数据链路流量控制 - 重温
(滑动窗口)

- 这种技术允许 *多* 帧是在传输过程中同时
- 两个站使用扩展缓冲器大小以保持多帧
- 发送/接收车站维持帧列表已经发送/接收的
- 这种技术允许多 更多 *高效的链路利用率*
- 所述传输链路被有效地视为一个
  管道 这可 *填充* 在中转许多帧
  同时

滑动窗口流量控制

- 站A和B 每 分配缓冲器空间 *w* ^

  帧

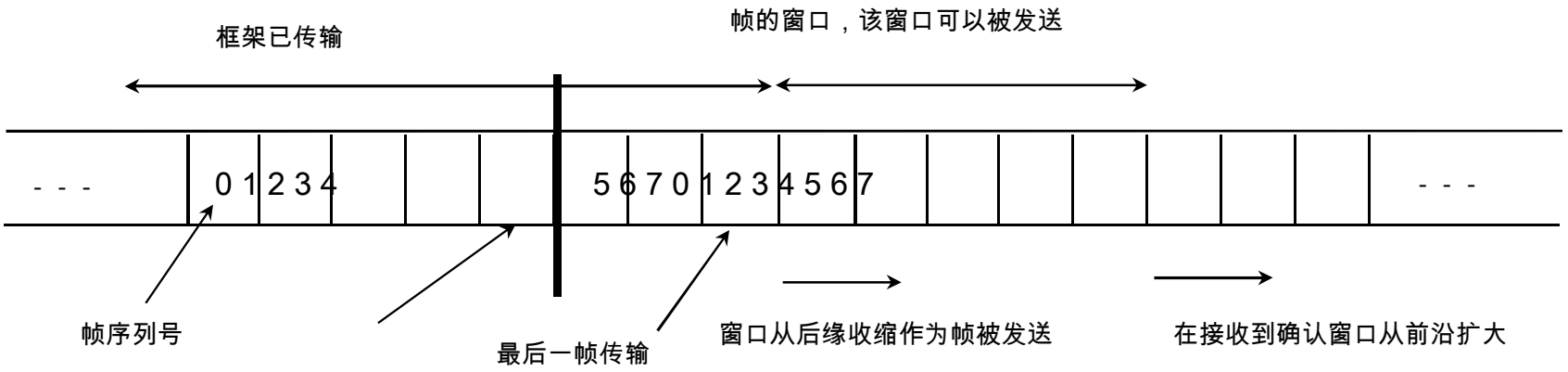
  - 即站B可 *接受w* ^帧和站A可以 *发送W*¯帧 无 任何确认正在发送或接收

- 每一帧包含 *序列号*
- 站B发送 *确认* 包含的序列号 *下一个* 框架预期

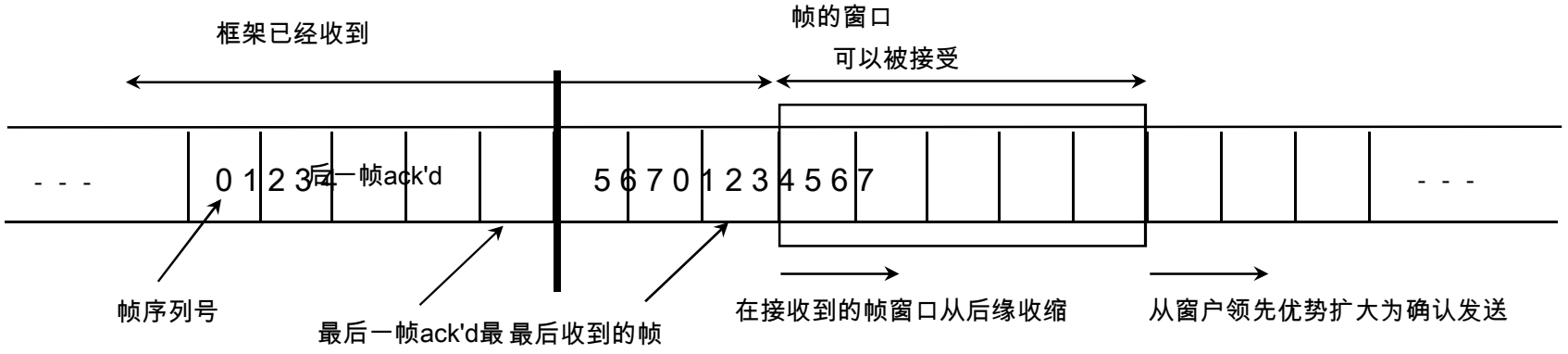
  - 即站B是准备接收 下一个 *w* ^帧

    开始 在 *序列号* 例如表示RR5

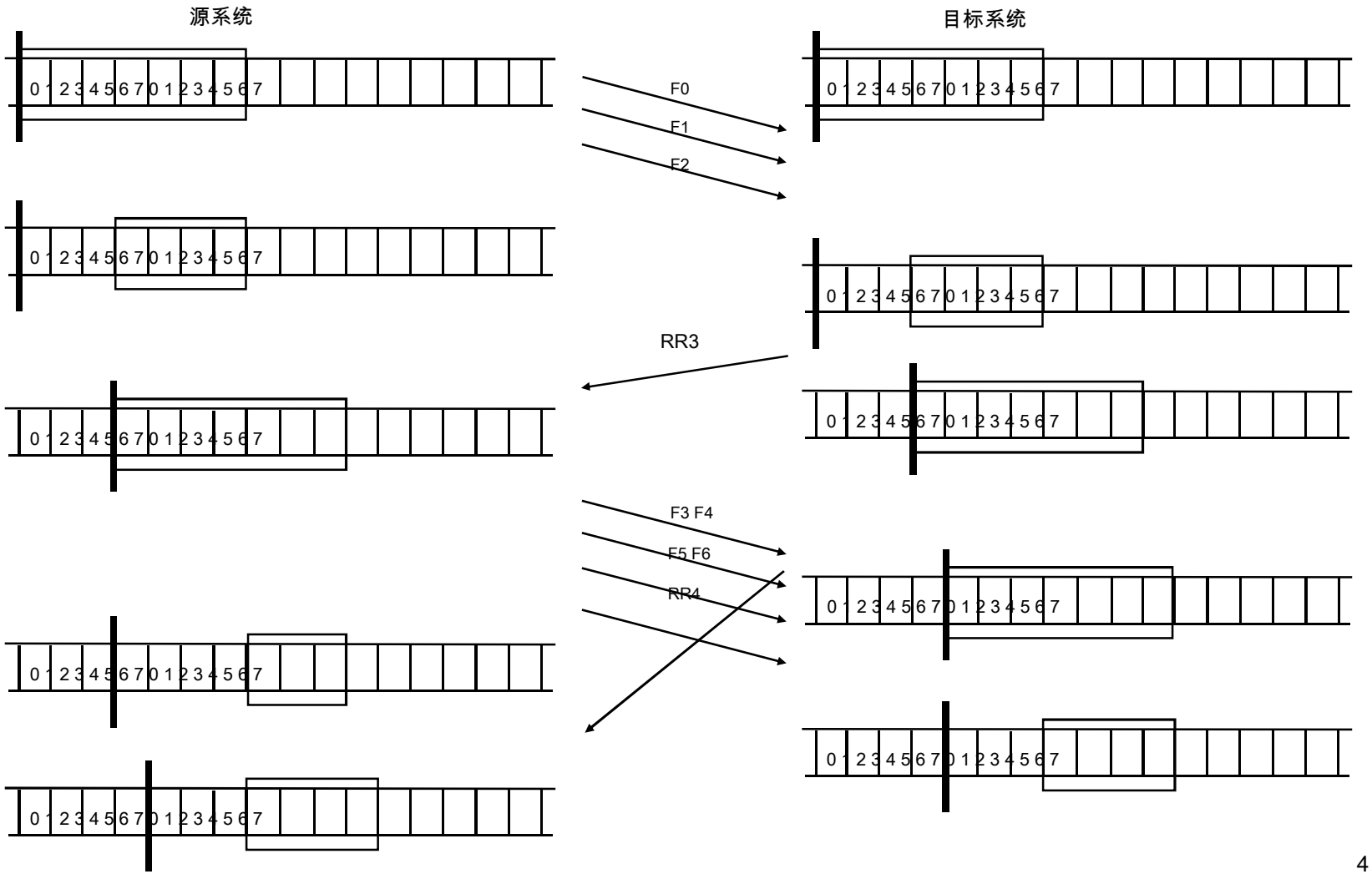
滑动窗口流量控制



接收器的角度看变送器



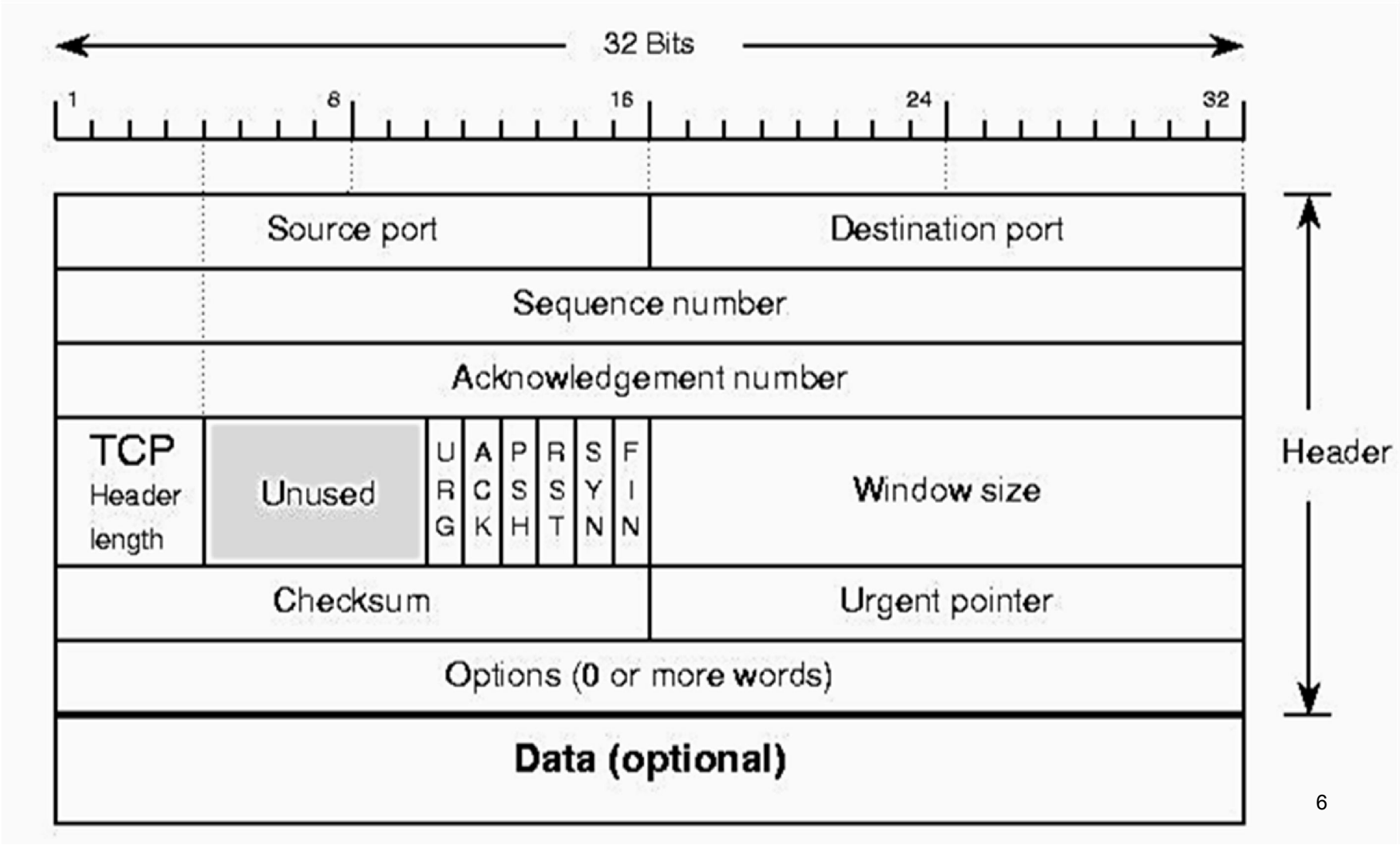
例如推拉窗



滑动窗口流量控制

- *多* 框架可以 *承认* 使用单个控制消息 (*隐含确认*)
  - 例如，ACK的接收帧 **2 (** RR3) 以后，接着ACK帧 **5 (** RR6) *暗示*
    帧的确认 **3** 和 **4**
- A站维护它允许帧编号列表 *发送*
- B站保持架号的列表，它准备接收
- 这些列表可以被视为 *视窗*

TCP段格式


32 Bits
1
8
16
24
32
Source port
Destination port
Sequence number
Acknowledgement number
TCP Header length
Unused
URG
ACK
PSH
RST
SYN
FIN
Window size
Checksum
Urgent pointer
Options (0 or more words)
Data (optional)
Header

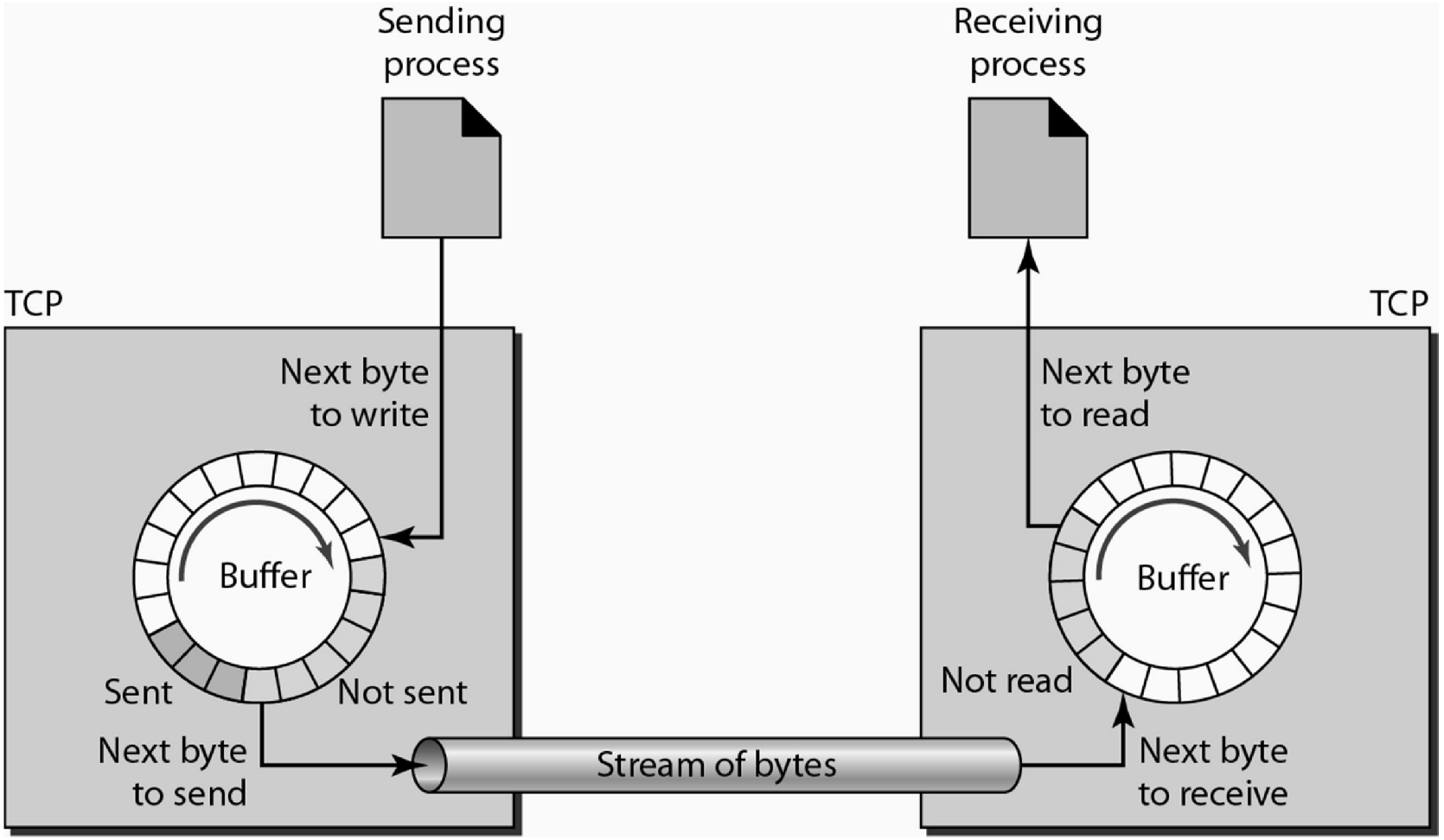
TCP流量控制 - *缓冲器* 和 *视窗*

- 回想一下，TCP创建每个插座两个缓冲区：
  - 一个用于传入数据（RECVQ）
  - 一个用于传出数据（的SendQ）
- 传入的缓冲区可以很容易溢出
- 为了防止这种情况的接收TCP实体使用 ***窗口机制***

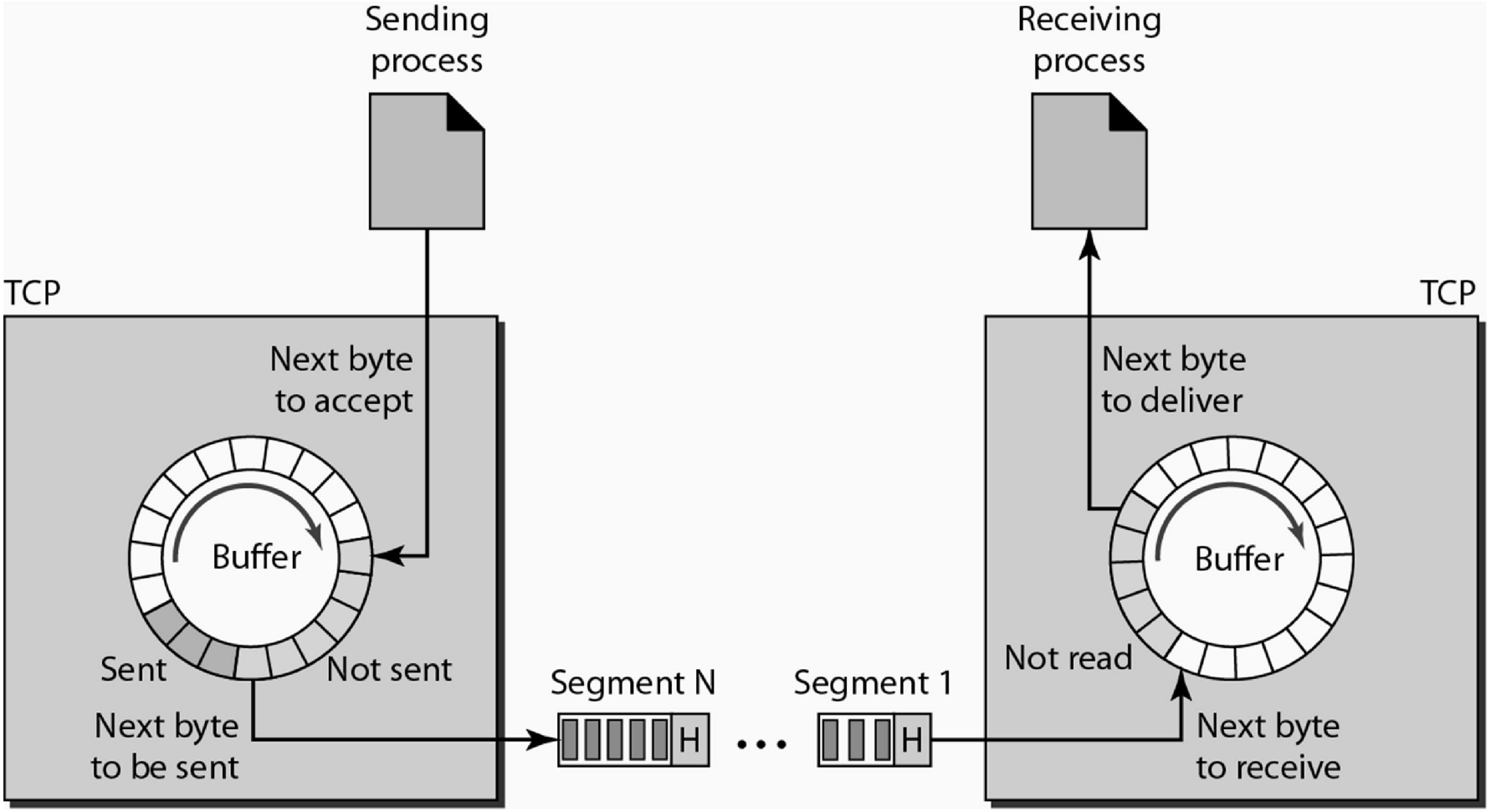
量控制 - *缓冲器 和 视窗*

- 连接的每一端分配一个 ***窗口*** 持有传入的数据：
  - 的大小 初始 窗口期间设置 *阶段1中，连接初始化，* 当双方发送SYN的消息 (使用 *窗口大小* 领域)
  - 此后，在整个 *第2阶段，数据交换，* 所有 ***确认*** 消息包括一个 ***窗口广告***
- 该 *窗口广告* 可以是正数或零取决于RECVQ空间可用性。 TCP流

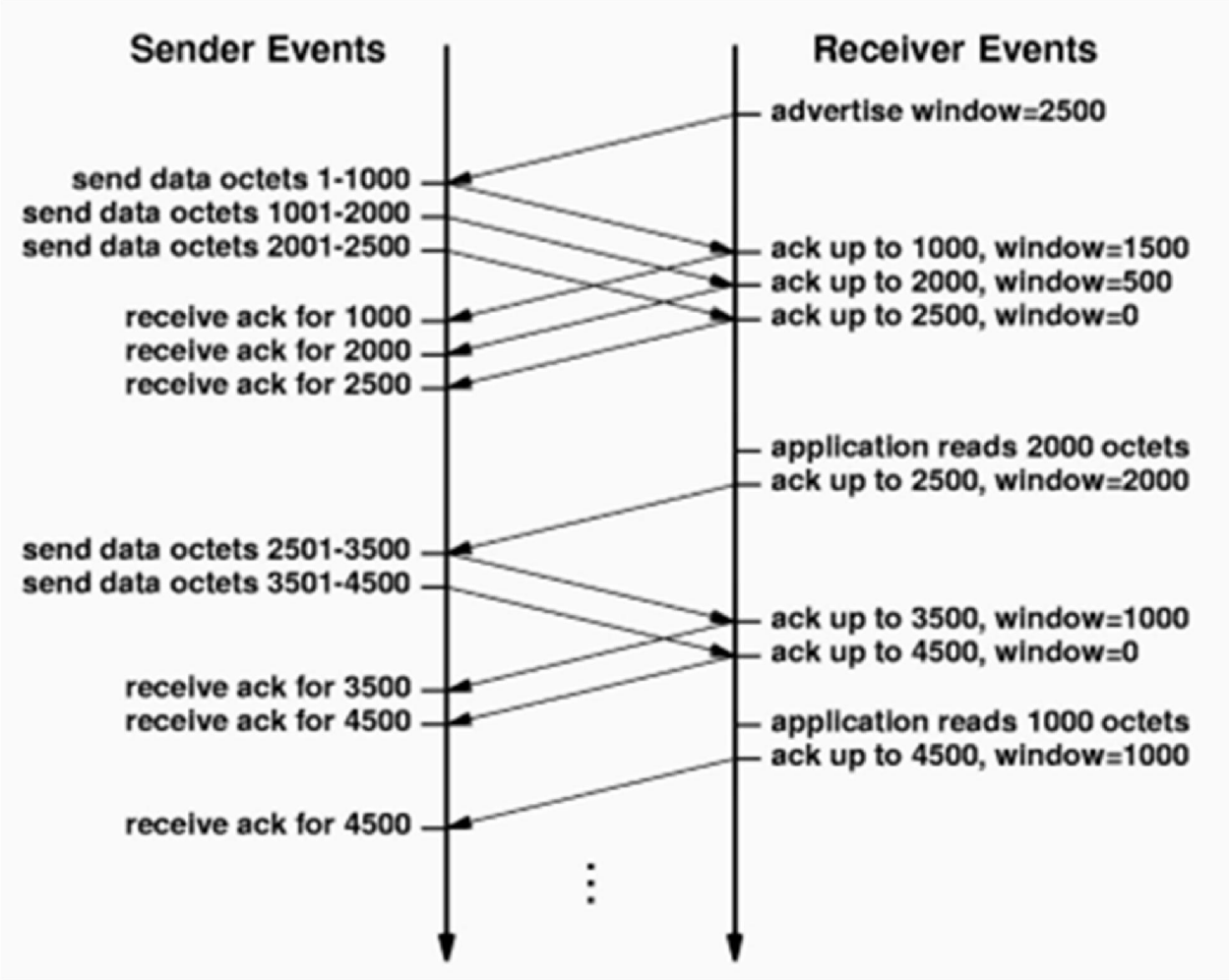
TCP内部数据缓冲器


Sending process
Receiving process
TCP
TCP
Next byte to write
Next byte to read
Buffer
Buffer
Sent
Not sent
Not read
Next byte to send
Stream of bytes
Next byte to receive

TCP内部数据缓冲器


Sending process
Receiving process
TCP
TCP
Next byte to accept
Next byte to deliver
Buffer
Buffer
Sent
Not sent
Not read
Segment N
Segment 1
H
H
Next byte to be sent
Next byte to receive

操作 *窗口公告*


Sender Events
Receiver Events
advertise window=2500
send data octets 1-1000
send data octets 1001-2000
send data octets 2001-2500
ack up to 1000, window=1500
ack up to 2000, window=500
ack up to 2500, window=0
receive ack for 1000
receive ack for 2000
receive ack for 2500
application reads 2000 octets
ack up to 2500, window=2000
send data octets 2501-3500
send data octets 3501-4500
ack up to 3500, window=1000
ack up to 4500, window=0
receive ack for 3500
receive ack for 4500
application reads 1000 octets
ack up to 4500, window=1000
receive ack for 4500

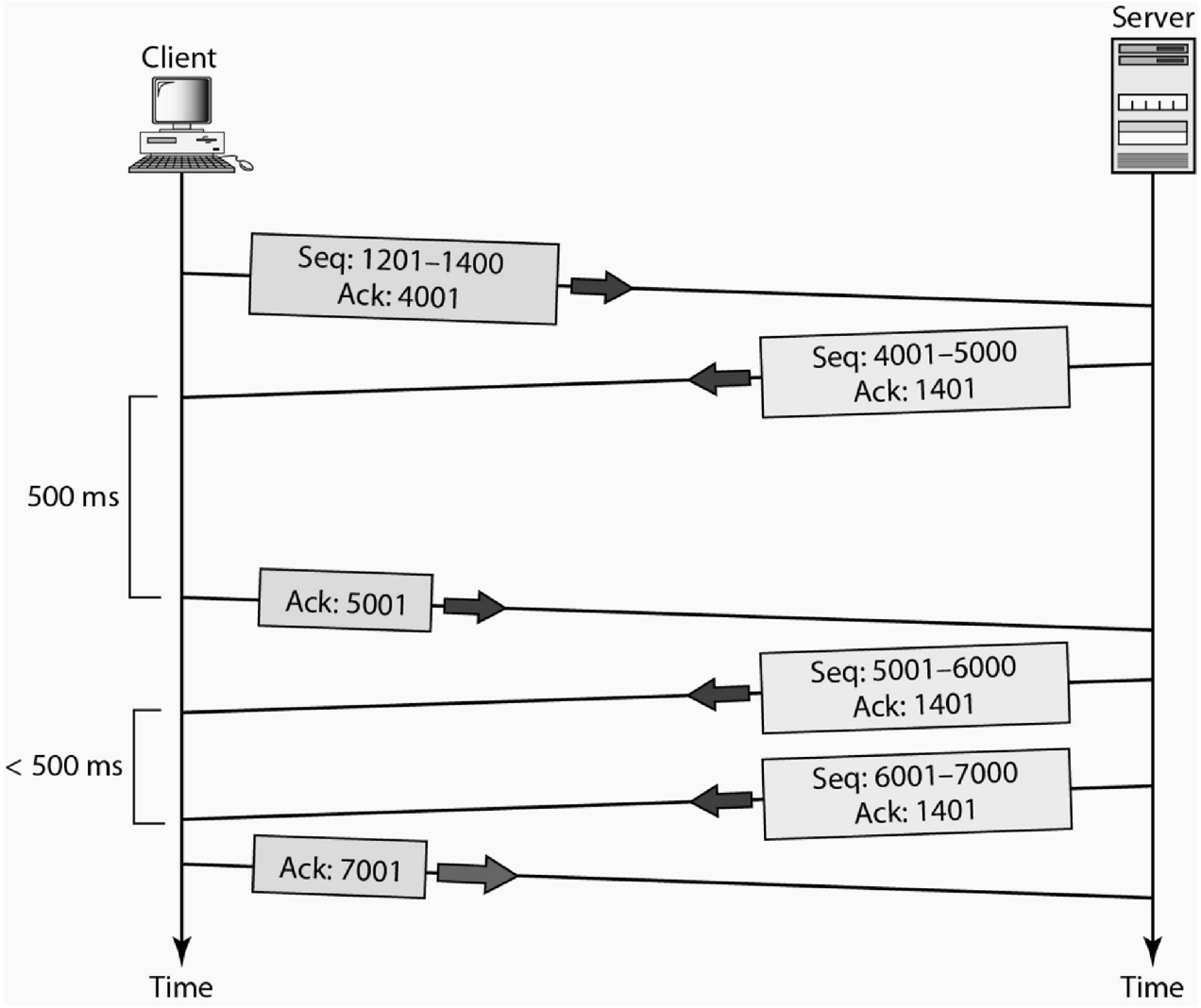
错误控制 - 实现可靠性

- TCP必须解决以下可靠性问题：
  - 靠不住 输送由基础通信系统
    - 段可以被丢失，重复延迟，或通过底层通信系统无序传递
  - 电脑重启
    - 从以前的连接段可以到达。接收主机刚刚重新启动会不知道该怎么跟他们做
- 为了克服这些问题，TCP必须：
  - 确保所有消息 明确的
  - 能够处理 *丢失的数据包*

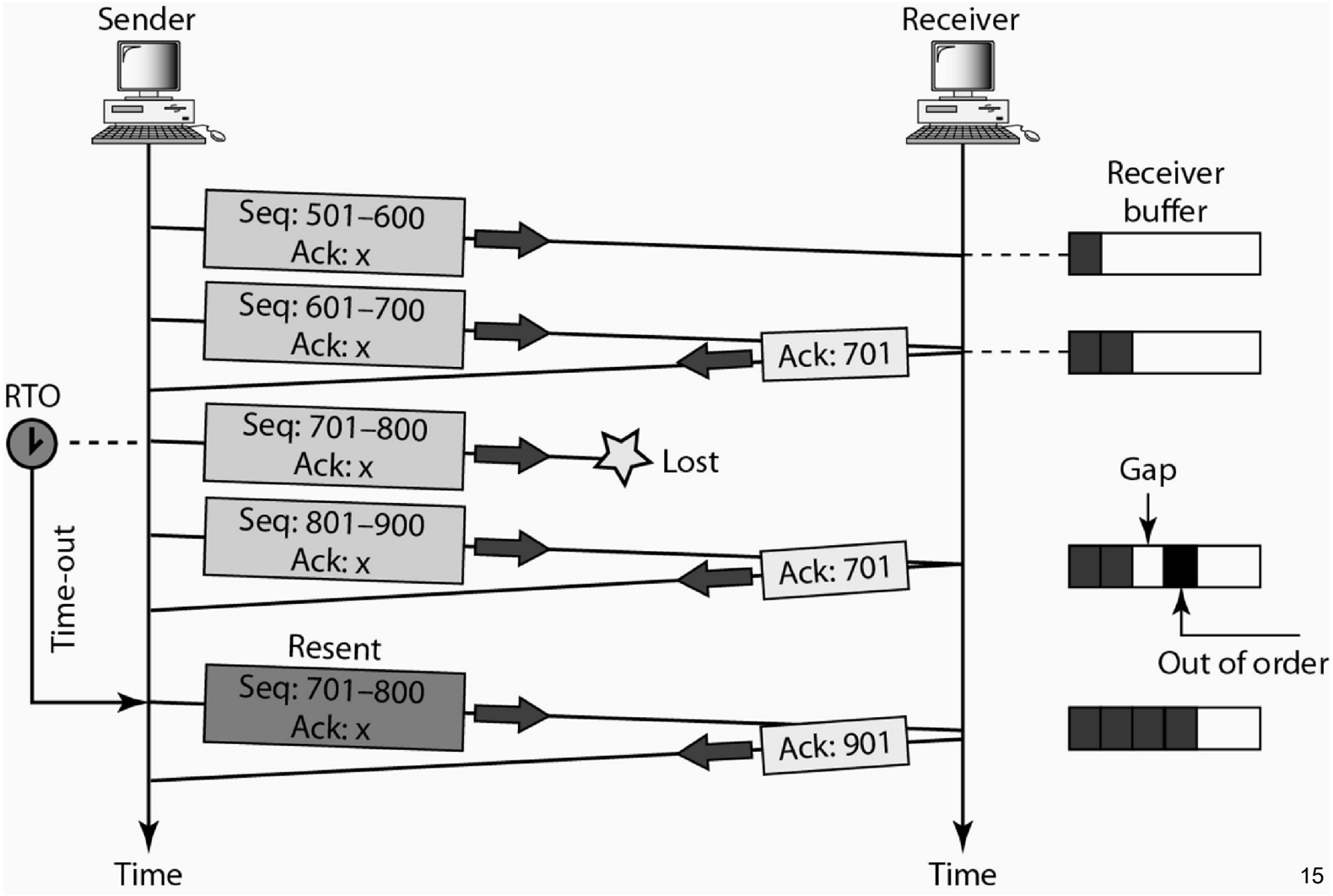
- TCP实现 *重发方案*
- 这包括发送 *确认* 和使用 *计时器*

- 该方案的关键在于TCP的成功
- 重传多久之前，应等待TCP？
  - 它取决于：
    - 底层网络
    - 流量水平
- TCP使用的 *自适应重传* 方案处理数据包丢失 - 重发

TCP数据流 - 正常操作


Client
Server
Seq: 1201–1400
Ack: 4001
Seq: 4001–5000
Ack: 1401
500 ms
Ack: 5001
Seq: 5001–6000
Ack: 1401
< 500 ms
Seq: 6001–7000
Ack: 1401
Ack: 7001
Time
Time

TCP数据流 - 失落段方案



- TCP估计 *往返延迟* 对于每个活动连接

- 用一个加权*平均* 连同一
  *应重传*方案
  方差 因子TCP计算用于一个值
  重发时间

- 这有助于TCP快速应对变化
  流量水平 并最大限度地提高每个TCP连接的吞吐量 *自适*