Programming with Socket API Part I

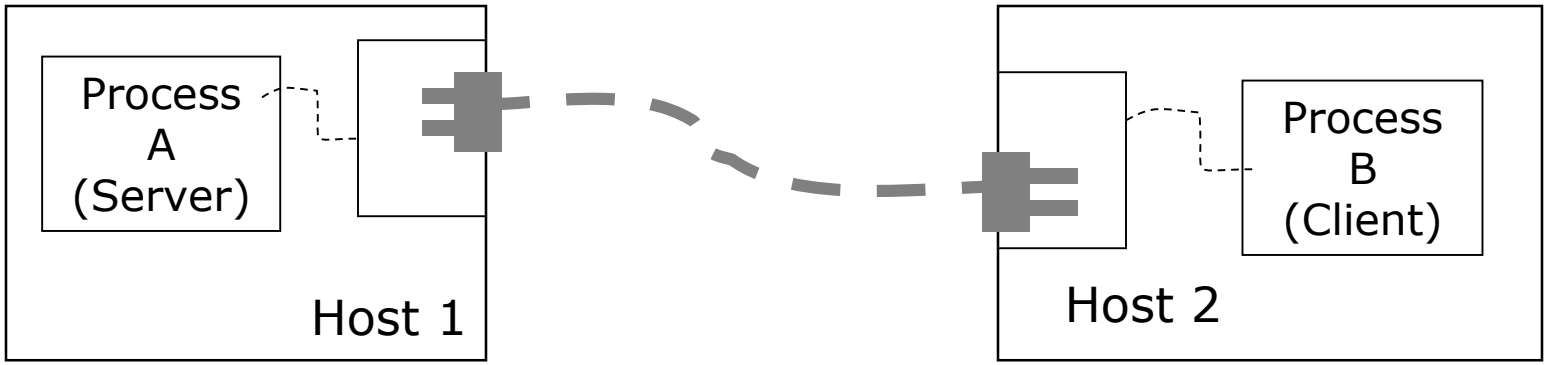
Hui Chen, Ph.D.
Computer Science
Dept. of Math & Computer Science
Virginia State University
Petersburg, VA 23806

Outline

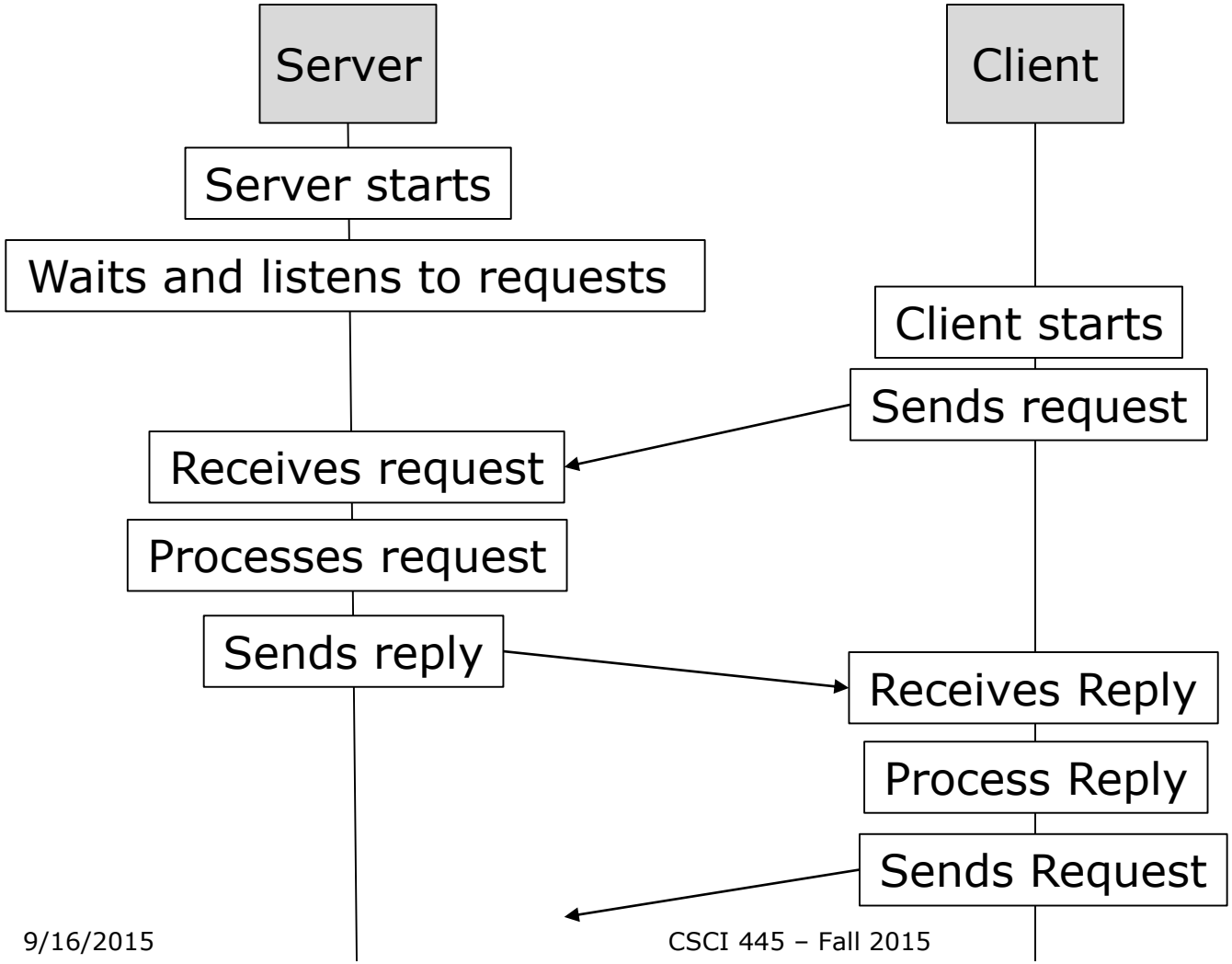
- Overview of network application architecture
- Programming and experimentation environment
- Ethernet implementation in practice
- Berkeley sockets for programming Ethernet

Network Application

- At least two processes
  - *Server* logic: listening to client's requests
  - *Client* logic: sending request to server
- Example setup
  - Process A: server logic
  - Process B: client logic



Server and Client Interaction: An Example



Connectionless and Connection-Oriented

- Network applications or protocols can follow either one of the two communication modes
- Connectionless communication
  - Does not require to *establish a connection* before transmitting data
- Connection-oriented communication
  - Requires to *establish a connection* before transmitting data

Connection-Oriented

- Setting up a connection
  - Determine whether there is a communication path between the two communication parties
  - Reserve network resources
- Transmitting and receiving data
- Tearing down the connection

Choosing between Connected-Oriented and Connectionless

- Consider application requirement and decide which one works best for the application*
  - How reliable must the connection be?
  - Must the data arrive in the same order as it was sent?
  - Must the connection be able to handle duplicate data packets?
  - Must the connection have flow control?
  - Must the connection acknowledge the messages it receives?
  - What kind of service can the application live with?
  - What level of performance is required?
- If reliability is paramount, then connection-oriented transport services (COTS) is the better choice.

*From *Transport Interfaces Programming Guide, SunSoft, 1995*

Client-Server and Peer-to-Peer Models

Client-Server Model

- Server process
  - Runs server logic
  - Providing service
  - Passively waiting: listening to client requests
  - Processing client requests
- Client process
  - Runs client logic
  - Actively requesting service from server

Peer-to-Peer Model

- Any of the communicating processes contains both server and client logics
- Each processes listens to requests from other clients
- Each processes sends requests or serves requests

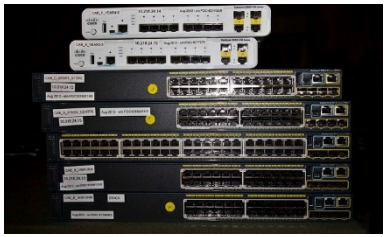
Network Programming

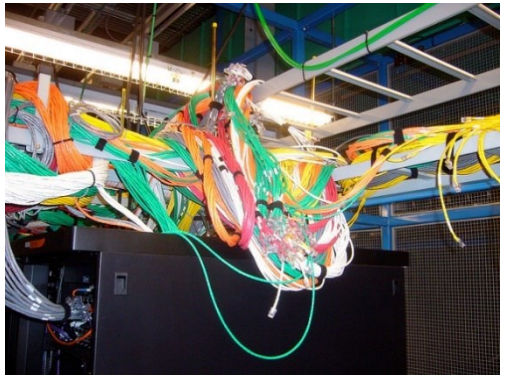
- Example programs using a client-server model
  - Write two programs (A, B)
    - Program A contains and runs the server logic
    - Program B contains and runs the client logic

Experiment Environment

- Use multiple Linux virtual machines

Ethernet: Where Are They?





Ethernet: Where Are They?

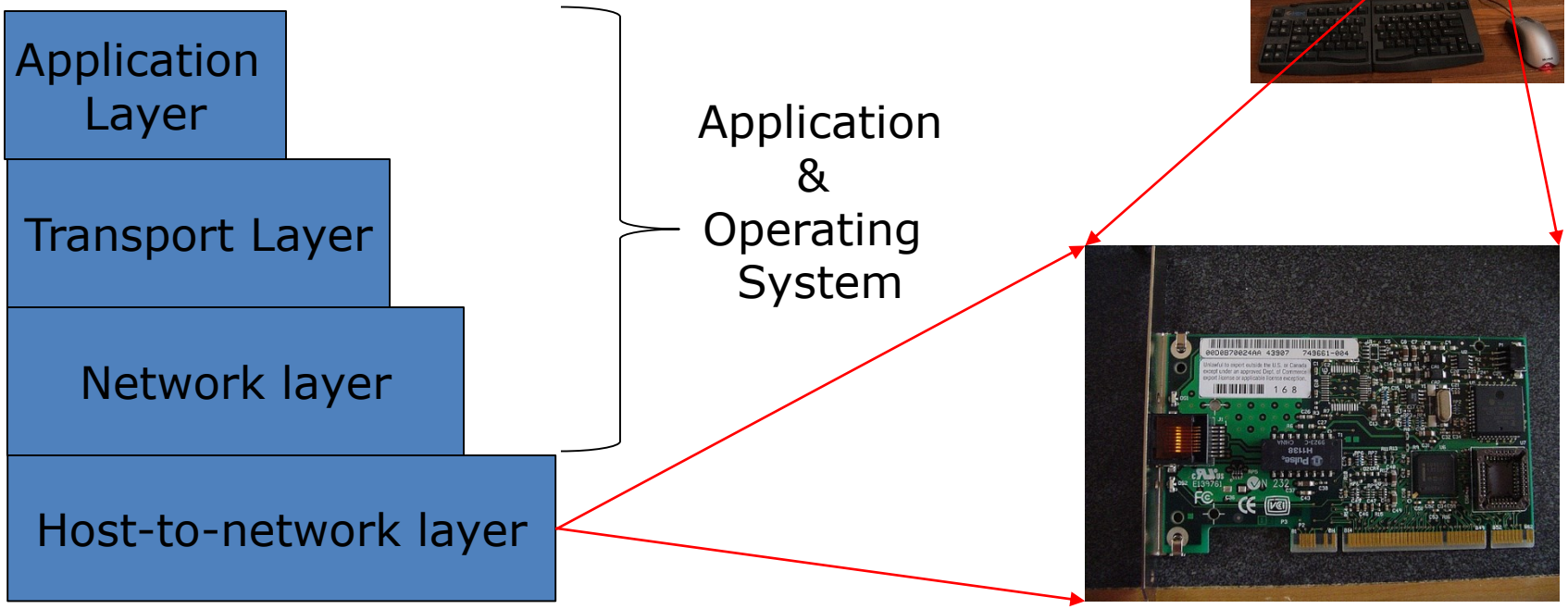
- Ethernet Adapter

Ethernet: Where Are They?

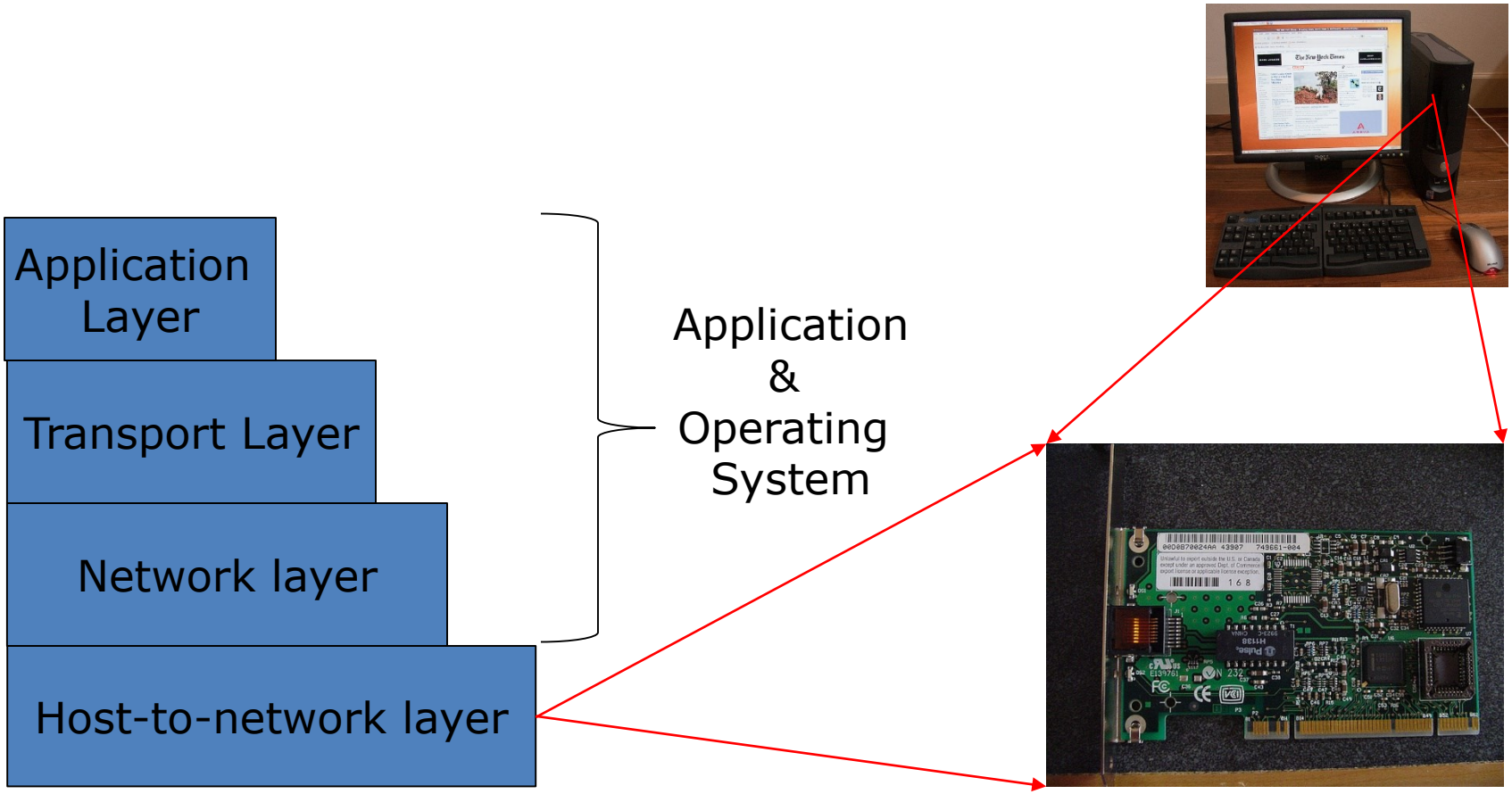
- Beside hardware, firmware inside
  - Example
    - Encoding
    - Error Detection
    - Medium Access Control (CSMA/CD)

Ethernet: Upper Layer Protocol Design and Programming

- How to access functionality of Ethernet adapter?

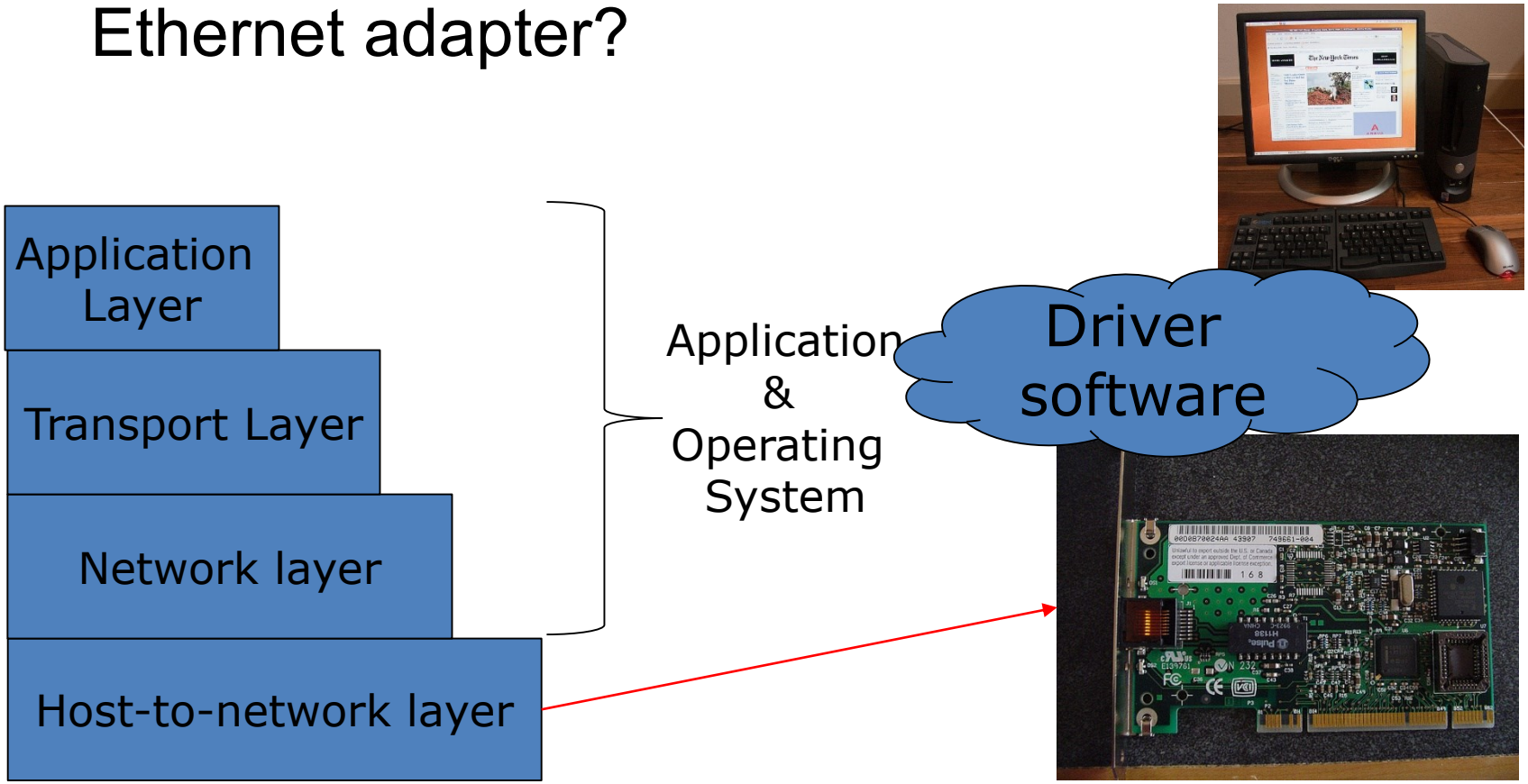


Ethernet: Upper Layer Protocol Design and Programming



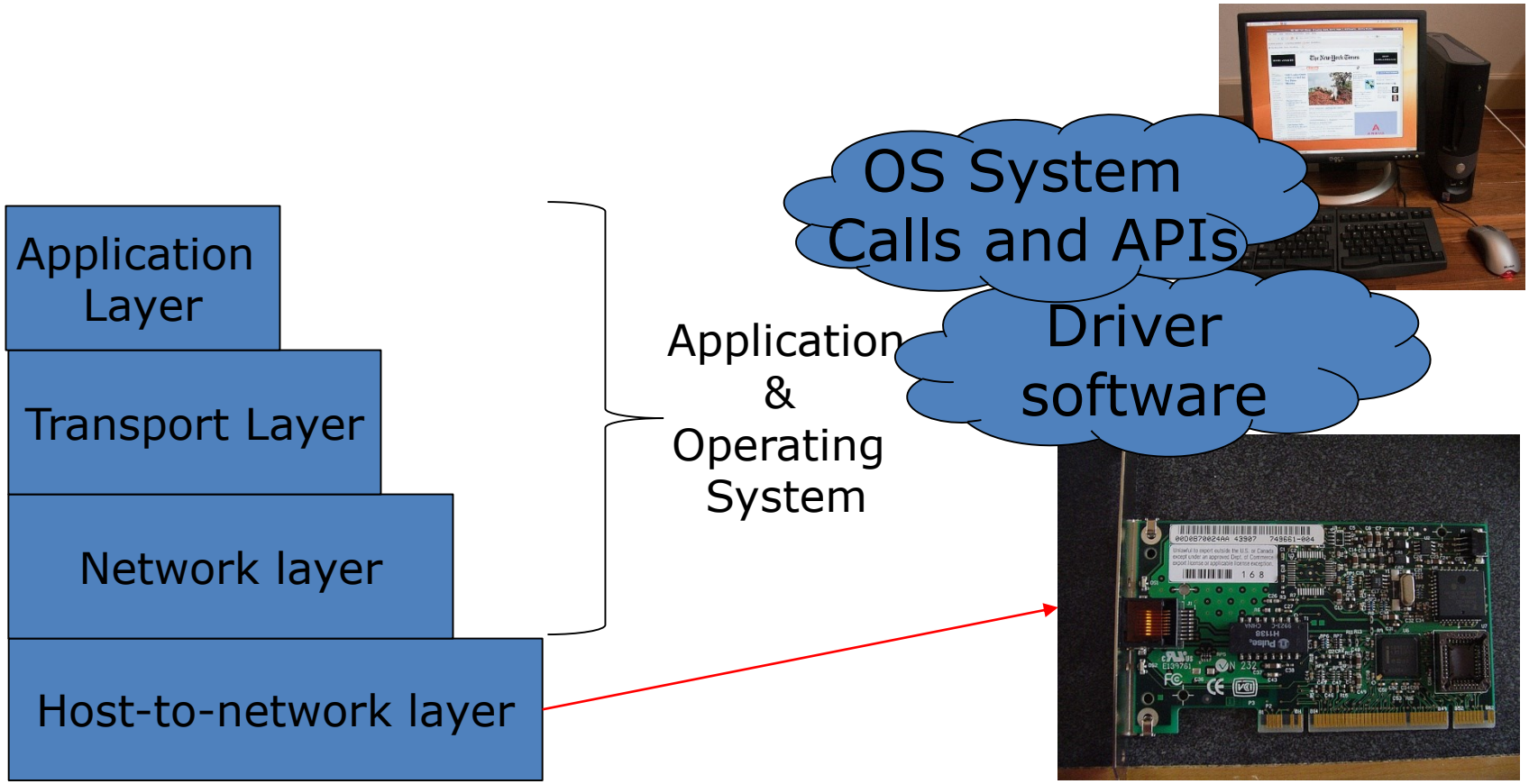
Ethernet: Upper Layer Protocol Design and Programming

- How to access functionality of Ethernet adapter?



Ethernet: Upper Layer Protocol Design and Programming

- How to access functionality of Ethernet adapter?



Programming Ethernet

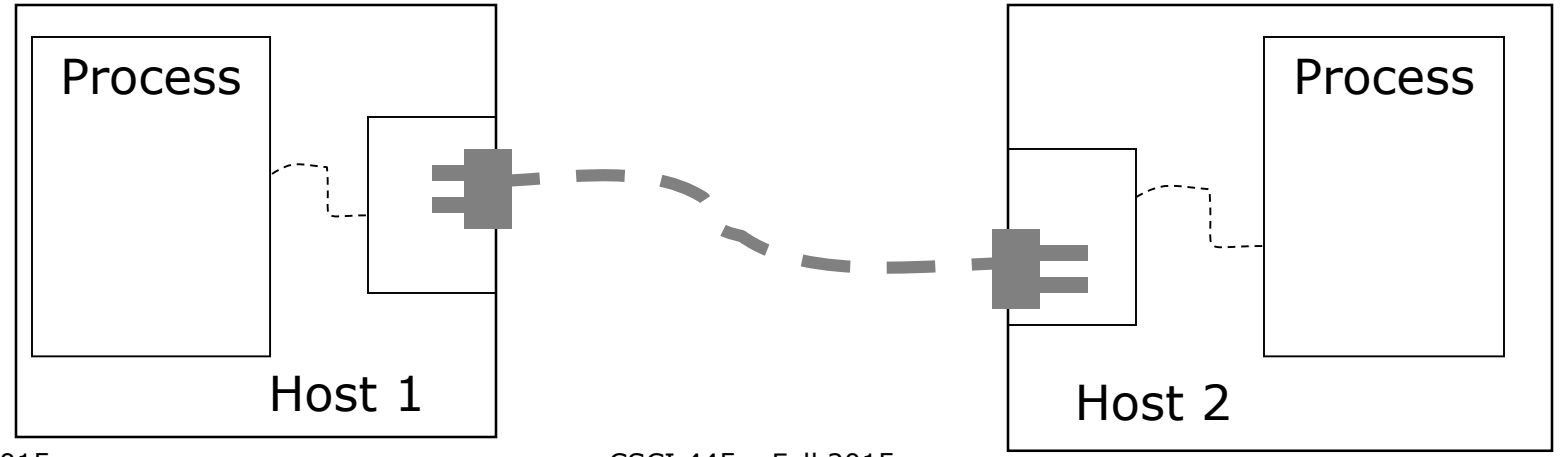
- Interested in programming Ethernet
- In fact, writing programs using functionality provided by Ethernet adapters and availed by their drivers

Working with Ethernet

- OS
  - Linux
- API
  - A few Berkeley Socket APIs
    - As Linux system calls

Berkeley Sockets

- Protocol provides a set of interfaces → abstract
- API (application programming interface) → how the interfaces exposed in a particular operating system
- Berkeley socket interfaces
  - APIs to multiple protocols
  - Socket: a “point” where an application process attaches to the network; “end-point” of communication



Programming Ethernet with Sockets

- Learn socket APIs to
  - Create a socket
  - Send messages via the socket
  - Receive message via the socket
- Example programs using a typical setup
  - Write two programs (A, B)
    - Program A contains and runs the server logic
    - Program B contains and runs the client logic

Creating Socket

- int socket(int domain, int type, int protocol)
  - A system call
  - Creates an endpoint for communication and returns a descriptor.
  - Look it up in Linux manual
    - man socket

Communication Domain

- int socket(**int domain**, int type, int protocol)
- AF_PACKET is our interest: Low level packet interface

  "*Packet sockets are used to receive or send raw packets at the device driver (OSI Layer 2) level. They allow the user to implement protocol modules in user space on top of the physical layer.*"
- More information
  - man 7 packet

Communication Type

- int socket(int domain, **int type**, int protocol)
- Specify a communication semantics with a communication domain
- For AF_PACKET
  - SOCK_RAW
    - For raw packets (including the link level header)
  - SOCK_DGRAM
    - For cooked packets (with the link level header removed)

Protocol

- int socket(int domain, int type, **int protocol**)
- Specifies a particular protocol to be used with the socket.
- Protocol is a protocol number in network order
- If domain is AP_PACKET
  - Protocol can be the IEEE 802.3 protocol number in network order.
  - linux/if_ether.h lists acceptable protocol numbers for Ethernet
    - Typical location: /usr/include/linux/if_ether.h

Protocol Number for Ethernet

- linux/if_ether.h lists acceptable protocol numbers for Ethernet
  - Typical location: /usr/include/linux/if_ether.h

```
……
#define ETH_P_LOOP  0x0060      /* Ethernet Loopback packet */
#define ETH_P_PUP   0x0200      /* Xerox PUP packet     */
#define ETH_P_PUPAT 0x0201      /* Xerox PUP Addr Trans packet  */
#define ETH_P_IP    0x0800      /* Internet Protocol packet */
……
#define ETH_P_802_3 0x0001      /* Dummy type for 802.3 frames  */
#define ETH_P_AX25  0x0002      /* Dummy protocol id for AX.25  */
#define ETH_P_ALL   0x0003      /* Every packet (be careful!!!) */
……
```

Protocol Number for Ethernet

- Which number to use?
- Depending on which protocol to use
  - Carry IP packet: 0x0800
  - Must be in network order, then htons(0x0800) or better htons(ETH_P_IP)
- What about developing a new protocol?
  - Choose a number not used
    - May run into the problem that other people also choose the number unused number
    - To prevent it, get approval from the IANA
- What about receiving all frames

Protocol Number for Ethernet

- What about receiving all frames
  - ETHER_P_ALL
  - In network order, htons(ETH_P_ALL)

Putting Together for Ethernet

```
#define MY_PROTOCOL_NUM 0x60001
int sockfd;

……
sockfd = socket(AP_PACKET,
                SOCK_RAW,
                htons(MY_PROTOCOL_NUM));

if (sockfd == -1) {
    /* deal with error */
}
```

Putting Together for Ethernet

```
#define MY_PROTOCOL_NUM 0x60001
int sockfd;


……
sockfd = socket(AP_PACKET,
                SOCK_DGRAM,
                htons(MY_PROTOCOL_NUM));

if (sockfd == -1) {
    /* deal with error */
}
```

Sending Messages

- ssize_t send(int sockfd, const void *buf, size_t len, int flags);
- ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);
- ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
- ssize_t write(int fd, const void *buf, size_t count);

Sending Messages

- Look them up in Linux manual
  - man send
  - man sendto
  - man sendmsg
  - man 2 write

Sending Message

- Relationship among the system calls
  - write(fd, buf, len);

    is equivalent to

    send(sockfd, buf, len, 0);
  - send(sockfd, buf, len, flags);

    is equivalent to

    sendto(sockfd, buf, len, flags, NULL, 0);
  - write(fd, buf, len);

    is equivalent to

    sendto(sockfd, buf, len, 0, NULL, 0);

Sending Messages

- ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen);
  - sockfd: the file descriptor of the sending socket
  - buf: message to send
  - len: message len
  - flags: the bitwise OR of flags or 0
  - dest_addr: the address of the target
  - addrlen: the size of the target address

Message

- Case 1

```
sockfd = socket(AP_PACKET, SOCK_RAW,
                htons(MY_PROTOCOL_NUM));
```

  - *buf* contains Ethernet header and data

- Case 2

```
sockfd = socket(AP_PACKET, SOCK_DGRAM,
                htons(MY_PROTOCOL_NUM));
```

  - *buf* contains data

Target Address

- struct sockaddr *desk_addr
  - struct sockaddr * is more of a place holder
  - desk_addr should points to an instance of struct sockaddr_ll

struct sockaddr_ll

```
struct sockaddr_ll {
    unsigned short sll_family;   /* Always AF_PACKET */
    unsigned short sll_protocol; /* Physical layer protocol */
    int            sll_ifindex;  /* Interface number */
    unsigned short sll_hatype;   /* ARP hardware type */
    unsigned char  sll_pkttype;  /* Packet type */
    unsigned char  sll_halen;    /* Length of address */
    unsigned char  sll_addr[8];  /* Physical layer address */
};
```

Receiving Messages

- ssize_t recv(int sockfd, void *buf, size_t len, int flags);
- ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen);
- ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
- ssize_t write(int fd, const void *buf, size_t count);

Receiving Message

- Look them up in Linux manual
  - man recv
  - man recvmsg
  - man recvfrom
  - man 2 read

Receiving Message

- Relationship among the system calls
  - read(fd, buf, len);

    is equivalent to

    recv(sockfd, buf, len, 0);
  - recv(sockfd, buf, len, flags);

    is equivalent to

    recvfrom(sockfd, buf, len, flags, NULL, NULL);
  - read(fd, buf, len);

    is equivalent to

    recvfrom(sockfd, buf, len, 0, NULL, NULL);

Message

- Case 1

```
sockfd = socket(AP_PACKET, SOCK_RAW,
                htons(MY_PROTOCOL_NUM));
```

  - *buf* contains Ethernet header and data

- Case 2

```
sockfd = socket(AP_PACKET, SOCK_DGRAM,
                htons(MY_PROTOCOL_NUM));
```

  - *buf* contains data

Socket Option

- Packet sockets can be used to configure *physical layer multicasting* and *promiscuous mode*.
- Get socket option
  - int getsockopt(int sockfd, int level, int optname, void *optval, socklen_t *optlen);
- Set socket option
  - int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);

Putting Together

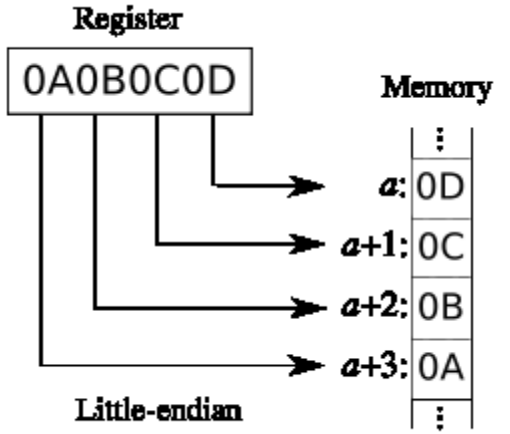
- Sample programs

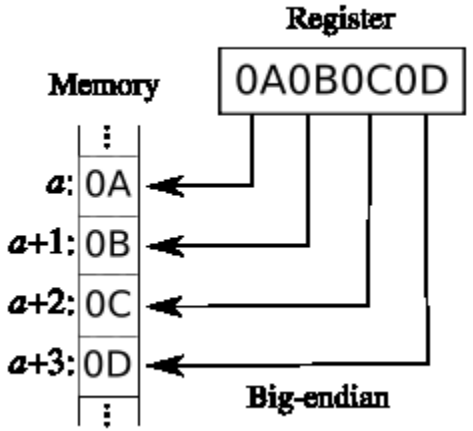
Summary

- Application implementation
  - Berkeley Socket APIs
- If you forget byte order, continue to read

Byte Order: Big Endian and Little Endian

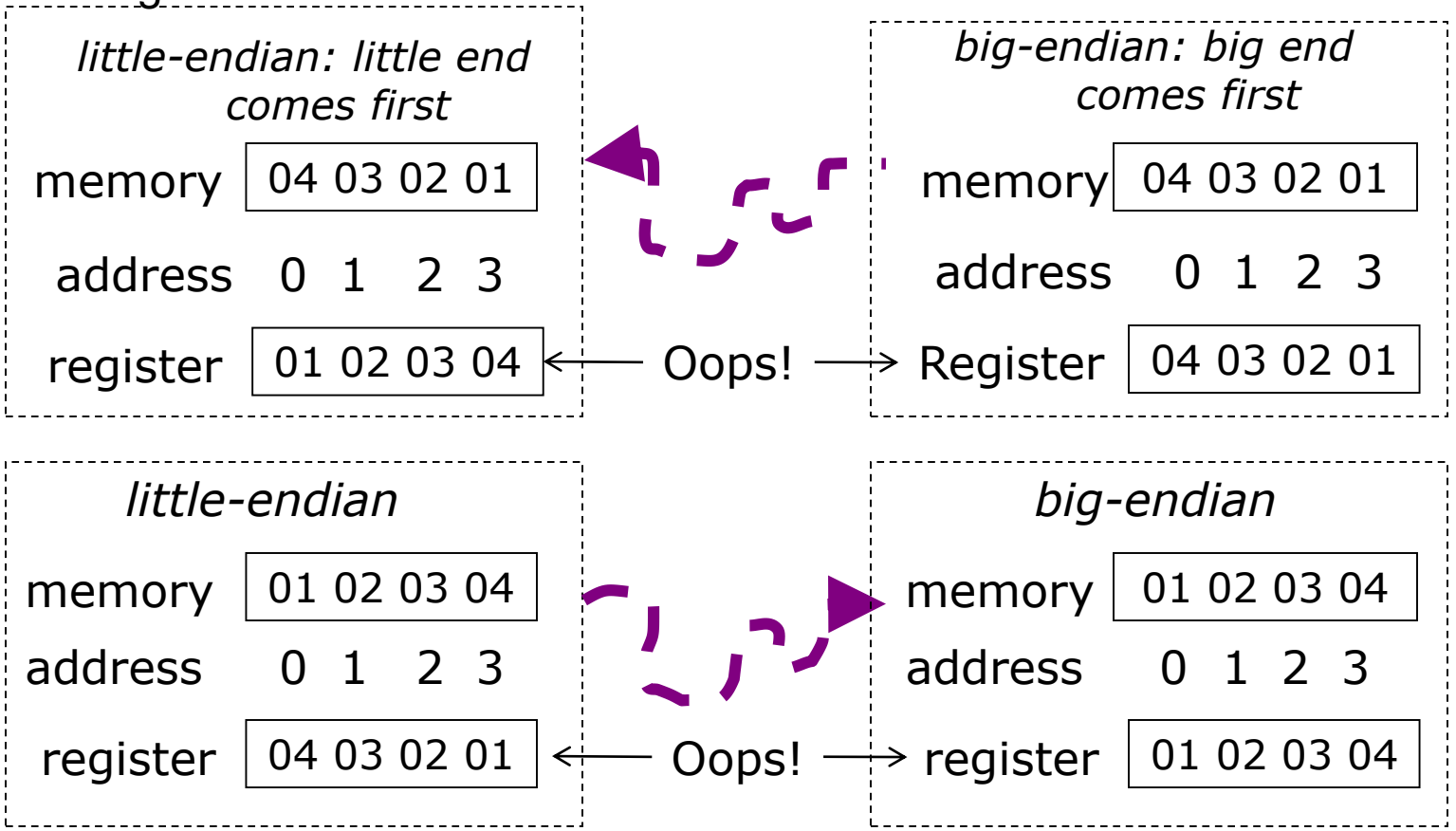
- Little Endian
  - Low-order byte of a word is stored in memory at the lowest address, and the high-order byte at the highest address → The little end comes first
- Big Endian
  - high-order byte of a word is stored in memory at the lowest address, and the low-order byte at the highest address → The big end comes first





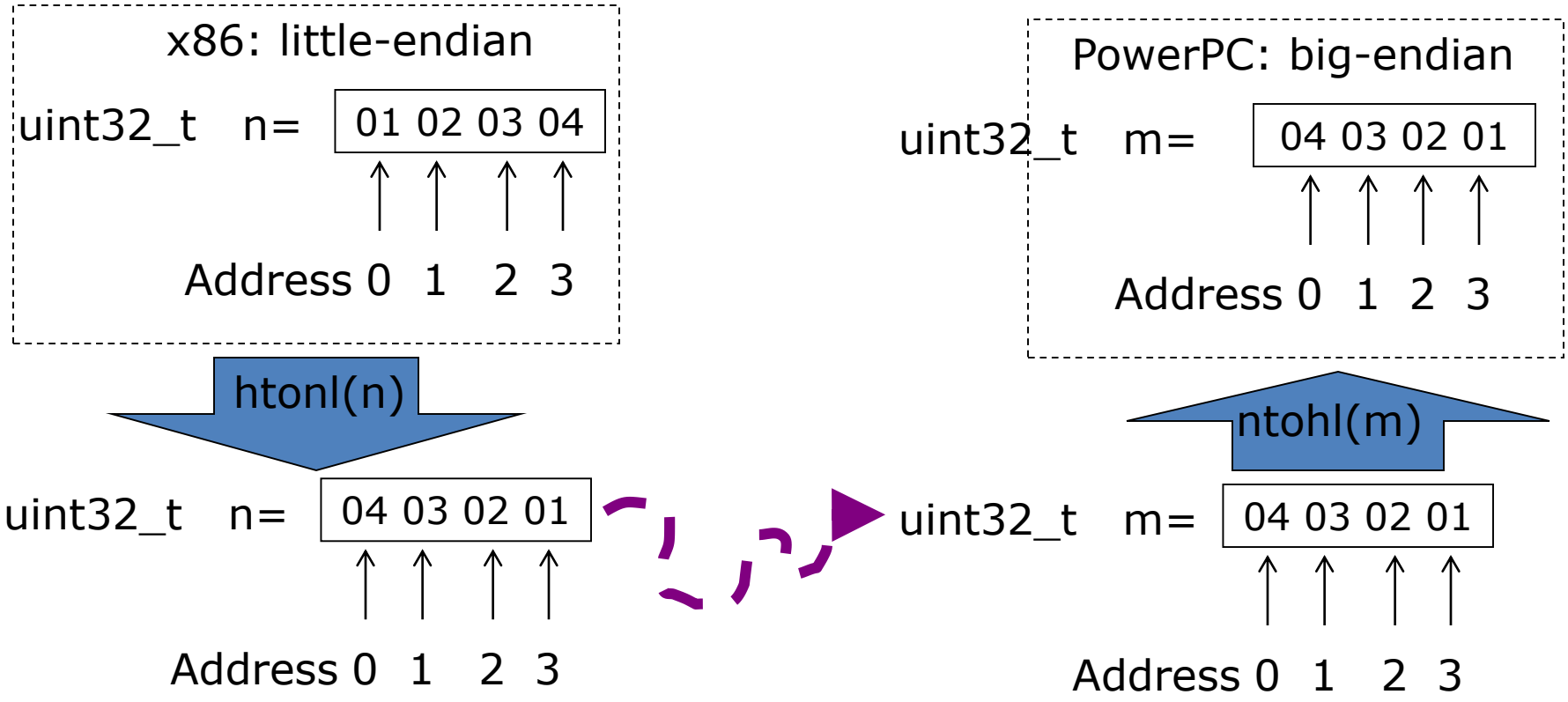
Endian-ness: Transfer Integer over Network

- Integer to transfer: 0x04030201



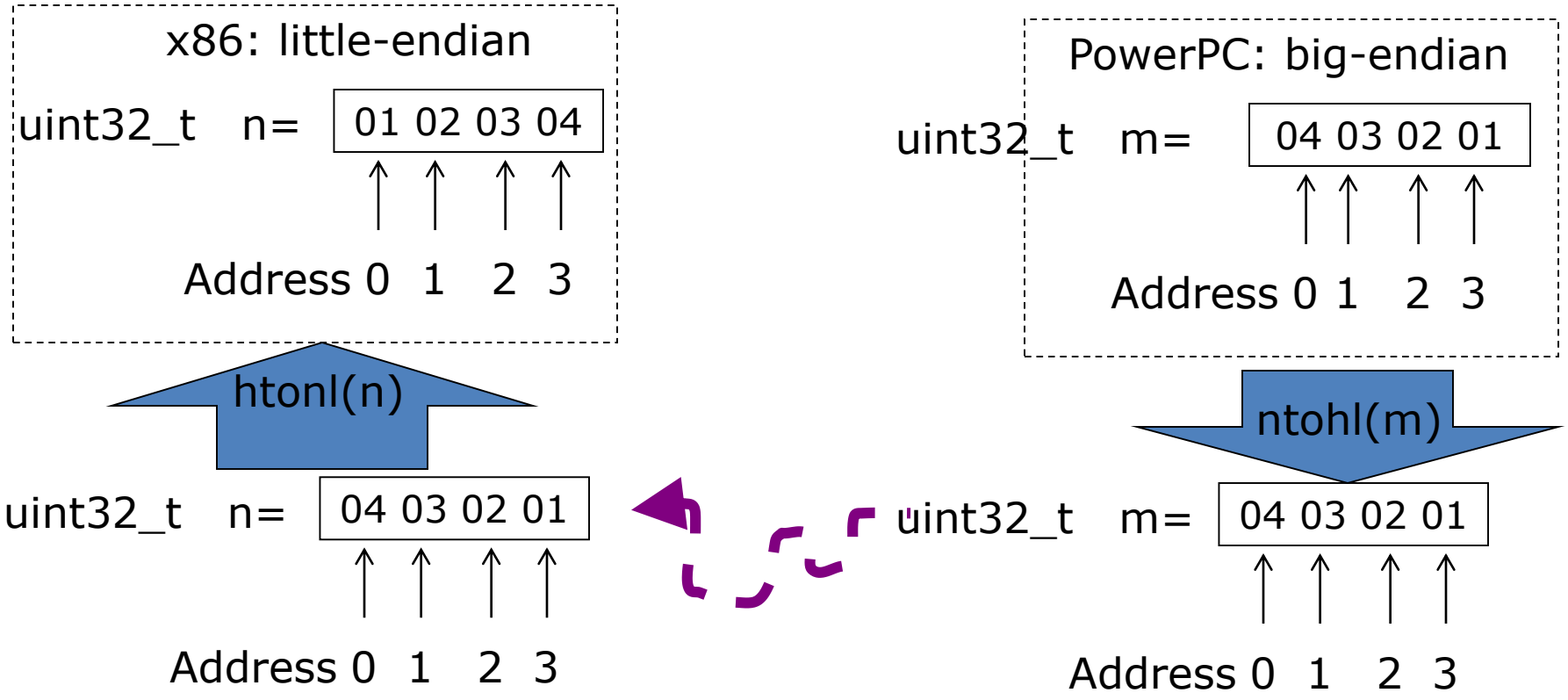
Network Order

- Integer to transfer: 0x04030201



Network Order

- Integer to transfer: 0x04030201



Acknowledgements

- Some pictures used in this presentation were obtained from the Internet
- The instructor used the following references
  - Larry L. Peterson and Bruce S. Davie, Computer Networks: A Systems Approach, 4th Edition, Elsevier, 2006
  - Dimitri Bertsekas and Robert Gallager, Data Networks, 2nd Edition, Prentice-Hall, 1992
  - Andrew S. Tanenbaum, Computer Networks, 4th Edition, Prentice-Hall, 2003
  - Larry L. Peterson's (http://www.cs.princeton.edu/~llp/) Computer Networks class web site
  - Tarek F. Abdelzaher's (http://www.cs.uiuc.edu/homes/zaher/) Computer Networks class web site:
  - IBM e-server iSeries Socket Programming Manual Version 5 Release 3 (http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=/rzab6/rzab6soxoverview.htm)