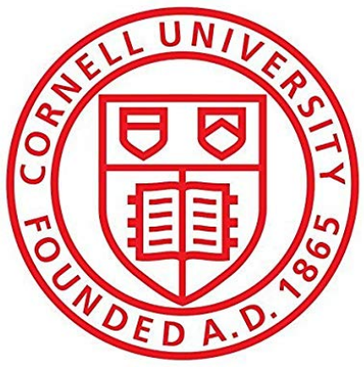
Towards High-speed Networking in the Post-Moore Era

Vishal Shrivastav

Ph.D. Thesis Defense

Cornell University 2020

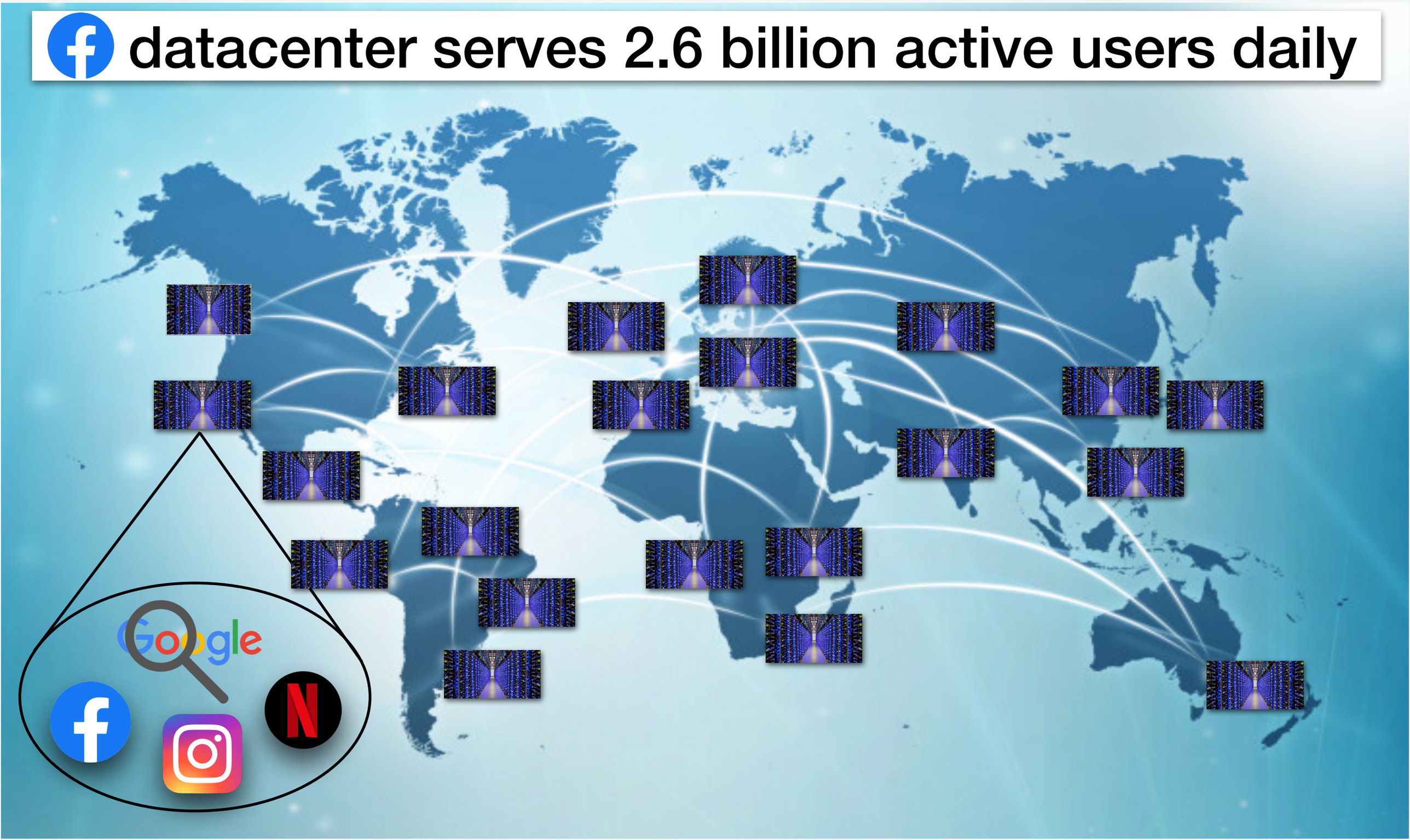


Datacenters 101

Datacenters 101

Datacenters 101

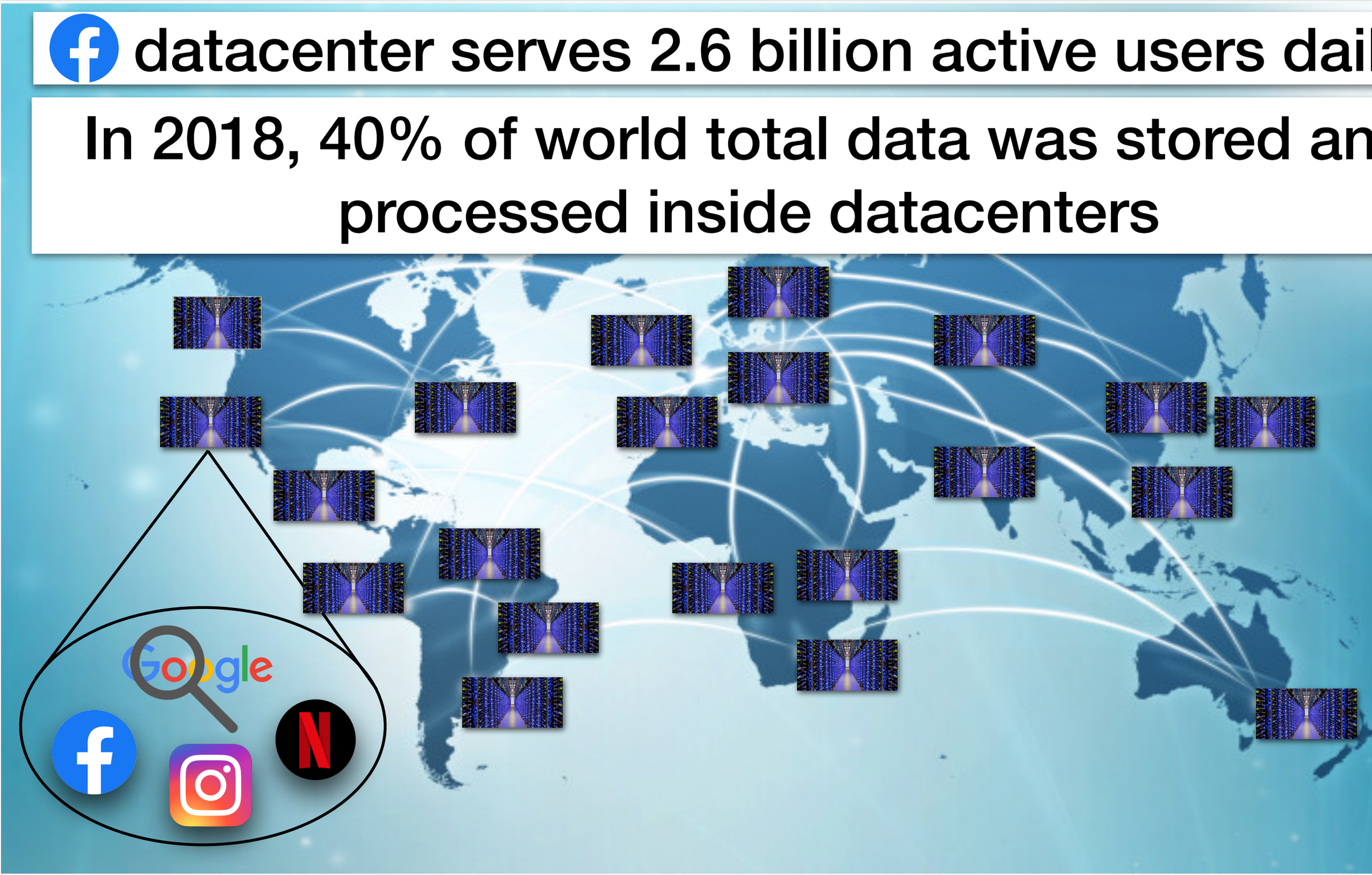
Datacenters 101



Datacenters 101

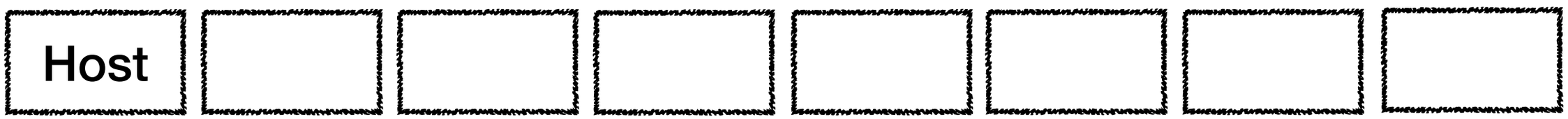
datacenter serves 2.6 billion active users daily

In 2018, 40% of world total data was stored and processed inside datacenters

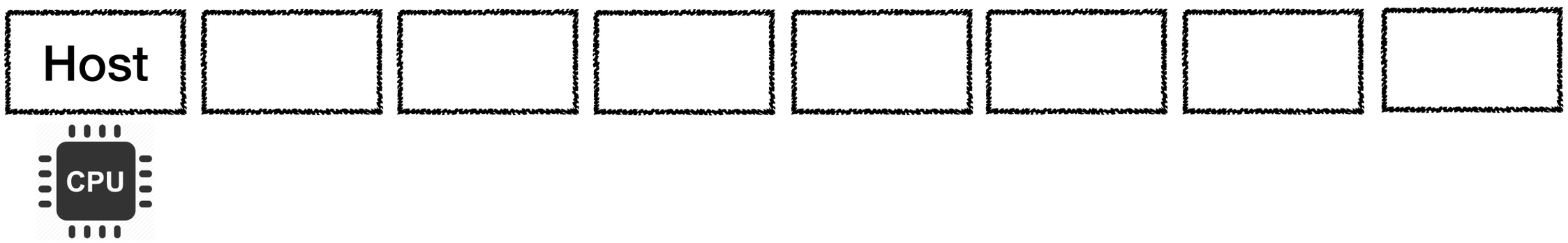


Inside a Typical Datacenter

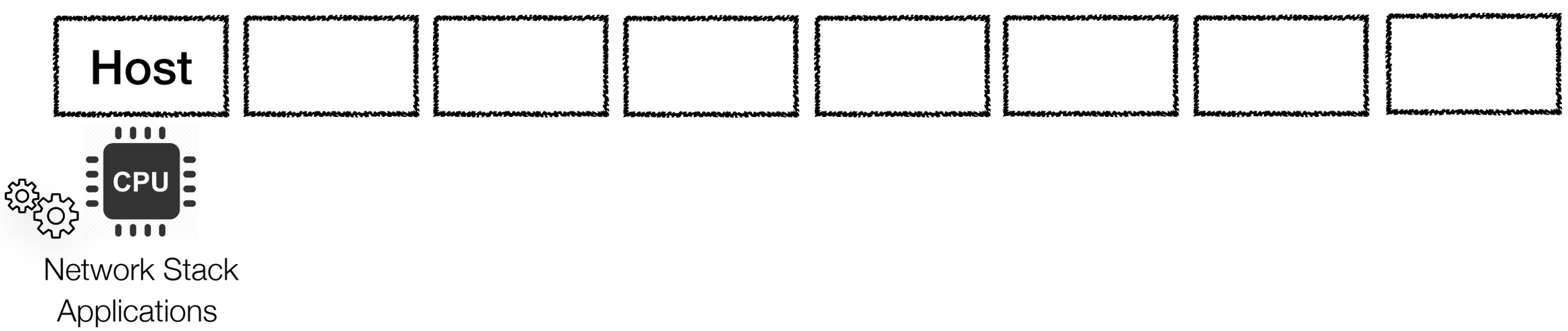
Inside a Typical Datacenter



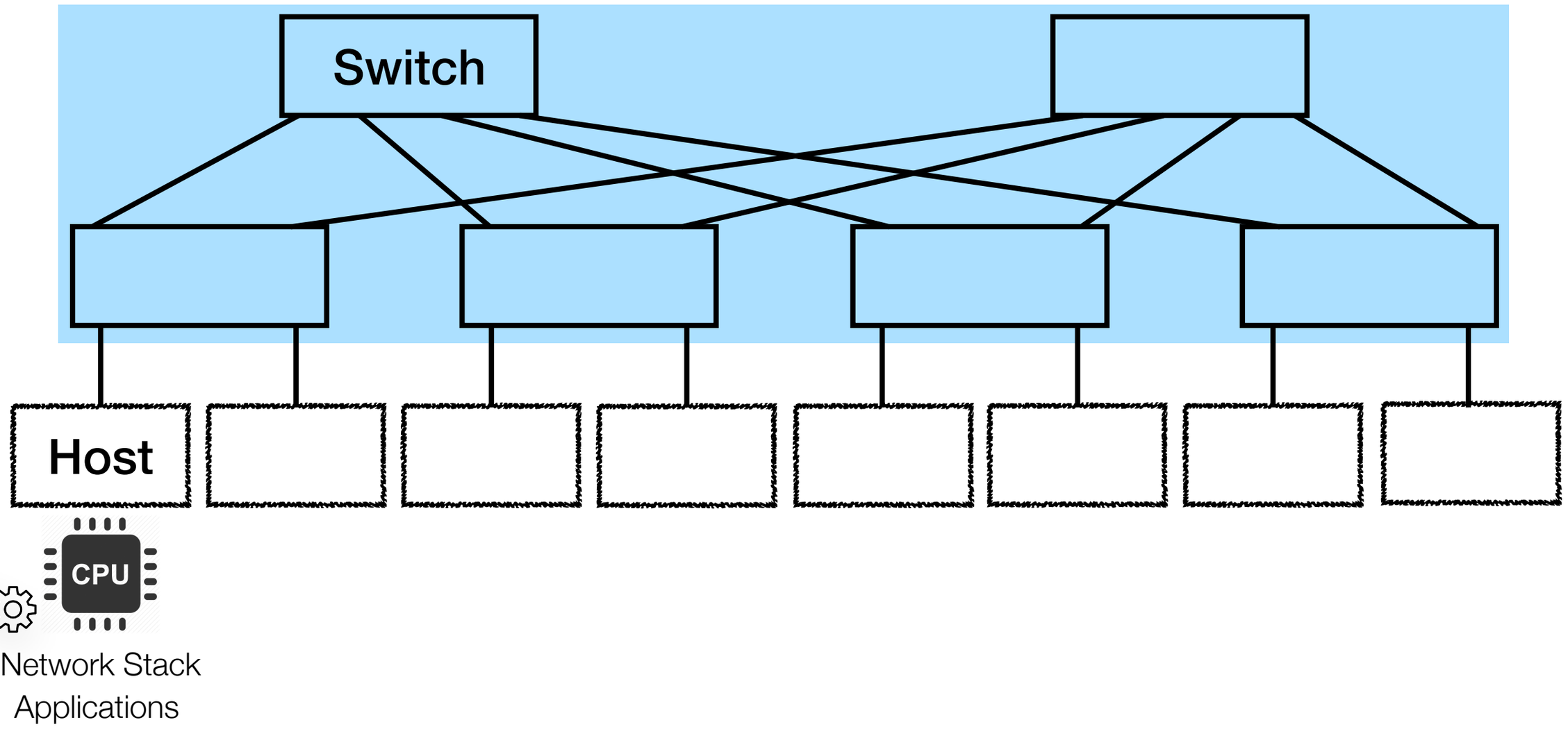
Inside a Typical Datacenter



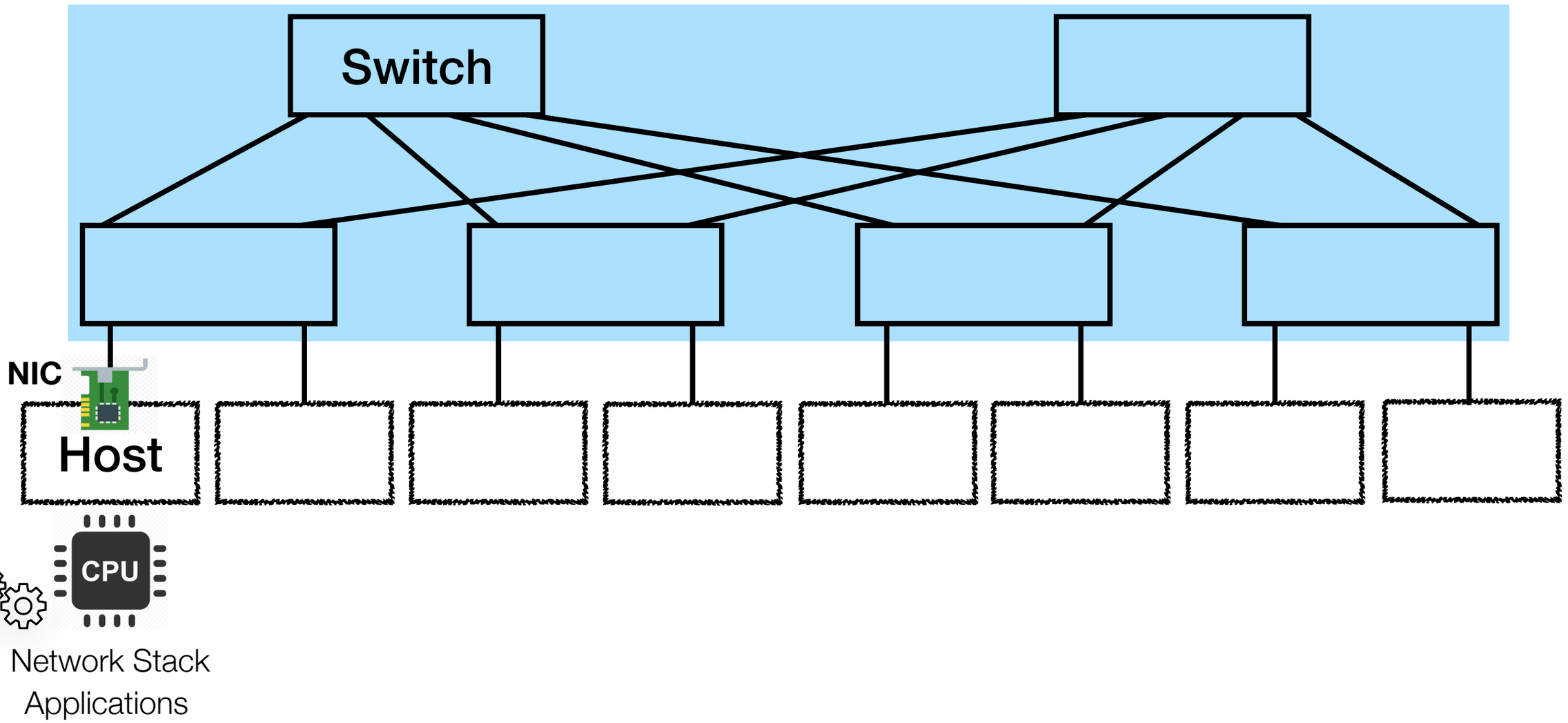
Inside a Typical Datacenter



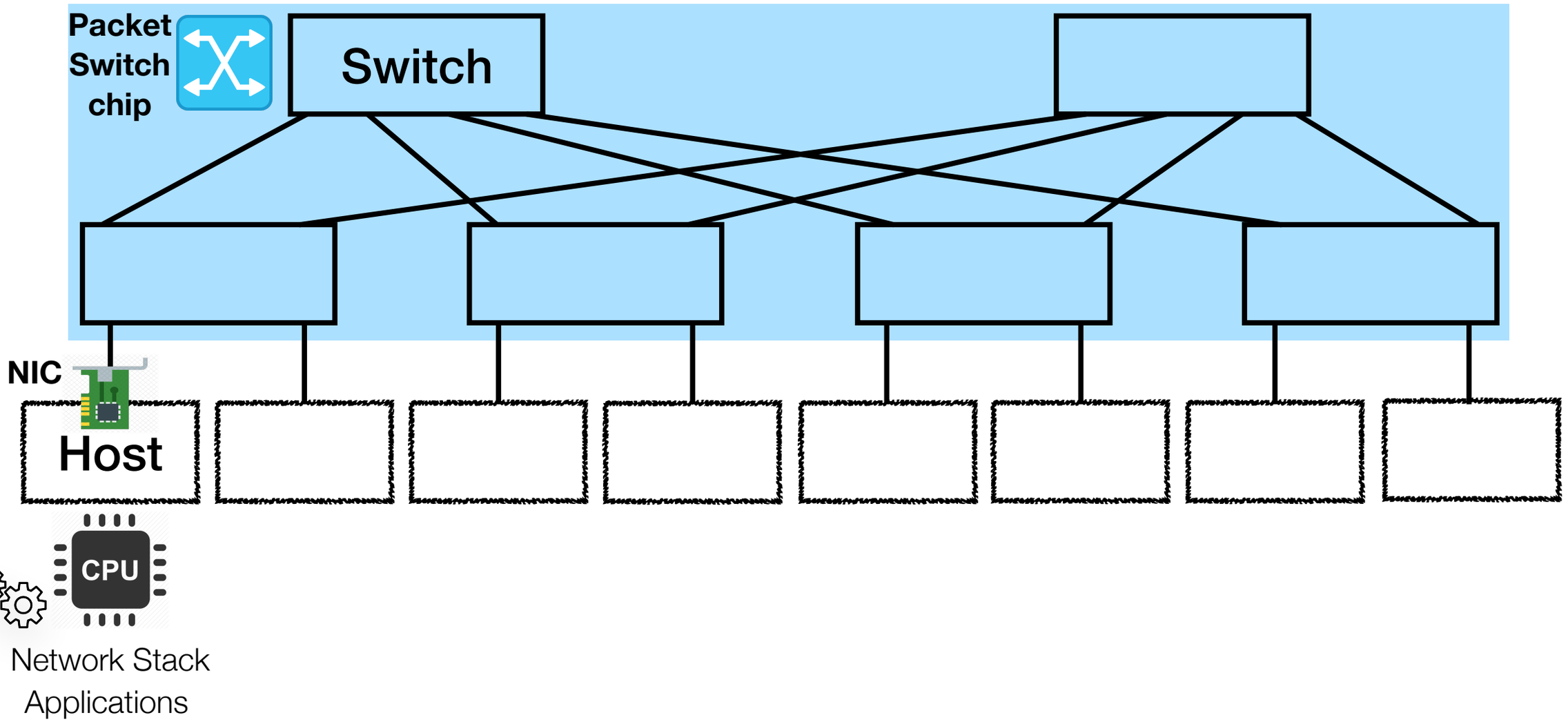
Inside a Typical Datacenter



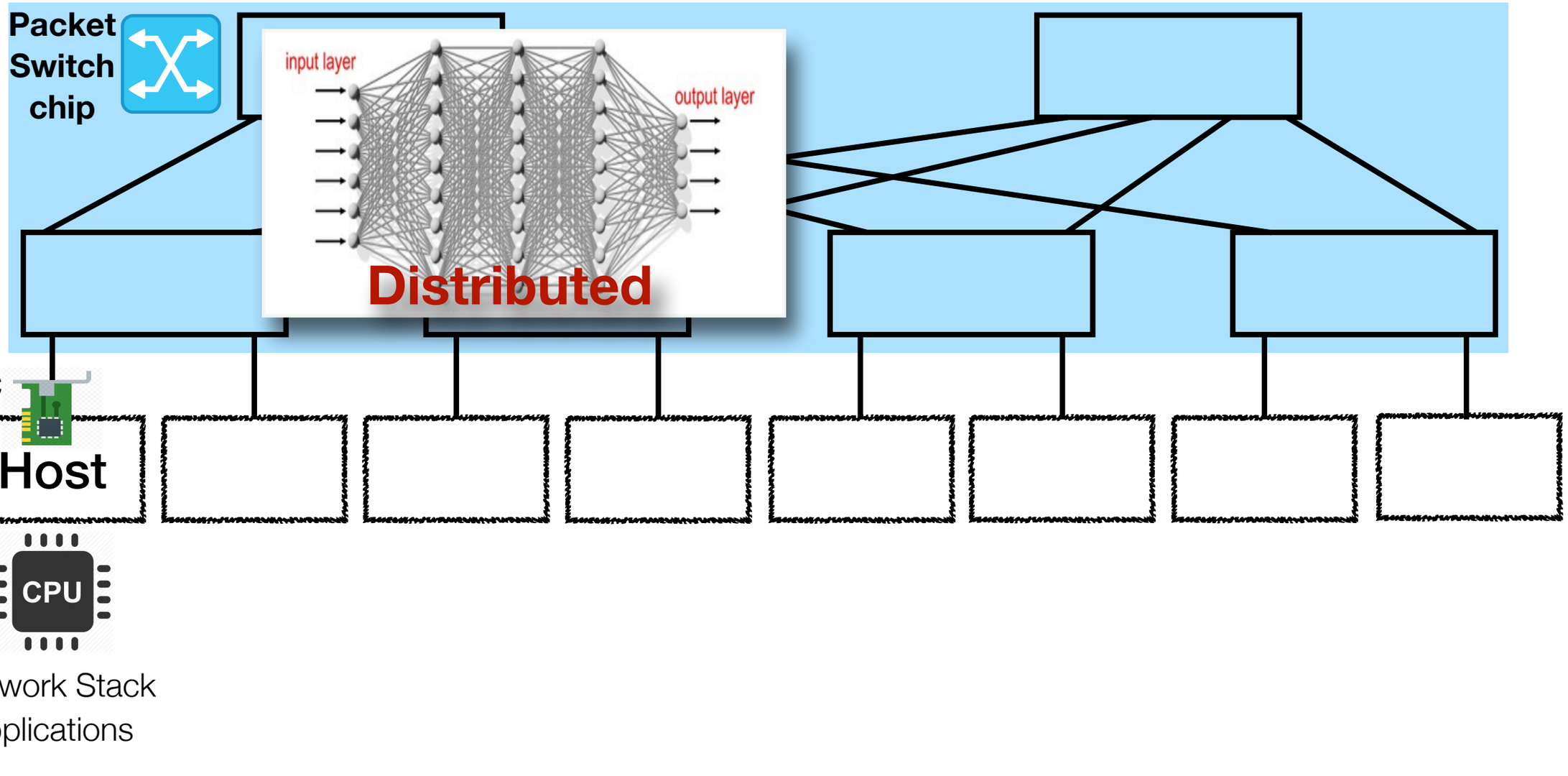
Inside a Typical Datacenter



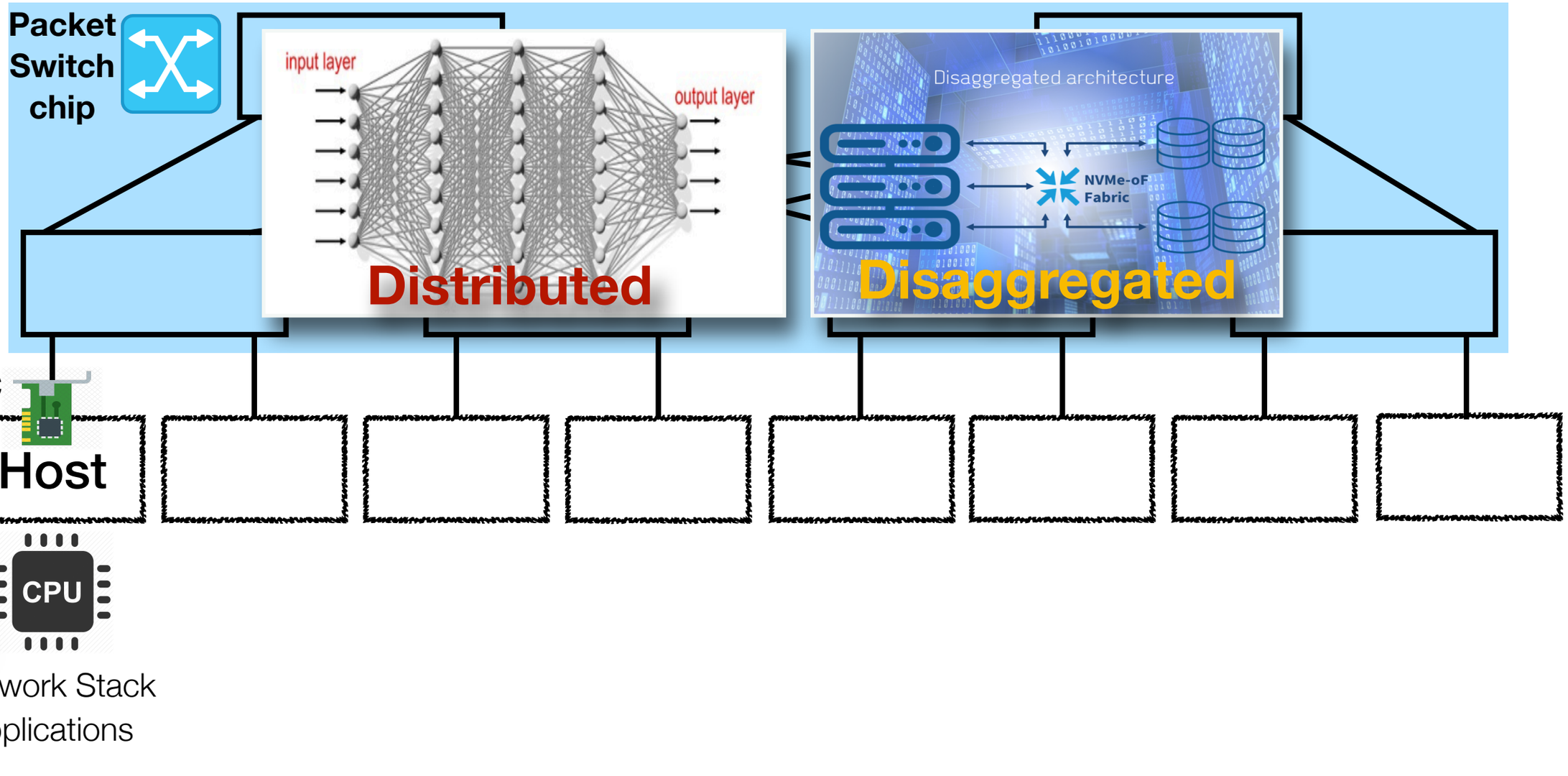
Inside a Typical Datacenter



Inside a Typical Datacenter



Inside a Typical Datacenter

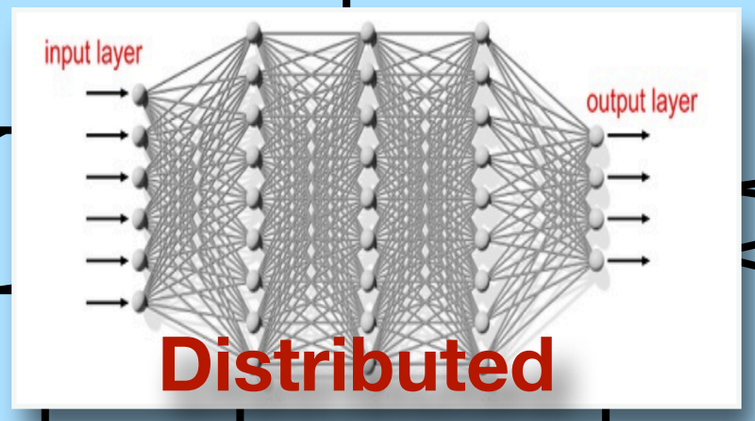


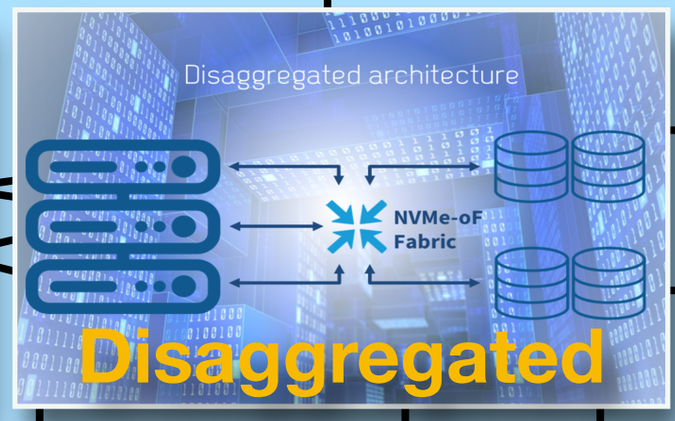
Inside a Typical Datacenter

Bandwidth demand increasing **2x / year**

[Source: Google]

Packet Switch chip





NIC

Host

CPU

Network Stack

Applications

Link rate increasing **400x / 10 years**

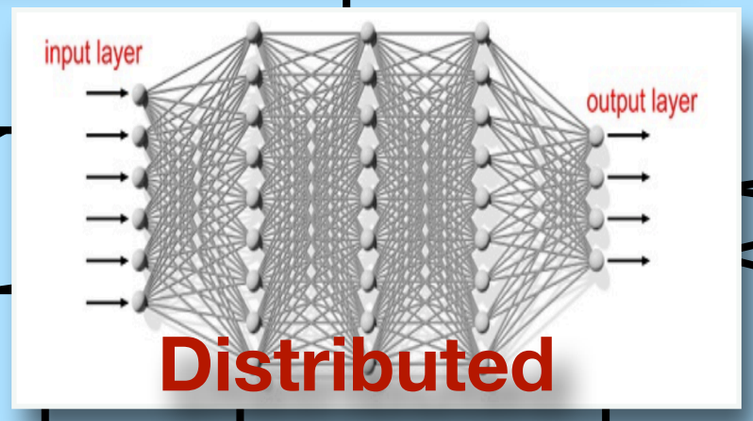
[Source: Microsoft]

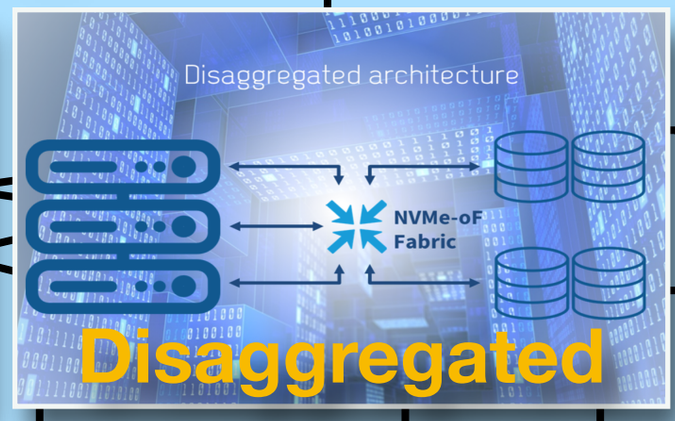
Inside a Typical Datacenter

Bandwidth demand increasing **2x / year**
[Source: Google]

Switch chip capacity increasing **2x / 2 years**
[Source: Broadcom]

Packet Switch chip





NIC

Host

CPU

Network Stack
Applications

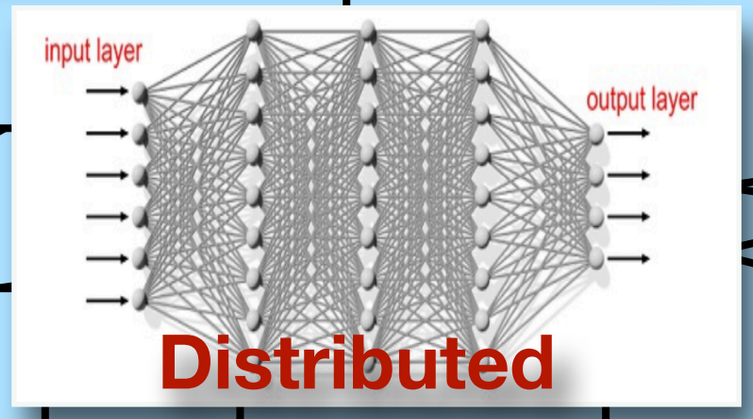
Link rate increasing **400x / 10 years**
[Source: Microsoft]

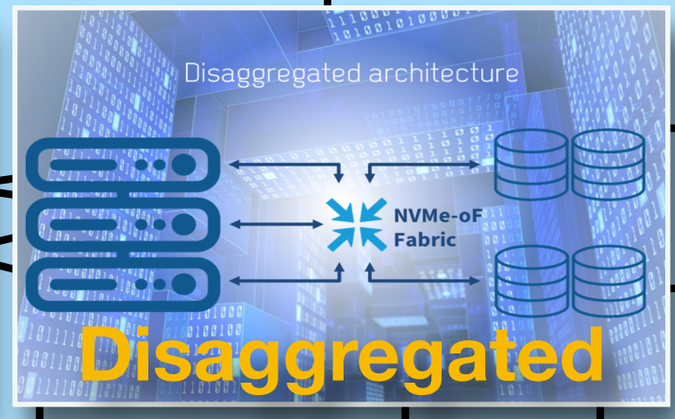
Inside a Typical Datacenter

Bandwidth demand increasing **2x / year**
[Source: Google]

Switch chip capacity increasing **2x / 2 years**
[Source: Broadcom]

Packet Switch chip





NIC

Host

CPU

Network Stack
Applications

Link rate increasing **400x / 10 years**
[Source: Microsoft]

CPU processing improving **2x / 20 years**
[Source: Hennessy & Patterson 6e]

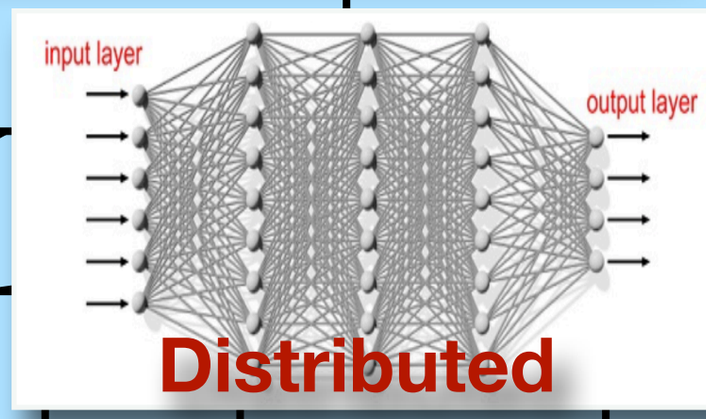
Inside a Typical Datacenter

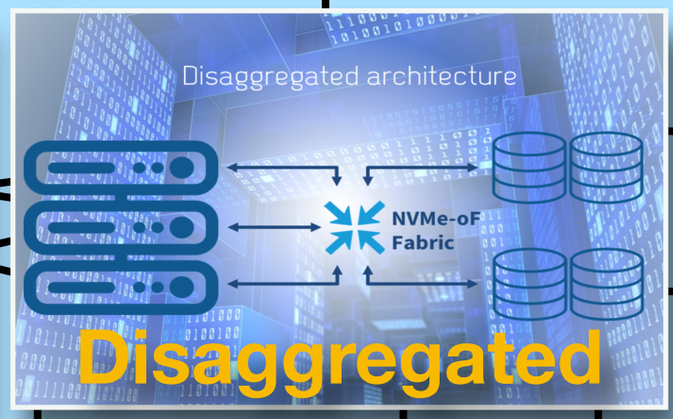
Bandwidth demand increasing **2x / year**
[Source: Google]

Switch chip capacity increasing **2x / 2 years**
[Source: Broadcom]

Slowdown in Moore's Law
End of Dennard scaling

Packet Switch chip





NIC

Host

Link rate increasing **400x / 10 years**
[Source: Microsoft]

CPU

Network Stack
Applications

CPU processing improving **2x / 20 years**
[Source: Hennessy & Patterson 6e]

Inside a Typical Datacenter

Bandwidth demand increasing **2x / year**
[Source: Google]

Switch chip capacity increasing **2x / 2 years**
[Source: Broadcom]

Slowdown in Moore's Law
End of Dennard scaling

Packet Switch chip

Switch

Distributed

Disaggregated

NIC

Host

CPU

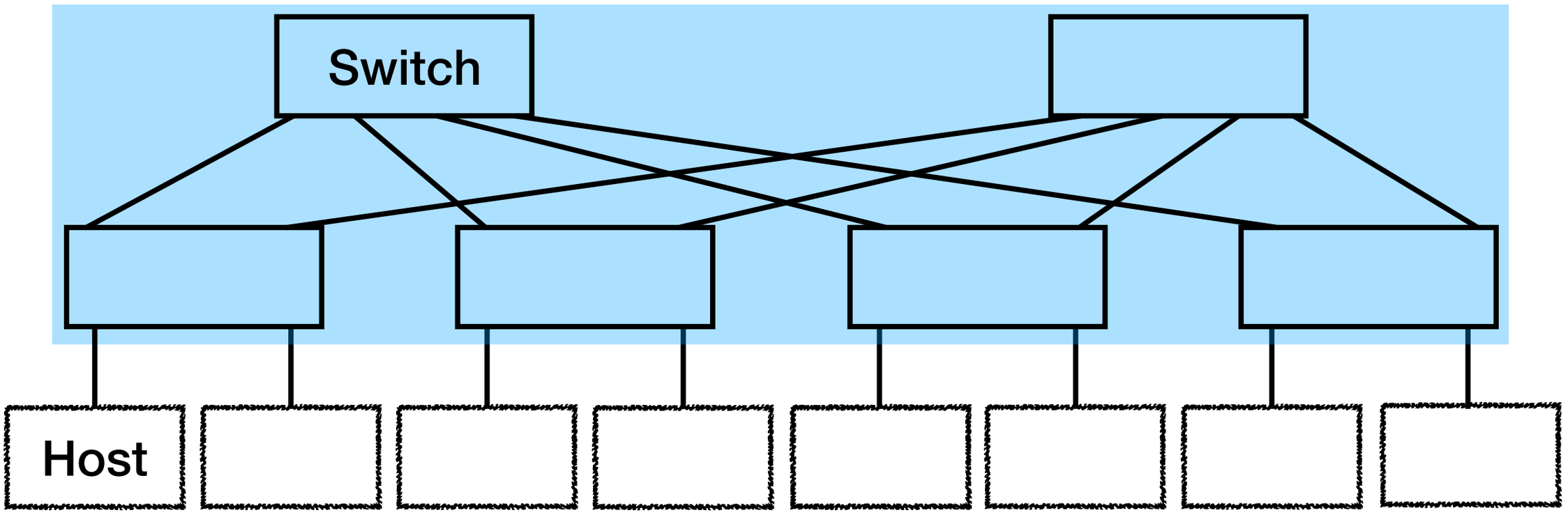
Network Stack

Applications

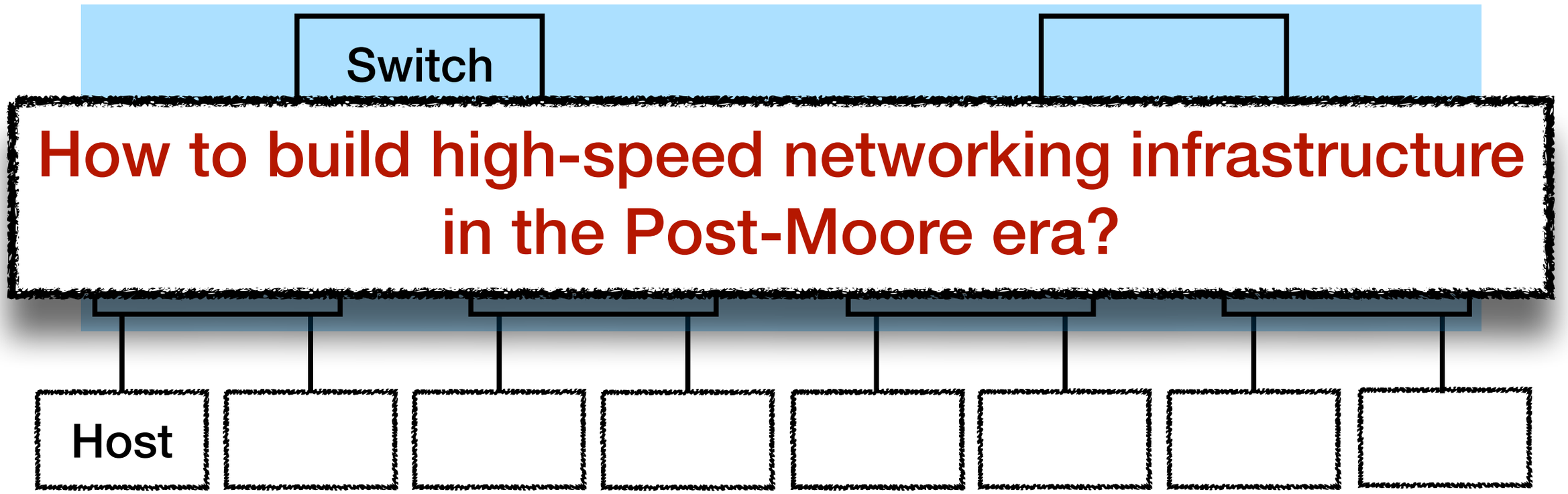
Link rate increasing **400x / 10 years**
[Source: Microsoft]

CPU processing improving **2x / 20 years**
[Source: Hennessy & Patterson 6e]

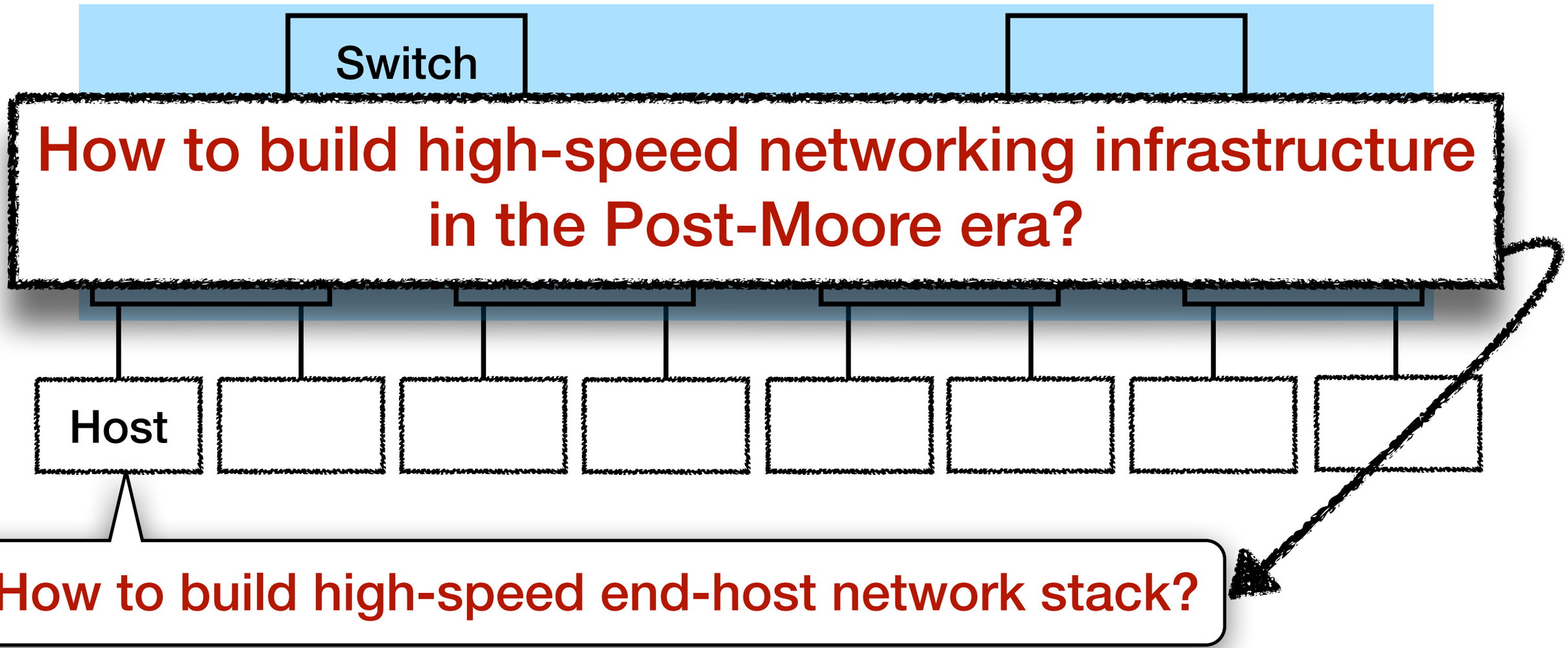
Big Research Question



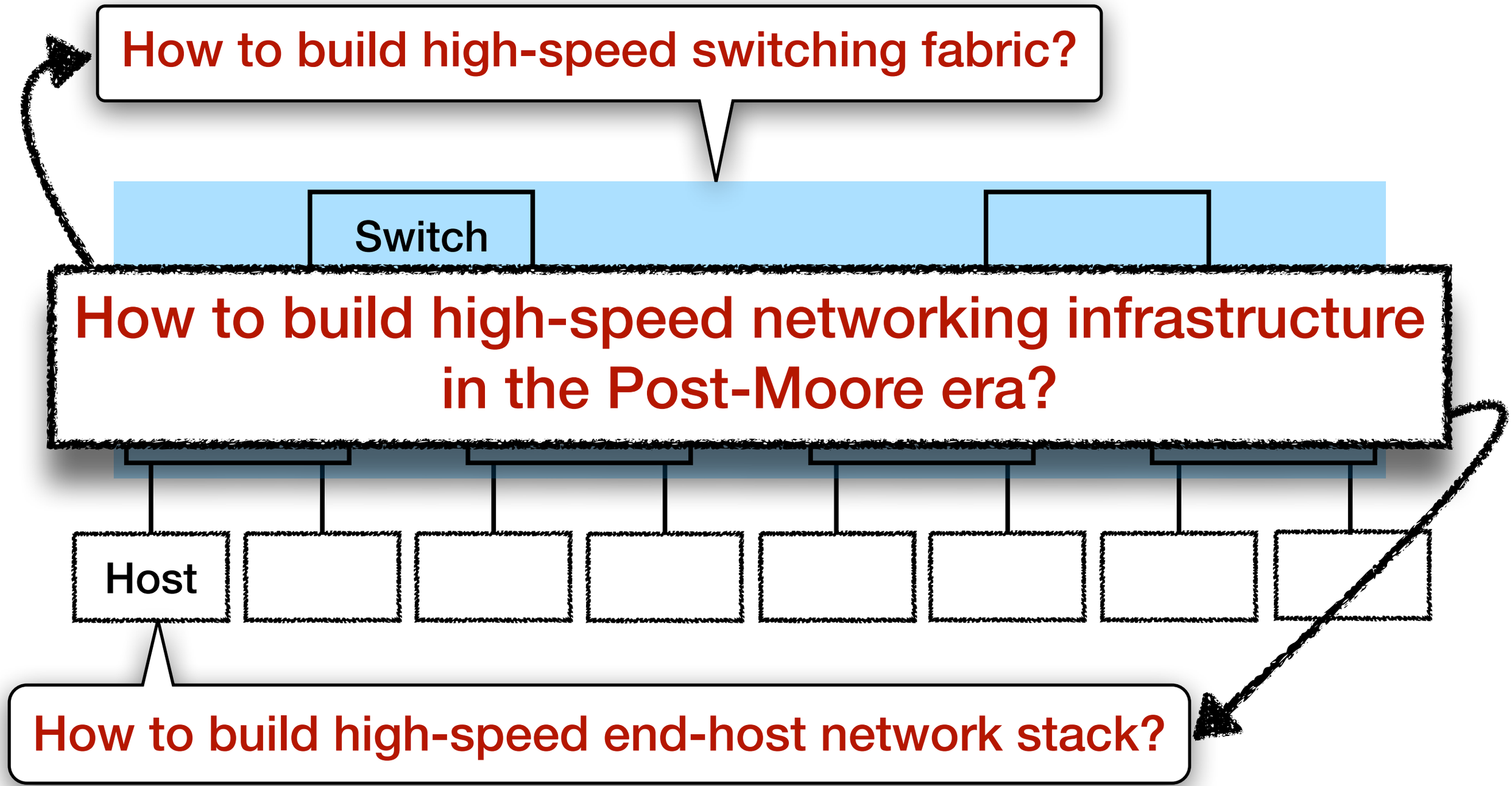
Big Research Question



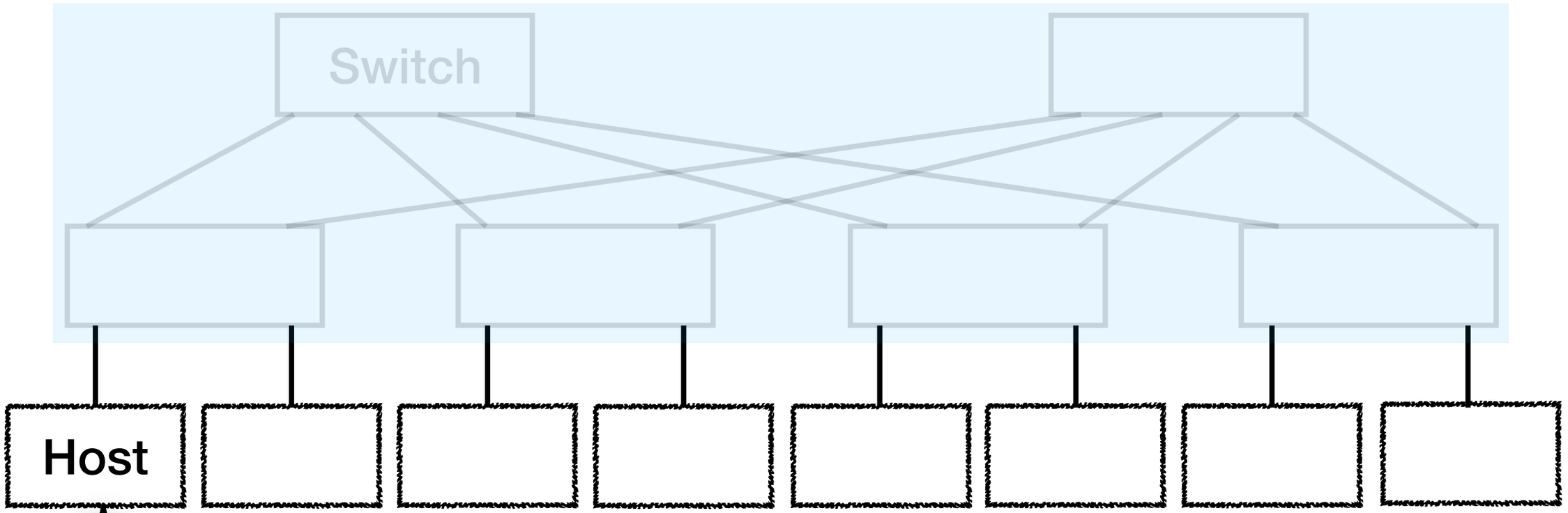
Big Research Question



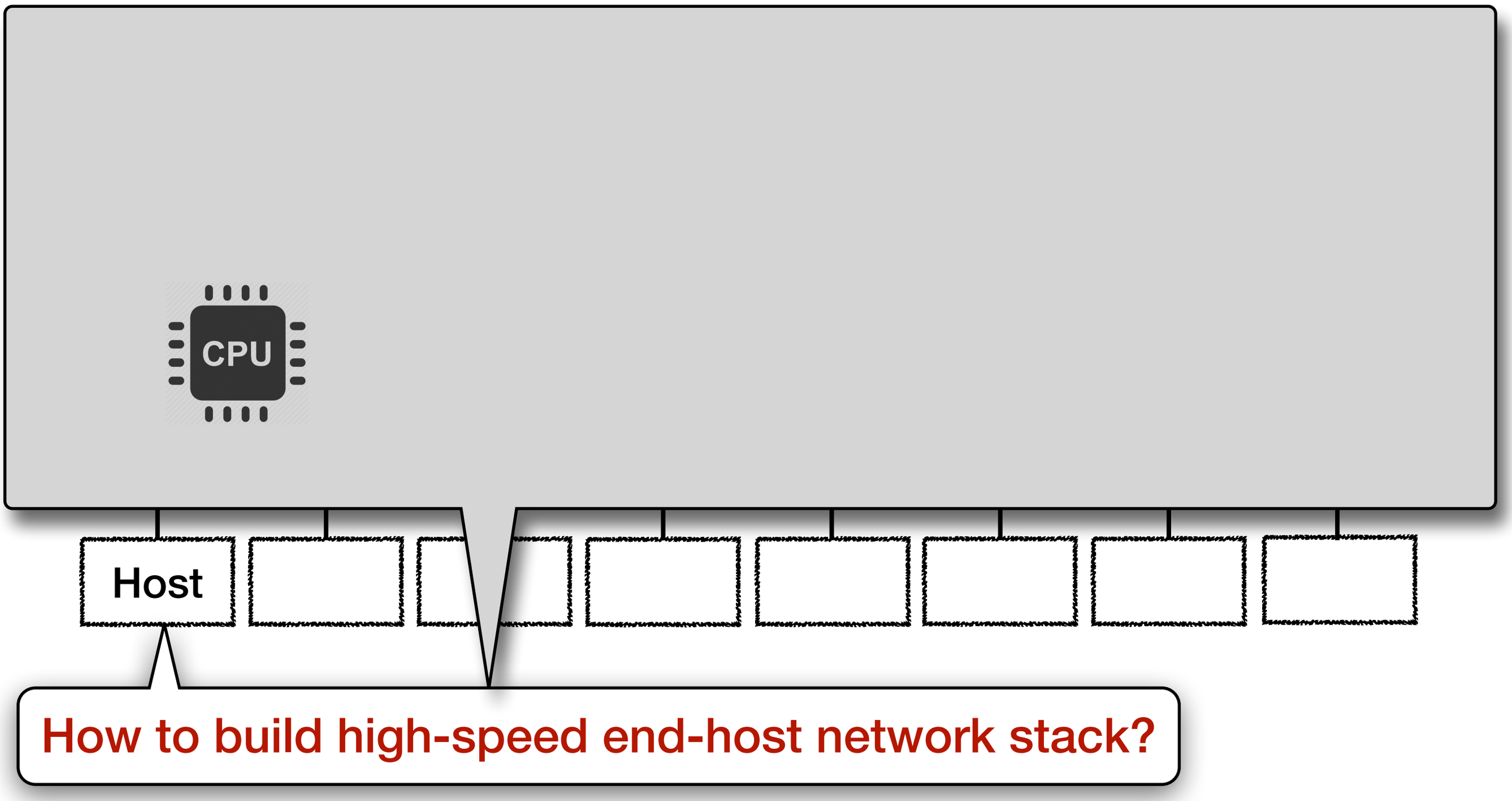
Big Research Question



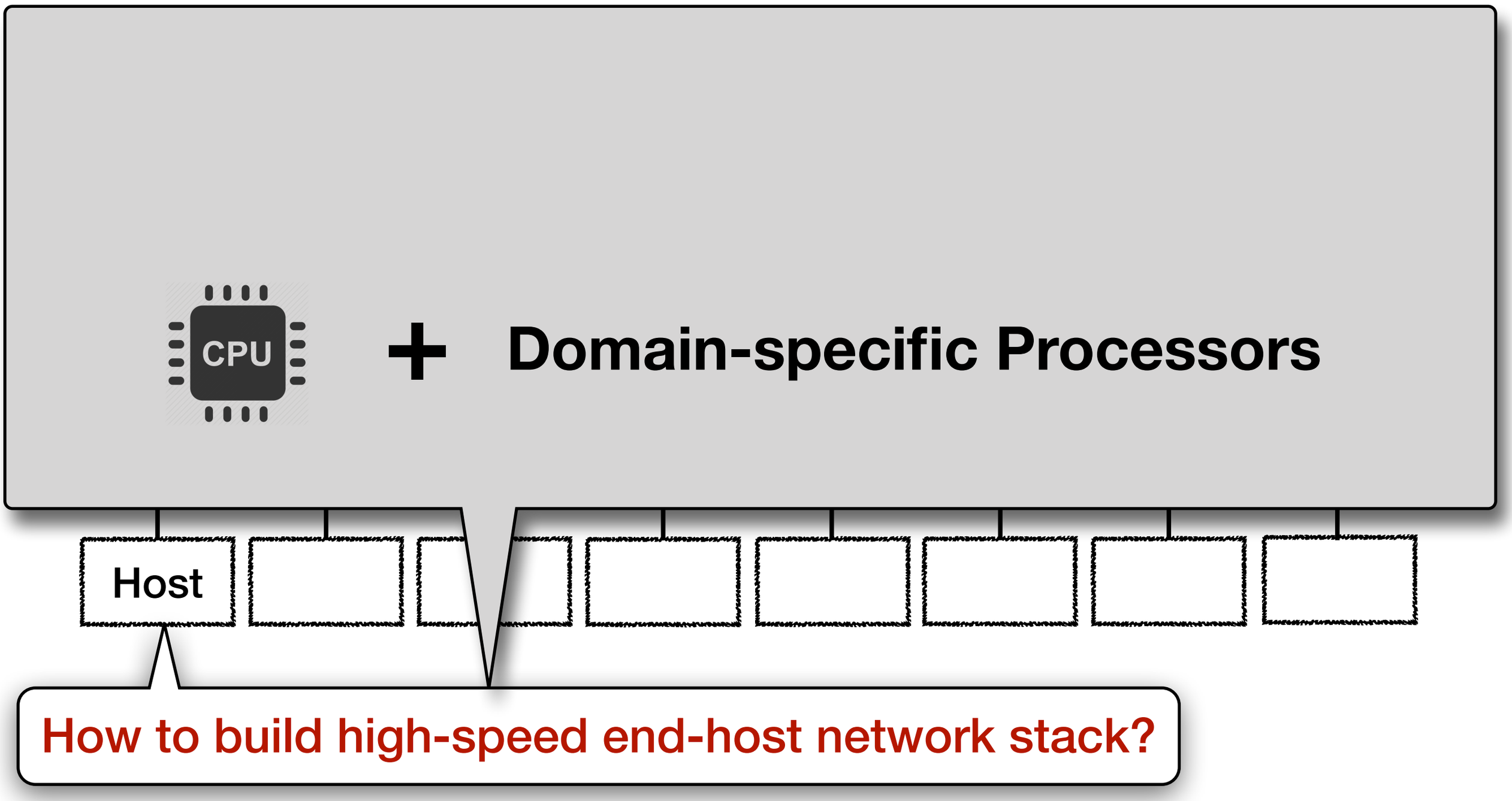
Part I : End-host Network Stack



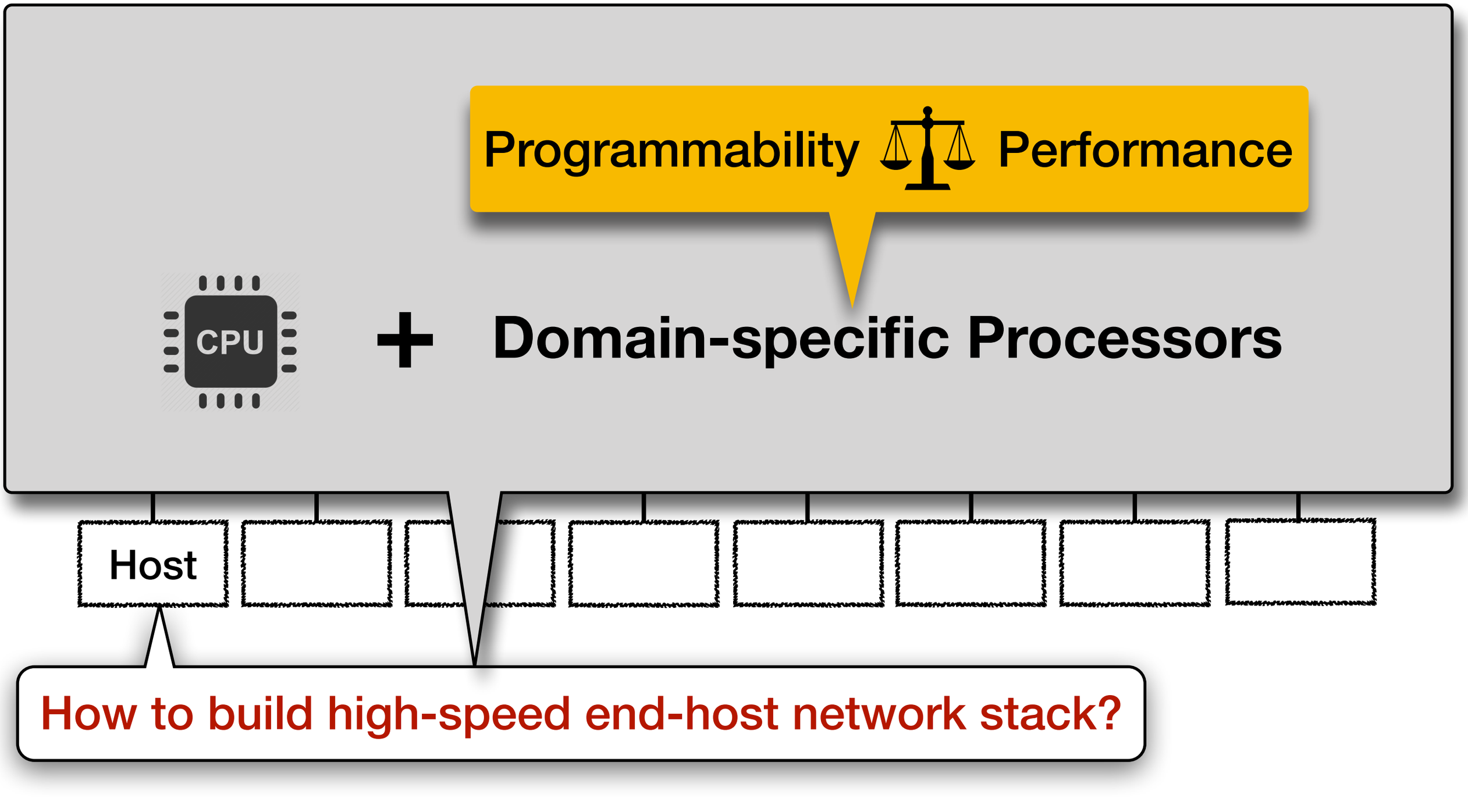
Part I : End-host Network Stack



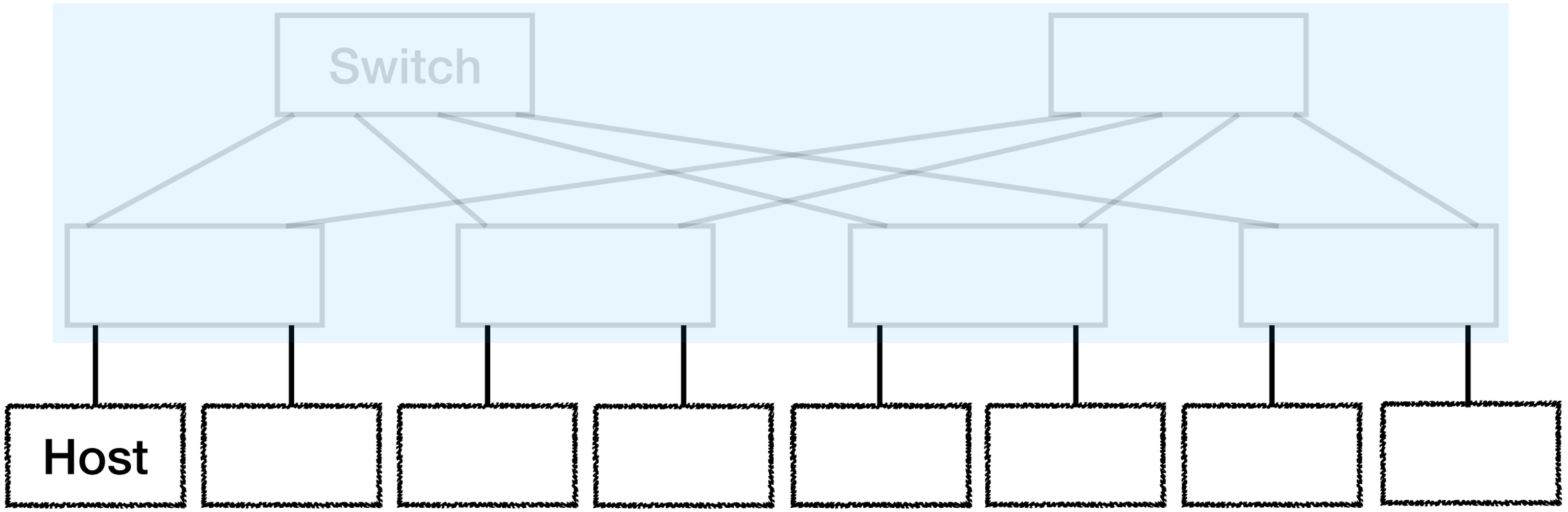
Part I : End-host Network Stack



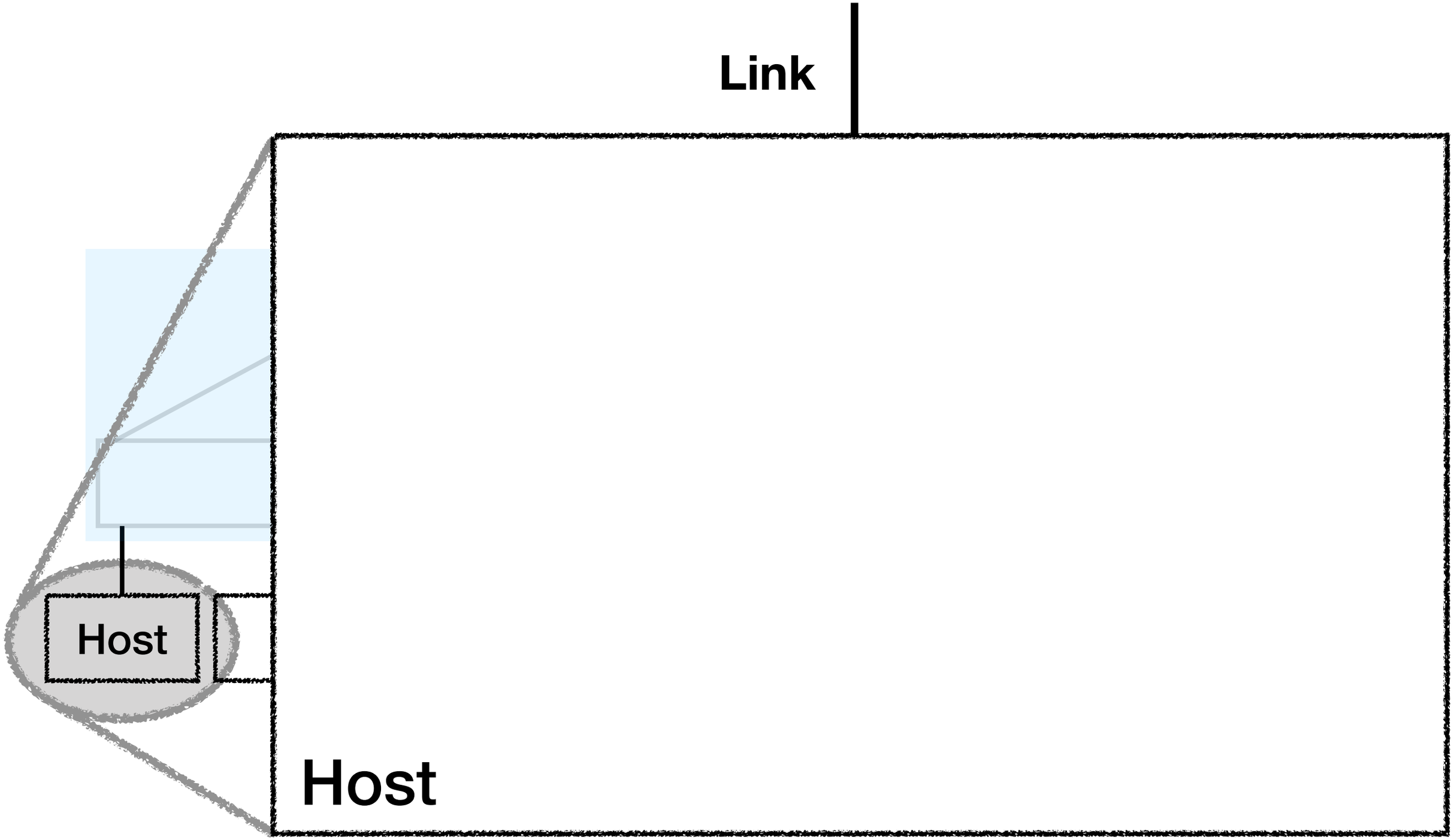
Part I : End-host Network Stack



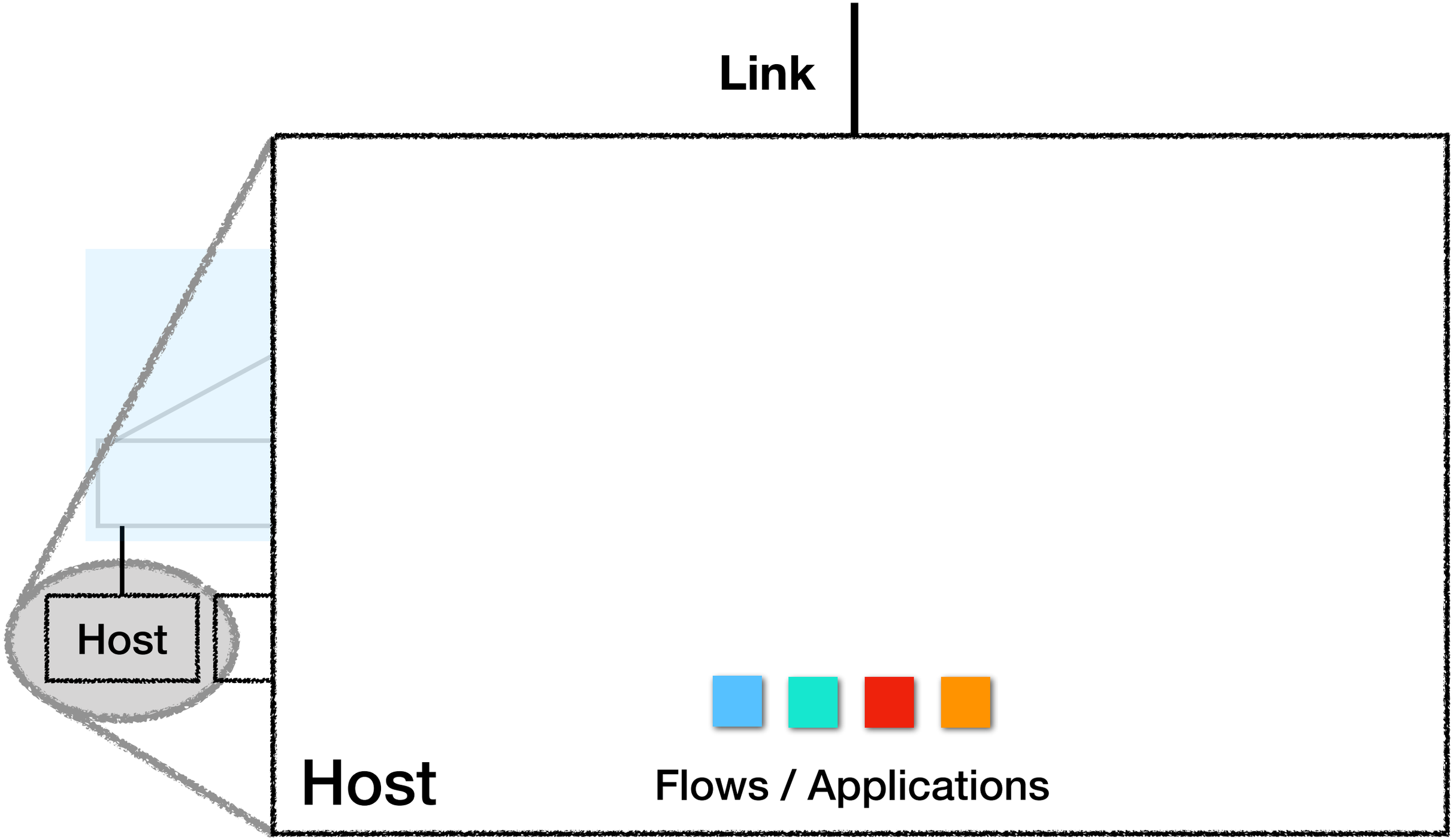
End-host Network Stack Operation : Packet Scheduling



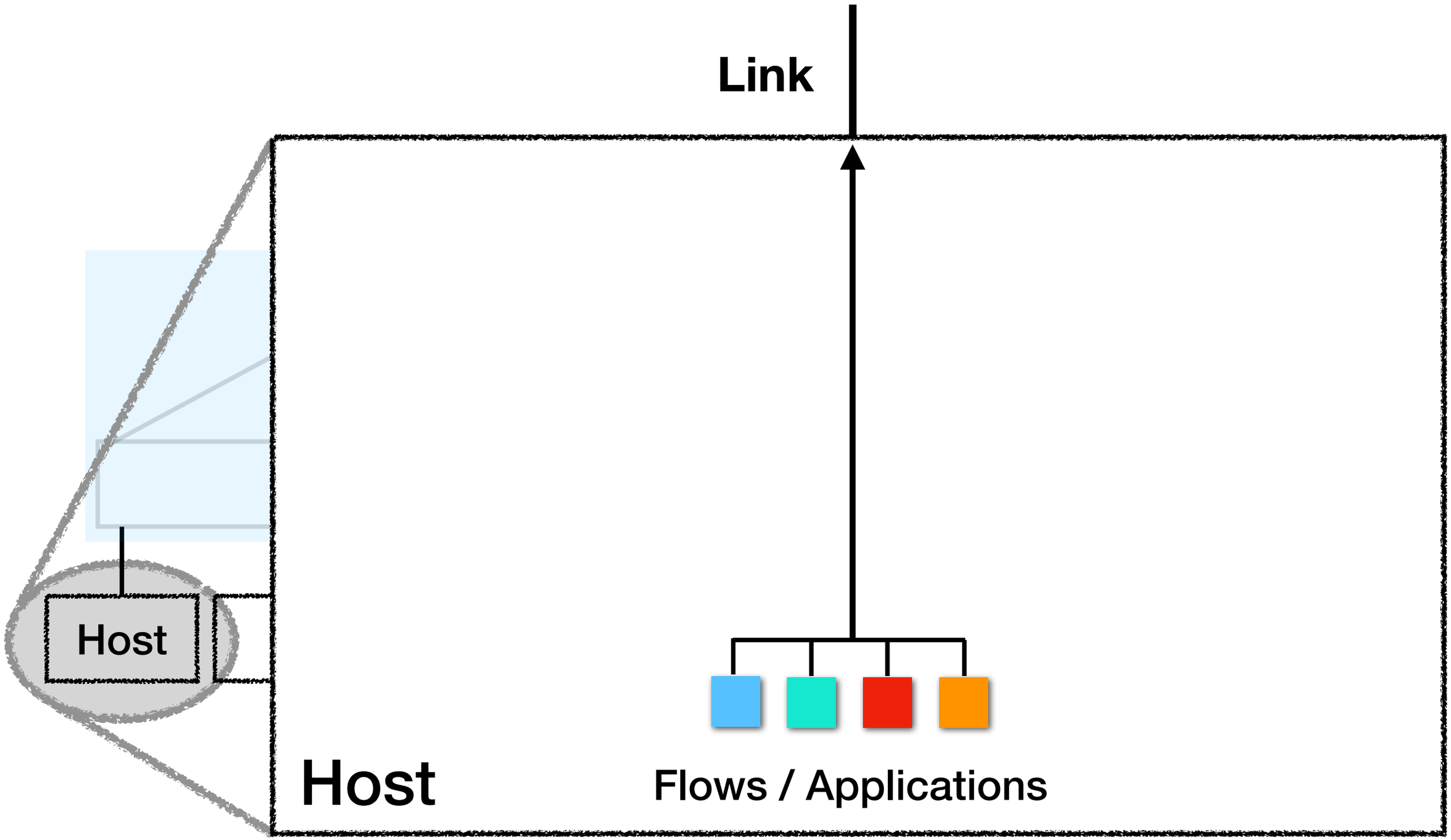
End-host Network Stack Operation : Packet Scheduling



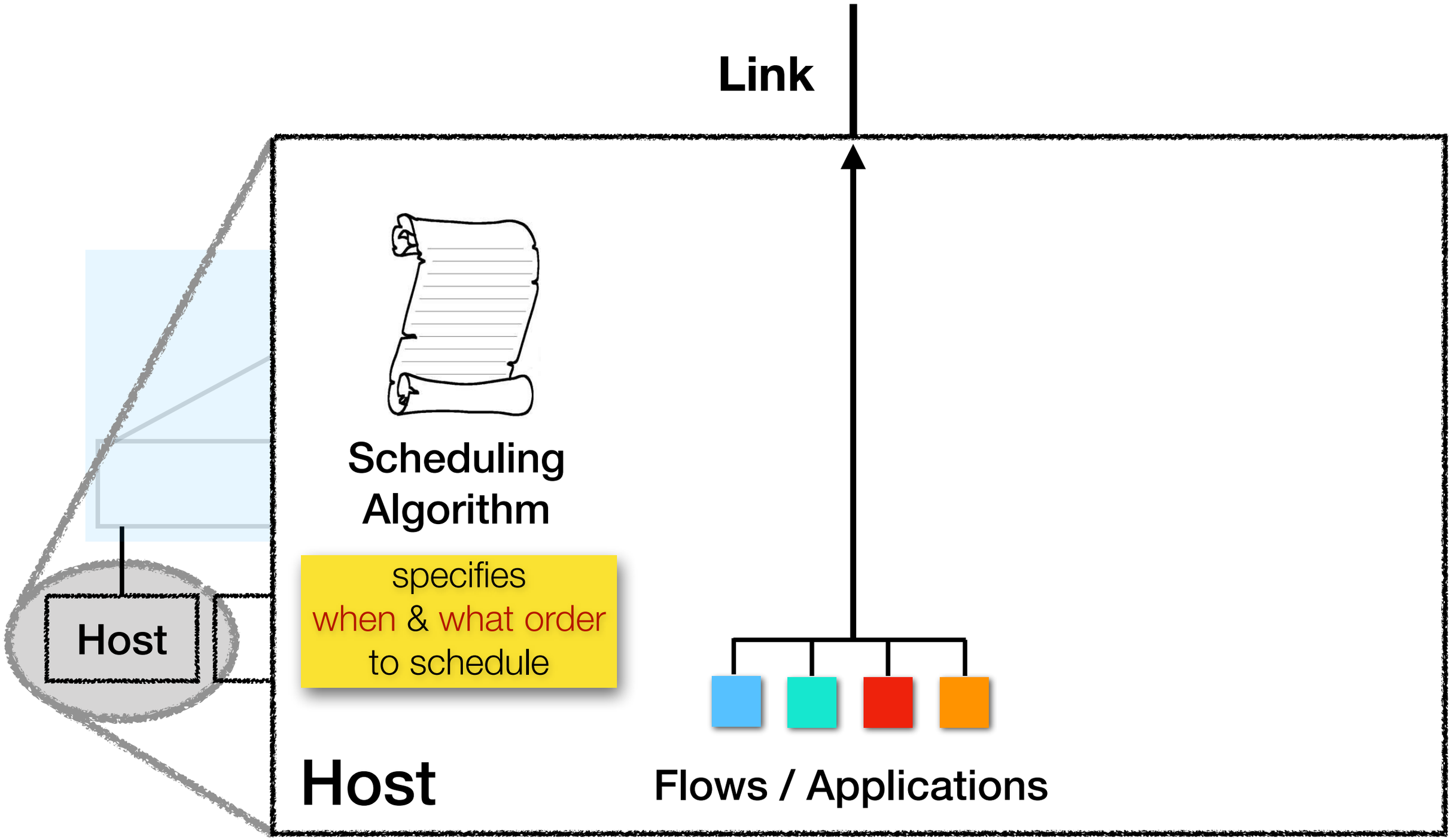
End-host Network Stack Operation : Packet Scheduling



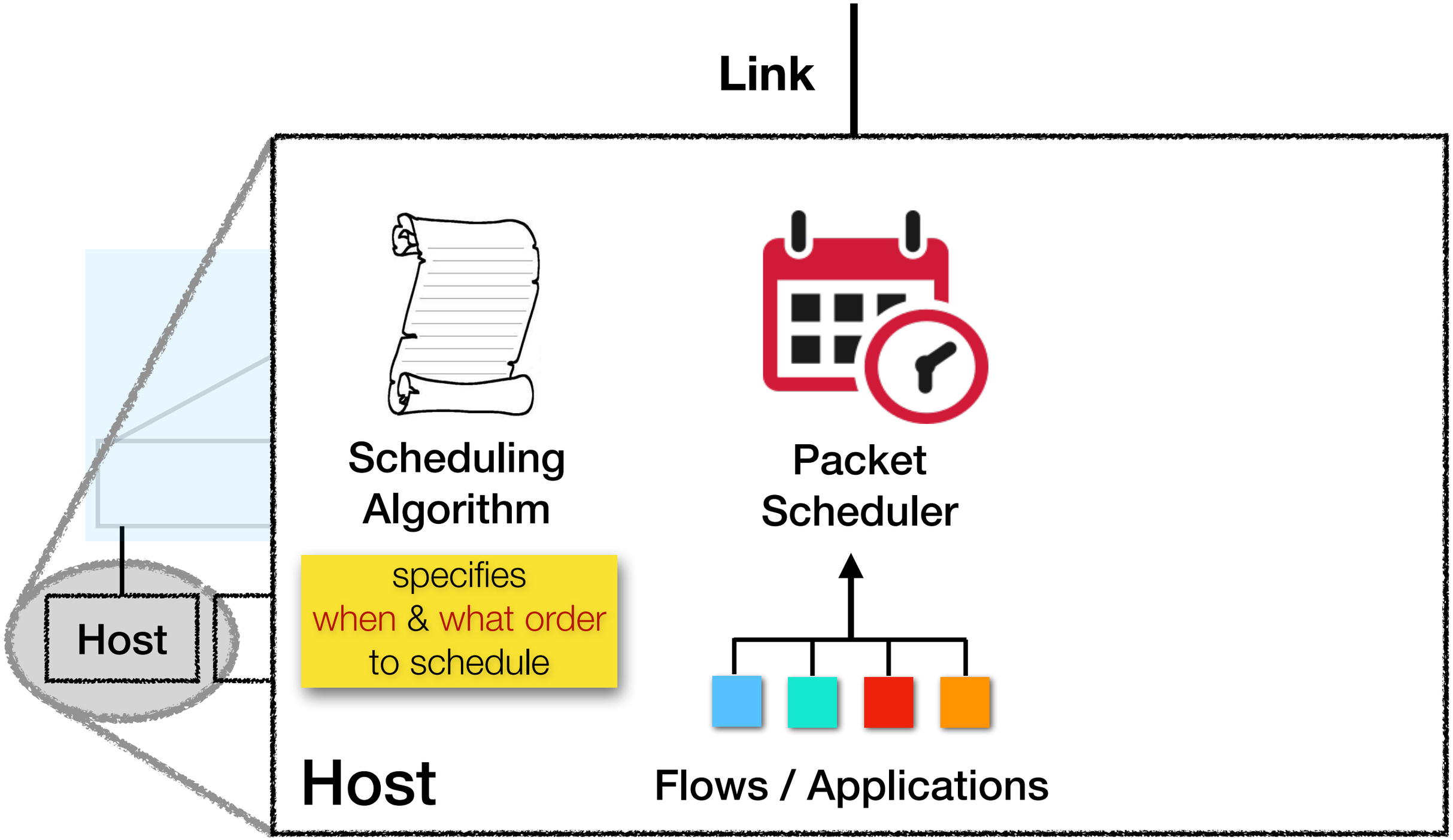
End-host Network Stack Operation : Packet Scheduling



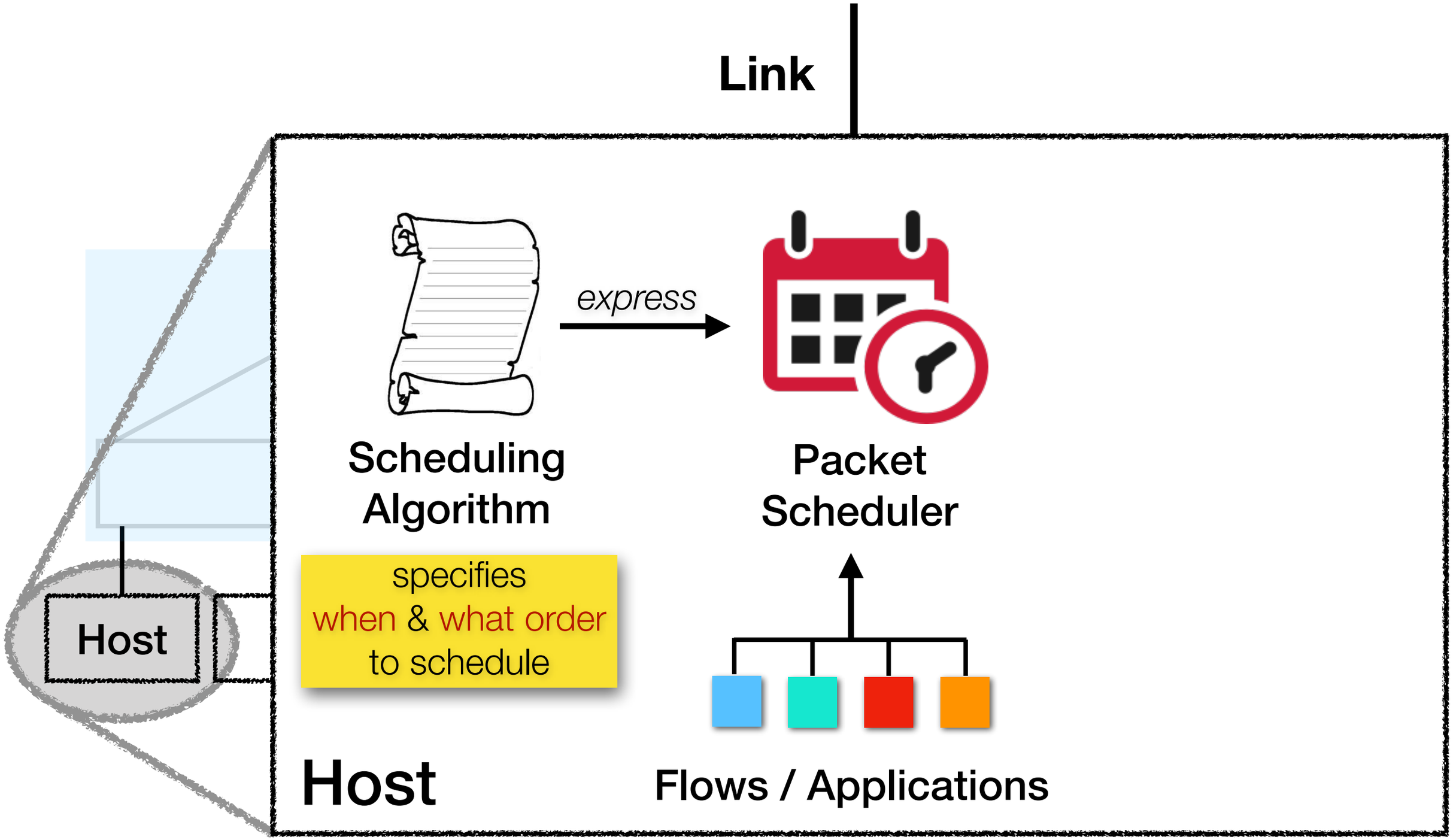
End-host Network Stack Operation : Packet Scheduling



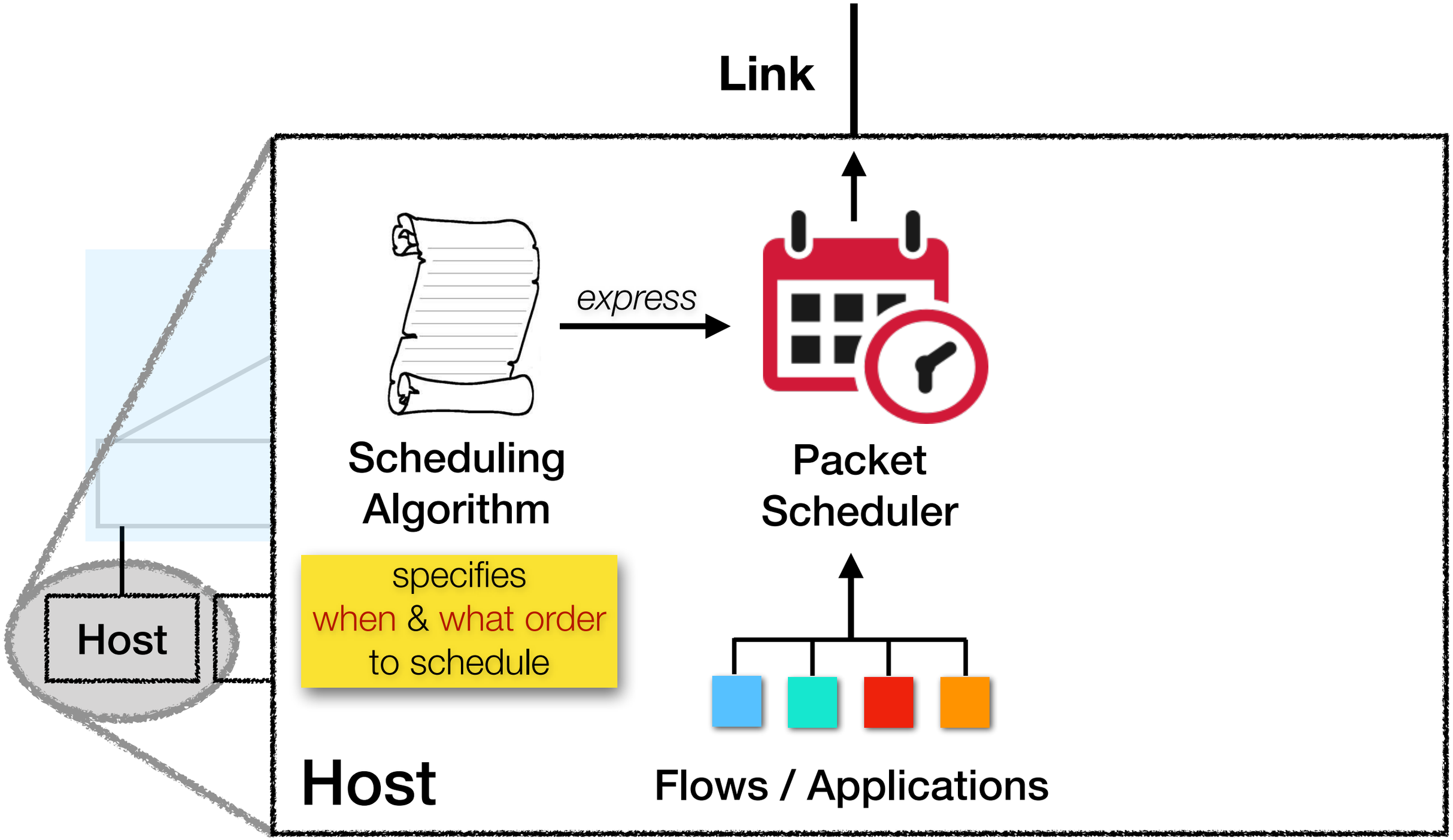
End-host Network Stack Operation : Packet Scheduling



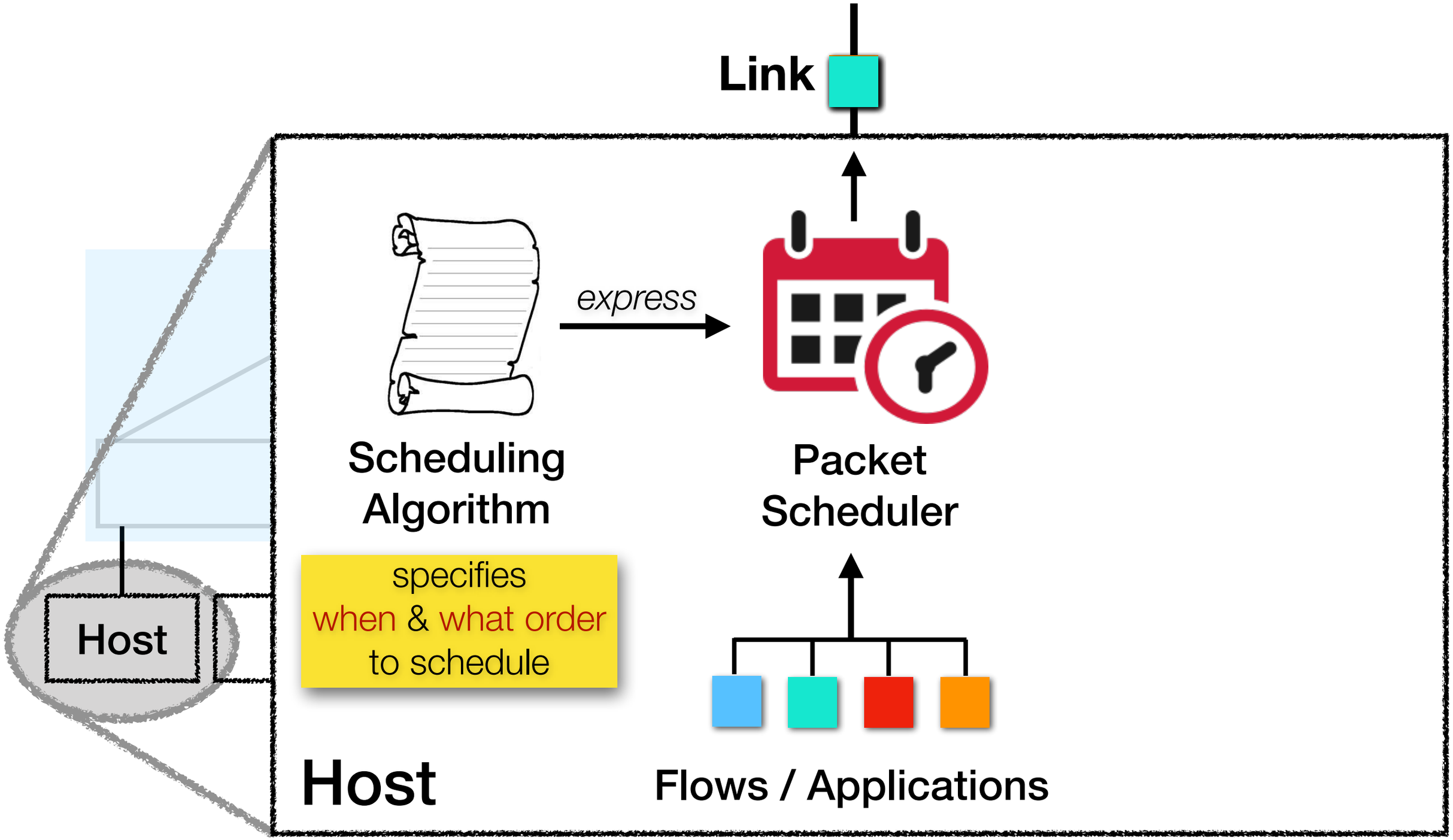
End-host Network Stack Operation : Packet Scheduling



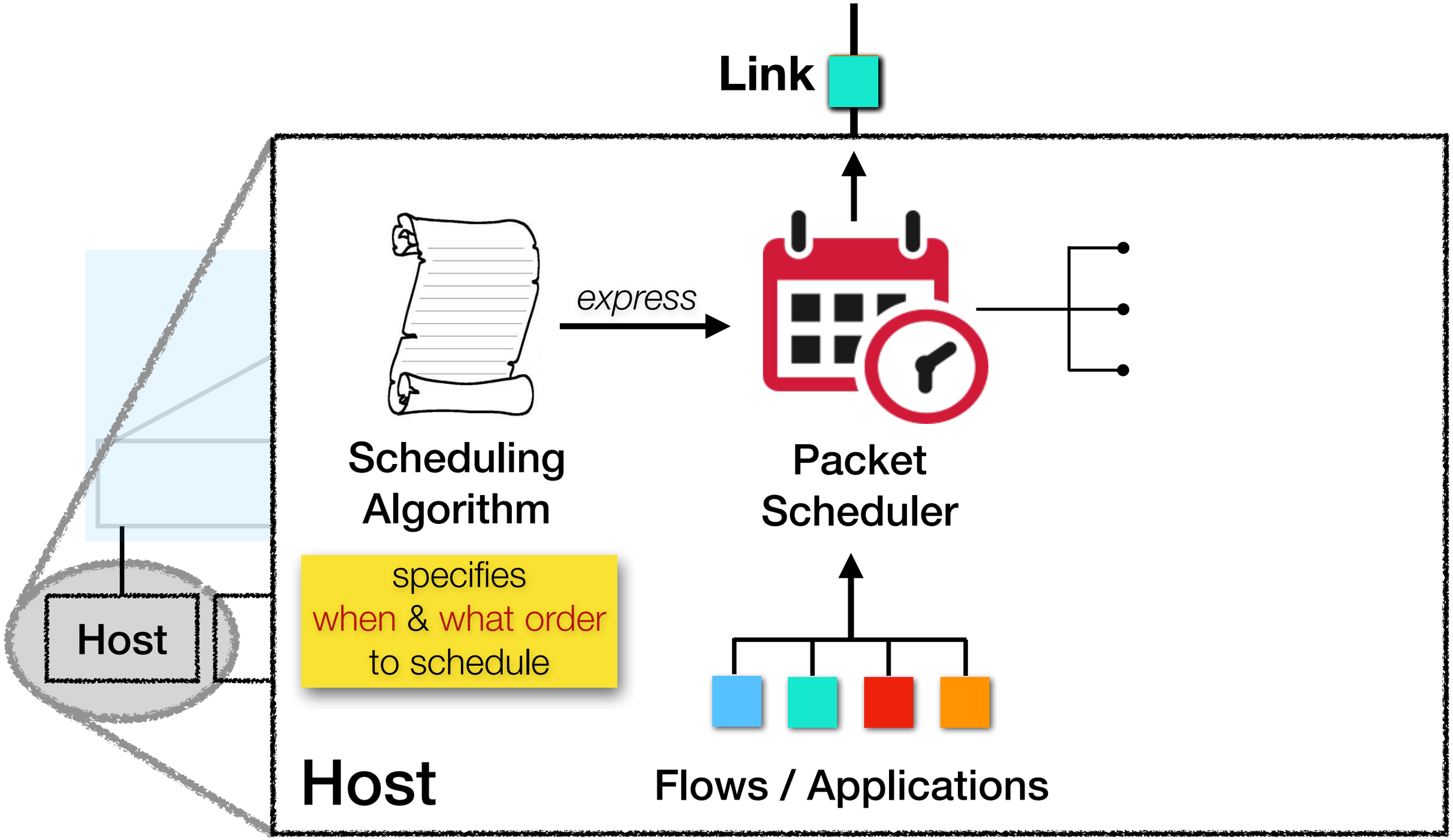
End-host Network Stack Operation : Packet Scheduling



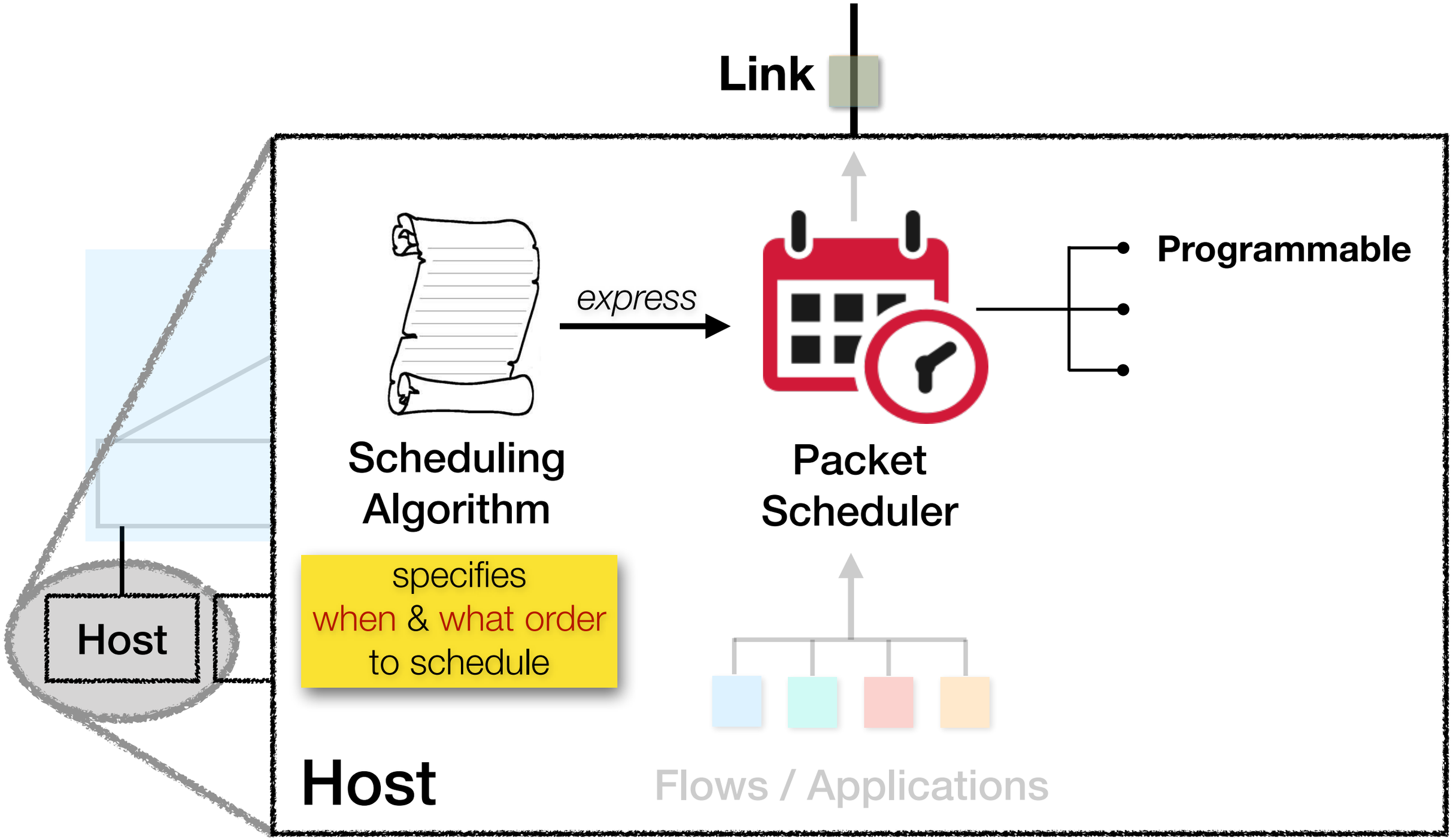
End-host Network Stack Operation : Packet Scheduling



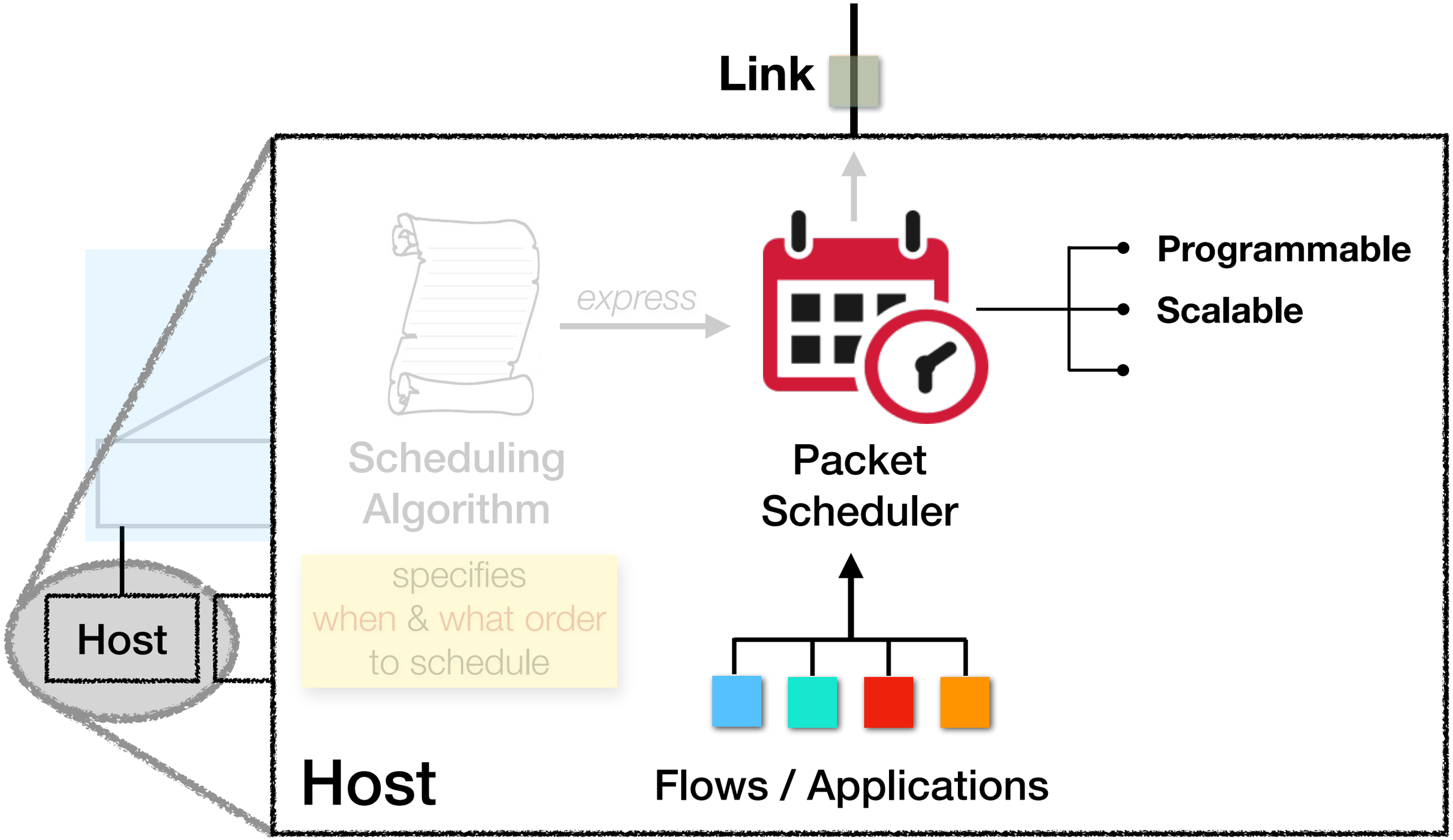
End-host Network Stack Operation : Packet Scheduling



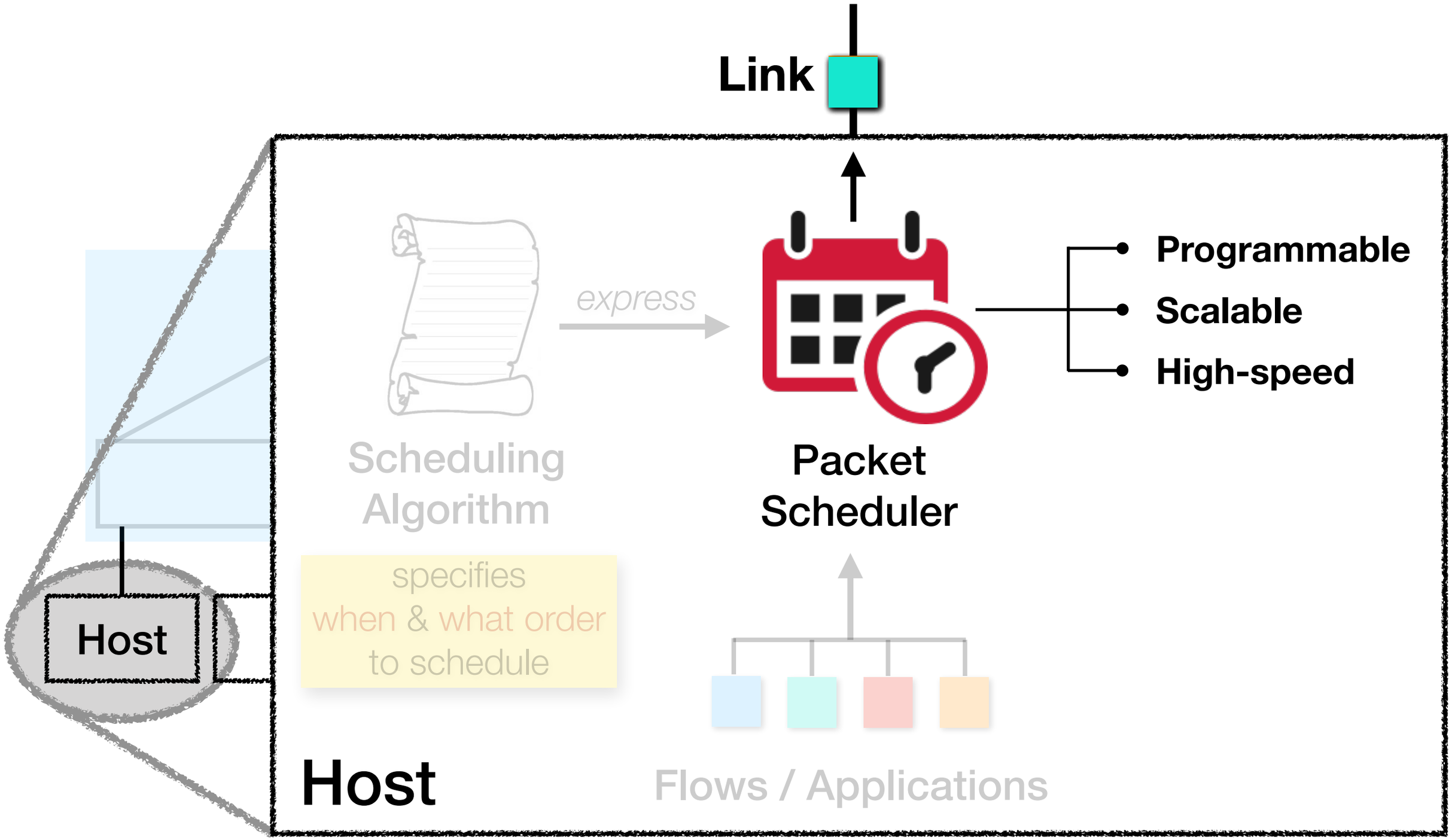
End-host Network Stack Operation : Packet Scheduling



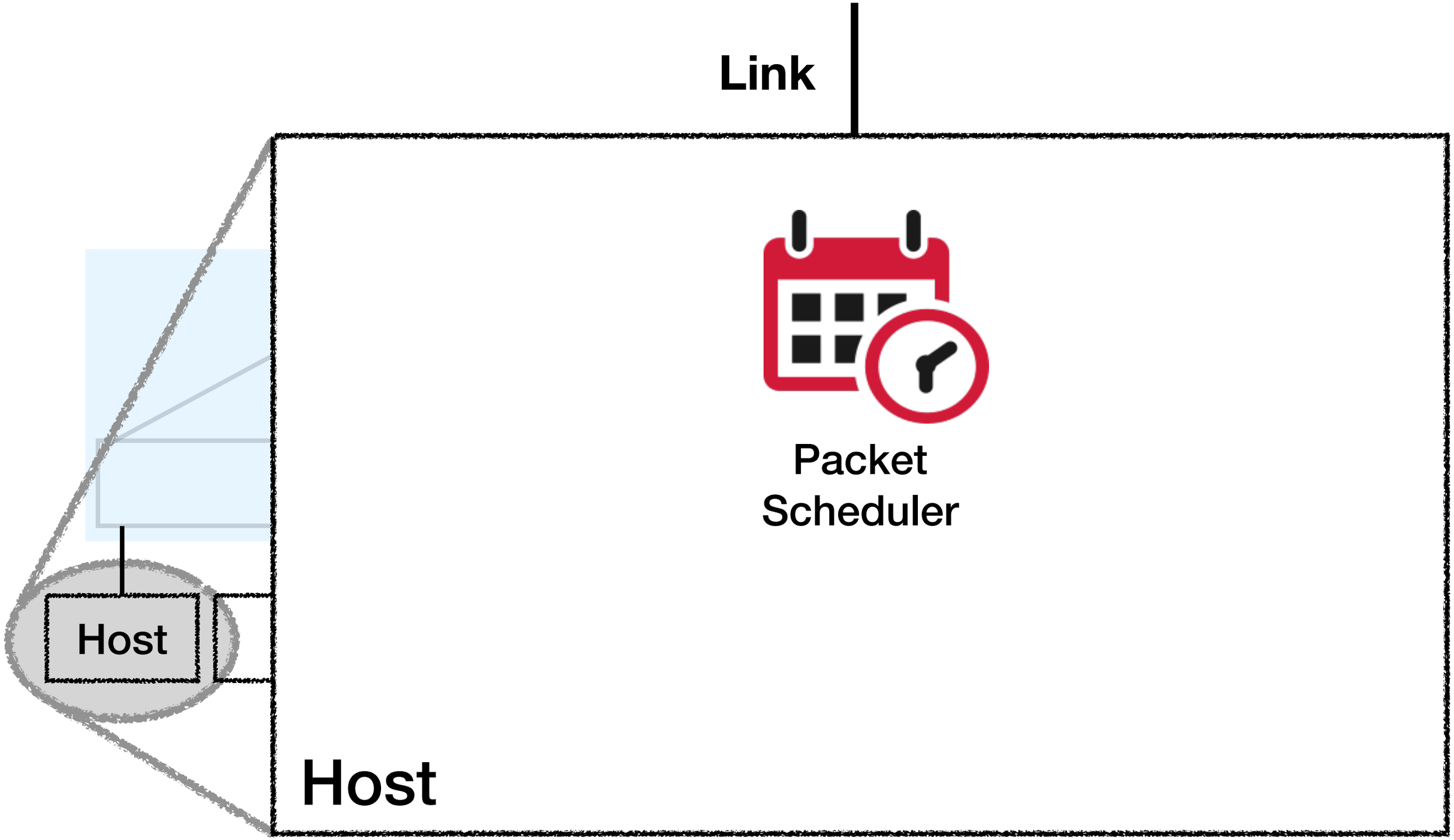
End-host Network Stack Operation : Packet Scheduling



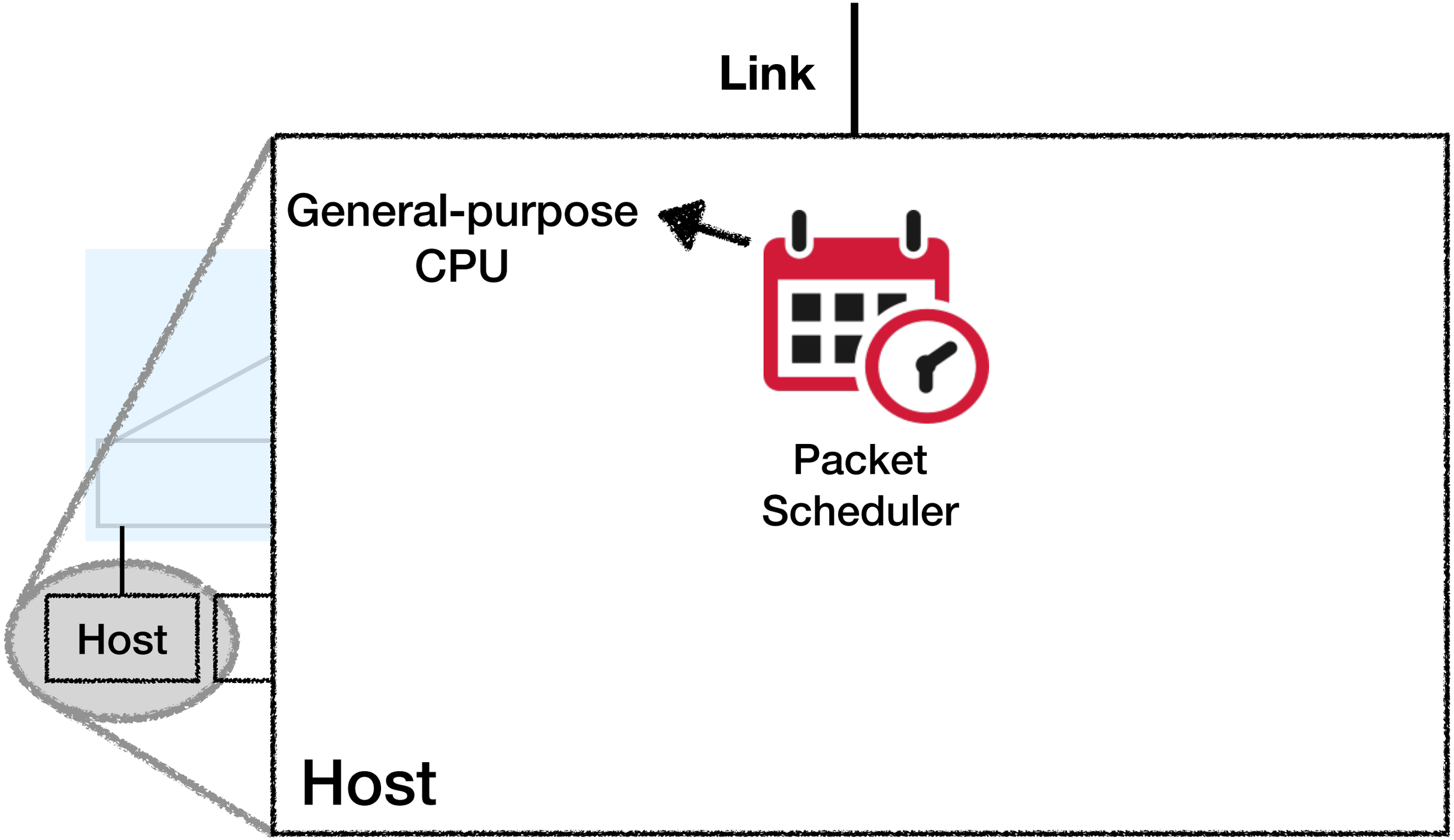
End-host Network Stack Operation : Packet Scheduling



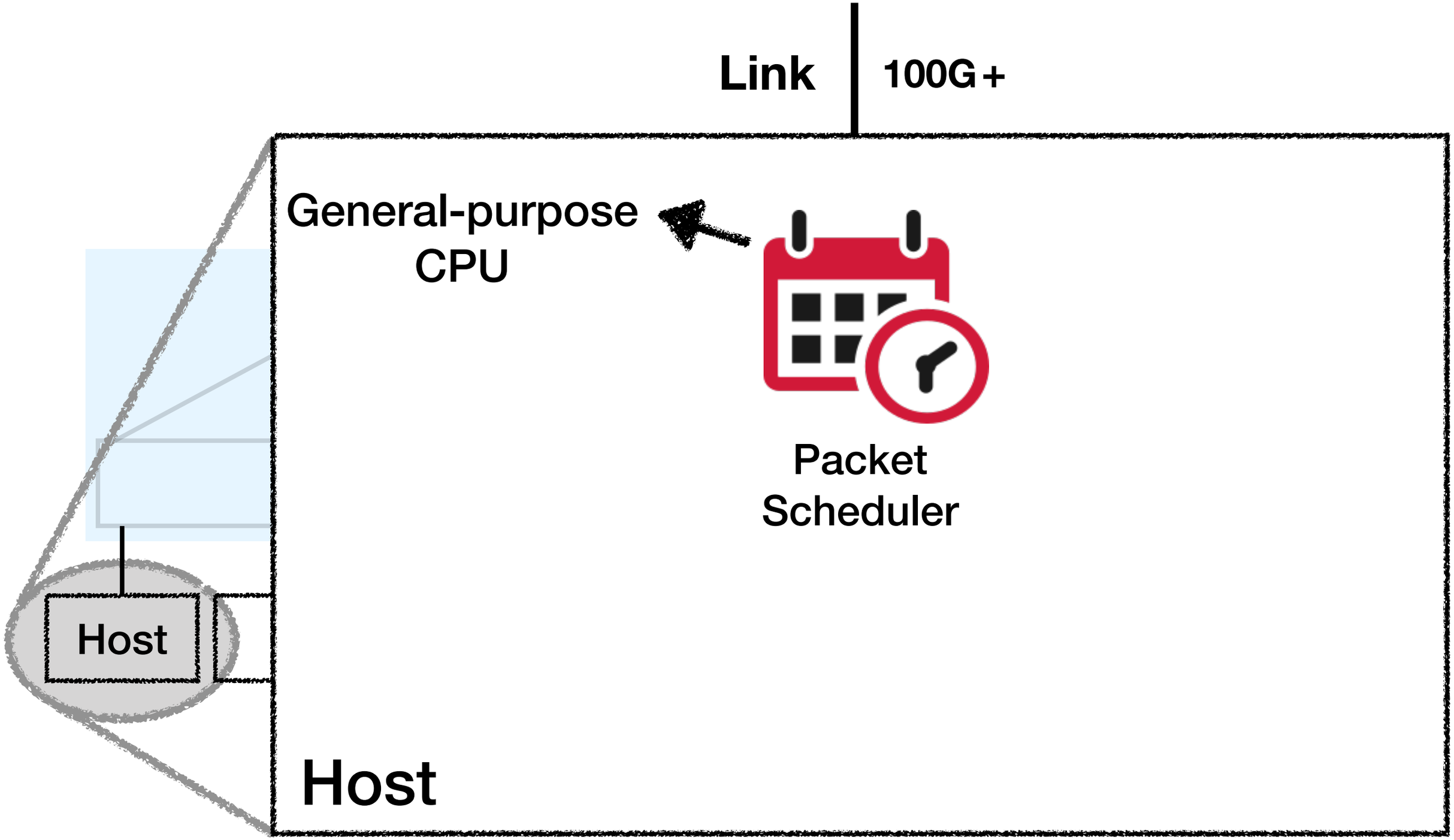
State-of-the-Art Packet Schedulers



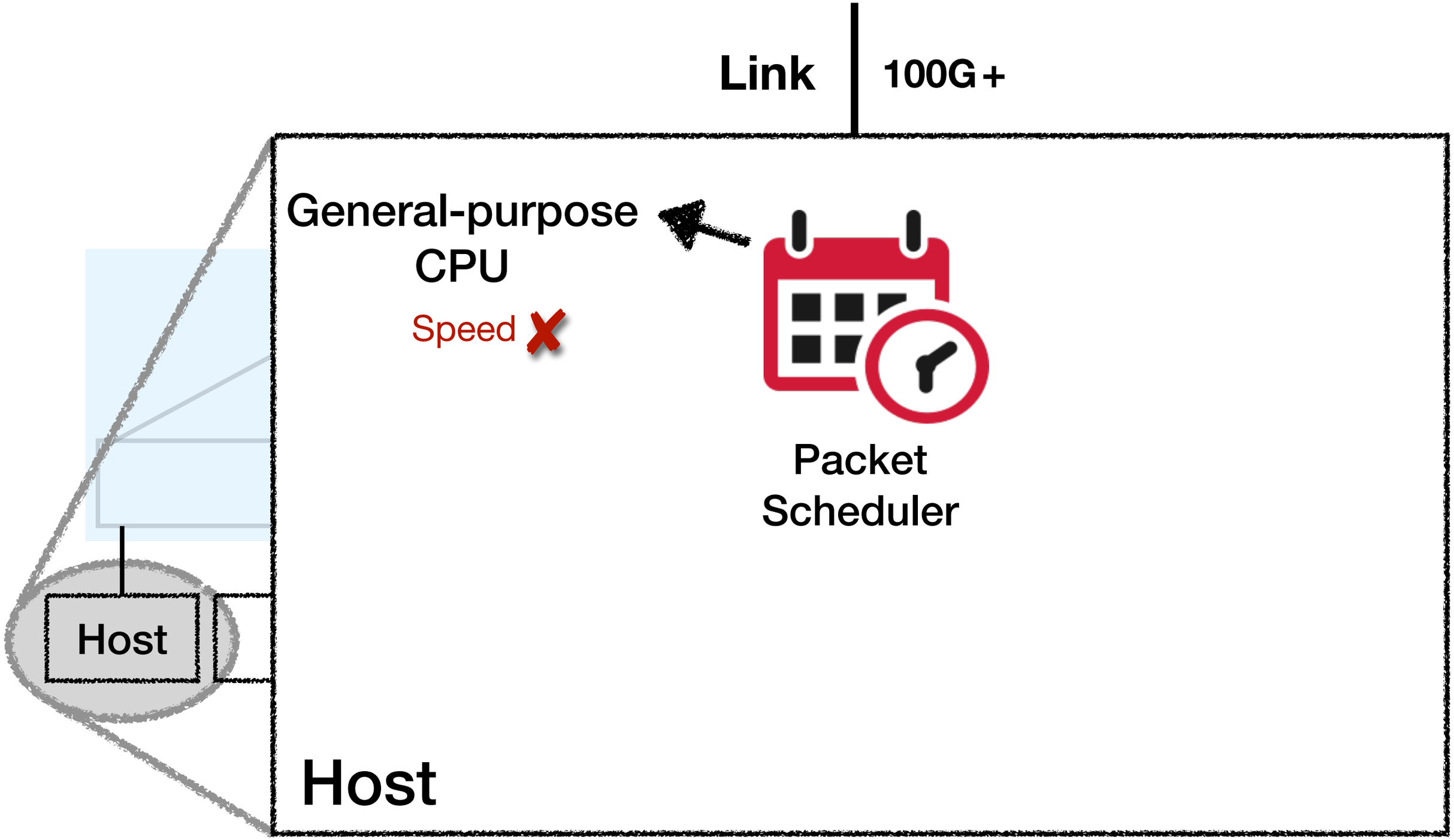
State-of-the-Art Packet Schedulers



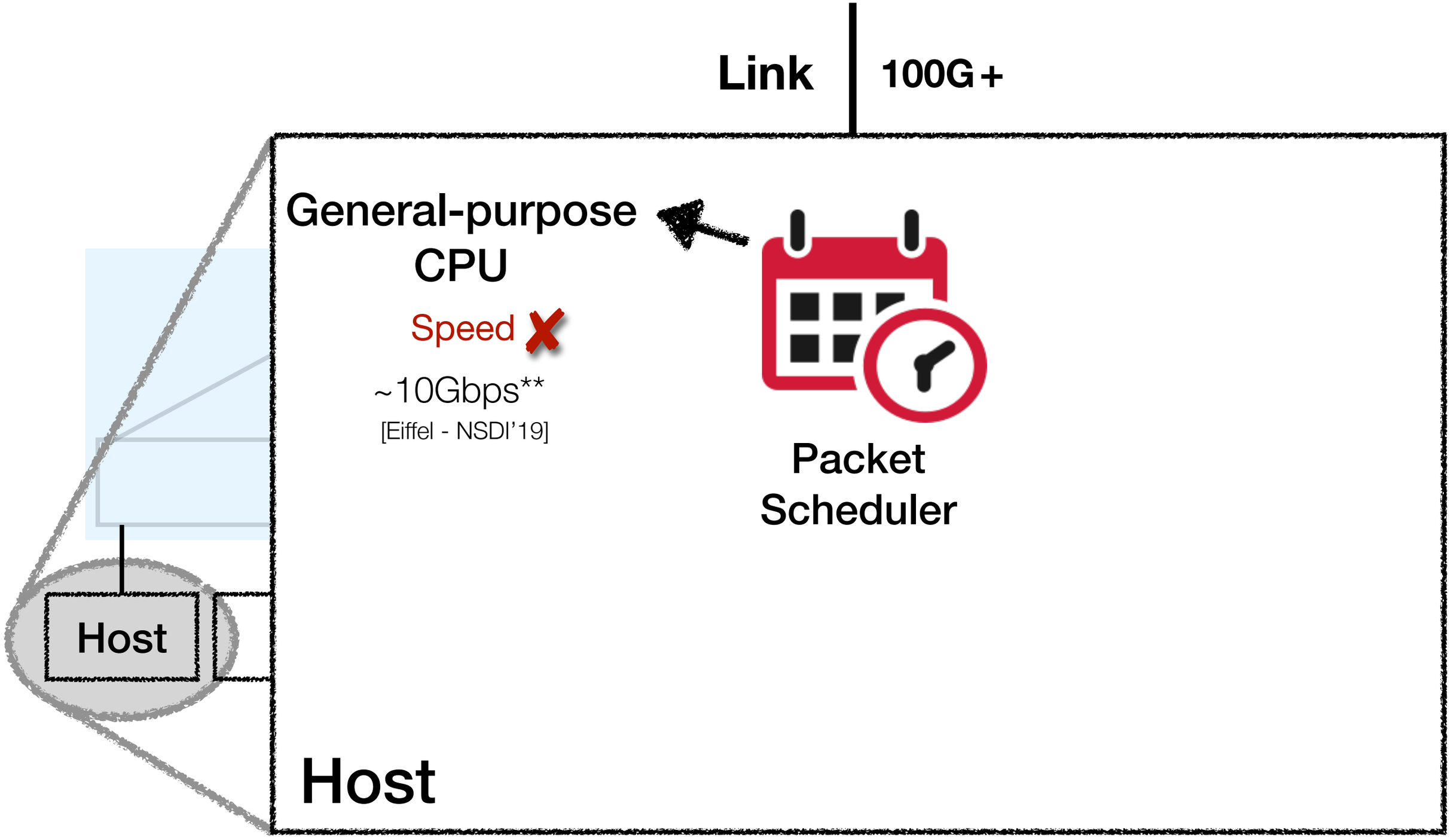
State-of-the-Art Packet Schedulers



State-of-the-Art Packet Schedulers

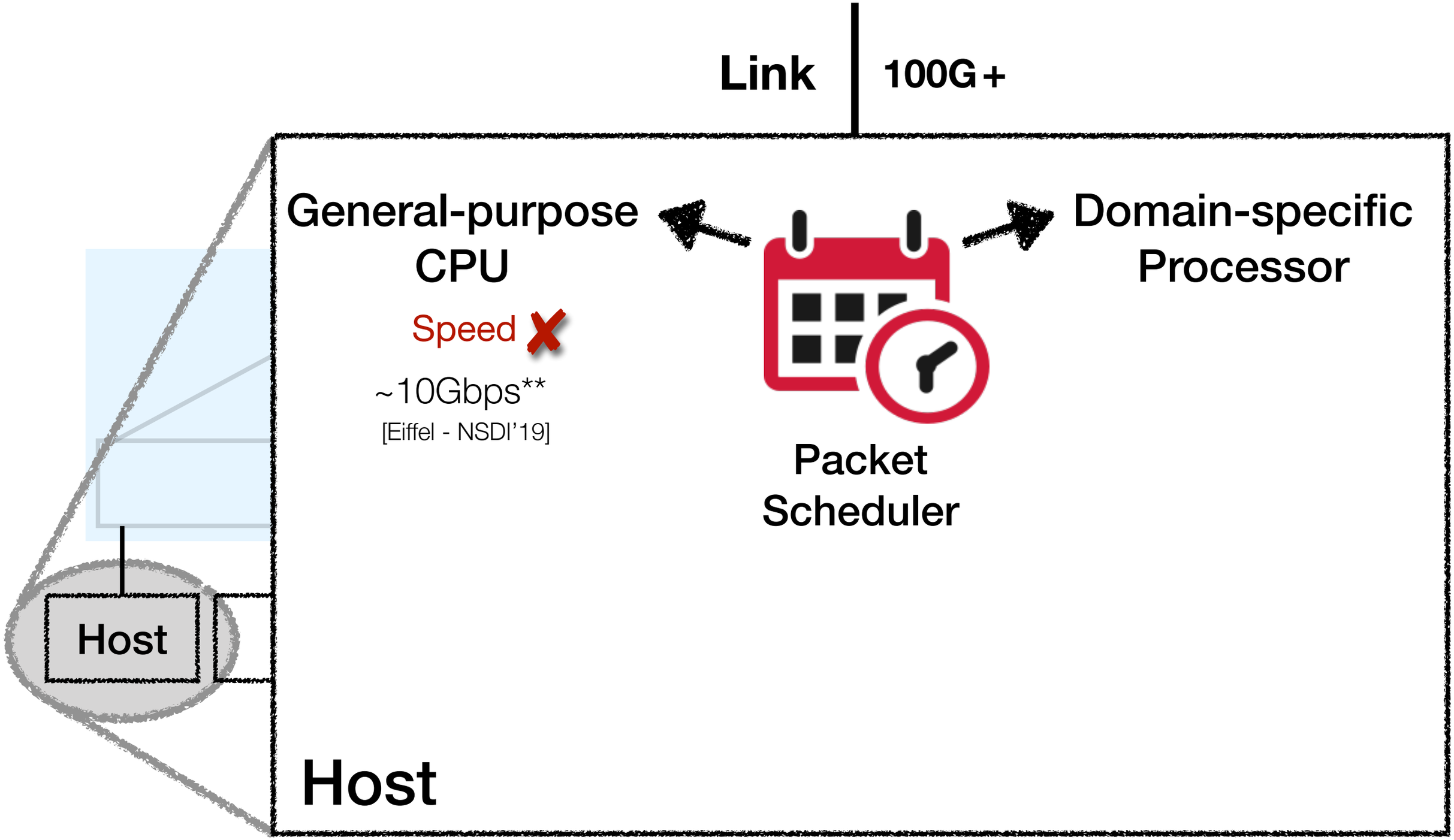


State-of-the-Art Packet Schedulers



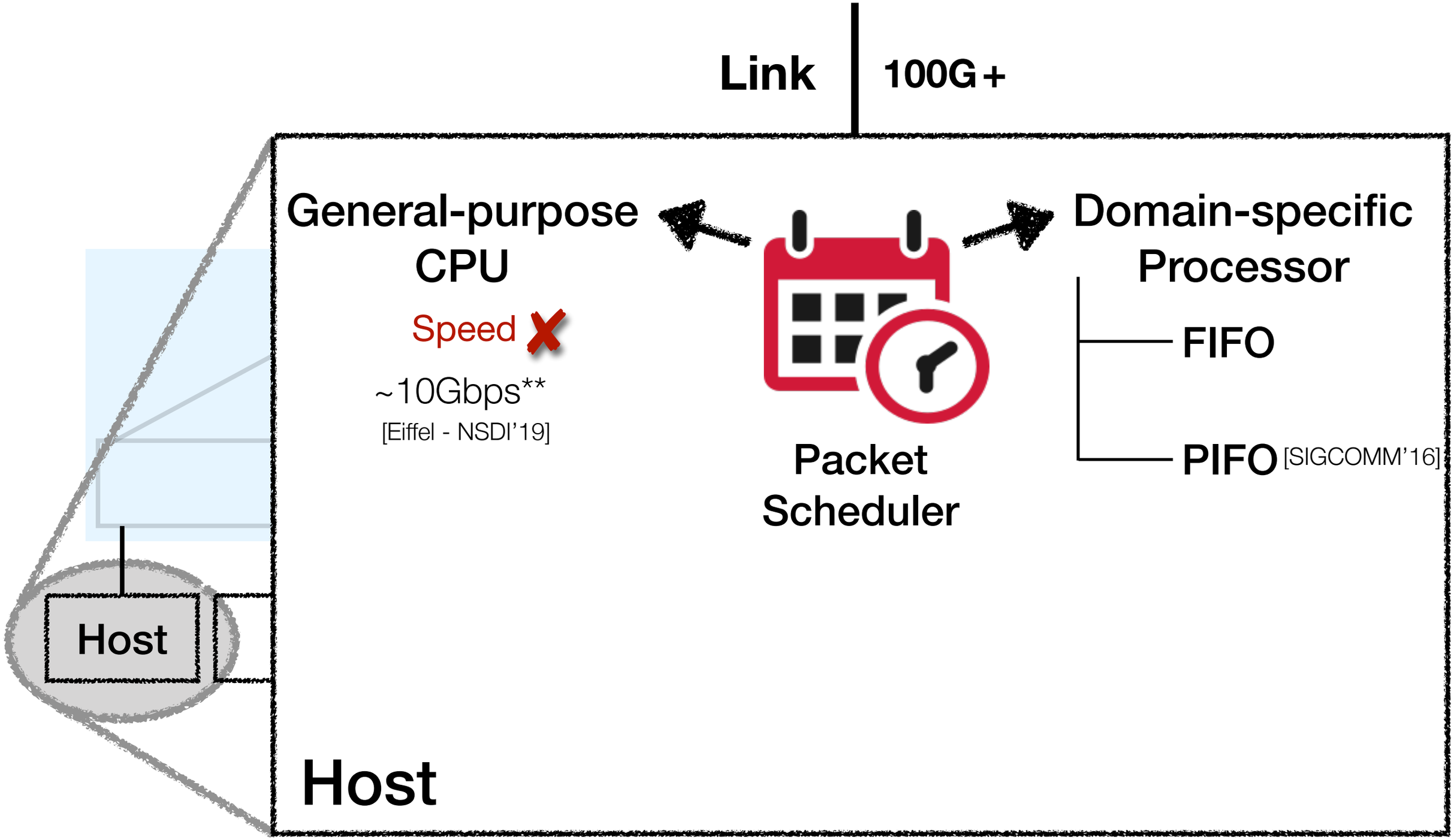
**1 core, 1500B packets, running variant of SJF algorithm

State-of-the-Art Packet Schedulers



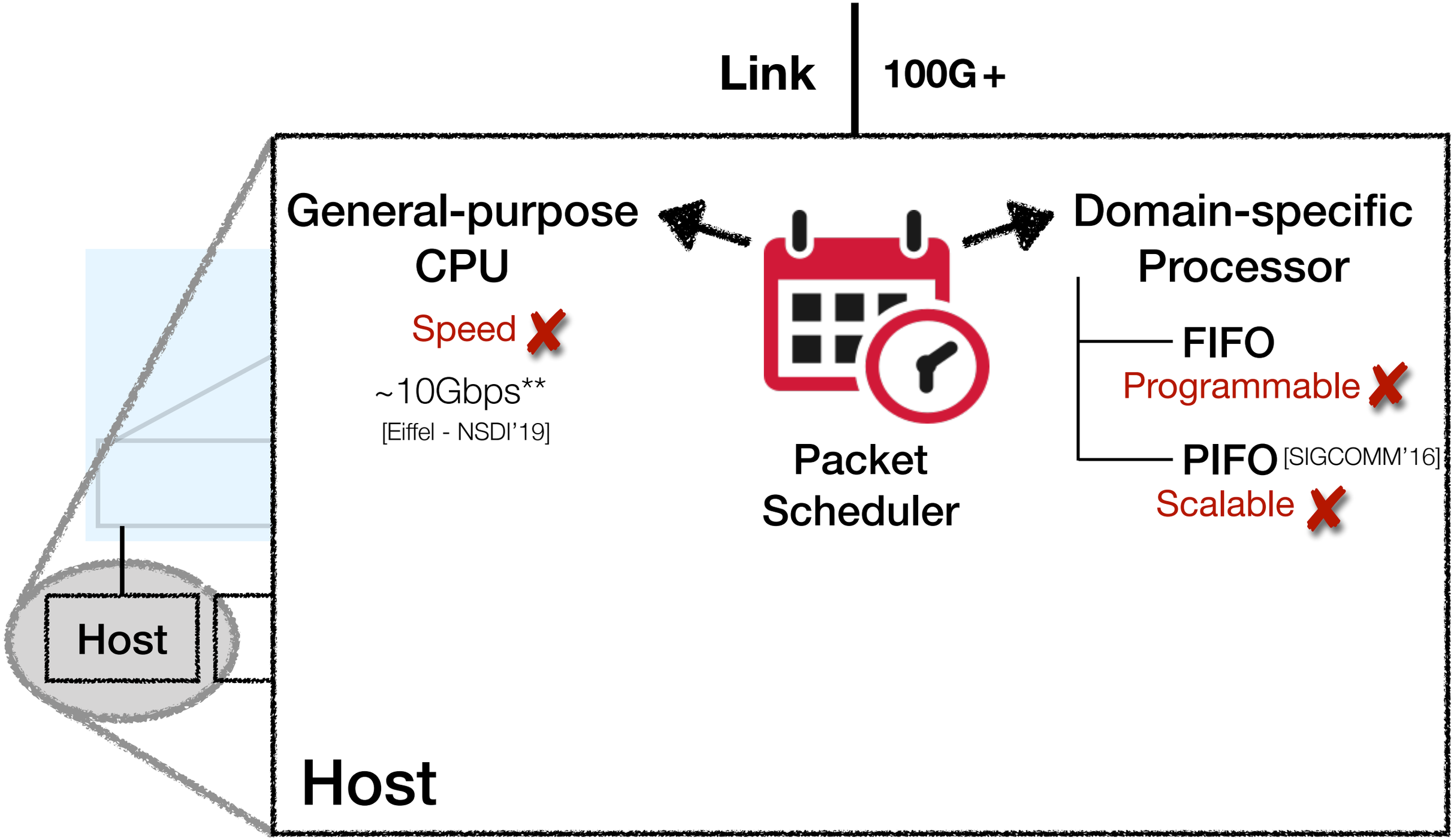
**1 core, 1500B packets, running variant of SJF algorithm

State-of-the-Art Packet Schedulers



**1 core, 1500B packets, running variant of SJF algorithm

State-of-the-Art Packet Schedulers



**1 core, 1500B packets, running variant of SJF algorithm

How to build a packet scheduler that is simultaneously Programmable, Scalable, High-speed?

express wide range of packet scheduling algorithms

schedule 10s of thousands of flows

ref. [SENIC - NSDI'14]
[Carousel - SIGCOMM'17]

schedule at line rate 100G+

How to build a packet scheduler that is simultaneously Programmable, Scalable, High-speed?

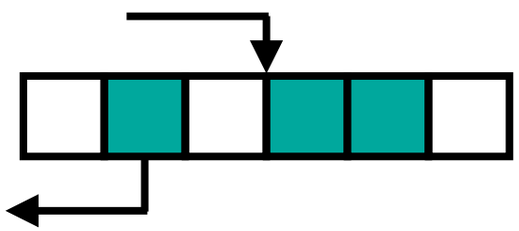
express wide range of packet scheduling algorithms

schedule 10s of thousands of flows

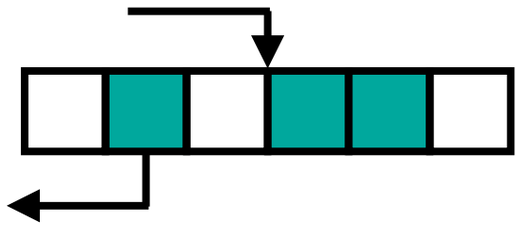
ref. [SENIC - NSDI'14]
[Carousel - SIGCOMM'17]

schedule at line rate 100G+

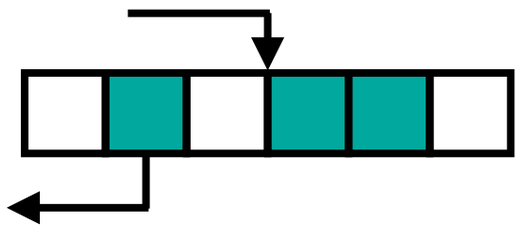
PIEO



PIEO



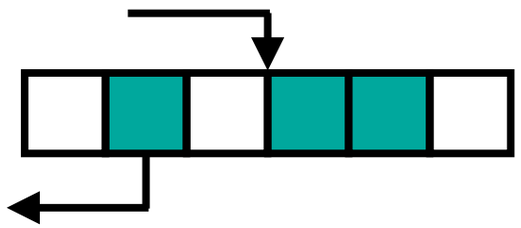
PIEO



Programmable

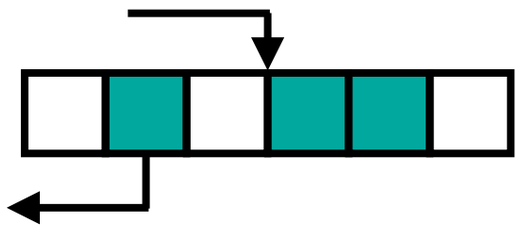
more expressive than any state-of-the-art packet scheduling primitive

PIEO



- **Programmable**
  more expressive than any state-of-the-art packet scheduling primitive
- **Scalable**
  easily scales to 10s of thousands of flows

PIEO



Programmable

more expressive than any state-of-the-art packet scheduling primitive

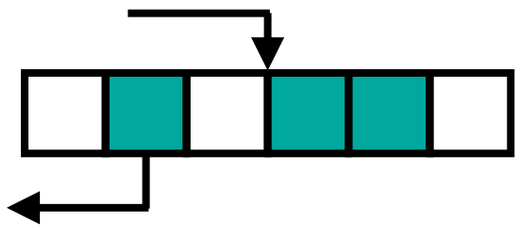
Scalable

easily scales to 10s of thousands of flows

High-speed

makes scheduling decisions in O(1) time [4 clock cycles]

PIEO



Programmable

more expressive than any state-of-the-art packet scheduling primitive

Abstraction

Scalable

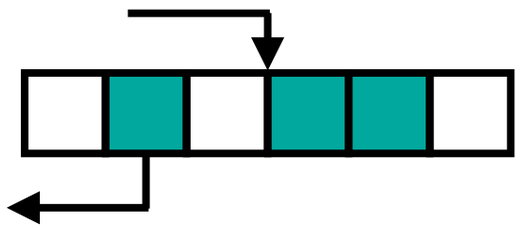
easily scales to 10s of thousands of flows

High-speed

makes scheduling decisions in O(1) time [4 clock cycles]

Hardware Design

PIEO



Programmable

more expressive than any state-of-the-art packet scheduling primitive

Abstraction

Scalable

easily scales to 10s of thousands of flows

Hardware Design

High-speed

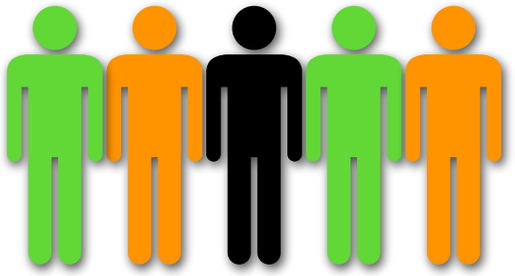
makes scheduling decisions in O(1) time [4 clock cycles]

PIEO Scheduler Abstraction

Scheduling Algorithms

PIEO Scheduler Abstraction

Scheduling Algorithms

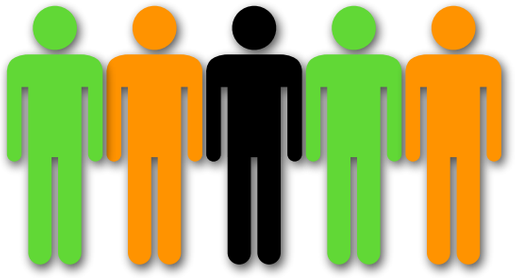


when an element becomes eligible for scheduling?

what order to schedule amongst eligible elements?

PIEO Scheduler Abstraction

Scheduling Algorithms



when an element becomes eligible for scheduling?

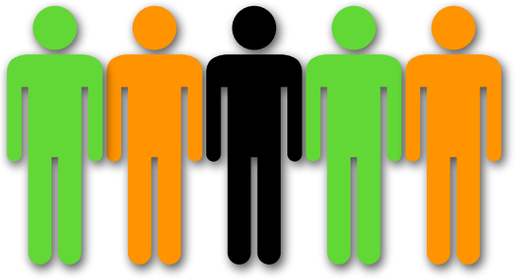
encode using a $t_{eligible}$ value

what order to schedule amongst eligible elements?

encode using a $rank$ value

PIEO Scheduler Abstraction

Scheduling Algorithms



<u>when</u> an element becomes eligible for scheduling?

encode using a $t_{eligible}$ value

<u>what order</u> to schedule amongst eligible elements?

encode using a $rank$ value

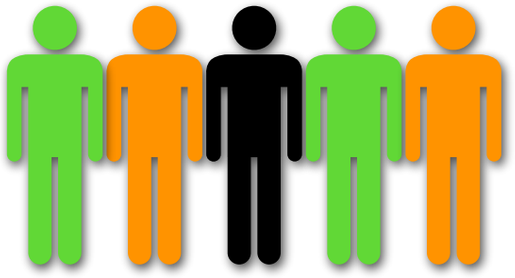
whenever the link is idle:

among all elements satisfying the eligibility predicate $t_{current} \geq t_{eligible}$:

schedule the smallest ranked element

PIEO Scheduler Abstraction

Scheduling Algorithms



<u>when</u> an element becomes eligible for scheduling?

encode using a $t_{eligible}$ value

<u>what order</u> to schedule amongst eligible elements?

encode using a $rank$ value

whenever the link is idle:

among all elements satisfying the eligibility predicate $t_{current} \geq t_{eligible}$:

schedule the smallest ranked element

PIEO scheduler simply schedules the smallest ranked eligible element at any given time

Push-In-Extract-Out Primitive

Push-In-Extract-Out Primitive

programmed based on the choice of scheduling algorithm

Push-In-Extract-Out Primitive

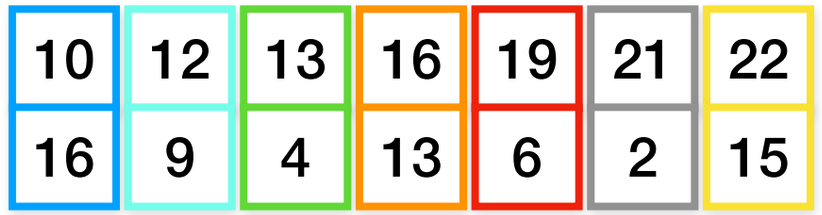
Push-In-Extract-Out Primitive

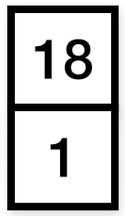
programmed based on the choice of scheduling algorithm

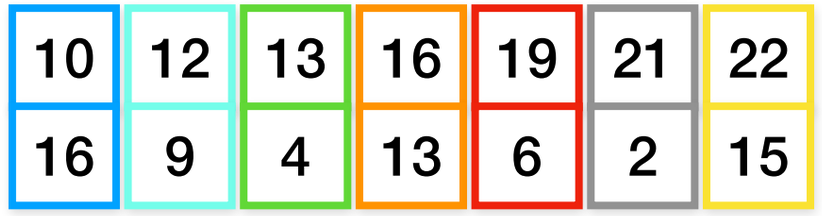
element

ordered list

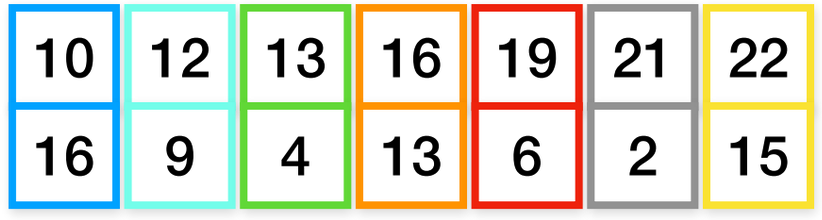
increasing rank value

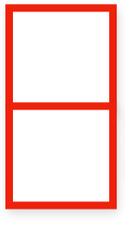


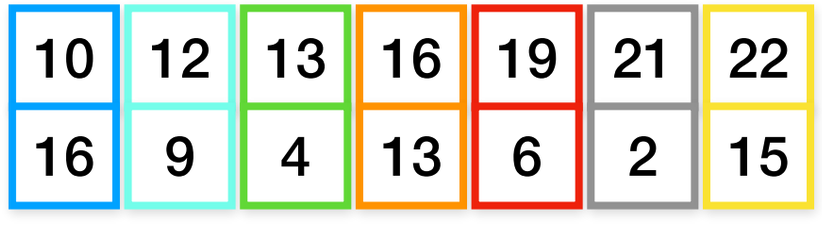
enqueue()



dequeue( )



dequeue()



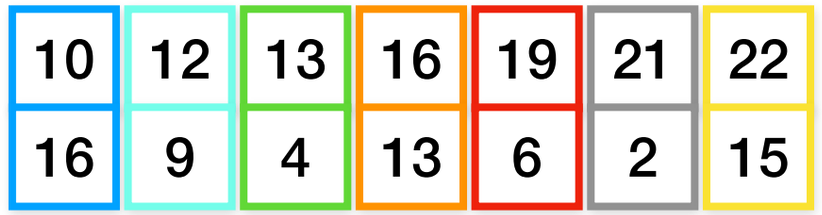
Push-In-Extract-Out Primitive

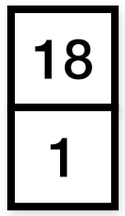
programmed based on the choice of scheduling algorithm

element

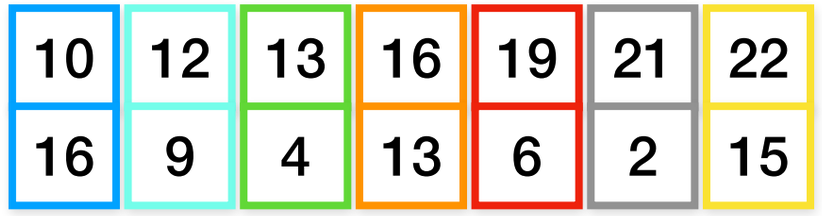
ordered list

increasing rank value

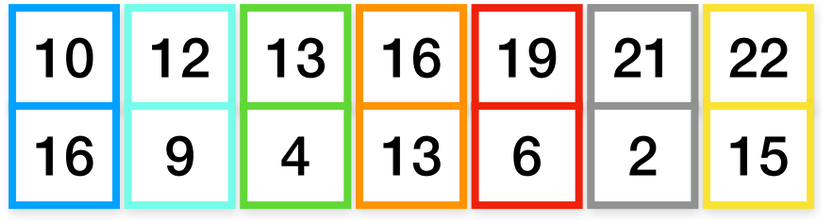


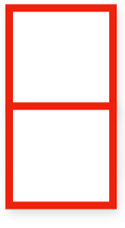
enqueue() "Push-In"

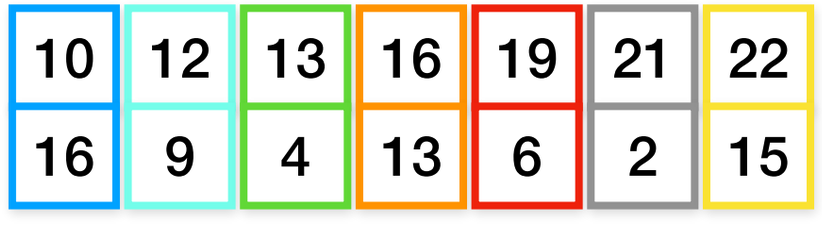
inserts element at position dictated by its rank value



dequeue( )



dequeue()



Push-In-Extract-Out Primitive

programmed based on the choice of scheduling algorithm

element: rank, $t_{eligible}$

ordered list — increasing rank value

10	12	13	16	19	21	22
16	9	4	13	6	2	15

enqueue(18 / 1) “Push-In”

inserts element at position dictated by its rank value

10	12	13	16	18	19	21	22
16	9	4	13	1	6	2	15

dequeue( )

10	12	13	16	19	21	22
16	9	4	13	6	2	15

dequeue( )

10	12	13	16	19	21	22
16	9	4	13	6	2	15

Push-In-Extract-Out Primitive

programmed based on the choice of scheduling algorithm

element: rank, $t_{eligible}$

ordered list

increasing rank value

10	12	13	16	19	21	22
16	9	4	13	6	2	15

enqueue(18 / 1) "Push-In"

inserts element at position dictated by its rank value

10	12	13	16	18	19	21	22
16	9	4	13	1	6	2	15

dequeue( ) "Extract-Out"

returns "smallest ranked eligible" element

10	12	13	16	19	21	22
16	9	4	13	6	2	15

dequeue( )

10	12	13	16	19	21	22
16	9	4	13	6	2	15

Push-In-Extract-Out Primitive

programmed based on the choice of scheduling algorithm

element: rank, $t_{eligible}$

ordered list — increasing rank value

10	12	13	16	19	21	22
16	9	4	13	6	2	15

enqueue(18 / 1) “Push-In”

inserts element at position dictated by its rank value

10	12	13	16	18	19	21	22
16	9	4	13	1	6	2	15

$t_{current} = 7$

dequeue( ) “Extract-Out”

returns “smallest ranked eligible” element

10	12	13	16	19	21	22
16	9	4	13	6	2	15

dequeue( )

10	12	13	16	19	21	22
16	9	4	13	6	2	15

Push-In-Extract-Out Primitive

programmed based on the choice of scheduling algorithm

element: rank, $t_{eligible}$

ordered list — increasing rank value

10	12	13	16	19	21	22
16	9	4	13	6	2	15

enqueue(18 / 1) “Push-In”

inserts element at position dictated by its rank value

10	12	13	16	18	19	21	22
16	9	4	13	1	6	2	15

$t_{current} = 7$ ***filter*** : $t_{current} \geq t_{eligible}$

dequeue( ) “Extract-Out”

returns “smallest ranked eligible” element

10	12	13	16	19	21	22
16	9	4	13	6	2	15

dequeue( )

10	12	13	16	19	21	22
16	9	4	13	6	2	15

Push-In-Extract-Out Primitive

programmed based on the choice of scheduling algorithm

element: rank, $t_{eligible}$

ordered list — increasing rank value

10	12	13	16	19	21	22
16	9	4	13	6	2	15

enqueue(18 / 1) “Push-In”

inserts element at position dictated by its rank value

10	12	13	16	18	19	21	22
16	9	4	13	1	6	2	15

$t_{current} = 7$

filter : $t_{current} \geq t_{eligible}$

dequeue( ) “Extract-Out”

returns “smallest ranked eligible” element

13
4

10	12	16	19	21	22
16	9	13	6	2	15

dequeue( )

10	12	13	16	19	21	22
16	9	4	13	6	2	15

Push-In-Extract-Out Primitive

programmed based on the choice of scheduling algorithm

element: rank, $t_{eligible}$

ordered list — increasing rank value

10	12	13	16	19	21	22
16	9	4	13	6	2	15

enqueue(18 / 1) “Push-In”

inserts element at position dictated by its rank value

10	12	13	16	18	19	21	22
16	9	4	13	1	6	2	15

$t_{current} = 7$

filter : $t_{current} \geq t_{eligible}$

dequeue( ) “Extract-Out”

returns “smallest ranked eligible” element

13
4

10	12	16	19	21	22
16	9	13	6	2	15

dequeue( )

returns a specific element

10	12	13	16	19	21	22
16	9	4	13	6	2	15

Push-In-Extract-Out Primitive

programmed based on the choice of scheduling algorithm

element: rank, $t_{eligible}$

ordered list — increasing rank value

10	12	13	16	19	21	22
16	9	4	13	6	2	15

enqueue(18 / 1) “Push-In”

inserts element at position dictated by its rank value

10	12	13	16	18	19	21	22
16	9	4	13	1	6	2	15

$t_{current} = 7$

filter : $t_{current} \geq t_{eligible}$

dequeue( ) “Extract-Out”

returns “smallest ranked eligible” element

13 / 4

10	12	16	19	21	22
16	9	13	6	2	15

dequeue( )

returns a specific element

19 / 6

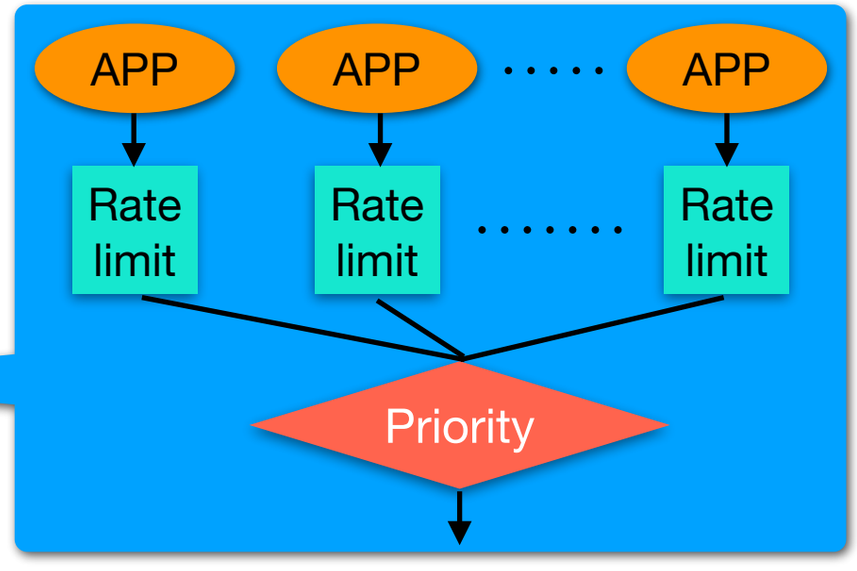
10	12	13	16	21	22
16	9	4	13	2	15

Expressiveness of PIEO

Expressiveness of PIEO

- Work conserving
  - e.g., DRR, WFQ, WF^2Q
- Non-work conserving
  - e.g., Token Bucket, RCSP
- Hierarchical scheduling
  - e.g., HPFQ
- Asynchronous scheduling
  - e.g., Starvation avoidance, D^3
- Priority scheduling
  - e.g., SJF, SRTF, LSTF, EDF
- Complex scheduling policies
  - mixture of shaping and ordering

e.g.

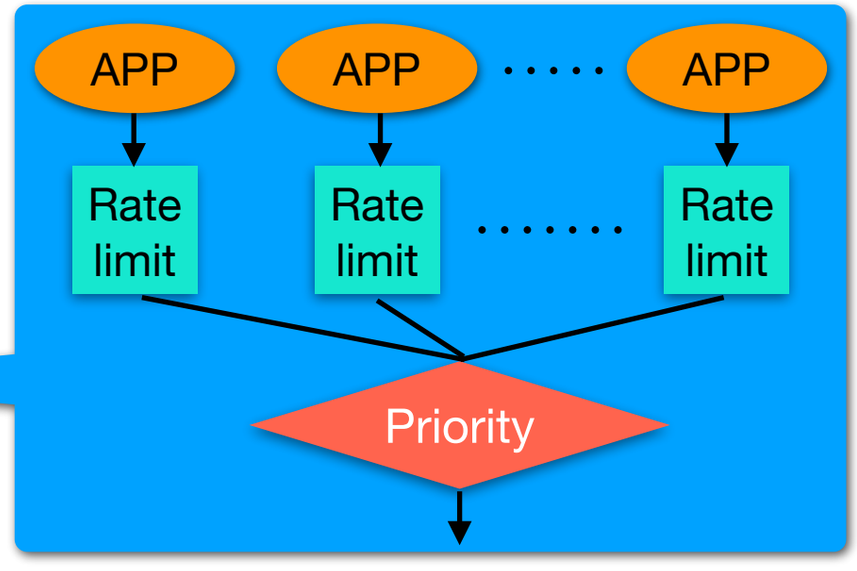


Expressiveness of PIEO

- Work conserving
  - e.g., DRR, WFQ, WF^2Q
- Non-work conserving
  - e.g., Token Bucket, RCSP
- Hierarchical scheduling
  - e.g., HPFQ
- Asynchronous scheduling
  - e.g., Starvation avoidance, D^3
- Priority scheduling
  - e.g., SJF, SRTF, LSTF, EDF
- Complex scheduling policies
  - mixture of shaping and ordering

for each element:
calculate **start_time** and **finish_time**
at time x, all elements s.t. **virtual_time(x) >= start_time:**
schedule element with **smallest finish_time**

e.g.



Expressiveness of PIEO

- Work conserving
  - e.g., DRR, WFQ, WF^2Q
- Non-work conserving
  - e.g., Token Bucket, RCSP
- Hierarchical scheduling
  - e.g., HPFQ
- Asynchronous scheduling
  - e.g., Starvation avoidance, D^3
- Priority scheduling
  - e.g., SJF, SRTF, LSTF, EDF
- Complex scheduling policies
  - mixture of shaping and ordering

for each element:
calculate **start_time** and **finish_time**
at time x, all elements s.t. **virtual_time(x) >= start_time:**
schedule element with **smallest finish_time**

programming PIEO

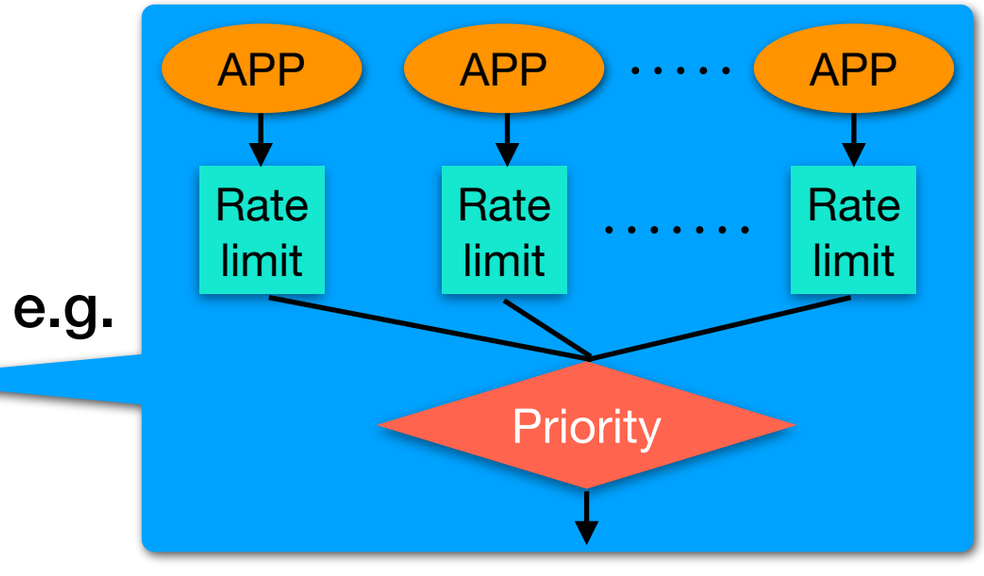
$rank = finish\_time$

$t_{eligible} = start\_time$

Predicate for filtering at dequeue at time x:

$(virtual\_time(x) \geq t_{eligible})$

e.g.



Expressiveness of PIEO

- Work conserving
  - e.g., DRR, WFQ, WF^2Q
- Non-work conserving
  - e.g., Token Bucket, RCSP
- Hierarchical scheduling
  - e.g., HPFQ
- Asynchronous scheduling
  - e.g., Starvation avoidance, D^3
- Priority scheduling
  - e.g., SJF, SRTF, LSTF, EDF
- Complex scheduling policies
  - mixture of shaping and ordering

for each element:
calculate **start_time** and **finish_time**
at time x, all elements s.t. **virtual_time(x) >= start_time:**
schedule element with **smallest finish_time**

programming PIEO

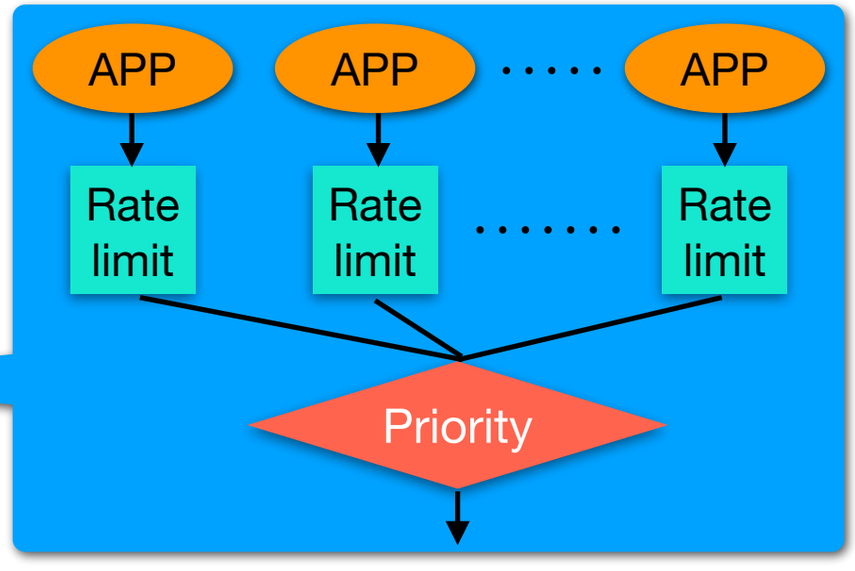
$rank = finish\_time$

$t_{eligible} = start\_time$

Predicate for filtering at dequeue at time x:

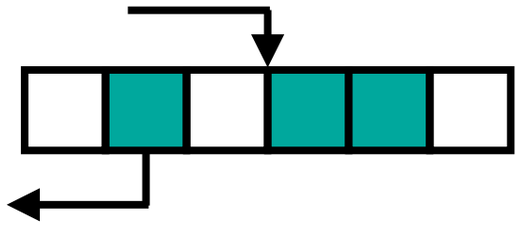
$(virtual\_time(x) \geq t_{eligible})$

e.g.



cannot express accurately using state-of-the-art (PIFO)

PIEO



Programmable

more expressive than any state-of-the-art packet scheduling primitive

Abstraction

Hardware Design

Scalable

easily scales to 10s of thousands of flows

High-speed

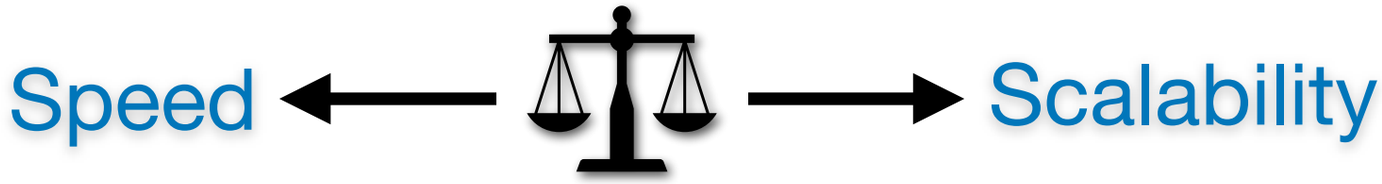
makes scheduling decisions in O(1) time [4 clock cycles]

Hardware Design

PIEO primitive relies on an ordered list datastructure

Hardware Design

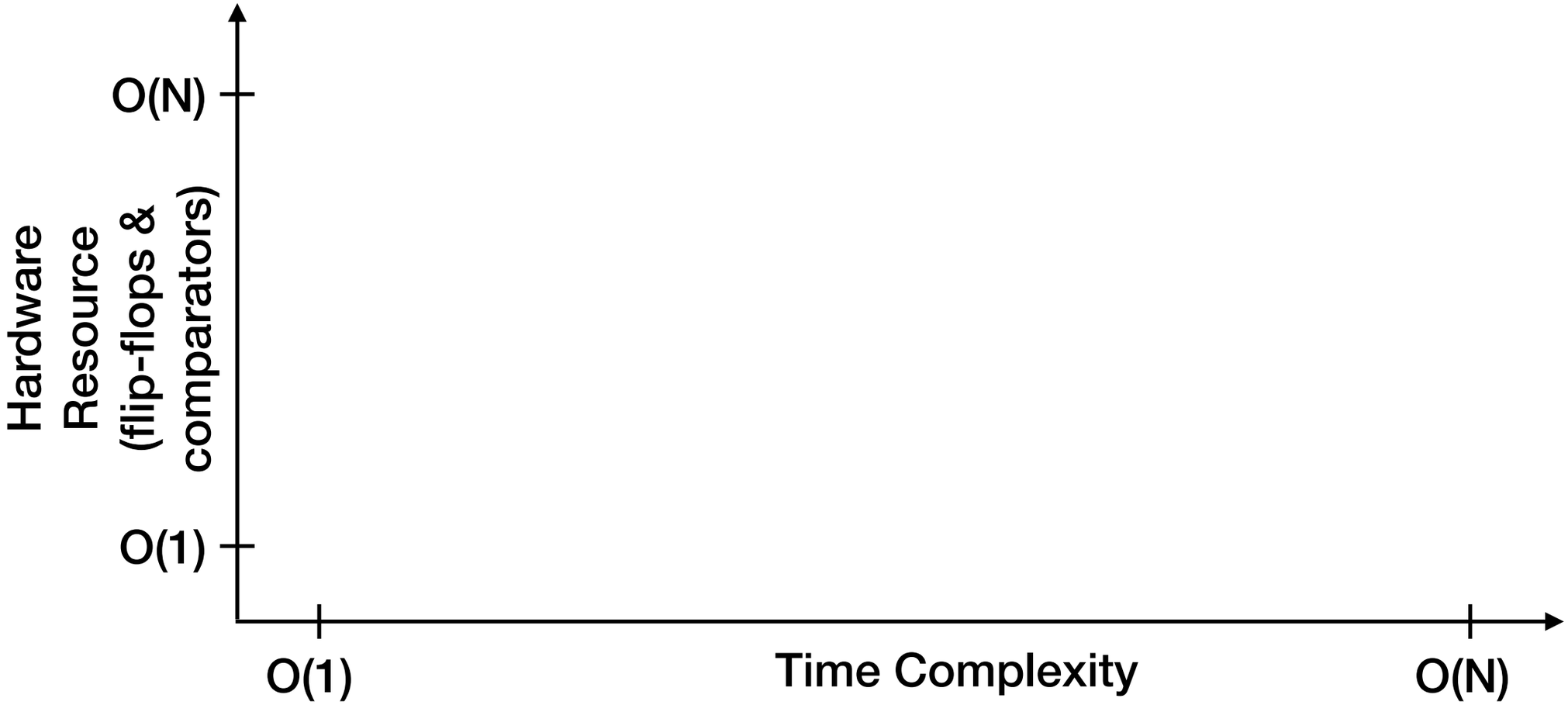
PIEO primitive relies on an ordered list datastructure



Hardware Design

PIEO primitive relies on an ordered list datastructure

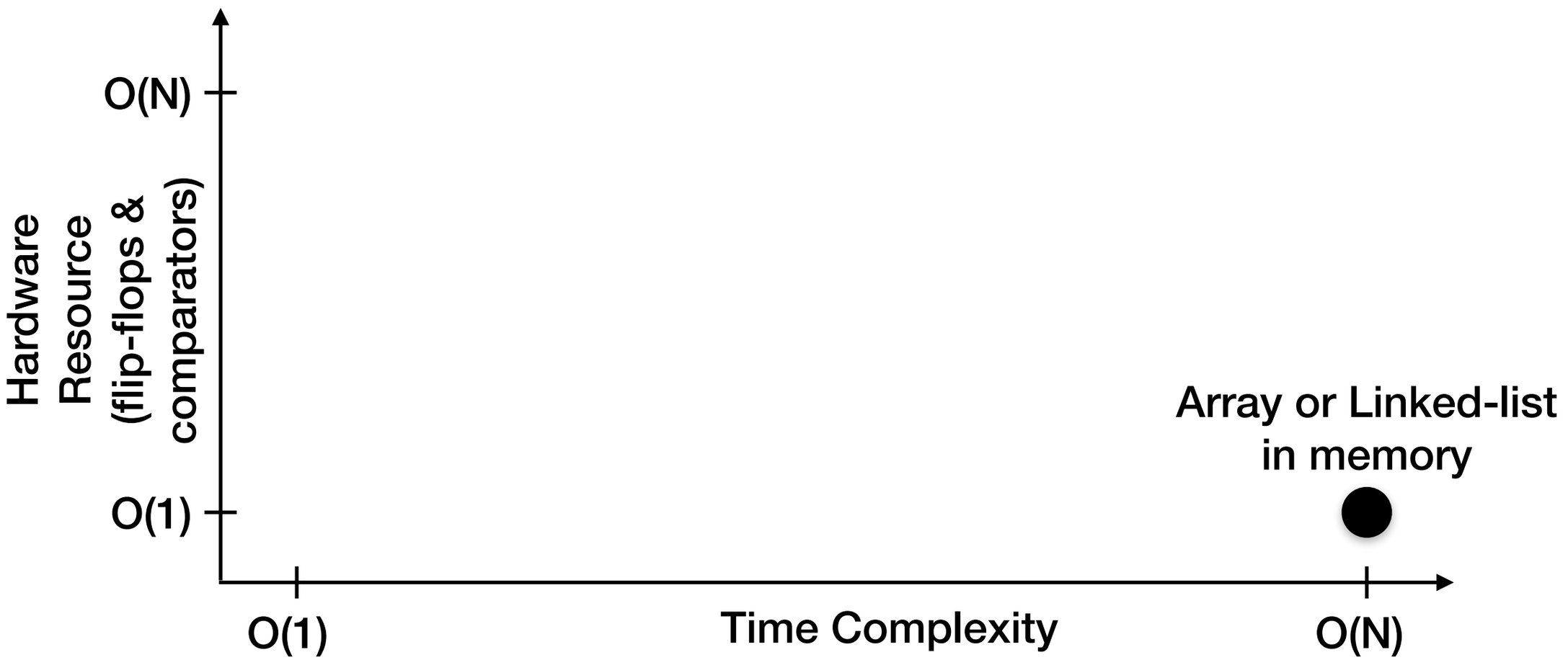
Speed ⟵ ⚖ ⟶ Scalability



Hardware Design

PIEO primitive relies on an ordered list datastructure

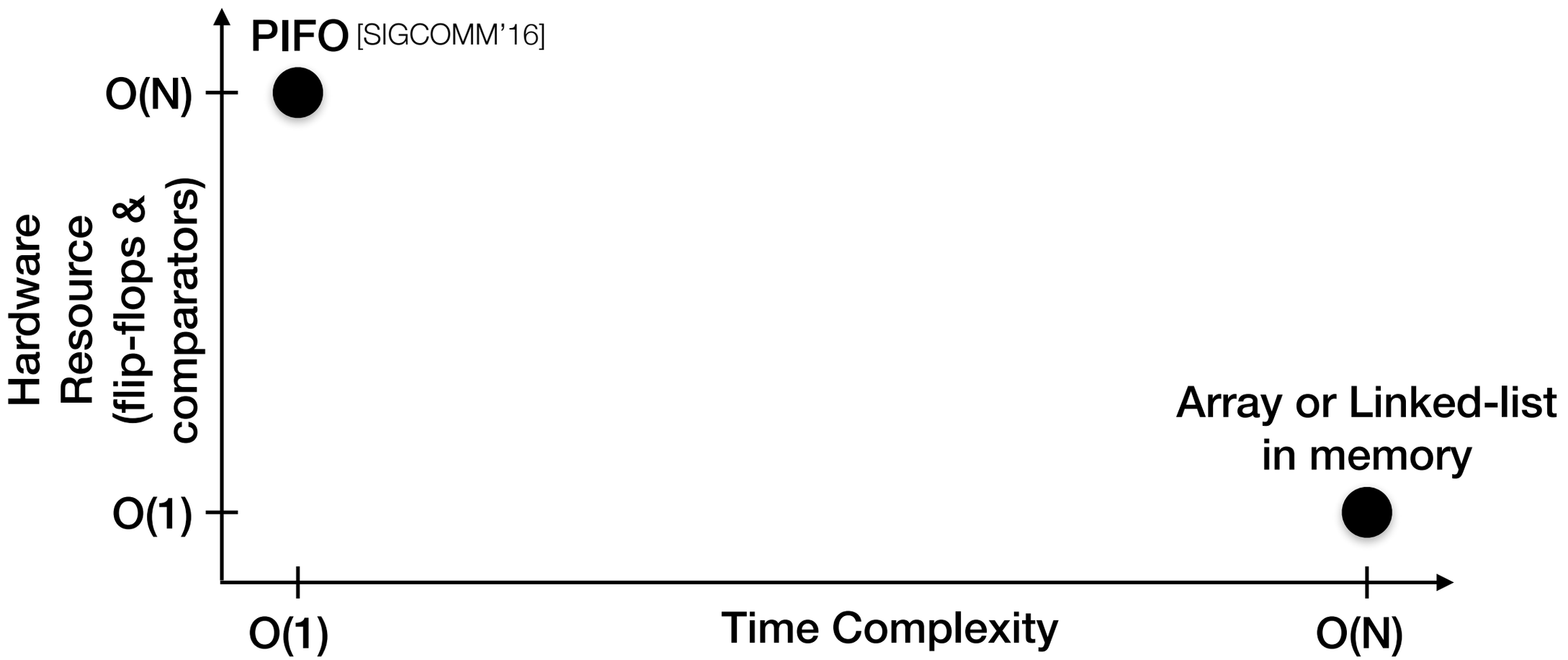
Speed ⟵ ⚖ ⟶ Scalability



Hardware Design

PIEO primitive relies on an ordered list datastructure

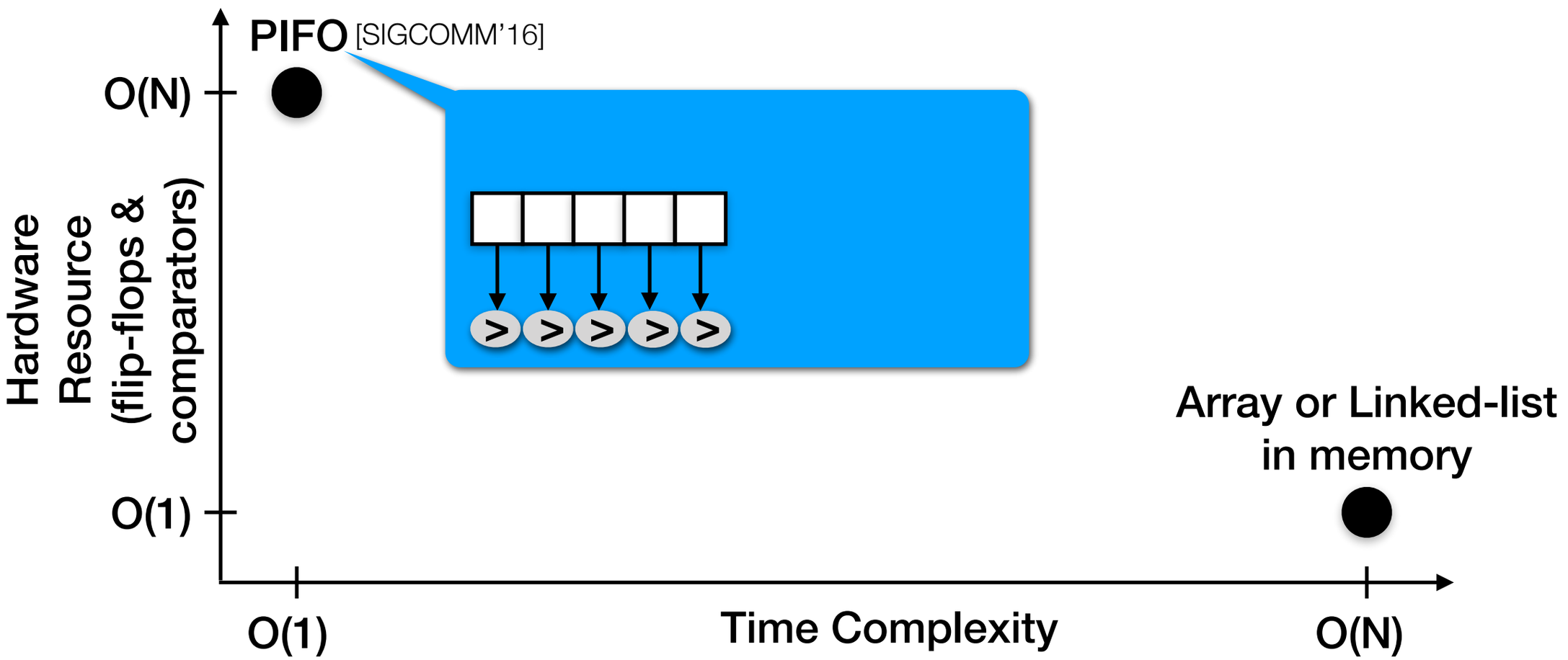
Speed ⟵ ⚖ ⟶ Scalability



Hardware Design

PIEO primitive relies on an ordered list datastructure

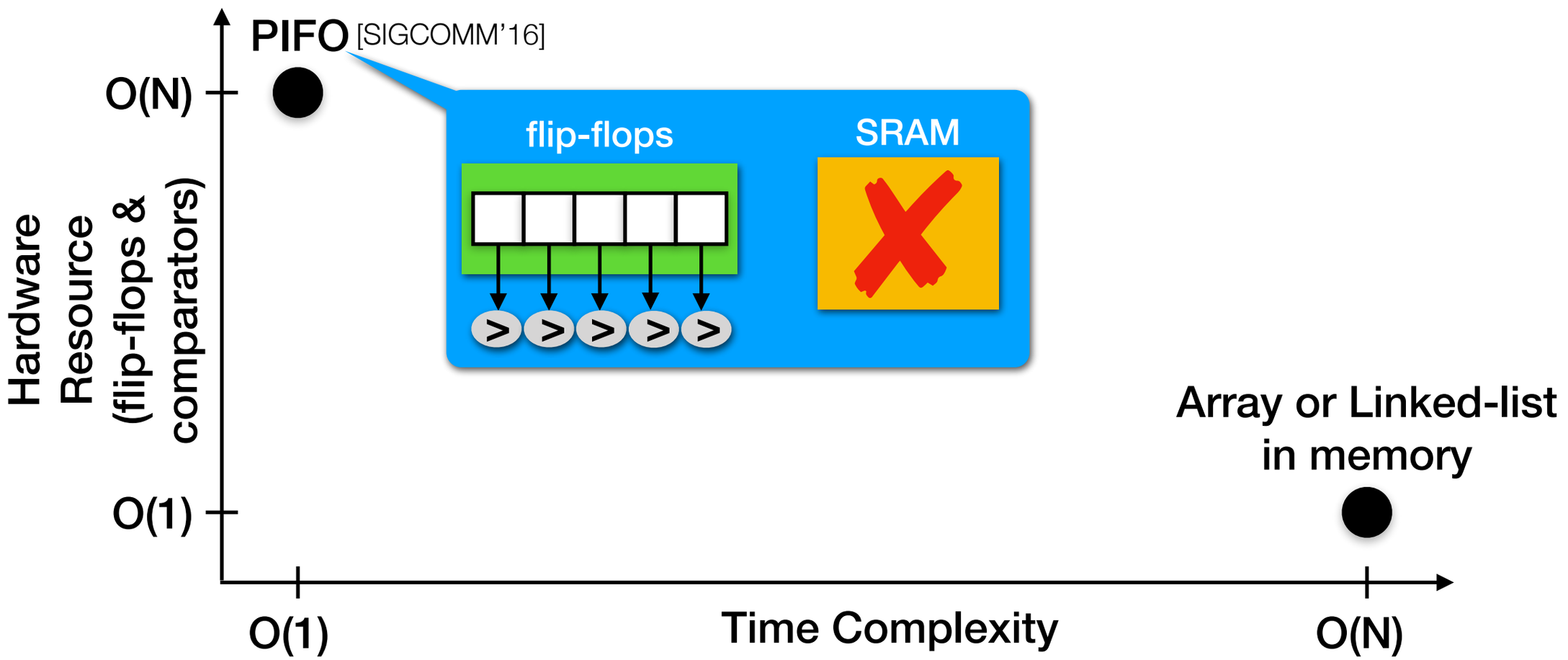
Speed ⟵ ⚖ ⟶ Scalability



Hardware Design

PIEO primitive relies on an ordered list datastructure

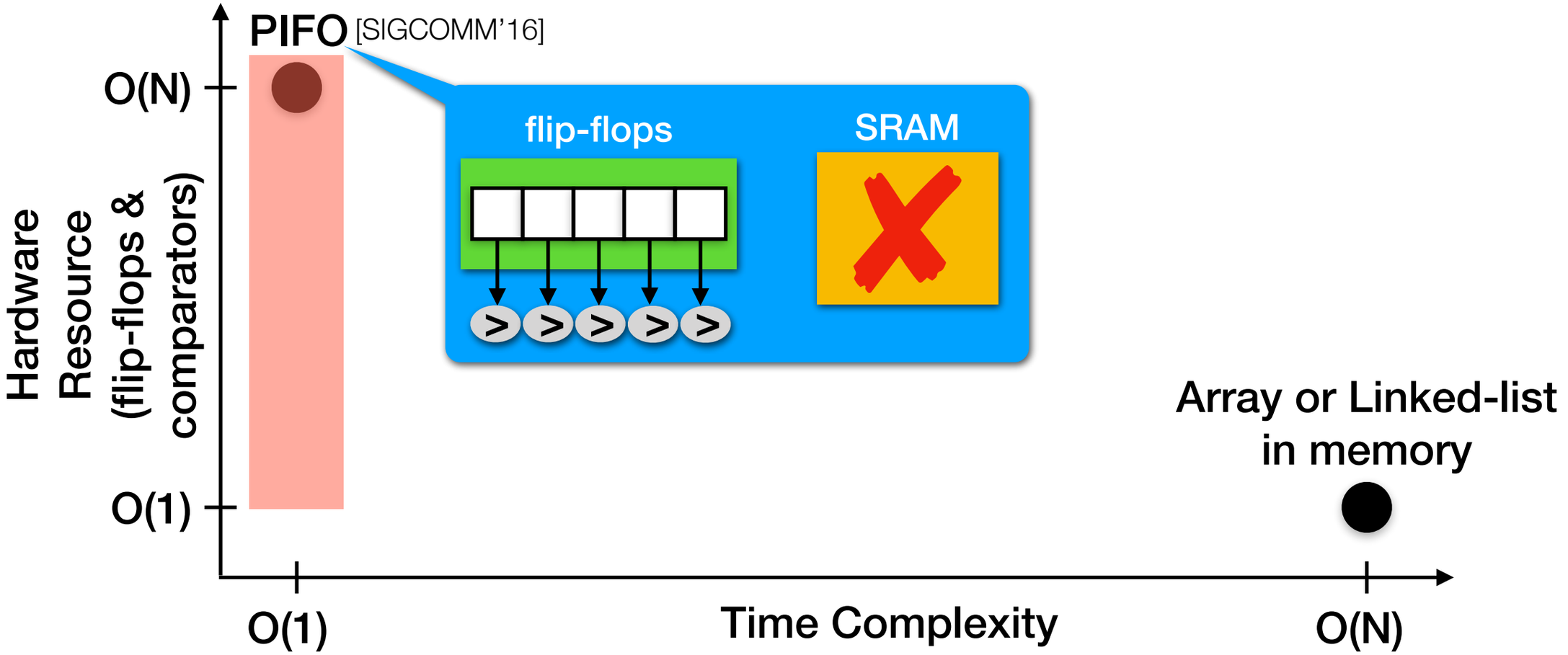
Speed ⟵ ⚖ ⟶ Scalability



Hardware Design

PIEO primitive relies on an ordered list datastructure

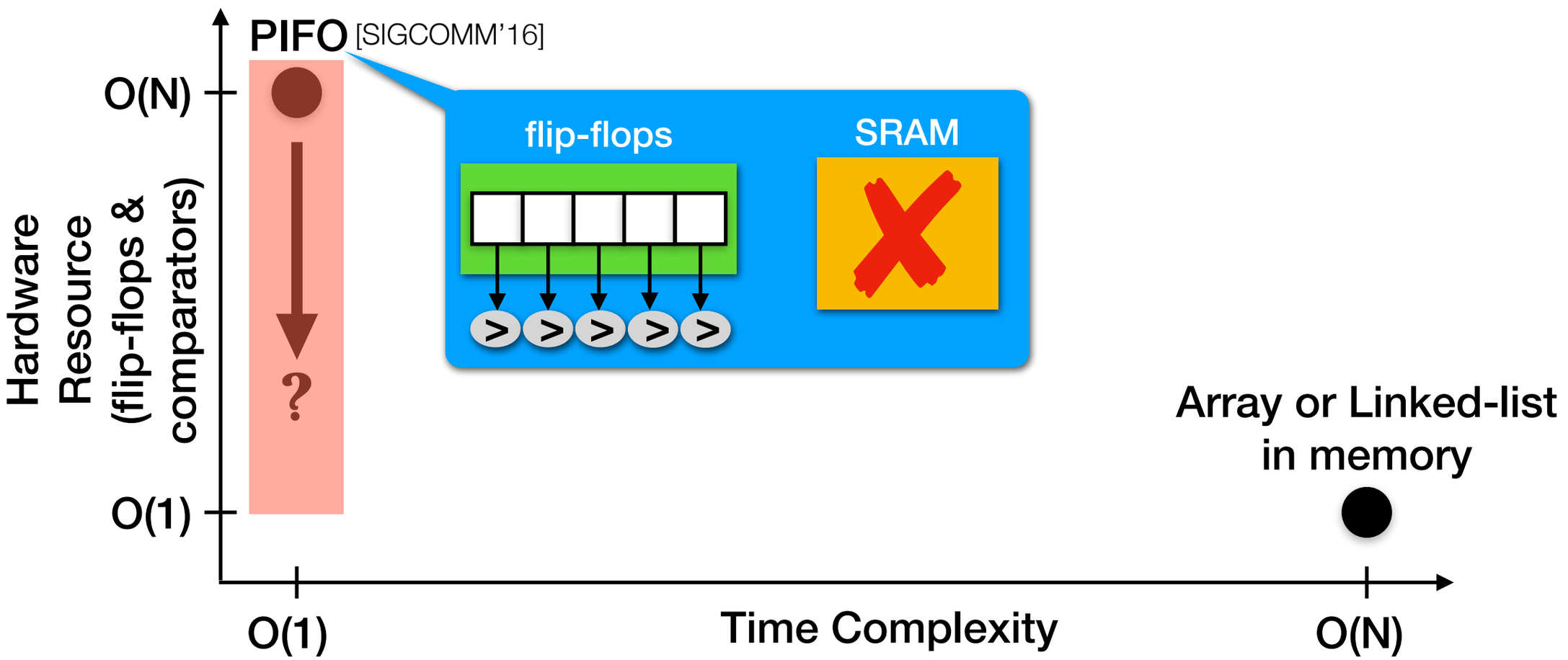
Speed ⟵ ⟶ Scalability



Hardware Design

PIEO primitive relies on an ordered list datastructure

Speed ⟵ ⟶ Scalability



Is it fundamentally necessary to access and compare $O(N)$ elements in parallel to maintain an (exact) ordered list (of size N) in $O(1)$ time?

Is it fundamentally necessary to access and compare $O(N)$ elements in parallel to maintain an (exact) ordered list (of size N) in $O(1)$ time?

We present a design that can maintain an (exact) ordered list in $O(1)$ time, but only needs to access and compare $O(\sqrt{N})$ elements in parallel.

Is it fundamentally necessary to access and compare $O(N)$ elements in parallel to maintain an (exact) ordered list (of size N) in $O(1)$ time?

We present a design that can maintain an (exact) ordered list in $O(1)$ time, but only needs to access and compare $O(\sqrt{N})$ elements in parallel.

Key Insight

"All problems in computer science can be solved by another level of indirection"

—David Wheeler

Hardware Architecture

Flip-Flops

SRAM

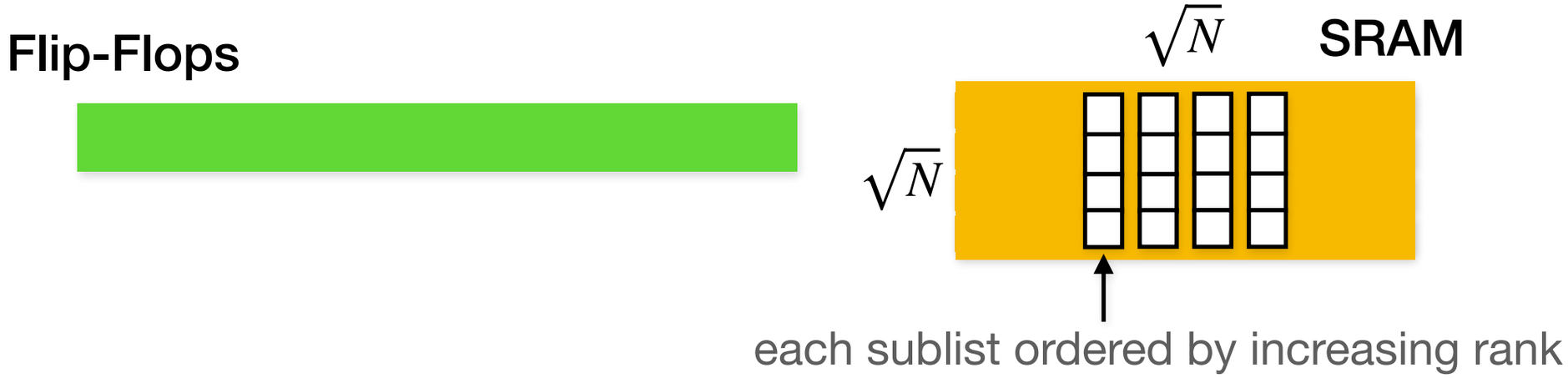
Hardware Architecture

Flip-Flops

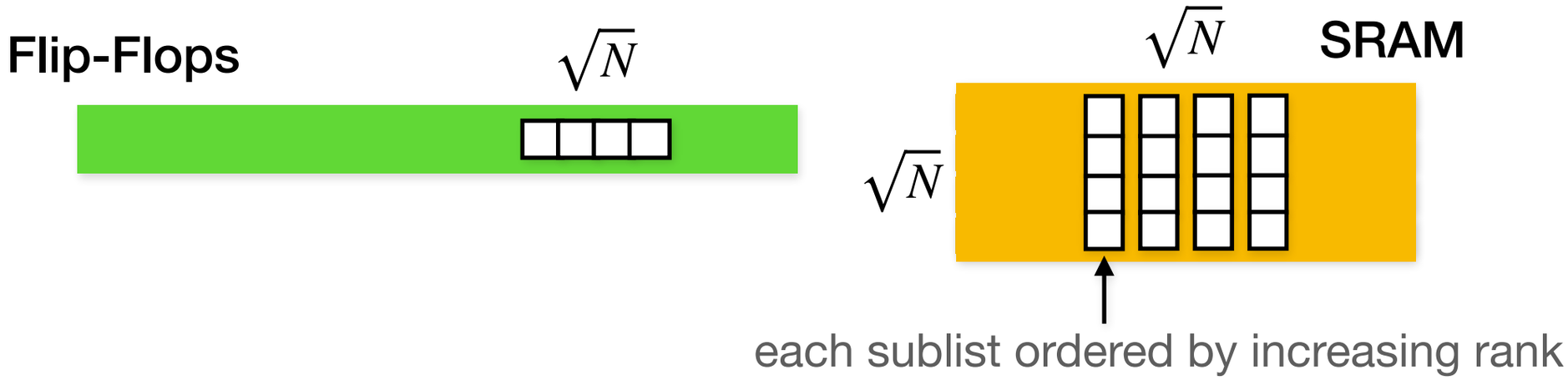
SRAM

Hardware Architecture

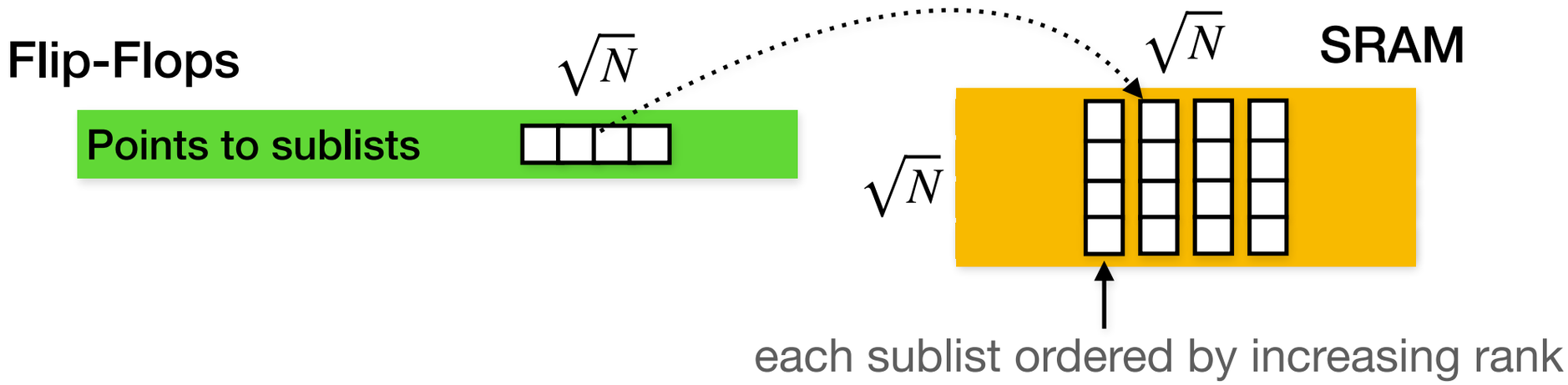
Hardware Architecture



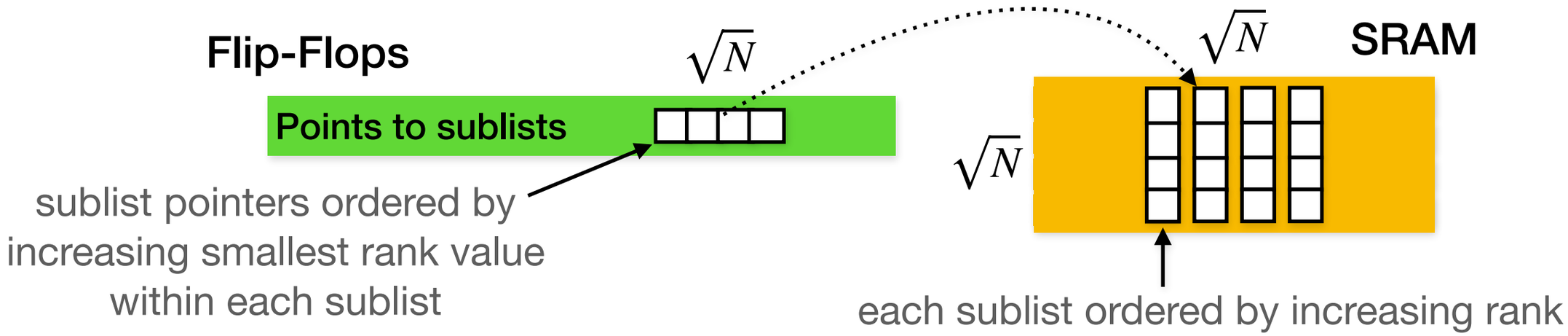
Hardware Architecture



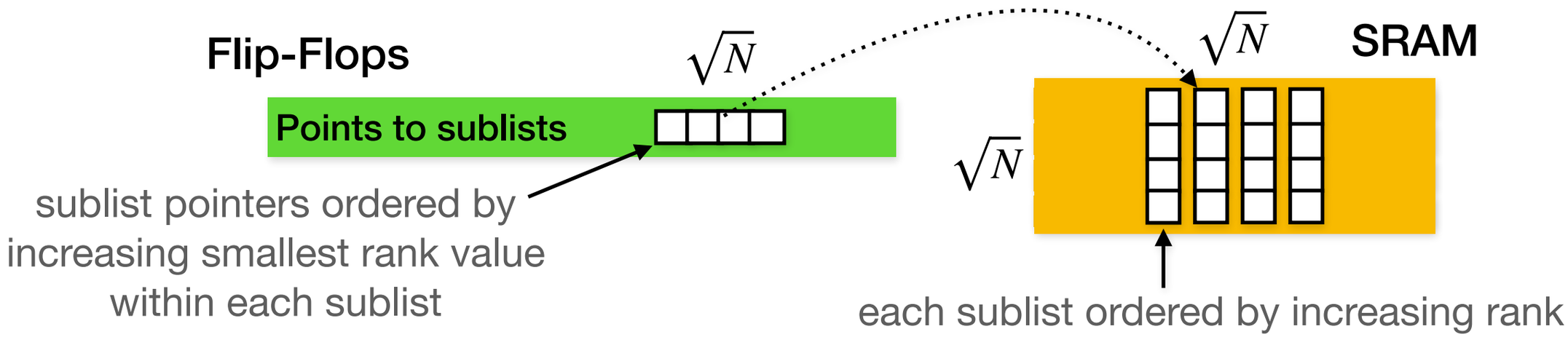
Hardware Architecture



Hardware Architecture

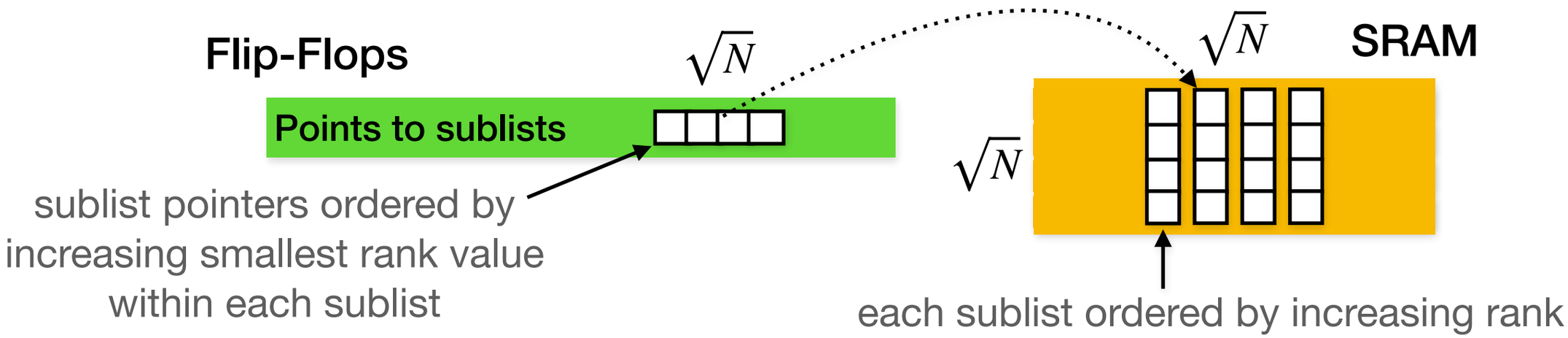


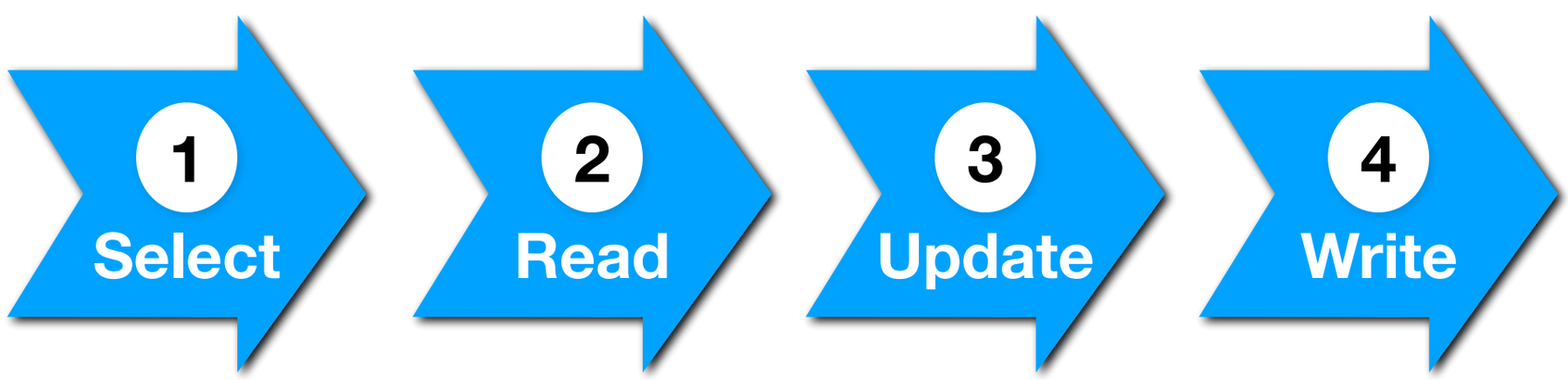
Hardware Architecture



enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

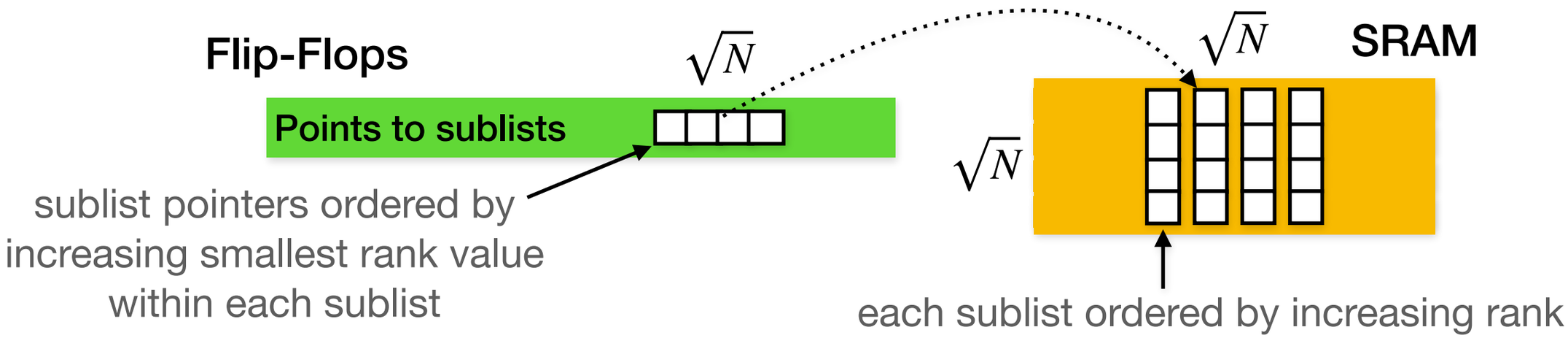
Hardware Architecture

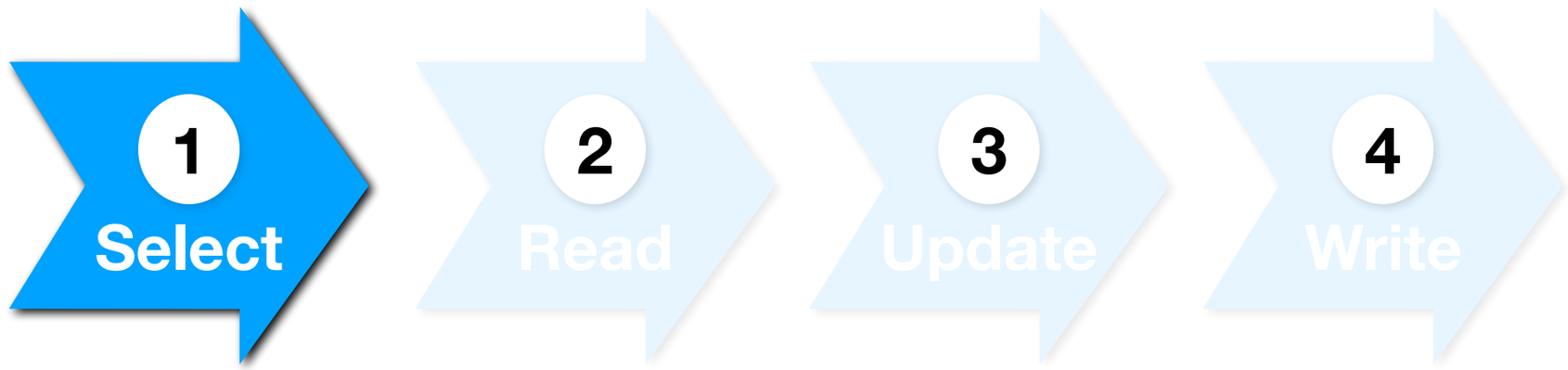




enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

Hardware Architecture

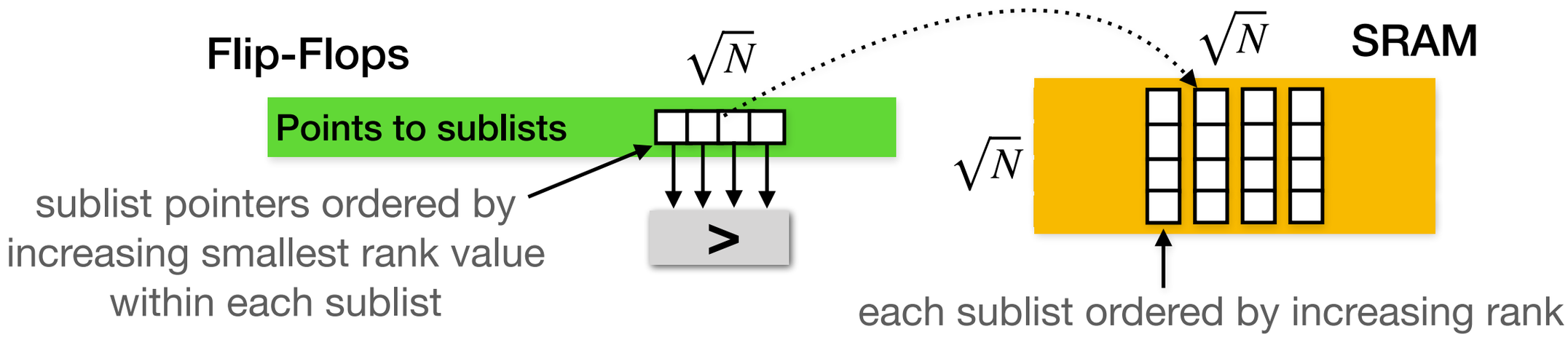


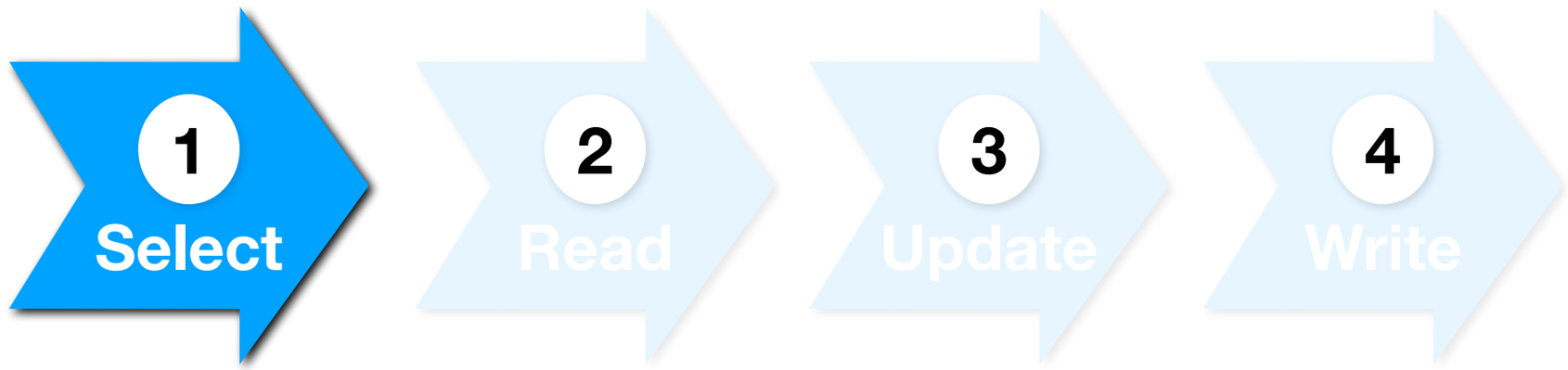


Select correct sublist S

enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

Hardware Architecture

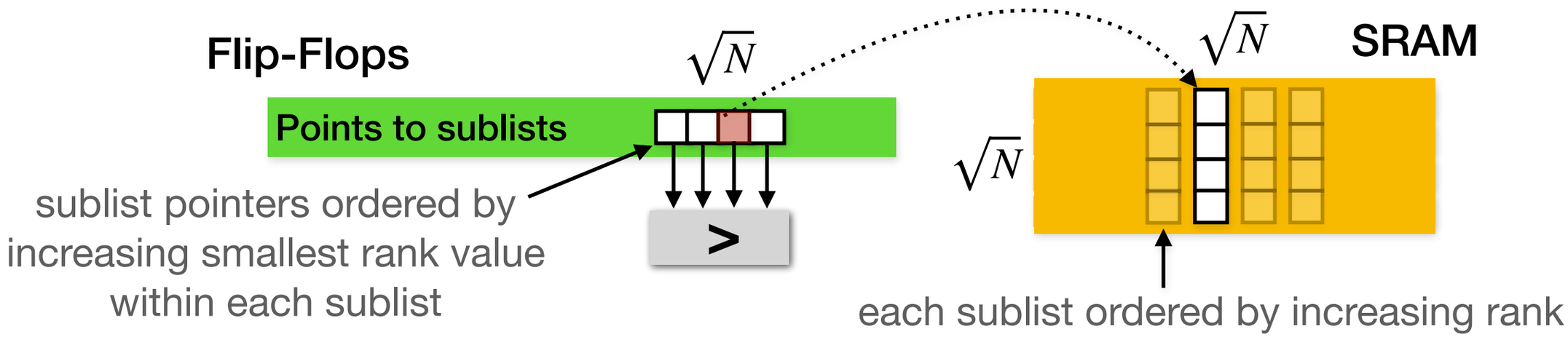


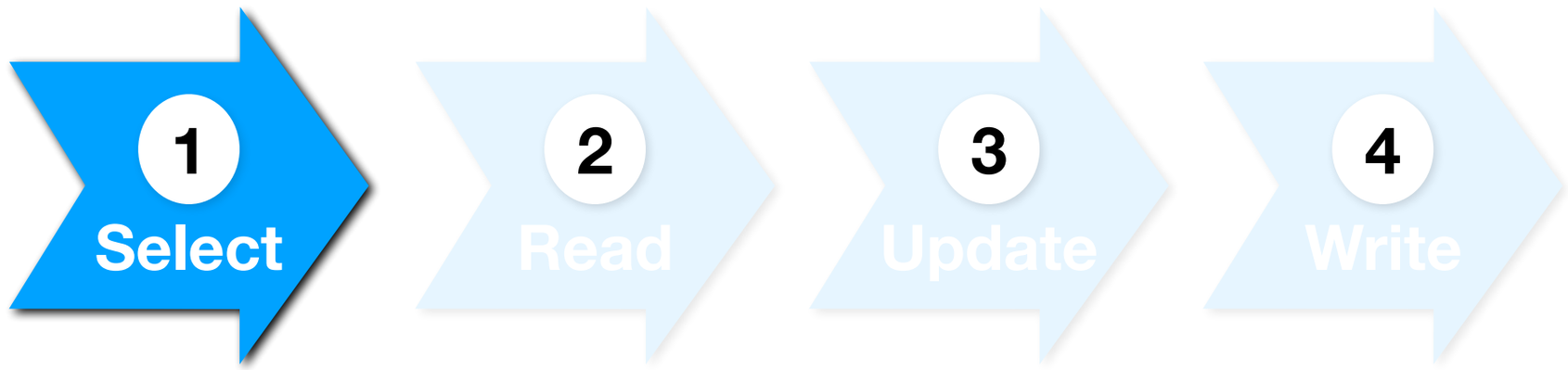


Select correct sublist S

enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

Hardware Architecture

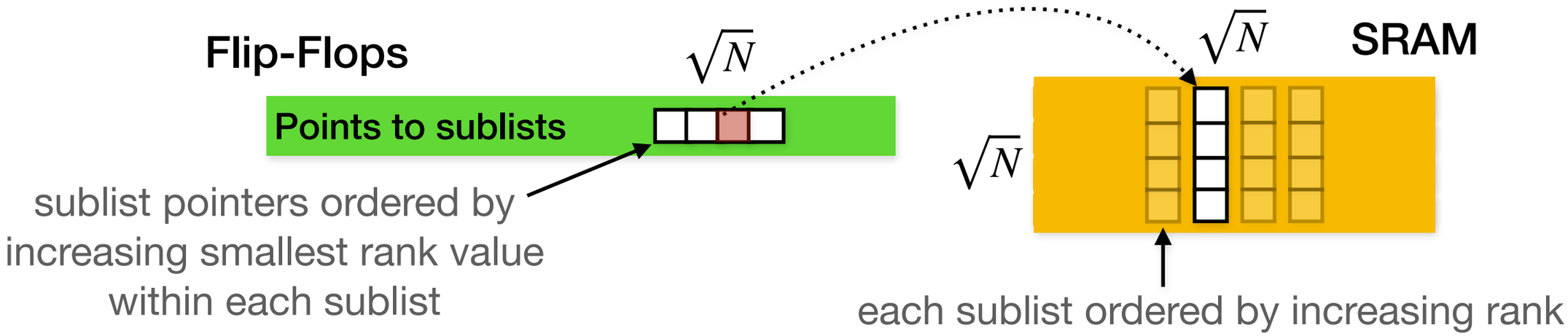


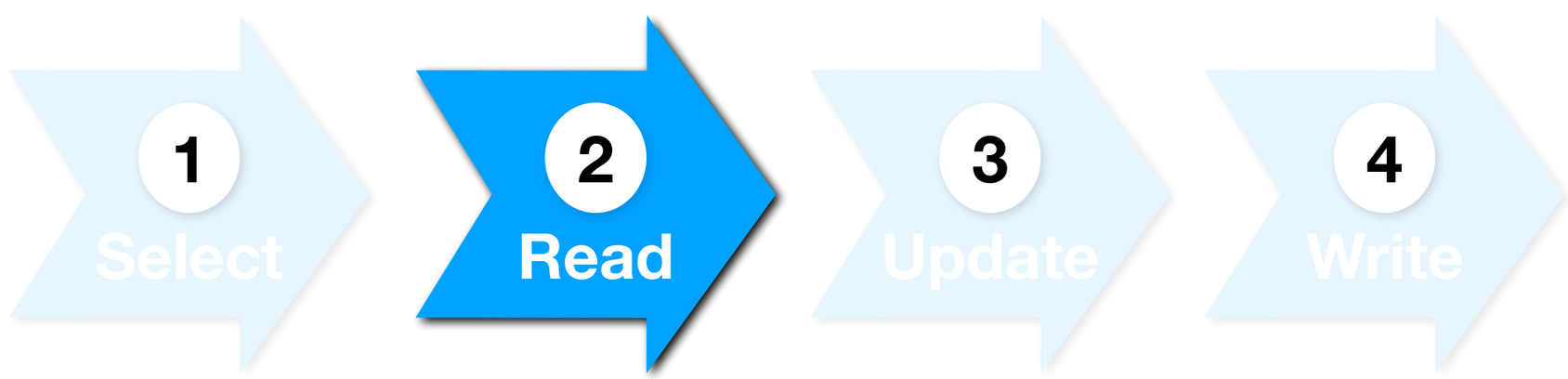


Select correct sublist S

enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

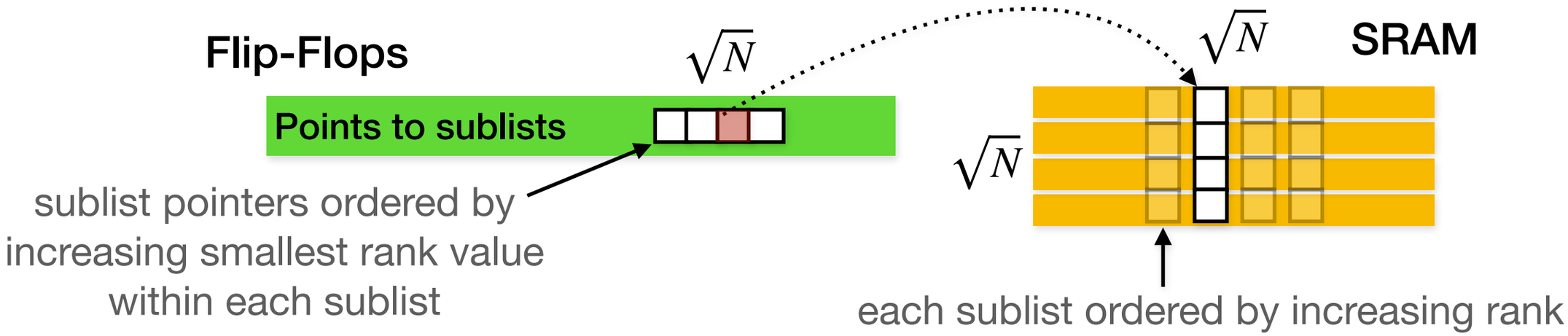
Hardware Architecture

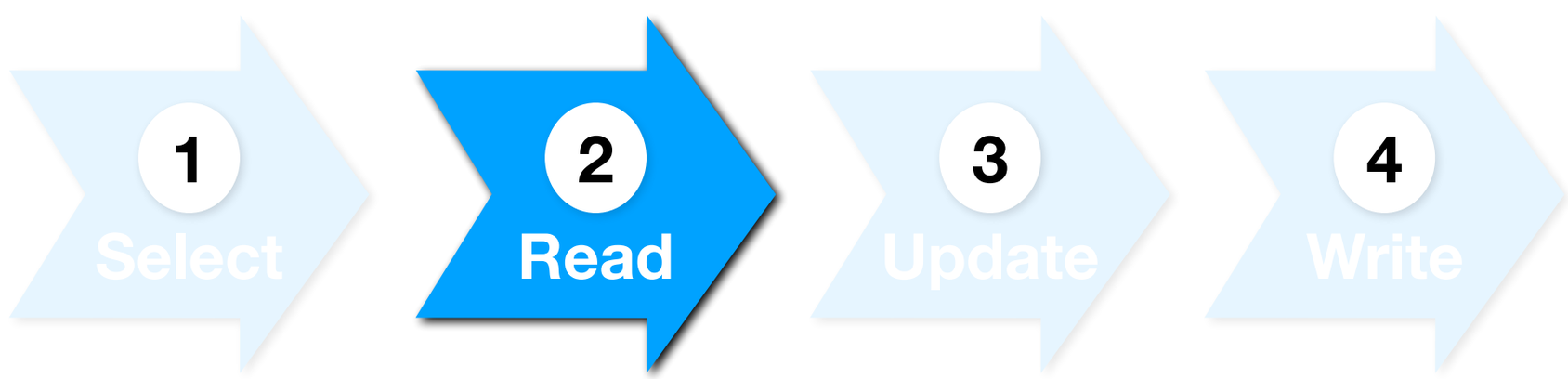




enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

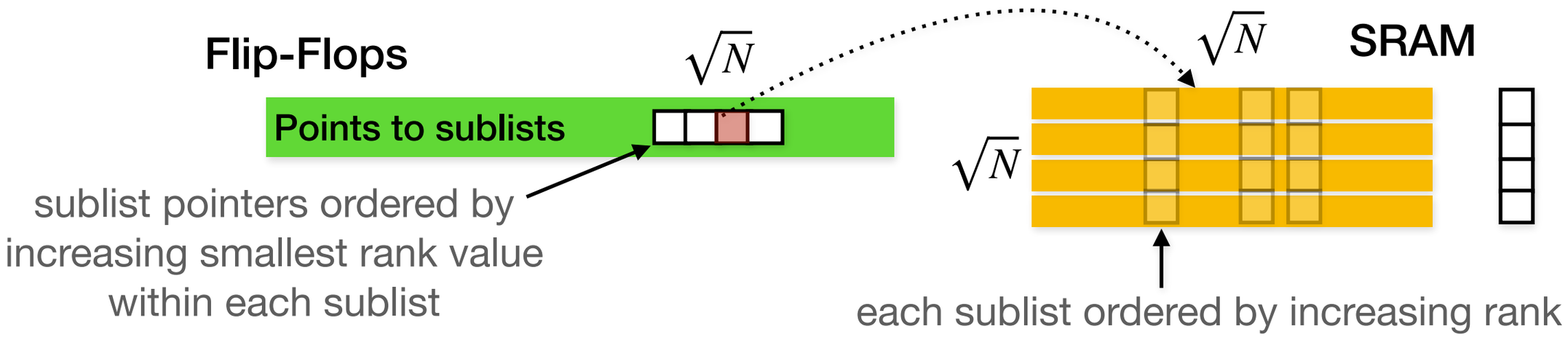
Hardware Architecture

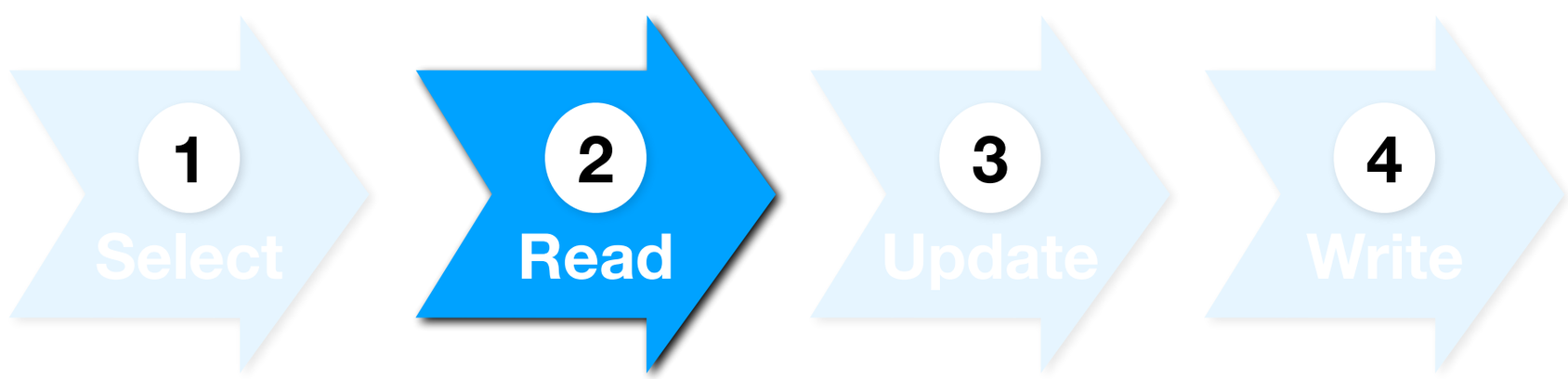




enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

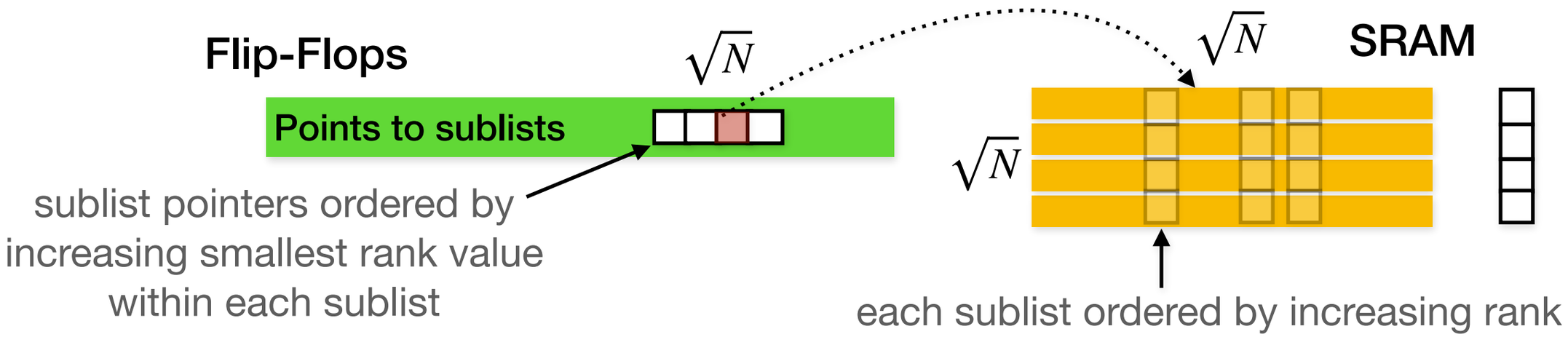
Hardware Architecture

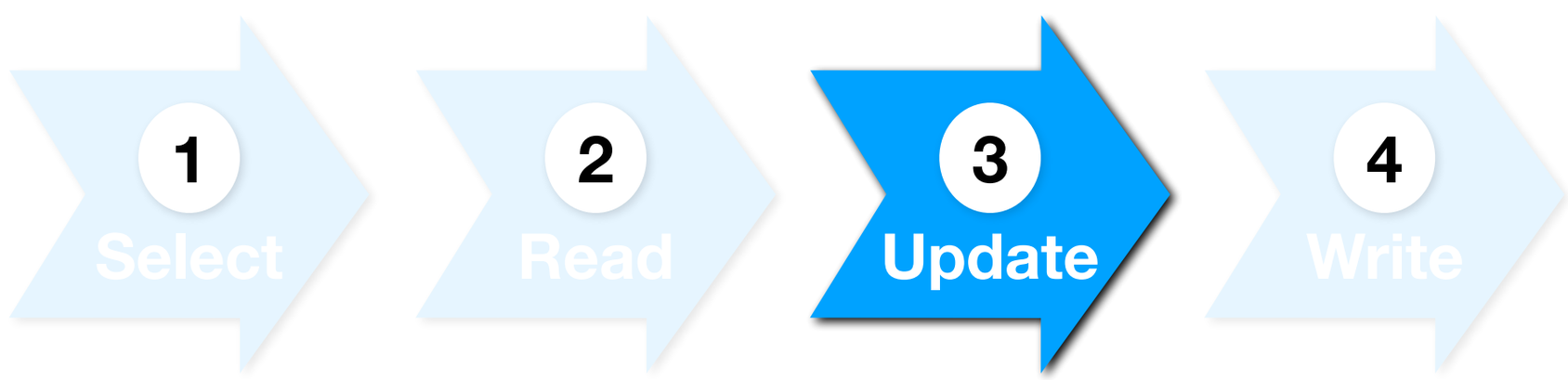




enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

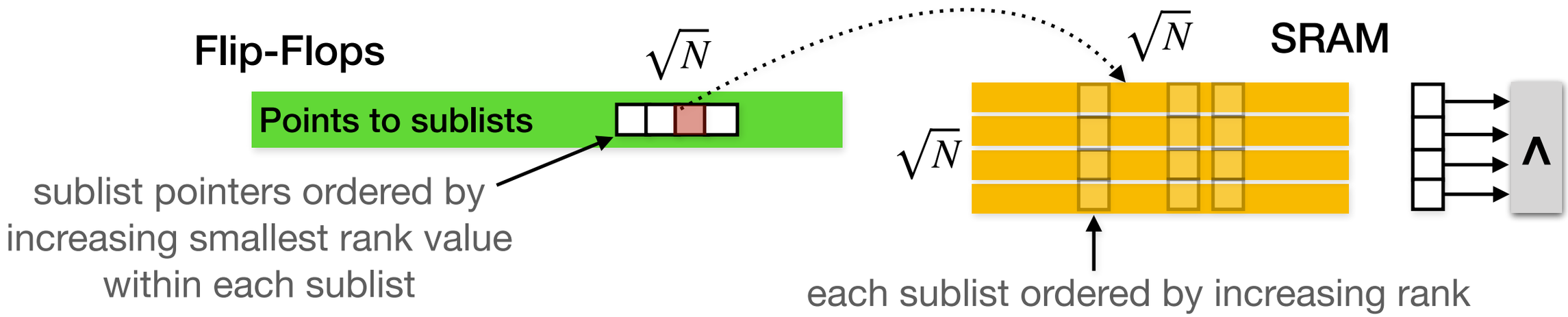
Hardware Architecture

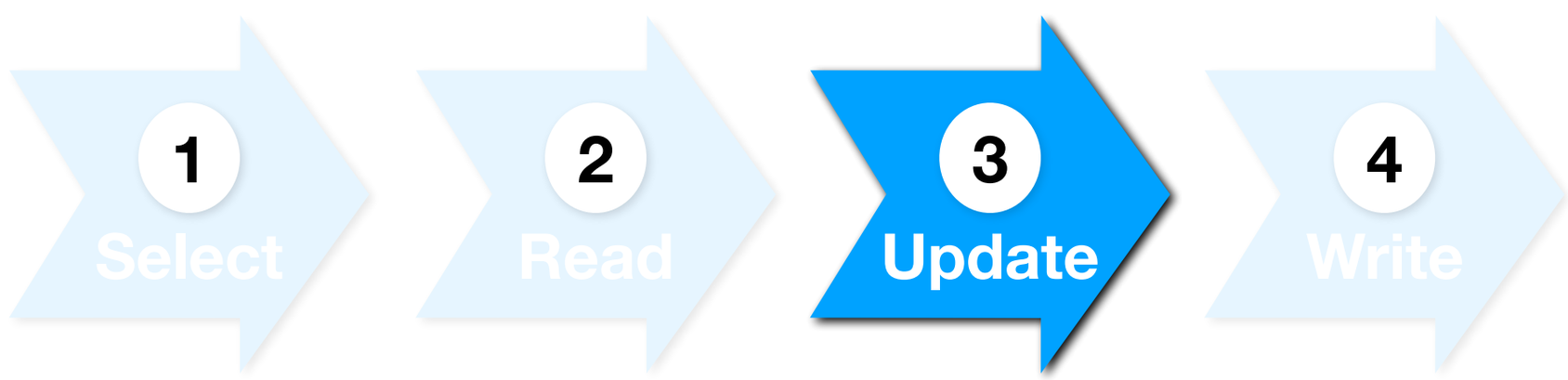




enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

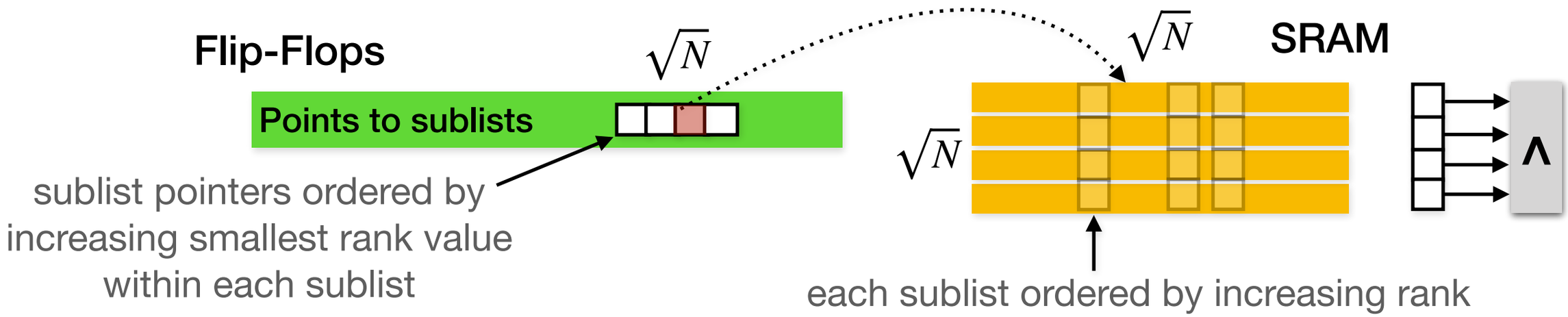
Hardware Architecture

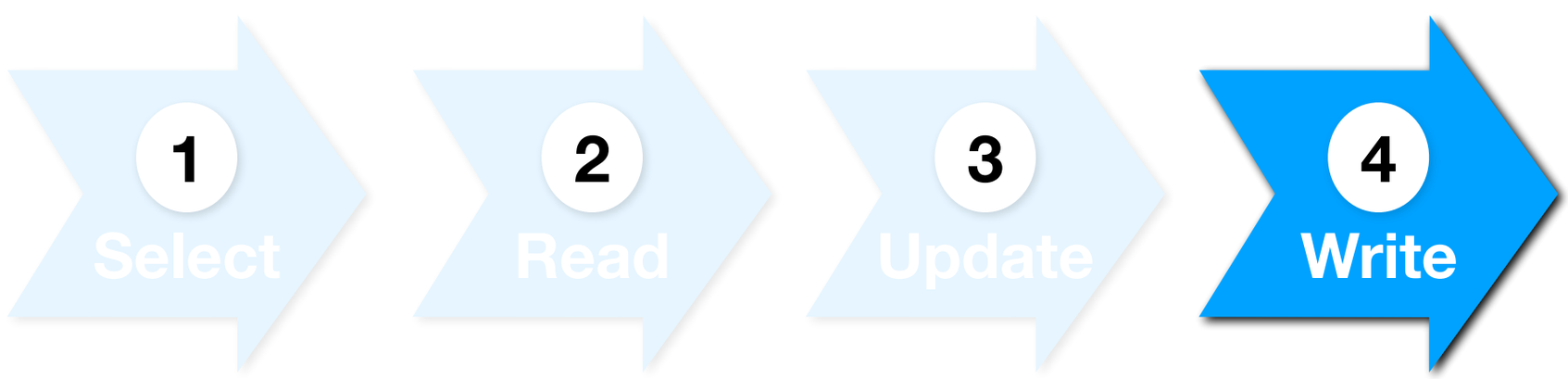




enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

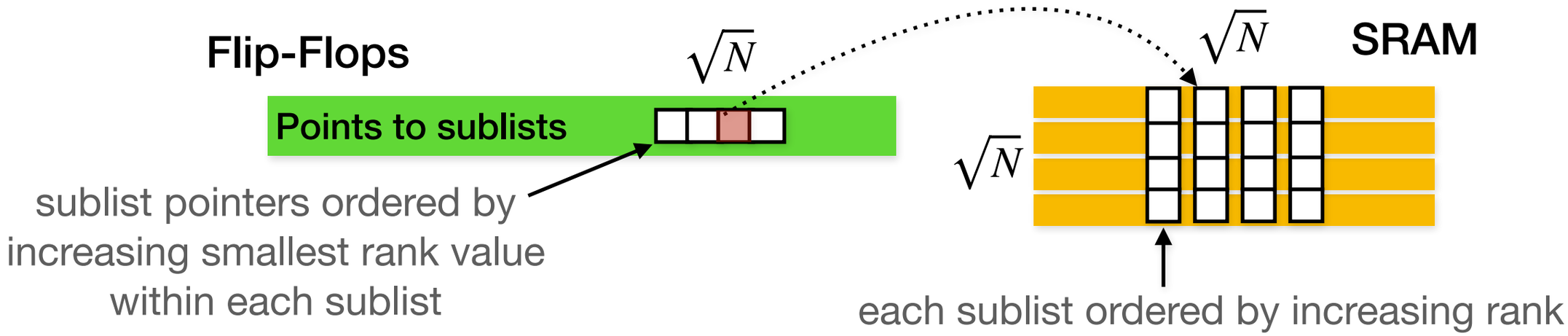
Hardware Architecture

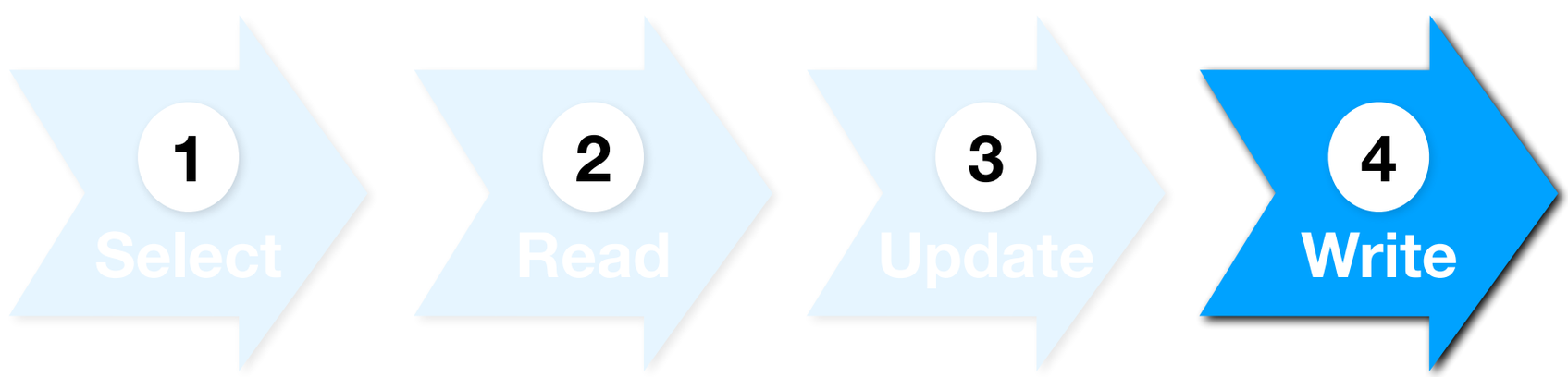




enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

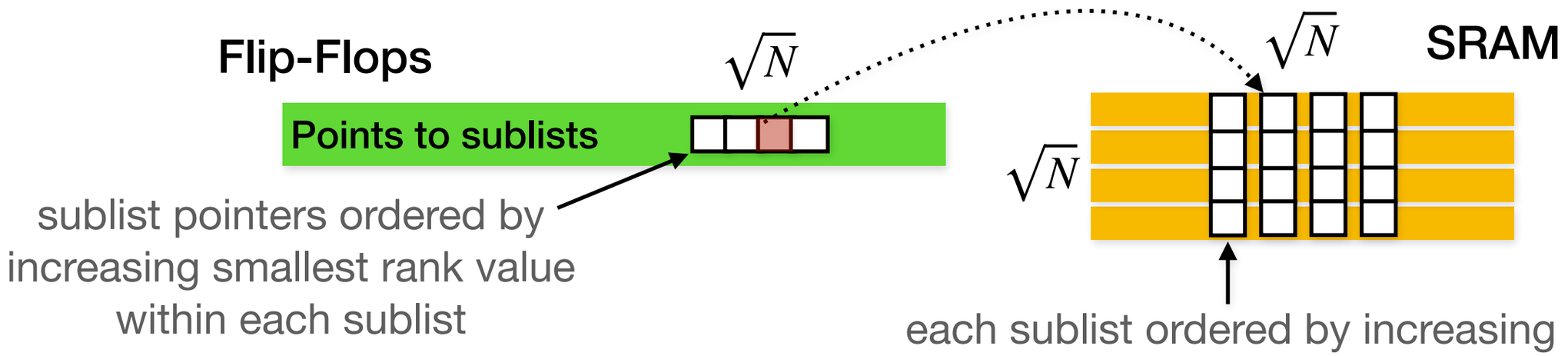
Hardware Architecture

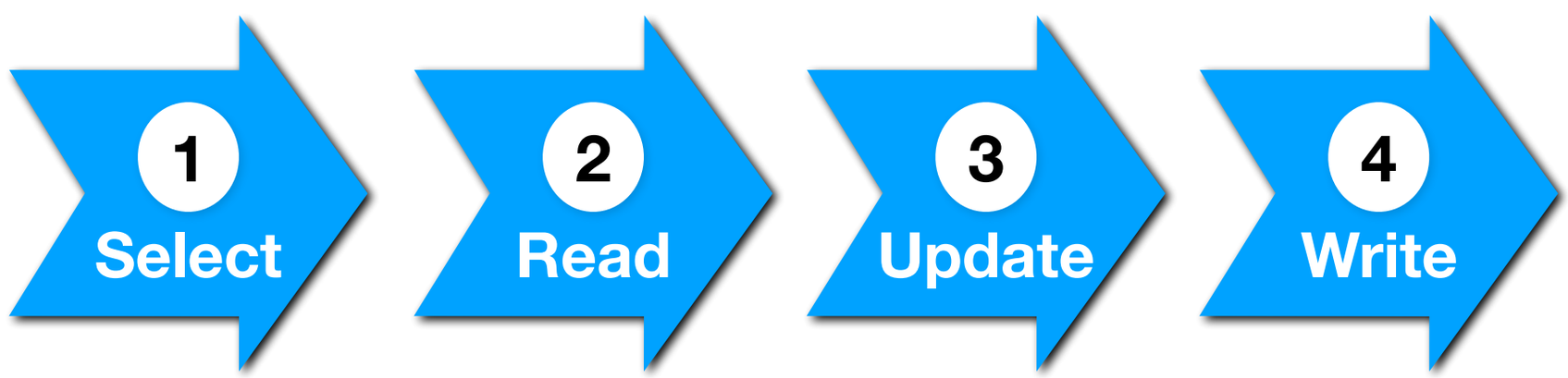




enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

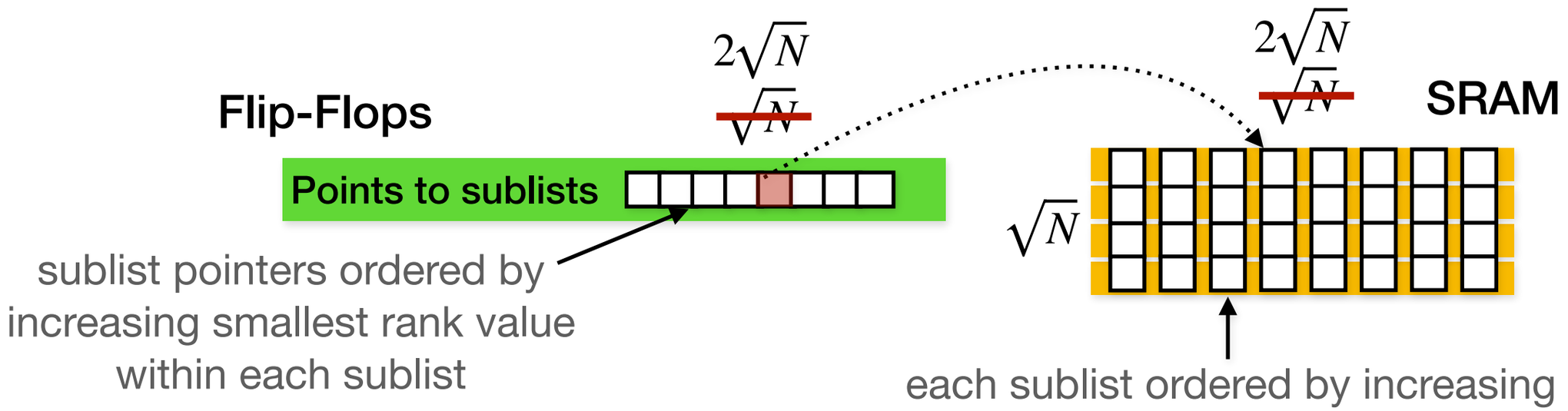
Hardware Architecture

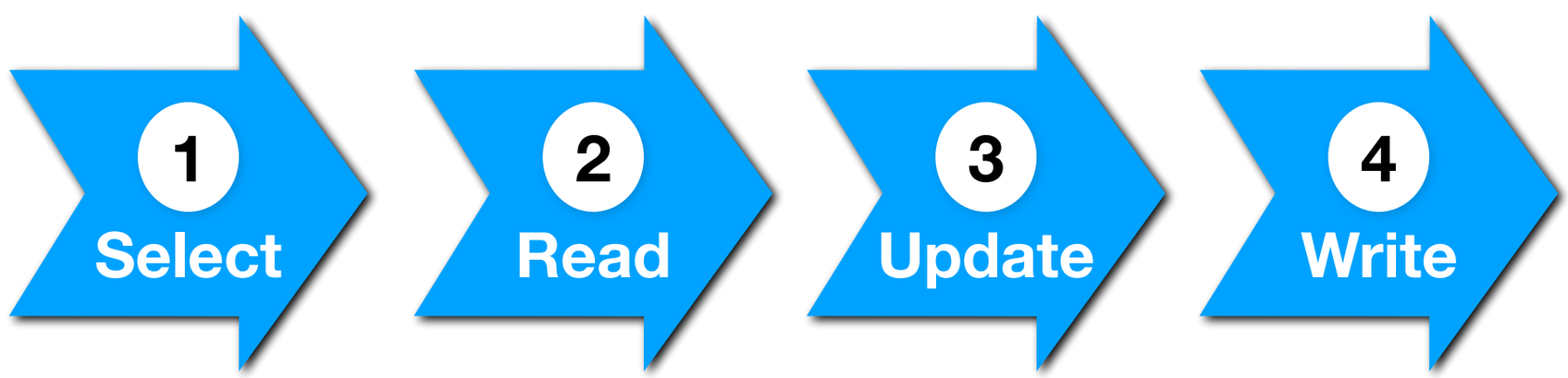




enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

Hardware Architecture





enqueue(f), dequeue(), dequeue(f) each execute in exactly 4 clock cycles

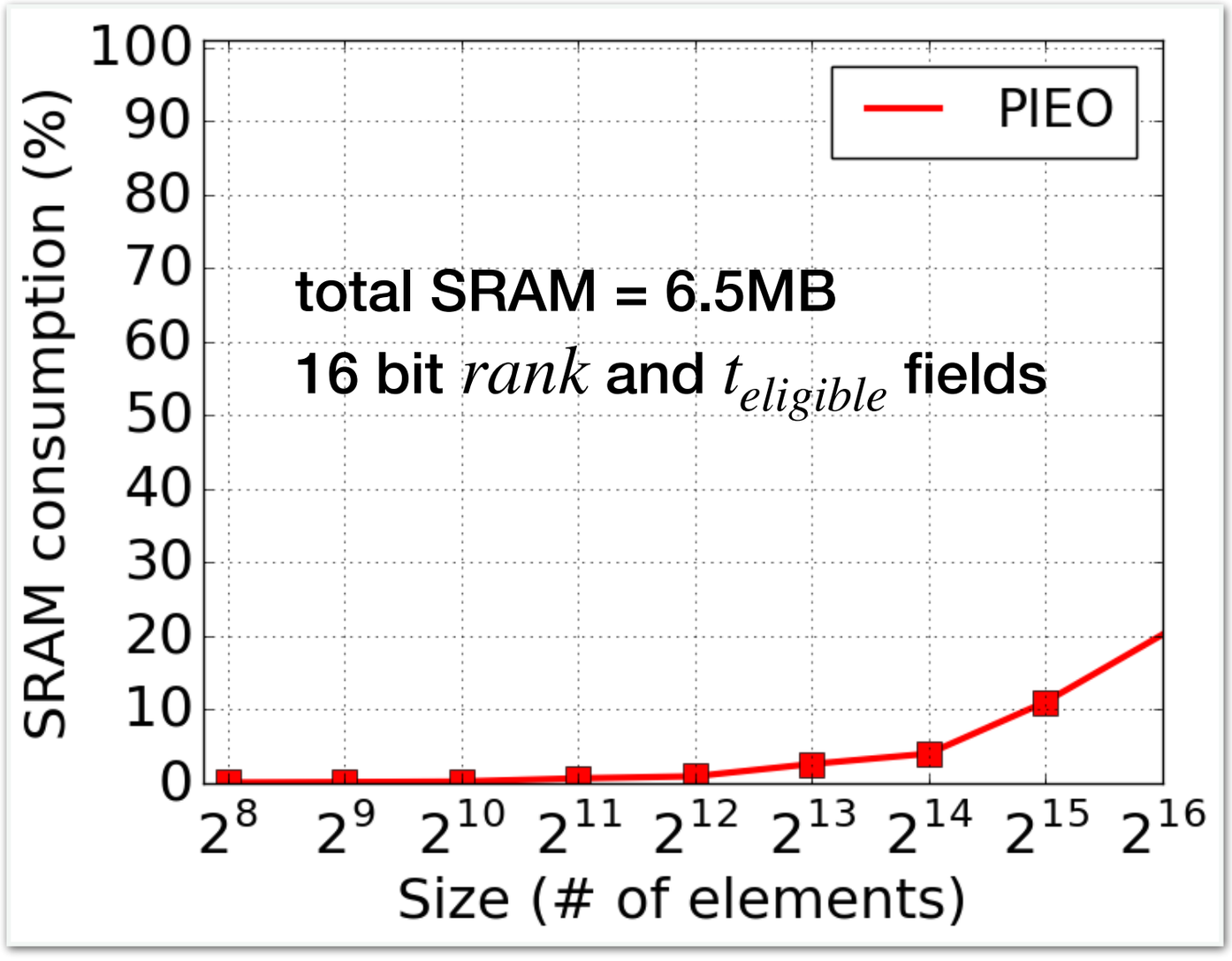
… at the cost of 2x memory overhead

Implementation

- Implemented PIEO scheduler on a **Stratix V FPGA**
  - 234K logic modules (ALMs)
  - 6.5MB SRAM
  - 40Gbps interface bandwidth
- ~1300 LOCs in System Verilog

FPGA code for the implementation of PIEO scheduler is available at:
https://github.com/vishal1303/PIEO-Scheduler

Memory Consumption



Speed

PIEO on FPGA : 80 MHz (for 32K flows)

Speed

PIEO on FPGA : 80 MHz (for 32K flows) ⟶ 50ns per primitive operation

Speed

PIEO on FPGA : 80 MHz (for 32K flows) ⟶ 50ns per primitive operation

~200 Gbps scheduling rate for 1500B (MTU) packets

Speed

PIEO on FPGA : 80 MHz (for 32K flows) ⟶ 50ns per primitive operation

~200 Gbps scheduling rate for 1500B (MTU) packets

PIEO on ASIC : >1 GHz (expected)

Speed

PIEO on FPGA : 80 MHz (for 32K flows) ⟶ 50ns per primitive operation

~200 Gbps scheduling rate for 1500B (MTU) packets

PIEO on ASIC : >1 GHz (expected) ⟶ <4ns per primitive operation

Speed

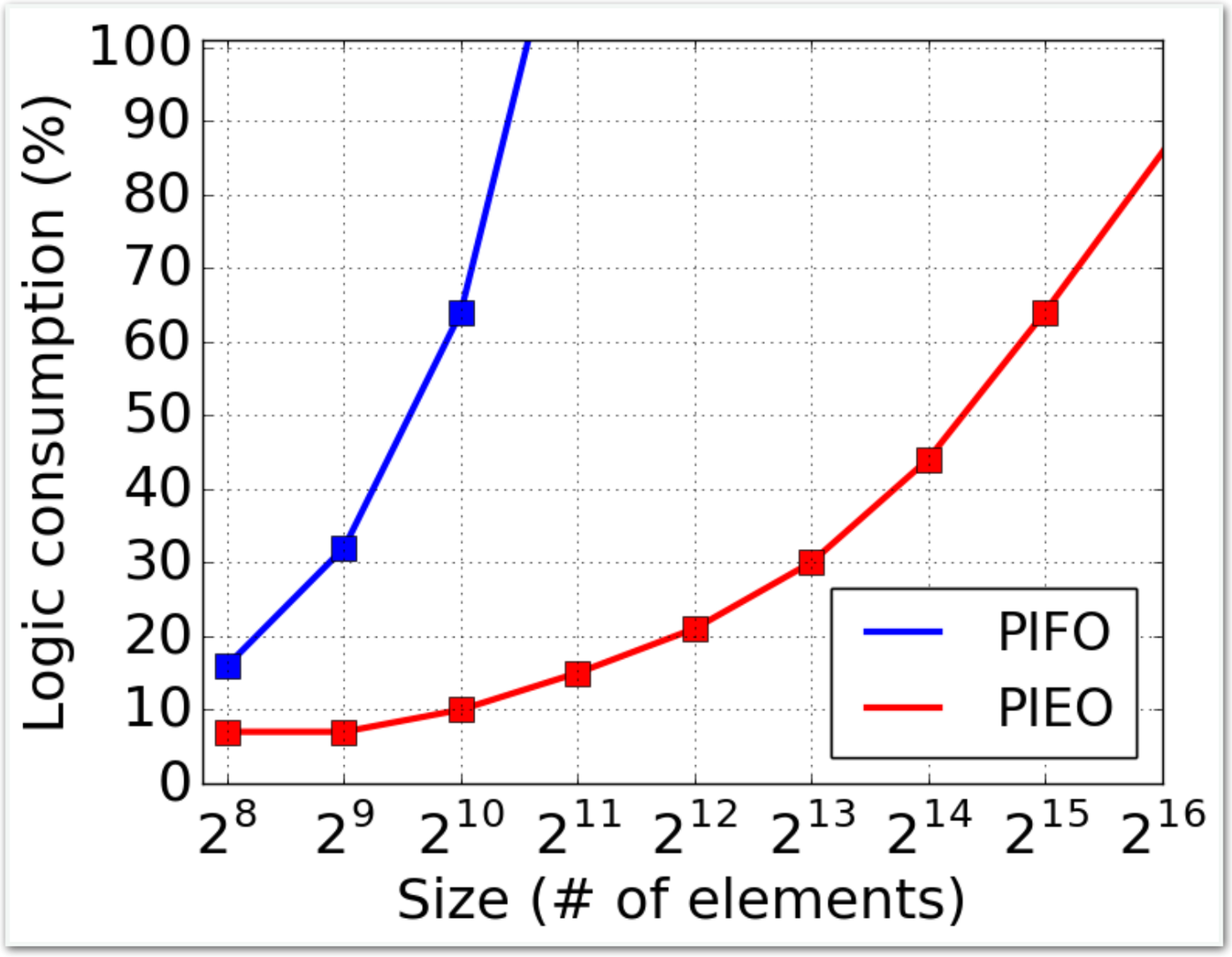
PIEO on FPGA : 80 MHz (for 32K flows) ⟶ 50ns per primitive operation

~200 Gbps scheduling rate for 1500B (MTU) packets

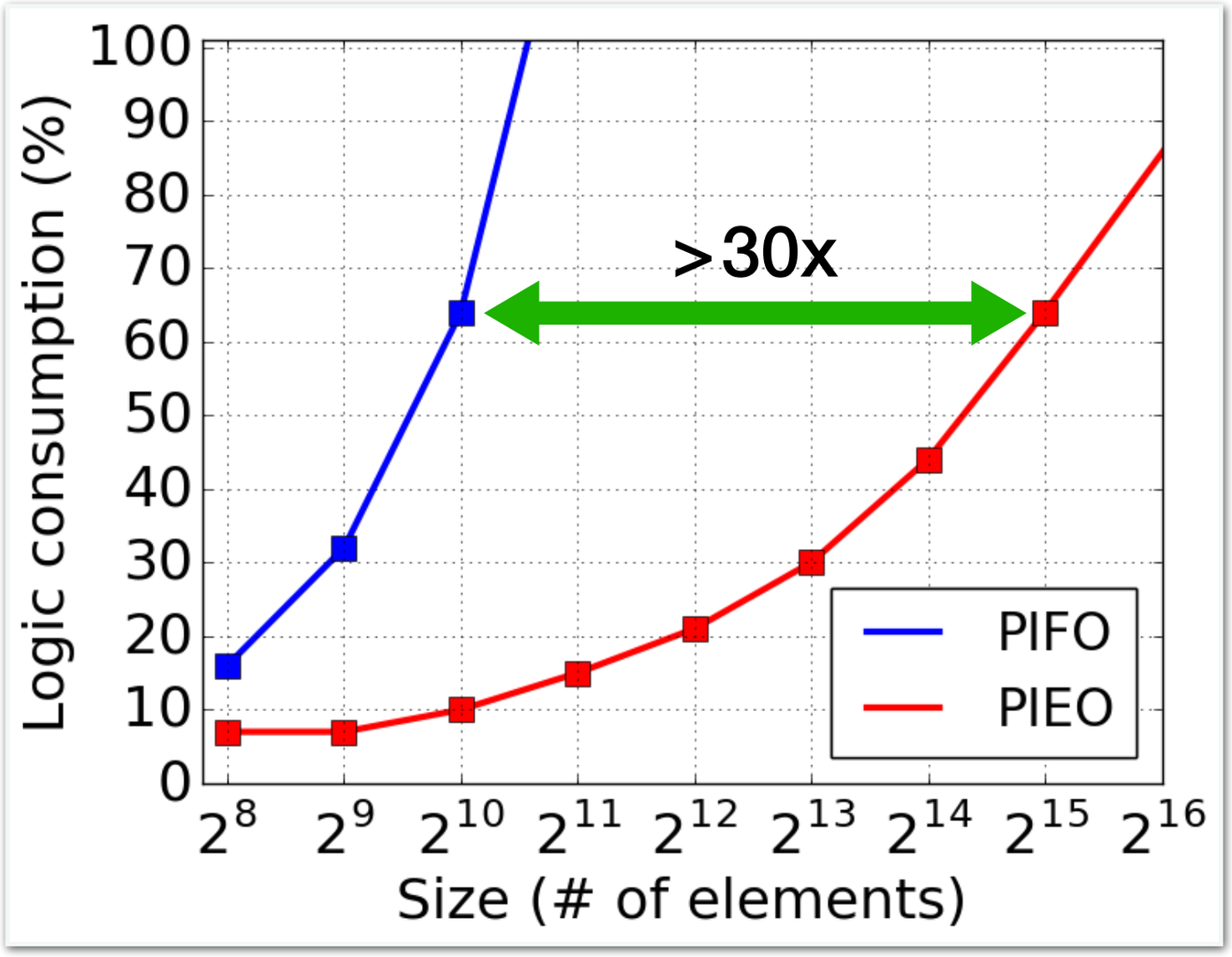
PIEO on ASIC : >1 GHz (expected) ⟶ <4ns per primitive operation

>3 Tbps scheduling rate for 1500B (MTU) packets

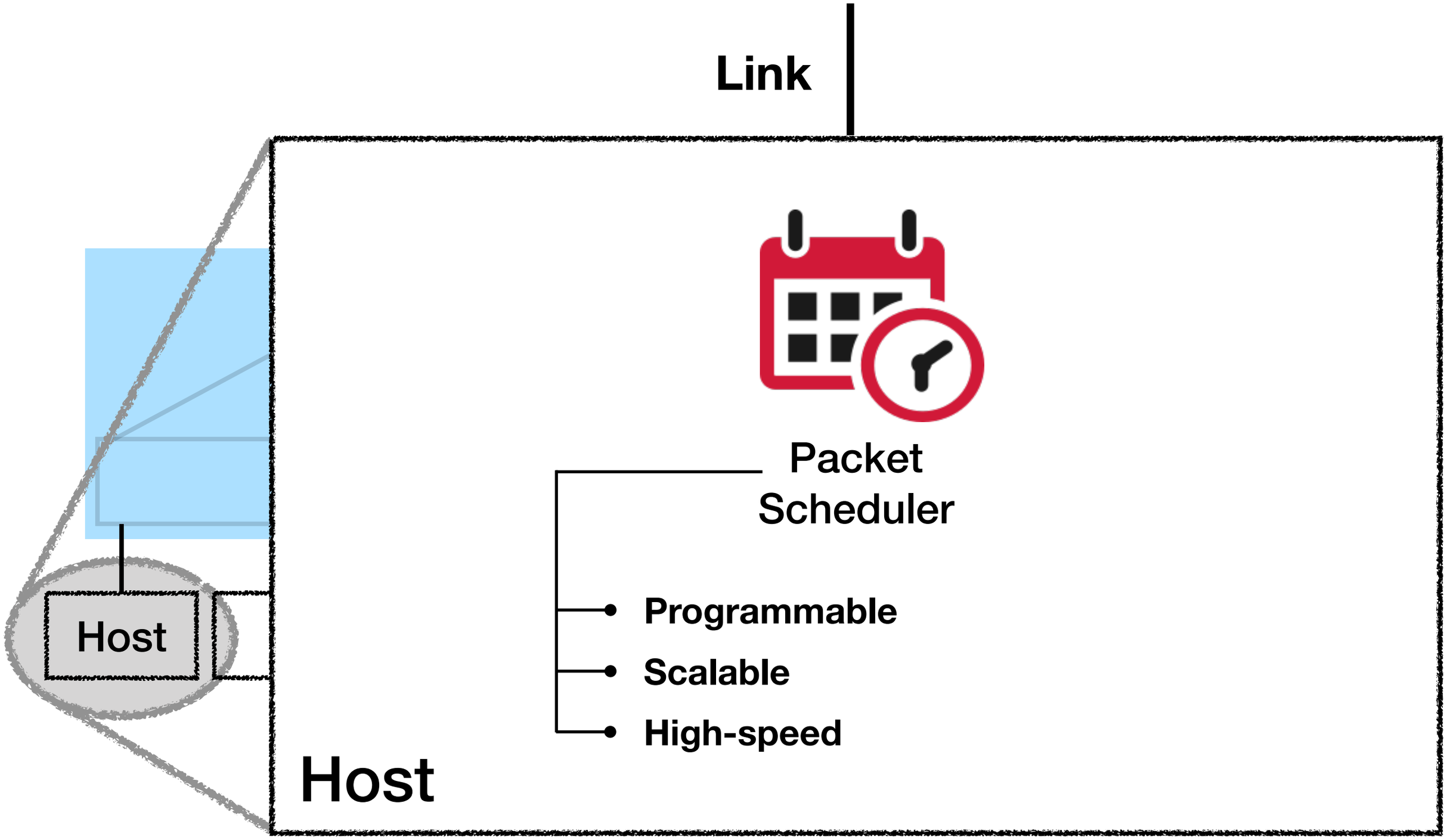
Scalability



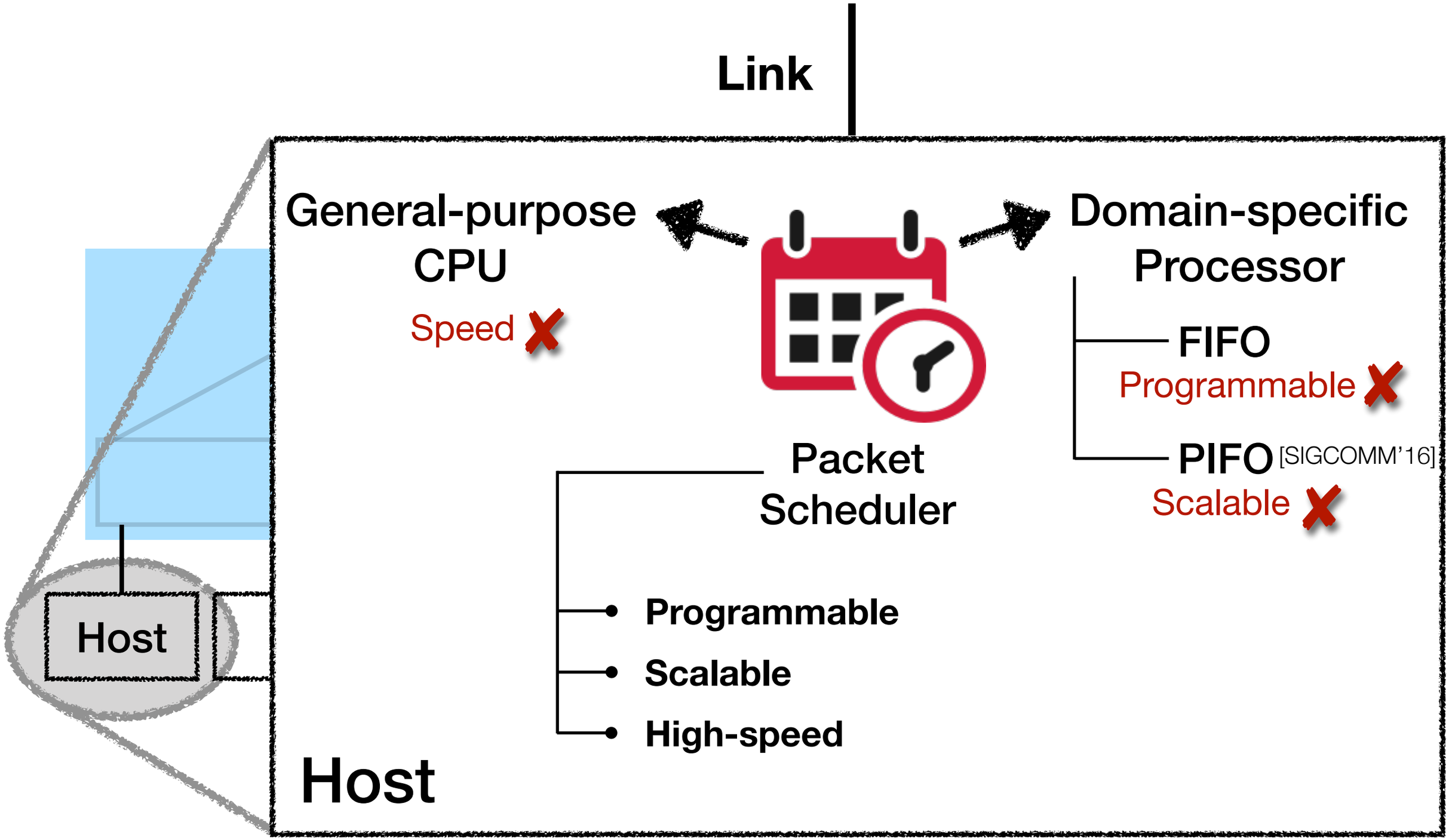
Scalability



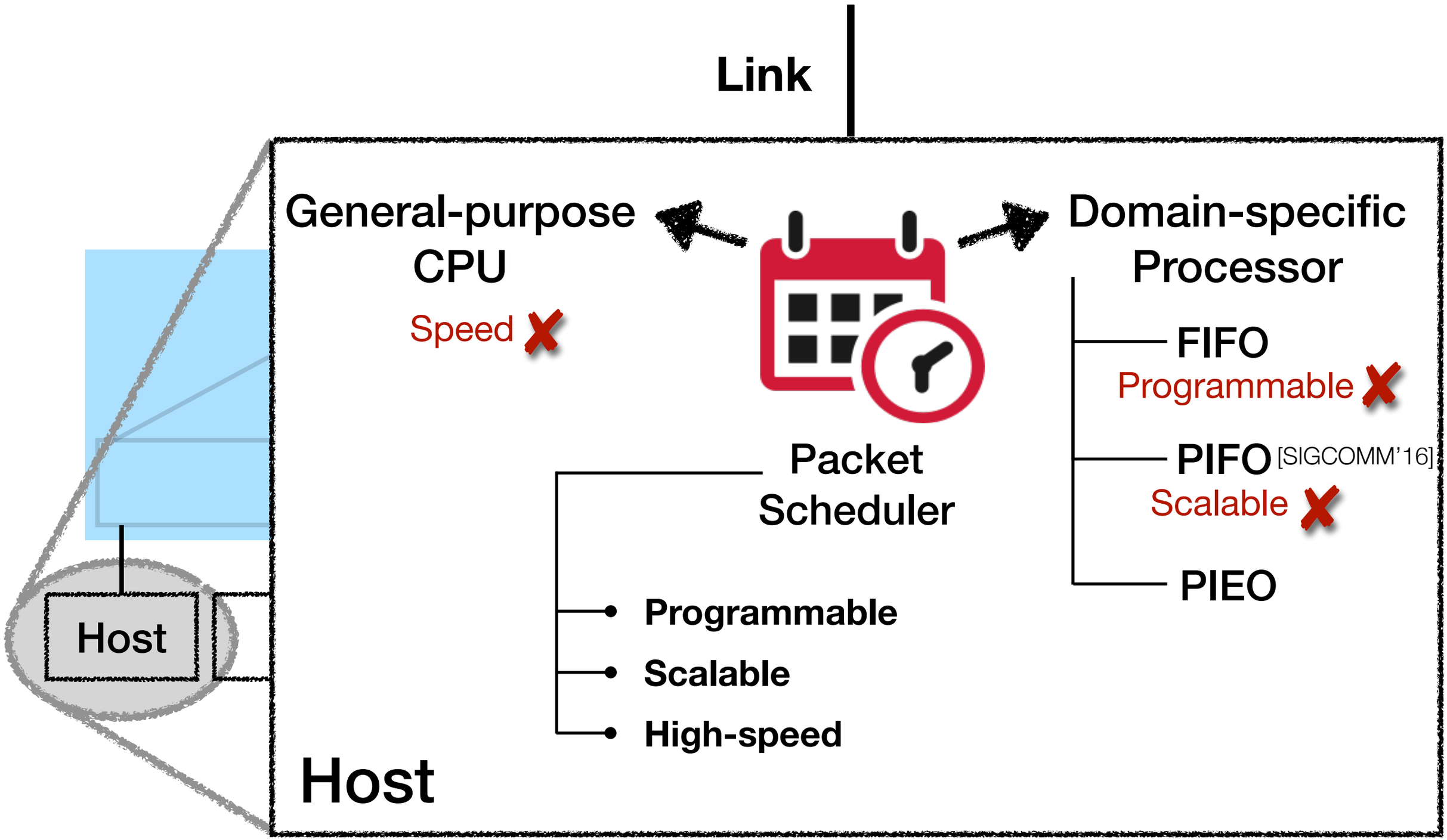
Part I : Summary



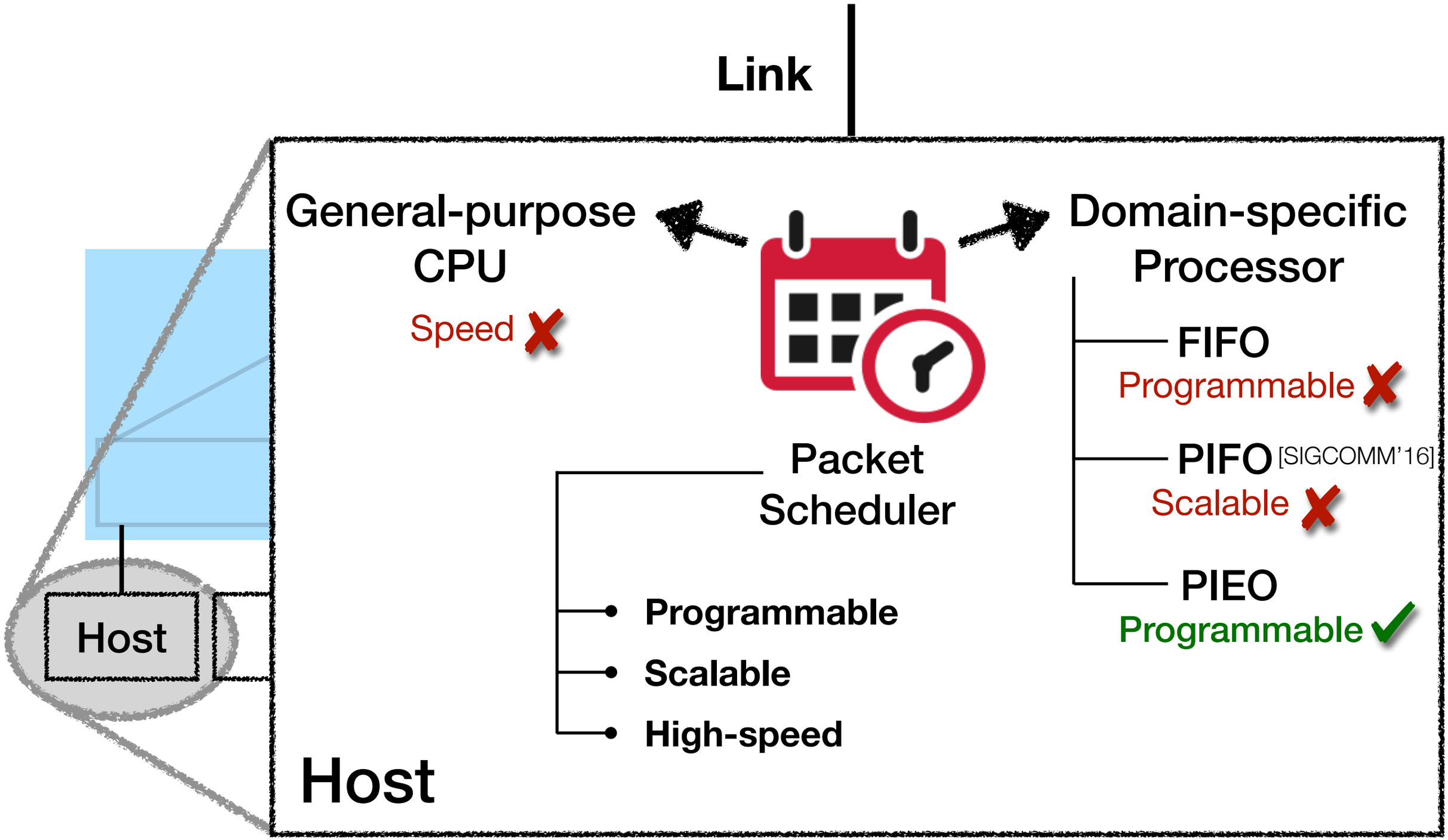
Part I : Summary



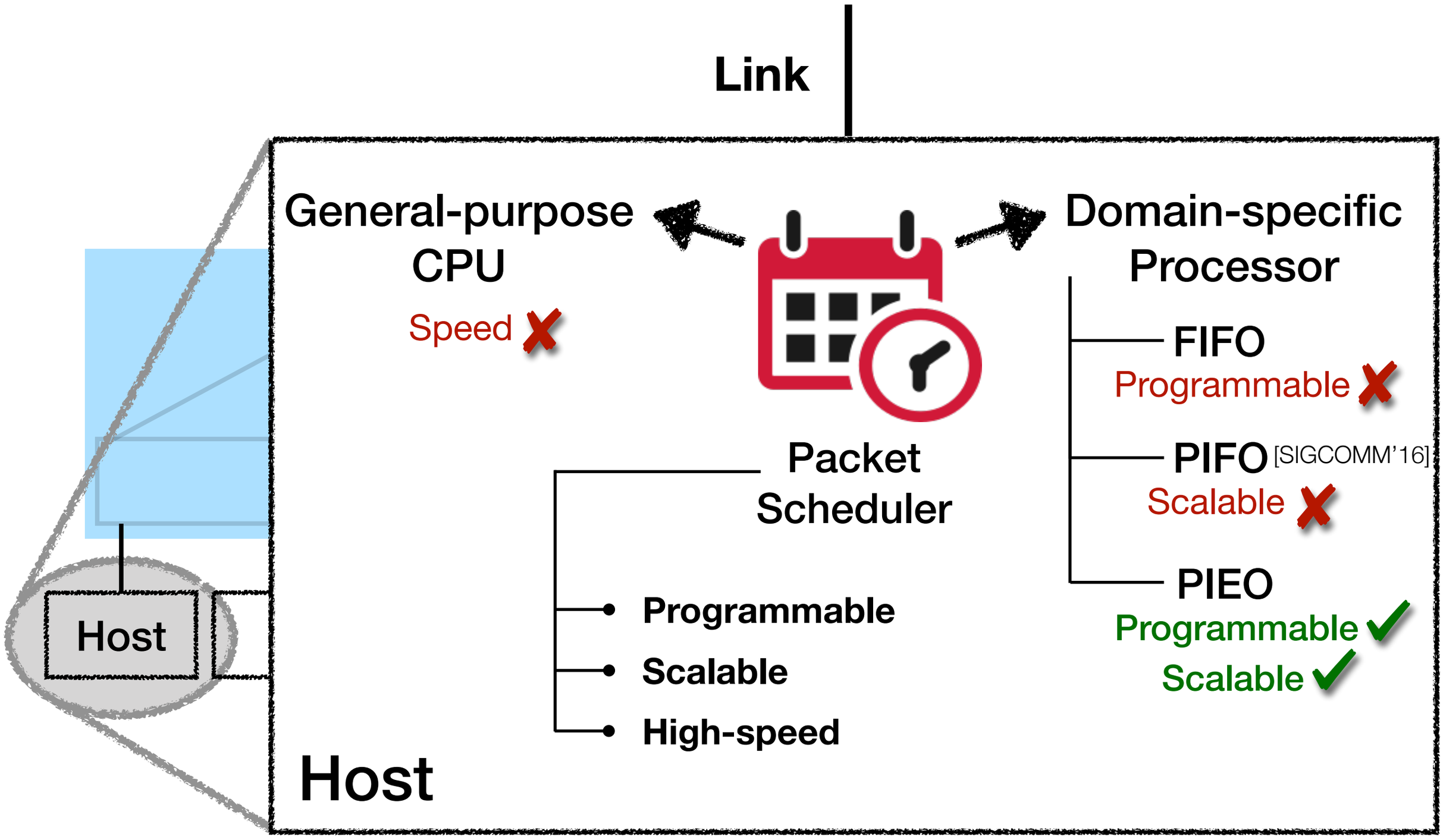
Part I : Summary



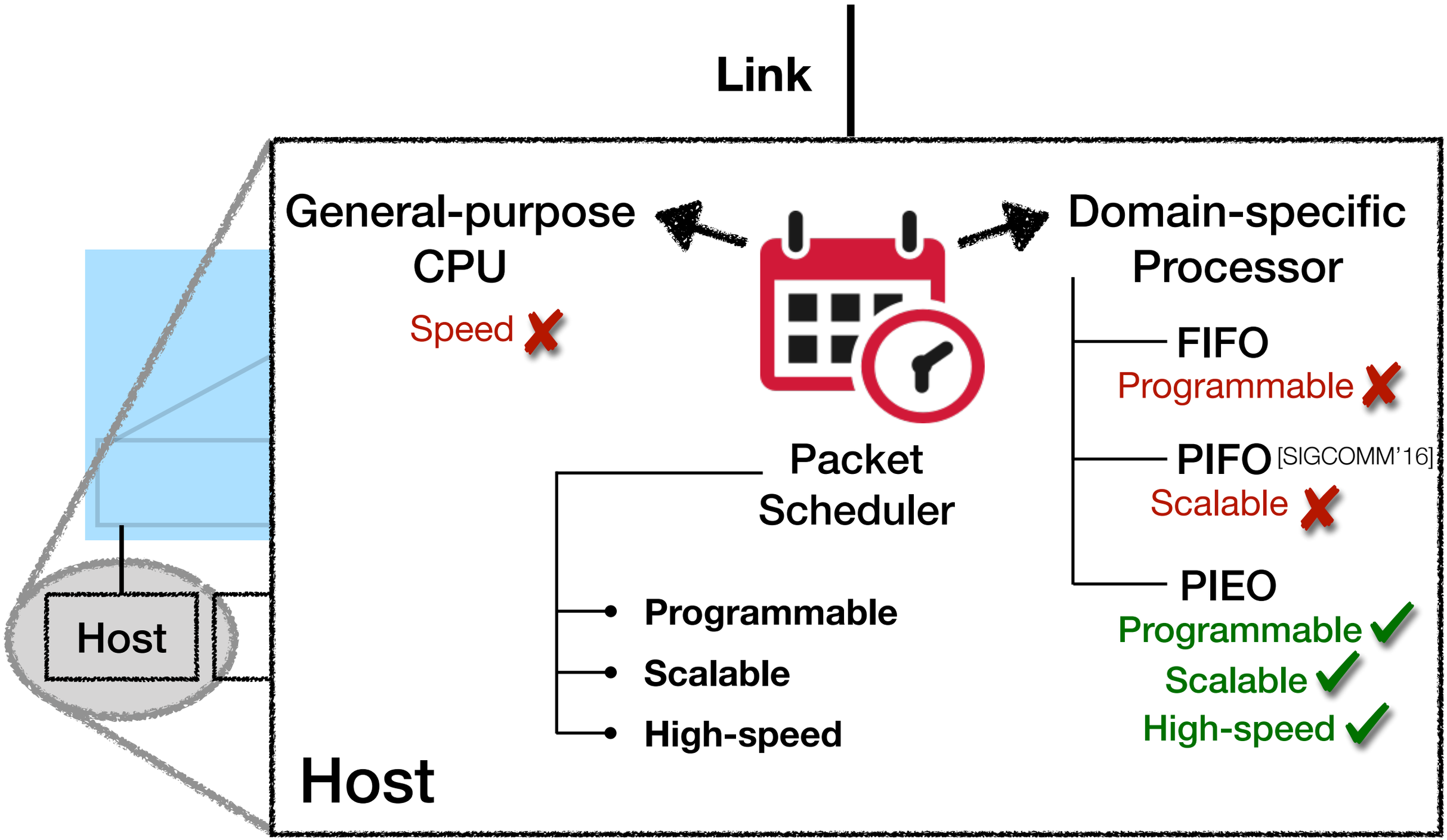
Part I : Summary



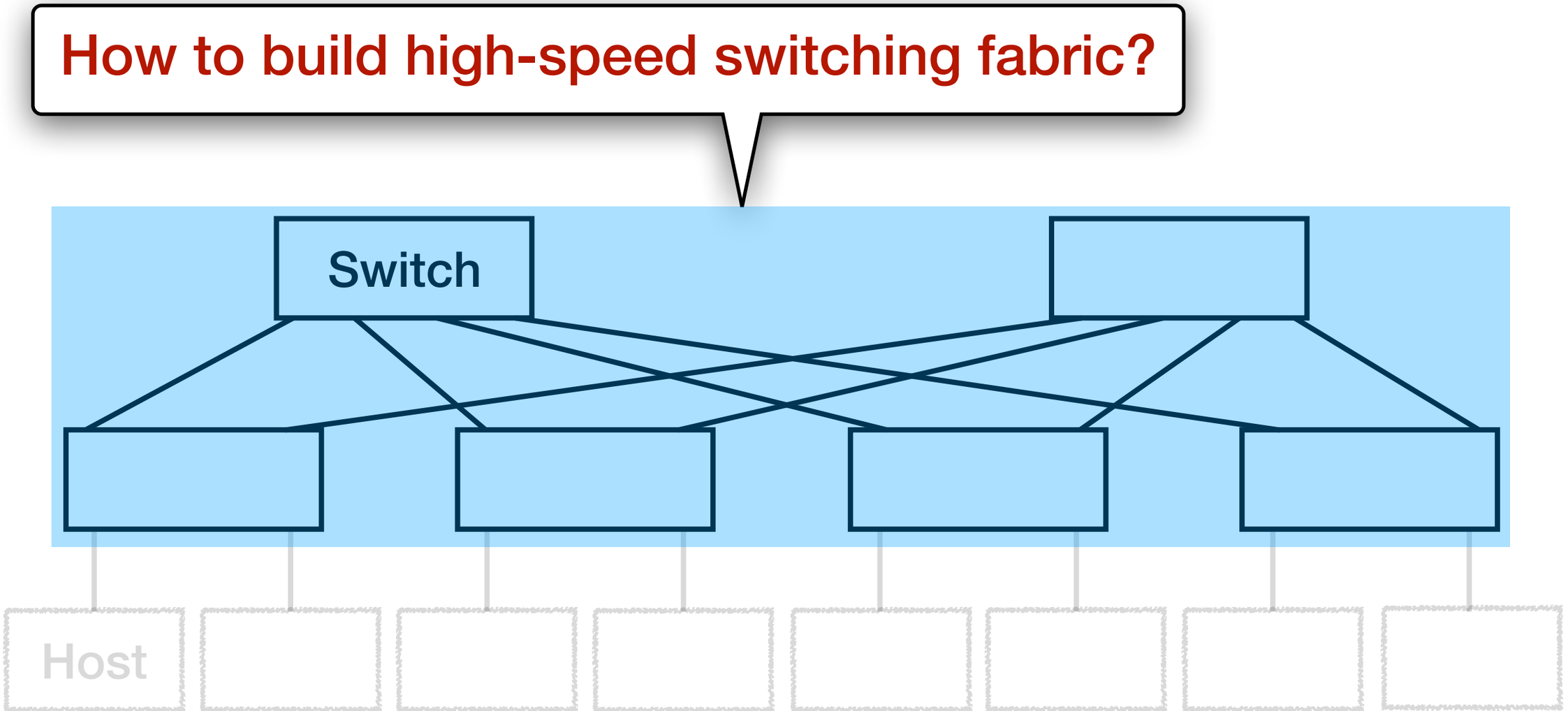
Part I : Summary



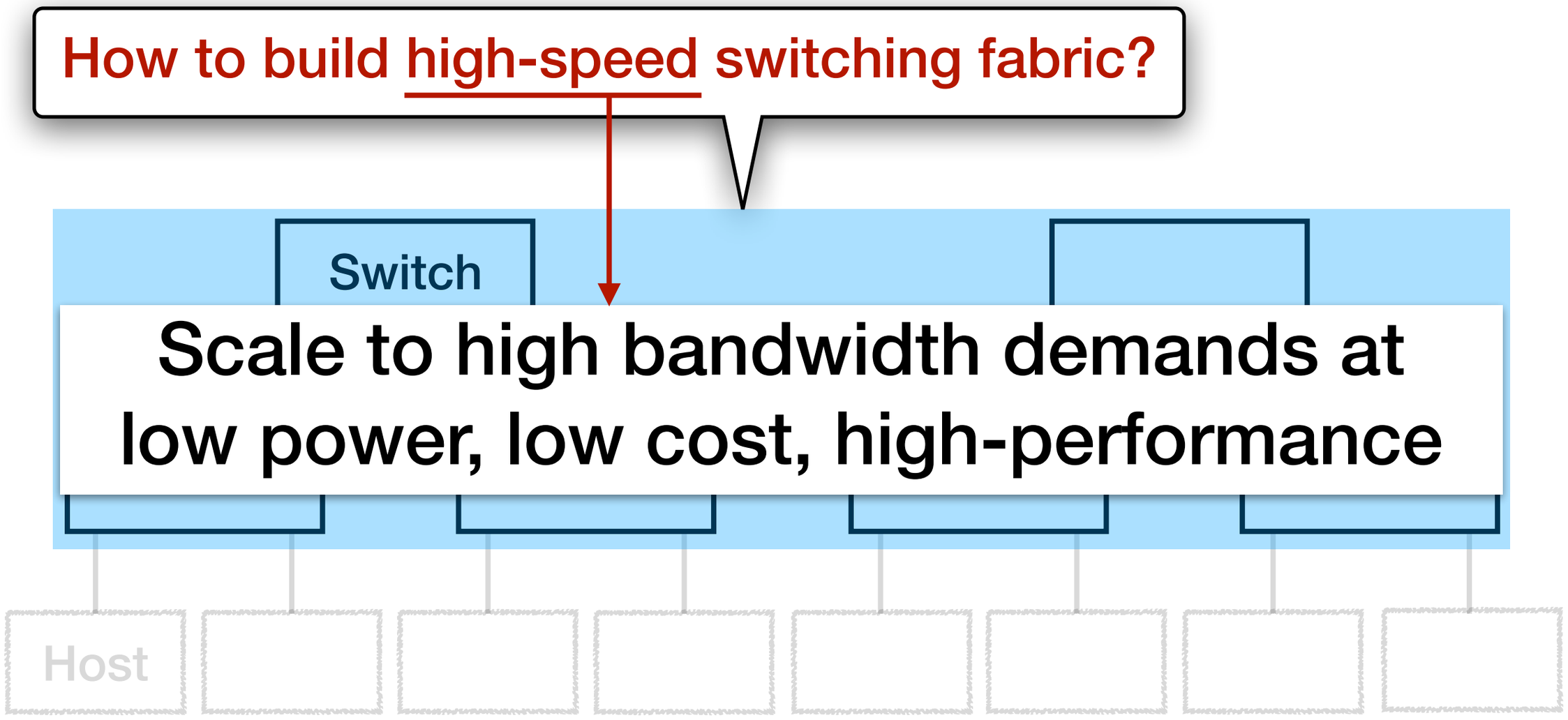
Part I : Summary



Part II : Switching Fabric



Part II : Switching Fabric



Part II : Switching Fabric

How to build high-speed switching fabric?

Switch

Scale to high bandwidth demands at low power, low cost, high-performance

Host

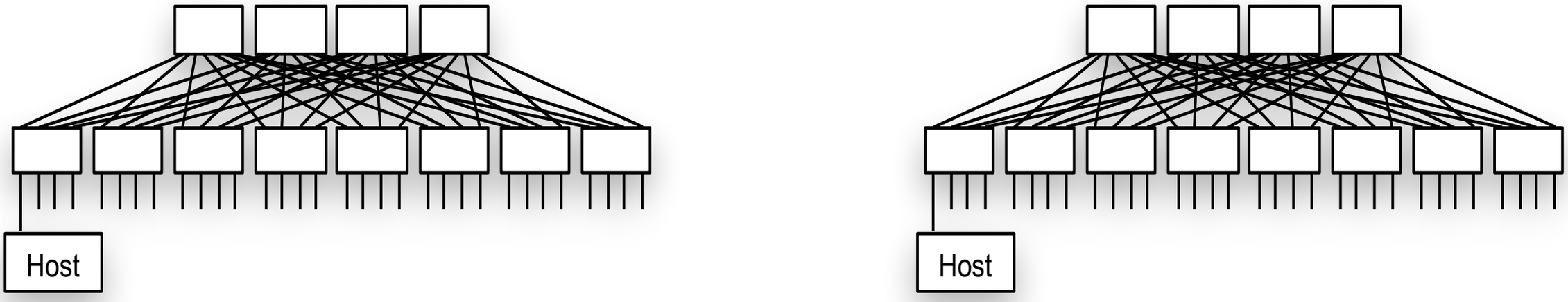
Circuit Switching

Promise of Circuit Switching

Packet Switching

VS.

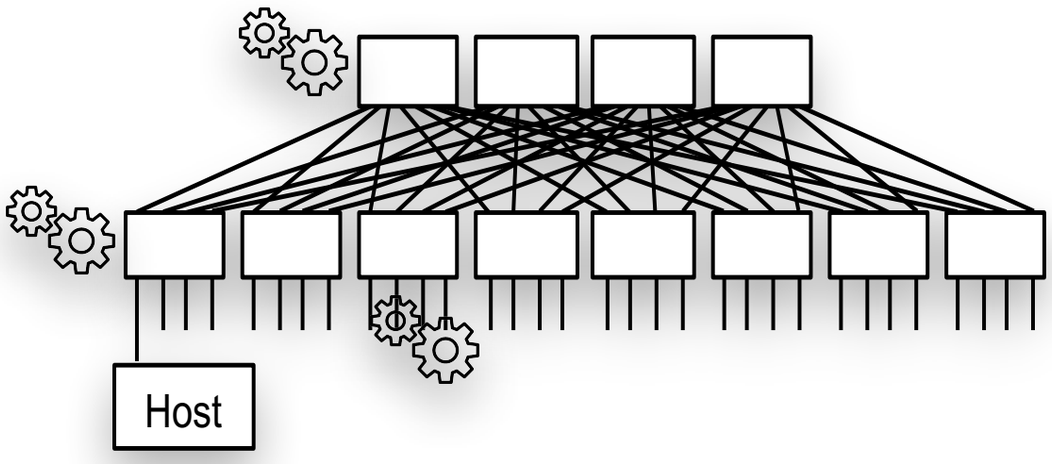
Circuit Switching



Promise of Circuit Switching

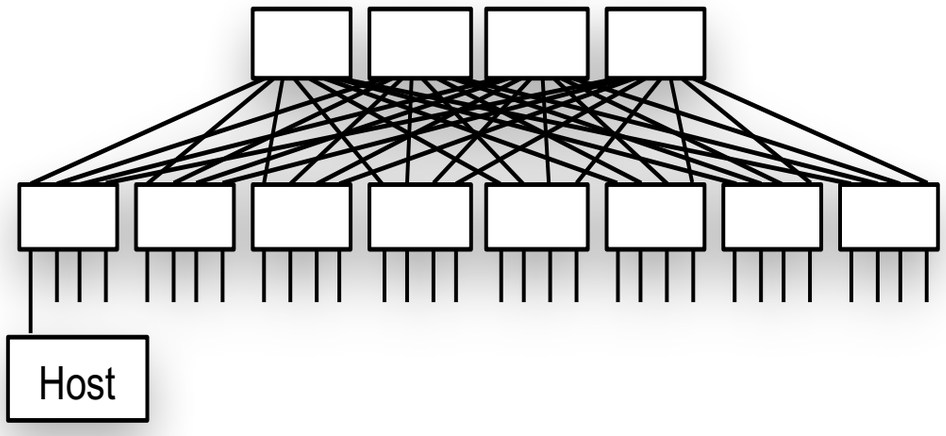
Packet Switching

(Per packet, Per hop Decisions)



VS.

Circuit Switching



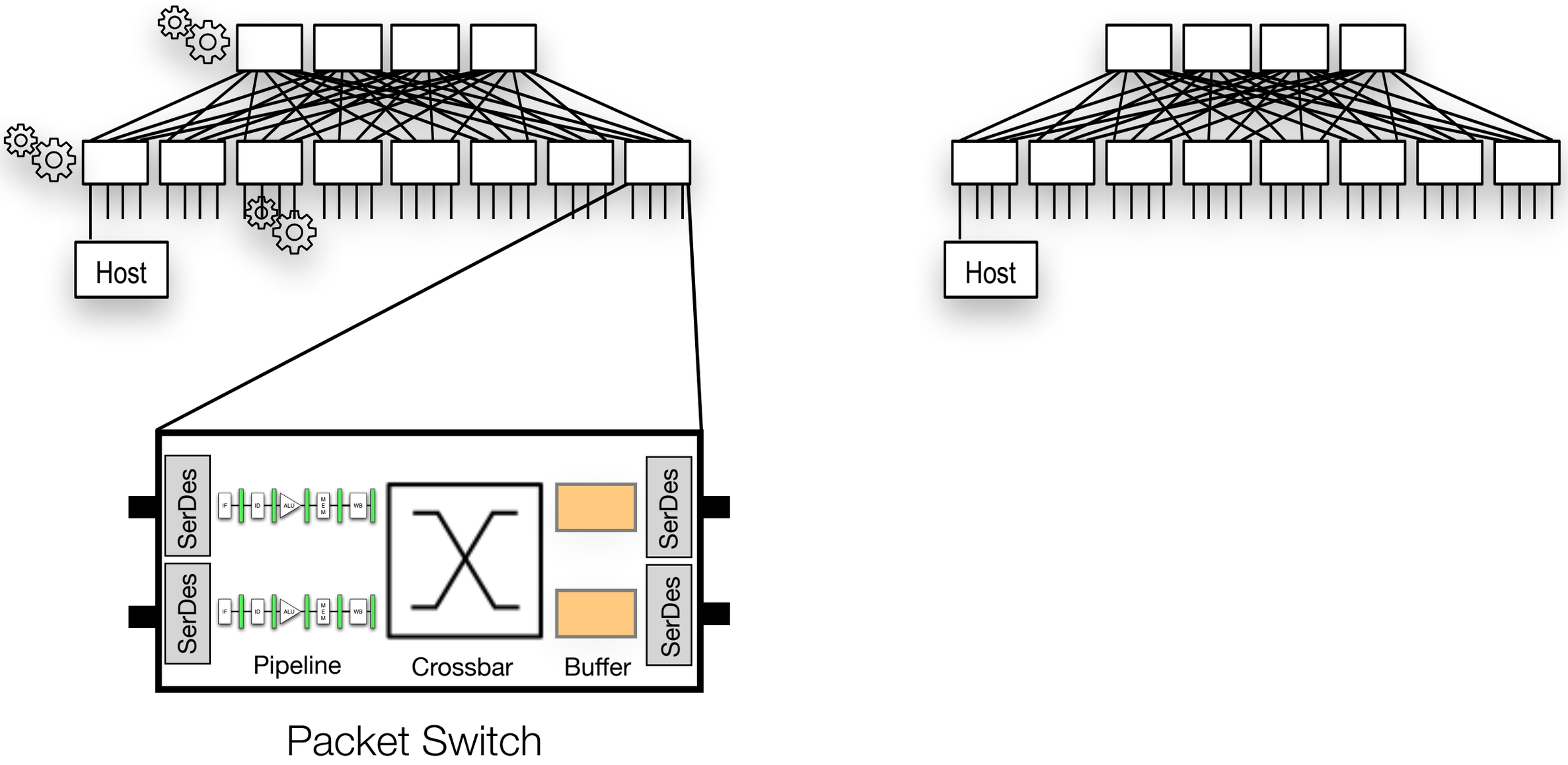
Promise of Circuit Switching

Packet Switching

(Per packet, Per hop Decisions)

VS.

Circuit Switching



Packet Switch

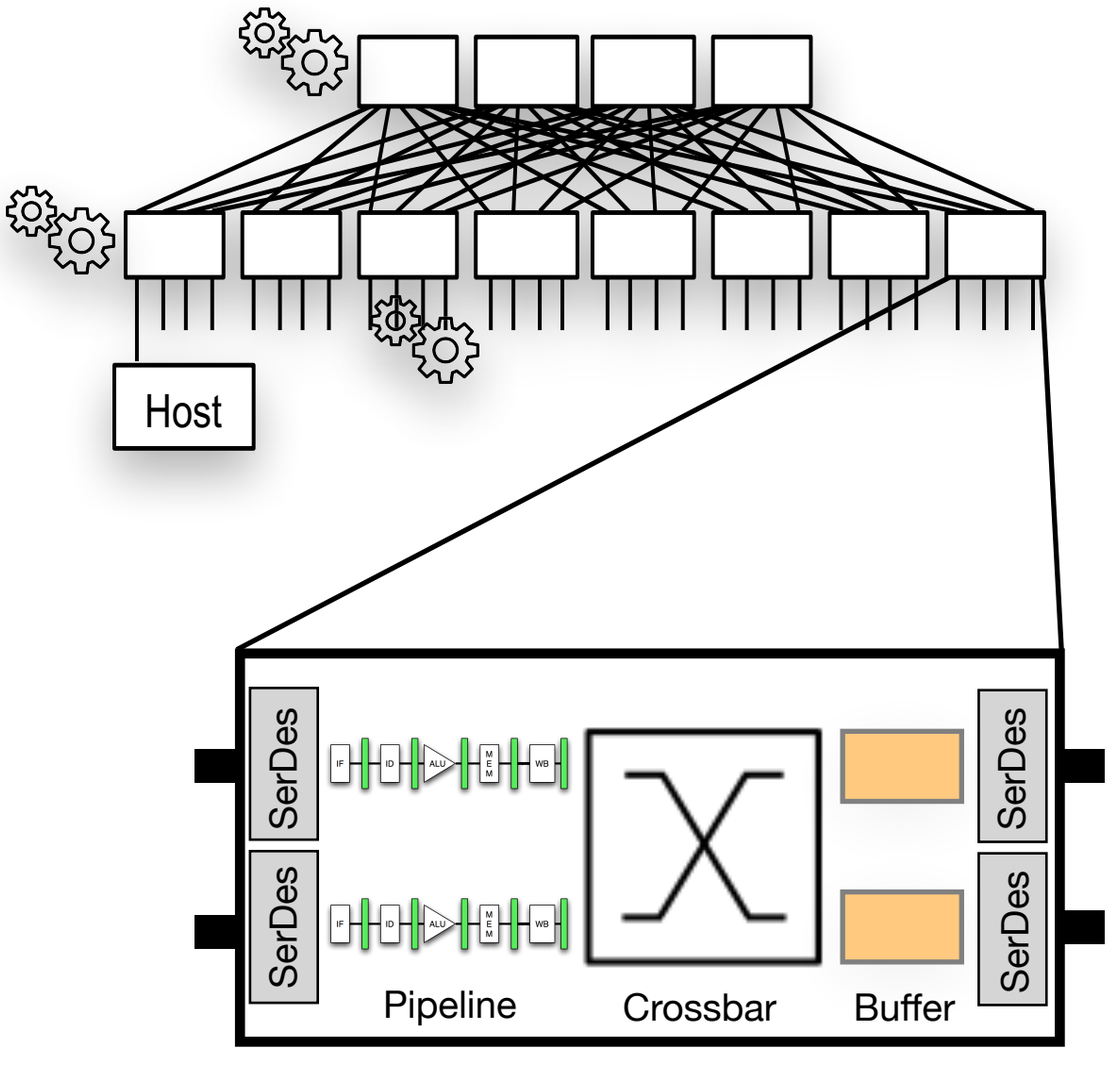
Promise of Circuit Switching

Packet Switching

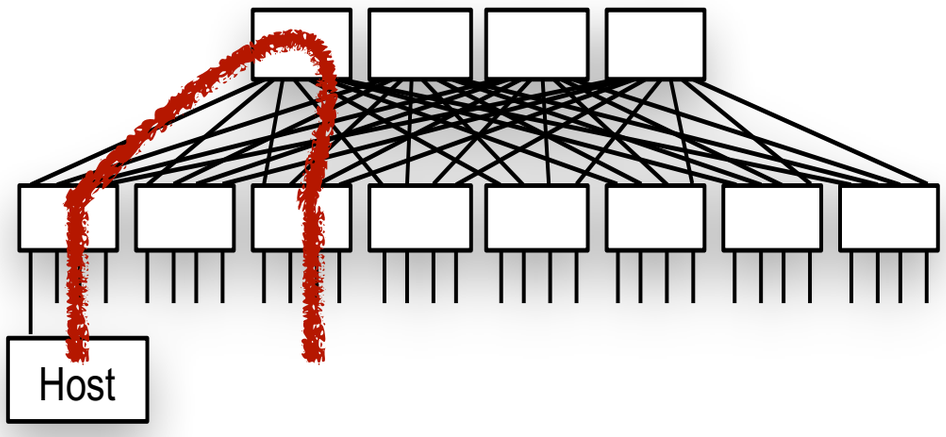
(Per packet, Per hop Decisions)

VS.

Circuit Switching



Packet Switch



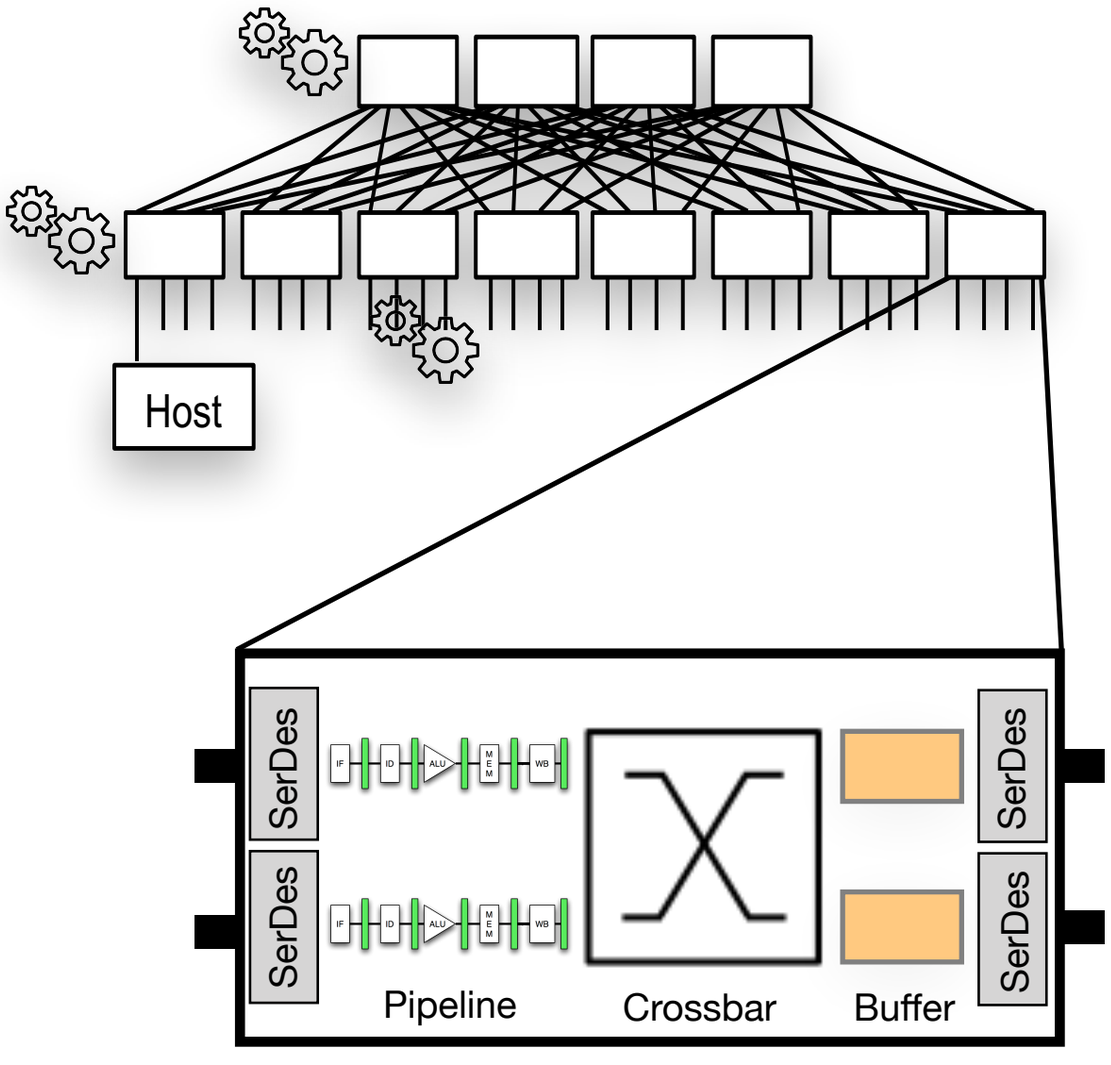
Promise of Circuit Switching

Packet Switching

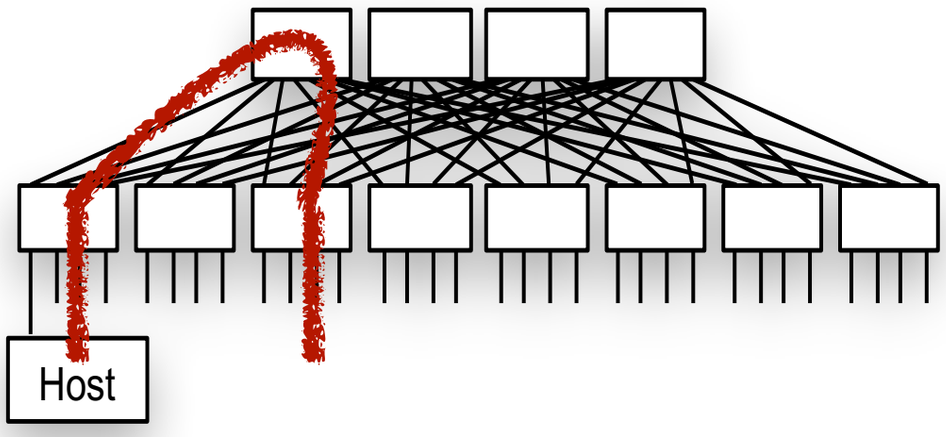
(Per packet, Per hop Decisions)

VS.

Circuit Switching



Packet Switch



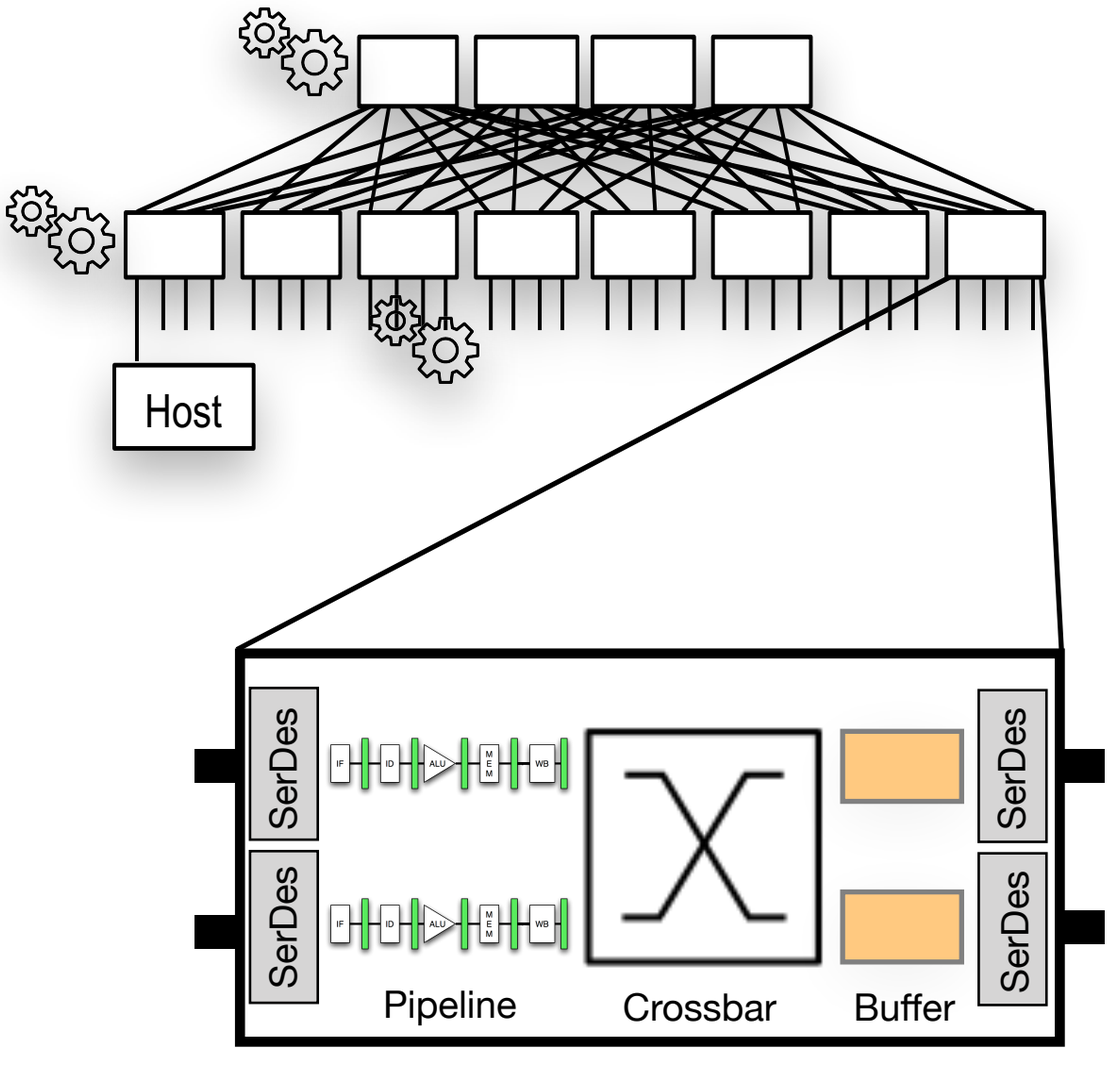
Promise of Circuit Switching

Packet Switching

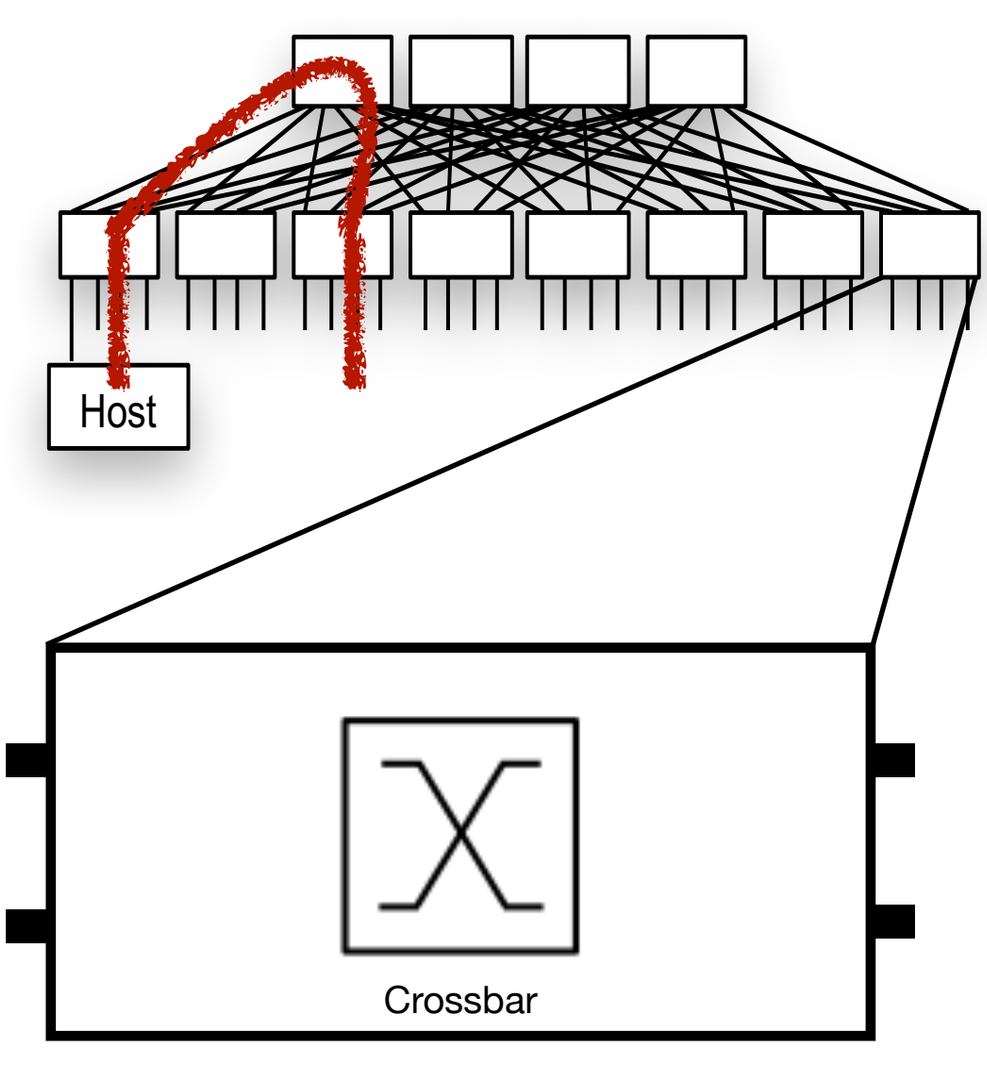
(Per packet, Per hop Decisions)

VS.

Circuit Switching



Packet Switch



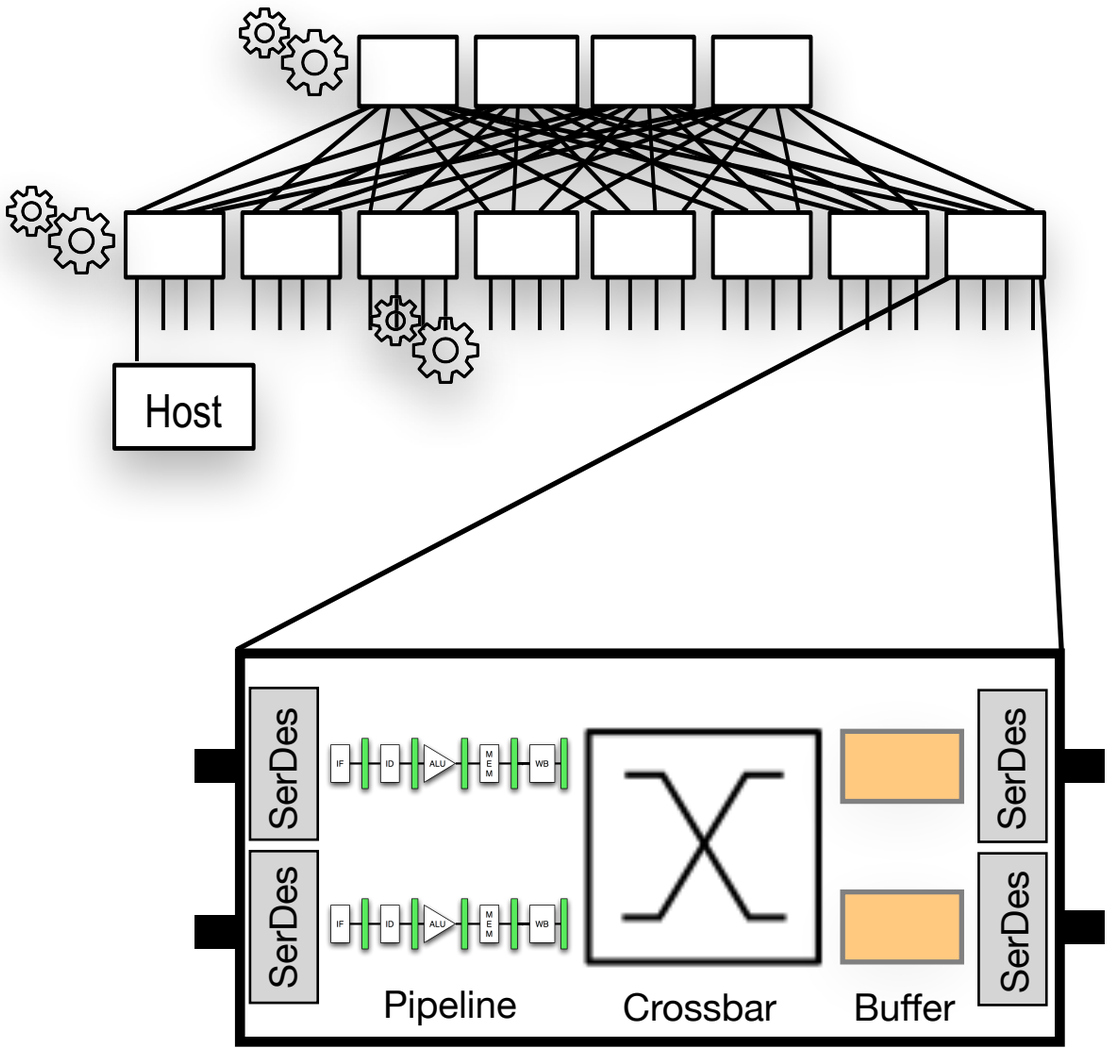
Circuit Switch

Promise of Circuit Switching

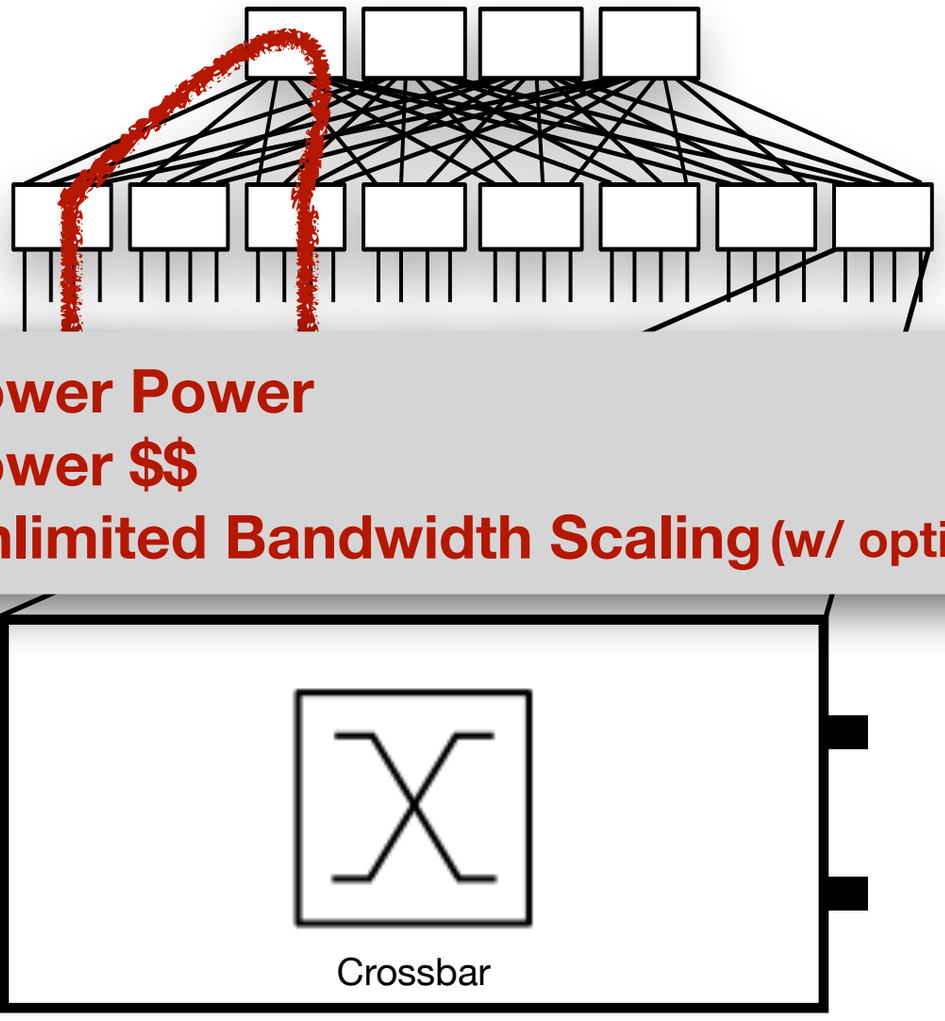
Packet Switching
(Per packet, Per hop Decisions)

VS.

Circuit Switching

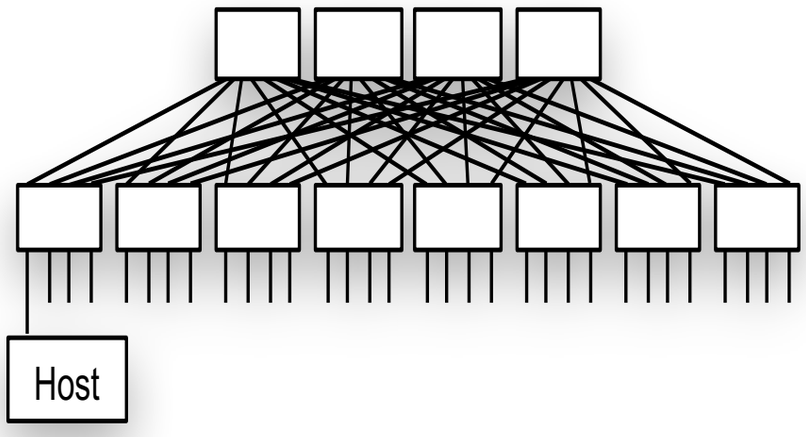


Packet Switch

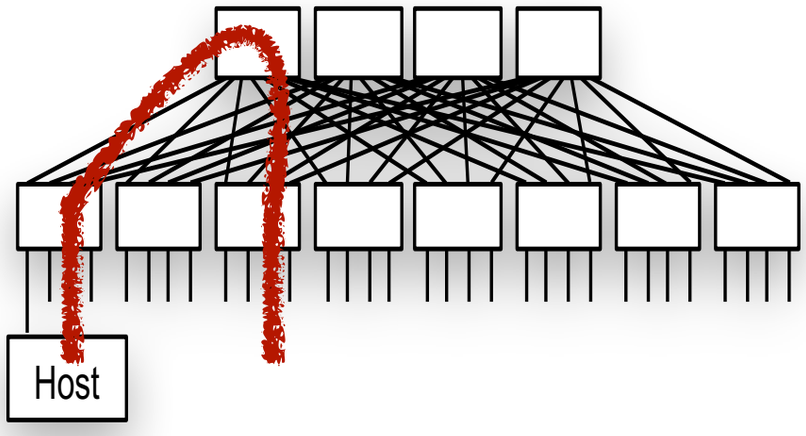


Circuit Switch

Challenge of Circuit Switching

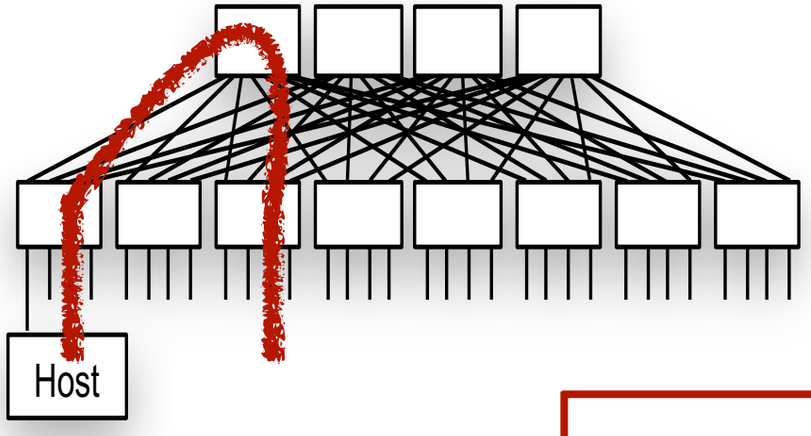


Challenge of Circuit Switching



- Circuit Configuration

Challenge of Circuit Switching



Circuit Configuration

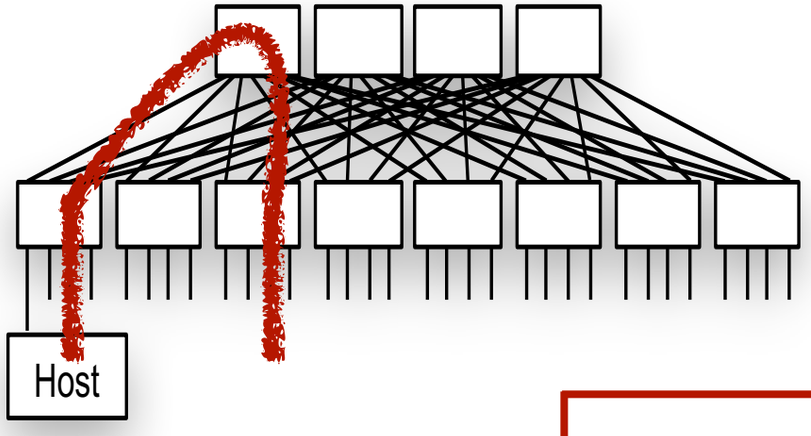
Data Plane

creating physical circuits

Control Plane

circuit scheduling

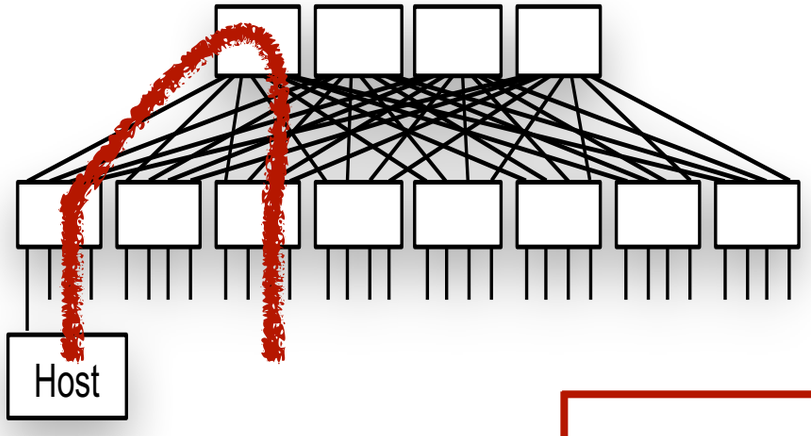
Challenge of Circuit Switching



Circuit Configuration

	Data Plane	Control Plane
	creating physical circuits	circuit scheduling
~2010	milliseconds	

Challenge of Circuit Switching



Circuit Configuration

Data Plane

creating physical circuits

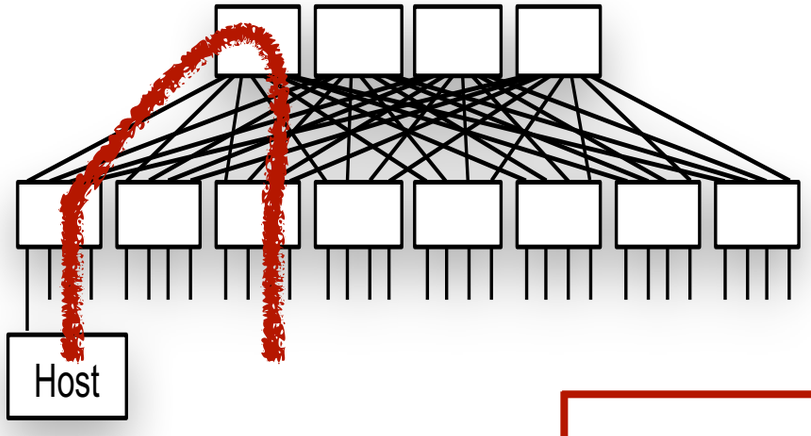
~2010 milliseconds

Today nanoseconds

Control Plane

circuit scheduling

Challenge of Circuit Switching



Circuit Configuration

Data Plane

creating physical circuits

~2010 milliseconds

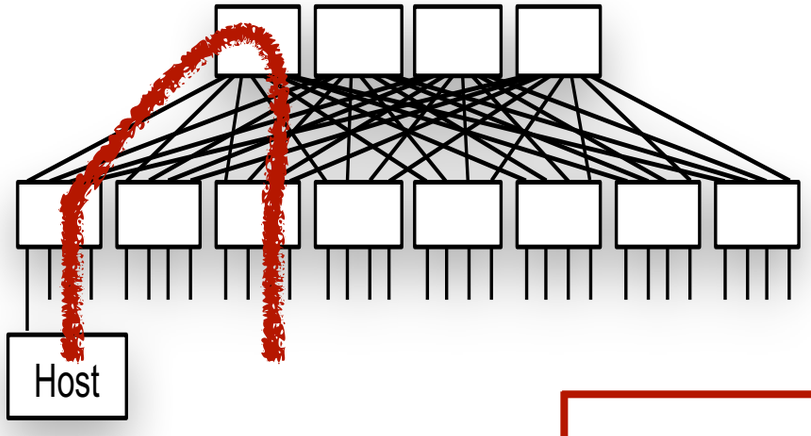
Today nanoseconds

Control Plane

circuit scheduling

Centralized

Challenge of Circuit Switching



Circuit Configuration

Data Plane

creating physical circuits

~2010 milliseconds

Today nanoseconds

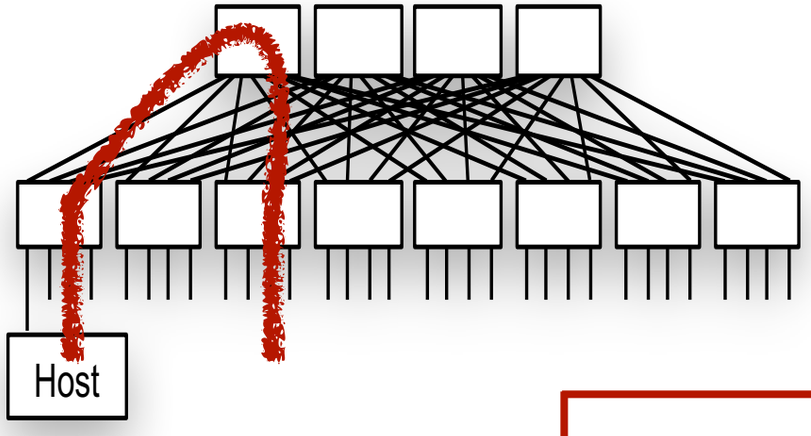
Control Plane

circuit scheduling

Traffic demand →  → Circuits

Centralized

Challenge of Circuit Switching



Circuit Configuration

Data Plane

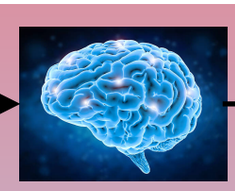
creating physical circuits

~2010 milliseconds

Today nanoseconds

Control Plane

circuit scheduling

Traffic demand →  → Circuits

Centralized

???

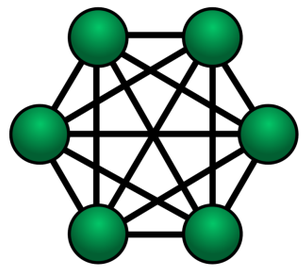
How to build a fast control plane for circuit switching?

nanosecond-scale circuit scheduling with high performance

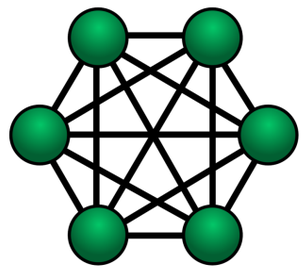
How to build a fast control plane for circuit switching?

nanosecond-scale circuit scheduling with high performance

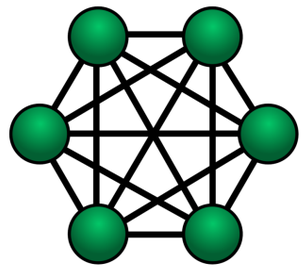
Shoal



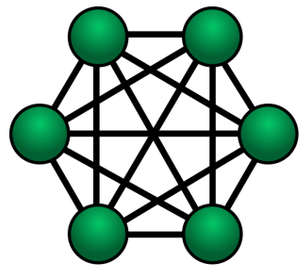
Shoal



Shoal



1. **Physical Layer :** Fast circuit scheduling mechanism
2. **Routing :** Bounded worst-case throughput
3. **Congestion Control :** Bounded worst-case queuing

Shoal 

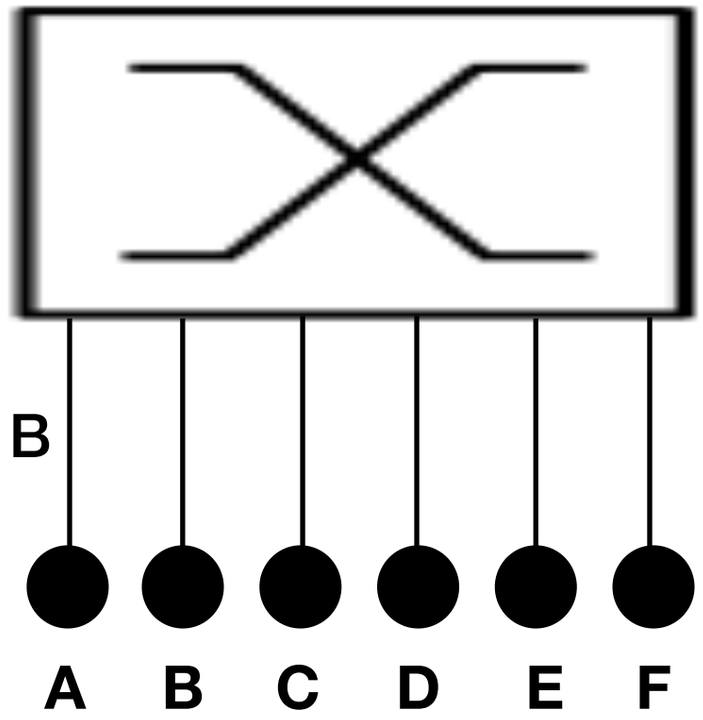
1. Physical Layer : Fast circuit scheduling mechanism

2. Routing : Bounded worst-case throughput

3. Congestion Control : Bounded worst-case queuing

Achieves comparable or better performance than several recent packet-switched network designs

Physical Layer



Physical Topology

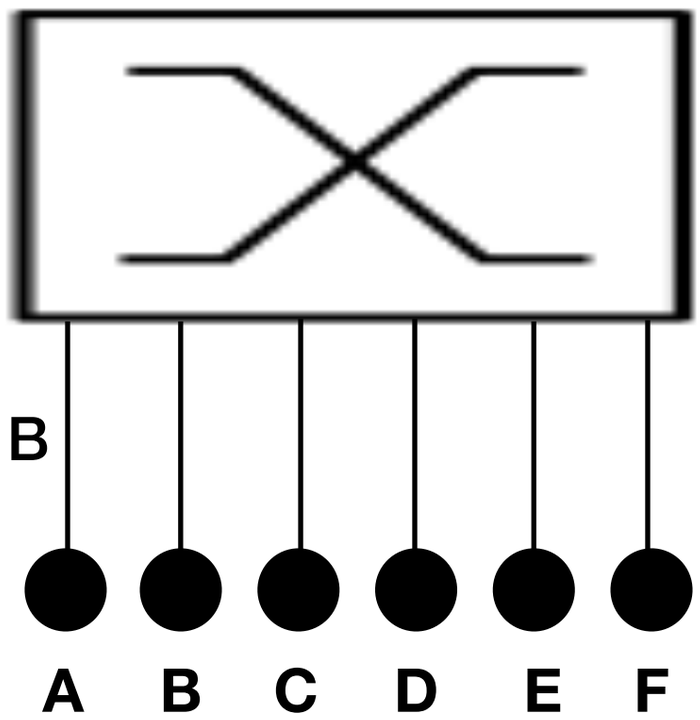
Physical Layer

End-hosts

	1	2	3	4	5
A					
B					
C					
D					
E					
F					

static, pre-defined schedule

+



Physical Topology

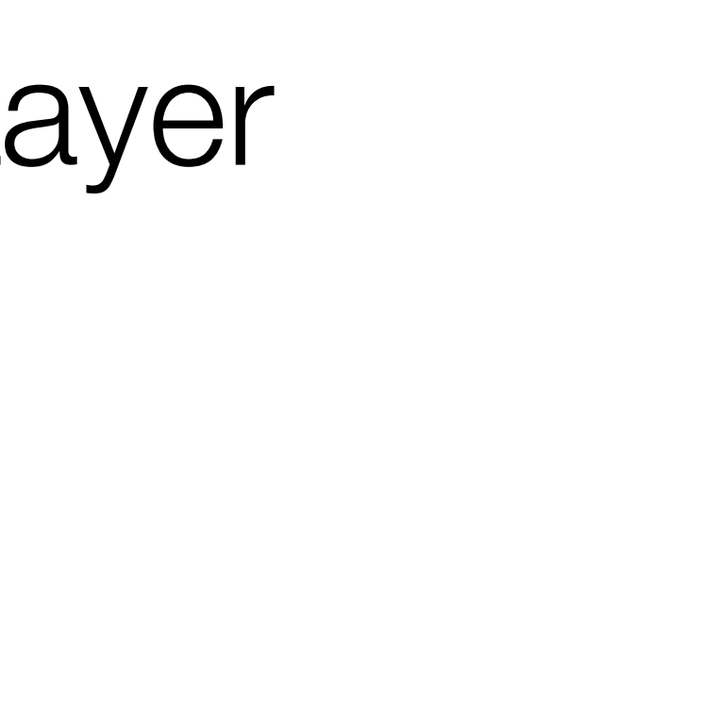
Physical Layer

N-1 timeslots (epoch)

End-hosts	1	2	3	4	5
A					
B					
C					
D					
E					
F					

static, pre-defined schedule

+



Physical Topology

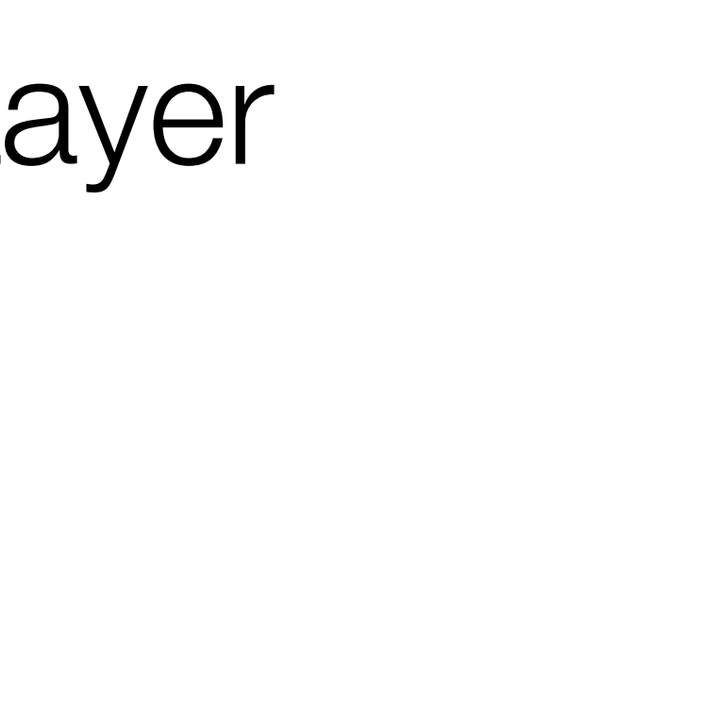
Physical Layer

N-1 timeslots (epoch)

End-hosts	1	2	3	4	5
A	B				
B	C				
C	D				
D	E				
E	F				
F	A				

static, pre-defined schedule

+



Physical Topology

Physical Layer

N-1 timeslots (epoch)

End-hosts	1	2	3	4	5
A	B				
B	C				
C	D				
D	E				
E	F				
F	A				

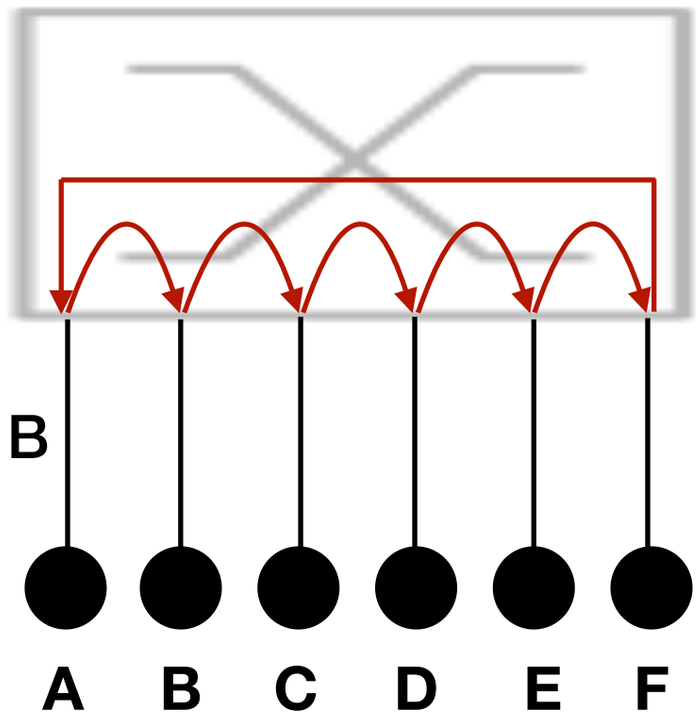
static, pre-defined schedule

+

Synchronous System

ns-precision time sync

DTP [SIGCOMM'16]



Physical Topology

Physical Layer

N-1 timeslots (epoch)

End-hosts	1	2	3	4	5
A	B	C	D	E	F
B	C	D	E	F	A
C	D	E	F	A	B
D	E	F	A	B	C
E	F	A	B	C	D
F	A	B	C	D	E

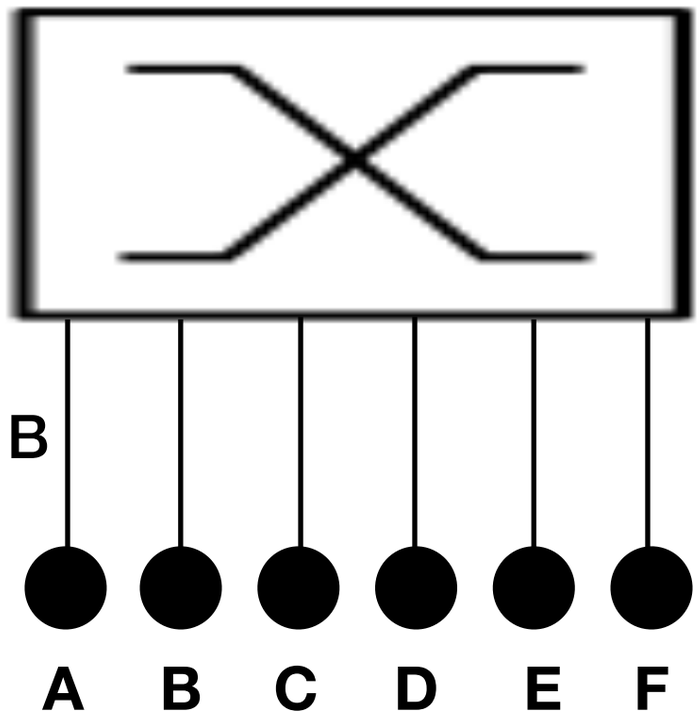
static, pre-defined schedule

+

Synchronous System

ns-precision time sync

DTP [SIGCOMM'16]



Physical Topology

Physical Layer

N-1 timeslots (epoch)

End-hosts	1	2	3	4	5
A	B	C	D	E	F
B	C	D	E	F	A
C	D	E	F	A	B
D	E	F	A	B	C
E	F	A	B	C	D
F	A	B	C	D	E

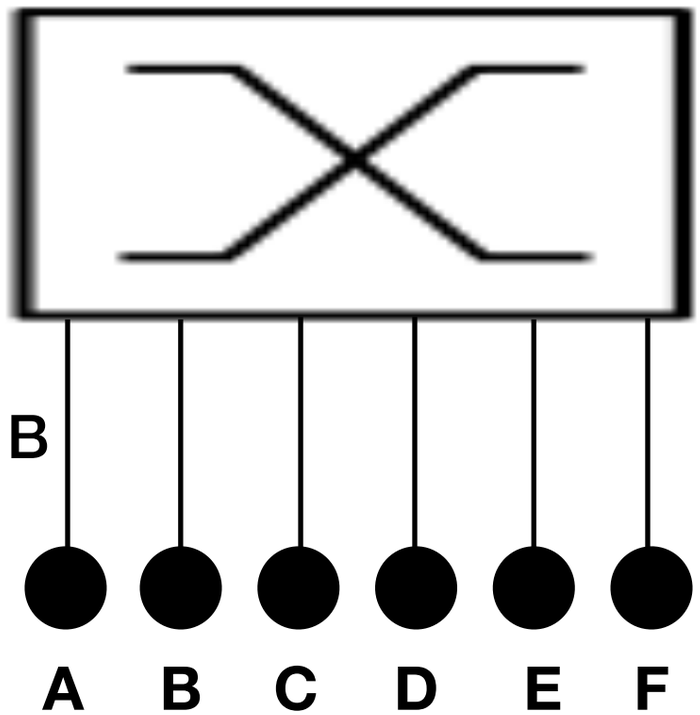
static, pre-defined schedule

+

Synchronous System

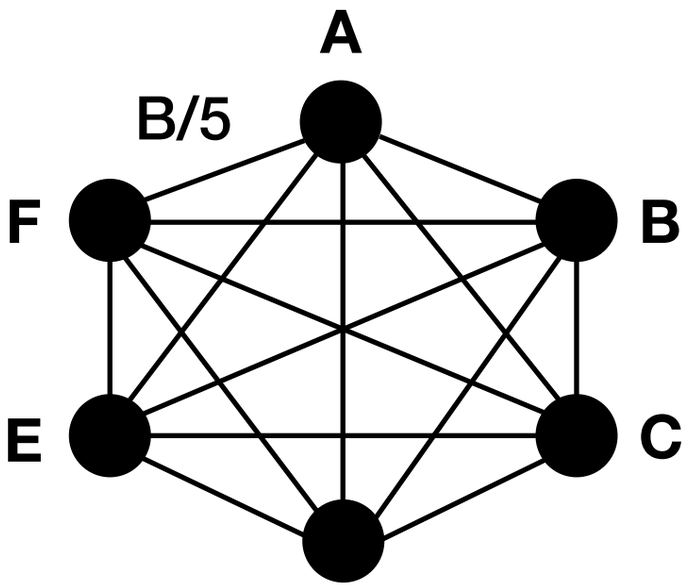
ns-precision time sync

DTP [SIGCOMM'16]



Physical Topology

Physical Layer



Full Mesh
Virtual Topology

N-1 timeslots (epoch)

End-hosts	1	2	3	4	5
A	B	C	D	E	F
B	C	D	E	F	A
C	D	E	F	A	B
D	E	F	A	B	C
E	F	A	B	C	D
F	A	B	C	D	E

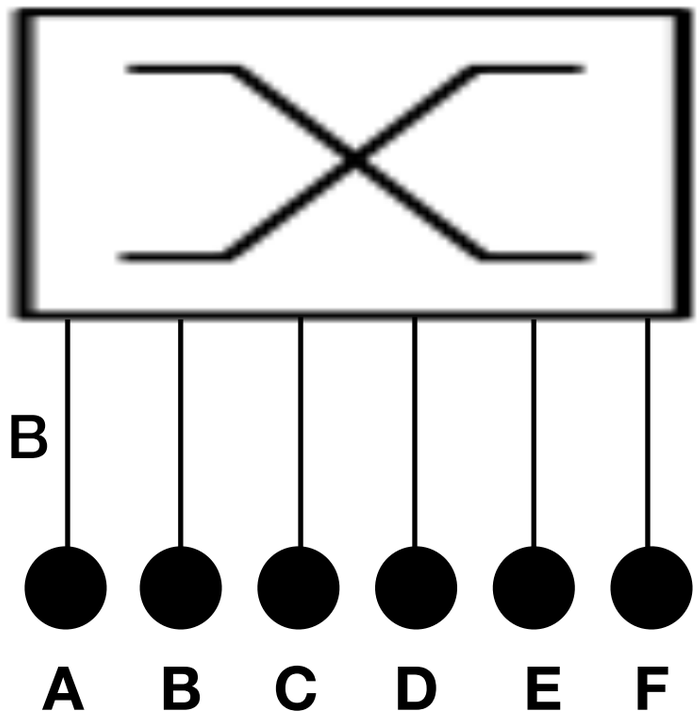
static, pre-defined schedule

+

Synchronous System

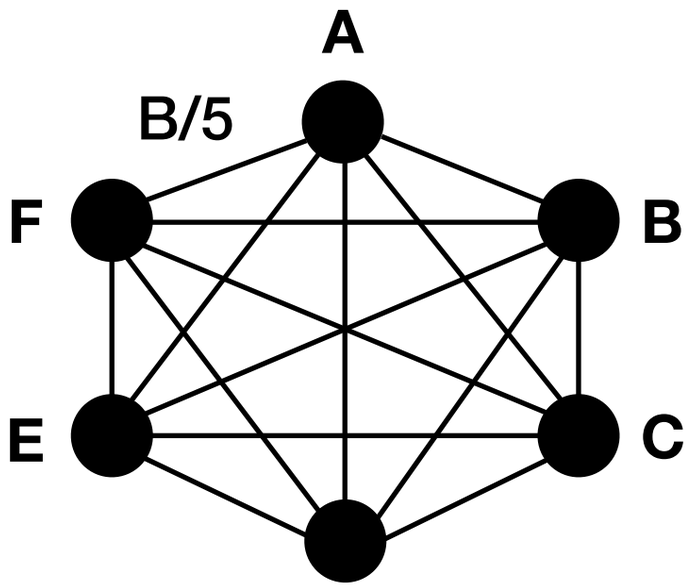
ns-precision time sync

DTP [SIGCOMM'16]



Physical Topology

Physical Layer



Full Mesh
Virtual Topology

N-1 timeslots (epoch)

End-hosts	1	2	3	4	5
A	B	C	D	E	F
B	C	D	E	F	A
C	D	E	F	A	B
D	E	F	A	B	C
E	F	A	B	C	D
F	A	B	C	D	E

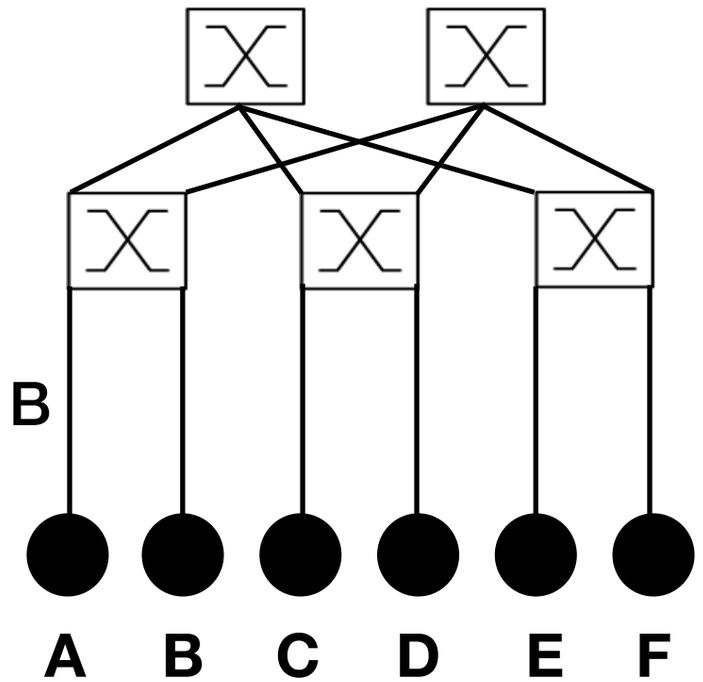
static, pre-defined schedule

+

Synchronous System

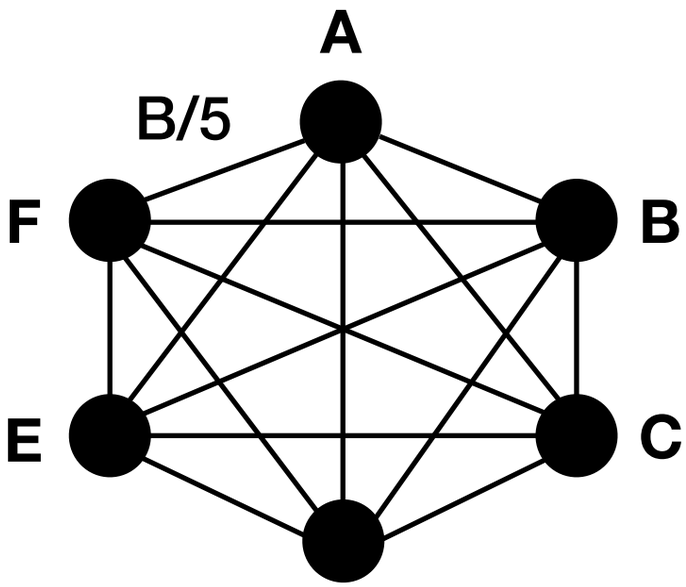
ns-precision time sync

DTP [SIGCOMM'16]



Physical Topology

Physical Layer



Full Mesh
Virtual Topology

N-1 timeslots (epoch)

End-hosts	1	2	3	4	5
A	B	C	D	E	F
B	C	D	E	F	A
C	D	E	F	A	B
D	E	F	A	B	C
E	F	A	B	C	D
F	A	B	C	D	E

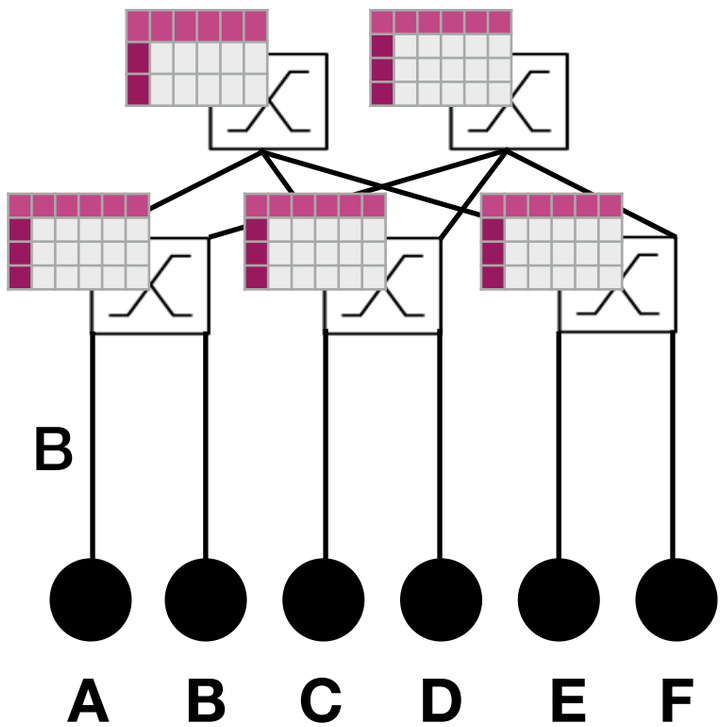
static, pre-defined schedule

+

Synchronous System

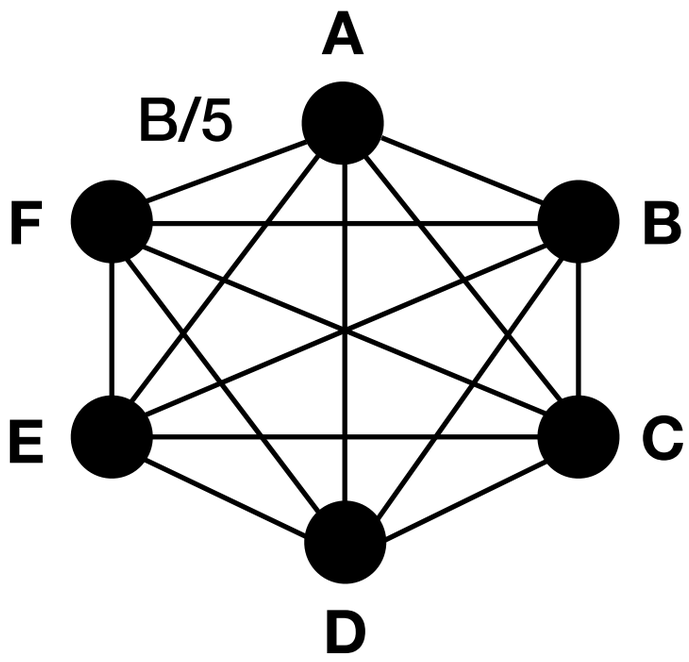
ns-precision time sync

DTP [SIGCOMM'16]

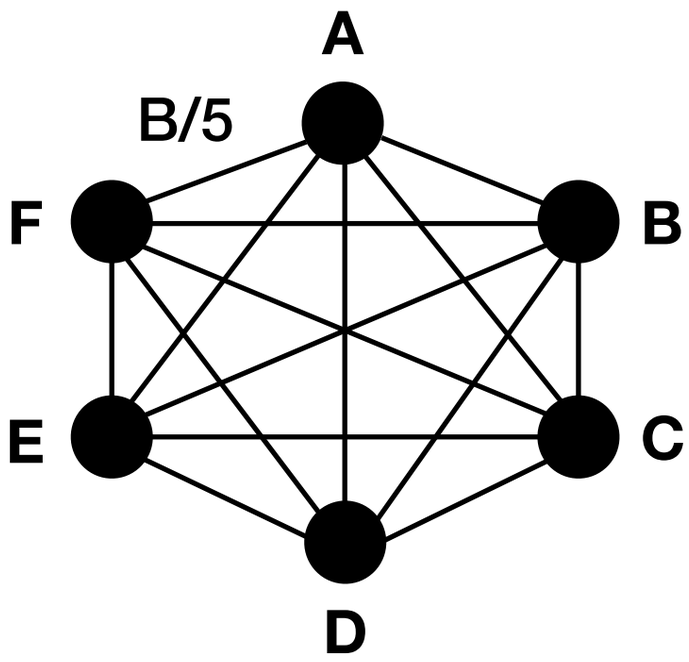


Physical Topology

Routing & Congestion Control

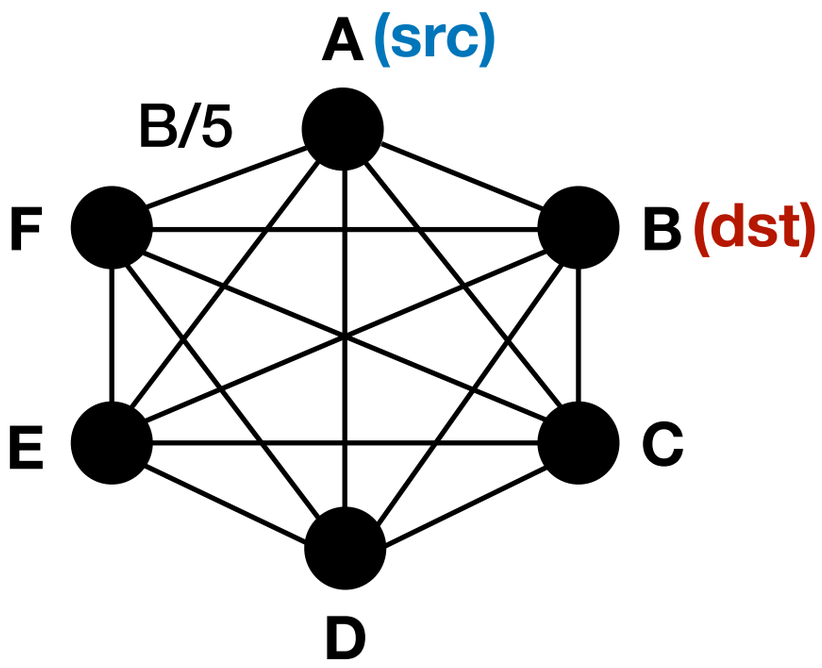


Routing & Congestion Control



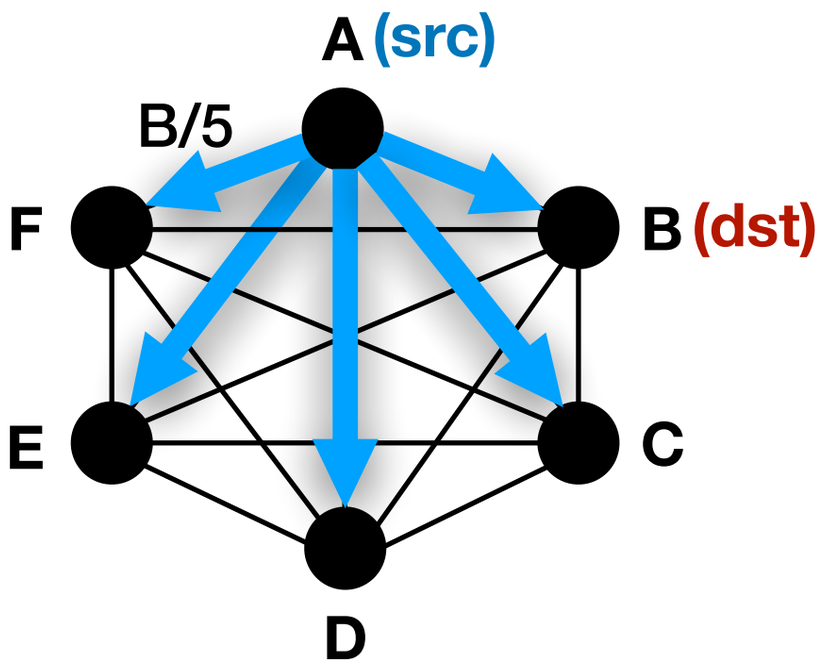
Valiant Load Balancing

Routing & Congestion Control



Valiant
Load
Balancing

Routing & Congestion Control

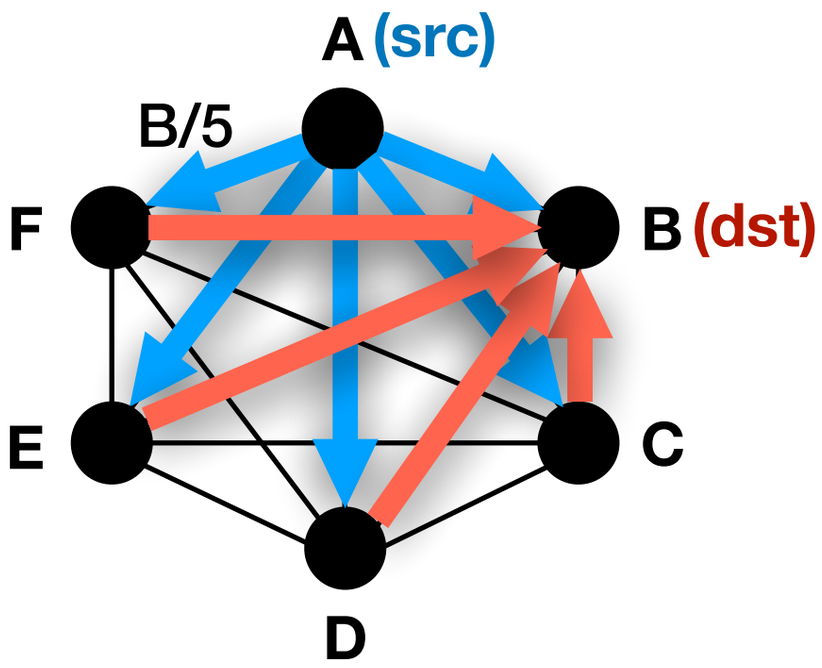


Valiant Load Balancing

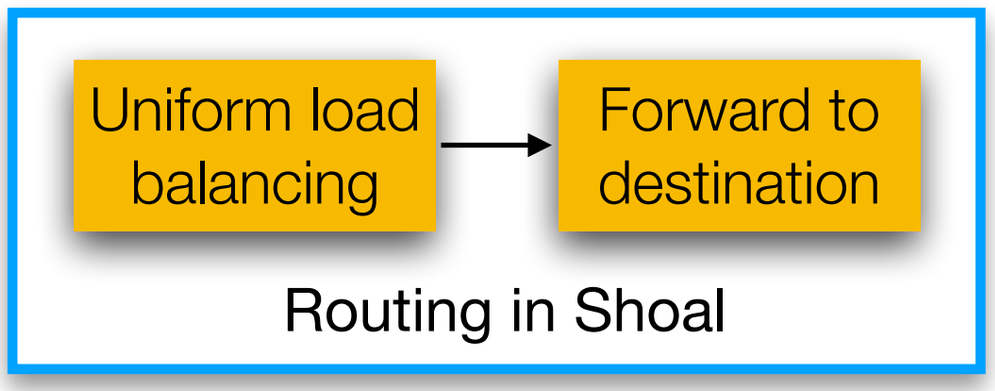
Uniform load balancing

Routing in Shoal

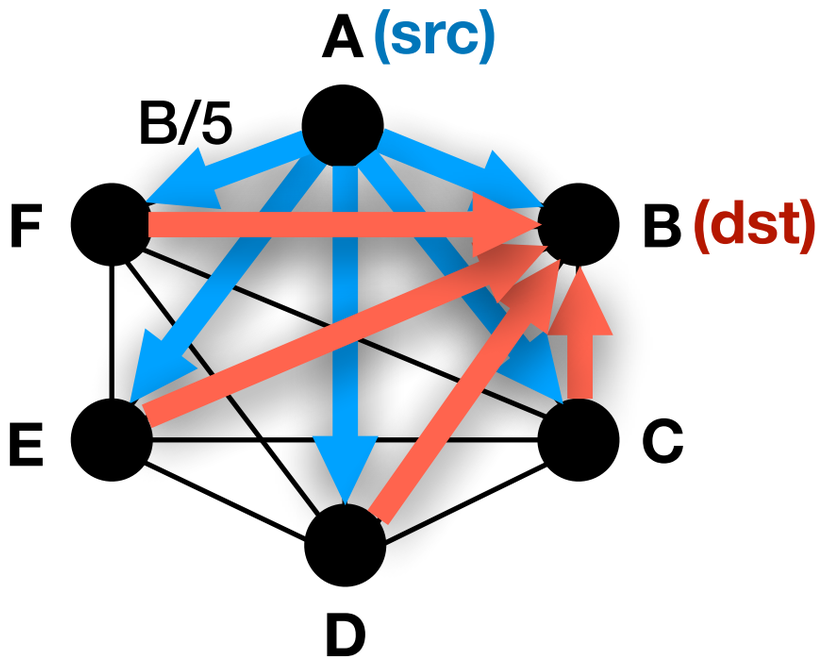
Routing & Congestion Control



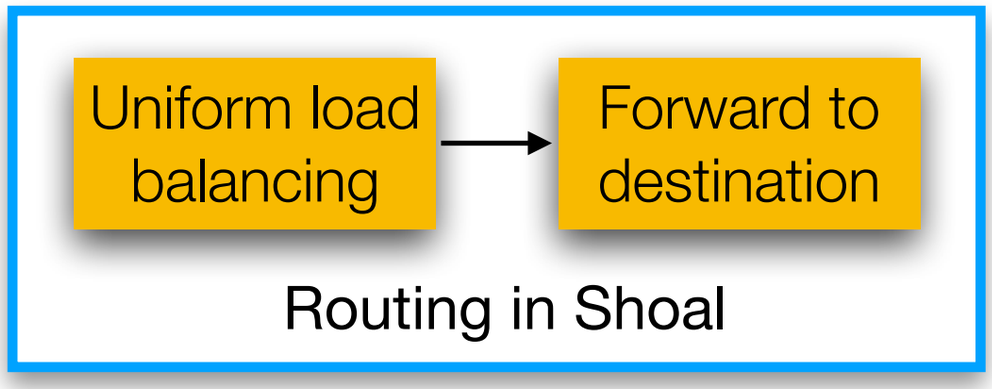
Valiant Load Balancing



Routing & Congestion Control

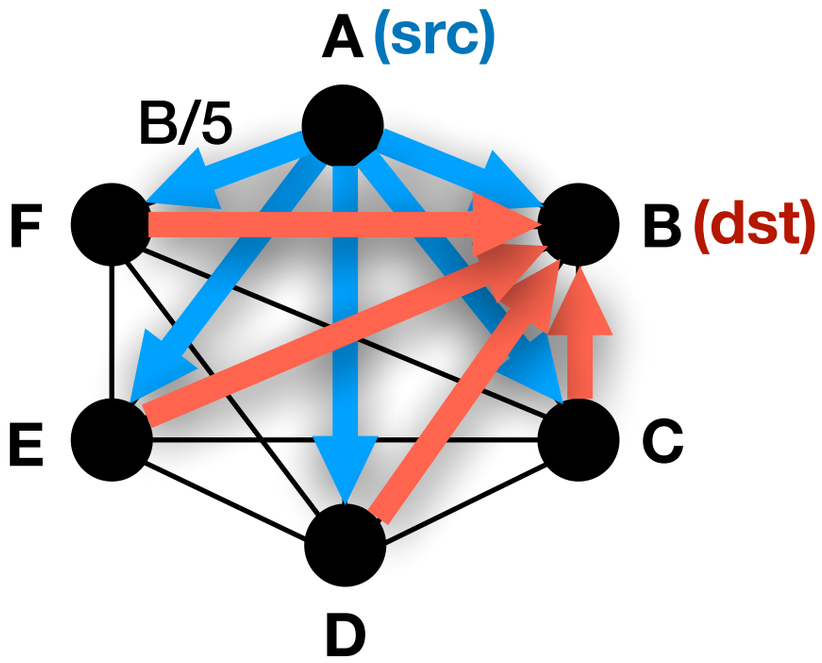


Valiant Load Balancing

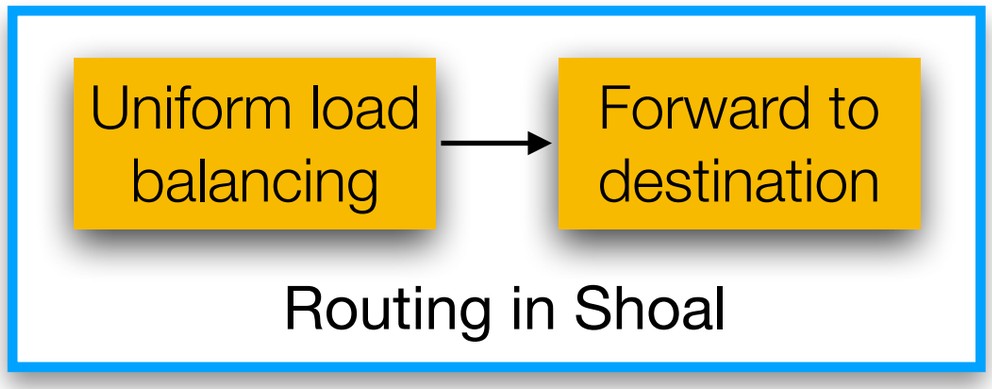


Bounded worst-case throughput of 50%
compared to an ideal packet-switched network

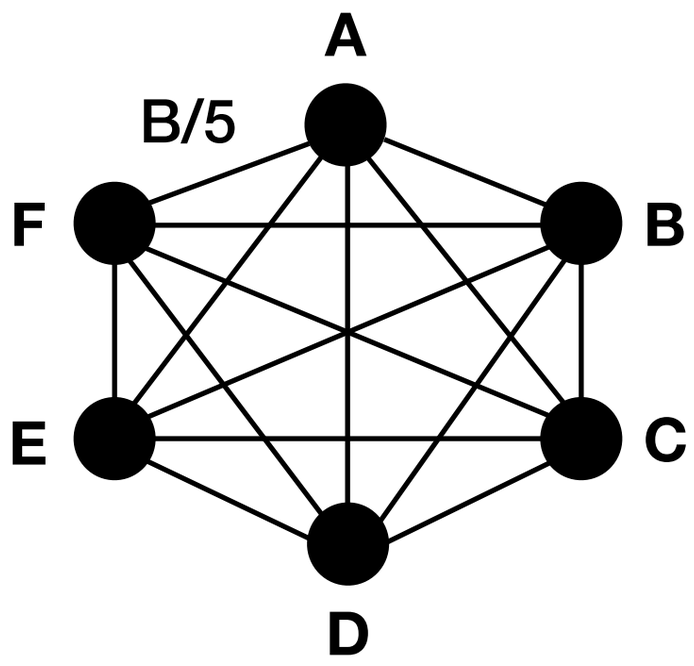
Routing & Congestion Control



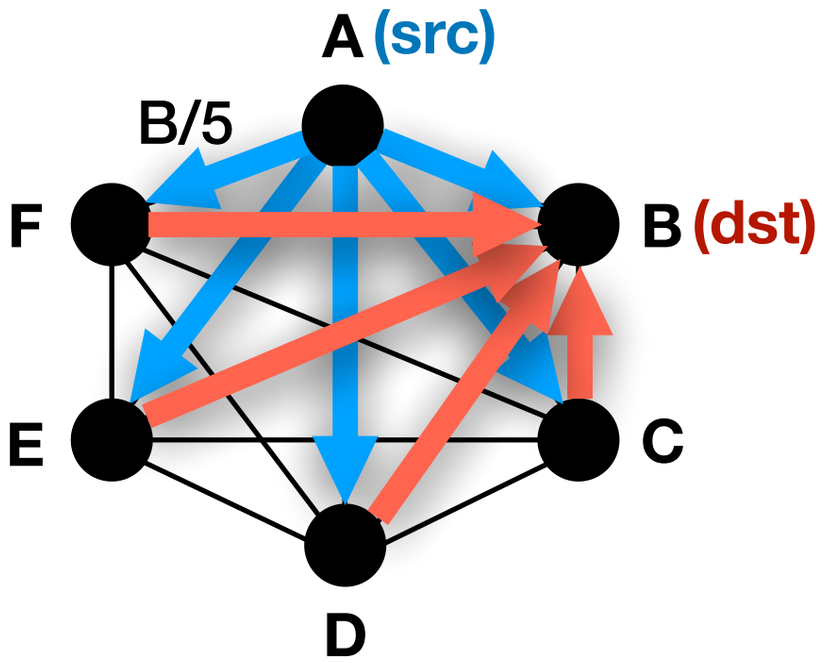
Valiant Load Balancing



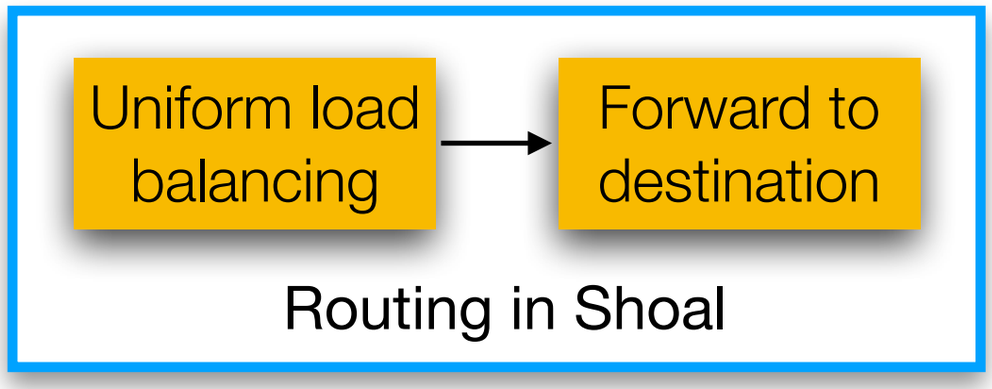
Bounded worst-case throughput of 50% compared to an ideal packet-switched network



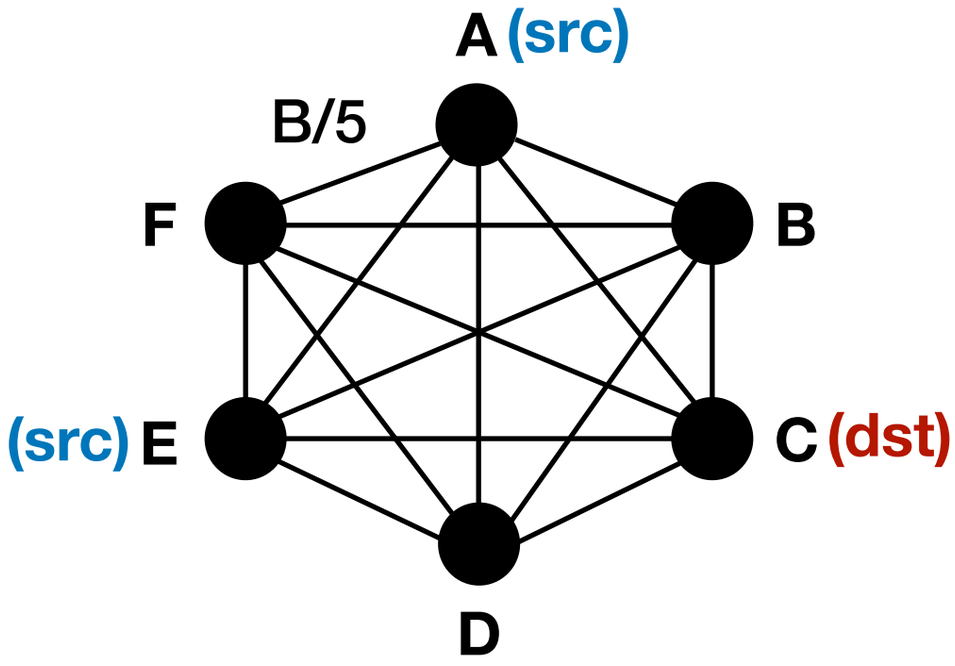
Routing & Congestion Control



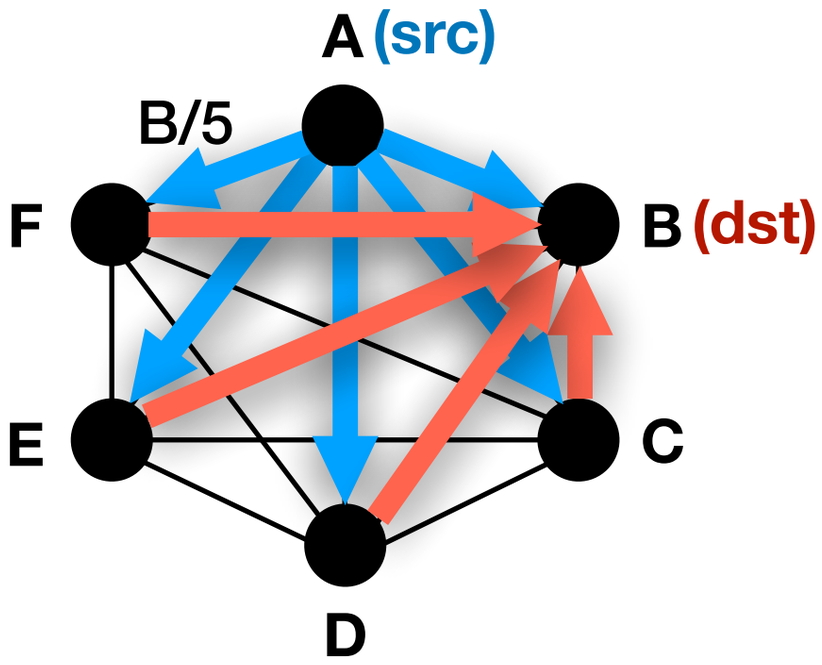
Valiant Load Balancing



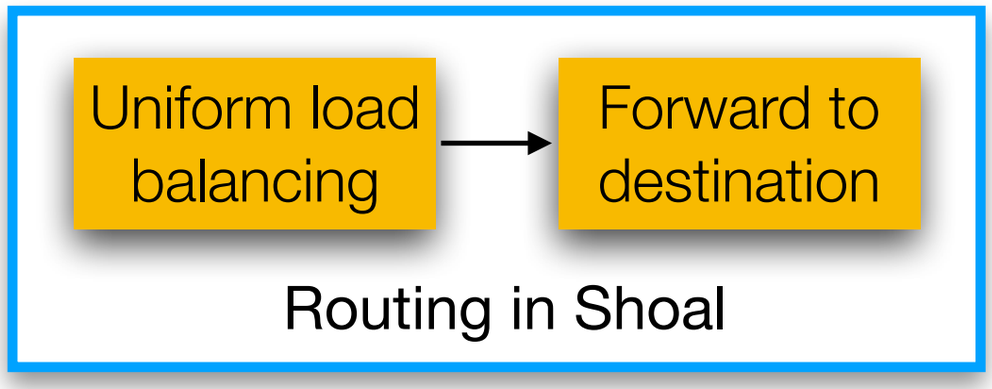
Bounded worst-case throughput of 50% compared to an ideal packet-switched network



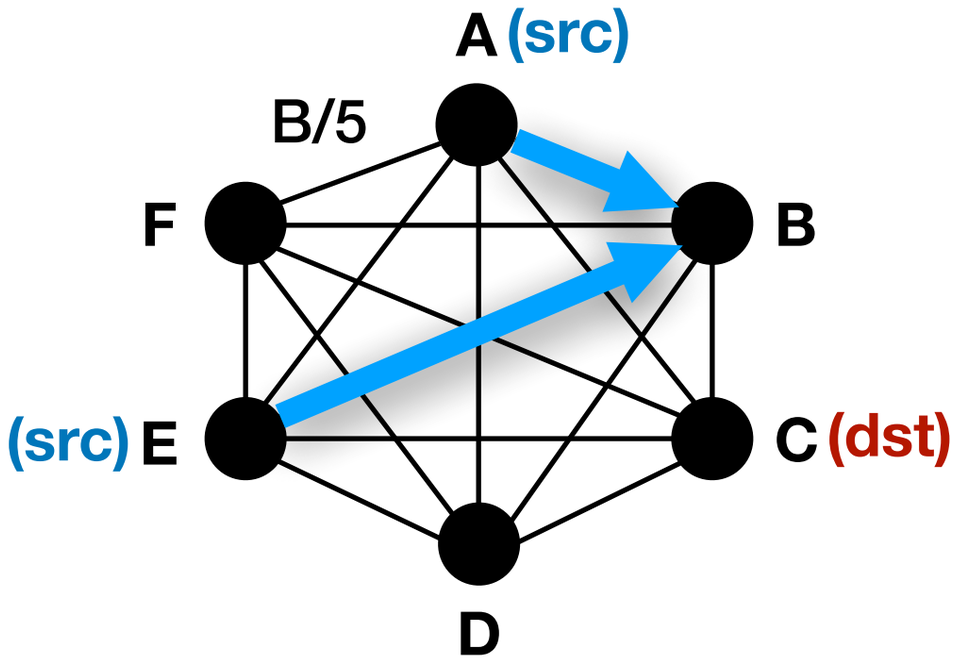
Routing & Congestion Control



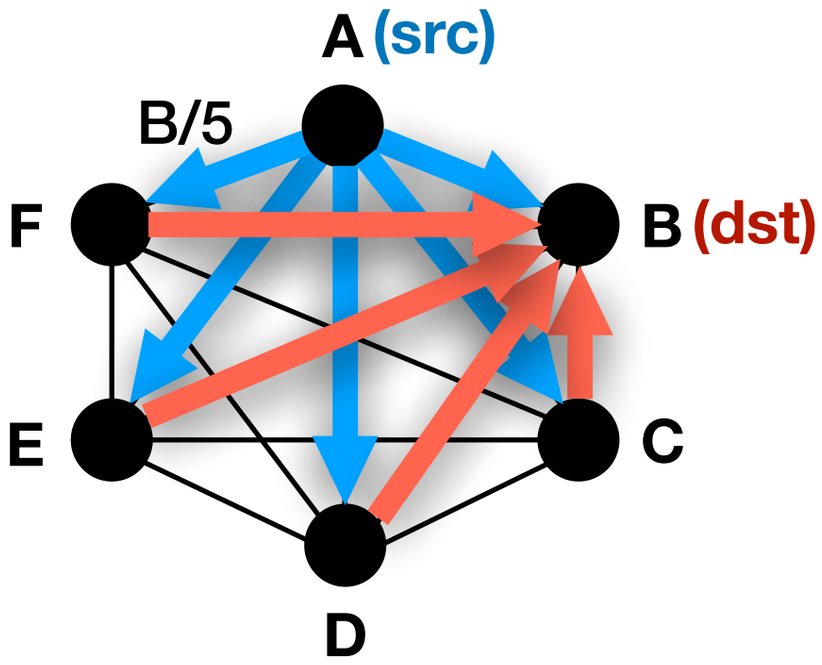
Valiant Load Balancing



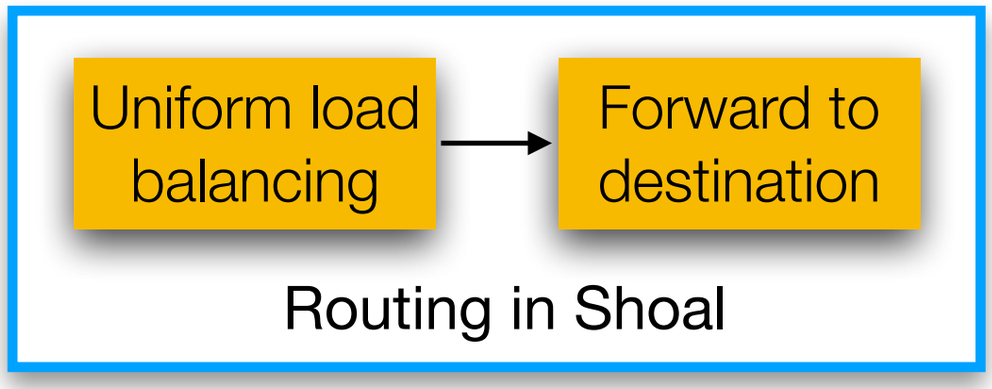
Bounded worst-case throughput of 50% compared to an ideal packet-switched network



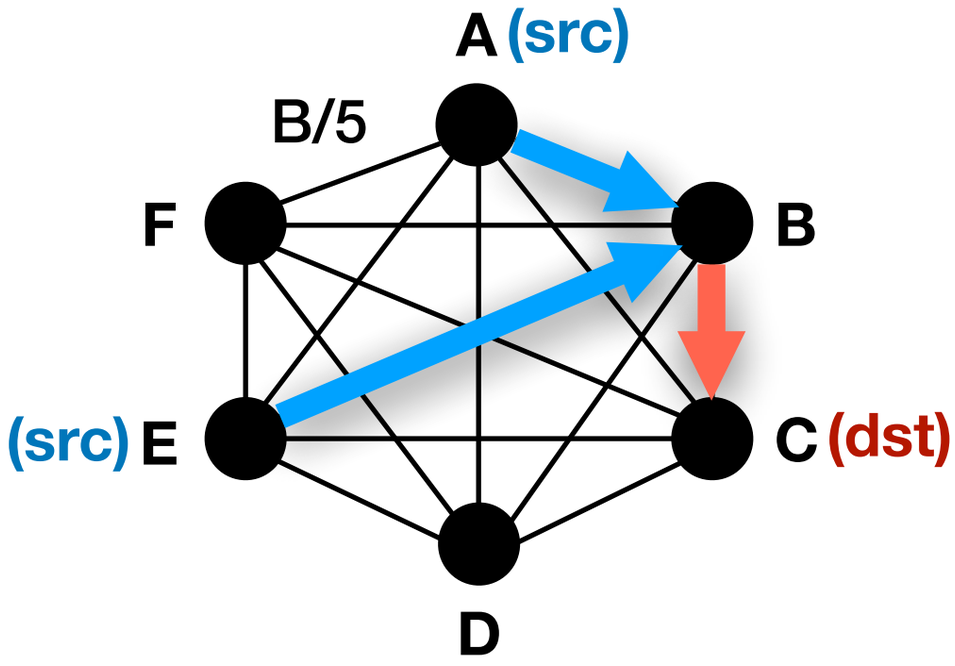
Routing & Congestion Control



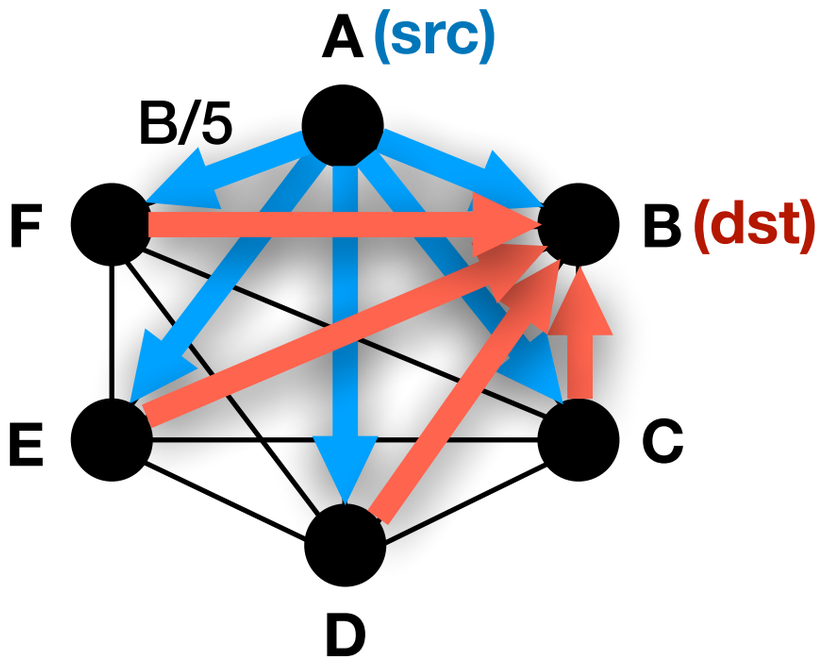
Valiant Load Balancing



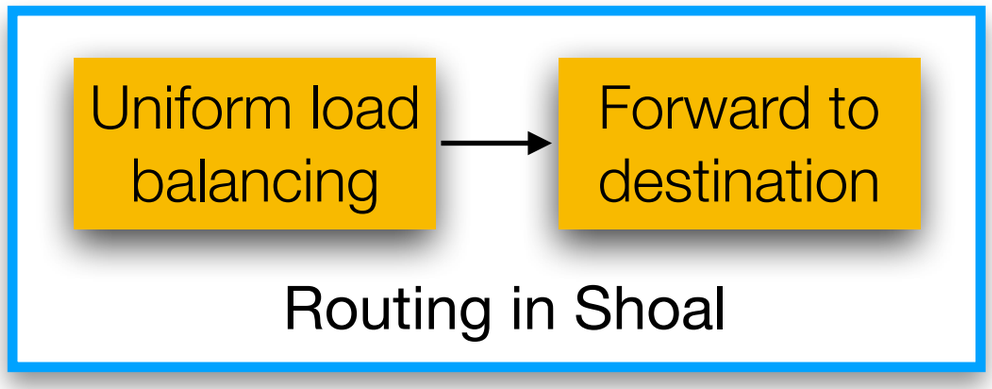
Bounded worst-case throughput of 50% compared to an ideal packet-switched network



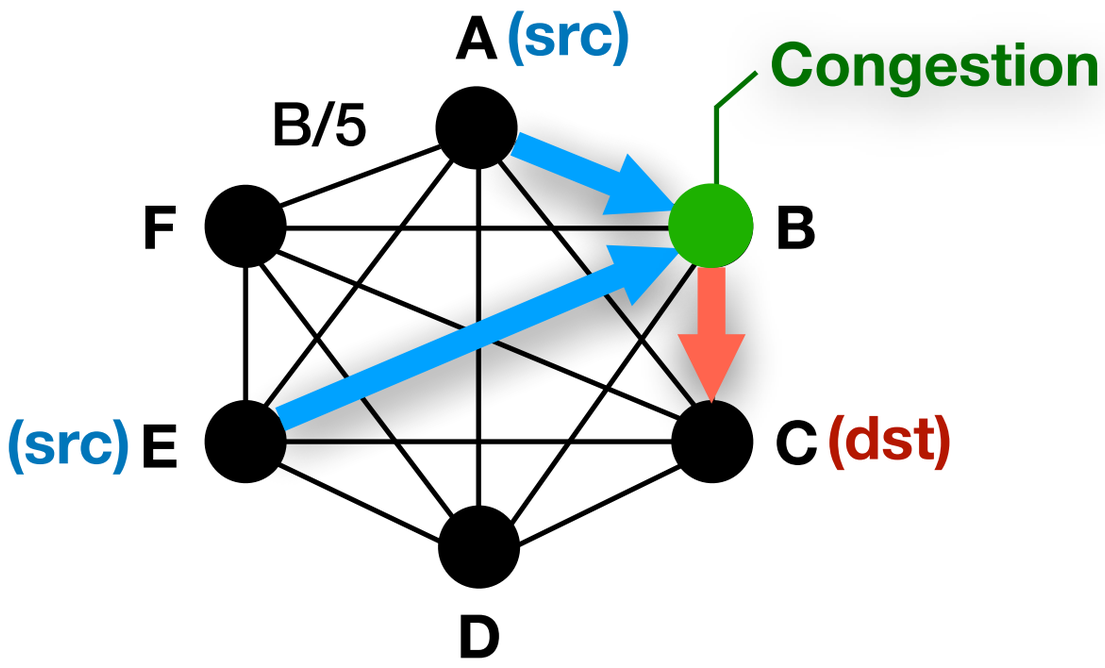
Routing & Congestion Control



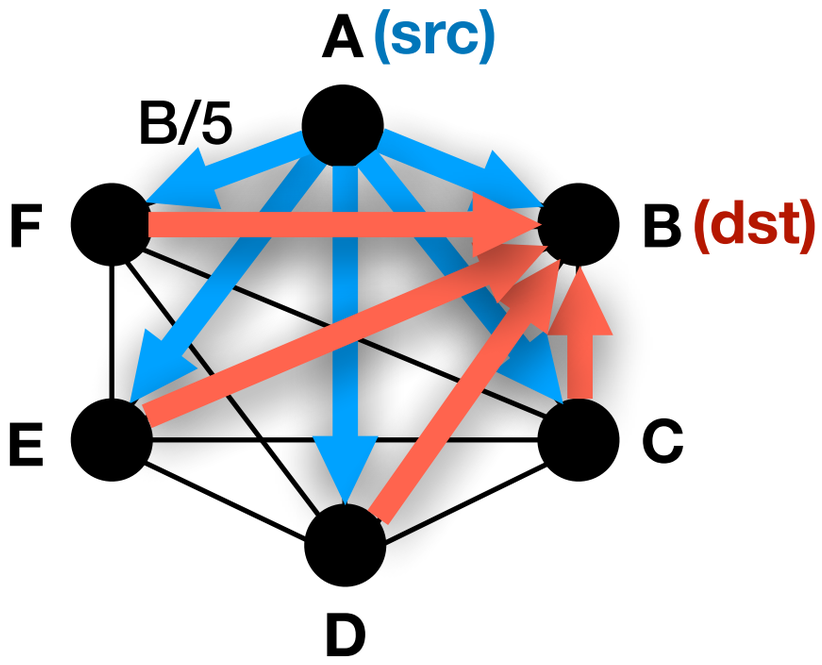
Valiant Load Balancing



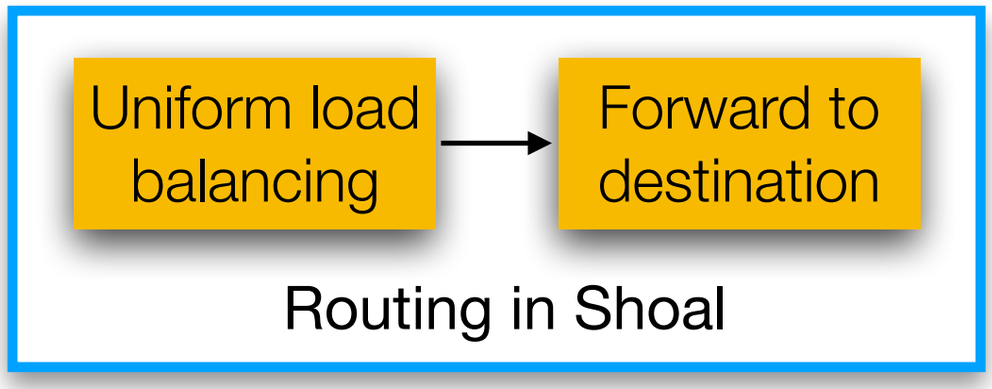
Bounded worst-case throughput of 50% compared to an ideal packet-switched network



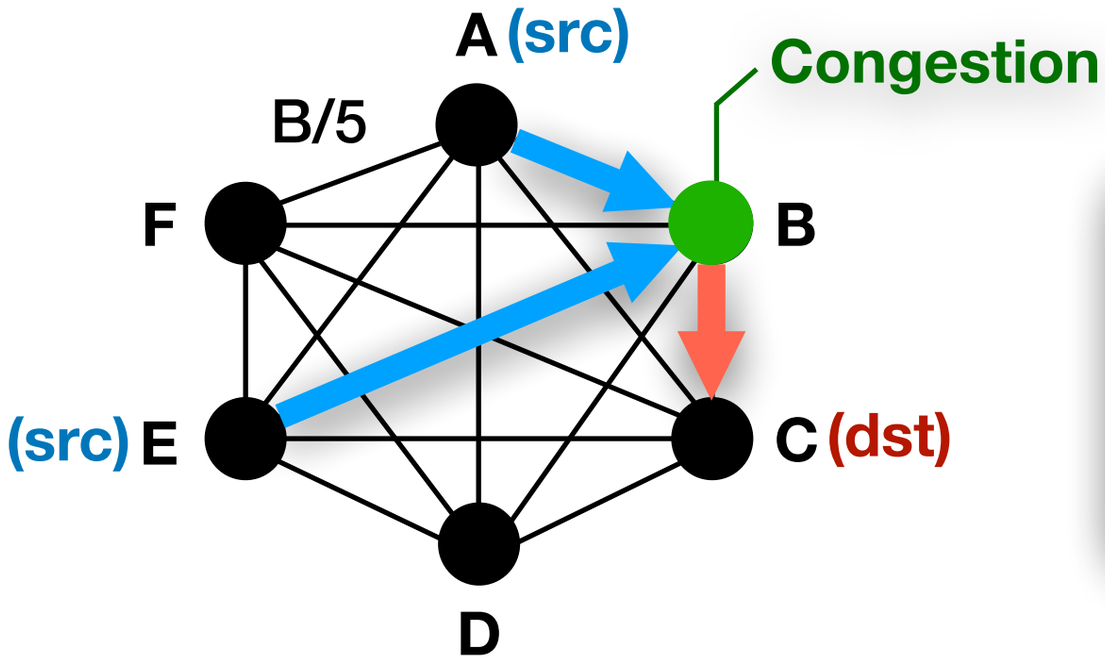
Routing & Congestion Control



Valiant Load Balancing



Bounded worst-case throughput of 50%
compared to an ideal packet-switched network



Congestion Control

Each per-destination queue Q_i for each destination i is bounded!

$len(Q_i) \leq 1 + incast\_degree(i)$ **packets**

Key Properties of Shoal

- Fast, de-centralized, and traffic-agnostic **circuit scheduling**
- **Routing** with bounded worst-case throughput
- **Congestion Control** with bounded worst-case queuing

Key Properties of Shoal

- Fast, de-centralized, and traffic-agnostic **circuit scheduling**
- **Routing** with bounded worst-case throughput
- **Congestion Control** with bounded worst-case queuing
- **Worst-case network throughput of 50%**
  (compared to an ideal packet-switched network)

Key Properties of Shoal

- Fast, de-centralized, and traffic-agnostic **circuit scheduling**
- **Routing** with bounded worst-case throughput
- **Congestion Control** with bounded worst-case queuing
- **Worst-case network throughput of 50%**
  (compared to an ideal packet-switched network)
  - Equip each node with 2x bandwidth

Key Properties of Shoal

- Fast, de-centralized, and traffic-agnostic **circuit scheduling**
- **Routing** with bounded worst-case throughput
- **Congestion Control** with bounded worst-case queuing
- **Worst-case network throughput of 50%**
  (compared to an ideal packet-switched network)
  - Equip each node with 2x bandwidth
  - $power$ (Shoal) < $power$ (packet-switched network with 1/2 bandwidth of Shoal)
  - $cost$ (Shoal) < $cost$ (packet-switched network with 1/2 bandwidth of Shoal)

Implementation

Implementation

Stratix V FPGA

- Circuit Switch
  - re-configures in < 6.4ns
- End-host NIC
  - fully pipelined design

FPGA code for the implementation of Shoal is available at:
https://github.com/vishal1303/Shoal

Implementation

- Routing
- Congestion Control
- Scheduling
- Synchronization

Stratix V FPGA

- Circuit Switch
  - re-configures in < 6.4ns
- End-host NIC
  - fully pipelined design

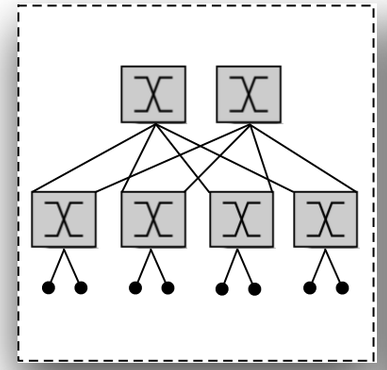
FPGA code for the implementation of Shoal is available at:
https://github.com/vishal1303/Shoal

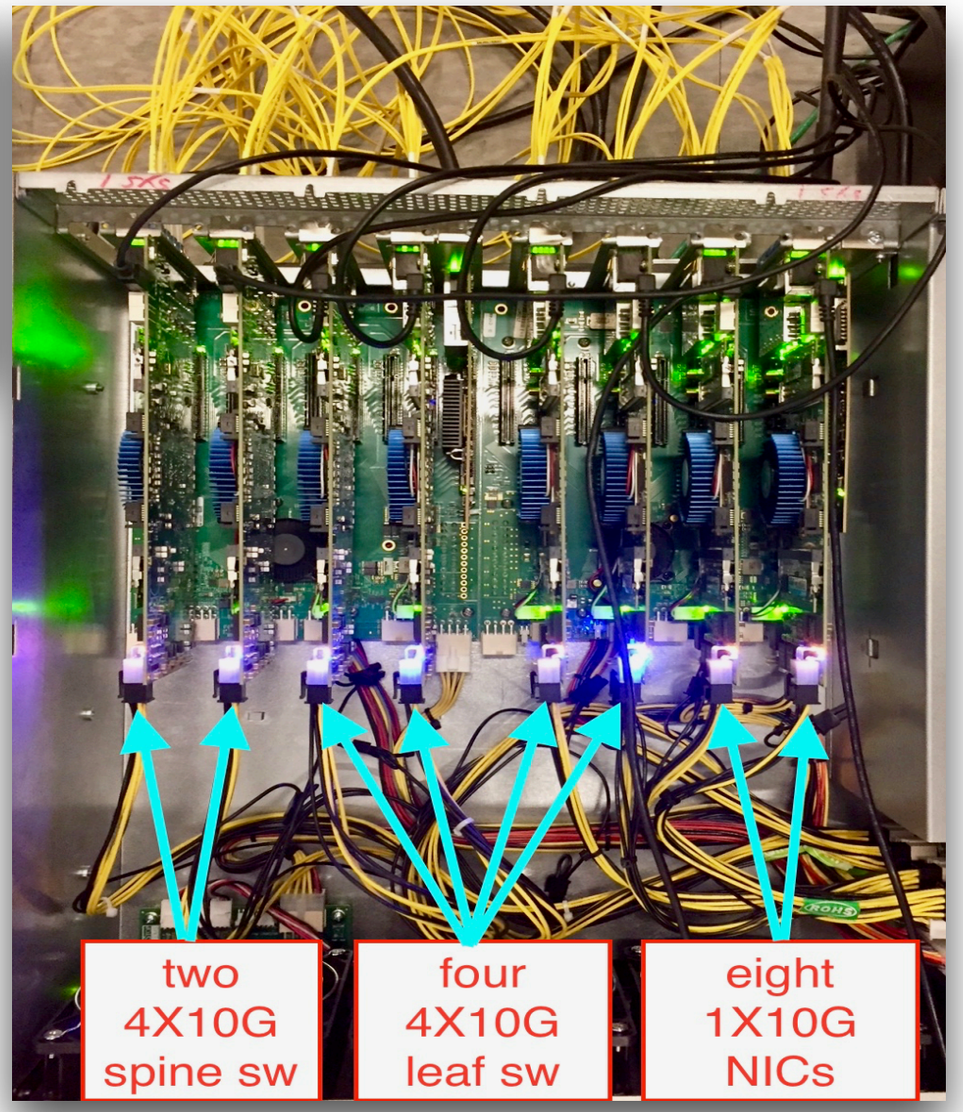
Implementation

- Routing
- Congestion Control
- Scheduling
- Synchronization

Stratix V FPGA

- Circuit Switch
  - re-configures in < 6.4ns
  - fully pipelined design
- End-host NIC

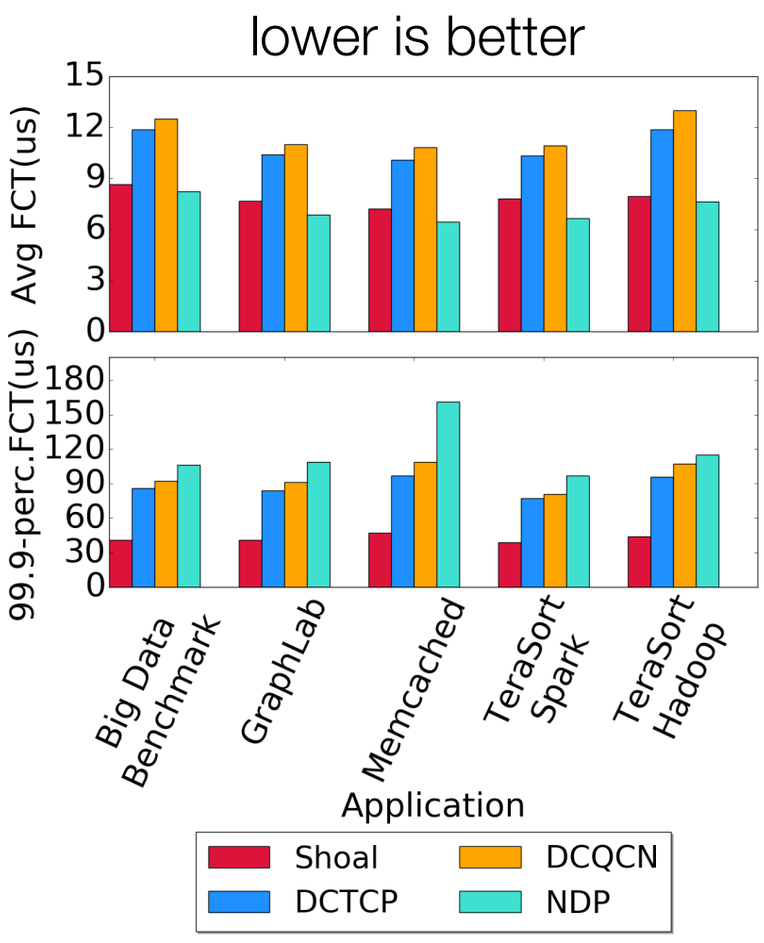




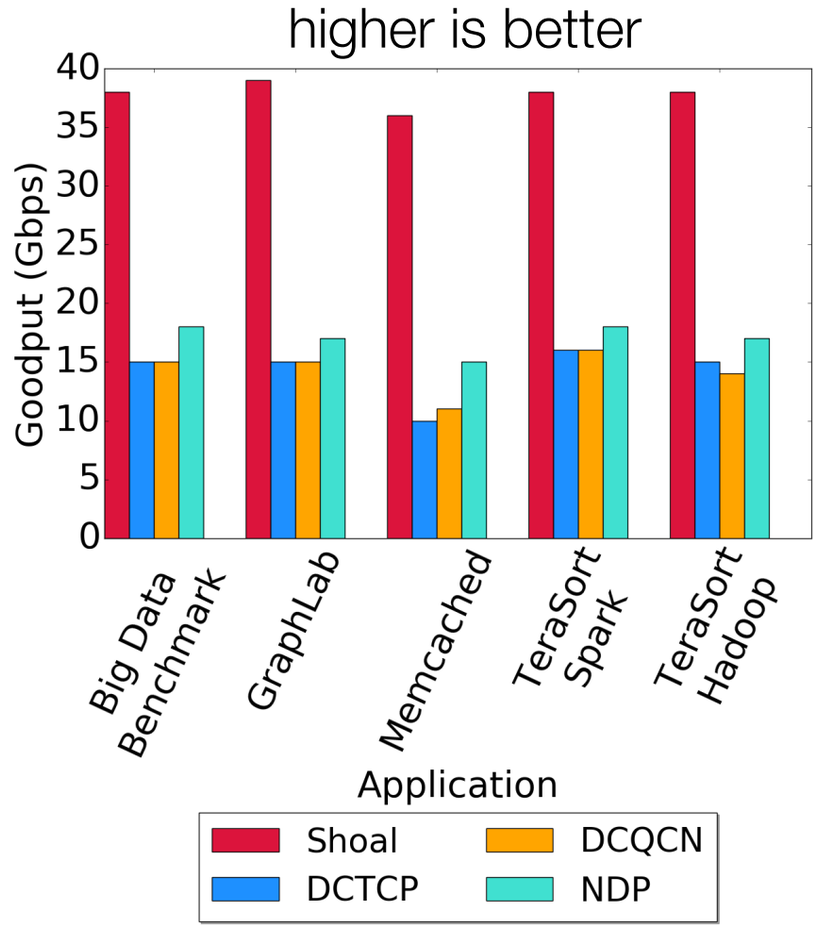
FPGA code for the implementation of Shoal is available at:
https://github.com/vishal1303/Shoal

Evaluation

- Packet-level simulator in C
- 512 end-hosts
- Shoal has 2x bandwidth
  (100Gbps vs. 50Gbps)
  at lower cost and power!



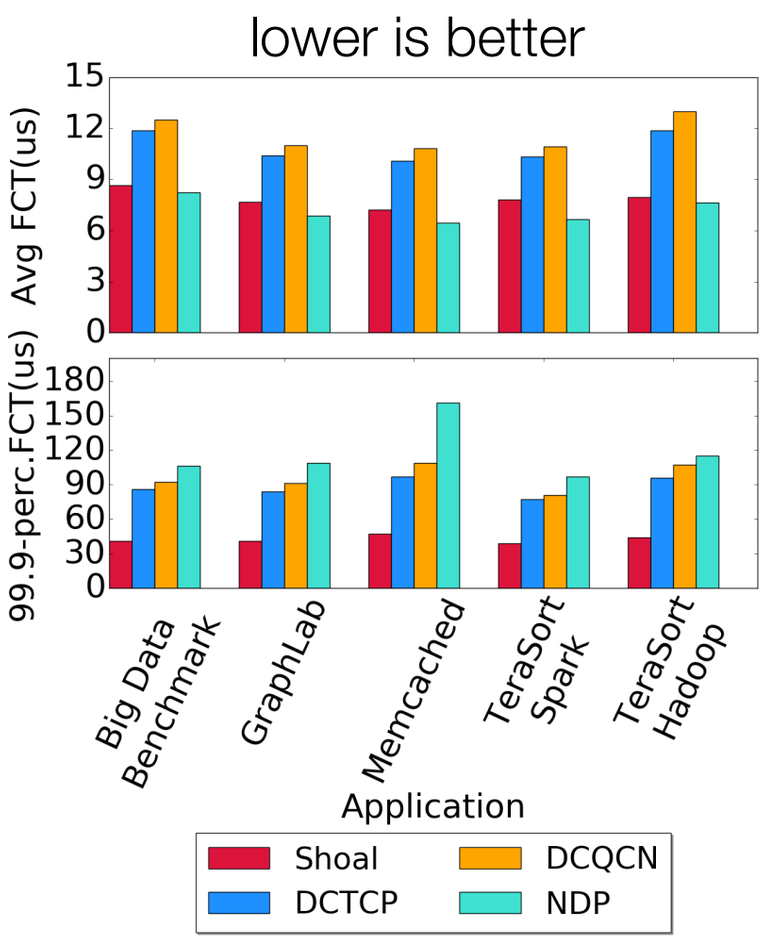
Short Flows (<100KB)



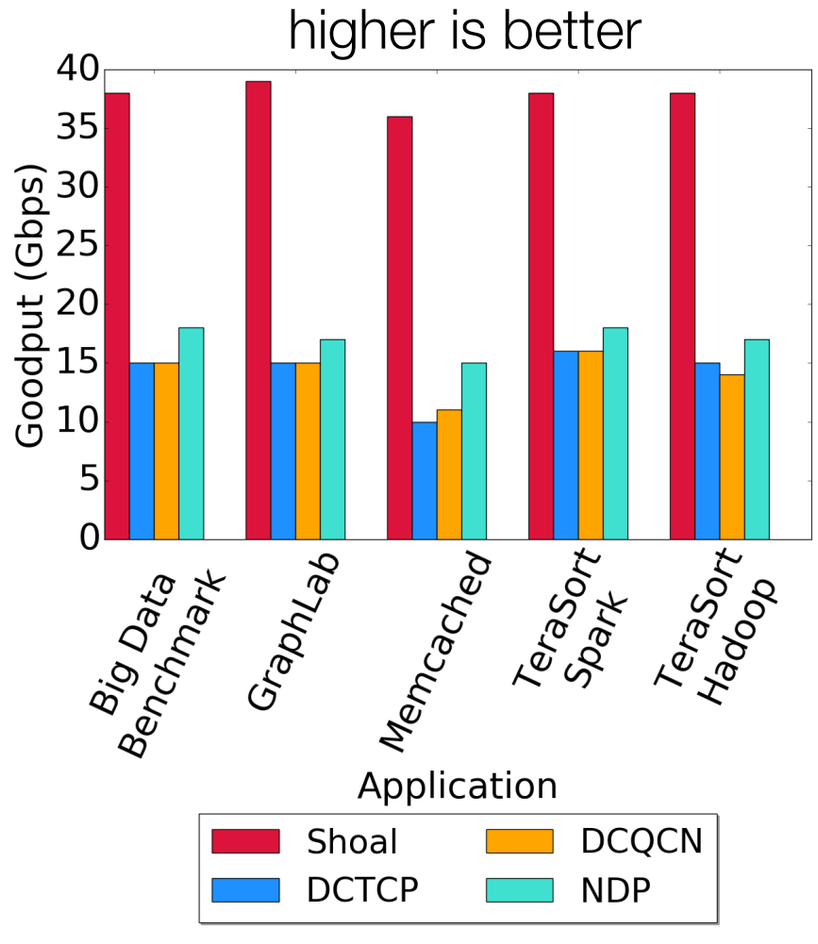
Long Flows (>1MB)

Evaluation

- Packet-level simulator in C
- 512 end-hosts
- Shoal has 2x bandwidth
  (100Gbps vs. 50Gbps)
  at lower cost and power!



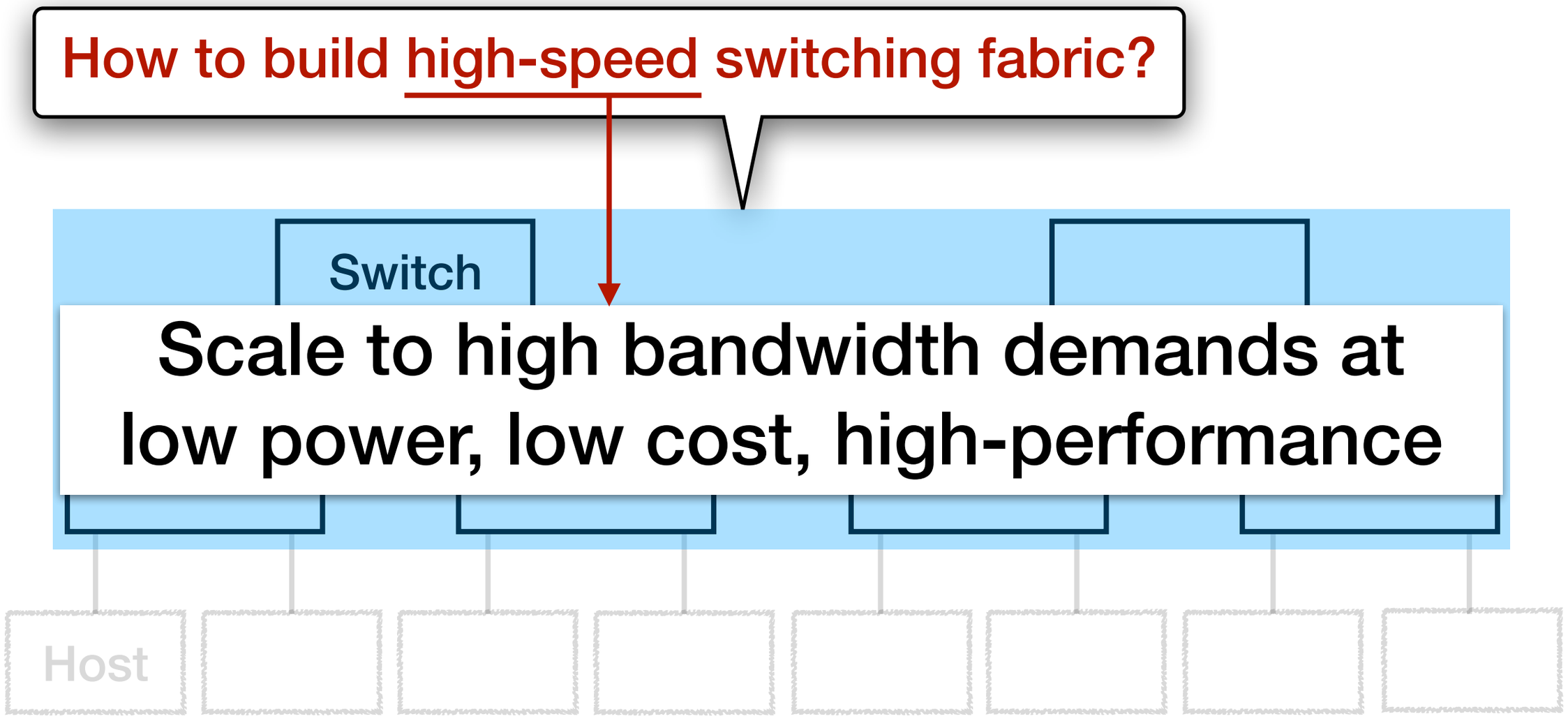
Short Flows (<100KB)



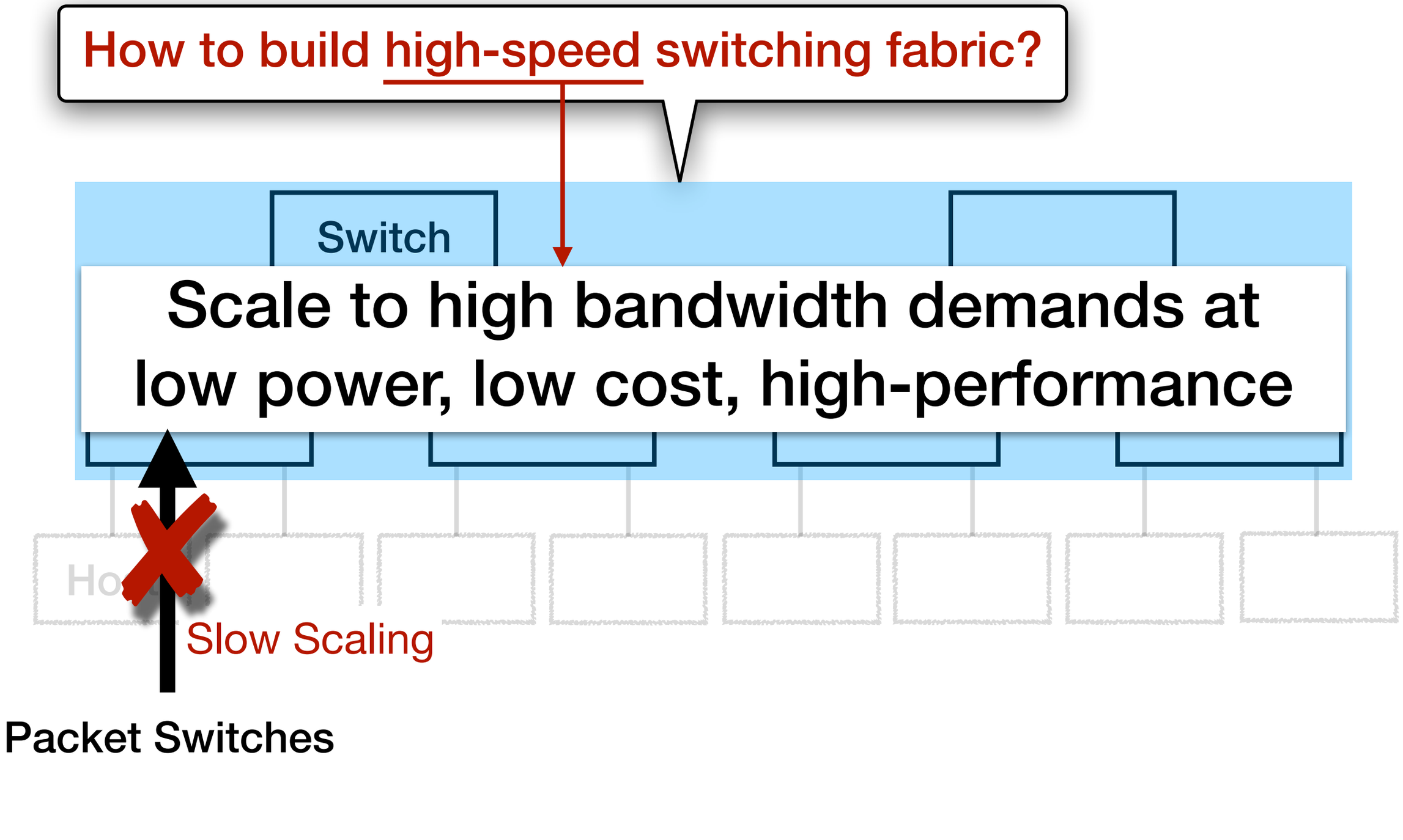
Long Flows (>1MB)

Shoal performs comparable or better than several recent packet-switched network designs

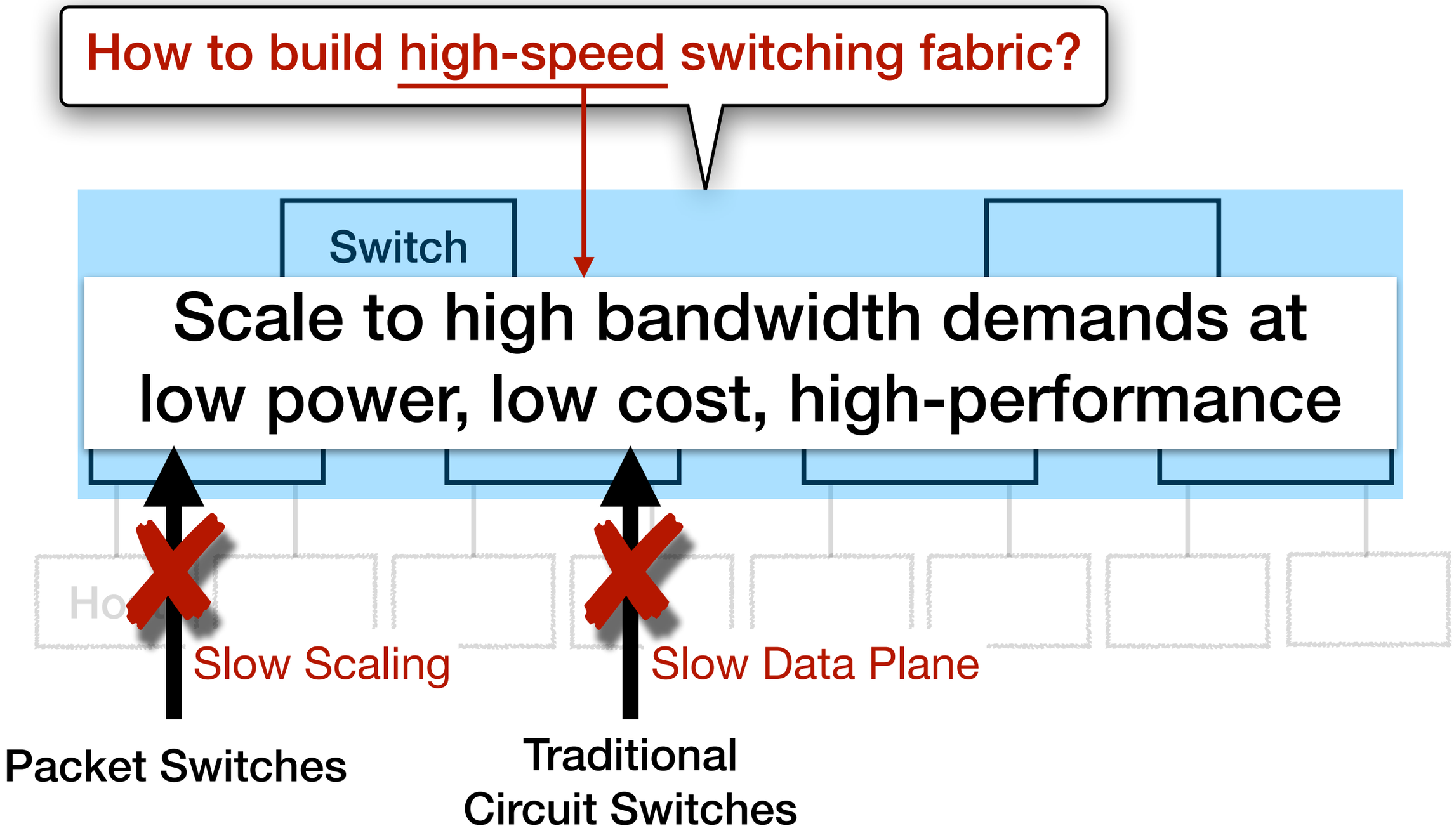
Part II : Summary



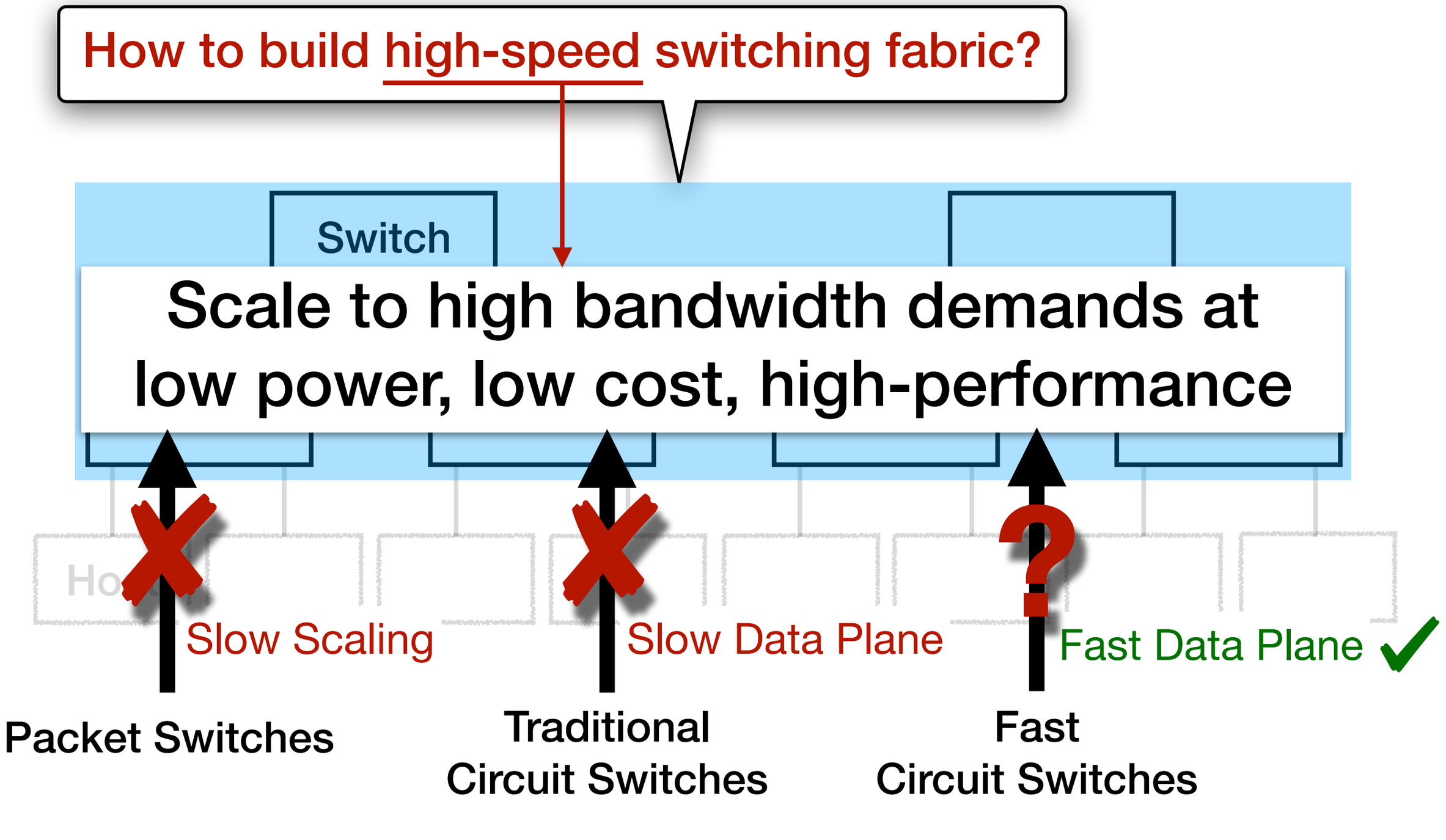
Part II : Summary



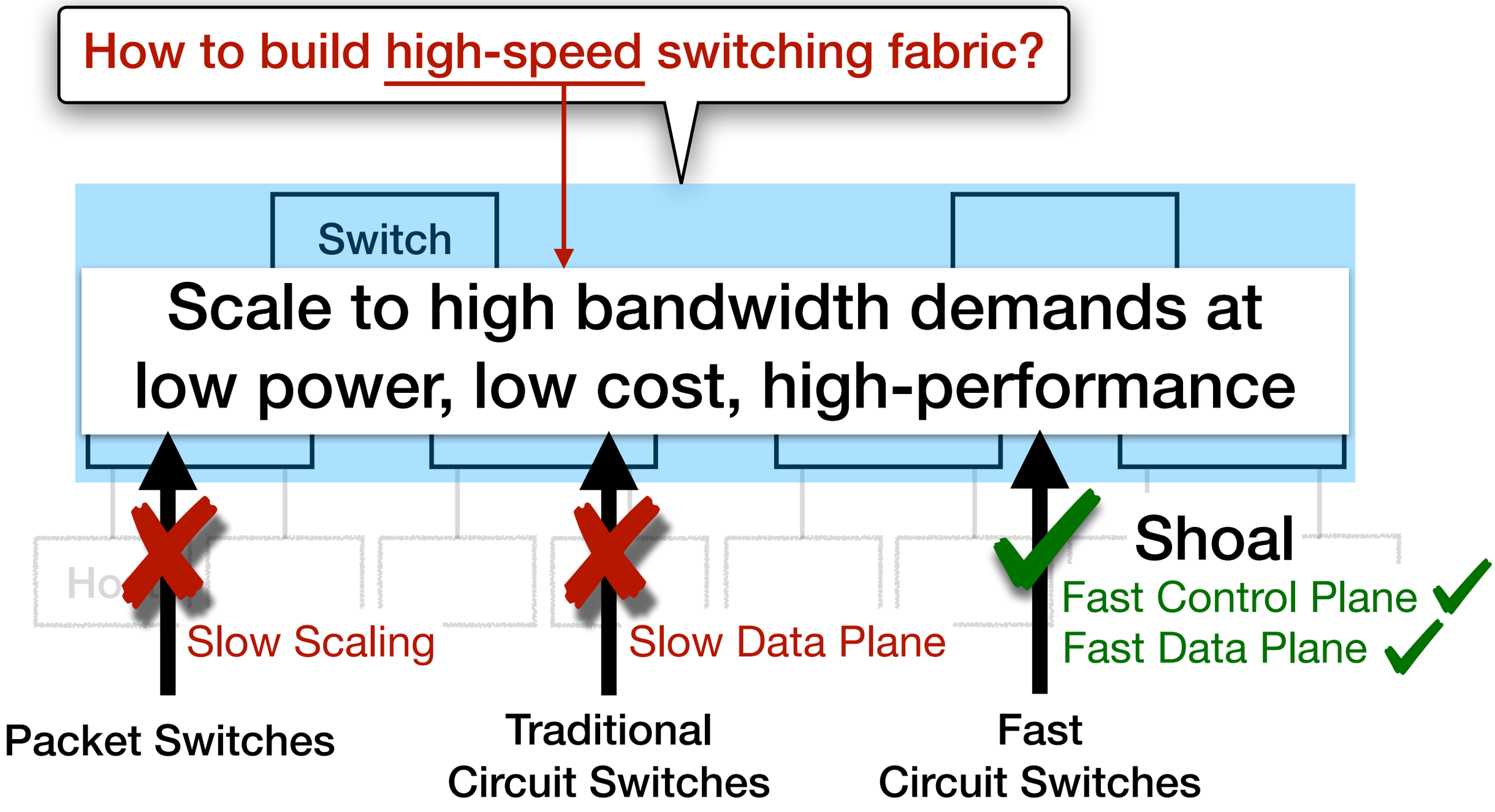
Part II : Summary



Part II : Summary



Part II : Summary



Future Directions

Future Directions

High-speed and Programmable Processors for (stateful) Network Functions

e.g., load balancers, firewalls, deep packet inspectors

Future Directions

High-speed and Programmable Processors for (stateful) Network Functions

e.g., load balancers, firewalls, deep packet inspectors

Low latency Optical Circuit Switching

Conclusion

Conclusion

Moore’s Law
Dennard Scaling

Conclusion

Moore's Law
Dennard Scaling

Networking Infrastructure Speed

end-host

General-purpose Processor

switching fabric

Packet Switch

Conclusion

Moore's Law
Dennard Scaling

Bandwidth Demand

Networking Infrastructure Speed

end-host

General-purpose Processor

switching fabric

Packet Switch

Conclusion

Moore's Law
Dennard Scaling

Bandwidth Demand

How to build high-speed networking infrastructure in the Post-Moore era?

Networking Infrastructure Speed

end-host

General-purpose Processor

switching fabric

Packet Switch

Conclusion

Moore's Law
Dennard Scaling

Bandwidth Demand

How to build high-speed networking infrastructure in the Post-Moore era?

Networking Infrastructure Speed

end-host | switching fabric

~~General-purpose Processor~~
Domain-specific Processor

Thesis

~~Packet Switch~~
Fast Circuit Switch

Conclusion

Moore's Law
Dennard Scaling

Bandwidth Demand

How to build high-speed networking infrastructure in the Post-Moore era?

Networking Infrastructure Speed

end-host | switching fabric

~~General-purpose Processor~~ Domain-specific Processor	**Thesis**	~~Packet Switch~~ Fast Circuit Switch
Programmability vs. Performance?	**Challenge**	Fast Control Plane?

Conclusion

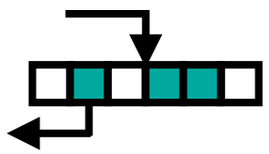
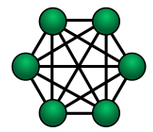
Moore's Law
Dennard Scaling

Bandwidth Demand

Networking Infrastructure Speed

How to build high-speed networking infrastructure in the Post-Moore era?

end-host | switching fabric

~~General-purpose Processor~~ Domain-specific Processor	**Thesis**	~~Packet Switch~~ Fast Circuit Switch
Programmability vs. Performance?	**Challenge**	Fast Control Plane?
PIEO  Programmable and high-performance domain-specific processor for packet scheduling	**Contributions**	**Shoal**  High-performance, ns-granularity control plane for circuit switching

Thank you!