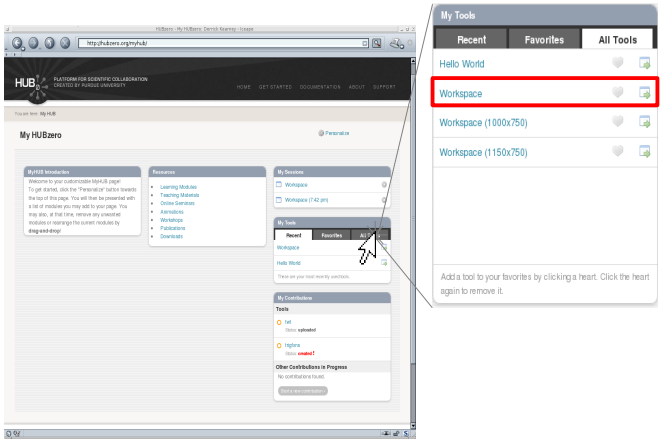
Automating Workspace Testing

Accessing a Workspace

User's Web Browser



HUBzero Infrastructure

Web Server

Middleware

Submit Proxy

Grid

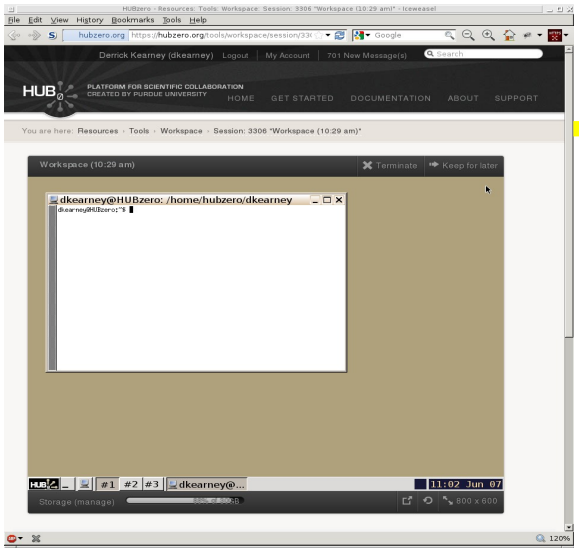
Tool Session Containers

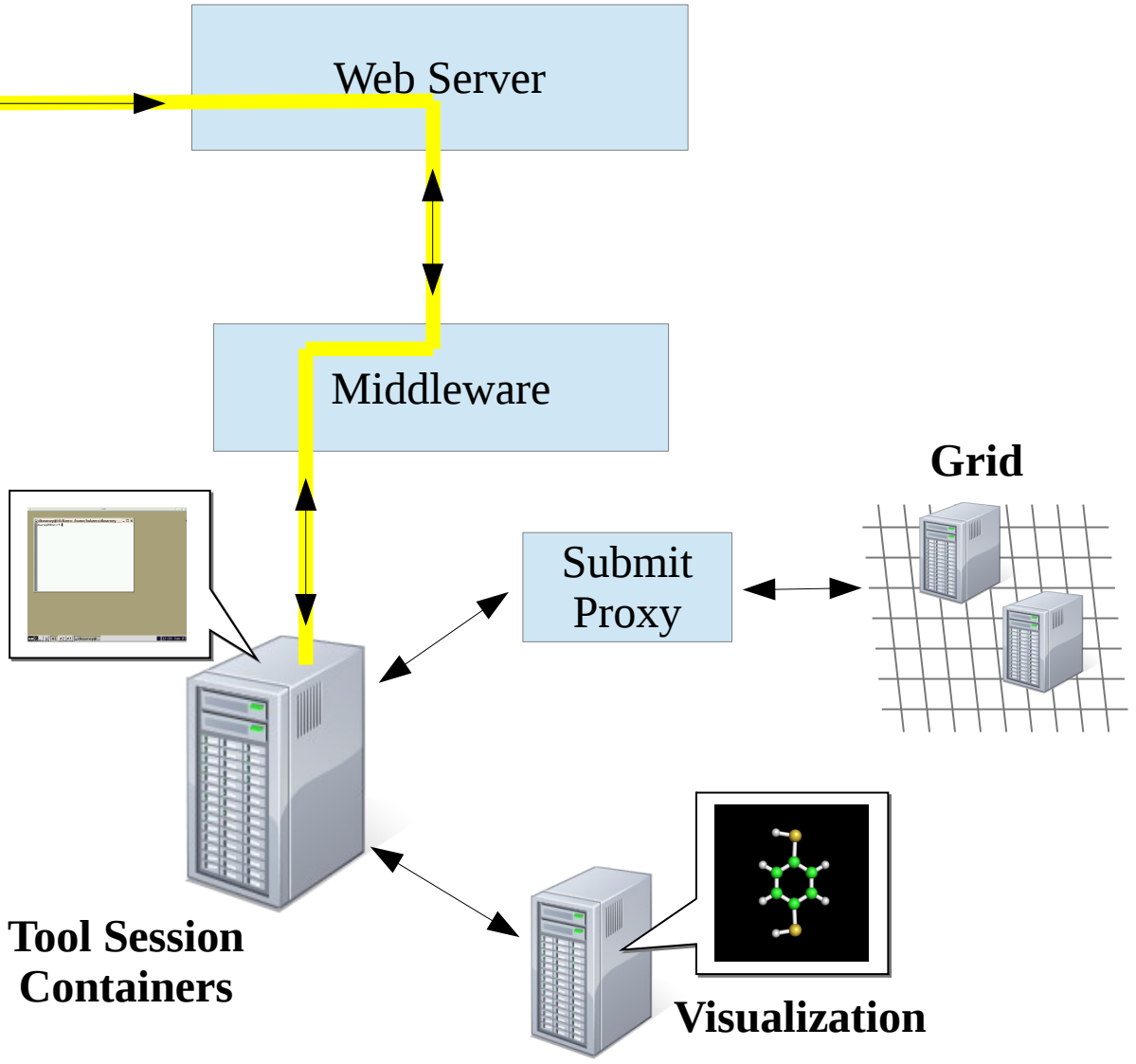
Visualization

Workspace via Website

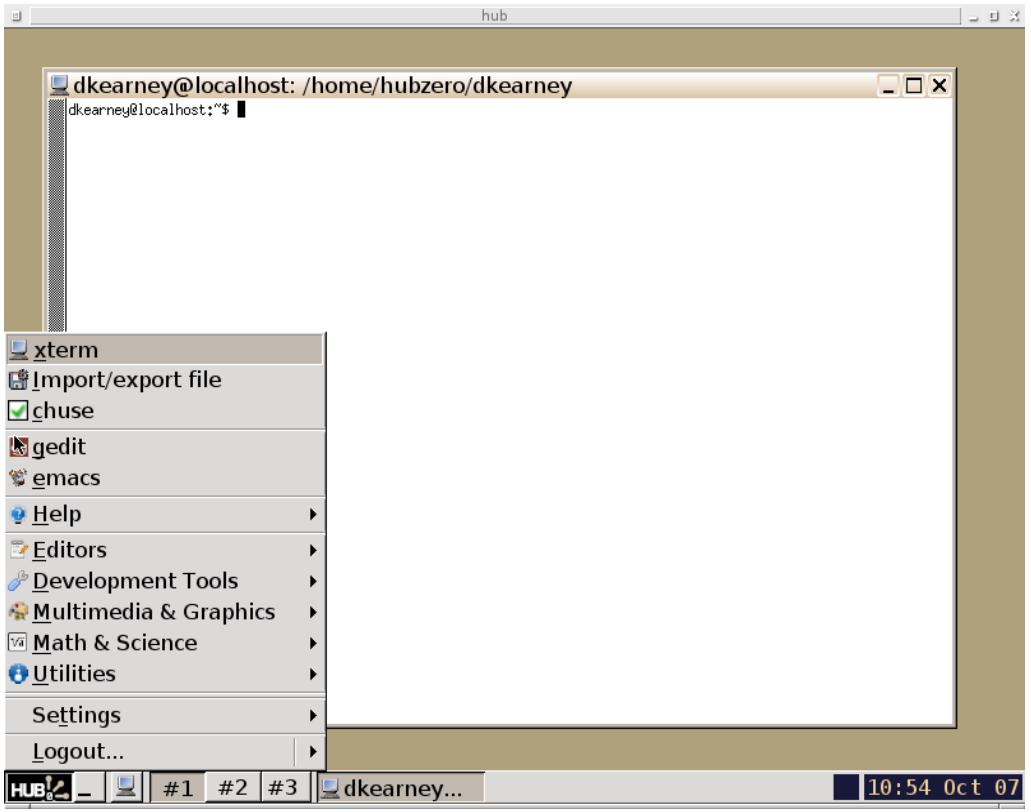
User's Web Browser

HUBzero Infrastructure





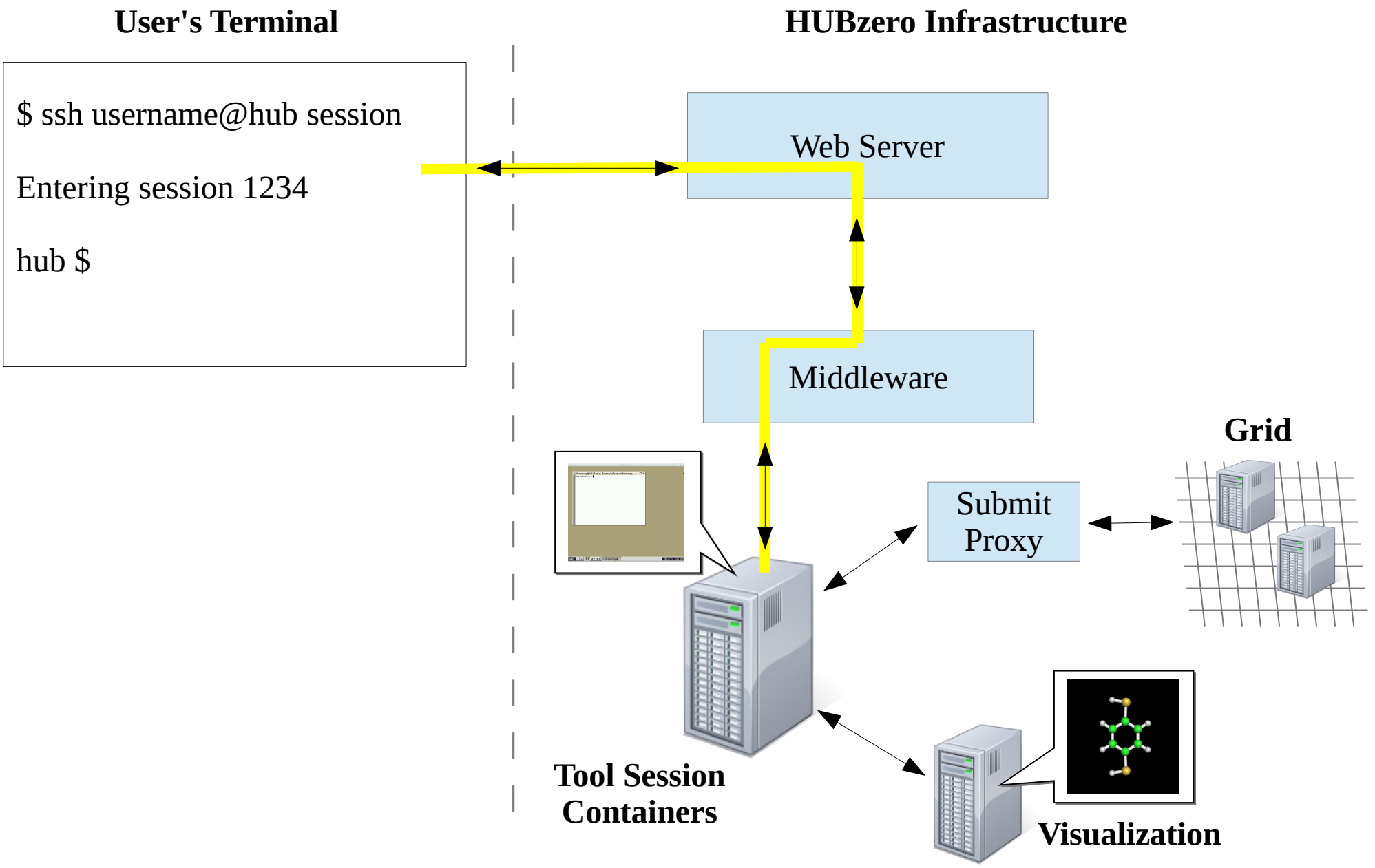
Using a Workspace



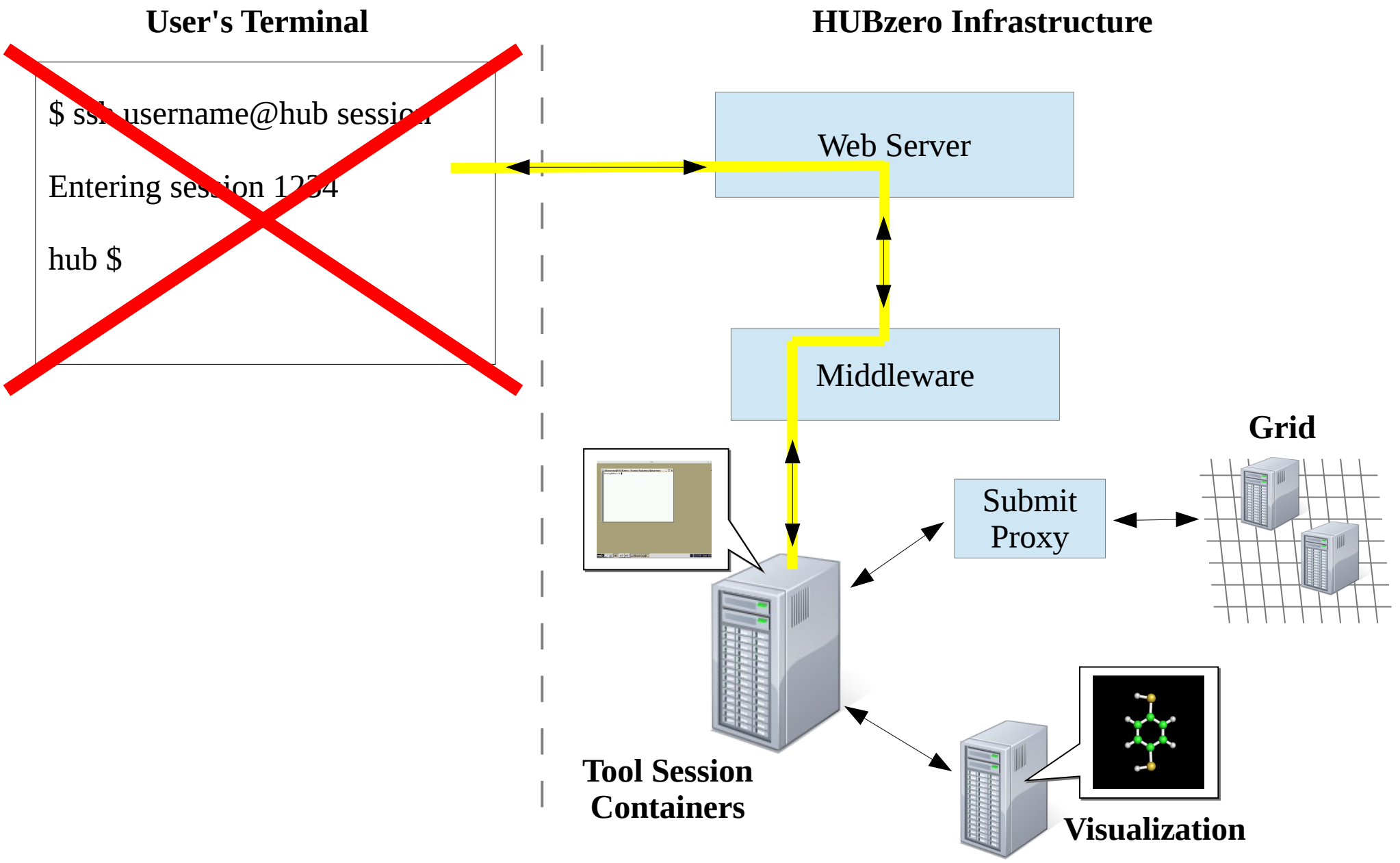
Development Tools Available
In Workspace:

* **Editors**: geany, emacs, vim
* **Debuggers**: gdb, ddd, valgrind
* **Compilers**: gcc, g++, gfortran
* **Interpreters**: wish, python, R, octave, irb, perl,

Workspace via VirtualSSH



Workspace via VirtualSSH



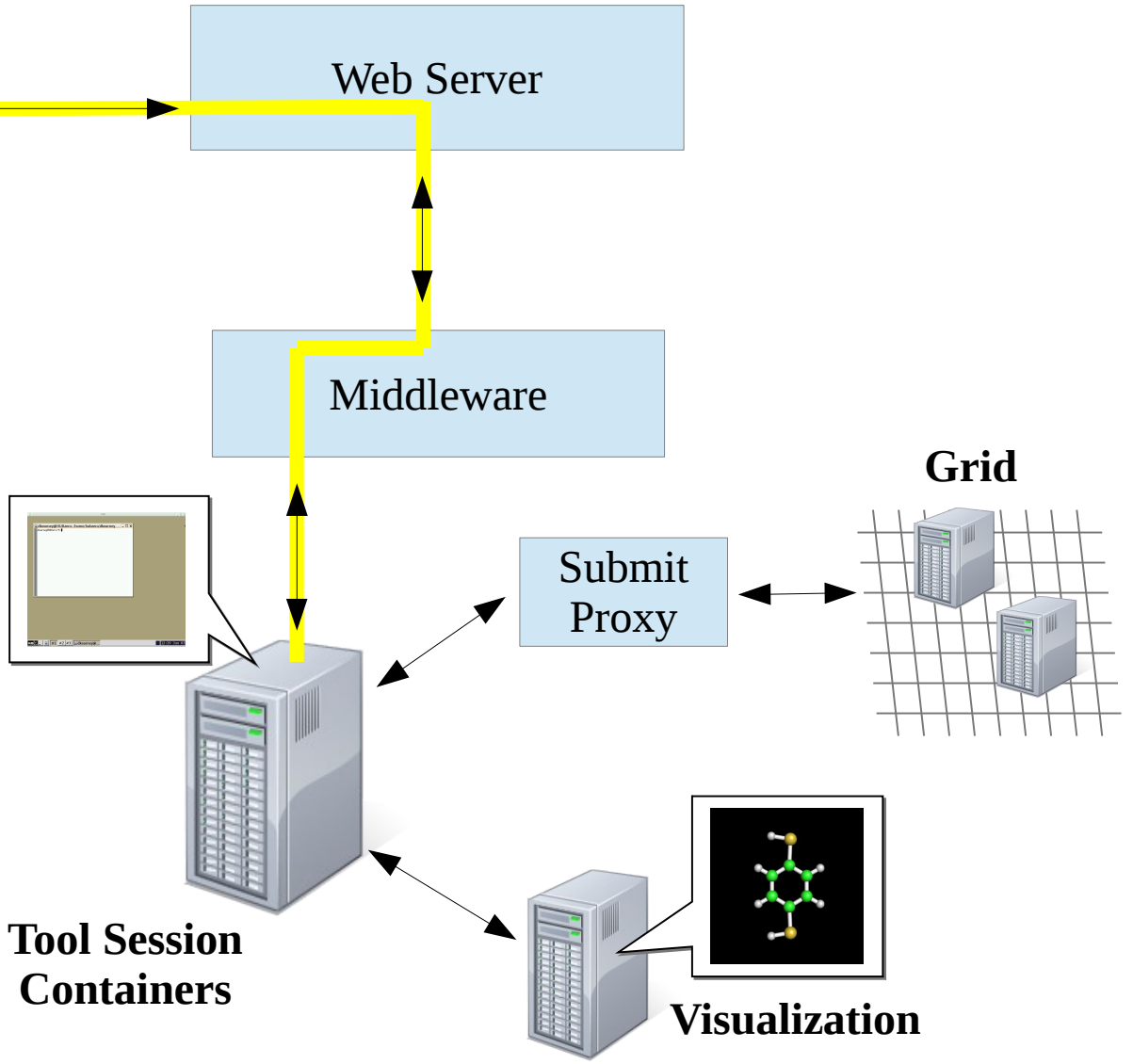
Workspace via HUBcheck

Python Script

```
from hubcheck import ToolSession

ts = ToolSession(host,user,passwd)

shell = ts.access()

out,status = shell.execute(' echo hi')
```

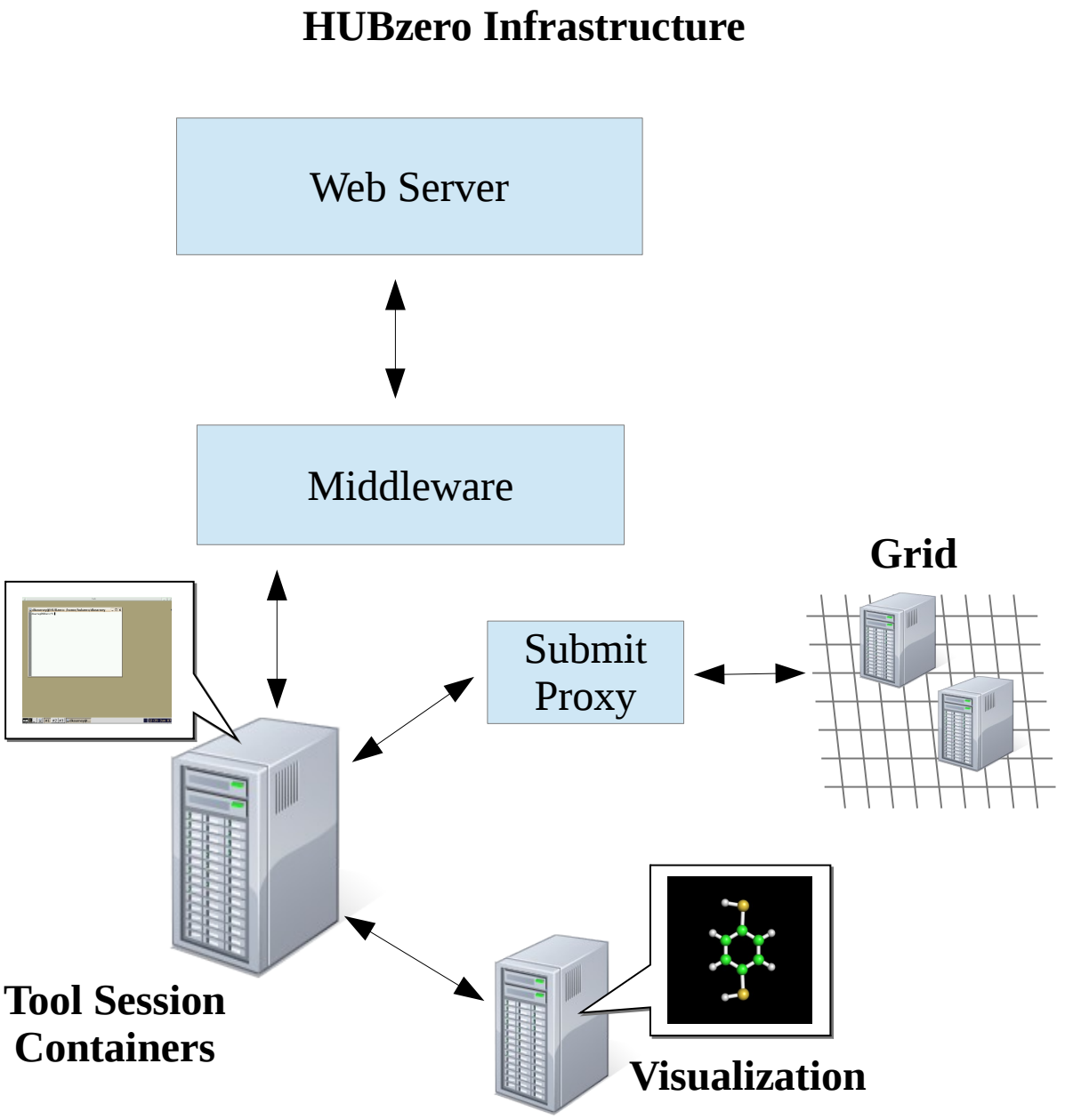
Expect-like interface for running commands

HUBzero Infrastructure



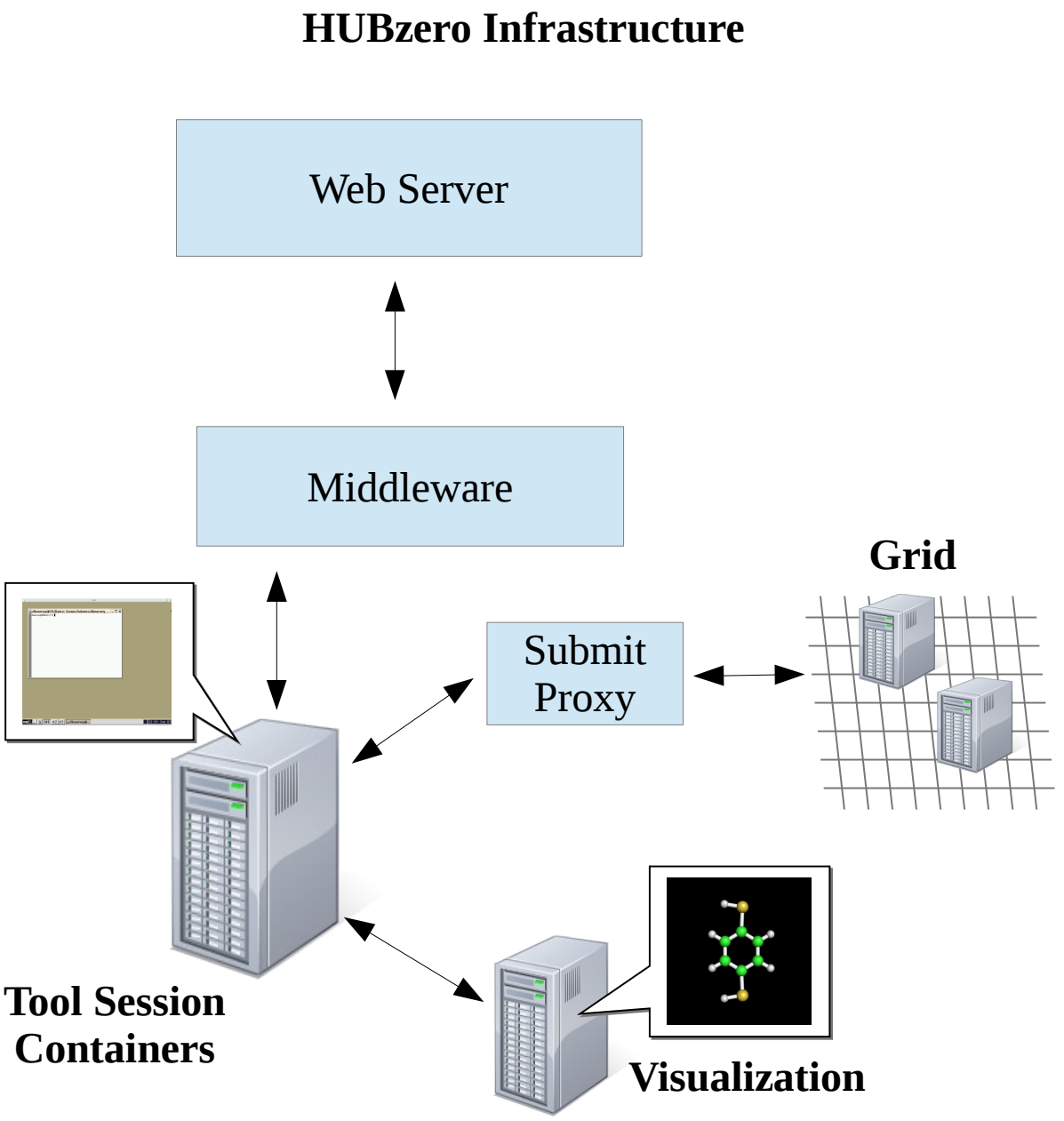
Testing in the Workspace

Debian Squeeze Packages
Container Setup
Network Firewall
Rappture Toolkit
Filexfer
Submit



Testing in the Workspace

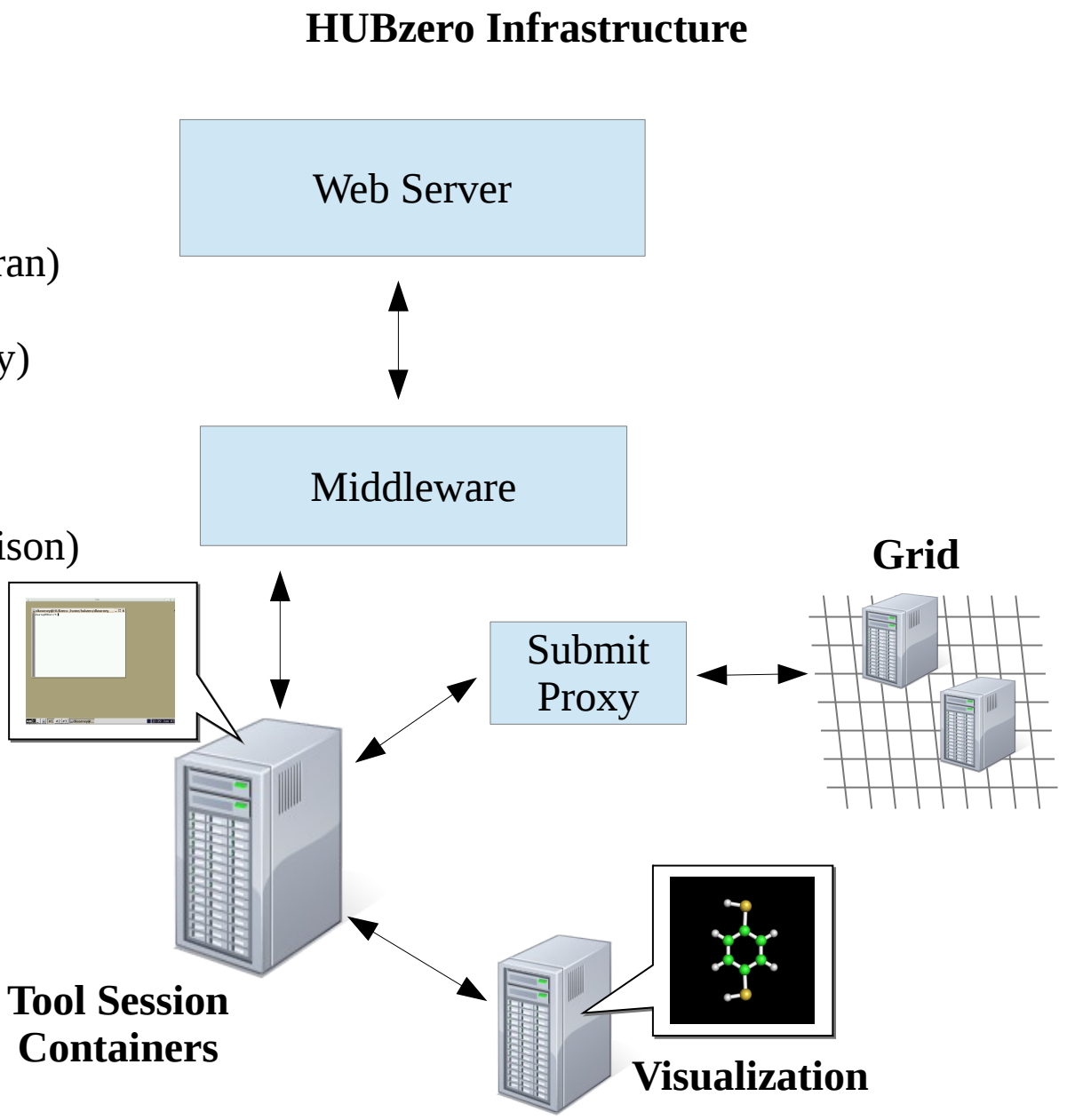
Debian Squeeze Packages
Container Setup
Network Firewall
Rappture Toolkit
Filexfer
Submit



Testing Packages

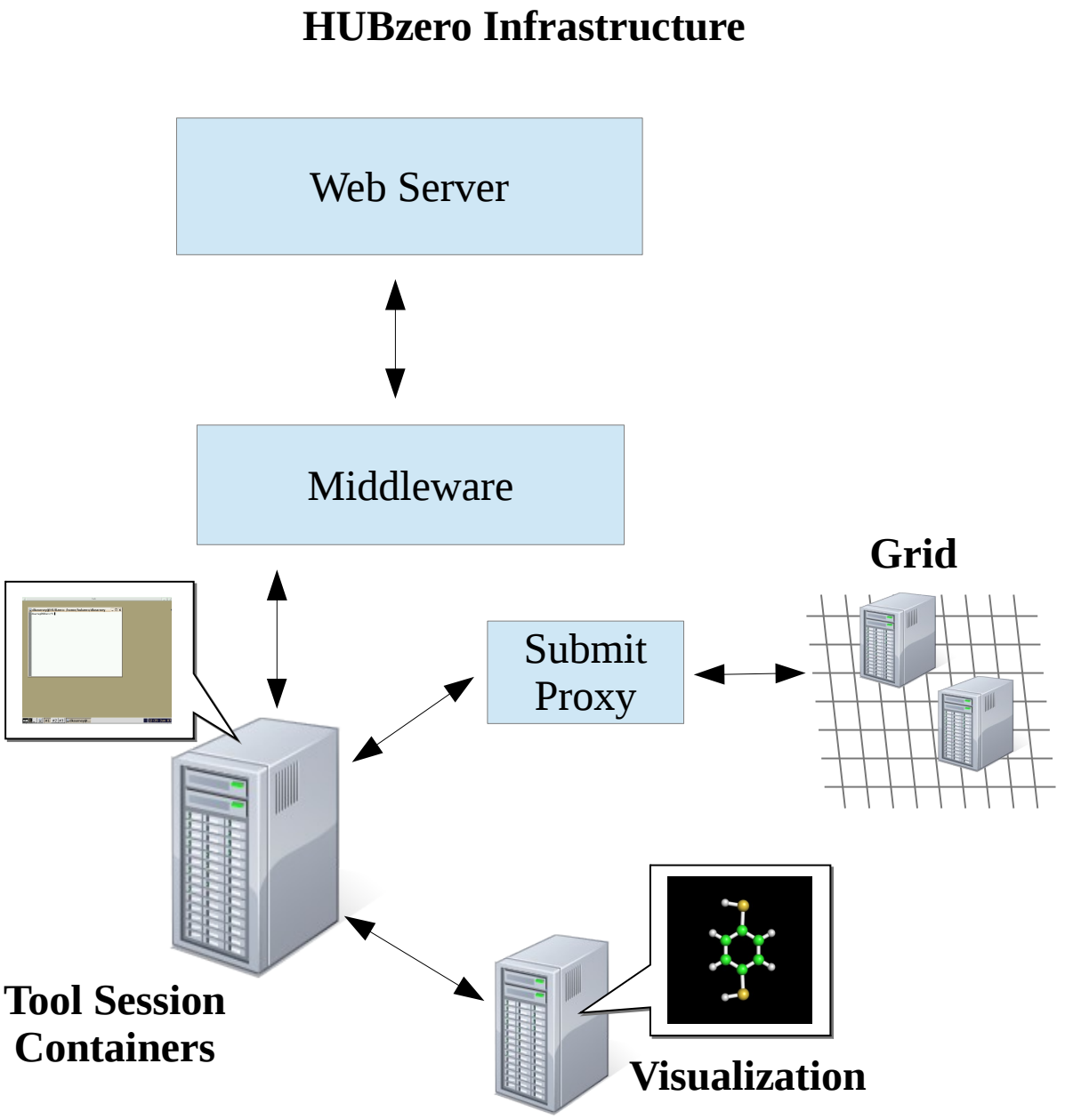
119 Debian Squeeze Packages

- Compilers (gcc, g++, javac, gfortran)
- Debuggers (gdb, valgrind)
- Interpreters (python, perl, tcl, ruby)
- Build tools (make, autoconf)
- Editors (gedit, vim, emacs)
- Utilities (zip, tar, ssh, rsync)
- Window Managers (icewm, ratpoison)
- Development tools



Testing in the Workspace

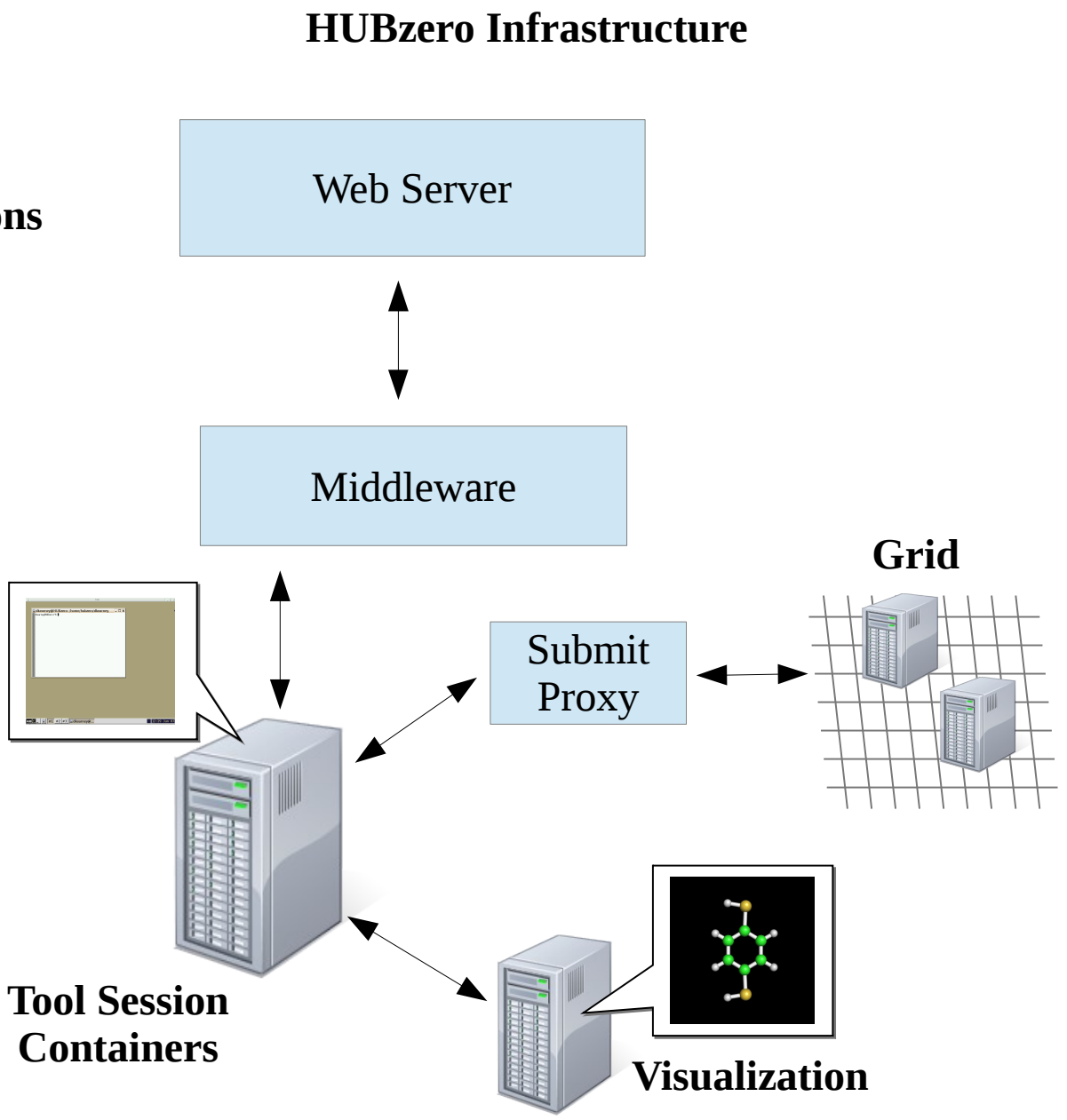
Debian Squeeze Packages
Container Setup
Network Firewall
Rappture Toolkit
Filexfer
Submit



Testing Container Setup

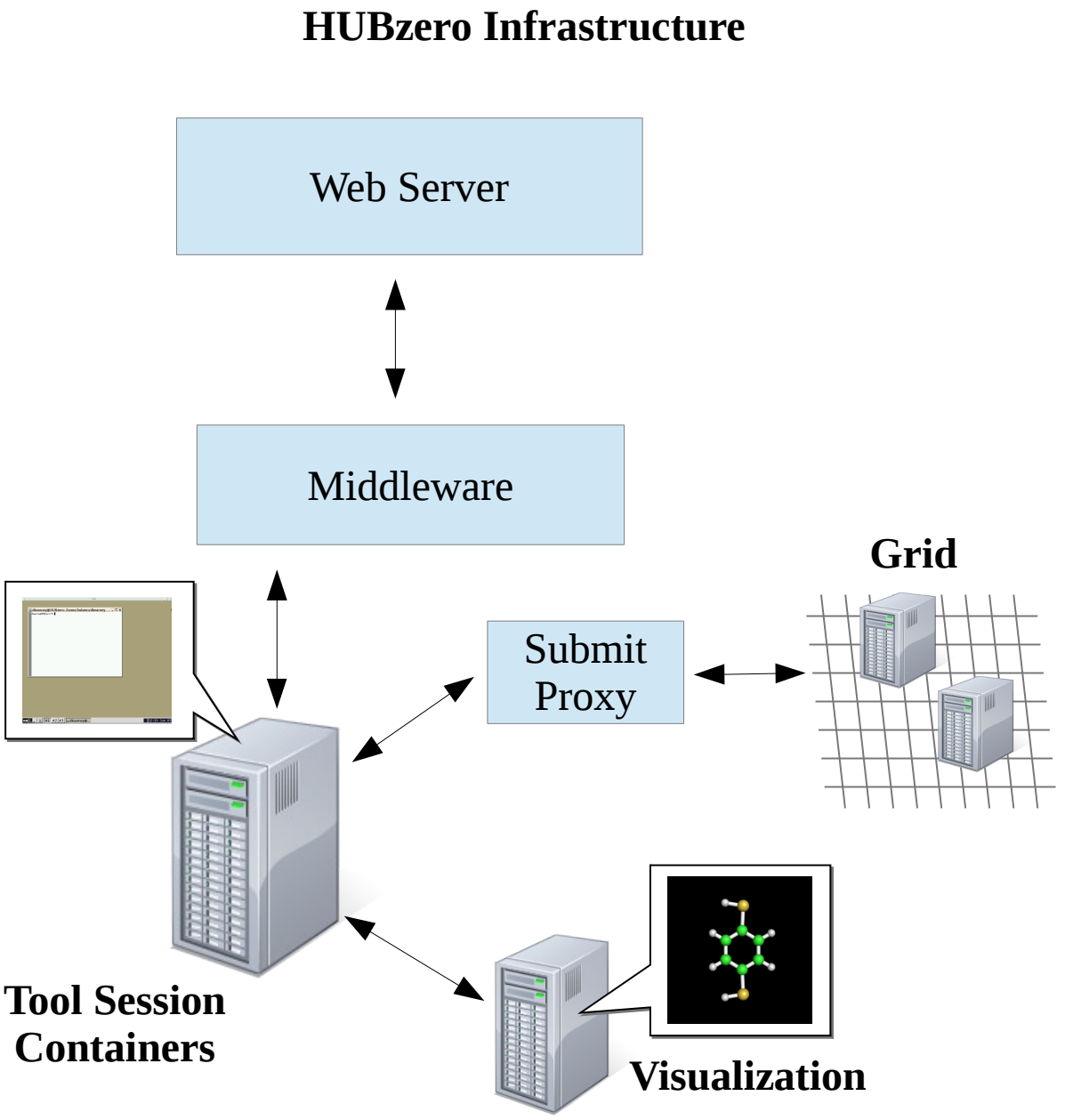
Check file locations and permissions

- filexfer, importfile, exportfile
- clientaction
- pixelflip
- mergeauth
- startxvnc
- xsetroot
- icewm, icewm-captive, ratpoison
- invoke_app
- Testing for old installations



Testing in the Workspace

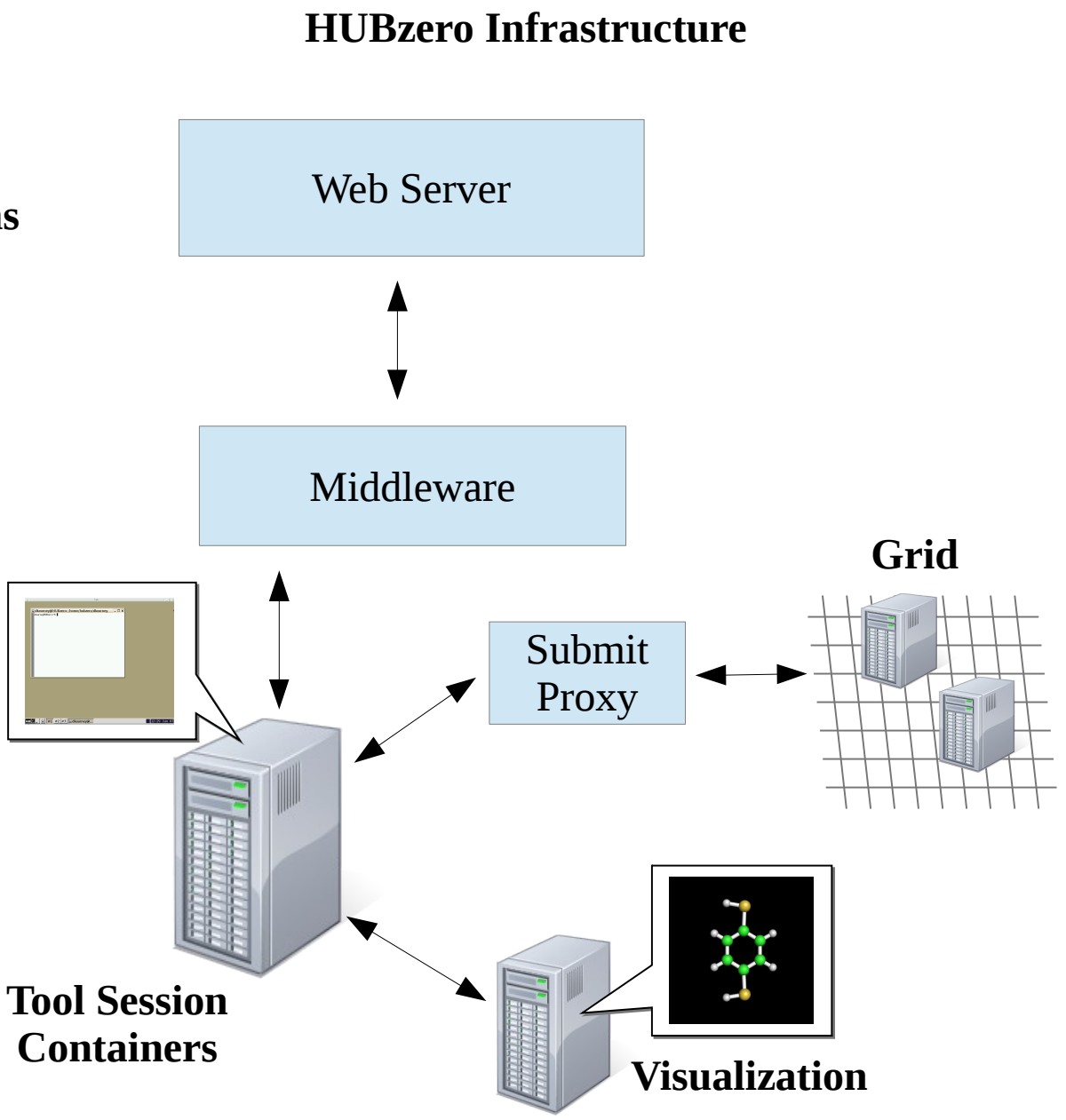
Debian Squeeze Packages
Container Setup
Network Firewall
Rappture Toolkit
Filexfer
Submit



Testing Container Network Firewall

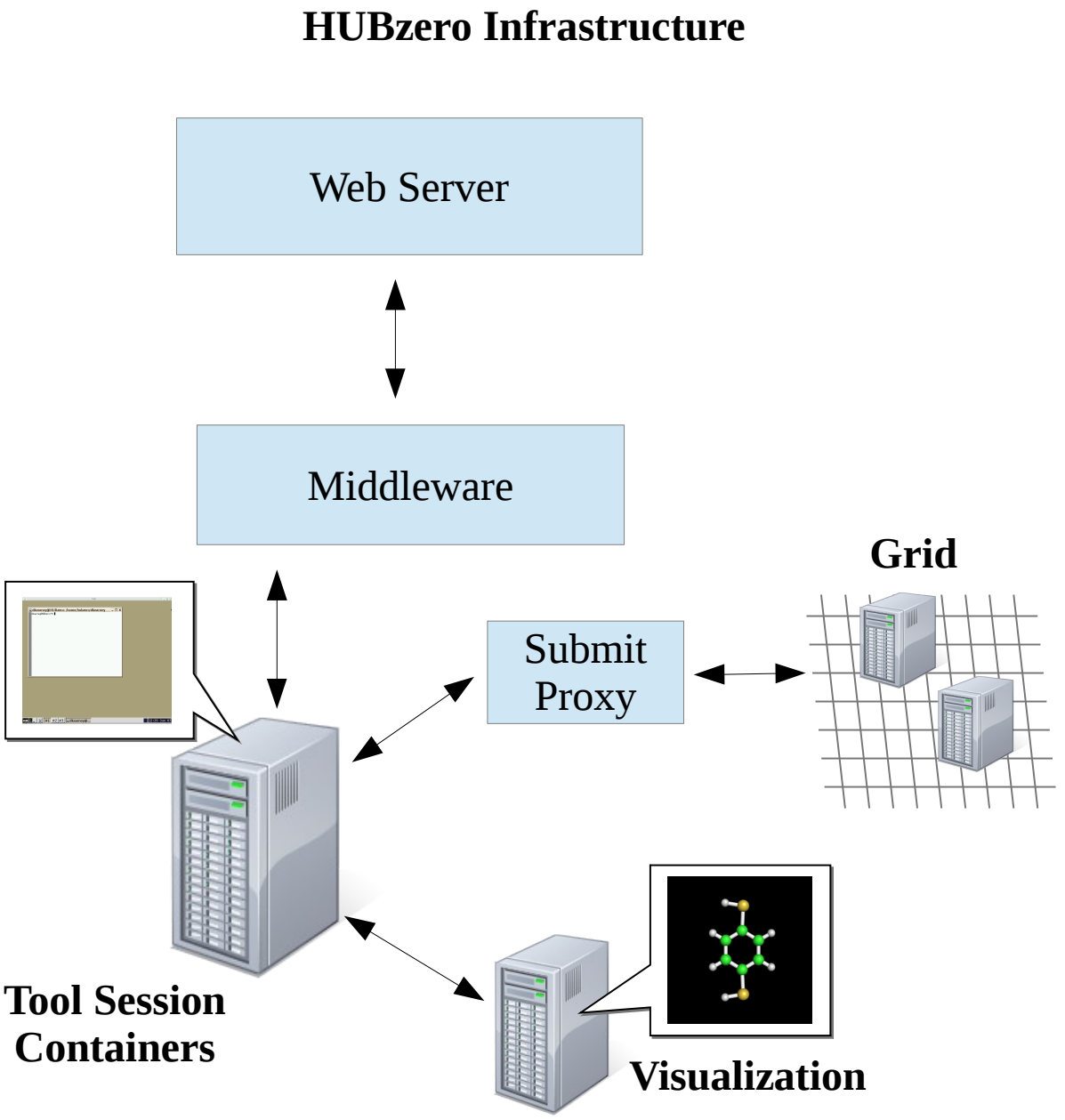
3 User Configurations, 30 locations

- Hub's website port 80 and 443
- Rappture website
- Google
- Campus ssh ports
- Hub nameservers
- OpenDNS nameservers
- Google nameservers
- License servers
- Visualization servers



Testing in the Workspace

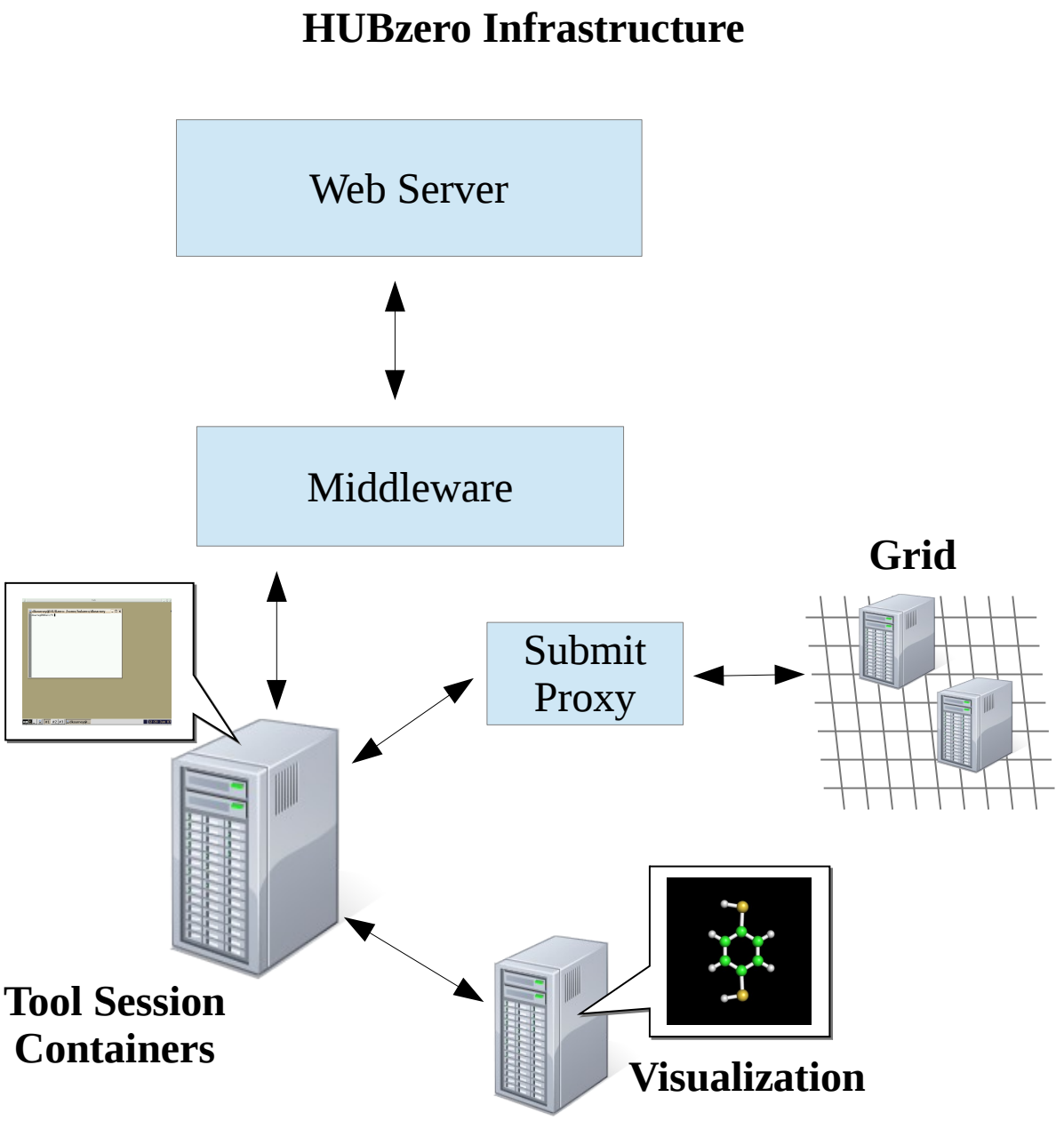
Debian Squeeze Packages
Container Setup
Network Firewall
Rappture Toolkit
Filexfer
Submit



Testing Rappture Toolkit

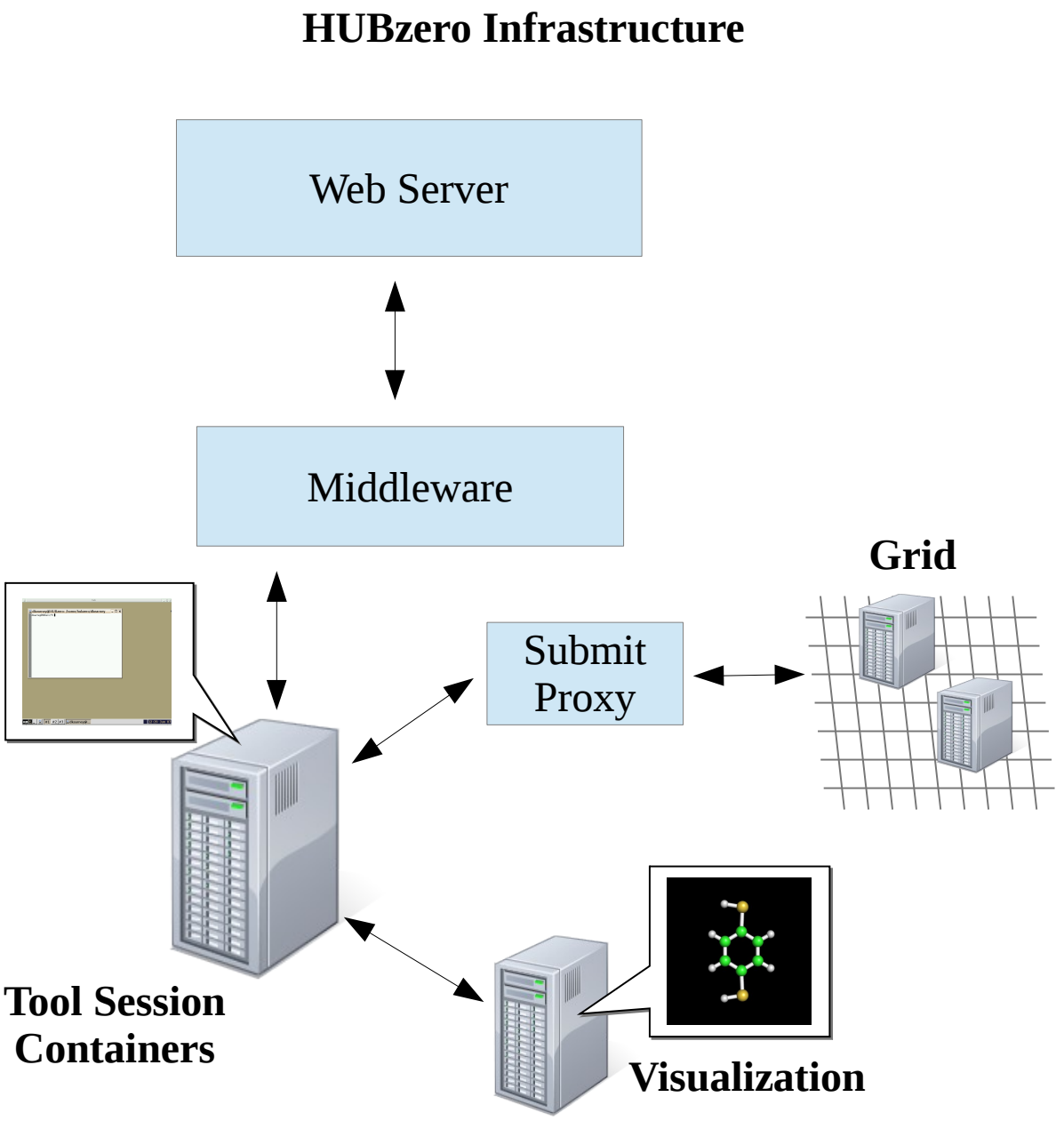
Rappture Toolkit

- Rappture Installation
- Examples Run
- Visualization Servers



Testing in the Workspace

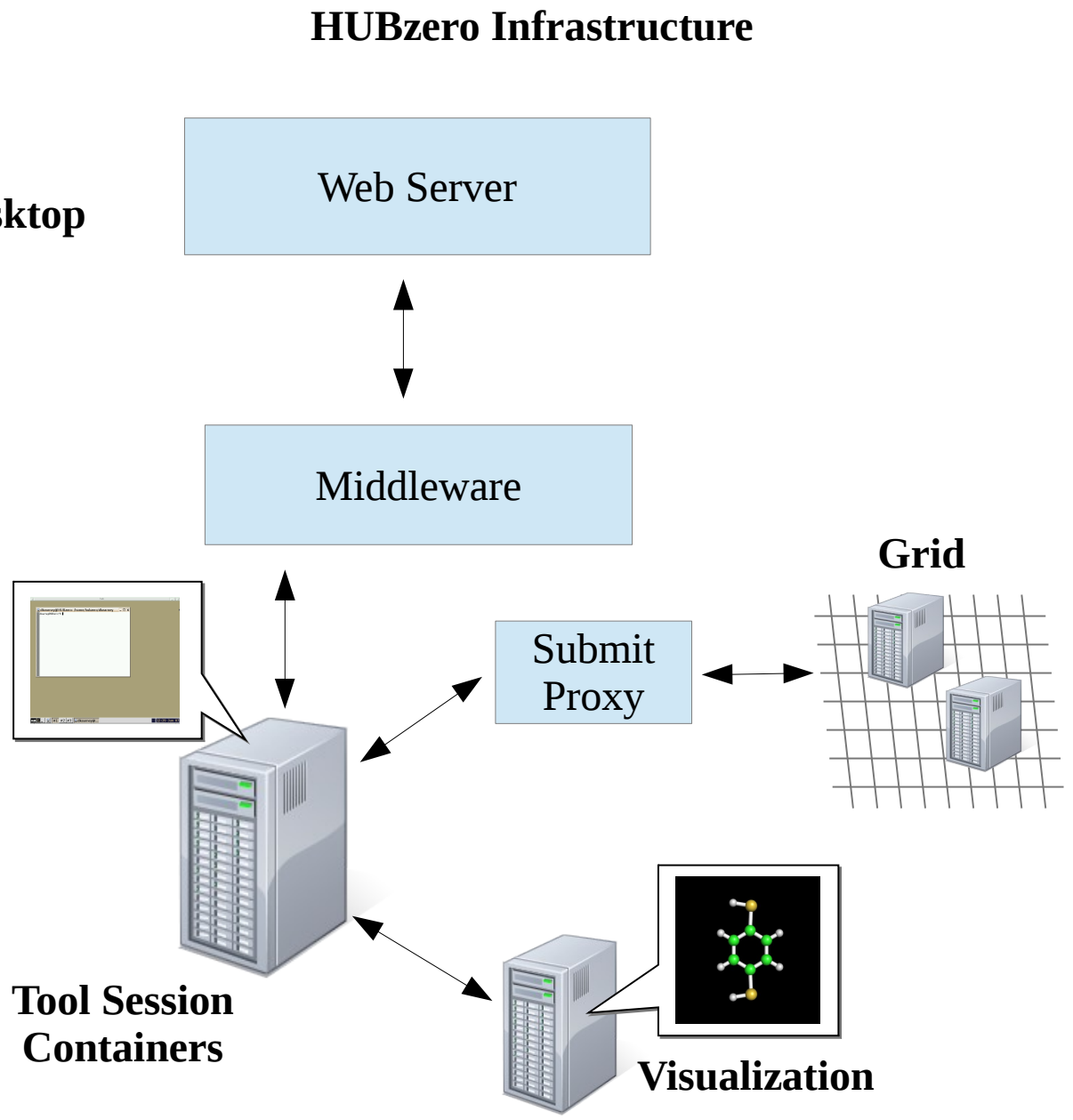
Debian Squeeze Packages
Container Setup
Network Firewall
Rappture Toolkit
Filexfer
Submit



Testing Filexfer

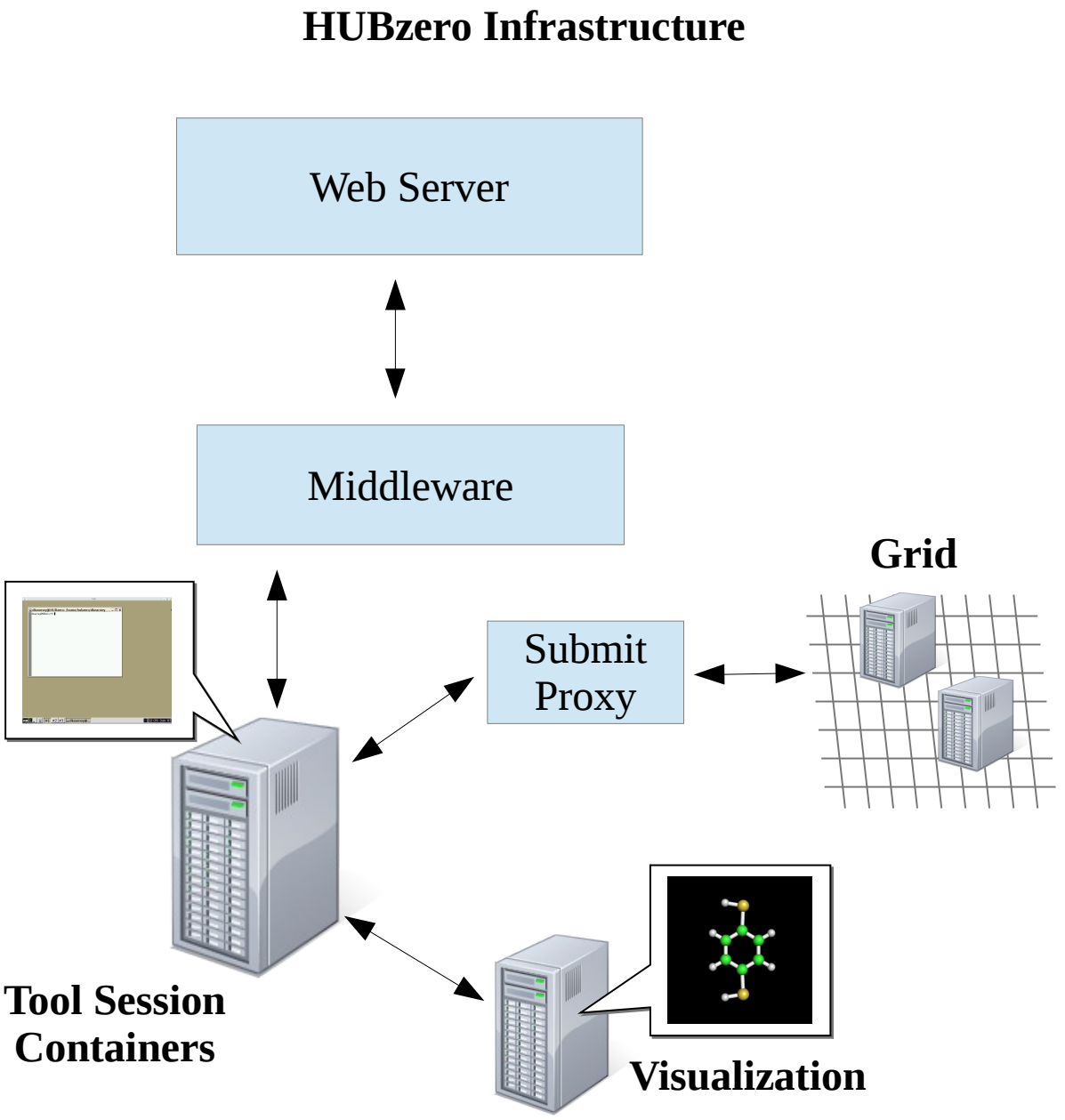
Transfer file between hub and desktop

- Filexfer Installation
- Filexfer GUI
- exportfile: hub to desktop
- importfile: desktop to hub
- Requires Selenium and Expect



Testing in the Workspace

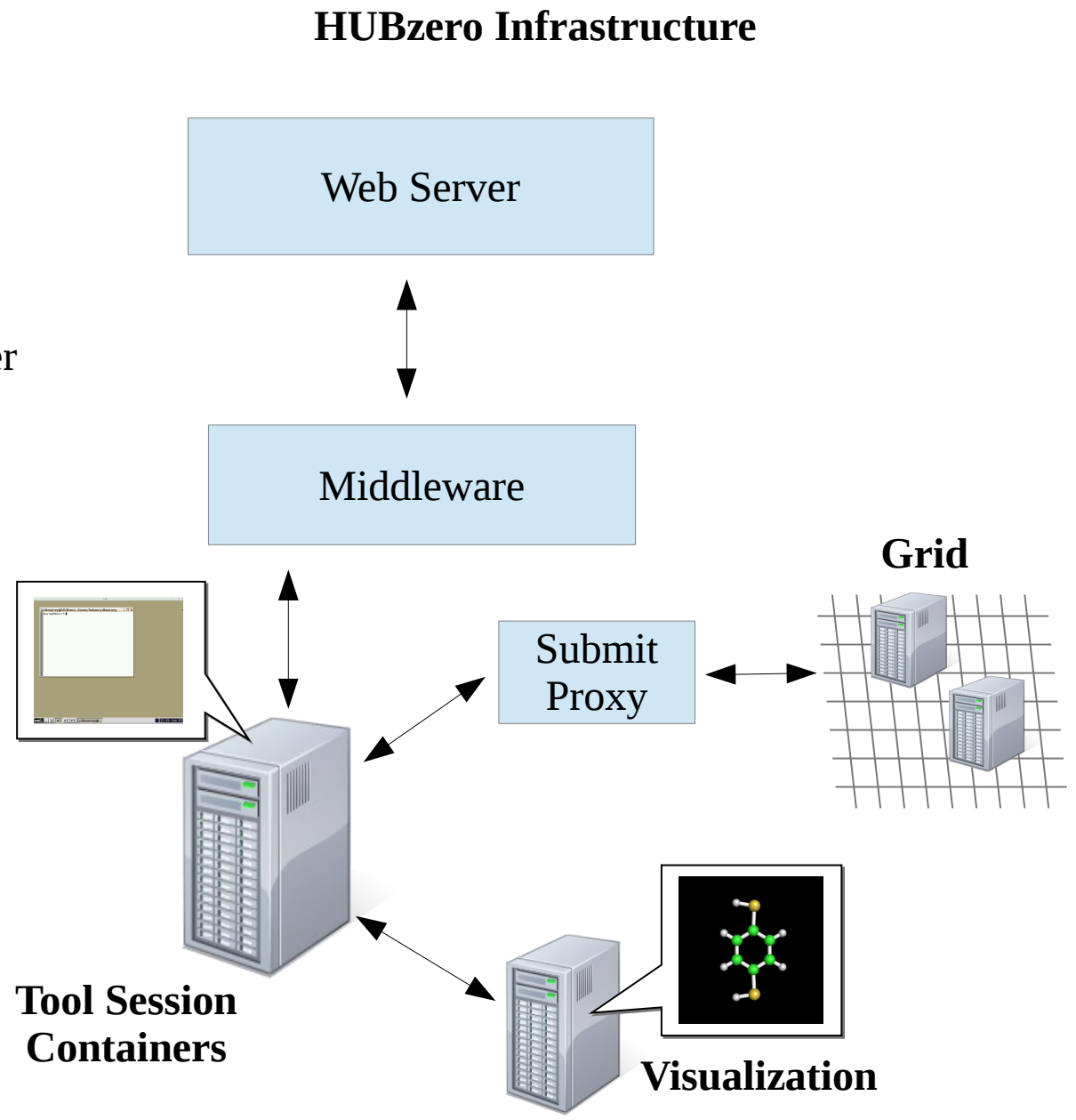
Debian Squeeze Packages
Container Setup
Network Firewall
Rappture Toolkit
Filexfer
Submit



Testing Submit

Job submission to the Grid

- Submit Installation
- Submit job as registered user
- Submit job as submit enabled user
- Submit locally vs to the Grid
- Collecting metrics from Submit



Writing Tests for the Workspace

 # we would like to run a command like this in the workspace to test Submit
$ submit -p @@vth=0:0.2:5 -p @@cap=10pf,100pf,1uf sim.exe @:indeck

About the command:

- Submits 78 jobs.
- Substitutes values for @@vth and @@cap into @:indeck file
  @@vth goes from 0 to 5 in steps of 0.2 (26 values).
  @@cap takes on 3 values, 10pf, 100pf, 1uf.

  26 x 3 = 78 jobs total.

- indeck is treated as a template
  values are substituted in place of @@vth and @@cap in that file.

  Example indeck:

  [inputs]
  C = @@cap
  Vin = @@vth

Converting VirtualSSH to HUBcheck

ssh [flags] [user@]hostname [command]

Virtual SSH related flags

-t Force pseudo-tty allocation

Virtual SSH Commands

session create [*session_title*]	create a new session
session start	start and enter a new session
session [*session_number*] [*command*]	access session
session list	list user's existing sessions
session stop *session_number*	stop the specified session
session help	print session help message

VirtualSSH → ToolSession

Using VirtualSSH you would type...

ssh [flags] [user@]hostname [command]

Virtual SSH related flags

-t Force pseudo-tty allocation

Virtual SSH Commands

session create [*session_title*]

session start

session [*session_number*] [*command*]

session list

session stop *session_number*

session help

In HUBcheck you would type...

ts = ToolSession(...)

ToolSession Object Methods

create(*title*=None)

start()

access(*snum*=None,*command*=None)

list()

stop(*session_number*)

help()

VirtualSSH - Create Session

Using VirtualSSH you would type...

ssh -t user@hostname session create

ssh -t user@hostname session create mytitle

In HUBcheck you would type...

```
ts = ToolSession(hostname,
        username = username,
        password = password)

(stdin,stdout,stderr) = ts.create()

(stdin,stdout,stderr) = ts.create('mytitle')
```

VirtualSSH - Start Session

Using VirtualSSH you would type...

ssh -t user@hostname session start

In HUBcheck you would type...

```
ts = ToolSession(hostname,
        username = username,
        password = password)

shell = ts.start()
```

VirtualSSH - Access Session

Using VirtualSSH you would type...

```
ssh -t user@hostname session

ssh -t user@hostname session 40032

ssh -t user@hostname session “echo hi”

ssh -t user@hostname session 40032 “echo hi”
```

In HUBcheck you would type...

```
ts = ToolSession(hostname,
        username = username,
        password = password)

shell = ts.access()

shell = ts.access(session_number=40032)

(in,out,err) = ts.access(command='echo hi')

(in,out,err) = ts.access(40032,'echo hi')
```

VirtualSSH - List Session

Using VirtualSSH you would type...

ssh -t user@hostname session list

In HUBcheck you would type...

```
ts = ToolSession(hostname,
        username = username,
        password = password)

(stdin,stdout,stderr) = ts.list()
print stdout.read(1024)
```

VirtualSSH - Stop Session

Using VirtualSSH you would type...

```
ssh -t user@hostname session stop 40032
```

In HUBcheck you would type...

```
ts = ToolSession(hostname,
        username = username,
        password = password)

(stdin,stdout,stderr) = ts.stop(40032)
print stdout.read(1024)
```

VirtualSSH – Session Help

Using VirtualSSH you would type...

ssh -t user@hostname session help

In HUBcheck you would type...

```
ts = ToolSession(hostname,
        username = username,
        password = password)

(stdin,stdout,stderr) = ts.help()
print stdout.read(1024)
```

ToolSession → ToolSessionShell

ts = ToolSession(hostname, username = username, password = password)

shell = ts.access()

Provides an Expect-like interface for running shell commands

ToolSessionShell Methods

shell.send(command)

shell.expect(patterns=[],flags=0)

shell.execute(commands)

Examples

shell.send('echo hi')

ToolSession → ToolSessionShell

ts = ToolSession(hostname, username = username, password = password)

shell = ts.access()

Provides an Expect-like interface for running shell commands

ToolSessionShell Methods

shell.send(command)

shell.expect(patterns=[],flags=0)

shell.execute(commands)

Examples

```
shell.send('echo hi')
shell.expect(['(\w+)'])
output = shell.match.groups()[0]
```

Expect Basics

Written by Don Libes in early 1990's

Core is written in C
Language bindings for Tcl
Ported to Python, Perl, other languages

Expect Scripts Typically Have 3 Parts:

- Spawn the interactive program
- Expect phrase
- Send response

Expect Example

Written by Don Libes in early 1990's

Core is written in C
Language bindings for Tcl
Ported to Python, Perl, other languages

Expect Scripts Typically Have 3 Parts:

- Spawn the interactive program
- Expect phrase
- Send response

User's Terminal	Expect Script
$ passwd libes	spawn passwd [lindex $argv 0]
Changing password for libes on thunder	set password [lindex $argv 1]
New password:	expect “password:”
Retype new password:	send “$password\r”
	expect “password:”
	send “$password\r”
	expect eof

Expect Example

Written by Don Libes in early 1990's

Core is written in C
Language bindings for Tcl
Ported to Python, Perl, other languages

Expect Scripts Typically Have 3 Parts:

- Spawn the interactive program
- Expect phrase
- Send response

User's Terminal	**Expect Script**
	spawn passwd [lindex $argv 0]
$ passwd libes	set password [lindex $argv 1]
Changing password for libes on thunder	expect “password:”
New password:	send “$password\r”
Retype new password:	expect “password:”
	send “$password\r”
	expect eof

Expect Example

Written by Don Libes in early 1990's

Core is written in C
Language bindings for Tcl
Ported to Python, Perl, other languages

Expect Scripts Typically Have 3 Parts:

- Spawn the interactive program
- Expect phrase
- Send response

User's Terminal	**Expect Script**
	spawn passwd [lindex $argv 0]
$ passwd libes	set password [lindex $argv 1]
Changing password for libes on thunder	expect “password:”
New password:	send “$password\r”
Retype new password:	expect “password:”
	send “$password\r”
	expect eof

Expect Example

Written by Don Libes in early 1990's

Core is written in C
Language bindings for Tcl
Ported to Python, Perl, other languages

Expect Scripts Typically Have 3 Parts:

- Spawn the interactive program
- Expect phrase
- Send response

User's Terminal	**Expect Script**
	spawn passwd [lindex $argv 0]
$ passwd libes	set password [lindex $argv 1]
Changing password for libes on thunder	expect “password:”
New password:	send “$password\r”
Retype new password:	expect “password:”
	send “$password\r”
	expect eof

Expect Example

Written by Don Libes in early 1990's

Core is written in C
Language bindings for Tcl
Ported to Python, Perl, other languages

Expect Scripts Typically Have 3 Parts:

- Spawn the interactive program
- Expect phrase
- Send response

User's Terminal	**Expect Script**
	spawn passwd [lindex $argv 0]
$ passwd libes	set password [lindex $argv 1]
Changing password for libes on thunder	expect “password:”
New password:	send “$password\r”
Retype new password:	expect “password:”
	send “$password\r”
	expect eof

Examples – Execute Command

```
ts = ToolSession(hostname, username = username, password = password)
shell = ts.access()

command = 'submit --local echo hi'
output, error_code = shell.execute(command)
print output                # hi
print error_code            # 0
```

Expect-like interface simplified using the *Execute Pattern*:

- **Send** *text*
- **Expect** sent *text*
- **Wait** for command to complete
- **Capture** command output
- **Capture** command exit status

Examples – Transferring Files

```
from hubcheck import SFTPClient, ToolSession

ts = ToolSession(hostname, username = username, password = password)

shell = ts.access()

shell.execute('cd $SESSIONDIR')
sessiondir,error_code = shell.execute('pwd')

shell.importfile('./sim1.py', '../examples/sim1.py', mode=0o700)
```

Examples – Transferring Files

```
from hubcheck import SFTPClient, ToolSession

ts = ToolSession(hostname, username = username, password = password)
sftp = SFTPClient(hostname, username = username, password = password)
shell = ts.access()

shell.execute('cd $SESSIONDIR')
sessiondir,error_code = shell.execute('pwd')

sftp.chdir(sessiondir)
sftp.put('../examples/sim1.py', './sim1.py')
sftp.chmod('./sim1.py', 0700)
```

Examples – Writing Files

```
from hubcheck import SFTPClient, ToolSession

ts = ToolSession(hostname, username = username, password = password)

shell = ts.access()

shell.execute('cd $SESSIONDIR')
sessiondir,error_code = shell.execute('pwd')


exe_path = './sim1.py'
indeckfn = 'indeck.template'
indeck_template = '[inputs]\nC = @@C\n'

shell.importfile(indeck_template,indeckfn,mode=0o600,is_data=True)



command = 'submit --local -p @@C=10e-12,100e-12 %s --inputdeck @:%s'  \
                    % (exe_path,indeckfn)
command += ' 0</dev/null'
output,error_code = self.ws.execute(command)
```

Examples – Writing Files

```
from hubcheck import SFTPClient, ToolSession

ts = ToolSession(hostname, username = username, password = password)
sftp = SFTPClient(hostname, username = username, password = password)
shell = ts.access()

shell.execute('cd $SESSIONDIR')
sessiondir,error_code = shell.execute('pwd')
sftp.chdir(sessiondir)

exe_path = './sim1.py'
indeckfn = 'indeck.template'
indeck_template = '[inputs]\nC = @@C\n'

f = sftp.open(indeckfn,mode='w')
f.write(indeck_template)
f.close()

command = 'submit --local -p @@C=10e-12,100e-12 %s --inputdeck @:%s'  \
                    % (exe_path,indeckfn)
command += ' 0</dev/null'
output,error_code = self.ws.execute(command)
```

Using HUBcheck to Write Tests

```
import hubcheck
import os

class container_firewall_registered_user(hubcheck.TestCase):

    def setUp():
        …

    def test_basic_connections():
        …

    def tearDown():
        …
```

Writing Tests – setUp Fixture

```
def setUp(self):

    self.remove_files = []
    self.ws = None

    # get user account info
    hubname = self.testdata.find_url_for('https')
    self.username,self.userpass = self.testdata.find_account_for('registeredworkspace')

    cm = hubcheck.ContainerManager()
    self.ws = cm.access(hubname, self.username, self.userpass)

    # copy the checknet executable to the session directory
    self.ws.execute('cd $SESSIONDIR')
    sessiondir,es = self.ws.execute('pwd')

    self.exe_fn = 'checknet.py'
    local_exe_path = os.path.join(hubcheck.config.macros_dir,self.exe_fn)
    self.exe_path = os.path.join(sessiondir,self.exe_fn)
    self.remove_files.append(self.exe_path)

    self.ws.importfile(local_exe_path,self.exe_path,mode=0o700)
```

Writing Tests – Test Method

```
def test_basic_connections(self):
    """
    login to a tool session container and check basic network firewall
    settings for a registered user in no network affecting groups.
    """

    conns = [
    #  (desc,              uri,                                port,  expected_result)
       ('rappture',        'rappture.org',                      80,  True),
       ('ecn_systems',     'shay.ecn.purdue.edu',               22,   False),
       ('google_https',    'google.com',                       443,  False),
       ('octave_ftp',      'ftp.octave.org',                    21,  False),
       ('localhost',       'localhost',                         80,  False),
       ('ecn_matlab',      'matlab-license.ecn.purdue.edu', 1703, False),
    ]

    results = ''
    for (desc,uri,port,eresult) in conns:
        results += self._run_checknet(desc,uri,port,eresult)

    self.assertTrue(results == '', results.strip())
```

Writing Tests – Test Method

```
def _run_checknet(self,desc,uri,port,eresult):

    command = '%s --protocol tcp4 %s %s' % (self.exe_path,uri,port)
    self.logger.debug('command = "%s"' % (command))
    aresult,es = self.ws.execute(command)

    if aresult == 'True':
        aresult = True
    else:
        aresult = False

    results = ''
    if eresult != aresult:
        results = '\n%s connection %s:%s received %s, expected %s' \
                % (desc,uri,port,aresult,eresult)

    return results
```

Writing Tests – tearDown Fixture

```
def tearDown(self):

    # remove the executable from the workspace
    for fname in self.remove_files:
        self.ws.execute('rm -f %s' % (fname))

    # get out of the workspace
    # shut down the ssh connection
    if self.ws is not None:
        self.ws.close()
```