CS 118: Data Link Intro, Framing and Error Detection

George Varghese

October 2022

PRIZE QUESTION

SLOW ELEVATOR PROBLEM: HOW TO DEAL WITH PROBLEM MORE CHEAPLY THAN BUYING A NEW ELEVATOR (HOW ENGINEERS THINK)

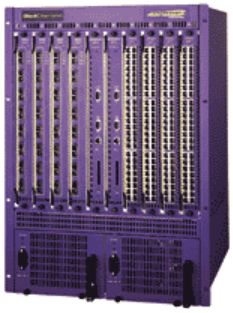
Physical layer (Lectures 2-4)

802.11b Wireless Access Point

2.4Ghz Radio DS/FH Radio (1-11Mbps)

Cat5 Cable (4 wires) 100Base TX Ethernet 100Mbps

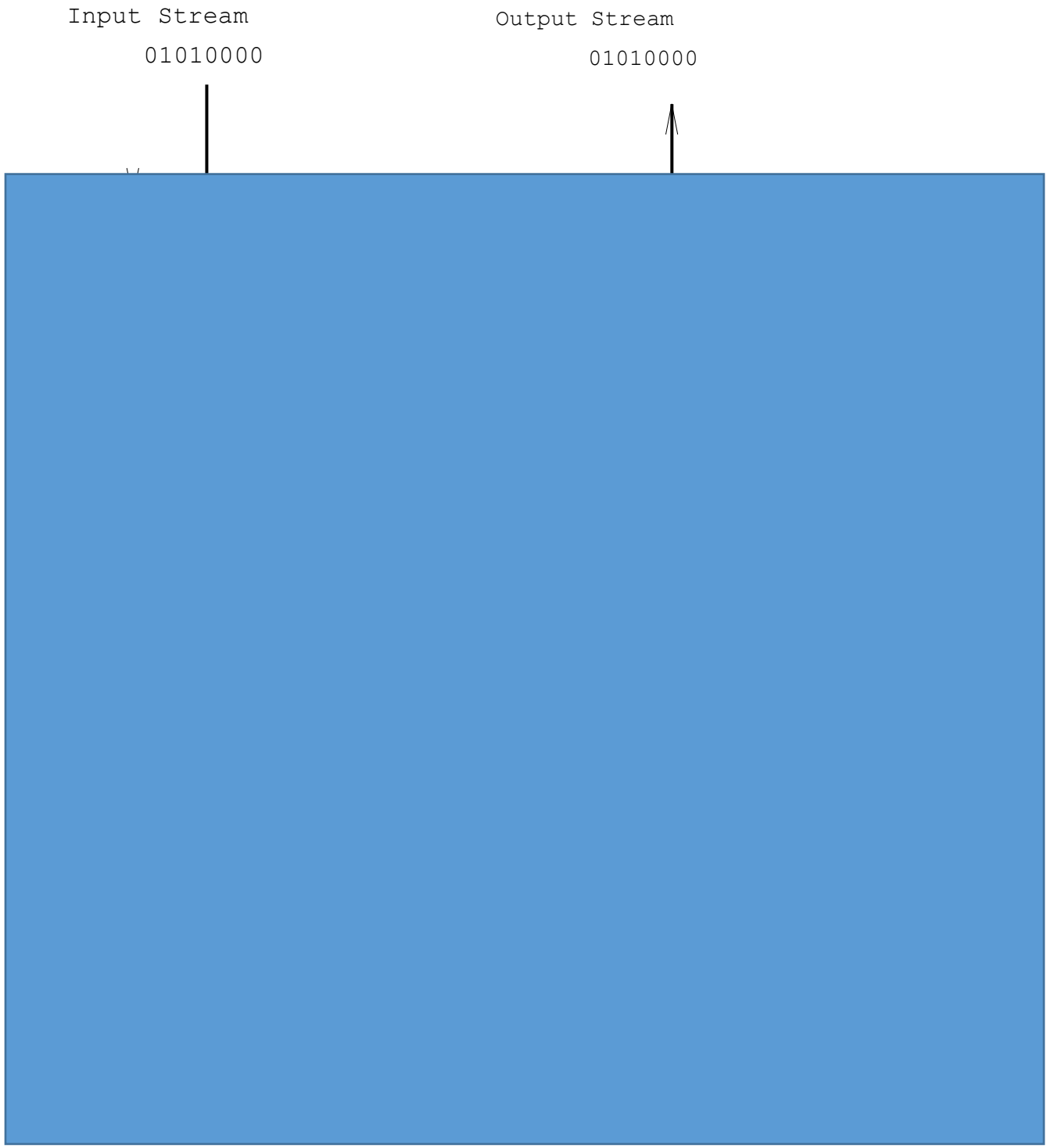
Ethernet switch/route



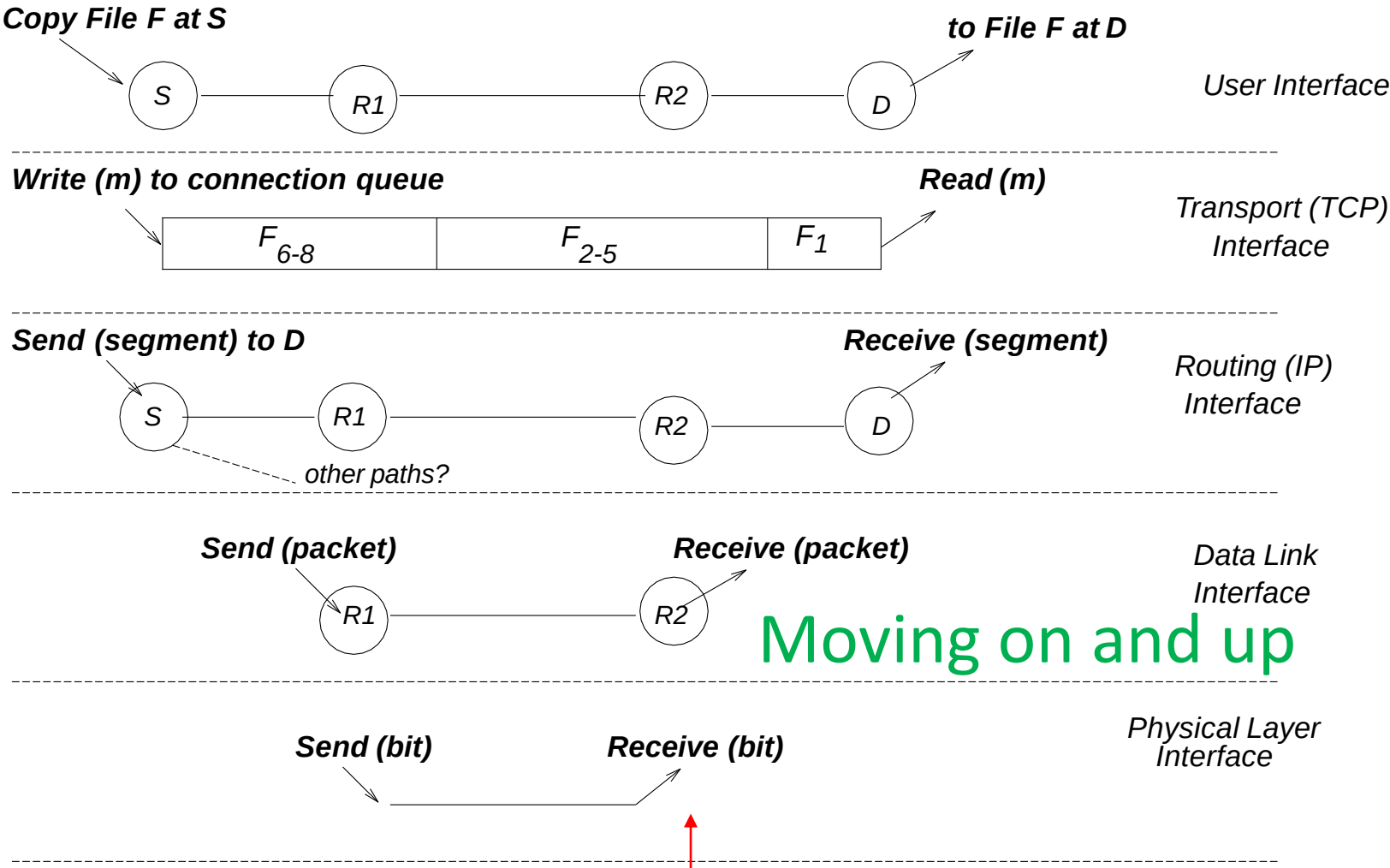
To campu backbone

62.5/125um 850nm MMF 1000BaseSX Ethernet 1 Gbps

PHYSICAL LAYER: SUBLAYERS



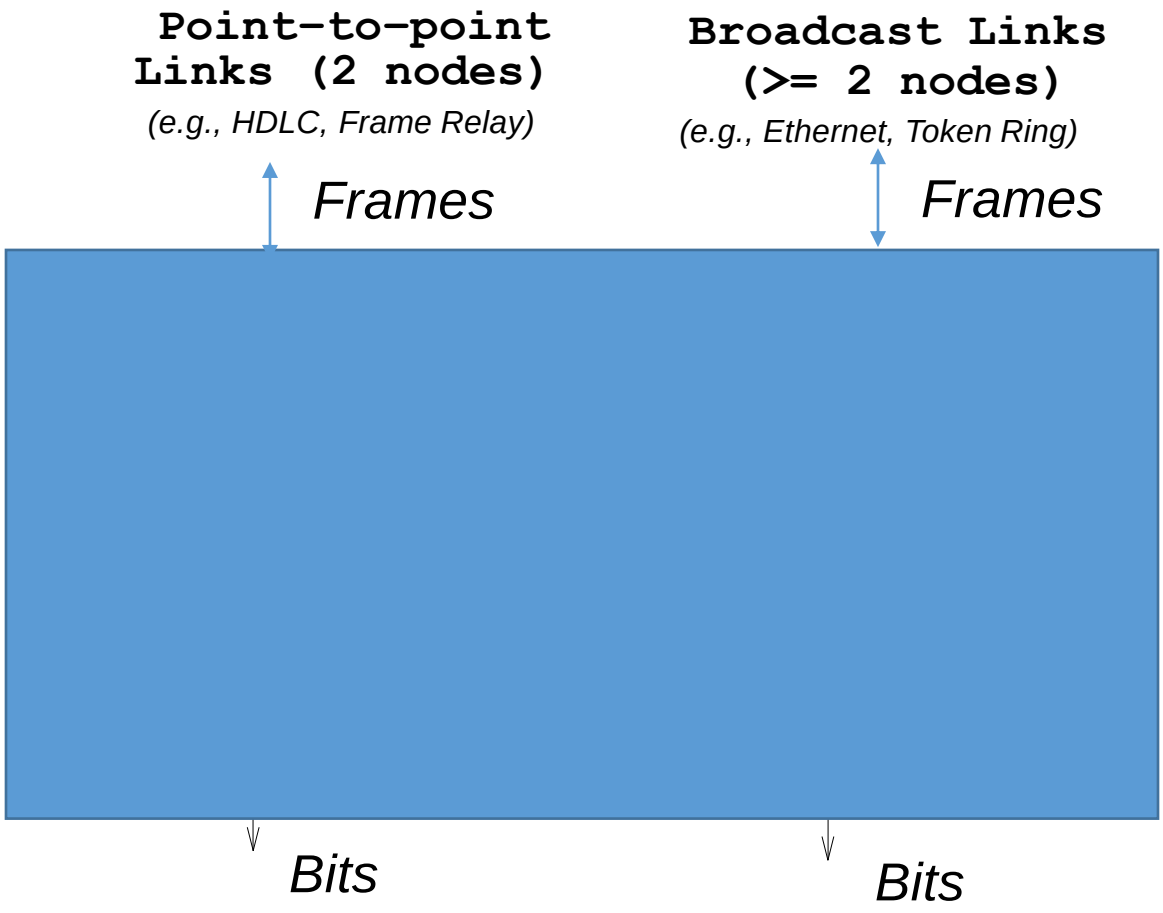
SEMI RELIABLE 1 HOP BIT PIPE



Been there, done that

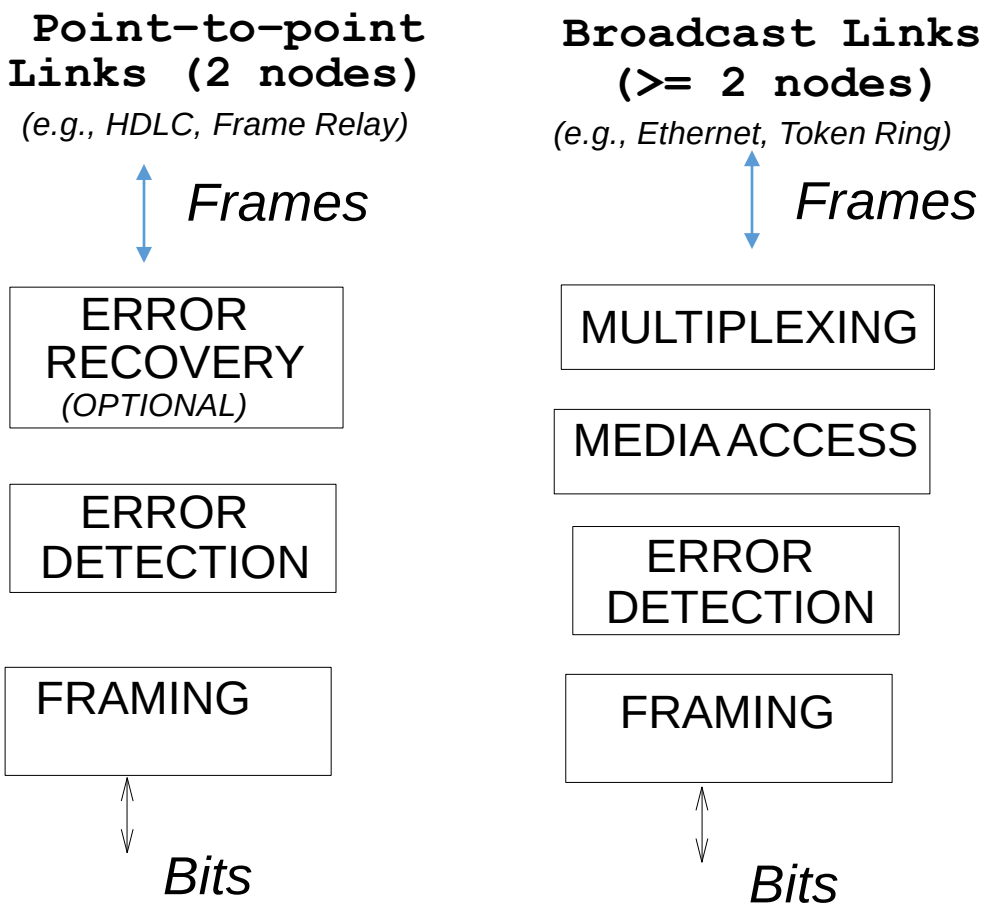
RECALL THE IP ABSTRACTIONS:

Data Link Sublayers



QUASI-RELIABLE 1 HOP FRAME PIPE

Data Link Sublayers



QUASI-RELIABLE 1 HOP FRAME PIPE

Five functions of Data Link

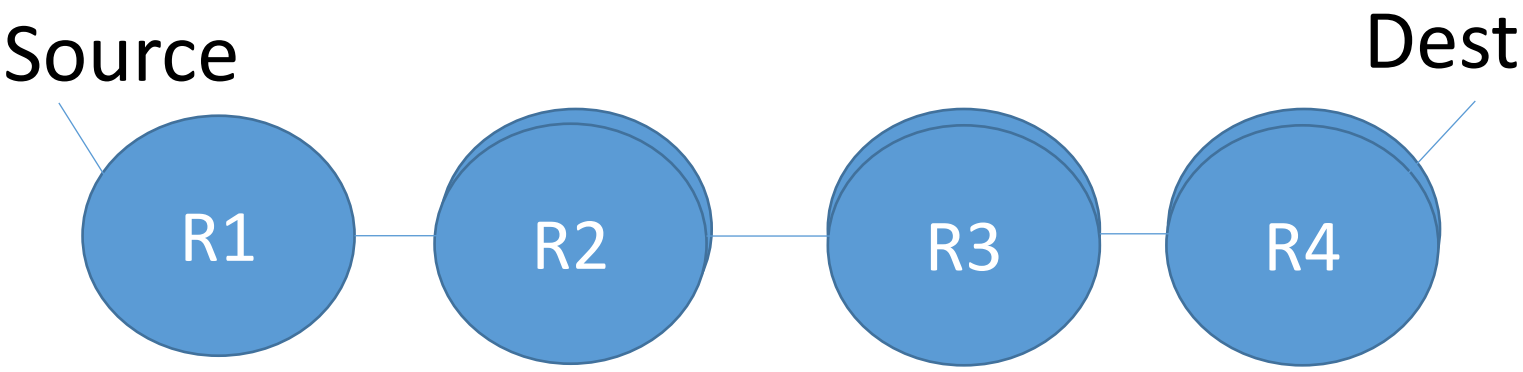
Five functions:

- Framing: breaking up a stream of bits into units called frames so that we can extra information like destination addresses and checksums to frames. (Required.)
- Error detection: using extra redundant bits called checksums to detect whether *any* bit in the frame was received incorrectly. (Required).
- Media Access: multiple senders. Need traffic control to decide who sends next. (Required for broadcast links).
- Multiplexing: Allowing multiple clients to use Data Link. Need some info in frame header to identify client. (Optional)
- Error Recovery: Go beyond error *detection* and take recovery action by retransmitting when frames are lost or corrupted. (Optional)

Why not do error recovery at each hop?

- End-to-end argument: Need end-to-end or transport error recovery anyway. Can't trust a series of hop-by-hop schemes because:
  - Crashes and other losses at intermediate nodes.
  - Transport must work over both reliable and unreliable links.
- Thus hop-by-hop only a performance optimization
- Extra cost (ack messages, buffering) not worth it when error rate is low.
  - Worth it (quicker recovery, less wasted resources) when error rate is high and the latency of end-to-end recovery is unacceptable (within a compute cluster or storage area network).

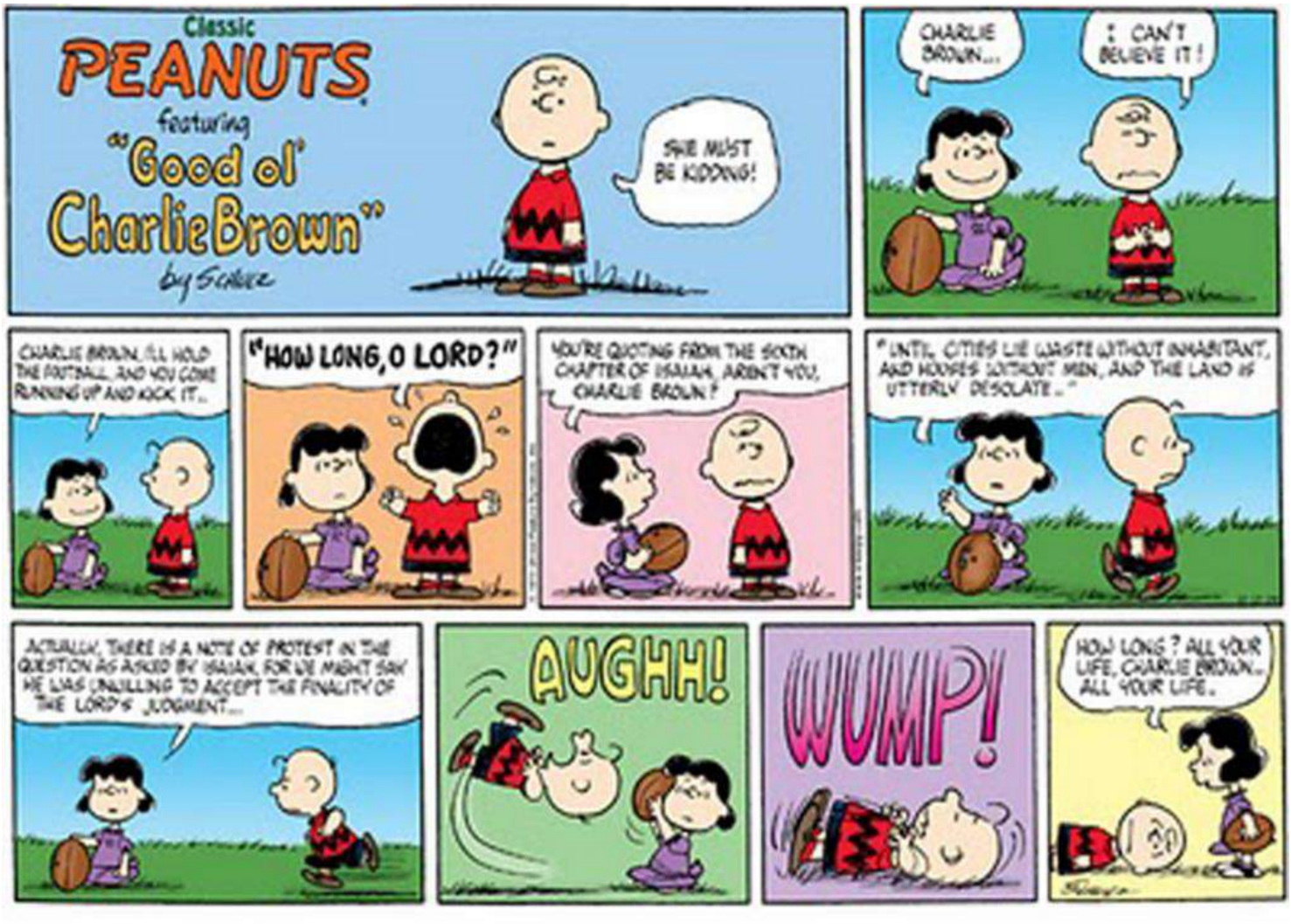
End to end vs hop by hop example



- Case 1: suppose no bit errors and messages are 100 bytes and acks 50 bytes. 50% overhead for hop by hop. Hop-by-hop worthless!
- Case 2: Suppose half the packets are being dropped on last hop between R3 and R4. Then without hop by hop we would retransmit on all hops using only end to end (6n versus 2n) where n is number of packets Hop by hop worthwhile!

How can we breakup a series of bits from the physical layer into a series of frames?

This Lecture: Framing



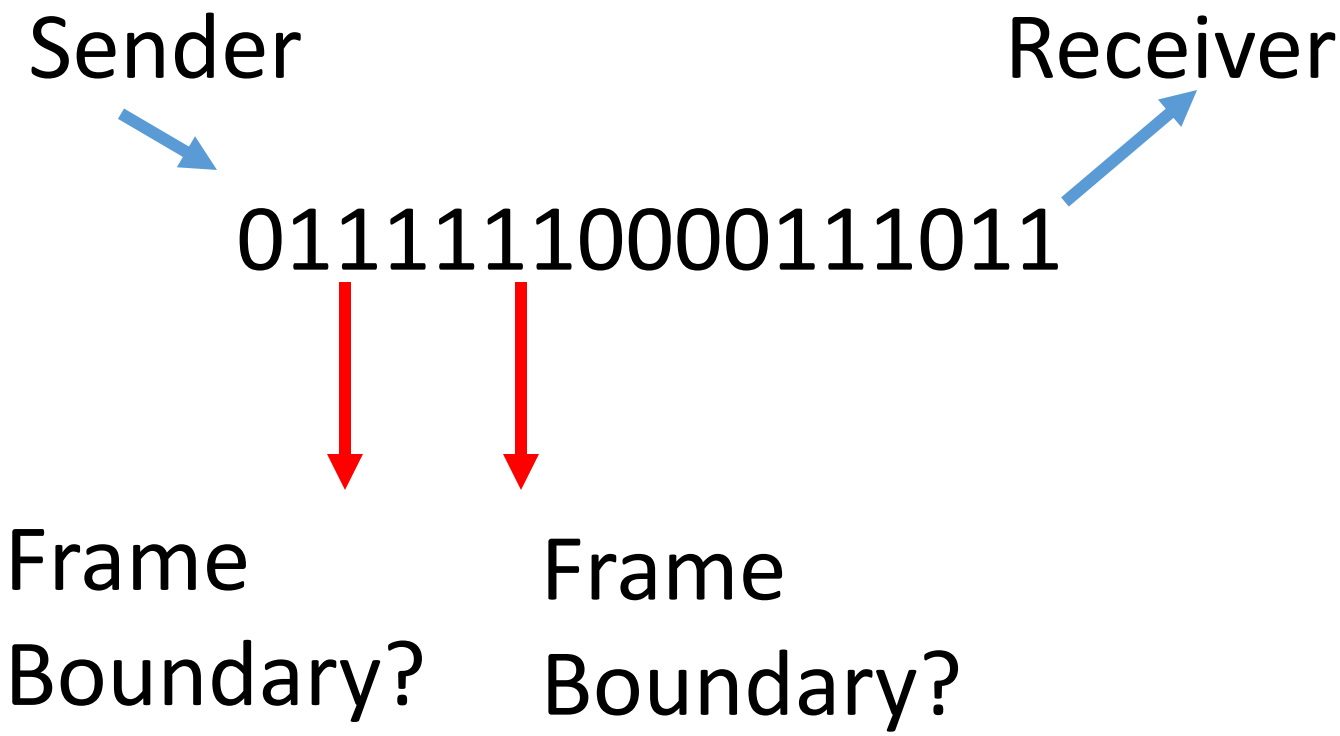
Breakout: why Framing

- Headers: ?
- Error recovery?
- Multiplexing?

What do I mean by this? Please think and discuss in your breakout groups

Breakout: how Framing

Brainstorm this as well in your breakout groups



Given a stream of bits, how in the world is a poor receiver going to find the frame boundaries? Think! There are lots of ways.

Why Framing

- Without framing, the bit stream is reserved for one sender and one client per sender. Need frames to add multiplexing information like destination addresses and destination client names.
- Frames offer a small, manageable unit for error recovery and error detection. Add checksums to frames for error detection and sequence numbers etc. for error recovery.

How Framing

- Flags and Bit Stuffing: Use special bit patterns or flags to delimit (i.e., mark boundaries) frames. Need *bit stuffing* to "encode" the user data not to contain the flags. (e.g., HDLC)
- Start Flags and Character Count: Use flags to indicate the start of data and a bit count to indicate end of frame. Bit stuffing not needed but less robust. (e.g., DDCMP)
- Flags supplied by Physical Layer: Use special physical layer symbols to delimit frames.

Fixed-Length Frames

- Easy to manage for receiver
  - Well understood buffering requirements

- Introduces inefficiencies for variable length payloads
  - May waste space (padding) for small payloads
  - Larger payloads need to be fragmented across many frames
  - Very common inside switches/routers

- Requires explicit design tradeoff
  - ATM uses 53-byte frames (cells)
  - Why 53? 48 + 5. Why 48?

Length-Based Framing

- To avoid overhead, we'd like variable length frames
  - Each frame declares how long it is
  - E.g. DECNet DDCMP

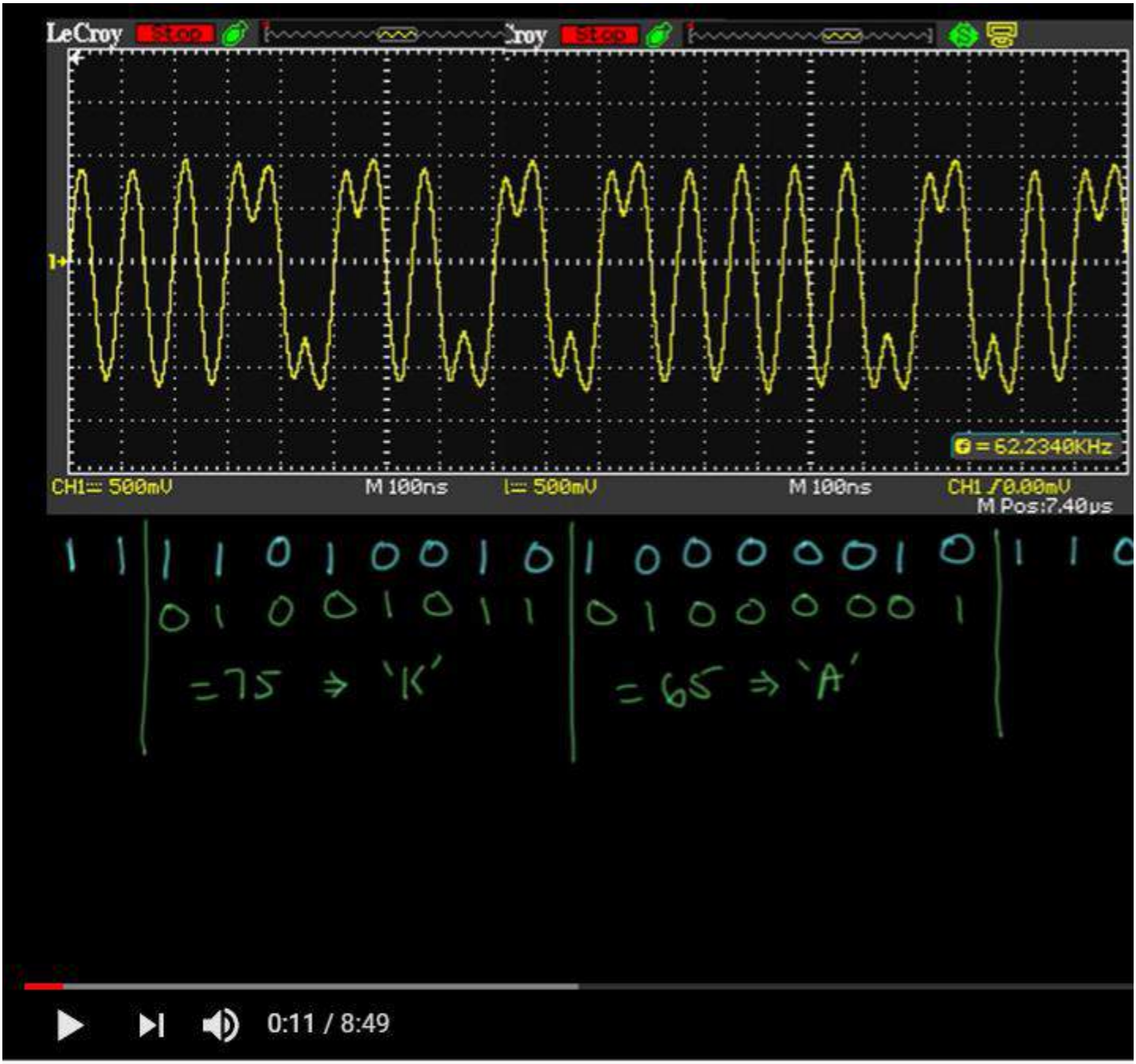
- What's the issue with explicit length field?
  - Must correctly read the length field (bad if corrupted)
    - Need to decode *while* receiving
  - Still need to identify the beginning…

Sentinel-based Framing

- Allow for variable length frames
- Idea: mark start/end of frame with special "marker"
  - Byte pattern, bit pattern, signal pattern
- But… must make sure marker doesn't appear in data

- Two solutions
  - Special non-data physical-layer symbol
    - Code efficiency (can't use symbol for data)?
  - Stuffing
    - Dynamically remove marker bit patterns from data stream
    - Receiver "unstuffs" data stream to reconstruct original data

Lets check out Ben Eater again



The importance of framing | Networking tutorial (5 of 13)

https://www.youtube.com/watch?v=xrVN9jKjIKQ

What Ben does well

- Shows how physical layer signals interact with framing
- Motivates framing by showing how you misread characters
- Shows why flags can go wrong and why you need bit stuffing

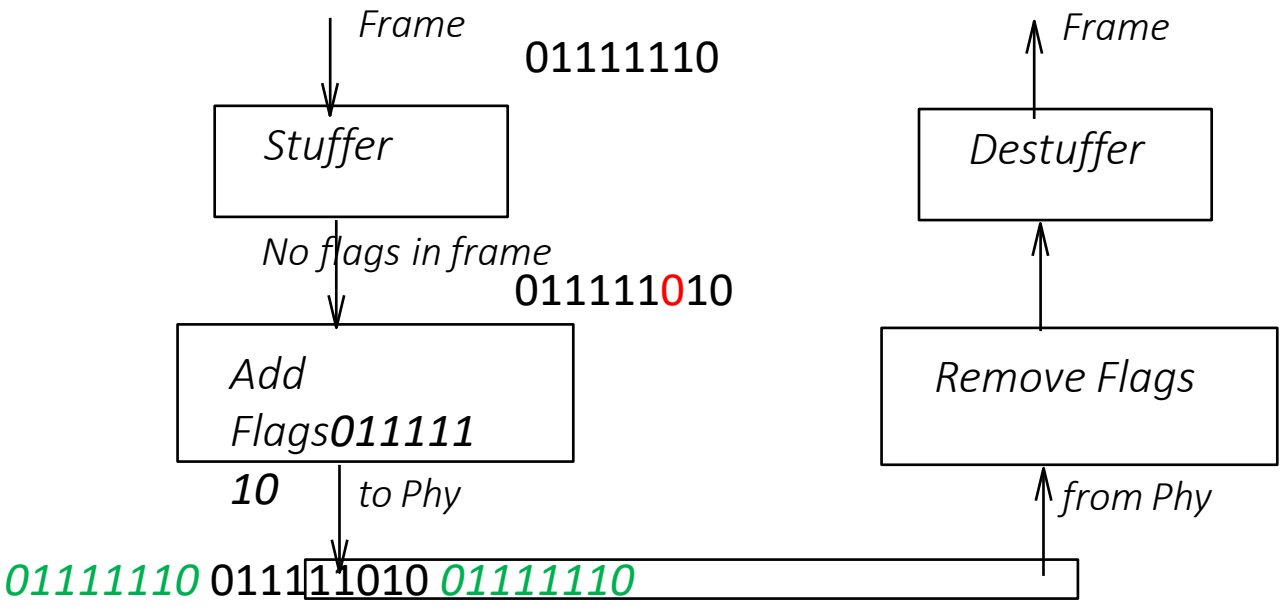
Do use Youtube videos but try and see the deeper issues. We want you all to be curious, creative computer scientists

What Ben misses

- Is framing needed only to get into byte synchronization for ASCII? No!
- How should you write code for bit stuffing?
- How do you know bit stuffing always works? Is there a proof?
- Do other flags work? 01010101?
- Are there other framing methods? (To be sure, Ben's shows one more: Ethernet). What are the tradeoffs

I'd like you as UCLA Computer Science Majors to go a little deeper . . . .

Stuffing Design by Sublayering



- Sublayering is a good design technique within layers as well!
- What happens if input data contains 0111110. If receiver gets 111110?

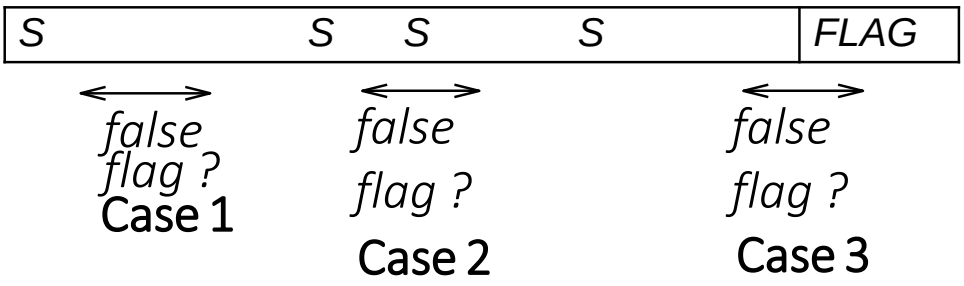
Bit-level Stuffing

- Avoid sentinel bit pattern in payload data
  - Commonly, sentinel is bit pattern 01111110 (0x7E)
  - Invented for SDLC/HDLC, now standard pattern
- Sender: any time five ones appear in outgoing data, insert a zero, resulting in 0111110

  01111110000111011101111011111001

  01111101000011101110111110011111**0**001
- Receiver: any time five ones appear, removes next zero
  - If there is no zero, there will either be six ones (sentinel) or
  - It declares an error condition!
  - Note bit pattern that cannot appear is 01111111 (0x7F)
- What's the worst case for efficiency?

Correct Bit Stuffing



- Case 1: Easy, or we would have added a stuffed bit
- Case 2: Stuff after 0 followed by 5 1's does not work! Counterexample?
- Case 3: Flag = 01010101 and stuff a 1 after 1 after 010101 does not work!

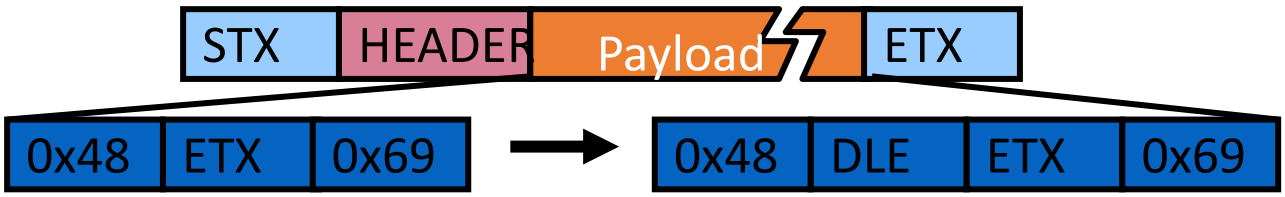
011111 1111110

011111 01111110

false
flag ?
Case 2

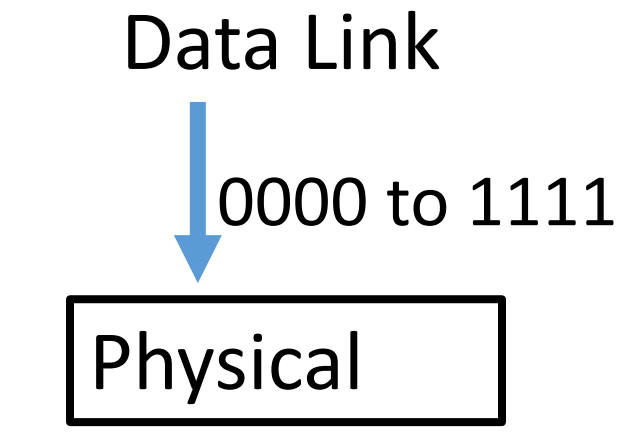
Byte Stuffing

- Same as bit stuffing, except at byte (character) level
  - Generally have two different flags, STX and ETX
  - Found in, DDCMP, BISYNC, etc.
- Need to stuff if either appears in the payload
  - Prefix with another special character, DLE (data-link escape)
  - New problem: what if DLE appears in payload?
- Stuff DLE with DLE! Like Escape characters . . .
  - Could be as bad as 50% efficient to send all DLEs

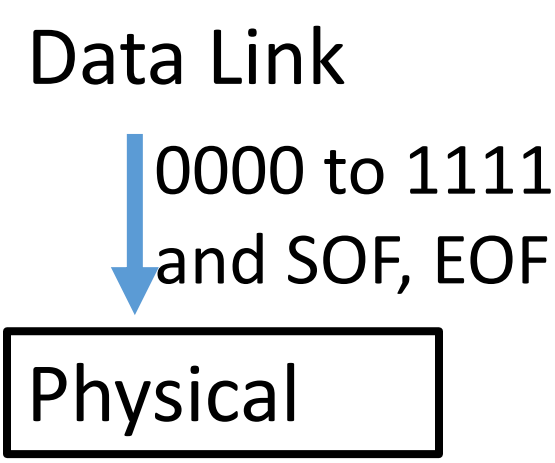


The Engineering solution

- Ask the Physical Layer for more symbols that are never used in data.
- Easily possible in 4-5 coding as we have 16 data symbols but 32 possible encoded values
- Even after ruling out 00000 and 111111 we still have 14 unused symbols.
- Change interface to allow sending one of 16 data symbols (0000 to 1111) and special symbols for start of frame (SOF) and End of Frame (EOF)



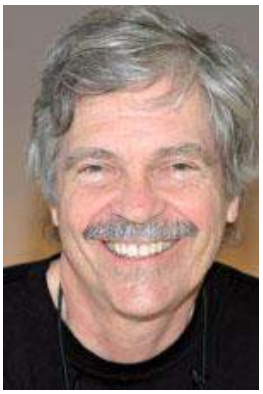
OLD INTERFACE



NEW INTERFACE

Insight versus Brute Force Cleverness

A change in perspective is **worth** 80 IQ **points**. ... Alan Kay



Useful Principles

- Each layer or sublayer exacts its penalty: e.g., clock recovery coding, framing bits.
- The end-to-end argument.
- Lower Layers should not depend for correctness on assumptions about higher layers.
- All layers have some common problems to solve: e.g., synchronization, multiplexing.
- The best principles will/should be violated for pragmatic reasons.
- Layering and Sublayering are a good way to understand and design new protocols.