Automating System Tests Using Declarative Virtual Machines

Sander van der Burg **Eelco Dolstra**

Department of Software Technology,
Delft University of Technology, Netherlands

21st IEEE International Symposium on Software Reliability Engineering

November 1–4, San Jose, CA, USA

Motivation: Regression testing

Automated regression testing (“make check”) is a good thing

```
[eelco@hagbard:~/Dev/strategoxt/strc-core/tests/test1]$ make check
make  check-TESTS
make[4]: Entering directory `/home/eelco/Dev/strategoxt/strc-core/tests/test1'
building check-TESTS
["./test01"]
PASS: test01
Call("./test02",[])
PASS: test02
(Call("./test03",[]),["./test03"])
PASS: test03
1
PASS: test04
F(3)
PASS: test05
Succ(Succ(Succ(Zero)))
PASS: test06
Succ(Succ(Succ(Zero)))
...
PASS: cs-test03
test suite: cs-test04
test 1
successes: 1
failures: 0
(1,0)
PASS: cs-test04
f_s1(3) = 8
f_s12(3) = 8
f3_s1(3) = 16
PASS: static-links
====================
All 129 tests passed
====================
make[4]: Leaving directory `/home/eelco/Dev/strategoxt/strc-core/tests/test1'

[eelco@hagbard:~/Dev/strategoxt/strc-core/tests/test1]$
```

Motivation: testing at the system level

The problem

- Some tests are easy to automate
  - Unit tests
  - Compiler test suites

Motivation: testing at the system level

The problem

- Some tests are easy to automate
  - Unit tests
  - Compiler test suites
- But others are hard, especially at the *integration* or **system** level

Motivation: testing at the system level

The problem

- Some tests are easy to automate
  - Unit tests
  - Compiler test suites
- But others are hard, especially at the *integration* or **system** level
  - E.g. distributed systems or OS-level software

Motivation: testing at the system level

The problem

- Some tests are easy to automate
  - Unit tests
  - Compiler test suites
- But others are hard, especially at the *integration* or **system** level
  - E.g. distributed systems or OS-level software
- So developers don't bother to write regression tests

Motivation: testing at the system level

The problem

- Some tests are easy to automate
  - Unit tests
  - Compiler test suites
- But others are hard, especially at the *integration* or **system** level
  - E.g. distributed systems or OS-level software
- So developers don't bother to write regression tests
  - Example: Linux kernel doesn't have a "make check"

Motivation: testing at the system level

The problem

- Some tests are easy to automate
  - Unit tests
  - Compiler test suites
- But others are hard, especially at the *integration* or **system** level
  - E.g. distributed systems or OS-level software
- So developers don't bother to write regression tests
  - Example: Linux kernel doesn't have a "make check"

Goal of this paper

Make system tests as easy to write as unit tests

Why are system tests hard to automate?

Environmental dependencies

All artifacts that a test requires from its environment

Why are system tests hard to automate?

Environmental dependencies

All artifacts that a test requires from its environment

Examples:

- Root privileges

Why are system tests hard to automate?

Environmental dependencies

All artifacts that a test requires from its environment

Examples:

- Root privileges
- System services

Why are system tests hard to automate?

Environmental dependencies

All artifacts that a test requires from its environment

Examples:

- Root privileges
- System services
- Multiple machines
  (for distributed systems)

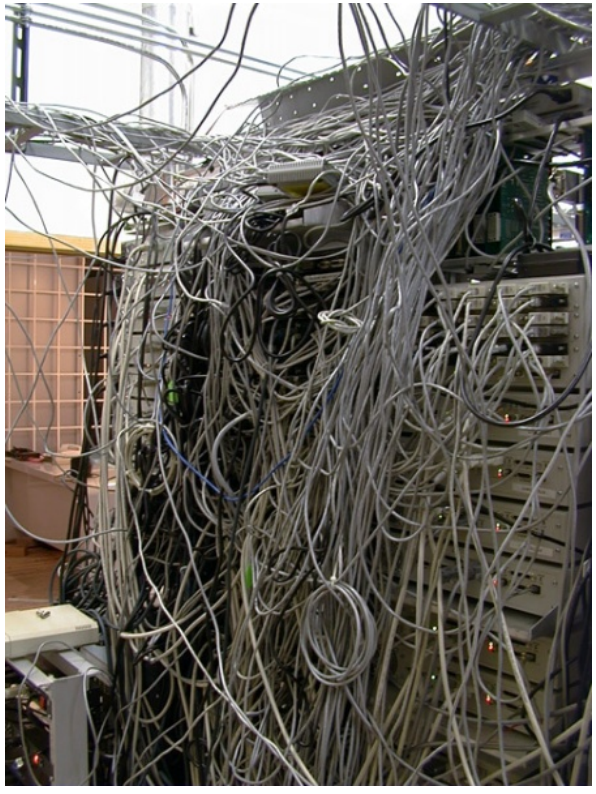
Why are system tests hard to automate?

Environmental dependencies

All artifacts that a test requires from its environment

Examples:

- Root privileges
- System services
- Multiple machines (for distributed systems)
- Specific network topologies



Example: Quake 3

- Quake 3: multiplayer first-person shooter
- Test needs multiple machines:
  - Client(s)
  - Server
- Test needs X11 server on the clients

Example: Transmission test

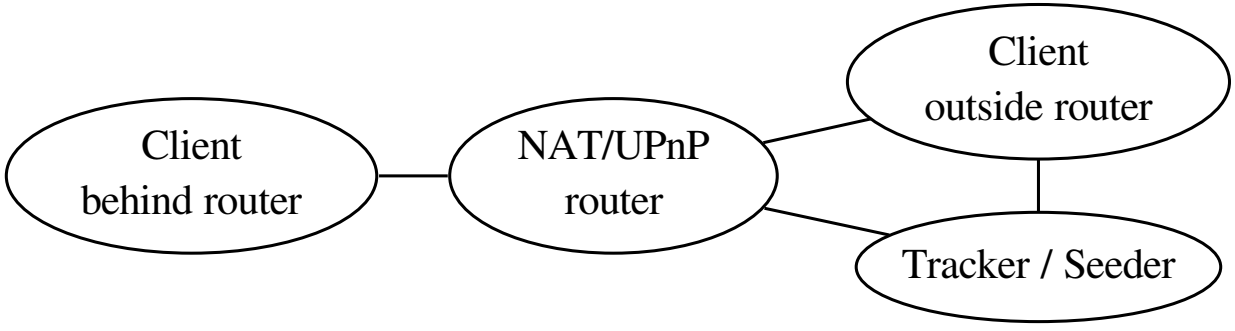
- Transmission is a Bittorrent client

Example: Transmission test

- Transmission is a Bittorrent client
- Needs multiple machines:
  multiple clients + a tracker

Example: Transmission test

- Transmission is a Bittorrent client
- Needs multiple machines:
  multiple clients + a tracker
- Needs special topology for testing
  NAT traversal feature:
  peers should be able to connect
  to peers behind NAT devices



Goal

- Implement the environment needed for a test by *instantiating (Linux) virtual machines*

Goal

- Implement the environment needed for a test by *instantiating (Linux) virtual machines*
- We don't want to build VMs manually!
  - Slow, expensive

Goal

- Implement the environment needed for a test by *instantiating (Linux) virtual machines*
- We don't want to build VMs manually!
  - Slow, expensive
- So the VMs should be *instantiated* automatically from a *specification*

Automated system test

=

Automated system test

$=$

declarative network specification

$+$

Automated system test

$=$

declarative network specification

$+$

imperative test script

Automated system test

$=$

declarative network specification

$+$

imperative test script

What do we need?

- A concise way to specify VM configurations
- An efficient way to build VMs

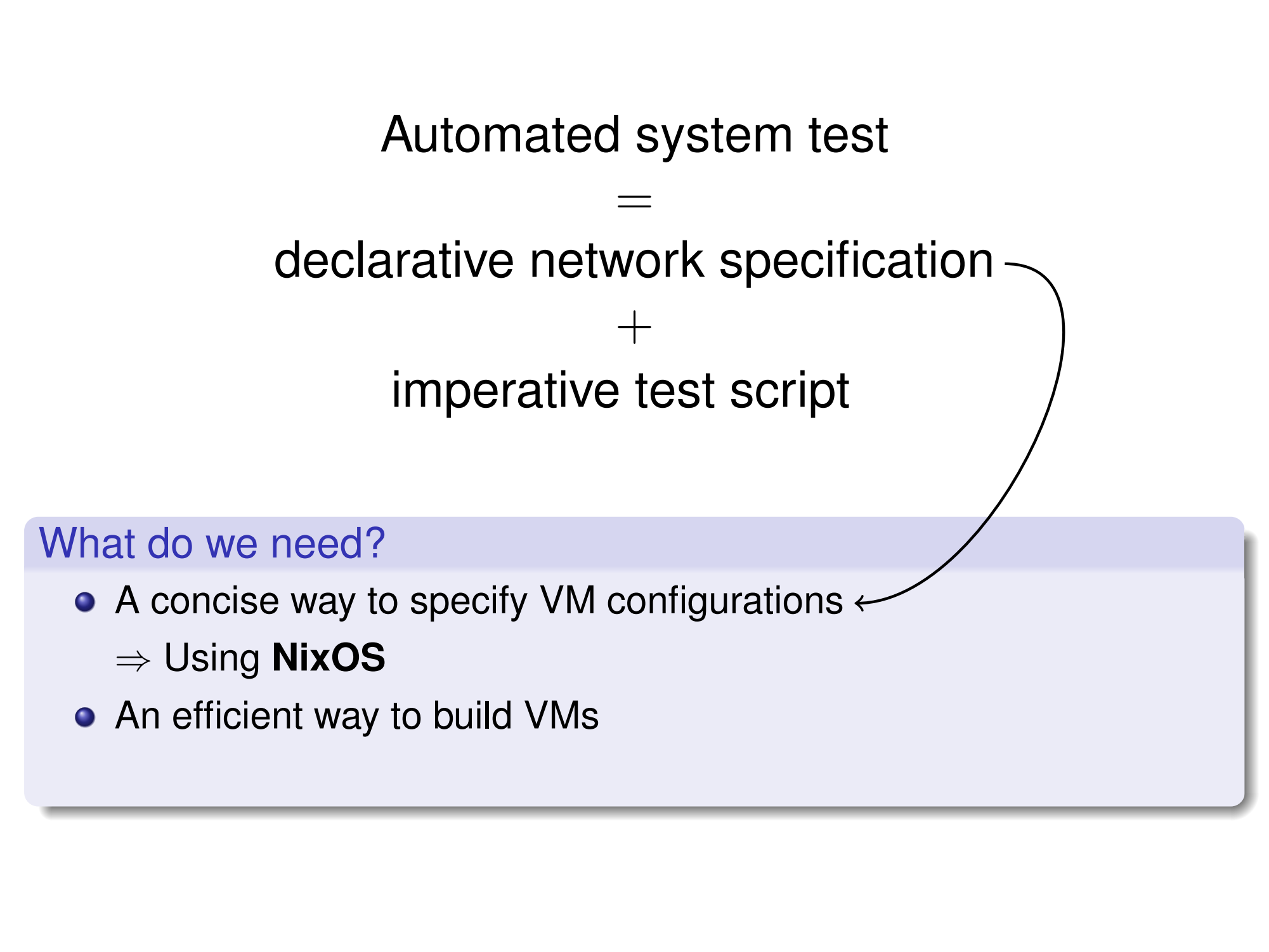
Automated system test

$=$

declarative network specification

$+$

imperative test script



What do we need?

- A concise way to specify VM configurations

  $\Rightarrow$ Using **NixOS**
- An efficient way to build VMs

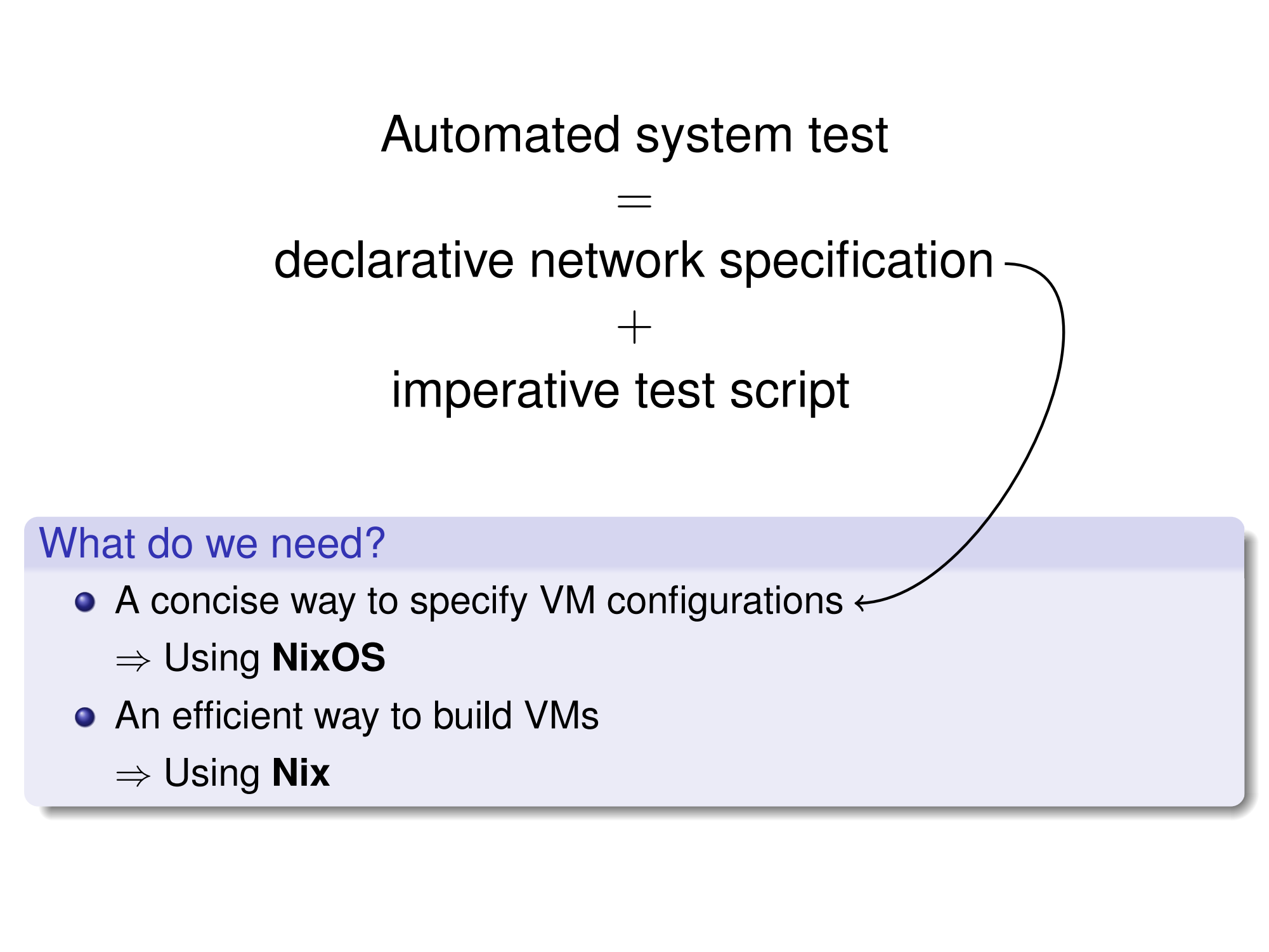
Automated system test

$=$

declarative network specification

$+$

imperative test script



What do we need?

- A concise way to specify VM configurations

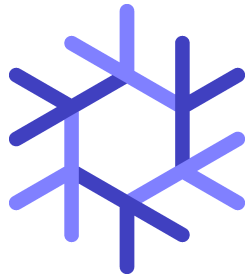
  $\Rightarrow$ Using **NixOS**
- An efficient way to build VMs

  $\Rightarrow$ Using **Nix**

NixOS

- NixOS: a Linux distribution with a *declarative configuration model*

NixOS



- NixOS: a Linux distribution with a *declarative configuration model*
- Machines configured using a declarative specification

```
{ networking.hostName = "hagbard";
  environment.systemPackages = [ pkgs.firefox ];
  services.xserver.enable = true;
  services.httpd.enable = true;
  services.httpd.documentRoot = "/webdata";
  ...
}
```

NixOS

- NixOS: a Linux distribution with a *declarative configuration model*
- Machines configured using a declarative specification

```
{ networking.hostName = "hagbard";
  environment.systemPackages = [ pkgs.firefox ];
  services.xserver.enable = true;
  services.httpd.enable = true;
  services.httpd.documentRoot = "/webdata";
  ...
}
```

- Usually used to install a machine, but here we'll use it to instantiate VMs

Quake 3 testing: network specification

```
nodes =
```

Quake 3 testing: network specification

```
nodes =
  { client =
      { services.xserver.enable = true;
        environment.systemPackages = [ pkgs.quake3 ];
      };
```

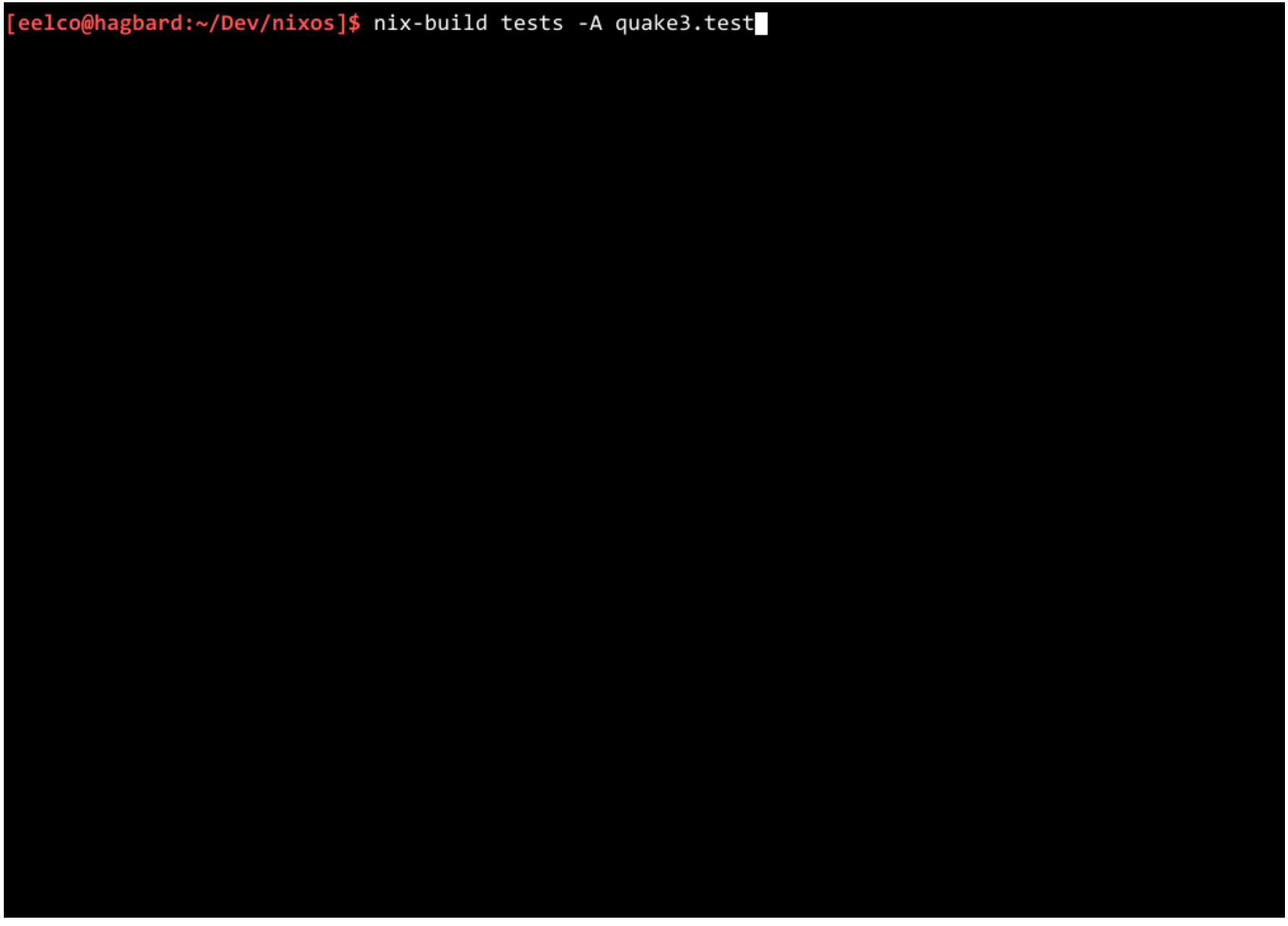
Quake 3 testing: network specification

```
nodes =
  { client =
      { services.xserver.enable = true;
        environment.systemPackages = [ pkgs.quake3 ];
      };

    server =
      { jobs.quake3Server =
          { startOn = "startup";
            exec =
              "${pkgs.quake3}/bin/quake3"
              + " +set dedicated 1 +set g_gametype 0"
              + " +map q3dm7 +addbot grunt 2> /tmp/log";
          };
      };
  };
```

Quake 3 testing: test script

```
testScript =
  ''
    startAll;
    $server→waitForJob("quake3-server");
    $client→waitForX;
    $client→succeed(
      "quake3 +set name Foo +connect server &");
    $server→waitUntilSucceeds("grep 'Foo.*entered the game' /tmp/log");
    sleep 20;
    $client→screenshot("screen.png");
  '';
```

Running the test



Running the test

```
server# [    0.162323] VFS: Disk quotas dquot_6.5.2
server# [    0.162730] Dquot-cache hash table entries: 512 (order 0, 4096 bytes)
server# [    0.163381] msgmni has been set to 742
server# [    0.163878] al
server# [    0.164265] iosafb: mode is 1024x768x16, linelength=2048, pages=1
server# [    0.164644] iosafb: scrolling: redraw
client1# [    0.217778] fbcondecor: console 0 using theme 'default'
server# [    0.169270] vesafb: Truecolor: size=0:5:6:5, shift=0:11:5:0
server# [    0.171344] Console: switching to colour frame buffer device 128x48
client1# [    0.222654] fbcondecor: switched decor state to 'on' on console 0
client1# [    0.223499] fb0: VESA VGA frame buffer device
client1# [    0.223978] Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled
client2# [    0.153270] Freeing initrd memory: 4474k freed
client2# [    0.154825] Scanning for low memory corruption every 60 seconds
client2# [    0.155652] audit: initializing netlink socket (disabled)
client2# [    0.156205] type=2000 audit(1285923052.156:1): initialized
client2# [    0.156962] HugeTLB registered 2 MB page size, pre-allocated 0 pages
client2# [    0.159641] VFS: Disk quotas dquot_6.5.2
client2# [    0.160093] Dquot-cache hash table entries: 512 (order 0, 4096 bytes)
client2# [    0.160777] msgmni has been set to 742
client2# [    0.161348] alg: No test for stdrng (krng)
client2# [    0.161725] io scheduler noop registered
client2# [    0.162143] io scheduler cfq registered (default)
client2# [    0.164892] vesafb: framebuffer at 0xf0000000, mapped to 0xffffc90000080000, using 3072k
, total 4096k
client2# [    0.165699] vesafb: mode is 1024x768x16, linelength=2048, pages=1
client2# [    0.166186] vesafb: scrolling: redraw
client2# [    0.166521] vesafb: Truecolor: size=0:5:6:5, shift=0:11:5:0
client2# [    0.168360] Console: switching to colour frame buffer device 128x48
server# [    0.220447] fbcondecor: console 0 using theme 'default'
server# [    0.224845] fbcondecor: switched decor state to 'on' on console 0
server# [    0.225680] fb0: VESA VGA frame buffer device
server# [    0.226189] Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled
client2# [    0.216342] fbcondecor: console 0 using theme 'default'
client2# [    0.220532] fbcondecor: switched decor state to 'on' on console 0
client2# [    0.221360] fb0: VESA VGA frame buffer device
client2# [    0.221864] Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled
```

Running the test

```
client1# running assembler < /tmp/ui.s_us9GT1 > /tmp/ui.o_acGjF5
client1# ^1as failed with status 255
client1# ui loaded in 2317696 bytes on the hunk
client1# 9 arenas parsed
client1# 6 bots parsed
client1# Loading vm file vm/cgame.qvm...
client1# compiling cgame
client1# running assembler < /tmp/cgame.s_vTUFV9 > /tmp/cgame.o_Btu2be
client1# ^1as failed with status 255
client1# cgame loaded in 5773088 bytes on the hunk
server: running command: grep -q 'Foo.*entered the game' /tmp/log
server: exit status 1
client1# stitched 0 LoD cracks
client1# ...loaded 5823 faces, 189 meshes, 49 trisurfs, 37 flares
server: running command: grep -q 'Foo.*entered the game' /tmp/log
server: exit status 1
client2# CL_InitCGame:  7.14 seconds
client2# 41 msec to draw all images
client2# Com_TouchMemory: 0 msec
client2# Foo^7 connected
client2# Bar^7 entered the game
server: running command: grep -q 'Foo.*entered the game' /tmp/log
server: exit status 1
server: running command: grep -q 'Foo.*entered the game' /tmp/log
server: exit status 1
client1# CL_InitCGame:  4.83 seconds
client1# 39 msec to draw all images
client1# Com_TouchMemory: 0 msec
server: running command: grep -q 'Foo.*entered the game' /tmp/log
client1# Bar^7 entered the game
client1# Foo^7 entered the game
server: exit status 0
server: running command: grep -q 'Bar.*entered the game' /tmp/log
server: exit status 0
client2# Foo^7 entered the game
client2# Bar^7 ate Daemia^7's rocket
client1# Bar^7 ate Daemia^7's rocket
```

Running the test

```
server: exit status 1
client2# CL_InitCGame:  7.14 seconds
client2# 41 msec to draw all images
client2# Com_TouchMemory: 0 msec
client2# Foo^7 connected
client2# Bar^7 entered the game
server: running command: grep -q 'Foo.*entered the game' /tmp/log
server: exit status 1
server: running command: grep -q 'Foo.*entered the game' /tmp/log
server: exit status 1
client1# CL_InitCGame:  4.83 seconds
client1# 39 msec to draw all images
client1# Com_TouchMemory: 0 msec
server: running command: grep -q 'Foo.*entered the game' /tmp/log
client1# Bar^7 entered the game
client1# Foo^7 entered the game
server: exit status 0
server: running command: grep -q 'Bar.*entered the game' /tmp/log
server: exit status 0
client2# Foo^7 entered the game
client2# Bar^7 ate Daemia^7's rocket
client1# Bar^7 ate Daemia^7's rocket
client1: sending monitor command: screendump /nix/store/vxaqkkdqihz6hlx3w3cy65790wasckjy-vm-test-run
/screen1.png.ppm
client2: sending monitor command: screendump /nix/store/vxaqkkdqihz6hlx3w3cy65790wasckjy-vm-test-run
/screen2.png.ppm
client1: running command: test -e /sys/kernel/debug/gcov
client1: exit status 1
server: running command: test -e /sys/kernel/debug/gcov
server: exit status 1
client2: running command: test -e /sys/kernel/debug/gcov
client2: exit status 1
killing client1 (pid 23758)
killing server (pid 23769)
killing client2 (pid 23780)
/nix/store/vxaqkkdqihz6hlx3w3cy65790wasckjy-vm-test-run

[eelco@hagbard:~/Dev/nixos]$
```

Running the test

```
server: exit status 1
client2# CL_InitCGame:  7.14 seconds
client2# 41 msec to draw all images
client2# Com_TouchMemory: 0 msec
client2# Foo^7 connected
client2# Bar^7 entered the game
server: running command: grep -q 'Foo.*entered the game' /tmp/log
server: exit status 1
server: running command: grep -q 'Foo.*ent
server: exit status 1
client1# CL_InitCGame:  4.83 seconds
client1# 39 msec to draw all images
client1# Com_TouchMemory: 0 msec
server: running command: grep -q 'Foo.*ent
client1# Bar^7 entered the game
client1# Foo^7 entered the game
server: exit status 0
server: running command: grep -q 'Bar.*ent
server: exit status 0
client2# Foo^7 entered the game
client2# Bar^7 ate Daemia^7's rocket
client1# Bar^7 ate Daemia^7's rocket
client1: sending monitor command: screendu
/screen1.png.ppm
client2: sending monitor command: screendu
/screen2.png.ppm
client1: running command: test -e /sys/ker
client1: exit status 1
server: running command: test -e /sys/kern
server: exit status 1
client2: running command: test -e /sys/ker
client2: exit status 1
killing client1 (pid 23758)
killing server (pid 23769)
killing client2 (pid 23780)
/nix/store/vxaqkkdqihz6hlx3w3cy65790wasckjy-vm-test-run

[eelco@hagbard:~/Dev/nixos]$
```

Network topologies: Transmission test

```
tracker =
  { environment.systemPackages = [ pkgs.transmission pkgs.bittorrent ];
    services.httpd.enable = true;
    services.httpd.documentRoot = "/tmp";
  };
router =
  { environment.systemPackages = [ iptables miniupnpd ];
    virtualisation.vlans = [ 1 2 ];
  };
client1 =
  { environment.systemPackages = [ transmission ];
    virtualisation.vlans = [ 2 ];
    networking.defaultGateway = nodes.router
      .config.networking.ifaces.eth2.ipAddress;
  };
client2 =
  { environment.systemPackages = [ transmission ];
  };
```

Network topologies: Transmission test

```
tracker =
  { environment.systemPackages = [ pkgs.transmission pkgs.bittorrent ];
    services.httpd.enable = true;
    services.httpd.documentRoot = "/tmp";
  };
router =
  { environment.systemPackages = [ iptables miniupnpd ];
    virtualisation.vlans = [ 1 2 ];
  };
client1 =
  { environment.systemPackages = [ transmission ];
    virtualisation.vlans = [ 2 ];
    networking.defaultGateway = nodes.router
      .config.networking.ifaces.eth2.ipAddress;
  };
client2 =
  { environment.systemPackages = [ transmission ];
  };
```

Implementation

Nix

- NixOS is based on **Nix**

Nix

- NixOS is based on **Nix**
- Nix is a purely functional package manager

Nix

- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**

Nix

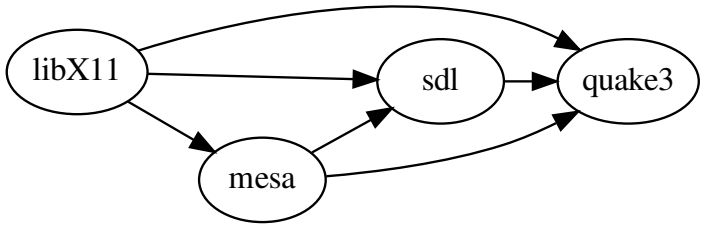
- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**
- Nix expression ≈ Makefile

```
quake3 = stdenv.mkDerivation {
  name = "quake3";
  src = ./quake3-srcs;
  buildInputs = [ libX11 sdl mesa ];
  buildCommand =
    ''
      ./configure --prefix=$out
      make
      make install
    '';
};

libX11 = stdenv.mkDerivation {
  name = "libX11-1.3.4";
  ...
};

sdl = ...;
mesa = ...;
```

Nix

- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**
- Nix expression $\approx$ Makefile
- Nix expressions evaluate to **dependency graph** of build actions (like a Makefile)

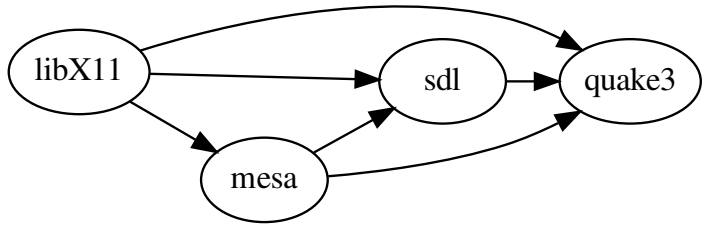
Dependency graph:



Nix

- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**
- Nix expression $\approx$ Makefile
- Nix expressions evaluate to **dependency graph** of build actions (like a Makefile)
- Packages are stored in isolation in the **Nix store**

Dependency graph:



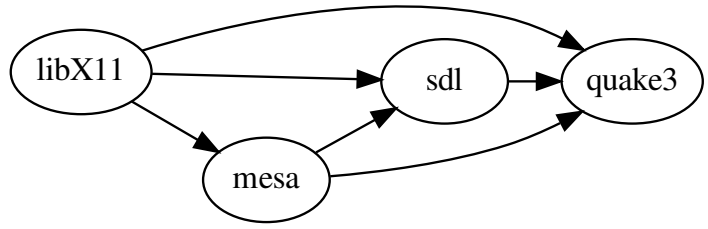
Nix store:

```
/nix/store
├─ n89h90y8k0r2...-gcc-4.4.3
│  └─ bin
│     ├─ gcc
│     └─ g++
└─ 9pq9d484l2dg...-glibc-2.11.1
   └─ lib
      ├─ libc-2.11.1.so
      └─ ld-linux-x86-64.so.2
```

Nix

- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**
- Nix expression $\approx$ Makefile
- Nix expressions evaluate to **dependency graph** of build actions (like a Makefile)
- Packages are stored in isolation in the **Nix store**
  - Immutable

Dependency graph:



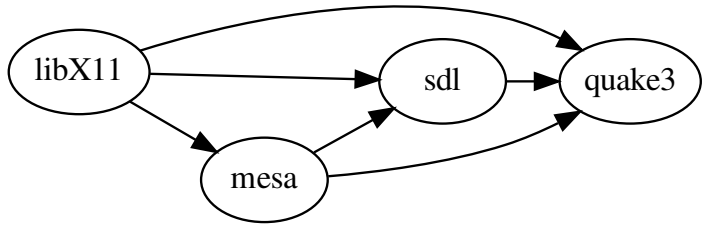
Nix store:

```
/nix/store
├─ n89h90y8k0r2...-gcc-4.4.3
│  └─ bin
│     ├─ gcc
│     └─ g++
└─ 9pq9d484l2dg...-glibc-2.11.1
   └─ lib
      ├─ libc-2.11.1.so
      └─ ld-linux-x86-64.so.2
```

Nix

- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**
- Nix expression $\approx$ Makefile
- Nix expressions evaluate to **dependency graph** of build actions (like a Makefile)
- Packages are stored in isolation in the **Nix store**
  - Immutable
  - Unique file names

Dependency graph:



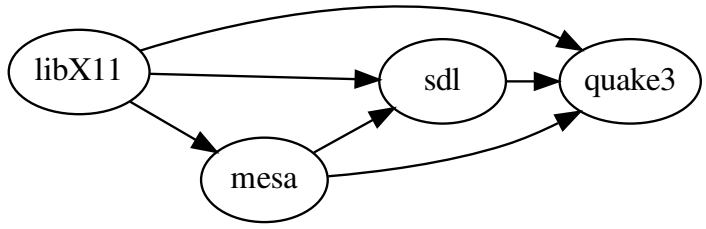
Nix store:

```
/nix/store
├─ n89h90y8k0r2...-gcc-4.4.3
│  └─ bin
│     ├─ gcc
│     └─ g++
└─ 9pq9d484l2dg...-glibc-2.11.1
   └─ lib
      ├─ libc-2.11.1.so
      └─ ld-linux-x86-64.so.2
```

Nix

- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**
- Nix expression $\approx$ Makefile
- Nix expressions evaluate to **dependency graph** of build actions (like a Makefile)
- Packages are stored in isolation in the **Nix store**
  - Immutable
  - Unique file names
- Each build action in the graph produces a path in the Nix store

Dependency graph:



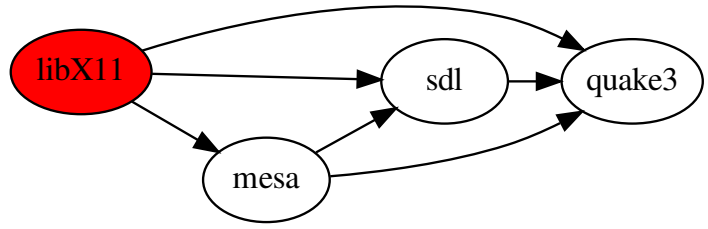
Nix store:

```
/nix/store
├─ n89h90y8k0r2...-gcc-4.4.3
│  └─ bin
│     ├─ gcc
│     └─ g++
└─ 9pq9d484l2dg...-glibc-2.11.1
   └─ lib
      ├─ libc-2.11.1.so
      └─ ld-linux-x86-64.so.2
```

Nix

- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**
- Nix expression $\approx$ Makefile
- Nix expressions evaluate to **dependency graph** of build actions (like a Makefile)
- Packages are stored in isolation in the **Nix store**
  - Immutable
  - Unique file names
- Each build action in the graph produces a path in the Nix store

Dependency graph:



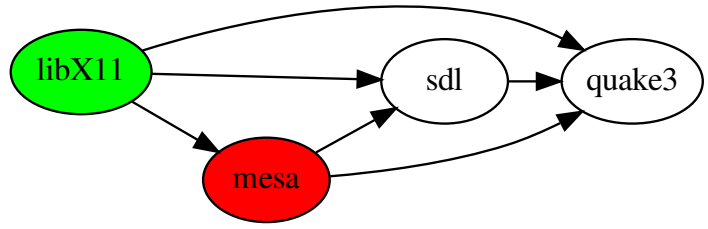
Nix store:

```
/nix/store
└─ 8asg5kbfsbd3...-libX11-1.3.4
   └─ lib
      └─ libX11.so.6
```

Nix

- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**
- Nix expression $\approx$ Makefile
- Nix expressions evaluate to **dependency graph** of build actions (like a Makefile)
- Packages are stored in isolation in the **Nix store**
  - Immutable
  - Unique file names
- Each build action in the graph produces a path in the Nix store

Dependency graph:



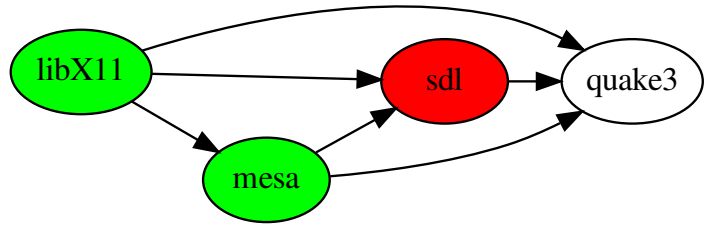
Nix store:

```
/nix/store
├─ 8asg5kbfsbd3...-libX11-1.3.4
│  └─ lib
│     └─ libX11.so.6
└─ 52abfi7a0nl8...-mesa-7.8.2
   └─ lib
      └─ libGL.so.1.2
```

Nix

- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**
- Nix expression $\approx$ Makefile
- Nix expressions evaluate to **dependency graph** of build actions (like a Makefile)
- Packages are stored in isolation in the **Nix store**
  - Immutable
  - Unique file names
- Each build action in the graph produces a path in the Nix store

Dependency graph:



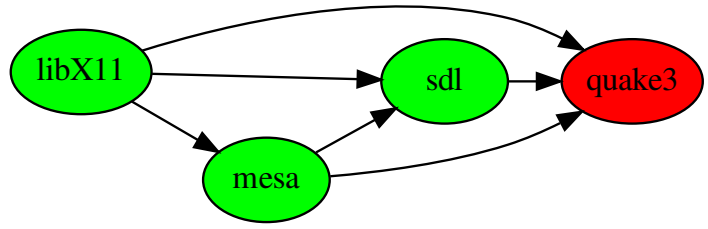
Nix store:

```
/nix/store
├─ 8asg5kbfsbd3...-libX11-1.3.4
│  └─ lib
│     └─ libX11.so.6
├─ 52abfi7a0nl8...-mesa-7.8.2
│  └─ lib
│     └─ libGL.so.1.2
└─ i5lxg4bl2zsa...-SDL-1.2.14
   └─ lib
      └─ libSDL-1.2.so.0.11.3
```

Nix

- NixOS is based on **Nix**
- Nix is a purely functional ~~package manager~~ **Make**
- Nix expression $\approx$ Makefile
- Nix expressions evaluate to **dependency graph** of build actions (like a Makefile)
- Packages are stored in isolation in the **Nix store**
  - Immutable
  - Unique file names
- Each build action in the graph produces a path in the Nix store

Dependency graph:

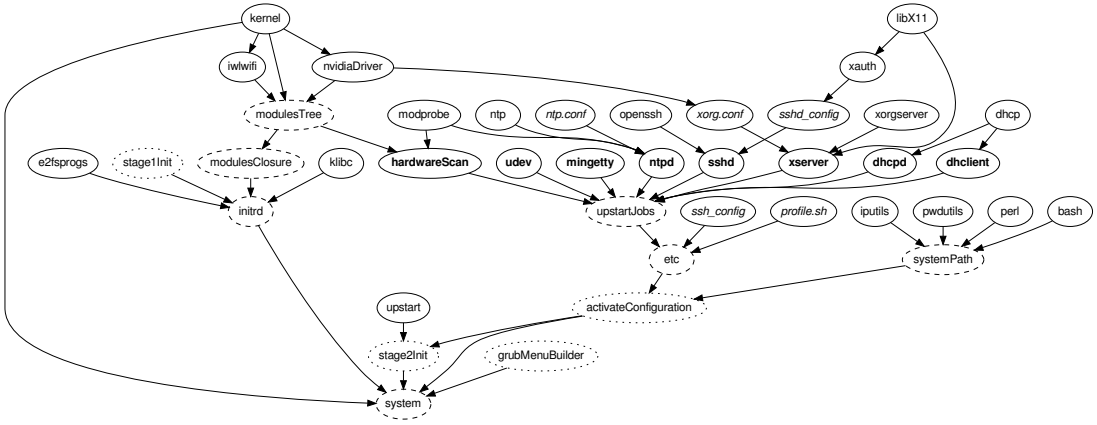


Nix store:

```
/nix/store
├─ 8asg5kbfsbd3...-libX11-1.3.4
│  └─ lib
│     └─ libX11.so.6
├─ 52abfi7a0nl8...-mesa-7.8.2
│  └─ lib
│     └─ libGL.so.1.2
├─ i5lxg4bl2zsa...-SDL-1.2.14
│  └─ lib
│     └─ libSDL-1.2.so.0.11.3
└─ bqd0my8b4f65...-quake3
   └─ bin
      └─ quake3
```

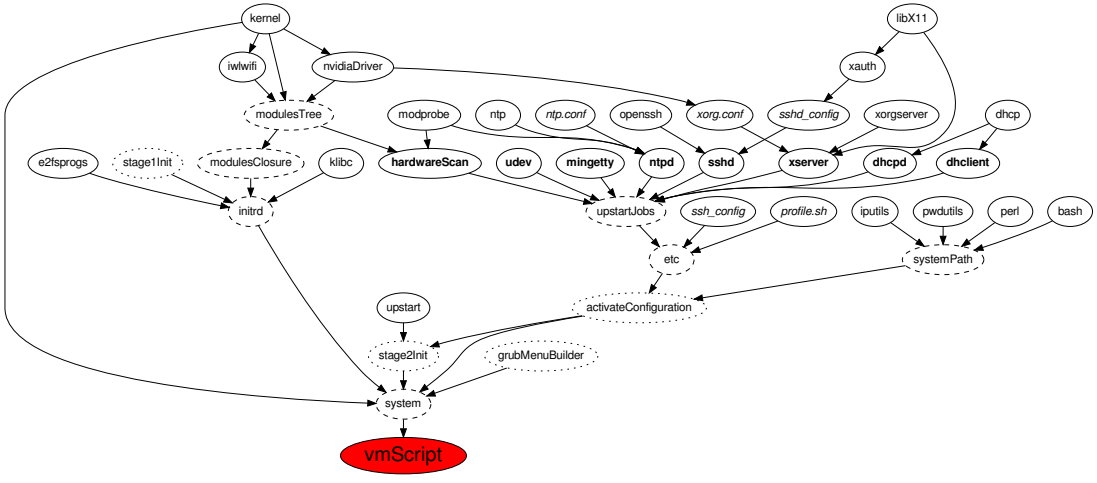
Building NixOS VMs

- NixOS = big dependency graph: packages, kernel, boot scripts, system services, static config files...



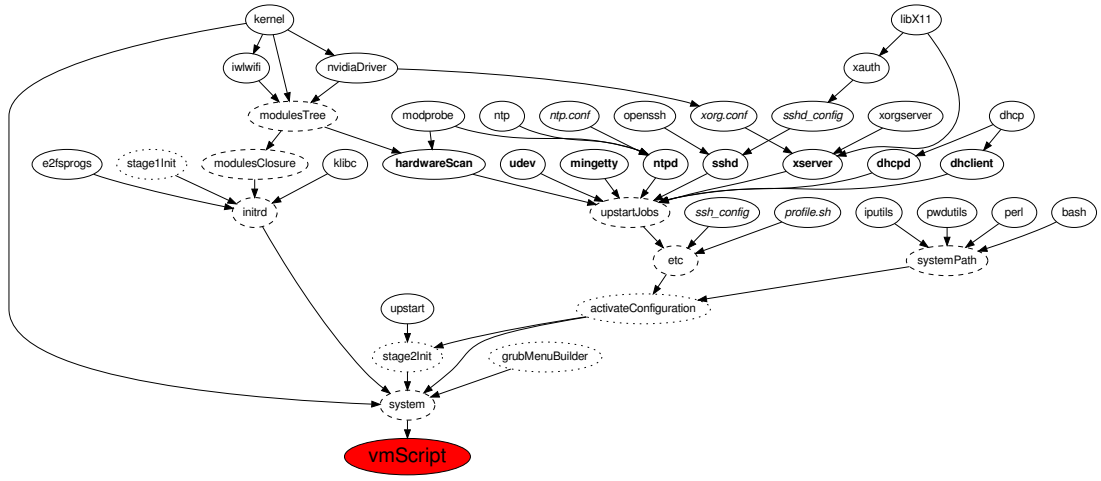
Building NixOS VMs

- NixOS = big dependency graph: packages, kernel, boot scripts, system services, static config files...
- NixOS VM: one extra step to build a **script** that runs **QEMU/KVM**



Building NixOS VMs

- NixOS = big dependency graph: packages, kernel, boot scripts, system services, static config files...
- NixOS VM: one extra step to build a **script** that runs **QEMU/KVM**




```
${pkgs.qemu_kvm}/bin/qemu-system-x86_64 -smb /
 -kernel ${config.boot.kernelPackages.kernel}
 -initrd ${config.system.build.initialRamdisk}
 -append "init=..."
```

Efficient VM instantiation

- We **don't** generate disk images

Efficient VM instantiation

- We **don't** generate disk images
- Rather, the VM mounts the Nix store of the host (using SMB/CIFS)

Efficient VM instantiation

- We **don't** generate disk images
- Rather, the VM mounts the Nix store of the host (using SMB/CIFS)
- Thanks to the purely functional nature of the Nix store:
  VMs don't interfere with each other
  - Not possible if we were using (say)
    `/bin` and `/etc` of an Ubuntu system

Experience

- NixOS continuous builds

- GNU Project integration testing
  - E.g. GNU C Library (Glibc)

- Other free software projects

Conclusion

Contributions

- Concise specifications of machines and networks needed for system tests
  - Thanks to the declarative model of NixOS
- Efficient method to instantiate those specifications
  - Thanks to the purely functional nature of Nix

Bottom line: makes it easy to write automated tests that would otherwise be infeasible

More information

Web: `http://nixos.org/`

E-mail: `e.dolstra@tudelft.nl`, `s.vanderburg@tudelft.nl`[1]

Bonus slides

Distributed code coverage

- Example of the advantage of a functional build specification language
- Can easily adapt the dependency graph to apply coverage instrumentation
- Gather coverage data from all VMs and combine it into one report
- Useful because different code paths may be exercised on the client and the server

httpd-2.2.13/os/unix		**36.6 %**	64 / 175	**75.0 %**	12 / 16
httpd-2.2.13/server		**48.0 %**	3601 / 7508	**60.1 %**	351 / 584
httpd-2.2.13/server/mpm/prefork		**47.1 %**	220 / 467	**60.9 %**	14 / 23
linux-2.6.28.10/arch/x86/include/asm		**49.7 %**	446 / 897	**6.2 %**	2 / 32
linux-2.6.28.10/arch/x86/include/asm/mach-default		**100.0 %**	5 / 5	-	0 / 0
linux-2.6.28.10/arch/x86/include/asm/xen		**0.0 %**	0 / 80	-	0 / 0
linux-2.6.28.10/arch/x86/lib		**62.3 %**	119 / 191	**62.8 %**	27 / 43
linux-2.6.28.10/arch/x86/mach-default		**59.4 %**	19 / 32	**87.5 %**	7 / 8
linux-2.6.28.10/arch/x86/mm		**42.5 %**	852 / 2006	**51.3 %**	80 / 156

Why NixOS?

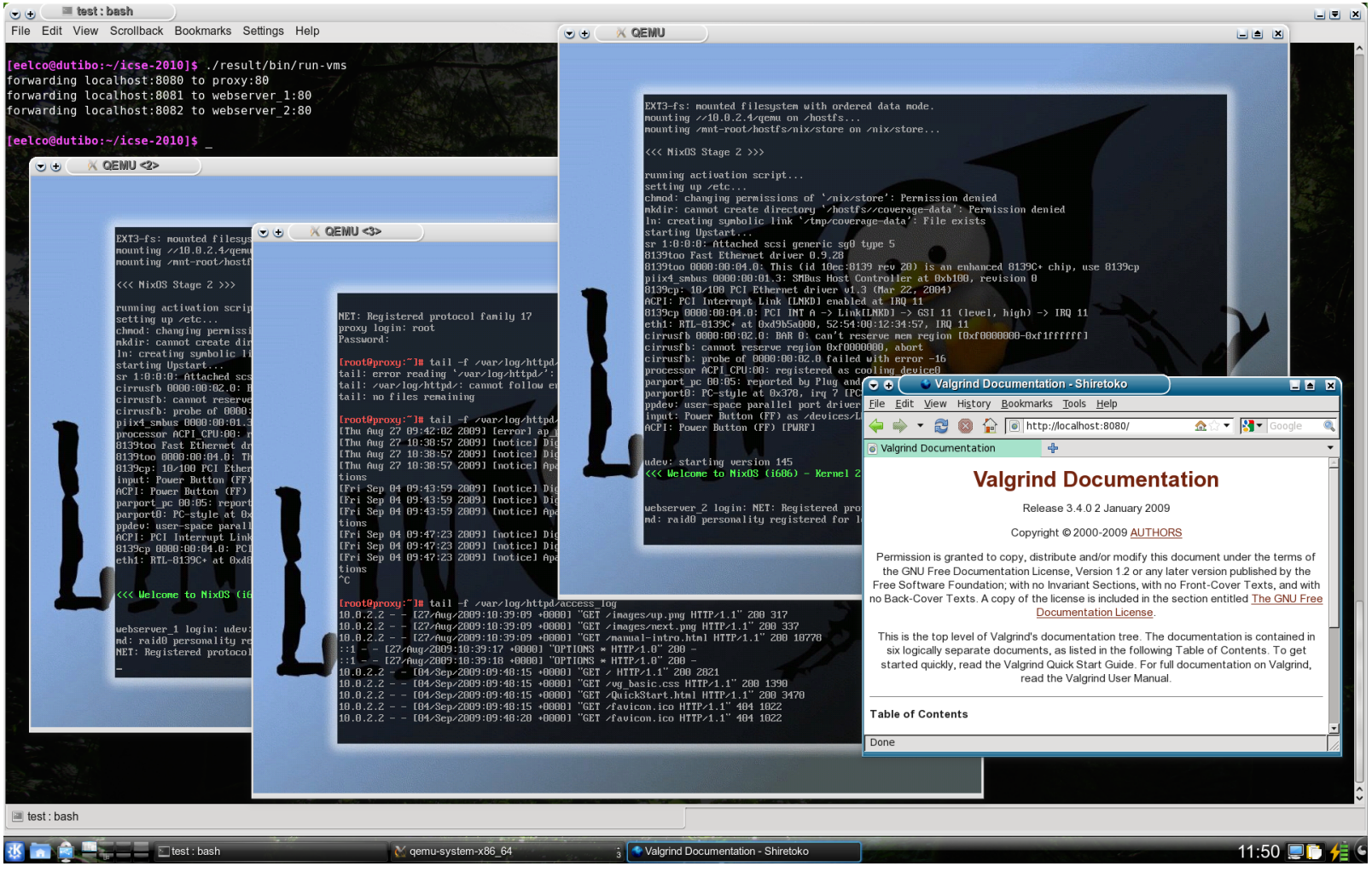
Why not just generate (say) Ubuntu 10.10 disk images?
We have a function for that, but...

- It's slow and expensive to generate full disk images.
- Not declarative; inconvient for specifying tests.

Note: host system can be any Linux distribution.

Interactive testing

We can also run the VMs from the declarative model interactively.



Evaluation

Test	# VMs	Duration (s)	Memory (MiB)
`empty`	1	45.9	166
`openssh`	1	53.7	267
`kde4`	1	140.4	433
`subversion`	2	104.8	329
`trac`	4	159.4	756
`proxy`	4	65.4	477
`quake3`	3	80.6	528
`transmission`	4	89.5	457
`installation`	2	302.7	751
`nfs`	3	259.7	358

Table: Test resource consumption