Advanced Computer Networks

Load Balancing in Data Center Networks (I)

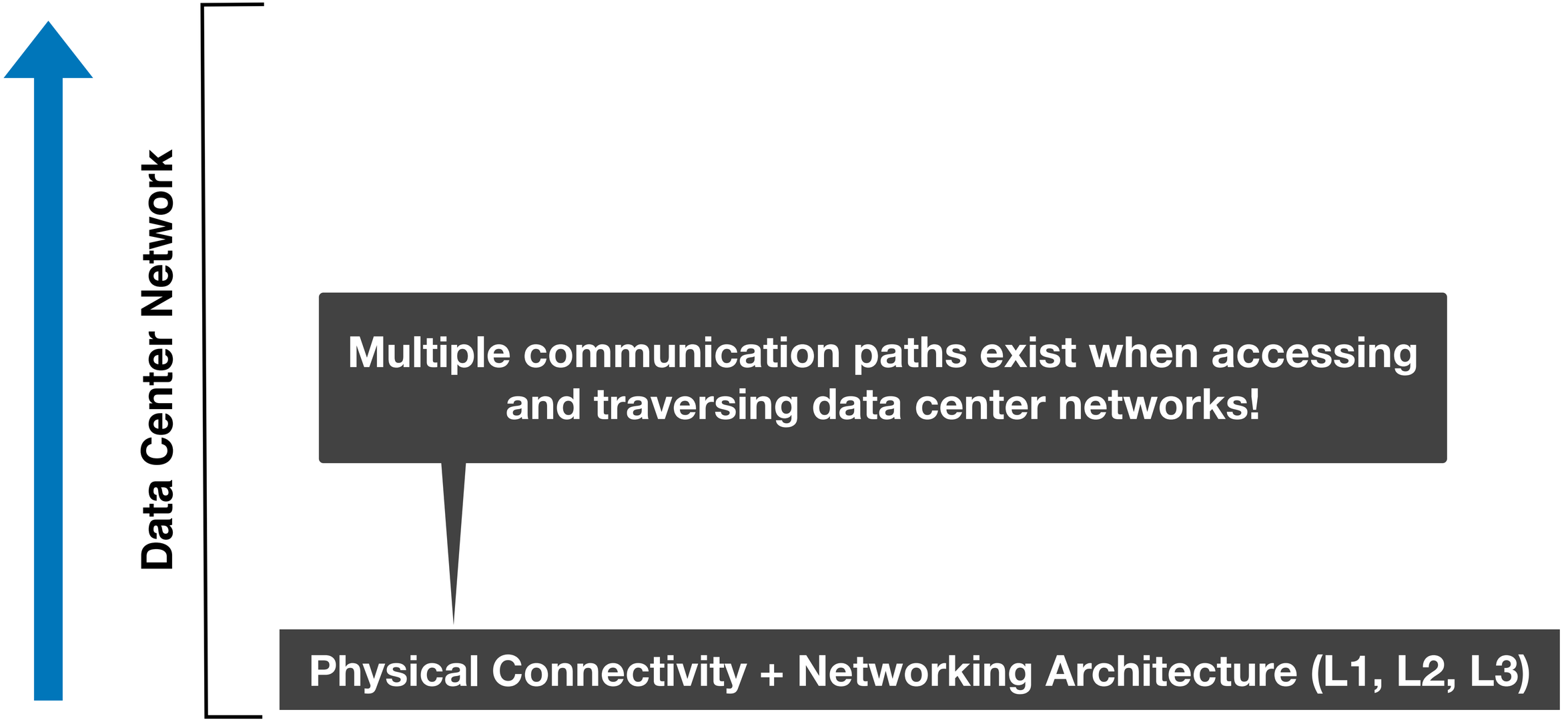
https://pages.cs.wisc.edu/~mgliu/CS740/F25/index.html

Ming Liu
mgliu@cs.wisc.edu

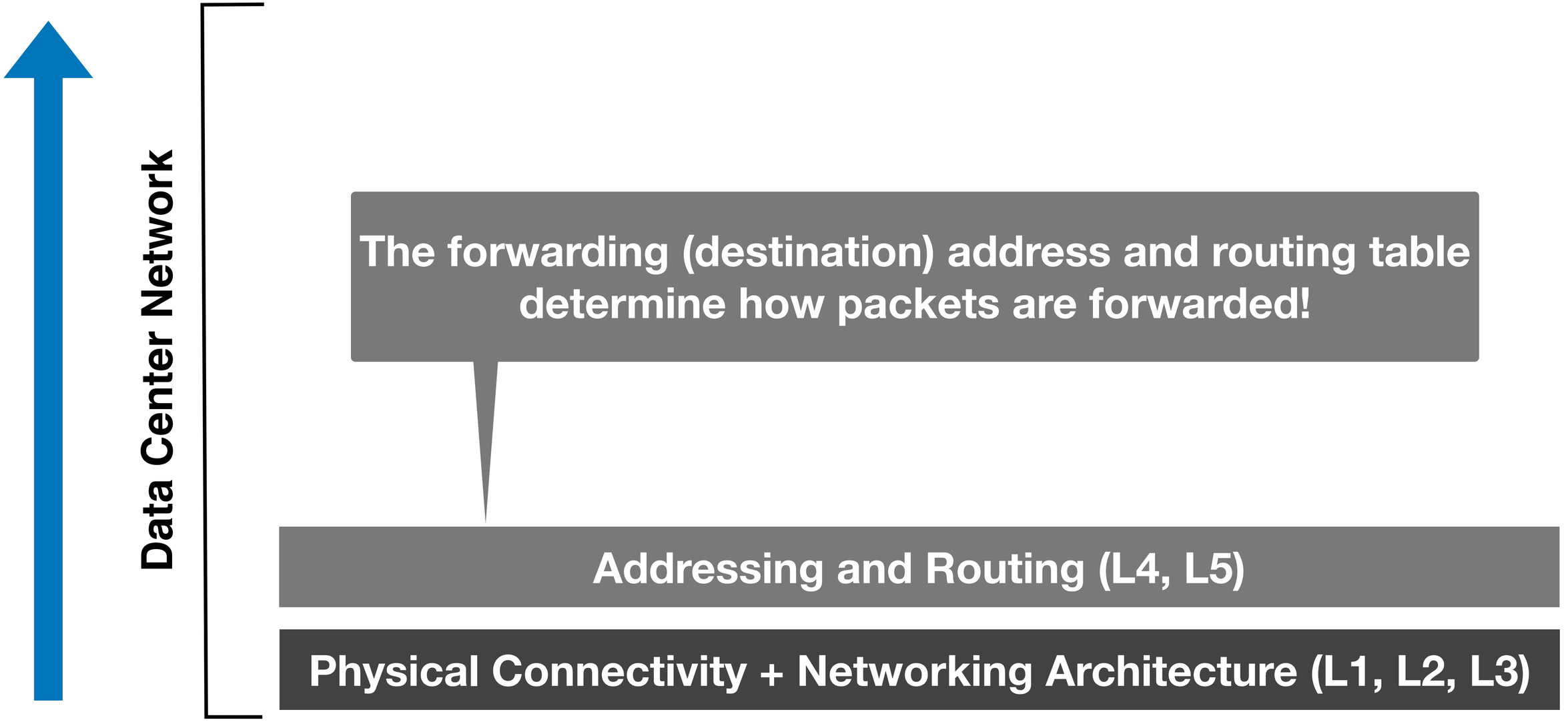
Outline

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

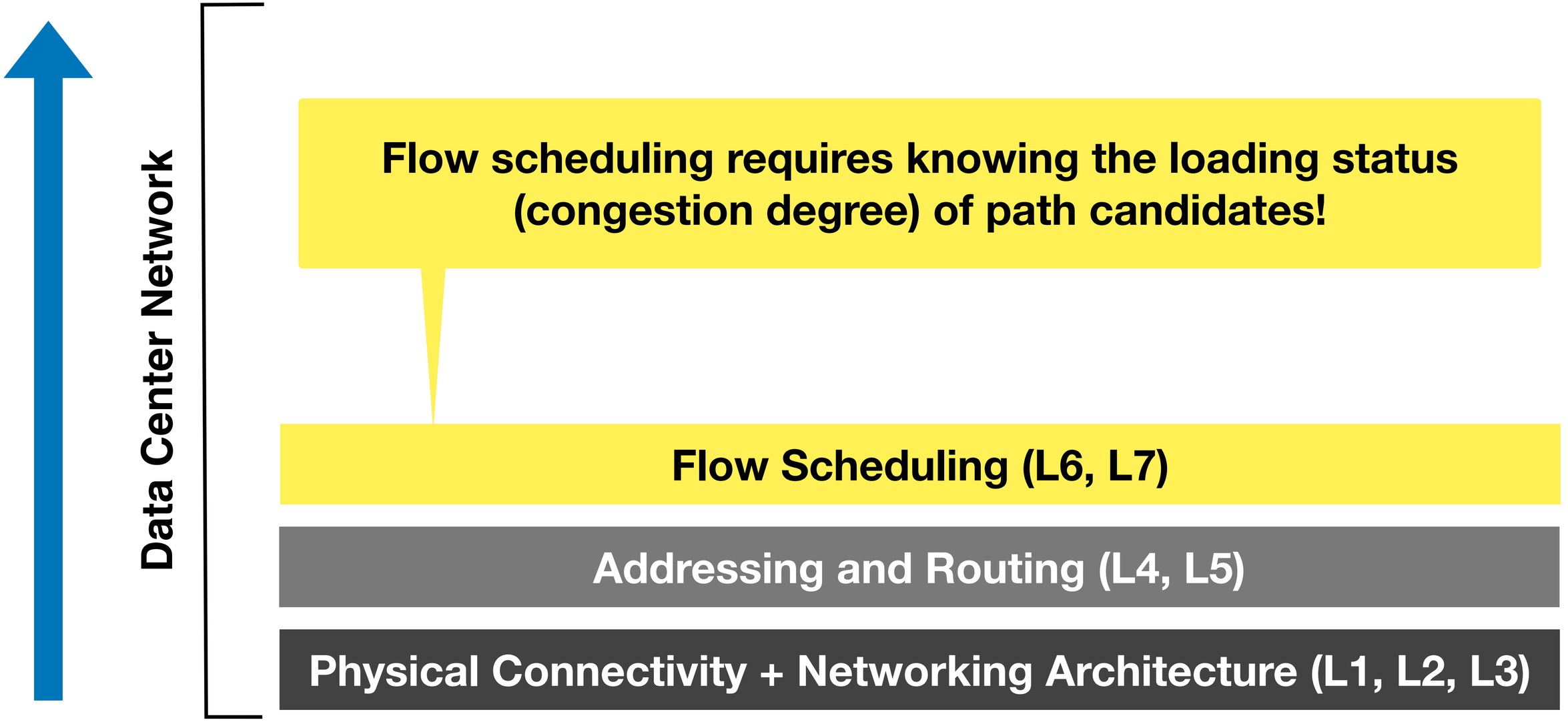
Where we are?


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)

Where we are?


Data Center Network
Flow scheduling requires knowing the loading status (congestion degree) of path candidates!
Flow Scheduling (L6, L7)
Addressing and Routing (L4, L5)
Physical Connectivity + Networking Architecture (L1, L2, L3)

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

1 Network Flow

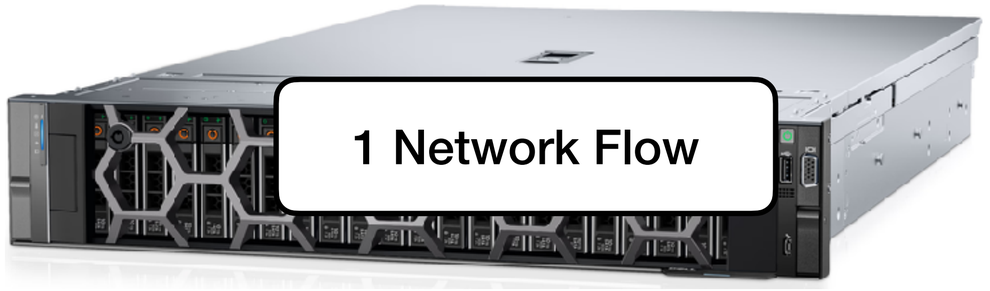
Network Flow

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

  } Host-Host communication channel

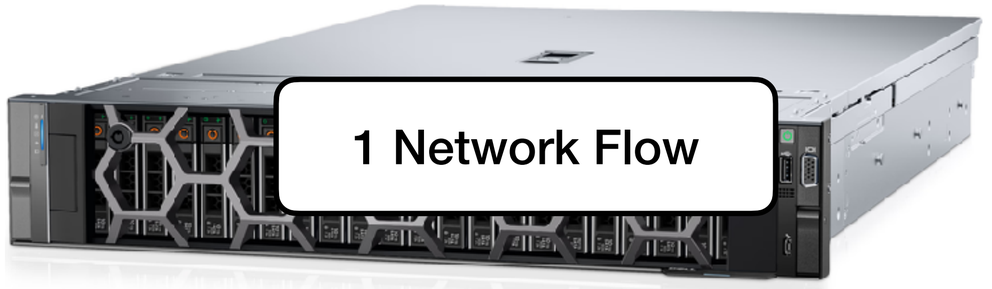
  - Source Port
  - Destination Port
  - Protocol Number

  } 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

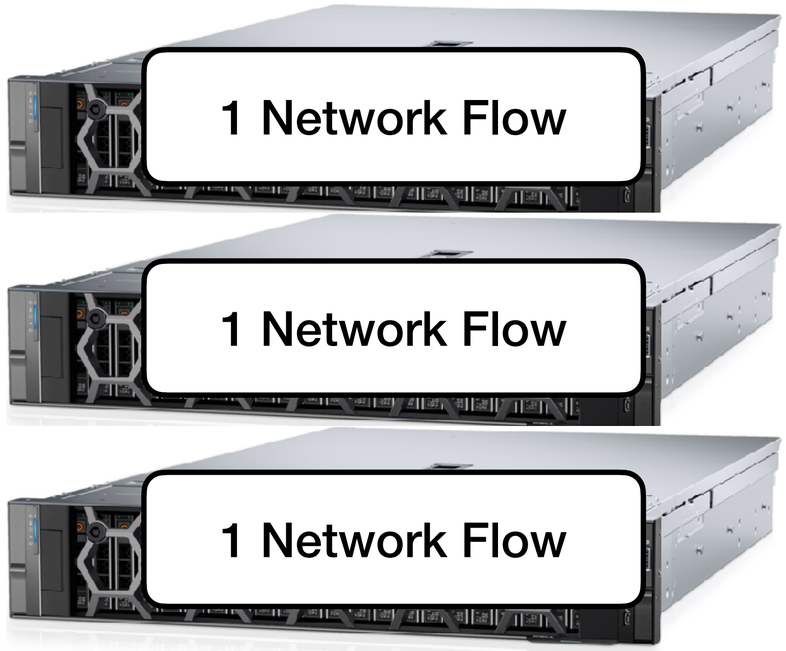


So, the throughput (BW) of a one-flow application is bounded!

How can we scale out the throughput (BW) of an application?

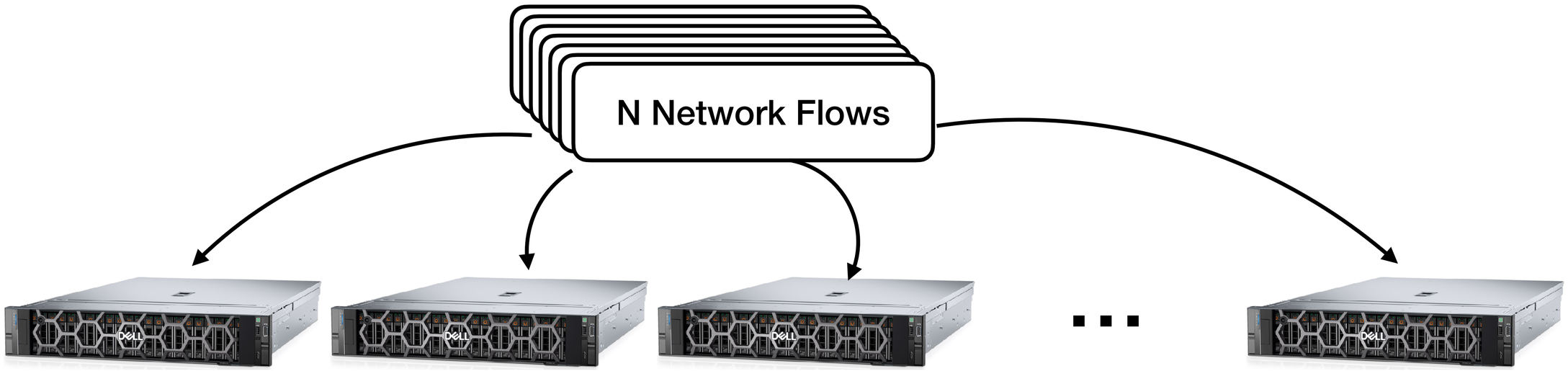
How can we scale out the throughput (BW) of an application?

How can we scale out the throughput (BW) of an application?

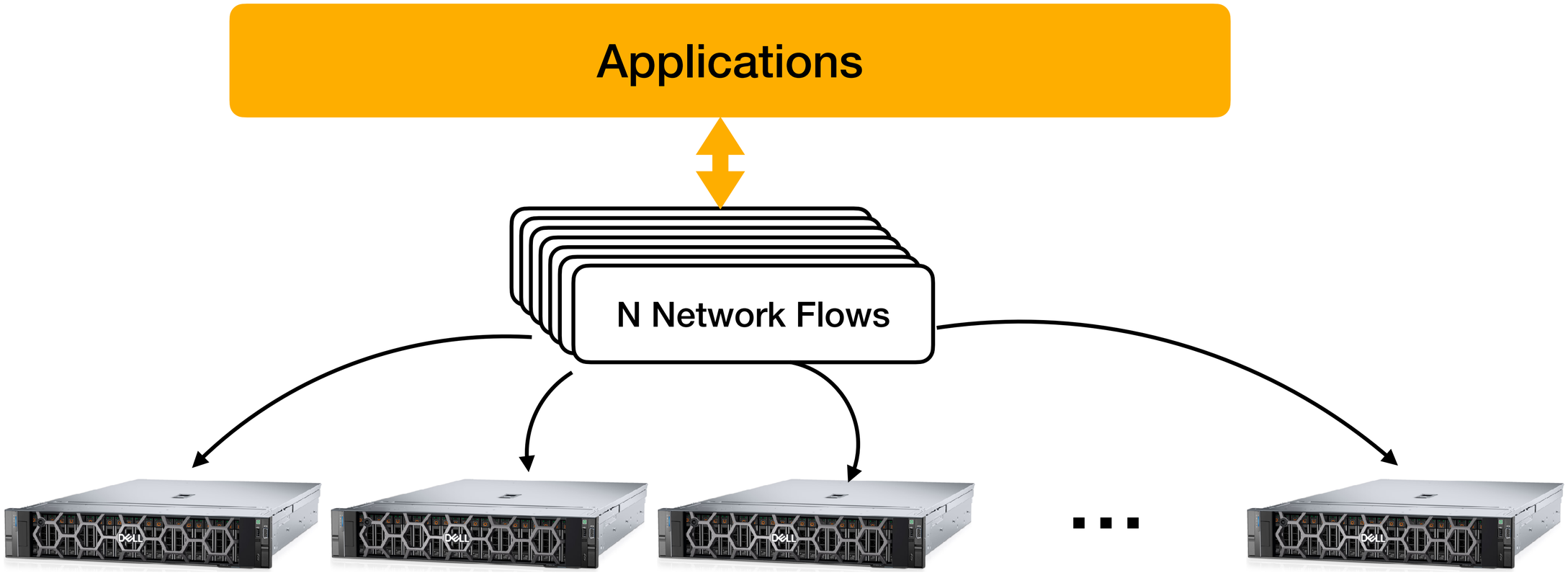


How can we scale out the throughput (BW) of an application?

Add more networking flows and distribute them to multiple servers!


N Network Flows

How can we build applications atop?



How can we build applications atop?

Applications

- How can we expose all network flows?
- How can we steer requests to different flows?

Approach #1: Outsourcing

- Expose all my flow addresses
  - Let the application (service) user make the decision

Approach #1: Outsourcing

- Expose all my flow addresses
  - Let the application (service) user make the decision
- For example,
  - DNS load balancing

Approach #1: Outsourcing

- Expose all my flow addresses
  - Let the application (service) user make the decision
- For example,
  - DNS load balancing
- Pros:
  - Simple
  - No need to do more implementation
- Cons:
  - Make internal states open (security)
  - Hard to use

I have developed a scalable and high-performance inference service.

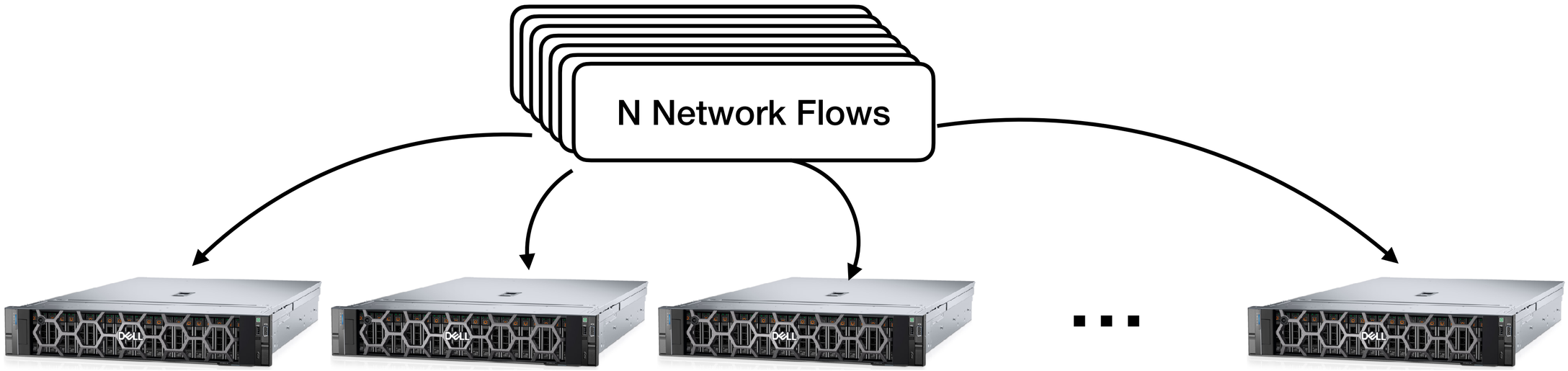
That’s great!!! How can I use it?

Here are all the service connection points:

- 1.2.3.4/1234, 1.2.3.4/5678
- 5.6.7.8/1234, 5.6.7.8/5678
- ……

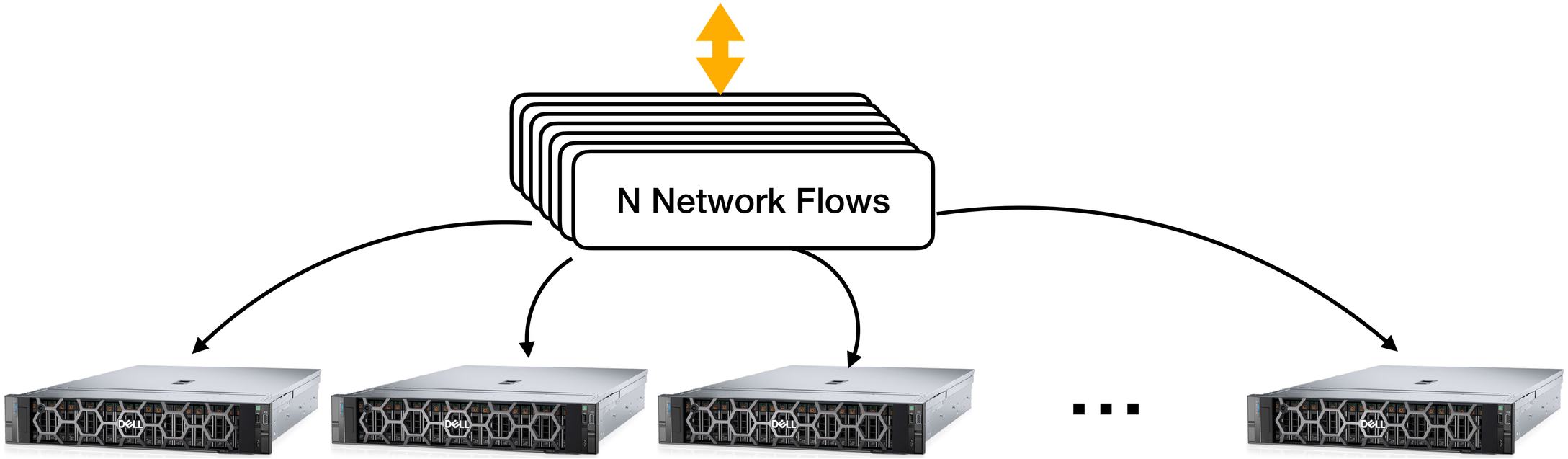
Okay……

Approach #2: Virtualization

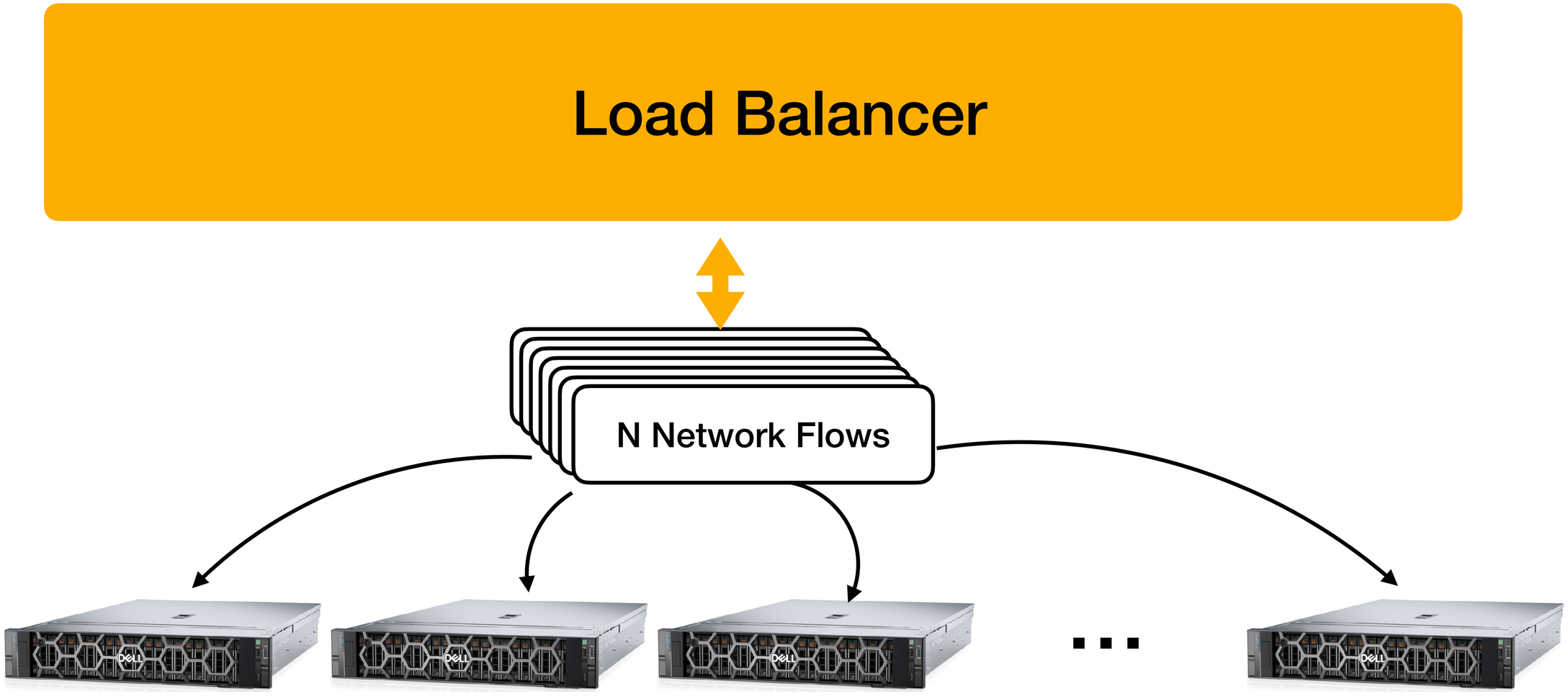


Approach #2: Virtualization

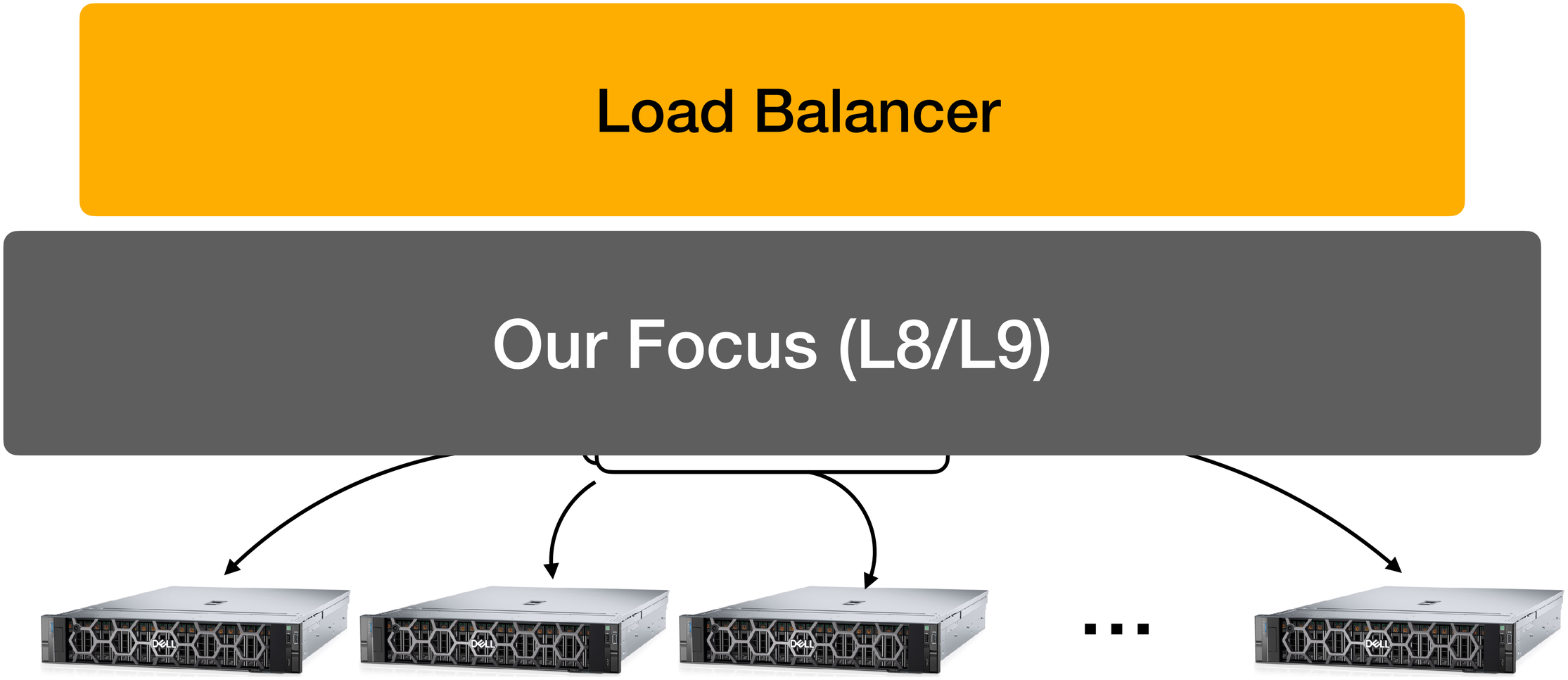
- Unify internal service addresses and expose one destination
- Steer traffic loads across multiple servers



Approach #2: Virtualization



Approach #2: Virtualization

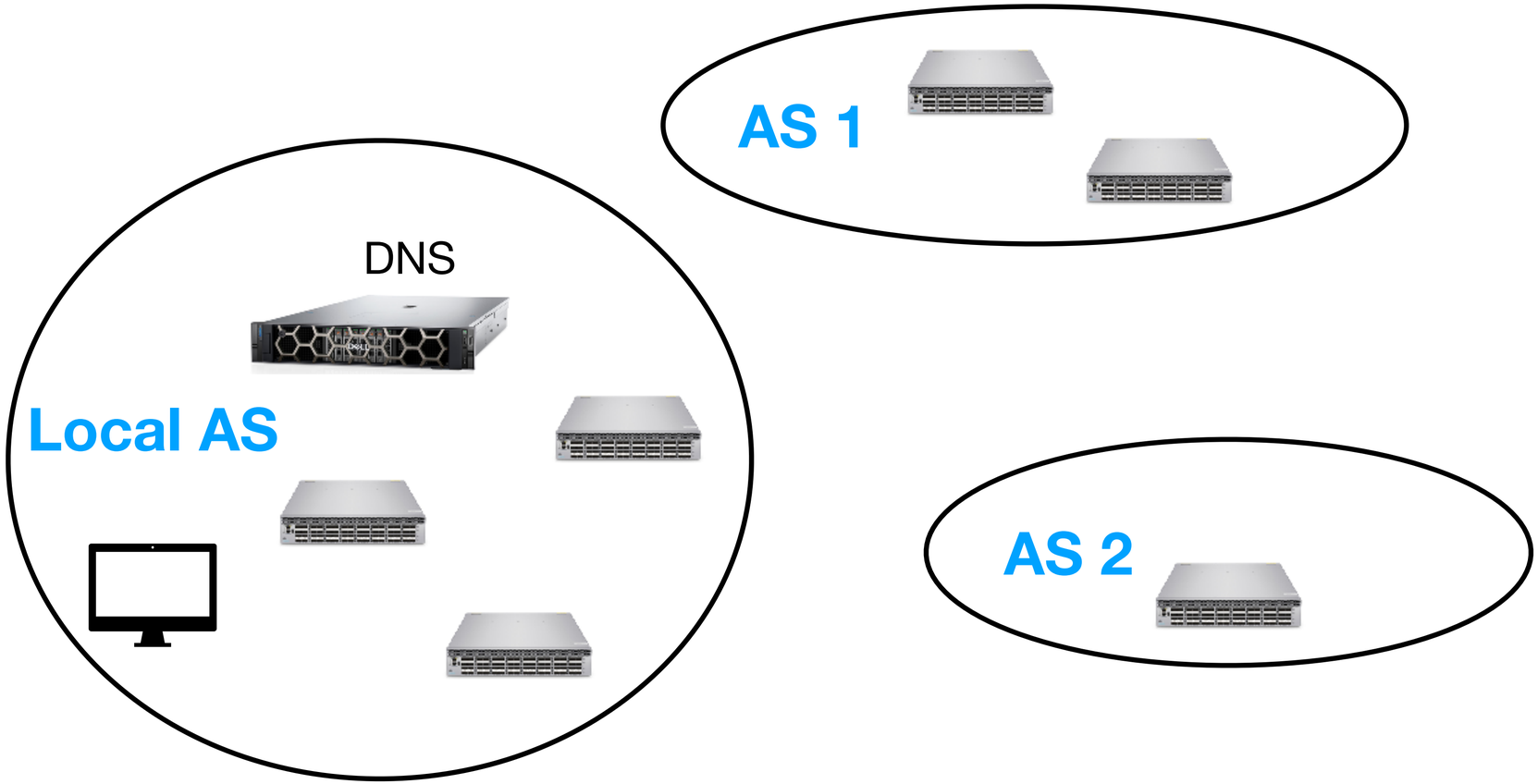


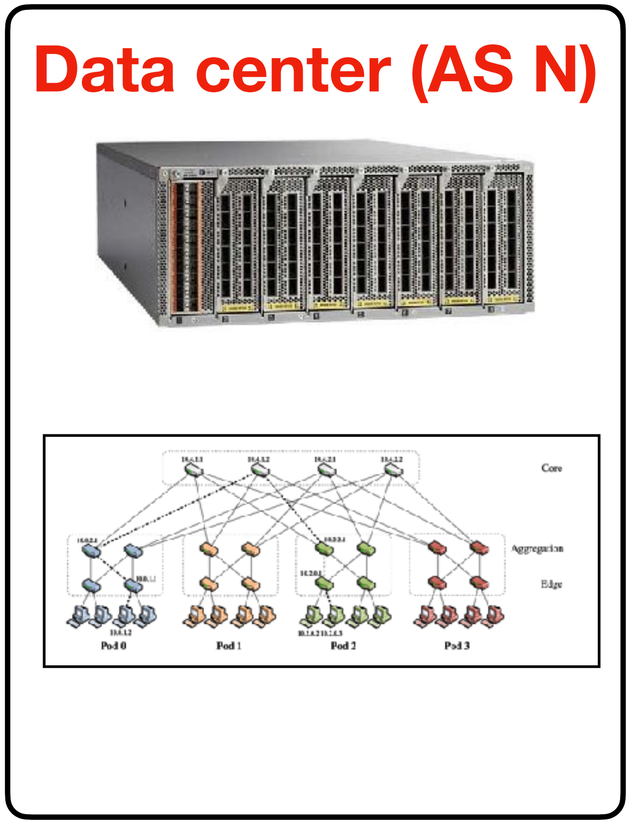
Virtual IP Address (VIP)

- Create VIPs and announce them to the outside world
  - A VIP is not assigned to any physical network interface

Virtual IP Address (VIP)

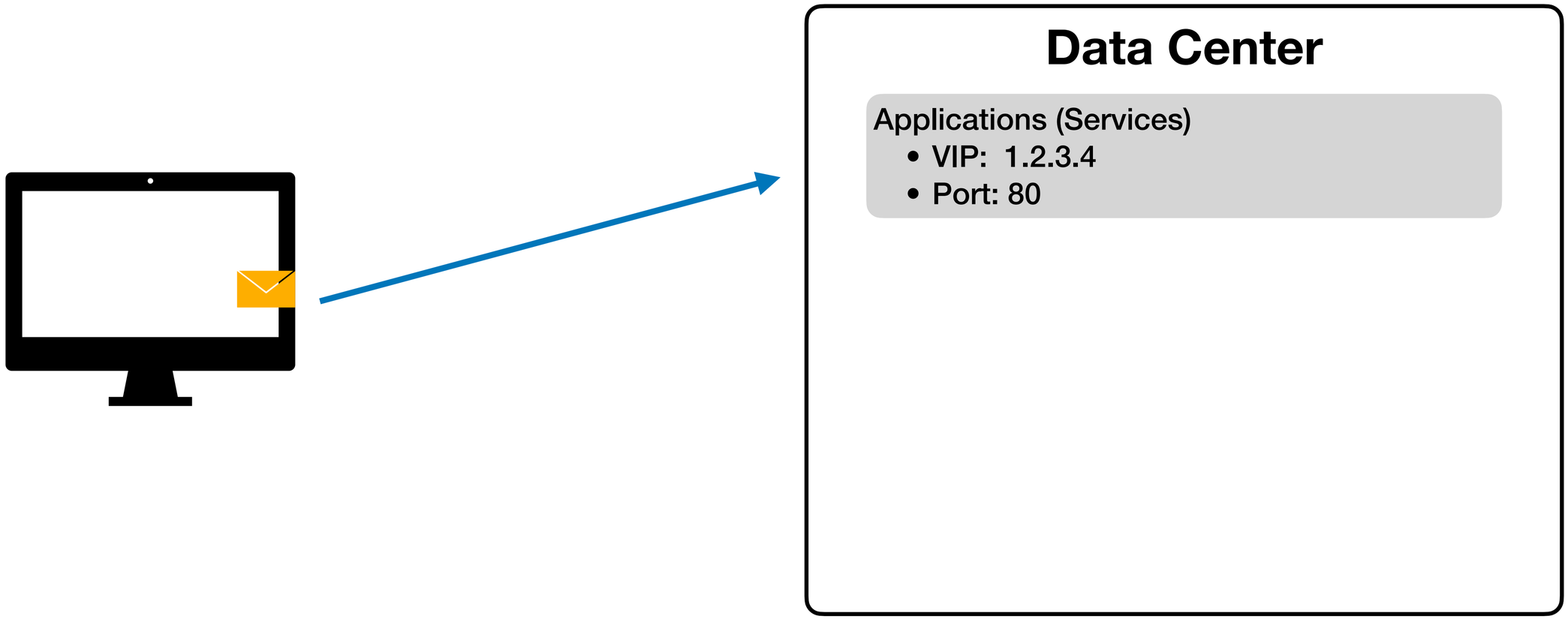
- Create VIPs and announce them to the outside world
  - A VIP is not assigned to any physical network interface





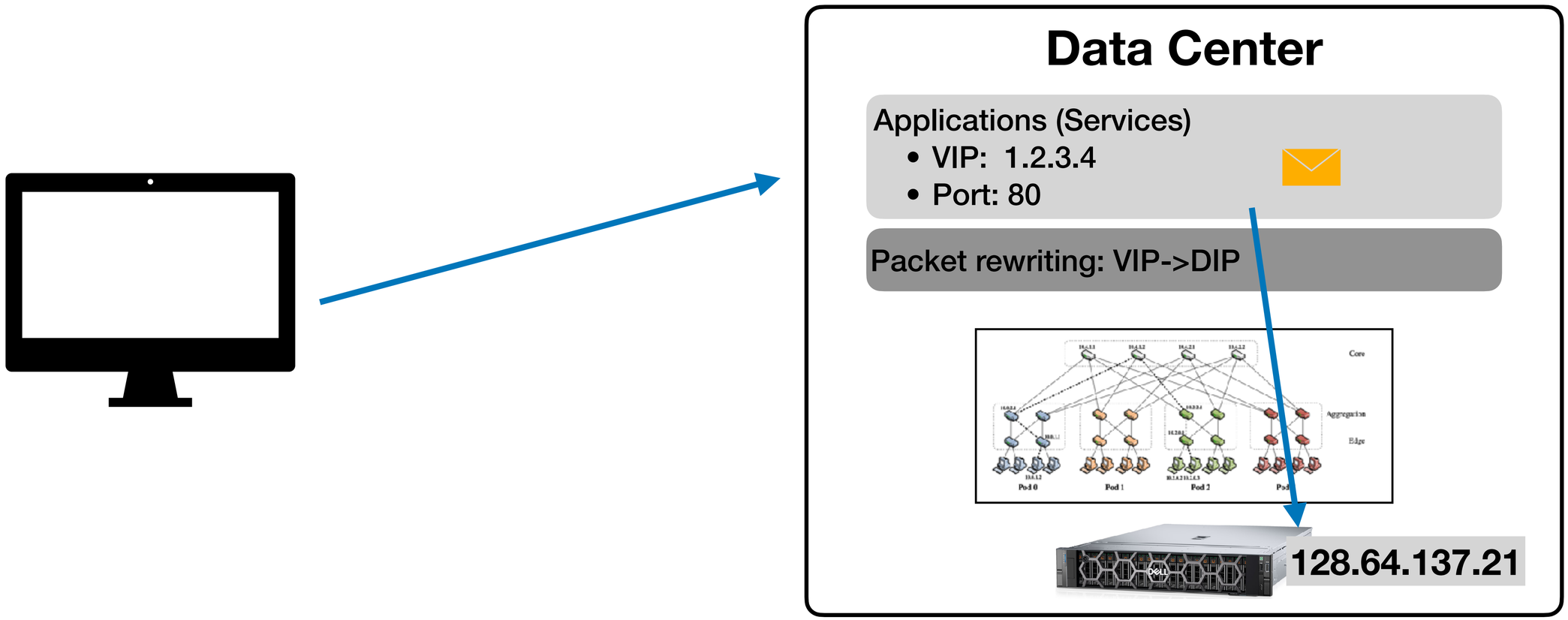
Virtual IP Address (VIP)

- Create VIPs and announce them to the outside world
  - A VIP is not assigned to any physical network interface



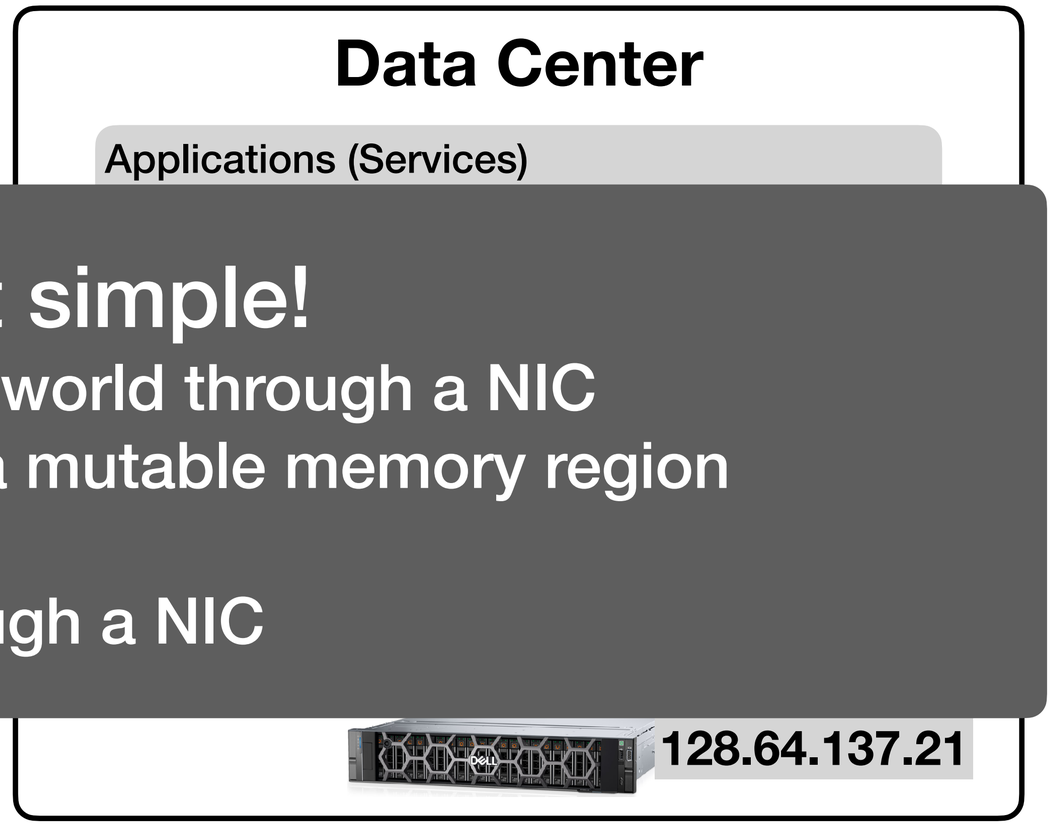
Address Rewriting VIP->DIP

- Modify the header field of each traversed packet
  - A DIP (destination IP address) is associated with a physical server



Address Rewriting VIP->IP

- Modify the header field of each traversed packet
  - A DIP (destination IP address) is associated with a physical server



Packet rewriting is not that simple!

- Receive packets from the external world through a NIC
- Store the partial (or full) packet in a mutable memory region
- Update the IP packet header
- Transmit to the internal world through a NIC

Address Rewriting VIP->IP

- Modify the header field of each traversed packet
  - A DIP (destination IP address) is associated with a physical server

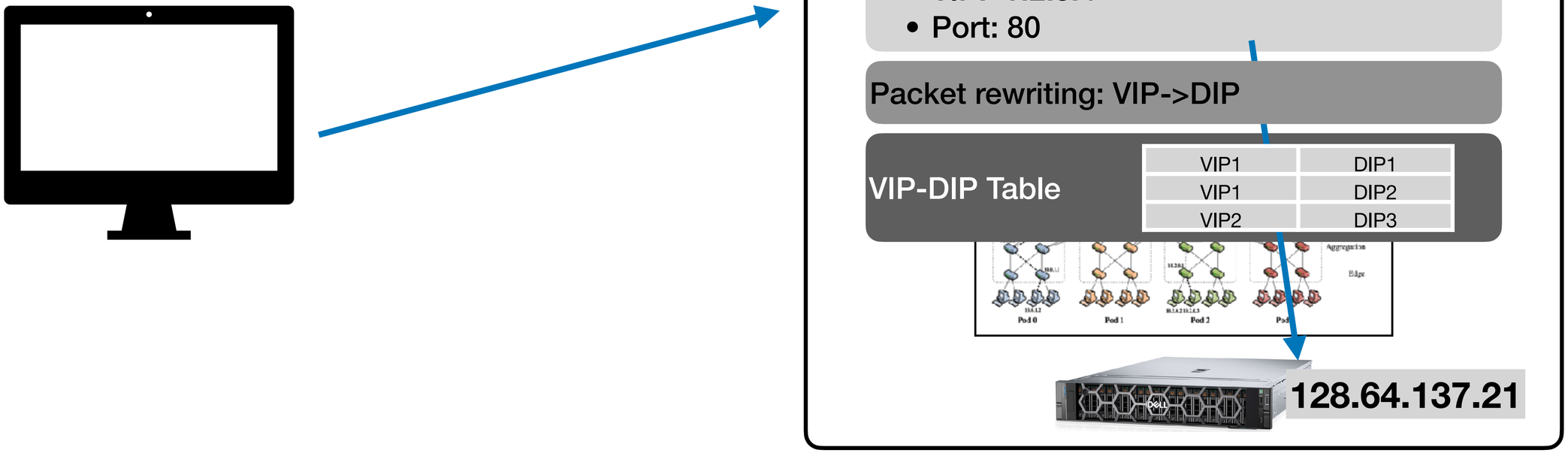
Design requirements of a load balancer

- **Multiple NIC interfaces**
- **Fast memory**
- **Line-rate processing**

128.64.137.21

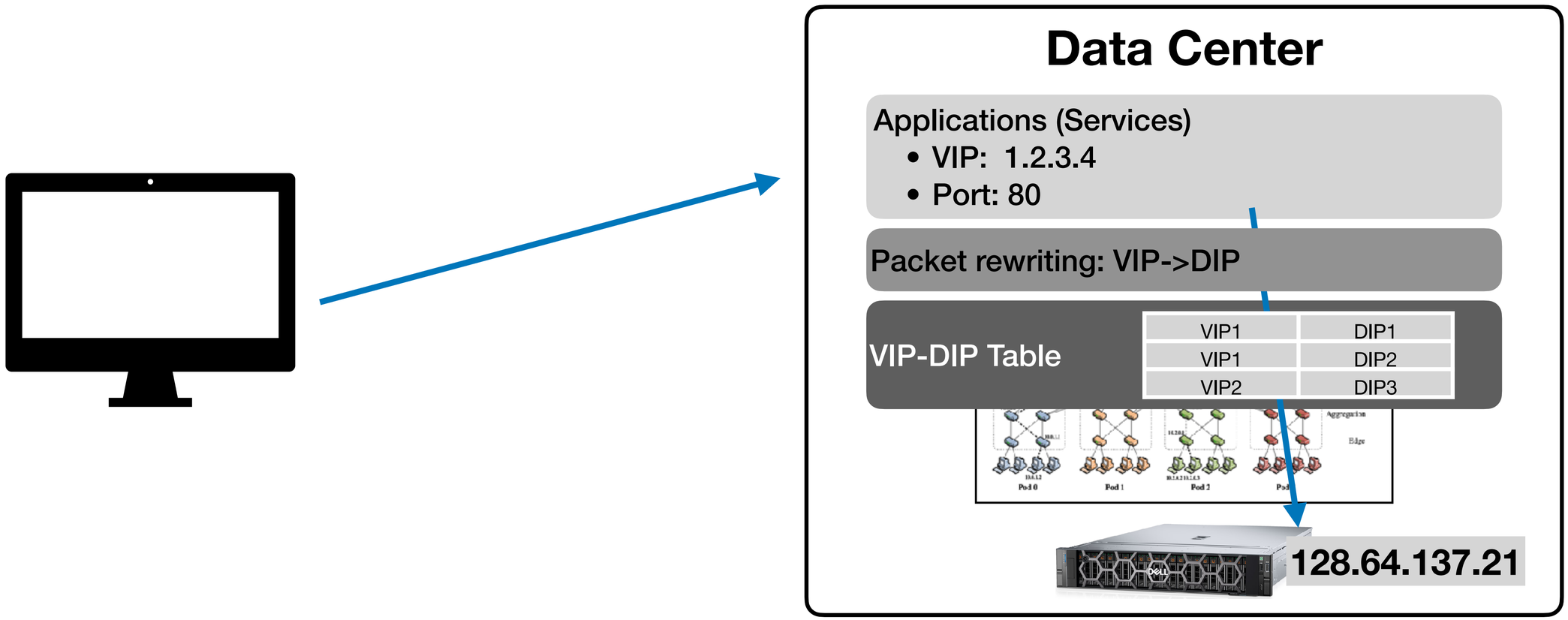
Where is DIP coming from?

VIP-DIP Mapping Table



VIP-DIP Mapping Table

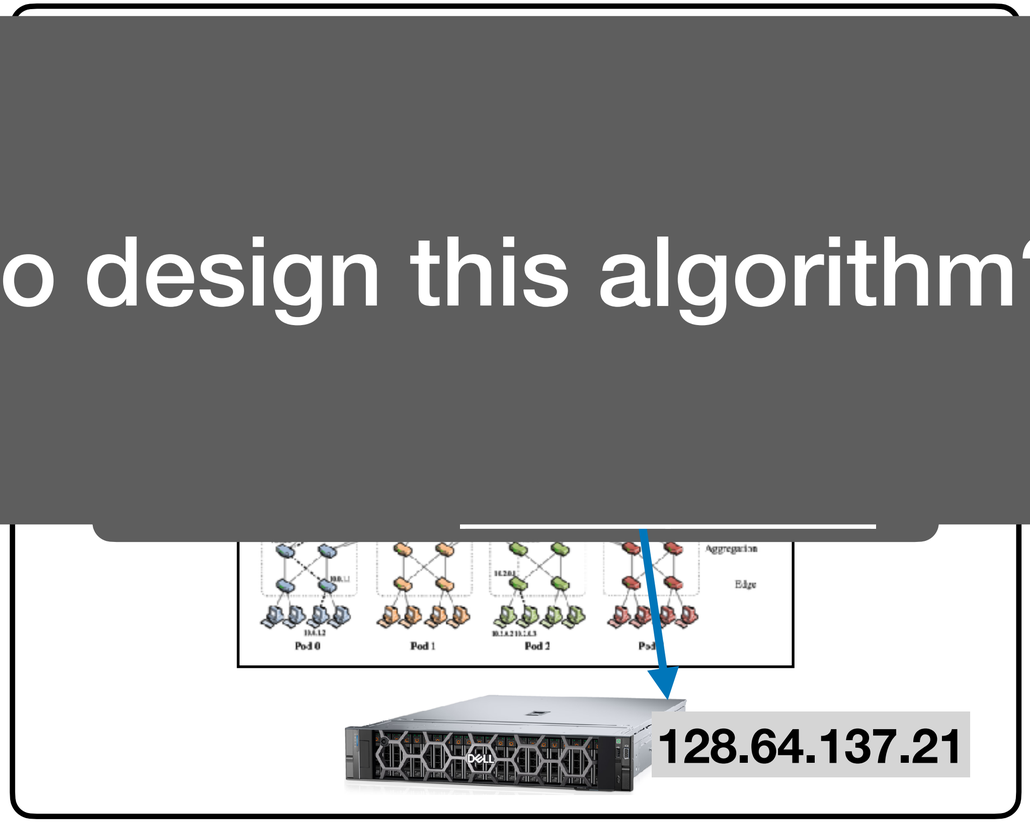
- An algorithm to select the DIP



VIP-DIP Mapping Table

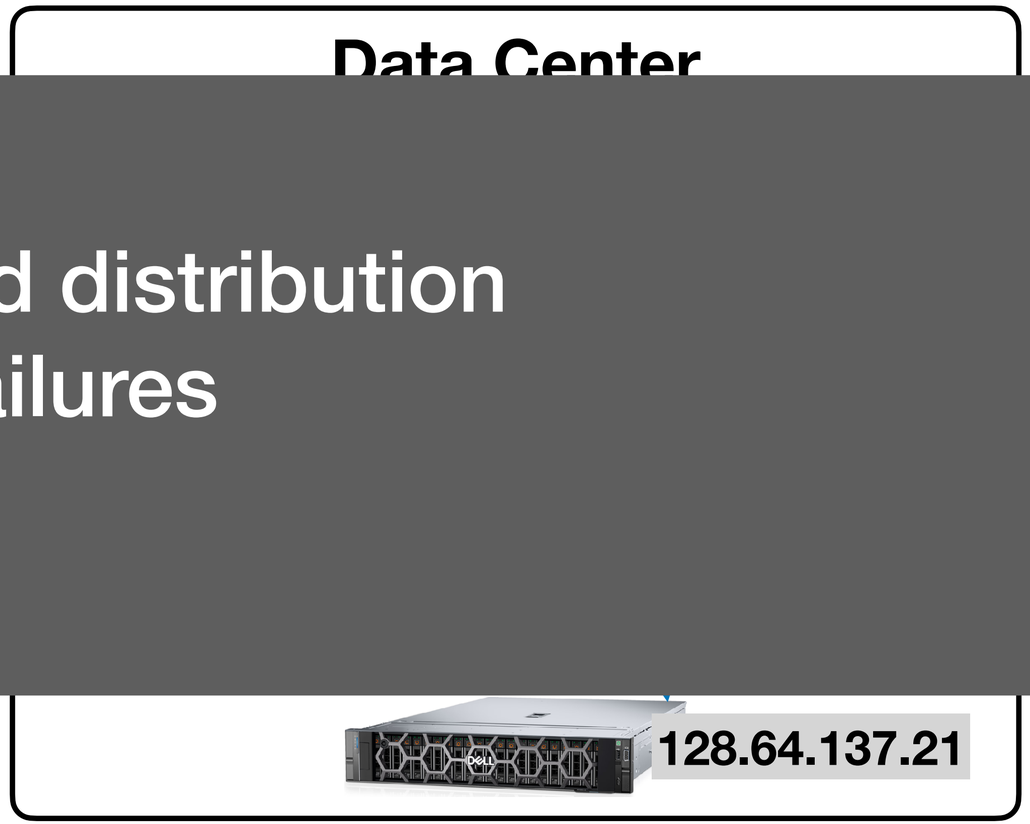
- An algorithm to select the DIP

What are the challenges to design this algorithm?



VIP-DIP Mapping Table

- An algorithm to select the DIP



- Functionality: evenly load distribution
- Reliability: work under failures
- Performance: fast

Maglev's Hashing Algorithm

- Idea: assign a preference list of all the lookup table positions
  - Consistent hashing
- Permutation table
  - offset = h1 (name[i]) mode M
  - skip = h2 (name[i]) mode (M-1) + 1

	B0	**B1**	**B2**
Offset	3	0	3
Skip	4	2	1

Maglev's Hashing Algorithm

- Idea: assign a preference list of all the lookup table positions
  - Consistent hashing
- Permutation table
  - offset = h1 (name[i]) mode M
  - skip = h2 (name[i]) mode (M-1) + 1

	B0	B1	B2
Offset	3	0	3
Skip	4	2	1

M=7, N=3, Permutation Table

	B0	B1	B2
0			
1			
2			
3			
4			
5			
6			

Maglev's Hashing Algorithm

- Idea: assign a preference list of all the lookup table positions
  - Consistent hashing
- Permutation table
  - offset = h1 (name[i]) mode M
  - skip = h2 (name[i]) mode (M-1) + 1

	B0	B1	B2
Offset	3	0	3
Skip	4	2	1

M=7, N=3, Permutation Table

	B0	B1	B2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Maglev's Lookup Algorithm

- Output: A lookup table

Pseudocode 1 Populate Maglev hashing lookup table.

```
1:  function POPULATE
2:      for each i < N do next[i] ← 0 end for
3:      for each j < M do entry[j] ← −1 end for
4:      n ← 0
5:      while true do
6:          for each i < N do
7:              c ← permutation[i][next[i]]
8:              while entry[c] ≥ 0 do
9:                  next[i] ← next[i] + 1
10:                 c ← permutation[i][next[i]]
11:             end while
12:             entry[c] ← i
13:             next[i] ← next[i] + 1
14:             n ← n + 1
15:             if n = M then return end if
16:         end for
17:     end while
18: end function
```

Permutation Table

	B0	B1	B2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	
1	
2	
3	
4	
5	
6	

Maglev's Lookup Algorithm

- Output: A lookup table

Pseudocode 1 Populate Maglev hashing lookup table.

```
1:  function POPULATE
2:      for each i < N do next[i] ← 0 end for
3:      for each j < M do entry[j] ← −1 end for
4:      n ← 0
5:      while true do
6:          for each i < N do
7:              c ← permutation[i][next[i]]
8:              while entry[c] ≥ 0 do
9:                  next[i] ← next[i] + 1
10:                 c ← permutation[i][next[i]]
11:             end while
12:             entry[c] ← i
13:             next[i] ← next[i] + 1
14:             n ← n + 1
15:             if n = M then return end if
16:         end for
17:     end while
18: end function
```

Permutation Table

	B0	B1	B2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	
1	
2	
3	B0
4	
5	
6	

Maglev's Lookup Algorithm

- Output: A lookup table

Pseudocode 1 Populate Maglev hashing lookup table.

```
1:  function POPULATE
2:      for each i < N do next[i] ← 0 end for
3:      for each j < M do entry[j] ← −1 end for
4:      n ← 0
5:      while true do
6:          for each i < N do
7:              c ← permutation[i][next[i]]
8:              while entry[c] ≥ 0 do
9:                  next[i] ← next[i] + 1
10:                 c ← permutation[i][next[i]]
11:             end while
12:             entry[c] ← i
13:             next[i] ← next[i] + 1
14:             n ← n + 1
15:             if n = M then return end if
16:         end for
17:     end while
18: end function
```

Permutation Table

	B0	B1	B2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	B1
1	
2	
3	B0
4	
5	
6	

Maglev's Lookup Algorithm

- Output: A lookup table

Pseudocode 1 Populate Maglev hashing lookup table.

```
1:  function POPULATE
2:      for each i < N do next[i] ← 0 end for
3:      for each j < M do entry[j] ← −1 end for
4:      n ← 0
5:      while true do
6:          for each i < N do
7:              c ← permutation[i][next[i]]
8:              while entry[c] ≥ 0 do
9:                  next[i] ← next[i] + 1
10:                 c ← permutation[i][next[i]]
11:             end while
12:             entry[c] ← i
13:             next[i] ← next[i] + 1
14:             n ← n + 1
15:             if n = M then return end if
16:         end for
17:     end while
18: end function
```

Permutation Table

	B0	B1	B2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	B1
1	
2	
3	B0
4	B2
5	
6	

Maglev's Lookup Algorithm

- Output: A lookup table

Pseudocode 1 Populate Maglev hashing lookup table.

```
1:  function POPULATE
2:      for each i < N do next[i] ← 0 end for
3:      for each j < M do entry[j] ← −1 end for
4:      n ← 0
5:      while true do
6:          for each i < N do
7:              c ← permutation[i][next[i]]
8:              while entry[c] ≥ 0 do
9:                  next[i] ← next[i] + 1
10:                 c ← permutation[i][next[i]]
11:             end while
12:             entry[c] ← i
13:             next[i] ← next[i] + 1
14:             n ← n + 1
15:             if n = M then return end if
16:         end for
17:     end while
18: end function
```

Permutation Table

	B0	B1	B2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	B1
1	B0
2	
3	B0
4	B2
5	
6	

Maglev's Lookup Algorithm

- Output: A lookup table

Pseudocode 1 Populate Maglev hashing lookup table.

```
1:  function POPULATE
2:      for each i < N do next[i] ← 0 end for
3:      for each j < M do entry[j] ← −1 end for
4:      n ← 0
5:      while true do
6:          for each i < N do
7:              c ← permutation[i][next[i]]
8:              while entry[c] ≥ 0 do
9:                  next[i] ← next[i] + 1
10:                 c ← permutation[i][next[i]]
11:             end while
12:             entry[c] ← i
13:             next[i] ← next[i] + 1
14:             n ← n + 1
15:             if n = M then return end if
16:         end for
17:     end while
18: end function
```

Permutation Table

	B0	B1	B2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	B1
1	B0
2	B1
3	B0
4	B2
5	B2
6	B0

Node Removal

Pseudocode 1 Populate Maglev hashing lookup table.

```
1:  function POPULATE
2:      for each i < N do next[i] ← 0 end for
3:      for each j < M do entry[j] ← −1 end for
4:      n ← 0
5:      while true do
6:          for each i < N do
7:              c ← permutation[i][next[i]]
8:              while entry[c] ≥ 0 do
9:                  next[i] ← next[i] + 1
10:                 c ← permutation[i][next[i]]
11:             end while
12:             entry[c] ← i
13:             next[i] ← next[i] + 1
14:             n ← n + 1
15:             if n = M then return end if
16:         end for
17:     end while
18: end function
```

Permutation Table

	B0	B1	B2
0	3	0	3
1	0	2	4
2	4	4	5
3	1	6	6
4	5	1	0
5	2	3	1
6	6	5	2

Lookup Table

0	B1
1	B0
2	B1
3	B0
4	B2
5	B2
6	B0

Node Removal

	Before	After
0	B1	B0
1	B0	B0
2	B1	B0
3	B0	B0
4	B2	B2
5	B2	B2
6	B0	B2

Node Removal

- Some backend changes have to happen

	Before	After
0	B1	B0
1	B0	B0
2	B1	B0
3	B0	B0
4	B2	B2
5	B2	B2
6	B0	B2

More Design Requirements

Design requirements of a load balancer

- Multiple NIC interfaces
- Fast memory
- Line-rate processing
- Maintain a load-balancing table
- Perform hashing and lookup

5	B2	B2
6	B0	B2

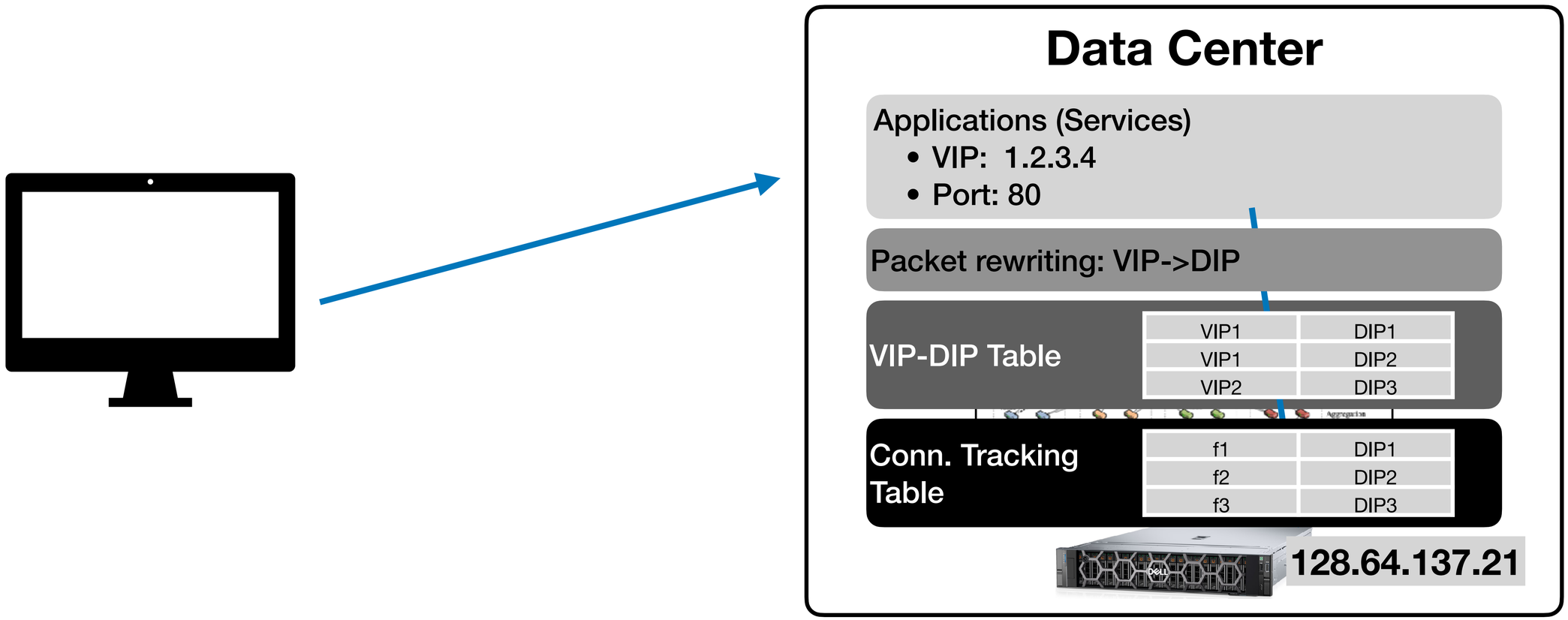
More Design Requirements

Do we need to remember the mapping between a VIP and a DIP?

5	B2	B2
6	B0	B2

Connection Tracking

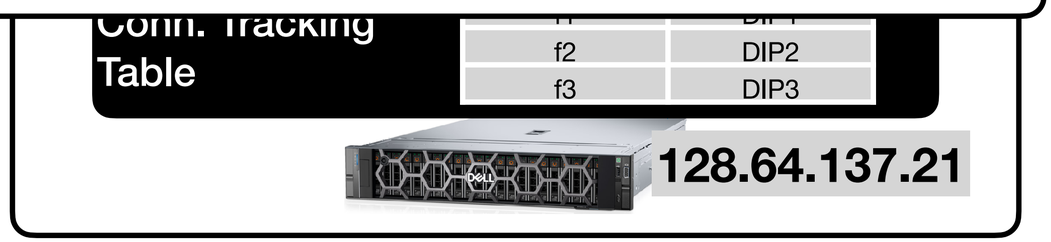
- A table bookkeeps all incoming connections
  - Remember the chosen DIP for a flow for performance and reliability



Connection Tracking

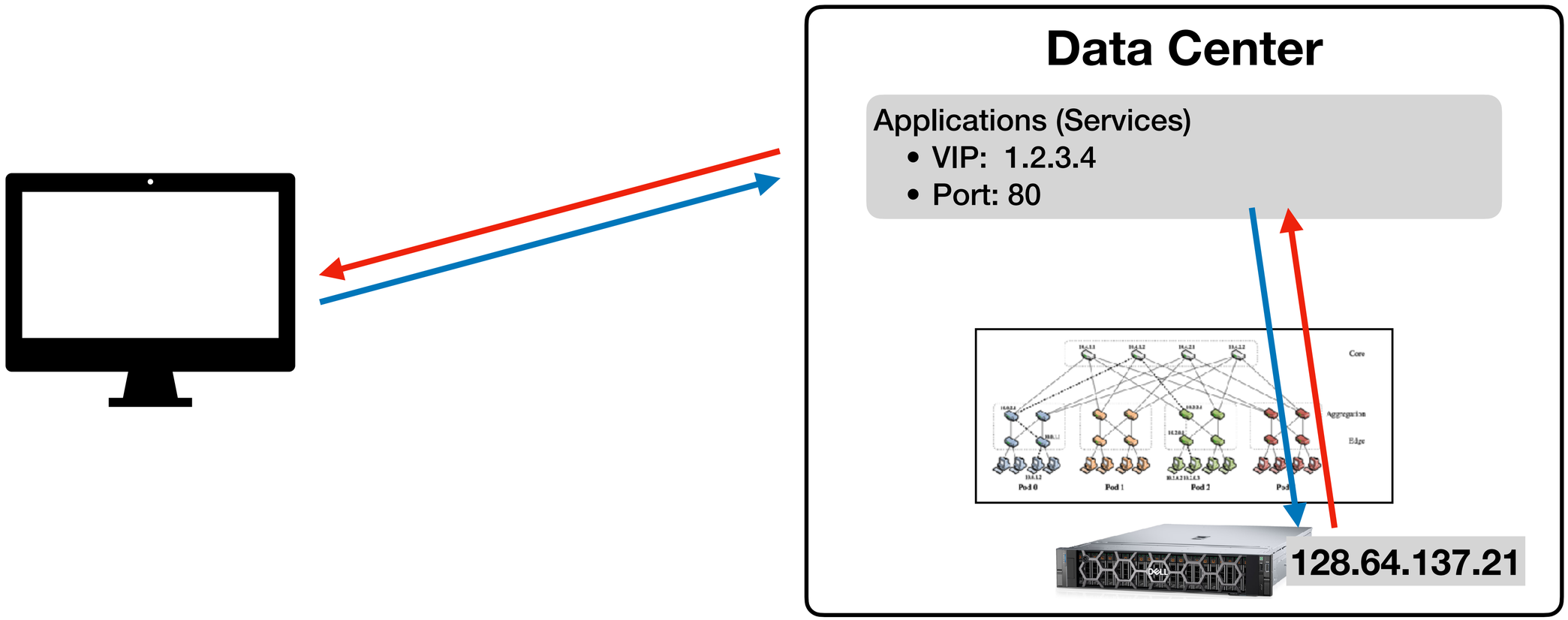
Design requirements of a load balancer

- Multiple NIC interfaces
- Fast memory
- Line-rate processing
- Maintain a load-balancing table
- Perform hashing and lookup
- Maintain a connection tracking table



Light Reply Paths

- Update the connection tracking table if the flow is done
  - People indeed perform other telemetry tasks

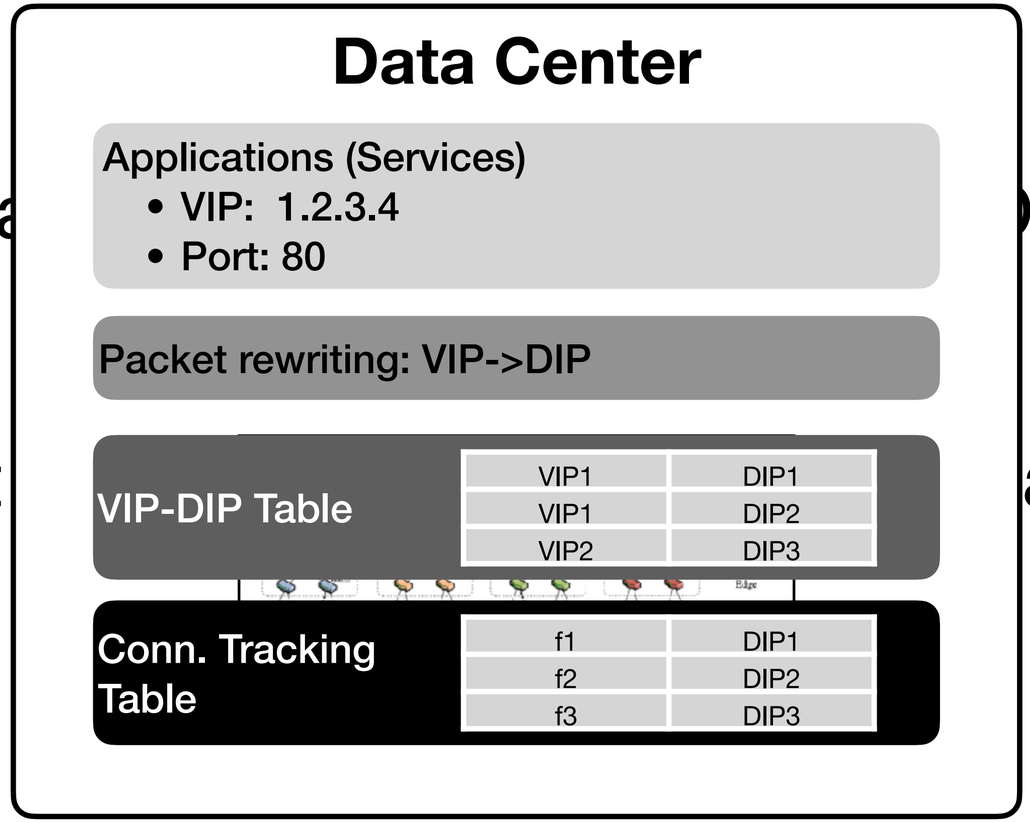


Combine Everything Together

- #1: Per-packet header rewriting
- #2: Per-flow load balancing based on a VIP-DIP table
- #3: Per-packet connection tracking based on an telemetry table

Combine Everything Together

- #1: Per-packet header rewriting
- #2: Per-flow loa IP table
- #3: Per-packet an telemetry table



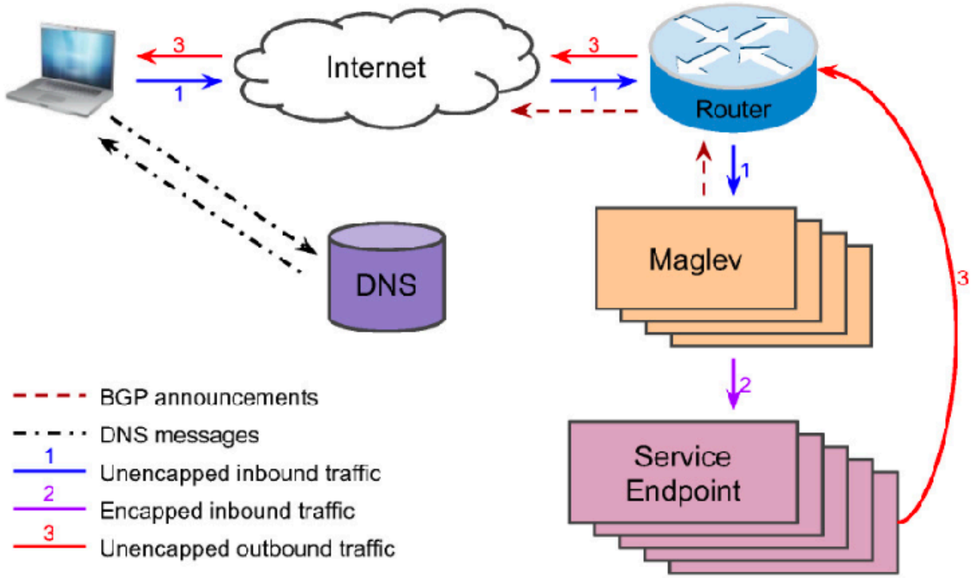
Combine Everything Together

Design requirements of a load balancer

- Multiple NIC interfaces
- Fast memory
- Line-rate processing
- Maintain a load-balancing table
- Perform hashing and lookup
- Maintain a connection tracking table

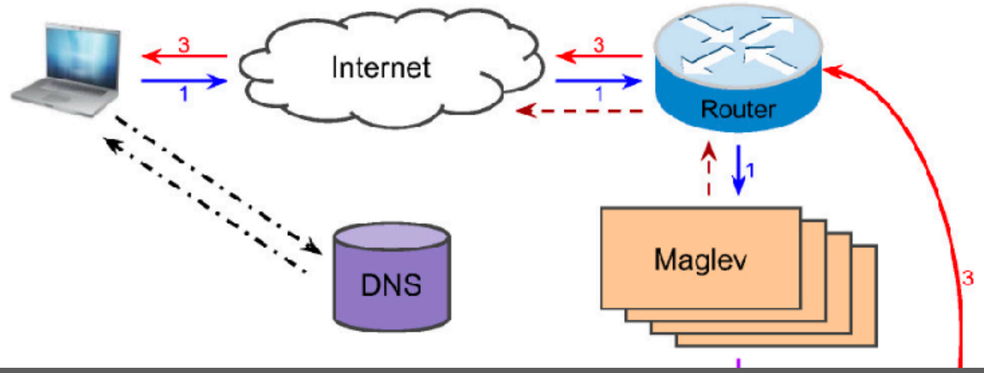
Implementation under Scale

- Maglev: software-only solution
  - A bunch of distributed Maglev servers



Implementation under Scale

- Maglev: software-only solution
  - A bunch of distributed Maglev servers



But how does a packet know which Magelev load balancers to use?

Implementation under Scale

- Maglev: software-only solution

But how does a packet know which Magelev load balancers to use?

ECMP!
A load balancer for load balancing

Summary

- Today
  - Load balancing in data center networks (I)

- Next
  - Load balancing in data center networks (II)