Advanced Computer Networks

Flow Scheduling in Data Center Networks (I)

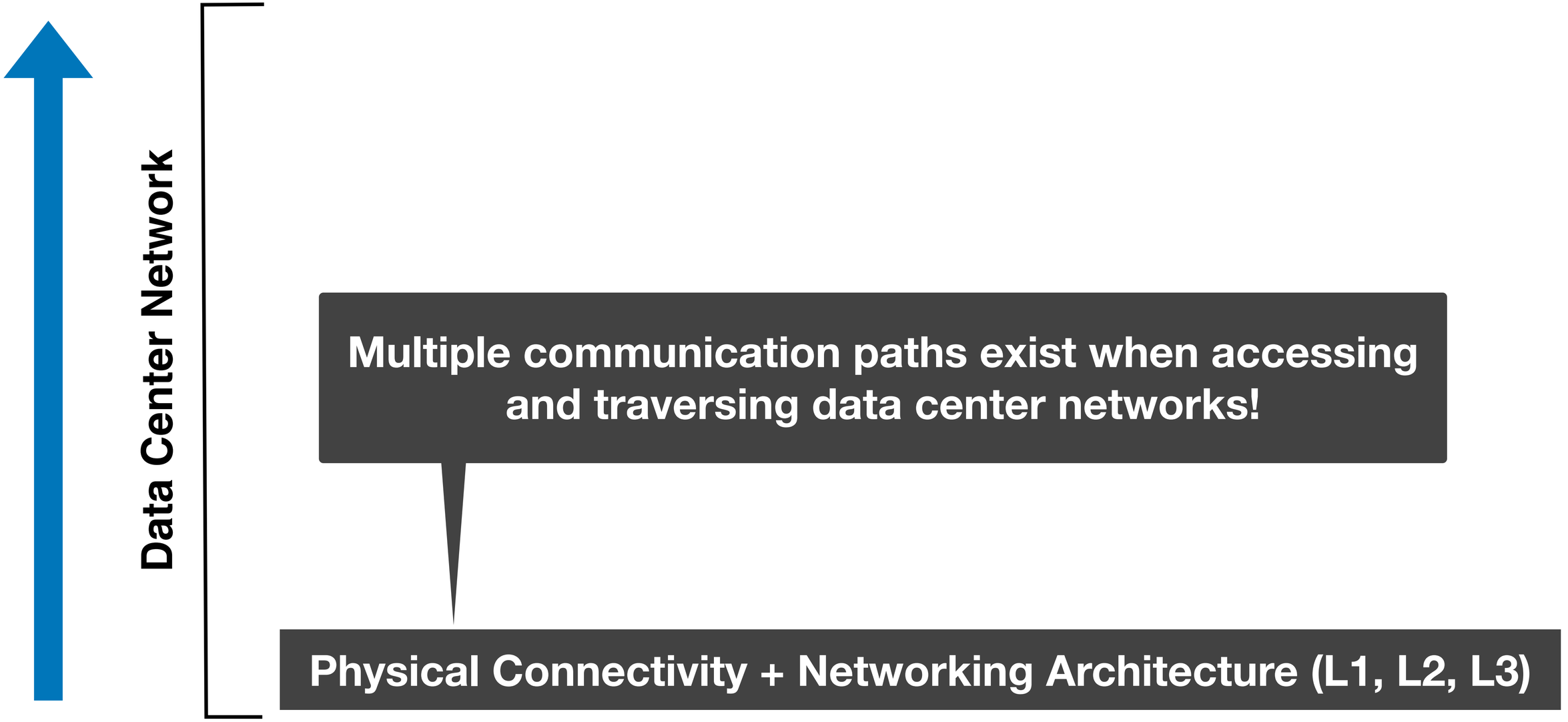
https://pages.cs.wisc.edu/~mgliu/CS740/F25/index.html

Ming Liu
mgliu@cs.wisc.edu

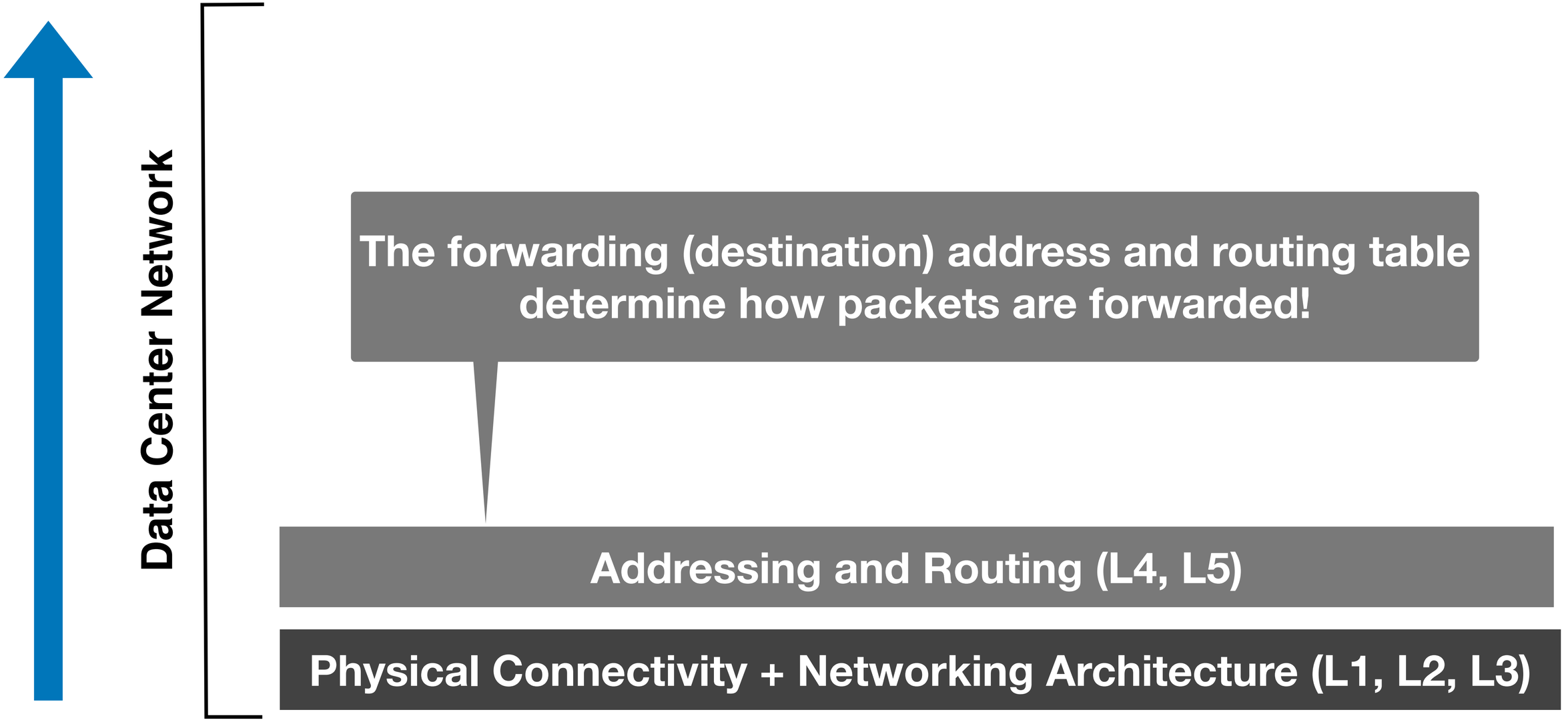
Outline

- Last lecture
  - Addressing and routing in data center networks (II)
- Today
  - Flow scheduling in data center networks (I)
- Announcements
  - Project proposal due 10/02/2025 11:59 PM
  - Lab1 due 10/08/2025 11:59 PM

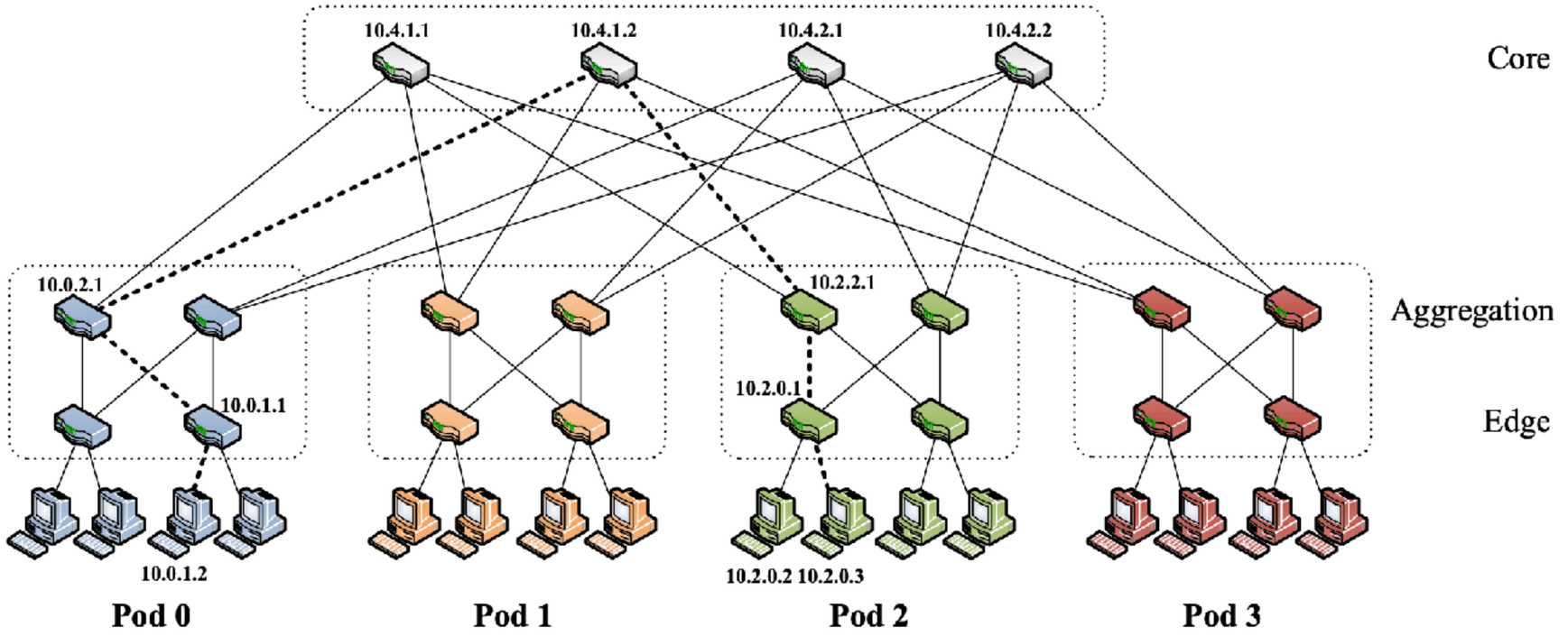
Where we are?


Data Center Network
Multiple communication paths exist when accessing and traversing data center networks!
Physical Connectivity + Networking Architecture (L1, L2, L3)

Where we are?

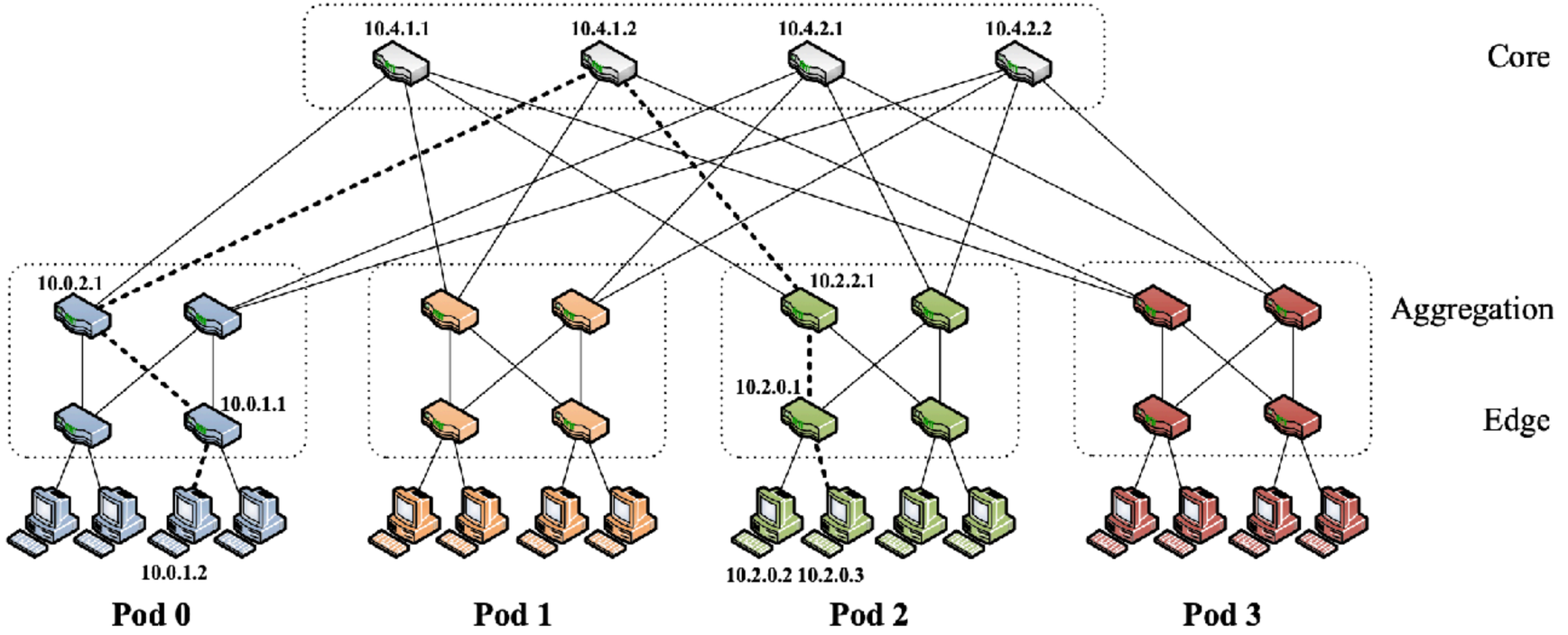

Data Center Network
The forwarding (destination) address and routing table determine how packets are forwarded!
Addressing and Routing (L4, L5)
Physical Connectivity + Networking Architecture (L1, L2, L3)

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

  } Host-Host communication channel

  - Source Port
  - Destination Port
  - Protocol Number

  } App-App communication channel

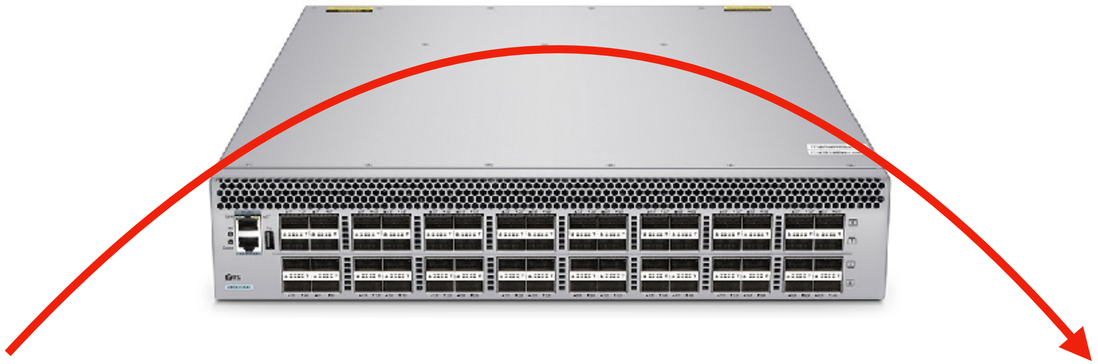
Network Flow

- A unique identifier in the data center networks: 5-Tuple
  - Source IP
  - Destination IP

  } Host-Host communication channel

  - Source Port
  - Destination Port
  - Protocol Number

  } App-App communication channel



Network Flow

- A unique identifier in the data center networks: 5-Tuple
  - Source IP
  - Destination IP

  } Host-Host communication channel

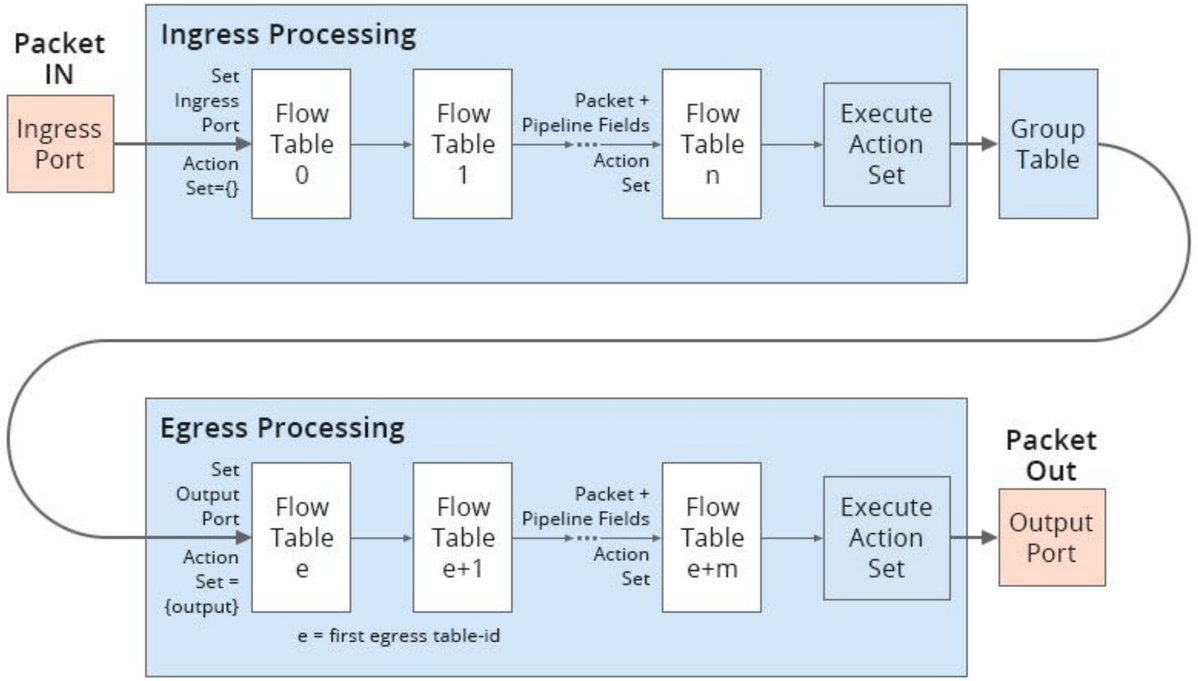
  - Source Port
  - Destination Port
  - Protocol Number

  } 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
  - 13B=src_ip(4B)+dst_ip(4B)+src_port(2B)+dst_port(2B)+procol_num(1B)
  - 3 SRAM lookups

Query 5-Tuple is costly!

- High memory footprint and compute cycles
  - 13B=src_ip(4B)+dst_ip(4B)+src_port(2B)+dst_port(2B)+procol_num(1B)
  - 3 SRAM lookups

Let’s do hashing!

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)

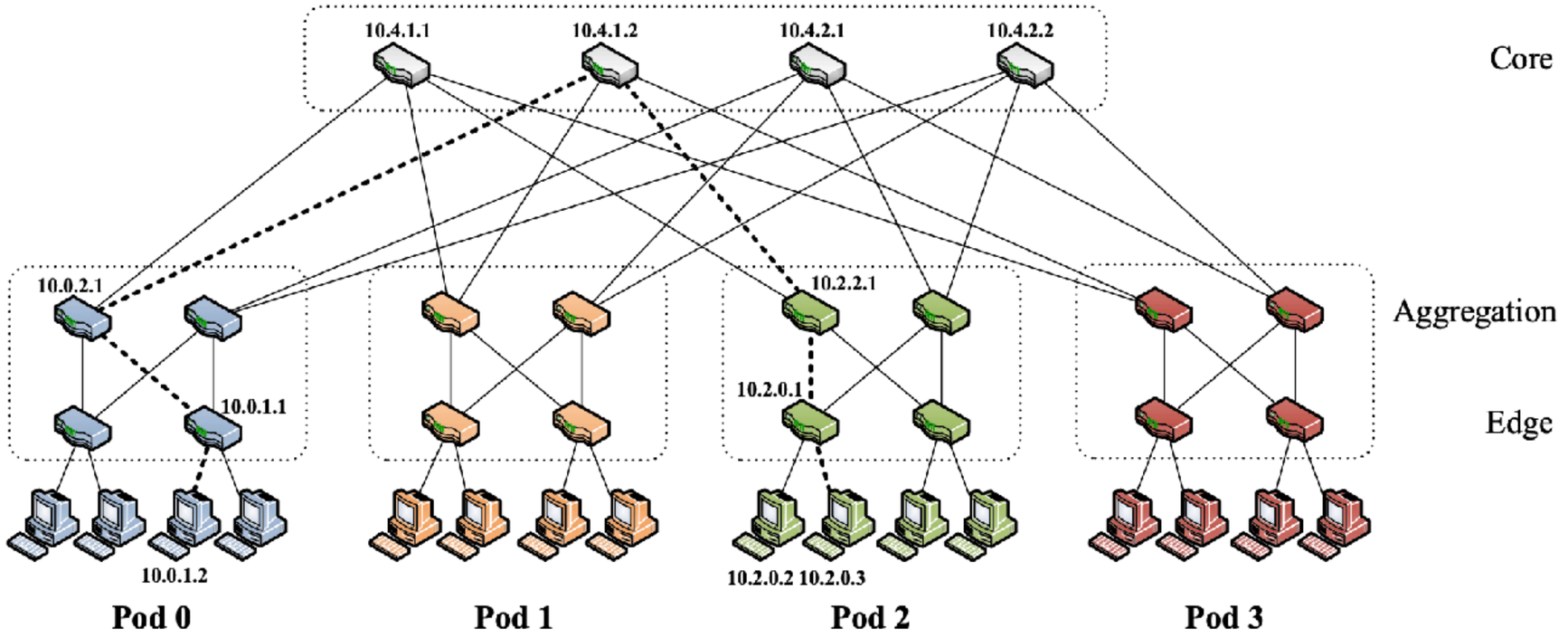
Is FlowID unique?

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 collosion

Which path should a network flow take?



Approach #1: ECMP

- Equal-Cost Multi-Path Fordwaring
  - chosen_path = FlowID mod (path #)

Approach #1: ECMP

- Equal-Cost Multi-Path Fordwaring
  - chosen_path = FlowID mod (path #)
- Cost = The number of hops
- Simple and it works!

Approach #1: ECMP

- Equal-Cost Multi-Path Fordwaring
  - chosen_path = FlowID mod (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 Fordwaring
  - chosen_path = FlowID mod (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!

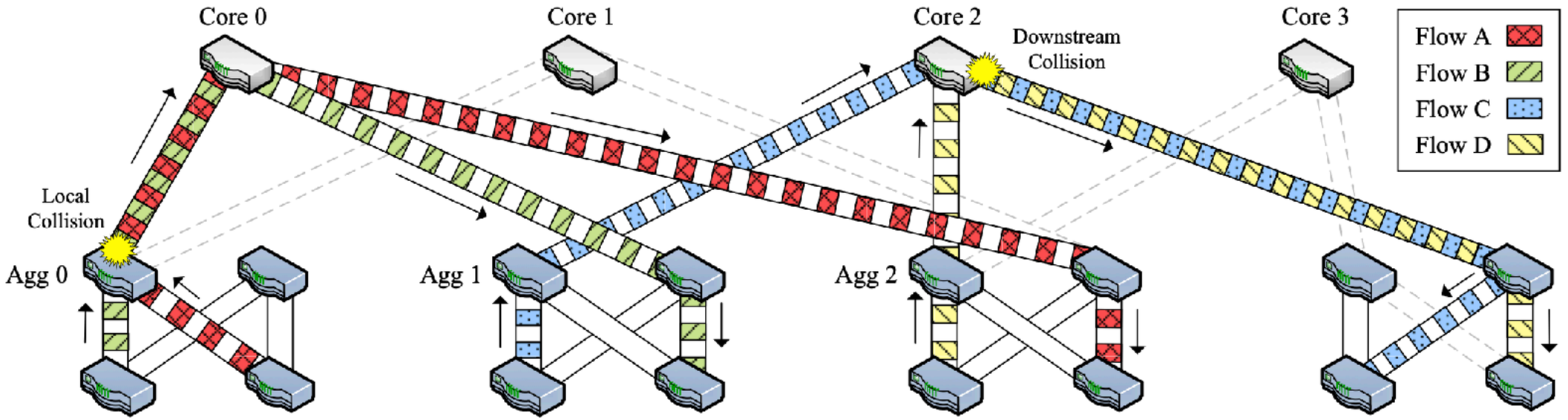
Approach #1: ECMP

- Equal-Cost Multi-Path Fordwaring
  - chosen_path = FlowID mod (path #)
- Cost = The number of hops
- Simple and it works!

What are the issues of ECMP?

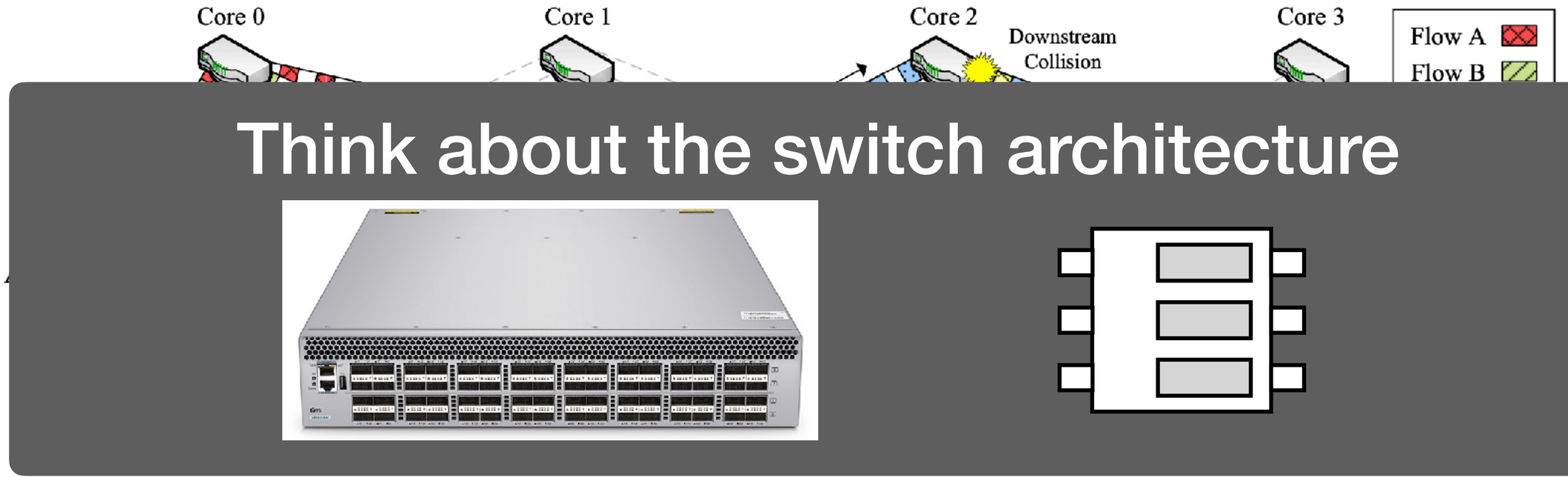
Path Sharing

- If incoming bandwidth <= outgoing bandwidth



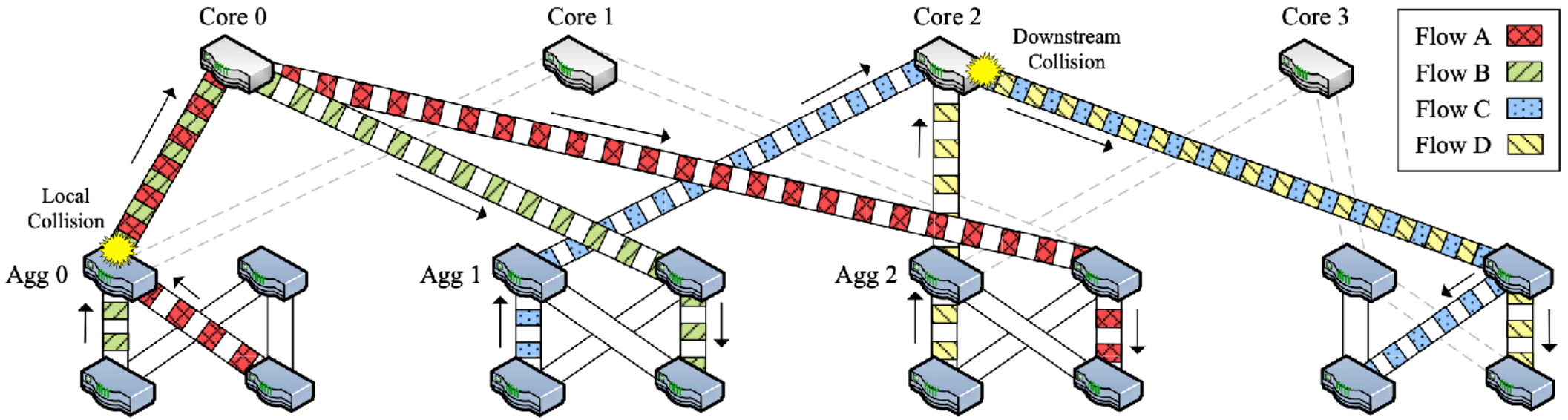
Path Sharing

- If incoming bandwidth <= outgoing bandwidth



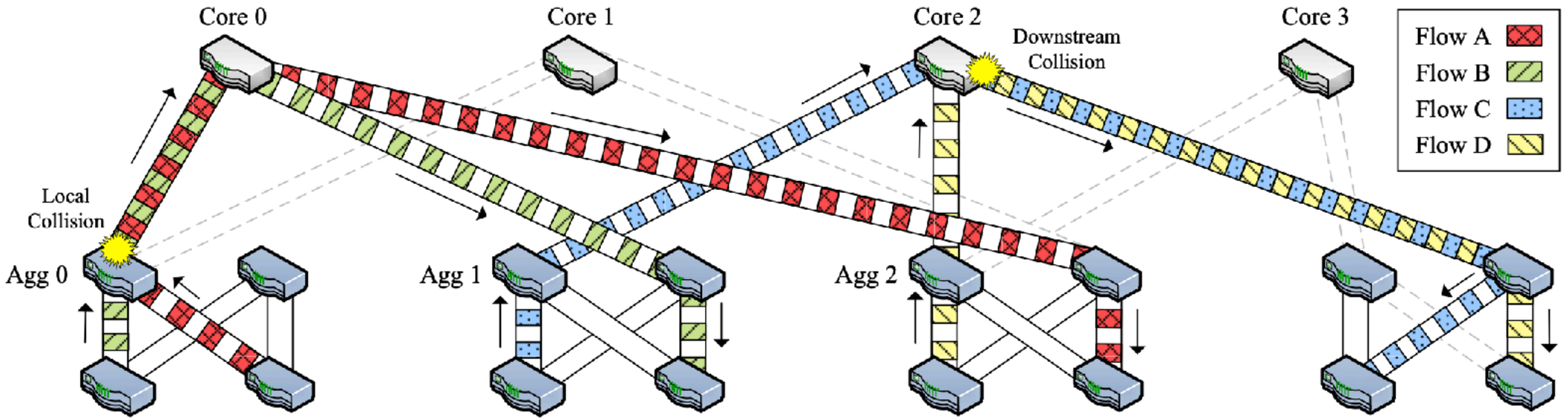
Path Sharing

- If incoming bandwidth <= outgoing bandwidth Okay!



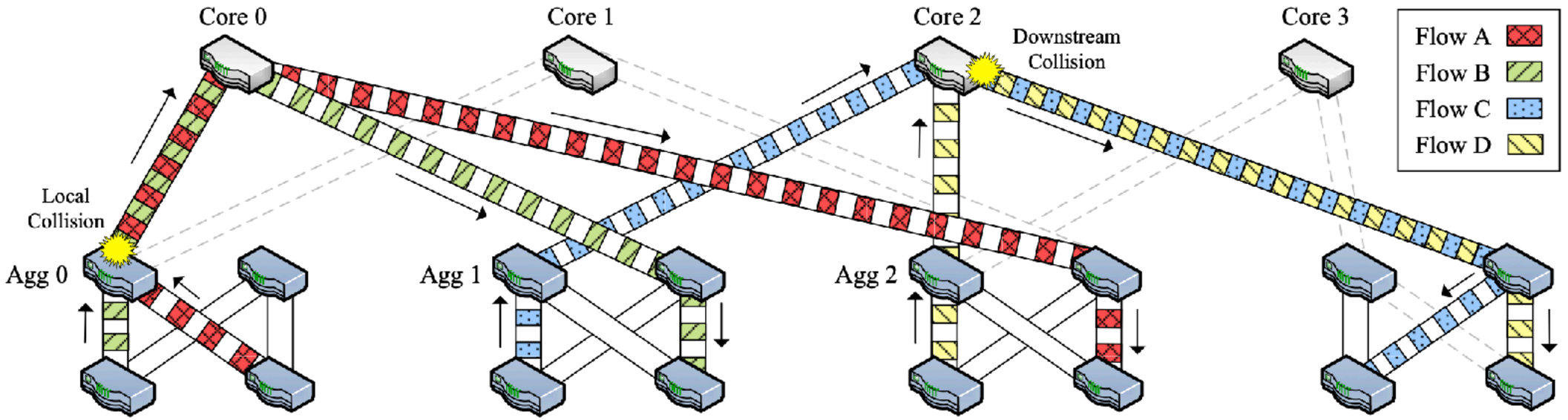
Path Sharing

- If incoming bandwidth <= outgoing bandwidth Okay!
- If incoming bandwidth > outgoing bandwidth



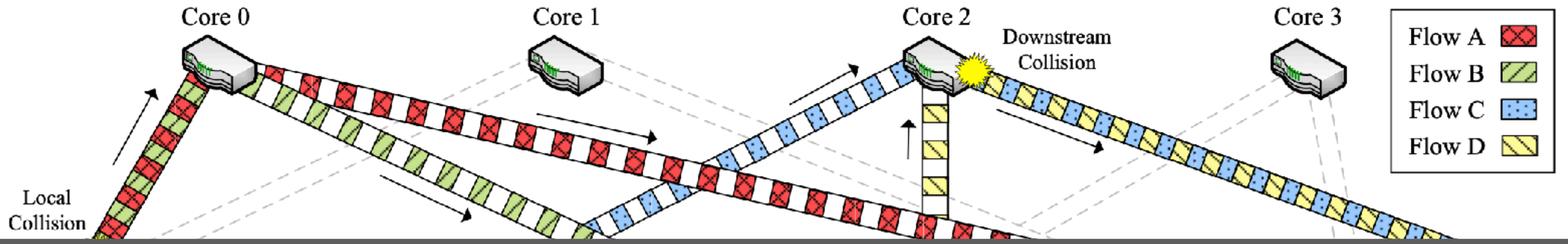
Path Sharing

- If incoming bandwidth <= outgoing bandwidth Okay!
- If incoming bandwidth > outgoing bandwidth Bad!



Path Sharing

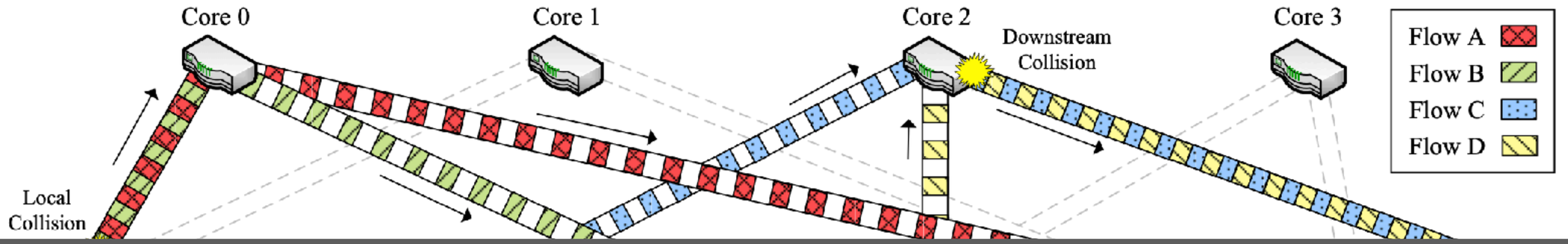
- If incoming bandwidth <= outgoing bandwidth Okay!
- If incoming bandwidth > outgoing bandwidth Bad!



How can we solve it?

Path Sharing

- If incoming bandwidth <= 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: MxN
  - 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} ← 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
```

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} ← 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
```

```
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 ← (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
```

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} ← 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
```

```
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 ← (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
2   foreach f ∈ ⟨src → dst⟩
3       f.rl ← true
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 ← 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 ← (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

```
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 ← 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 ← (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 ← (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
...
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
```

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

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 Herdra 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!

Summary

- Today
  - Flow scheduling in data center networks (I)

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