SAPIENZA
UNIVERSITÀ DI ROMA

Networking for big data
Data centers

Lecturer: Andrea Baiocchi

email: andrea.baiocchi@uniroma1.it

Department of Information Engineering, Electronics, and Telecommunications
SAPIENZA University of Rome

a.y. 2020/2021

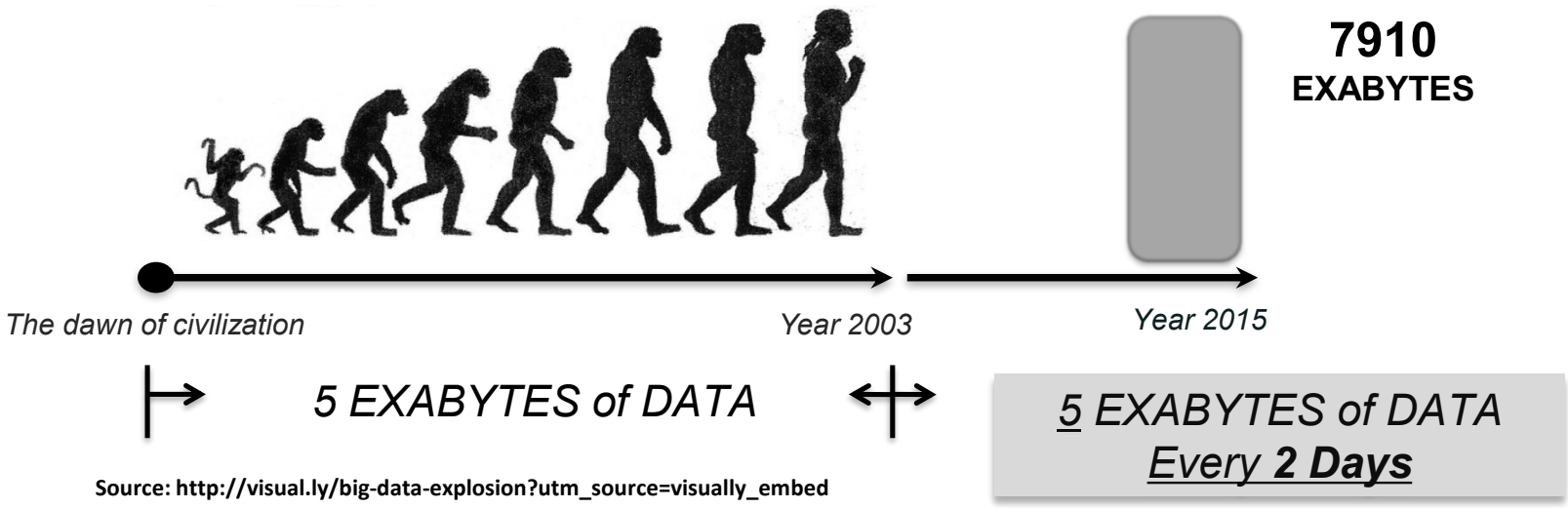
Outline

1. High level view of cloud computing
2. Data center networks
   - Structure and components
   - Data center topology design
3. Resource management and scheduling
   - Single-server models
   - Multiple servers and Load Balancing
4. Congestion control
   - Data Center TCP (DCTCP)
   - Quantized Congestion Notification (QCN) in Ethernet

High level view of cloud computing

Big data growth

- Tremendous growth of data



- Humans generate 2.5 Exabytes of data every day (1 EB $= 10^6$ TB)

Sources of data

- Digitization of audio, video, but also books, journals, archives, documents.
- Massive collection of data from **sensors** → Internet of Things
  - (Smart) environment
  - Vehicles
  - Safety
  - Home automation
  - e-Health
  - Industrial processes
- Profiling, customization, tracking of human activities.
- Virtualization of networked facilities (processing, netwokring).

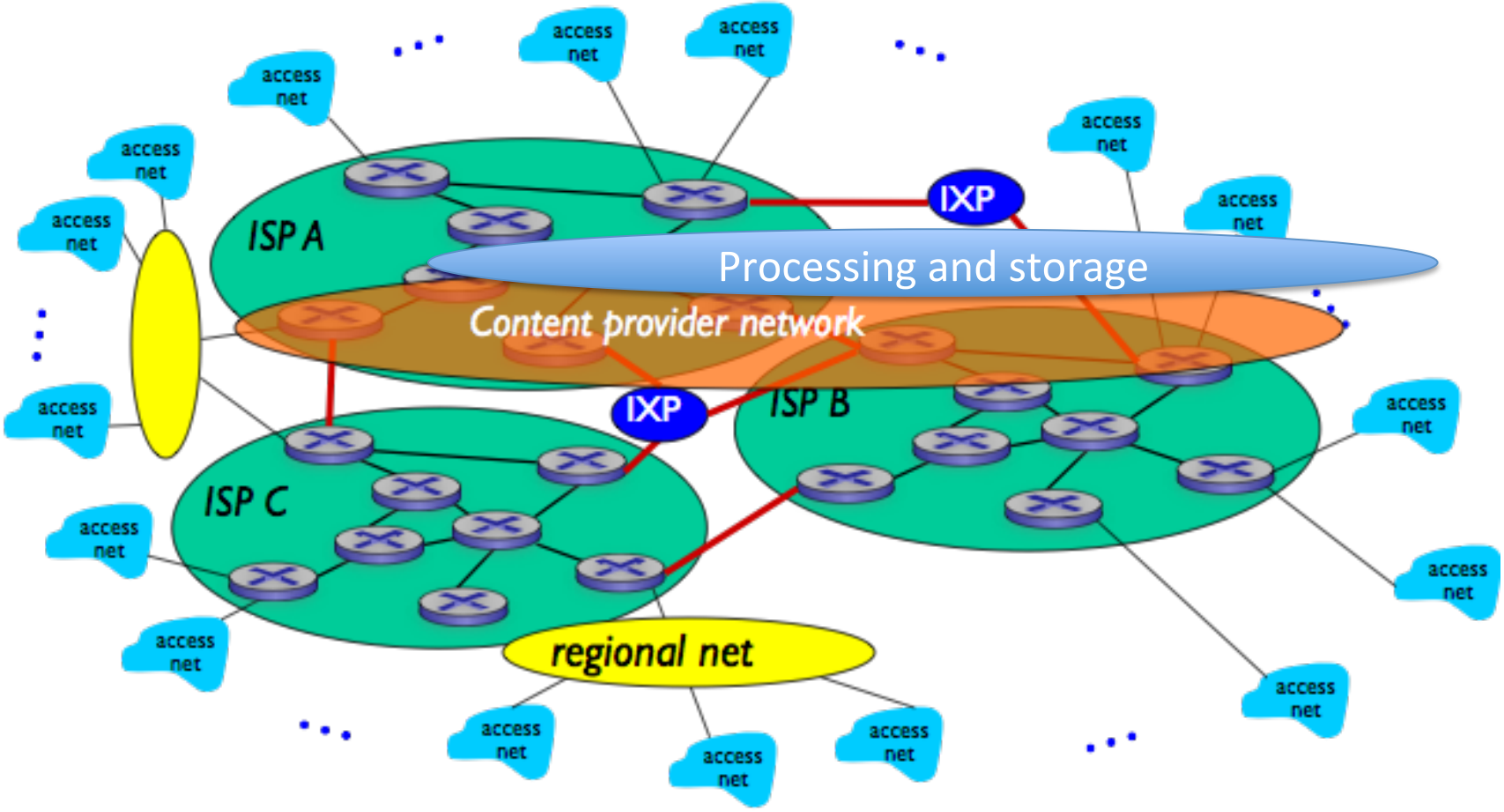
From data to knowledge

$$\text{Data} \neq \text{Information} \neq \text{Knowledge}$$

The Cloud Computing paradigm

- Computing resources used to be shared (also remotely: one of the drivers of the Arpanet project).
- Then, PCs were invented and computing became a commodity on our 'desktops'.
- Cloud computing has introduced a 'paradigm shift' in computing: no longer dedicated physical computing resources owned by the user $\rightarrow$ sharing again!
- The key point is agility: Data Centers turns the servers into a single large configurable pool, thus letting services dynamically expand and contract their footprint as needed.
- Data Centers are the 'engine' of the cloud computing paradigm.

Internet today



Cloud Computing

NIST[1] definition of cloud computing:

"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction."

[1] National Institute of Standards and Technology

Cloud computing characteristics

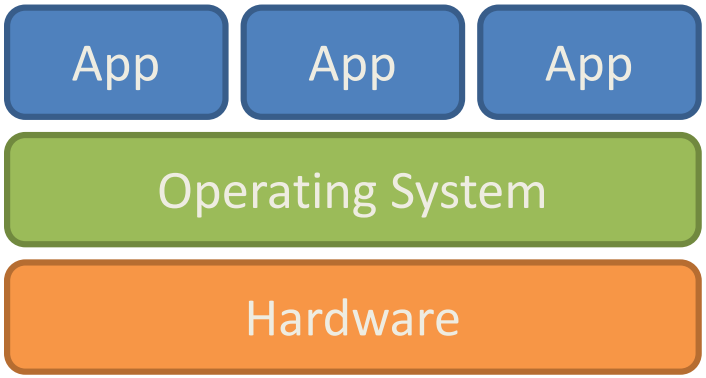
- **On-demand self-service**
  - A 'consumer' (e.g., a service provider) can acquire resources based on service demand.
- **Broad network access**
  - Cloud services can be accessed remotely from heterogeneous client platforms.
- **Resource pooling**
  - Resources are shared by consumers in a multi-tenant fashion.
- **Rapid elasticity**
  - Cloud resources can be provisioned in 'real-time' with minimal human intervention.
- **Measured service**
  - Resources are controlled (and possibly priced) by leveraging a metering capability (e.g., pay-per-use).

The key characteristic

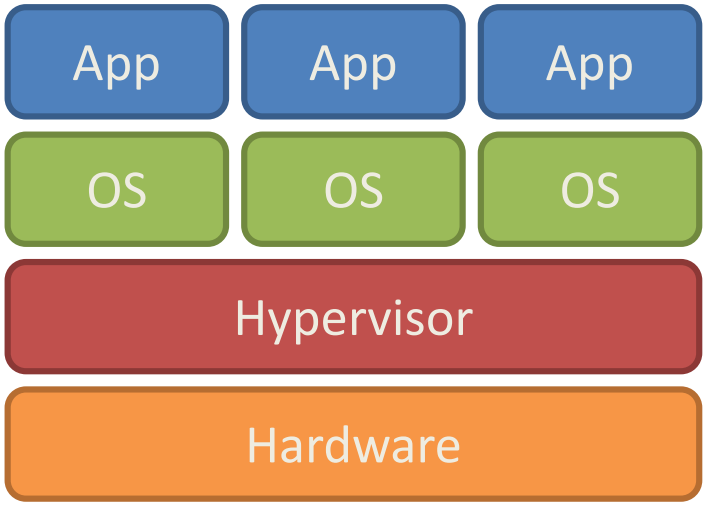
Elasticity or on demand, in two dimensions:

- On demand in time
  - a cloud service allows its users to change the requests for resources at short notice, and possibly only for a short period of time.
- On demand in scale
  - A cloud service allows its users to start at a very small minimum level of resource request, and yet can go to really large scale.
    - Target (`target.com`), the second-largest retail store chain in the USA, runs its web and inventory control in a rented cloud.

Virtualization



Traditional Stack

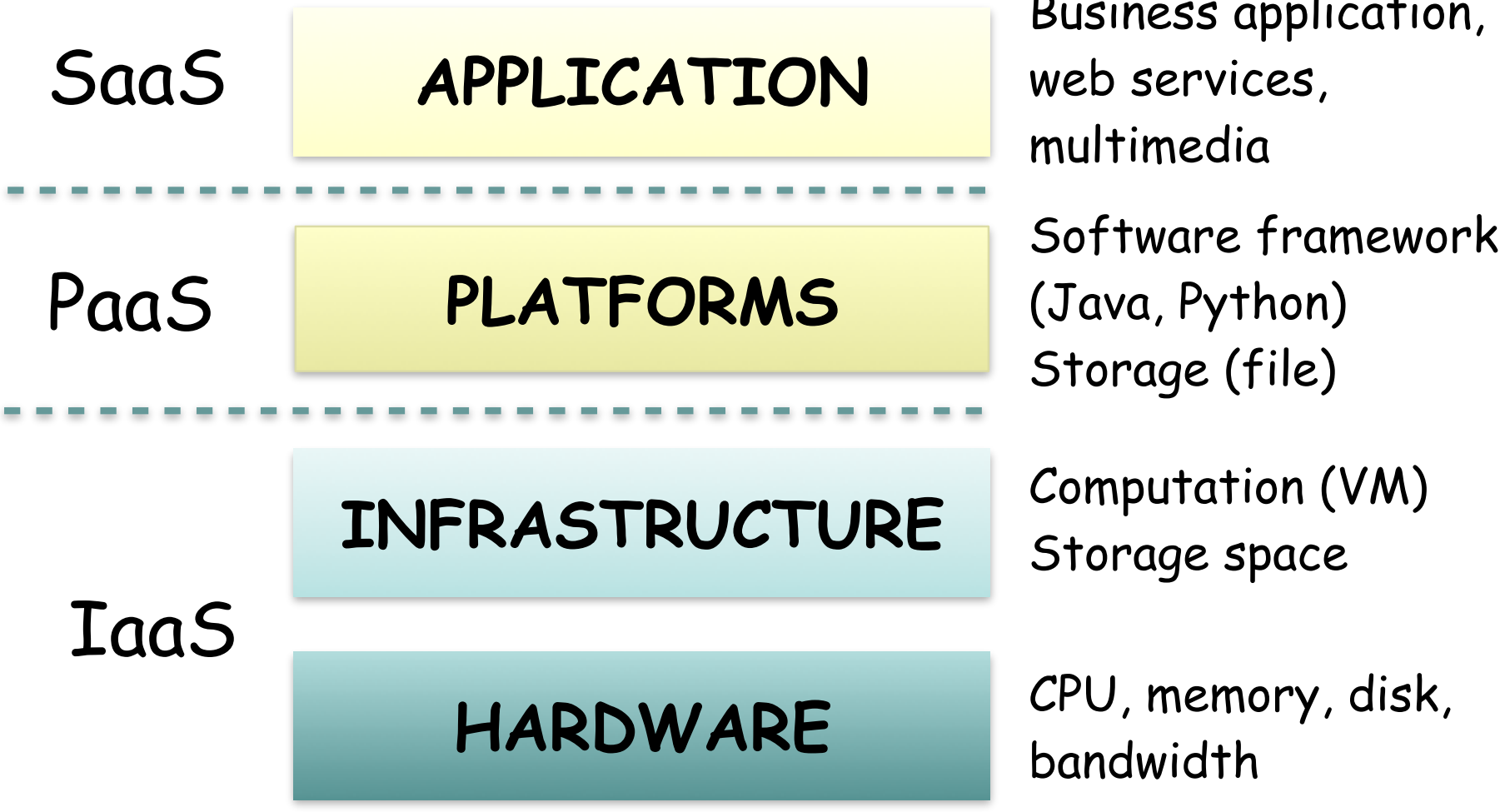


Virtualized Stack

Resources provided as a service

- User can request, configure and access cloud resources using cloud-specific APIs.
- Enables consumers to get computing resources as and when required, without any human intervention.
- Facilitates consumer to leverage “ready to use” services or, enables to choose required services from the service catalog.

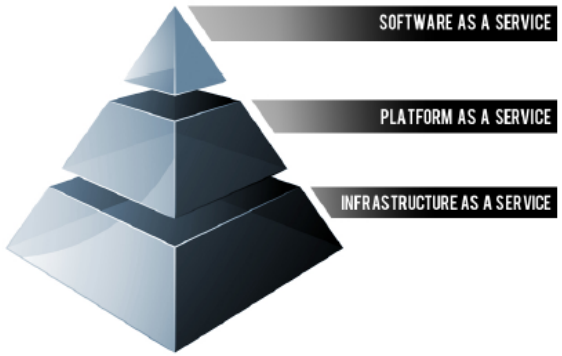
Cloud computing architecture



Resources provided as a service

- Infrastructure-as-a-Service
  - Provides capability to the consumer to hire infrastructure components such as servers, storage, and network.
  - Enables consumers to deploy and run software, including OS and applications.
  - Clients have control of virtual resources (virtualization).

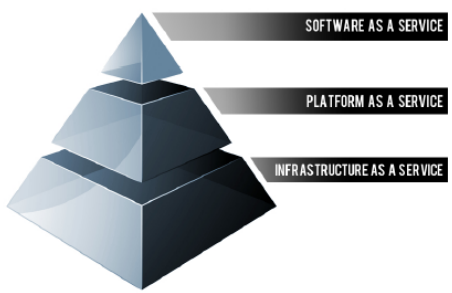
Examples: AWS, VirtualBox, VMWare, OpenStack



Resources provided as a service

- Platform-as-a-Service
  - Capability provided to the consumer to deploy consumer created or acquired applications on the Cloud provider's infrastructure.
  - Consumer has control over deployed applications and application hosting environment configurations.
  - Consumer is billed for platform software components.
  - Clients use proprietary language.

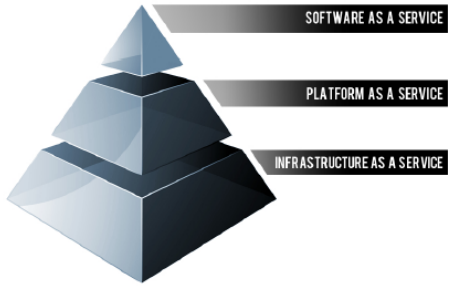
Examples: Google App Engine, Azure, Facebook platform



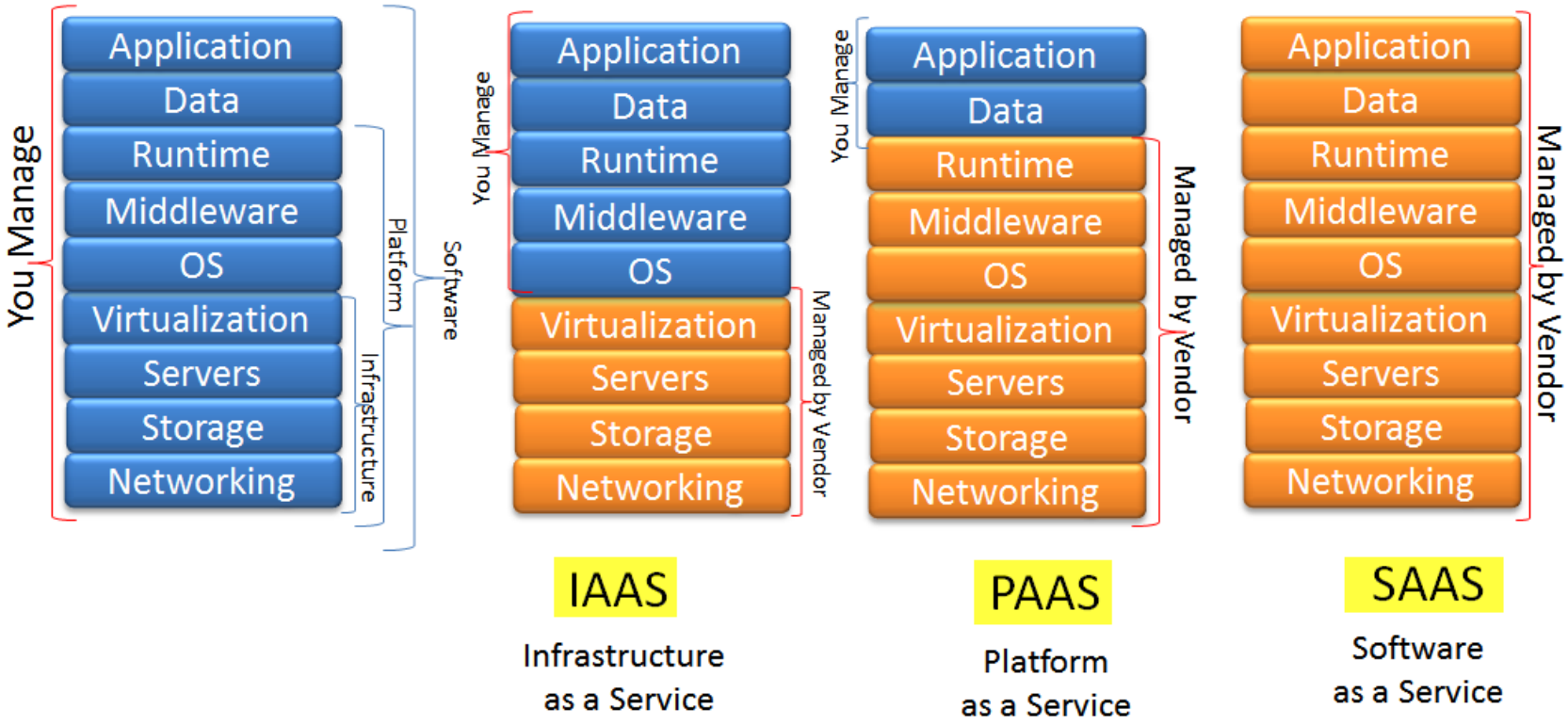
Resources provided as a service

- Software-as-a-Service
  - Capability provided to the consumer to use provider's applications running in a Cloud infrastructure.
  - Complete stack including application is provided as a service.
  - Application is accessible from various client devices, for example, via a thin client interface such as a Web browser.
  - Billing is based on the application usage.

Examples: Dropbox, Google Docs, Google Apps of G.suite, web email, Salesforce.com



Cloud computing service categories



Pay-per-use business model

- Consumers only pay for resources actually used.
- Resource usage is monitored and reported, which provides transparency for charge back to both Cloud service provider and consumer about the utilized service.
- Pricing/billing model is tied up with the required service levels.

Pay-per-use example: Amazon EC2 - general purpose

General Purpose	vCPU	Memory (GB)	Instance Storage (GB)	Linux/UNIX Usage
t2.nano	1	0.5	EBS Only	$0.0075 per Hour
t2.micro	1	1	EBS Only	$0.015 per Hour
t2.small	1	2	EBS Only	$0.03 per Hour
t2.medium	2	4	EBS Only	$0.06 per Hour
t2.large	2	8	EBS Only	$0.12 per Hour
m4.large	2	8	EBS Only	$0.143 per Hour
m4.xlarge	4	16	EBS Only	$0.285 per Hour
m4.2xlarge	8	32	EBS Only	$0.57 per Hour
m4.4xlarge	16	64	EBS Only	$1.14 per Hour
m4.10xlarge	40	160	EBS Only	$2.85 per Hour
m3.medium	1	3.75	1 x 4 SSD	$0.079 per Hour
m3.large	2	7.5	1 x 32 SSD	$0.158 per Hour
m3.xlarge	4	15	2 x 40 SSD	$0.315 per Hour
m3.2xlarge	8	30	2 x 80 SSD	$0.632 per Hour

EBS = Elastic Block Store

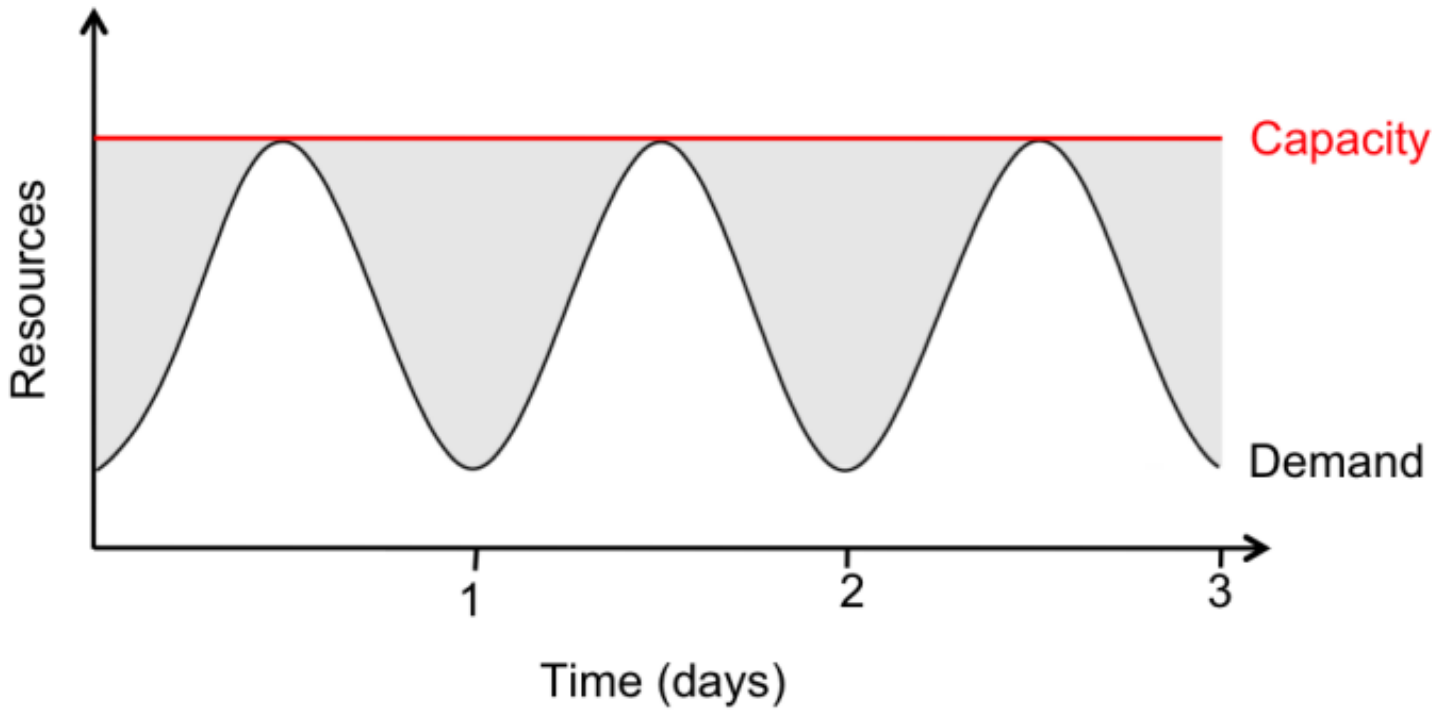
Elasticity

- Consumers can acquire or release resources on demand.
- Ability to scale IT resources rapidly, as required, to fulfill the changing needs without interruption of service.
- Resources can be both scaled up and scaled down dynamically.
- To the consumer, the Cloud appears to be infinite.
- Consumers can start with minimal computing power and can expand their environment to any size.

Elasticity

Pay by use instead of provisioning for peak.

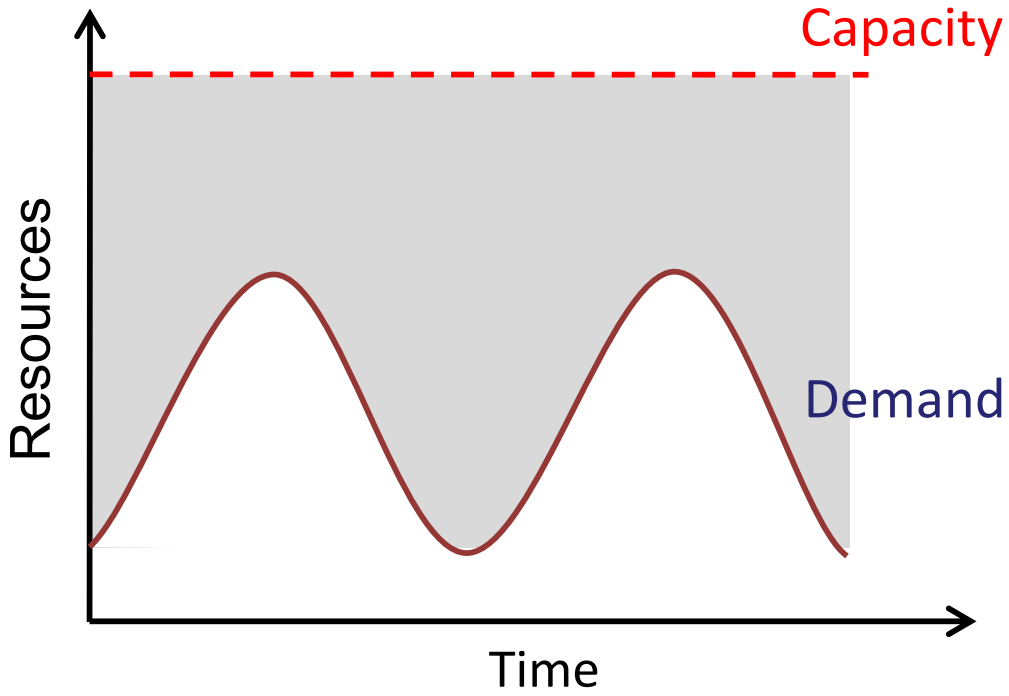
- Recall: DC costs >\$150M and takes 24+ months to design and build.



Elasticity

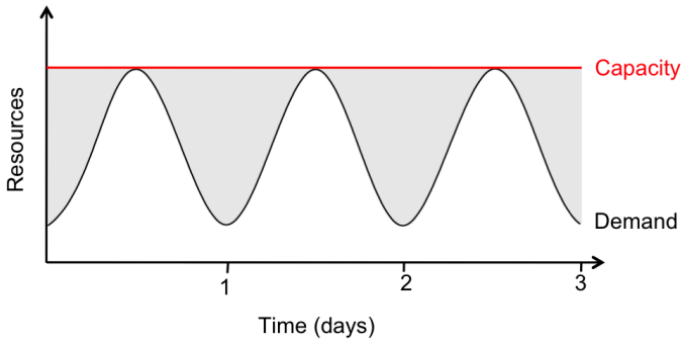
Risk of over-provisioning: underutilization.

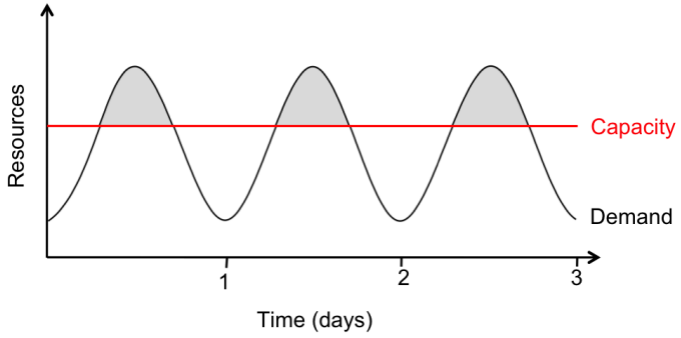
- Huge sunk cost in infrastructure.



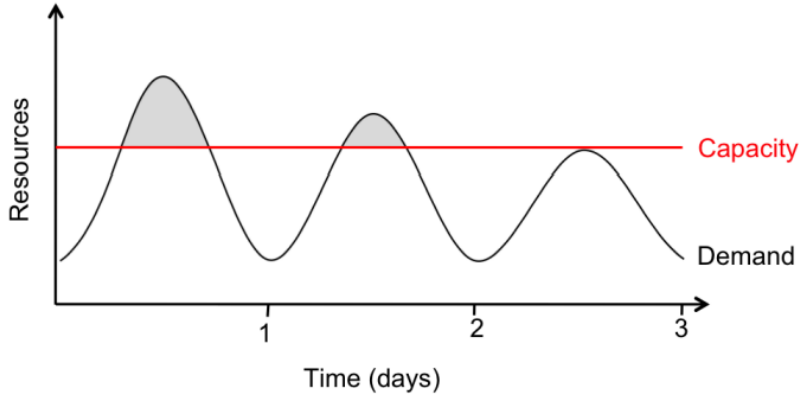
Elasticity

Heavy penalty for under-provisioning.





Lost revenue



Lost users

Enabling technologies

- Data centers
  - Machine virtualization
  - Networking
- Data storage and management
  - Distributed storage $\rightarrow$ issue of consistency and availability in spite of failures
- Distributed processing (e.g., Map/Reduce paradigm)
- Resource management and scheduling
- Energy management
- Security and privacy

Data center networks

What is a Data Center made of?

A Data Center (DC) is NOT a super-computer.

Server exchange files and messages → *DC network*

Processing power and storage capacity are obtained by piling up Commercial Off-The-Shelf (COTS) components massively.

DC network 'Lego' pieces:

- Blade servers in racks.
  - Typically up to $\sim$ 40 servers in a single rack.
- Ethernet switches to interconnect them.
  - typical: 10 / 25 / 40 / 100 / 200 / 400 Gbps ports.
  - High-end switches offer up to thousands ports, but costly.
  - COTS switches have typically 16, 24, 48 and 64 ports.
- Cabling

An example switch for enterprise networking

- Interfaces: 48 x 10Base-T/100Base-TX/1000Base-T - RJ-45
- Width: 43.9 cm
- Depth: 41 cm
- Height: 4.4 cm
- Weight: 7.5 kg

Servers

42u Server Rack

- Servers are computers that provide "services" to "clients".
- They are typically designed for reliability and to service a large number of requests.
- Organizations typically require many physical servers to provide various services (Web, Email, Database, etc.).
- Servers are typically placed in racks.
  - Designed to fit into rack units (1u, 2u, 4u).
  - A single rack can hold up to 42 1u servers.

Blades

- A blade server is a stripped down computer with a modular design.
- A blade enclosure holds multiple blade servers and provides power, interfaces and cooling for the individual blade servers.

Data center components

Besides servers (storage, processing) a Data Center consists of:

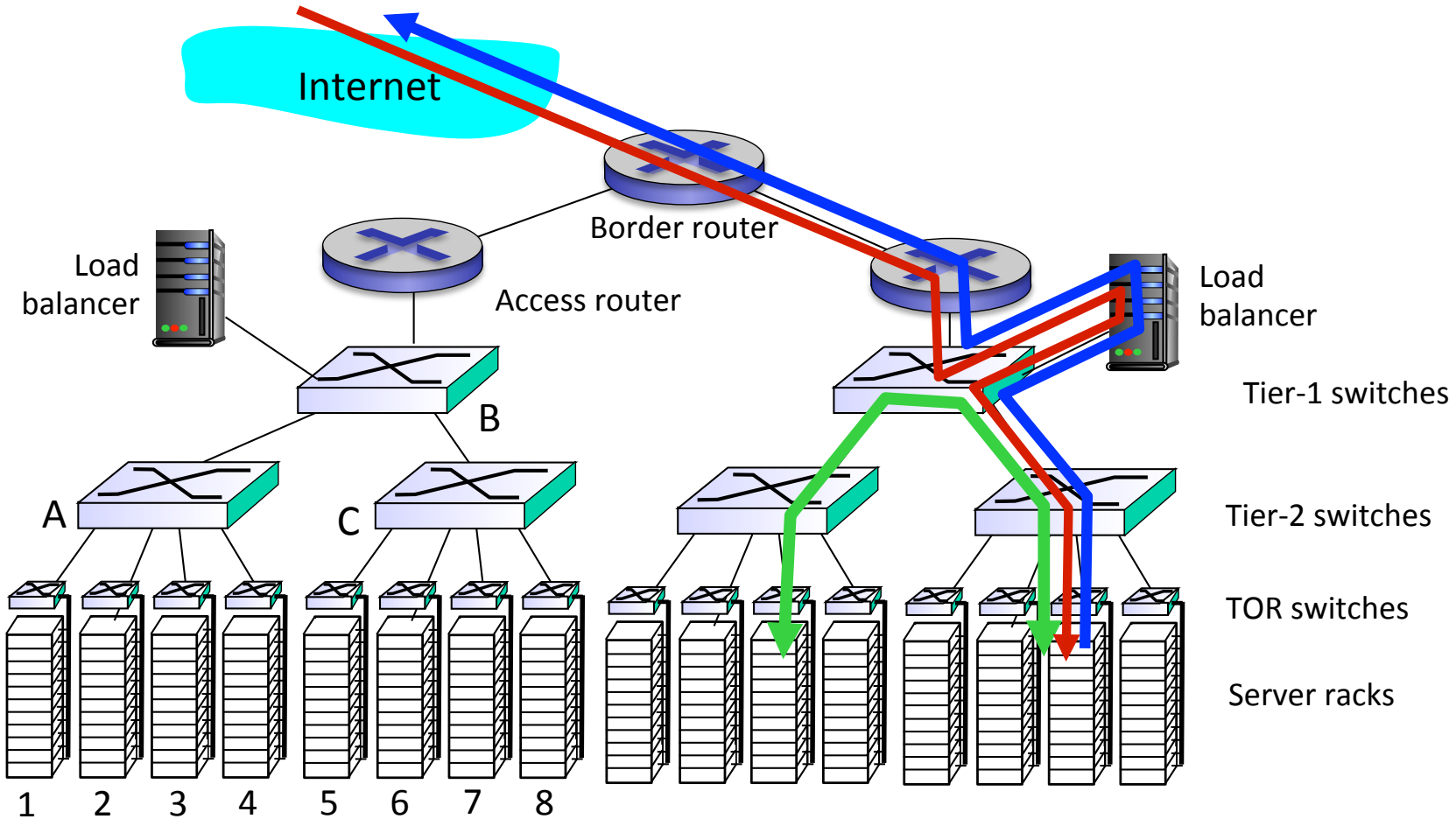
- Air conditioning, to keep all components in the manufacturer's recommended temperature range.
- Powering, including redundant power (UPS/generators, multiple power feeds).
- Fire protection.
- Physical security.
- Monitoring systems.
- Connectivity elements
  - routers/load balancers from/to the Internet;
  - switches among servers.

Communications in Data Centers

- Communications in data centers are most often based on networks running the IP protocol suite.
- Data centers contain a set of routers and switches that transport traffic between the servers and to the outside world.
- Traffic in today's data centers:
  - 80% of the packets stay inside the data center.
  - Trend is towards even more internal communications.
- Typically, data centers run two kinds of applications:
  - Outward facing (serving end-users).
  - Internal computation (data mining and indexing).

Key requirement: live migration of VM from one physical machine to another physical machine with minimal network reconfiguration.

Interconnection to the Internet



Ethernet vs IP

In the following we refer to the inter-connected devices in a Data Center network as "nodes".

- The nodes can be configured to operate in Ethernet-switched mode or IP-routed mode.
- There are pros and cons of these two modes of operation.
- Key concern: **scalability**.
  - Data center networks increasingly connect 100k - 1 million servers.

Addressing paradigms and VM migration

- IP addressing is **hierarchical**.
  - Hierarchical means the address assigned to an interface depends on its topological location.
- Advantage of hierarchical addressing: **aggregation** → smaller forwarding tables.
- Disadvantage of hierarchical addressing: VM migration requires one of three approaches.
  1. IP address needs to change to reflect new topological position, which means loss of live TCP connections;
  2. forwarding tables in all intermediate routers are updated with the address of the migrating machine (unscalable);
  3. a solution such as mobile IP is required.
- In an Ethernet-switched network, the migrating machine can retain its IP address; just change IP-MAC address resolution.

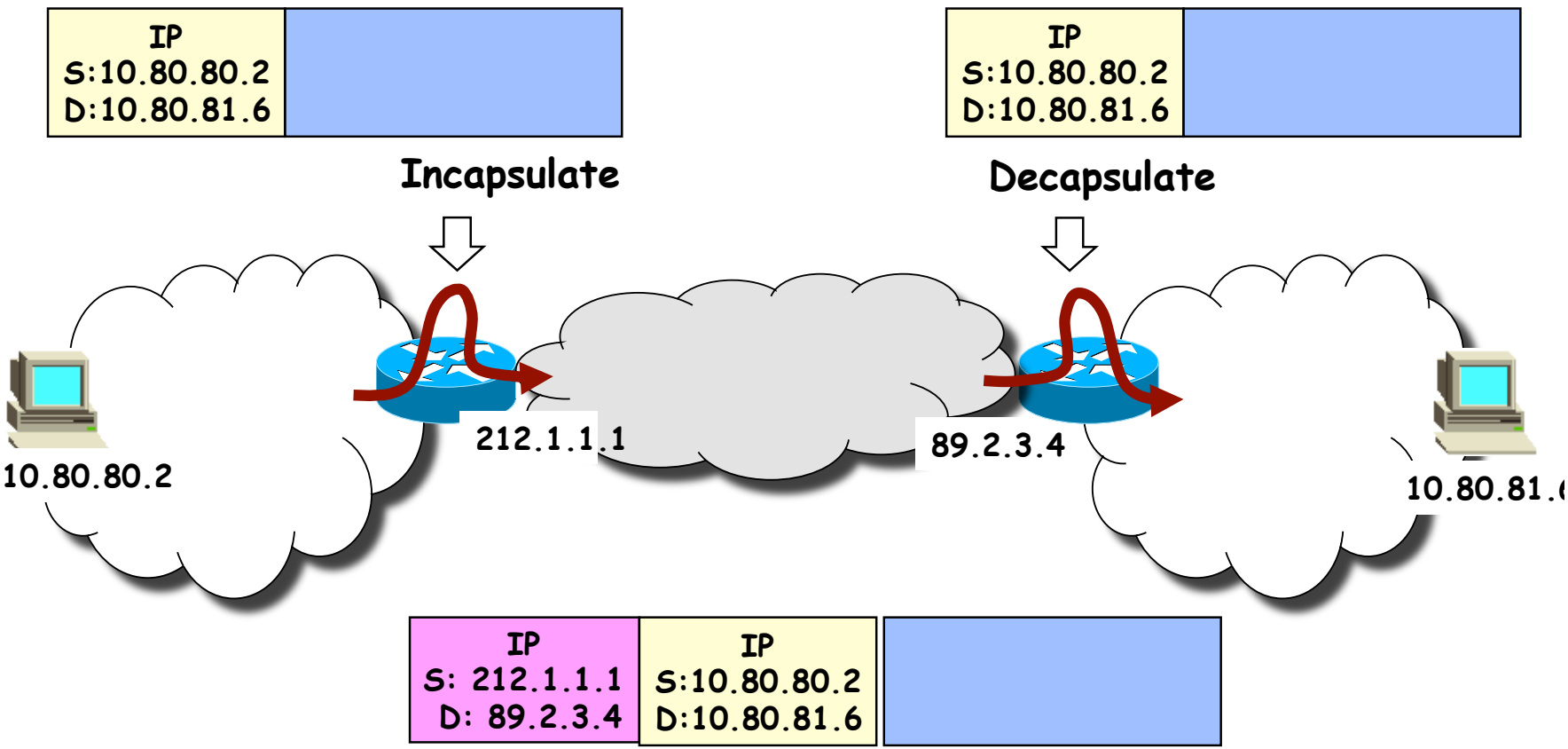
Flat addressing and routing

- The Ethernet 6-byte MAC addressing is **flat**.
- Ethernet-switched networks require no address configuration (flat addressing).
  - Server interfaces come ready for plug-n-play deployment with manufacturer-configured addresses.
  - Address configuration is required with hierarchical addressing schemes.
- No routing protocols for flat addressing: **flooding** and **address learning** are the methods used to slowly create forwarding table entries.
- Spanning tree protocol avoids loops by disabling certain links, but this is wasteful of network bandwidth resources.
- To improve efficiency, Virtual LANs with multiple spanning trees have been standardized in IEEE 802.1Q.

Ad hoc approaches

- Neither a completely flat addressing network nor a completely hierarchical addressing network appears suitable for a highly **scalable** and **agile** network that supports host mobility (VM migration).
- Solutions are:
  - IETF's TRansparent Interconnection of Lots of Links (TRILL).
  - Provider Backbone Bridging in IEEE's Shortest Path Bridging solution.
- Both use tunneling, with the original packets encapsulated in new packets whose outer headers carry the addresses used for packet forwarding through the core (IP-in-IP or MAC-in-MAC).

Encapsulation concept



Latency in Data Centers

- Propagation delay in the data center is essentially 'zero'.
  - The signal is carried by an e.m. field: it goes ~ 200 m in a μs within an optical fiber and ~ 250 m in a μs in copper wires.
- End to end latency comes from:
  - Switching latency
    - 10G to 10G: ~ 2.5 μs (store&forward);
    - 2 μs (cut-through).
    - Typical intra-rack base RTT: ≈ 5 μs.
    - Typical intra-pod base RTT: ≈ 10 μs.
  - Queueing latency
    - Depends on size of queues and network load; we will see more when dealing with congestion control algorithms and protocols.
- Typical times across a quiet data center: $10 \div 20$ μs.

A look at a Data Center

Modular Data Centers

(a)

(b)

(c)

(d)

Data center size

How big is a data center?

(e) Data center at 350 East Cermak, Chicago, USA: 1.1 million sq ft

(f) Olympic stadium of Roma, Italy: 360,000 sq ft

Wembley stadium: 857,000 sq ft
The Vatican City: 4,74 million sq ft
The Pentagon: 6,5 million sq ft

Data center size

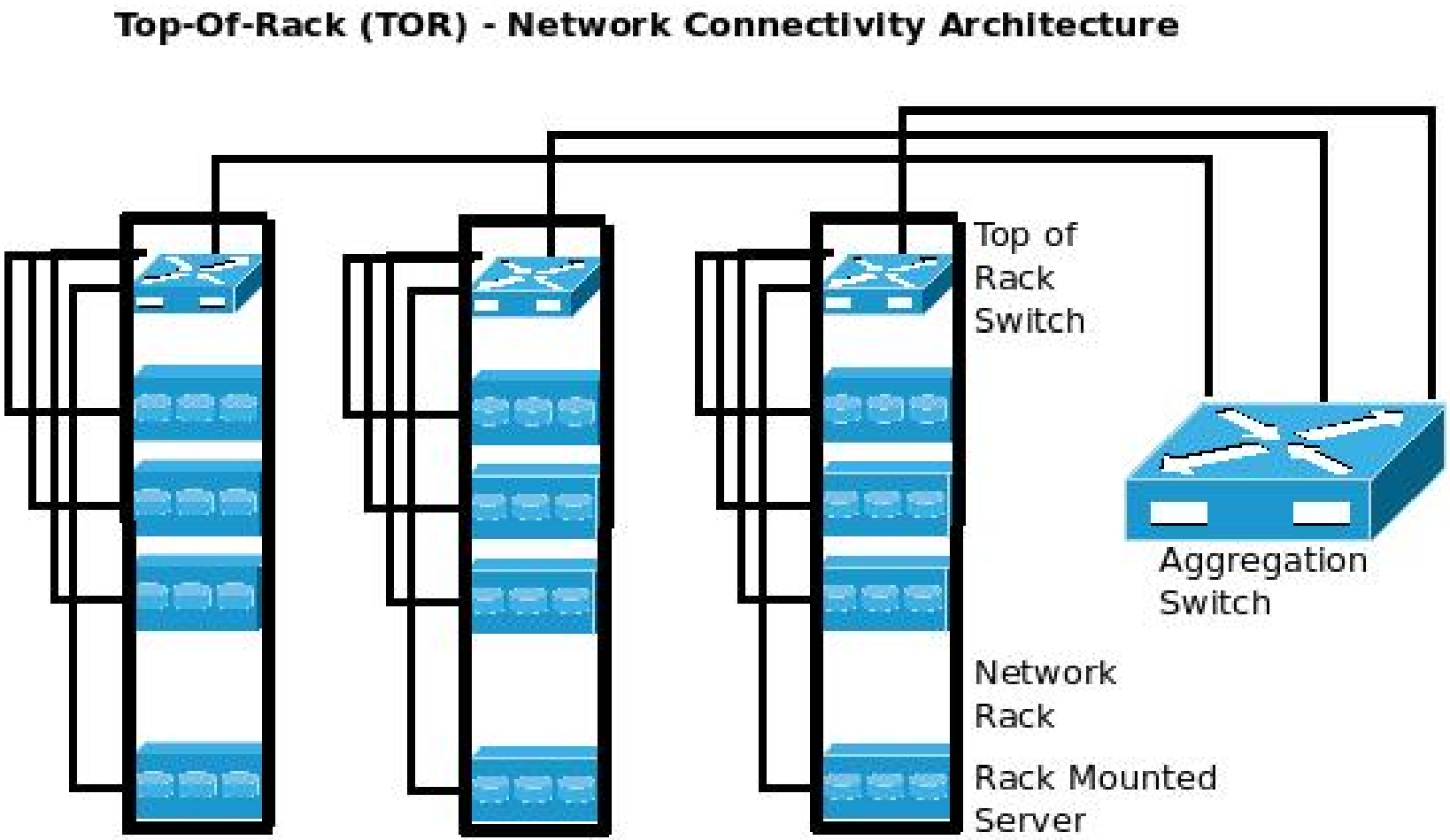
Big Data Centers

- 100.000 or more servers.
- Energy consumption in the order of 10s MWs.
- Applications:
  - Amazon EC2, Windows Azure, Google AppEngine

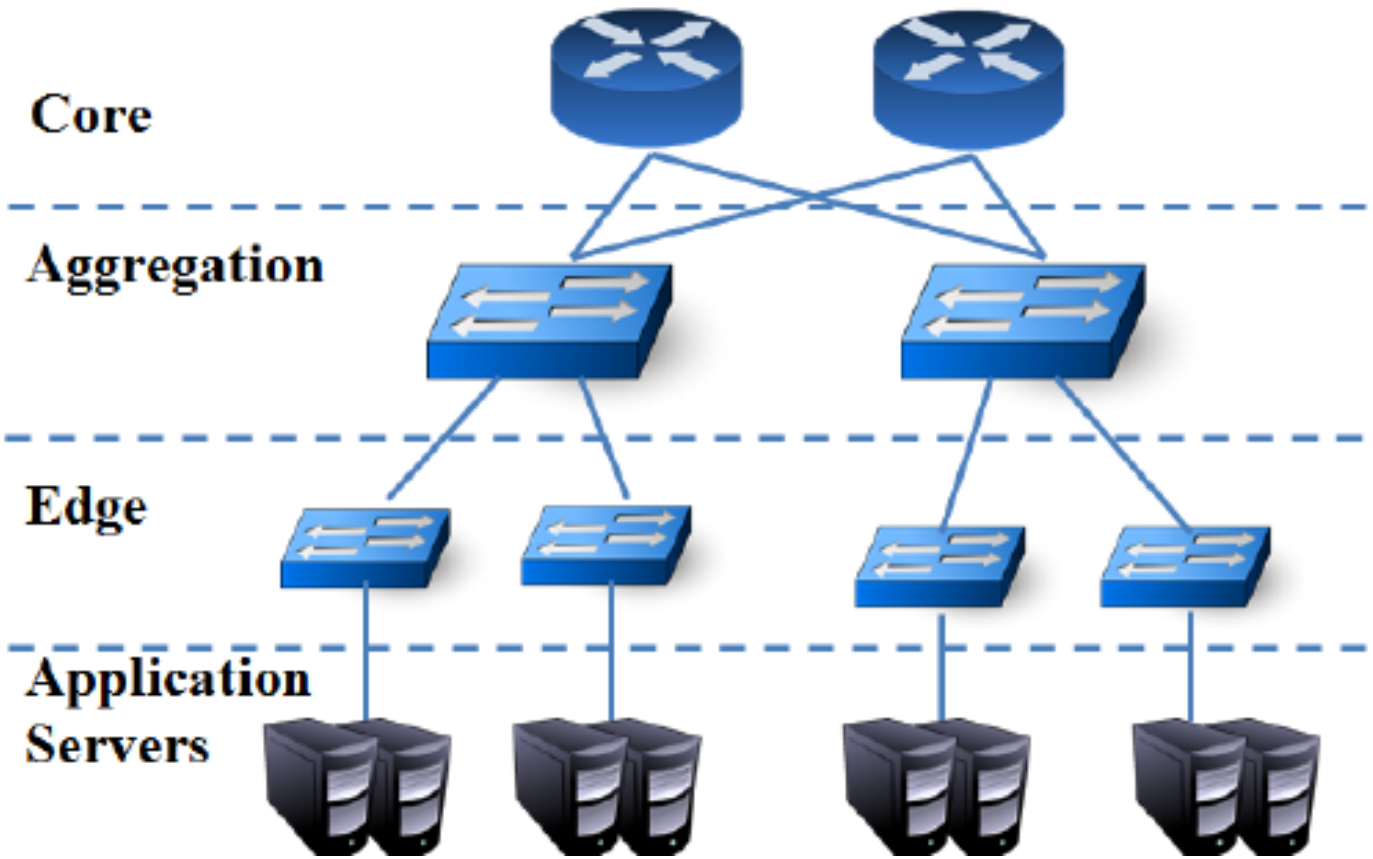
Modular DCs: ICE Cube DC

- Up to 46,080 cores.
- 30 petabytes of storage.
- Low cooling and energy costs.

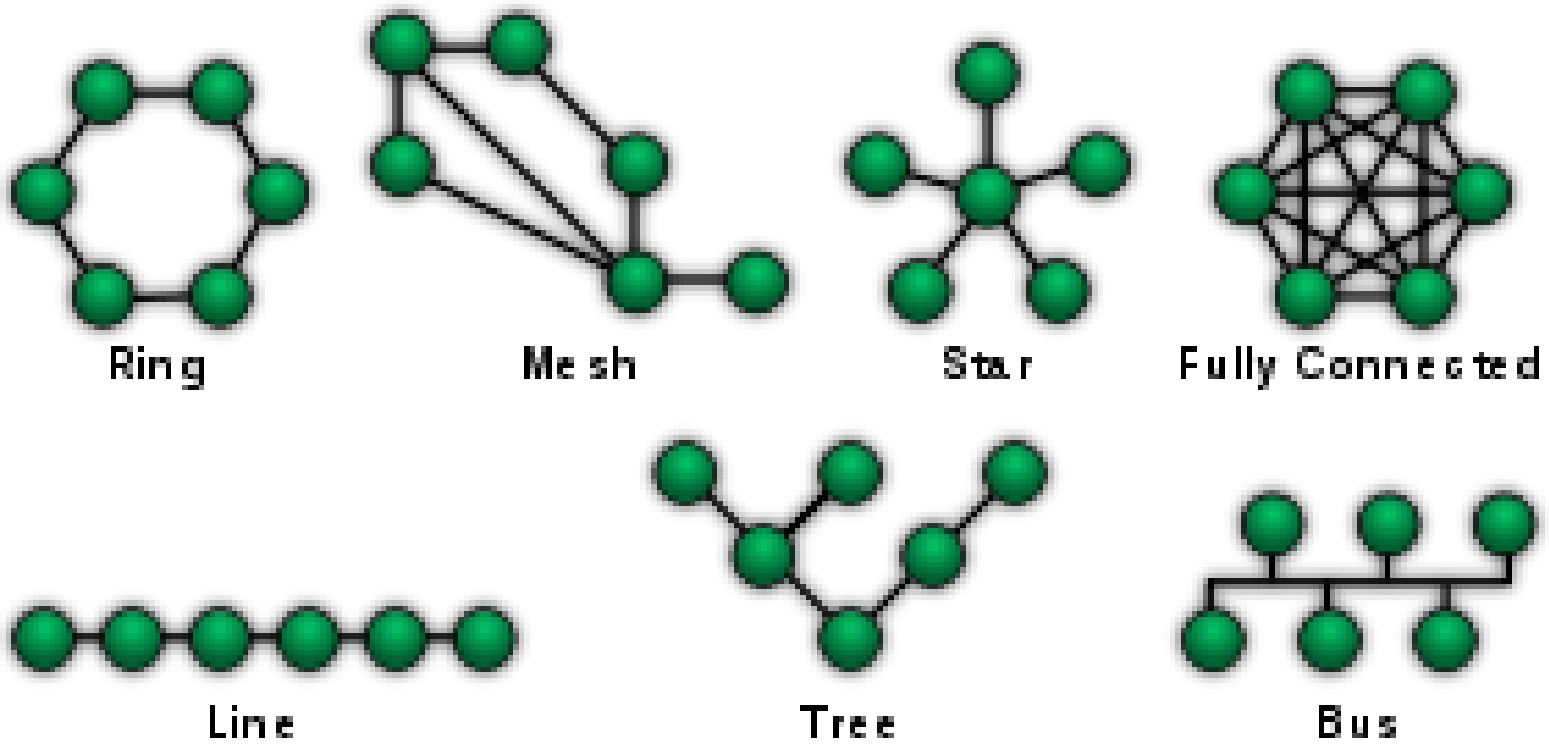
Top of Rack (ToR) switch



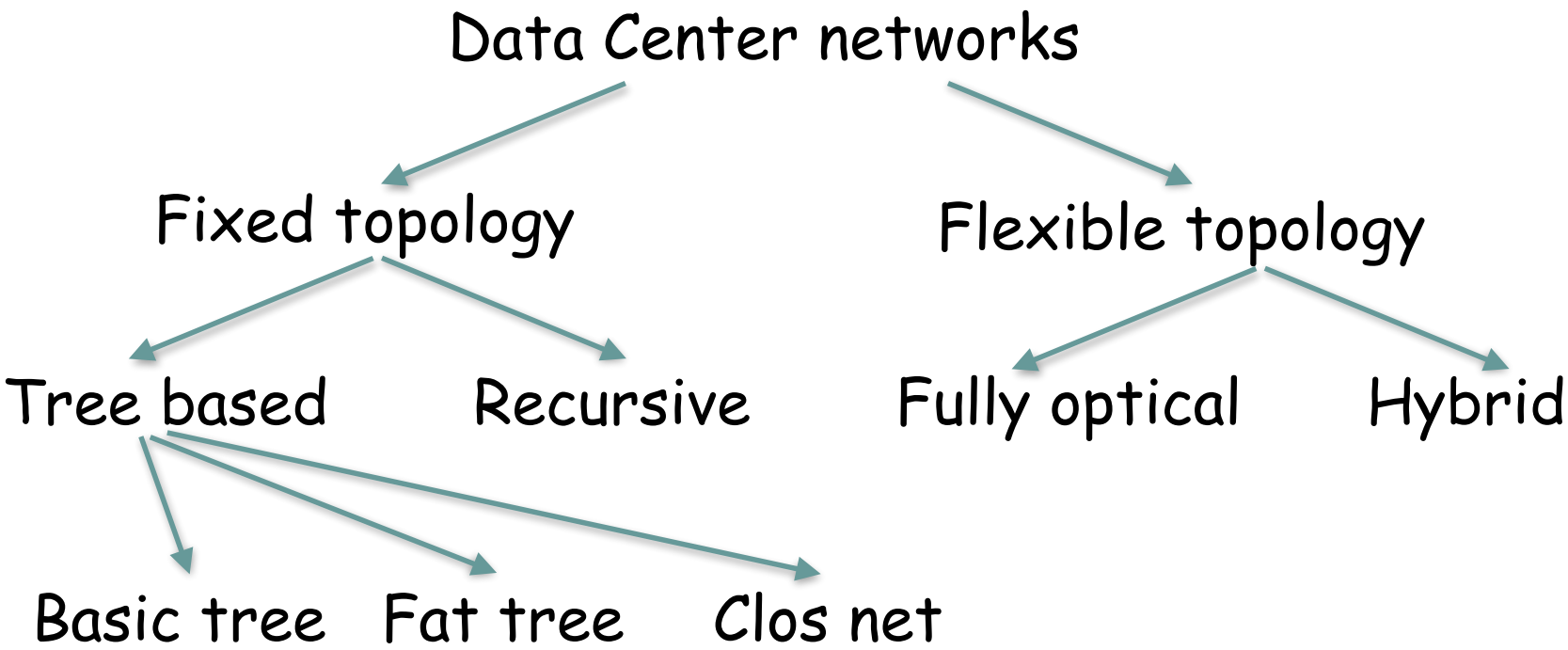
Canonical three tier data center topology



Topologies



Taxonomy of data center topologies



Another classification

- A different classification of DC topologies is based on the networked elements, i.e., the type of devices that are equipped with communication ports and that make up the inter-connection network:
  - switch, i.e., only switches are used to build the inter-connection network of the DC.
  - server+switch, i.e., both switches and servers are used to build the inter-connection network.
  - server only, i.e., only servers are used (no switches in the DC).
- In the two last cases servers must be equipped with more than a single port and should be able to forward frames.
- Network processing adds to application oriented task processing.

Key performance indicators

- The DC network topologies can be compared under various perspectives.
  - connectivity, i.e., number of alternative paths between any pair of servers.
  - bandwidth, i.e., the capacity available for transferring data between any two servers.
  - reliability, i.e., the robustness of the topology to maintain connectivity in spite of link/node failures.
  - expandability, i.e., the easiness of incremental growth of the DC network.
  - deployment, i.e., the complexity of laying out the designed topology in a physical plant (cabling, racks, switches and server requirements, length of links).

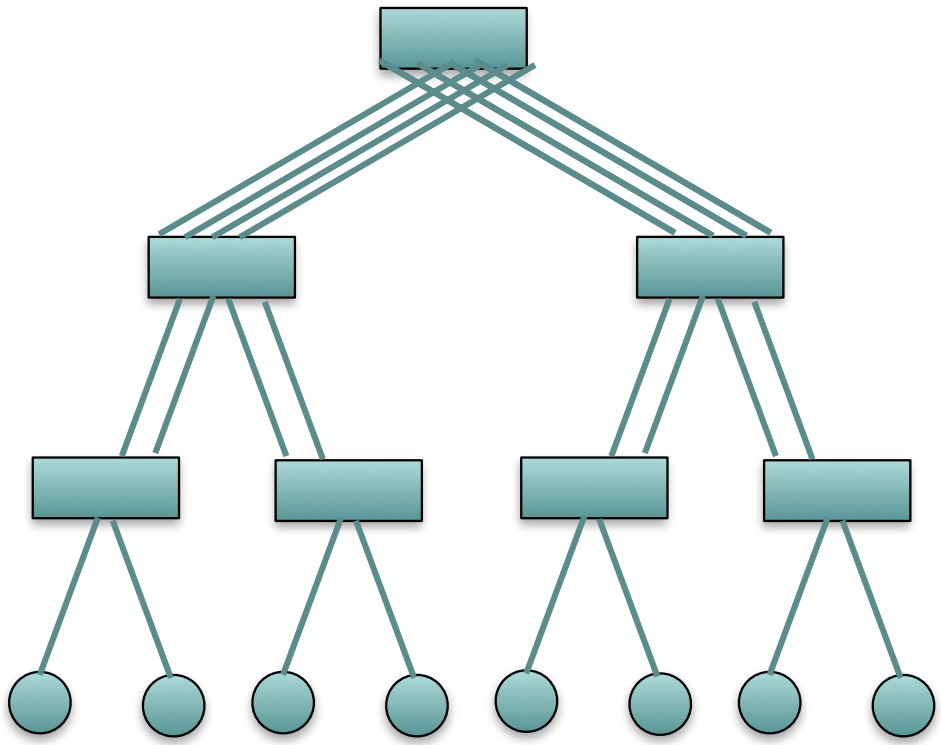
Data Center design does not reduce to topology . . .

Besides topology, other key issues must be considered:

- addressing
- routing
- scheduling and resource allocation
- flexibility and elasticity
- performance metering
- congestion control
- energy consumption
- survivability
- security

Tree topology

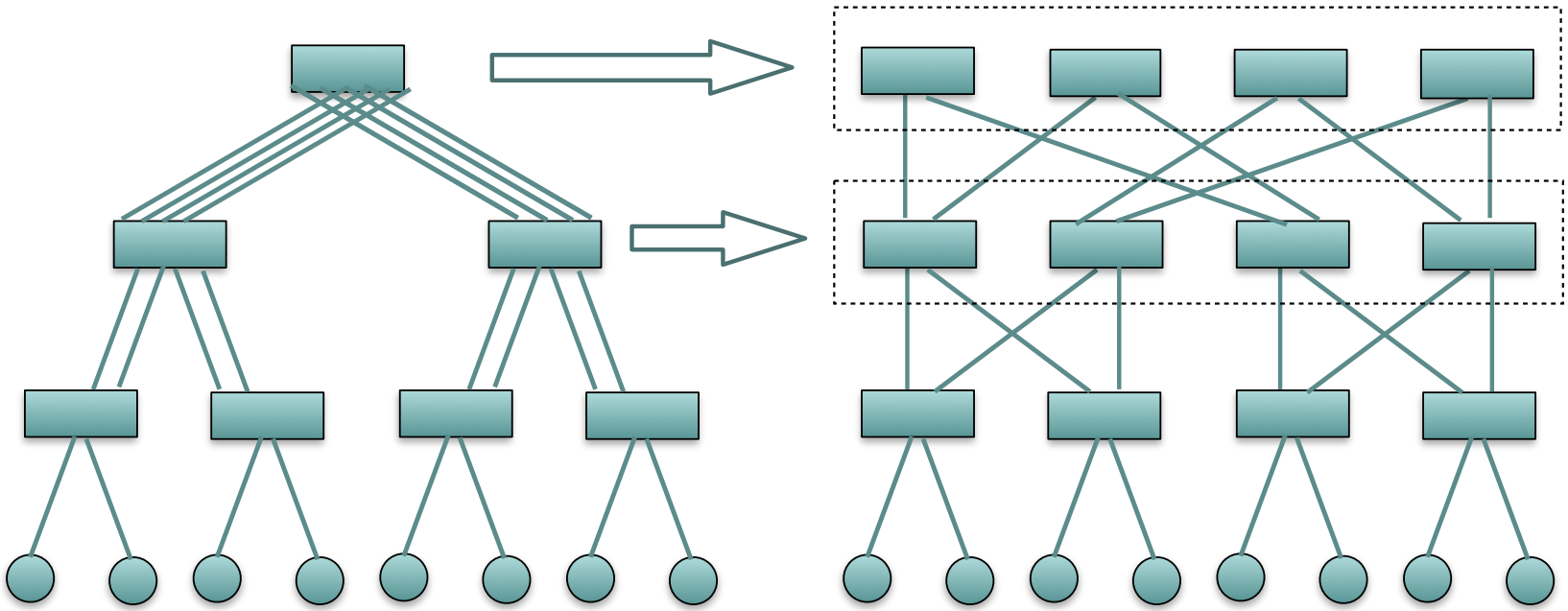
A naïve approach: tree topology



More powerful switches in upper layers: does not scale!

Fat tree topology

A smarter approach: *fat* tree topology - same hw at each layer

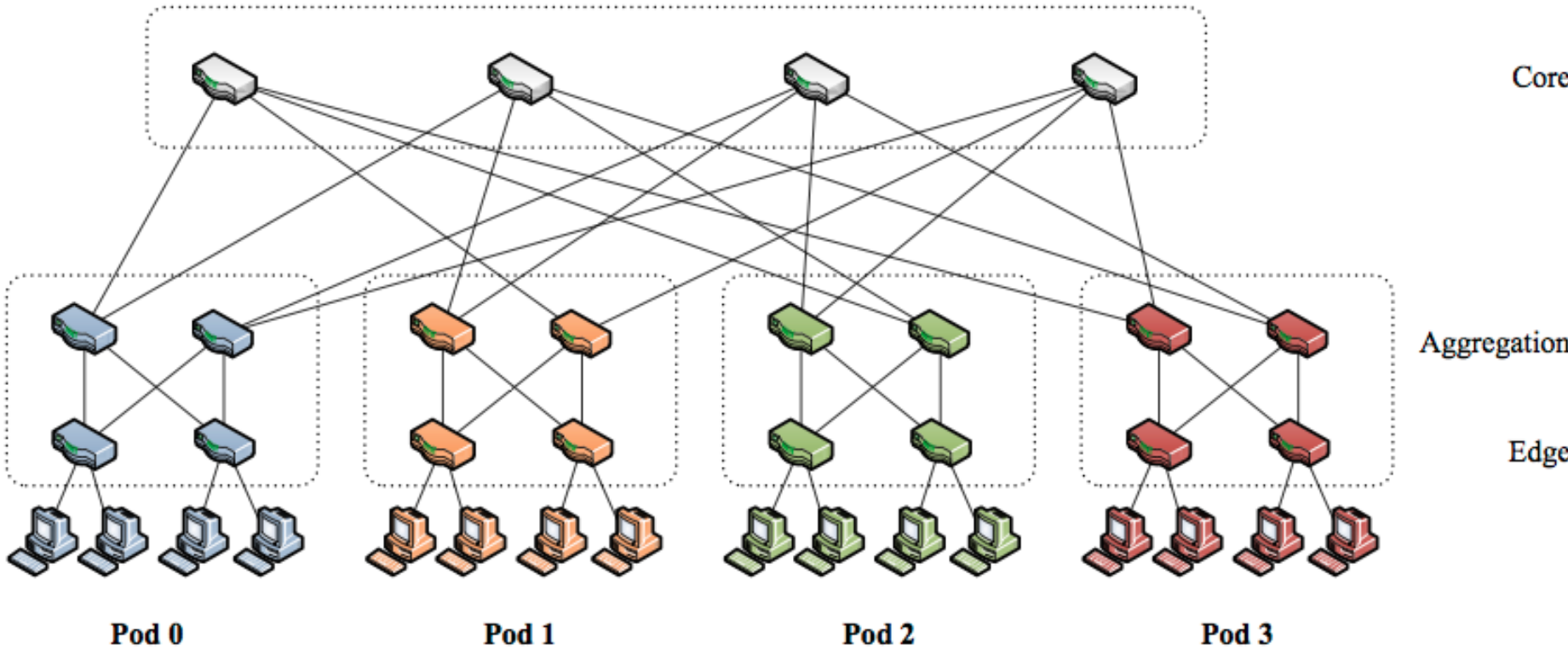


How do we find this topology?

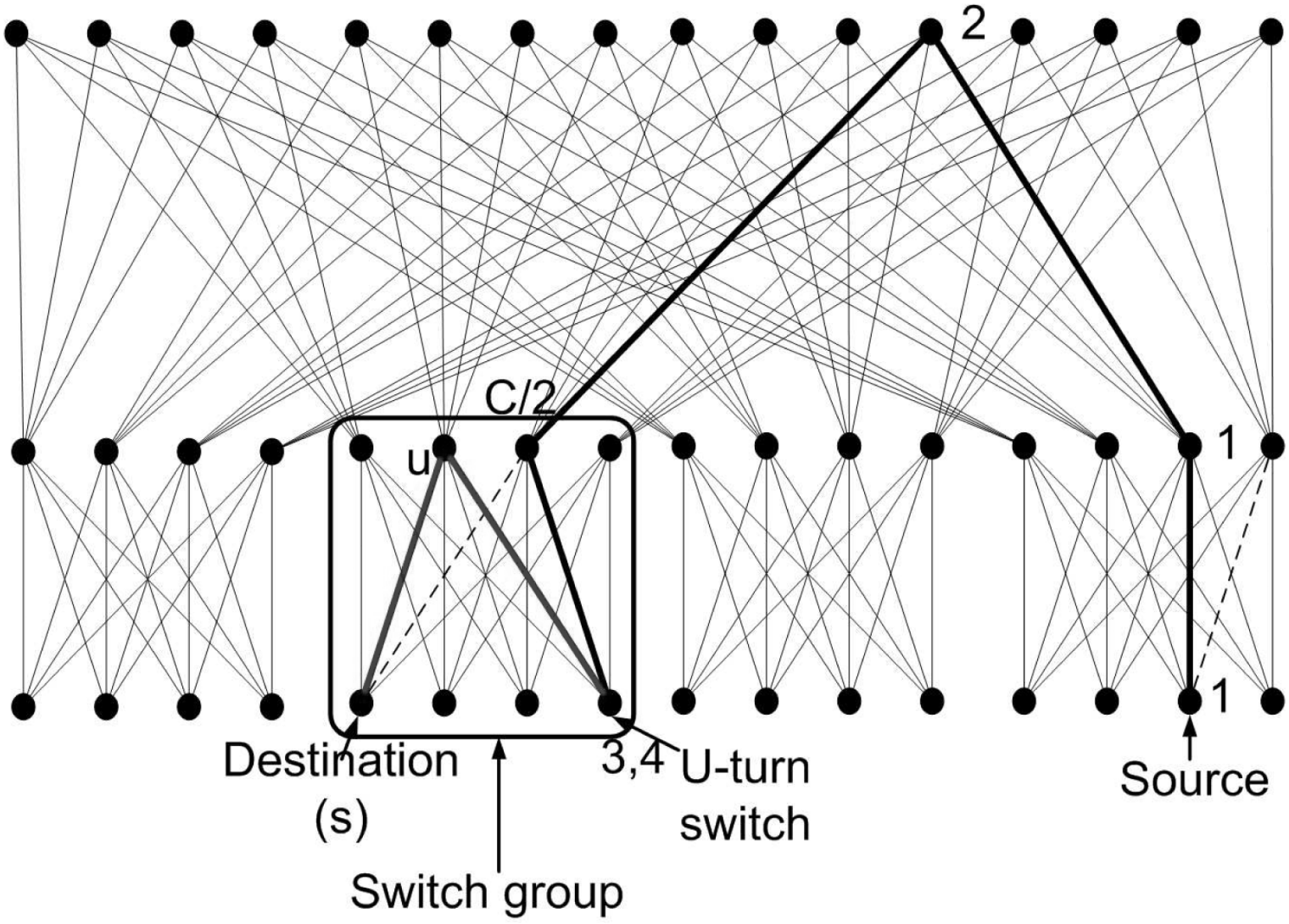
Fat tree construction

- A fat tree is a network based on a complete binary tree made up of n-port switches.
  - Each n-port switch in the edge tier is connected to $n/2$ servers.
  - The remaining $n/2$ ports are connected to $n/2$ switches in the aggregation level.
  - The $n/2$ aggregation-level switches, the $n/2$ edge-level switches and the servers connected to them form a basic cell of a fat tree, called a pod.
  - Each pod has $(n/2)^2$ links to the core layer ($n/2$ for each of the $n/2$ edge switches of the pod).
- In the core level, there are $(n/2)^2$ n-port switches, each one connecting to each of the n pods.
- The maximum number of servers is $n^3/4$.
- The number of required switches is $5n^2/4$.

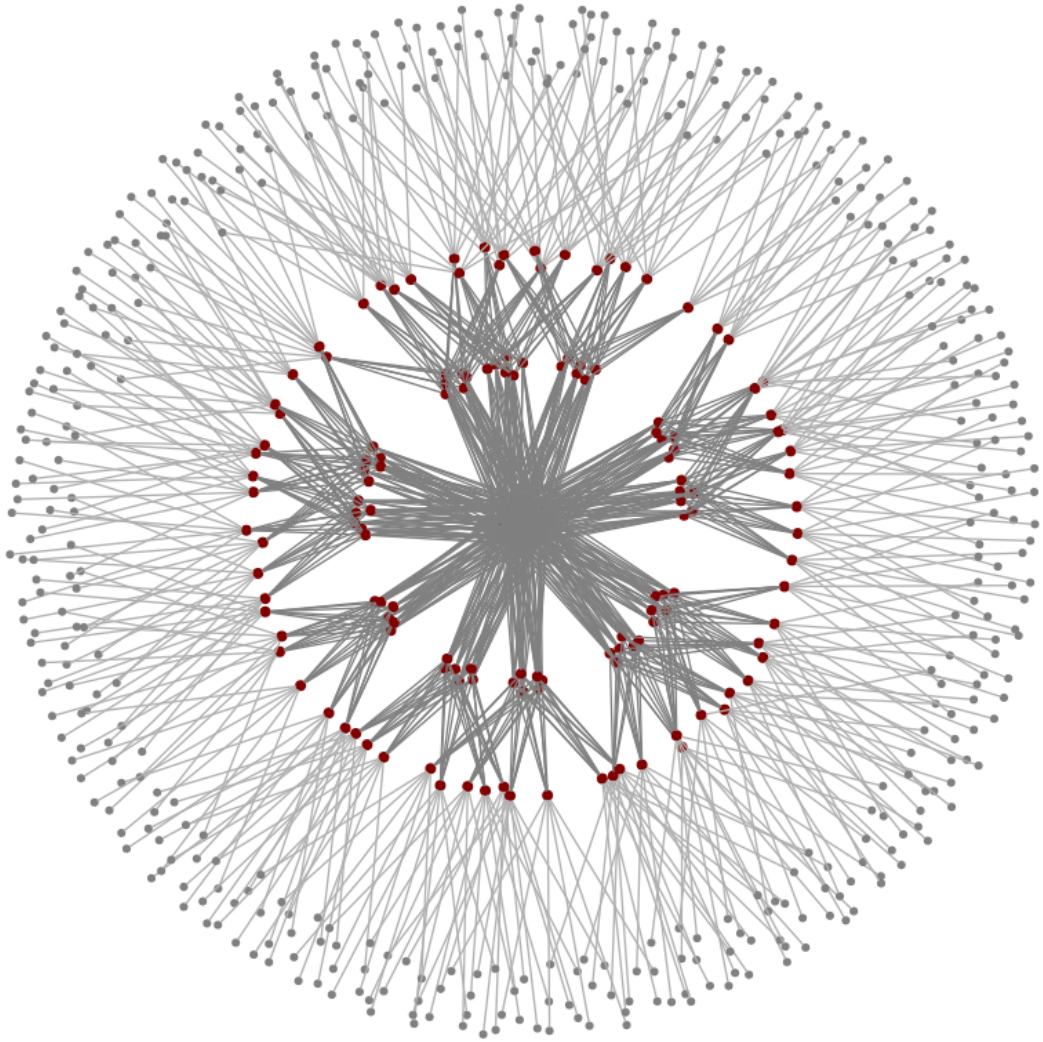
Fat Tree topology example with $n = 4$



Fat Tree topology example with $n = 8$



Fat tree graph



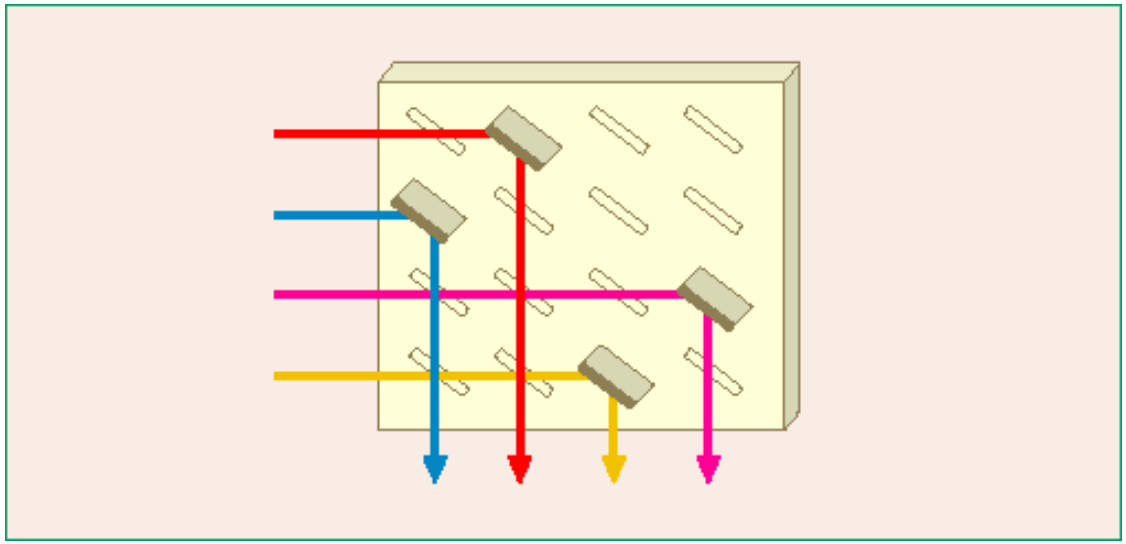
Clos networks

- The DC network topology design bears many similarities with the design of the switch fabric of circuit switched toll offices and ATM switches.
- Let us consider an $N \times N$ switch (N is the number of I/O ports of the switch).
- The issue is how to find a modular design of a fully connected I/O switch fabric that scales as N grows.
- Brilliant and simple proposal made by Clos in 1953.
- Let $N = n \cdot r$ and assume you have $n \times m$, $m \times n$ and $r \times r$ switches.
  - The Clos network is made up of r switches $n \times m$ (first stage), m switches $r \times r$ (middle stage), r switches $m \times n$ (last stage).

Crossbar switch

A conceptual model of an $N \times N$ switch matrix:

- $N \times N$ square array of cross-points
- Cross-points can be implemented in different ways, depending on technology (electronic switches, non-sequential read-write memories, optical reflecting movable mirrors, ...).

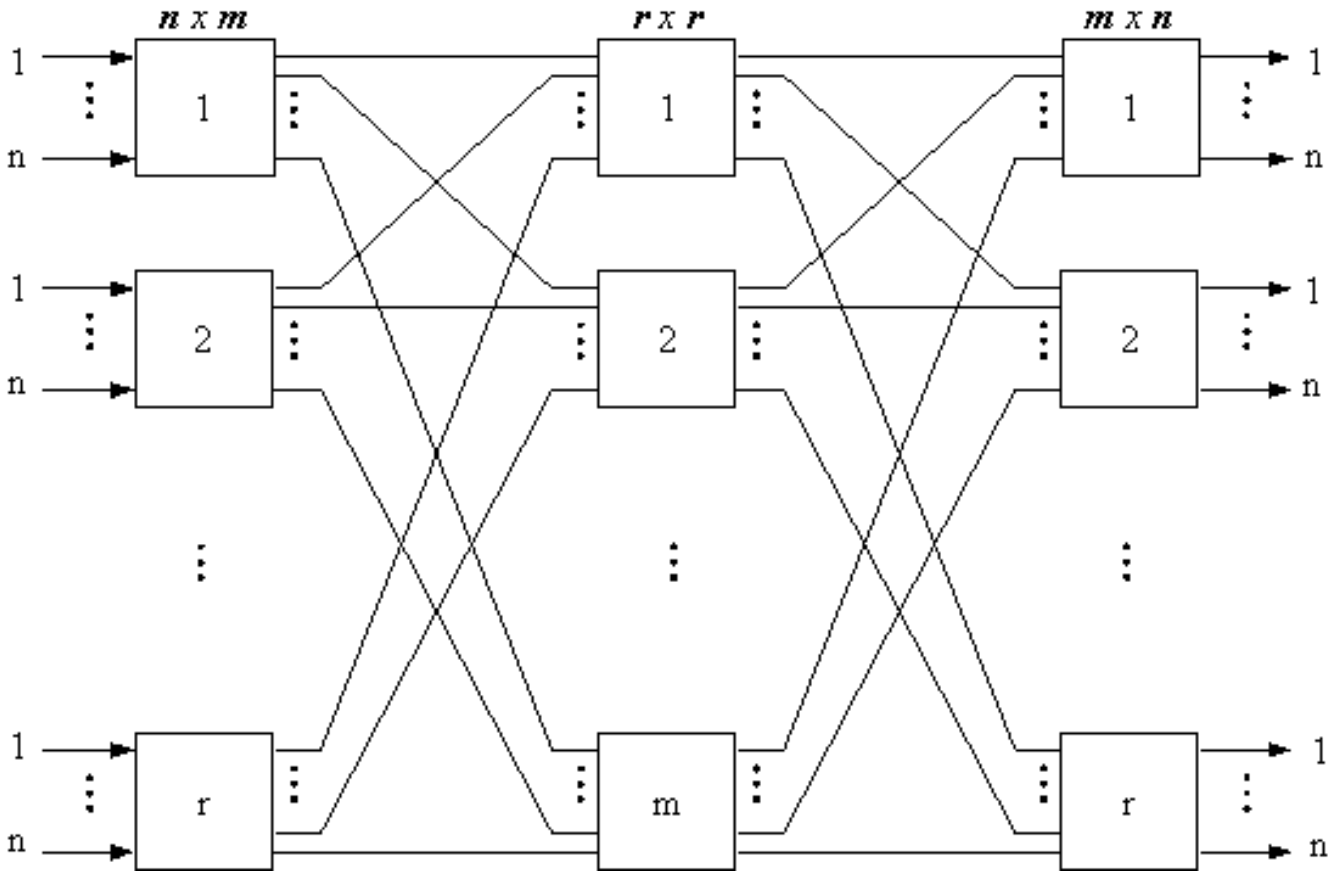


Crossbar switch

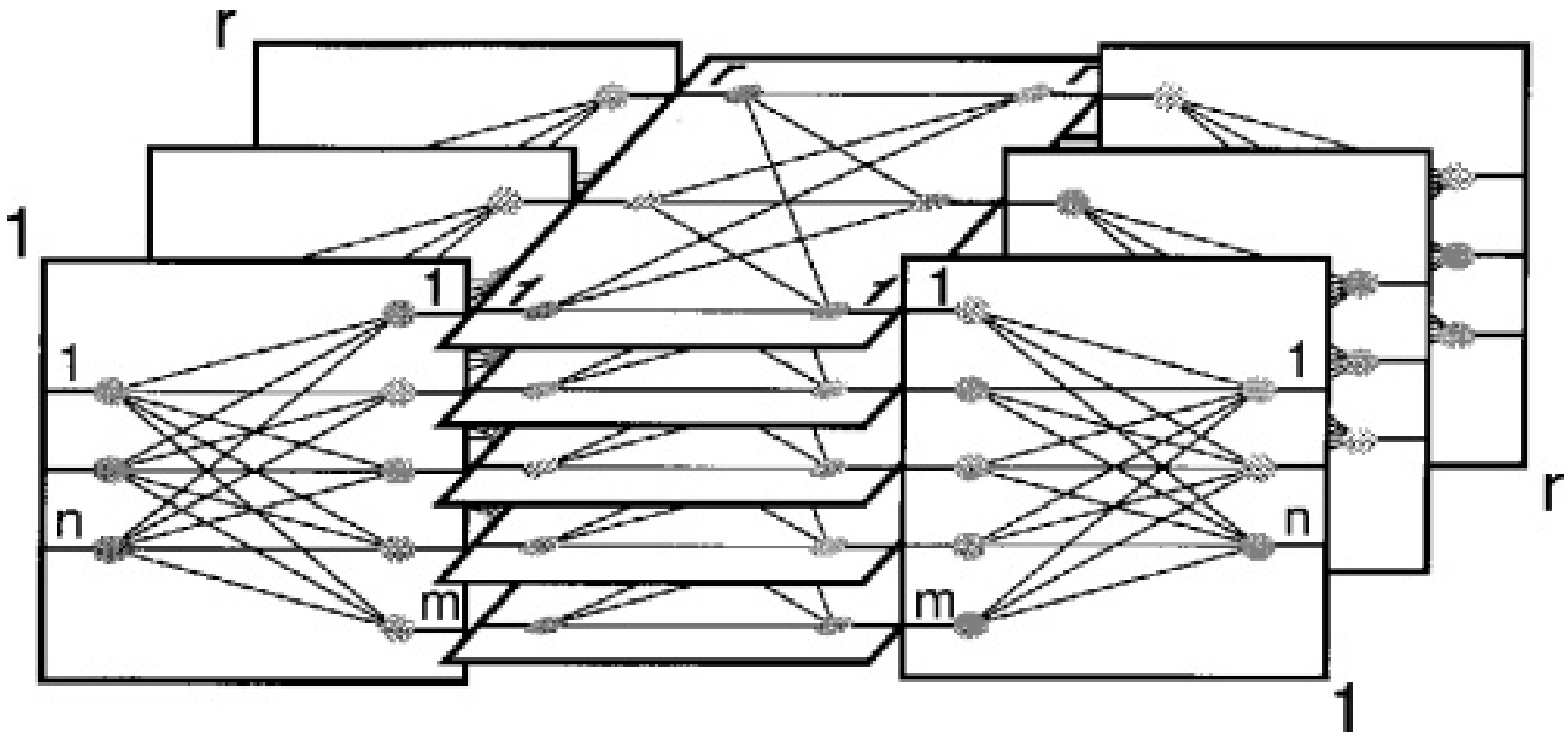
- The cross-bar switch in just one example of a switch fabric connecting N input lines to N output lines ($N \times N$ switch).
- An $N \times N$ switch fabric realizes a permutation of the inputs to the output.
- The complexity of the switch fabric is measured by the number of cross-points.
- For the cross-bar switch the complexity is N^2.

Can we build an $N \times N$ switch fabric as "functional" as the cross-bar switch, but with lower complexity?

Clos network



Clos network: an alternate view



Interconnection network connectivity

- A path between input i and output j is an ordinate sequence of switch cross points and inter-stage links that connects i to j; when i and j are connected, the path resources are assigned to the (i, j) connection and cannot be used for other connections.
- An interconnection network is said to be:
  - **Strictly non-blocking** if, given an input-output couple (i, j), with j an idle output, it is possible to find an available path that connects i to j, whatever be the state of all other input and output lines.
  - **Rearrangeably non-blocking** if, given an input-output couple (i, j), with j an idle output, it is possible to find an available path that connects i to j, possibly moving some existing connection to another path.

Clos network connectivity

The Clos network is strictly non blocking if $m \geq 2n - 1$

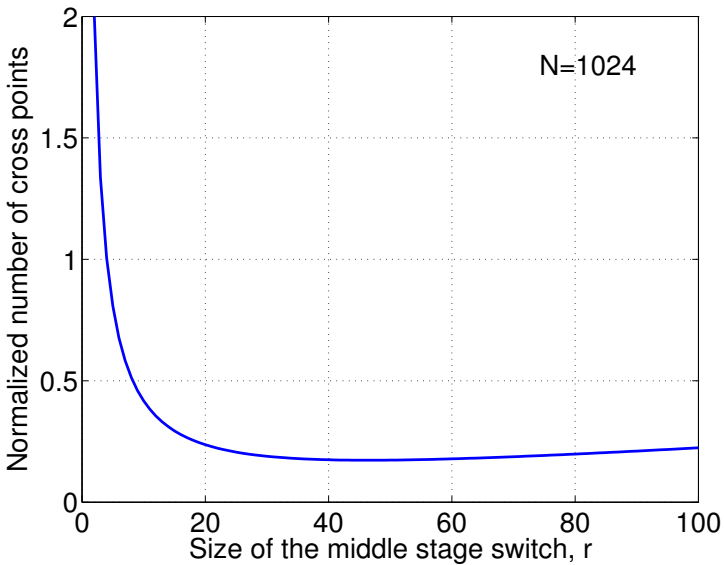
- **Proof** Let us consider a couple (i, j) with i and j idle and belonging to switch S_i and S_j respectively.
  In the worst case $n - 1$ output lines of S_j are already busy.
  Hence there are at least $m - (n - 1) = k \geq n$ available links connecting S_j to k distinct switches of the middle-stage. Let $\mathcal{M}$ denote the set of such middle-stage switches.
  At most $n - 1 < k$ links to the middle-stage switches in $\mathcal{M}$ can be already assigned to paths starting from S_i.
  Therefore, there is at least one available path from i to j.

The Clos network is rearrangeably non blocking if $m \geq n$

Optimization of the non blocking Clos network

- A measure of the network complexity is the number of cross points X.
- For a general Clos network it is $X = 2rmn + mr^2$. With $m = n$, the number of cross points becomes:

$$X = 2rn^2 + nr^2 = N\left(\frac{2N}{r} + r\right)$$



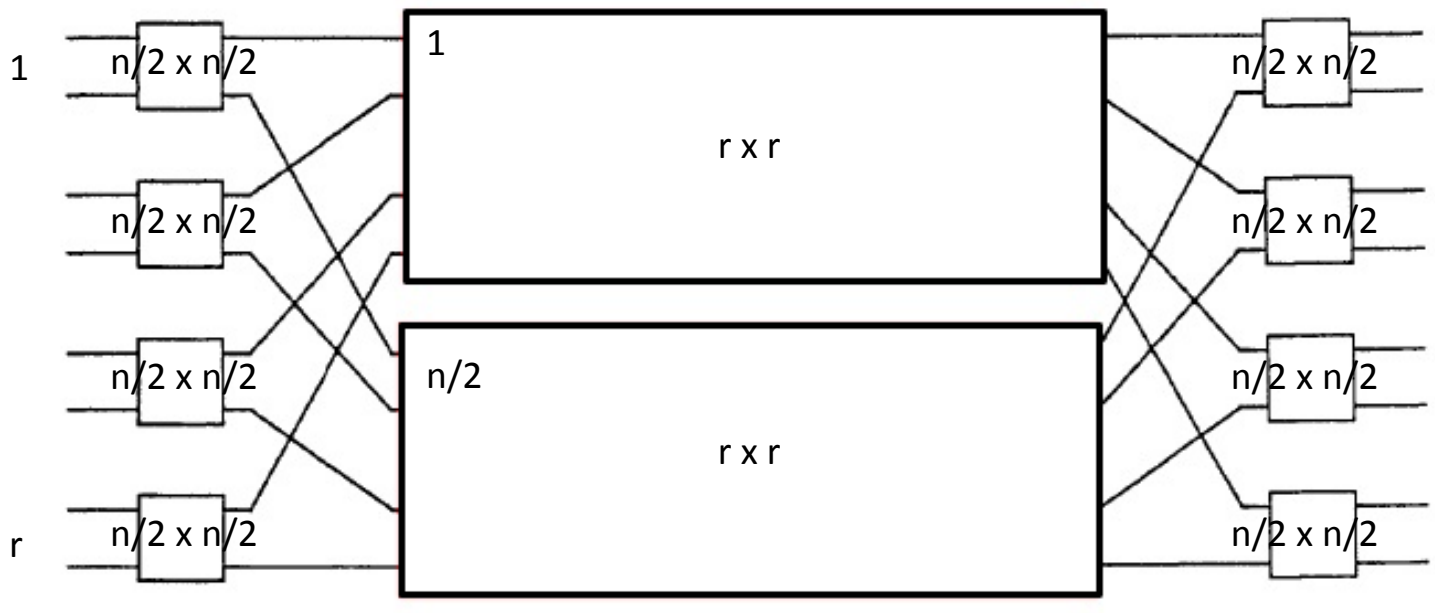
Optimal configuration:

$$r^* = \sqrt{2N}$$

$$X^* = 2N\sqrt{2N}$$

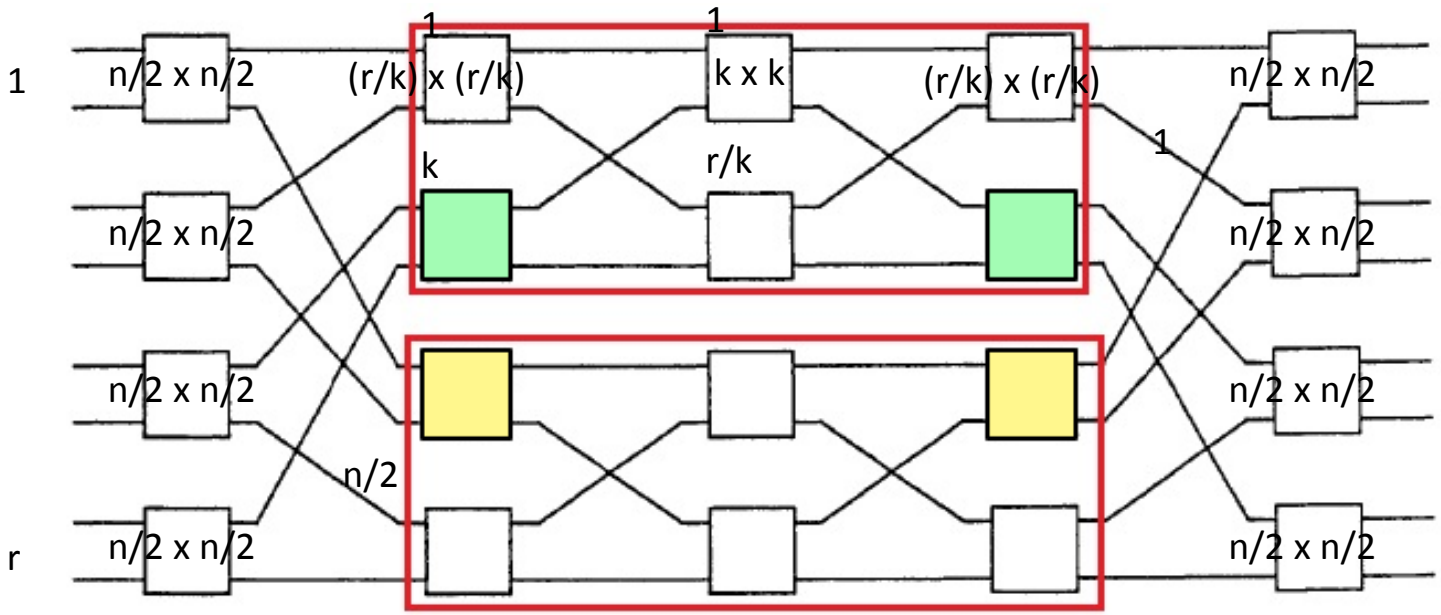
Example construction of a Clos topology - step 1

Let us consider an example Clos network with $m = 2n - 1$ (strictly non-blocking).



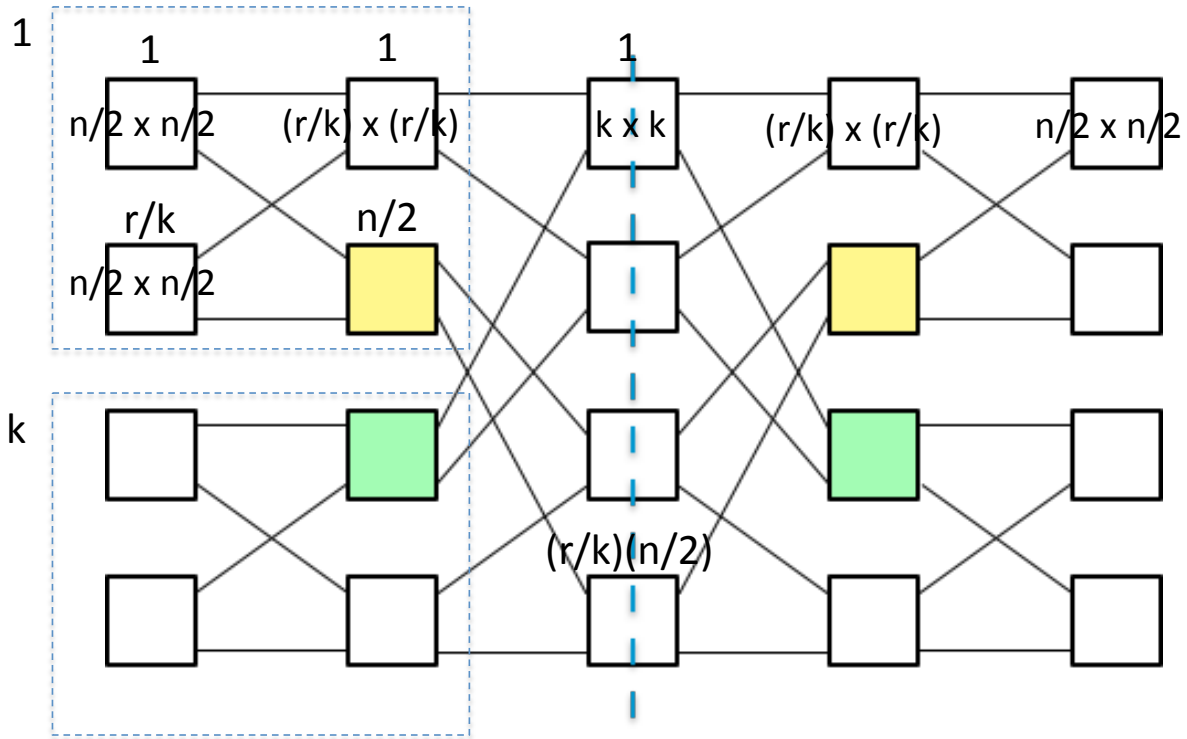
Replace the middle big $r \times r$ switches with a Clos network, with k front switches.

Example construction of a Clos topology - step 2



Now let us swap the yellow switches with the green ones.

Example construction of a Clos topology - step 3



The final step: fold the obtained feed-forward network around the central symmetry vertical axis (dashed blue line) to obtain a fat tree topology with bi-directional links.

Construction of a Clos topology - comments

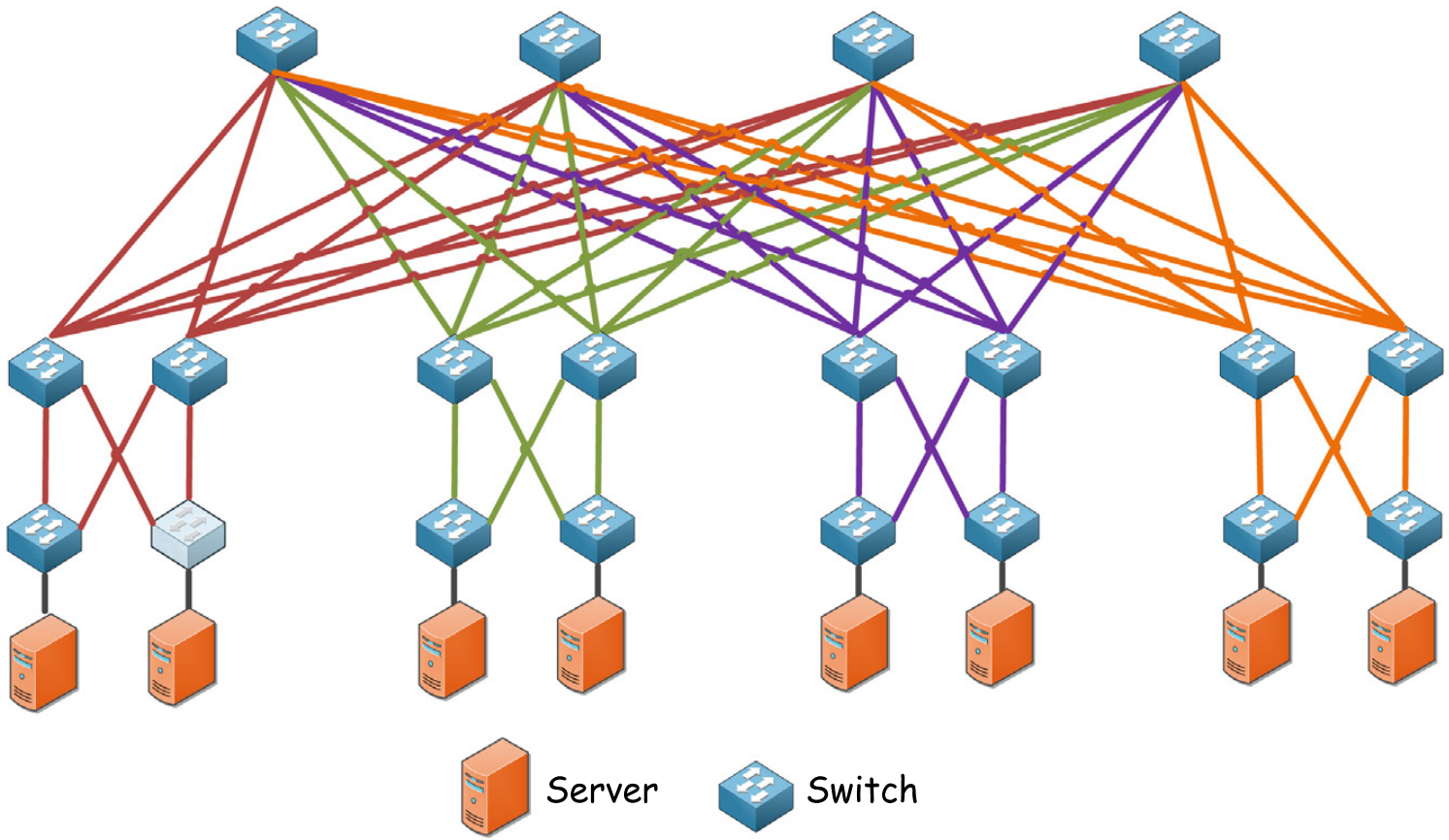
To use only one kind of switch, we require that the number of ports be the same. To that end, it suffices to let

$$\begin{cases} r/k = n/2 \\ k = n \end{cases}$$

Hence, it is $r = n^2/2$.

- Number of servers: $S = n/2 \times r = n^3/4$.
- Number of switches:
  $SW = r + kn/2 + (r/k)(n/2) = n^2/2 + n^2/2 + (n/2)^2 = 5n^2/4$.

VL2 topology

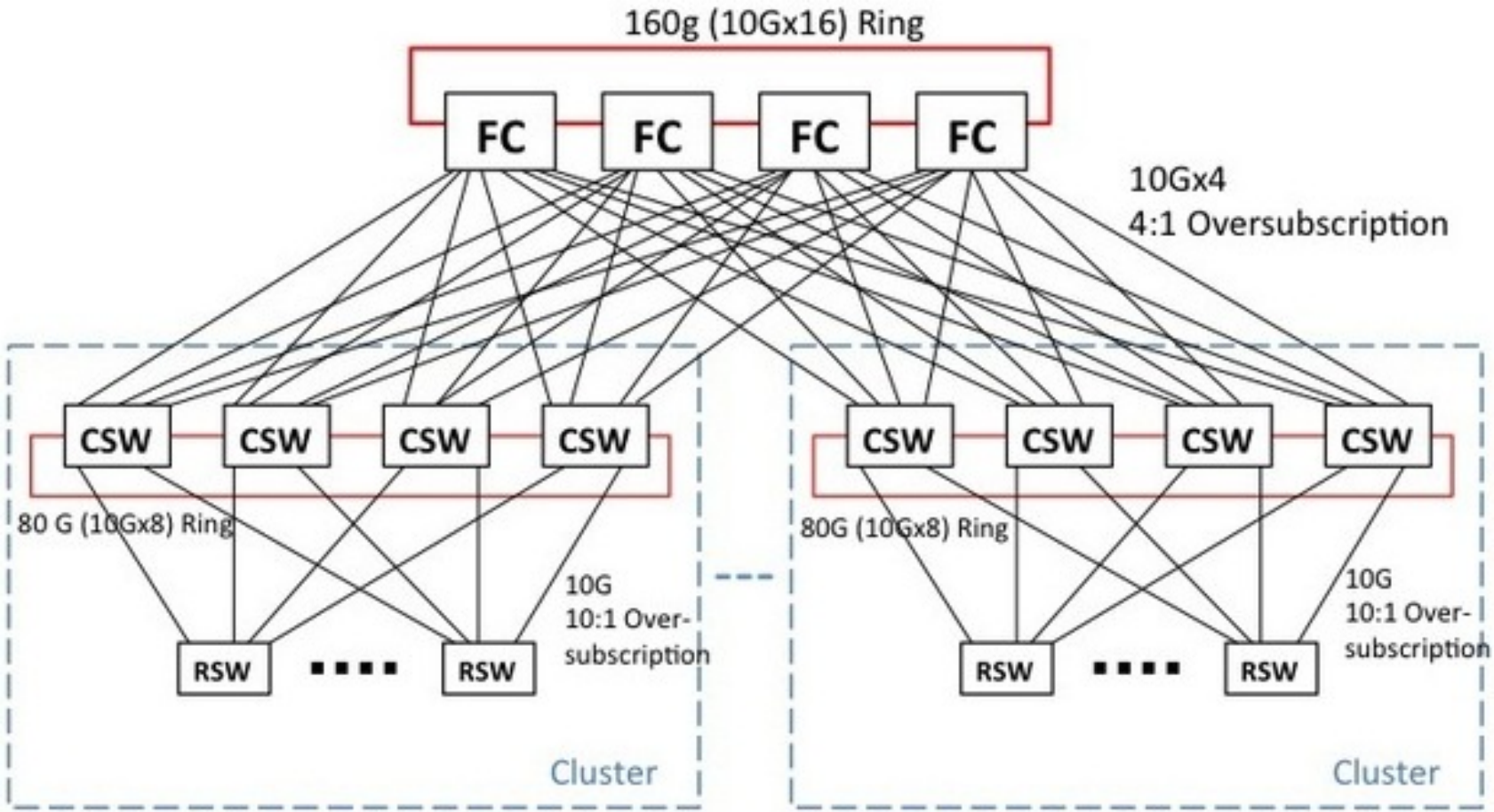


VL2 is an example of Clos network topology.

Facebook 4-post DC network (1/2)

- Components:
  - Rack Switch (RSW);
  - Cluster Switch (CSW);
  - Fatcat Switch (FC).
- Each RSW has up to 48 10G downlinks and 4-8 10G uplinks to CSW (10:1 oversubscription).
- Each CSW has 4 40G uplinks, one to each of the 4 FC aggregation switches (4:1 oversubscription).
- Protection:
  - 4 CSW's are connected in a 10G×8 protection ring.
  - 4FC's are connected in a 10G×16 protection ring

Facebook 4-post DC network (2/2)



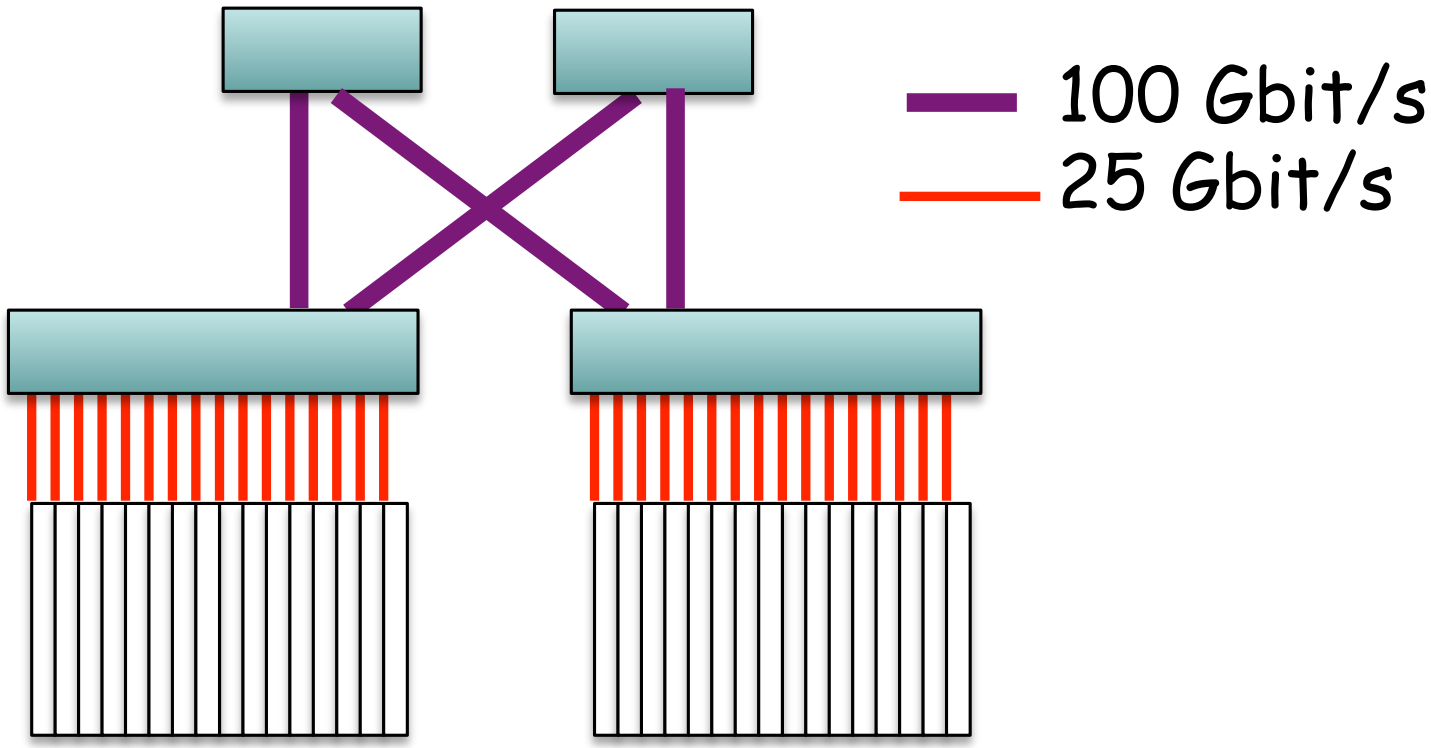
Tree topology limitations

- The higher-tier switches need to support data communication among a large number of servers.
- The number of servers in a tree architecture is limited by the numbers of ports on the switches.
- Switches with higher performance and reliability may be required.
- Oversubscription issue.

Oversubscription

- Assume we use 16 25 *Gbps* (25G) ports ToR switches.
  - 16 servers are connected to each ToR switch via 25G links.
  - If the ToR switch is connected to two aggregation-level switches by means of 100 *Gbps* (100G) ports, oversubscription takes place, i.e. the maximum aggregate bit rate offered by the servers ($16 \times 25 = 400$ *Gbps*) exceeds the capacity available from the ToR switch to the upper switching layer ($2 \times 100 = 200$ *Gbps*).
- Oversubscription becomes heavier as we move to higher tier switches.
  - E.g., if a core switch connects to 4 aggregation switches that in turn connect to 2 ToR switches each, the aggregated link rate to/from the core switch is $4 \times 2 \times 100$ *Gbps* $= 800$ *Gbps*.

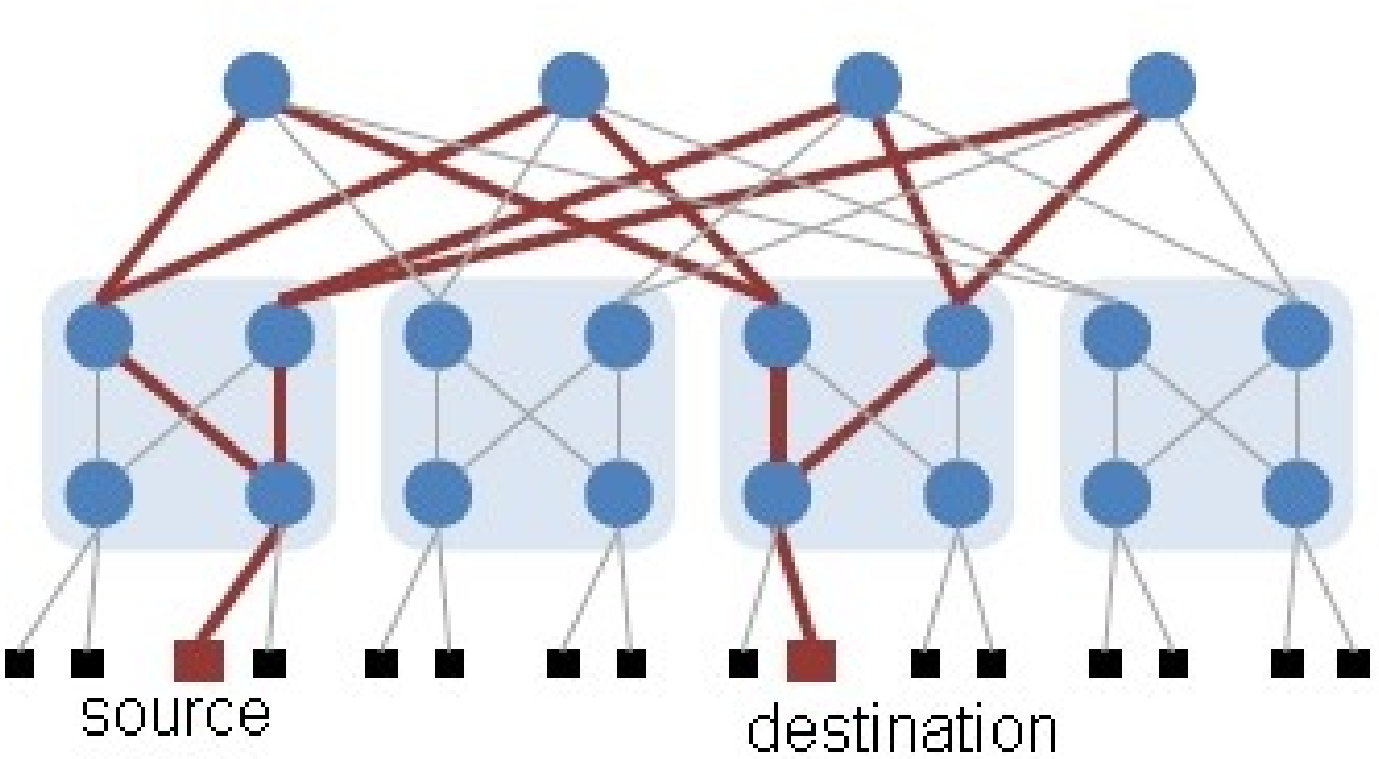
Example of oversubscription



Fat Tree forwarding and routing

- All inter-pod traffic goes through core switches.
- $(n/2)^2$ shortest paths exists between any two servers in different pods.
- Each shortest path has 6 hops:
  - 3 hops from the source up to a core switch;
  - 3 hops from the core switch down to the destination.
- The main goal of routing in fat-tree is to distribute the traffic evenly among core switches.

Fat Tree server-server paths



An application-oblivious throughput bound

- Let x_i denote the bit rate of the data flow of the i-th source-destination pair.
- Assume there are ℓ links and all links of the graph have the same capacity C.
- The normalized throughput of the network TH is defined as

$$TH \equiv \frac{\min_i x_i}{C}$$

- The following bound can be shown:

$$TH \leq \frac{\ell}{\overline{h}\nu_f}$$

where $\overline{h}$ is the average path length and ν_f is the number of traffic flows.

Proof of the bound

- The network has a total of ℓ links (counting both directions), hence a capacity of $\ell \cdot C$.
- The i-th flow x_i using a shortest path of length h_i consumes at least a capacity $x_i h_i$.
- The total capacity consumed is $\sum_{i=1}^{\nu_f} x_i h_i \geq C \cdot TH \sum_{i=1}^{\nu_f} h_i$, by the definition of throughput.
- By definition, it is $\overline{h} = (\sum_{i=1}^{\nu_f} h_i)/\nu_f$, hence we derive $\sum_{i=1}^{\nu_f} x_i h_i \geq C\ TH \overline{h} \nu_f$.
- The overall capacity consumed $\sum_{i=1}^{\nu_f} x_i h_i$ cannot exceed the available capacity ℓC, hence it must be $C\ TH \cdot \overline{h} \nu_f \leq \sum_{i=1}^{\nu_f} x_i h_i \leq \ell C$, and the claimed result follows.

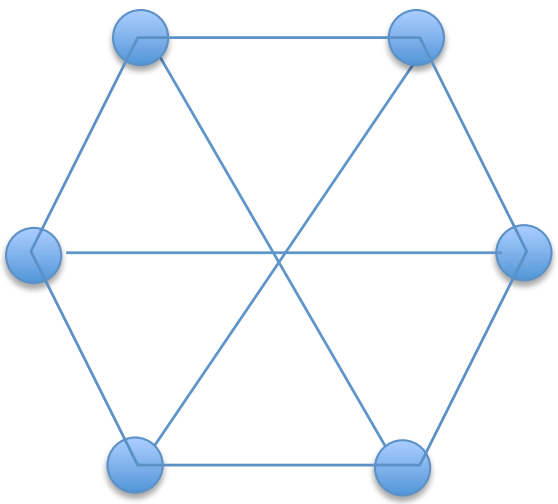
Bound for r-regular graphs

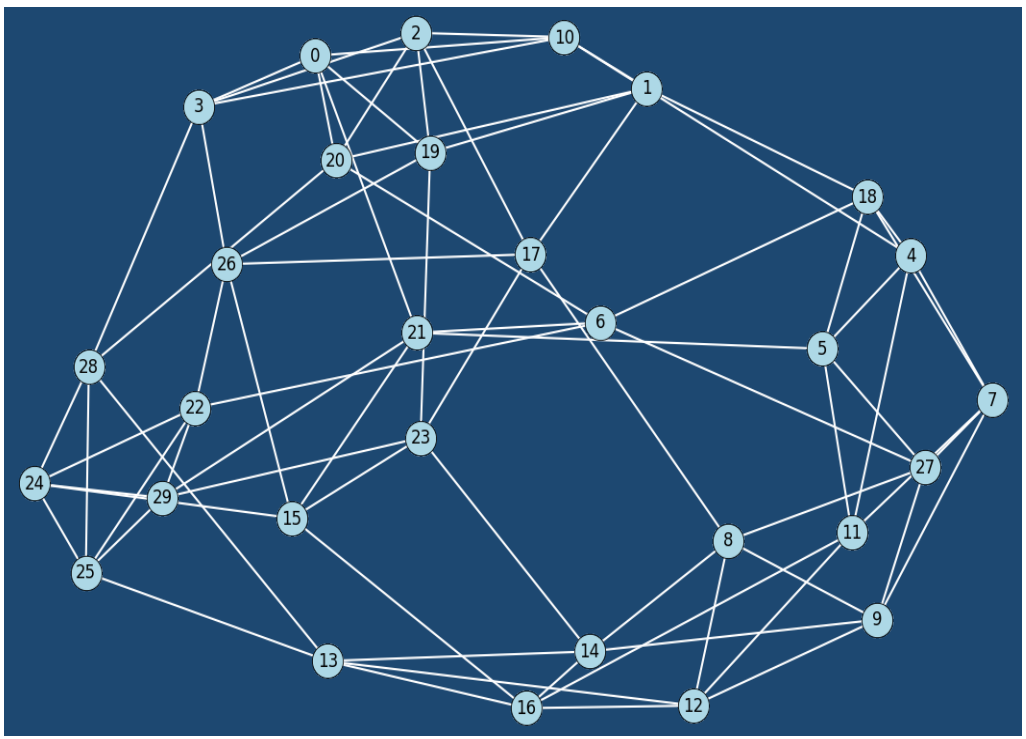
- As a special case, let us consider an N node $r-$regular graph.
  - A graph where each node has exactly r neighbors.
  - The number of nodes must be at least $N \geq r+1$ and the product $N \cdot r$ must be even (why?).
  - We also assume $r \geq 3$ (graphs with $r = 0, 1, 2$ are trivial [why?]).
- Each switch has n ports, r of which are used to connect with other switches, the remaining $n - r$ to servers. Then, $S = N(n - r)$ servers can be accommodated.
- It is $\ell = Nr$. Then, the bound for this class of graphs can be stated as:

$$TH \leq \frac{Nr}{\overline{h}\nu_f}$$

r-regular graphs

Examples of r regular graphs with $r = 3$ and $N = 6$ (left) and $r = 5$ and $N = 30$ (right).





Bound on $\overline{h}$

Cerf et alii (1974) proved the following bound on the average shortest path length of an r-regular graph of N nodes:

$$\overline{h} \geq \frac{\sum_{j=1}^{k-1} jr(r-1)^{j-1} + kR}{N-1}$$

where k is the largest integer such that $R \geq 0$ with

$$R = N - 1 - \sum_{j=1}^{k-1} r(r-1)^{j-1}$$

i.e., $k = 1 + \left\lfloor \frac{\log(N-2(N-1)/r)}{\log(r-1)} \right\rfloor$.

Recursive topologies

- Tree-based topologies can scale up by inserting more levels of switches, while each server is connected to only one of the bottom level switches.
- Recursive topologies use lower level structures as cells to build higher level structures.
- The servers in recursive topologies may be connected to switches of different levels or even other servers.
- There are multiple network ports on the servers of recursive topologies.
- We will touch two proposals:
  - DCell
  - BCube

DCell

- The most basic element of a DCell is called DCell_0.
  - A DCell_0 consists of n servers and one n-port switch.
  - Each server in a DCell_0 is connected to the switch in the same DCell_0.
- Construction of a DCell_1 from $n+1$ DCell_0's:
  - Each server of every DCell_0 in a DCell_1 is connected to a server in another DCell_0 of the same DCell_1, respectively.
  - As a result, the DCell_0's are connected to each other, with exactly one link between every pair of DCell_0's.
- In a DCell_k, each server will eventually have $k+1$ links
  - the first link (level-0 link) connected to a switch when forming a DCell_0;
  - level-i link connected to a server in the same DCell_i but a different DCell_{i-1}, $i \geq 1$.

Recursive construction

- The DCell_k is built from a collection of DCell_{k-1}.
- Let t_{k-1} denote the number of servers in a DCell_{k-1}.
- A link is defined between a server in DCell_{k-1} and a server in each one of the other DCell_{k-1}'s, so that the resulting link topology among the DCell_{k-1}'s of the DCell_k is a full mesh.
- Since we want to connect the DCell_{k-1}'s in a full mesh with a single arc between each couple of DCell_{k-1}'s, the maximum number of DCell_{k-1} in a DCell_k is $t_{k-1} + 1$.

Size of DCell

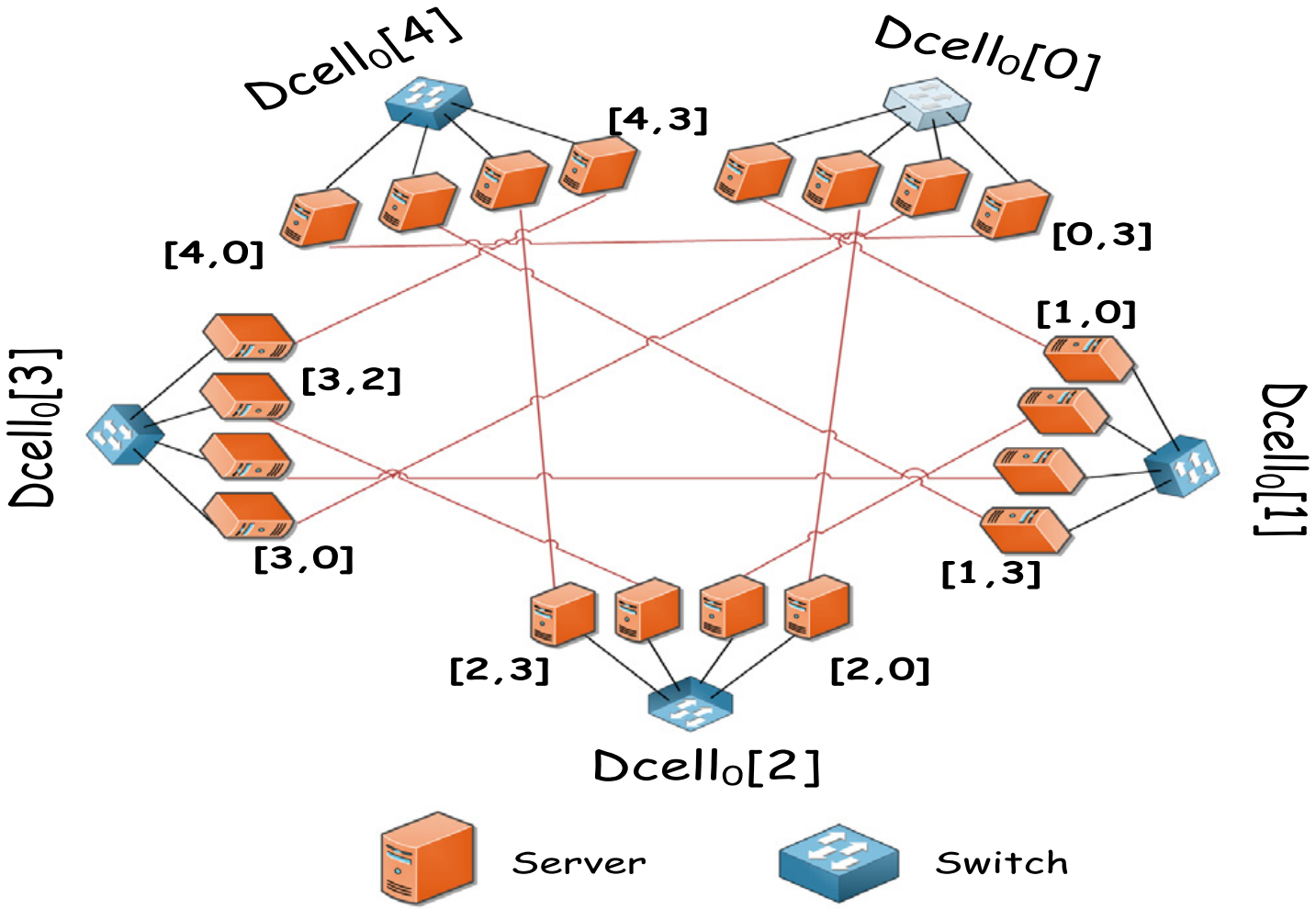
- Let t_k denote the number of servers of a DCell$_k$.
- Let g_k denote the number of DCell$_{k-1}$'s making up a DCell$_k$.
- We let $t_0 = n$ and $g_0 = 1$. Then, for $k \geq 1$, we have

$$t_k = g_k \cdot t_{k-1}$$

$$g_k = t_{k-1} + 1$$

It can be verified that $t_k, g_k \sim n^{2^k}$ as k grows.

Example of DCell topology



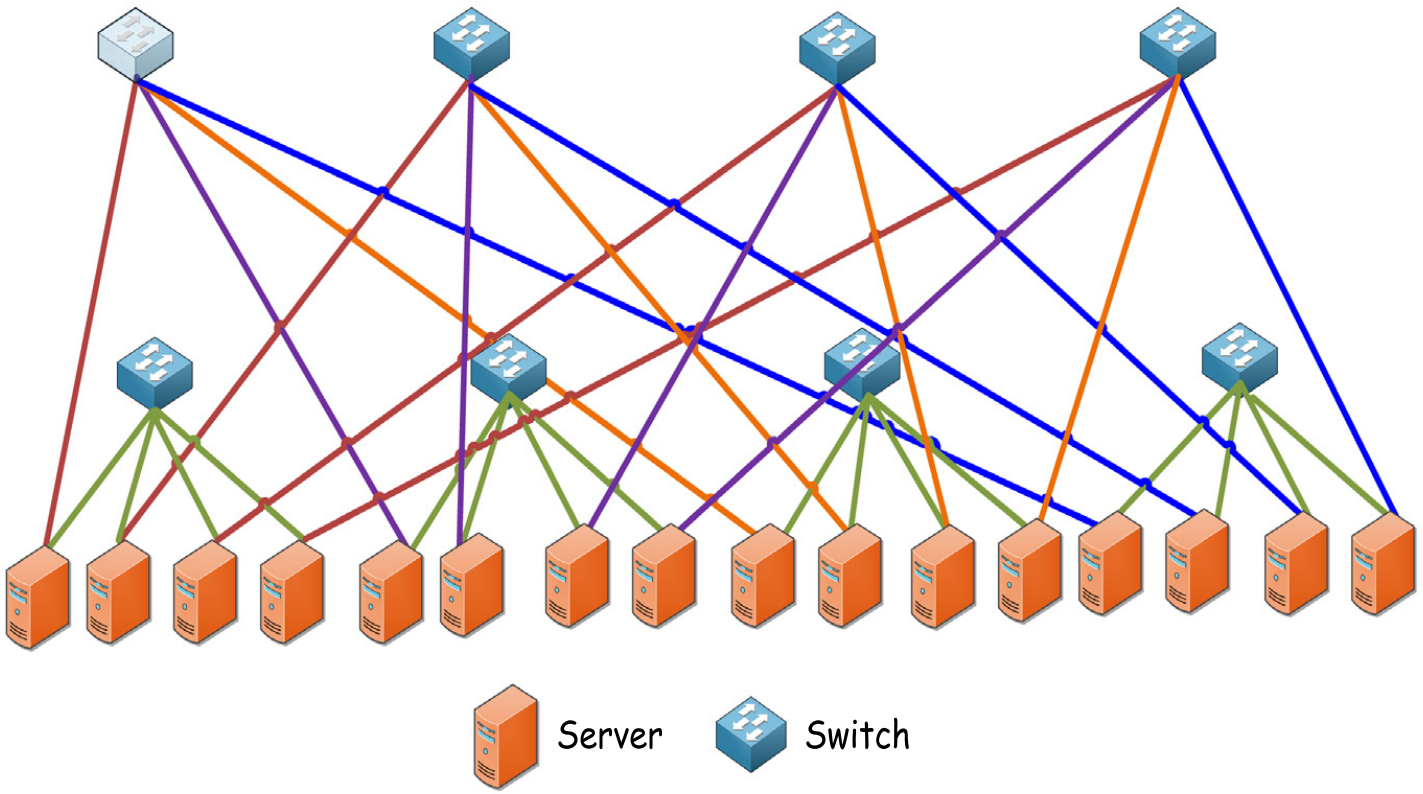
BCube topology

- BCube is a recursive topology specially designed for container-based, modular data centers.
- The BCube_0 is the most basic element of a BCube: it consists of n servers connected to one n-port switch.
- A BCube_1 is constructed by taking n BCube_0's and n extra switches.
- In general, a BCube_k is constructed from n BCube_{k-1}'s and n^k extra n-port switches.
  - The extra switches of the BCube_k are connected to exactly one server in each BCube_{k-1}.
  - The number of servers of a BCube_k is $t_k = n \cdot t_{k-1}$. Since $t_0 = n$, we have $t_k = n^{k+1}$.
  - The number of switches of a BCube_k is $s_k = n \cdot s_{k-1} + n^k$. Since $s_0 = 1$, we have $s_k = (k+1)n^k$.

BCube vs DCell

- BCube makes use of more switches when constructing higher level structures, while DCell uses only level-0 n-port switches.
- Both require servers to have $k + 1$ NICs.
- The implication is that servers will be involved in switching more packets in DCell than in BCube.
- The number of levels k depends on the number of ports on the servers both in DCell and in BCube.
- The number of servers in BCube grows exponentially with the levels, much slower than DCell.

Example of BCube topology

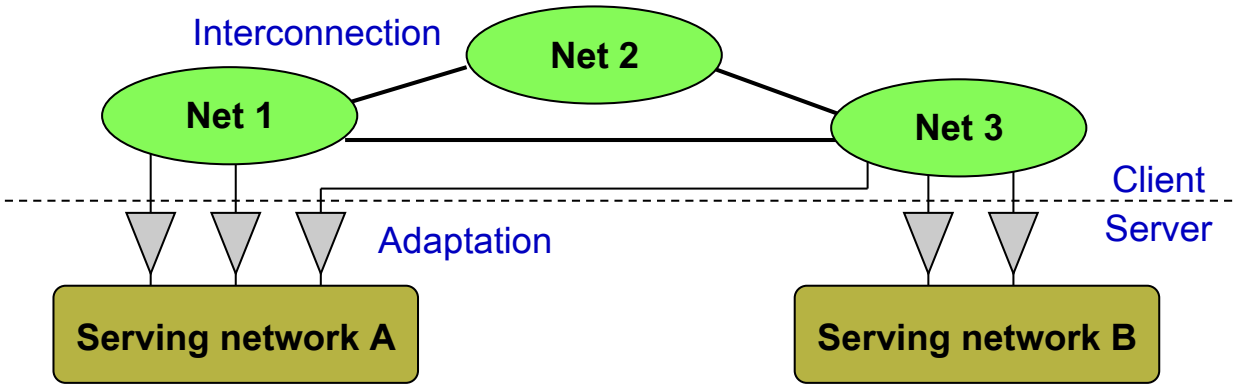


Flexible topologies

- Optical switching technology has been considered to construct DCNs.
- Main reasons why:
  - High bandwidth (up to Tbps per fiber with WDM techniques).
  - Significant flexibility of reconfiguring the topology during operation.
- Some literature proposals: c-Through, Helios, OSA, Petabit.

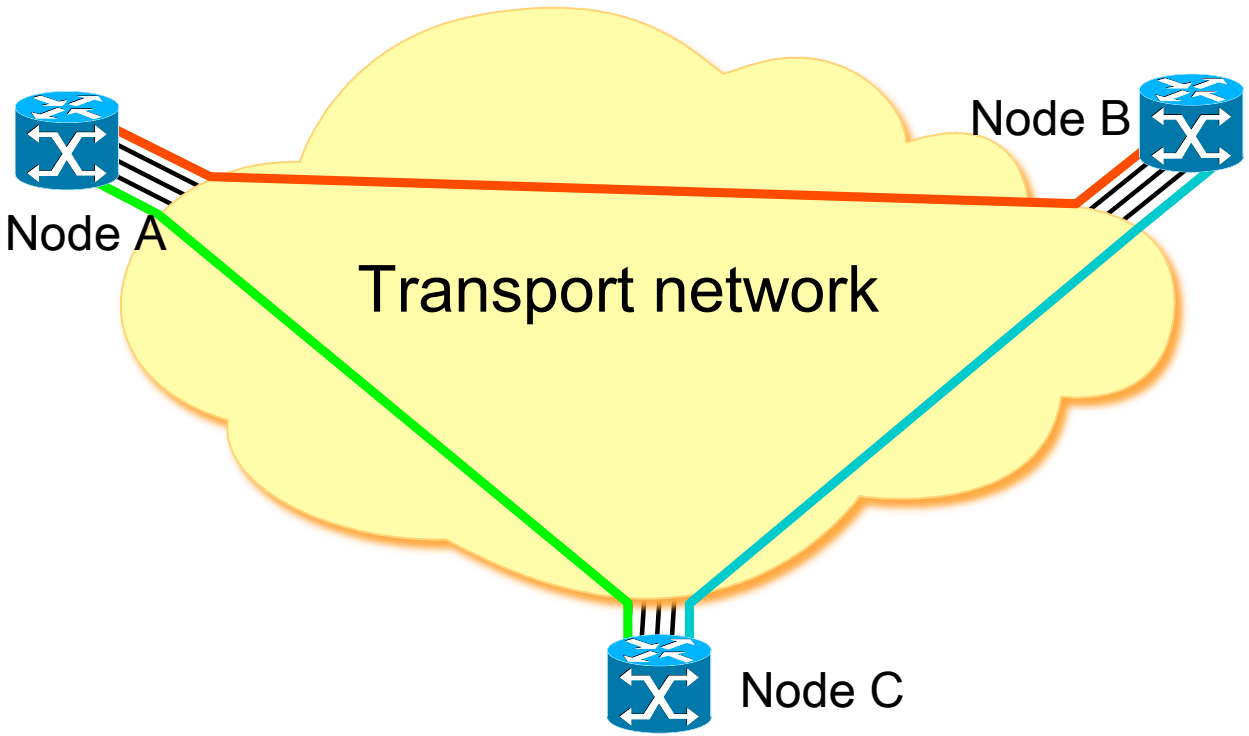
Network virtualization concept

- Basic principles to decompose the network design, analysis, management problems:
  - layering;
  - standard interfaces.
- **Layered networks**
  - Client network.
  - Serving network.
- **Network partition**
  - Peer inter-connected networks.



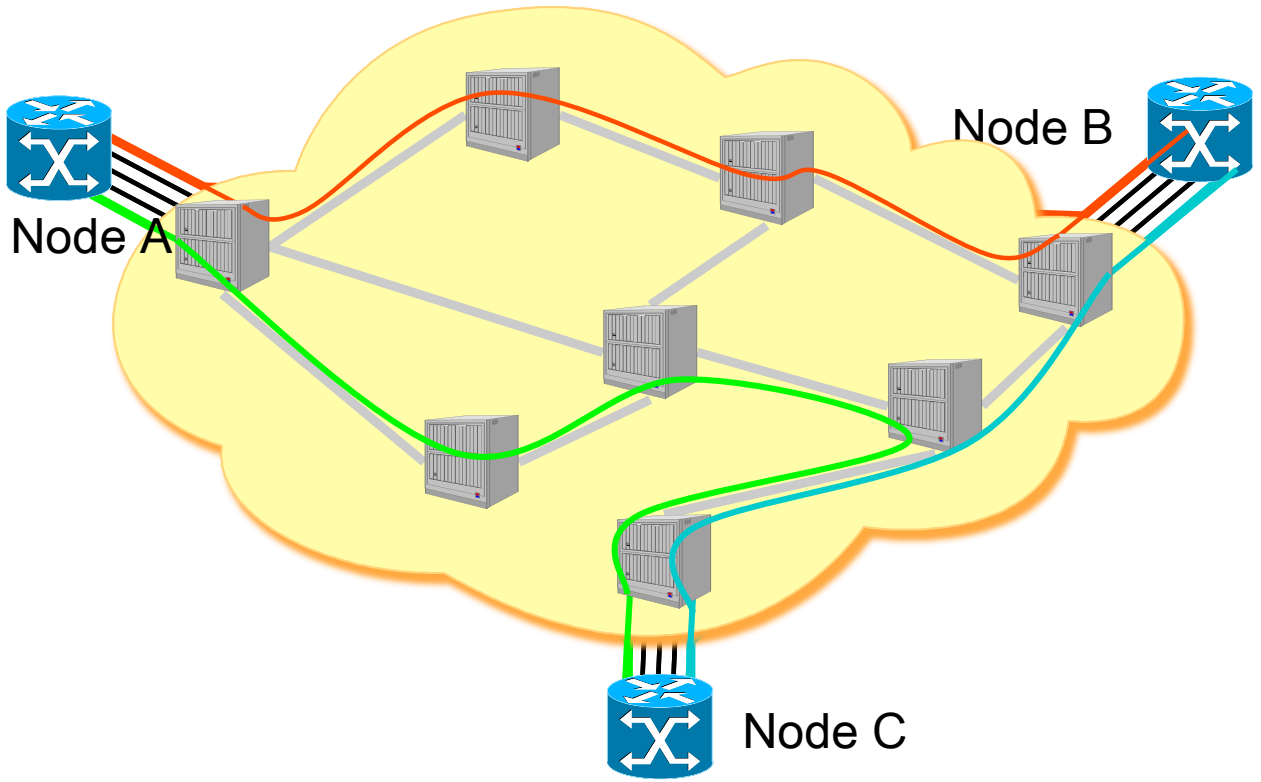
Network virtualization example (1/2)

The IP level view

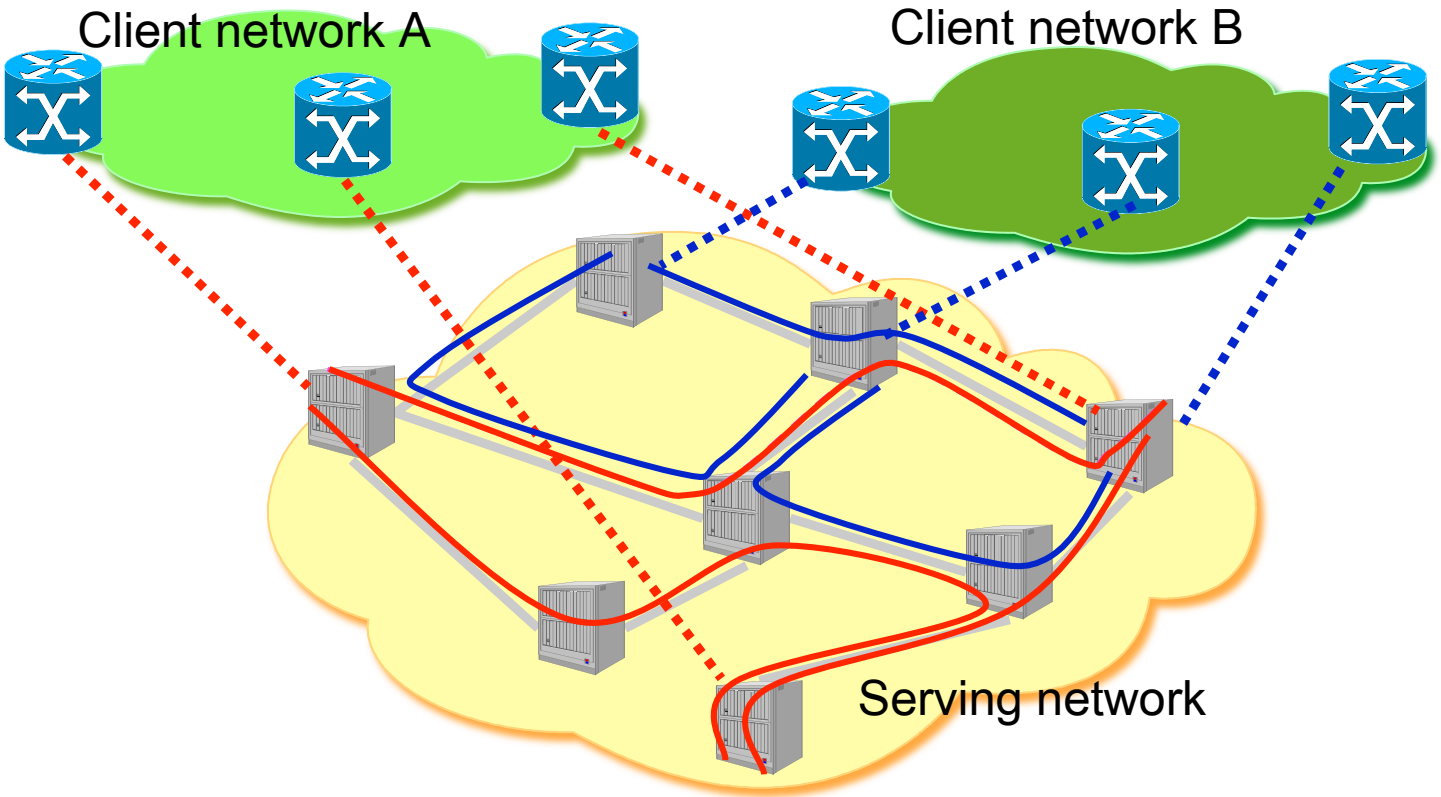


Network virtualization example (2/2)

The transport network level view



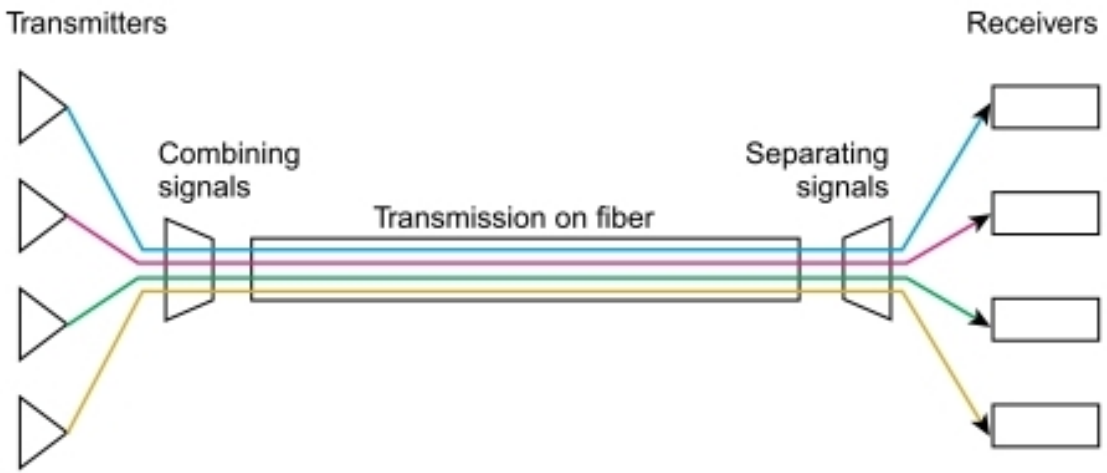
Client and serving network example



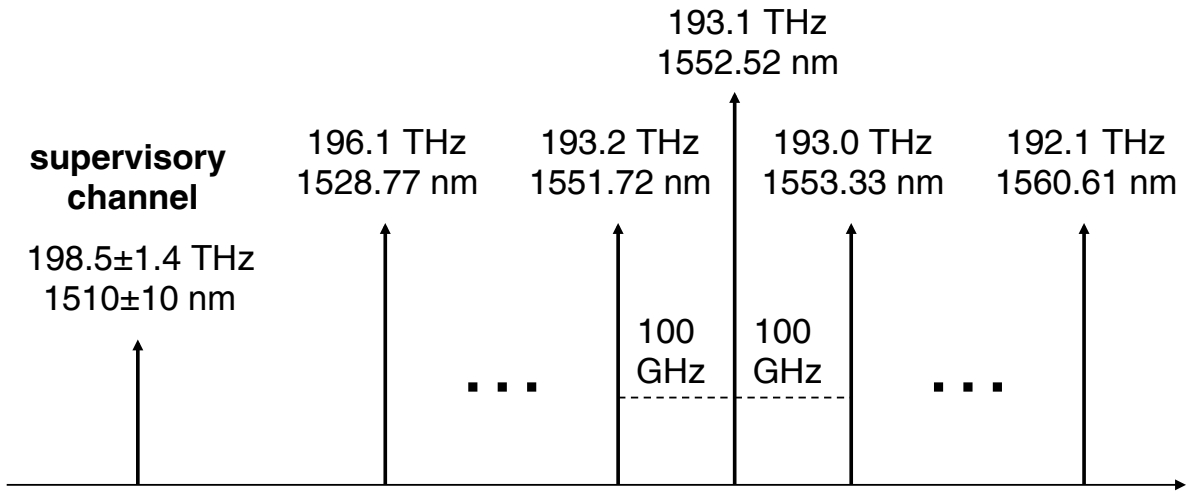
DWDM

Dense Wavelength Division Multiplexing

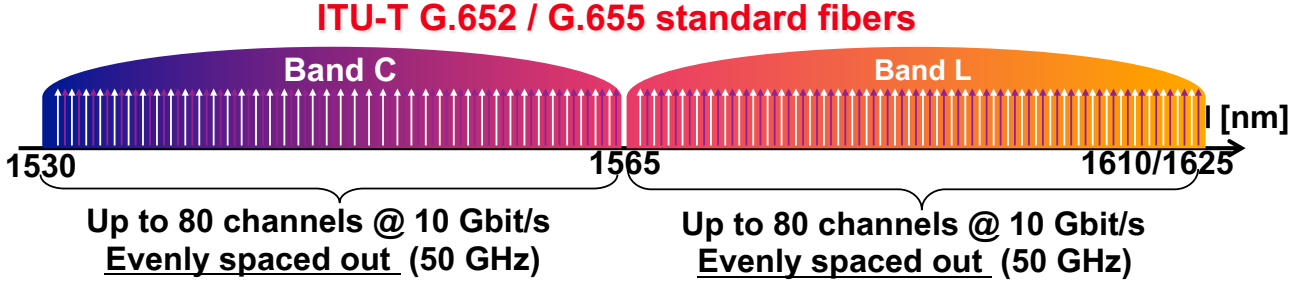
- A form of frequency multiplexing used for optical fiber communication links.
- The most used frequency band in mono mode fibers is the so called 3rd window.
  - Minimum of attenuation ($\sim 0.2\ dB/km$).
  - Limited dispersion (few $ps/nm/km$).



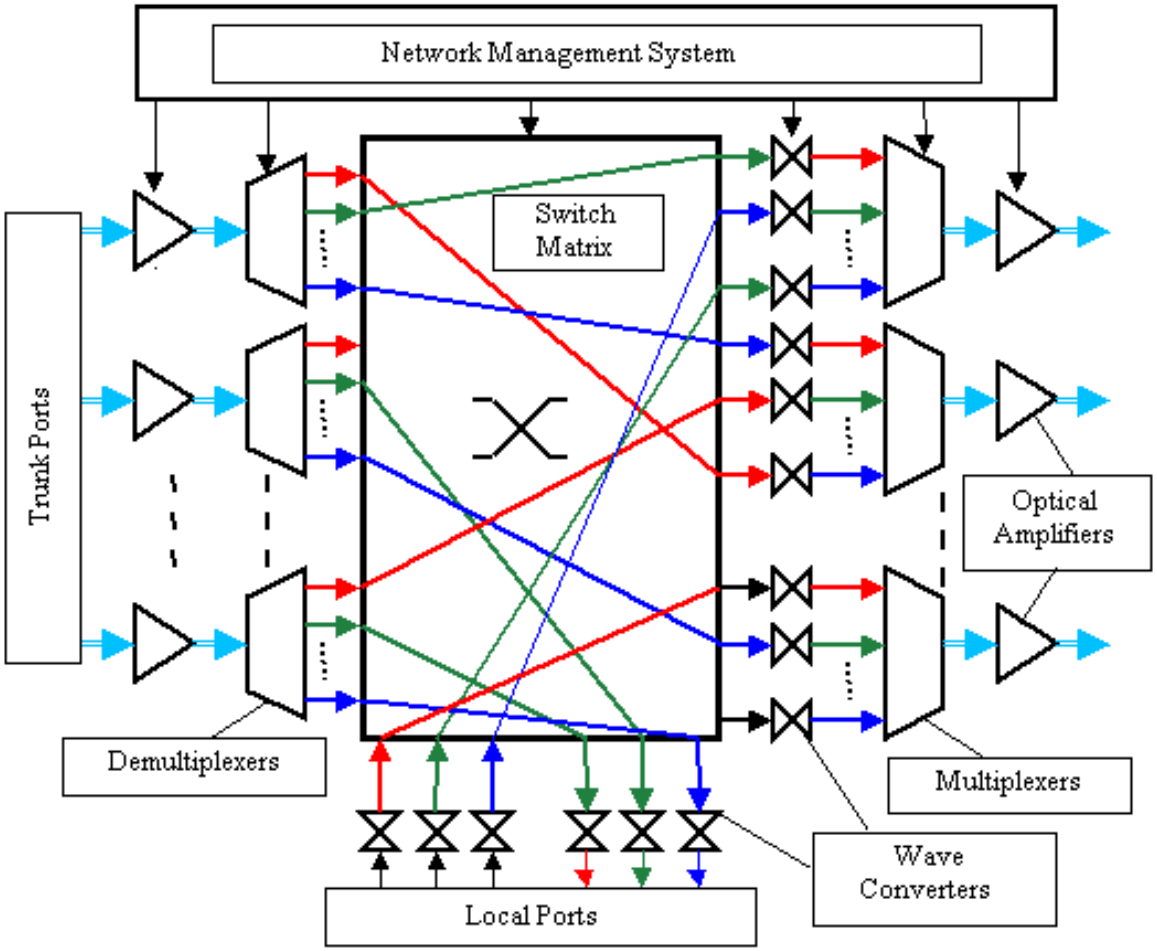
DWDM carrier frequency grid



Reference DWDM carrier grid (ITU-T G.692, 100 GHz spacing)

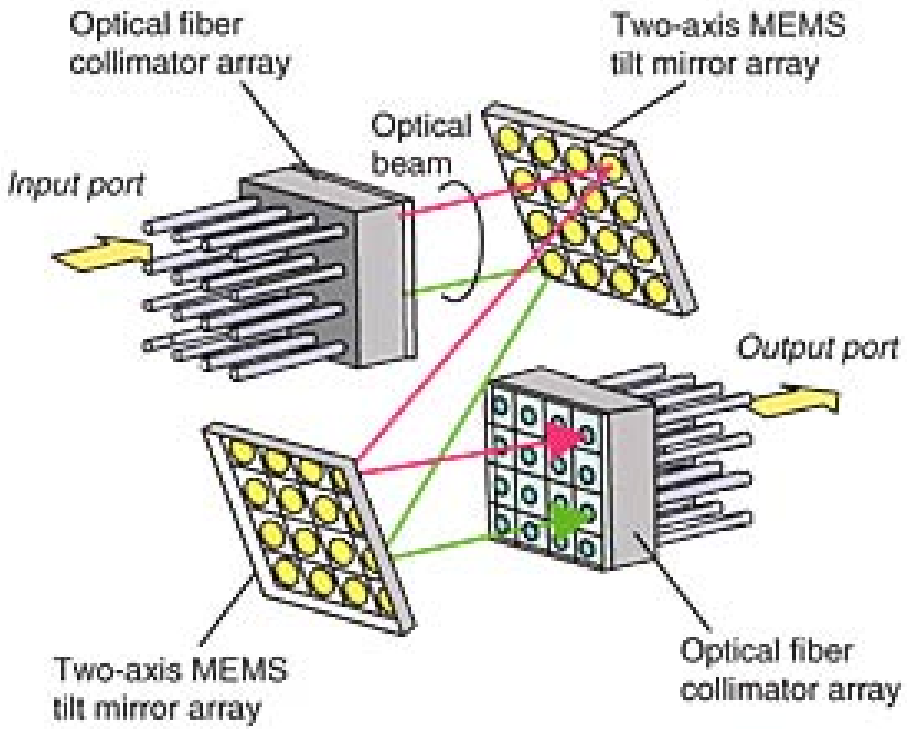


DWDM Optical Cross Connect (OXC)

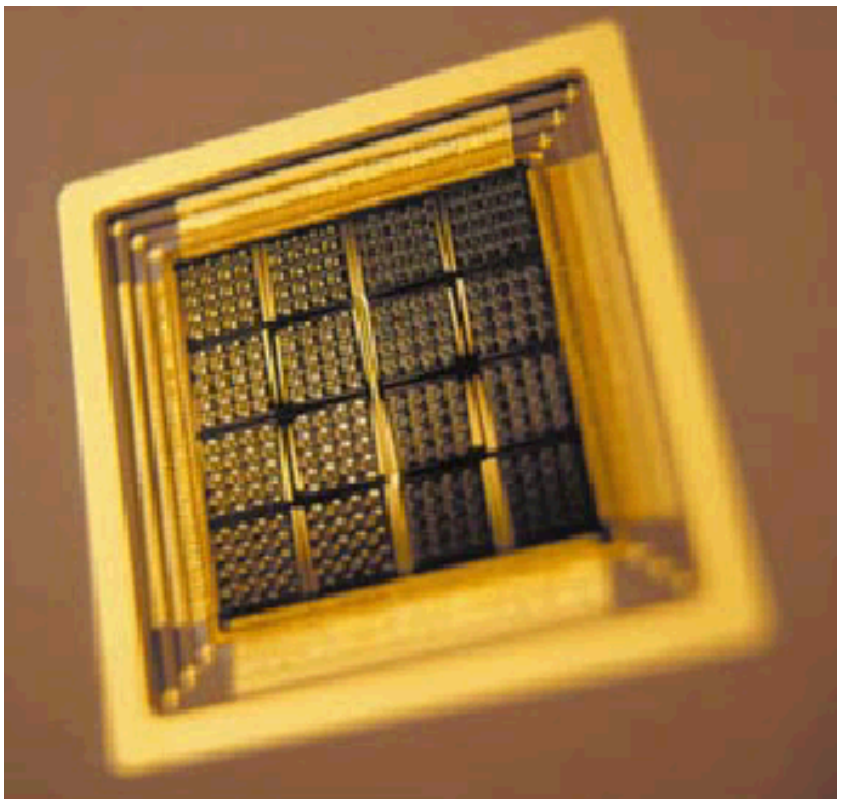


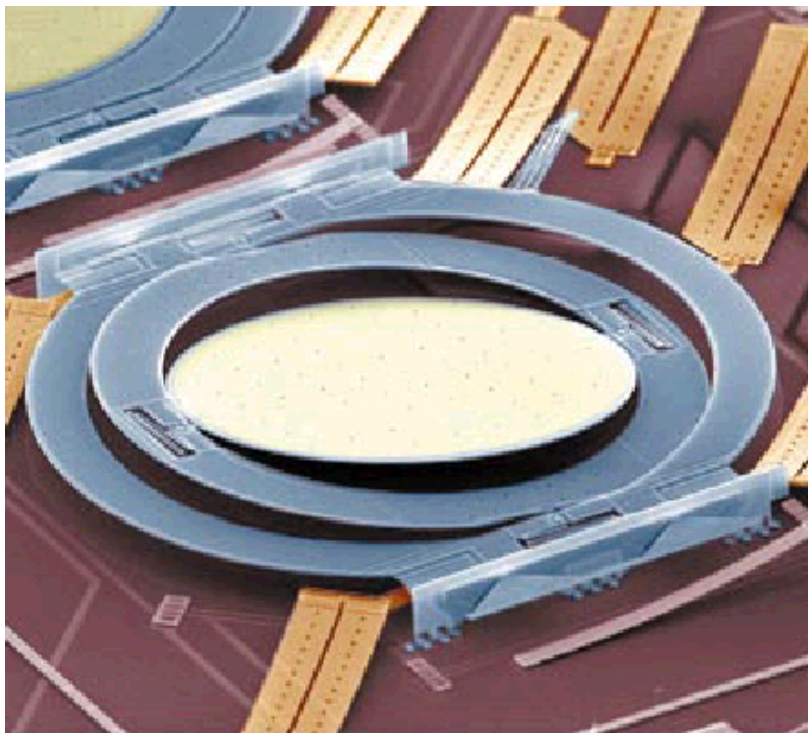
DWDM MEMS (1/3)

MEMS = Micro Electro-Mechanical System

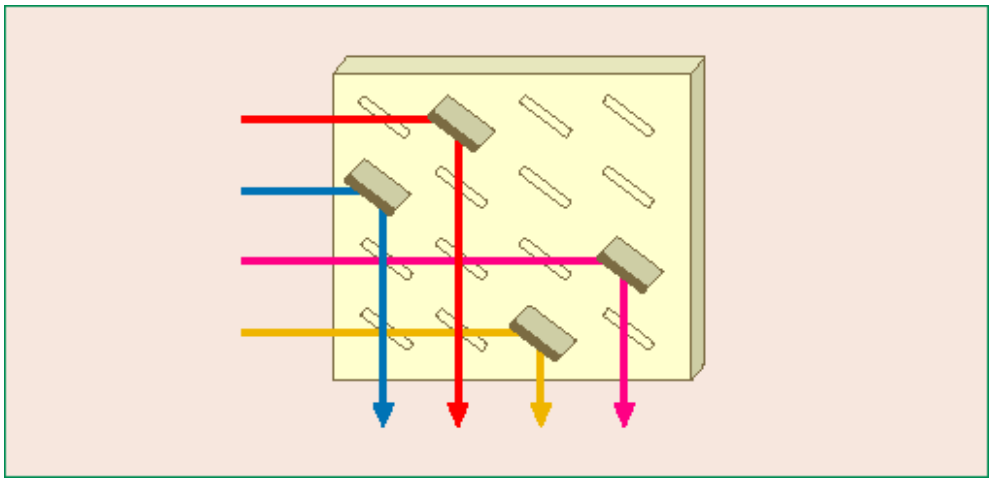


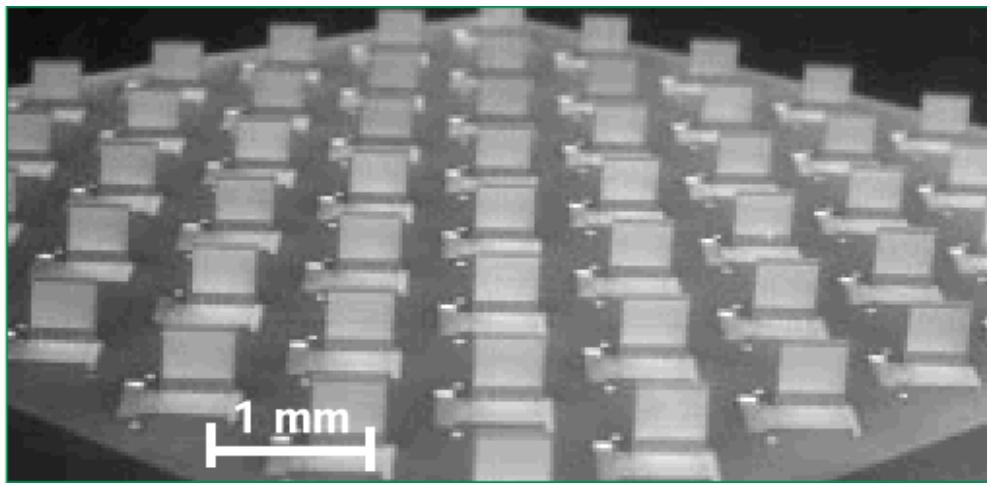
DWDM MEMS (2/3)





DWDM MEMS (3/3)

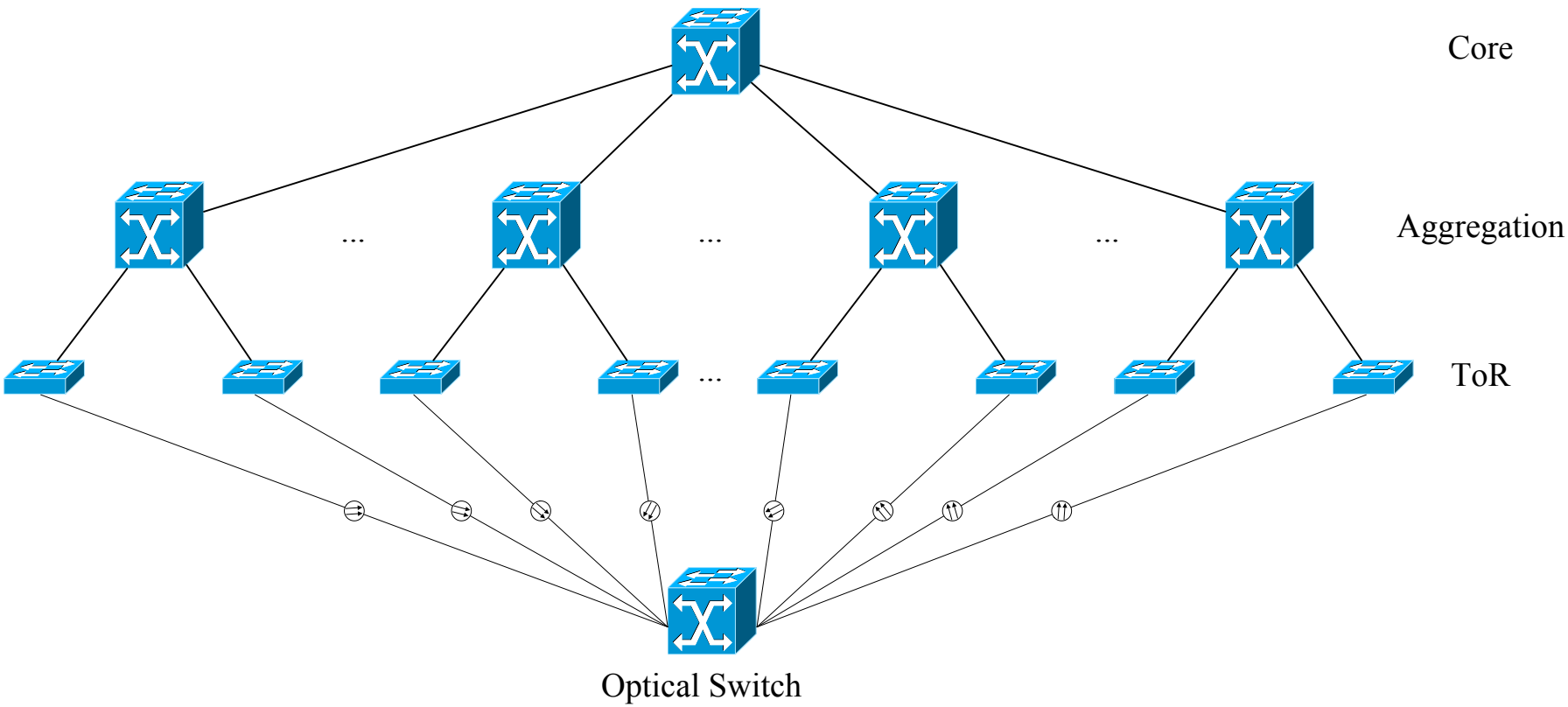




c-Through DCN

- Hybrid network architecture which makes use of both electrical packet switching network and optical circuit switching network.
- The hybrid network of c-Through consists of two parts:
  - a tree-based electrical network which maintains connectivity between each pair of ToR switches.
  - a reconfigurable optical network which offers high bandwidth interconnection between selected racks.
- It is unnecessary and not cost-effective to maintain an optical link between each pair of racks. Hence, the high capacity optical links are offered to pairs of racks transiently according to the traffic demand.
- Estimation of traffic between racks and reconfiguration of the optical network is accomplished by the system control plane.

Example of c-Through topology



Metrics

Degree The number of network ports on the servers in the data center.

Diameter The longest of the shortest paths between two servers in a data center.

Bandwidth The maximum capacity offered to servers under a given traffic pattern.

- "one-to-one": one arbitrary server sends data to another arbitrary server.
- "all-to-all": every server establishes a flow to all the other servers.

Bisection (band)width The minimum number of wires that must be removed when dividing the network into two equal sets of nodes.

Bisection width

- Idea:
  - Split N nodes into two groups of $N/2$ nodes such that the bandwidth between these two groups is minimum: that is the bisection width.
- Formally:
  - given a graph with N nodes, partition the nodes into two equally sized groups ($N/2$ nodes each) and find the minimum set of arcs $\mathcal{A}$ that need be removed to disconnect the two groups. Let $\nu(\mathcal{A})$ be the number of removed arcs (cardinality of $\mathcal{A}$). The minimum of $\nu(\mathcal{A})$ over all possible partitions is by definition the bisection width.

Bisection bandwidth

- The bisection bandwidth is obtained in case the arcs are weighted with the corresponding link capacity (bit/s), by substituting the number of removed links with the amount of capacity of those links.
- Why is it relevant? If traffic is completely random, the probability of a message going across the two halves is $1/2$. If all nodes send a message flow of rate R, the bisection bandwidth will have to be $R \cdot N/2$.

Topological performance comparison

	Basic Tree	Fat Tree	Clos net	DCell	BCube
Deg	1	1	1	$k+1$	$k+1$
D	6	6	6	$2^{k+1}-1$	$k+1$
SW	$\frac{(n^3-1)(n-1)^2}{n^3}$	$\frac{5n^2}{4}$	$\frac{6n+n^2}{4}$	N/n	$(k+1)n^k$
SE	$(n-1)^3$	$\frac{n^3}{4}$	$\frac{n^2}{4}n_{ToR}$	N (*)	n^{k+1}
W	$\frac{n[(n-1)^3-1]}{n-1}$	$\frac{3n^3}{4}$	$n^2+\frac{n^2}{4}n_{ToR}$	$(1+k/2)N$	$(k+1)n^{k+1}$

- Deg is the degree of servers, D is the network diameter, SW is the number of switches, W is the number of wires, SE is the number of servers.
- N is the number of servers.

(*) $\left(n+\frac{1}{2}\right)^{2^k}-\frac{1}{2} \leq N \leq (n+1)^{2^k}-1$

Number of servers

n	Tree-based Architecture			k	Recursive Architecture	
	Basic Tree	Fat Tree [3]	Clos Network [4]		DCell [5]	BCube [6]
4	64	16	8	2	420	64
				3	176,820	256
				4	$> 3 \times 10^{10}$	1,024
6	216	54	36	2	1,806	216
				3	3,263,442	1,296
				4	$> 10^{13}$	7,776
8	512	128	96	2	5,256	512
				3	27,630,792	4,096
				4	$> 7 \times 10^{14}$	32,768
16	4,096	1,024	896	2	74,256	4,096
				3	$> 5 \times 10^{9}$	65,536
48	110,592	27,648	26,496	2	5,534,256	110,592

Traffic performance comparison

	Basic Tree	Fat Tree	Clos net	DCell	BCube
One-to-one	1	1	1	$k+1$	$k+1$
One-to-many	1	1	1	$k+1$	$k+1$
One-to-all	1	1	1	$k+1$	$k+1$
All-to-all	n	N	$\frac{2N}{n_{ToR}}$	$> \frac{N}{2^k}$	$\frac{n(N-1)}{n-1}$
BW	$\frac{n}{2}$	$\frac{N}{2}$	$\frac{N}{n_{ToR}}$	$> \frac{N}{4 \log_n N}$	$\frac{N}{2}$

- BW is the bisection width.
- N is the number of servers.

Hardware redundancy

Node-disjoint Paths The minimum of the total number of paths that share no common intermediate nodes between any arbitrary servers.

Edge-disjoint Paths The minimum of the total number of paths that share no common edges between any arbitrary servers.

f-fault tolerance A network is f-fault tolerant if for any f failed components in the network, the network is still connected.

Redundancy Level A network has a redundancy level equal to r if and only if, after removing any set of r links, it remains connected, and there exists a set of $r + 1$ links such that after removing them, the network is no longer connected.

Hardware redundancy comparison

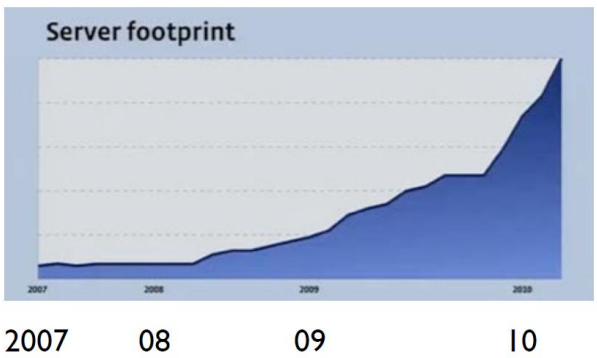
			Tree-based Architecture			Recursive Architecture	
			Basic Tree	Fat Tree	Clos	DCell	BCube
Node-disjoint Paths			1	1	1	$k+1$	$k+1$
Edge-disjoint Paths			1	1	1	$k+1$	$k+1$
Redundancy Level	Switch	Edge/ToR	0	0	0	k	k
		Aggregation	0	$\frac{n}{2}-1$	1		
		Core/Intermediate	0	$\frac{n^2}{4}-1$	$\frac{n}{2}-1$		
	Server		-	-	-	k	k

LEGUP

- Expansion of a data center based on a regular topology (e.g., fat tree) is highly constrained.
- LEGUP attacks the expansion problem by trying to find optimal upgrades for Clos networks.
  - The Authors develop the theory of heterogeneous Clos networks.
  - They show that their construction needs only as much link capacity as the classic Clos network to route the same traffic matrices.
  - They provide details to design the topology and physical arrangement of network upgrades or expansions.
- Still the rigid structure of the Clos network poses a strong limit to expandability.

Jellyfish design goals

- High throughput
  - Remove bottlenecks.
  - Agile placement of VMs.
- Incremental expansion
  - Easily add/replace servers and switches.



Commercial products useful to add servers:

- SGI Ice Cube (Expandable Modular Data Center).
- HP EcoPod (Pay-as-you-grow).

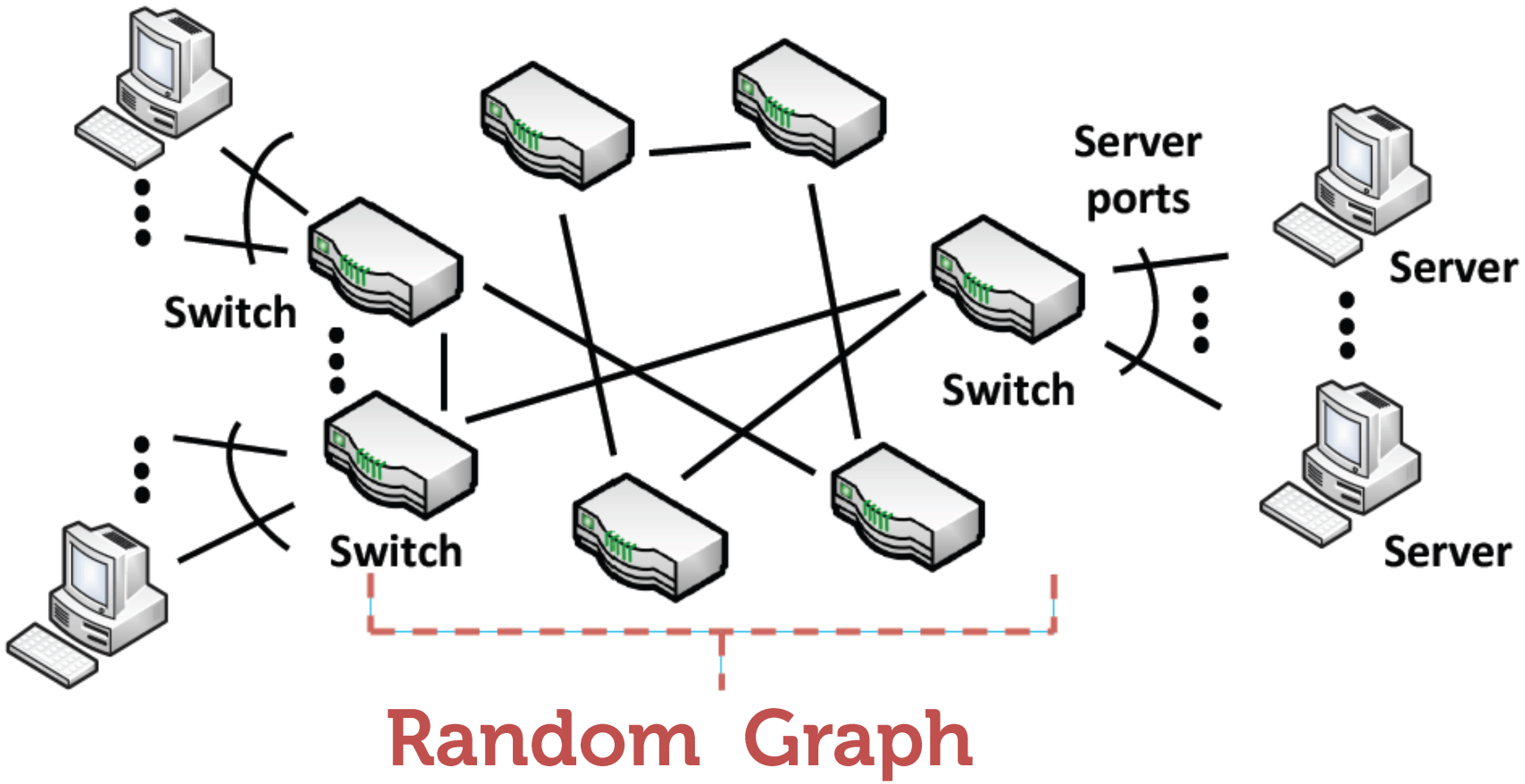
Structured topologies limits

- Coarse design constraints
  - Hypercube: $(k+1)n^k$ switches.
  - Three-layer fat tree: $5n^2/4$ switches.
- Number of servers with commodity switches in three-layer fat tree networks
  - 24-port switches: 3456 servers
  - 32-port switches: 8192 servers
  - 48-port switches: 27648 servers
  - 64-port switches: 65536 servers

Jellyfish approach:

forget about structure $\rightarrow$ random graphs.

Jellyfish network



Jellyfish construction

- The Jellyfish approach is to construct a random graph at the top-of-rack (ToR) switch layer.
- ToR switch i has n_i ports:
  - r_i are connected to other switches;
  - $n_i - r_i$ are connected to servers.
- If the number of switches is S, then the number of servers can be up to $S(n - r)$, by assuming n_i and r_i equal for all i.

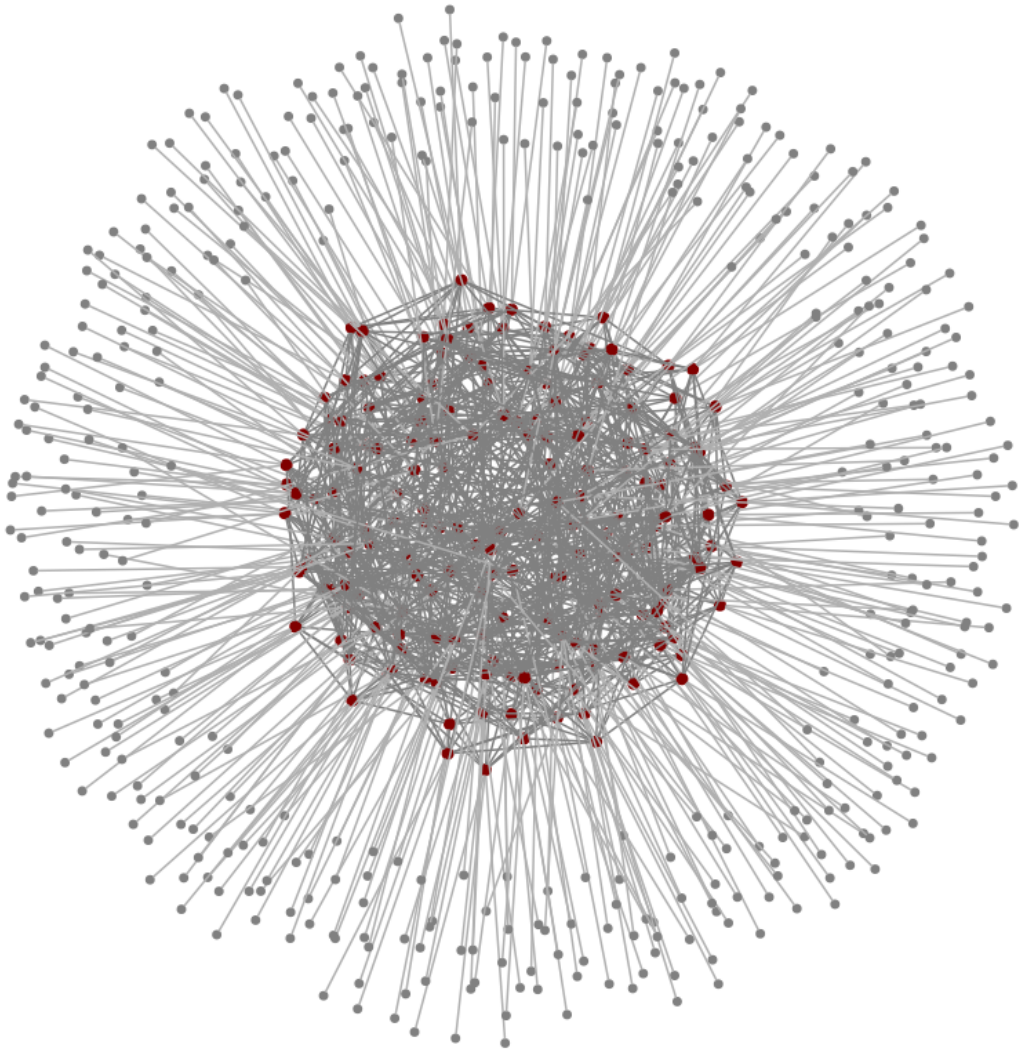
Construction procedure:

1. Pick a random pair of switches with free ports and not already connected, join them with a link, and repeat until no further links can be added.
2. If a switch has ≥ 2 free ports left, say (p_1, p_2), choose a random link (x, y), remove it and replace it with (p_1, x) and (p_2, y).

Incremental expandability

- Jellyfish's construction makes it amenable to incremental expansion by adding either servers and/or switches.
- Assume we add one rack of servers, with its ToR switch u, to the existing network.
  - Pick a random link $\ell = (v, w)$ such that this new ToR switch is not already connected with either v or w.
  - Remove ℓ.
  - Add the two links (u, v) and (u, w), thus using 2 ports on u.
  - This process is repeated until all ports are filled or a single odd port remains.
- The outcome is still a regular random graph of degree r as the initial graph.

Jellyfish topology



Why should a random graph work?

$$\text{Number of flows at full throughput (1 Gbps)} \leq \frac{\sum_{\text{all links}} \text{Capacity(link)}}{(1\text{ Gbps}) \cdot (\text{mean path length})}$$

Intuition: make paths as short as possible.

Minimize average path length

Proof of throughput equation

- C_j: the capacity of link j;
- x_i: the throughput of flow i;
- h_i: the number of links of flow i routing path;
- r_{ji}: routing variable, defined by:

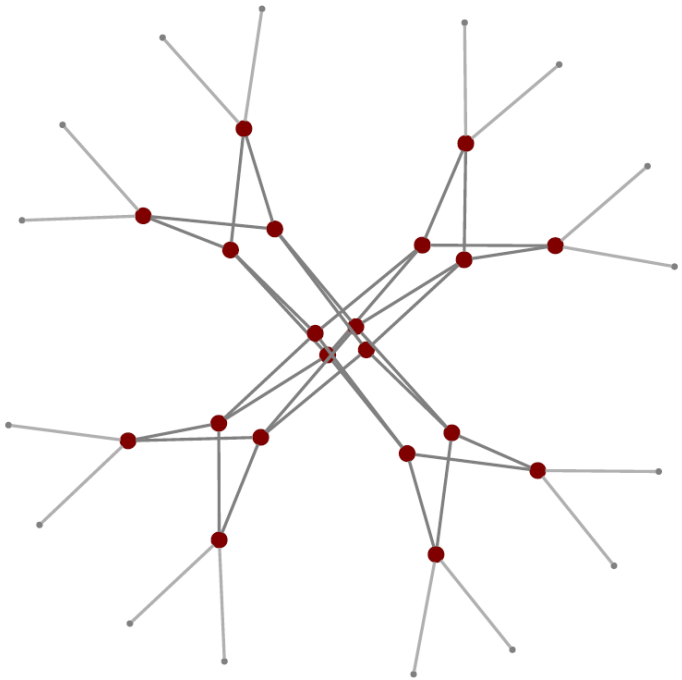
$$r_{ji} = \begin{cases} 1 & \text{if link } j \text{ belongs to the routing path of flow } i \\ 0 & \text{otherwise} \end{cases}$$

By assuming all flows get the same (maximum) throughput x:

$$\sum_{i \in \mathcal{F}} r_{ji} x_i \leq C_j \quad \Rightarrow \quad x \sum_{j \in \mathcal{L}} \sum_{i \in \mathcal{F}} r_{ji} = x \sum_{i \in \mathcal{F}} h_i \leq \sum_{j \in \mathcal{L}} C_j$$

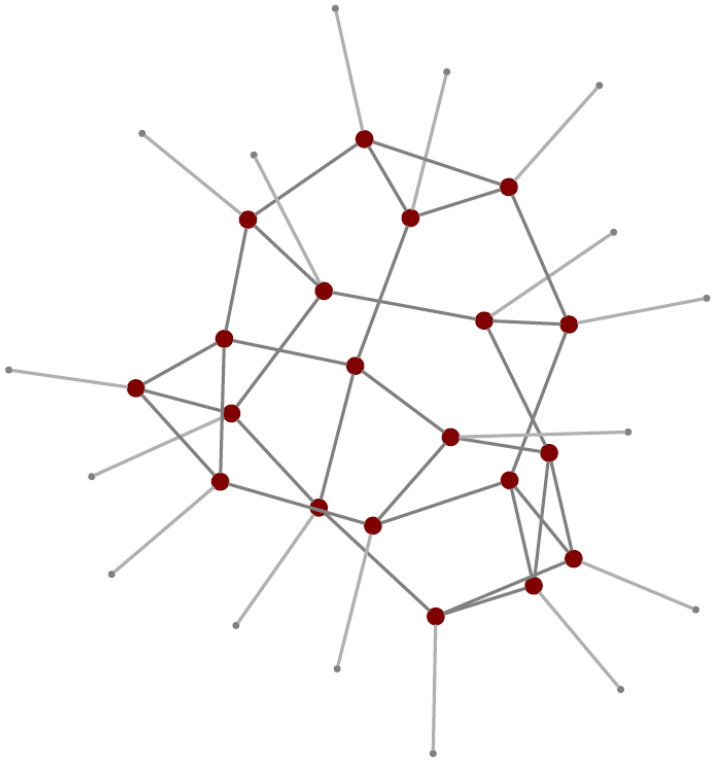
Then $|\mathcal{F}| \leq \frac{\sum_{j \in \mathcal{L}} C_j}{x \frac{1}{|\mathcal{F}|} \sum_{i \in \mathcal{F}} h_i} = \frac{\sum_{j \in \mathcal{L}} C_j}{x \bar{h}}$

An example of path length reduction (1/3)



Fat tree

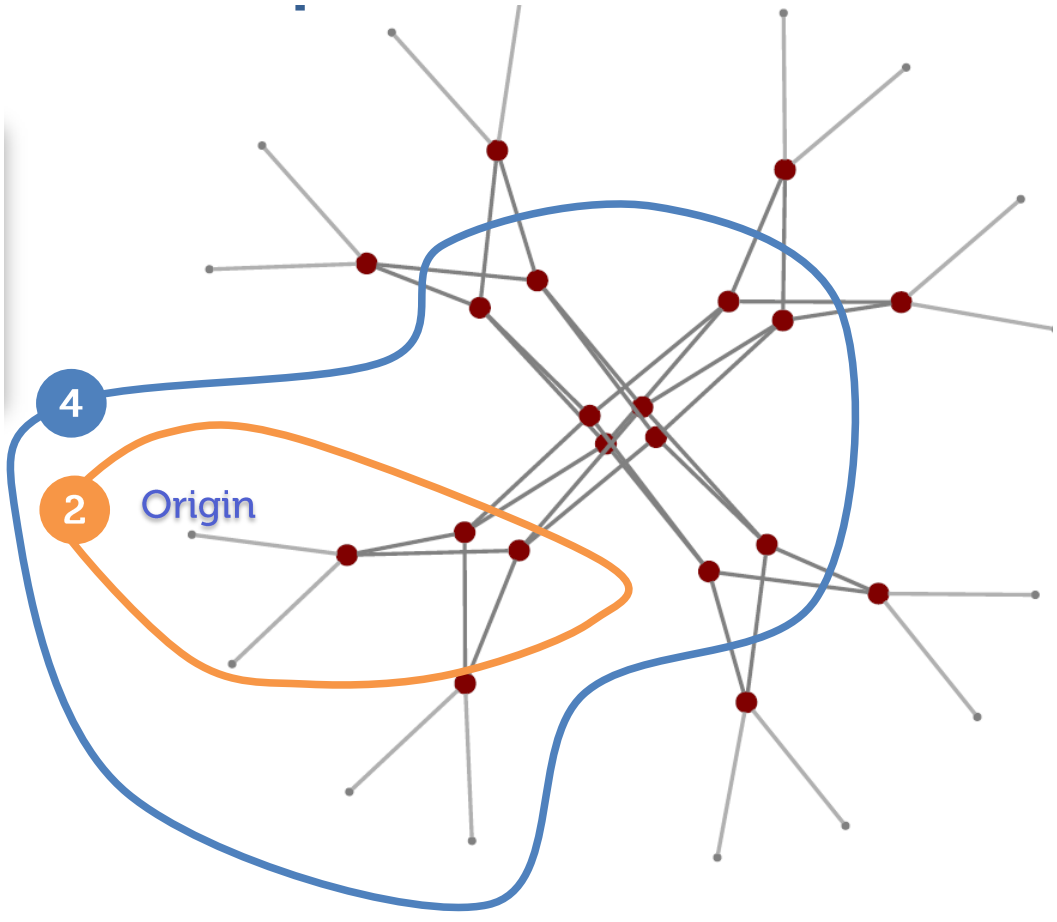
16 servers, 20 switches, degree 4



Jellyfish random graph

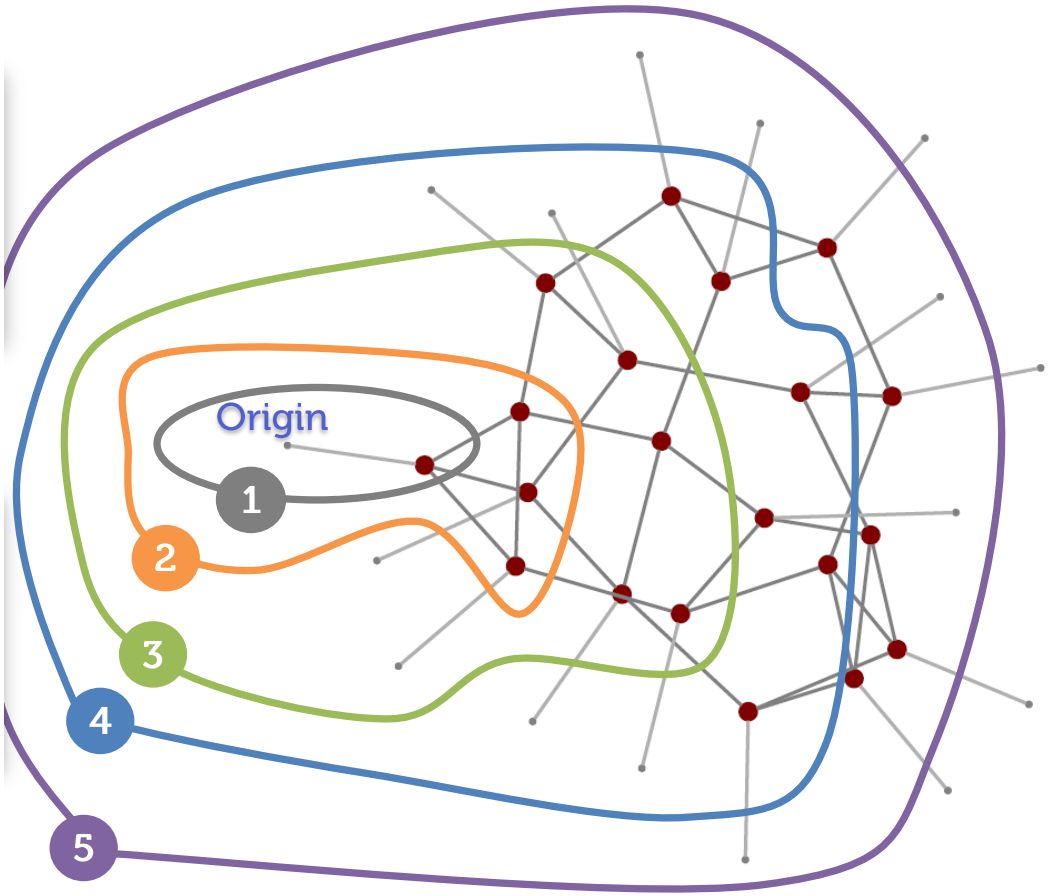
16 servers, 20 switches, degree 4

An example of path length reduction (2/3)



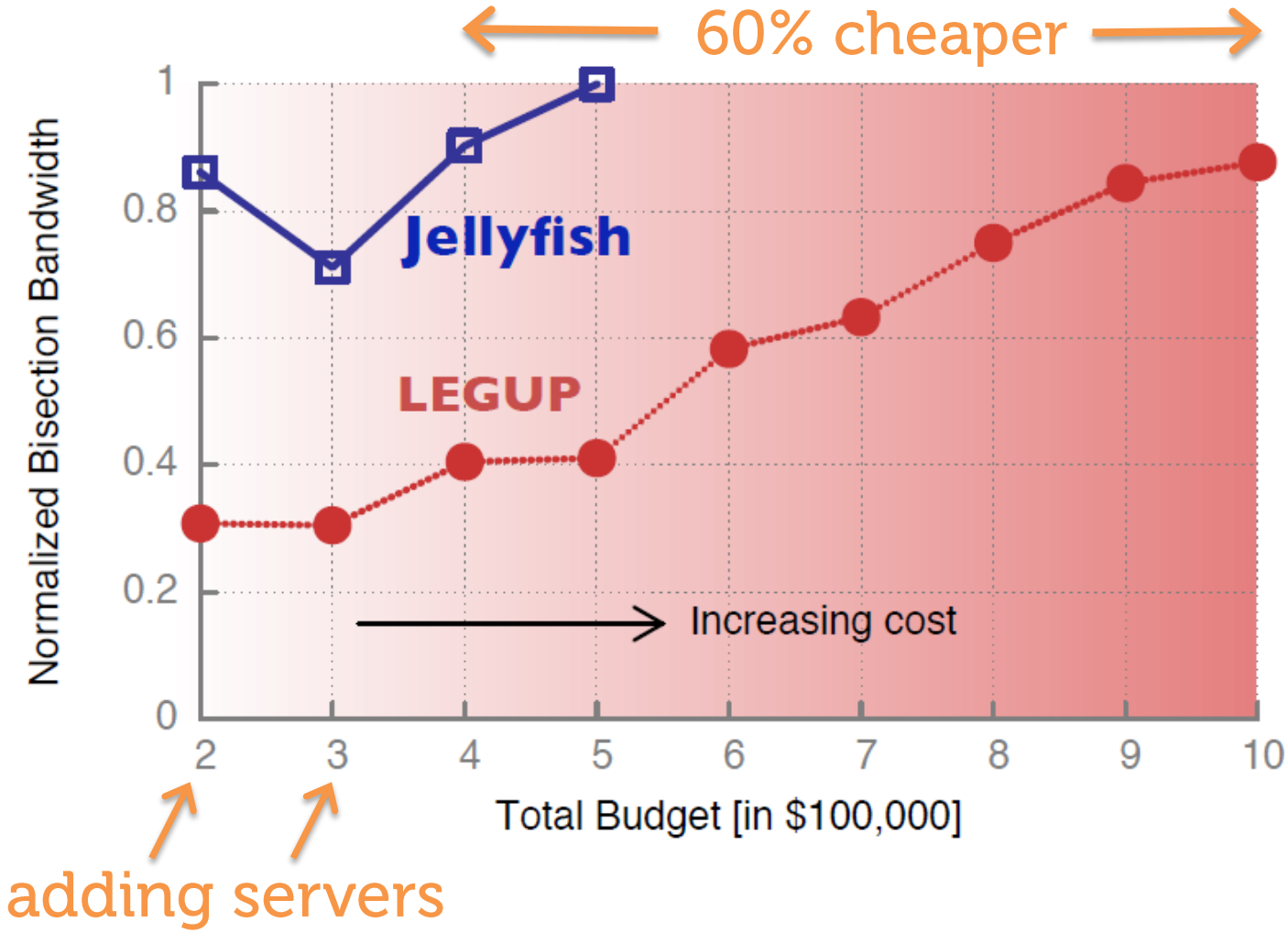
3 of 15 in less than 6 hops.

An example of path length reduction (3/3)

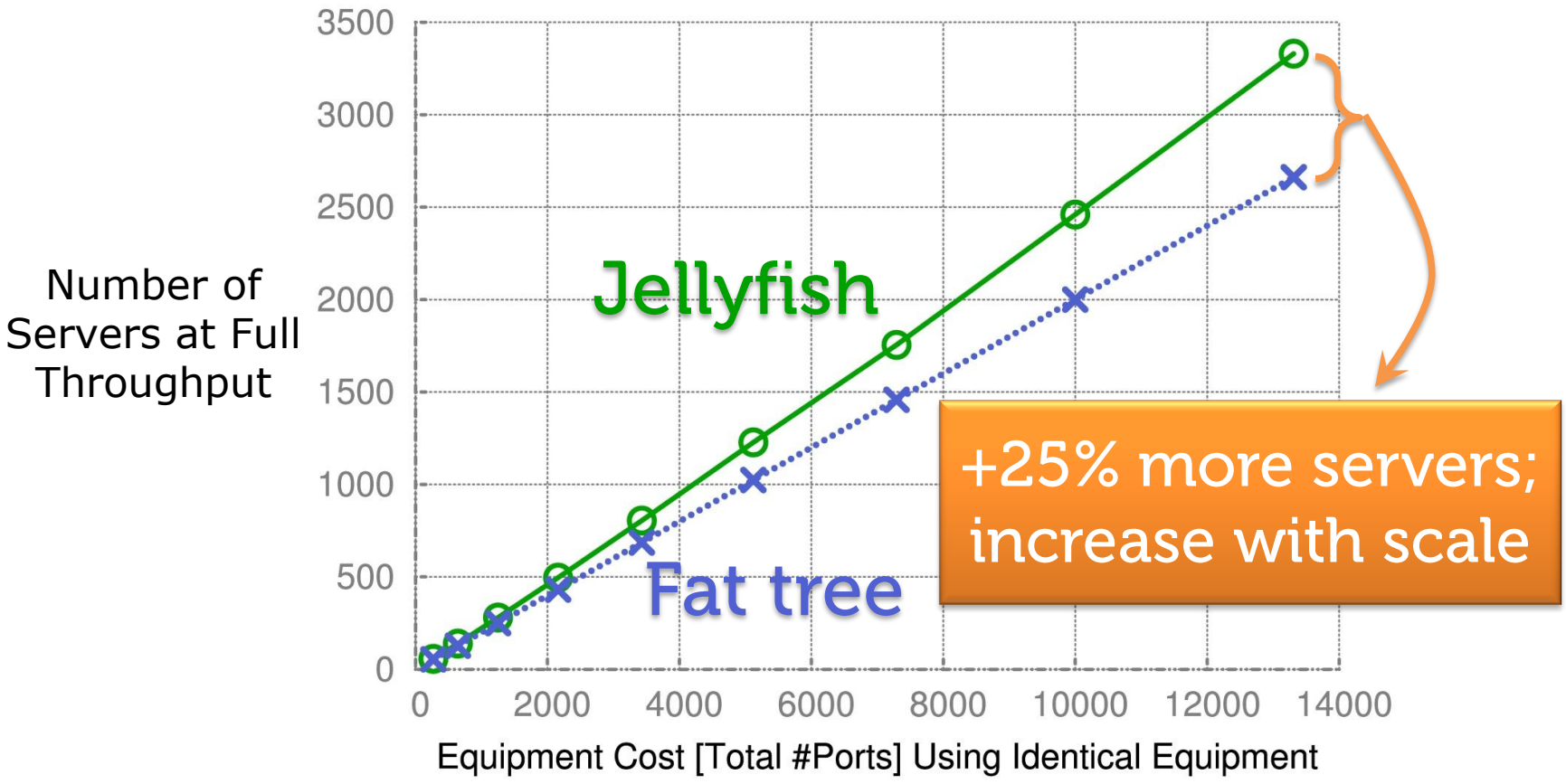


11 of 15 in less than 6 hops.

Expandability

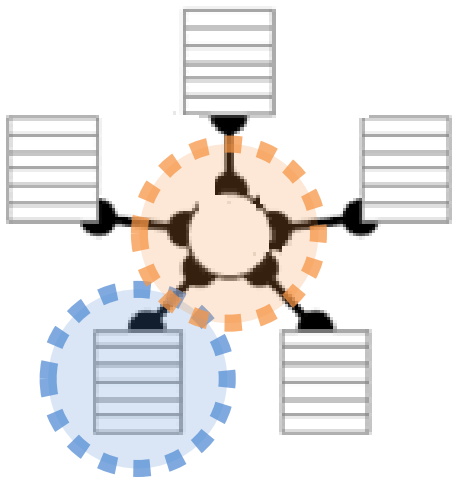


Number of servers



Cabling

- Server racks and Switches are clustered.
- Switches are placed centrally.
- Cable bundles can be aggregated.



cluster

Challenge #1: topology design

Definitions

- A graph is a couple $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is a finite, nonempty set of nodes and $\mathcal{A}$ is a subset of $\mathcal{N} \times \mathcal{N}$.
  - A very useful model for representing network connectivity as seen at a given architectural level.
  - The meaning of nodes and arcs depends on the considered architectural level, e.g., nodes can be routers and arcs subnets connecting them at IP level or switches connected with physical links at layer 2.
  - We will use equivalently the terms arc, edge or link.
- A graph is said to be *undirected* if $(i, j) \in \mathcal{A} \Rightarrow (j, i) \in \mathcal{A}$.
- We define:

$$n = \text{number of nodes;}$$
$$k = \text{number of arcs.}$$

Adjacency matrix

- The $n \times n$ matrix $\mathbf{A}$ whose entries a_{ij} are defined as:

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \mathcal{A} \\ 0 & \text{otherwise} \end{cases}$$

  is called the adjacency matrix of the graph.
- The adjacency matrix of an undirected graph is symmetric.
- The sum of rows (columns) of $\mathbf{A}$ is the out-(in-)degree of the graph nodes, i.e., the number of arcs outgoing from (incoming into) a graph node.

Connectivity

- A path between nodes s and t is an ordered sequence of nodes and arcs connecting the two nodes, i.e., $(s, k_1), (k_1, k_2), \ldots, (k_{\ell-1}, t)$ is a path of length ℓ.
- A graph is said to be connected if there exists a path connecting any two nodes.

Criteria to check graph connectivity for undirected graphs.

- Starting from an arbitrary node, all graph nodes can be visited (BSF algorithm).
- The adjacency matrix $\mathbf{A}$ is irreducible.
- The second smallest eigenvalue of the graph Laplacian matrix $\mathbf{L}$ is positive.

Irreducibility

- Say nodes of the graph are labeled with positive integers $1, \ldots, n$.
- The adjacency matrix is reducible if and only if there exists a re-labeling of nodes such that the matrix can be partitioned as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A_{11}} & \mathbf{0} \\ \mathbf{A_{21}} & \mathbf{A_{22}} \end{bmatrix}$$

  where $\mathbf{A_{11}}$ and $\mathbf{A_{22}}$ are square blocks and $\mathbf{0}$ is a block of zeros.
- It can be shown that an $n \times n$ matrix $\mathbf{A}$ is irreducible if

$$\mathbf{I} + \mathbf{A} + \mathbf{A}^2 + \cdots + \mathbf{A}^{n-1} > \mathbf{0}$$

Breadth-First-Search algorithm

Finds all paths form a given node.

Given a graph 'Graph' and a node 'root' in it:

```
01 for each node n in Graph:
02    n.distance = INFINITY
03    n.parent = NIL
04 endfor
05 create empty queue Q
06 root.distance = 0
07 Q.enqueue(root)
08 while Q is not empty:
09    current = Q.dequeue()
10    for each node n that is adjacent to current:
11    if n.distance == INFINITY:
12     n.distance = current.distance + 1
13     n.parent = current
14     Q.enqueue(n)
15    endif
16 endwhile
```

Laplacian

- The laplacian of a graph in an $n \times n$ matrix $\mathbf{L}$ defined as follows:
  - $L_{ii} = d_i$, where d_i is the degree of node i;
  - $L_{ij} = -1$ if and only if $(i, j) \in \mathcal{A}$.
- We can write:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

- $\mathbf{L}$ is a symmetric matrix, if the graph is undirected.
- The sum of each row and each column of $\mathbf{L}$ is 0. Therefore 0 is always an eigenvalue of $\mathbf{L}$

Eigenvalues of the Laplacian

- The incidence matrix of a graph in an $k \times n$ matrix $\mathbf{M}$ defined as follows:
  - $M_{\ell i} = 1$, if $\ell = (i, j) \in \mathcal{A}$ (outgoing arc);
  - $M_{\ell j} = -1$, if $\ell = (i, j) \in \mathcal{A}$ (incoming arc);
  - $M_{\ell j} = 0$, otherwise.
- It can be verified that:

$$\mathbf{L} = \frac{1}{2}\mathbf{M}^T\mathbf{M}$$

- Hence $\mathbf{L}$ is a positive semi-definite matrix, i.e., $\mathbf{x}^T\mathbf{L}\mathbf{x} \geq 0$ for any vector $\mathbf{x}$.
- Positive semi-definite and symmetric $\rightarrow$ the eigenvalues of $\mathbf{L}$ are real and non-negative.

Connectivity through Laplacian

Theorem (From algebraic graph theory)

For an undirected graph the number of connected components is equal to the algebraic multiplicity of the smallest eigenvalue of the graph Laplacian.

Corollary. An undirected graph is connected if and only if the smallest eigenvalue of the graph Laplacian is simple, i.e., the second smallest eigenvalue of the Laplacian is positive.

- The property stated in the Corollary can be used to assess the connectivity of a given graph.
- Let $\eta_1 = 0 \leq \eta_2 \leq \cdots \leq \eta_n$ denote the n real, non-negative eigenvalues of $\mathbf{L}$.
- The graph is connected if and only if $\eta_2 > 0$.

Random graphs

- **Erdős-Rényi random graphs**: two models are possible.
  - $G(n, m)$ model: the number m of arcs is given; they are assigned uniformly at random to m out of the $n(n-1)/2$ node couples.
  - $G(n, p)$ model: a graph is constructed by connecting nodes randomly. An arc is included in the graph with probability p independently of any other arc.
- **r-regular random graph**: a graph selected from $\mathcal{G}_{n,r}$, which denotes the probability space of all r-regular graphs on n vertices, where $3 \leq r < n$ and nr is even.
  - An r regular graph is a graph where each vertex has the same number r of neighbors.
  - r regular graphs for $r = 0, 1, 2$ are trivial.

Assignment - Part 1

1. Use library scripts to generate p-ER random graphs and r-regular random graph. Let K denote the number of nodes.
2. Write a script to check the connectivity of a given graph.
   - algebraic method 1 (irreducibility);
   - algebraic method 2 (eigenvalue of the Laplacian matrix);
   - breadth-first search algorithm.
3. Compare the complexity as a function of K of the methods above by plotting curves of a complexity measure vs K.
4. Let $p_c(\mathcal{G})$ denote the probability that a graph $\mathcal{G}$ is connected. By running Monte Carlo simulations, estimate $p_c(\mathcal{G})$ and produce two curve plots:
   - $p_c(\mathcal{G})$ vs. p for Erdős-Rényi graphs with $K = 100$.
   - $p_c(\mathcal{G})$ vs. K, for $K \leq 100$, for r-regular random graphs with $r = 2$ and $r = 8$.

Definitions

We aim to compare throughput offered by Fat-Tree and Jellyfish for the same amount of resources.

- $N = \#$ of servers.
- $S = \#$ of switches.
- $L = \#$ of links connecting switches.
- $n = \#$ number of ports of a switch.
- $\overline{h} =$ mean shortest path lengths for server-to-server paths.

Assignment - Part 2

1. Find r (# of switch ports to be connected to other switches in Jellyfish) as a function of n so that N, S and L are the same for Jellyfish and Fat-tree.
2. Write the expression of the application-oblivious throughput bound TH for an all-to-all traffic matrix among servers. The expression of TH must be a function of $\overline{h}$ and n only.
3. Using the exact value of $\overline{h}$ as a function of n for Fat-Tree (you must calculate it!) and the lower bound of $\overline{h}$ for r-regular random graphs (see slides, but be careful with notation) for Jellyfish, evaluate TH for $n = 20, 30, 40, 50, 60$.

Delivery of the assignment

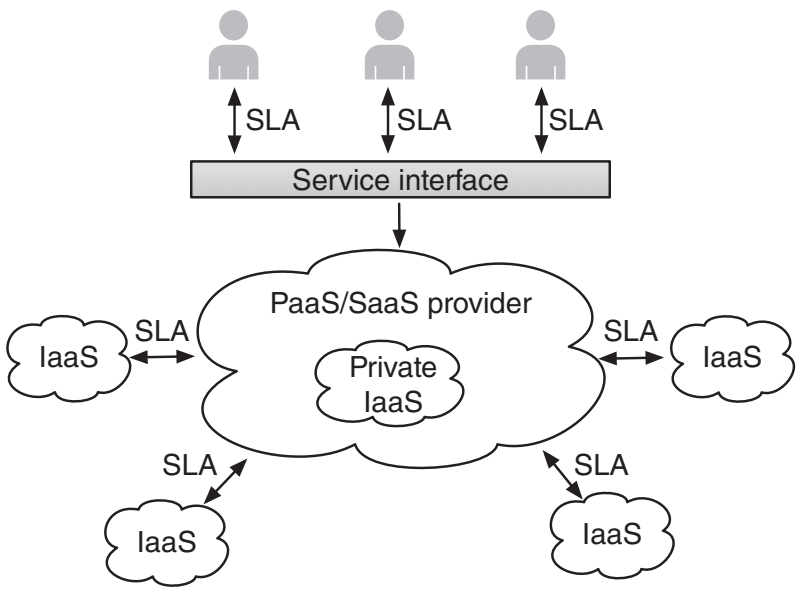
The delivery of the assignment consists of a **2 pages written report** (Font size: 12 pt). Remember to put your given and family names, and enrollment number as a header on top of each page.

1. **PAGE 1** - Three curve plots: (i) Curve plot of complexity versus number of nodes K, for the three connectivity checking algorithms; (ii) Probability of a connected ER random graph as a function of p for $K = 100$ nodes; (iii) Probability of a connected r-regular random graph as a function of the number of nodes K for $r = 2$ and 8. Accuracy and quality of the graphs will be fundamental in the evaluation. Also concise and significant comments will be appreciated.
2. **PAGE 2** - (i) Derivation of expression of r as a function of n; (ii) Expression of TH as a function of n and $\overline{h}$; (iii) Table with six columns of numerical values: n, N, S, L, $TH_{\text{Fat-Tree}}$, $TH_{\text{Jellyfish}}$ computed for $n = 20,\ 30,\ 40,\ 50,\ 60$.

Resource management and scheduling

Relationship among cloud providers

- Service-Level Agreements (SLAs) define relationship between customer and provider.
- A cloud provider can be a client of another provider offering different types of service.
  - In this case, more than one level of SLAs are necessary.



Management problems

- Arrangements between providers on more than two service levels are possible.
- Management problems involving different levels of SLAs must be dealt with, including the dependencies of the upper layers on the services of lower layers.
- In order to offer a service level guarantee to the user, the cloud provider must receive guarantees from its lower level providers.
- The cloud provider must consider its margin of profit when pricing its services, which depends on the agreement it has with other providers.

Management functions

- The offering of cloud services is based on SLAs on a pay-per-use basis.
- Management systems vary according to SLAs:
  - In **SaaS** and **PaaS**, management should be able to automatically increase or decrease **computing power** for a user according to his/her demands.
  - In **IaaS** management must act according to client demands for different VM types, allocating these requests to **physical machines** on the basis of the policies defined by the provider.

Charging

- Actual VM allocation may depend on the model of charging selected by the client.
  - **On-demand VM leasing**. Price per unit time per VM.
  - **Reserved model**. Predefined price for access to VMs on demand.
  - **Spot model**. Works as a market, with VM prices varying with demand.
    - The user offers a price he/she is willing to pay for a type of VM, and it can be used as long as the actual price remains lower than the initial offer. If demands on the provider increase, the price can increase, and the provider has the right to deny access to VMs which, at least momentarily, cost more on the market.

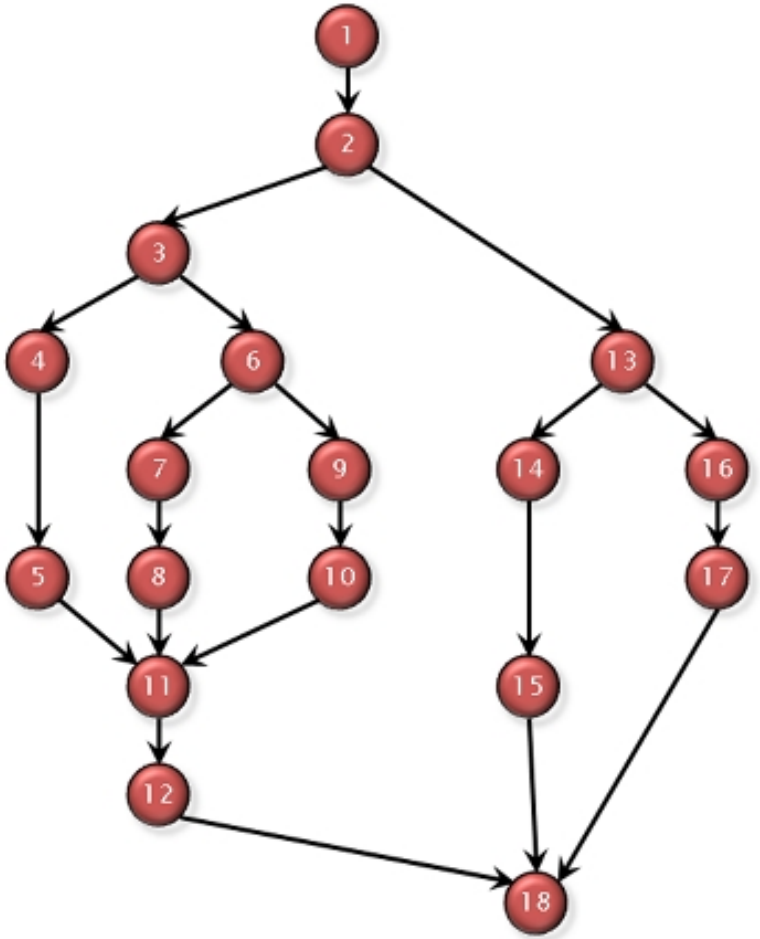
Services vs jobs

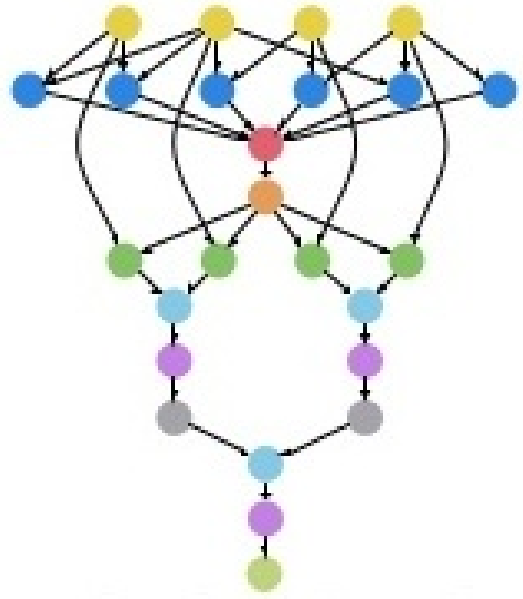
- Service-oriented computing relies on services instead of job dispatching.
- Service invocation assumes an already deployed code with an interface to be called, and which will remain running after results are delivered
  - **Service repository**: to control where each service is already running
  - **Service deployment**: to transfer and deploy services across the resources
- Job dispatching involves code that is to be transferred, run, and finished with the results delivered to the user or to another application.
- Task is a generic term for both instances.

Workflows and DAGs

- Tasks and data transfer required to run them can depend on one another and hence precedence-constrained $\rightarrow$ tasks form a **workflow**.
- The ordering of workflows with dependent tasks can be represented by a directed acyclic graph (DAG) $\mathcal{G} = (V, E)$.
  - $t_a \in V$ is a task with an associated computation cost (weight) w_{t_a}.
  - $e_{a,b} \in E$ denotes that task t_b is dependent on the completion of task t_a, with an associated communication cost of $c_{a,b}$.
- DAG-related variations include applications where the DAG can change itself during execution, due to the presence of conditional tasks in the DAG specifications.

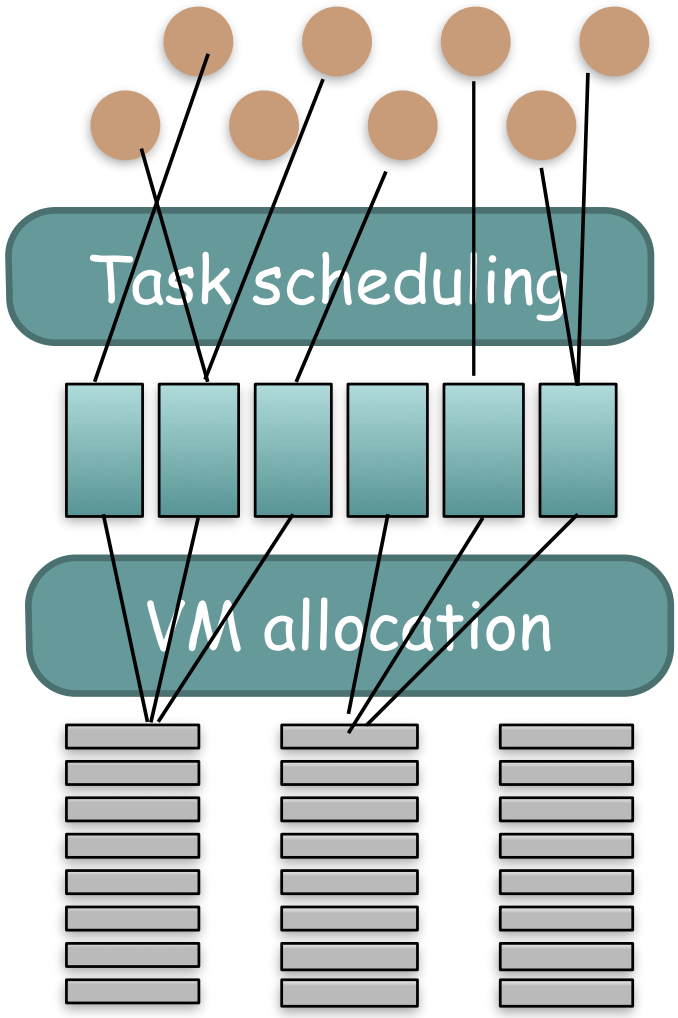
Examples of workflow DAGs





Montage is a portable toolkit for constructing custom, science-grade mosaics by composing multiple astronomical images. The mosaics constructed by Montage preserve the astrometry (position) and photometry (intensity) of the sources in the input images.

Overall resource optimization

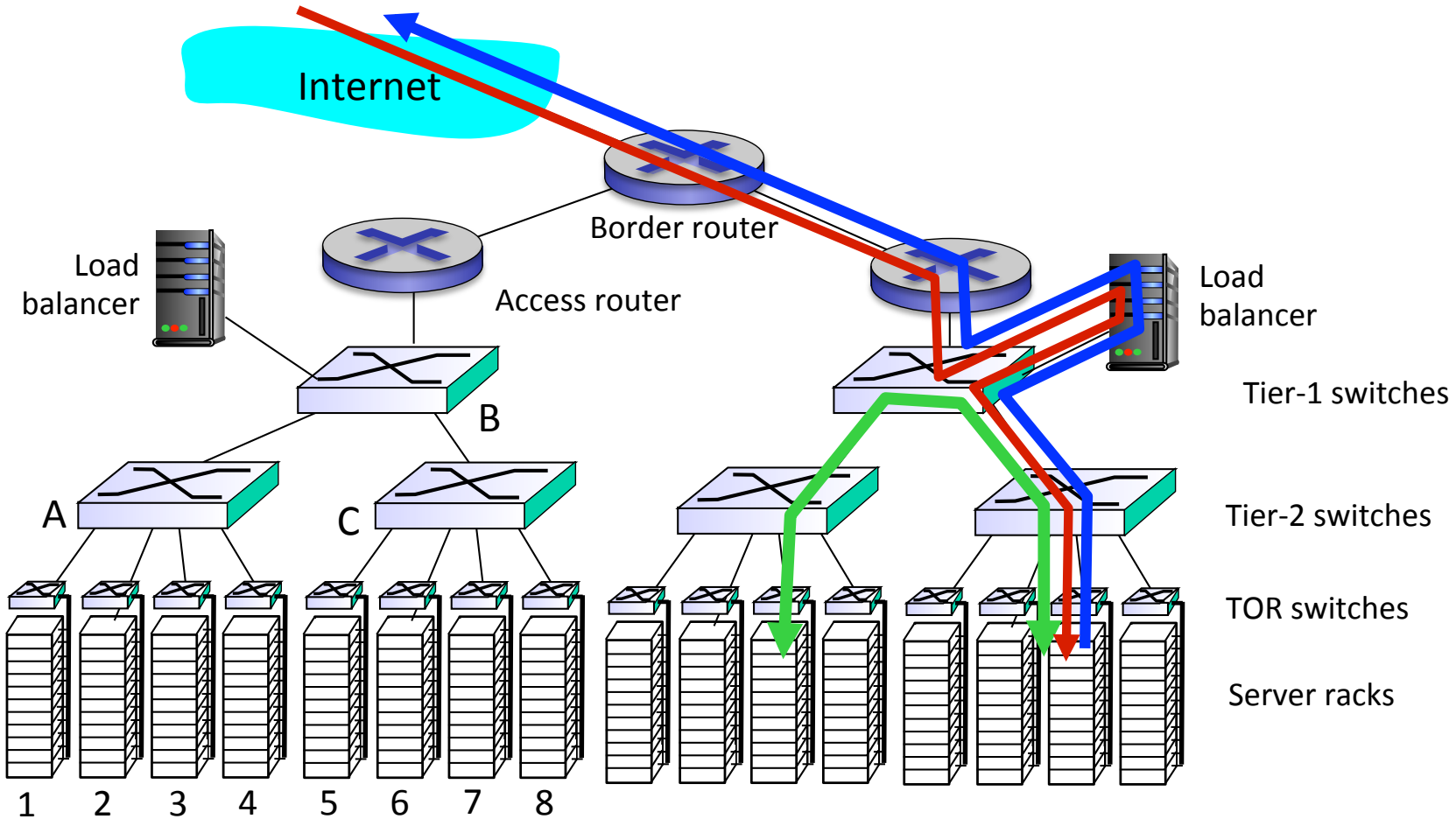


Application tasks

Virtual Machines

Physical machines
(servers, DC network)

Where is the dispatcher in a Data Center?



Importance of delay – E-commerce

- Reducing latency to support interactive services is crucial.
  - Time To Interact (TTI): the time it takes for the primary page content to render and become interactive (item can be clicked).
- Very high economic impact
  - Estimated to hit 6.5 billion $ by 2023[2]
  - Amazon believes that every 100 ms of latency costs 1% in sales.
  - Walmart reduced TTI from 7.2 s to 2.9 s in 6 months (`www.radware.com`).

[2] (`http://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/`).

Importance of delay – Web searching

Google: 10 to 30 results displayed, delay grows from 400 ms to 900 ms, traffic drops by 20% [3]

- Users want low latency even more than additional results!
- Google is obsessed by web latency — target is around 100 ms.
- Fast websites rewarded in Google's ranking algorithms [4]

[3] http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html

[4] http://googlewebmastercentral.blogspot.com/2010/04/using-site-speed-in-web-search-ranking.html

Importance of delay – More applications

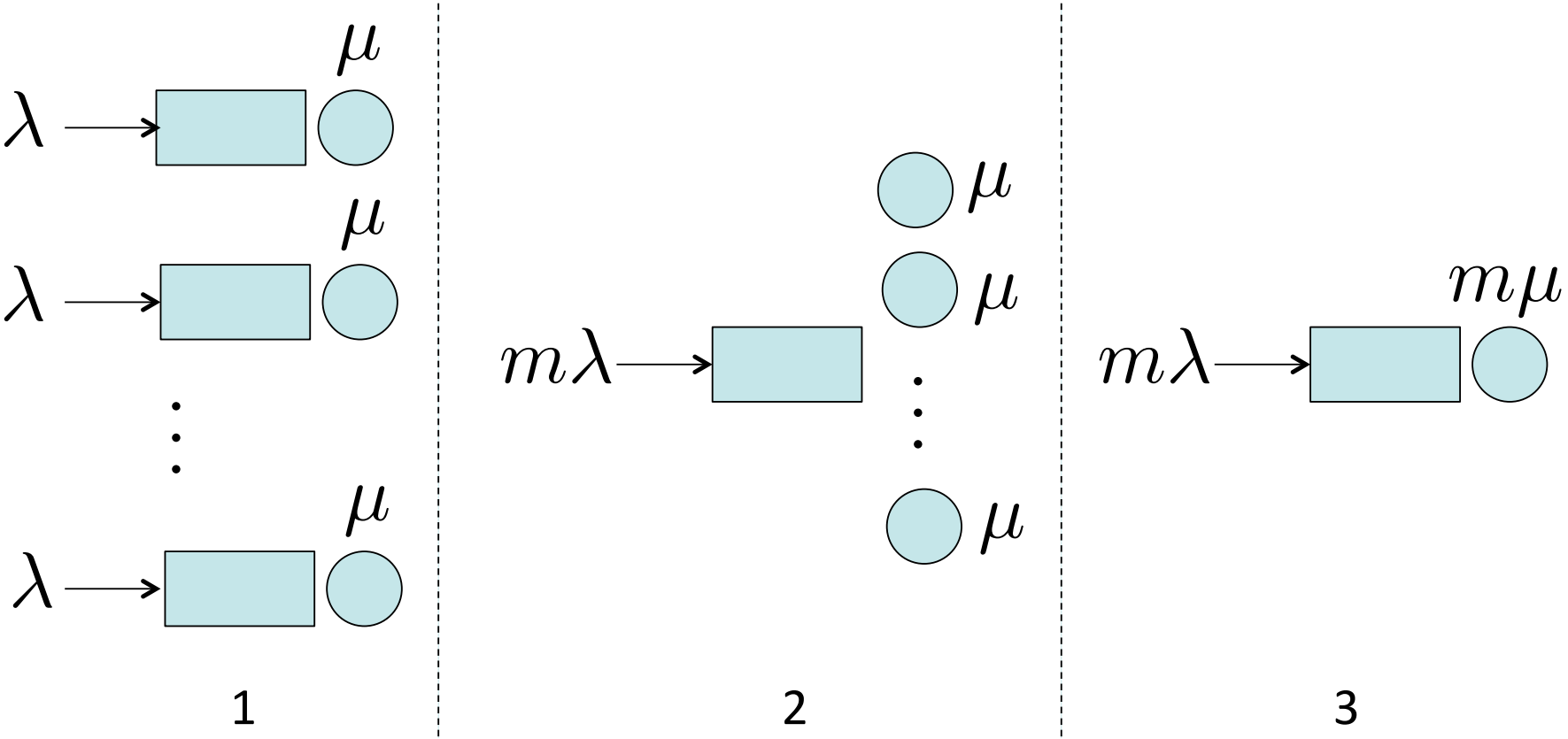
Other areas where delay has a high impact

- E-health applications.
- Industrial processes control.
- Safety applications of the Intelligent Transportation System paradigm.
- Network Intrusion Detection Systems applied to traffic flows in real time.
- On-line gaming.

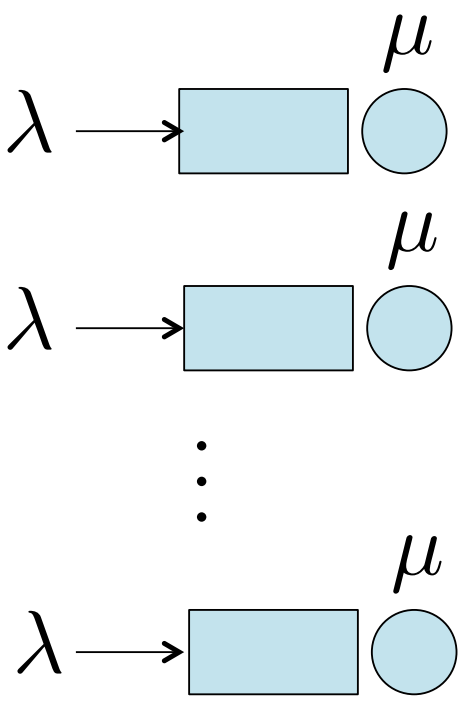
Roadmap

1. Single vs multi server systems: should we pool resources?
2. Head-Of-Line (HOL) priorities.
3. Processor sharing: theory and practice.
4. Revisiting server pooling: a case for multiple servers.
5. Load Balancing: taxonomy, optimality and state-of-the-art.
6. An adaptive policy for (asymptotically) optimal load balancing.

Resource separation vs pooling



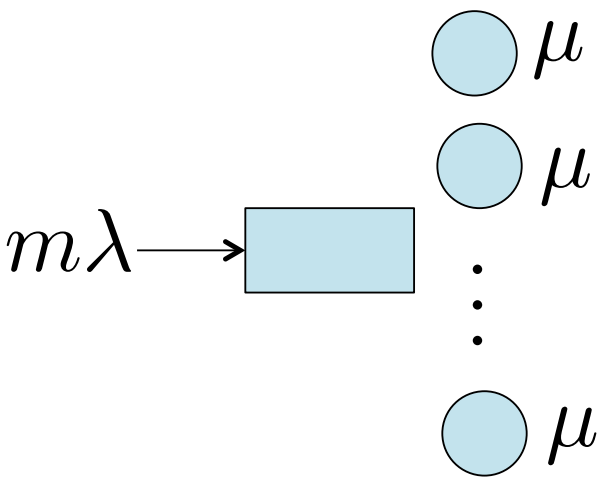
Multi-queue: mean delay



Multiple-server, multiple waiting line:

$$\mathrm{E}[S] = \frac{1}{\mu - \lambda}$$

Multi-server: mean delay



Multiple-server, single waiting line:

$$\mathrm{E}[S] = \frac{P_d}{m\mu - m\lambda} + \frac{1}{\mu}$$

$$P_d = C(m, m\lambda/\mu)$$

[see next slide for the definition of $C(\cdot, \cdot)$]

Erlang functions

C function

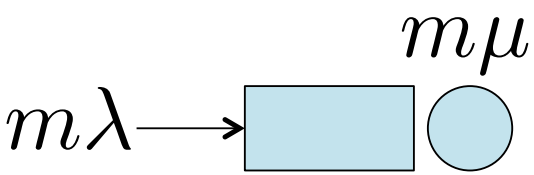
$$C(m, A) = \frac{B(m, A)}{1 - \frac{A}{m} + \frac{A}{m} B(m, A)}$$

B function

$$B(m, A) = \frac{\frac{A^m}{m!}}{\sum_{k=0}^{m} \frac{A^k}{k!}} = \begin{cases} 1 & m = 0, \\ \frac{B(m-1,A)}{m+A\ B(m-1,A)} & m \geq 1 \end{cases}$$

Try implementing these iterative formulas to see how fast you can compute them!

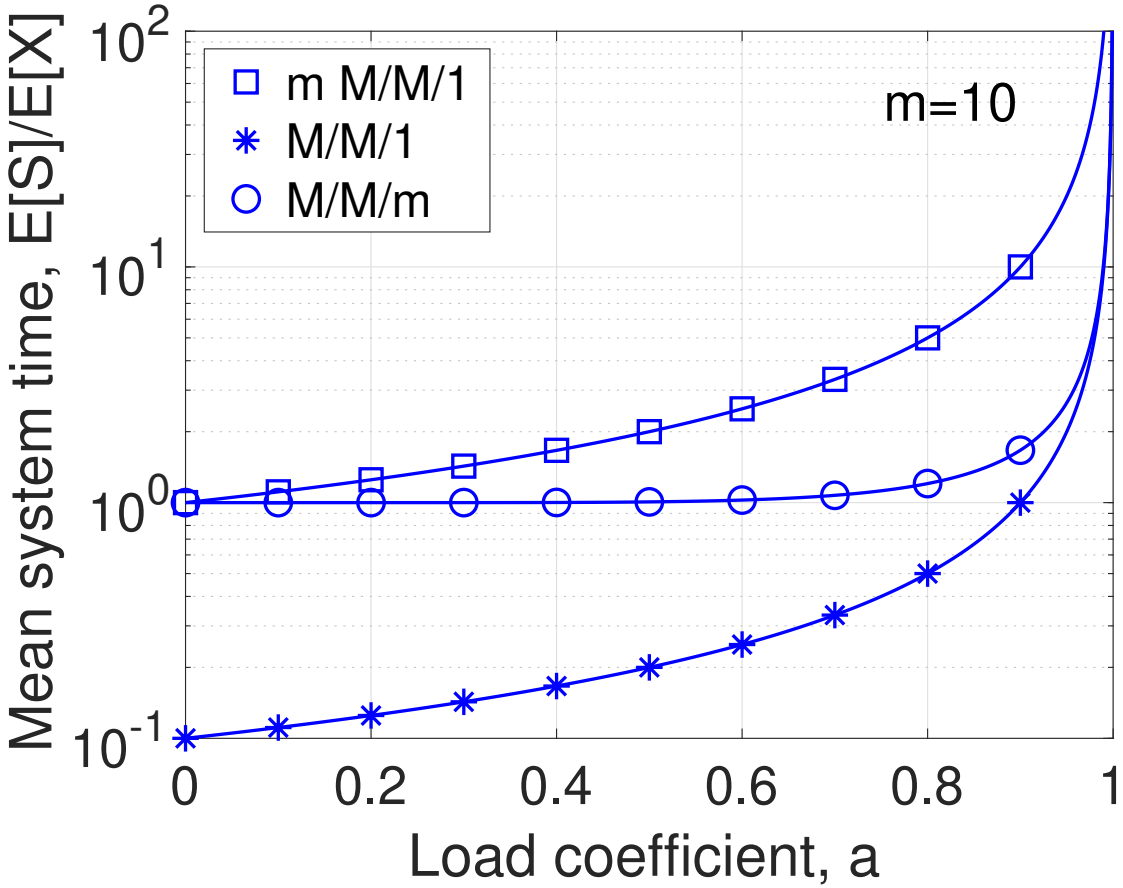
Single-server: mean delay



Single-server, single waiting line:

$$\mathrm{E}[S] = \frac{1}{m\mu - m\lambda}$$

Numerical example



$$a = \frac{\lambda}{\mu}$$

Notation

Let us consider a single server system fed by P classes of customers.

- Class j has priority over class i for $j < i$.
- T_j is the inter-arrival time random variable of class j. We assume Poisson arrivals, unless otherwise stated.
  - λ_j mean arrival rate of class j.
- X_j service time of class j. We assume a general renewal process for service times with CDF $F_{X_j}(t) = \mathcal{P}(X_j \leq t)$.
  - $\mathrm{E}[X_j] = 1/\mu_j$ and $\mathrm{E}[X_j^2] = \sigma_{X_j}^2 + 1/\mu_j^2$ are the first two moments of the service time of class j.
- $\rho_j = \lambda_j \mathrm{E}[X_j]$ is the utilization coefficient of class j.
- $\rho = \rho_1 + \cdots + \rho_P$ is the utilization coefficient of the server. It must be < 1 for stability.

Priority terminology – Definitions

- **SHARING**
  - **Head-Of-Line (HOL)** : the server is assigned to at most a *single* customer (not shared).
  - **Server sharing** : the service capability of the system is sliced and shared by customers: more then one customer can be partially served at any given time.
- **PREEMPTION**
  - **Non-preemptive priority** : once a customer is assigned the server, it completes its service before releasing the server.
  - **Preemptive priority** : a lower priority customer service can be interrupting upon the arrival of a higher priority customer.
- **JOB SIZE**
  - **Non-anticipative** : the job size is not known a priori and scehduling does not dpeend on it.
  - **Anticipative** : scheduling can be based also on job size.

Scheduling

SCHEDULING = Algorithm for managing the capacity of a single server.

We will see the following:

- **Head-Of-Line (HOL) strategies**
  - Non-preemptive, non-anticipative: FCFS, static priority.
  - Non-preemptive, anticipative: SJF.
  - Preemptive, non-anticipative: LAS.
  - Preemptive, anticipative: SRPT.
- **Server sharing policies**
  - Processor Sharing.→ Round Robin.
  - Generalized Processor Sharing.→ Credit Based Scheduling..

Conservation law

Theorem (Conservation law for HOL systems)

For a work-conserving HOL priority system in equilibrium we have

$$\sum_{j=1}^{P} \frac{\rho_j}{\rho} \mathrm{E}[W_j] = \frac{W_0}{1-\rho} = \mathrm{E}[W_{FCFS}]$$

where $\rho_j = \lambda_j \mathrm{E}[X_j]$ *is the utilization coefficient of class* j, $\rho = \rho_1 + \cdots + \rho_P$ *is the utilization coefficient of the system,* $\mathrm{E}[W_j]$ *is the mean waiting time of class* j*, and* $W_0 = \frac{1}{2}\sum_{j=1}^{P} \lambda_j \mathrm{E}[X_j^2]$ *is the residual service time found by an arriving customer.*

The conservation law says that improving one class comes to the detriment of the performance experienced by other classes.

Mean waiting time of the $M/G/1$ queue

Let us consider a single server queue with Poisson arrival of mean rate λ and general i.i.d. service times X.
First-Come First-Served policy implies that the waiting time of a tagged arriving customers is

$$W = \tilde{X}_0 + \sum_{j=1}^{N} X_j$$

where

- $\tilde{X}_0$ is the time since the arrival of a tagged customer until when the server completes its current service (if any)..
- N is the number of customers found by the arriving customer in the waiting line.
- X_j is the service time of the j-th customer in the waiting line.

Mean waiting time of the $M/G/1$ queue

The mean waiting time is

$$\mathrm{E}[W] = \mathrm{E}[\tilde{X}_0] + \mathrm{E}[N]\mathrm{E}[X] = W_0 + \mathrm{E}[N]\mathrm{E}[X]$$

We apply Little's law to the *waiting line*.

- Mean arrival rate: λ
- Mean number of customers in the waiting line: $\mathrm{E}[N]$.
- Mean time spent into the system $=$ mean waiting time: $\mathrm{E}[W]$.

Hence

$$\mathrm{E}[W] = W_0 + \lambda\mathrm{E}[W]\mathrm{E}[X] = W_0 + \rho\mathrm{E}[W] \quad \Rightarrow \quad \mathrm{E}[W] = \frac{W_0}{1-\rho}$$

What is W_0?

Let us consider a customer arriving at the single-server queue.

By definition, $W_0 = \mathrm{E}[\tilde{X}_0]$, where $\tilde{X}_0$ is the time it takes for the server to complete the current service, if any:
Let B be a binary random variable such that $B = 1$ if and only if the server is busy upon arrival.

$$\mathrm{E}[\tilde{X}_0 | B = b] = \begin{cases} 0 & b = 0, \text{ w.p. } 1 - \rho \\ \frac{\mathrm{E}[X^2]}{2\mathrm{E}[X]} & b = 1, \text{ w.p. } \rho \end{cases}$$

Then, it is

$$W_0 = \mathrm{E}[\tilde{X}_0] = \rho \frac{\mathrm{E}[X^2]}{2\mathrm{E}[X]} = \frac{\lambda \mathrm{E}[X^2]}{2}$$

Static non-preemptive HOL priorities

Arriving customers are assigned a "label" expressing their priority in the set $\{1, \ldots, P\}$ ($1 =$ highest priority).

If arrivals are Poisson, the mean waiting time of class j customer is

$$\mathrm{E}[W_j] = \frac{W_0}{\left(1 - \sum_{i=1}^{j-1} \rho_i\right)\left(1 - \sum_{i=1}^{j} \rho_i\right)}, \qquad j = 1, \ldots, P$$

High variance service times of low priority customers affect also high priority customers (non-preemptive priority).

HOL static priorities minimize the mean cost rate $\sum_{j=1}^{P} c_j \overline{N}_j$: if c_j is the cost per unit time of waiting for customer of class j, assign priorities in decreasing order of the ratios $c_j/\mathrm{E}[X_j]$.

Static non-preemptive priorities: overall mean

The overall mean waiting time is

$$\mathrm{E}[W] = \sum_{j=1}^{P} \frac{\lambda_j}{\lambda} \mathrm{E}[W_j]$$

This is not necessarily the same as FCFS mean wait time.

That is the case, if the mean service times of all classes are the same.

Example

- Two flows of jobs with same mean arrival rate: $\lambda_1 = \lambda_2$.
- Service times of type 2 jobs are 10 timer bigger than type 1.
- the overall load is $\rho = \rho_1 + \rho_2 = 0.9$. Then $\rho = \lambda_1 \mathrm{E}[X_1] + \lambda_2 \mathrm{E}[X_2] = \lambda_1 \mathrm{E}[X_1] + \lambda_1 \cdot 10 \cdot \mathrm{E}[X_1] = 11 \cdot \rho_1 = 0.9 \Rightarrow \rho_1 \approx 0.082$ and $\rho_2 \approx 0.818$).

We have

$$\mathrm{E}[W_1] = \frac{W_0}{1-\rho_1} \qquad \mathrm{E}[W_2] = \frac{W_0}{(1-\rho_1)(1-\rho_1-\rho_2)} = 10 \cdot W_1$$

hence $\mathrm{E}[W_1] \approx 1.089 W_0$ and $\mathrm{E}[W_2] \approx 10.89 W_0$. Note that $\mathrm{E}[W_{FCFS}] = 10 W_0$.

Shortest Job First (SJF)

An arriving customer with a service time requirement of x is placed in front of any waiting customers with service time greater than x (non-preemptive priority).

Assume customer arrive according to a Poisson process with mean rate λ. Then the mean waiting time, conditional on the required service time being x, is:

$$\overline{W}(x) = \mathrm{E}[W|X = x] = \frac{W_0}{\left(1 - \lambda \int_0^x t \, dF_X(t)\right)^2}$$

- For very short service times $\overline{W}(x) \to W_0$ as $x \to 0$.
- For very long service times $\overline{W}(x) \to W_0/(1-\rho)^2$ as $x \to \infty$, where $\rho = \lambda \mathrm{E}[X]$.

Least Attained Service (LAS)

Least Attained Service also know as *Foreground-Background*.

Question: how can we give preference to shiort jobs if we assume we do NOT know the job size?

Answer: the **age** of a job may be an indicator for the job size.

- Policy used in UNIX to schedule jobs – also known as Foreground-Background (FB).
- Two queues:
  - New jobs join queue 1.
  - Upon hitting age a_{th} (a given threshold configured in the system), a job is moved to queue 2.
  - PS scheduling is used is used within each queue; queue 1 has priority over queue 2.
- LAS is obtained as a limiting policy that generalizes the above to infinite queues.

Least Attained Service (LAS)

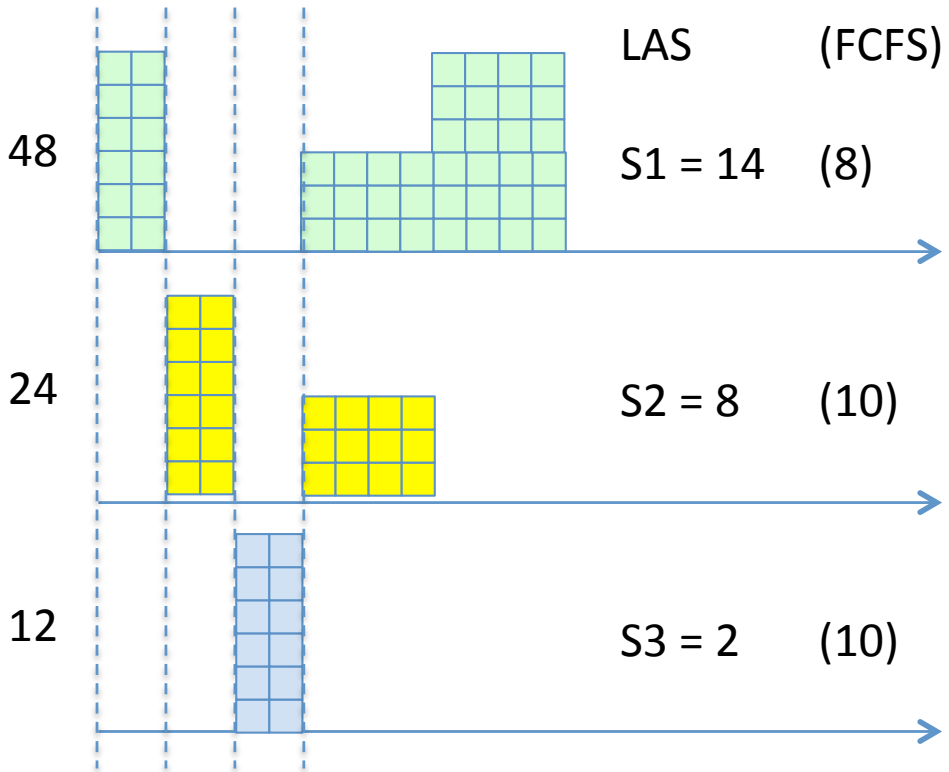
Prefer the customer that has currently achieved the least amount of received service.

- If the set of customers having received the least amount of service at time t is made up of n customers, each one of them gets μ/n; other customers wait.
- This sharing goes on until either
  1. a new customer arrives;
  2. or, one of currently served customers end its service.
- Response time conditional on requesting a service time x:

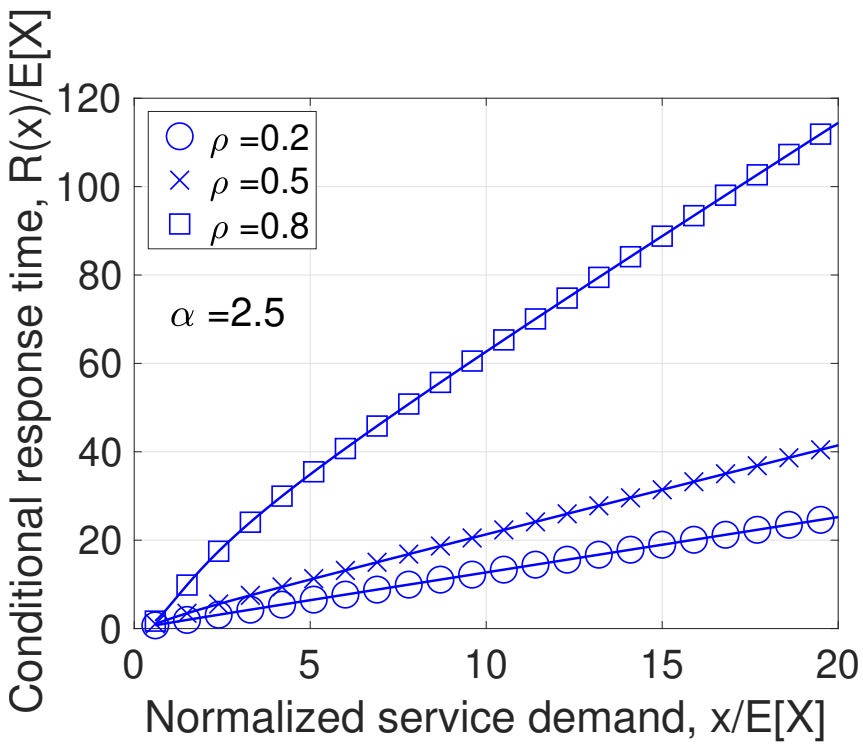
$$S(x) = \frac{\lambda m_2(x)}{2[1-\lambda m_1(x)]^2} + \frac{x}{1-\lambda m_1(x)}$$

where $m_h(x) = \int_0^x t^h \, dF_X(t) + x^h[1 - F_X(x)]\,, h = 1, 2..$

Least Attained Service: example



LAS numerical example



Mean conditional response time $S(x)$ with LAS scheduling as a function of $x/\mathrm{E}[X]$ for three values of ρ (Poisson arrivals, Pareto service times).

Shortest Remaining Processing Time (SRPT)

An arriving customer with a service time requirement of x is placed in front of any customers with remaining service time greater than x and can preempt the server (preemptive priority).

It can be proved that SRPT minimizes the mean system time for a single server queue in the class of HOL queueing policies.

$$\mathrm{E}[S_{\mathrm{SRPT}}] \leq \mathrm{E}[S_{\mathrm{SJF}}] \leq \mathrm{E}[S_{\mathrm{FCFS}}]$$

Note that $\mathrm{E}[S] = \mathrm{E}[W] + \mathrm{E}[X]$.

Sharing the server

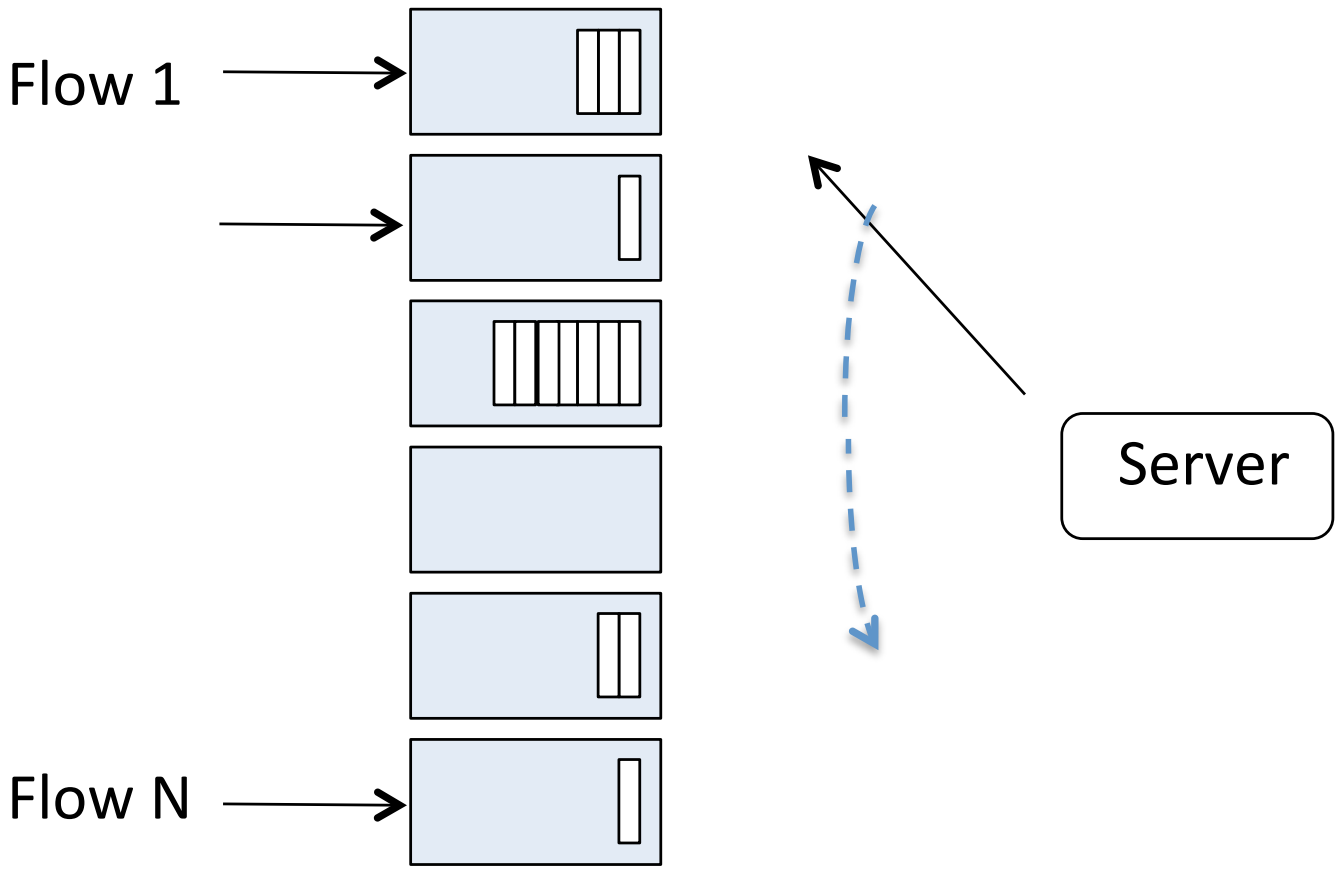
Up to now we have seen priority schemes where the server is assigned to one customer at a time (HOL priority). They define essentially the order of service.

A key issue in service system is **ISOLATION** of customer flows.

ISOLATION is meant to:

- *protect* one flow from all the others, guaranteeing access to service notwithstanding how "aggressive" other flows are.
- give *differentiated* service to different flows.
- provide service *guarantees*.

Processor sharing model



Processor sharing

- Ideal model of a shared server.
- Each customer in the queue gets an equal share of the server capacity μ: if there are n customers, each gets μ/n.
- We consider a shared server of capacity μ (mean service rate), where customers arrive according to a Poisson process of mean rate λ.
- Assume a finite work quantum Δ. The system time of a tagged customer arriving at time t_0 and requesting a service time x is:

$$S(x) = \sum_{k=0}^{x/\Delta-1} [\Delta + \Delta\, Q(t_k)] = x + \frac{x}{x/\Delta} \sum_{k=0}^{x/\Delta-1} Q(t_k)$$

where $t_{k+1} = t_k + \Delta[1 + Q(t_k)]$, $k \geq 0$.

Processor sharing (cont'd)

- Taking averages at equilibrium:

$$\mathrm{E}[S|X = x] = x + x\,\mathrm{E}[Q]$$

- Removing the conditioning on x, we get

$$\mathrm{E}[S] = \mathrm{E}[X] + \mathrm{E}[X]\,\mathrm{E}[Q] = \mathrm{E}[X] + \mathrm{E}[X]\,\lambda\mathrm{E}[S]$$

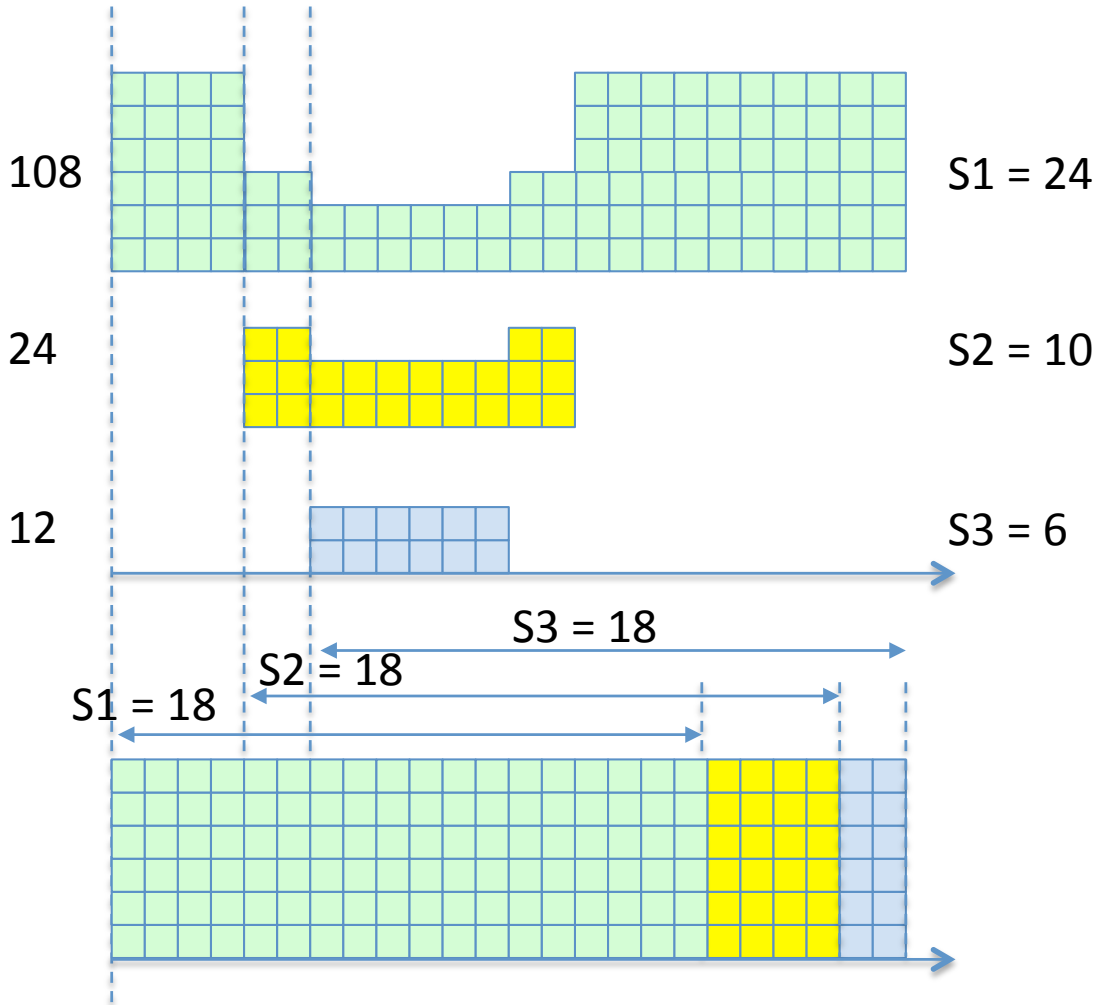
  where we have applied Little's law.

- The mean system time is:

$$\mathrm{E}[S] = \frac{\mathrm{E}[X]}{1-\rho}$$

  where $\rho = \lambda\mathrm{E}[X]$ is the utilization factor of the server.

Processor Sharing: example



Processor Sharing response time

Summing up:

$$\text{MEAN RESPONSE TIME} = \frac{\mathrm{E}[X]}{1-\rho}$$

$$\text{MEAN RESPONSE TIME FOR WORKLOAD } x = \frac{x}{1-\rho}$$

$$\text{SLOWDOWN FOR WORKLOAD } x = \frac{1}{1-\rho}$$

Processor Sharing: comment

FCFS:

$$\mathrm{E}[S] = \mathrm{E}[X] + \frac{\rho \mathrm{E}[X]}{1-\rho}\frac{1+C_X^2}{2}$$

PS:

$$\mathrm{E}[S] = \mathrm{E}[X] + \frac{\rho \mathrm{E}[X]}{1-\rho}$$

Compare FCFS and PS:

- Bothe mean response times diverge as $\rho \to 1$.
- PS is **insensitive** to the probability distribution of the service times: its mean system time depends only on the mean service time (no variance!).
- For a fixed $\rho < 1$, FCFS response time increases as the variance of service times grows. PS only depends on the mean service time.

Generalization and implementation

- PS can be generalized to unequal server shares by using weights (GPS — Generalized Processo Sharing).
  - Let $\phi_i > 0$ be the weight of the i-th customer (or class of customers or also flow), at any time t a backlogged customer gets a server share eqaul to

$$\mu \frac{\phi_i}{\sum_{j \in \mathcal{B}(t)} \phi_j},$$

    where $\mathcal{B}(t)$ is the set of backlogged customers at time t.
- PS and GPS are theoretical models (infinitesimal service quanta). Practical implementations are
  - Round-Robin for PS
  - Deficit Round Robin, Weighted Fair Queueing, Credit-Based Schduling for GPS.

Round Robin (RR)

- For round robin to be effective, flows should be composed of "small" slices of work demand.
- The server takes care of one work slice from each class, cyclically. After having completed a slice of service at flow j, the server moves on to flow $1 + (j + 1) \bmod (P + 1)$.
- It can be turned into weighted round robin, by simply completing ϕ_j slices of work at any visit to flow j.

(Weighted) RR is a practical, approximate realization of (G)PS.

- The service of a single slice of work is atomic: it cannot be interrupted. As a consequence, the order in which served slices leave the system is not necessarily the same as with (G)PS.
- There might be an overhead associated with swapping from one flow to the next one.

Credit-Based Scheduling (CBS) algorithm

1. Let $t = 0$, $h = 1$, and $K_j = 0$ for $j = 1, \ldots, N$.
2. Let $\mathcal{J}(t)$ be the set of classes backlogged at time t and L_j be the amount of work requested by the head-of-line customer of class j.
3. Let $y_j \equiv \frac{L_j - K_j}{\phi_j}$, $j \in \mathcal{J}(t)$ and let j^* denote the index of the minimum among the y_j, $j \in \mathcal{J}(t)$.
4. Serve the head-of-line customer of class j^*, set $X_h = L_{j^*}/\mu$ and update counters as follows:

$$K_j \leftarrow K_j + \phi_j \frac{L_{j^*} - K_{j^*}}{\phi_{j^*}}, \qquad j \in \mathcal{J}(t) \setminus \{j^*\} \tag{1}$$

and

$$K_{j^*} \leftarrow 0 \tag{2}$$

5. Set $t \leftarrow t + X_h$, $h \leftarrow h + 1$, and go back to step 2

Fairness of CBS

Let $V_j(a, b)$ be the amount of work of class j served in the time interval $[a, b]$.

It can be shown that:

Theorem

Let $\mathcal{J}(a, b)$ be the set of classes that are continuously backlogged from scheduling step a to step b. Then, $\forall t_1, t_2$, $\forall i, j \in \mathcal{J}(t_1, t_2)$, we have

$$\left| \frac{V_i(t_1, t_2)}{\phi_i} - \frac{V_j(t_1, t_2)}{\phi_j} \right| \leq \frac{L_{max}}{\phi_i} + \frac{L_{max}}{\phi_j} \tag{3}$$

where $V_k(t_1, t_2)$ is the overall amount of demand of class k that has been served in the interval $[t_1, t_2]$.

Fairness of CBS

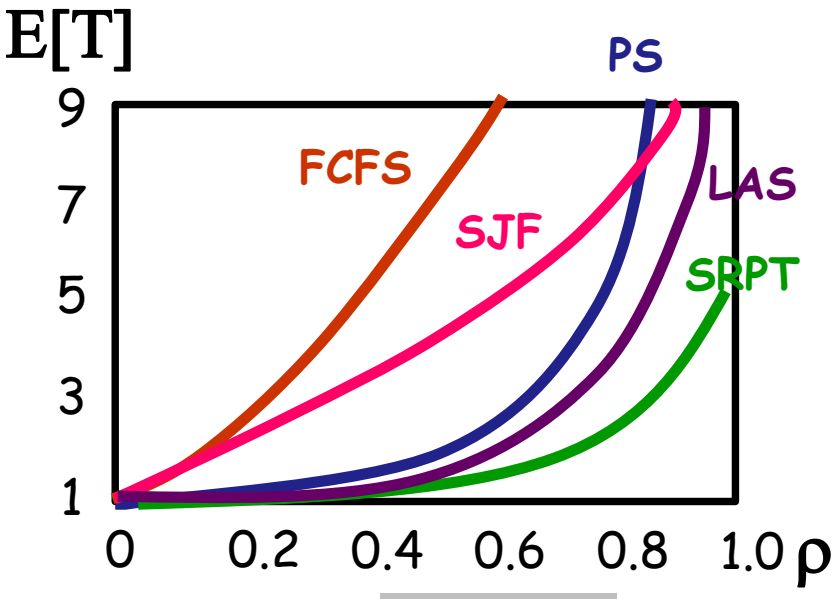
Note that the amount of work carried out by the server during $[t_1, t_2]$, if it is continuously busy, is $V(t_1, t_2) = (t_2 - t_1)\mu$. This amount of work is split among the active flows:

$$\sum_{j \in \mathcal{J}(t_1, t_2)} V_j(t_1, t_2) = V(t_1, t_2) = (t_2 - t_1)\mu$$

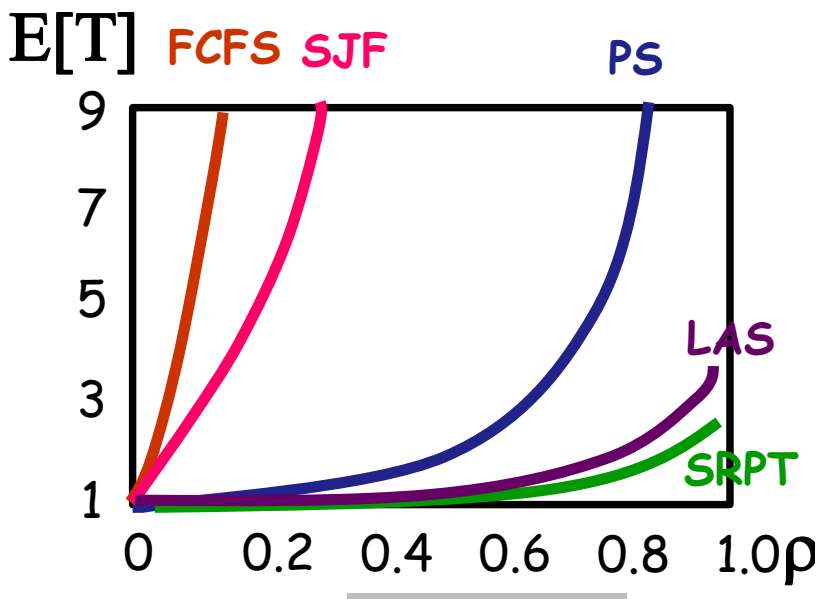
Say $t_1 = 0$. As $t_2 \to \infty$ the amount of work done by the server grows unboundedly.

The fairness theorem says that each one of the summands $V_j(t_1, t_2)$ tends to infinity as well (no flow is left behind!).

Comparison of all policies



$C^2=10$



$C^2=100$

Fairness

Size-based scheduling tend to discriminate against large jobs.

concerrn $\rightarrow$ FAIRNESS

PS is fair: same slowdown for all jobs!.

Q: Should we give up the response time improvement offered by SRPT?

All-can-win theorem

Theorem (Bansal, Harchol-Balter, 2001)

In an $M/G/1$ system, for any job size PDF, if $\rho < 1/2$ it is $R_{SRPT}(x) \leq R_{PS}(x), \quad \forall x.$

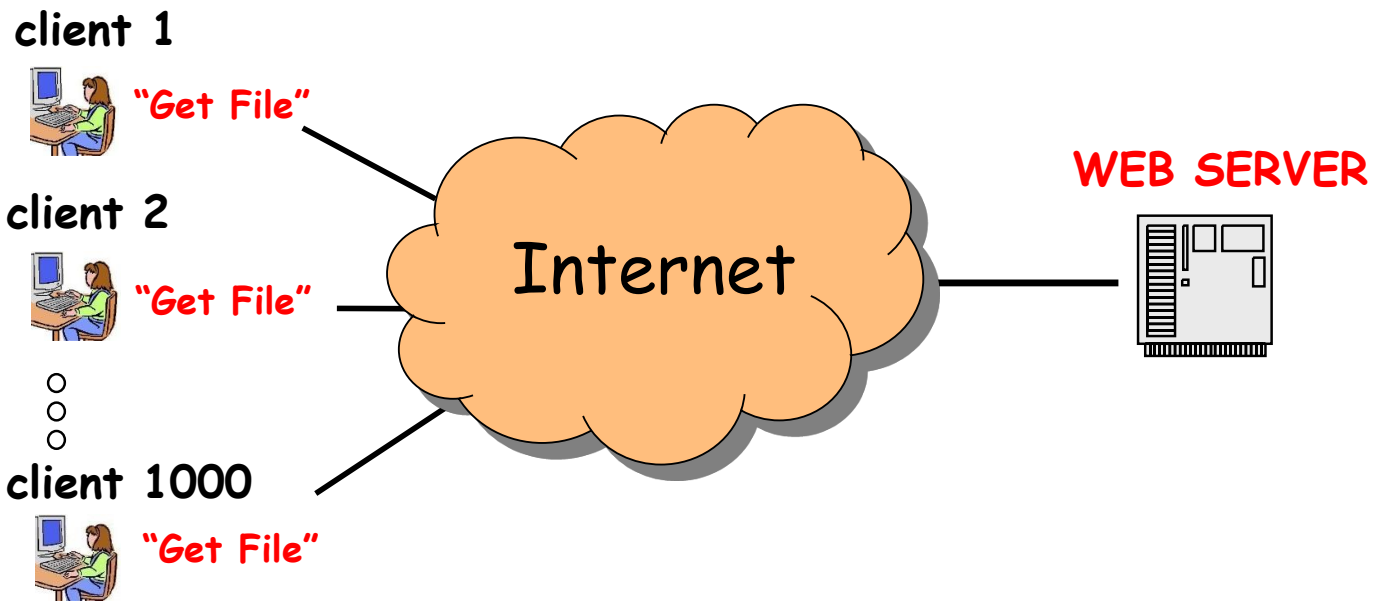
INTUITION.

- SRPT apparently penalizes large jobs. However, this is true only until they start getting some service. At that point the job gains priority.
- Imagine load is light. An arriving job has a high probability to find the server idle, hence the wait-before-service-starts component of response time is small.
- The time-to-completion component is clearly better in SRPT than with PS.

All-can-win theorem – lesson learned

- Keeping the load factor below a suitable threshold can improve policies significantly.
- The threshold of the All-can-win theorem can be relaxed in case of heavy-tailed job sizes (e.g., it can be as high as s 0.96 with suitable Pareto PDF).
- Besides adopting smart policies, we can buy performance benefits by spending part of the serving capacity.
- Often capacity is an abundant resource: we do not get it for free, but it comes easier than other features.

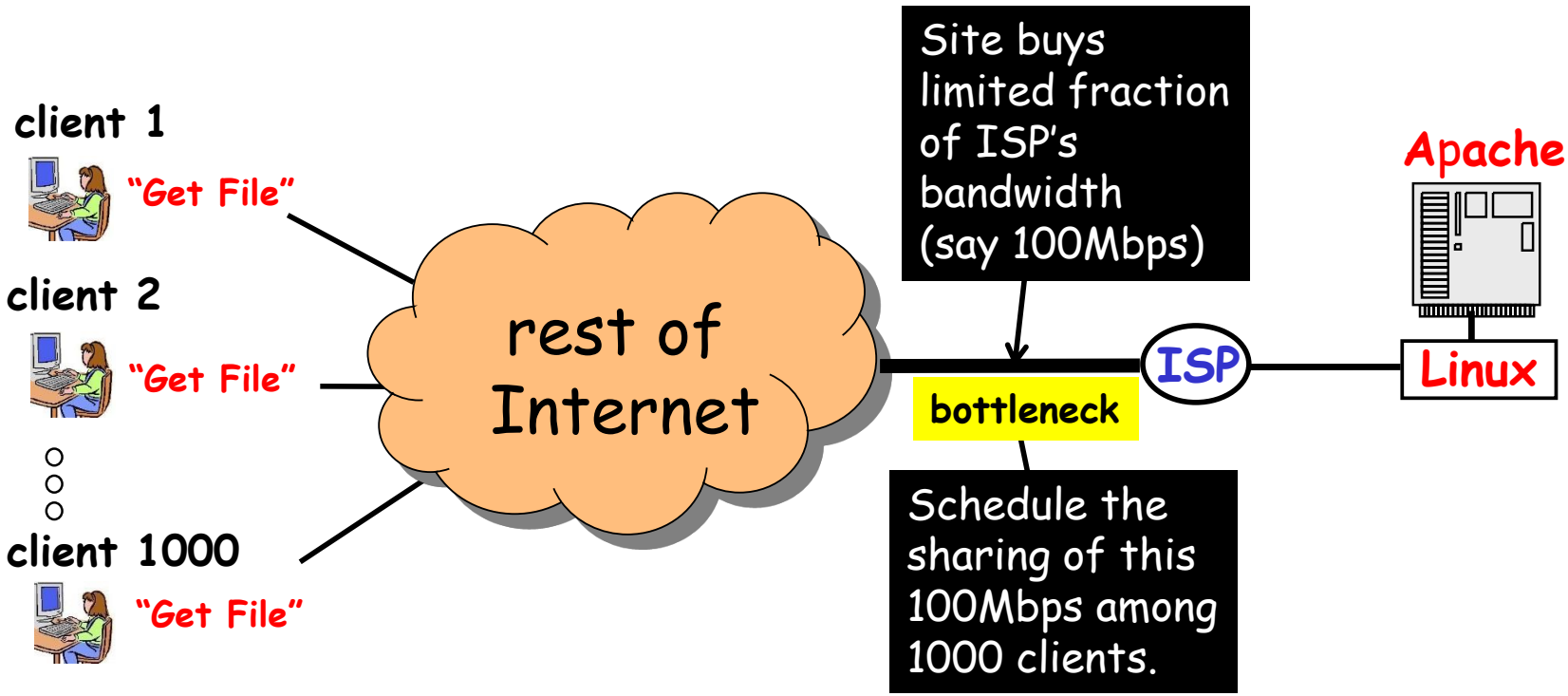
Application of SRPT to web server farms



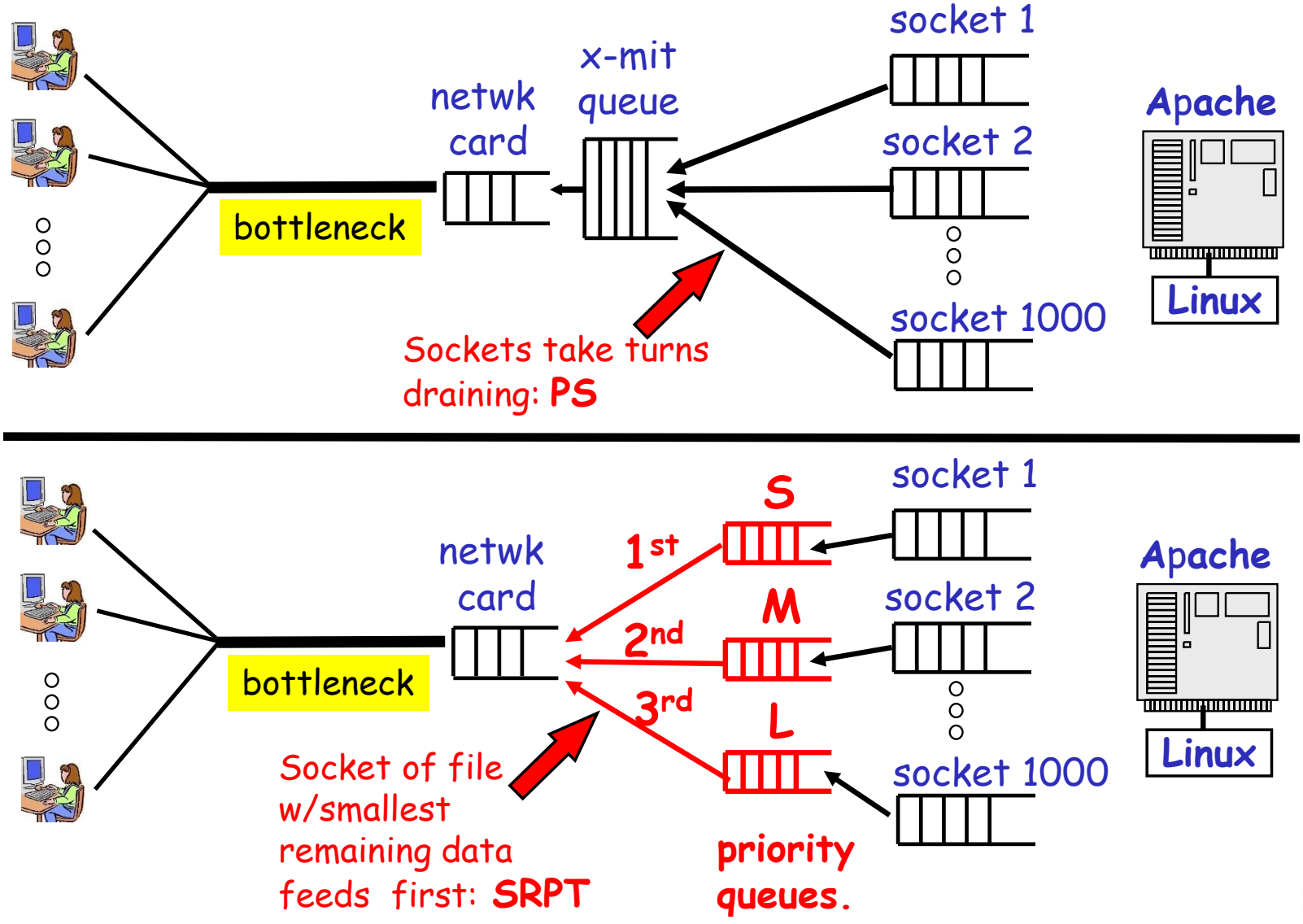
Identify:

- what is the resource.
- what is a "job". and what is the "job size".

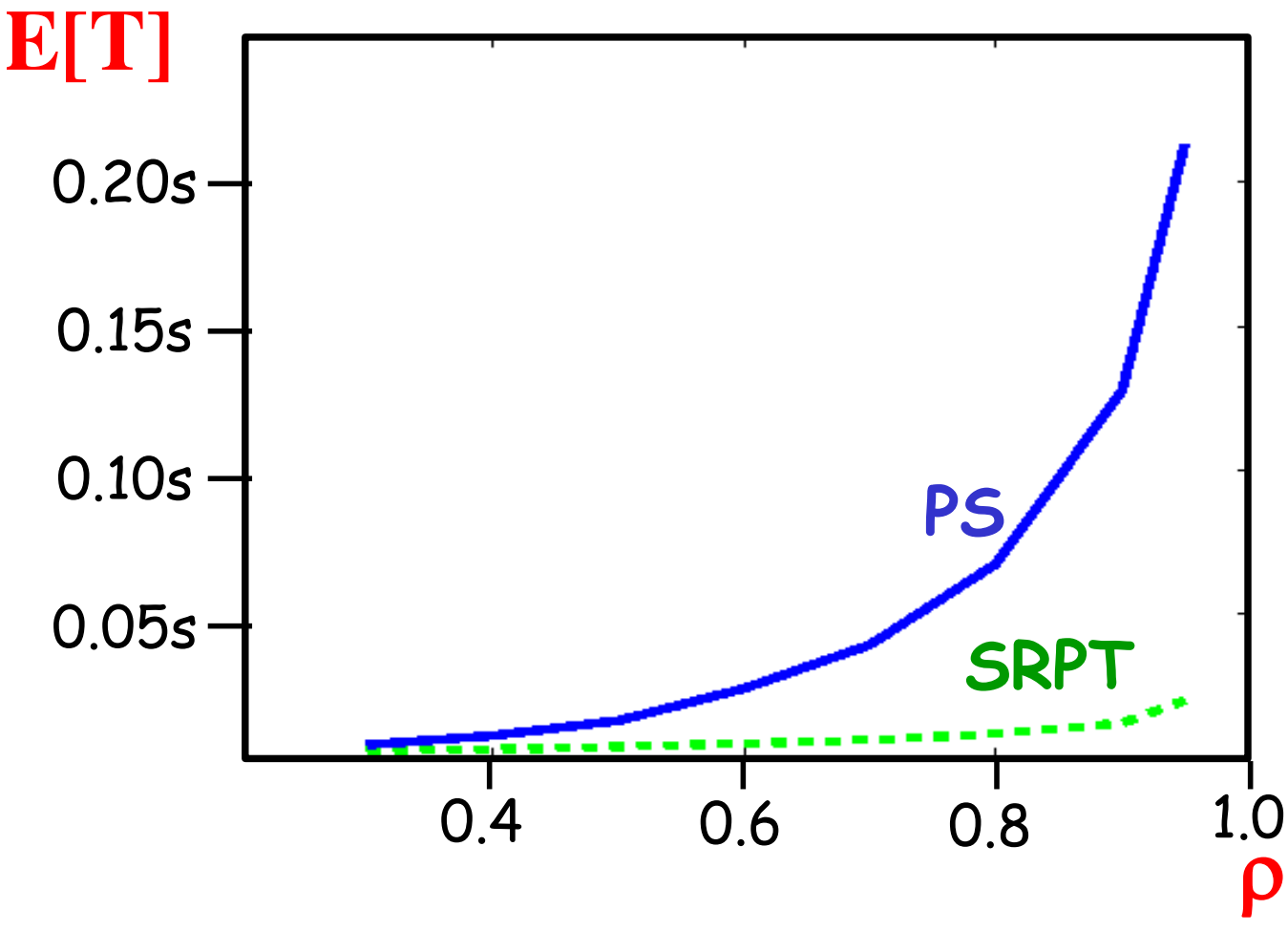
Application of SRPT to web server farms (cont'd)



Application of SRPT to web server farms (cont'd)



Application of SRPT to web server farms – performance



Single vs multiple servers

- Until this point we have considered *single* server systems.
- We have seen that pooling the service capability into one server offers better mean delay performance than breaking it up into a multiplicity of separated servers.

Is this the whole story?

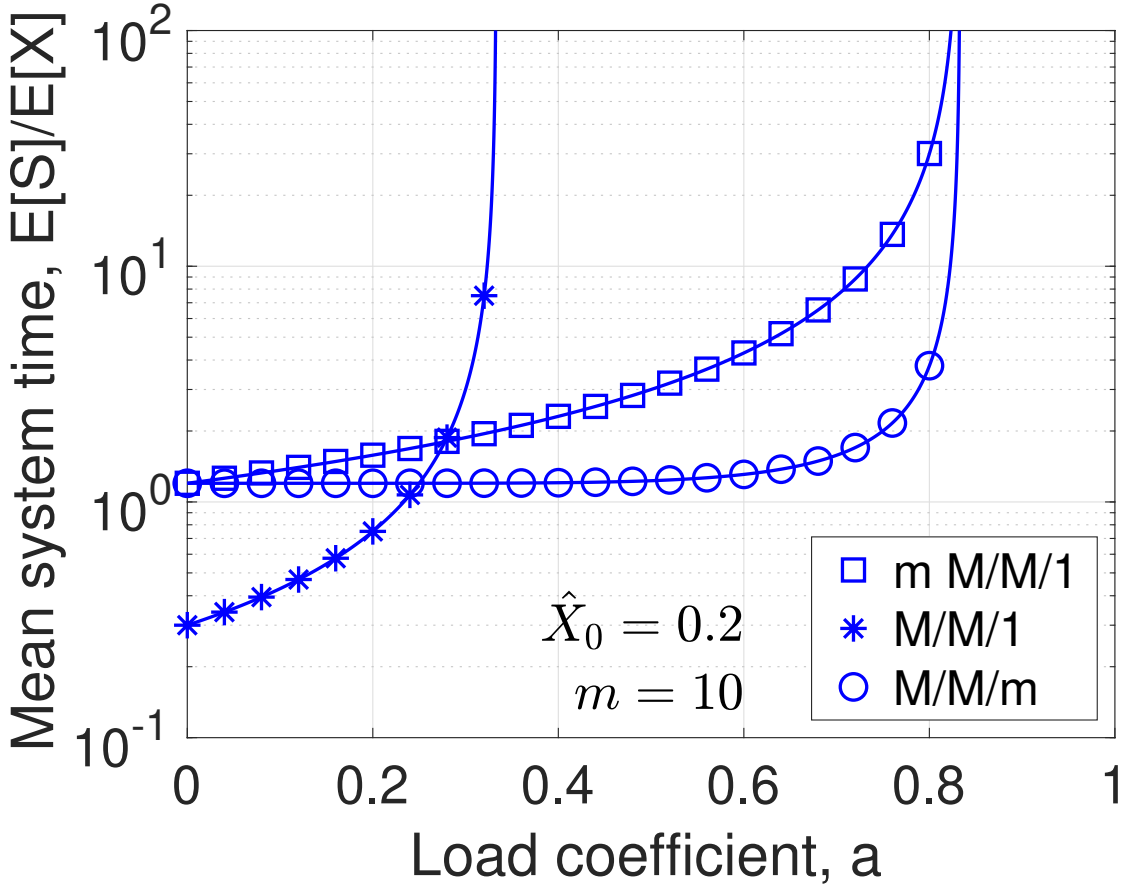
Why multiple servers (1/3)

A theoretical objection to single servers

What if service implies a overhead that does not scale?

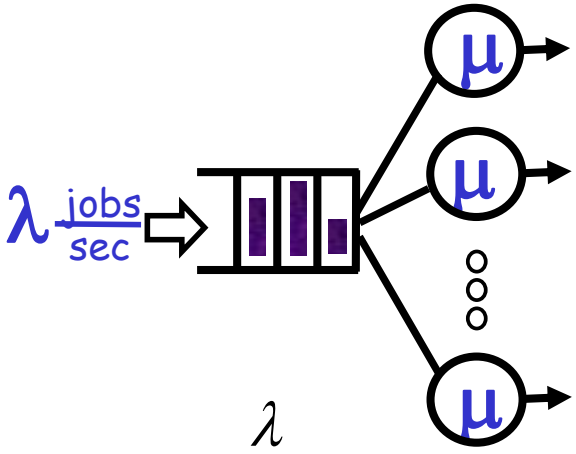
- Assume service time is the sum of a fixed component X_0 and a component that scales with the parameter m, i.e., it is $X = X_0 + X_1/m$.
- The average service time becomes $\mathrm{E}[X] = X_0 + \frac{1}{m\mu}$.
- We re-do the comparison of single-server/single-line, single-server/multiple-line, multiple-server/multiple-line systems.

Numerical example

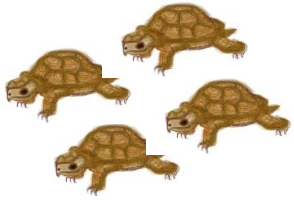


Multi-server vs pooled server revisited

M/G/k

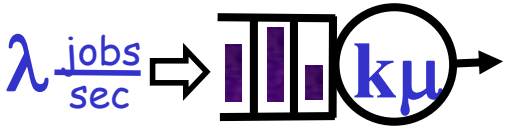


$$\rho = \frac{\lambda}{k\mu}$$



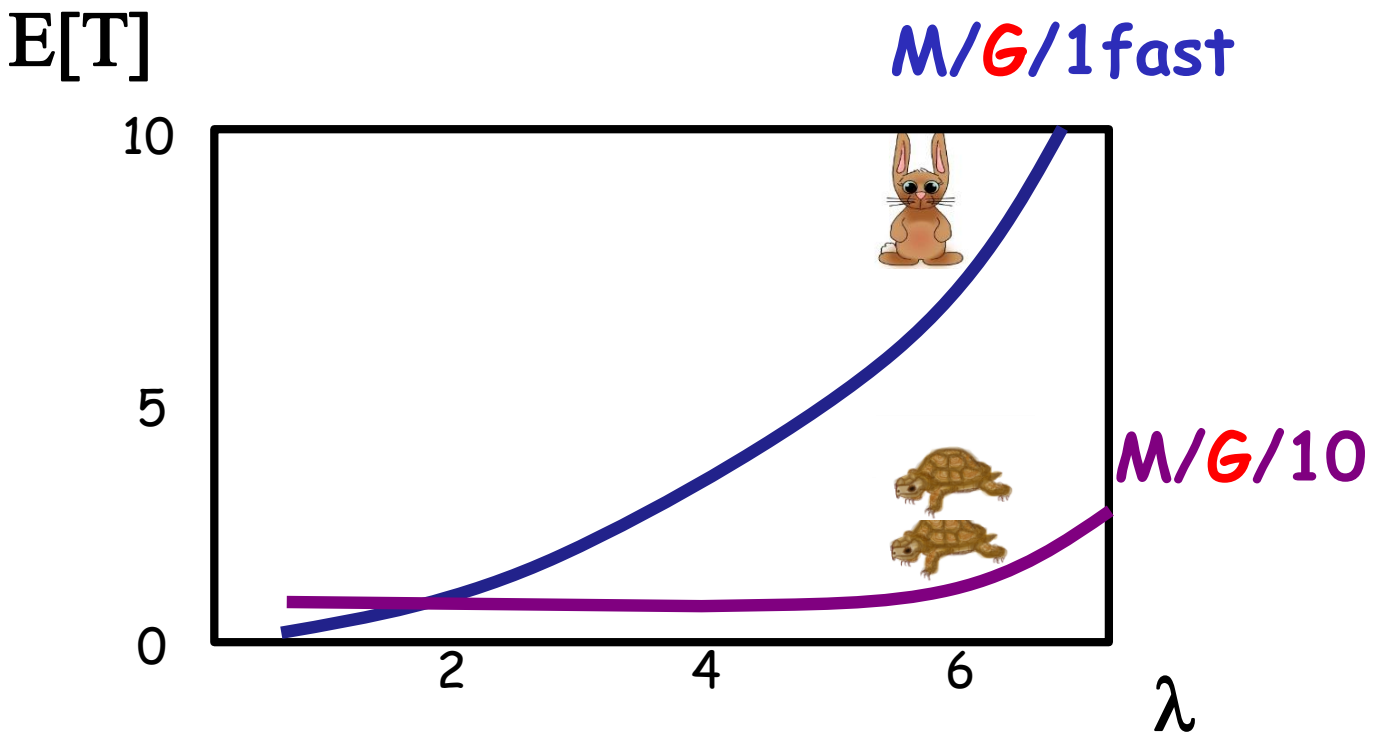
vs.

M/G/1fast



$$\rho = \frac{\lambda}{k\mu}$$

Multi-server vs pooled server revisited



Service time PDF has SCOV = 100.

Why multiple servers (2/3)

Multi-server system can scale

We'd like to have a service system that can be utilized heavily (convenient for the system provider) and *simultaneously* imposes nearly zero delay on customers (high QoS).

- **Utilization coefficient** ~ 1 .
- **Mean waiting time** ~ 0 .

Is that possible?

Why multiple servers (2/3)

Multi-server system can scale

We'd like to have a service system that can be utilized heavily (convenient for the system provider) and *simultaneously* imposes nearly zero delay on customers (high QoS).

- **Utilization coefficient** ~ 1 .
- **Mean waiting time** ~ 0 .

Is that possible?

YES!

Multi-server system with unlimited queue size

- Consider a service system with N servers and one waiting line.
- Poisson arrivals of mean rate λ.
- Negative exponential service times with mean $1/\mu$.
- The ratio $A = \lambda/\mu$ is the mean offered traffic intensity. For stability it must be $A/N < 1$.

The normalized mean waiting time is

$$\overline{W} = \frac{\mathrm{E}[W]}{\mathrm{E}[X]} = \mu \frac{C(N, \lambda/\mu)}{N\mu - \lambda} = \frac{C(N, A)}{N - A}$$

where $C(N, A) = \frac{B(N,A)}{1 - \frac{A}{N} + \frac{A}{N} B(N,A)}$ and $B(N, A)$ is Erlang-B function.

Let us consider the asymptotic regime with $N = A + \beta\sqrt{A}$, for $A \to \infty$ (Halfin-Whitt regime).

Many-server system asymptotic analysis

It can be shown that[5]

$$B(A+\beta\sqrt{A},A) \sim \frac{\phi(\beta)}{\Phi(\beta)}\frac{1}{\sqrt{A}}, \qquad A\to\infty$$

Then

$$C(A+\beta\sqrt{A},A) = \frac{B(A+\beta\sqrt{A},A)}{1-\rho+\rho B(A+\beta\sqrt{A},A)} \sim \frac{\left(1+\frac{\beta}{\sqrt{A}}\right)\phi(\beta)}{\beta\Phi(\beta)+\phi(\beta)}$$

The normalized mean waiting time becomes:

$$\overline{W} = \frac{\mathrm{E}[W]}{\mathrm{E}[X]} = \frac{C(A+\beta\sqrt{A})}{A+\beta\sqrt{A}-A} \sim \frac{\phi(\beta)}{\beta\left[\beta\Phi(\beta)+\phi(\beta)\right]}\frac{1}{\sqrt{A}}$$

[5] $\phi(x) = e^{-x^2/2}/\sqrt{2\pi}$ and $\Phi(x) = \int_{-\infty}^{x}\phi(u)\,du$.

Many-server system asymptotic regime: comment

We see that in the asymptotic regime of a large system

- number of servers N tending to infinity;
- mean offered traffic A tending to infinity;
- same order;

we have

$$\text{Utilization coefficient } = \rho = \frac{A}{N} \sim \frac{1}{1+\beta/\sqrt{A}} \to 1$$

and

$$\text{Mean wait } = \overline{W} \sim \frac{\phi(\beta)}{\beta\left[\beta\Phi(\beta)+\phi(\beta)\right]}\frac{1}{\sqrt{A}} \to 0$$

Many-server system asymptotic regime: example

Let $A = 100$ and $\beta = 1$

- The number of servers is $N = 100 + 1 \cdot \sqrt{100} = 110$.
- We have $\phi(\beta) \approx 0.2420$ and $\Phi(\beta) \approx 0.8413$.
- Hence $\rho \approx 0.9091$ and $\overline{W}/\mathrm{E}[X] \approx 0.0223$.
- Utilization is beyond 90%, while the mean delay is 2.23% of the mean service time.

Let $A = 900$ and $\beta = 2$

- The number of servers is $N = 900 + 2 \cdot \sqrt{900} = 960$.
- We have $\phi(\beta) \approx 0.0540$ and $\Phi(\beta) \approx 0.9772$.
- Hence $\rho \approx 0.9375$ and $\overline{W}/\mathrm{E}[X] \approx 0.0004$.
- Utilization is almost 94%, while the mean delay is a megligible fraction of the mean service time.

Why multiple servers (3/3)

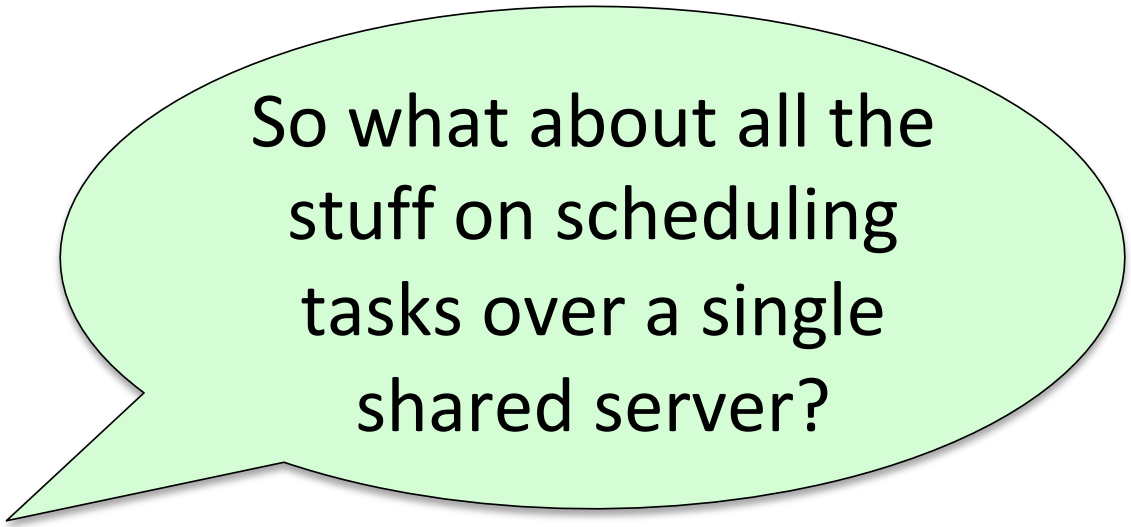
A practical objection to single servers

- As m grows, the single server with processing power $m\mu$ can become unfeasible.
- Moreover, it would be a single point of failure.
- It is more practical and cheaper to exploit Commercial Off The Shelf (COTS) hardware and software (e.g., commercial servers with standard OS) and reap high processing capacity by clumping together a large number of COTS devices.
  - Data Centers leverage on this philosophy.

Key issues

- **Communication among servers** (DC network) .
- **Load balancing** .

Single vs multiple servers: question ...

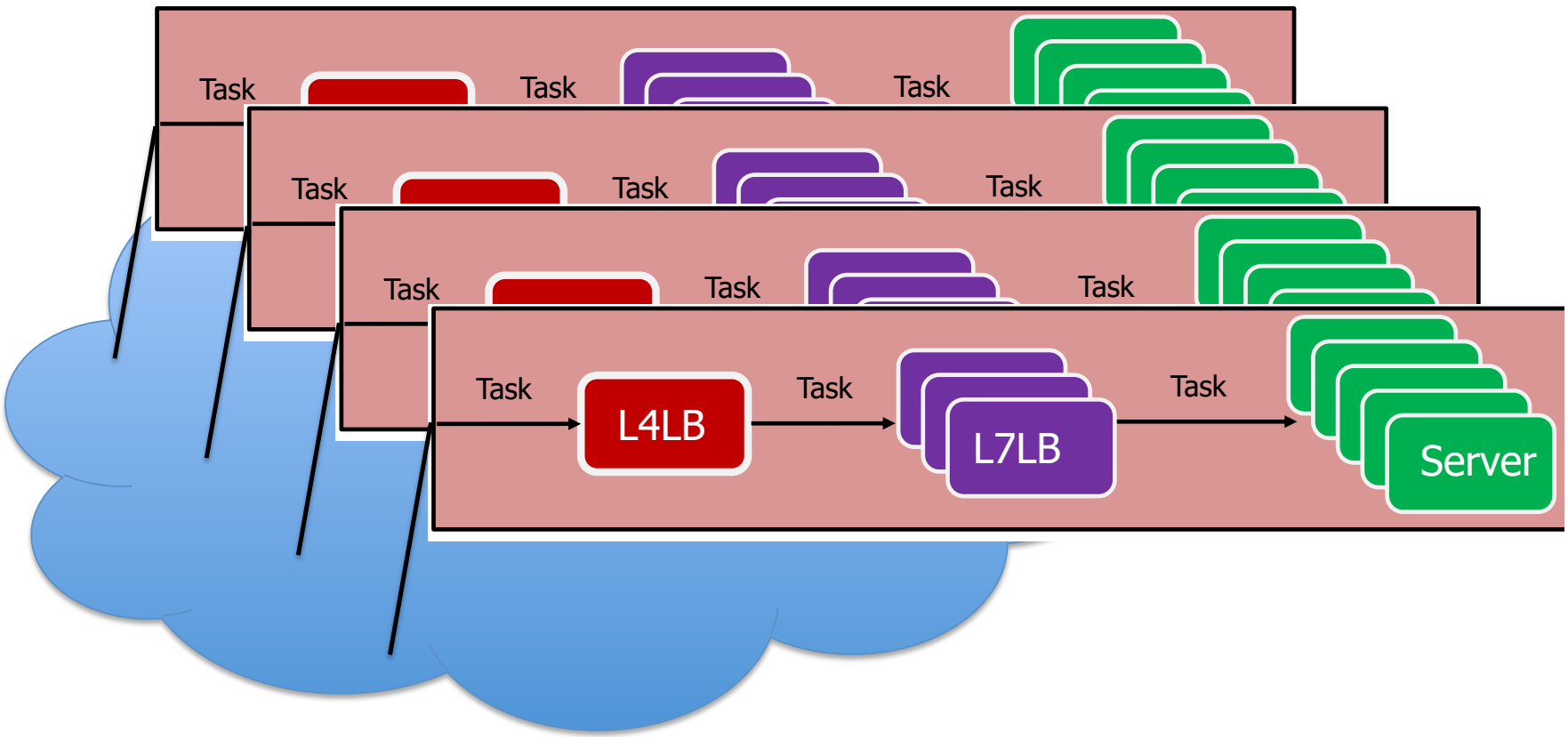


Single vs multiple servers: ... and answer

So what about all the stuff on scheduling tasks over a single shared server?

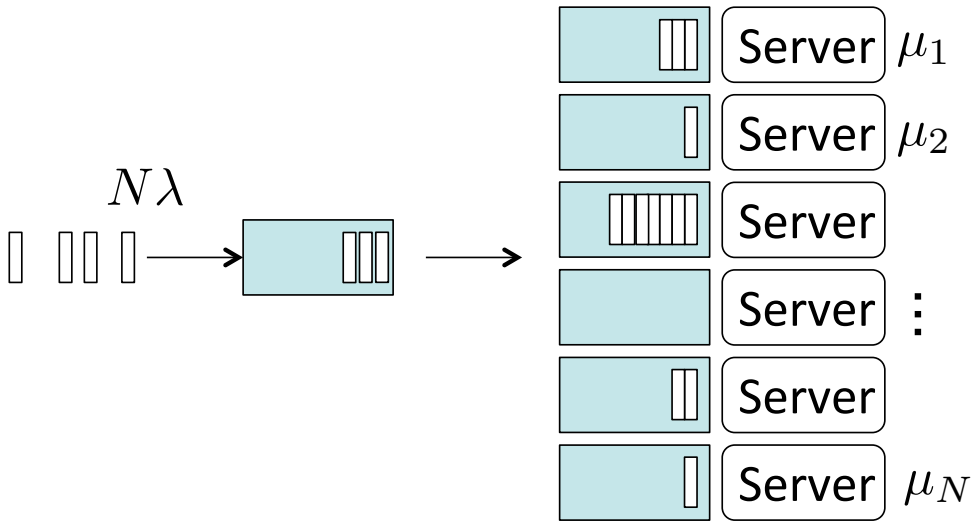
Virtual Machines , Dockers and Applications share processing power of a SINGLE physical machine!

Load balancing cluster.



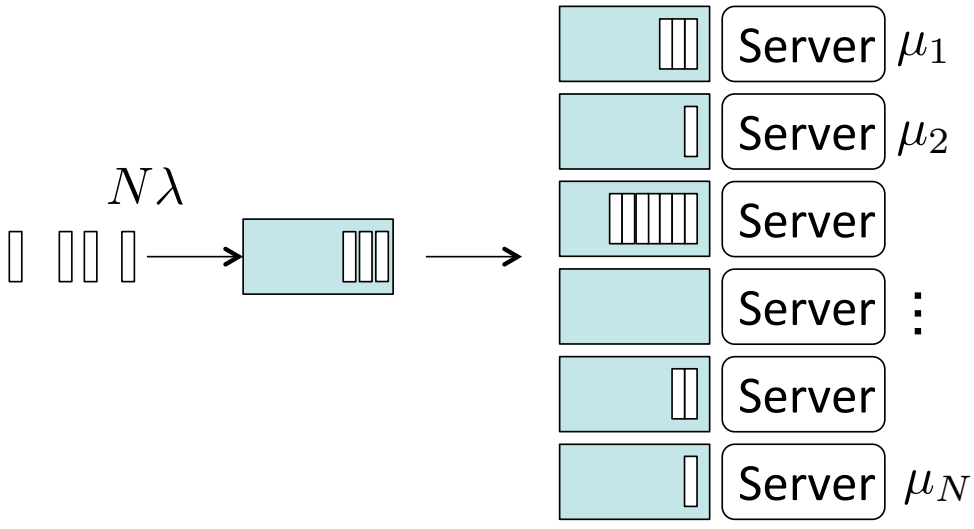
The LB is key to reduce latency and improve throughput.

System model



- Focus on a single load balancer (dispatcher).
- Tasks arrive to a dispatcher and are routed to the servers.
- Each server maintains a queue of tasks.
- Task remains in its queue until its service is finished.

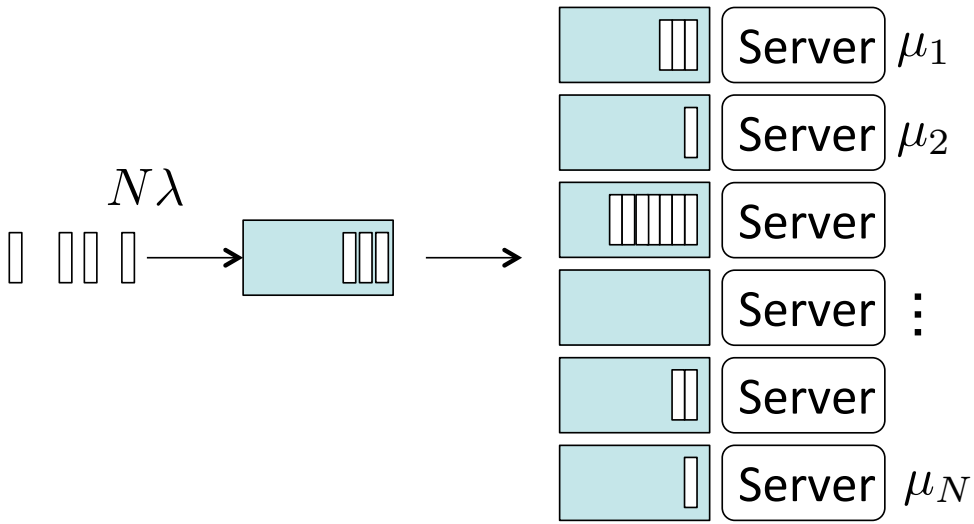
Performance goal



Goal of the LB: choose the right server

"RIGHT" = LOW LATENCY

Delay components

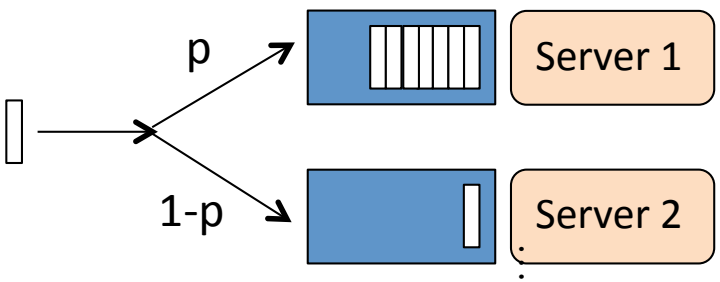


Delay is made up of two contributions:

- **Dispatching delay**: from arrive at dispatcher to join a server.
- **Response time** : from join a server to leave the server.

Example of load balancing (1/3)

- Jobs arrive according to a Poisson process of mean rate Λ.
- The servers work according to PS, with mean rates μ_1 and μ_2 respectively.
- Server 1 is faster: $\mu_1 = \alpha \cdot \mu_2$, with $\alpha > 1$.
- **Random dispatching policy**: The job dispatcher decides to send a job to server 1 with probability p.



Which is the best policy as α grows?

Example of load balancing (2/3)

Mean system time for $\rho = \frac{\Lambda}{\mu_1+\mu_2} < 1$:

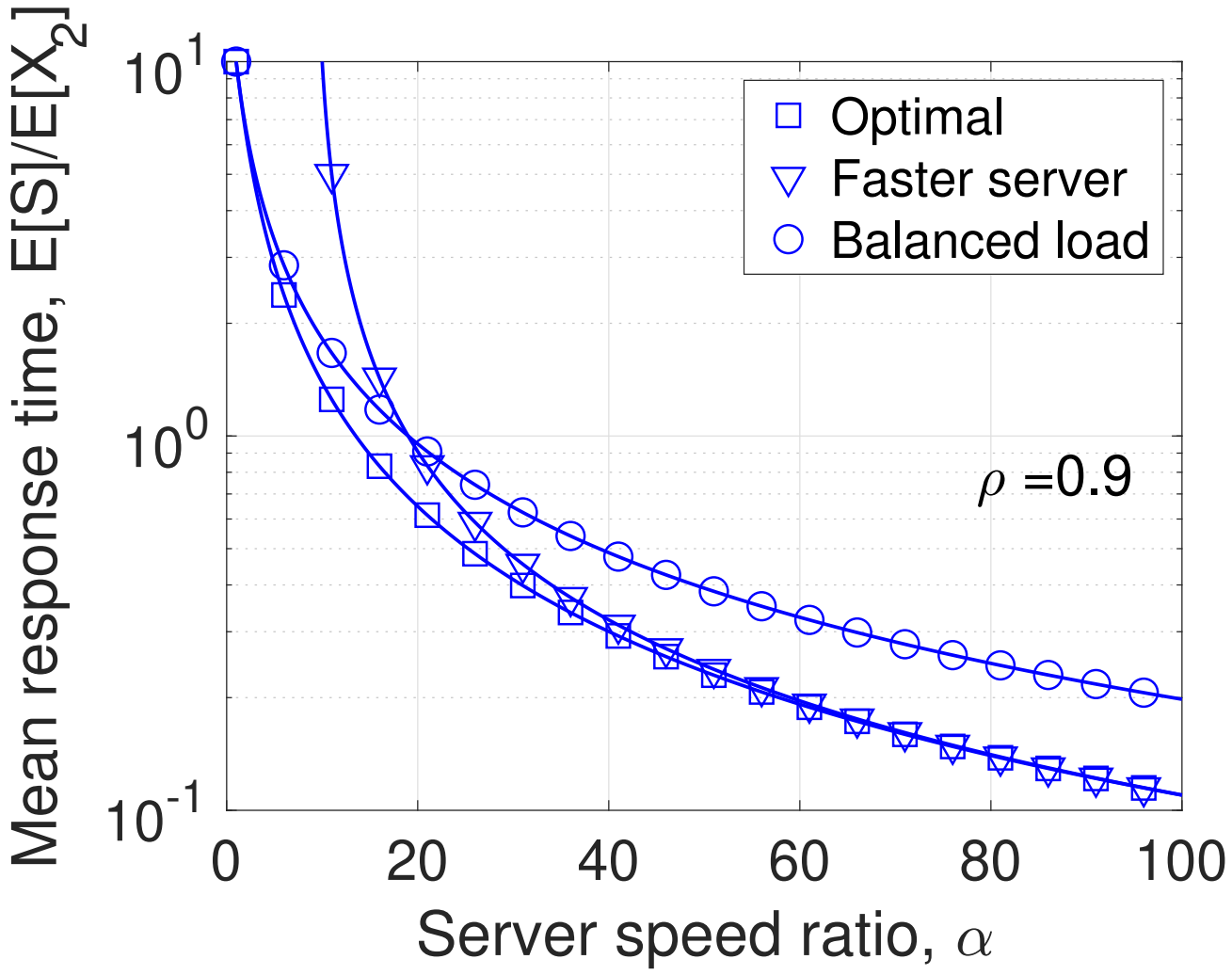
$$\mathrm{E}[S] = p \cdot \frac{1}{\mu_1 - \lambda_1} + (1-p) \cdot \frac{1}{\mu_2 - \lambda_2} = \frac{p}{\mu_1 - p\Lambda} + \frac{1-p}{\mu_2 - (1-p)\Lambda}$$

for $p_{\min} = \max\{0, (\Lambda - \mu_2)/\Lambda\} < p < p_{\max} = \min\{1, \mu_1/\Lambda\}$.

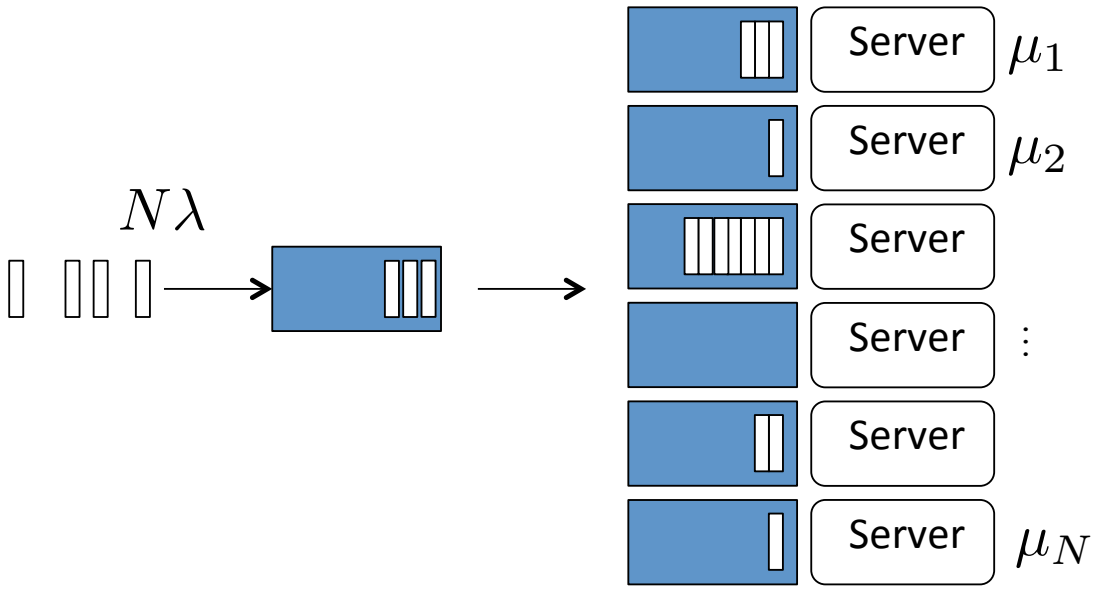
COMPARE:

- Optimized: $p = p^*$ such that $\mathrm{E}[S]$ is minimized as a function of p.
- Faster: $p = 1$ (Switch off the slower server!)
- Balanced: $\rho_1 = \rho_2$ that is $\frac{p\Lambda}{\mu_1} = \frac{(1-p)\Lambda}{\mu_2}$, hence $p = \frac{\mu_1}{\mu_1+\mu_2}$.

Example of load balancing (3/3)



The queueing model



Definition

- N : # of servers.
- $\Lambda = N\lambda$: mean arrival rate of tasks.
- μ_j : task processing rate of server j for $j = 1, \ldots, N$. We let also $\mu = \mu_1 + \cdots + \mu_N$.

Throughput optimality

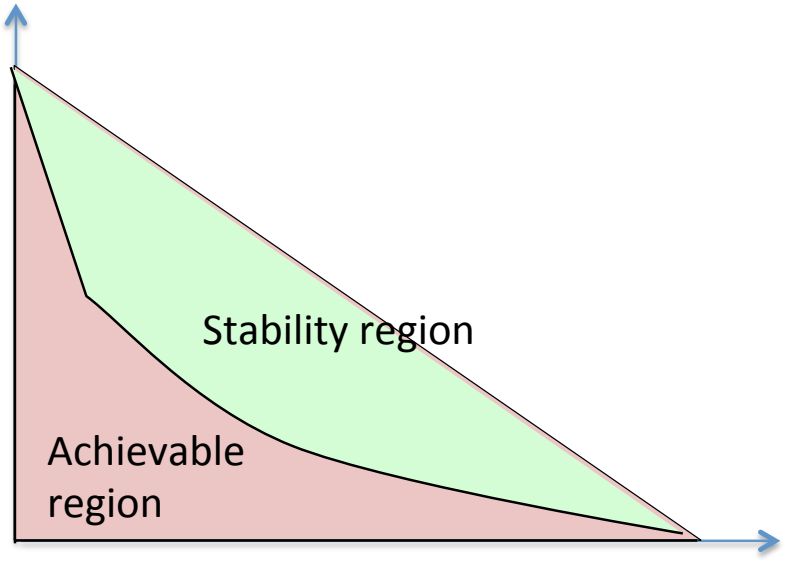
$$\textbf{Stability:} \quad \limsup_{C\to\infty} \limsup_{t\to\infty} \mathcal{P}\left(\sum_{j=1}^{N} Q_j(t) > C\right) = 0$$

In words: queue lengths are finite with probability 1.

A load balancing policy is said to be throughput optimal if it stabilizes the system under any arrival rate in the capacity region.

The capacity region in our case is $\Lambda = N\lambda < \sum_j \mu_j = \mu$.

Throughput optimality – Idea

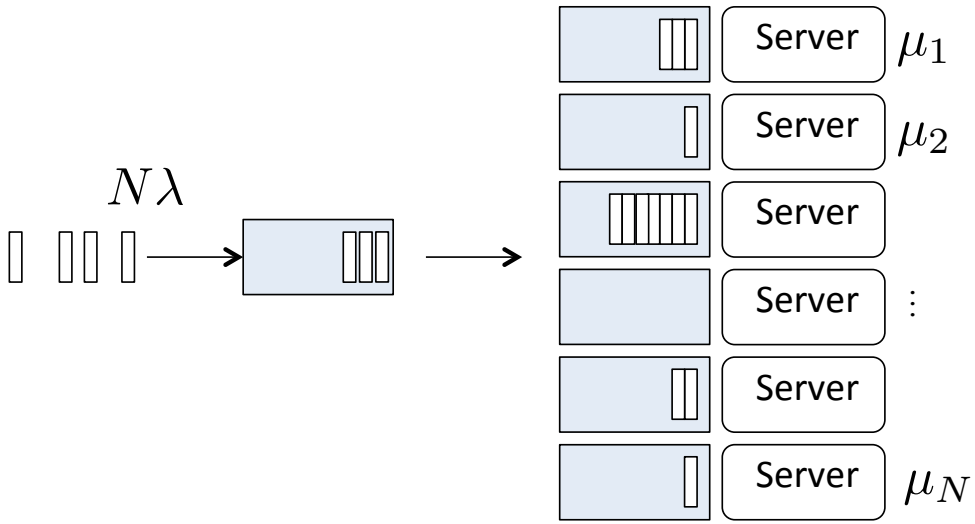


STABILITY REGION = set of system load points where the system can maintain finite queue lengths (aka STABILITY REGION).

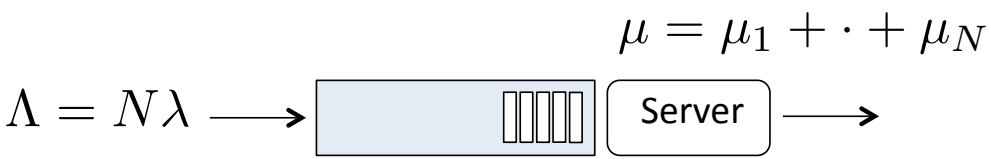
ACHIEVABLE REGION = set of system load points where the considered LB algorithm is able to maintain finite queue lengths.

Resource pooled system

Multi-queue system



Resource pooled system



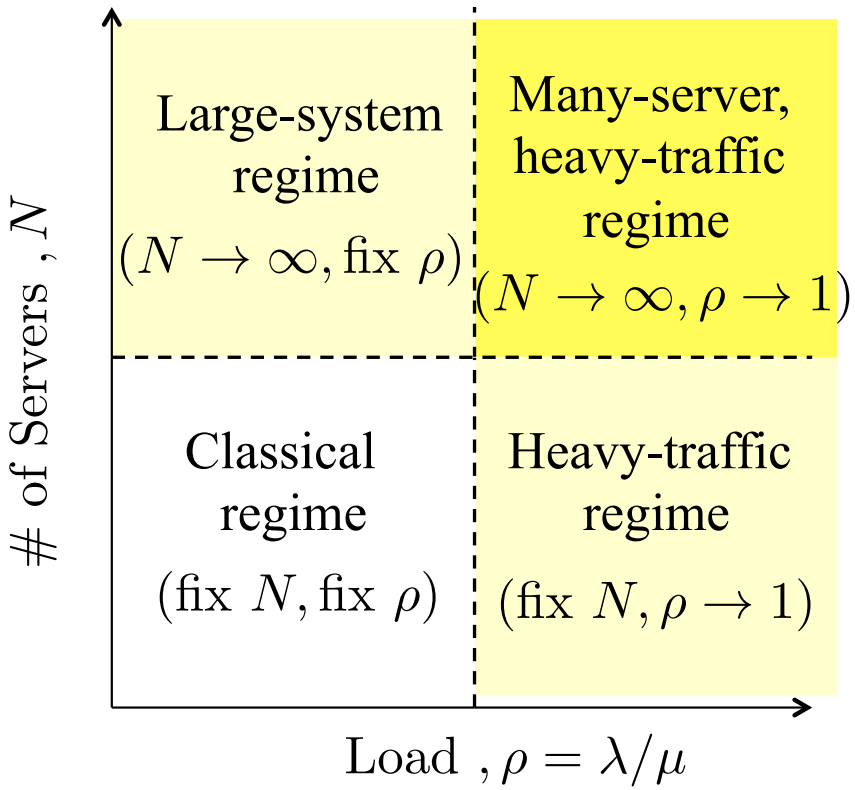
Resource pooled system

A single-server FCFS (first-come, first-serve) system $\{q(t), t \geq 0\}$ with arrival process $a(t)$ and service process $s(t)$ is said to be the resource pooled system with respect to the multi-queue system, if its arrival and service process satisfy $a(t) = A(t)$ and $s(t) = \sum_{n=1}^{N} S_n(t)$, for all $t \geq 0$, where $A(t)$ is the arrival process of the multi-queue and $S_n(t)$ is the service process of the n-th queue. Then, we have

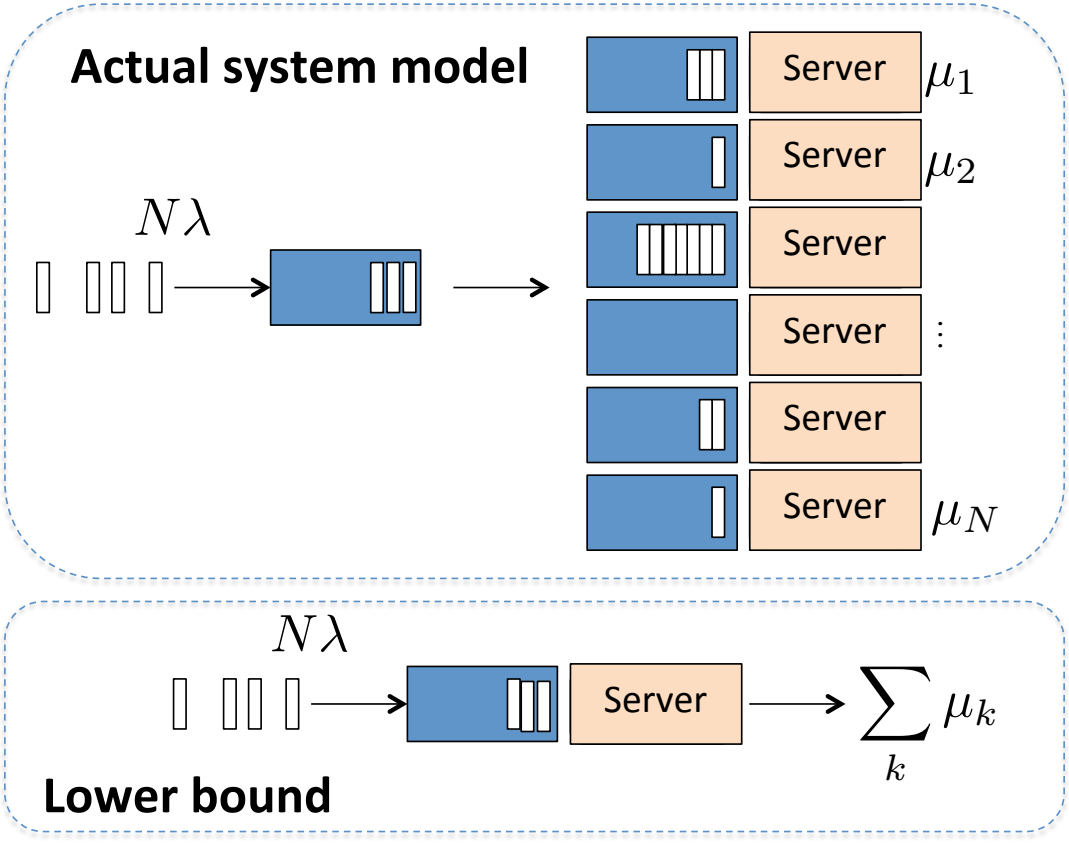
$$\mathrm{E}[q(t)] \leq \mathrm{E}\left[\sum_{n=1}^{N} Q_n(t)\right]$$

Studying delay optimality

- Goal: define delay-optimal load balancing schemes.
  - Difficult because even characterizing exact delay is hard.
- Possible approach to gain insight: turn to asymptotic regimes.



Heavy-Traffic (HT) delay optimality



HT optimality = the load balancer can achieve the delay performance **lower bound** of the resource pooled queueing system in the heavy traffic regime.

HT delay optimality

- Let q denote the steady-state queue length of the resource pooled single server queue.
- Let $Q = Q_1 + \cdots + Q_N$ be the steady-state number of tasks standing in the multi-queue system.
- For stability it must be $\mu - \Lambda = \epsilon > 0$. The HT regime is obtained for $\epsilon \to 0$.

HT delay optimality means that

$$\lim_{\epsilon \to 0} \epsilon \mathrm{E}[Q] = \lim_{\epsilon \to 0} \epsilon \mathrm{E}[q]$$

Although it sounds only a theoretical setting, HT is practically relevant. Virtually any algorithm works at low traffic level!

Classification of load balancing schemes

Load balancing scheme can be categorized according to whom takes the initiative for task assignment.

Pull

- The servers themselves tell the dispatcher that they are ready for the task (they "pull" the task).
- In practice, servers send messages to the dispatcher to update a system state kept at the dispatcher.
- Task assignment is based on the state at the dispatcher.

Push

- The servers respond to polling requests sent by the dispatcher.
- The dispatcher collects a number of replies from the server and decides where to send the task (push the task).

Most famous policies

- Non-anticipative, no state information:
  - **RANDOM**
  - **ROUND-ROBIN**
- Non-anticipative, with state information:
  - **JSQ** : Join Shortest Queue
  - **JIQ** : Join Idle Queue
  - **JBT** : Join Below Threshold
- Anticipative, with state information:
  - **SITA** : Size-Interval Task Assignment
  - **LWL** : Least Work Left

Performance ranking not obvious. Depends strongly on

- Job size variability.
- Policy at each server.
- Immediate dispatching or central queueing.

Simple algorithms

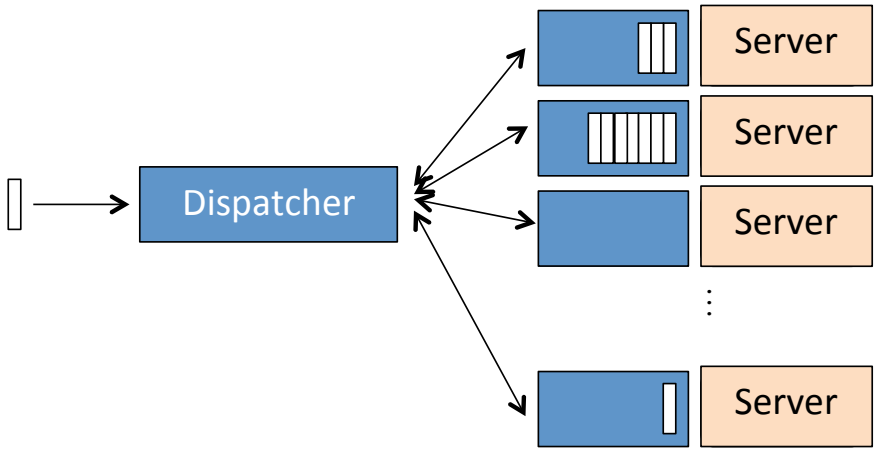
RANDOM: select one server uniformly at random upon task arrival.

- No state information required.
- No messaging.
- Job size knowledge not required.
- Bad performance under heavy traffic, at least for high variability job size.

ROUND ROBIN: select servers in a round-robin cyclic mode upon each task arrival.

- Similar to RANDOM, except that information on last assigned server must be maintained..

Push algorithm: Join Shortest Queue (JSQ)



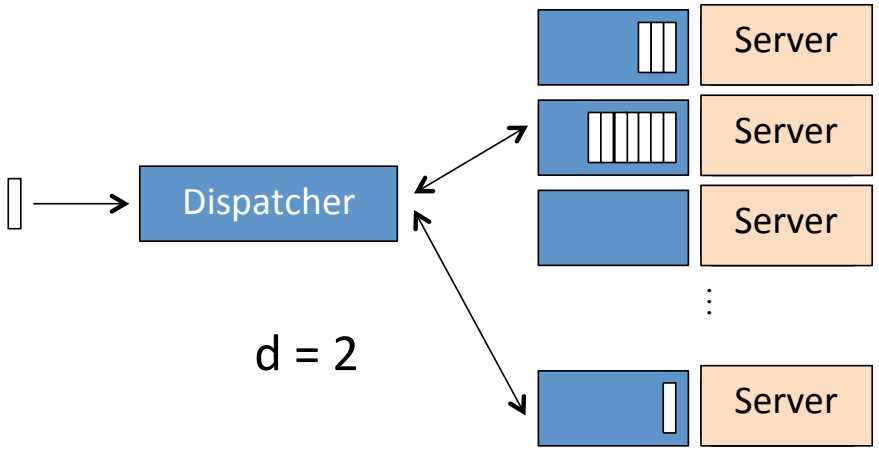
Pros

- Delay optimal in a stochastic order sense [Weber, 1978].
- Asymptotically delay optimal in heavy traffic [Foschini and Salz, 1978], [Eryilmaz and Srikant, 2012].

Cons

- High message overhead ($2N$/arrival) .
- Non-zero dispatching delay.

Push algorithm: Power-of-d (Pod)



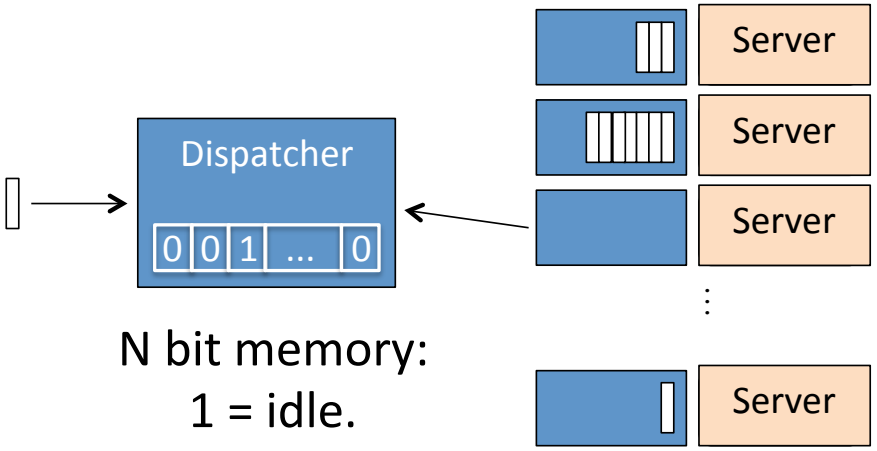
Pros

- Double exponential decay in delay tail when N is large [Mitzenmacher, 1996].
- Asymptotically delay optimal in heavy traffic [Chen and Ye, 2012], [Maguluri et al., 2014].

Cons

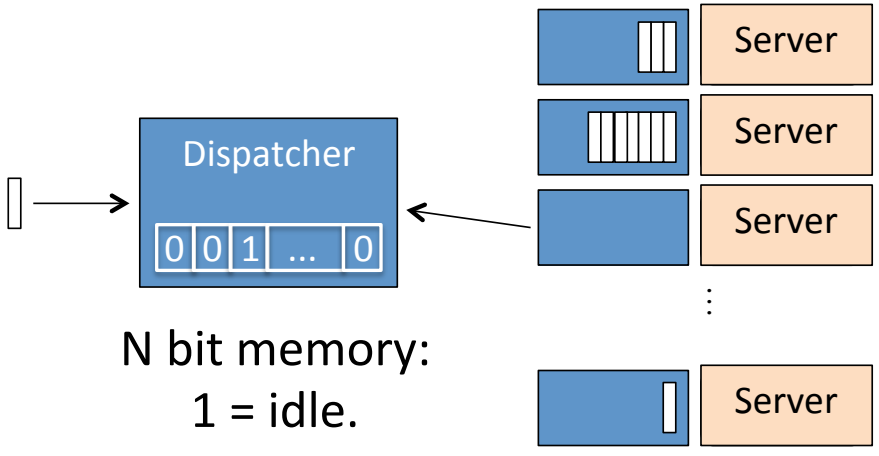
- Non-zero dispatching delay.

Pull algorithm: Join Idle Queue (JIQ)



- Dispatcher keeps a 1-bit-per-server memory state ($b_k = 1 \Leftrightarrow Q_k = 0$).
- Whenever a server goes idle (last task leaves), it sends a message to the dispatcher.
- Upon task arrival, the dispatcher routes the task to
  - a randomly chosen idle server, if any;
  - one server chosen at random among all servers, if none is idle.

Pull algorithm: Join Idle Queue (JIQ)



Pros

- Better delay performance than Pod under moderate loads.
- Overhead: ≤ 1 message per departure (=per arrival: why?).
- Zero dispatching delay.

Cons

- Performance downgrades substantially in HT.

Big picture

Push algorithms, e.g., JSQ, Pod.

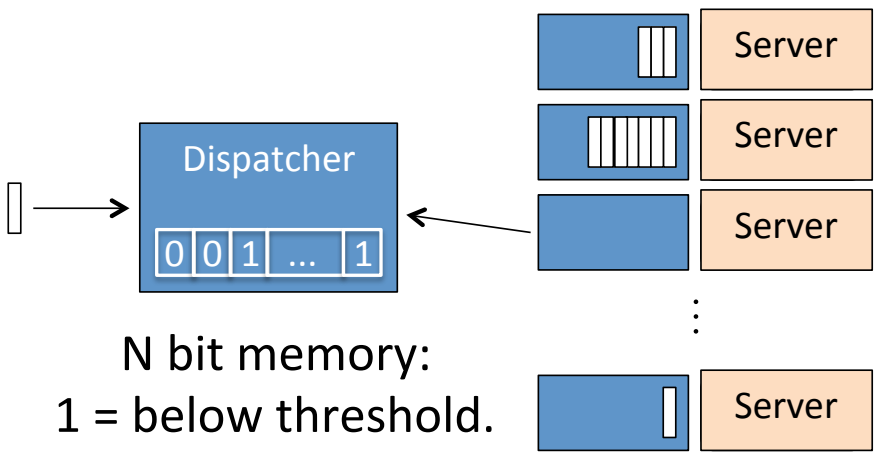
- Delay optimal (response time) in HT.
- Non-zero dispatching delays (push algorithms).
- Relatively high message overhead.

Pull algorithms, e.g., JIQ.

- Poor delay in HT.
- Zero dispatching delays .
- Low message overhead .

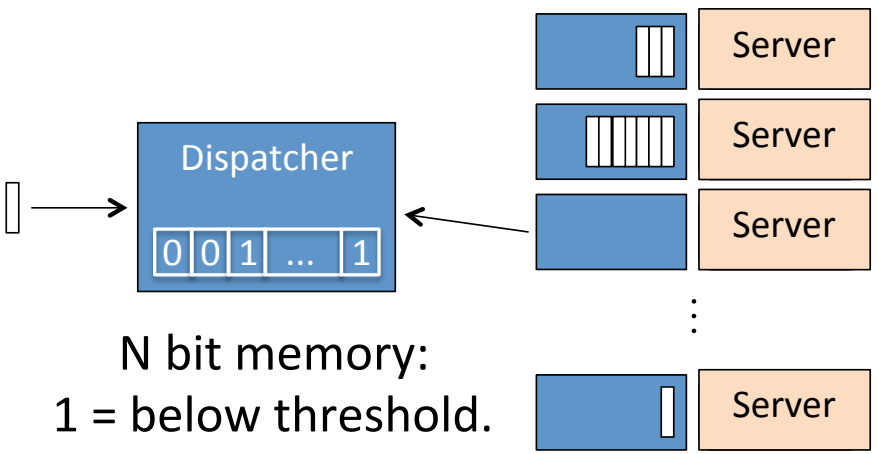
JBT(r) algorithm - dispatcher

- The dispatcher maintains a list of servers IDs with queue length less than r. The list is initialized with the IDs of all servers.
- Upon task arrival, the dispatcher picks an ID of the list at random, sends the task to the corresponding server and removes the ID from the list.
- The dispatcher updates its list with the ID of servers that announce that they have queue length below threshold.



JBT(r) algorithm - server

- The server monitors its task queue.
- As soon as the queue falls below the threshold r (transition from r to $r-1$), the server sends a message to the dispatcher to announce this event.



Practical implementation of JBT

Adaptive JBT(r) : **JBT-d**

1. A threshold is updated every T units of time by sampling d randomly selected servers, and taking the shortest queue length among the d servers as the new threshold. The dispatcher has to inform servers of the new value of r.
2. Each server sends its ID to the dispatcher when its queue length falls below the threshold.
3. Upon a new arrival, the dispatcher checks the available IDs in the memory. If the list is not empty, it removes one uniformly at random, and sends all the new arrivals in the current time unit to the corresponding server. Otherwise, all the new arrivals will be dispatched uniformly at random to one of the servers in the system.

Anticipative algorithms

SITA

- Divide job size range in N intervals, by setting thresholds $s_0 = 0 < s_1 < \cdots < s_N$.
- Assign job of size X such that $s_{i-1} < X \leq s_i$ to server i.
- Big issue: identify optimal threshold levels.

LWL

- Upon task arrival, poll all servers and ask their current unfinished work.
- Select a server having minimum unfinished work.
- Ties are broken at random

Performance ranking with FCFS servers

High E[T]

generally

Low E[T]

1. **Round-Robin**

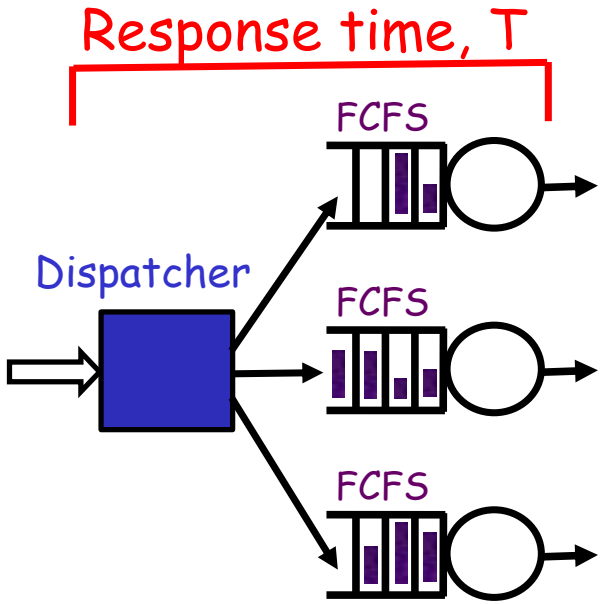
2. **Join-Shortest-Queue**
   Go to host w/ fewest # jobs.

3. **Least-Work-Left**
   Go to host with least total work.

‖

4. **Central-Queue (M/G/k)**
   Host grabs next job when free.

5. **Size-Interval Splitting**
   Jobs are split up by size among hosts.

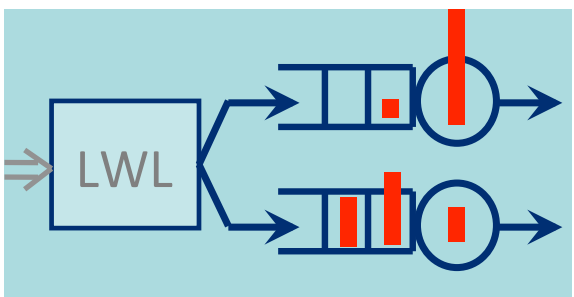


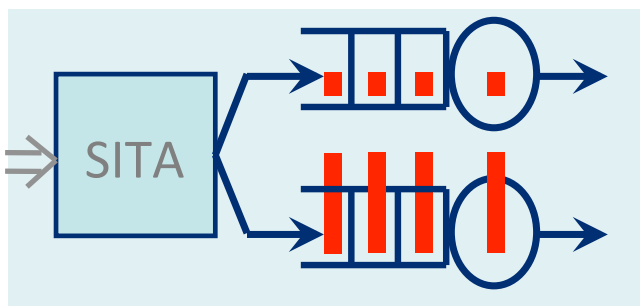
- All hosts identical.
- Jobs i.i.d. with highly variable size distrib.

[Harchol-Balter, Crovella, Murta JPDC 99], [Harchol-Balter JACM 02], [Harchol-Balter,Scheller-Wolf,Young SIGMETRICS 09]

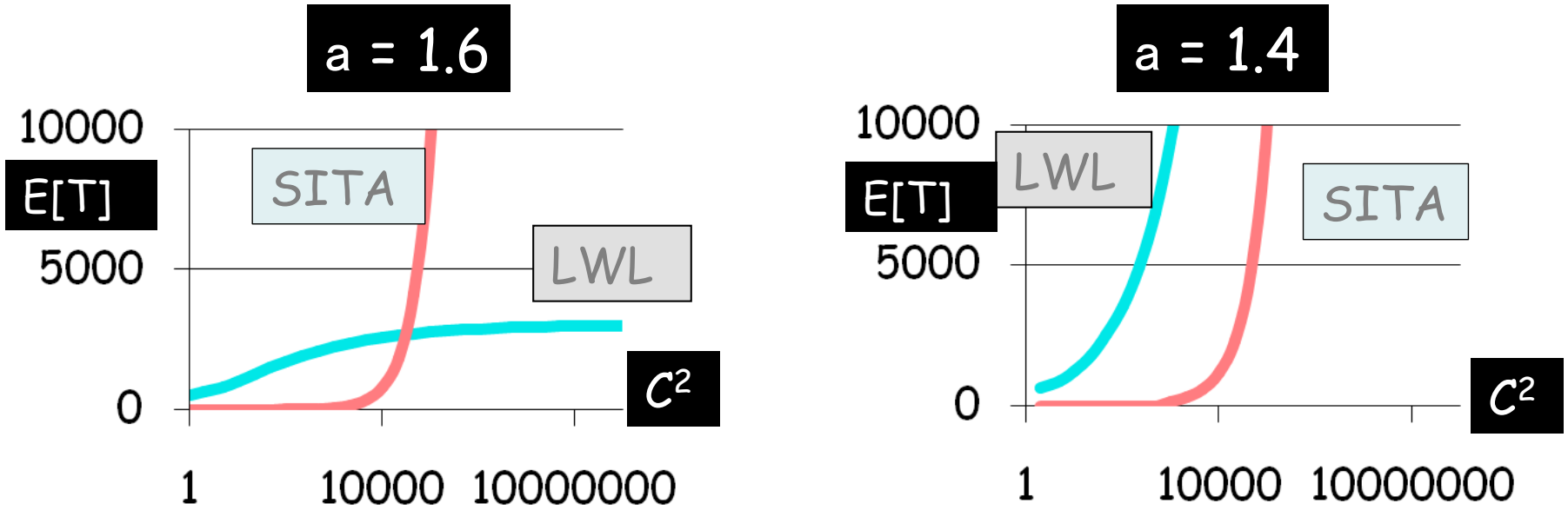
Performance ranking with FCFS servers – deceptive!

- Let us consider LWL vs SITA for fixed mean job size and growing SCOV (mathematically, in the limit for SCOV $\to \infty$).
- QUESTIONS:
  1. SITA diverges & LWL diverges?
  2. SITA converges & LWL diverges?
  3. SITA diverges & LWL converges?
  4. SITA converges & LWL converges?
- ANSWER: all of the above!





LWL vs SITA



Service time PDF: Bounded Pareto with exponent a.
Mean offered load $A = 0.95$.

FCFS server farms – summary

- High variability service times: no winner between SITA and LWL.
- Round Robin, RANDOM, JSQ: no good for high service variability.
- How about *low* service variability? It turns out that Round Robin performs well, as well as RANDOM.
  - Think of deterministic service times: then Round Robin is the optimum.
  - In case of deterministic service times and random splitting of Poisson traffci, each server can be modeled as an $M/D/1$ queue.

PS server farms – SITA

- Let $s_i, i = 0, 1, \ldots, N$ be the thresholds, with $s_0 = 0$, $s_N = \infty$.
- If service time is such that $s_{i-1} \leq X < s_i$, the job joins queue i, for $i = 1, \ldots, N$.
- Let $p_i = \int_{s_{i-1}}^{s_i} f_X(t)dt$ and $\theta_i = \int_{s_{i-1}}^{s_i} t\, f_X(t)dt$ for $i = 1, \ldots, N$. Then $\lambda_i = N\lambda p_i$ and $\mathrm{E}[X_i] = \theta_i/p_i$.
- The load factor of queue i is $\rho_i = \lambda_i \mathrm{E}[X_i] = N\lambda\theta_i$. Note that $\rho_1 + \cdots + \rho_N = N\lambda\mathrm{E}[X] = A < N$, where the overall load level A of the system is assigned.
- The mean response time is

$$\mathrm{E}[R] = \sum_{i=1}^{N} p_i \frac{\mathrm{E}[X_i]}{1-\rho_i} = \frac{1}{N\lambda} \sum_{i=1}^{N} \frac{\rho_i}{1-\rho_i}$$

PS server farms – SITA (cont'd)

- We search for the thresholds that minimizes the mean response time, given the overall load level.

$$\text{Minimize: } \mathrm{E}[R] = \frac{1}{N\lambda} \sum_{i=1}^{N} \frac{\rho_i}{1-\rho_i}$$

$$\text{Subject to: } \sum_{i=1}^{N} \rho_i = A, \quad 0 \leq \rho_i < 1$$

- It is easy to find that the optimum is achieved when $\rho_i = \rho = A/N, \forall i$.
- In that case $\mathrm{E}[R]/\mathrm{E}[X] = \frac{1}{N-A}$.

PS server farms – ranking of policies

- It turns out that the optimized SITA is the same as RANDOM.
- LWL performs far better than RANDOM and SITA, but it still departs significantly from optimality under increasing SCOV of service times.
- JSQ turns out to be almost insensitive to job size variability and very close to OPT.
- the OPT policy is so defined: for each incoming job, assign it to the queue such that the response time of *all* jobs currently in the system is minimized, *under the assumption of no future arrival*.

Join Below Threshold (JBT)

Target: minimize delay = dispatching + response

- Kelly and Laws (1993) proposed **Join Below Threshold(r) – JBT(r)** policy:
  - A table is available at the dispatcher, reporting whether a server has less than r on-going tasks (1 bit per server).
  - A task joins randomly any server below the threshold r, if any;
  - otherwise it selects a random server and joins it.
- Initiated by server: *zero* dispatching time.
- Low message overhead: at most *one* message per arriving task.

Join Below Threshold (JBT) optimality

- Kelly and Laws prove that JBT(r) is delay-optimal in heavy traffic for 2 servers, Poisson arrivals and negative exponential task holding times, provided that

$$r \geq K \cdot \log \mathrm{E}[\#\text{ of tasks in the system}]$$

- They *conjecture* the result might hold more generally.
- It has recently been proven that the conjecture holds true for general arrivals, general service times (discrete-time model).
- Practical Implications:
  - Allows design of simple load balancing schemes with zero dispatching time and low message overhead.
  - Provides design guidelines for practical systems.

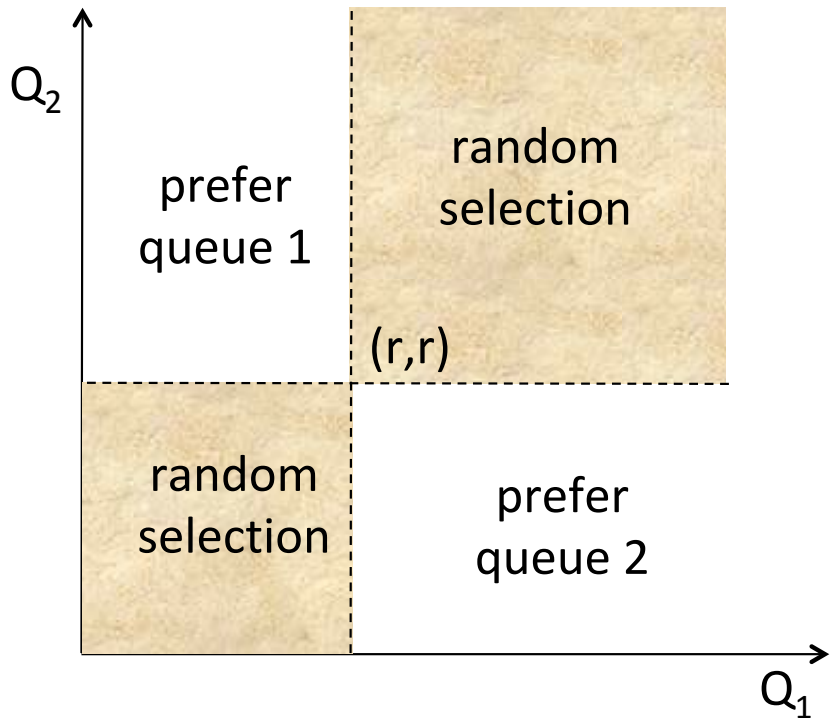
Generalized JBT(r)

- The memory at the dispatcher stores IDs of servers with queue lengths less than r.
- The parameter r can change adaptively with the system state (we will see later on how this can implemented).
- If the memory is non-empty, dispatcher randomly picks one of the IDs and sends the task to the corresponding server.
- Otherwise, dispatcher randomly picks a server to join

REMARKS

- JIQ is a special case of JBT(r) with constant $r = 1$.
- For heterogeneous servers, we can simply replace pure random selection by a random selection that is linearly weighted by the service rate.

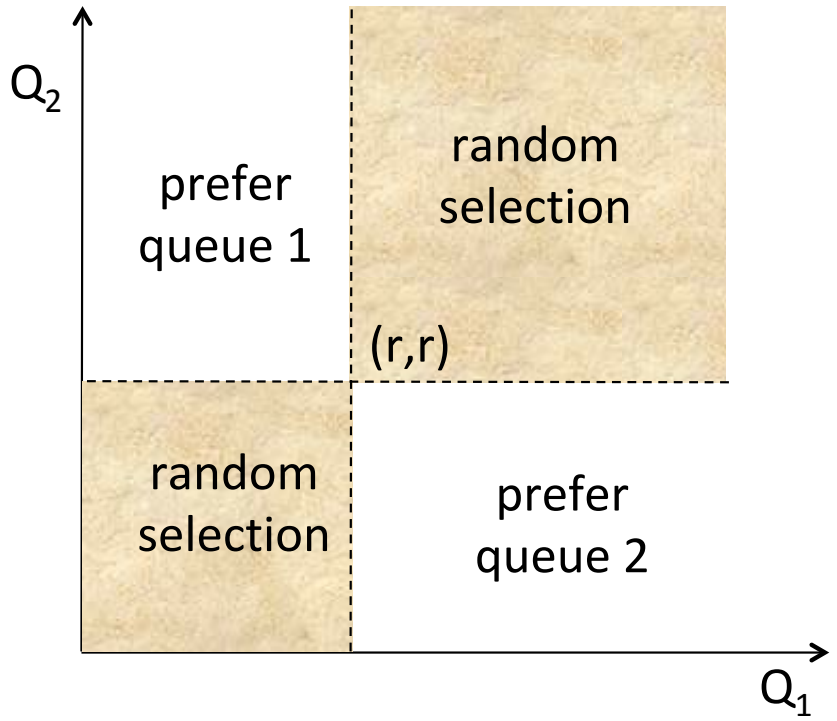
JBT(r) for two servers: a geometric view (1/2)



The queue length vector $\mathbf{Q} = [Q_1, Q_2]$ can be displayed in the non-negative quadrant of the plane.

- If $\mathbf{Q}$ falls inside the shaded areas:
  - both queue lengths are above threshold (upper shaded box); or,
  - both queue lengths are below threshold (lower shaded box).
- In either case JBT(r) reduces to random routing.

JBT(r) for two servers: a geometric view (2/2)



The queue length vector $\mathbf{Q} = [Q_1, Q_2]$ can be displayed in the non-negative quadrant of the plane.

- If $\mathbf{Q}$ falls inside the white areas preference is given to the shorter queue.
- Note that new arrivals tend to move the queue length vector towards the shaded areas.

Grow, but not too fast

Theorem

Consider a load balancing system adopting JBT(r). Let D_{pol} denote the mean delay under policy "pol".

(i) *For any constant threshold r, in* **heavy traffic**, *we have*

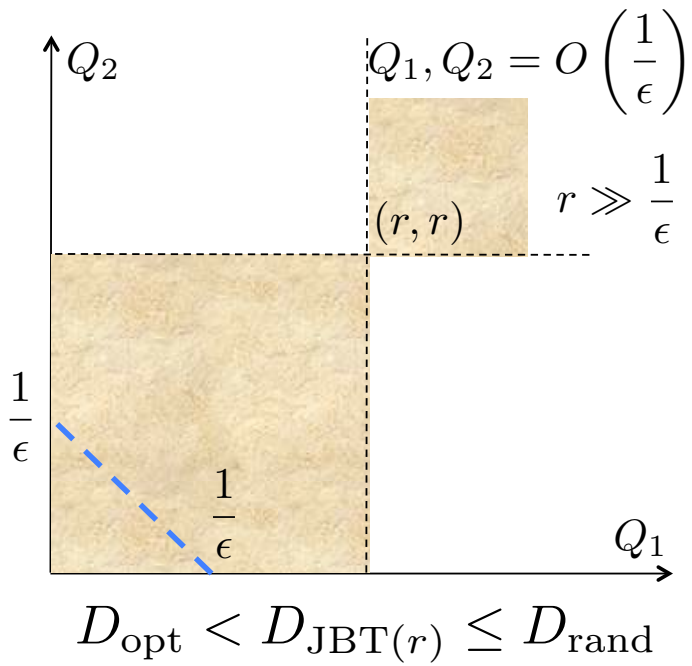
$$D_{opt} < D_{JBT(r)} < D_{rand}$$

(ii) *For $r \sim (1/\epsilon)^{1+\alpha}$ with $\alpha > 0$, in* **heavy traffic**, *we have*

$$D_{opt} < D_{JBT(r)} \leq D_{rand}$$

- JIQ (=JBT(1)) is strictly between optimal and random policy.
- r should grow with the queue length (which grows $\sim 1/\epsilon$), but not too fast.

Geometric intuition: r is too large

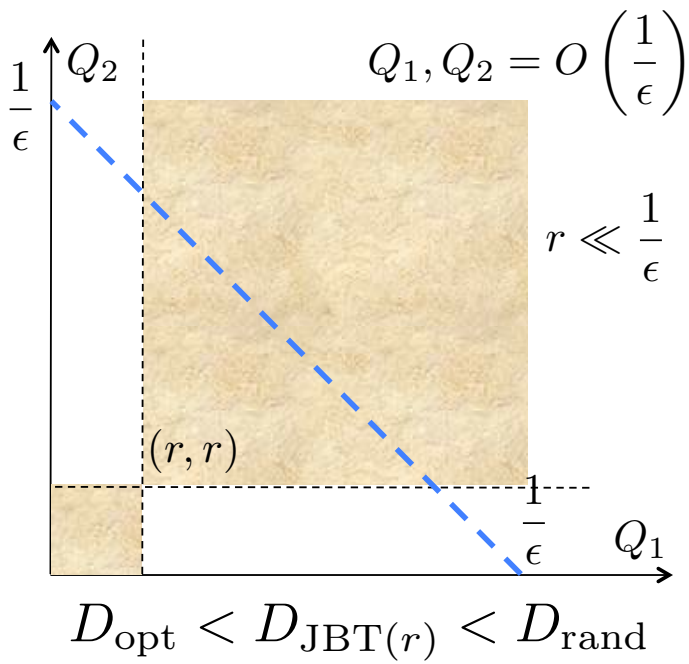


$$D_{\text{opt}} < D_{\text{JBT}(r)} \leq D_{\text{rand}}$$

In **heavy traffic** queue lengths grow as $Q_1, Q_2 \sim 1/\epsilon$, with $\epsilon = \mu - \Lambda$.

- If $r \gg 1/\epsilon$, queue lengths fall in the bottom-left corner of the lower box with high probability.
- Then JBT(r) is asymptotically identical to random routing in **heavy traffic**.

Geometric intuition: r is too small



In **heavy traffic** queue lengths grow as $Q_1, Q_2 \sim 1/\epsilon$, with $\epsilon = \mu - \Lambda$.

- If $r \ll 1/\epsilon$, queue lengths have a positive probability to fall outside the shaded area.
- Therefore JBT(r) mean delay is strictly less than random, but also strictly not optimal.

Sufficient condition for delay otpimality

Theorem (Zhou, Tan, Shroff, 2019)

Consider a load balancing system adopting JBT(r). If $r \geq K \cdot \log(1/\epsilon)$ *and* $r = o(1/\epsilon)$ *for some positive constant* K*, then JBT(r) is heavy-traffic delay optimal.*

- Holds for general (light-tailed) i.i.d arrivals and service times (time slotted queueing model).
- Holds for any number of servers.
- Optimality is in steady-state.
- Solves a generalized version of the long-standing conjecture put forward by Kelly and Laws in 1978.

JBT-d property

Theorem (Shroff et alii, 2018, 2019)

For any finite T and $d \geq 1$, the JBT-d policy is throughput and delay optimal in heavy traffic.

- JBT-d can be generalized to the case of unequal servers.
- Each server reports its serving rate when sending a message to the dispatcher.
- The dispatcher assigns tasks according to the following weights, where $\mathcal{L}(t)$ is the ID list at time t:

$$\phi_j = \frac{\mu_j}{\sum_{k \in \mathcal{L}(t)} \mu_k}, \quad |\mathcal{L}(t)| > 0; \qquad \phi_j = \frac{\mu_j}{\mu}, \quad |\mathcal{L}(t)| = 0$$

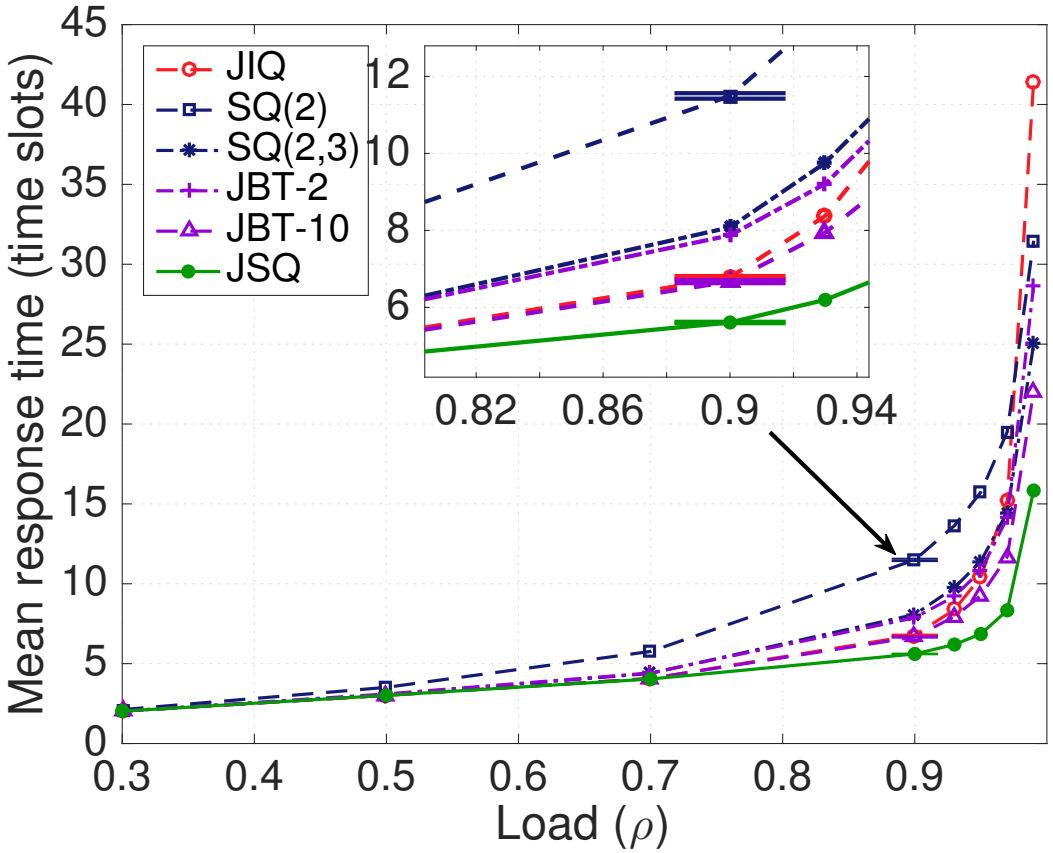
Policy comparison

Policy	**Msg/task**	**TO-Ho**	**TO-He**	**DO-Ho**	**DO-He**
Random	0	✓	×	×	×
JSQ	$2N$	✓	✓	✓	✓
Pod	$2d$	✓	×	✓	×
JIQ	≤ 1	✓	×	×	×
JBT-d	$\leq 1 + \frac{N+2d}{T}$	✓	×	✓	×
JBTG-d	$\leq 1 + \frac{N+2d}{T}$	✓	✓	✓	✓

TO/DO = Throughput / Heavy-traffic Delay Optimal
Ho / He = Homogeneous / Heterogeneous servers.

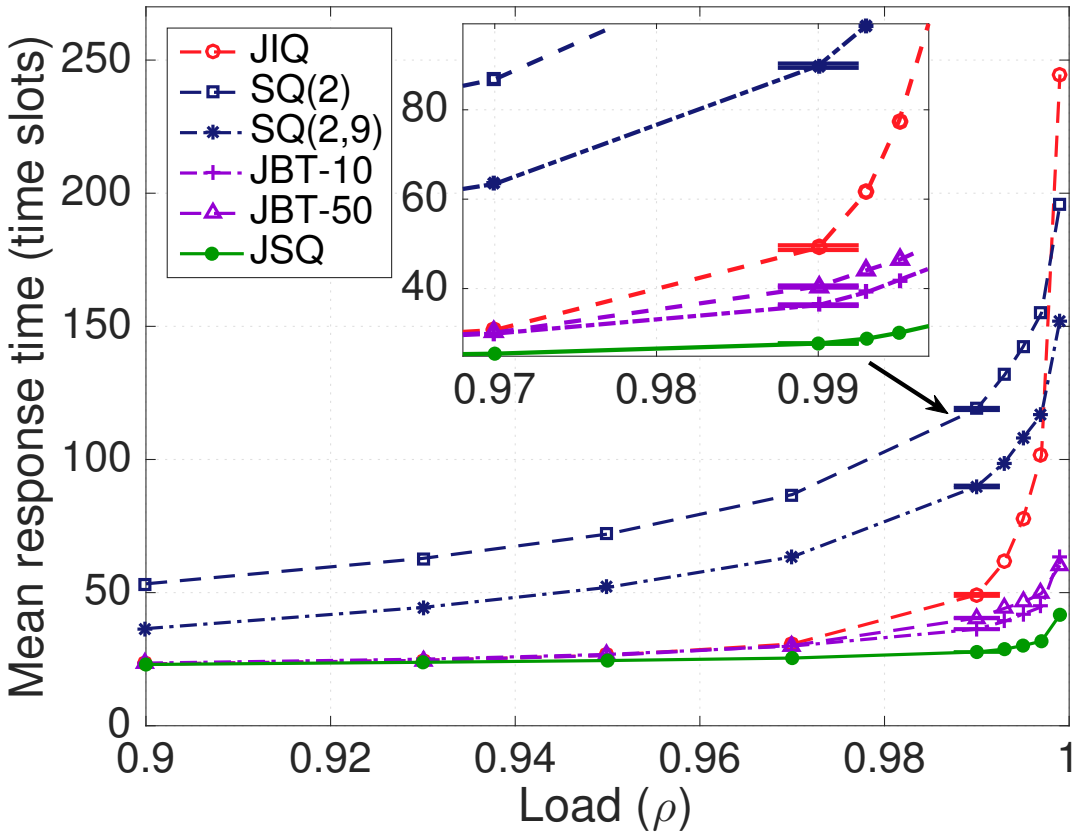
PoD is also known as SQ(d). It is generalized to SQ(d,m), i.e., Pod with memory m (take the minimum over the last m polls).

Numerical results (1/3)



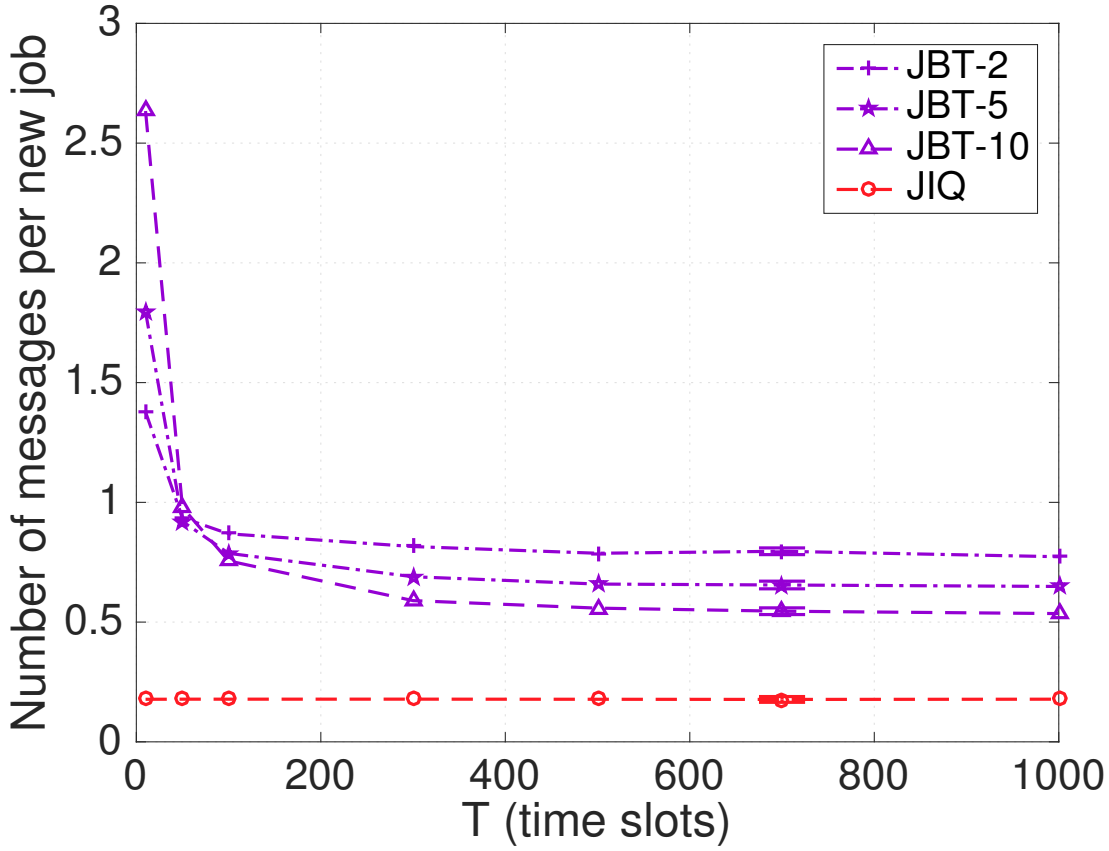
Delay performance under 10 homogeneous servers.

Numerical results (2/3)



Heavy-traffic delay performance under 50 homogeneous servers.

Numerical results (3/3)



Message per new job arrival under 10 homogeneous servers.

Challenge #2: load balancing

Definitions

- Consider a cluster of 2 servers to which a flow of jobs is offered through a dispatcher.
- Jobs arrive at the dispatcher according to a Poisson process with mean rate Λ.
- Job workload is distributed as a Pareto random variable L with Complementary Cumulative Distribution Function (CCDF):

$$G_L(x) = \mathcal{P}(L > x) = \left(\frac{b}{x}\right)^{\alpha}, \quad x \geq b.$$

- Task processing time at server j is $X_j = L/\mu_j$, $j = 1, 2$, where μ_1 and μ_2 are processing rates of the two servers.
- Servers use FCFS scheduling.

Formulas and numerical values

- The first two moments of L are:

$$\mathrm{E}[L] = \frac{\alpha b}{\alpha - 1} \qquad \mathrm{E}[L^2] = \frac{\alpha b^2}{\alpha - 2}$$

- The system load is $\rho = \frac{\Lambda \mathrm{E}[L]}{\mu_1 + \mu_2}$.
- Numerical values are as follows:
  1. $0.5 \leq \rho \leq 0.9$
  2. $\mu_2 = 10$ and $\mu_1 = 1$.
  3. $\alpha = 2.01, 2.05, 2.25$ (three different values) and $\mathrm{E}[L] = 1$.
- Hint: the mean system time (aka response time) of the $M/G/1$ queue (a queue with single server, general service times X and Poisson arrivals with mean rate λ) is:

$$\mathrm{E}[S] = \mathrm{E}[X] + \frac{\lambda \mathrm{E}[X^2]}{2(1 - \lambda \mathrm{E}[X])}$$

Assignment

1. Consider SITA policy. With a given *workload* threshold θ, state a queueing model of the two server cluster.
   1. Identify the arrival processes at queue 1 and 2. Motivate your models.
   2. Identify the service time probability distributions at server 1 and server 2.
   3. Write the expression of the mean delay $\mathrm{E}[D]$ through the two server cluster.
2. State the optimization problem to find the workload threshold θ^* that minimizes $\mathrm{E}[D]$. Solve the problem numerically.
3. With $\theta = \theta^*$, plot $\mathrm{E}[D]$ as a function of ρ.
4. Repeat points 1-3 above assuming random routing, where an arriving task is assigned to server 1 with probability p, to server 2 with probability $1 - p$. Set $p = p^*$ to minimize $\mathrm{E}[D]$.

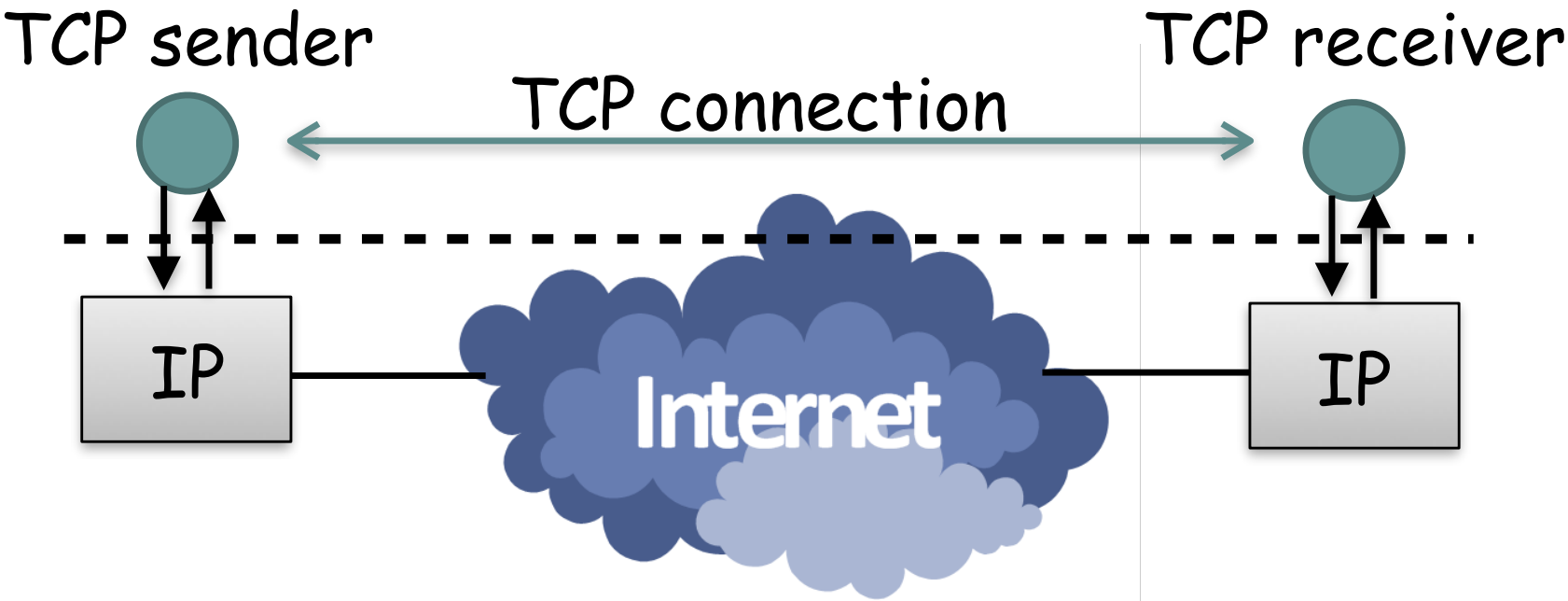
Delivery of the assignment

The delivery of the assignment consists of a **2 pages written report** (Font size: 12 pt). Remember to put your given and family names, and enrollment number as a header on top of each page.

1. **PAGE 1** - (i) Statement and discussion of system queueing model in case of SITA; (ii) Statement of the optimization problem to find θ^*; (iii) Statement and discussion of system queueing model in case of RANDOM; (iv) Statement of the optimization problem to find p^*.
2. **PAGE 2** (i) Plots of θ^* and p^* (two separate plots) as a function of ρ for the three considered values of α; (ii) Plots of $\mathrm{E}[D]$ as a function of ρ for SITA and RANDOM for the three considered values of α (make three different plots, one for each value of α). Scale the plots axes so as to make comparison between the two policies as readable as possible.

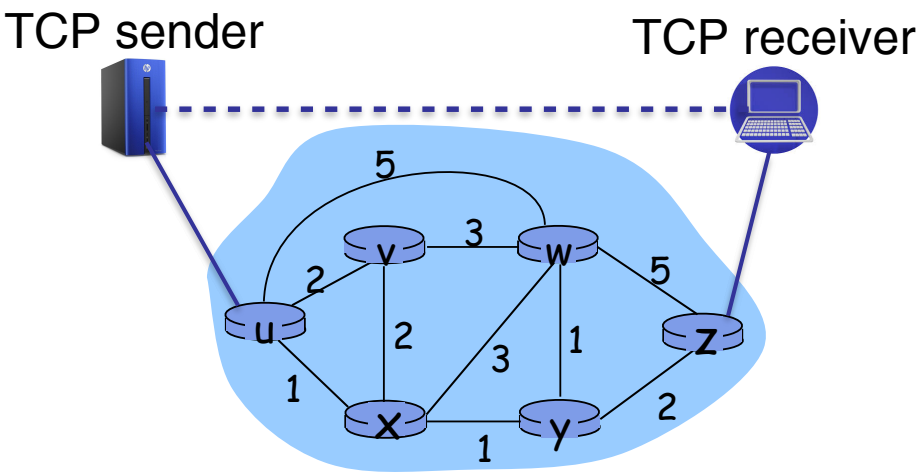
Congestion control

TCP scenario



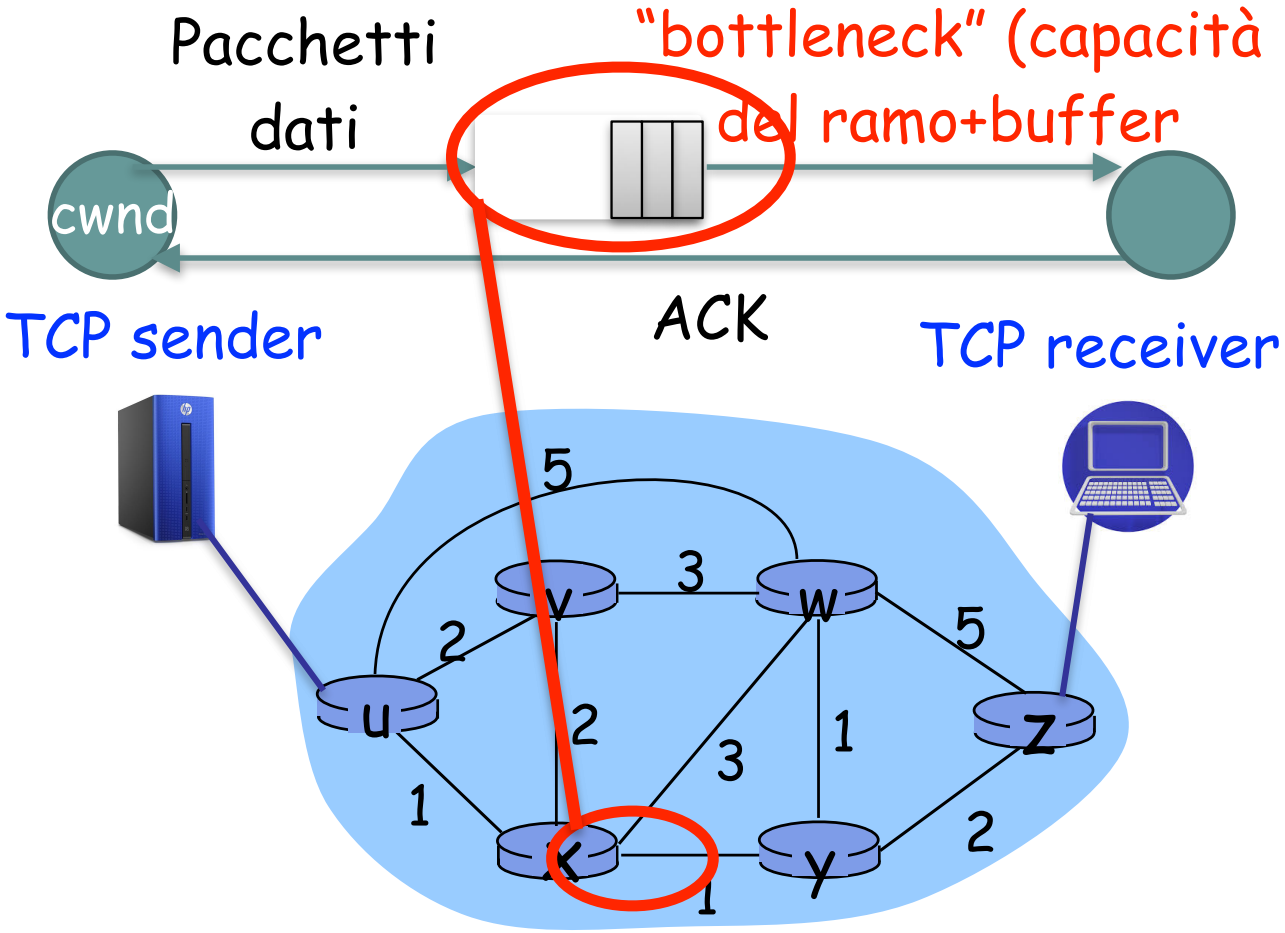
Congestion control and network optimization

- **Congestion control**: protection of node and link resources from overload.
  - Carried out by traffic sources. Implemented in the Transmission Control Protocol (TCP).
- **Routing**: selection of optimal network path (minimum cost.
  - Carried out by network nodes, using routing protocols (e.g., OSPF-Dijkstra). Based on the Internet Protocol (IP).



Rete -> grafo pesato

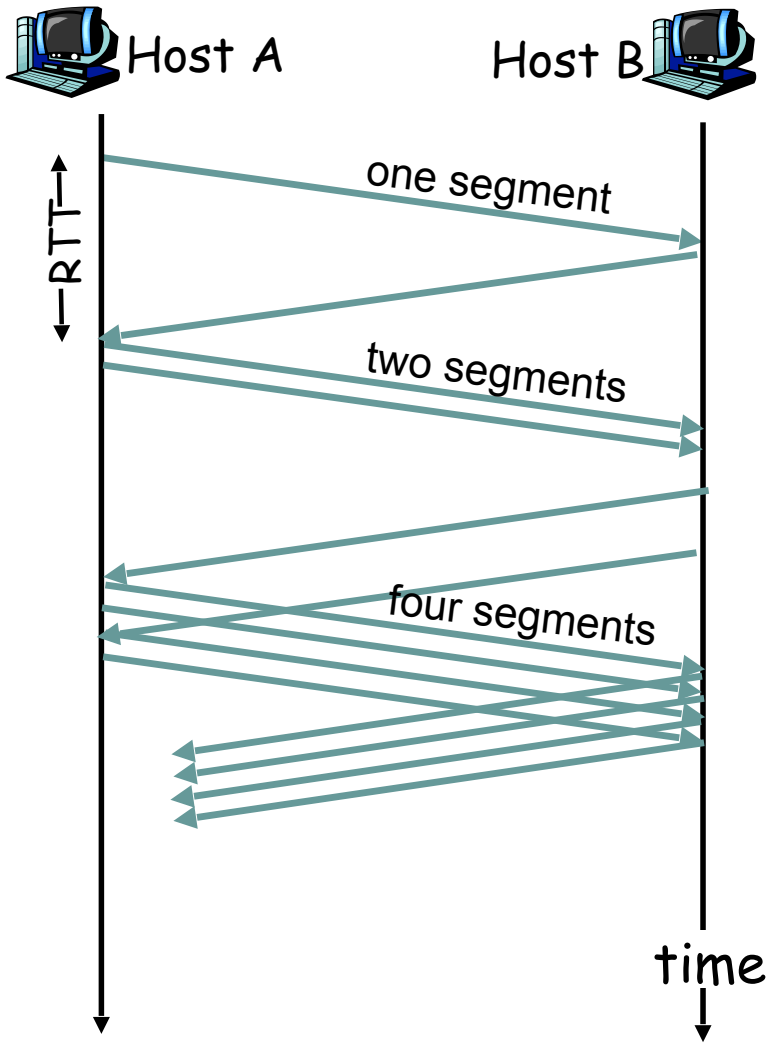
"Single bottleneck" model of TCP



TCP congestion control in a nutshell

- In flight data limited by $W = \min\{\texttt{rwnd}, MSS \cdot \texttt{cwnd}\}$.
- The `cwnd` is calculated by the sender, according to the cc state machine.
  - The initial state is Slow Start and `cwnd` is initialized at IW.
  - $\texttt{cwnd} \leftarrow \texttt{cwnd} + 1$ for each received 'good' ACK in Slow Start state.
  - When `cwnd` exceeds `ssthresh`, the state moves to Congestion Avoidance (CA); then $\texttt{cwnd} \leftarrow \texttt{cwnd} + 1/\texttt{cwnd}$ for each received 'good' ACK.
  - On packet loss, $\texttt{cwnd} \leftarrow (1 - \beta)\texttt{cwnd}$ ($0 < \beta < 1$, multiplicative decrease). If packet loss was detected through *duplicated ACKs*, the CA state is resumed after loss recovery.

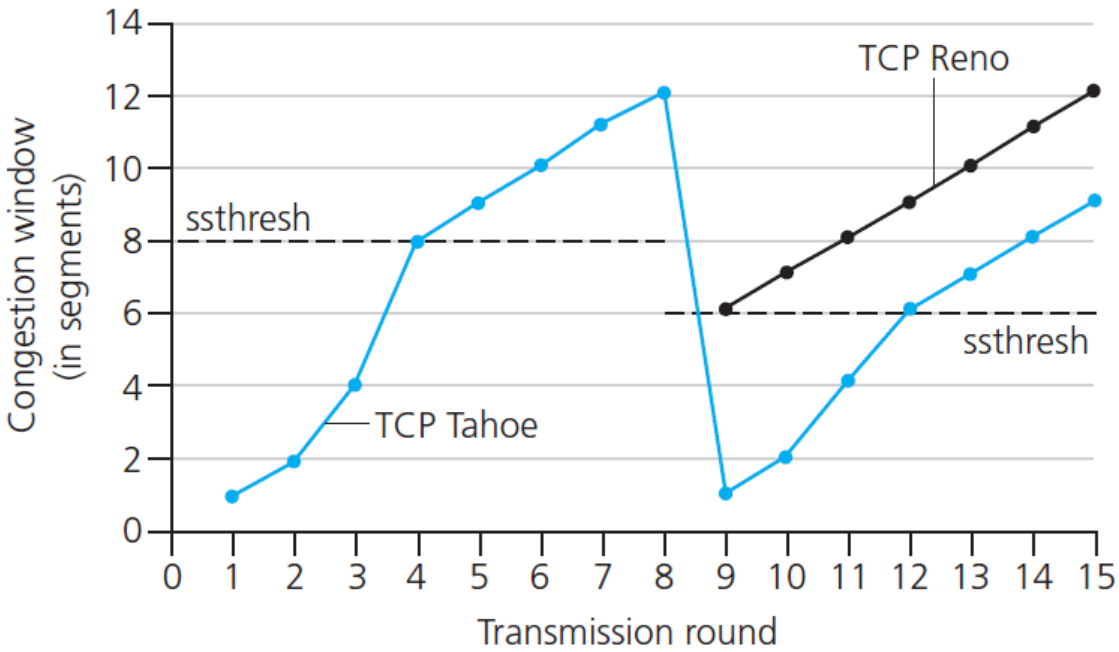
Slow Start



- Slow start is a fast probing state.
- `cwnd` increments by 1 per ACK until it overcomes `ssthresh` or a packet loss is detected or the connection ends up.

Congestion Avoidance

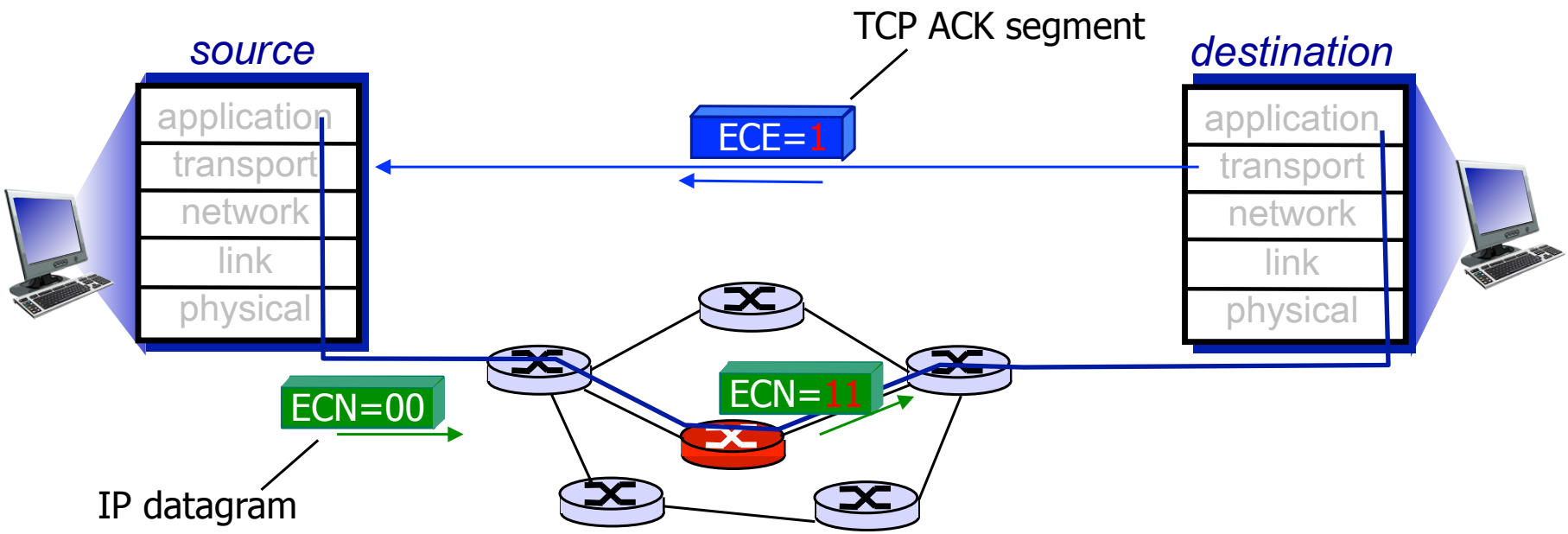
Classic paradigm: TCP (New)Reno



- Congestion Avoidance aims at a prudent probing.
- `cwnd` increments by 1 per a window worth of ACKs until it a packet loss is detected or a timeout occurs or the connection ends up.

Explicit Congestion Notification (ECN)

- Two bits of the ToS field of IPv4 header are devoted to signaling the ECN capability and the Congestion Experienced (CE) flag.
- The CE flag is echoed by the TCP receiver by using the ECN Echo (ECE) flag in the TCP ACK header.



DCTCP approach

- DCTCP uses a marking scheme at switches that sets the CE codepoint of packets as soon as the buffer occupancy exceeds a fixed threshold.
- The DCTCP source reacts by reducing the window by a factor that depends on the fraction of marked packets: the larger the fraction, the bigger the decrease factor.
  - The idea of reacting in proportion to the extent of congestion is also used by delay-based congestion control algorithms.
  - At very high data rates and with low-latency network fabrics, sensing the queue buildup in shallow-buffered switches can be extremely noisy.
  - For example, a 10 packet backlog constitutes 120 μs of queuing delay at 1 *Gbps*, and only 12 μs at 10 *Gbps*.

DCTCP components

- **Simple Marking at the Switch**: an arriving packet is marked with the CE codepoint if the queue occupancy is greater than K upon its arrival.
- **ECN-Echo at the Receiver**: every packet is ACK'ed, setting the ECN-Echo flag if and only if the packet has a marked CE codepoint. Provision is made to account for delayed ACKs.
- **Controller at the Sender**: The sender maintains an estimate of the average fraction α of packets that are marked:

$$\alpha \leftarrow (1-g)\alpha + gF$$

  - α is updated once for every window of data ($\approx$ one RTT).
  - F is the fraction of packets marked in the last window of data.
  - At steady state, α converges to $\mathrm{E}[F]$.

DCTCP `cwnd` updating

- Slow Start: $\texttt{cwnd} \leftarrow \texttt{cwnd} + 1$ per ACK.
- Congestion Avoidance: $\texttt{cwnd} \leftarrow \texttt{cwnd} + 1/\texttt{cwnd}$ per ACK.
- Upon receiving marked ACK(s) in a window of data, the TCP sender reduces the `cwnd`:

$$\texttt{cwnd} \leftarrow \texttt{cwnd}\left(1 - \frac{\alpha}{2}\right)$$

  - When α is near 0 (low congestion), the window is only slightly reduced.
  - When congestion is high ($\alpha = 1$), DCTCP cuts window in half, just like TCP.

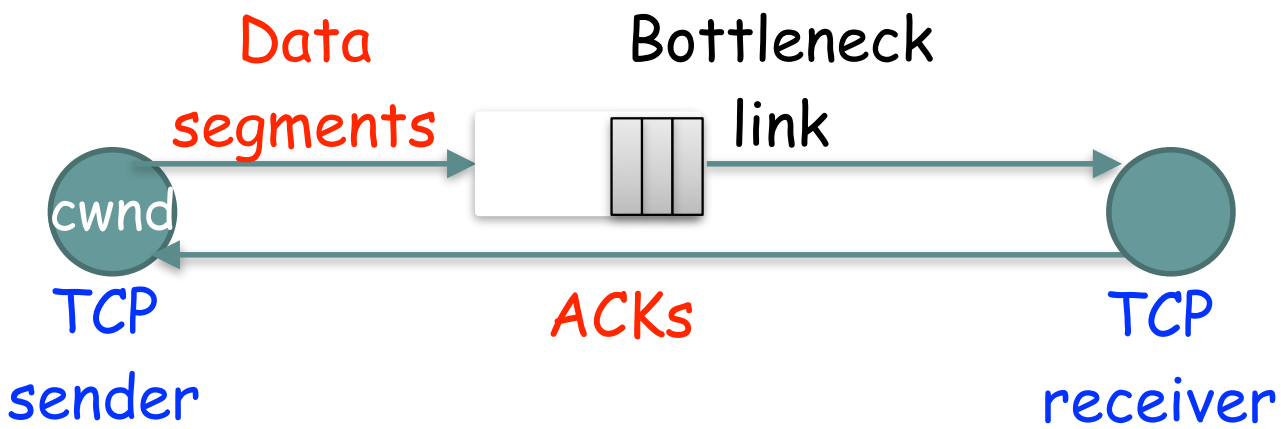
Single-bottleneck TCP connection model

C bottleneck link capacity.

T base round-trip time, RTT_{base}.

$W(t)$ size of `cwnd` at time t.

$Q(t)$ amount of data in the bottleneck buffer at time t.



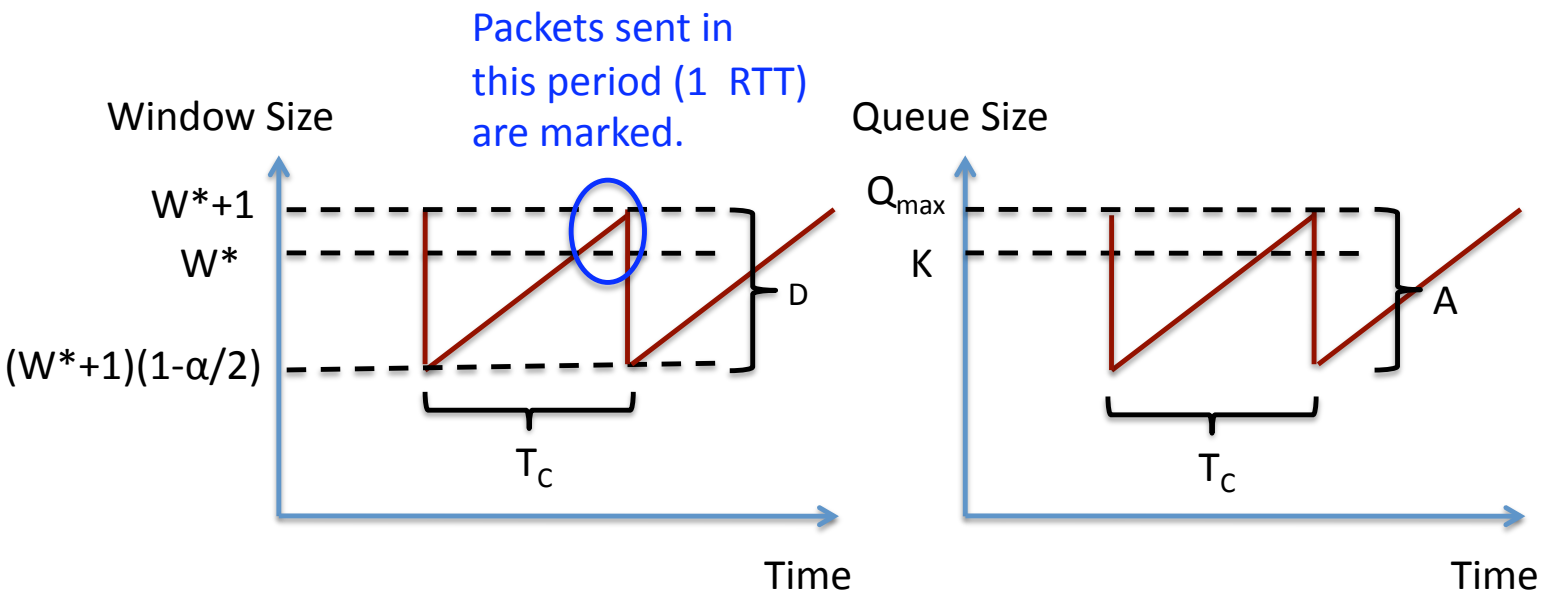
DCTCP analysis: assumptions

- N infinitely long-lived flows.
- Only Congestion Avoidance is considered.
- Identical base round-trip times: $RTT_{base} = T$.
- Single, shared bottleneck link of capacity C.
- The N flows are synchronized; i.e., their "sawtooth" window dynamics are in-phase.
  - This assumption is only realistic when N is small.
  - This is the case we care about most in data centers.

$W_j(t)$ for each flow $j =, 1 \ldots, N$, and $Q(t)$ are periodic functions of t.

Derivation of α: key observation

- With synchronized senders, the queue size exceeds the marking threshold K for exactly one RTT in each period.
- We can compute the fraction of marked packets α by simply dividing the number of packets sent during the last RTT of the period by the total number of packets sent during a full period of the sawtooth.



Derivation of α: calculation

- The buffer content at time t is $Q(t) = NW(t) - CT$.
- The critical `cwnd` size that triggers packet marking is when $Q(t) = K$, i.e., $W^* = (CT + K)/N$.
- In the last RTT, `cwnd` grows from W^* to $W^* + 1$, then it is reduced to $(W^* + 1)(1 - \alpha/2)$. Hence

$$\alpha = \frac{S(W^*, W^* + 1)}{S((W^* + 1)(1 - \alpha/2), W^* + 1)}$$

where $S(a, b)$ is the number of TCP data segments sent when `cwnd` grows from a to b.

Derivation of α: calculation (cont'd)

- From $S(a,b) = \int_a^b wow = (b^2 - a^2)/2$ (fluid model), we get:

$$\alpha^2 \left(1 - \frac{\alpha}{4}\right) = \frac{2W^* + 1}{(W^* + 1)^2} \approx \frac{2}{W^*}$$

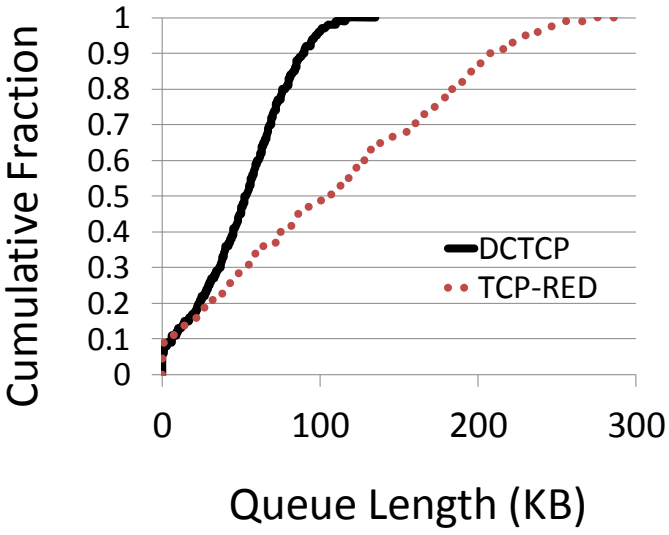
- For $W^* \gg 1 \Rightarrow N \ll CT + K$ we have $\alpha \approx \sqrt{2/W^*}$. The maximum and minimum levels of the bottleneck buffer are:

$$Q_{max} = N(W^* + 1) - CT = N + K$$

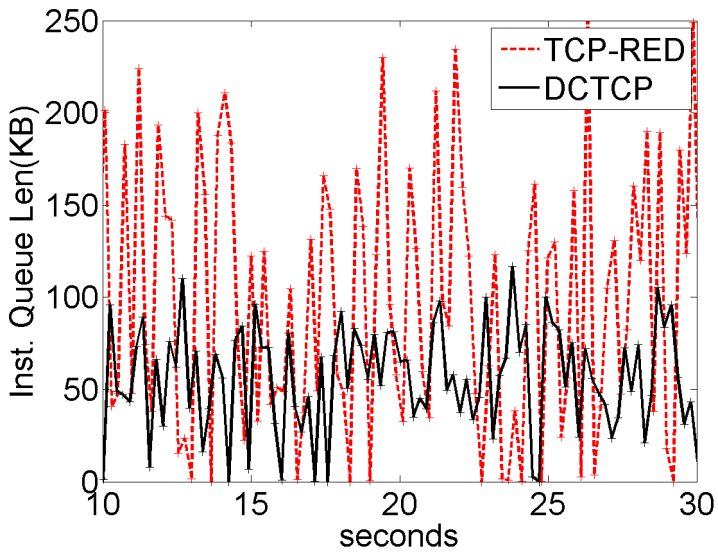
$$Q_{min} = N(W^* + 1)(1 - \alpha/2) - CT \approx N + K - \sqrt{\frac{N(CT + K)}{2}}$$

- Minimizing Q_{min} with respect to N and requiring that $Q_{min} > 0$, we find that it must be $K > CT/7$.

DCTCP performance vs TCP+RED



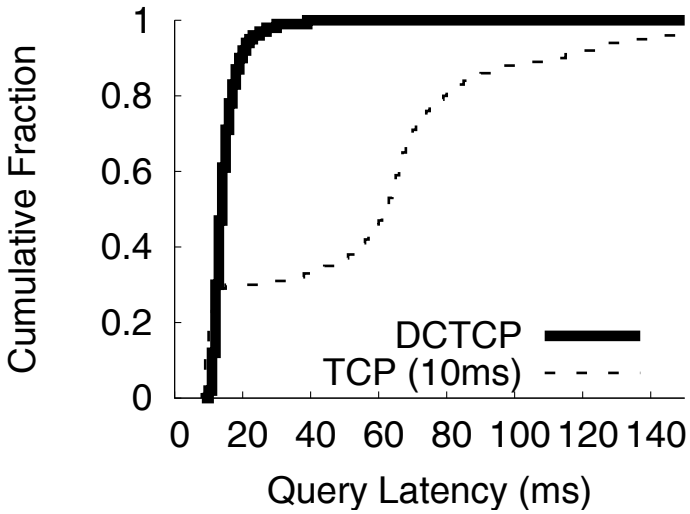
(a) CDF of queue length



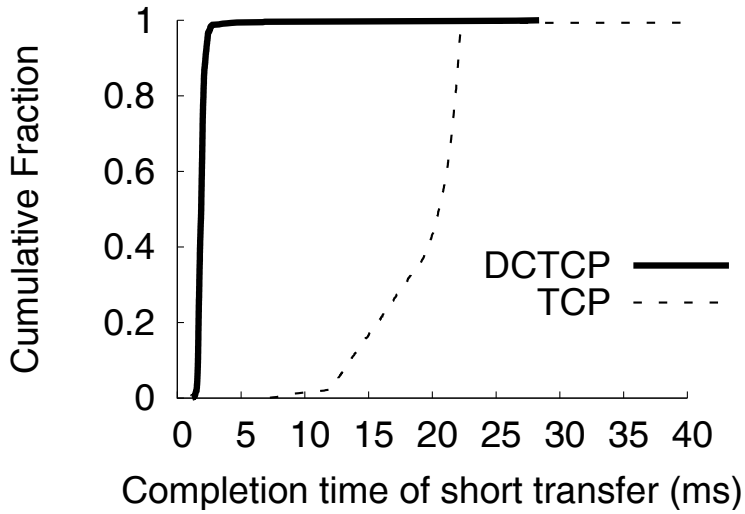
(b) Time series of queue length

DCTCP incast performance (all-to-all traffic)

Dynamic buffering



Short file transfers.



Dynamic behavior of DCTCP: fluid model

$$\frac{dW}{dt} = \frac{1}{R(t)} - \frac{W(t)\alpha(t)}{2R(t)} p(t - T^*) \tag{4}$$

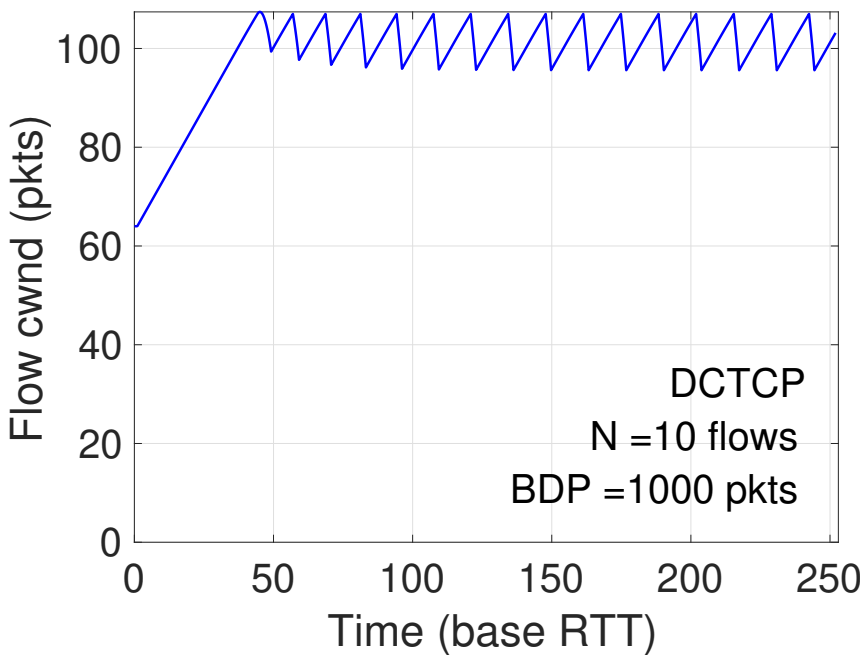
$$\frac{d\alpha}{dt} = \frac{g}{R(t)} [p(t - T^*) - \alpha(t)] \tag{5}$$

$$\frac{dQ}{dt} = \begin{cases} \max\left\{0, \frac{NW(t)}{R(t)} - C\right\} & Q(t) = 0, \\ \frac{NW(t)}{R(t)} - C & 0 < Q(t) < B, \\ \min\left\{0, \frac{NW(t)}{R(t)} - C\right\} & Q(t) = B. \end{cases} \tag{6}$$

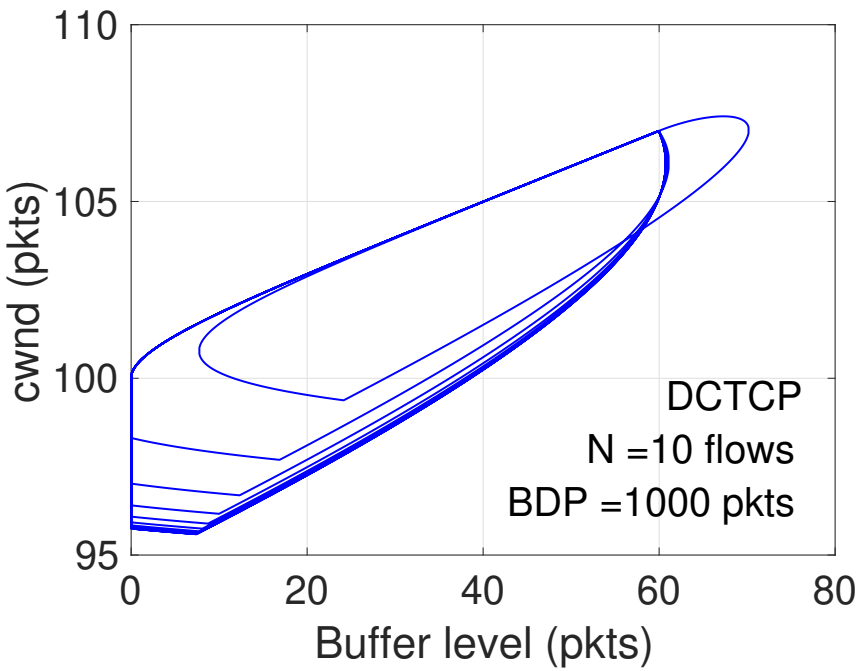
with $R(t) = T + Q(t)/C$, $T^* = T + K/C$ and $p(t) = I(Q(t) > K)$.

DCTCP incast performance (all-to-all traffic)

Congestion window time evolution.



Limit cycle of state trajectory in the $Q - W$ plane



Congestion control: switched Ethernet vs. the Internet

- No per-packet acks in Ethernet.
  - Path delays (round trip times) are not knowable.
  - Congestion must be signaled by switches directly to sources.
- Packets may not be dropped.
- No packet sequence numbers.
- Sources start at the line rate.
- Very shallow buffers.
  - Ethernet switch buffers are typically 100s of kBytes deep.
- Small number-of-sources regime is typical.
- Multipathing.

Congestion Notification (CN) in Ethernet

- Provides a means for a switch to notify a source of congestion causing the source to reduce the flow rate.
- Targeted at:
  - Networks with low delay: e.g., data center.
  - Long lived data flows.
- Goals:
  - Avoid frame loss.
  - Reduce latency.
- Operates on frames in a VLAN priority.
  - Allows for sharing the network between congestion controlled and non-controlled traffic.

Relationship with Priority Flow Control

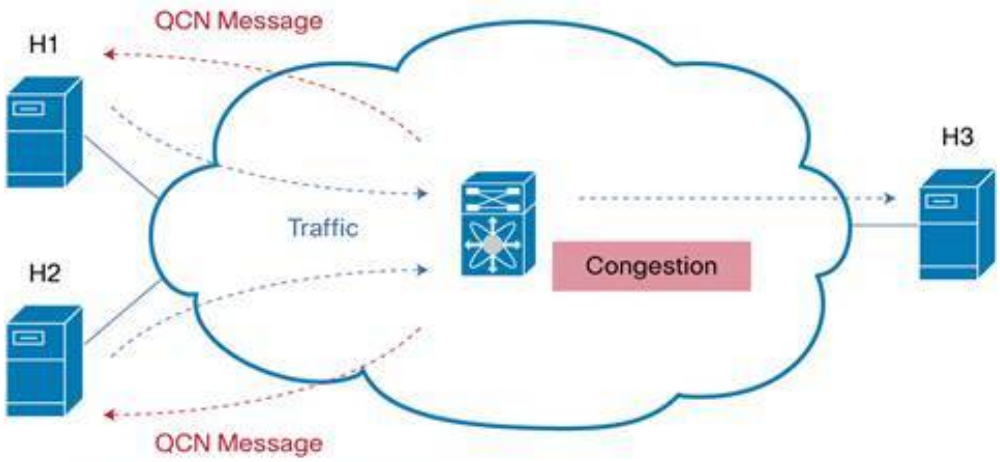
- **Priority Flow Control** (IEEE 802.1Qbb).
  - Hop-by-hop, per-priority pausing of traffic at congested links.
  - Issues a PAUSE message to upstream buffers, when the buffer at a congested link fills up,
  - Ensures packets do not get dropped due to congestion.
- **Congestion spreading**
  - A consequence of link-level pausing.
  - Domino effect of buffer congestion propagating upstream causing secondary bottlenecks.
  - Secondary bottlenecks are highly undesirable as they affect sources whose packets do not pass through the primary bottleneck.
- An L2 congestion control scheme allows a primary bottleneck to directly reduce the rates of those sources whose packets pass through it.

CN objectives

- Independent of upper layer protocol.
- Coexisting with TCP.
- Designed for unicast traffic.
- Performance: stable, responsive, fair.
- Simple: implemented in hw.
  - No per flow state or per flow queuing in switchs.
  - No complicated calculations of rates/parameters.

Reactive congestion control in Ethernet networks

- The Quantized Congestion Notification (QCN) is defined in the IEEE 802.1Qau standard (now incorporated into IEEE 802.1Q-2011) and provides a reactive congestion control mechanism for Ethernet switches at layer 2.
- When the switch buffer queue exceeds a prescribed level of congestion, a feedback frame is sent to the traffic sources to reduce their sending rate.



QCN elements

- **Congestion Point (CP)**
  - It detects a congestion by monitoring the switch queue length.
  - The aim is to prevent the queue length from exceeding the queue threshold Qeq.
  - **Algorithm**: a congested switch samples outgoing frames and generates a feedback message (Congestion Notification Message or CNM) to the source of the sampled frame with information about the extent of congestion at the CP.
- **Reaction point (RP)**
  - It is associated with a source to adjust the sending flow rate.
  - **Algorithm**: a Rate Limiter (RL) associated with a source decreases its sending rate based on feedback received from the CP, and increases its rate unilaterally (without further feedback) to recover lost bandwidth and probe for extra available bandwidth.

Congestion Notification Message (CNM)

- Version - 4 bit.
- Quantized Feedback - 6 bits
  - A function of cnmQoffset and cnmQDelta.
- Congestion Point Identifier (CPID) - 8 byte (*).
- cnmQoffset, units of 64 bytes - 2 byte (*).
- cnmQDelta, units of 64 bytes - 2 byte (*).
- Encapsulated priority (priority of the sampled frame) - 3 bit.
- Encapsulated destination MAC address (DA of sampled frame) - 6 byte.
- Encapsulated MSDU length - 2 byte (max value 64).
- Encapsulated MSDU - up to 64 bytes.

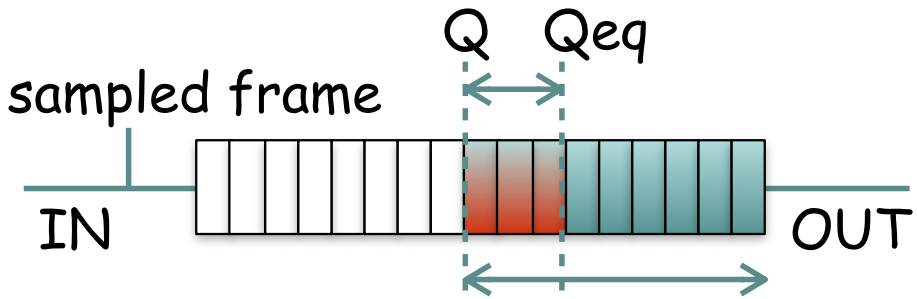
(*) Not used by reaction point algorithm

Congestion Notification Tag

- An end station may add a Congestion Notification Tag (CN-TAG) to every frame it transmits from a Congestion Controlled Flow (e.g., same src/dst MAC + priority).
- CN-TAG contains a Flow Identifier (Flow ID) field.
  - Flow ID is 2 byte long.
  - Flow ID meaning is local to source.
- The destination_address, Flow ID, and a portion of the frame that triggered the transmission of the CNM are the means by which a station can determine to which RP a CNM applies.

CP algorithm (1/2)

CP thought of as an output buffered switch.



The goal of the CP is to maintain the buffer occupancy at a desired operating point, Q_{eq}.

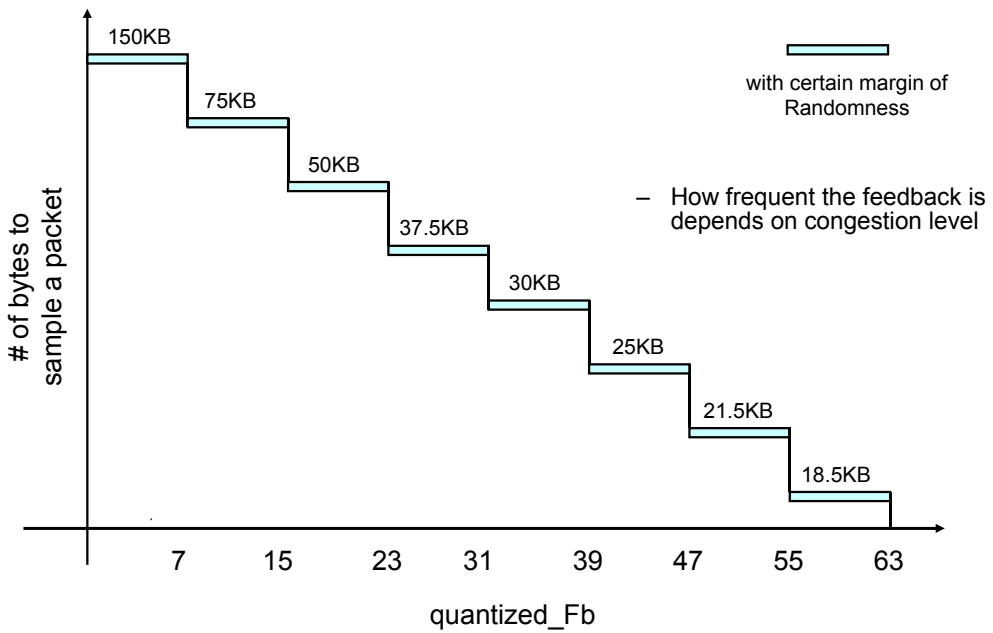
Calculated congestion measure (quantized to 6 bits):

$$F_b = -(Q - Q_{eq}) - w(Q - Q_{old})$$

$w = 2$ by default

CP algorithm (2/2)

- A random output packet is sampled.
- Time to sample is defined according to congestion level $|F_b|$.
- F_b is calculated at sampling time; the feedback is sent to the source of the sampled packet only if $F_b < 0$.



Randomness is 30%.

RP algorithm (1/3)

- No positive feedback → needs a mechanism for increasing its sending rate on its own.
- The increases of rate is clocked internally at the RP.

Current Rate (CR) : The transmission rate of the RL at any time.

Target Rate (TR) : The sending rate of the RL just before the arrival of the last feedback message.

Byte Counter (BC) : counts the number of bytes transmitted by the RL at the RP. Used for timing rate increases at the RL.

Timer (TI) : A clock at the RP used for timing rate increases at the RL.

RP algorithm (2/3)

- **Rate decrease**: upon reception of a feedback message.

$$TR \leftarrow CR \qquad CR \leftarrow CR(1 - G_d|F_b|)$$

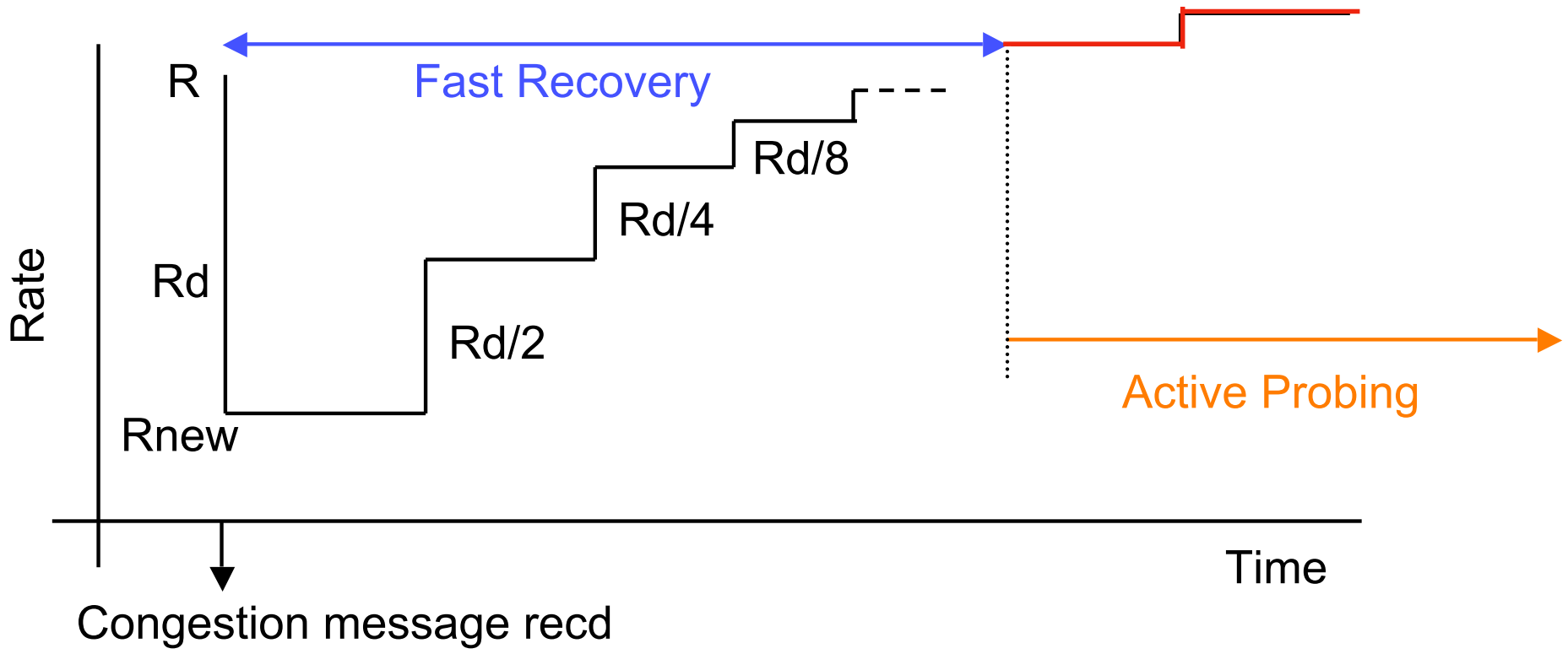
  - It is $G_d|F_{bmax}| = 1/2$.
  - BC and TI are reset.

- **Fast Recovery (FR)**: 5 cycles triggered by TI or BC

$$CR \leftarrow \frac{1}{2}(CR + TR)$$

  - TI defaults to $T = 10$ ms.
  - BC defaults to $B = 150$ kBytes of data transmitted by the RL.

Example of rate recovery with RP algorithm



RP algorithm (3/3)

- **Active Increase**: the RL probes for extra bandwidth:

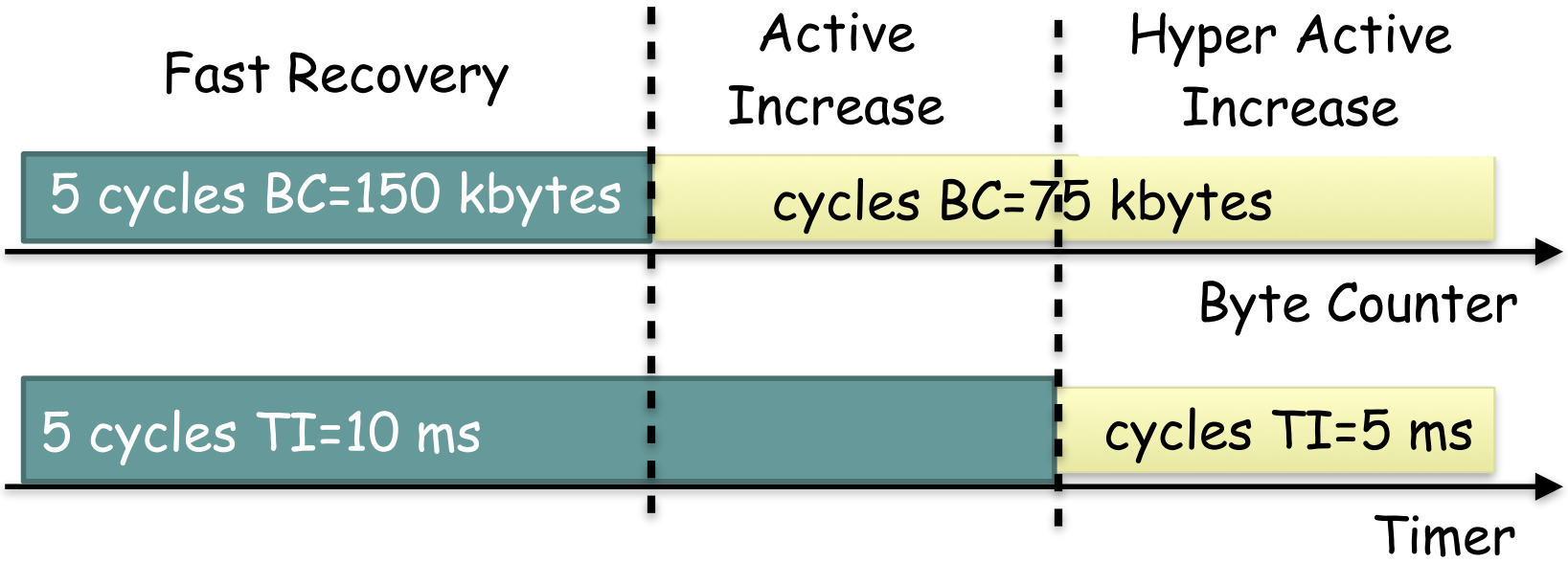
$$TR \leftarrow TR + R_{AI} \qquad CR \leftarrow \frac{1}{2}(CR + TR)$$

  - AI is entered when *either* TI or BC completes its FR cycles (5 cycles), *but not both*. After the 5 cycles, BC and TI are halved.
  - $R_{AI} = 5$ Mbit/s.
- **Hyper-Active Increase (HAI)**: when one cycle of either BC or TI is completed

$$TR \leftarrow TR + R_{HAI} \qquad CR \leftarrow \frac{1}{2}(CR + TR)$$

  - HAI is entered when *both* TI or BC complete their 5 FR cycles (500 pkts, 50 ms).
  - $R_{HAI} = 50$ Mbit/s.

Summary of BC, TI and phase evolution of the RP algorithm



- TI has been introduced to avoid slack source rate recovery and increase in case CR gets too low or there are not enough data to send