ETH zürich DINFK Systems@ETH zürich ENZIAN

Sponsored by: Google

The NIC should be part of the OS.

Pengcheng Xu, Timothy Roscoe

Tightly **couple NIC and OS**
in *scheduling* and *flow steering* in order to

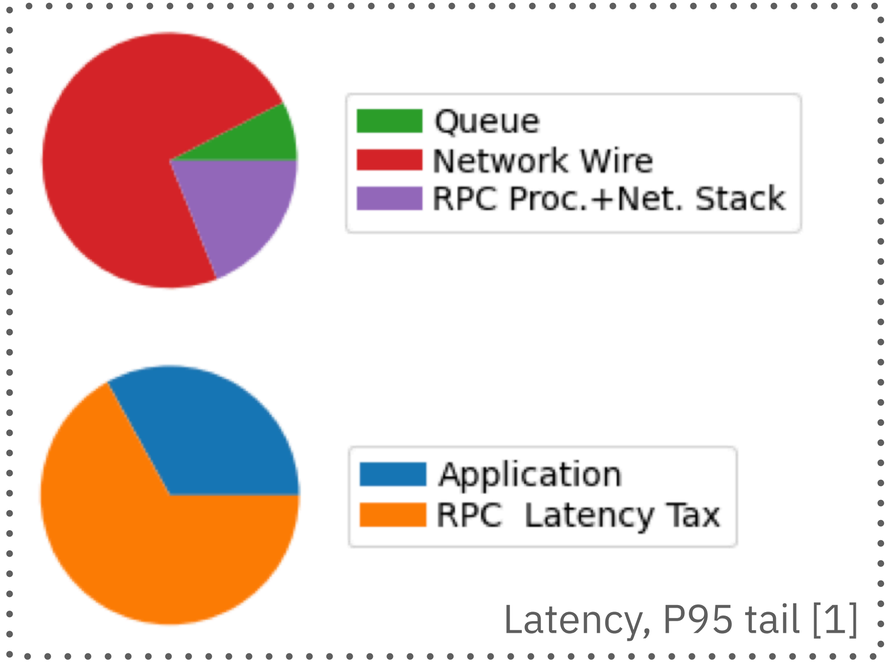
Dispatch RPC* in ***3 instructions***

** Packet at NIC → CPU executing handler in user space*

Low-latency RPCs

What we focus on

- Micro-services, serverless
  - *Up to 25% of Google's RPC calls are ~300 us!* [1]
- **RPC tax** is significant
- Latency is a *proxy* of **efficiency**
- What we focus less on: bulk transfers
  - *Throughput-oriented techniques work fine*



Latency, P95 tail [1]

[1] K. Seemakhput, et. al., „A Cloud-Scale Characterization of Remote Procedure Calls“, SOSP’23.

What's new?

- **Smart** NICs: *CPU cores, FPGA, ASICs*
- Server CPUs with **100+ cores**
  - *People dedicate cores to networking workloads*
- **Cache-coherent** interconnects
  - *Fast, low-overhead message passing between peripheral and CPU*

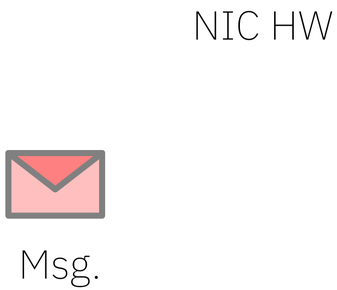
User-level schedulers

Demikernel, Shinjuku, Shenango, …

- Similar packet delivery process: **decouple OS and NIC**
  - *NIC gets buffer from* ***receive descriptor***
  - ***DMA*** *into DRAM/LLC*
  - ***Notify CPU****: poll, mwait, IRQ*
- CPU then tries to **rebalance workloads**
  - *As fast as possible, but* ***out-of-band****!*

Receive fast path

- Packet arrives

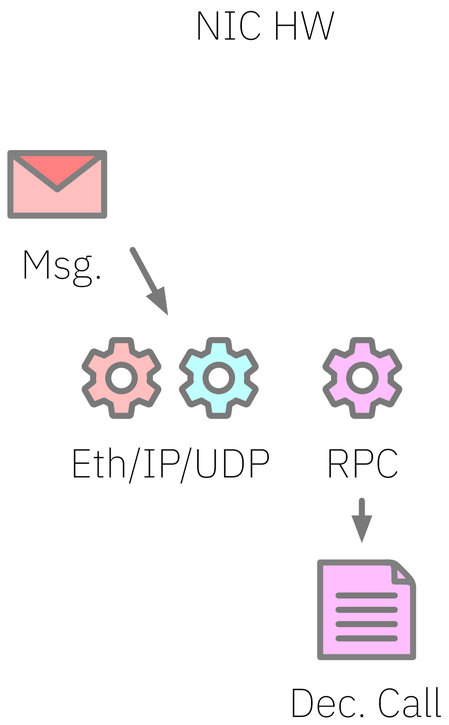


Receive fast path

- Packet arrives
- Protocol processing
  - *Up to transport layer*
- Full RPC decode
  - *Service ID, unmarshalled arguments*
  - Look up *function pointer* in table

TCP offload, RDMA

ProtoAcc, Cerebros, …



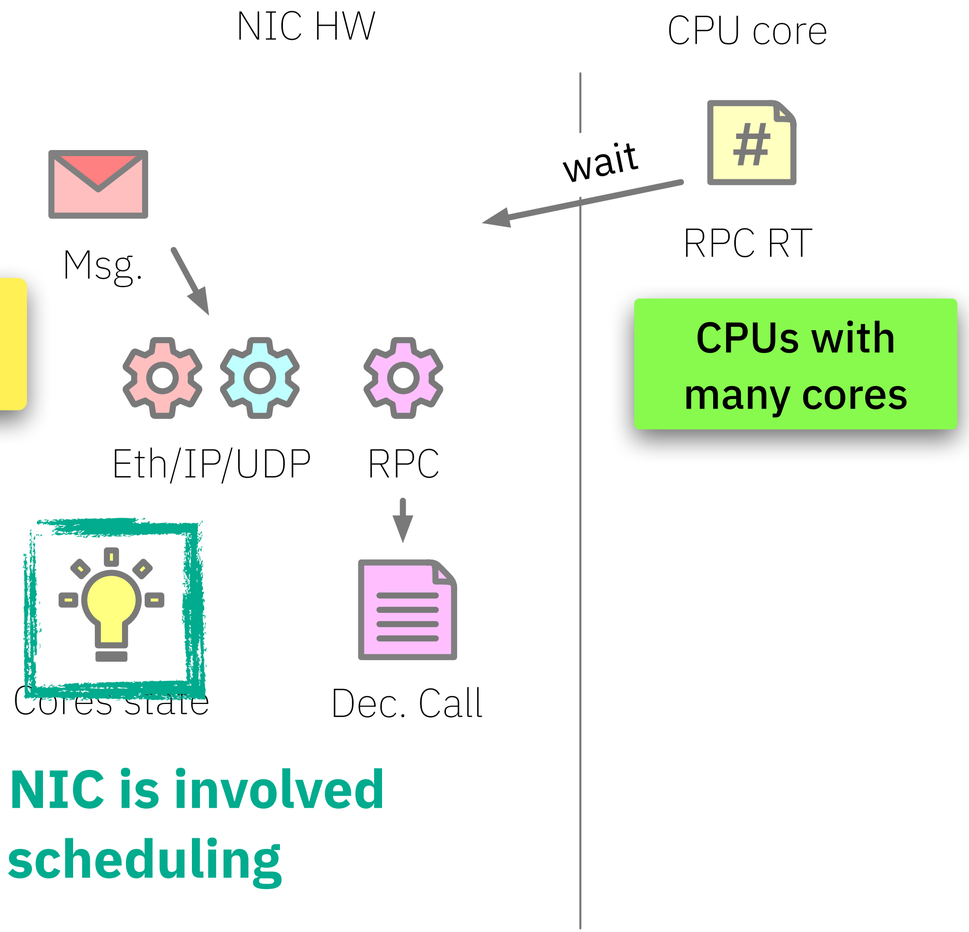
Smart NICs

Receive fast path

- Packet arrives
- Protocol processing, full RPC decode
- NIC knows **state of cores**:
  - *Which services are on which cores*
  - *Which cores are waiting for a request*

kernel-bypass, user-level schedulers

Not used before!

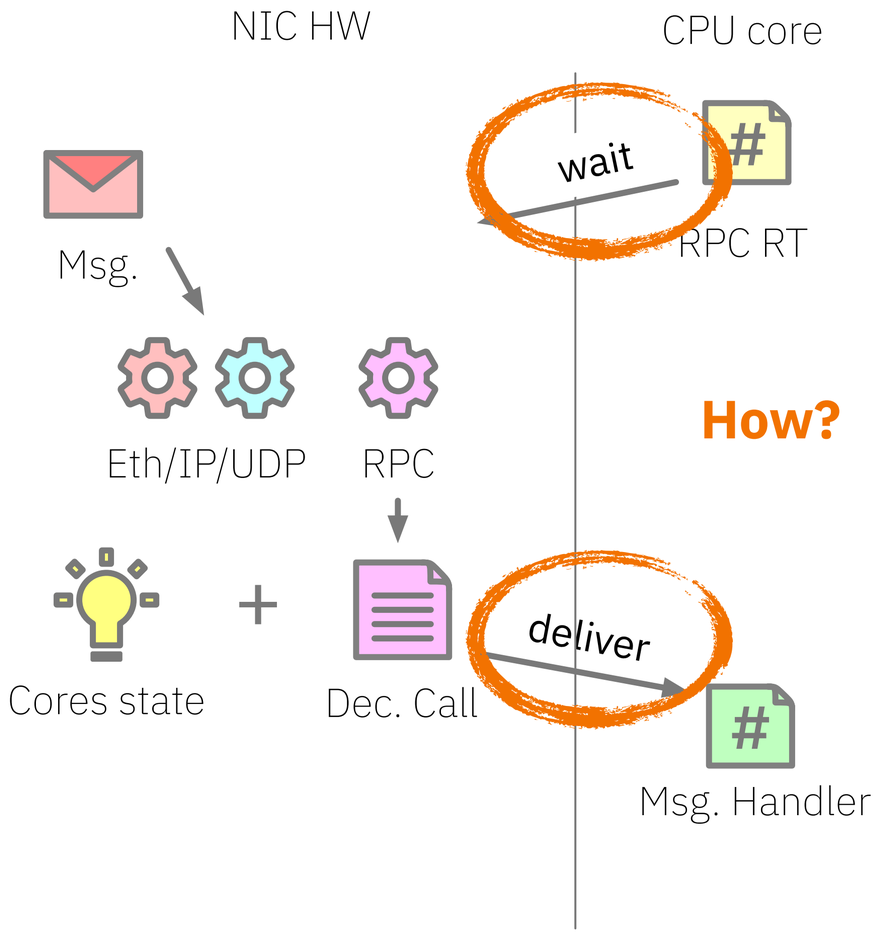


CPUs with many cores

How? NIC is involved in scheduling

Receive fast path

- Packet arrives
- Protocol processing, full RPC decode
- NIC chooses **a waiting core**
- NIC delivers request to registers, core calls handler
  - Function pointer + arguments: *just a simplest function call!*



PIO over cache coherence

Pass a message from NIC to CPU

- **Message-level access** to cache-coherence protocol
- **One blocking read** on CPU
  - *Request in register & L1 cache!*
  - *Result back in one write*
- Closely **coupling** CPU and NIC for request delivery

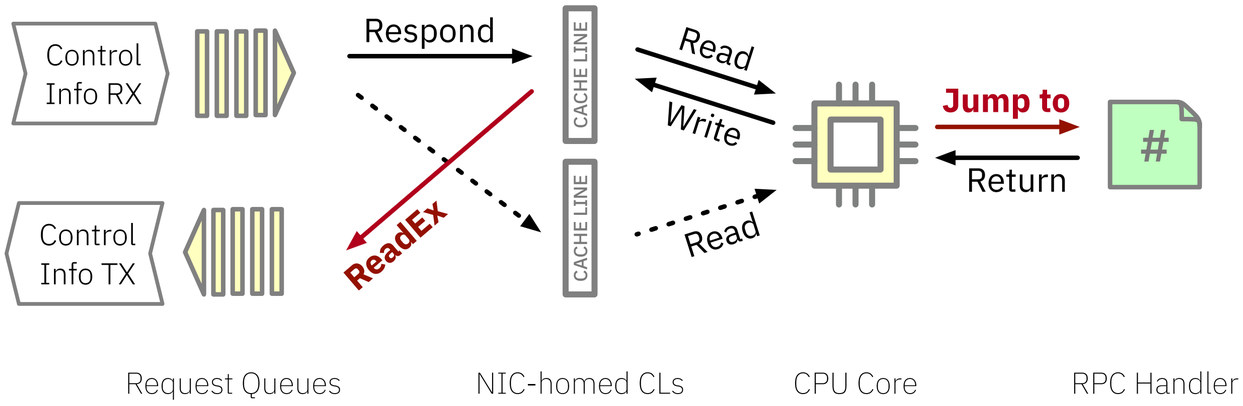
Rethinking Programmed I/O for Fast Devices, Cheap Cores, and Coherent Interconnects

Anastasiia Ruzhanskaia
Systems Group, D-INFK, ETH Zürich
Zürich, Switzerland

Pengcheng Xu
Systems Group, D-INFK, ETH Zürich
Zürich, Switzerland

David Cock
Systems Group, D-INFK, ETH Zürich
Zürich, Switzerland

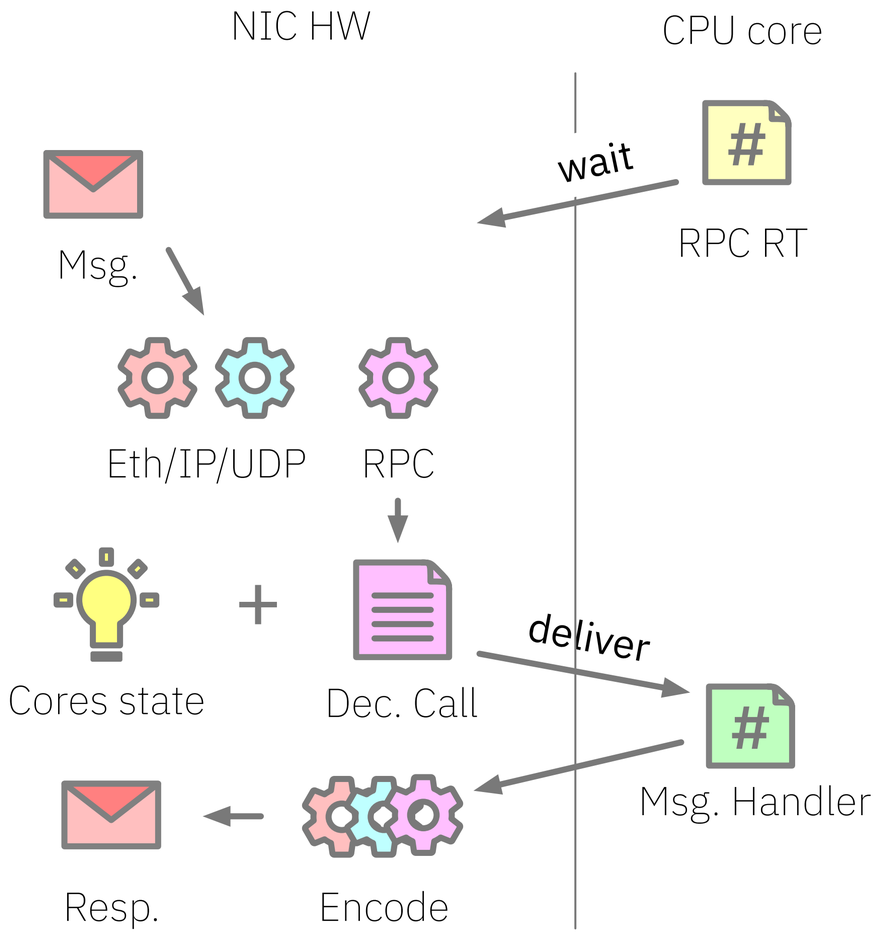
Timothy Roscoe
Systems Group, D-INFK, ETH Zürich
Zürich, Switzerland



cf. DMA: no descriptors / DRAM access / polling
cf. nanoPU: no architectural changes!

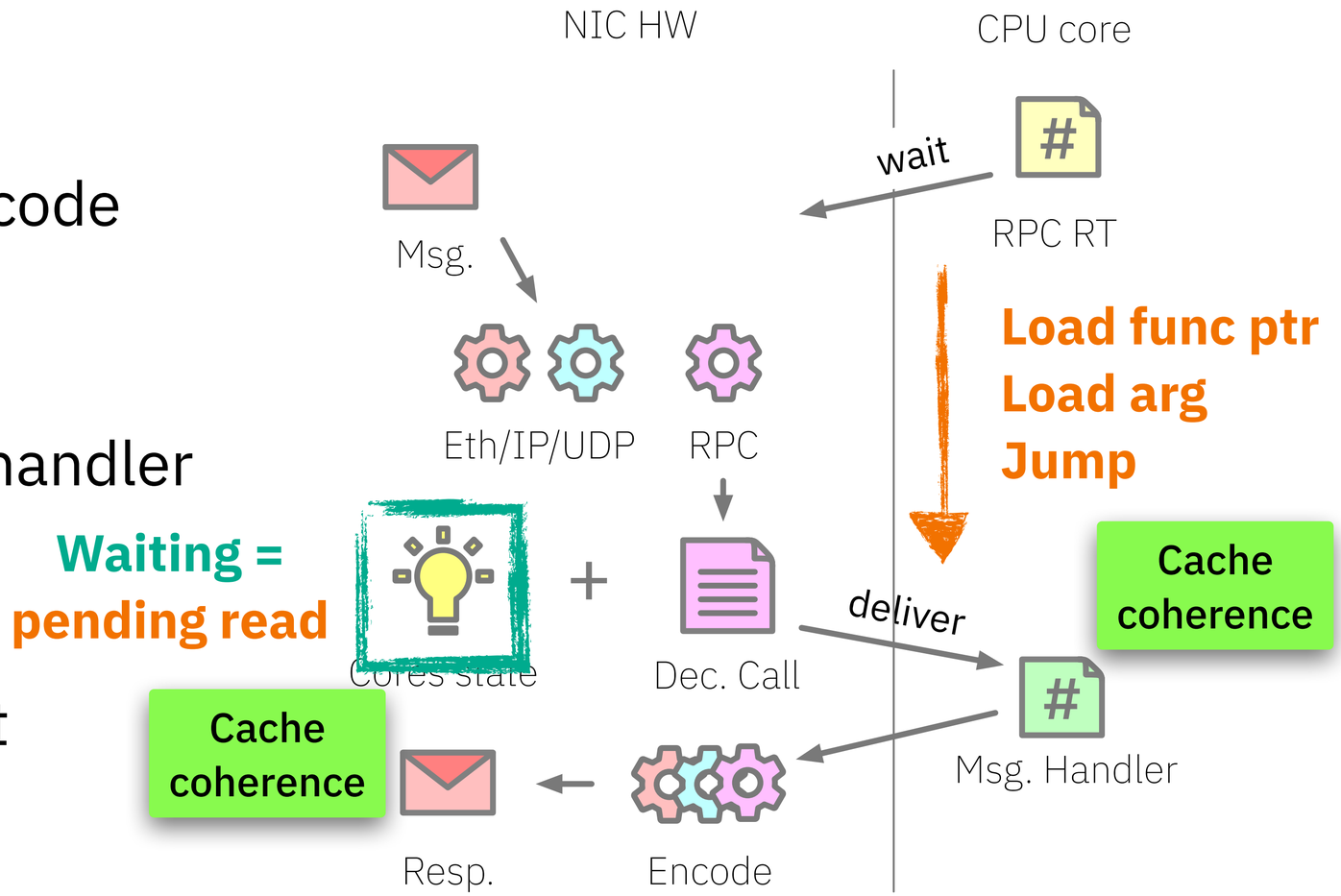
Receive fast path

- Packet arrives
- Protocol processing, full RPC decode
- NIC chooses **a waiting core**
- NIC delivers request, core calls handler
- Core passes call result to NIC
- NIC sends back response packet



Receive fast path

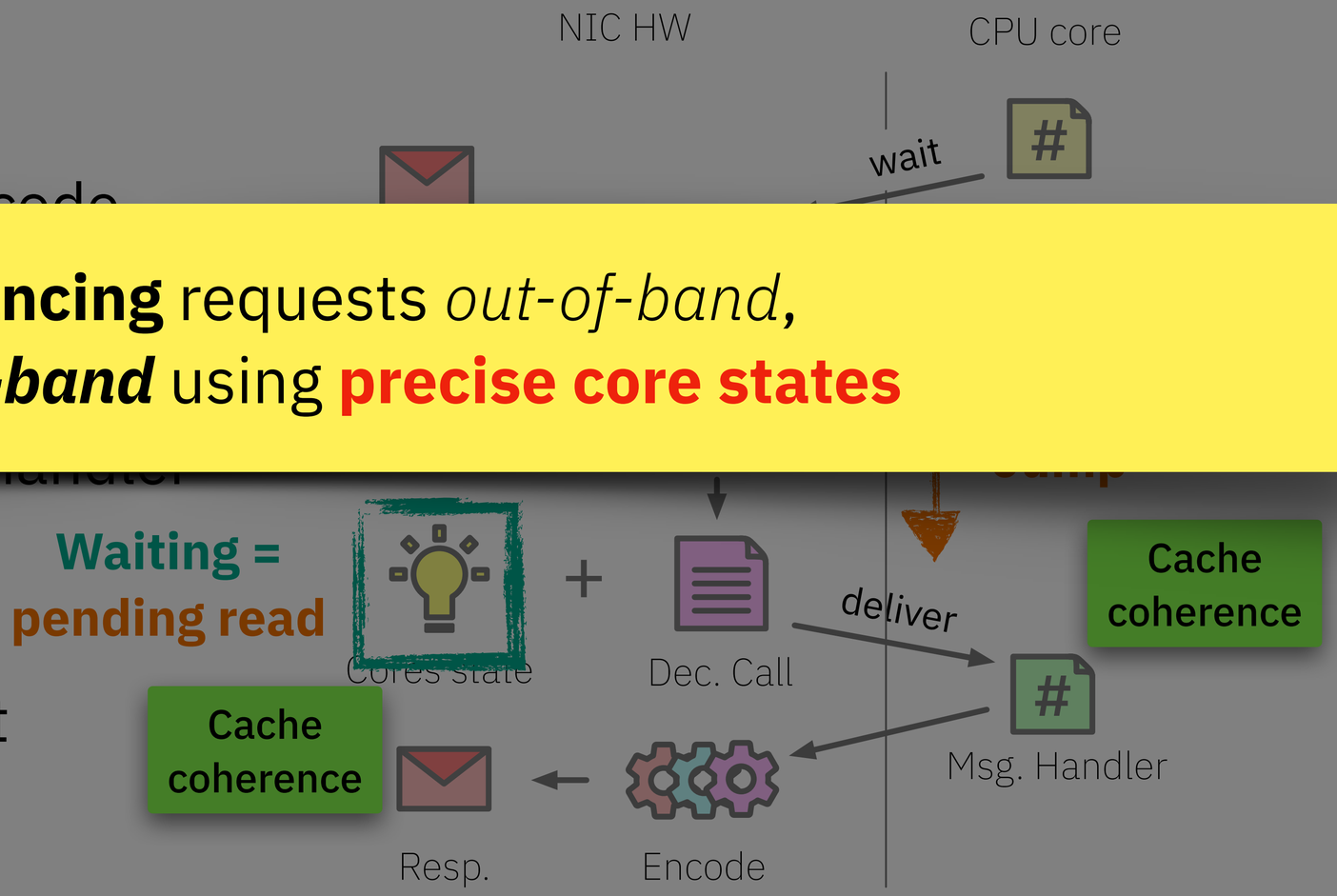
- Packet arrives
- Protocol processing, full RPC decode
- NIC chooses **a waiting core**
- NIC delivers request, core calls handler
- Core passes call result to NIC
- NIC sends back response packet



Receive fast path

- Packet arrives
- Protocol processing, full RPC decode
- NIC delivers request, core calls handler
- Core passes call result to NIC
- NIC sends back response packet

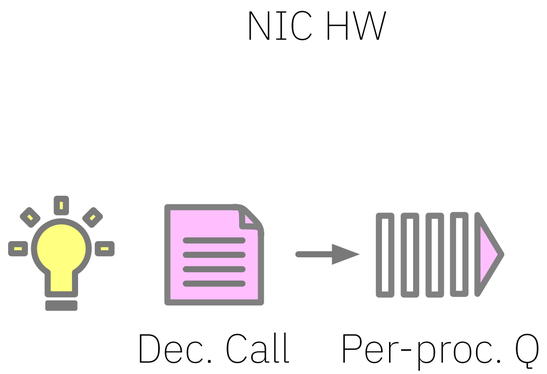
Instead of **rebalancing** requests *out-of-band*,
we steer requests ***in-band*** using **precise core states**



What if no cores are waiting?

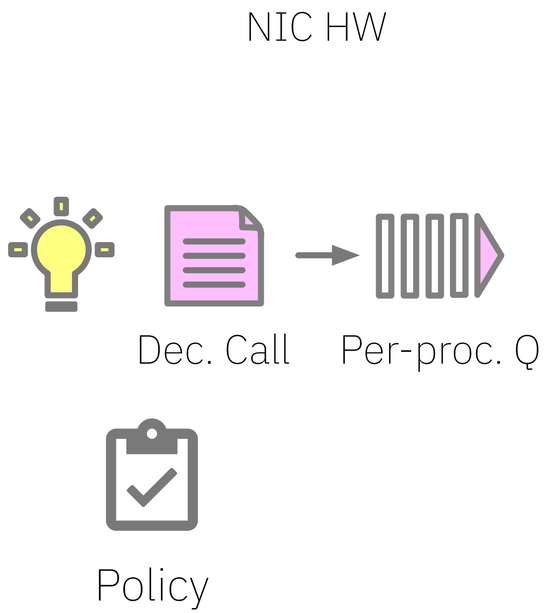
Request needs a CPU core

- NIC decoded request—
  - *Nobody waiting for this service!*
  - Enqueue in **per-proc** queue on NIC



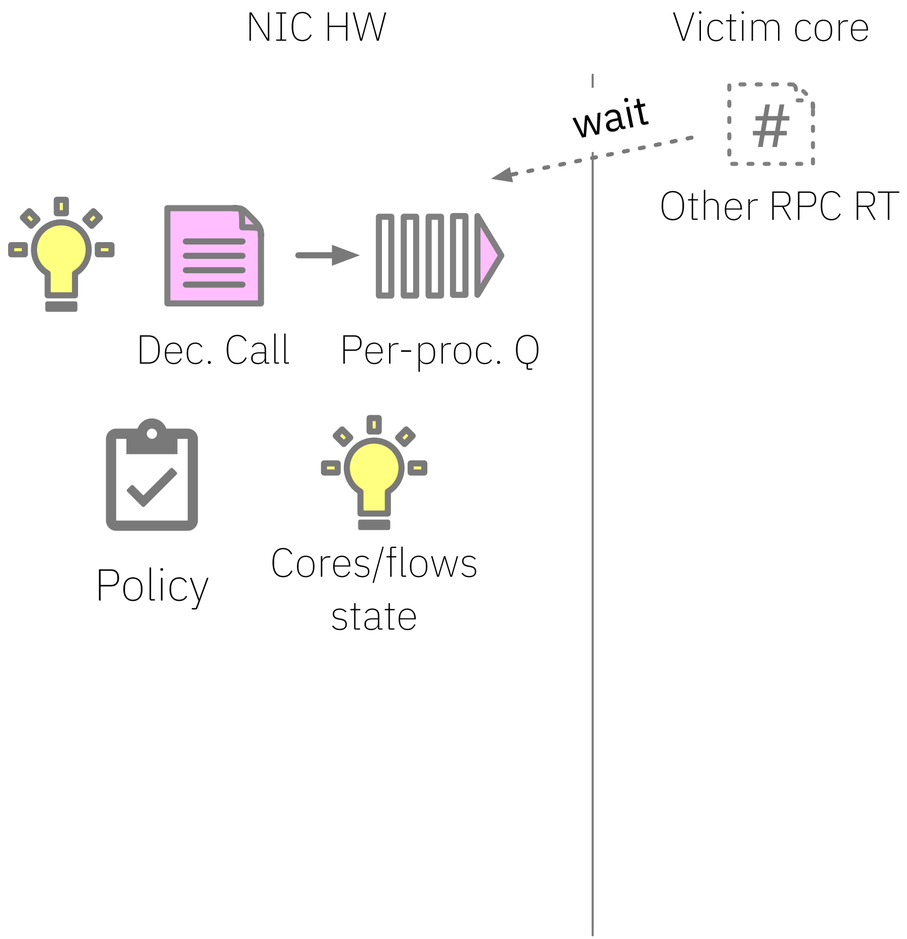
Request needs a CPU core

- NIC enqueues decoded request
- At some point: need to **reassign** a core
  - *When: policy decision*



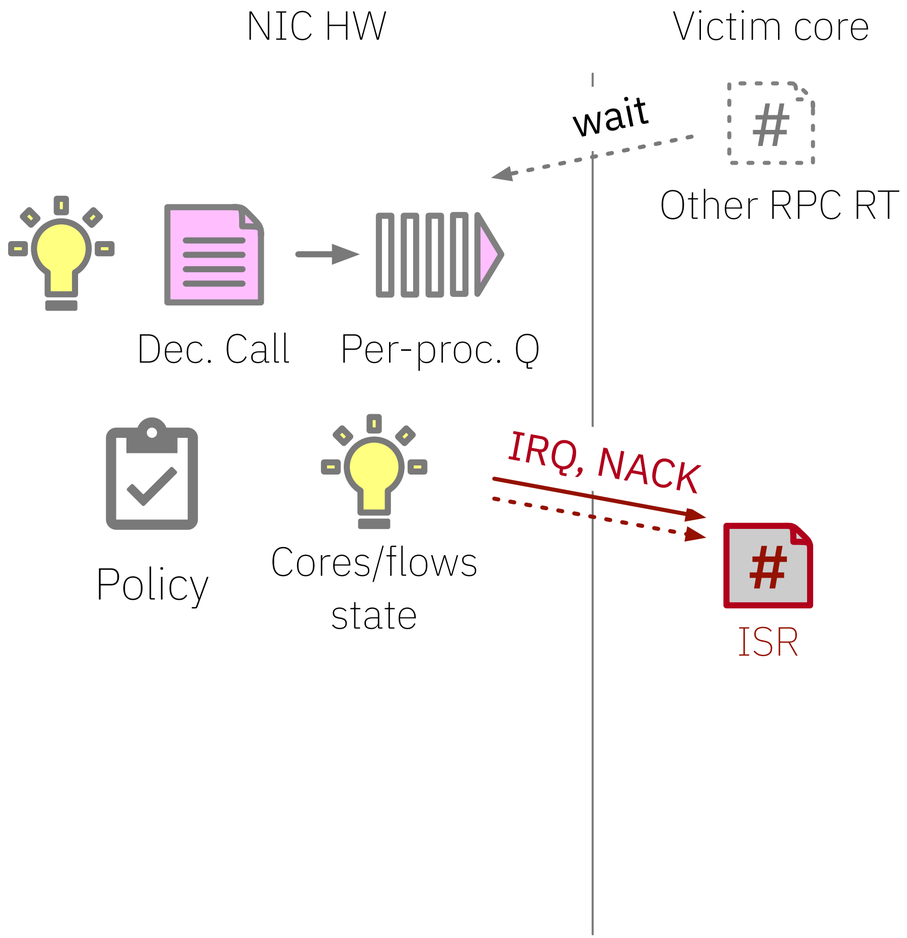
Request needs a CPU core

- NIC enqueues decoded request
- Policy warrants a reschedule
- Select core to **reassign** to service
  - *e.g. Core waiting for inactive service*



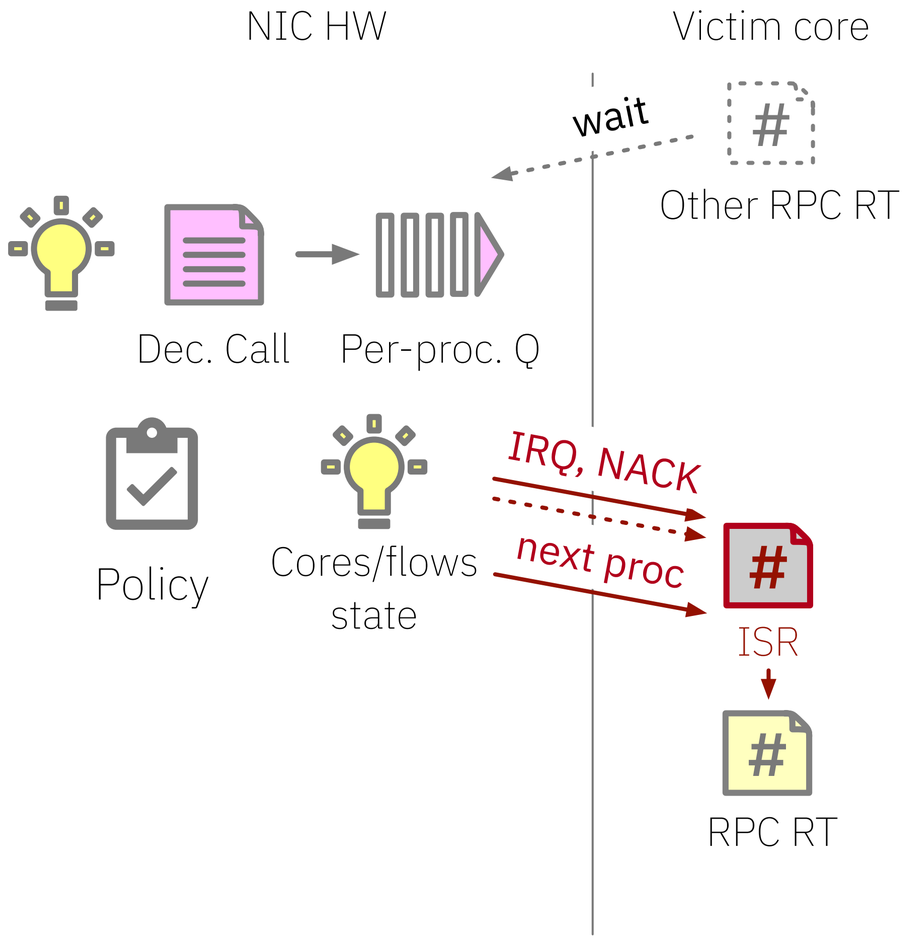
Request needs a CPU core

- NIC enqueues decoded request
- Policy warrants a reschedule
- Select core to **reassign** to service
- Send IRQ and NACK to bring into kernel
  - *Unblock core and cause interrupt at the same time*



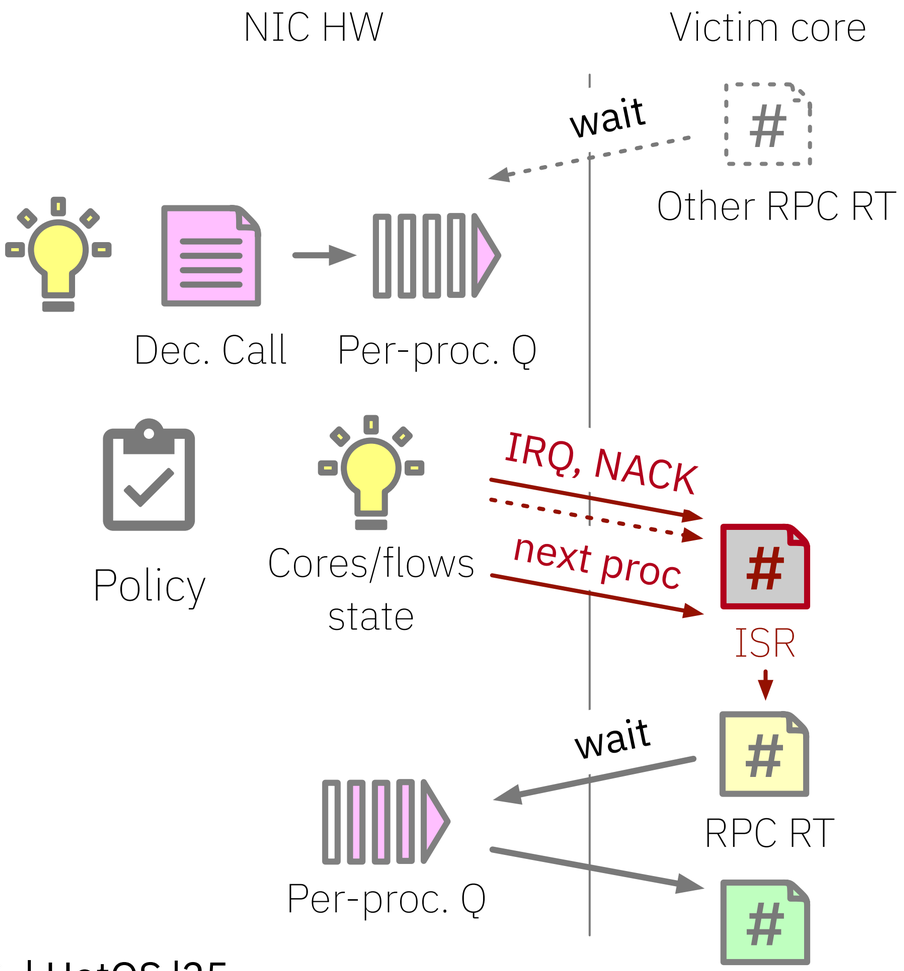
Request needs a CPU core

- NIC enqueues decoded request
- Policy warrants a reschedule
- Select core to **reassign** to service
- Send IRQ and NACK to bring into kernel
- Tell core to context switch into new proc
  - *Kernel schedules thread on core*



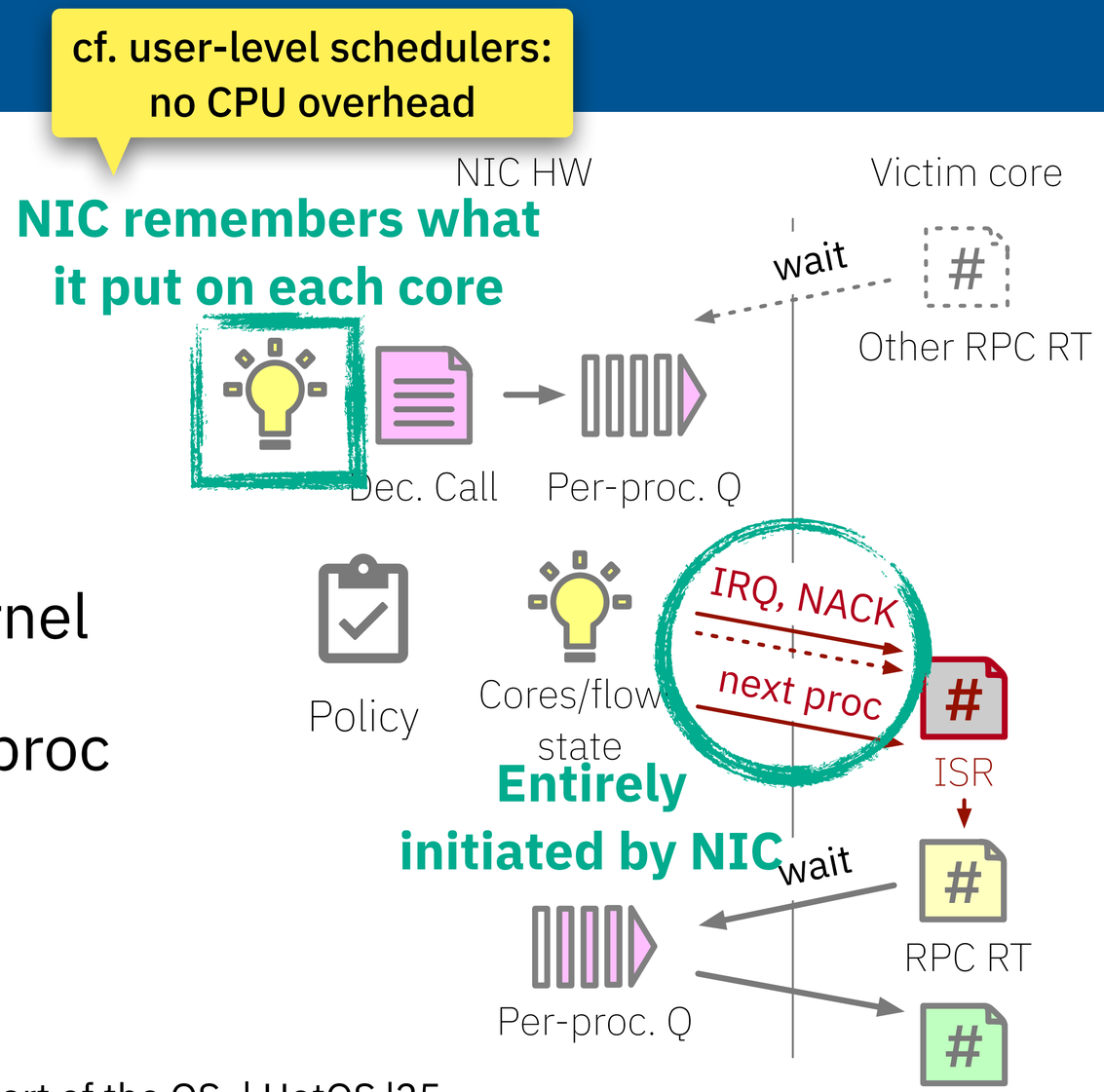
Request needs a CPU core

- NIC enqueues decoded request
- Policy warrants a reschedule
- Select core to **reassign** to service
- Send IRQ and NACK to bring into kernel
- Tell core to context switch into new proc
- Core handles queued requests



Request needs a CPU core

- NIC enqueues decoded request
- Policy warrants a reschedule
- Select core to **reassign** to service
- Send IRQ and NACK to bring into kernel
- Tell core to context switch into new proc
- Core handles queued requests

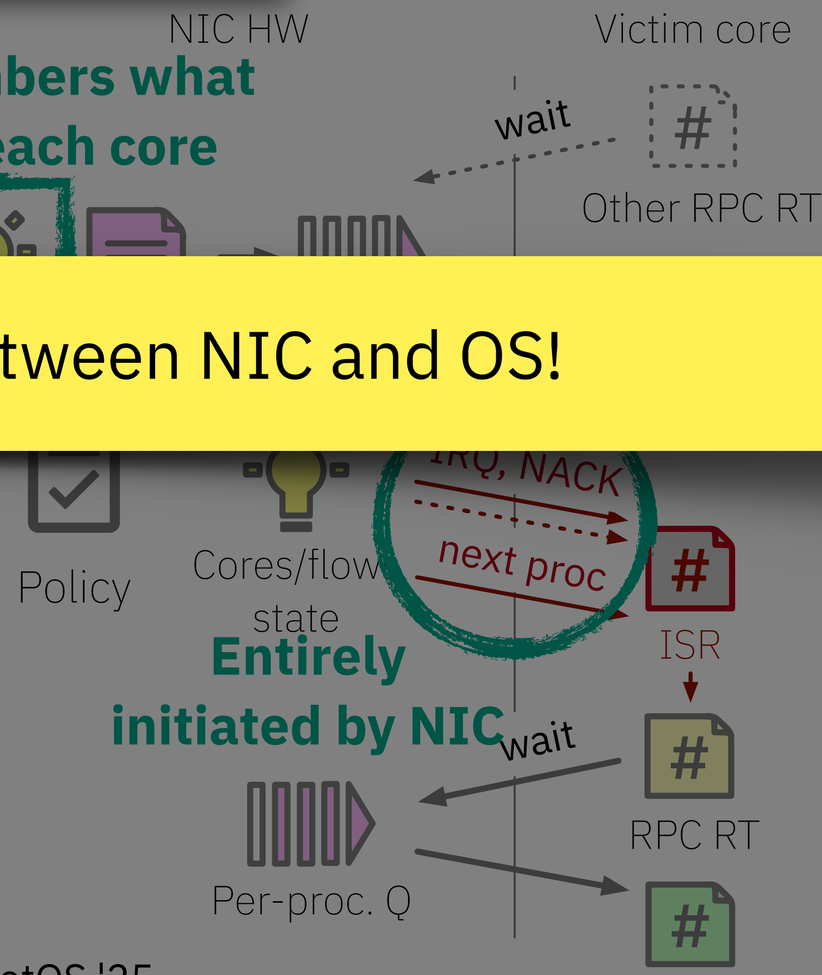


Request needs a CPU core

- NIC enqueues decoded request
- Policy warrants a reschedule
- Send IRQ and NACK to bring into kernel
- Tell core to context switch into new proc
- Core handles queued requests

cf. user-level schedulers: no CPU overhead

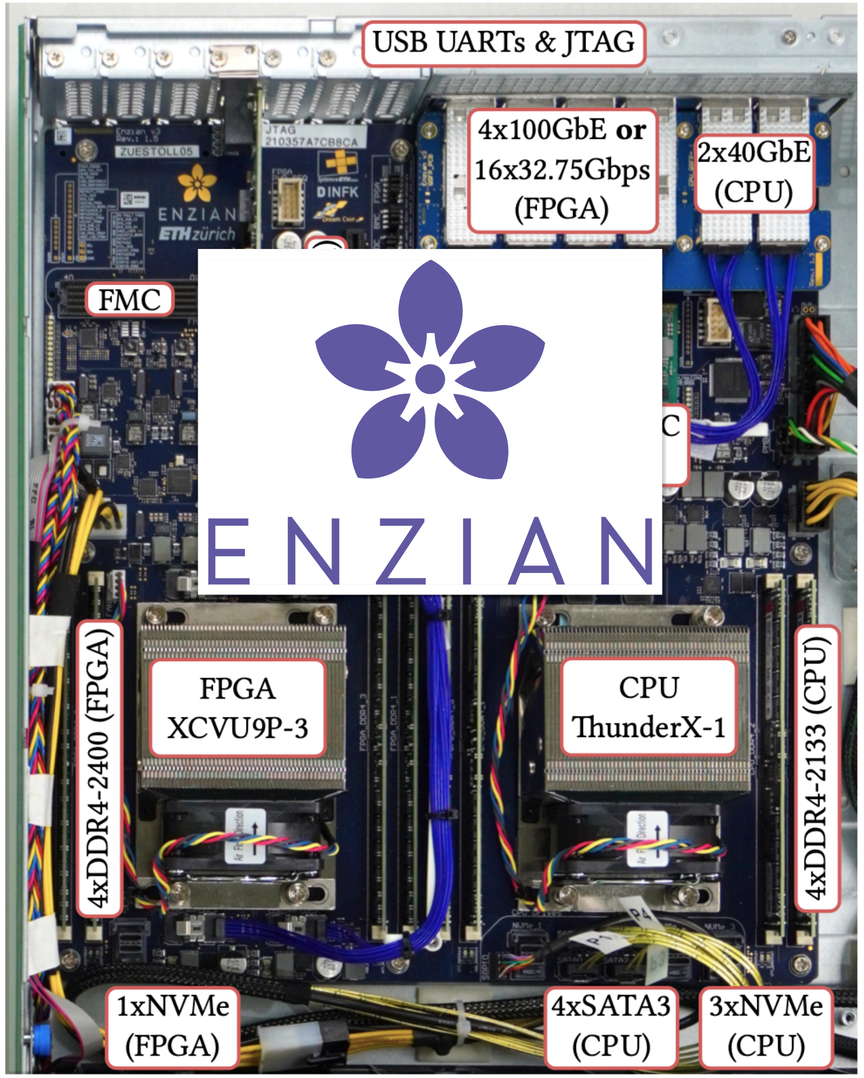
NIC remembers what it put on each core



Scheduling is now a **shared task** between NIC and OS!

Prototype: *Lauberhorn*
Build on Enzian

- Cavium ThunderX-1 CPU
  - *48 cores @ 2 GHz, 128 GB RAM*
- Xilinx VU9P FPGA
  - *4x 100 Gbps Ethernet*
- Enzian Coherent Interconnect (ECI)
  - *Also possible on CXL.mem*



Open questions

Answer these as we build *Lauberhorn*!

- Nested RPCs: crucial for microservices
  - *Send request as "response", synthesise continuation*
- Formal verification: NIC is now in a critical position
  - *Ground-up: specify hardware and prove equivalence*
- Tracing and debugging

Conclusion

- Existing solutions **decouple** the NIC and OS
  - *Out-of-band rebalancing of incoming requests*
- Tightly **couple NIC and OS** in *flow steering* and *scheduling*
  - *Share states with NIC to allow precise, in-band decisions*
- Dispatch RPC in ***3 CPU instructions***