Started May, 2011

libbitcoin

ohloh.net analysis:
64,115 lines
Estimated Effort:
16 person-years
Estimated Cost:
$864,843

libbitcoin.dyne.org

github.com/spesmilo/libbitcoin 991 commits libbitcoin is an asynchronous library.

operates with components called 'services'.

services take a threadpool as their first argument.

```
threadpool disk_pool(4);  // 4 threads spawned
leveldb_blockchain chain(disk_pool);
```

The dependencies for a service follow the threadpool.

```
threadpool ~~tx~~memop_pool(1);  // 1 thread spawned.
transaction_pool txp(memop_pool, chain);
```

Currying is fundamental to libbitcoin.

Currying takes a function and changes its signature.

This is how we make libbitcoin asynchronous ~~[illegible]~~ and modular.

~~Old approach:~~

~~class FooWidget:~~

~~def onClick(self, event):~~

~~... blaa~~

~~def~~

Old approach:

```
~~class IRCRobot~~
class IrcBot:

        def on_connect(self, ~~event~~ ...):
            # join a channel
            ...

        def on_join(self, ...):
            pass

        def on_receive_message(self, channel, user, message)
            # do something
            send (reply)
```

Problems:

INFLEXIBLE DESIGN.

- You must use their class layouts.
- Their flavour of OOP.
- keep track of temporary variables between method calls (somehow).
- And synchronise shared values.

MACAO ✓
HAGUIS ✓
URBAN CENTER

What is Currying?

Currying transforms functions signatures.

```
~~f~~void f(a, b, c, d)
~~g=f(120, _2, _1, "hello")~~
g = bind(f, 120, _2, _1, "hello")
```

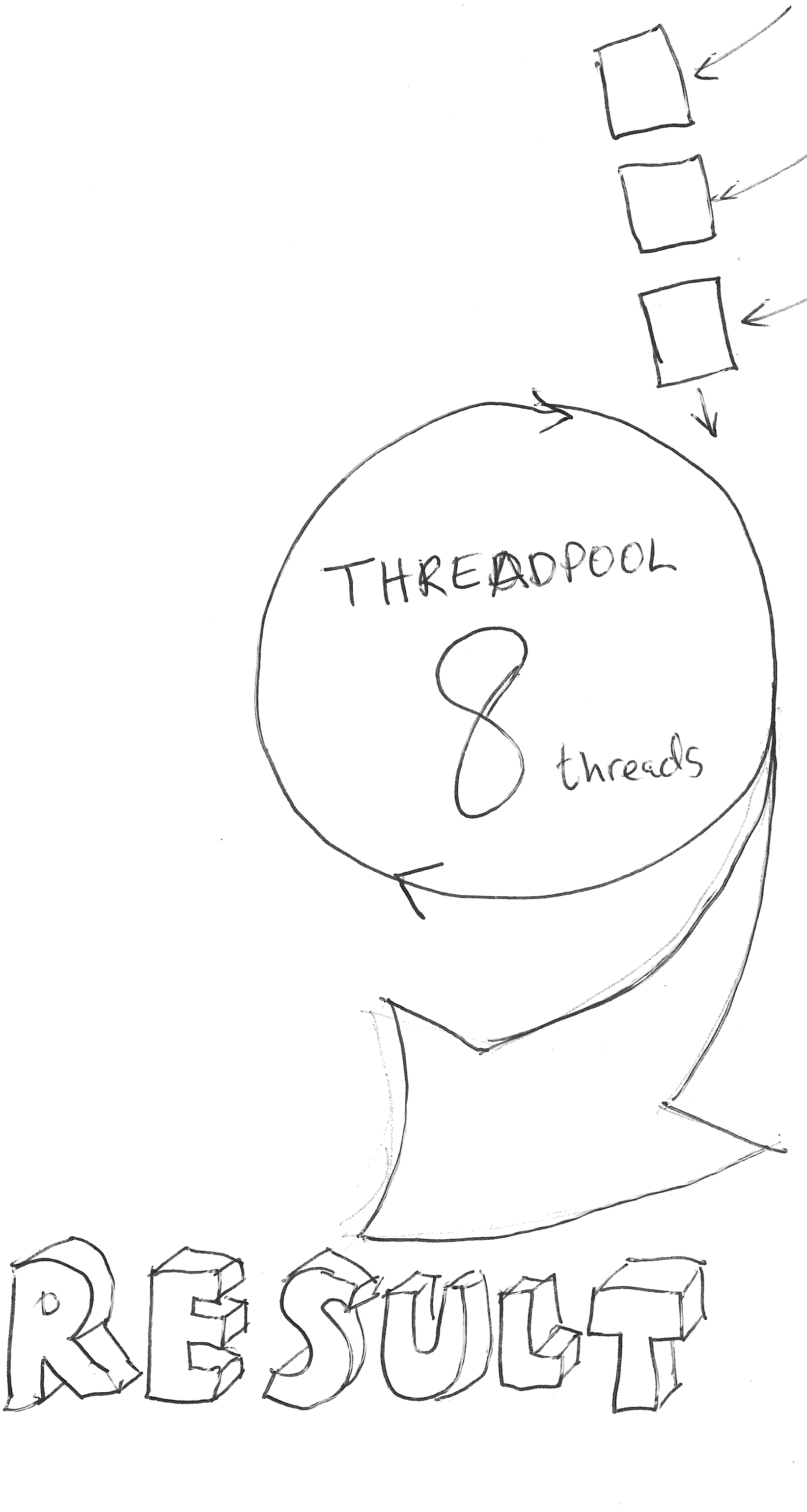
~~Calling~~

```
g(4, foo) → f(120, foo, 4, "hello"
```

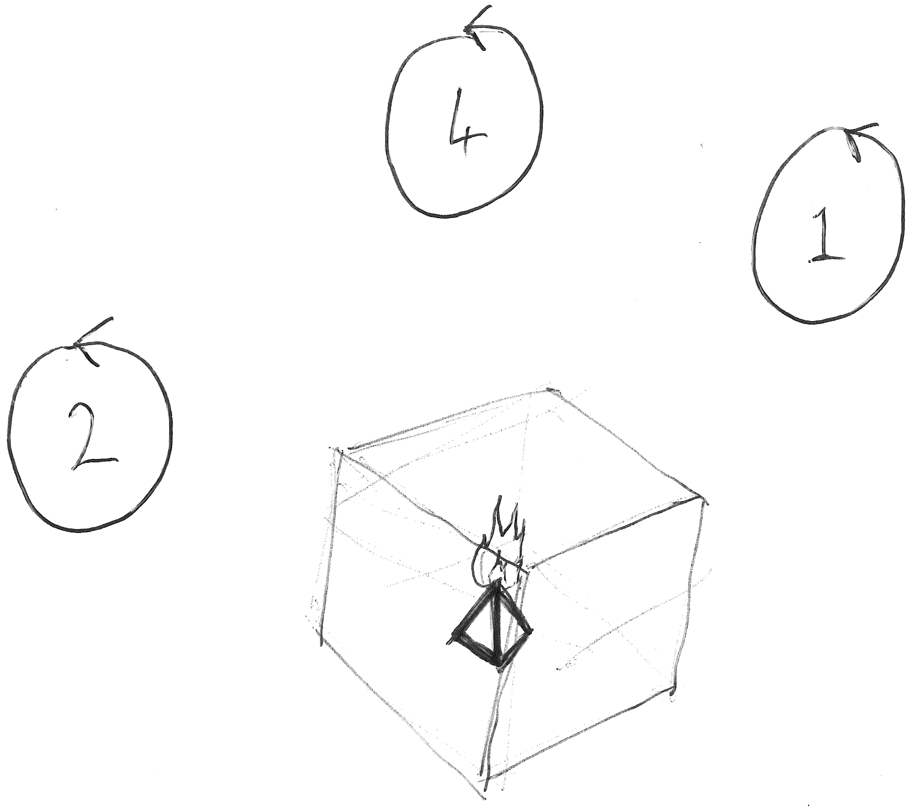
In C++

```
#include <functional>
using std::placeholders::_1;
using std::placeholders::_2;

...

void f(int a, Object b, int c, string d);

g = std::bind(f, 120, _2, _1, "hello");

g(4, foo);
```

- Connect functions of different signatures to each other.
- Keep ~~local~~ temporary variables as bounded arguments (in bind).


THREADPOOL
8 threads
RESULT

```
#include <bitcoin/bitcoin.hpp>
using namespace bc;
~~bool stopped = false;~~
void my_function()
{
    std::cout << "Hello World!!!" << std::endl;
    ~~stopped = true;~~
}

int main()
{
    threadpool pool(8);     // 8 threads!
    pool.dispatch(my_function);
    ~~while (!stopped)~~
        ~~sleep(0.1);~~
    ~~pool.stop();          // stop or~~ pool.shutdown();
    pool.join();           // join running threads
                              and wait for them to finish.
    return 0;
}
```



SHARED
STATE

```
#include <bitcoin/bitcoin.hpp>
using namespace bc;

class stupid_example
{
public:
        stupid_example(threadpool& pool)
          : strand_(pool)
        {
        }

        void ~~ad~~ foo_add(int v)
        {
             ~~foo~~ strand_.queue(
                   [this, v]
                   {
                       foo_ += v;
                   });
         }
        void    foo_incr[illegible]()
        {
             strand_.queue(
                 [this]
                 {
                     ++foo_;
                 });
         }
private:
        async_strand strand_;
        int foo = 0;
};
```

CONTINUED...
→
cont.

```
int main()
{
    threadpool pool(2);
    stupid_example example(pool); // But it's an example nonetheless!
   //Returns immediately
   example.foo_add(10);
   //Returns immediately
   example.foo_incr();
   std::cout << "Press enter to shutdown." << std::endl;
   std::cin.get();
   pool.stop();
   pool.join();
   return 0;
}
```

libbitcoin operations take a handler (last argument)

```
foo.do_something(arguments...., handler);
```

~~void handler~~

do something.

then call this

- -

```
void handler(~~const~~ std::error_code ec, arguments...)
{
    ...
}
```

error code as first argument.

handlers differ depending on the different operations

```
std::error_code ec = bc::error::bad_stream;

if (ec == bc::error::bad_stream)
    // handle bad_stream errors.
else if (ec)
    // handle all other errors.
else
{

    // main body
}
```

my usual handler looks like:

```
Void ~~ka~~ Something_happened(std::error_code ec, ...)
{
    if (ec)
    {
        std::cerr << "app: Something failed to happen: "
            << ec.message() << std::endl;
        return;
    }
    // do stuff ...
}
```

WHIRLWIND TOUR

services:

blockchain (pluggable backends.
default is leveldb.
deprecated versions: bdb, postgresql)

transaction-pool
transaction_indexer ← lookup transactions by address.
~~[illegible]~~ remains in sync with pool.

network, acceptor, channel
← connect and accept connections.

protocol ← p2p network. manages connections, seeding, broadcasting, ...
hosts ← ~~[illegible]~~ "list of hosts

handshake ← initial connection handshake (exchange version messages and verack responsees)
poller ← poll network for new blocks.
getx-responder

utilities and types:

- payment_address ← encoding and decoding of Bitcoin addresses.
- script_type ← bitcoin script
- transaction_type, block_type, ...
- base58, ripemd, sha256, minikeys, ...
- serialization using iterators. buffers preallocated:

```
data_chunk rawtx(satoshi_raw_size(tx));
auto end_iter = satoshi_save(tx, rawtx.begin());
BITCOIN_ASSERT(end iter == rawtx.end());
```

encode_hex(data), decode_hex(str), satoshi_to_btc(satoshis)

- magic numbers in ~~include~~ <bitcoin/constants.hpp>
- <bitcoin/block.hpp>, <bitcoin/transaction.hpp>

```
hash_digest hash_block_header(block_header)
hash_digest hash_transaction(tx)
block_type genesis_block();
```

- elliptic_curve_key:
  - new privkey
  - get/set privkey
  - get/set pubkey
  - sign/verify

12 words:

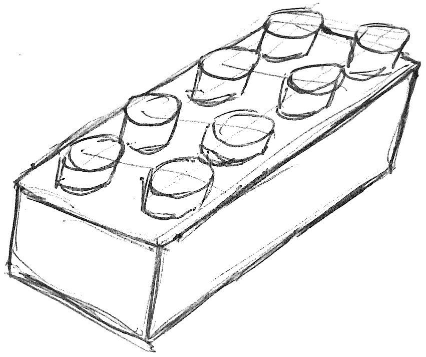
```
words = encode_mnemonic(seed);
seed = decode_mnemonic(words);
```

deterministic_wallet: ← electrum compatible.
- new seed
- get/set seed
- get/set mpk
- ~~genpubkey~~ generate pubkey
- generate privkey (secret)

Validation of blocks and unconfirmed transactions is in <bitcoin/validate.hpp>

Focus:

- Scalability.
- Intuitive.
- Extendable.
- Never block.
- We ♡ UNIX design.



Design:

#1 Simplicity (of implementation)
#2 Correctness (good design)
#3 Consistent (but not if we sacrifice #1 or #2.
less common circumstances are not critical)
#4 Completeness (be practical though).

FRAMEWORK BAD.
TOOLKIT GOOD.

http://libbitcoin.dyne.org/doc

libbitcoin/examples/fullnode.cpp

obelisk

obeli

github.com/spesmilo/obelisk

Obelisk

blockchain server infrastructure:

clients use libobelisk.

```
#include <bitcoin/bitcoin.hpp>
#include <obelisk/obelisk.hpp>

...

threadpool pool(1);
obelisk::fullnode_interface fullnode(
    pool, "tcp://localhost:90Ø1");
fullnode.address.fetch_history(
    address, history_fetched_handler);

...
```

Origin:

libbitcoin fullnode example (300 Lines of code)

✚

Apache Thrift

[redacted]

"framework for scalable cross-language services development"
~ it's website

⇩

blockchain daemon
with network interface.

(github.com/genjix/query)

Apache Thrift:

- Made by Facebook.
- We don't like frameworks.
- Scalable? Don't lie.

ØMQ

"Simplest Way to Connect Pieces"

~ zeromq.org

↳ Click 'Learn'

↳ 'the guide'

Chapter 1 - Fixing the world

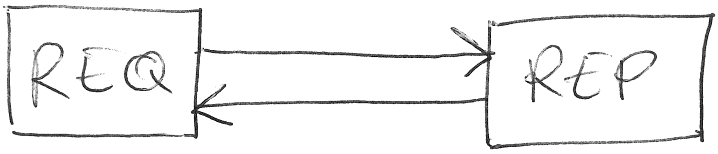
"We can leave the political philosophy
for another book.

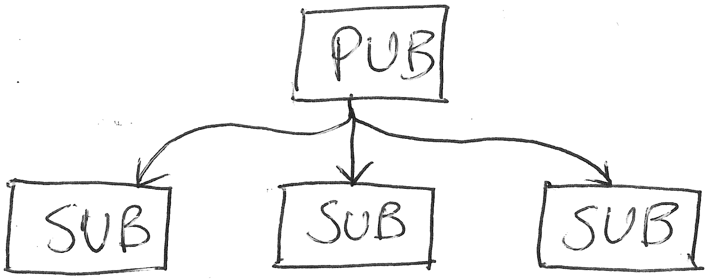
↳ softwareandsilicon.com

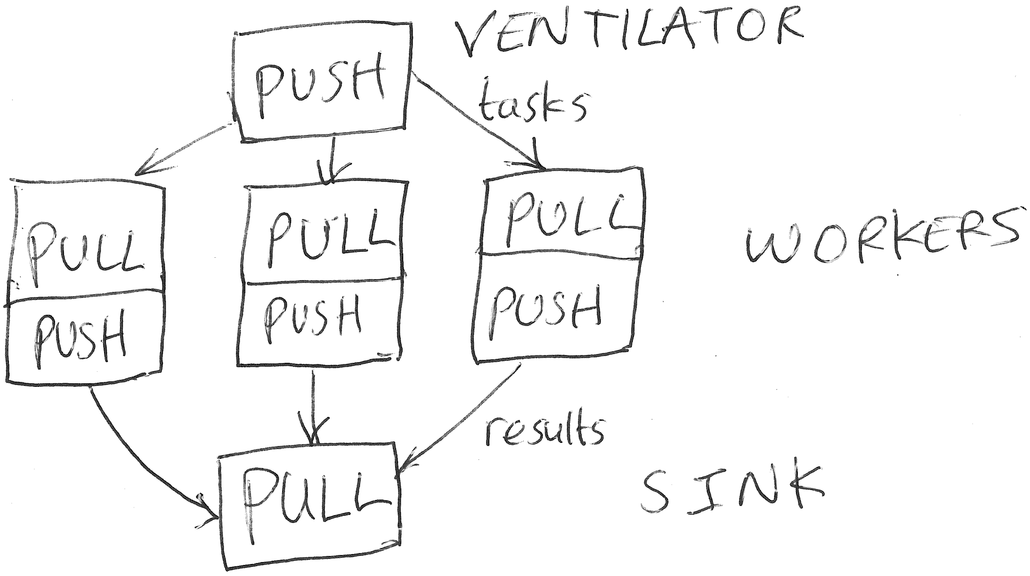
The ZeroMQ library author, Pieter Hintjens,
is a genius.

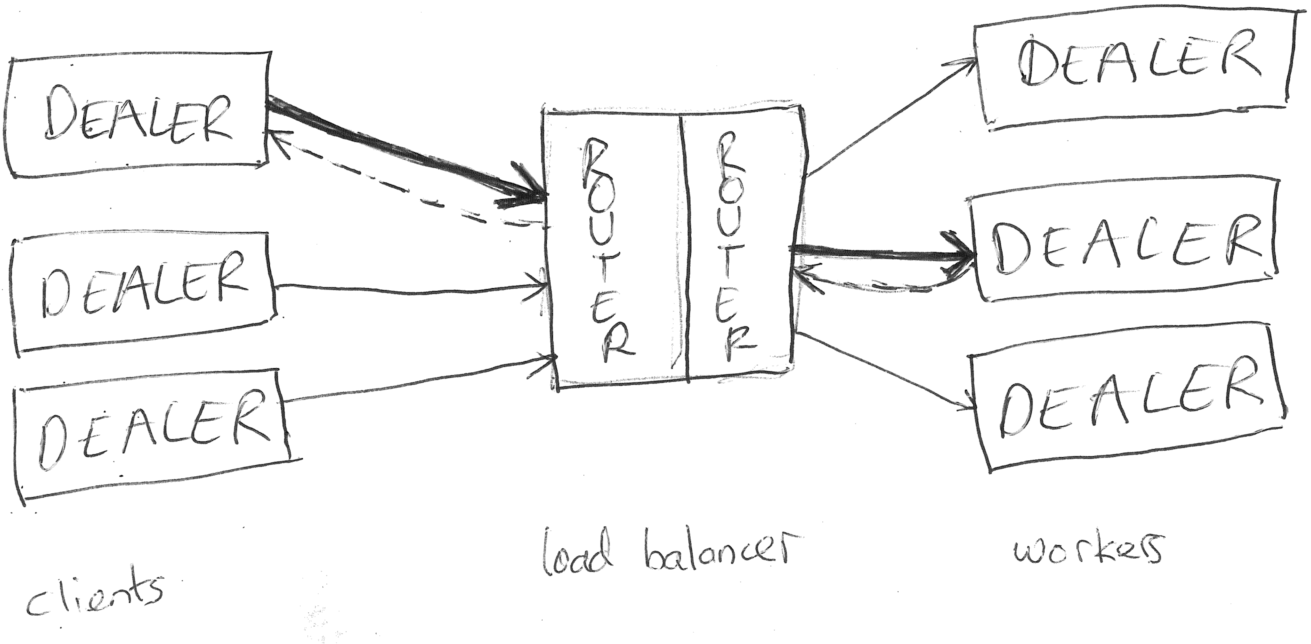
hintjens.com/blog:17

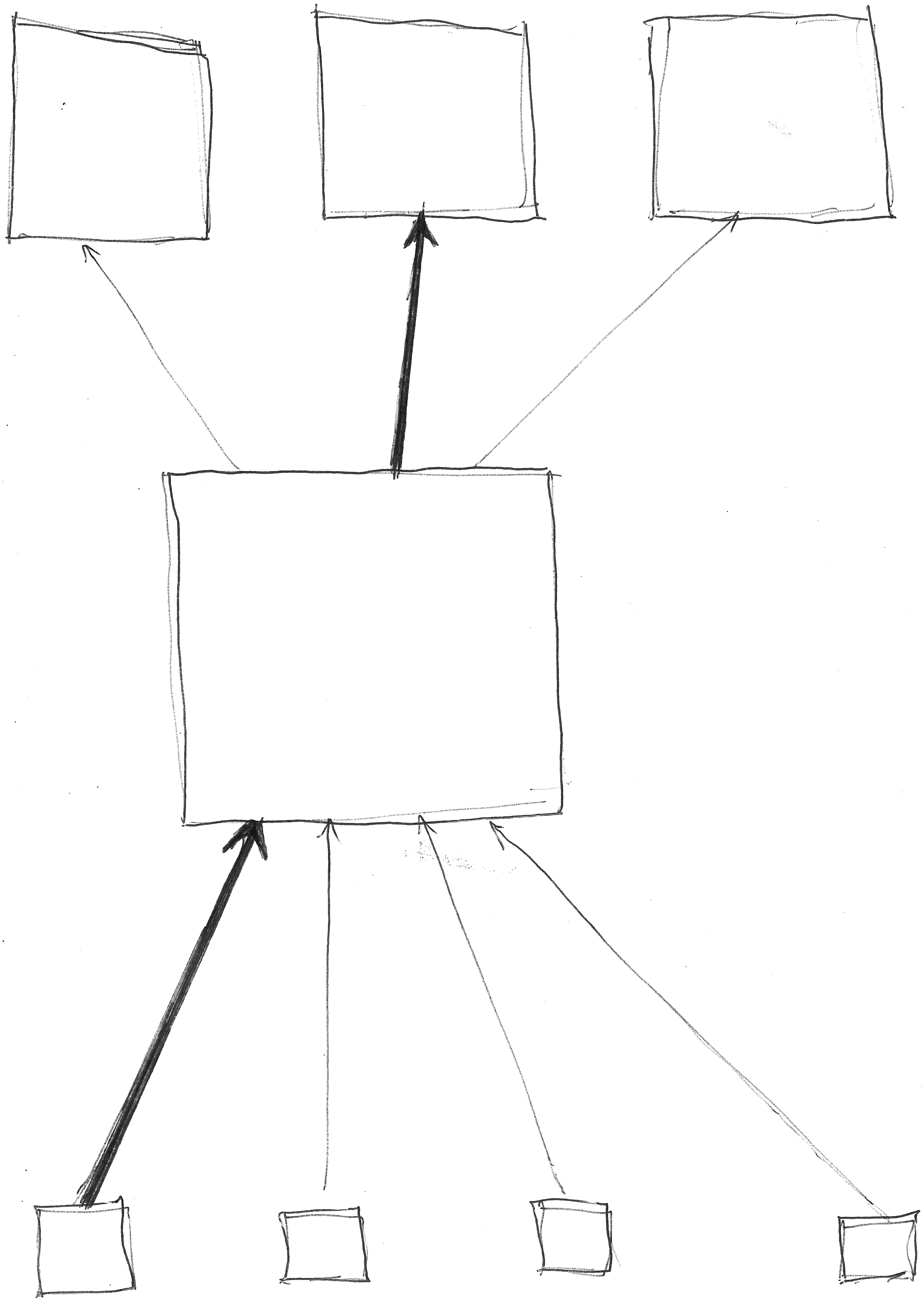
ZeroMQ = A Few Basic Building Blocks











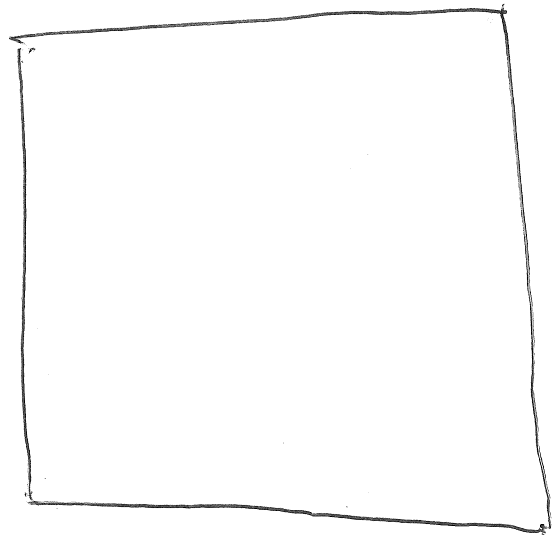
□

□

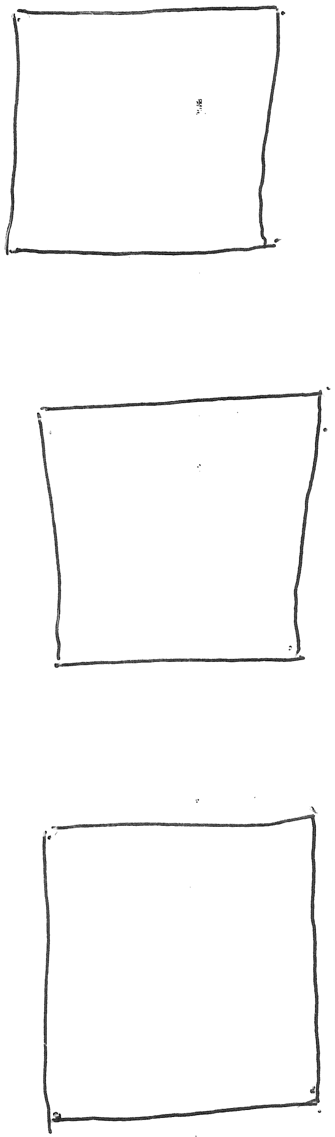
□

□

WALLETS



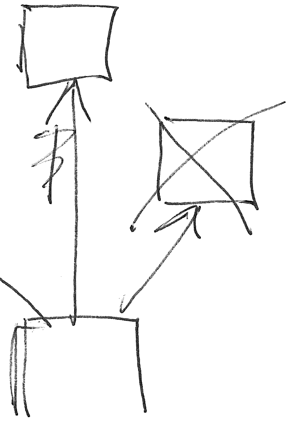
BALANCER



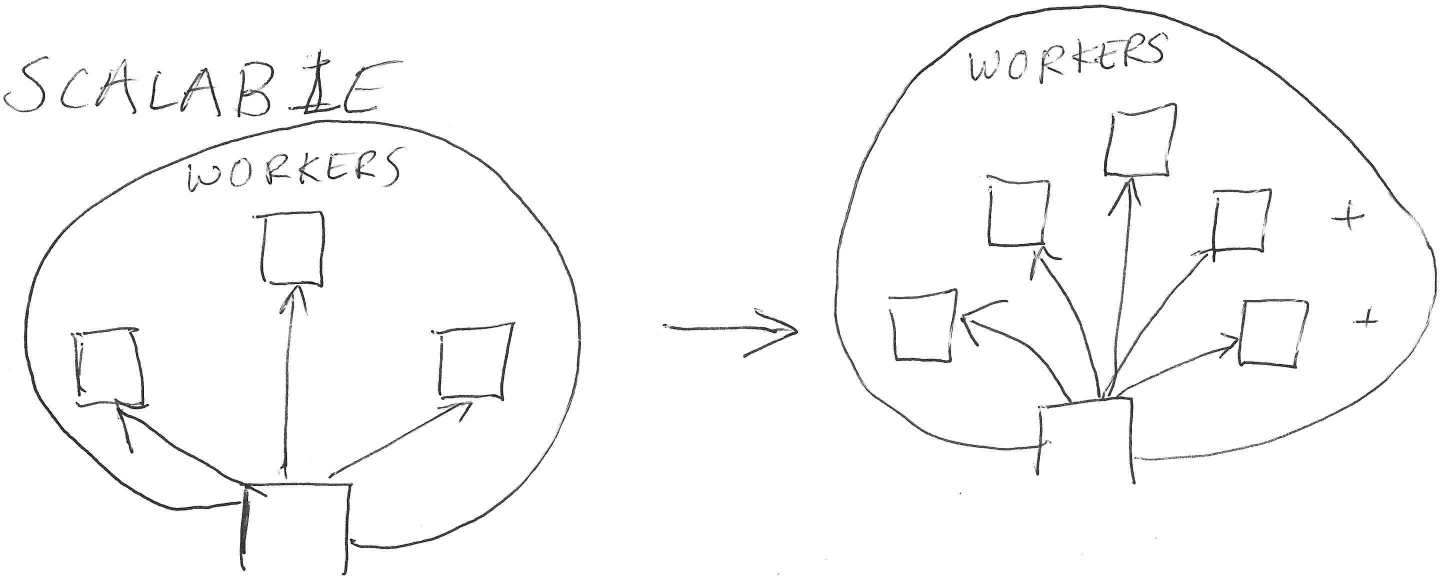
BLOCKCHAIN WORKERS

BITCOIN NODES.

REDUNDANCY



SCALABLE



1 BLOCKCHAIN WORKER =

FULLNODE BITCOIN DAEMON.

HOSTS NO KEYS.

FULL BLOCK VALIDATION.

ASYNCHRONOUS BITCOIN IMPLEMENTATION.

SX

sx.dyne.org

github.com/spesmilo/sx

libbitcoin:

MORE POWER TO DEVELOPERS...

...

Sx:

AND ADMINS!

Give people the building blocks and they will make stuff.

Sx possibilities:

- offline transactions.
- multisignature.
- QR codes.
- deterministic wallets.
- embed file hashes in blockchain.
- commands for querying obelis blockchain, working with transactions (show, validate, broadcast)
- many possibilities.
- ~~experi~~ prototype ~~ncurses~~ terminal wallet ← ncurses.

```
$ wget http://sx.dyne.org/install-sx.sh
$ sudo bash install-sx.sh
```

OR

```
$ bash install-sx-sh INSTALLPREFIX/
```

http://sx.dyne.org

```
$ sx help
```

↓ list of commands

```
$ sx help COMMAND
```

```
sx COMMAND [ARGS]...
```