A Network in a Laptop: Rapid Prototyping for Software-Defined Networks

Bob Lantz, Brandon Heller, Nick McKeown

Stanford University

HotNets 2010, 10/20/10

Wouldn't it be amazing…

if systems papers were *runnable*.

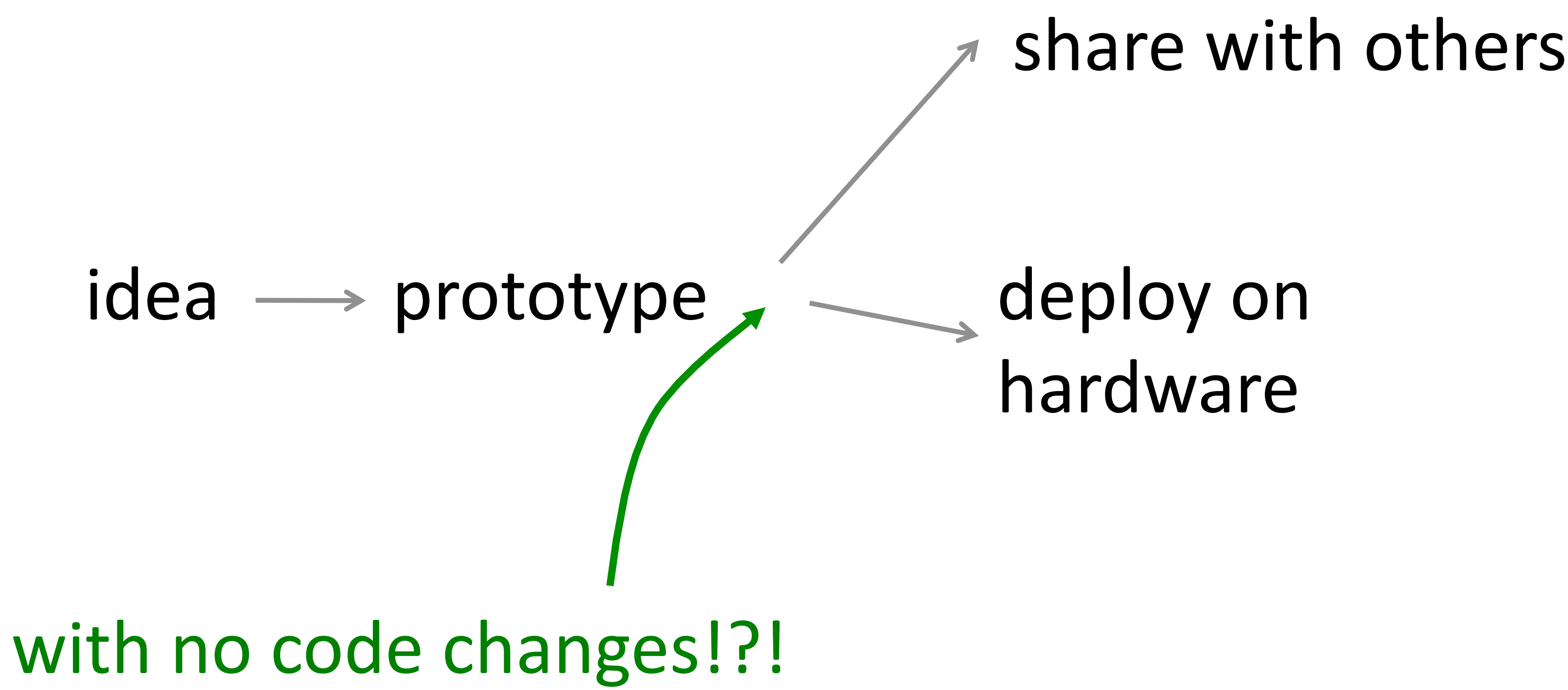
Wouldn't it be amazing…

If systems papers made
- replicating their results,
- modifying the described system,
- and sharing it with others…

… as easy as downloading a file.

Wouldn't it be amazing…

if network systems papers were *more* than runnable.



Mininet: a platform for rapid *network* prototyping.

scales to **usefully large nets**
runs **unmodified applications**
provides **path to hardware**
facilitates **sharing**

openflow.org/mininet

140+ users

45+ on mailing list

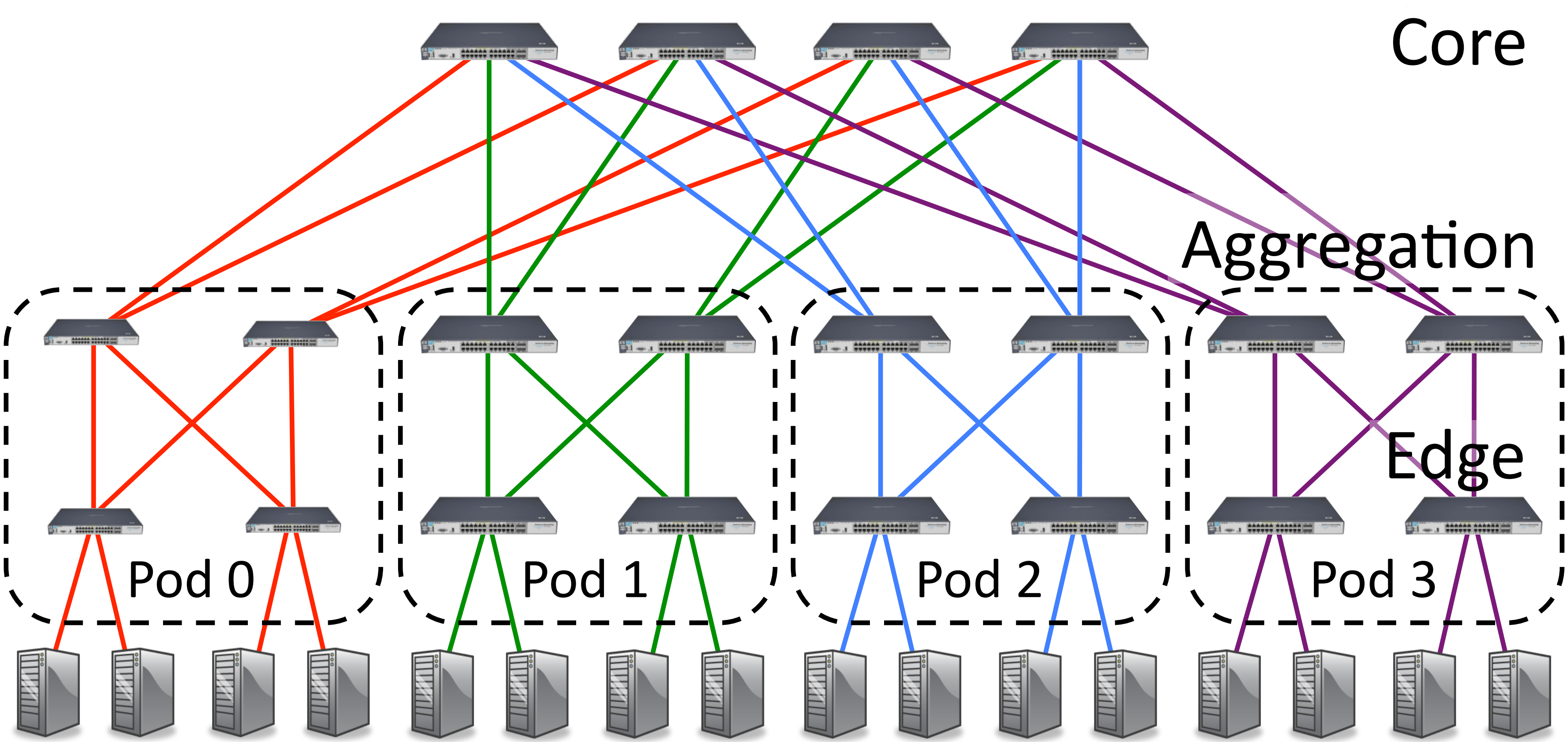
20+ institutions

open source (BSD license)

[don’t download now! save the WiFi!]

Demo

Demo Topology: Fat Tree



described in Scalable Commodity Data Center, SIGCOMM 2008, Al Fares et al.

(1) share-able

(2) runs on hardware

Date

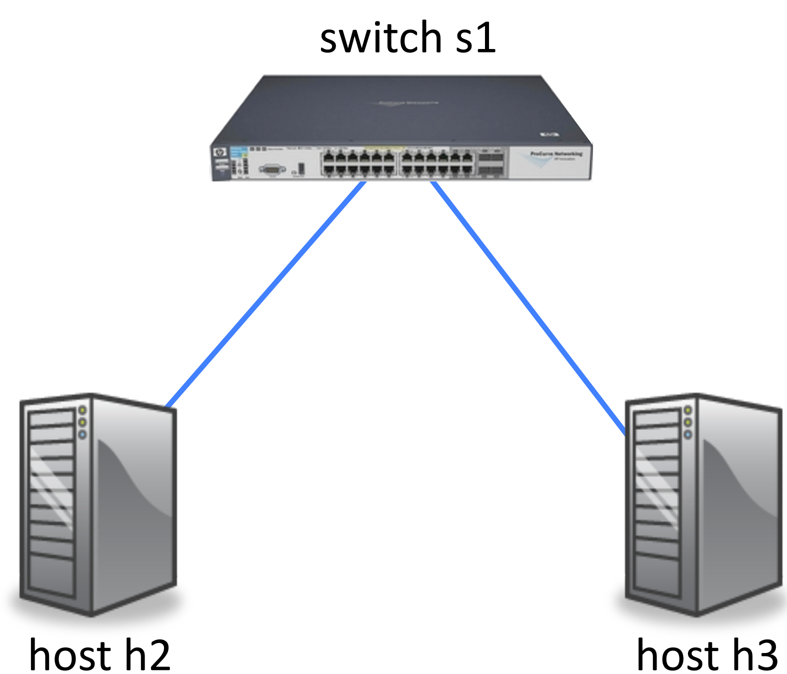
Nov 2009: deadline in 3 months

[based on a true story]

Resources:
a laptop

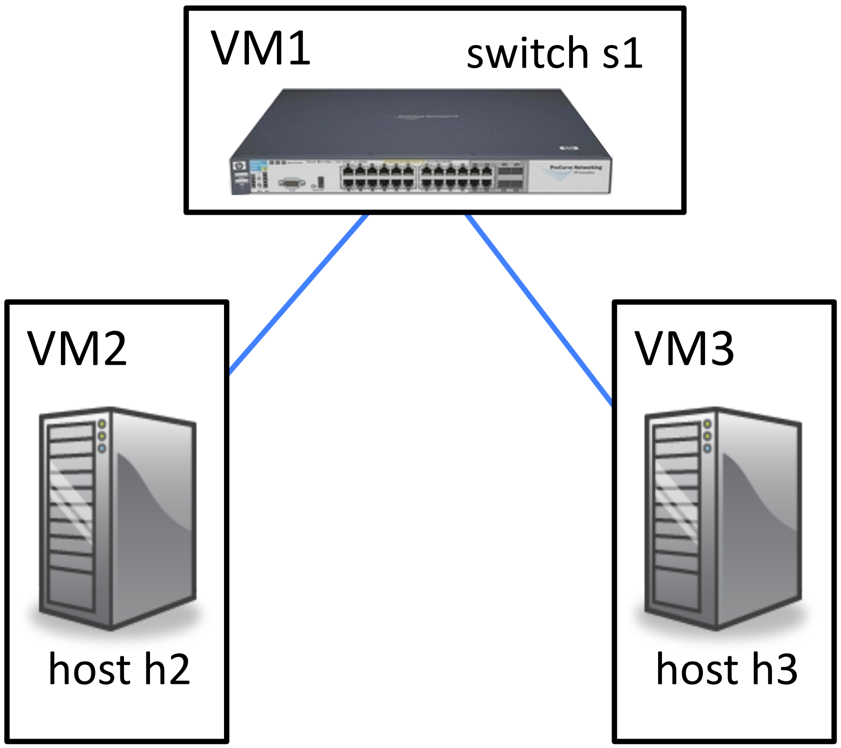
Goal:
build/eval/demo a realistic new networked system

Why not a real system?



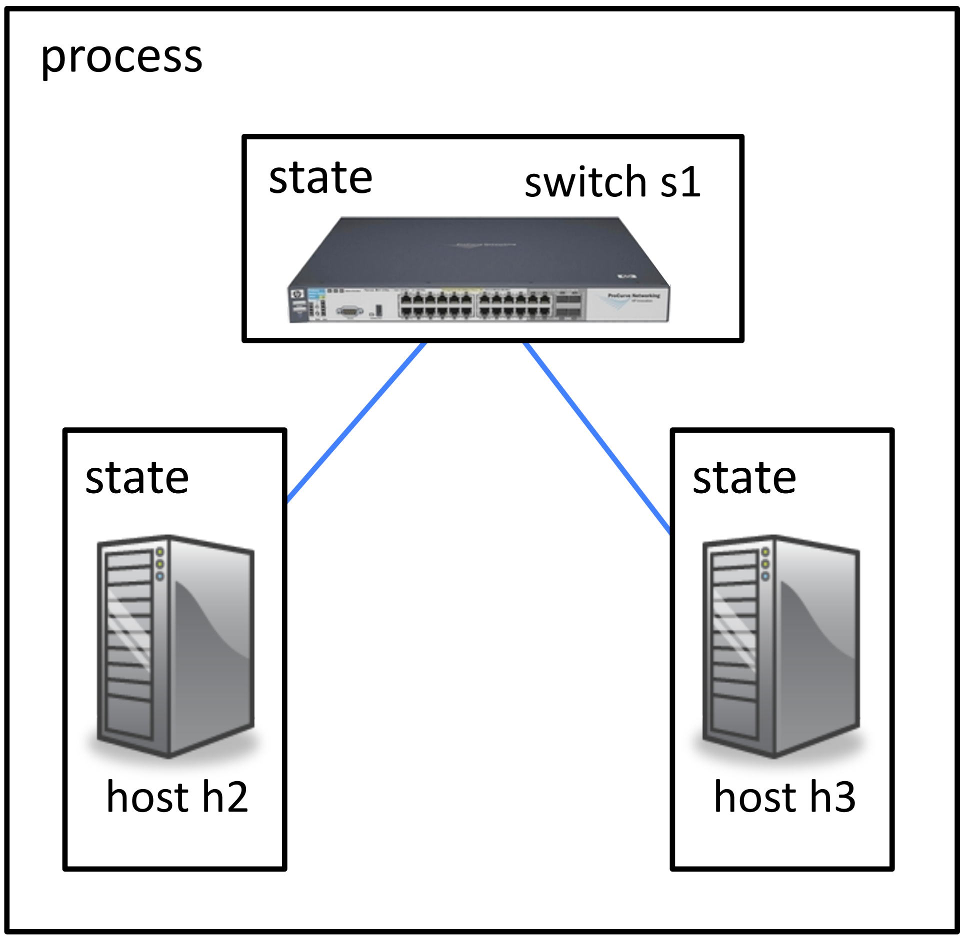
+ as real as it gets
- a pain to reconfigure

Why not networked virtual machines?



+ easier topology changes
- scalability

Why not a simulator?



+ good visibility
- no path to hardware

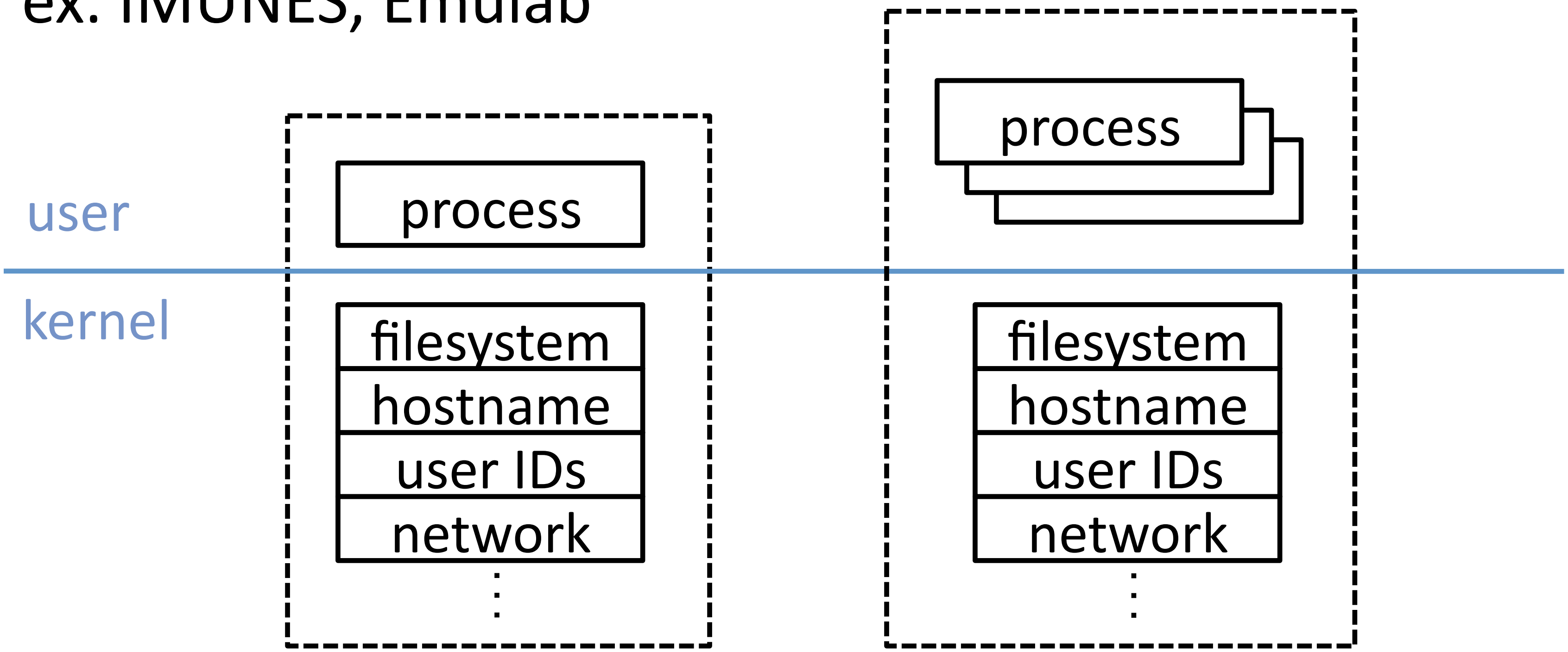
Problem 1:
Want scale with unmodified applications.

→ Use lightweight, OS-level virtualization.

OS-level Virtualization

Same system, different view. Almost zero overhead.

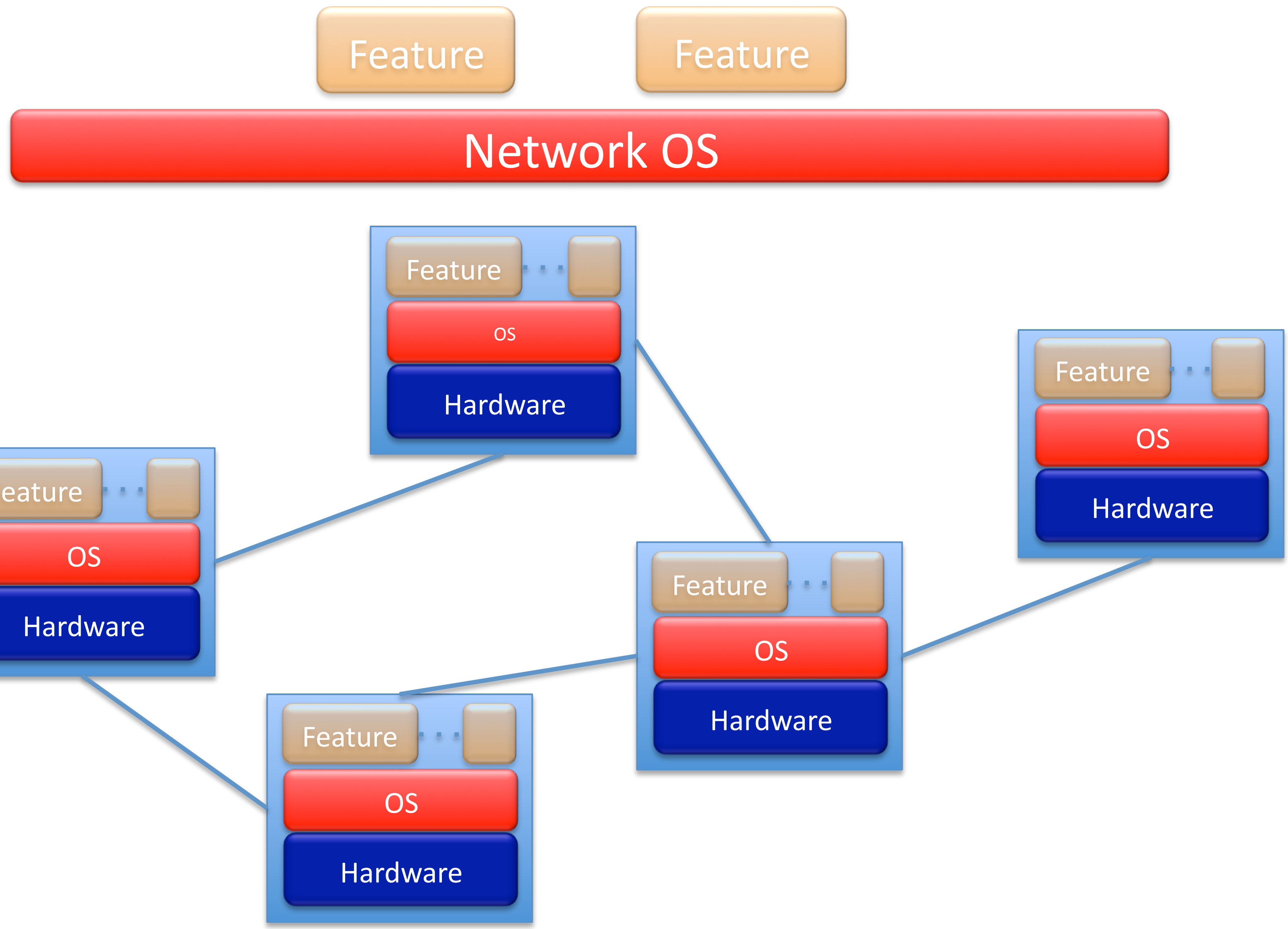
ex. IMUNES, Emulab



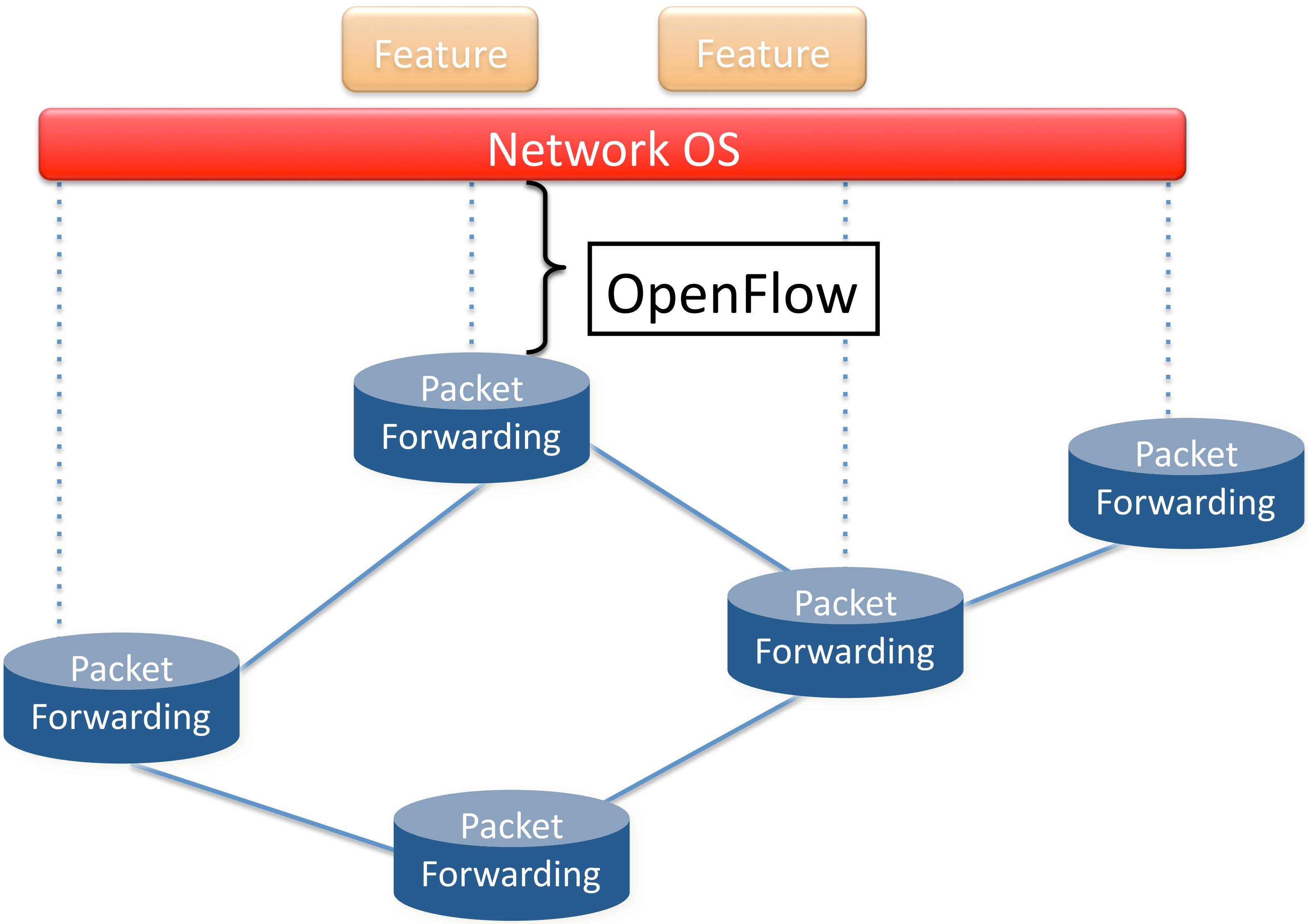
one OS kernel

Problem 2:
Want a smooth path to hardware deployment.

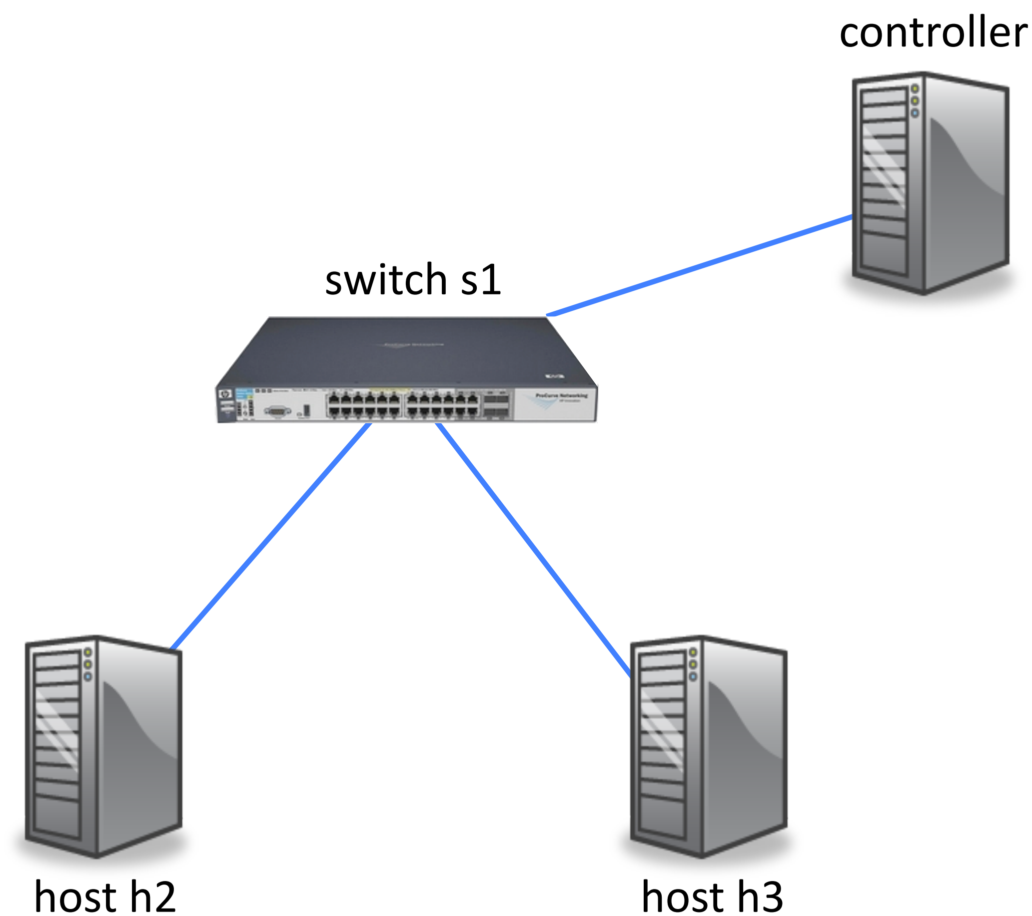
→ Use Software-Defined Networking.


Feature
Feature
Network OS
Feature
OS
Hardware
Feature
OS
Hardware
Feature
OS
Hardware
Feature
OS
Hardware
Feature
OS
Hardware

Software-Defined Network

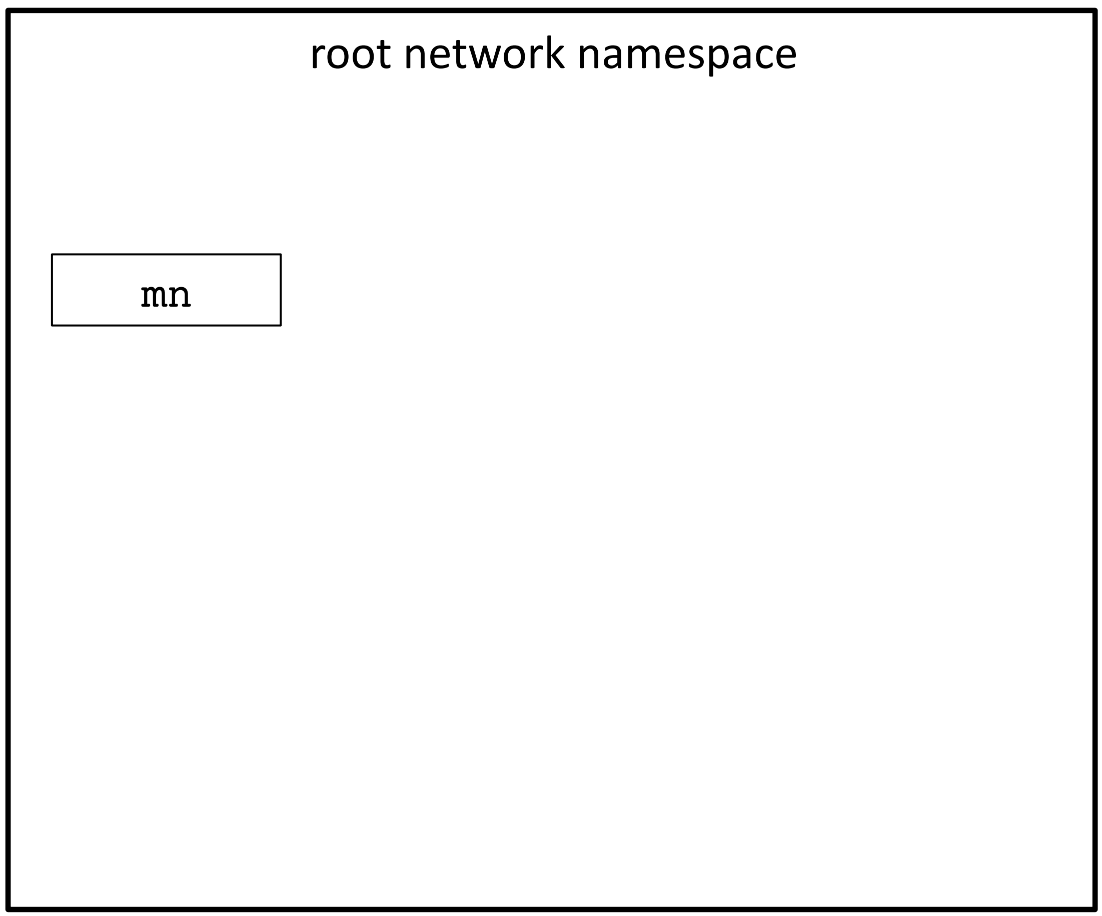


Mininet Walkthrough


controller
switch s1
host h2
host h3

```
$> mn --topo minimal \
      --switch ovsk \
      --controller ref
```

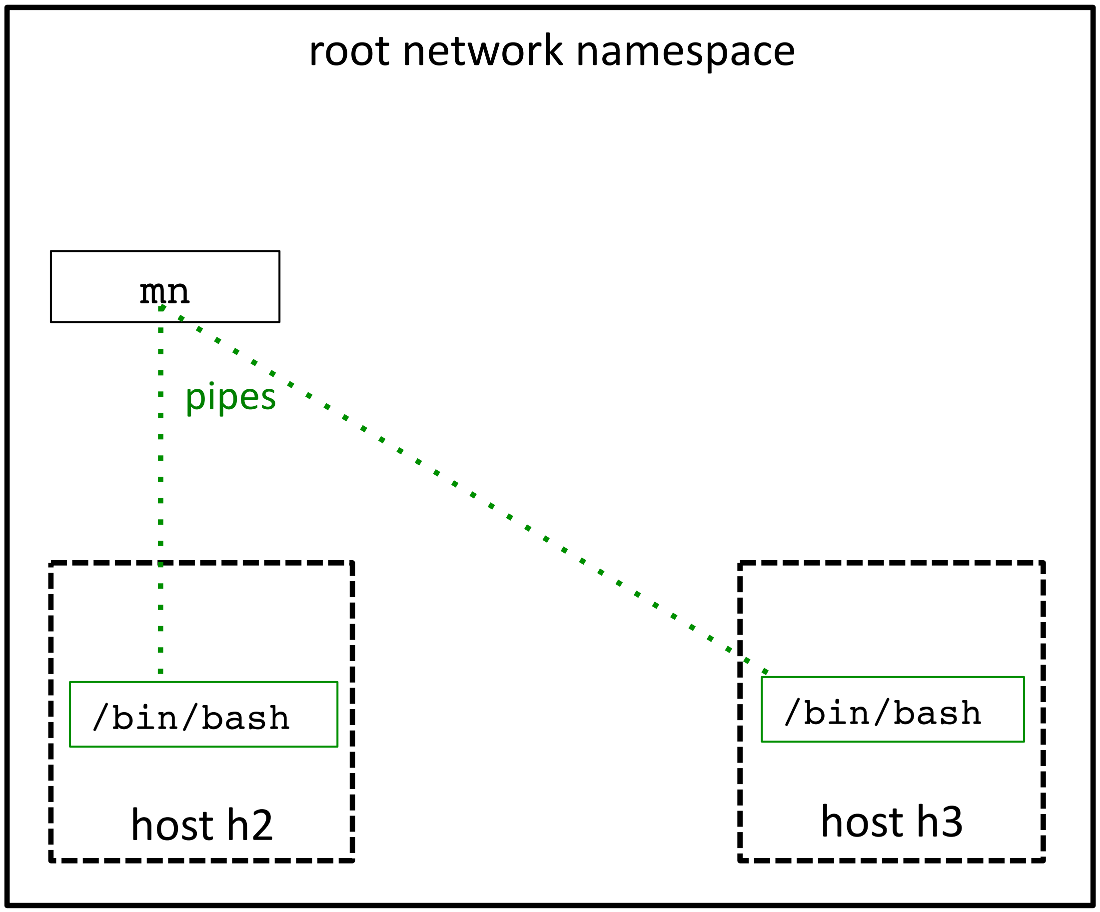
run Mininet launcher



```
$> mn --topo minimal \
      --switch ovsk \
      --controller ref
```

Hosts

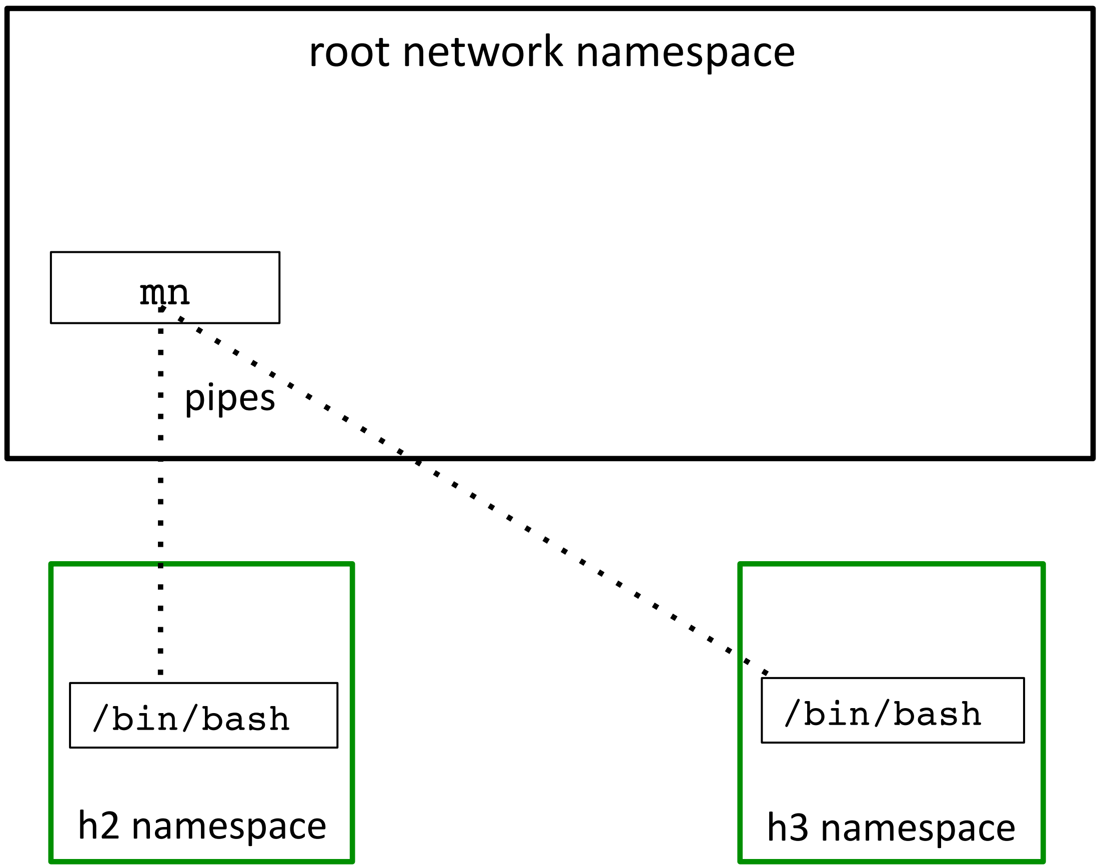
create bash processes



Hosts

```
$> mn --topo minimal \
      --switch ovsk \
      --controller ref
```

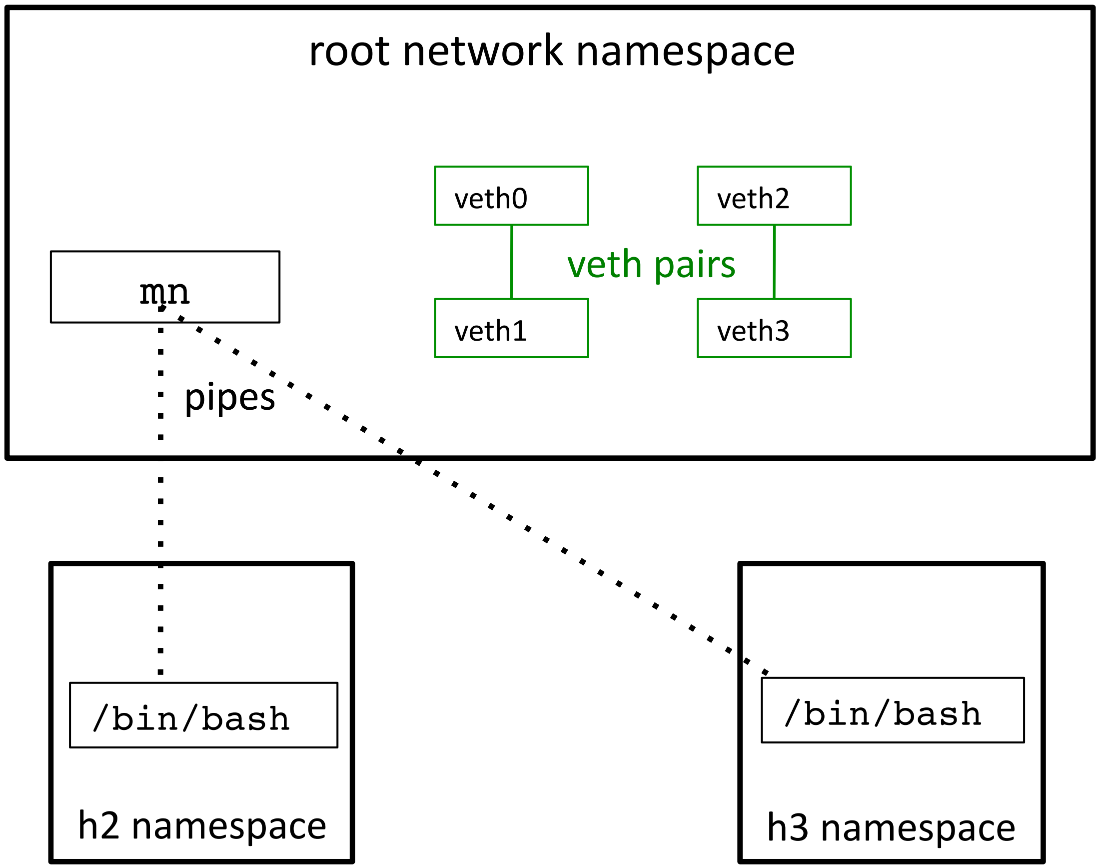
`unshare(CLONE_NEWNET)`



```
$> mn --topo minimal \
      --switch ovsk \
      --controller ref
```

Links

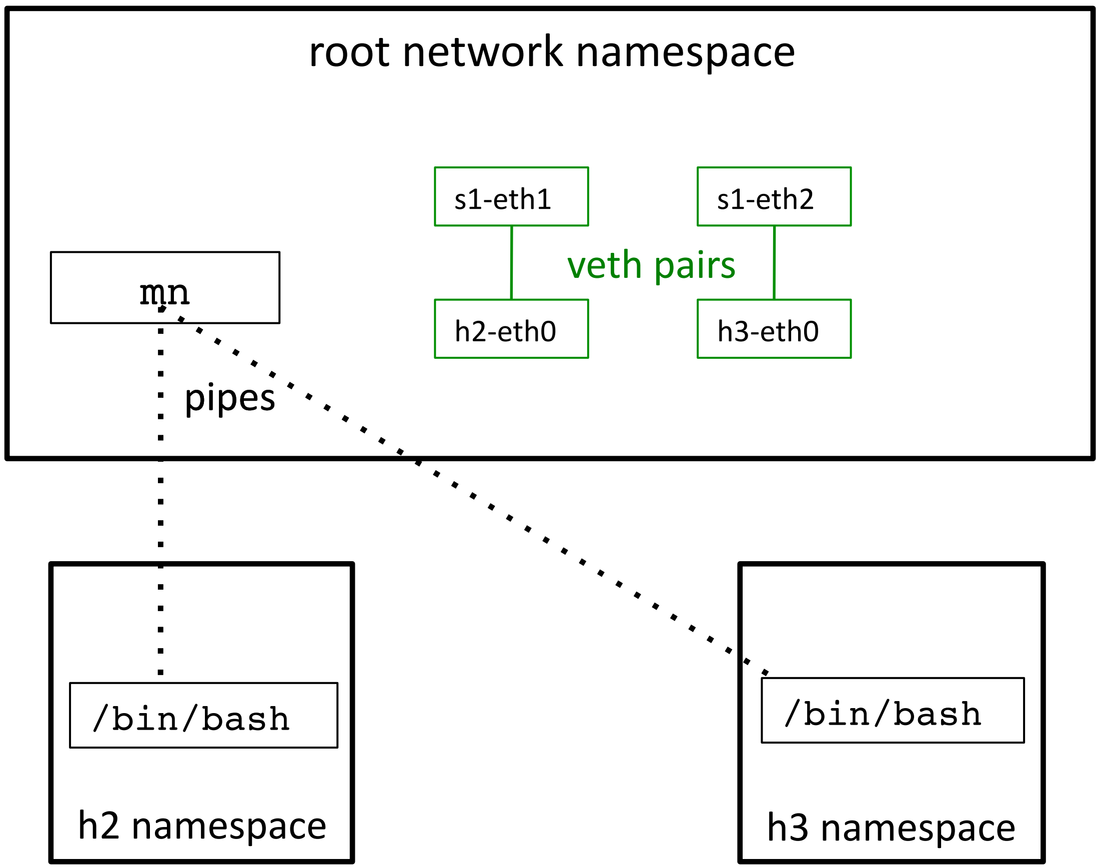
ip link add



Links

```
$> mn --topo minimal \
      --switch ovsk \
      --controller ref
```

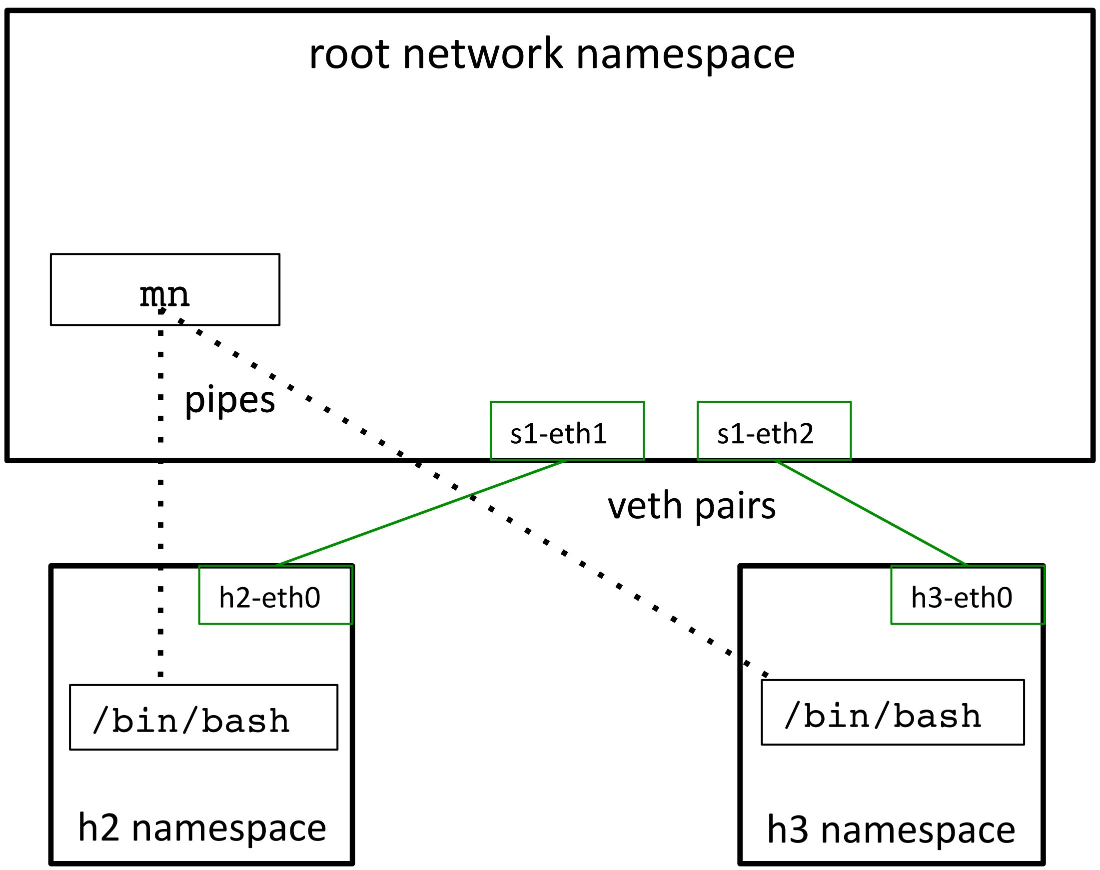
`ip link set name`



```
$> mn --topo minimal \
      --switch ovsk \
      --controller ref
```

Links

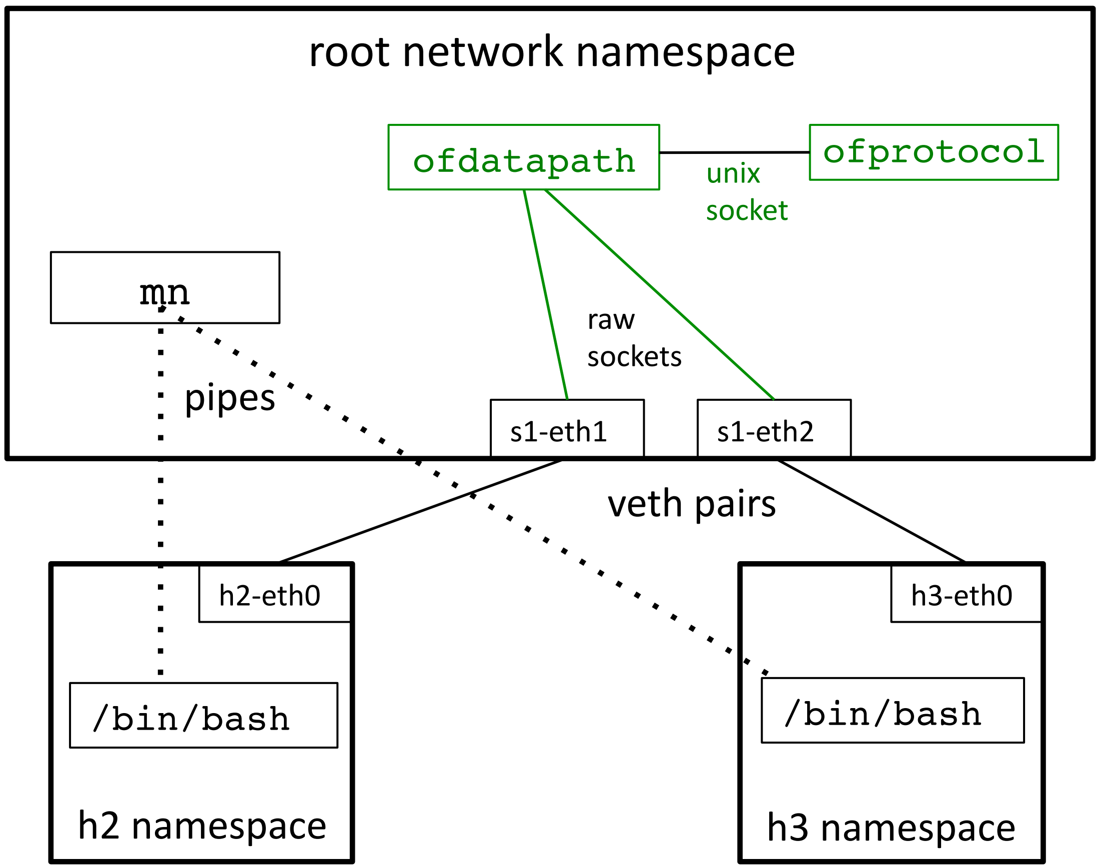
ip link set netns



```
$> mn --topo minimal \
      --switch ovsk \
      --controller ref
```

Switch

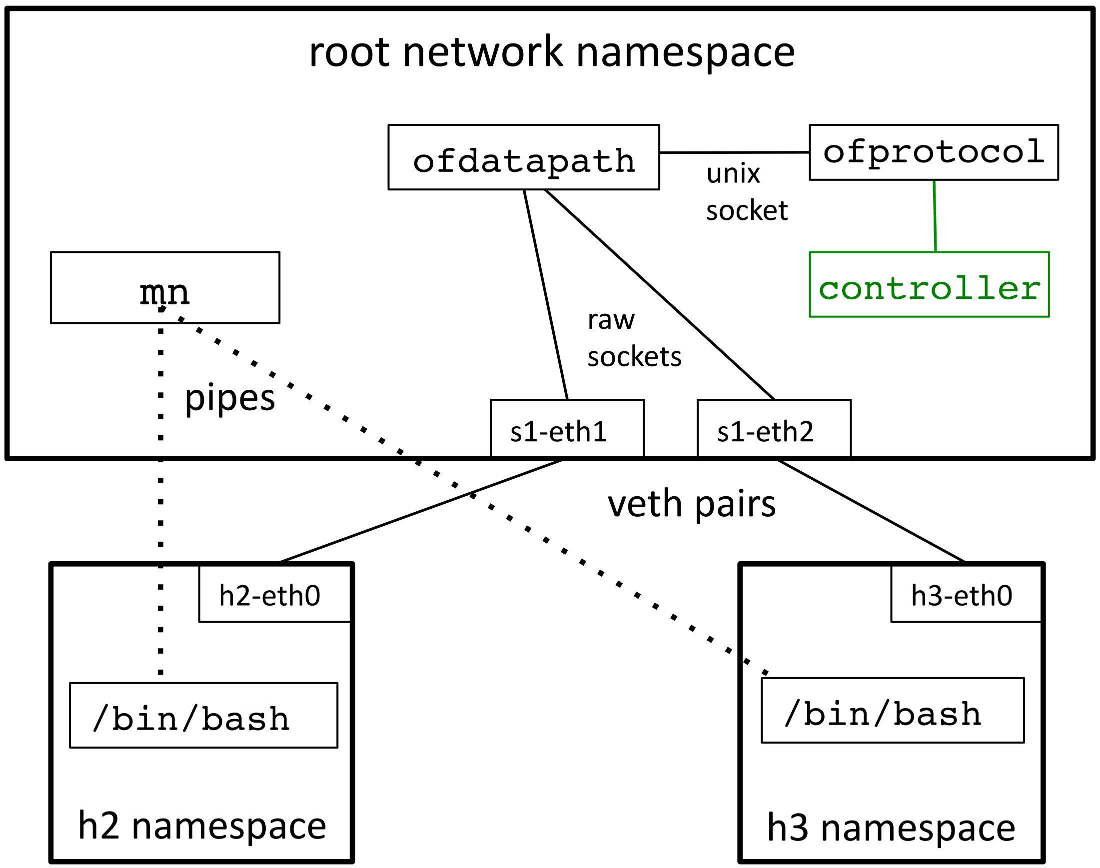
create OpenFlow Switch

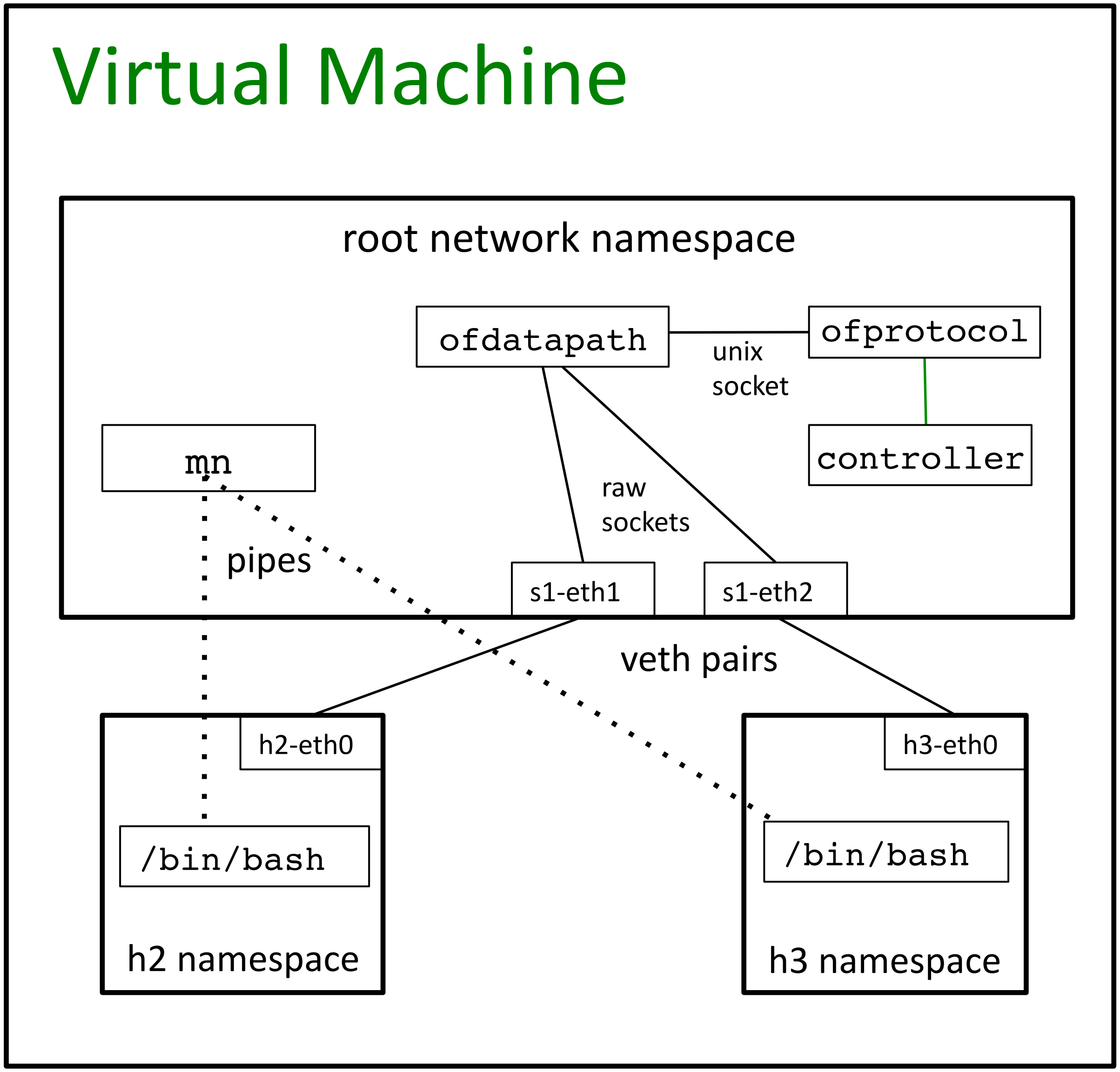


```
$> mn --topo minimal \
      --switch ovsk \
      --controller ref
```

Controller

create controller




Virtual Machine
root network namespace
ofdatapath
unix
socket
ofprotocol
controller
mn
raw
sockets
pipes
s1-eth1
s1-eth2
veth pairs
h2-eth0
h3-eth0
/bin/bash
/bin/bash
h2 namespace
h3 namespace

Mininet example commands

Create a network using mn launcher:

```
mn --switch ovsk --controller nox --topo \ tree,depth=2,fanout=8
    --test pingAll
```

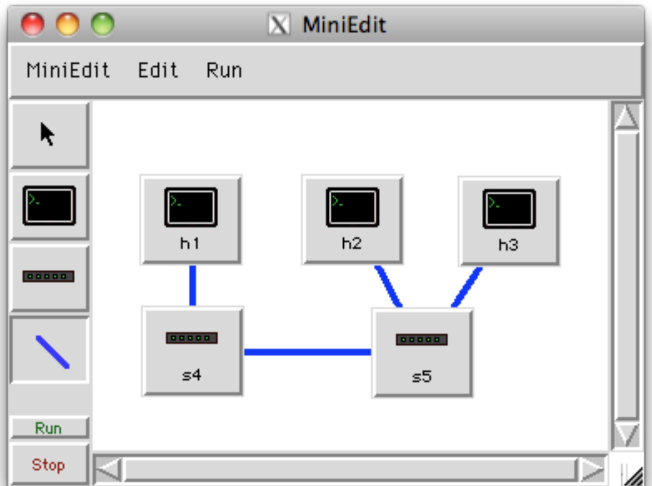
Interact with a network using CLI:

```
mininet> h2 ping h3
mininet> h2 py dir(locals())
```

Customize a network w/API:

```
from mininet.net import Mininet
from mininet.topolib import TreeTopo
tree4 = TreeTopo(depth=2,fanout=2)
net = Mininet(topo=tree4)
net.start()
h1, h4 = net.hosts[0], net.hosts[3]
print h1.cmd('ping -c1 %s' % h4.IP())
net.stop()
```

Apps made with the Mininet API



Evaluation

Startup/Shutdown/Memory

Topology	H	S	Setup(s)	Stop(s)	Mem(MB)
Minimal	2	1	1.0	0.5	6
Linear(100)	100	100	70.7	70.0	112
VL2(4, 4)	80	10	31.7	14.9	73
FatTree(4)	16	20	17.2	22.3	66
FatTree(6)	54	45	54.3	56.3	102
Mesh(10, 10)	40	100	82.3	92.9	152
Tree(4ˆ4)	256	85	168.4	83.9	233
Tree(16ˆ2)	256	17	139.8	39.3	212
Tree(32ˆ2)	1024	33	817.8	163.6	492

lots of switches & hosts
w/reasonable amounts of memory

Microbenchmarks

Operation	Time (ms)
Create a node (host/switch/controller)	10
Run command on a host ('echo hello')	0.3
Add link between two nodes	260
Delete link between two nodes	416
Start user space switch (OpenFlow reference)	29
Stop user space switch (OpenFlow reference)	290
Start kernel switch (Open vSwitch)	332
Stop kernel switch (Open vSwitch)	540

link management is slow

Bandwidth

S (Switches)	User(Mbps)	Kernel(Mbps)	Ratio
1	445	2120	~5x
10	49.9	940	
20	25.7	573	
40	12.6	315	
60	6.2	267	
80	4.15	217	
100	2.96	167	~50x

usable amount of bandwidth

Case Studies

Research Examples

- Ripcord: modular data center
- Asterix: wide-area load balancing
- SCAFFOLD: new internet architecture
- Distributed snapshot demo

Unexpected Uses

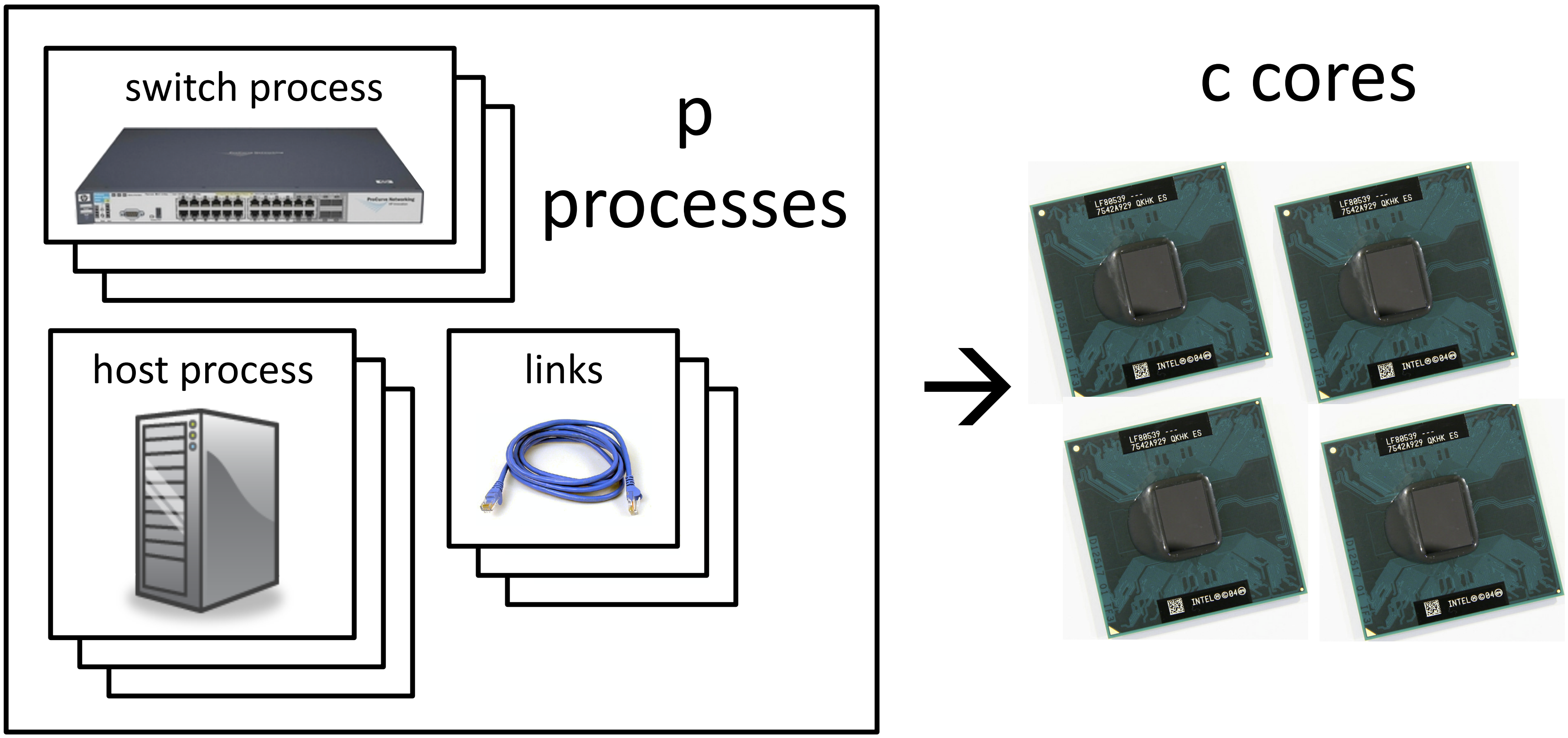
- Tutorials
- Whole-network regression suites
- Bug replication

Limitations

Inherent Limitations

- OS-level virtualization → one kernel only
- Linux containers → Linux programs only
- Cannot match the introspection of an event-driven simulation

Performance Fidelity



p >> c ? time multiplexing

Issues: performance predictability, isolation

Wouldn't it be amazing…

if systems papers were *runnable*.

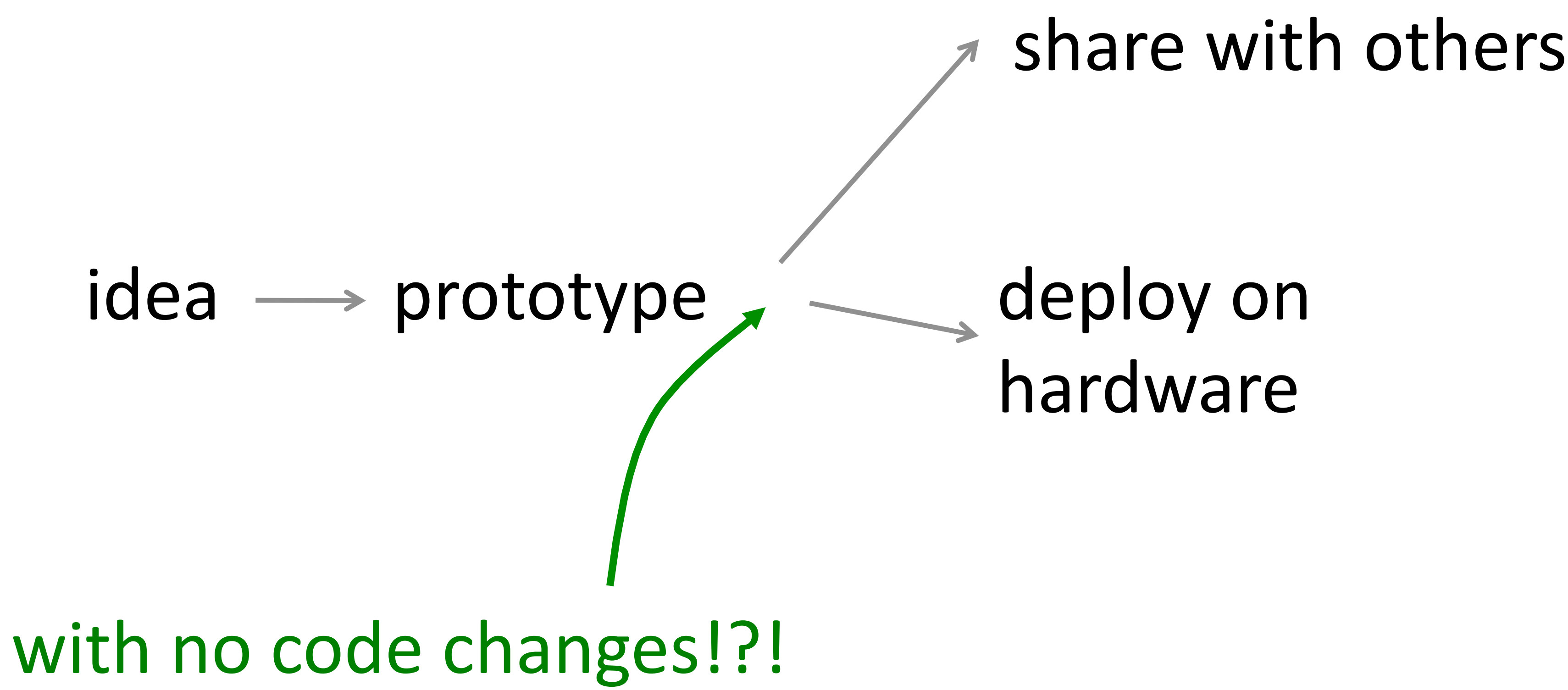
Wouldn't it be amazing…

If systems papers made
replicating their results,
modifying the described system,
and sharing it with others…

… as easy as downloading a file.

Wouldn't it be amazing…

if network systems papers were *more* than runnable.



"A Network in a Laptop…"
is a runnable paper

…which itself describes how to make other runnable papers.

Mininet

- Rapid prototyping
- Scalable
- Shareable
- Functionally correct
- Path to hardware

enables
“runnable papers”
for a subset of
networking

openflow.org/mininet

The SDN Approach

Separate control from the datapath

- i.e. separate policy from mechanism

Datapath: Define minimal network instruction set

- A set of “plumbling primitives”
- A narrow interface: e.g. OpenFlow

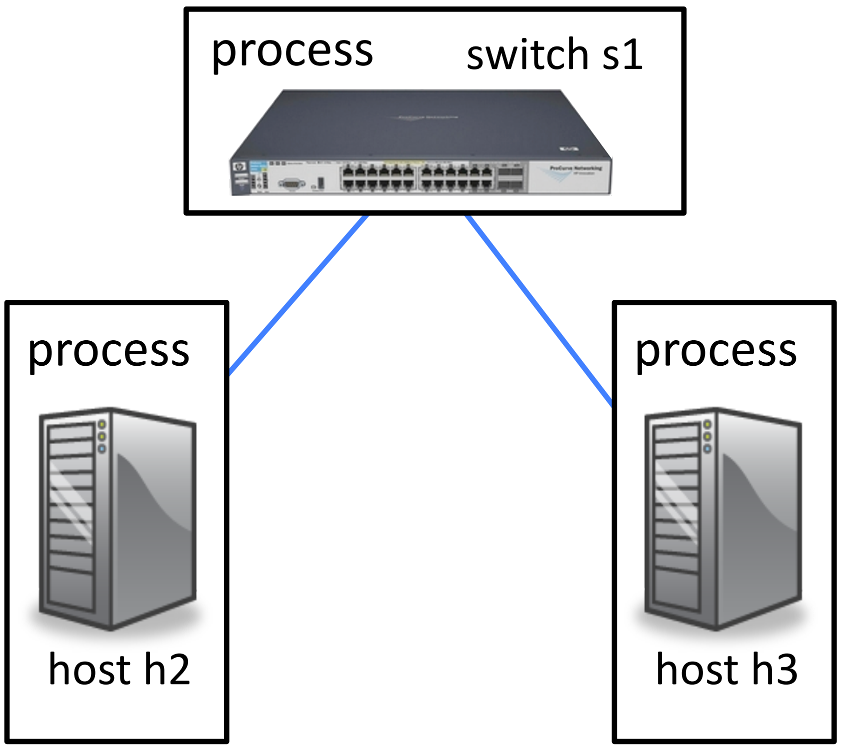
Control: Define a network-wide OS

- An API that others can develop on

How to get performance fidelity?

- Careful process-to-core allocation
- Bandwidth limits
- Scheduling priorities
- Real-time scheduling
- Scheduling groups w/resource isolation

Why not processes?



+ scales better
- breaks applications