LECTURE 15: DATACENTER NETWORK: TOPOLOGY AND ROUTING

Xiaowei Yang

OVERVIEW

- Portland: how to use the topology feature of the datacenter network to scale routing and forwarding

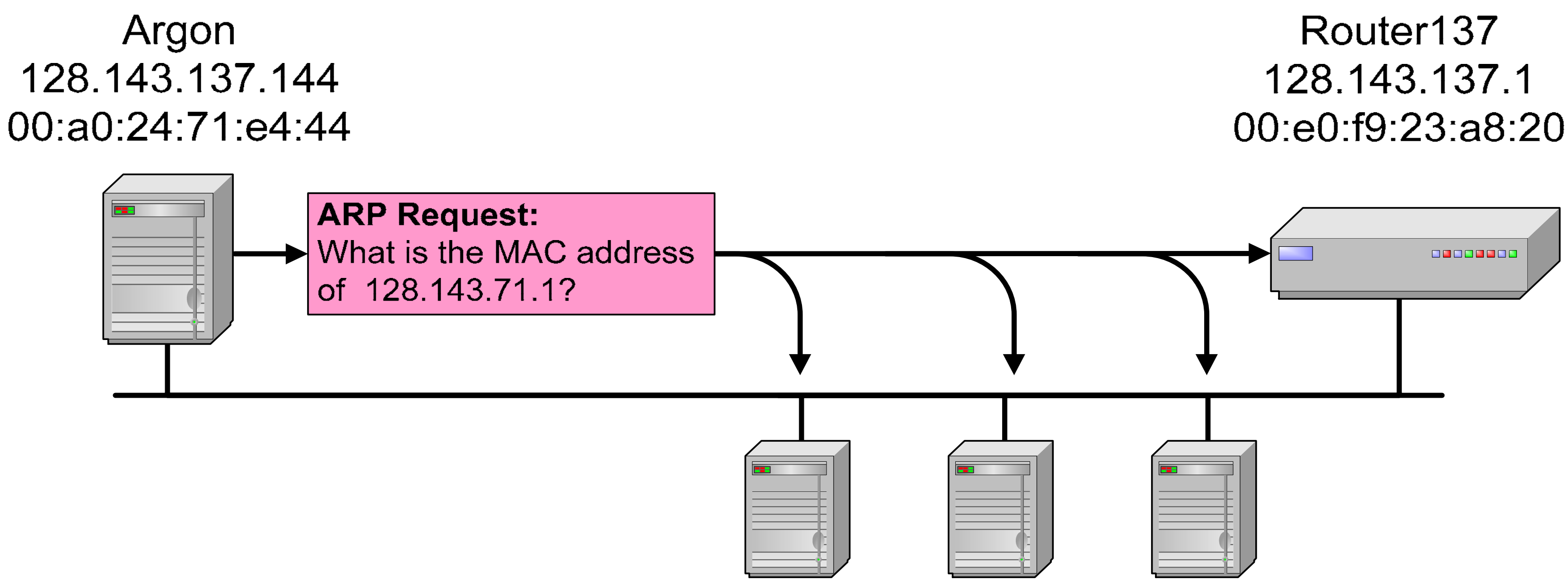
- ElasticTree: topology control to save energy
  - Briefly

BACKGROUND

- Link layer (layer 2) routing and forwarding
- Network layer (layer 3) routing and forwarding

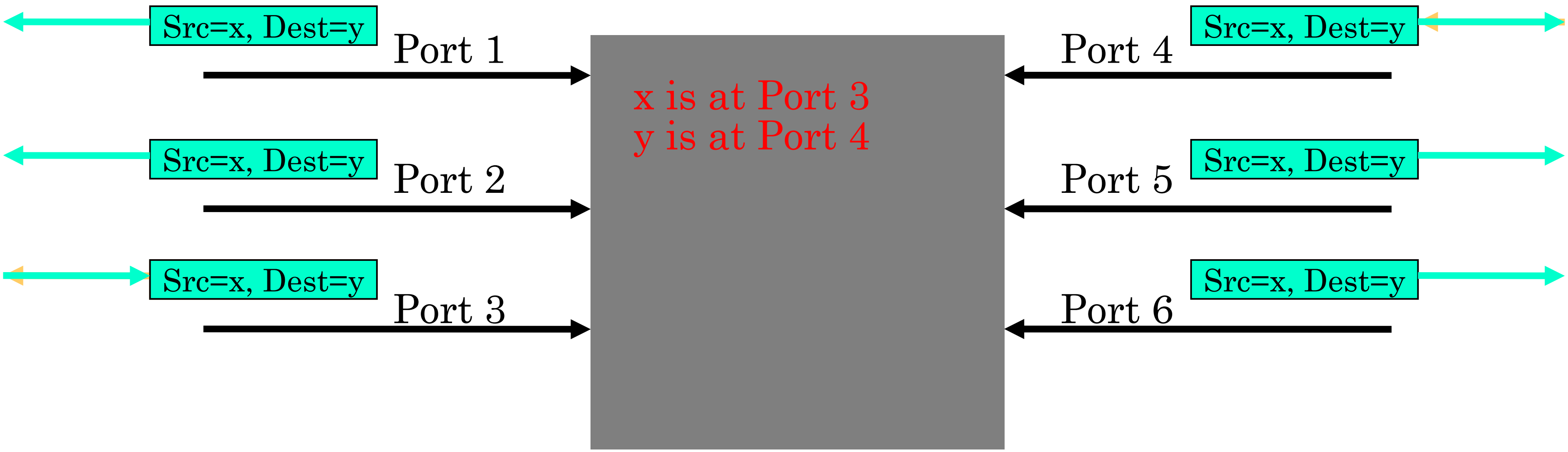
- The FatTree topology

LINK LAYER ADDRESSING



- To send to a host with an IP address p, a sender broadcasts an ARP request within its IP subnet
- The destination with the IP address p will reply
- The sender caches the result

LINK LAYER FORWARDING



- Done via learning bridges
- Bridges run a spanning tree protocol to set up a tree topology
- First packet from a sender to a destination is broadcasted to all destinations in the IP subnet along the spanning tree
- Bridges on the path learn the sender's MAC address and incoming port
- Return packets from a destination to a sender are unicast along the learned path

NETWORK LAYER ROUTING AND ADDRESSING

- Each subnet is assigned an IP prefix

- Routers run a routing protocol such as OSPF or RIP to establish the mapping between an IP prefix and a next hop router

QUESTION

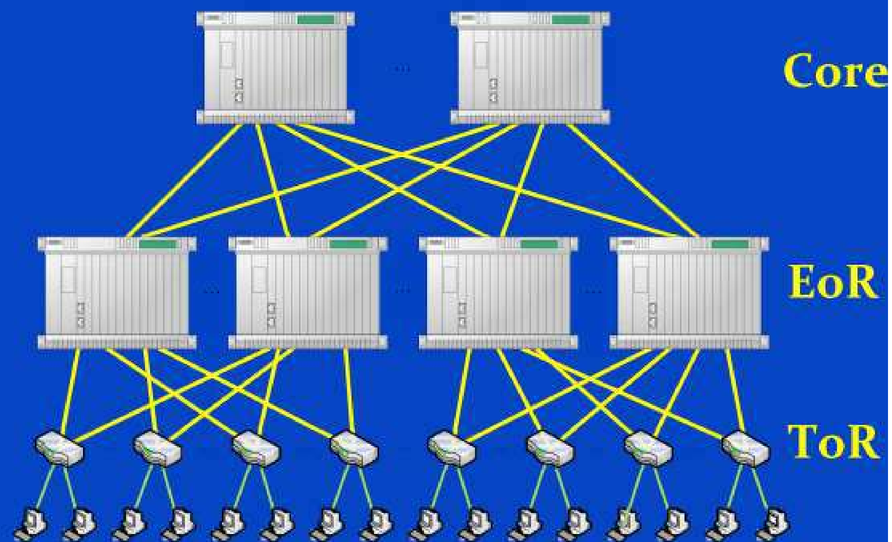
- Which one is better for a datacenter network that may have hundreds of thousands of hosts?

DATACENTER TOPOLOGIES

- Hierarchical
- Racks, Rows

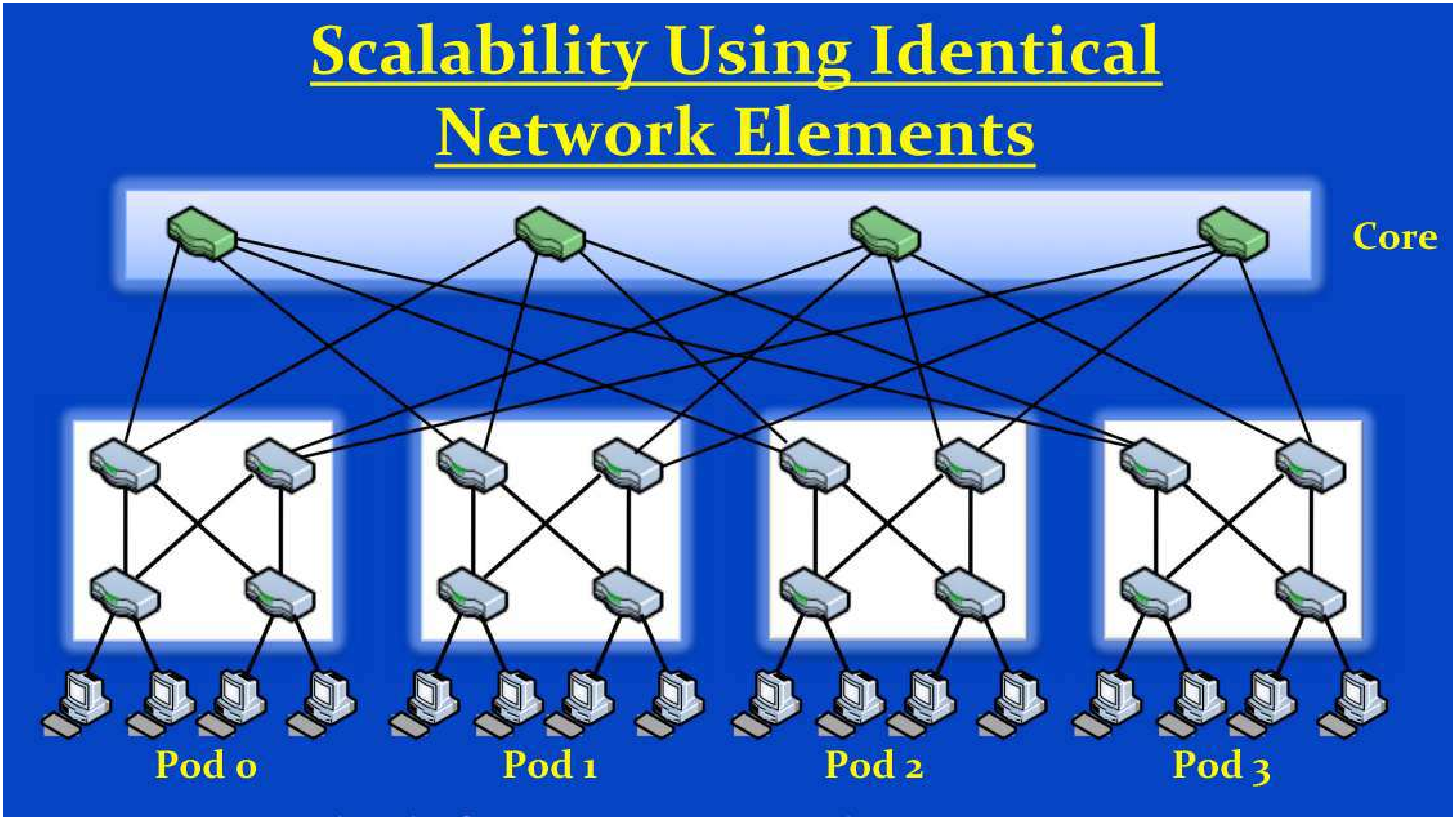
Current Data Center Topologies

- Edge hosts connect to 1G Top of Rack (ToR) switch
- ToR switches connect to 10G End of Row (EoR) switches
- Large clusters: EoR switches to 10G core switches
  - Oversubscription of 2.5:1 to 8:1 typical in guidelines
- No story for what happens as we move to 10G to the edge

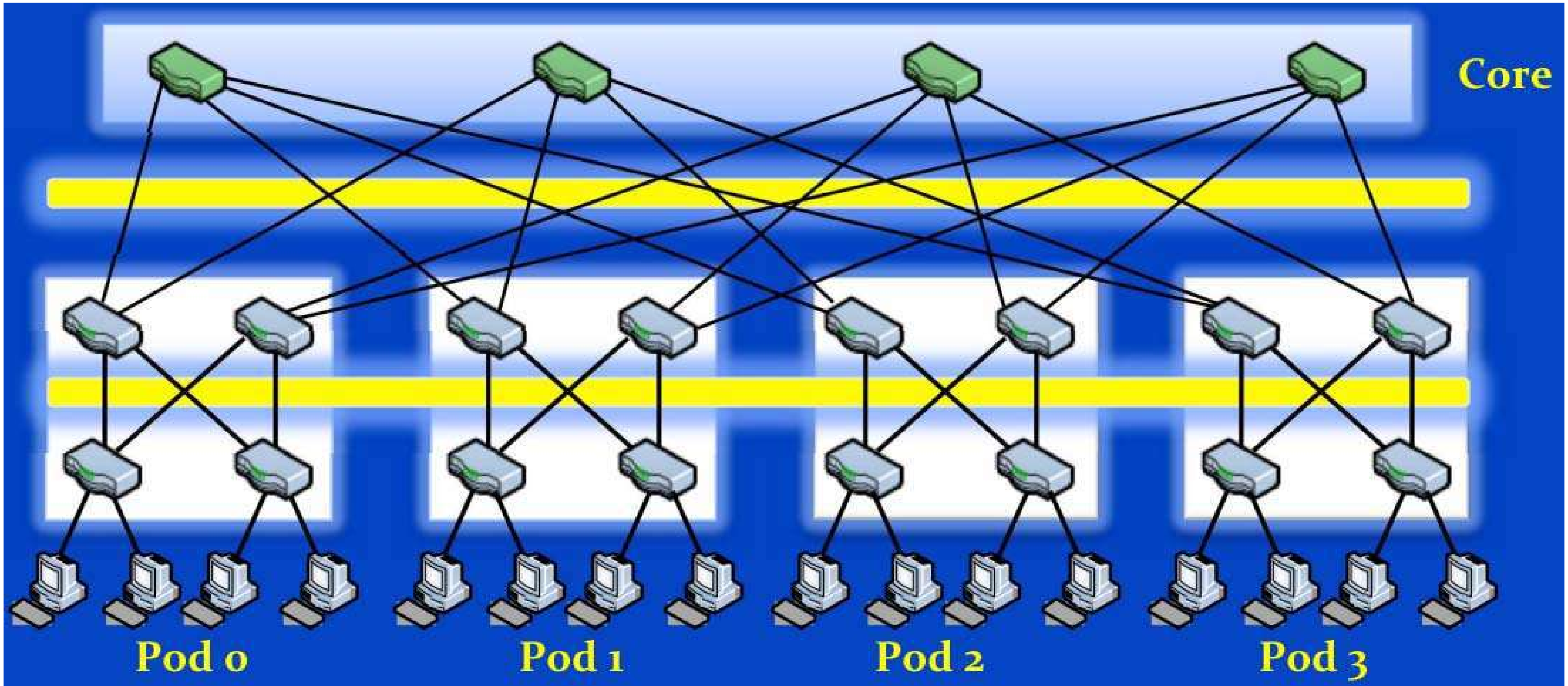



M
L
K

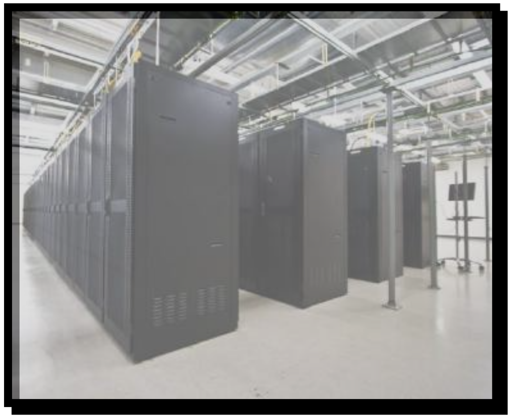
FAT TREE WITH 4-PORT SWITCHES



k-port homogeneous switches, k/2-ary 3-tree, $5/4k^2$ switches, $k^3/4$ hosts



- Full bisection bandwidth at each level of fat tree
  - Rearrangeably Nonblocking
  - Entire fat-tree is a 2-ary 3-tree



PortLand

A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric

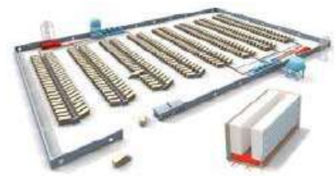
Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya and Amin Vahdat

PortLand In A Nutshell

- PortLand is a single logical layer 2 data center network fabric that scales to millions of endpoints
- PortLand internally separates host identity from host location
  - Uses IP address as host identifier
  - Introduces "Pseudo MAC" (PMAC) addresses internally to encode endpoint location
- PortLand runs on commodity switch hardware with unmodified hosts

Data Centers Are Growing In Scale

Mega Data Centers



- **Microsoft DC at Chicago**
- **500,000+ servers**

Large Scale Applications

- **Facebook, Search**
- **All to all communication**
- **Most Data centers run more than one application**

Virtualization in Data Center

- **10 VMs per server → 5 million routable addresses!**

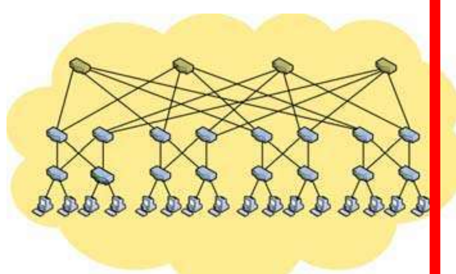
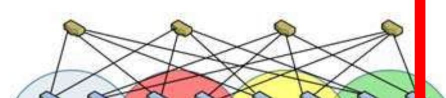
Goals For Data Center Network Fabrics

- Easy configuration and management – Plug and play
- Fault tolerance, routing, and addressing – Scalability
- Commodity switch hardware – Small switch state
- Virtualization support – Seamless VM migration

Layer 2 versus Layer 3 Data Center Fabrics

Technique	Plug and play	Scalability	Small Switch State	Seamless VM Migration
Layer 2: Flat MAC Addresses	+	-	-	+
Layer 3: IP Addresses	-	+	+	-

Layer 2 versus Layer 3 Data Center Fabrics

Technique	Plug and play	Scalability	Small Switch State	Seamless VM Migration
Layer 2: **Flat MAC Addresses**	+ 	– Flooding	– **Host MAC Address** / **Out Port** 9a:57:10:ab:6e / 1 62:25:11:bd:2d / 3 ff:30:2a:c9:f3 / 0 …. / …	+ No change to IP endpoint
Layer 3: **IP**	– 	+ LSR –	+ **Host IP Address** / **Out Port**	– IP endpoint

Location-based addresses mandate manual configuration

Cost Consideration:
Flat Addresses vs. Location Based Addresses

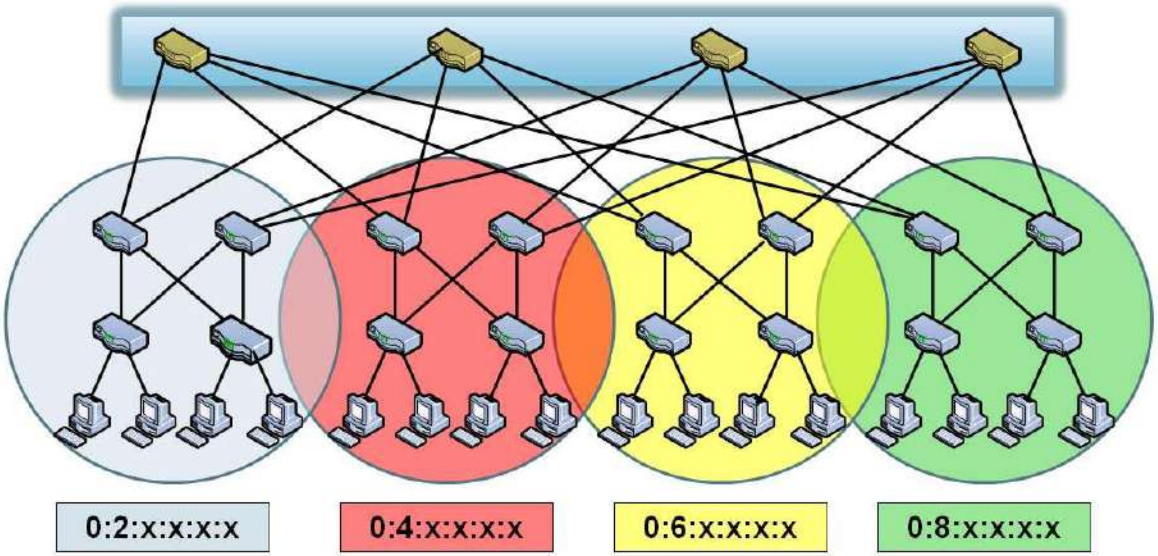
- Commodity switches today have ~640 KB of low latency, power hungry, expensive on chip memory
  - Stores 32 – 64 K flow entries

- Assume 10 million virtual endpoints in 500,000 servers in data center

- Flat addresses ➔ 10 million address mappings ➔ ~100 MB on chip memory ➔ ~150 times the memory size that can be put on chip today

- Location based addresses ➔ 100 – 1000 address mappings➔ ~10 KB of memory ➔ easily accommodated in switches today

PortLand: Plug and Play + Small Switch State



+ Pair-wise communication

Auto Configuration

PMAC Address	**Out Port**
0:2:x:x:x:x	0

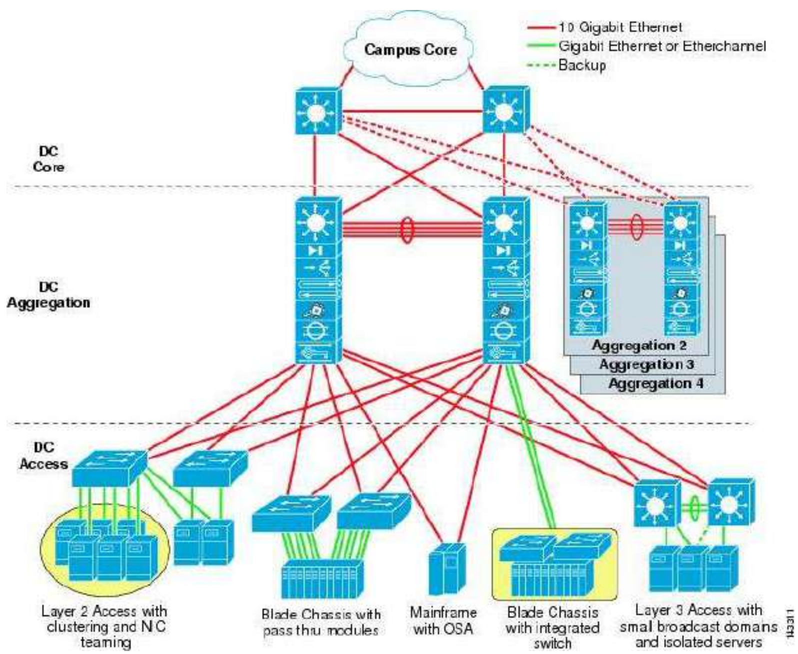
- 10 million virtual endpoints in 500,000 servers in data center

1. PortLand switches learn location in topology using pair-wise communication
2. They assign topologically meaningful addresses to hosts using their location

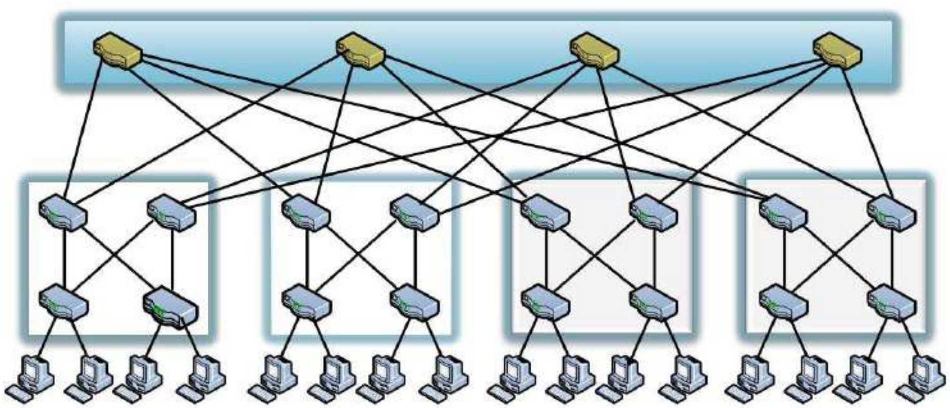
PortLand: Main Assumption

Hierarchical structure of data center networks:

They are multi-level, multi-rooted trees



Cisco Recommended Configuration



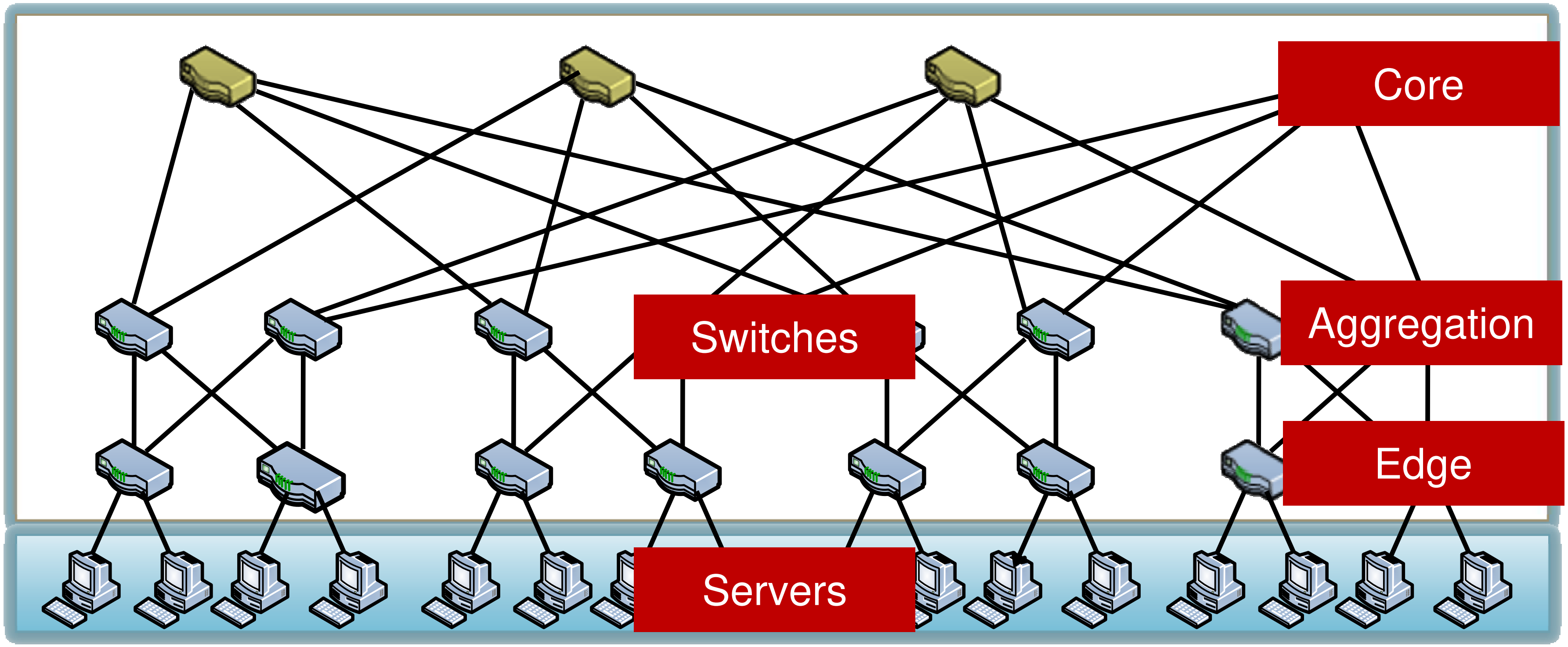
Fat Tree

Center for Networked Systems

PortLand: Scalability Challenges

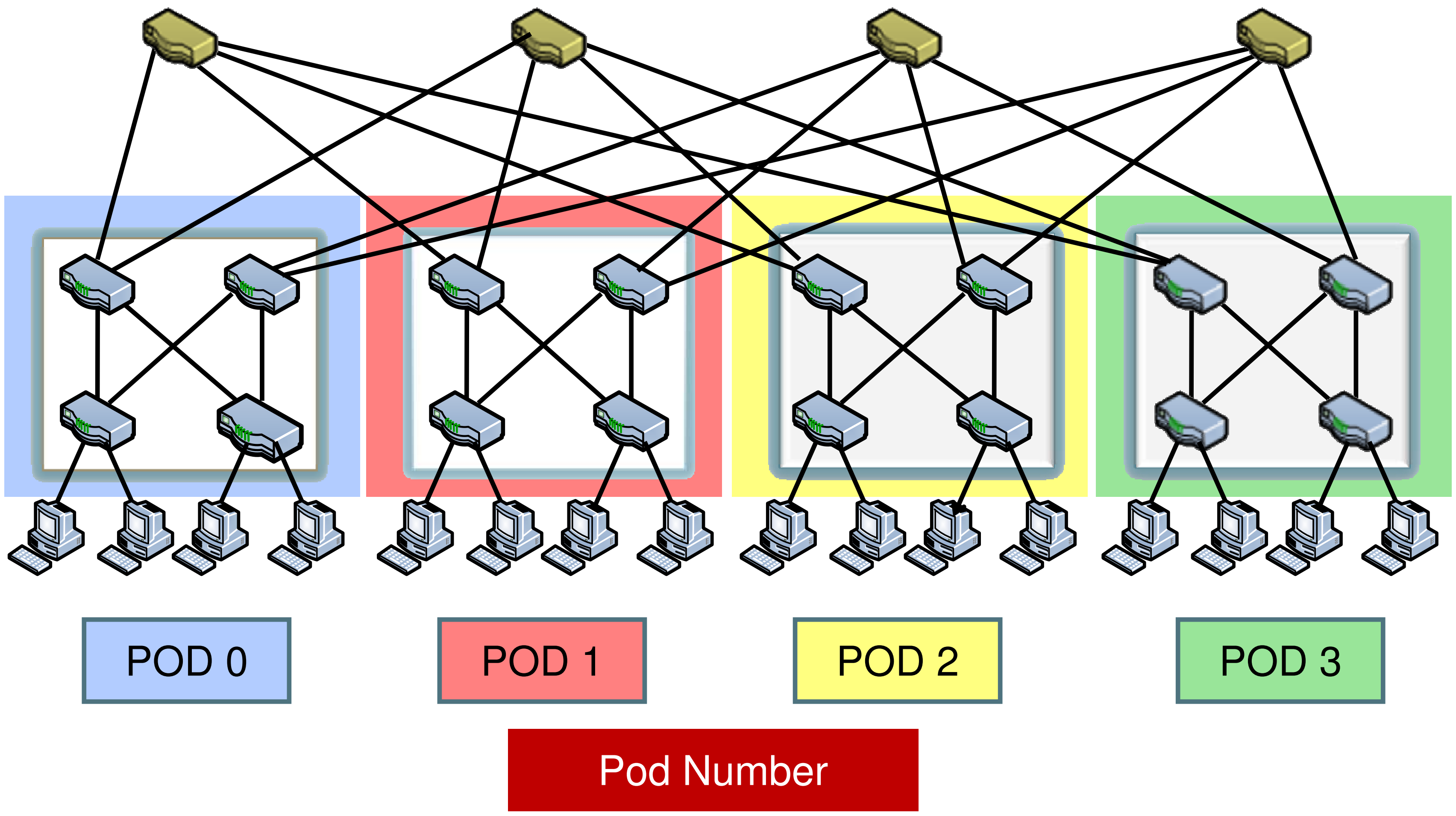
Challenge	State Of Art
Address Resolution	Broadcast based
Routing	Broadcast based
Forwarding	Large switch state

Data Center Network



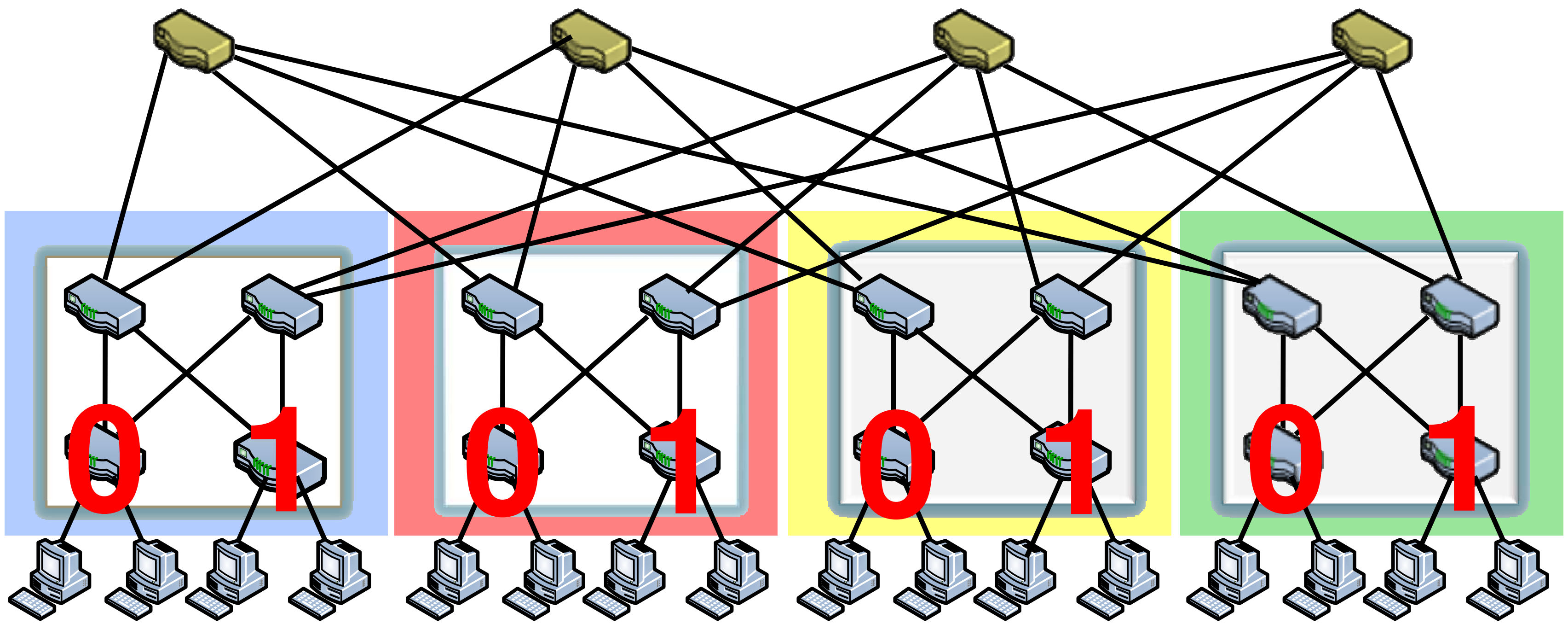
Center for Networked Systems
UC San Diego

Imposing Hierarchy On A Multi-Rooted Tree



Center for Networked Systems

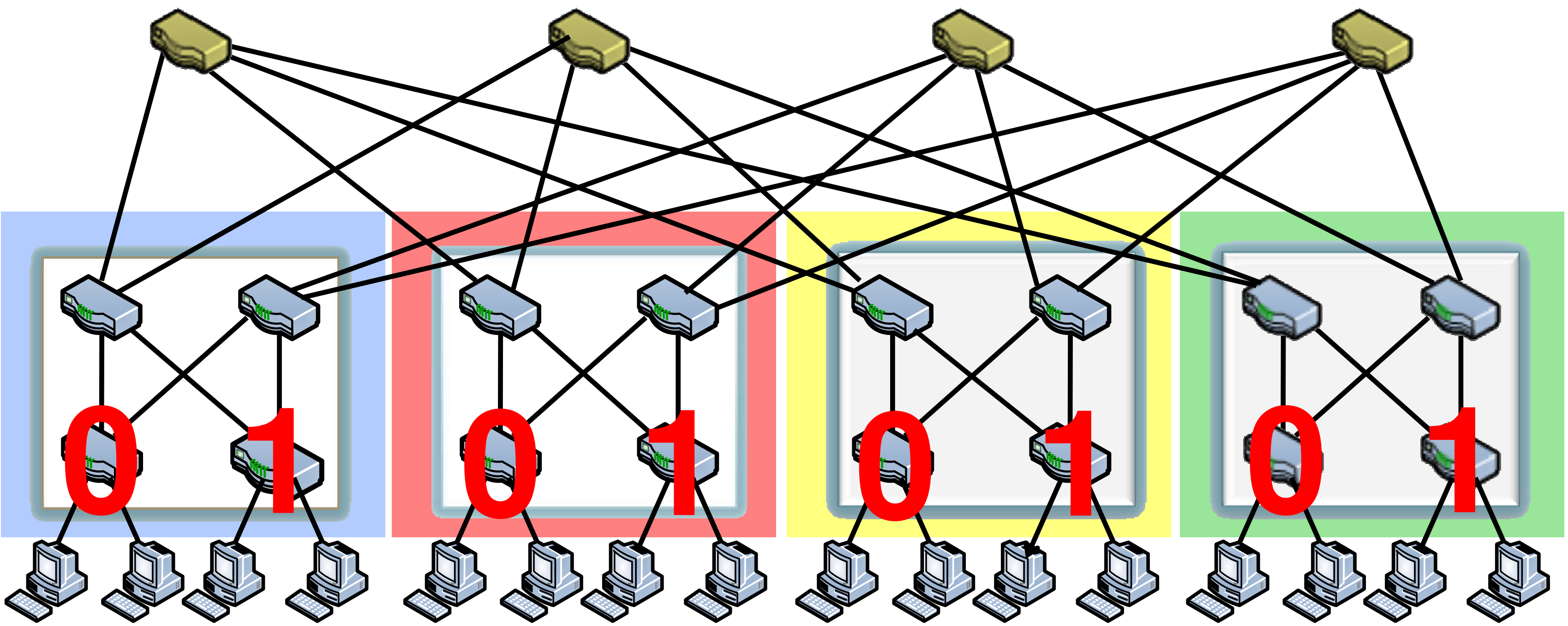
Imposing Hierarchy On A Multi-Rooted Tree



Position Number

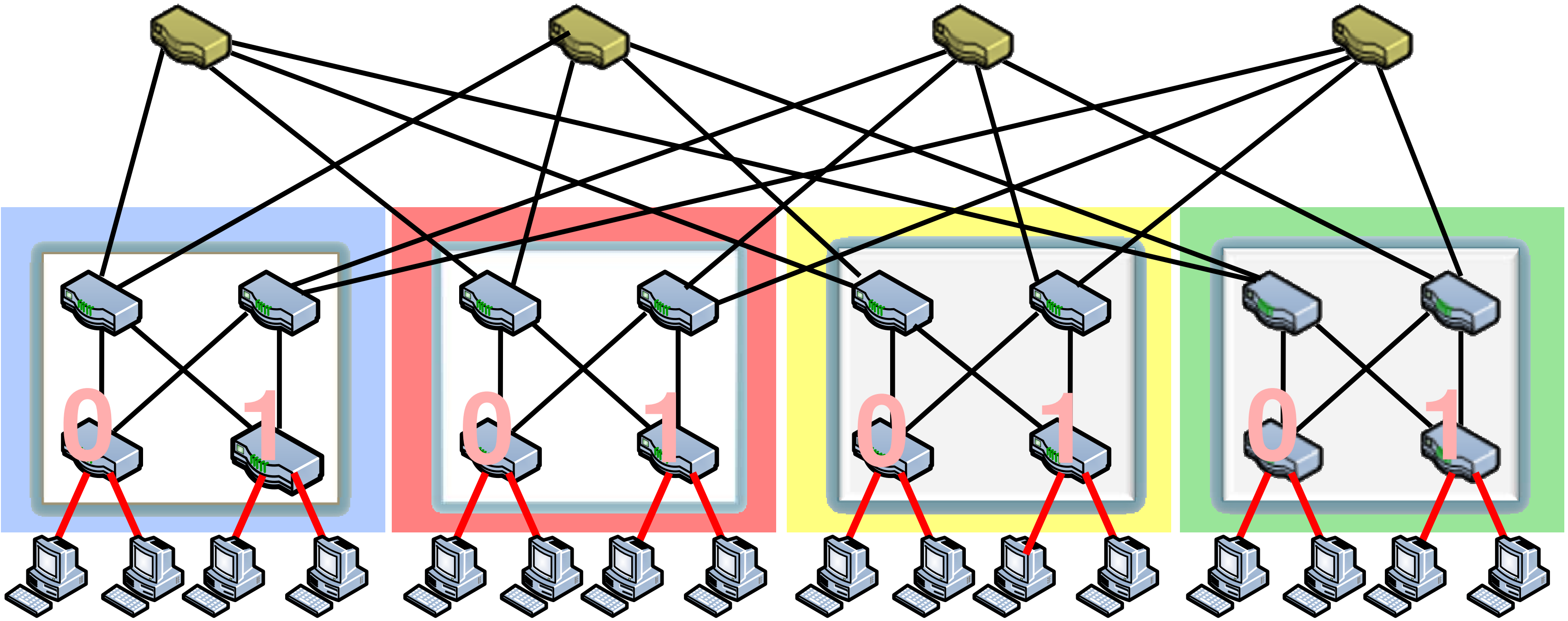
Center for Networked Systems

Imposing Hierarchy On A Multi-Rooted Tree



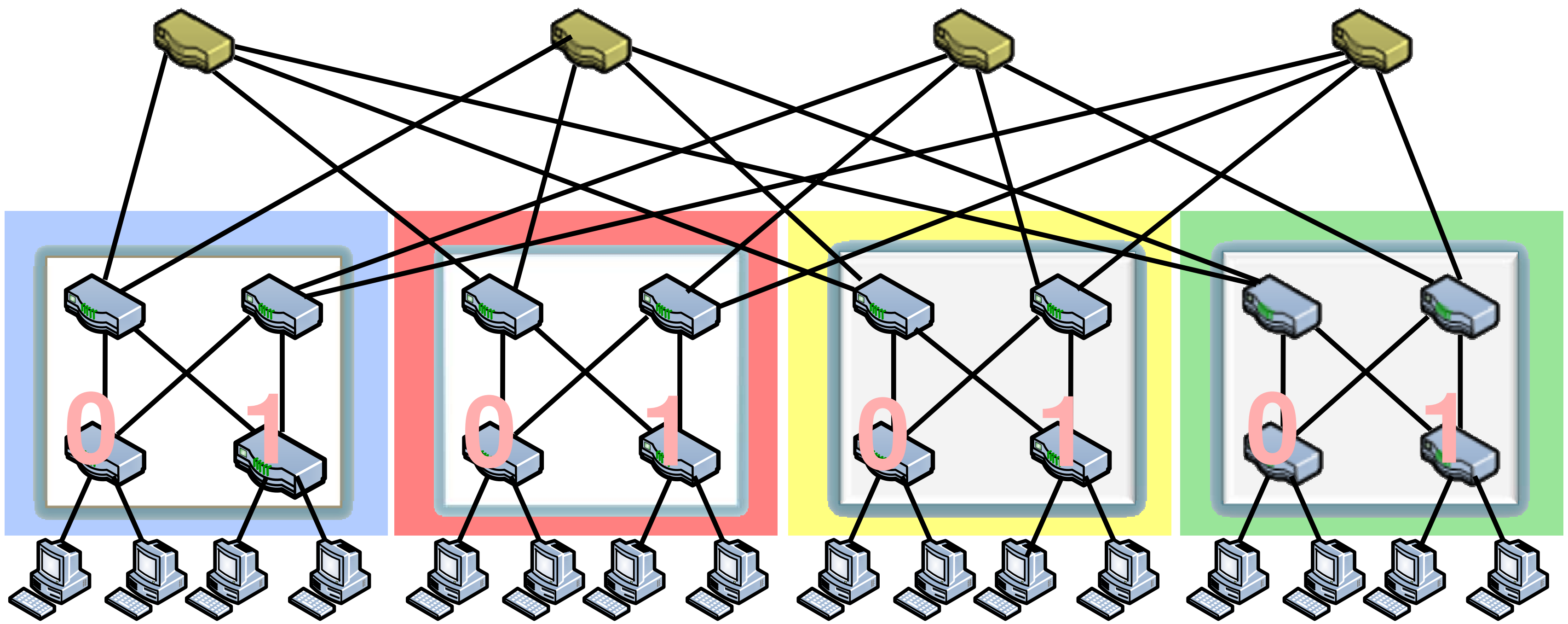
PMAC: pod.position.port.vmid

Imposing Hierarchy On A Multi-Rooted Tree



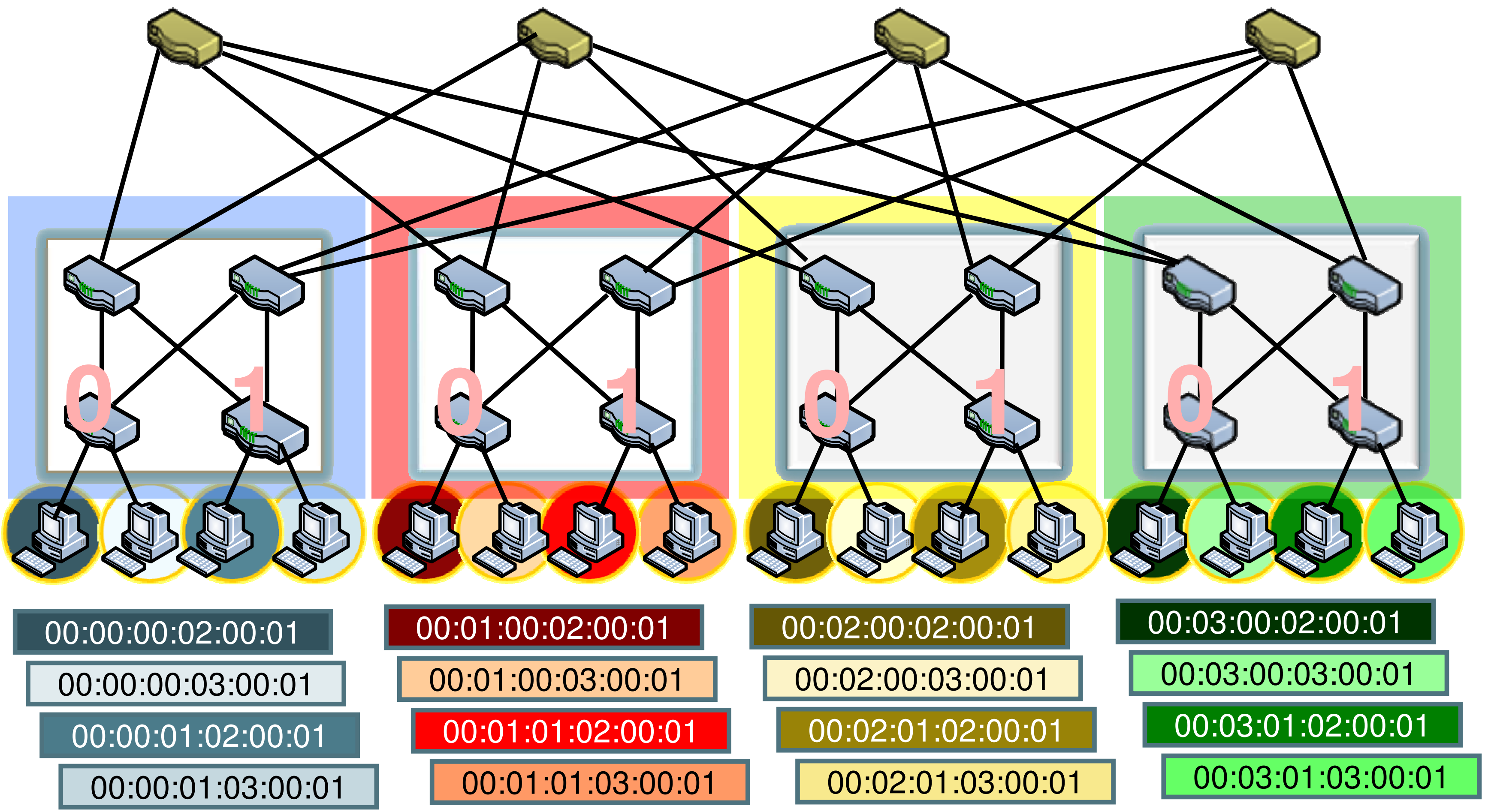
PMAC: pod.position.port.vmid

Imposing Hierarchy On A Multi-Rooted Tree



PMAC: pod.position.port.vmid

Imposing Hierarchy On A Multi-Rooted Tree



PROXY-BASED ARP

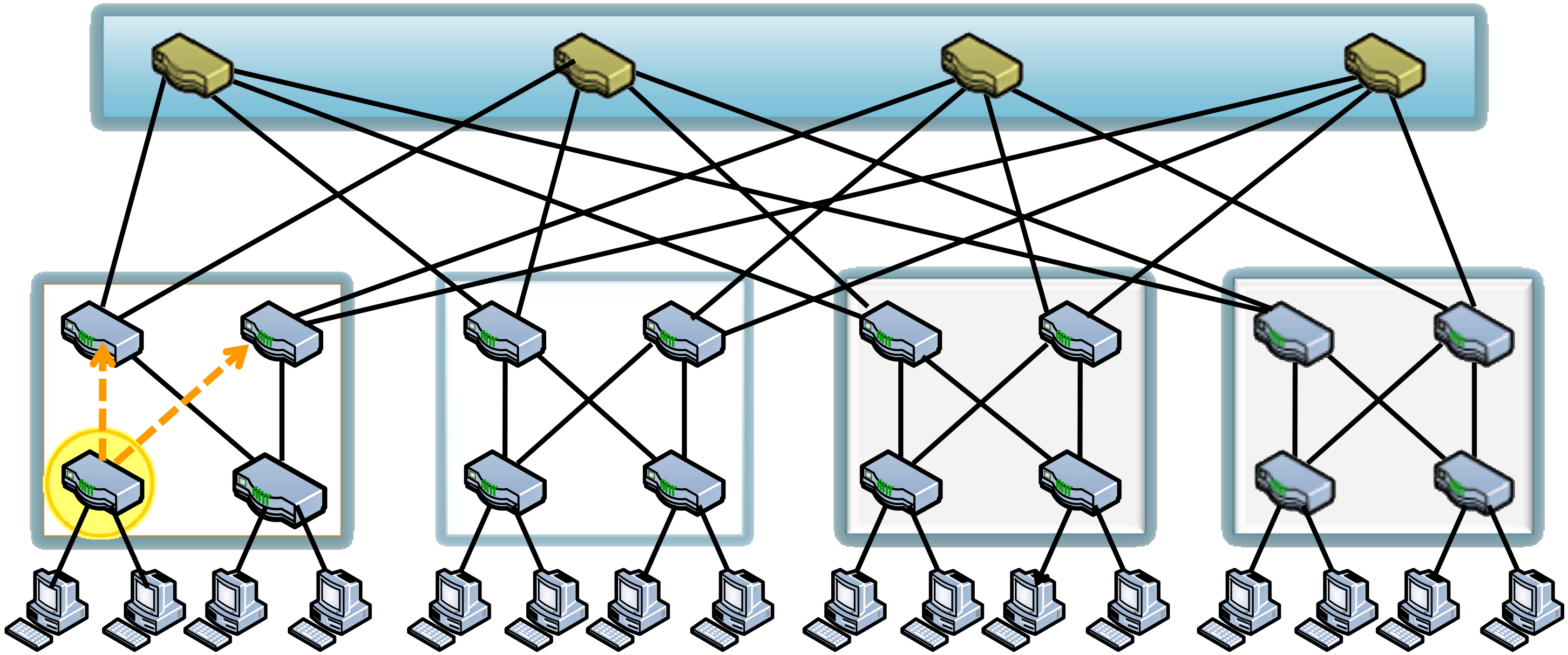
- When an edge switch sees a new AMAC, it assigns a PMAC to the host
- It then communicates the PMAC to IP mapping to the fabric manager.
- The fabric manager servers as a proxy-ARP agent, and answers ARP queries

PortLand: Location Discovery Protocol

- Location Discovery Messages (LDMs) exchanged between neighboring switches
- Switches self-discover location on boot up

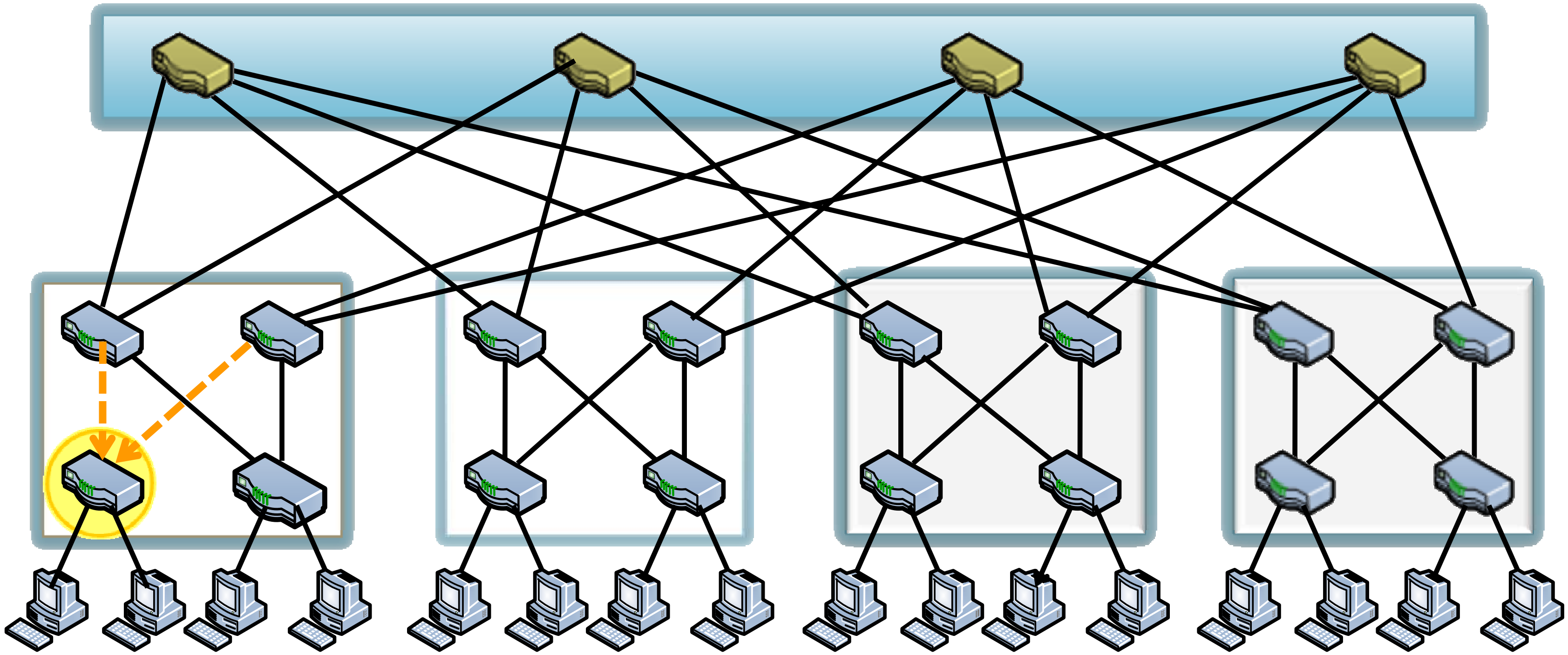
Location characteristic	Technique
1) Tree level / Role	Based on neighbor identity
2) Pod number	Aggregation and edge switches agree on pod number
3) Position number	Aggregation switches help edge switches choose unique position number

PortLand: Location Discovery Protocol



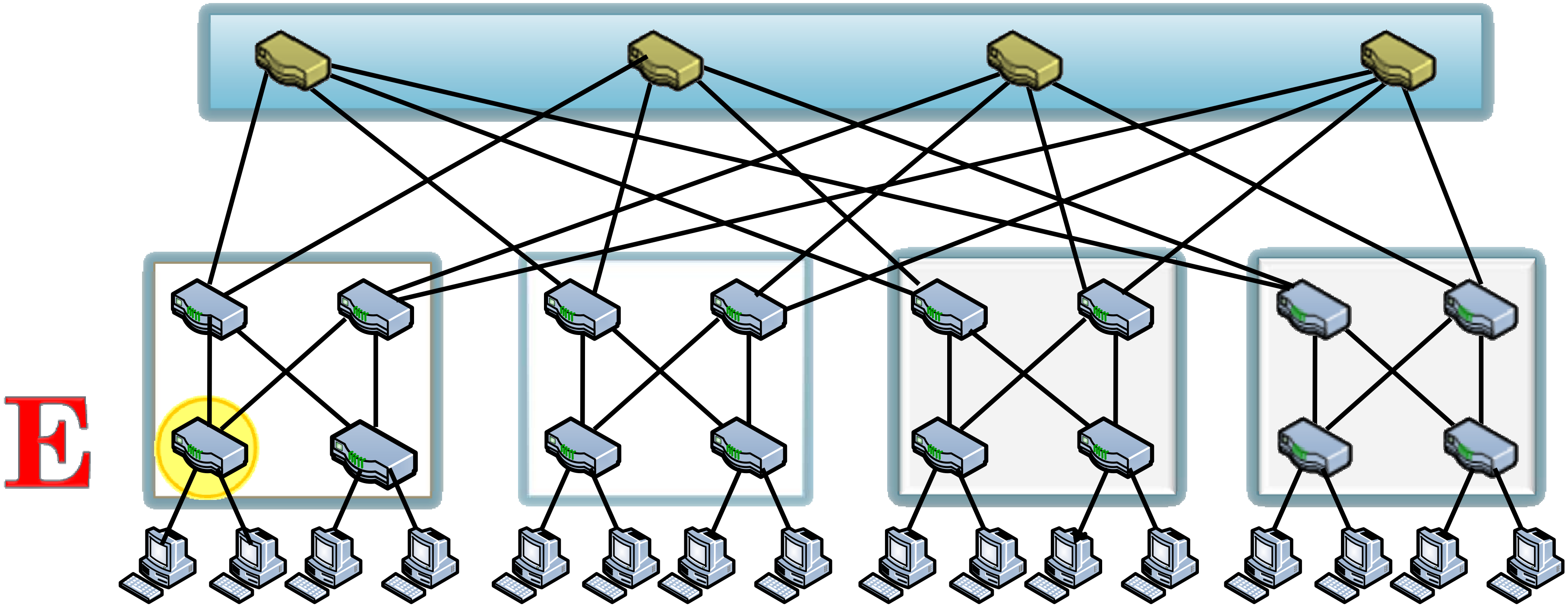
Switch Identifier	Pod Number	Position	Tree Level
A0:B1:FD:56:32:01	??	??	??

PortLand: Location Discovery Protocol



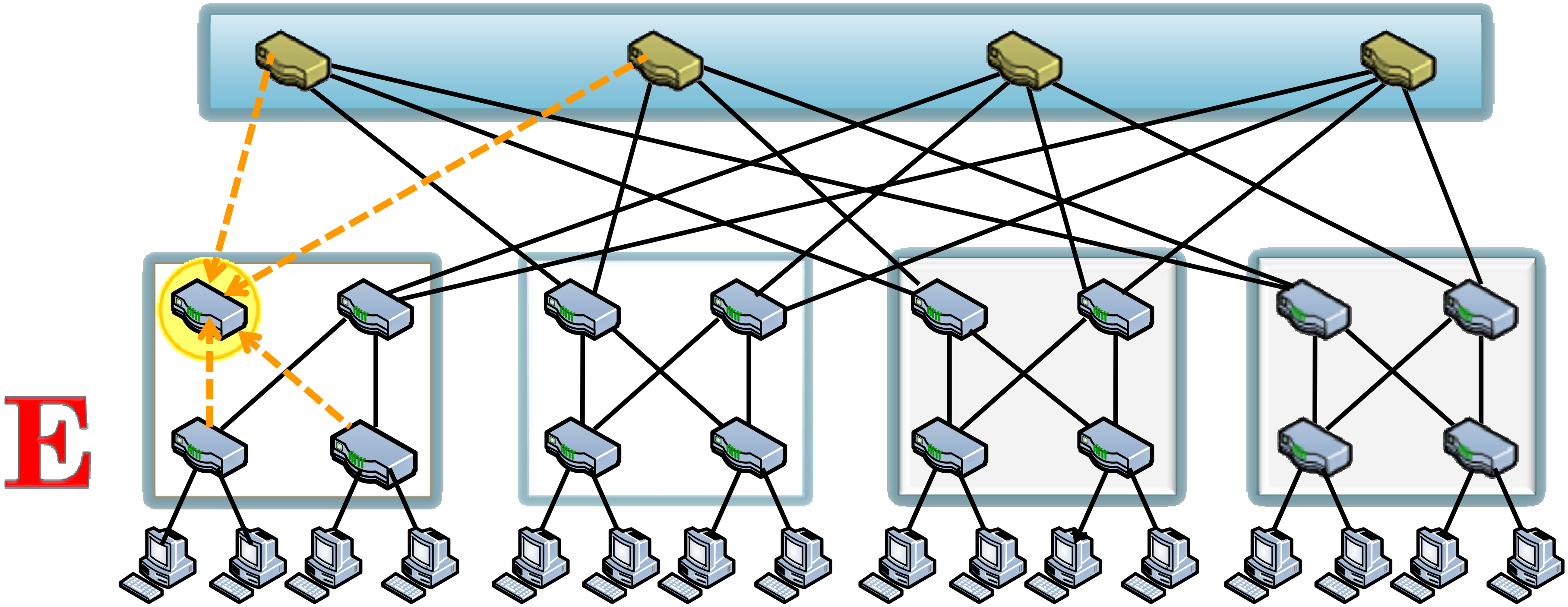
Switch Identifier	Pod Number	Position	Tree Level
A0:B1:FD:56:32:01	??	??	??

PortLand: Location Discovery Protocol



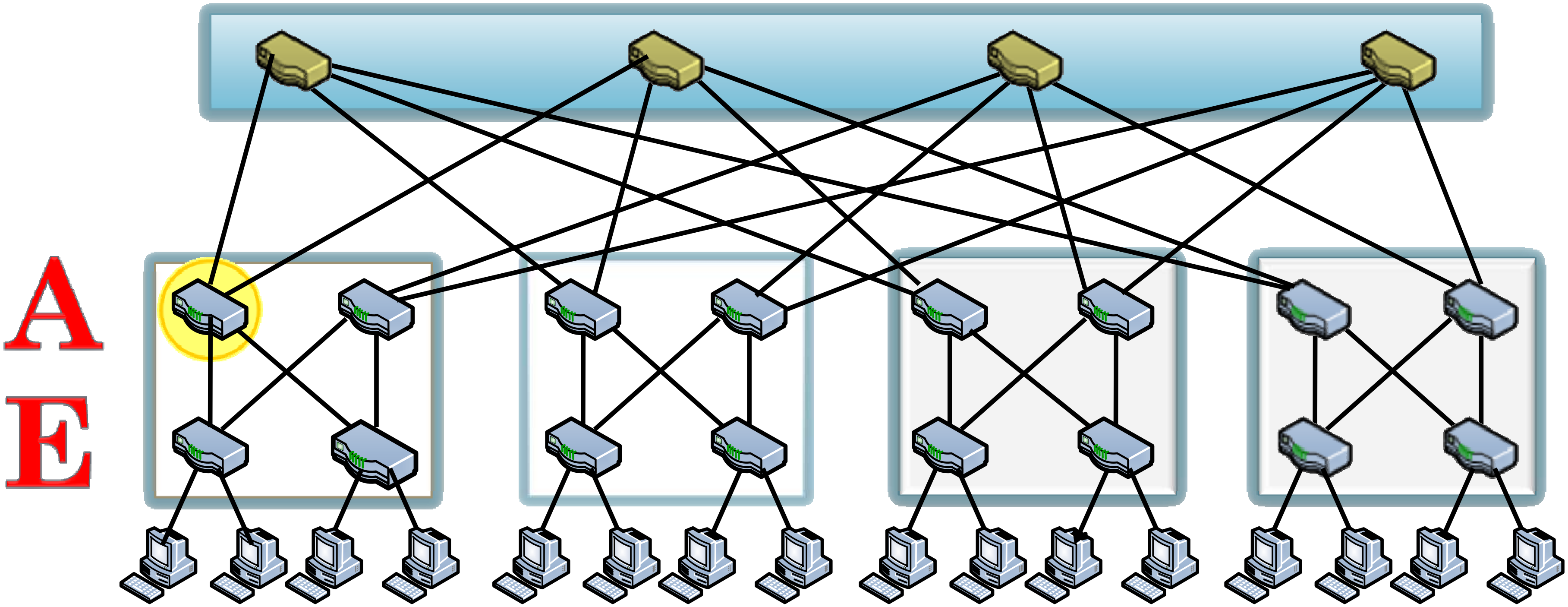
Switch Identifier	Pod Number	Position	Tree Level
A0:B1:FD:56:32:01	??	??	0

PortLand: Location Discovery Protocol



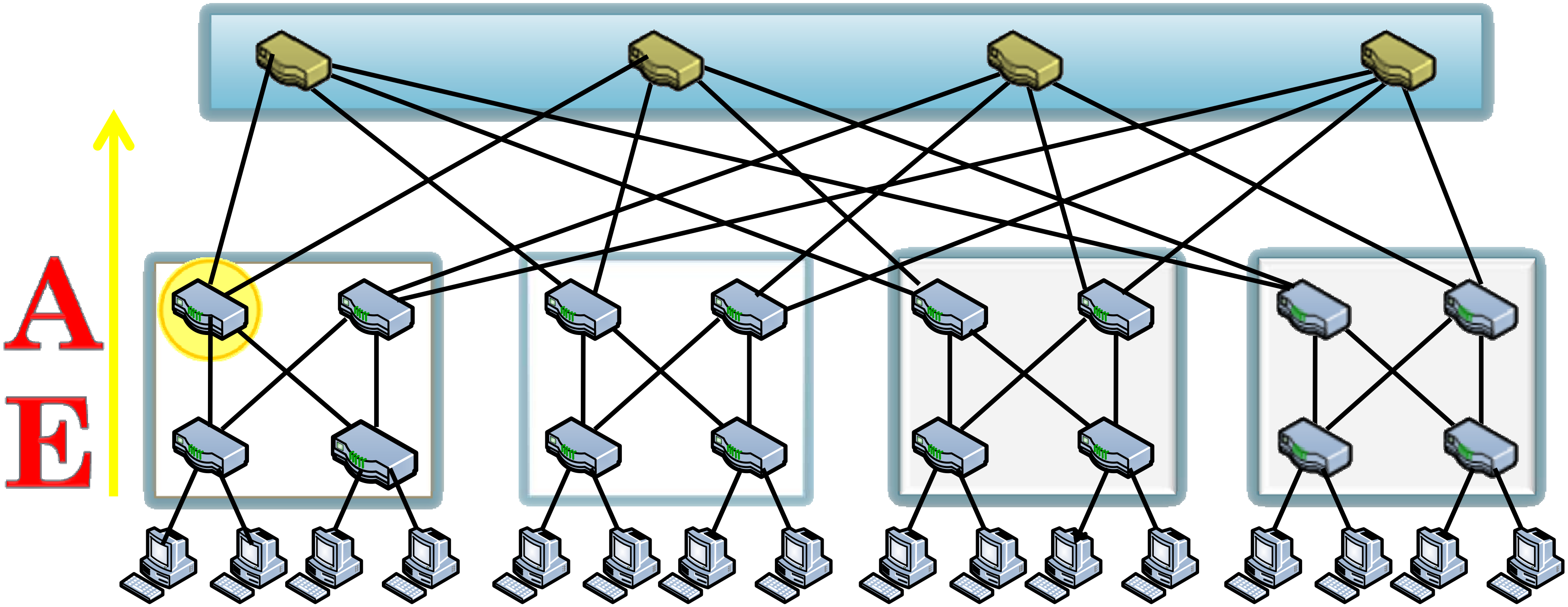
Switch Identifier	Pod Number	Position	Tree Level
B0:A1:FD:57:32:01	??	??	??

PortLand: Location Discovery Protocol



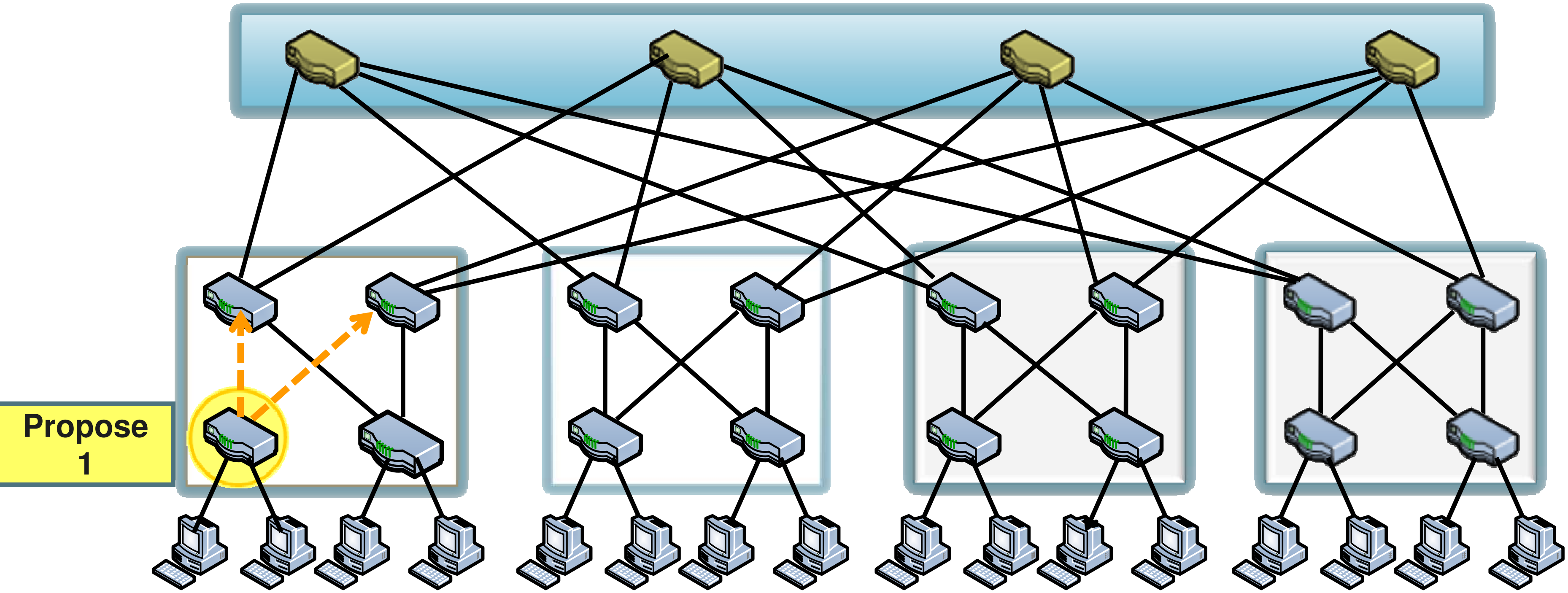
Switch Identifier	Pod Number	Position	Tree Level
B0:A1:FD:57:32:01	??	??	1

PortLand: Location Discovery Protocol



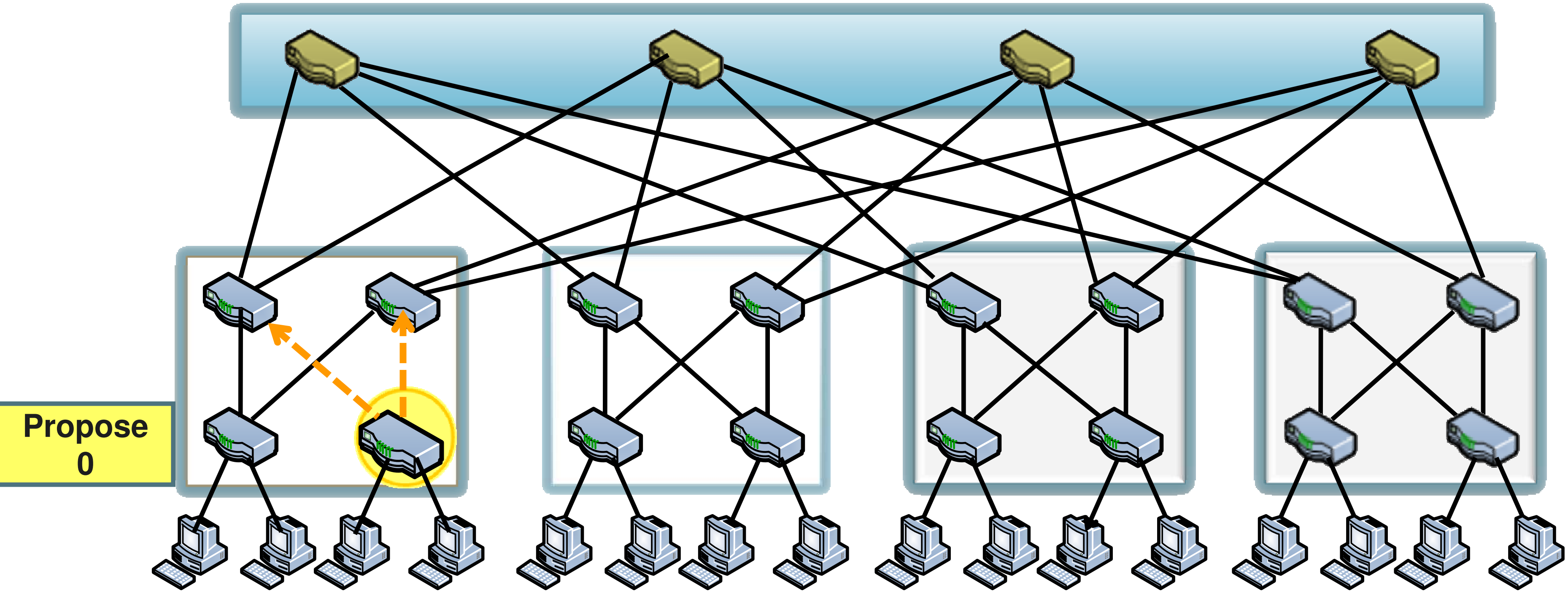
Switch Identifier	Pod Number	Position	Tree Level
B0:A1:FD:57:32:01	??	??	1

PortLand: Location Discovery Protocol



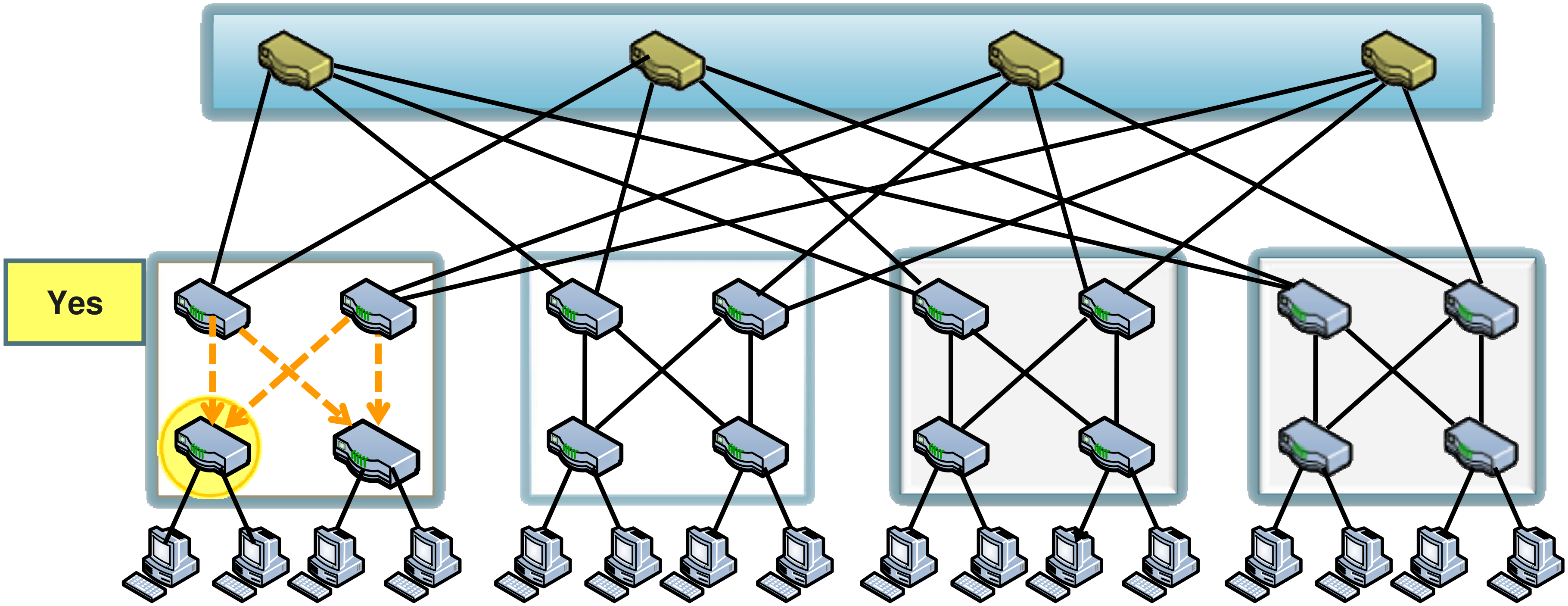
Switch Identifier	Pod Number	Position	Tree Level
A0:B1:FD:56:32:01	??	??	0

PortLand: Location Discovery Protocol



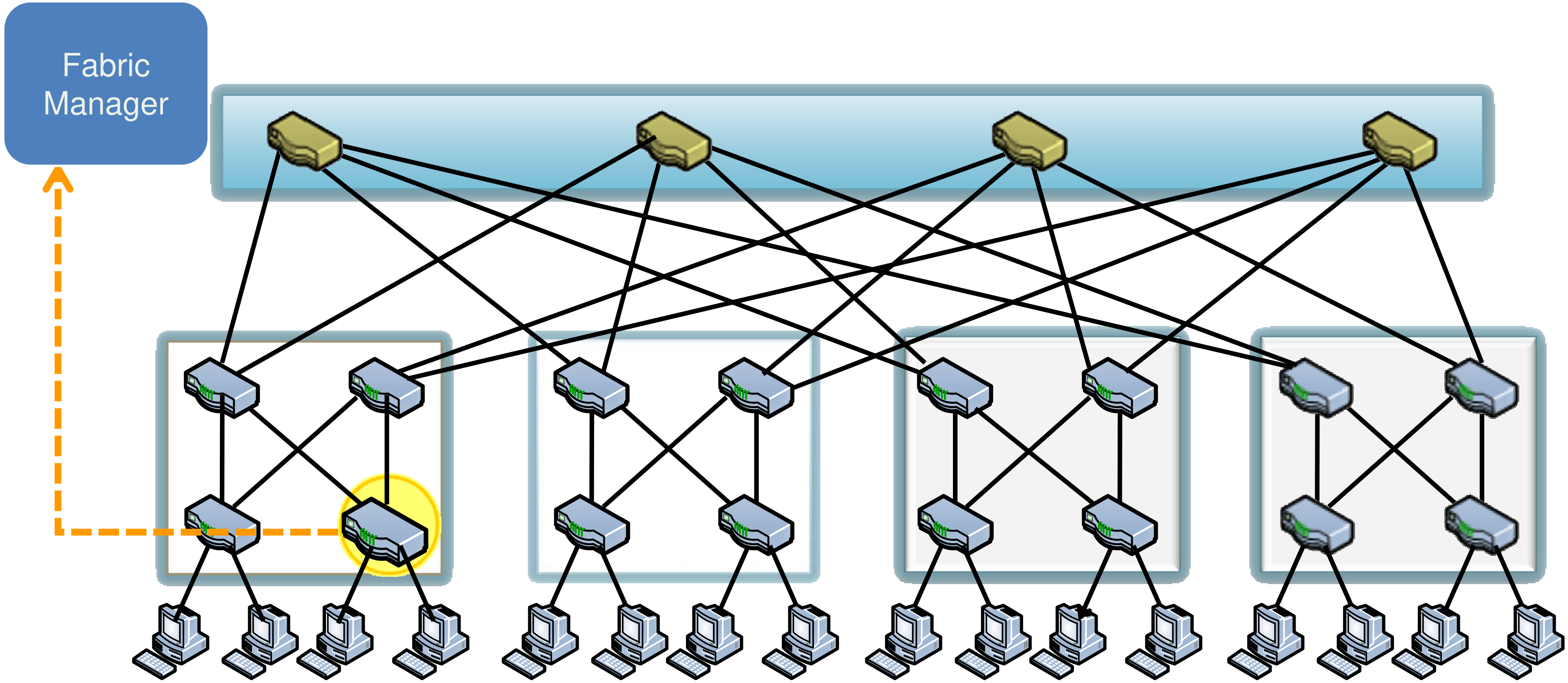
Switch Identifier	Pod Number	Position	Tree Level
D0:B1:AD:56:32:01	??	??	0

PortLand: Location Discovery Protocol



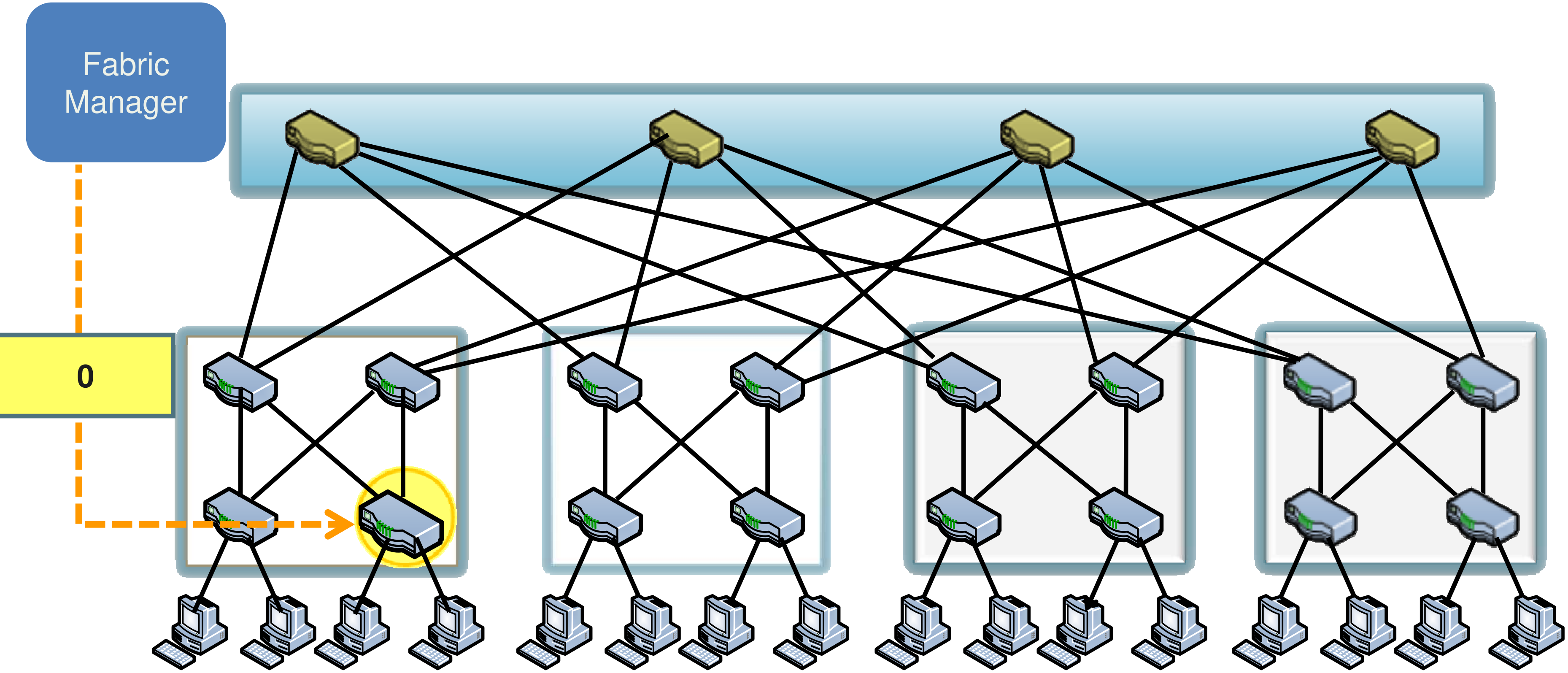
Switch Identifier	Pod Number	Position	Tree Level
A0:B1:FD:56:32:01	??	1	0

PortLand: Location Discovery Protocol



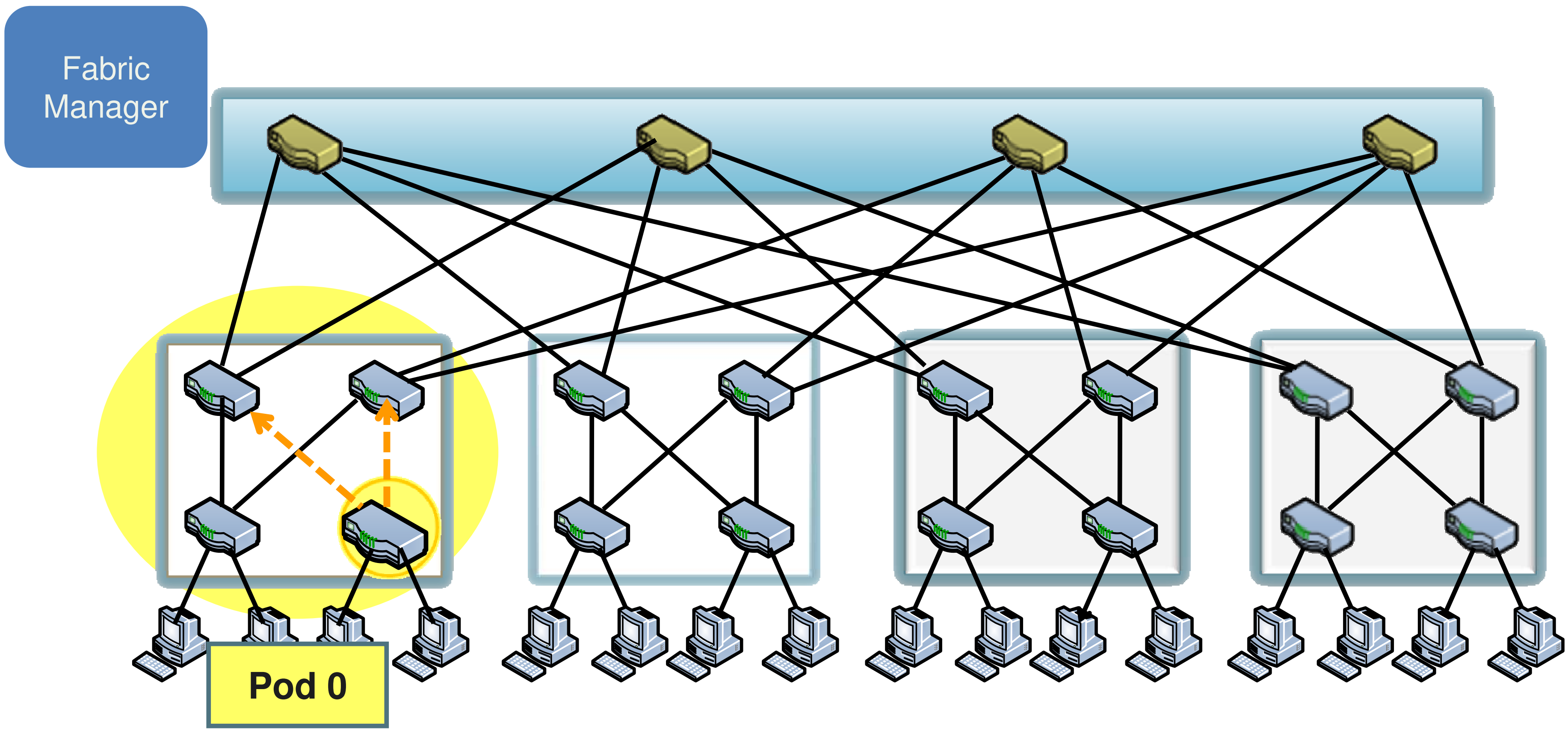
Switch Identifier	Pod Number	Position	Tree Level
D0:B1:AD:56:32:01	??	0	0

PortLand: Location Discovery Protocol



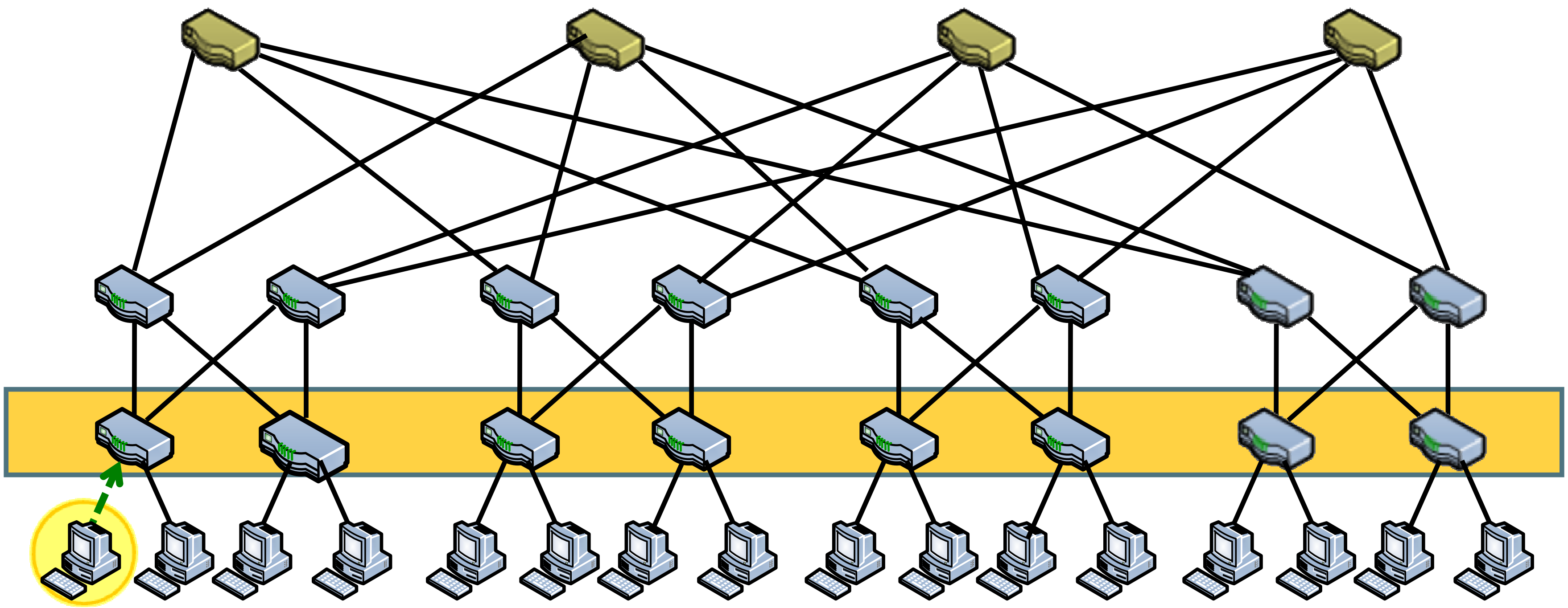
Switch Identifier	Pod Number	Position	Tree Level
D0:B1:AD:56:32:01	??	0	0

PortLand: Location Discovery Protocol



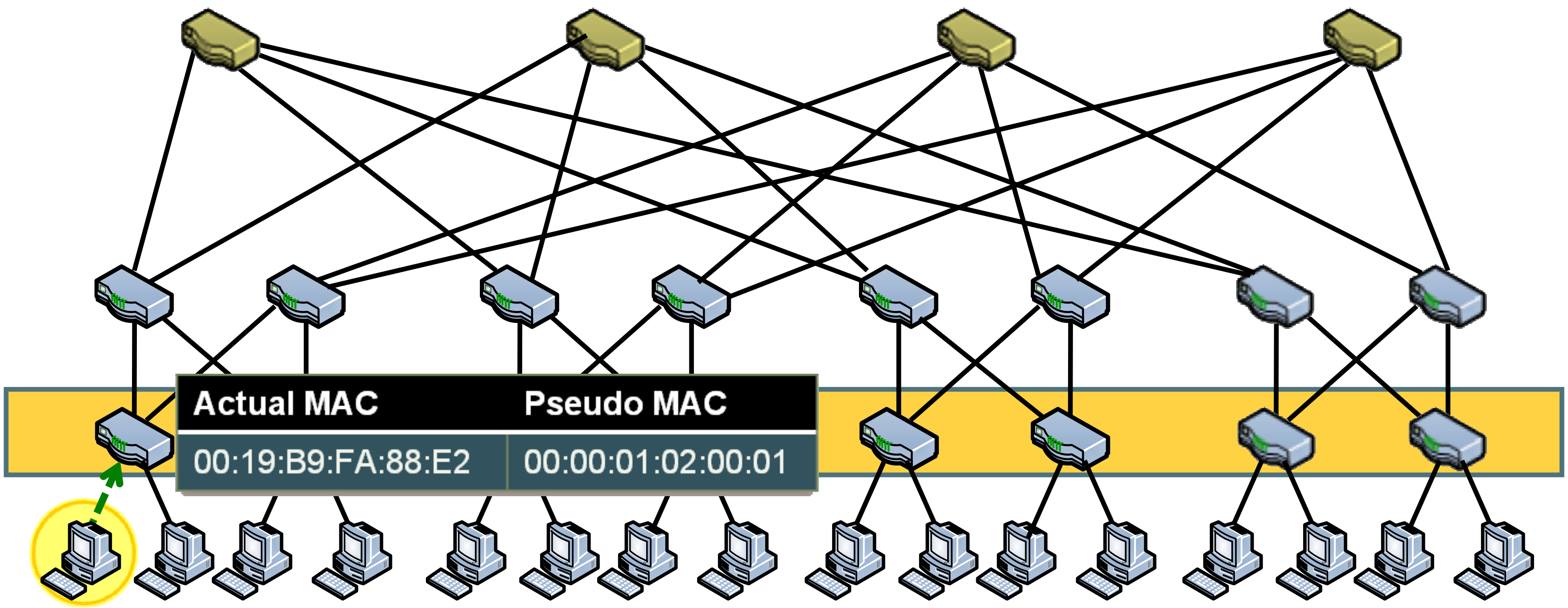
Switch Identifier	Pod Number	Position	Tree Level
D0:B1:AD:56:32:01	0	0	0

PortLand: Name Resolution



Intercept all ARP packets

PortLand: Name Resolution

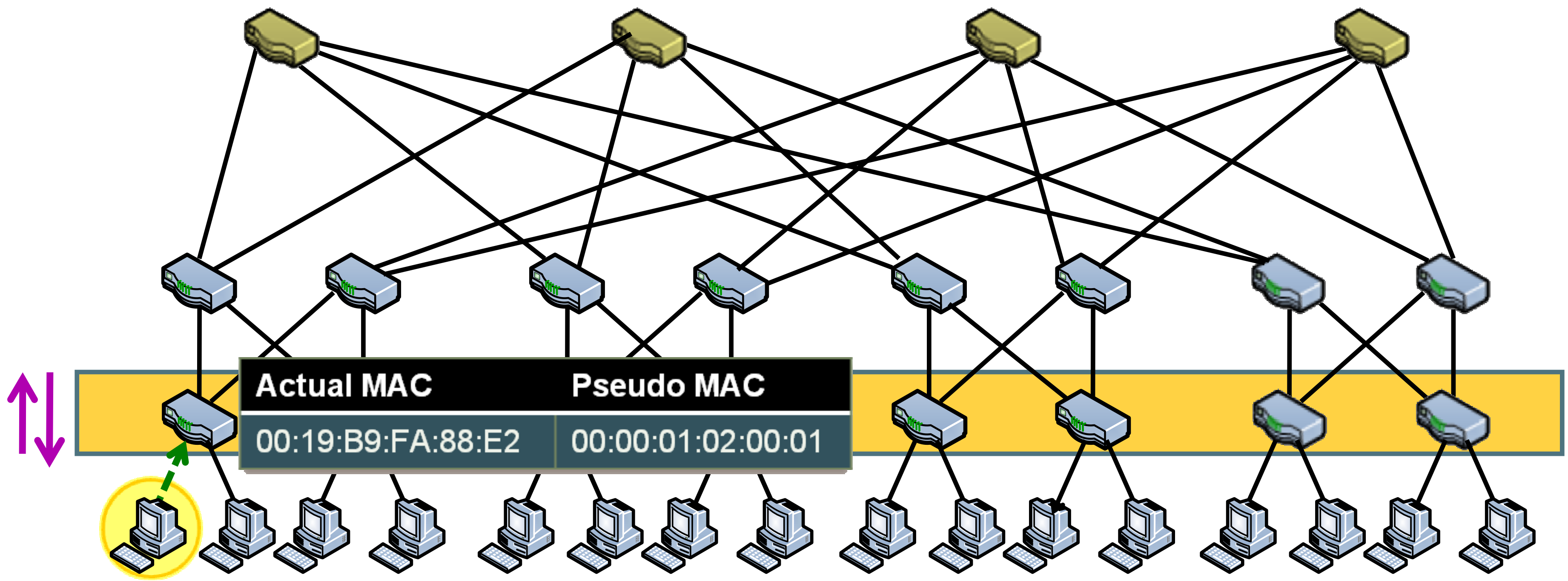


Actual MAC	Pseudo MAC
00:19:B9:FA:88:E2	00:00:01:02:00:01

Intercept all ARP packets

Assign new end hosts with PMACs

PortLand: Name Resolution



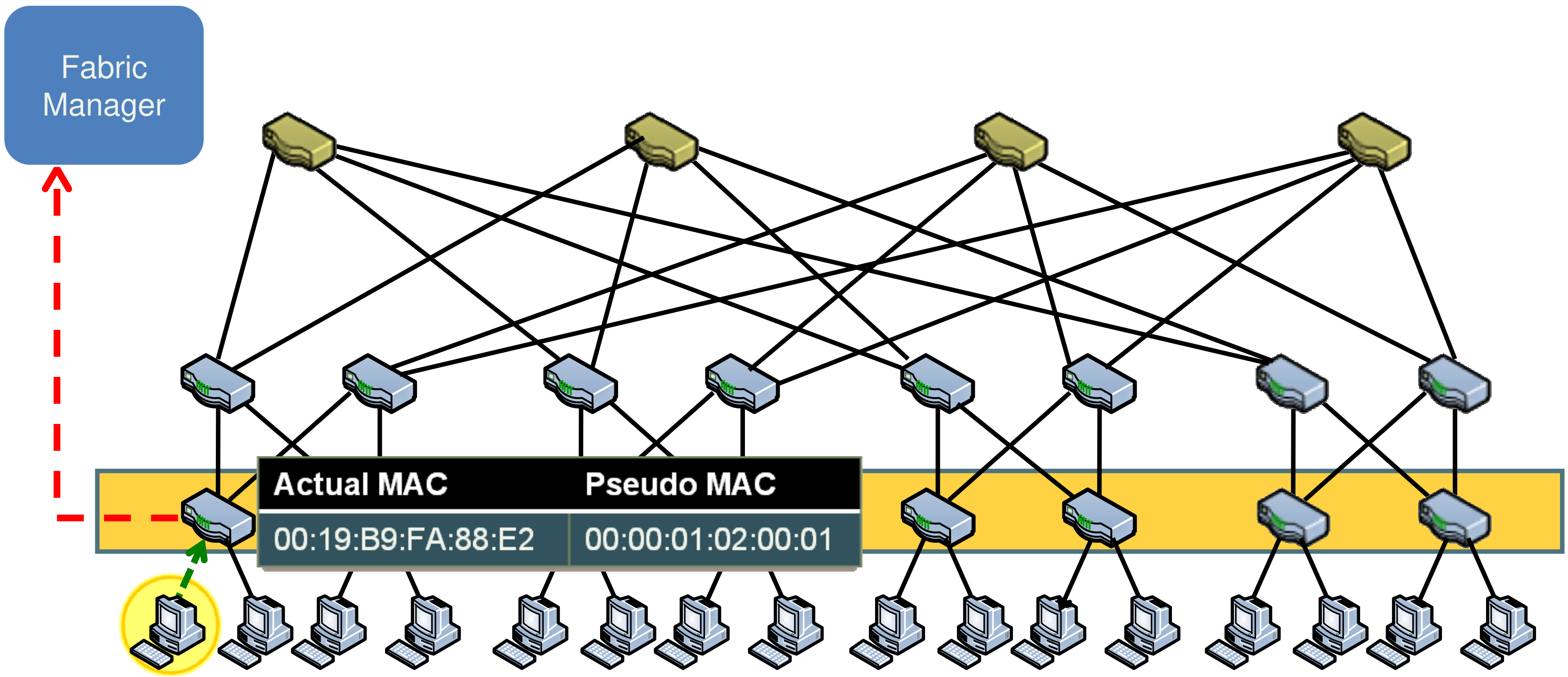
Actual MAC	Pseudo MAC
00:19:B9:FA:88:E2	00:00:01:02:00:01

Intercept all ARP packets

Assign new end hosts with PMACs

Rewrite MAC for packets entering and exiting network

PortLand: Name Resolution

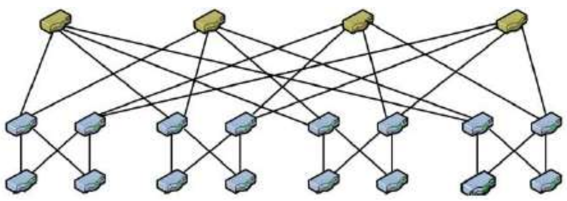


PortLand: Fabric Manager

Fabric Manager

IP	Pseudo MAC
10.5.1.2	00:00:01:02:00:01
10.2.4.5	00:02:00:02:00:01

ARP mappings

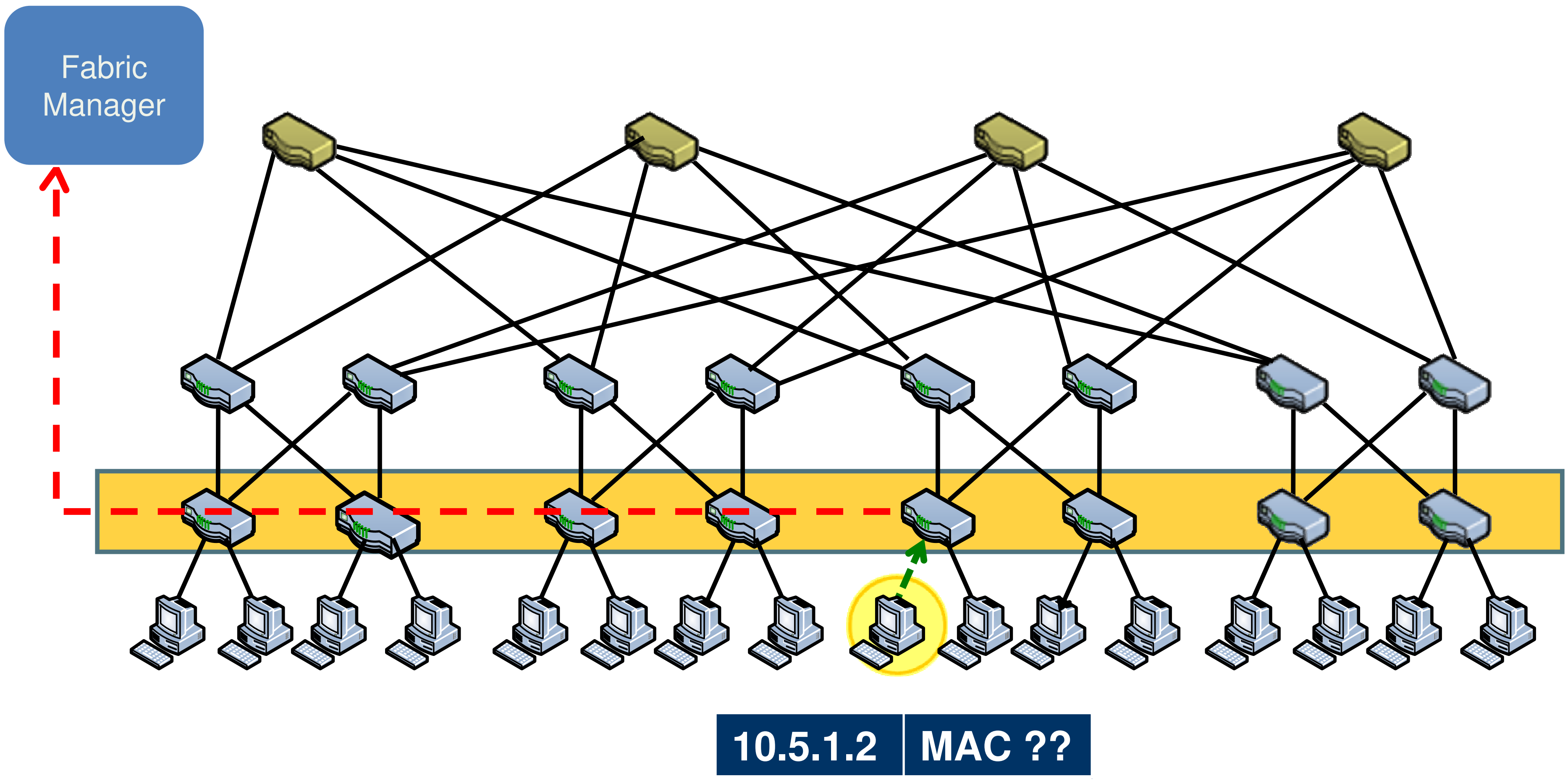


Network map

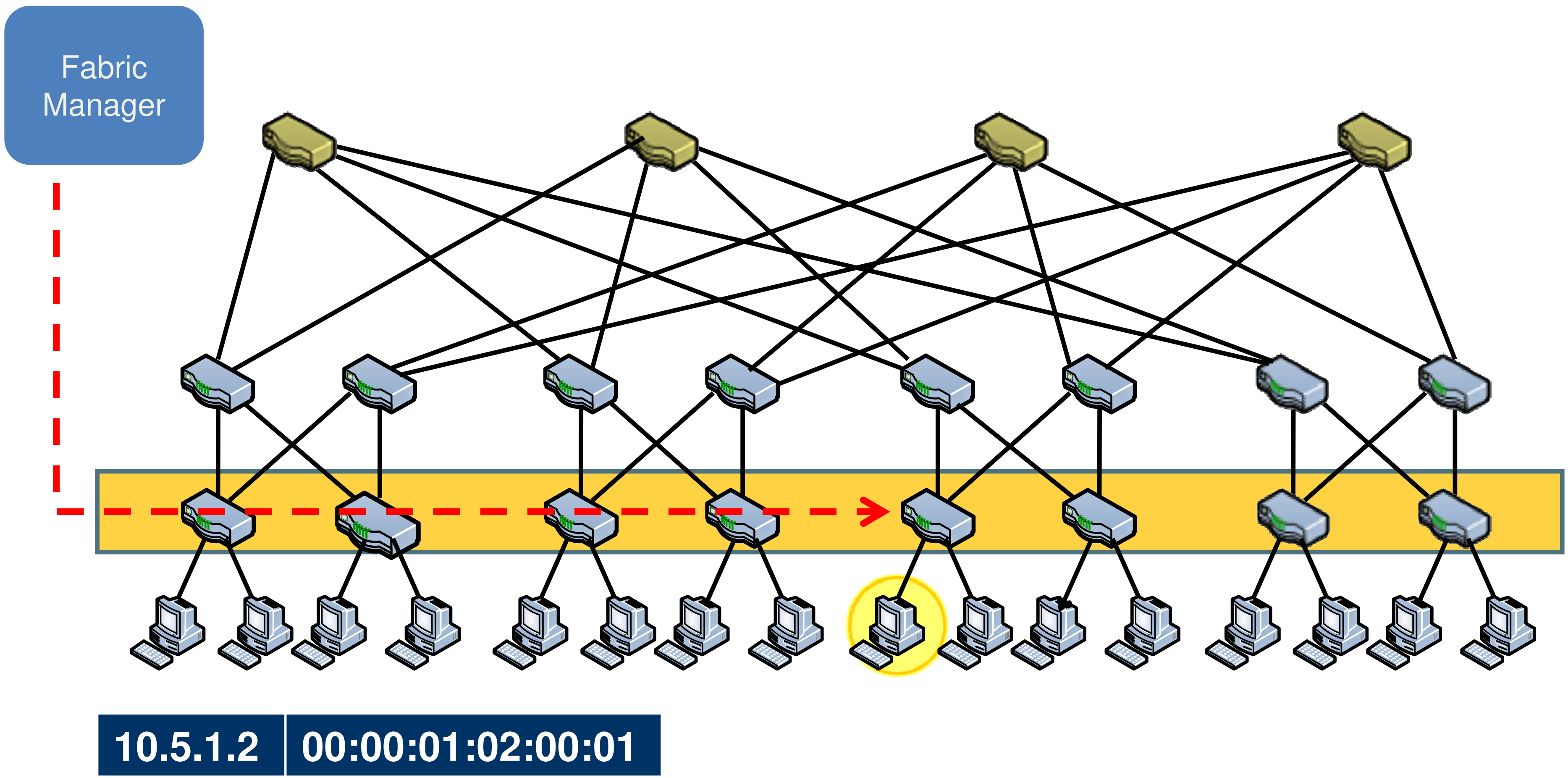
Soft state

~~Administrator configuration~~

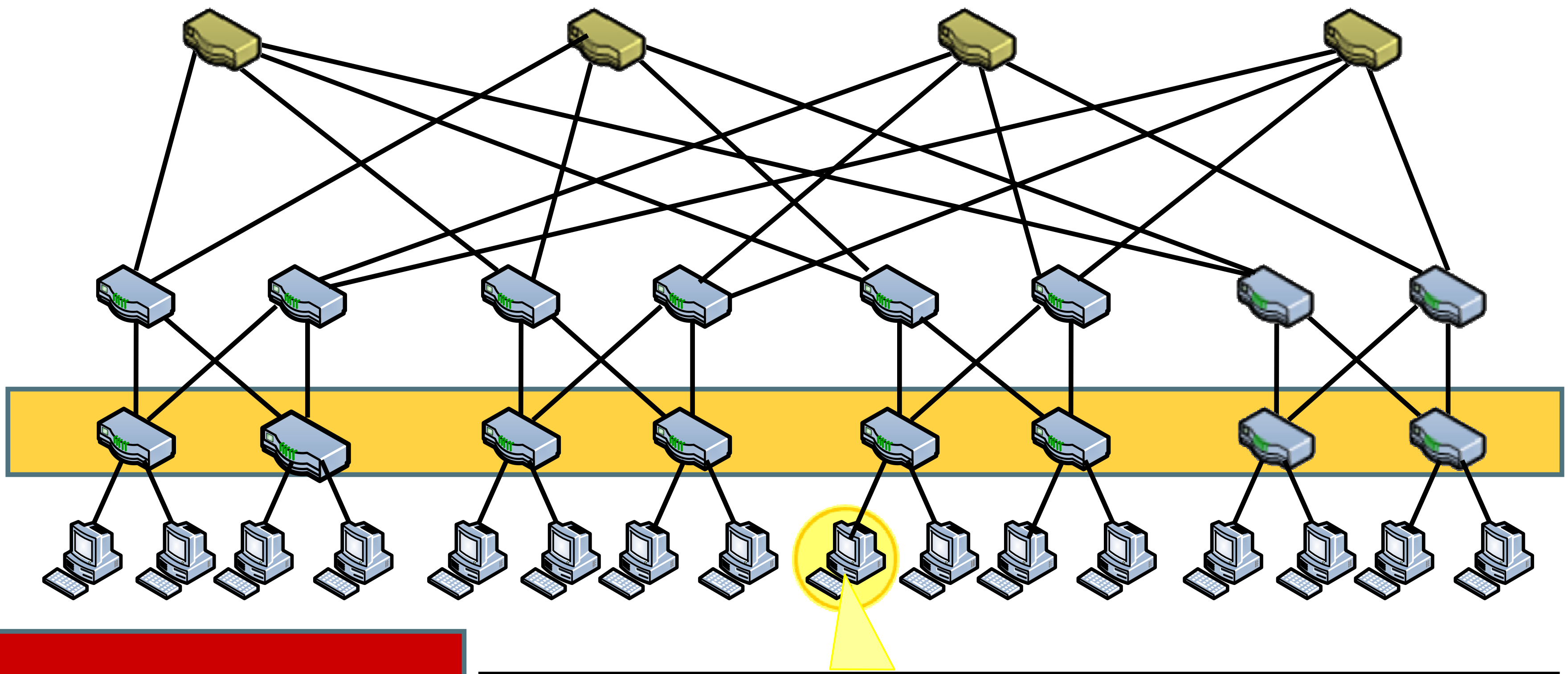
PortLand: Name Resolution



PortLand: Name Resolution



PortLand: Name Resolution



ARP replies contain only PMAC

Address	HWtype	HWAddress	Flags	Mask	Iface
10.5.1.2	ether	00:00:01:02:00:01	C		eth1

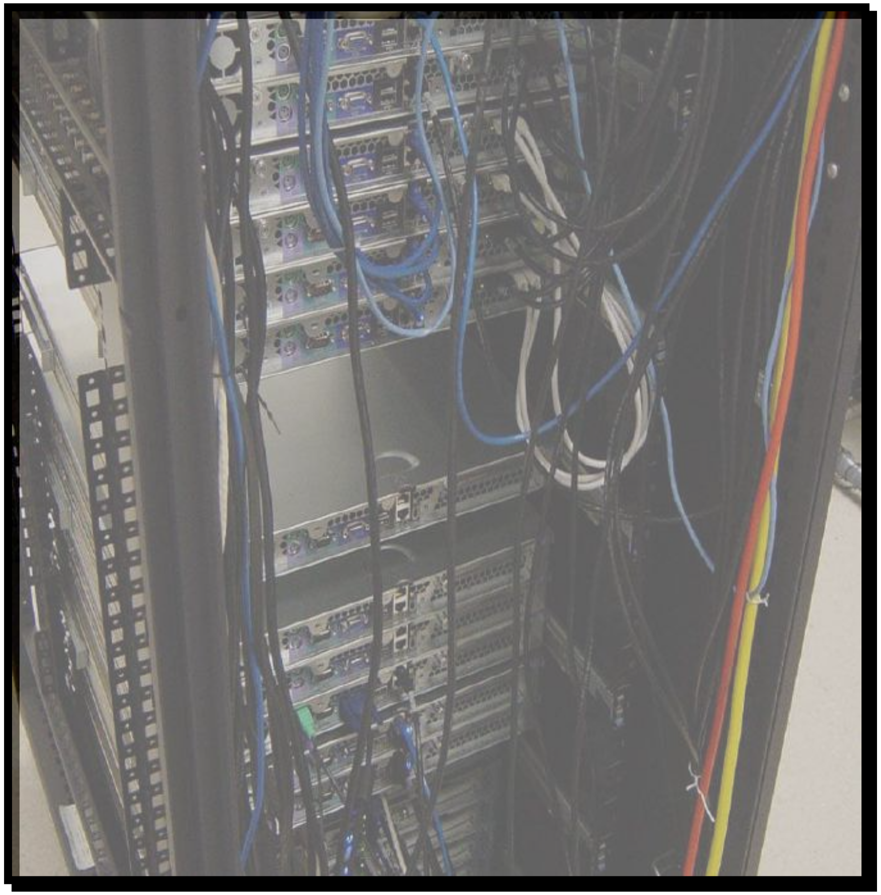
PROVABLY LOOP-FREE FORWARDING

- Switches populate their forwarding tables after establishing local positions
- Core switches forward according to pod numbers
- Aggregation switches forward packets destined to the same pod to edge switches, to other pods to core switches
- Edge switches forward packets to the corresponding hosts

FAULT TOLERANT ROUTING

- LDP exchanges serve as keepalive
- A switch reports a dead link to the fabric manager (FM)
- The FM updates its faulty link matrix, and informs affected switches the failure
- Affected switches reconfigure their forwarding tables to bypass the failed link
- → No broadcasting of the failure

Portland Prototype



- 20 OpenFlow NetFPGA switches
- TCAM + SRAM for flow entries
- Software MAC rewriting
- 3 tiered fat-tree
- 16 end hosts

PortLand: Evaluation

Measurements	Configuration	Results
Network convergence time	Keepalive frequency = 10 ms Fault detection time = 50 ms	65 ms
TCP convergence time	RTO_{min} = 200ms	~200 ms
Multicast convergence time		110ms
TCP convergence with VM migration	RTO_{min} = 200ms	~200 ms – 600 ms
Control traffic to fabric manager	27,000+ hosts, 100 ARPs / sec per host	400 Mbps → non trivial
CPU requirements of fabric manager	27,000+ hosts, 100 ARPs / sec per host	70 CPU cores → non trivial

Summarizing PortLand

- PortLand is a single logical layer 2 data center network fabric that scales to millions of endpoints
- Modify network fabric to
  - Work with arbitrary operating systems and virtual machine monitors
  - Maintain the boundary between network and end-host administration
- Scale Layer 2 via network modifications
  - Unmodified switch hardware and end hosts

DISCUSSION

- Unmodified hosts: why is it desirable?
- Does location-based addressing necessarily mandate manual configuration?
  - Their own solution implies a big NO

ElasticTree: Saving Energy in Data Center Networks

Brandon Heller (Stanford)
Srini Seetharaman (Deutsche Telekom R&D, Los Altos)
Priya Mahadevan (Hewlett-Packard Labs, Palo Alto)
Yiannis Yiakoumis (Stanford)
Puneet Sharma (Hewlett-Packard Labs, Palo Alto)
Sujata Banerjee (Hewlett-Packard Labs, Palo Alto)
Nick McKeown (Stanford)

2

NETWORK CONSUMES MUCH POWER

Network Power Consumption: 6B kWh in 2006!

~267K average size homes

$50M a month

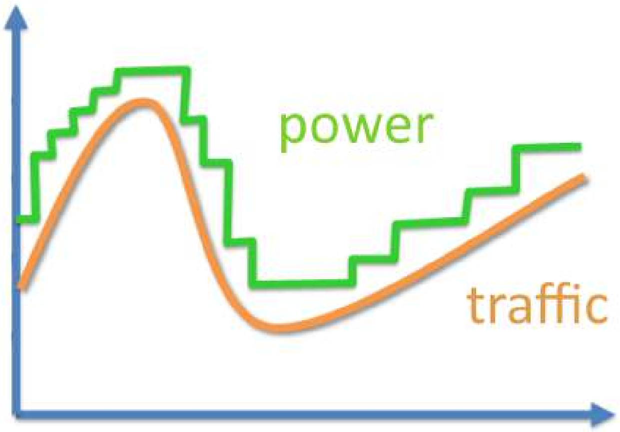
a ginormous amount of CO_2

2x increase projected for 2011

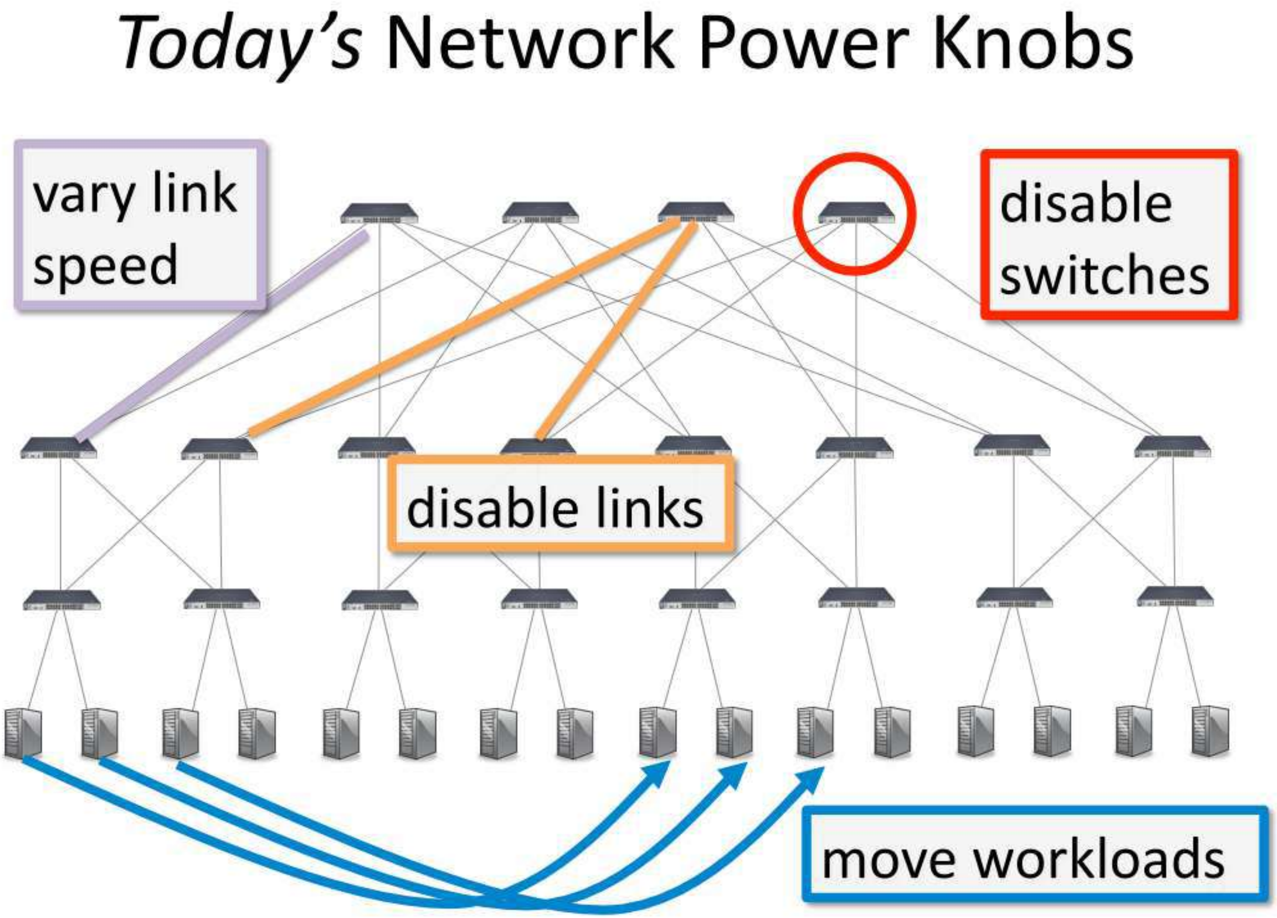
GOAL: ENERGY PROPORTION NETWORKING

End goal:

Create an
energy-proportional data center **network**
from non-proportional components.



APPROACH: TURN OFF UNNEEDED LINKS AND SWITCHES CAREFULLY AND AT SCALE



ELASTIC TREE ARCHITECTURE

Optimize for

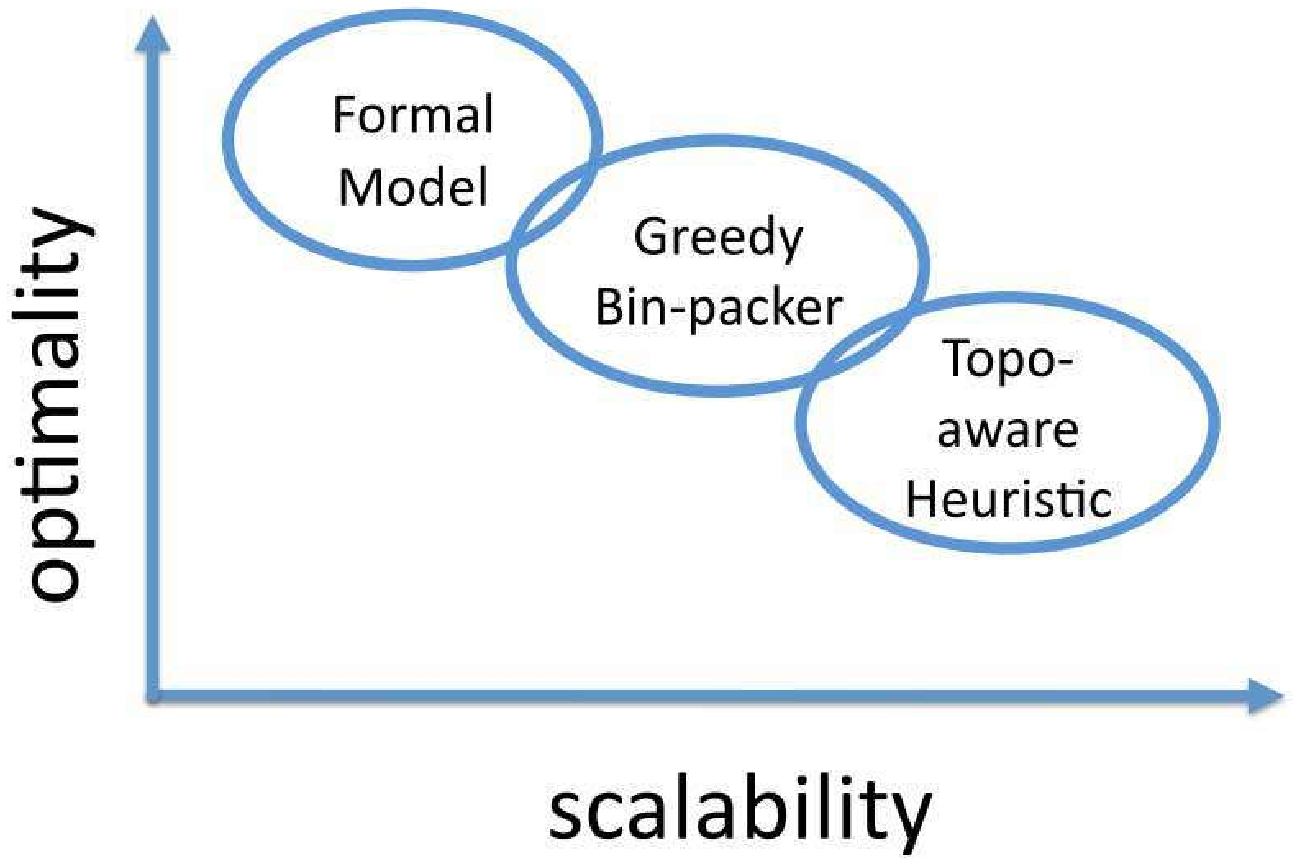
Power Efficiency

Later, balance:

+ Fault Tolerance

+ Utilization Bounds

THREE OPTIMIZERS



FORMAL MODEL: MCF

Variables

Type	Description
Real	Amount of each flow along each link
Boolean	Switch power state
Boolean	Link power state

Optimization Goal

minimize Σ (link + switch power)

Constraints

Type	Constraint	Description
Multi-Commodity Flow	Capacity	traffic <= link rate?
	Flow Conservation	packets in = packets out?
	Demand Satisfaction	bandwidth >= demand?
Our Additions	Flow on active links only	link off ←→ no flow
	Connect switches and links	switch off ←→links off

Does not scale

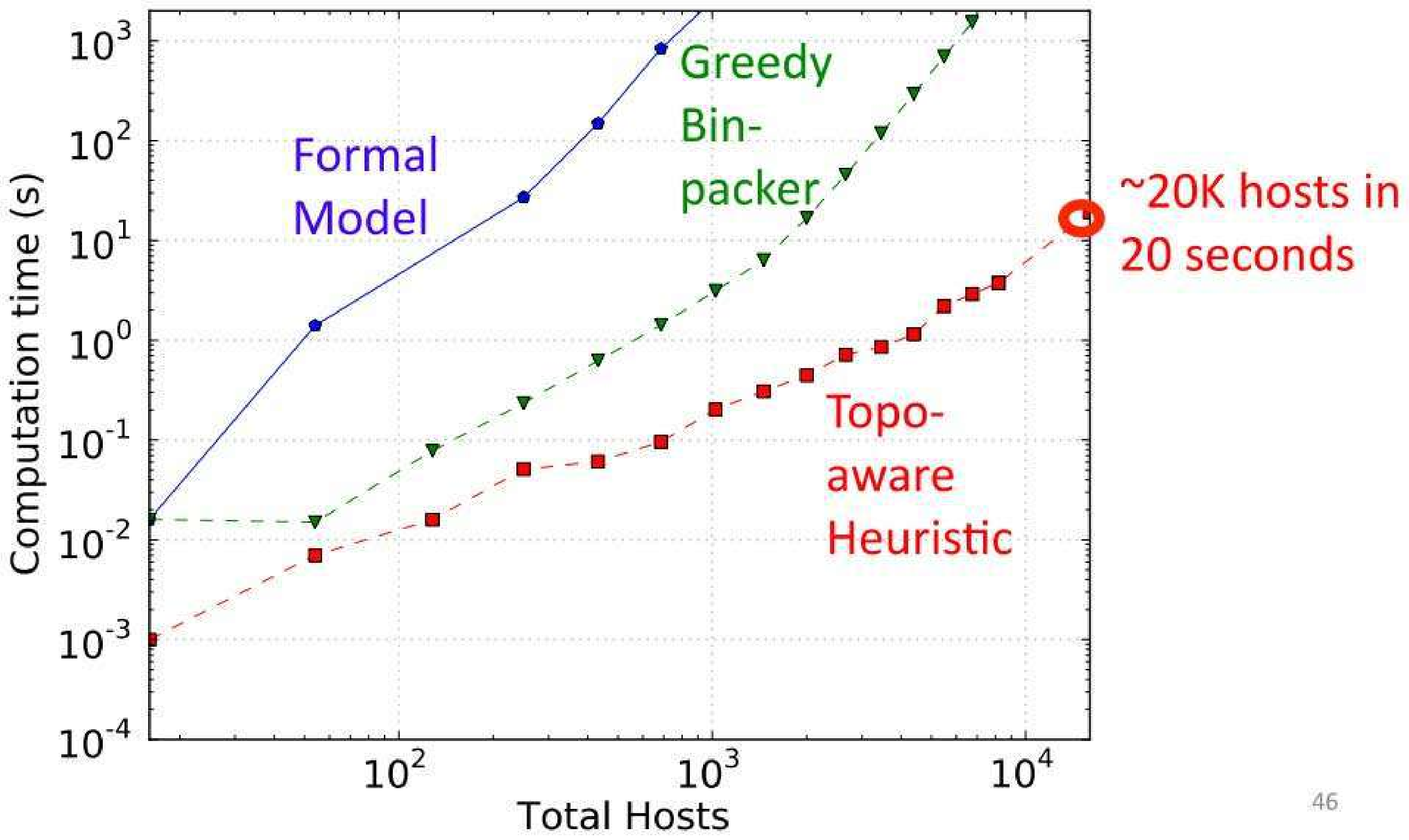
GREEDY BIN PACKING

- For each flow, evaluates all possible flows, and chooses the left-most one with sufficient capacity

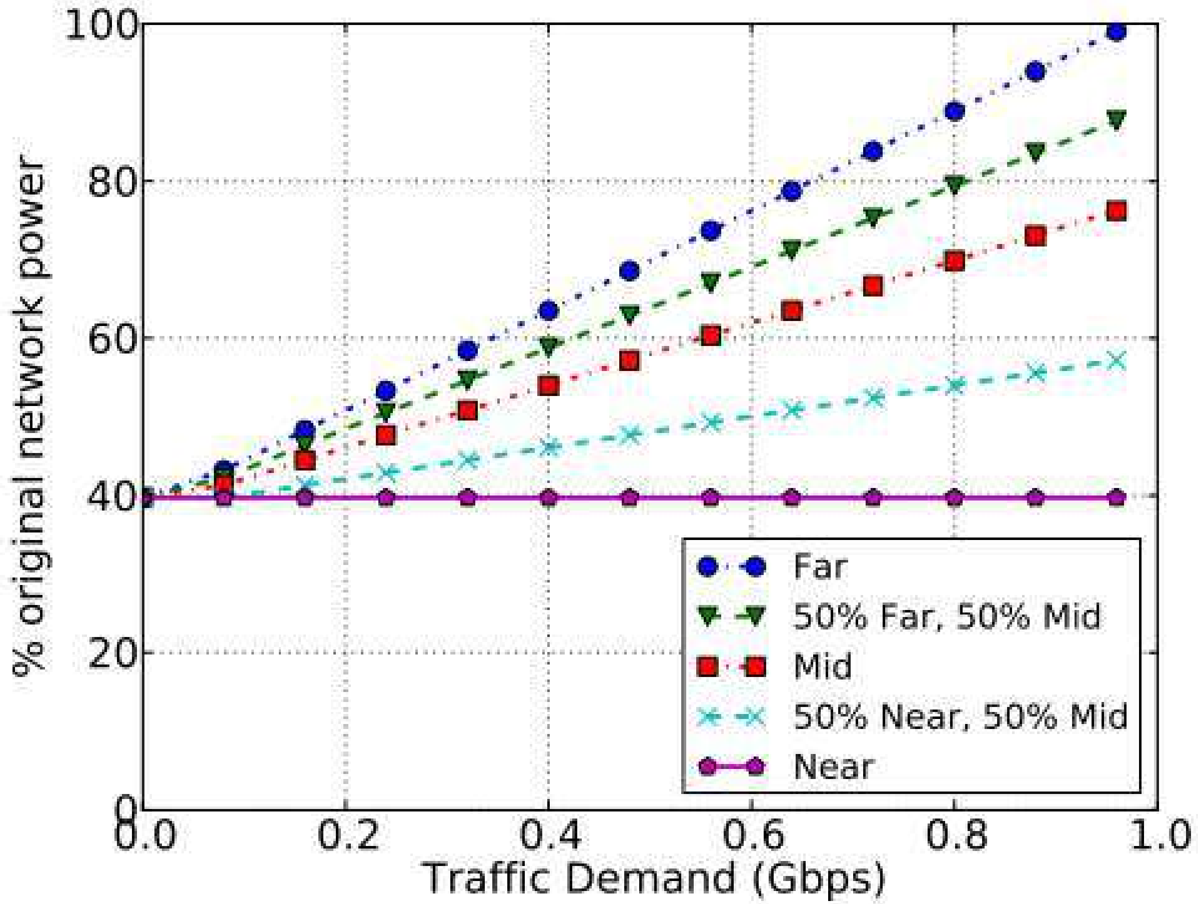
TOPOLOGY-AWARE HEURISTICS

- Active switches == total bandwidth demand / capacity per switch
- Determine which switches are active, and pack flows to the active switches
- Add more switches for fault tolerance and connectivity

SCALABILITY



POTENTIAL POWER SAVINGS



- Near traffic: within the same edge switches
- Far: remote traffic

REALISTIC DATA CENTER TRAFFIC

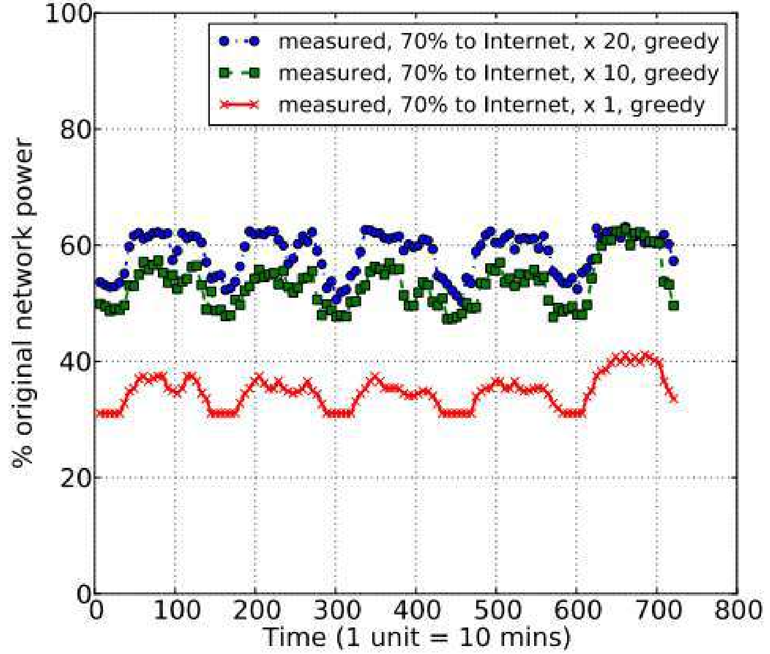


Figure 10: Energy savings for production data center (e-commerce website) traces, over a 5 day period, using a k=12 fat tree. We show savings for different levels of overall traffic, with 70% destined outside the DC.

- Savings range from 25-62%
- A single E-commerce application

SUMMARY

- An interesting idea: energy-proportional networking
- Realized it on realistic datacenter topologies
- Three energy optimizers
  - Heuristics work well

DISCUSSION

- Evaluation does not use traffic from multiple applications

- Not sure what the savings are on EC2, AppEngine, or Azure