DIRECT COMMUNICATION LINKS

v 4.0 30th/March/2020

Slides, their text and graphics are Based on textbook *Conceptual Computer Networks* by:

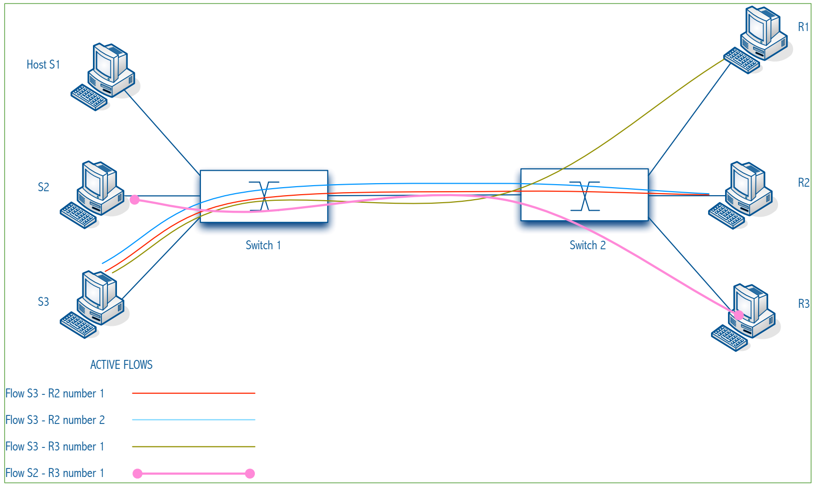
Chapter 2 outline

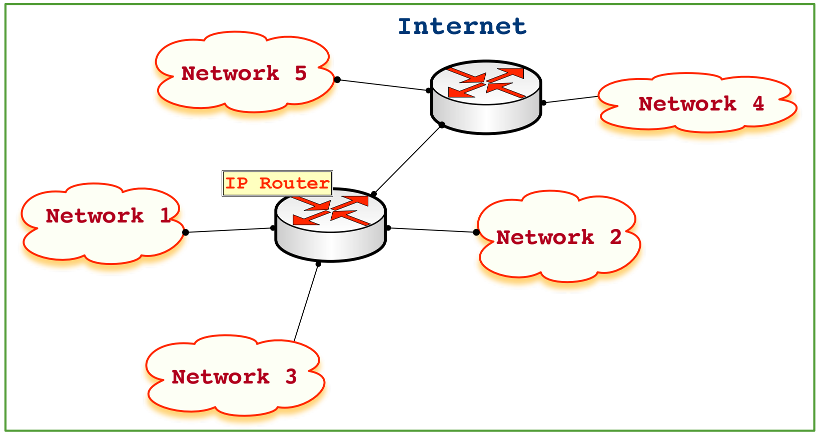
- Section 1
  - How Datalink protocols build frames (Framing)
  - Overview of Link protocols for point to point communication
  - Intro to multiple-access protocols and the original Ethernet
- Section 2
  - Ethernet's CSMA/CD and Exponential Backoff

The scenario for Chapter 2

3

- **Statistical multiplexing**
  - Switching according to distribution of demand across all connected nodes
- **Directly connected nodes**
  - Host – Switch
  - Host – Host
  - PDU is Frame
    - 1 Packet into 1 Frame
- **Packet Switched Networks**
  - Information is broken down into individual Packets





Building frames

What is a Frame?

- Datalink protocols are used for <u>direct links</u>

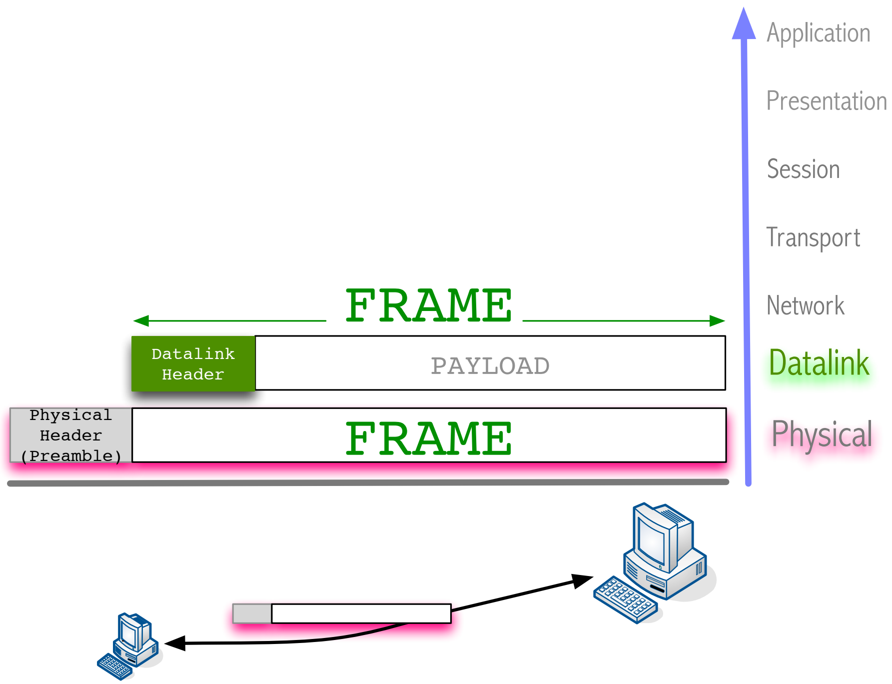
- The PDU (Protocol Data Unit) of Datalink protocols is known as Frame

- Datalink FRAME =

  Datalink Header

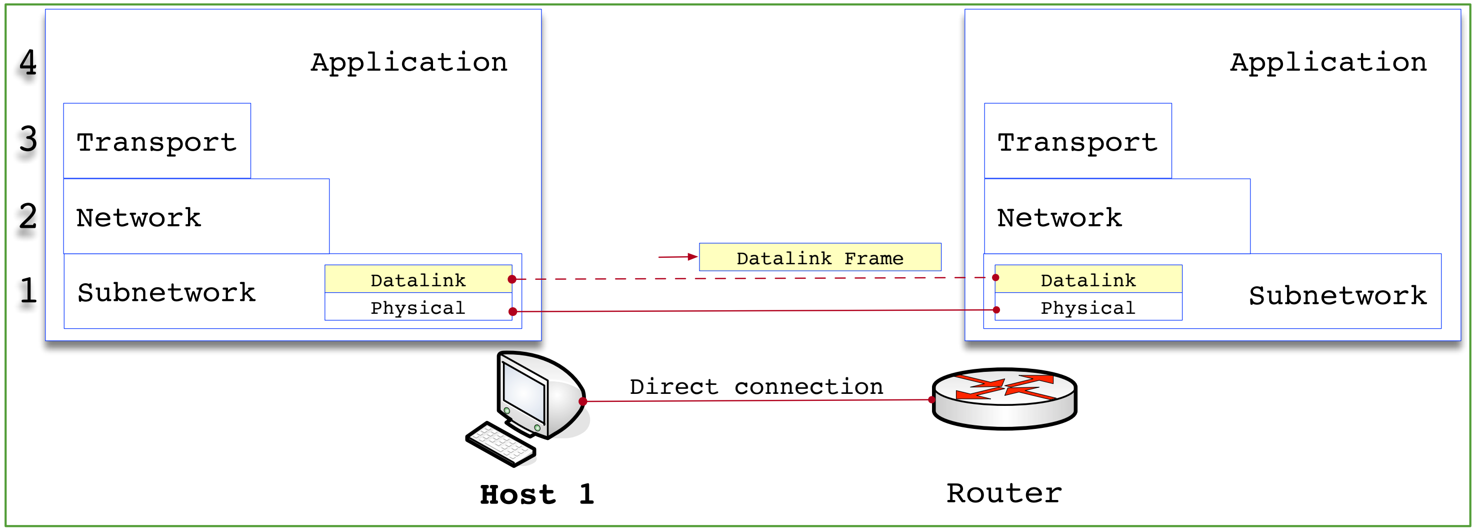
  \+

  Upper layer payload



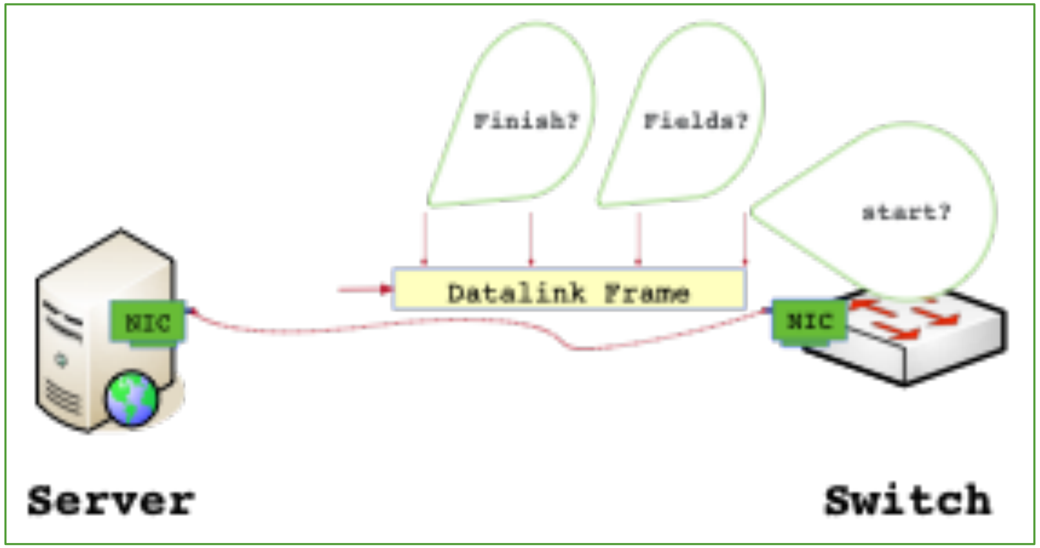
What is a direct connection?

- Example: Host1 is directly connected to the Router
- Frame:
  - Payload: Encapsulates an upper-layer PDU
  - Header contains
    - A mux key + Host 1 address + Router address, etc



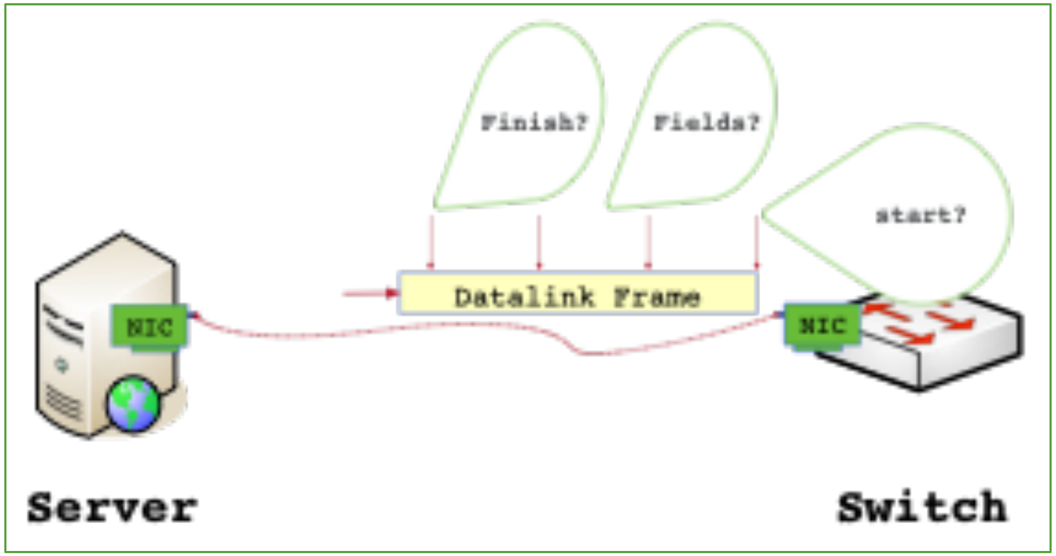
Detection of Frame's fields

- Server transmits a frame to the Switch
  - Network Interface Card = NIC
  - Transmission electronics
- NIC at the receiver (Switch) stores the received sequence of bits
- The Switch NIC must be able to recognize the frame:
  - Where the frame <u>begins</u> and <u>ends</u>
  - Which are the frame's <u>fields</u>



Three strategies for delineating a frame

- Detection of frame start, its fields and its end
- Three strategies:
  - Byte-oriented protocols: BISYNC, PPP, DDCMP
  - Bit-oriented protocols: HDLC, Ethernet
  - Clock-based protocols: SONET/SDH



An analogy with C lang strings

- C strings are byte-oriented ☺
- How is a C constant character string delimited in the source code?

```
char s[] = "Hello world!";
```

- “ **sentinel** marks the beginning
- **Next “ sentinel** marks the end
  - ASCII Characters are stored in between the two delimiters

Framing in Byte-oriented protocols

- A frame is made of a collection of bytes

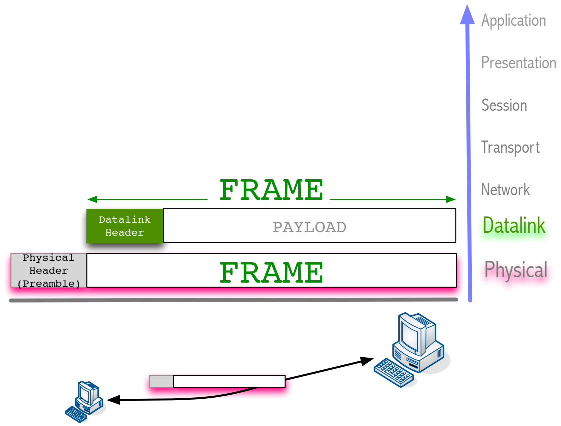
- BISYNC (Binary Synchronous Communication, BSC)
  - Developed by IBM (late 1960)

- DDCMP (Digital Data Communication Protocol)
  - Used in DECNet

- PPP (Point to Point Protocol)
  - IP packets over various media

Framing in the BiSync protocol

- BISYNC is a byte-oriented protocol
- Each byte represents an ASCII/EBCDIC code
- **Sentinels:**
  - **SYN SYN** characters mark the beginning of a new frame
  - **SOH** (Start of Header): Mark the start of the Header
  - **STX** (Start of Text): Mark the start of the data (Payload)
  - **ETX** (End of text): Marks the end of the data



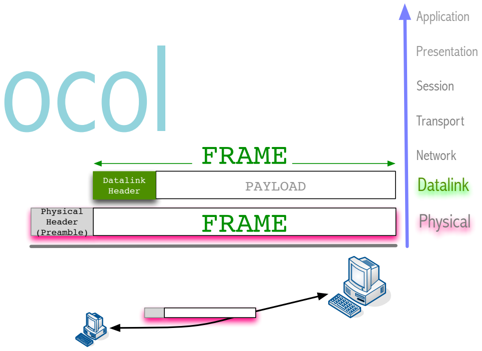
Frame Start		Header Start			Payload Start		Payload End	
SYN	**SYN**	**SOH**	Identifier	Station Address	**STX**	Payload	**ETX**	CRC-16
			Header					Error Control

Single-block BSC Frame format (Data)

Transparency in the BiSync protocol

- What if the payload sent by the upper protocol contains a byte coincident with any of the sentinels? This would confuse the receiving protocol
- A special control character known as DLE (Data Link Escape) indicates that the next character is not to be understood as a sentinel but as pure literal data (Byte stuffing)
- Example:
  - We want to send the following ASCII character sequence as data:
    `[A][B][C][D][E][STX][F][G]`
  - The STX char is not to be understood as meaning "Start of TeXt" but its 8 bits mean only payload data

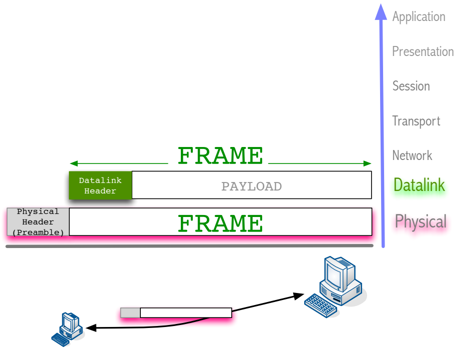
Transparency in the BiSync protocol



- Example of Byte-stuffing
  - We want to send the following ASCII character sequence as data: `“[A][B][C][D][E][`**`STX`**`][F][G]”`
  - The STX char must not be understood to mean “Start of TeXt” but its 8 bits are only payload data
  - A [DLE] character is included prior to [STX] meaning: `“The next character is data, it is not the Bisync sentinel known as[STX]”`
  - The transmitted sequence becomes:
    - `“[A][B][C][D][E]`**`[DLE][STX]`**`[F][G]”`

- What if DLE itself appears in the layer-3 payload? Same as in the C language: Include an escaping DLE character that escapes the special meaning of the next character: [DLE][DLE]

Framing in the BiSync protocol



- What if [DLE] itself is to be included as payload data?
  - Same as in the C language
  - Include an escaping DLE character that escapes the special meaning of the next character: [DLE][DLE]

- Example. The payload is the next byte sequence:
  - [1] [2] [3] [DLE] [4] [5] [6]
  - BiSync will transmit the following byte sequence:
    [1] [2] [3] [DLE] [DLE] [4] [5] [6]

Framing in PPP (Point To Point Protocol)

- Byte-oriented (A variant of HDLC-ABM protocol)
  - Address and Control fields use constant values since PPP is used only for point-to-point communication

- Uses the sentinel approach
- Over Internet links (ISDN/ADSL/ATM)
- Frame start character sentinel is denoted as `Flag`

  `0 1 1 1 1 1 1 0`
- Protocol: A multiplexing key (Example: IP / IPX)
- Payload: The data transported, max size negotiated (MRU = 1500 bytes)
- CRC16 or CRC-32 for error detection

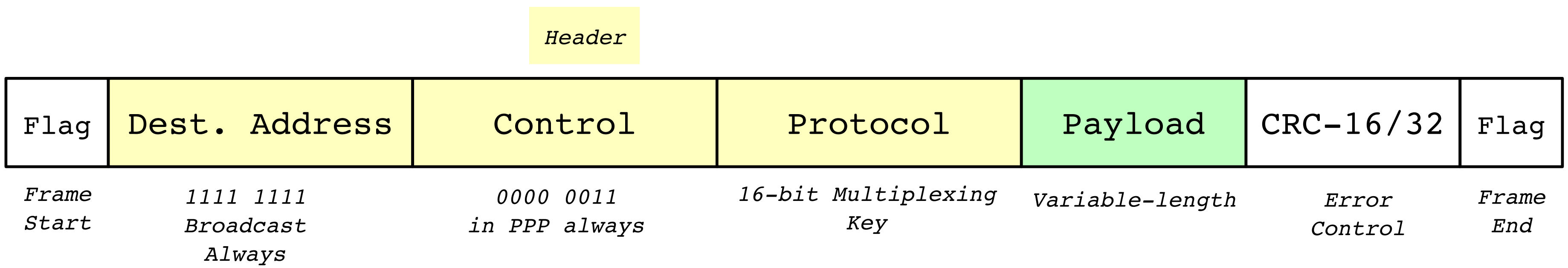
Header

Flag	Dest. Address	Control	Protocol	Payload	CRC-16/32	Flag
Frame Start	*1111 1111 Broadcast Always*	*0000 0011 in PPP always*	*16-bit Multiplexing Key*	*Variable-length*	*Error Control*	*Frame End*

Generic PPP frame

Point To Point Protocol

- Works in tandem with another two protocols
  - Negotiate parameters with:
  - LCP (Link Control Protocol): For testing and managing the link
  - NCP (Network Control Protocol): IP address, default router, etc

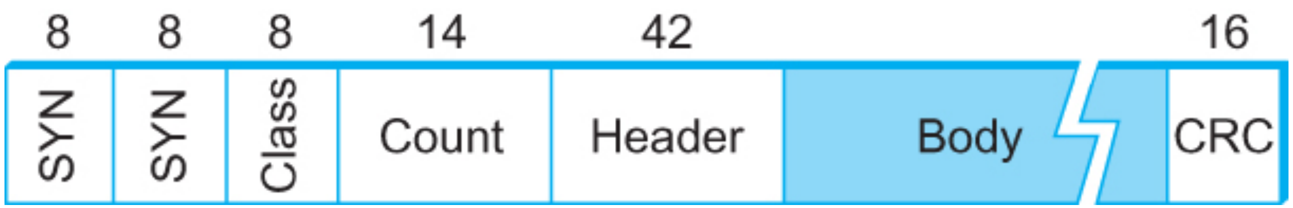


Generic PPP frame

Framing

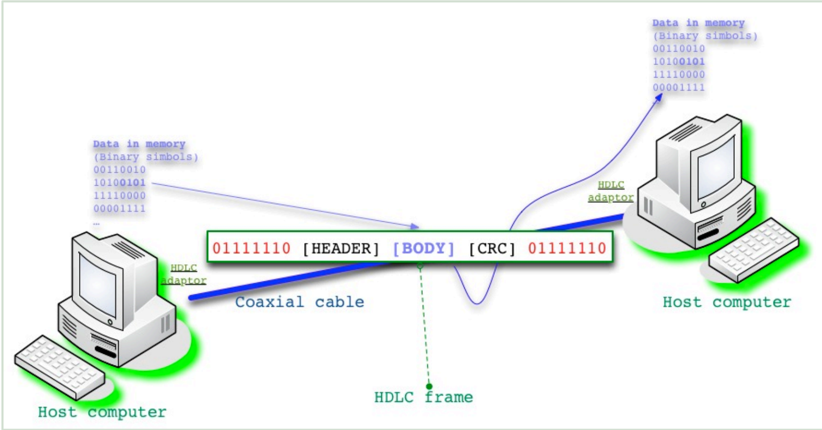
Byte-counting approach

- DDCMP
  - *count* : how many bytes are contained in the frame body
  - *-the rest of fields are fixed-size*
- If *count* is corrupted
  - Framing error



DDCMP Frame Format

Framing in HDLC



- HDLC : High Level Data Link Control
- HDLC is a bit-oriented protocol
  - Transmits data in blocks of 1 bit
- Beginning and Ending Sequence (Sentinel is the FLAG character)

FLAG = 0 1 1 1 1 1 1 0

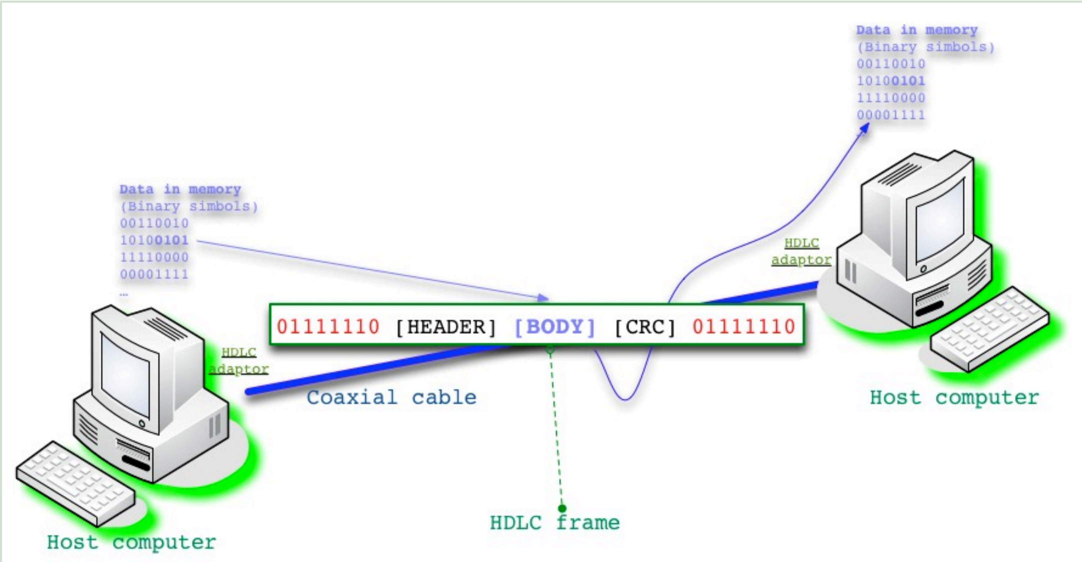
- Any amount of bits, not necessarily a multiple of 8 bits(1 byte)

		Header			
Flag	Address	Control	Information field	CRC-16/32	Flag
Frame Start	*8/16 bits*	*8/16 bits:* *· Information* *· Supervisory* *· Unnumbered*	*0 to N bits*	*Error Control*	*Frame End*

Generic HDLC frame

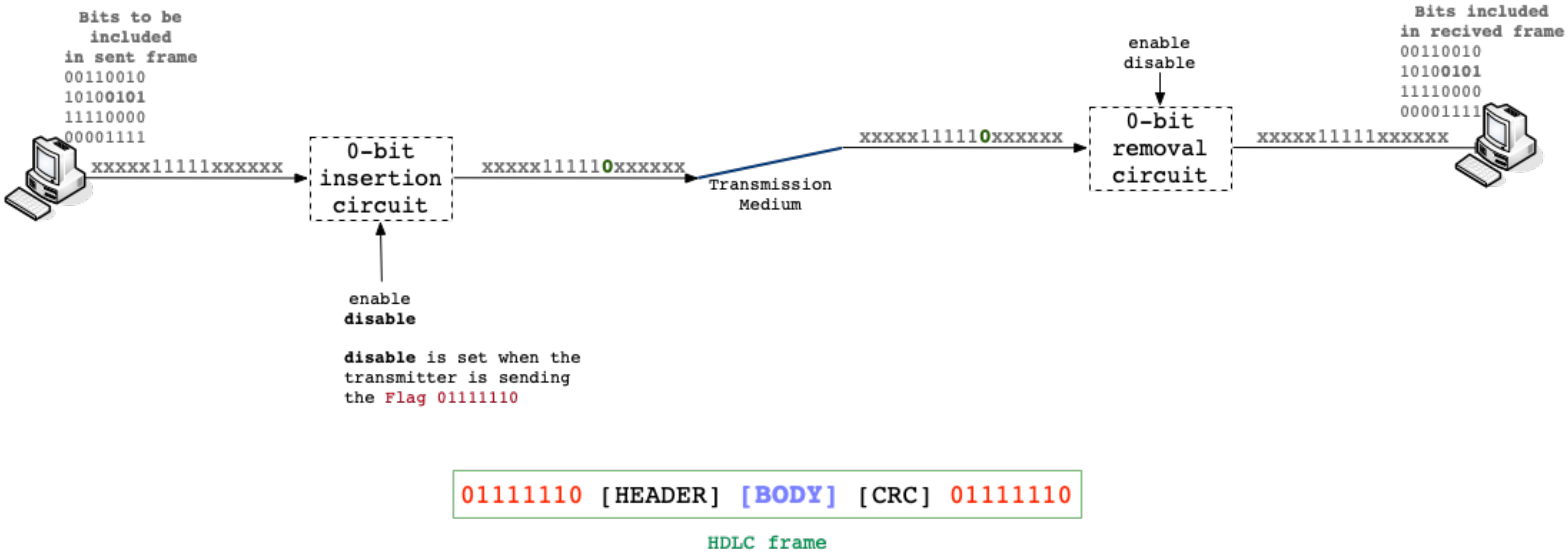
Data transparency in HDLC

- Problem with the Flag sentinel
  - What if the FLAG 0 1 1 1 1 1 1 0 is contained anywhere after the initial Flag?
  - The receiver would confuse this bit sequence with a Flag (A terminating flag, in this case)
- Solution: Bit Stuffing or Zero-Bit Insertion
  - A **transparency** mechanism for allowing the sender to send any bit sequence, including the sequence of bits that comprise the **Flag**



Bit stuffing in HDLC

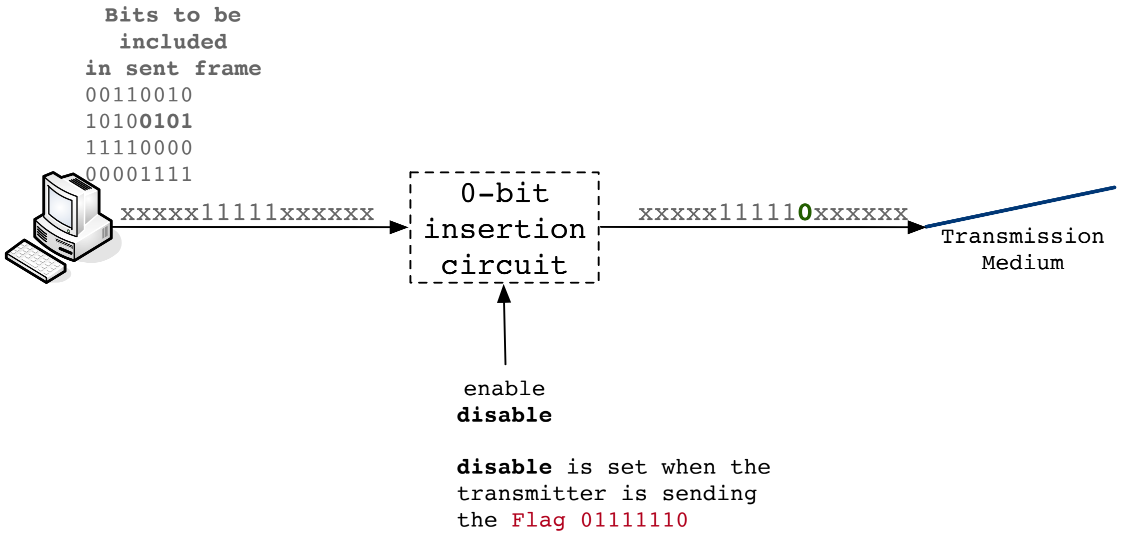
- At the sender: Whenever the sender observes 5 bits 1 after the frame start, the sender inserts a bit 0 before transmitting the next bit:
  - x x x x 1 1 1 1 1 0 x x x x x x x x
- At the receiver: The receiver removes the inserted bit 0 whenever it observes the 5 bits 1 and the next bit is a 0:
  - x x x x 1 1 1 1 1 _ x x x x x x x x



Bit stuffing in HDLC on the sending side

Bit stuffing on the <u>sending side</u>

- Any time five consecutive 1's appear in the frame (Between the start and end of frame)
  - x x x x 1 1 1 1 1 x x x x x x x x
- The sender inserts *(stuffs)* 0 before transmitting the next bit
  - x x x x 1 1 1 1 1 **0** x x x x x x x x

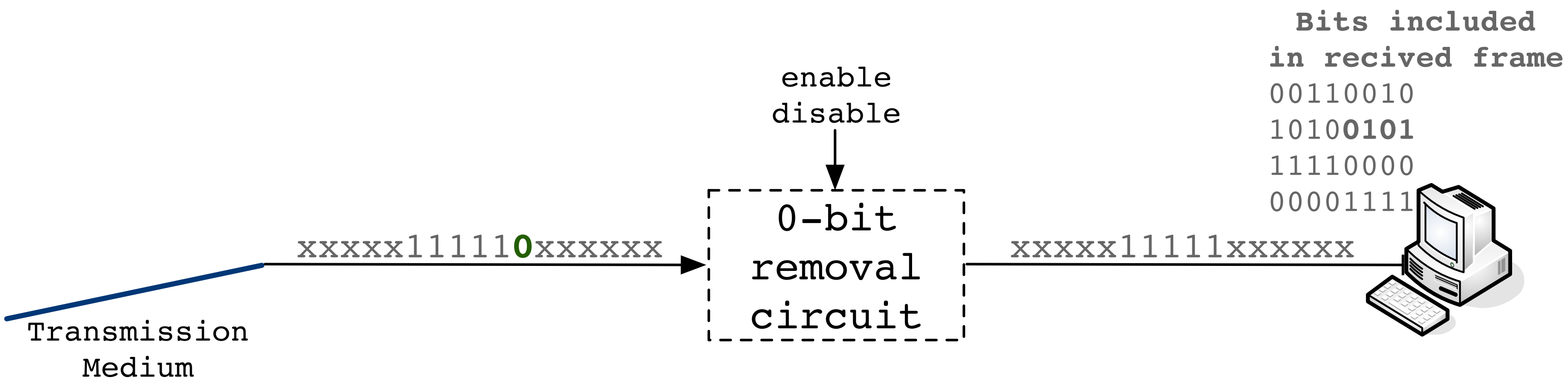


Bit stuffing in HDLC on the receiving side

Bit stuffing on the <u>receiving side</u>

- After receiving the initial flag, whenever the receiver receives 5 bits 1 and a one bit 0:
  - x x x x 1 1 1 1 1 **0** x x x x x x x x

- The receiver removes the bit 0 (The inserted bit):
  - x x x x 1 1 1 1 1 _ x x x x x x x x



Bit stuffing on the receiving side

- When 5 consecutive 1's are received 11111?
- **If next bit 0** : (01111100001010…)
  - It's a stuffed 0, so remove it and keep receiving the ensuing bits (0001010…) as data
- **If next bit 1** : (01111110001010…)
  - Further look at next bit b (0111111b)
    - If 0: End of frame marker (01111110)
    - If 1: Error has been introduced in the bitstream (01111111)