

系统域互连网络

——基础理论和案例分析

杨帆

yangfan2020@ict.ac.cn

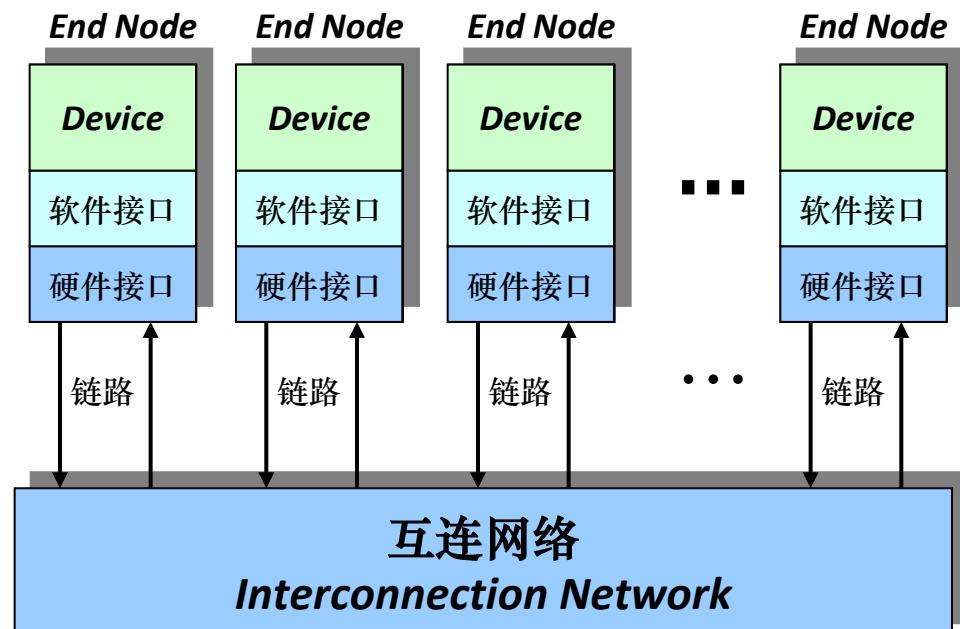
参考书籍

- *Principles and Practices of Interconnection Networks*, Elsevier, William James Dally and Brian Patrick Towles
- *Appendix E, Interconnection Networks, Computer Architecture: A Quantitative Approach*, 4th edition, Hennessy and Patterson
- *Interconnection Networks—An Engineering Approach*, Revised Printing, Jos 'e Duato

什么是系统域互连网络？

什么是互连网络

- 互连网络将不同的装置连接到一起，并允许装置之间相互通信
- 装置 (*Device*)
 - 单个计算机内的组件
 - 单个计算机
 - 多机系统
- 关联单元
 - 终端节点
(*End Node = Device + Interface*)
 - 链路 (*Link*)
 - 互连网络
(*Interconnection Network*)
- 用尽可能少的消耗（时间、成本、功耗）传输尽可能多的信息



不同种类的互连网络(1/2)

- 片上网络 (*OCN or NoC*)

- 连接单个芯片内的不同单元，如cache、寄存器堆、处理器核等
- 连接单元数目通常在10~100，例如多核处理器
- 特征尺寸：毫米
- 举例：**ARM AMBA**总线

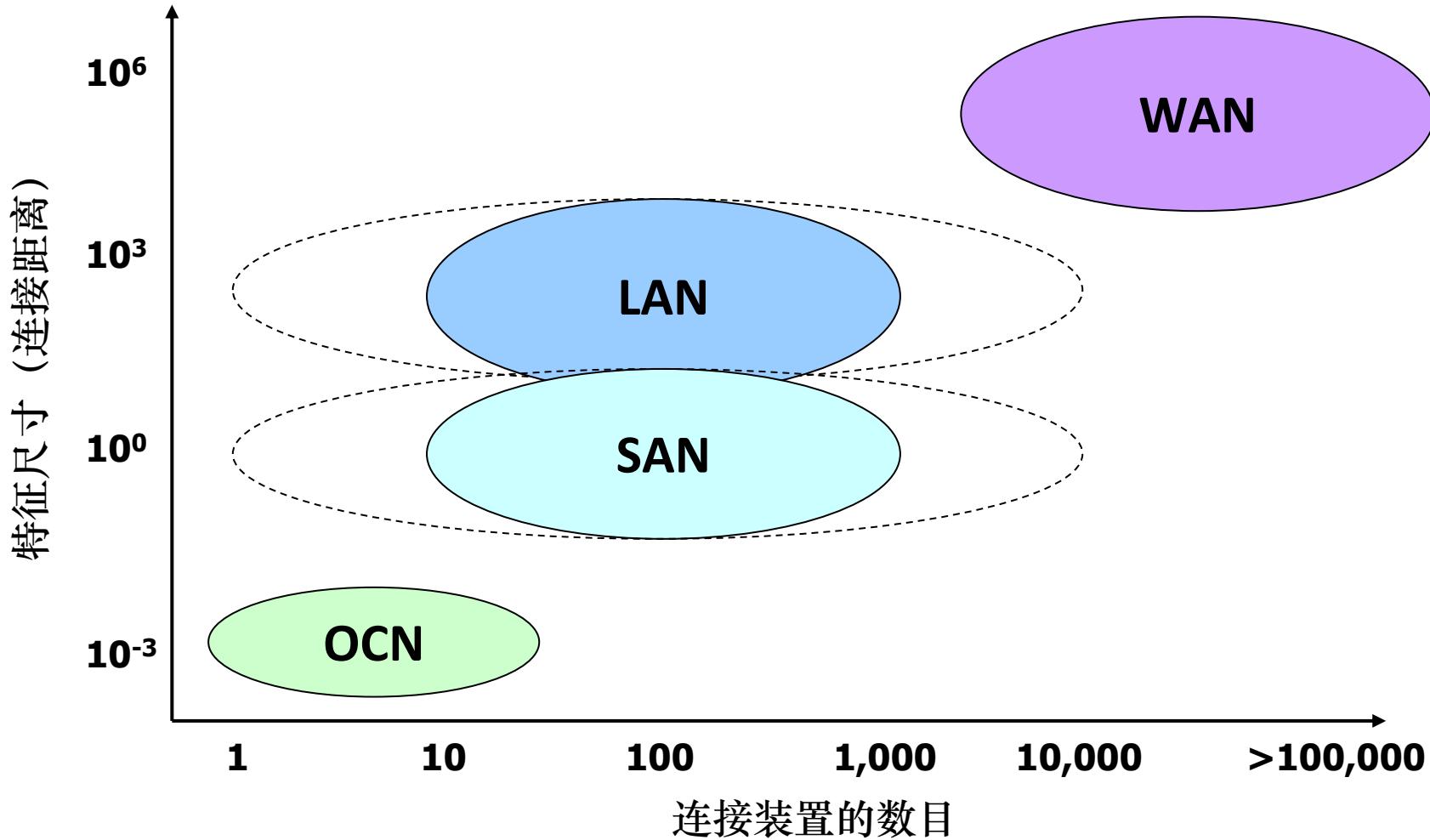
- 系统域网络 (*SAN*)

- 连接单个计算机系统内的不同组件，这里的系统可以是单机系统（多路服务器），也可以是多机系统（如超级计算机）
- 多机系统内连接单元数目通常在1000~100000，例如IBM Blue Gene/L超级计算机内的64000个节点
- 特征尺寸：十米到百米
- 举例：**Infiniband**、**Myrinet**

不同种类的互连网络(2/2)

- 局域网 (**LAN**)
 - 连接多个自治的计算机系统，例如：校园网里的多台计算机
 - 连接单元数目通常在100~1000，通过交换机
 - 特征尺寸：数米到数千米
 - 举例：*Ethernet*
- 广域网 (**WAN**)
 - 在广阔的范围内连接不同的系统（计算机系统或网络系统）
 - 连接单元数目通常在数百万
 - 特征尺寸：数十千米到数百千米
 - 举例：城域网、程控交换机

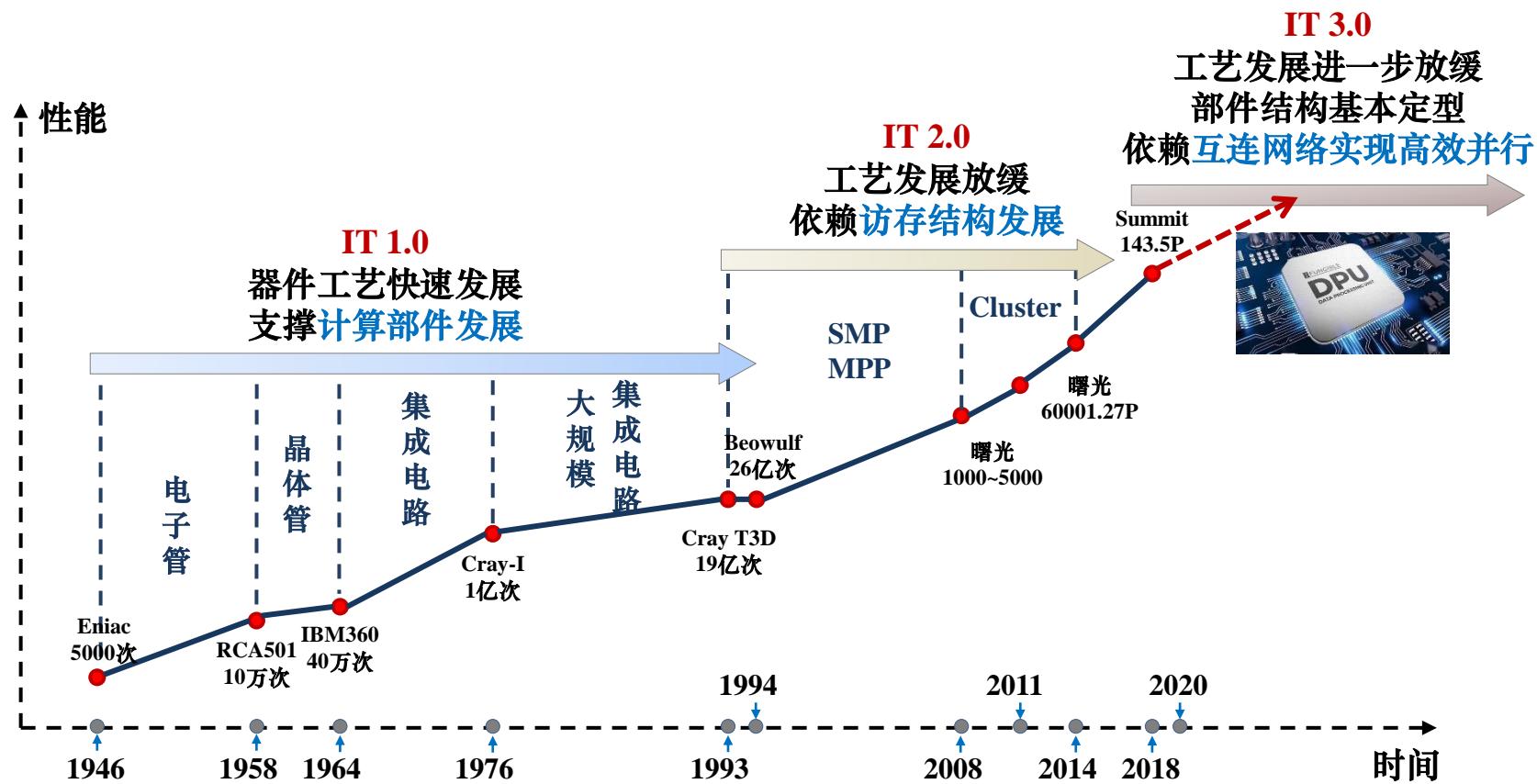
界定系统域互连网络的研究范畴



为什么研究系统域网络

- 课程需求
- 实际系统需求
 - 高性能计算需求—超级计算机
 - 大数据处理需求—仓储级数据中心
 - 人工智能需求—智能计算机

互连网络将成为新一代系统结构演进的发力点



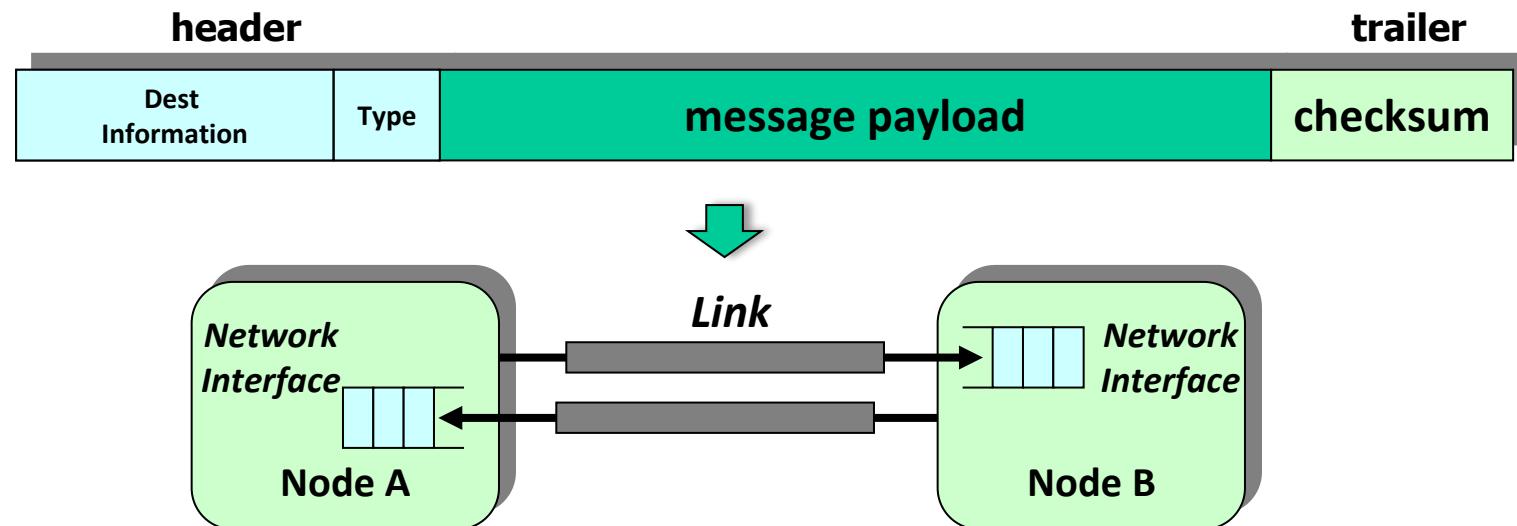
如何研究系统域网络

- 不同互连网络共享同一套研究要素，系统域场景具体分析
 - 接口、拓扑、路由、流控、交换、链路
- 首先解决点对点通信
- 然后解决多对多通信

解决系统域网络的点对点通信问题

系统域点对点通信基本元素

- **节点**: 对应机群或MPP架构中的单个节点
- **网络接口**: 网络接口卡, 如万兆以太网卡、Infiniband HCA
- **链路**: 包含线缆和接插件两部分, 例如光纤+光模块
- **网络包**: 网络传输的基本逻辑单元, 分为包头、载荷、包尾三部分



系统域网络接口

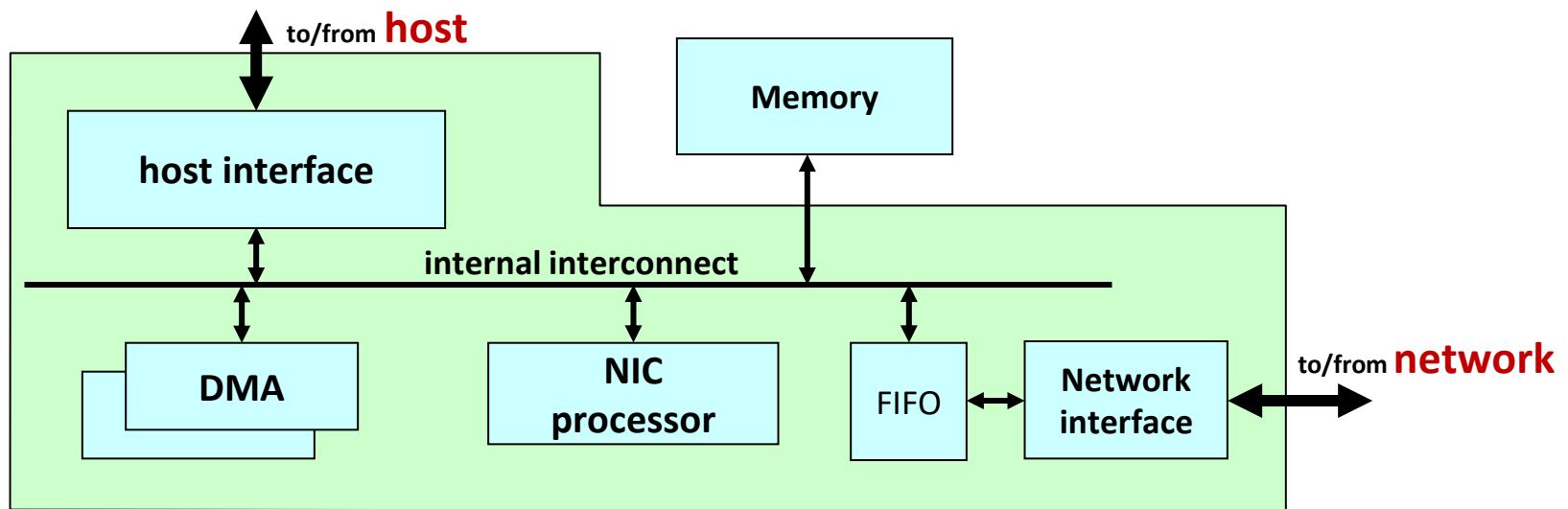
- 网络接口顾名思义，是主机节点和互连网络的桥接接口

- **主机端**

- 与节点内其他硬件（CPU、内存）交互，一般通过PCIe总线；
 - 与节点上部署的软件（OS、应用）交互，Register、Buffer等；

- **网络端**

- 网络包接收与发送，如组包、解包功能，流控功能等；
 - 网络连接状态管理，可靠网络流Request/Response，不可靠数据报等；



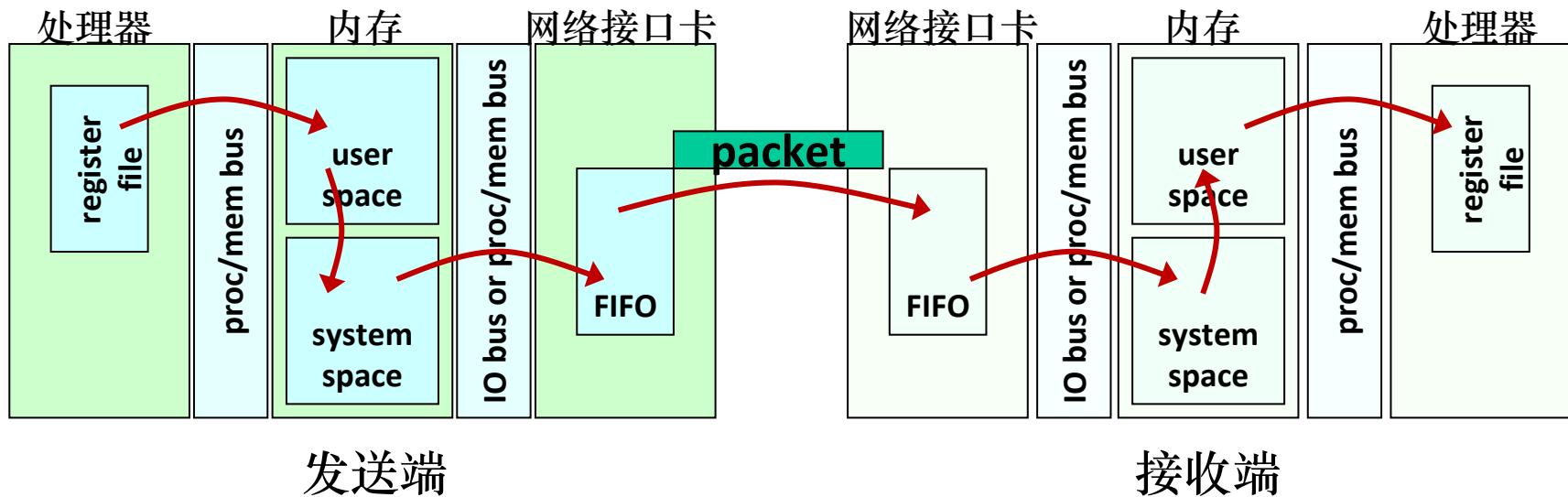
网络接口的工作流程示例

• 发送端

- 应用在用户空间准备好数据，通过系统调用启动网络发送
- 数据从用户空间拷贝入内核空间，再通过DMA送到网卡上打包发送

• 接收端

- 网卡接收，解包后通过DMA将数据发送到内核缓冲区，产生中断
- OS将数据从内核空间拷贝入用户空间，返回用户程序处理流程



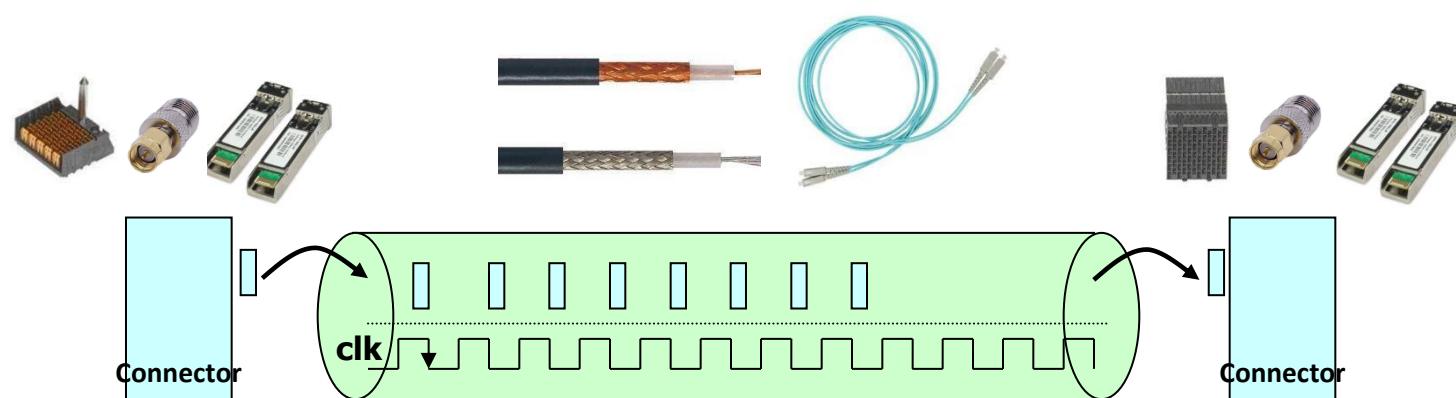
物理链路介绍

- **传输介质**

- 短距离：铜线、同轴电缆
- 长距离：五类双绞线、光纤

- **物理信号**

- 编码格式：8B/10B、64/66、128/130，平衡位流中的1和0，接收端正确还原信号（计算机、电子信息领域）
- 线路驱动：串并转换、预加重与去加重、均衡（增强高频信号传输质量）（微电子领域）



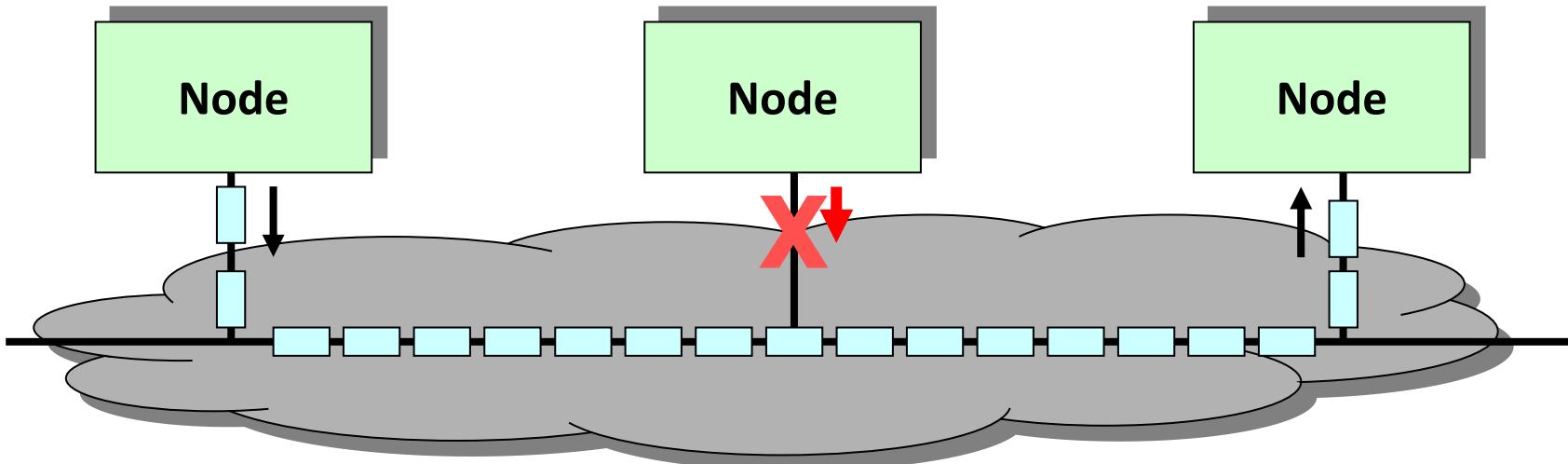
解决系统域网络的多对多通信问题

多对多通信主要考虑的问题

- 拓扑
 - 表征节点和链路的静态连接关系
- 路由
 - 决定数据包在网络拓扑中的传输路径
- 流控
 - 决定数据包被传输的时机和数量
- 交换
 - 解决不同数据包之间的碰撞关系

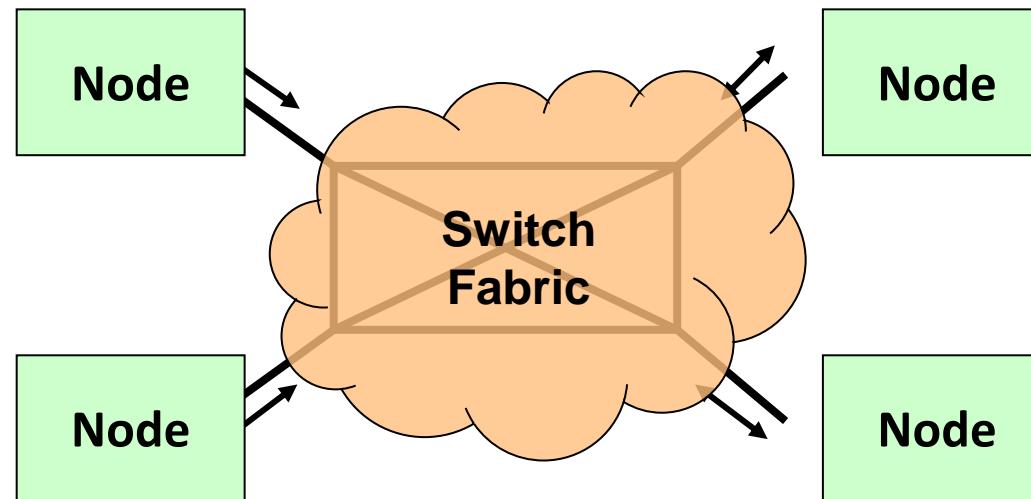
共享媒介网络

- 网络媒介被所有节点以单工或双工模式共享
 - 拓扑：一维总线或二维环形总线
 - 路由：广播或定向发送
 - 流控：N/A
 - 交换：CSMA/CD（时分复用）



交换媒介网络

- 网络媒介允许节点之间有独立的连接路径，相互发送数据
 - 拓扑：基于Switch/Router的连接图
 - 路由：确定路径路由/自适应多路径路由
 - 流控：On/Off、基于信用的流控
 - 交换：数据缓冲模式、仲裁模式、Crossbar



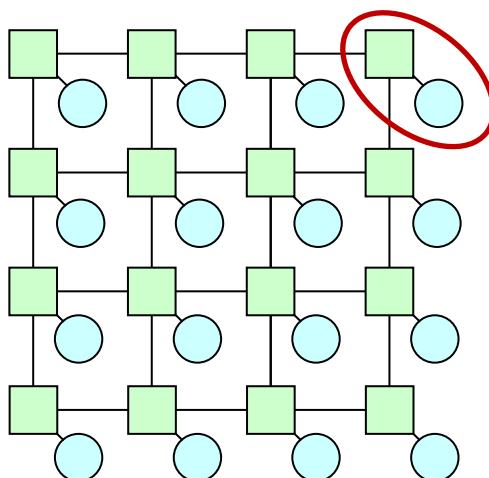
两种网络的比较

- 基于共享媒介的网络
 - 成本：较低
 - 性能：受系统总线带宽制约，无法随节点个数增加而扩展
 - 实现难度：简低或中等
- 基于交换媒介的网络
 - 成本：较高
 - 性能：随节点个数增加可扩展、但与具体实现紧密相关
 - 实现难度：较高
- 本课程重点研究基于交换媒介的网络
 - 从拓扑、路由、交换+流控三方面进行考察

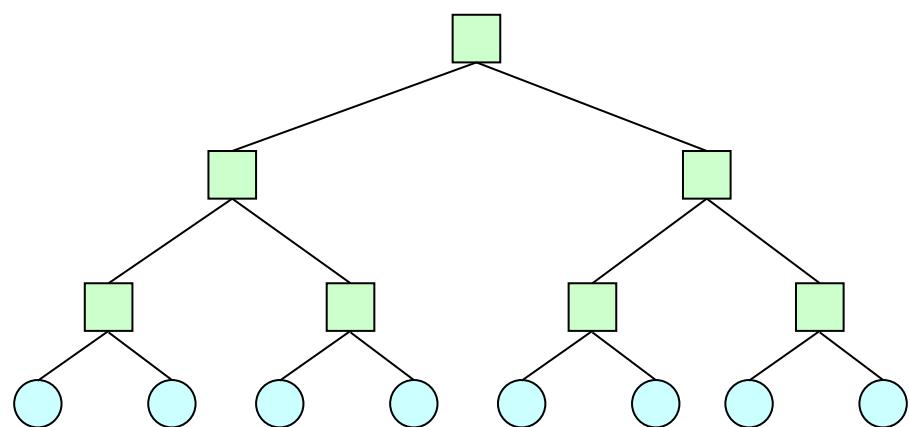
网络拓扑结构

网络拓扑结构

- 拓扑结构是设计交换网络的**第一步**，拓扑可以分为两类
 - 直接网络拓扑结构
 - 计算部件和交换部件紧密集成在一起，节点之间直接互连
 - 间接网络拓扑结构
 - 使用独立的交换机/路由器，节点之间间接互连



直接网络拓扑结构



间接网络拓扑结构

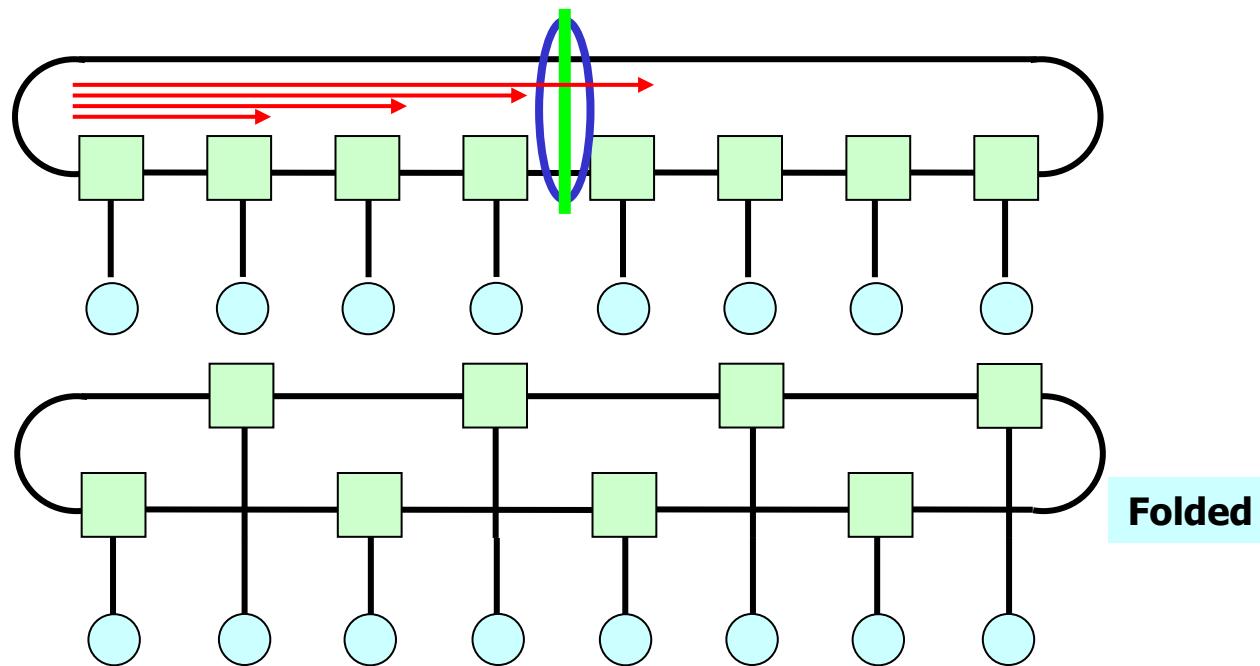
考察网络拓扑结构的重要指标

- 对分带宽
 - **对分**: 将一个包含 N 个节点的拓扑切割为两个分别包含 N_1 和 N_2 个节点的拓扑子集, $N_2 \leq N_1 \leq N_2+1$
 - **对分带宽**: 联通 N_1 和 N_2 节点子集的所有通道的带宽总和
- 网络跳步
 - **最短路径**: 源节点和目的节点之间跳步数最少的可联通路径
 - **网络直径**: 全拓扑所有节点对中**最长的最短路径**
 - **平均跳步**: 所有源节点和目的节点最短路径的平均值
- 构建成本
 - **芯片成本**: 交换/路由芯片的面积、引脚数、结构等
 - **封装成本**: 背板或交换机柜的体积、功耗、接口数量等
 - **链路成本**: 电缆或光纤光的数量、长度、布线难度等

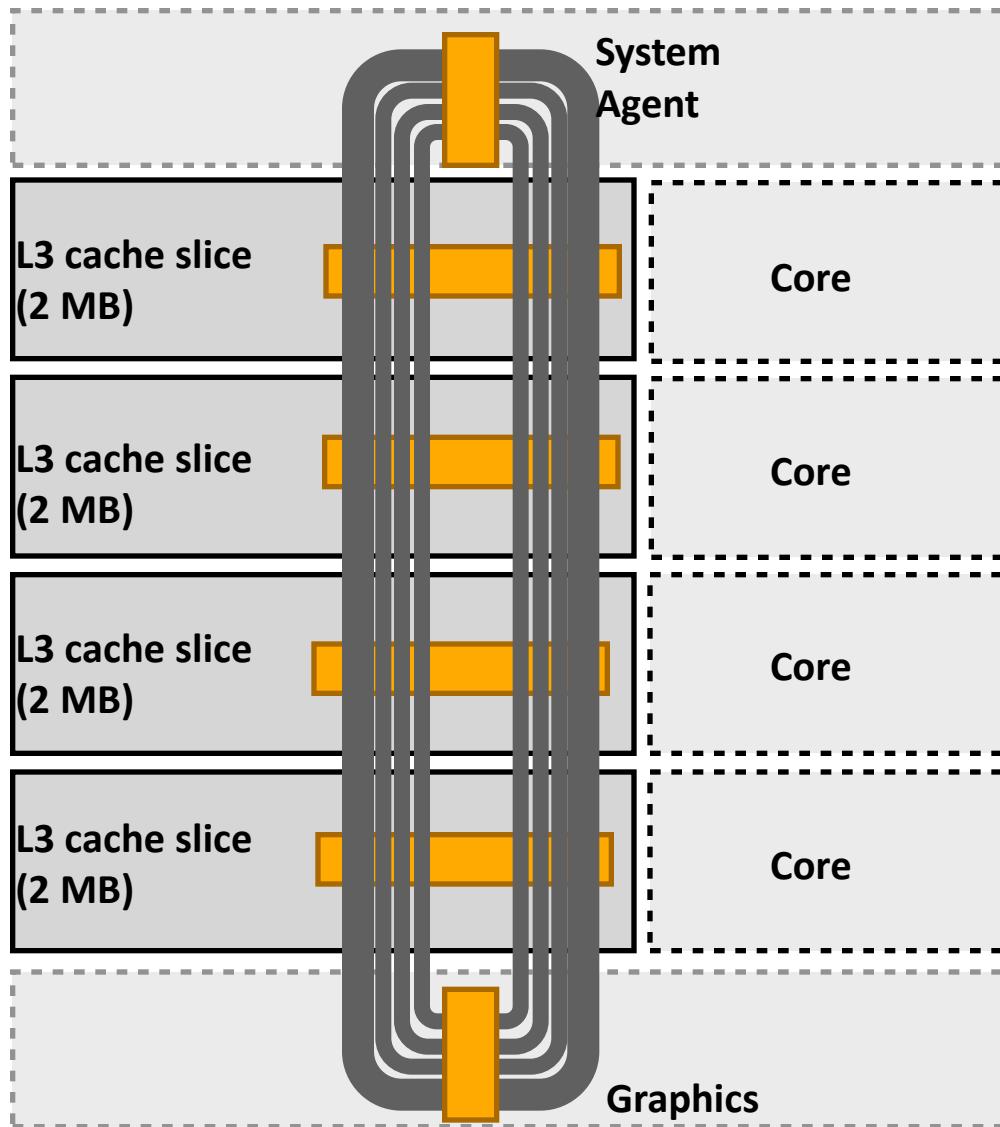
直接网络拓扑结构—Ring

- **Bidirectional Ring (节点数N)**

- 对分带宽 (B_c) : 2 links, $4B_{\text{link-undirection}}$
- 网络直径: $\lceil N/2 \rceil$
- 平均跳步 (H_{avg}) : $\lceil N/4 \rceil$
- 构建成本: 3端口Switch/Router, N条连接线, 通过折叠缩短最长连线



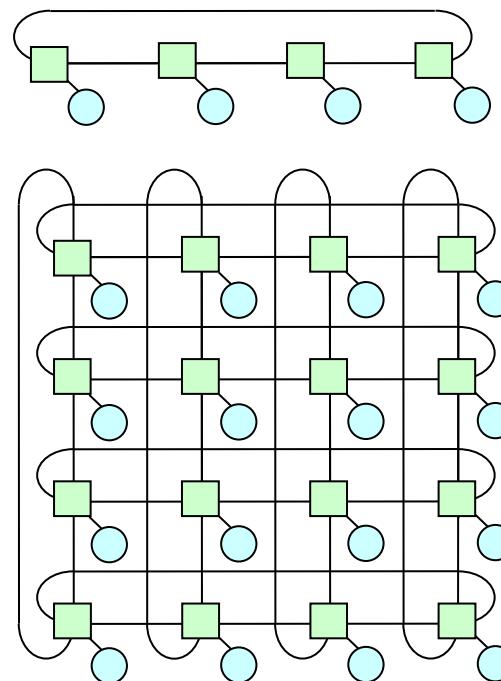
Intel处理器中的Ring互连



- **Four rings**
 - request
 - snoop
 - Ack
 - data (32 bytes)
- **Six interconnect nodes:**
four “slices” of L3 cache +
system agent + graphics
- Each bank of L3 connected
to ring bus twice
- Theoretical peak BW from
cores to L3 at 3.4 GHz is
approx. 435 GB/sec
 - When each core is
accessing its local slice

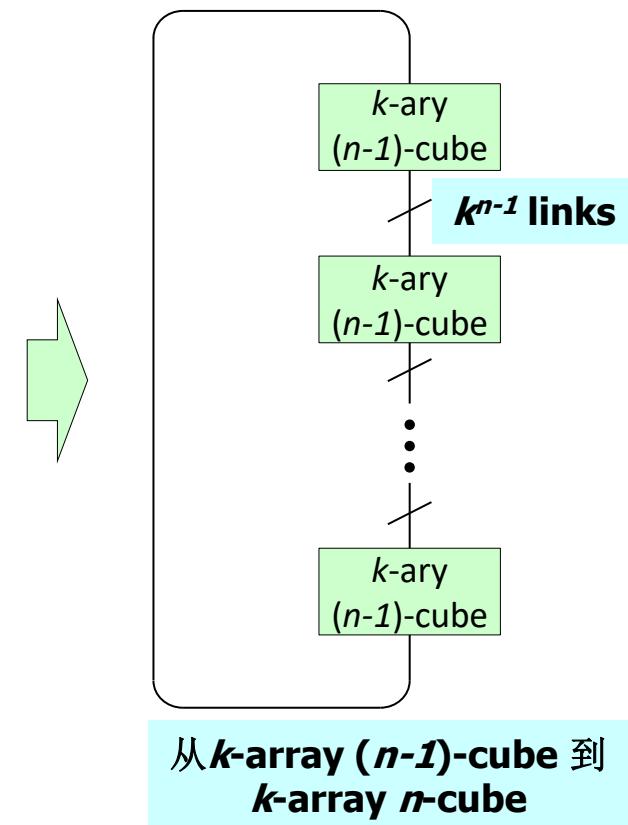
直接网络拓扑结构—Torus

- **n-dimensional, radix-k torus (k-ary n-cube)**
 - 基本思想：构建成本接近Ring，尽可能提升网络带宽延迟性能
 - 结构定义：互连规模 $N = k^n$
 - k : 每个维度上的节点个数
 - n : 维度的数量



4-array 1-cube

4-array 2-cube

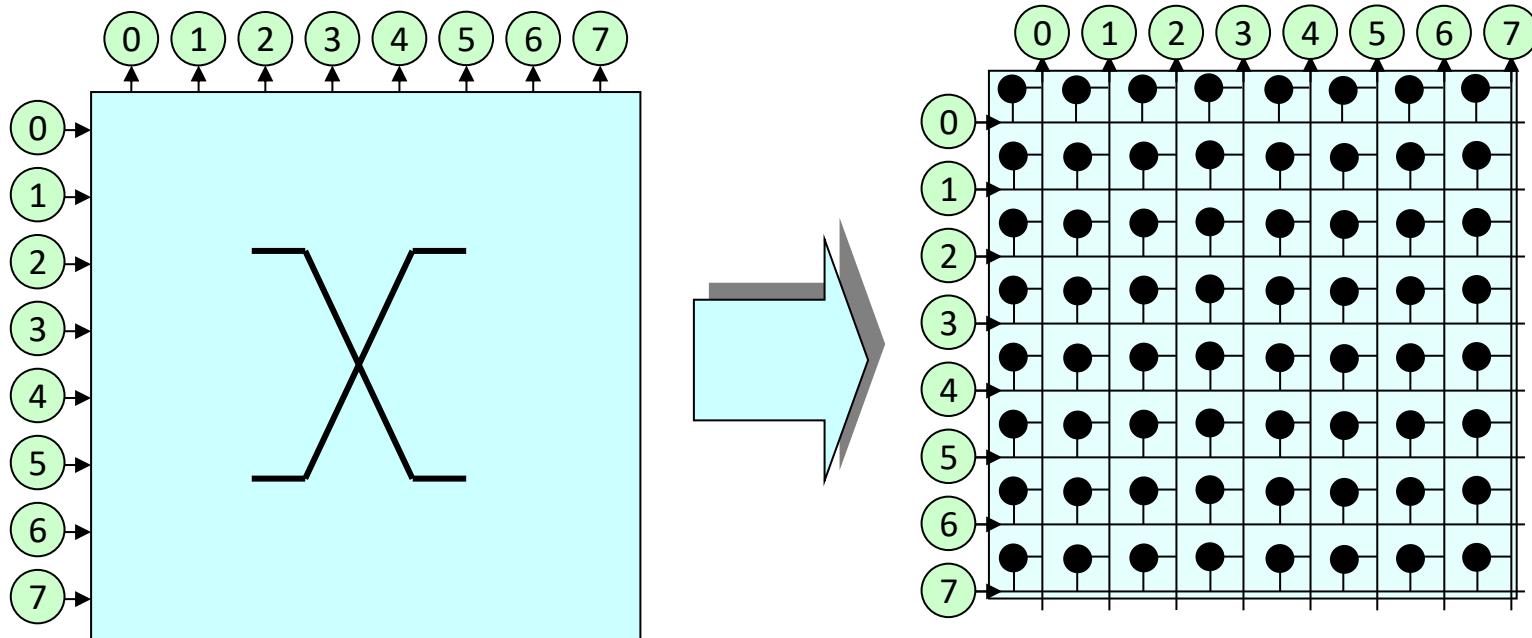


从 **k -array $(n-1)$ -cube** 到
 k -array n -cube

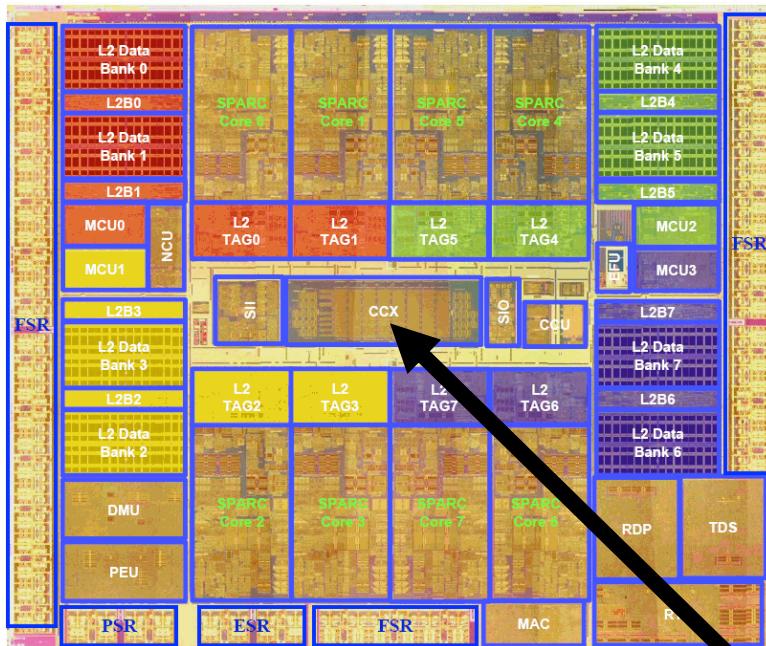
间接网络拓扑结构

- 理想的间接网络

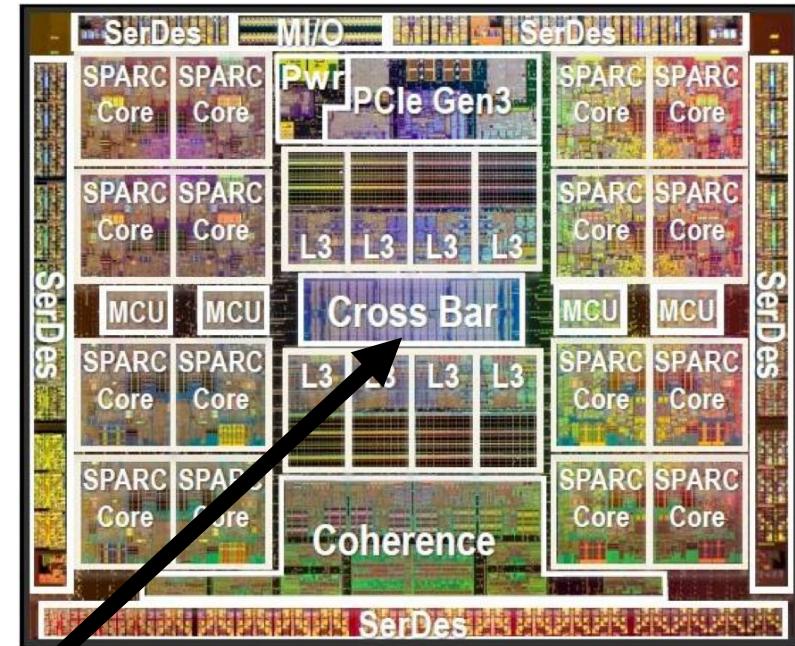
- **Crossbar**: 每个输入节点和输出节点对之间通过一个开关连接
- 复杂度: $O(N^2)$
- 无阻塞网络: 去往不同输出节点的数据包不会相互阻塞



多核处理器中的Crossbar



Sun SPARC T2 (8 cores, 8 L2 cache banks)

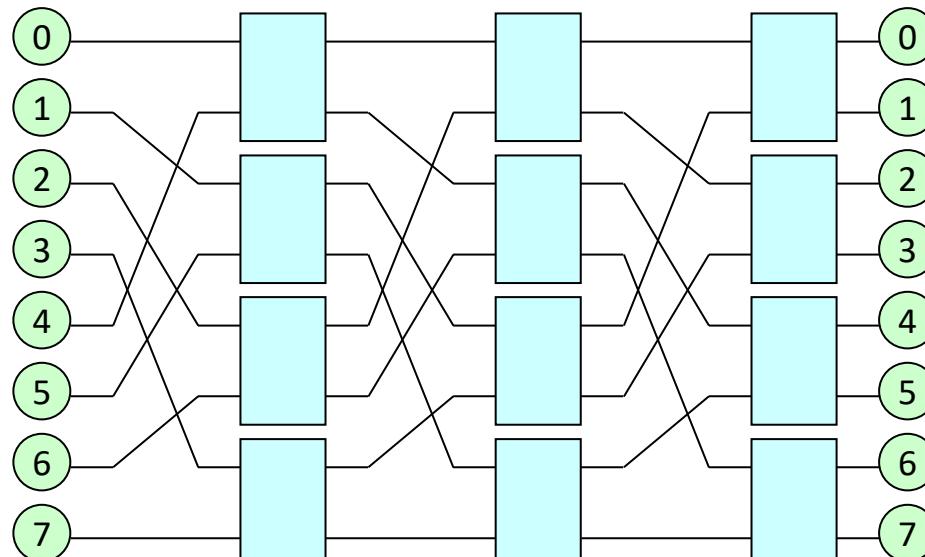


Oracle SPARC T5 (16 cores, 8 L3 cache banks)

Note that crossbar (CCX) occupies about the same chip area as a core

多级间接网络拓扑结构

- 通过多级交叉开关（Switch/Router）搭建的间接网络
 - 全局大交叉开关拆分成多级小交叉开关
 - 复杂度降低： $O(N \times \log N)$
 - 各级交叉开关之间通过规则的排列函数实现连接
 - 可能产生阻塞



Omega topology, perfect-shuffle exchange

- 标号相同的源和目的视为一个节点
- 拓扑中的连线均为从左向右的单向链路

多级间接网络——Butterfly

- 使用同构交叉开关的多级网络

- 结构定义：

- N : 互连节点总数
 - k : 每级crossbar或switch/router的**单向连接度**（入端口/出端口数）
 - n : 全网交叉开关的**层数**, 每层包含 k^{n-1} 个交叉开关
 - 要保证基本联通需求: $k^n = N \Rightarrow n = \log_k N$

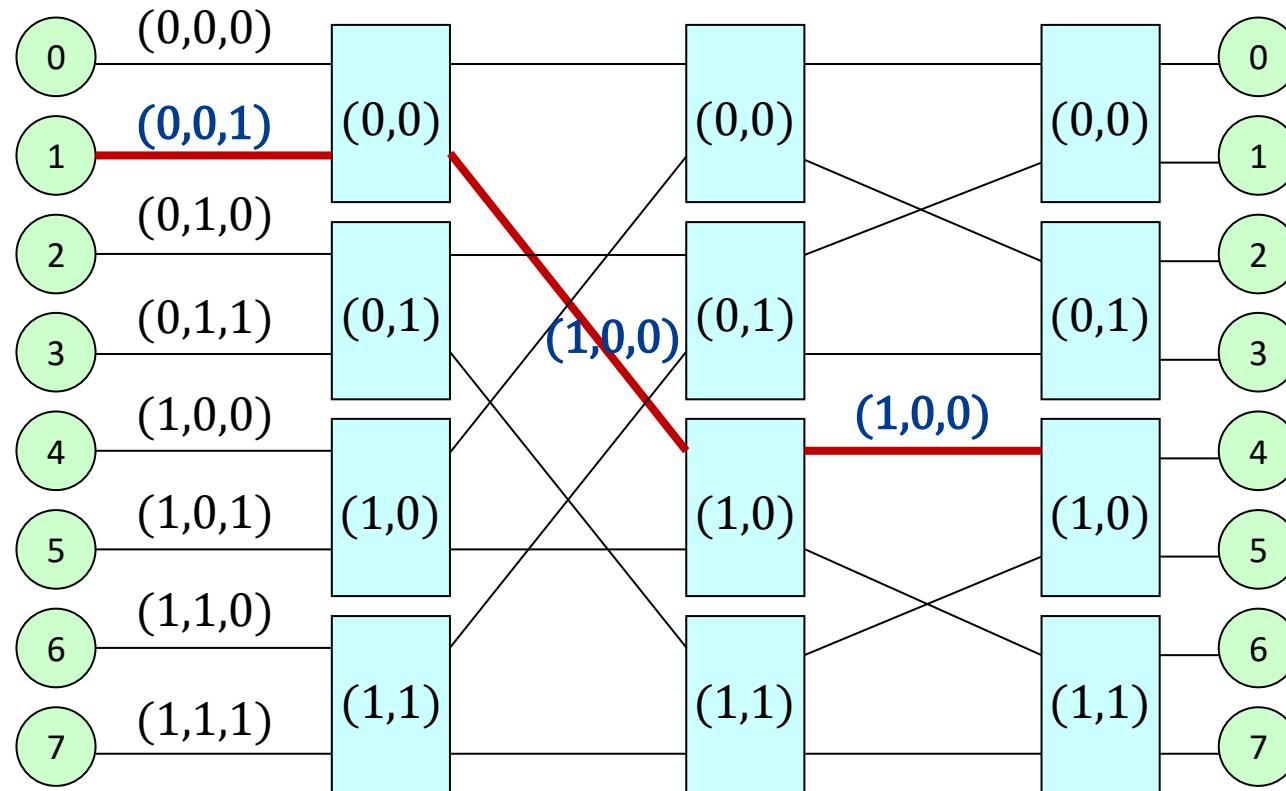
- Butterfly or k -ary n -fly

- 排列函数：

- 从入节点与第一级交叉开关的连线开始, 每一级的每条连线都可以用 $(d_{n-1}, d_{n-2}, \dots, d_0)$ 向量进行编号, 其中 d 为 k 进制数
 - 其中, $(d_{n-1}, d_{n-2}, \dots, d_1)$ 表示交叉开关的编号, d_0 表示交叉开关的第几号连线, 合起来就是该连线在该级的全局编号
 - 第一级编号确定后, 从第 $i - 1$ 级到第 i 级, 交换 d_{n-i} 和 d_0 获得新的连线编号

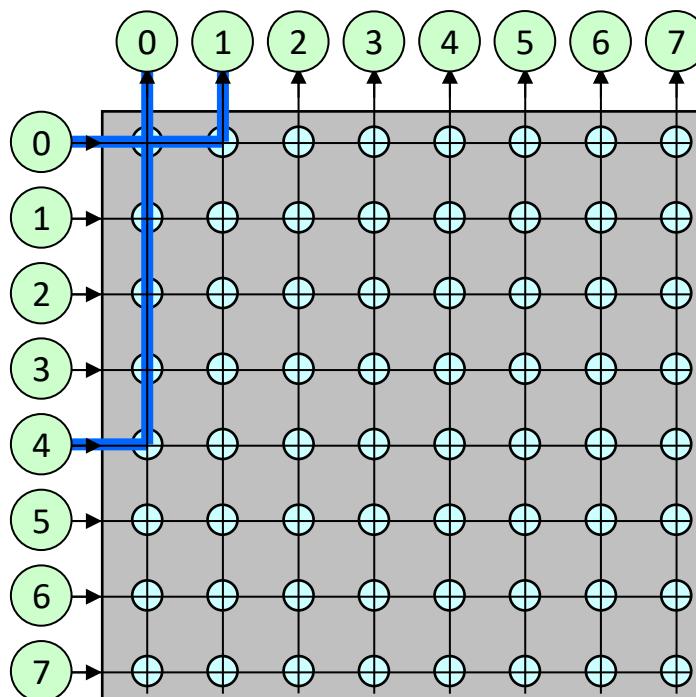
Butterfly连接排列函数举例

观察第一级第二根线的编号变化轨迹

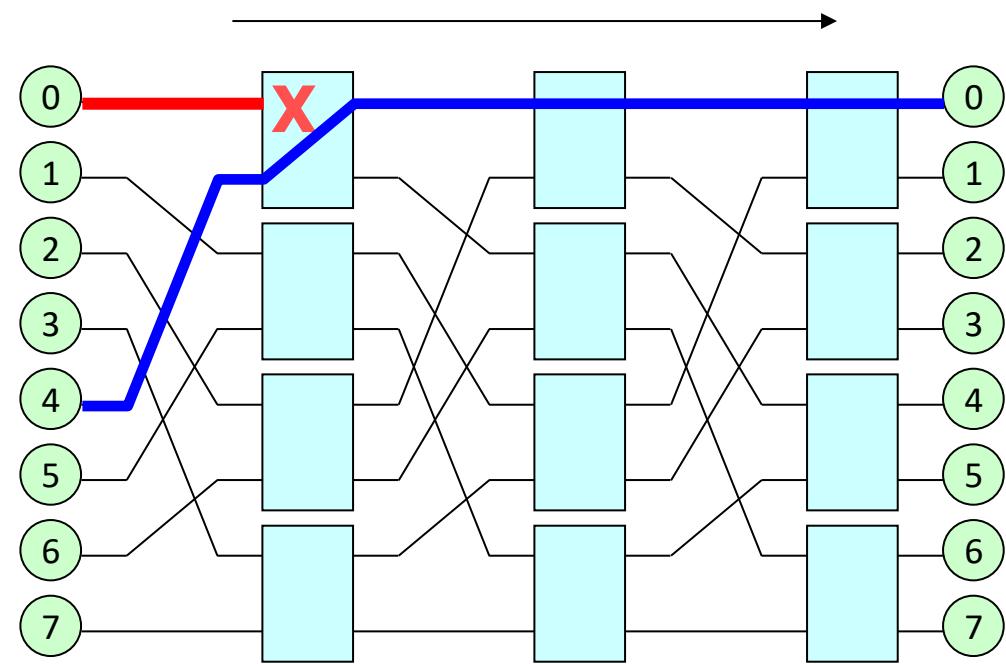


多级间接网络—阻塞问题

- 网络阻塞的原因：资源竞争，解决方法：增加资源（路径）
 - 使用更“大”的交叉开关
 - 使用更“多”的交叉开关



non-blocking topology



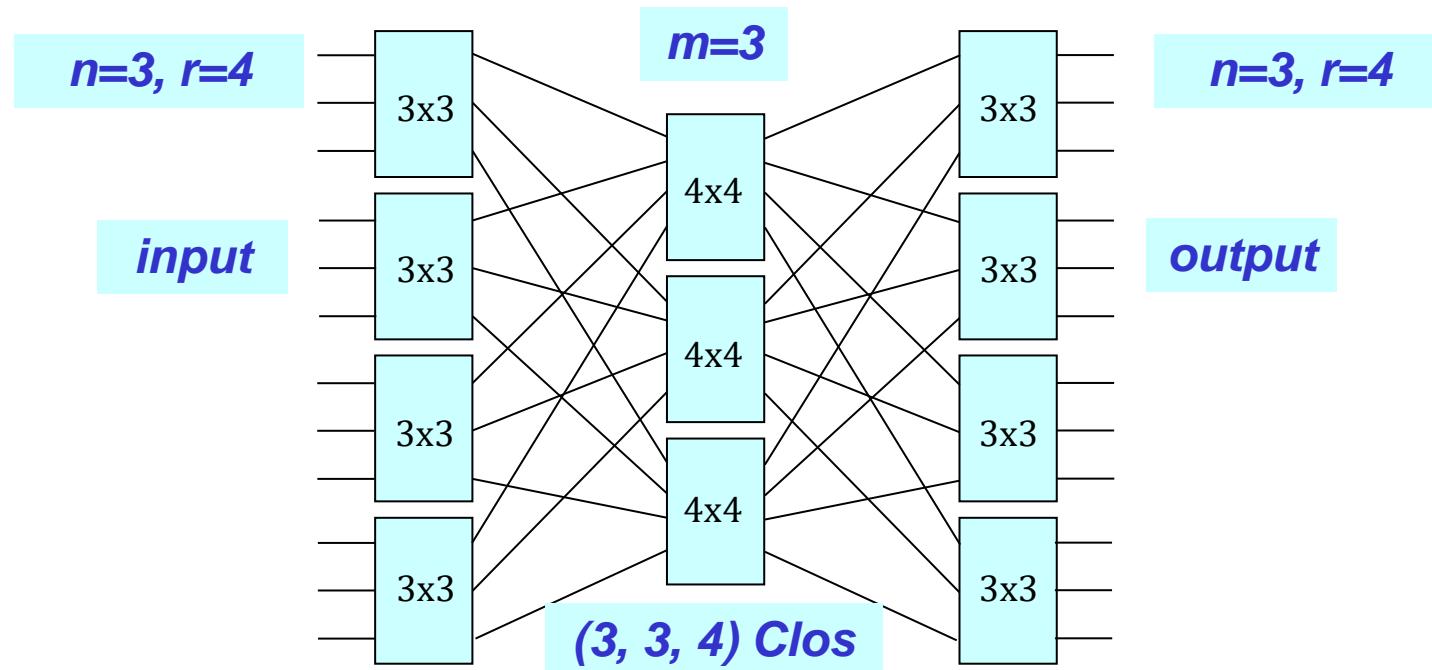
blocking topology

Clos网络

- (m, n, r) Clos网络

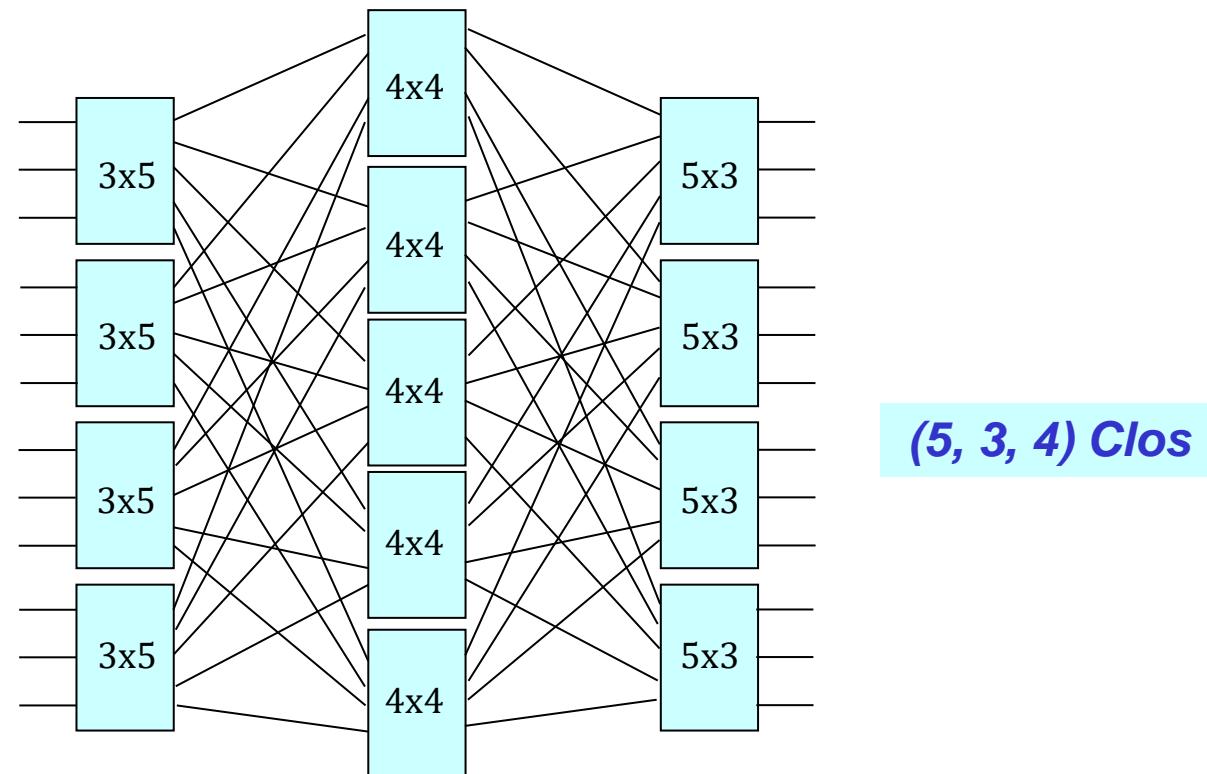
- 结构定义：

- Clos网络包含三层交叉开关/Switch
- m : 中间层Switch的个数
- n : 输入层和输出层Switch的输入端口数
- r : 输入层和输出层的Switch个数



严格无阻塞Clos网络

- **严格无阻塞**: 不管网络处于何种状态, 任何时刻都可以在网络中建立一个连接, 只要这个连接的起点和终点是空闲的, 而不会影响网络中已经建立起来的连接
- Clos网络严格无阻塞所需满足的必要条件: $m \geq 2n - 1$

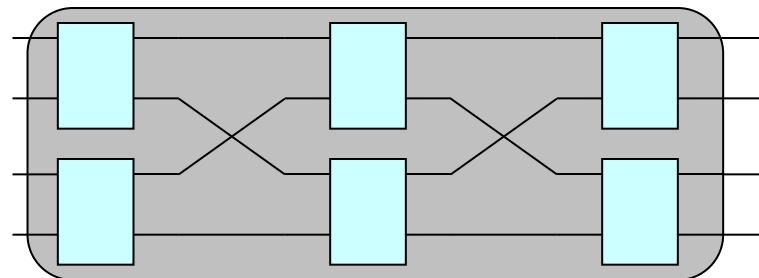


可重排无阻塞Clos网络

- **可重排无阻塞**: 不管网络处于何种状态，任何时刻都可以在网络中直接或对已有的连接重选路由来建立一个连接，只要这个连接的起点和终点是空闲的，而不会影响网络中已经建立起来的连接
- Clos网络可重排无阻塞所需满足的必要条件: $m \geq n$
 - 利用 $n \times n$ 交叉开关/Switch搭建整个无阻塞网络（有利于工程实施）

从Clos网络到Fat-Tree网络

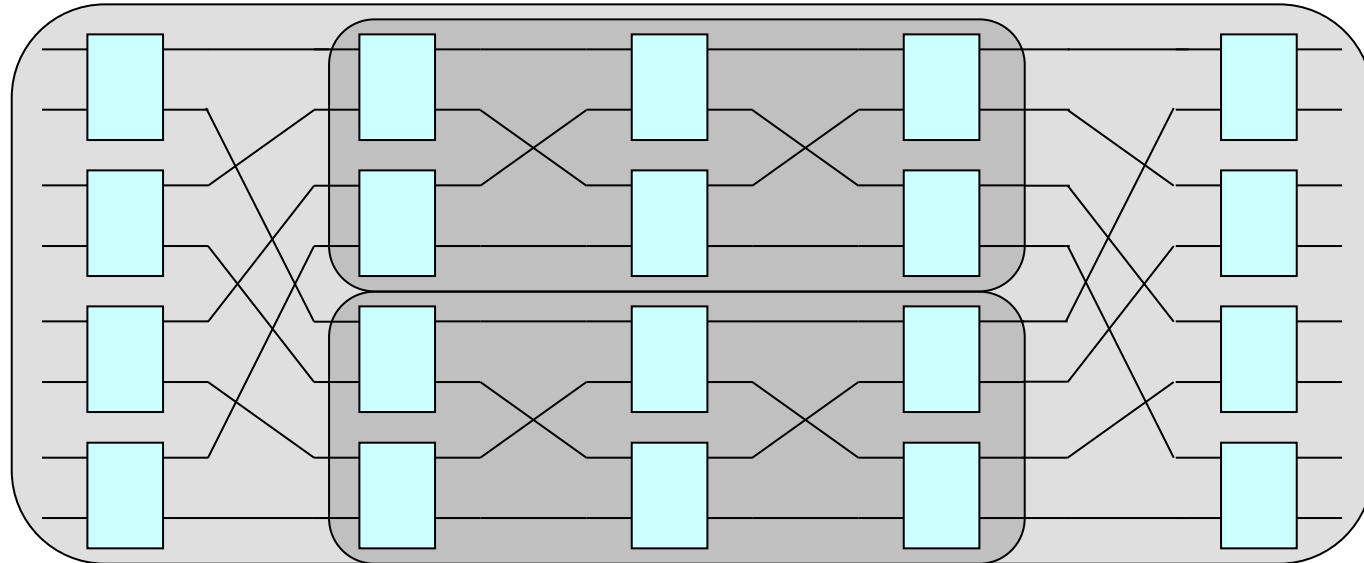
- 大于3个Stage的Clos网络
 - 从3层(n, n, n)Clos网络出发



将3层 (n, n, n) clos网络看作一个整体，作为middle switch，
构建 (n, n, n^2) clos网络



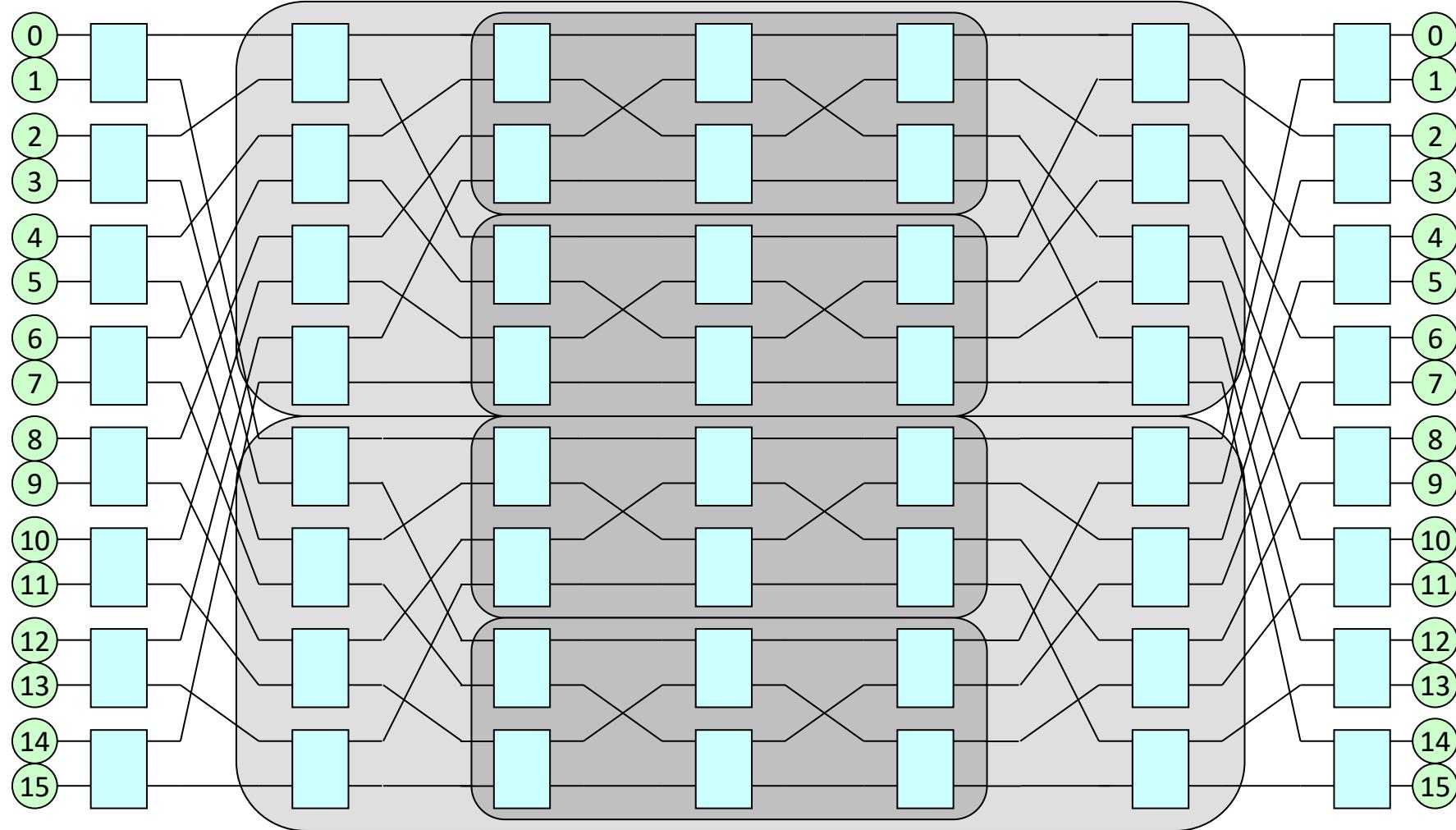
(n, n, n^2) 5层clos网络



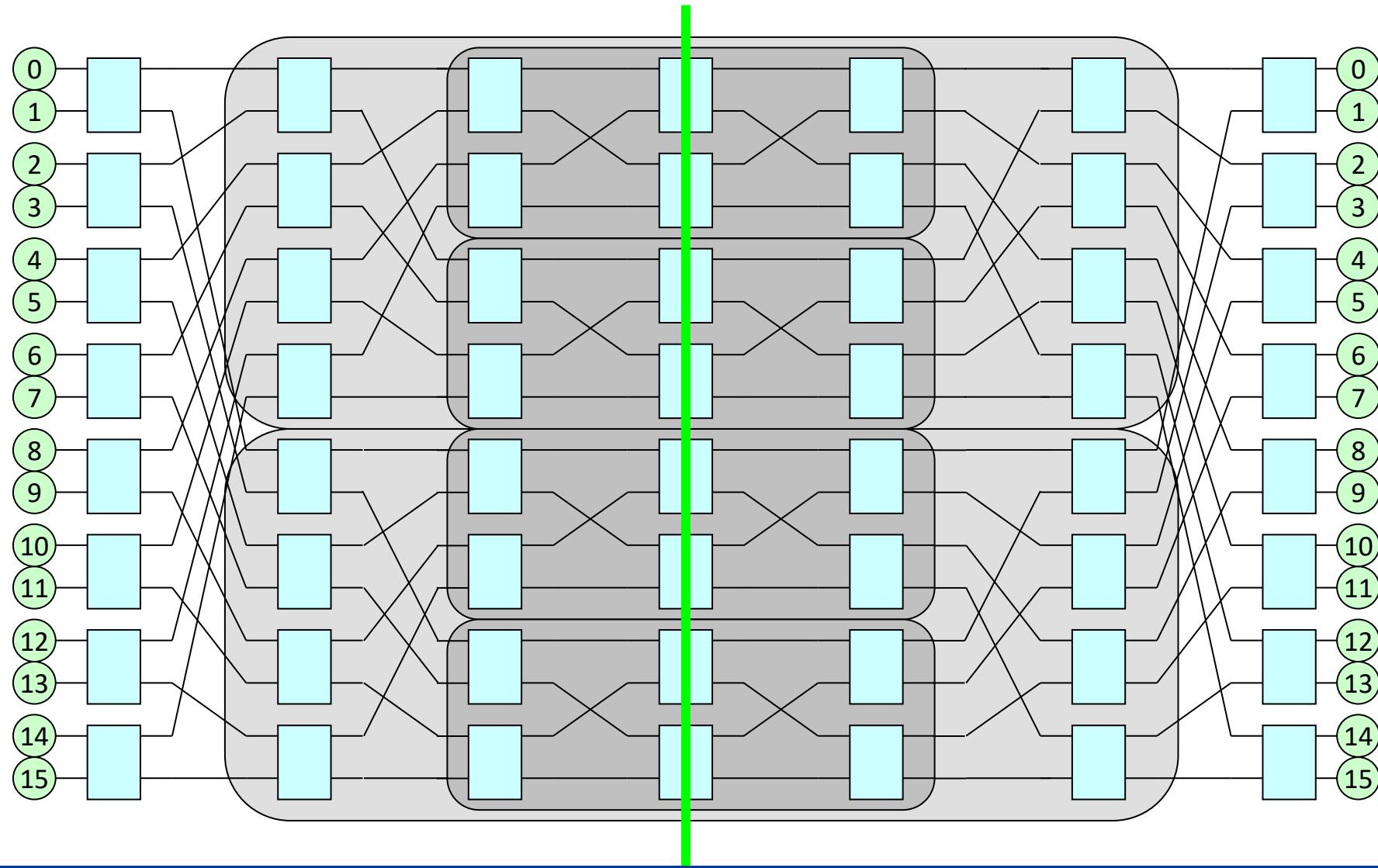
递归操作



(n, n, n^3) 7层 clos 网络

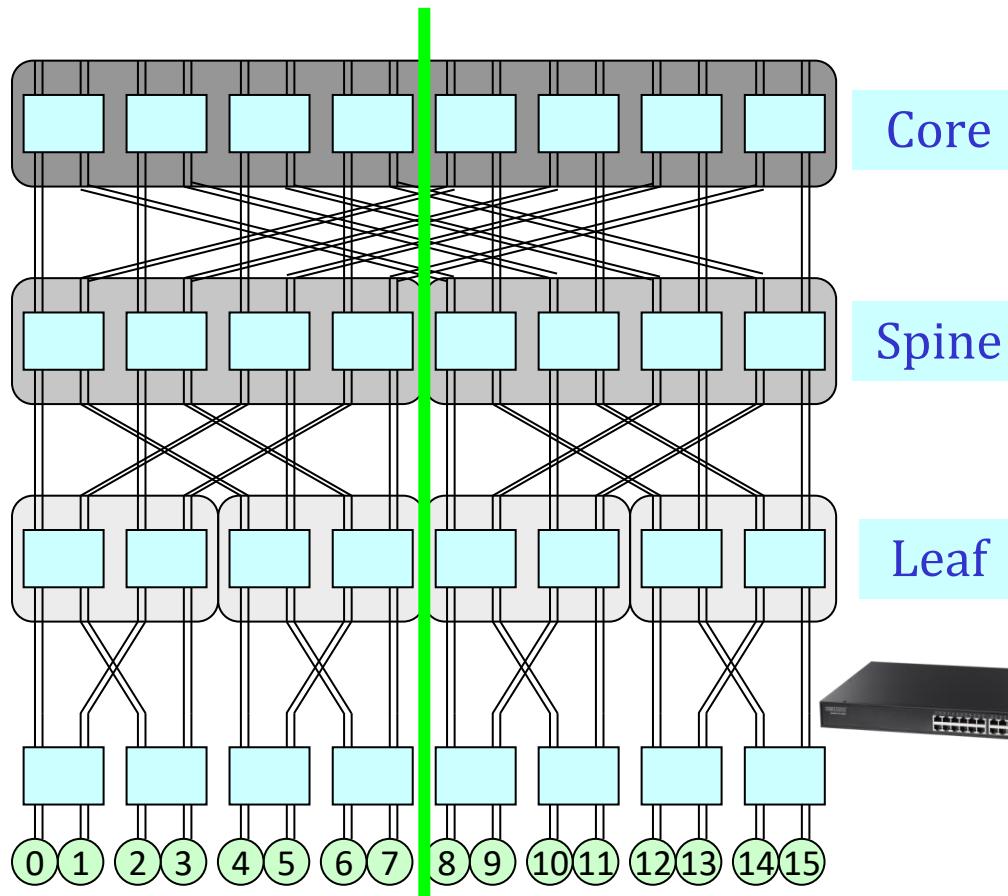


$(n, n, n^i)(2i + 1)$ 层clos网络折叠⇒Fat-Tree



常见的Fat-Tree网络

- m-port, n-tree Clos网络，m是交叉开关/交换芯片端口数，n为树的层数



优势:

- 可重排无阻塞
- 低网络直径
- 多路径数据交换实现容错
- 全网使用完全同构的交换芯片降低成本
- 系统规模可随单个交换芯片端口数的增长而扩展
- Infiniband网络最常用拓扑



网络拓扑比较

- 互连规模为N个节点

	类别	Bus	Ring	2D Torus	3D Torus	Butterfly	(m,n) Fat-Tree
性能	对分带宽 (in links)	1	2	$4N/\sqrt{N}$	$4N/\sqrt[3]{N}$	$N/2$	N
	网络直径	1	$N/2$	\sqrt{N}	$3\sqrt[3]{N}/2$	$\log_k N + 1$	$2n$
	平均跳步	1	$[N/4]$	$\sqrt{N}/2$	$3\sqrt[3]{N}/4$	$\log_k N + 1$	$[Nn + mn(n - 1) - 2]/N$
成本	Switch端口数	0	3	4	6	$2k$	m
	Switch个数	0	N	N	N	nk^{n-1}	nN/m
	全网连线数	1	N	$2N$	$3N$	$(n - 1)N$	$(n - 1)N$

路由算法

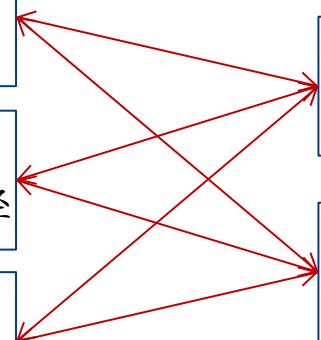
路由算法

- **目的：**在交换机中通过查表或路径计算，从全网中找出特定的路径，可将数据包从源节点送达目的节点
- **约束条件：**负载均衡、最短路径、容错、无死锁和活锁等
- **路由算法的分类：**

根据路径选择的方式分类

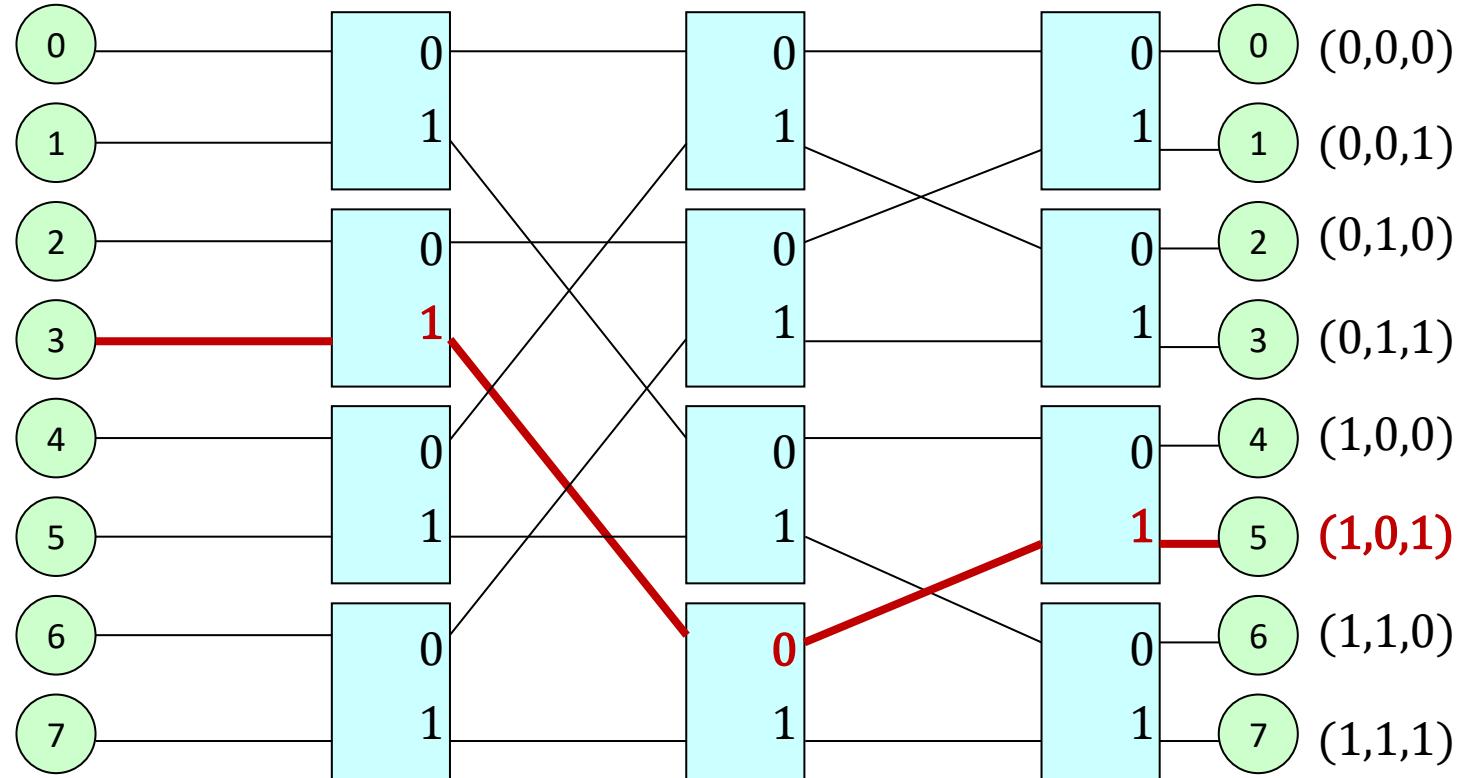


根据路由信息获取的方式分类



确定路由举例——Butterfly

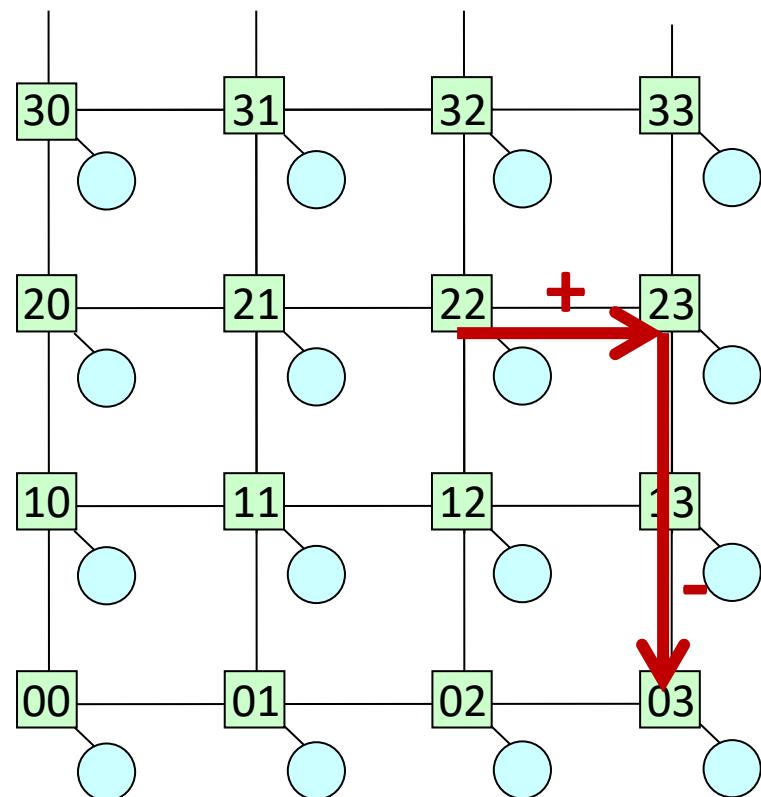
- 根据目的节点编号计算的确定路由
 - $output_port = dest_tag[msb]$
 - $dest_tag \ll 1$



确定路由举例——Cube (Cray T3D)

- 按照维度顺序 (dimension-order) 的确定路由

以 (2,2) 到 (0,3) 为例



第一步：方向计算：

$$m_i = d_i - s_i \bmod k$$

$$m = (0,3) - (2,2) \bmod 6 = (4,1)$$

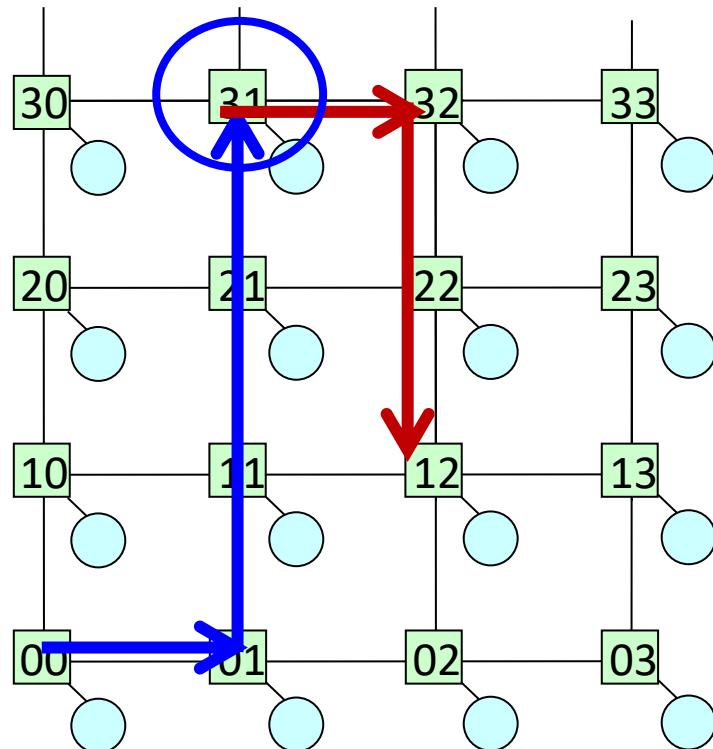
$$\Delta_i = m_i - \begin{cases} 0 & \text{if } m_i \leq k/2 \\ k & \text{otherwise} \end{cases}$$

$$\Delta = (4,1) - (6,0) = (-2,1)$$

第二步：按照先走X维，再走Y维路由

遗忘路由举例——Valiant随机路由

- 普适性好：任意流量Pattern，任意网络拓扑
- 核心思路：将 s to d 的路由，拆分成 s to x 和 x to d 两段，其中 x （中间目的地）随机选择，两段分别的路由算法也可自由选择



以 (0,0) 到 (1,2) 路由为例

- 随机选择中间节点(3,1)
- (0,0)到(3,1)维序确定路由
- (3,1)到(1,2)维序确定路由

⌚ Valiant将任一流量都拆分成两个同等数据量的Random流量，等价于将网络容量 (Capacity) 缩减了1/2

⌚ 消除了某些流量的拥塞点

自适应路由

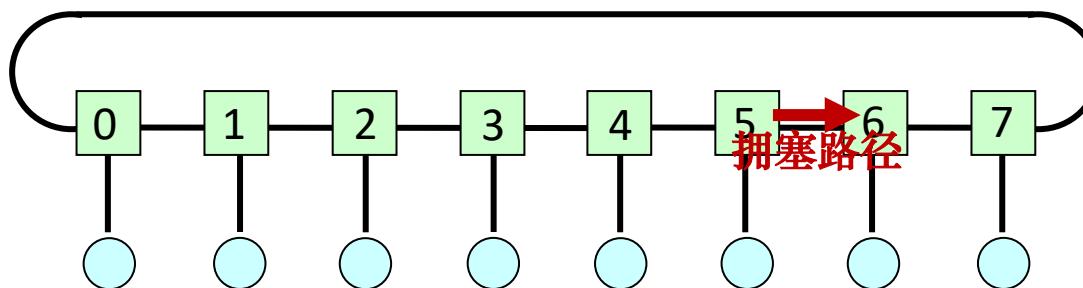
- 利用**网络状态信息**（通常是交换队列的占用情况）来选择合适的路径传递数据包
 - Local or Global Information?*
 - Current or Historical Information?*

以节点3到节点7路由为例

case 1: 5和6的拥塞未传递到3和4 (Local信息)，选择右向路径

case 2: 3感知到5和6拥塞 (Historical信息)，选择左向路径

case 3: queue的深度影响自适应路由的灵敏度



- 自适应路由算法要结合具体机器的网络拓扑、流控、网络管理等机制进行case by case地设计

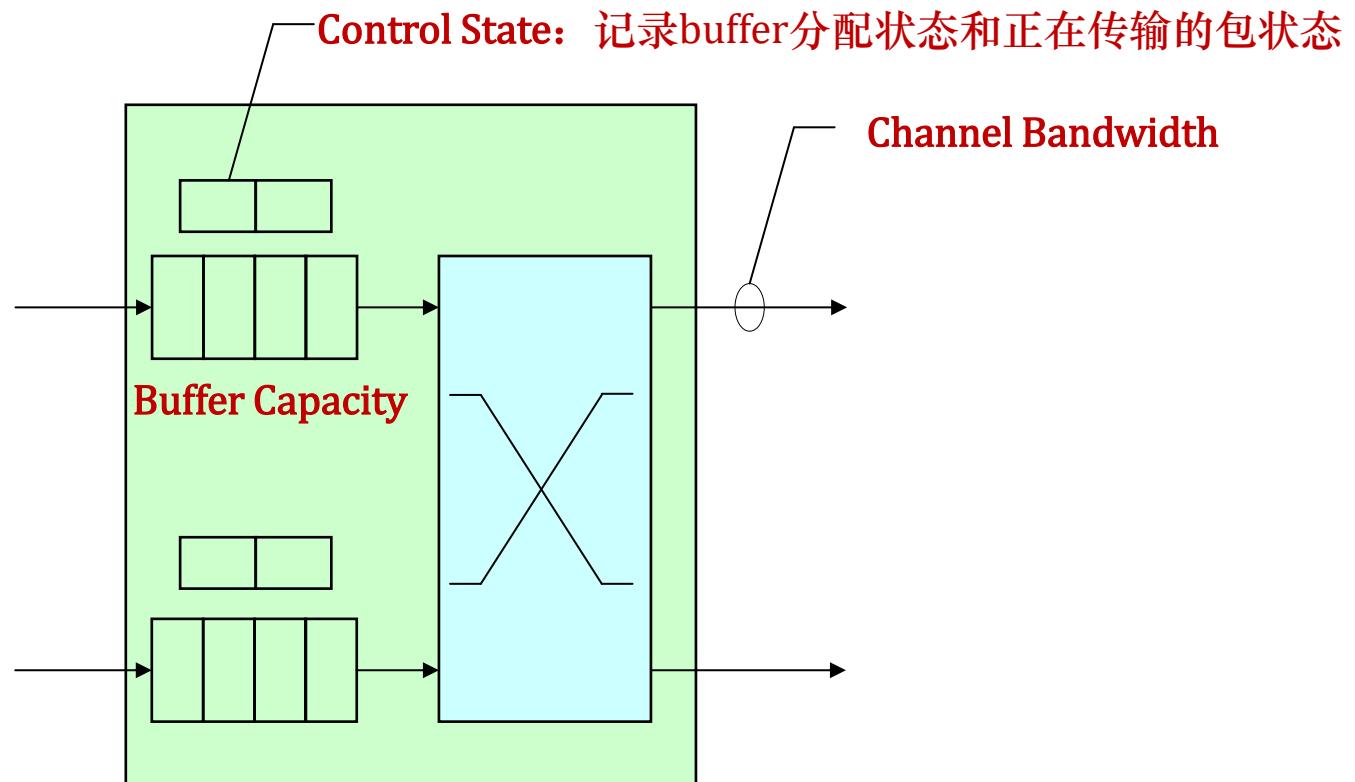
三种路由的比较

	确定路由	遗忘路由	自适应路由
复杂度	最简单	较为简单 (无需考虑网络状态)	最为复杂
负载均衡	较差	较好	与反馈信息的使用紧密相关
容错能力	差	一般	较好
特殊性	优秀的保序能力 (满足协议特殊要求, 如 cache一致性)	Valiant清除流量模式	Case by case的设计

交换和流控

总述

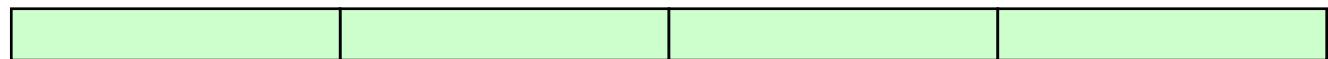
- 交换和流控解决网络资源（主要是带宽、缓存及控制状态）在不同网络包之间的分配问题



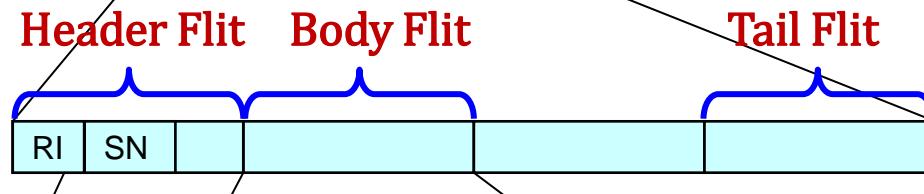
交换和流控涉及的网络单元的资源举例

按流控粒度分类

消息 (Message)

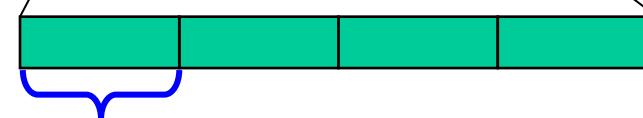


包 (Packet)



路由信息

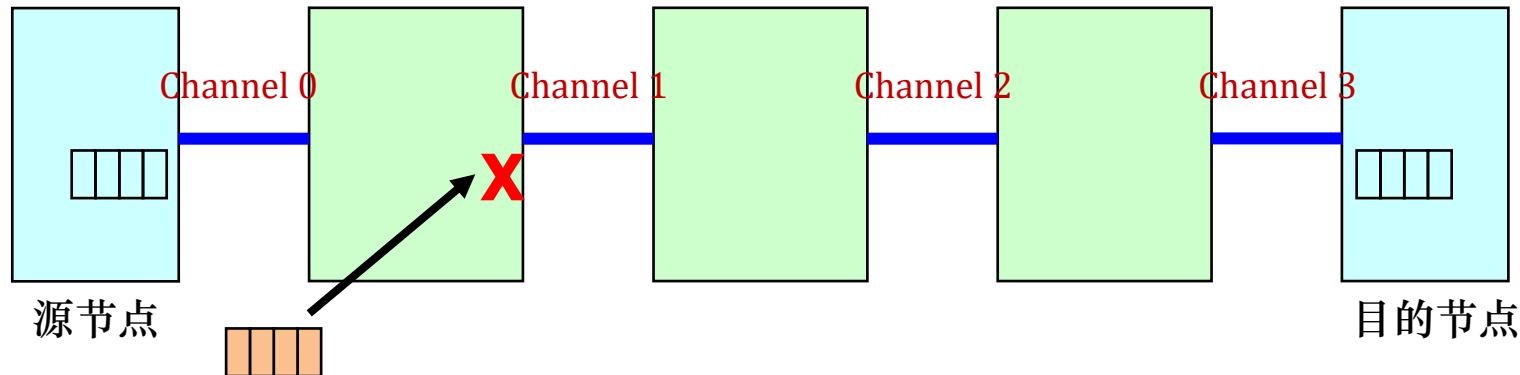
信元 (Flit)



Phit: 物理链路传输单元, 如: 100G以太网是4bit,
GPU PCIe接口是16bit

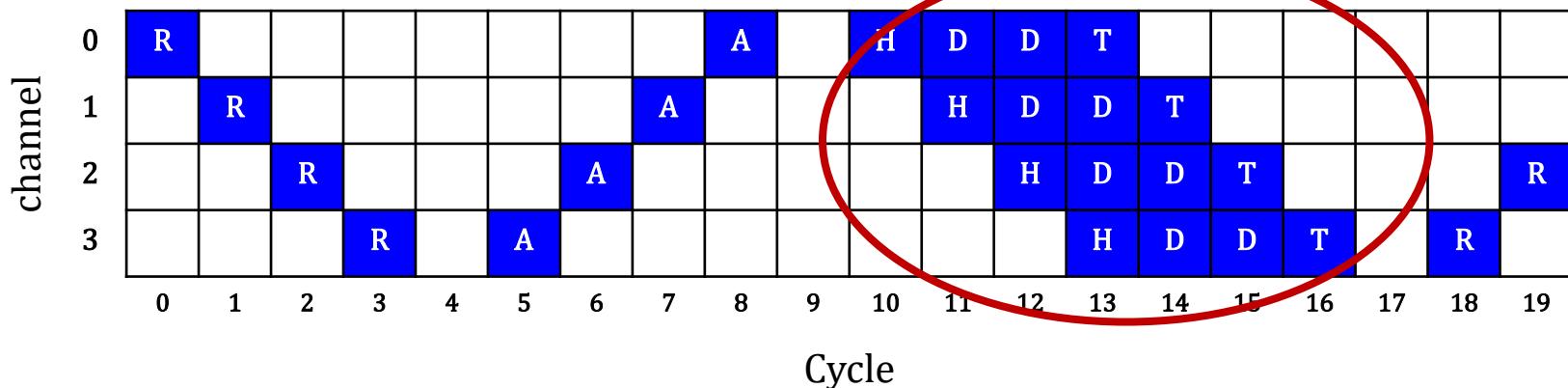
电路交换

- 为每个消息建立从源到目的的专线，可以做到 Bufferless (优)，但是资源利用率低 (劣)



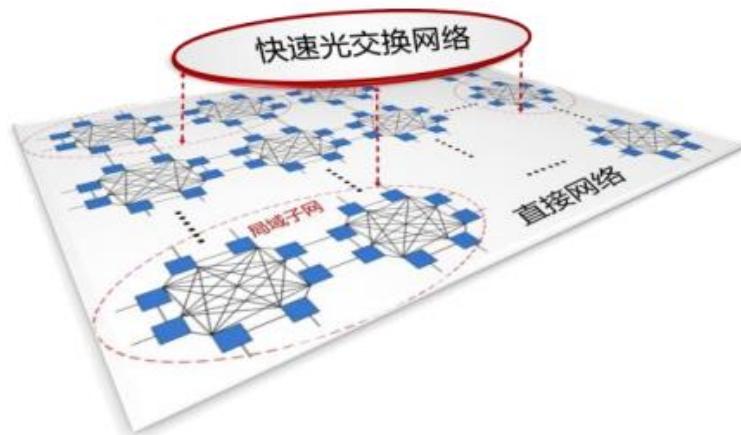
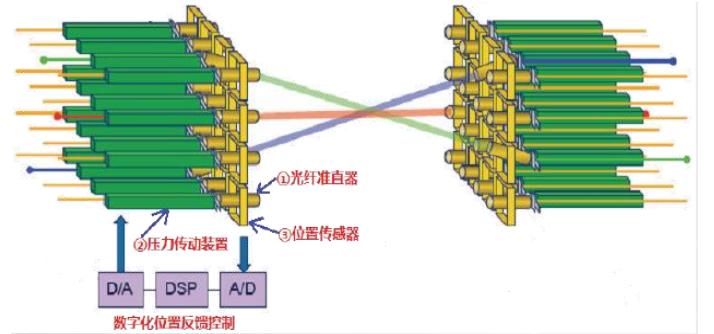
R=Request A=Acknowledgement H=Header D=Data T=Tail

适合大块数据传输



基于新型光器件的电路交换Case

- 光器件的特点：高带宽、低延迟、低功耗但难以缓存



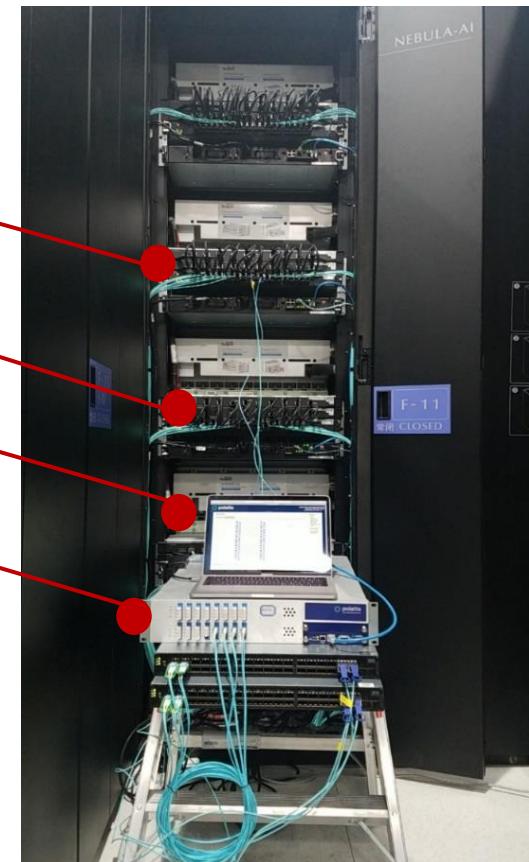
曙光7000光电混合网络

机顶交换机
ToR

计算节点

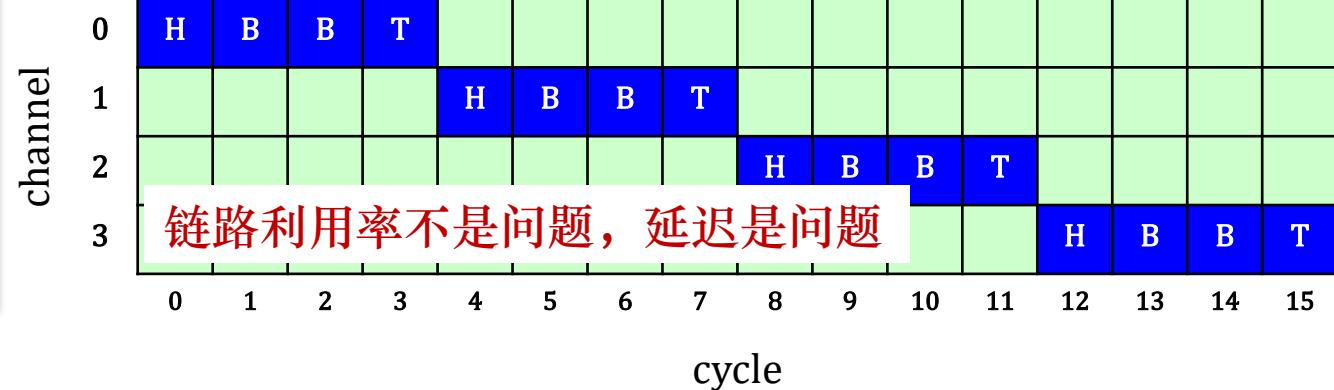
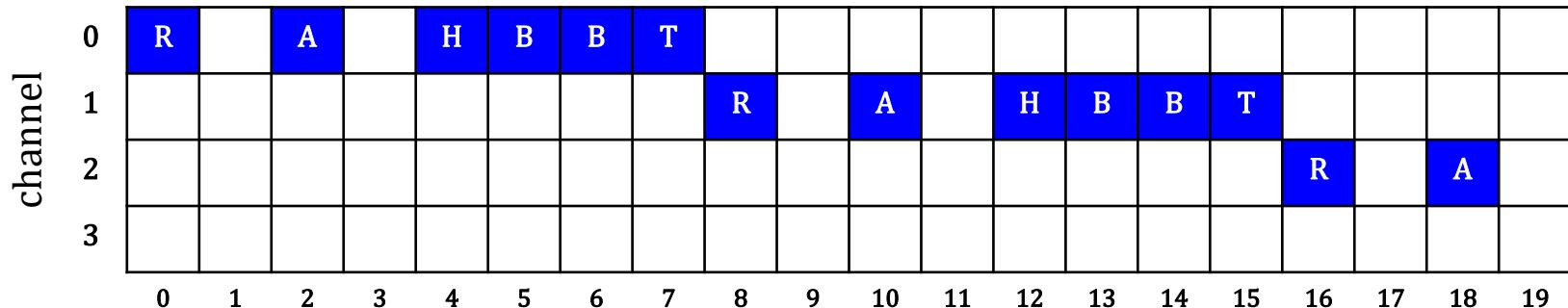
SDI控制器

Polatis光交换机



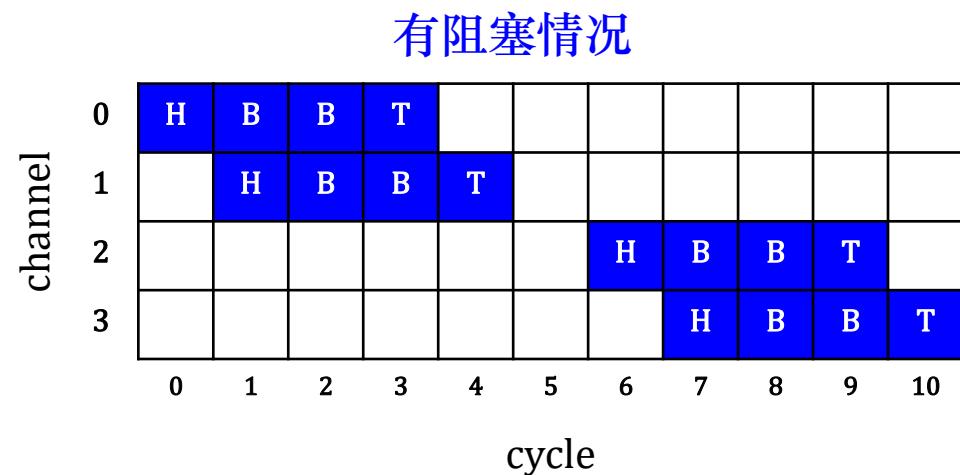
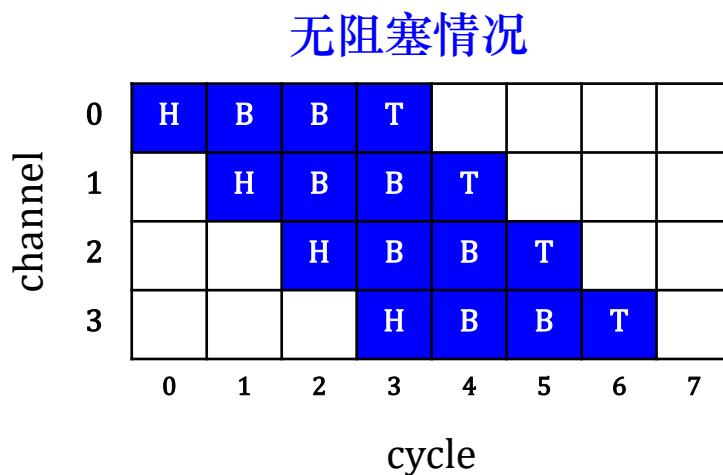
包缓存交换——Store-forward

- 包交换：路径上的网络单元相互独立，包只能占用一级单元
- SF：每级网络单元要把包完整接收下来，才请求下一级



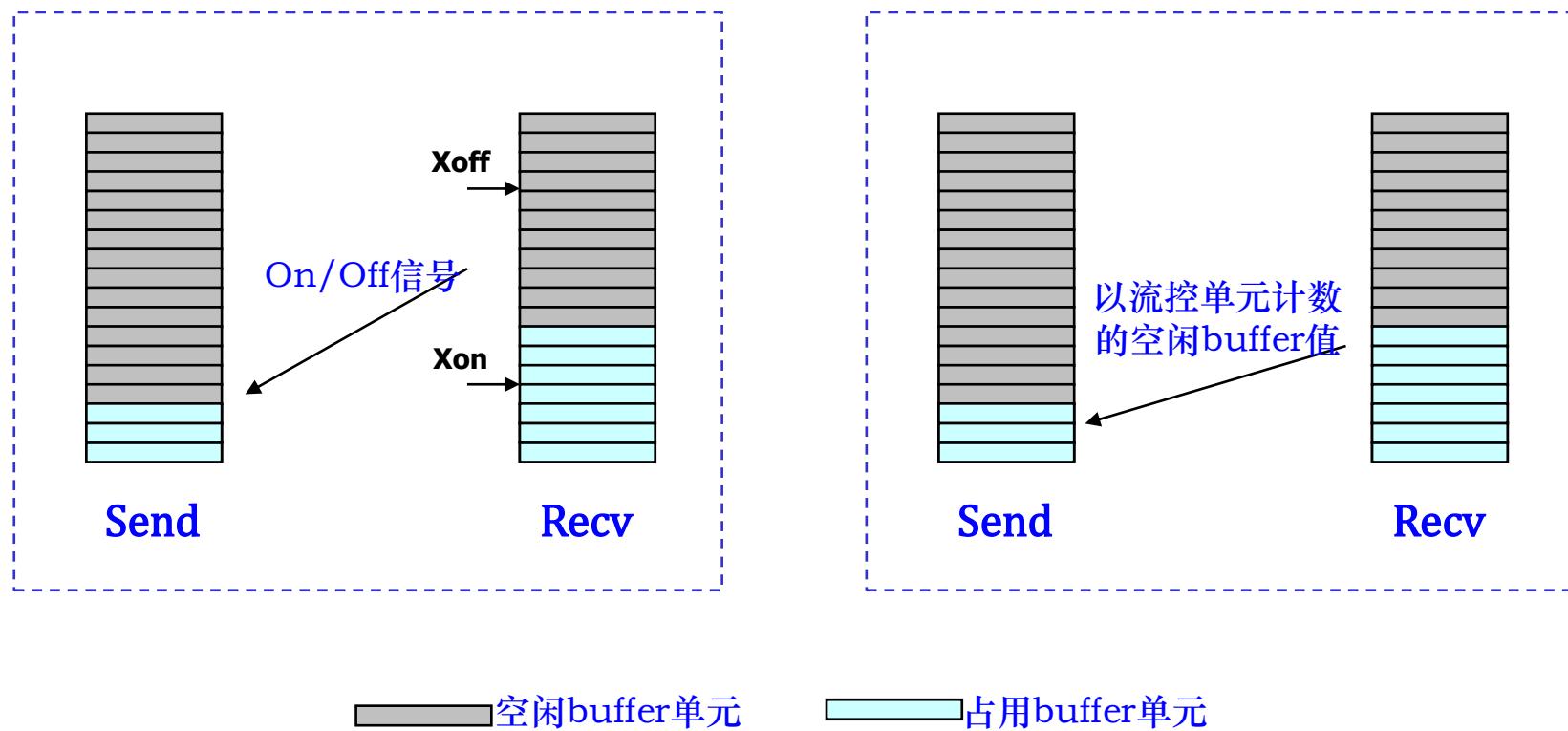
包缓存交换——Cut-through

- 每级网络单元都只需要把包头接收下来，就可请求下一级
- 无阻塞情况表现出非常好的延迟性能
- 有阻塞情况退化为Store-forward (**Buffer消耗量高**)



流控——Buffer管理

- 粗粒度，简单，带宽占用小：Xon/Xoff流控
- 细粒度，复杂，带宽占用高：基于信用的流控（Credit-based）
- 两者都需要考虑数据飞行时间



流控和路由结合的子问题

——死锁

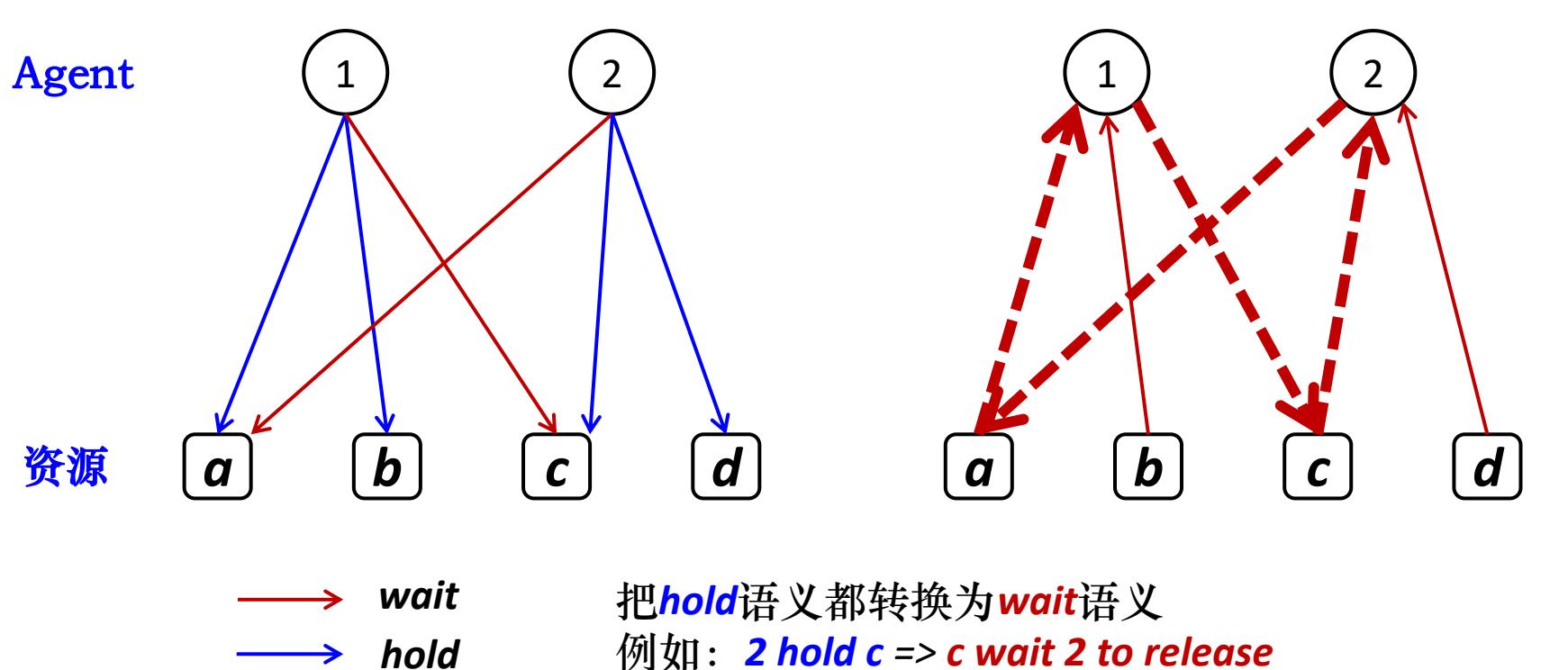
死锁

- **死锁**: 是指两个或两个以上的Agent，因争夺资源（链路、缓存等）而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去

Agent: 代理人，代表应用去争夺资源，因交换和流控类型不同，代理也不相同

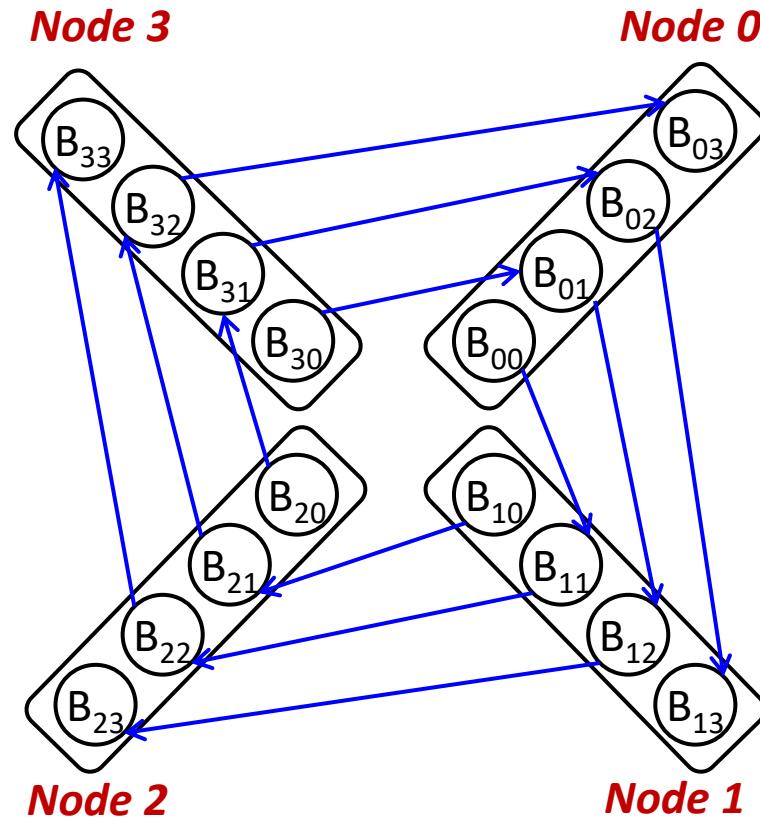
交换和流控模式	Agent	Resource
电路交换	连接	一组链路
包缓存	包	Buffer

死锁原因——资源依赖关系成环



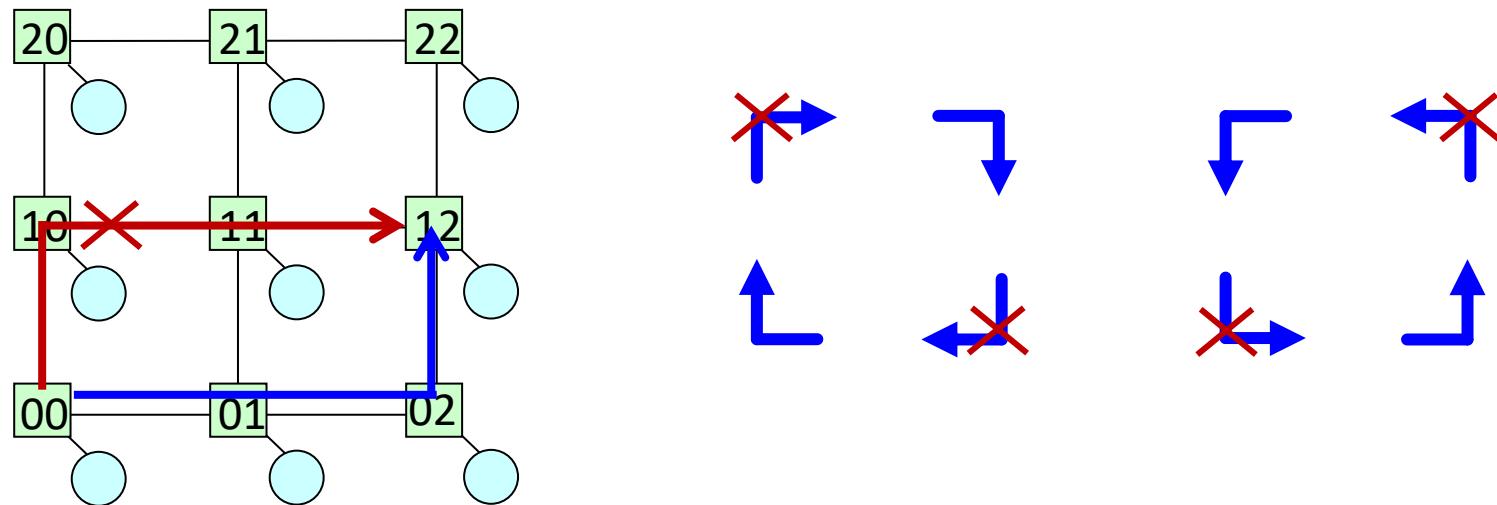
避免死锁的方法1：资源按序分类

- 将传输路径上的资源按序分类，每跳所请求资源的序不断上升，例如：*Packet*在距离源节点 i 跳时，就只能请求序为 i 的*buffer*



避免死锁的方法2：严格规定路由

- 资源分类方法有可能引入Buffer消耗过多或Buffer利用率低等问题
- 可以通过设定特殊的路由规则来破除资源依赖环
 - Case: k-ary n-mesh拓扑上的维序路由 (Dimension Order Routing)
 - 每个数据包每次只在一个维度上路由，当在该维度上达到目的坐标之后，才按低维到高维的顺序在其他维度上路由
 - 由于严格按照维度升序单调请求下一级资源，所以不会死锁



避免死锁的方法3：逃逸虚通道

- 资源分类方法有可能引入Buffer消耗过多或利用率低等问题
- 严格规定路由顺序又无法躲避拥塞



- 自适应路由+逃逸虚通道
- 感兴趣可以深入研究 (*Duato's Protocol*)

系统互连网络案例分析

史前—基于晶体管电路的向量机

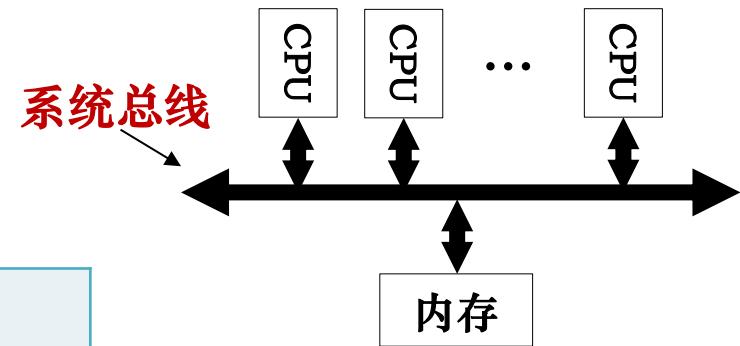
- 体系结构：向量机
- 系统互连：部件间互连

CPU耦合度	直接与CPU内部各部件耦合
功能	为CPU内的向量部件供数 (原始)
物理介质	板级部件间金属走线



单机能力型

- 体系结构：共享内存架构
- 系统互连：系统总线



CPU耦合度	最紧密，处理器内部总线的延伸
功能	CPU内存读写功能的基本扩展
物理介质	板级芯片间金属走线

代表机型：

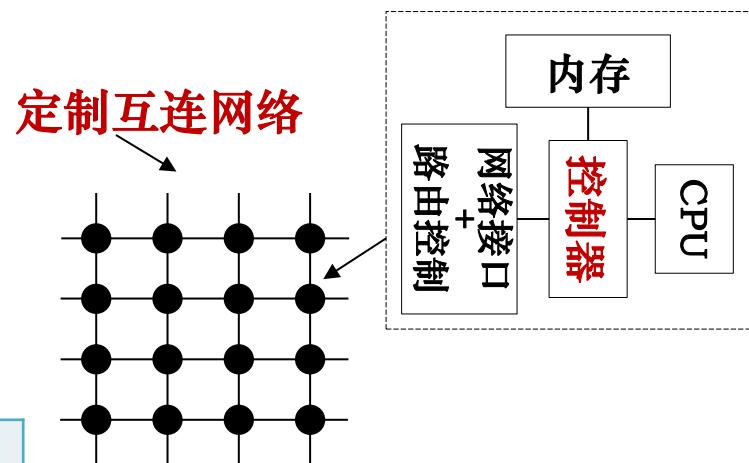
曙光1号（1993年）



多机能力型

- 体系结构：并行多处理架构
- 系统互连：定制互连网络

CPU耦合度	紧密，挂载在内存控制器上
功能	首先支持内存读写原语 其次扩展了消息式通信原语
物理介质	节点间背板走线或铜缆



代表机型：

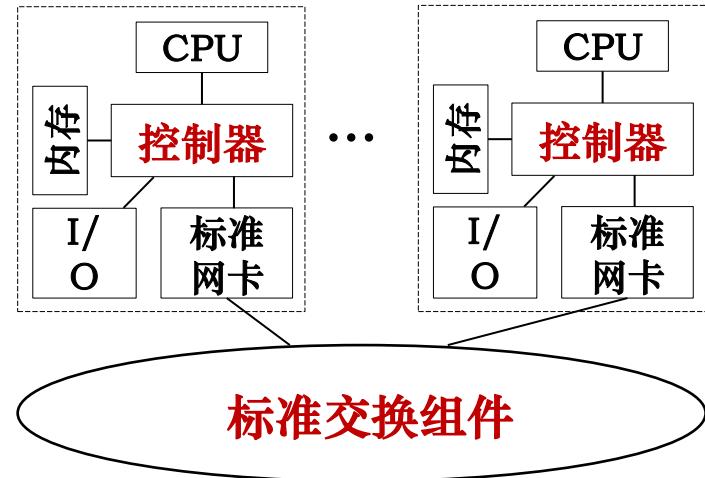
曙光1000 (1995年)
Cray T3E (2002年)



多机容量型

- 体系结构：机群架构
- 系统互连：基于标准的互连网络

CPU耦合度	较松，挂载在I/O总线上
功能	支持内存读写原语 支持消息式通信原语 卸载远程内存直接读写（RDMA）
物理介质	节点间背板走线、铜缆及光纤 (系统规模增大，光纤用于远距离数据传输)

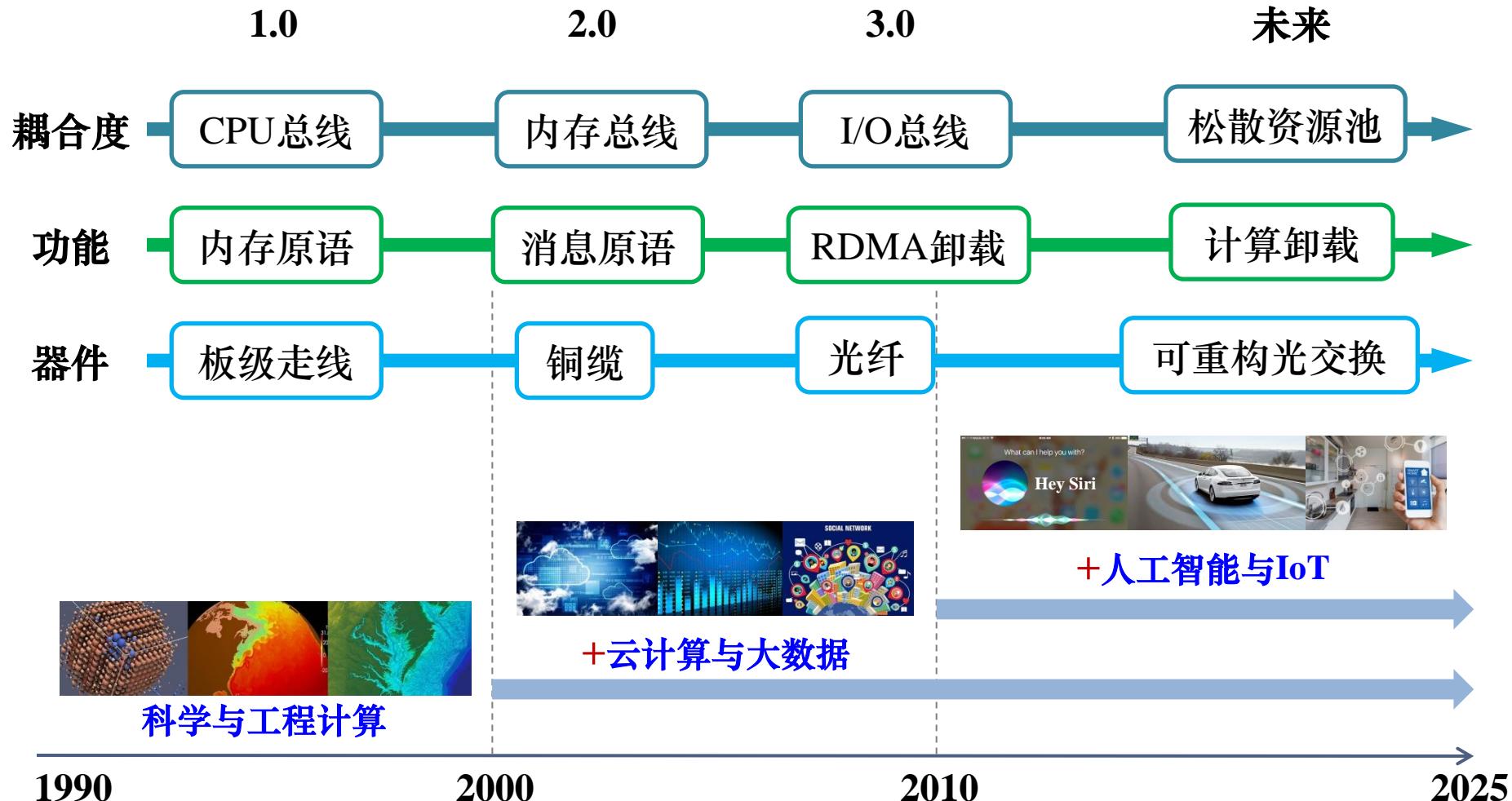


代表机型：

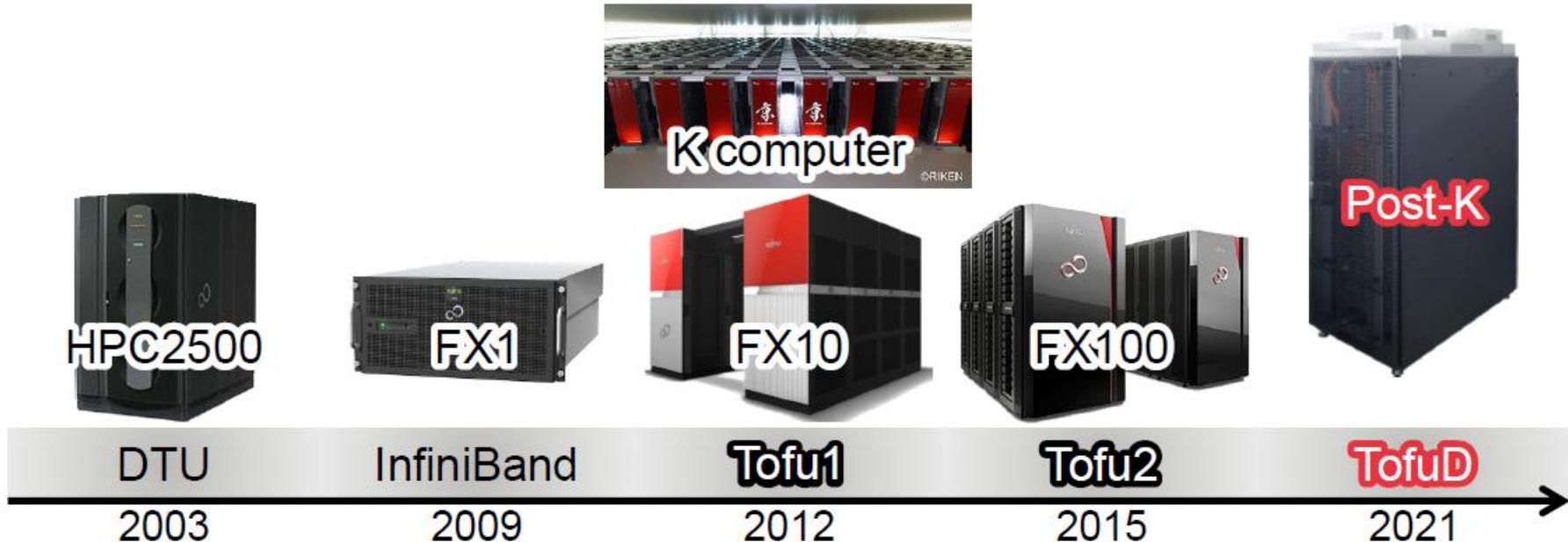
Berkeley NOW (1997年)
曙光2000—7000 (2000年~)



高性能互连网络发展脉络



K-Computer Tofu

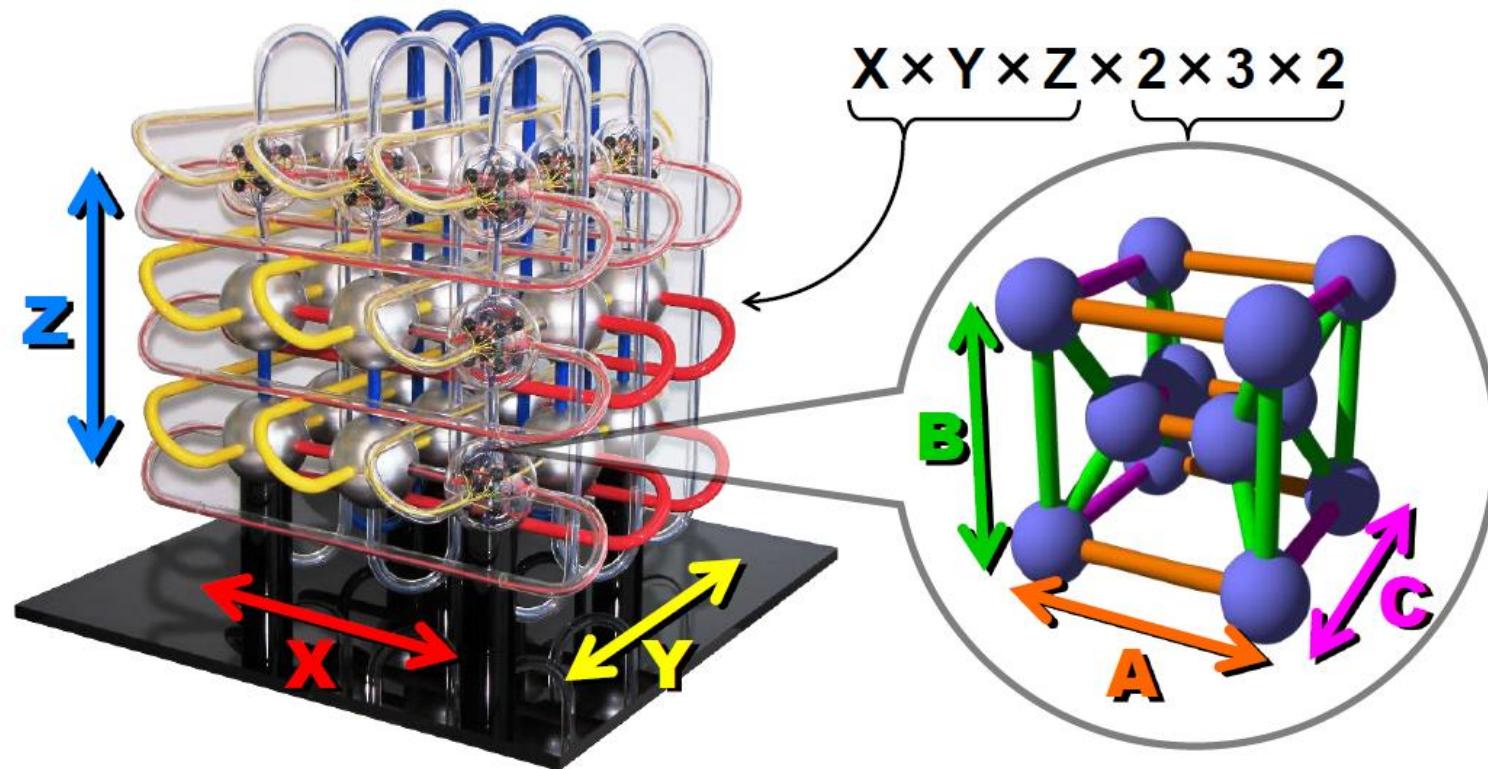


- **初代Tofu**: 用于2011年TOP500 6月榜单第一名的K Computer
 - 特性: 6D Mesh/Torus网络, 可容错, 高可扩展
- **TofuD**: 用于2021年日本的E级计算系统Post-K

- 基本情况
- 特色功能
 - 按需拓扑映射
 - RDMA
 - Memory Bypass
 - Barrier网络
- 具体实现

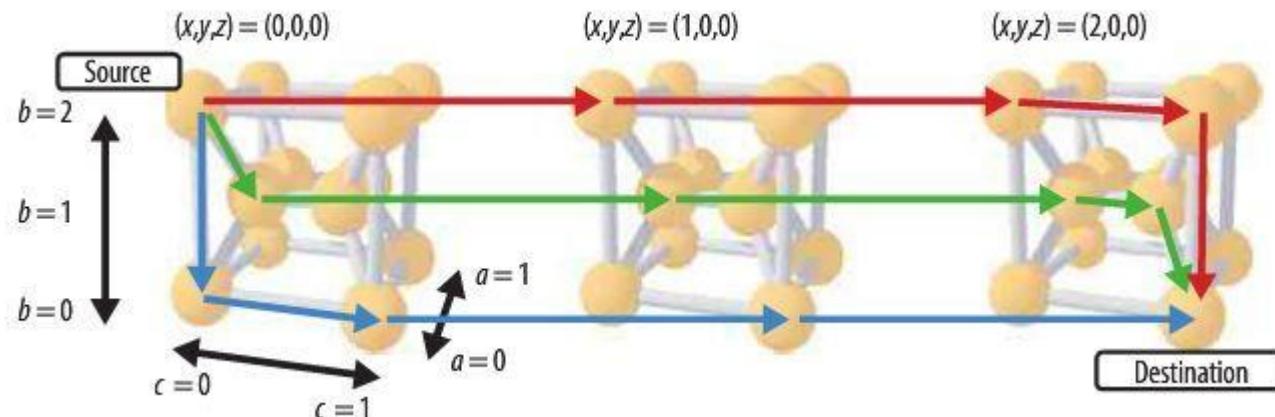
Tofu 6D Mesh/Torus 拓扑结构

- **6D**: 6个坐标维度X、Y、Z, A、B、C
 - A、B、C固定为 $2 \times 3 \times 2$, X、Y、Z根据系统规模扩展
 - 整个网络是 $(A, B, C) \times (X, Y, Z)$ 两个Torus的融合 (**Torus Fusion**)



路由、交换和流控

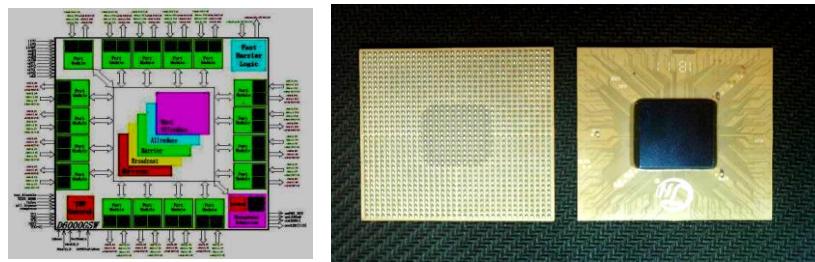
- 维序多路径路由，先走 (A,B,C) 再走 (X,Y,Z)



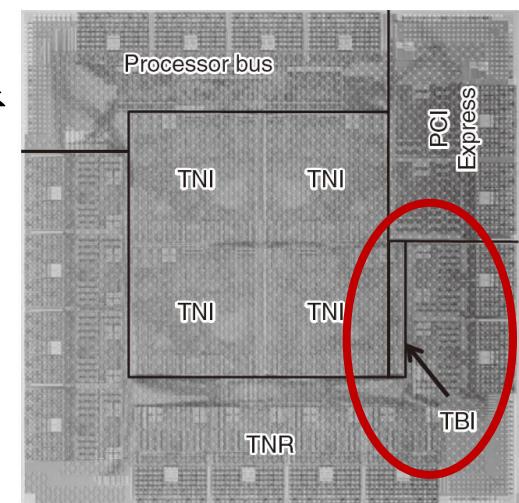
- 每条链路设置4虚通道，2防止路由死锁，2防止协议死锁

特色功能—RDMA、Barrier网络等

- RDMA: Remote Direct Memory Access 远程内存直接访问
 - RDMA Put: 向远程内存写入
 - RDMA Get: 从远程内存读出
 - RDMA Atomic: 修改远程内存中的一个共享变量
- 低延迟支持——Memory Bypass
 - 从CPU寄存器中直接获取通信描述符，消除内存访问开销
 - 将收取到的数据直接写入CPU Cache，消除内存访问开销
- Tofu Barrier
 - 将集合操作如barrier等卸载到网络芯片中执行

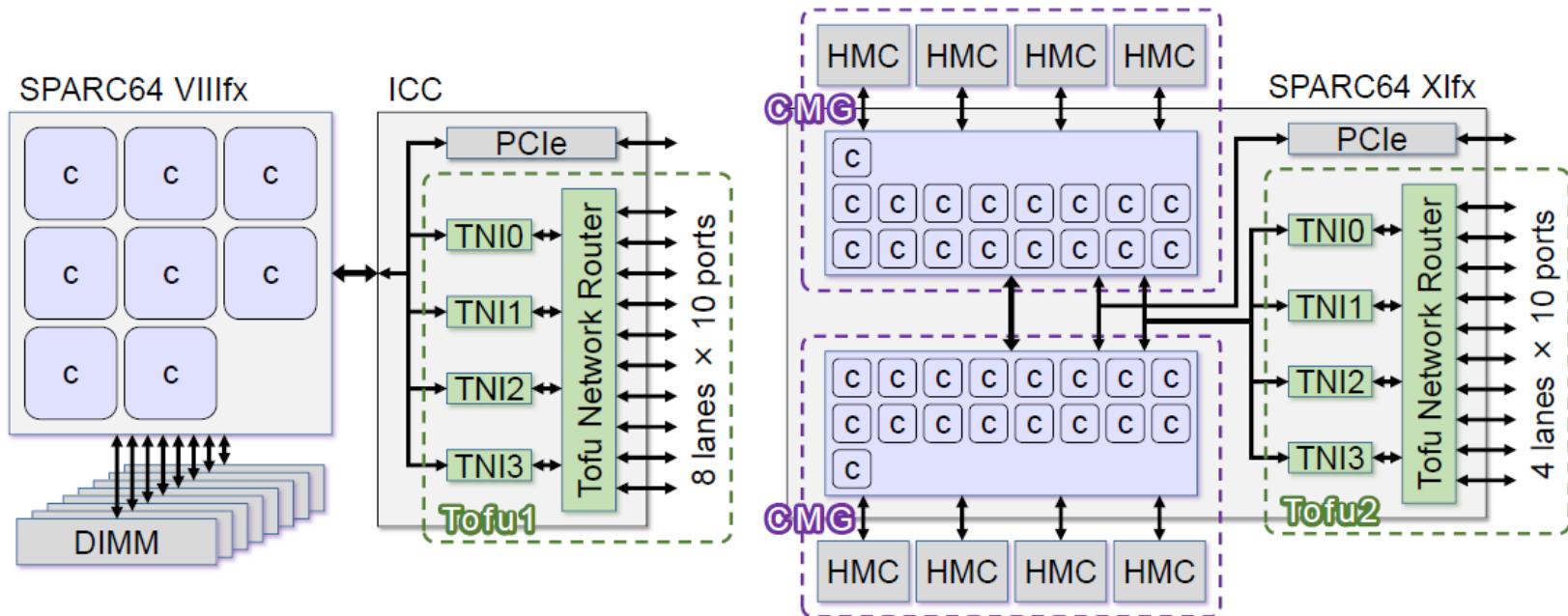


● 曙光5000集操作卸载芯片



具体实现

- Tofu1：独立实现了ICC芯片
 - ICC包含对CPU的网络接口和对网络的路由器接口
 - 10 Port其中4个Port用于互连 $2 \times 3 \times 2$, 6个Port互连X、Y、Z
 - 每个Port包含8条Lane
- Tofu2：将ICC功能集成进了处理器芯片，每Port Lane减少



典型样例分析—Cray XC Aries

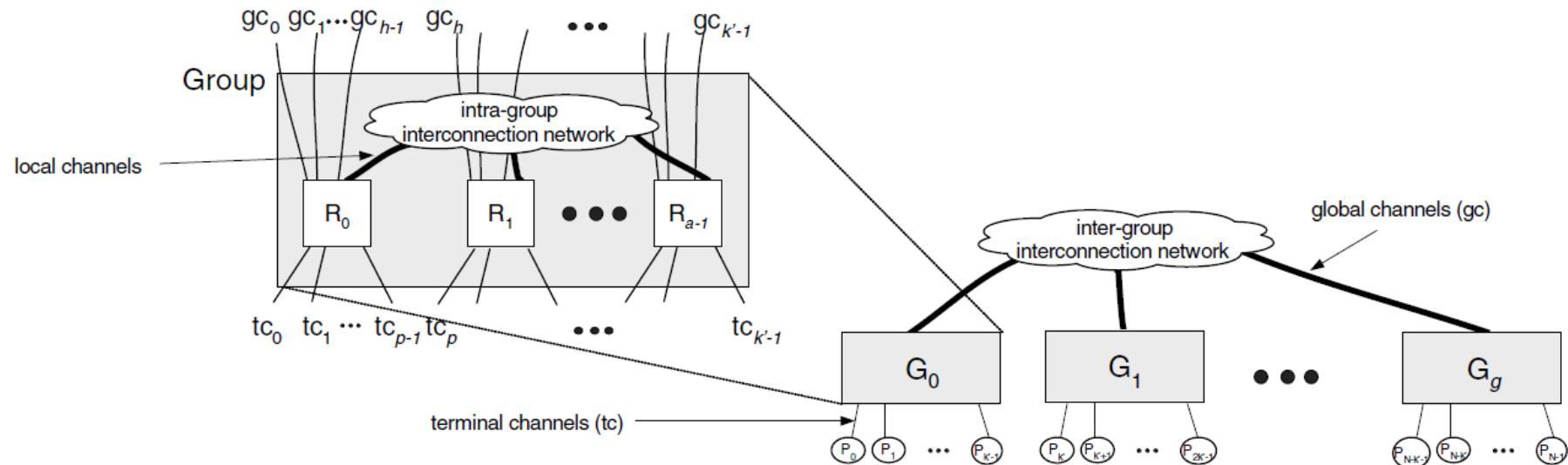


- 2012年起Cray在其新的XC系列高性能计算机产品中推出了Aries互连网络架构，除了更高性能的网络芯片设计之外，其核心是**Dragonfly拓扑**： John Kim, etc. *Technology-Driven, Highly-Scalable Dragonfly Topology, ISCA 2008*

- 基本情况
- 特色功能
 - RDMA
 - 优化短消息传输机制
 - 优化集合操作

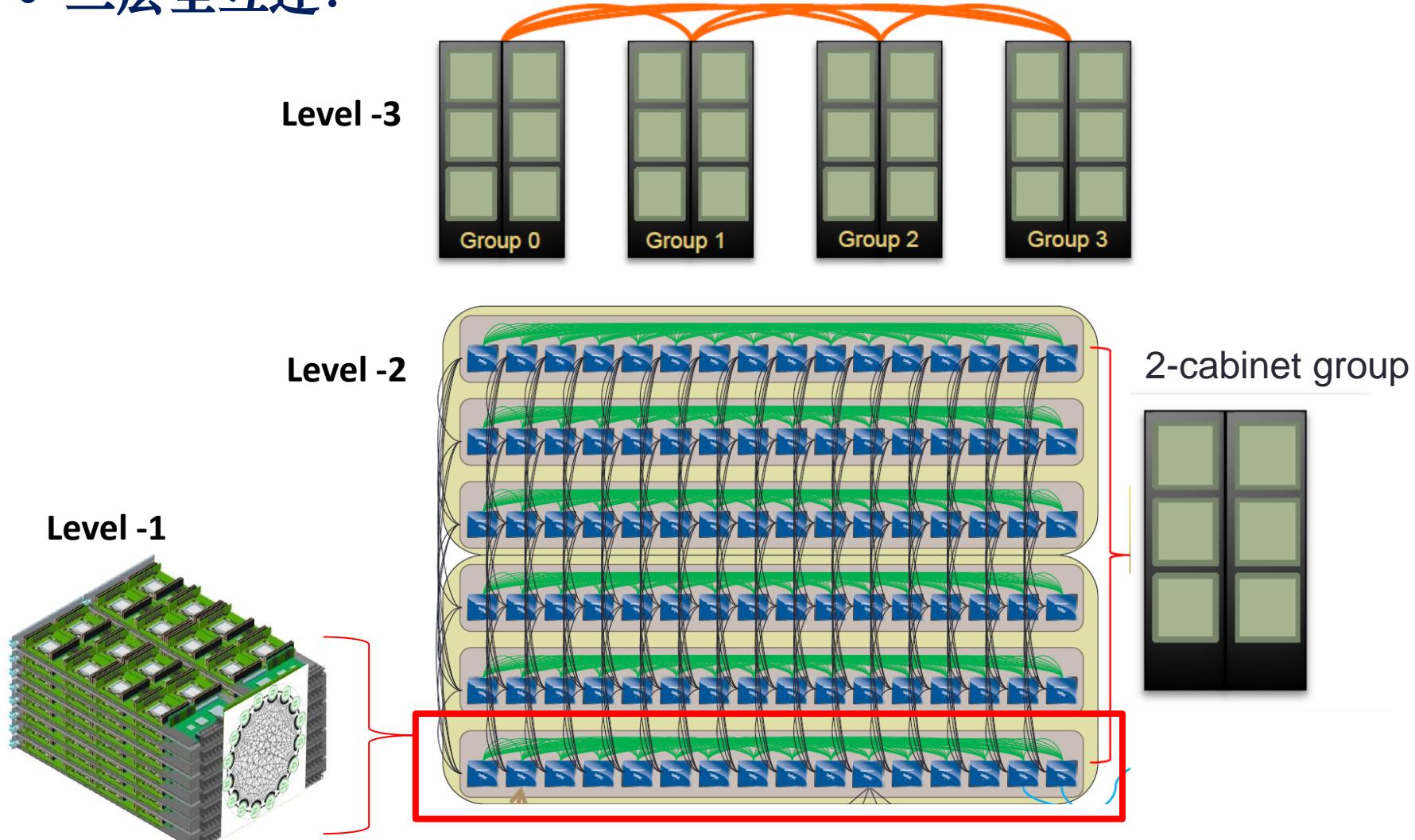
Dragonfly拓扑结构

- 核心思想：缩短网络直径（性能），减少全局连接长线（成本）
- Dragonfly网络分为三个结构层次：
 - 单个路由节点（R），直接连接多个终端
 - 多个路由节点互连形成一个超级节点（G）
 - 多个超级节点互连形成整个网络（Dragonfly）



物理连接关系示意图

- 三层全互连：



特色功能

- RDMA
 - RDMA可以实现全局内存访问
- 优化短消息传输
 - 支持高速率的8-64 Byte的内存原子操作访问
- 优化集合操作
 - 优化卸载MPI_Allreduce集合操作，提高集合操作性能

典型样例分析—TianHe2



TianHe1



TianHe2

TianHe1 于2010.10 TOP500 No.1

TianHe2 于2013.07 TOP500 No.1

国防科大

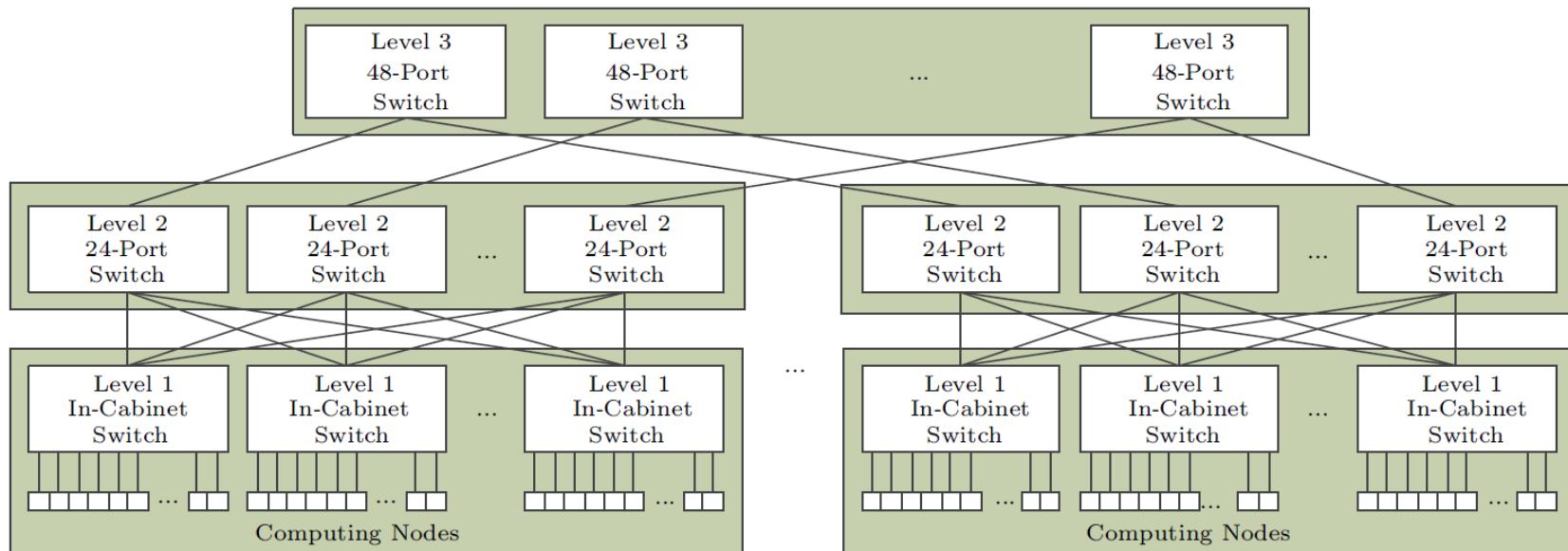
- **TianHe2:** 最多支持18432 计算节点，使用自主网络TianHe Express，共5856 块网络交换芯片NRC、18432 块网络接口芯片NIC

- 基本情况
- 特色功能
 - 虚拟端口及队列机制
 - 多种传输层协议：MP、RDMA等
- 具体实现

TianHe Express

- 拓扑结构：3层胖树

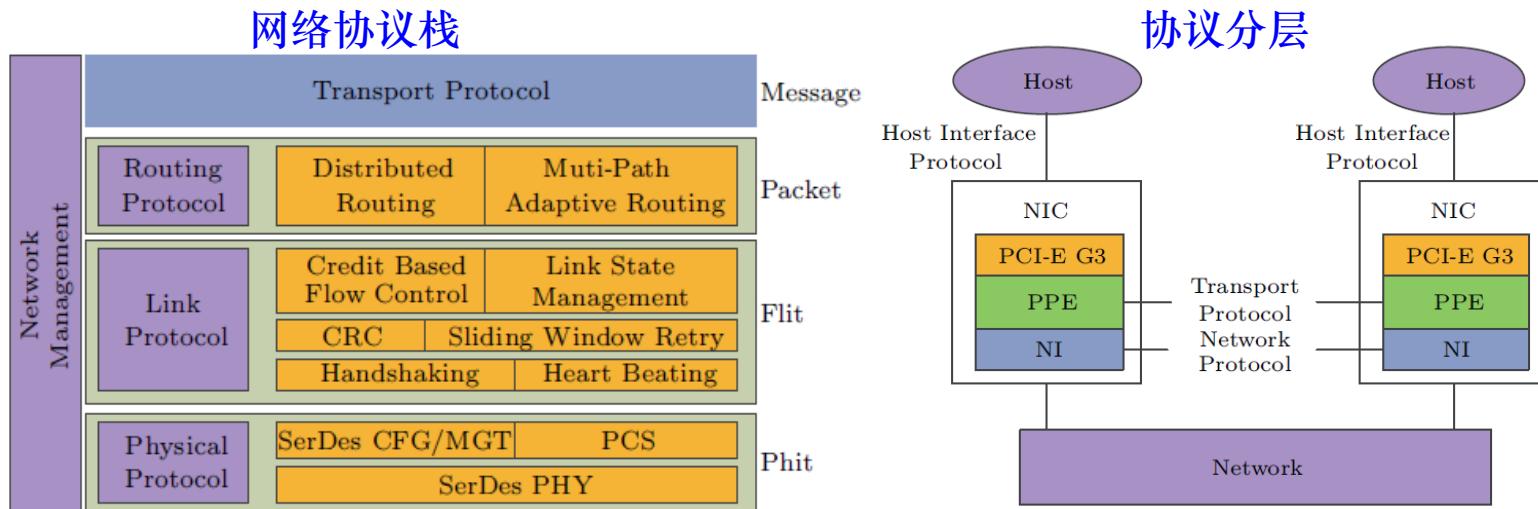
- 每个节点配备1个NIC网络接口
- L1层switch用6个NRC构建（3下3上），L2 switch用1个 NRC构建，L3 switch用6个 NRC构建（内部为两层fat tree，4下2上）
- 以(24, 3) Fat-tree拓扑为基础，下一级到上一级的带宽缩减比为3:1



特色功能

- Host 端多种队列机制

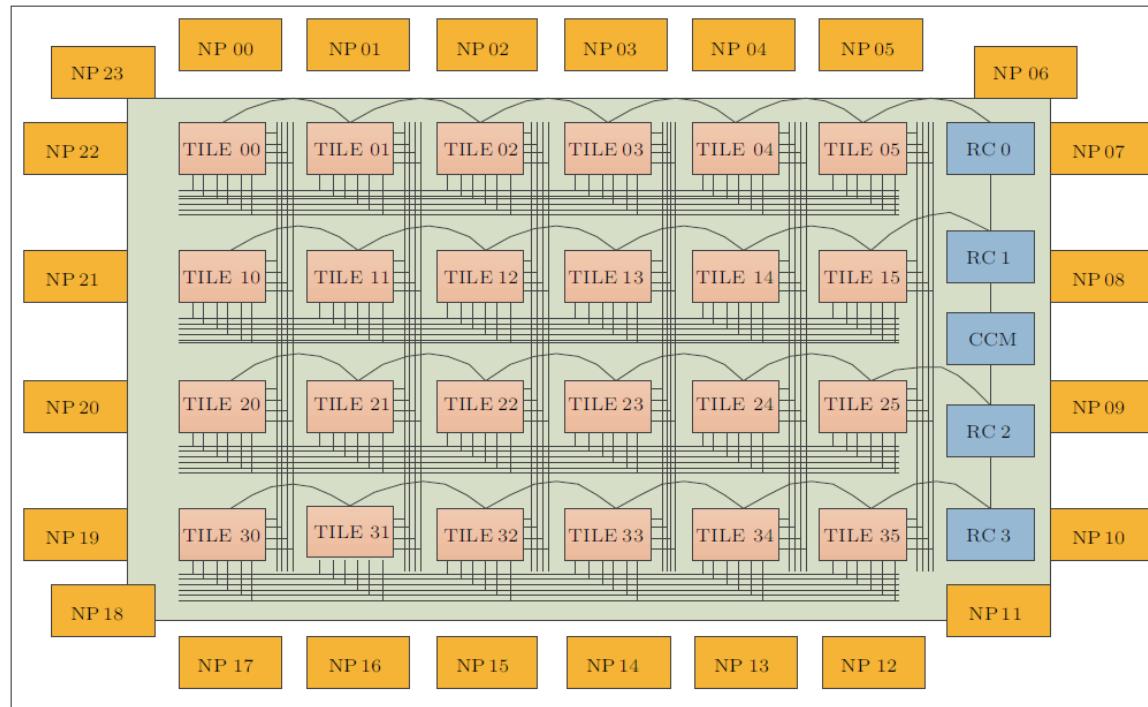
- 最多支持40 Virtual Port， 提供各种用于通信的队列：位于网卡的HDQs、位于内存的SDQs、 MPQ (mini-packet queue) 、 EQ (event queue) 、 INTQ (interrupt queue) 、 EPQ (error packet queue)
- 传输层对不同类型数据使用不同接口
 - MP—mini-packet
 - RDMA—block data
 - Collective Communication



具体实现

- 微体系构——NRC (聚合带宽5.376 Tbps)

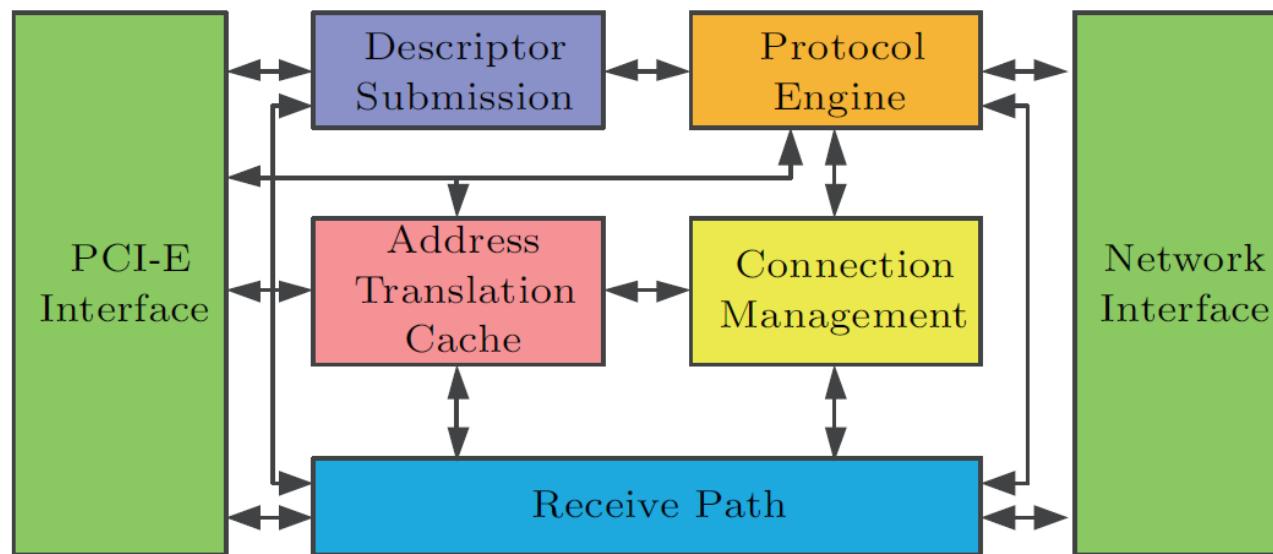
- 24 ports (8-lane, 14Gbps/lane, 224 Gbps 双向带宽, 图中NPs) ;
- 芯片配置管理模块 (CMM) ;
- 24个port对应24个TILE (每个TILE为6*4 的Crossbar)



具体实现

- 微体系构——NIC

- 主机接口：16-lane PCIe 3.0
- 网络接口：8-lane，14Gbps/lane SerDes
- 软硬件交互：Descriptor Submission（管理描述符、提交VP的处理队列）；PE（处理RDMA语义）、ATC（处理地址转换）、CM（处理网络连接）等



典型样例分析—Sunway TaihuLight



2016.06 TOP500 No.1

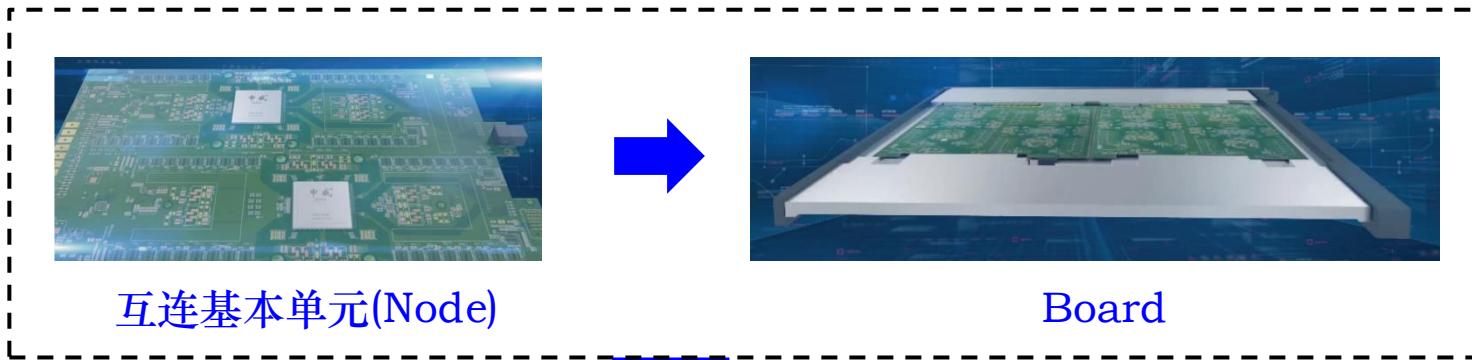
无锡超算中心

- **Sunway TaihuLight** : 40 compute cabinets , 40960 Sunway 26010 CPUs , 8 network cabinets (Mellanox FDR 56Gbps)

- 基本情况
- 特色功能
 - 三套网络，专用网络机柜
- 具体实现

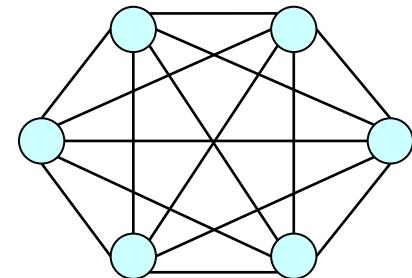
拓扑结构——层次化类3层胖树拓扑

- Level 1: Supernode全互连



supernode

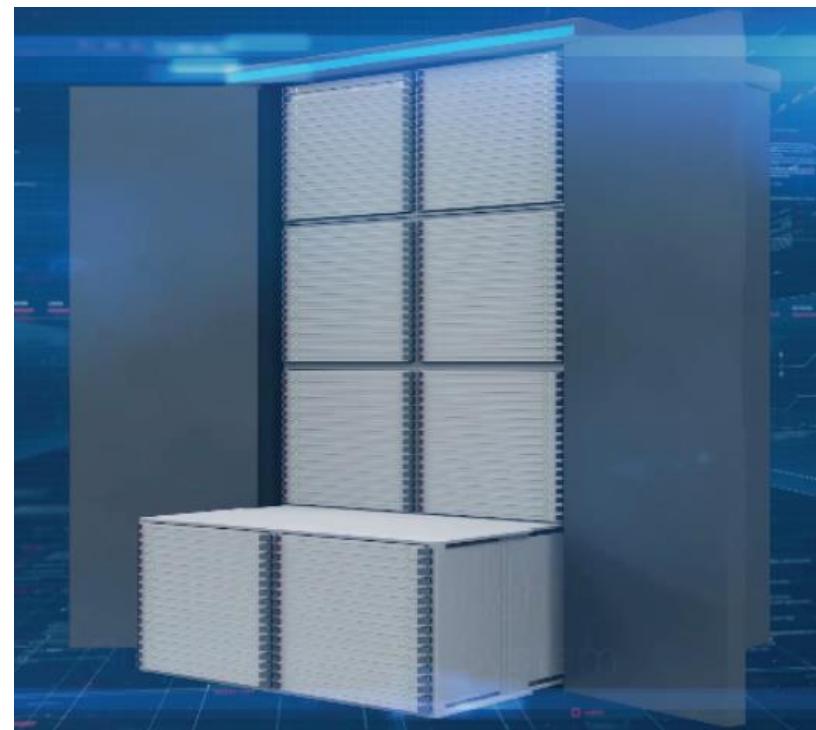
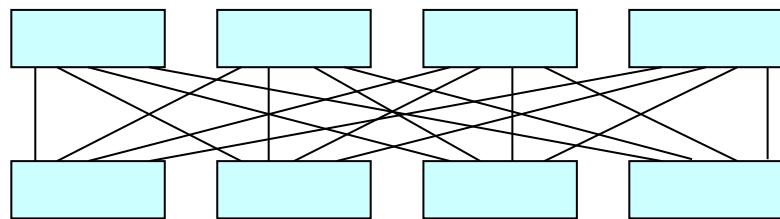
全连接示意



拓扑结构——层次化类3层胖树拓扑

- Level 2: Cabinet

根据胖树拓扑结构猜测

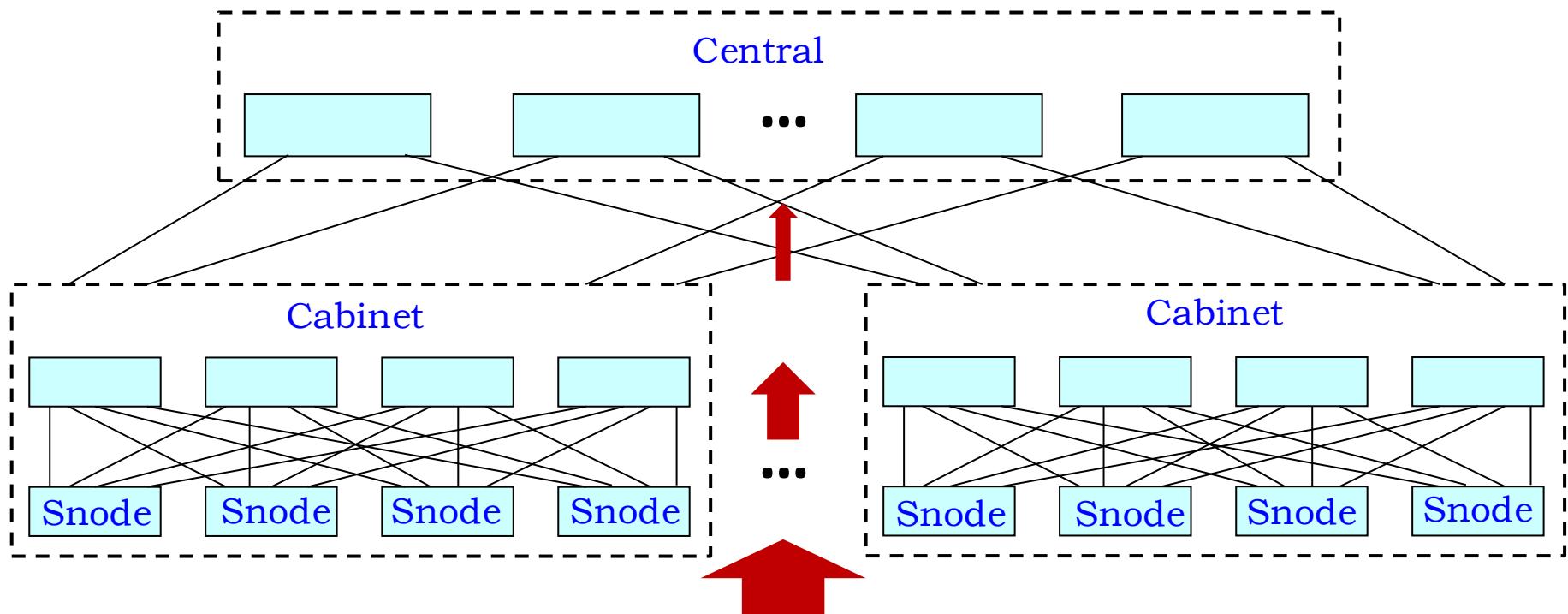


拓扑结构——层次化类3层胖树拓扑

- Level 3: Central Switch Network层次化类胖树拓扑

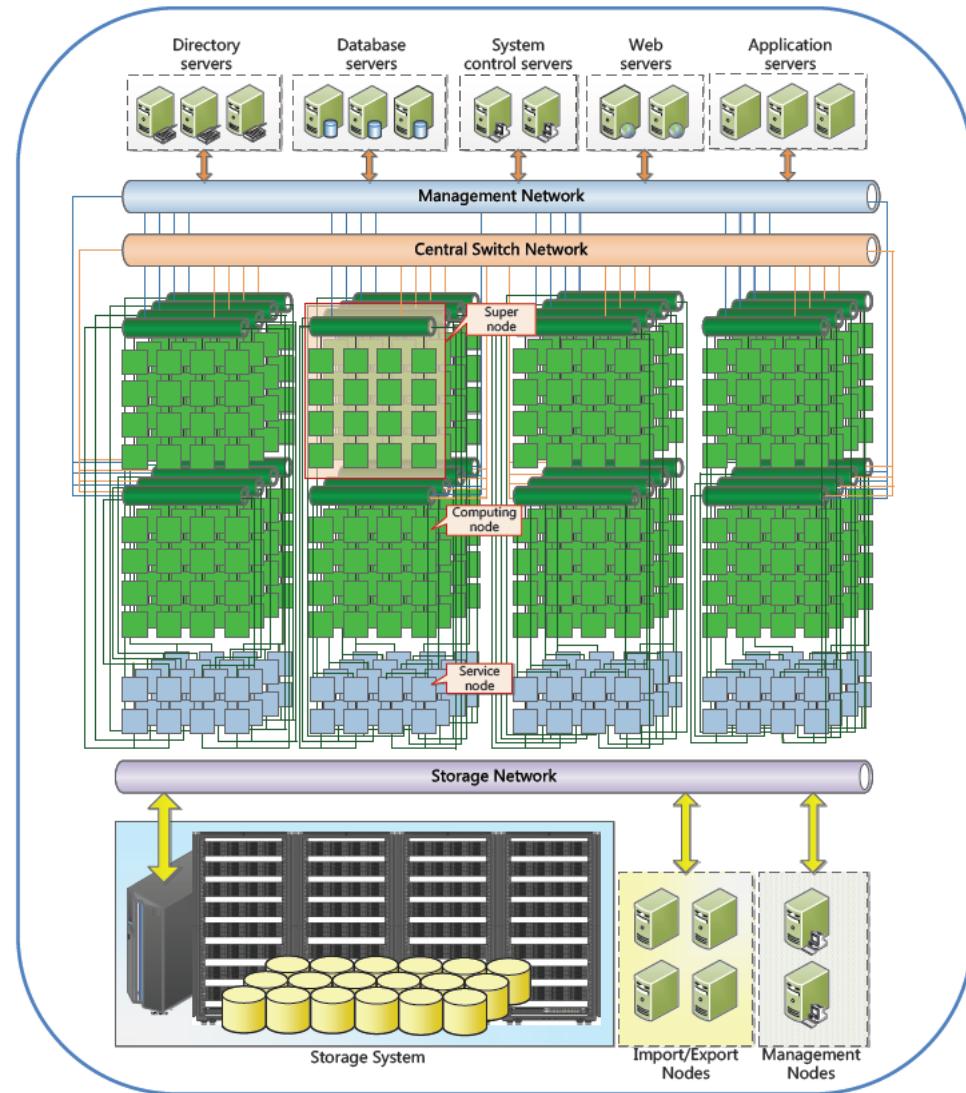
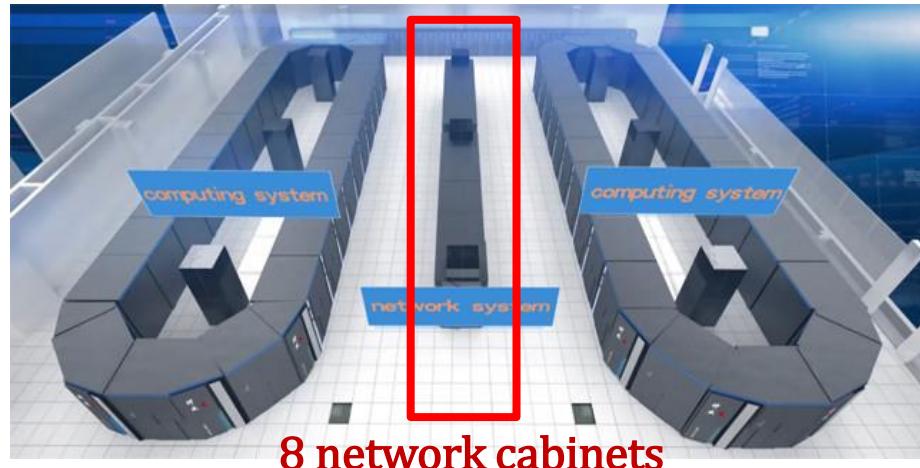
- 总体对分带宽为70 TB/s，网络直径7。
- 中央网络连接不同super nodes，**1/4**网络带宽消减比

根据胖树拓扑结构猜测



特色功能—三套网络

- **Central Switch Network**: 连接super node，每个super node含256 CPU
- **Management Network**: 连接核心计算系统和管理服务器
- **Storage Network**: 连接核心计算网络和存储系统、Import/export节点及管理节点。



具体实现

- 8 network cabinets, 使用Mellanox FDR 56Gbps产品



大规模网络仿真

网络仿真

- **概念**: 基于仿真软件（离散时间驱动，时间戳同步等），对网络组件微体系结构、网络架构、通信协议等进行性能仿真测试
- **功能**: 用户可以使用软件自带库或者自定制的节点模型、链路模型、协议模型等模块组建网络结构，通过一段时间的通信行为模拟运行，收集统计测试数据 (e.g. 时延，吞吐，带宽，能耗等)
- **仿真组件**
 - 节点模型: 终端模型、链路模型、交换节点模型等
 - 协议模型: TCP/IP协议栈, 802.11协议组, ARP, 自定义通信协议等
 - 数据采集模型: 时延采集, 吞吐采集, 误码率采集等

网络软件介绍——NS2/3

- NS-2 (Network Simulator-2)是著名的用于网络研究的离散事件仿真工具，里面包括了大量的用于在有线或无线、本地连接或通过卫星连接进行TCP协议、路由算法、多播协议仿真的网络协议、调度器和工具
- 优点
 - OSI全层次的仿真
 - 详细的过程跟踪和回放
 - 开放源码，有丰富的扩展包
- 缺点
 - 对数据包级进行非常详细的仿真，接近于运行时的数据包数量，使得其无法进行大规模网络的仿真

网络软件介绍——OPNET

- 商业化的通信网络仿真平台

- **优点**

- 丰富的协议支持
- 对网络的QoS进行评价

- **缺点**

- 不开源

网络软件介绍——OMNET++

- 开源的基于组件、离散时间驱动的网络仿真平台，是近年来在科学和工业领域里逐渐流行的一种优秀的网络仿真平台
- 优点
 - 图形界面接口
 - 多操作系统支持
 - 简化的拓扑定义、跟踪和调试
 - 开源支持
- 缺点
 - 必须使用NED编程语言

OMNET++仿真介绍

OMNET++仿真器的组件

- 网络模块例化和关系描述文件
 - 由NED语言编写，使用参数例化网络模块、按特定关系连接不同模块的门、信道等
- 网络包格式定义文件
 - OMNET++本身提供的网络包类型具备一些简单参数，用户还可以根据具体要求通过网络包格式定义文件定义具体字段
- 网络模块功能描述文件
 - 网络模块功能的行为描述文件，使用C++编写，功能包括缓存、路由、交换等；
- 仿真内核
 - OMNET提供的仿真类库代码
- 用户接口
 - 该接口用于仿真运行时的调试、演示等工作

网络拓扑描述文件

- **使用NED语言描述**

- NED可以描述一组元件（通道，简单/复杂模型），这些组件的描述可以在其他网络描述中得以重用

- **NED语言**

- 文件带有.Ned的后缀，.Ned文件动态地载入到模拟程序，或者用Ned编译器或C++代码链接到模拟器执行
- 可以使用任何文本编辑器或GNED图形编辑器来编写。NED语言用来定义模型中的网络拓扑结构，较为简单的网络拓扑可以使用GNED，但复杂网络的拓扑描述还应该用NED源文件方式书写
- 可引进其它网络描述文件，引进一个网络描述后，可以使用它所包含的模块通道等组件，如：import "ethernet"; // imports ethernet.ned

网络拓扑描述文件示例

7x9 Mesh拓扑描述文件

```
package networks;

import node.Node;

network Mesh 拓扑名称
{
    parameters:
        double height @prompt("Number of rows") = default(9);
        double width @prompt("Number of columns") = default(7);

    types:
        channel C extends ned.DatarateChannel
        {
            parameters:
                delay = default(0.1ms);
                datarate = default(1Gbps);
        }

    submodules:
        node[height*width]: Node {
            parameters:
                address = index;
                @display("p=,,m,$width,40,40");
            gates:
                port[4];
        }

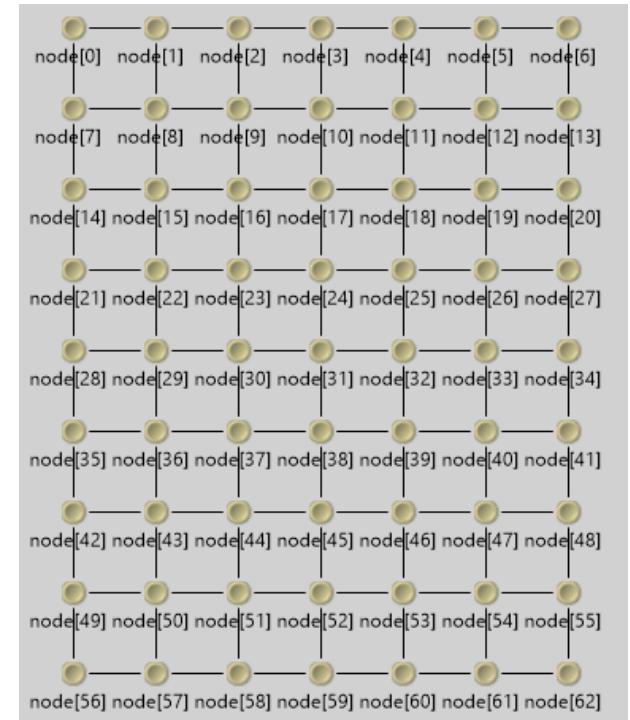
    connections allowunconnected:
        for i=0..height-1, for j=0..width-1 {
            node[i*width+j].port[0] <--> C <--> node[(i+1)*width+j].port[1] if i!=height-1;
            node[i*width+j].port[2] <--> C <--> node[(i*width+j)+1].port[3] if j!=width-1;
        }
}
```

结构参数

通道参数

节点参数

节点连接关系



图形化输出

网络包格式定义文件和示例

- 网络包格式定义由msg文件实现

- 直接新建一个msg文件是比较快的一种定义网络分组格式的方法，以下图为例，定义好这些字段后，仿真器在编译时自动生成packet_m.cc 和packet_m.h文件，对这些字段作为变量进行声明，并生成setter/getter函数，以及copy等基本功能函数

```
19 packet Packet {  
20     int srcAddress;  
21     int destAddress;  
22     int head;  
23     int body;  
24     int tail;  
25     int vc;
```

网络模块功能描述文件

- **简单模块**

- 复合模块的基本构建成分，它通过申明它的参数和门（gate，有input, output, inout方向，用来连接模块）来定义

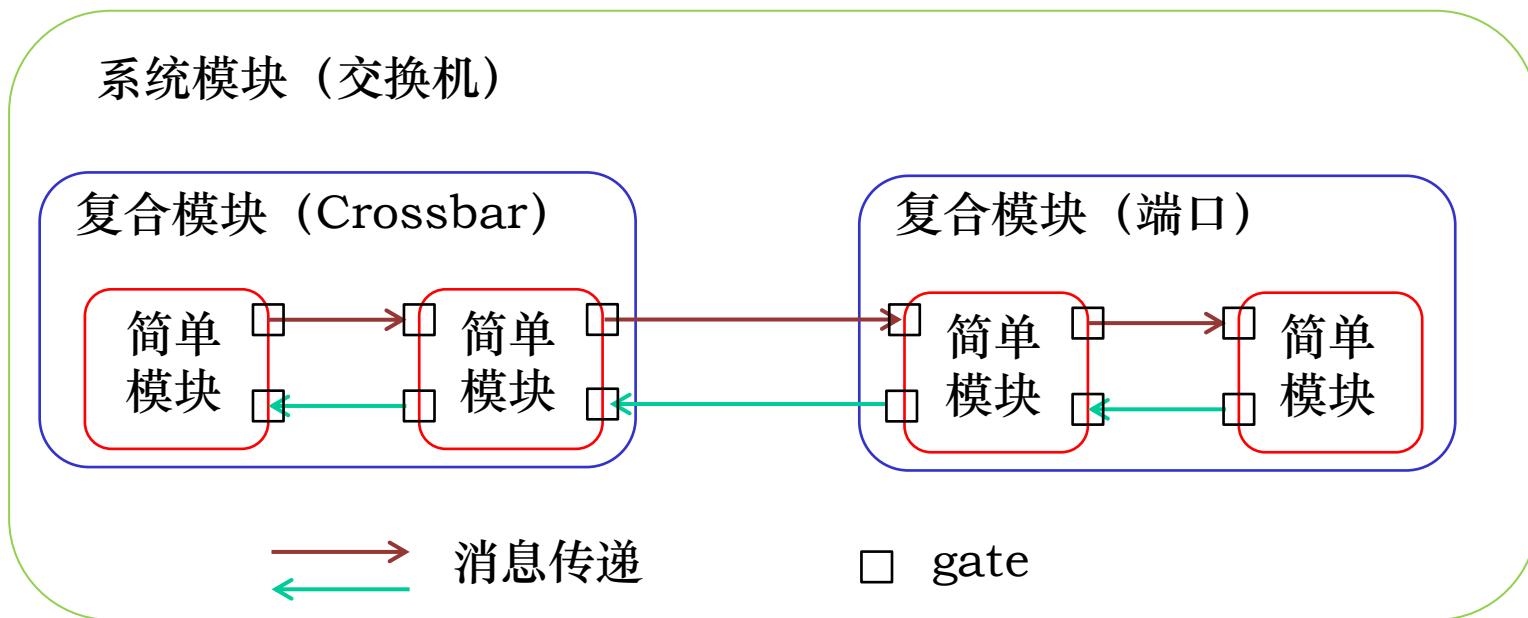
- **复合模块**

- 它由一个或多个子模块组成。不管是简单模块还是复合模块都可以用做子模块。它们也都能有门和参数，在简单模块能够使用的任何地方复合模块都能使用。它还有两个部分：子模块和链接

- **系统模块**

- 系统模块是将各种复合模块整合在一起，形成对用户来说最顶层的一个网络结构(e.g. Mesh, Torus ...)

网络模块功能描述文件简单示例



网络功能模块描述代码示例

```
class L2Queue : public cSimpleModule
{
    private:
        long frameCapacity;

        cQueue queue;
        cMessage *endTransmissionEvent;
        bool isBusy;

        simsignal_t qLenSignal;
        simsignal_t busySignal;
        simsignal_t queueingTimeSignal;
        simsignal_t dropSignal;
        simsignal_t txBytesSignal;
        simsignal_t rxBytesSignal;

    public:
        L2Queue();
        virtual ~L2Queue();

    protected:
        virtual void initialize() override;
        virtual void handleMessage(cMessage *msg) override;
        virtual void refreshDisplay() const override;
        virtual void startTransmitting(cMessage *msg);

};

Define_Module(L2Queue);
```

继承父类，简单模块

自定义成员变量，其中c开头的变量类型基本都是OMNET自带的数据结构

自定义信号（OMNET中的一种数据结构，一般用于数据统计）

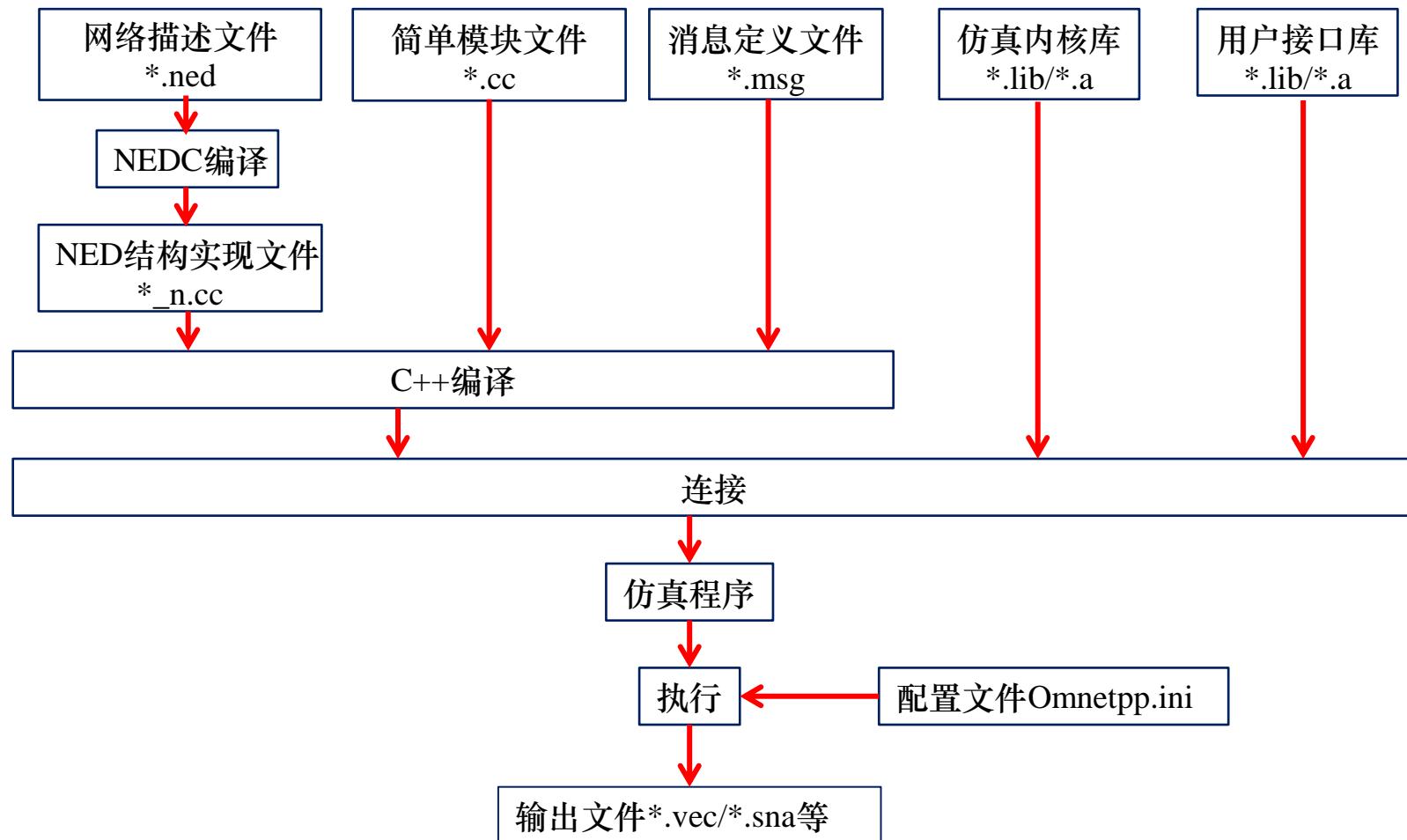
必要的构造函数

必要的override函数

自定义函数

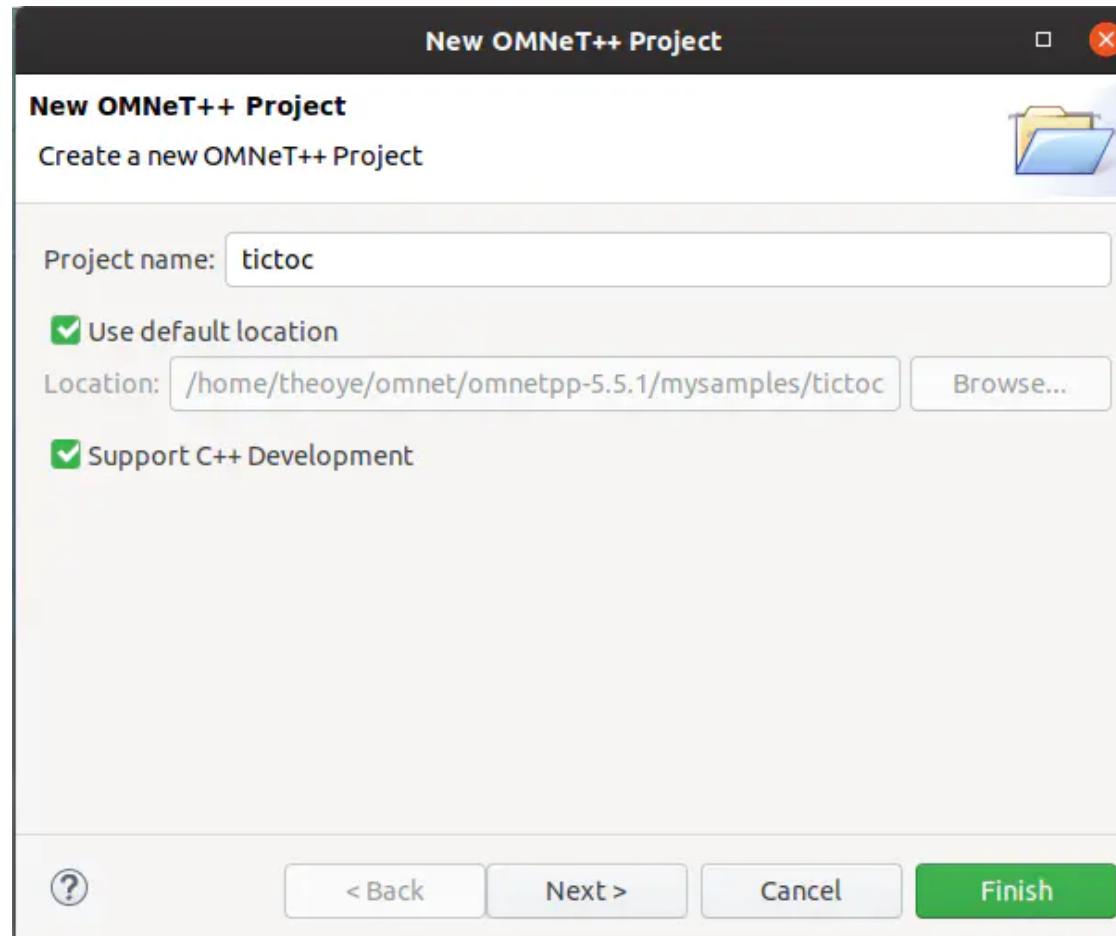
模块注册声明！以便ned文件可对其端口和连接关系进行定义

OMNET++仿真流程



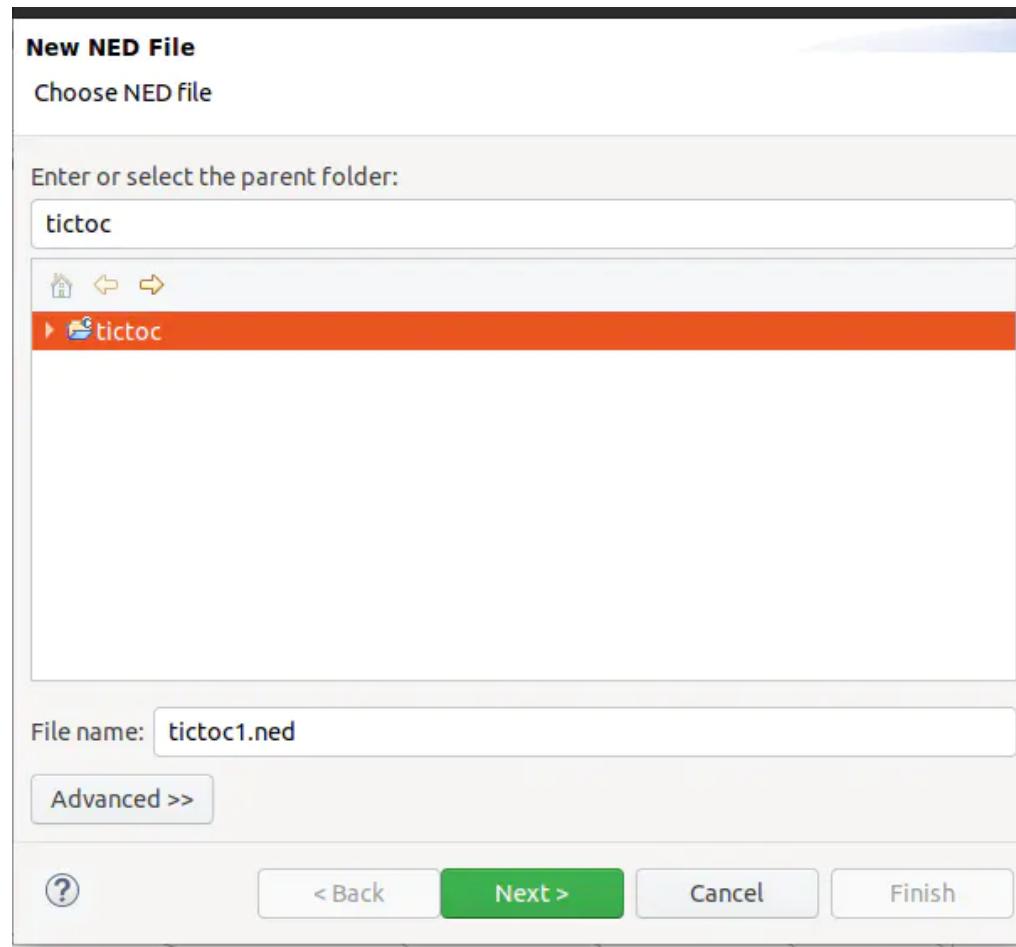
OMNET++仿真过程——新建工程

From: <https://www.jianshu.com/p/3b22b3e7ca10>



添加NED文件

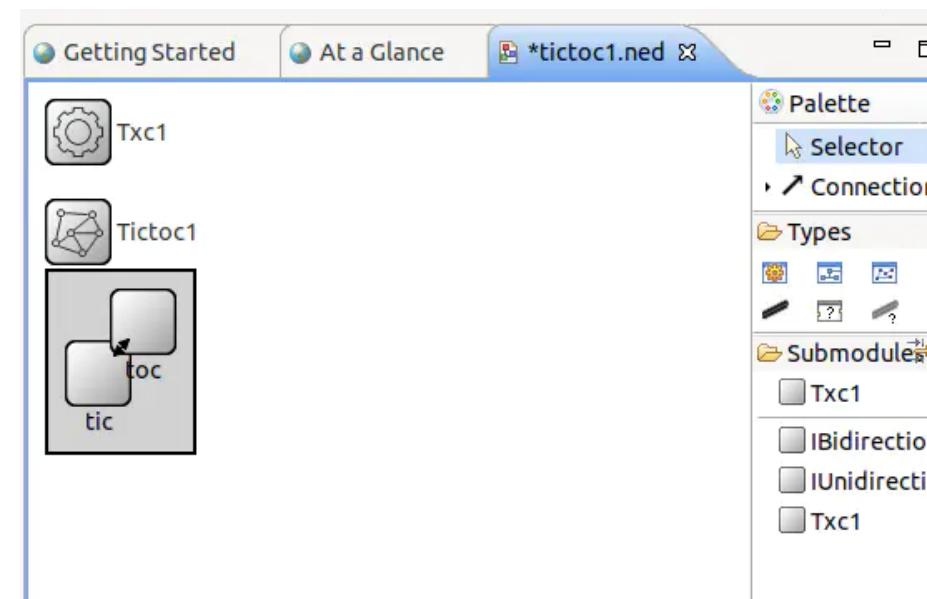
- NED文件有两种模式：源码模式和设计模式



编辑NED文件实现特定拓扑

- 编辑NED文件，两个简单模块Txc1直连，一发一收

```
1 simple Txc1
2 {
3     gates:
4         input in;
5         output out;
6 }
7 //
8 // Two instances (tic and toc) of Txc1 connected both ways.
9 // Tic and toc will pass messages to one another.
10 //
11 network Tictoc1
12 {
13     submodules:
14         tic: Txc1;
15         toc: Txc1;
16     connections:
17         tic.out --> { delay = 100ms; } -> toc.in;
18         toc.in <- { delay = 100ms; } <- toc.out;
19     }
20 }
```



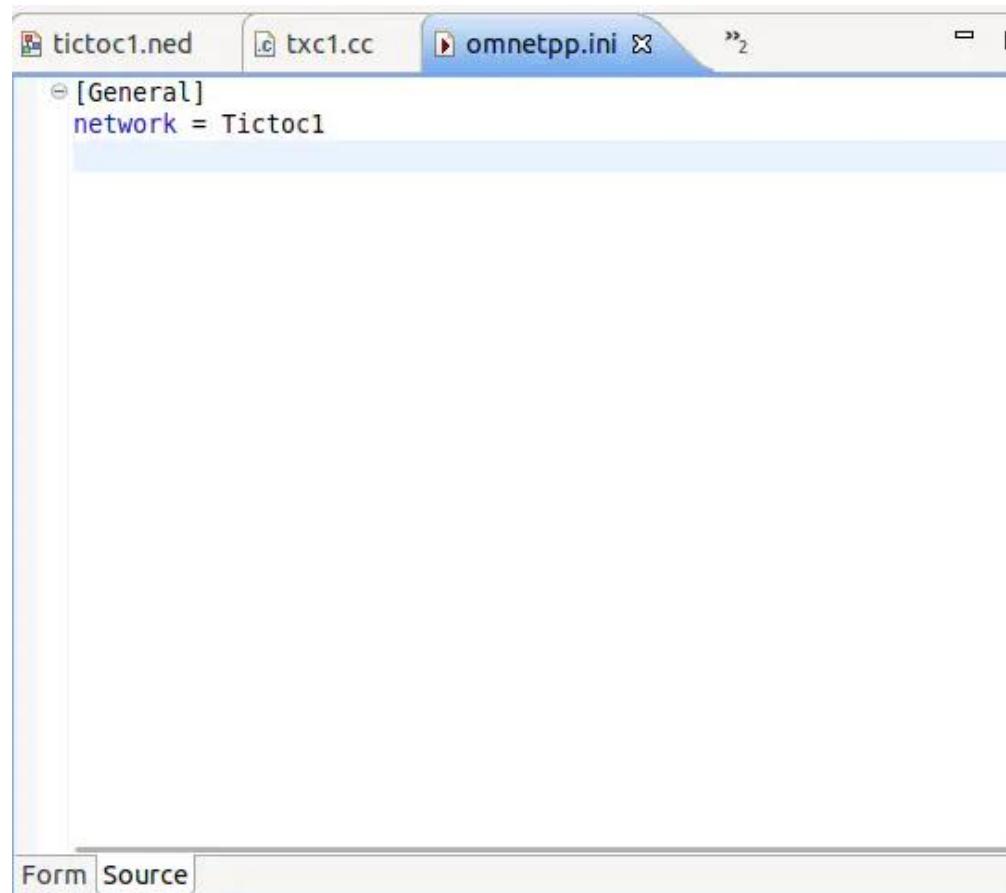
编辑模块行为描述C++文件

```
1 #include <string.h>
2 #include <omnetpp.h>
3
4 using namespace omnetpp;
5
6 /**
7  * Derive the Txc1 class from cSimpleModule. In the Tictoc1 network,
8  * both the `tic` and `toc` modules are Txc1 objects, created by OMNeT++
9  * at the beginning of the simulation.
10 */
11 class Txc1 : public cSimpleModule
12 {
13     protected:
14         // The following redefined virtual function holds the algorithm.
15         virtual void initialize() override;
16         virtual void handleMessage(cMessage *msg) override;
17 };
18
19 // The module class needs to be registered with OMNeT++
20 Define_Module(Txc1);
21
22 void Txc1::initialize()
23 {
24     // initialize is called at the beginning of the simulation.
25     // To bootstrap the tic-toc-tic-toc process, one of the modules needs
26     // to send the first message. Let this be 'tic'.
27
28     // Am I Tic or Toc?
29     if (strcmp("tic", getName()) == 0) {
30         // create and send first message on gate "out". "tictocMsg" is an
31         // arbitrary string which will be the name of the message object.
```

```
32             cMessage *msg = new cMessage("tictocMsg");
33             send(msg, "out");
34         }
35     }
36
37     void Txc1::handleMessage(cMessage *msg)
38     {
39         // The handleMessage() method is called whenever a
40         // message arrives at the module. Here, we just send it to the other
41         // module on the same link. Because both 'tic' and 'toc' does this,
42         // the message will bounce between the two.
43         send(msg, "out"); // send out the message
44     }
45 }
```

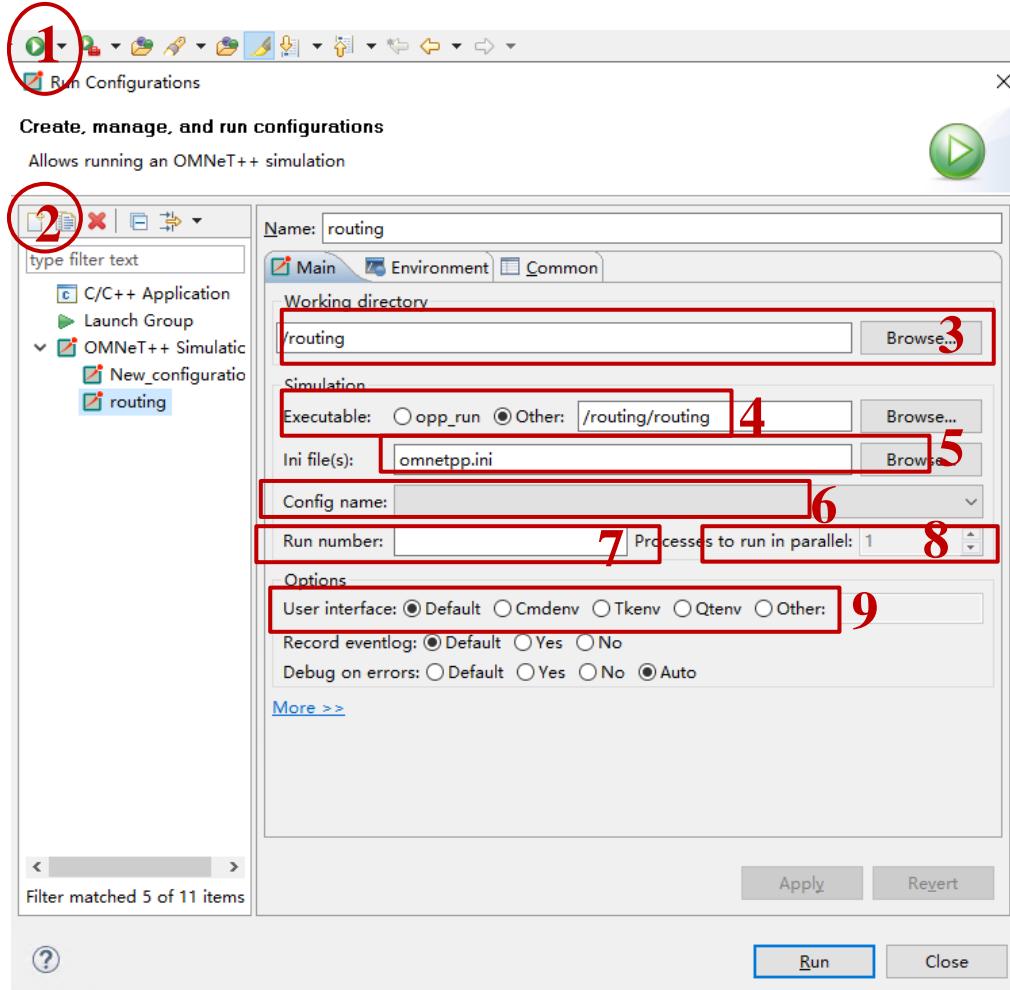
添加omnetpp.ini配置文件

- 文件告诉模拟器程序你想要模拟哪一个网络(由于ned文件可以包含多个网络)，还可以显示的指定随机数生成器的随机数种子



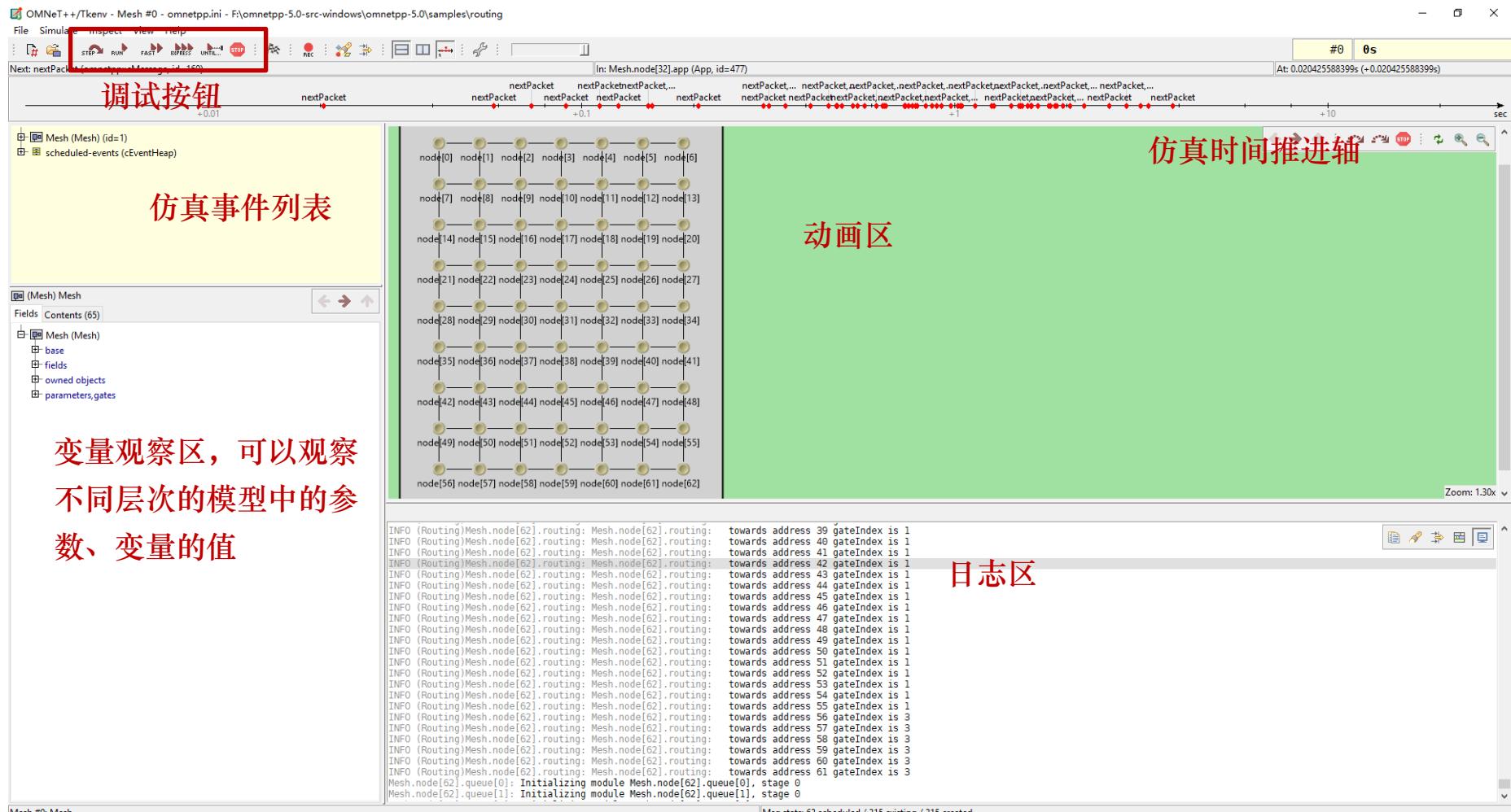
运行仿真

● 仿真运行配置



1. 选择run configurations
2. 新建run configuration
3. 选择工程目录
4. 选择执行文件(首次运行可以选opp_run, 待编译生成可执行文件之后再选)
5. 选择ini配置文件
6. 可选ini文件中的不同config
7. 运行次数(repeat参数有关)
8. 运行时的进程数(Cmdenv模式下有效)
9. 运行模式:
 - default,
 - Cmdenv为命令行模式
 - Tkenv图形化界面
 - Qtenv图形化界面

仿真界面



课程练习

- 第1题

- 以tictoc为例，熟悉仿真器运行流程

- 第2题

- 以sample/routing为例，改写路由算法部分实现x优先维序路由，掌握使用selfmsg为基础的仿真推进方式

- 第3题

- 实现一个Torus网络，要求1) 规模可定义；2) 实现x优先维序路由算法；3) 实现热点区域流量模式

总结

- 对并行计算来说，主要研究系统域互连网络
- 系统域互连网络影响并行系统的性能和扩展性
 - 系统互连N个节点的代价是什么：延迟、带宽、功耗、成本等
- 设计并行系统互连的几个要素
 - 互连接口、拓扑、路由、交换、流控、链路
- 互连网络设计的第一步：网络仿真

谢谢！