

Synchronization and Flow Control

A Wide Range of Possible I/O Rates

Device	Behaviour	Partner	Data Rate (KB/s)
Keyboard	Input	Human	0.01
Mouse / tablet	Input	Human	0.02
Voice Input	Input	Human	0.02
Scanner	Input	Human	200.00
Voice Output	Output	Human	0.60
Line Printer	Output	Human	1.00
Laser Printer	Output	Human	100.00
CRT	Output	Human	30,000.00
Network Terminal	I/O	Machine	0.05
Network LAN	I/O	Machine	200.00
Floppy Disc	Storage	Machine	50.00
CD-ROM	Storage	Machine	500.00
Magnetic Tape	Storage	Machine	2000.00
Hard Disc	Storage	Machine	2000.00

Source: Computer Organization & Design: The Hardware/Software Interface,
by David A. Patterson and John L. Hennessy, (Morgan Kaufmann, 1994), p. 513.

Mismatch between CPU and I/O Devices

Observation: A ColdFire clocked at 150 MHz executes roughly 50 million instructions per second.

Example #1: Keyboard input

maximum data rate = 0.01 KB/s = 10 bytes / sec
= 10 ASCII characters / second

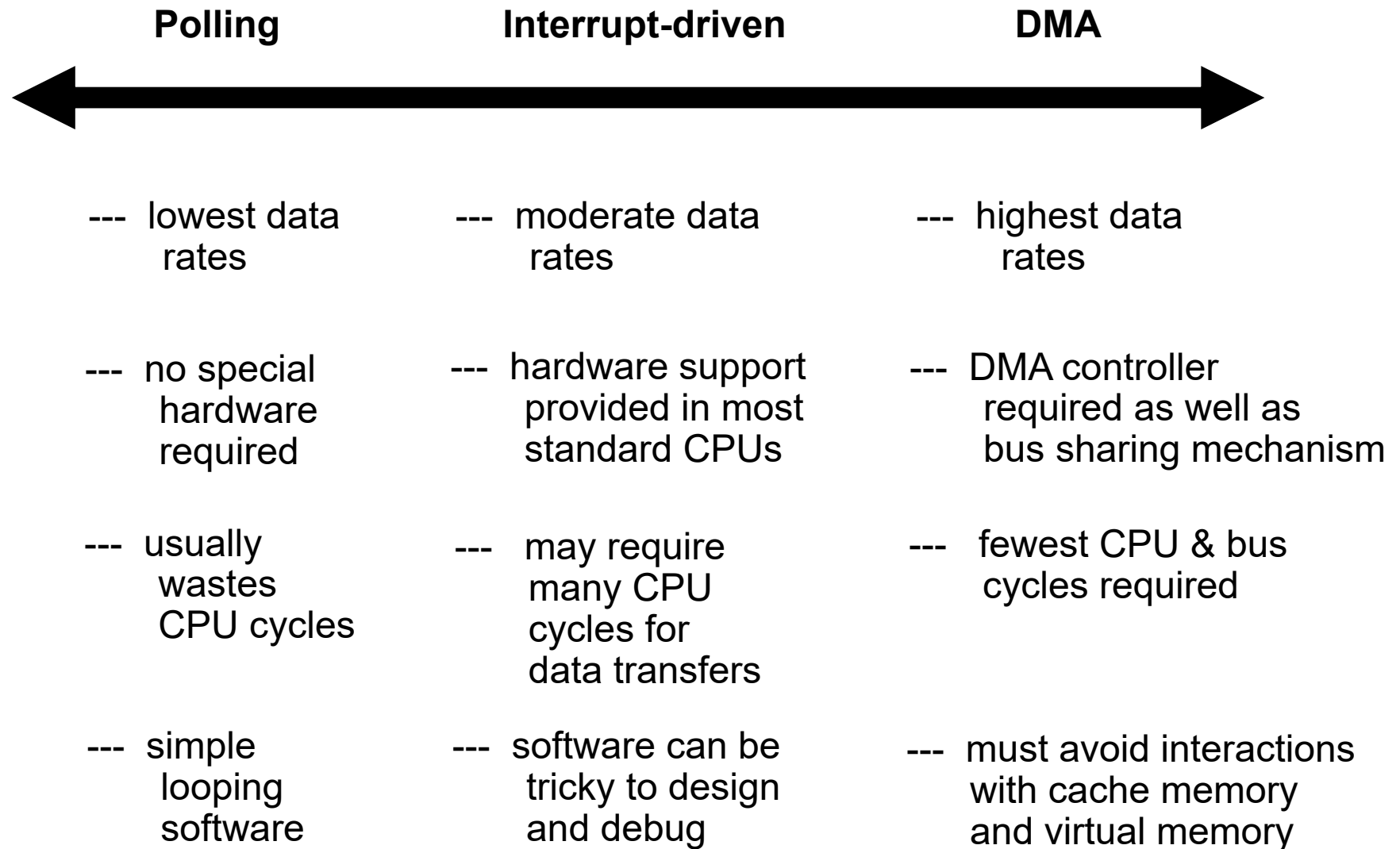
If a CPU takes roughly 100 instructions to process a character, then the CPU has the capacity to process keystrokes 500,000 times faster than they can be entered by a human typist.

Example #2: CRT (Video) Output

data rate = 30,000 KB/s = 15 million 16-bit words / sec

A 150-MHz ColdFire can move blocks of data at a maximum rate of about 75 million longwords per second (150 Mwords/sec).

Spectrum of I/O Data Transfer Methods



Examples of Polling

Determine the "polling overhead" for three different I/O devices. Assume that 100 clock cycles are required to perform one polling operation, and that the CPU executes using a 150-MHz clock. Determine the fraction of CPU time consumed for the following three cases, where it is a requirement that no data be lost.

- 1) The mouse input must be polled at least 30 times per second to smoothly track user movements.
- 2) The floppy disc transfers data to the CPU in 16-bit words and the data rate is 500 KB/s.
- 3) The hard disc transfers data to the CPU in 512-byte bursts at an average data rate of 20 MB/s. Assume that two clock cycles are required to transfer one longword.

Example of Polling (cont'd)

1) Mouse

Number of clock cycles required to poll the mouse

$$= 30 \times 100$$

$$= 3000 \text{ cycles per second}$$

Thus, fraction of CPU cycles consumed by the mouse

$$= 3000 / (150 \times 10^6)$$

$$= 0.002 \%$$

Conclusion: Impact of mouse polling on CPU performance is insignificant.

Example of Polling (cont'd)

2) Floppy Disc

Number of clock cycles per second used up polling the floppy disc controller

$$\begin{aligned} &= 0.5 \times (500 \times 2^{10}) \times 100 \\ &= 25.6 \times 10^6 \text{ cycles per second} \end{aligned}$$

Thus, fraction of CPU time consumed

$$\begin{aligned} &= \frac{25.6 \times 10^6}{150 \times 10^6} \\ &= 17.1 \% \end{aligned}$$

Conclusion: Impact of polling on CPU performance is significant. Transferring individual words by polling is very inefficient. Consider using a DMA controller.

Example of Polling (cont'd)

3) Hard disc

Number of clock cycles per second used up transferring bytes and polling the hard disc controller

$$\begin{aligned} &= ((20 \times 2^{20}) \times 2 / 4) + ((20 \times 2^{20}) \times 100 / 512) \\ &= 10,485,650 + 4,096,000 \text{ cycles per second} \end{aligned}$$

Thus, fraction of CPU time consumed

$$\begin{aligned} &= \frac{14,581,650}{150 \times 10^6} \\ &= 9.7 \% \end{aligned}$$

Conclusion: Impact of polling on CPU performance is significant. Direct Memory Access (DMA) should be considered to speed up the block transfers.

Interrupt Management

- An interrupt-handling mechanism is a standard feature on all modern microprocessors.
- The interface designer must make several important design decisions with respect to interrupt handling:
 - How many different independent interrupt sources?
 - How should the possible interrupts be prioritized?
 - Will the interrupts will be “autovectored” or “user vectored”? Or are the exception vectors for interrupts stored in an Interrupt Controller Module (INTC)? The MCF54415 has three INTCs that manage interrupts from 173 different possible interrupt sources.
 - How will the interrupt vectors be updated when device drivers are installed or changed?

Direct Memory Access (DMA)

- A suitable method when large quantities of data need to be transferred between an I/O device and main memory
- A *DMA controller* generates bus signals that cause data blocks to be transferred over the bus without the CPU. Multiple CPU "MOVE" instructions are thus avoided.
- Interrupts are used in DMA to signal important events:
 - initial call to device driver to set up the DMA transfer e.g., source and/or destination addresses, byte count
 - IRQ to signal that an error has occurred during DMA
 - IRQ to signal the end of a DMA transfer
- A digital system may contain more than one controller capable of performing DMA. However, using more than one DMA controller at a time will produce declining benefits since they will all be sharing the one system bus.

Direct Memory Access Controllers (DMACs)

- A DMAC is a special-purpose processor for performing Direct Memory Access operations over the shared system bus.
- The data transfer algorithm in a DMAC is hard-coded—it behaves like one *block move instruction* that avoids a large number of fetch-decode-execute cycles for many CPU moves.
- The details of the DMA block move can be configured by the CPU by loading control registers in the DMAC.
- The DMAC can produce interrupts for selected events, such as when the block move completes successfully, or when the block move encounters errors.
- DMACs can exist as separate modules in a microcomputer, or they can be incorporated as part of peripheral modules (e.g., network interfaces, video controllers, memory controllers, etc.)

Potential Complications with DMA

Observation: When a digital system contains a CPU and other controllers capable of performing DMA, then there are two or more processors accessing the memory:

Potential Problems:

1) *CPU with Cache Memory*

- Some words in physical memory may be out of date (the new data is only in the cache).
- If DMA updates memory words, then some lines in the CPU's cache may need to be marked invalid.

2) *Virtual Memory*

- Memory locations have both virtual and physical addresses. Which addresses should DMA use?
- At page boundaries in physical memory, adjacent physical addresses are not necessarily adjacent virtual addresses (i.e., addresses used by CPU S/W).

Options for Solving these Complications

1) *Provide DMA controller with address translation H/W.*

- CPU and DMAC use the same virtual addresses.
- Both CPU and DMAC cause page faults when virtual addresses are used that belong to pages that are not already loaded into physical memory.

2) *Do DMA only within page boundaries in physical RAM.*

- CPU uses virtual addresses with address translation.
- DMAC uses physical addresses without translation.
- Operating system must refrain from moving affected pages during the DMA transfer.

Data Structures Used in I/O Software

- 1) Buffers
- 2) Double Buffers
- 3) FILO Stacks
- 4) FIFO Queues
- 5) Circular Buffers
- 6) Semaphores, flags, etc.

1) Buffers

- It is usually more efficient to process larger blocks of data rather than individual words.
- Why? The overhead per block is often not that different from the overhead per byte or per word.

A *buffer* is the term used to refer to the following:

- 1) a block of data that is processed together
- 2) a region in main memory used to hold such a block of data
- 3) a dedicated memory device, separate from main memory, that is used to store such a block of data

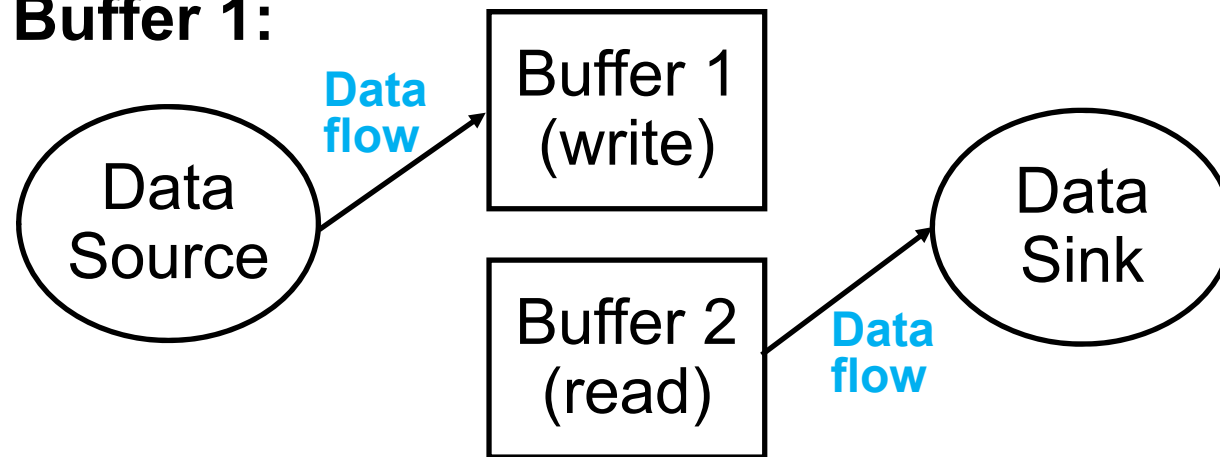
2) Double Buffers

- Two equal-sized storage areas are provided.
- At any one time, one area is the *write buffer*, where newly generated data can be written by one agent.
- At the same time, the other area is the *read buffer*, where previously written data can be read without the danger of having new data written into the area.
- After all of the data in the read buffer has been read, all or some of the contents of the write buffer can be transferred into the read buffer.

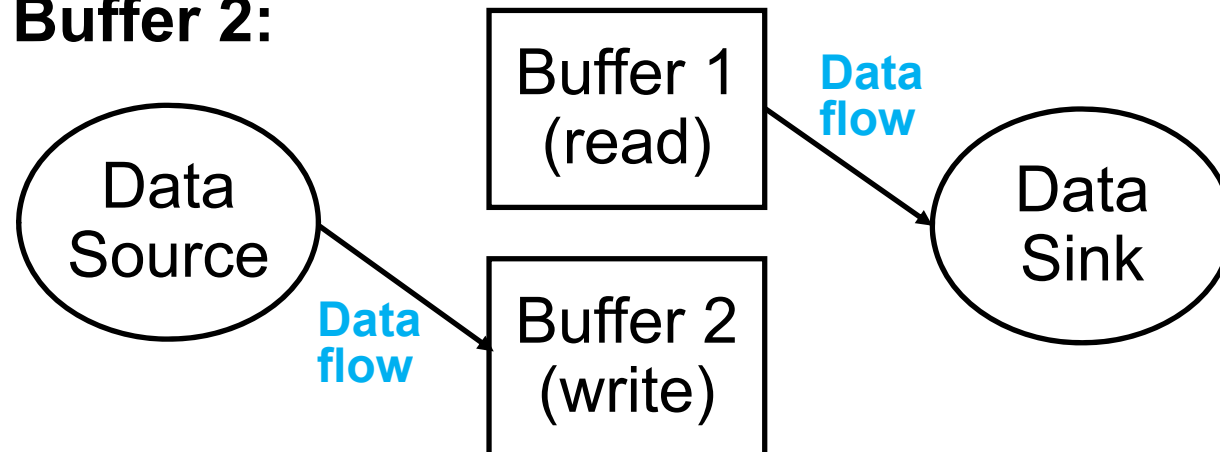
Note: The “transfer” can usually be accomplished by simply swapping the roles of the two buffers, say by swapping the base pointers to the two buffers.

Double Buffers / Ping-Pong Buffers

Writing to Buffer 1:



Writing to Buffer 2:



3) Stacks

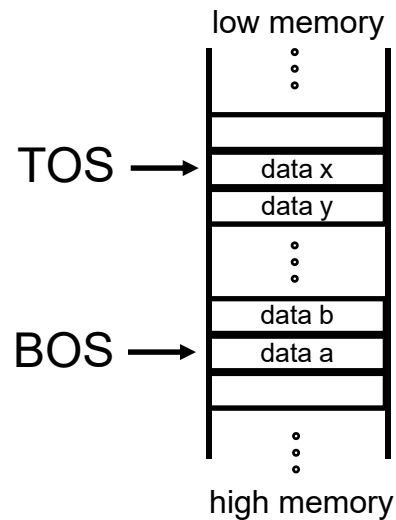
- It is a common requirement to access stored data on a *first-in last-out* (FILO) basis
 - e.g. store the CPU state prior to starting an exception handling routine or a subroutine
- The *FILO stack* is a data structure that provides the following three basic operations:
 - 1) PUSH (datum) -- add a new datum onto the stack
 - 2) TOP -- examine the most recently stored datum in the stack
 - 3) POP -- remove the most recently stored datum from the stack

Note: Sometimes the TOP and POP operations are combined into one operation (POP).

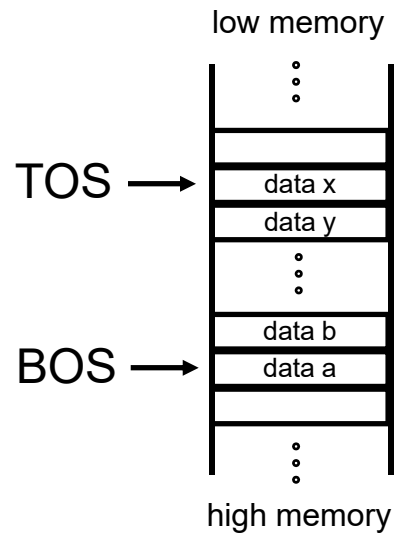
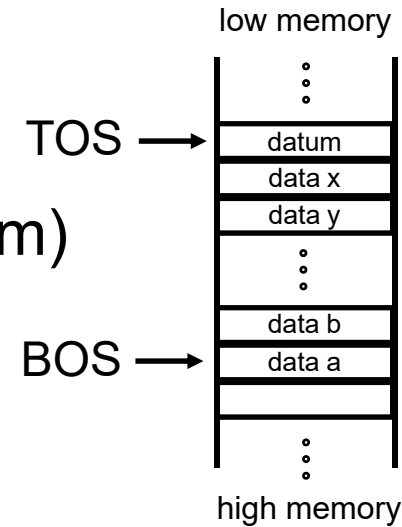
3) Stacks (cont'd)

- Additional stack terminology:
 - **Top of Stack** (TOS) is the storage area occupied by the most recently stored datum.
 - **Bottom of Stack** (BOS) is the storage area occupied by the first datum that was stored.
- A stack may be implemented either as:
 - a) a special-purpose hardware device
 - b) a region in main memory, together with TOS and BOS stack pointers
- Data items are not moved around in memory once they have been pushed onto the stack.
- Instead, a fixed pointer records the BOS and a changing pointer records the TOS.

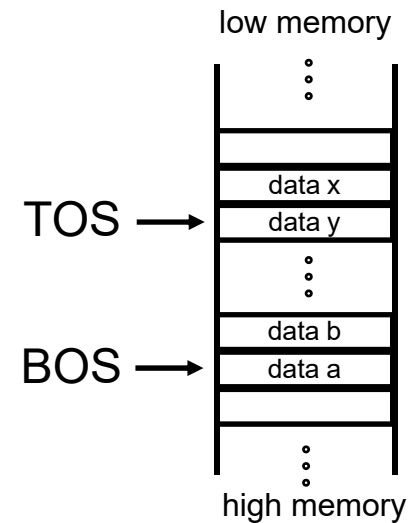
Stacks Growing Toward Low Memory



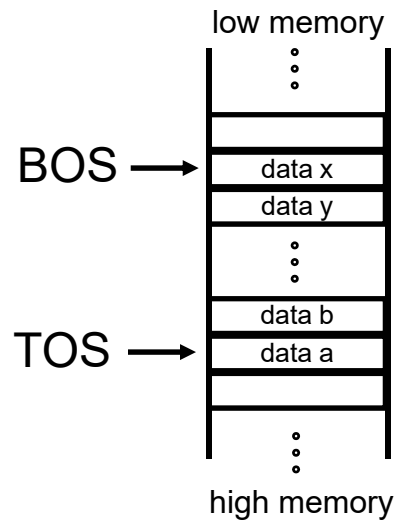
after PUSH(datum)



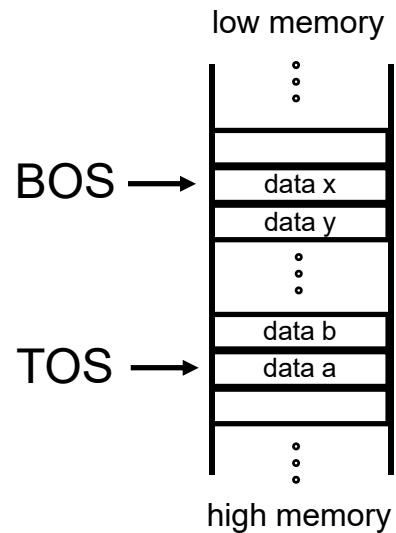
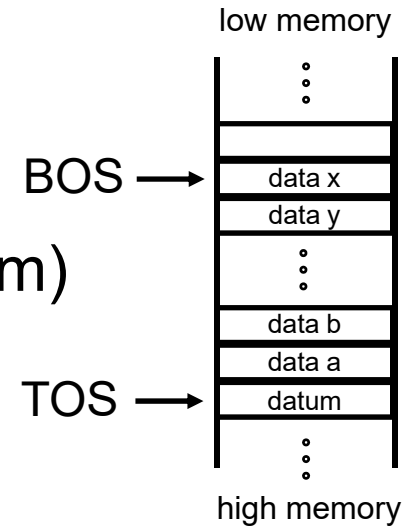
after POP



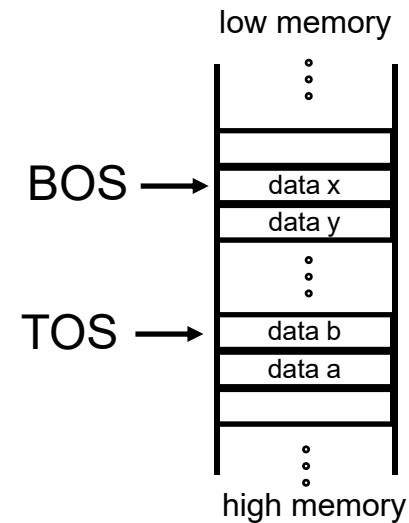
Stacks Growing Toward High Memory



after PUSH(datum)



after POP



4) FIFO Queues

--- A queue is a data structure in which the data items are accessed on a *first-in first-out* (FIFO) basis.

--- A *FIFO queue* provides the following three basic operations:

1) PUSH (datum) -- Add a datum to the back end of the queue.

2) FRONT -- Examine the datum at the front end of the queue.

3) PULL -- Remove the datum from the front end of the queue.

Note: Sometimes the FRONT and PULL operations are combined into one operation (PULL).

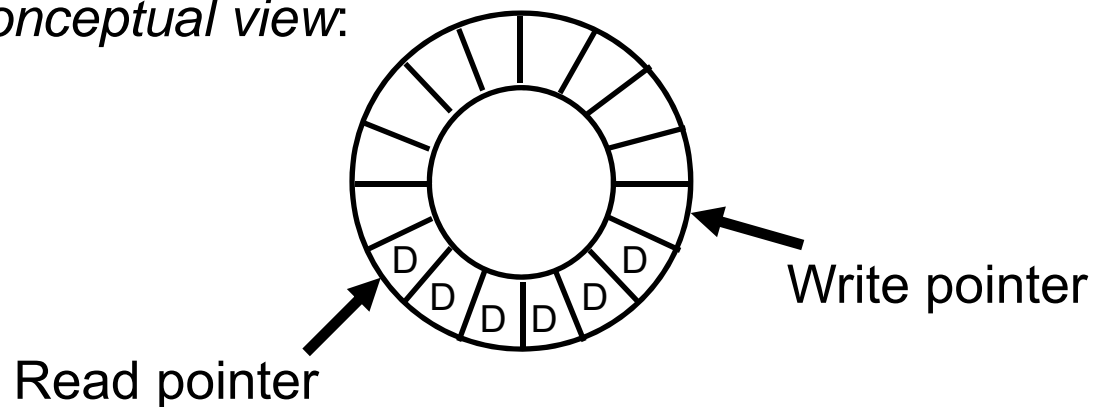
4) FIFO Queues (cont'd)

- Additional queue terminology
 - **Front of Queue** (FOQ) is the storage area occupied by the earliest stored data item that has yet to be removed from the queue.
 - **Back of Queue** (BOQ) is the storage area occupied by the most recently stored data item.
- A queue may be implemented either as:
 - a) a region in main memory, together with FOQ and BOQ stack pointers
 - b) a special-purpose hardware device

5) Circular Buffers

- One practical problem with implementing a queue is to contain the movement of the queue to a limited region in main memory.
- A *circular buffer* is a data structure that provides the same functionality as a queue, within a region in memory defined by two fixed address limits.
- The capacity of the circular buffer is fixed.

Conceptual view:

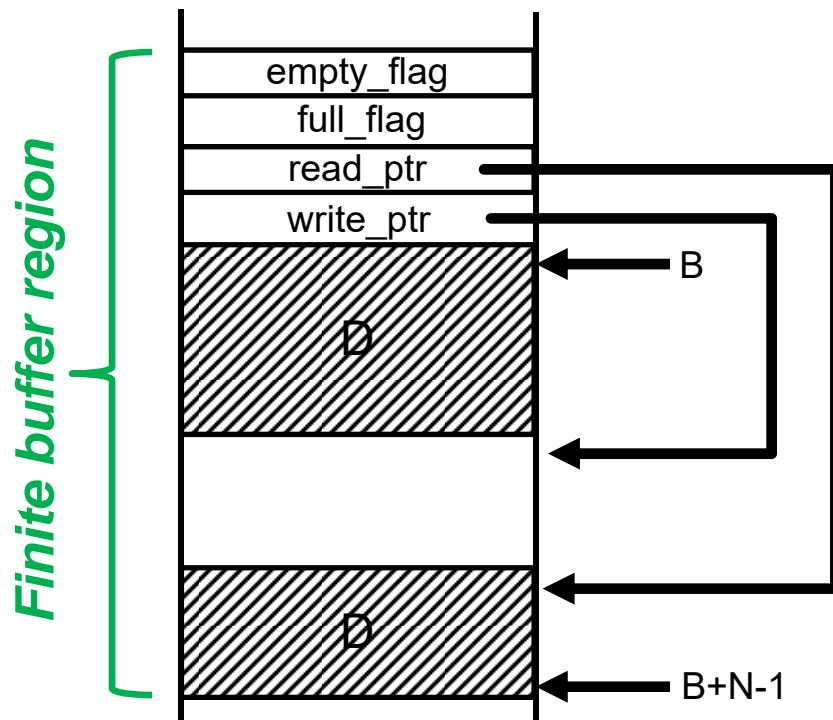


5) Circular Buffers

- Two pointers are used to keep track of data input/output:
 - a) **read pointer** = pointer to the front of the queue
= address where the next datum to be *read* from the buffer is located
 - b) **write pointer** = pointer to the back of the queue
= address where the next datum will be *written* into the buffer
- The read and write pointers advance in the same direction through a finite-sized buffer region in memory.
- When a pointer must be moved beyond the first address limit, it is set to the other address limit, (i.e., wrapped around) instead of being incremented (or decremented) outside of the allocated buffer region in memory.

5) Circular Buffers (cont'd)

Practical Implementation of an N-word Circular Buffer



Initialization routine:

```
/* base address is B */  
init_circular_buffer();  
begin  
    empty_flag = true;  
    full_flag = false;  
    write_ptr = B;  
    read_ptr = B;  
end;
```

5) Circular Buffers (cont'd)

Write routine for the circular buffer.

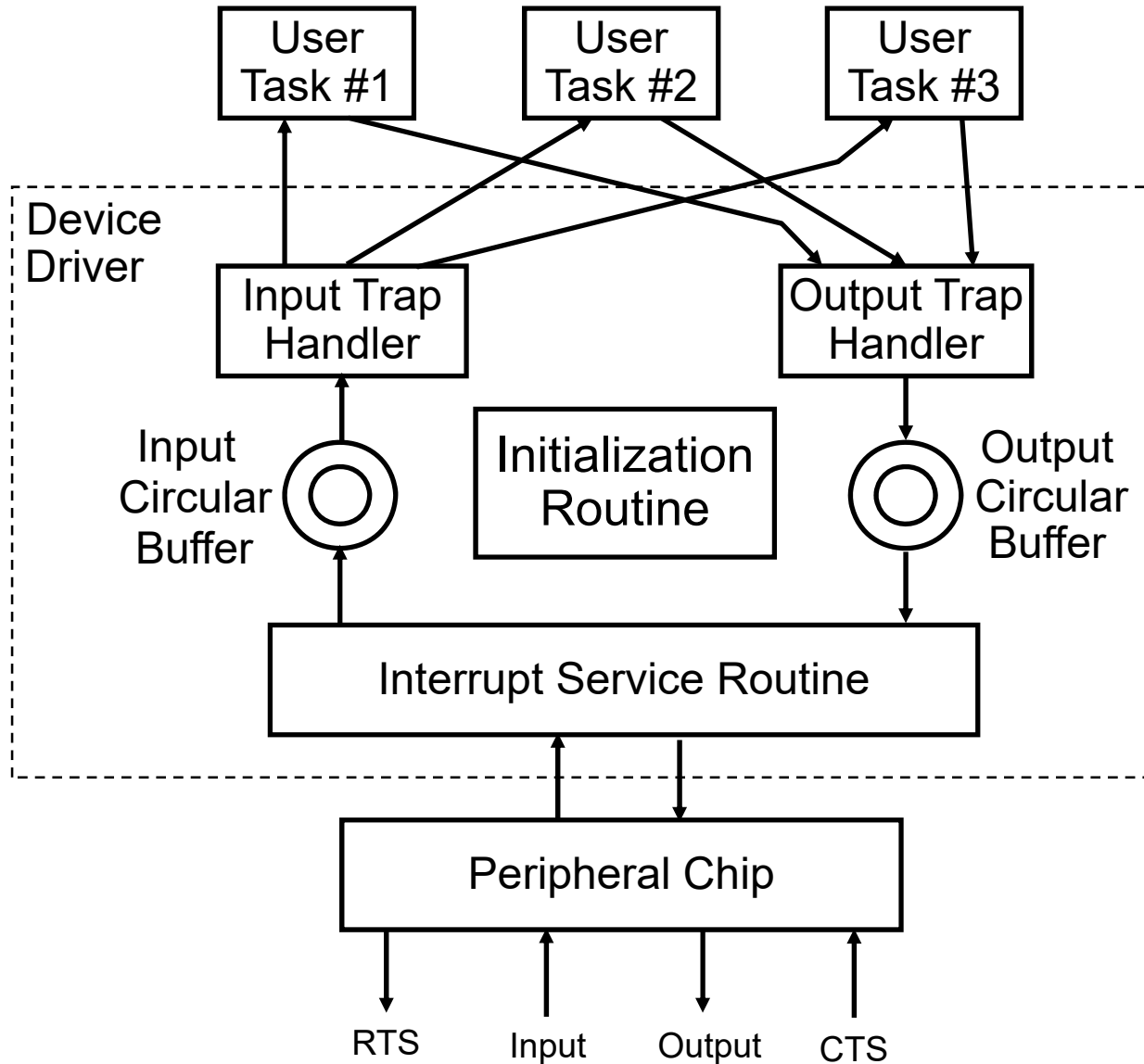
```
err_type procedure write( data_type data );
begin
    if full_flag then return( ERR_BUF_FULL );
    else
        memory(write_ptr) = data;
        empty_flag = false;
        write_ptr = (((write_ptr - B) + 1) mod N) + B;
        if ( read_ptr == write_ptr ) then
            full_flag = true
        endif;
        return( ERR_OK );
    endif;
end;
```

5) Circular Buffers (cont'd)

Read routine for the circular buffer.

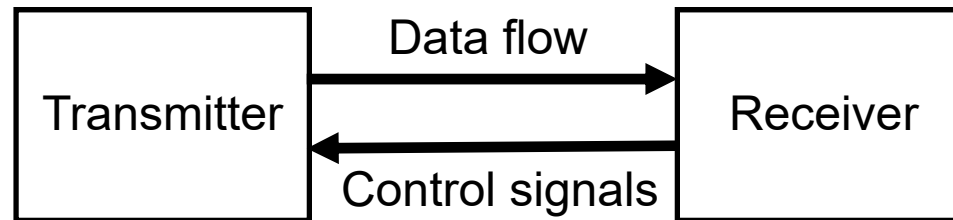
```
err_type procedure read( data_type *Pdata );
begin
    if empty_flag then return( ERR_BUF_EMPTY );
    else
        *Pdata = memory(read_ptr);
        full_flag = false;
        read_ptr = (((read_ptr - B) + 1) mod N) + B;
        if ( read_ptr == write_ptr ) then
            empty_flag = true
        endif;
        return( ERR_OK );
    endif;
end;
```

Circular Buffers in a Device Driver



Note: The device driver must manage the flow of data to and from each user task so that the data flows are not mixed up. For example, a buffer (or file) structure could be used so that the device driver can identify the user task for each buffer and the size of each buffer.

Flow Control



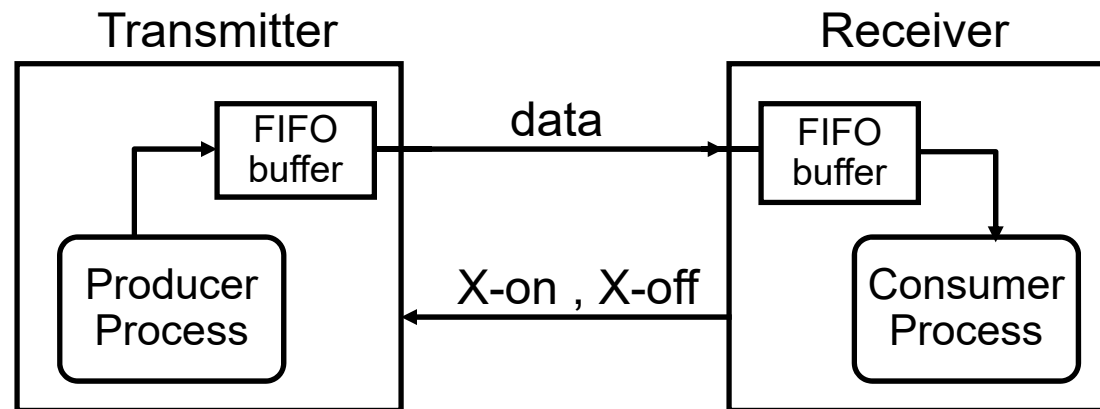
- In an asynchronous communications channel from a transmitter to a receiver, some control mechanism is required to prevent the transmitter from overwhelming the receiver with more data than it can handle.
- A buffer in the receiver helps deal with only short-term fluctuations in data rates, not sustained mismatches.
- Also required is a buffer in the transmitter and some signalling method, called *flow control*, to permit the receiver to stop (or slow down) and restart (or speed up) the flow of data from the transmitter.

Flow Control Mechanisms

Five common flow control mechanisms:

- 1) X-on / X-off for RS-232C
- 2) Ethernet Flow Control using Pause Frames
- 3) Hardware Flow Control
- 4) ENQ / ACK for RS-232C
- 5) Window-based methods, including TCP

1) X-on / X-off Flow Control



- The channel is duplex (bi-directional) and carries ASCII data.
- The transmitter stops sending more data after its receiver gets an ASCII DC3 character, which is also known as X-off (code \$13). On a keyboard, X-off is generated by typing control-S.
- The transmitter can resume sending data to the receiver after its receiver gets an ASCII DC1 character, which is also known as X-on (code \$11). On a keyboard, an X-on is generated by typing control-Q.

Receiver Using X-on/X-off Flow Control

```
/* Receiver polling loop process */
repeat
    if receive_buffer_too_full then
        /* too full could mean 90% full */
        send X-off character back to transmitter;
        repeat
            process some characters in receive buffer;
        until receive_buffer_sufficiently_empty;
        /* sufficiently empty could mean 40% full */
        send X-on character back to transmitter;
    endif;
    if a_character_has_been_received then
        add new character to receive buffer;
    endif;
    if receive_buffer_not_empty then
        process some characters in receive buffer;
    endif;
forever;
```

Transmitter Using X-on/X-off Flow Control

```
/* Transmitter polling loop process */
repeat
  if transmit_buffer_full then
    stop producing more data characters;
  endif;
  if X-off_character_received_from_receiver then
    repeat
      stop transmitting data to receiver;
      if transmit_buffer_full then
        stop producing more data characters;
      else
        produce more data and add to buffer;
      endif;
      do something else, or busy wait, for a while;
    until X-on_received_from_receiver;
  endif;
  if transmit_buffer_not_empty then
    transmit some characters from buffer to receiver;
  endif;
  if transmit_buffer_not_full then
    produce more data characters;
    add data characters to transmit buffer;
  endif;
forever;
```

Pros and Cons of RS-232C X-on/X-off

- Pros:**
- Simple
 - Can easily be implemented in software
 - No need for special hardware features

- Cons:**
- Relatively slow response time if done in software
 - X-on/X-off characters consume data bandwidth
 - How to recover if X-on/X-off chars are lost?

2) Ethernet Flow Control Using Pause Frames

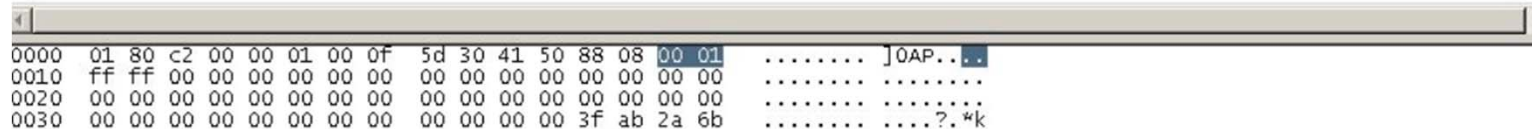
- Ethernet standard IEEE 802.3x (1997) provides a simple flow control method, which allows a receiving node to send a special “pause frame” to a transmitting node that causes that node to stop transmitting frames to the receiving node for a finite time given in the pause frame.
- The transmission pause time is expressed in units of “quanta”, where a quanta is equal to 512 bit times (64 serial byte times).
- When the 16-bit Length/Type field parameter in the Ethernet frame header is 0x8808 (decimal 34,824), the data field in an Ethernet frame is interpreted as a “MAC Control” command.
- A “pause frame” is encoded with the 16-bit opcode 0x0001, followed a 16-bit pause time (in units of quanta), and then 42 all-zero bytes.
- Either the actual destination address (for the transmitter to be stopped) or the reserved multicast address 01:80:C2:00:00:01 must be used in the header of the pause frame. The source address in the pause frame is the address of the node that sends the pause frame.

Structure of an Ethernet Pause Frame

```

Frame 2 (64 bytes on wire, 64 bytes captured)
  Arrival Time: Jan 30, 2008 10:25:52.012139558
  [Time delta from previous captured frame: 0.036914825 seconds]
  [Time delta from previous displayed frame: 0.036914825 seconds]
  [Time since reference or first frame: 0.036914825 seconds]
  Frame Number: 2
  Frame Length: 64 bytes
  Capture Length: 64 bytes
  [Frame is marked: False]
  [Protocols in frame: eth:mac]
  [Coloring Rule Name: Broadcast]
  [Coloring Rule String: eth[0] & 1]
Ethernet II, Src: 42networ_30:41:50 (00:0f:5d:30:41:50), Dst: spanning-tree-(for-bridges)_01 (01:80:c2:00:00:01)
  Destination: spanning-tree-(for-bridges)_01 (01:80:c2:00:00:01)
    Address: spanning-tree-(for-bridges)_01 (01:80:c2:00:00:01)
    .... 1 .... = IG bit: Group address (multicast/broadcast)
    .... 0 .... = LG bit: Globally unique address (factory default)
  Source: 42networ_30:41:50 (00:0f:5d:30:41:50)
    Address: 42networ_30:41:50 (00:0f:5d:30:41:50)
    .... 0 .... = IG bit: Individual address (unicast)
    .... 0 .... = LG bit: Globally unique address (factory default)
  Type: MAC Control (0x8808)
MAC Control
  Pause: 0x0001
  Quanta: 65535

```



```

0000  01 80 c2 00 00 01 00 0f 5d 30 41 50 88 08 00 01  .... ]OAP...
0010  ff ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ....
0030  00 00 00 00 00 00 00 00 00 00 00 00 3f ab 2a 6b  ....?..*k

```

Courtesy of Wikipedia

Destination address = 01:80:c2:00:00:01

MAC Control opcode = 00:01 “Pause frame”

Source address = 00:0f:5d:30:41:50

Pause time = ff:ff 65535 x 512 bit times

Length/type = 88:08 “MAC Control”

Frame checksum = 3f:ab:2a:6b

Pros and Cons of Ethernet Pause Frames

Pros: • Simple mechanism, like X-off with timeout to X-on.

Cons: • The delay to stop transmission is at least the transmission time of the pause frame. This is the time to transmit 84 bytes (12-byte minimum interframe gap, 7-byte preamble, 1-byte SFD, 64 bytes in the rest of the frame).

- The pause frame stops *all* transmission from one node to a second node. This mechanism does not distinguish between different kinds of traffic, and so it can be disruptive to operation.

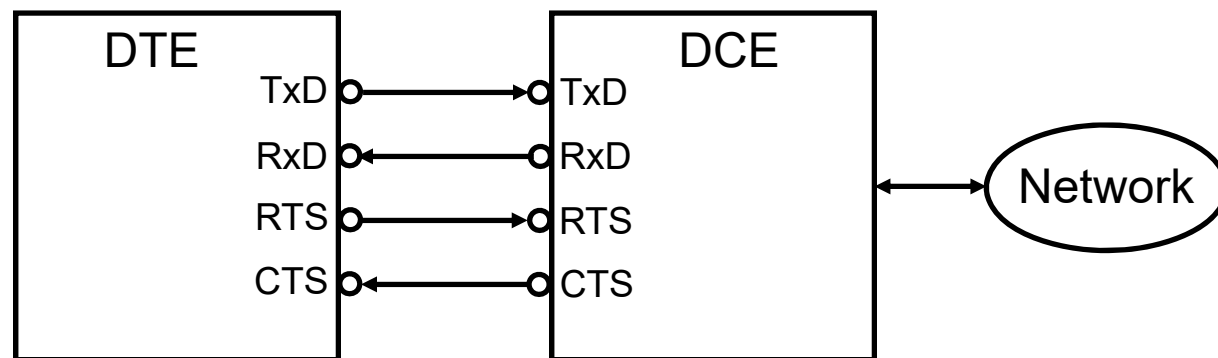
Ethernet standard IEEE 802.1Qbb (2011) defines an enhanced “Priority pause frame” that provides separate pause times for traffic at eight different “classes of service”.

3) Hardware Flow Control

- Instead of using special data characters for flow control signalling, use hardware handshake signals.
- Appropriate handshake signals are provided in the RS232c asynchronous communications standard.

Request to Send (RTS) is used by the DTE to signal to the DCE its desire to transmit more data.

Clear to Send (CTS) is used by the DCE to signal its ability to receive more data from the DTE.



Pros and Cons of Hardware Flow Control

- Pros:**
- Faster than X-on/X-off flow control
 - Simple handshake protocol is easily implemented

- Cons:**
- Extra hardware signals (and wires) are required to carry the control signals (e.g., CTS, RTS) in parallel with the data signals (e.g., TxD, RxD).

4) ENQ / ACK Flow Control

- A block-oriented mechanism: i.e., data is sent from transmitter to receiver in fixed-sized blocks.
- Transmitter sends ASCII “ENQ” control character (hex value \$05) to ask the receiver whether it is ready to receive the next block of data.
- Receiver sends ASCII “ACK” control character (hex value \$06) to transmitter when it is ready to receive the block of data.
- After the transmitter receives the “ACK”, it sends one block of data to the receiver.
- ENQ-ACK cycle begins again for the new block of data to be transmitted.

Pros and Cons of ENQ/ACK Flow Control

Similar pros and cons as in X-on/X-off flow control.

- Pros:**
- Simple
 - Can easily be implemented in software
 - No need for special hardware features
- Cons:**
- Relatively slow response time if done in software
 - ENQ/ACK characters consume data bandwidth
 - How to recover if ENQ/ACK chars are lost?
 - The data flow is limited by the control communication delay (sending ENQ & then receiving ACK).

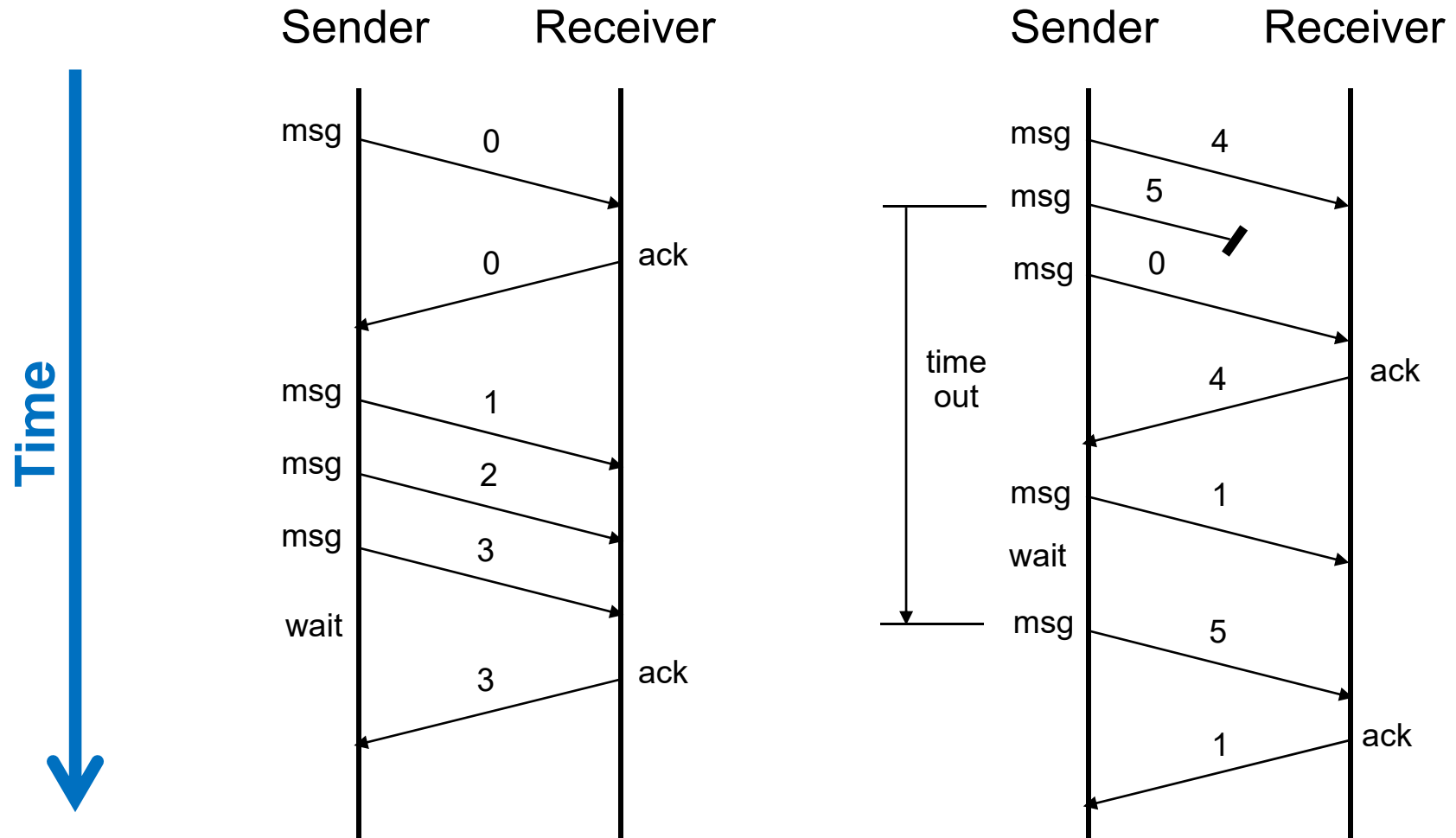
5) Window-based Flow Control

- A method of flow control together with a mechanism for retransmitting lost messages, and detecting out-of-order and duplicate received messages.
- The sender tags its message with a sequence number $0, \dots, N-1$ that is incremented after each message.
- The receiver acknowledges single messages or groups of messages by sending back the sequence number of the last message/byte that was correctly received in order.
- The sender is allowed to have up to $W \leq N$ messages sent to the receiver but not yet acknowledged.
 W is the *window size*, which may be fixed or adjustable.

Handling Lost Messages / Data Packets

- It is possible for data packets to get lost
 - they may get discarded because of detected errors
 - they may get discarded because of buffer overflow
 - they may get greatly delayed because of congestion
- Once the transmitter has sent out W data packets, it must stop and wait for an acknowledgement. But it should not wait forever.
- A timer is used to keep track of the time elapsed since each data packet was transmitted. If the elapsed time exceeds some value (e.g., twice the expected round trip delay), then the transmitter assumes that the data packet was lost, and then retransmits all unacknowledged packets.

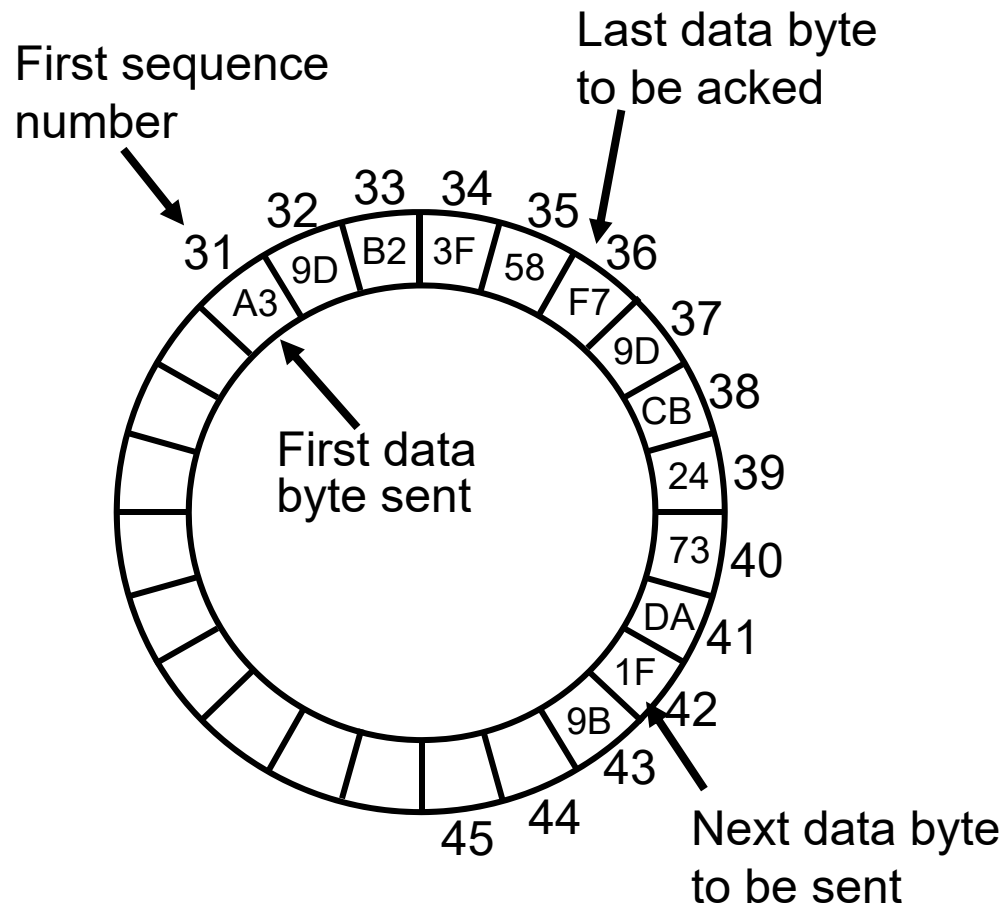
Time Sequence Diagrams



Ex: Flow Control in TCP

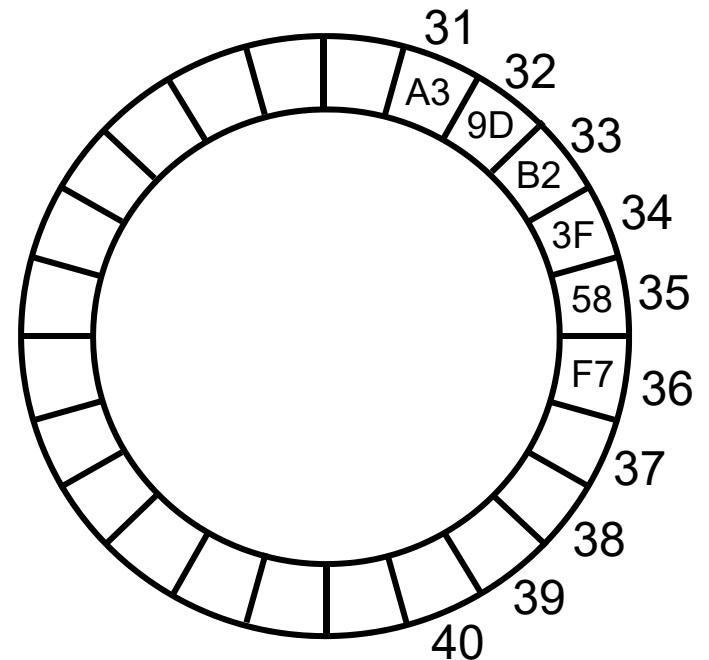
- The *data bytes* are numbered by TCP to allow it to:
 - (1) detect the loss of segments (loss of data bytes)
 - (2) detect out-of-order and duplicate segments
 - (3) implement flow control
- Data bytes (not the data packets) in the transmit and receive directions are numbered separately.
- The first data byte in either direction is assigned a random initial 32-bit unsigned integer number.
- Each TCP segment is assigned a 32-bit sequence number, which is the number of first data byte in the segment.
- The 32-bit acknowledgment number in the transmit direction is the byte number of the next expected data byte in the receive direction.

Ex: Flow Control in TCP



Buffer in Sender

Bytes in transit: 9D, CB, 24, 73, DA



Buffer in Receiver

Ex: Flow Control in TCP

- The receiver (with respect to one of the two directions) controls the window size in the sender.
- The sender window size is communicated in a field in the segments that are travelling in the opposite direction.
- The sender window could be set equal to the current number of free byte locations in the receiver's buffer.
- A sender window value of zero prevents the transmitter from sending any more data (like an X-off command).
- A sender window value of greater than zero allows the transmitter to send data bytes (but not exceeding the window size if there are unacknowledged data bytes).

Pros and Cons of Window-based Flow Control

- Pros:**
- Flexible
 - Accommodates flow control, lost segments, out-of-order segments, duplicate segments
- Cons:**
- More complicated than other flow control methods
 - Need timers to detect lost segments (without timeouts, the TCP connection could “hang” and be prevented from transmitting any more data).