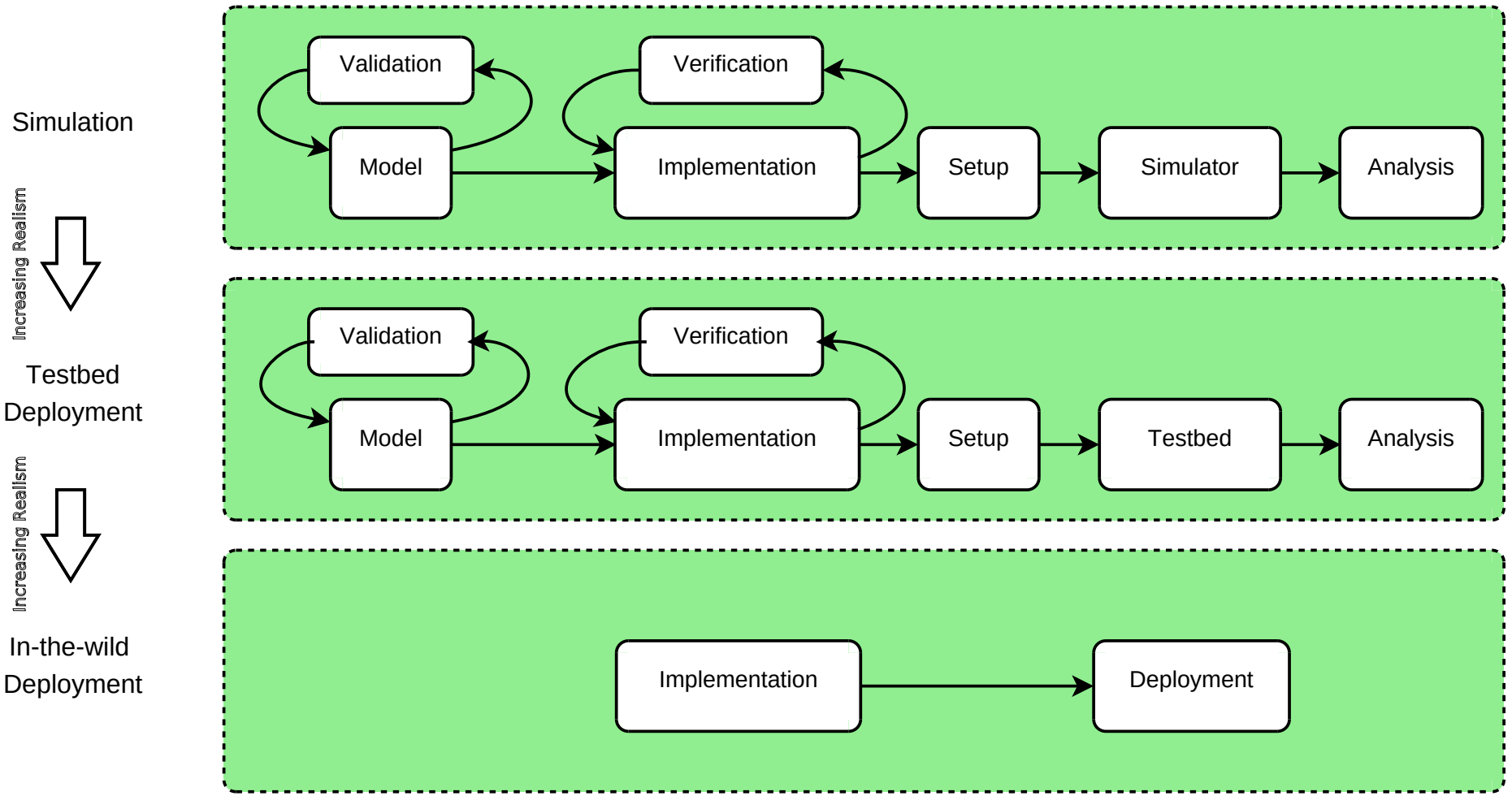
Direct Code Execution with ns-3

Mathieu Lacage
mathieu.lacage@sophia.inria.fr

INRIA

Workshop on ns-3, March 15th, 2010

A typical experimentation workflow



A lot of wasted effort

Duplication:

- Modeling
- Implementation
- Validation
- Verification

A lot of wasted effort

Duplication:

- Modeling
- Implementation
- Validation
- Verification

Direct Code Execution is a way to avoid all this wasted effort:

- Run the real application in the simulator
- Run the real application in the testbed
- Run the real application in-the-wild

How can this be done ?

- Use a virtual machine (network containers, etc.) and emulation
- Patch the application source code:
  - global variables
  - use simulation APIs
- Automate source code patching (ala NSC):
  - globalizer
  - re-implement API used by application

Lots of downsides

- VMs:
  - Hard to debug: distributed debugging
  - Hard to deploy, control, monitor
  - Costly from CPU and memory perspectives
- Source code patching:
  - Almost impossible to maintain
- Automate source code patching
  - Lack of robustness of the source parser

ELF PIC Globalization

At compile time:

- CFLAGS=-fPIC
- CXXFLAGS=-fPIC
- LDFLAGS=-pie

At runtime:

- ns-3 calls dlmopen: a variant of dlopen

Pros:

- No distributed debugging
- Lightweight: memory+cpu

Cons:

- dlmopen implementation broken (beyond repair) in glibc

Our implementation

Re-implement glibc:

- A new ELF loader: dlmopen works !
- A new set of POSIX functions: sockets, files, etc.

Current status:

- ping
- traceroute
- iperf

Future work:

- bittorrent client and tracker
- quagga
- ccnd+ccn apps (ccnx)

What it looks like

```
ProcessManagerHelper processManager;
processManager.Install (nodes);

ProcessHelper process;
process.SetStackSize (1<<16);
process.SetBinary ("build/debug/iperf");
process.AddArgument ("-s");

apps = process.Install (nodes.Get (1));
apps.Start (Seconds (1.0));
```

Questions ?