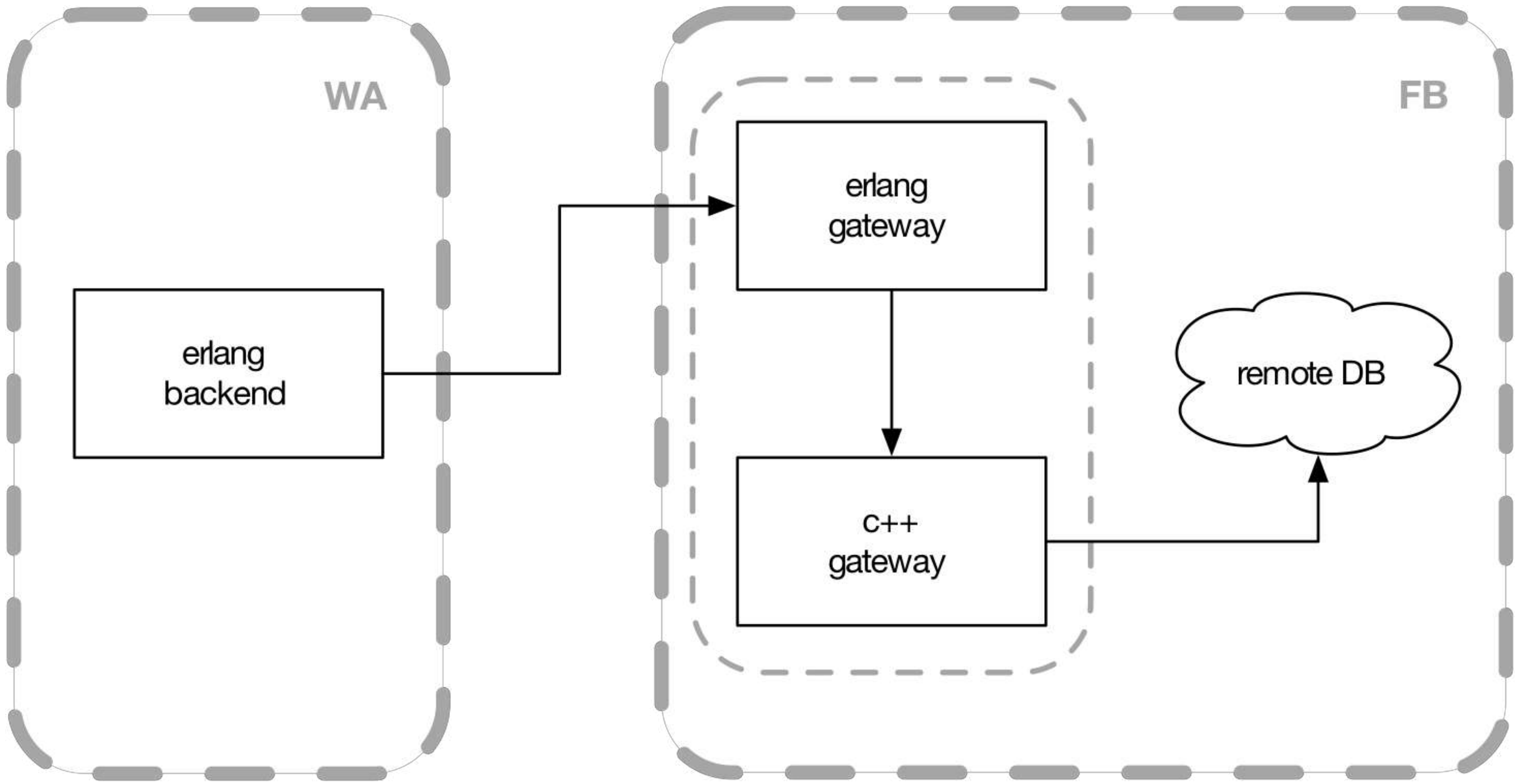


Migration Plan



Erlang → C++ Communication

Erlang gateway



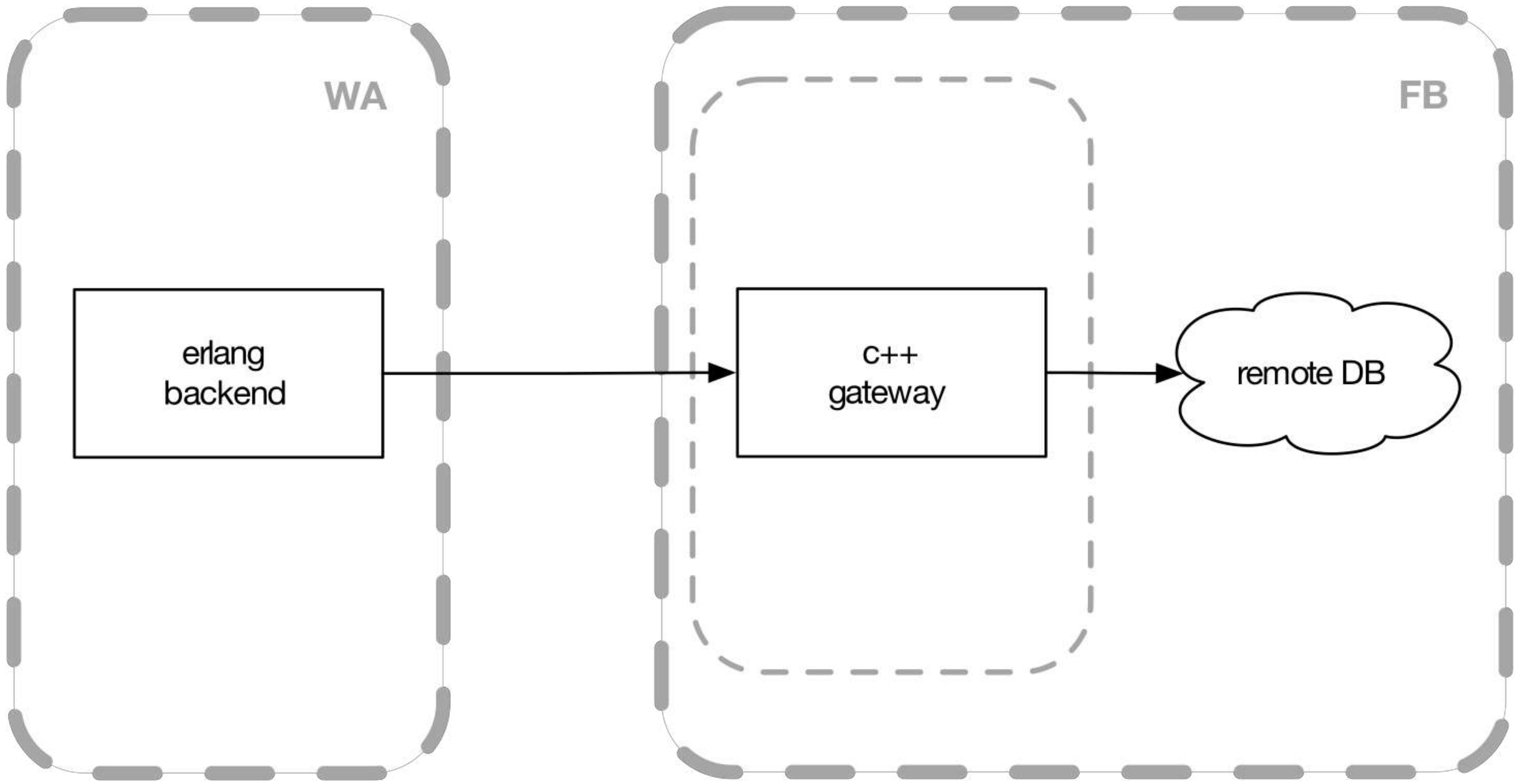
Erlang → C++ Communication

Erlang gateway learnings (R16)

- ports are very fast and easy to use
  - one c++ process per port – memory and CPU waste
- gen_tcp is nice but slower
  - by default max 1 outstanding request per worker
  - single event loop in beam that contends on a single global lock (PollSet)
- gen_socket supports PollSet per scheduler
  - https://github.com/alibaba/erlang_multi_pollset
  - lower single thread perf, no improvements under load
- gen_tcp + multiplexing / pipelining via ref() works
  - HOWEVER very hard to tune Erlang & C++ on one machine
  - erlang node does nothing but message forwarding

Erlang → C++ Communication

C++ gateway



Erlang → C++ Communication

C++ gateway learnings

- no need to tune 2 processes on one machine
- no erlang node that does just message forwarding
- more complex technology stack
- custom TCP based protocol
    - listen & connect modes
- request load balancer
    - power-of-two algorithm
- thrift serialization
    - #records{} + code hot load

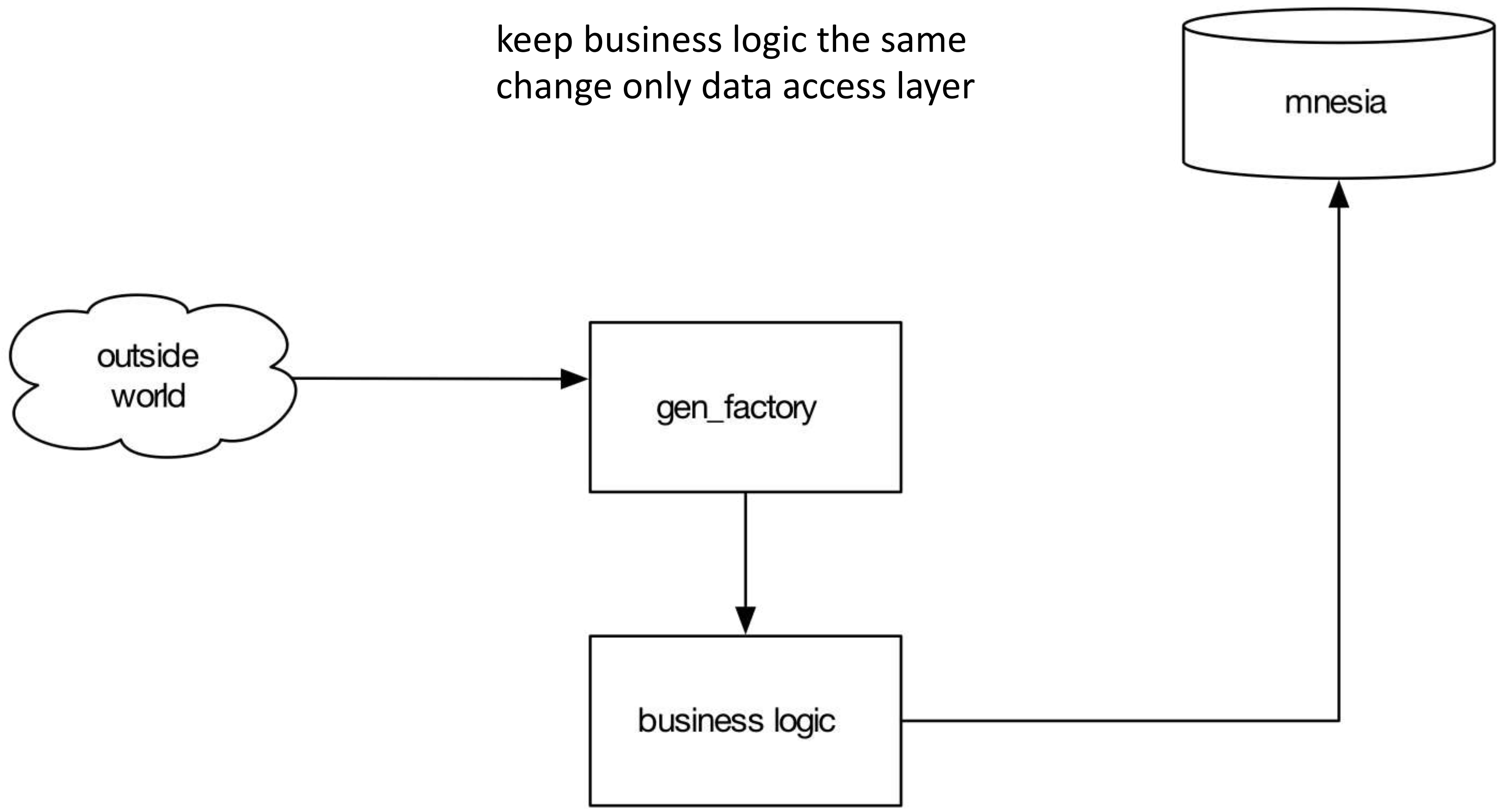
Migration Preparation

gen_factory

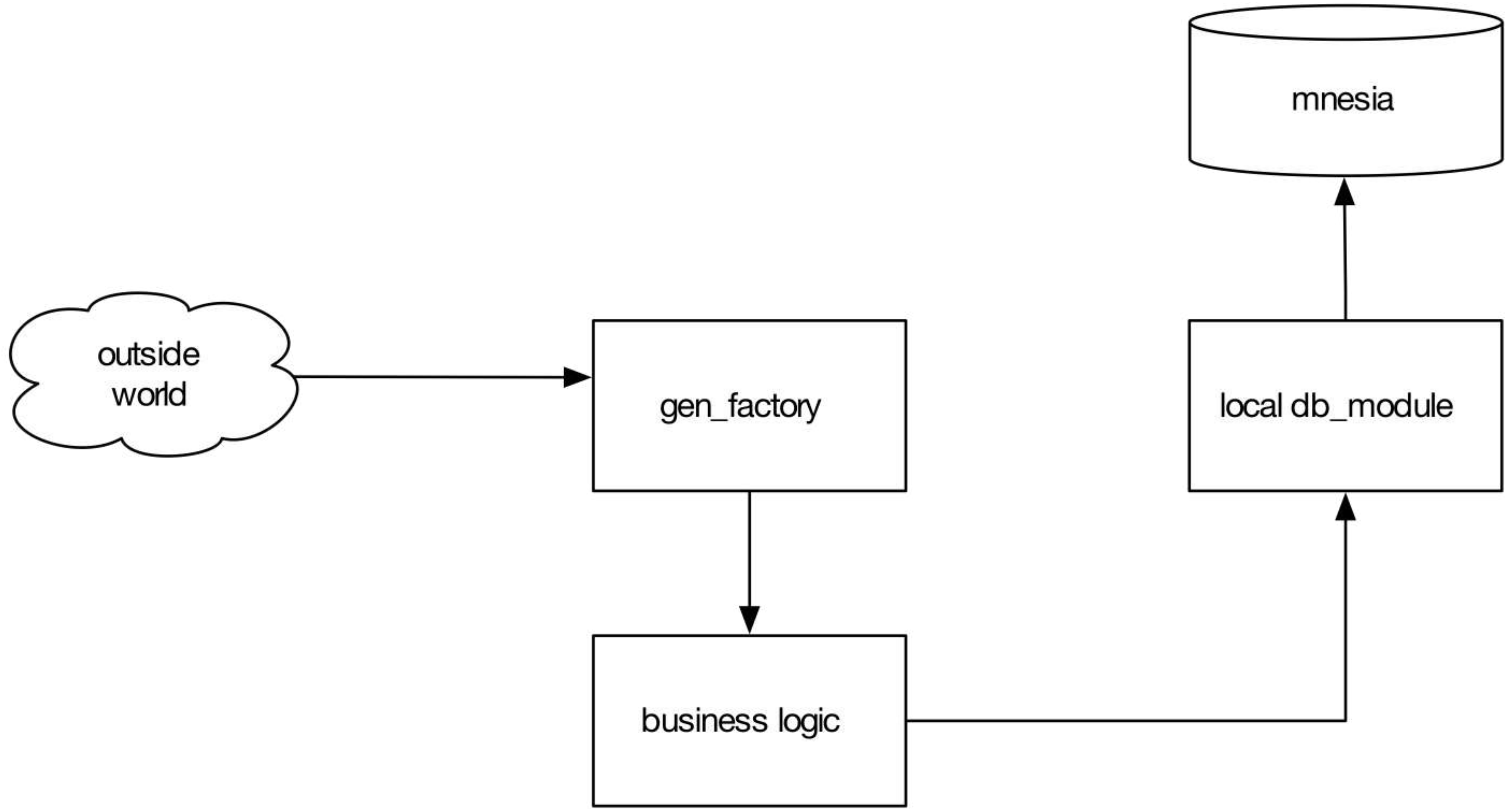
- Erlang Factory SF 2014
- group of gen_servers
- dispatcher & workers
- configurable routing
    - consistent hashing
    - queue
    - sticky queue

Migration Preparation
goals

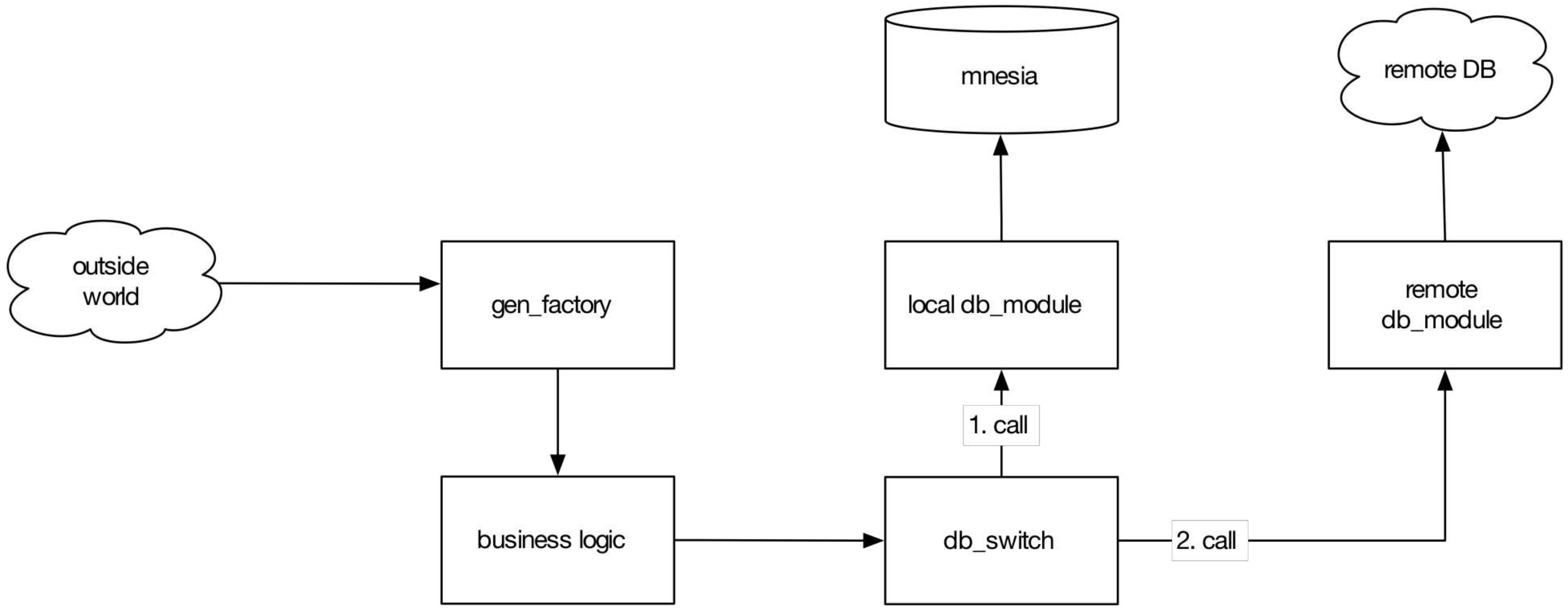
keep business logic the same
change only data access layer



Migration Preparation
db_module

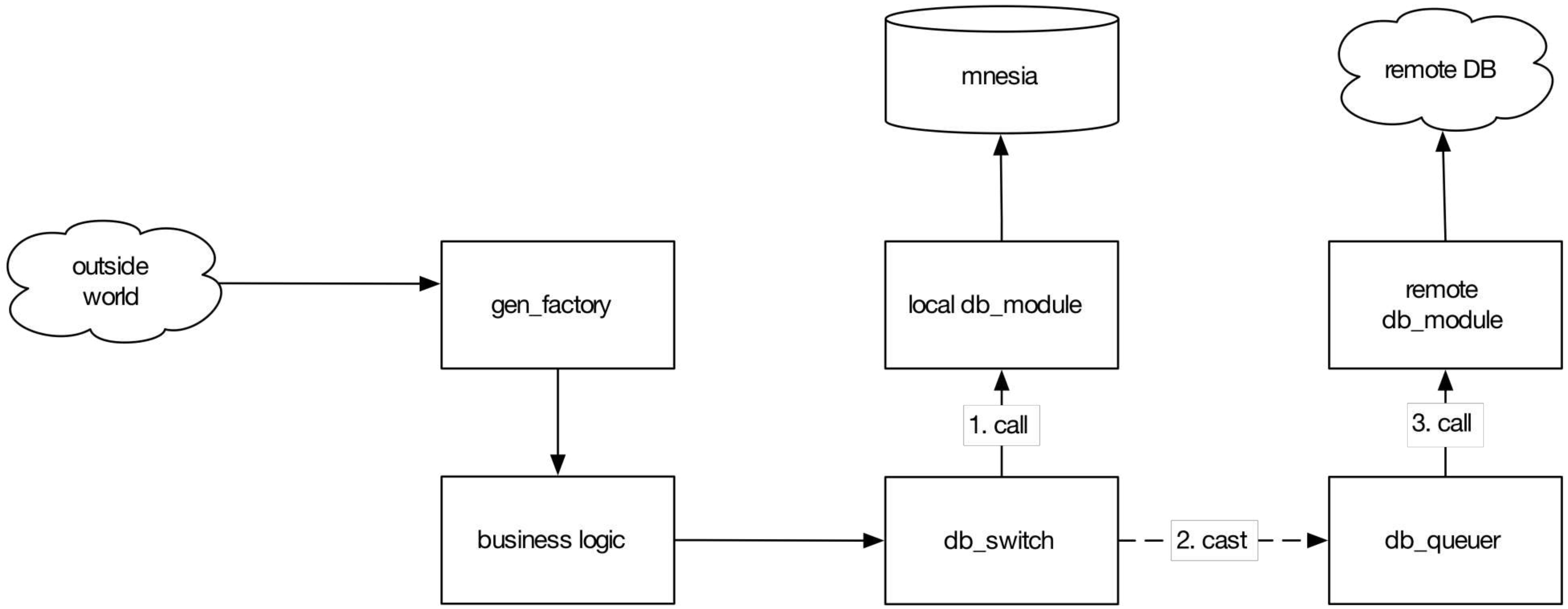


Migration Preparation
db_switch



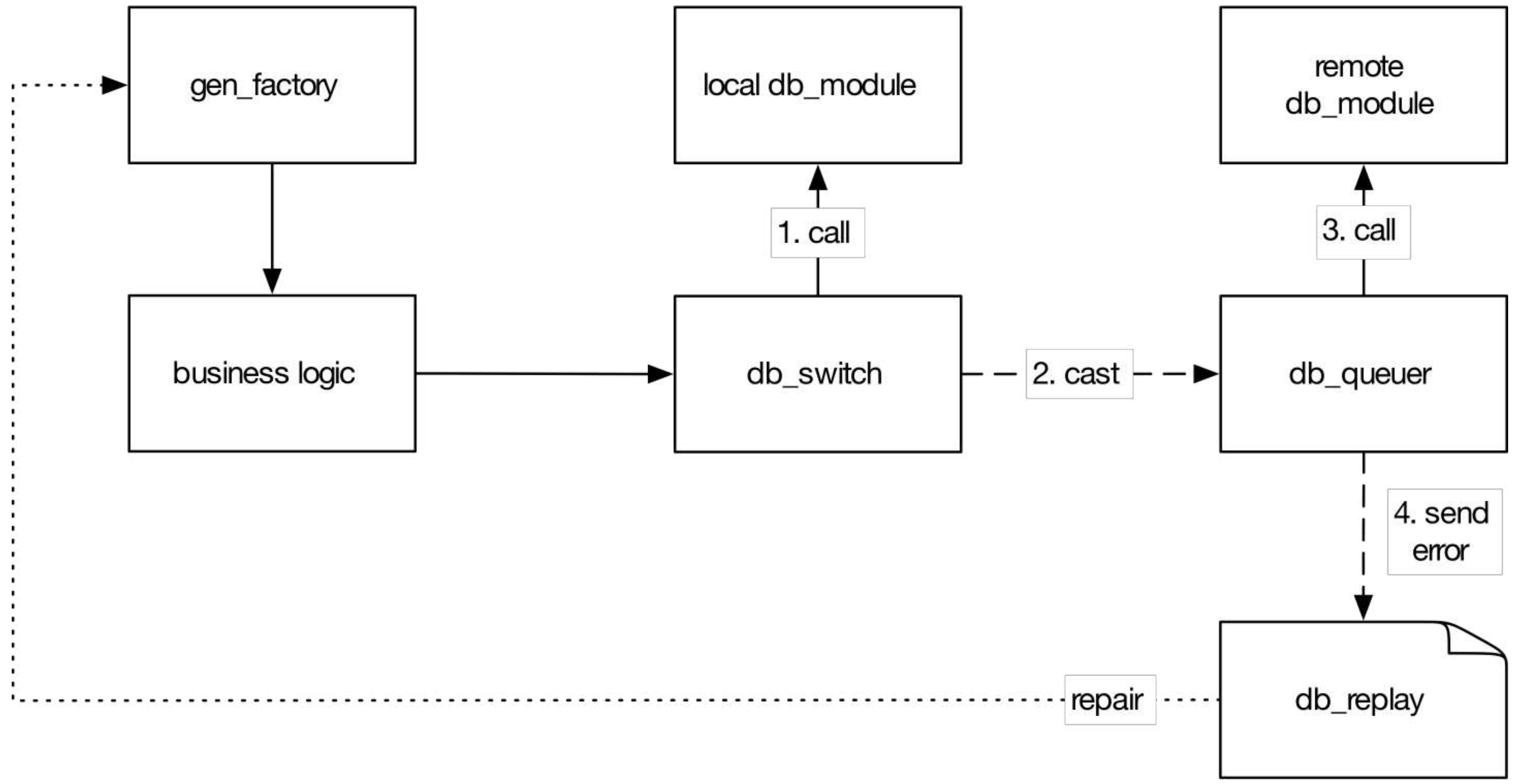
What if replication traffic is slow?

db_queuer



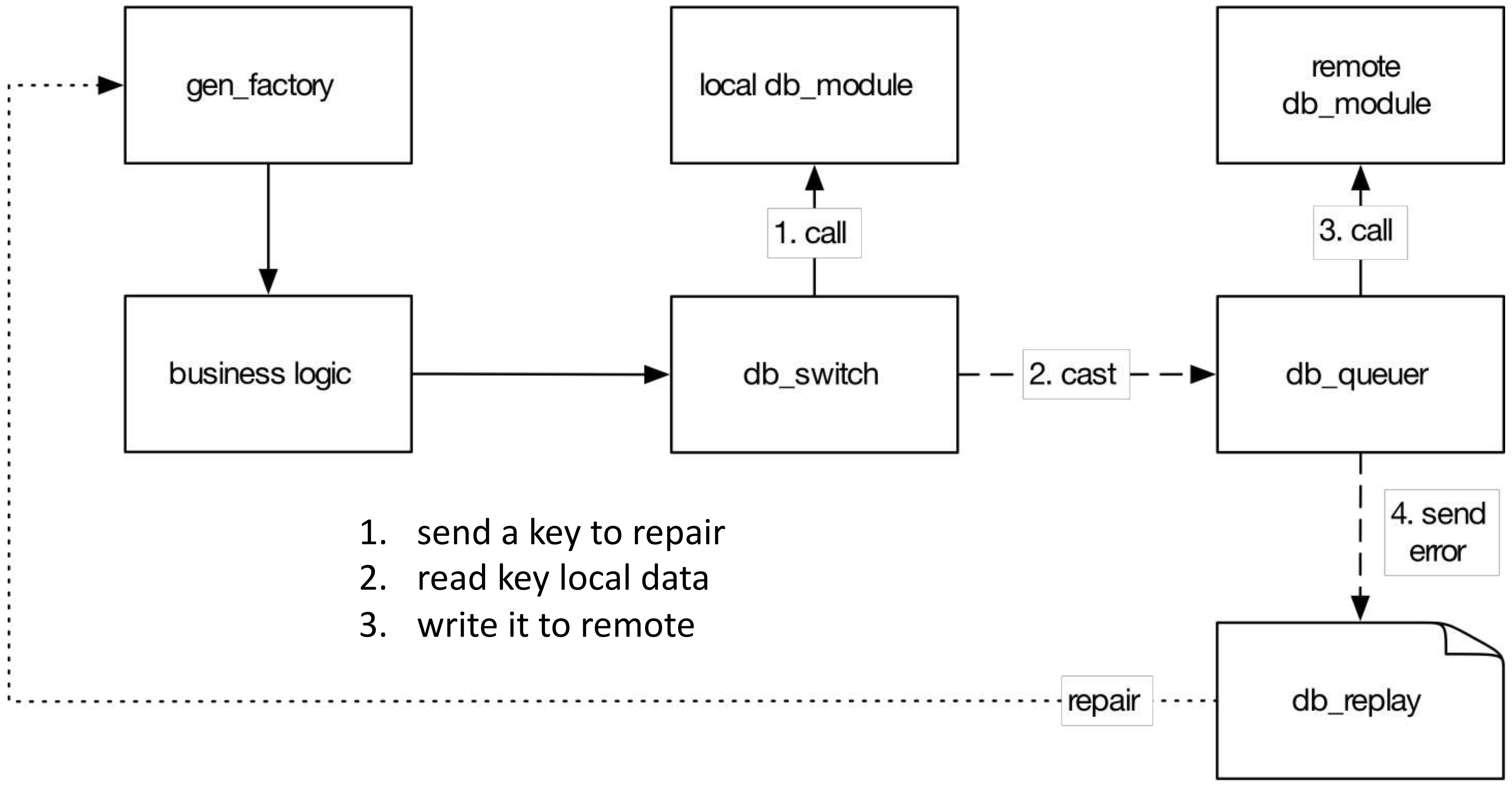
What if an error happen?

db_replay



What if an error happen?

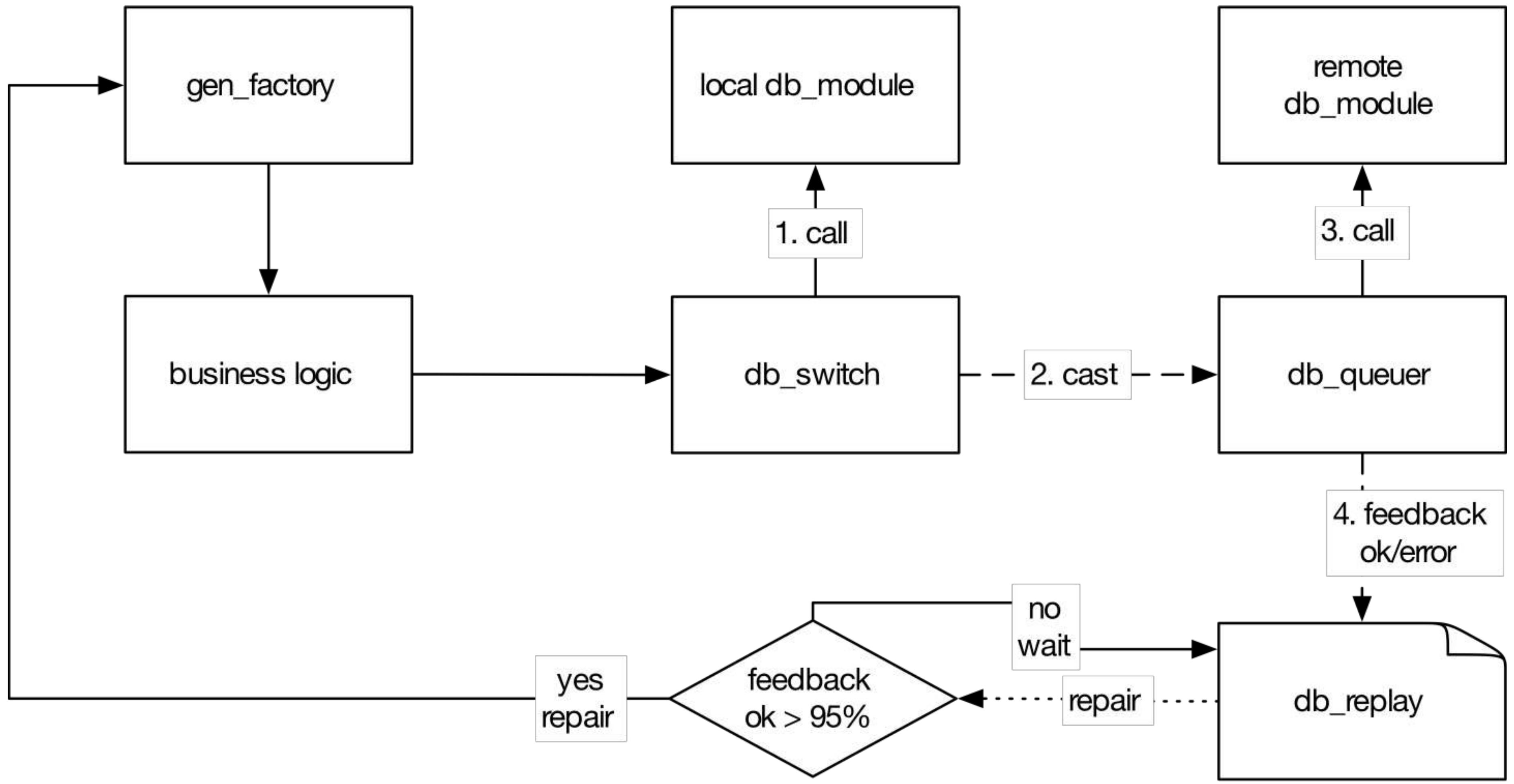
db_replay



1. send a key to repair
2. read key local data
3. write it to remote

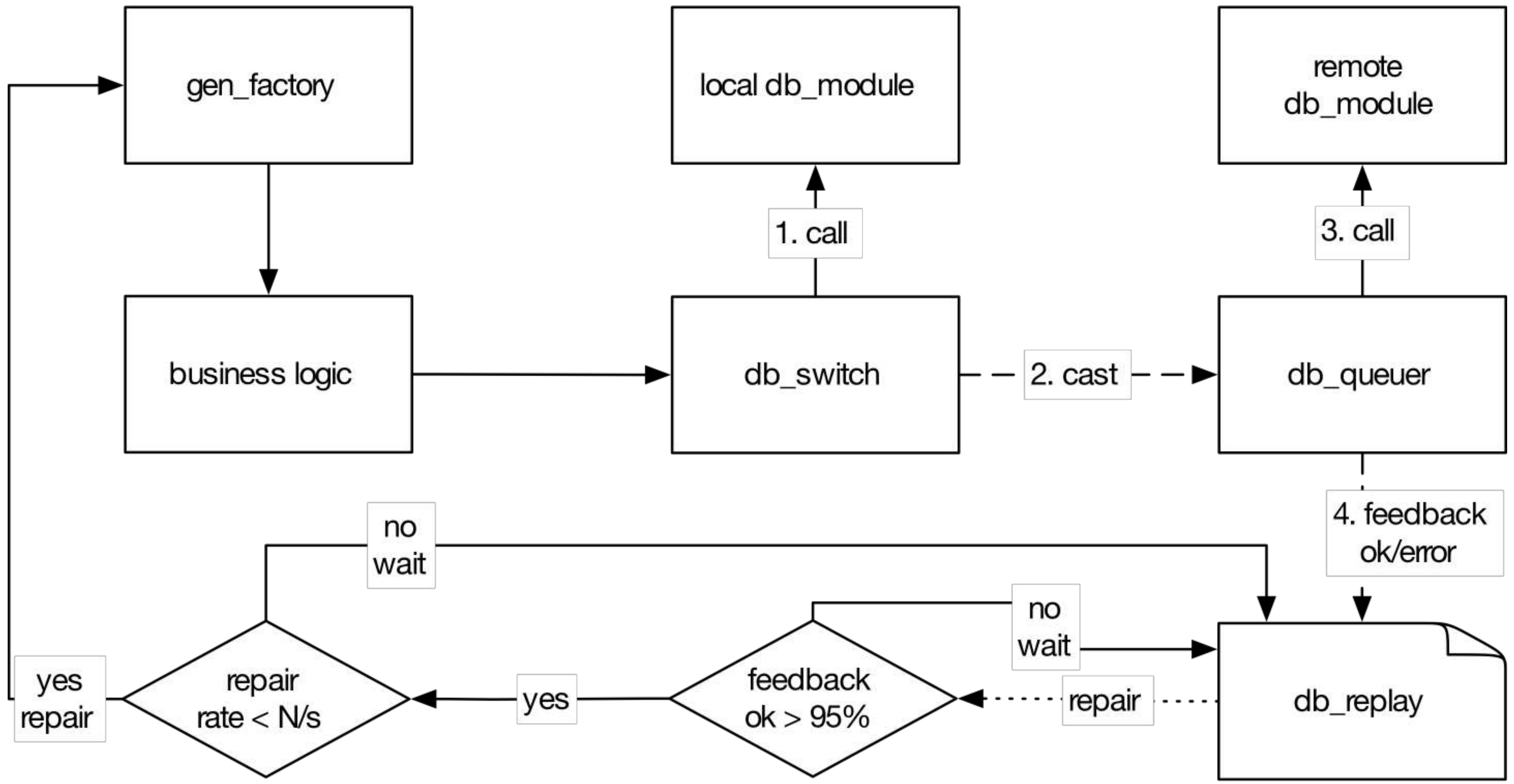
What if too many errors happen?

db_replay feedback



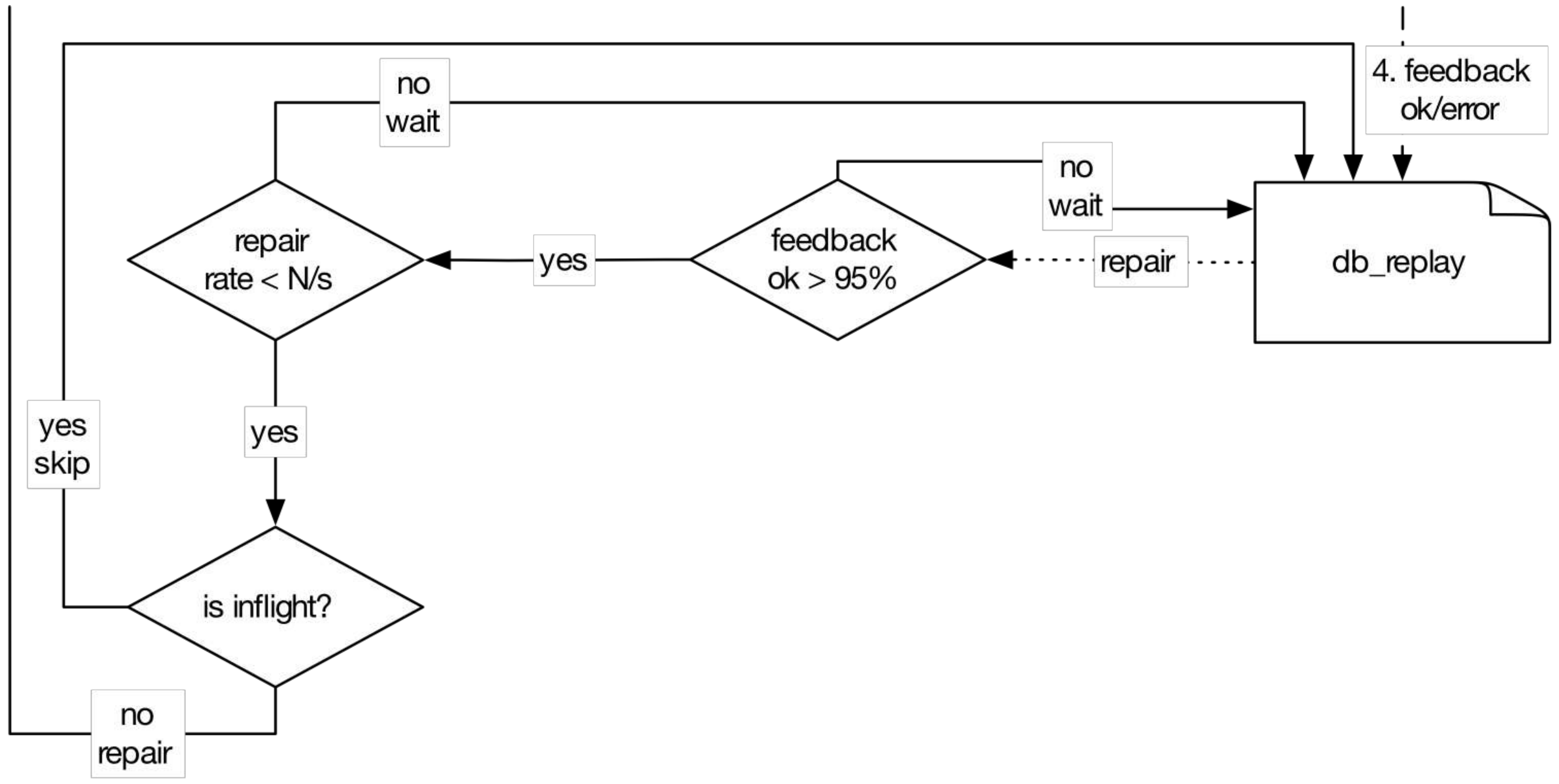
What if too many repairs happen?

db_replay rate limit



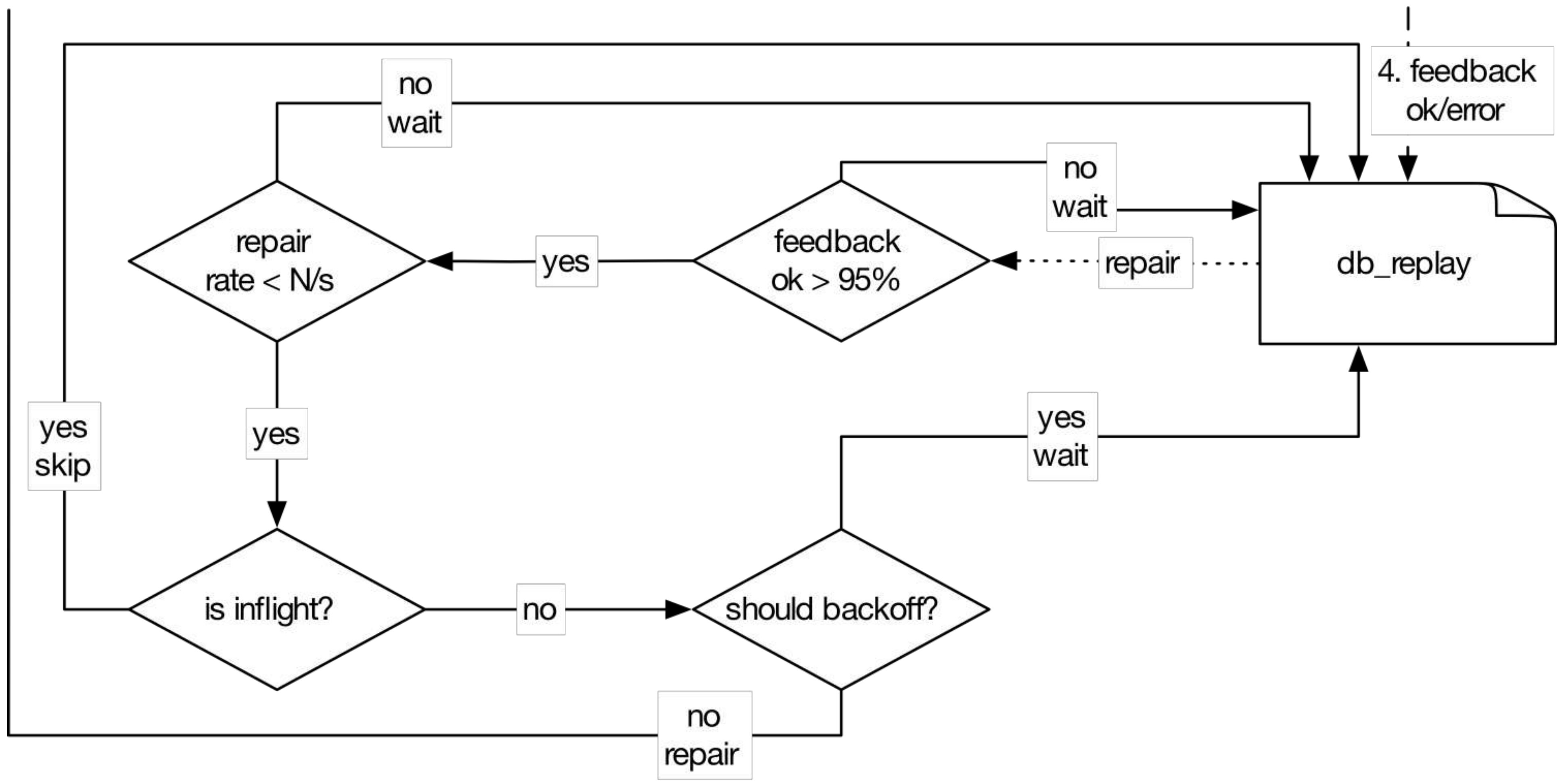
What if a key is hot?

db_replay inflight



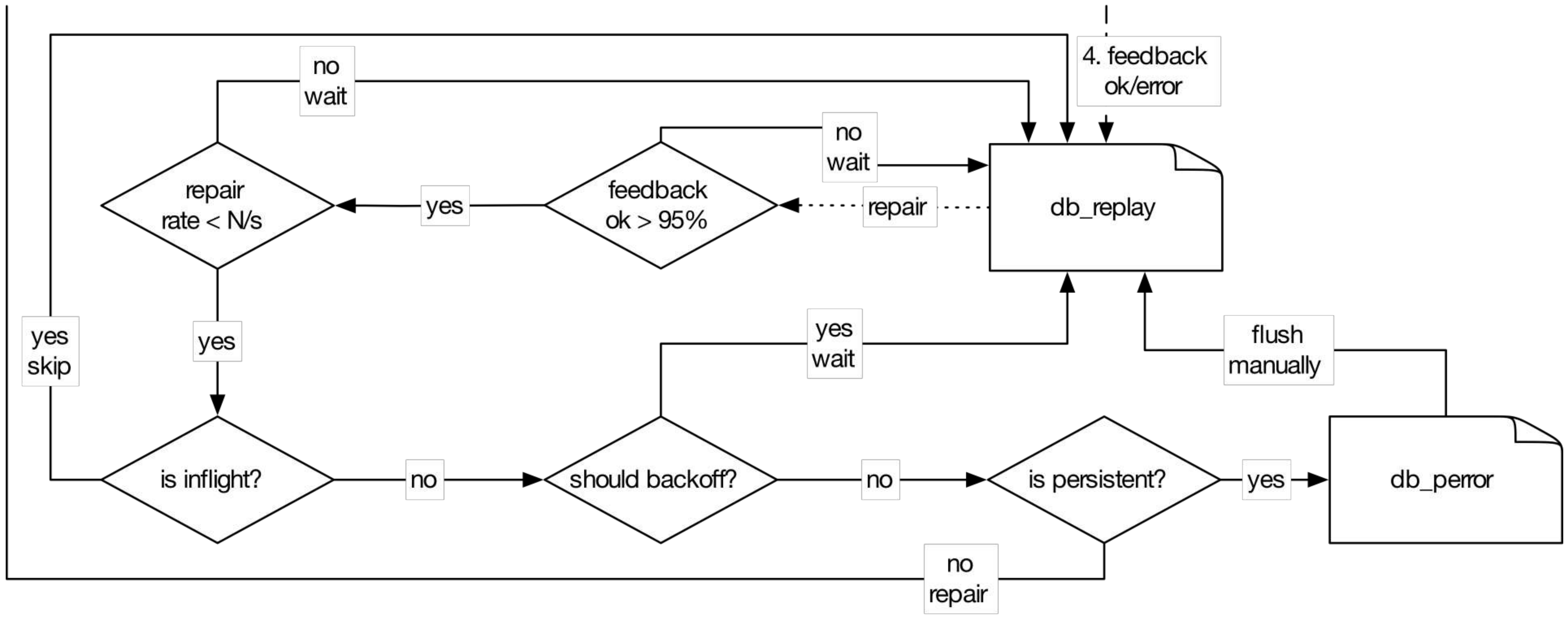
What if repairs return error?

db_replay backoff



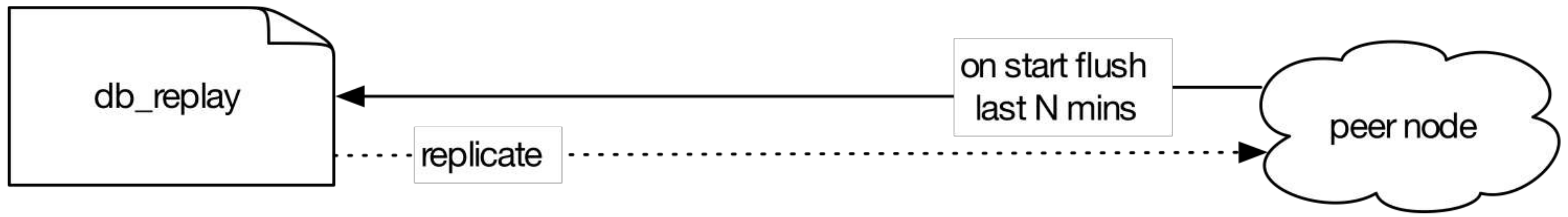
What if a key can’t be repaired?

db_replay + db_perror



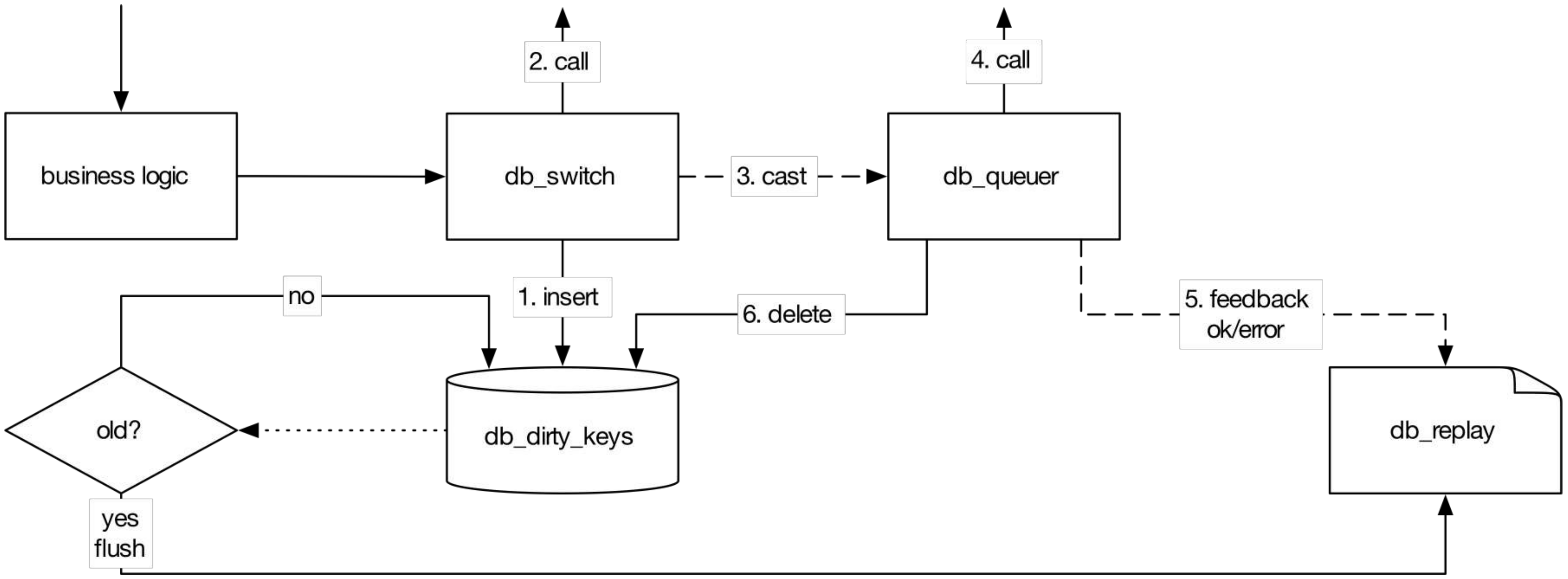
What if a node loses disk?

db_replay replication



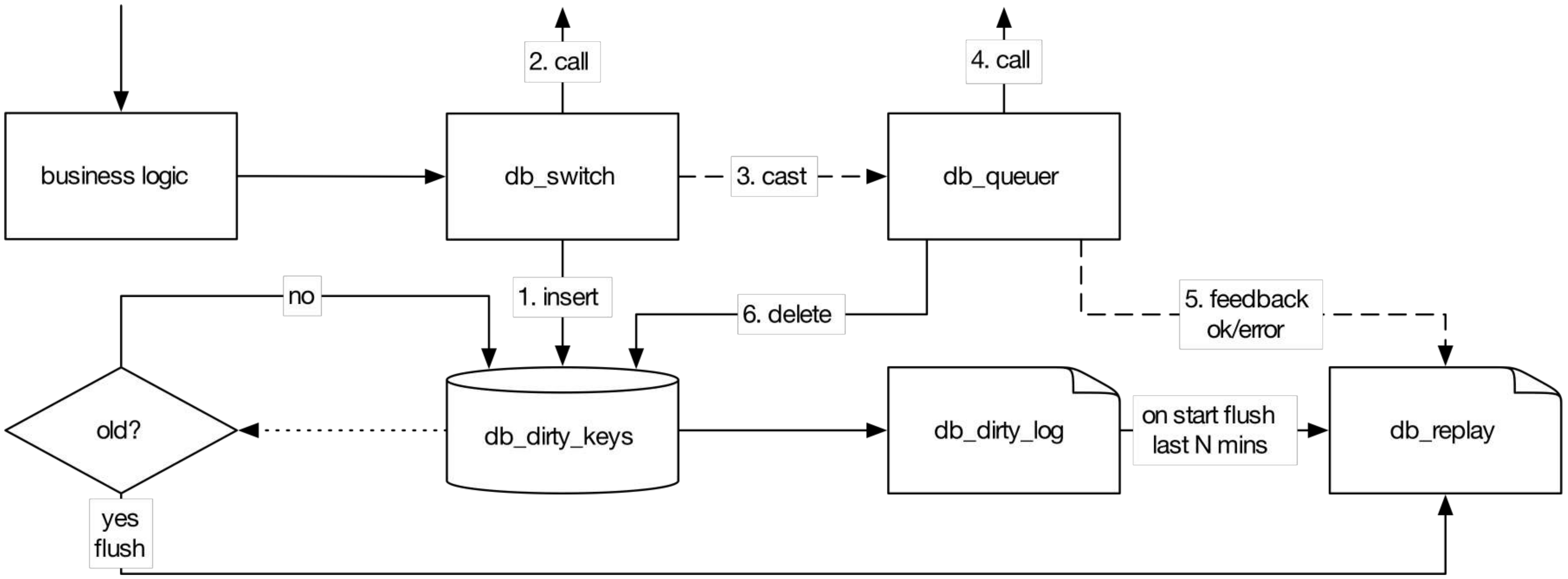
What if db_queuer discarded?

db_dirty_keys



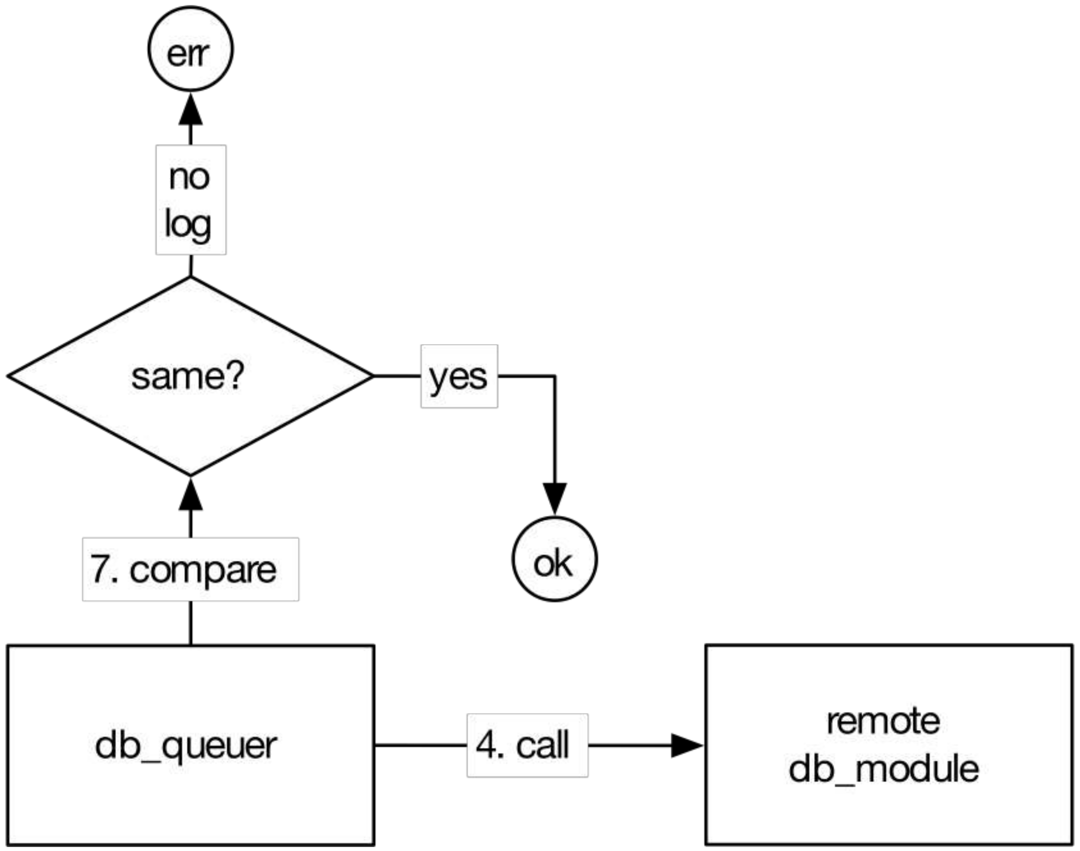
What if a node crashes?

db_dirty_log



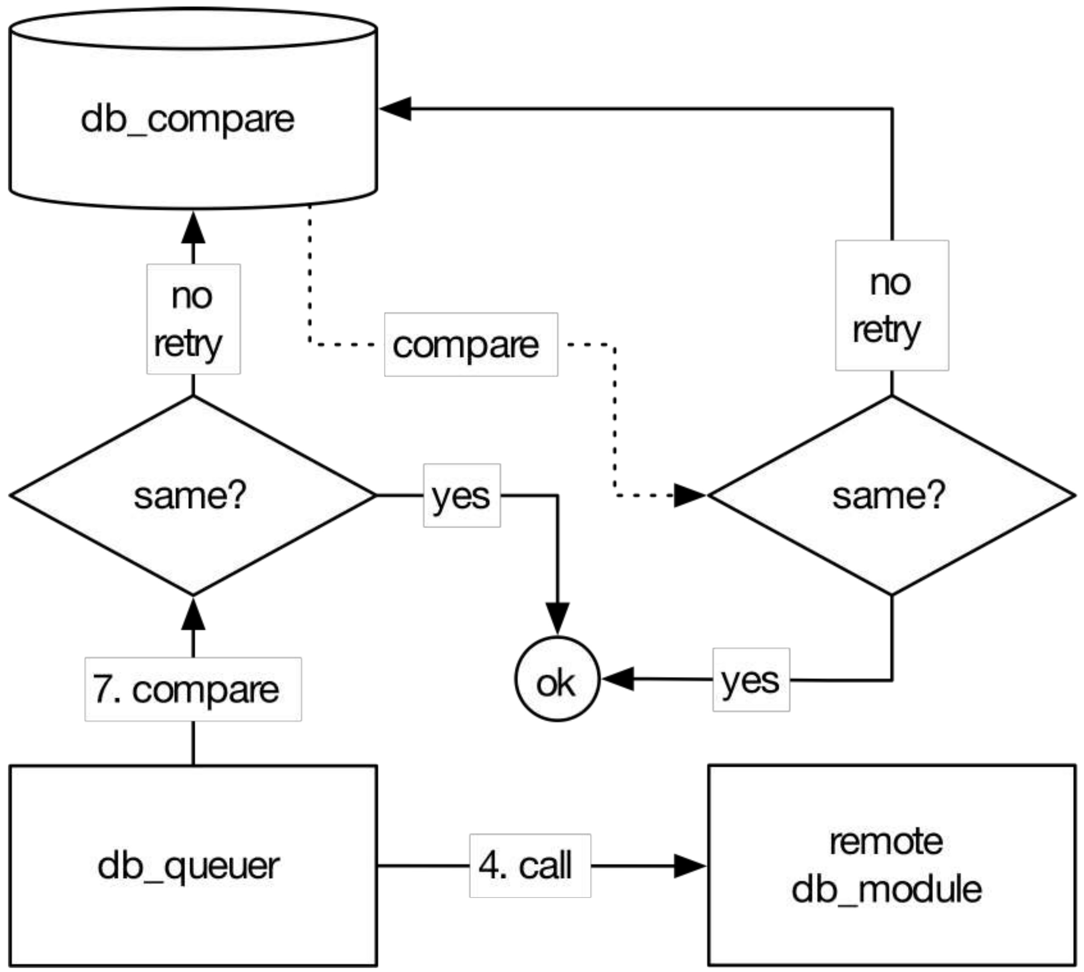
How to validate data?

compare



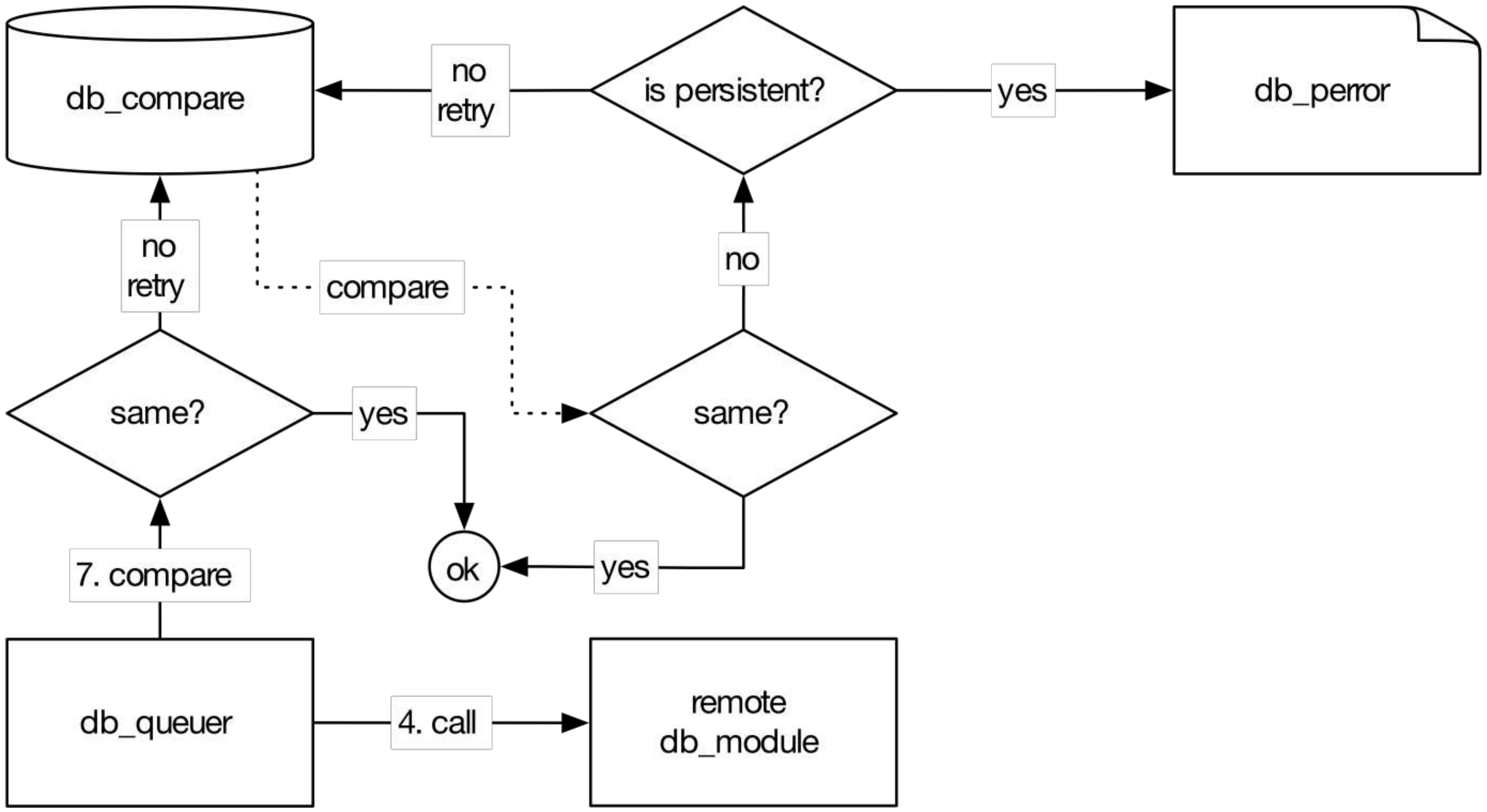
How to deal with flaky compare?

db_compare



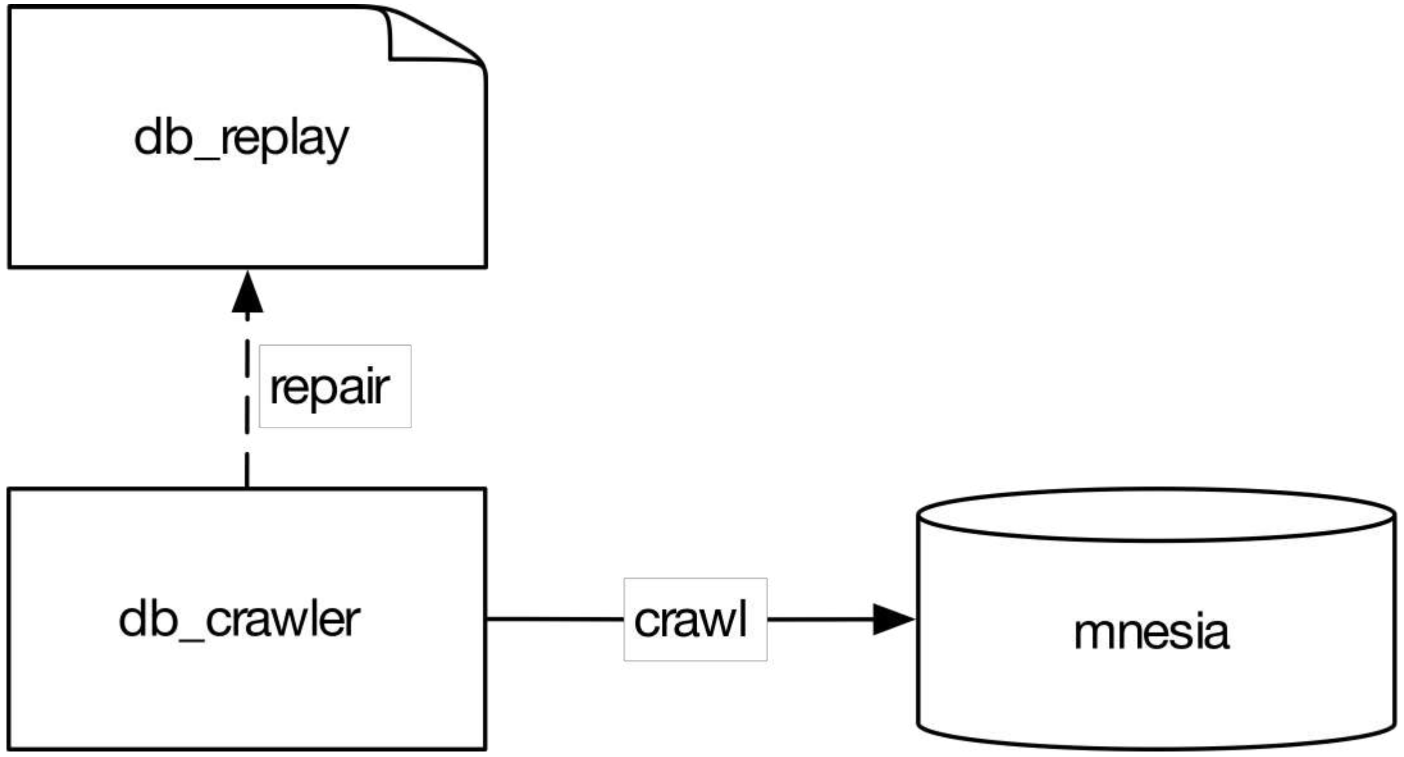
What if compare fails persistently?

db_compare + db_perror



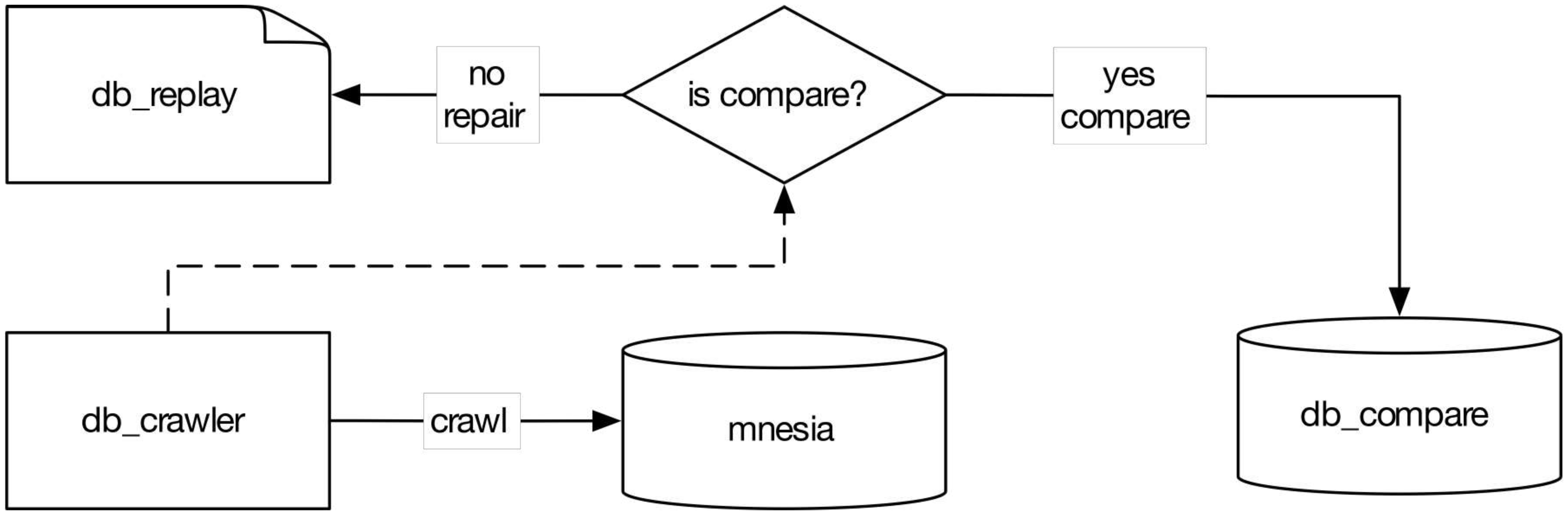
How to copy offline data?

db_crawler + db_replay



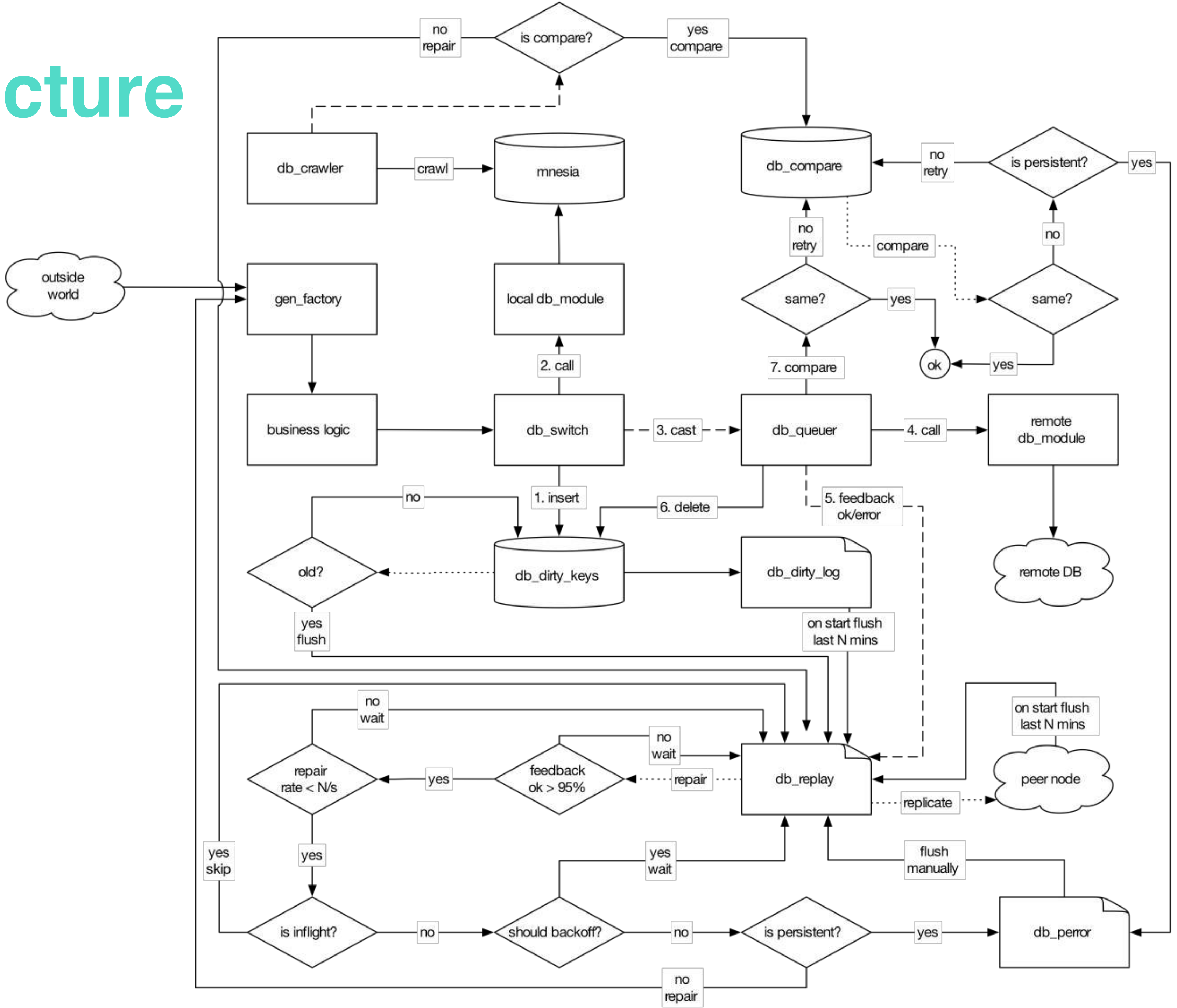
How to validate offline data?

db_crawler + db_compare



The whole picture

db_migration



How to move users to the other side?

db_failover

1. take a phone number prefix (country)
2. make the prefix read-only
3. accelerate db_replay repairs
4. gather remaining persistent errors
5. move traffic to the new data center
6. enable writes unless persistent error
7. manually reconcile persistent errors

The whole process takes < 5 mins

What's there on the other side?

same thing!

- local db_module = old remote db_module
- remote db_module = old local db_module
- add db_crawler support for the new DB
- everything else exactly the same

Learnings

- data migration is (super) (very) HARD!
- ability to validate remote data is crucial
- think about possible failure domains beforehand
- do not overthink solutions to the failure cases initially
- expect new issues to occur during the process
- be ready to tackle them
- rinse and repeat

Useful Links

- ForgETS
    - ForgETS: a globally distributed database - Code Beam STO
    - https://youtu.be/kHzmrWD7iEY
- TAO
    - Large-Scale Low-Latency Storage for the Social Network - Data@Scale
    - https://youtu.be/5RfFhMwRAic
- ZippyDB
    - Data @Scale Seattle- Muthu Annamalai
    - https://youtu.be/DfiN7pG0D0k
- gen_factory
    - That's 'Billion' with a 'B': Scaling to the Next Level at WhatsApp
    - http://www.erlang-factory.com/sfbay2014/rick-reed