Congestion Control II

Lecture 18

http://www.cs.rutgers.edu/~sn624/352-F24

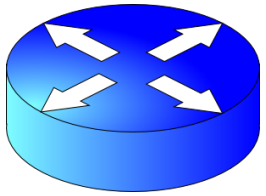
Srinivas Narayana

RUTGERS
UNIVERSITY | NEW BRUNSWICK

sender

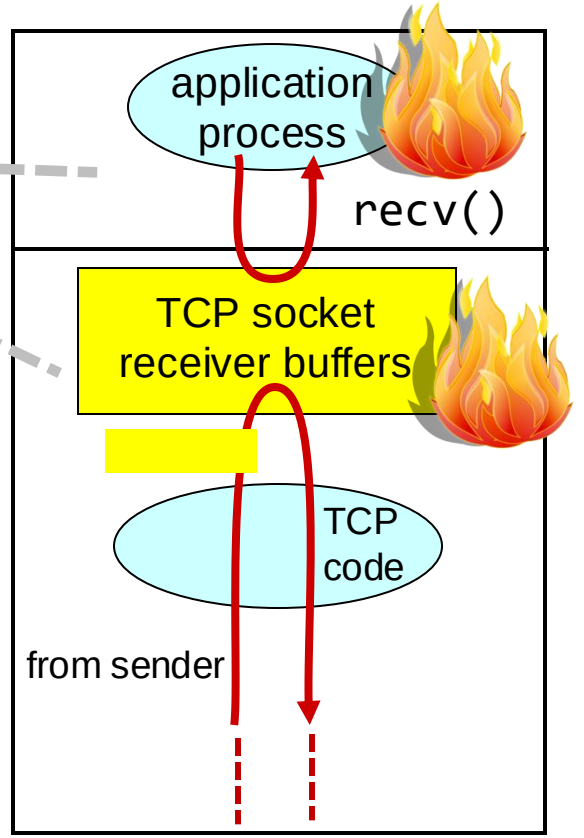
Multiple locations for bottlenecks

Congestion Control



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

Distributed algorithm converging to an efficient and fair outcome



Flow Control

Sense and React

Signals

Knobs

receiver

TCP congestion control algorithm

Loss, ACKs, etc.

Congestion window, sending rate

Bottleneck link

Steady state?

How to get to steady state?

Congestion window

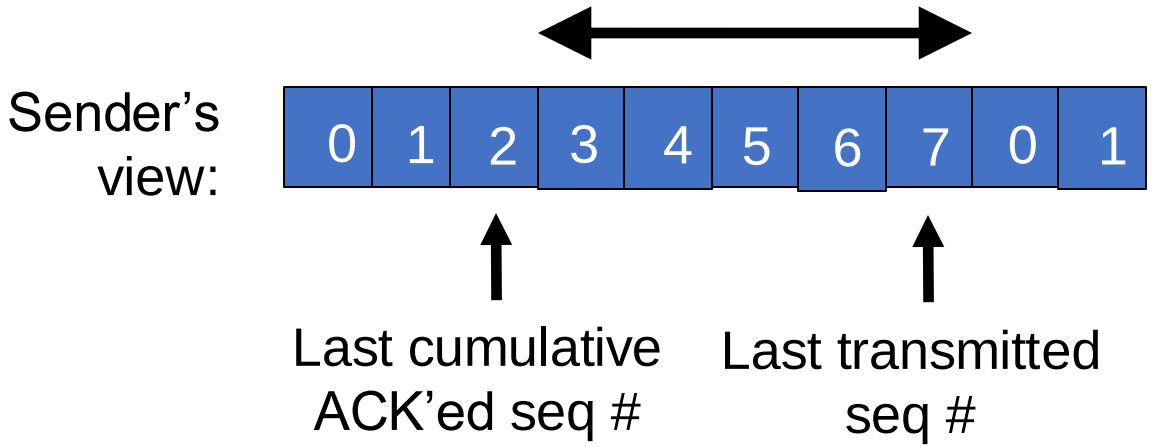
- The sender maintains an estimate of the amount of in-flight data needed to keep the link fully busy without congesting it
- This estimate is called the congestion window (cwnd)
- Recall: There is a relationship between the sending rate (throughput) and the sender's window: sender transmits a window's 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

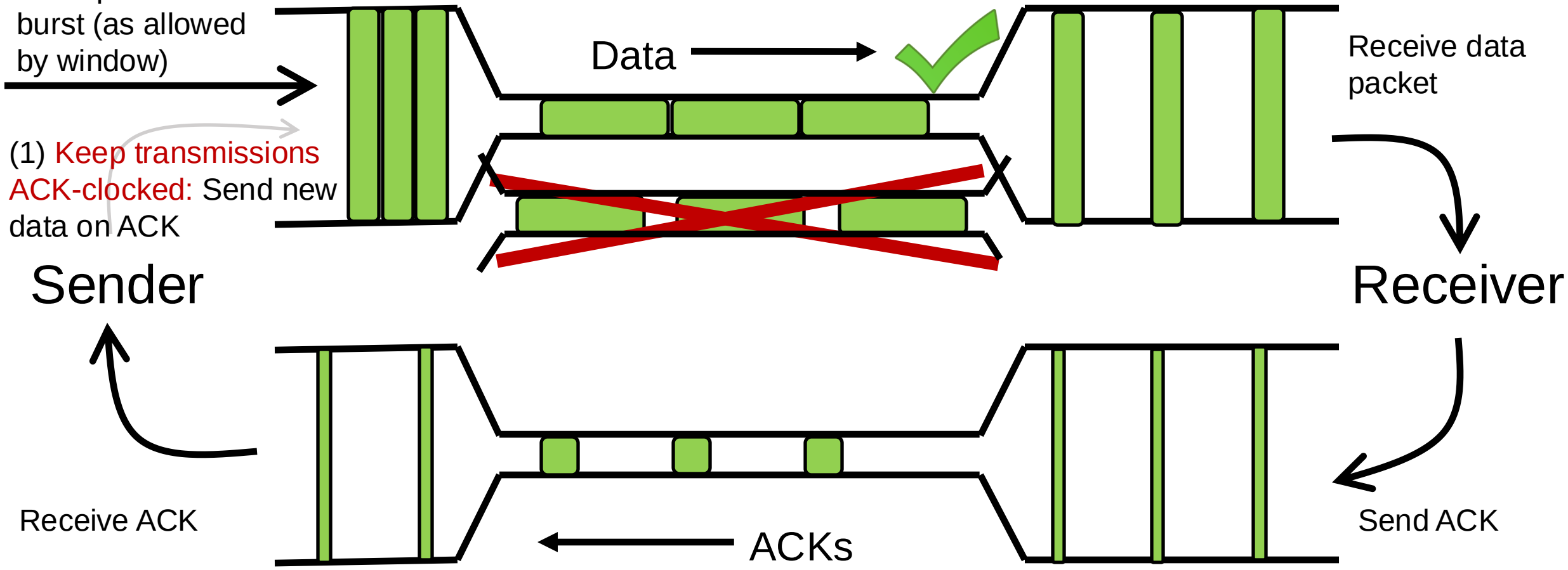
Window <= Congestion window (congestion control)

Window <= Advertised window (flow control)



Review: Steady state operation

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



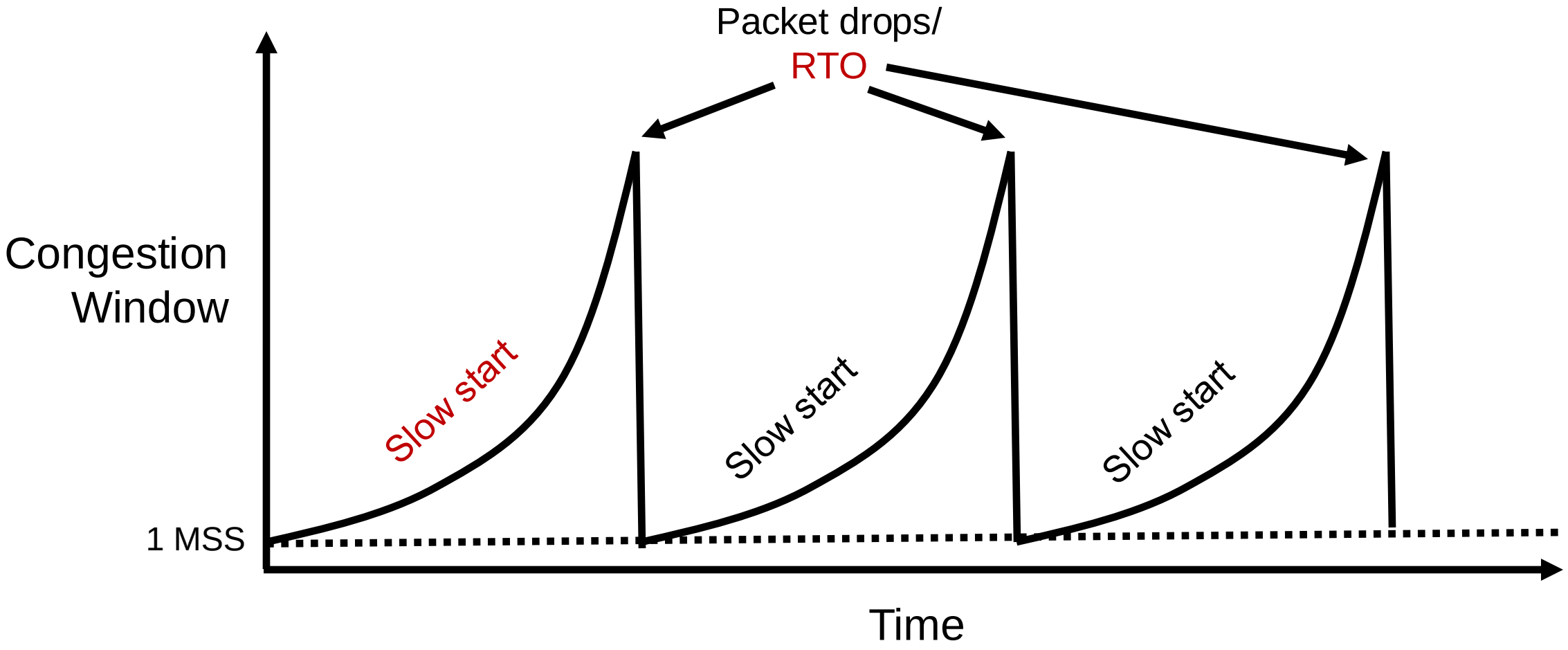
Review: Slow start

Q: How to get to steady state?

Problems:

- Congestion window grows too fast!
- Congestion window drops too fast!

Need gentler adaptation of cwnd closer to steady state



TCP Congestion Avoidance

Two congestion control algorithms

TCP New Reno

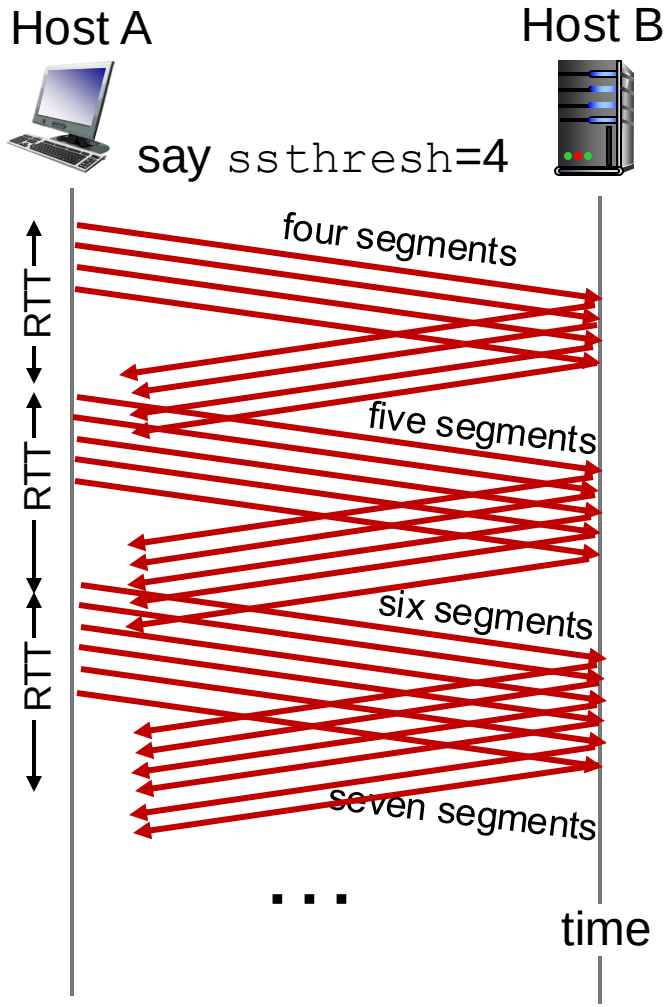
- The most studied, classic “textbook” TCP algorithm
- The primary knob is congestion window
- The primary signal is packet loss (RTO)
- Adjustment using additive increase

TCP BBR

- Recent algorithm developed & deployed by Google
- The primary knob is sending rate
- The primary signal is rate of incoming ACKs
- Adjustment using gain cycling and filters

TCP New Reno: Additive Increase

- Remember the recent past to find a good estimate of link rate
- The last good `cwnd` without packet drop is a good indicator
  - TCP New Reno calls this the slow start threshold (`ssthresh`)
- Increase `cwnd` by 1 MSS every RTT after `cwnd` hits `ssthresh`
  - Effect: increase window additively per RTT



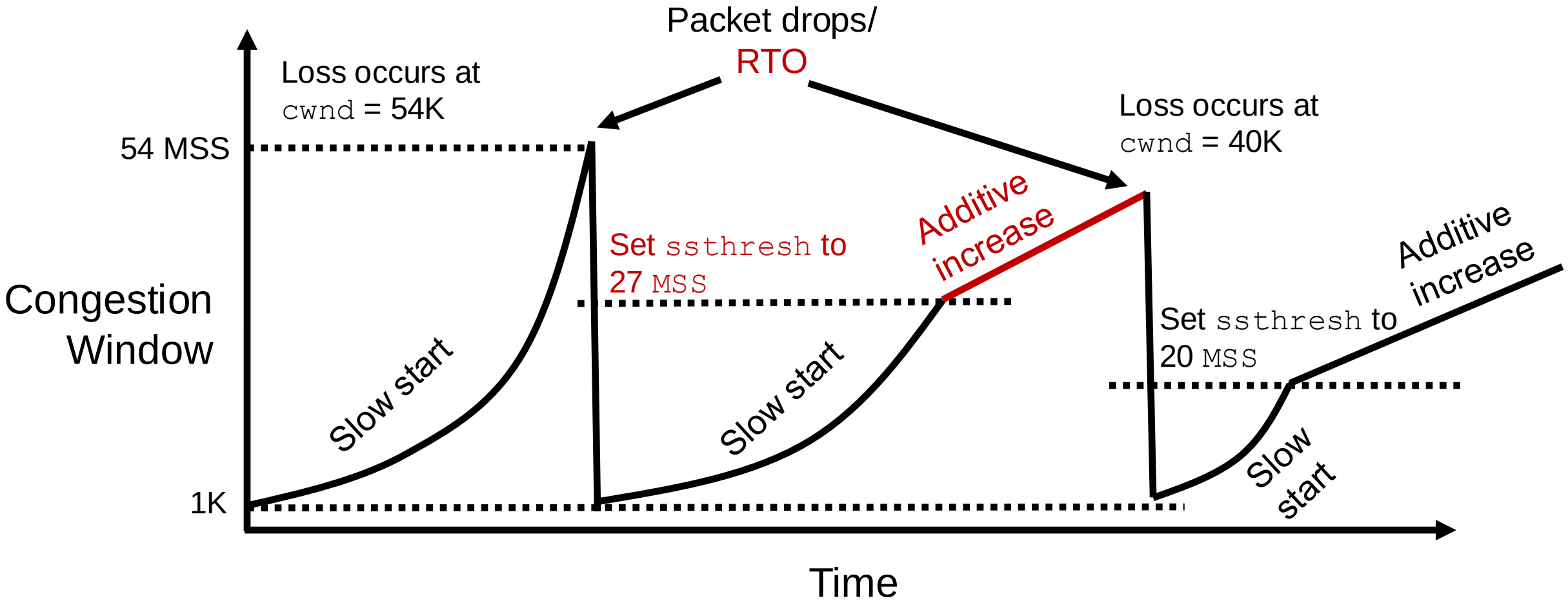
TCP New Reno: Additive increase

- Start with `ssthresh = 64K bytes` (TCP default)
- Do slow start until `ssthresh`
- Once the threshold is passed, do additive increase
  - Add one MSS to `cwnd` for each `cwnd` worth data ACK'ed
  - For each MSS ACK'ed, `cwnd = cwnd + (MSS/cwnd) * MSS`
- Upon a TCP timeout (RTO),
  - Set `cwnd = 1 MSS`
  - Set `ssthresh = max(2 * MSS, 0.5 * cwnd)`
  - i.e., the next linear increase will start at half the current `cwnd`

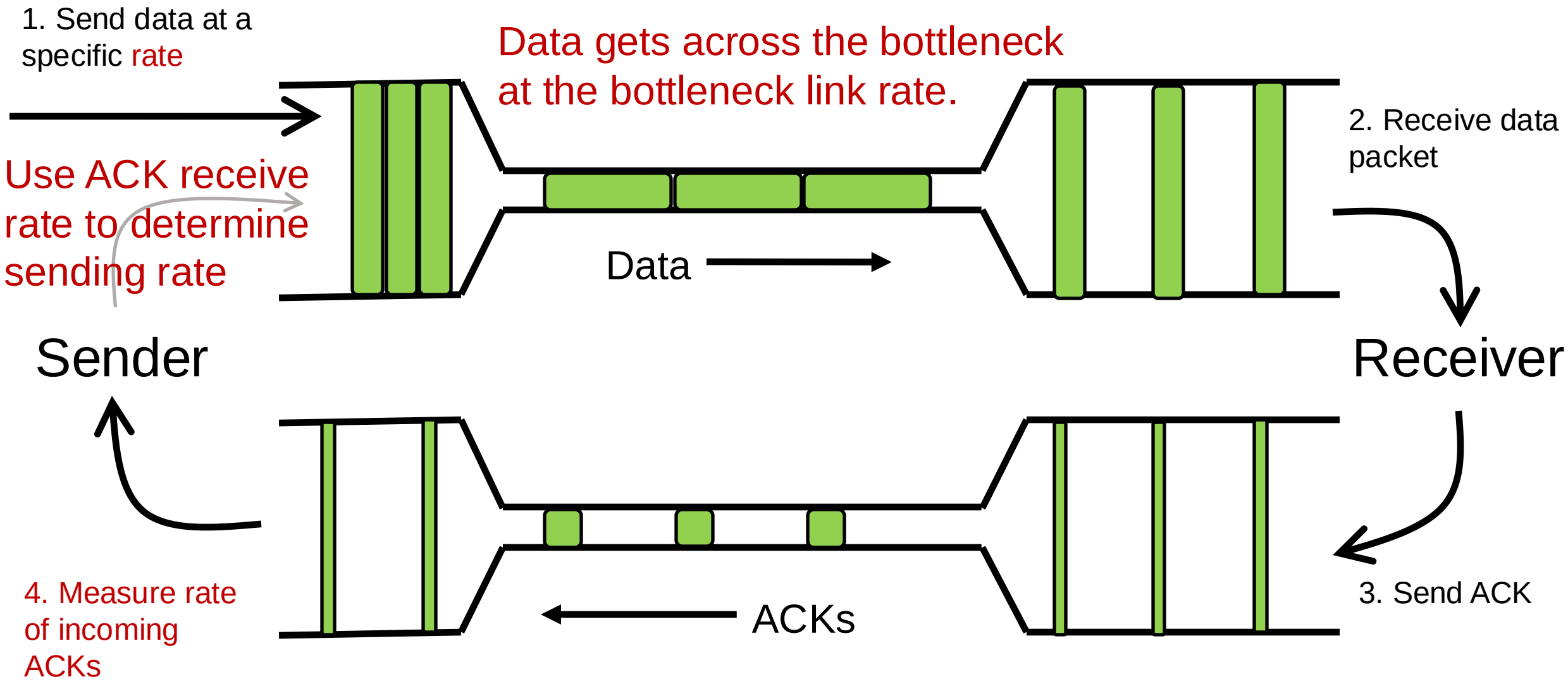
Behavior of Additive Increase

Say `MSS` = 1 KByte

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



TCP BBR: finding the bottleneck link rate



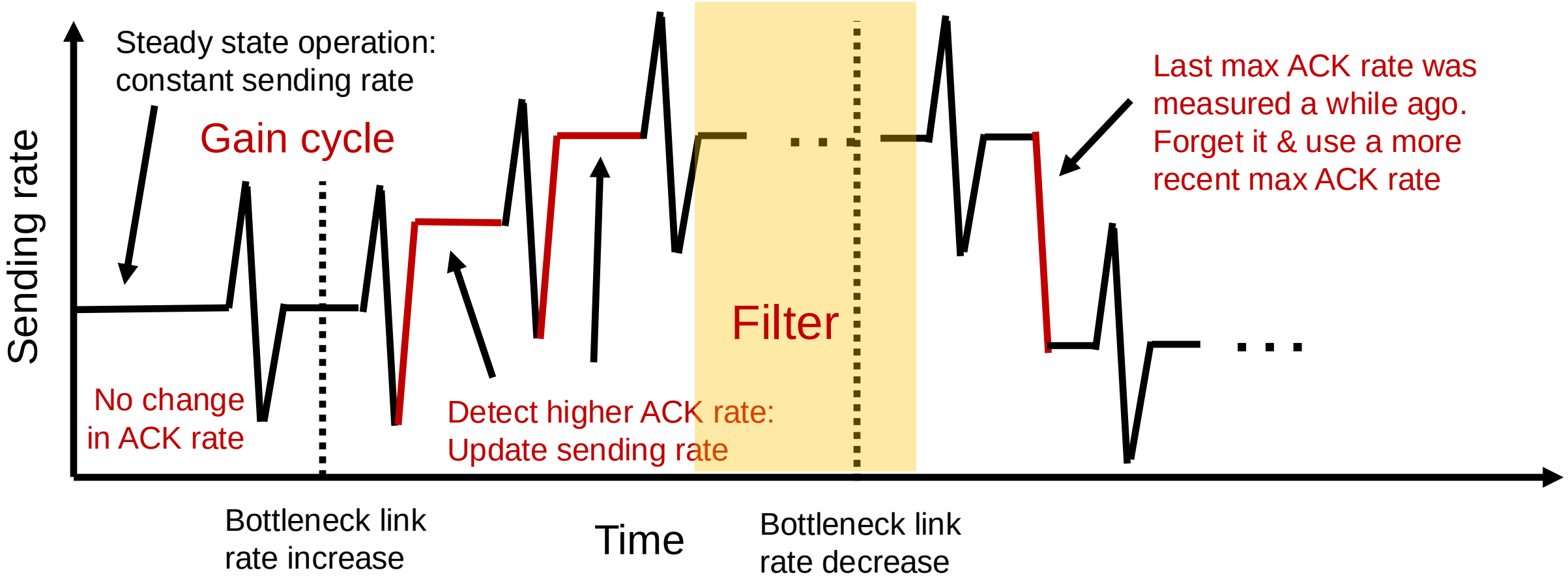
TCP BBR: finding the bottleneck link rate

- 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; back-to-back packets over bottleneck link

- 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 pipes”