Virtualization as a CyberInfrastructure Architecture

Jerry Sobieski

Cyber-Infrastructure Strategist/coPI BRIDGES Project

George Mason University

Presented [virtually] to:
Netcentric 2020
Paris,FR
Dec 4, 2020

What is “Virtualization” ?

Interestingly, finding a definition for “Virtual” as it is used today in cyber-infrastructure context is not easy:

vir·tu·al (vûr’chōō-əl) (the Free Dictionary)

adj.

1. Existing or resulting in essence or effect though not in actual fact, form, or name: *the virtual extinction of the buffalo.*

2. Existing in the mind, especially as a product of the imagination. Used in literary criticism of a text.

3. *Computers* Created, simulated, or carried on by means of a computer or computer network: *virtual conversations in a chatroom.*

Virtualization (wikipedia)

In computing, **virtualization** refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, operating systems, storage devices, and computer network resources.

What is “Virtualization” ?

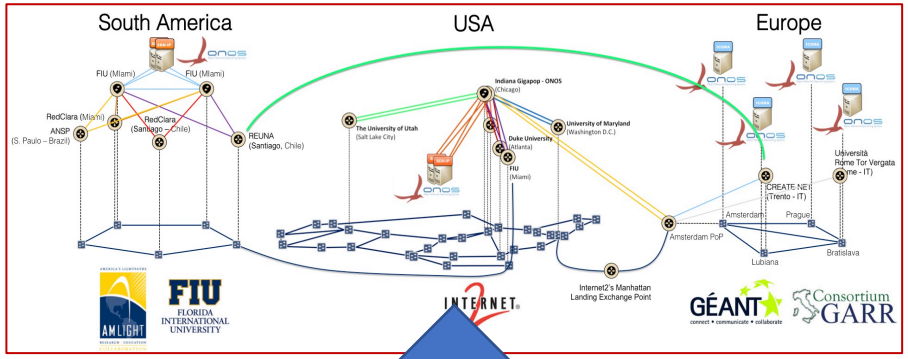
- For our purposes…. **Virtualization** is the process of creating an *abstract* user facing service object that behaves in a specific manner, often resembling some physical component or device, yet whose actual realization in the physical infrastructure is independent of any specific hardware device or technology.
  - Virtual Machines(VMs) – behave as if they are actual X86 servers, yet they may be realized in actual similar hardware, or emulated in software on any number of diverse hardware devices.
  - Virtual circuits – perceived as (“behaves as”) a point to point data transport conduit -regardless how it is realized as sub-rate link sharing, inverse muxing (lag), or multi-layer transport technologies.
  - Virtual addressing - behaves as if the user can address limitless memory space.
  - Virtual storage - networked filesystems vs drive/cyl/head/sector
- These objects are “virtually” the same as the objects they represent
  - Traditionally, these virtual objects are hardware analogs, e.g. VMs, VCs, etc.
  - …but…

Virtualization – a broader sense

- Virtual objects are *abstractions* - they define the user facing object behaviour– not the physical implementation.
  - These are not simply Virtual Machines or clouds…
  - Virtual objects can be anything – CI elements, network functions, instruments or sensors, …
- Given that "virtual" objects are abstracted resources that only define measurable/verifiable behavior attributes that the user experiences, and have no requisite physical implementation…
  - …We can thus extend the notion of a "virtual object" to include any resource that has well defined user facing attributes regardless of whether there is a precursor hardware object on which it is modeled
- We can define any virtual object we like –
  - The object attributes bound the limits of performance of individual virtual object instances
  - It is the service providers' responsibility to implement the object instance so as to meet those performance attributes…but it is left to the provider to decide how best to accomplish this deterministic performance given their infrastructure.

Basic virtualization service model

User facing virtual resources environments



Virtualization Services Layer

Resource Allocation, Scheduling, and Provisioning software

Physical Infrastructure

A Virtual Cyber-Infrastructure Architecture? ...

- Virtual Machines are enterprise quality production resources
- Virtual Circuits are standard production services for 30+ years
- Bare Metal Servers are managed in huge clusters with standard IPMI tools..
- Virtual Storage in various forms is ubiquitous and enterprise quality
- Emerging Virtual Routers and Switching
  - Quaga, VMX, OVS, Virtual OpenFlow Switch, P4 ....

Virtual ≠ Imaginary **!!** 

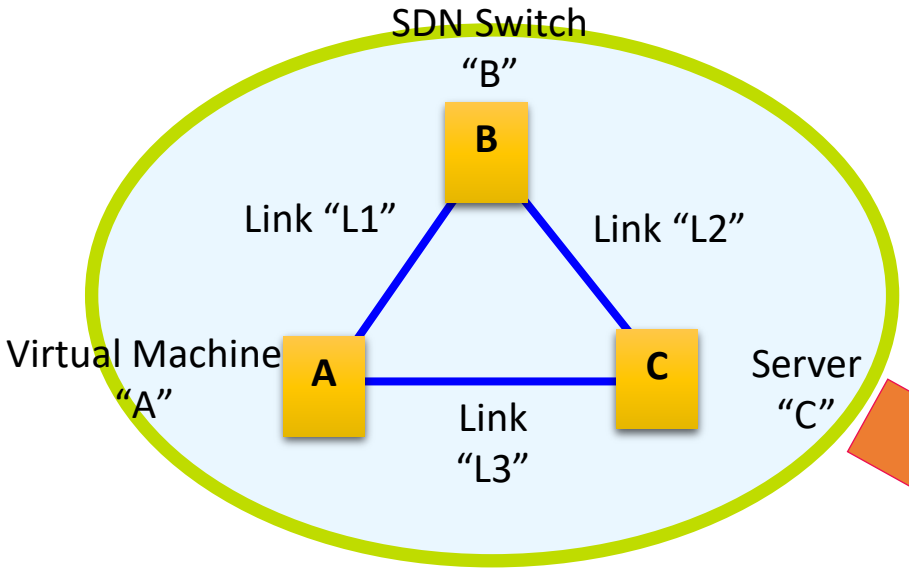
  - Not {emulated, simulated, fake, toy, pretend, faux, ephemeral,... }
- Virtual Environments can support and co-exist mature production network services, advanced distributed applications, and experimental pilots – all in parallel.

How do we go from individual virtual objects to a virtual CI *architecture*?

- We need a means of defining objects in a common model – the characteristics all objects contain:
  - Object class, Instance id, Object attributes, port specifications, children objects, port adjacencies
- We need a comprehensive construction model – grouping basic virtual resources together to form more sophisticated and powerful constructs – "composite" virtual resources
- This begins with *atomic* virtual objects that are mapped through a software abstraction layer onto physical infrastructure
- Atomic virtual objects can be grouped to create *composite* virtual objects
- Composite objects can themselves be grouped hierarchically using OO techniques to construct still more complex CI virtual objects – *virtual environments* or cyber-infrastructure slices
  - A virtual environment or CI slice is simply a top level composite virtual object
- We need a simple lifecycle model for virtual objects that define their state: their creation, operation, and release.
  - We need a simple set of lifecycle primitives (API) to manipulate these objects and to progress them through their lifecycle,
- We need a well defined means for individual virtual object instances to exchange information with other virtual objects

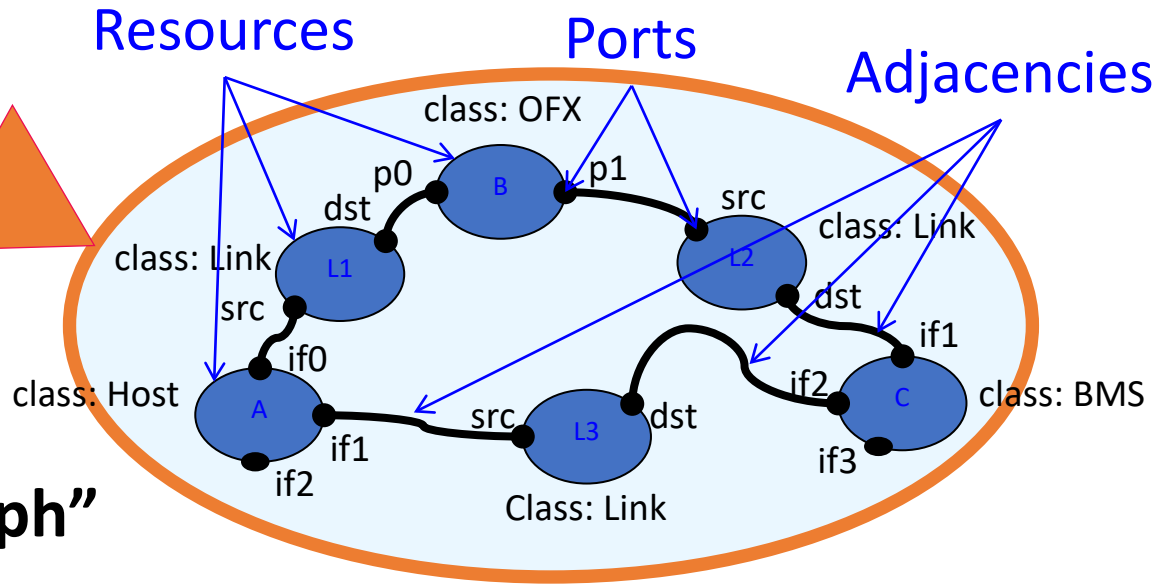
A Generalized Virtualization Model - “GVM”

Virtualized resources, in user defined/controlled topologies



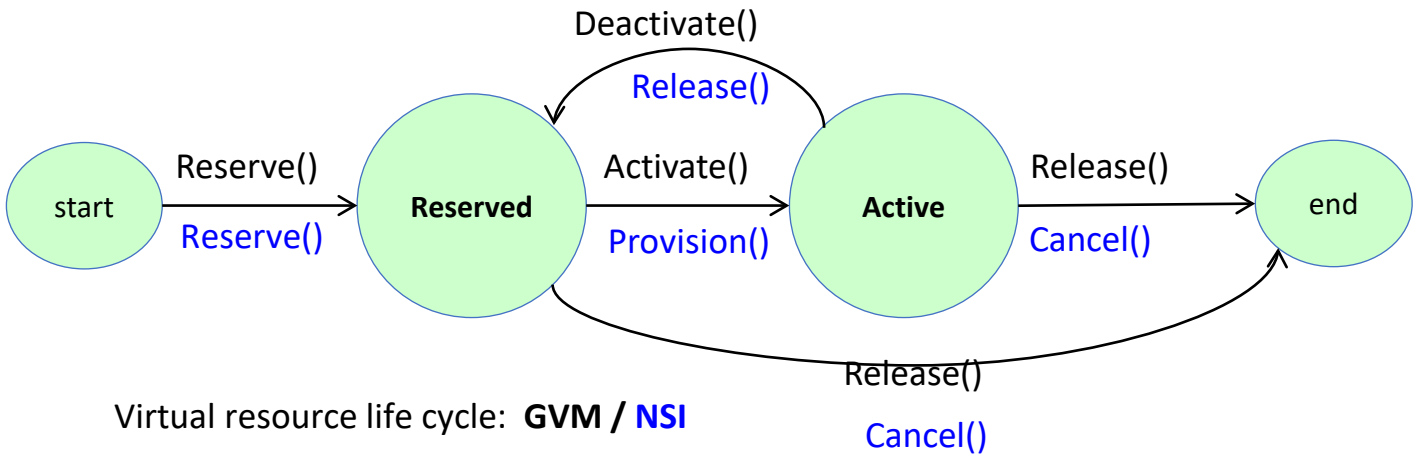
- All network components (the nodes and links in the graph) are treated as generalized **Resources**
- Data transits resources thru explicitly defined interfaces, or **Ports**
- Data flow topology is defined by port **Adjacencies**

Virtual environment/slice:
“Alpha” as conceived



“Derived Resource Graph”
data plane

GVM Life Cycle Model



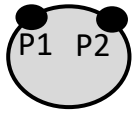
Virtual resource life cycle: **GVM / NSI**

- **Reserve()** – A request to find/create a resource instance and to reserve the infrastructure components needed by that resource
- **Activate()** – Given a reserved resource, this primitive provisions the resource and places the resource into service.
- **Query()** – Obtain the state information for a particular resource instance
- **Deactivate()** – Take a resource instance out of service, but retain the reservation.
- **Release()** – deactivate a resource and release the reservation

Atomic Resources, Composite Resources

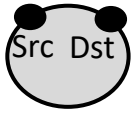
From atomic resources to running virtual environments

"Host" atomic class:

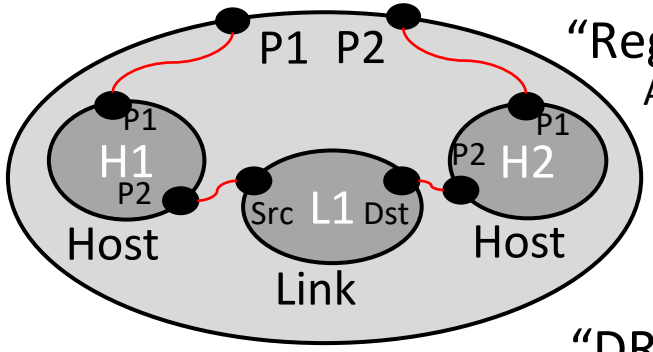


Ubuntu VM, 4GB, 1 core. Two ports

"Link" atomic class:



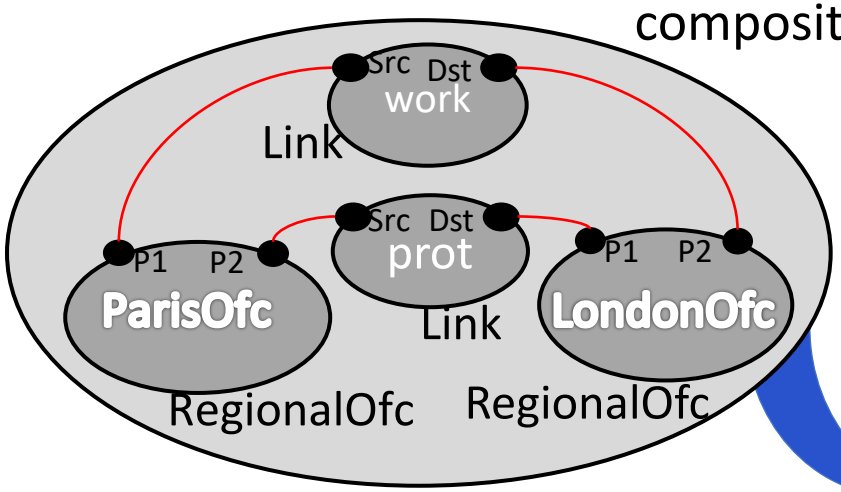
EoMPLS VC, BW, two ports



"RegionalOfc" composite class

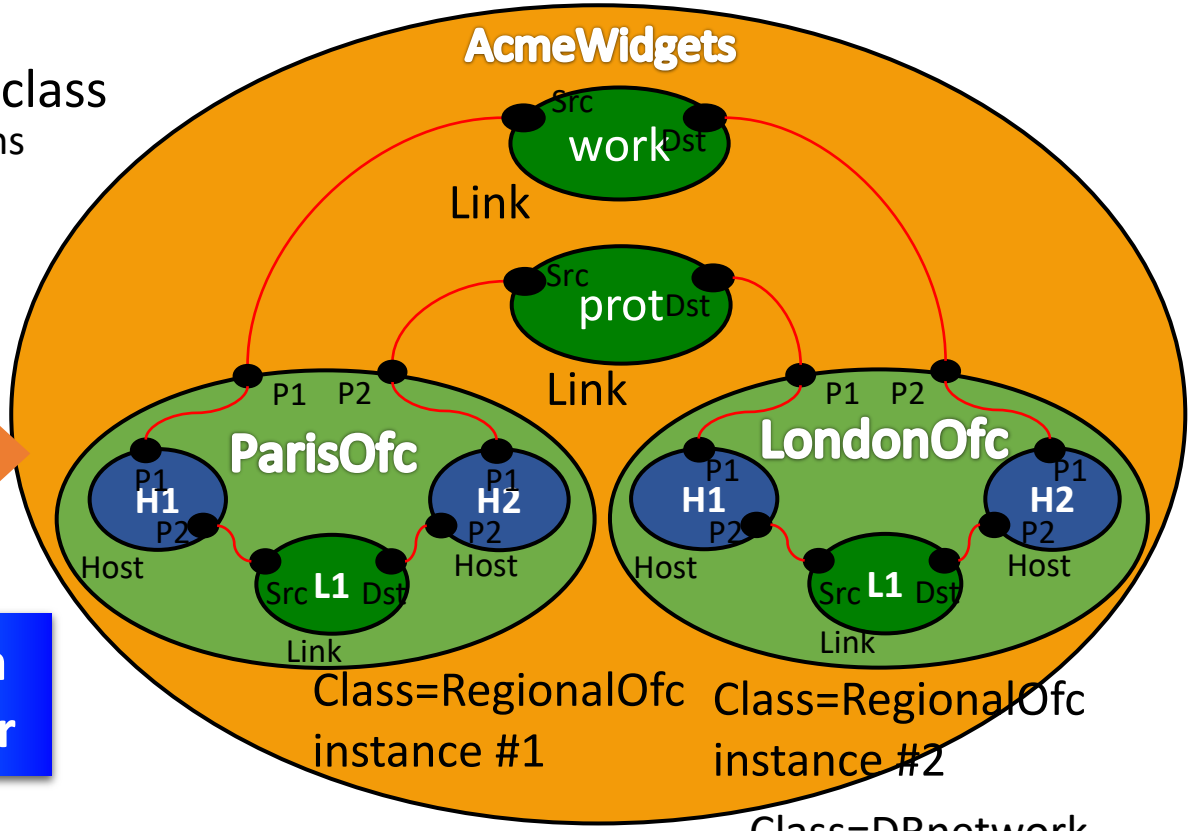
A composite resource contains other Resources, external ports, and port adjacencies

"DRnetwork" composite class

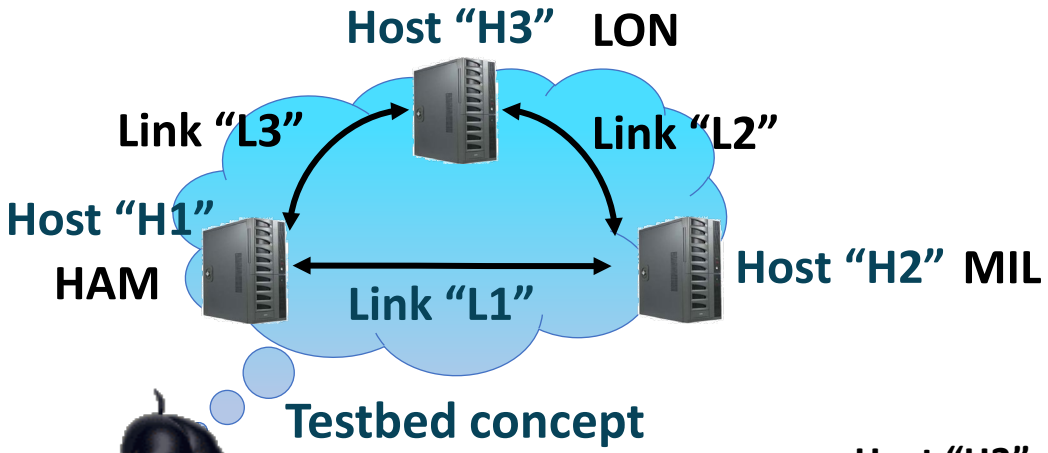


Virtualization mapping layer

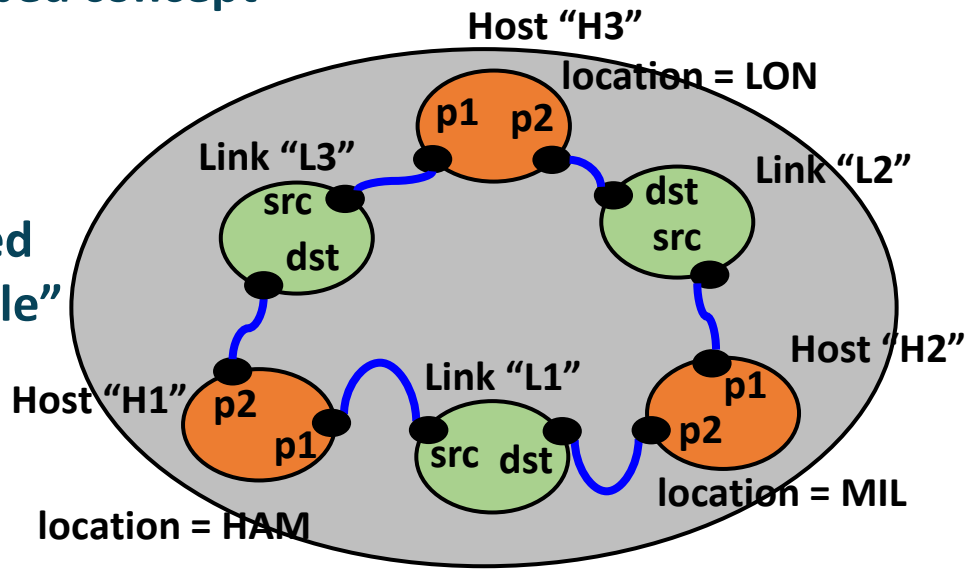
Instance of class "DRnetwork" named "AcmeWidgets"



Describing testbeds using the Groovy DSL – Composite resources descriptions



Testbed “Triangle”

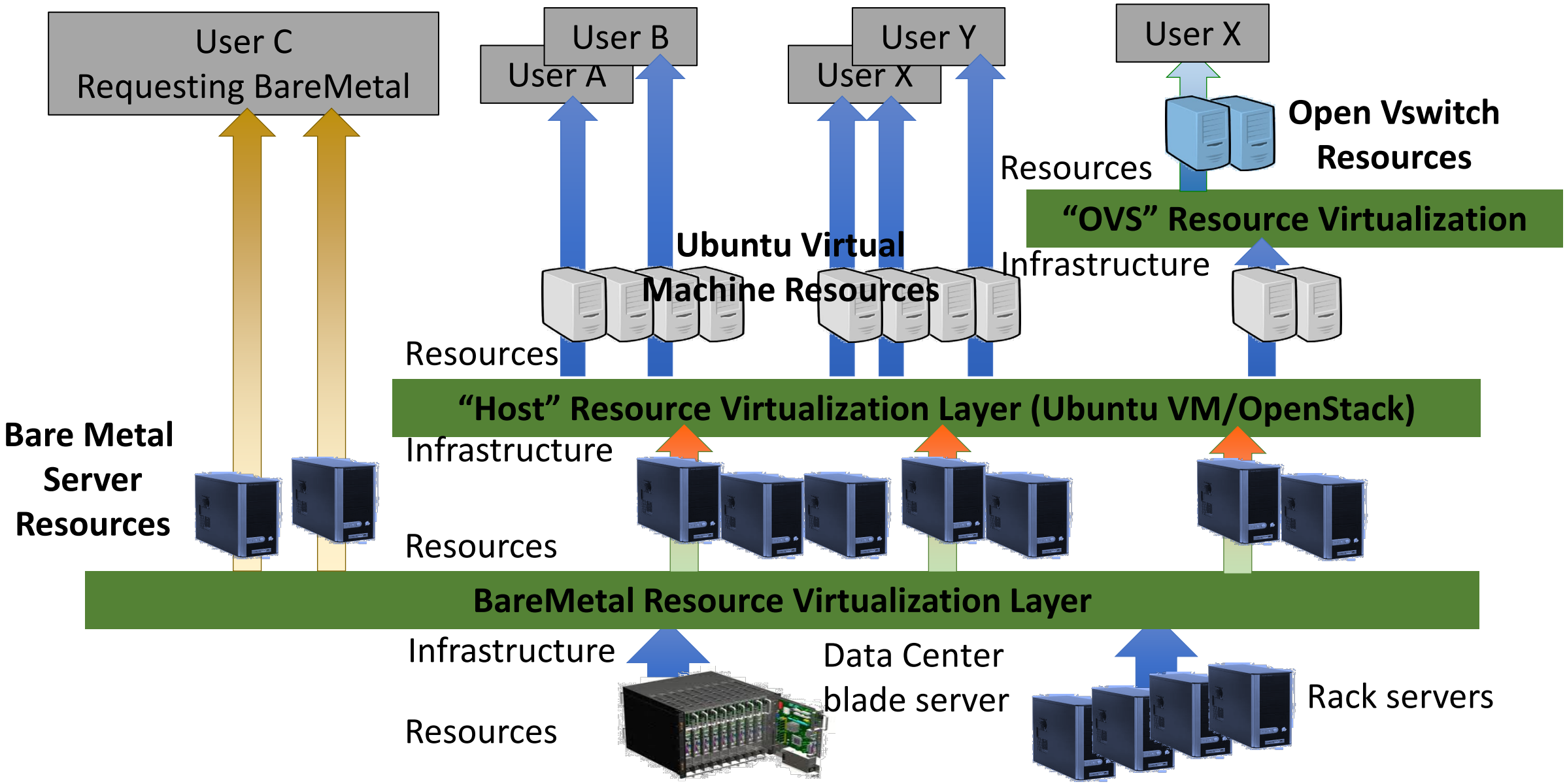


Bubble diagram

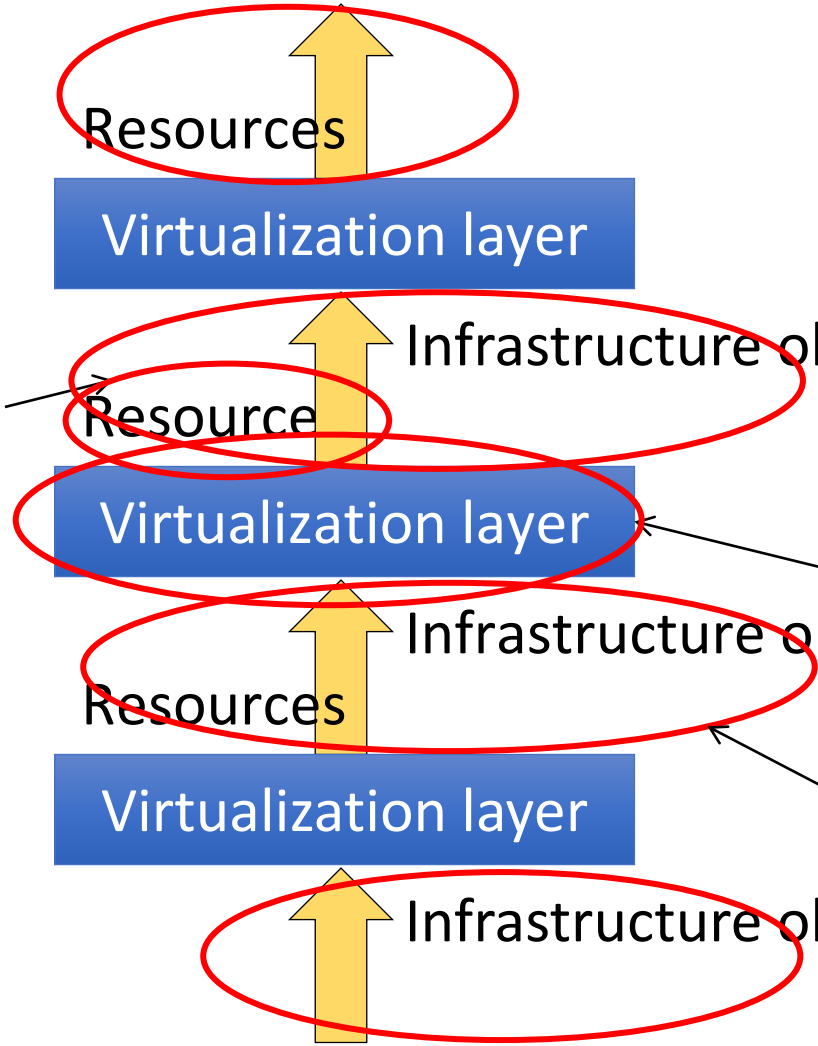
```
triangle {
  host {
    id="h1"
    location="ham"
    port { id=”p1" }
    port { id=”p2" }
  }
  host {
     id="h2"
     location="mil"
     port { id=”p1" }
     port { id=”p2" }
   }
   host {
     id="h3"
     location="lon"
     port { id=”p1" }
     port { id=”p2" }
  }
   link {
     id="l1"
     port { id="src" }
     port { id="dst" }
   }
 link {
     id="l2"
     port { id="src" }
     port { id="dst" }
   }
   link {
      id="l3"
      port { id="src" }
      port { id="dst" }
    }

    adjacency h1.p1, l1.src
    adjacency h2.p2, l1.dst

    adjacency h2.p1, l2.src
    adjacency h3.p2, l2.dst

    adjacency h3.p1, l3.src
    adjacency h1.p2, l3.dst

}
```

Done.

GVM Layered Resource Virtualization



Virtualization Layering –
Why a formal specification is important



Resources are formally described using the DSL specification

The virtualization model allows stacking of virtualization layers so that each layer can act as provider or client as needed.

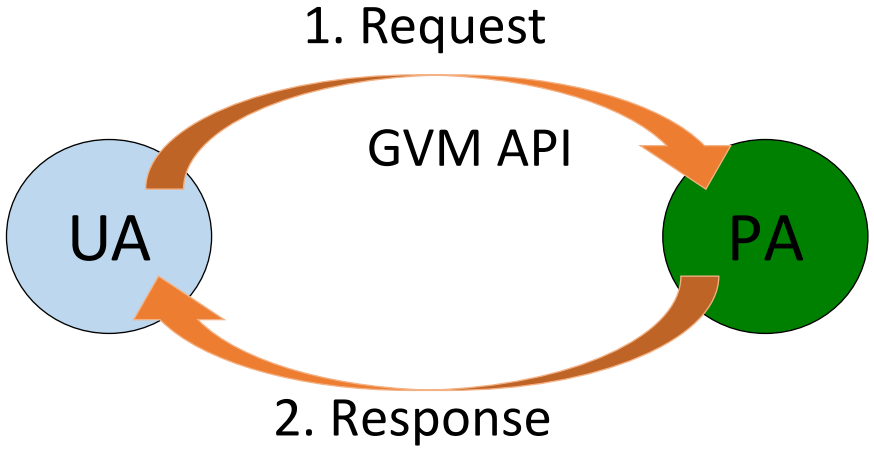
The resources are instantiated using class specific configuration processes acting upon infrastructure object(s) to generate an output "resource" of the specified type or class.

Since the "resources" and the "infrastructure" are at times the the same object(s), they can be described using the same DSL specifications!

This allows the same tools that monitor and manage Resources to also handle Infrastructure, and they can function at all levels for any client/provider

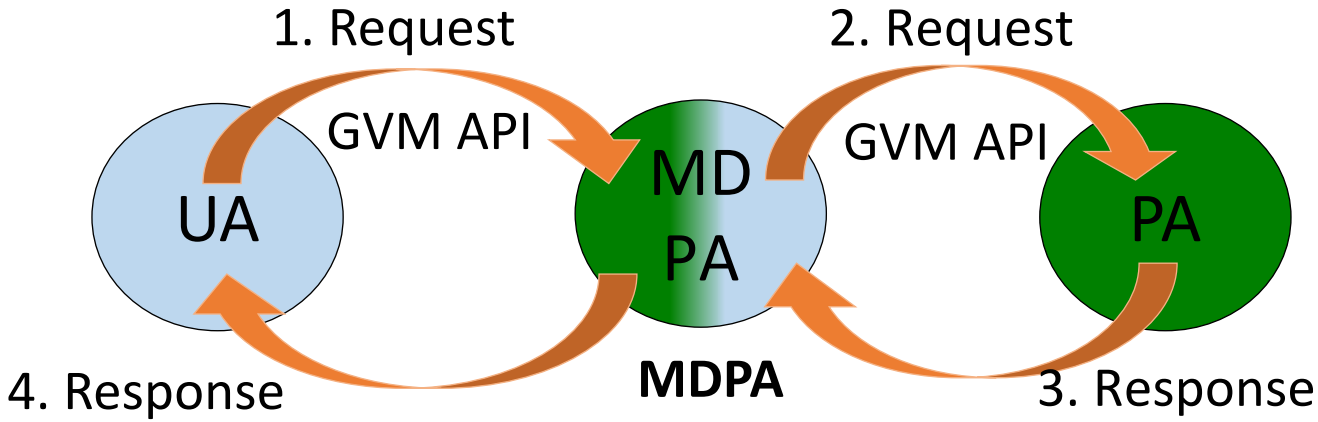
GVM Multi-Domain Peering

Basic User-Provider relationship



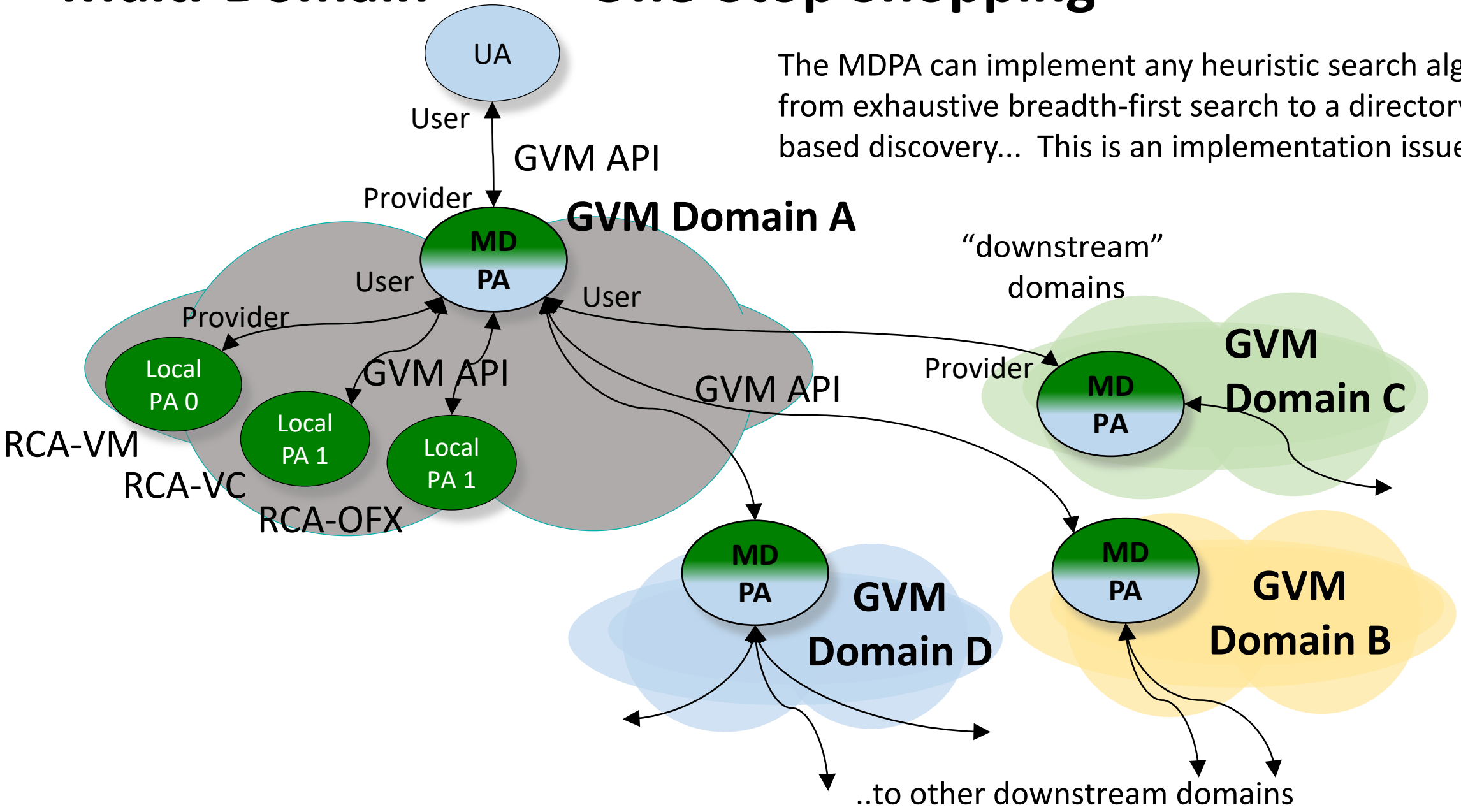
Basic User-Provider agent interaction

In the Mult-Domain model, the Multi-Domain Provider Agent plays both roles: “Provider” to upstream User Agents, and “User” to downstream Provider Agents



Multi-Domain One-Stop Shopping

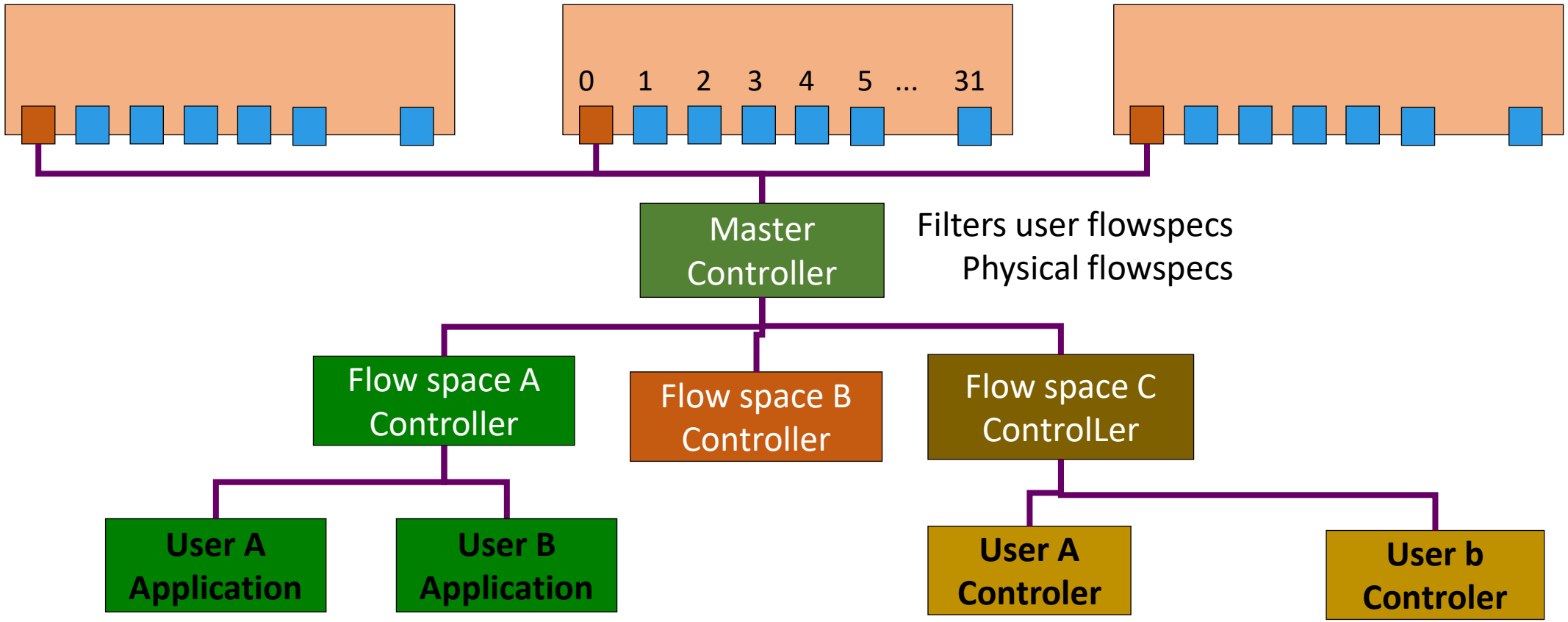
The MDPA can implement any heuristic search algorithm from exhaustive breadth-first search to a directory based discovery... This is an implementation issue.



Innovation: Runtime Virtual SDN Switching Fabrics

- Problem: SDN controllers don't share switch fabrics
  - This poses problems where a single SDN application does not require an entire switch, or where multiple SDN applications want to control their own fabric/ports.
  - Poses problems where different researchers require different controller agents
- Solution1: "Slicing"
  - Applications assigned a sub-space or "slice" (typically a subset of VLANs) from of a global network flow space
  - A proxy controller must filter and authorize all flow specs (e.g. FLowSpaceFireWall)
- Solution2: "Partitions" (sometimes called "port delegation")
  - Split the switch into multiple fabric "instances" each with its own context, ports are assigned to an instance
  - Still slices a single network flow space into port based subspaces, so flowspecs must still be inspected before installation
  - No port sharing means backbone links can only be assigned to one particular instance
  - Flowspecs still reference physical port and label information - Instances cannot be relocated/remapped without breaking the flowspecs and thereby breaking the application.

Controller based Partitioning



Master Controller must inspect all flow specs to insure each controller only manipulates its own flowspace.

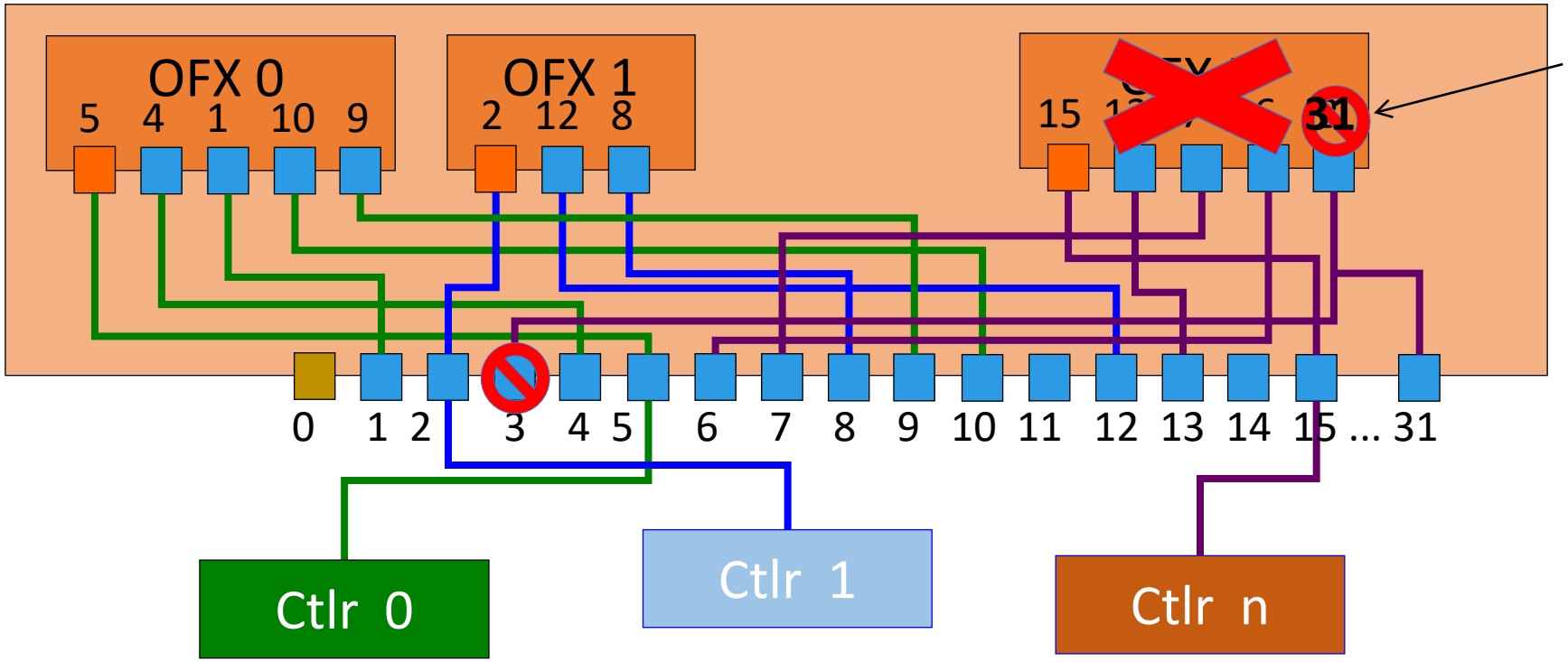
Controllers are allocated only a subset flow space of the full physical network flow space, typically (though not always), the same VLAN set in all platforms

Openflow Switch Sharing

- **Flowspace Delegation (Internet2 FSFW)**
  - A resource manager allocates flow subspaces to users (the whole system shares a single network flowspace)
  - A single master controller intercepts and inspects all flow specs to insure they fall within the user's [sub]space
  - The flowspace is a physical subspace, all control traffic is proxied, inverted proxy trees for become complex
  - Early innovative thinking to implement slicing in OpenFlow environment
- **Partitioning / Port Delegation (HPOS on HP5900)**
  - Separate switching instances in the switch device, delegates specific phy ports to each instance
  - The Openflow protocol handler on the switch inspects flowspecs to insure subspace conformance by instance
  - Each instance can have its own controller (good), but..
  - Uses physical flowspecs (no migration), does not share ports across multiple instances (no grooming)

Switch Partitioning

OFX Instances with port partioning



Breaks application [physical] flow specs

Port Map

Port ->	OFX
0	n/ a
1	0
2	1
3	5
4	0
5	0
6	5
7	5
8	1
9	0
10	0
11	n/a
12	1
...	

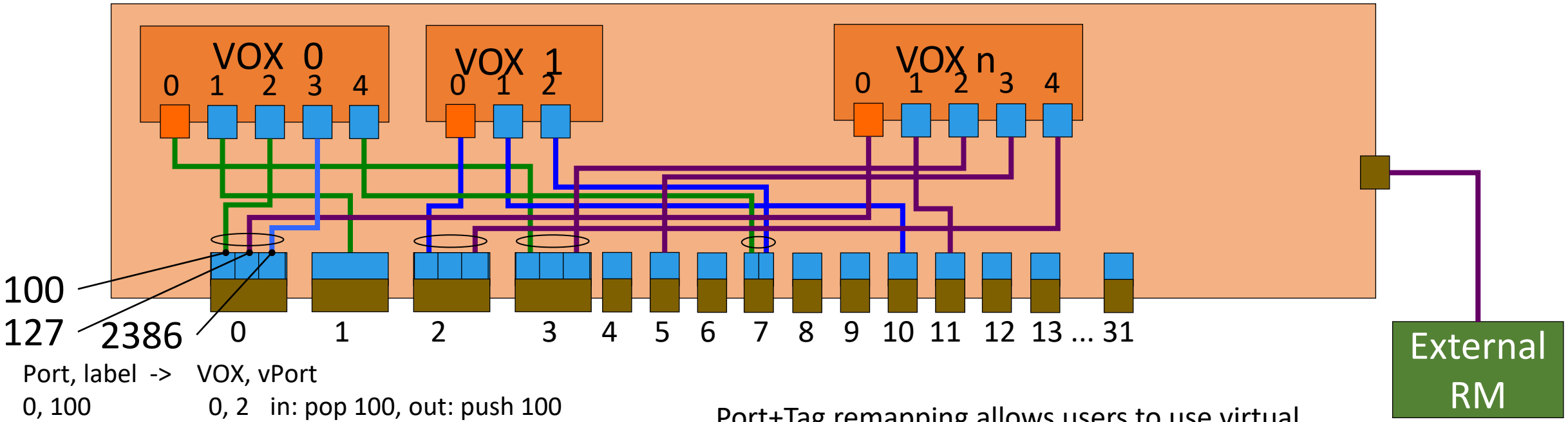
Each instance has its own controller, the switch processor only installs flowspecs from controller that match ports assigned to that instance.
Except for the port dimension, the user has full network flow space (no VLAN slicing is needed)
User Flow specs are ***physical port*** based flowspecs – moving the instance will break the flowspecs,
Ports cannot be split – the entire port is assigned to an instance – no flow grooming / aggregation
Useful in small ways, but not scalable.

Innovation in GVS: Virtual OpenFlow Fabrics

- Solution: *Virtualized Openflow Switch Fabrics*
  - This is Partitioning with a twist:
  - Each fabric instance has “virtual Ports” defined by the user:
    vport(1) ... vPort(n)
  - For in-packets, physical Port/outerVLAN (Label) 2-tuple of the frame are remapped (rewtritten) to an instanceID/virtualPort
    - The outer tag can be optionally popped on input, or pushed on output
- The result:
  - The user writes flowspecs against the *virtual ports*,
  - When combined with fully encapsulating virtual circuits (ala GVS), the user is able to command *their own full network flowspace   i.e. no more flowspace proxies, or slicing*
  - Because the Virtual Switch is no longer tied to specific physical ports, it can be moved and remapped-> Enables operational migration and grooming – and “save” of the switch state.
  - The instance can be relocated (vports remapped to different physical ports) without the user application needing to rediscover the topology and rewrite all their flowspecs.

Virtual Switch Instances

Virtual Switch with Virtual Circuit port mapping



Port, label ->	VOX, vPort	
0, 100	0, 2	in: pop 100, out: push 100
0, 127	n, 0	in: pop 127, out: push 127
0, 2386	0, 3	in: pop 2386, out: push 2386
1, -	0, 1	(no xport header processing)
2, 100	n, 4	in: pop 100, out: push 100
2, 3140	1, 0	in: pop 100, out: push 100
3, 25	0, 0	in: pop 100, out: push 100
3, 1870	n, 2	in: pop 100, out: push 100
...		

Port+Tag remapping allows users to use virtual flowspecs

Allows instances to share a physical port

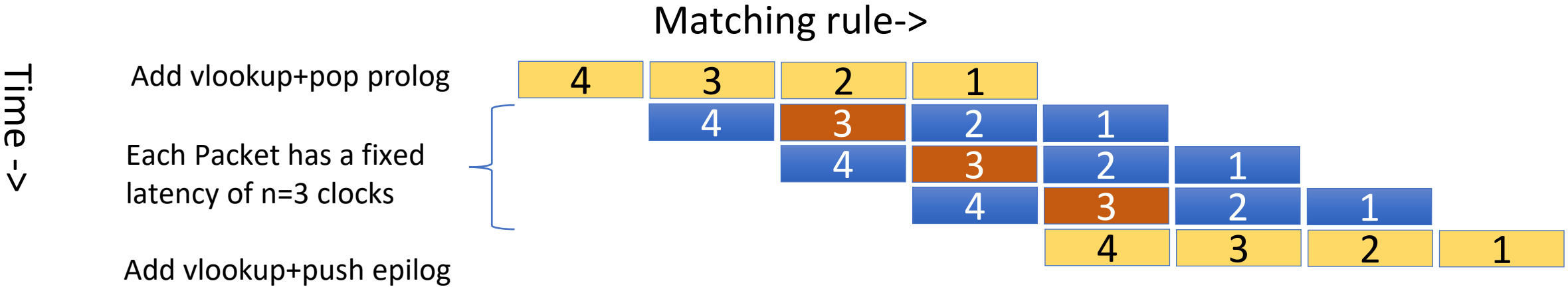
Allows transport tagging to be used for VCs, and to be popped before user sees it. Enables migration and grooming, Checkpoint/restart.

How did we do this?

- What does it require?
  - Modify switch/router software to support virtual OpenFlow instances
    - Add configuration commands to device mgmt API to define instances, specify port map
  - Define an “instance” – Each virtual instance has its own context: memory for counters, FIB, fabric ID, routing processor, protocol state for each instance,...
  - Add the port map prolog/epilog microcode into the fast path microcode:
    - Ingress: Lookup PhyPort : qtag/label in table; set metaData to corresponding Vswitch : Vport, pop outer qtag/label.
    - Egress: Lookup Vswitch, Vport in table, push outer qtag, queue to PhyPort
  - Virtualization sw (GVS) coordinates VC tags/labels, STPs, and Vports to build the port mapping table.
  - Vendor collaboration – We need vendor buy-in that such virtual routers can be a useful service model for global virtual environments
    - Corsa Technologies is the first vendor to work with us to deliver this feature
    - Demo’d in the lab, integration into GTS this summer, available in production Sep 2016
    - Line rate- at 100 Gbps (!)

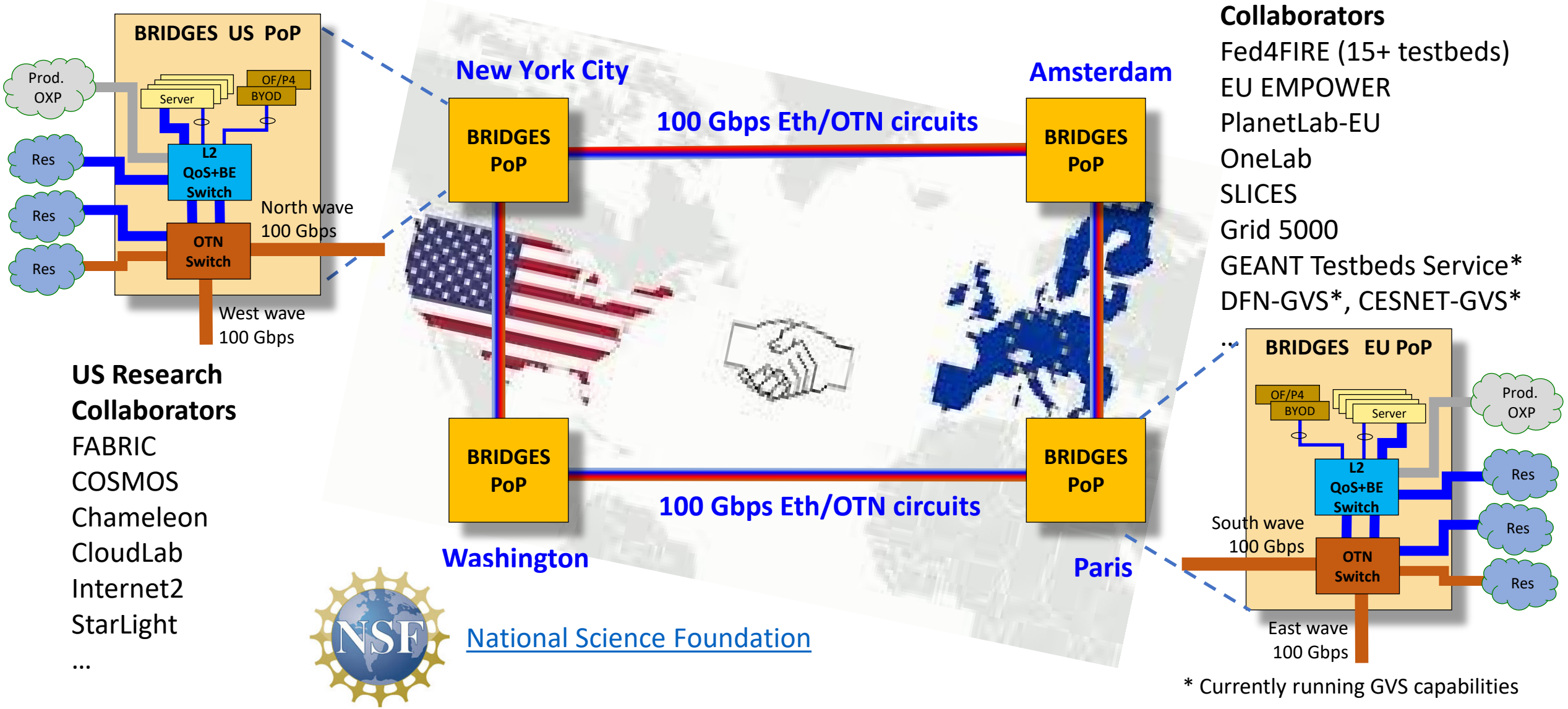
Why does this work?

- The pipeline design of fastpath packet handling is required to handle very high packet rates ... 100G
- The pipeline completes the processing of one packet on each cycle



Result is slight increase in latency (prolog+epilog), but the aggregate packet processing rate of 1 pkt/clock is the same !!

BRIDGES- Binding Research Infrastructures for the Deployment of Global Experimental Science



US Research Collaborators

FABRIC
COSMOS
Chameleon
CloudLab
Internet2
StarLight
…

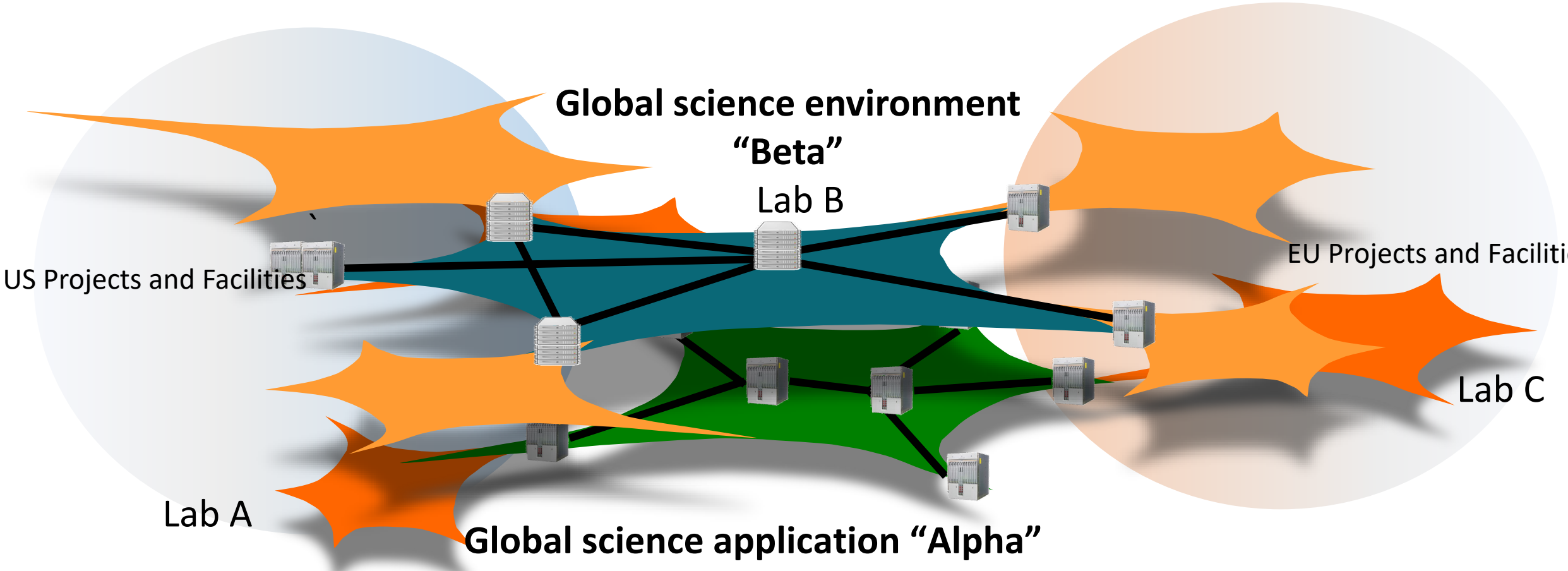
EU Research Collaborators

Fed4FIRE (15+ testbeds)
EU EMPOWER
PlanetLab-EU
OneLab
SLICES
Grid 5000
GEANT Testbeds Service*
DFN-GVS*, CESNET-GVS*
…

[National Science Foundation]

* Currently running GVS capabilities

BRIDGES – Virtualization as an Architecture
Application Specific Distributed Environments



A customized WAN infrastrcuture consisting of a broad range of dynamically allocated resources that are controlled by the client using SDN principles

Why is Virtualization important?

- ***Virtualization enables global common services ...***.
  - Allows common services to be defined, without dependence on specific hardware infrastructure particulars (technology agnostic) – Think Globally, Act Locally
- Virtualization enables automated orchestrated service delivery
  - Common service model is a necessity for automate processes.
  - Resource mgmt is software driven – service delivery is measured in seconds
  - Reduced error, enhanced flexibility of services.
- Operational considerations
  - Hardware sharing dramatically improves cost efficiency (reduced CapEx!)
  - Migration and grooming can efficiently distribute/concentrate workload as needed
  - Secure: Virtual objects are fully isolated and insulated from one another
  - Well bounded virtual service objects can be easily managed across many users

Futures: What is the roadmap?

- Increased adoption – get more projects and networks to adopt and deploy a generalized virtualization capability –
  - Replicate BRIDGES model between US and Asia/Pacific, and US and Latin America
- Community development and evolution
  - Simplify the software configuration – easier pilot deployments
    - Increased contiguous canvas in US and EU regional R&E networks (and other regions)
  - Formalize the specification – many details need thinking and a formalized approach
  - Multi-domain services – enable global virtual slicing ala BRIDGES
  - Migration and grooming capabilities of remapping resources for operational considerations and checkpoint restart
- Common object specifications
- Common policy to enable multi-domain resource acquisition

The End

- FFI:
  - Jerry Sobieski jsobiesk@gmu.edu or jerry@sobieski.net