中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

Lecture 05：云中数据通信

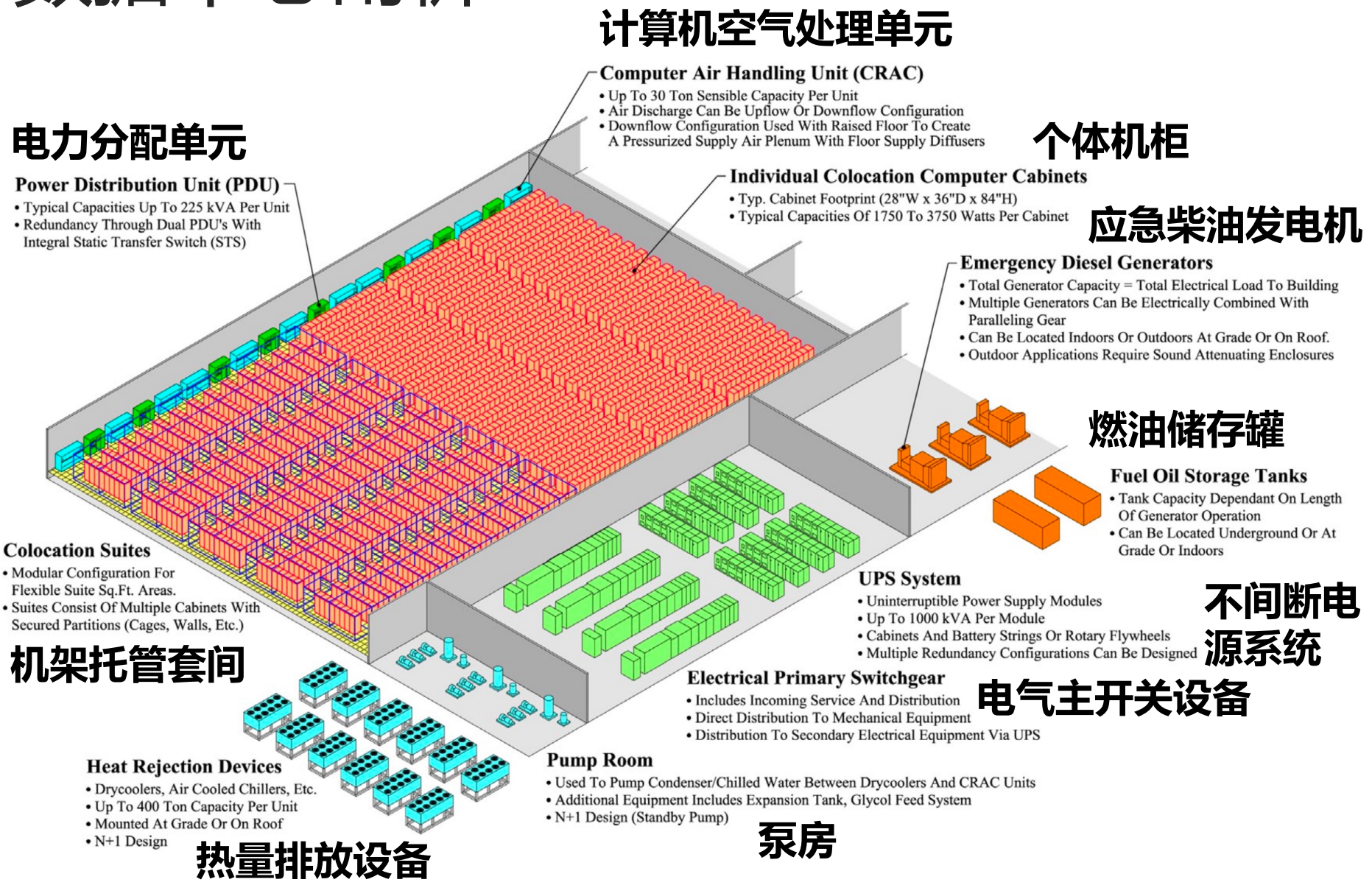
SSE316：云计算技术
Cloud Computing Technologies

陈壮彬

软件工程学院

chenzhb36@mail.sysu.edu.cn

数据中心剖析



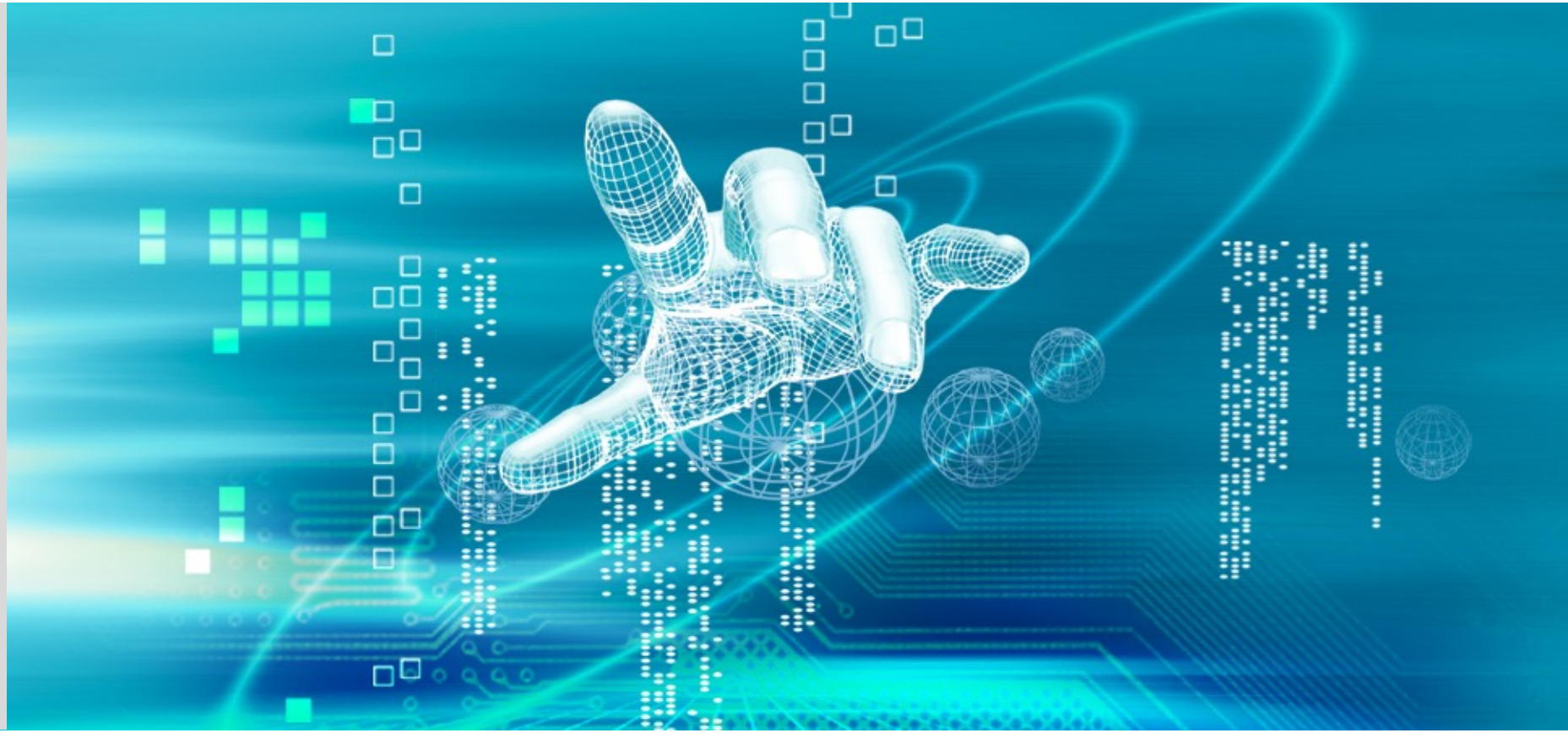
The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines

集装箱数据中心节能技术

定义

将数据中心的服务器设备、网络设备、空调设备、供电设备等**高密度地装入固定尺寸的集装箱中**，使其成为**数据中心的标准构建模块**，进而通过若干集装箱模块网络和电力的**互连互通构建完整的数据中心**。

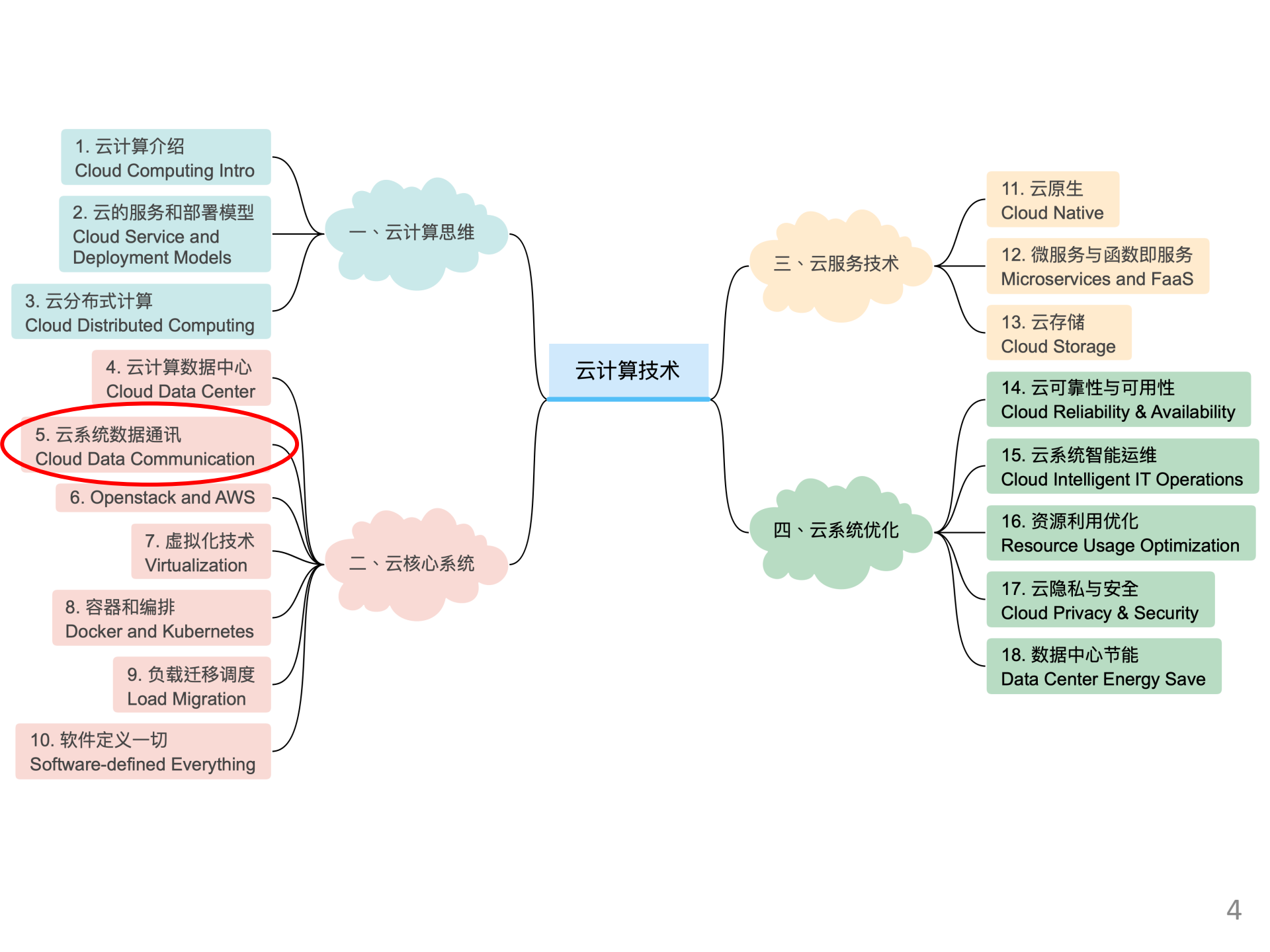
1. **高密度**
2. **模块化**
3. **按需快速部署**
4. **移动便携**



自动化管理

云计算数据中心的自动化管理使得在规模较大的情况下，实现**较少工作人员**对数据中心的**高度智能管理**。

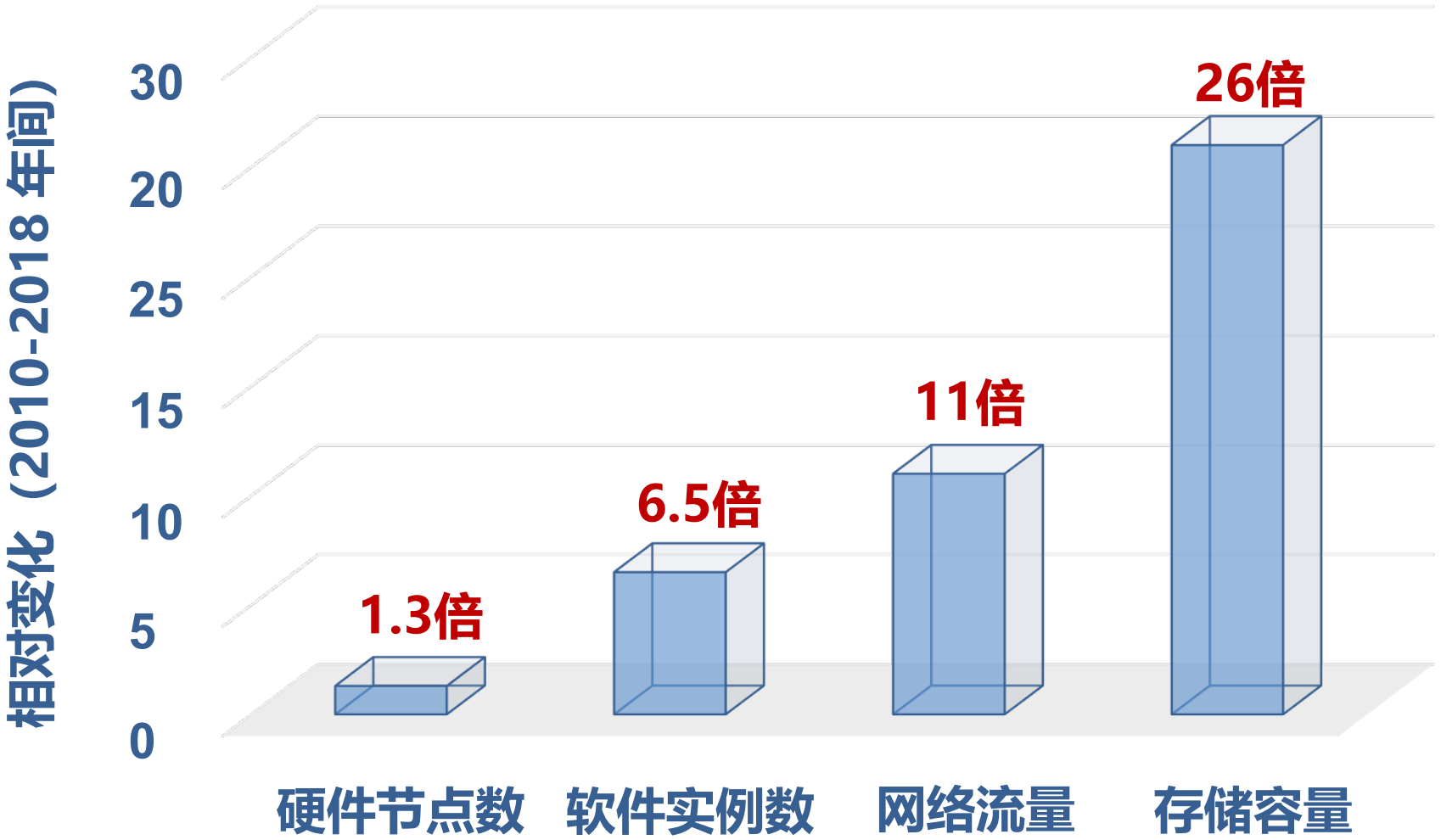



云计算技术
一、云计算思维
1. 云计算介绍
Cloud Computing Intro
2. 云的服务和部署模型
Cloud Service and
Deployment Models
3. 云分布式计算
Cloud Distributed Computing
二、云核心系统
4. 云计算数据中心
Cloud Data Center
5. 云系统数据通讯
Cloud Data Communication
6. Openstack and AWS
7. 虚拟化技术
Virtualization
8. 容器和编排
Docker and Kubernetes
9. 负载迁移调度
Load Migration
10. 软件定义一切
Software-defined Everything
三、云服务技术
11. 云原生
Cloud Native
12. 微服务与函数即服务
Microservices and FaaS
13. 云存储
Cloud Storage
四、云系统优化
14. 云可靠性与可用性
Cloud Reliability & Availability
15. 云系统智能运维
Cloud Intelligent IT Operations
16. 资源利用优化
Resource Usage Optimization
17. 云隐私与安全
Cloud Privacy & Security
18. 数据中心节能
Data Center Energy Save

Today’s topics

- 云中数据通信的特点
- 数据中心网络架构概览
- Clos / Fat-tree 网络架构
- AI 时代的数据中心网络
- 内容分发网络 (CDN)

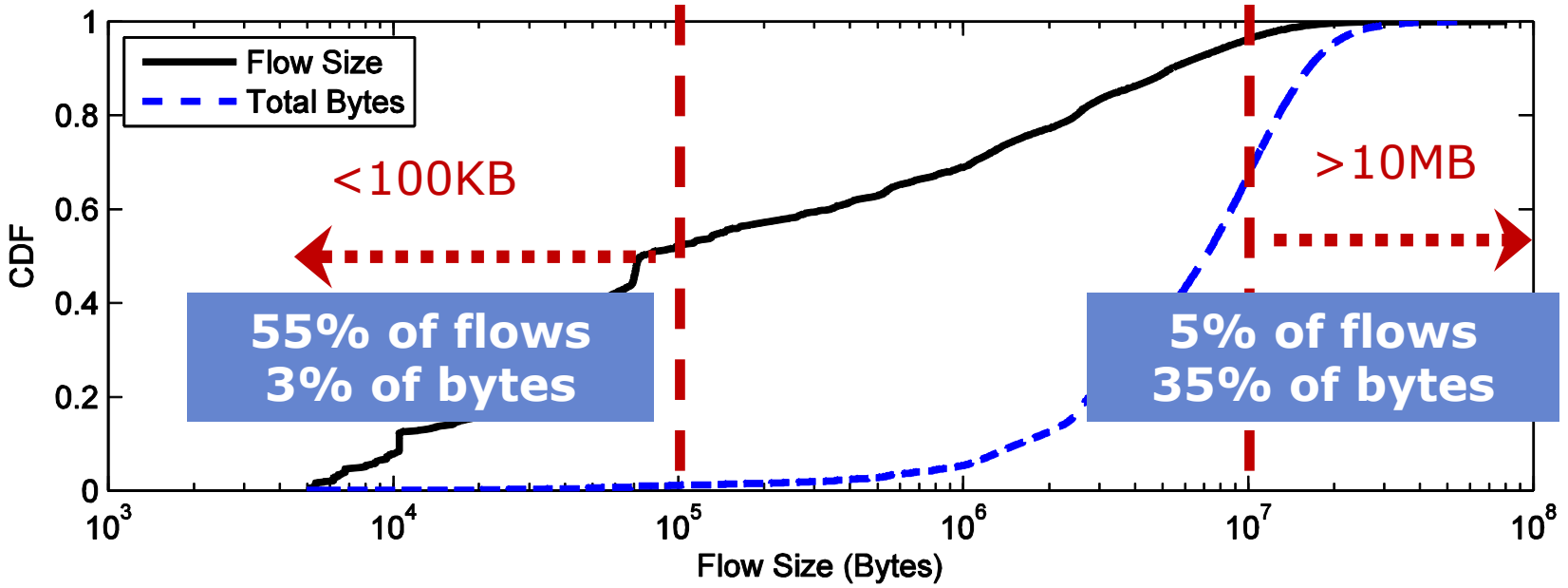
数据通信与存储在数据中心中的分量上升巨大



E. Masanet, et al. Recalibrating global data center energy-use estimates, Science, Vol 367, Issue 6481, 2020

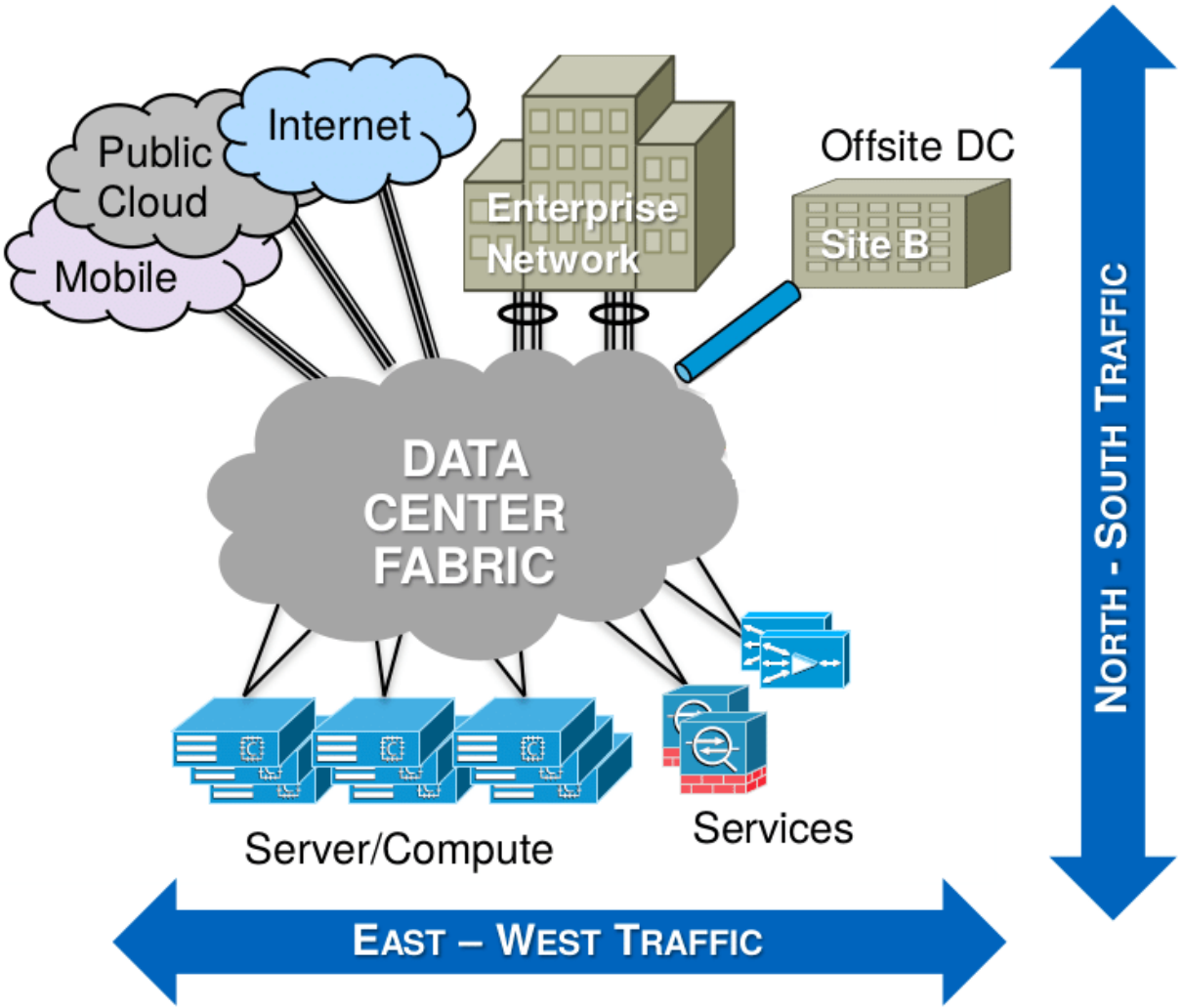
数据中心网络的流量特点

- 大部分流量由**大流**产生，小部分由**小流**产生
  - 大流：带宽密集型的应用，如数据备份、虚拟机迁移等
  - 小流：一些交互式的应用，如 web 服务、数据库查询等
- 对数据中心网络的管理和优化很重要
  - 大流： 采取策略来保障其带宽，以防止网络拥塞或中断
  - 小流： 优化其传输延时，以提高应用的响应速度



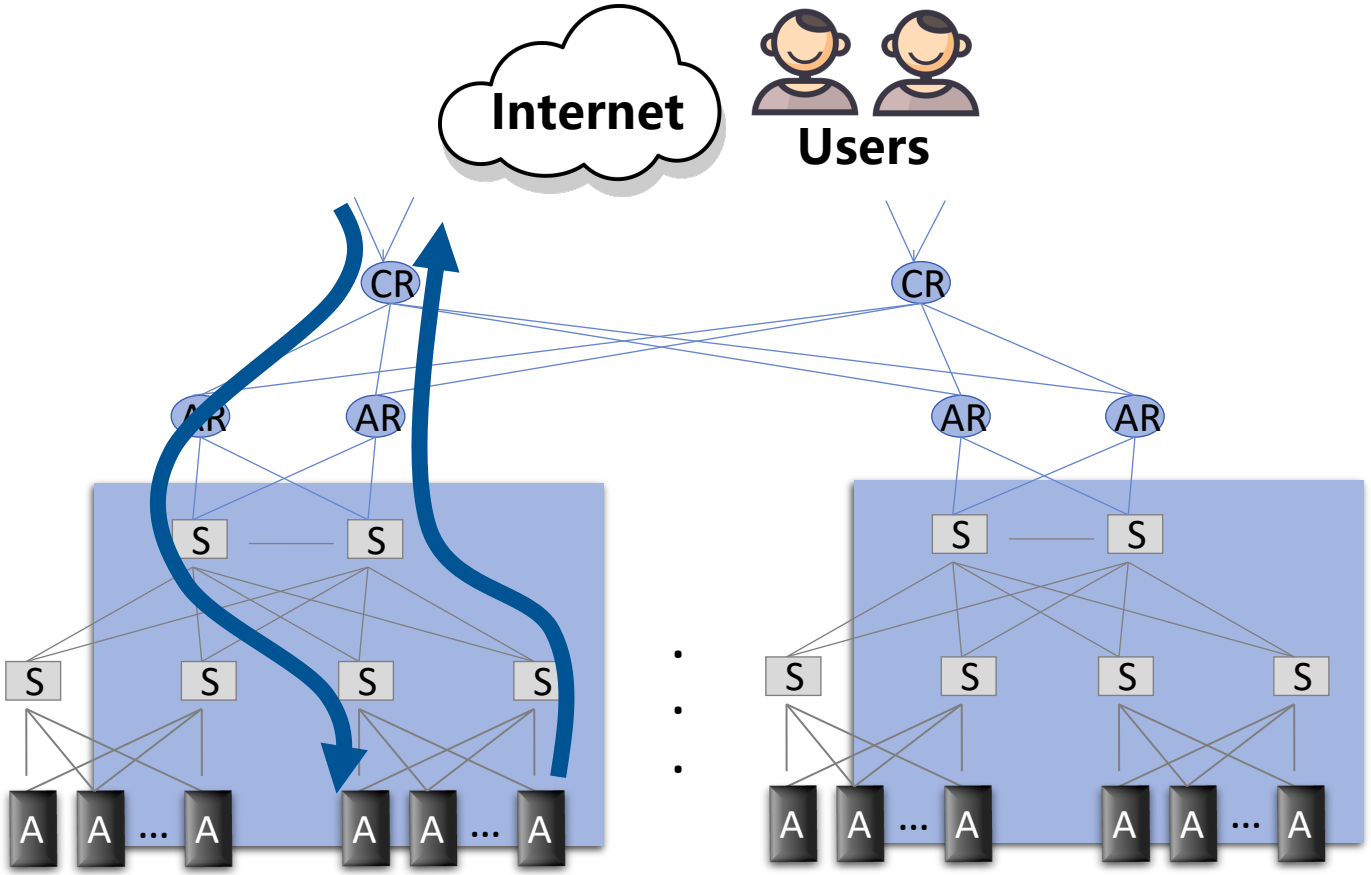
Web search, data mining (Microsoft) [Alizadeh 2010]

数据中心内的流量类型


Internet
Public Cloud
Mobile
Enterprise Network
Offsite DC
Site B
DATA CENTER FABRIC
Server/Compute
Services
NORTH - SOUTH TRAFFIC
EAST – WEST TRAFFIC

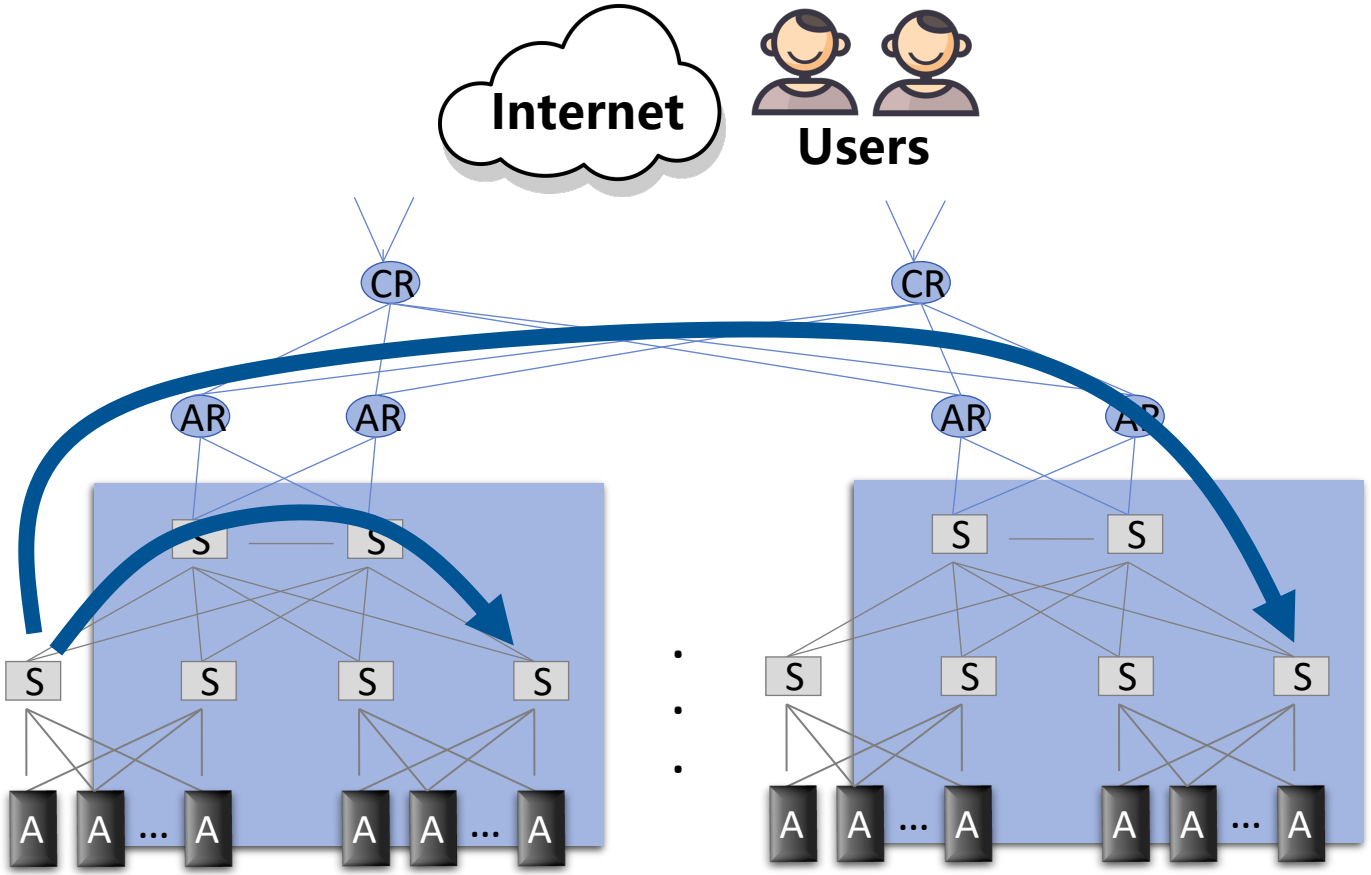
南北向流量（North-south Traffic）

- **数据中心内部**与**外部客户端**之间的流量
- 例如，用户通过互联网访问数据中心中的一个应用或服务



东西向流量（East-west Traffic）

- **数据中心内部的流量**，即服务器与服务器之间的通信
- 例如，“大数据”分布式计算、大模型训练中的流量



为什么要区分不同方向的流量?

- 具有不同的特征和需求，对数据中心网络设计和管理非常重要

南北向流量

用户↔数据中心

带宽需求：相对较低

延迟敏感度：中等

典型应用：Web 请求、API 调用

优化重点：负载均衡、CDN 加速

东西向流量

服务器↔服务器

带宽需求：非常高

延迟敏感度：高

典型应用：分布式计算、AI 训练

优化重点：网络拓扑、ECMP路由

数据中心网络设计要求

高带宽

支撑海量东西向流量满足 AI/大数据等负载需求

低延迟

微秒级端到端通信减少 tail latency

可扩展性

支持从数百到数万节点无缝横向扩展

高容错性

链路/设备故障自动切换多路径冗余设计

成本效益

使用商用交换机(COTS)避免厂商锁定

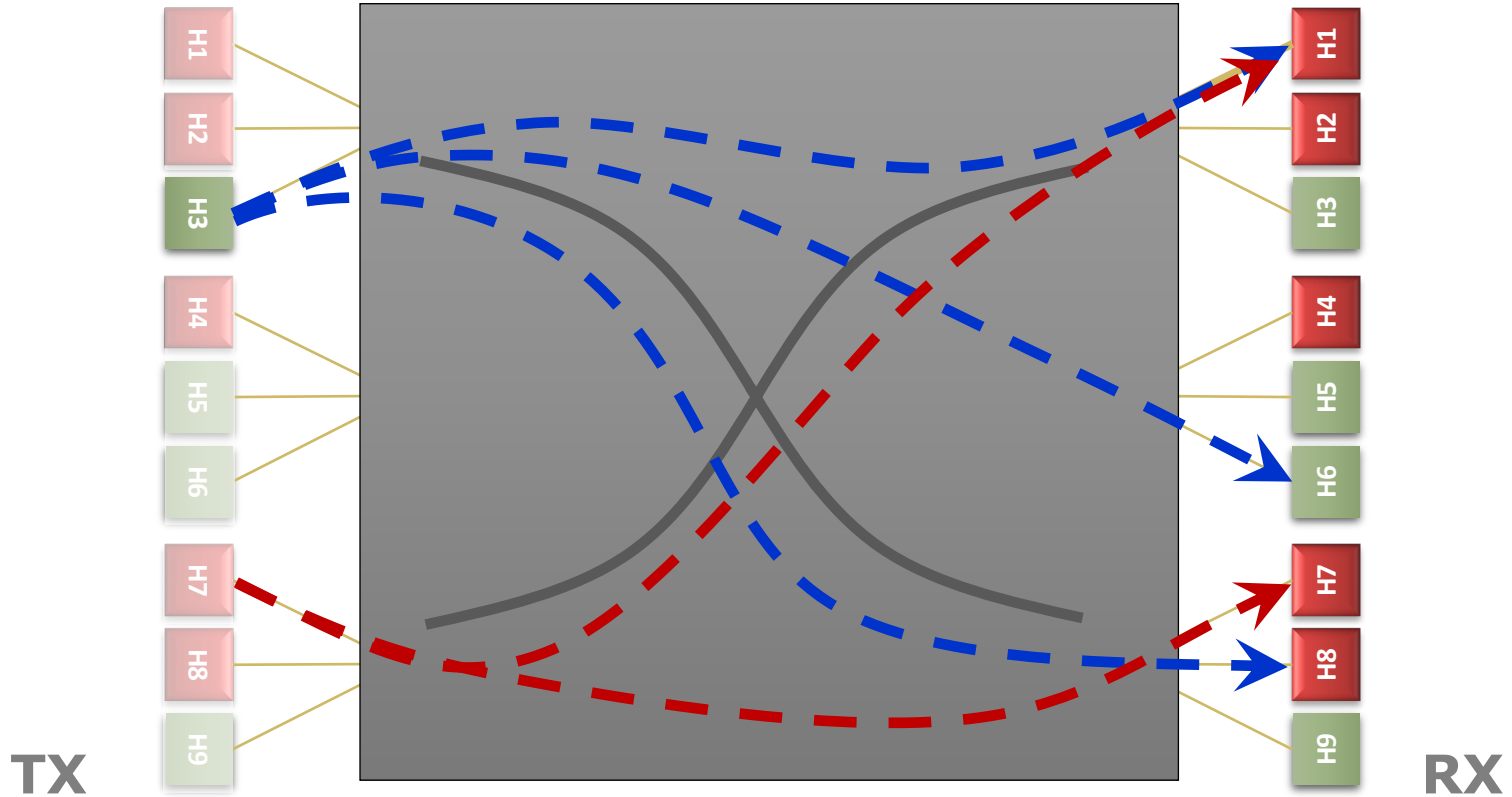
数据中心网络架构

数据中心网络架构是指用于组织和连接数据中心内部各种硬件设备（如服务器、存储设备、交换机等）的网络设计。

它是数据中心的基础架构之一，对数据中心的性能、可靠性和可扩展性都有很大的影响

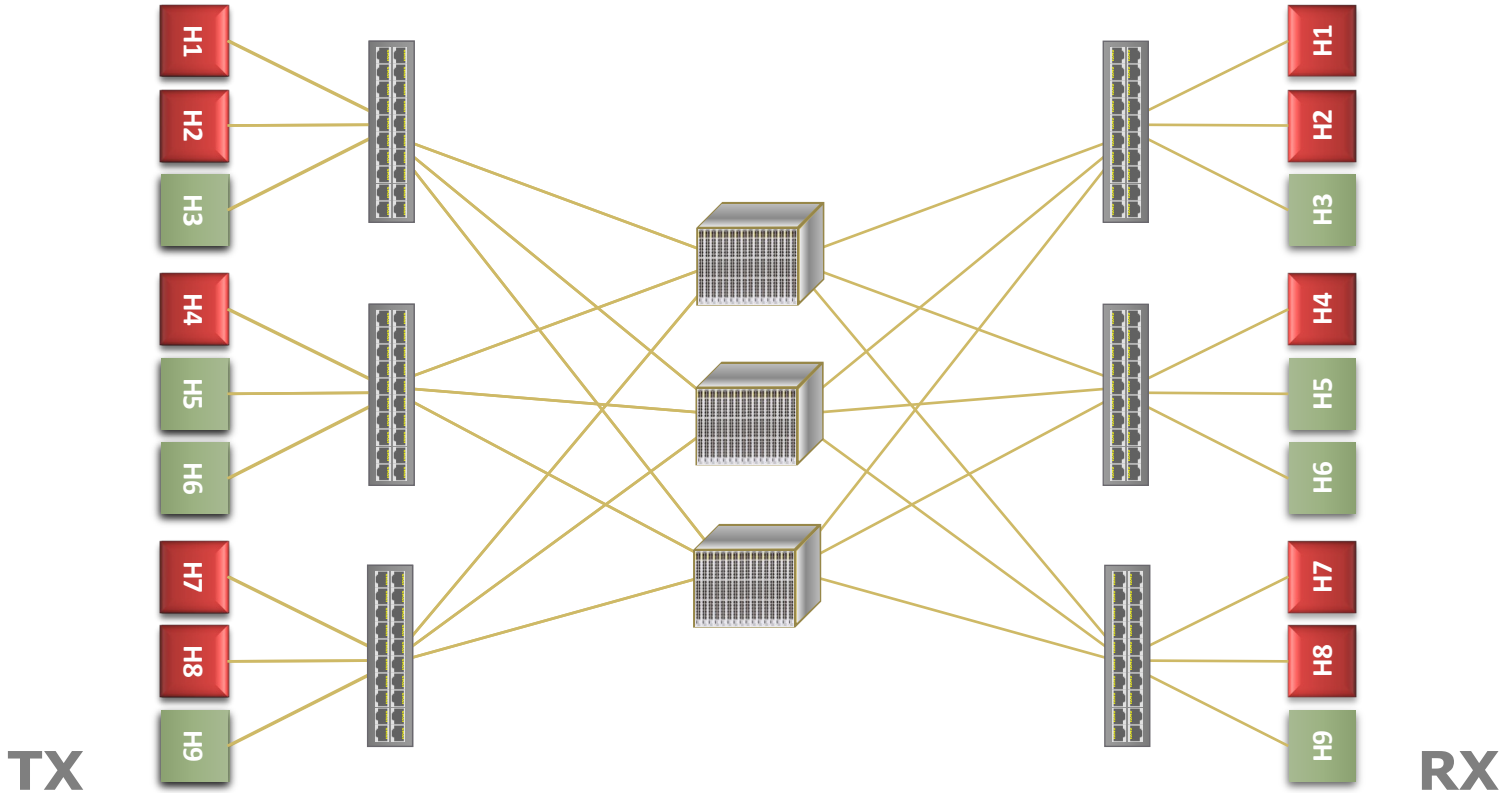
数据中心网络：一个大型交换机

- 将数据从**发送端（TX）**传输至**接收端（RX）**

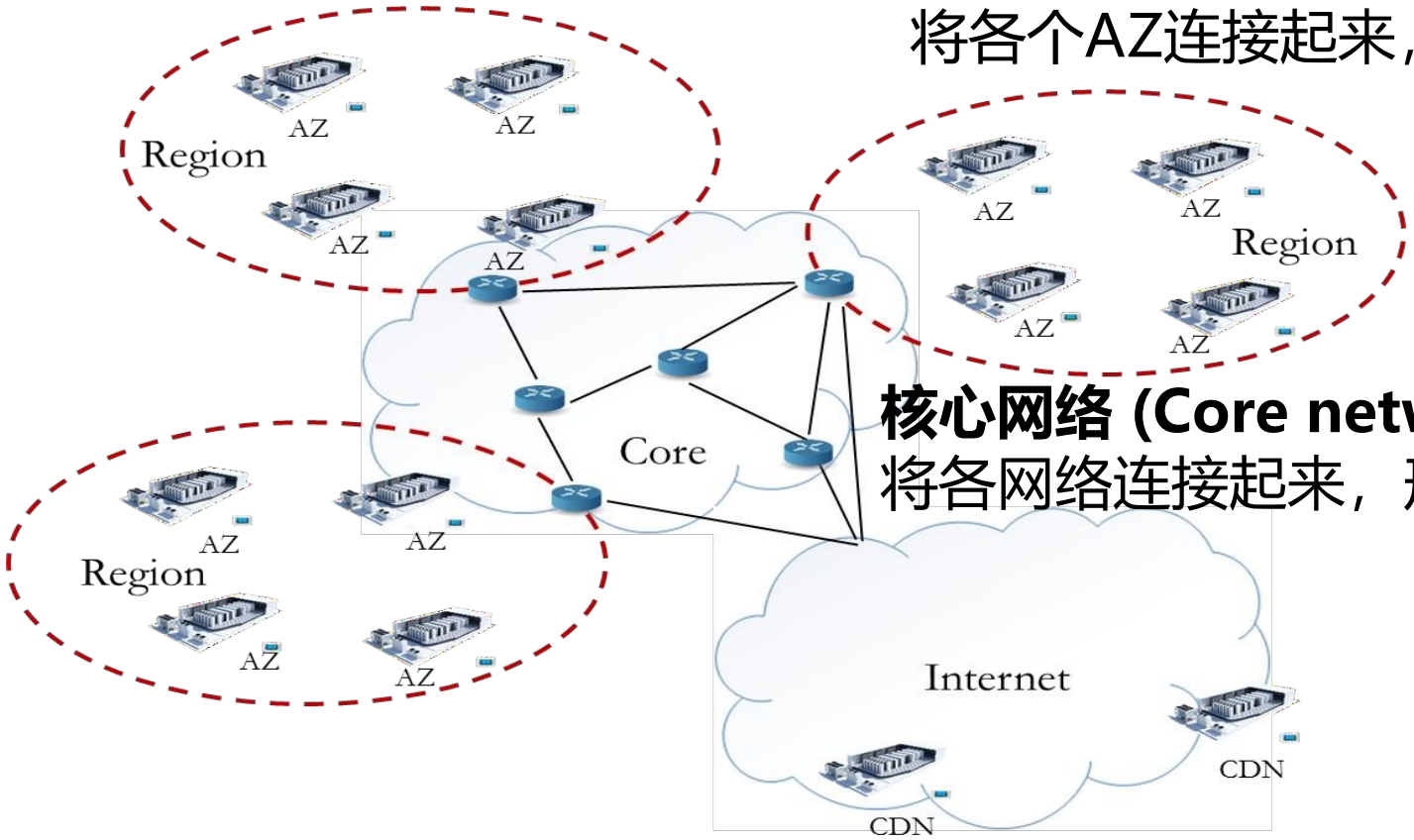


数据中心网络：一个大型交换机

- 内部需要**复杂的网络架构**来确保数据的**高效和准确传输**



一朵云的主要网络组成部分



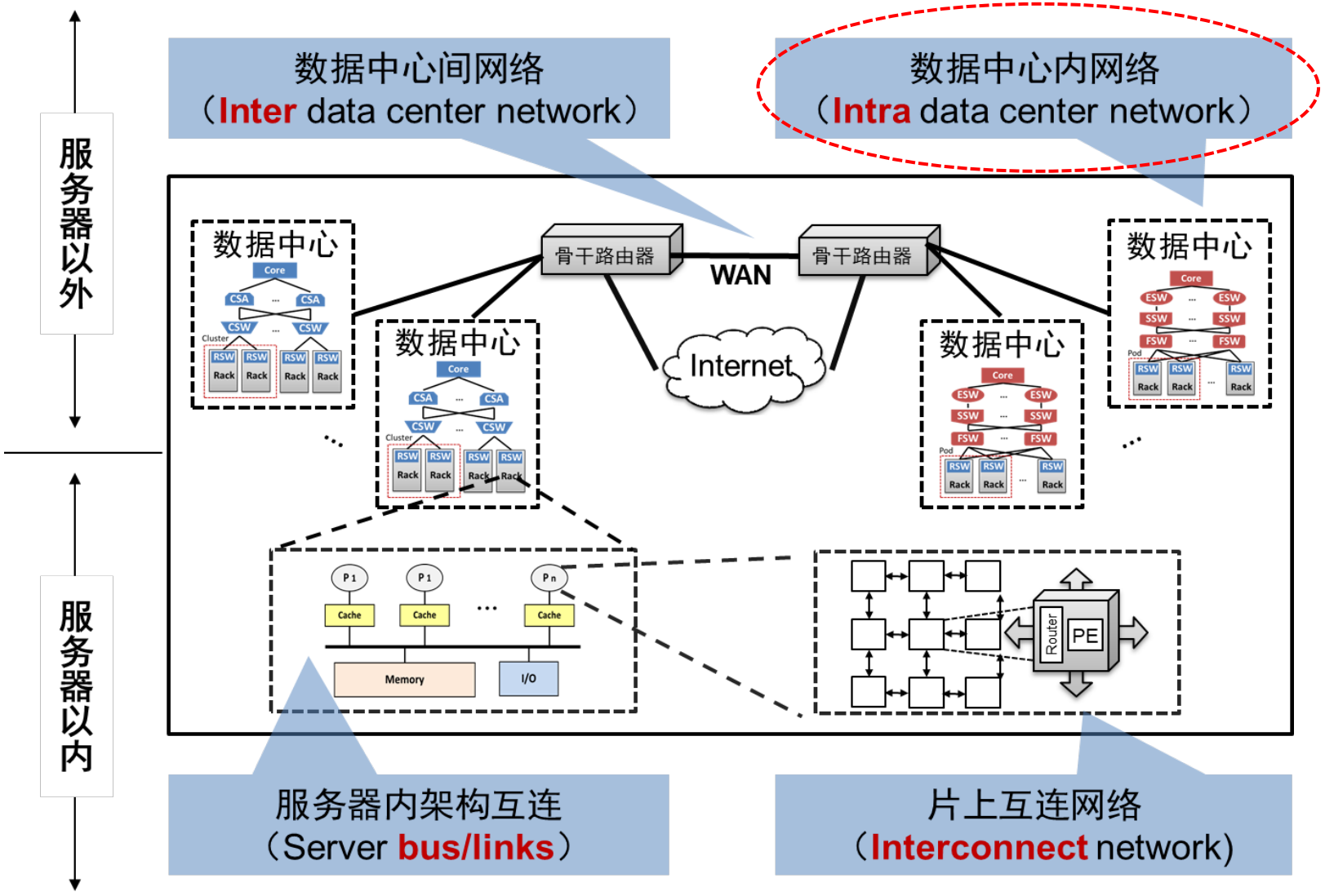
区域网络 (Regional network)
将各个AZ连接起来，以覆盖较大范围

核心网络 (Core network)
将各网络连接起来，形成更大范围整体

边缘或内容分发网络 (Edge/CDN)
连接了区域网络和公网或服务提供商

数据中心网络拓扑

两大四小层面看数据通信



如何高效地互联多个主机?

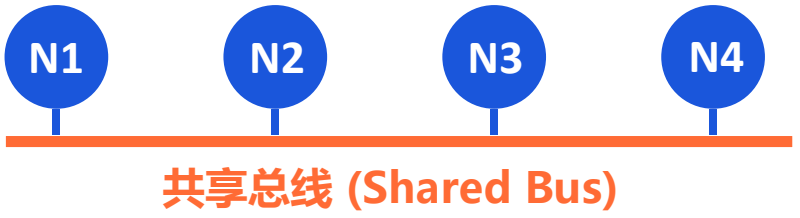

Don’t worry about me

互联架构：总线 (Bus)

Bus：所有节点共享一条通信线路

- 任意时刻只有一个节点可以发送数据
- 其他节点必须等待（争用机制：CSMA/CD）
- 常见于早期以太网、嵌入式系统
- 结构最简单，成本最低

Bus 拓扑示意图



1	N	O(1)	✗ 无
直径	度数	带宽	容错

⚠ 冲突问题

N1 和 N2 同时发送 → 冲突！
必须等待重传，性能随节点数增加而急剧下降。
∴ 不适合大规模数据中心

📖 网络拓扑的关键指标

直径

任意两个节点之间最短路径的最大值

↓ 越小 = 延迟越低

度数

每个节点的直接连接数

↑ 越大 = 成本越高

对分宽度

将网络一分为二需移除的最少边数

↑ 越大 = 带宽越高

互联成本

总连接/交换机数量的增长关系

$O(N)$ vs $O(N^2)$

线性 vs 平方

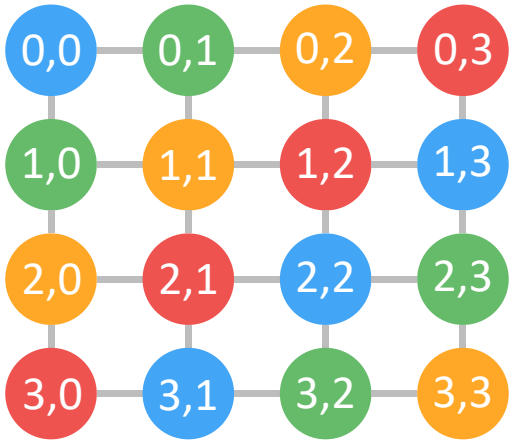
互联架构：网格 (Mesh)

Mesh：节点按网格状连接

- 每个节点与相邻节点直接连接（上下左右）
- 多条路径可选 → 具备一定容错能力
- 不同节点对之间延迟不均匀（角落 vs 中心）
- 常见于片上网络 (NoC)、超算集群
- 互联成本：O(N)，线性增长

2(√N-1)	4	O(N)	✓ 多路径
直径	度数	成本	容错

4×4 Mesh 网格拓扑



● 直径 = 2(√16-1) = 6（左上角⇒右下角）

Mesh vs Bus 对比

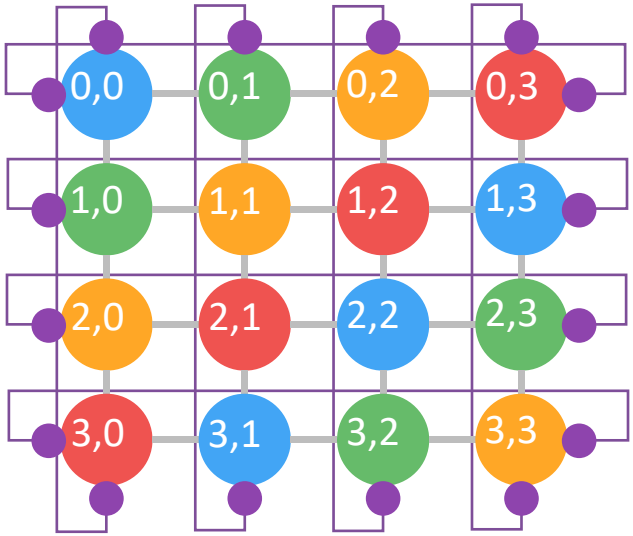
	Bus	Mesh
拓扑	单线共享	网格状
直径	1	2(√N-1)
并发通信	✗ 不支持	✓ 支持
容错	✗ 单点故障	✓ 多路径
成本	最低	O(N)

互联架构：环面 (Torus)

Torus：在 Mesh 基础上添加首尾连接

- 每行/每列的首尾节点通过 wrap-around 链路相连
- 效果：大幅降低直径（从 2(√N-1) 降到 √N）
- 所有节点的角色更对称，延迟更均匀
- 常见于超算集群（如 IBM Blue Gene）
- 互联成本略高于 Mesh（额外 wrap-around 线缆）

4×4 Torus 环面拓扑



√N	4	O(N)+	✓ 更好
直径	度数	成本	容错

● 紫色圆点 = wrap-around 连接点

Torus 如何降低直径？—— 以 4×4 为例

Mesh：(0,0) → (3,3) 需要 3+3 = 6 跳

路径：(0,0)→(1,0)→(2,0)→(3,0)→(3,1)→(3,2)→(3,3)

Torus：(0,0) → (3,3) 只需 1+1 = 2 跳（反方向 wrap）

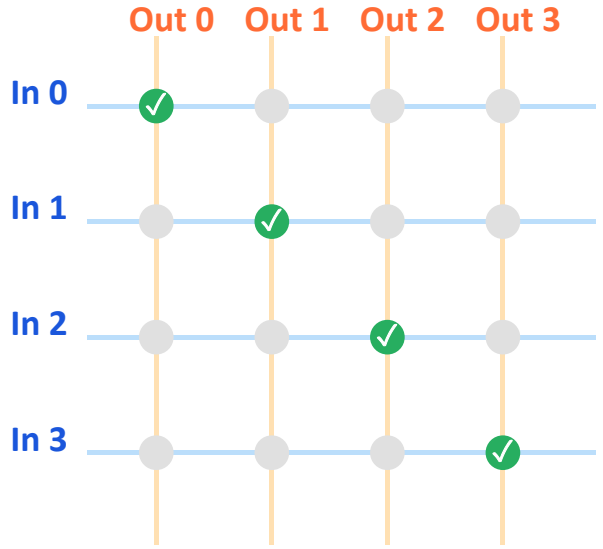
路径：(0,0)→(3,0)→(3,3)（通过 wrap-around）

直径：6 → 2，减少 67%！

互联架构：交叉开关 (Crossbar)

Crossbar：全互联交叉矩阵

- 每个输入都能直接连接到每个输出
- N 个端口需要 N^2 个交叉点 → 成本 $O(N^2)$
- 并行度最高：多对可同时通信（无冲突）
- **仅适用于小规模（芯片内、小型交换机）**



✓ 绿色 = 当前活跃连接（可同时 4 对）

1	N	O(N)	$O(N^2)$
直径	度数	带宽	成本

四种互联架构对比总结

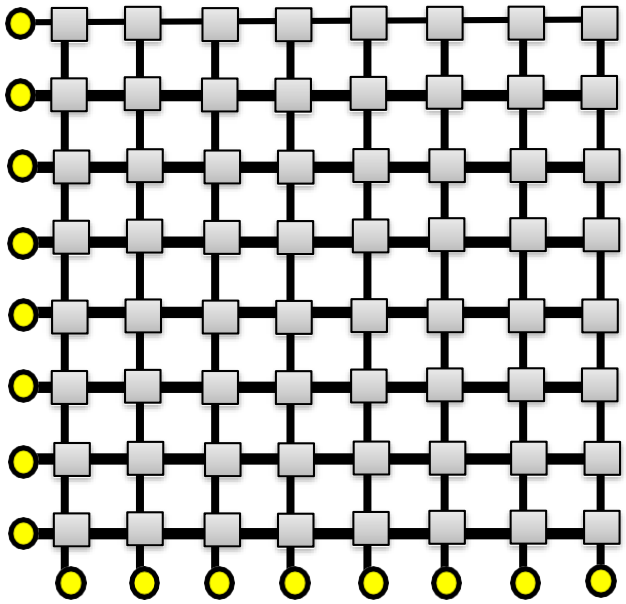
	Bus	Mesh	Torus	Crossbar
直径	1	2(√N-1)	√N	1
度数	N	4	4	N
成本	O(1)	O(N)	O(N)+	$O(N^2)$
带宽	O(1)	中	中-高	O(N)
容错	无	多路径	更好	无
适用场景	嵌入式	NoC/HPC	超算	芯片内

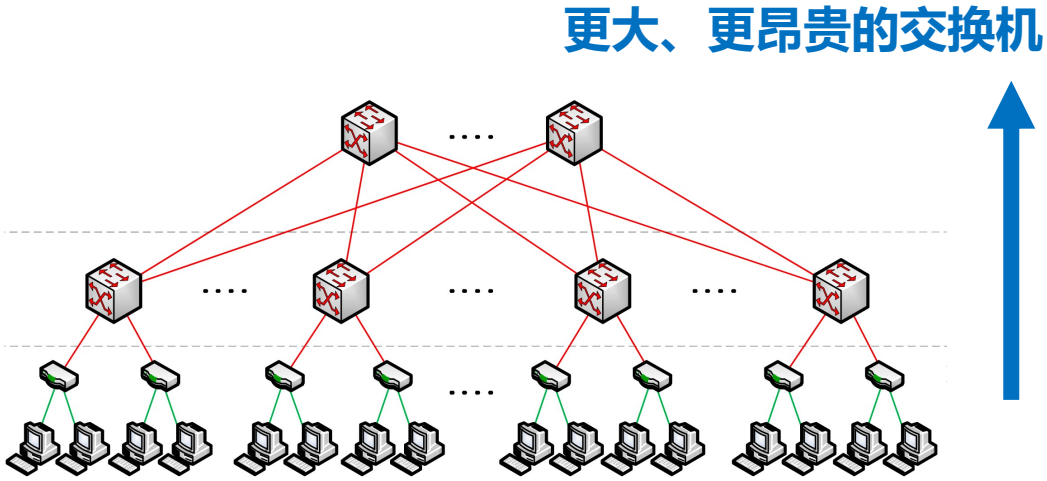
➡ **这些架构各有优劣，数据中心需要更好的方案 → Clos / Fat-tree 架构**

数据中心经典网络架构

Clos 网络架构

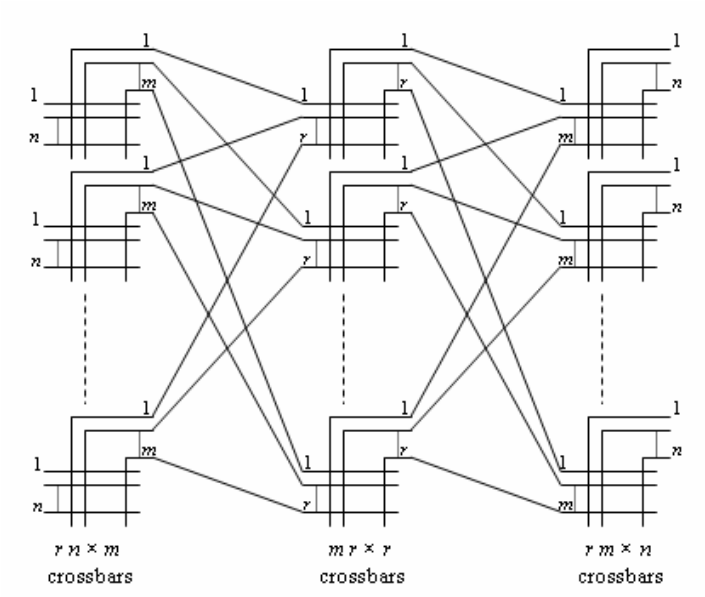
- Clos 网络是一种**非阻塞的多级交换网络**（non-blocking, multistage switching network）
- 在1950年代引入，以**提高电话交换网络的效率**并**降低其成本**（在此之前，交换机的数量须等于输入和输出数量的乘积，即**n的平方 n^2**）
- 用多个**小规模、低成本**的交换机构建复杂、大规模的网络架构



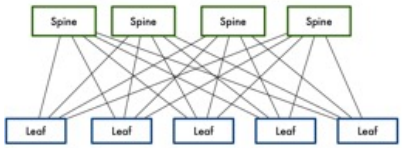


Clos 网络架构

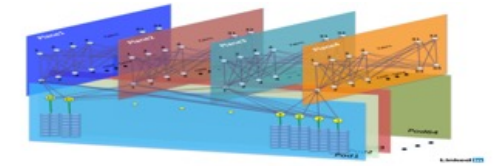
- 克洛斯利用数学理论证明，有可能**使用更少的交换机实现非阻塞连接性**，被称为 Fabric
- 至今，Clos 网络仍然是许多**互联结构的基础**，包括驱动云计算的大规模数据中心中的结构



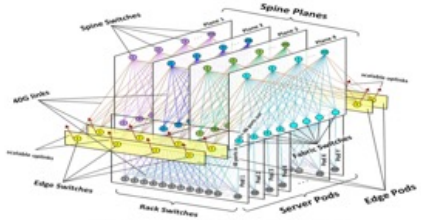
Three-stage Clos network



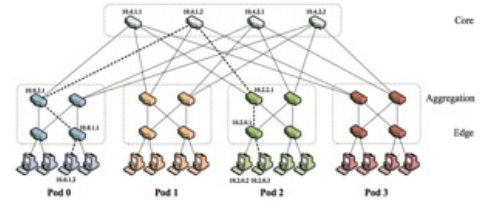
Leaf and Spine



Reference: https://engineering.linkedin.com/blog/2016/03/the-linkedin-data-center-100g-transformation



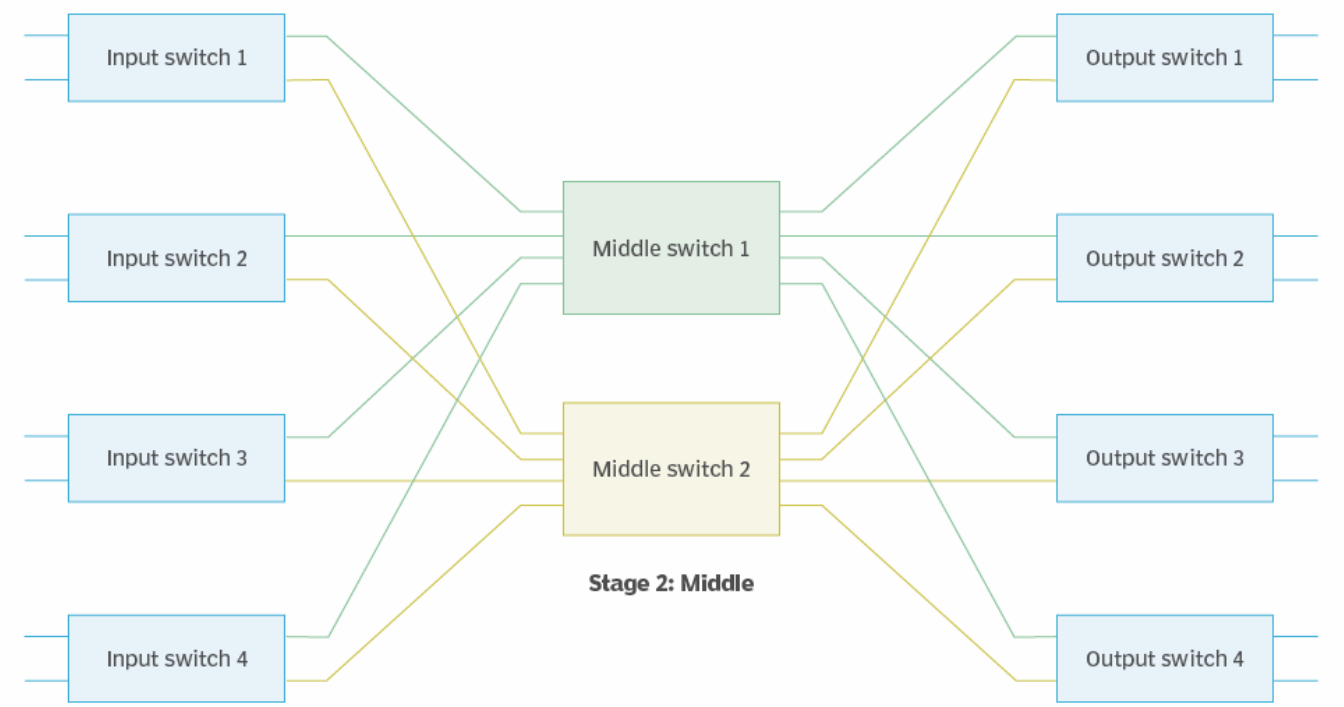
Reference: https://code.fb.com/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/



Reference: A Scalable, Commodity Data Center Network Architecture

Clos 网络架构

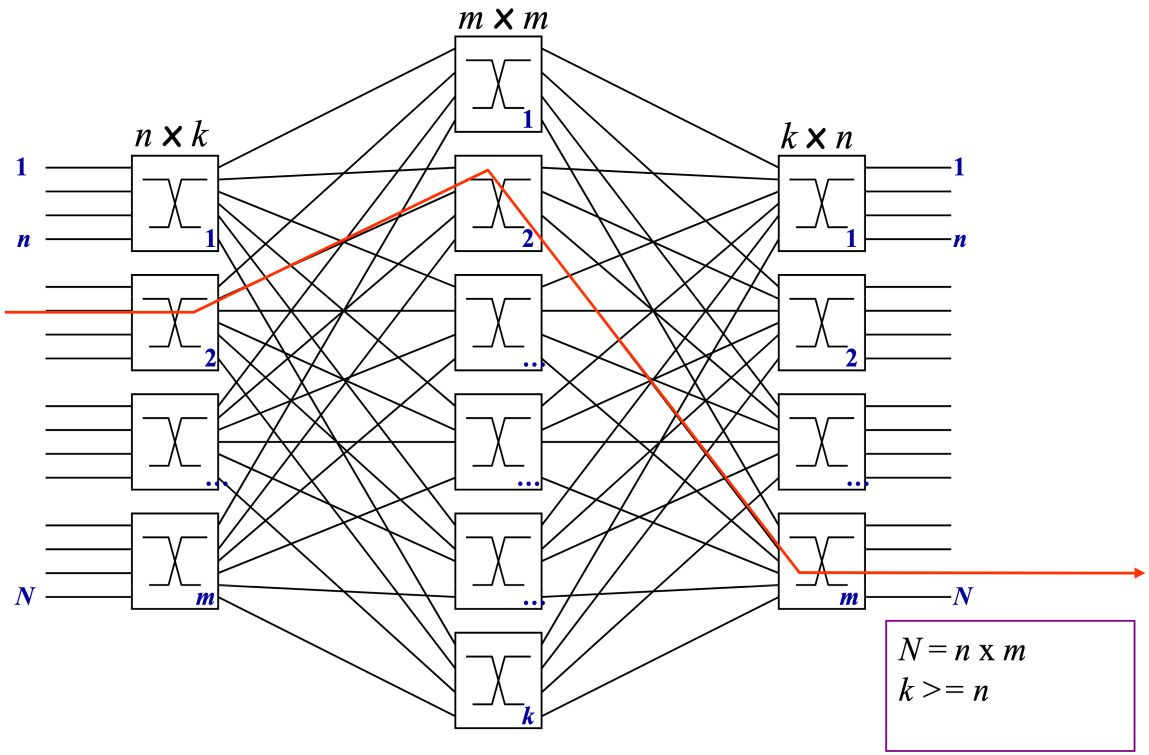
- 为了实现该连接性，交换机被组织成一个三阶段的架构：
  - Ingress（输入）阶段
  - Middle（中间）阶段
  - Egress（输出）阶段

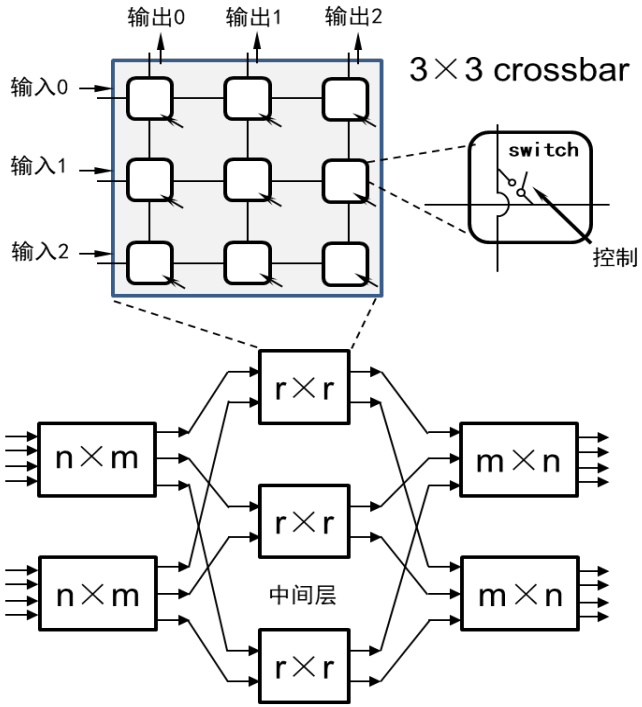


Three-stage Clos network

Clos 网络架构

- 三阶段 Clos 网络由 3 个整数定义：n, m, k
  - Ingress 阶段有 m 个交换机，每个包含 n 个输入和 k 个输出
  - Middle 阶段有 k 个交换机，每个包含 m 个输入和 m 个输出
  - Egress 阶段有 m 个交换机，每个包含 k 个输入和 n 个输出





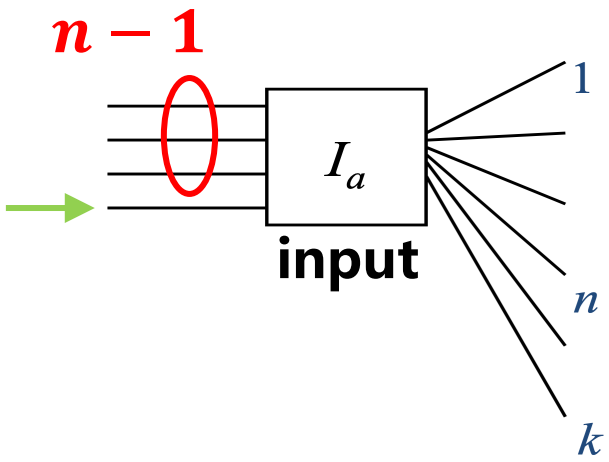
Clos(n, m, r)结构。上例中n=4,m=3,r=2

Clos 网络架构

Clos 定理

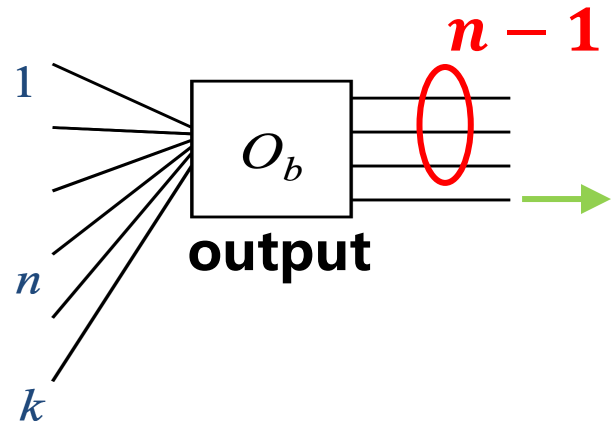
若 $k \geq 2n-1$，则 Clos 网络是严格意义上的非阻塞

□==> 在 Ingress 交换机上一个未使用的输入总可以连接到 Egress 交换机上一个未使用的输出，而无需重新排列



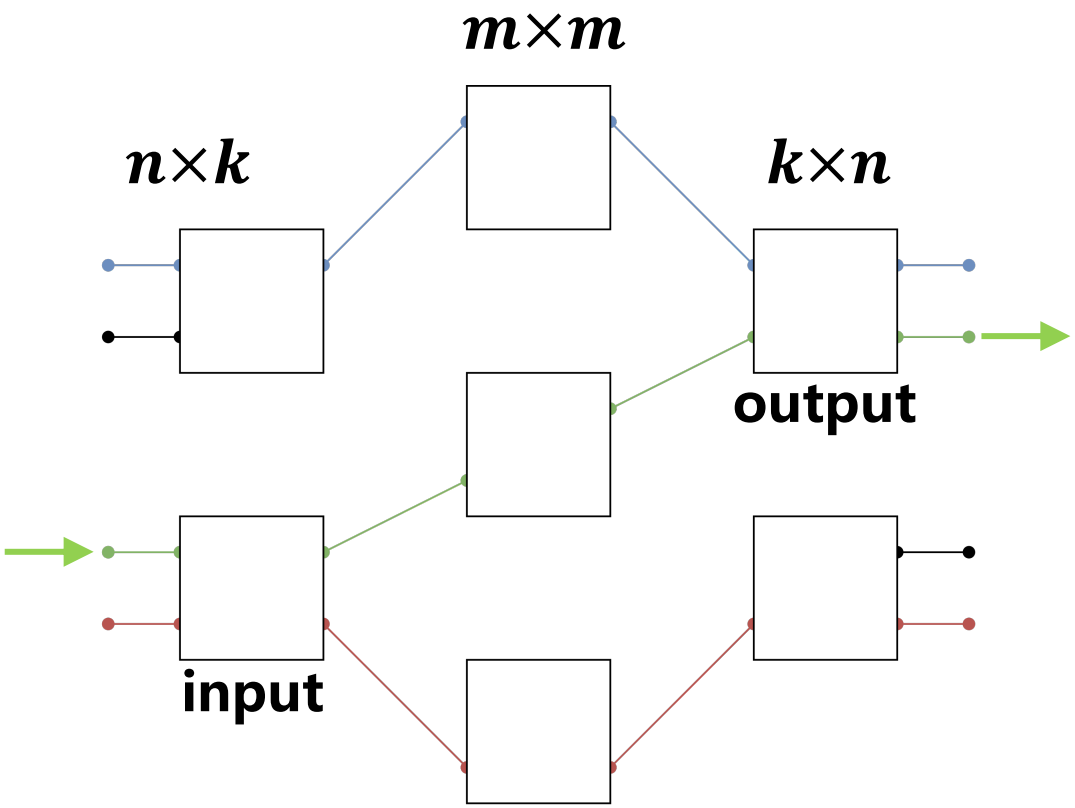
最差的情况是，输入和输出的其他 $n-1$ 条通道分别占用了不同的 Middle 交换机

因此，我们需要第 $(n-1)+(n-1)+1$ 个 Middle 交换机，即 $k \geq 2n-1$



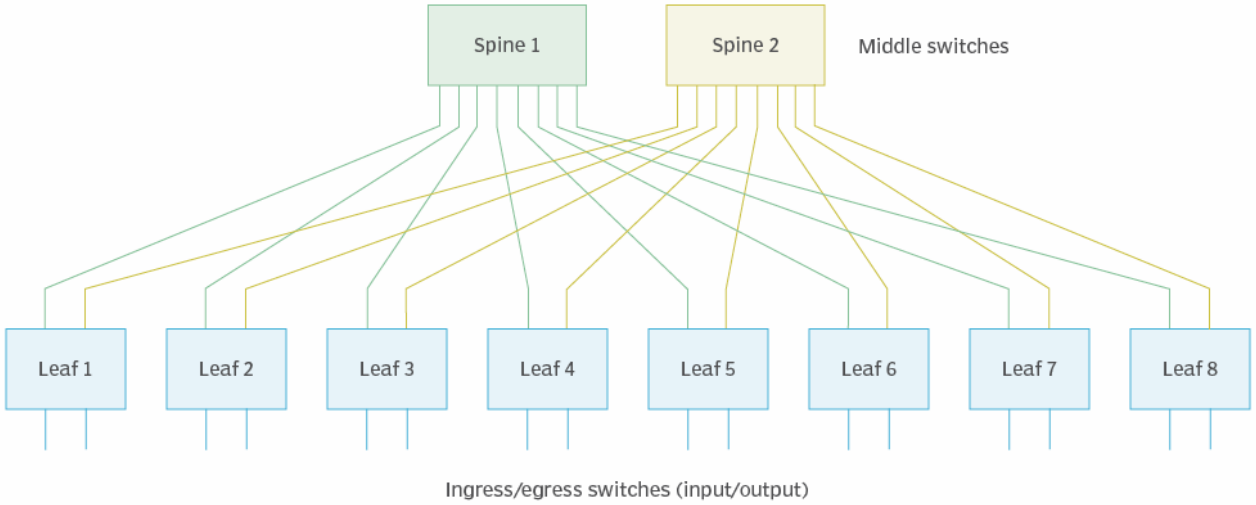
Clos 网络架构

- 一个简单的例子
  - $n = 2, k = 1, m = 1$
  - 红色和蓝色的路径分别占用了不同的 Middle 交换机
  - 因此，我们需要至少有 $2n - 1 = 3$ 个 Middle 交换机

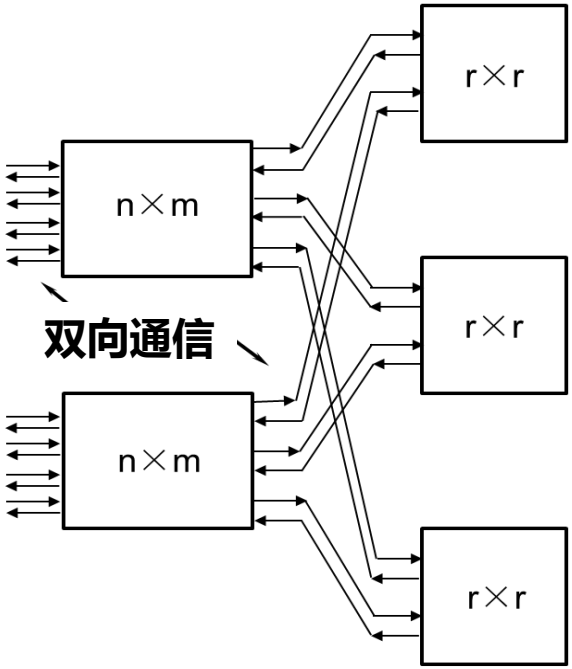


Clos 网络架构

- 如今的数据中心中，Clos多以叶脊 (leaf-spine) 形式布局
- 这种设计通常被称为折叠的 Clos 网络
- 更常被称为**胖树网络 (Fat-tree network)**



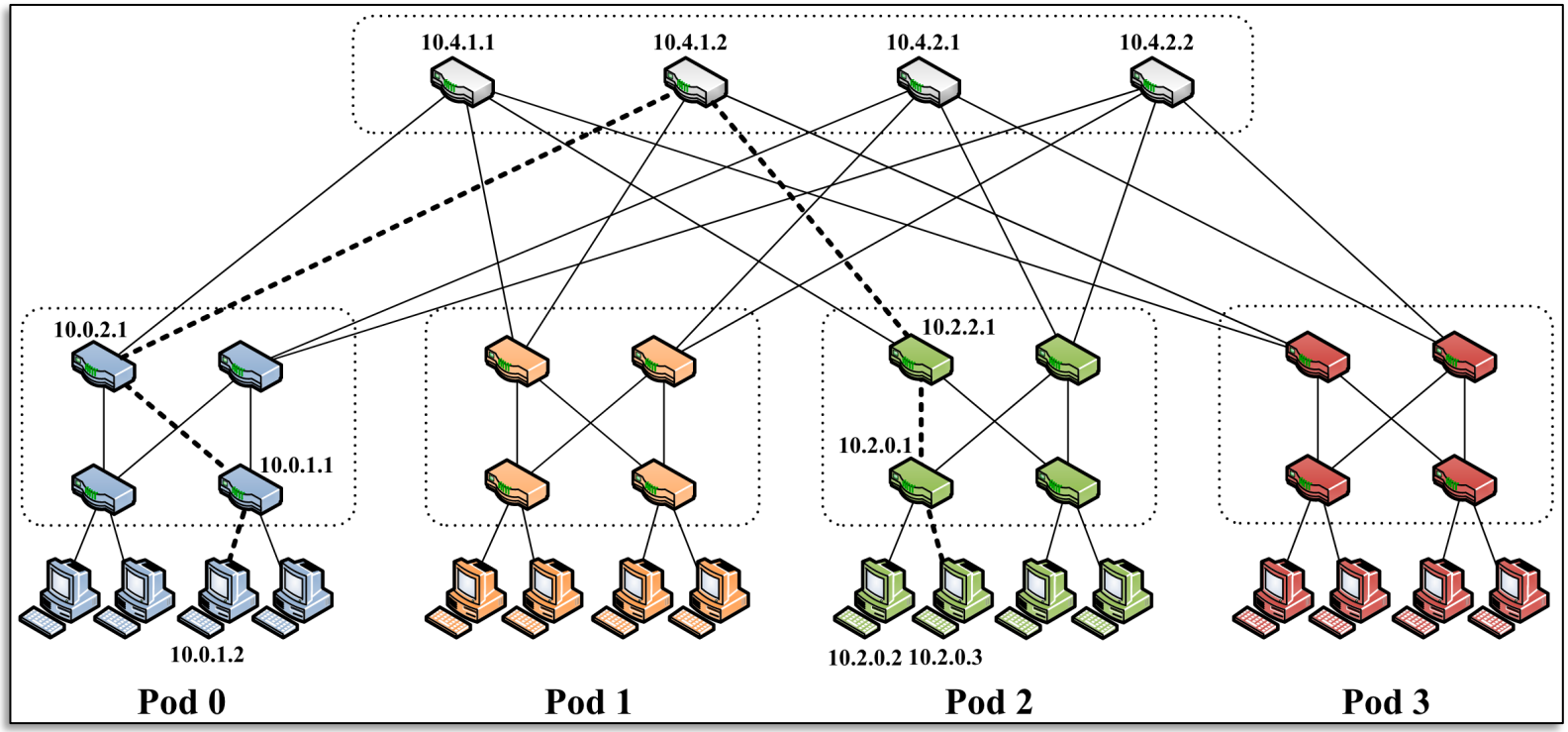
Clos leaf-spine network / Fat-tree network



Folded-Clos 结构

胖树网络架构

- **由三层网络交换机组成的多根树形网络拓扑结构**
  - 核心层连接汇聚层交换机和公网
  - 汇聚层连接多个内部交换机
  - 最底层服务器和接入交换机相连



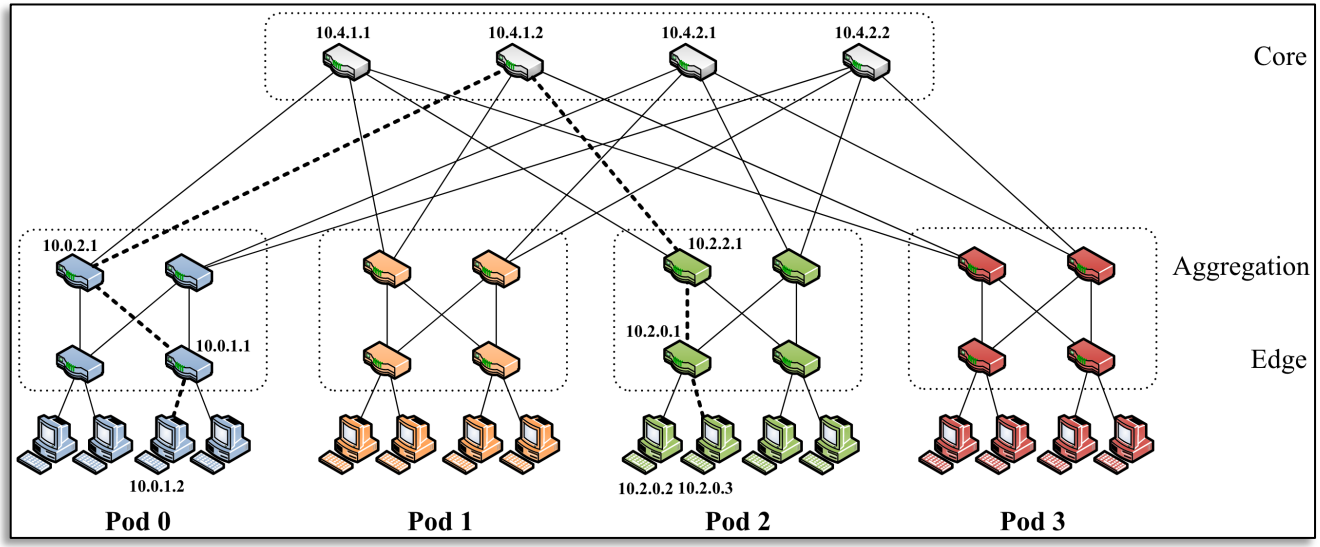
Core
核心层

Aggregation
汇聚层

Access/Edge
接入/边界层

Hosts
主机

Fat-tree 网络架构



An example Fat-tree architecture with k=4

□k-ary Fat-tree：

- ✓ 每个交换机有 k 个端口（编号 0-3），网络包含 k 个 pod
- ✓ 每个 pod 包含两层交换机（汇聚和接入），每层有 $k/2$ 个交换机
- ✓ 每个接入交换机分别与 $k/2$ 个主机和 $k/2$ 个汇聚交换机相连
- ✓ 网络中共有 $(k/2)^2$ 个核心交换机，所有核心交换机的第 i 个端口连接第 i 个 pod
- ✓ 汇聚交换机的端口以 $k/2$ 的划窗连接核心交换机
- ✓ 能支持 $k^3/4$ 个主机，当 $k = 4$ 时，$k^3/4 = 4^3/4 = 16$

传统 Fat-tree 架构的优缺点

□传统 Fat-tree 架构的优点

- ✓所有交换机相同，可使用廉价的商用机降低成本，可随时替换
- ✓任意两个主机有 $(k/2)^2$ 条最短路径，主机获得完整带宽

□传统 Fat-tree 架构的两个问题

- ✓经典的 IP/Ethernet 网络只会构建**单路径路由协议** (如 ECMP 的静态负载均衡)，**无法充分利用多路径资源**，导致部分链路拥塞
- ✓大型数据中心网络中的**布线非常复杂**

改进 Fat-tree 架构

A Scalable, Commodity Data Center Network Architecture

Mohammad Al-Fares
malfares@cs.ucsd.edu

Alexander Loukissas
aloukiss@cs.ucsd.edu

Amin Vahdat
vahdat@cs.ucsd.edu

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0404

SIGCOMM’ 08

问题	解决思路
1. 性能瓶颈	升级版转发协议，提升扇出效率
2. 布线复杂	打包和放置技术（详见论文）

课堂问答

口与 10.110.12.29 mask 255.255.255.254 属于同一网段的主机 IP 地址是（）

A. 10.110.12.0
B. 10.110.12.28
C. 10.110.12.30
D. 10.110.12.31

回顾子网掩码

- 将一个大型网络分割成许多小的子网
- 用于标识 IP 地址中的**网络和主机部分**，确定目标网络

IP地址	子网掩码
192.168.1.100	255.255.255.0

11111111.11111111.11111111.0

AND运算

192.168.1.0

- 任何以 192.168.1 开头的 IP 地址都在同一个子网中
- 也记作 192.168.1.100/24（前面 24 个 1）

升级版转发协议 – 两级路由表

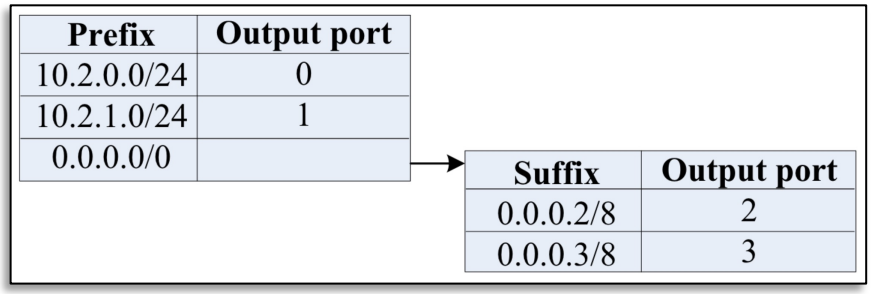
□TCP/IP 路由表

- Destination：目标主机 IP 地址
- Gateway：到达目标主机的网关或下一跳地址

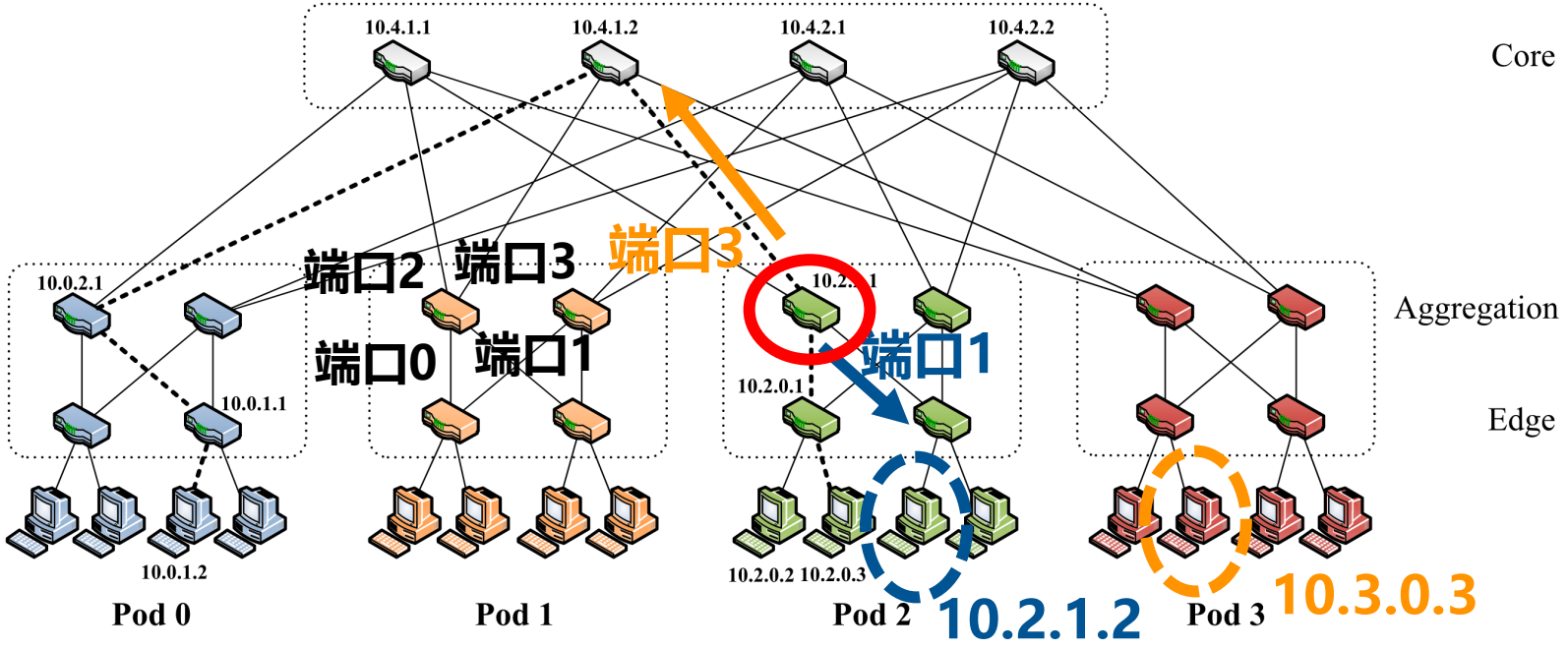
```
Routing tables
Destination      Gateway         Flags Refcnt Use        Interface
140.252.13.65    140.252.13.35   UGH   0      0          emd0
127.0.0.1        127.0.0.1       UH    1      0          lo0
default          140.252.13.33   UG    0      0          emd0
140.252.13.32    140.252.13.34   U     4      25043      emd0
```

□两级路由表

- 第一级是**前缀查找**
  - ✓用于将拓扑向下路由到端主机
- 第二级是**后缀查找**
  - ✓用于向核心路由
  - ✓扩散和分散交通
  - ✓通过对相同的端主机使用相同的端口来维护数据包顺序



两级路由表例子



Prefix	Output port
10.2.0.0/24	0
10.2.1.0/24	1
0.0.0.0/0	

前缀查找

后缀查找

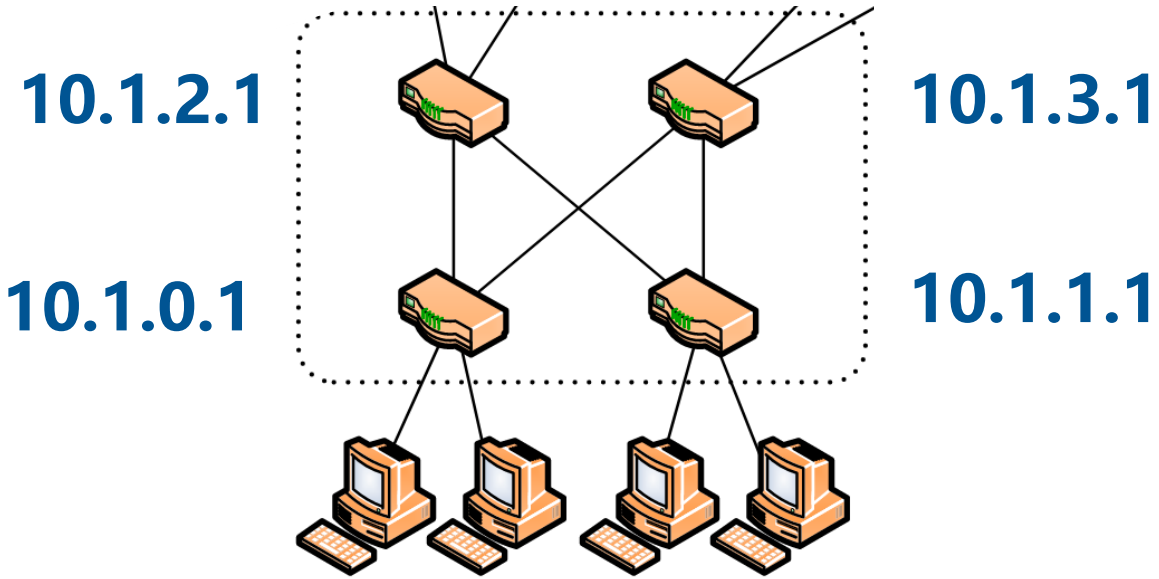
Suffix	Output port
0.0.0.2/8	2
0.0.0.3/8	3

10.2.2.1 的路由表

目的地IP地址为10.2.1.2的传入数据包在端口1上转发，而目的地IP名称为10.3.0.3的数据包则在端口3上转发

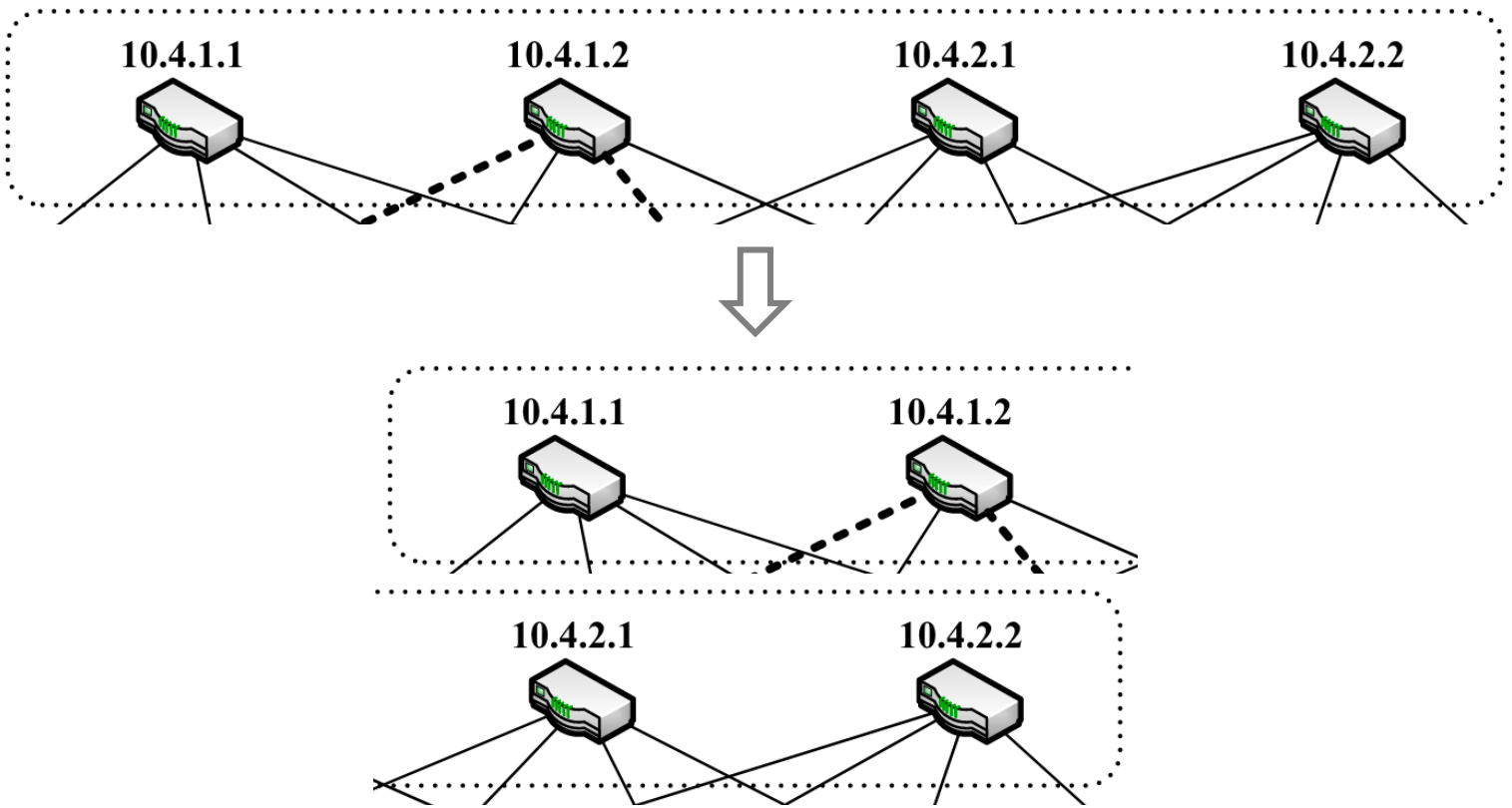
IP 地址编排

- Pod 交换机 IP 地址：$10.pod.switch.1$
  - ✓pod 表示 pod ID（$[0, k-1]$）
  - ✓$switch$ 表示 pod 内的 switch ID（$[0, k-1]$），从左到右，从下到上



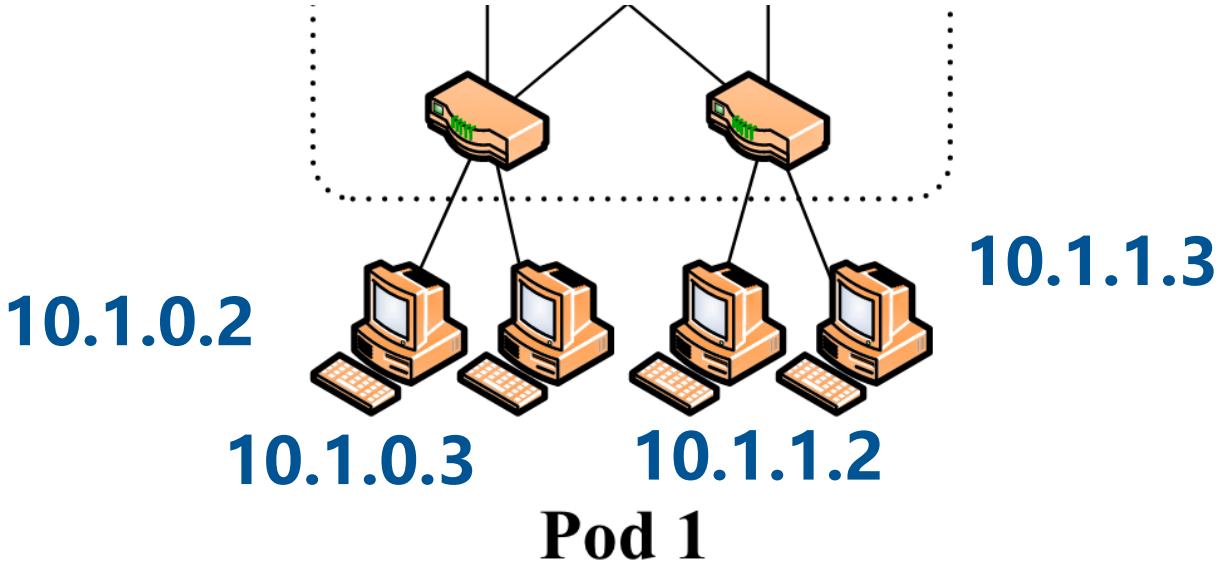
IP 地址编排

- Core 交换机的 IP 地址：$10.k.j.i$
  - ✓j 和 i 表示交换机在 $(k/2)^2$ 个 Core 交换机网格中的坐标，每个在区间 $[1,(k/2)]$



IP 地址编排

- 主机的 IP 地址：$10.pod.switch.ID$
  - ✓ID 表示主机在子网中的 ID（$[2, k/2+1]$）
  - ✓每个 edge 交换机负责 /24 子网中的 $k/2$ 个主机，$k < 256$



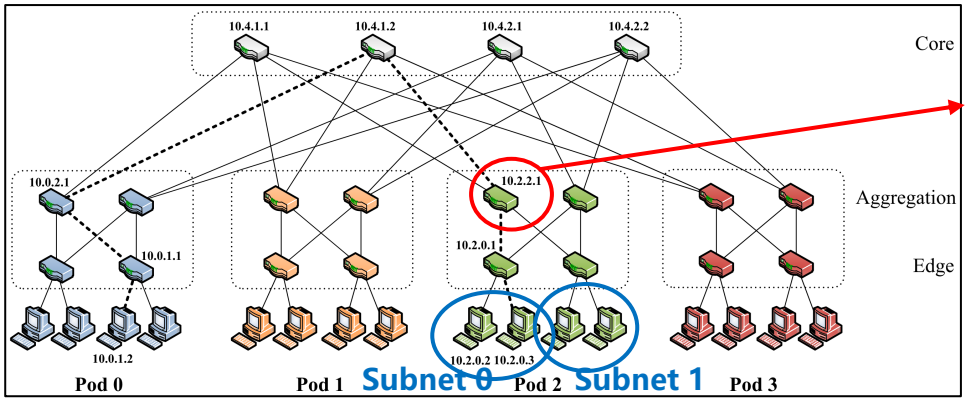
路由表生成

□Aggregation 汇聚交换机路由表

```
1  foreach pod x in [0, k − 1] do
2      foreach switch z in [(k/2), k − 1] do
3          foreach subnet i in [0, (k/2) − 1] do
4              addPrefix(10.x.z.1, 10.x.i.0/24, i);
5          end
6          addPrefix(10.x.z.1, 0.0.0.0/0, 0);
7          foreach host ID i in [2, (k/2) + 1] do
8              addSuffix(10.x.z.1, 0.0.0.i/8,
                   (i − 2 + z)mod(k/2) + (k/2));
9          end
10     end
11 end
```

Prefix	Output port
10.2.0.0/24	0
10.2.1.0/24	1
0.0.0.0/0	

Suffix	Output port
0.0.0.2/8	2
0.0.0.3/8	3



$10.pod.switch.1$

$pod\ x = 2$

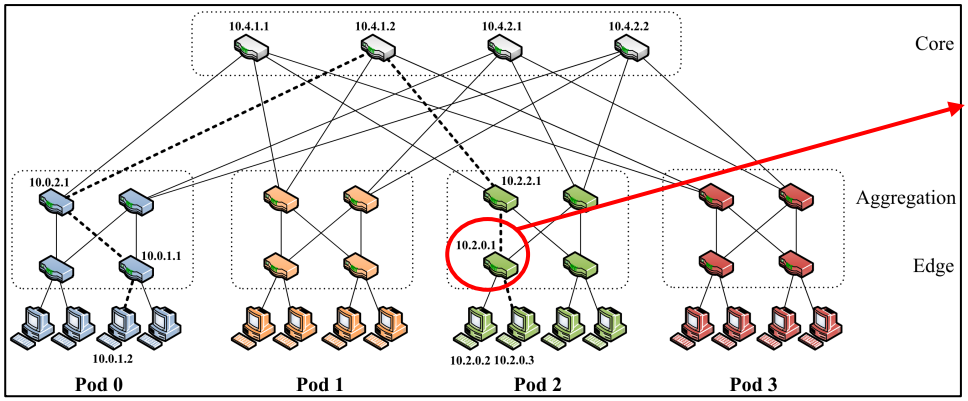
$switch\ z = 2$

路由表生成

□Aggregation 汇聚交换机路由表

```
1  foreach pod x in [0, k − 1] do
2      foreach switch z in [(k/2), k − 1] do
3
4
5
6          addPrefix(10.x.z.1, 0.0.0.0/0, 0);
7          foreach host ID i in [2, (k/2) + 1] do
8              addSuffix(10.x.z.1, 0.0.0.i/8,
                   (i − 2 + z)mod(k/2) + (k/2));
9          end
10     end
11 end
```

Suffix	Output port
0.0.0.2/8	?
0.0.0.3/8	?



$10.pod.switch.1$

$pod\ x = 2$

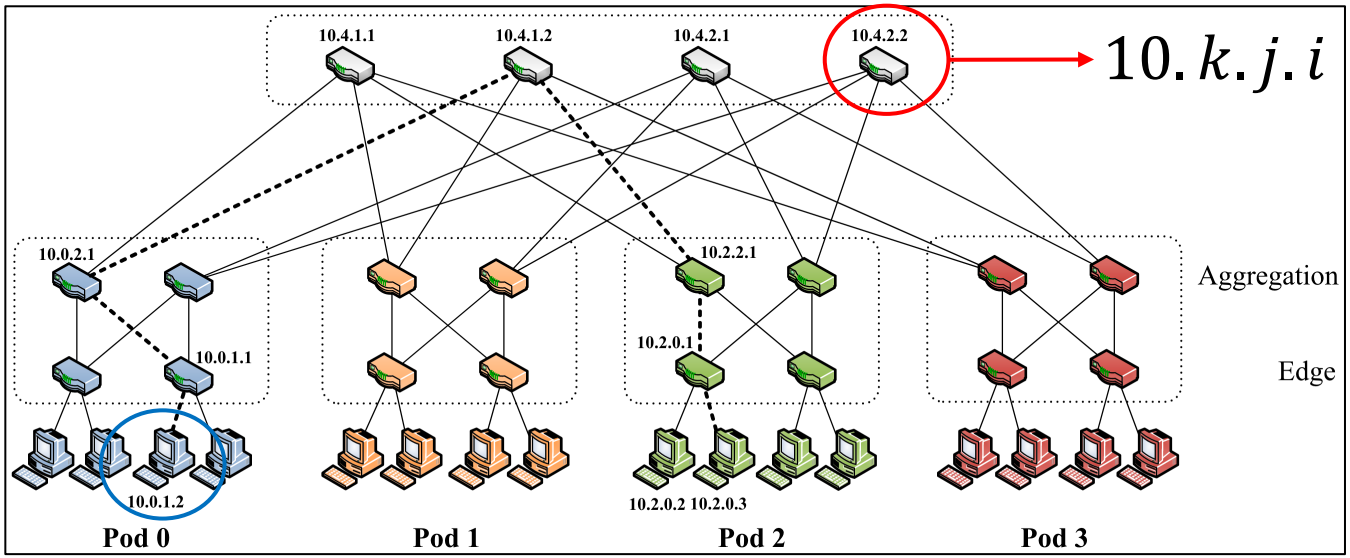
$switch\ z = 0$

路由表生成

□Core（核心）交换机路由表

```
1 foreach j in [1, (k/2)] do
2     foreach i in [1, (k/2)] do
3         foreach destination pod x in [0, (k/2) − 1] do
4             addPrefix(10.k.j.i,10.x.0.0/16, x);
5         end
6     end
7 end
```

Prefix	Output port
10.0.0.0/16	0
10.1.0.0/16	1
10.2.0.0/16	2
10.3.0.0/16	3



$10.k.j.i$

$j = 2$

$i = 2$

传统 Fat-tree 架构的主要不足

- 扩展性问题
- 路由/布线复杂度高
- 成本较高

- 布线复杂性高：k-port 交换机构成的 Fat-tree 需要 $5k^2/4$ 根线缆，k=48 时超过 2800 根
- 核心层交换机数量多：需要 $(k/2)^2$ 个核心交换机，成本随规模平方增长
- 扩展受限于交换机端口数：网络规模被单个交换机端口数锁定，升级需整体替换
- 等价路径利用不充分：ECMP 哈希可能导致流量不均（hash collision问题）
- 不适合非对称流量模式：AI 训练中的 All-Reduce 等模式需要针对性的拓扑优化
- 故障影响范围大：核心层单点故障影响面较广

大量新的网络架构被提出

网络拓扑	规模	带宽	容错性	扩展性	布线复杂性	成本	兼容性	配置开销	流量隔离	灵活性
FatTree	中	中	中	中	较高	较高	高	较高	无	低
VL2	大	大	中	中	较高	较高	中	较高	无	中
OSA	小	大	差	中	较低	较高	低	中	无	高
WDCN	小	大	较好	中	较低	中	中	中	无	高
DCell	大	较大	较好	较好	高	较高	中	较高	无	较高
FiConn	大	较大	较好	较好	较高	中	中	较高	无	较高
BCube	小	大	好	较好	高	较高	中	较高	无	较高
MDCube	大	大	较好	较好	高	高	中	较高	无	较高

AI 时代的数据中心网络

为什么 AI 训练需要特殊的网络?

大模型训练的通信挑战

- GPT-4 级别模型需要数千甚至上万张 GPU 协同训练
- **数据并行 (Data Parallelism)**：每个 GPU 计算梯度后需要 All-Reduce 同步 → 大量集合通信
- **模型并行 (Model/Tensor Parallelism)**：模型切片分布在不同 GPU → 需要超低延迟、高带宽通信
- **流水线并行 (Pipeline Parallelism)**：不同层在不同设备 → 需要高效的前向/反向传递通信

400-800 Gbps
（单节点）
带宽需求

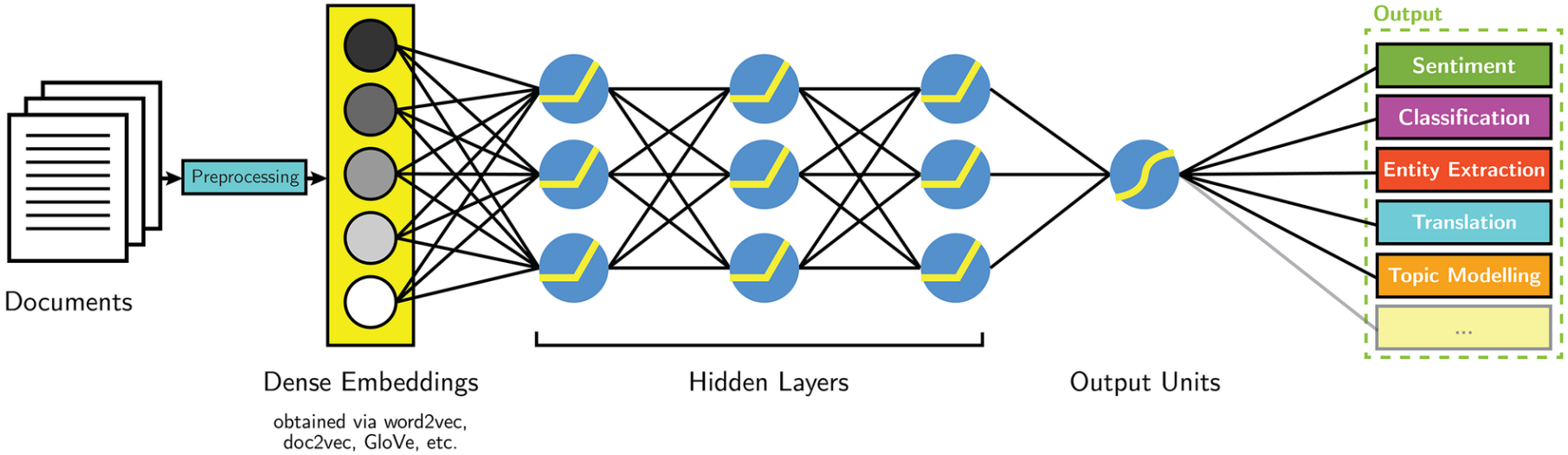
< 5 μs
（GPU间通信）
延迟要求

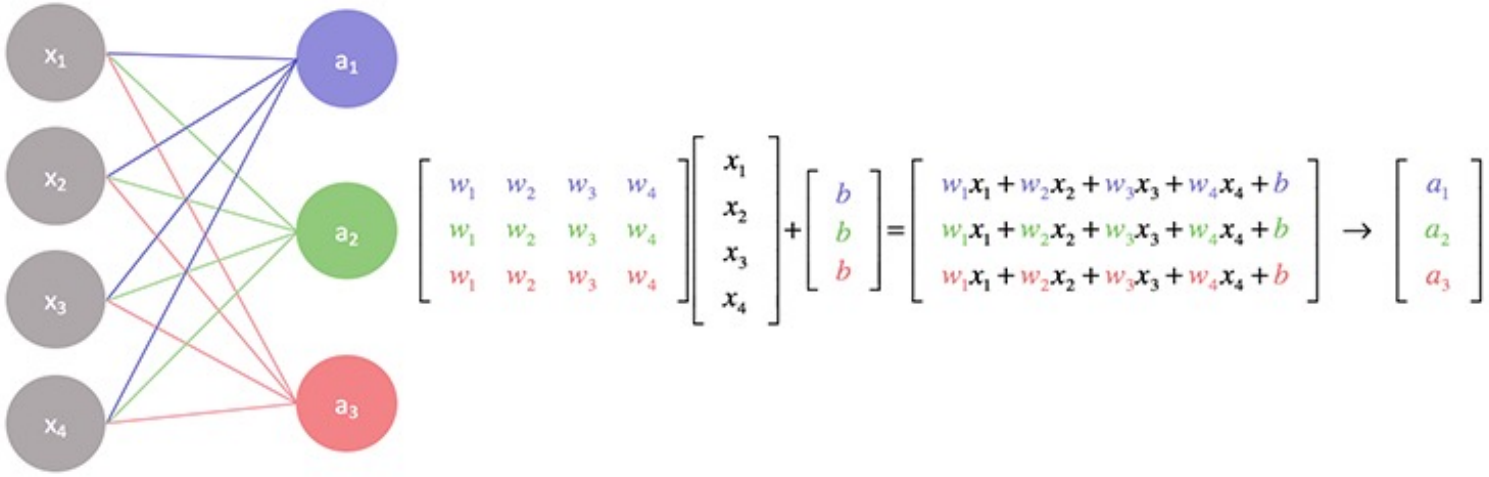
数千~数万
（单任务）
GPU 集群规模

30-50%
（训练时间）
通信占比

参考：Shoeybi et al., Megatron-LM (2019); Narayanan et al., Efficient Large-Scale Language Model Training on GPU Clusters (2021)

神经网络模型训练过程


Output
Sentiment
Classification
Entity Extraction
Translation
Topic Modelling
...
Preprocessing
Documents
Dense Embeddings
obtained via word2vec,
doc2vec, GloVe, etc.
Hidden Layers
Output Units


x1
x2
x3
x4
a1
a2
a3
$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \rightarrow \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$

数据并行 (Data Parallelism)

核心思想

- 每个 GPU 持有完整的模型副本
- 训练数据被分割成多份 (shards)
- 每个 GPU 独立计算不同数据批次的梯度
- 通过 All-Reduce 同步梯度
- 所有 GPU 更新后保持一致

优点

✓ 实现简单，框架支持广泛

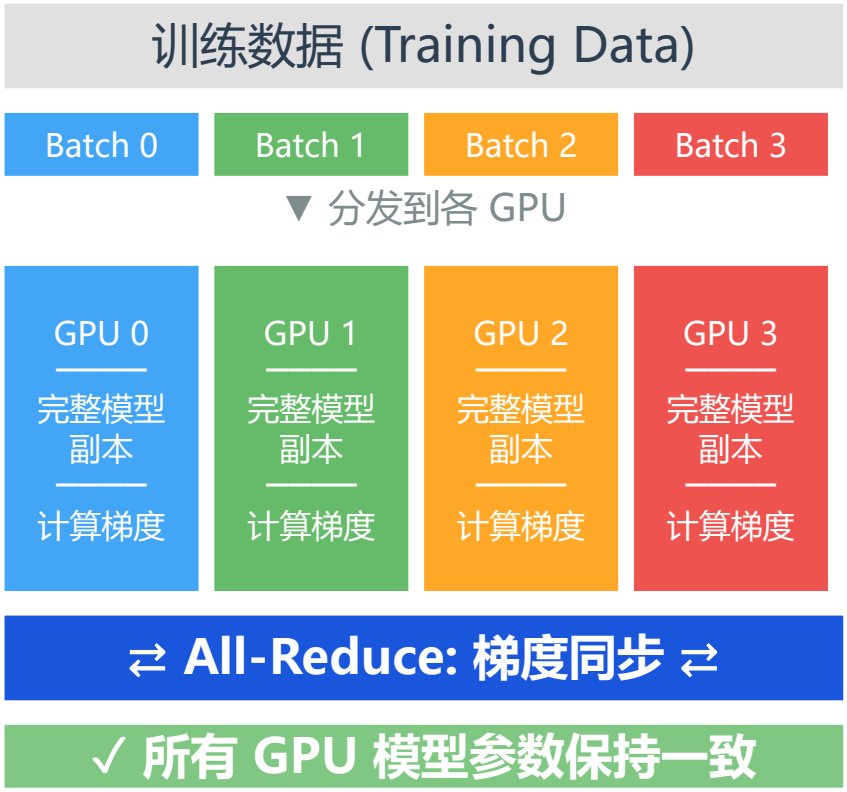
✓ 几乎线性的加速比（理想情况）

缺点

✗ 每个 GPU 都需存储完整模型 → 内存瓶颈

✗ All-Reduce 通信开销随 GPU 数增加

数据并行示意图



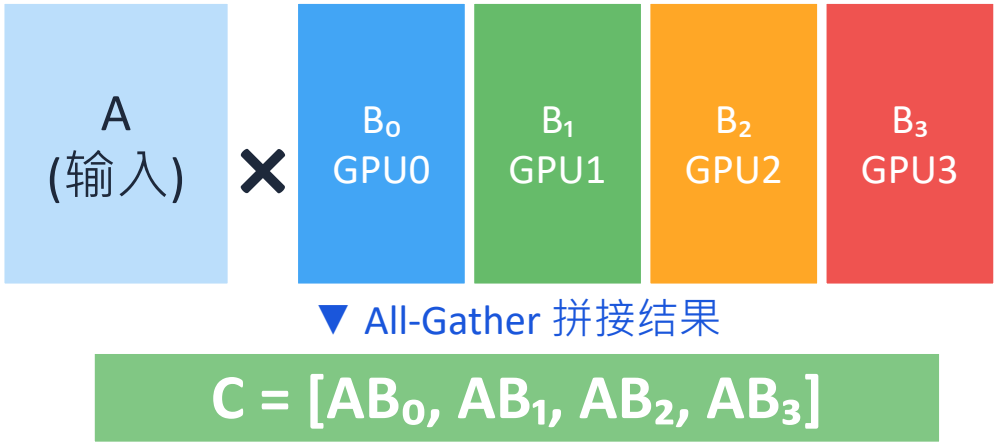
参考：Li et al., "PyTorch Distributed: Experiences on Accelerating Data Parallel Training", VLDB 2020

张量并行

张量并行 (Tensor Parallelism)

单层的权重矩阵切分到多个 GPU

- 例：矩阵乘法 C = A × [B0, B1, B2, B3]
- 每个 GPU 计算部分结果，然后 All-Gather 拼接
- 适用于节点内（NVLink 高带宽）



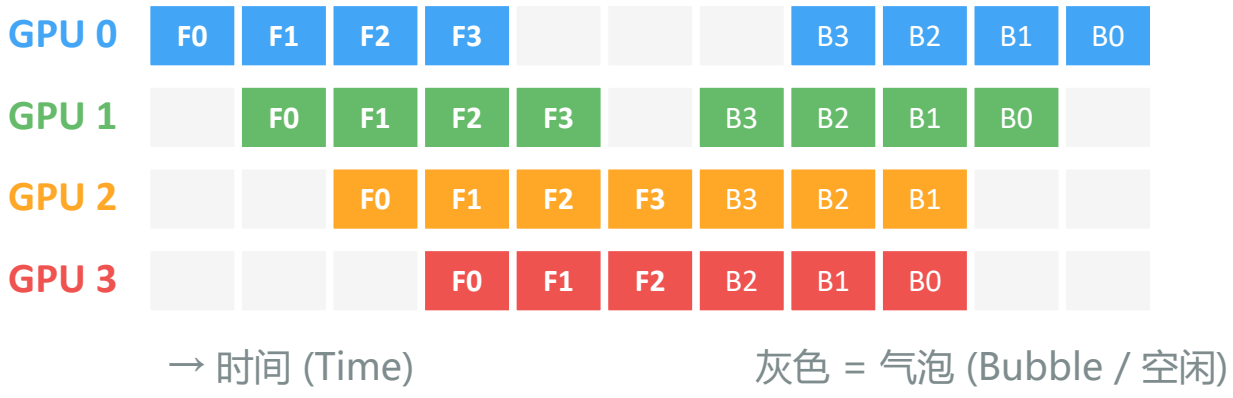
参考：Megatron-LM (Shoeybi et al., 2019)

流水线并行

流水线并行 (Pipeline Parallelism)

模型按层切分到不同 GPU

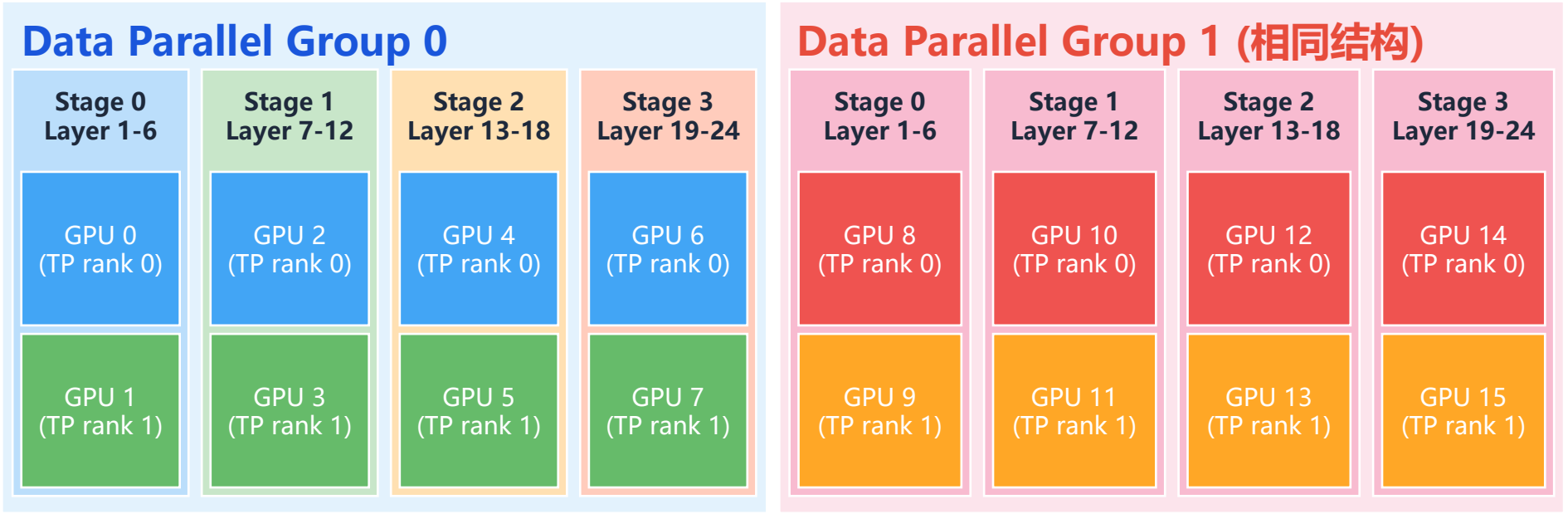
- 例：24 层模型，4 个 GPU，每个处理 6 层
- 微批次 (micro-batch) 流水线执行
- 气泡 (bubble) 是主要开销 → 需要优化调度



参考：GPipe (Huang et al., 2019); PipeDream (Narayanan et al., 2019)

3D 并行与专家并行

大规模模型训练通常组合使用多种并行策略：总 GPU 数 = DP × TP × PP



● Data Parallel: 模型副本×2，数据分割 ● Pipeline Parallel: 模型分 4 阶段 ● Tensor Parallel: 每层分 2 个 GPU

💡 **总计：2 (DP) × 4 (PP) × 2 (TP) = 16 个 GPU ｜ 实际中可扩展到数千个 GPU (如 Megatron-LM 3D 并行)**

🧩 专家并行 (Expert Parallelism) — MoE 模型的第四种并行

Mixture-of-Experts 模型（如 DeepSeek-V3, Mixtral）中，不同专家分布在不同 GPU。每个 token 只路由到部分专家，通过 All-to-All 通信分发和收集。

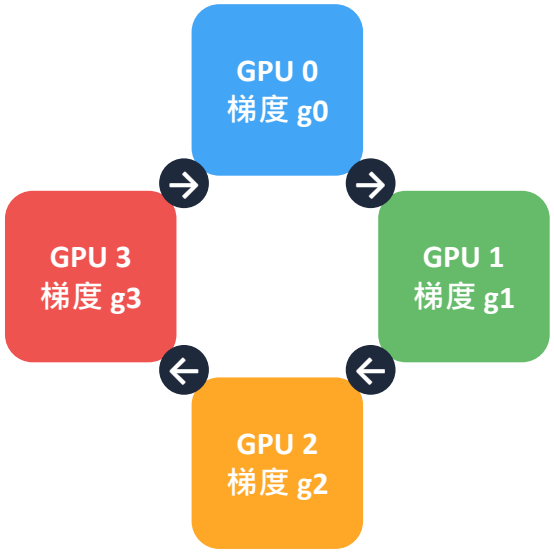
参考：GShard (Lepikhin et al., 2020); Switch Transformer (Fedus et al., 2021); DeepSeek-MoE (Dai et al., 2024)

集合通信 (Collective Communication)

分布式训练中的核心通信模式

- **All-Reduce**
  所有节点的数据聚合并广播回所有节点
  用于数据并行中的梯度同步（最常见）
- **All-Gather**
  每个节点的数据被收集到所有节点
  用于模型并行中的参数收集
- **Reduce-Scatter**
  聚合后分散到不同节点
  常与 All-Gather 组合使用
- **All-to-All**
  每个节点向所有其他节点发送不同数据
  用于专家混合模型 (MoE) 的路由

Ring All-Reduce



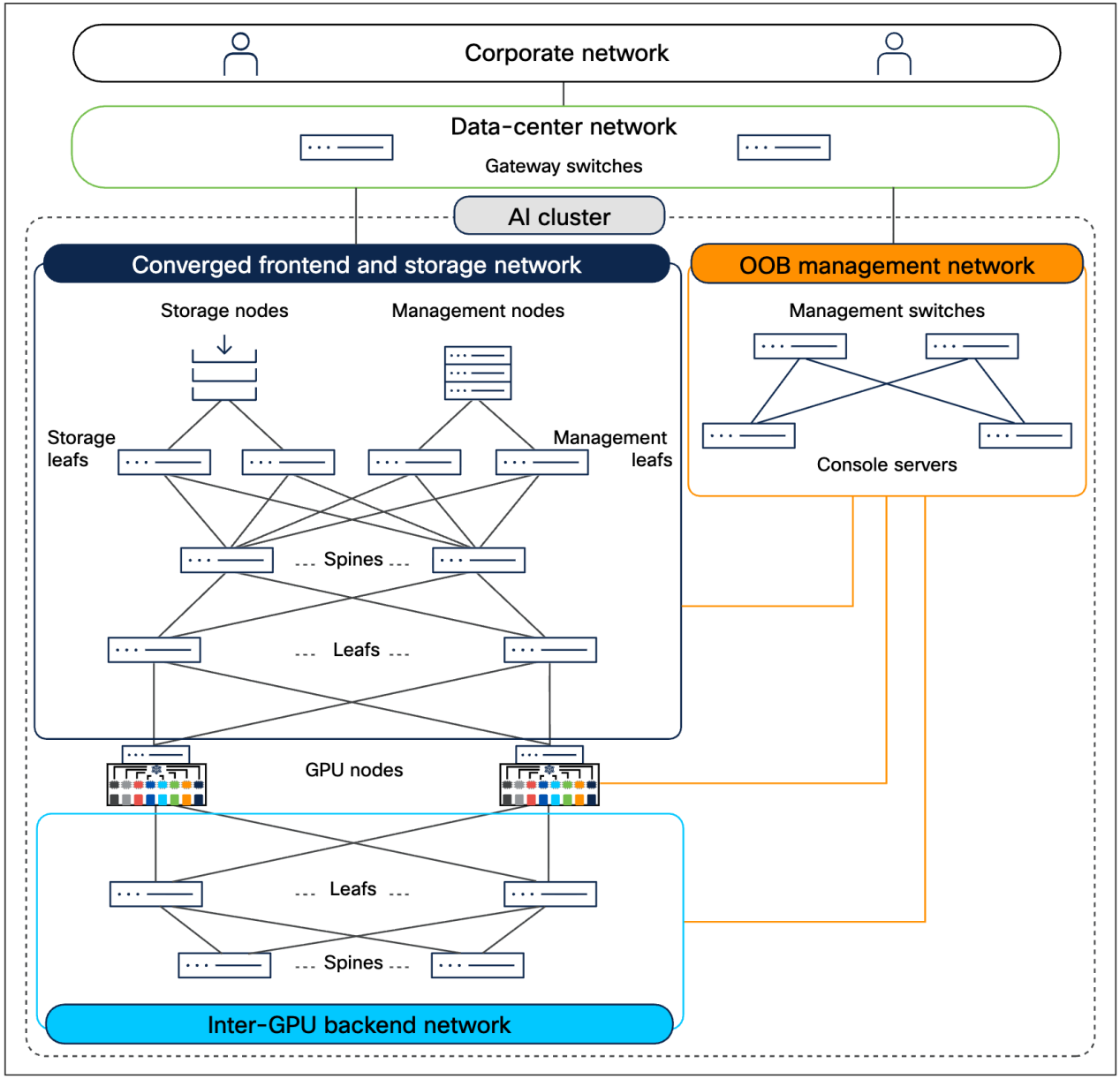
Ring All-Reduce 步骤

1. **Scatter-Reduce**
   每个 GPU 将梯度分成 N 份，
   沿环传递并累加
2. **All-Gather**
   将合并后的结果再沿环
   传递，每个 GPU 获得完整结果

通信量：2(N-1)/N × 模型大小
步数：2(N-1)，与 GPU 数线性增长

实现：NVIDIA NCCL, Gloo, Horovod

典型的 AI 集群架构


Corporate network
Data-center network
Gateway switches
AI cluster
Converged frontend and storage network
OOB management network
Storage nodes
Management nodes
Management switches
Storage leafs
Management leafs
Console servers
Spines
Leafs
GPU nodes
Leafs
Spines
Inter-GPU backend network

RDMA 与 InfiniBand

RDMA (Remote Direct Memory Access)

绕过操作系统内核，直接访问远程内存

零拷贝 (Zero-copy)：数据不经过 CPU

内核旁路 (Kernel Bypass)：减少系统调用开销

延迟低至 ~1-2 μs（传统 TCP 为 ~50 μs）

三种实现方式：

- InfiniBand：专用 RDMA 网络，性能最优
- RoCE v2：基于以太网的 RDMA，成本较低
- iWARP：基于 TCP 的 RDMA，兼容性好

InfiniBand 在 AI 集群中

当前 AI 训练集群的主流互联技术

最新标准：NDR 400Gbps / XDR 800Gbps

NVIDIA 主导（2019 年收购 Mellanox）

典型配置：

- 每个 GPU 节点 8×GPU + 8×400G IB 网卡
- 多级 Fat-tree IB 交换网络
- NCCL 库优化集合通信

代表案例：

- Meta AI 集群：16K H100 + 400G IB
- Microsoft Azure：ND H100 v5

GPU 节点内互联：NVLink 与 NVSwitch

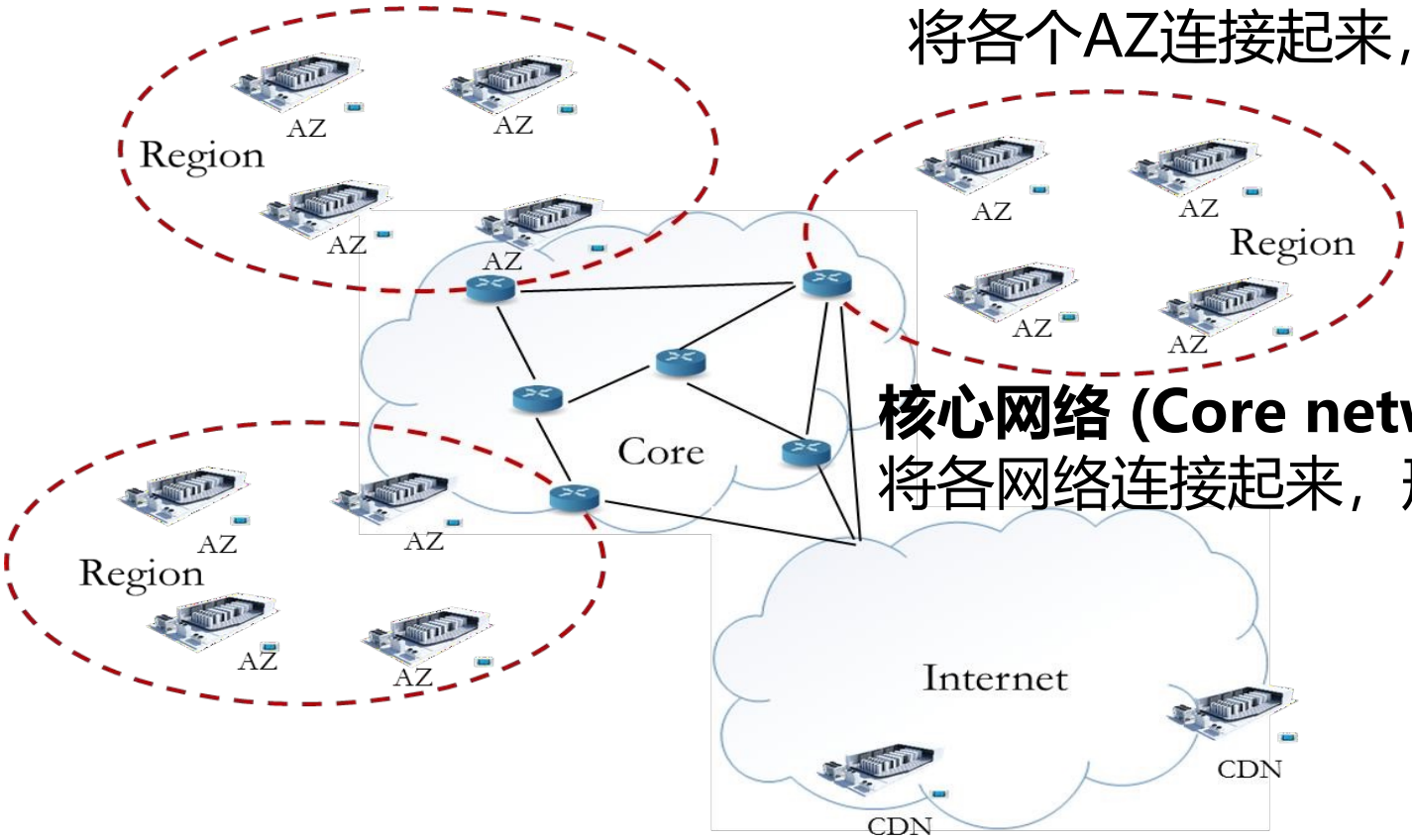
节点内 GPU 互联技术演进

- PCIe：传统方案，Gen5 约 64 GB/s，延迟高、带宽不足
- NVLink：NVIDIA 专有高速互联，第4代 NVLink 单链路 900 GB/s 双向
- NVSwitch：全互联交换芯片，8 GPU 间任意两个均可全速通信
- DGX/HGX 架构：8×GPU + NVSwitch 组成一个“超级节点”，节点内通信远快于节点间

	PCIe Gen5	NVLink 4	NVSwitch
带宽	64 GB/s	900 GB/s	全互联 900 GB/s
延迟	~μs 级	~ns 级	~ns 级
GPU 连接	通过 CPU	点对点	全对全
成本	低	中	高

内容分发网络

一朵云的主要网络组成部分



区域网络 (Regional network)
将各个AZ连接起来，以覆盖较大范围

核心网络 (Core network)
将各网络连接起来，形成更大范围整体

边缘或内容分发网络 (Edge/CDN)
连接了区域网络和公网或服务提供商

从端到云的数据传输困境

- **Using Satellite/Microwave in remote areas?**

5 Mbps x 2 Mbps Usage over 100GB is $0.14 per MB	**$9,000**/month
5 Mbps x 2 Mbps Usage over 250GB is $0.14 per MB	**$19,800**/month
5 Mbps x 2 Mbps Usage over 500GB is $0.14 per MB	**$40,500**/month

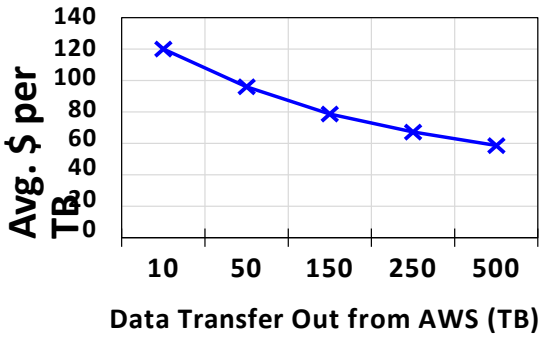
Source: http://www.groundcontrol.com/

- **Accelerate bulk data movement using HDD?**

Microsoft < 4 TB per HDD, $80 per disk

amazon $2.49 per data-loading-hour

Google $80 per disk



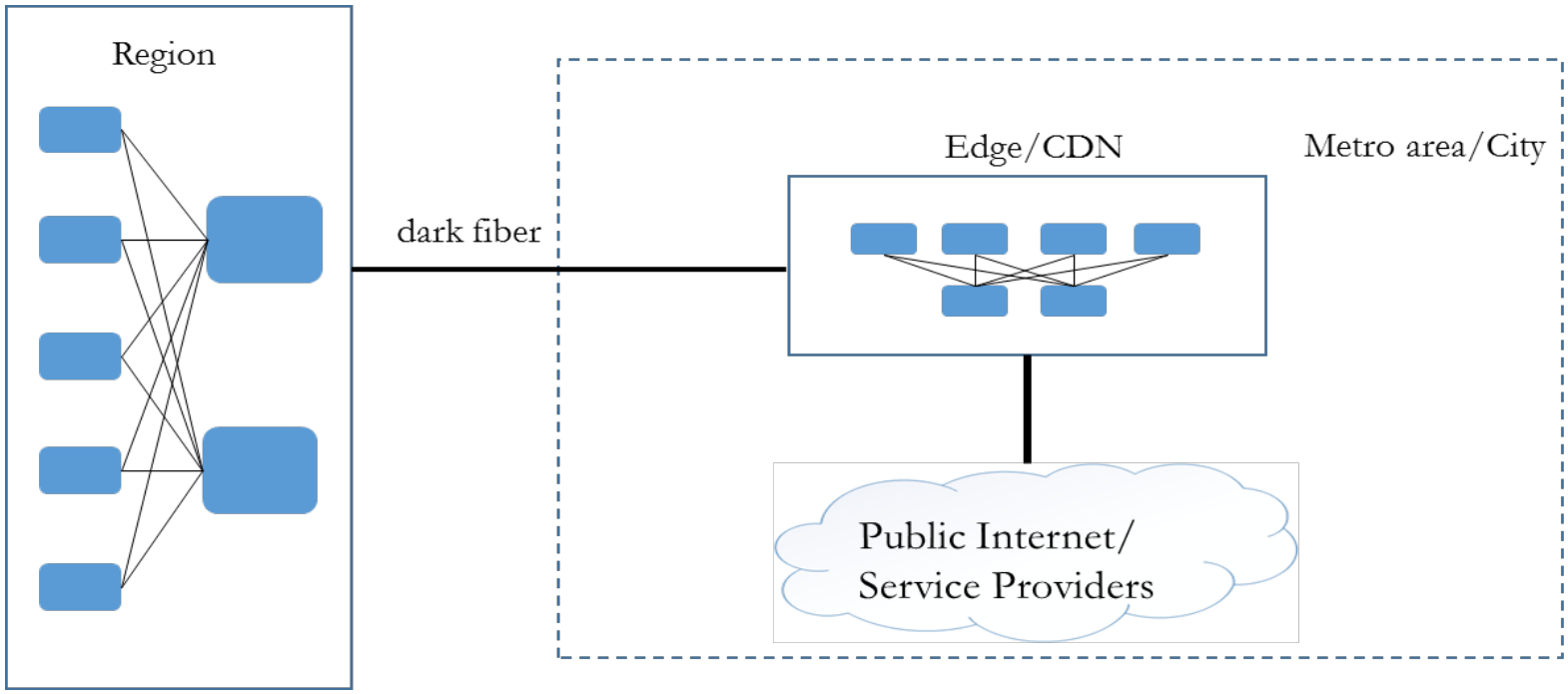
华为云 CDN

了解内容分发网络 CDN

10亿+	2,800+	150Tbps+	7*24小时
为超过十亿华为手机用户提供全球加速服务	中国大陆2000+加速节点，中国大陆境外800+加速节点	海量带宽储备，全网带宽能力高达150Tbps	全天候全网健康度管理，基于服务质量智能精准调度

边缘或内容分发网络 CDN

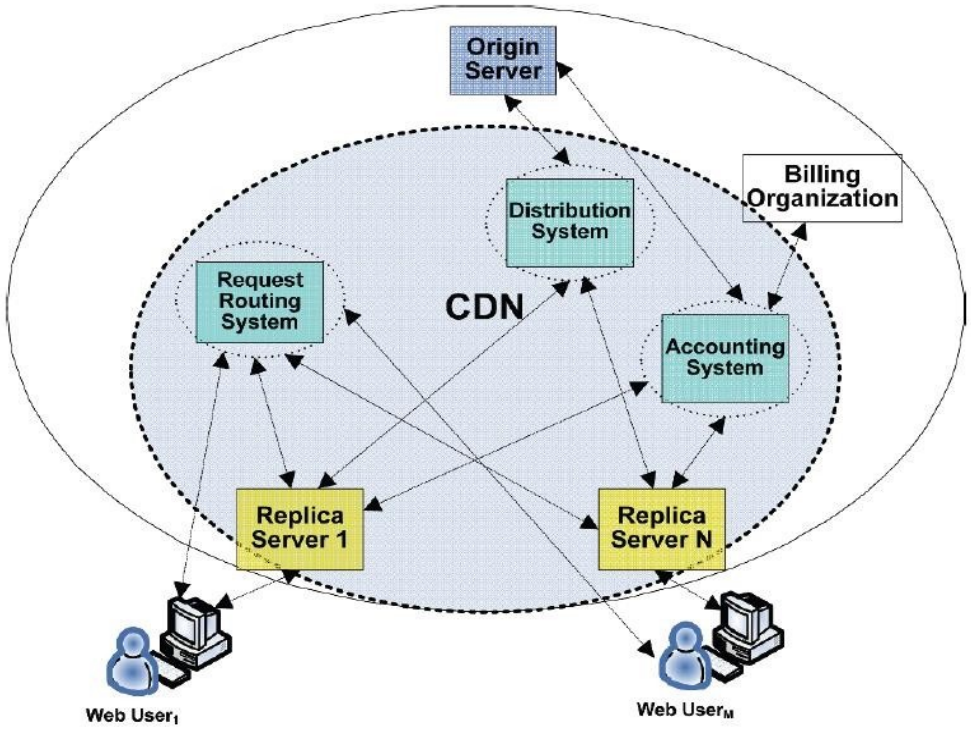
- 一般在特定地理区域，多是人口较为稠密的城区
- Edge 指设立的 PoP（Point of Presence, 即入网点）
- 用户访问网站时，CDN 通过算法选择地理最近的 PoP



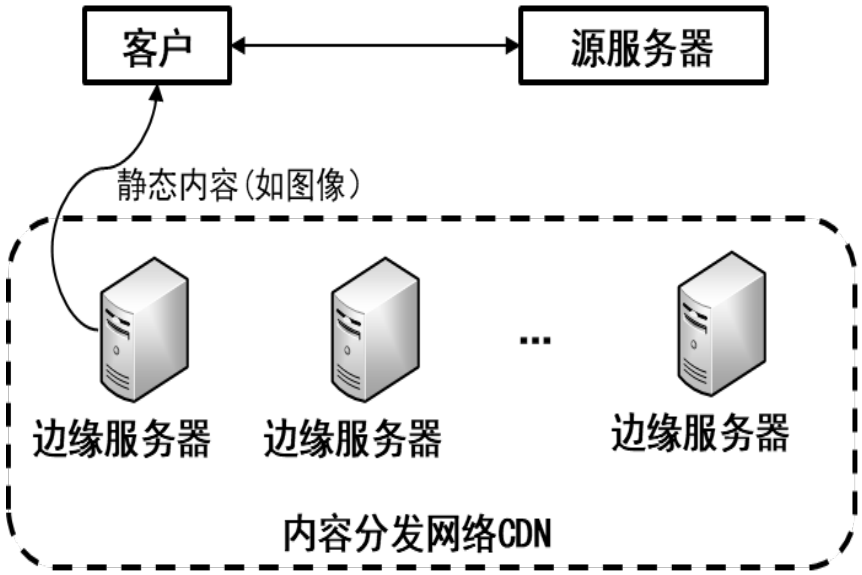
边缘或内容分发网络

- **内容分发网络 CDN (Content Delivery Network)**
  - 互联网基础之上放置节点服务器所构成的的一层智能虚拟网络

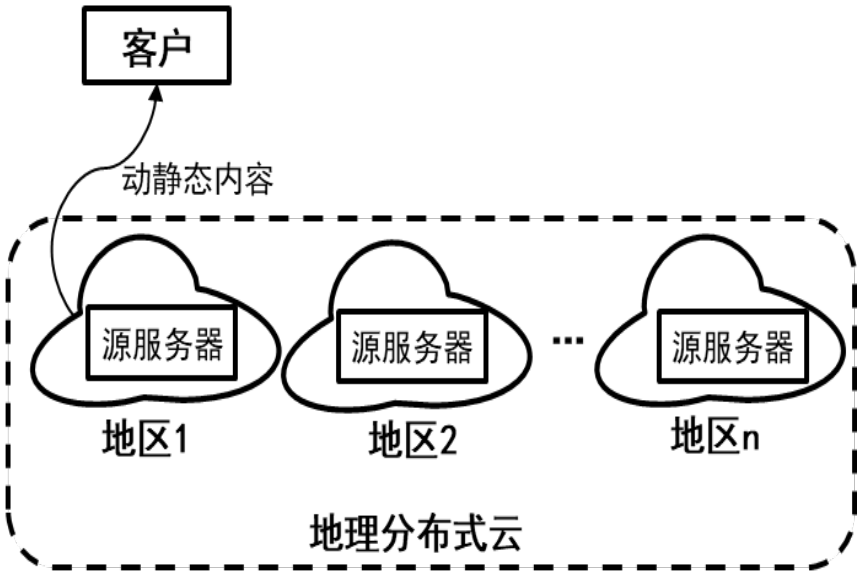
内容分发组件 (Content Delivery Component) - 包含原始服务器和备份服务器
请求路由组件 (Reuest Routing System) - 负责转发给合适的边界服务器
分布式组件 (Distribution System) - 负责内容迁移，并保证一致性
计费组件 (Accounting System) - 访问日志、流量报告



CDN 技术对比地理分布式云

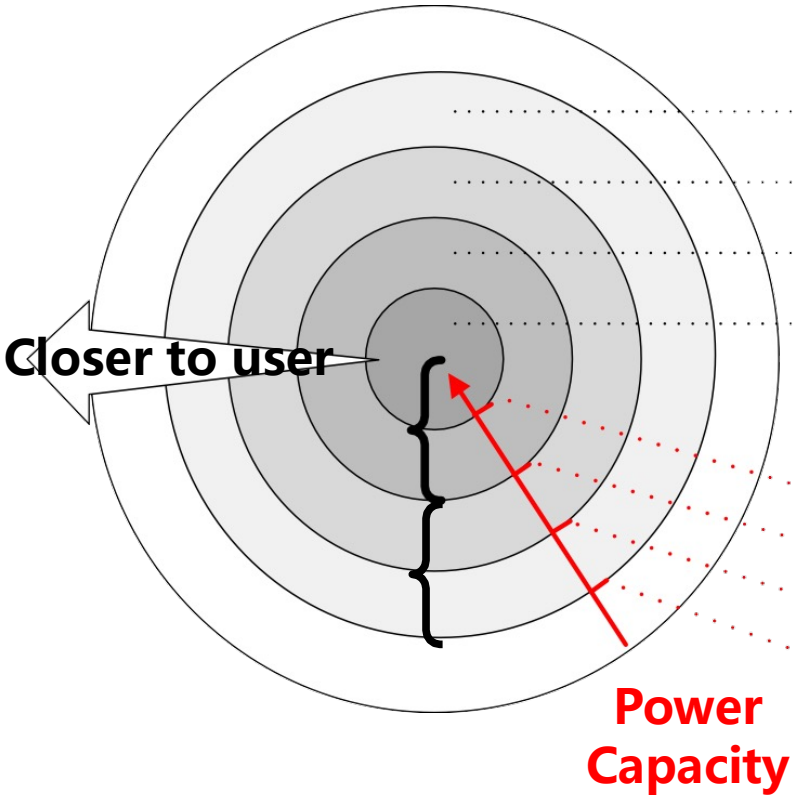


a) 基于CDN的服务



b) 基于地理分布式云的服务

不同层次的数据中心基础设施



- Mini data center at the network edge
- Local, small server cluster
- Distributed, modestly-sized data center
- Centralized mega data center

- Several MWs ~ tens of MWs
- Tens of KWs ~ hundreds of KWs
- Hundreds of Watts ~ several KWs
- Less than a watt ~ tens of watts

中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

谢谢

陈壮彬

软件工程学院

chenzhb36@mail.sysu.edu.cn