Data Communications & Networks

Session 3 – Main Theme
Data Encoding and Transmission

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Agenda

1 Session Overview

2 Data Encoding and Transmission

3 Summary and Conclusion

- Course description and syllabus:
  - » http://www.nyu.edu/classes/jcf/csci-ga.2262-001/
  - » http://cs.nyu.edu/courses/Fall13/CSCI-GA.2262-001/index.html

- Textbooks:
  - » **_Computer Networking: A Top-Down Approach (6th Edition)_**

    James F. Kurose, Keith W. Ross
    Addison Wesley
    ISBN-10: 0132856204, ISBN-13: 978-0132856201, 6th Edition (02/24/12)

- Computer Networks and the Internet

- Application Layer

- Fundamental Data Structures: queues, ring buffers, finite state machines

- Data Encoding and Transmission

- Local Area Networks and Data Link Control

- Wireless Communications

- Packet Switching

- OSI and Internet Protocol Architecture

- Congestion Control and Flow Control Methods

- Internet Protocols (IP, ARP, UDP, TCP)

- Network (packet) Routing Algorithms (OSPF, Distance Vector)

- IP Multicast

- Sockets

- Introduction to Basic Networking Concepts (Network Stack)
- Origins of Naming, Addressing, and Routing (TCP, IP, DNS)
- Physical Communication Layer
- MAC Layer (Ethernet, Bridging)
- Routing Protocols (Link State, Distance Vector)
- Internet Routing (BGP, OSPF, Programmable Routers)
- TCP Basics (Reliable/Unreliable)
- Congestion Control
- QoS, Fair Queuing, and Queuing Theory
- Network Services – Multicast and Unicast
- Extensions to Internet Architecture (NATs, IPv6, Proxies)
- Network Hardware and Software (How to Build Networks, Routers)
- Overlay Networks and Services (How to Implement Network Services)
- Network Firewalls, Network Security, and Enterprise Networks

- Data Transmission and Encoding Concepts

- ADTs and Protocol Design

- Summary and Conclusion

Information

Common Realization

Knowledge/Competency Pattern

Governance
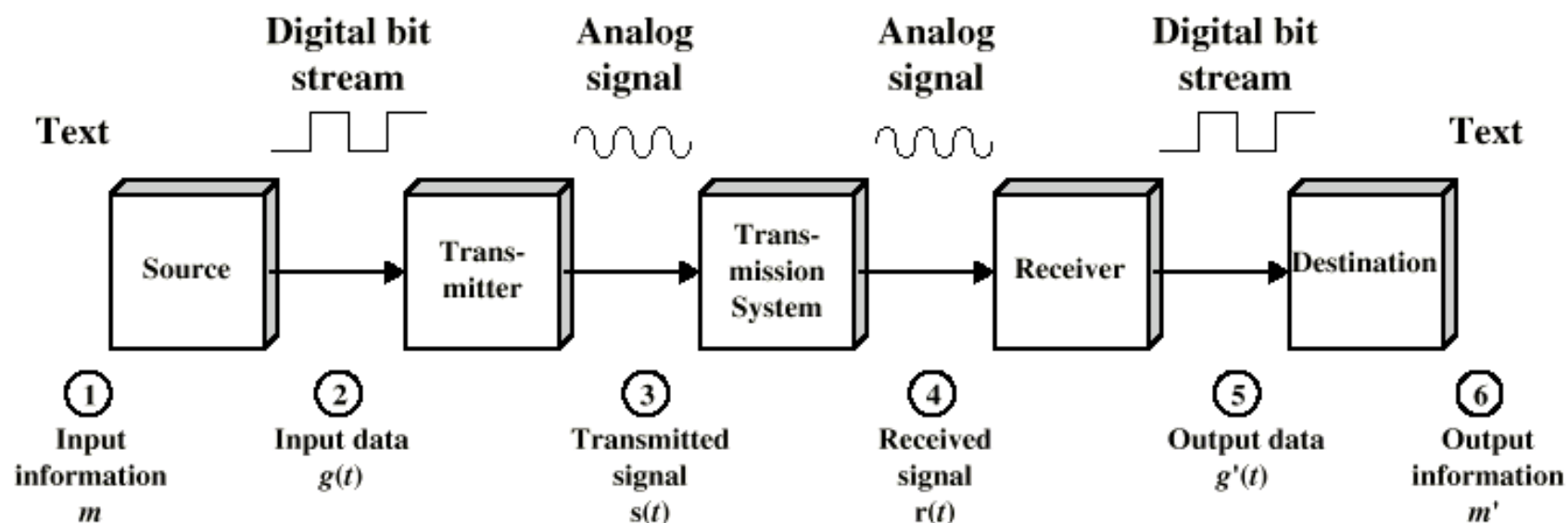
Alignment

Solution Approach

Agenda

1 **Session Overview**

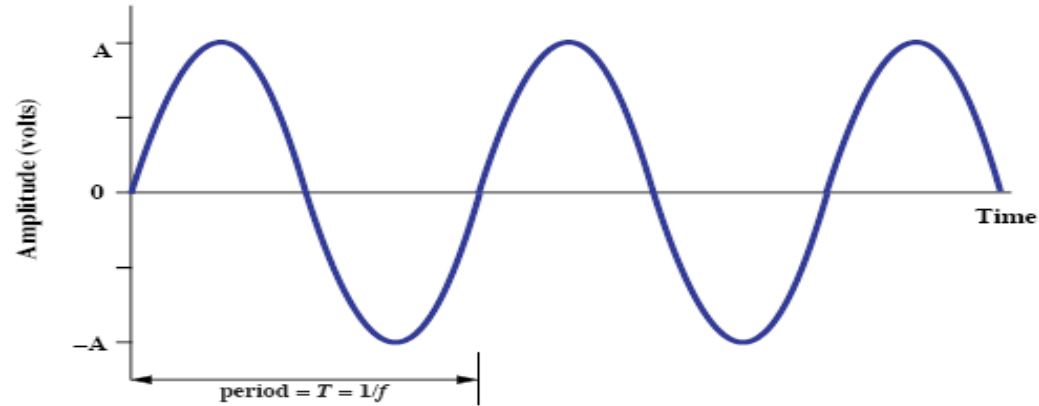2 **Data Encoding and Transmission**

3 **Summary and Conclusion**

Data Encoding and Transmission - Roadmap

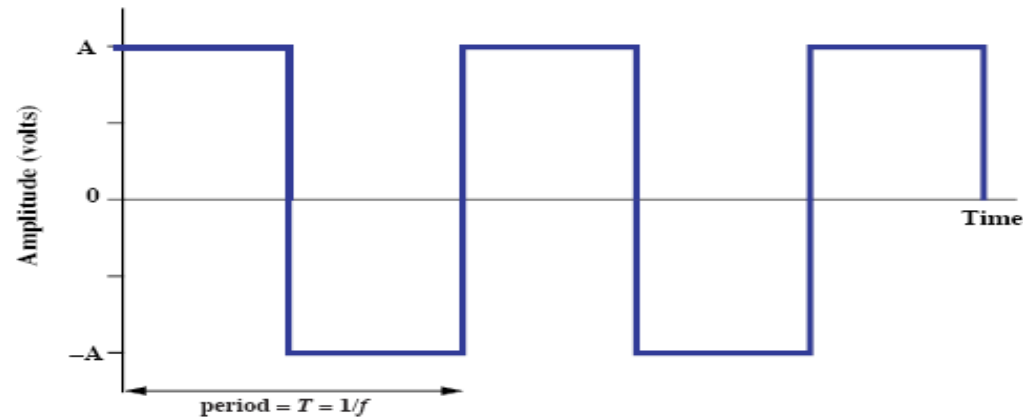**2**   **Data Encoding and Transmission**

Data Encoding and Transmission Concepts

ADTs and Protocol Design

(a) Sine wave

(b) Square wave

11

- Transmitter

- Receiver

- Medium
  - Guided medium
    - E.g., twisted pair, optical fiber
  - Unguided medium
    - E.g., air, water, vacuum

- Direct link
  - No intermediate devices
- Point-to-point
  - Direct link
  - Only 2 devices share link
- Multi-point
  - More than two devices share the link

- ## Simplex

  - ### One direction

    - #### e.g., television

- ## Half duplex

  - ### Either direction, but only one way at a time

    - #### e.g. police radio

- ## Flux duplex

  - ### Both directions at the same time

    - #### e.g., telephone

- Data
  - Entities that convey meaning

- Signals
  - Electric or electromagnetic representations of data

- Transmission
  - Communication of data by propagation and processing of signals

- Analog
  - Continuous values within some interval
    - e.g., sound, video

- Digital
  - Discrete values
    - e.g., text, integers

- Means by which data are propagated

- Analog

  - Continuously variable

  - Various media

    - e.g., wire, fiber optic, space

  - Speech bandwidth 100Hz to 7kHz

  - Telephone bandwidth 300Hz to 3400Hz

  - Video bandwidth 4MHz

- Digital

  - Use two DC components

- Usually use digital signals for digital data and analog signals for analog data

- Can use analog signal to carry digital data
  - Modem

- Can use digital signal to carry analog data
  - Compact Disc audio

- Analog signal transmitted without regard to content

- May be analog or digital data

- Attenuated over distance

- Use amplifiers to boost signal

- Also amplifies noise

Digital Transmission

- Concerned with content

- Integrity endangered by noise, attenuation etc.

- Repeaters used

- Repeater receives signal

- Extracts bit pattern

- Retransmits

- Attenuation is overcome

- Noise is not amplified

- Cheaper

- Less susceptible to noise

- Greater attenuation

  - Pulses become rounded and smaller

  - Leads to loss of information

Voltage at transmitting end
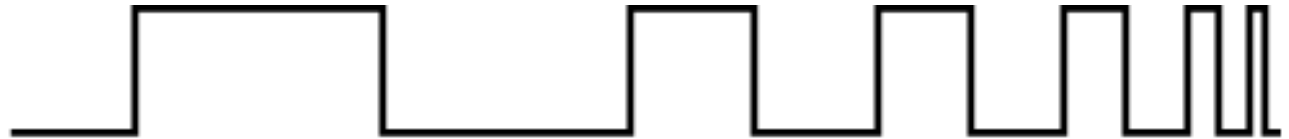
Voltage at receiving end

- Need to know
  - Timing of bits - when they start and end
  - Signal levels
- Factors affecting successful interpreting of signals
 - Signal to noise ratio
  - Data rate
  - Bandwidth

- Non-return to Zero-Level (NRZ-L)

- Non-return to Zero Inverted (NRZI)

- Bipolar –AMI

- Pseudoternary

- Manchester

- Differential Manchester

- B8ZS

- HDB3

- Two different voltages for 0 and 1 bits

- Voltage constant during bit interval

  - No transition (i.e. no return to zero voltage)

  - e.g., Absence of voltage for zero, constant positive voltage for one

- More often, negative voltage for one value and positive for the other

- This is NRZ-L

- Nonreturn to zero inverted on ones

- Constant voltage pulse for duration of bit

- Data encoded as presence or absence of signal transition at beginning of bit time

- Transition (low to high or high to low) denotes a binary 1

- No transition denotes binary 0

- An example of differential encoding
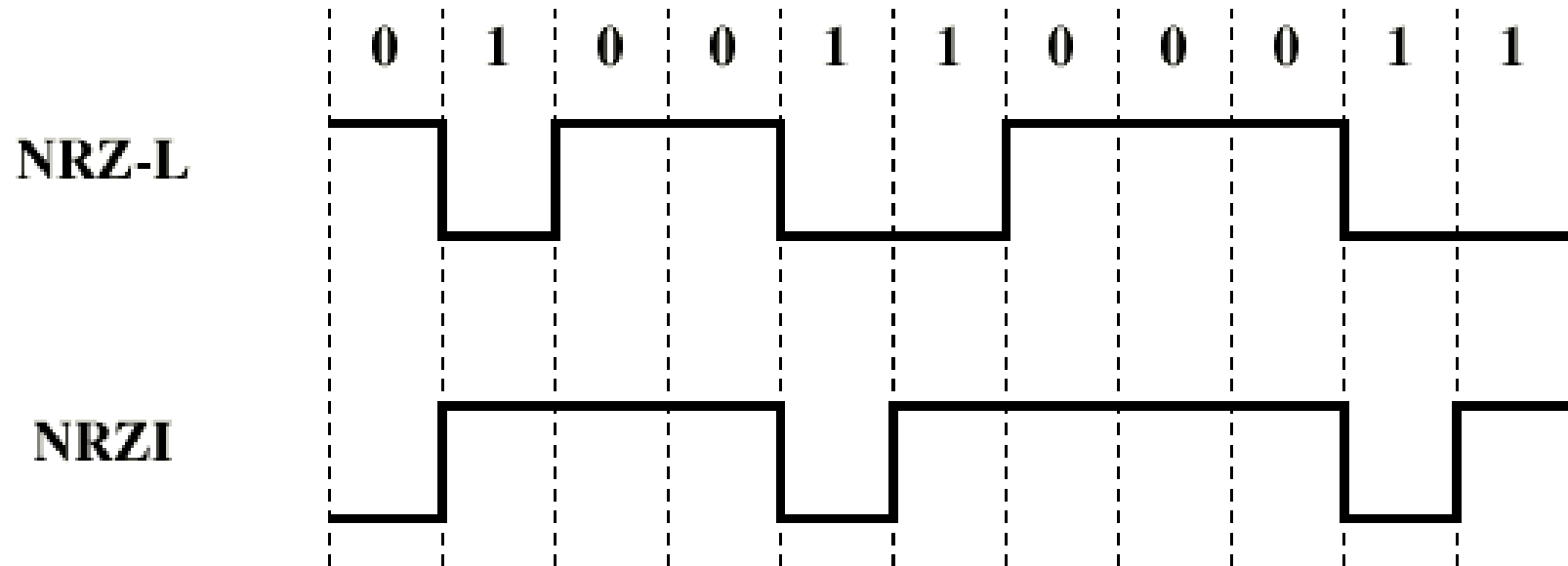
- Data represented by changes rather than levels

- More reliable detection of transition rather than level

- In complex transmission layouts it is easy to lose sense of polarity
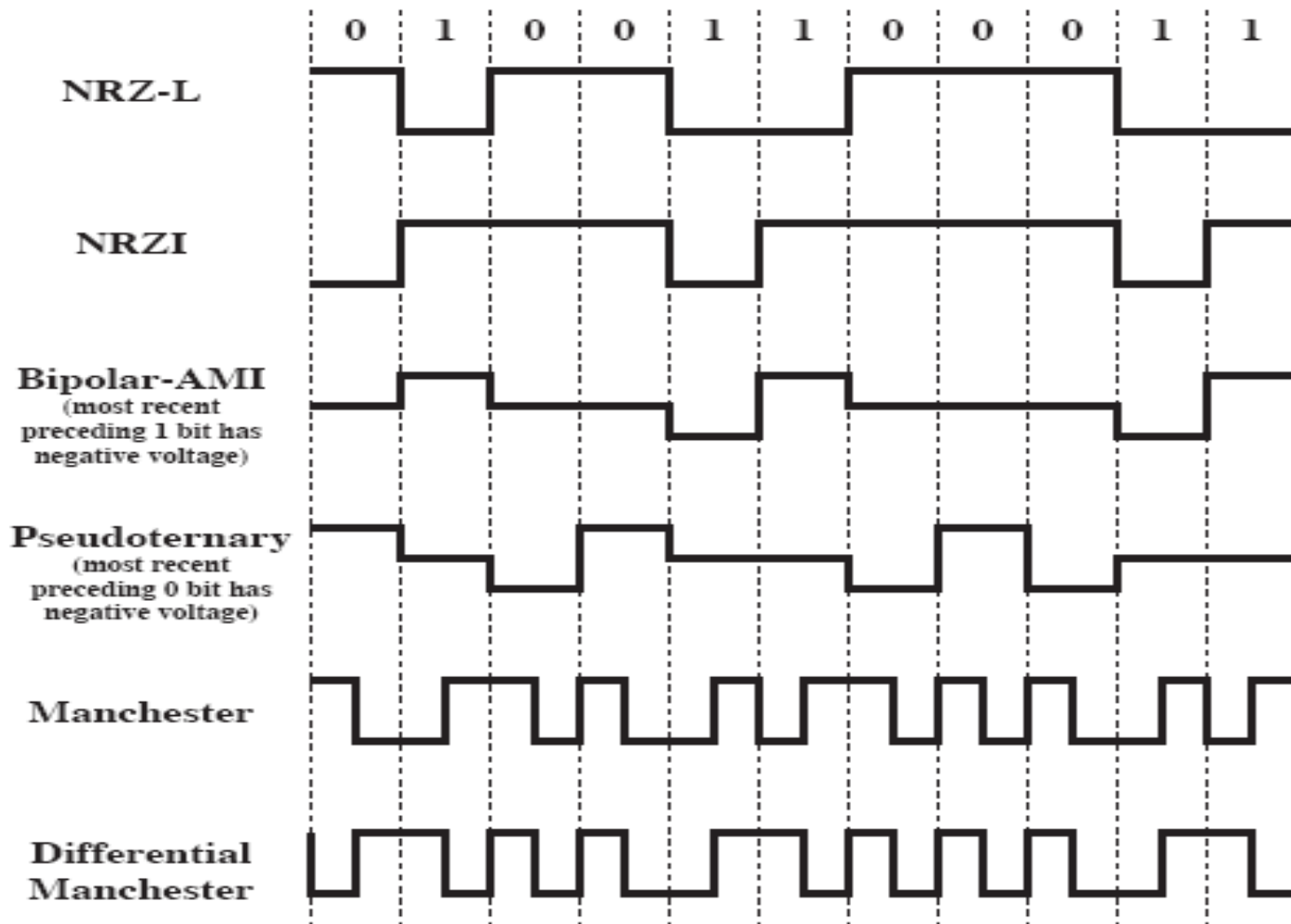
- Pros
  - Easy to engineer
  - Make good use of bandwidth
- Cons
  - DC component
  - Lack of synchronization capability
- Used for magnetic recording
- Not often used for signal transmission

- ## Manchester

    - Transition in middle of each bit period

    - Transition serves as clock and data

    - Low to high represents one

    - High to low represents zero

    - Used by IEEE 802.3

- ## Differential Manchester

    - Mid-bit transition is clocking only

    - Transition at start of a bit period represents zero

    - No transition at start of a bit period represents one

    - Note: this is a differential encoding scheme

    - Used by IEEE 802.5

- Con

  - At least one transition per bit time and possibly two

  - Maximum modulation rate is twice NRZ

  - Requires more bandwidth

- Pros

  - Synchronization on mid bit transition (self clocking)

  - No dc component

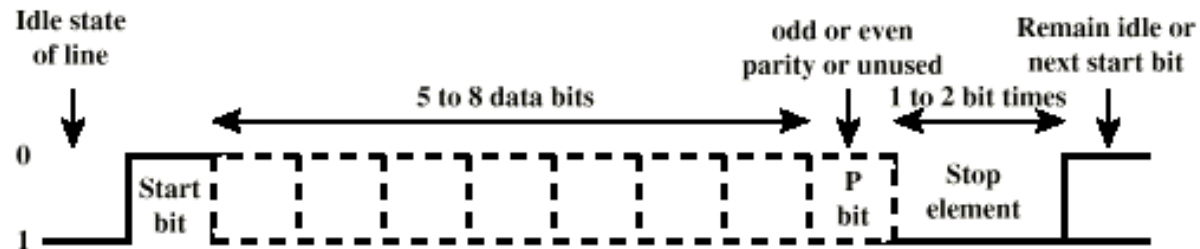 - Error detection

  - Absence of expected transition

- Timing problems require a mechanism to synchronize the transmitter and receiver

- Two solutions
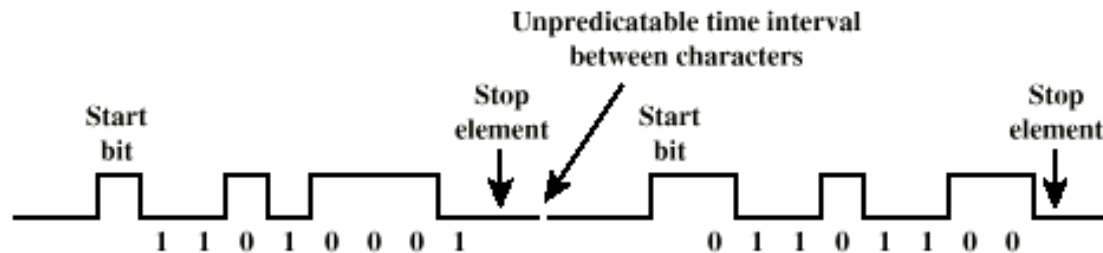
  - Asynchronous

  - Synchronous

- Data transmitted on character at a time

  - 5 to 8 bits

- Timing only needs maintaining within each character
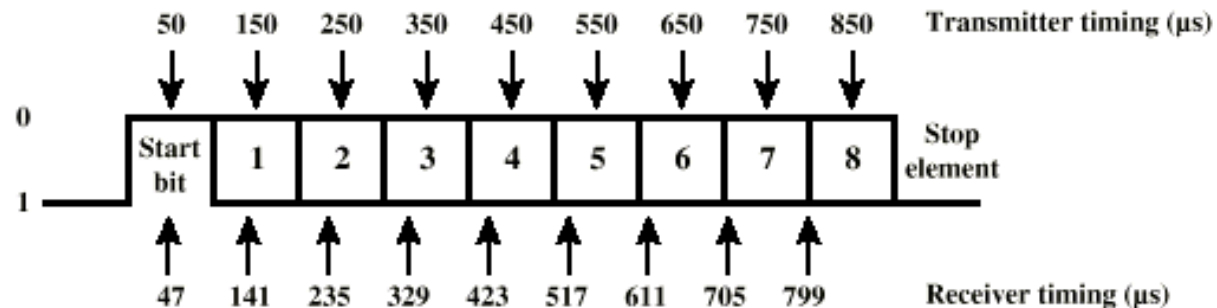
- Resync with each character

(a) Character format

(b) 8-bit asynchronous character stream

(c) Effect of timing error

Asynchronous - Behavior

- In a steady stream, interval between characters is uniform (length of stop element)

- In idle state, receiver looks for transition 1 to 0

- Then samples next seven intervals (char length)

- Then looks for next 1 to 0 for next char

- Simple

- Cheap

- Overhead of 2 or 3 bits per char (~20%)

- Good for data with large gaps (keyboard)

- Block of data transmitted without start or stop bits

- Clocks must be synchronized

- Can use separate clock line

  - Good over short distances

  - Subject to impairments

- Embed clock signal in data

  - Manchester encoding

  - Carrier frequency (analog)

- Need to indicate start and end of block

- Use preamble and postamble

  - e.g. series of SYN (hex 16) characters

  - e.g. block of 11111111 patterns ending in 11111110

- More efficient (lower overhead) than async
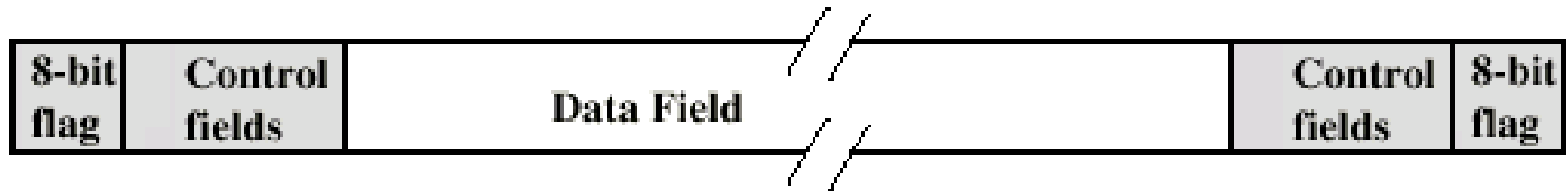
8-bit flag	Control fields	Data Field	Control fields	8-bit flag

Data Encoding and Transmission - Roadmap

2   **Data Encoding and Transmission**

Data Encoding and Transmission Concepts

ADTs and Protocol Design

- When building protocol software, there are two common problems that designers face:

  1) How to handle data that arrives from two independent sources

    - Down from the higher layer
   - Up from the lower layer

  2) How to implement the protocol

- **Down from the Higher Layer (HL)**

    - Higher layer (HL) sends requests (control and data)

    - Cannot always process the request immediately, so we need a place to hold the request

    - We may get "many" HL users (e.g., many TCP, only one IP)

    - Requests may need to be processed out of order (out of band, QOS, etc)

- Up from the Lower Layer (LL)

  - Lower layer sends data and indications

  - Data must be separated from indications

  - Read requests from HL may use different data boundaries than LL

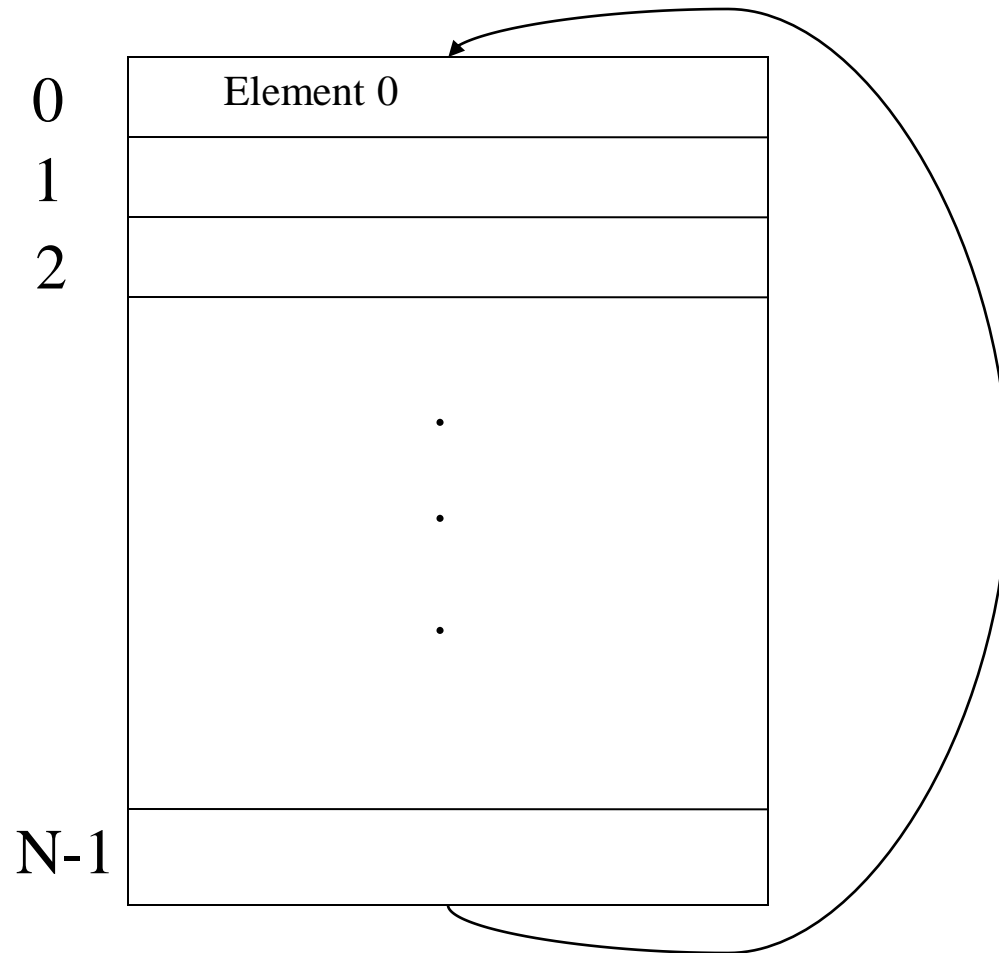 - LL may be providing data at same time as HL wants to read it

0
1
2

.
.
.

N-1
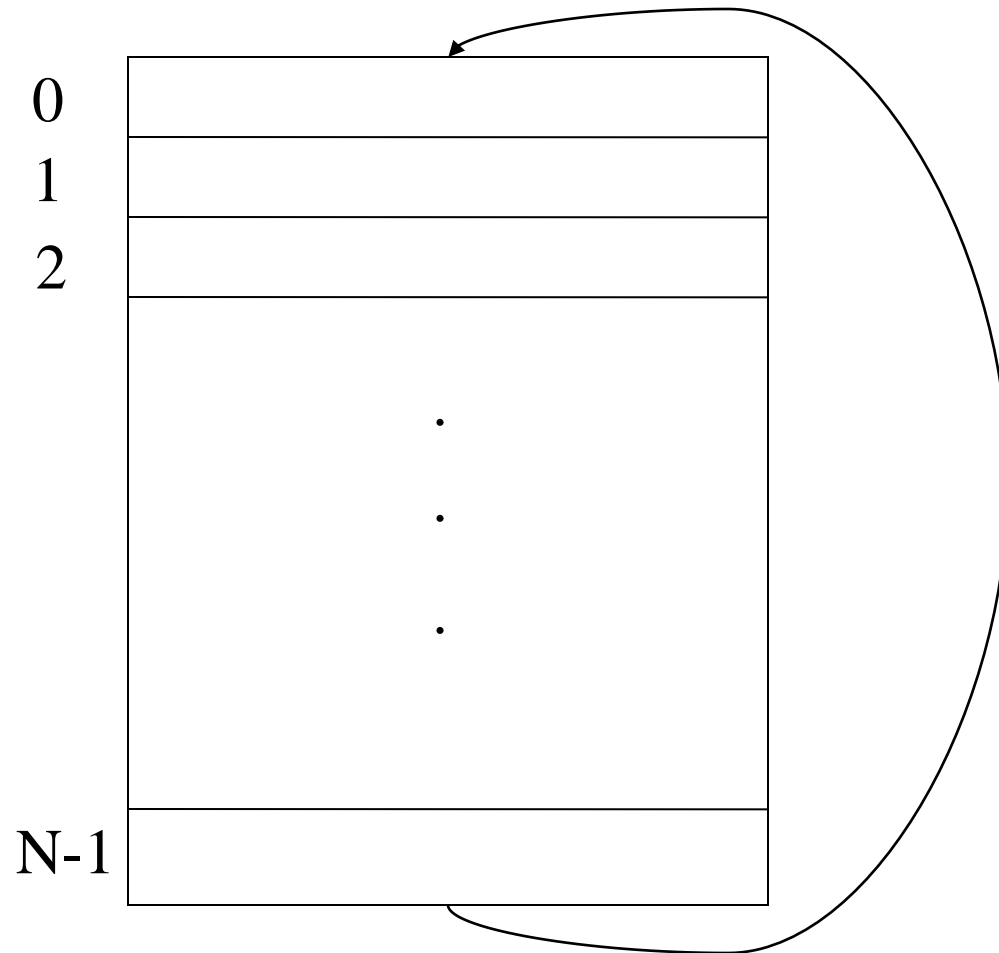
Inititial State

Input: 0

Output: 0

0
1
2

N-1

Element 0

.
.
.

New Element Arrives

Input: 1

Output: 0

0 | Element 0
1 | Element 1
2 |

.

.

.

N-1 |
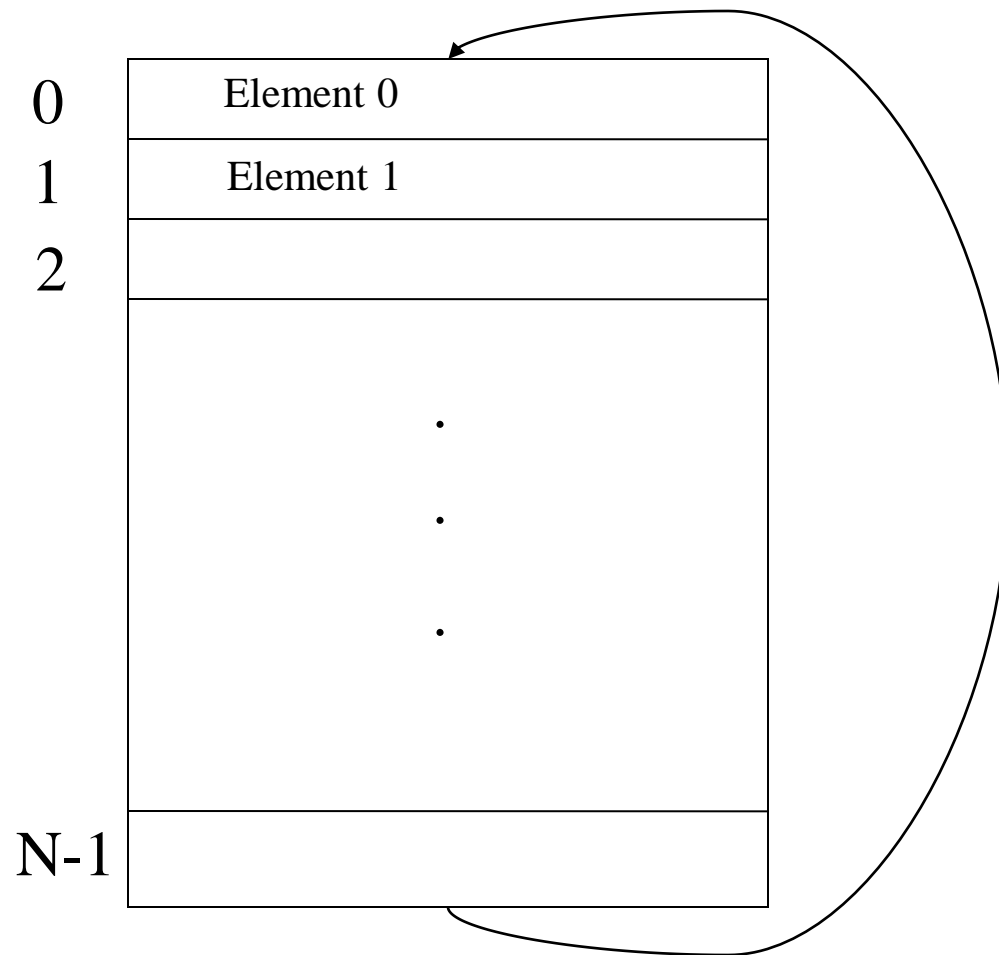
**New Element
Arrives**

Input: 2

Output: 0

0	Element 0
1	Element 1
2	
	.
	.
	.
N-1	
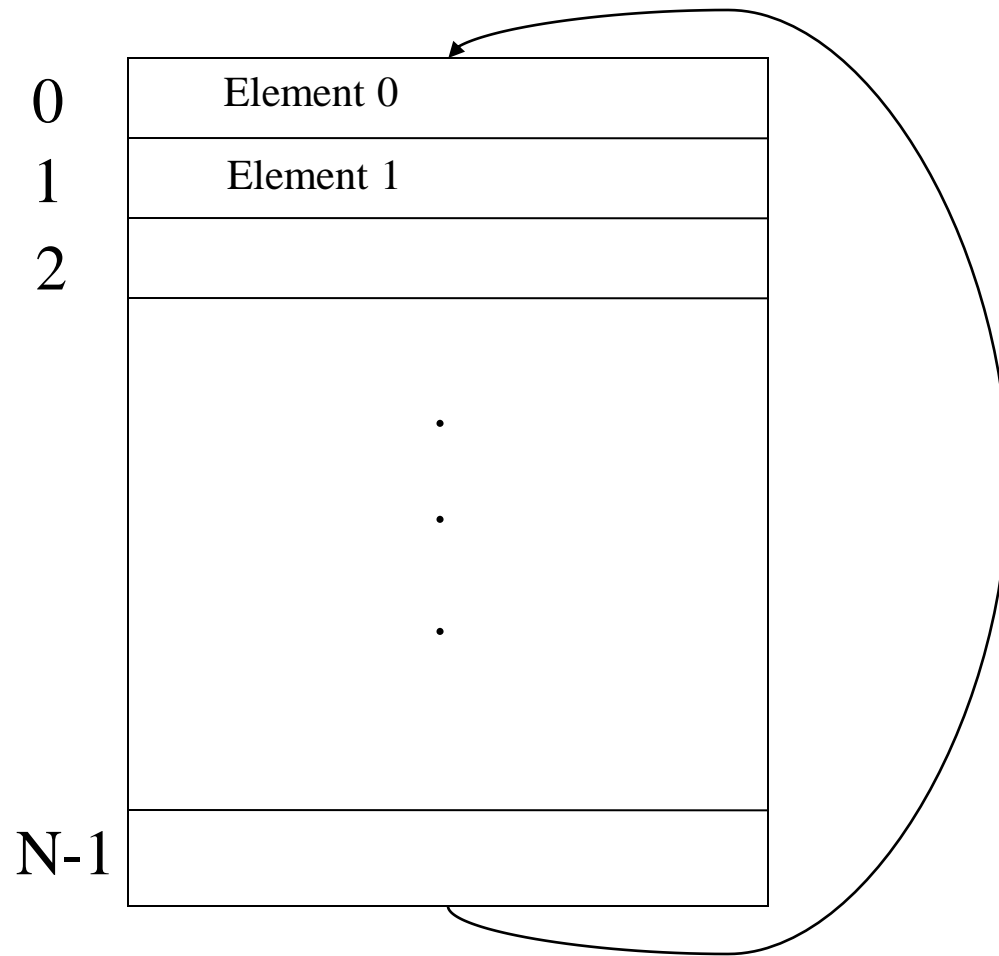
Read next (element 0)

Input: 2

Output: 1

Read next (element 1)

Input: 2

Output: 2

0 | Element 0
1 | Element 1
2 | Element 2

.

.

.

N-1 | Element N-1
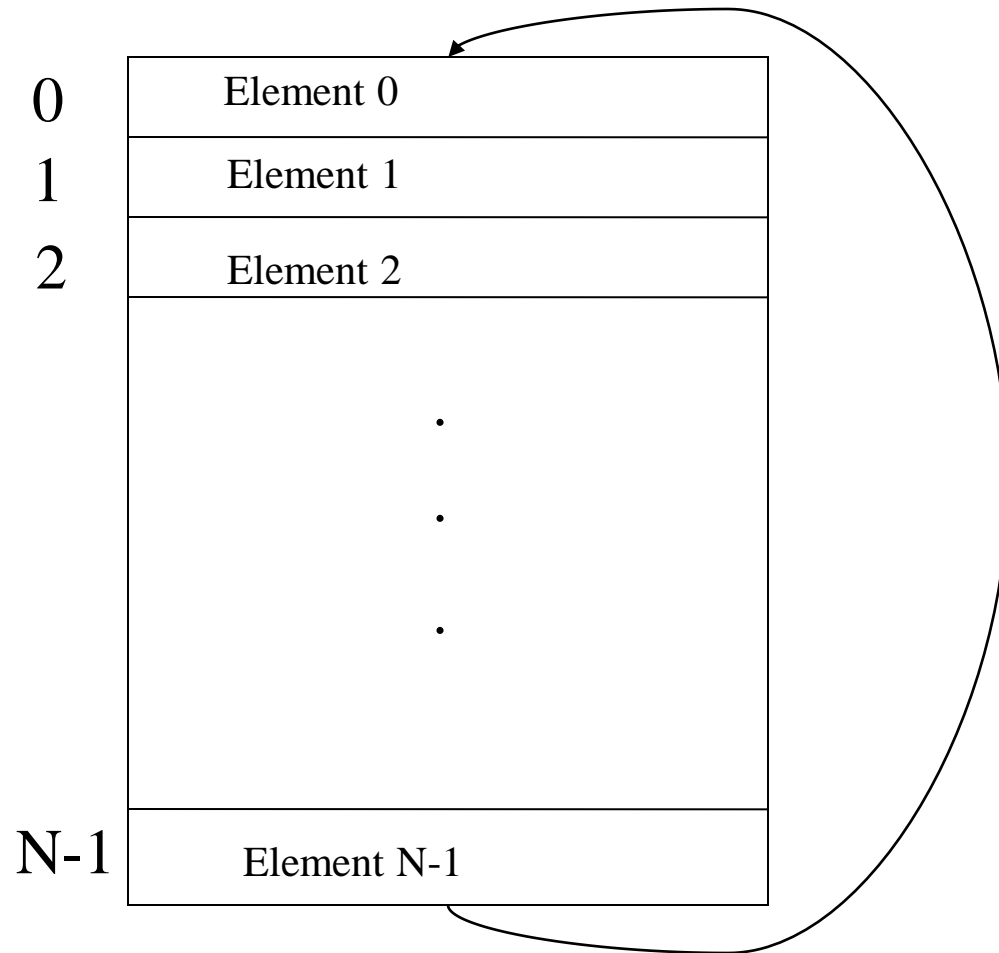
After Nth input:

 Input: 0

Output: 2

How many more input elements can we accept?

Let B be a buffer.

Let S be the size of the buffer B in bytes.

Let I be an index into the buffer where the producer will store the next new byte of data.

Let O be the index of the next byte that the consumer should remove from the buffer.

Let N be the number of unconsumed bytes in the buffer.

Define % as the modulus operator.

Initially, I = O = N = 0.

The buffer is full (has no room for new data) when N == S.

The available space (for new data) A = S - N

To Add m bytes of data from buffer D to the buffer B the producer will:

(1) Check that m <= A (if not an error has occurred)
(2) put bytes into the buffer using this model:

```
int j = I;
I = (I+m)%S
N += m;

for (int q = 0; q < m; q++)
    B[(j+q)%S] = D[q]
```

To remove r bytes from the buffer B to buffer D, the consumer will:

(1) Check that r <= N. If not, adjust r (r = N) or signal error.
(2) take bytes from the buffer using this model:

```
int j = O;
O = (O+r)%S
N -= r

for (int q = 0; q < r; q++)
  D[q] = B[(j+q)%S]
```

So, you see that the idea is that the input (I) and output (O) pointers change continuously from the beginning of the buffer to the end and then wrap around back to the beginning again. Conceptually, it appears as if the end of the buffer is connected back the front of the buffer as if to form a ring (or circle). We enforce that the input pointer never tries to overtake the output pointer!

To make these two methods thread safe, we need only to protect the 3 lines of code that update the class variables O, N, I: NOT the loops that move data! This is a better real-time approach than serializing access to the loop itself, or worse, the entire object.

- Elements are all same size and type

  - Elements are typically primitives (byte, int, etc) but can be pointers or even structures

- Finite

  - Fixed space must be allocated a priori

- Low overhead

  - No "per element" costs like we have in a Queue

- Elements MUST be processed in order.

- Elements are linked together in a list

- List can be single (forward) or double (forward and backward) linked

- Queue Control Block contains (as a minimum) pointer to first element (head) and last element (tail)

- Queues are almost always used as FIFOs, but can support iteration, random access, and reverse (LIFO) processing

Queue Control Block

head | a
tail | z

**Payload can be ANY object or structure.
Elements need not contain similar payloads**.

Forward link

b → z → null

Payload

element a   element b   element z

Queue Control Block

head | a
tail | z

Forward link | b → z → null
Backward link | null ← a ← b
Payload

element a   element b   element z

- # Required Operations

  - Put (add to tail)

  - Get (get from head)

- # Nice to Have Operations

  - Remove (remove specific element)

  - Insert (add element after a specific element)

- # *Deluxe* Operations

  - Peek (non-destructive Get)

  - Put to head

  - Get from tail

  - Iterate (head to tail or tail to head)

- Not fixed in length ("unlimited" in length)

- Does not require pre-allocated memory

- Allows processing of elements in arbitrary order

- Can accommodate elements of different type

- Additional per element cost (links)

- Stream data: Use a ring buffer

  - Arriving elements are primitives that make up a data "stream" (no record boundaries)

    - TCP data is an example

- Service requests: Use a queue

  - Arriving elements are requests from a user layer (or clients) and must be processed individually.

- Let's define the idea of a "**machine**"

  - Organism (real or synthetic) that responds to a countable (**finite**) set of stimuli (**events**) by generating predictable responses (**outputs**) based on a history of prior events (current **state**)

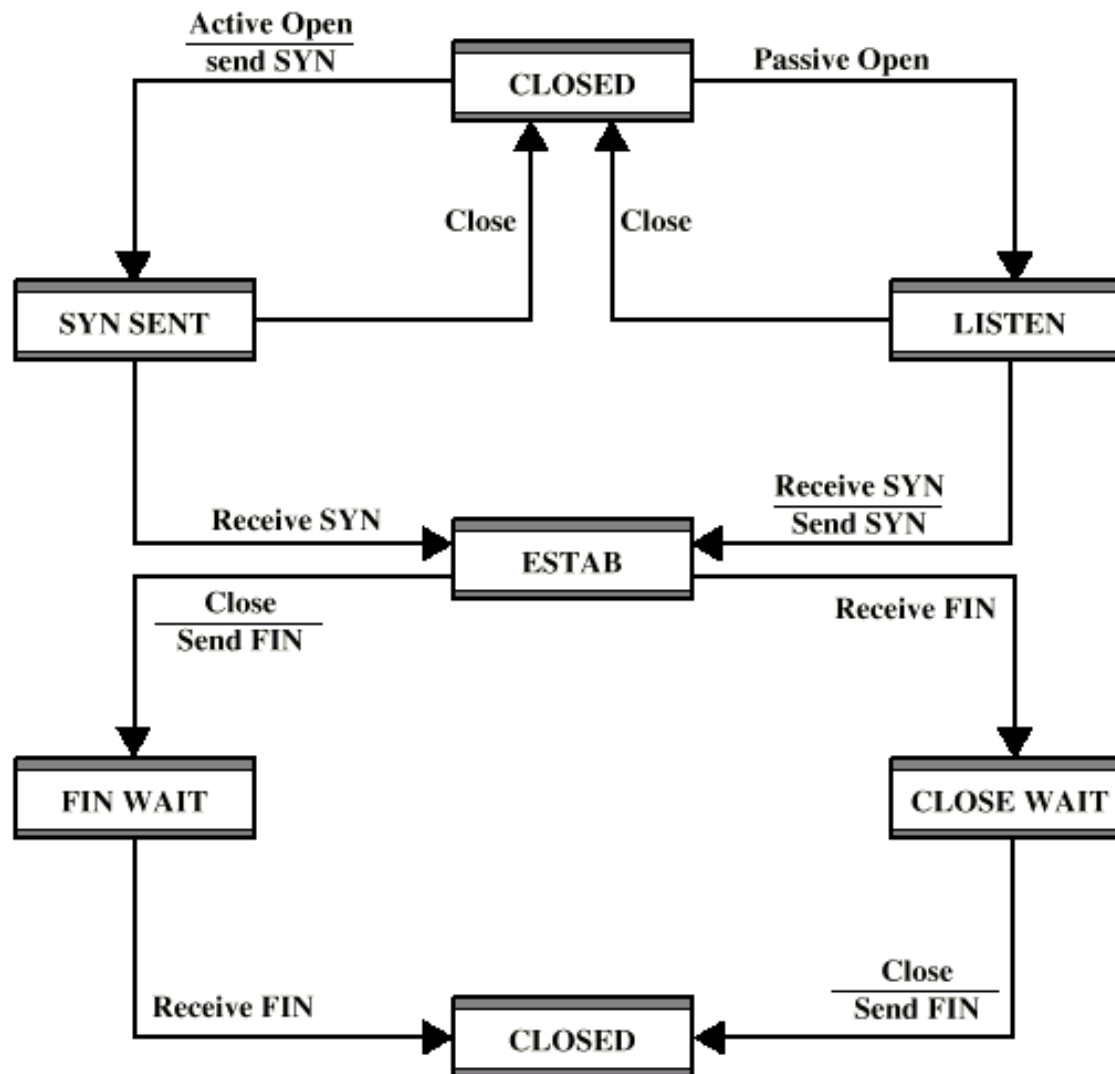- A finite state machine (fsm) is a computational model of a machine

- **States** represent the particular configurations that our machine can assume

- **Events** define the various inputs that a machine will recognize

- **Transitions** represent a change of state from a current state to another (possibly the same) state that is dependent upon a specific event

- The **Start State** is the state of the machine before is has received any events

- **Mealy** machine

    - one that generates an output for each transition

- **Moore** machine

    - one that generates an output for each state

- Moore machines can do anything a Mealy machine can do (and vice versa)

- In my experience, Mealy machines are more useful for implementing communications protocols

- The fsm that I'll provide is a Mealy machine

- Identify

    - *States*

    - *Events*

    - *Transitions*

    - *Actions* (outputs)

- Program these elements into an FSM

- Define an event classification process

- Drive the events through the FSM

- Example ….

Agenda

1	Session Overview
2	Application Layer
3	Summary and Conclusion

Assignments & Readings

- Readings
  - » Chapters 1 and 5
- Assignment #3

Next Session: Data Link Control