Refuge

Building a decentralized data platform in Erlang

Erlang Factory SF Bay Area 2013
Benoît Chesneau

about me?

- Apache CouchDB committer and PMC member
- PSF Member
- Web Craftsman
- Doing opensource for a living

What is refuge?

- A way to store, sync and share data
- Decentralized
- Over and On the web
- Opensource
- Built in Erlang

Why?

I played a lot with Apache CouchDB

- A document Oriented Database
- Blobs can be attached to a document
- Replication Master-Master (P2P)
- Over and On the web
- Opensource
- Built in Erlang

But

What we really need at the end is

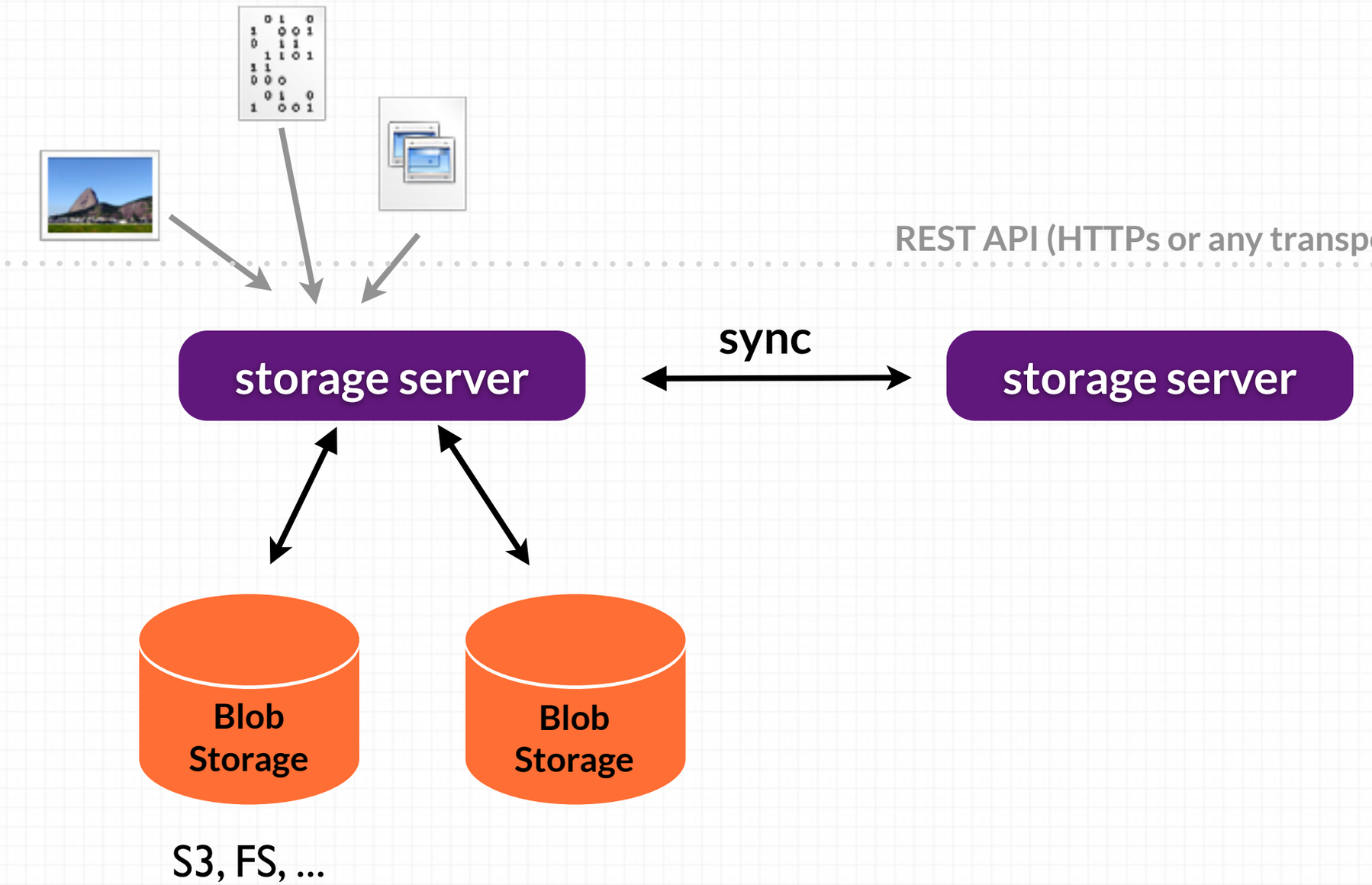
- A simple and efficient way to store any blobs
- Index or render them
- and share them among peoples or machines.
- Can work with offline devices

Coffer

The storage service

- multi-backend: FS, Distributed FS, Haystack, S3...
- GET, PUT, DELETE, LIST
- SYNC
- All blobs are uniquely identified. The ID is the content-hash. <hashtype>;<hash>
- handle partial uploads
- HTTP transport (optionnal)

Storage Service

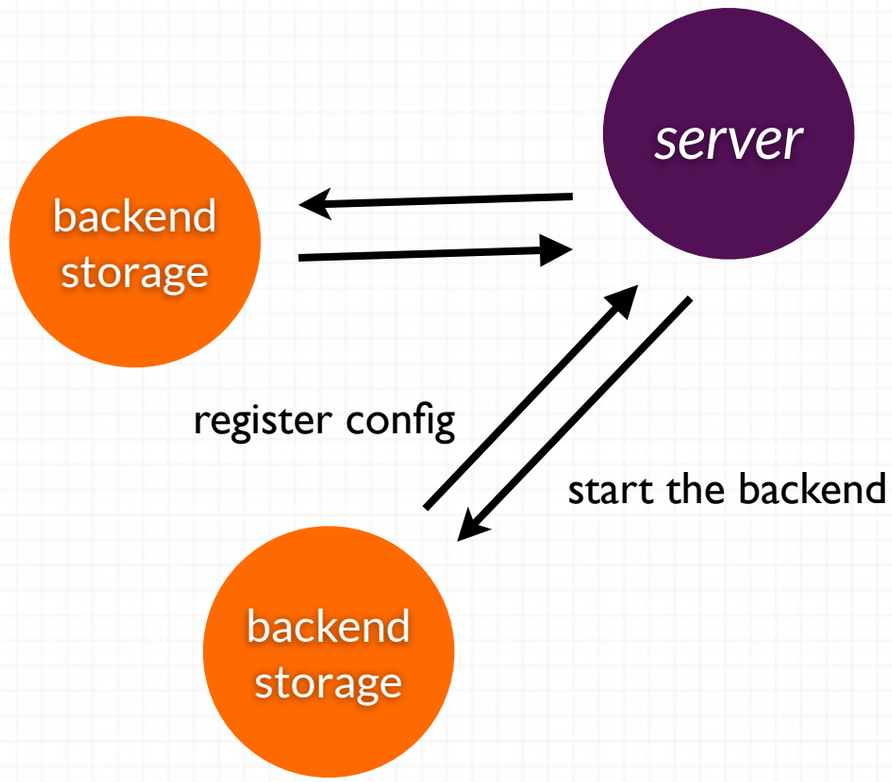


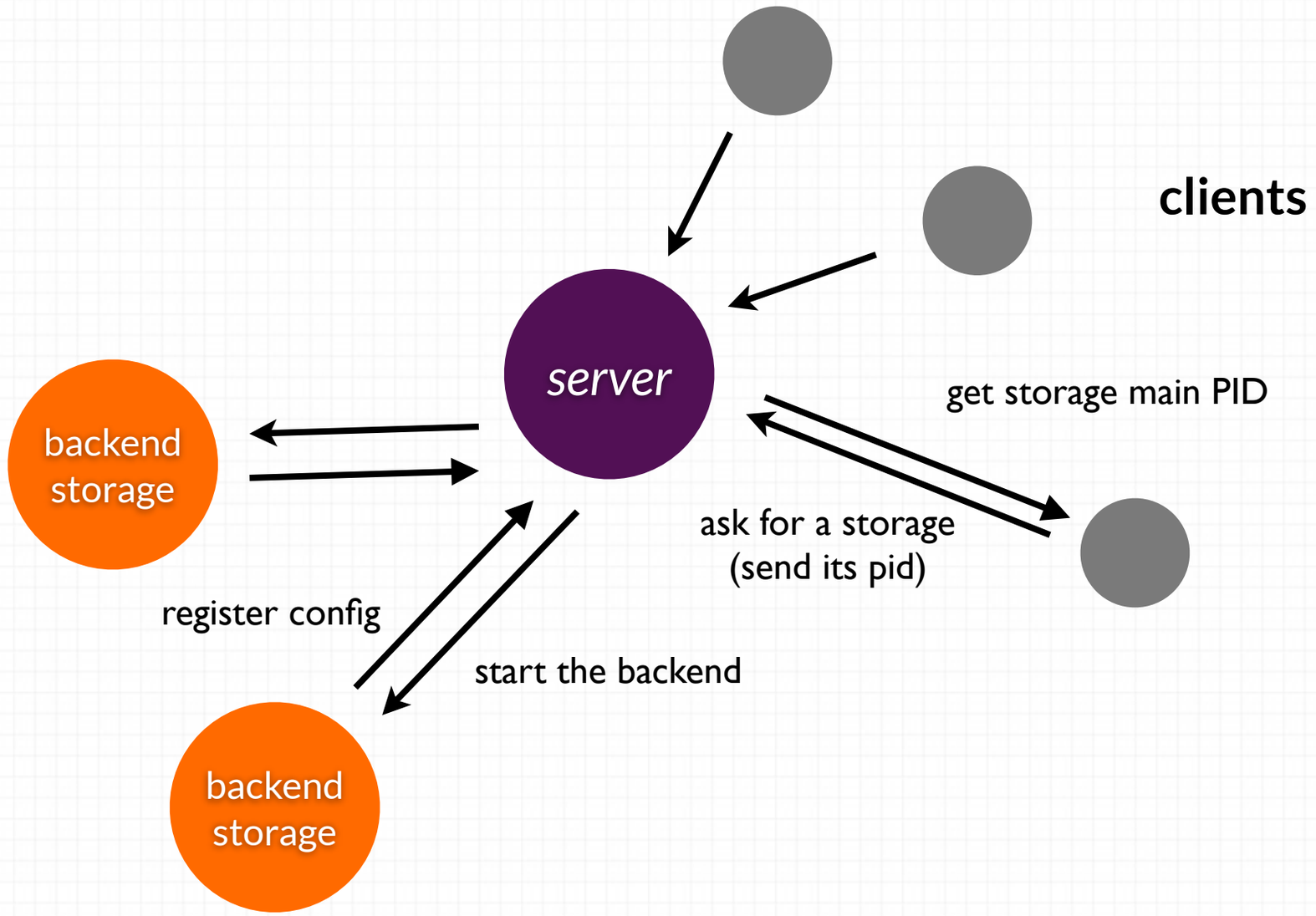
How the synchronization works?

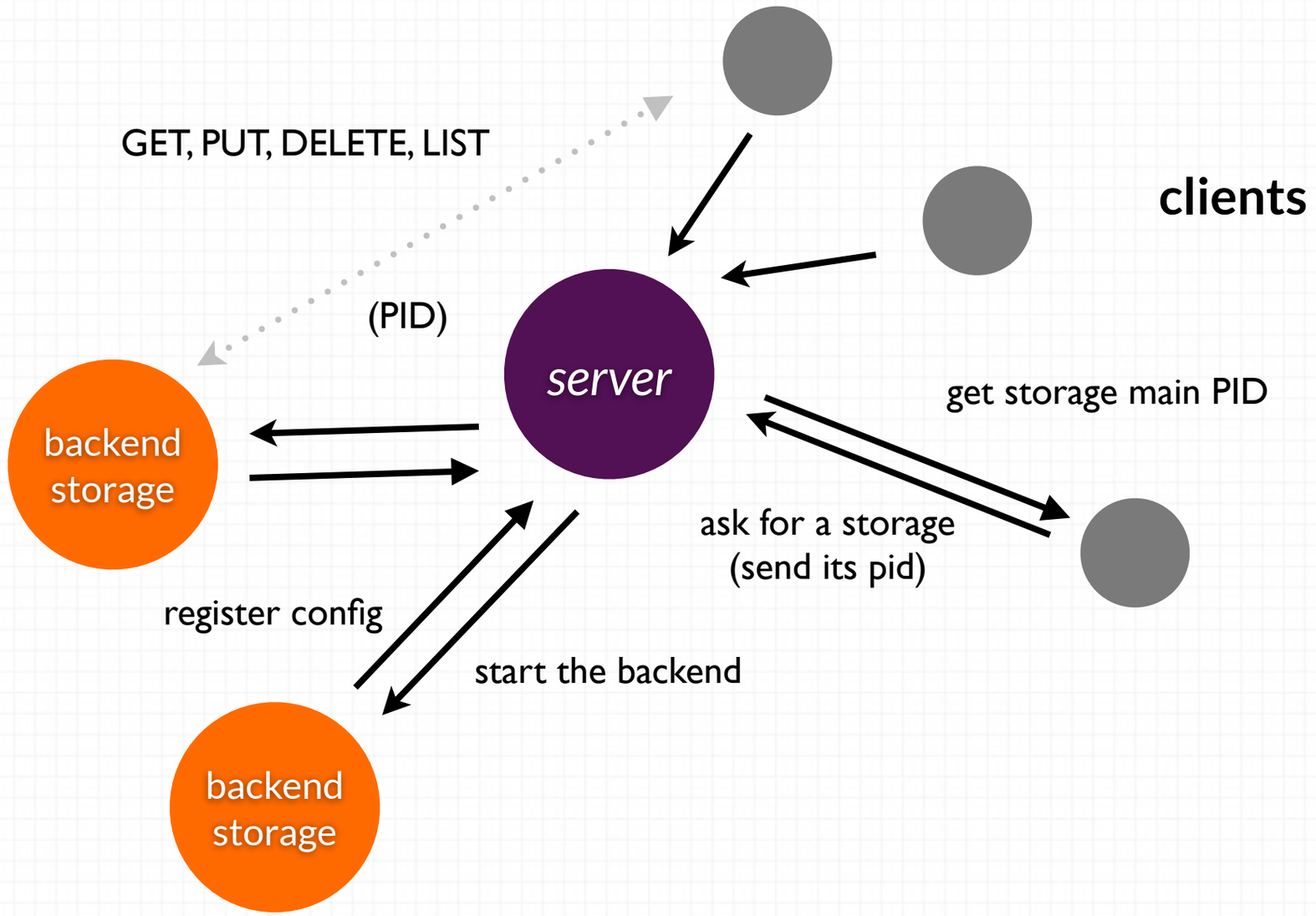
1. **bootstrap**: **LIST** (sorted) all blobs on the source and the target and copy the blobs not on the target.
2. when in sync, keep for each (source, target) replication a **queue on the source**
3. **New blobs go first in the source queue** and are **replicated** to the destination (or re-enumerated)
4. Blobs already on the target aren't sent.

How it woks in Erlang

- A gen_server to keep all the storage backends configuration
- **gen_storage**: A behavior similiar to gen_server but keeping a storage state
- handle conflicts in the backend. (A file can't be uploaded by 2 clients)
- Each consumer of the api are registered


server
backend
storage
register config
start the backend
backend
storage


clients
server
backend storage
get storage main PID
ask for a storage (send its pid)
register config
start the backend
backend storage


GET, PUT, DELETE, LIST
(PID)
clients
server
backend storage
get storage main PID
ask for a storage (send its pid)
register config
start the backend
backend storage

How the sync works in Erlang

- Queues are kept in memory
- A process / queue
- On update (or delete) an event is broadcasted to each queues
- make sure the target is always up
- enumerate is cheap (we only compare blobs ids)

How to use the blobs?

- no metadata on the disk.
- no history
- just blobs

How to use the blobs?

- use **refs** (aka permalinks) , link to to your data
- index your data
- share them

Example: backup a folder

- 3 kinds of blobs: 2 schema & the binary
- 1 "commit" schema to describe the file if needed
- 1 "tree" schema to describe a folder
- A schema is a **blob**.
- 1 ref to keep track of latest tree
- similar to git? yes.

file

```
{
        "blobid": "blobobid",
        "prev": "prevref or null"
}
```

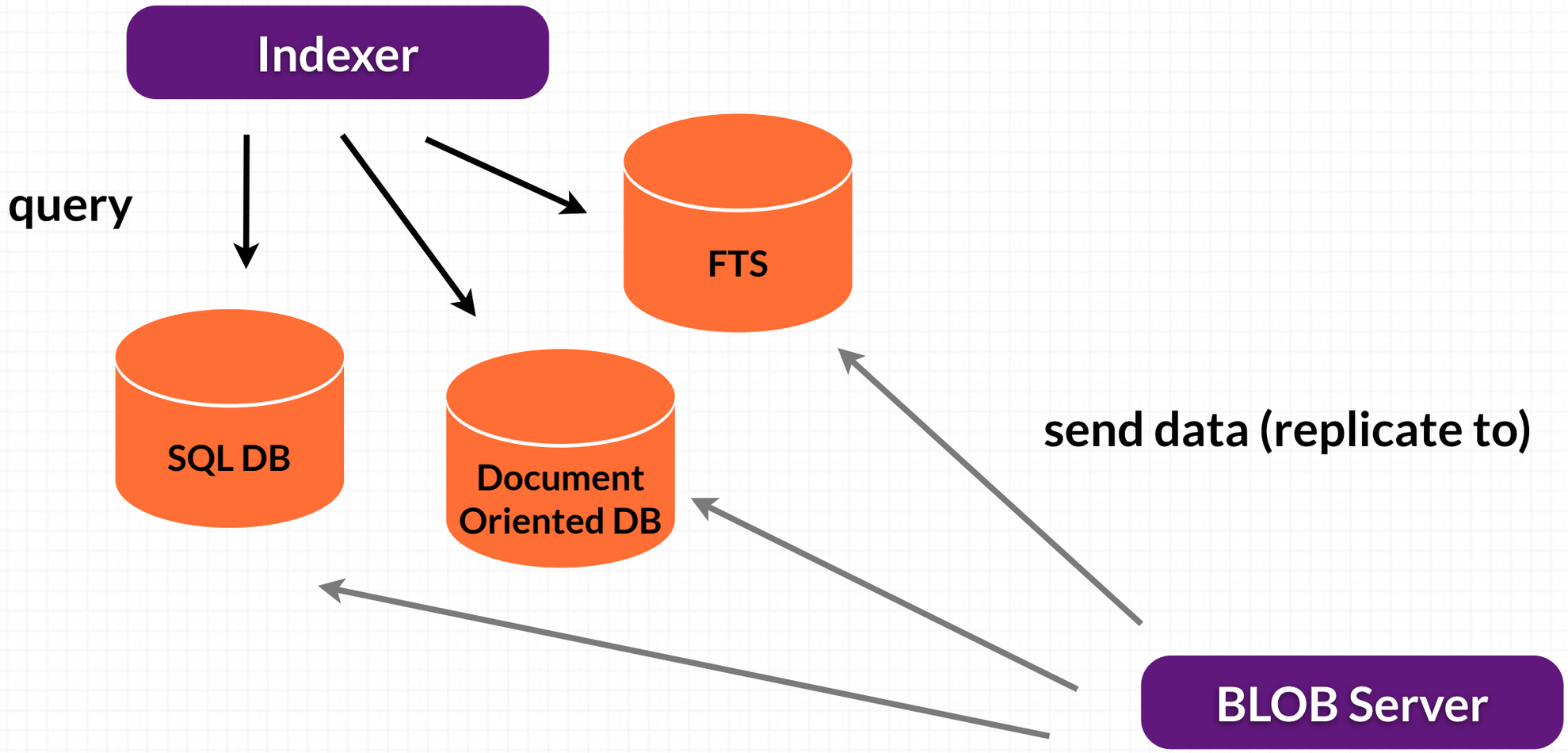
tree

```
{
        "filename": {
            “blobid: "blobbid",
            “type”: “blob”
        },
        "foldername": {
            “blobid: "blobbid",
            “type”: “tree”
        }
    }
```

What about my blog

- A post is a blob
- A category is a blob linking to posts (like a tree)
- home, either blobs / date indexed or create a special blob linking to those you want on the home

Indexer Service



Index your content

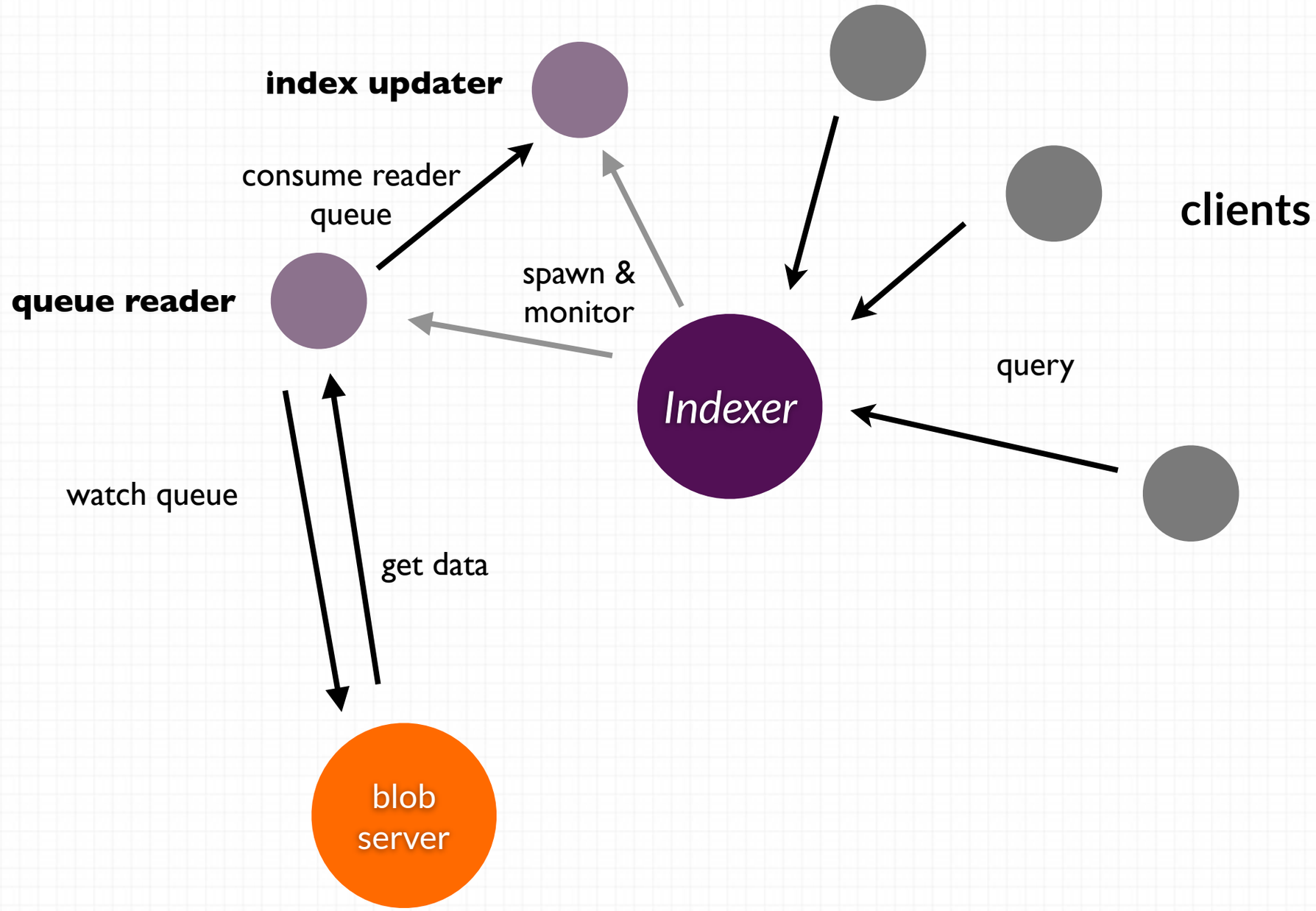
- "just" **replicate to your index**
- An indexer receive `(blobid, blob, time)` from the replication queue in quasi RT or enumerate it.
- can be **any kind of index**: sql, apache couchdb, a document oriented DB, an FTS (like elasticsearch) ....

Behind the scene

- Mostly work like a blob server
- except it only pass the data to the indexer
- Possibility to transform the data before indexing it (mapping)
- No JS (by default): but a simple DSL allows you to map fields or use scripts (luerl, ...)

Behind the scene

- Use a websockets (actually sockjs) or tcp
- Pass simple messages (json right now)
- Each queue is load balanced on each reader (to allows index balancing and stuff like it)


index updater
consume reader queue
queue reader
spawn & monitor
clients
query
Indexer
watch queue
get data
blob server

The refuge Node

- Frontend to blobs servers and indexers
- manage blobs claims and access
- share collections of data and for some allows remote queries/filtering.
- HTTP REST API
- hackney & wsock

USER CONTROL

PRIVATE CONTENT

refuge node

PEER REST SERVER

Create collections.
Expose query API

Give access to blobs &
check permissions

Indexer Service

Indexer

query

FTS

SQL DB

Document Oriented DB

send data (replicate to)

Storage Service

REST API (HTTPs or any transport)

storage server

syn

storage server

Blob Storage

Blob Storage

The hub

- Each refuge node can open a connection to an hub (websocket)
- Once connected a node identified itself with its identity
- An heartbeat (NOP) is sent to maintain the connection

The hub

- true decentralized system
- Like epmd but different
- Once found, nodes are directly connected
- A node can authenticate against a signature or a key (oauth bearer token)
- webfinger & host-meta

refuge node

Search

hub

hub

Register node & data channels

refuge node

direct connection

USER CONTROL

PRIVATE CONTENT

refuge node

PEER REST SERVER

Create collections.
Expose query API

Give access to blobs & check permissions

Indexer Service

Indexer

query

FTS

SQL DB

Document Oriented DB

send data (replicate to)

Storage Service

REST API (HTTPs or any transport)

storage server

syn

storage server

Blob Storage

Blob Storage

refuge node

hub

refuge node

direct connection

hub

direct connection

USER CONTROL

PRIVATE CONTENT

refuge node

PEER REST SERVER

Create collections.
Expose query API

Give access to blobs &
check permissions

Indexer Service

Indexer

query

FTS

SQL DB

Document Oriented DB

send data (replicate to)

Storage Service

REST API (HTTPs or any transport)

storage server

syn

storage server

Blob Storage

Blob Storage

The hub

- Each refuge node open a connection to an hub (websocket)
- A node can connect to multiple hubs
- Once connected a node identified itself with is identity
- An heartbeat (NOP) is sent to maintain the connection

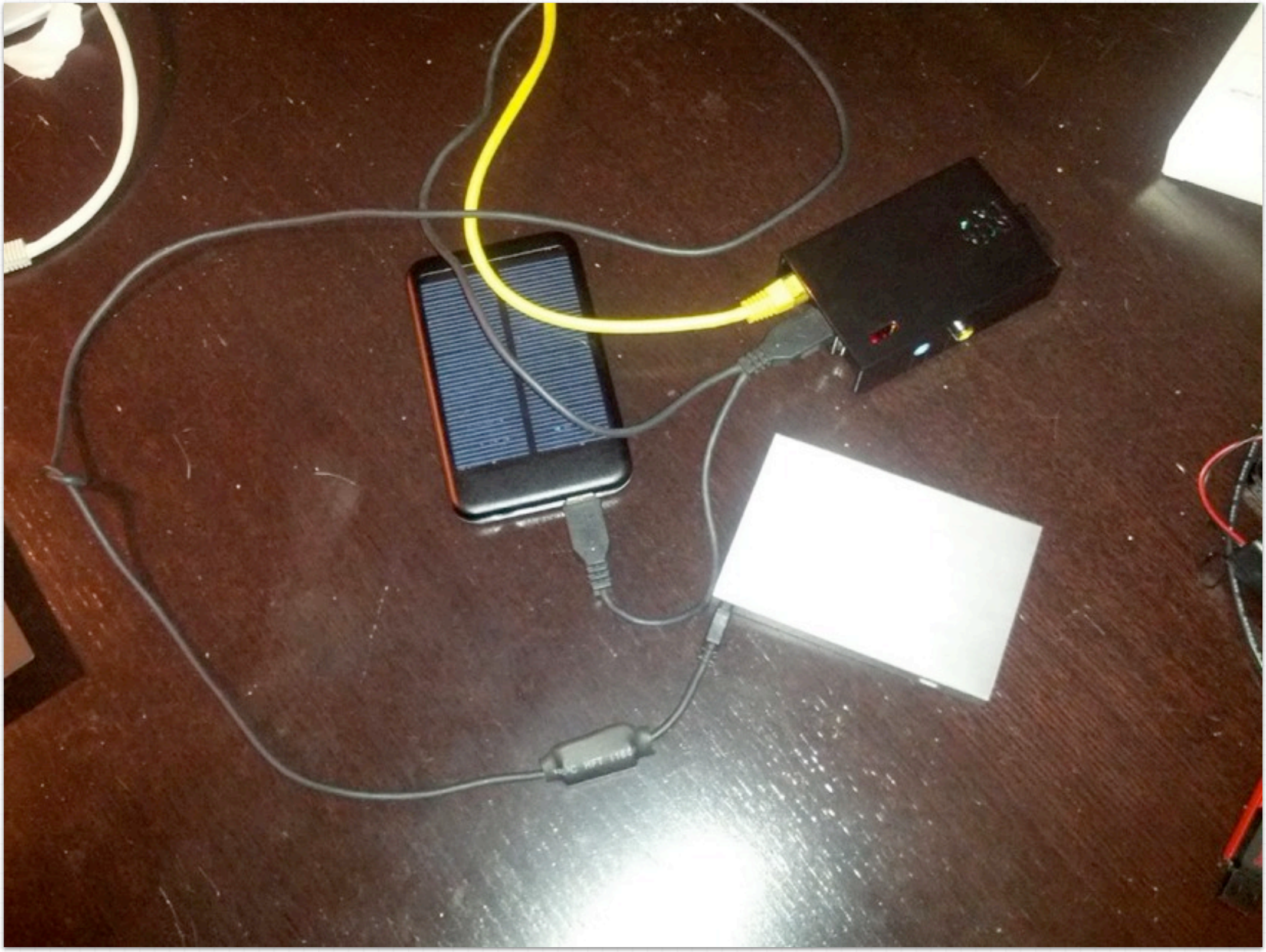
Status

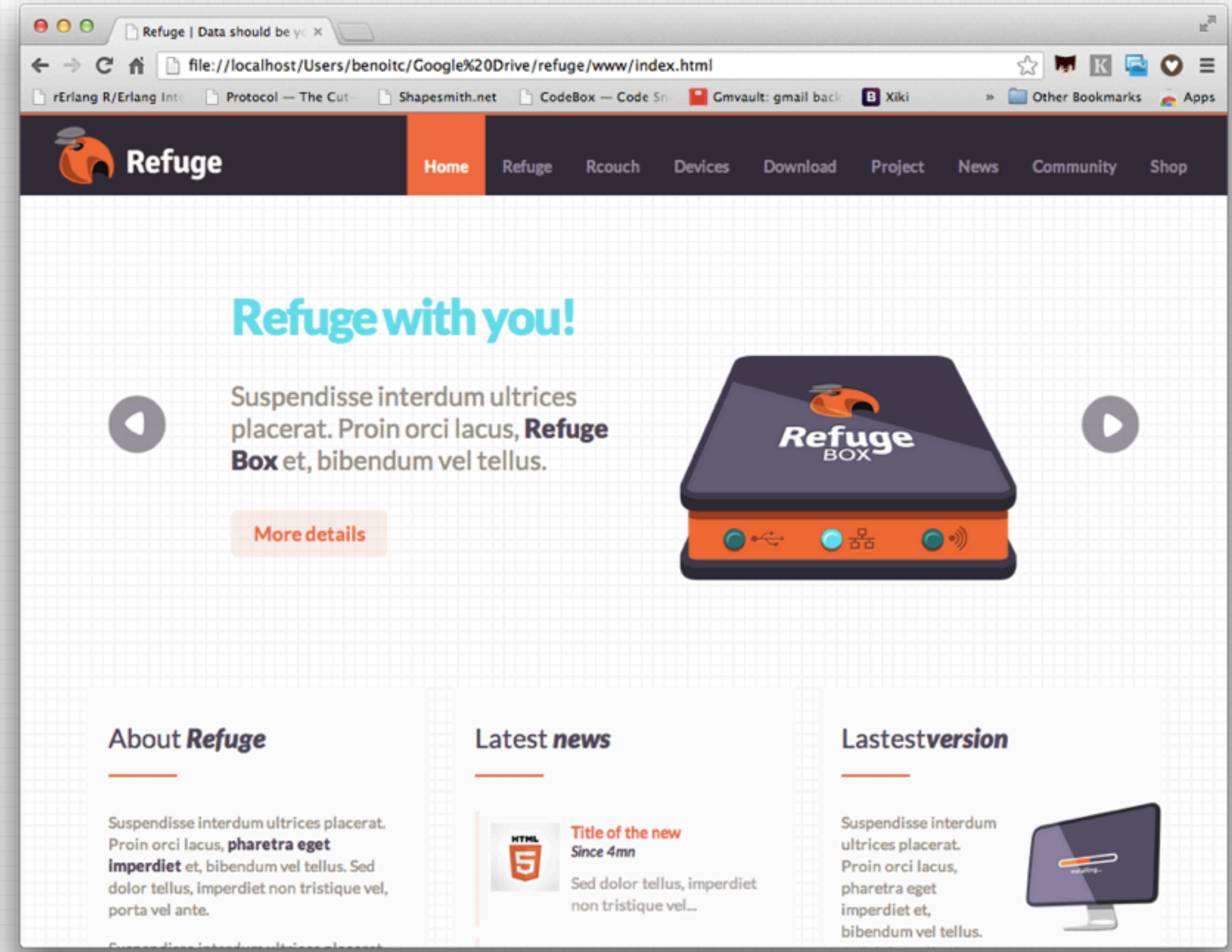
- Coffer released next week
- Refuge released in april 2013

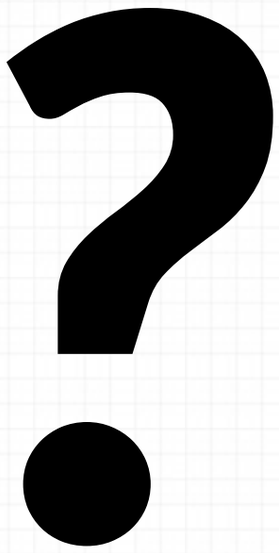
More things...

The refuge box

- arm platform
- standalone installation of refuge
- internet of things
- dns-sd & udp discovery




Refuge | Data should be y
file://localhost/Users/benoitc/Google%20Drive/refuge/www/index.html
rErlang R/Erlang Int
Protocol — The Cut
Shapesmith.net
CodeBox — Code Sn
Gmvault: gmail back
Xiki
Other Bookmarks
Apps
Refuge
Home
Refuge
Rcouch
Devices
Download
Project
News
Community
Shop
Refuge with you!
Suspendisse interdum ultrices placerat. Proin orci lacus, Refuge Box et, bibendum vel tellus.
More details
Refuge BOX
About Refuge
Suspendisse interdum ultrices placerat. Proin orci lacus, pharetra eget imperdiet et, bibendum vel tellus. Sed dolor tellus, imperdiet non tristique vel, porta vel ante.
Latest news
HTML 5
Title of the new
Since 4mn
Sed dolor tellus, imperdiet non tristique vel...
Lastest version
Suspendisse interdum ultrices placerat. Proin orci lacus, pharetra eget imperdiet et, bibendum vel tellus.



@benoitc

http://refuge.io

Thanks to
Laurent (@lolograph) for the website & logo design
Nicolas (@nrdufour) for Code and Ideas
Others for their feedback

introducing blanket

- a document oriented database
- multiple backends (sqlite3, leveldb, hanoidb, couchdb)
- can replicate with Apache CouchDB
- designed for embedded device
- works with coffer
- used as a basic indexer

simple api

```
create_db(DbName, [{backend, Name}, {default_blob_backend} ...])
                -> {ok, Db} | {error, Reason}

open_db(DbName) -> {ok, Db} | {error, Reason}

save_doc(Db, Id, Props) -> {ok, #doc{}} | {error, Reason}.
save_doc(Db, Id, Props, Options) -> {ok, #doc{}} | {error, Reason}.
open_doc(Db, DocId) -> {ok, #doc{}} | {error, Reason}. (always last rev)
```