

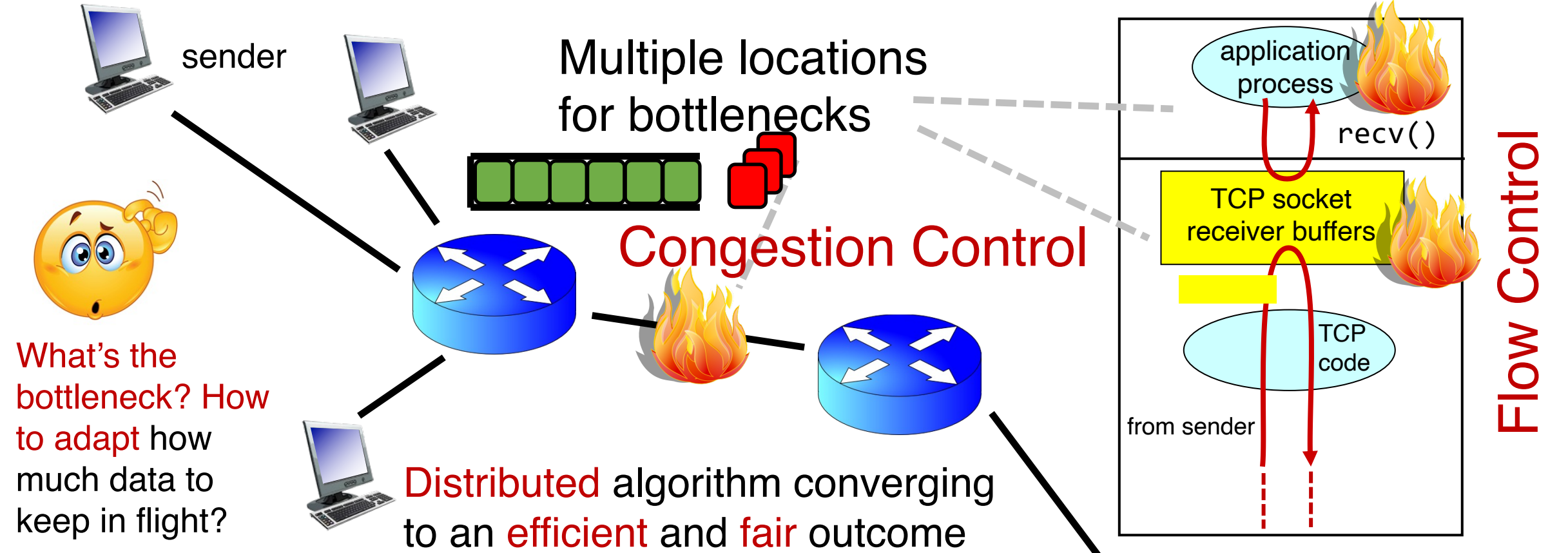
CS 352

Congestion Control (Part 2)

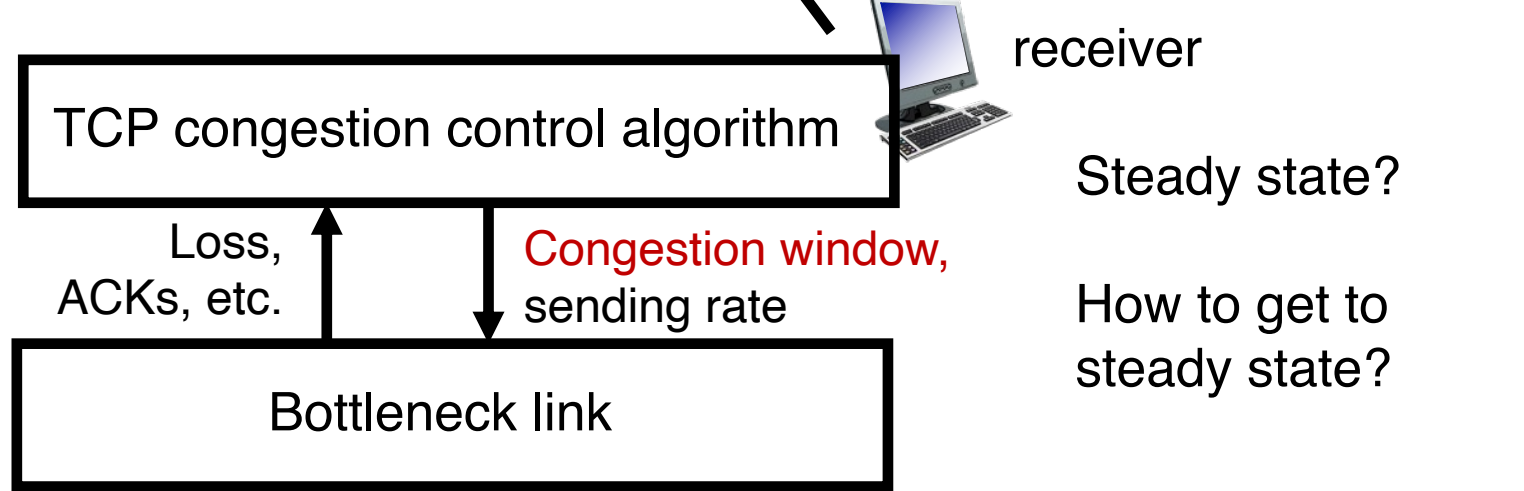
Lecture 17

<http://www.cs.rutgers.edu/~sn624/352-F22>

Srinivas Narayana



Sense and React



Congestion window

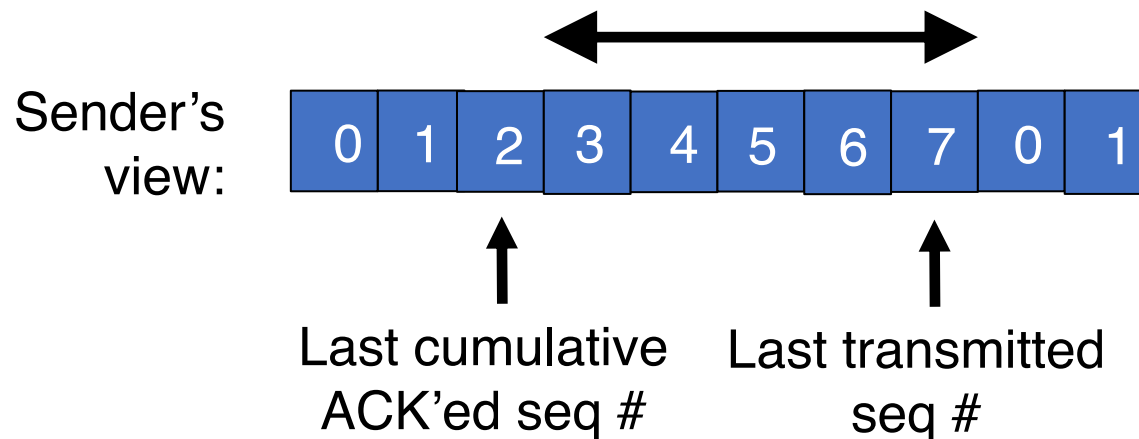
- The sender maintains an estimate of the amount of in-flight data needed to keep the pipe full without congesting it.
- This estimate is called the **congestion window (cwnd)**
- Recall: There is a relationship between the sending rate (throughput) and the sender's window: sender transmits a window's worth of data over an RTT duration
 - **Rate = window / RTT**

Interaction b/w flow & congestion control

- Use window = **min**(congestion window, receiver advertised window)
- Overwhelm neither the receiver nor network links & routers

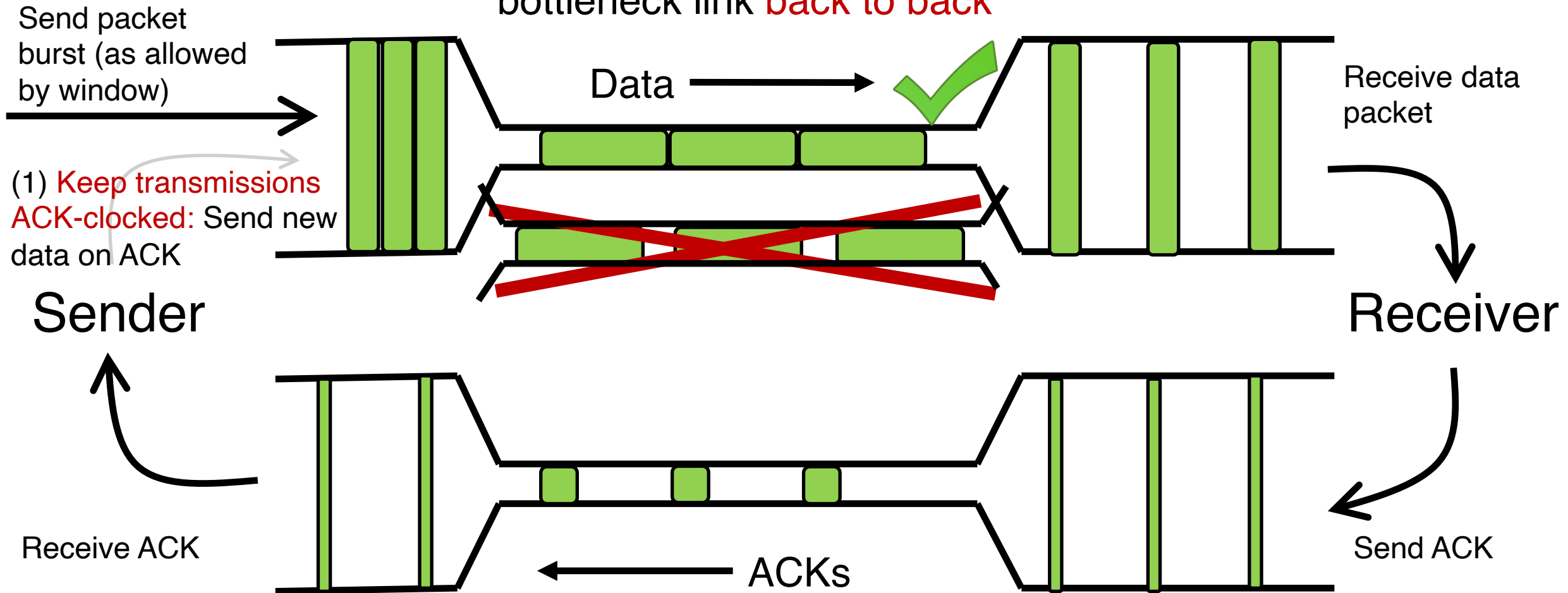
Window \leq Congestion window (congestion control)

Window \leq Advertised window (flow control)



Review: Goal of steady state operation

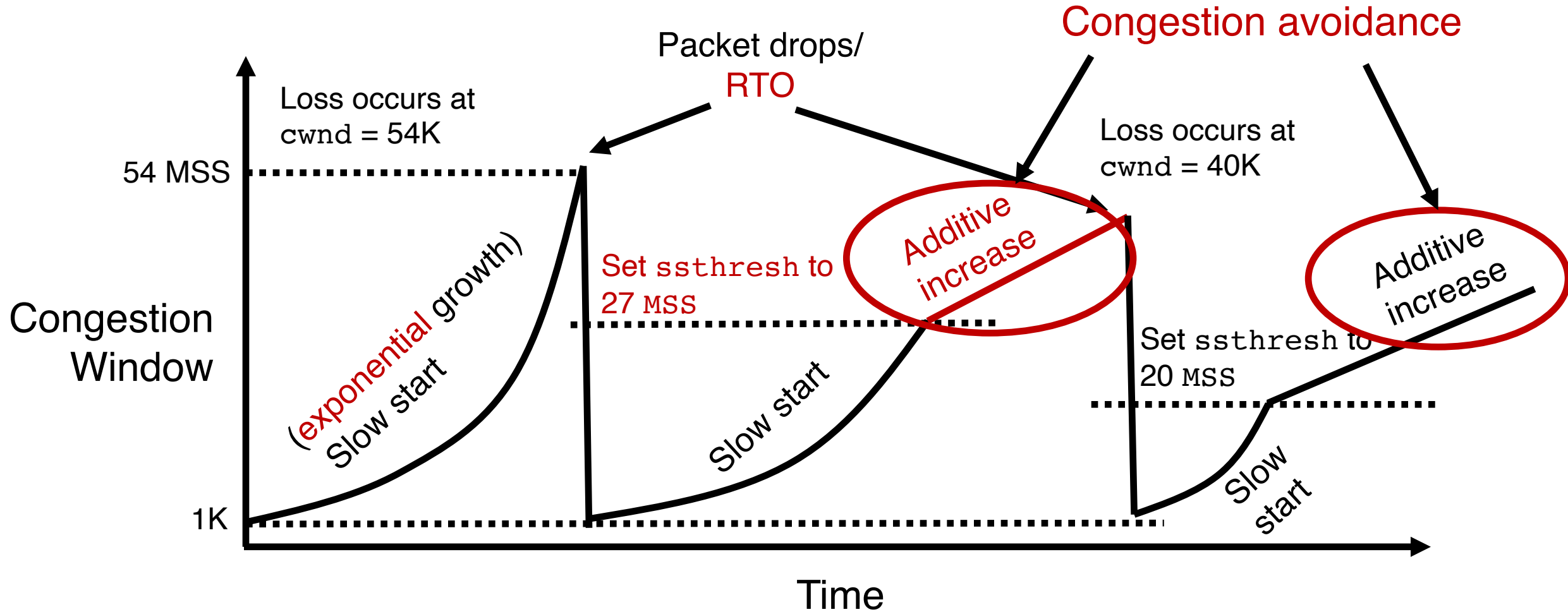
(2) Keep transmissions over the bottleneck link **back to back**



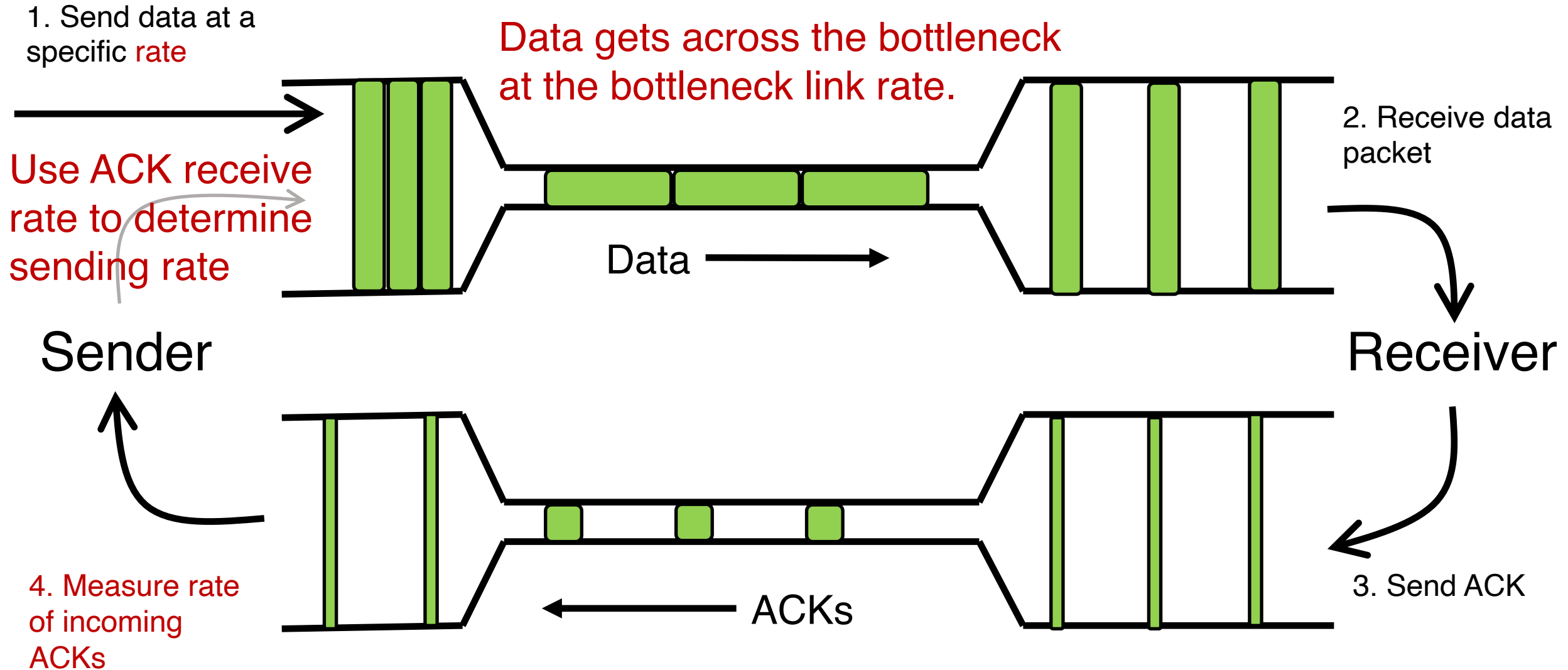
Review: Getting to steady: TCP New Reno

Say MSS = 1 KByte

Adjusting closer to steady state:
different methods...



TCP BBR: finding the bottleneck link rate

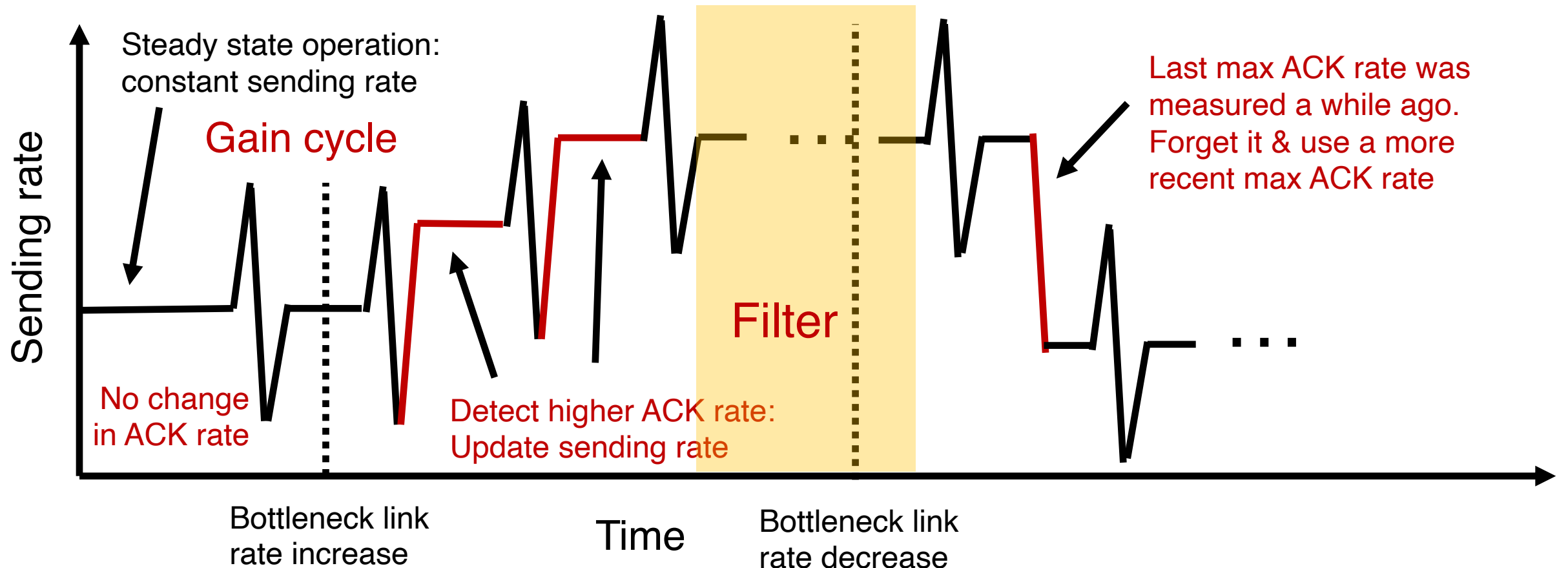


TCP BBR: finding the bottleneck link rate

- Assuming that the link rate of the bottleneck
 - == the rate of data getting across the bottleneck link
 - == the rate of data getting to the receiver
 - == the rate at which ACKs are generated by the receiver
 - == the rate at which ACKs reach the sender
- Measuring ACK rate provides an estimate of bottleneck link rate
- **BBR: Send at the maximum ACK rate measured in the recent past**
 - Update max with new bottleneck rate estimates, i.e., larger ACK rate
 - Forget estimates last measured a long time ago
 - Incorporated into a rate **filter**

TCP BBR: Adjustments by gain cycling

- BBR periodically increases its sending rate by a gain factor to see if the link rate has increased (e.g., due to a path change)



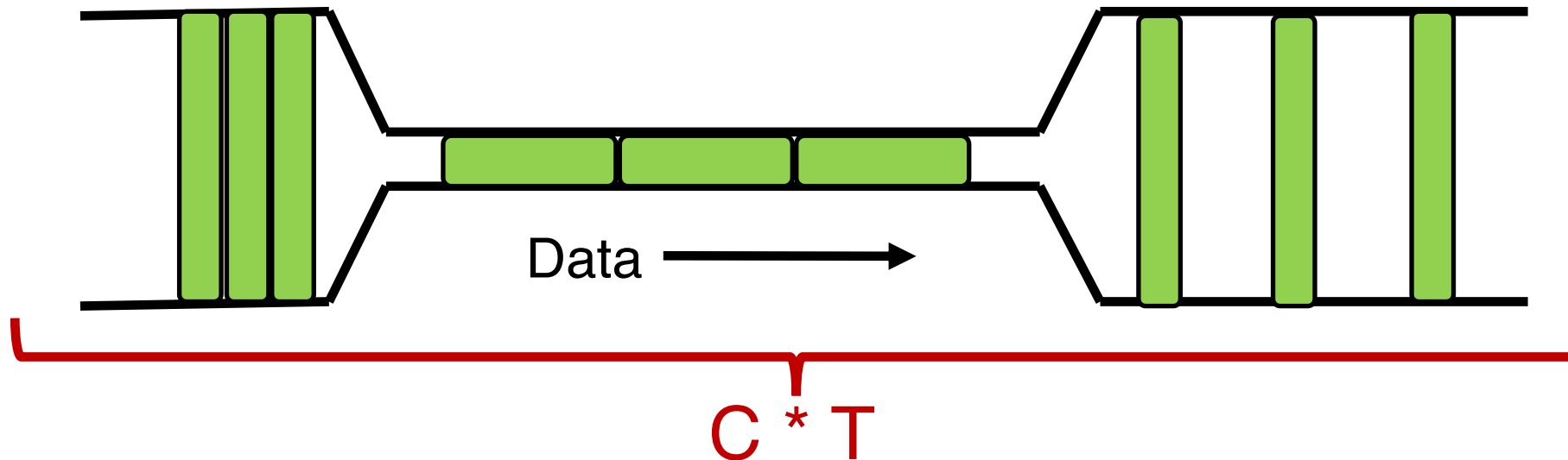
Bandwidth-Delay Product

Steady state cwnd for a single flow

- Suppose the bottleneck link has rate C
- Suppose the propagation round-trip delay (propRTT) between sender and receiver is T
- Ignore transmission delays for this example;
- Assume steady state: highest sending rate with no bottleneck congestion
- Q: how much data is in flight over a single RTT?
- $C * T$ data i.e., amount of data unACKed at any point in time
- ACKs take time T to arrive (without any queueing). In the meantime, sender is transmitting at rate C

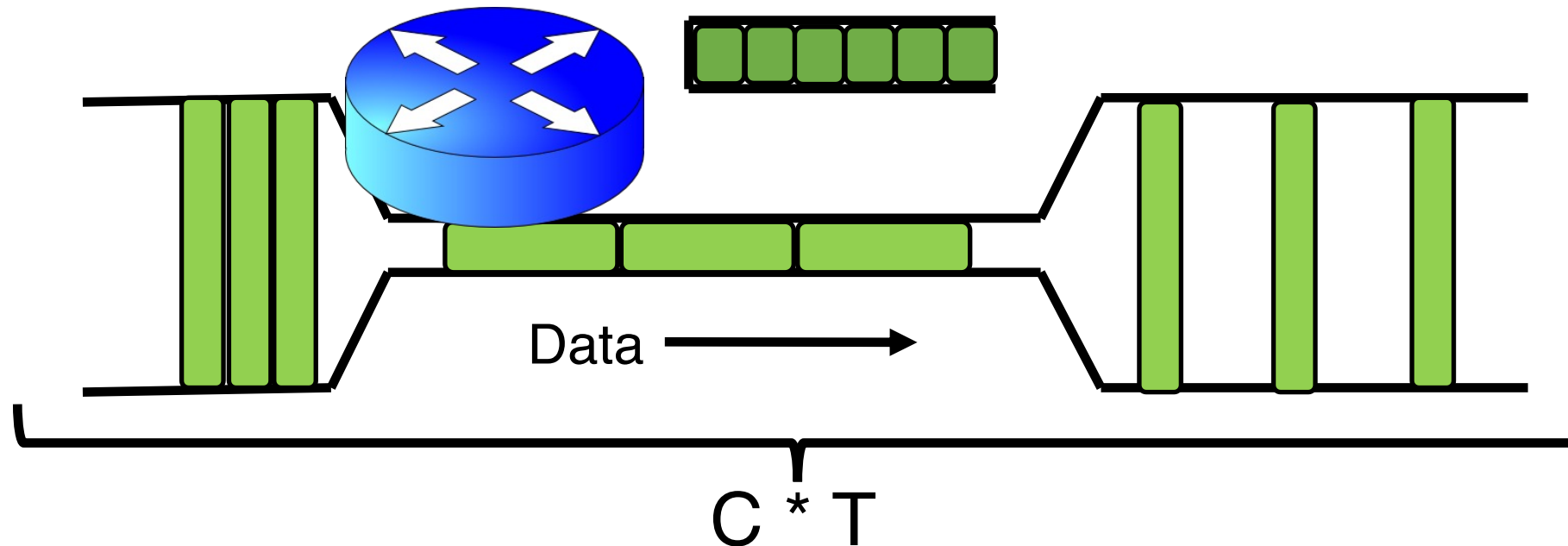
The Bandwidth-Delay Product

- $C * T =$ **bandwidth-delay product**:
 - 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”



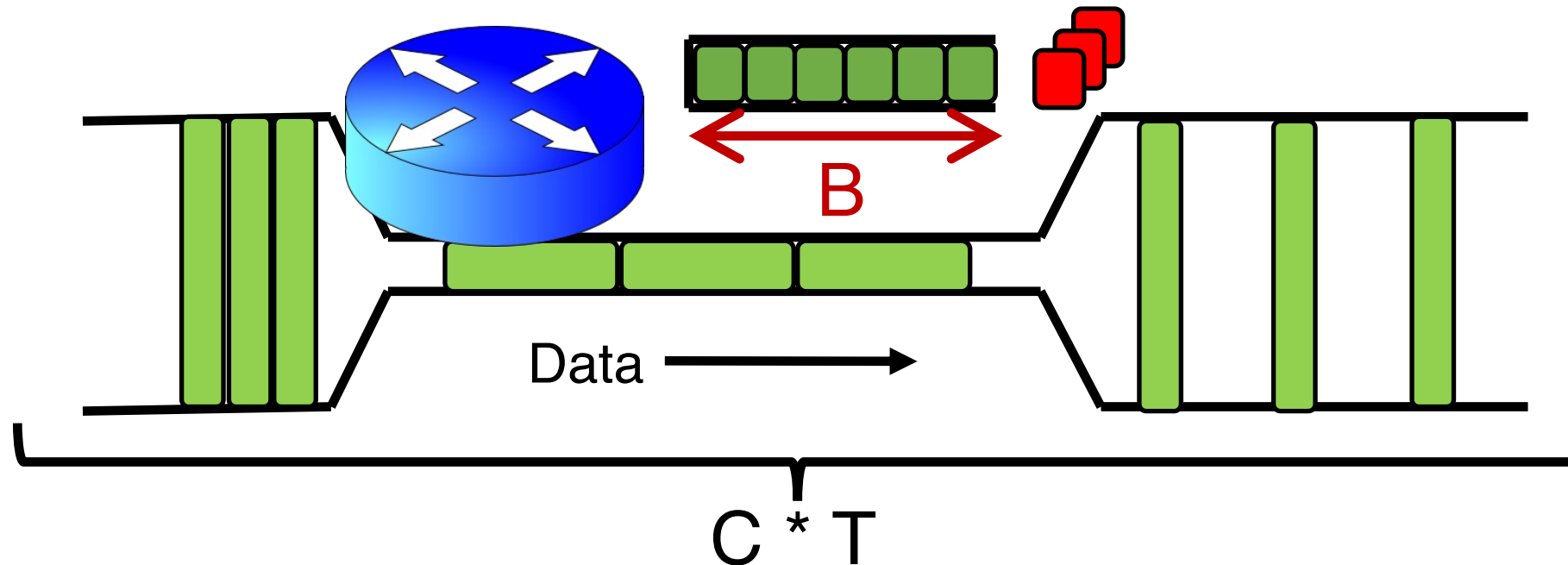
The Bandwidth-Delay Product

- 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 window size of a single flow at steady state
- The bottleneck router buffer size governs how much the `cwnd` can exceed the BDP before packet drops occur
- BDP is the ideal desired window size to use the full bottleneck link, without any queueing.
 - Accommodating flow control, also the min socket buffer size to use the bottleneck link fully

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 a car
 - You're waiting until the next car in front is super close to you (RTO) and then hitting the brakes really hard (set `cwnd := 1`)
 - Q: Can you see obstacles from afar and slow down proportionately?
- That is, can the sender see packet loss coming in advance?
 - And reduce `cwnd` more 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 one (but not most) 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

Distinction: In-flight versus window

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

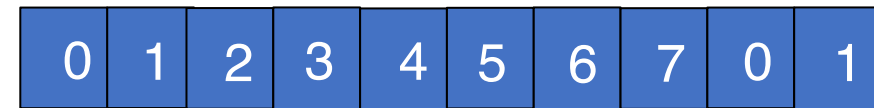
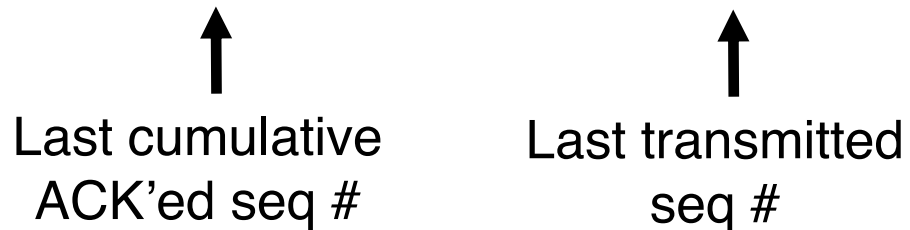
cwnd = 6



inflight = 3



Sender's view:



Triple duplicate ACKs
(assume subsequent 3 pieces of data were successfully received)

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)

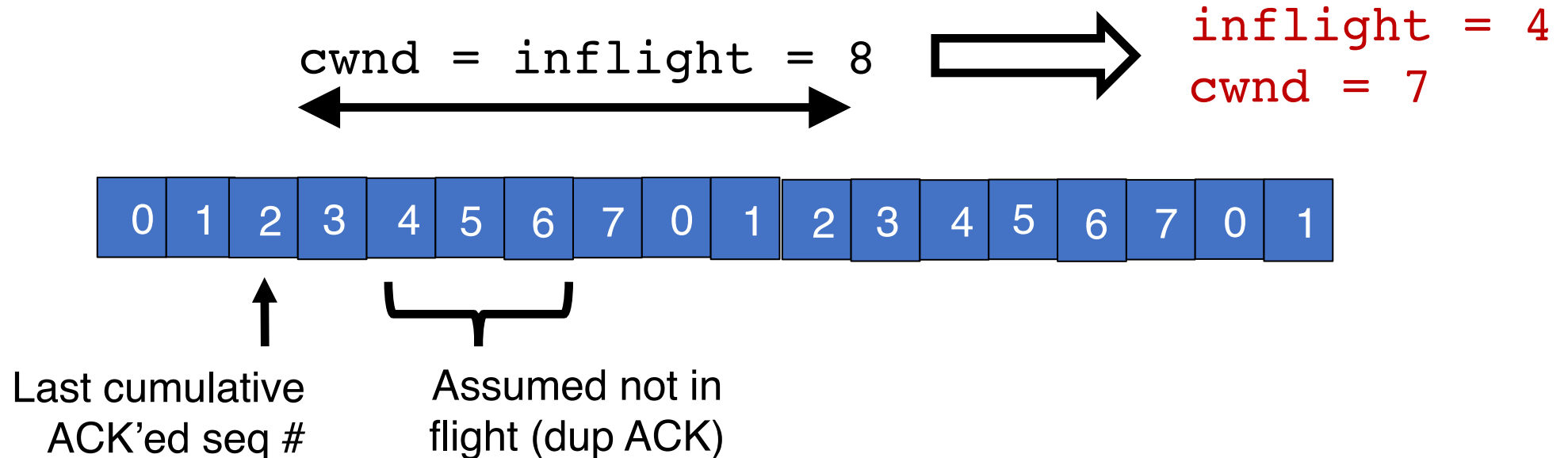
- The fact that ACKs are coming means that data is getting delivered to the receiver, albeit with some loss.
- Note: Before the dup ACKs arrive, we assume `inflight = cwnd`
- TCP sender does two actions with fast retransmit

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` \rightarrow `inflight / 2`
 - That is, set `cwnd` $=$ `(inflight / 2) + 3MSS`
 - This step is called **multiplicative decrease**
 - Algorithm also sets `ssthresh` to `inflight / 2`

TCP **fast retransmit** (RFC 2581)

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



TCP fast retransmit (RFC 2581)

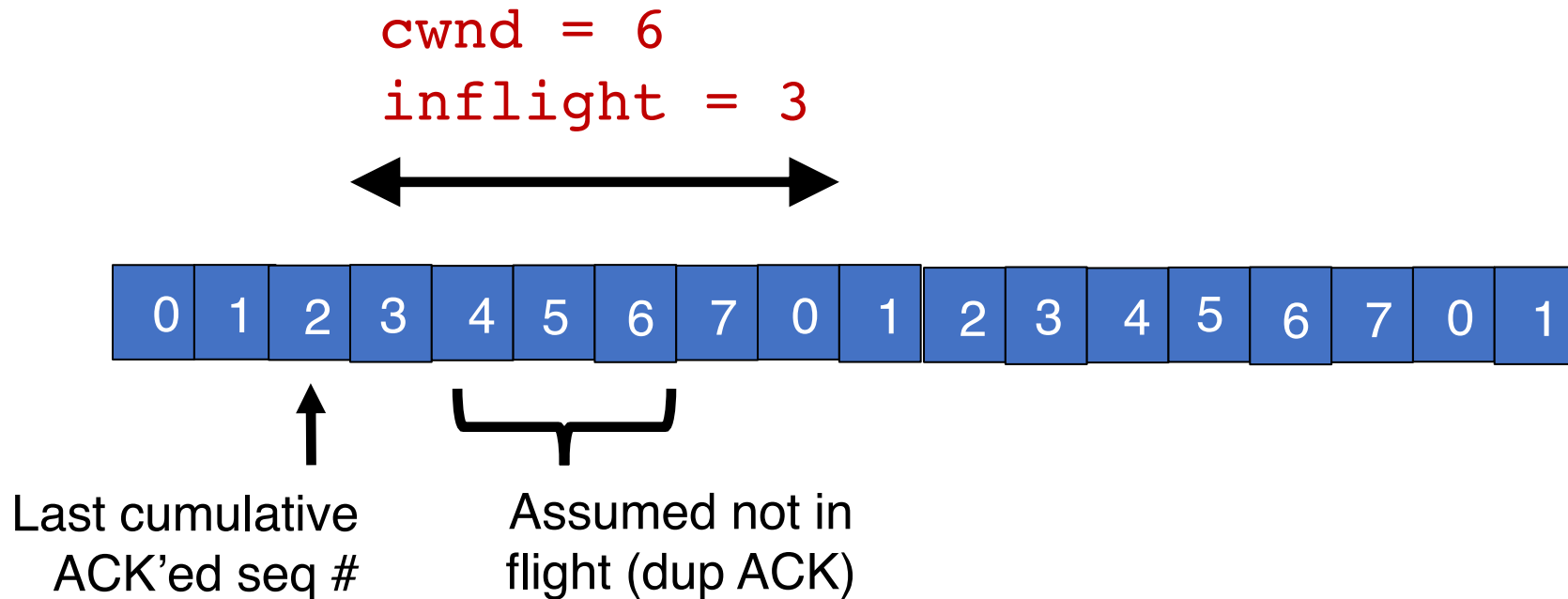
- (2) The seq# from dup ACKs is **immediately retransmitted**
- That is, **don't wait for an RTO** if there is sufficiently strong evidence that a packet was lost

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`)

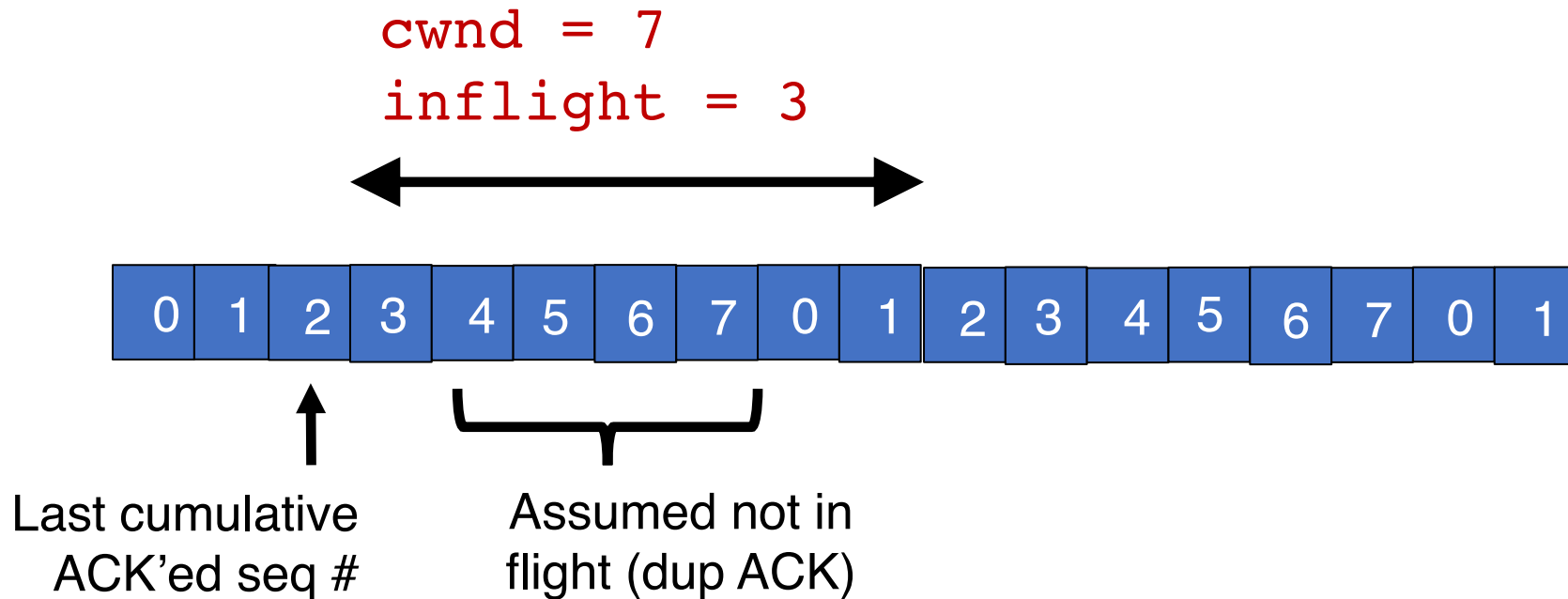
TCP *fast recovery* (RFC 2581)

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



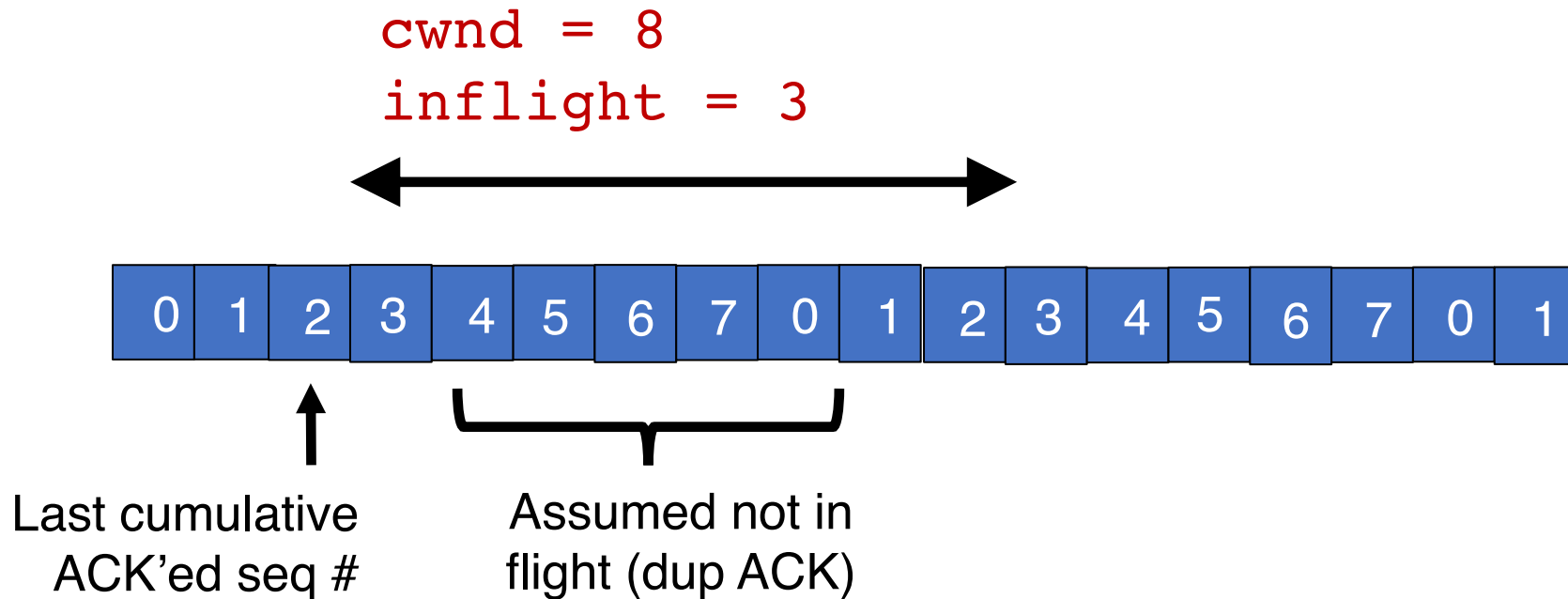
TCP *fast recovery* (RFC 2581)

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



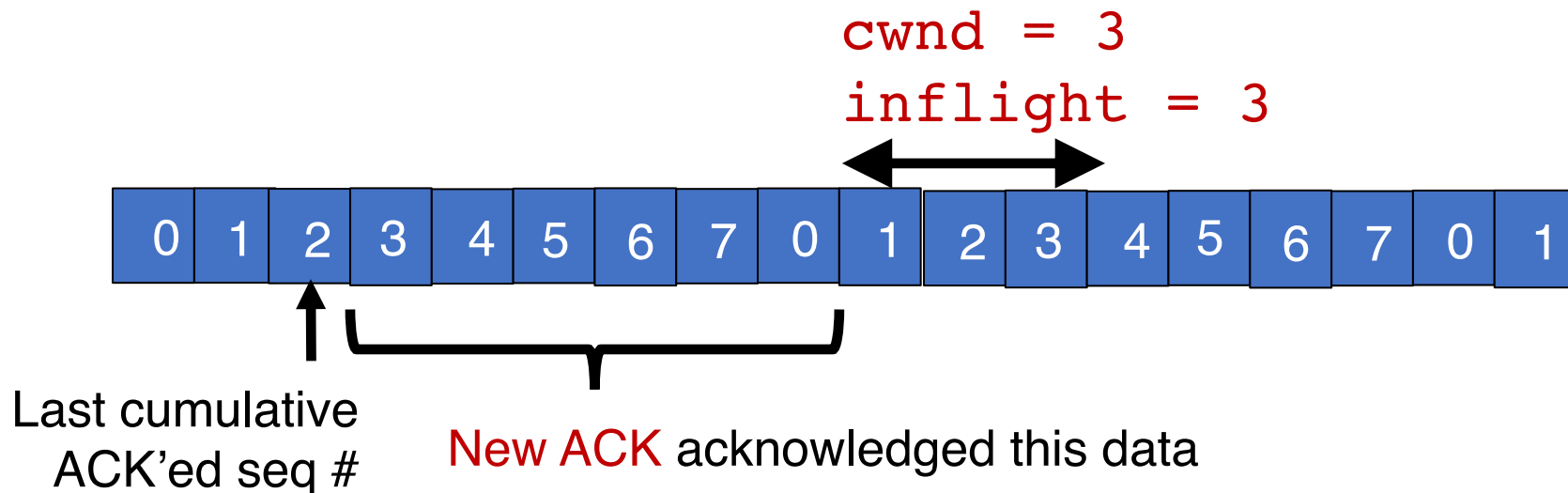
TCP *fast recovery* (RFC 2581)

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



TCP **fast recovery** (RFC 2581)

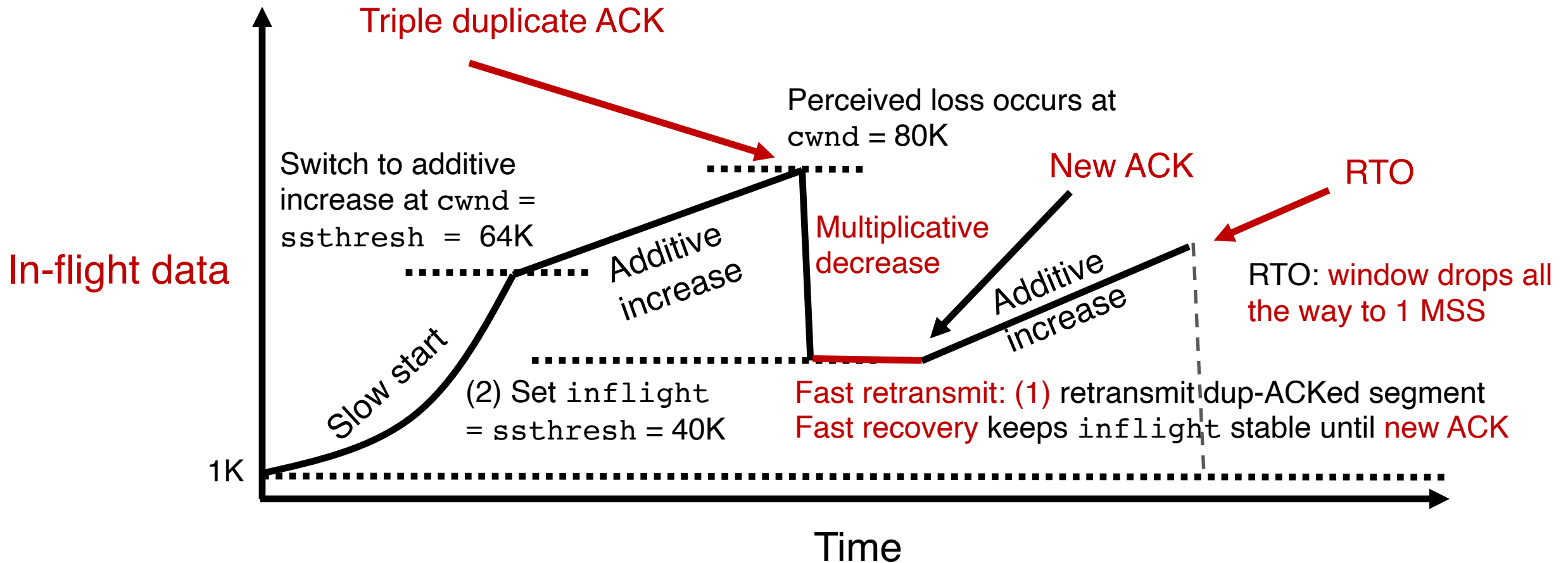
- Eventually a **new ACK** arrives, acknowledging the retransmitted data and all data in between
- Deflate `cwnd` to half of `cwnd` before fast retransmit.
 - `cwnd` and `inflight` are aligned and equal once again
- Perform **additive increase** from this point!



Additive Increase/Multiplicative Decrease

Say MSS = 1 KByte

Default ssthresh = 64KB = 64 MSS



TCP New Reno performs additive increase and multiplicative decrease of 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 really 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