Erlang User Conference - 2014-06-09

Benoit Chesneau @benoitc

enkidb

an alternative to mnesia

camp.io

why using Erlang to build a database?

Database challenges

- Collecting and organising data so they can be retrieved
- Concurrency
- ACID transactions

ACID

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Atomicity

- each transaction is "all or nothing".
- if one fail, the database stay unchanged
- Erlang: let it crash & fault tolerance
- processes fail fast

Consistency

- take the database from one valid state to another.
- Erlang supervision helps to maintain a consistent system
- process recovery

Isolation

- **seriability**: concurrent transactions result in the same system state as if they were executed serially.

- Erlang: transactions processes are isolated from others

- process messages queue

- no shared memory

- independent recovery

Durability

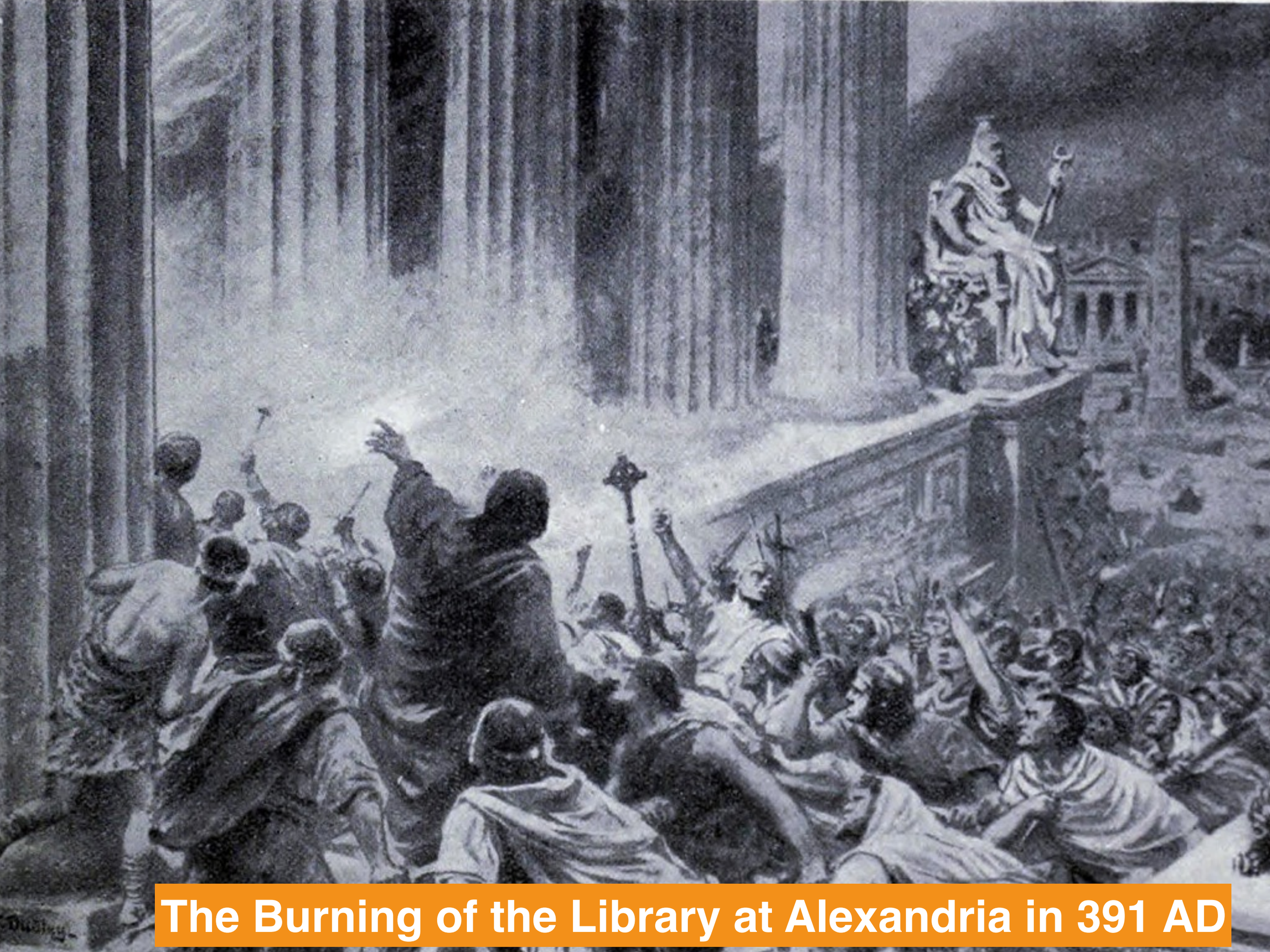
- Once a transaction has been committed, it has been recorded in durable storage
- Erlang **reliability** helps to maintain the **durability**.

a need for a specific database...

camp.io

Easily coordinate multiple data sources coming from devices, peoples or services around the world through a decentralised data platform[1].

[1] using the open source refuge solution: http://refuge.io



The Burning of the Library at Alexandria in 391 AD

copyists

camp.io

- take the control of your data back
- decentralizing data
- replicas and snapshots around the world

queries should be decentralized

- replicate snapshots data in difference parts of the world, offices or devices
- queries happen on snapshots
- sometimes offline
- or disconnected from the primary source
- and can be disconnected from other sources.

writes happen independently of reads

- writes can be centralised
- … or replicated
- without interactions with other nodes.
- over the net using HTTP(s) or not.
- support transactional writes

mnesia partly fit the bill

mnesia, partly fit the bill

- replication
- Location transparency.
- diskless nodes
- transactions support with realtime capabilities (locks selection)

but

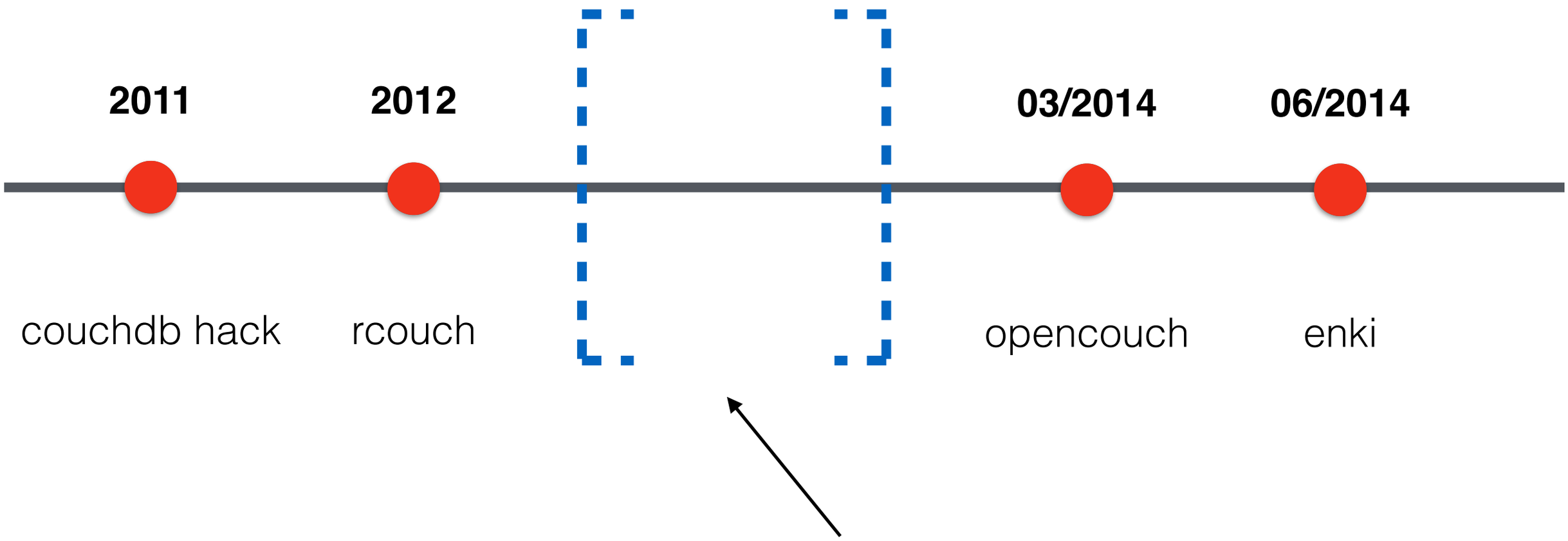
- replication works only between connected Erlang nodes
- no offline capabilities
- transactions imply dialog between different nodes where there is a replica (write lock)

facts and a bit of history...

we started by… using couchdb vs mnesia

- limit of a database > 2 GB
- master-master replication
- no nodes connections needed: P2P
- View indexation
- Modern storage

refuge.io project



hack apache couchdb. make it OTPish

- rcouch (http://github.com/rcouch)
- major refactoring to create an Erlang CouchDB releases
- some patches and new features
- the view changes
- ***WIP:*** merge back in Apache CouchDB

opencouch - the diet cure…

- rcouch was too complicated to embed
- in a need of a simpler API to add new features
- need to able to use different transports
- need something without all the extra
- https://github.com/benoitc/opencouch

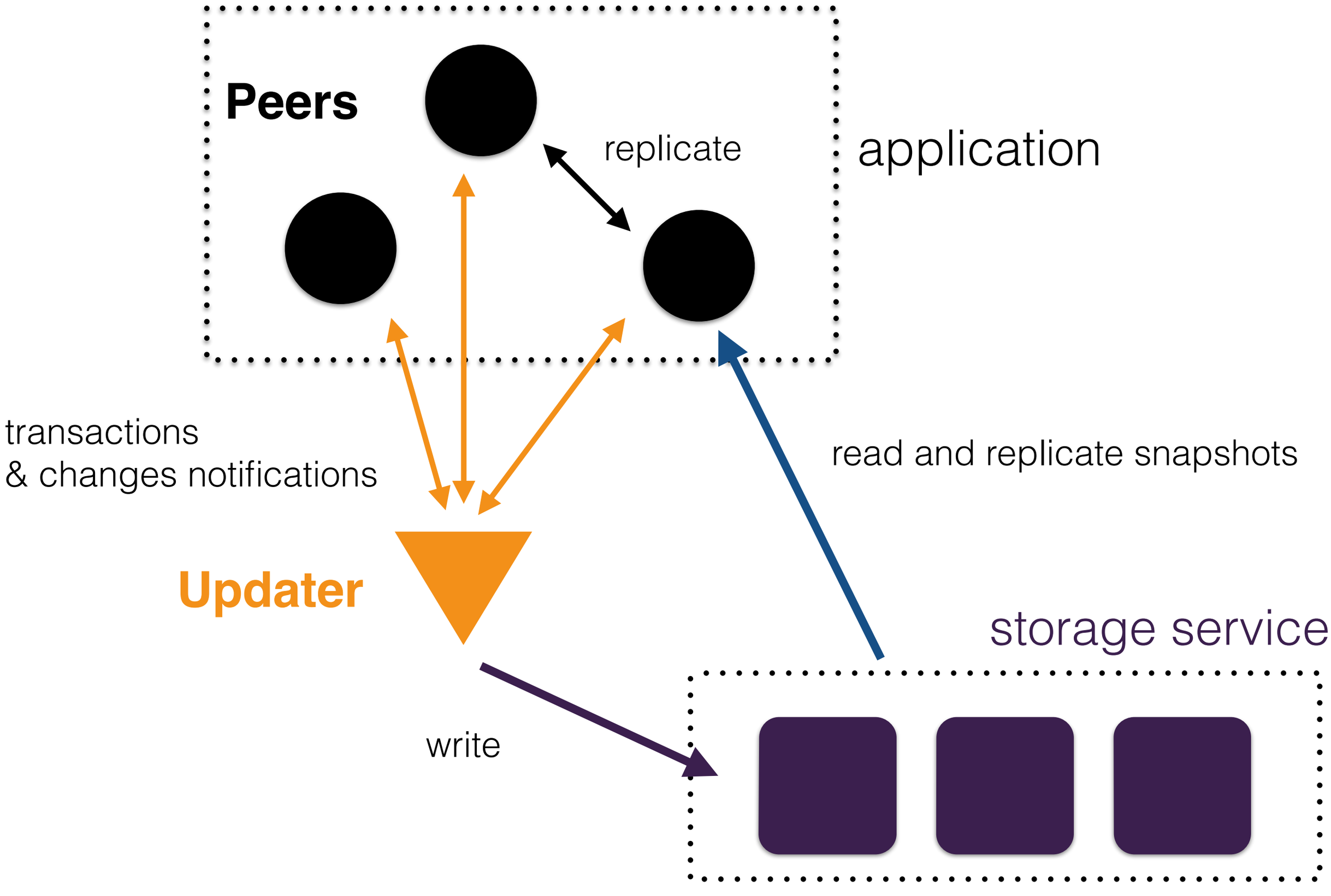
enki

one step further...

enki design

- document oriented database
- blob support
- 3 components
  - Peers
  - Updaters
  - Storage services

enki design


Peers
replicate
application
transactions
& changes notifications
read and replicate snapshots
Updater
storage service
write

peers

- Erlang library embedded in Erlang applications
- send transactions to the updaters
- query the storage services
- edit locally (offline or not)
- replication between peers
- discovery of updaters and peers handled at the application level

peers

- can replicate from Apache CouchDB
- a REST server exists

replication

- couchdb uses a revision tree
- tested other solutions:
  - dotted version clock: https://github.com/ricardobcl/Dotted-Version-Vectors
  - interval tree clocks: https://github.com/ricardobcl/Interval-Tree-Clocks
- settled to a revision tree with minor adjustments

enki revision tree

- add concurrent edit concept (also defined by damien katz)
- multi-backend support

updater

- only manage the transactions
- can manage conflicts via stored functions or transaction functions
- accept connections over different transport and using Erlang RPC.
- more complicated than a gen_server but not so much.

how a document is stored in couchdb?

- 2 indexes: by ID, by seq,
- transaction happen at document level.
- the value is the revision tree. There is one revision tree / document.
- Each revisions are stored as immutable chunks in the database file, only reference are passed to the revision tree.

storage

- key-value interface and CAS for updating
- revision tree is stored as a value associated to the document key
- revisions are stored as immutables values
- can be remote (amazon dynamodb, postgres, riak..) or local (leveldb, cowdb)
- use transaction capabilities of the storage if existing

cowdb : local storage engine

- based on the Apache CouchDB btree
- pure Erlang append only btree
- Handle transactions
- provide an easy api to store objects

the couchdb btreee

- copy-on-write (COW)
- append-only
- can store multiple btrees
- but use a lot of space (need to compact)

cbt: first attempt to extract it

- https://bitbucket.org/refugeio/cbt
- low level.
- wasn’t really usable by the end-developer
- wanted to provide a simple way to handle it.

1. create a database and initialize a btree

```
1> {ok, Fd} = cbt_file:open("test.db").
{ok,<0.35.0>}
2> {ok, Btree} = cbt_btree:new(Fd).
{ok,{btree,<0.35.0>,nil,undefined,undefined,undefined,nil,
           snappy,1279}}
```

2. initialize the btree

```
3> {ok, Btree2} = cbt_btree:add(Btree, [{a, 1}]).
                {ok,{btree,<0.35.0>,
                           {0,[],32},
          undefined,undefined,undefined,nil,snappy,1279}}
```

3. read a value

```
4> Root = cbt_btree:get_state(Btree2).
{0,[],32}
5> Header = {1, Root}.
{1,{0,[],32}}
6> cbt_file:write_header(Fd, Header).
```

1. read the header

```
1> {ok, Fd} = cbt_file:open("test.db").
{ok,<0.44.0>}
2> {ok, Header} = cbt_file:read_header(Fd).
{ok,{1,{0,[],32}}}
```

2. initialize the btree

```
10> {_, Root} = Header.
{1,{0,[],32}}
11> {ok, SnapshotBtree} = cbt_btree:open(Root, Fd).
{ok,{btree,<0.44.0>,
           {0,[],32},
           undefined,undefined,undefined,nil,snappy, 1279}}
```

3. read a value

```
12> cbt_btree:lookup(SnapshotBtree, [a]).
[{ok,{a,1}}]
```

useful but not for the end developer.

cowdb another object database

- https://bitbucket.org/refugeio/cowdb

- wrapper around the couchdb btree

- doesn’t depends on cbt (but should be probably)

initialize a database

```
1> {ok, Pid} = cowdb:open("testing.db",
1>      fun(St, Db) -> cowdb:open_store(Db, "test") end
!> ).
{ok,<0.35.0>}
```

initialize a store

simple transaction

```
2> cowdb:lookup(Pid, "test", [a,b]).
[{ok,{a,1}},{ok,{b,2}}]
3> cowdb:transact(Pid, [{remove, "test", b}, {add, "test", {c,
        3}}]).
ok
4> cowdb:lookup(Pid, "test", [a,b,c]).
[{ok,{a,1}},not_found,{ok,{c,3}}]
5> cowdb:get(Pid, "test", a).
{ok,{a,1}}
```

transaction functions

transaction function

```
7> cowdb:transact(Pid, [
     {fn, fun(Db) -> [{add, "test", {d, 2}}] end}
]).
ok
8> cowdb:lookup(Pid, "test", [d]).
[{ok,{d,2}}]
```

opensourcing Enki

Enki will be released under an opensource license. Paying support will be available.

Refuge.io

?

camp.io

@benoitc