5G Connected Edge Cloud for Industry 4.0 Transformation

ONF SPOTLIGHT

The **nanoPU**: Redesigning the CPU-Network Interface to Minimize RPC Tail Latency

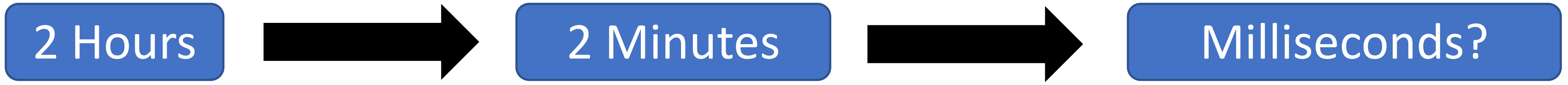
Stephen Ibanez, Alex Mallery, Serhat Arslan, Theo Jepsen, Muhammad Shahbaz, Changhoon Kim, Nick McKeown

Stanford University

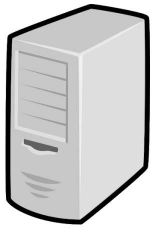
2020

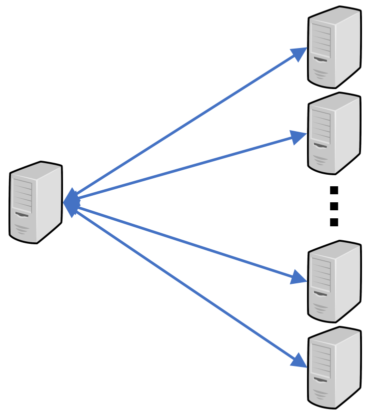
Towards Fine-Grained Computing

- Serverless computing (deployed as microservices) enables "fine-grained" computing on thousands of cores, reducing completion times:
  - Video encoding (ExCamera NSDI'17)
  - Object classification (Sprocket SoCC'18)
  - Software compilation (gg ATC'19)
  - MapReduce-style analytics (Pyren SoCC'17 , Flint CLOUD'18, Locus NSDI'19)

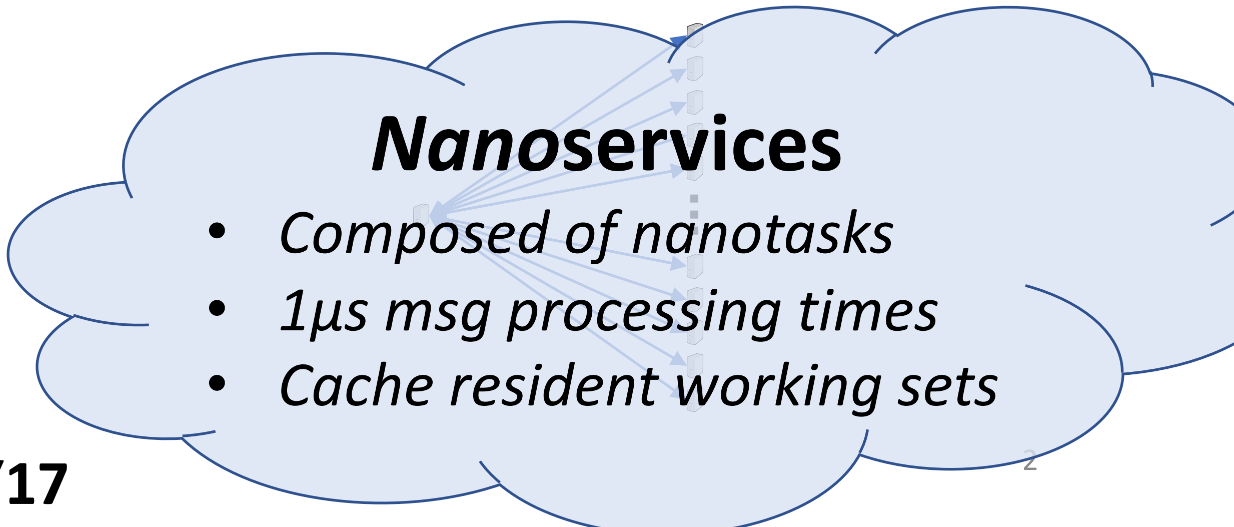


Video Encoding





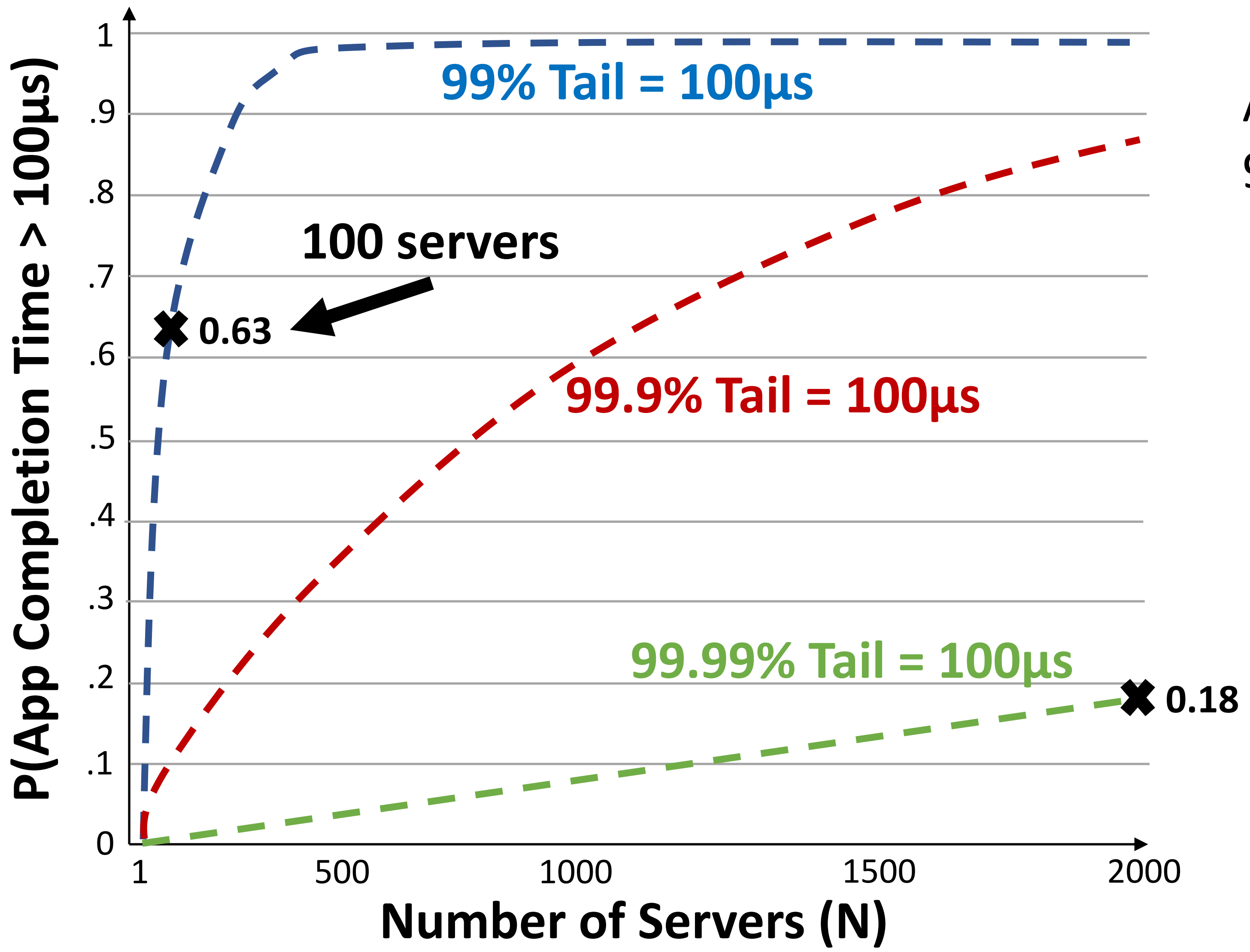
ExCamera NSDI '17



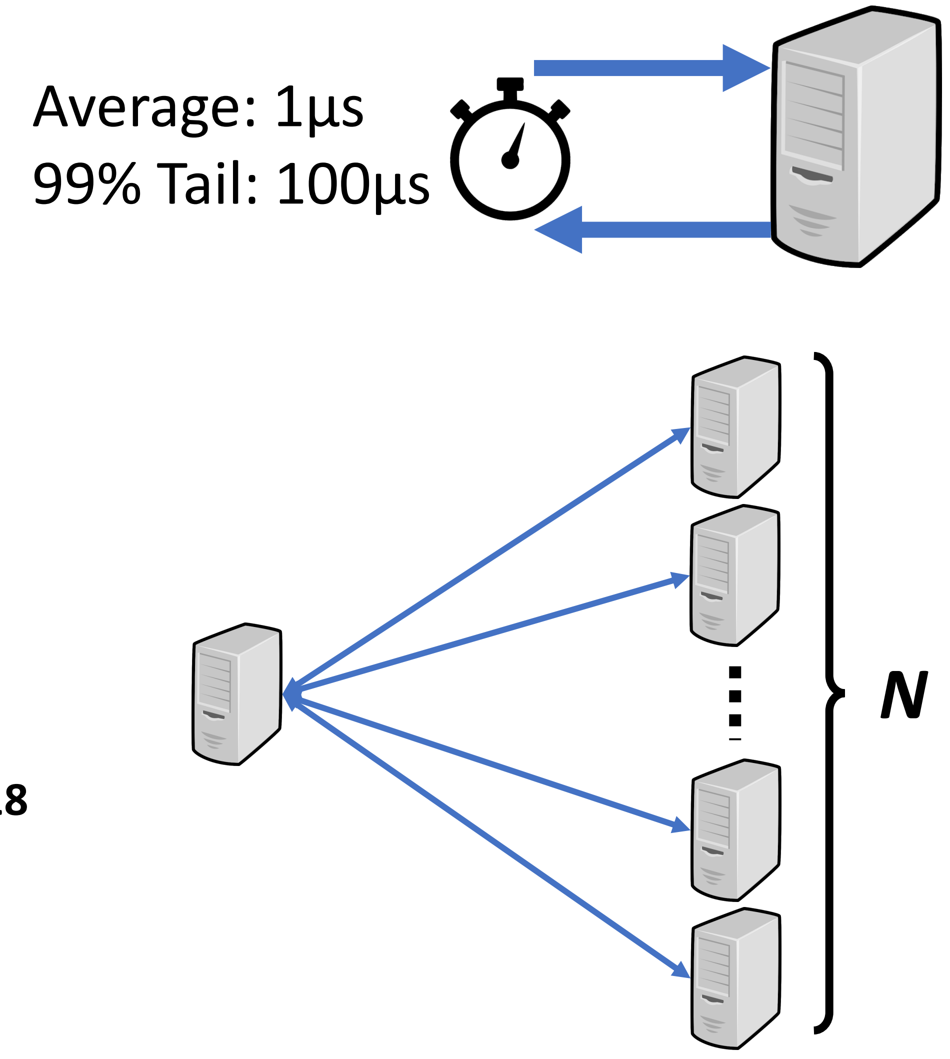
Nanoservices

- *Composed of nanotasks*
- *1µs msg processing times*
- *Cache resident working sets*

Tail Latency Matters!



Credit: *L. Barroso, et al. "The Datacenter As a Computer"*, Ch. 2



What causes high RPC tail latency?

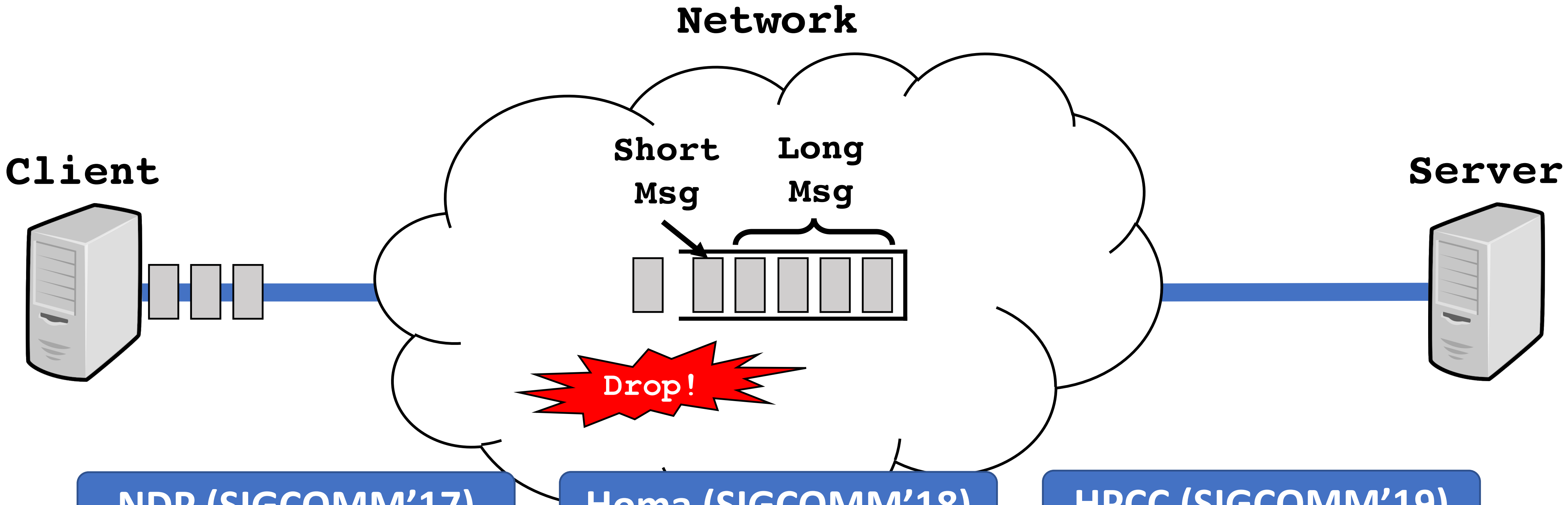
Primary Cause

Poor job scheduling access to critical shared resources:

- *Network fabric resources*
- *CPU cores*
- *Host memory bandwidth and cache space*

Problems with Modern CPU/NIC/OS Designs

1. Suboptimal congestion control for scheduling of the network fabric



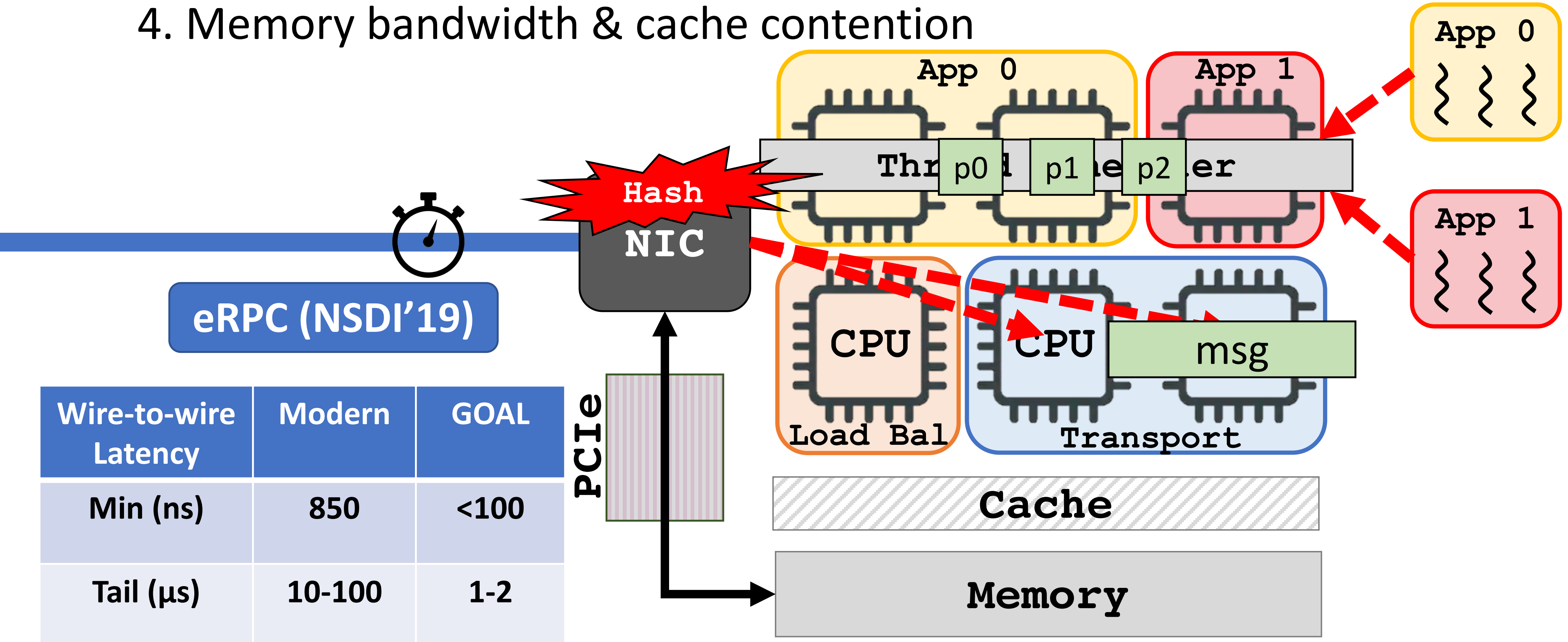
NDP (SIGCOMM'17)

Homa (SIGCOMM'18)

HPCC (SIGCOMM'19)

Problems with Modern CPU/NIC/OS Designs

1. Suboptimal congestion control for scheduling of the network fabric
2. Inefficient load balancing across cores
3. Inefficient thread scheduling on each core
4. Memory bandwidth & cache contention

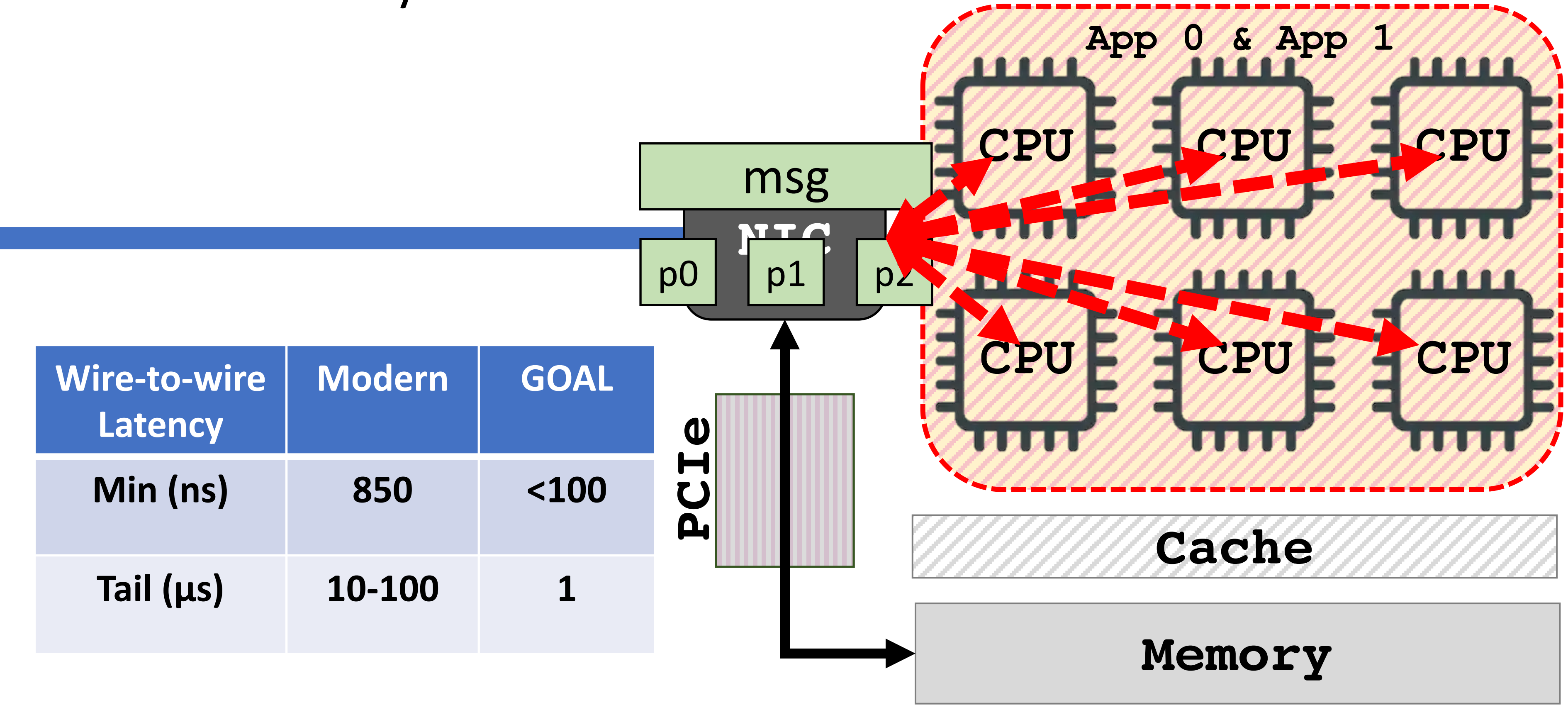


eRPC (NSDI'19)

Wire-to-wire Latency	Modern	GOAL
Min (ns)	850	<100
Tail (µs)	10-100	1-2

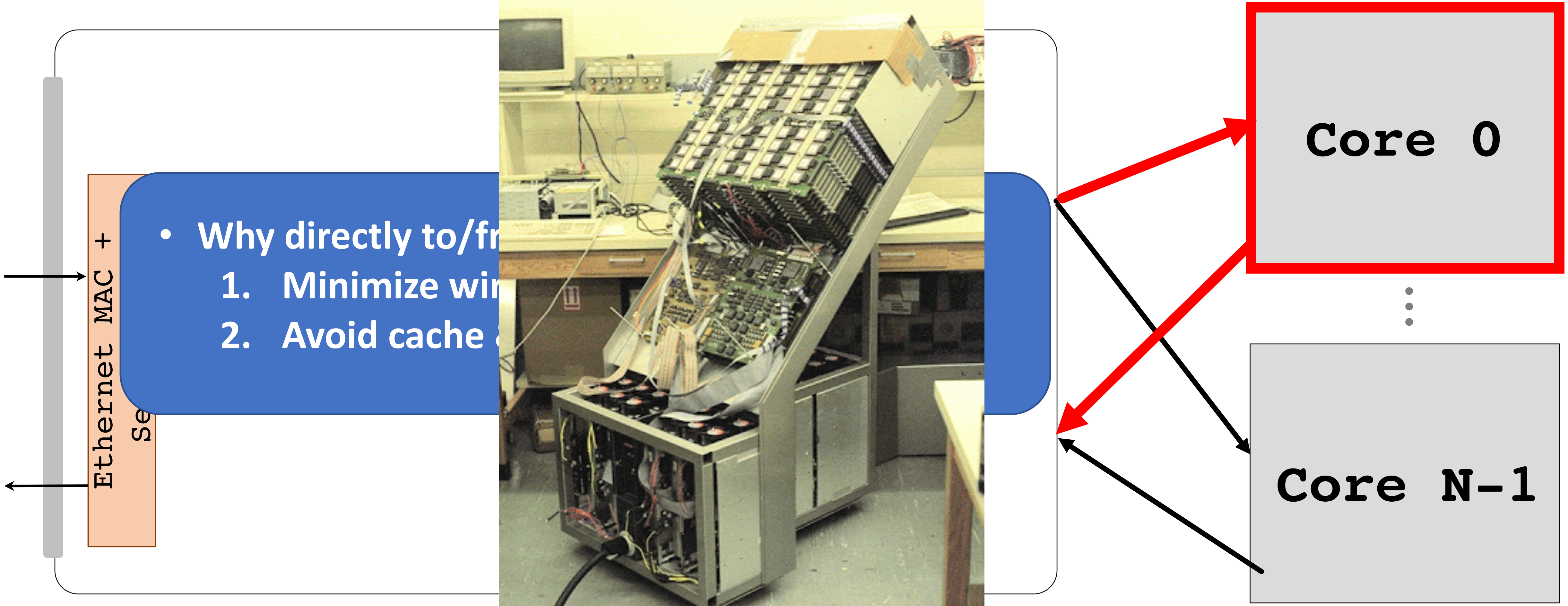
Problems with Modern CPU/NIC/OS Designs

1. Suboptimal congestion control for scheduling of the network fabric
2. Inefficient load balancing across cores
3. Inefficient thread scheduling on each core
4. Memory bandwidth & cache contention

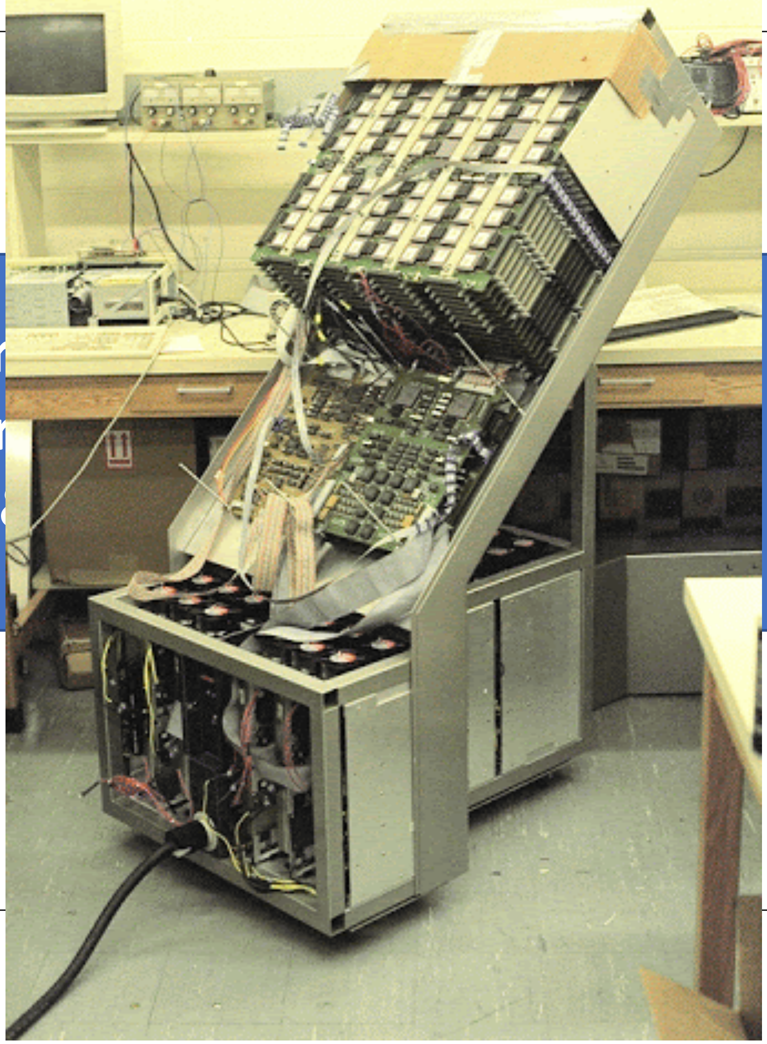


Wire-to-wire Latency	Modern	GOAL
Min (ns)	850	<100
Tail (µs)	10-100	1

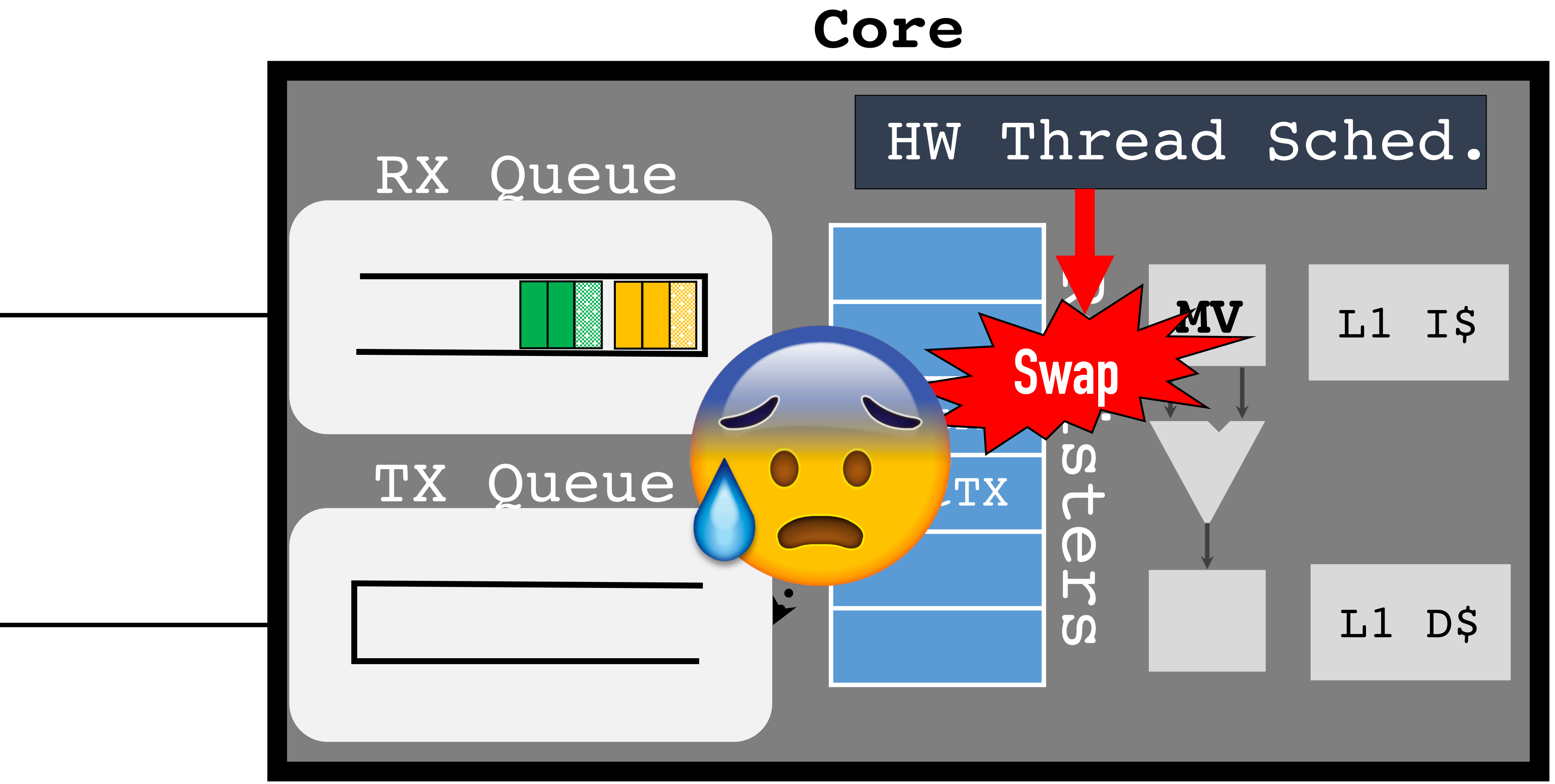
The nanoPU Design



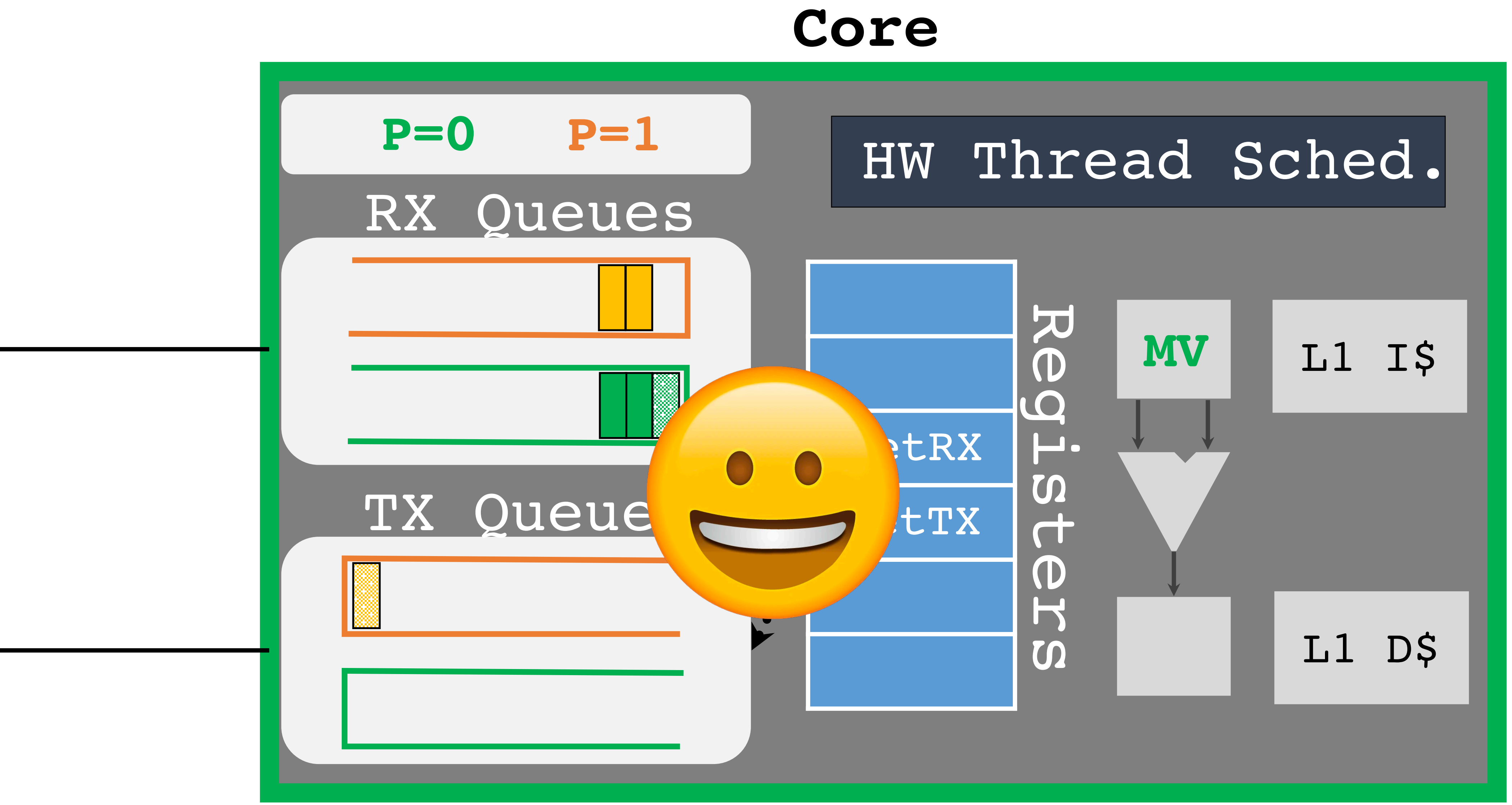
- Why directly to/fr
  1. Minimize wir
  2. Avoid cache


J-Machine (1989)

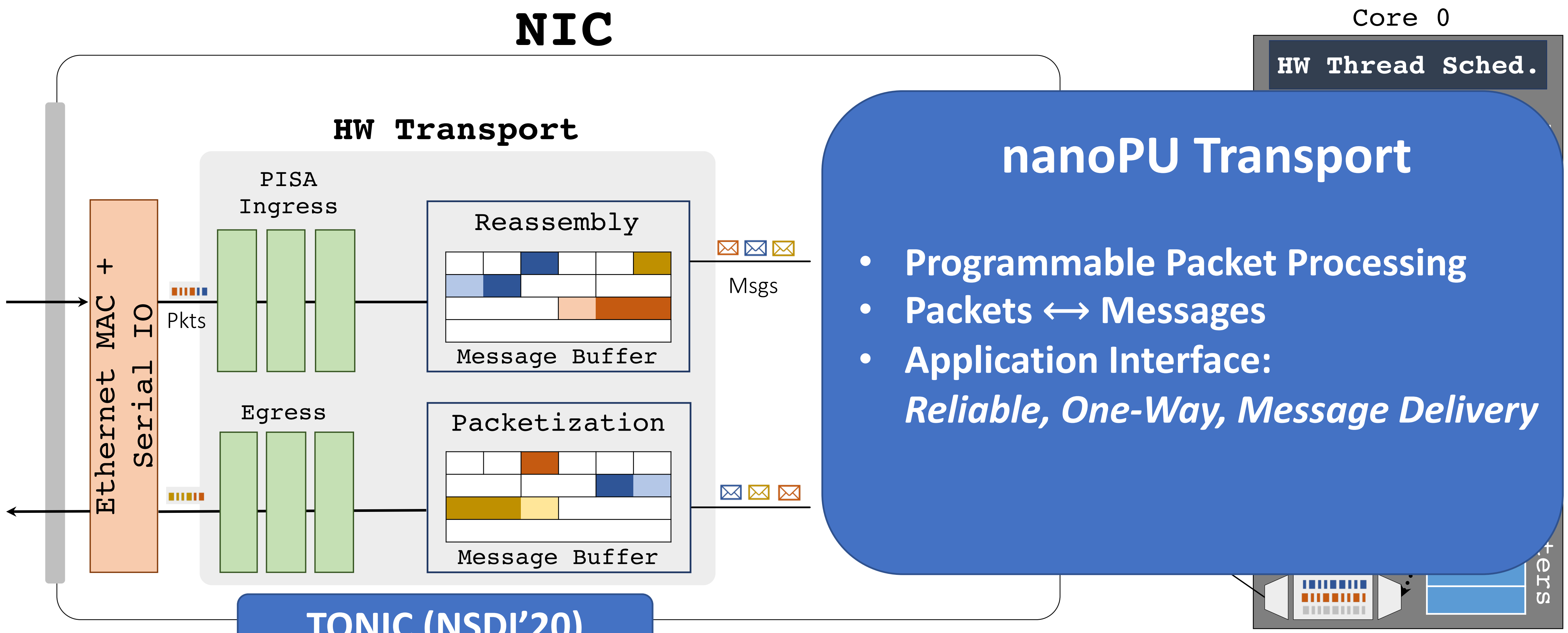
nanoPU’s Register File Network Interface



nanoPU's Register File Network Interface



nanoPU Transport

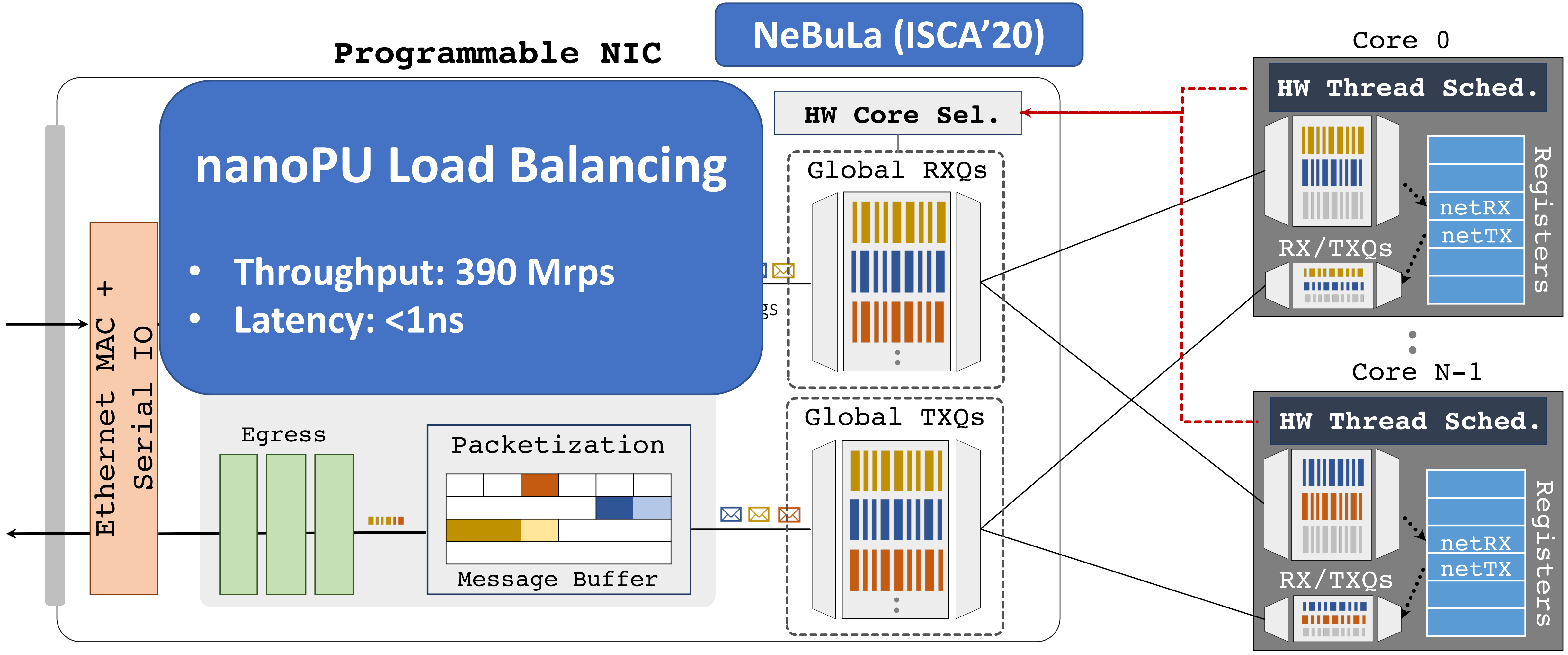


TONIC (NSDI'20)

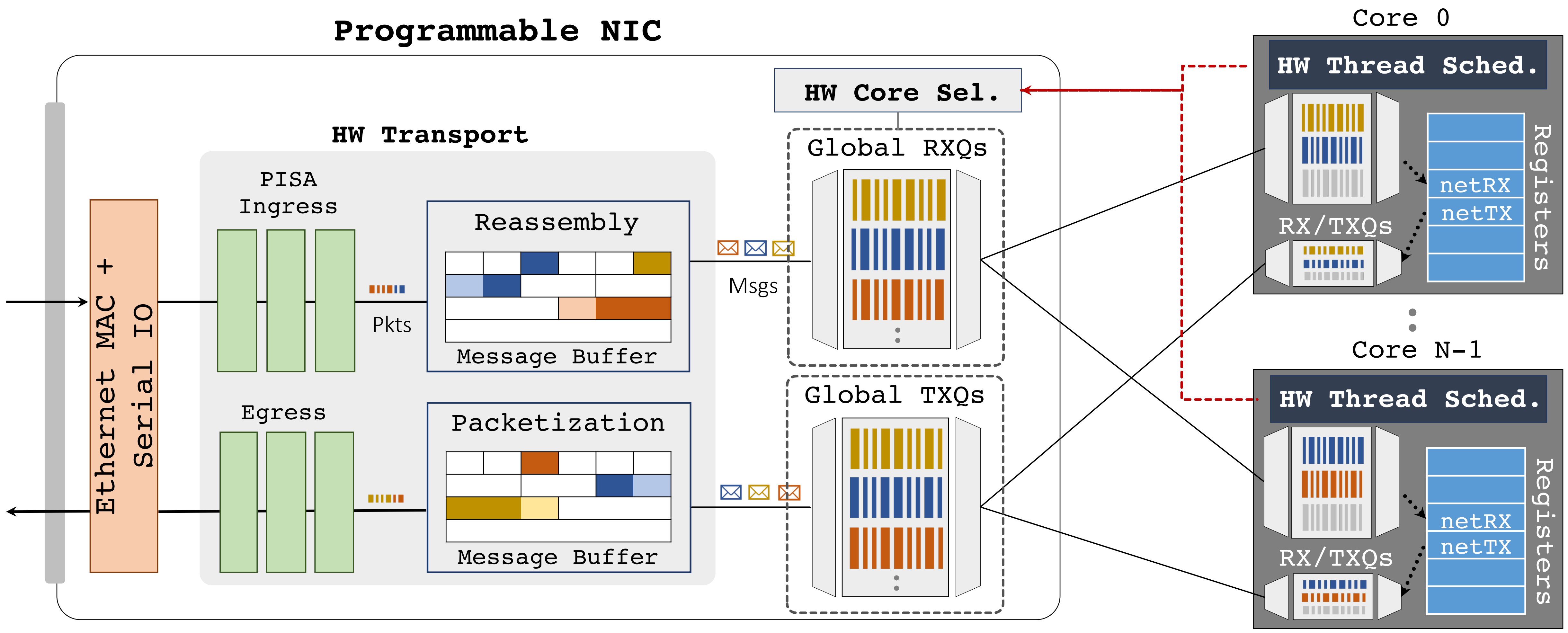
nanoPU Transport

- Programmable Packet Processing
- Packets ⟷ Messages
- Application Interface:
  Reliable, One-Way, Message Delivery

nanoPU Core Load Balancing

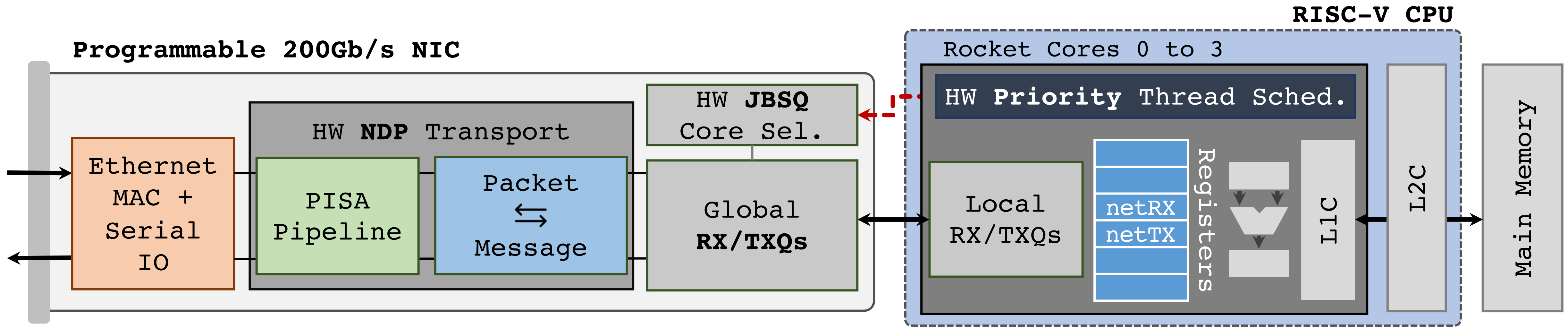


The nanoPU Design



nanoPU Prototype

- Quad-core nanoPU based on the 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



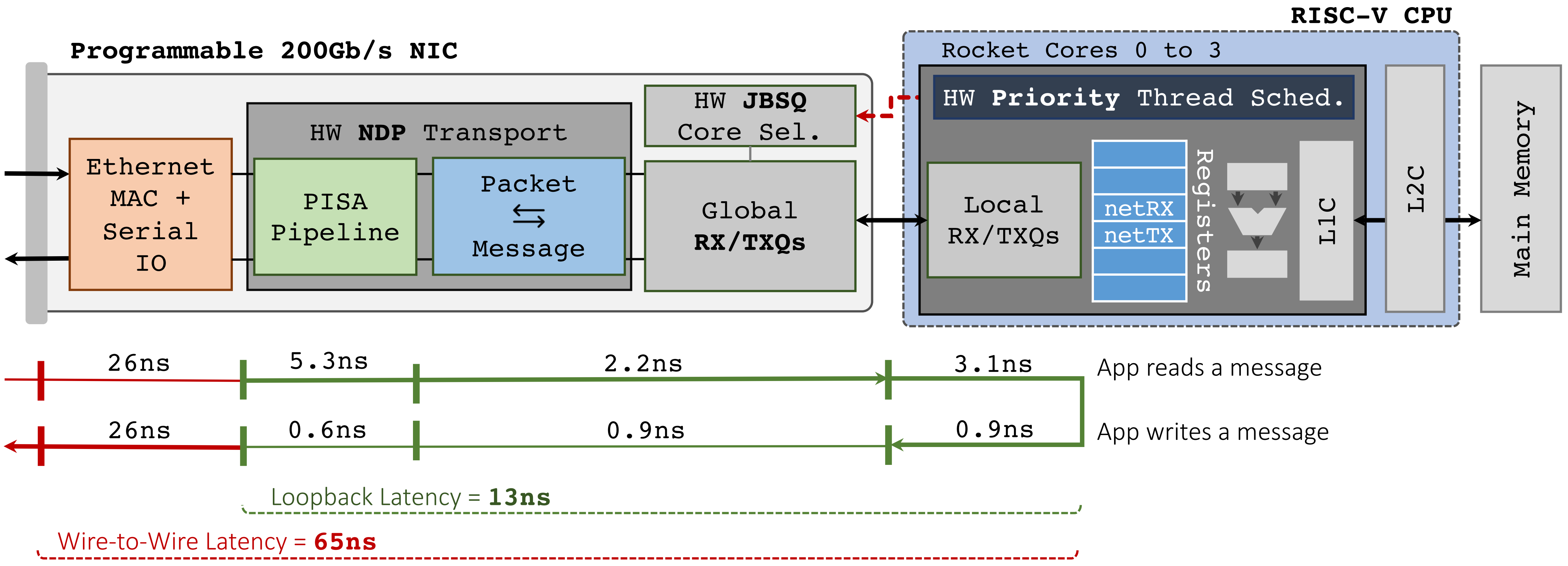
Evaluation Methodology

- nanoPU prototype running on AWS FPGAs
- Large-scale (hundreds of cores), cycle-accurate simulations with Firesim
- FPGA clock rate is 90MHz, simulated target clock rate is 3.2GHz

+

ISCA’18

Microbenchmarks

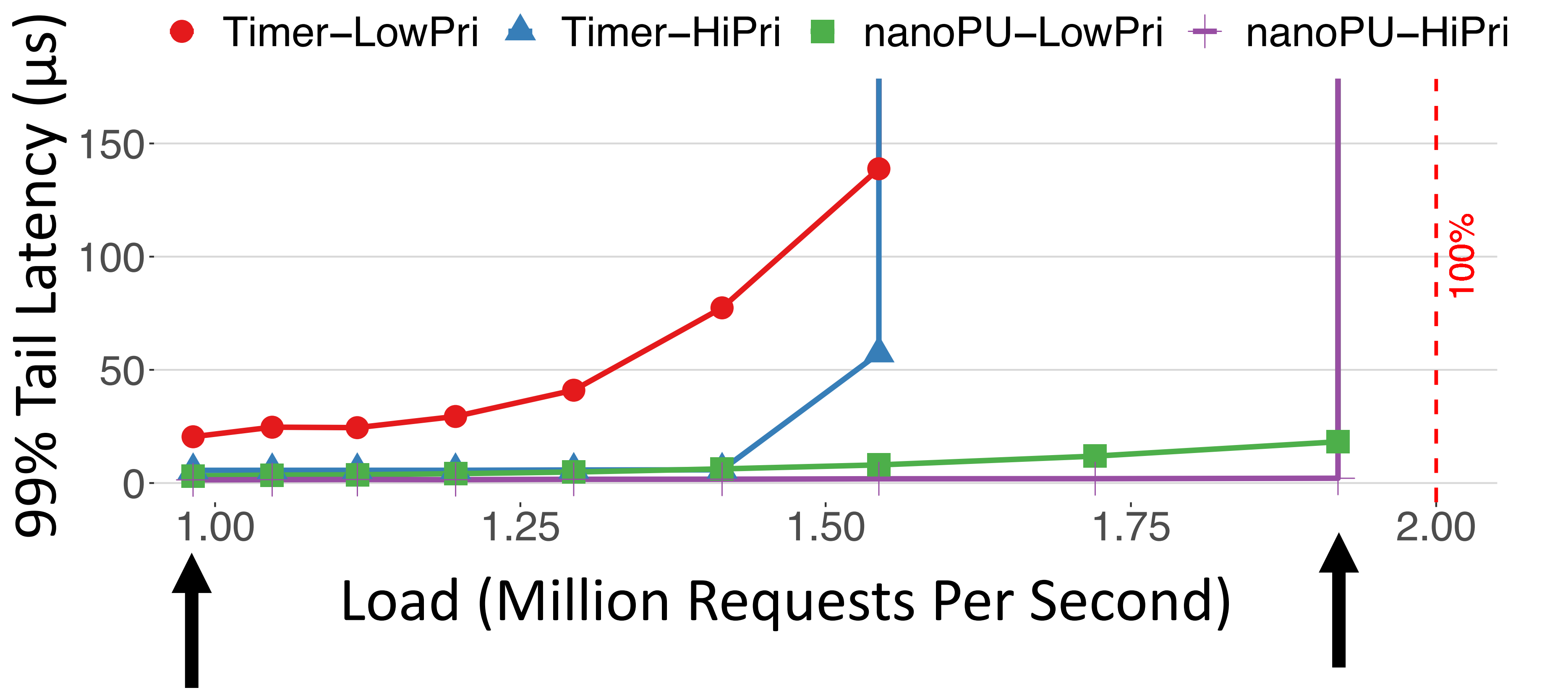


	Wire-to-Wire Latency (ns)
nanoPU	**65**
eRPC	850

Thread Scheduling Evaluation

- Single core running two threads: high priority and low priority
- 500ns request processing time
- 10K requests per thread
- nanoPU HW thread scheduler vs. 5us timer-driven thread scheduler

Thread Scheduling Evaluation



50% load

Hi Priority – 4x lower (1.4µs)

Low Priority – 6.5x lower (3.2µs)

96% load

Hi Priority – 2.1µs

Low Priority – 18.2 µs

Additional Evaluations

✓ **Core Load Balancing**

✓ **NDP Transport**

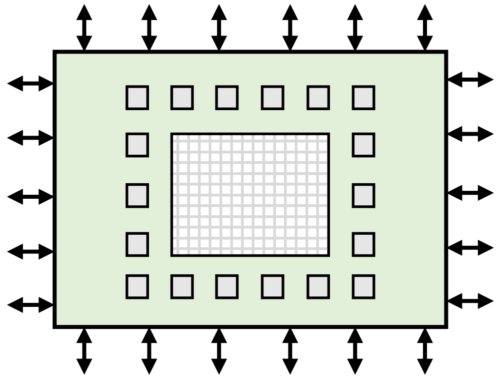
Real Applications Running on the nanoPU

- MICA Key Value Store
- Chain Replication
- Raft Consensus

nanoPU Deployment Possibilities

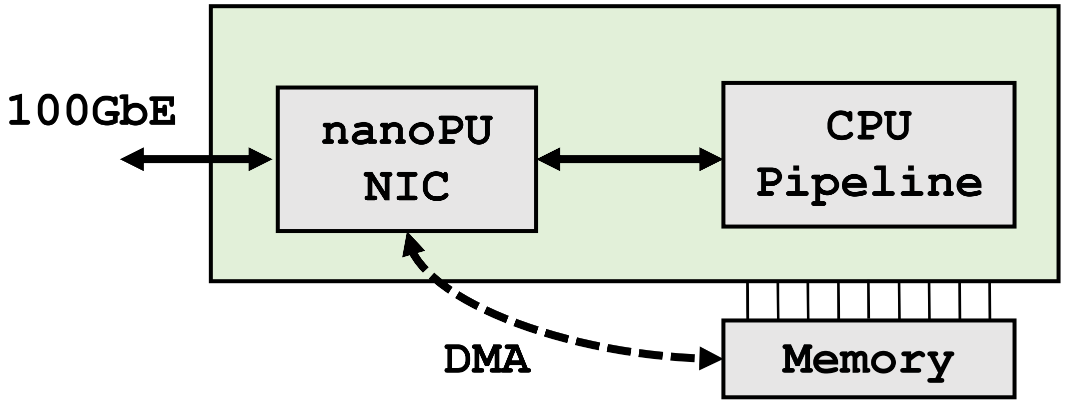
1 nanoPU Cluster

2 nanoPU High IO Capacity Package

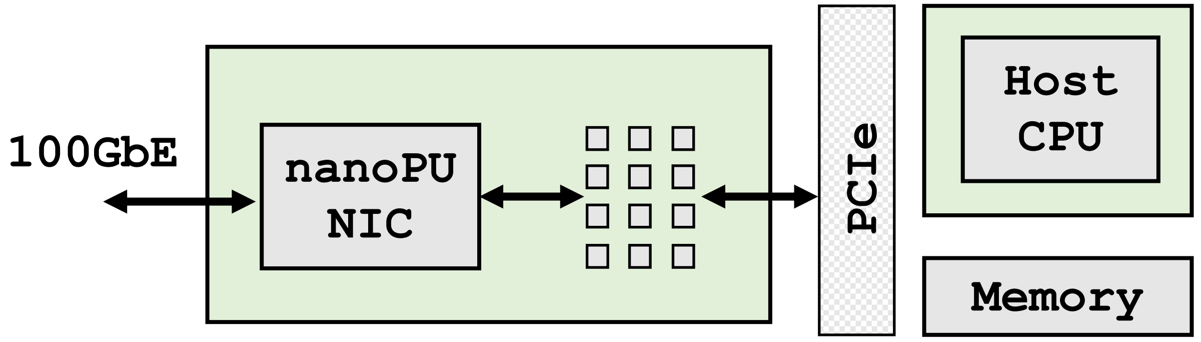


- 512 cores
- 256 x 100Gb/s

3 Modified Conventional CPU



4 nanoPU SmartNIC



nanoPU Conclusions

Key Takeaway:

To truly minimize average and tail RPC latency:

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

Challenges:

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

5G Connected Edge Cloud for Industry 4.0 Transformation

ONF SPOTLIGHT

Thank You

sibanez@stanford.edu