CS 524 A

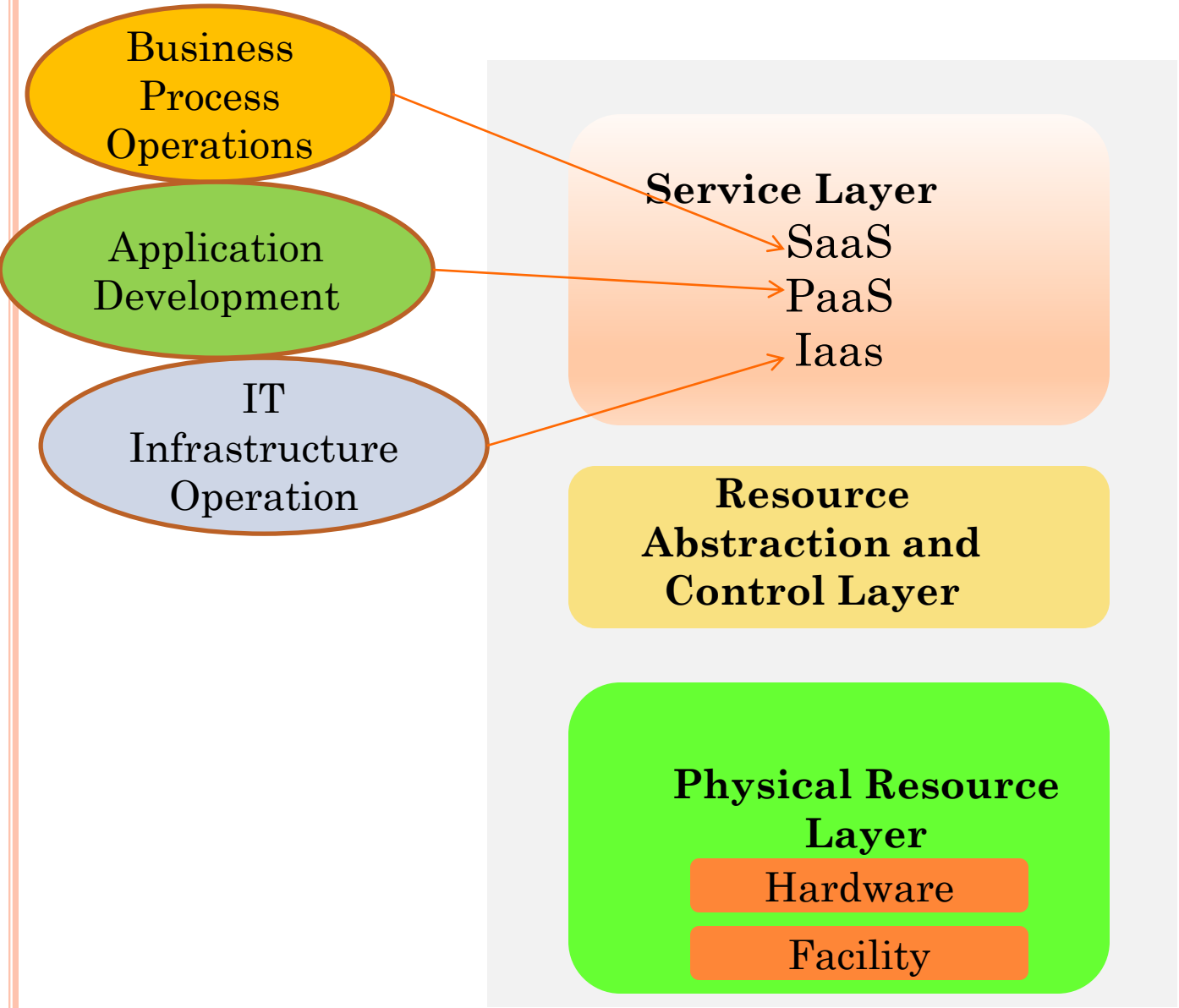
Introduction to Cloud Computing

Lecture 11:
Operations, Management, and Orchestration in the Cloud

Outline

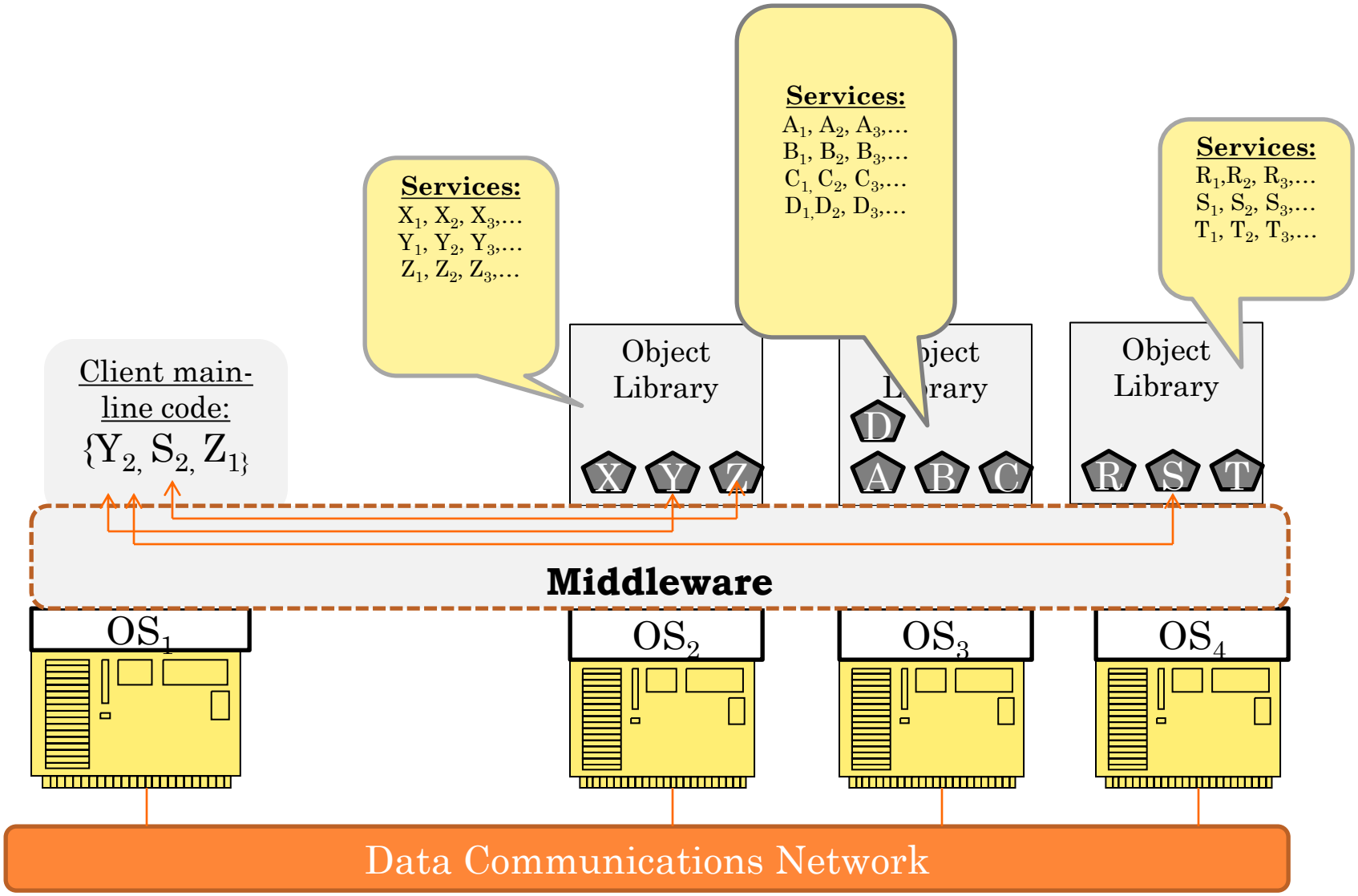
- The NIST Reference Architecture
- Workflows
  - Examples (logic design, data flow machines)
  - Basic structure
  - Optimization
  - References
- Network Management
  - The Model
  - CMIP and SNMP
  - Policy Management
  - NETCONF
- Cloud Service Lifecycle
  - The Model
  - Amazon Cloud Formation as an automation mechanism for the lifecycle
  - *OpenStack* architecture in support of the Cloud Service Lifecycle
  - OASIS *TOSCA*

Service Orchestration (after NIST SP 500-292): The Cloud Provider view



Service Management

- NIST description underlines the distinction between *service orchestration* and *service management* tasks performed by a Cloud provider.
- Service management includes "all of the service-related functions that are necessary for the management and operation of those services required by or proposed to Cloud consumers."
- The three service management categories are *business support*, *provisioning and configuration*, and *portability and interoperability*.
  - In the **business support** category are the tasks of
    - customer-, contract-, and inventory management
    - accounting and billing,
    - reporting and auditing, and
    - pricing and rating.
  - The actual **operation** of the Cloud is the subject of the **provisioning and configuration** category, whose tasks include
    - Provisioning
    - Resource change,
    - Monitoring and reporting,
    - metering, and SLA management.
  - The tasks of **portability and interoperability:**
    - data transfer,
    - VM image migration and
    - application- and service migration

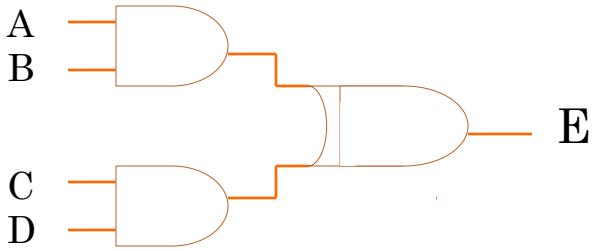


Distributed object-oriented computing model

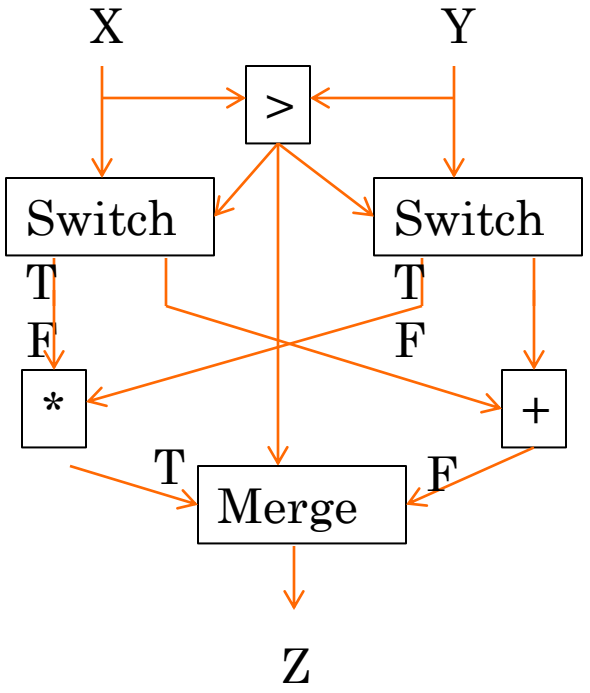
Workflows

- In describing a task one lists all the activities that are involved in carrying the task to a completion.
- Some of these activities may run in parallel;
- Others need to wait for the completion of prerequisite activities.

  A workflow is a specification that defines and orders all the activities within a task.
- To automate a task involving a distributed system, its workflow must be defined in a way that it is executable in a distributed environment.

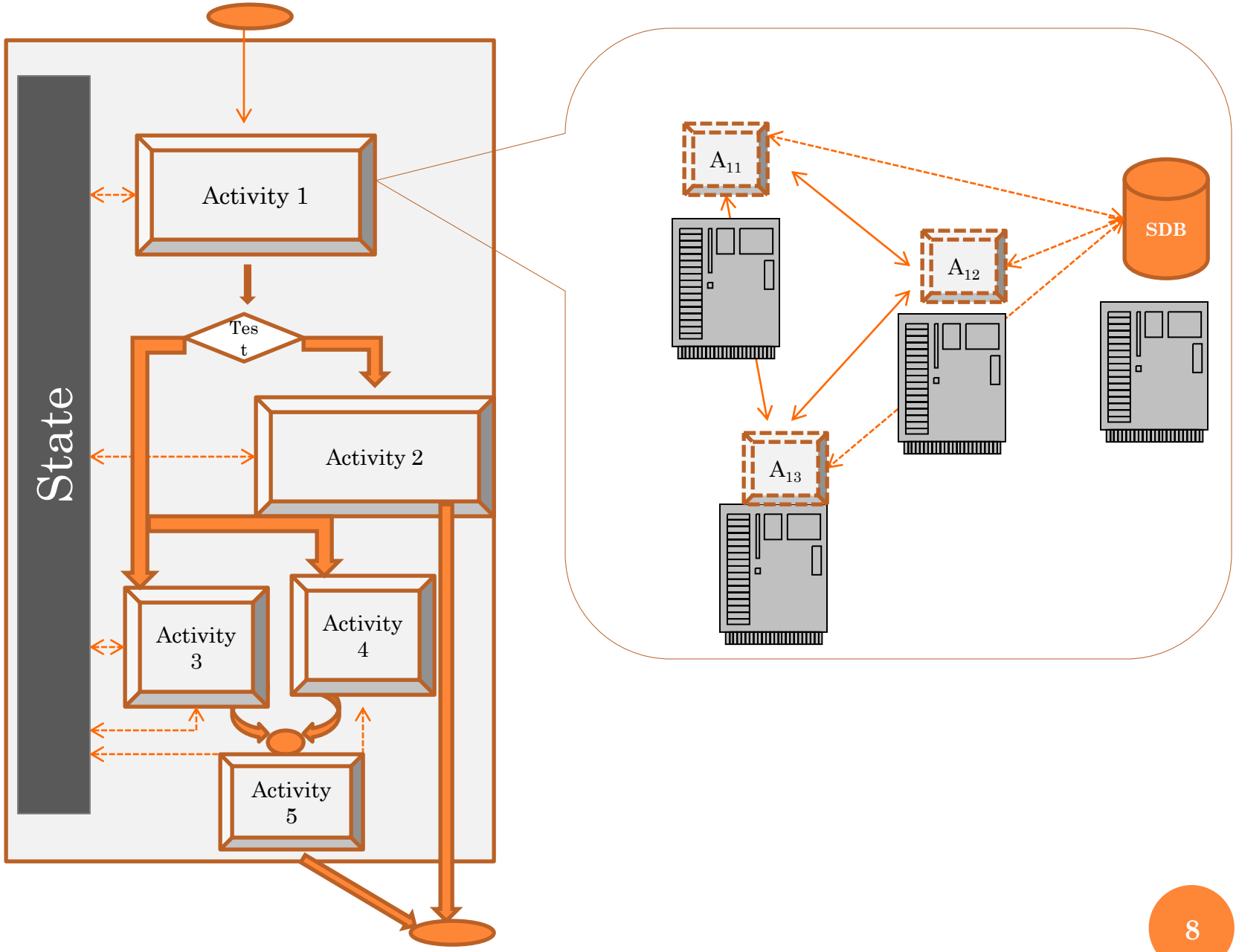


a) A logic design circuit for computing E = (A ∧B) V (C ∧ D)

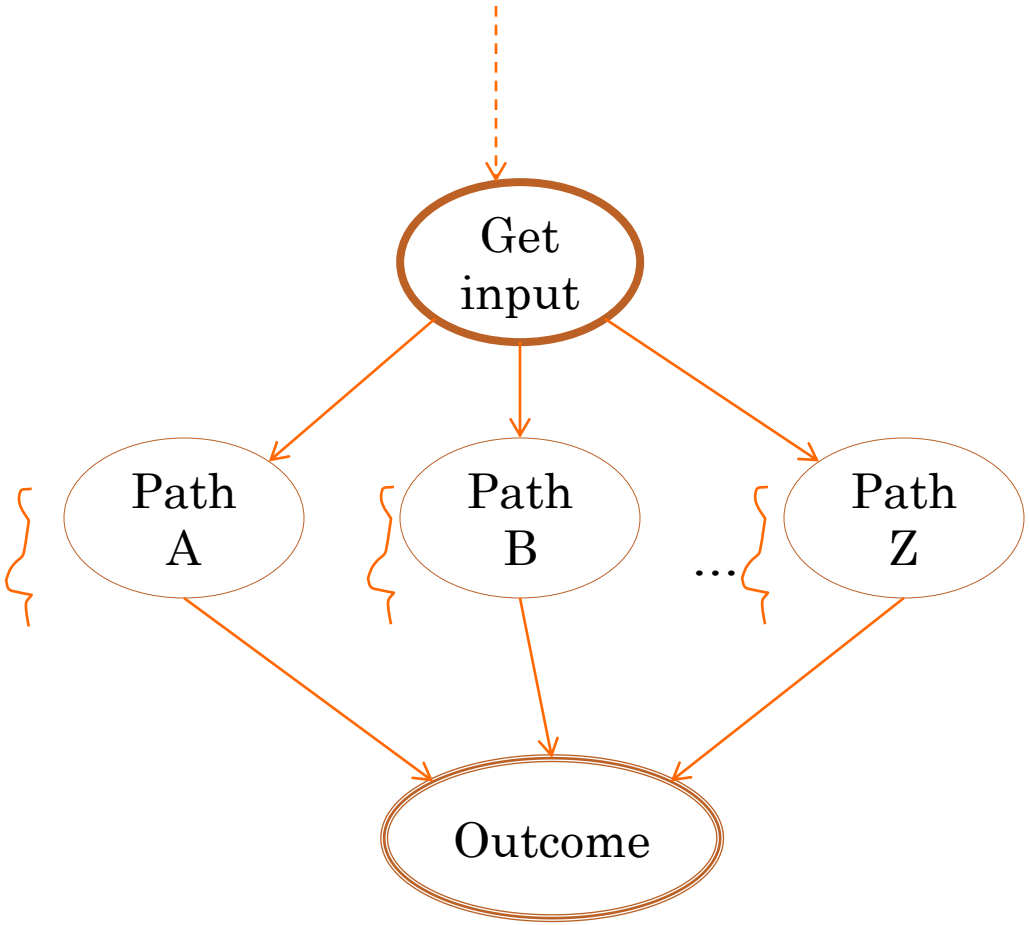


b) A data-flow machine for computing
Z = *if* (X > Y) *then* (X*Y) else (X+Y)

Flow-based computing examples



Workflow as a directed graph of activities



Optimization: Path analysis

Some References

- Microsoft Windows Workflow http://msdn.microsoft.com/en-us/library/dd851337.aspx
- *Amazon Simple Workflow Service (AWS)* API along with *the AWS Flow Framework (extra-credit assignment)*

http://aws.amazon.com/swf

http://aws.amazon.com/code/2535278400103493

- Amin, K., M. Hategan, G. von Laszewski, N. Zaluzec, S. Hampton, and A. Rossi. (2004) *GridAnt: A Client-Controllable Grid Workflow System*, in 37th Hawai'i International Conference on System Science, Island of Hawaii, Big Island. pp 210-220. (Also available in an Argonne National Laboratory preprint ANL/MCS-P1098-1003 at www.mcs.anl.gov/papers/P109Apdf.)
- Kalenkova, A. (2012) *An algorithm of automatic workflow optimization*. Programming and Computer Software, Vol. 38, No. 1. (January 2012), pp. 43-56. Springer-Verlag.

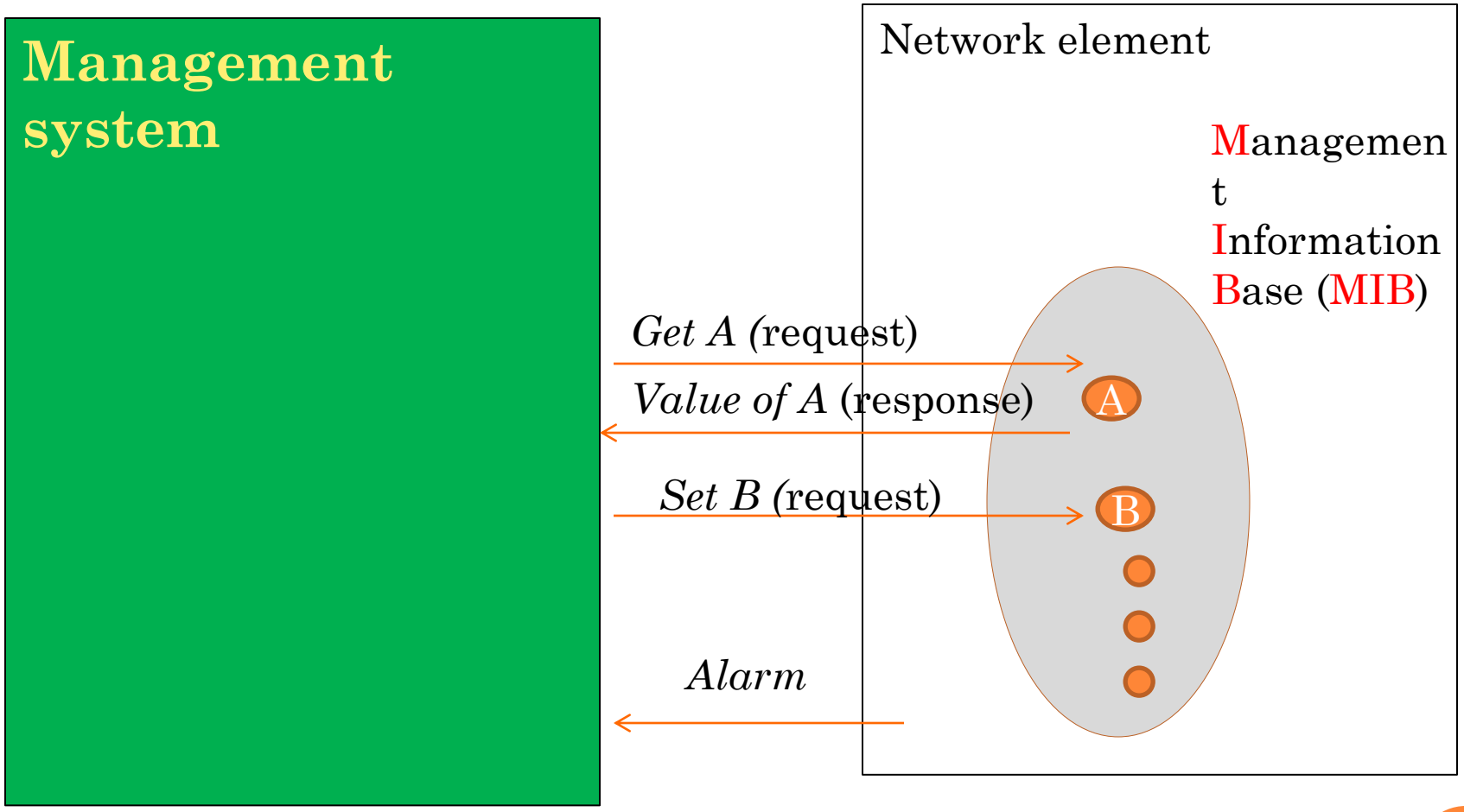
10

The OSI Network Management Framework: Five aspects

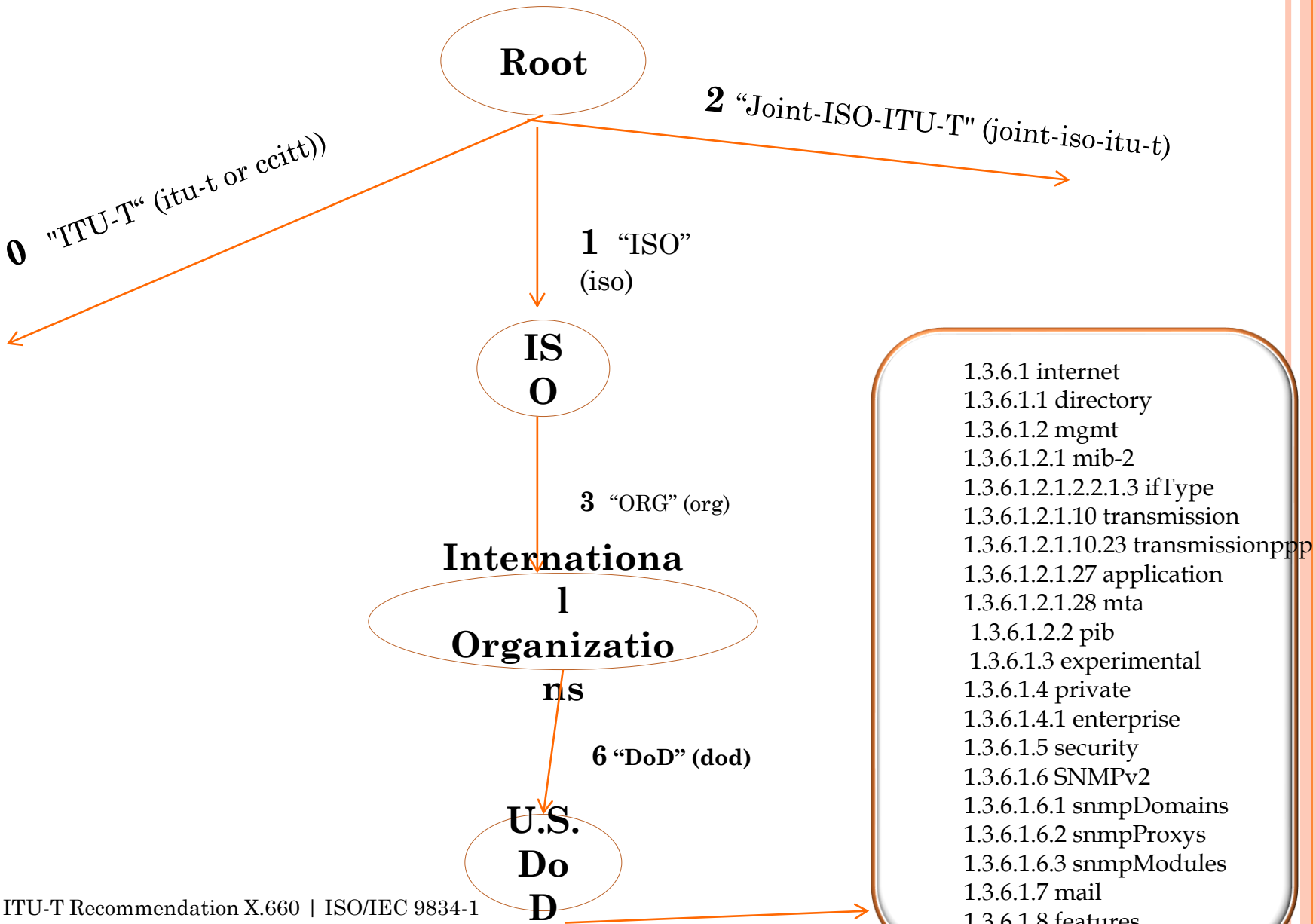
- **Configuration management**
  - Change (*set)* values of *settable* parameters
  - Read (*get*) values of read-only parameters
- **Fault management**
  - Respond to any abnormal condition
  - Log abnormal events
  - Report abnormal conditions through an *alarm* mechanism
- **Performance management**
  - Measure utilization of the resources
  - Plan capacity (long-term)
  - Performance management
- **Identity and Access management**
  - Authentication and Authorization for access to network resources
- **Accounting Management**

11

The OSI network management model



The tree of the SMI ASN. 1 Object Identifiers

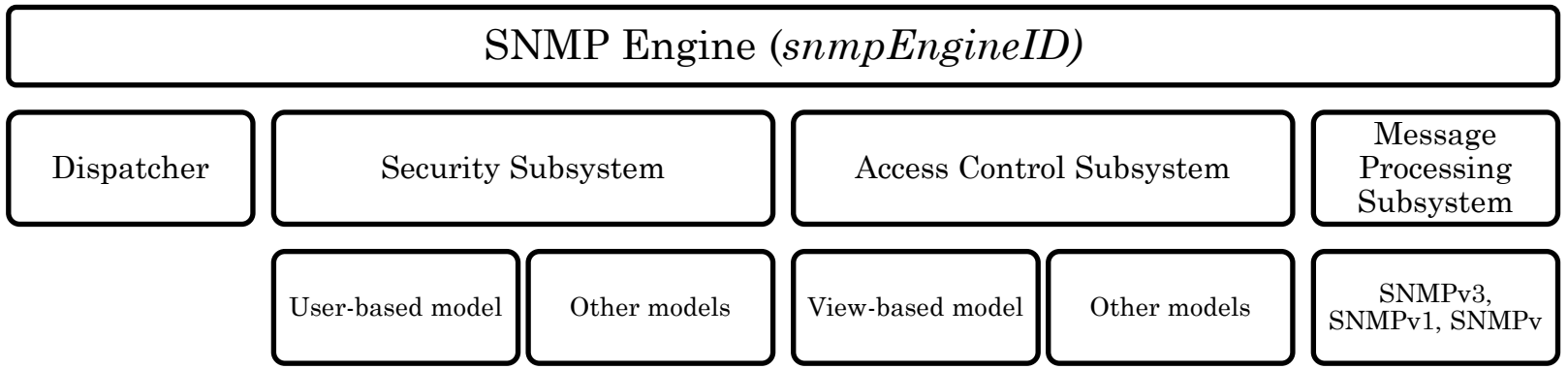


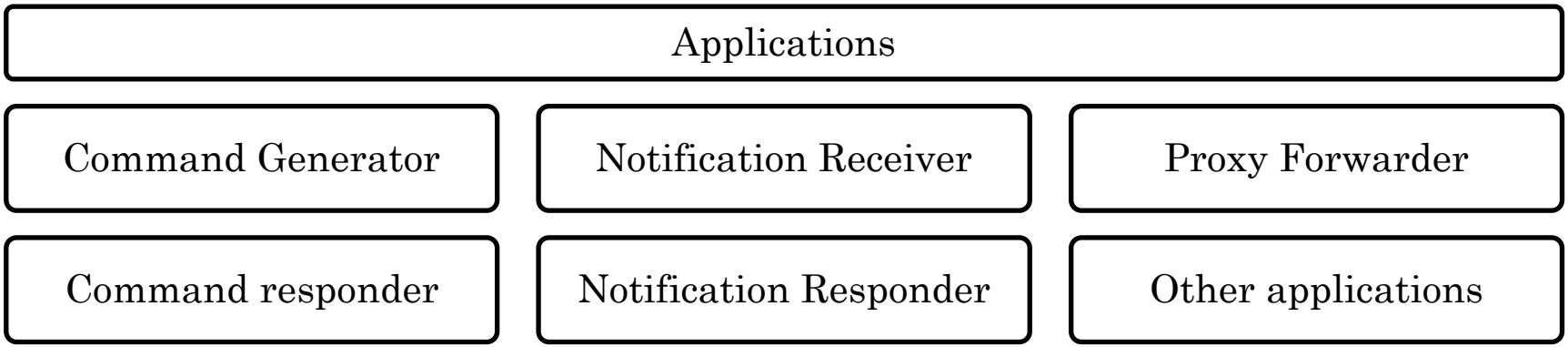
Sources: ITU-T Recommendation X.660 | ISO/IEC 9834-1
IETF RFC 1155
http://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml

CMIP AND SNMP

- ITU has developed, jointly with ISO/IEC JTC 1, the *Common Management Information Protocol (CMIP)*
- Since CMIP was using the OSI Application Layer services (such as the OSI *Remote Operations Service Element*) that were unavailable in the Internet, the IETF had decided to proceed with its own protocol, and here the development of the network management standards has forked.
- Based on CMIP and other modules, ITU-T has come up with a large set of specifications (the M.3000 series) called the *Telecommunications Management Network (TMN)*, while the IETF produced the so called *Simple Network Management Protocol (SNMP)*, now in its third version *SNMPv3*.

SNMP Entity (after RFC 3411)





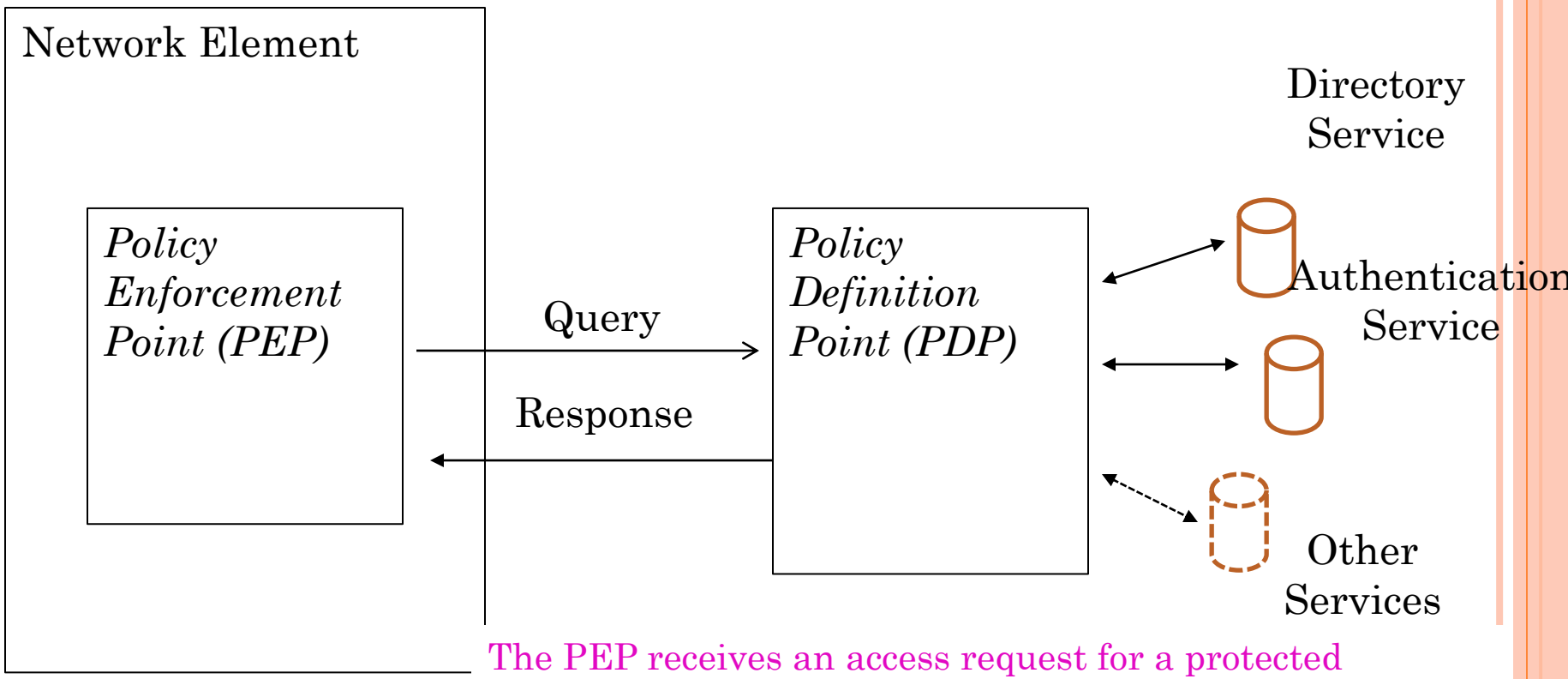
The IETF Policy Control Framework

- **The initial objective**: to describe the execution of policy-based control over the QoS admission control decisions.
- Later the framework expanded to include the whole range of other resources under operator's control: private IP addresses, port numbers, NAT pinholes, etc.

- **Goals:** support for preemption (the ability to remove a previously granted resource so as to accommodate a new request), various policy styles, monitoring and accounting. Policy styles:
  - Bi-lateral and multi-lateral service agreements
  - Relative priority, as defined by the provider.
  - Monitoring and accounting
  - Fault tolerance and recovery in cases when a PDP fails or cannot be reached.

16

Policy Control Architecture (after RFC 2753)



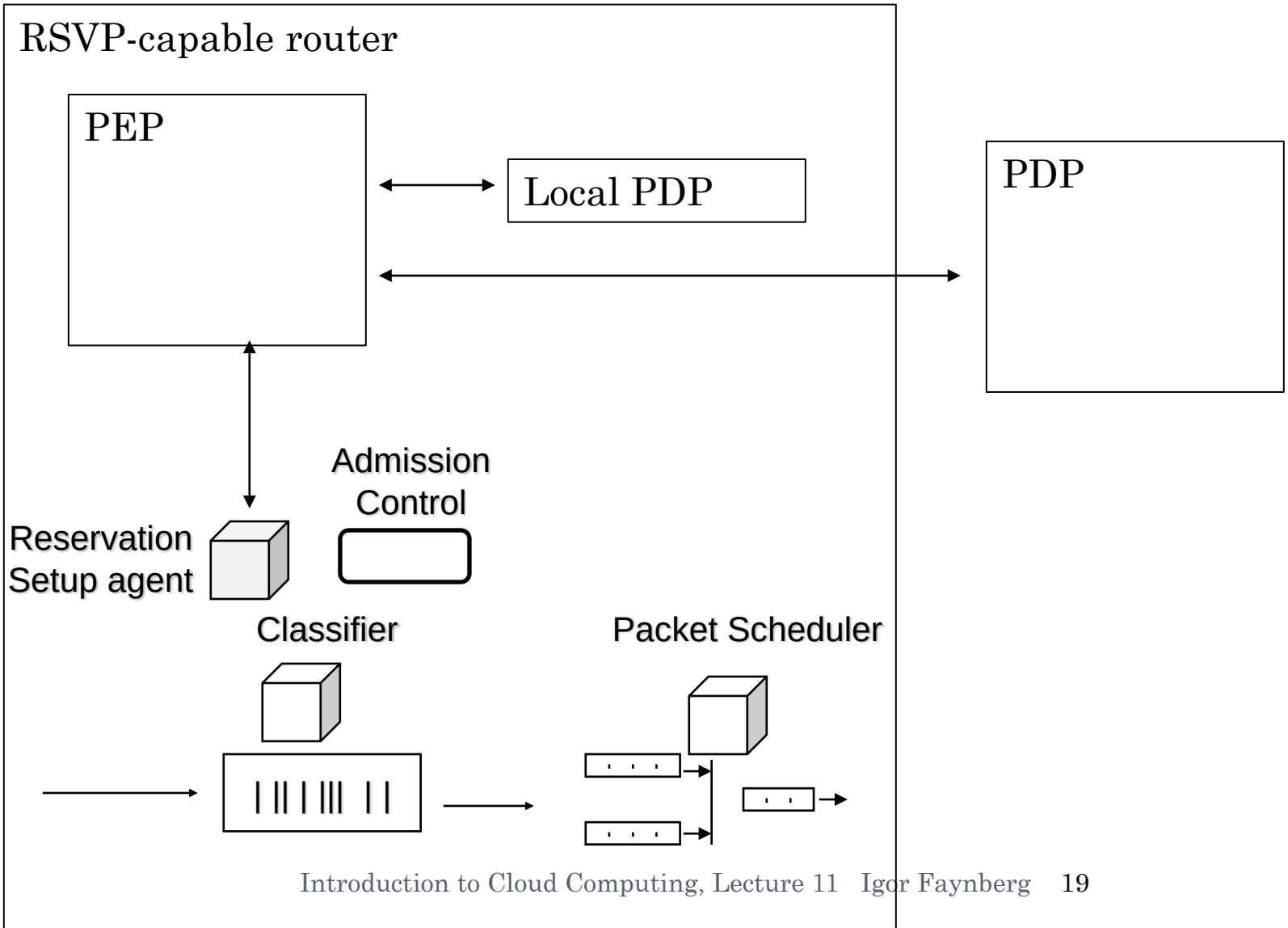
The PEP receives an access request for a protected resource (or an object).
The PEP passes the request to the PDP.
The PDP fetches the applicable policy from the policy repository.
The PDP, upon making the access decision, returns the result to the PEP.

COPS PROTOCOL

- An intrinsically new feature of COPS—as compared with SNMP or the ISO CMIP—is that COPS employs a *stateful* client-server model, which is different from that of the RPC.
- The PEP (client) sends *requests* to the remote PDP (server), and the PDP responds with the *decisions*. But all the requests from the client PEP are *installed* and remembered by the remote PDP until they are explicitly deleted by the PEP.
- The *decisions* can come in the form of a series of notifications to a single request. This introduces a **new behavior**: two identical requests may result in different responses because the states of the system when the first and the second of these requests arrive may be different—depending on which states had been installed.
- Another *stateful* feature of COPS is that PDP may *push* the configuration information to the client and later remove it.

18

Policy Control in an RSVP Router (after RFC 2753)



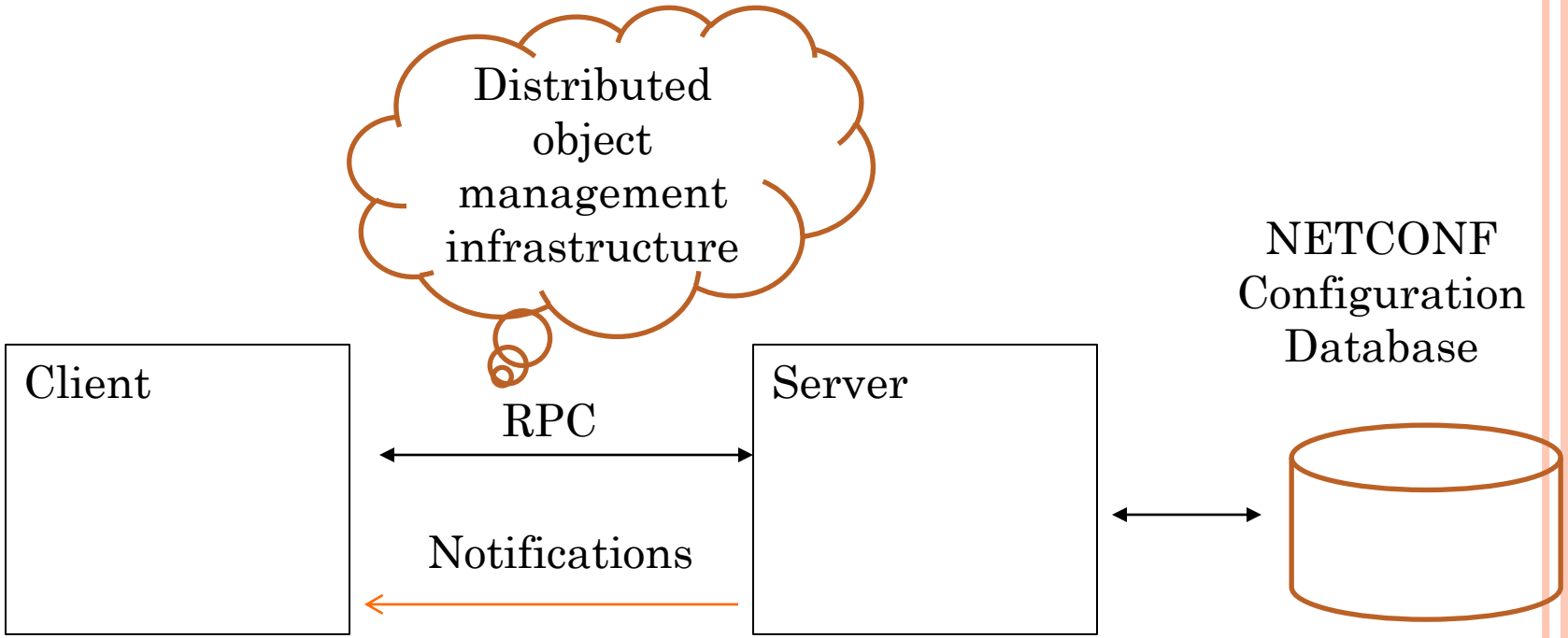
NETCONF

- Objectives:
  - Manage a common set of configurations objects (rather than peace-meal MIBs)
  - Make uniform **CLI** across communications devices from multiple vendors
  - Ditto for **HTTP API** and web-based

- Protocol basics:
  - Client issues commands
  - The server executes commands over the *configuration datastore*. "...a conceptual place to store and access information"
  - The model is RPC—the REST camp lost its case!
  - Full-blown distributed ***ACID*** database management implementation (***Atomic, Consistent, Isolated,* and *Durable***)

20

NETCONF Architecture



NETCONF Layers

Defined using YANG (RFC 6020)

Content	Configuration data	Notification data
Operations	<get>, <get-config>, <edit-config>, etc.	
Messages	<rpc>, <rpc-reply>, <rpc-error>, <ok>.	<notification>, <filter>, stream>, etc. (RFC 5277)
Secure Transport	Secured path between the client and server	

YANG

- "Yin and Yang (陰-陽)"—the two dialectically-opposite but inter-related concepts in Chinese philosophy (first introduced in writing around 700 BC in I. Chin's *Book of Changes*.)
- RFC 6020 specifies YIN as the XML-based representation of YANG (a data-modeling language): "YANG modules can be translated into an equivalent XML syntax called YANG Independent Notation (YIN) allowing applications using XML parsers and Extensible Stylesheet Language Transformations (XSLT) scripts to operate on the models. The conversion from YANG to YIN is lossless, so content in YIN can be round-tripped back into YANG."

23

Invocation of the the *Deck-the-halls* RPC method and Reply with the positive result

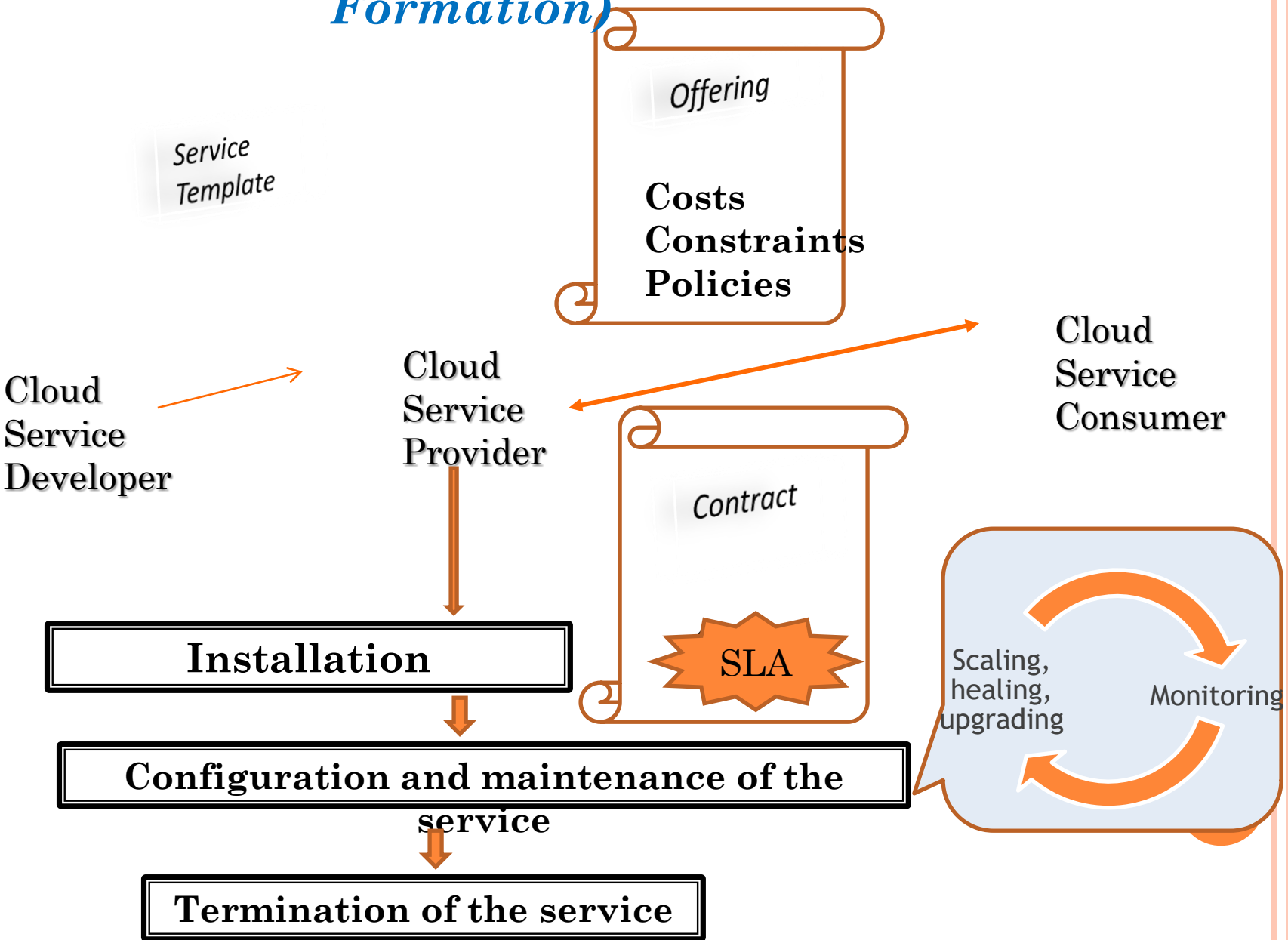
a)

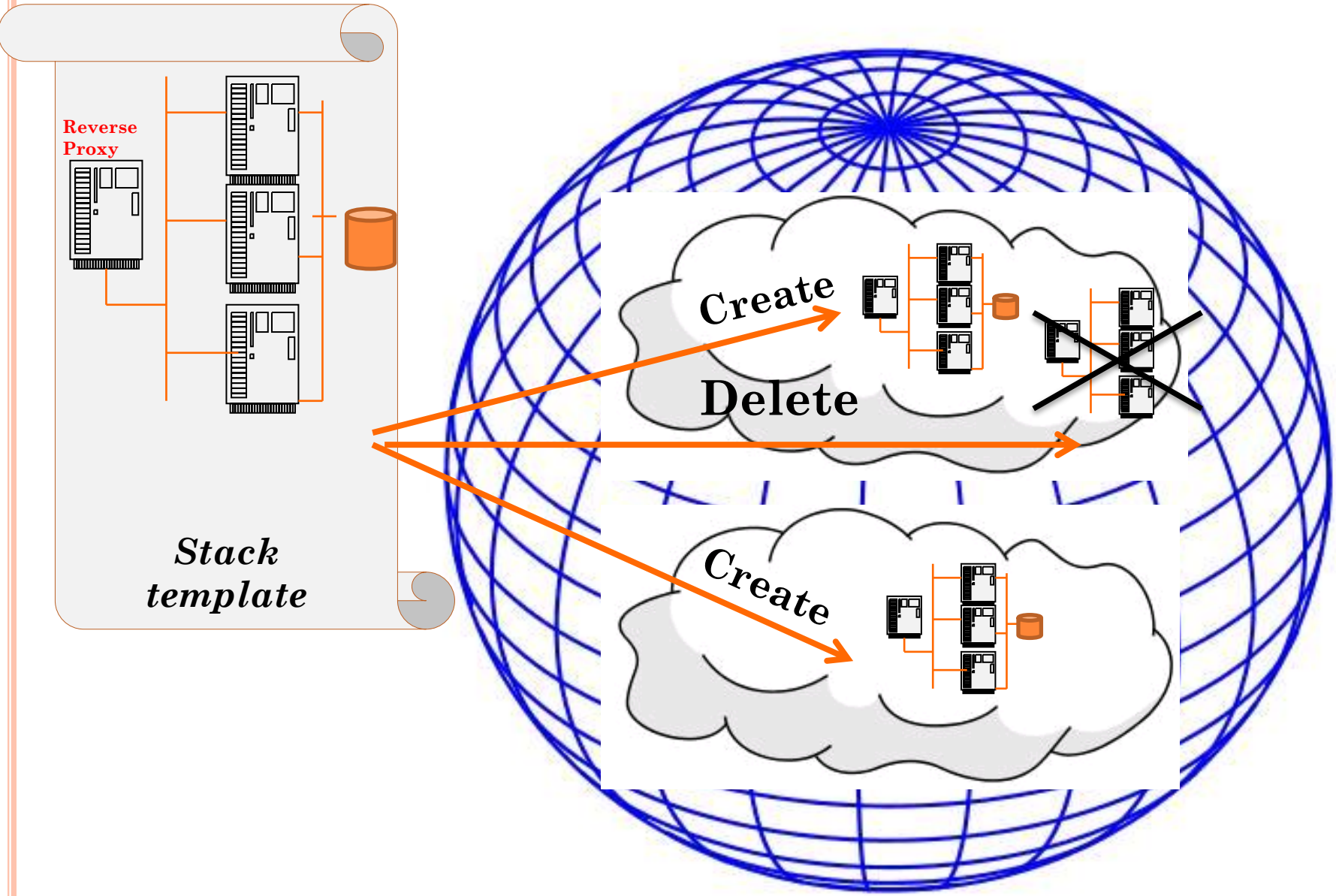
```
<rpc message-id="123"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

     <deck-the-halls
      xmlns="http://example.net/Cloud/deck-the-halls/1.0">
         <fixture boughs_of_holly>
         </fixture>
     </deck-the-halls>
  </rpc>
```

b)

```
<rpc-reply message-id="123"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
  </rpc-reply>
```

The Cloud Service Lifecycle (cf. AWS Cloud Formation)





Operations on a stack (an example)

AWS CloudFormation

- Endpoints (with known URLs) are deployed across the world regions
- The template is written in the *JavaScript Object Notation (JSON) format*

```
{
  "AWSTemplateFormatVersion" : <date>
  "Description" : <string>

 "Resources" :
     { <resource-name/type/properties list>
     },

"Parameters" :
     { <parameter list>
     },

  "Mappings" :
     { <mapping/key/value list>
     },

  "Conditions":
     { <name/intrinsic function/arguments list>
     },

 "Outputs" :
     {
     }
}
```

Mandatory

The *AWS CloudFormation* Template

Source: *AWS CloudFormation User Guide (API Version 2010-05-15)*

The AWS Template

- A *parameter* is just a name string, whose specification may list the conditions that constrain the values that parameters may take
- *Mappings* automate the parameters' value assignments. One can define a subset of parameter's values and associate it with a *key*. A typical example of a key is the name of a region; all region-specific values (e.g., current time or local regulations) are assigned to the respective parameters automatically.
- *Conditions* are Boolean functions that compare parameter values, either with one another, or with constants. If the result of a comparison is positive, resources are created. All conditions are evaluated when a stack is a) created or b) updated (and only then).
- *Outputs* are parameters declared specifically in support of the feedback mechanism. The end-user can query the value of any *output* via a *describe-stack* command. Again, conditions can be employed to guide the value assignment.

AUTO-SCALING

- Is an orchestrator-provided *service that enables elasticity*.
- In AWS auto-scaling enables
  - launching or terminating an instance according to user-defined policies as well as
  - run-time characteristics (such as application's "health" gauged through monitoring).
- Scaling can be achieved *vertically*, by changing the compute capacity of an instance; or *horizontally*, by changing the number of load-balanced instances. It is particularly the horizontal scaling that demonstrates the unique economic advantages of the Cloud environment. Compare with the physical deployment!

An example

- A template for operating the environment that involves a group of the web servers would specify under the *Resource* header a group of the *type "AWS: :AutoScaling: :AutoScalingGroup"*, with the *properties* that list
  - the availability zones;
  - the *configuration name* (another resource, pointing to the image of the instance to launch); and
  - and both the minimum and a maximum sizes of the group.
- The actual ready-to-use sample template with the superset of the capabilities described here can be found at https://s3-us-west-2.amazonaws.com/Cloudformation-templates-us-west-2/AutoScalingMultiAZWithNotifications.template.

EXAMPLE (CONT.)

- If *notifications* of the events to the operator are expected, the notification topic can be also specified as a resource with the *type "AWS: :SNS: Topic,"* which would
  - refer to the appropriate resource—the endpoint (the operator's email) and specify the protocol (*"email").*
  - In this case, the common group specification would also list specific notification message strings (e.g., "Instance launched," "instance terminated," or "error," the latter also supplying an appropriate error code)
- The scale-up and scale-down policies can be specified, using the resource type *AWS::AutoScaling::ScalingPolicy*. The actual alarm event that triggers scaling up (or down) can be specified as the resource with the type "AWS::CloudWatch::Alarm."
- Say, if the requirement is for scaling-up is a burst of CPU utlilization exceeding *80% for 5* minutes, the properties of the scaling up alarm will include, using the "WS/EC2" namespace
  - *"MetricName": CPUUtlilization,*
  - *"Period: "300",* and *"Threshold: 90."*
- The *"AlarmActions"* refer to the name of the scale-up policy defined above.
- The intrinsic function used here is "ComparisonOperator," with the value *"GraterThanThreshold."*

32

EXAMPLE (CONT.)

- Then we specify the load balancer "*AWS::ElasticLoadBalancing::LoadBalancer,"* with the properties that include
  - the port number to listen to ,
  - an instance port number, and
  - the protocol (HTTP).
- Last but not least, the instance security group of the *type "InstanceSecurityGroup"* must be created. The typical use is enabling *Secure Shell (SSH)*-based) access to the load balancer only.
- The "*Parameters"* section defines the structures referred to above: the types of instances allowed, specific port numbers, the operator's email, the key pair for the SSH access, and the (CIDR) IP address patterns.
- The *"Mappings"* section supplies the parameters' values (pre-defined in AWS)
- The *"Outputs"* lists the only output—the URL of the website provided by the server.

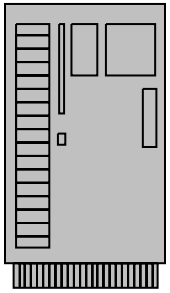
EXAMPLE (CONT.)

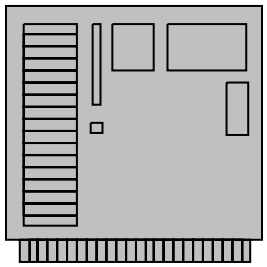
- Obtaining the URL can be achieved using intrinsic functions.
- The scheme of the URL is, of course, always "http"; the rest of the string is obtained via an intrinsic function *GetAtt,* with two parameters—the name of the elastic load balancer resource, specified in the *"Resources"* section and the string "DNSName." These two strings can be concatenated using the intrinsic function *Join* with an empty delimiter. Thus, if the name of the elastic load balancer is *MyLB*, the *Outputs* section will look as follows:

```
"Outputs":
 {
  "URL":
    {Value": {"Fn::Join": ["",[http://", {"Fn::GetAtt": ["MyLB",
"DNSName"]}]]} }
```

The introduction to *OpenStack* (*Finally, we have made it!*)

Mapping the *OpenStack* components into a physical architecture (an example)





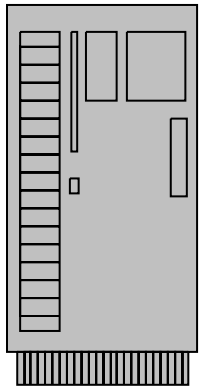
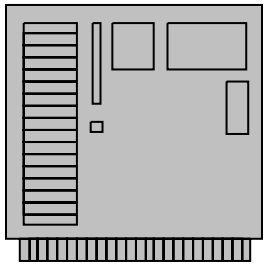


Image Node:
Glance registry
Telemetry agent

Controller Node:
Resource database
[Message queue server]
Cloud controller
Network controller
Volume controller
Scheduler
CLI server
Portals to
Image services
Storage services
Identity and Access Management
Dashboard API server
Orchestration API and engine
Telemetry Collector and database

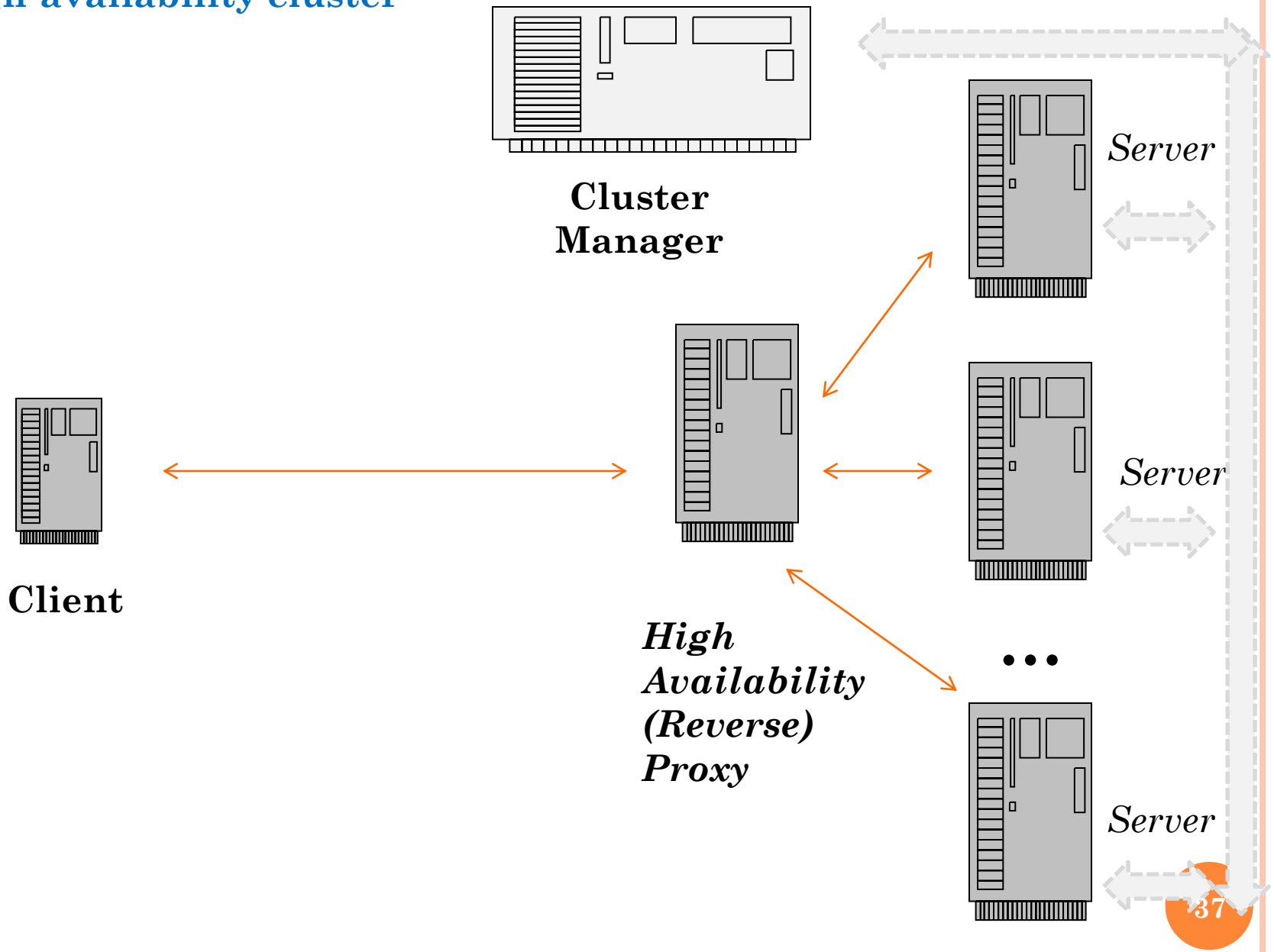
(via conductor)

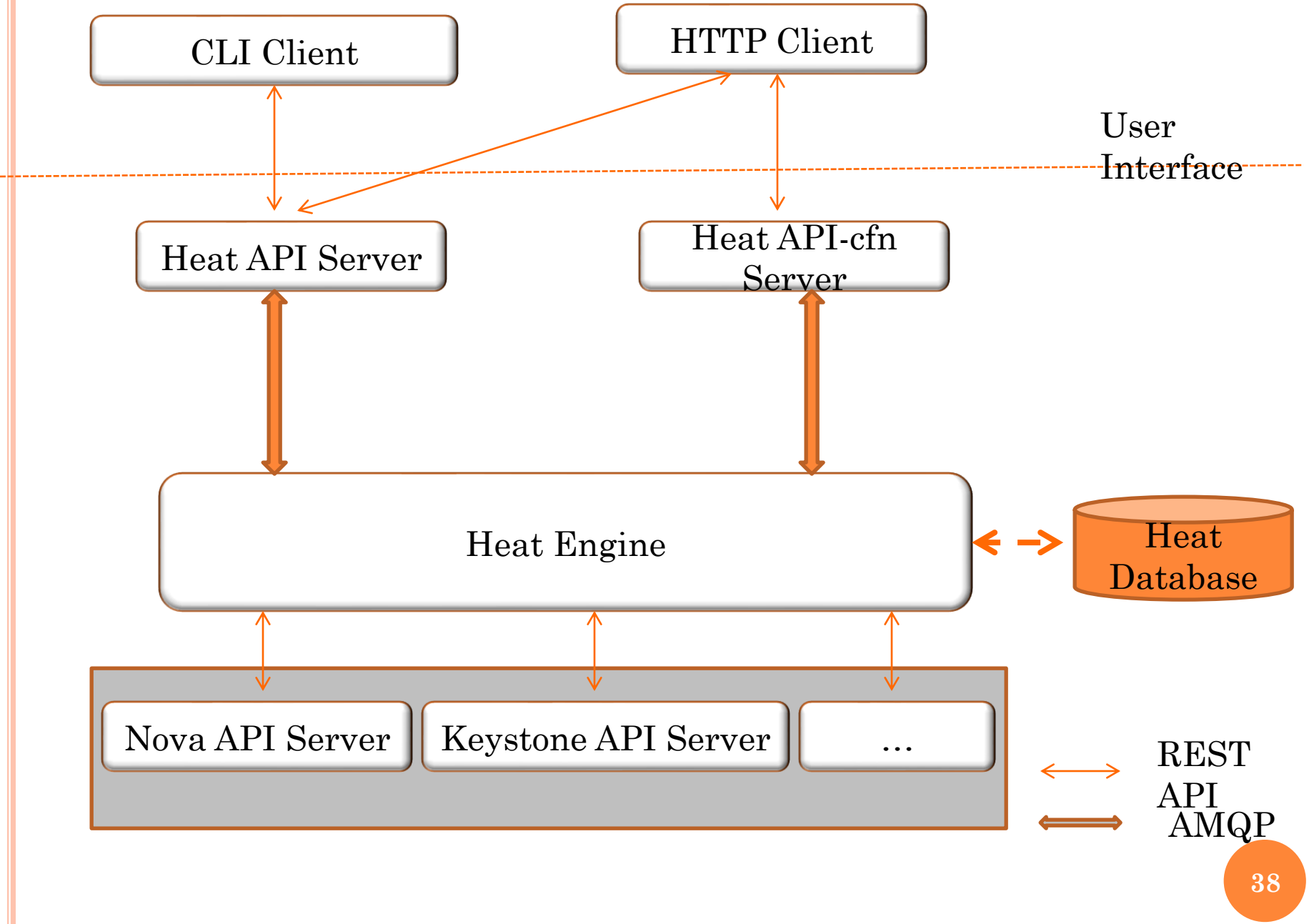
Compute Node:
Hosted VMs
Compute Drive (Hypervisor)
Compute Agent
Telemetry Agent



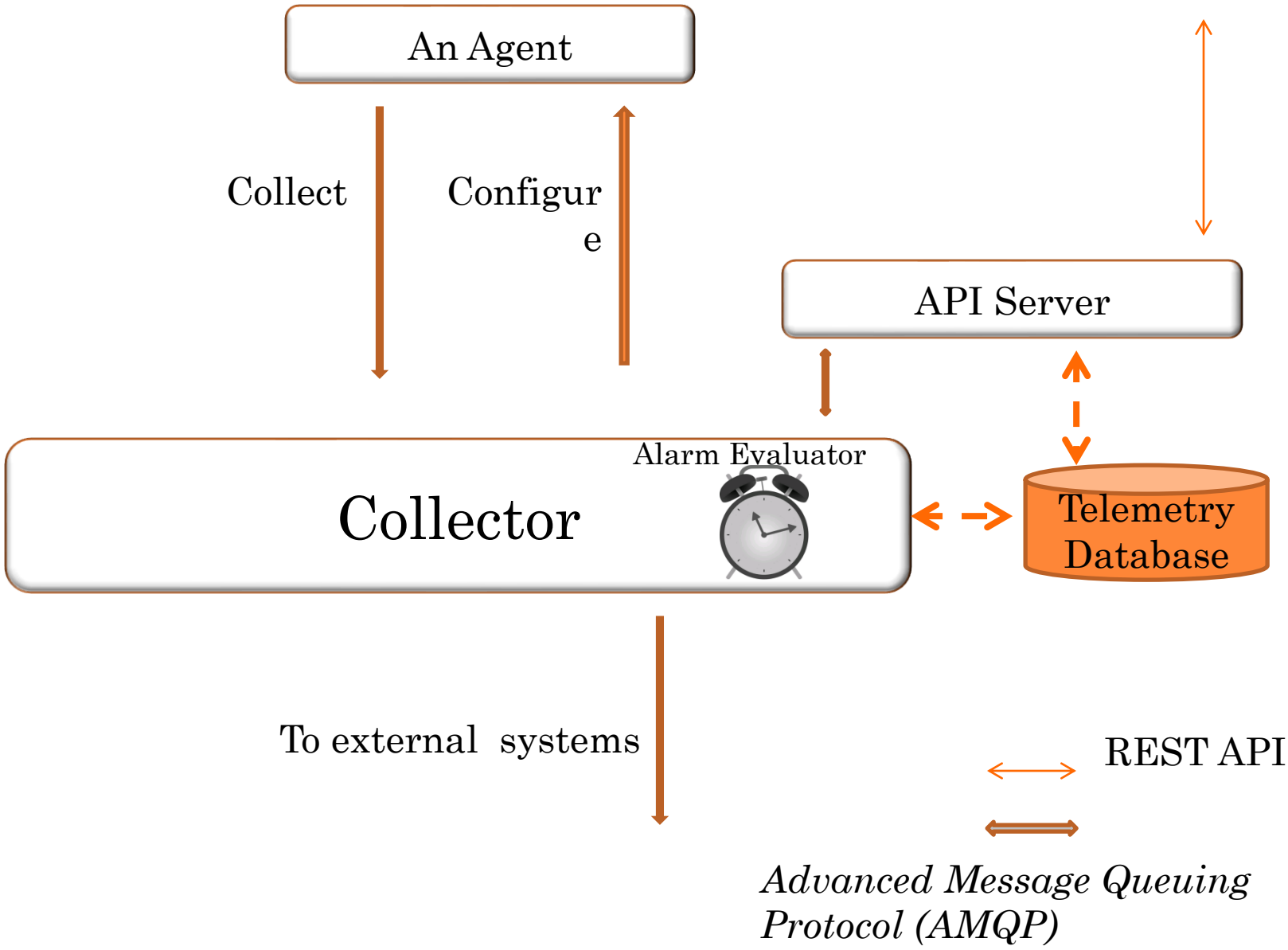
Storage Node:
Telemetry agent

A high-availability cluster





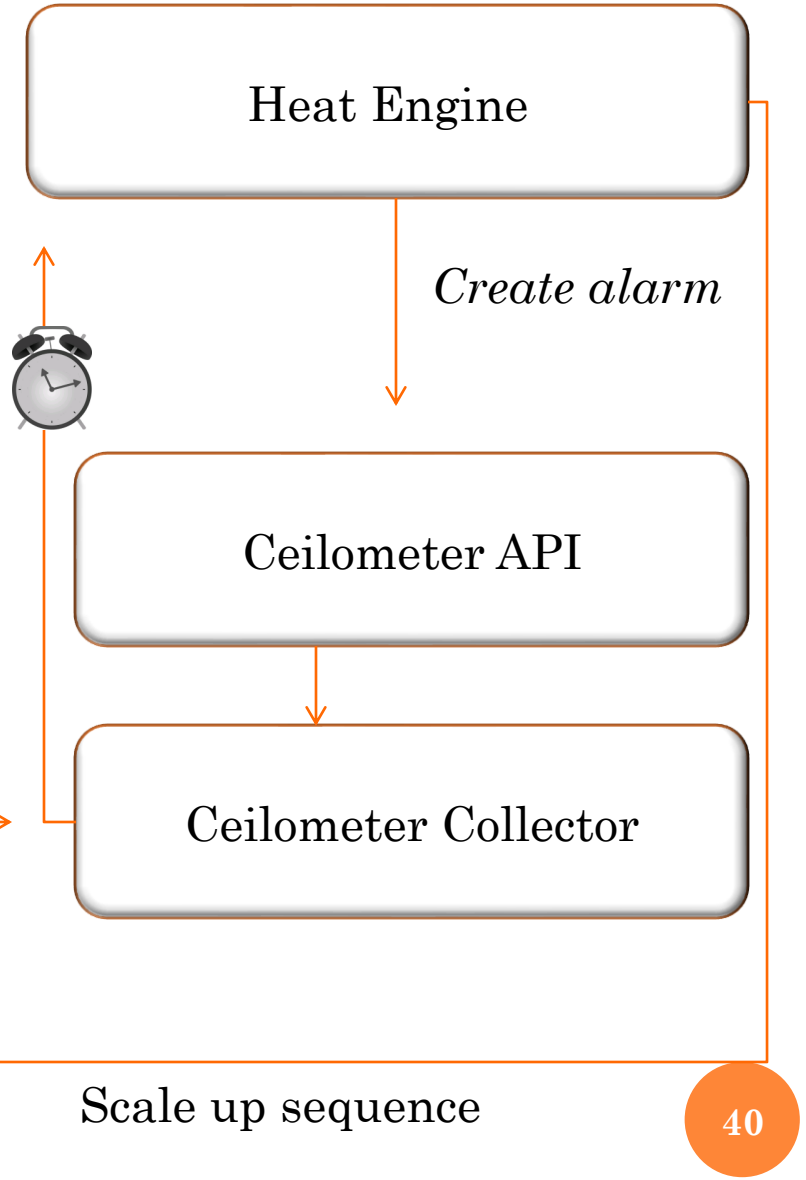
The *Heat* computing architecture



The *Ceilometer* computing architecture

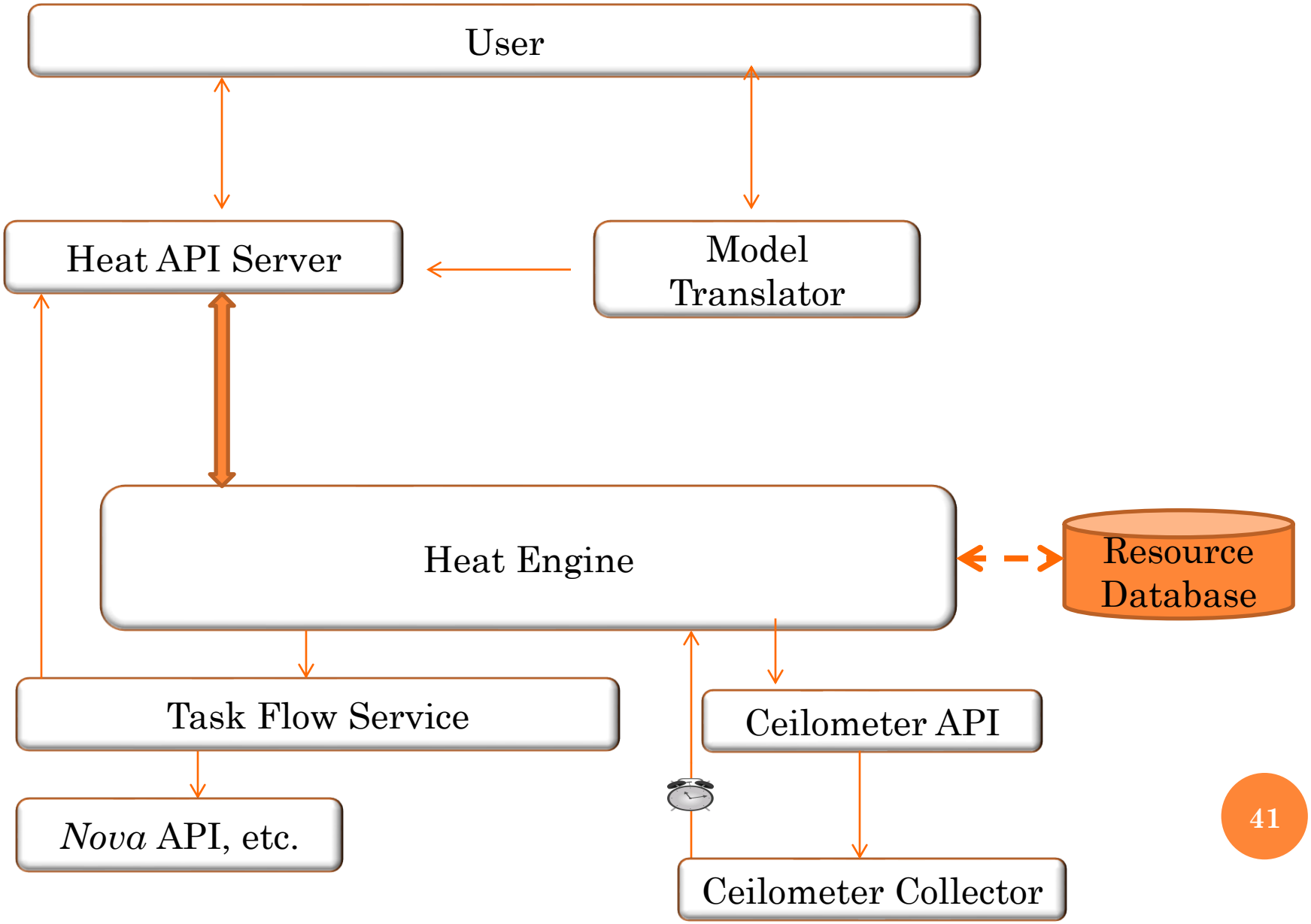
The template

```
"CPUAlarmHigh":
 {
 "Type": "OS::Metering::Alarm",
 "Properties":
    {
        "meter_name": "cpu_util",
        "description": "Scale-up if CPU >
       70%",
       ”  "alarm_actions":[…ScaleUpPolicy",
       "AlarmUrl ,
        Group“
    }
}
```

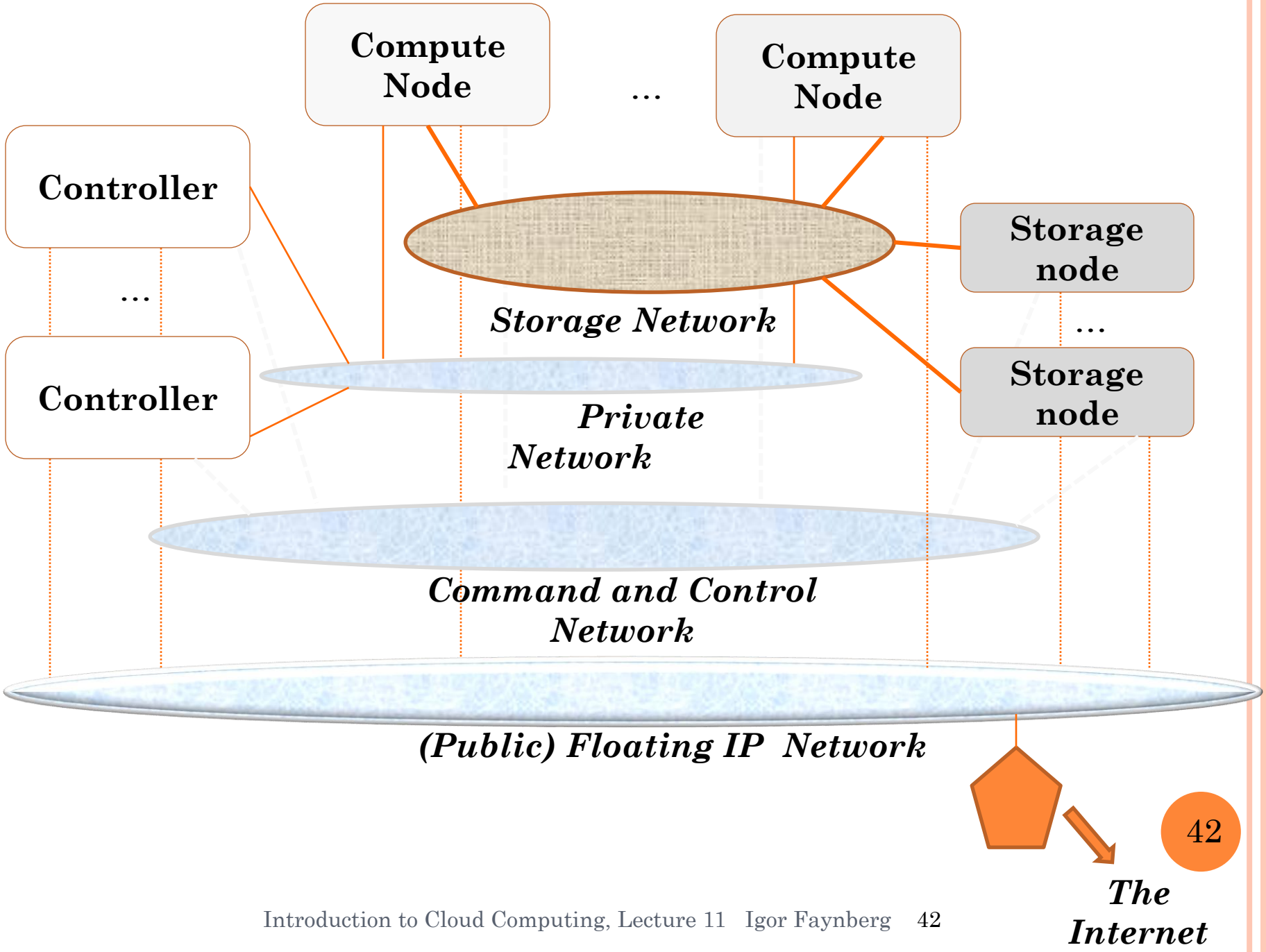


Interworking *Heat* and *Ceilometer*: An autoscaling example

Integrated Orchestration architecture



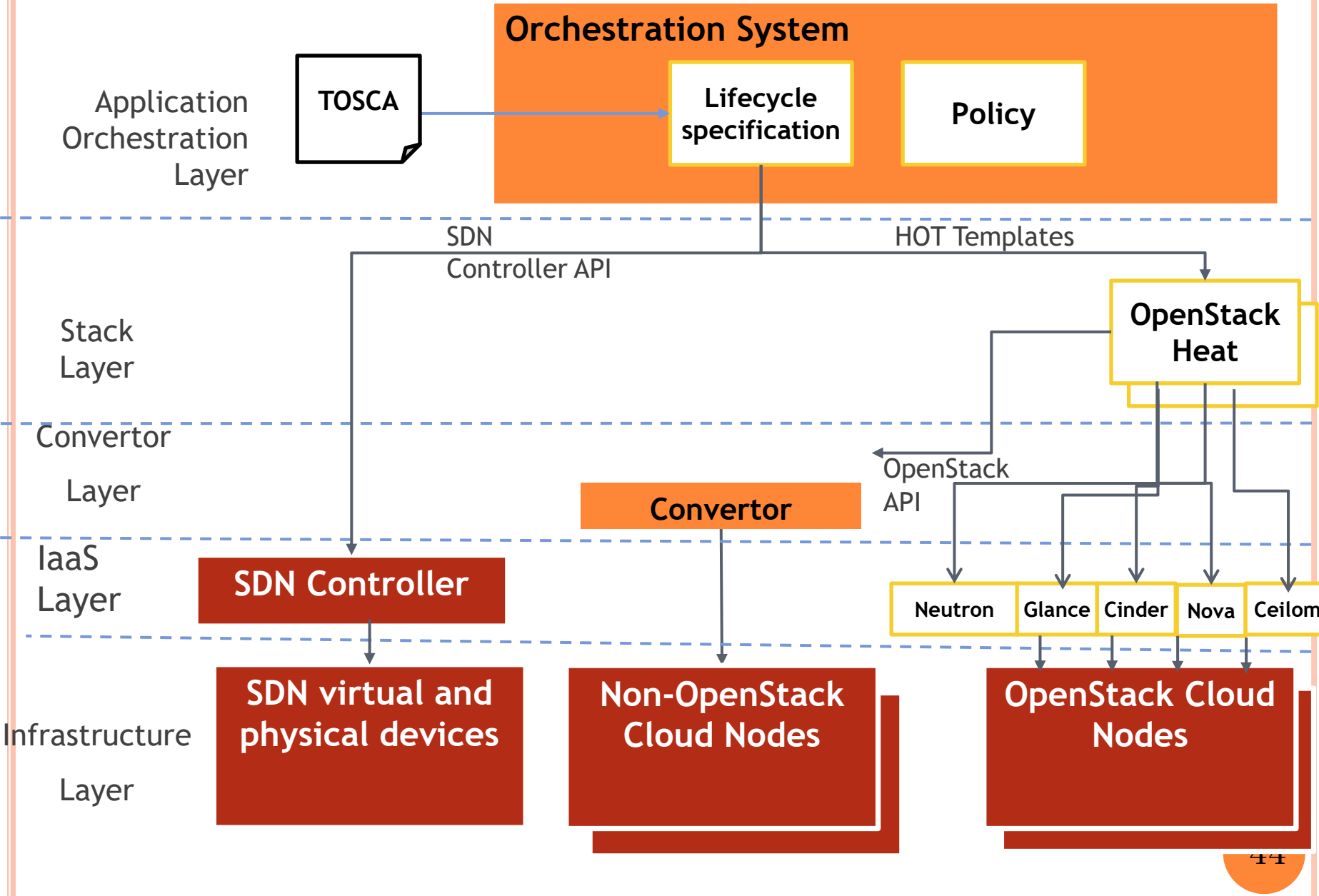
Networking with OpenStack nodes

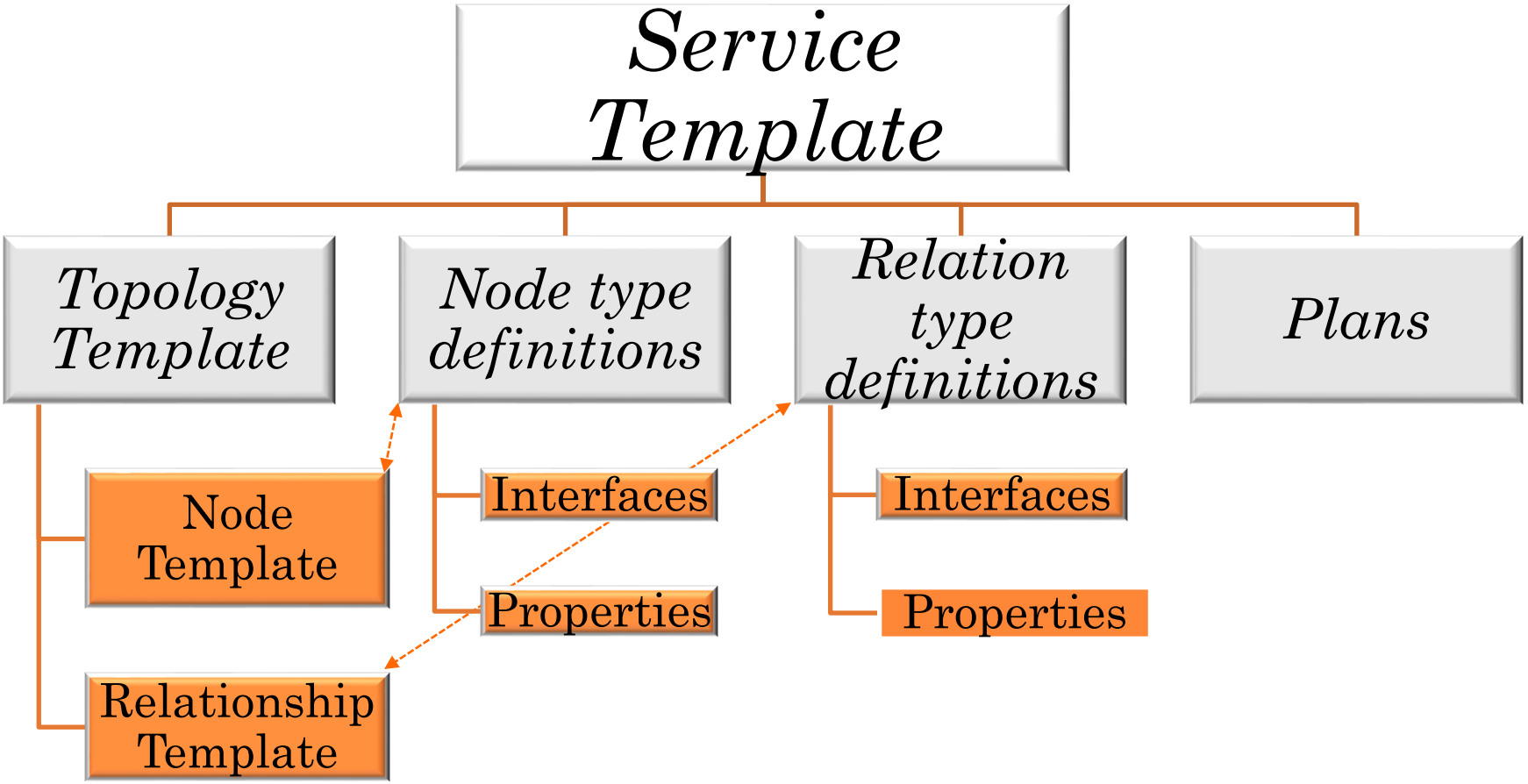


Orchestration with *Topology and Orchestration Specification for Cloud Applications* (TOSCA)

- An *OASIS standard*
- *Based on much research in modeling languagues (including N. Prywes' MODEL)*
- Integrates
  - Wide-Area Networking
  - Stack orchestration
  - Behavior pecification

43

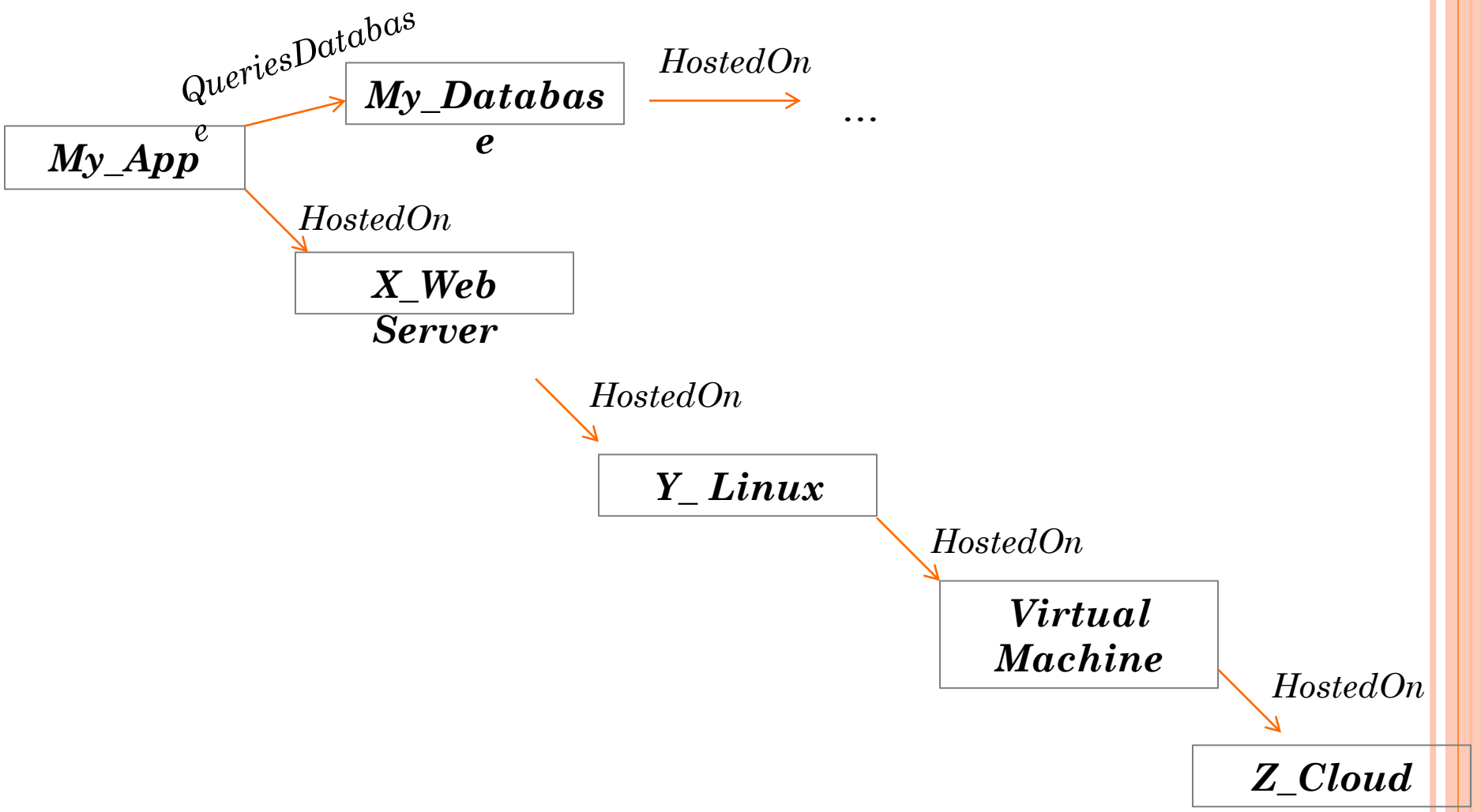

Orchestration System
Application Orchestration Layer
TOSCA
Lifecycle specification
Policy
SDN Controller API
HOT Templates
Stack Layer
OpenStack Heat
Convertor Layer
OpenStack API
Convertor
IaaS Layer
SDN Controller
Neutron
Glance
Cinder
Nova
Ceilometer
Infrastructure Layer
SDN virtual and physical devices
Non-OpenStack Cloud Nodes
OpenStack Cloud Nodes



The structure of an OASIS *TOSCA* template

45

A Topology Template Example



An example of translation of a *TOSCA* template into the corresponding *HOT* template

```
tosca_definitions_version: tosca_simple_yaml_1_0_0_build_1

description: My most simple server template

inputs:
 image_name:
    type: string
    default: tinyCentos
    keypair_name:
    type: string
    default: TEST-KEYPAIR-FOR-BDD

node_templates:
    server1:
    type: Ournodes.Compute
    properties:
    num_cpus: 1
    mem_size: 1024
    image_name: { get_input: image_name }
    keypair_name: { get_input: keypair_name }
```

```
heat_template_version: '2013-05-23'

resources:
  server1_119677379087827598:
   metadata: {deploymentId: '119677379087827597',
      deploymentNodeId: '119677379087827598'}
    properties:
      availability_zone: zone0
      flavor: default
      image: 41d1769d-a6fc-4849-8c70-8f5b0d64fcb5
      key_name: TEST-KEYPAIR-FOR-BDD
      metadata: {deploymentId: '119677379087827597',
       deploymentNodeId: '119677379087827598'}
      name: cPaaS_server1_119677379087827598
      networks:
      - port: {get_resource:
       server1_119677379087827598_mgnt_port}
    type: OS::Nova::Server
  server1_119677379087827598_mgnt_port:
    properties:
      fixed_ips:
      - {ip_address: 10.38.237.63, subnet_id: 66bc11c4-2312-
       4c55-a169-20ca0d21f5f1}
      network_id: 8796324e-7a75-43f7-8be8-32581d846f5c
      security_groups: [d8be0483-77da-41b5-bc17-
       a55c2eaa0384]
    type: OS::Neutron::Port
```

47