Atlanta, Georgia, April 19, 2010
in conjunction with IPDPS 2010

UNIBUS: ASPECTS OF HETEROGENEITY AND FAULT TOLERANCE IN CLOUD COMPUTING

Magdalena Slawinska
Jaroslaw Slawinski
Vaidy Sunderam

{magg, jaross, vss}@mathcs.emory.edu

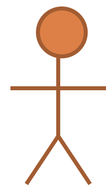
Emory University, Dept. of Mathematics and Computer Science
Atlanta, GA, USA

Creating a problem

1. What do I want?
   - ❑ Execute an MPI application

2. What do I need?
   - ❑ Target resource: MPI cluster
   - ❑ FT services: Checkpoint, Heartbeat

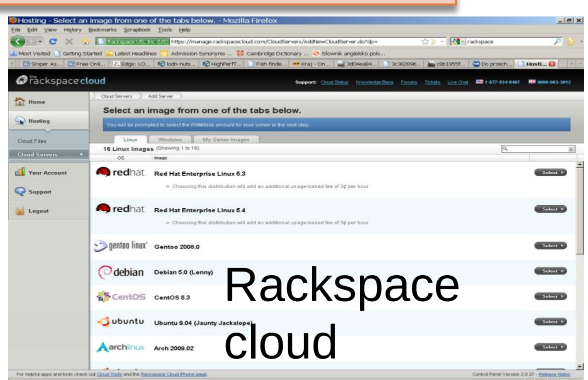
3. What do I have?
   - ❑ Access to the Rackspace cloud



4. Why might I want FT on cloud?
   - ❑ To reduce costs (money, time, energy, …)
   - ❑ Reliability
   - ❑ …

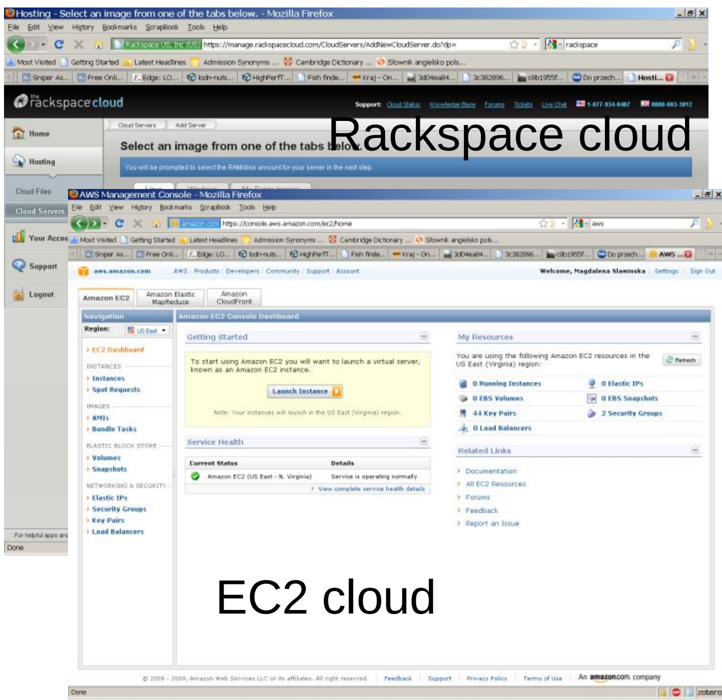
5. What is the overhead introduced by FT?

6. Can I do that? How?



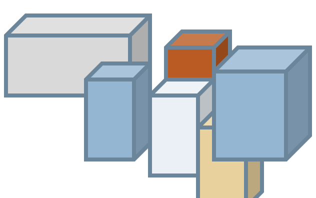
Problem

Available resource



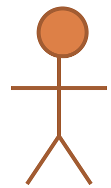
Rackspace cloud

EC2 cloud



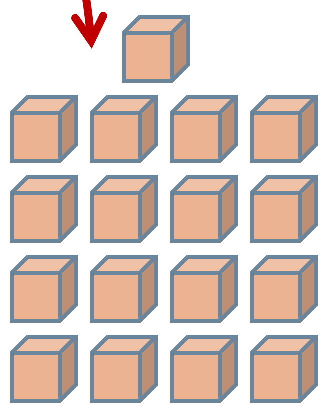
Target resource

Workstations



User

Resource transformation



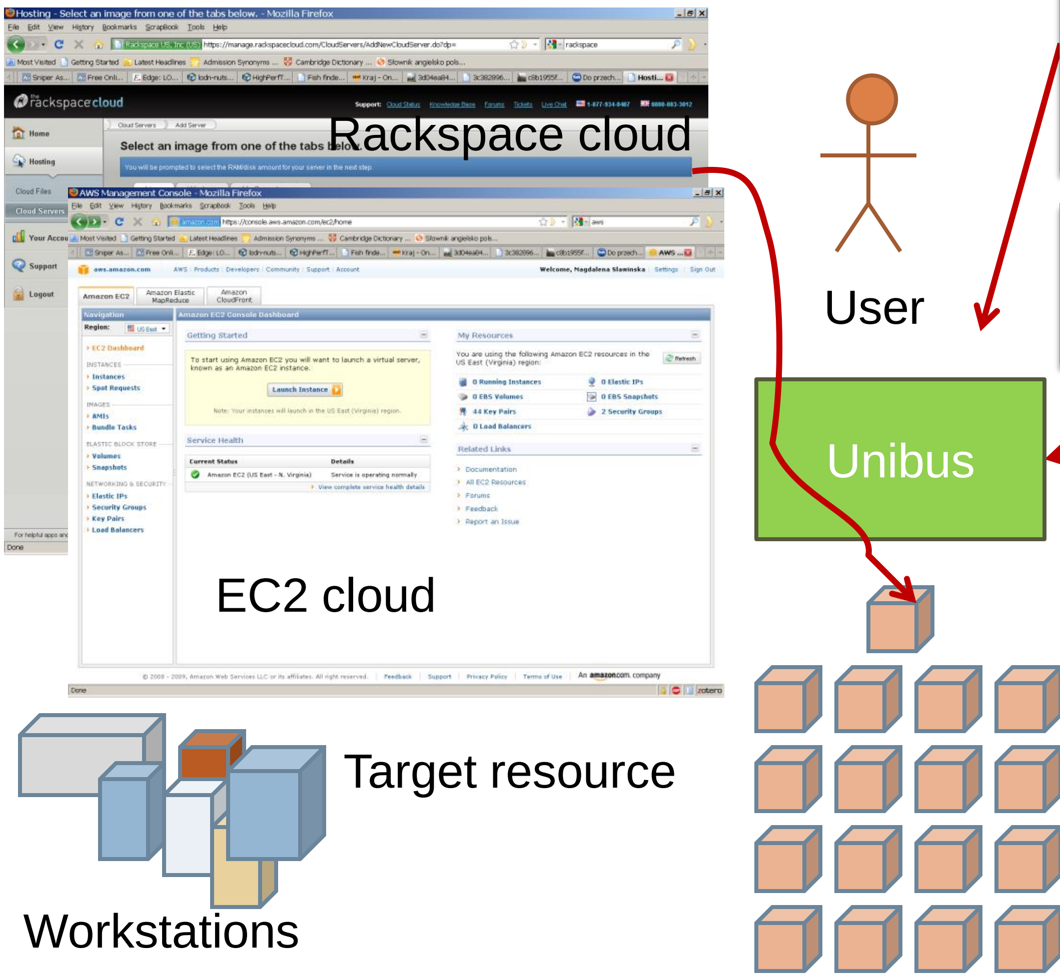
User's requirements
- ❑ Execute MPI software
- ❑ Target resource: MPI cluster
- ❑ Target platform: FT-flavor

User's resources
- ❑ Rackspace cloud (credentials)

Manually:
- ❑ interaction with web page
- ❑ prepare the image: install required software and dependencies
- ❑ instantiate servers
- ❑ configure passwordless authentication
- ❑ ….
- ❑ 1 man-hour for 16+1 nodes

Unibus: a resource orchestrator

Available resource



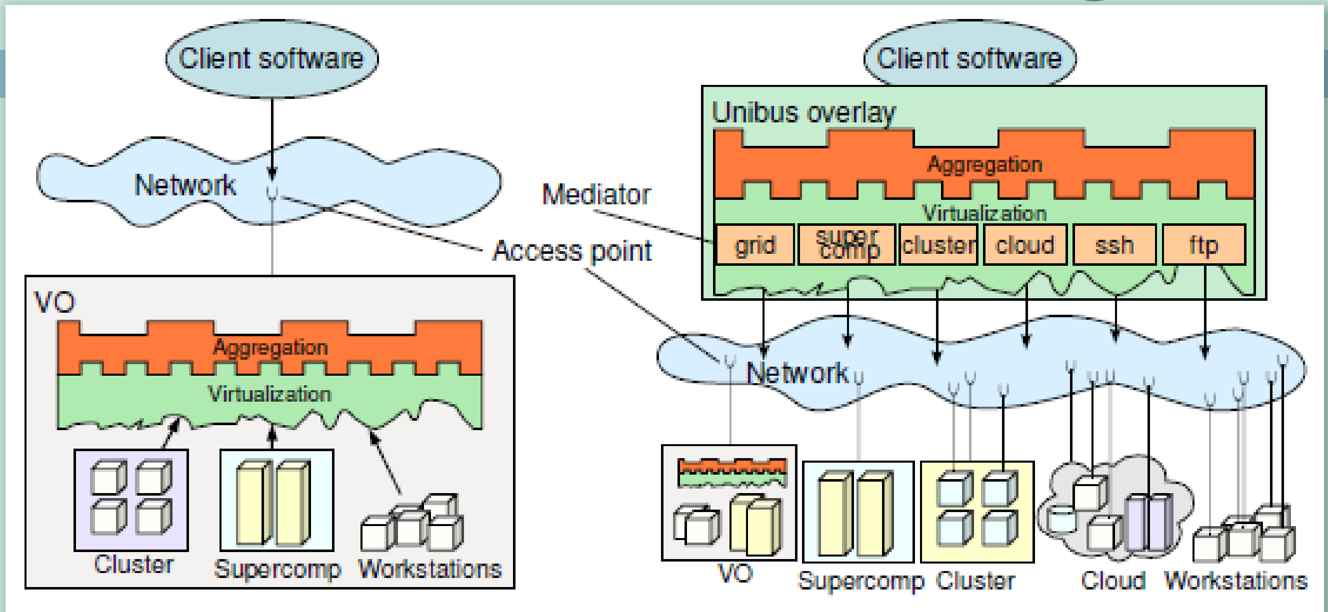
User's requirements
- ❑ Execute MPI software
- ❑ Target resource: MPI cluster
- ❑ Target platform: FT-flavor

User's resources
- ❑ Rackspace cloud (credentials)

Outline

- Unibus – an infrastructure framework that allows to orchestrate resources
  - Resource access virtualization
  - Resource provisioning
- Unibus – FT MPI platform on demand
  - Automatic assembly of an FT MPI-enabled platform
  - Execution of an MPI application on the Unibus-created FT MPI-enabled platform
  - Discussion of the FT overhead

Unibus resource sharing model

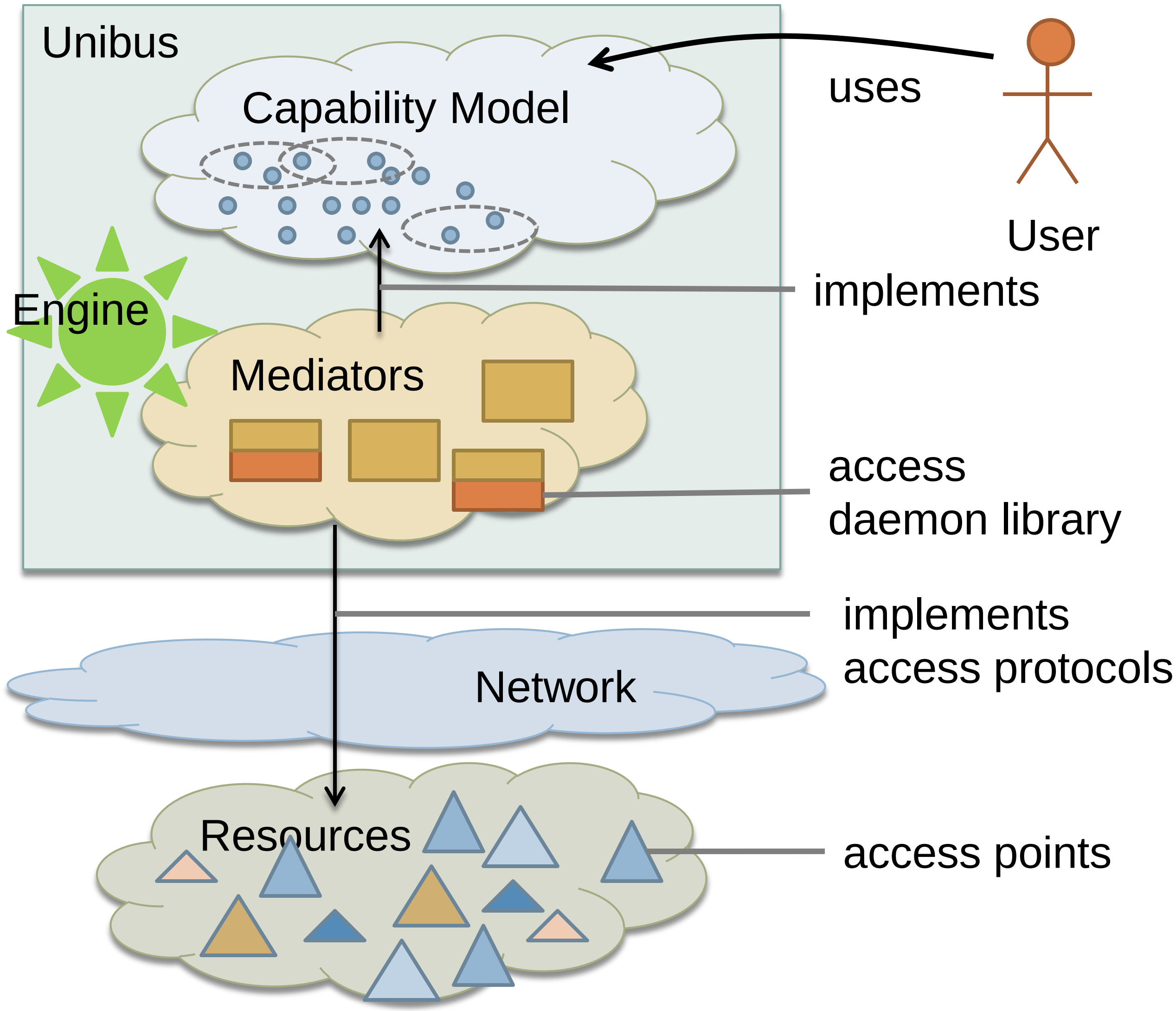


(a) Traditional Model (b) Proposed Model

	Traditional Model	Proposed Model
Resource exposition	Virtual Organization (VO)	Resource provider
Resource usage	Determined by VO	Determined by a particular resource provider
Resource virtualization and aggregation	Resource providers belonging to VO	Software at the client side

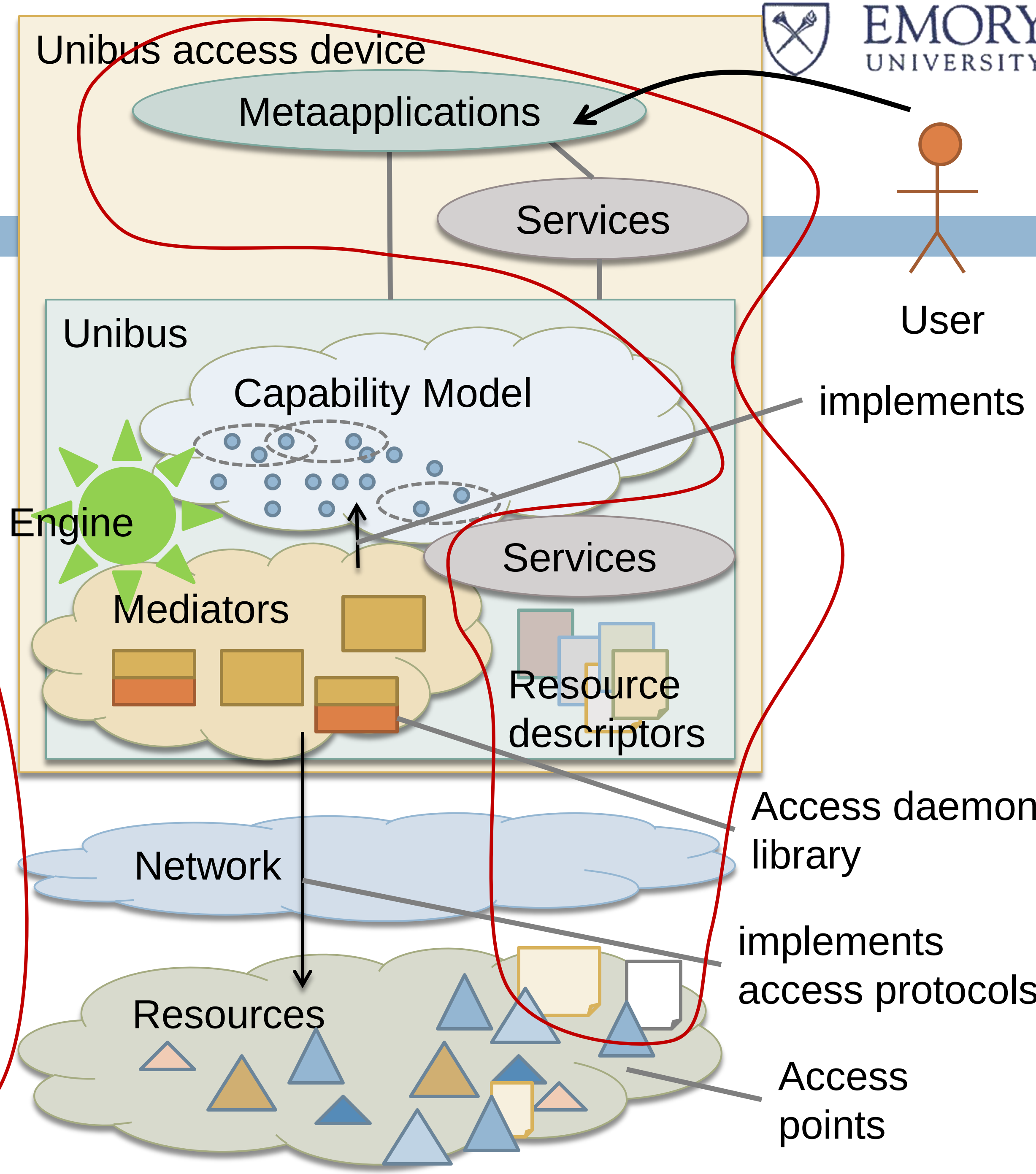
Handling heterogeneity in Unibus

- Resources exposed in an arbitrary manner as *access points*
- Capability Model to abstract operations available on provider's resources
- Mediators to implement the specifics of access points
- Knowledge engine to infer relevant facts

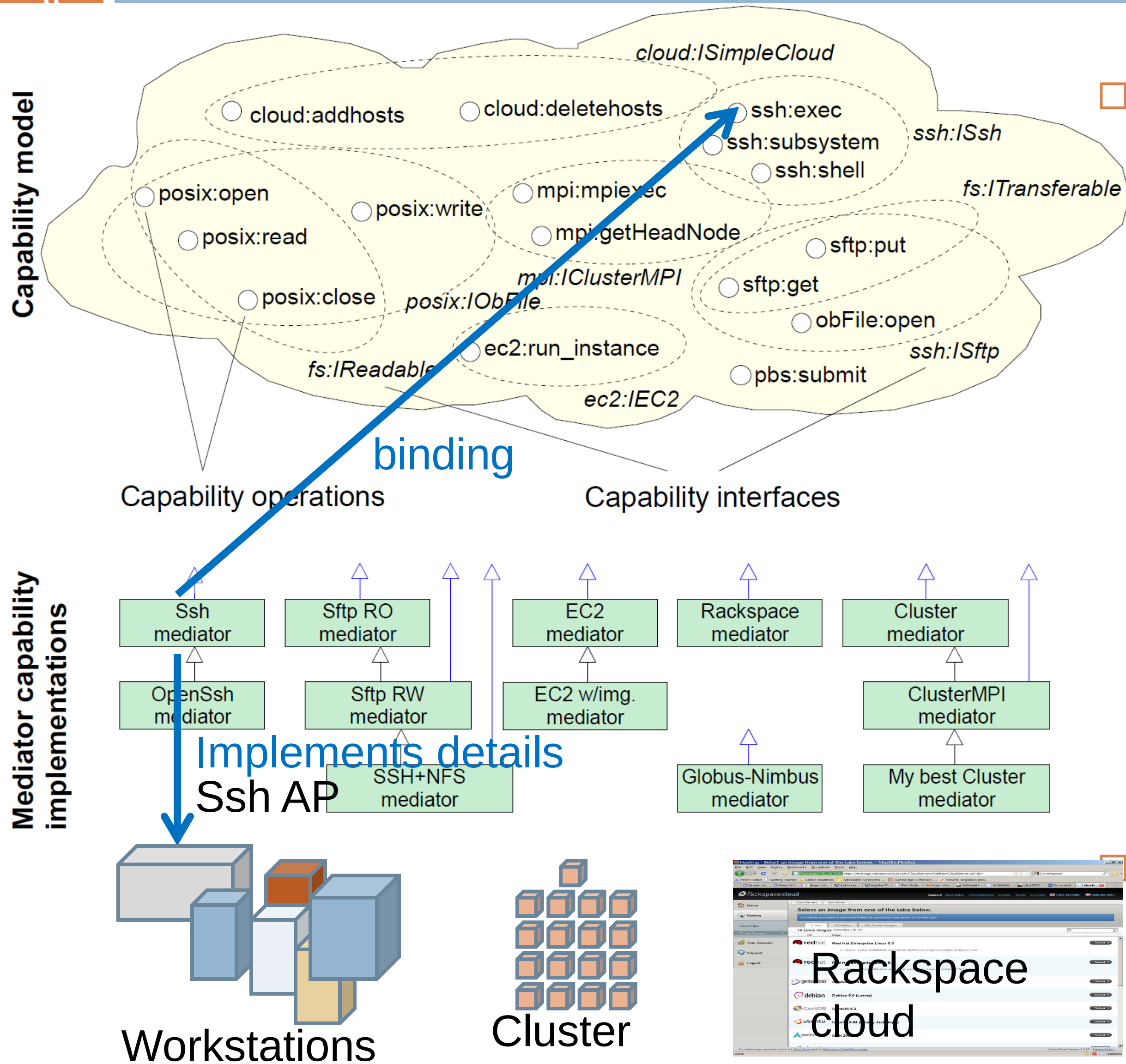


Complicating a big picture …

- Resources exposed in an arbitrary manner as *access points*
- Capability Model to abstract operations on resources
- Mediators to implement the specifics of access points
- Knowledge engine to infer relevant fact
- Resource descriptors to describe resources *semantically* (OWL-DL)
- Services (standard and third parties), e.g., heartbeat, checkpoint, resource discovery, etc.
- Metaapplications to orchestrate execution of applications on relevant resources

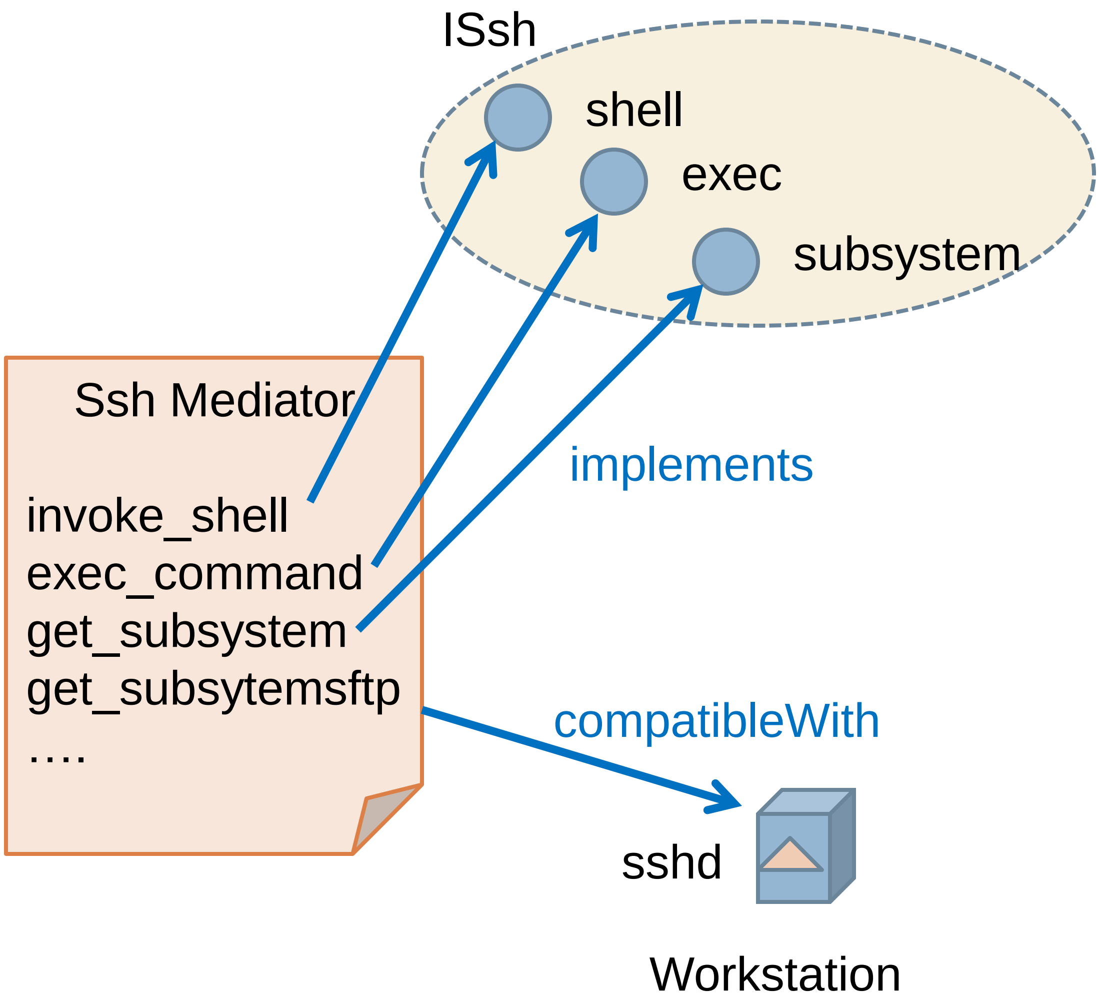


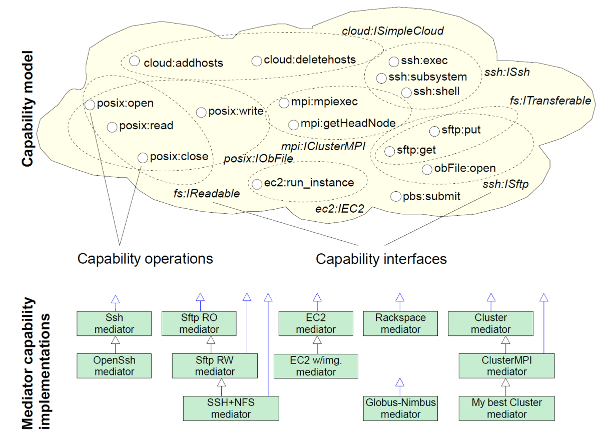
Virtualizing access to resources Capability Model and mediators

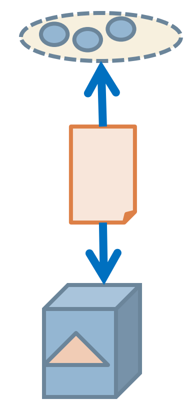


- Capability Model
  - Provides virtually homogenized access to heterogeneous resources
  - Specifies *abstract operations*, grouped in *interfaces*
  - Interface hierarchy not appropriate (e.g. fs:ITransferable and ssh:ISftp)
- Mediators
  - Implement resource access point protocols

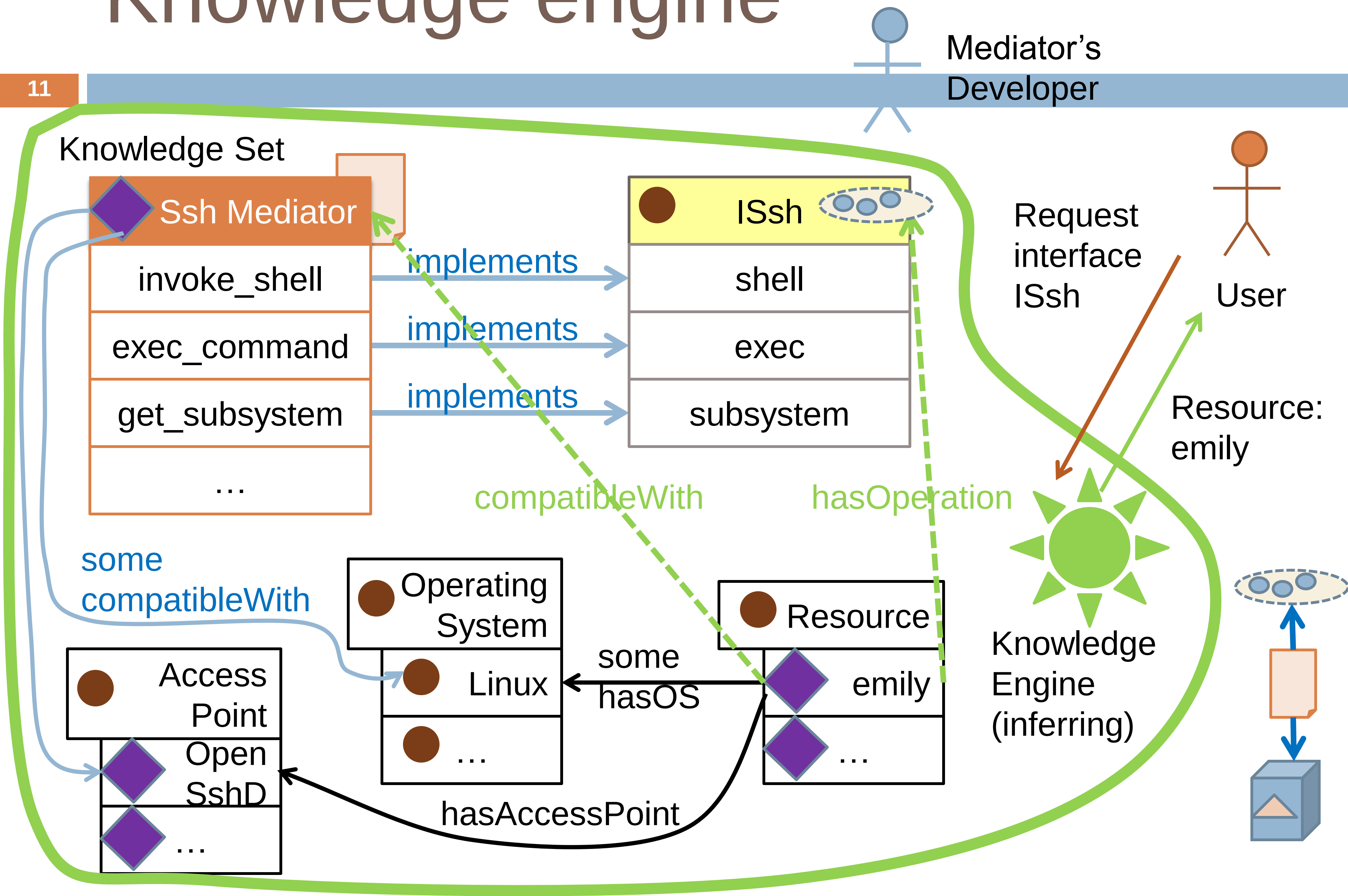
Virtualizing access to resources





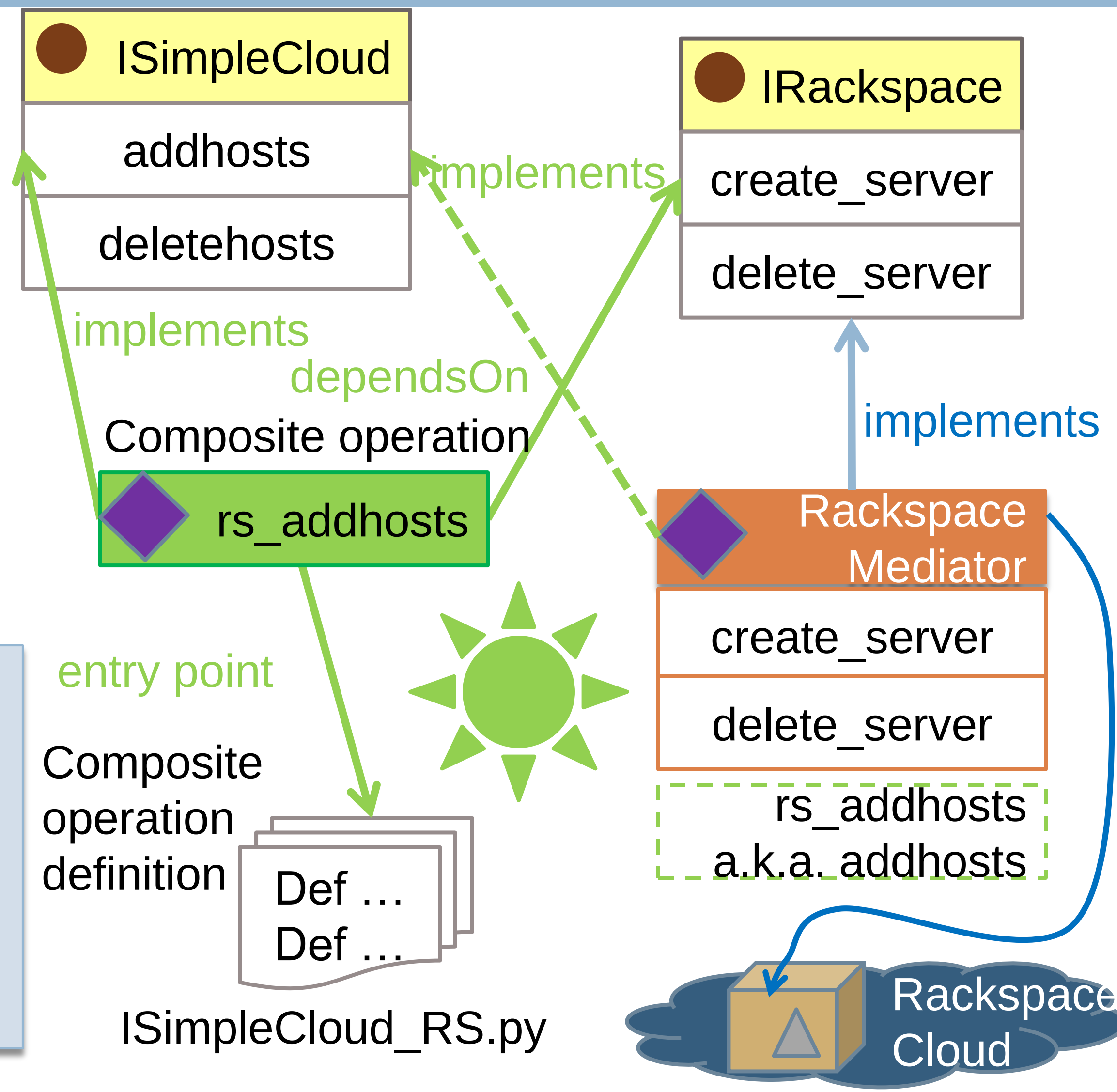


Knowledge engine



Composite operations

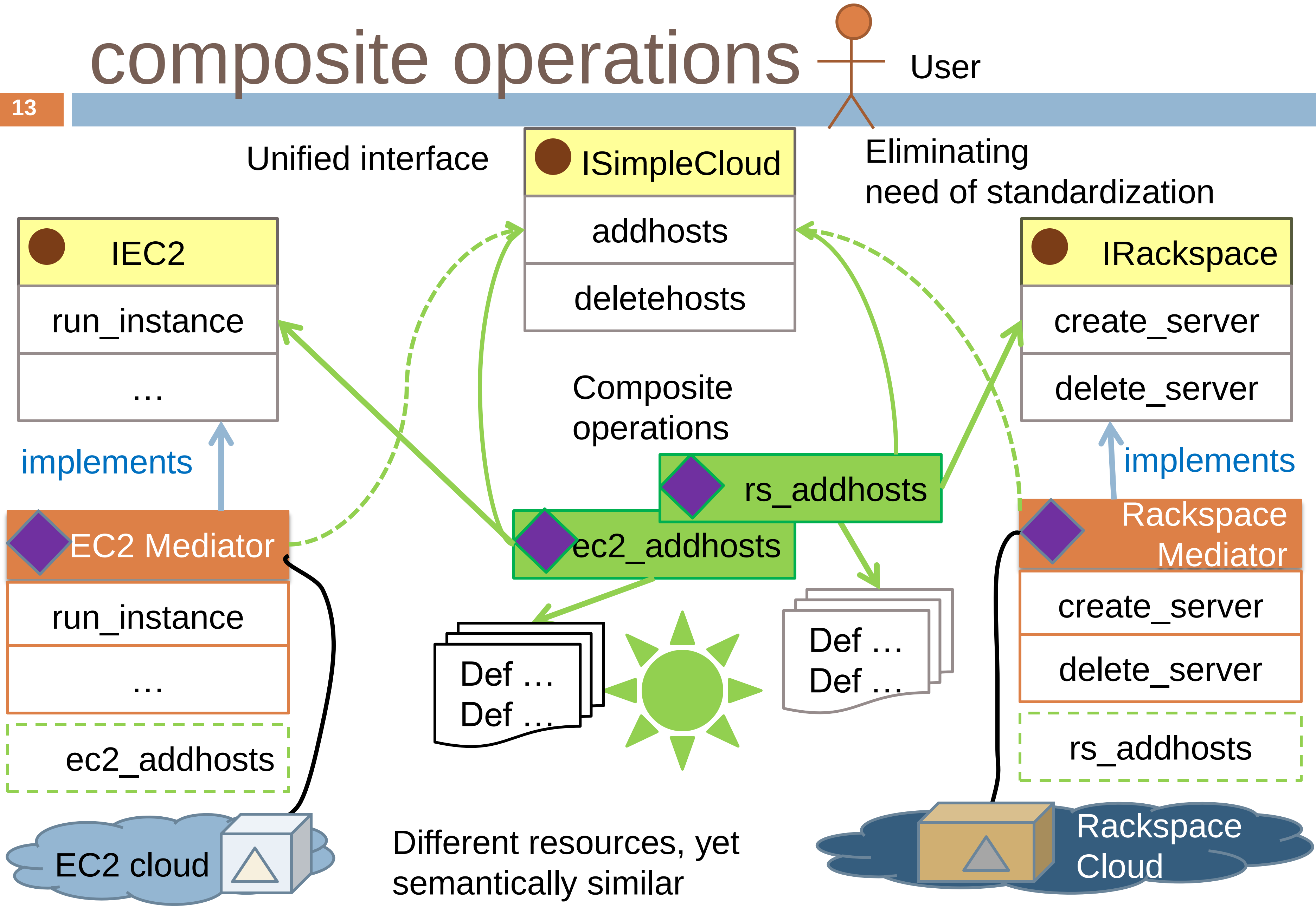
- Rs_addhosts dependsOn create_server
- Create_server is implemented by RS Mediator
- Rs_addhosts implements addhosts
- So RS mediator implements addhosts



Composite operations

- Dynamically expand mediator's operations
- May result in classification of mediators and compatible resources to new interfaces

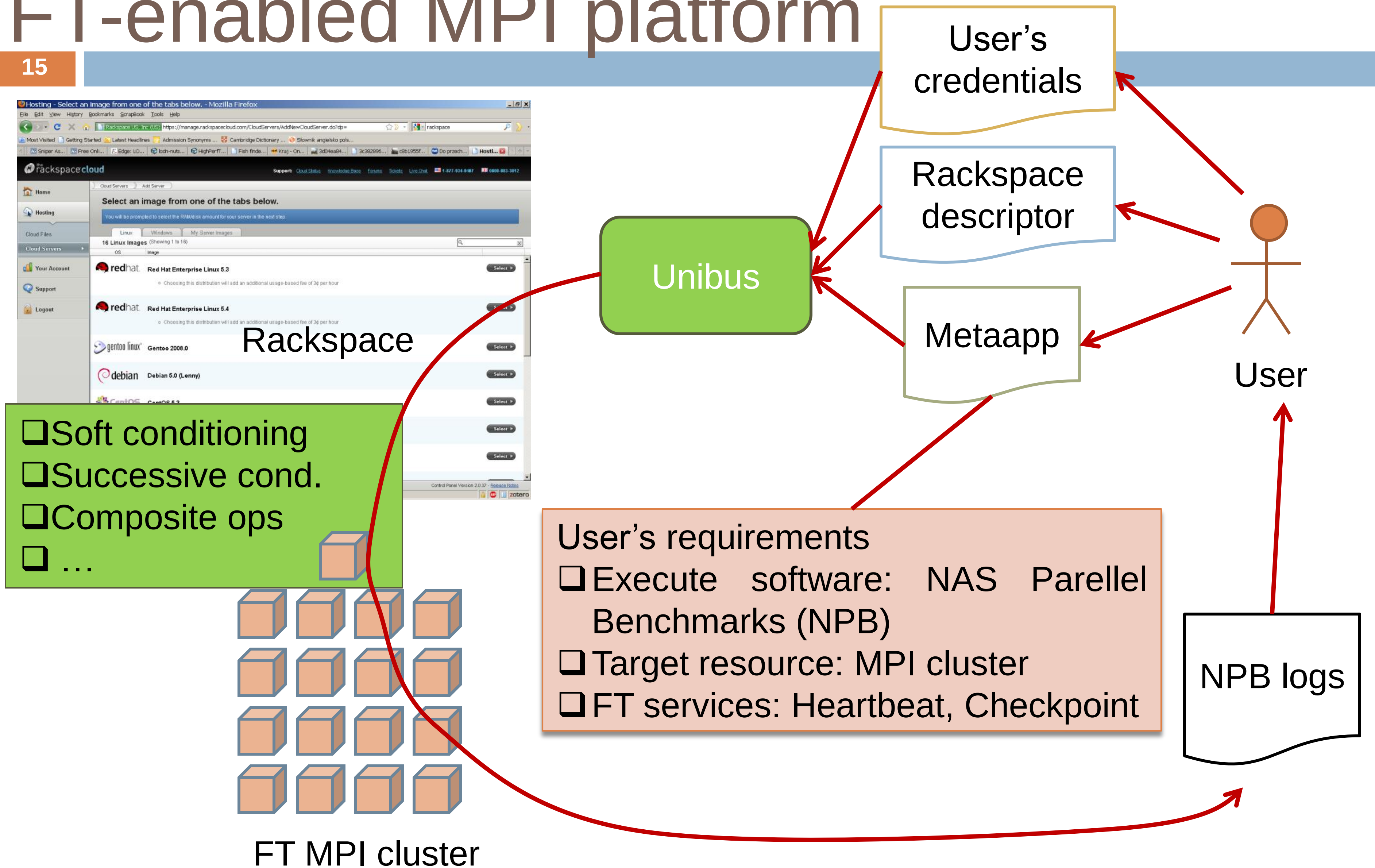
Resource access unification via composite operations



Resource provisioning
Homogenizing resource heterogeneity

- Conditioning increases resource specialization levels
  - **Soft conditioning**
    - changes resource **software capabilities**
    - e.g., installing MPI enables execution of MPI apps
  - **Successive conditioning**
    - enhances resource capabilities in terms of available **access points** (may use soft conditioning)
    - e.g., deploying Globus Toolkit makes the resource accessible via Grid protocols

Transforming Rackspace to FT-enabled MPI platform



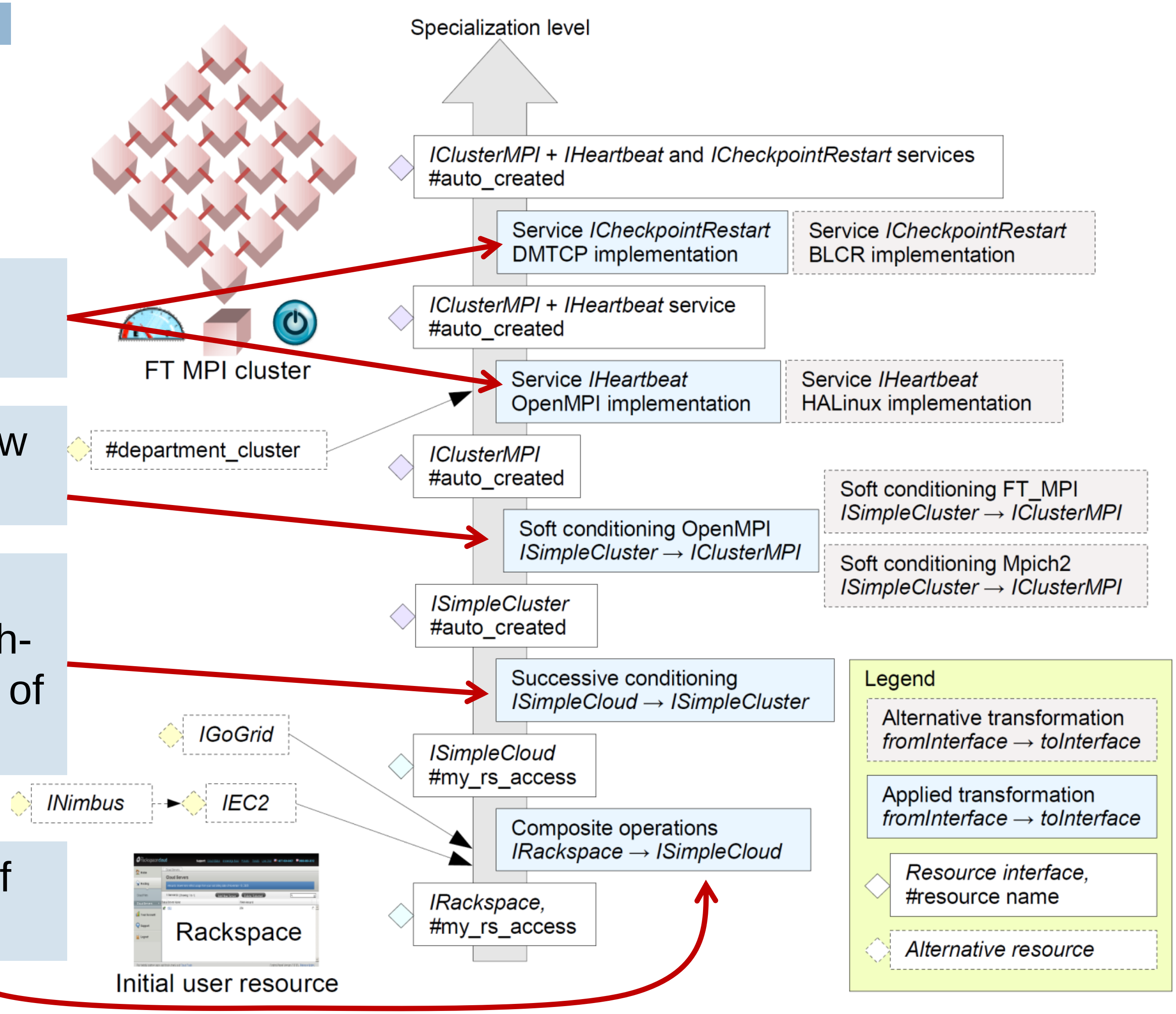
Rackspace Cloud to MPI cluster

Installing other services (FT)

Deployment of MPI on new resources

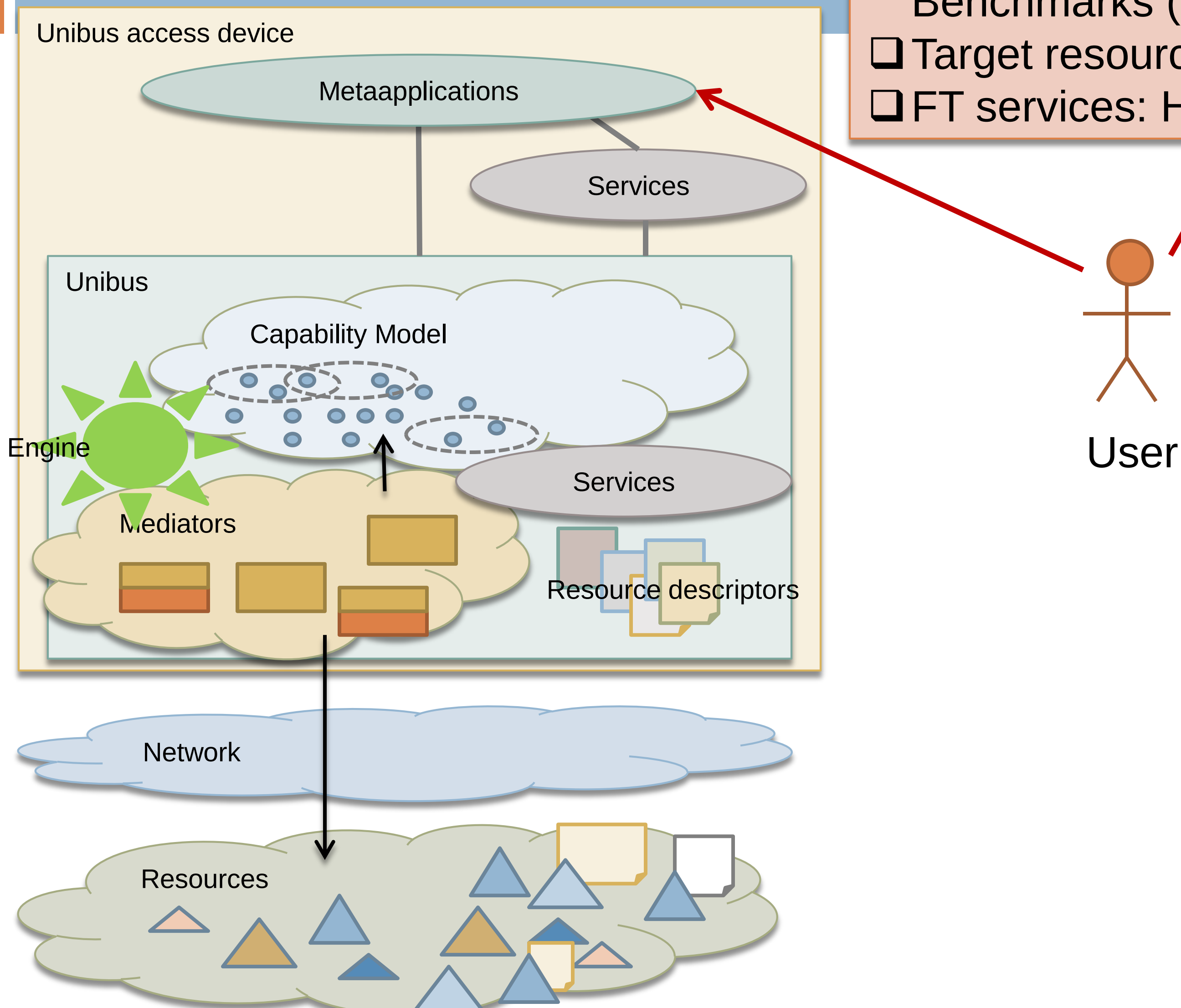
Creating a new group of resources (Rackspace ssh-enabled servers) in terms of new access points

Obtaining a higher level of abstraction



Metaapplications

User's requirements
- Execute software: NAS Parellel Benchmarks (NPB)
- Target resource: MPI cluster
- FT services: Heartbeat, Checkpoint



Metaapplication

Metaapplication

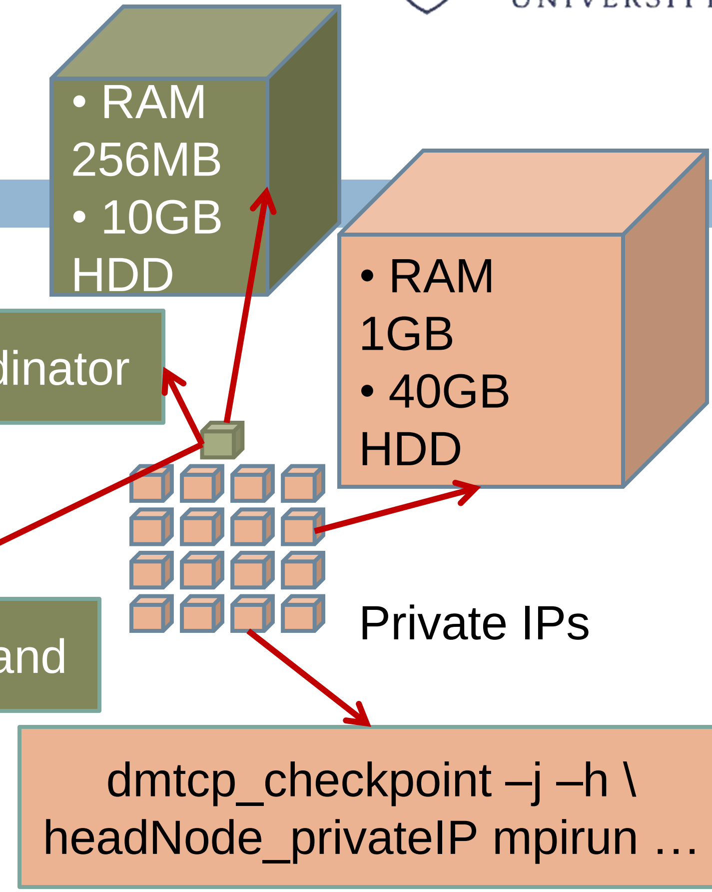
- Requests
  - IClusterMPI
  - FT services:
    - IHeartbeat
    - ICheckpointRestart
- Specifies available resources
- Performs benchmarks
- Transfers benchmarks execution logs to the head node
  - Requests ISftp

```
def create_cluster(resource):
  proxy = resource.createProxy('IClusterMPI', services=('#IHeartbeat',
    '#ICheckpointRestart'), cpus/4) #4 core processors
  #install NPB3.3 and dependencies (gcc, gfortran, OpenMPI)
  #checkpoint every 1 minutes, heartbeat every 10 seconds
  proxy.services['#IHeartbeat'].init(interval = 10, death_after = 2)
  proxy.services['#ICheckpointRestart'].init(interval = 1*60)

import_resources('file:resources.owl')
#the resource needs to be MPI cluster and have some features
res = get_resource('a cluster:IClusterMPI;\
  uc:compatibleWithService uc:ICheckpointRestart, uc:IHeartbeat;\
  cluster:cpuCores ?n . filter(?n >= %d)' % cpus)
proxy = create_cluster(res)

tests = BenchmarkTestsIterator()
for test in tests:
  test_file = '.'.join(test, class_, cpus)
  cmd = ''.join(path, test_file, '>', log_path)
  try:
    proxy['#mpiexec'](cmd, cpus)
  except proxy.services['#IHeartbeat']['#Exception']():
    # is any chkp files available (to restart)?
    if proxy.services['#ICheckpointRestart']['#get']():
      while True:
        new_proxy = create_cluster(res)
        ckpt = proxy.services['#ICheckpointRestart']['#get']()
        new_proxy.services['#ICheckpointRestart']['#set'](ckpt)
        #remove old, failed cluster
        proxy = new_proxy
        try:
            proxy.services['#ICheckpointRestart']['#restart']()
        except proxy.services['#IHeartbeat']['#Exception']():
            continue
        else:
            break
    #no ckpt available yet -- just restart the same test
    else:
        tests.repeat()
  # test done - save the result log locally
  with open('_'.join((test_file, time.strftime('%y%m%d%H%M%S'))), 'w') as f:
    #get the ISftp interface to the head node
    head = proxy['#get_head_node']().createProxy('ISftp')
    f.write(head['#open'](log_path).read())
```

EMORY UNIVERSITY

Rackspace testbed

- 16 working nodes (WN) + 1 head node (HN)
- Node: 4-core, 64-bit, AMD 2GHz
- Debian 5.0 (Lenny)
- OpenMPI v. 1.3.4 (GNU suite v. 4.3.2 (gcc, gfortran)
- NAS Parallel Benchmarks v.3.3, class B



FT setup:

Heartbeat service:

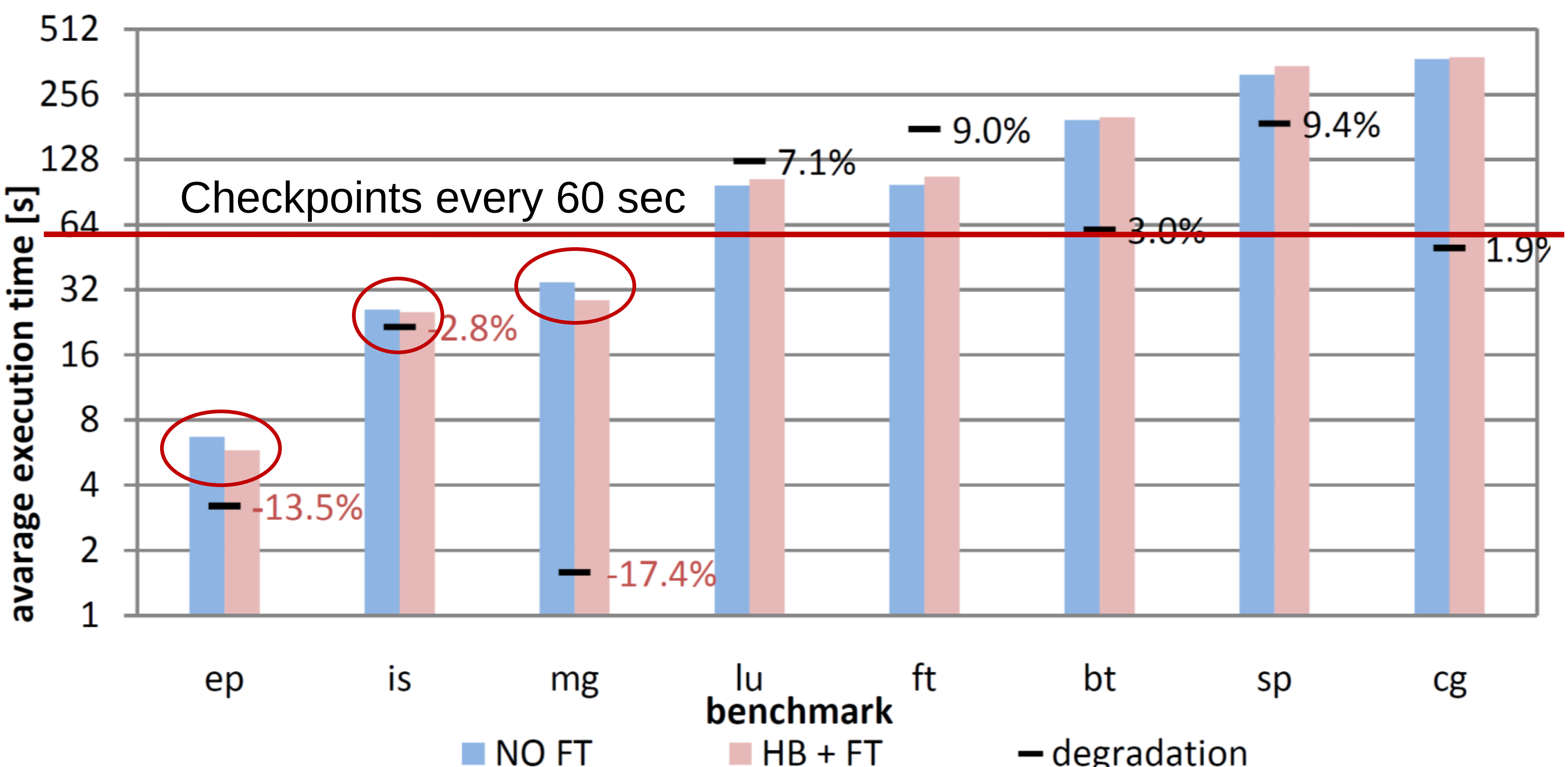
- OpenMPI-based – in case of failure, the service determines failes node(s) and raises an exception

Checkpoint/restart service:

- DMTCP – Distributed MultiThreaded CheckPointing
  - user-level transparent checkpointing
- Executes dmtcp_command every 60 secs on HN to checkpoint 81 processes (64 MPI processes, 16+1 OpenMPI supervisor processes)
- Moves local checkpoint files from WN to HN (in parallel)
- Checkpoint time – 5 sec; moving checkpoints from WN -> HN less than10 sec; compressed checkpoint size c.a.1GB

Results: NPB, class B, Rackspace, DMTCP, OpenMPI Heartbeat

16 Worker Nodes (WN) + 1 Head Node
WN: 4-core, 64-bit, AMD Opteron 2GH, 1GB RAM, 40 GB HDD
Checkpoints every 60 sec, average of 8 series



FT overhead 2% - 10%

HB - Heartbeat

Summary

- The Unibus infrastructure framework
  - Virtualization of access to various resources
  - Automatic resource provisioning
- Innovatively used to assemble an FT MPI execution platform on cloud resources
  - Reduces effort to bare minimum (servers instantiation, etc)
    - 15-20 min from 1 man-hour
  - Observed FT overhead 2%-10% (expected at least 8%)
- Future work
  - Migration and restart of MPI-based computations on two different clouds or a cloud and a local cluster
  - Work with an MPI application