CS3600 — Systems and Networks

NORTHEASTERN UNIVERSITY

Lecture 18: Data-link layer

Prof. Alan Mislove (amislove@ccs.neu.edu)

Slides used with permissions from Edward W. Knightly, T. S. Eugene Ng, Ion Stoica, Hui Zhang

Bit Stream Encoding

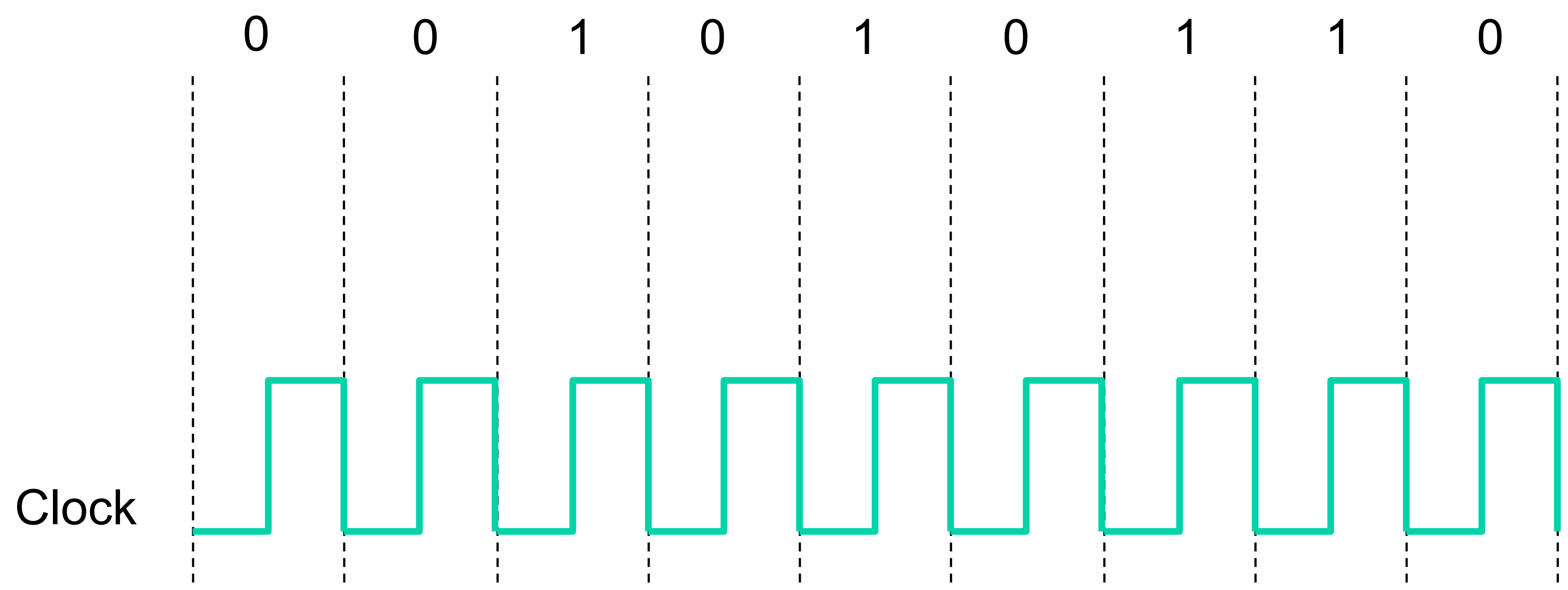
- Specify how bits are represented in the analog signal
  - This service is provided by the physical layer
- Challenges:
  - Efficiency: ideally, bit rate is maximized
  - Robust: avoid de-synchronization between sender and receiver when there is a large sequence of 1's or 0's

Assumptions

- We use two discrete signals, high and low, to encode 1 and 0
- The transmission is synchronous, i.e., there is a clock used to sample the signal
- If the amplitude and duration of the signals is large enough, the receiver can do a reasonable job of looking at the distorted signal and estimating what was sent.

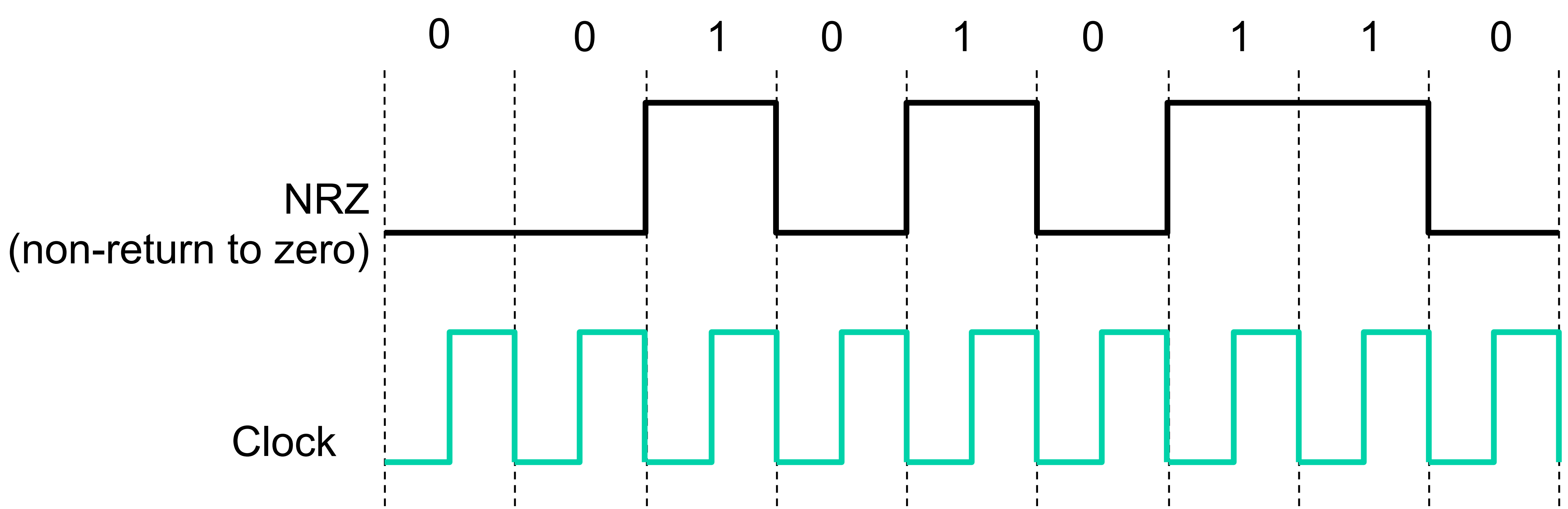
Non-Return to Zero (NRZ)

- 1 → high signal; 0 → low signal



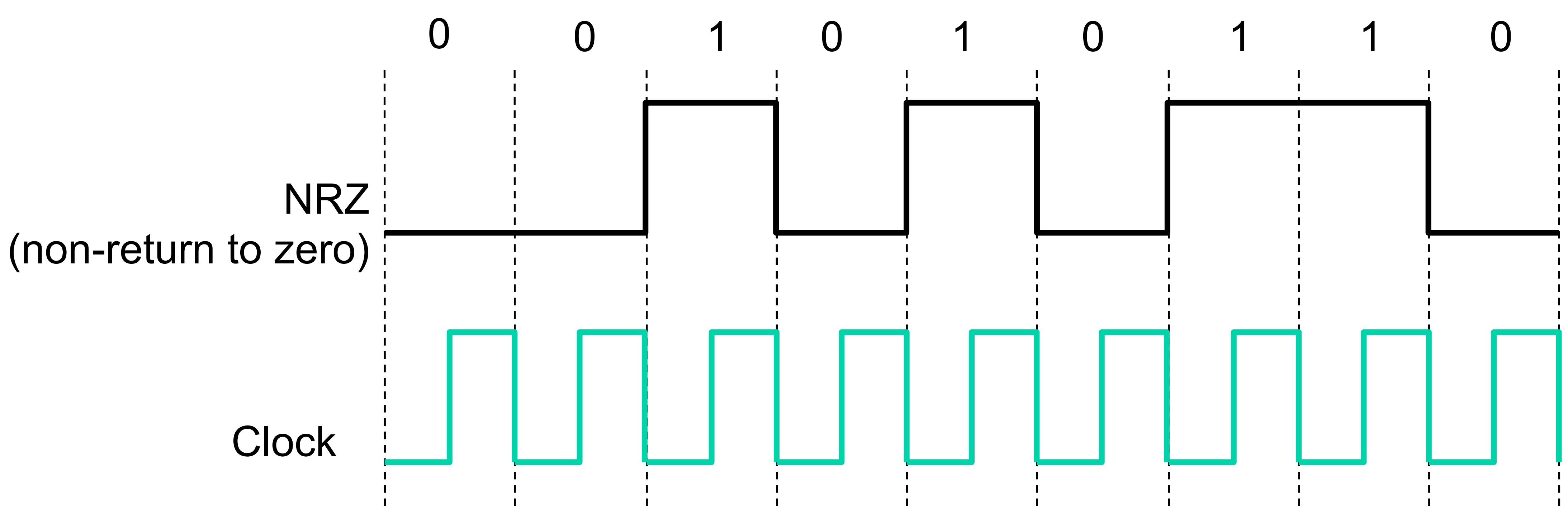
Non-Return to Zero (NRZ)

- 1 → high signal; 0 → low signal



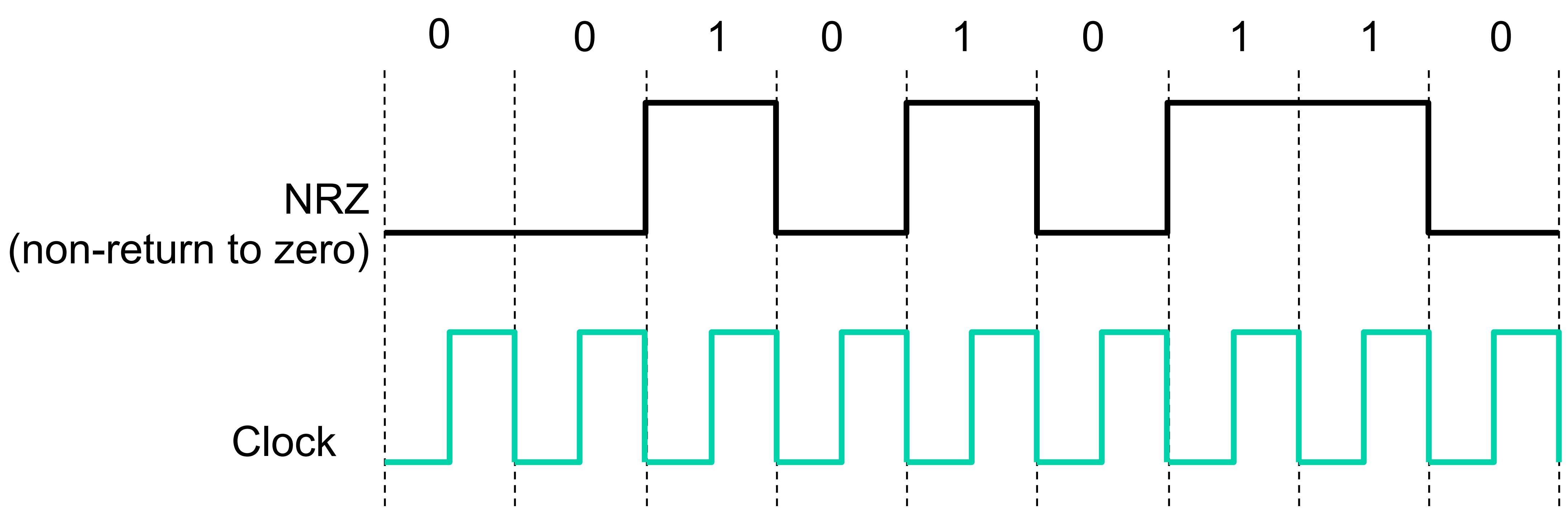
Non-Return to Zero (NRZ)

- 1 → high signal; 0 → low signal
- Disadvantages: when there is a long sequence of 1's or 0's
  - Sensitive to clock skew, i.e., difficult to do clock recovery



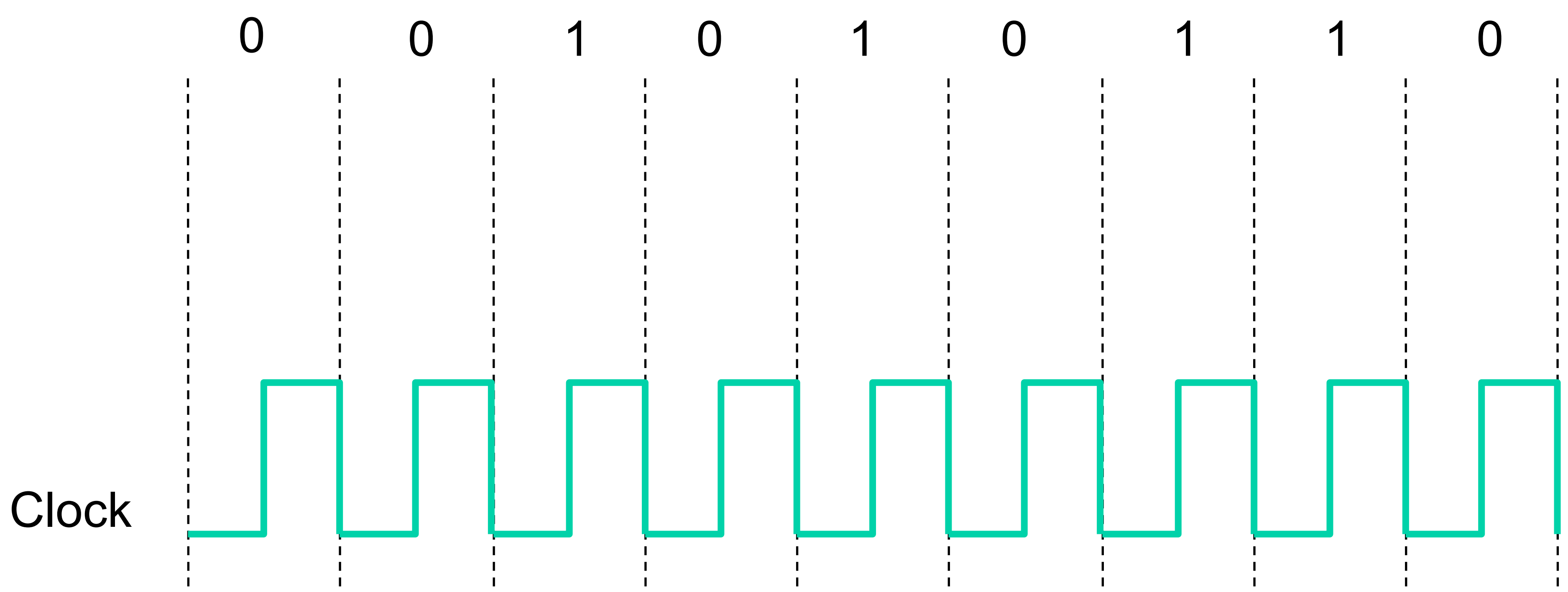
Non-Return to Zero (NRZ)

- 1 → high signal; 0 → low signal
- Disadvantages: when there is a long sequence of 1's or 0's
  - Sensitive to clock skew, i.e., difficult to do clock recovery
  - Also, sensitive to baseline wander



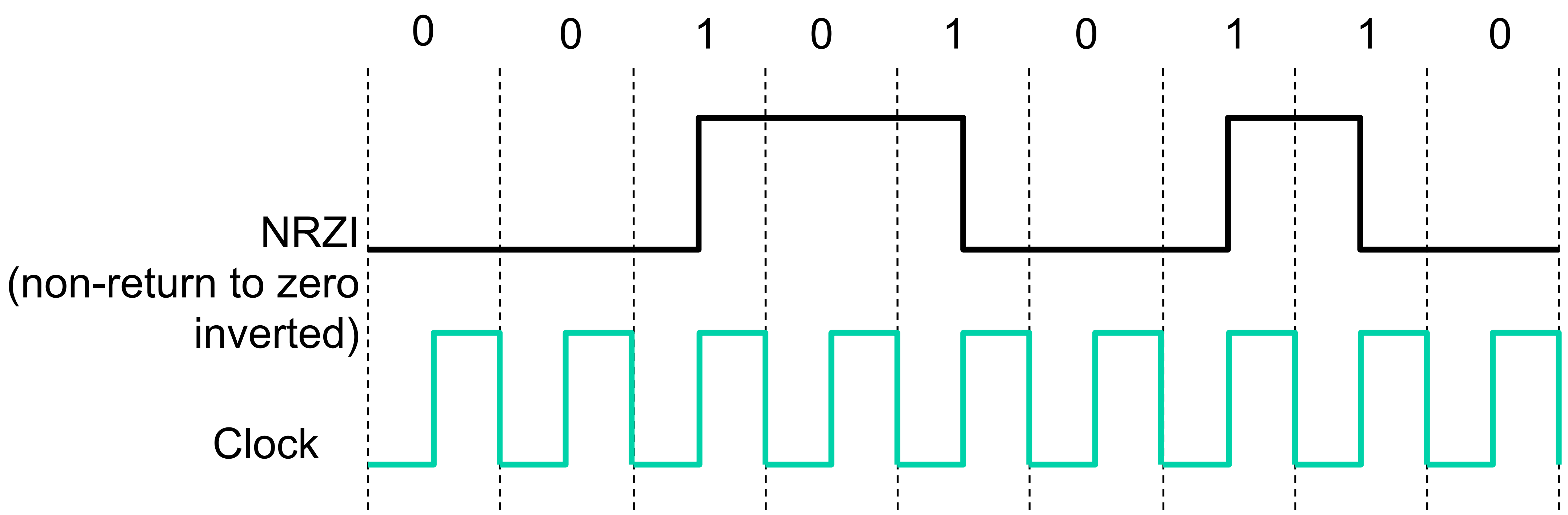
Non-Return to Zero Inverted (NRZI)

- 1 → make transition; 0 → stay at the same level
- Solve previous problems for long sequences of 1's, but not for 0's

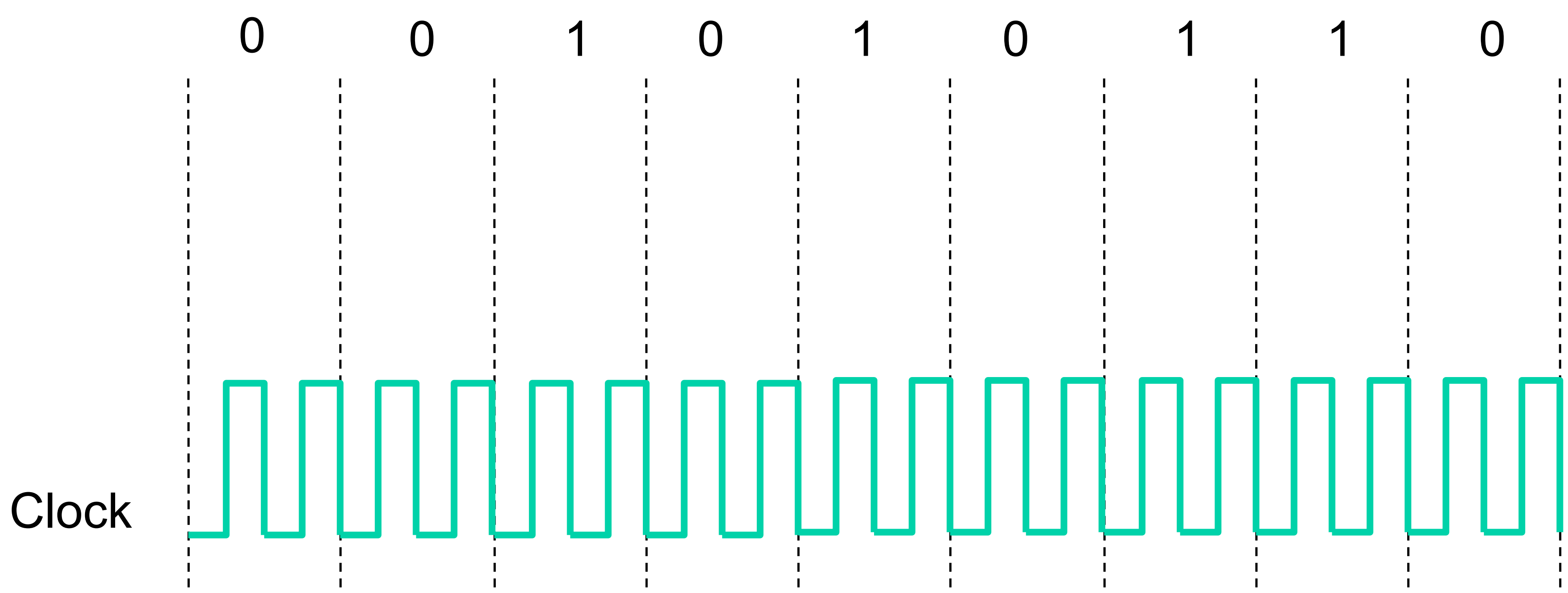


Non-Return to Zero Inverted (NRZI)

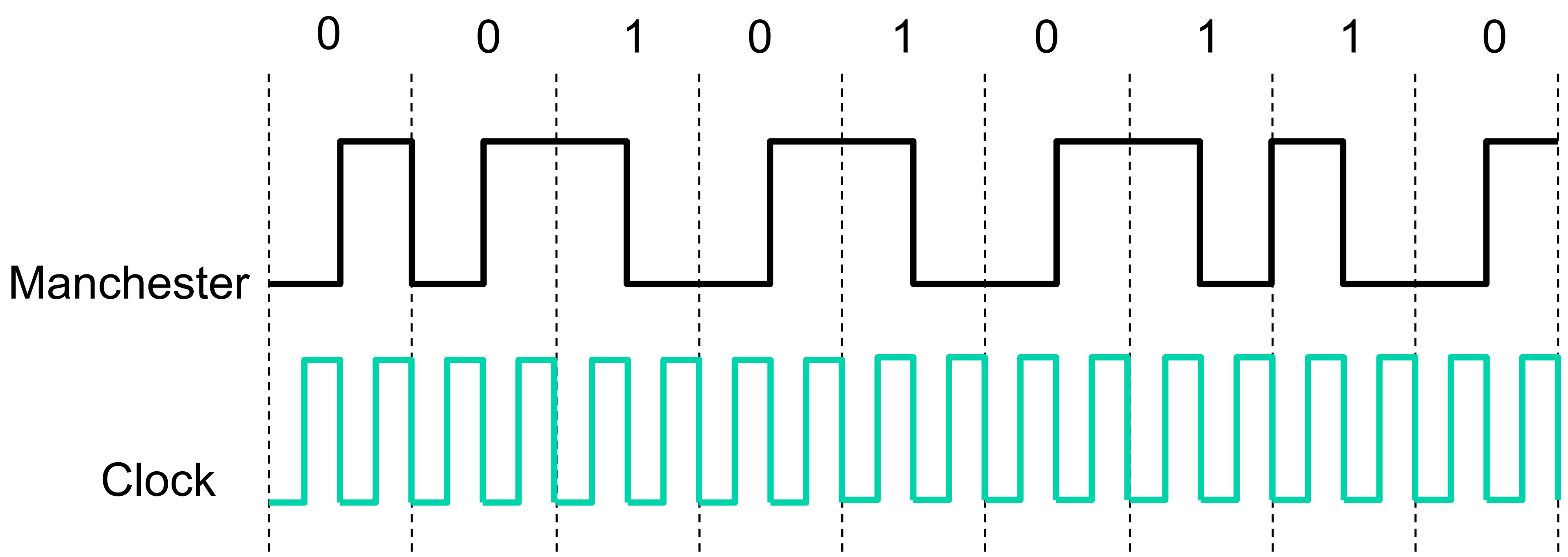
- 1 → make transition; 0 → stay at the same level
- Solve previous problems for long sequences of 1's, but not for 0's



Manchester

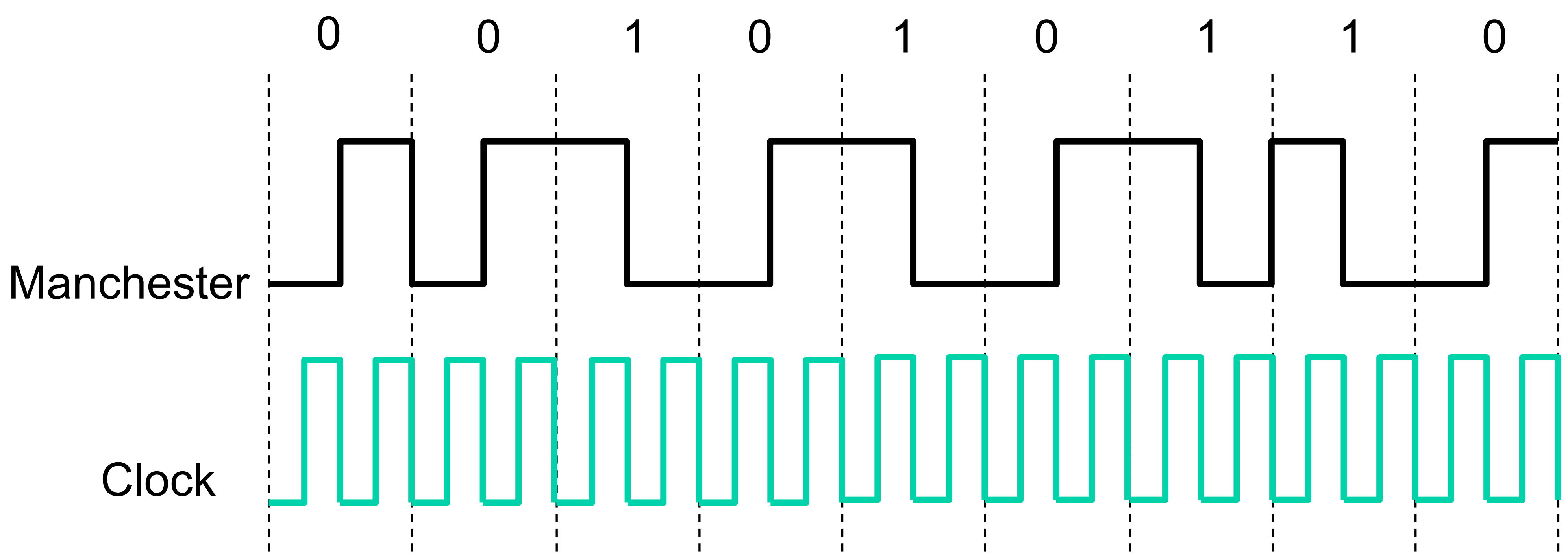


Manchester



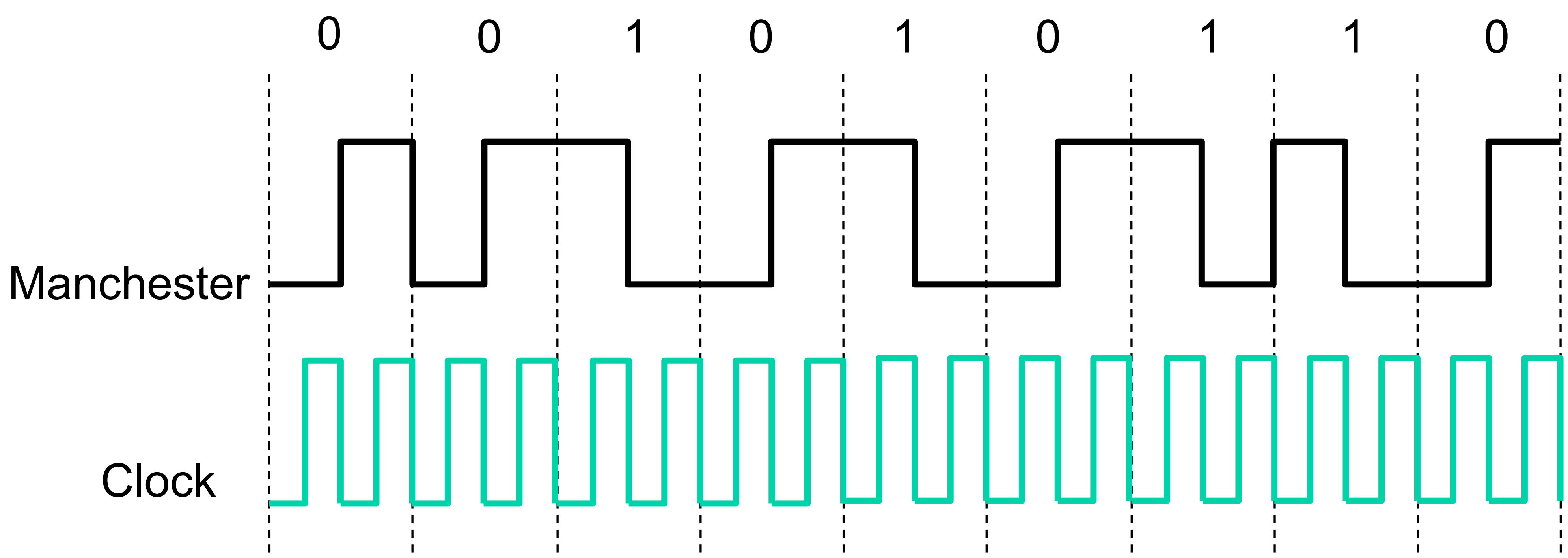
Manchester

- 1 → high-to-low transition; 0 → low-to-high transition



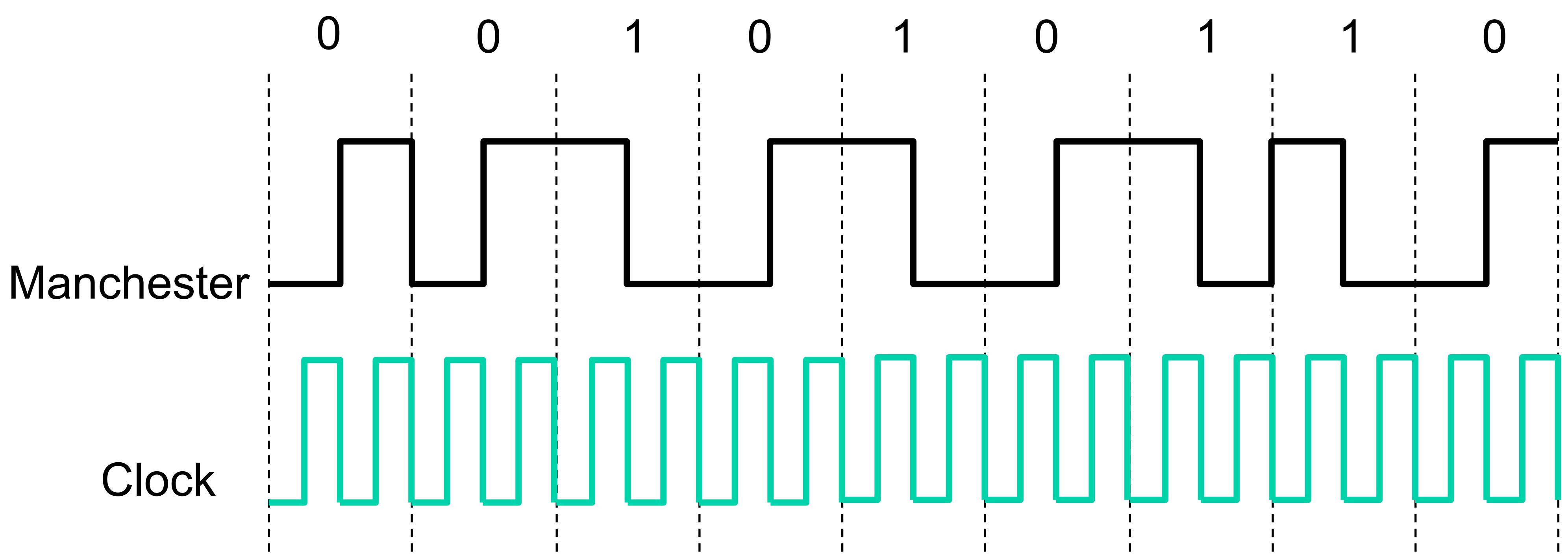
Manchester

- 1 → high-to-low transition; 0 → low-to-high transition
- Addresses clock recovery problems
- Disadvantage: signal transition rate doubled



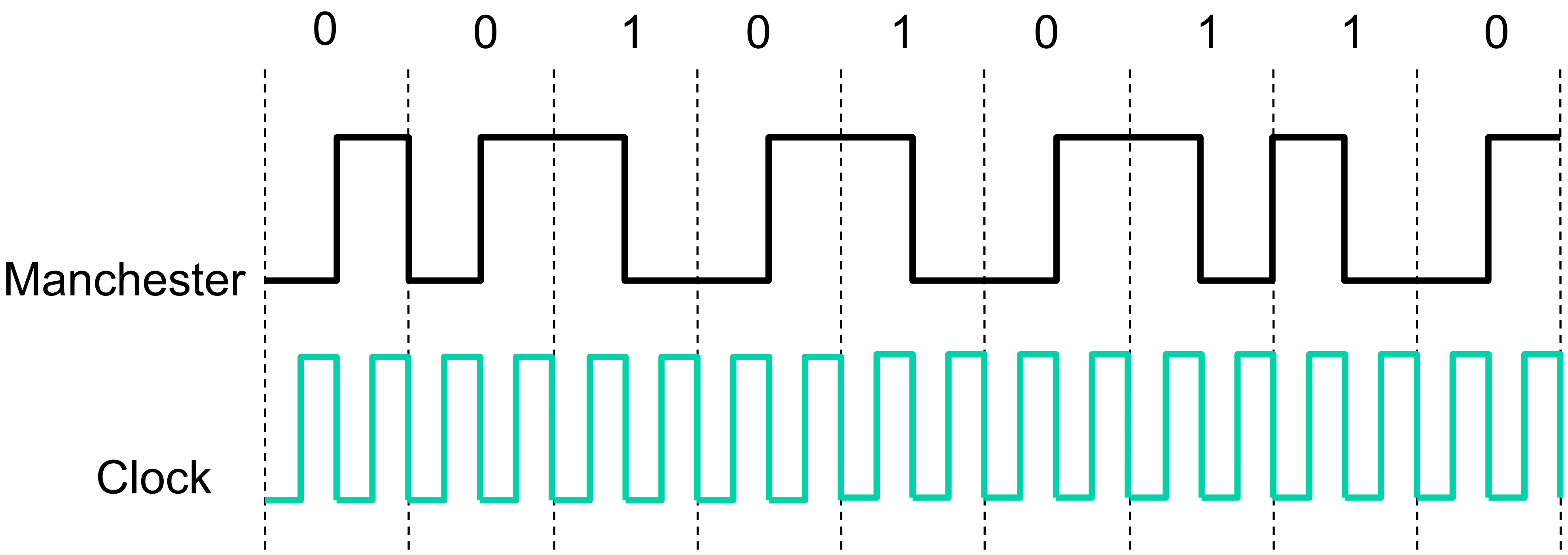
Manchester

- 1 → high-to-low transition; 0 → low-to-high transition
- Addresses clock recovery problems
- Disadvantage: signal transition rate doubled
  - I.e. useful data rate on same physical medium halved



Manchester

- 1 → high-to-low transition; 0 → low-to-high transition
- Addresses clock recovery problems
- Disadvantage: signal transition rate doubled
  - I.e. useful data rate on same physical medium halved
  - Efficiency of 50%



4-bit/5-bit (100Mb/s Ethernet)

- Goal: address inefficiency of Manchester encoding, while avoiding long periods of low signals
- Solution:
  - Use 5 bits to encode every sequence of four bits such that no 5 bit code has more than one leading 0 and two trailing 0's
  - Use NRZI to encode the 5 bit codes
  - Efficiency is 80%

4-bit/5-bit (100Mb/s Ethernet)

- Goal: address inefficiency of Manchester encoding, while avoiding long periods of low signals
- Solution:
  - Use 5 bits to encode every sequence of four bits such that no 5 bit code has more than one leading 0 and two trailing 0’s
  - Use NRZI to encode the 5 bit codes
  - Efficiency is 80%

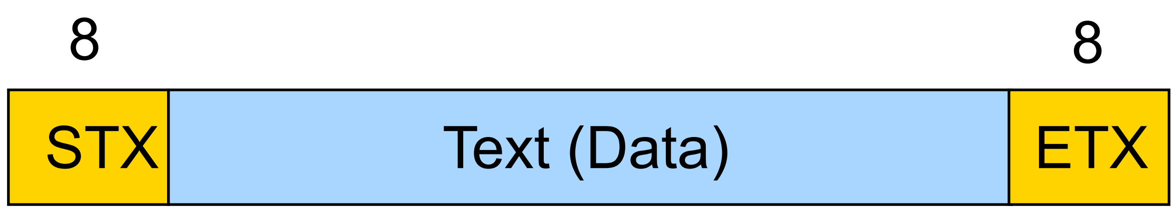
4-bit	5-bit
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111

4-bit	5-bit
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

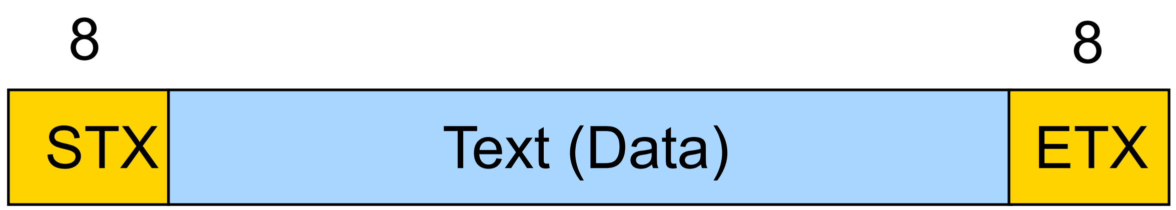
Framing

- Specify how blocks of data are transmitted between two nodes connected on the same physical media
  - This service is provided by the data link layer
- Challenges
  - Decide when a frame starts/ends
  - If use special delimiters, differentiate between the true frame delimiters and delimiters appearing in the payload data

Byte-Oriented Protocols: Sentinel Approach

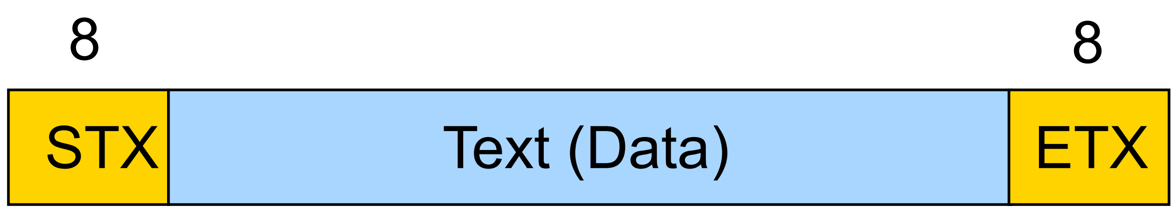


Byte-Oriented Protocols: Sentinel Approach



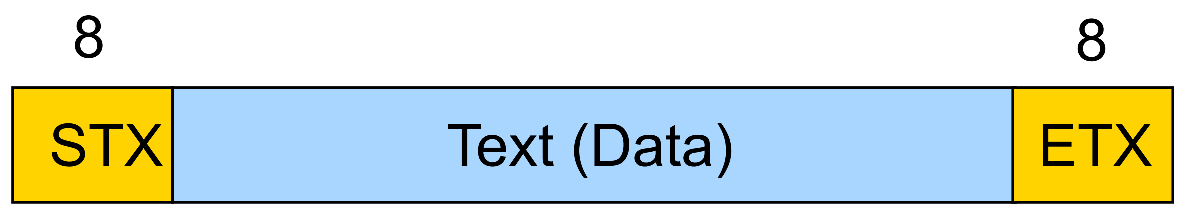
- STX – start of text

Byte-Oriented Protocols: Sentinel Approach



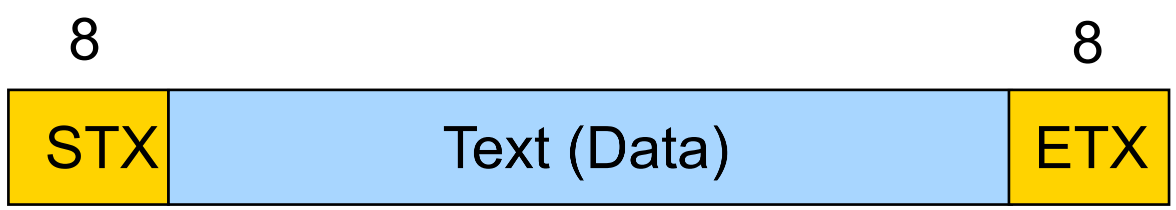
- STX – start of text
- ETX – end of text

Byte-Oriented Protocols: Sentinel Approach



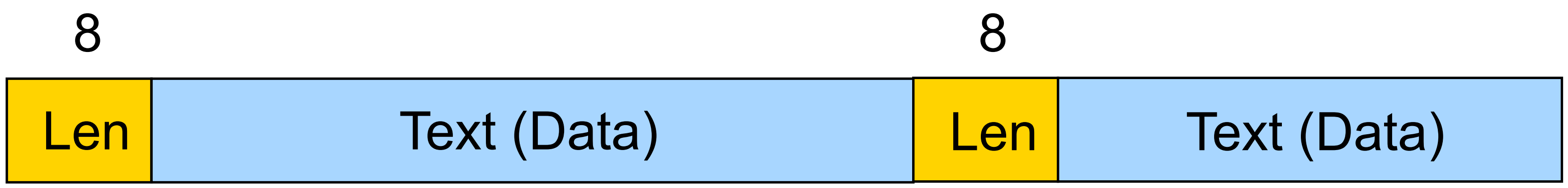
- STX – start of text
- ETX – end of text
- Problem: what if ETX appears in the data portion of the frame?

Byte-Oriented Protocols: Sentinel Approach



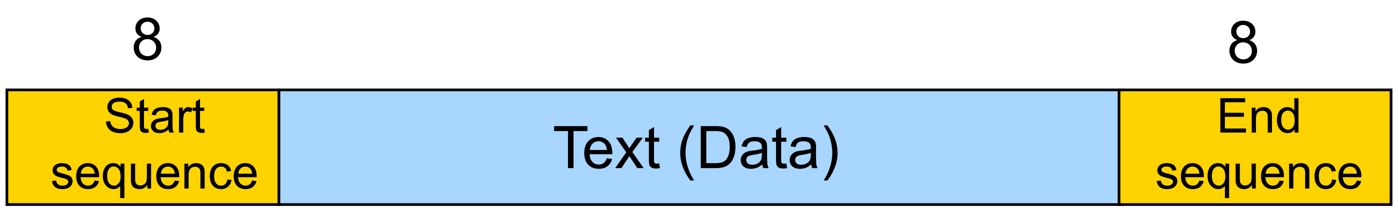
- STX – start of text
- ETX – end of text
- Problem: what if ETX appears in the data portion of the frame?
- Solution
  - If ETX appears in the data, introduce a special character DLE (Data Link Escape) before it
  - If DLE appears in the text, introduce another DLE character before it
  - Like in C programming, “Say \”Hello\””, (\ is the escape character)

Byte-Oriented Protocols: Byte Counting Approach



- Sender: insert the length of the data (in bytes) at the beginning of the frame, i.e., in the frame header
- Receiver: extract this length and decrement it every time a byte is read. When this counter becomes zero, we are done

Bit-Oriented Protocols



- Both start and end sequence can be the same
  - E.g., 01111110 in HDLC (High-level Data Link Protocol)
- Sender: in data portion inserts a 0 after five consecutive 1s
  - “Bit stuffing”
- Receiver: when it sees five 1s makes decision on the next two bits
  - If next bit 0 (this is a stuffed bit), remove it
  - If next bit 1, look at the next bit
    - If 0 this is end-of-frame (receiver has seen 01111110)
    - If 1 this is an error, discard the frame (receiver has seen 01111111)

Error detection

- How to determine if errors (via noise) were introduced?
- Could send 2 copies of data
  - Has poor efficiency
  - Poor protection against errors
- Will discuss three approaches
  - Two-dimensional parity
  - Checksum
  - CRCs

Two-dimensional parity

- Add extra bits to keep number of 1s even
  - Add parity bits and parity bytes

0101001	1	Parity bit for each 7 bits
1101001	0	
1011110	1	
0001110	1	
0110100	1	
1011111	0	
1111011	0	Parity byte for each frame

Two-dimensional parity

- Add extra bits to keep number of 1s even
  - Add parity bits and parity bytes

0101001	1	Parity bit for each 7 bits
1101001	0	
1011110	1	
0001110	1	
0110100	1	
1011111	0	
1111011	0	Parity byte for each frame

- Can detect all 1-, 2-, and 3- bit errors!
  - But with at least 14% overhead

Checksums

- Simple: add up bytes of messages, include the sum
  - Hence *check-sum*
- View data as series of unsigned 16-bit integers
  - Use ones-complement arithmetic
- Much lower overhead (16 bits/frame)
- But, not resilient to errors
  - Why? Error which increments/decrements any two ints
- Used in UDP, TCP, and IP, though

CRCs

- Cyclic redundancy check (CRC)
- Addresses limitations of prior approaches
  - Uses field theory
- Much better performance
  - Fixed overhead per frame
  - Only 1 in 2^{32} chance of missed error with 32-bit CRC
- Details in the book, if you're curious