On The Integration Of

SPLAY ⇄ SIMGRID

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

Université de Neuchâtel, Switzerland

SimGrid User Days 2012 - Ecully, France

Institut d'informatique
Université de Neuchâtel

SPLAY

UNIVERSITÉ DE
NEUCHÂTEL

About Me

- 2010-now : PhD student at the University of Neuchâtel, Switzerland.
  - Dependable and Distributed Computing Group.

- Topic: large-scale distributed systems, cloud computing.
- 2007-2009: Research Engineer at INRIA Grenoble, SARDES team.

Outline

- SPLAY overview
- Integrating SPLAY and SimGrid
  - Based on rough ideas
  - Suggestions are welcome, collaborations even more!

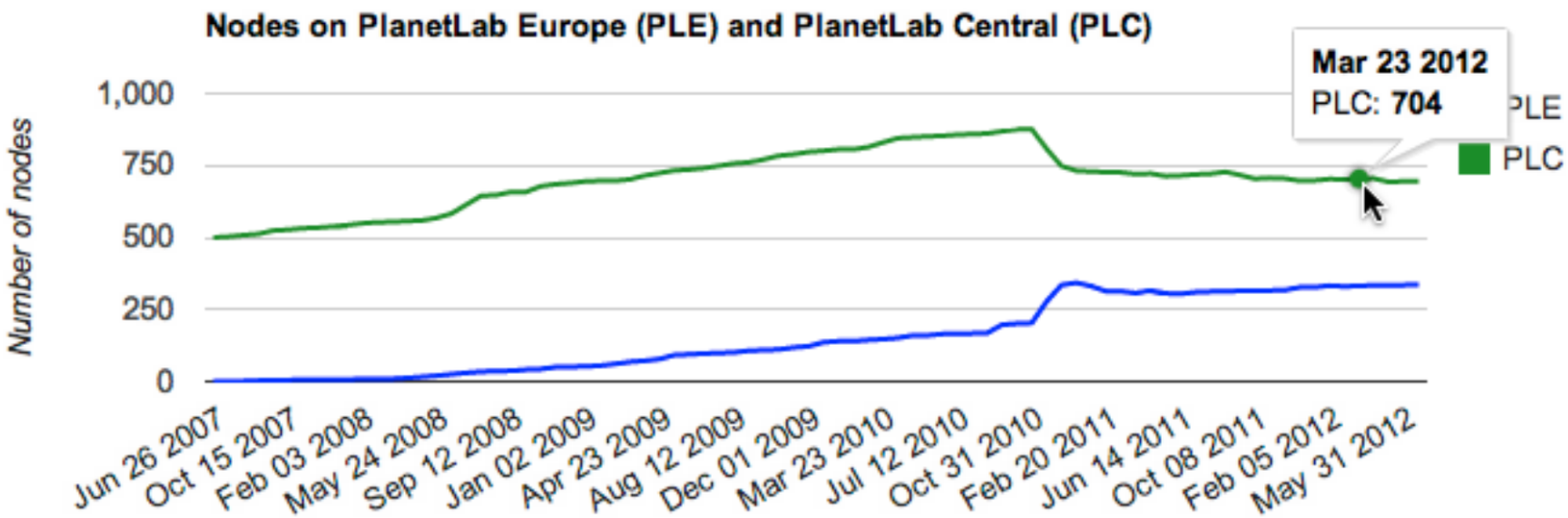
Motivations

- 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



What is PLANETLAB

- Machines contributed by universities, companies,etc.
  - ~1000 nodes at 531 sites (May 2012)
  - Shared resources, no privileged access
- University-quality Internet links
- High resource contention
- Faults, churn, packet-loss is the norm
  - Challenging conditions

- A testbed is a set of machines for testing your distributed applications/ protocols
- Several different testbeds!

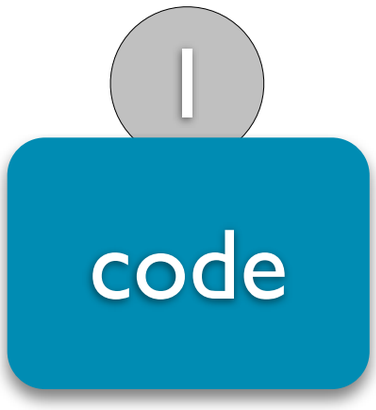
your local machine

networks of idle workstations

our new cluster@UniNE

Daily Job With Distributed Systems

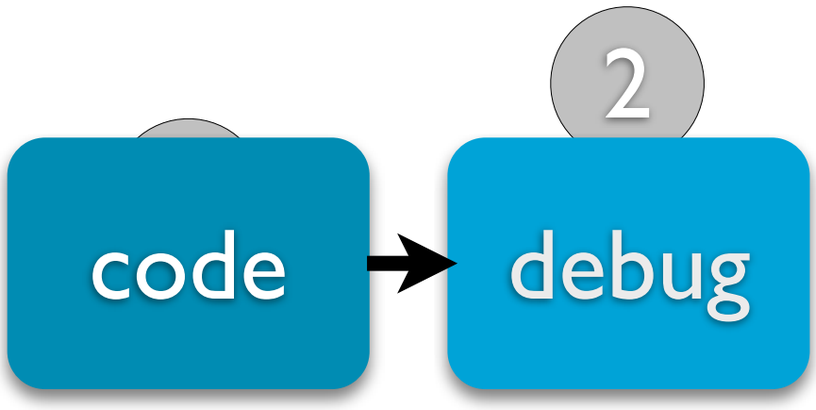
Daily Job With Distributed Systems



1

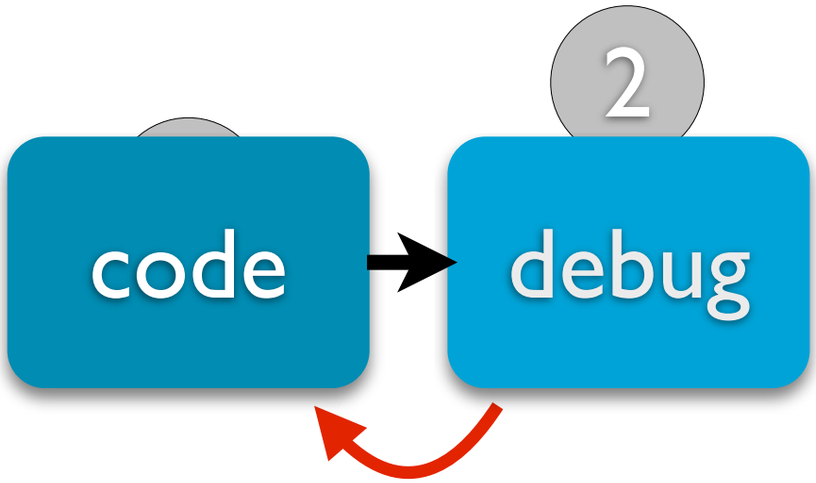
- Write (testbed specific) code
  - Local tests, in-house cluster, PlanetLab...

Daily Job With Distributed Systems



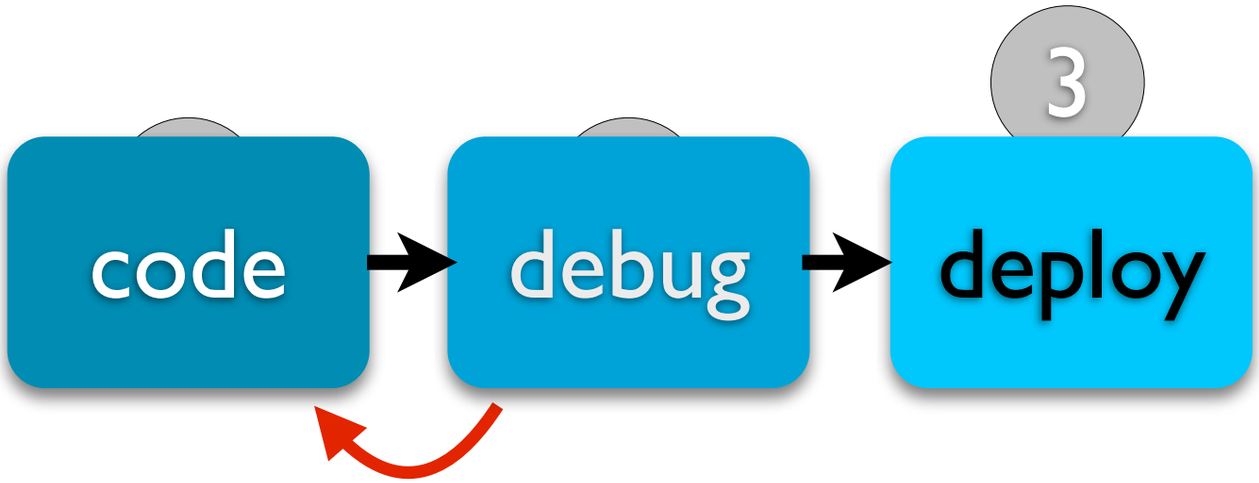
2 • Debug (in this context, a nightmare)

Daily Job With Distributed Systems



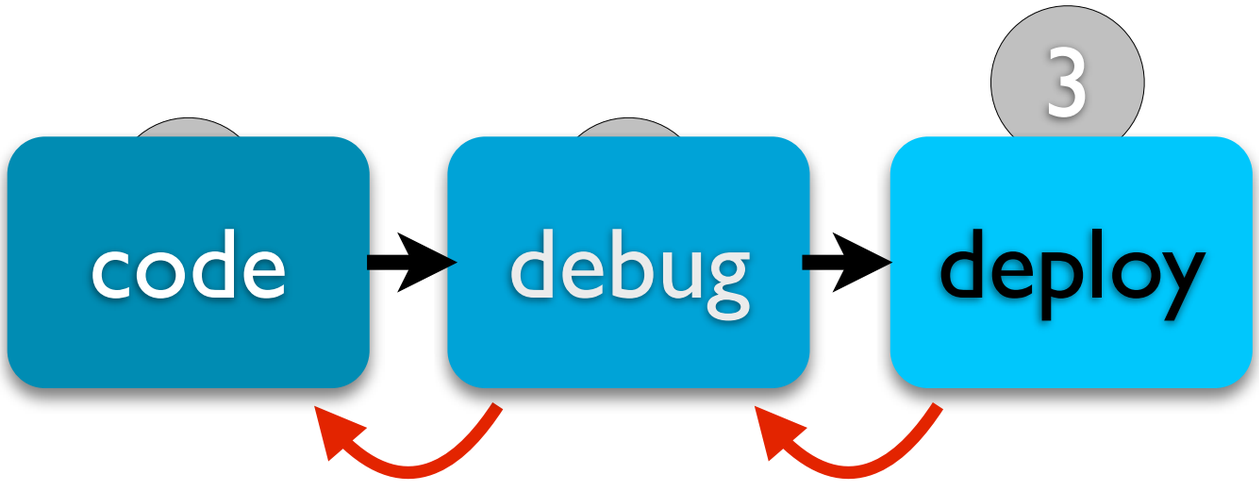
2 • Debug (in this context, a nightmare)

Daily Job With Distributed Systems



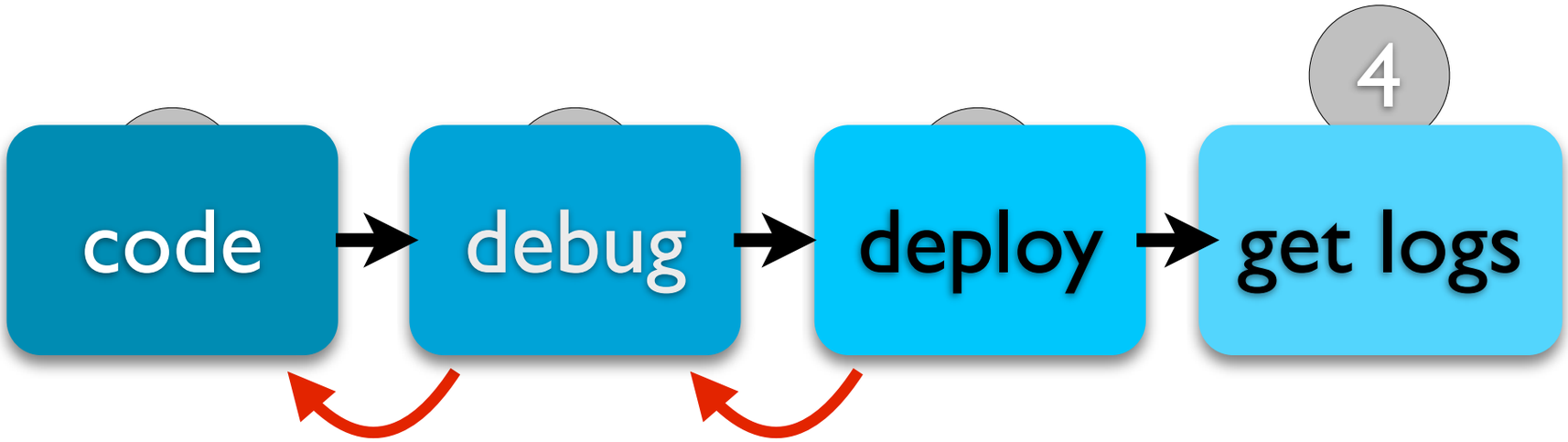
3 • Deploy, with testbed specific scripts

Daily Job With Distributed Systems



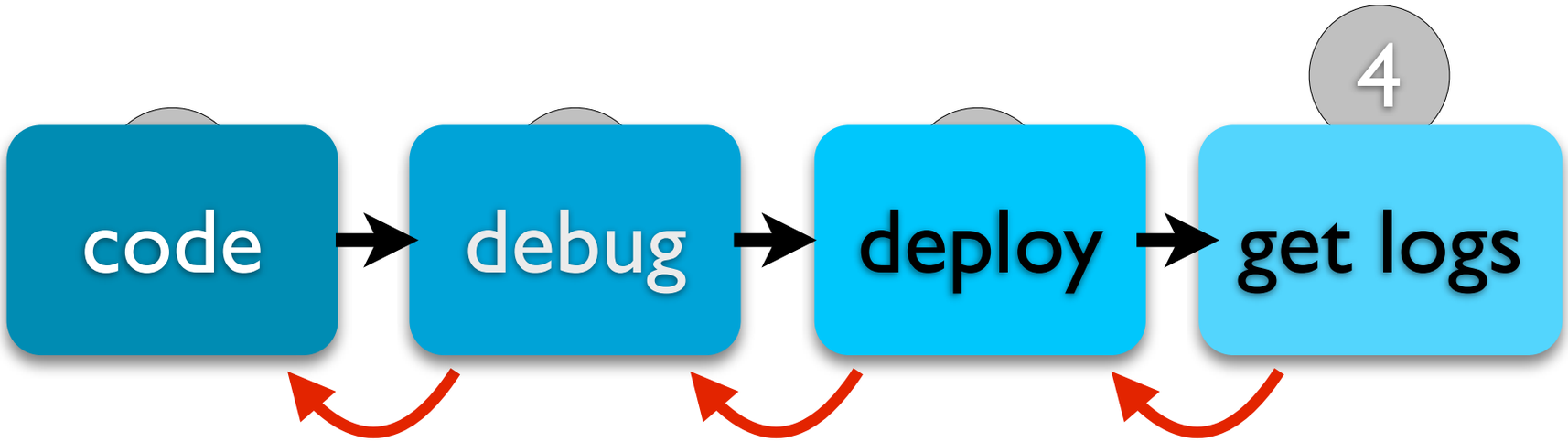
3 • Deploy, with testbed specific scripts

Daily Job With Distributed Systems



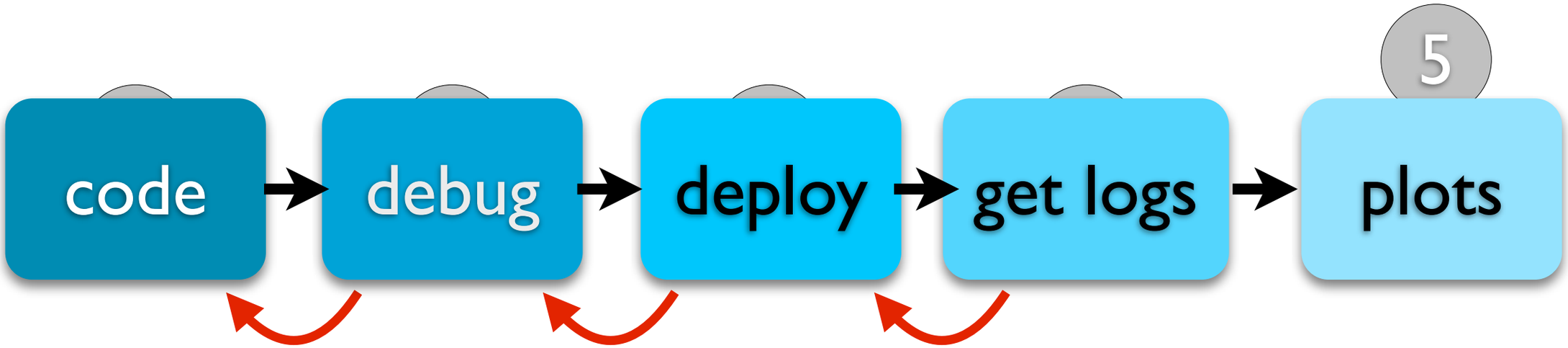
4 • Get logs, with testbed specific scripts

Daily Job With Distributed Systems



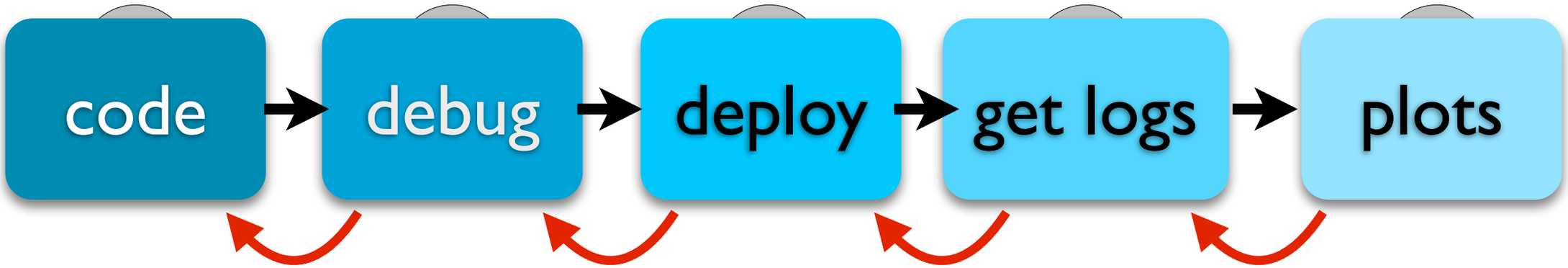
4 • Get logs, with testbed specific scripts

Daily Job With Distributed Systems



5 • Produce plots, hopefully

Daily Job With Distributed Systems

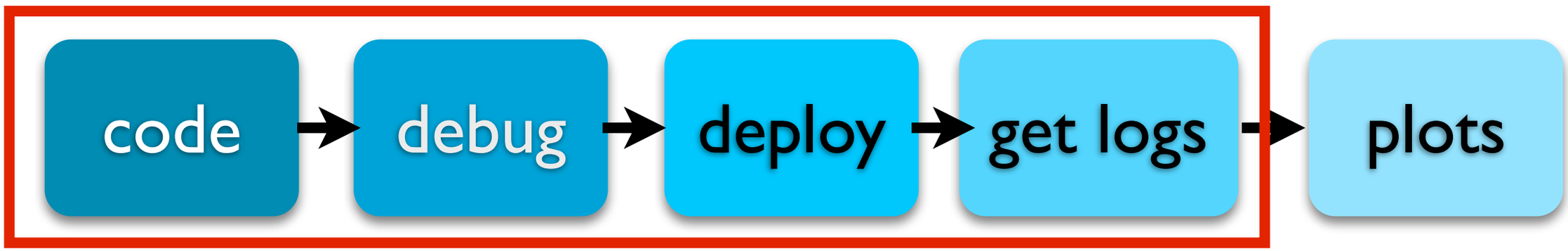


5 • Produce plots, hopefully

SPLAY at a Glance

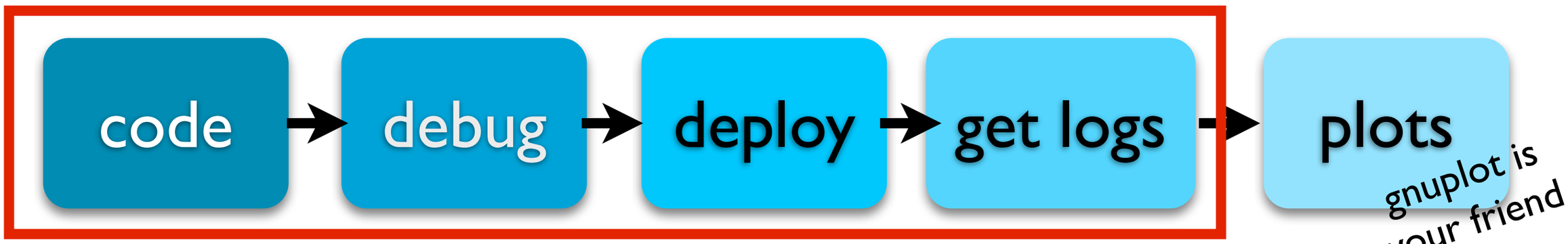
- Supports the development, evaluation, testing, and tuning of distributed applications on any testbed:
  - In-house cluster, shared testbeds, emulated environments
- Provides an easy-to-use pseudocode-like language based on Lua
  - *Write Once, Deploy&Run Everywhere*
- Open-source: http://www.splay-project.org

SPLAY at a 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 based on Lua
  - *Write Once, Deploy&Run Everywhere*
- Open-source: http://www.splay-project.org

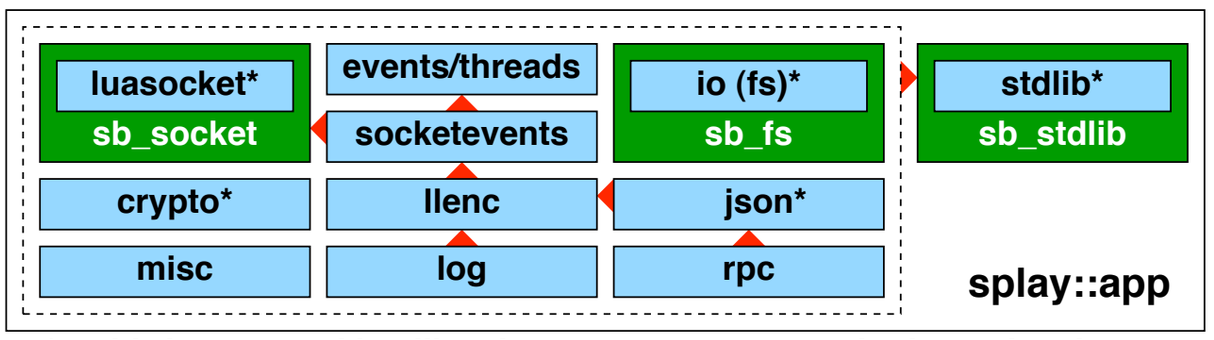
SPLAY at a Glance



- 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 based on Lua
  - *Write Once, Deploy&Run Everywhere*
- Open-source: http://www.splay-project.org

SPLAY Language

- Based on Lua
  - High-level scripting language
  - Made for interaction with C
  - Bytecode-based, garbage collection
- Must be close to pseudo code (focus on the algorithm)
  - & favor "natural ways" of expressing algorithms (e.g. RPCs)
- SPLAY applications can be run locally without the deployment infrastructure (for debugging & testing)
- Comprehensive set of libraries (can be extended)



Why  ?

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

Concise

Concise

Pseudo code as published on original paper ←

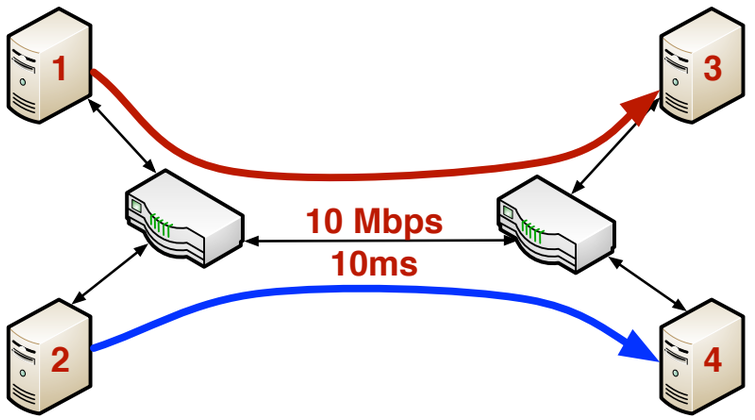
```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    n' = closest_preceding_node(id);
    return n'.find_successor(id);

// 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;
```

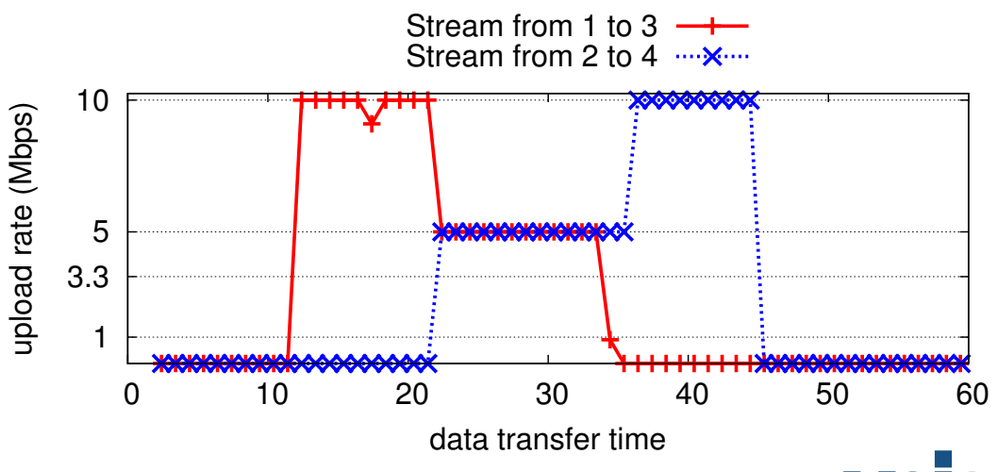
Executable code using SPLAY libraries →

```
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
```

SPLAY Network Emulation

- Similar *in the spirit* to existing network emulators (ModelNet, Emulab)
- Novel features:
  - multi-user
  - multi-topology
  - user-space





SPLAY Network Emulation

- XML/TCS formats to define topologies
- Support for delay, lossy channels
- User-land bandwidth shaping
- Emulation of congestion at inner nodes in a completely decentralized fashion
  - Dynamic adjustments of token-bucket bandwidth shapers

SPLAY ⇄ SIMGRID

- Integrating the two systems to get the best of both world

- Two options:

1. Use Splay apps within SimGrid
2. Use SimGrid apps within Splay

Splay within SimGrid

- Evaluate Splay applications with SimGrid
  - Different execution models, network emulation
- Splay adapter for SimGrid Lua Bindings
- Customized sandbox
- Customized Splay RPC libraries

SimGrid within Splay

- Evaluate SimGrid applications within Splay testbed
- Easier if applications are written in Lua
- Alternatively, exploit Splay binary shipping
  - On-the-fly per-job relaxation of Splay sandbox to allow binary code
  - What about sandboxing C applications?

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