Systems@ETH Zürich

Advanced Computer Networks

263-3501-00

End Host Optimization

Patrick Stuedi

Spring Semester 2017

Today

- End-host optimizations:
  - NUMA-aware networking
  - Kernel-bypass
  - Remote Direct Memory Access (RDMA)

Trends

- Networks get faster
  - 10Gb/s $\rightarrow$ 40Gb/s $\rightarrow$ 100Gb/s
  - Latency decreases ($\approx 1\mu$s)
- CPUs do not
  - But there are more of them
  - Latency is increasingly a software factor
- Moore's law continues
  - More transistors *on a NIC*

Key challenges

- Scaling
  - One fast network, lots of cores
- Latency
  - Dominated by software processing cost
  - Multiplied by multi-tier server architectures
- CPU load
  - Packet processing and data copying cost can be high

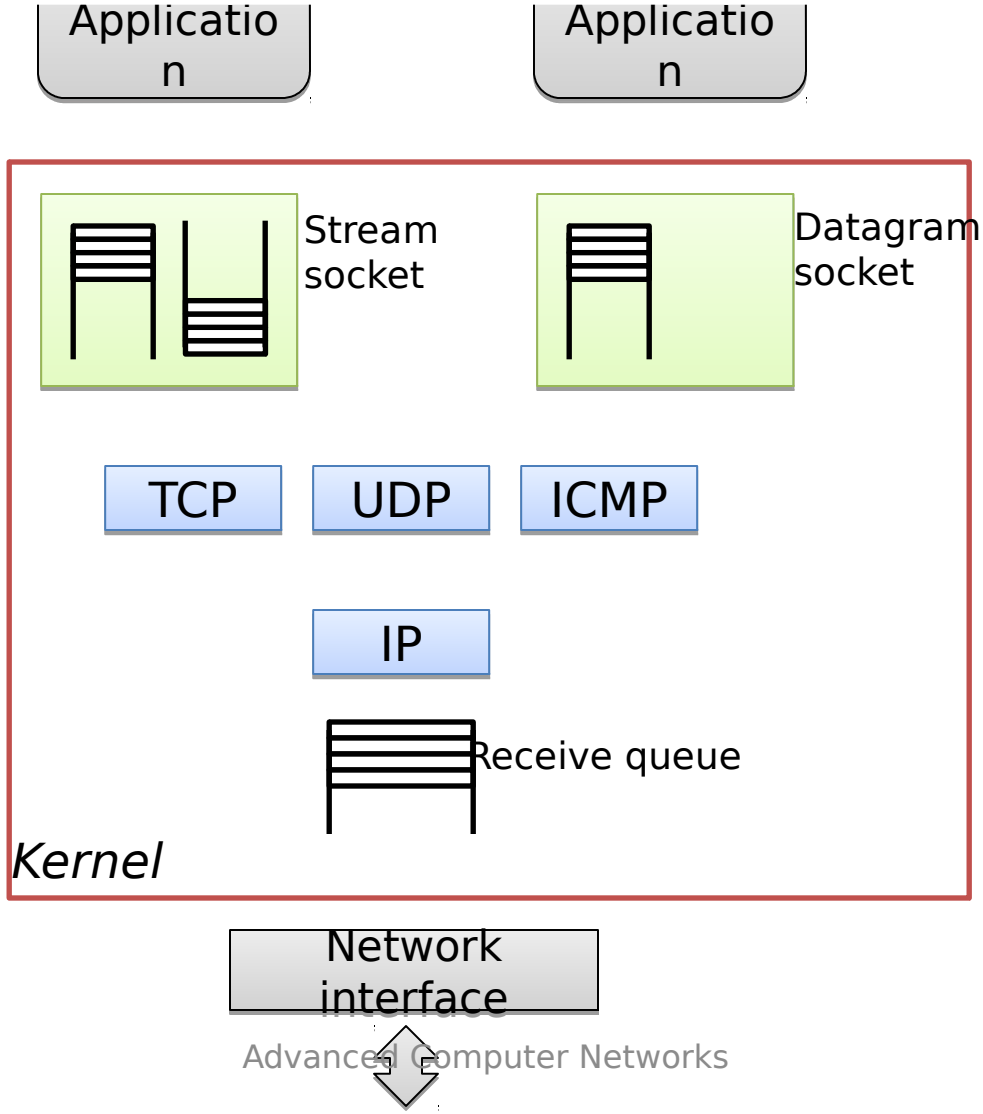
Delay numbers

Component	Delay
Network switch	10-30 μs
Network adaptor	2.5-32 μs
OS network stack	15 μs
Speed of light (in fibre)	5 ns/m ⇒ 0.05-0.5 μs

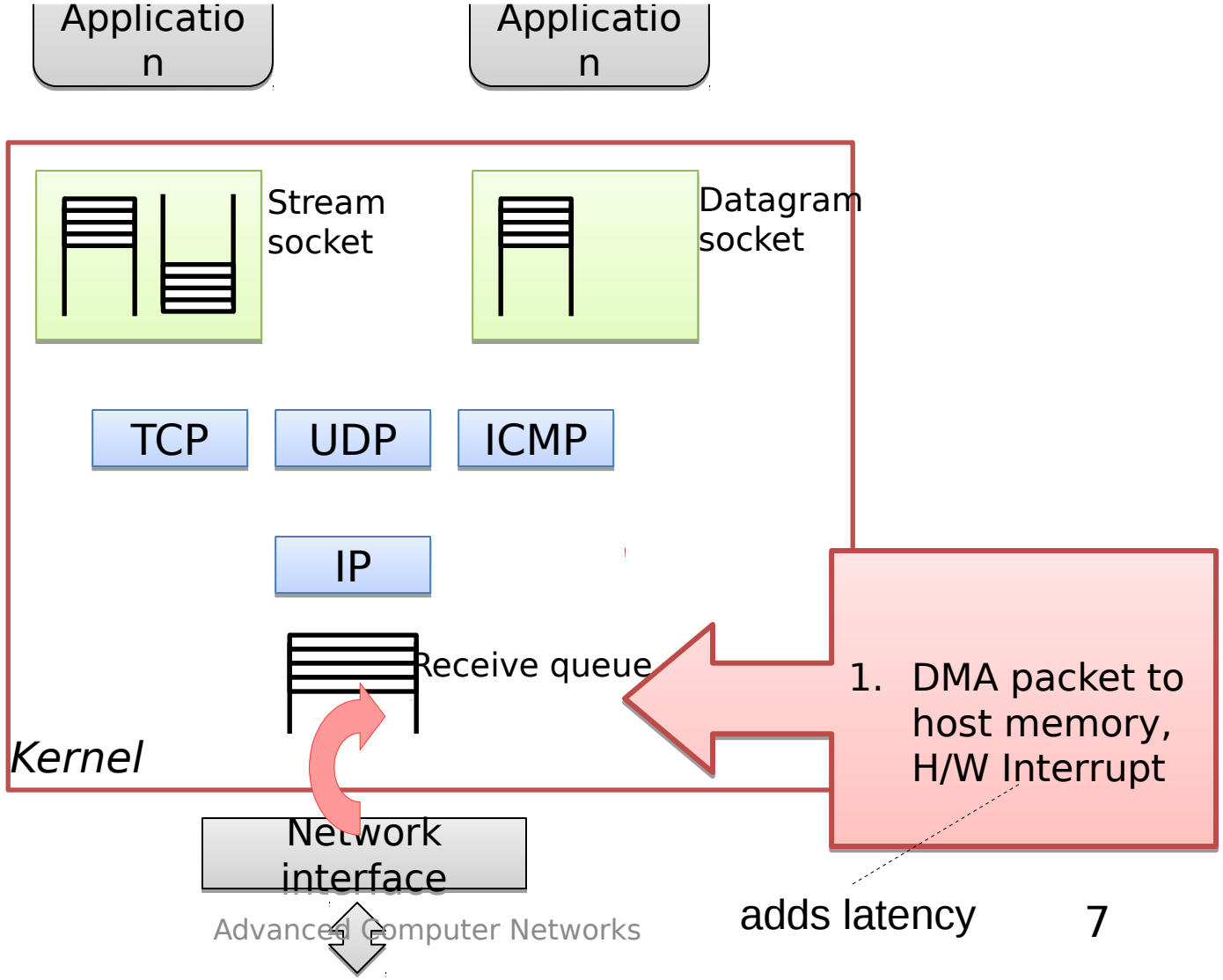
- OS overhead per packet exchanged between two hosts attached to the same switch:

  (2*15)/(2*2.5+2*15+10)=66% (!)

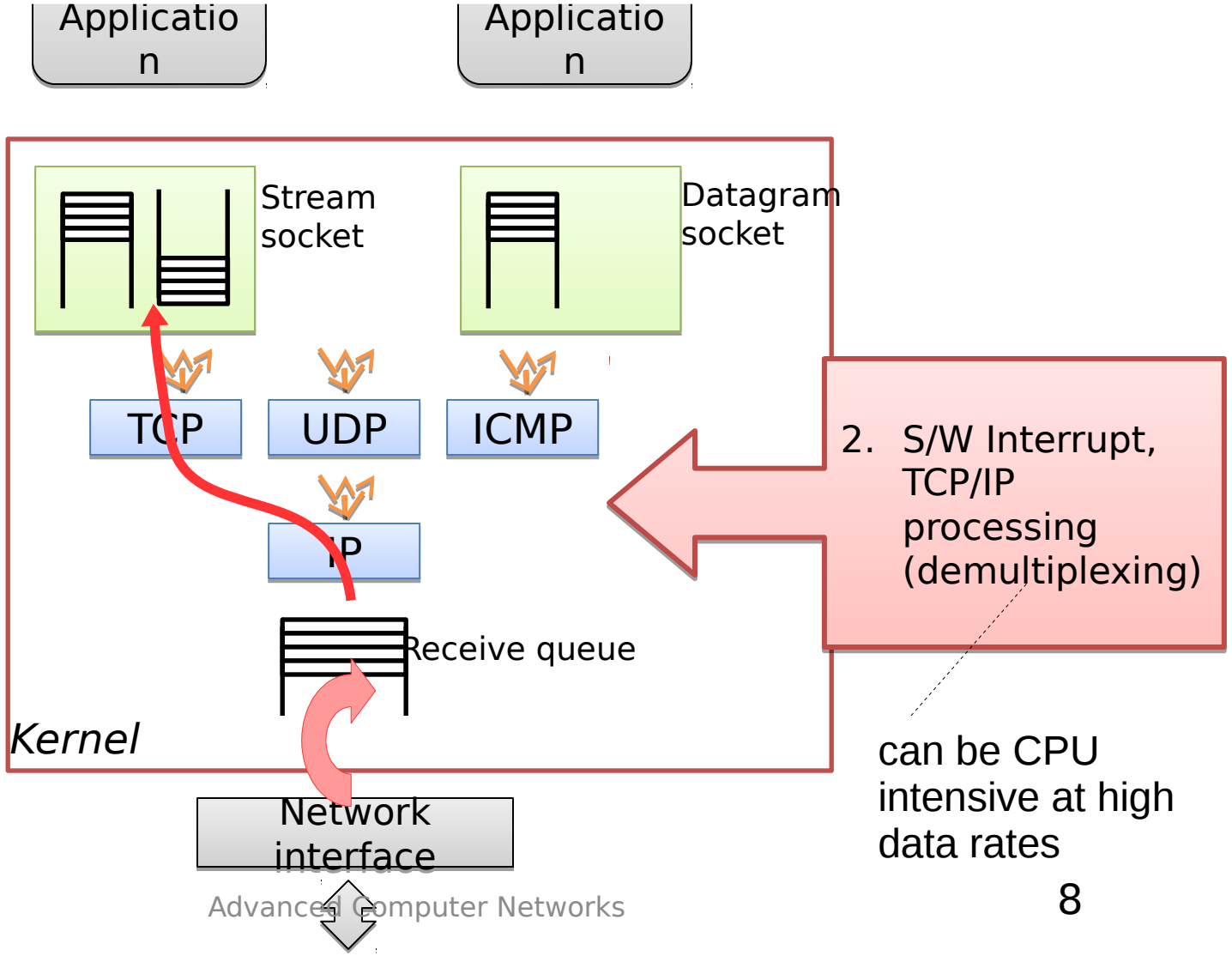
Receiving a packet



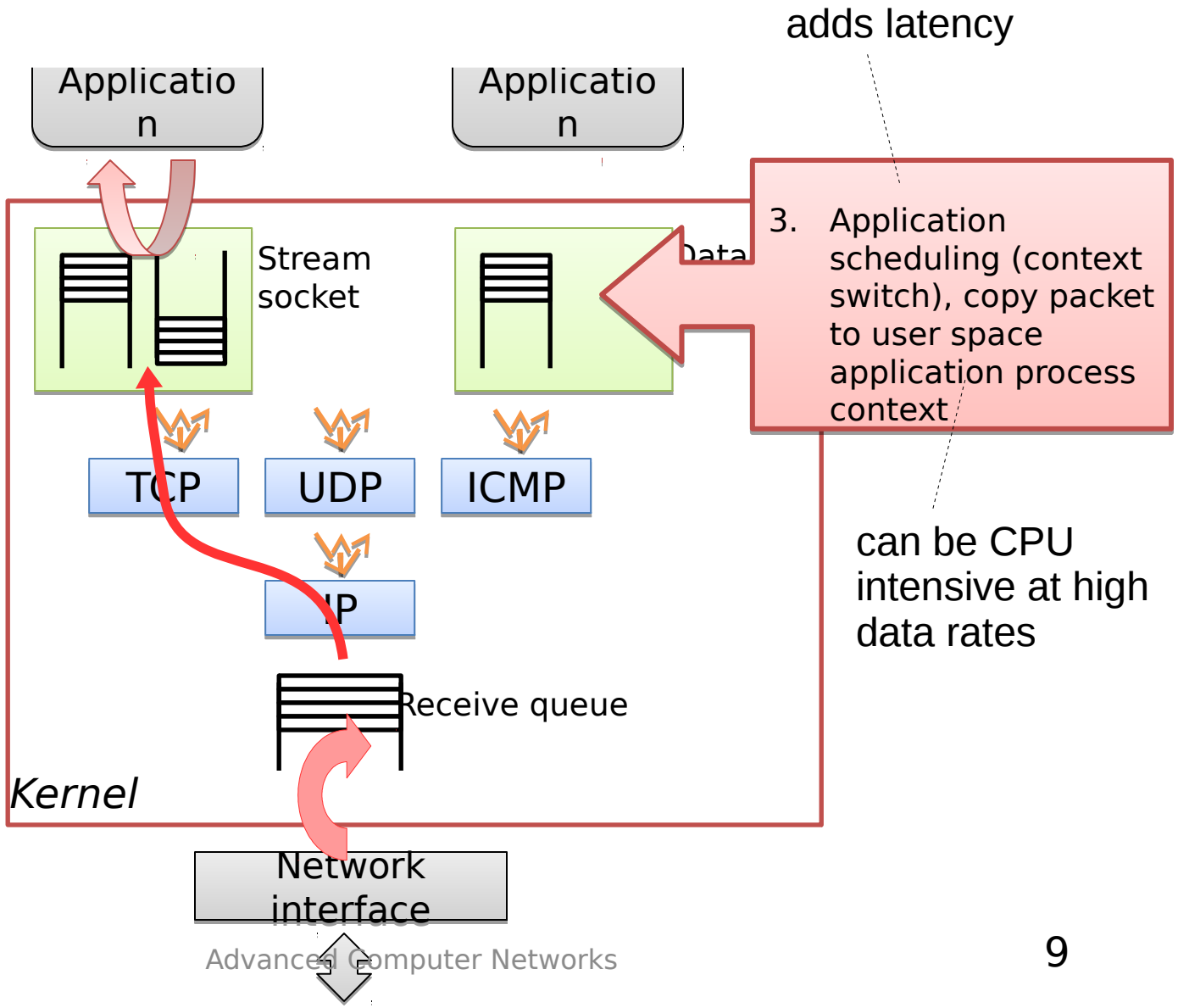
Receiving a packet



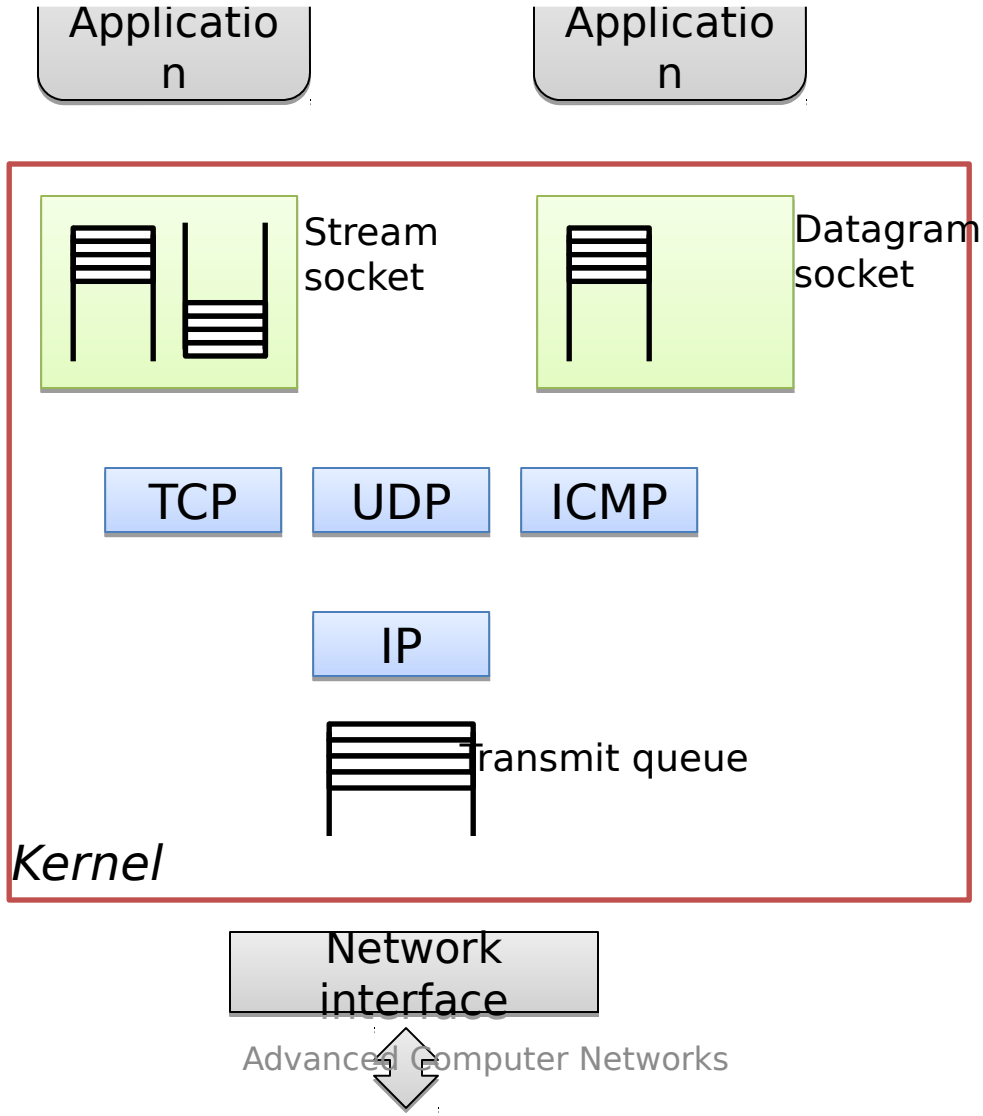
Receiving a packet



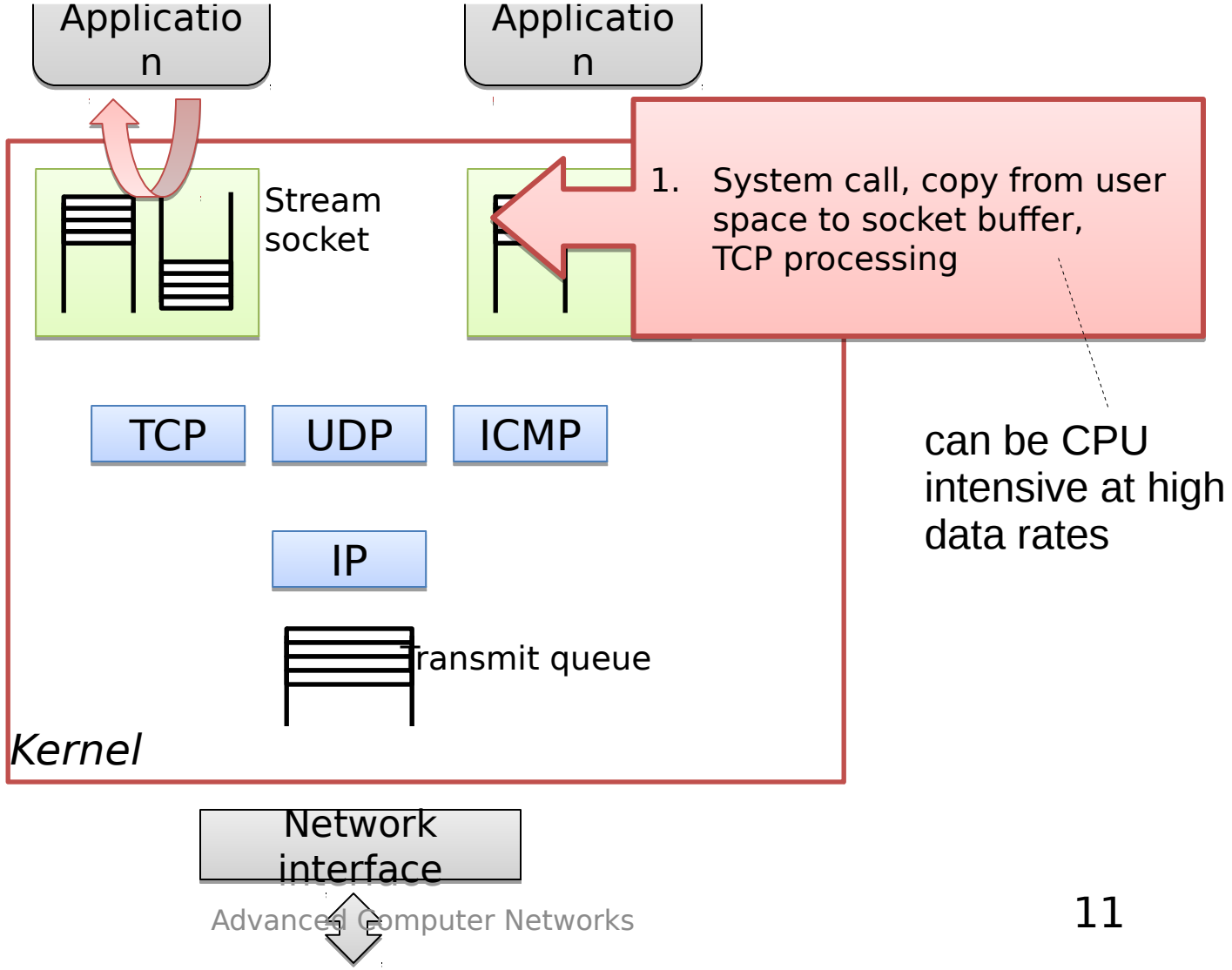
Receiving a packet



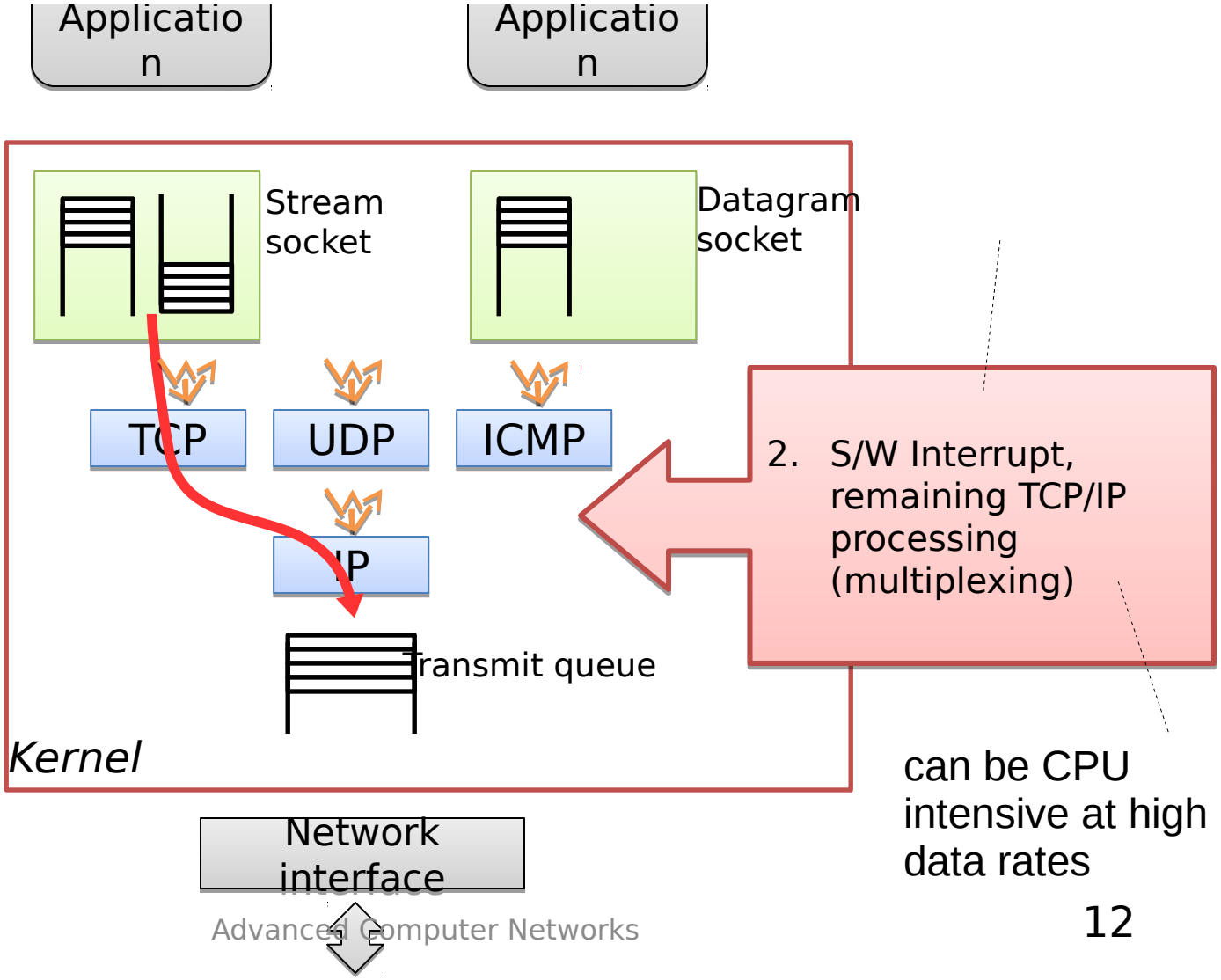
Sending a packet



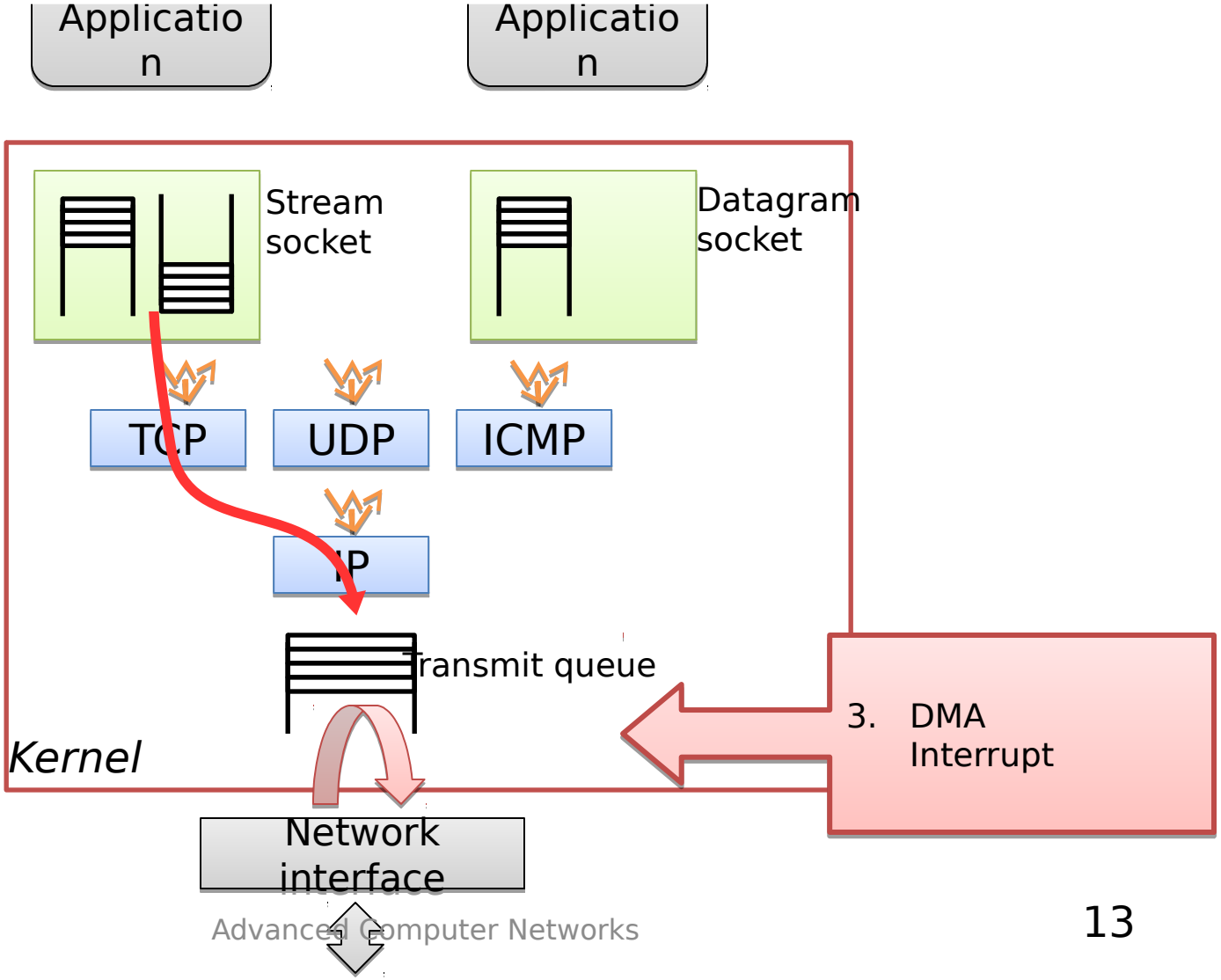
Sending a packet



Sending a packet

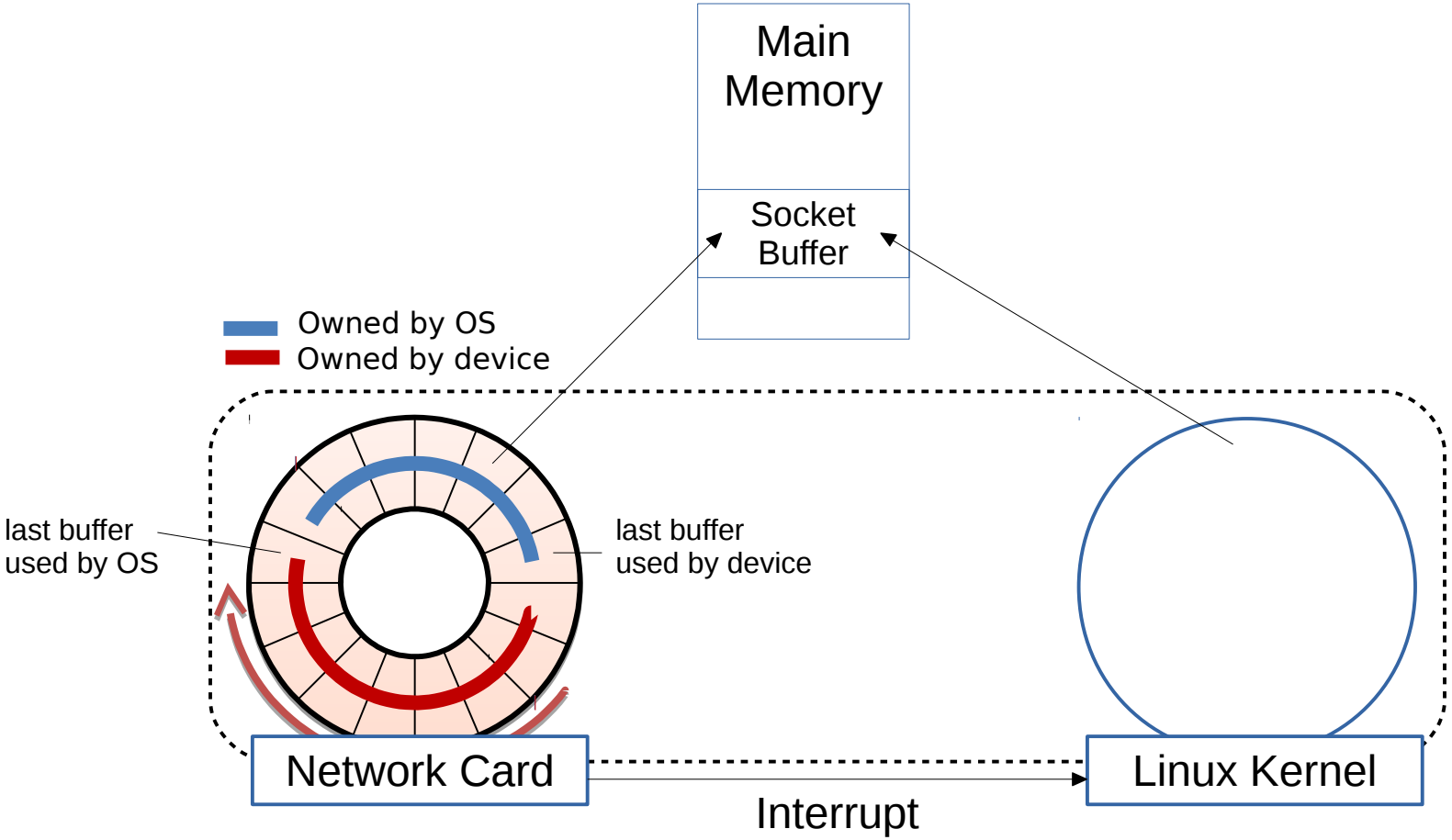


Sending a packet



What does the hardware do?

Descriptor rings



15

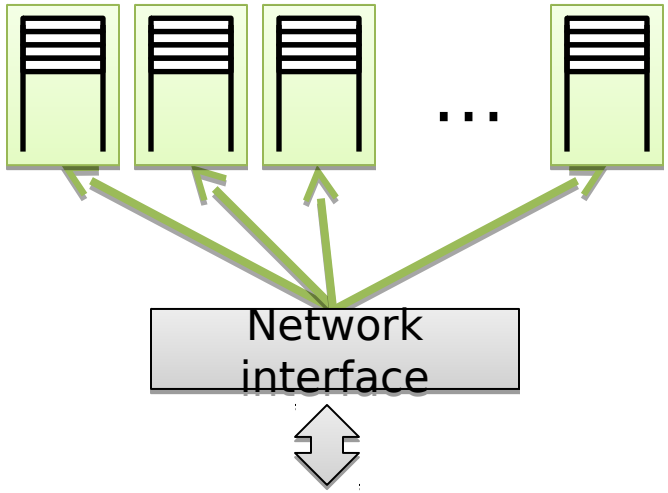
Overruns

- Device has no buffers for received packets
  ⇒ starts discarding packets
  - Not as bad as it sounds
  - Signals that it's lost some in a status register
- CPU has no more slots to send packets ⇒ must wait
  - Can spin polling, but inefficient
  - Signals device to interrupt it when a packet has been sent
    i.e. a buffer slot is now free

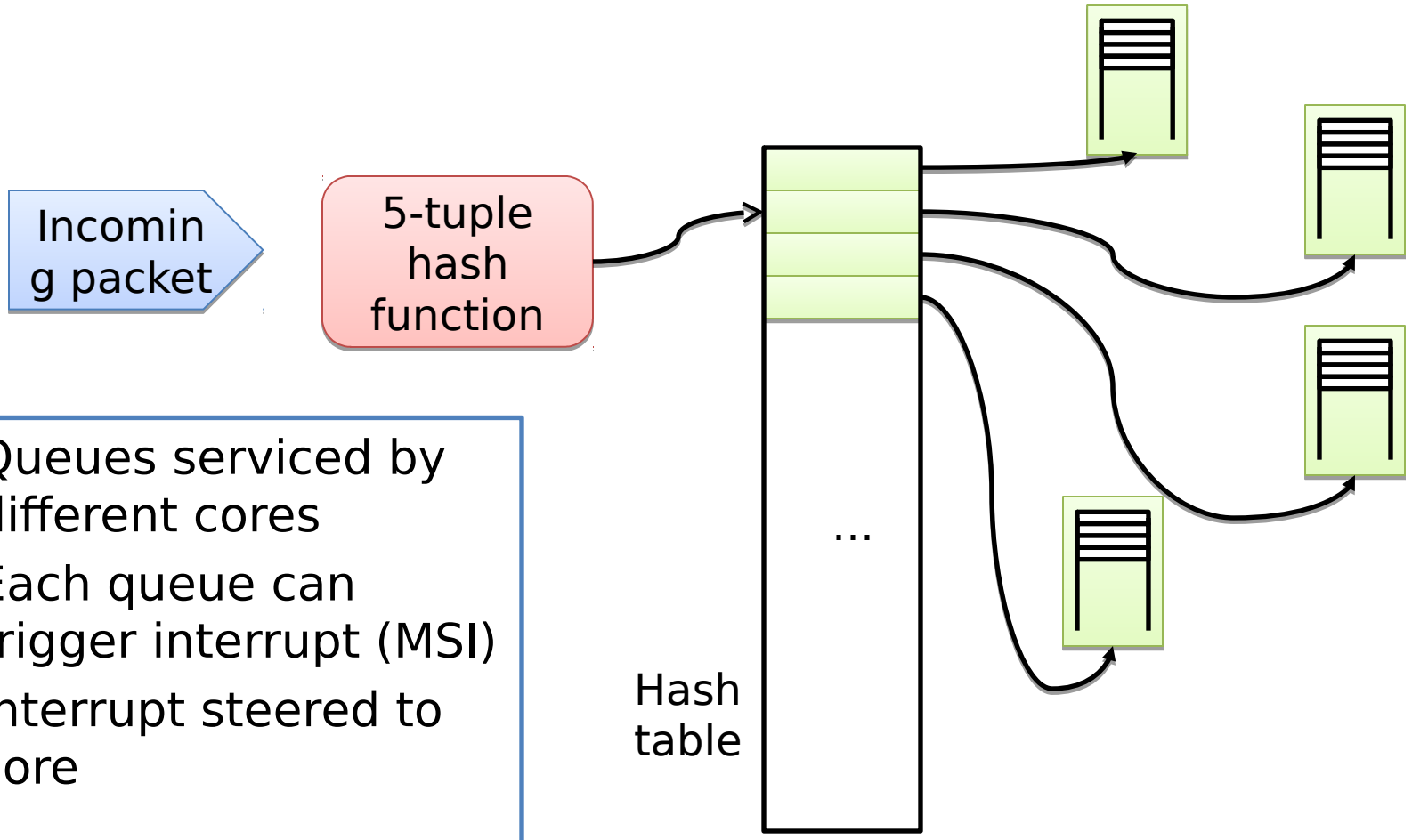
Receive-Side Scaling (RSS)

Scaling with cores

- Use multiple queues (descriptor rings)
  - One queue per core ⇒ better scaling
  - Per-queue interrupts ⇒ direct to core
- Key question: which Rx queue gets a packet?



RSS



- Queues serviced by different cores
- Each queue can trigger interrupt (MSI)
- Interrupt steered to core

More sophisticated filters

- Wildcard matches on 5-tuples
- VLAN matching
- SYN filter
  - Direct connection setups to a different queue
- Filters based on other protocols
- etc.

What about transmit?

- Multiple transmit queues:
  - Better scaling: each core has a tx queue
  - No locks or synchronization between cores
- Performance isolation:
  - NIC can schedule different tx queues
  - CPU scheduler not involved

Example: Intel 82599EB 10Gb NIC

- 128 Send queues
- 128 Receive queues
- RSS filters
- 256 5-tuple filters
- Large number of "Flow Director" filters
- SYN filter
- etc.

What this buys you…

- Scaling with cores
- Reduced CPU load
  - Synchronization
  - Multiplexing
  - Scheduling
- Performance isolation
  - Receive: livelock

If the OS is designed accordingly

What it doesn't

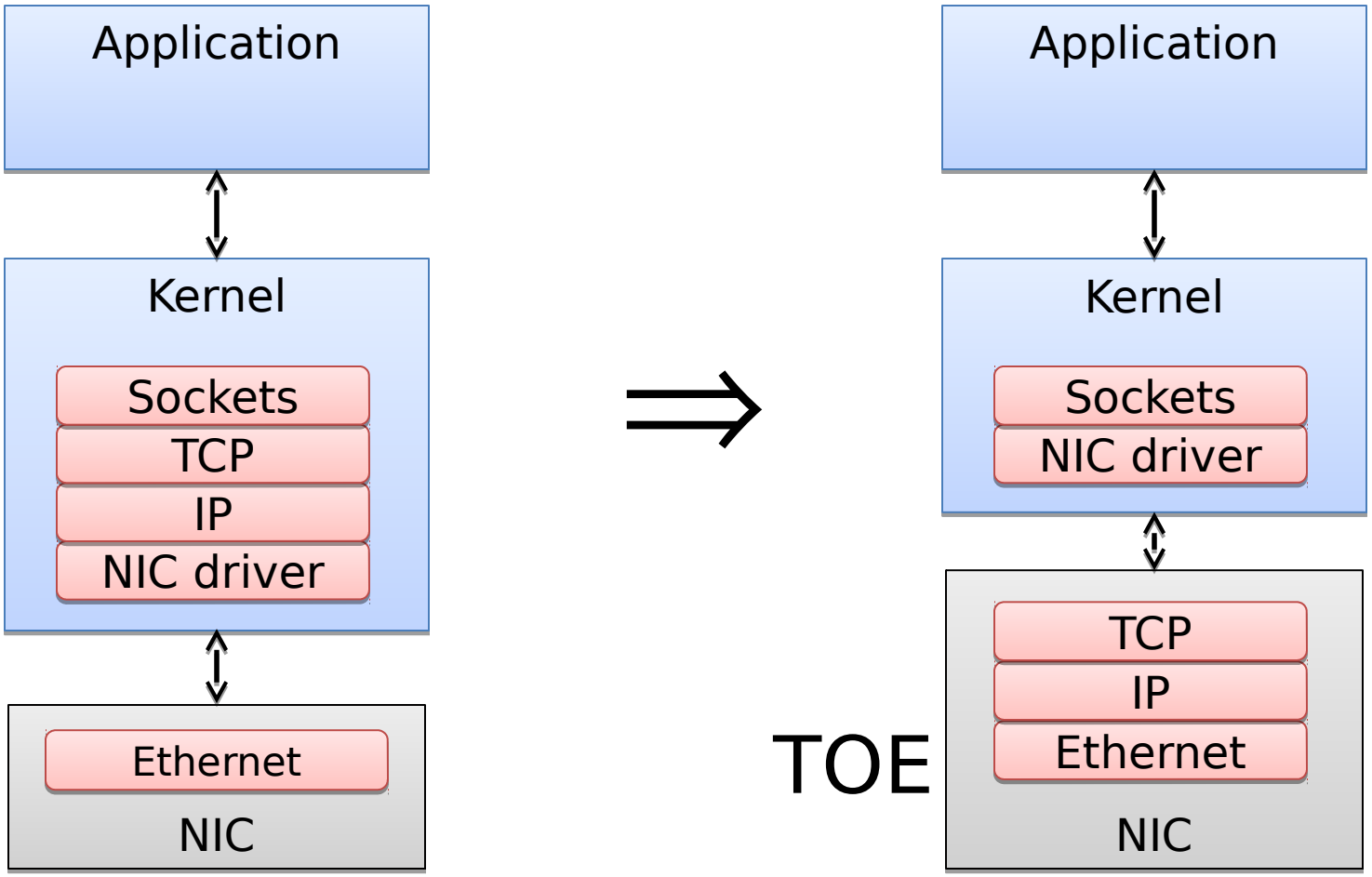
- CPU load:
  - Demultiplexing on a queue
  - Protocol processing
  - Connection setup/teardown
- Context switches
  - Must still switch to receiving process
  - Enter/leave kernel

TCP Offload

TCP Offload

- Moving IP and TCP processing to the Network Interface (NIC)
- Main justification:
  - Reduction of host CPU cycles for protocol header processing, checksumming
  - Fewer CPU interrupts
  - Fewer bytes copied over the memory bus
  - Potential to offload expensive features such as encryption

TCP Offload Engines (TOEs)



Why TCP offload never worked

- Moore’s Law worked against “smart” NICs
  - CPU's used to be fast enough
- TCP/IP headers don’t take many CPU cycles
  - ≈ 30 instructions if done with care.
- TOEs impose complex interfaces
  - TOE ↔ CPU protocol can be worse than TCP!
- Connection management overhead
  - For short connections, overwhelms any savings
- OS can't control protocol implementation
  - Bug fixes
  - Protocol evolution (DCTCP, D2TCP, etc.)

Why TOEs sometimes help

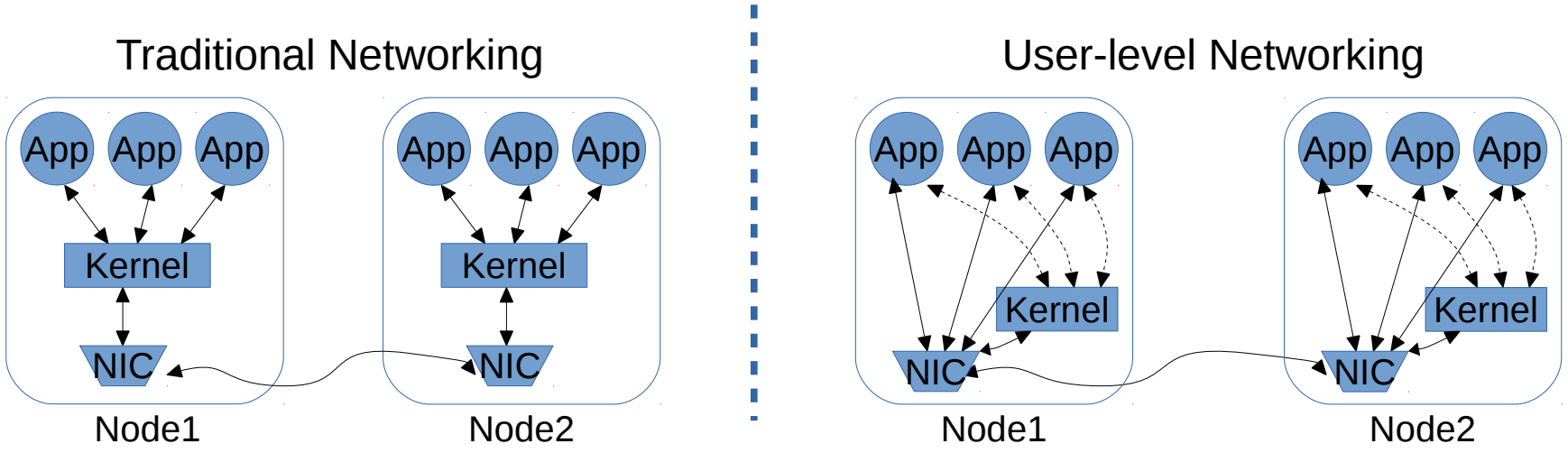
- Now many cores, cores don't get faster
  - Network processing is hard to parallelize
- Sweet spot might be apps with:
  - Very high bandwidth
  - Relatively low end-to-end latency network paths
  - Long connection durations
  - Relatively few connections
- For example:
  - Storage-server access
  - Cluster interconnects

User-level networking

User-level Networking

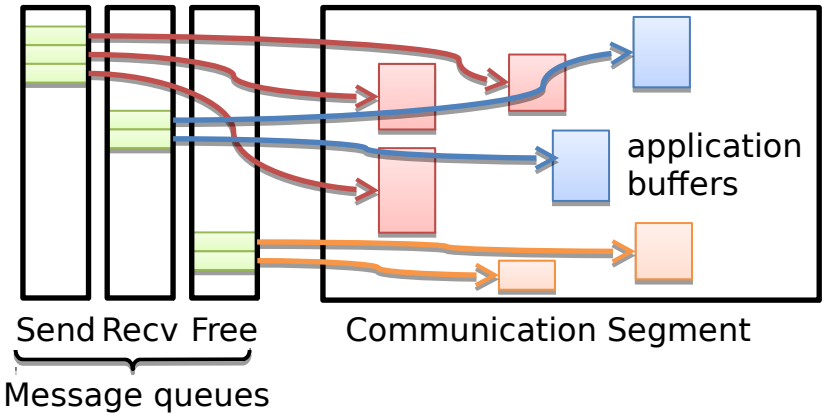
- TCP offloading is not enough:
  - System call overhead
  - Context switches
  - Memory copies
- User-level networking key idea: remove overhead
  - Map individual queues to application
  - Allow applications to poll
  - OS-byapss (OS still needed for interrupts)
- Requires hardware which:
  - Can validate queue entries
  - Demultiplex messages to applications

User-space networking (2)



- Traditional networking
  - All communication through kernel
- User-level networking
  - Applications acces NIC directly
  - Kernel involved only during connection setup/teardown
- “U-Net: a user-level network interface for parallel and distributed computing”
  - Eicken, Basu, Buch, Vogels, Cornell University, 1995

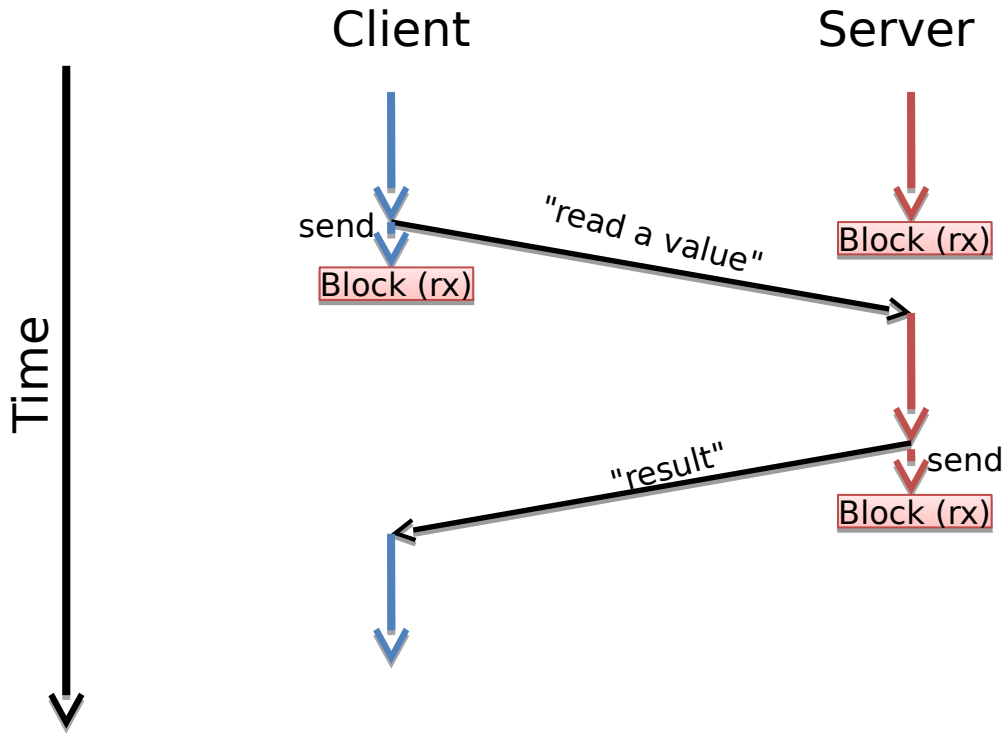
U-Net Data Structures & API



- Data structures:
  - Communication segment: application's handle
  - Buffers: hold data for sending and receiving
  - Message queues: hold descriptors to buffers
- Programming U-Net
  - Sending: compose data in buffer, post descriptor to send queue
  - Receiving: post descriptor to receive queue, poll status
- Messages contain TAG identifying source or destination queue

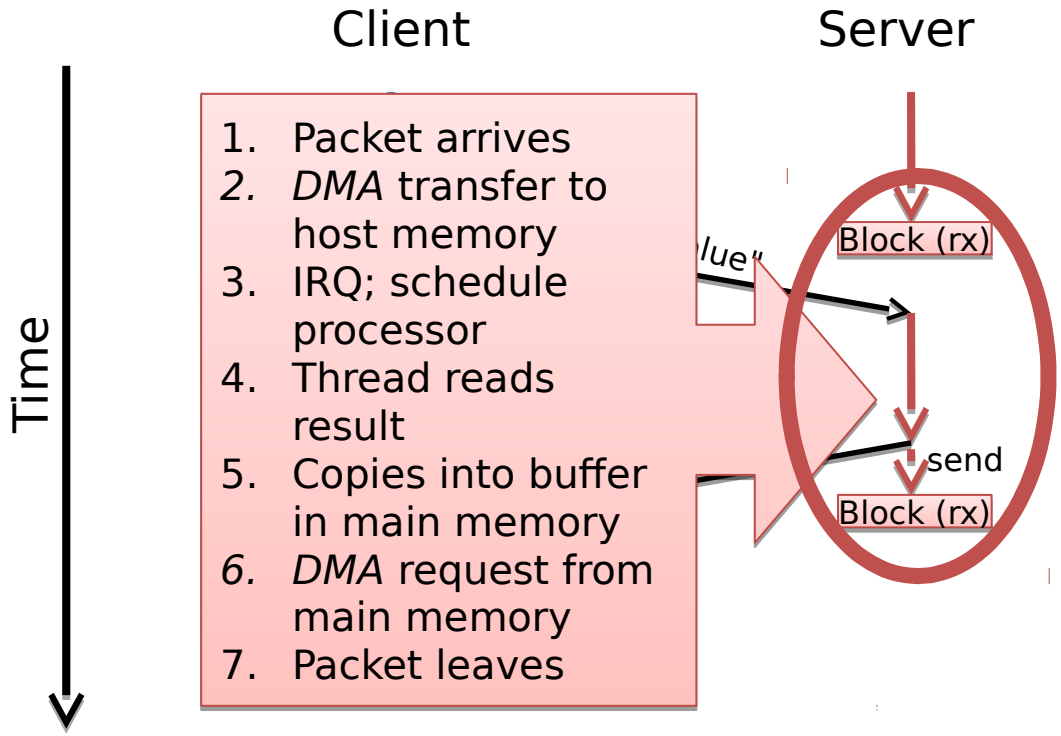
RDMA

Message passing exchange



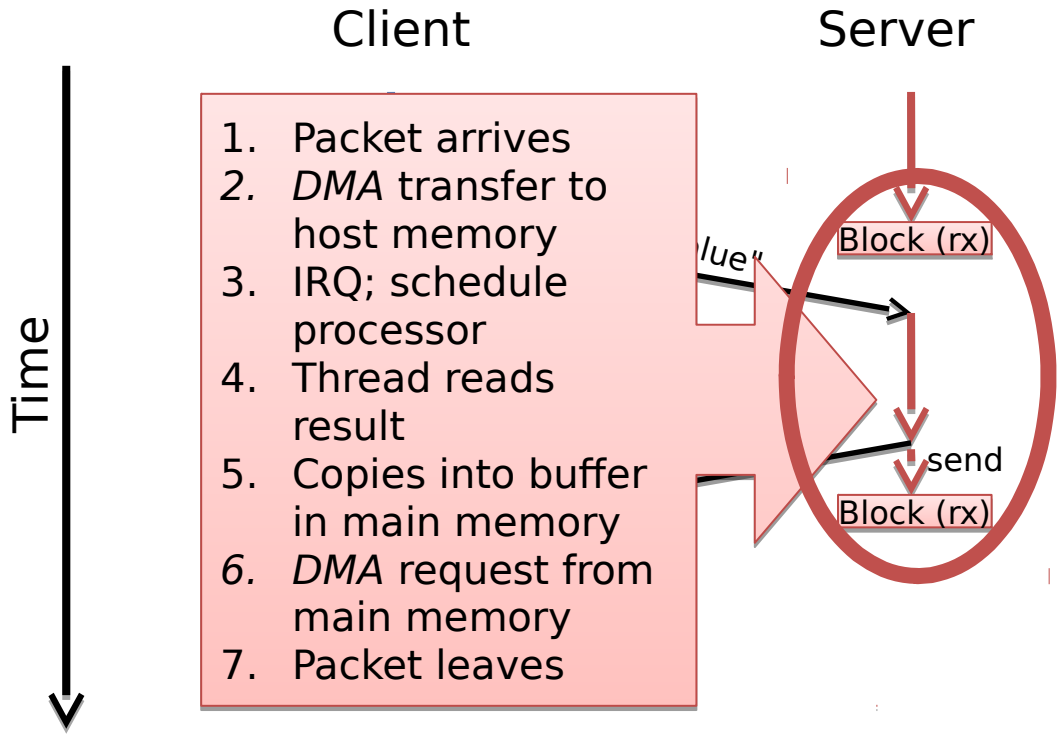
- So far: messages between processes

Message passing exchange



- So far: messages between processes

Message passing exchange

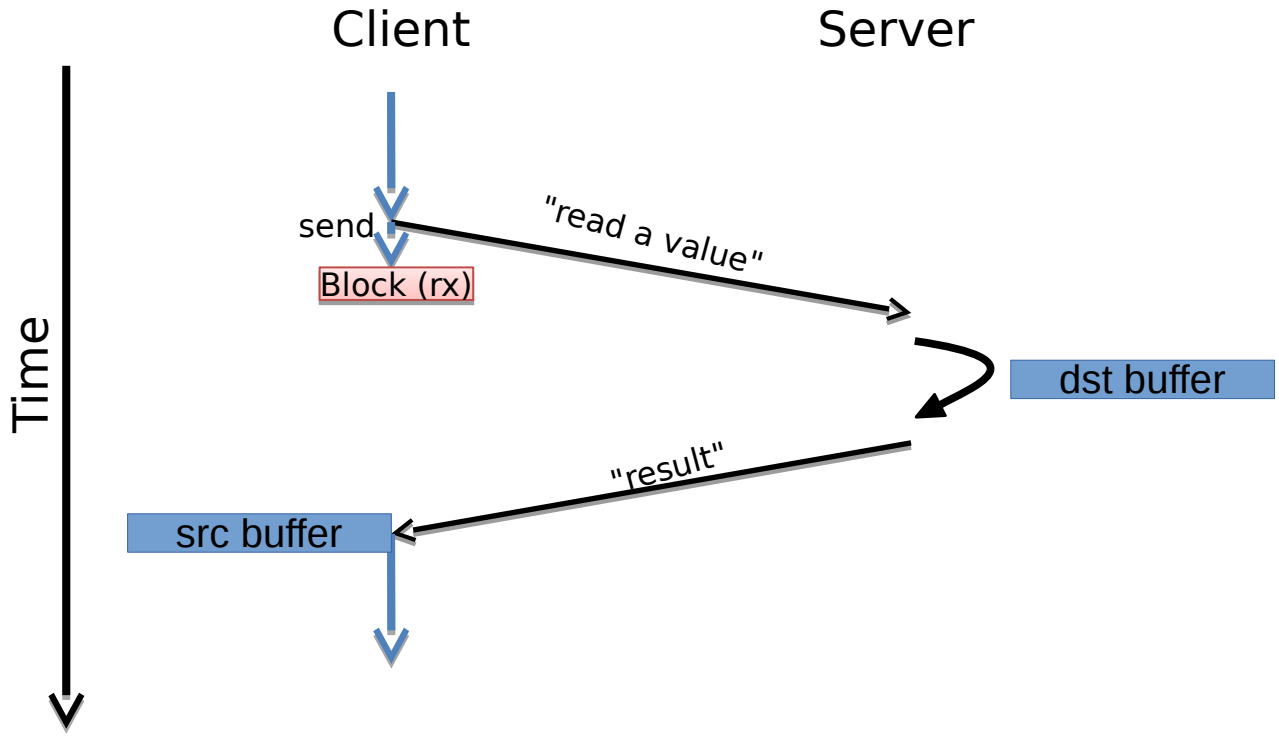


- So far: messages between processes
- What about removing the server software entirely?

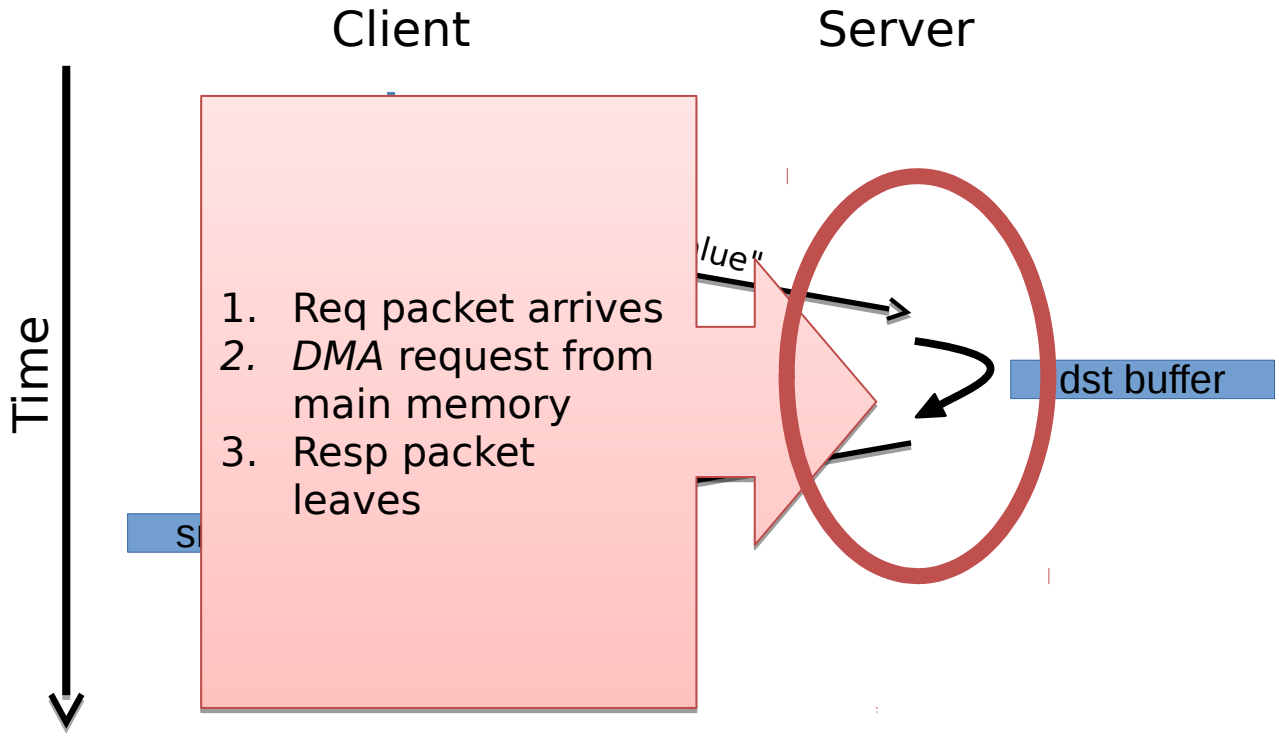
Remote Direct Memory Access (RDMA)

- Only the "client" actively involved in transfer
  - "One-sided" operation
- RDMA Write specifies:
  - where the data should be taken from locally
  - where it is to be placed remotely
- RDMA Read:
  - where the data should be taken from remotely
  - Where it is to be placed locally
- Require buffer advertisement prior to data exchange

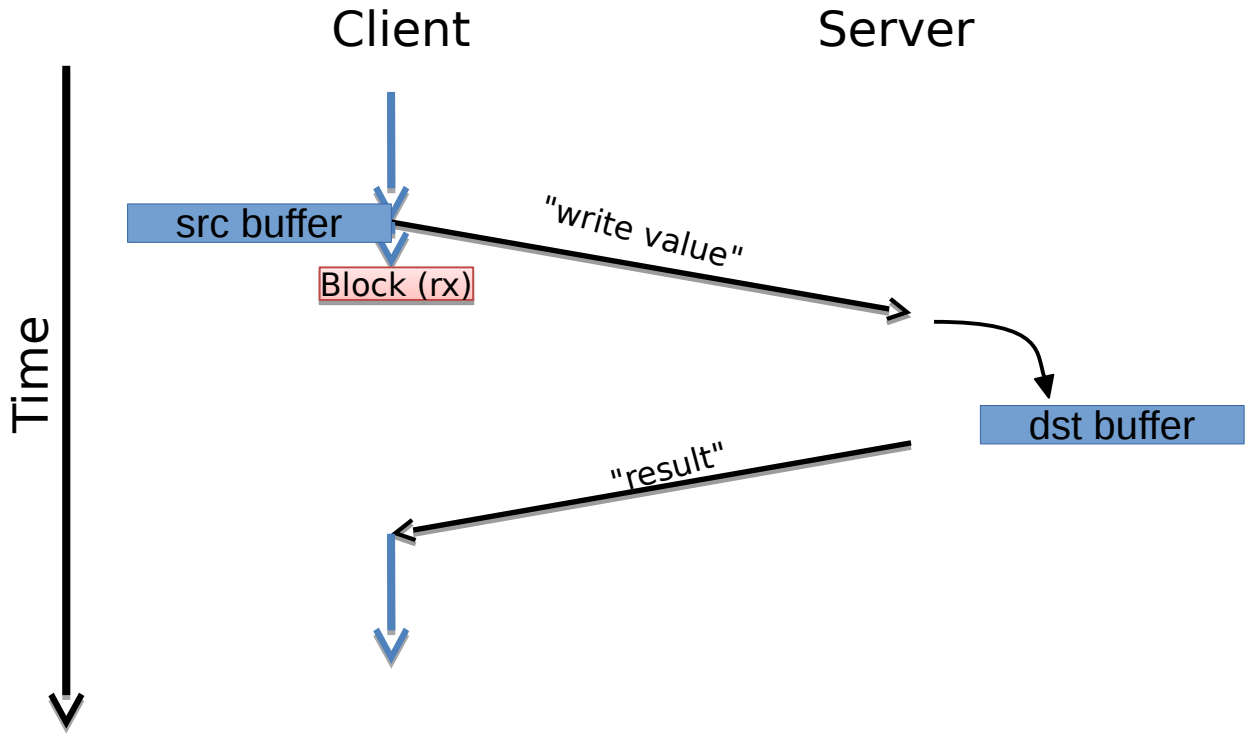
RDMA Read


Client
Server
Time
send
Block (rx)
"read a value"
dst buffer
"result"
src buffer

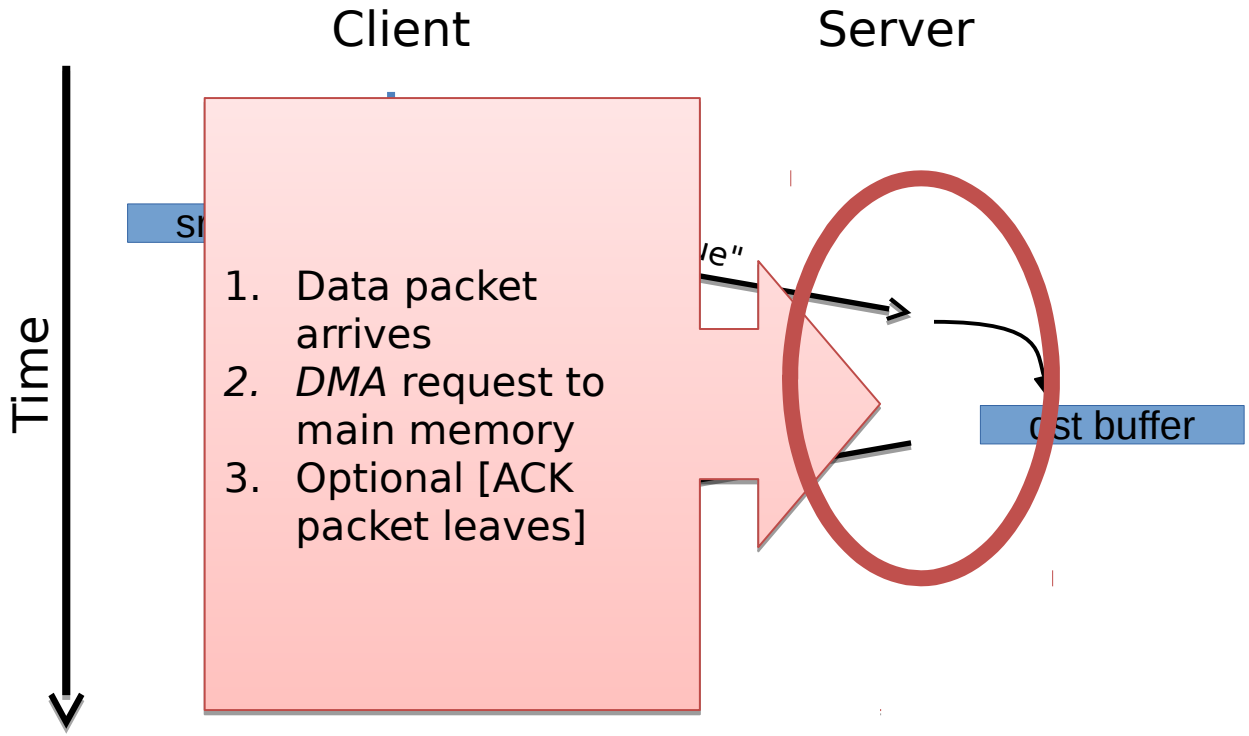
RDMA Read


Client
Server
Time
1. Req packet arrives
2. *DMA* request from main memory
3. Resp packet leaves
dst buffer

RDMA Write


Client
Server
Time
src buffer
Block (rx)
"write value"
dst buffer
"result"

RDMA Write

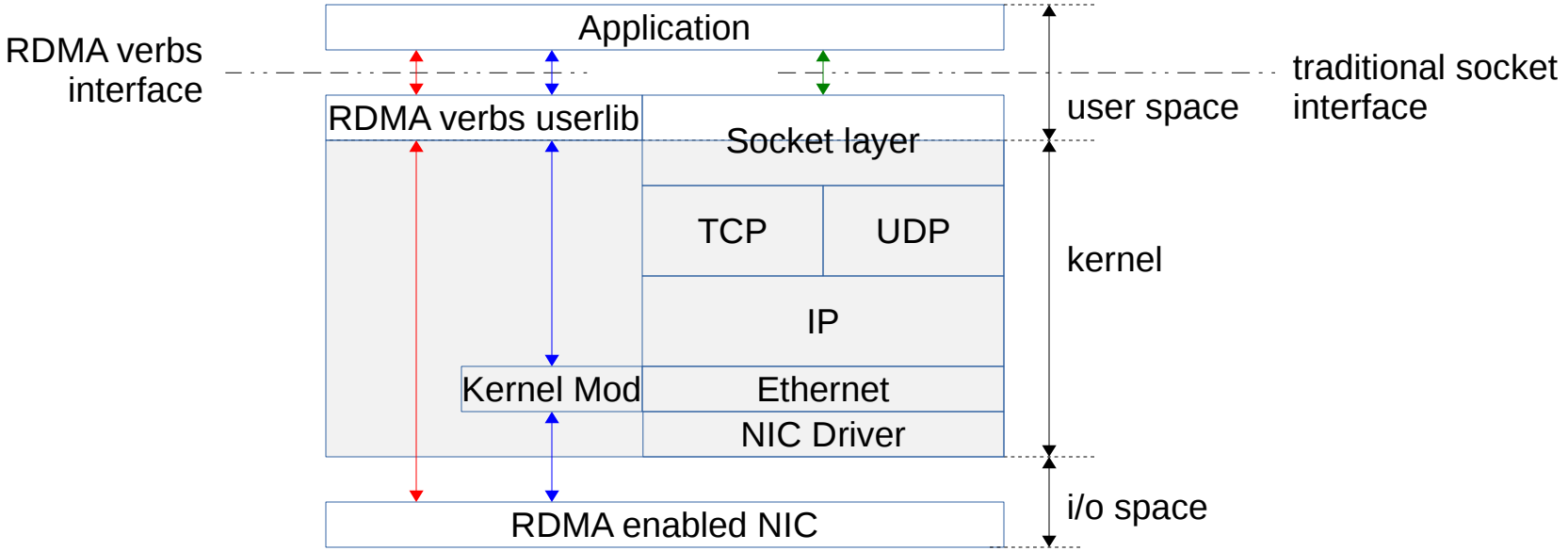

Client
Server
Time
1. Data packet arrives
2. *DMA* request to main memory
3. Optional [ACK packet leaves]

RDMA implementations

- Infiniband
  - Compaq, HP, IBM, Intel Microsoft and Sun Microsystems, 2000
  - New network from ground up (see Lecture Flow Control)
- IWARP (Internet Wide Area RDMA Protocol)
  - RDMA semantics over offloaded TCP/IP
  - Requires custom Ethernet NICs
- RoCE
  - RDMA semantics directly over Ethernet

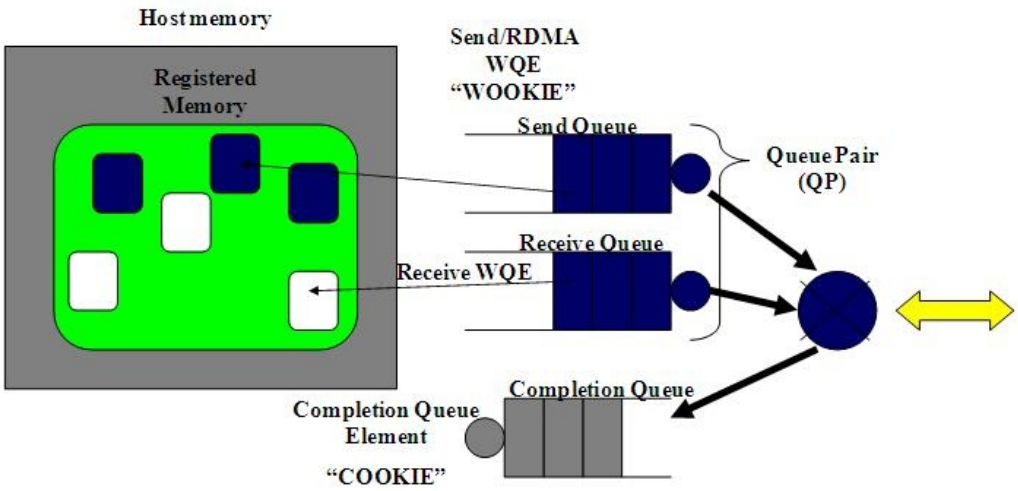
- All implementations provide both:
  - Classical message passing (send/recv similar to U-Net)
  - RDMA operations (RDMA read/write)

Open Fabrics Enterprise Distribution (OFED)



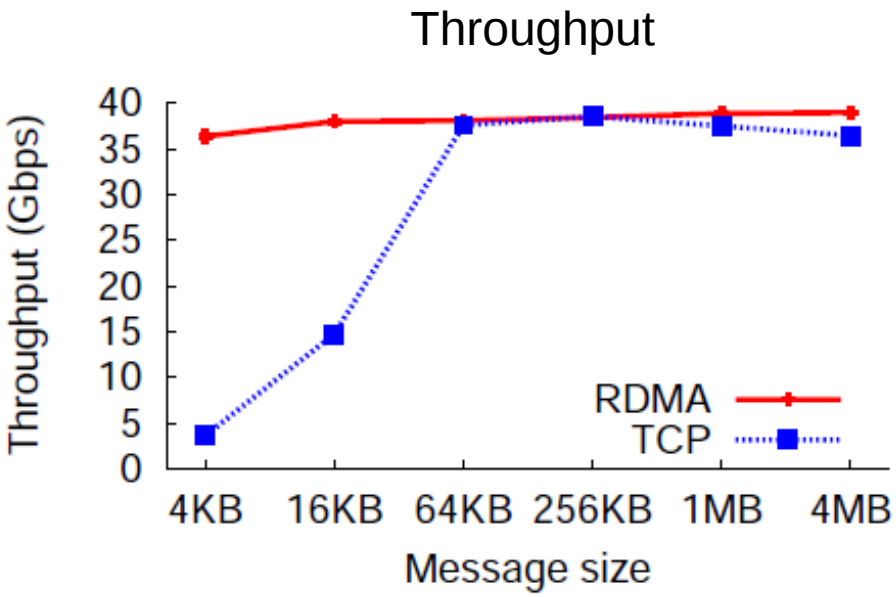
- Unified RDMA stack different OS and hardware
  - Linux/Windows
  - Infiniband, iWARP, RoCE
- Vendors write their own device drivers and user library
  - Device driver implements allocation of message queues on device
  - User driver provides access to message queues from user user space
- Stack provides common application interface calls “verbs” interface

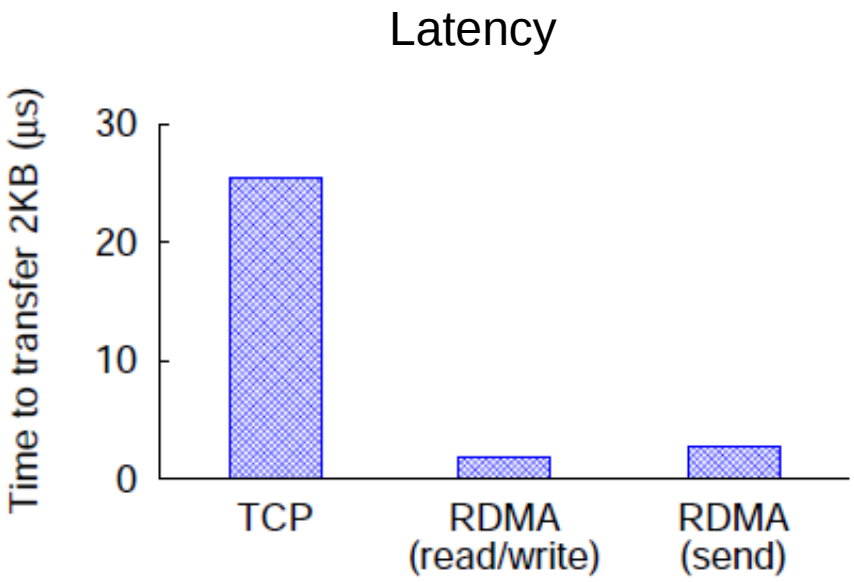
“Verbs” Data Structures & API



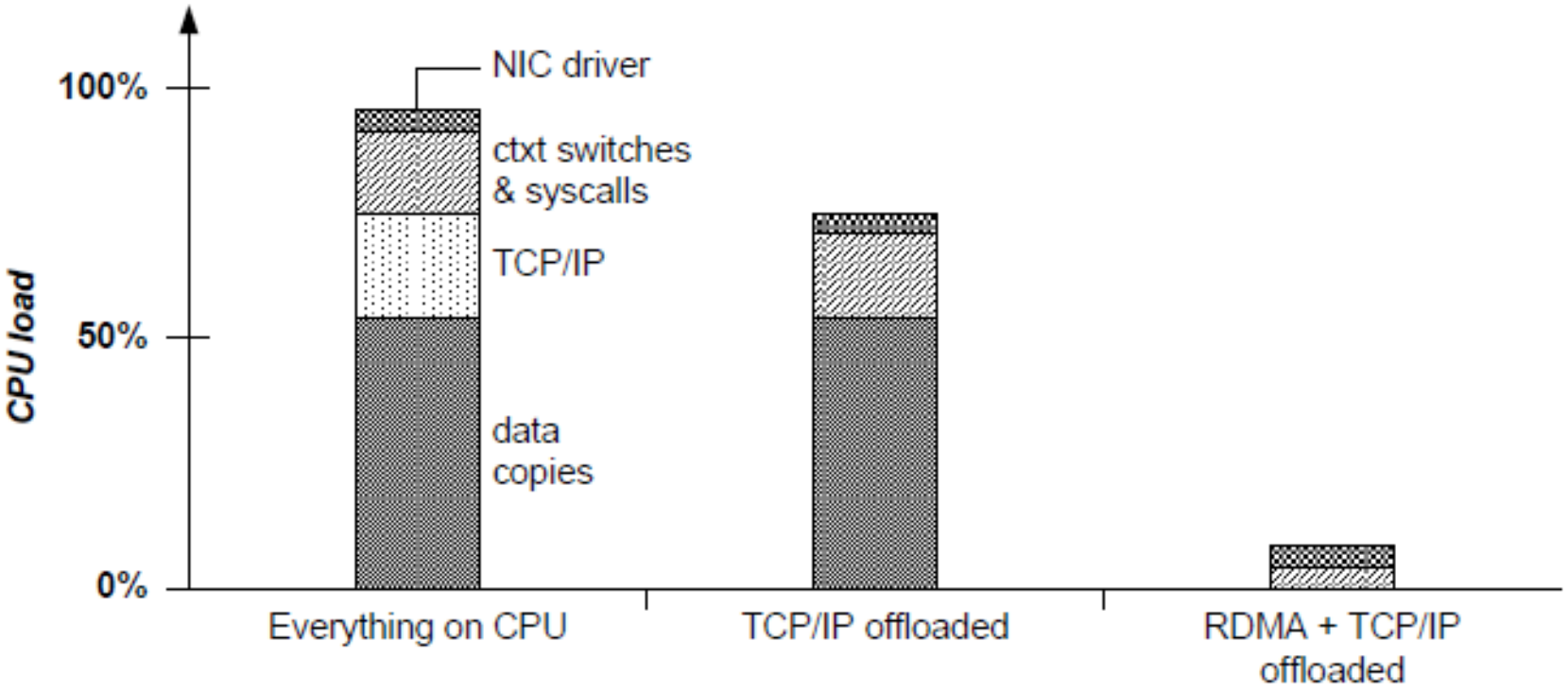
- Applications use 'verbs' interface to
  - Register application memory:
    - Operating system will make sure the memory is pinned and accessible by DMA
  - Create a queue pair (QP)
    - send/recv queue
  - Create a completion queue (CQ)
    - RNIC puts a new completion-queue element into the CQ after an operation has completed
  - Send/Receive/Read/Write data
    - Place a work-request element (WQE) into the send or recv queue
    - WQE points to user buffer and defines the type of the operation (e.g., send, recv, read, write)

RDMA vs TCP-Sockets

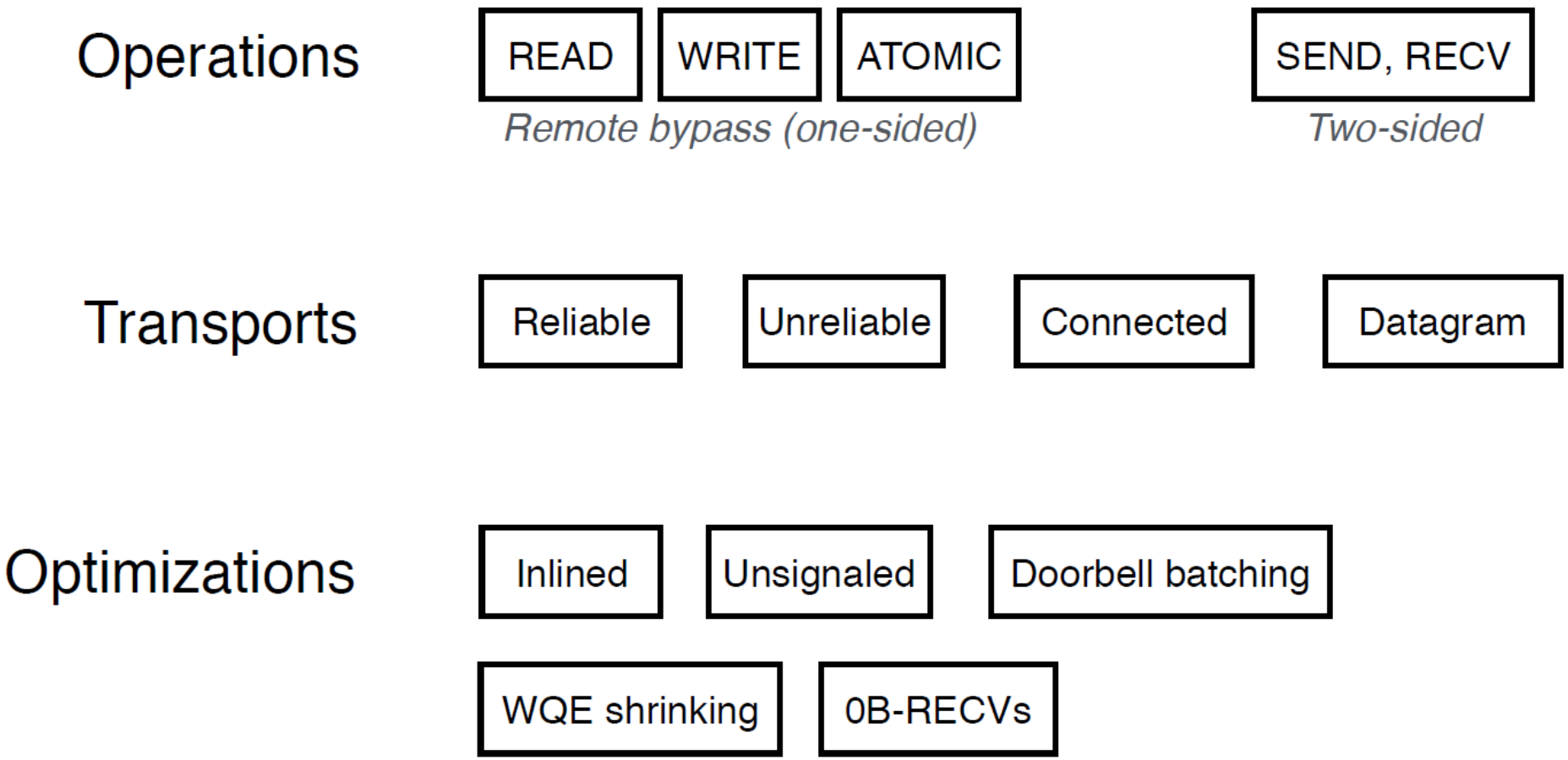




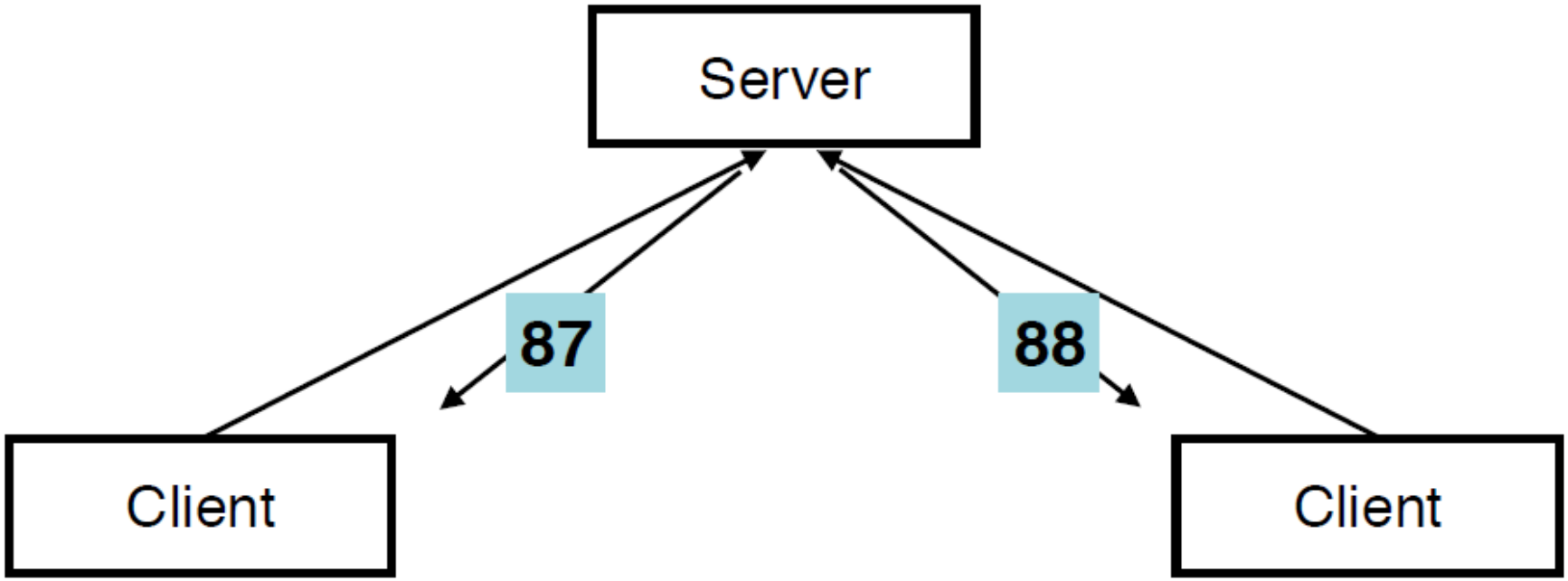
Typical CPU loads for three network stack implementations



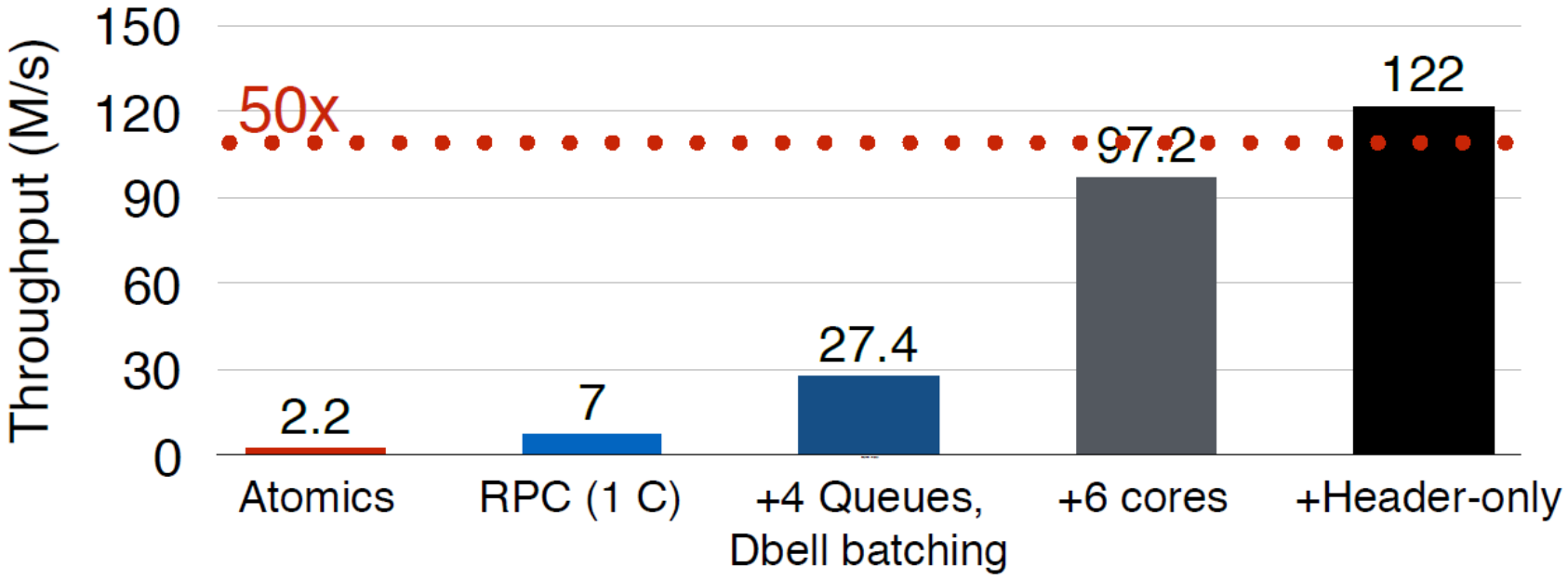
Large RDMA Design Space



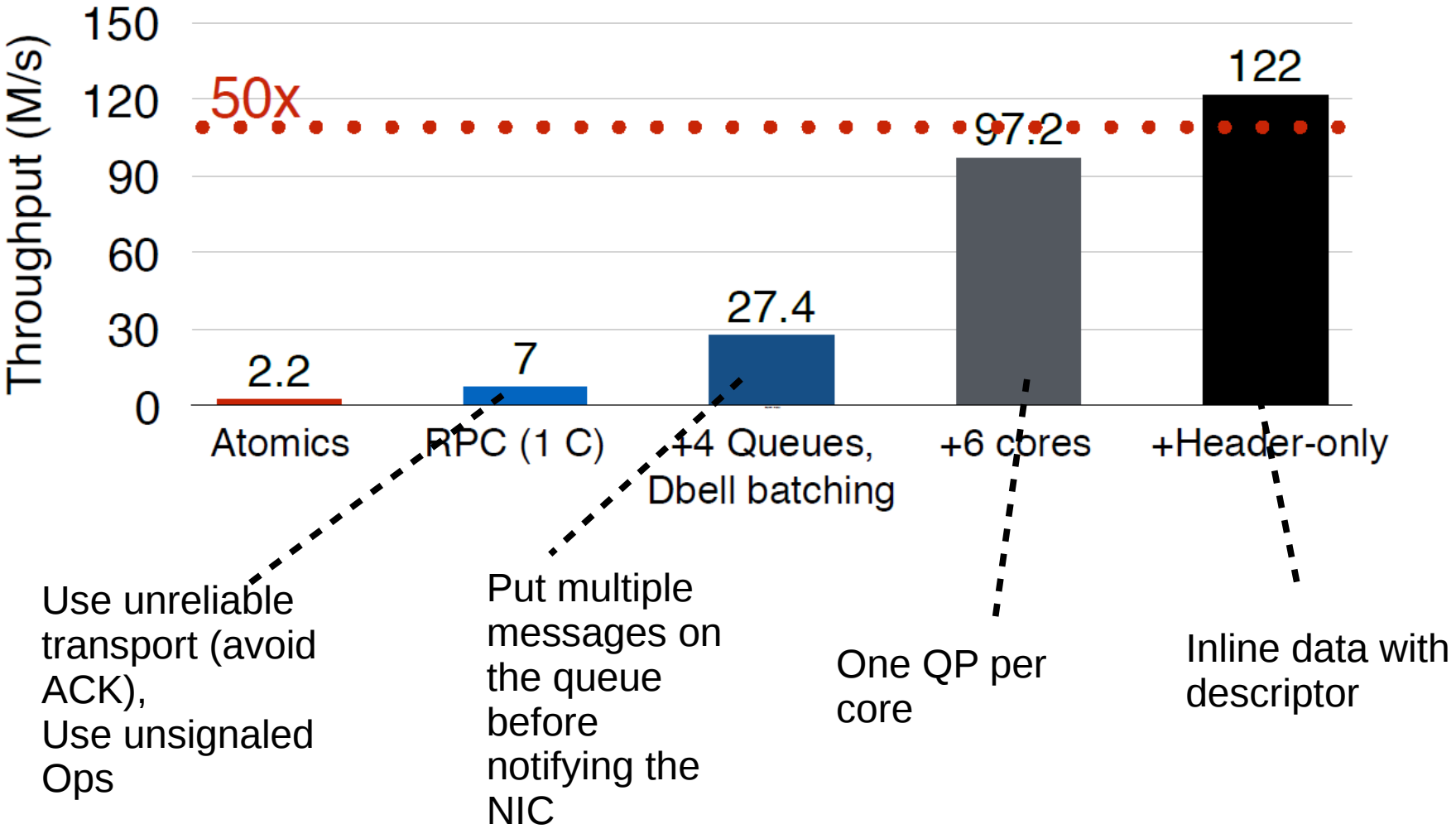
How to Design a Sequencer Service



Sequencer Throughput



Sequencer Throughput



Summary & Open Questions

- NICs getting faster, CPUs are not
  - Multiple tx/rx queues
  - Sophisticated mux/demux filters
  - Offload engines
  - Kernel bypass
- Open questions:
  - How to program closely-coupled set of RDMA devices? Requires a new way to think about software construction.
    - See Crail: www.crail.io
  - How to manage the sheer complexity and diversity of NICs?
  - What new architectures are required as bottlenecks move from NIC to PCI? (e.g., Intel OmniPath, Direct Cache Access, etc)