CCNx experimentation on PlanetLab using NEPI

Alina Quereilhac

{alina.quereilhac@inria.fr}

Team DIANA

INRIA Sophia Antipolis, France

Outline

- Introduction to NEPI
- CCNx experiment example

Introduction to NEPI

Experiment resources

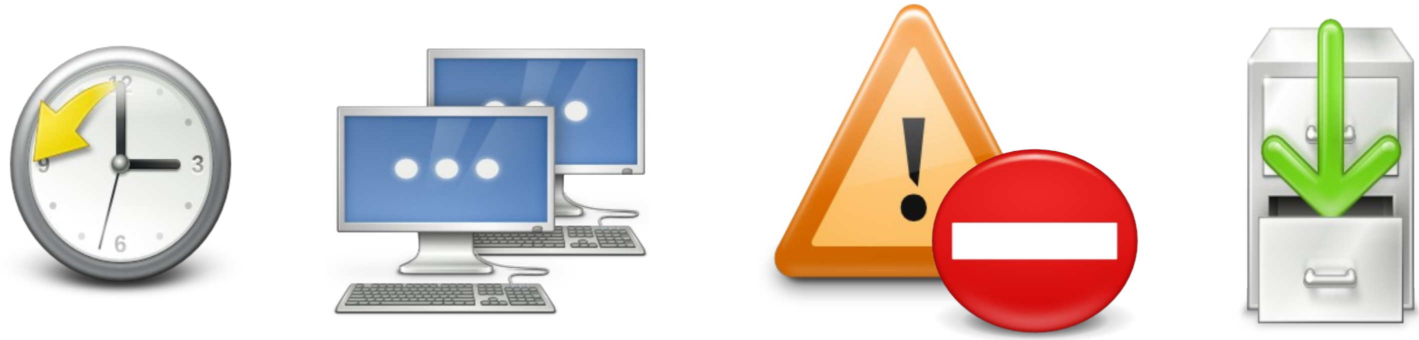
- We need resources to conduct network experiments
- There is a large offer of resources provided by different platforms/testbeds
- But different platforms are accessed and used in different ways, making it necessary to master different tools and technologies

The challenge

How to make it easier to take advantage of the wide offer of network experimentation resources ?

Experimentation issues

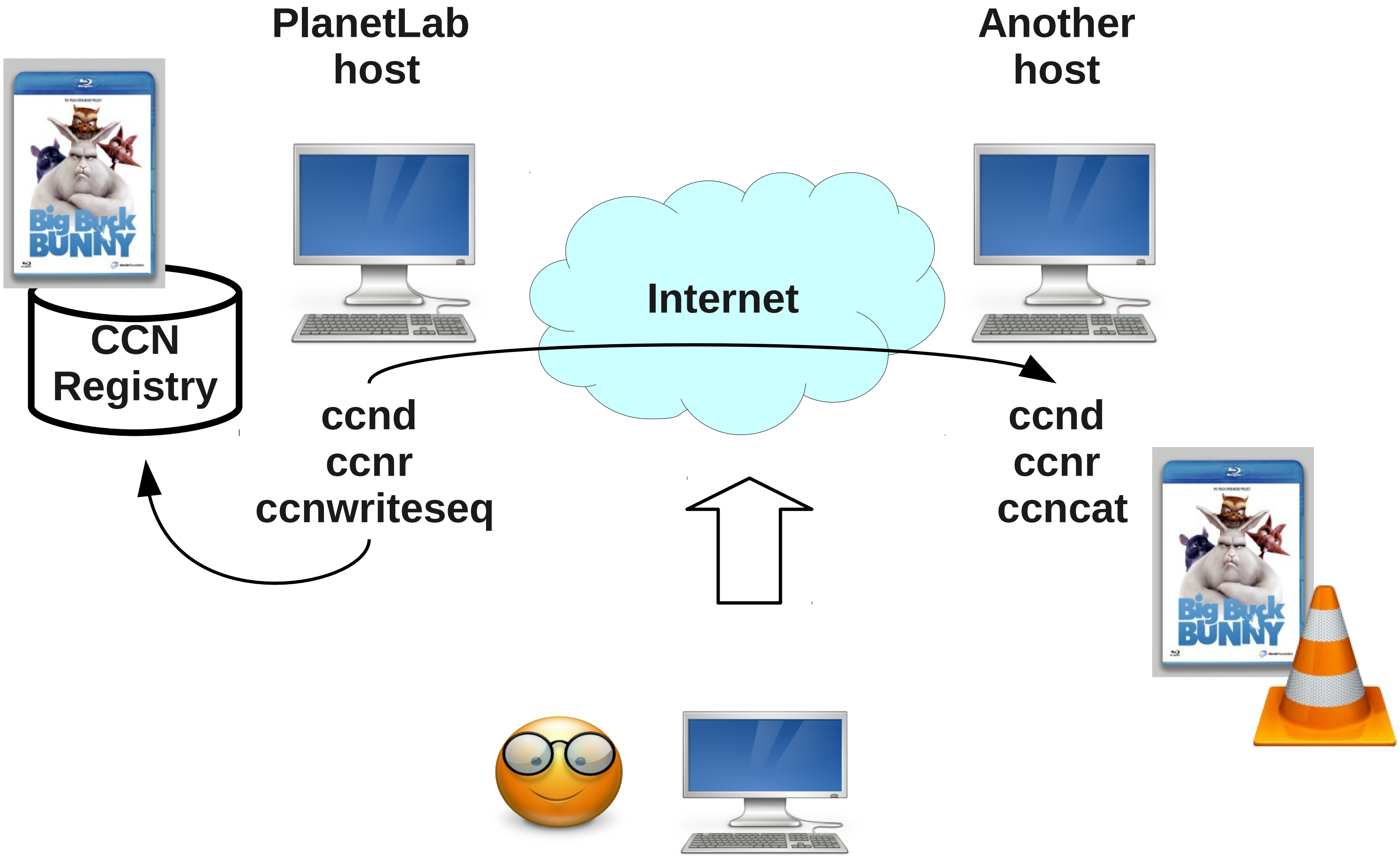
- Even if a specific technology or tool is mastered, other problems exist:
  - Time consumed on experiment implementation
  - Synchronization of resource deployment
  - Error handling and detection during execution
  - Results gathering
- Automation of these aspects can alleviate the problem



The challenge

How to solve the issues related to experiment execution ?

Simple CCNx scenario



PlanetLab host

Another host

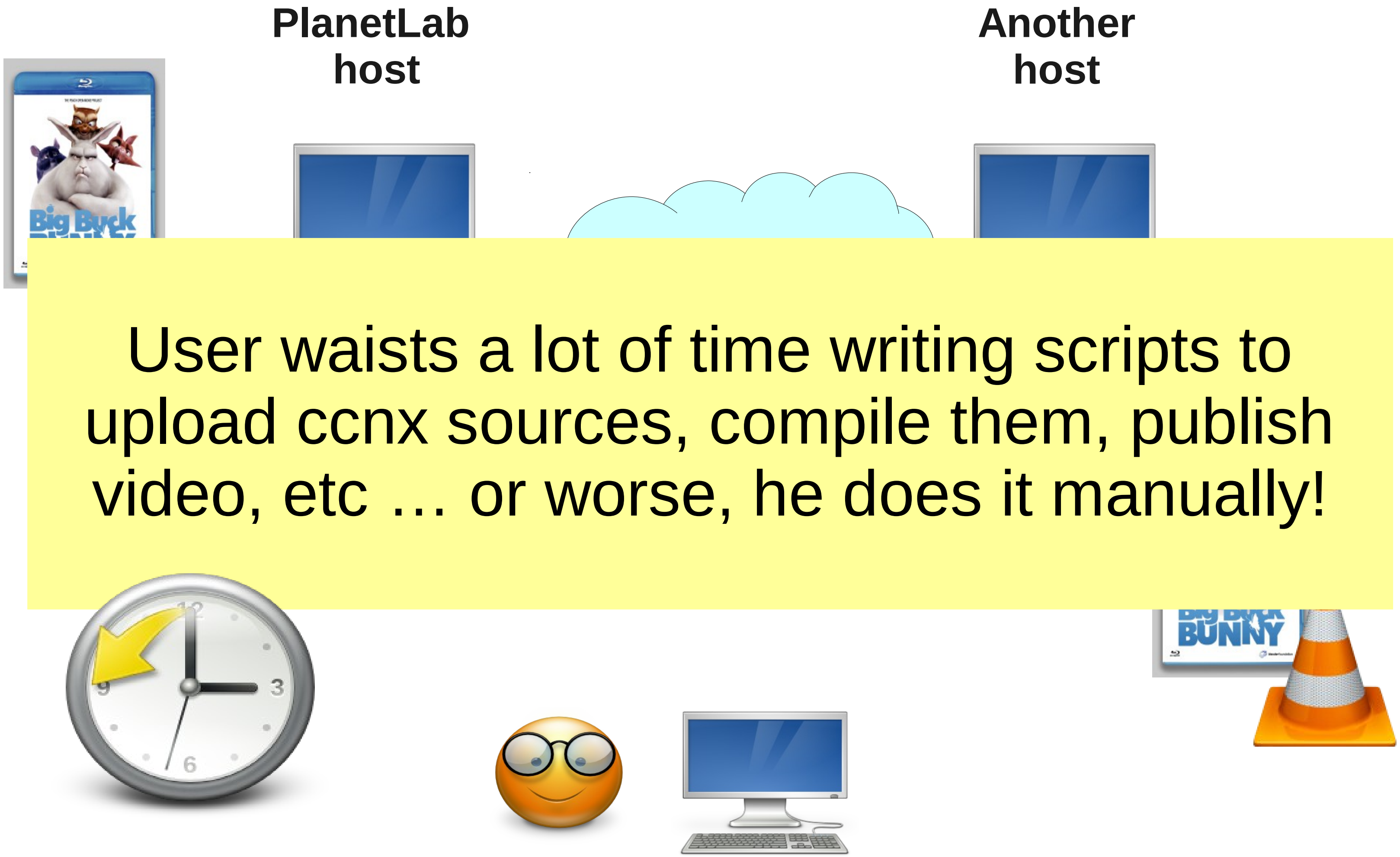
CCN Registry

Internet

ccnd
ccnr
ccnwriteseq

ccnd
ccnr
ccncat

Simple CCNx scenario

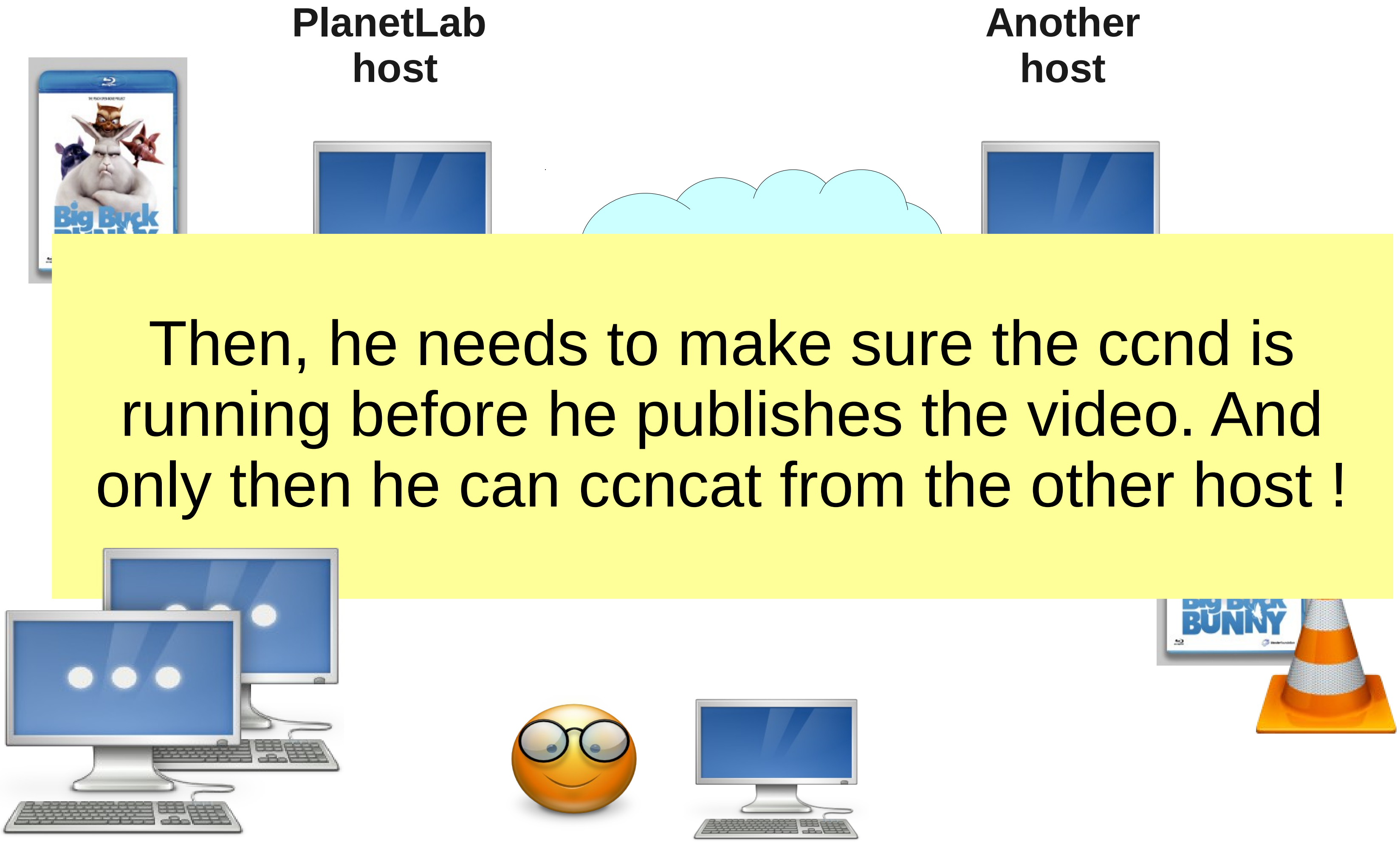


PlanetLab host

Another host

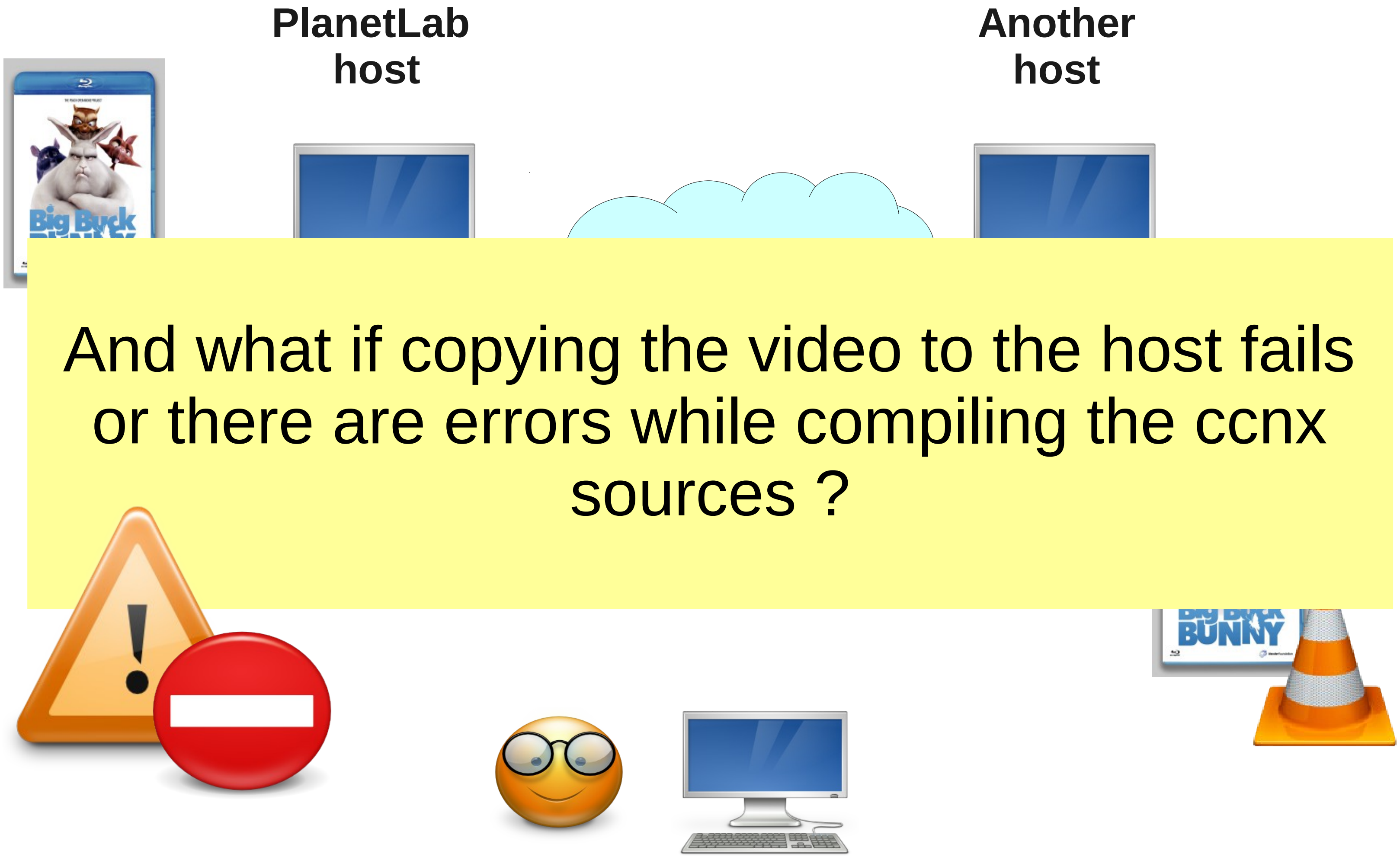
User waists a lot of time writing scripts to upload ccnx sources, compile them, publish video, etc … or worse, he does it manually!

Simple CCNx scenario



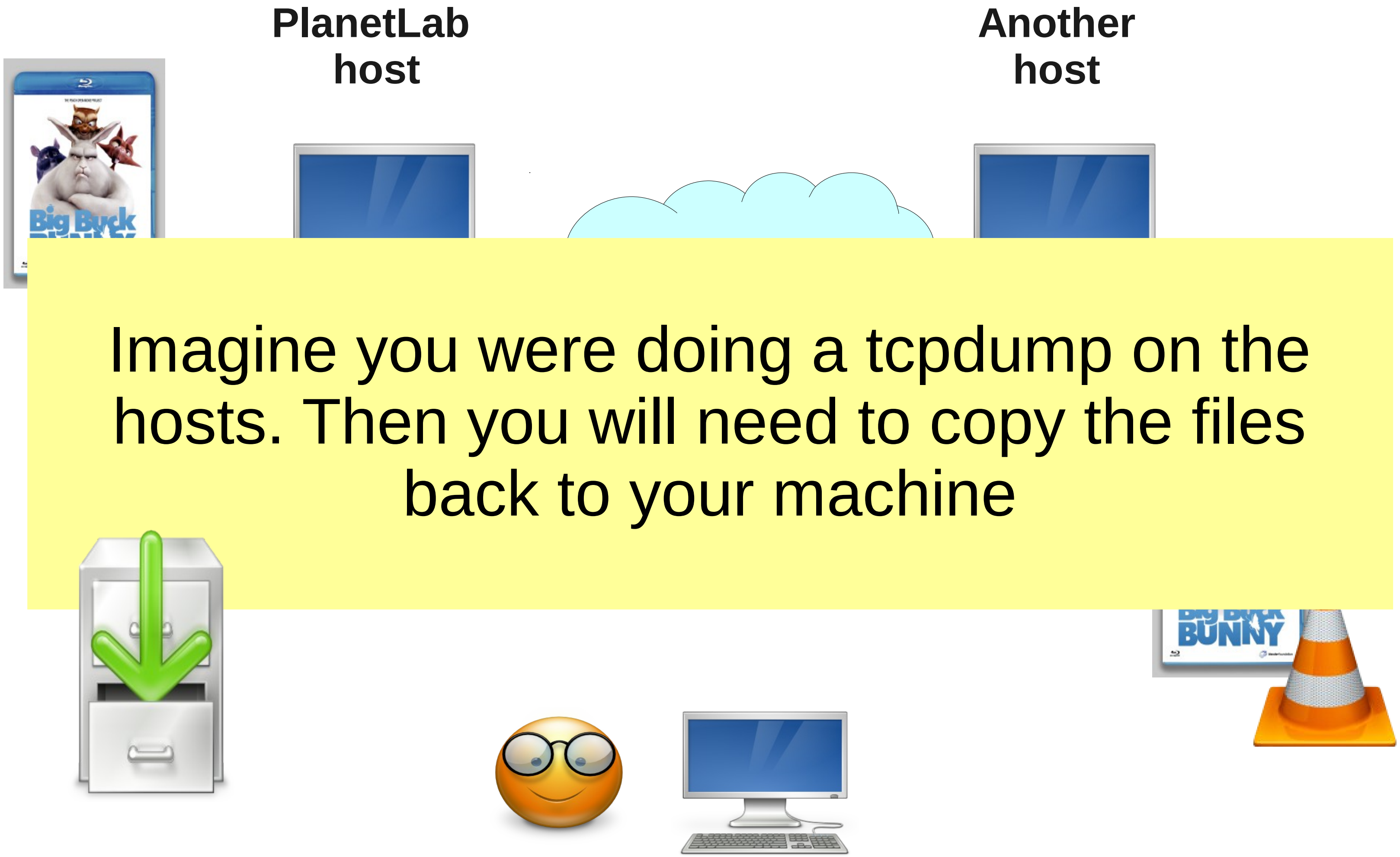
Then, he needs to make sure the ccnd is running before he publishes the video. And only then he can ccncat from the other host !

Simple CCNx scenario



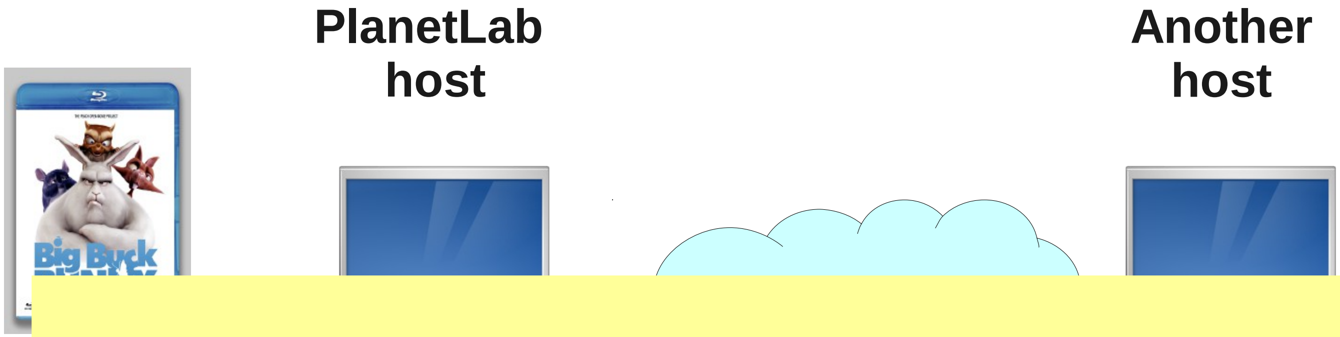
And what if copying the video to the host fails or there are errors while compiling the ccnx sources ?

Simple CCNx scenario

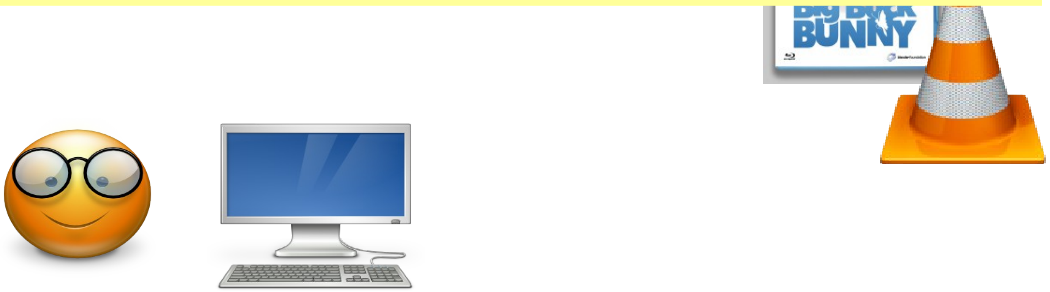


Imagine you were doing a tcpdump on the hosts. Then you will need to copy the files back to your machine

Simple CCNx scenario



Ok, this might not seem like a big problem. But now imagine that instead 2 hosts there are 50 or 100?

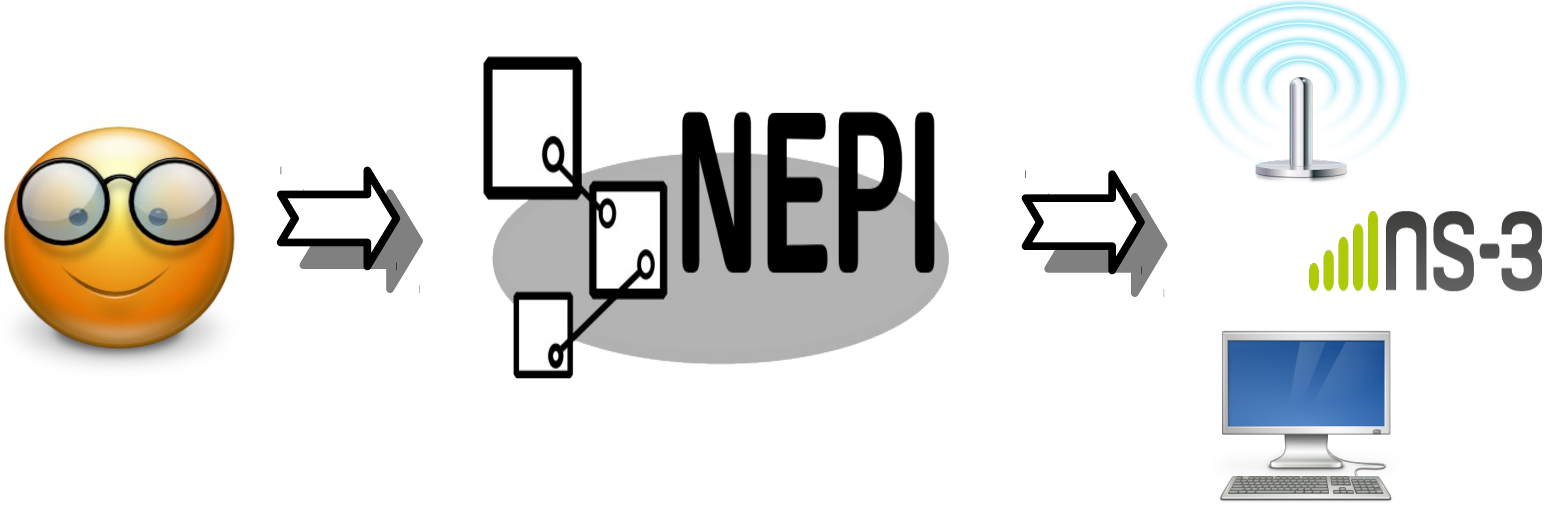


The challenge

How can we make it really simple to run such scenarios ?

NEPI: Network Experiment Programming Interface

- NEPI is a framework to manage network experiments
  - Provides a uniform interface to interact with resources from different testbeds
  - Automates execution of network experiments



Access to testbed resources

- Different testbeds can provide a difference technologies to access/manage resources

Access to testbed resources

- Different testbeds can provide a difference technologies to access/manage resources
- PlanetLab nodes can be managed using SSH

SSH

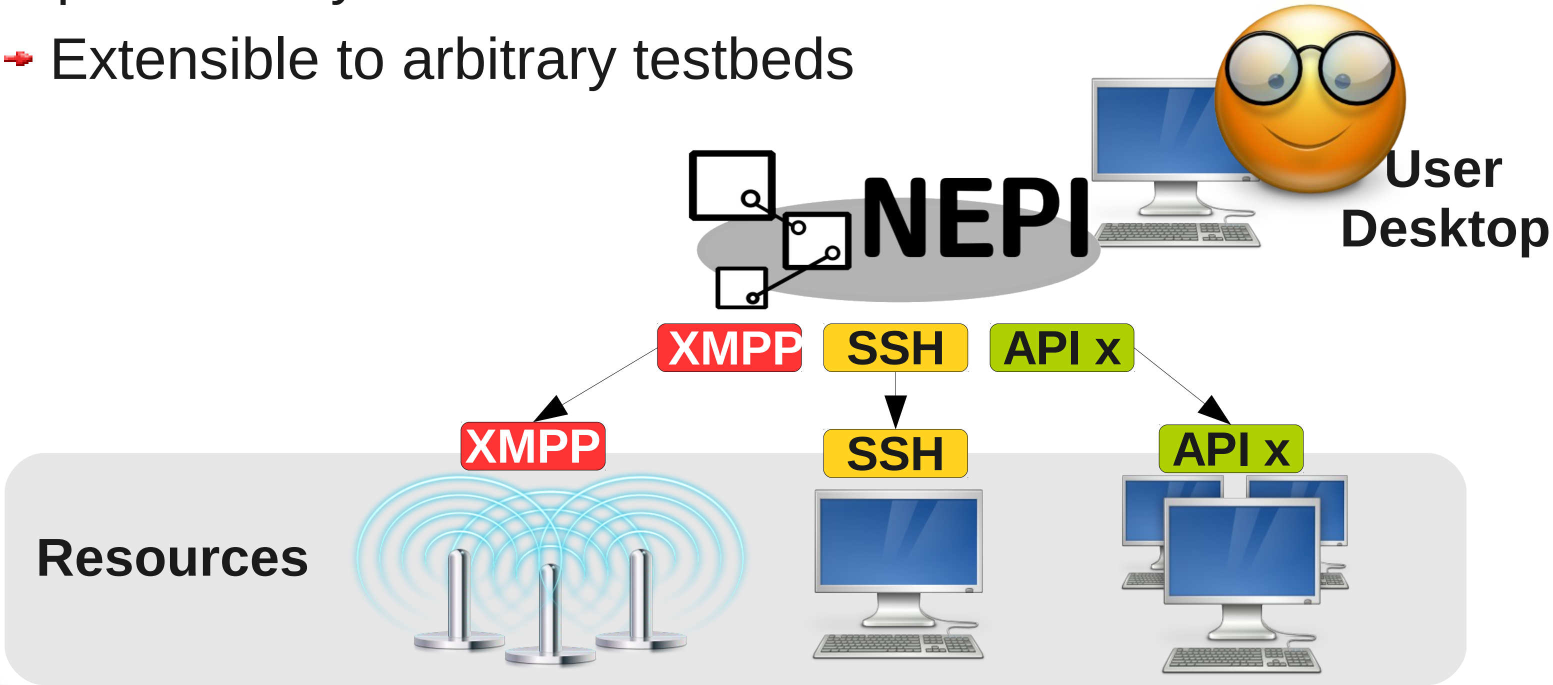
Access to testbed resources

- Different testbeds can provide a difference technologies to access/manage resources
- PlanetLab nodes can be managed using SSH
- OMF (wireless) nodes can be managed using pub/sub XMPP service

XMPP

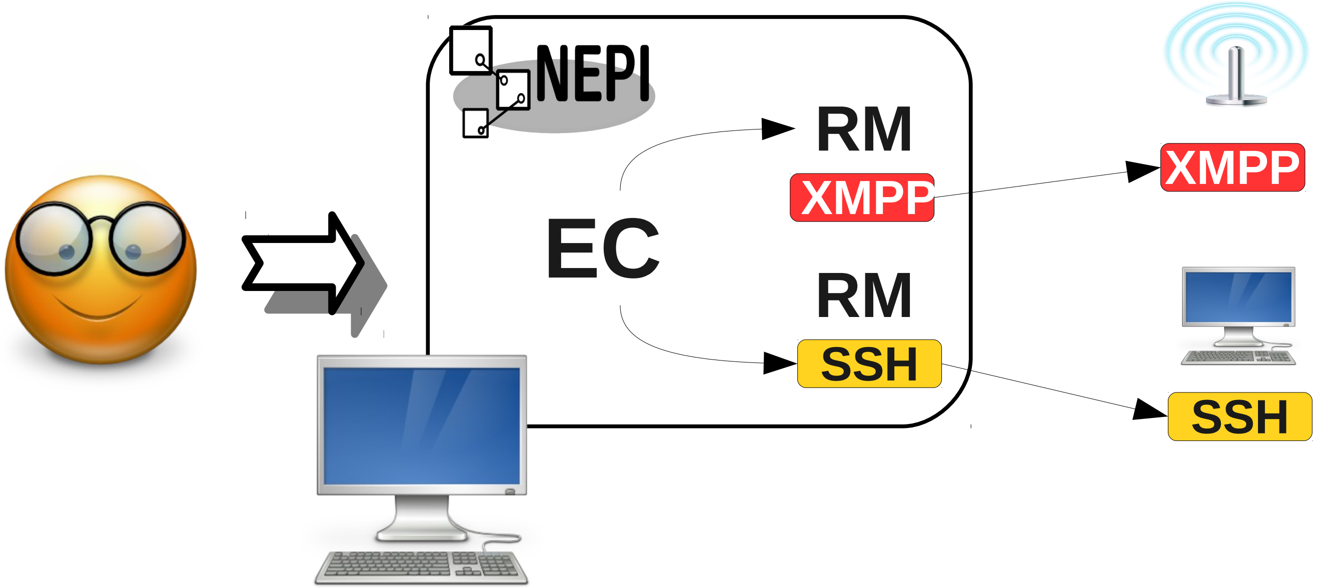
Access to resources in NEPI

- NEPI runs as a client in the user side (e.g. user desktop)
- NEPI does not need to run specific services on resources
  - Not intrusive (no need to modify the testbed)
- NEPI can adapt to different communication mechanisms provided by the testbeds
  - Extensible to arbitrary testbeds



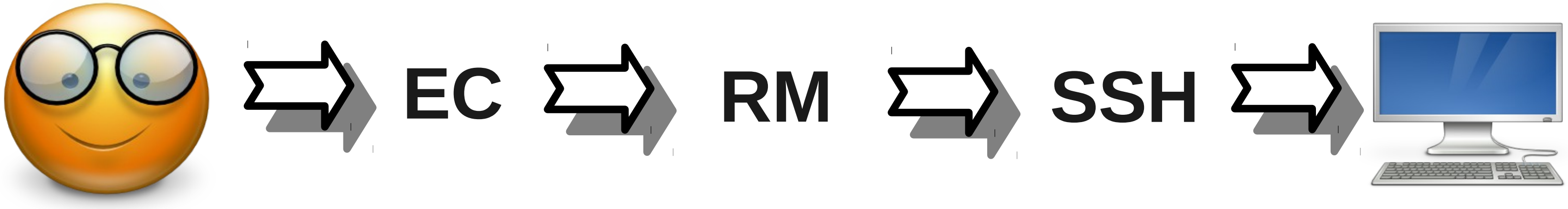
Resource management

- The Experiment Controller (EC) is the entity in NEPI responsible for orchestrating the experiment
- The Resource Managers (RMs) are responsible for managing individual resources



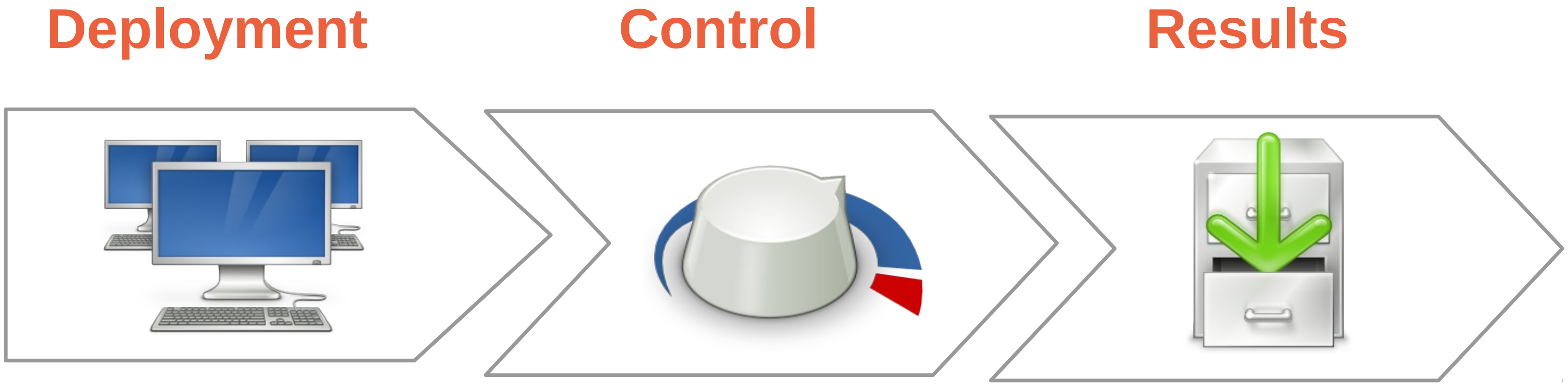
Resource management II

- The EC doesn't 'know' about specific ways of communicating with resources
- The RMs are the ones that 'know' how to configure a node, start or stop applications etc
- All RMs implement a same interface to control resources (e.g. deploy, start, stop, etc)



Resource control interface

- The RM interface reflects the resource life cycle



Deployment

Control

Results

- Resource discovery & provision
- Resource configuration
- Software installation
- Resource synchronization
- Instrumentation
- Resource start

- Configuration changes
- Status monitoring
- Error detection/handling
- Resource release

- Result information
- Result download

Interesting features

Task scheduling

- Conditions can be specified to start/stop resources or to change configuration
- A scheduler is used to execute tasks in the right order, taking conditions into account

Task scheduling

- Conditions can be specified to start/stop resources or to change configuration
- A scheduler is used to execute tasks in the right order, taking conditions into account

- Two types of conditions:
  - Structural → Defined by the developer
    - *Node needs to be ready before application can run*
  - Behavioral → Specified by the user
    - *App X must start after app Y*

Task scheduling

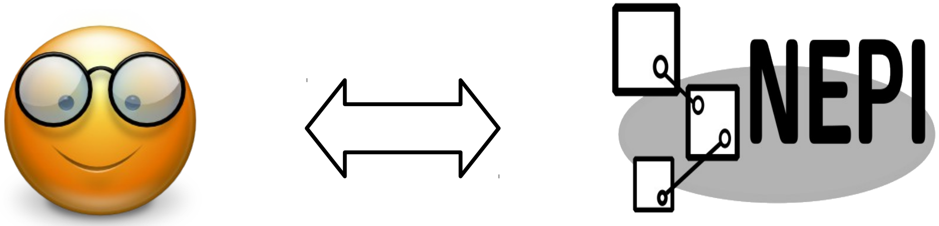
- Conditions can be specified to start/stop resources or to change configuration
- A scheduler is used to execute tasks in the right order, taking conditions into account

- Two types of conditions:
  - Structural → Defined by the developer
    - *Node needs to be ready before application can run*
  - Behavioral → Specified by the user
    - *App X must start after app Y*

- Conditions are state and time based
  - *Start app X after app Y has started*
  - *Start app X 5 seconds after app Y stopped*

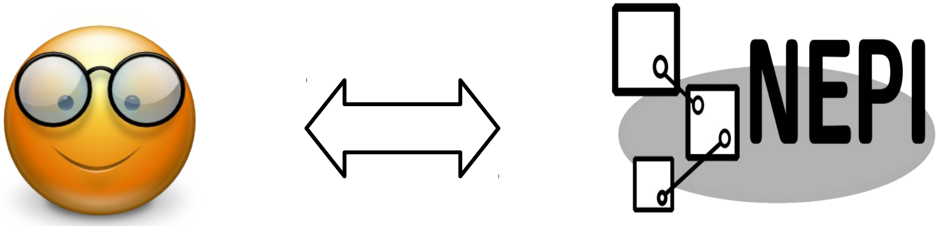
Interactive experimentation

- No need to specify the complete experiment beforehand
- At any moment it is possible to:
  - Deploy new resources
  - Change configuration
  - Retrieve or query results
  - Query configuration and state
  - Start/Stop resources



Interactive experimentation
What for ?

- Initial test and exploration of technologies
- Education
  - What happens if we change bw on link X ?
- Dynamic experiments
  - Elastic cloud provisioning
  - Dynamically controller routing
  - ...

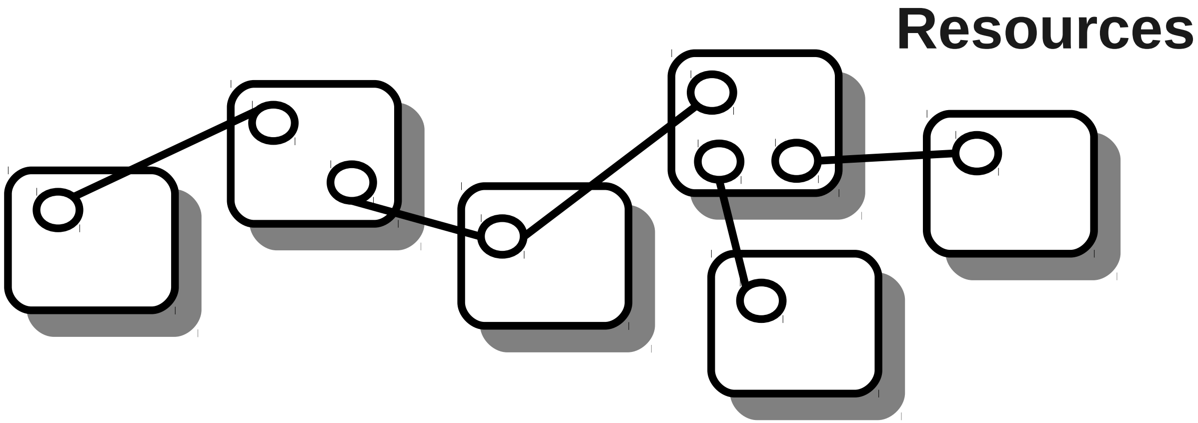


Experiment representation

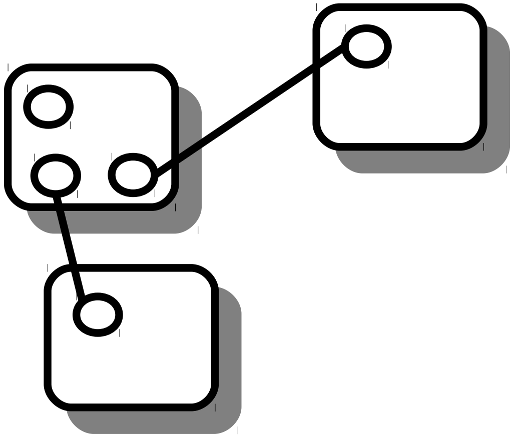
Experiment representation

Experiment representation

- Experiments are represented as graphs of resources
  - Interconnected resource managers
- Any element that can used to describe an experiment is a resource



Resource properties



Connections

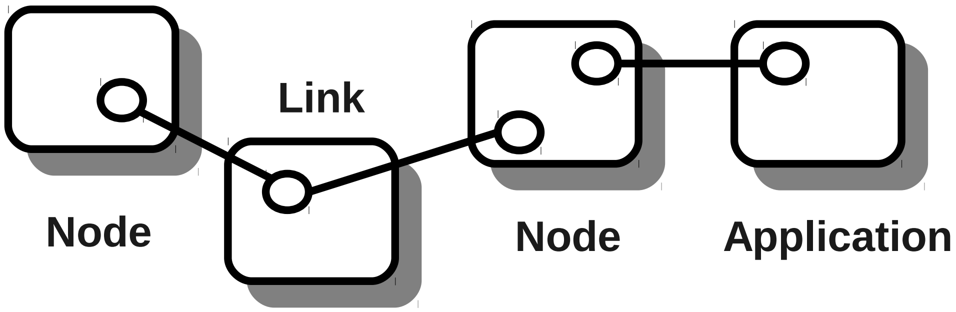
Attributes

States

Traces

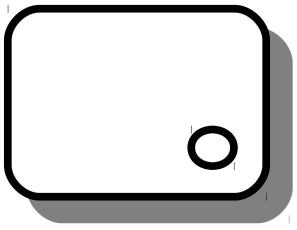
Resource connections

- Represent resources interacting during the experiment
- The meaning of a specific connection depends on the type of objects associated and is implicit



Resource attributes

- Resources are associated to a list of attributes
- Attributes expose the resource configuration
- Attributes are defined by {name, value, type}



LinuxNode

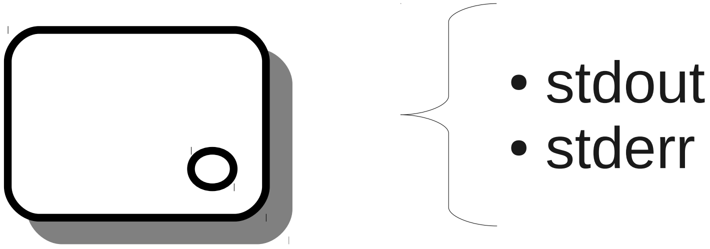
Hostname: nepi1.pl.sophia.inria.fr – String

SSH port: 22 - Integer

CleanHome: False – Bool

Resource traces

- Resources are associated to a list of traces
- A trace defines data to be collected into a file during experiment execution
- This data can be obtained from measurements or application output (e.g. stderr, tcpdump)
- Different boxes expose different traces



- stdout
- stderr

LinuxApplication

Resource states

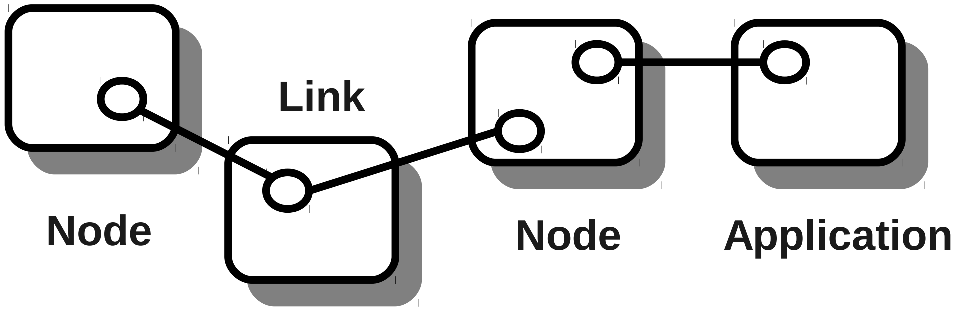
- All RM are expected to transition through the same states
  - NEW – Resource is not deployed
  - DISCOVERED - Resource availability information was retrieved
  - PROVISIONED - The resource is accessible to the user
  - READY – Resource is configured or ready to start
  - STARTED – Resource is taking part of the experiment
  - STOPPED – The user interrupted the resource
  - FINISHED – The resource finished taking part of the exp.
  - FAILED – The resource failed
  - RELEASED – Resource is no longer accessible by the RM

Resource states

- All RM are expected to transition through the same states
  - NEW – Resource is not deployed
  - DISCOVERED – Resource availability information was retrieved
  - PROVISIONED - The resource is accessible to the user
  - READY – Resource is configured or ready to start
  - STARTED – Resource is taking part of the experiment
  - STOPPED – The user interrupted the resource
  - FINISHED – The resource finished taking part of the exp.
  - FAILED – The resource failed
  - RELEASED – Resource is no longer accessible by the RM
- Same states might have different semantical meanings depending on the type of resource

Benefits of this representation

- Really simple way of modeling experiments by connecting resource managers and setting attributes, traces, etc
- Basic resource managers are provided by NEPI
- New ones can be added if by users if needed



Experiment execution

The NEPI project

- NEPI is written in Python
  - Fast scripting
- NEPI is licensed under GPLv2
  - Everybody can use it
  - Everybody can extend it

python™

GPLv2

The experiment script

- Import NEPI modules

```
from nepi.execution.ec import ExperimentController
from nepi.execution.resource populate_factory,
                       ResourceState, ResourceAction
```

- Instantiate the ExperimentController

```
ec = ExperimentController (exp_id = “my-exp”)
```

The experiment script - Resources

- Create resources

```
node = ec.register_resource ("LinuxNode")
app = ec.register_resource ("LinuxApplication")
```

- Configure resources

```
ec.set (node, "hostname", “node1.pl.sophia.inria.fr”)
ec.set (node, "username", “inria_nepi”)
```

- Connect resources

```
ec.register_connection (node, app)
```

The experiment script - Deployment

- Register condition (e.g. start app1 5s after app2)

```
ec.register_condition (app1, ResourceAction.START,
             app2, ResourceState.STARTED, time = “5s”)
```

- Deploy resources

```
ec.deploy ()
```

- Deploy a group of resources

```
my-group = [ node, app1, app2 ]
ec.deploy (group = my-group,  wait_all_ready = True)
```

The experiment script - Results

- Enable trace

```
ec.register_trace (app, “stdout”)
```

- Retrieve trace path

```
path = ec.trace (app, “stdout”, attr = TraceAttr.PATH )
```

- Retrieve trace

```
stdout = ec.trace (app, “stdout”)
```

- Retrieve stream

```
path = ec.trace (app, “stdout”,
                attr = TraceAttr.STREAM,  block, offset )
```

The experiment script - Control

- Wait until finished

```
apps = [ app1, app2 ]
ec.wait_finished (apps)
```

- Query state

```
state = ec.state (app)
```

- Query configuration

```
host = ec.get (app, “hostname”)
```

The experiment script – Termination

- Stop one resource

```
ec.stop (app1)
```

- Stop all resources

```
ec.release ()
```

- Shutdown EC (stop prcessing events)

```
ec.shutdown ()
```

How to run the experiment

- To run the experiment …

```
python my-experiment.py
```

Logging

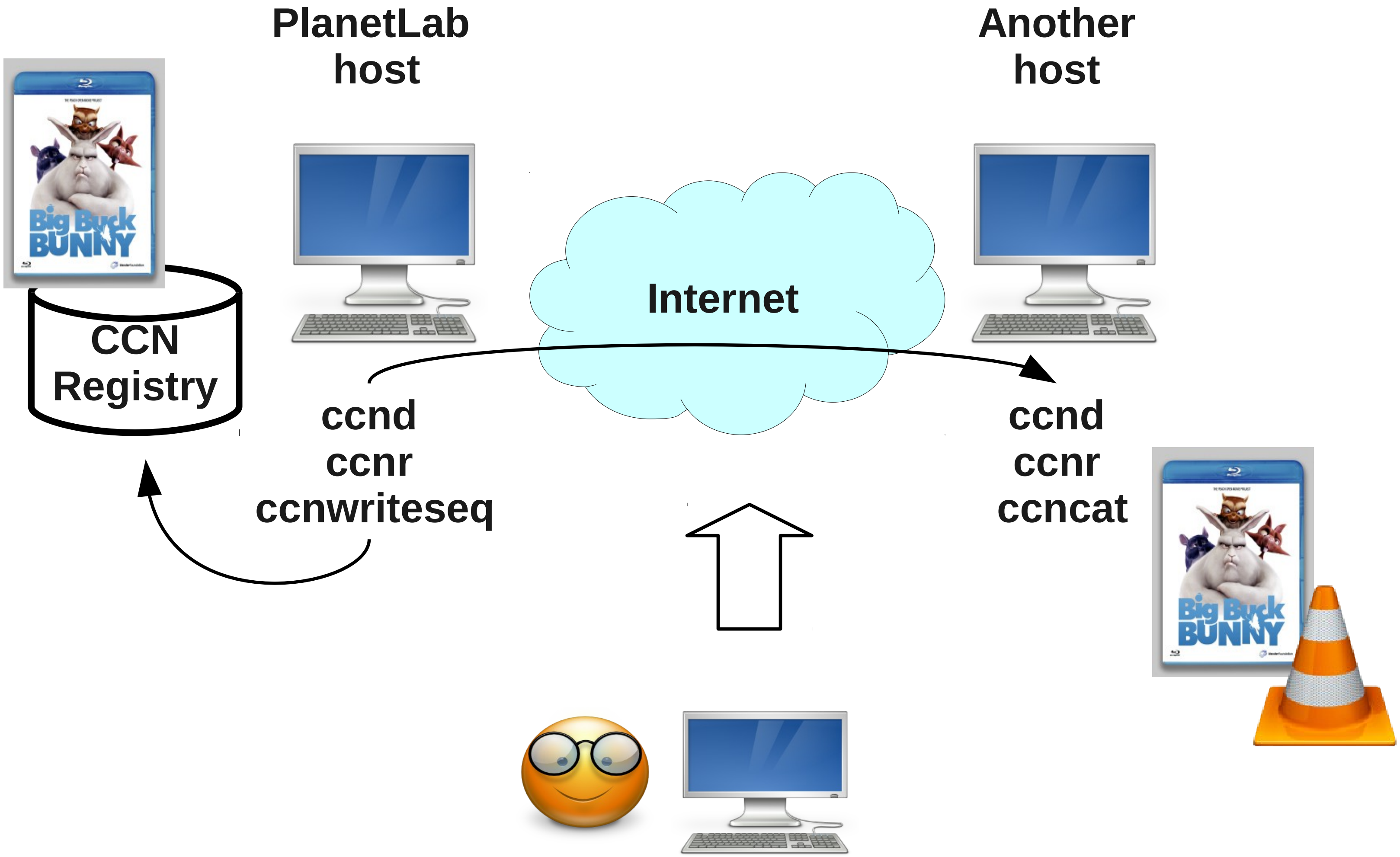
- Logging level can be controlled with the "NEPI_LOGLEVEL" environment variable

```
NEPI_LOG_LEVEL=DEBUG python my-experiment.py
```

```
2013-05-13 15:34:14,798 LinuxNode INFO  guid 4 - host roseval.pl.sophia.inria.fr -
Cleaning up processes
2013-05-13 15:34:16,513 LinuxNode INFO  guid 1 - host planetlab2.u-strasbg.fr -
Cleaning up processes
2013-05-13 15:34:22,118 LinuxApplication INFO  guid 3 - host planetlab2.u-strasbg.fr
-  Deploying command
2013-05-13 15:34:22,124 LinuxApplication INFO  guid 2 - host planetlab2.u-strasbg.fr
-  Deploying command
2013-05-13 15:34:24,176 LinuxApplication INFO  guid 3 - host planetlab2.u-strasbg.fr
-  Uploading stdin
2013-05-13 15:34:25,376 LinuxApplication INFO  guid 2 - host planetlab2.u-strasbg.fr
-  Uploading sources
```

CCNx experiment example

Simple CCNx scenario



PlanetLab host

Another host

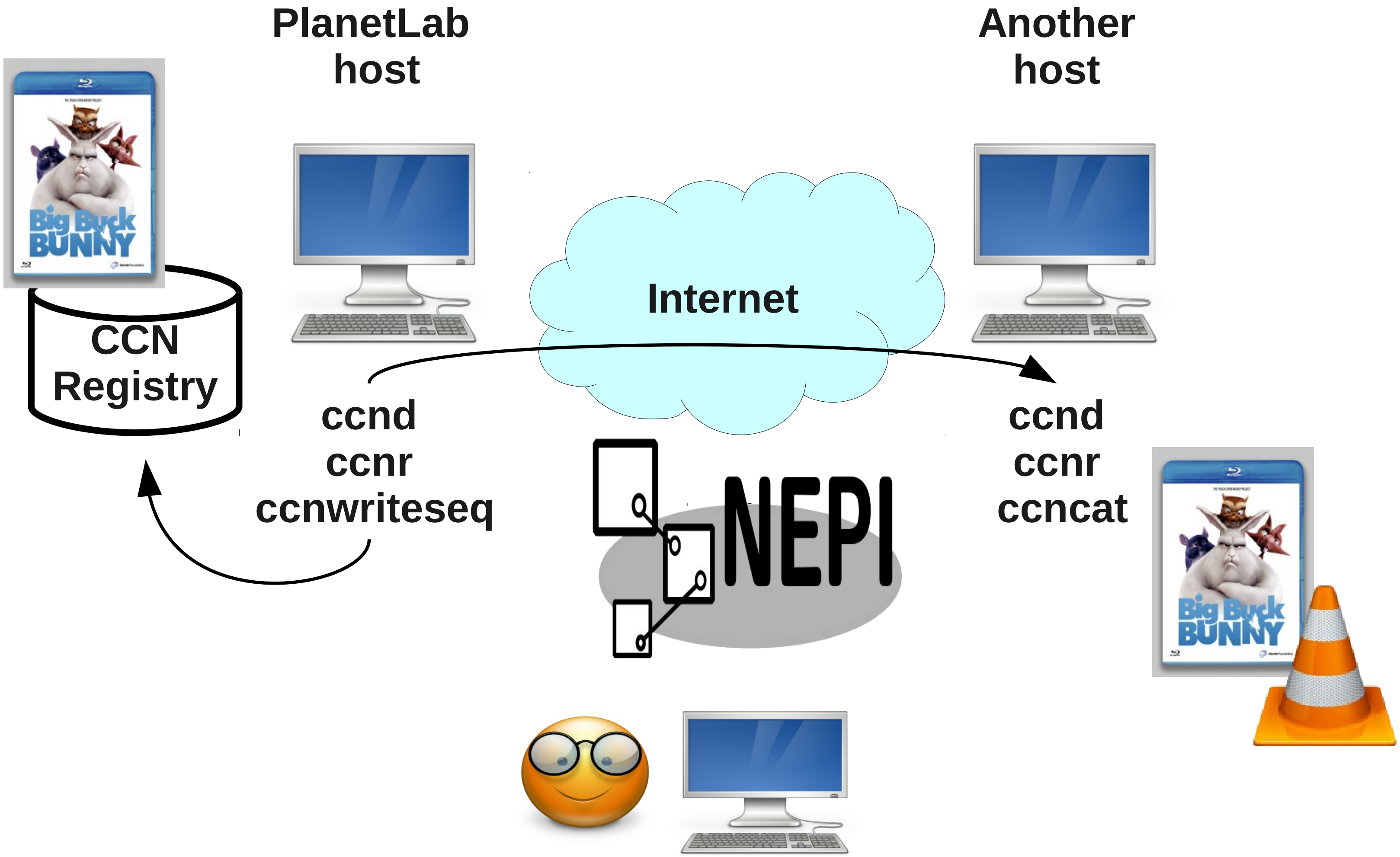
CCN Registry

Internet

ccnd
ccnr
ccnwriteseq

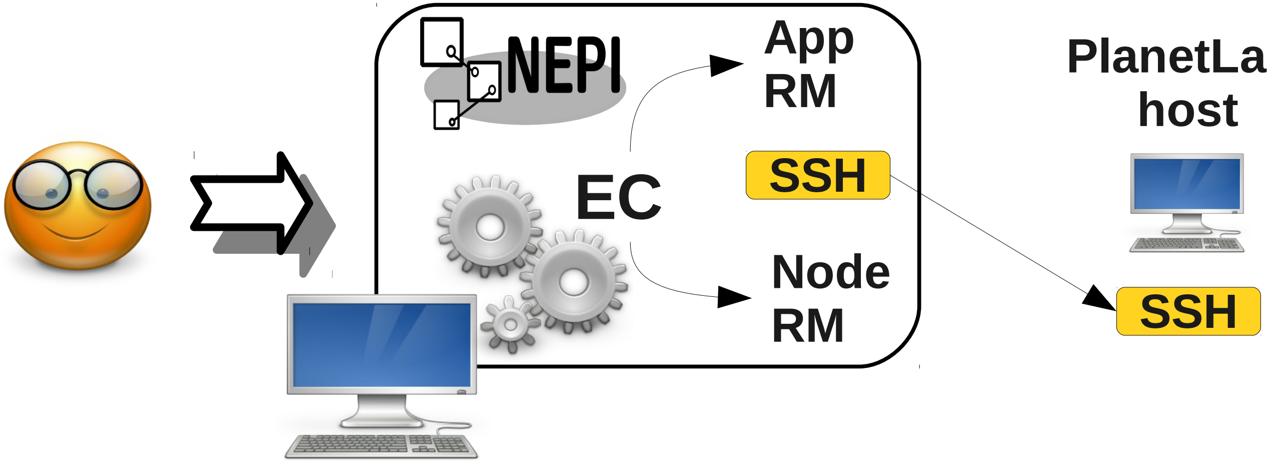
ccnd
ccnr
ccncat

Simple CCNx scenario



NEPI & PlanetLab

- NEPI provides LinuxNode & LinuxApplication RMs
- They use SSH to to deploy and control resources and collect results
- Each RM creates remote experiment specific directories to upload sources ans store results



The CCNx experiment script

```
ec = ExperimentController (exp_id = “my-exp”)
```

The CCNx experiment script

```
node = ec.register_resource ("LinuxNode")
ec.set (node, "hostname", “myplnode.inria.fr”)
ec.set (node, "username", “slicename”)
ec.set (node, "cleanHome", True)
ec.set (node, "cleanProcesses", True)
```

The CCNx experiment script

```
app = ec.register_resource("LinuxApplication")
```

The CCNx experiment script

```
app = ec.register_resource("LinuxApplication")

ec.set(app, "depends", “gcc make”)
```

The CCNx experiment script

```
 app = ec.register_resource("LinuxApplication")

 ec.set(app, "depends", “gcc make”)

 sources = "http://www.ccnx.org/releases/ccnx-0.7.1.tar.gz"

ec.set(app, "sources", sources)
```

The CCNx experiment script

```
 app = ec.register_resource("LinuxApplication")

 ec.set(app, "depends", “gcc make”)

 sources = "http://www.ccnx.org/releases/ccnx-0.7.1.tar.gz"

ec.set(app, "sources", sources)

 build = "  tar xf ${SOURCES}/ccnx-0.7.1.tar.gz ;
       cd ${SOURCES}/ccnx-0.7.1 ;
       ./configure && make ; "

 ec.set(app, "build", build)
```

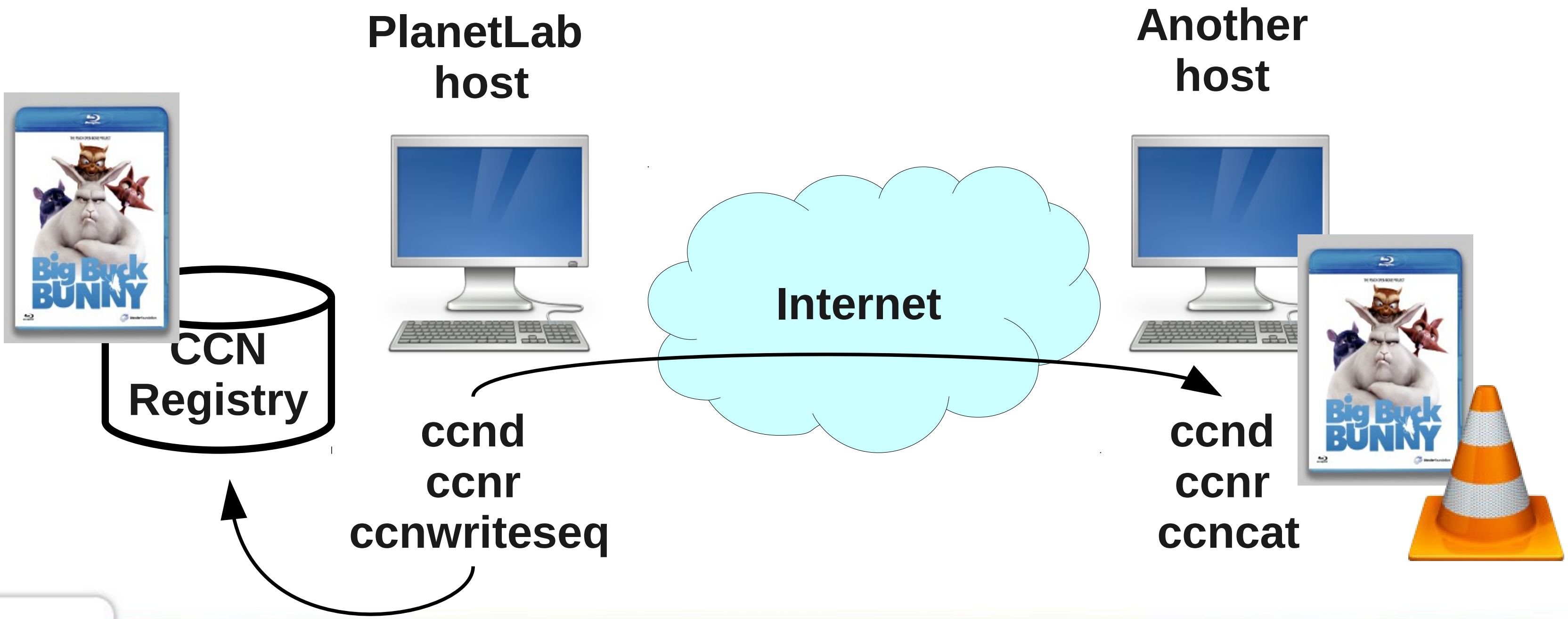
The CCNx experiment script

```
command = " ccndstart ;
     ccndc add ccnx:/ udp  host2.inria.fr ;
     ccnr  "

ec.set(app, "command", command)
```

The CCNx experiment script

- We can easily register more LinuxNodes
- We can easily register other LinuxApplications following the same steps



Lets see what happens...

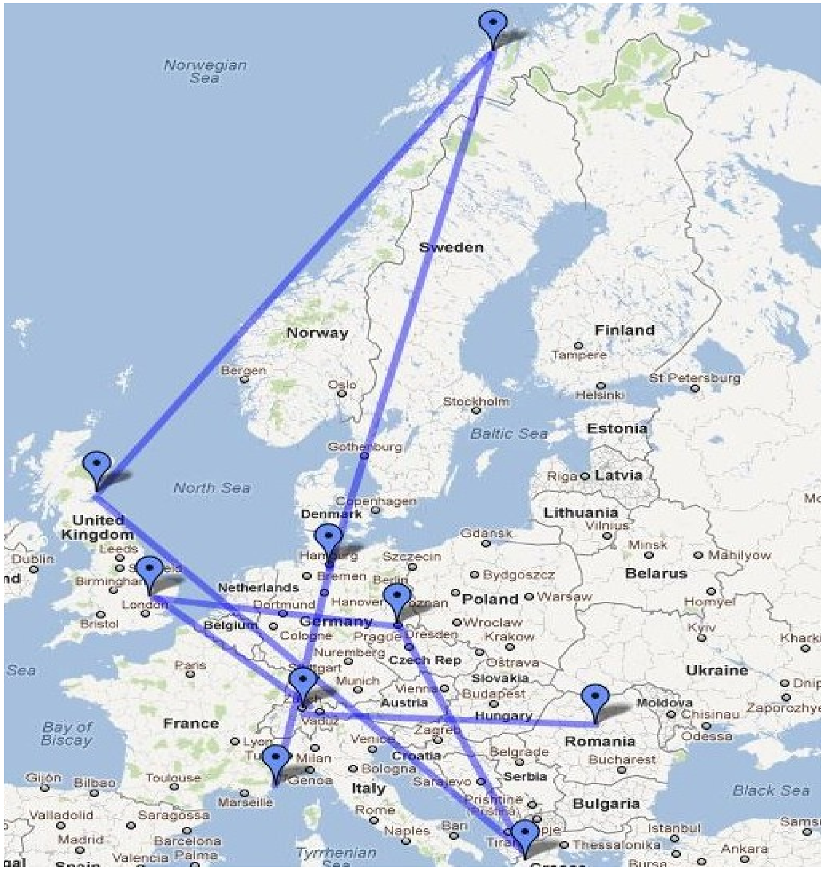
THE PEACH OPEN MOVIE PROJECT PRESENTS
Big Buck BUNNY
WRITTEN AND DIRECTED BY SACHA GOEDEGEBURE – ART DIRECTOR ANDREAS GORALCZYK – LEAD ARTIST ENRICO VALENZA
ANIMATORS NATHAN VEGDAHL, WILLIAM REYNISH – TECHNICAL DIRECTORS CAMPBELL BARTON, BRECHT VAN LOMMEL
MUSIC BY JAN MORGENSTERN – PRODUCED BY TON ROOSENDAAL, BLENDER FOUNDATION
LICENSED AS CREATIVE COMMONS 3.0 ATTRIBUTION
WWW.BIGBUCKBUNNY.ORG
SPONSORS:
creative commons nederland
wavemage
Maqina
blenderinstitute

CCNx extended example

Extended example – The PL nodes

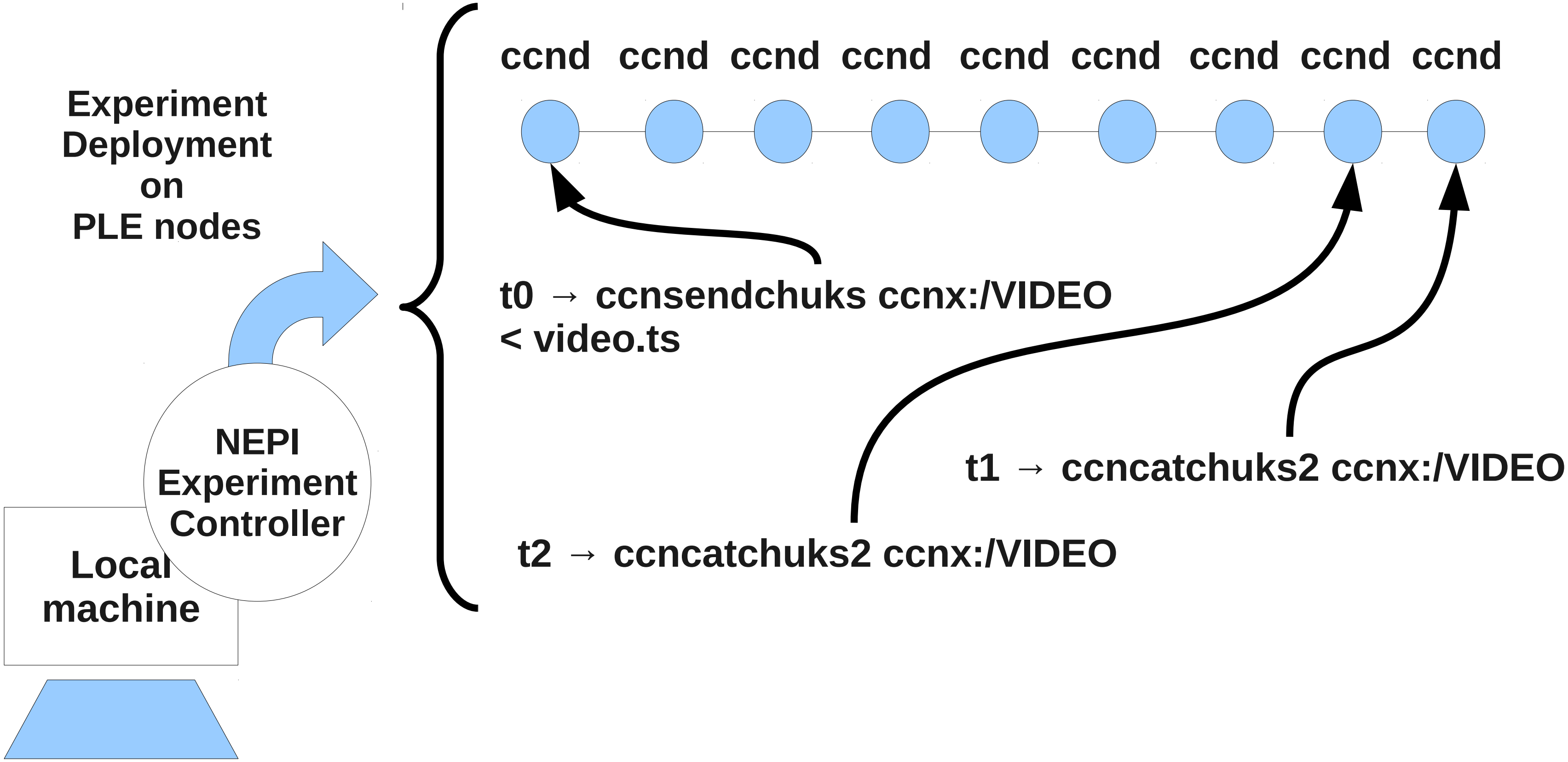
- 9 PLE nodes in 8 different countries

1. openlab02.pl.sophia.inria.fr
2. merkur.planetlab.haw-hamburg.de
3. planetlab1.cs.uit.no
4. planetlab3.cs.st-andrews.ac.uk
5. planetlab2.cs.uoi.gr
6. planet2.inf.tu-dresden.de
7. planetlab3.xeno.cl.cam.ac.uk
8. planetlab2.csg.uzh.ch
9. planetlab2.upm.ro



Extended example

- Observe effects of CCNx caching when simultaneously retrieving a video stream along several PlanetLab nodes associated in series through UDP unicast FIB entries



Experiment Deployment on PLE nodes
ccnd ccnd ccnd ccnd ccnd ccnd ccnd ccnd ccnd
t0 → ccnsendchuks ccnx:/VIDEO < video.ts
t1 → ccncatchuks2 ccnx:/VIDEO
t2 → ccncatchuks2 ccnx:/VIDEO
NEPI Experiment Controller
Local machine

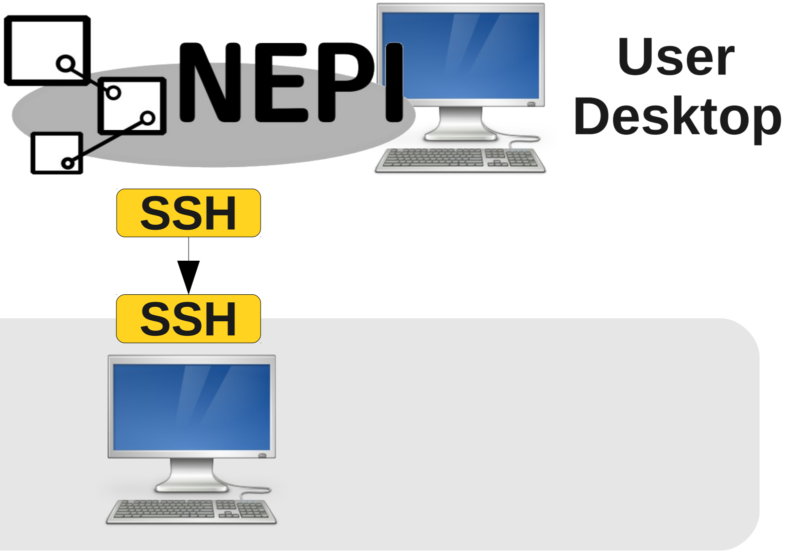
Lets see what happens now ...

THE PEACH OPEN MOVIE PROJECT PRESENTS
Big Buck
BUNNY
WRITTEN AND DIRECTED BY SACHA GOEDEGEBURE – ART DIRECTOR ANDREAS GORALCZYK – LEAD ARTIST ENRICO VALENZA
ANIMATORS NATHAN VEGDAHL, WILLIAM REYNISH – TECHNICAL DIRECTORS CAMPBELL BARTON, BRECHT VAN LOMMEL
MUSIC BY JAN MORGENSTERN – PRODUCED BY TON ROOSENDAAL, BLENDER FOUNDATION
LICENSED AS CREATIVE COMMONS 3.0 ATTRIBUTION
WWW.BIGBUCKBUNNY.ORG
SPONSORS
creative commons nederland
digitale pioniers
wavemage
Sun
Maqina
blenderinstitute

NEPI project status

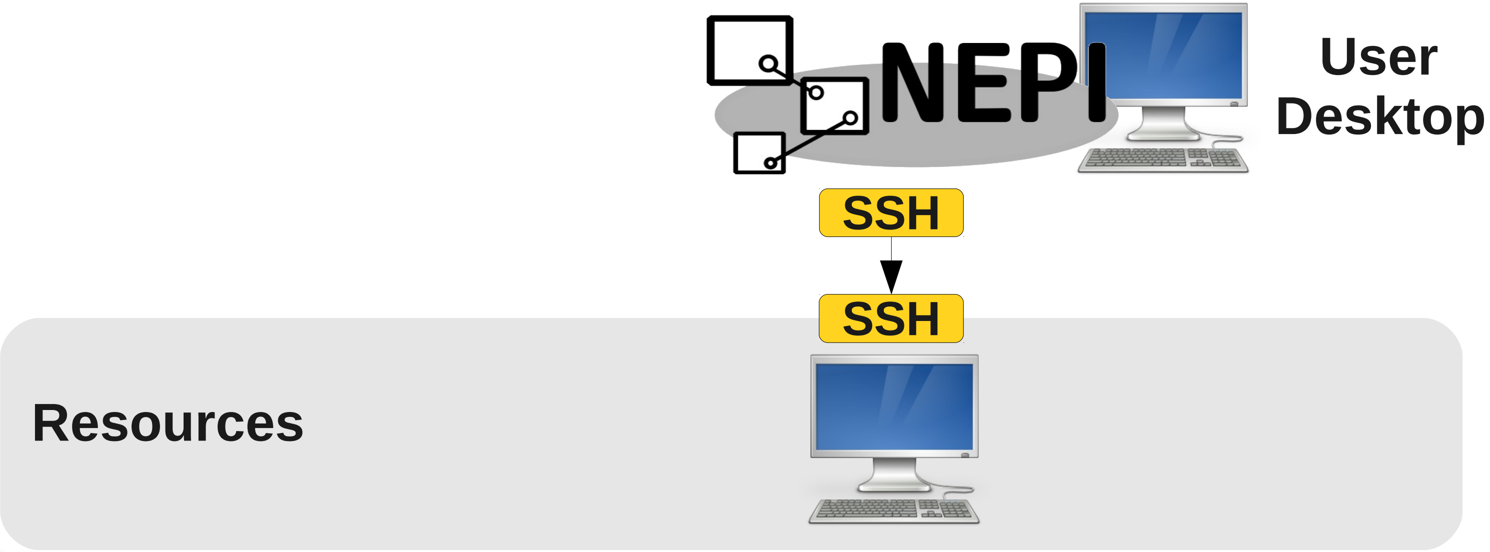
Current status – what is supported ?

- SSH based linux resources
  - PlanetLab
  - Your laptop or desktop
  - Machines in your lab



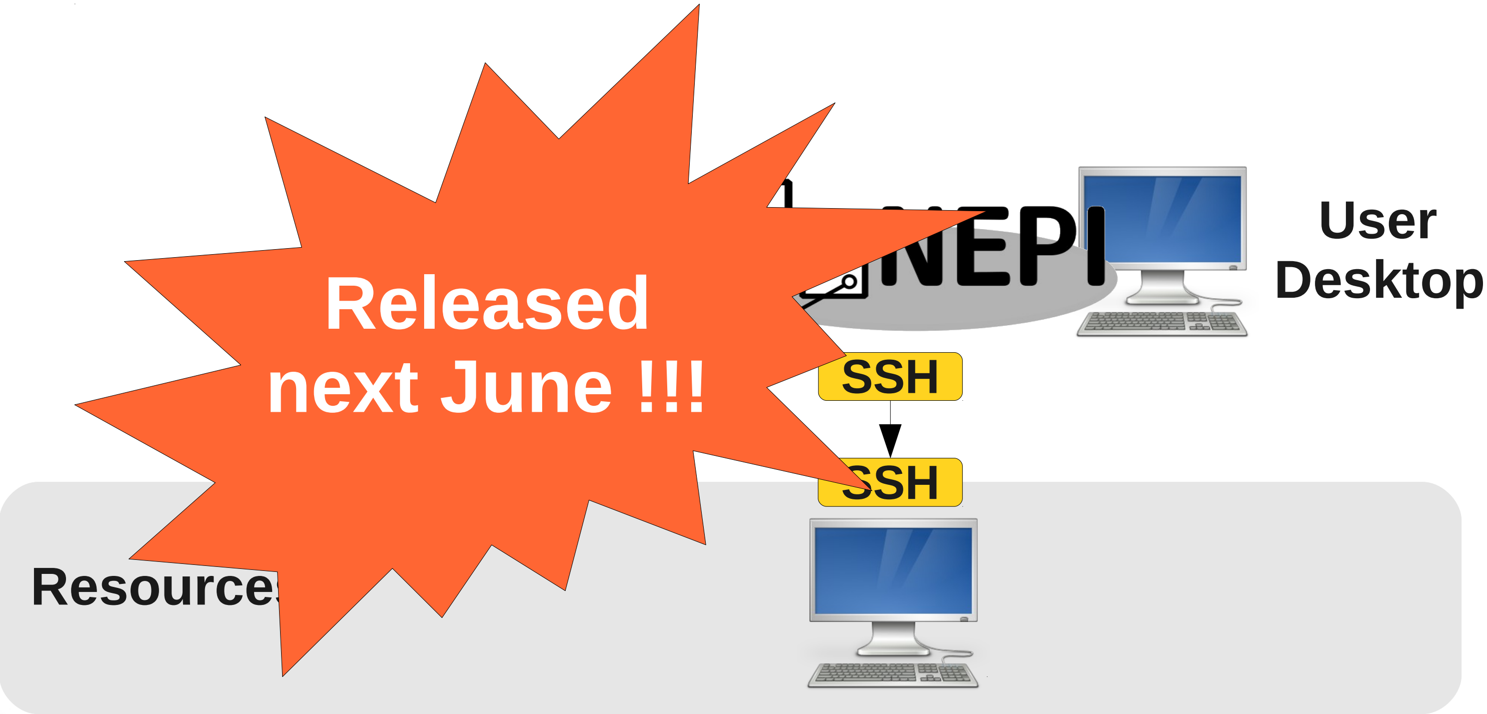
Current status – Release

- Next release end of June (NEPI v3)



Current status – Release

- Next release end of June (NEPI v3)



Current status – Release

- Next release end of June (NEPI v3)

Future releases

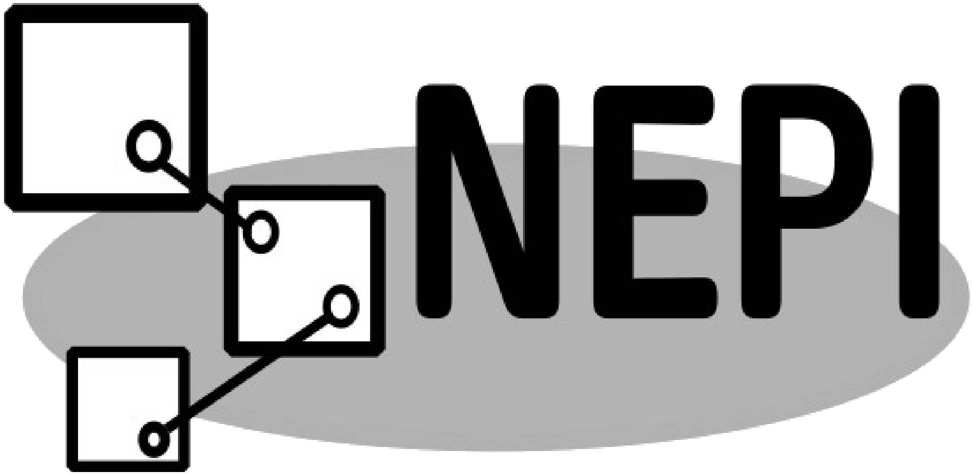
- NEPI is a work in process
- We plan to port all features from NEPI v1
  - Support OMF
  - Support ns-3 simulations
  - Automatic discovery of PlanetLab hosts
  - Graphical interface

Future future releases

- Support new testbeds
  - Grid5000
- Show realtime result information in the GUI
- Add smarter result collection and aggregation

Trying out NEPI

- First release of NEPI v3 is on its way !
- You can check out NEPI web **http://nepi.inria.fr**
- We look for users and contributors
- We hope to give a tool to the community that will make conducting network experiments easier



Thank you

http://nepi.inria.fr

alina.quereilhac@inria.fr

Questions?

http://nepi.inria.fr

alina.quereilhac@inria.fr