The nanoPU: A Nanosecond Network Stack for Datacenters

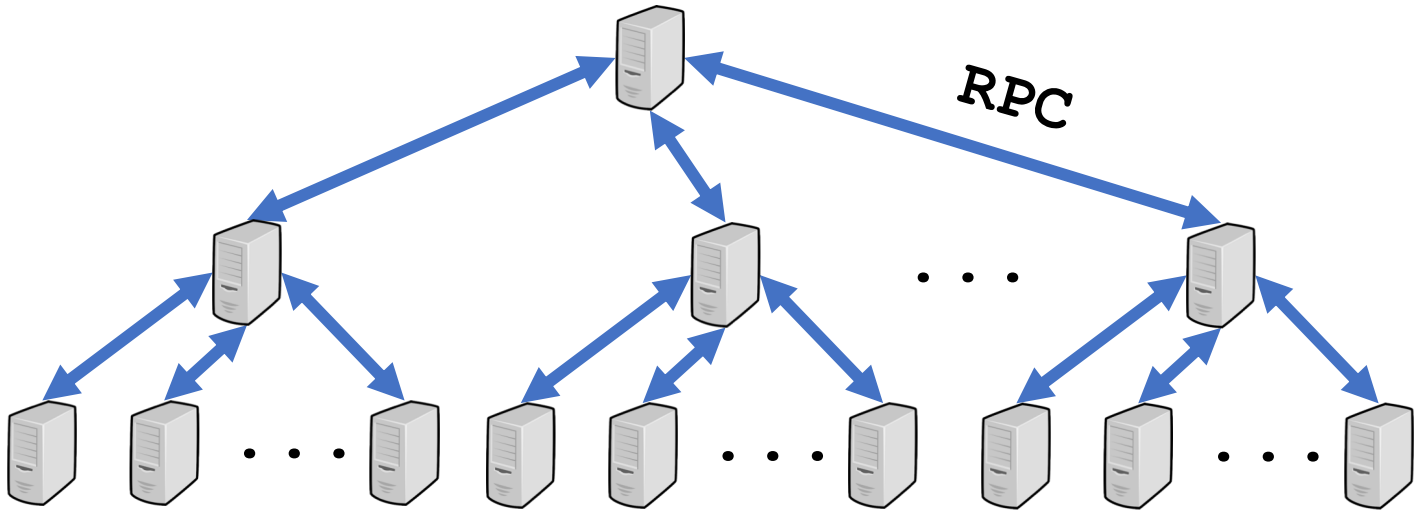
Stephen Ibanez, Alex Mallery, Serhat Arslan, Theo Jepsen, *Muhammad Shahbaz, Changhoon Kim, Nick McKeown

*Stanford University, *Purdue University*

The Need to Minimize RPC Latency and Software Overheads

Large Online Interactive Services

- Web Search
- Recommendation systems
- Online transaction processing



Fine-grained Computing

- Video encoding (ExCamera NSDI'17)
- Object classification (Sprocket SoCC'18)
- Software compilation (gg ATC'19)
- MapReduce-style analytics (Locus NSDI'19)
- Flash Bursts (NSDI '21)

The Need to Minimize RPC Latency and Software Overheads

Large Online Interactive Services

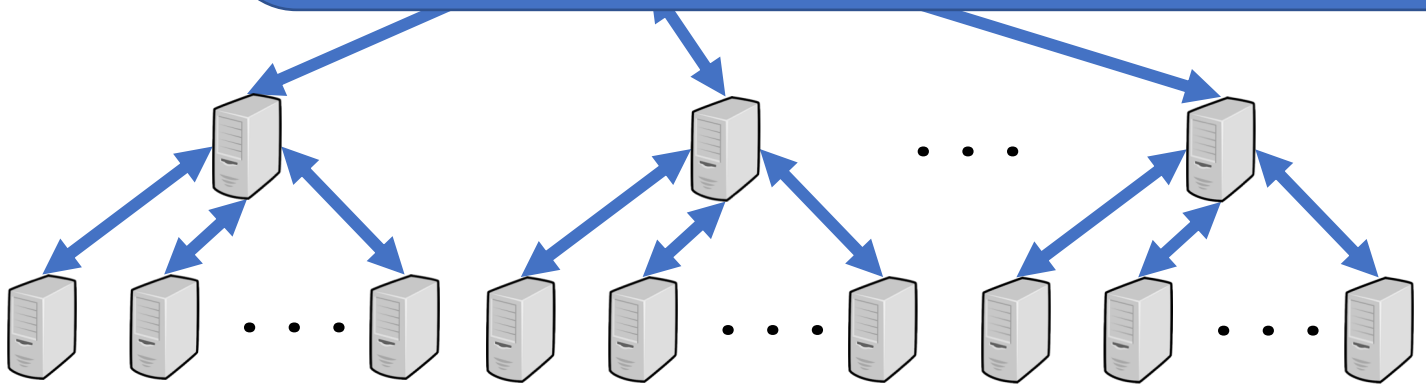
- Web Search
- Re
- O

Fine-grained Computing

- Video encoding (ExCamera NSDI'17)
- ocket
- ATC'19)
- cs (Locus
- Flash Bursts (NSDI '21)

Question:

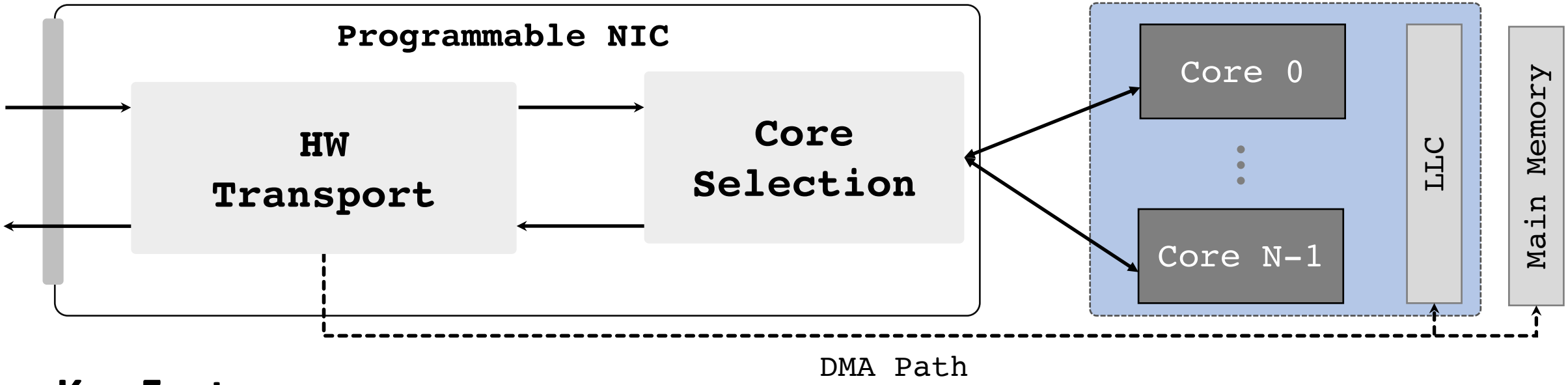
What would it take to *absolutely minimize* RPC median and tail latency as well as software processing overheads?



Previous Approaches are Insufficient

Approach	Limitation	Wire-to-Wire Latency	RPC Throughput
Dataplane operating systems (e.g. Shinjuku, Shenango)	Too coarse grained	Median: ~2-5µs Tail: 10-100µs	<10Mrps
Efficient RPC software libraries (e.g. eRPC)	Neglect tail latency optimizations	Median: 850ns Tail: 10-100µs	~10Mrps / core
Transport protocol offload (e.g. Tonic)	Only part of the solution	N/A	>100Mrps
RDMA NICs	Need low latency to remote *compute*, not *memory*	Median: ~700ns	N/A
Integrated NICs (e.g. NeBuLa)	Still room for improvement of latency and throughput	Median: ~100ns Tail: ~2-5µs	~20Mrps / core

The nanoPU

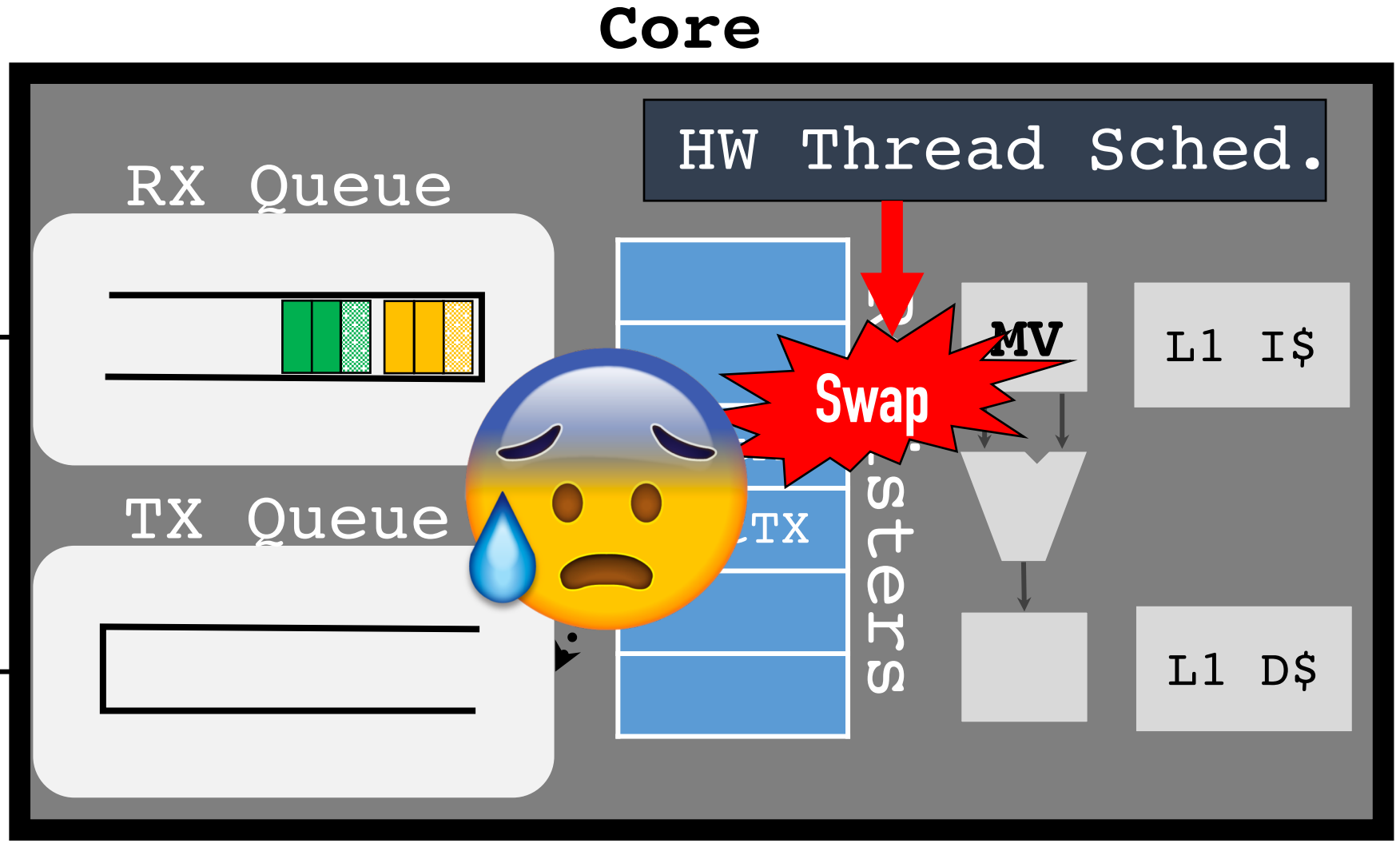


Key Features:

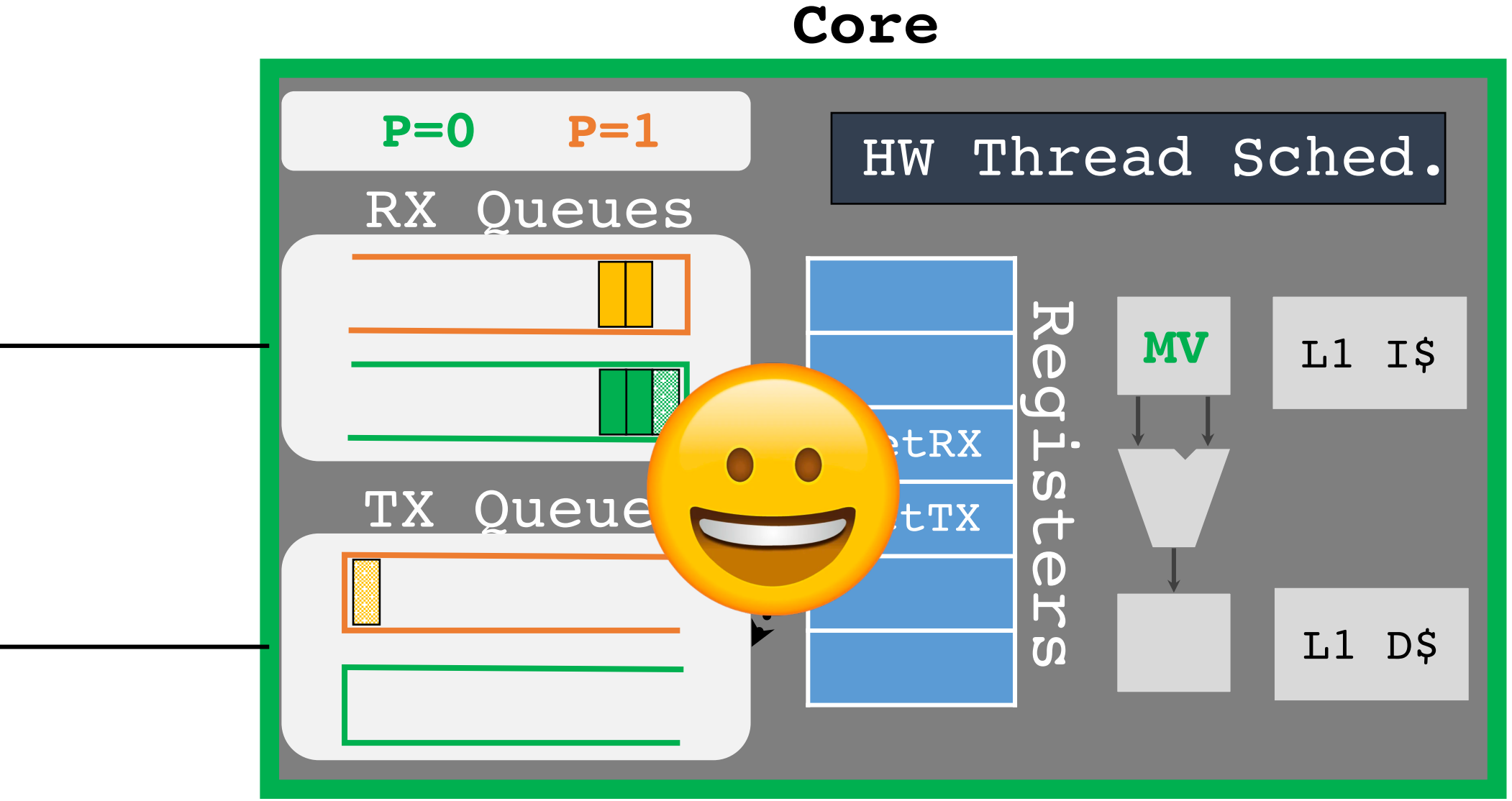
- Integrated NIC
- Efficient core selection in HW
- Programmable transport in HW
- Direct path to CPU register file
- Hardware-accelerated thread scheduling

Wire-to-wire latency: **69ns**
Single-core throughput: **118Mrps**

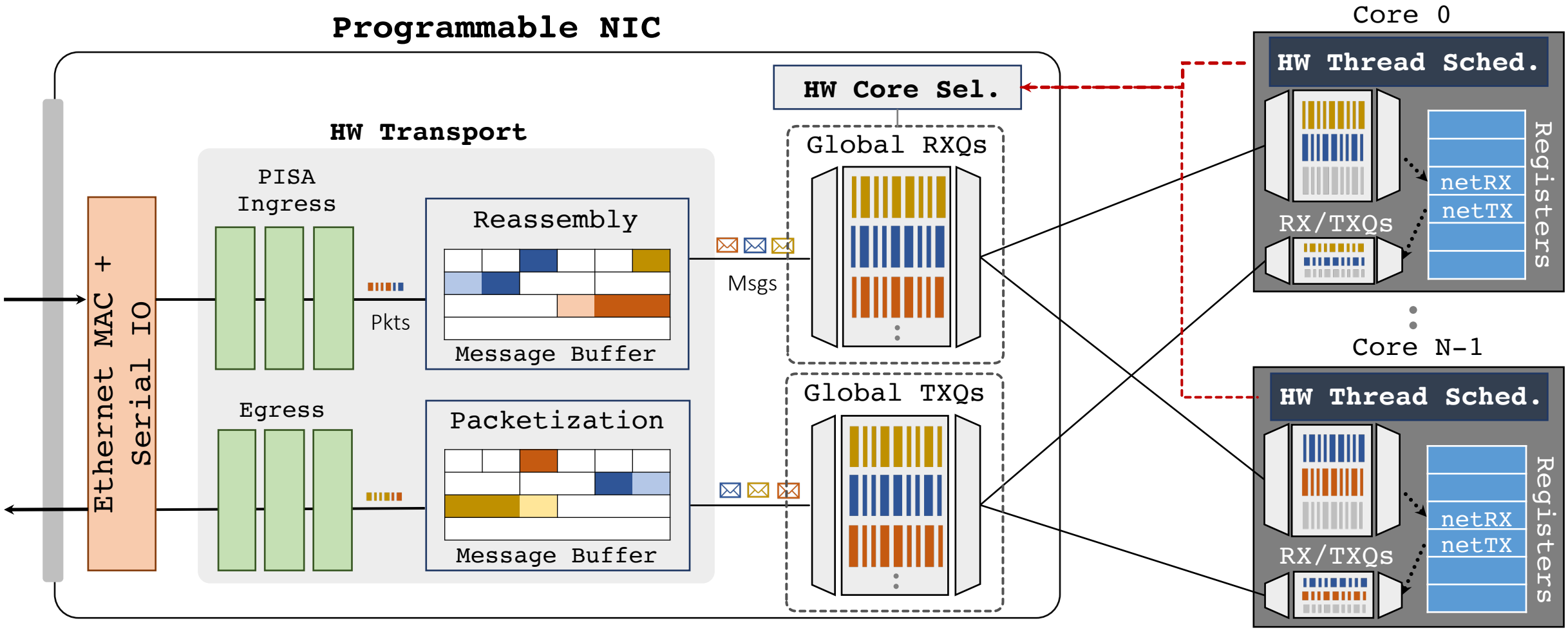
The nanoPU Core



The nanoPU Core

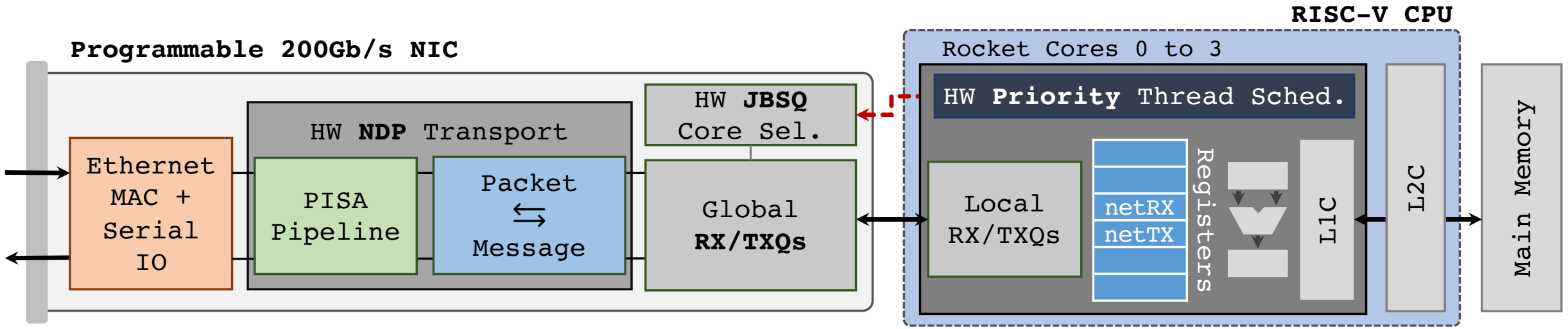


The nanoPU Fast Path

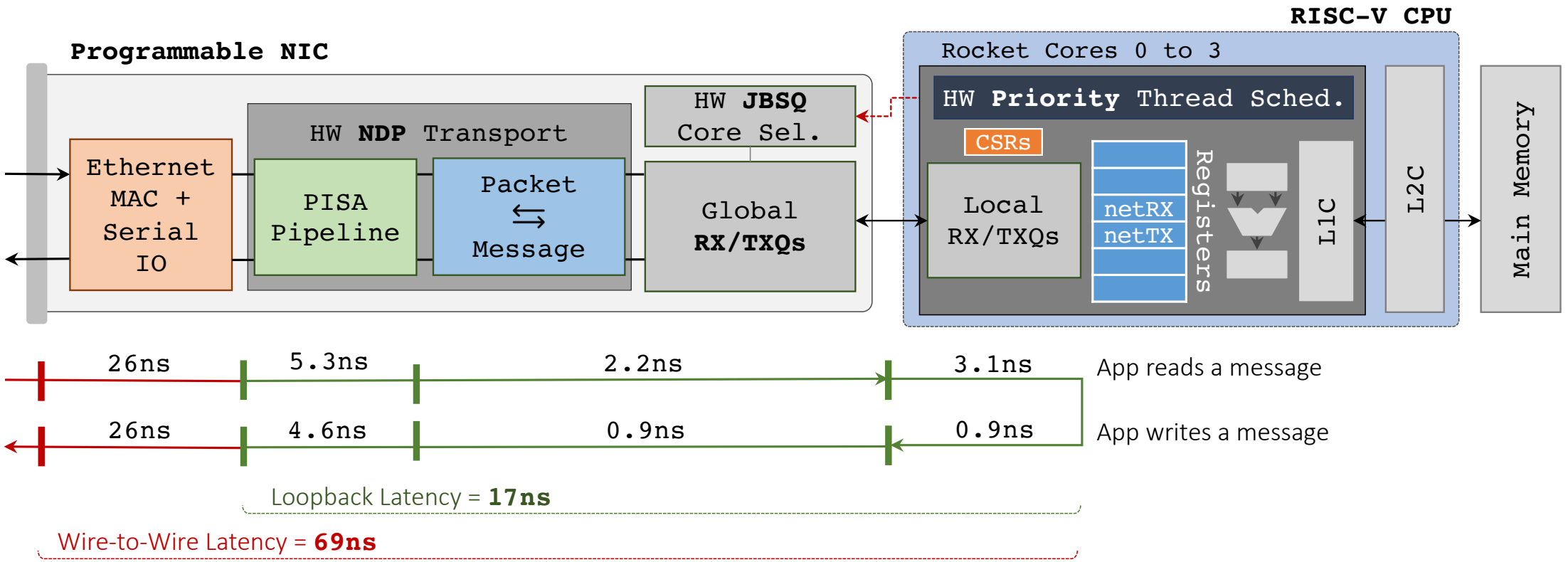


nanoPU Prototype

- Quad-core nanoPU based on open source RISC-V Rocket core
- 4,300 lines of Chisel code & 1,200 lines of C and RISC-V assembly for custom *nano*kernel
- Implements NDP transport
- Cycle-accurate simulations (3.2GHz) on AWS FPGAs using Firesim (ISCA '18)



Microbenchmarks



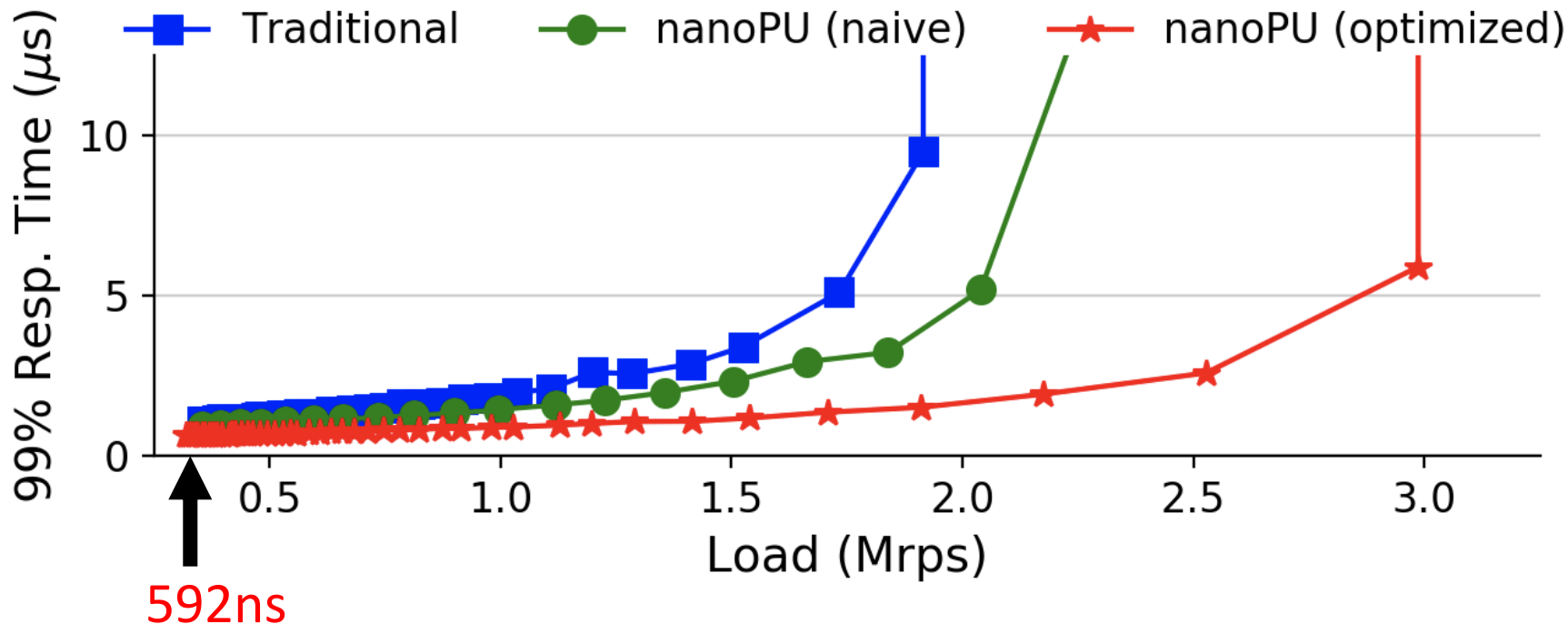
	Wire-to-Wire Latency (ns)	Single Core Loopback Throughput (Mrps)
nanoPU	**69**	**118**
IceNIC	103	16
eRPC	850	10

nanoPU HW Thread Scheduling

- Reduces tail latency by 4-6x
- 20% higher load

nanoPU Applications

- MICA Key-Value Store:

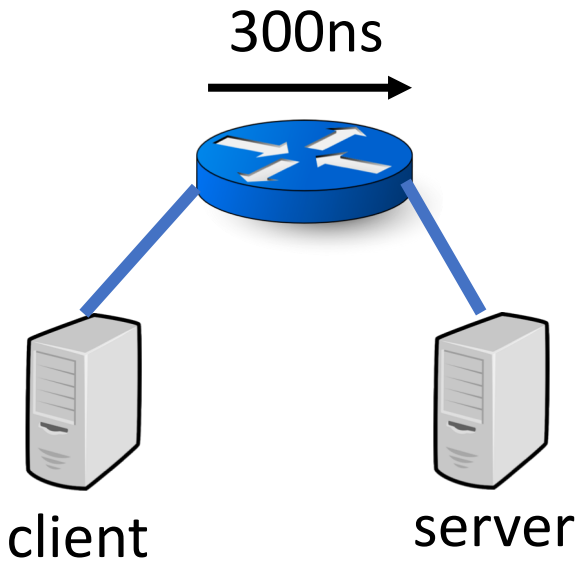


- Raft Consensus, Chain Replication, Set Algebra, and more!

Programmable One-Sided RDMA Operations

State-of-the-art RDMA NIC

- Implemented in NIC HW
- End-to-end latency: ~2µs



The nanoPU

- Implemented in SW
- Can support arbitrary one-sided operations

One-sided RDMA	Latency (ns)	
	Median	90th %ile
Read	678	680
Write	679	686
Compare-and-Swap	687	690
Fetch-and-Add	688	692
Indirect Read	691	715

nanoPU Conclusions

Key Takeaway:

To truly minimize median and tail RPC latency and SW overheads:

1. Fast path directly between network and CPU register file
2. Move entire network stack to HW: ***transport***, ***load balancing***, ***thread scheduling***

Challenges:

- Need to rewrite applications
- Figure out how to use more sophisticated processors

Thank You!

Contact Email:
sibanez@g.hmc.edu