Bit and Byte Stuffing

Synchronous versus Asynchronous Transmissions

- There exists a hierarchy of synchronization tasks:
  - *Bit level* : recognizing the start and end of each bit
  - *Character or byte level* : recognizing the start and end of each character (or small unit of data)
  - *Block or message level* : recognize the start and end of each large unit of data (in networks this is a **frame**).

Synchronous versus Asynchronous Transmissions [Halsall]

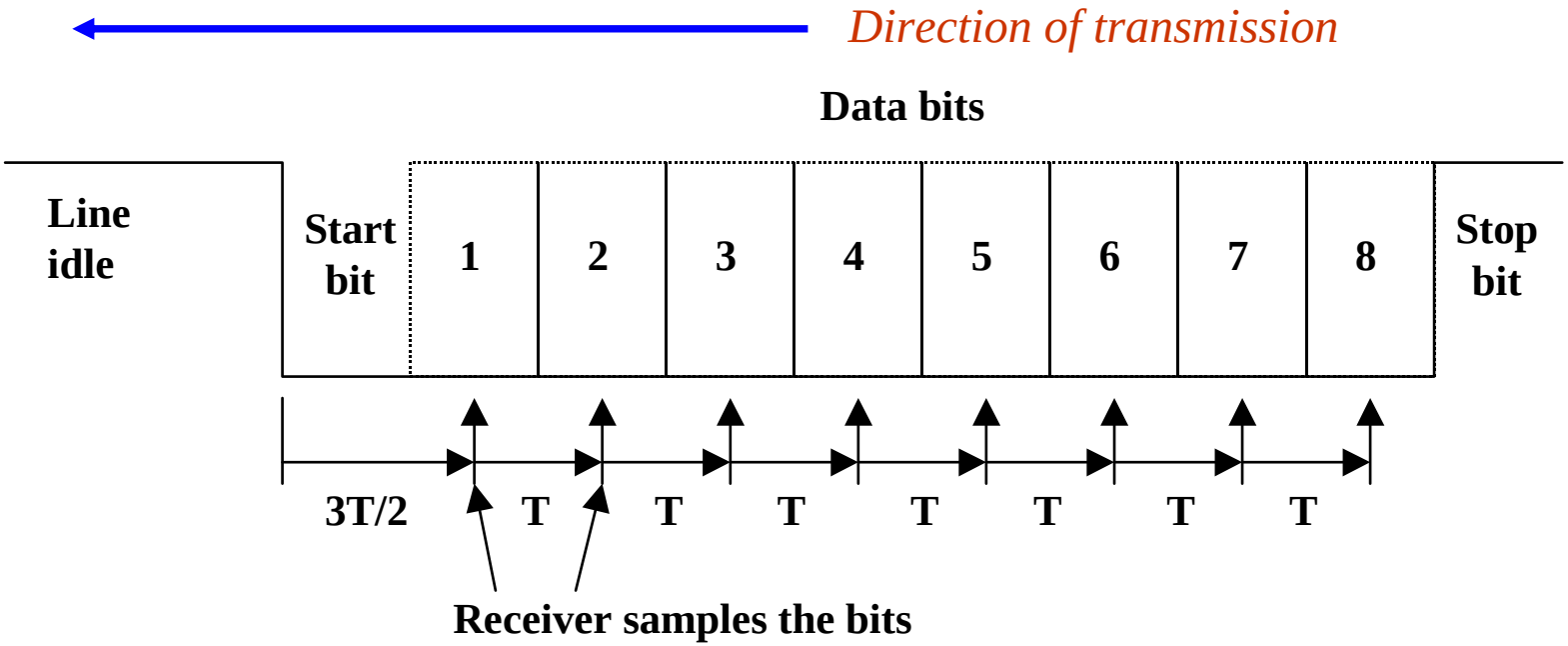
*A fundamental requirement of digital data communications is that the **receiver** knows the starting time and the duration of each bit.*

Asynchronous transmission :: each character (or byte) is treated independently for clock (bit) and character (byte) synchronization purposes and the receiver resynchronizes at the start of each character received.

Synchronous transmission :: the complete **frame** is transmitted as a contiguous string of bits and the receiver endeavors to keep in synchronism with the incoming bit stream for the duration of the frame.

Synchronization in Asynchronous Transmissions

Transmitted at random intervals (e.g., keyboard)



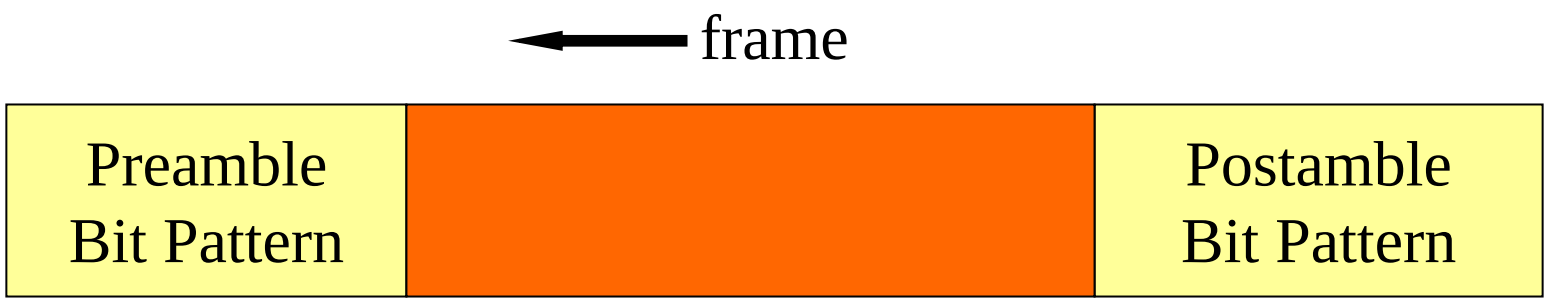
Synchronous Transmissions

- More efficient, i.e., less overhead
- Blocks of characters transmitted without start and stop codes
- The transmitted stream is suitably encoded so the receiver can stay *in "synch"* by:
  - Using a separate clock line
  - Embedding clocking information into data (e.g. biphase coding).

Frame Identification Methods [Tanenbaum]

1. Byte counts
2. Starting/ending bytes [byte stuffing]
3. Starting/ending flags [bit stuffing]
4. Using physical layer coding violations (i.e., invalid physical codes)

The contents of each frame are *encapsulated* between a pair of reserved characters or bytes for frame synchronization.



BISYNC Frame Format

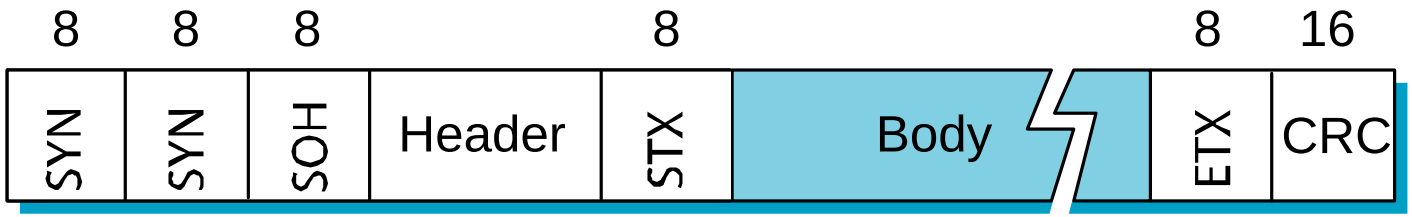


Figure 2.9

P&D slide

PPP Frame Format

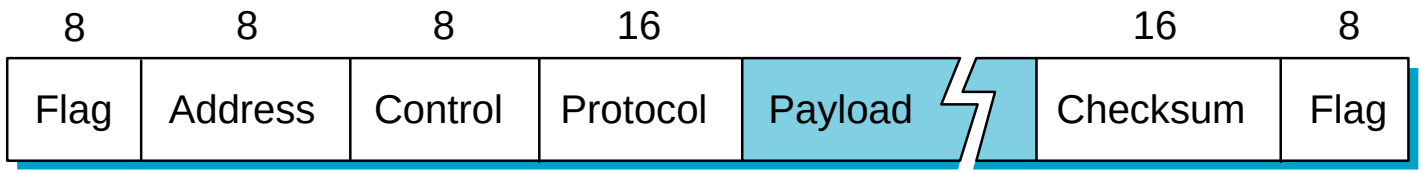


Figure 2.10

P&D slide

DDCMP Frame Format

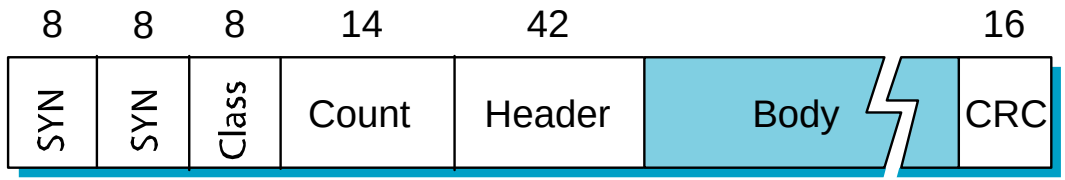


Figure 2.11

P&D slide

HDLC Frame Format

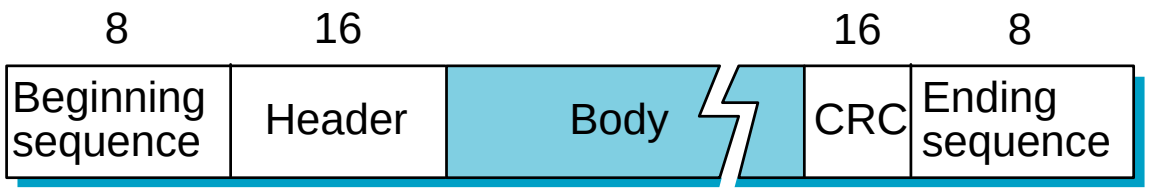


Figure 2.12

P&D slide

Byte Stuffing
[HDLC Example]

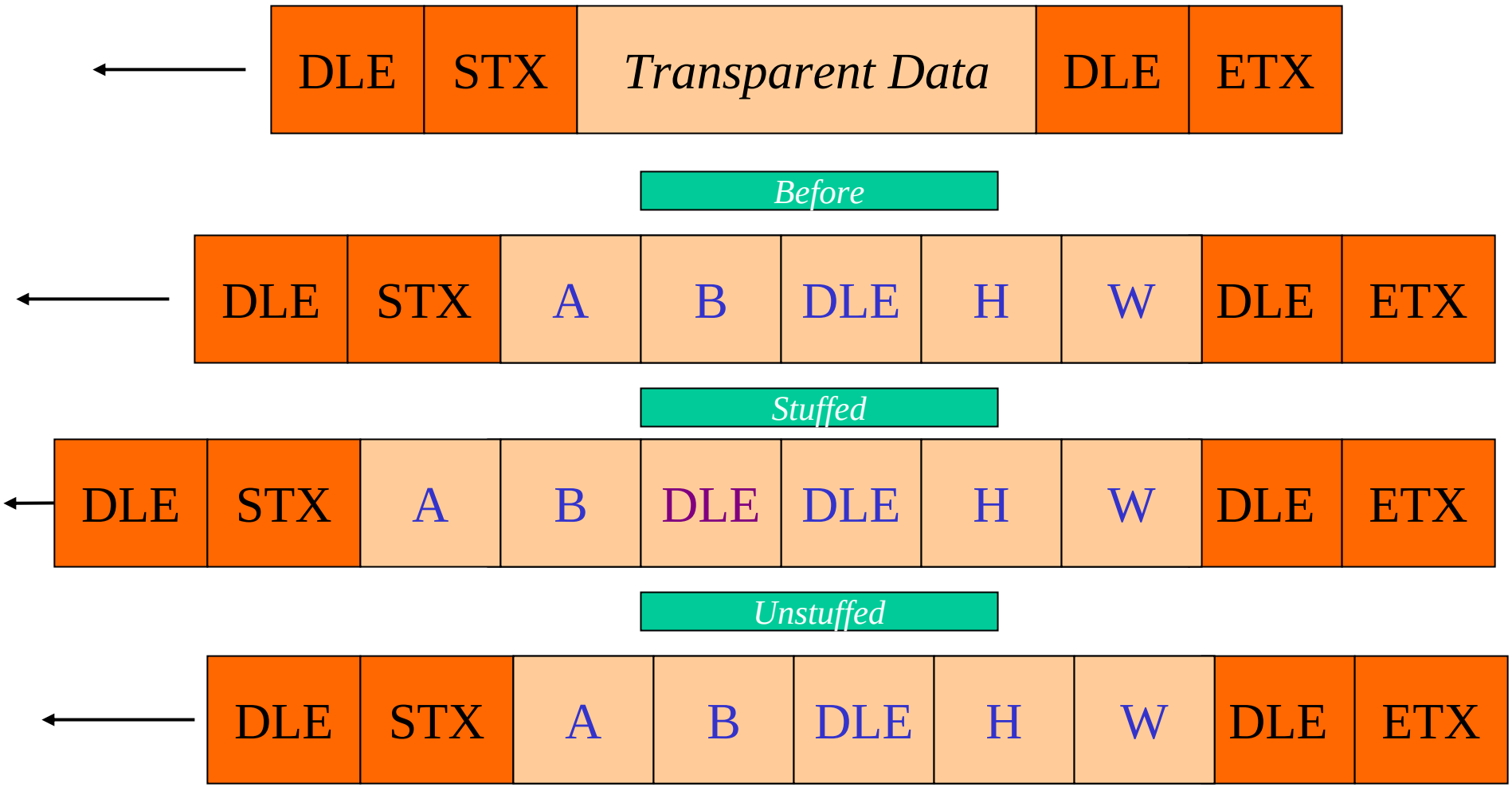
- *Also referred to as character stuffing.*
- ASCII characters are used as framing delimiters (e.g. DLE STX and DLE ETX)
- The problem occurs when these character patterns occur within the "transparent" data.

Solution: sender stuffs an **extra DLE** into the data stream just before each occurrence of an "accidental" DLE in the data stream.

The data link layer on the receiving end unstuffs the DLE before giving the data to the network layer.

Byte Stuffing

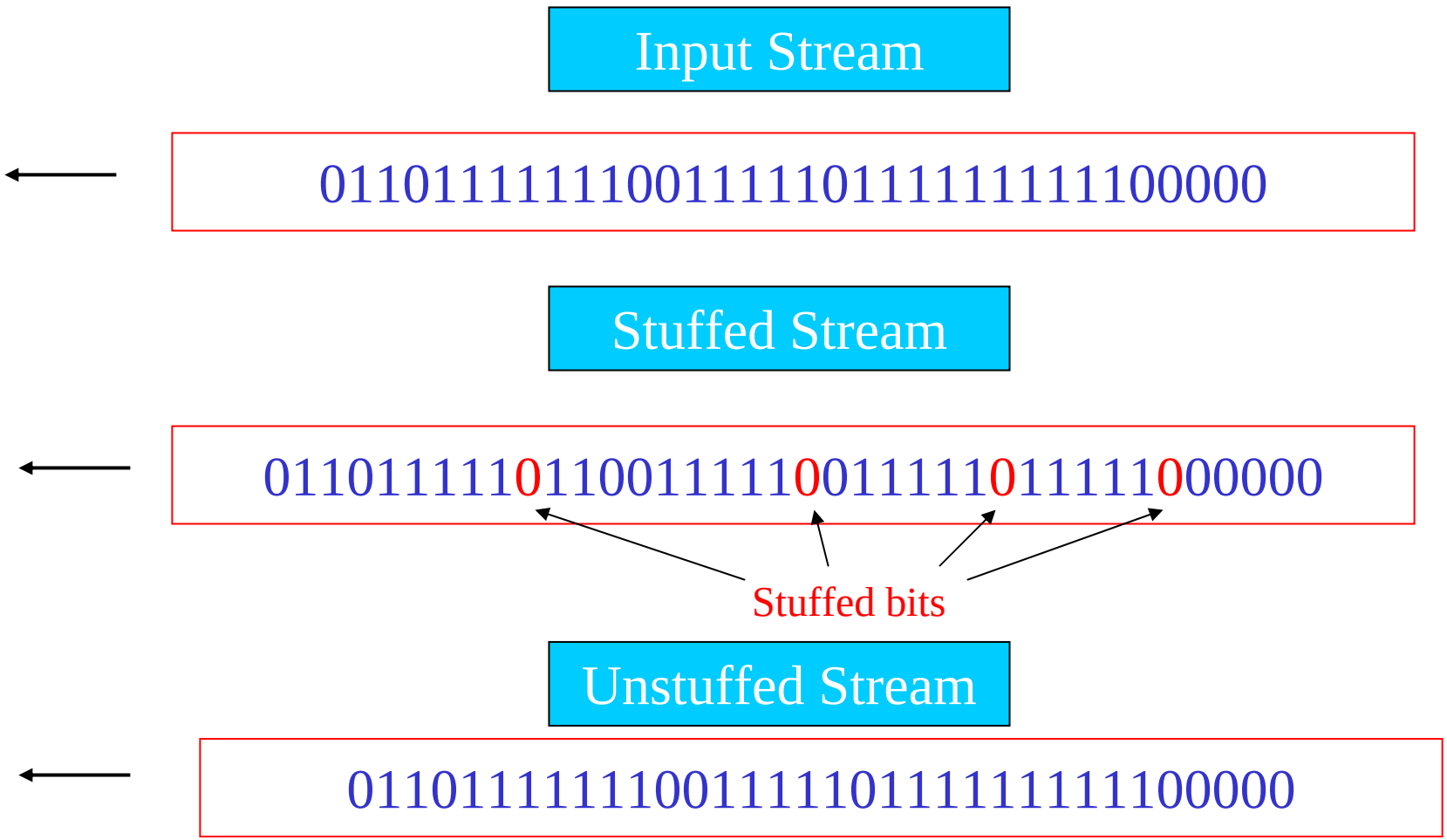
[HDLC Example]



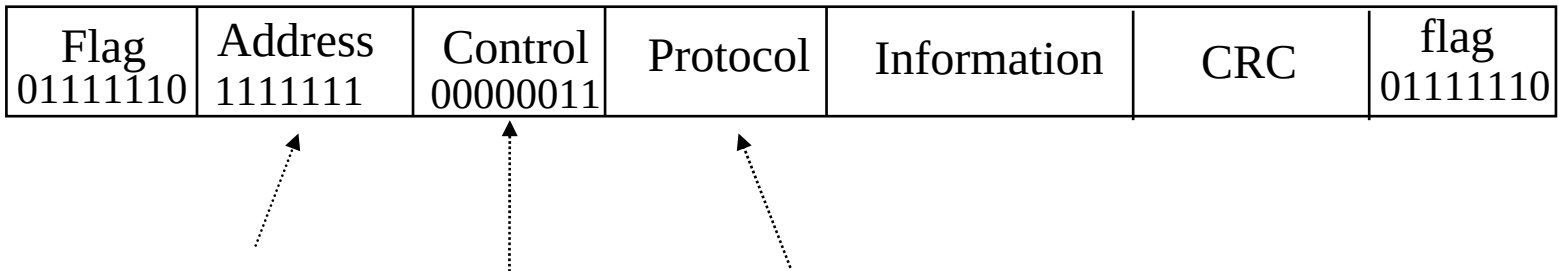
Bit Stuffing

- Each frame begins and ends with a special bit pattern called a flag byte [01111110]. {Note this is **7E** in hex}
- Whenever sender data link layer encounters *five consecutive ones* in the data stream, it automatically stuffs a 0 bit into the outgoing stream.
- When the receiver sees *five consecutive incoming ones followed by a 0 bit*, it automatically destuffs the 0 bit before sending the data to the network layer.

Bit Stuffing



PPP (Point-to-Point Protocol) Frame Format



PPP Byte Stuffing