SPLAY

Distributed Systems Made Simple

Pascal Felber, Lorenzo Leonini, Etienne Rivière,
Valerio Schiavoni, José Valerio

Lua Workshop, 8 September 2011
Frick, Switzerland

Institut d'informatique
Université de Neuchâtel

SPLAY

UNIVERSITÉ DE
NEUCHÂTEL

About Me

- 2010-now : PhD student at the University of Neuchatel, Switzerland (and Lua user since then)
  - Topic: large-scale distributed systems, cloud computing. Daily job:
    1. Invent new protocols for cloud applications
    2. Do experiments, write papers, go to 1
- 2007-2009: Research engineer at INRIA, France
- BSc and MSc in Computer Engineering at the Università degli Studi Roma Tre, Italy

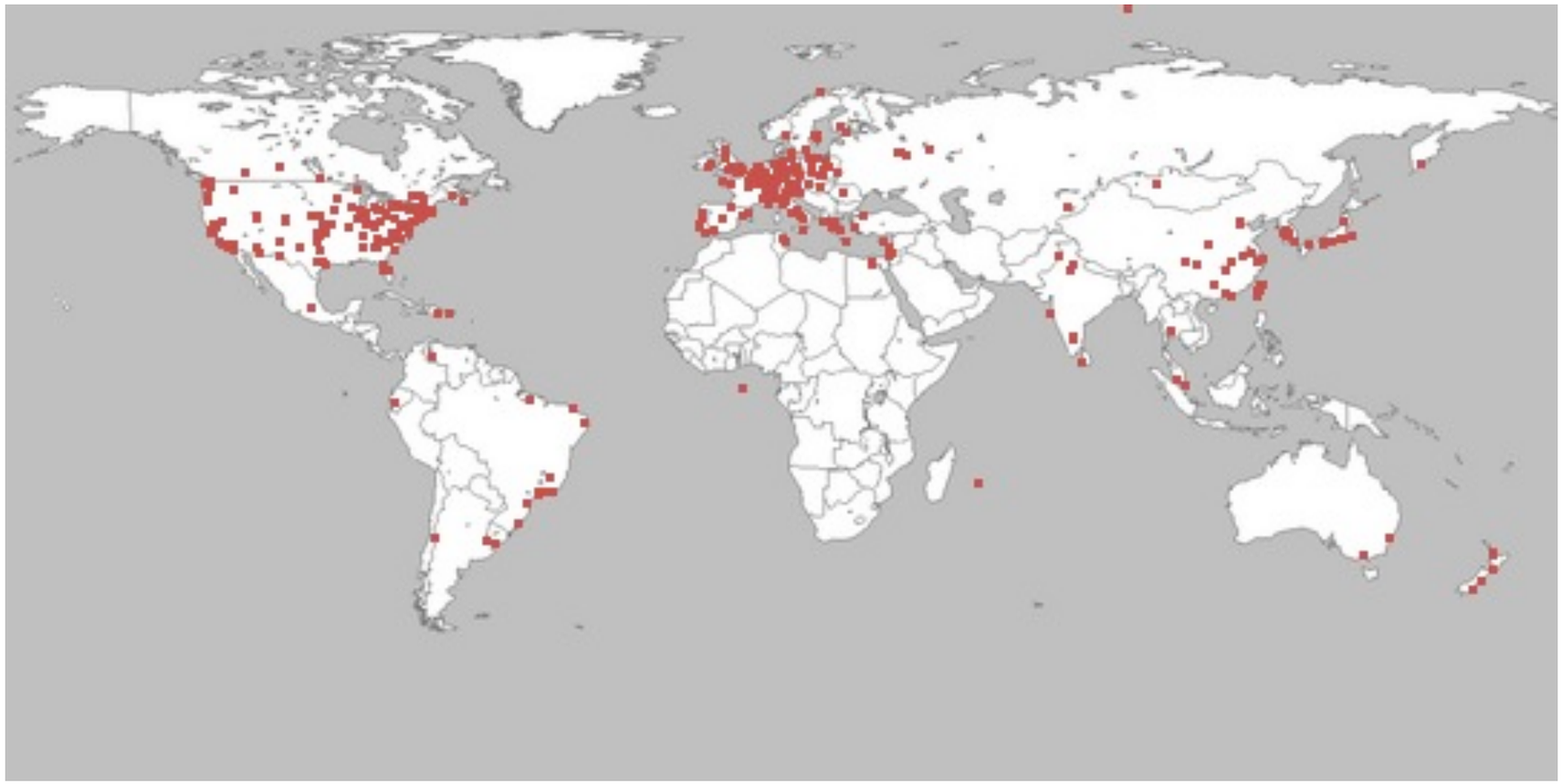
Motivations

Motivations

- Developing, testing and tuning distributed applications is hard
- In Computer Science research, fixing the gap of simplicity between pseudocode description and implementation is hard
- Using worldwide testbeds is hard

What is  PLANETLAB

What is PLANETLAB



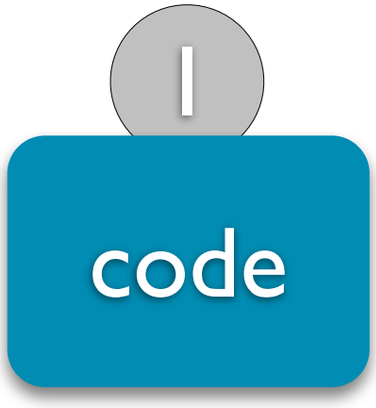
What is PLANETLAB

- Machines contributed by universities, companies,etc.
  - 1098 nodes at 531 sites (02/09/2011)
  - Shared resources, no privileged access
- University-quality Internet links
- High resource contention

- Faults, churn, packet-loss is the norm
  - Challenging conditions

Daily Job With Distributed Systems

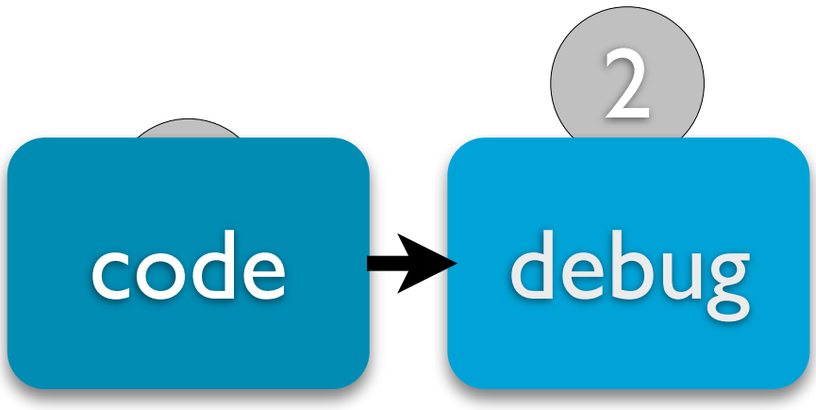
Daily Job With Distributed Systems



1

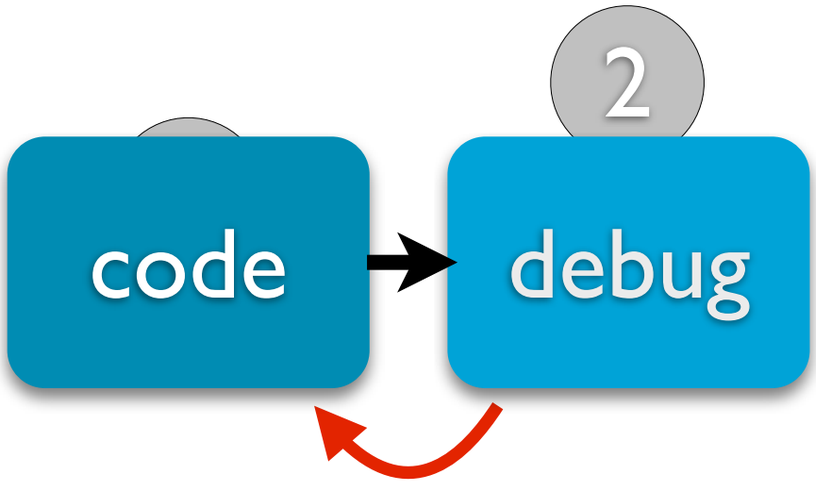
- Write (testbed specific) code
  - Local tests, in-house cluster, PlanetLab...

Daily Job With Distributed Systems



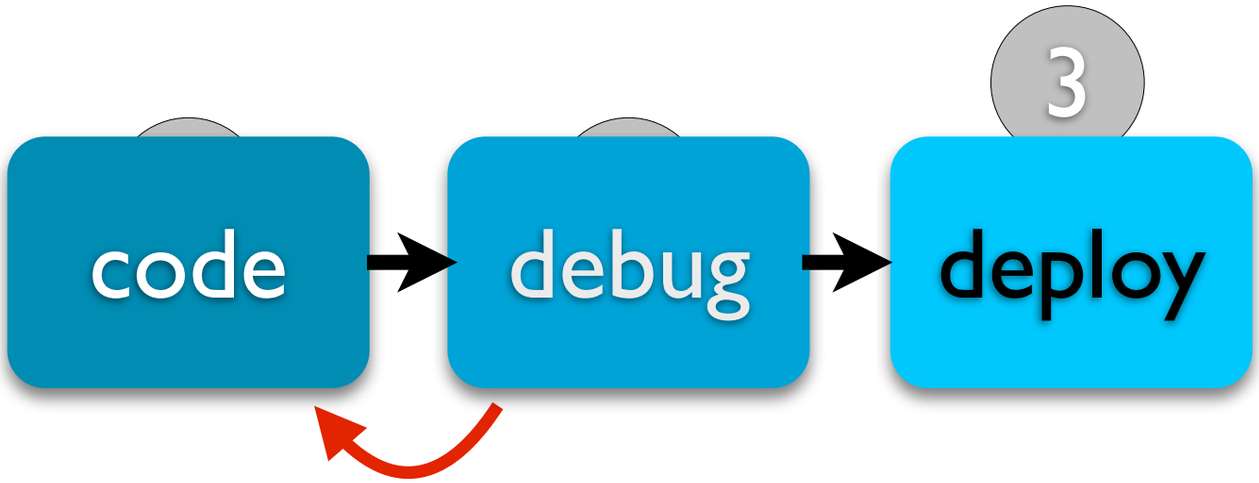
2 • Debug (in this context, a nightmare)

Daily Job With Distributed Systems



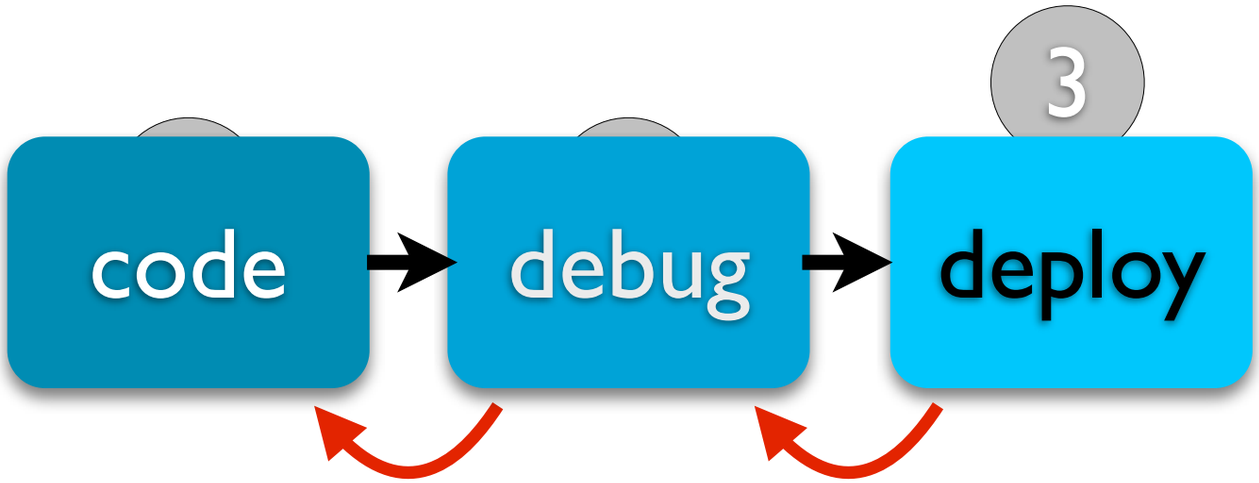
2 • Debug (in this context, a nightmare)

Daily Job With Distributed Systems



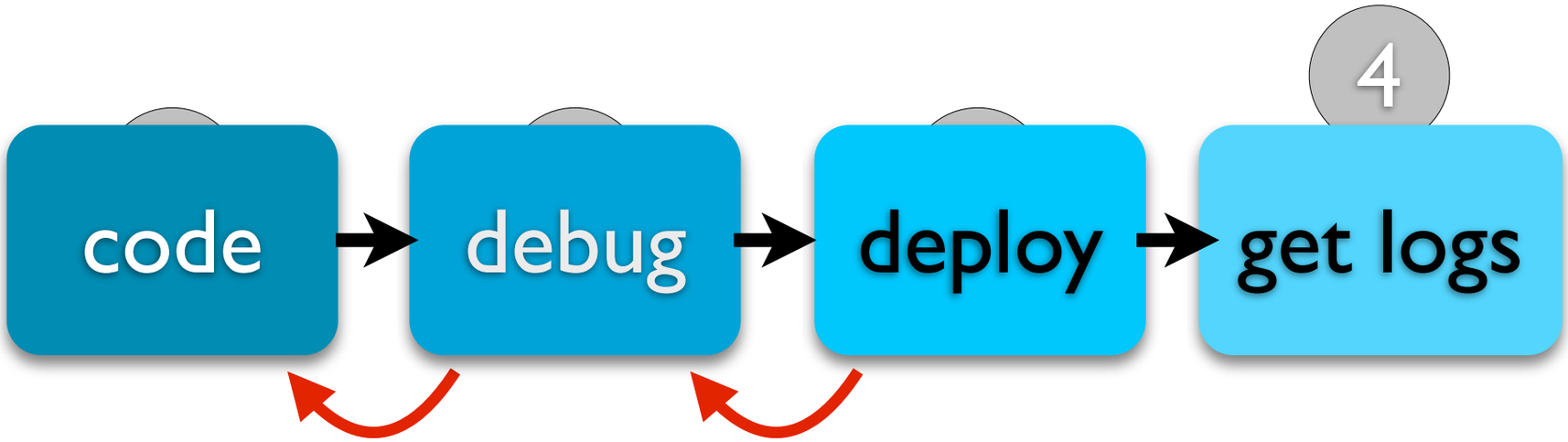
3 • Deploy, with testbed specific scripts

Daily Job With Distributed Systems



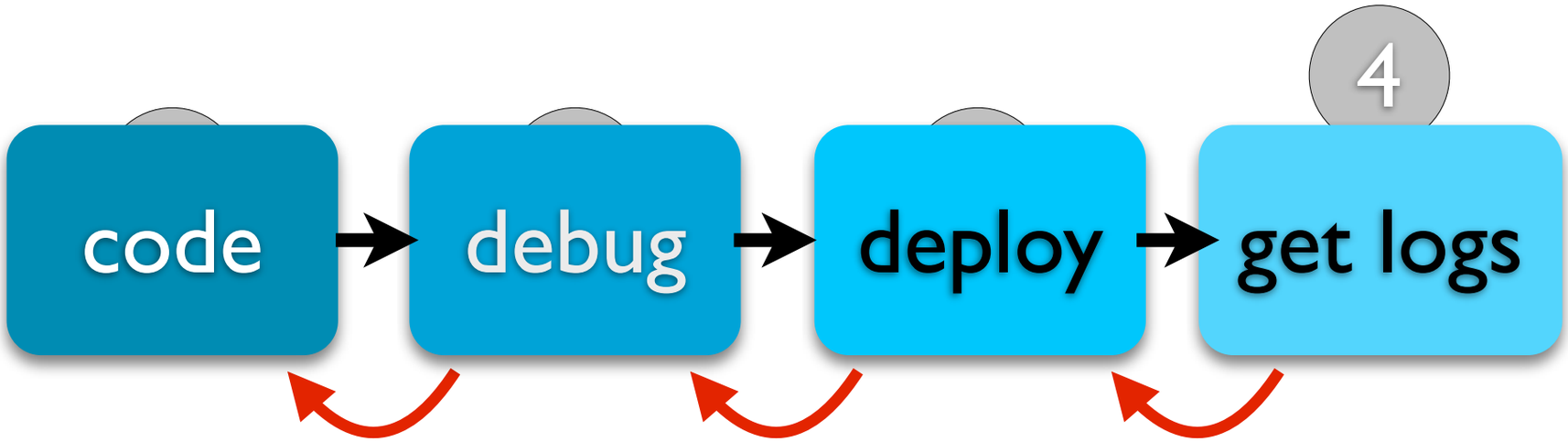
3 • Deploy, with testbed specific scripts

Daily Job With Distributed Systems



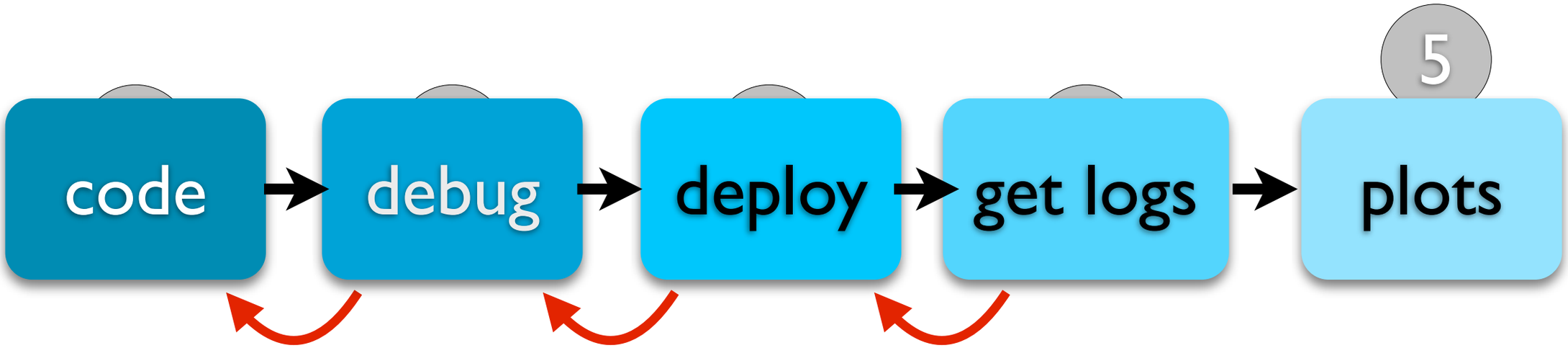
4 • Get logs, with testbed specific scripts

Daily Job With Distributed Systems



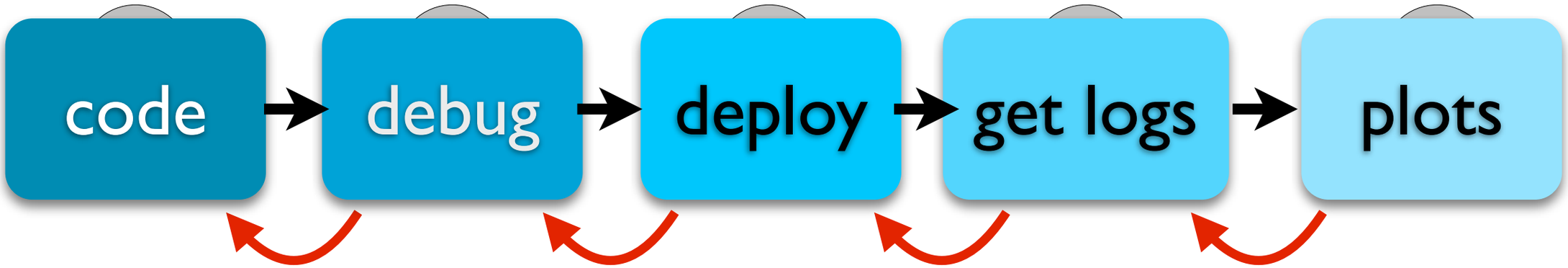
4 • Get logs, with testbed specific scripts

Daily Job With Distributed Systems



5 • Produce plots, hopefully

Daily Job With Distributed Systems

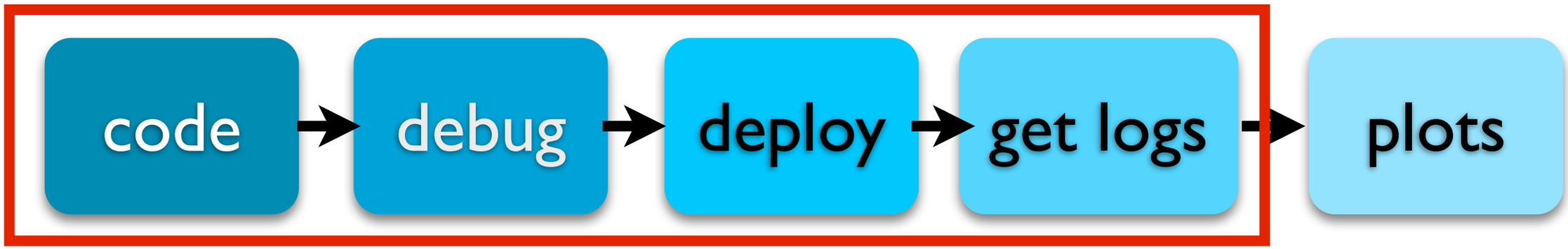


5 • Produce plots, hopefully

SPLAY At Glance

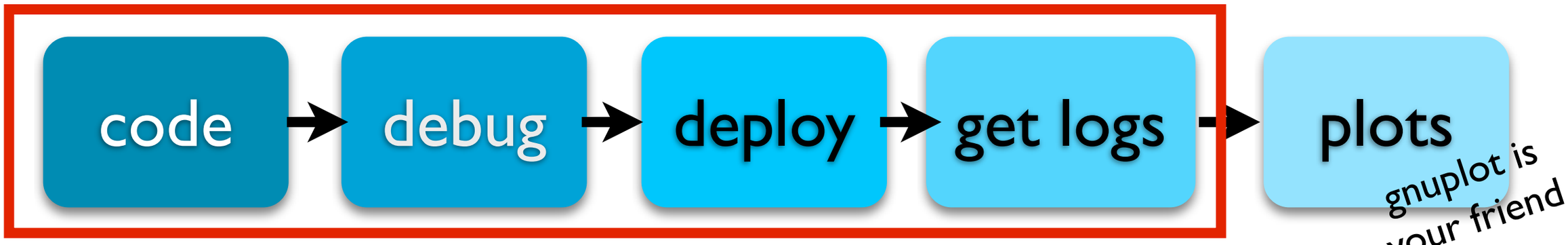
- Supports the development, evaluation, testing, and tuning of distributed applications on any testbed:
  - In-house cluster, shared testbeds, emulated environments
- Provides an easy-to-use pseudocode-like language implemented in Lua

SPLAY At Glance



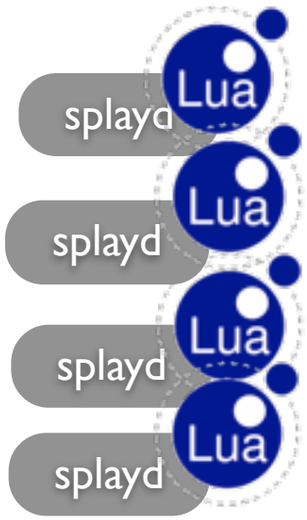
- Supports the development, evaluation, testing, and tuning of distributed applications on any testbed:
  - In-house cluster, shared testbeds, emulated environments
- Provides an easy-to-use pseudocode-like language implemented in Lua

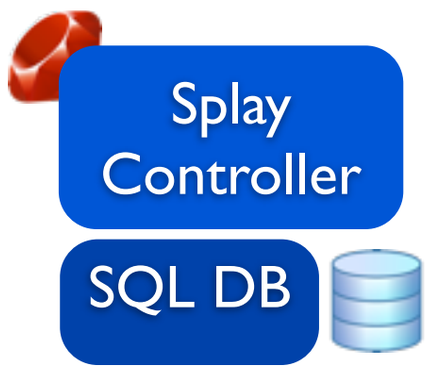
SPLAY At Glance



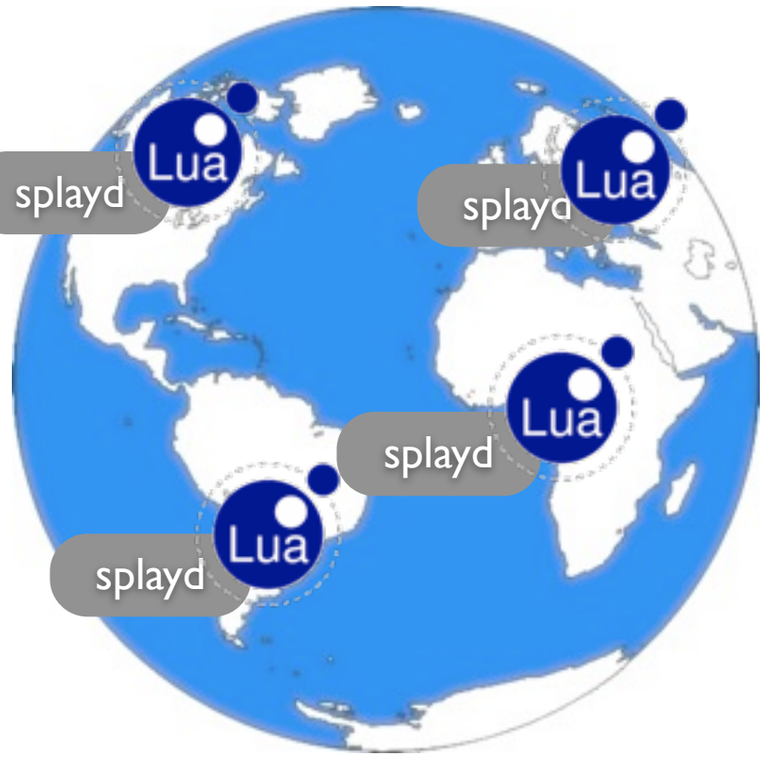
- Supports the development, evaluation, testing, and tuning of distributed applications on any testbed:
  - In-house cluster, shared testbeds, emulated environments
- Provides an easy-to-use pseudocode-like language implemented in Lua

The Big Picture





The Big Picture

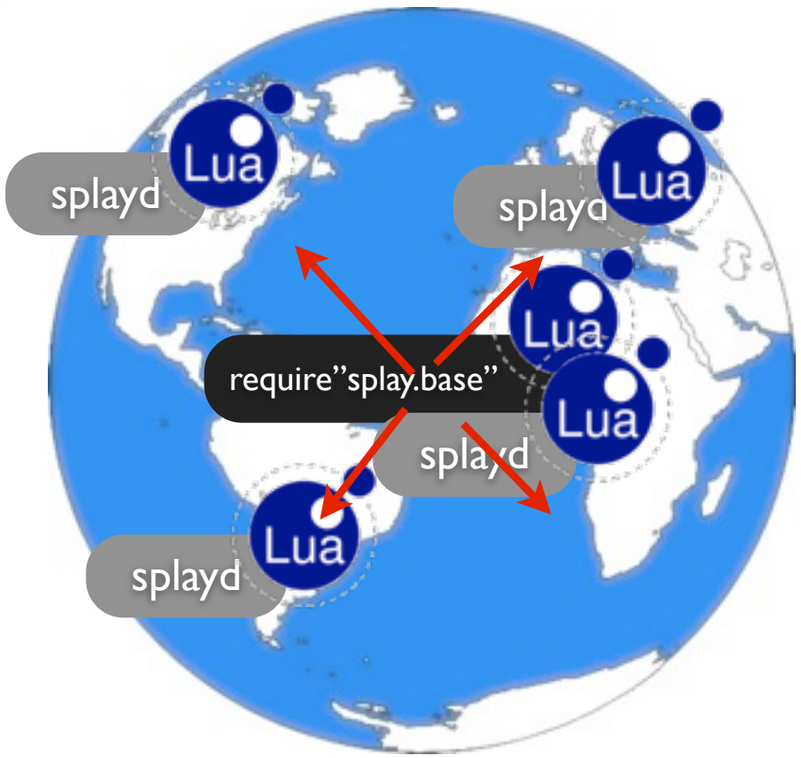


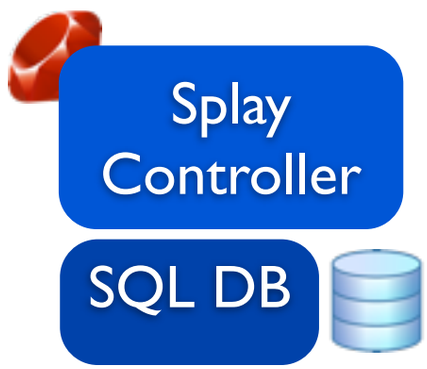
require"splay.base"

Splay Controller

SQL DB

The Big Picture





Why  ?

Why  ?

- Light & Fast
  - (Very) Close to equivalent code in C

Why  ?

- Light & Fast
  - (Very) Close to equivalent code in C
- Concise
  - Allow developers to focus on ideas more than implementation details
  - Key for researchers

Why  ?

- Light & Fast
  - (Very) Close to equivalent code in C
- Concise
  - Allow developers to focus on ideas more than implementation details
  - Key for researchers
- Sandbox thanks to the possibility of easily redefine (even built-in) functions

Concise

Concise

Pseudo code as published on original paper ←

```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    n' = closest_preceding_node(id);
    return n'.find_successor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
  for i = m downto 1
    if (finger[i] ∈ (n, id))
      return finger[i];
  return n;

// create a new Chord ring.
n.create()
  predecessor = nil;
  successor = n;

// join a Chord ring containing node n'.
n.join(n')
  predecessor = nil;
  successor = n'.find_successor(n);

// called periodically. verifies n's immediate
// successor, and tells the successor about n.
n.stabilize()
  x = successor.predecessor;
  if (x ∈ (n, successor))
    successor = x;
  successor.notify(n);

// n' thinks it might be our predecessor.
n.notify(n')
  if (predecessor is nil or n' ∈ (predecessor, n))
    predecessor = n';

// called periodically. refreshes finger table entries.
// next stores the index of the next finger to fix.
n.fix_fingers()
  next = next + 1;
  if (next > m)
    next = 1;
  finger[next] = find_successor(n + 2^(next-1));

// called periodically. checks whether predecessor has failed.
n.check_predecessor()
  if (predecessor has failed)
    predecessor = nil;
```

Executable code using SPLAY libraries →

```
require"splay.base"
rpc = require"splay.rpc"
between, call, thread, ping = misc.between_c, rpc.call, events.thread, rpc.ping
n, predecessor, finger, timeout, m = {}, nil, {}, 5, 24
function join(n0) -- n0: some node in the ring
    predecessor = nil
    finger[1] = call(n0, {'find_successor', n.id})
    call(finger[1], {'notify', n})
end
function closest_preceding_node(id)
    for i = m, 1, -1 do
        if finger[i] and between(finger[i].id, n.id, id) then
            return finger[i]
        end
    end
    return n
end
function find_successor(id)
    if finger[1].id == n.id or between(id, n.id, (finger[1].id + 1) % 2^m) then
        return finger[1]
    else
        local n0 = closest_preceding_node(id)
        return call(n0, {'find_successor', id})
    end
end
function stabilize()
    local x = call(finger[1], 'predecessor')
    if x and between(x.id, n.id, finger[1].id) then
        finger[1] = x -- new successor
        call(finger[1], {'notify', n})
    end
end
function notify(n0)
    if n0.id ~= n.id and
       (not predecessor or between(n0.id, predecessor.id, n.id)) then
        predecessor = n0
    end
end
function fix_fingers()
    refresh = (refresh and (refresh % m) + 1) or 1 -- 1 <= next <= m
    finger[refresh] = find_successor((n.id + 2^(refresh - 1)) % 2^m)
end
function check_predecessor()
    if predecessor and not rpc.ping(predecessor) then
        predecessor = nil
    end
end
n.id = math.random(1, 2^m)
finger[1] = n
if job then
    n.ip, n.port = job.me.ip, job.me.port
    thread(function() join({ip = "192.42.43.42", port = 20000}) end)
else
    n.ip, n.port = "127.0.0.1", 20000
    if arg[1] then n.ip = arg[1] end
    if arg[2] then n.port = tonumber(arg[2]) end
    if not arg[3] then
        print("RDV")
    else
        print("JOIN")
        thread(function() join({ip = arg[3], port = tonumber(arg[4])}) end)
    end
end
```

Sandbox: Motivations

- Experiments should access only their own resources
  - Required for non-dedicated testbeds
    - In universities, companies
    - Totally available at night/holiday time

- Memory-allocation, filesystem, network resources are restricted

Sandboxing LuaSocket

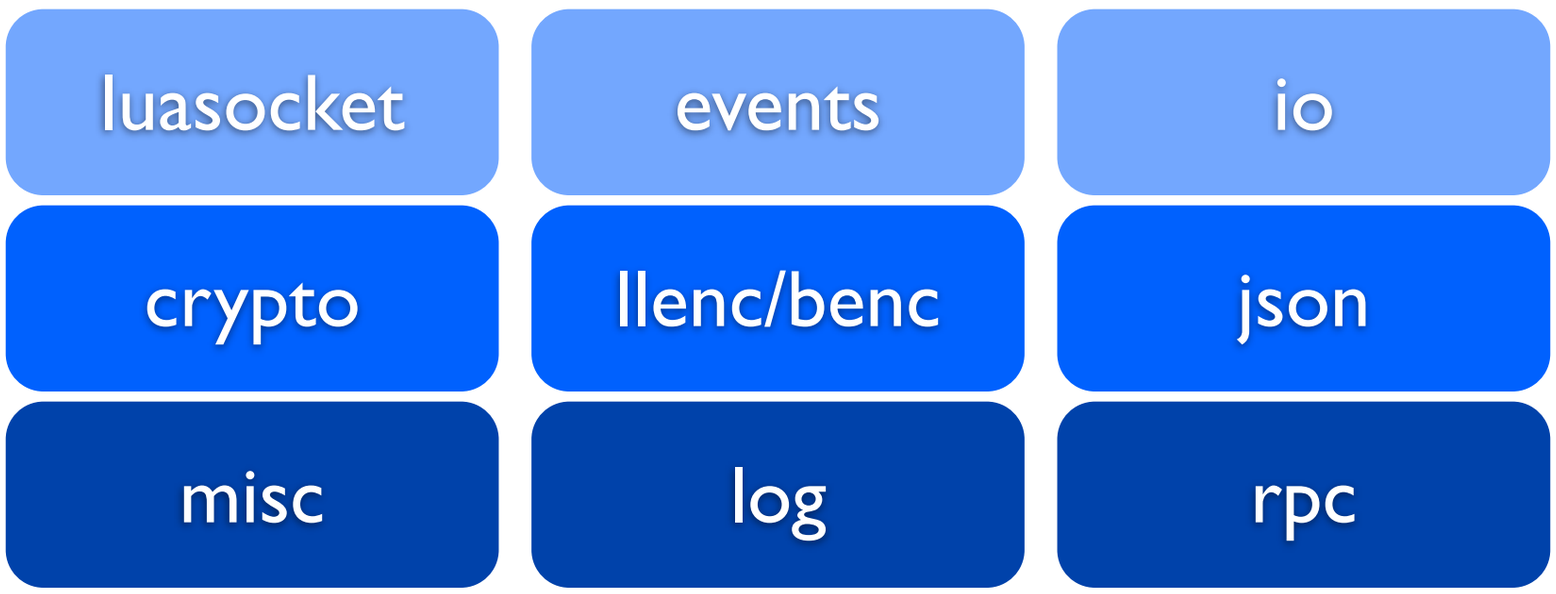
- Same API as plain LuaSocket
- On-demand sandboxed sockets
  - Very easy thanks to Lua's metatable
- Limits chosen by the SPLAYd deployer
- Both TCP and UDP
- Example: UDP's socket.send

Sandboxing LuaSocket

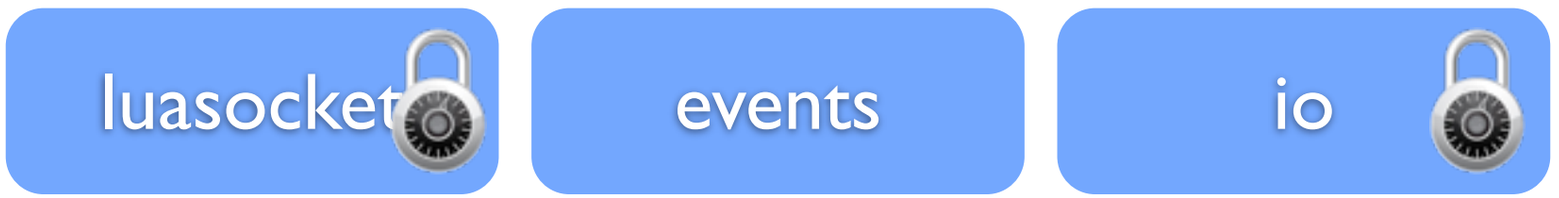
```lua
if sock.send then
        new_sock.send = function(self, data)
            local len = #data
            if total_sent + len > max_send then
                l_o:warn("Send restricted (total: "..total_sent..")")
                return nil, "restricted"
            end
            local n, status
            if math.random(1000) > udp_drop_ratio then
                n, status = sock:send(data)
            else
                n = len
            end
            if n then
                total_sent = total_sent + len
            end
            return n, status
        end
    end
```

```
for _,module in pairs(splay)
```

```
for _,module in pairs(splay)
```

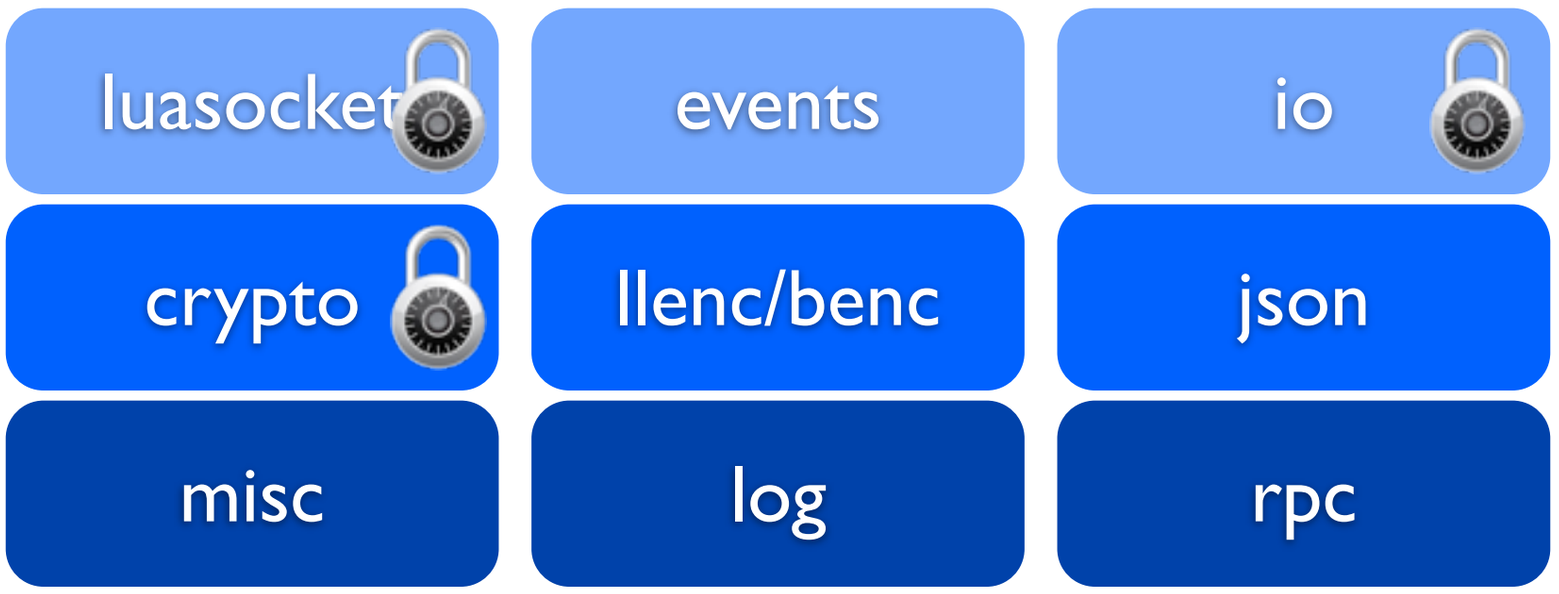


```
for _,module in pairs(splay)
```



Modules sandboxed to prevent access to sensible resources

splay.events



Modules sandboxed to prevent access to sensible resources

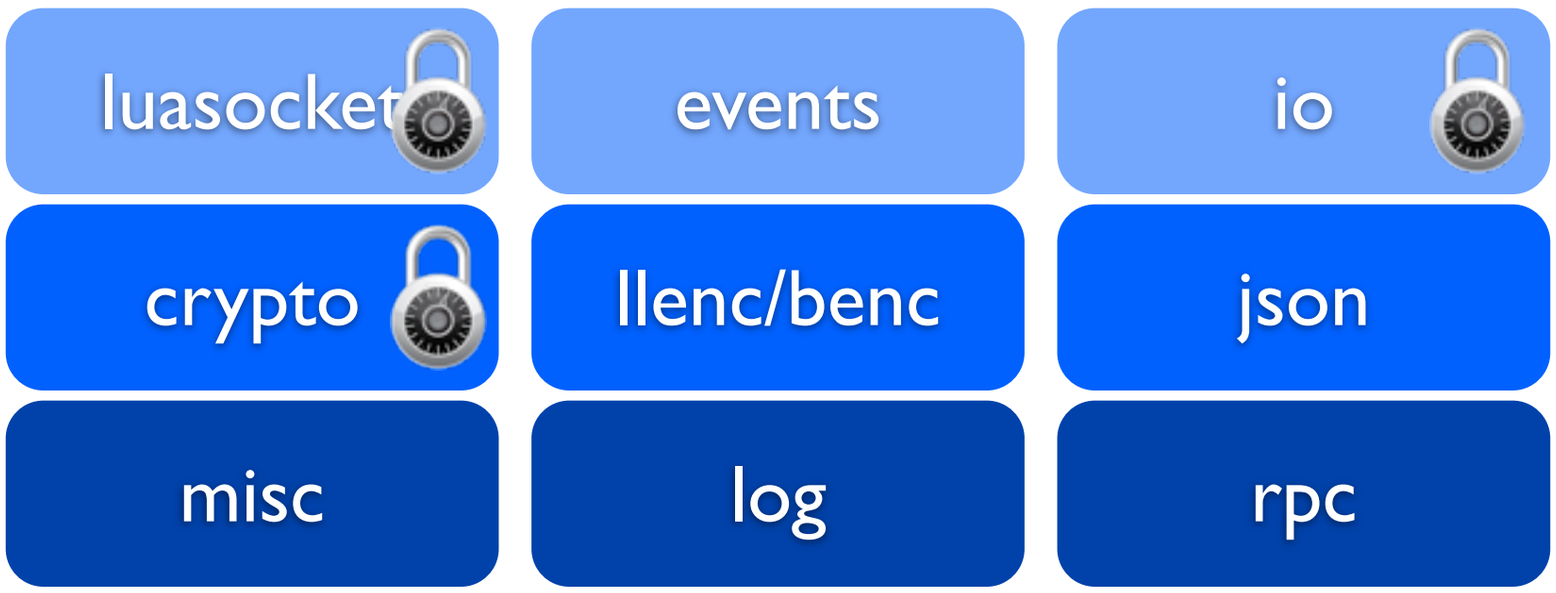
splay.events

events

splay.events

- Distributed protocols can use message-passing paradigm to communicate
- Nodes react to events
  - Local, incoming, outgoing messages
- The core of the Splay runtime
  - `splay.socket`, `splay.io` designed to provide a non-blocking operational mode
- Based on Lua's coroutines

splay.rpc



splay.rpc

rpc

splay.rpc

- Default mechanism to communicate between nodes
- Support for UDP/TCP
- Efficient BitTorrent-like encoding
- Experimental binary encoding

rpc

Life Before SPLAY

- Time spent on developing testbed specific protocols
- Or fallback on simulations...
- The focus should be on ideas
- Researchers usually have no time to produce industrial-quality code

```
public void declareNeighbourInactive(final DHTNode node) {
  // DO NOTHING HERE:
}

public void addToLeafSet(final DHTNode node) {
  // DO NOTHING
}

/**
 * Empty UDP message is 64 bytes, to which we must add 1 bytes for the header,
 * 1 bytes for the messageId, 28 bytes (IP,port,Bamboo ID) for message src, 28
 * bytes for message dest + bytes for the specific data carried in the
 * message. NOTE: the size of informations to ack a message is 10 bytes,
 */
@Override
public long calculateMessageBytes(final Message msg) {
  long result = 122; // 64 + 1 + 1 + 28 + 28;

  if (msg.header == MessageType.PING) {
    /* nothing to add */
    return result;
  }

  if (msg.header == MessageType.PONG) {
    /* some bytes for the informations about which message is being acked */
    result += ACK_INFORMATIONS_SIZE;
    return result;
  }

  if (msg.header == MessageType.BAMBOO_JOIN_LOOKUP_REQUEST
      || msg.header == MessageType.BAMBOO_LOOKUP_REQUEST
```

Life Before SPLAY

- Time spent on developing testbed specific protocols
- Or fallback on simulations...
- The focus should be on ideas
- Researchers usually have no time to produce industrial-quality code

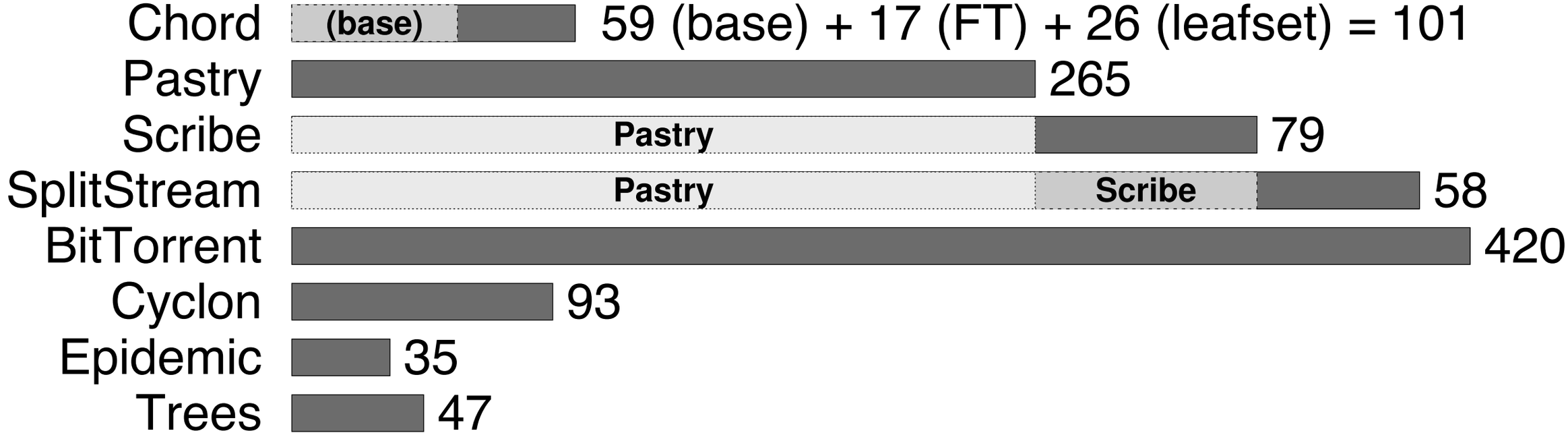
```
responseCallback.onResponseReceived(msg);
}/*
  * CAN BE COMMENTED AS SEND RETRIAL OF MESSAGES IS OFF else { it is a
  * duplicate lookup response, send a PONG to (hopefully) stop source to
  * send us again that message final Message pong = new
  * Message(MessageType.PONG, msg.messageId, this, msg.src, msg.ackedMsg
  * = msg; simulator.send(pong); }
  */
}
/* ELSE IF IS SOMETHING NOT A LOOKUP RESPONSE */
else {

 final Message beingAcked = msg.ackedMsg;
 final ResponseArrivedCallback handler = this.sentMessagesCallbacks
     .remove(beingAcked);

 /* if there is a mapping */
 if (handler != null) {
  handler.onResponseReceived(msg);
 }/*
  * else { this may happen in case of a response arrive later than
  * expected... log .info("%%% " + this + " received the response => " +
  * msg +
  * " but no handler was expecting it! So, simply send back a PONG to make it stop sending again..\n"
  * ); send a PONG to (hopefully) stop source to send us again final
  * Message pong = new Message(MessageType.PONG, msg.messageId, this,
  * msg.src, pong.ackedMsg = msg; simulator.send(pong); }
  */
 }

// /* it replied at last, so "reset" the value */
// this.nodeConsecutiveTimeouts.put((DHTNode) msg.src, 0);
```

Life Before SPLAY

- Time spent on developing testbed specific protocols
- Or fallback on simulations...
- The focus should be on ideas
- Researchers usually have no time to produce industrial-quality code

there is more :-(

Life With SPLAY



- Lines of pseudocode ~== Lines of executable code

Live Demo

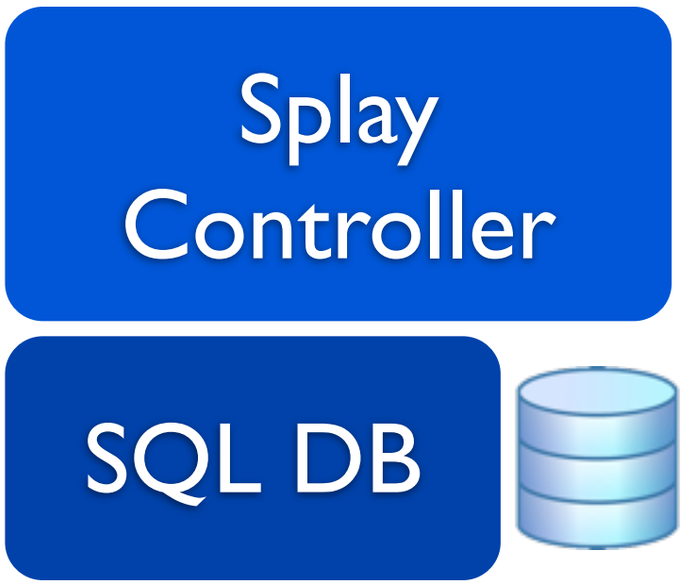
www.splay-project.org

- Distributed systems raise a number of issues related to their evaluation
- Their implementation, debug, deployment and tuning is hard
- SPLAY leverages Lua to produce an easy to use yet powerful working environment to solve these issues

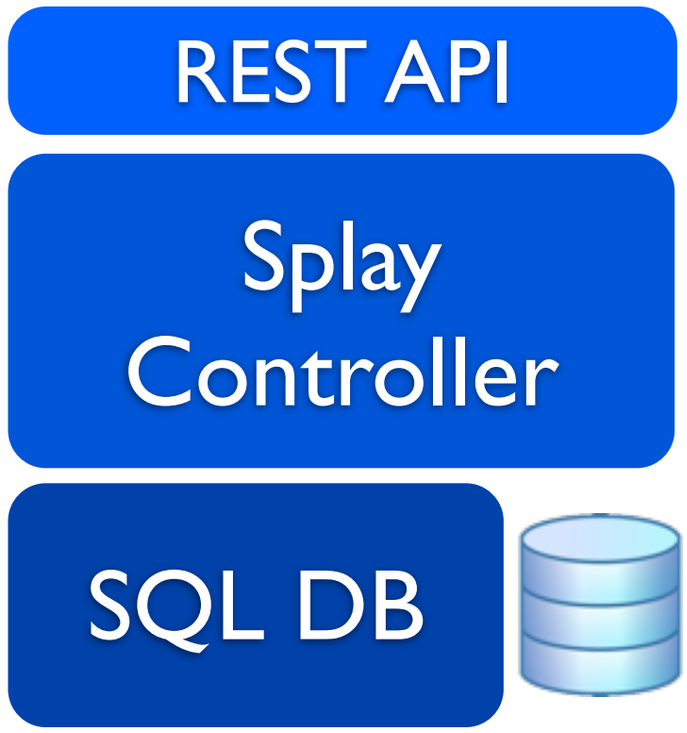
Backup Slides

The Big Picture

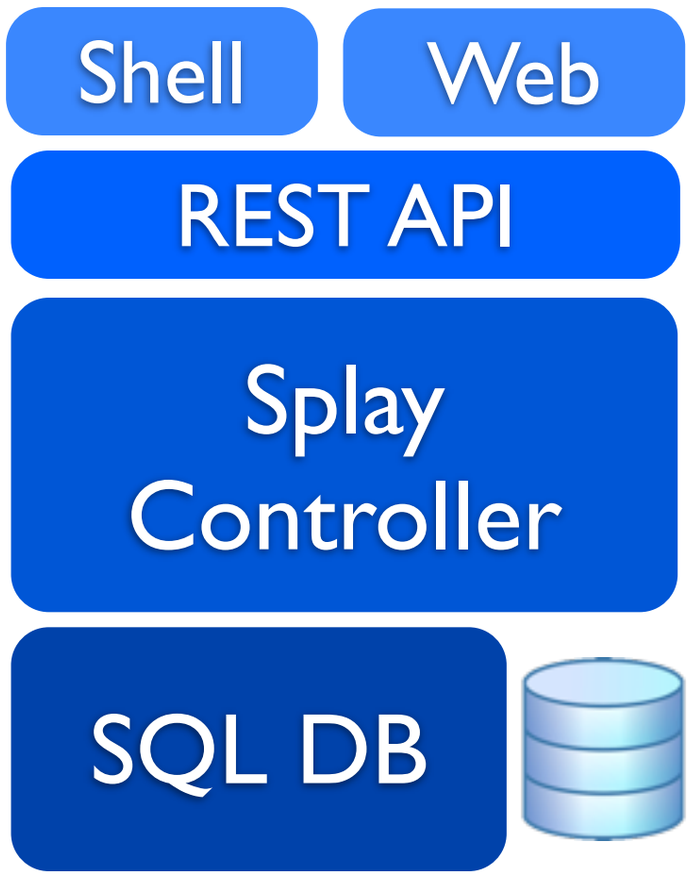
The Big Picture



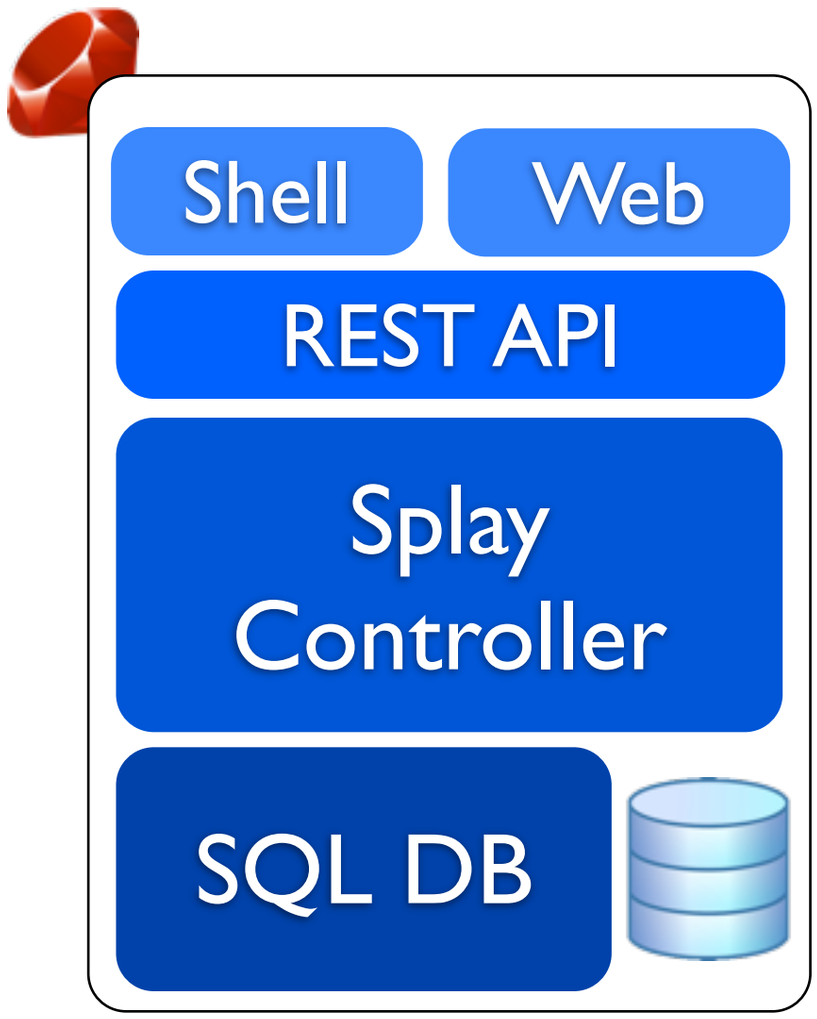
The Big Picture



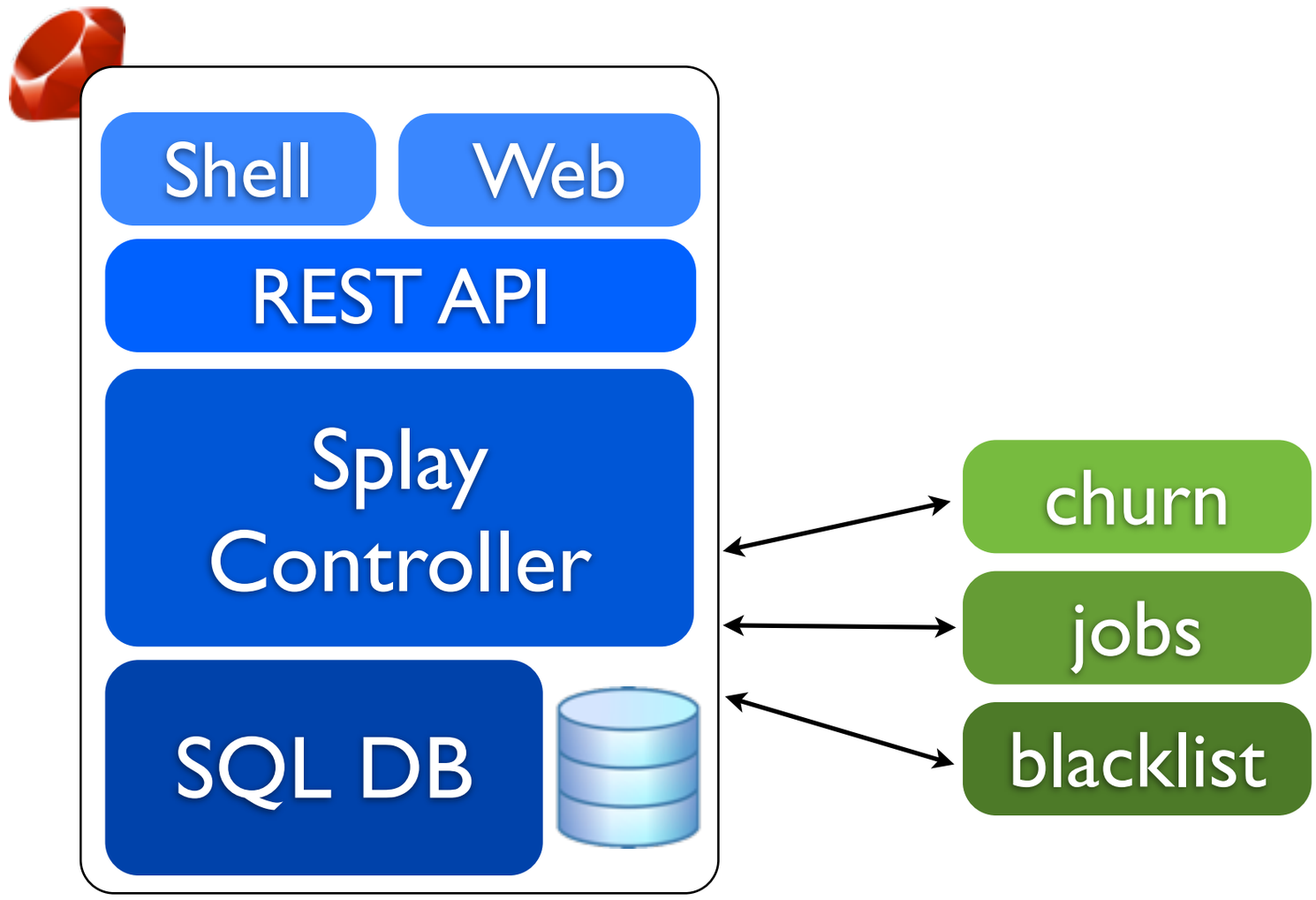
The Big Picture



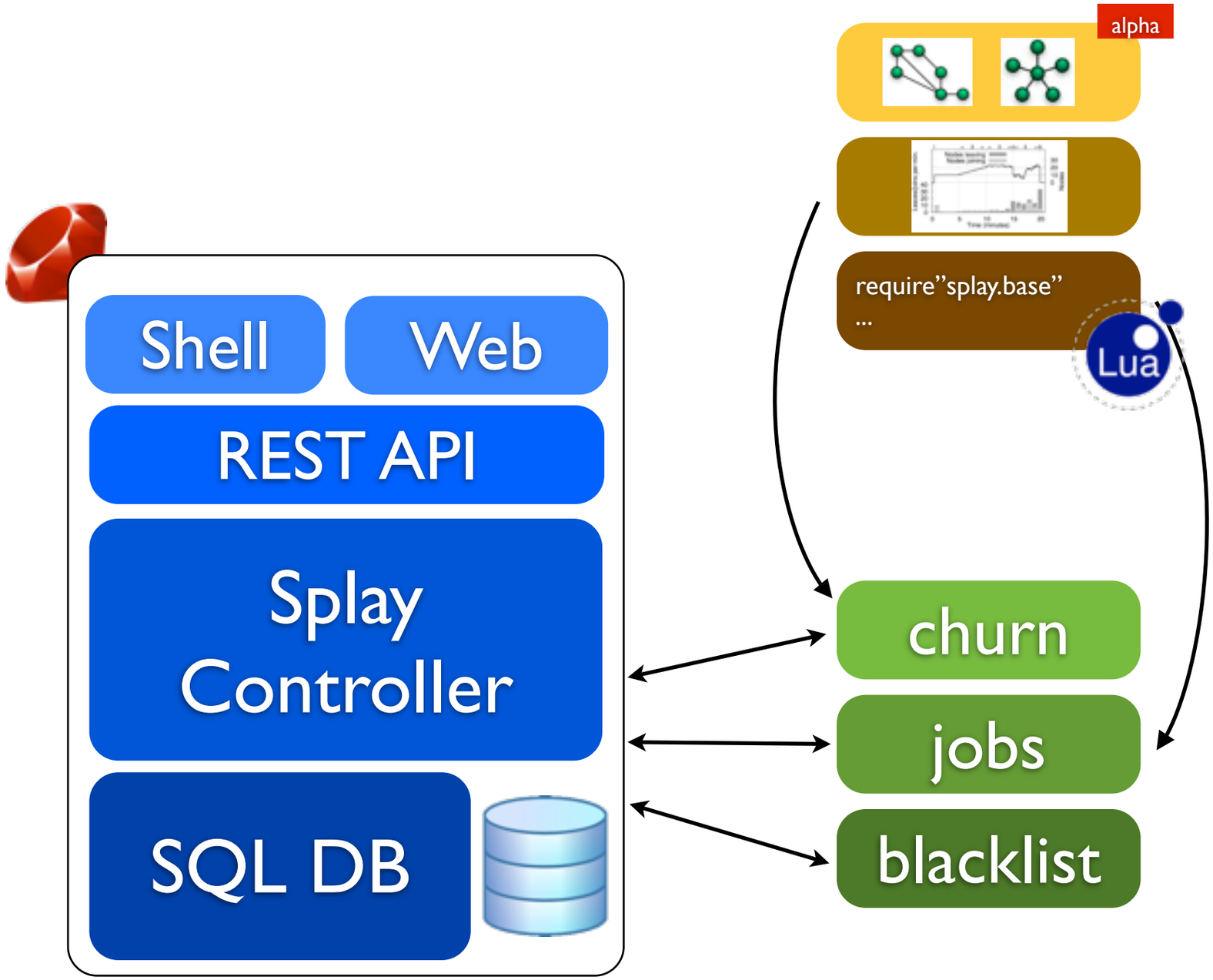
The Big Picture



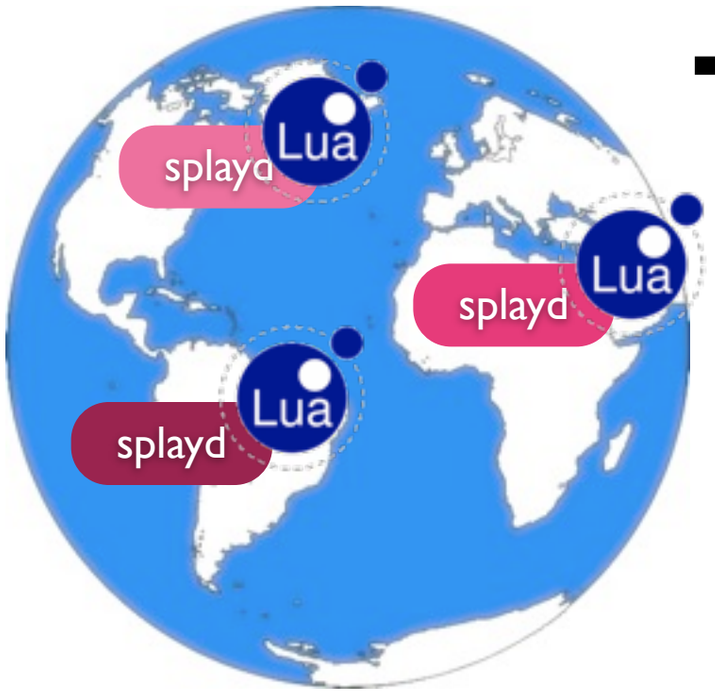
The Big Picture

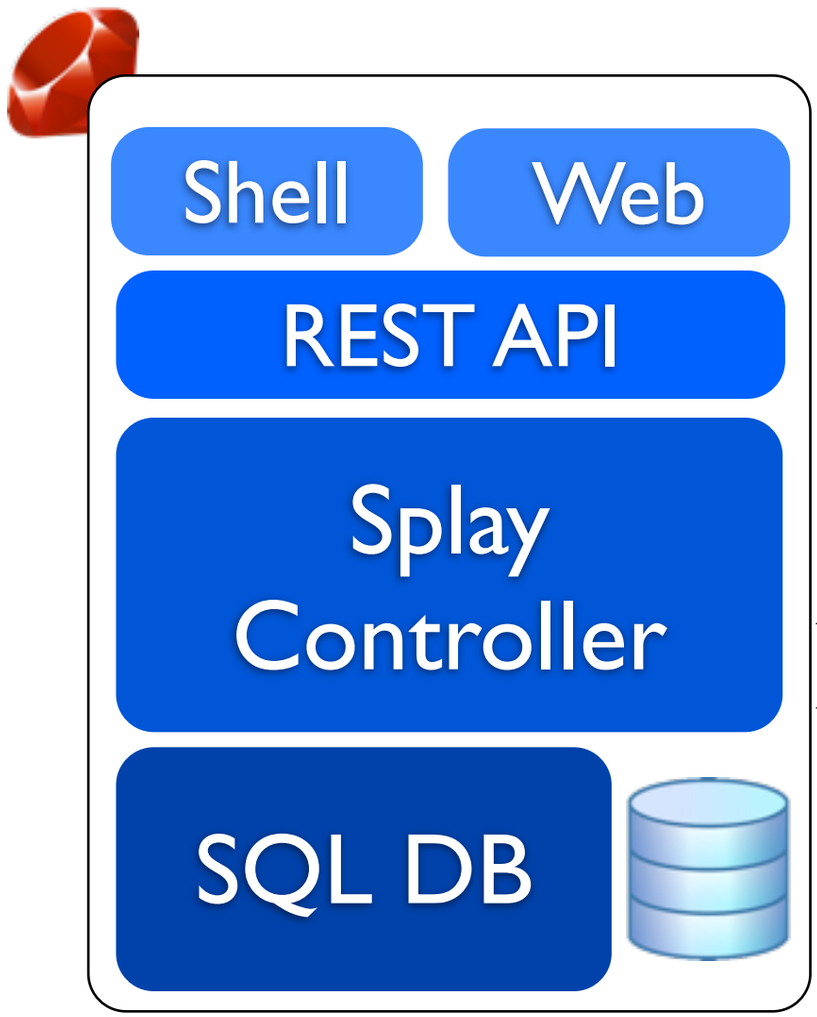


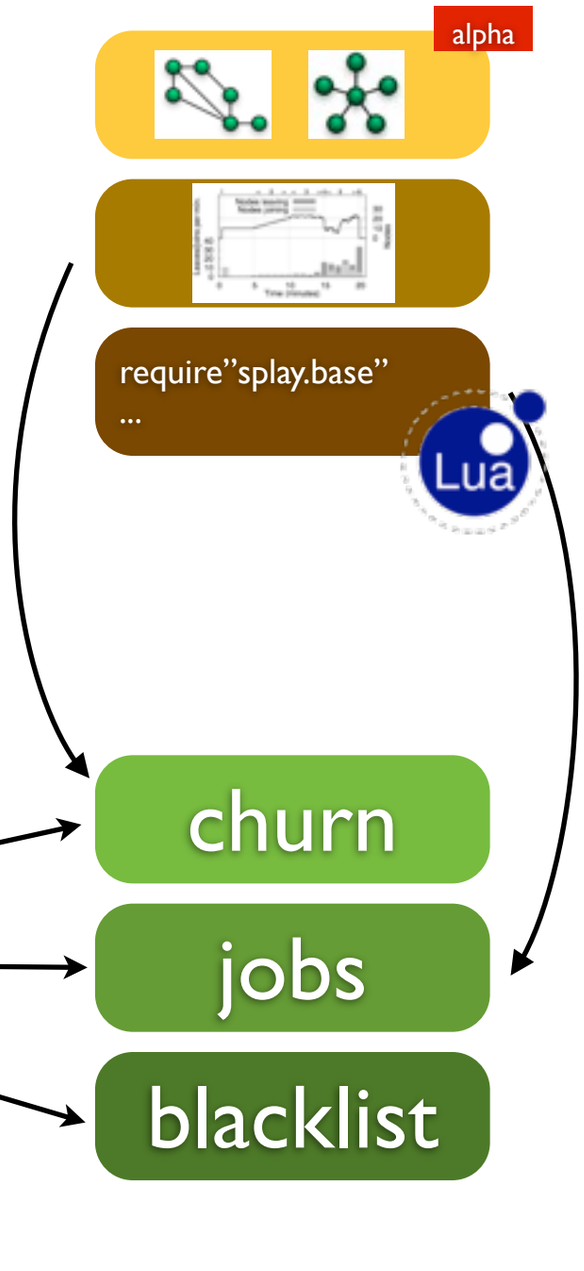
The Big Picture



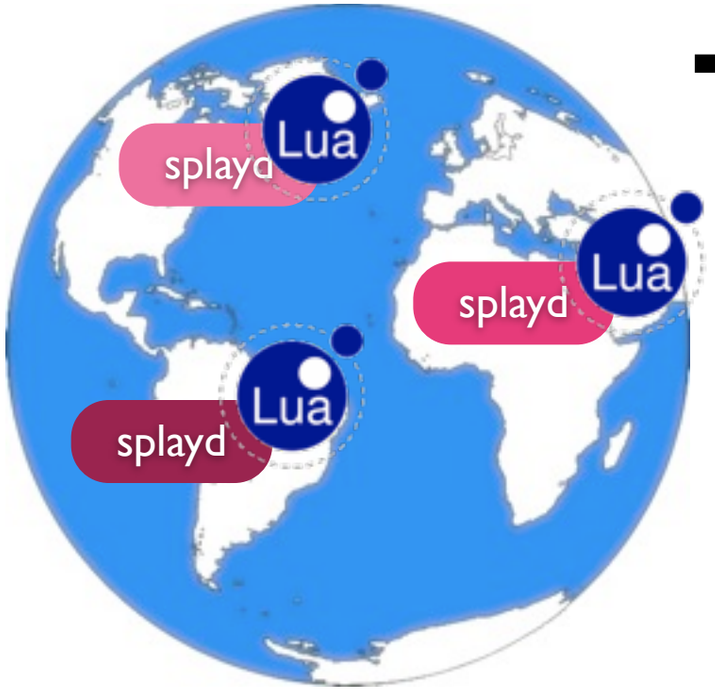
The Big Picture







The Big Picture



Shell

Web

REST API

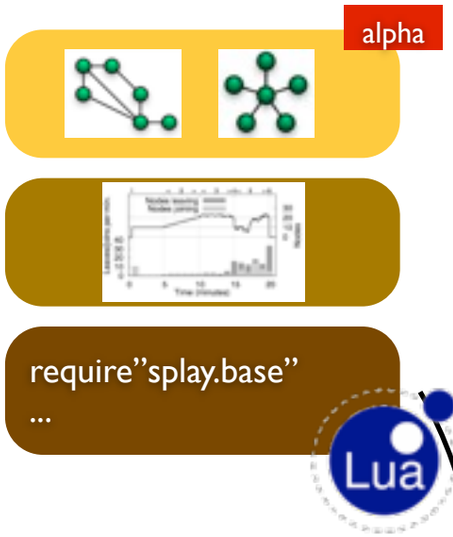
Splay Controller

SQL DB

log

log

log

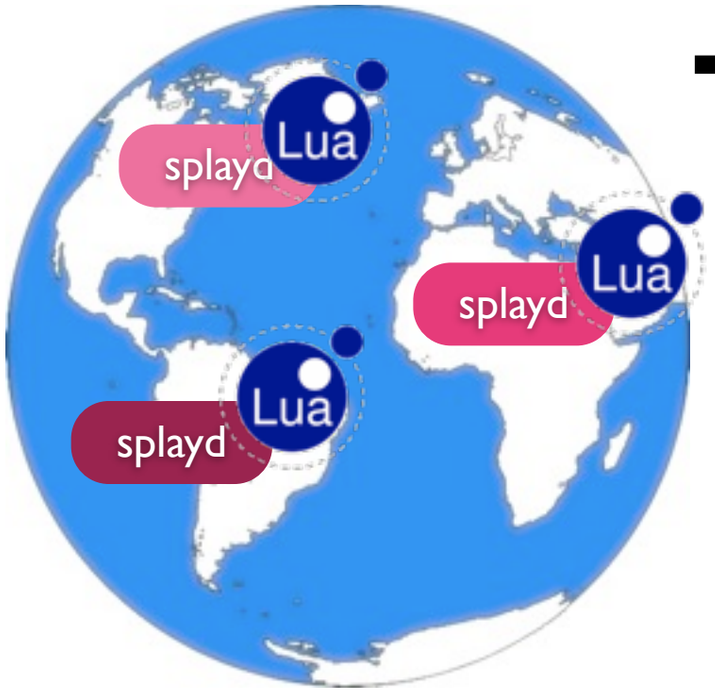


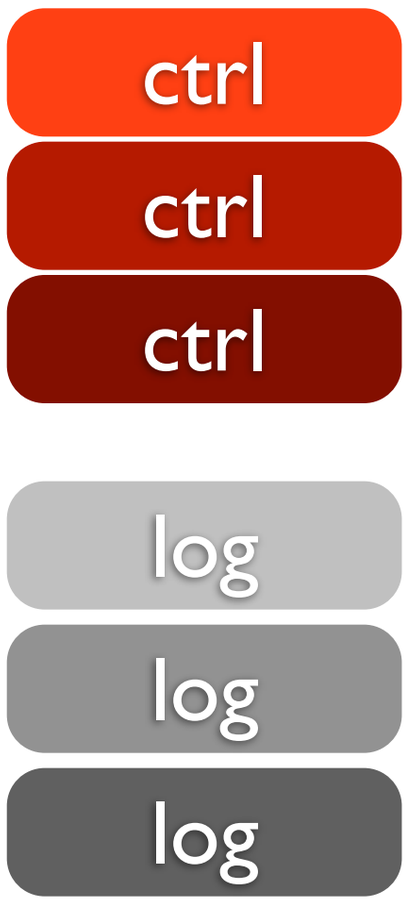
churn

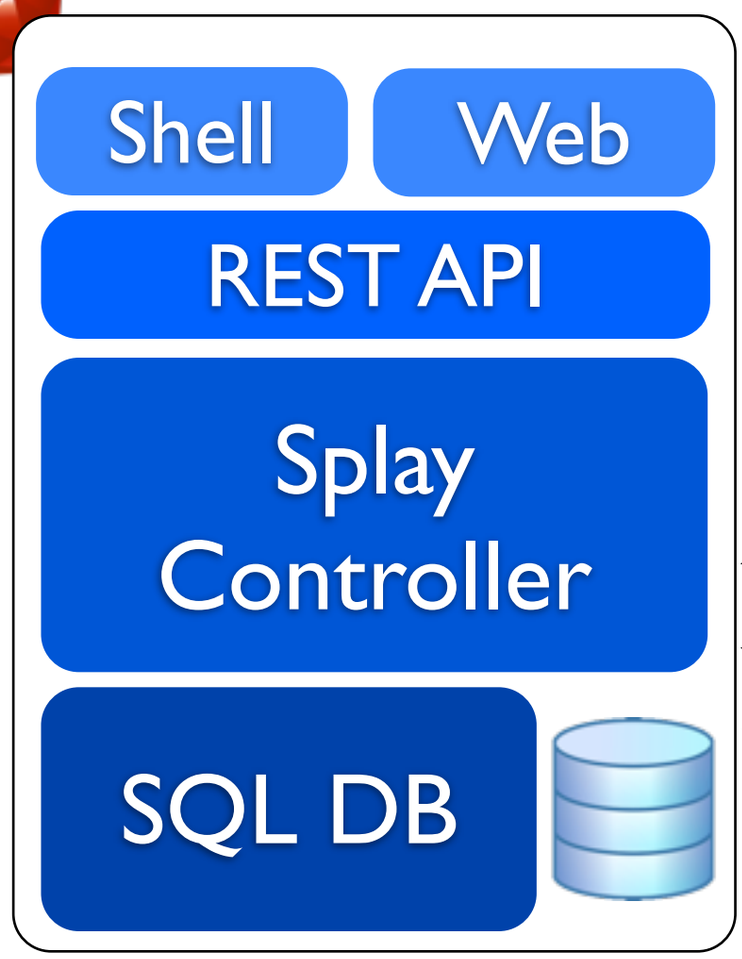
jobs

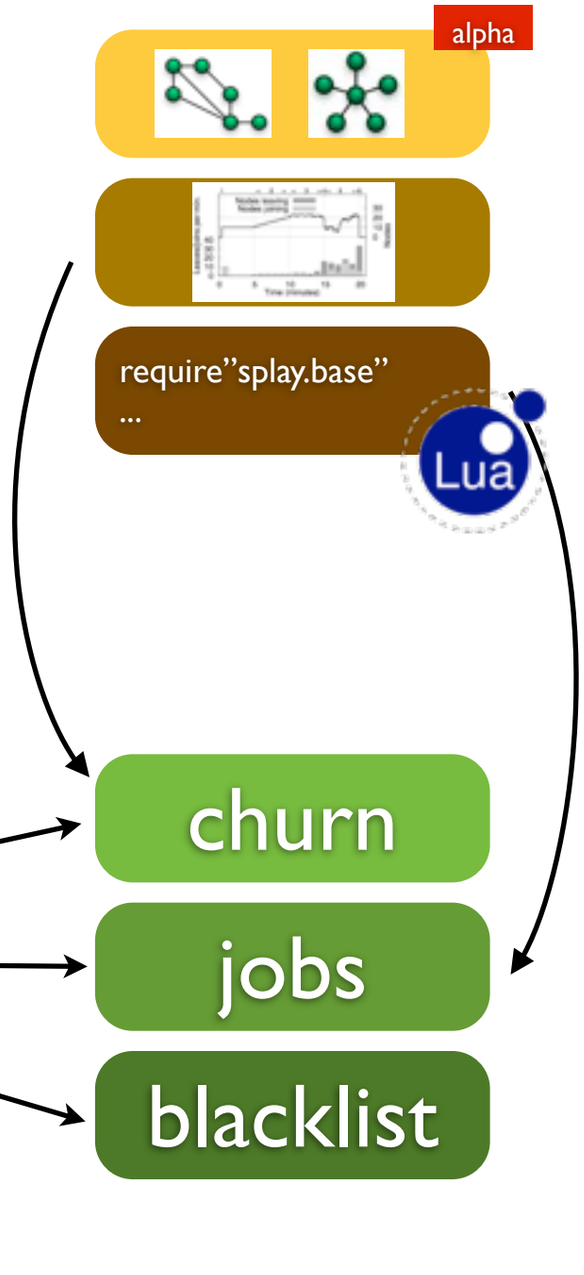
blacklist

The Big Picture

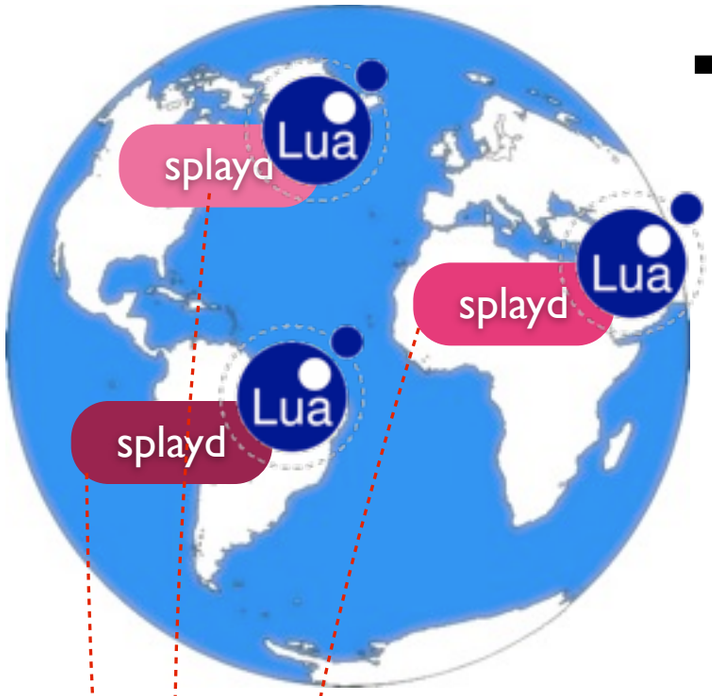


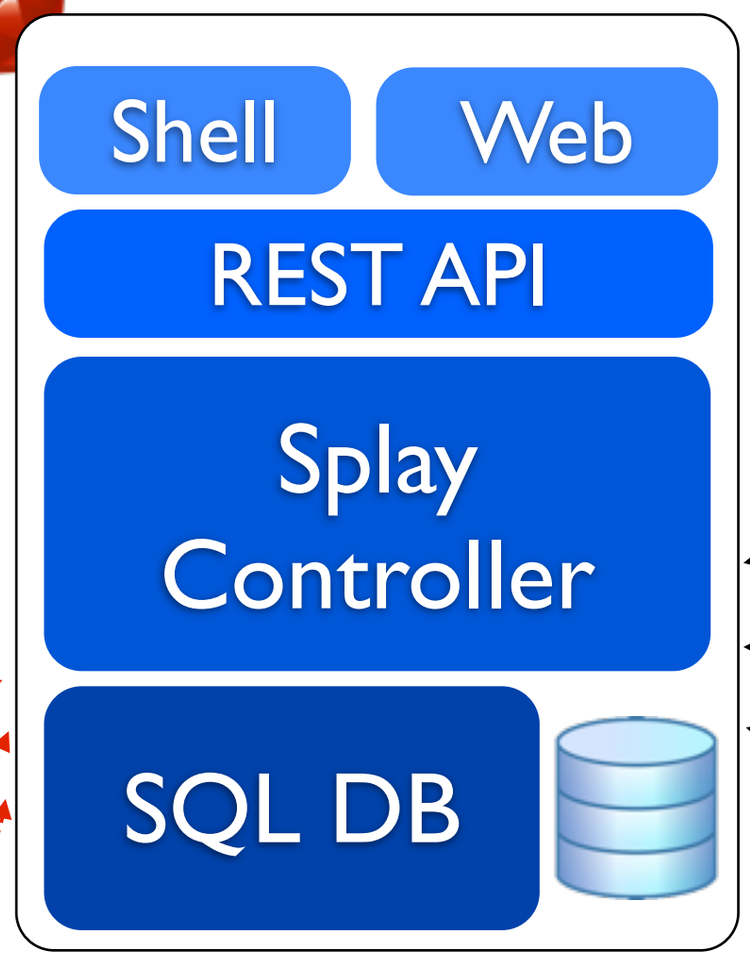


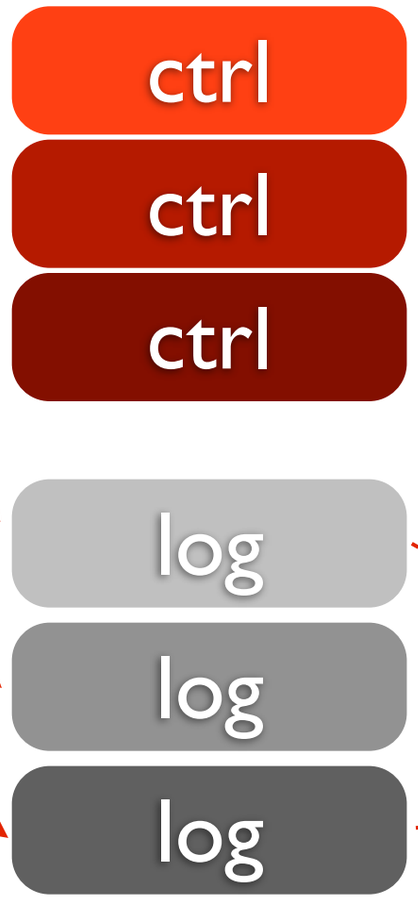


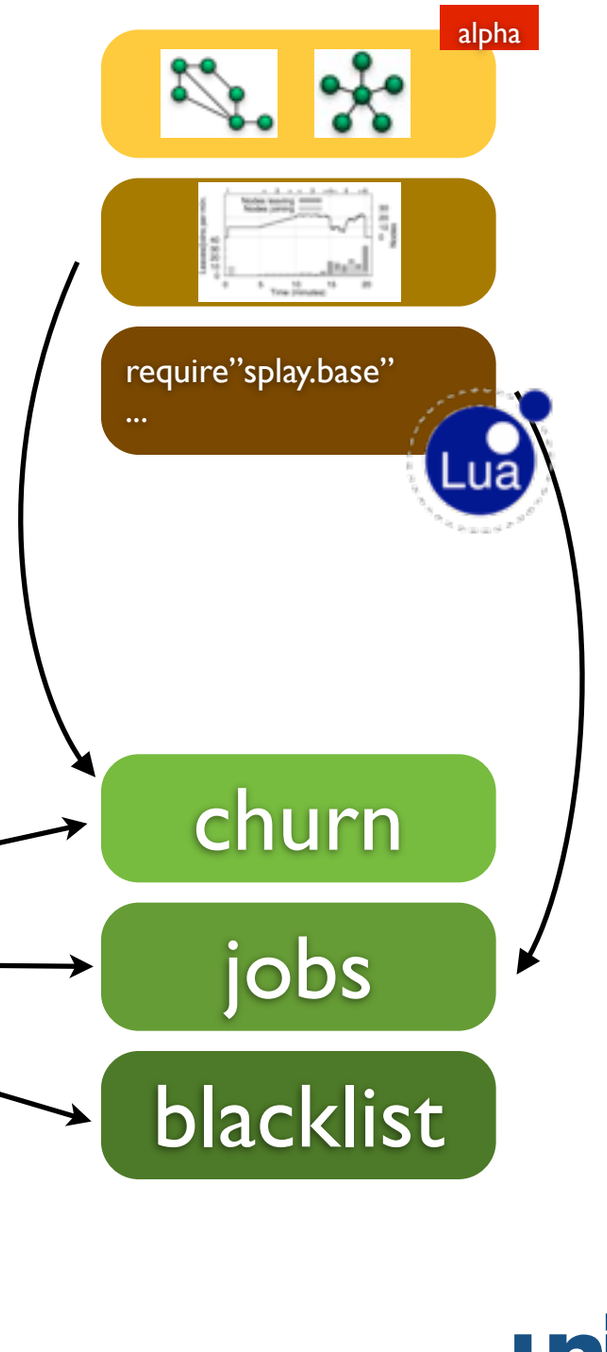


The Big Picture

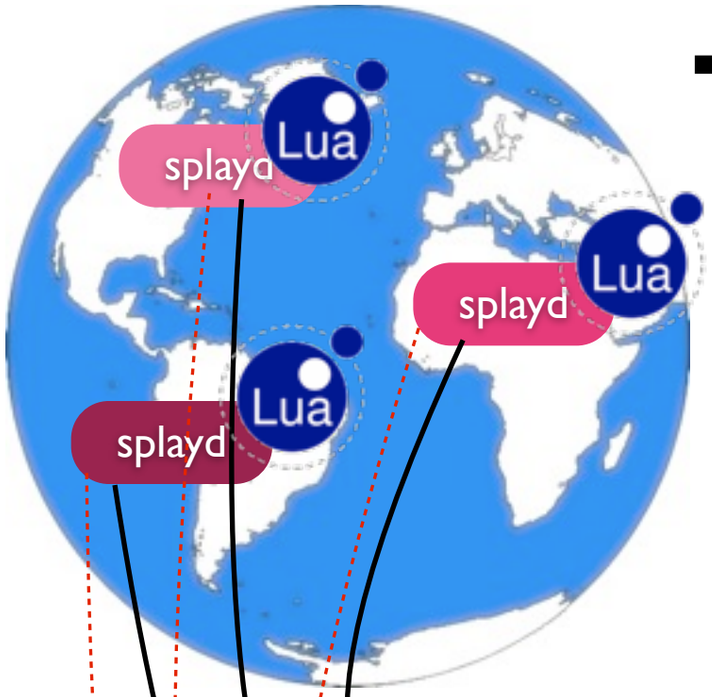








The Big Picture

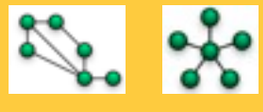


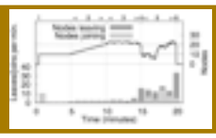
ctrl
ctrl
ctrl

log
log
log

Shell
Web
REST API
Splay Controller
SQL DB

alpha





require"splay.base"
...

Lua

churn
jobs
blacklist