Georgia Tech Computer Science

CS2200
Systems and Networks
Spring 2022

Lecture 23:
PIO, DMA, Networking

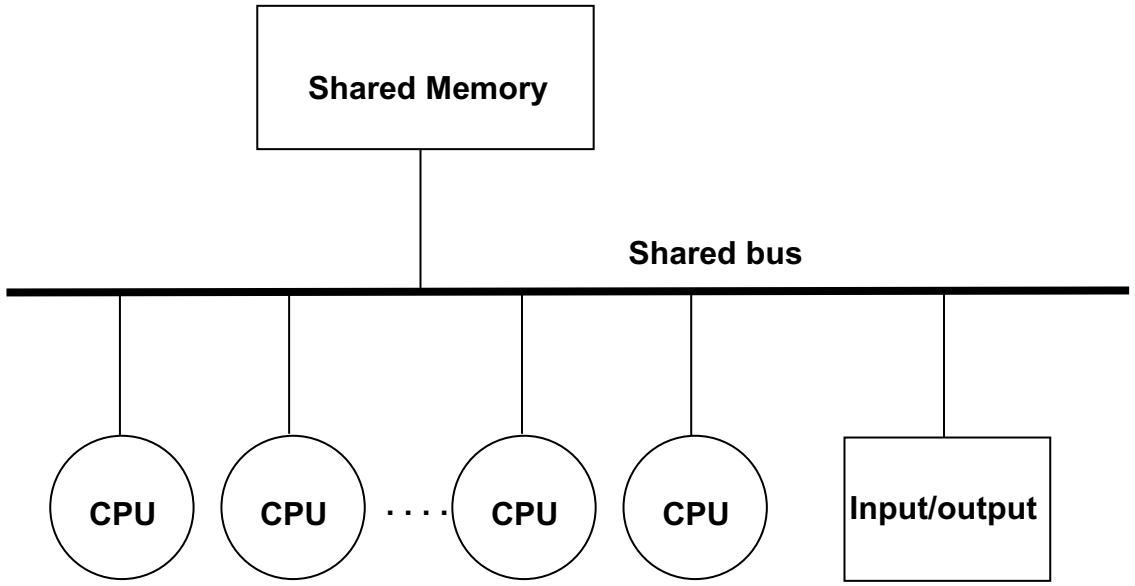
Alexandros (Alex) Daglis
School of Computer Science
Georgia Institute of Technology
adaglis@gatech.edu

Lecture slides adapted from Bill Leahy and Charles Lively of Georgia Tech

Roadmap

- Finish up parallel systems (Chapter 12)
  - Hardware support for SMP
- Programmed IO and DMA
  - Chapter 10 (10.1 – 10.7)
- Networking
  - Chapter 13

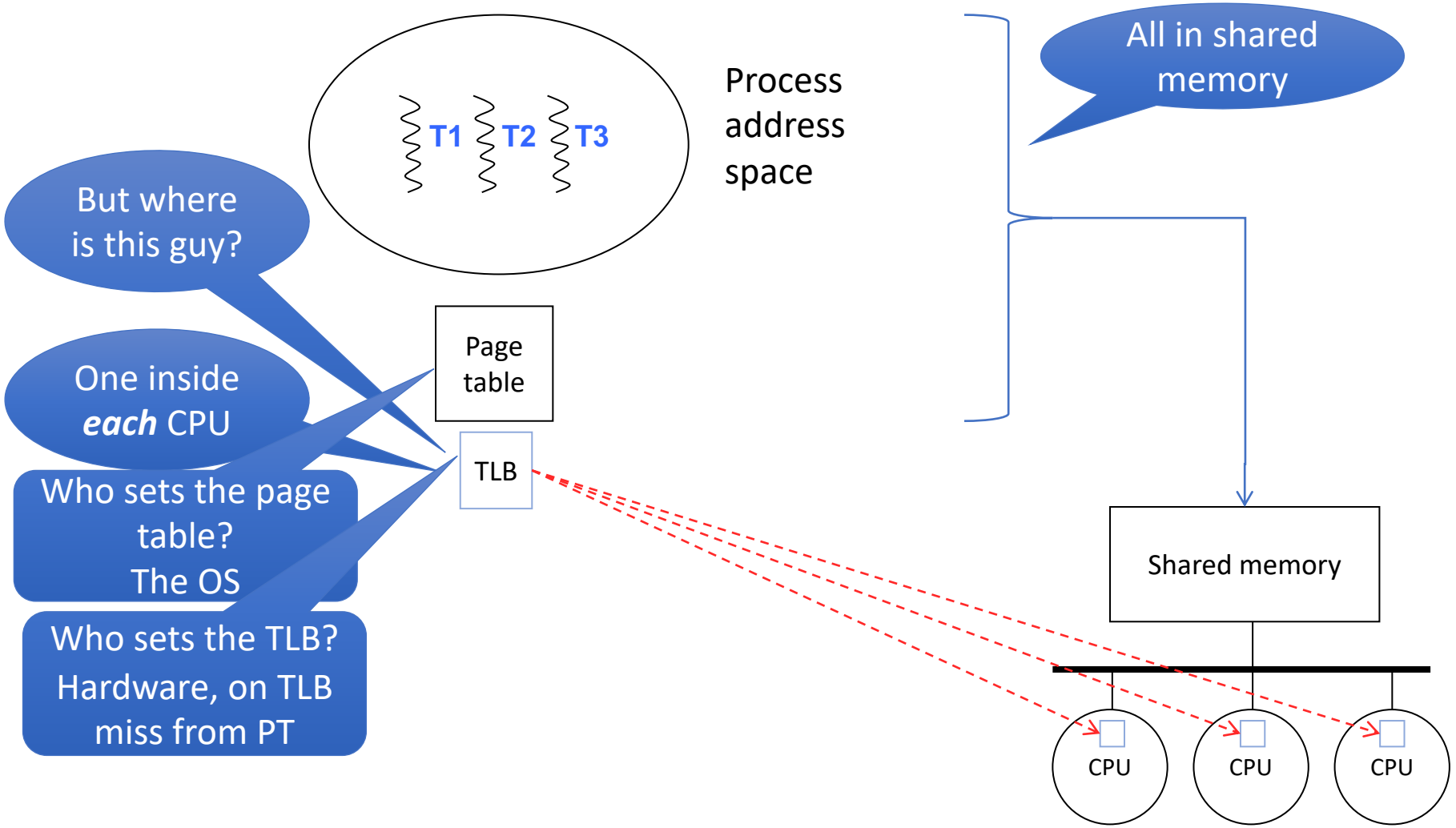
How do we implement Symmetric Multi Processing (SMP)?



The System (hardware+OS) has to ensure 3 things:

1. Threads of the same process share the same PT
2. Threads have synchronization atomicity
3. Threads have identical views of memory

1) All threads share the same page table



SMP context switch handling

- As in the single-CPU case, the TLB must be flushed of user-space addresses on context switch
- How do multiple processors complicate this?
  - Basically, they don't
  - Any time a CPU is switched to a new thread, the OS flushes user entries from that CPU's TLB
  - There's no need to affect other TLBs

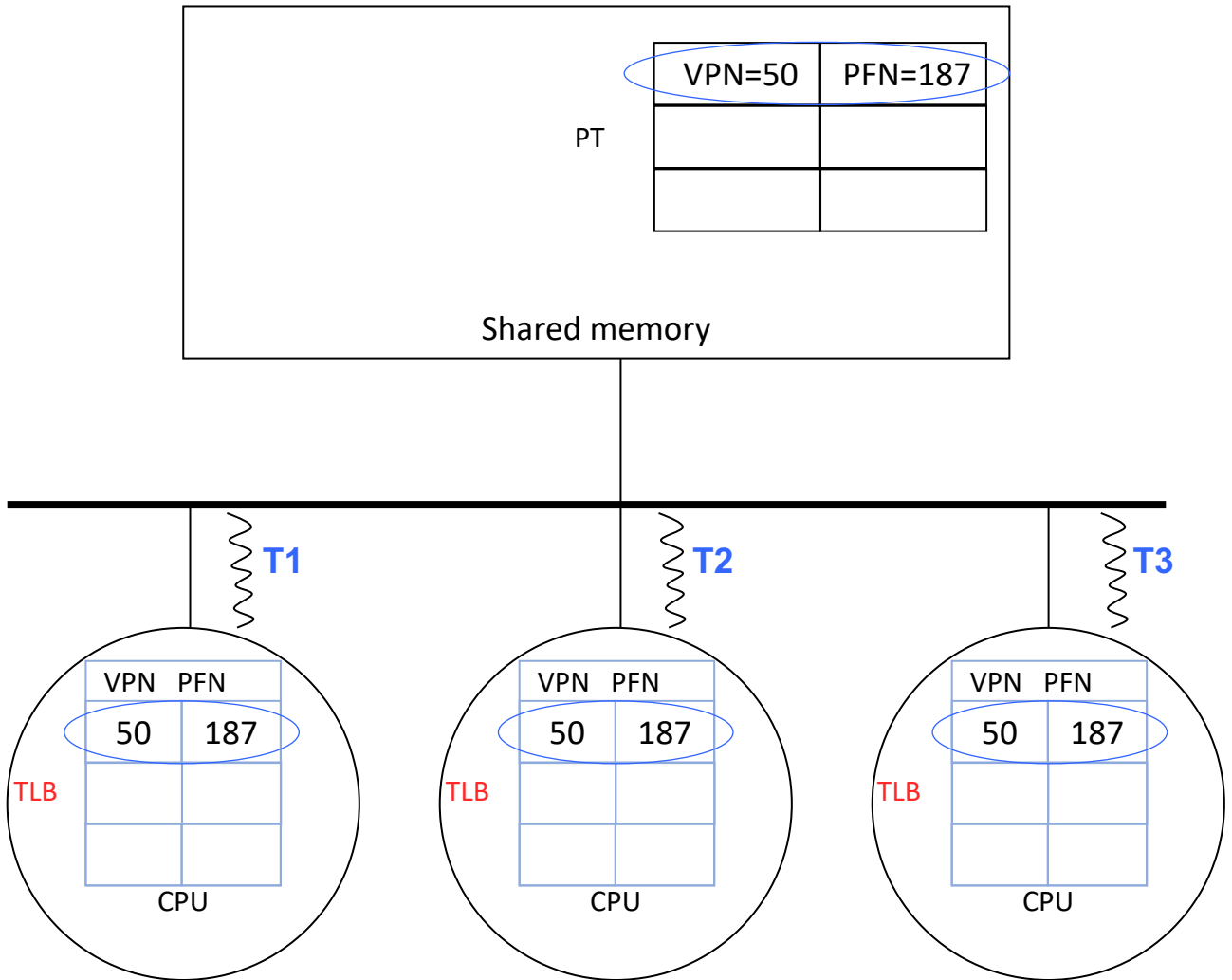
SMP page replacement handling

- On page replacement by the OS (which can happen on any of the CPUs)
- OS must
  - Evict the TLB entry for that page
    (must happen even on a uniprocessor)
  - Tell the OS on other CPUs to evict the corresponding entry (if present) using software interrupts
  - This is called TLB Shootdown
- All of this happens in software by the OS

  ➔ Another partnership of hardware and software

Example: Handling an SMP page fault

Note that the TLBs have each pulled in VPN=50 because the threads each referenced that page



Example: Handling an SMP page fault

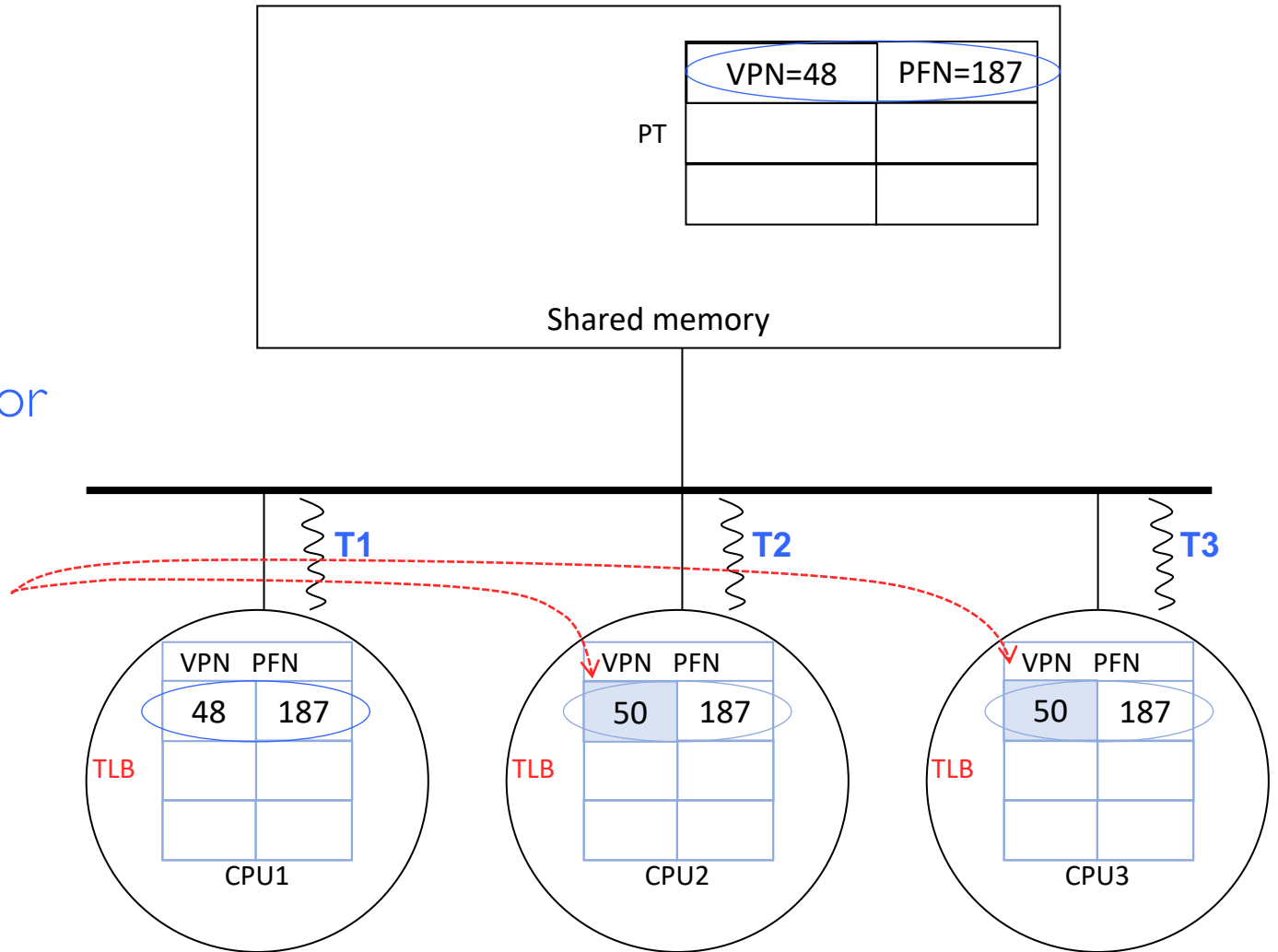
- Assume
  - T1 encounters a page fault on VPN=48
  - OS decides to evict VPN=50
  - And use PFN=187 for hosting VPN=48

Example: Handling an SMP page fault

OS changes the page table entry for VPN=50

Then because it's running on CPU1, it evicts the TLB entry for VPN 50

Now we've got stale TLB entries in CPU2 and CPU3

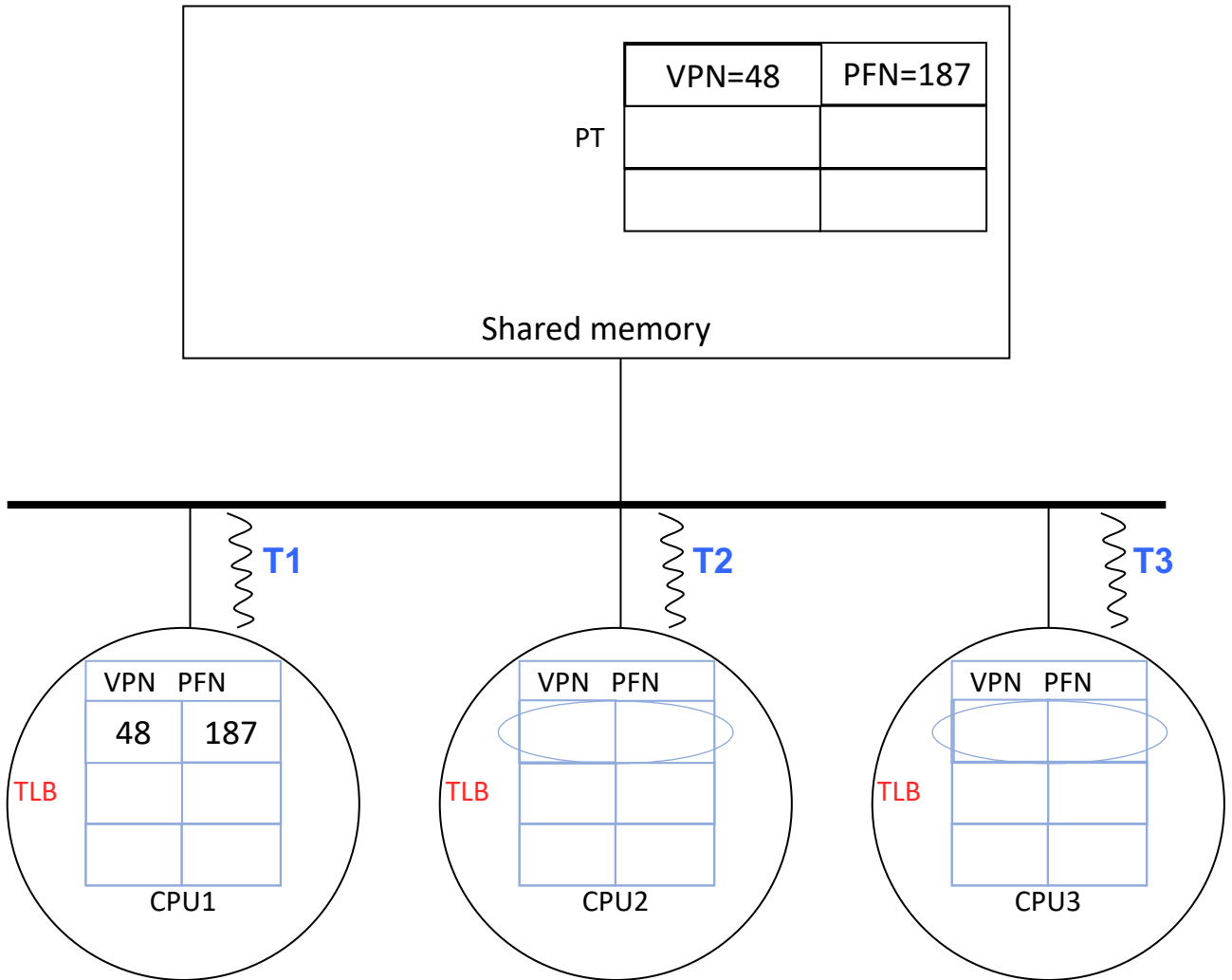


Example: Handling an SMP page fault

Then we have the TLB Shootdown

The OS arranges to invalidate the corresponding TLB entries on the other CPUs

And the CPUs can pull in the PTE when they next reference VPN=48



Session ID: dag2200

Ensuring that all threads of a process share an address space in an SMP is

A. I just want the participation credit
B. Impossible
C. Trivially achieved since the page table resides in shared memory
D. Achieved by careful replication of the page table by the operating system for each thread
E. Achieved by special-purpose hardware that no one has told us about yet

Session ID: dag2200

Keeping the TLBs consistent in an SMP

A. I just want the participation credit
B. Is the responsibility of the programmer
C. Is the responsibility of the hardware
D. Is the responsibility of the operating system
E. Is not possible

2) Threads have synchronization atomicity

- We already introduced the TEST-AND-SET instruction to acquire locks atomically
- It should be easy on a multiprocessor, right?
- The location we use for synchronization is in shared memory, so no sweat.
- What could go wrong?

2) Threads have synchronization atomicity

We have memory location L for TEST-AND-SET

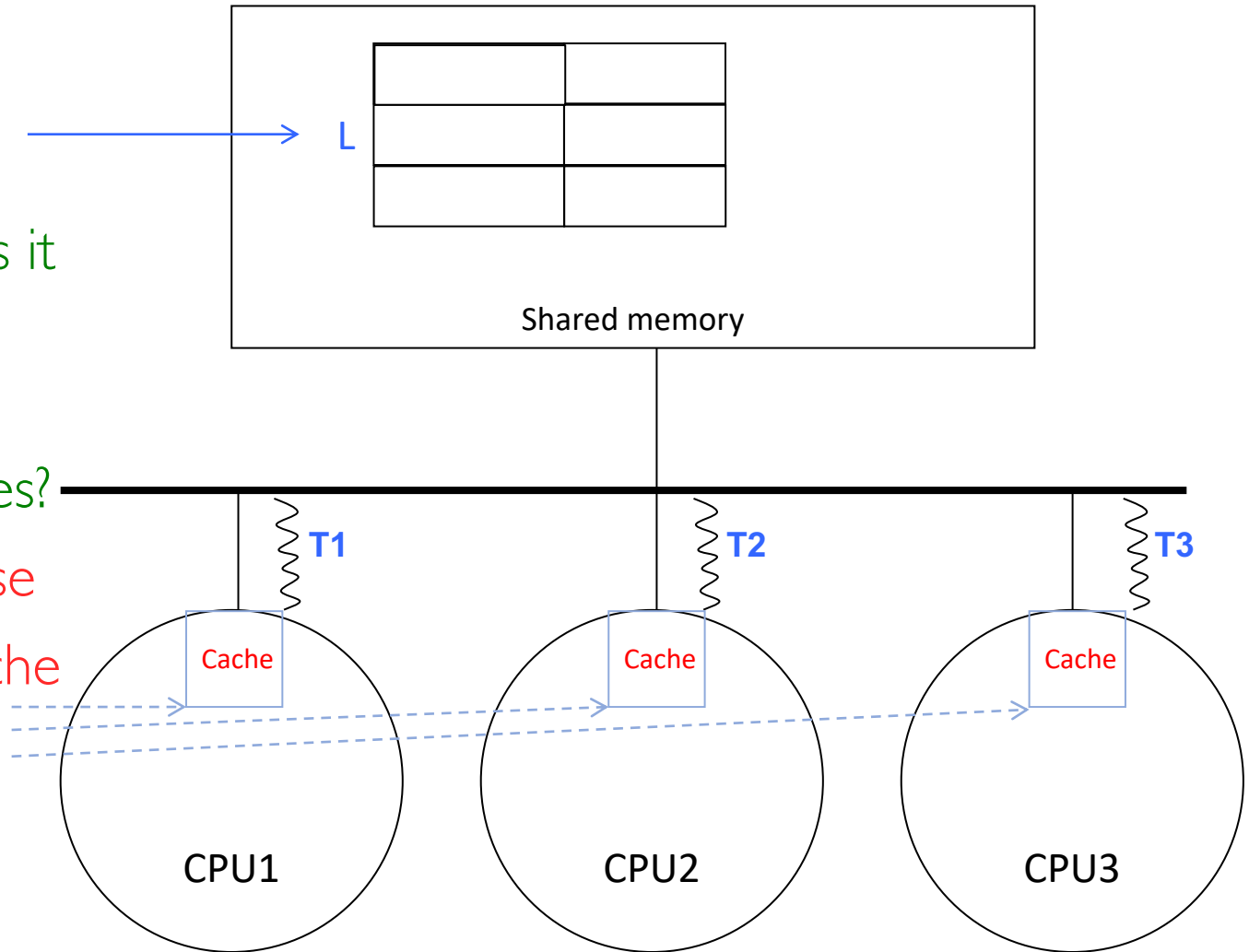
Each CPU can access it

But…what did we forget?

Where are the caches?

In the CPUs of course

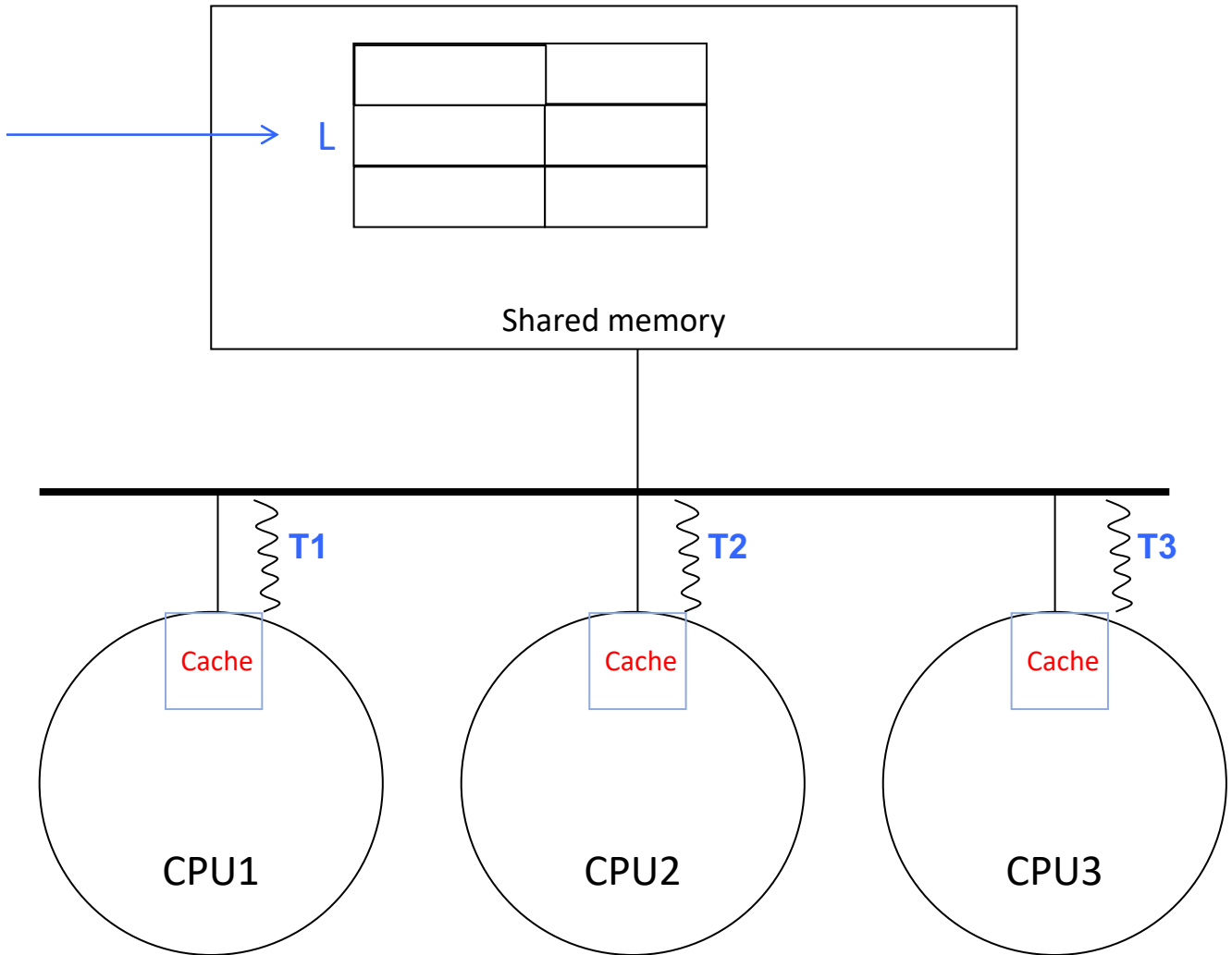
If L is cached, the cache is wrong!



2) Threads have synchronization atomicity

What shall we do?

One solution is to bypass the cache for the T&S instruction

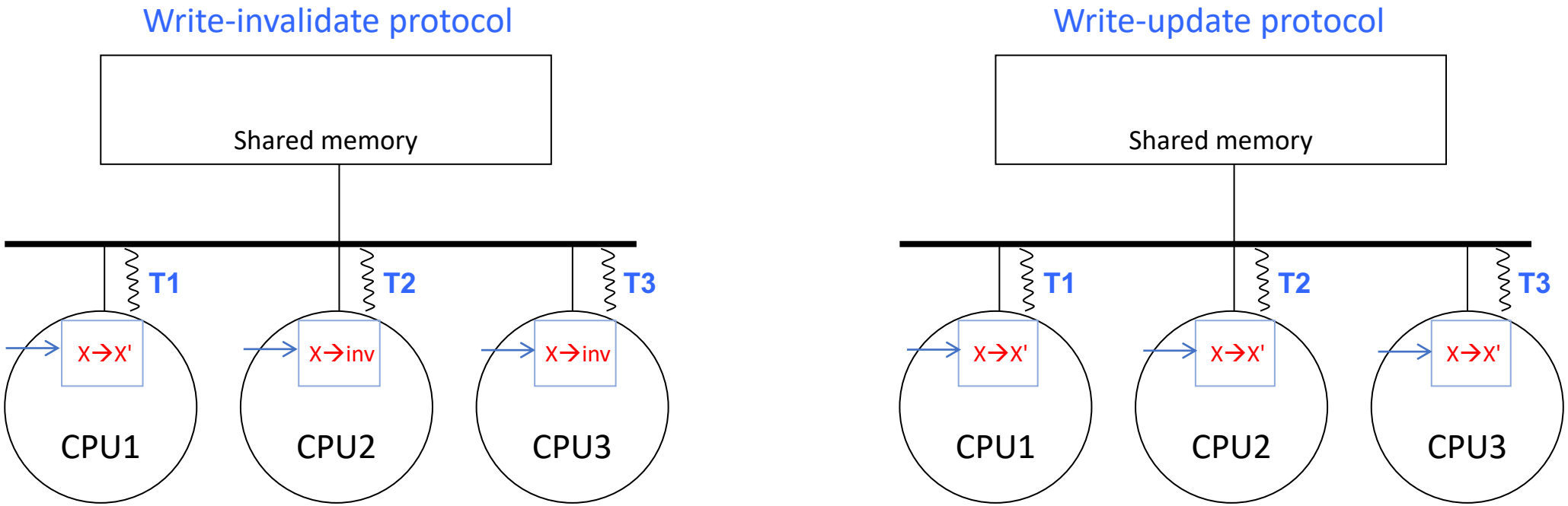


Requirements for SMP

1. ~~Threads of the same process share the same PT~~
2. ~~Threads have synchronization atomicity~~
3. Threads have identical views of memory
   - ➔ This implies that access to a memory location returns the same value on all CPUs
   - ➔ We'll refer to the method of keeping all the copies of the same data across caches as a cache coherence protocol

3) Threads have identical views of memory

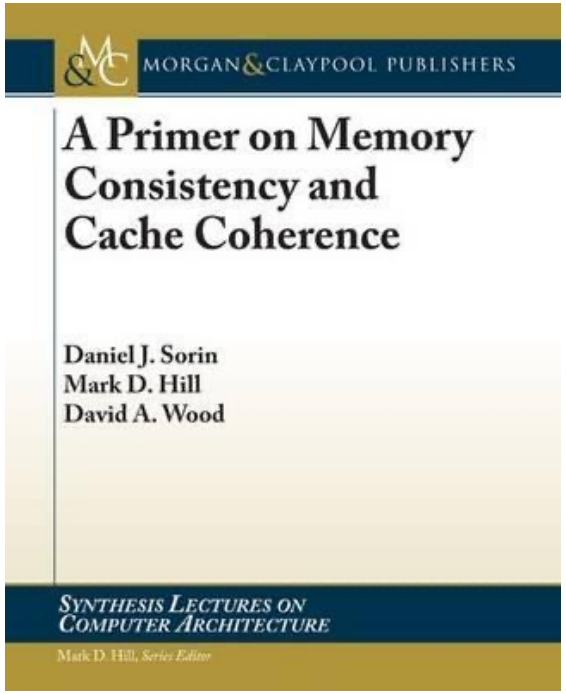
- Two possible solutions, in hardware
- Both: Cache becomes active agent and monitors or ***snoops*** the bus
- Let's watch a memory location change value from X to X'



Session ID: dag2200

Keeping the caches coherent in an SMP

A. I just want the participation credit
B. …is the responsibility of the user program
C. …is the responsibility of the hardware
D. …is the responsibility of the operating system
E. …is impossible
F. …is why we don't allow caches in SMP systems



SMP Hardware Support Summary

- Page tables in shared memory
  - Set up by the OS
  - Used by the hardware
- TLB consistency in software by the OS
  - Hardware brings PTE into the TLB from the PT
  - Page replacement algorithm changes the PT and does the TLB shoot-down
- Synchronized atomicity
  - Test-and-set instruction serialized by the shared bus
  - Atomic read-modify-write transaction
- Cache coherence in hardware
  - Invalidation based or update based

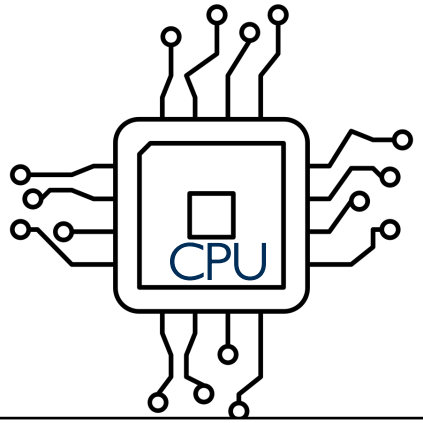
Roadmap

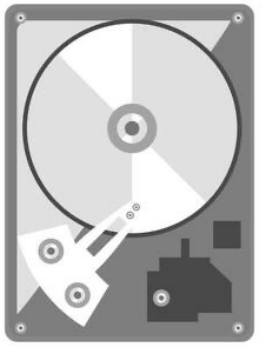
- ~~Finish up parallel systems (Chapter 12)~~
  - ~~Hardware support for SMP~~

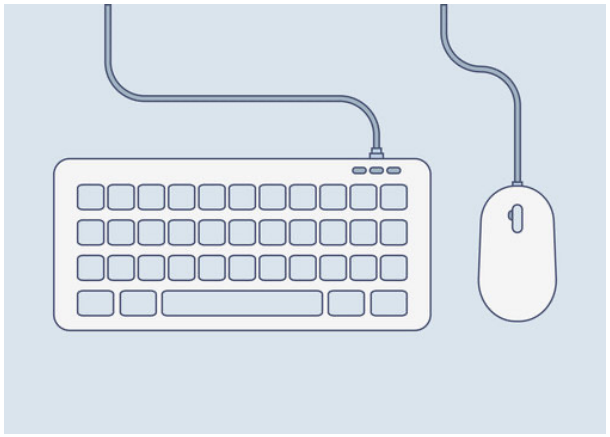
- Programmed IO and DMA
  - Chapter 10 (10.1 – 10.7)

- Networking
  - Chapter 13

Beyond the processor and memory



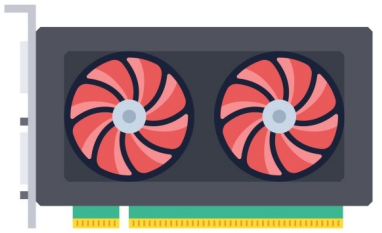


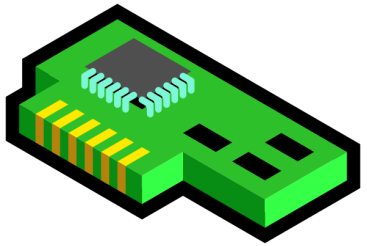


Memory Hierarchy

$$$ Caches

DRAM

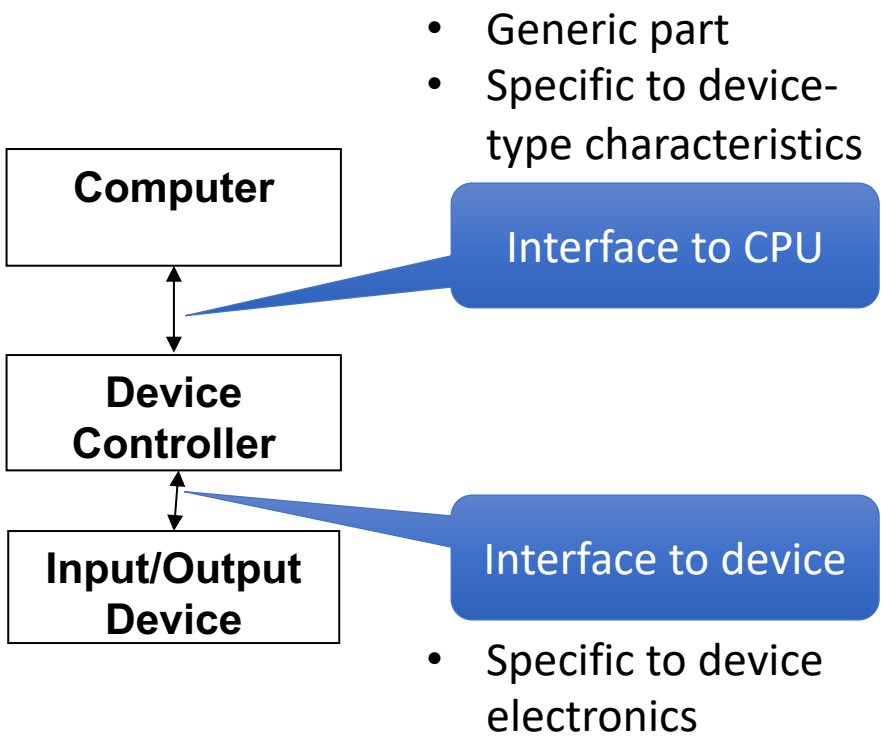




Anything else??

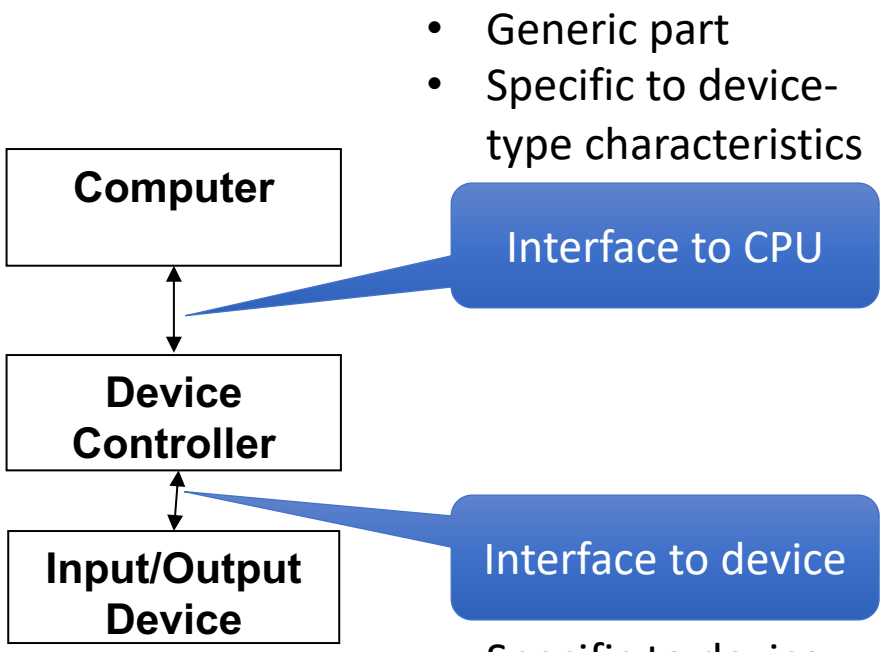
A lot of peripheral devices!

Communication: CPU and I/O Devices



- How does the CPU "talk" to the controller
  - Memory-mapped I/O
  - Load/store instructions
  - (Alternative is special I/O instructions)
- How is data movement effected?
  - Slow speed
    - Programmed I/O
  - High speed
    - DMA & interrupts

Communication: CPU and I/O Devices

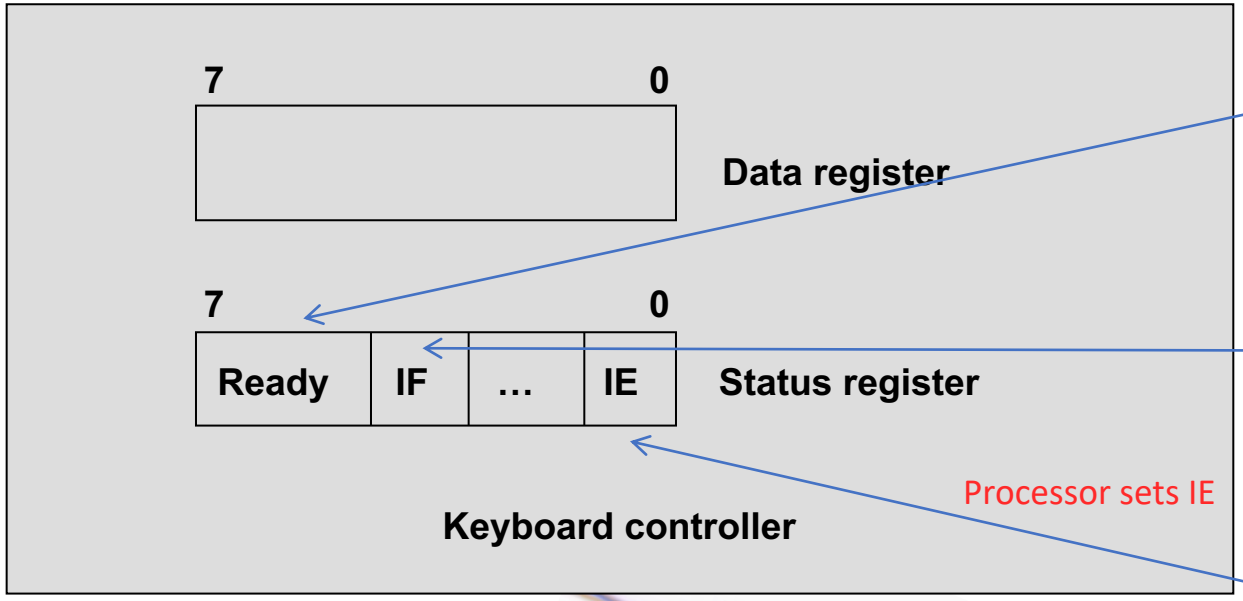


- Generic part
- Specific to device-type characteristics
- Specific to device electronics

- What commands do we issue to devices?
  - Camera
  - Display
  - Audio
  - …

A keyboard device

Controller

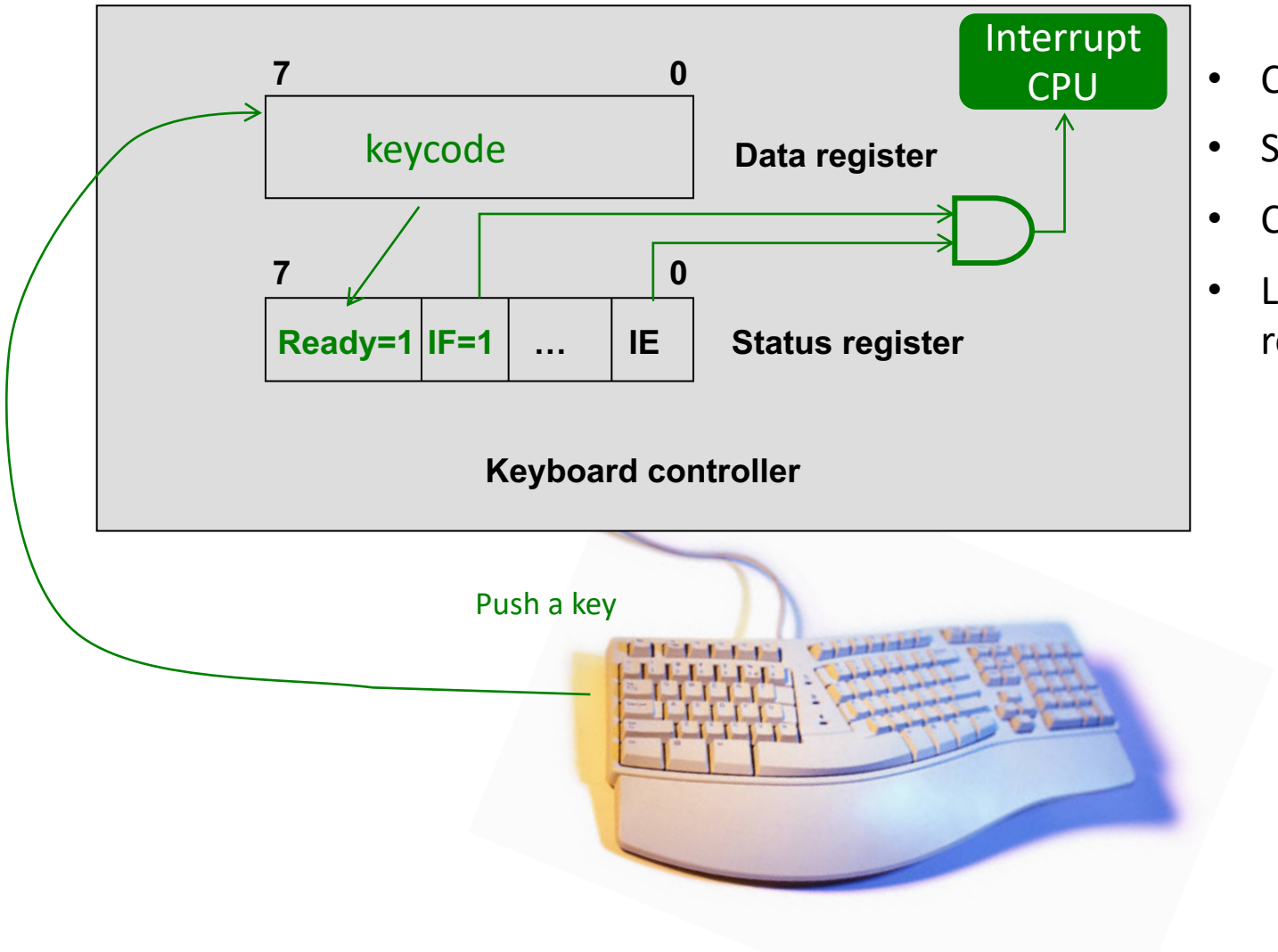


- Ready: there's a byte waiting in the data register
- IF: the device has signaled an interrupt
- IE: If 1 interrupt when Ready

- Slow speed device
- Service it with programmed I/O (PIO)

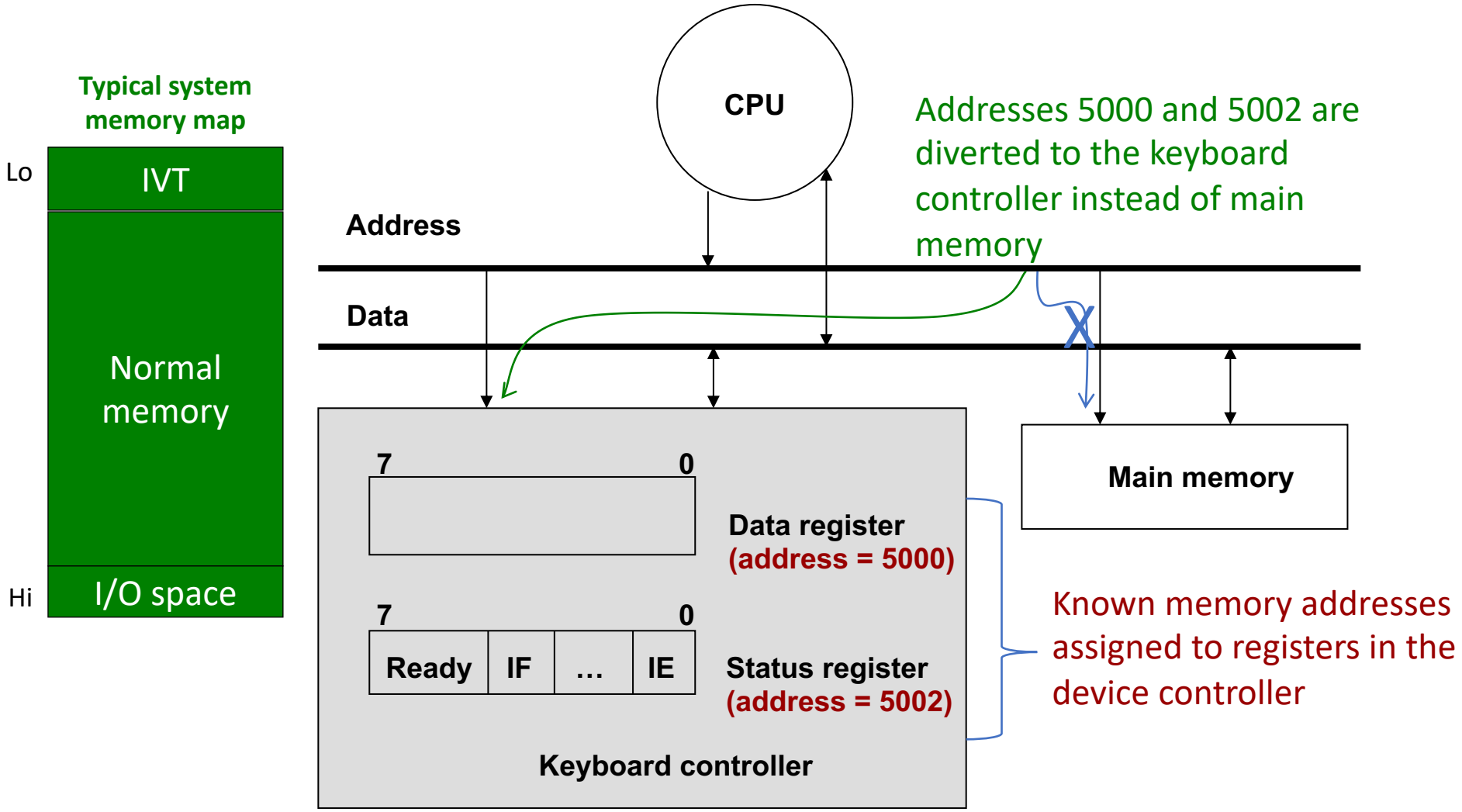
Device

A keyboard device



- Commands from the CPU:
- Set IE
- Check Ready equals 1
- Load keycode from data register

Memory mapped I/O



Memory mapped I/O

Commands?

Check for new data

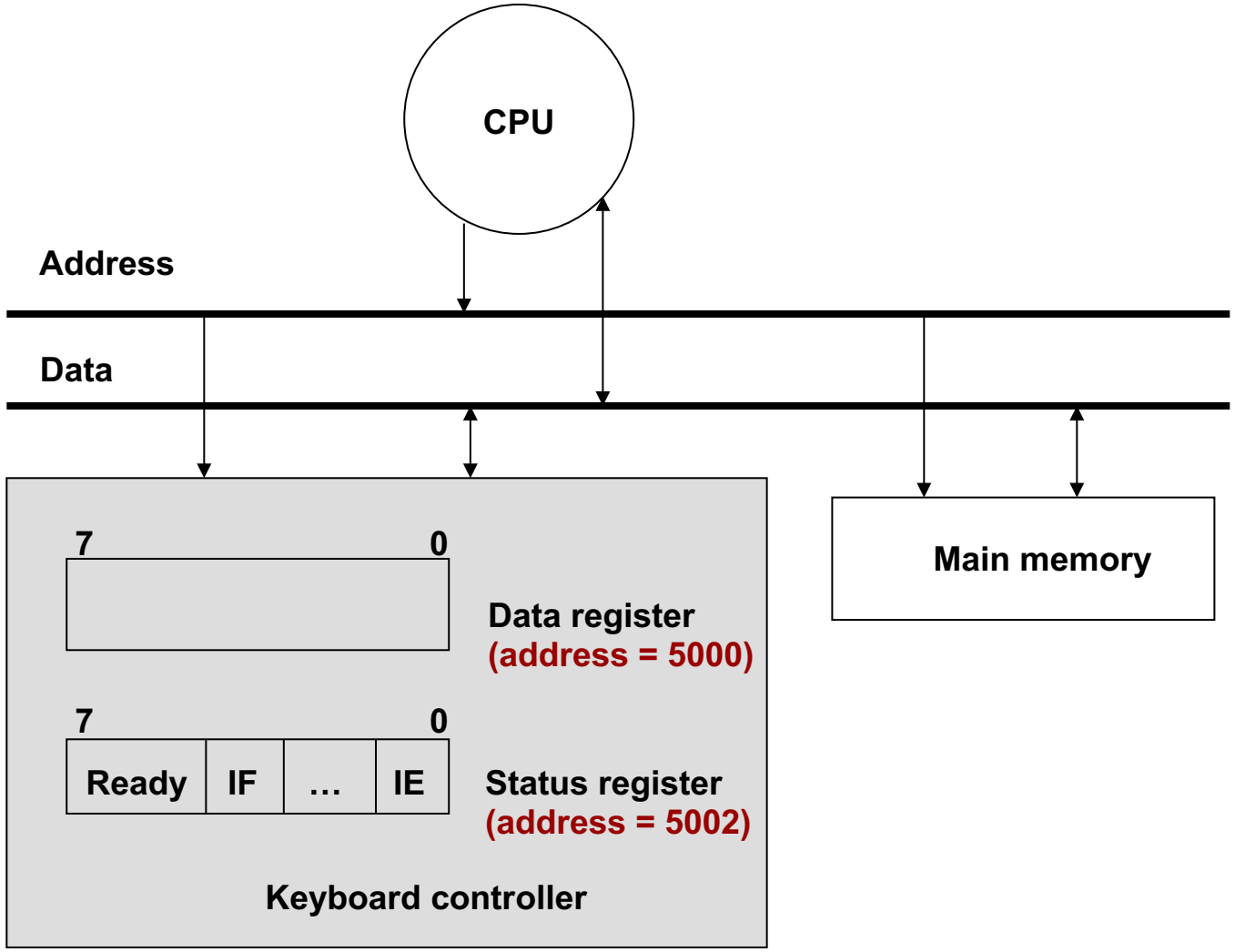
LD R1,mem[5002]

Read data

LD R2,mem[5000]

Set IE

ST #1,mem[5002]



Programmed I/O (PIO)

- Slow speed devices (Keyboard, mouse, etc.)
- CPU executes program to move data
- Often uses polling

```
X:  LD      R1, statusreg     ; Get the status register
    BRZP    X                 ; if high bit is 0, branch back one
    LD      R2, datareg       ; Load the data into R2
    ST      #0, statusreg     ; Clear the status register
```

LD usually clears the ready bit as a side effect of reading the data register

- Busy waiting – wastes processor resources

Interrupt-driven I/O

- Slow speed devices (Keyboard, mouse, etc.)
- CPU executes program for moving data
- Can be **interrupt driven**

```
ST      #1, statusreg ; Set the IE bit
```

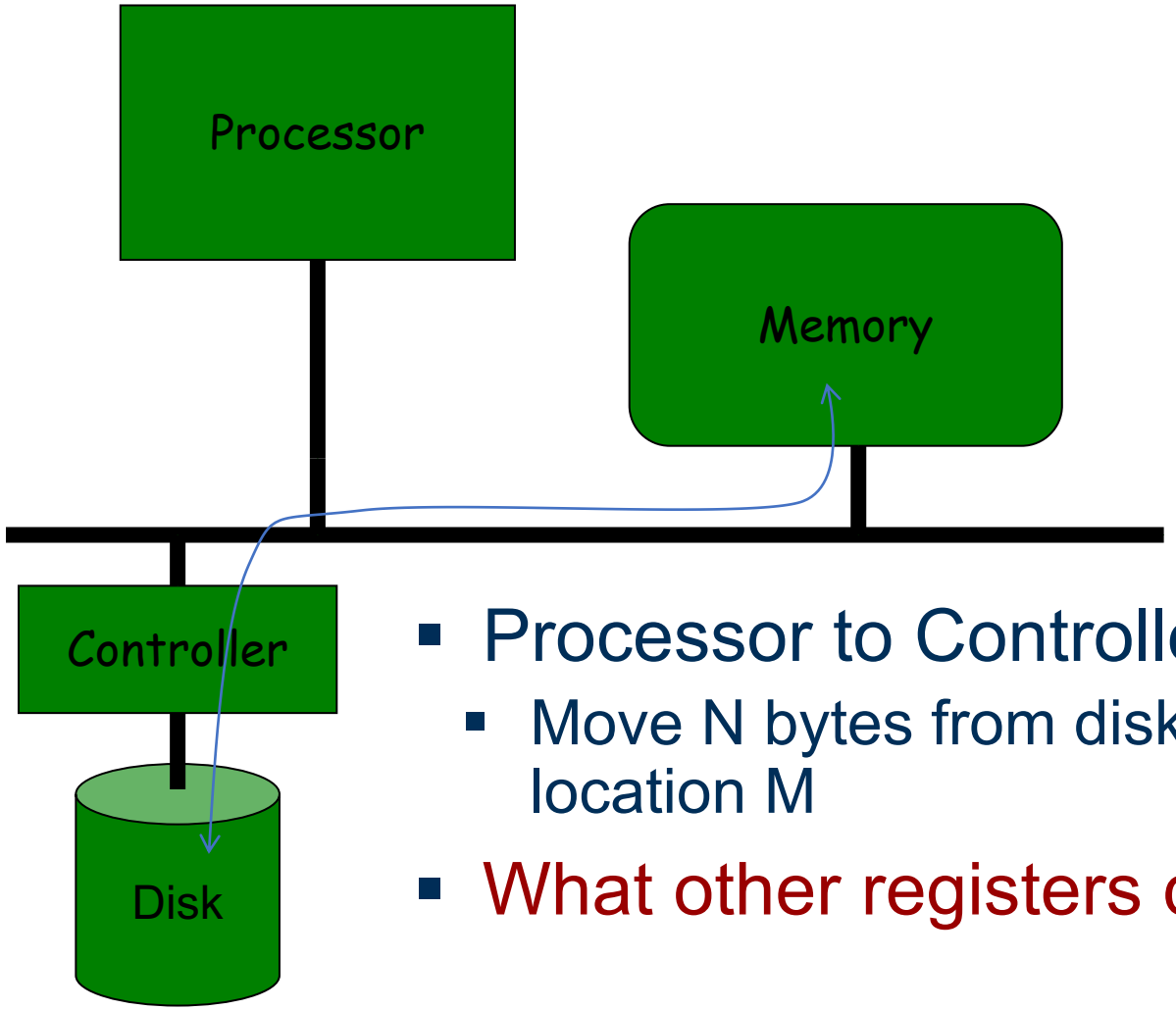
Upon interrupt, handler code is executed:

```
LD      R2, datareg     ; Load the data into R2
```

High speed devices

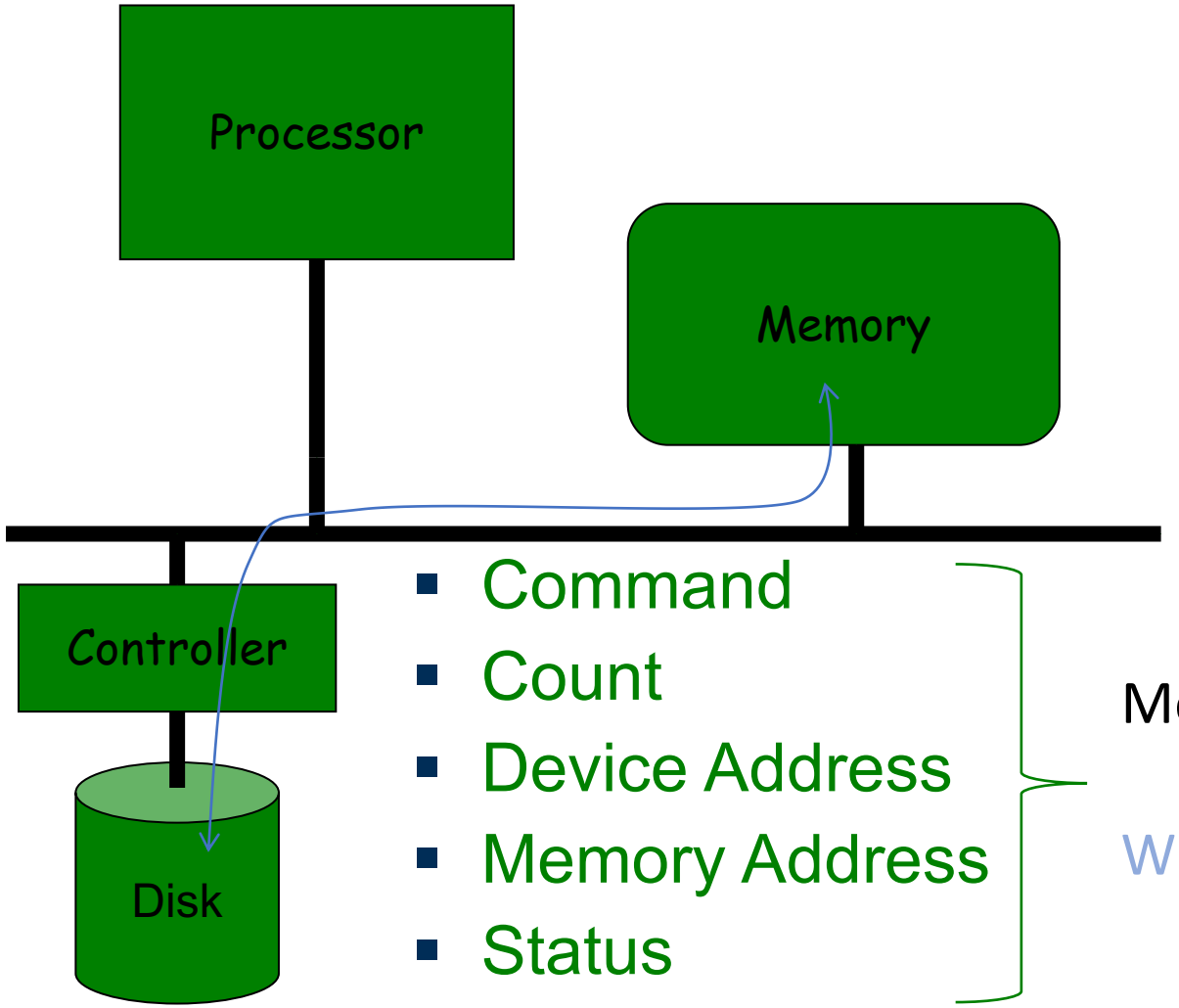
- Once a command is given by the CPU to the controller, data comes in continually
  - Streaming
- PIO has potential for data loss if the CPU doesn't poll or respond to the interrupt quickly enough
- Streaming devices move data to/from memory autonomously
- The CPU to controller interface?
  - Convey commands (read/write, IE, etc.)
  - Check status (error, etc.)

Direct Memory Access - DMA



- Processor to Controller:
  - Move N bytes from disk location X to memory location M
- What other registers do we need?

DMA registers



- Command
- Count
- Device Address
- Memory Address
- Status

Memory mapped registers

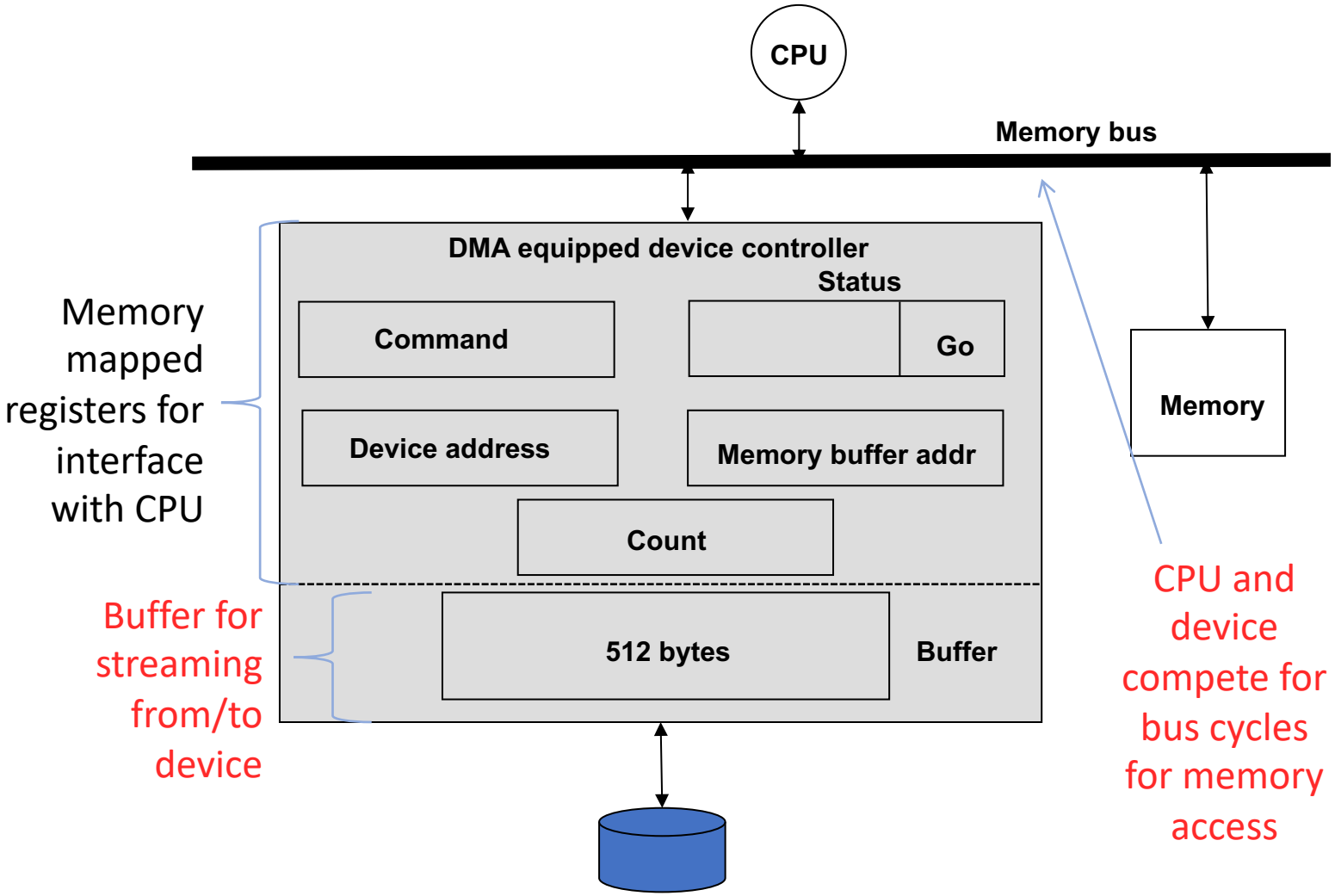
Where have you seen this before?

Program for conveying I/O commands

```
Store  N, count
Store  #block_num, device_addr
Store  #mem_buf_addr, mem_addr
Store  #write_command, command
Store  #1, status  ;  the signal to execute the command
```

- At this point the CPU's work is done
  - The device controller takes over to do the actual data movement
- Modern CPU chips already contain device controllers for PCI-E devices which share the on-chip memory controllers with the CPU cores

DMA


CPU
Memory bus
DMA equipped device controller
Status
Command
Go
Device address
Memory buffer addr
Count
512 bytes
Buffer
Memory
Memory mapped registers for interface with CPU
Buffer for streaming from/to device
CPU and device compete for bus cycles for memory access

Data transfer speeds

Device	Input/ output	Human in the loop	Data rate (circa 2008)	Data rate (circa 2020)	PIO	DMA
Keyboard	Input	Yes	5-10 bytes/sec	5-10 bytes/sec	X	
Mouse	Input	Yes	80-200 bytes/sec	80-200 bytes/sec	X	
Graphics display	Output	No	200-350 MB/sec	200-350 MB/sec		X
Disk (hard drive)	I/O	No	100-200 MB/sec	500 MB/sec (each)		X
Network (LAN)	I/O	No	1 Gbit/sec	1-400 Gbit/sec		X
Modem	I/O	No	1-8 Mbit/sec	1-8 Mbit/sec		X
Inkjet printer	Output	No	20-40 KB/sec	20-40 KB/sec	X	X
Laser printer	Output	No	200-400 KB/sec	200-400 KB/sec		X
Voice (microphone/ speaker)	I/O	Yes	10 bytes/sec	10 bytes/sec	X	
Solid State (SSD)	I/O	No	10-50 MB/sec	0.1-7GB/sec		X
CD, DVD, Blu-Ray	I/O	No	10-20 MB/sec	10-20 MB/sec		X

DMA is used instead of PIO for I/O because

14% A. I just want the participation credit

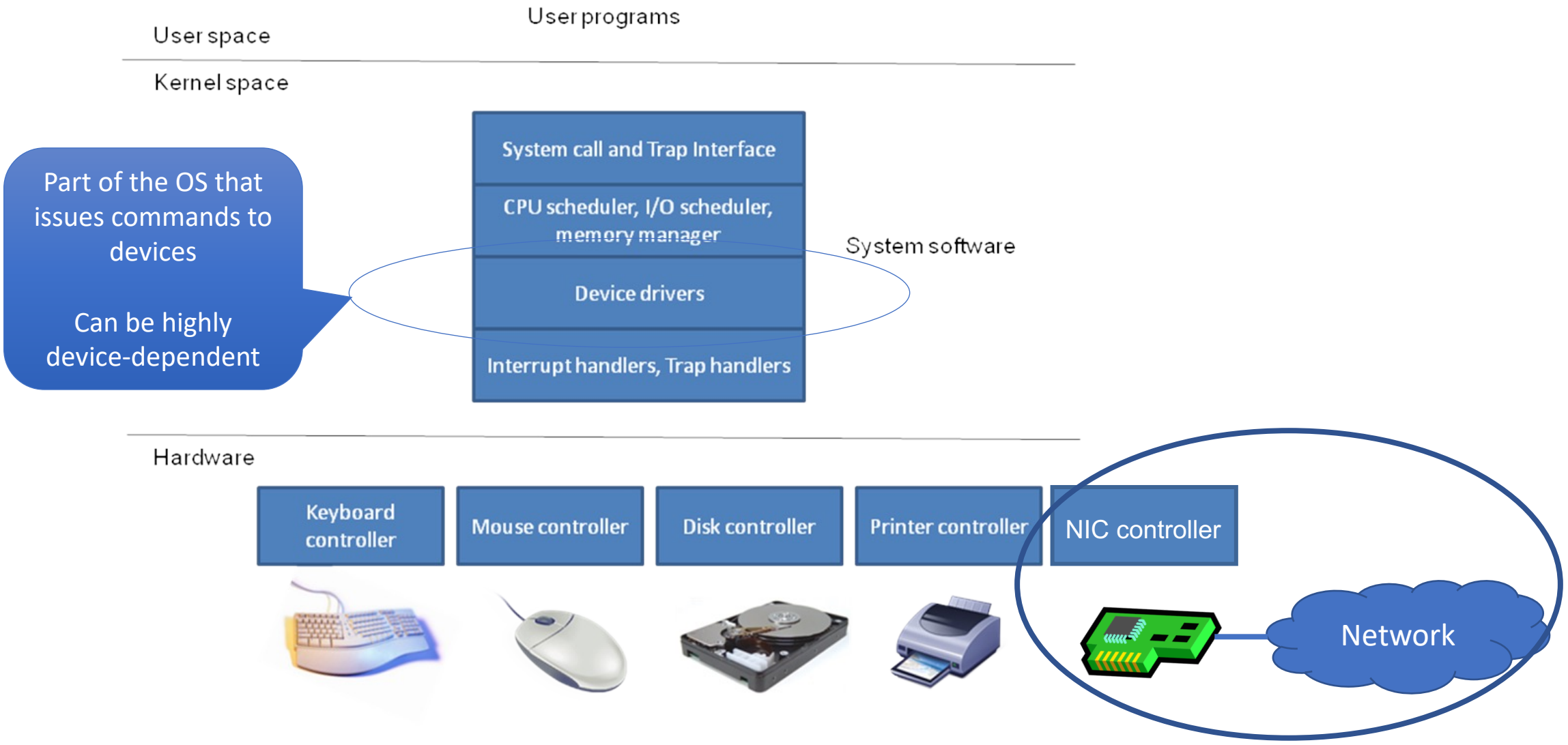
14% B. PIO can lose data if the CPU doesn't poll or respond to the interrupt quickly enough

21% C. Servicing an interrupt in the CPU is expensive especially if it has to be done for every word of I/O

21% D. A DMA controller can transfer data into memory in fewer cycles than a programmed loop in the CPU

29% E. All of the above

Device drivers and OS



Roadmap

- ~~Finish up parallel systems (Chapter 12)~~
  - ~~Hardware support for SMP~~

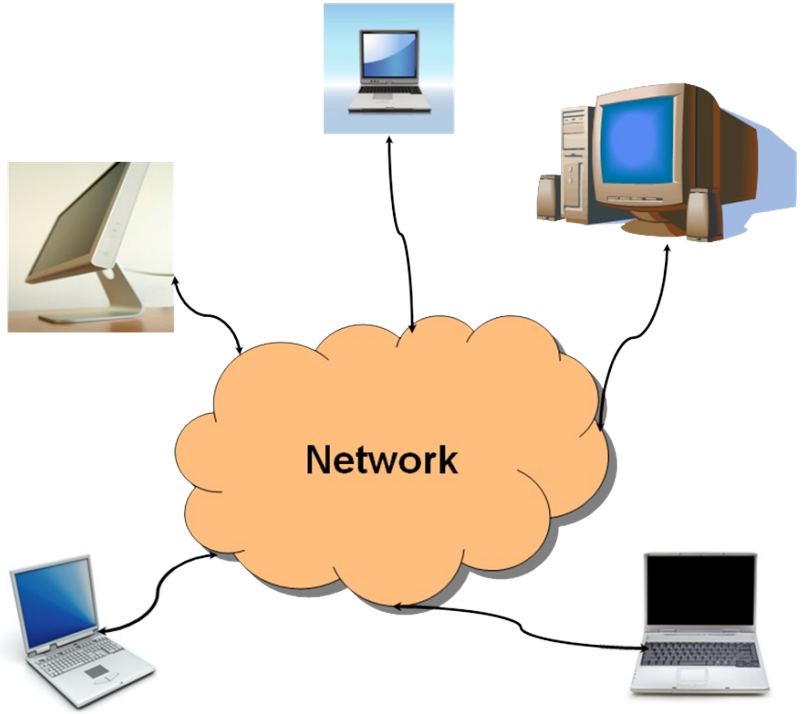
- ~~Programmed IO and DMA~~
  - ~~Chapter 10 (10.1 – 10.7)~~

- Networking
  - Chapter 13

Network of computers

- A computer connected to a network is called a host
- The connection is made using a device called a *Network Interface Card* or *NIC*

- What exactly is “the network"?
- It's a big collection of interconnected networks that all use the same protocols to communicate, hence "the internet"



The traditional way

- What is the *Internet*? Consider the postal system…
- Clio sends a birthday letter to her grandmother

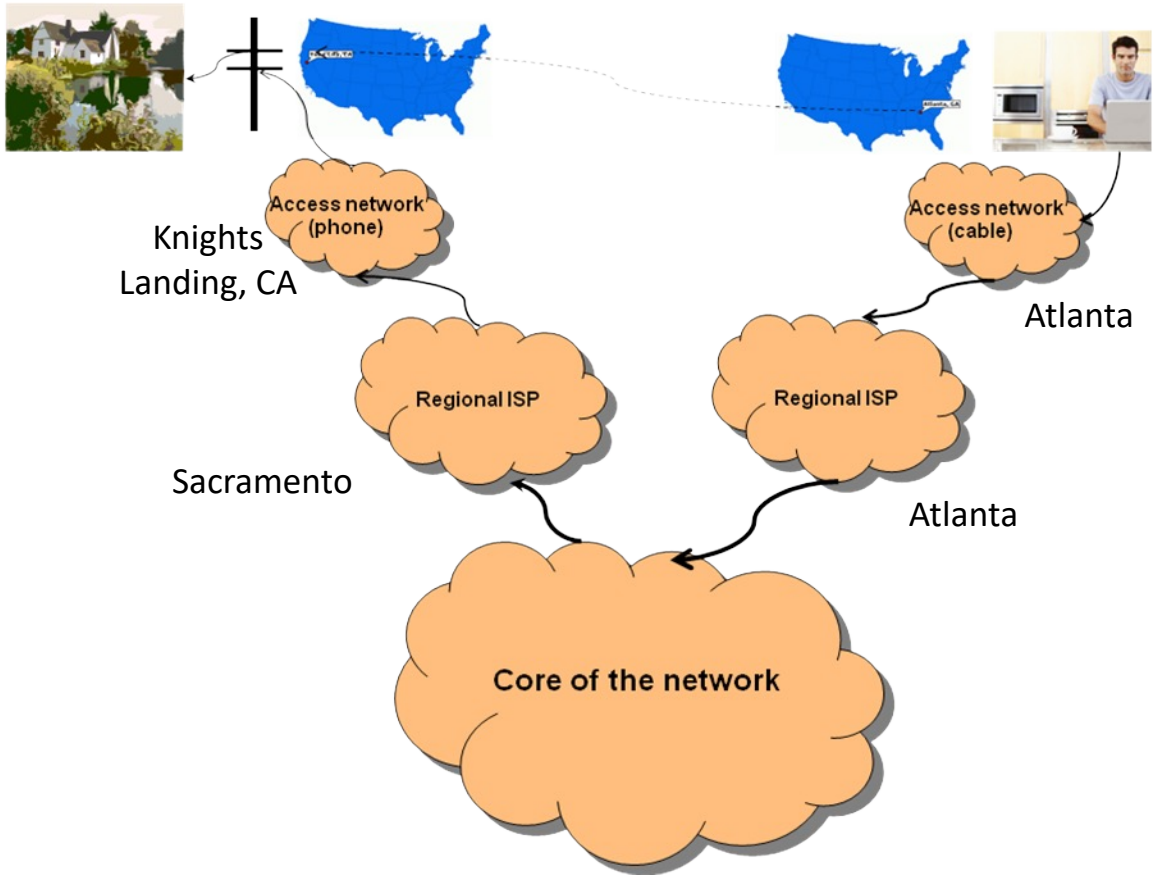
Atlanta

Athens, Greece

Koroni, Peloponnese

The modern way

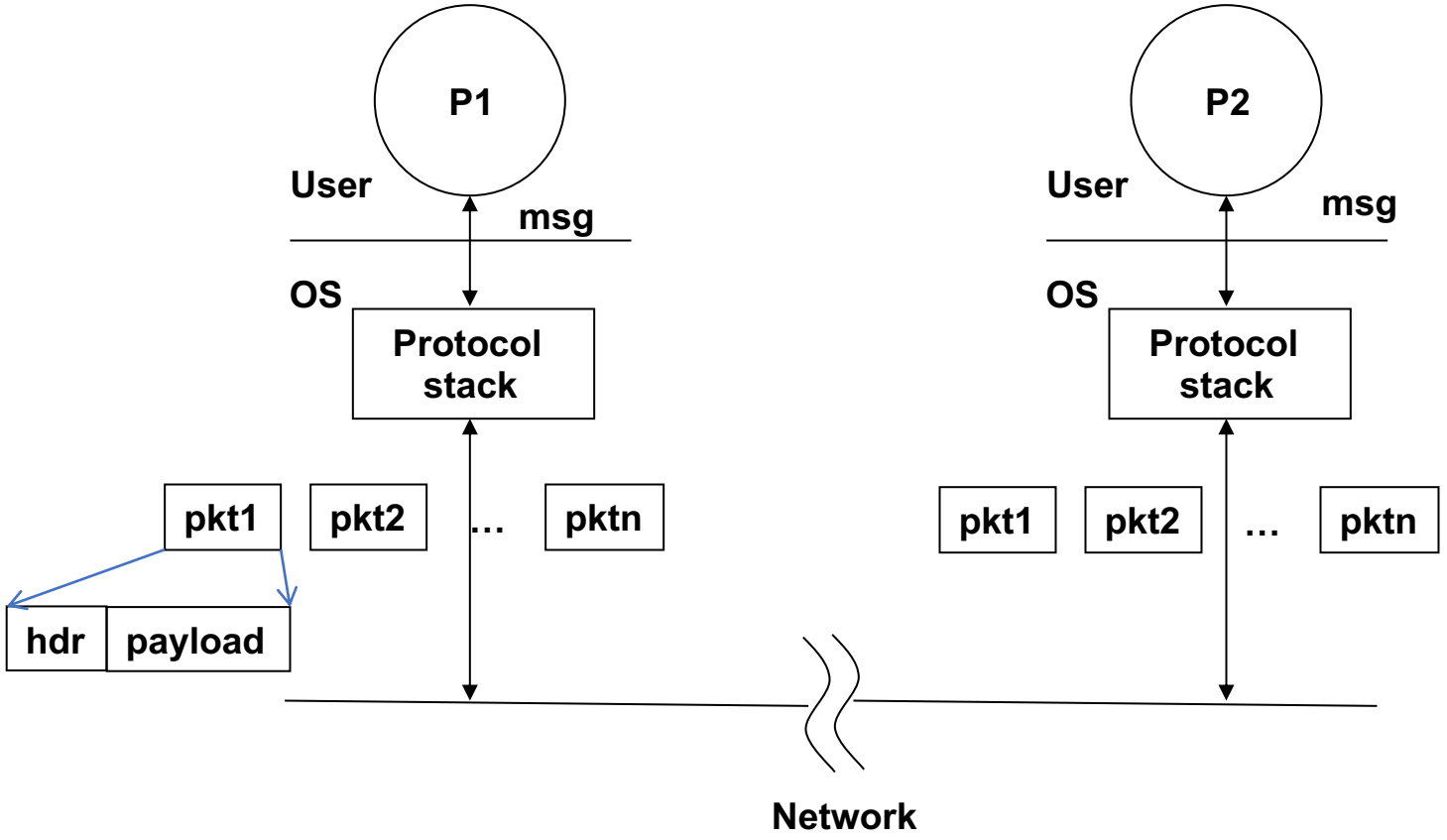
- Now consider an email from Joe to his grandmother



Commonality

- What do those two paradigms have in common?
- Universally understood addresses!
  - A country and a postal code
    - The staff at the Atlanta regional post office probably have no idea where Koroni, Peloponnese, Greece is.
    - But they know how to get mail to Greece and specifically to deliver mail for the postal code for the Peloponnese region via the capital Athens
    - Once the mail gets to Athens, the staff there know how to get it to Peloponnese, then Koroni
  - An IP address
    - The routers at the ISP in Atlanta probably have no idea where Knights Landing, CA is, but they know how to get packets to the internet core
    - The internet core does know which ISP services the IP addresses in Knights Landing, CA
    - The message goes to a mail server at the ISP and waits for Joe's grandmother to dial up her internet connection
- Key point
  - No individual has to know where all the addresses are located, but at each hop they know where to forward an item to get it a hop closer

IP network connections



Big deal: Note that the Protocol Stacks are in the hosts, not in the middle of the Network

Contrast: Datagram or VC network

Internet (Datagram)

- data exchange among computers
  - "elastic" service, no strict timing req.
- many link types
  - different characteristics
  - uniform service difficult
- "smart" end systems (computers)
  - can adapt, perform control, error recovery
- ***simplicity inside network layer, complexity at edge***

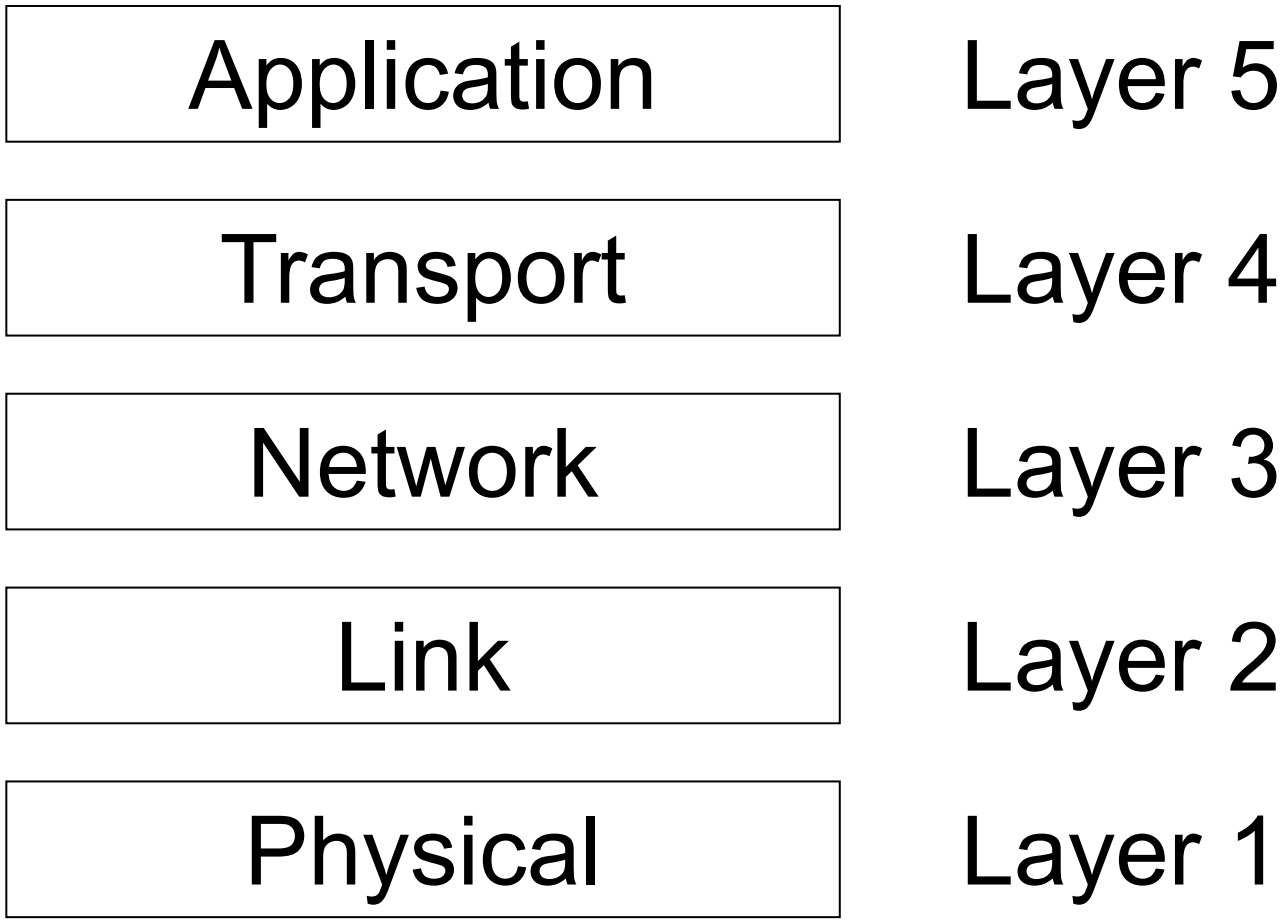
ATM (Virtual Circuits)

- evolved from telephony
- human conversation:
  - strict timing, reliability requirements
  - need for guaranteed service
- "dumb" end systems
  - telephones
- ***complexity inside network layer, simplicity at edge***

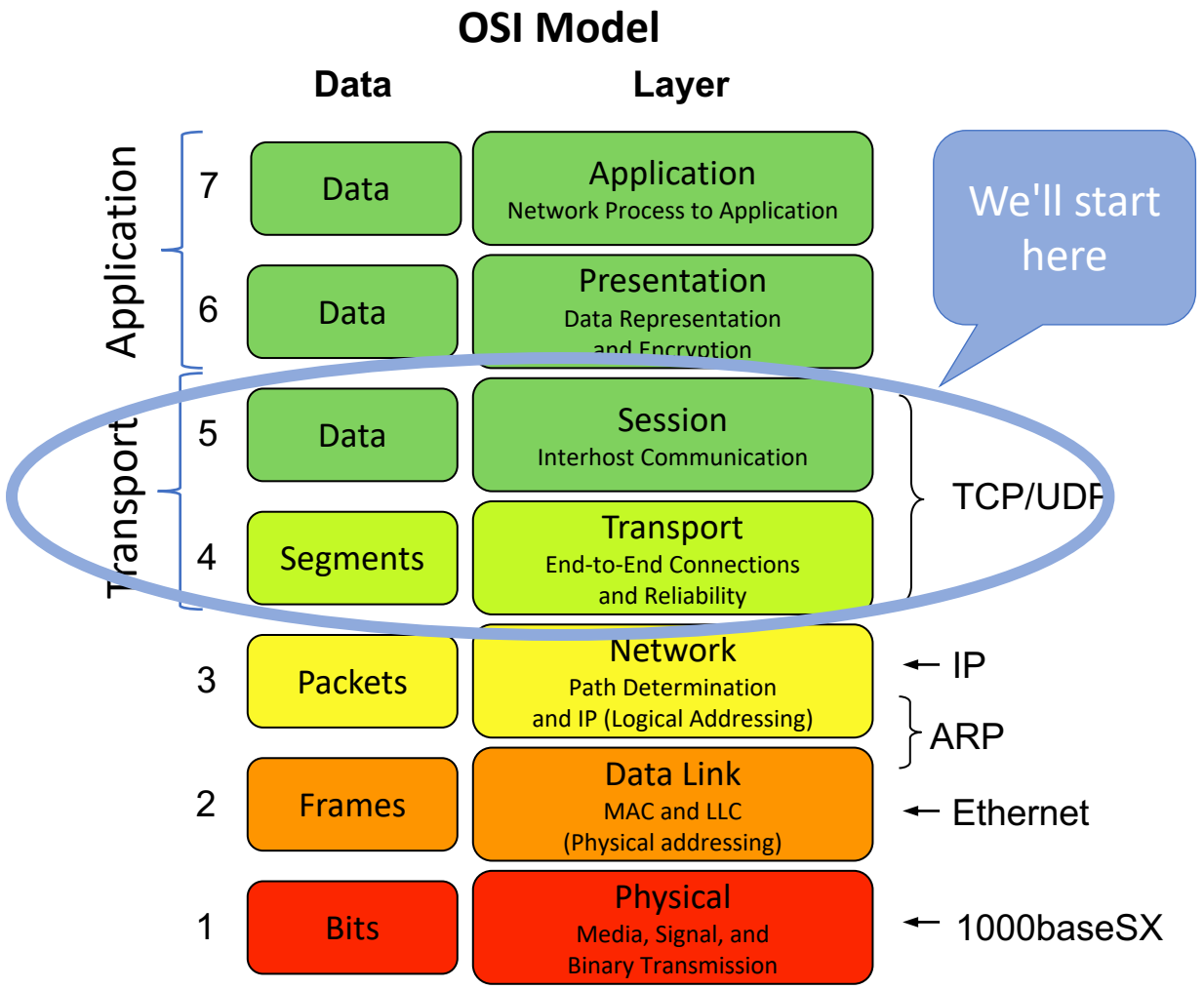
Why does IP need a protocol stack?

- Arbitrary message sizes
- Out of order delivery
- Packet loss
- Bit errors
- Queuing delays

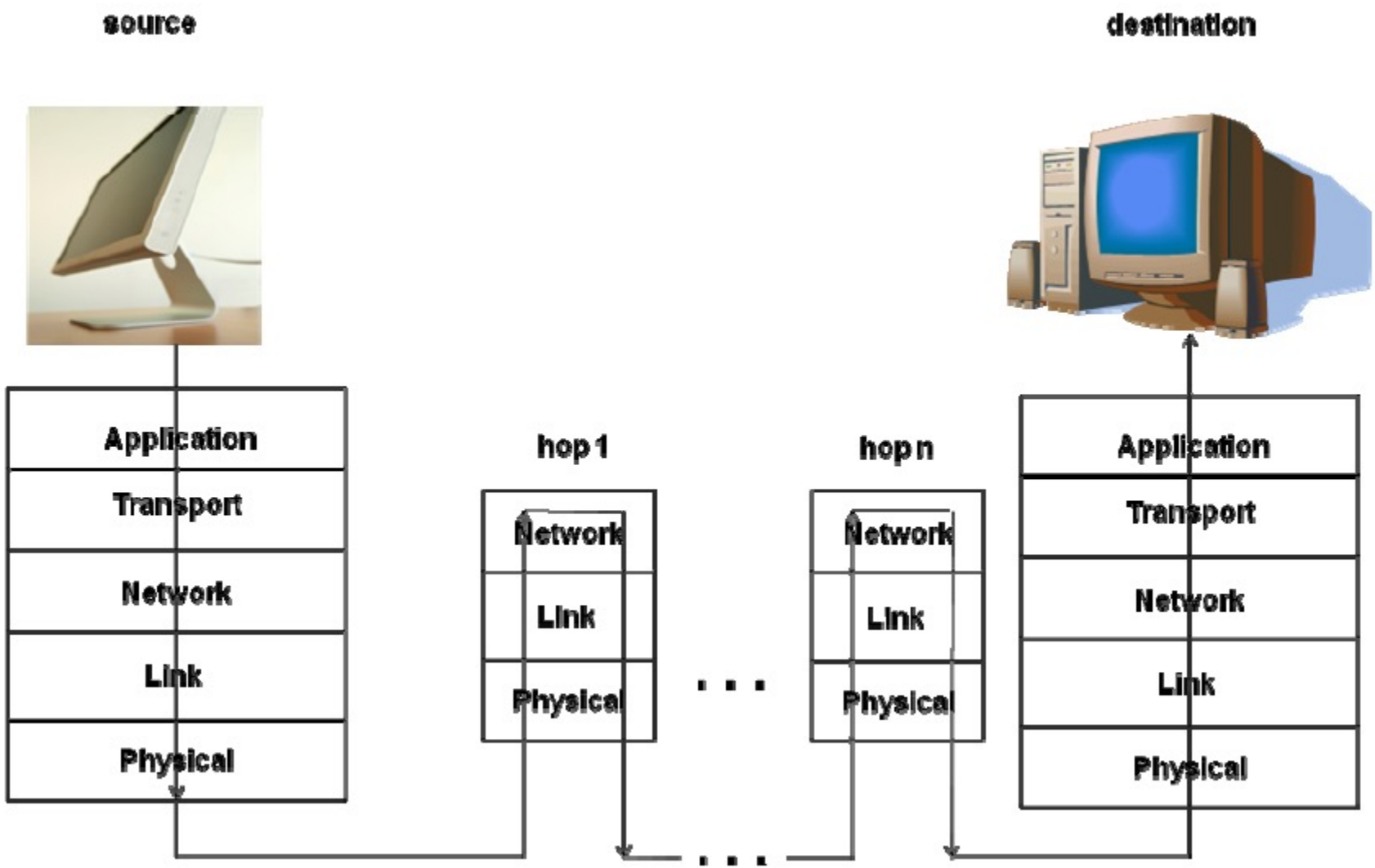
Internet Protocol Stack



The OSI Model



Getting from here to there…

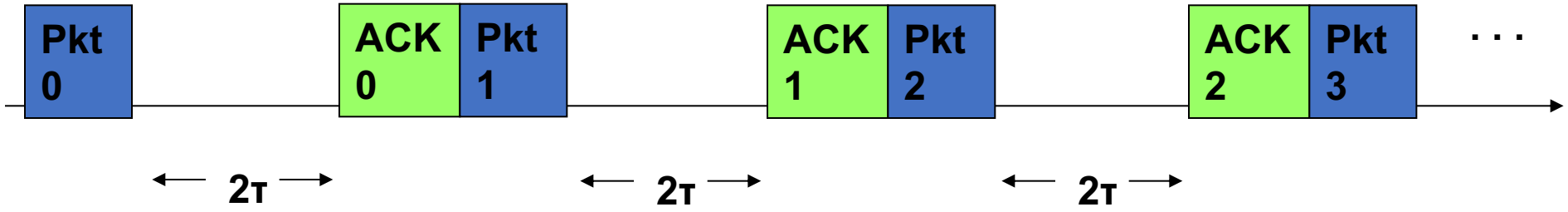


Transport (session) protocol

We’re going to talk about these protocol types

- Stop and wait
- Pipelined transmission
- Reliable pipelined transmission

Stop and wait (for ack)



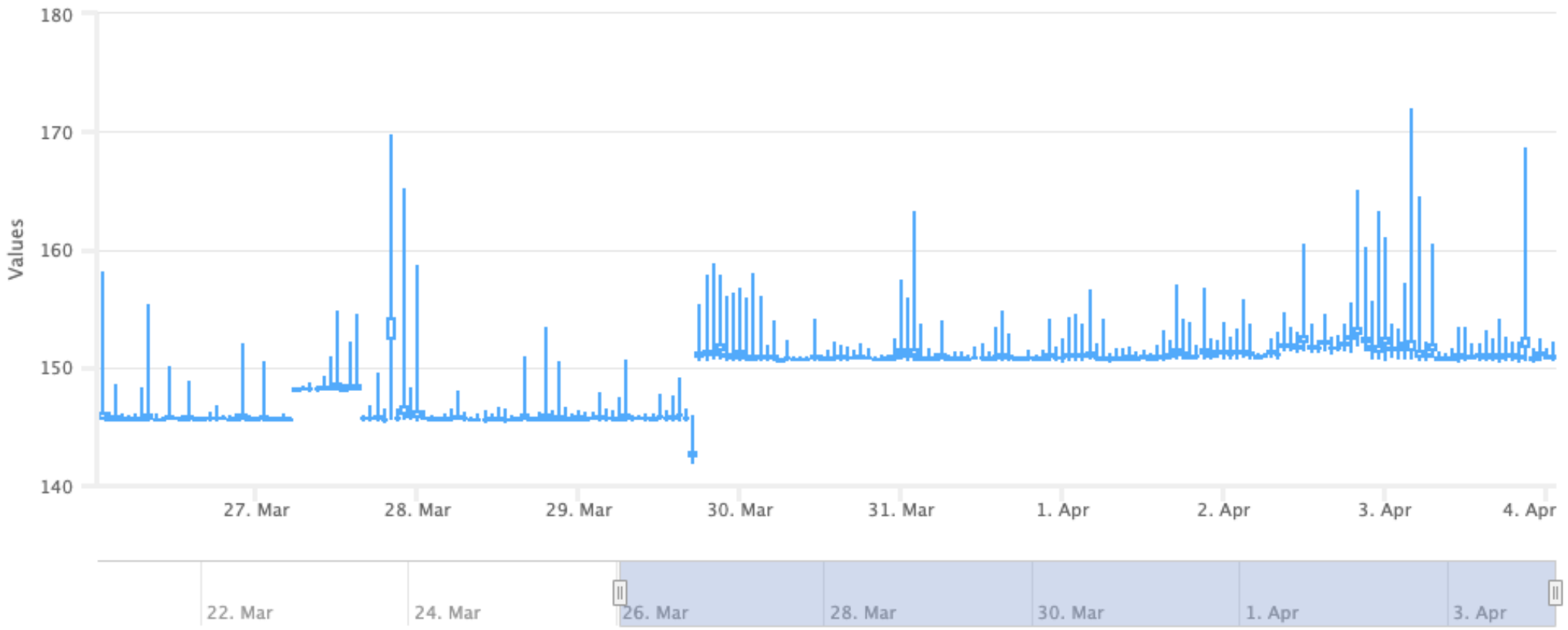
Timeline of sender

Latency

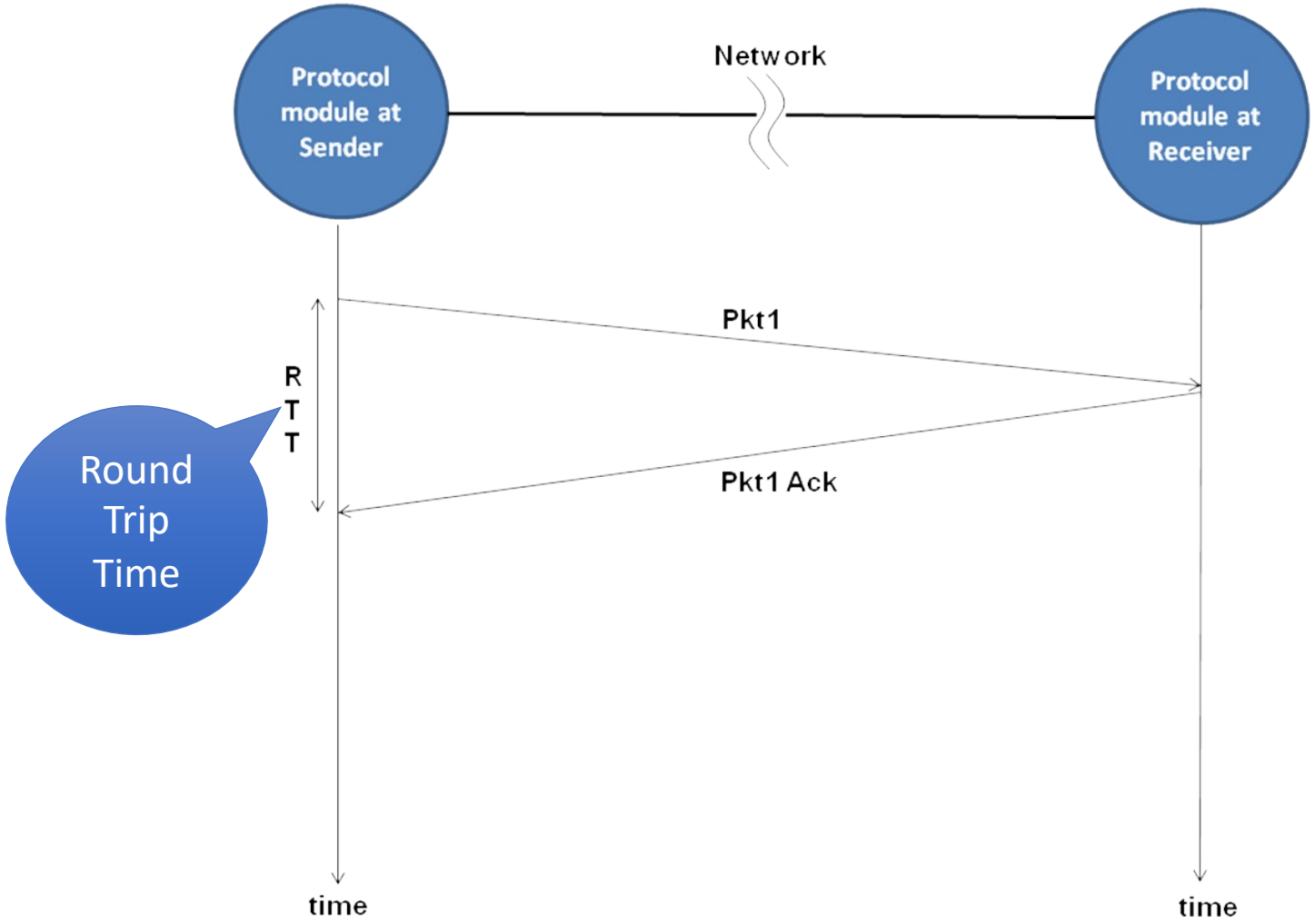
Greece

Global Ping Statistics → Atlanta and Athens

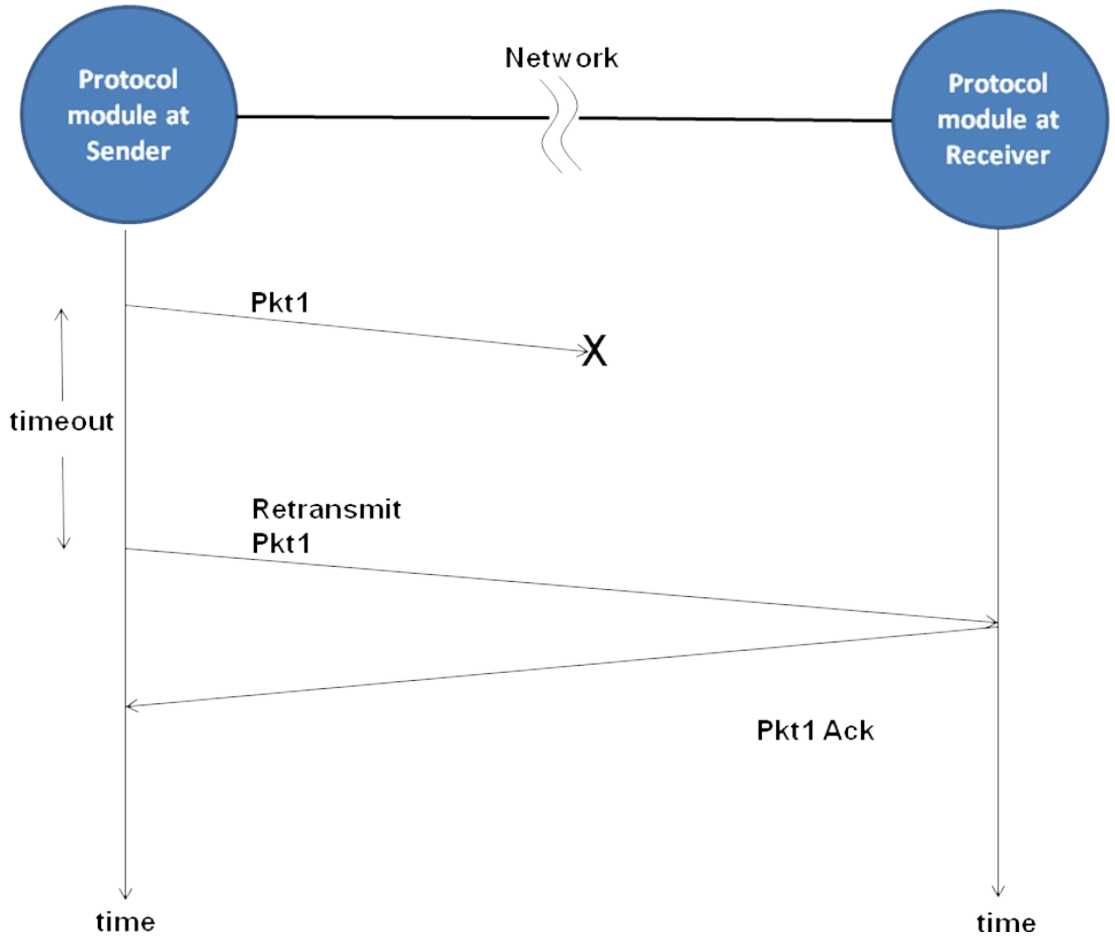
Showing historical ping data between major cities. Bar shows Average ± Median Deviation.



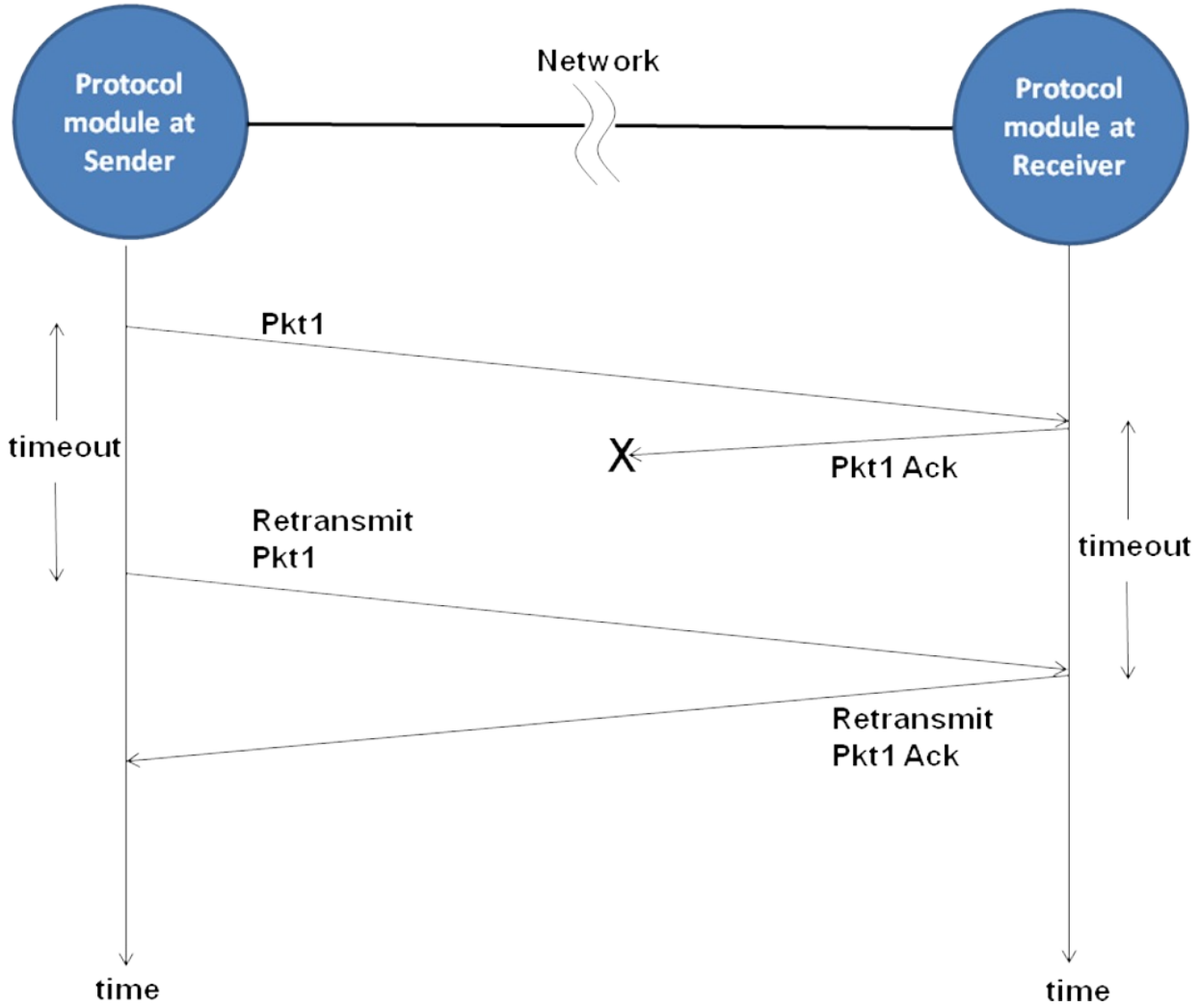
One round trip



Lost packet


Protocol module at Sender
Network
Protocol module at Receiver
Pkt1
X
timeout
Retransmit Pkt1
Pkt1 Ack
time
time

Lost ack


Protocol module at Sender
Network
Protocol module at Receiver
Pkt1
timeout
X
Pkt1 Ack
Retransmit Pkt1
timeout
Retransmit Pkt1 Ack
time
time

More details

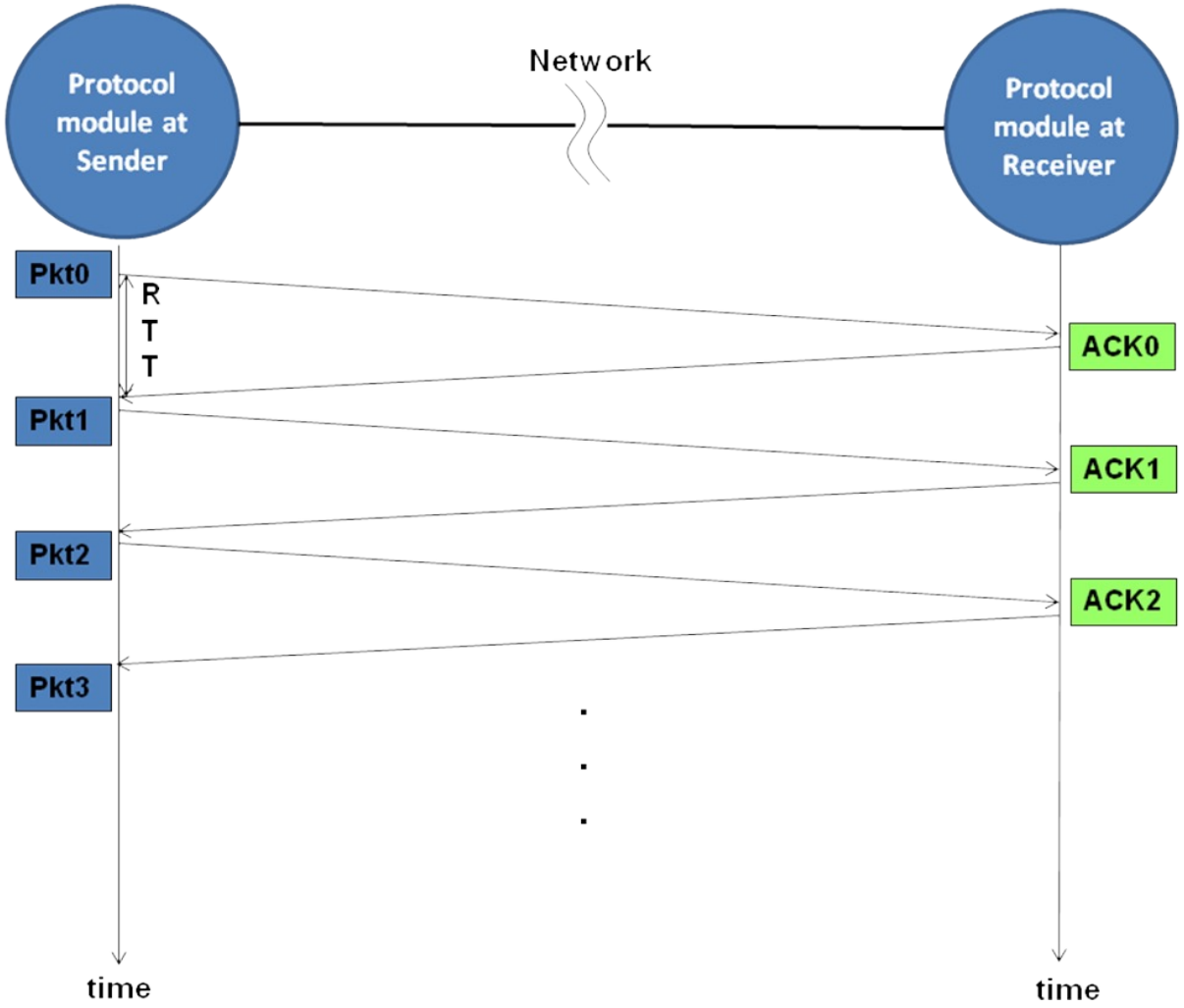
- How much buffer does the stack need at the sender?
  - Just 1
- How much buffer does the stack need at the receiver?
  - Just 1
- How do I know at the receiver that a packet I'm getting is not a duplicate
  - Include a sequence number
- How big should a sequence number be?
  - It might only need to be 1 bit if there's only one message in flight

Assumptions

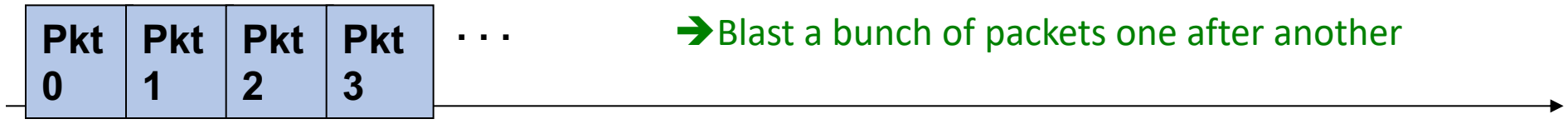
- In our stop-and-wait protocol, we make some assumptions
  - Packet loss is possible
  - Packets don't get reordered or arbitrarily delayed (how can they, if only one packet is in flight)
  - So we can get away with a 1-bit sequence number
- However, there are more fundamental assumptions in a packet-switched network
  - Packet loss is possible (i.e. best-effort delivery)
  - Packets may be reordered or arbitrarily delayed
  - Packets may be damaged in transit
- These additional assumptions mean that a 1-bit sequence number isn't going to hold up on the internet
  - So we use monotonically increasing sequence numbers

Working at top speed!

- Remember Latency?
- 150 ms to Athens
- RTT = 150ms
- ~7 packets per second throughput?
- No matter how fast the network?



Pipelined protocol

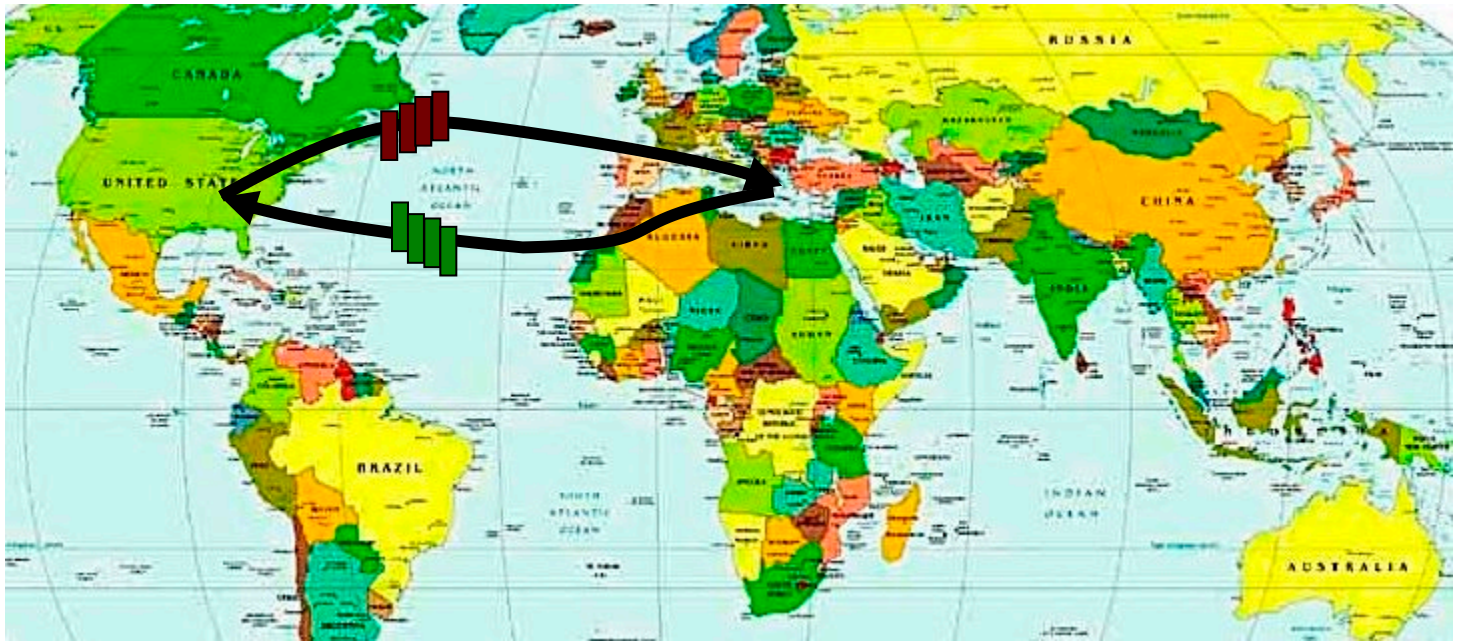


➔Blast a bunch of packets one after another

What could go wrong?

Timeline of sender

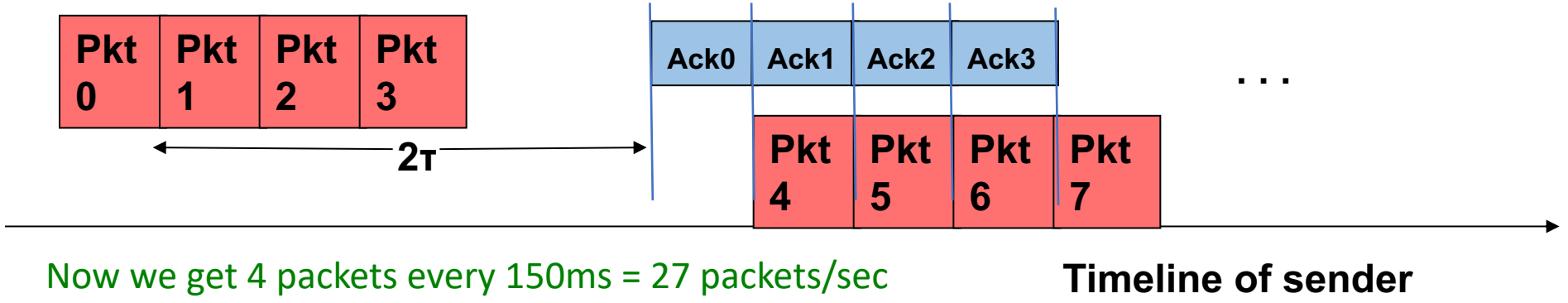
Reliable pipelined protocol



Sometimes some packets don't make it all the way!

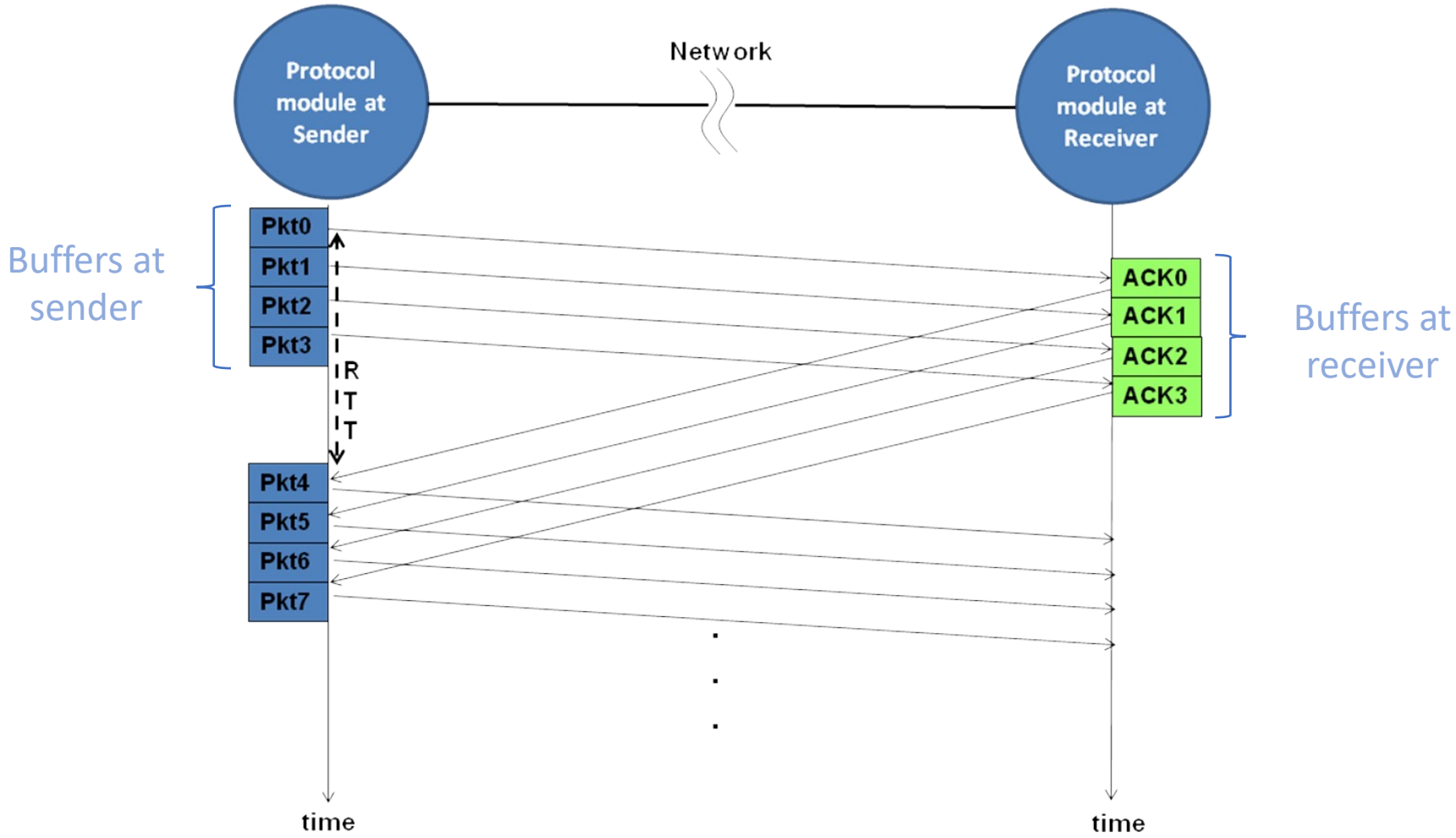
We have to keep track

Reliable pipelined protocol



Now we get 4 packets every 150ms = 27 packets/sec

Reliable pipelined protocol



How many packet buffers does the sender need for stop-and-wait protocol?

A. I just want the participation credit
B. 1
C. 2
D. Designer's choice
E. No way to predict in advance

How many?

How many packet buffers does the sender need to implement a reliable pipelined protocol?

A. I just want the participation credit
B. 1
C. 2
D. One buffer for each packet in the pipeline
E. Protocol designer's choice

What might a packet look like?

```
struct header_t {
  int destination_address;  /* destination address */
  int source_address;       /* source address */
  int num_packets;          /* total number of packets in the message */
  int sequence_number;      /* sequence number of this packet */
  int packet_size;          /* size of data contained in the packet */
  int checksum;             /* for integrity check of this packet */
};
```

```
struct packet_t {
  struct header_t header;   /* packet header */
  char *data;               /* pointer to the memory buffer containing the data
                               of size packet_size  */
};
```

Transport layer concerns

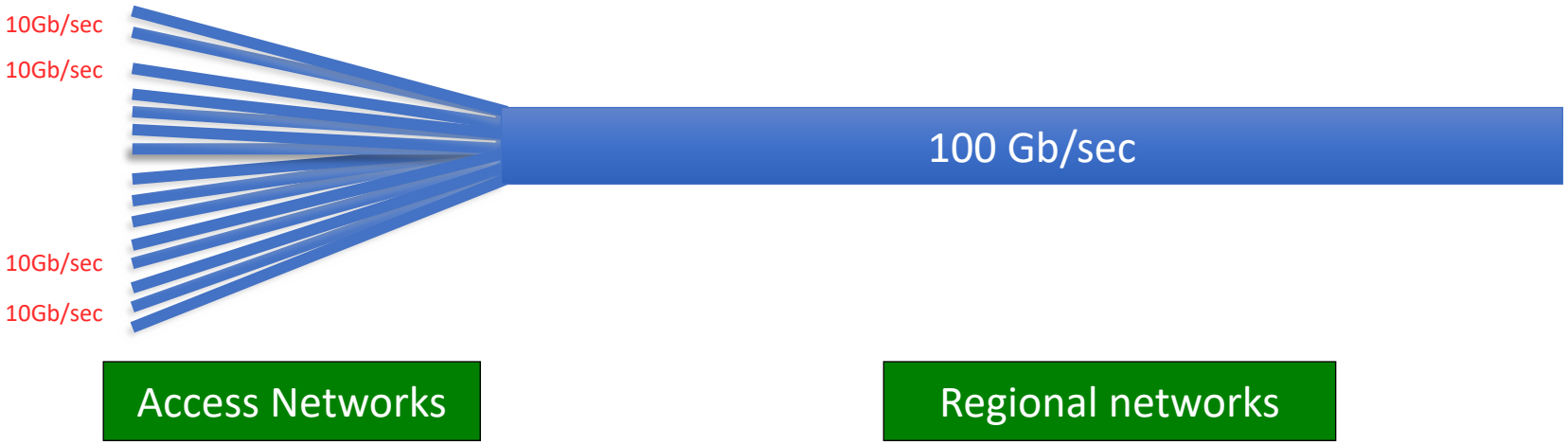
- Arbitrary message size
- Out of order delivery
- Packet loss
- Bit errors
- Queuing delays

How do we address these?

Transport layer concerns

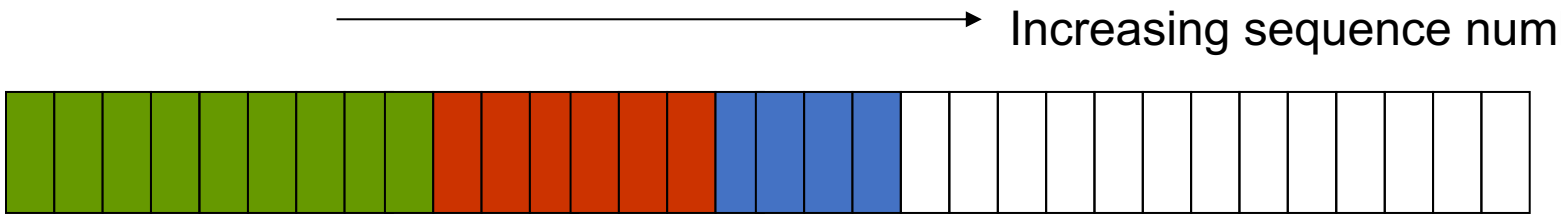
- Arbitrary message size ➔ Big messages use multiple packets
- Out of order delivery ➔ Sequence numbers
- Packet loss ➔ Positive ACK and buffers
- Bit errors ➔ Checksum for packet
- Queuing delays ➔ But how to handle this one?

Network congestion



- Networks tend to grow
- Network connections get overcommitted
- Packets are queued, but get so stale they must be dropped
- Because of the sequence number, we can ignore heavily delayed packets if they ever make it

Reliable protocol with windowing



Active window of sequence numbers

Packets sent and acknowledged

Packets sent but not yet acknowledged

Packets that are in the active window that can be sent without waiting for any further ACKS

Packets that cannot yet be sent since they are outside the active window

Congratulations! You've now been introduced to TCP from the IPv4 protocol

Transport protocols on the Internet

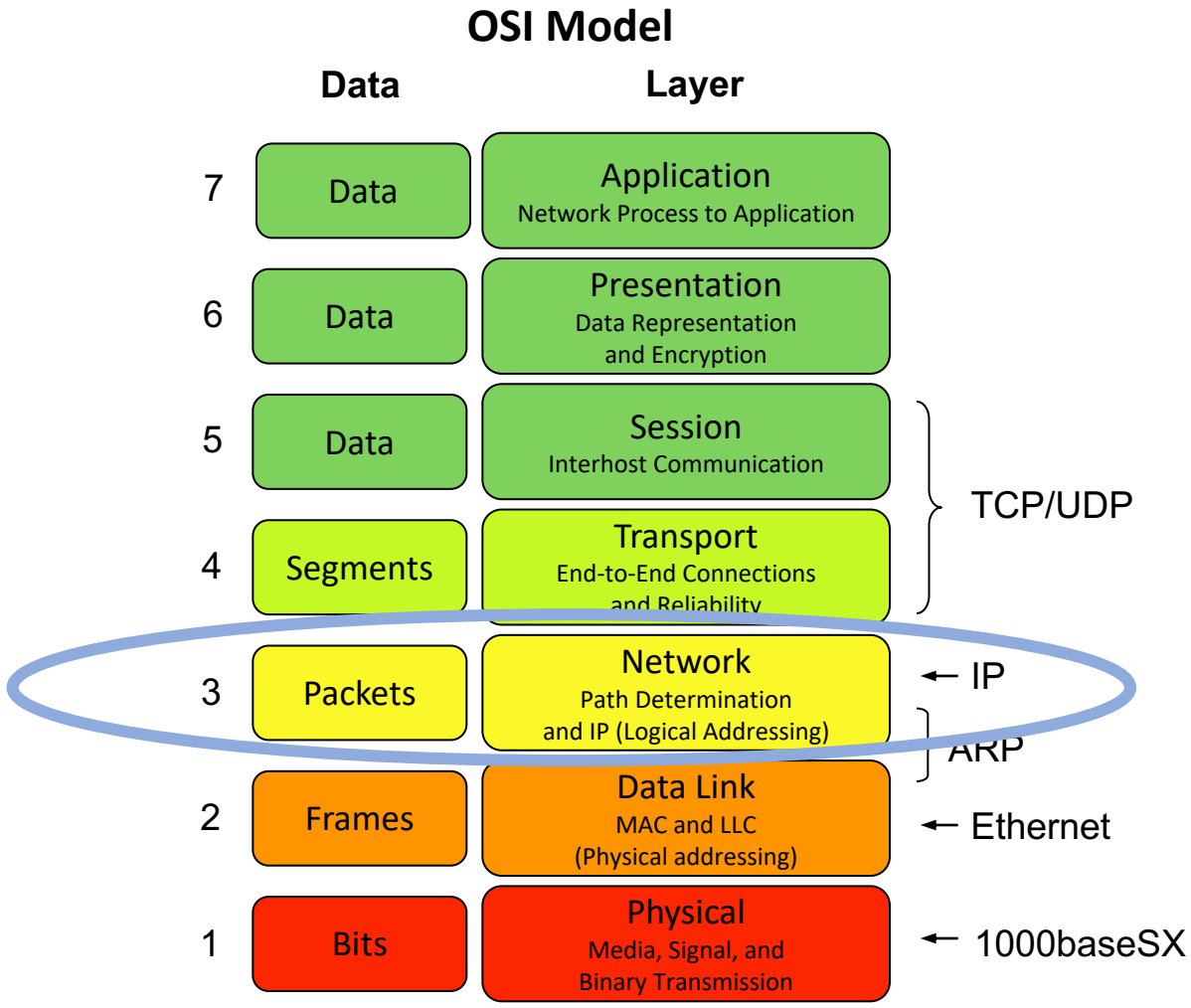
Transport protocol	Features	Pros	Cons
TCP	Connection-oriented; self-regulating; data flow as stream; **supports windowing and ACKs**	Reliable; messages arrive in order; well-behaved due to self-policing	Complexity and extra latency in connection setup and tear-down; at a disadvantage when mixed with unregulated flows; no guarantees on delay or transmission rate
UDP	Connection-less; unregulated; message as datagram; **no ACKs or windowing**	Simplicity; no frills; especially suited for environments with low chance of packet loss and applications tolerant to packet loss;	Unreliable; message may arrive out of order; may contribute to network congestion; no guarantees on delay or transmission rate

Choice of transport protocols

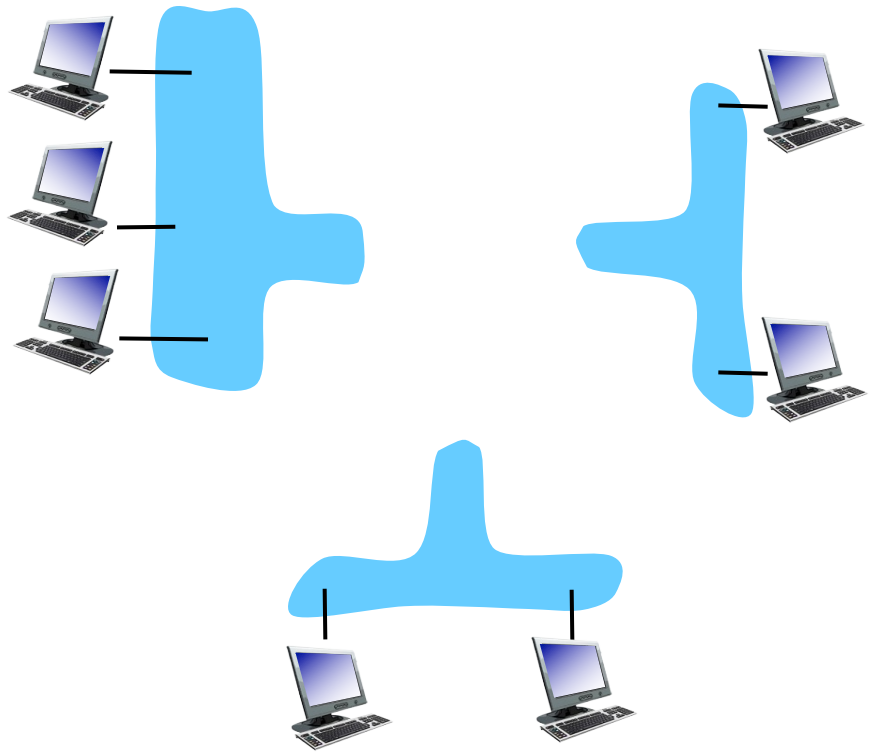
Application	Key requirement	Transport protocol
Web browser	Reliable messaging; in order arrival of messages	TCP
Instant messaging	Reliable messaging; in order arrival of messages	TCP
Voice over IP	Low latency	Usually UDP
Electronic Mail	Reliable messaging	TCP
Electronic file transfer	Reliable messaging; in order delivery	TCP
Video over Internet	Low latency	Usually UDP; may be TCP
File download on P2P networks	Reliable messaging; in order arrival of messages	TCP
Network file service on LAN	Reliable messaging; in order arrival of messages	TCP; or reliable messaging on top of UDP
Remote terminal access	Reliable messaging; in order arrival of messages	TCP

Now let's look lower

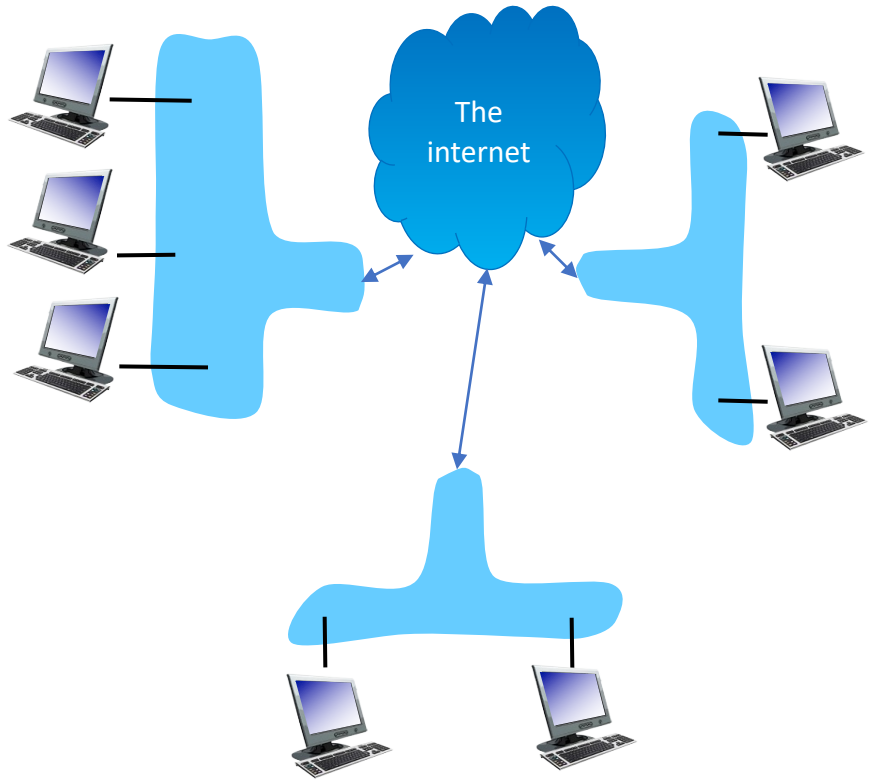
- IP lives at the network layer
- Let's look specifically at IP as a layer 3 as well as its relationship to the Transport (Session) layers



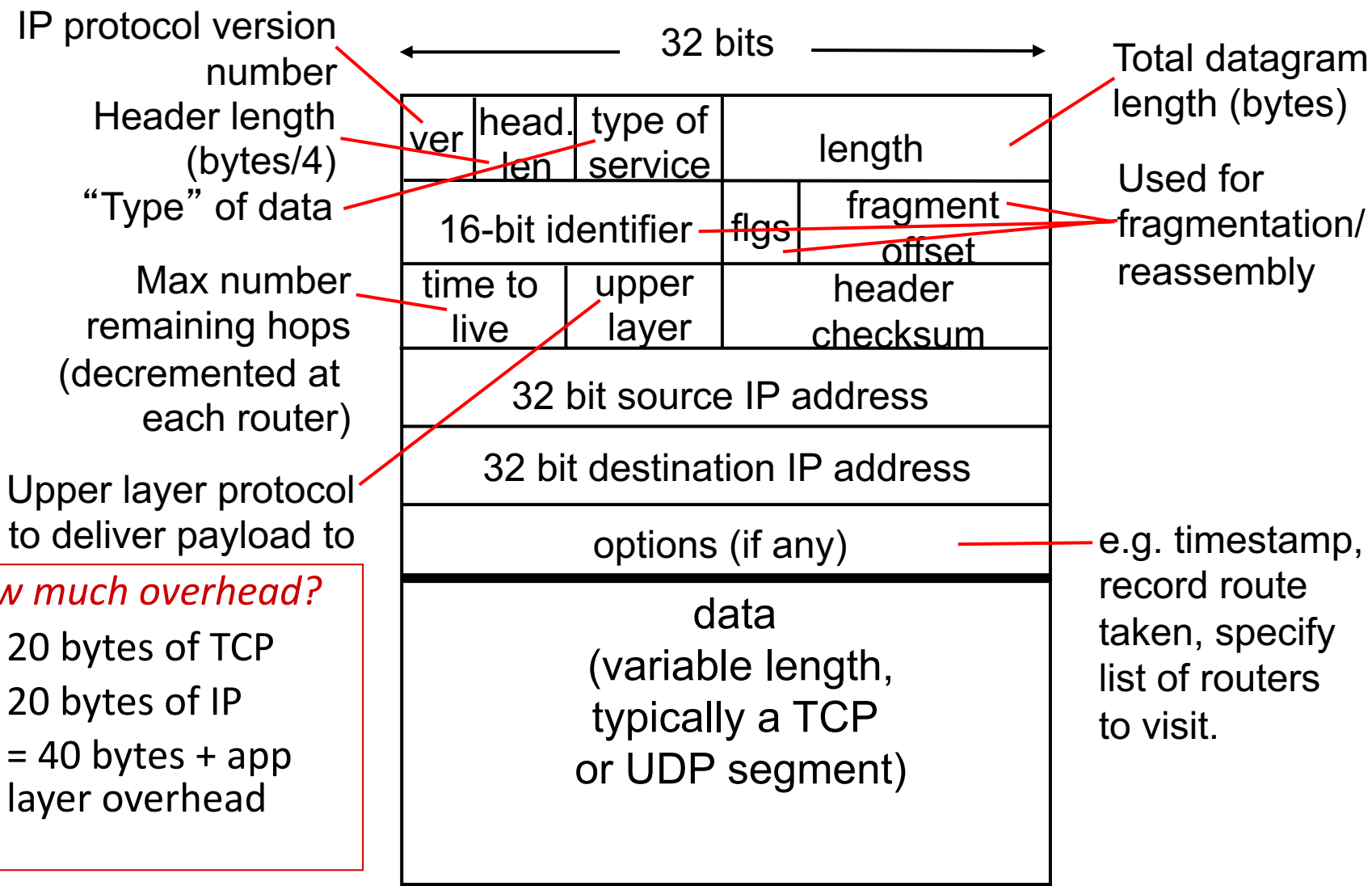
- No public internet
- Islands of connectivity
  - Data link layer networks all over, even in 1983
    - Ethernet
    - 802.11 wireless
    - Token Ring
    - AppleTalk
    - RS232 serial
    - ATM
    - And plenty more
- So how do we build a network of networks?



- Build a network layer that can address and interconnect all sorts of different data link networks
- After a decade of innovation and another decade of growth
- IP for everybody (version 4 that is) with two built-in transport protocols, TCP and UDP



IP datagram (packet) format

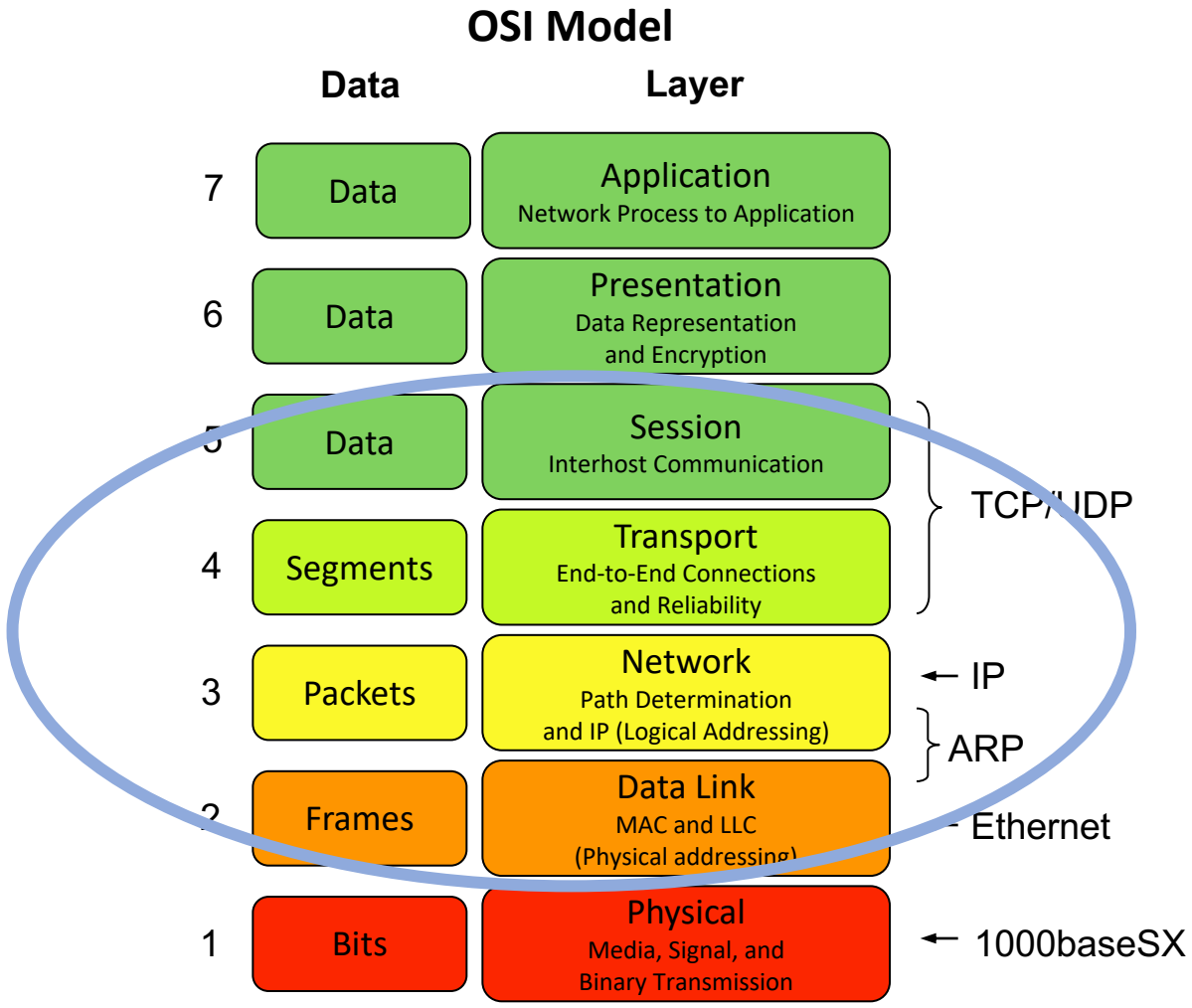


how much overhead?

- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

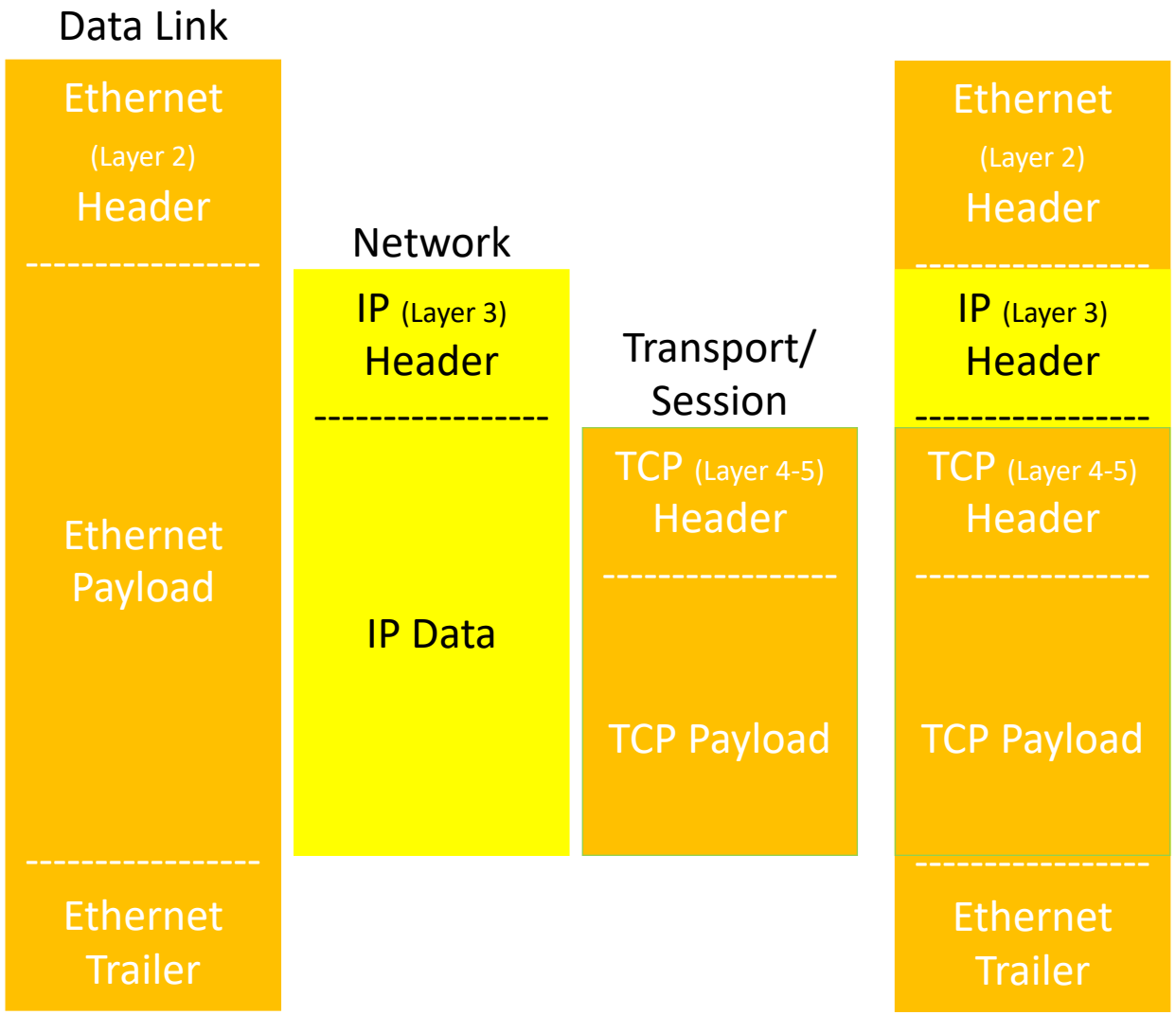
Visible relations between layers

- Layers 2 through 5 are manifested in the packets we send on the network
- (Layers 1, 6, and 7 are too, but showing them makes this too complicated)
- Let's see how…



Encapsulated packets

- The network protocol is designed in layers
- Thus, the lower-level packets encapsulate (enclose) the higher-level packets



Remember: Layer 2 doesn't have to be ethernet and layer 4-5 doesn't have to be TCP

IPv4 Header

IPv4 Header Format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															
24	192																																
28	224																																
32	256																																

TCP Segment Header

TCP segment header

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0			N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size																
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

Fragmentation

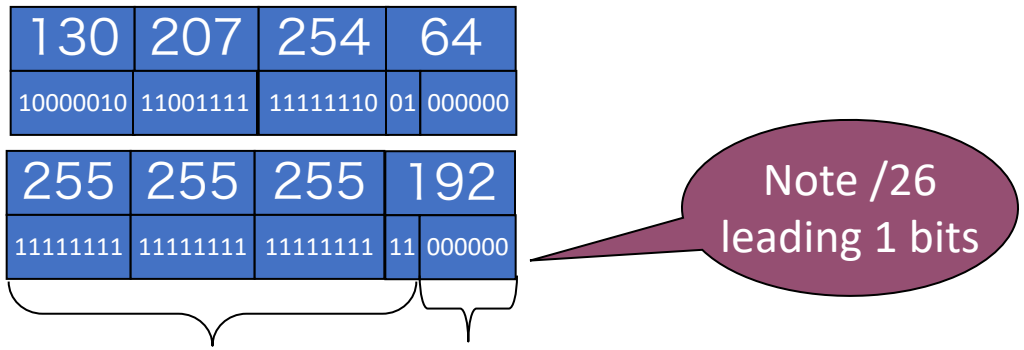
- One way to deal with segments that are too big to fit in a single packet
- Probably should be considered deprecated these days
- IPv6 doesn’t allow it

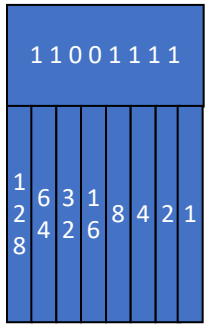
IPv4 addresses

- 32 bits / 4 octets / four-decimal dotted quad
  - Did you know that 130.207.7.31 becomes 0x82CF071F?
  - Each of those four numbers is decimal for a single byte
- And we need a mask
- We use it to divide the IP address into a "network" part and a "host" part
  - It's a 32-bit number that starts with 1s and ends with 0s
  - The network part is "under" the 1s in the mask
  - E.g., mask 255.255.255.0 indicates we are looking for host 31 in network 130.207.7.0

Subnet and CIDR Notation

- Two different notations, same meaning:
  - 130.207.254.64 255.255.255.192
  - 130.207.254.64/26
- In this example, the network part is the first 26 bits, host is last 6
- If you're doing this every day, you learn to do it in your head; if not, use a subnet calculator, e.g. http://www.subnet-calculator.com/cidr.php
- Cisco equipment often uses the former notation; many other modern tools use the latter





What does it mean to be on the same network?

- In the IP world it's simple
  - If network bits between two addresses are equal, they are on the same network
  - The means the data link layer will be called on to deliver the packet

Source IP address	Destination IP address	Mask	On the same network?
130.207.7.23	130.207.254.16	/16 (255.255.0.0)	Yes
192.168.1.2	192.168.2.2	/24 (255.255.255.0)	No
0x81880507	0x818910F1	/15 (255.254.0.0)	Yes
172.16.32.12	10.0.5.2	/16 (255.255.0.0)	No

Are these two IP addresses on the same network?

143.215.14.243 and 143.214.14.242 if the mask for each is /24

A. I just want the participation credit
B. Yes
C. No
D. Can't tell
E. All of the above

Keeping the first 24 bits of the two addresses, we get

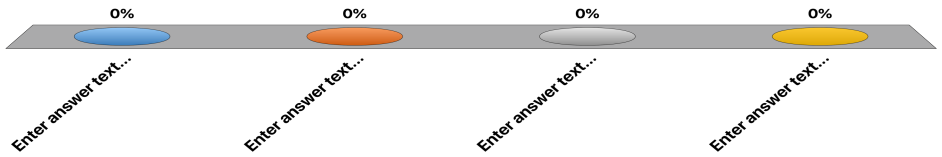
143.215.14.0

and

143.214.14.0

Are they the same number?

Nope.



Are these two IP addresses on the same network?

143.215.14.243 and 143.214.14.242 if the mask for each is /15

A. I just want the participation credit
B. Yes
C. No
D. Can't tell
E. None of the above (B to D)

Keeping the first 15 bits of the two addresses, we get

143.214.0.0 (215=11010111_2; first 7 bits are 11010110_2)

and

143.214.0.0 (214=11010110_2; first 7 bits are 11010110_2)

Are they the same number?

Yes!