

**Advanced Computer Networks**

# **Flow Scheduling in Data Center Networks (I)**

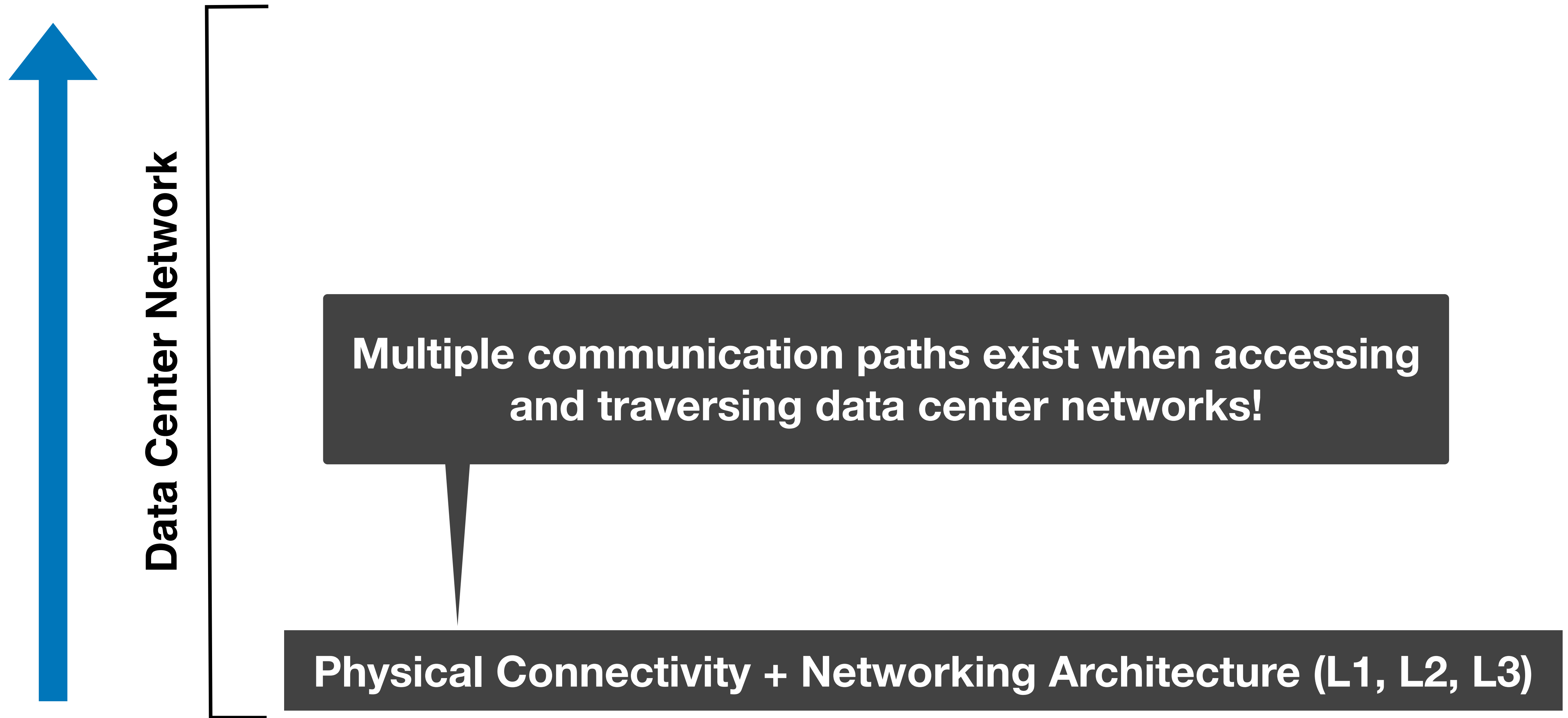
**<https://pages.cs.wisc.edu/~mgliu/CS740/F26/index.html>**

**Ming Liu**  
**mgliu@cs.wisc.edu**

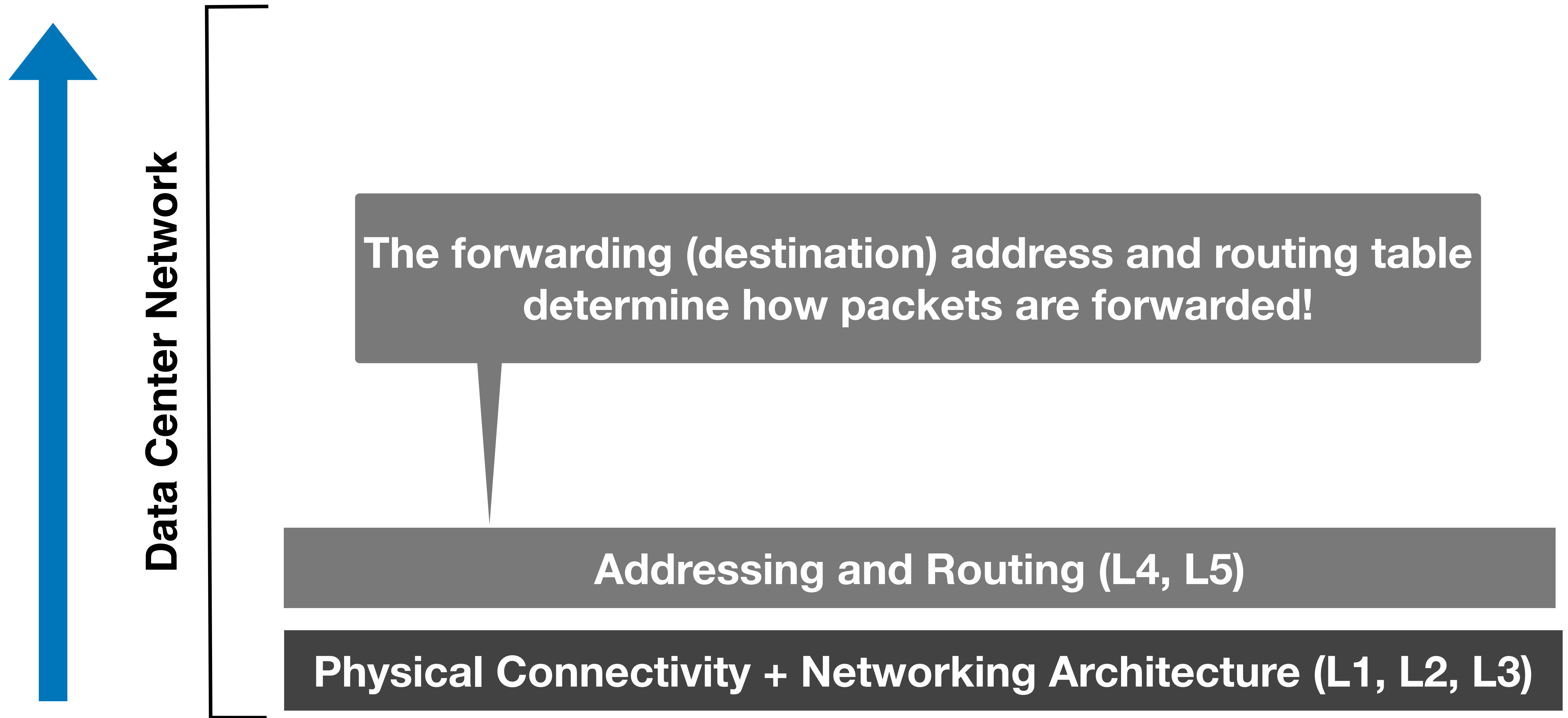
# Outline

- Last lecture
  - Addressing and routing in data center networks (II)
- Today
  - Flow scheduling in data center networks (I)
- Announcements
  - Project proposal due 10/12/2026 11:59 PM
  - Lab1 due 10/09/2026 11:59 PM

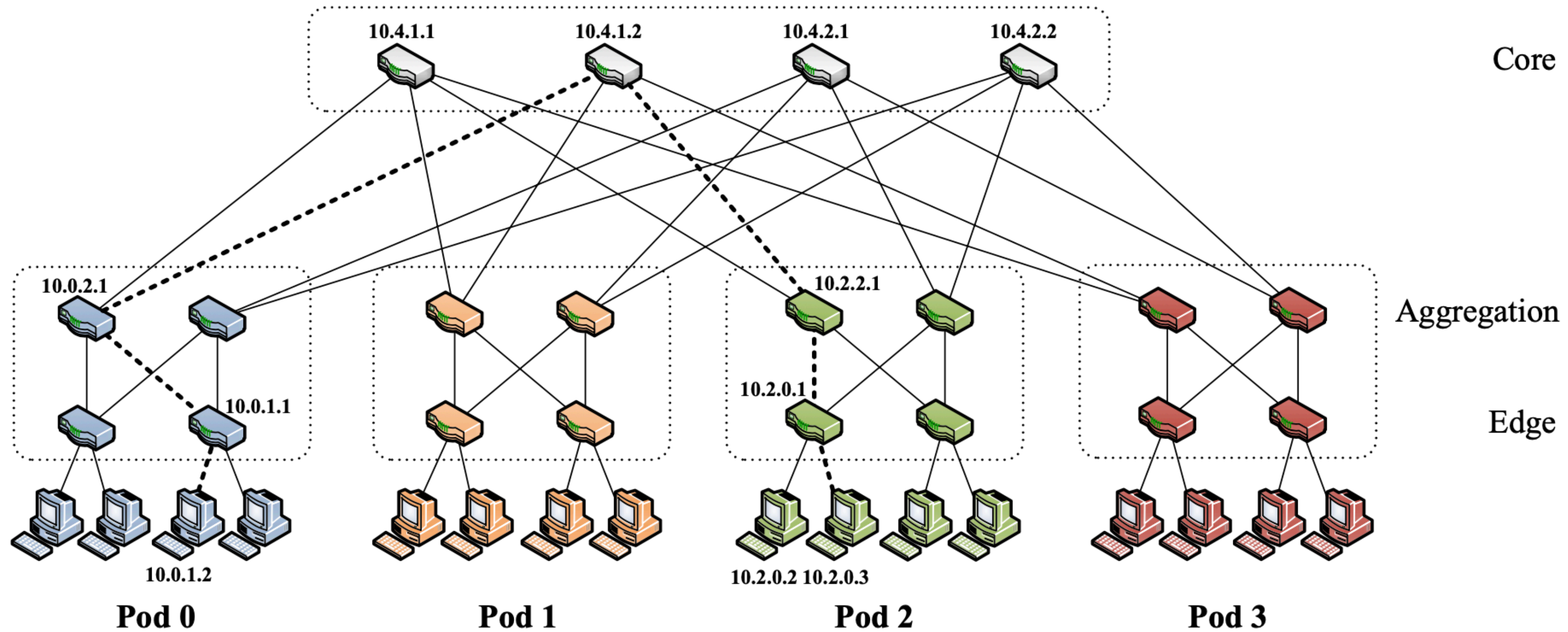
# Where we are?



# Where we are?

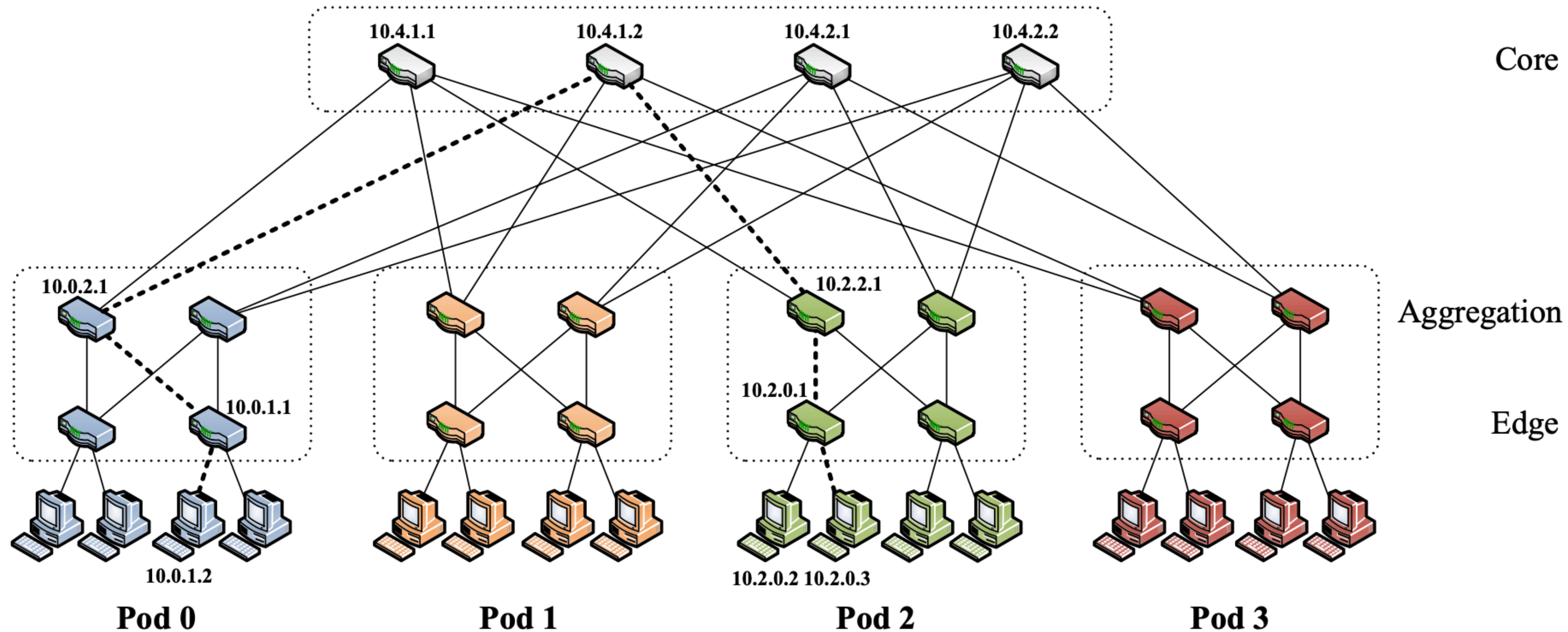


# Which path should a network flow take?



What is a “network flow”?

Which path should a **network flow** take?



# Network Flow

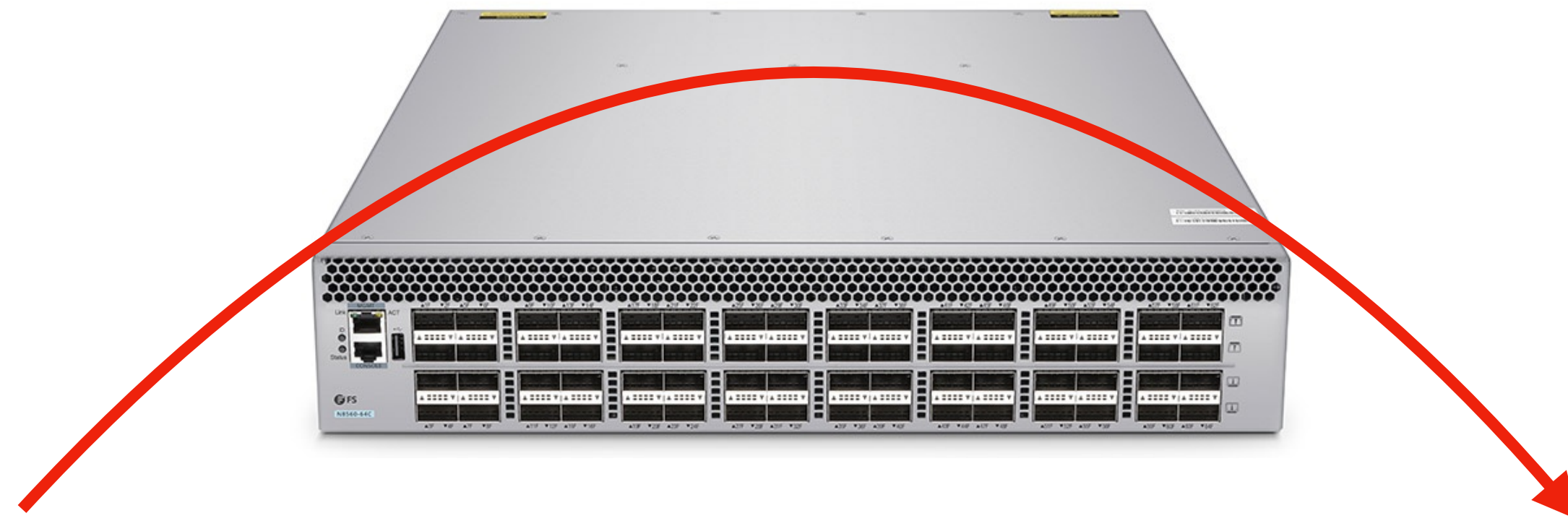
- A unique identifier in the data center networks

# Network Flow

- A unique identifier in the data center networks: 5-Tuple
    - Source IP
    - Destination IP
    - Source Port
    - Destination Port
    - Protocol Number
- } Host-Host communication channel
- } App-App communication channel

# Network Flow

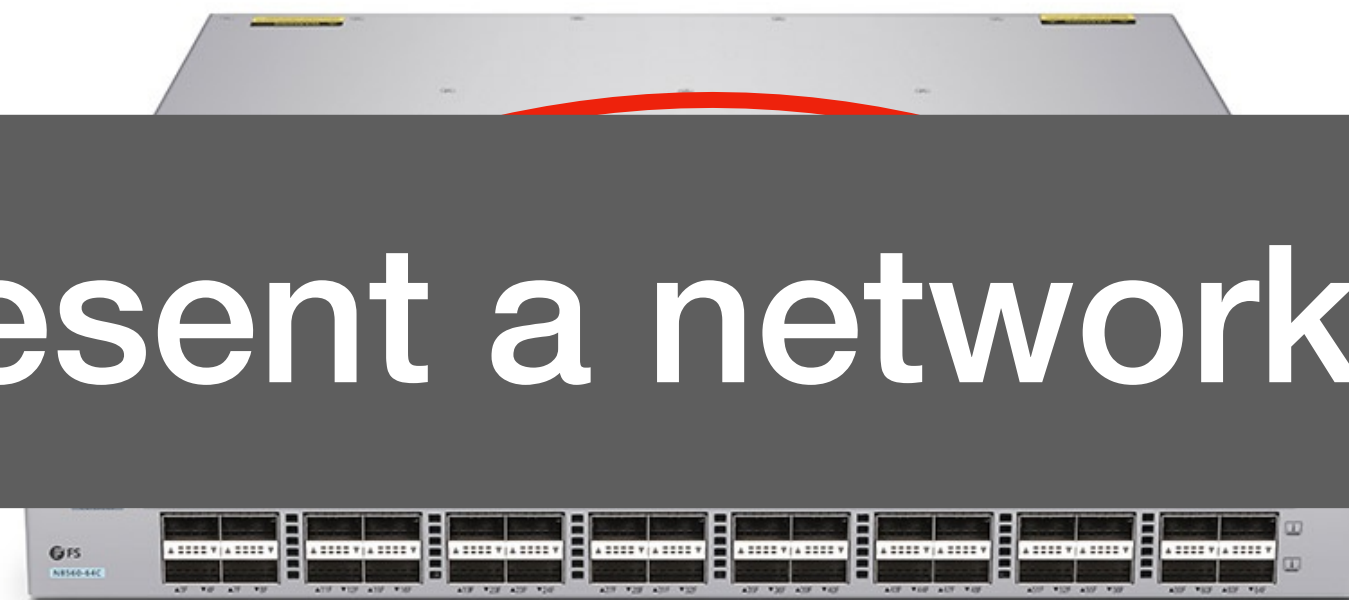
- A unique identifier in the data center networks: 5-Tuple
    - Source IP
    - Destination IP
    - Source Port
    - Destination Port
    - Protocol Number
- } Host-Host communication channel
- } App-App communication channel



# Network Flow

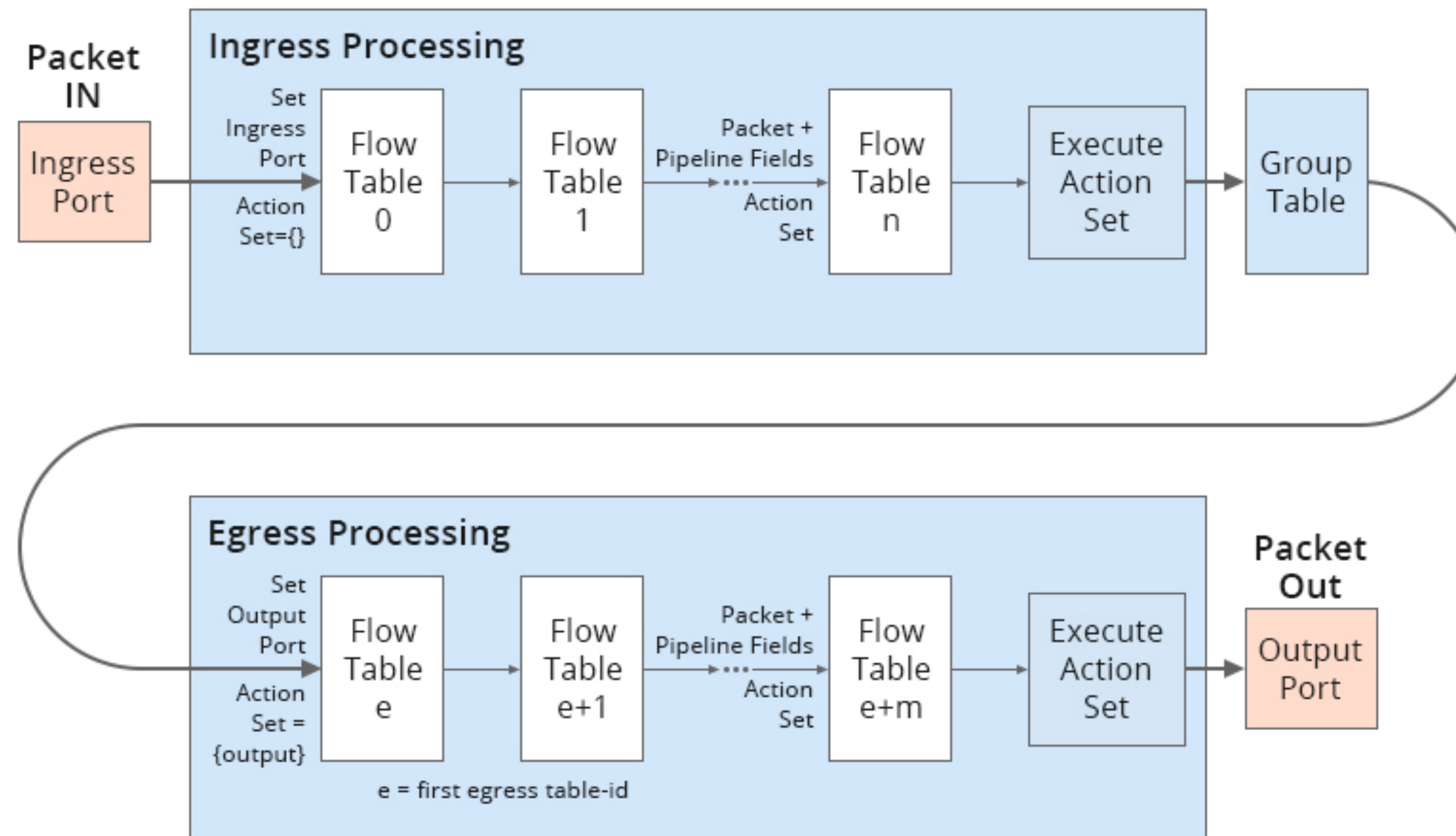
- A unique identifier in the data center networks: 5-Tuple
    - Source IP
    - Destination IP
    - Source Port
    - Destination Port
    - Protocol Number
- } Host-Host communication channel
- } App-App communication channel

How to represent a network flow in a switch?



# Some Switch Internals

- Switch ASIC: per-packet processing under high bandwidth!
  - Ingress/egress pipeline: table lookup, header modifications
  - Traffic manager: packet scheduling, rate limiting, and mirroring



# Query 5-Tuple is costly!

- High memory footprint and compute cycles
  - $13\text{B} = \text{src\_ip}(4\text{B}) + \text{dst\_ip}(4\text{B}) + \text{src\_port}(2\text{B}) + \text{dst\_port}(2\text{B}) + \text{procol\_num}(1\text{B})$
  - 3 SRAM lookups

# Query 5-Tuple is costly!

- High memory footprint and compute cycles
  - $13\text{B} = \text{src\_ip}(4\text{B}) + \text{dst\_ip}(4\text{B}) + \text{src\_port}(2\text{B}) + \text{dst\_port}(2\text{B}) + \text{procol\_num}(1\text{B})$
  - 3 SRAM lookups

Let's do hashing!

# Network Flow Representation in Commodity Switches

- $\text{FlowID} = \text{Hash}(\text{src\_ip}, \text{dst\_ip}, \text{src\_port}, \text{dst\_port}, \text{proto\_num})$ 
  - The hashing algorithm is mostly proprietary
  - Perform at the switch ingress side before moving to the pipeline
  - FlowID becomes a N-bit metadata field (N=16, for example)

# Network Flow Representation in Commodity Switches

- FlowID=Hash(src\_ip, dst\_ip, src\_port, dst\_port, proto\_num)
  - The hashing algorithm is mostly proprietary
  - Perform at the switch ingress side before moving to the pipeline
  - FlowID becomes a N-bit metadata field (N=16, for example)

Is FlowID unique?

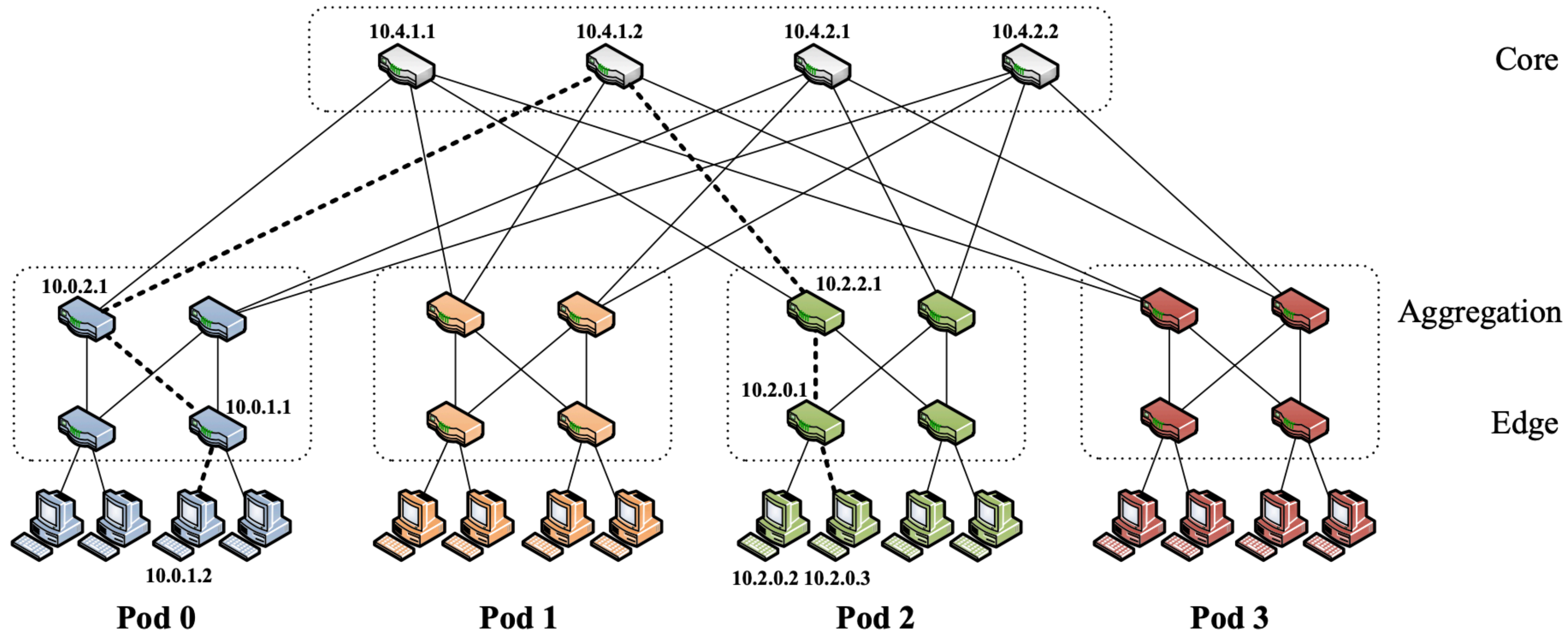
# Network Flow Representation in Commodity Switches

- FlowID=Hash(src\_ip, dst\_ip, src\_port, dst\_port, proto\_num)
  - The hashing algorithm is mostly proprietary
  - Perform at the switch ingress side before moving to the pipeline
  - FlowID becomes a N-bit metadata field (N=16, for example)

**Is FlowID unique?**

**No, because of hash collision**

# Which path should a network flow take?



# Approach #1: ECMP

- Equal-Cost Multi-Path Forwarding
  - $\text{chosen\_path} = \text{FlowID} \bmod (\text{path \#})$

# Approach #1: ECMP

- Equal-Cost Multi-Path Forwarding
  - $\text{chosen\_path} = \text{FlowID} \bmod (\text{path \#})$
- Cost = The number of hops
- Simple and it works!

# Approach #1: ECMP

- Equal-Cost Multi-Path Forwarding
  - $\text{chosen\_path} = \text{FlowID} \bmod (\text{path \#})$
- Cost = The number of hops
- Simple and it works!

Is this aligned with our discussion last week?

# Approach #1: ECMP

- Equal-Cost Multi-Path Forwarding
  - $\text{chosen\_path} = \text{FlowID} \bmod (\text{path \#})$
- Cost = The number of hops
- Simple and it works!

**Is this aligned with our discussion last week?**  
**No. In the scalable DCNet, the host ID determines the routing path!**

# ECMP More

- Why ECMP has become ubiquitous
  - Stateless in the network
  - Extremely cheap in switch ASICs
  - No global traffic knowledge required
  - Preserves packet ordering with a flow
  - Works surprising well when there are many independent flows

# ECMP More

- Why ECMP has become ubiquitous
  - Stateless in the network
  - Extremely cheap in switch ASICs
  - No global traffic knowledge required
  - Preserves packet ordering with a flow
  - Works surprising well when there are many independent flows

**Hidden assumption: ECMP assumes enough flow entropy to statistically balance traffic!**

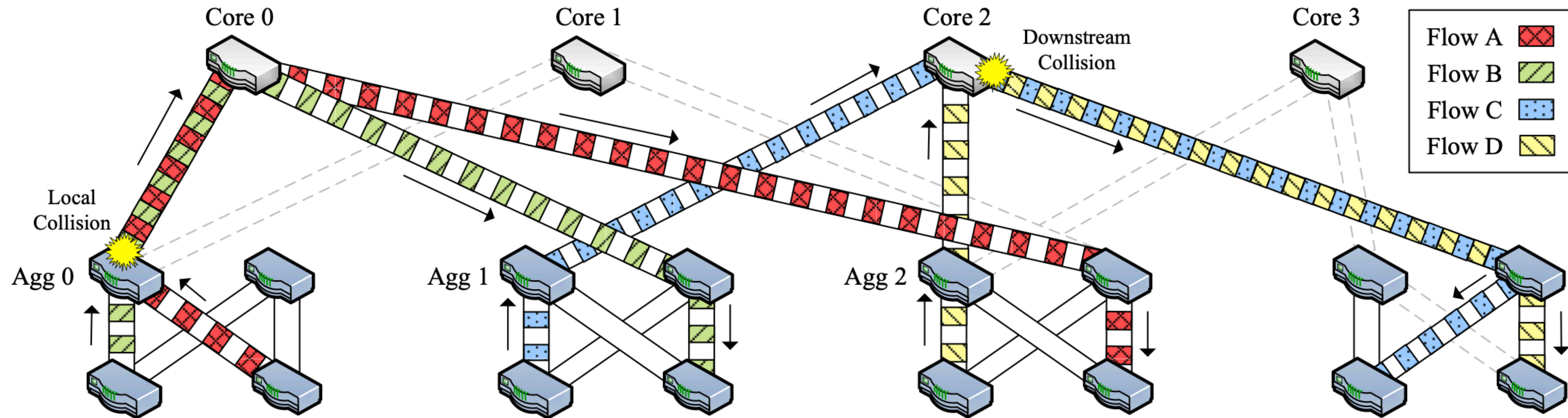
# ECMP More

- Why ECMP has become ubiquitous
  - Stateless in the network
  - Extremely cheap in switch ASICs
  - No global traffic knowledge required
  - Preserves packet ordering with a flow
  - Works surprising well when there are many independent flows

**What are the issues of ECMP?**

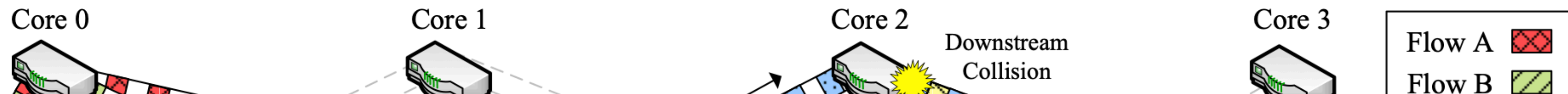
# Path Sharing

- If incoming bandwidth  $\leq$  outgoing bandwidth

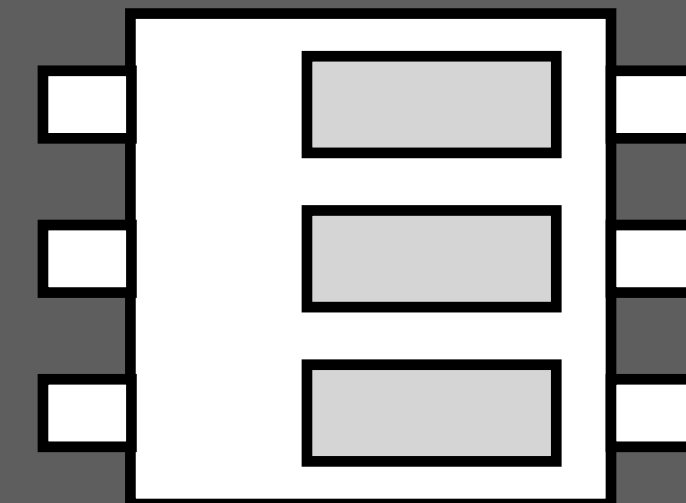


# Path Sharing

- If incoming bandwidth  $\leq$  outgoing bandwidth

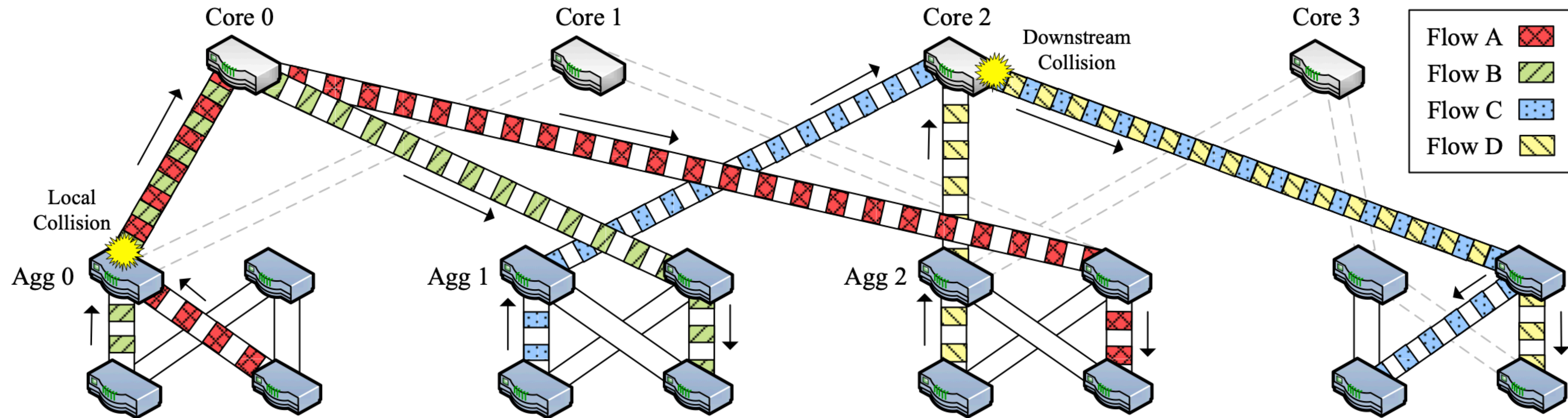


Think about the switch architecture



# Path Sharing

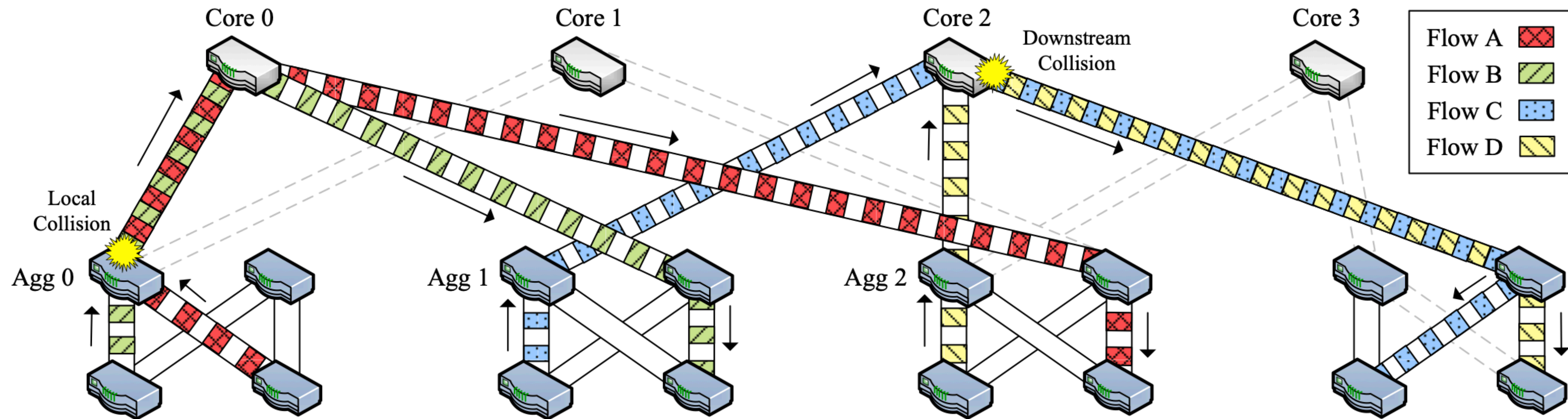
- If incoming bandwidth  $\leq$  outgoing bandwidth **Okay!**



# Path Sharing

- If incoming bandwidth  $\leq$  outgoing bandwidth
- If incoming bandwidth  $>$  outgoing bandwidth

Okay!

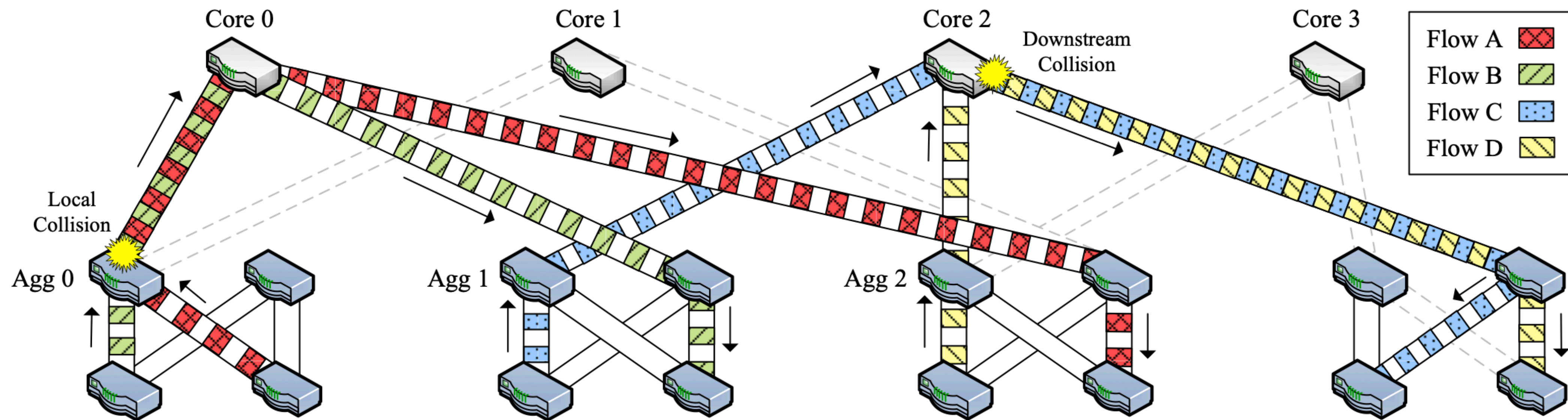


# Path Sharing

- If incoming bandwidth  $\leq$  outgoing bandwidth
- If incoming bandwidth  $>$  outgoing bandwidth

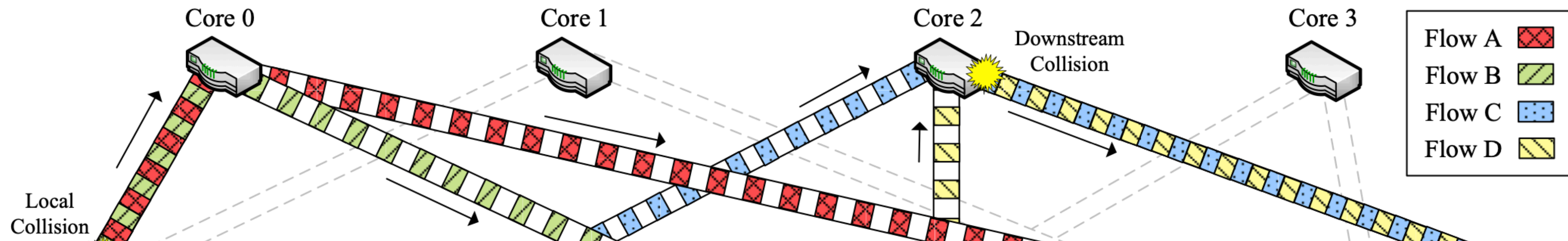
Okay!

Bad!



# Path Sharing

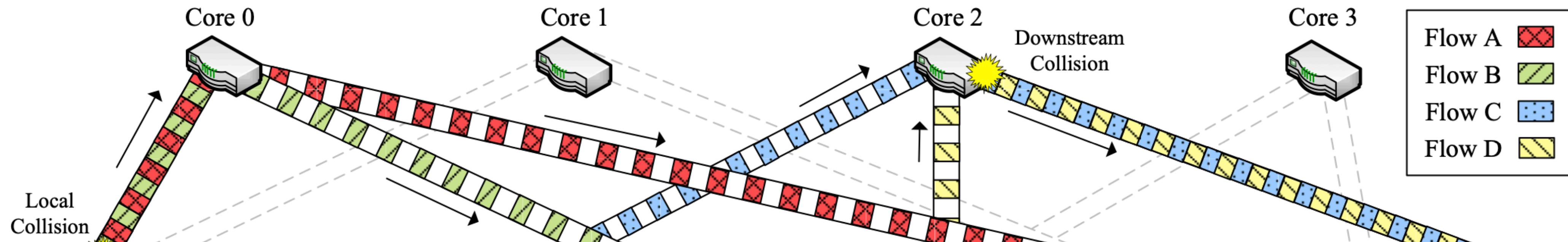
- If incoming bandwidth  $\leq$  outgoing bandwidth **Okay!**
- If incoming bandwidth  $>$  outgoing bandwidth **Bad!**



How can we solve it?

# Path Sharing

- If incoming bandwidth  $\leq$  outgoing bandwidth **Okay!**
- If incoming bandwidth  $>$  outgoing bandwidth **Bad!**



How can we solve it?  
Mitigate bandwidth oversubscription by  
leveraging more paths!

# Problem Formulation

- Multi-commodity flow problem
  - NP-complete for integer flows
- Approximation hint
  - Match flow **demand** and link bandwidth **supply**

# Problem Formulation

- Multi-commodity flow problem
  - NP-complete for integer flows
- Approximation hint
  - Match flow **demand** and link bandwidth **supply**

- How can we know the flow demand?
- How can we know the bandwidth supply?
- Which path should a flow choose?

## Proposal #2: Hedra (NSDI'10)

- Let's start with a simple example
- Network-limited flows

An example:

- H0 sends 1 flow to H1, H2, and H3
- H1 sends 2 flows to H0 and 1 flow to H2
- H2 sends 1 flow each to H0 and H3
- H3 sends 2 flows to H1

# Demand Estimation Algorithm

- Traffic matrix:  $M \times N$ 
  - $M$ : the number of senders
  - $N$ : the number of receivers
- Each element in the matrix
  - The number of flows from host  $i$  to host  $j$
  - The estimated demand of each of the flows from host  $i$  to host  $j$
  - A converged flag that marks flows whose demands have converged

# Algorithm Walkthrough

- dF-> The “converged” demand
- dU-> The number of unconverged flows
- eS-> The computed equal share rate
- dT-> The total demand
- dS-> The sender limited demand
- f.rl-> A flag for a receiver limited flow
- nR-> The number of receiver limited flow

```
ESTIMATE-DEMANDS()  
1  for all  $i, j$   
2     $M_{i,j} \leftarrow 0$   
3  do  
4    foreach  $h \in H$  do EST-SRC( $h$ )  
5    foreach  $h \in H$  do EST-DST( $h$ )  
6  while some  $M_{i,j}$ .demand changed  
7  return  $M$ 
```

# Algorithm Walkthrough

- dF-> The “converged” demand
- dU-> The number of unconverged flows
- eS-> The computed equal share rate
- dT-> The total demand
- dS-> The sender limited demand
- f.rl-> A flag for a receiver limited flow
- nR-> The number of receiver limited flow

```
EST-SRC(src: host)
1   $d_F \leftarrow 0$ 
2   $n_U \leftarrow 0$ 
3  foreach  $f \in \langle \text{src} \rightarrow \text{dst} \rangle$  do
4      if  $f.\text{converged}$  then
5           $d_F \leftarrow d_F + f.\text{demand}$ 
6      else
7           $n_U \leftarrow n_U + 1$ 
8   $e_S \leftarrow \frac{1.0 - d_F}{n_U}$ 
9  foreach  $f \in \langle \text{src} \rightarrow \text{dst} \rangle$  and not  $f.\text{converged}$  do
10      $M_{f.\text{src}, f.\text{dst}}.\text{demand} \leftarrow e_S$ 
```

```
ESTIMATE-DEMANDS()
1  for all  $i, j$ 
2       $M_{i,j} \leftarrow 0$ 
3  do
4      foreach  $h \in H$  do EST-SRC( $h$ )
5      foreach  $h \in H$  do EST-DST( $h$ )
6  while some  $M_{i,j}.\text{demand}$  changed
7  return  $M$ 
```

# Algorithm Walkthrough

- dF-> The “converged” demand
- dU-> The number of unconverged flows
- eS-> The computed equal share rate
- dT-> The total demand
- dS-> The sender limited demand
- f.rl-> A flag for a receiver limited flow
- nR-> The number of receiver limited flow

```

ESTIMATE-DEMANDS()
1  for all  $i, j$ 
2     $M_{i,j} \leftarrow 0$ 
3  do
4    foreach  $h \in H$  do EST-SRC( $h$ )
5    foreach  $h \in H$  do EST-DST( $h$ )
6  while some  $M_{i,j}$ .demand changed
7  return  $M$ 
    
```

```

EST-SRC(src: host)
1   $d_F \leftarrow 0$ 
2   $n_U \leftarrow 0$ 
3  foreach  $f \in \langle \text{src} \rightarrow \text{dst} \rangle$  do
4    if  $f$ .converged then
5       $d_F \leftarrow d_F + f$ .demand
6    else
7       $n_U \leftarrow n_U + 1$ 
8   $e_S \leftarrow \frac{1.0 - d_F}{n_U}$ 
9  foreach  $f \in \langle \text{src} \rightarrow \text{dst} \rangle$  and not  $f$ .converged do
10     $M_{f.\text{src}, f.\text{dst}}.\text{demand} \leftarrow e_S$ 
    
```

```

EST-DST(dst: host)
1   $d_T, d_S, n_R \leftarrow 0$ 
2  foreach  $f \in \langle \text{src} \rightarrow \text{dst} \rangle$ 
3     $f$ .rl  $\leftarrow$  true
4     $d_T \leftarrow d_T + f$ .demand
5     $n_R \leftarrow n_R + 1$ 
6  if  $d_T \leq 1.0$  then
7    return
8   $e_S \leftarrow \frac{1.0}{n_R}$ 
9  do
10     $n_R \leftarrow 0$ 
11    foreach  $f \in \langle \text{src} \rightarrow \text{dst} \rangle$  and  $f$ .rl do
12      if  $f$ .demand  $< e_S$  then
13         $d_S \leftarrow d_S + f$ .demand
14         $f$ .rl  $\leftarrow$  false
15      else
16         $n_R \leftarrow n_R + 1$ 
17     $e_S \leftarrow \frac{1.0 - d_S}{n_R}$ 
18  while some  $f$ .rl was set to false
19  foreach  $f \in \langle \text{src} \rightarrow \text{dst} \rangle$  and  $f$ .rl do
20     $M_{f.\text{src}, f.\text{dst}}.\text{demand} \leftarrow e_S$ 
21     $M_{f.\text{src}, f.\text{dst}}.\text{converged} \leftarrow$  true
    
```

# Algorithm Walkthrough

- dF-> The “converged” demand
- dU-> The number of unconverged flows
- eS-> The computed equal share rate
- dT-> The total demand

```

EST-SRC(src: host)
1  d_F ← 0
2  n_U ← 0
3  foreach f ∈ ⟨src → dst⟩ do
4    if f.converged then
5      d_F ← d_F + f.demand
6    else
7      n_U ← n_U + 1

```

$$\begin{bmatrix} & (\frac{1}{3})_1 & (\frac{1}{3})_1 & (\frac{1}{3})_1 \\ (\frac{1}{3})_2 & & (\frac{1}{3})_1 & 0_0 \\ (\frac{1}{2})_1 & 0_0 & & (\frac{1}{2})_1 \\ 0_0 & (\frac{1}{2})_2 & 0_0 & \end{bmatrix} \Rightarrow \begin{bmatrix} & [\frac{1}{3}]_1 & (\frac{1}{3})_1 & (\frac{1}{3})_1 \\ [\frac{1}{3}]_2 & & (\frac{1}{3})_1 & 0_0 \\ [\frac{1}{3}]_1 & 0_0 & & (\frac{1}{2})_1 \\ 0_0 & [\frac{1}{3}]_2 & 0_0 & \end{bmatrix} \Rightarrow \begin{bmatrix} & [\frac{1}{3}]_1 & (\frac{1}{3})_1 & (\frac{1}{3})_1 \\ [\frac{1}{3}]_2 & & (\frac{1}{3})_1 & 0_0 \\ [\frac{1}{3}]_1 & 0_0 & & (\frac{2}{3})_1 \\ 0_0 & [\frac{1}{3}]_2 & 0_0 & \end{bmatrix} \Rightarrow \begin{bmatrix} & [\frac{1}{3}]_1 & (\frac{1}{3})_1 & [\frac{1}{3}]_1 \\ [\frac{1}{3}]_2 & & (\frac{1}{3})_1 & 0_0 \\ [\frac{1}{3}]_1 & 0_0 & & [\frac{2}{3}]_1 \\ 0_0 & [\frac{1}{3}]_2 & 0_0 & \end{bmatrix}$$

```

ESTIMATE-DEMANDS()
1  for all i, j
2    M_{i,j} ← 0
3  do
4    foreach h ∈ H do EST-SRC(h)
5    foreach h ∈ H do EST-DST(h)
6  while some M_{i,j}.demand changed
7  return M

```

```

4  d_T ← d_T + f.demand
5  n_R ← n_R + 1
6  if d_T ≤ 1.0 then
7    return
8  e_S ← \frac{1.0}{n_R}
9  do
10   n_R ← 0
11   foreach f ∈ ⟨src → dst⟩ and f.rl do
12     if f.demand < e_S then
13       d_S ← d_S + f.demand
14       f.rl ← false
15     else
16       n_R ← n_R + 1
17   e_S ← \frac{1.0 - d_S}{n_R}
18   while some f.rl was set to false
19   foreach f ∈ ⟨src → dst⟩ and f.rl do
20     M_{f.src, f.dst}.demand ← e_S
21     M_{f.src, f.dst}.converged ← true

```

# Algorithm Walkthrough

- dF-> The “converged” demand
- dU-> The number of unconverged flows
- eS-> The computed equal share rate
- dT-> The total demand
- dS-> The sender limited demand
- f.rl-> A flag for a receiver limited flow
- nR-> The number of receiver limited flow

```
EST-SRC(src: host)
1  d_F ← 0
2  n_U ← 0
3  foreach f ∈ ⟨src → dst⟩ do
4    if f.converged then
5      d_F ← d_F + f.demand
6    else
7      n_U ← n_U + 1
8  e_S ←  $\frac{1.0 - d_F}{n_U}$ 
9  foreach f ∈ ⟨src → dst⟩ and not f.converged do
10    M_{f.src, f.dst}.demand ← e_S
```

```
EST-DST(dst: host)
1  d_T, d_S, n_R ← 0
```

- SRC host: max out the bandwidth supply
- DST host: restrict the bandwidth demand
- Maxmin fairness at a global scale

```
18 while some f.rl was set to false
19 foreach f ∈ ⟨src → dst⟩ and f.rl do
20   M_{f.src, f.dst}.demand ← e_S
21   M_{f.src, f.dst}.converged ← true
```

# Path Selection

- Global First Fit
  - Search all possible paths and choose the first fit one
- Simulated Annealing
  - Choose the one that satisfies the energy function
  - Pruning the search space is key
  - Assign a single core switch for each destination host

# Hedra in Practice

- #1: Central scheduler
  - Generate the forwarding table
  - Distribute to all switches
- #2: Flow demand estimator
  - Max-min fairness based on the traffic matrix
- #3: Approximate path selection
  - Employ heuristic algorithms

# Hedra in Practice

- #1: Central scheduler
  - Generate the forwarding table
  - Distribute to all switches
- #2: Flow demand estimator
  - Max-min fairness based on the traffic matrix
- #3: Approximate path selection
  - Employ heuristic algorithms

**Does Hedra always work?**

# Challenges in Deployment

- #1: Networking bandwidth capacity (supply) is ephemeral!
- #2: Flow demand and flow # are unpredictable!
  - Hedra targets long-lived flows
- #3: Real-time global view is hard to maintain!

**Fast Forward 15 Years: GPUs bring the problem back!**

# ECMP Performs Poorly in AI Training

- Elephant RDMA flows break ECMP at vastly larger bandwidths
  - Low entropy, burstiness, and elephant flows

## RDMA over Ethernet for Distributed AI Training at Meta Scale

Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, Hongyi Zeng  
Meta

### ABSTRACT

The rapid growth in both computational density and scale in AI models in recent years motivates the construction of an efficient and reliable dedicated network infrastructure. This paper presents the design, implementation, and operation of Meta’s Remote Direct Memory Access over Converged Ethernet (RoCE) networks for distributed AI training.

Our design principles involve a deep understanding of the workloads, and we translated these insights into the design of various network components: **Network Topology** - To support the rapid evolution of generations of AI hardware platforms, we separated GPU-based training into its own “backend” network. **Routing** - Training workloads inherently impose load imbalance and burstiness, so we deployed several iterations of routing schemes to achieve near-optimal traffic distribution. **Transport** - We outline how we initially attempted to use DCQCN for congestion management but then pivoted away from DCQCN to instead leverage the collective library itself to manage congestion. **Operations** - We share our experience operating large-scale AI networks, including toolings we developed and troubleshooting examples.

### CCS CONCEPTS

• **Networks** → **Network architectures**; **Data center networks**; **Network protocols**; **Traffic engineering algorithms**.

### KEYWORDS

RDMA, Distributed Training

#### ACM Reference Format:

Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, Hongyi Zeng. 2024. RDMA over Ethernet for Distributed AI Training at Meta Scale. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM ’24)*, August 4–8, 2024, Sydney, NSW, Australia. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3651890.3672233>

Distributed training, in particular, imposes the most significant strain on data center networking infrastructure. For instance, a typical generative AI job may necessitate tight coordination of thousands of GPUs over the course of several weeks. Constructing a reliable, high-performance network infrastructure capable of accommodating this burgeoning demand necessitates a reevaluation of data center network design.

The inter-GPU communication in distributed training workloads typically comprises two stages. In each training node, housing 4-8 GPUs, these GPUs are interconnected using high-speed transports like NVLink[26], commonly known as “intra-node communication.” When a training job requires additional GPUs, the “inter-node communication” will happen across a network. The industry commonly employs two design approaches for this inter-node communication. One utilizes standard TCP/IP or modified socket implementations (e.g., fastsocket[7]). However, this method is susceptible to performance degradation because of CPU overhead and increased latency. The second design approach involves proprietary interconnects, such as InfiniBand[23], NVSwitch[26], Elastic Fabric Adaptor[2], and Inter-rack ICI[8]. Although these methods deliver significantly improved performance, their proprietary nature restricts their deployment flexibility.

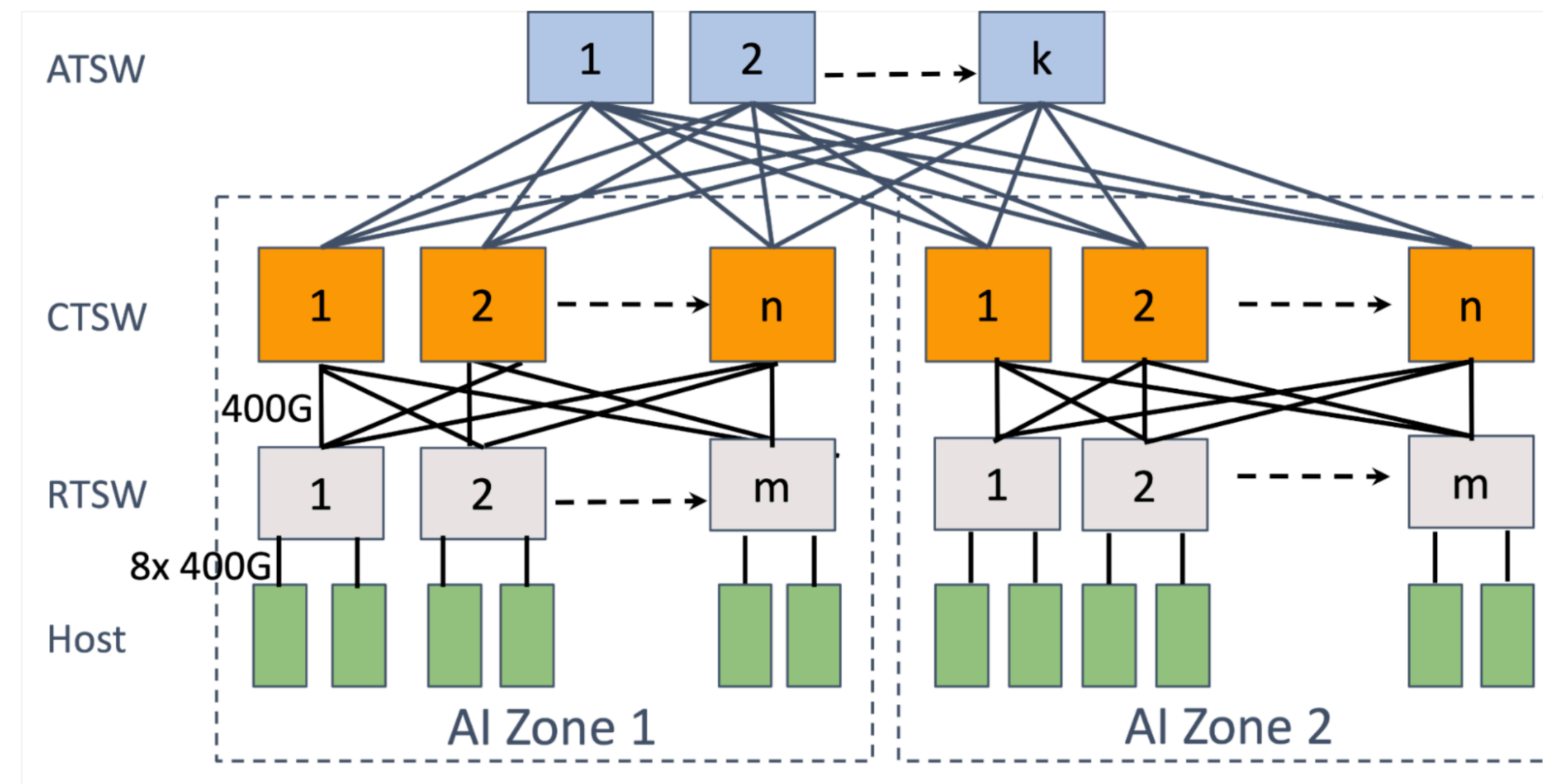
When Meta introduced distributed, GPU-based training, we decided to construct specialized data center networks tailored for these GPU clusters. We opted for RoCE[11] as the inter-node communication transport. Remote Direct Memory Access (RDMA) is a method to access the remote memory without involving the CPU. The choice of RoCE was motivated by:

1) RoCE adheres to the established RDMA verbs semantics, well-known to the training workload community. This ensured a seamless transition for existing training applications and facilitated the development of new ones.

2) By utilizing Ethernet, we could employ a significant portion of our existing data center design components and tools. This enabled us to build the network using a largely consistent Clos-based design,

# Why?

- AI training traffic
  - Tens of thousands of GPUs participate in synchronized communications.
  - RoCEv2 carries RDMA traffic over UDP.
  - Individual bursts can approach the NIC line rate.
  - Traffic patterns repeat with collective operations.
  - A few elephant flows can dominate links.



# Meta Tried Path Pinning

- Route traffic based on destination slices
- Work well under ideal placement

# Meta Tried Path Pinning

- Route traffic based on destination slices
- Work well under ideal placement
  
- But, fragmented GPU job placement/failures cause imbalance!
- More than 30% training degradation!

# Meta Tried Path Pinning

- Route traffic based on destination slices
- Work well under ideal placement
- But, fragmented GPU job placement/failures cause imbalance!
- More than 30% training degradation!

Can we use Herdra to solve it?

# Meta Tried Path Pinning

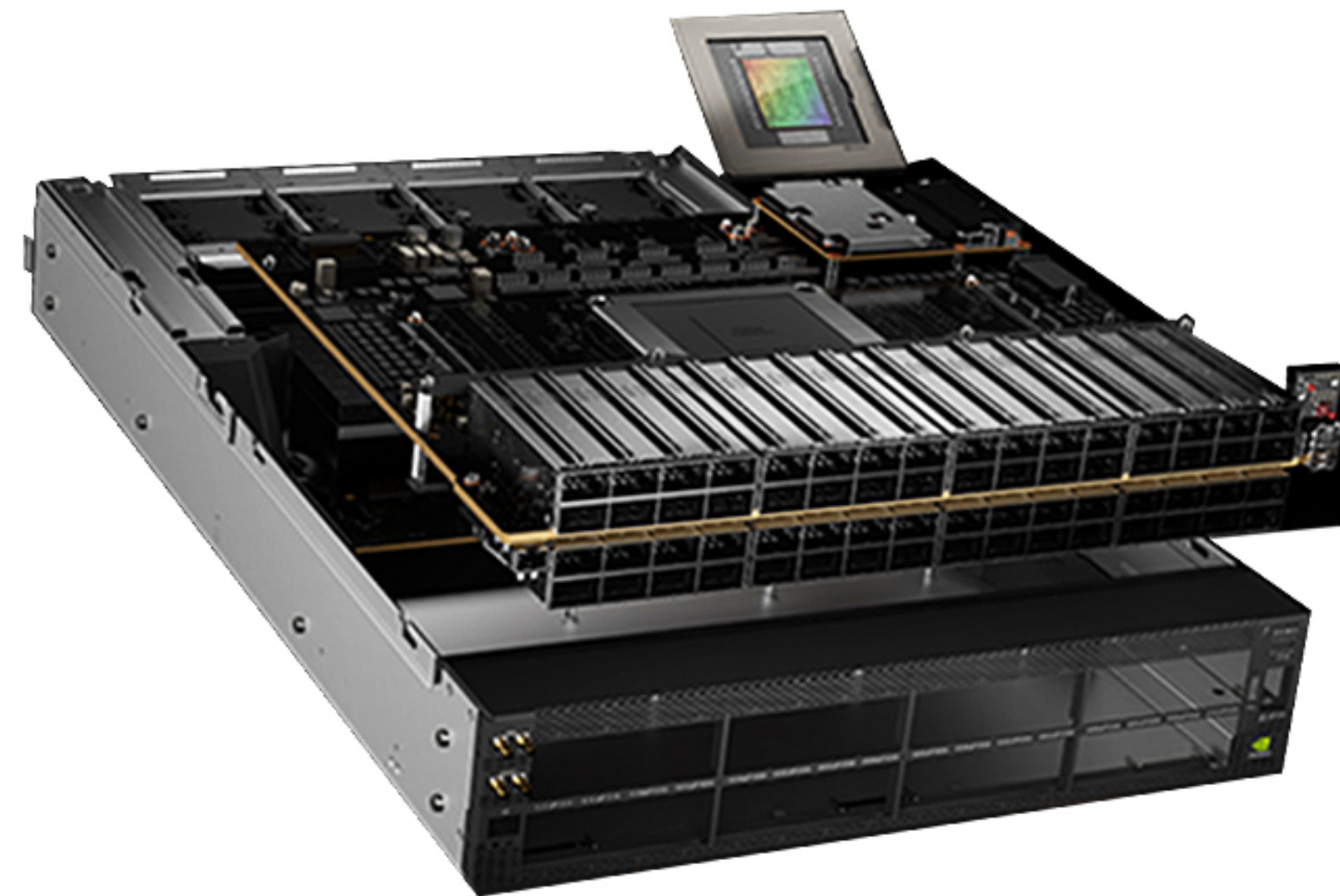
- Route traffic based on destination slices
- Work well under ideal placement
- But, fragmented GPU job placement/failures cause imbalance!
- More than 30% training degradation!

Can we use Herdra to solve it?

Hard, because bandwidth is ephemeral, demand is unpredictable, and maintaining a global view is difficult.

# Spectrum-X: Adaptive Routing

- Choose paths based on the current **network state**
- Path = F(Queue occupancy, link utilization, congestion)
  - Continuously observe network conditions
  - Dynamically select a less-congested path
  - Operate at per-packet granularity
  - Better utilize multiple paths and avoid ECMP collisions



# Summary

- Today
  - Flow scheduling in data center networks (I)
- Next
  - Flow scheduling in data center networks (II)
  - Conga (SIGCOMM'14)