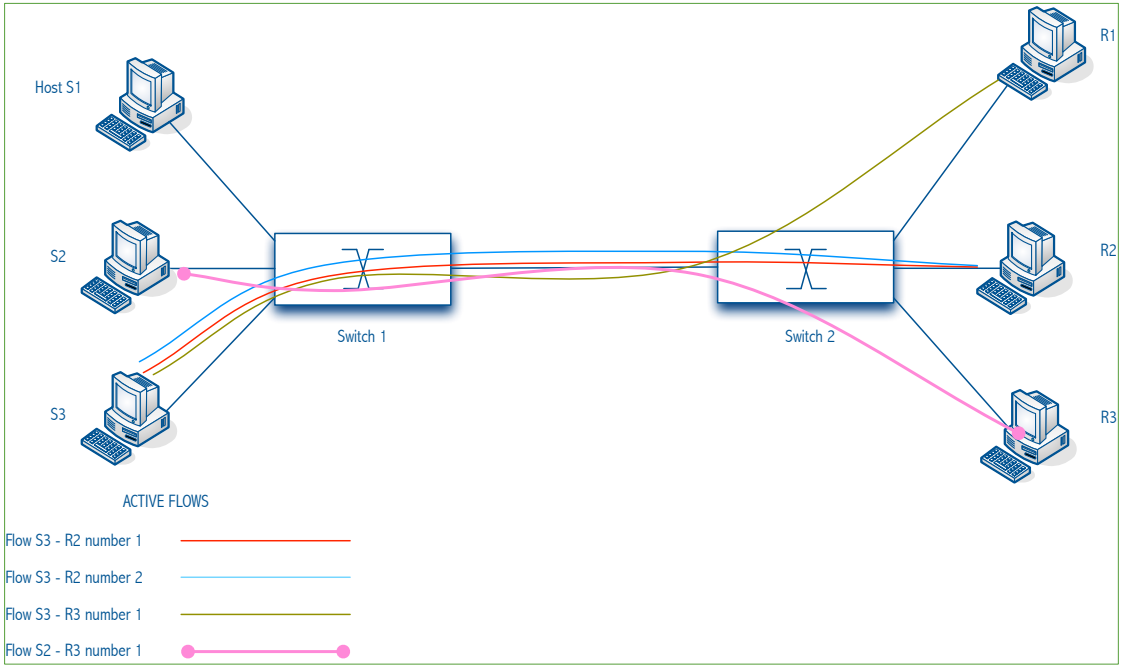
DIRECT COMMUNICATION LINKS

v 5.0 27th/April/2022

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

The scenario for Chapter 2

- **Statistical multiplexing**
  - Switching according to distribution of demand across all connected nodes
- **Directly connected nodes**
  - Host – Switch
  - Host – Host



Frame: Datalink PDU

PDU stands for *Protocol Data Unit*

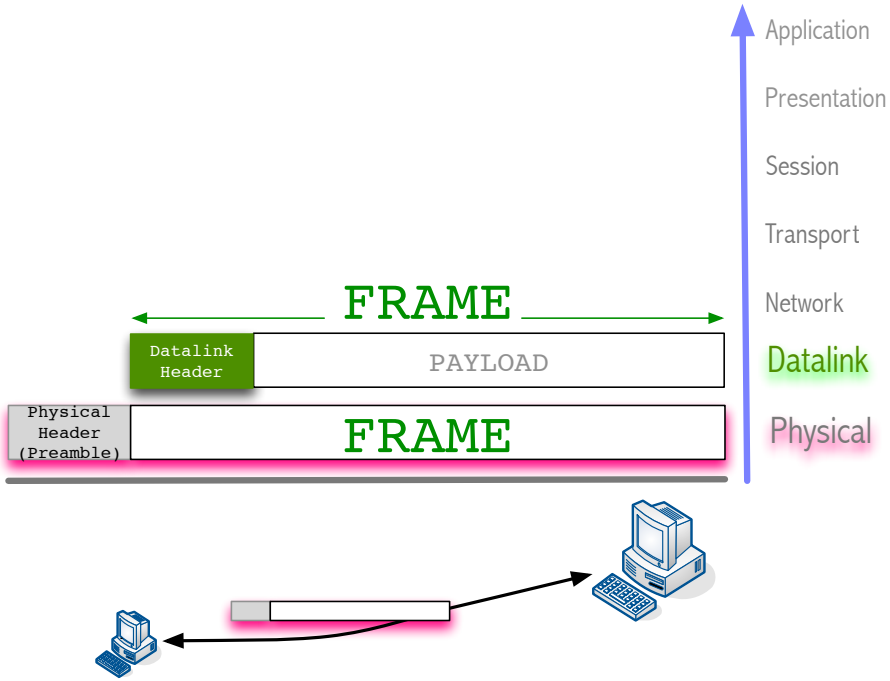
What is a Frame?

- Datalink protocols govern transmission onto direct links
- The PDU (Protocol Data Unit) of Datalink protocols is known as Frame
- Datalink FRAME =

  Datalink Header

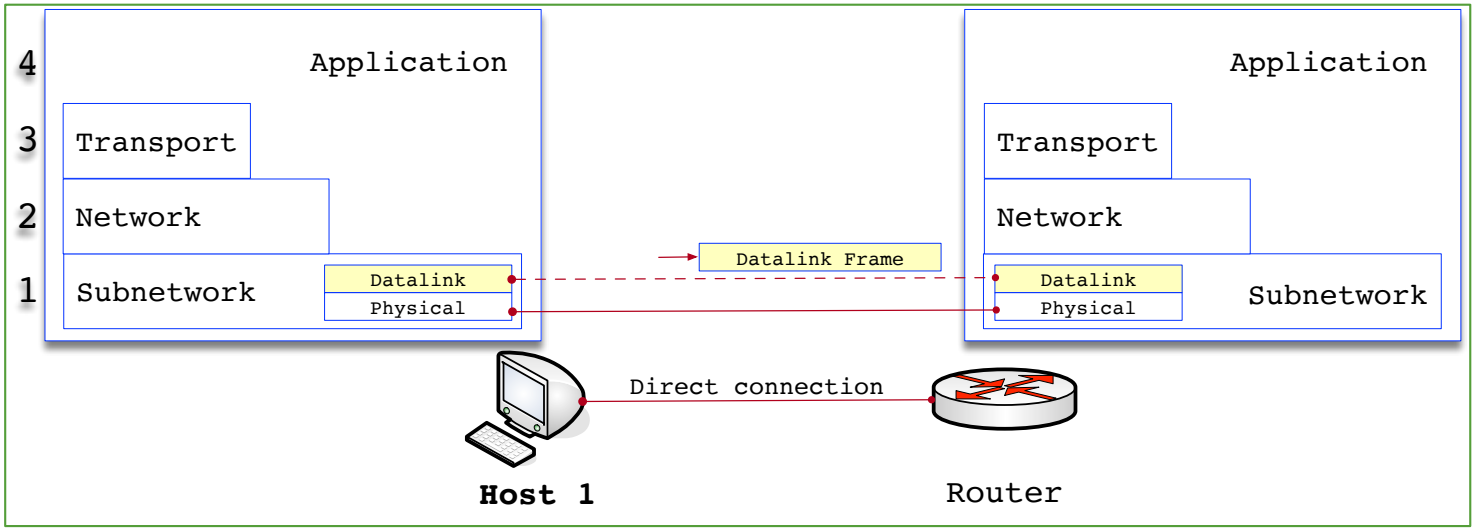
  +

  Upper layer payload



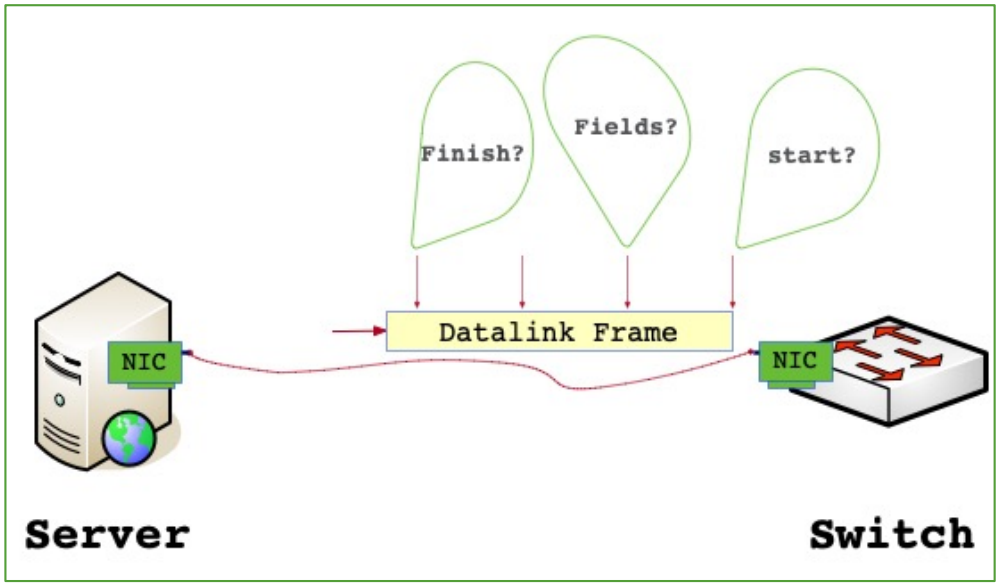
What is a direct connection?

- Example: Host1 is directly connected to the Router
- Frame:
  - Payload: Encapsulates an upper-layer PDU
  - Header contains
    - A mux key + Src host address + Dest host address + other fields



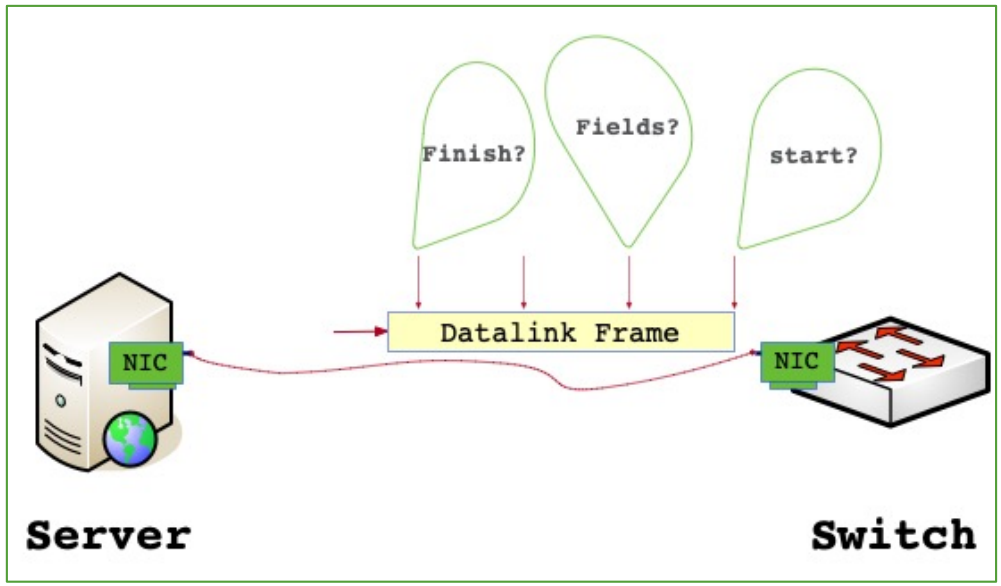
Detection of Frame's fields

- Example: 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's fields:
  - Where the frame begins and ends
  - Which are the frame's fields



Three strategies for delineating a frame

- 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

ASCII table

NAME

ascii - ASCII character set encoded in octal, decimal, and hexadecimal

DESCRIPTION

ASCII is the American Standard Code for Information Interchange. It is a 7-bit code. Many 8-bit codes (e.g., ISO 8859-1) contain ASCII as their lower half. The international counterpart of ASCII is known as ISO 646-IRV.

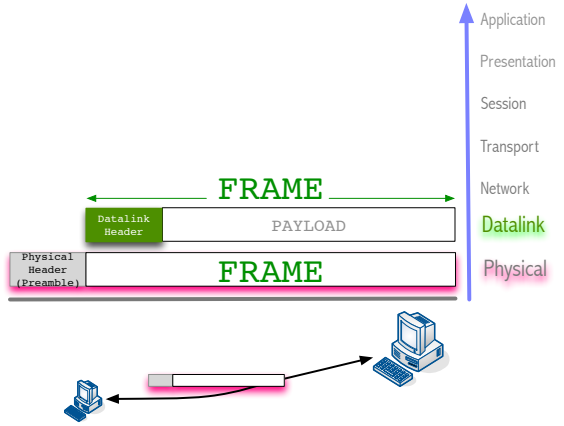
The following table contains the 128 ASCII characters.

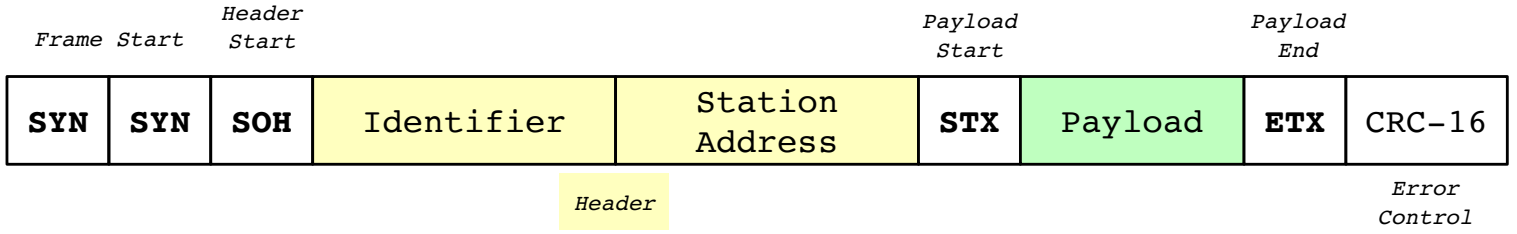
C program '\X' escapes are noted.

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0' (null character)	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
005	5	05	ENQ (enquiry)	105	69	45	E
006	6	06	ACK (acknowledge)	106	70	46	F
007	7	07	BEL '\a' (bell)	107	71	47	G
010	8	08	BS '\b' (backspace)	110	72	48	H
011	9	09	HT '\t' (horizontal tab)	111	73	49	I
012	10	0A	LF '\n' (new line)	112	74	4A	J
013	11	0B	VT '\v' (vertical tab)	113	75	4B	K
014	12	0C	FF '\f' (form feed)	114	76	4C	L
015	13	0D	CR '\r' (carriage ret)	115	77	4D	M
016	14	0E	SO (shift out)	116	78	4E	N
017	15	0F	SI (shift in)	117	79	4F	O
020	16	10	DLE (data link escape)	120	80	50	P
021	17	11	DC1 (device control 1)	121	81	51	Q
022	18	12	DC2 (device control 2)	122	82	52	R
023	19	13	DC3 (device control 3)	123	83	53	S
024	20	14	DC4 (device control 4)	124	84	54	T
025	21	15	NAK (negative ack.)	125	85	55	U
026	22	16	SYN (synchronous idle)	126	86	56	V
027	23	17	ETB (end of trans. blk)	127	87	57	W
030	24	18	CAN (cancel)	130	88	58	X
031	25	19	EM (end of medium)	131	89	59	Y
032	26	1A	SUB (substitute)	132	90	5A	Z
033	27	1B	ESC (escape)	133	91	5B	[
034	28	1C	FS (file separator)	134	92	5C	\ '\\'
035	29	1D	GS (group separator)	135	93	5D	]
036	30	1E	RS (record separator)	136	94	5E	^
037	31	1F	US (unit separator)	137	95	5F	_
040	32	20	SPACE	140	96	60	`
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27	'	147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	-	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x

BiSync protocol (a.k.a. BSC)

- 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





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

- 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", actually its 8 bits are *payload* data

Transparency in the BiSync protocol

- Byte-stuffing
  - 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 is to be sent as a data byte?
  - 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

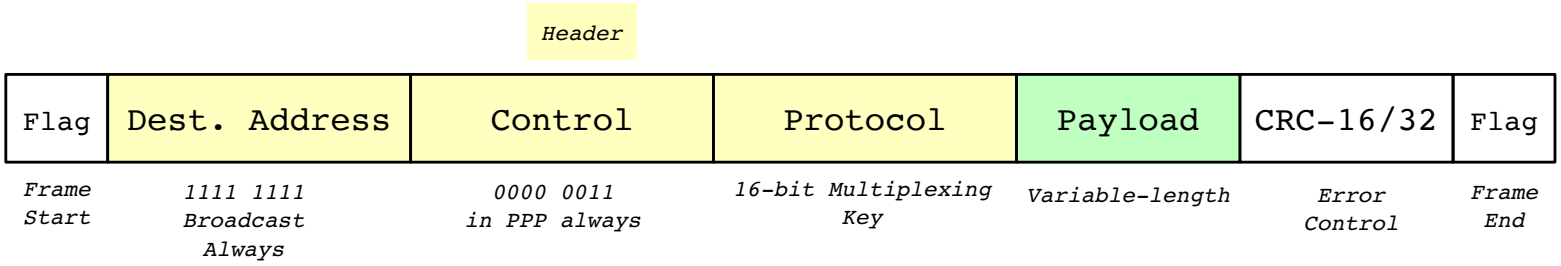
- What if **[DLE]** itself is to be sent as a data byte?
  - Same as in the C language: Include an escaping DLE character that escapes the special meaning of the next character: **[DLE][DLE]**

- 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]

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`

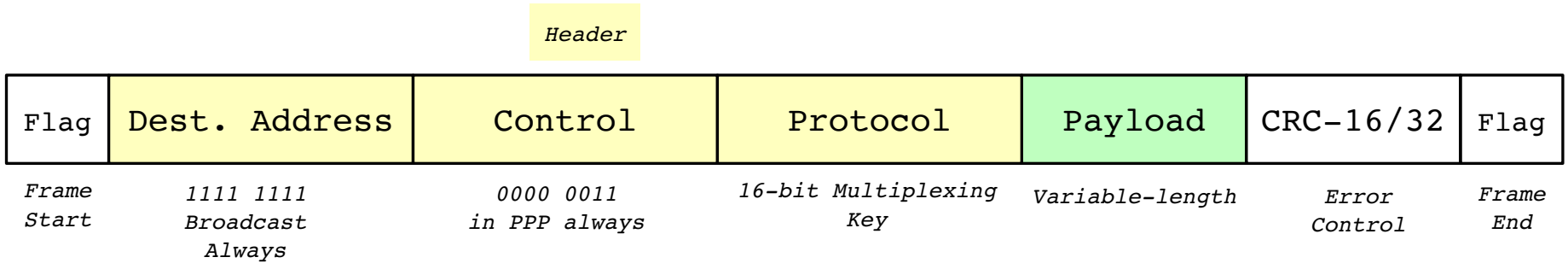
  0 1 1 1 1 1 1 0
- Protocol: A multiplexing key (Example: IP / IPX)
- Payload: The data transported, max size negotiated (MTU = 1500 bytes)
- CRC16 or CRC-32 for error detection



Generic PPP frame

PPP

- 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

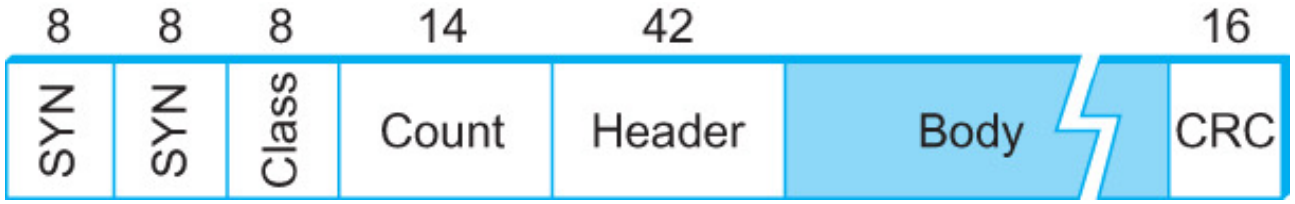


Generic PPP frame

DDCMP

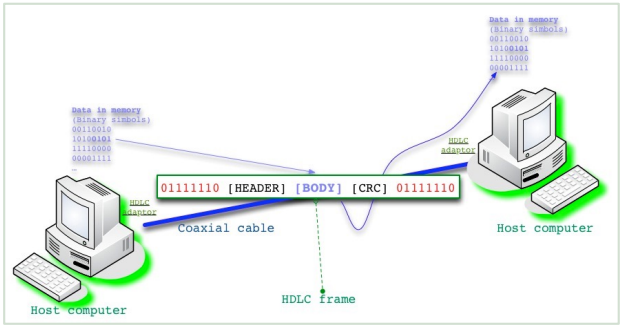
Byte-counting approach

- **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
  - *Payload can be made of any number of **bits, not necessarily an 8-bit multiple***
- Beginning and Ending Sequence (Sentinel is the FLAG character)

```
FLAG = 0 1 1 1 1 1 1 0
```

Header

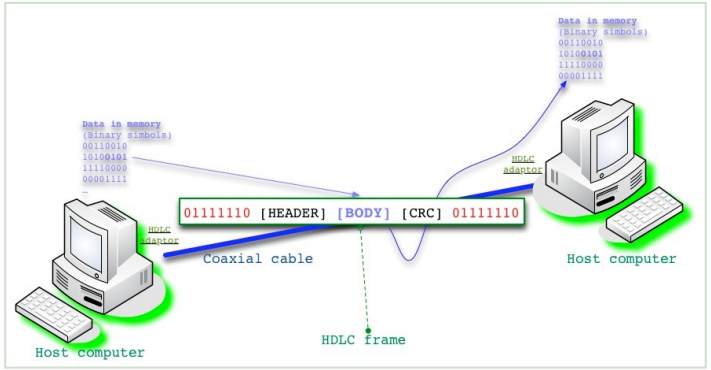
Flag	Address	Control	Information field	CRC-16/32	Flag
Frame Start	*8/16 bits*	*8/16 bits:* *· Information* *· Supervisory* *· Unnumbered*	*0 to N bits*	*Error Control*	*Frame End*

Generic HDLC frame

Transparency in HDLC

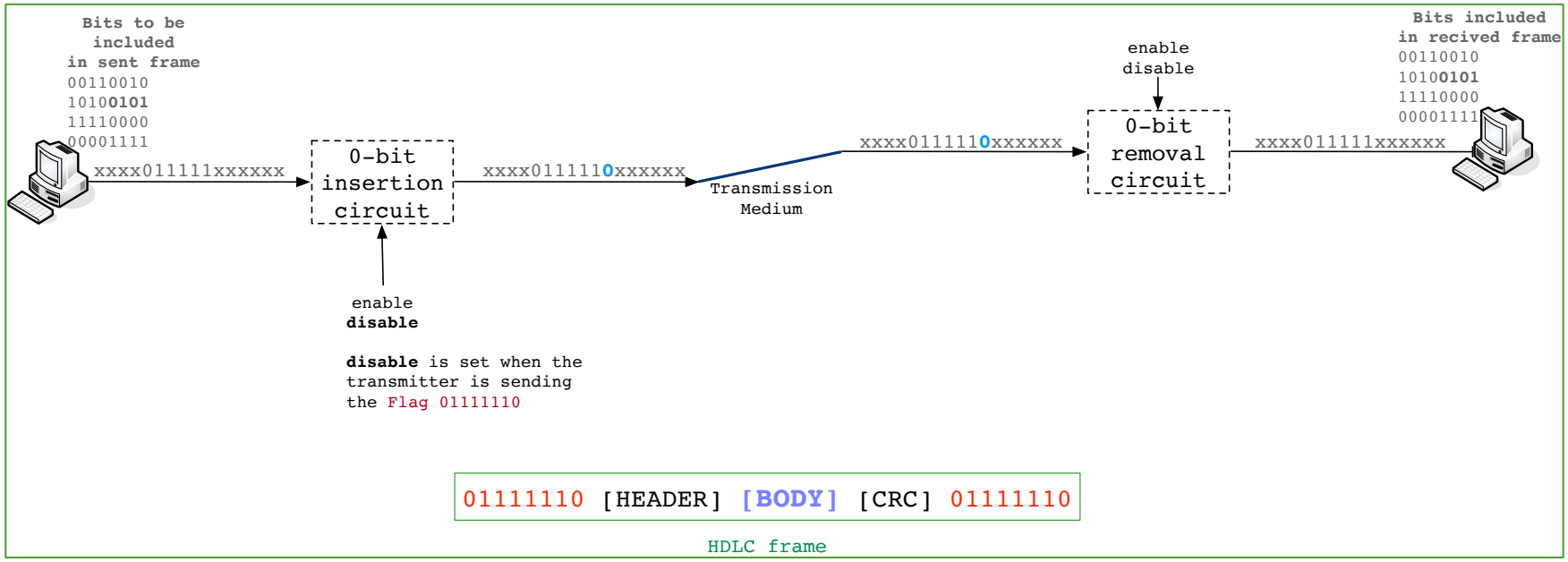
- Problem with the Flag sentinel
  - What if the FLAG 0 1 1 1 1 1 1 0 is contained anywhere in the frame after the initial Flag?
  - The receiver would take this bit sequence to be a terminating Flag

- Solution: Bit Stuffing (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

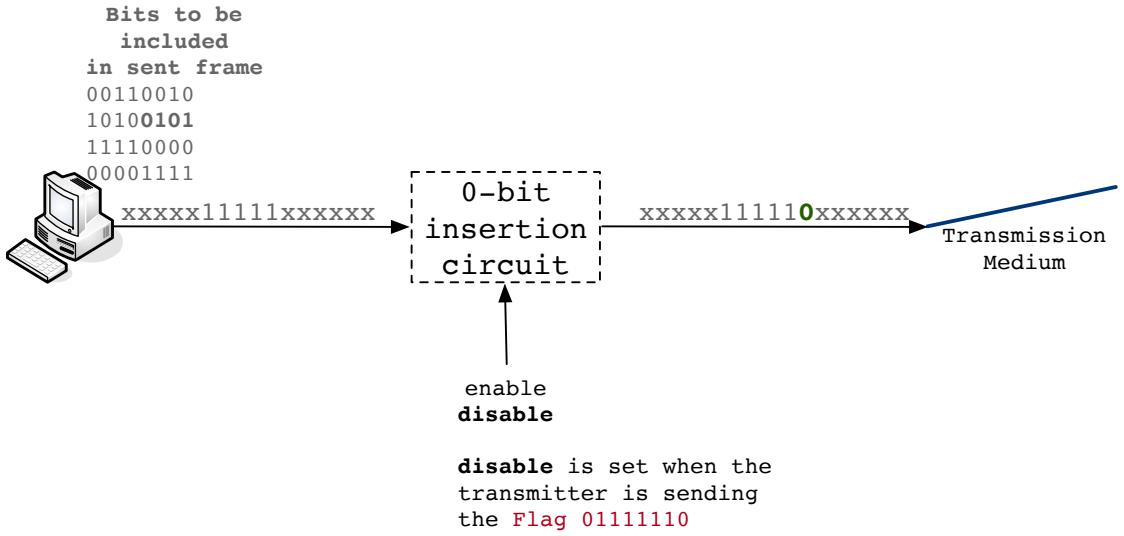
- **At the sender:** Whenever the sender observes a block of 5 bits 1 after the frame start, the sender inserts a bit 0 before transmitting the next bit:
  - x x x 0 1 1 1 1 1 **0** x x x x x x x x
- **At the receiver:** The receiver removes the inserted bit 0 whenever it observes the block of 5 bits 1 followed by a bit 0 (*The stuffed bit*)
  - x x x 0 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 sending side

- Any time five consecutive 1's appear in the frame (Between the start and end of frame)
  - x x x 0 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

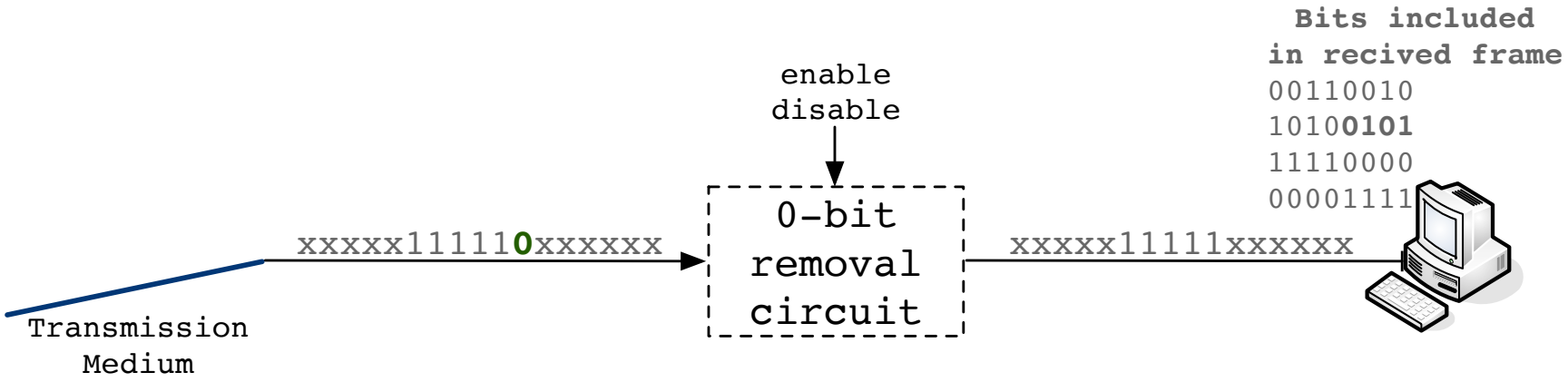


Bit stuffing in HDLC on the receiving side

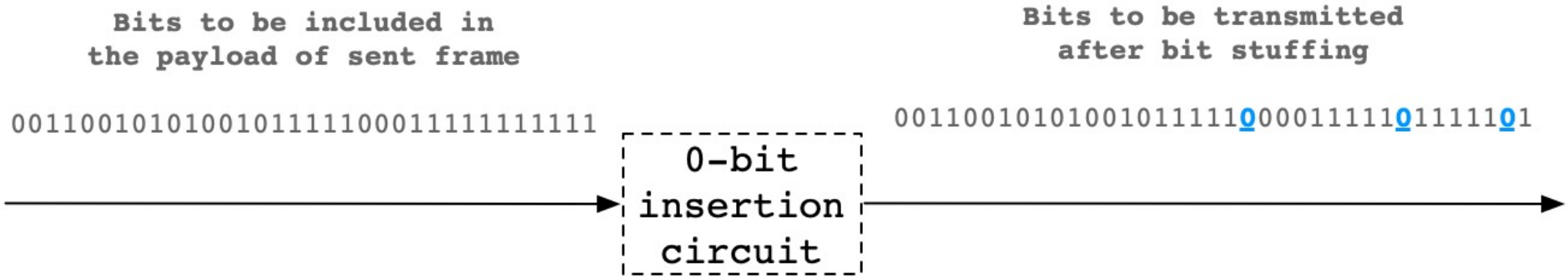
Bit stuffing on the receiving side

- 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

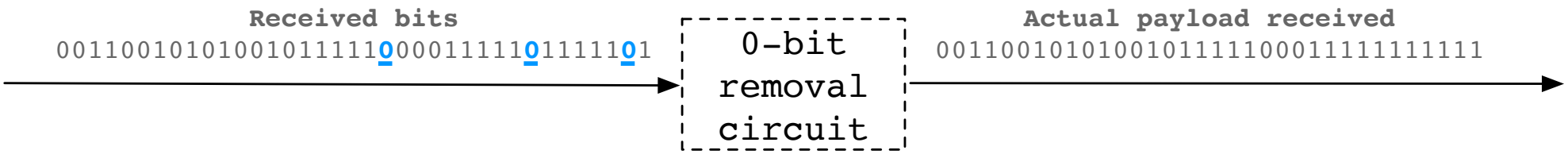
- The receiver removes the bit 0 (The inserted bit):
  - x x x x 1 1 1 1 1 _ x x x x x x x x



Bit stuffing, example



Bit stuffing, example



Bit stuffing on the receiving side

- When a string of 5 consecutive 1's is received:
  011111?
- If next bit 0 : (0111110 0001010…)
  - It's a stuffed 0, so remove it and keep receiving the ensuing bits (0001010…) as the effective payload of the received frame
- If next bit 1 : (0111111 0 0001010…)
  - In turn, look at next bit b (0111111b)
    - If 0: End of frame marker (01111110)
    - If 1: Error has been introduced in the bitstream (01111111)