A Simpler and Faster NIC Driver Model for Network Functions

Solal Pirelli,
George Candea

EPFL

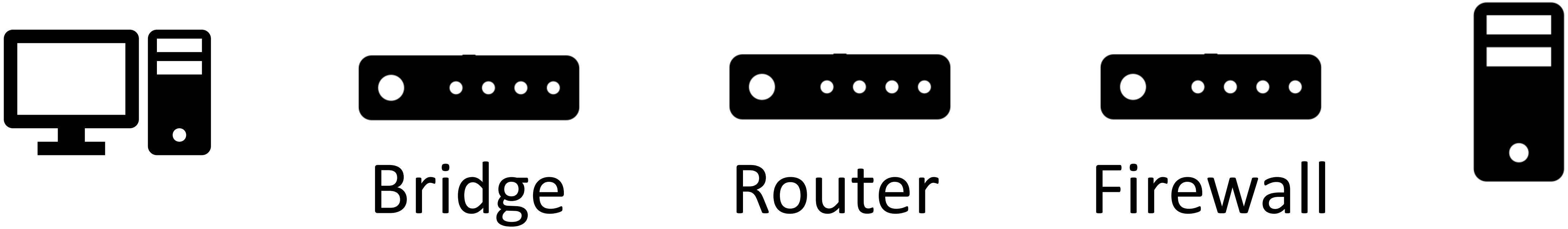
ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

FUNCTIONAL

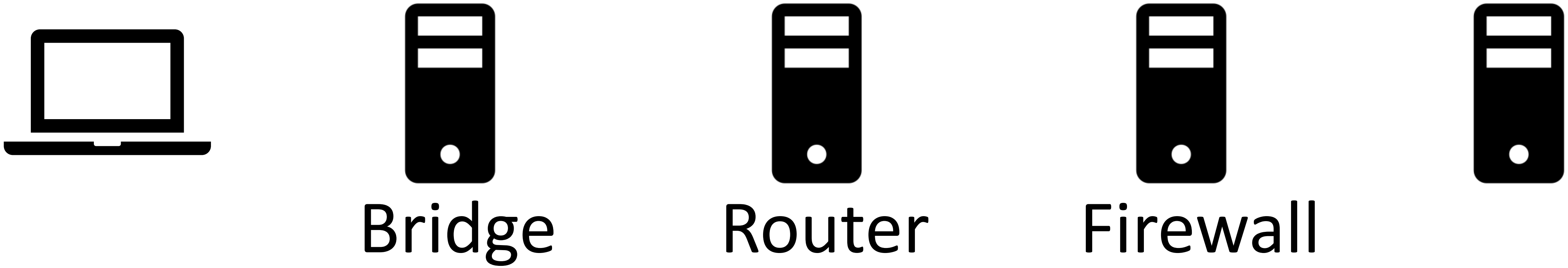
REPRODUCED

Designing for verification can help with performance!

Network



Network future

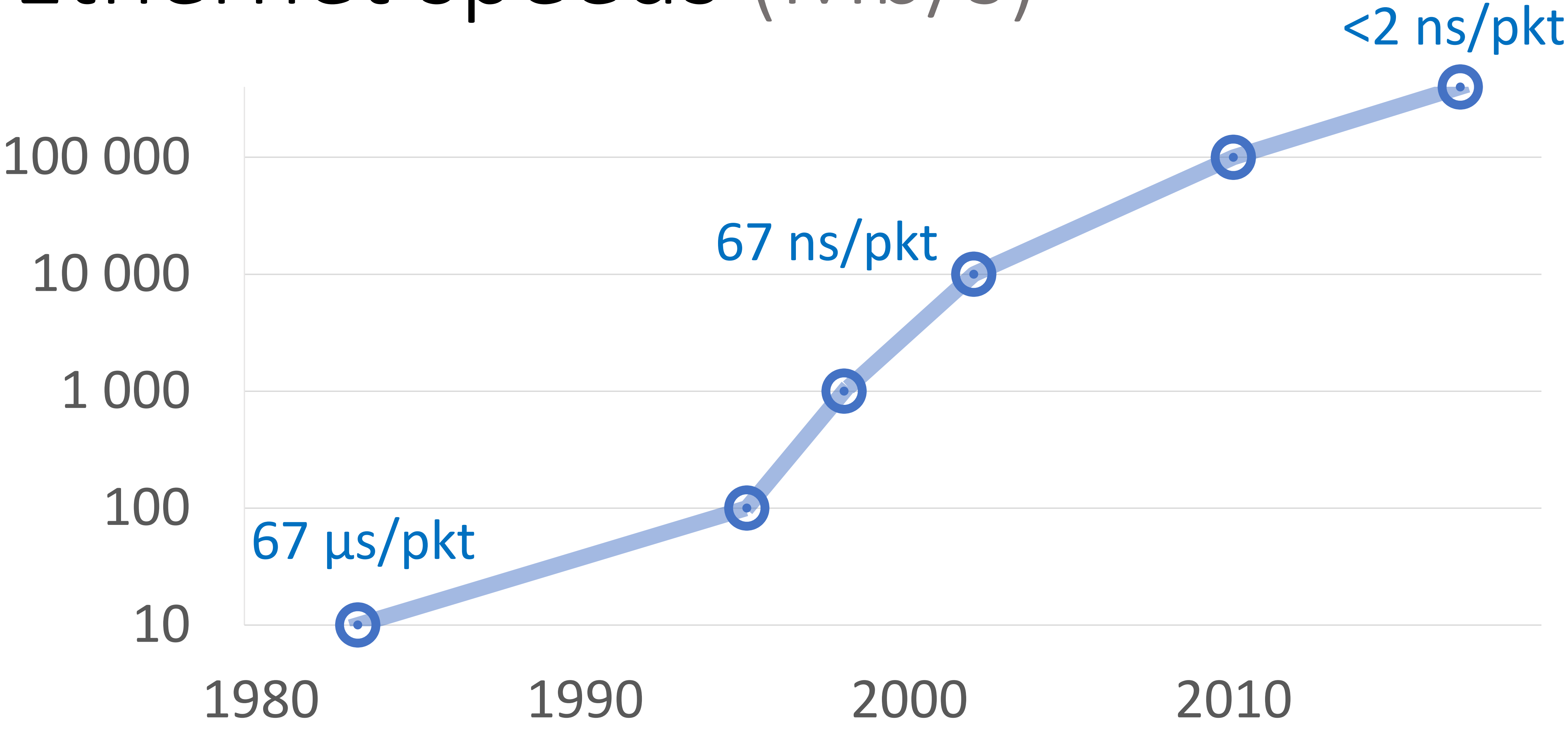


↑ Flexibility

↓ Dependability

↓ Performance

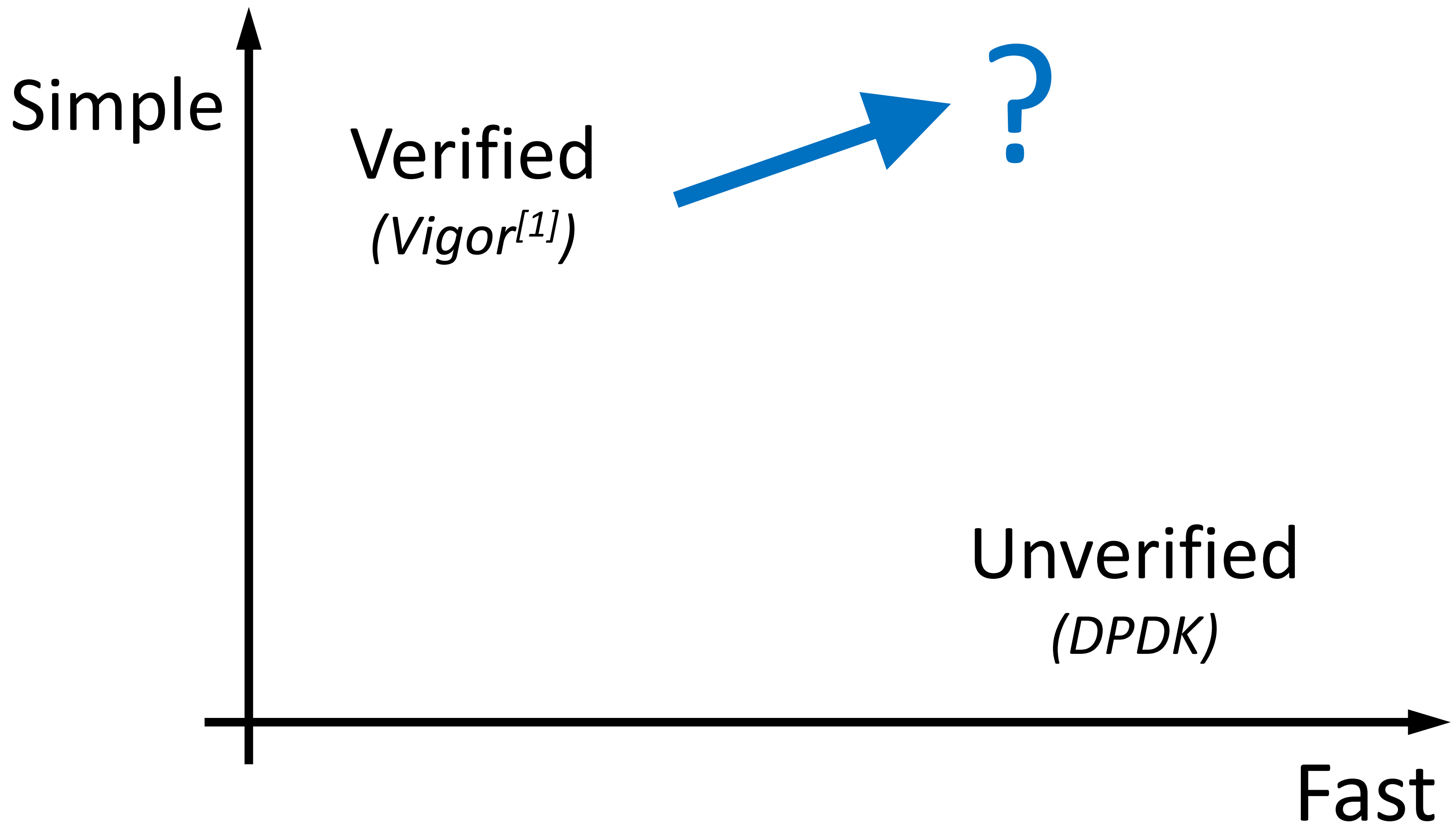
Ethernet speeds (Mb/s)



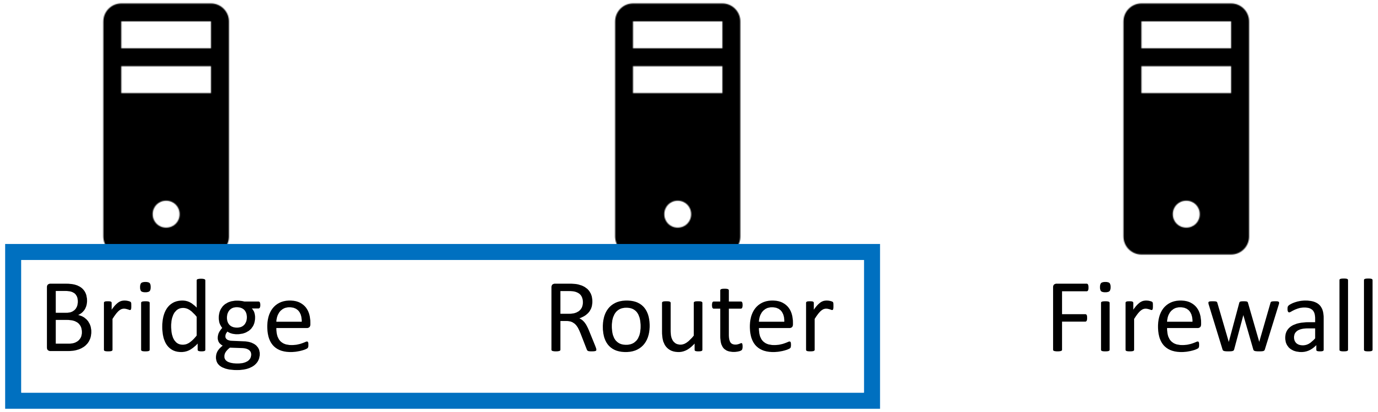
Verifying fast stacks

Network Function *NAT, IP router, firewall, bridge, …* **I/O framework** *DPDK, Netmap, Click, …*	p4v (SIGCOMM'18) Alembic (NSDI'19) Vigor (SOSP'19) Gravel (NSDI'20) …
Network Driver	Complexity bottleneck!
Network Card	Out of scope

The driver bottleneck



[1]: A. Zaostrovnykh et al., Verifying software network functions with no verification expertise, SOSP'19



Process packets 1 by 1, in order

also: NAT, load balancer, … non-TCP functions

Core networking can be fast and simple (and verified)

Key results

25% more throughput than full DPDK

160% more than Vigor

8x faster to verify

Pure C implementation

How?

New driver model

Efficient use of NIC

From-scratch implementation

Outline

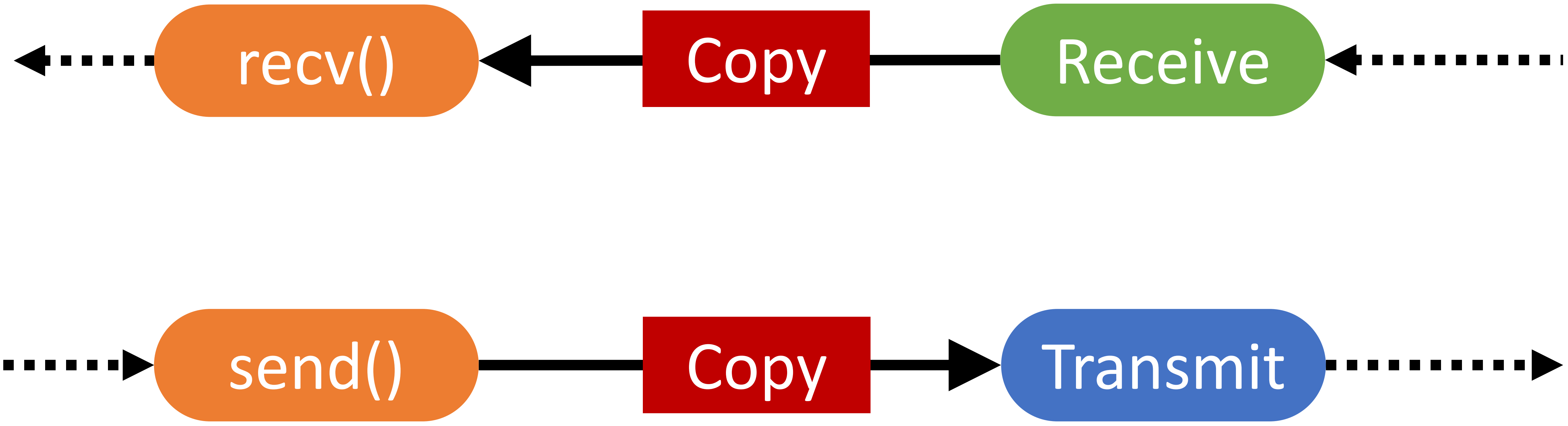
Intro

Design

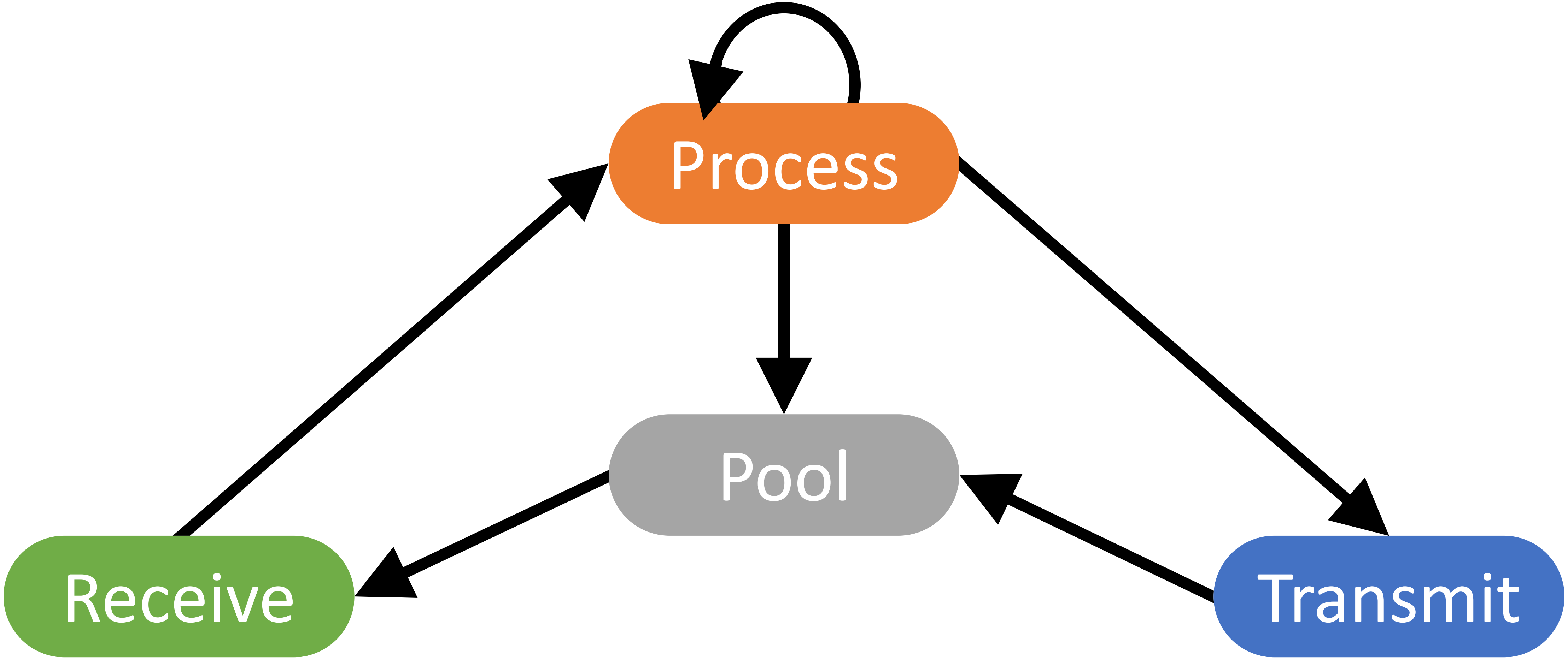
Implementation

Evaluation

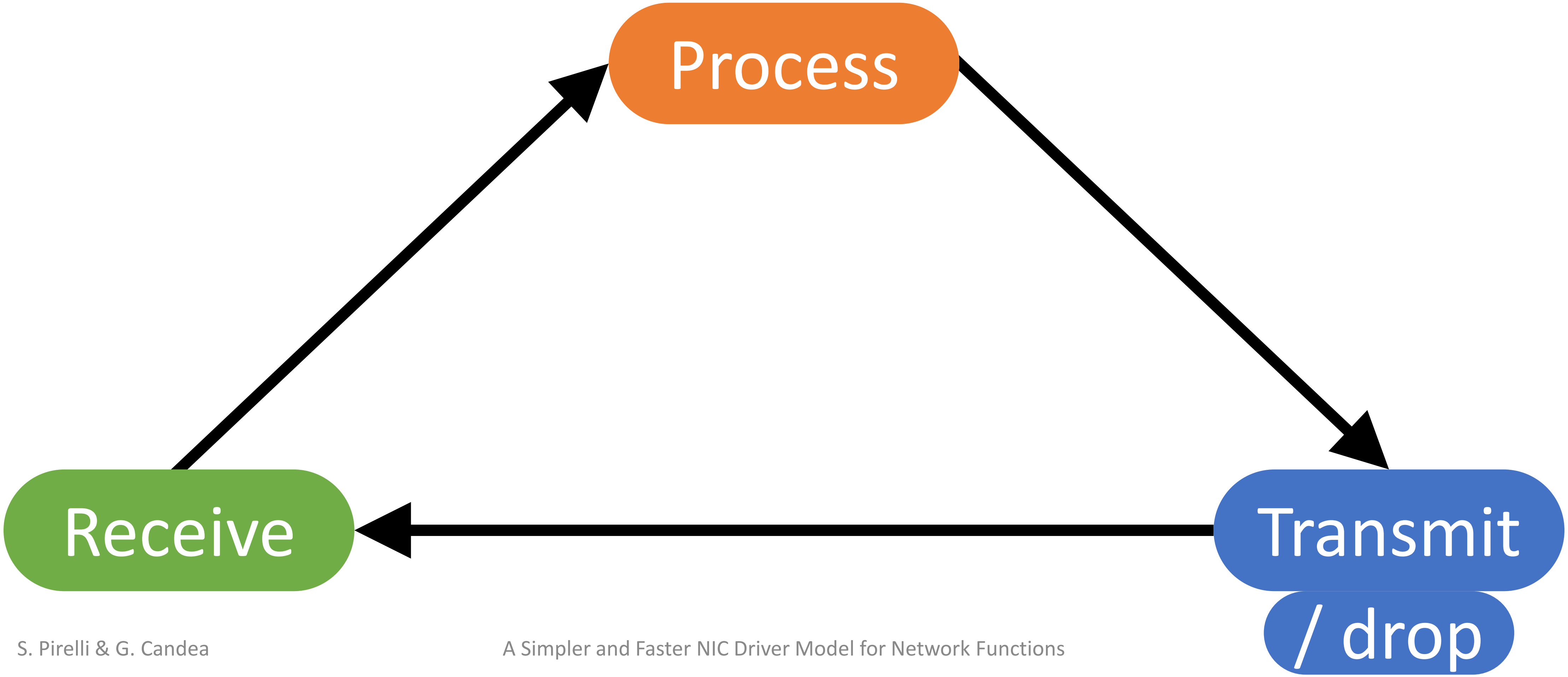
Classic model (e.g., BSD)



Closed model (e.g., DPDK)



Our model: “TinyNF”



Outline

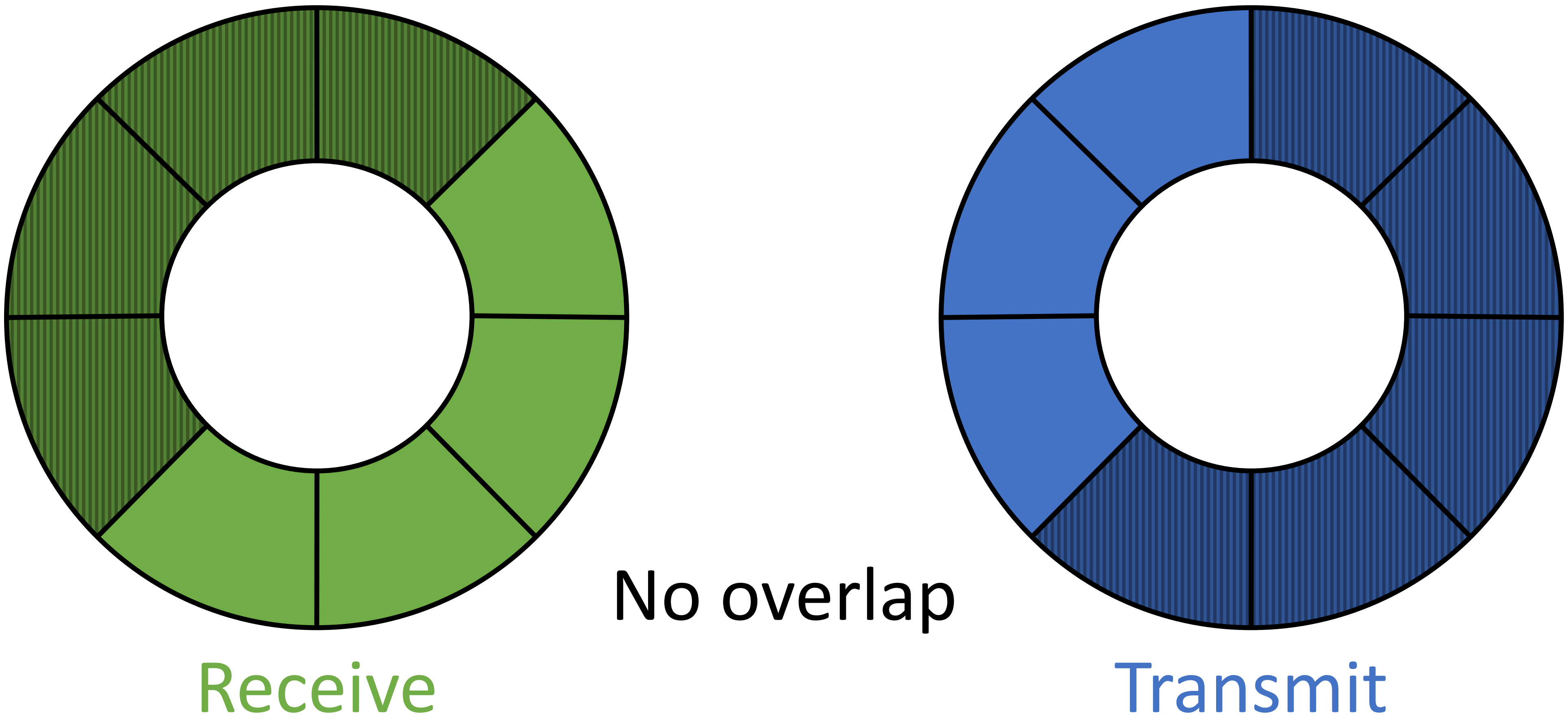
Intro

Design

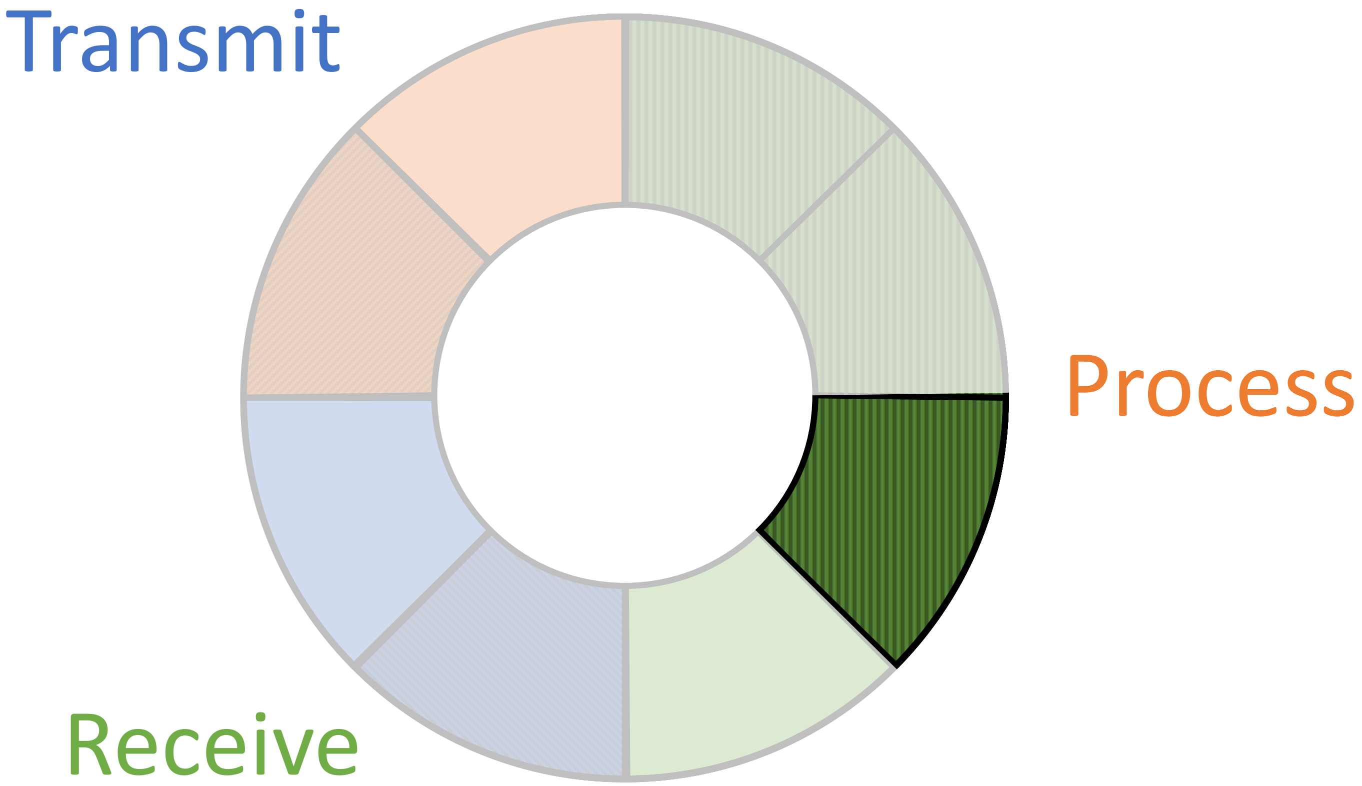
Implementation

Evaluation

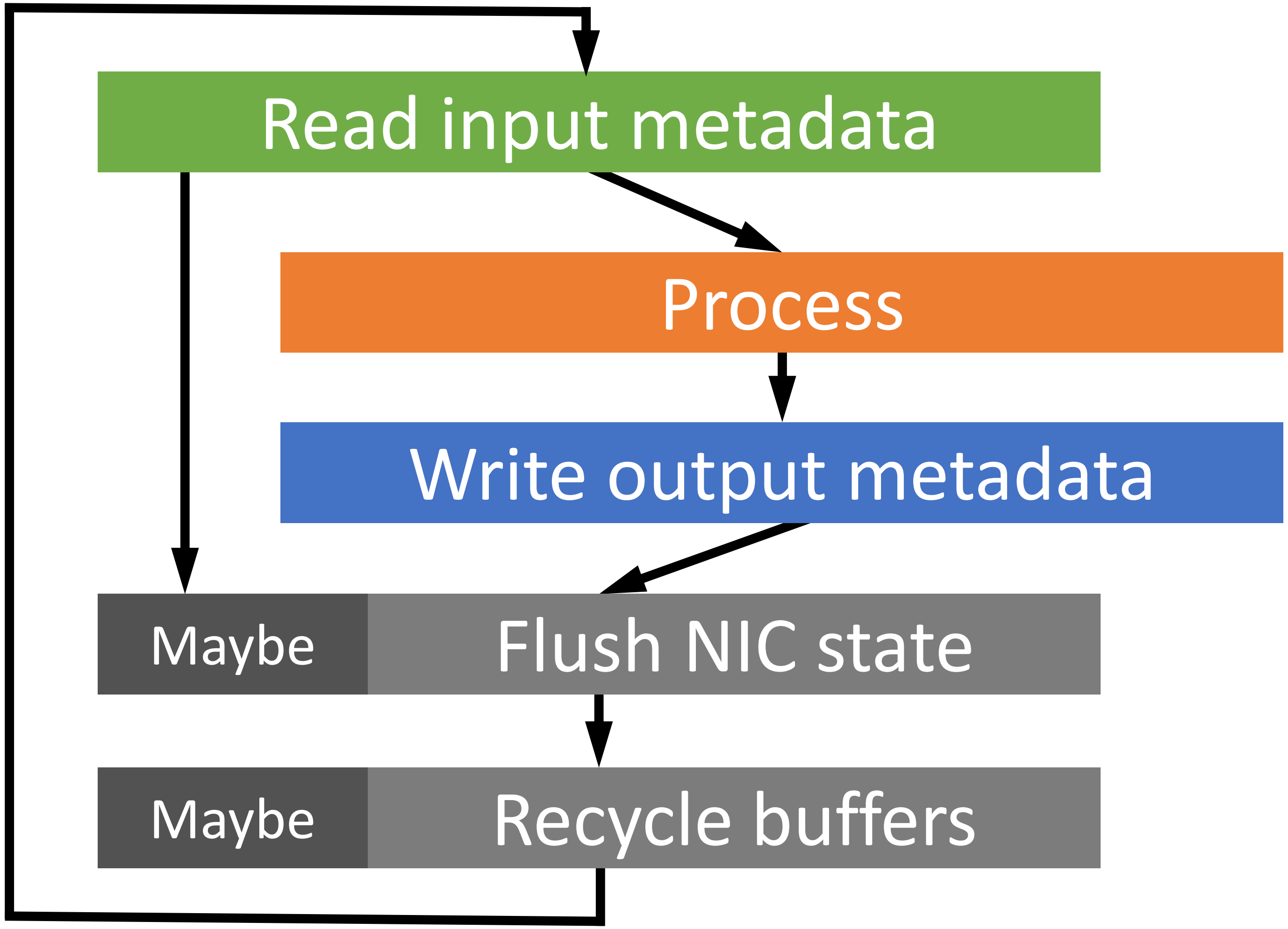
Separate rings



Merged rings



Packet path



No pointer changes
No pool operations
No explicit batches

Flushing NIC state

Expensive operation

DPDK “batching” estimates network load

We flush when idle or every N packets

Code

Driver: 550 lines

Environment abstraction: 300 lines

(endianness, memory, PCI, time)

100% user mode

Writing drivers

Not as complex as one would think

Publicly available data sheet

Many interpretations; most are trivial

Outline

Intro

Design

Implementation

Evaluation

Evaluation

NAT, Bridge, Policer, Firewall, Load Balancer

Throughput, latency, complexity

Baseline: DPDK

Complexity

	Lines of code		Number of paths (L = #links)	
	DPDK	TinyNF	DPDK	TinyNF
Init	3204	245	-	-
Receive	136	17	$1 + A_F + 288A_S$	3
Transmit	122	29	$(8 + 14(F_F^T + P((F_S + F_F)^T - F_F^T))^L$	$2 + 2^L$

➔ 7x fewer paths in real NFs

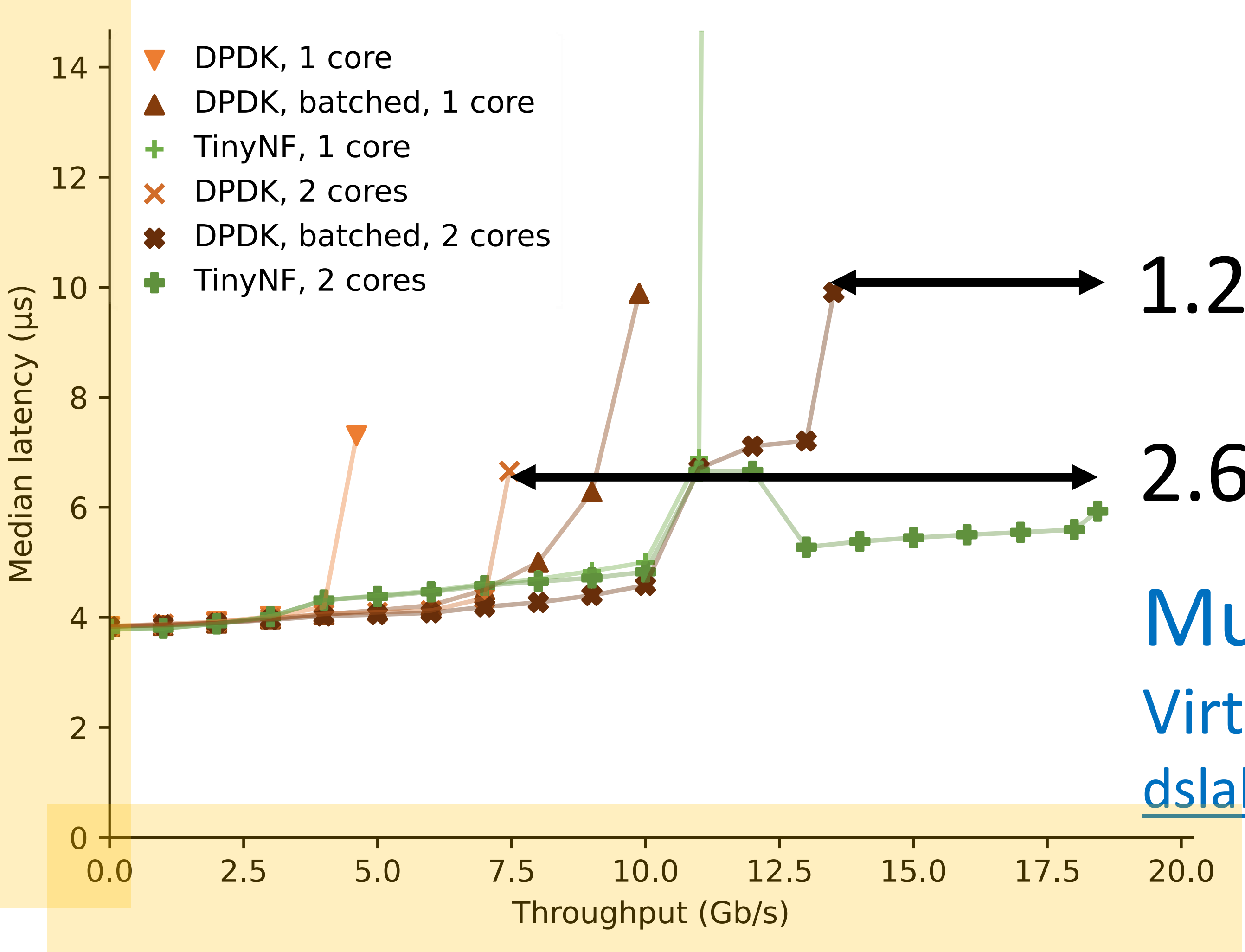
➔ 8x lower verification time

Complexity

	Lines of code		Number of paths (L = #links)	
	Ixy[1]	TinyNF	Ixy[1]	TinyNF
Init	3204	245	-	-
Receive	136	17	$1 + A_F + A_S$	3
Transmit	122	29	14^L	$2 + 2^L$

[1]: P. Emmerich et al., User Space Network Drivers, ANCS'19

Performance



1.25x vs. unverified

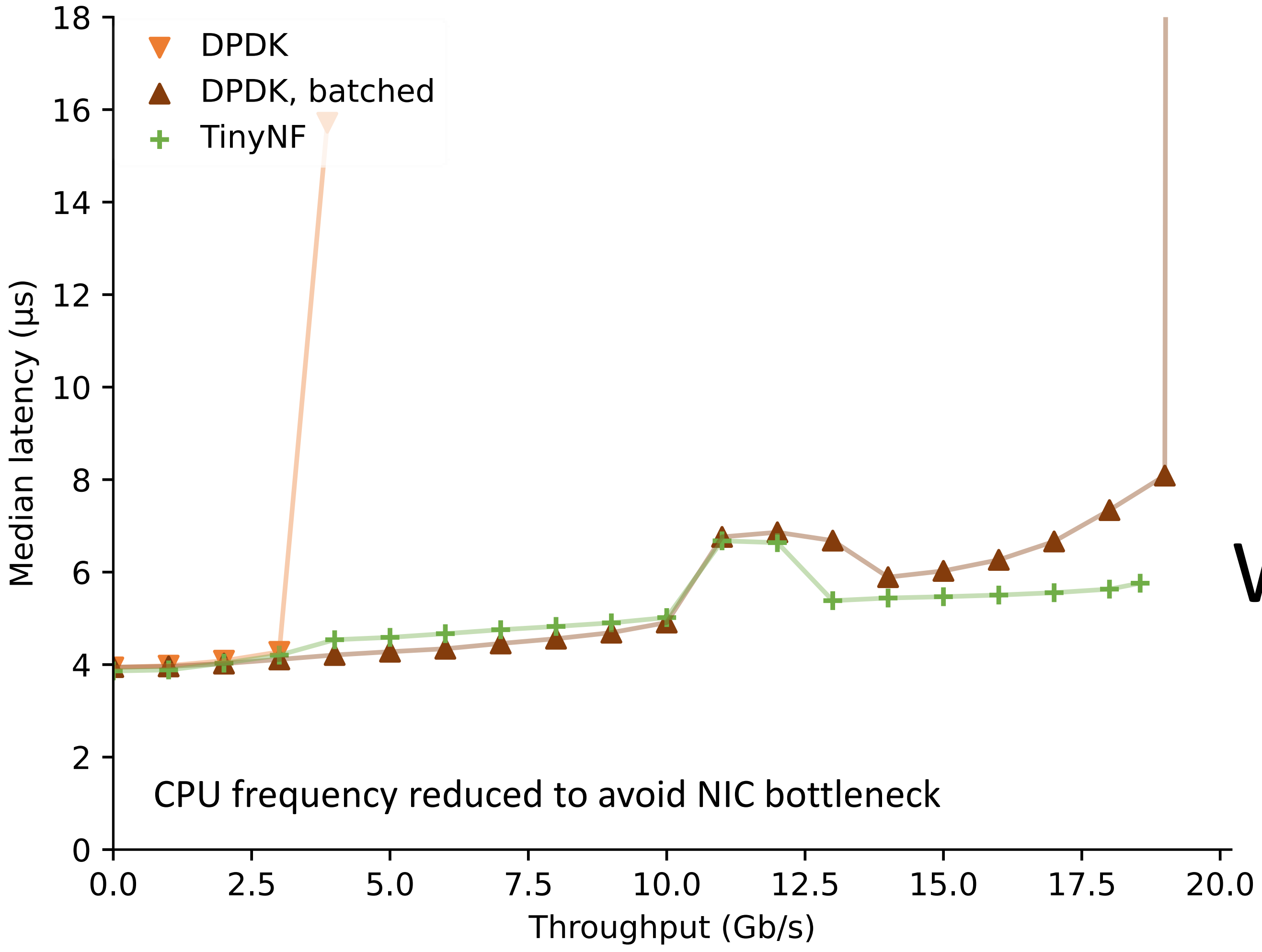
2.6x throughput vs. verified

Much more in the paper

Virtualization, low-level metrics…

dslab.epfl.ch/research/tinynf

Performance



Worse in “no-op”

Low-level performance

35 instructions to receive + transmit

100 for DPDK

Lower cache footprint

➔ Less interference

Conclusion

Designing for verification can help performance!

Core network functions
can be both fast and verified

dslab.epfl.ch/research/tinynf

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED