Networking Operating System from Scratch towards High-Performance COTS Network Facilities

Hirochika Asai <panda@hongo.wide.ad.jp>
The University of Tokyo

Invited Talk @ SFC, Keio University
July 6, 2015

In "Design and Implementation of System Software"

BRIEF INTRODUCTION TO NETWORKING OPERATING SYSTEM

Background: Trends on Network Functionalities by Software on COTS hardware

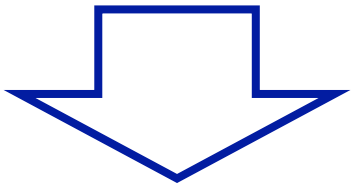
- Commercial Off-The-Shelf (COTS)
  - Inexpensive
  - Flexible
  - Extensible

- Related technologies
  - SDN: Software Defined Network
    - Separation of
      - Forwarding Plane; by hardware
      - Control Plane; by software
  - NFV: Network Function Virtualization
    - Network function by software with virtualization technologies (e.g., virtual machine, container, process)

“Networking” “Operating System”?

"Networking" Operating System

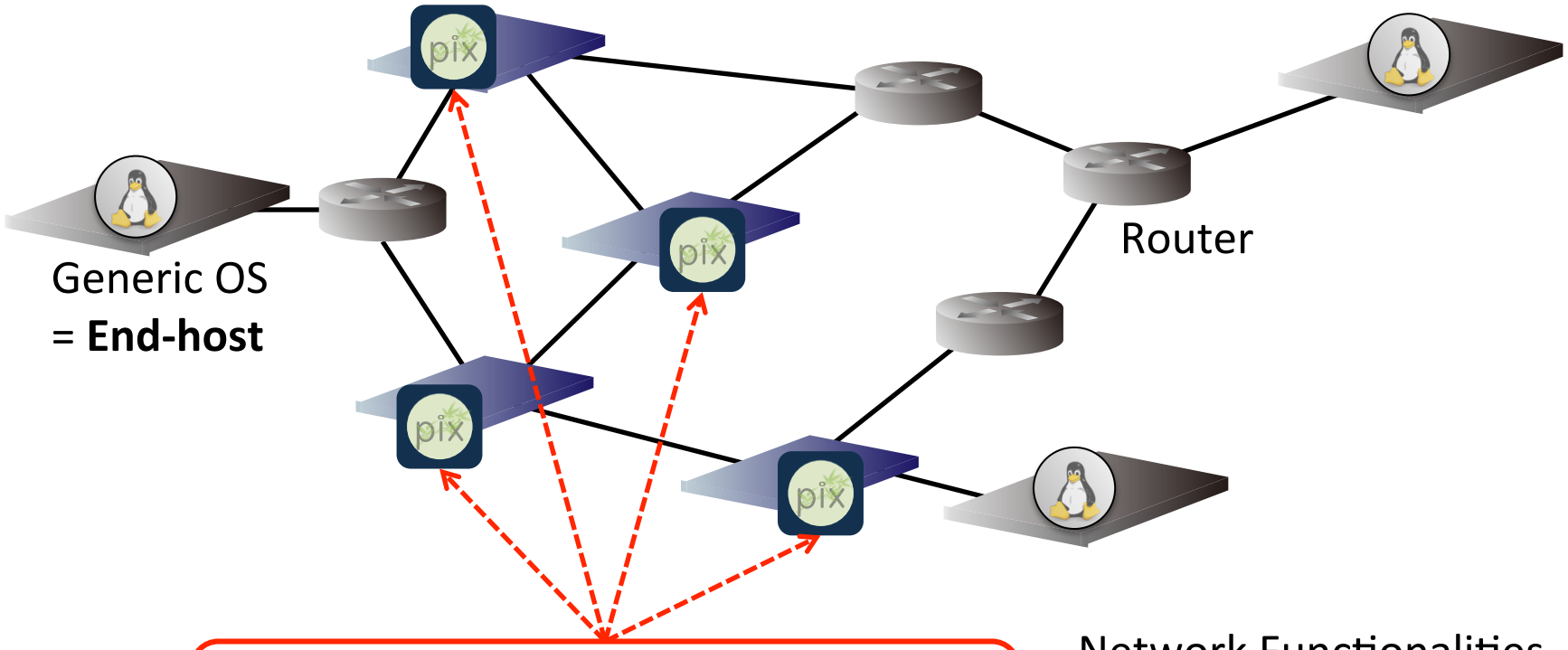
"**Networked**" (or "**Network**") Operating System
manages "**computing**" resources *over* a "**network**".
= Distributed Operating System



"**Networking**" Operating System
manages "**computing**" *and* "**network**" resources
for "**network functionalities**".

N.B., "Networking Operating System" may not have been a common phrase yet.

A Target Environment of Networking Operating Systems



Networking Operating Systems
(as intermediate nodes)

= FIFO

Network Functionalities

- IP Router
- Firewall
- Load balancer
- etc.

Revisit: What is an Operating System?

- Extended Machine
  - Abstraction of underlying hardware
    - File system
    - Timer and Interrupts
    - Character/block devices
    - Memory management (virtual memory / segmentation)
- Resource Manager
  - Resource sharing (spatio-temporal)
    - CPU
    - Memory
    - Peripheral devices (e.g., disks, network interfaces)
    - I/O

What is a "Networking" Operating System?

- Extended Machine
  - Abstraction of underlying hardware
    - File system
    - Timer and Interrupts
    - Character/block devices
    - Memory management (virtual memory / segmentation)
    - Network port (interface)
    - Routing/Forwarding engine
- Resource Manager
  - Resource sharing (spatio-temporal)
    - CPU
    - Memory (e.g., packet buffer)
    - Peripheral devices (e.g., disks, network interfaces)
    - I/O

Abstraction of Network Resources in Linux Operating System

- Network ports
  - Abstracted as character devices (in MINIX)
  - Interfaces
    - I/O: read/write system calls
    - Control: ioctl system call (in POSIX) / Netlink (in Linux)
- Routing/Forwarding engines
  - No abstraction
  - Interfaces
    - Routing sockets (in BSD)
    - Netlink (in Linux)

Not good performance, then need a new design

Design and Implementation of Networking Operating System

- **Goal**: Operating system (and library) to run high-performance networking functionalities as its applications
  - Router: IPv4/IPv6 router
  - Middlebox: Firewall, Load-balancer, etc.
  - Server apps: HTTP, Authentication, Accounting, etc.

DESIGN OF NETWORKING OPERATING SYSTEM

Characteristics of Networking Operating System

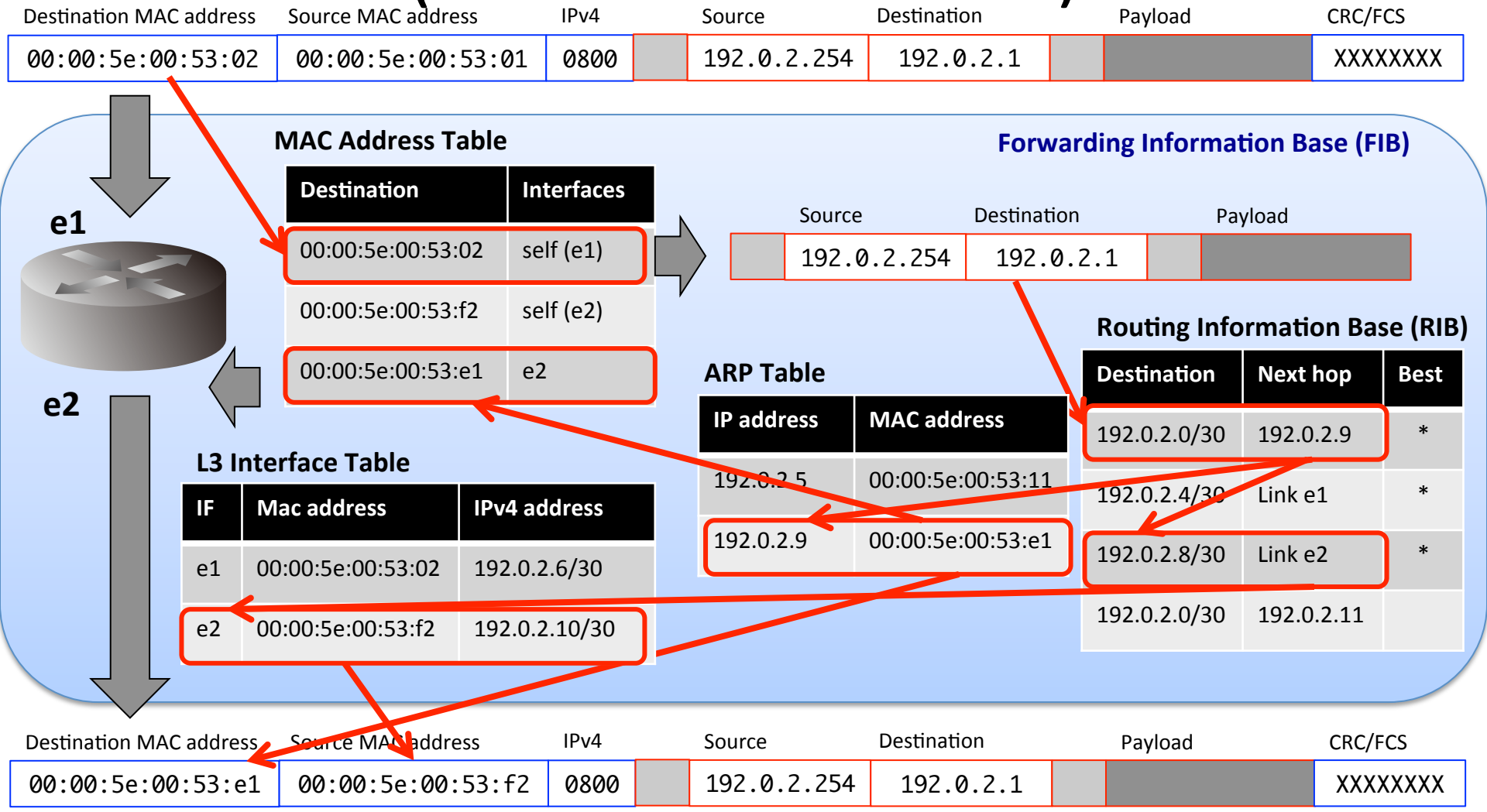
- Applications
  - App. 1) A single core application
    - Ethernet forwarding, IP routing

Parallelization on multiple CPU cores

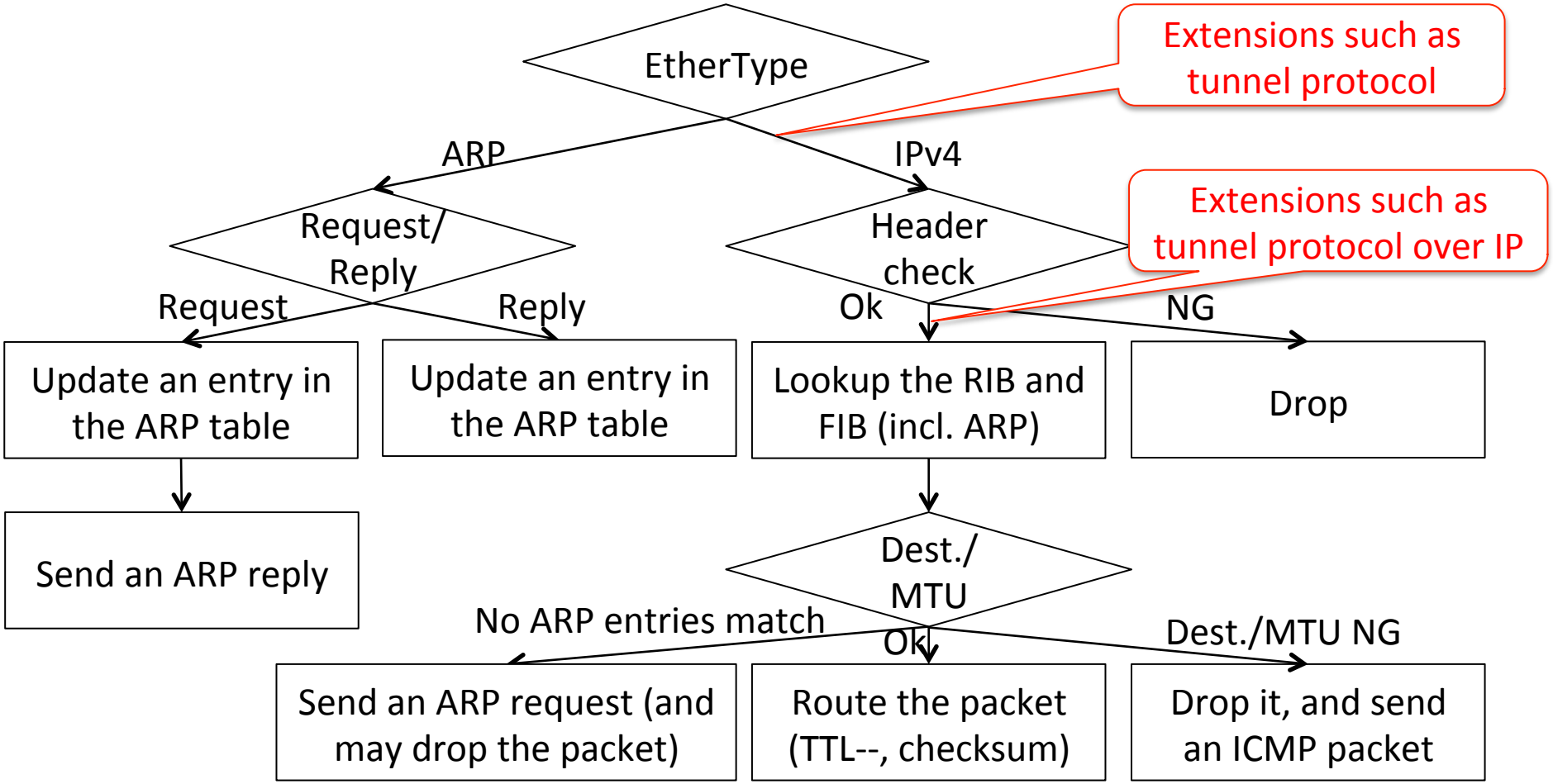
  - App. 2) A few extensions to the core application
    - Packet processing
      - IPsec, Tunneling protocol
  - App. 3) A few dedicated applications
    - L3/L4-L7 functionalities
      - (IPsec), HTTP load balancer
  - App. 4) Many control applications
    - Control/Routing protocol
      - Shell, SSH, OSPF, BGP

Timesharing multiprogramming (like a generic OS)

Revisit: IPv4 Routing (IPv4 over Ethernet)



Revisit: What do IPv4 routers do?



Functional Requirements

- Abstraction for applications
  - Network ports
  - Forwarding/routing engines (FE/RE)
- Mechanism
  - App. 1) In-order I/O for inter-port/application communication
    - Packet reordering avoidance
  - App. 2) Extensible FE/RE (modular applications)
    - e.g., Tunneling protocol
  - App. 3) Exclusive resource allocation of applications (tickless applications)

Performance Requirements

- High throughput
  - e.g., 100 Gbps+ for forwarding/routing
- High packet rate
  - e.g., 100 Mpps+ for forwarding/routing

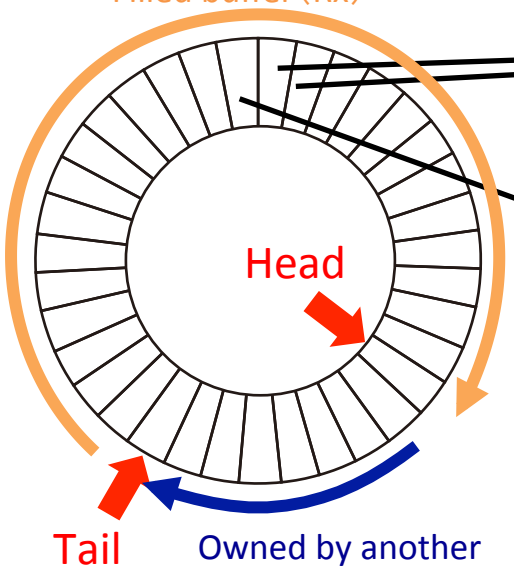
Functional Requirements

- Abstraction for applications
  - Network ports
  - Forwarding/routing engines (FE/RE)
- Mechanism
  - In-order I/O for inter-port/application communication
    - Packet reordering avoidance
  - Extensible FE/RE (modular applications)
  - Exclusive resource allocation of applications (tickless applications)

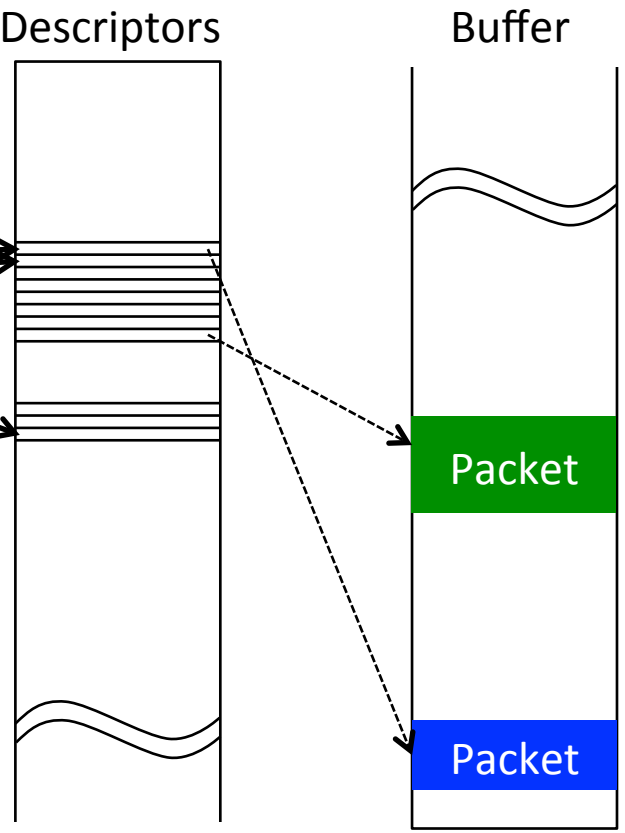
Use of "**Ring Buffer**" over a new API set

Ring Buffer

Ring buffer:
An intuitive representation



Mapping on the memory



= Meta info.
+ pointer to a packet

= packet

Management data structure for each ring buffer

- Ring buffer
  - Descriptor array (contiguous)
- Head pointer (Read-only from myself)
  - Proceeded by another
- Tail pointer (Read-only from another)
  - Proceeded by myself

Exclusive Inter-processor Messaging with Ring Buffers

Producer of CPU #X

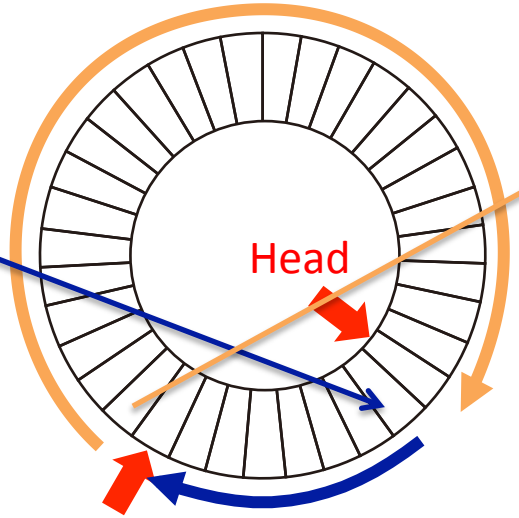
Ring buffer for CPU #X at CPU #Y

Consumer of CPU #Y

Owned by myself (CPU #Y)
- Vacant buffer（Tx）
- Filled buffer（Rx）

Message 1

Produce: Put the message and proceed the head when completed.

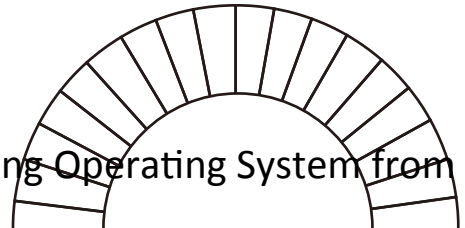


Message 2

Consume: Take the message and proceed the tail when released.

Owned by another (CPU #X)
- Queued buffer（Tx）
- Vacant buffer（Rx）

Ring buffer for CPU #X at CPU #Z



* A ring buffer is assumed to be owned by the consumer side.

Functional Requirements

- Abstraction for applications
  - Network ports
  - Forwarding/routing engines (FE/RE)
- Mechanism
  - In-order I/O for inter-port/application communication
    - Packet reordering avoidance
  - Extensible FE/RE (modular applications)
  - Exclusive resource allocation of applications (tickless applications)

Discussion on Extensions in FE/RE: Process, Thread, Fiber, and Subroutine

	Memory	Context	Multitasking	Performance	Protection
Process	Isolated	Isolated	Preemptive	Worst	Good
Thread	Shared	Isolated	Preemptive	Bad	Context-only
Fiber/Co-routine	Shared	Isolated	Co-operative	Good	Context-only
Subroutine	Inherit	Inherit	N/A	Best	None

Q. Is performance important?

A. *Yes, because "packet" forwarding/routing = A lot of small (short) function calls.*

Q. Is the context-level protection needed?

A. *Yes, in order to avoid a fatal crash because an extension module may contain a bug.*

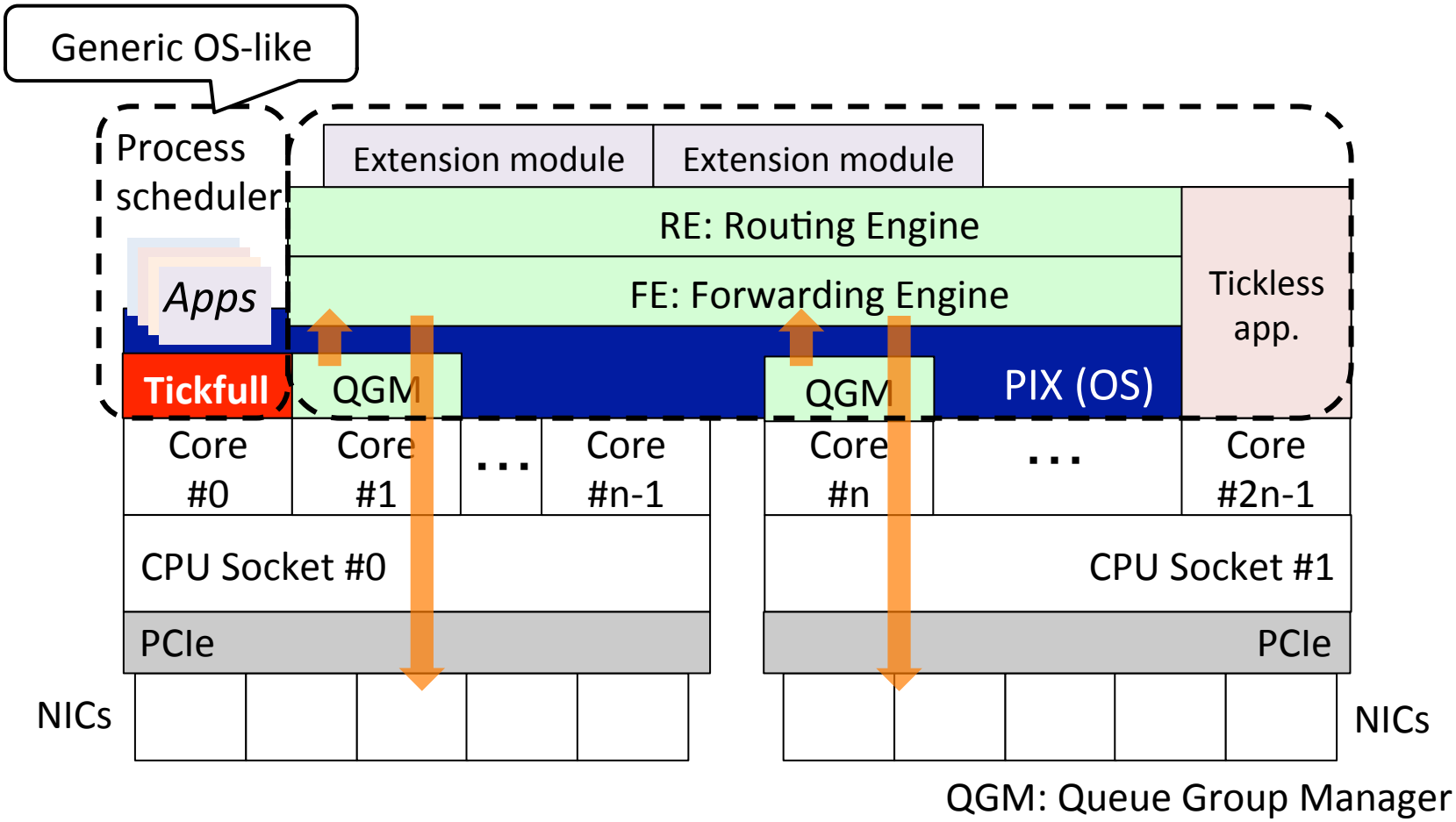
Modular applications extended to FE/RE

➔ Use Fiber/Co-routine approach

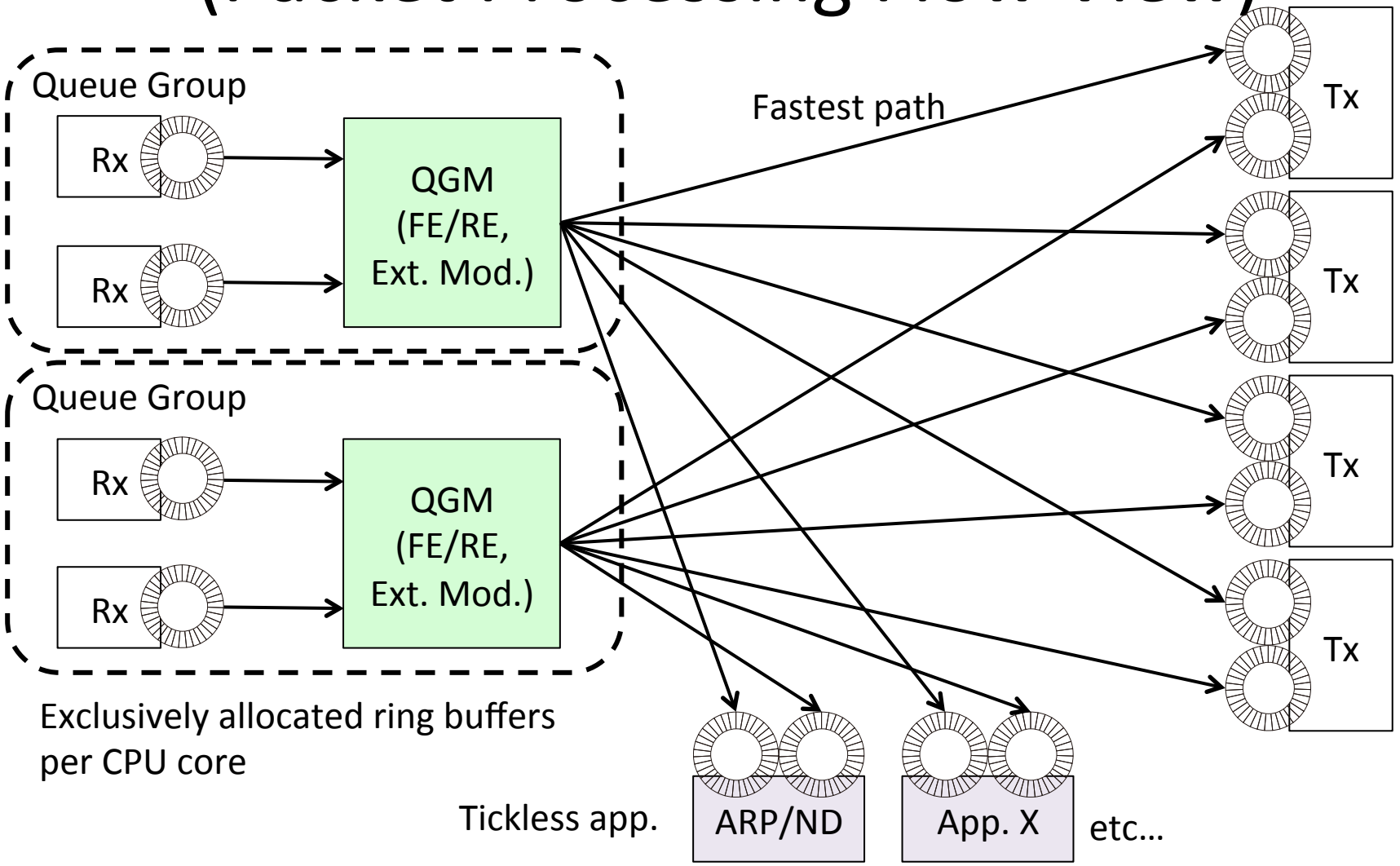
Functional Requirements

- Abstraction for applications
  - Network ports
  - Forwarding/routing engines (FE/RE)
- Mechanism
  - In-order I/O for inter-port/application communication
    - Packet reordering avoidance
  - Extensible FE/RE (modular applications)
  - Exclusive resource allocation of applications (tickless applications) ⇦ No scheduler (timesharing)

Architecture of PIX (Programming Stack View)



Architecture of PIX (Packet Processing Flow View)



APIs for User Programming

- Basically, plan to follow POSIX/socket APIs, but need some extensions
  - Socket API to handle Ethernet packets
    - SOCK_DGRAM + AF_ETHER
      - Bind to a MAC address
  - Zero-copy & packet-based (non-stream) recv/send read/write
    - Ring buffer (descriptor) based APIs
      - Skip the detail in this talk…

Summary of the Design Features

- Ring buffer-based
  - network port abstraction
  - inter-port/application communication
- Fiber-based modular applications for the extensions to FE/RE
- Tickless applications (process/thread)

IMPLEMENTATION OF NETWORKING OPERATING SYSTEM

Networking Operating System from Scratch

- *Why "from scratch"?*
  1. It's fun ☺
  2. No more blackbox!
     - I don't like "guess in the blackbox"; e.g., it slows because of the overhead of context switches, perhaps.
       - Hey, why don't you try it without context switches and compare the results?
  3. (IMO) Too complicated and difficult to implement tickless process in the existing operating systems
  4. Need to control as much as possible (e.g., cache as discussed later)

Before Going to the Implementation

- *"Implementation techniques are more complicated than you have learned from the lectures."*
  - In your books (focusing on theoretical things);
    - Scheduling strategy (Completely Fair Scheduler, etc.)
    - Memory allocation algorithm (Buddy system, slab, etc.)
    - Filesystem data structure (B+ tree, etc.)
  - In reality;
    - BIOS/UEFI, ACPI, APIC, and other specs. steal our time
    - Context switch and fork-exec are a headache...
    - Heterogeneous memory access
      - Complicated cache behavior
      - Non Uniform Memory Access (NUMA)
    - Superscalar processor
    - Hyperthreading
    - etc.

Contents of Implementation

- Bootstrap of x86 (multiprocessor)
- Timer (tick)
- Context switch
- Memory protection (page table)
- System calls
- Fork-exec [work in progress…]
- Queue Group Manager (+NIC driver)

Documents are in public!

- CPU・OS
  - Intel® 64 and IA-32 Architectures Software Developer's Manual
  - Intel® 64 and IA-32 Architectures Optimization Reference Manual
  - Advanced Configuration and Power Interface Specification
- NIC
  - Intel® Ethernet Controller XL710 Datasheet
  - Intel® 82599 10 GbE Controller Datasheet

Practices

- If you are interested in developing your operating system from scratch, please visit
  - Lecture note (in progress): http://aos.jar.jp/
  - Source code: https://github.com/drpnd/aos/

Contents of Implementation

- Bootstrap of x86 (multiprocessor)
- Timer (tick)
- Context switch
- Memory protection (page table)
- System calls
- Fork-exec [work in progress…]
- Queue Group Manager (+NIC driver)

IMPLEMENTATION ISSUE 1/2: DARK MATTER IN CPU TIME

Dark Matter in CPU Time

Q. How long does the following underlined instruction take? (*)

```
mov (tx_tail),%eax
mov $NIC_MMIO_BASE,%rbx          # Just write the tail pointer of the NIC's
mov %eax,NIC_TX_TAIL(%rbx)       # ring buffer, %eax, to the PCIe NIC
```

(*)
CPU: Intel Core i7 4770K (Haswell)
Memory: Corsair DDR3-1866 8GB x4
NIC: Intel X520-DA2 (2 port 10GbE)

Dark Matter in CPU Time

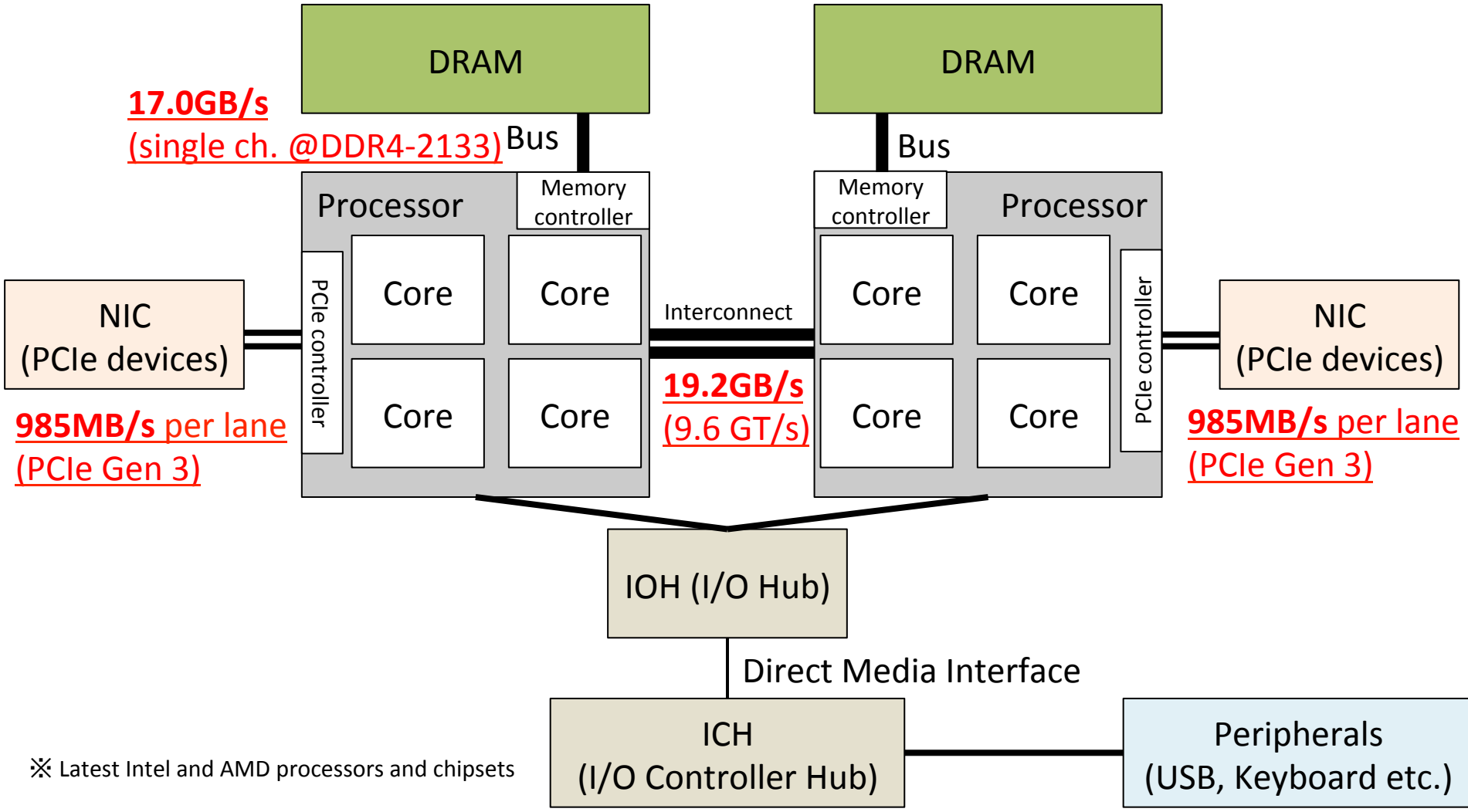
Q. How long does the following underlined instruction take? (*)

```
mov (tx_tail),%eax
mov $NIC_MMIO_BASE,%rbx          # Just write the tail pointer of the NIC's
mov %eax,NIC_TX_TAIL(%rbx)       # ring buffer, %eax, to the PCIe NIC
```

A. More than 70 ns. (= 250+ CPU cycles!)

(*)
CPU: Intel Core i7 4770K (Haswell)
Memory: Corsair DDR3-1866 8GB x4
NIC: Intel X520-DA2 (2 port 10GbE)

Take a look at “I/O resources”



※ Latest Intel and AMD processors and chipsets

Take a look at "I/O resources"

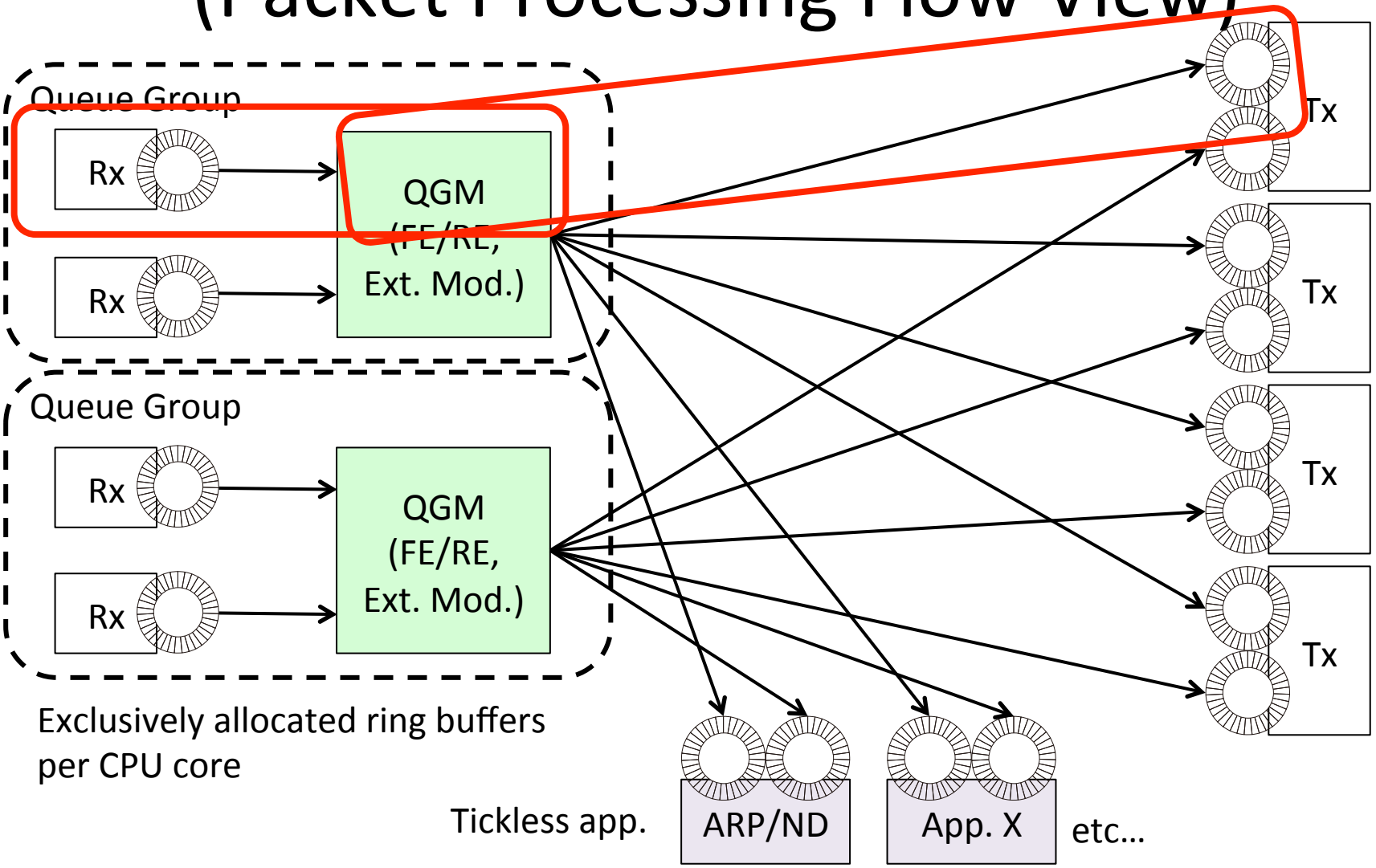
- Data access latency
  - L1 cache: 4-5 cycles (~1 ns)
  - L2 cache: 12 cycles (~3 ns)
  - L3 cache: 27.85 cycles (~7 ns)
  - DRAM: ~65 ns
- PCIe memory mapped I/O access latency(※)
  - **Read**
    - 1529.17 cycles / read
    - 392.1 ns / read
  - **Write**
    - 282.621 cycles / write
    - 72.47 ns / write

Ref. Maximum packet rate in Ethernet
10 GbE: 14.88 Mpps = 67 ns/packet
40 GbE: 59.52 Mpps = 17 ns/packet
100 GbE: 148.8 Mpps = 6.7 ns/packet

(Measurement host)
CPU: Intel Core i7 4770K (Haswell)
Memory: Corsair DDR3-1866 8GB x4
NIC: Intel X520-DA2 (2 port 10GbE)

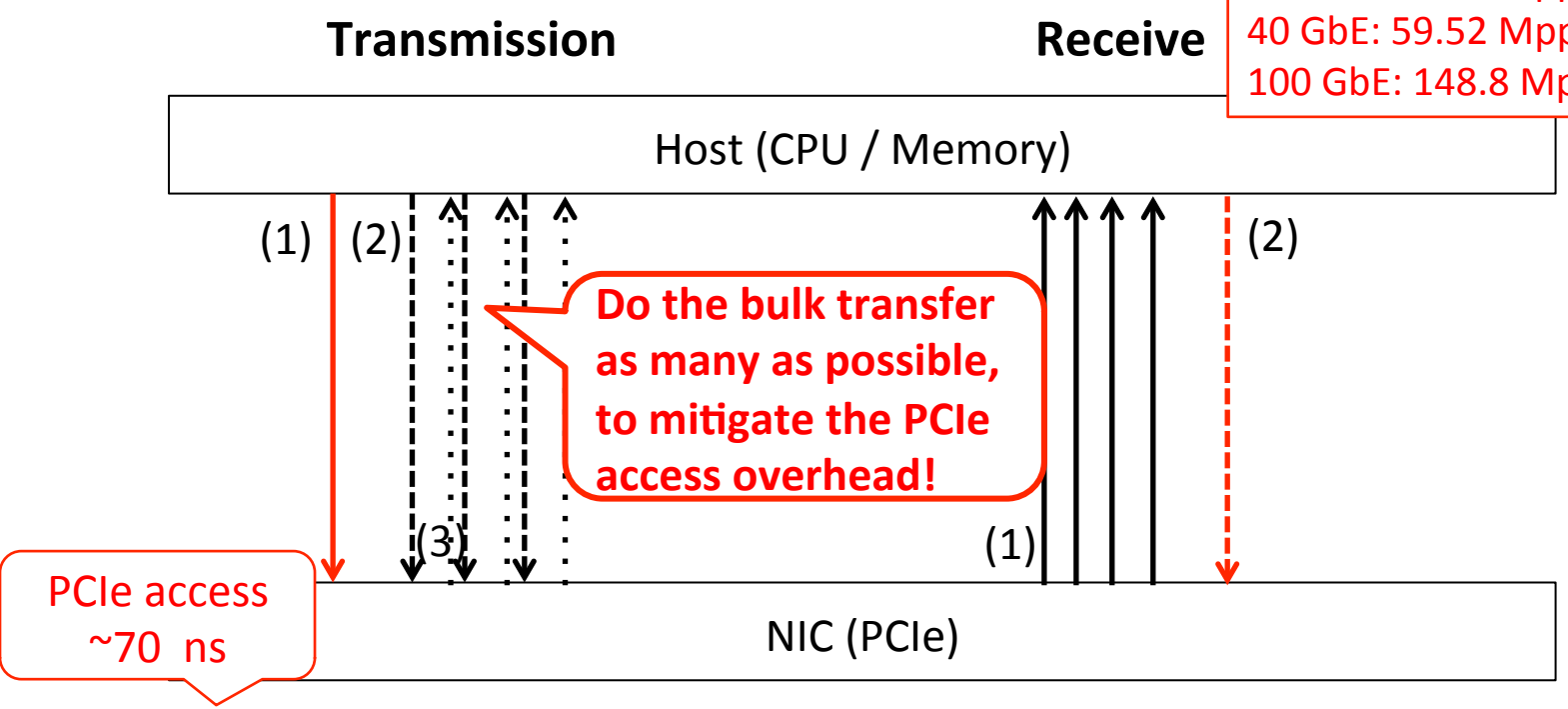
※Measured CPU cycles to access to the same register
1 million times by Performance Monitoring Counter (PMC)

Architecture of PIX (Packet Processing Flow View)



NIC Operations (Intel® 10/40 GbE)

Line-rate
10 GbE: 14.88 Mpps = 67 ns/packet
40 GbE: 59.52 Mpps = 17 ns/packet
100 GbE: 148.8 Mpps = 6.7 ns/packet

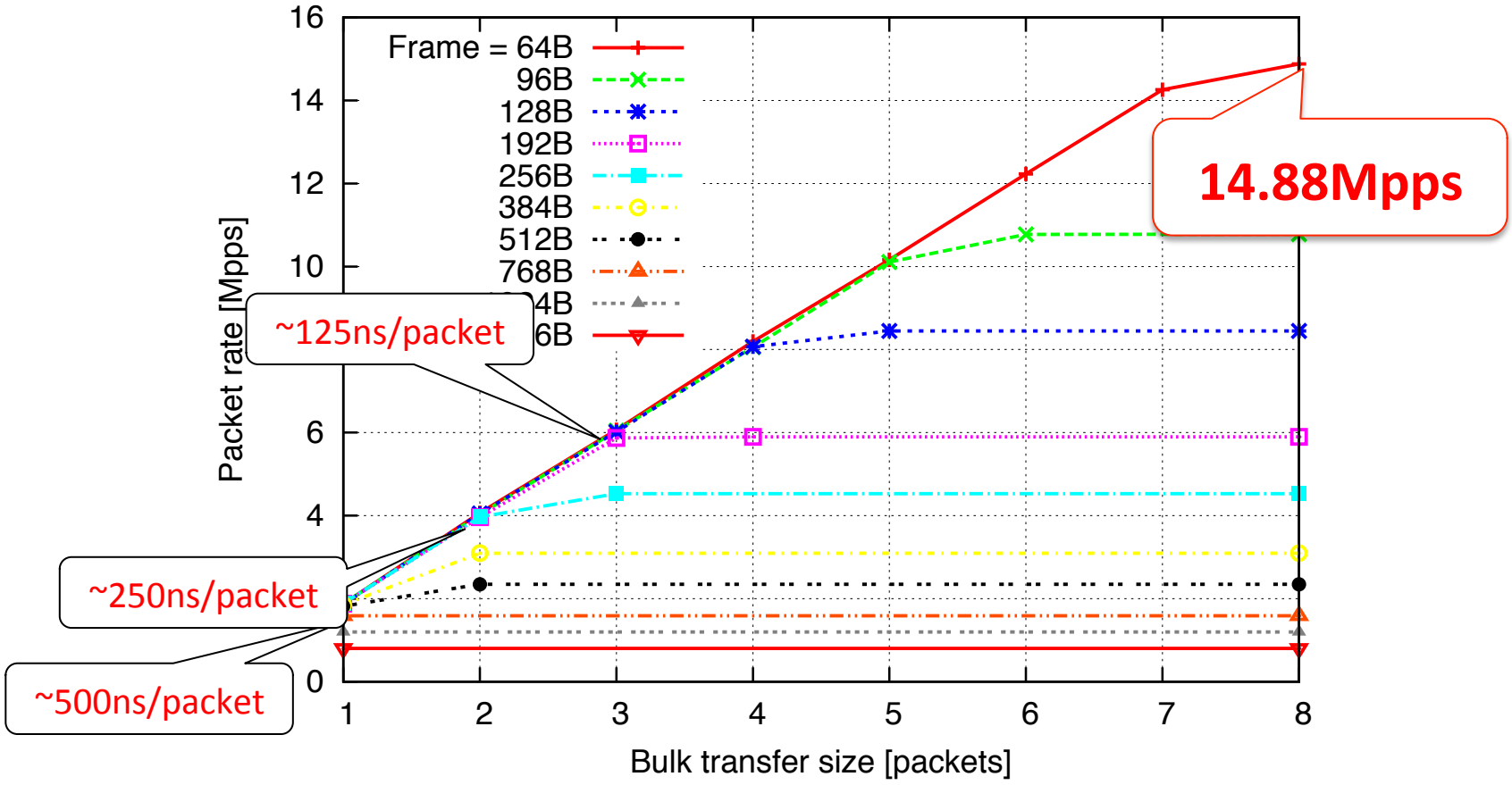


(1) Proceed the tail pointer
(2) Packet transmission via DMA
(3) Write-back a completion flag to the descriptor entry

(1) Packet transmission via DMA and write a completion flag to the corresponding descriptor entry
(2) Proceed the tail pointer (to free)

PCIe access ~70 ns

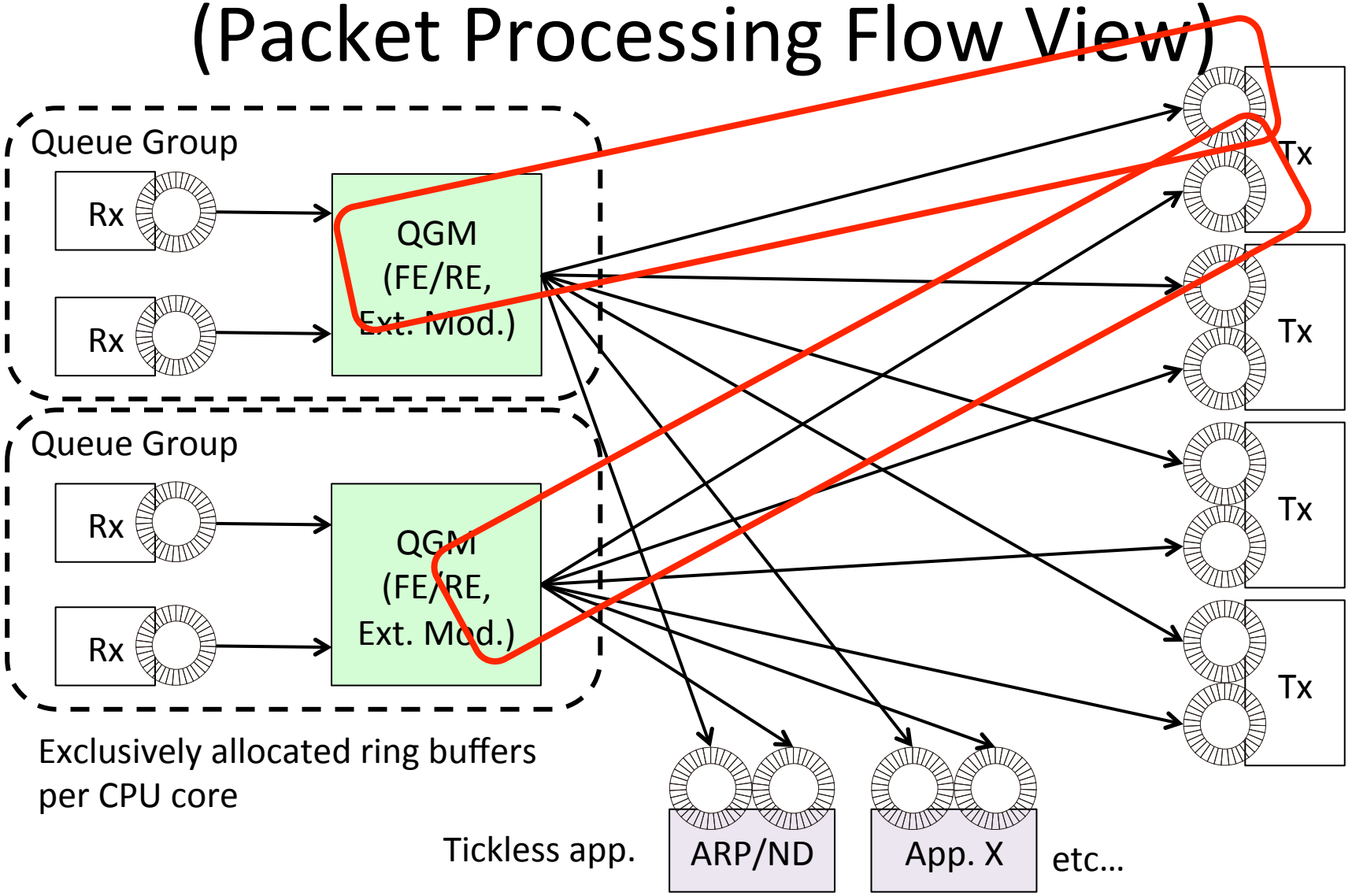
Tx Performance by bulk size



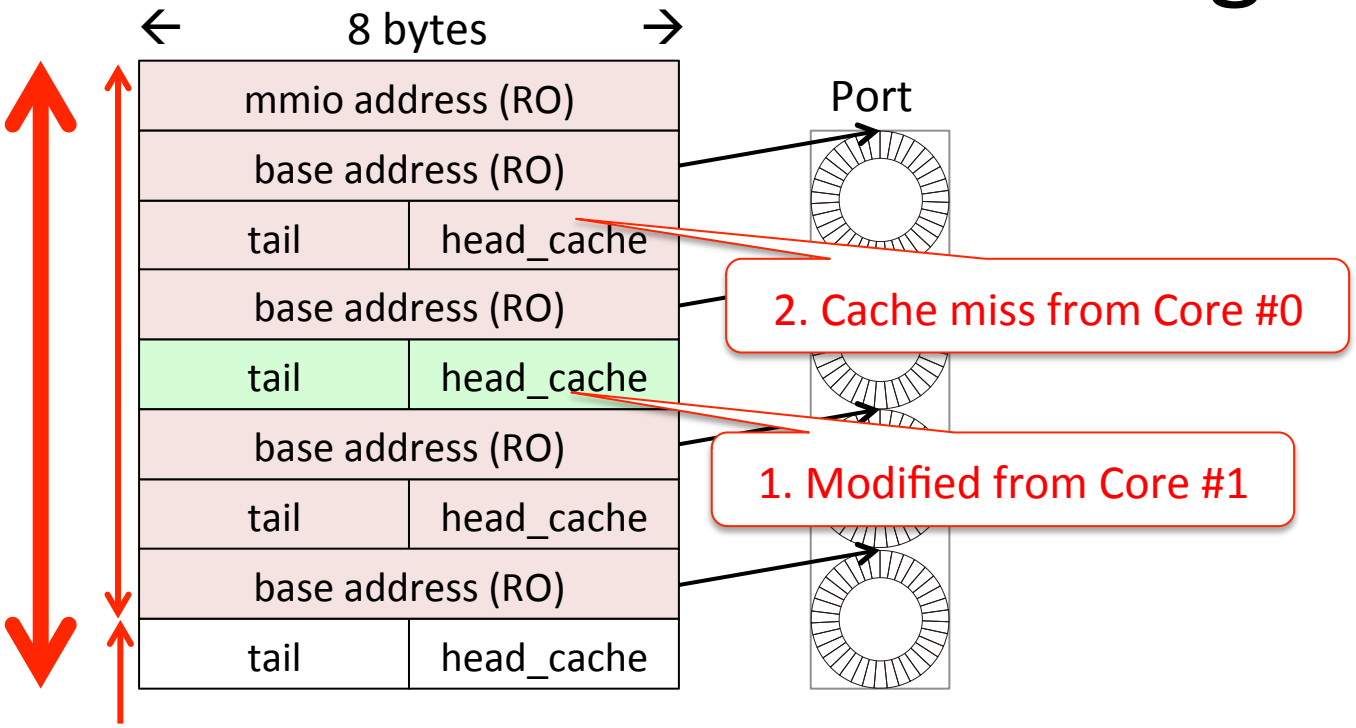
Note: Also confirmed 59.52 Mpps Tx (2 Intel® X520-DA2) @ 1 core from Intel® Core i7-4770K

IMPLEMENTATION ISSUE 2/2: CACHE COHERENT

Architecture of PIX (Packet Processing Flow View)

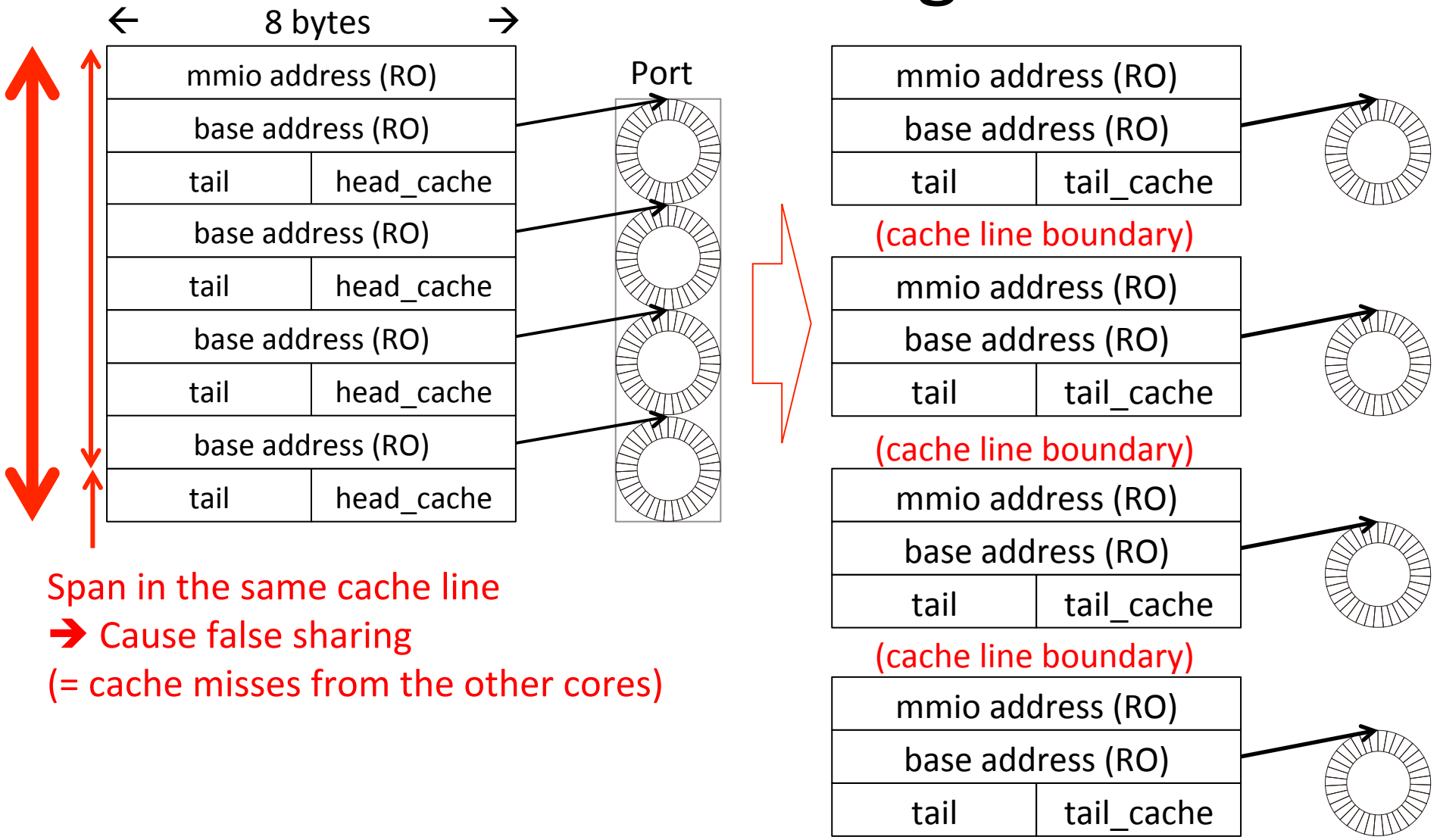


False Sharing



Span in the same cache line
➔ Cause false sharing
(= cache misses from the other cores)

False Sharing



Span in the same cache line
➔ Cause false sharing
(= cache misses from the other cores)

Performance Degradation by False Sharing

	w/ False Sharing	w/o False Sharing
8 port IPv4 routing (random destination)	11 Mpps	55 Mpps

Intel® Xeon E5-2670 x2
Intel® X520-DA2 x4 (8 port)
QGM = 8 (1 QGM per port)
Queue length per ring buffer = 256

Gaps between Design and Implementation

- OS designers need to know hardware (as well as theories)
  - Restrictions
    - e.g., Memory protection: Segmentation? Page table? Key protection?
      - sbrk() and brk() are designed for segmentation
  - Characteristics and Dependencies
    - e.g., Memory: Cache? DRAM? Swap?
    - e.g., Timer: Local APIC? ACPI, HPET?
- OS implementers need to understand the design (as well as hardware and programming)
  - Abstraction
  - Separation of mechanisms and policies (one of UNIX philosophies)
    - Structure of data structures and APIs

Summary

- Networking Operating System
  - Design
    - Ring buffer-based messaging
    - Fiber-based extensible modules
    - Tickless applications
  - Implementation
    - Presented two interesting issues
- Enjoy the design and implementation of your own software