Congestion Control (Part 3)

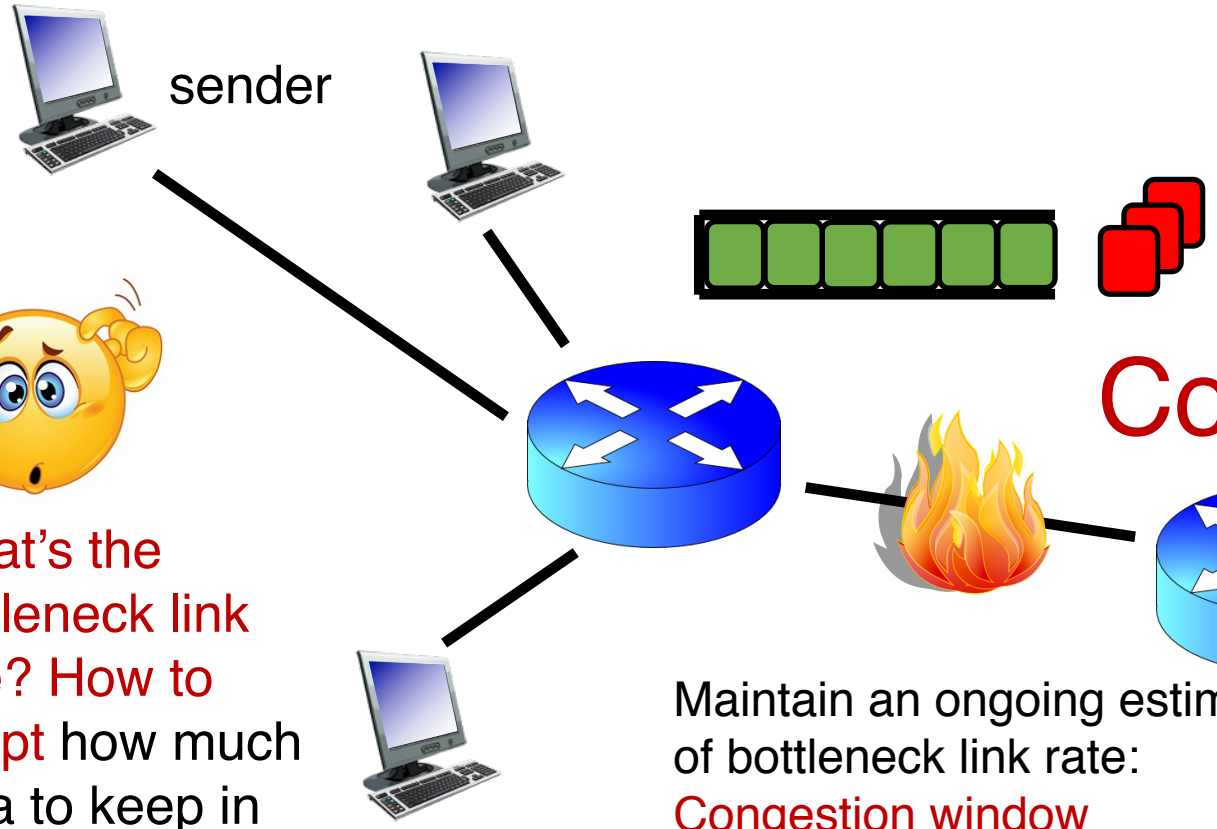
Lecture 17

http://www.cs.rutgers.edu/~sn624/352-S22

Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

Congestion Control

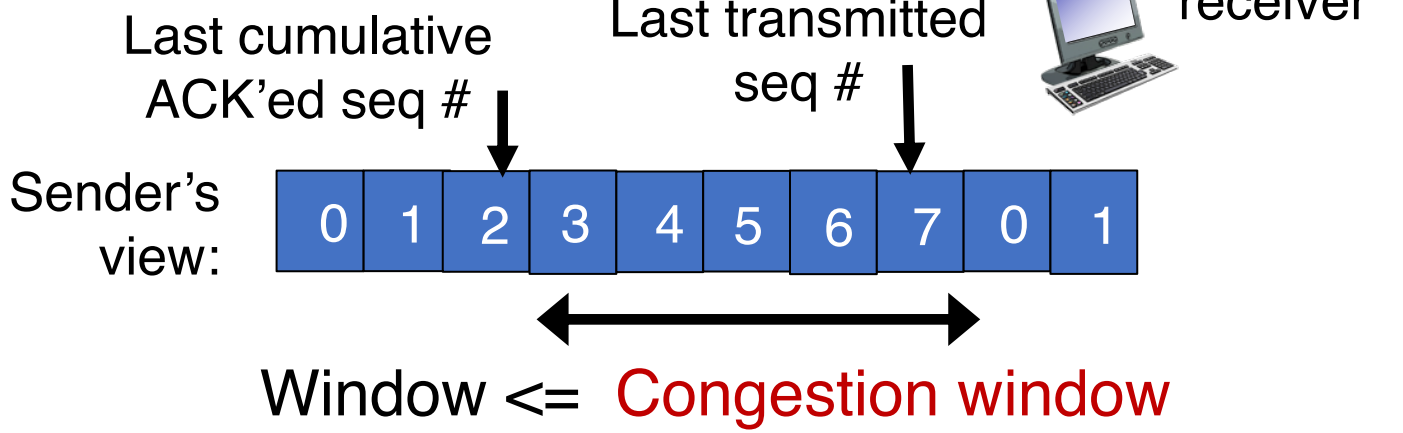


What's the bottleneck link rate? How to adapt how much data to keep in flight?

Maintain an ongoing estimate of bottleneck link rate: Congestion window

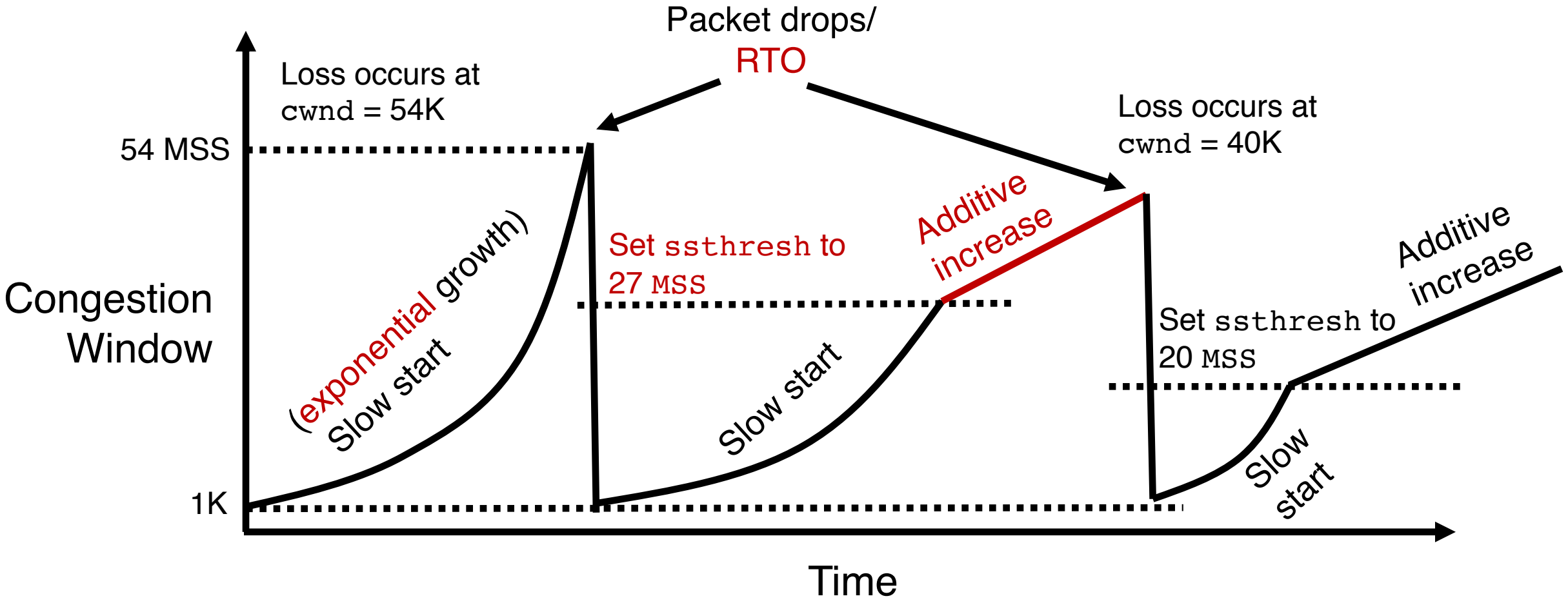
Distributed algorithm converging to an efficient and fair outcome

1. Get to a "reasonable" estimate of bottleneck link rate fast: Slow start
2. Adapt slowly once you get closer to the link rate: congestion avoidance

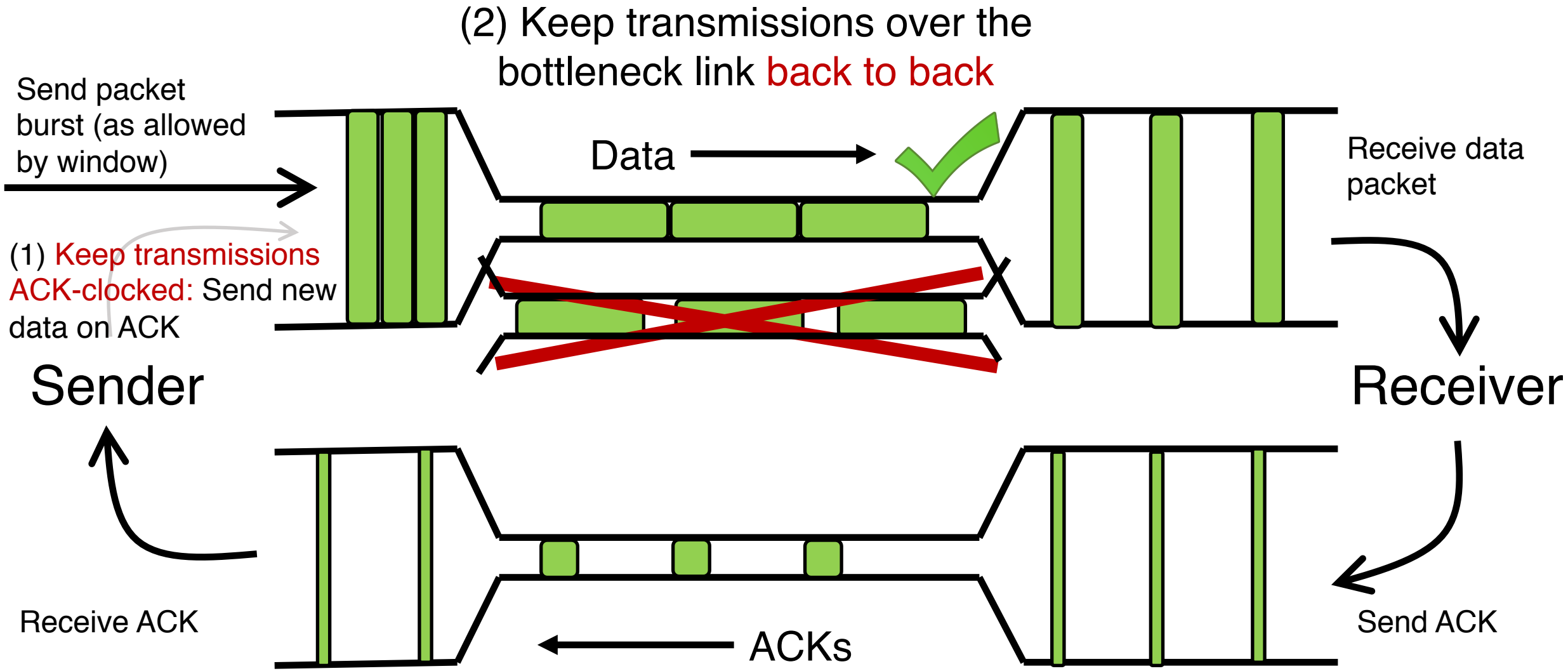


Review: TCP New Reno

Say `MSS` = 1 KByte



Review: Goal of steady state operation



Bandwidth-Delay Product

Steady state `cwnd` for a single flow

- Suppose the bottleneck link has rate C
- Suppose the propagation round-trip time is T
- Suppose:
  - Ignore transmission delays for this example
  - Sender transmits at highest rate with ACK clocking (steady state)

- Q1: What's the queueing delay at the bottleneck link?

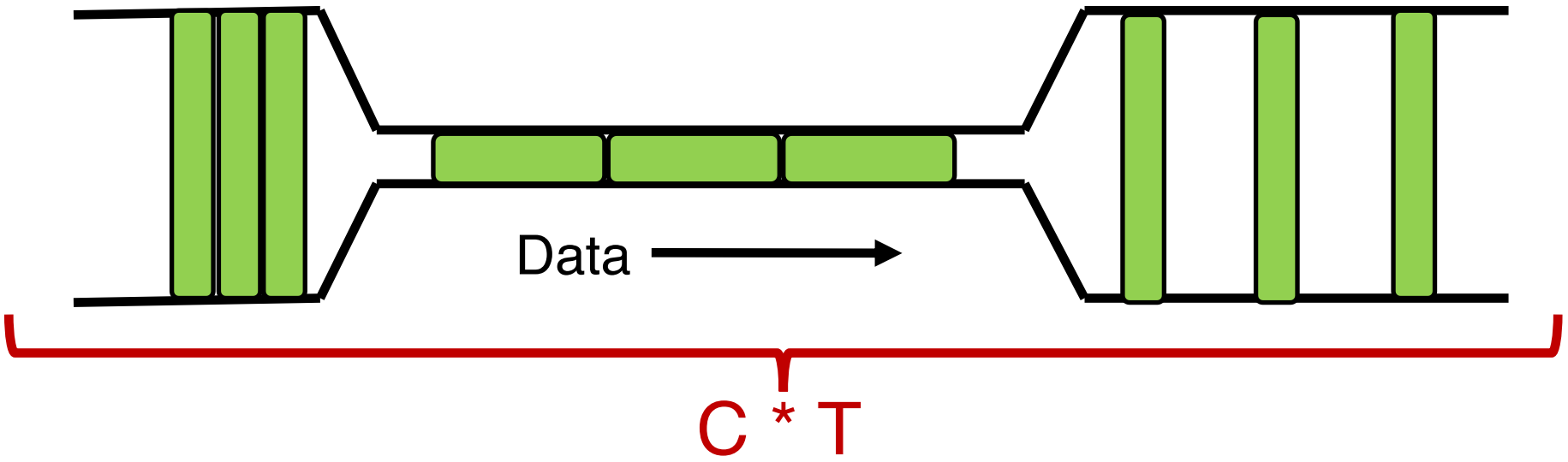
- Q2: how much data is in flight over a single RTT?

- C * T
  - ACKs take time T to arrive (total RTT = propagation RTT)
  - In the meantime, sender is transmitting at rate C

The Bandwidth-Delay Product

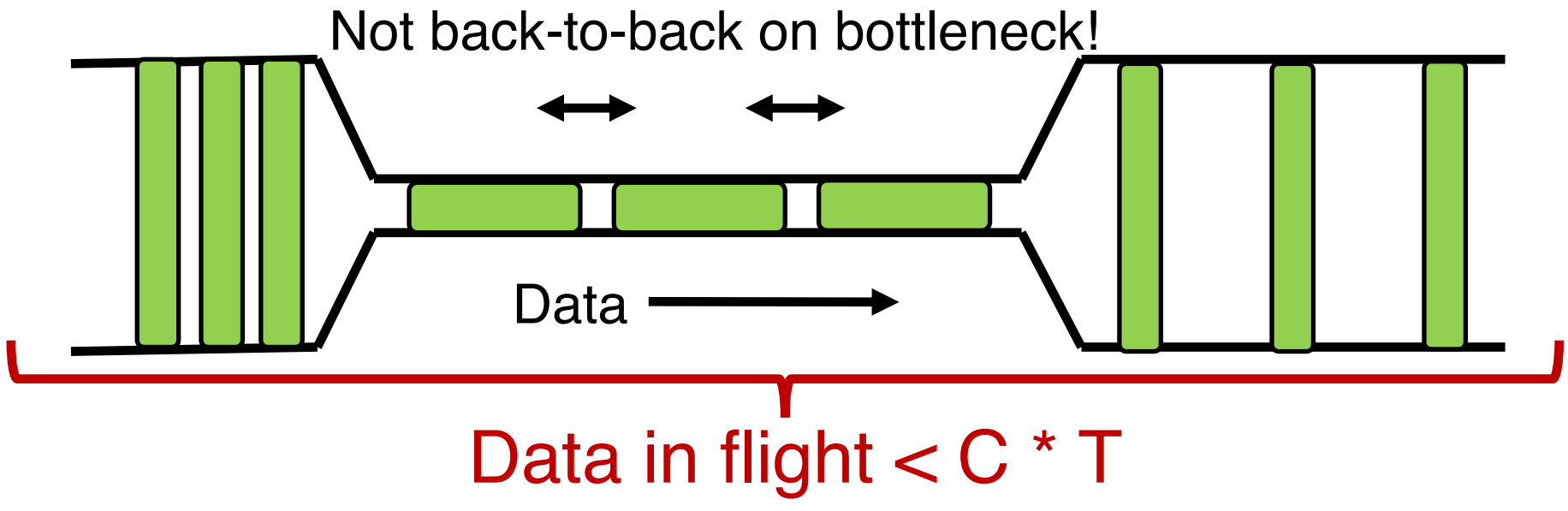
- C * T = bandwidth-delay product (BDP):
  - The amount of data in flight for a sender transmitting at the ideal rate during the ideal round-trip delay of a packet

- Note: this is just the amount of data “on the pipe” at steady state



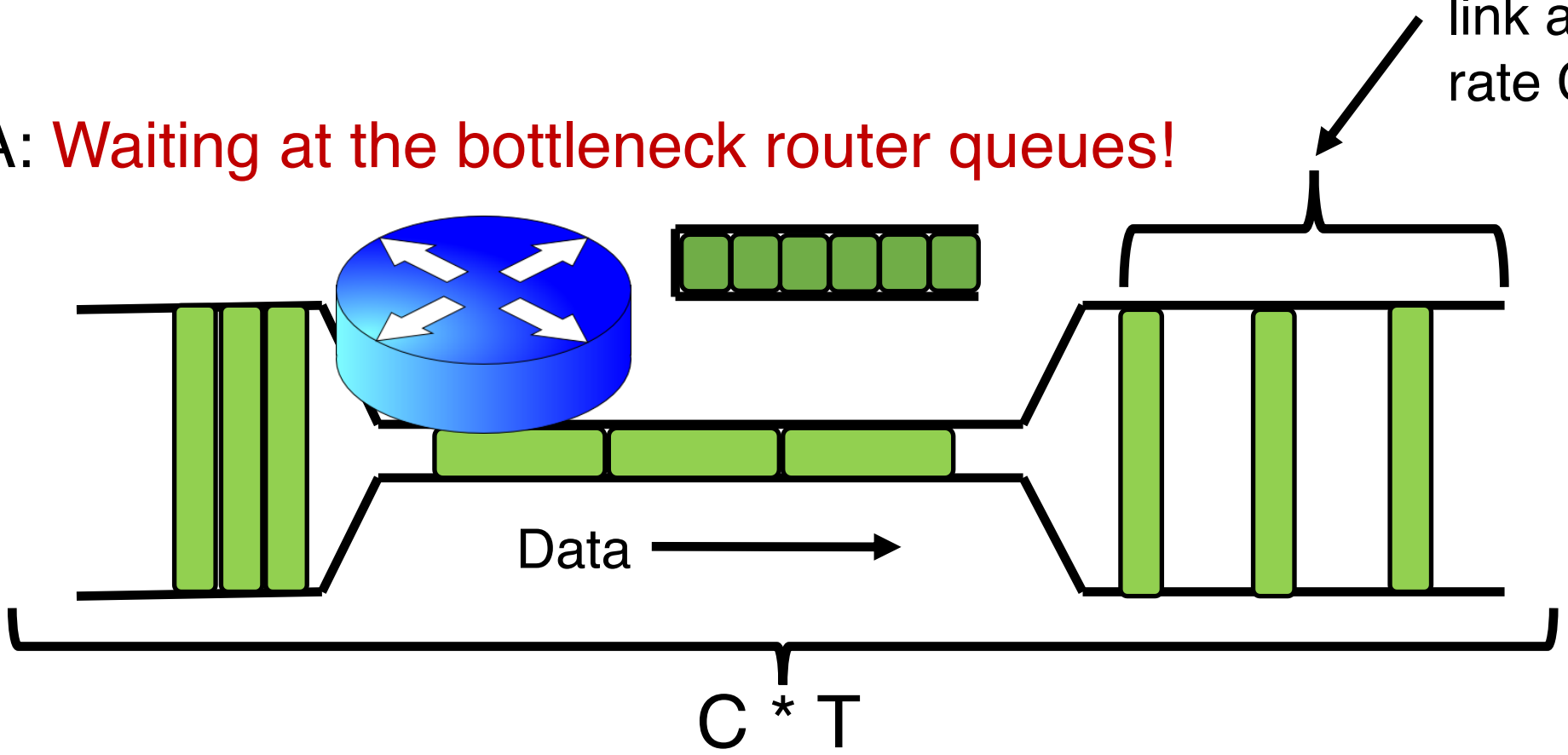
What happens if `cwnd` < C * T?

- i.e., the bottleneck link isn't kept fully busy
- i.e., the sender is sending even slower than the bottleneck link
- Since window = min(flow control window, congestion window)
  - Flow control window must be larger than the BDP to use network effectively



The Bandwidth-Delay Product

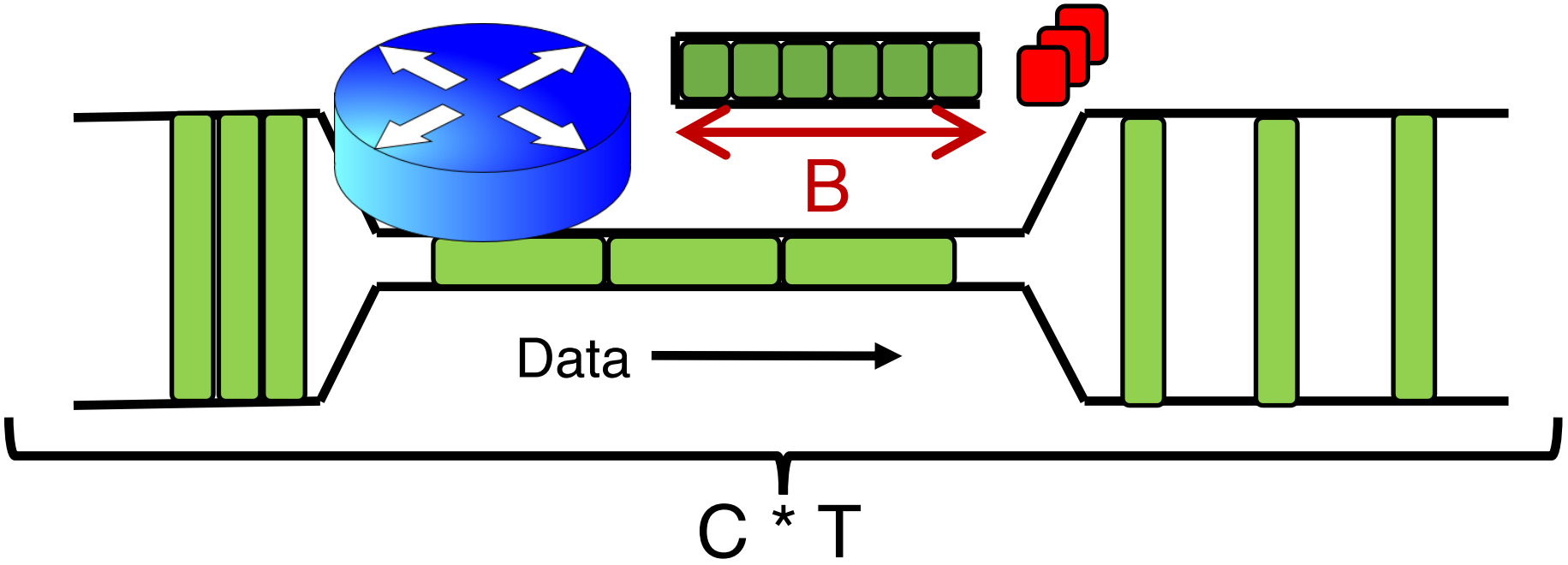
- Q: What happens if cwnd > C * T?
  - i.e., where are the rest of the in-flight packets?
- A: Waiting at the bottleneck router queues!



Router buffers and the max `cwnd`

- Router buffer memory is finite: queues can only be so long
  - If the router buffer size is B, there is at most B data waiting in the queue

- If `cwnd` increases beyond C * T + B, data is dropped!



Summary

- Bandwidth-Delay Product (BDP) governs the desired window size of a single flow at steady state
- If window < BDP, sender is using the network ineffectively
  - Corollary: flow control (advertised) window must be BDP or more
- If window > BDP + B, packet drops
  - If BDP < window < BDP + B, queueing and increased delays
- The bottleneck router buffer size governs how much the `cwnd` can exceed the BDP before packet drops occur

Detecting and Reacting to Packet Loss

Detecting packet loss

- So far, all the algorithms we've studied have a coarse loss detection mechanism: RTO timer expiration
  - Let the RTO expire, drop `cwnd` all the way to 1 MSS
- Analogy: you're driving
  - Get super close to the car in front (RTO) and then jam the brakes really hard (set cwnd := 1)
  - Q: Can you see the car in front from afar and slow proportionately?
- That is, can the sender see packet loss coming in advance?
  - And reduce `cwnd` gently?

Can we detect loss earlier than RTO?

- Key idea: use the information in the ACKs. How?

- Suppose successive (cumulative) ACKs contain the same ACK#
  - Also called duplicate ACKs
  - Occur when network is reordering packets, or a few (but not all) packets in the window were lost

- Reduce `cwnd` when you see many duplicate ACKs
  - Consider many dup ACKs a strong indication that packet was lost
  - Default threshold: 3 dup ACKs, i.e., triple duplicate ACK
  - Make cwnd reduction gentler than setting cwnd = 1; recover faster

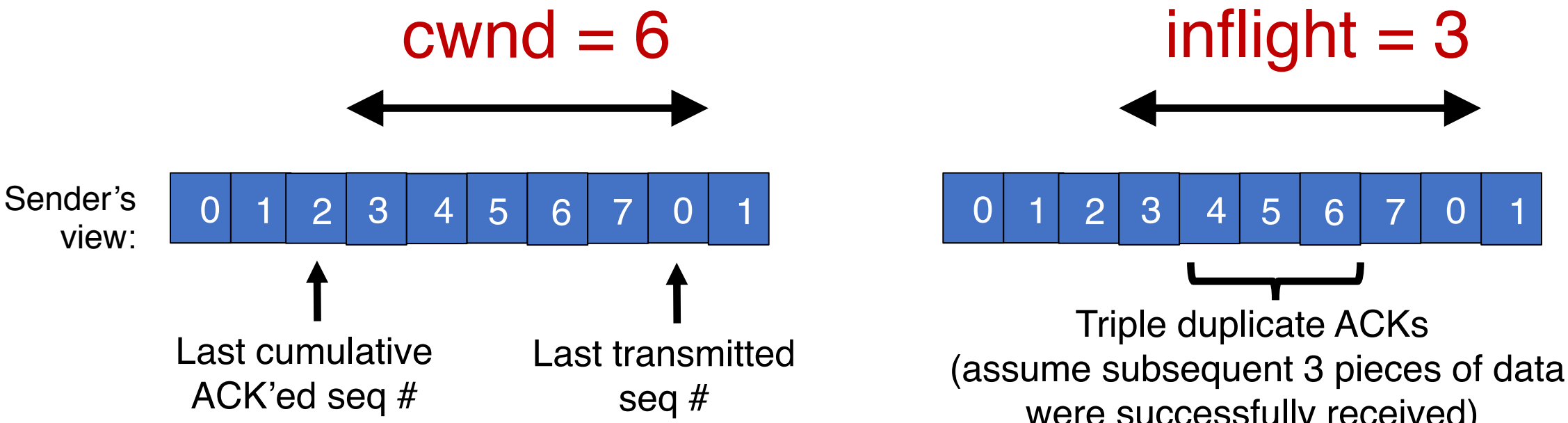
Fast Retransmit & Fast Recovery

Introduced in TCP New Reno

Used today by all TCP congestion control algorithms!

Distinction: In-flight versus window

- So far, window and in-flight referred to the same data
- Fast retransmit & fast recovery differentiate the two notions



`cwnd` is the interval between the last cumulatively ACK'ed seq# and the last transmitted seq#

`inflight` is the data currently believed to be in flight.

TCP fast retransmit (RFC 2581)

- The fact that ACKs are coming means that data is getting delivered to the receiver, albeit with some loss.
- Note: Before the dup ACKs arrive, we assume `inflight = cwnd`

- TCP sender does two actions with fast retransmit

TCP fast retransmit (RFC 2581)

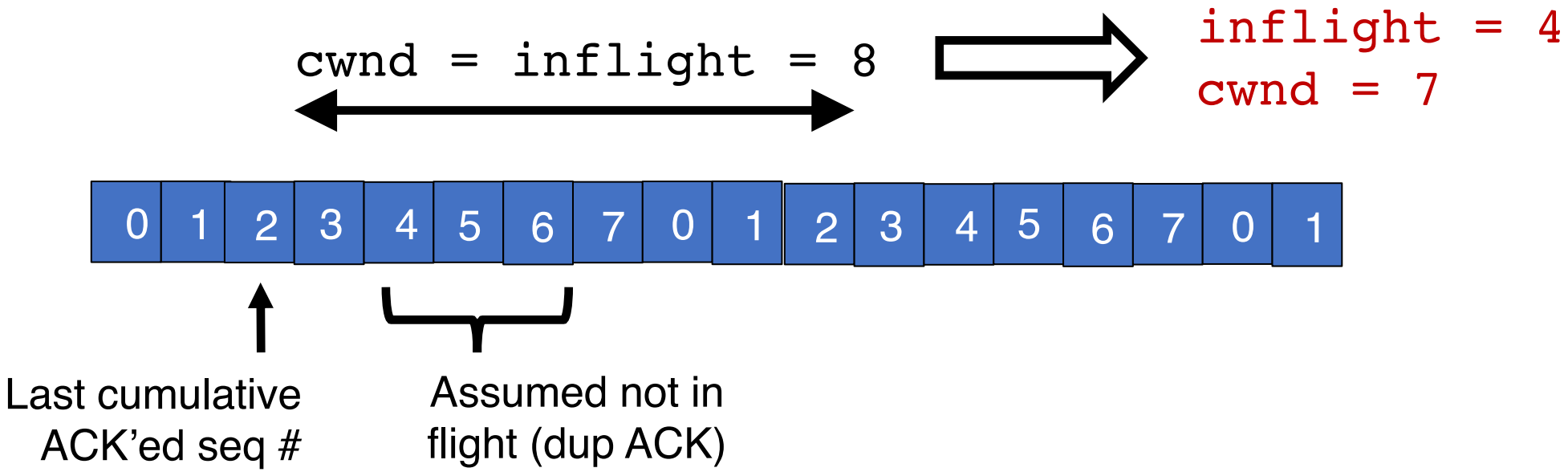
- (1) Reduce the `cwnd` and `in-flight` gently
  - Don't drop `cwnd` all the way down to 1 MSS

- Reduce the amount of in-flight data multiplicatively
  - Set `inflight → inflight / 2`
  - That is, set `cwnd = (inflight / 2) + 3MSS`
  - This step is called multiplicative decrease
  - Algorithm also sets `ssthresh` to (old)`inflight / 2`

- Aside: Multiplicative decrease is essential for fairness among TCP connections.

TCP fast retransmit (RFC 2581)

- Example: Suppose `cwnd` and `inflight` (before triple dup ACK) were both 8 MSS.
- After triple dup ACK, reduce `inflight` and `ssthresh` to 4 MSS
- *Assume* 3 of those 8 MSS no longer in flight; set `cwnd` = 7 MSS



TCP fast retransmit (RFC 2581)

- (2) The seq# from dup ACKs is immediately retransmitted
- That is, don't wait for an RTO if there is sufficiently strong evidence that a packet was lost

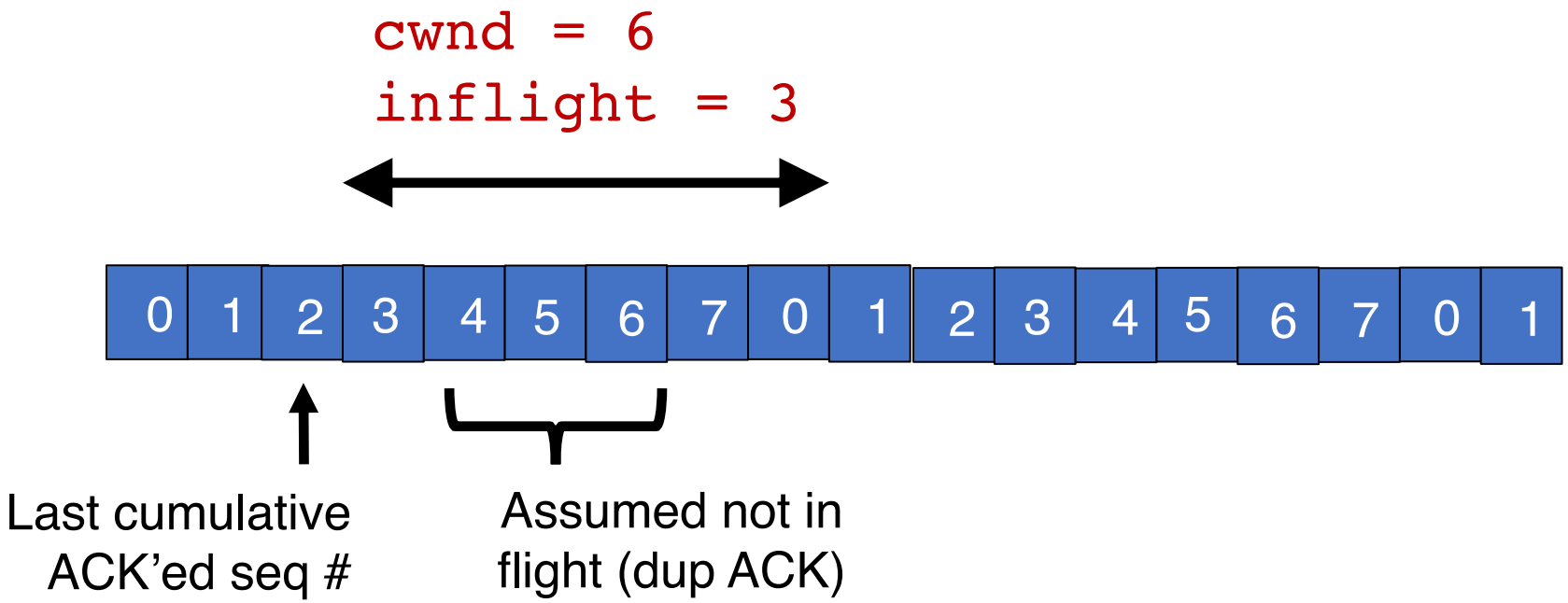
TCP fast recovery (RFC 2581)

- Sender keeps the reduced `inflight` until a new ACK arrives
  - New ACK: an ACK for the seq# that was just retransmitted
  - May also include the (three or more) pieces of data that were subsequently delivered to generate the duplicate ACKs

- Conserve packets in flight: transmit *some* data over lossy periods (rather than no data, which would happen if `cwnd` := 1)

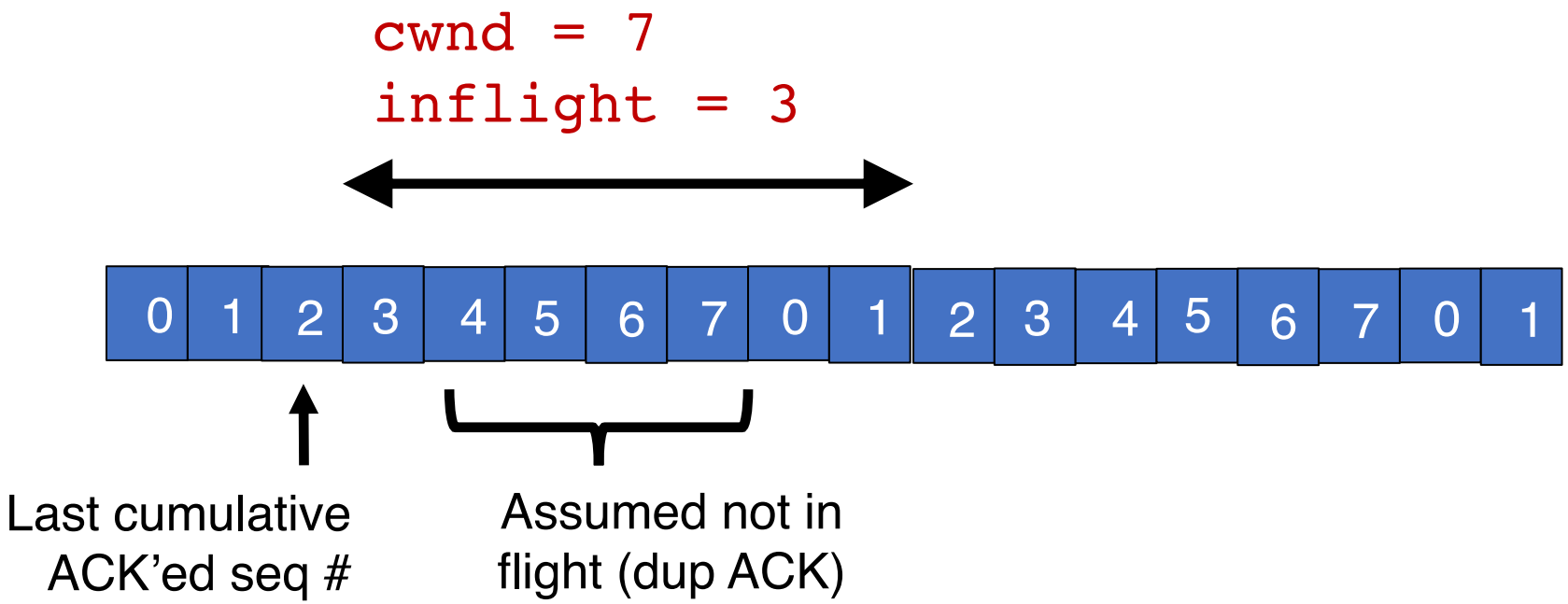
TCP fast recovery (RFC 2581)

- Keep incrementing `cwnd` by 1 MSS for each dup ACK



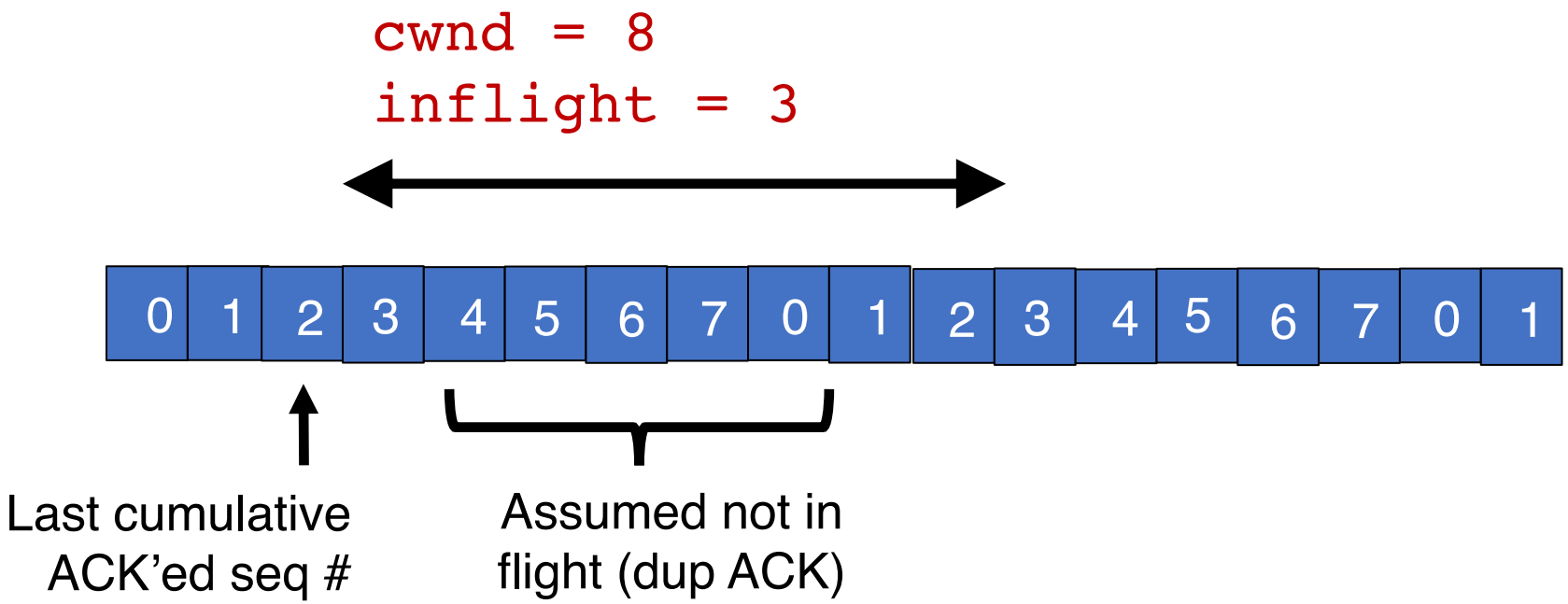
TCP fast recovery (RFC 2581)

- Keep incrementing `cwnd` by 1 MSS for each dup ACK



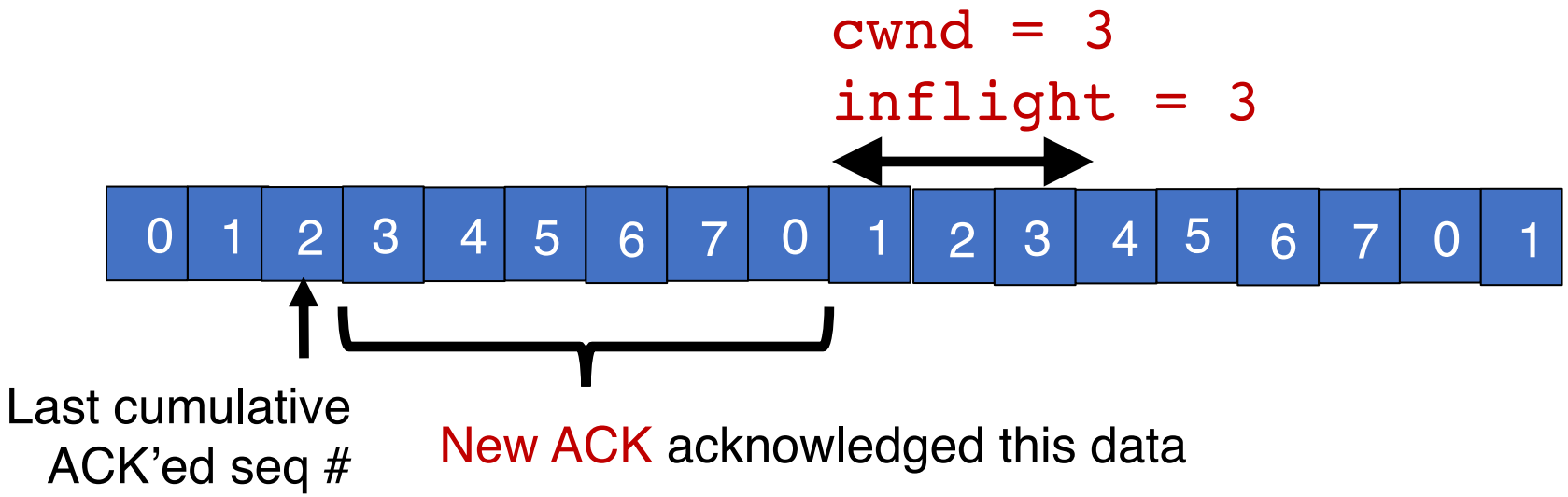
TCP fast recovery (RFC 2581)

- Keep incrementing `cwnd` by 1 MSS for each dup ACK



TCP fast recovery (RFC 2581)

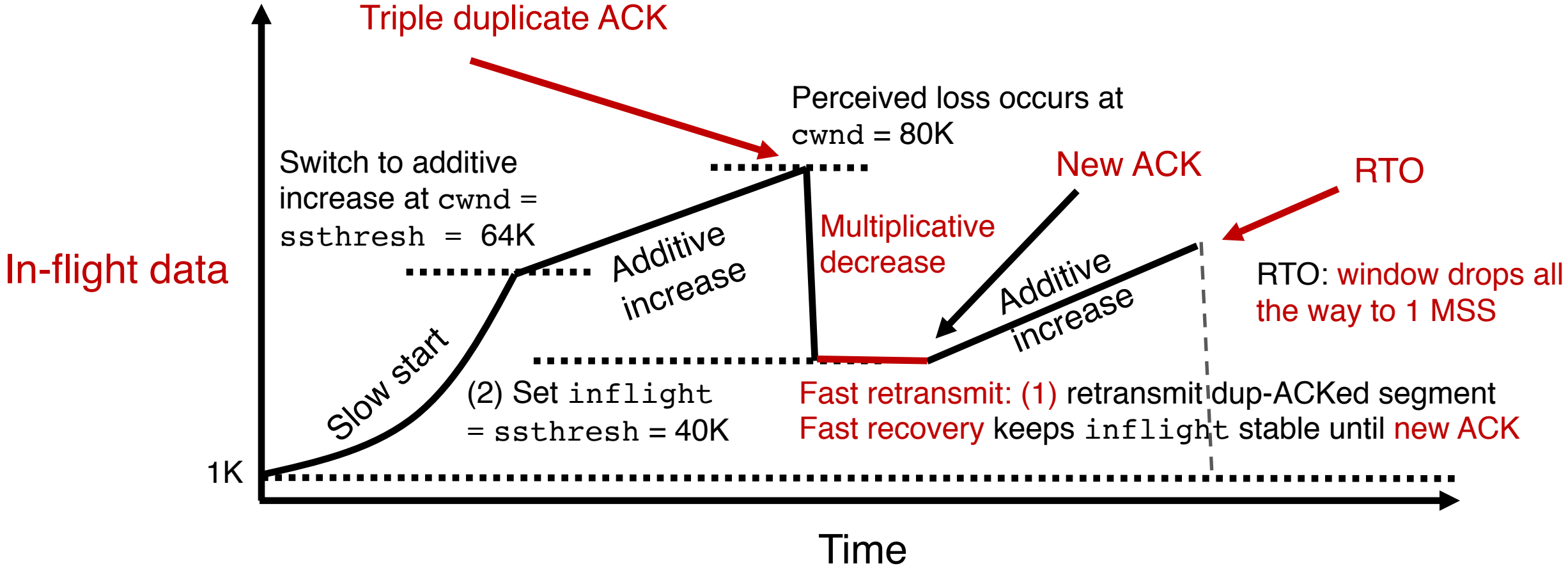
- Eventually a new ACK arrives, acknowledging the retransmitted data and all data in between
- Deflate `cwnd` to half of `cwnd` before fast retransmit.
  - `cwnd` and `inflight` are aligned and equal once again
- Perform additive increase from this point! (`cwnd = ssthresh`)



Additive Increase/Multiplicative Decrease

Say `MSS` = 1 KByte

Default `ssthresh` = 64KB = 64 `MSS`



TCP New Reno performs additive increase and multiplicative decrease of its congestion window.

In short, we often refer to this as AIMD.

Multiplicative decrease is a part of all TCP algorithms, including BBR.
[It is necessary for fairness across TCP flows.]

Summary: TCP loss detection & reaction

- Don't wait for an RTO and then set the `cwnd` to 1 MSS
  - Tantamount to waiting to get super close to the car in front and then jamming the brakes very hard
- Instead, react proportionately by sensing pkt loss in advance

Fast Retransmit

- Triple dup ACK: sufficiently strong signal that network has dropped data, before RTO
- Immediately retransmit data
- Multiplicatively decrease in-flight data to half of its value

Fast Recovery

- Maintain this reduced amount of in-flight data as long as dup ACKs arrive
  - Data is successfully getting delivered
- When new ACK arrives, do additive increase from there on