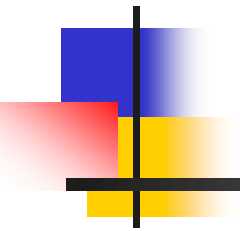


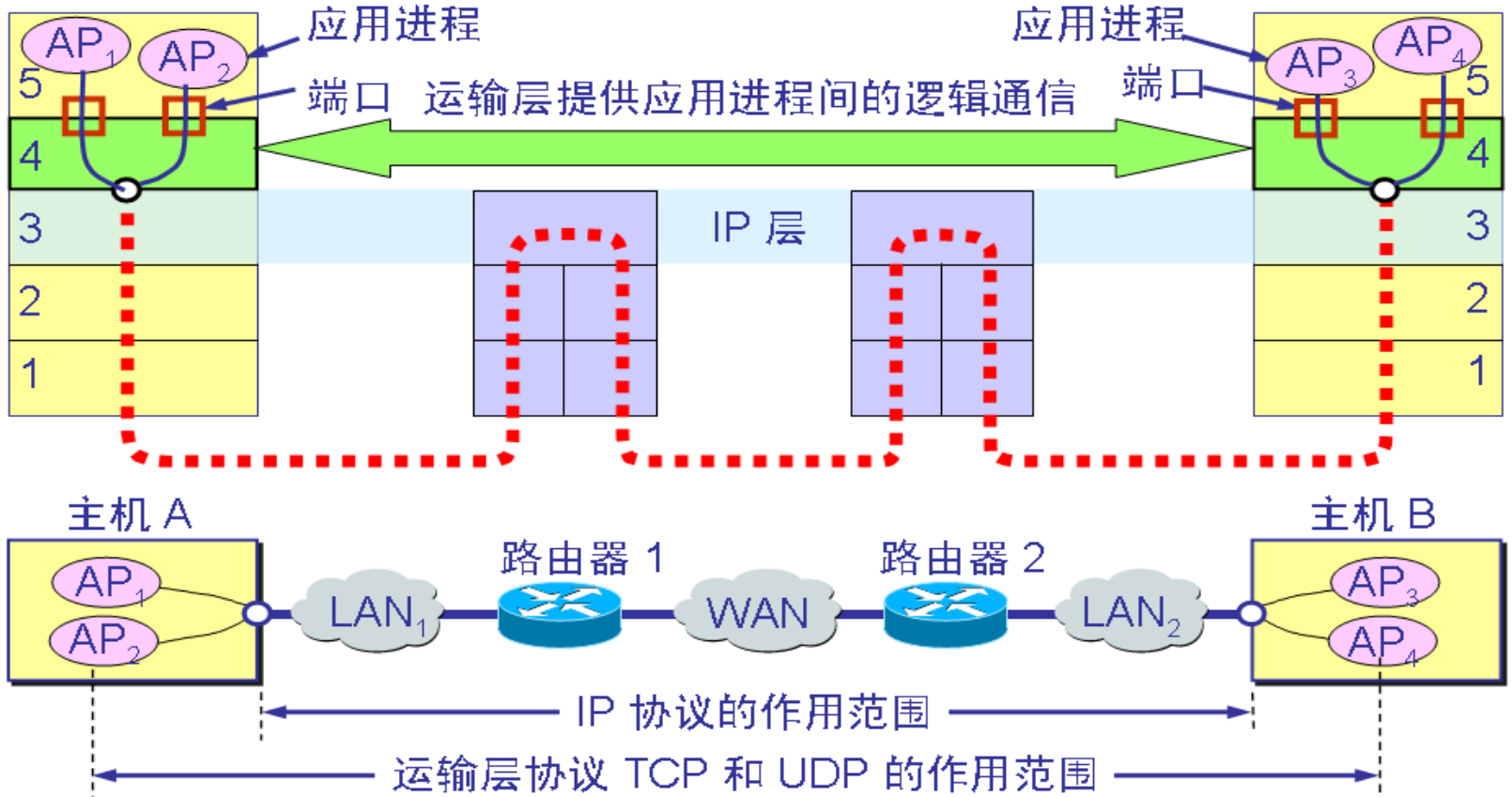
第 5 章　运输层

第 5 章 运输层

5.1 运输层协议概述
5.2 用户数据报协议 UDP
5.3 传输控制协议 TCP 概述
5.4 可靠传输的工作原理
5.5 TCP 报文段的首部格式
5.6 TCP 可靠传输的实现
5.7 TCP的流量控制
5.8 TCP 的拥塞控制
5.9 TCP 的运输连接管理

5.1 运输层协议概述
5.1.1 进程之间的通信

运输层为相互通信的应用进程提供了逻辑通信



应用进程之间的通信

- 两个主机进行通信实际上就是两个主机中的应用进程互相通信。
- 应用进程之间的通信又称为端到端的通信。

运输层的主要功能

- 运输层的一个很重要的功能就是复用和分用。
  - 复用:应用层不同的应用进程使用一种运输层协议传送数据。
  - 分用:接收方的运输层在剥去报文的首部后将数据正确交付到目的进程。
- 运输层为应用进程之间提供端到端的逻辑通信。
- 运输层还要对收到的报文进行差错检测。

5.1.2 运输层的两个主要协议

TCP/IP 的运输层有两个不同的协议：

(1) 用户数据报协议 UDP
(User Datagram Protocol)

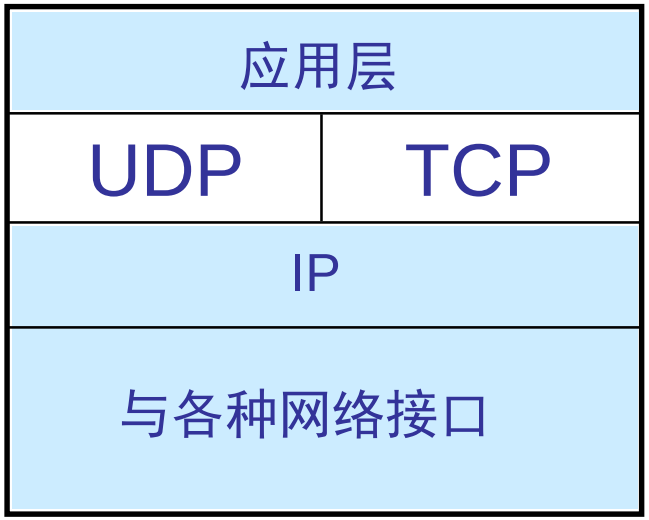
(2) 传输控制协议 TCP
(Transmission Control Protocol)

TCP 与 UDP

- TCP 传送的数据单位协议是 TCP 报文段(segment)
- UDP 传送的数据单位协议是 UDP 用户数据报。

TCP/IP 体系中的运输层协议

运输层



TCP 与 UDP

- UDP 在传送数据之前不需要先建立连接。对方的运输层在收到 UDP 报文后，不需要给出任何确认。虽然 UDP 不提供可靠交付，但在某些情况下 UDP 是一种最有效的工作方式。

TCP 与 UDP

- TCP 则提供面向连接的服务。由于 TCP 要提供可靠的、面向连接的运输服务，因此不可避免地增加了许多的开销。

5.1.3 运输层的端口

- 端口用一个 16 位端口号进行标志。
- 端口号只具有本地意义，即端口号只是为了标志本计算机应用层中的各进程。
- 两个计算机机中的进程要互相通信，不仅知道对方的IP地址，还必须包括对方的端口号。

二类端口

- 服务器端使用的端口号
  - 熟知端口，数值一般为 0~1023。这些端口指派给最重要的一些应用程序。
  - 登记端口号，数值为1024~49151，为没有熟知端口号的应用程序使用的。

应用程序	FTP	TELNET	SMTP	DNS	TFTP	HTTP	SNMP	SNMP(trap)
熟知端口	21	23	25	53	69	80	161	162

二类端口

- 客户端口号或短暂端口号，数值为49152~65535，留给客户进程选择暂时使用。当服务器进程收到客户进程的报文时，就知道了客户进程所使用的动态端口号。通信结束后，这个端口号可供其他客户进程以后使用。

5.2 用户数据报协议 UDP
5.2.1 UDP 概述

- UDP 只在 IP 的数据报服务之上增加了很少一点的功能，即复用和分用（端口）的功能和差错检测的功能。

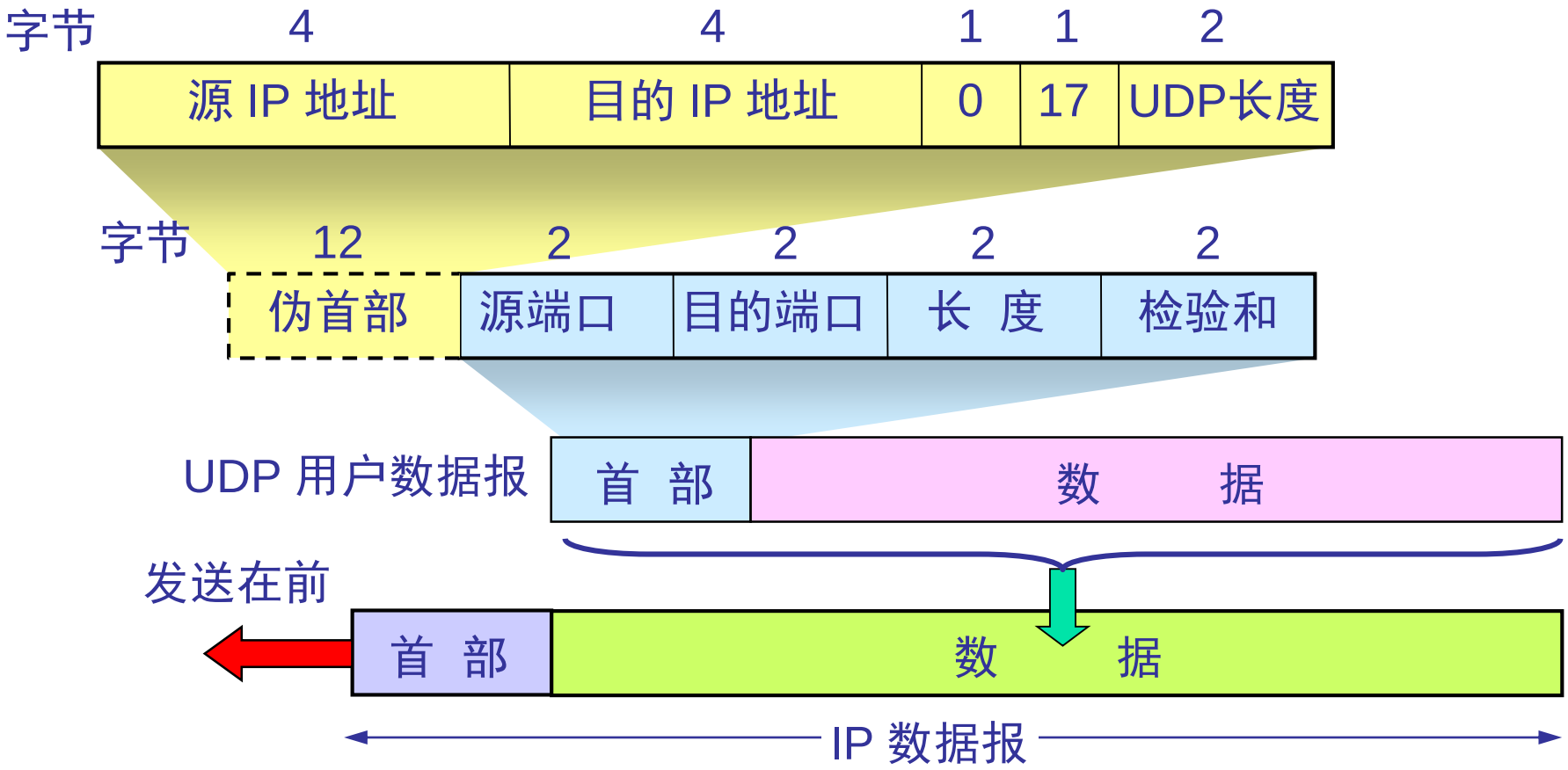
UDP 的主要特点 -1

- UDP 是无连接的，即发送数据之前不需要建立连接。
- UDP 使用尽最大努力交付，即不保证可靠交付。
- UDP 是面向报文的。 UDP 对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。

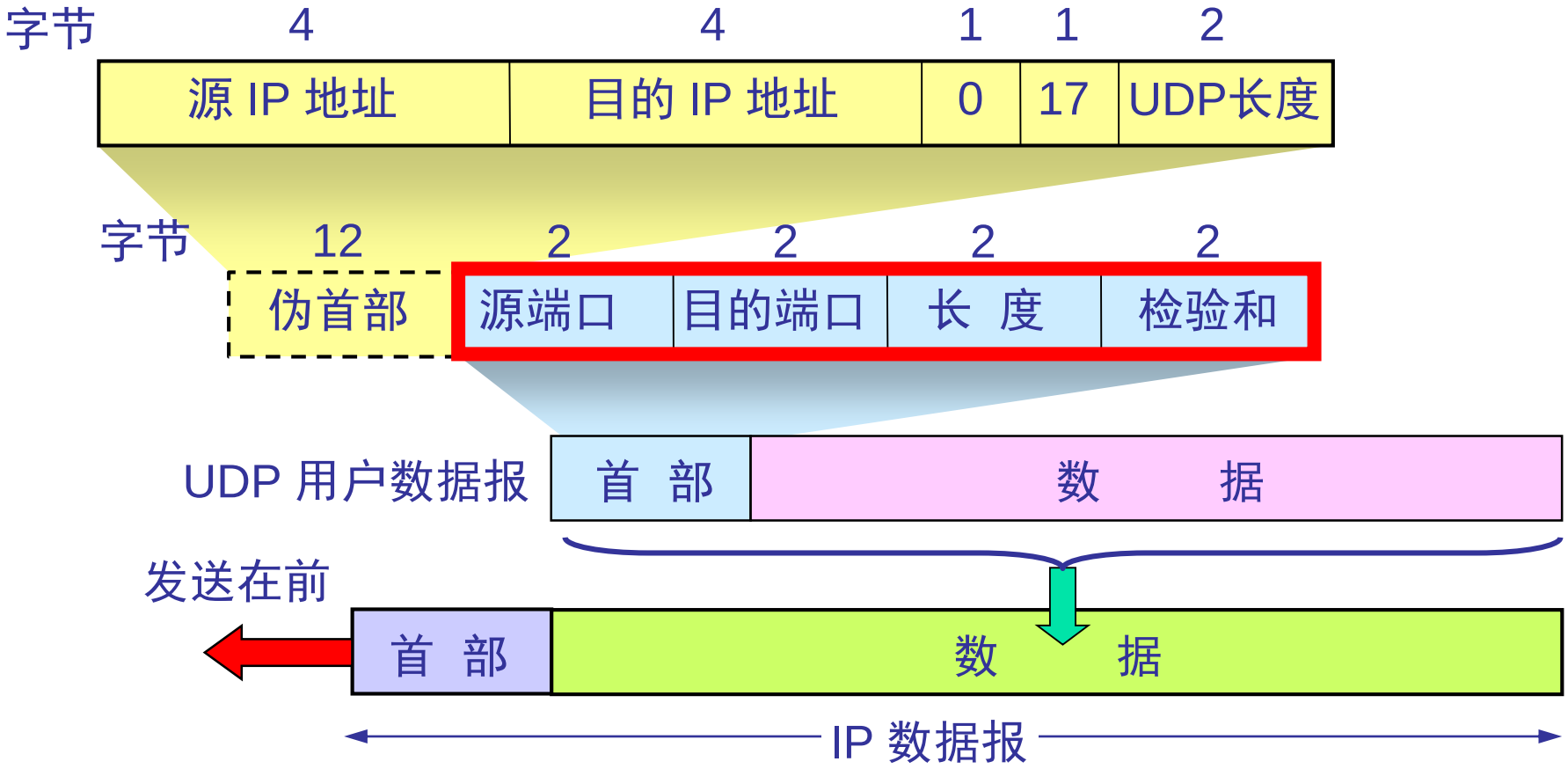
UDP 的主要特点

- UDP 没有拥塞控制，很适合多媒体通信的要求。
- UDP 支持一对一、一对多、多对一和多对多的交互通信。
- UDP 的首部开销小，只有 8 个字节。

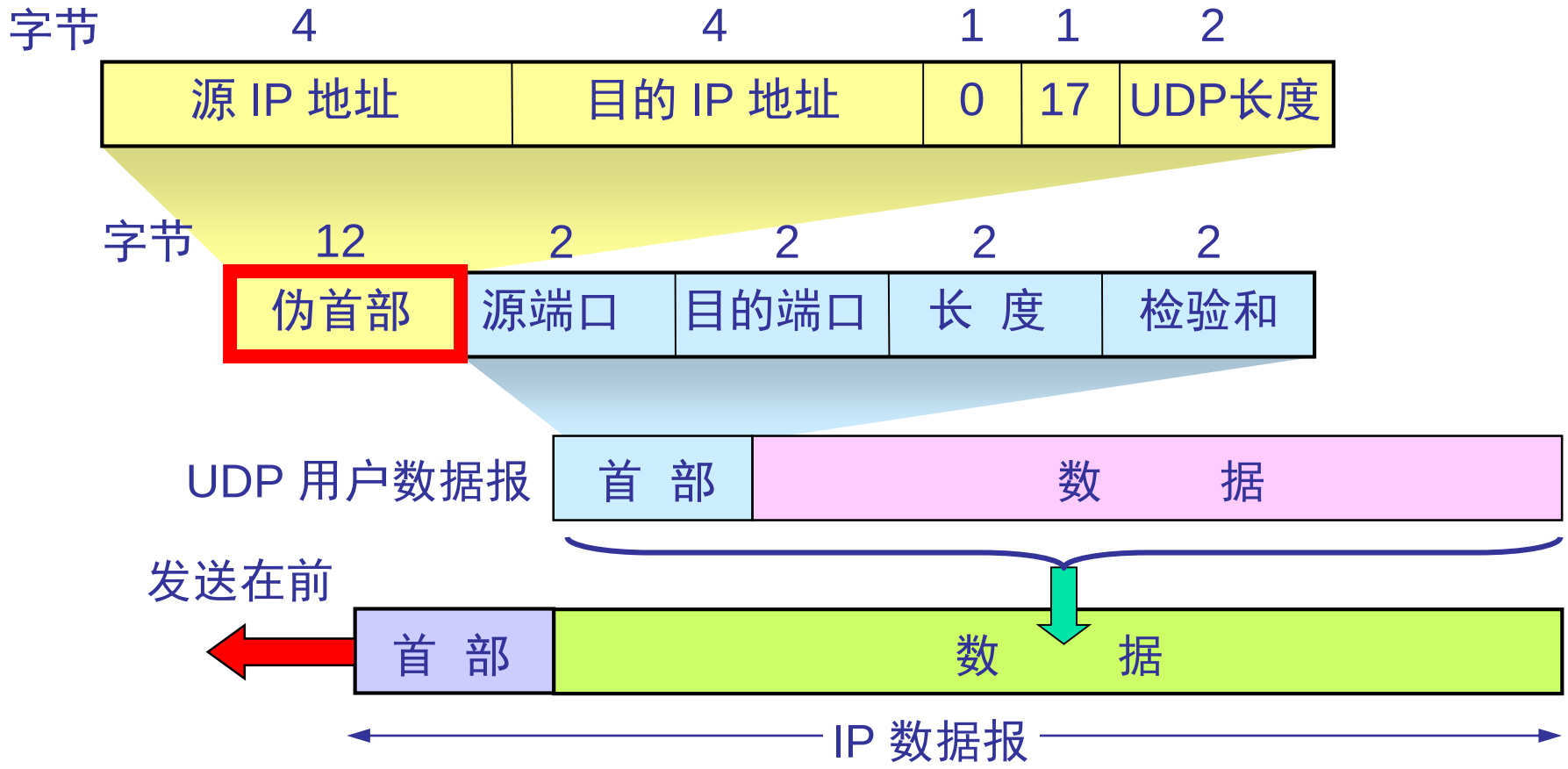
5.2.2 UDP 的首部格式



用户数据报 UDP 有两个字段：数据字段和首部字段。首部字段有 8 个字节，由 4 个字段组成，每个字段都是两个字节。



在计算检验和时，临时把“伪首部”和 UDP 用户数据报连接在一起。伪首部仅仅是为了计算检验和。



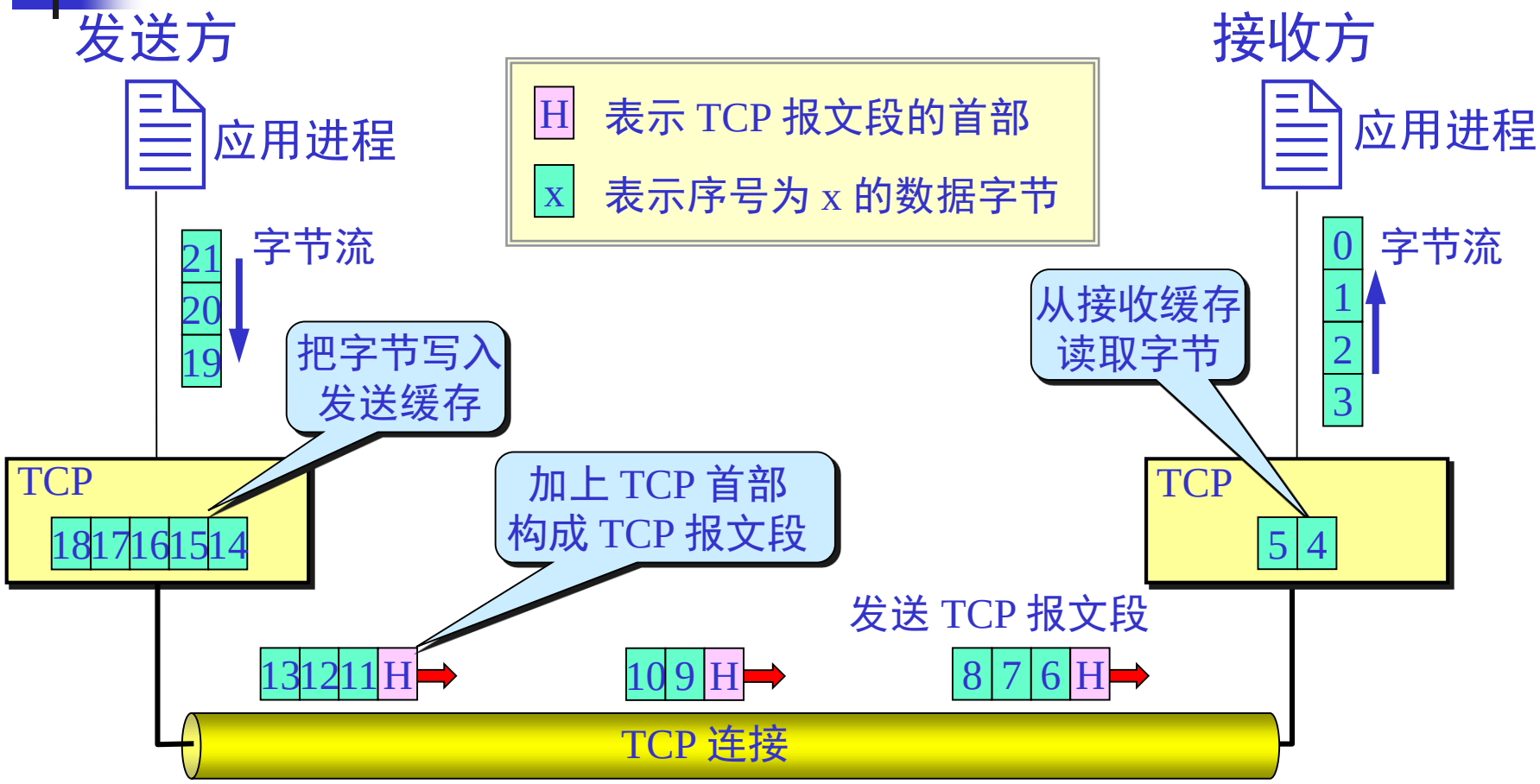
5.3 传输控制协议 TCP 概述
5.3.1 TCP 最主要的特点

- TCP 是面向连接的运输层协议。
- 每一条 TCP 连接只能有两个端点(endpoint)，每一条 TCP 连接只能是点对点的（一对一）。
- TCP 提供可靠交付的服务。数据要求无差错,不丢失,不重复、并且按序到达。

TCP 最主要的特点

- TCP 提供全双工通信。
- 面向字节流。TCP将应用程序交下来的数据看成是一连串的无结构的字节流。

TCP 面向流的概念


发送方
应用进程
字节流
21
20
19
把字节写入
发送缓存
TCP
18 17 16 15 14
H 表示 TCP 报文段的首部
x 表示序号为 x 的数据字节
加上 TCP 首部
构成 TCP 报文段
13 12 11 H
10 9 H
8 7 6 H
发送 TCP 报文段
TCP 连接
接收方
应用进程
字节流
0
1
2
3
从接收缓存
读取字节
TCP
5 4

5.3.2 TCP 的连接

- 每一条 TCP 连接有两个端点。
- TCP 连接的端点叫做套接字(socket)或插口。
- 端口号拼接到(contatenated with) IP 地址即构成了套接字。

套接字 (socket)

套接字 socket = (IP地址: 端口号)

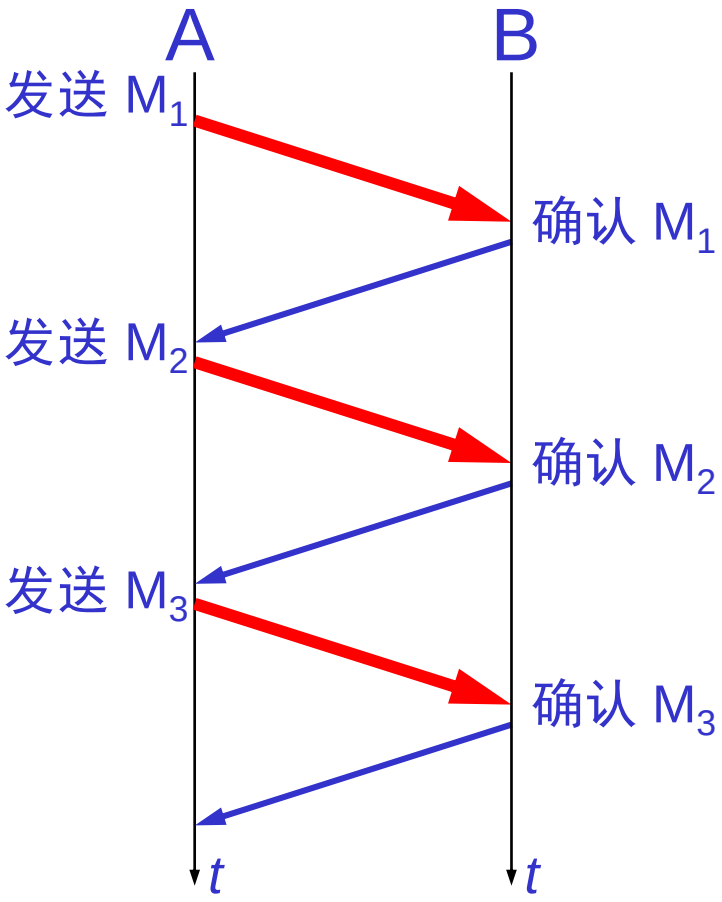
每一条 **TCP** 连接唯一地被通信两端的两个端点（即两个套接字）所确定。

TCP 连接 ::= {socket1, socket2}

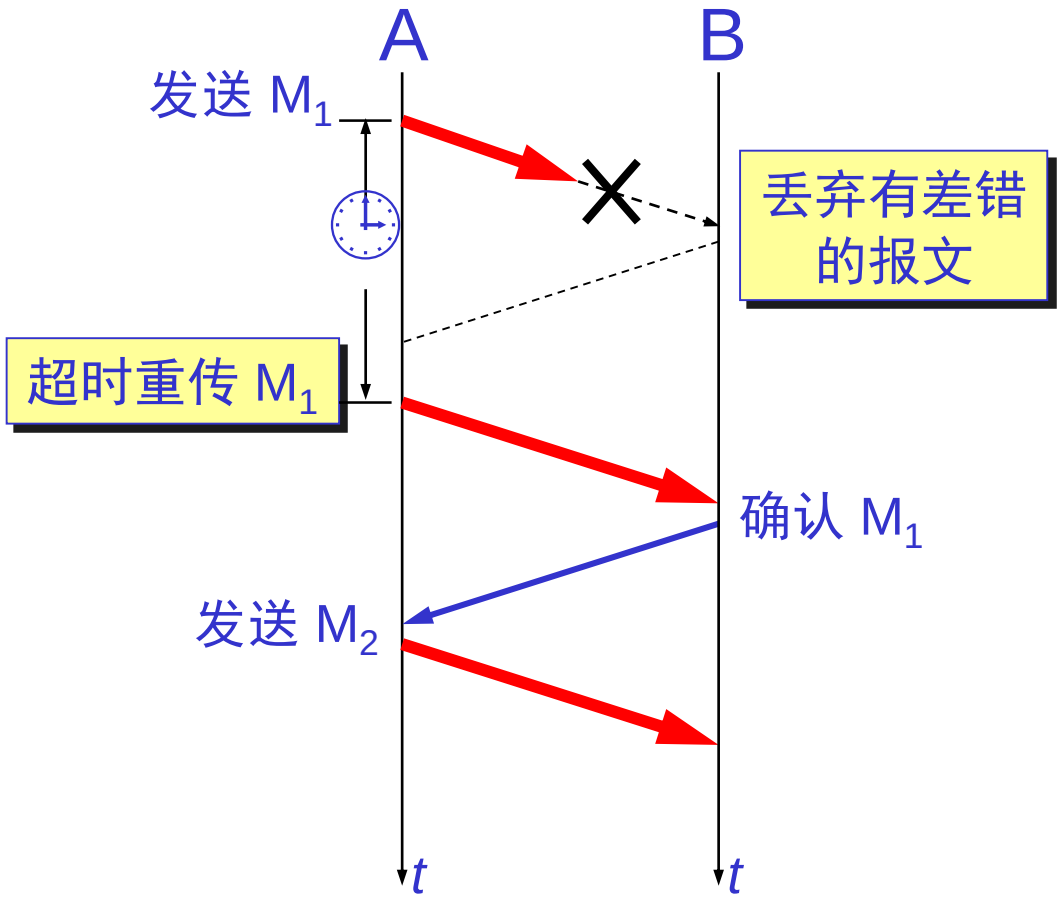
= {(IP1: port1), (IP2: port2)}

5.4 可靠传输的工作原理

5.4.1 停止等待协议



(a) 无差错情况

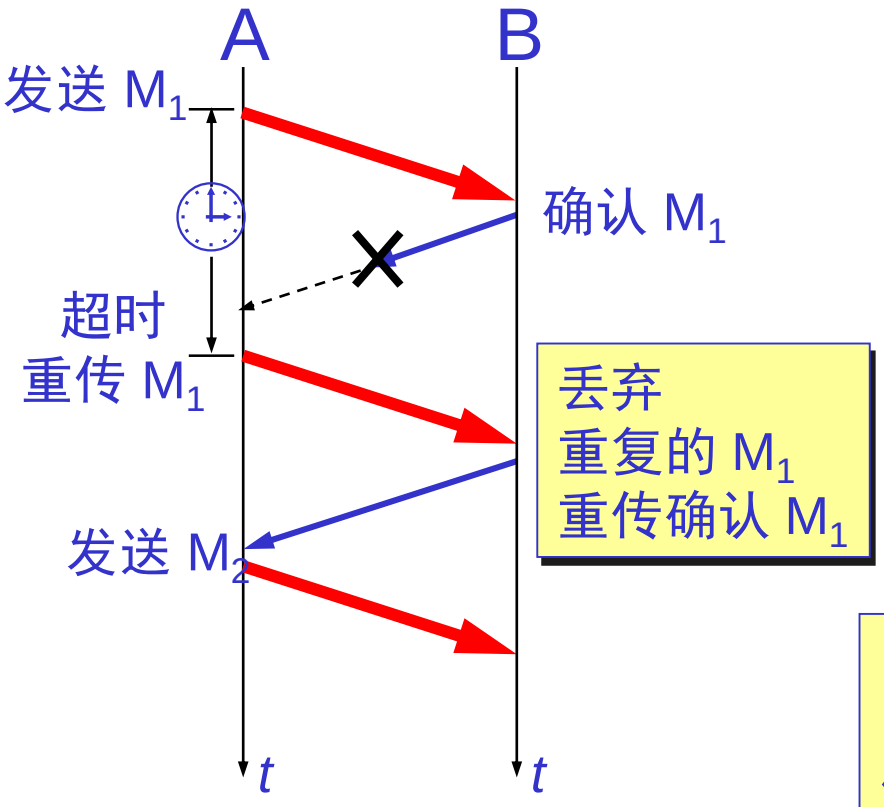


(b) 超时重传

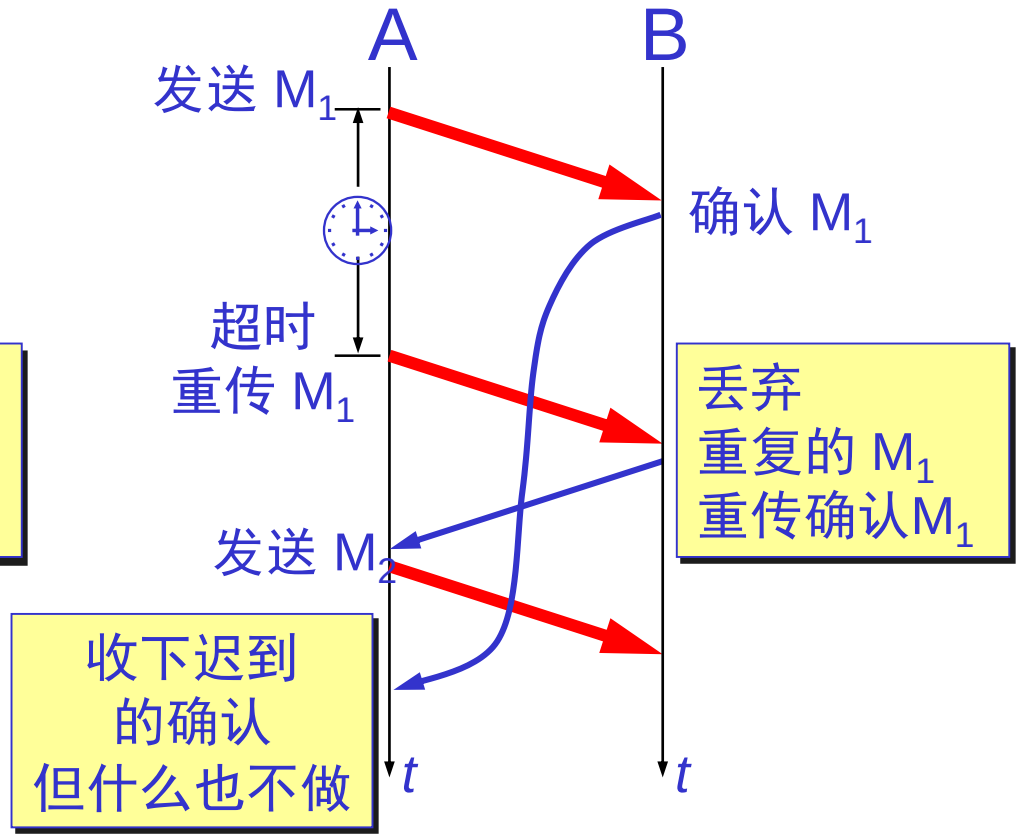
请注意

- 在发送完一个分组后，必须暂时保留已发送的分组的副本。
- 分组和确认分组都必须进行编号。
- 超时计时器的重传时间应当比数据在分组传输的平均往返时间更长一些。

确认丢失和确认迟到



(a) 确认丢失



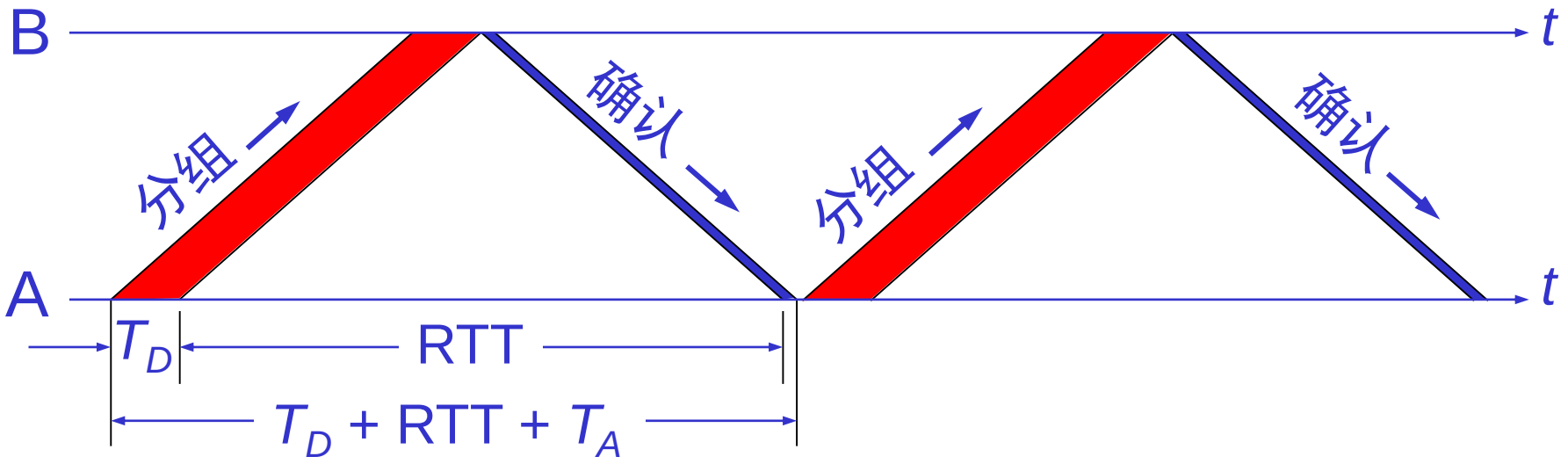
(b) 确认迟到

可靠通信的实现

- 使用上述的**确认**和重传机制，我们就可以在不可靠的传输网络上实现可靠的通信。
- 这种可靠传输协议常称为自动重传请求 ARQ (Automatic Repeat reQuest)。
- ARQ 表明重传的请求是自动进行的。接收方不需要请求发送方重传某个出错的分组 。

信道利用率

- 停止等待协议的优点是简单，但缺点是信道利用率太低。

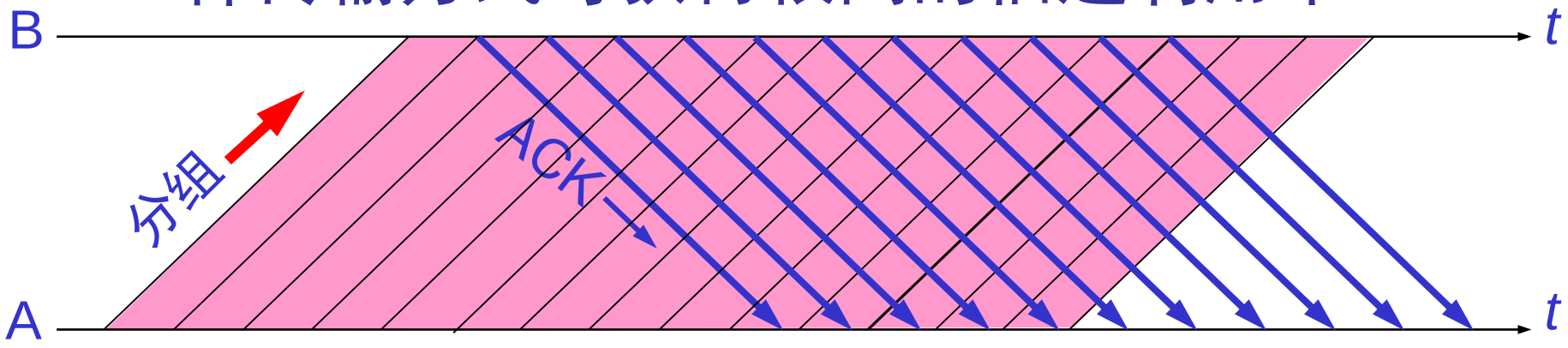


信道的利用率 U

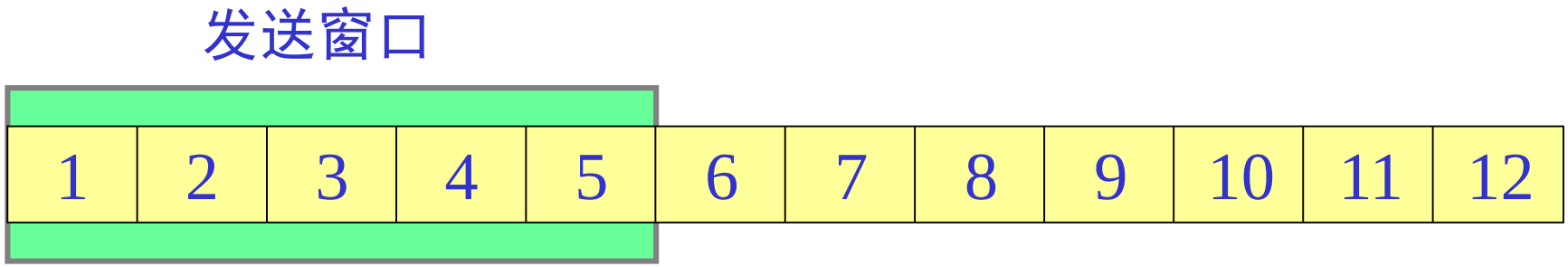
$$U = \frac{T_D}{T_D + \mathrm{RTT} + T_A}$$

流水线传输

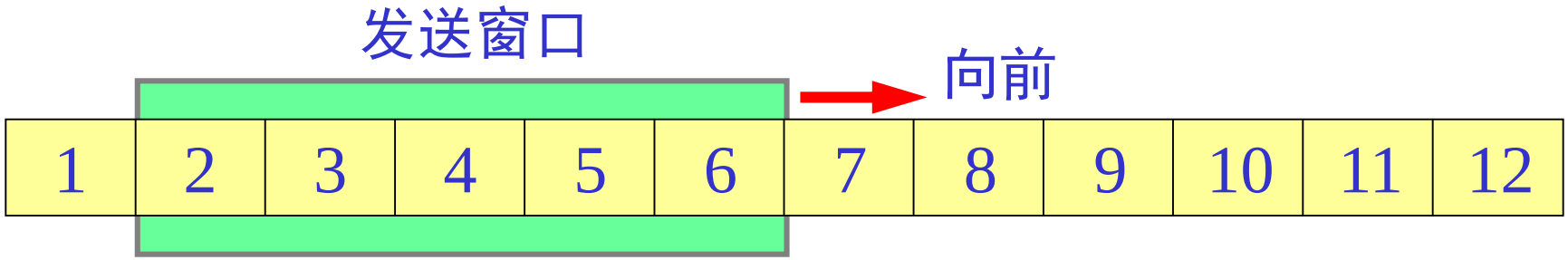
- 发送方可连续发送多个分组，不必每发完一个分组就停顿下来等待对方的确认。
- 由于信道上一直有数据不间断地传送，这种传输方式可获得很高的信道利用率。



5.4.2 连续 ARQ 协议



(a) 发送方维持发送窗口（发送窗口是 5）



(b) 收到一个确认后发送窗口向前滑动

累积确认

- 接收方一般采用累积确认的方式。即不必对收到的分组逐个发送确认，而是对按序到达的最后一个分组发送确认，这样就表示：到这个分组为止的所有分组都已正确收到了。

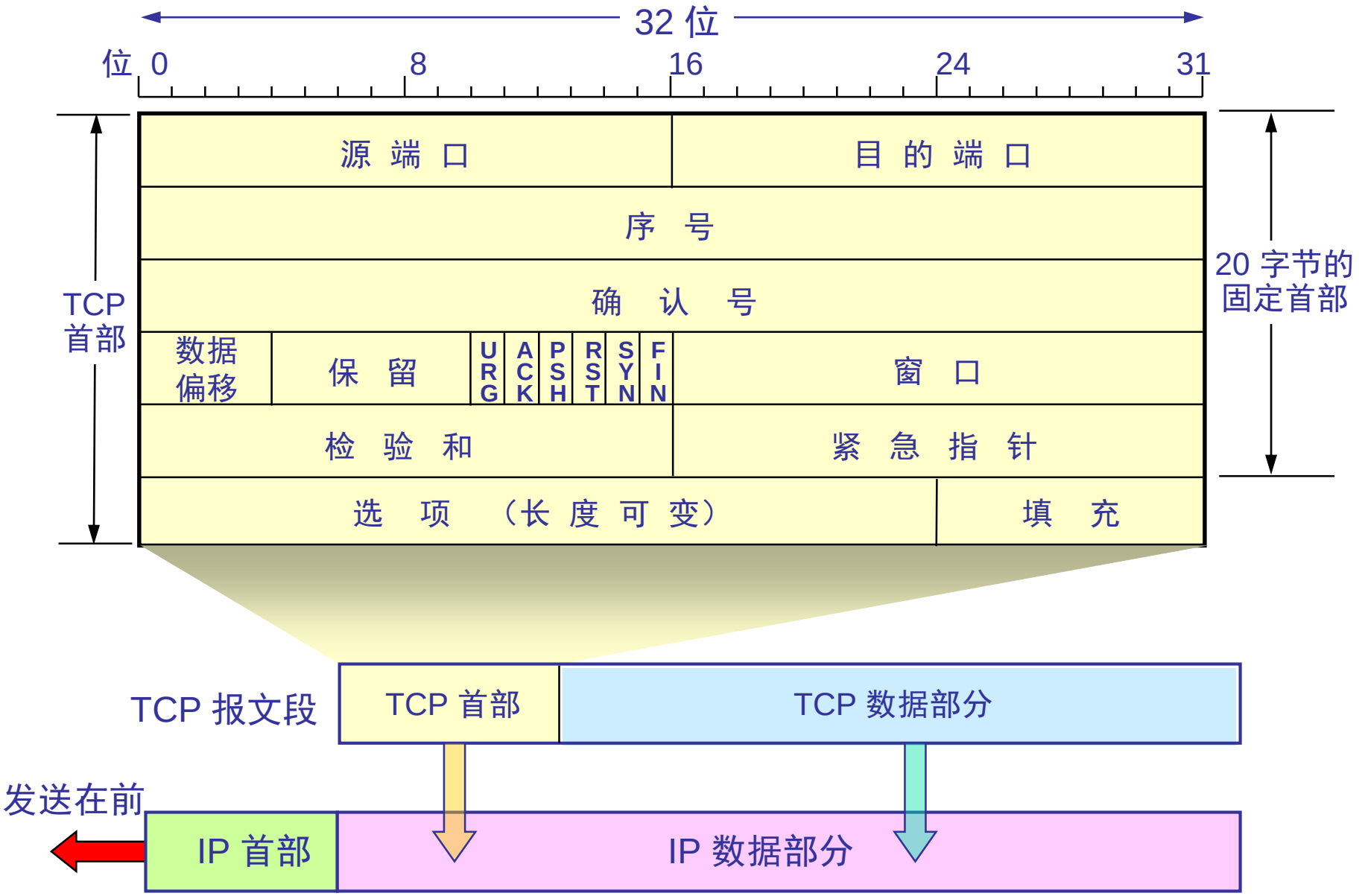
Go-back-N（回退 N）

- 如果发送方发送了前 5 个分组，而中间的第 3 个分组丢失了。这时接收方只能对前两个分组发出确认。发送方无法知道后面三个分组的下落，而只好把后面的三个分组都再重传一次。
- 这就叫做 Go-back-N（回退 N），表示需要再退回来重传已发送过的 *N* 个分组。

课后作业

- 习题：5-01,5-09,5-13,5-14

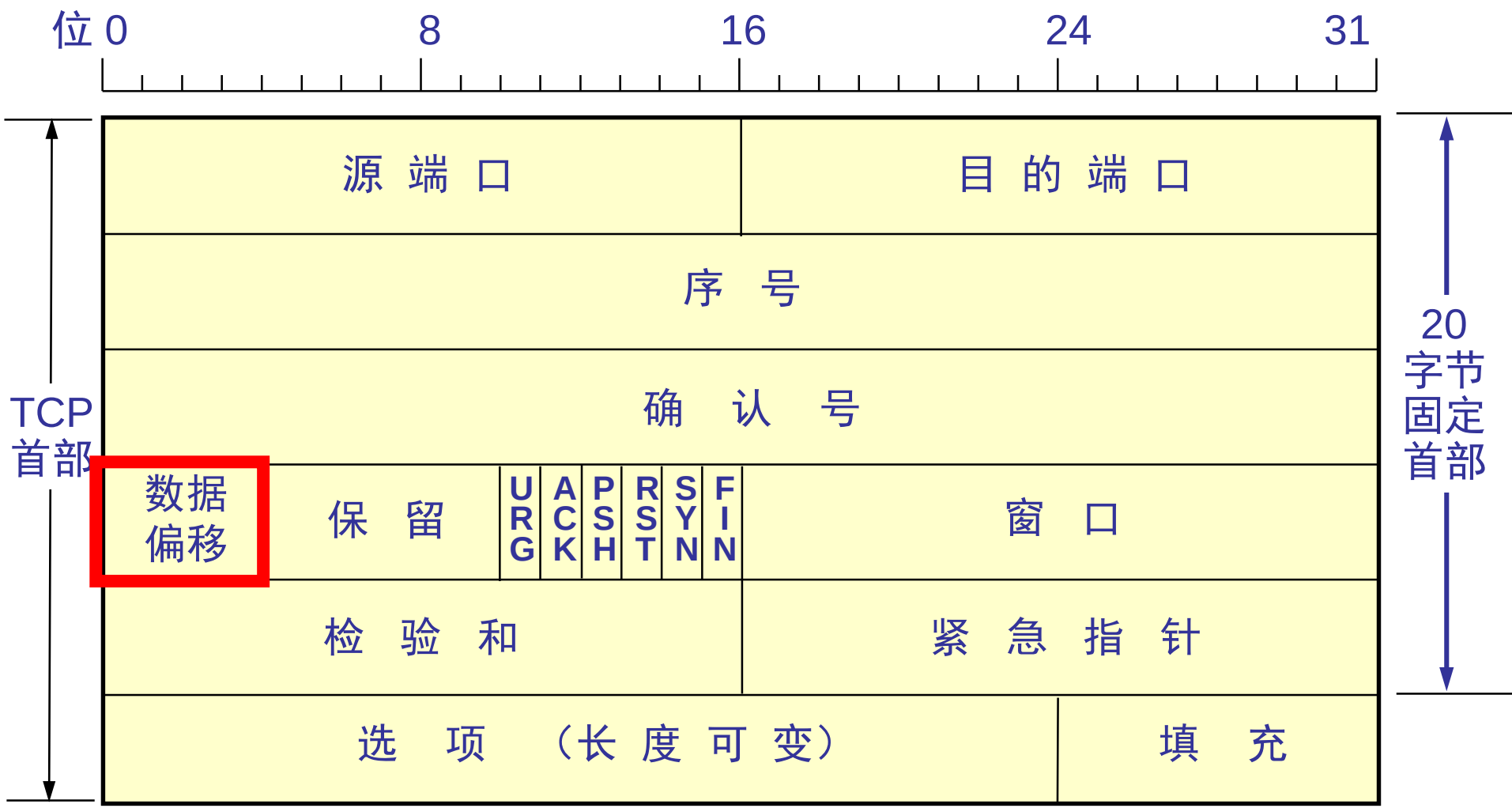
5.5 TCP 报文段的首部格式



源端口和目的端口字段——各占 2 字节。端口是运输层与应用层的服务接口。运输层的复用和分用功能都要通过端口才能实现。

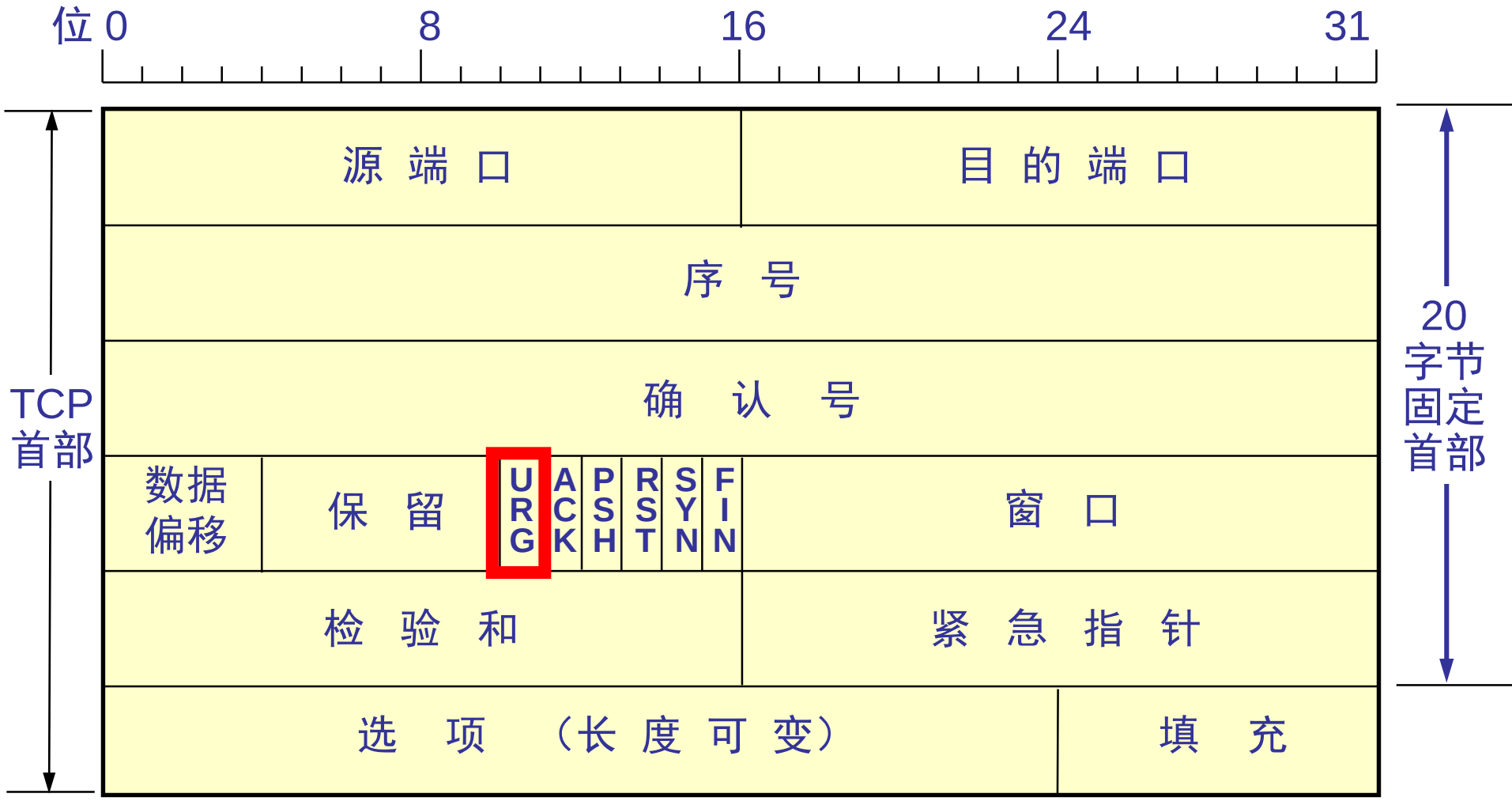
序号字段——占 4 字节。TCP 连接中传送的数据流中的每一个字节都编上一个序号。序号字段的值则指的是本报文段所发送的数据的第一个字节的序号。

确认号字段——占 4 字节，是期望收到对方的下一个报文段的数据的第一个字节的序号。

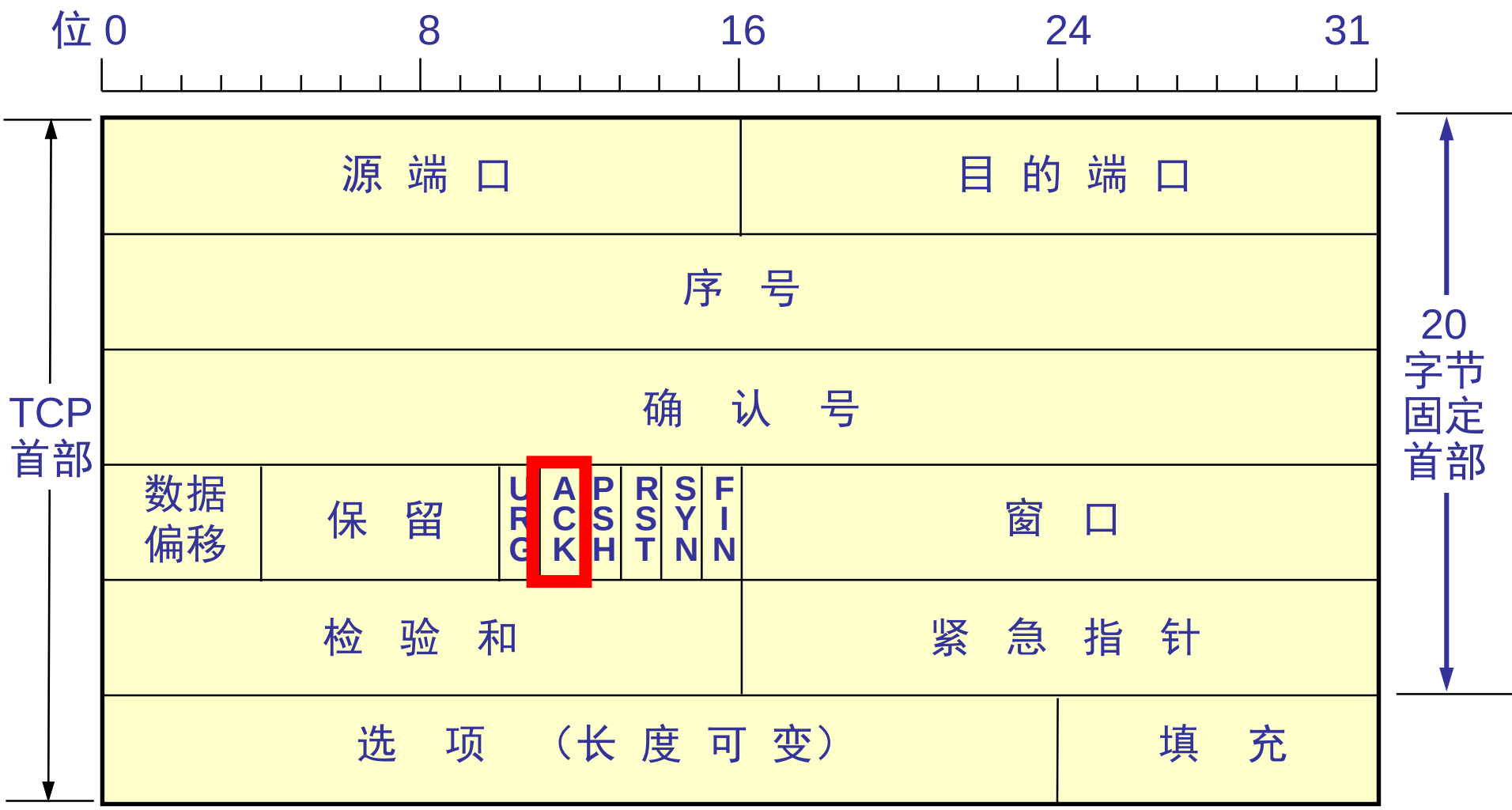


数据偏移（即首部长度）——占 4 位，它指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远。“数据偏移”的单位是 32 位字（以 4 字节为计算单位）。

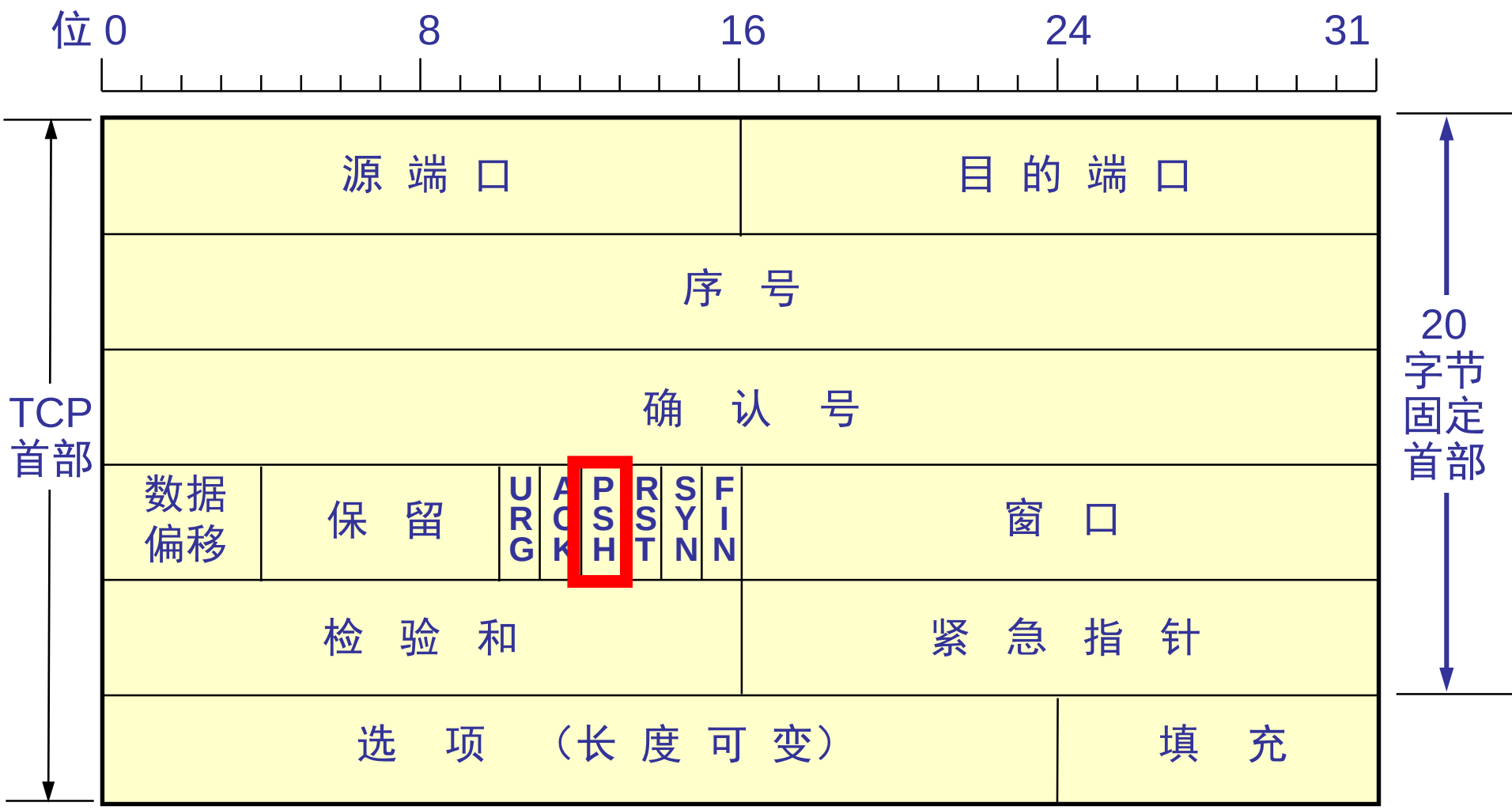
保留字段——占 6 位，保留为今后使用，但目前应置为 0。



紧急 URG —— 当 URG = 1 时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。

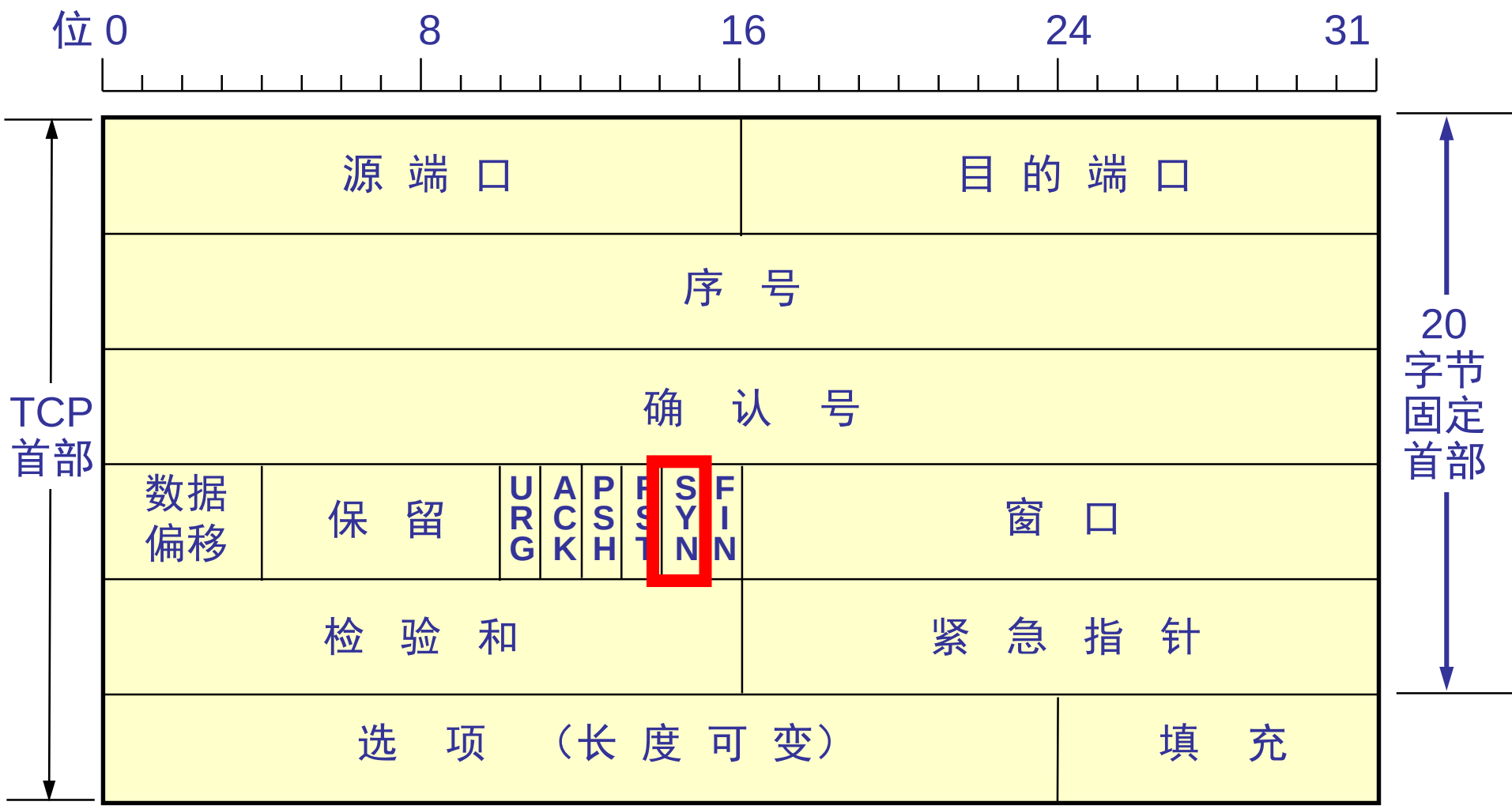


确认 ACK —— 只有当 ACK = 1 时确认号字段才有效。当 ACK = 0 时，确认号无效。



推送 PSH (PuSH) —— 接收 TCP 收到 PSH = 1 的报文段，就尽快地交付接收应用进程，而不再等到整个缓存都填满了后再向上交付。

复位 RST (ReSeT) —— 当 RST = 1 时，表明 TCP 连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。



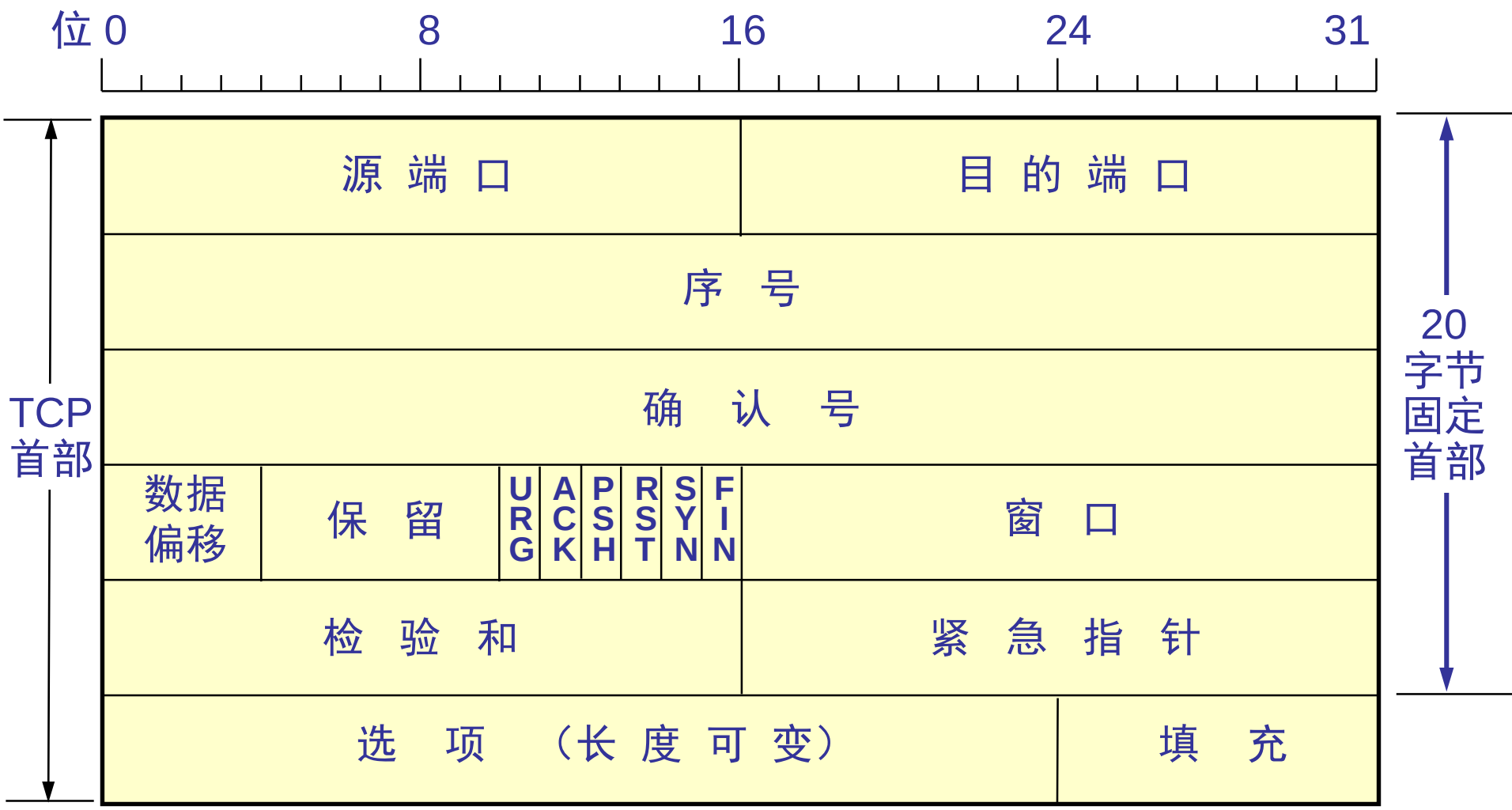
同步 SYN —— 同步 SYN = 1 表示这是一个连接请求或连接接受报文。

终止 FIN (FINis) —— 用来释放一个连接。FIN = 1 表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。

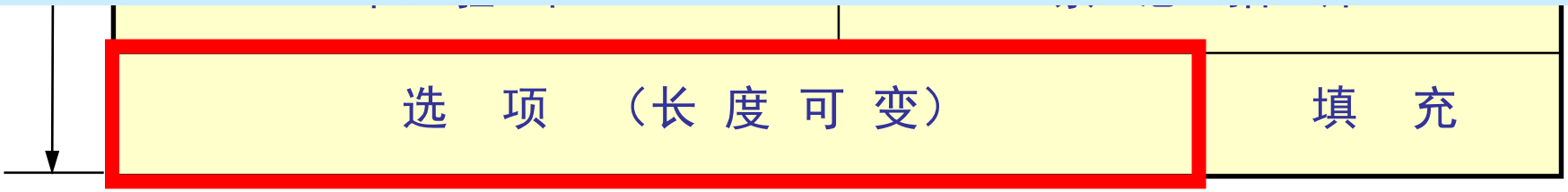
窗口字段 —— 占 2 字节，用来让对方设置发送窗口的依据，单位为字节。

检验和 —— 占 2 字节。检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在 TCP 报文段的前面加上 12 字节的伪首部。

紧急指针字段 —— 占 16 位，指出在本报文段中紧急数据共有多少个字节（紧急数据放在本报文段数据的最前面）。


位 0
8
16
24
31
源 端 口
目 的 端 口
序 号
确 认 号
数据偏移
保 留
URG
ACK
PSH
RST
SYN
FIN
窗 口
检 验 和
紧 急 指 针
选 项 （长 度 可 变）
填 充
TCP首部
20字节固定首部

MSS (Maximum Segment Size)
是 TCP 报文段中的数据字段的最大长度。
数据字段加上 TCP 首部
才等于整个的 TCP 报文段。



选项字段 —— 长度可变。TCP 最初只规定了一种选项，即最大报文段长度 MSS。MSS 指每一个报文中的数据字段的最大长度。

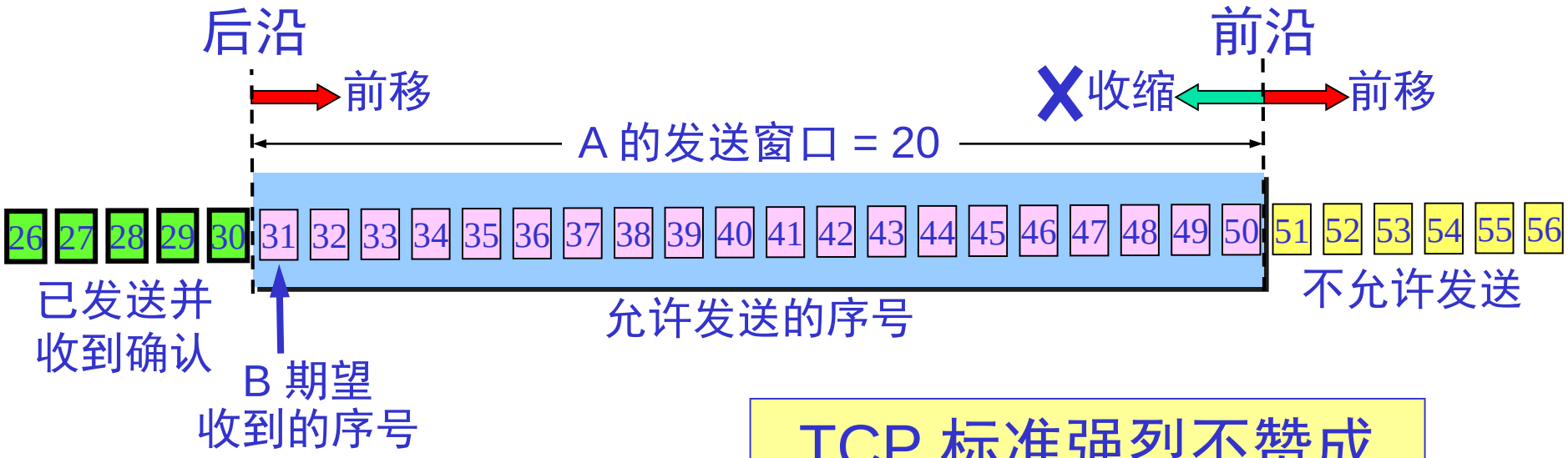
其他选项

- 窗口扩大选项 ——占 3 字节，其中有一个字节表示移位值 S。新的窗口值等于 TCP 首部中的窗口位数增大到(16 + S)，相当于把窗口值向左移动 S 位后获得实际的窗口大小。
- 时间戳选项——占10 字节，其中最主要的字段时间戳值字段（4 字节）和时间戳回送回答字段（4 字节）。
- 选择确认选项——在后面的 5.6.3 节介绍。

填充字段 —— 这是为了使整个首部长度是 4 字节的整数倍。

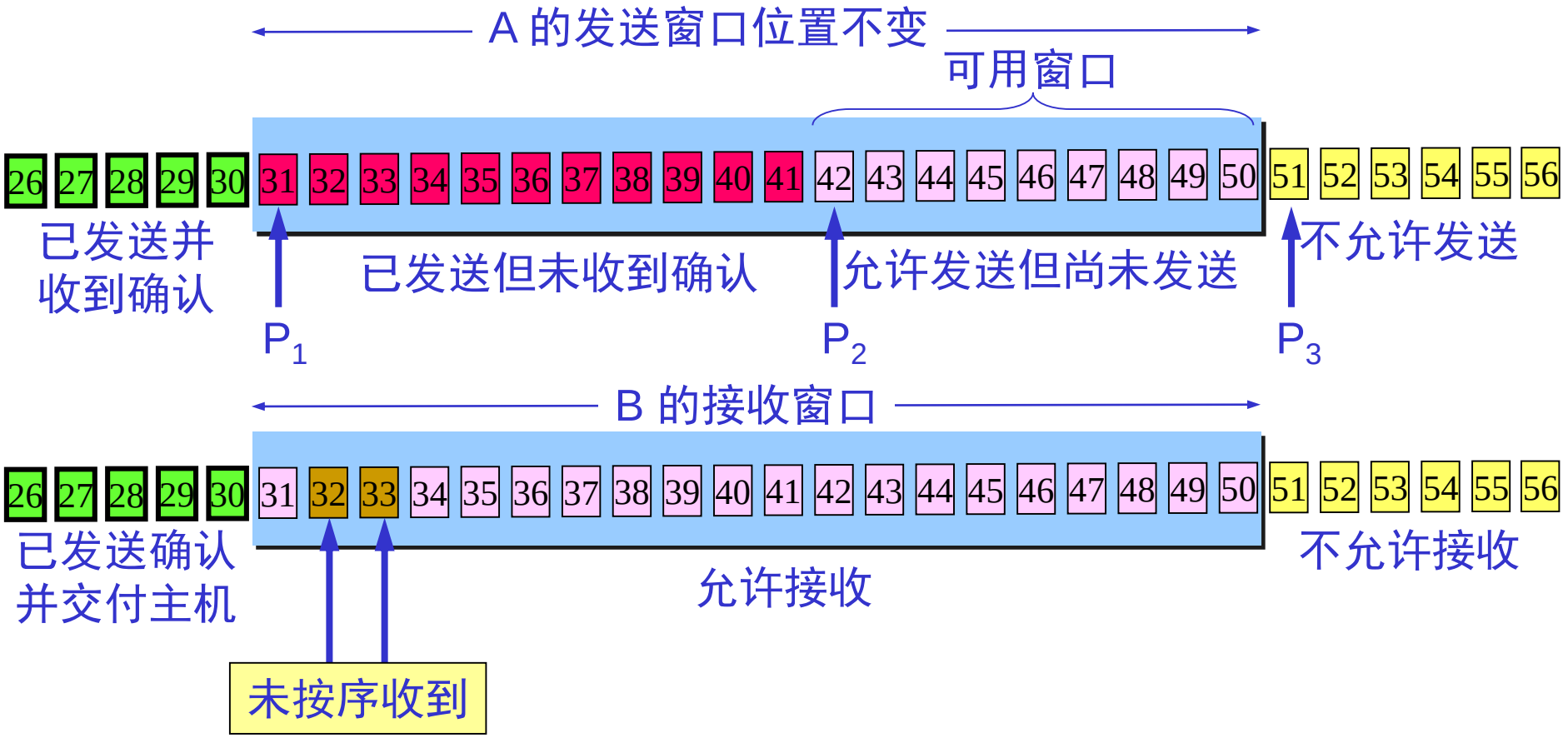
5.6 TCP 可靠传输的实现
5.6.1 以字节为单位的滑动窗口

根据 B 给出的窗口值
A 构造出自己的发送窗口



TCP 标准强烈不赞成
发送窗口前沿向后收缩

A 发送了 11 个字节的数据

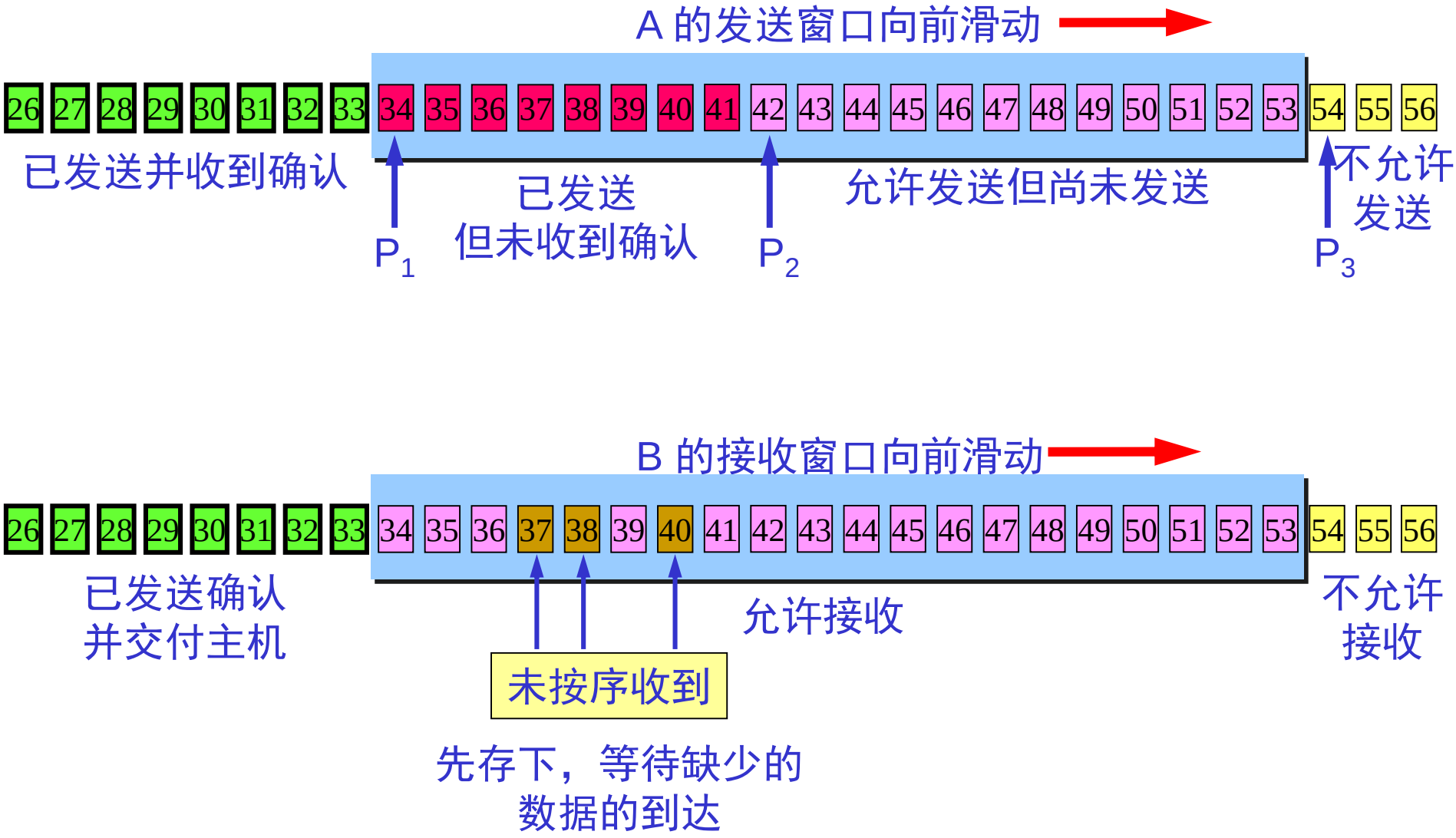


$P_3 - P_1$ = A 的发送窗口（又称为通知窗口）

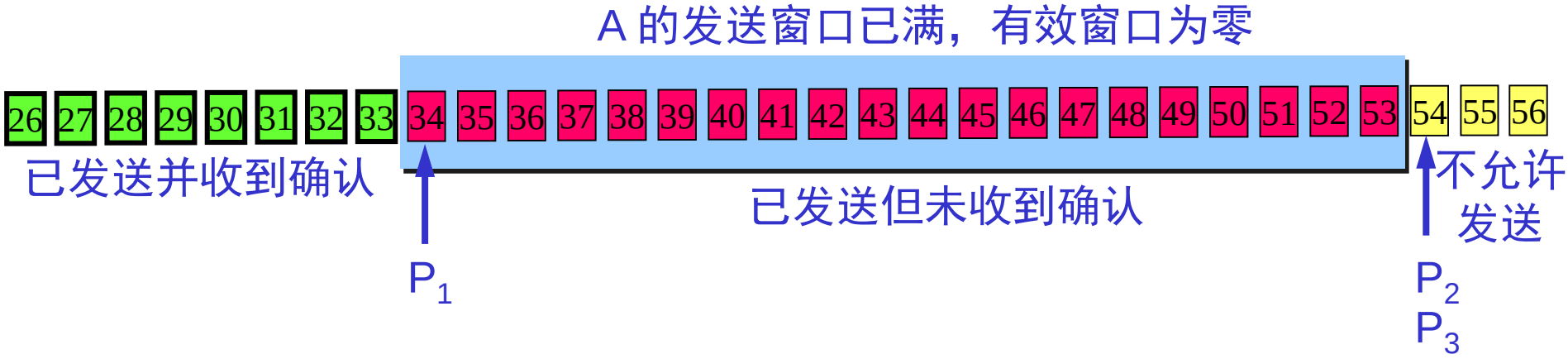
$P_2 - P_1$ = 已发送但尚未收到确认的字节数

$P_3 - P_2$ = 允许发送但尚未发送的字节数（又称为可用窗口）

A 收到新的确认号，发送窗口向前滑动

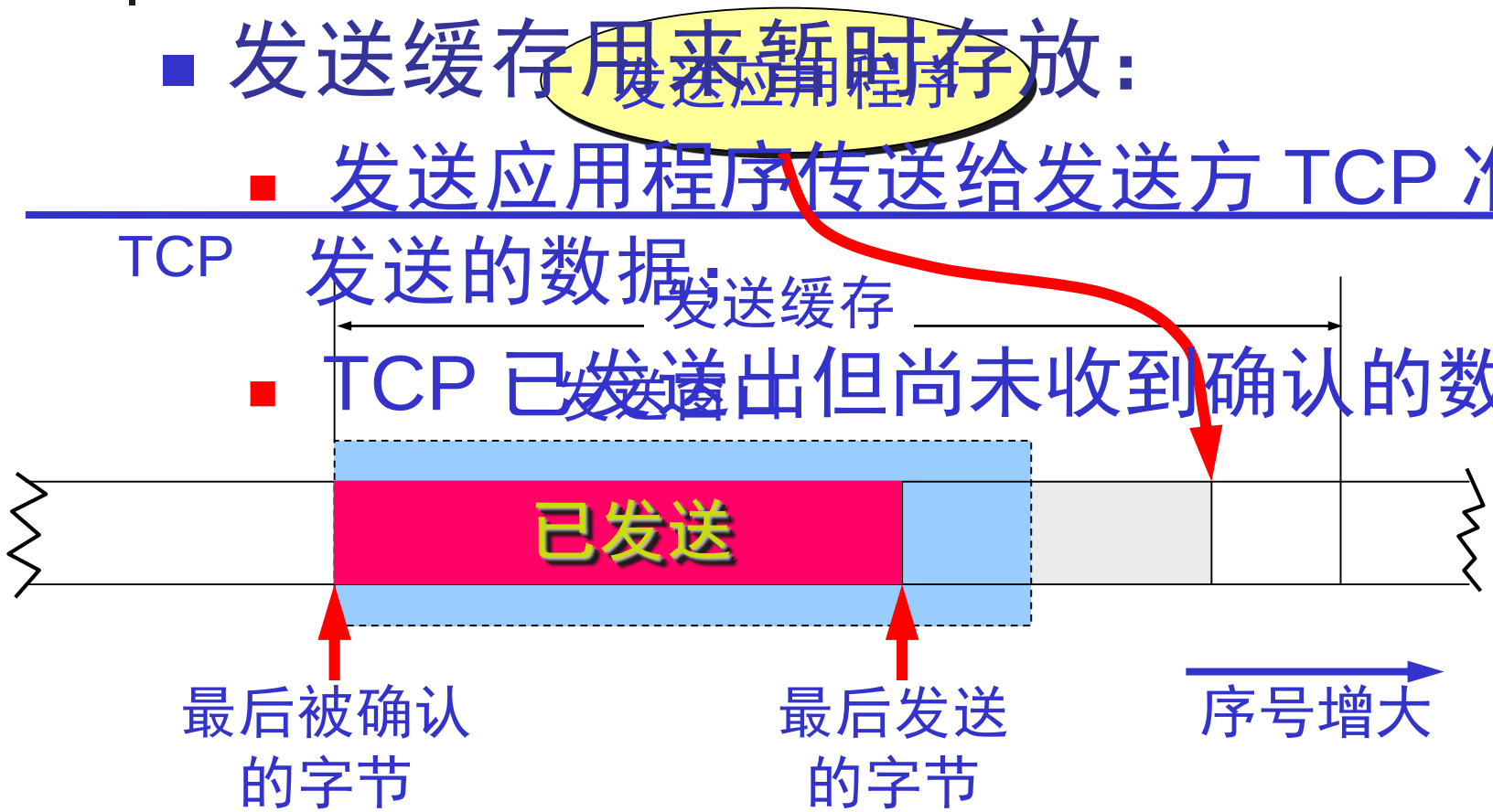


A 的发送窗口内的序号都已用完，
但还没有再收到确认，必须停止发送。



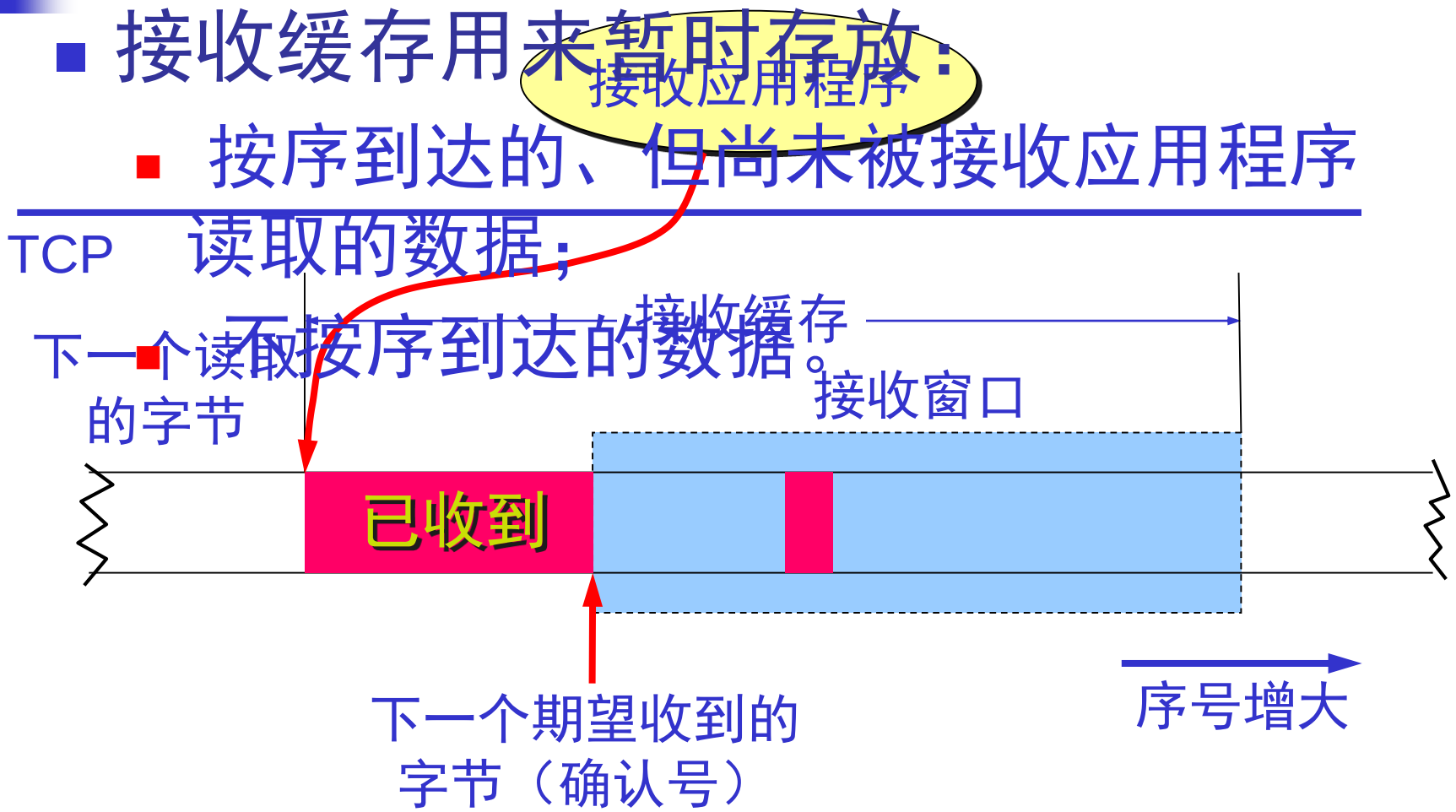
发送缓存

- 发送缓存用来暂时存放：
  - 发送应用程序传送给发送方 TCP 准备发送的数据；
  - TCP 已发送出但尚未收到确认的数据。



接收缓存

- 接收缓存用来暂时存放：
  - 按序到达的、但尚未被接收应用程序读取的数据；
  - 不按序到达的数据。



需要强调三点

- A 的发送窗口并不总是和 B 的接收窗口一样大（因为有一定的时间滞后）。
- TCP 对不按序到达的数据应的处理，通常是先临时存放在接收窗口中，等到字节流中所缺少的字节收到后，再按序交付上层的应用进程。
- TCP 要求接收方必须有累积确认的功能，这样可以减小传输开销。

5.6.2 超时重传时间的选择

- TCP 保留了 RTT 的一个加权平均往返时间 RTT_S。
- 第一次测量到 RTT 样本时，RTT_S 值就取为所测量到的 RTT 样本值。以后每测量到一个新的 RTT 样本，就按下式重新计算一次 RTT_S：

$$\text{新的 } RTT_S = (1-\alpha) \times (\text{旧的 } RTT_S) + \alpha \times (\text{新的 RTT 样本})$$

式中，$0 \le \alpha < 1$。

RFC 2988 推荐的 α 值为 1/8，即 0.125。

超时重传时间 RTO (RetransmissionTime-Out)

- RTO 应略大于上面得出的加权平均往返时间 RTT_S。
- RFC 2988 建议使用下式计算 RTO：
- $$RTO = RTT_S + 4 \times RTT_D$$

  RTT_D 是 RTT 的偏差的加权平均值。
- 第一次测量时，RTT_D 值取为测量到的 RTT 样本值的一半。在以后的测量中，则RTT_D：

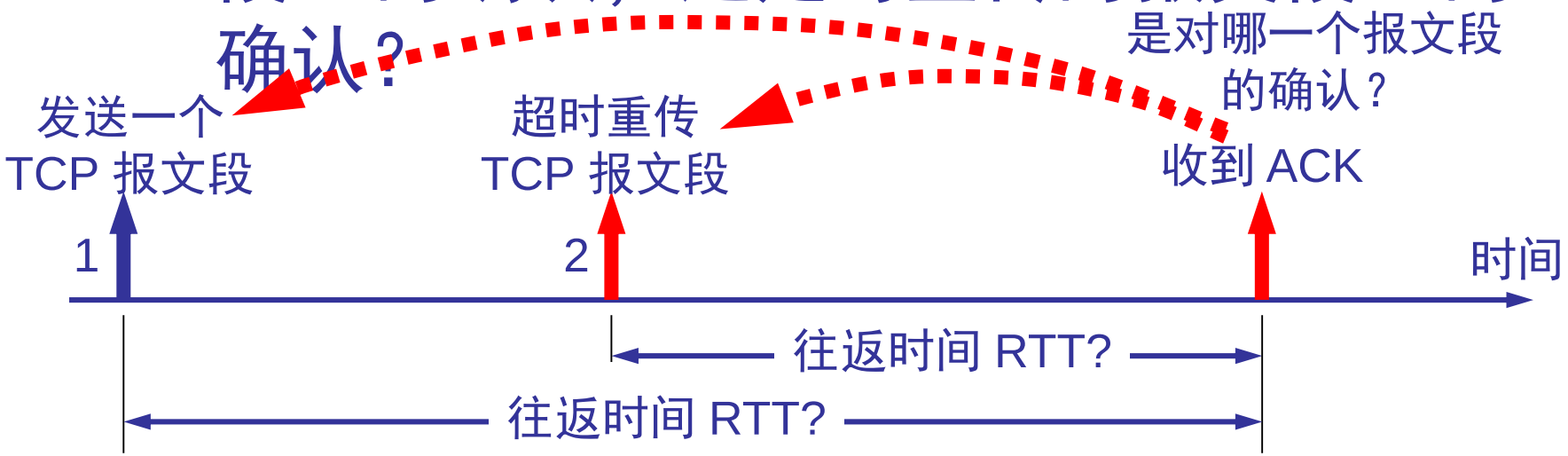
新的 $RTT_D = (1 - \beta) \times (\text{旧的} RTT_D)$

$+ \beta \times |RTT_S - \text{新的 RTT 样本}|$

β 是个小于 1 的系数，其推荐值是 1/4，即 0.25。

往返时间的测量相当复杂

- TCP 报文段 1 没有收到确认。重传（即报文段 2）后，收到了确认报文段 ACK。
- 如何判定此确认报文段是对原来的报文段 1 的确认，还是对重传的报文段 2 的确认？



Karn 算法

- 在计算平均往返时间 RTT 时，只要报文段重传了，就不采用其往返时间样本。
- 这样得出的加权平均平均往返时间 RTT_S 和超时重传时间 RTO 就较准确。

修正的 Karn 算法

- 报文段每重传一次，就把 RTO 增大一些：

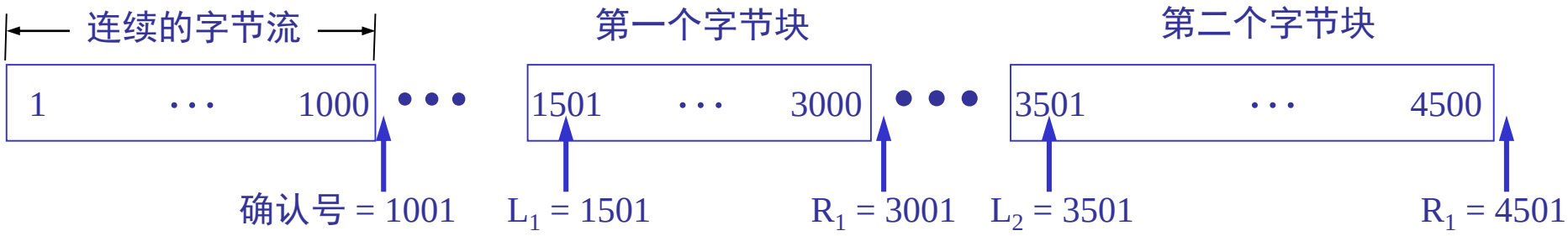
$$\text{新的 RTO} = \gamma \times (\text{旧的 RTO})$$

- 系数 γ 的典型值是 2 。
- 当不再发生报文段的重传时，才根据报文段的往返时延更新平均往返时延 RTT 和超时重传时间 RTO 的数值。
- 实践证明，这种策略较为合理。

5.6.3 选择确认 SACK (Selective ACK)

- 接收方收到了和前面的字节流不连续的两个字节块。
- 多数的重传还是所有未被确认的数据块。

接收到的字节流序号不连续



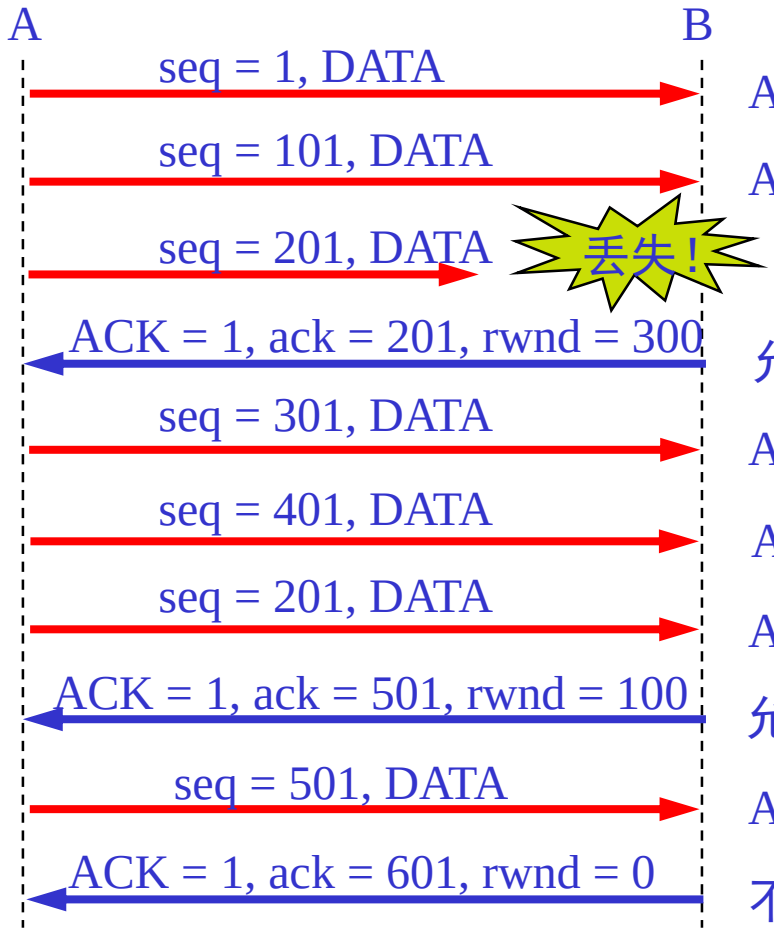
5.7 TCP 的流量控制

5.7.1 利用滑动窗口实现流量控制

- 流量控制(flow control)就是让发送方的发送速率不要太快，既要让接收方来得及接收，也不要使网络发生拥塞。
- 利用滑动窗口机制可以很方便地在 TCP 连接上实现流量控制。

流量控制举例

A 向 B 发送数据。在连接建立时，
B 告诉 A：“我的接收窗口 rwnd = 400（字节）”。



A 发送了序号 1 至 100，还能发送 300 字节

A 发送了序号 101 至 200，还能发送 200 字节

允许 A 发送序号 201 至 500 共 300 字节

A 发送了序号 301 至 400，还能再发送 100 字节新数据

A 发送了序号 401 至 500，不能再发送新数据了

A 超时重传旧的数据，但不能发送新的数据

允许 A 发送序号 501 至 600 共 100 字节

A 发送了序号 501 至 600，不能再发送了

不允许 A 再发送（到序号 600 为止的数据都收到了）

持续计时器

- TCP 为每一个连接设有一个持续计时器。
- 只要 TCP 连接的一方收到对方的零窗口通知，就启动持续计时器。
- 若持续计时器设置的时间到期，就发送一个零窗口探测报文段（仅携带 1 字节的数据），而对方就在确认这个探测报文段时给出了现在的窗口值。

5.7.2 TCP的传输效率

- 可以用不同的机制来控制 TCP 报文段的发送时机:
- 第一种机制是 TCP 维持一个变量，它等于最大报文段长度 MSS。只要缓存中存放的数据达到 MSS 字节时，就组装成一个 TCP 报文段发送出去。
- 第二种机制是由发送方的应用进程指明要求发送报文段，即 TCP 支持的推送(push)操作。
- 第三种机制是发送方的一个计时器期限到了，这时就把当前已有的缓存数据装入报文段（但长度不能超过 MSS）发送出去。

糊涂窗口综合症

- 当发送端应用进程产生数据很慢、或接收端应用进程处理接收缓冲区数据很慢，或二者兼而有之；就会使应用进程间传送的报文段很小，特别是有效载荷很小。
- 极端情况下，有效载荷可能只有1个字节；而传输开销有40字节(20字节的IP头+20字节的TCP头) 这种现象就叫糊涂窗口综合症 .

课后作业

- 习题：5-22,5-23,5-30,5-31,5-33

5.8 TCP的拥塞控制

5.8.1 拥塞控制的一般原理

- 在某段时间，若对网络中某资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏——产生拥塞(congestion)。
- 出现资源拥塞的条件：

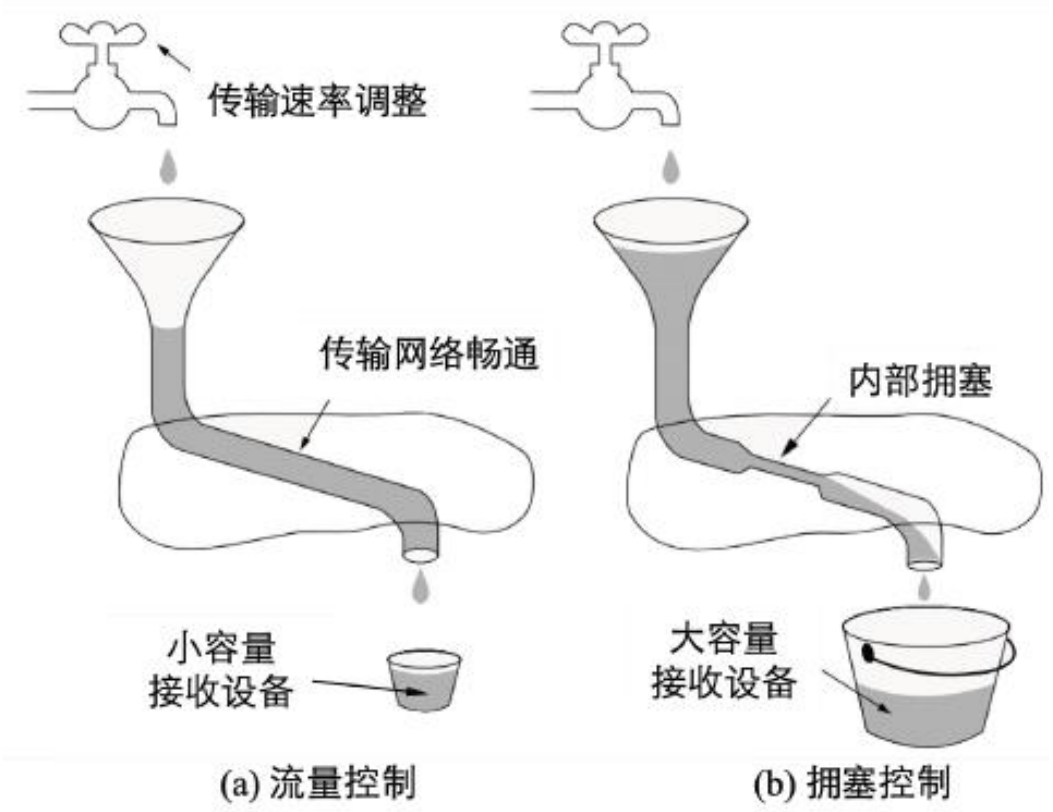
对资源需求的总和 > 可用资源

若网络中有许多资源同时产生拥塞，网络的性能就要明显变坏，整个网络的吞吐量将随输入负荷的增大而下降。

拥塞控制与流量控制的关系

- 拥塞控制所要做的都有一个前提，就是网络能够承受现有的网络负荷。
- 拥塞控制是一个全局性的过程，涉及到所有的主机、所有的路由器，以及与降低网络传输性能有关的所有因素。
- 流量控制往往指在给定的发送端和接收端之间的点对点通信量的控制。
- 流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。

拥塞控制与流量控制的关系



5.8.2 TCP拥塞控制方法
1. 慢开始和拥塞避免

- 发送方维持一个叫做拥塞窗口 cwnd (congestion window)的状态变量。拥塞窗口的大小取决于网络的拥塞程度，并且动态地在变化。发送方让自己的发送窗口等于拥塞窗口。

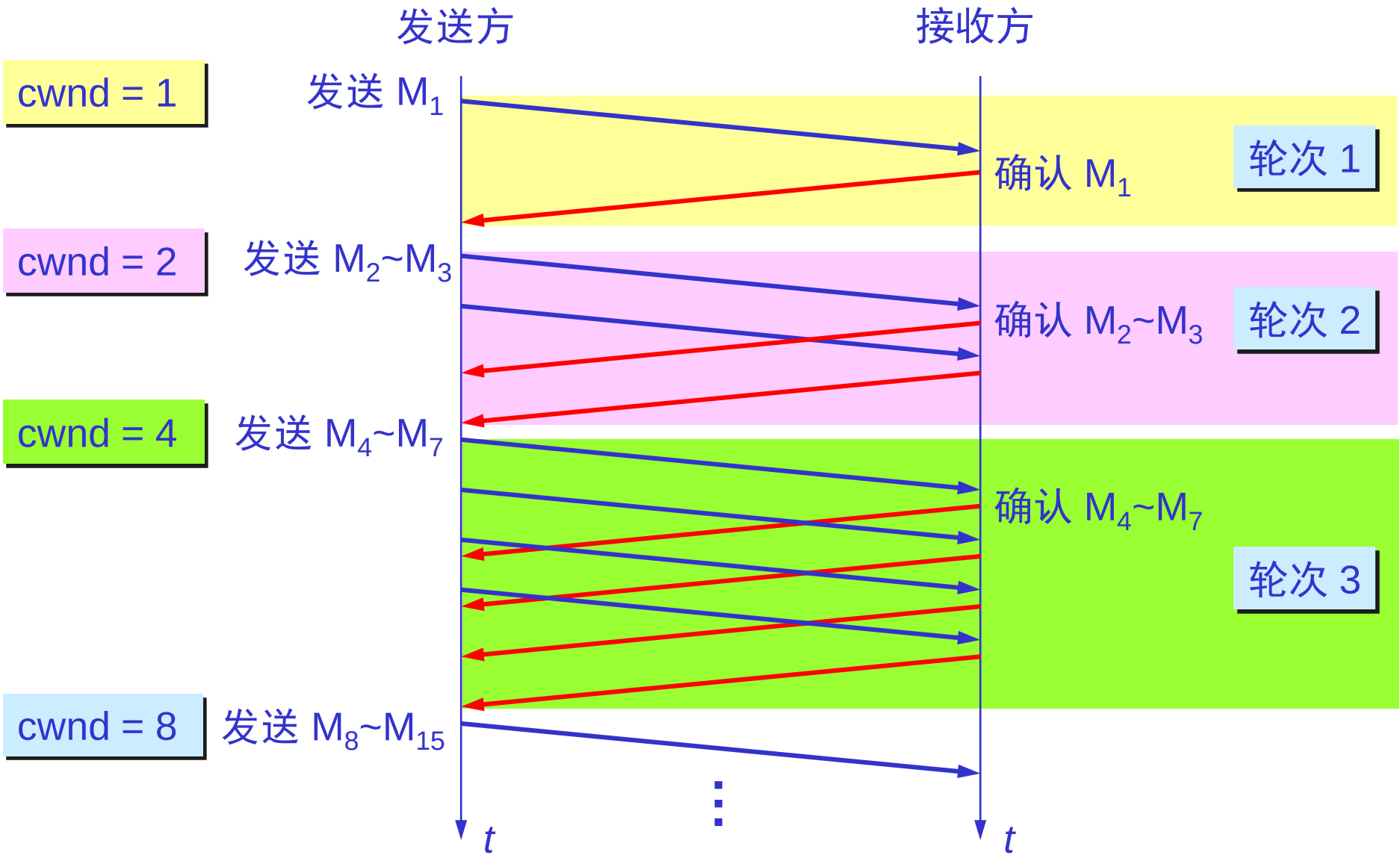
发送方控制拥塞窗口的原则

- 只要网络没有出现拥塞，拥塞窗口就再增大一些，以便把更多的分组发送出去。但只要网络出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数。

慢开始算法的原理

- 在主机刚刚开始发送报文段时可先设置拥塞窗口 cwnd = 1个报文段的数值。
- 在每收到一个对新的报文段的确认后，将拥塞窗口增加1个报文段的数值。
- 用这样的方法逐步增大发送端的拥塞窗口 cwnd，可以使分组注入到网络的速率更加合理。

发送方每收到一个对新报文段的确认（重传的不算在内）就使 cwnd 加 1。



设置慢开始门限状态变量 ssthresh

- 慢开始门限 ssthresh 的用法如下：

  当 cwnd < ssthresh 时，使用慢开始算法。

  当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。

  当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞避免算法。

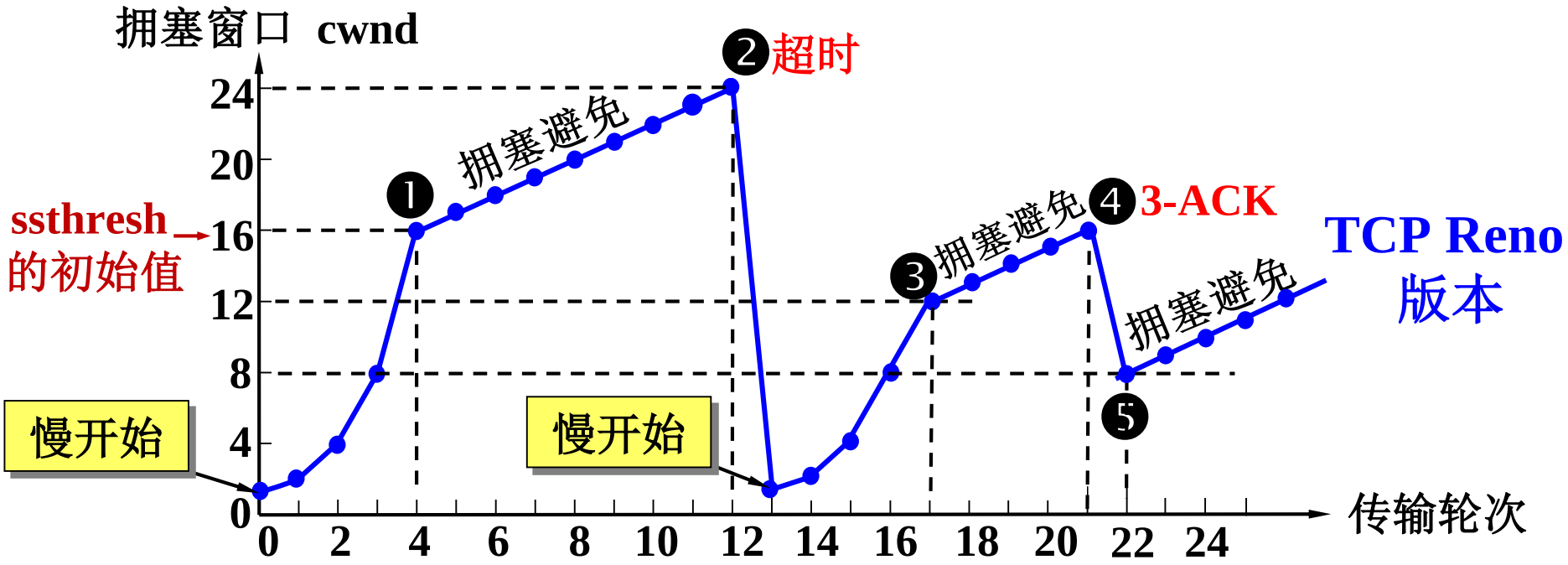
拥塞避免算法

- 拥塞避免算法的思路是让拥塞窗口 cwnd 缓慢地增大，即每经过一个往返时间 RTT 就把发送方的拥塞窗口 cwnd 加 1，而不是加倍，使拥塞窗口 cwnd 按线性规律缓慢增长。

当网络出现拥塞时

- 发送方判断网络出现拥塞（其根据就是没有按时收到确认），就要把慢开始门限 ssthresh 设置为出现拥塞时的发送方窗口值的一半（但不能小于2）。
- 然后把拥塞窗口 cwnd 重新设置为 1，执行慢开始算法。

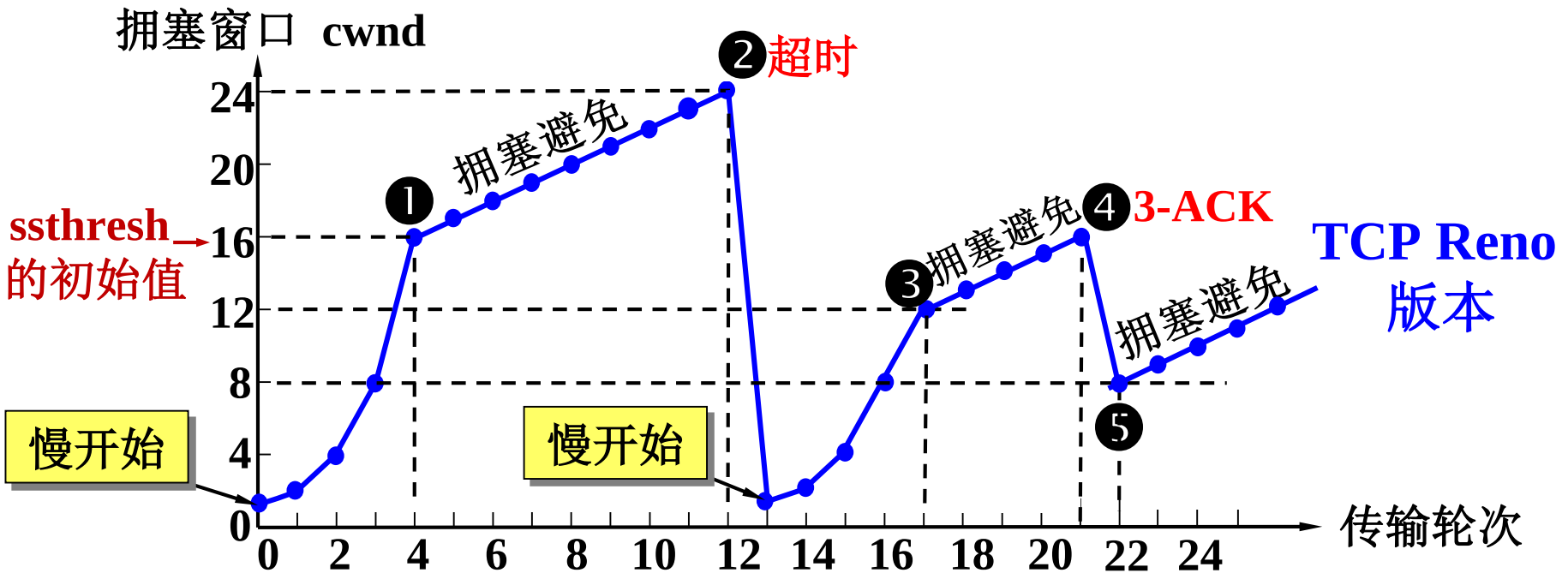
慢开始和拥塞避免算法的实现举例



当 TCP 连接进行初始化时，将拥塞窗口置为 1。图中的窗口单位不使用字节而使用报文段。

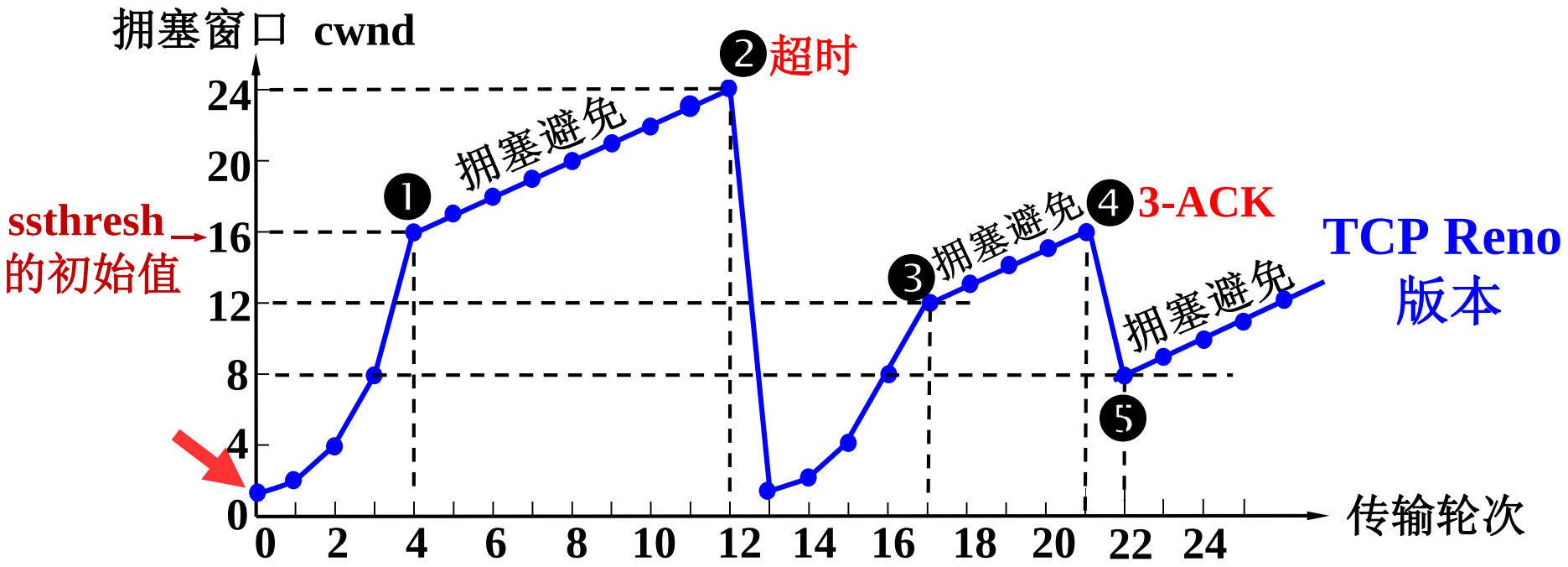
慢开始门限的初始值设置为 16 个报文段，即 ssthresh = 16。

慢开始和拥塞避免算法的实现举例



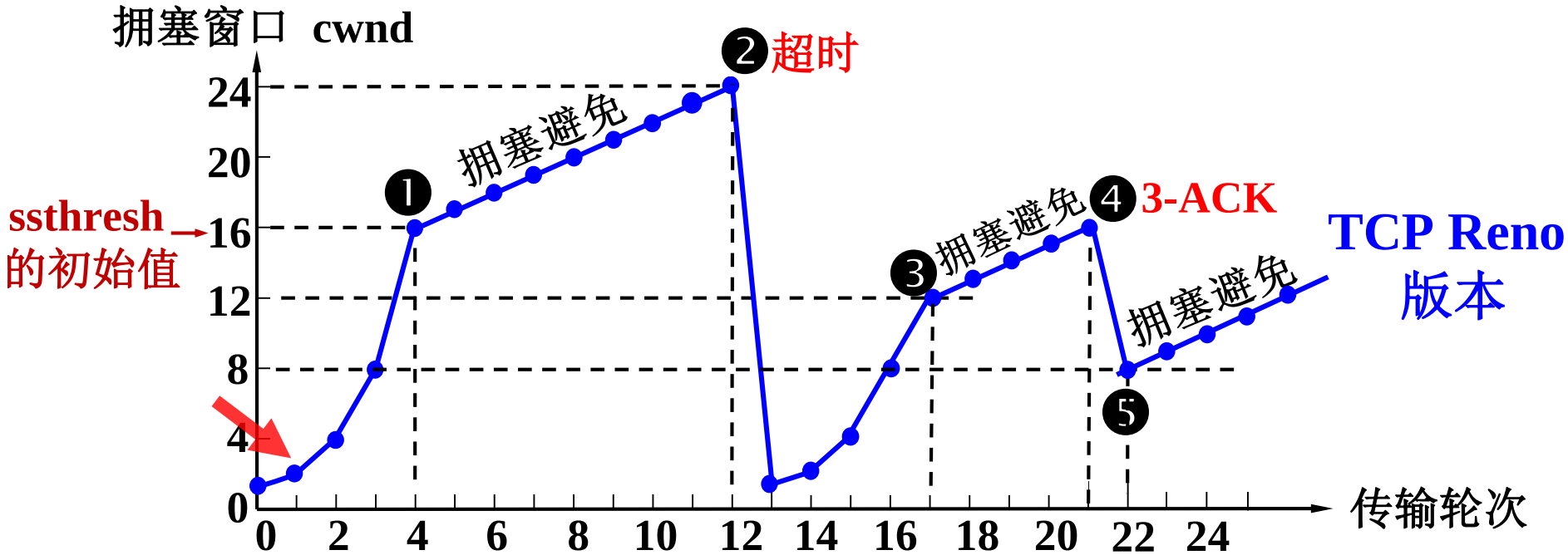
发送端的发送窗口不能超过拥塞窗口 cwnd 和接收端窗口 rwnd 中的最小值。我们假定接收端窗口足够大，因此现在发送窗口的数值等于拥塞窗口的数值。

慢开始和拥塞避免算法的实现举例



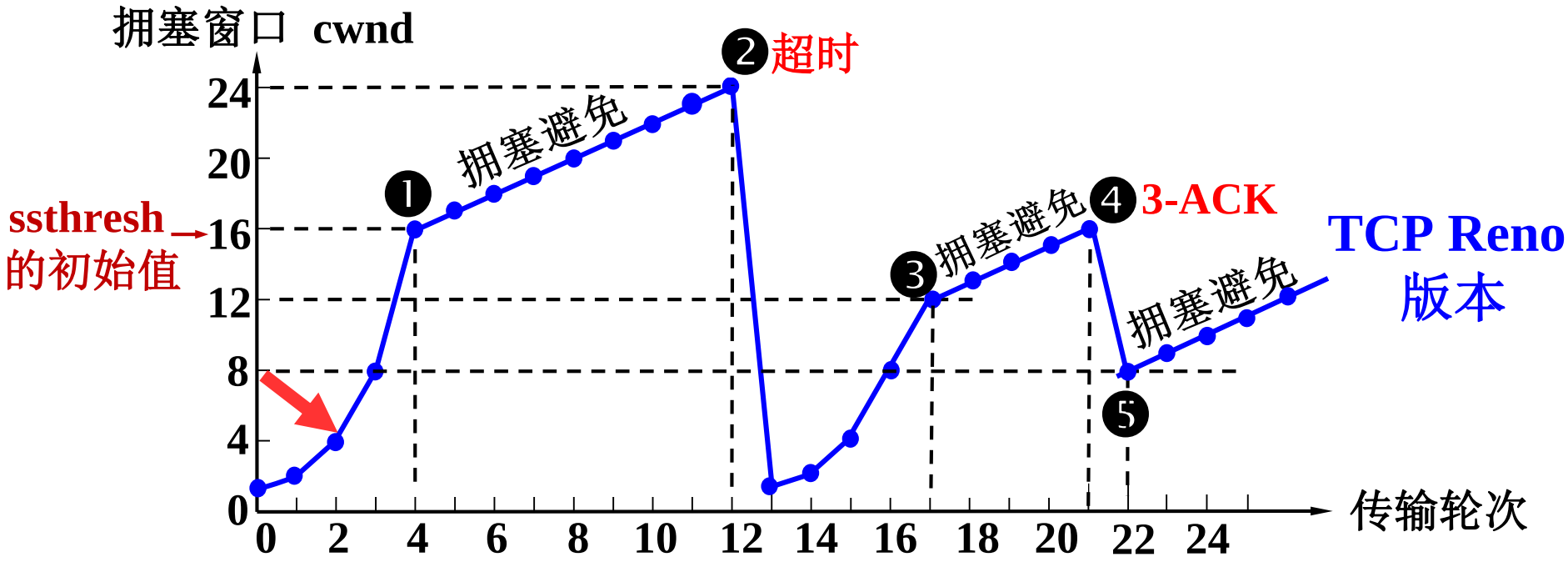
在执行慢开始算法时，拥塞窗口 cwnd 的初始值为 1，发送第一个报文段 M_0。

慢开始和拥塞避免算法的实现举例



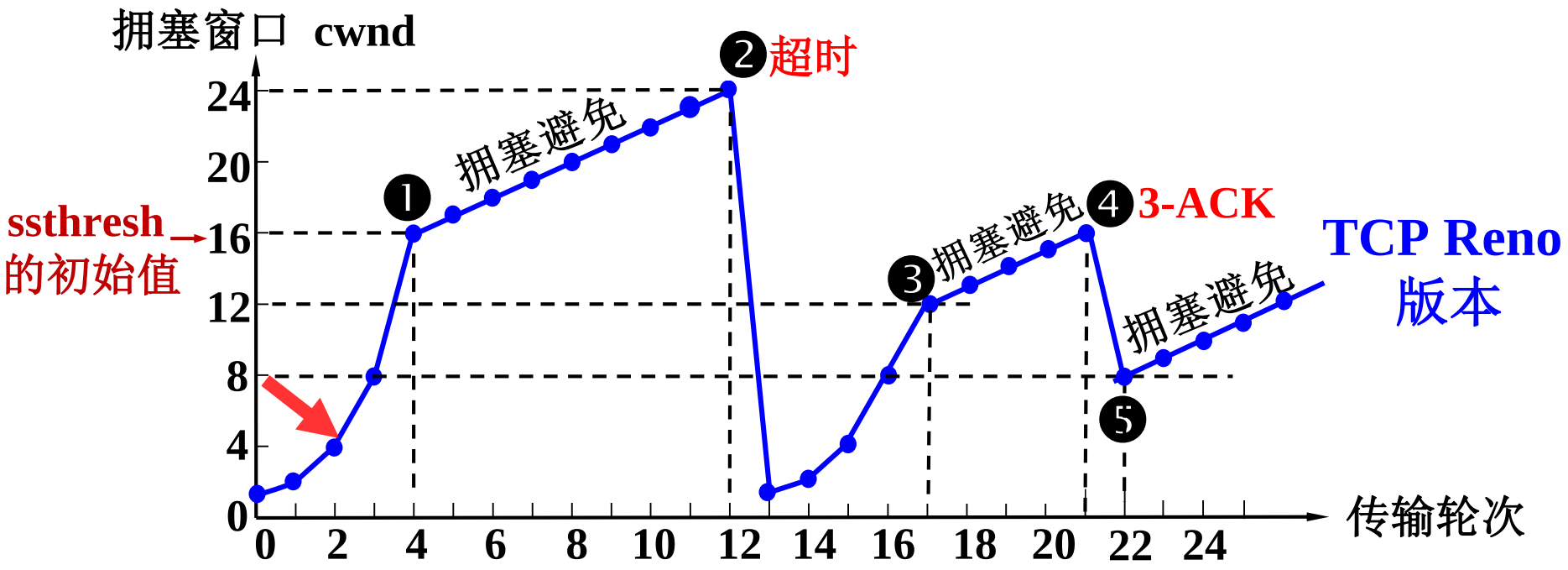
发送端每收到一个确认，就把 cwnd 加 1。于是发送端可以接着发送 M_1 和 M_2 两个报文段。

慢开始和拥塞避免算法的实现举例



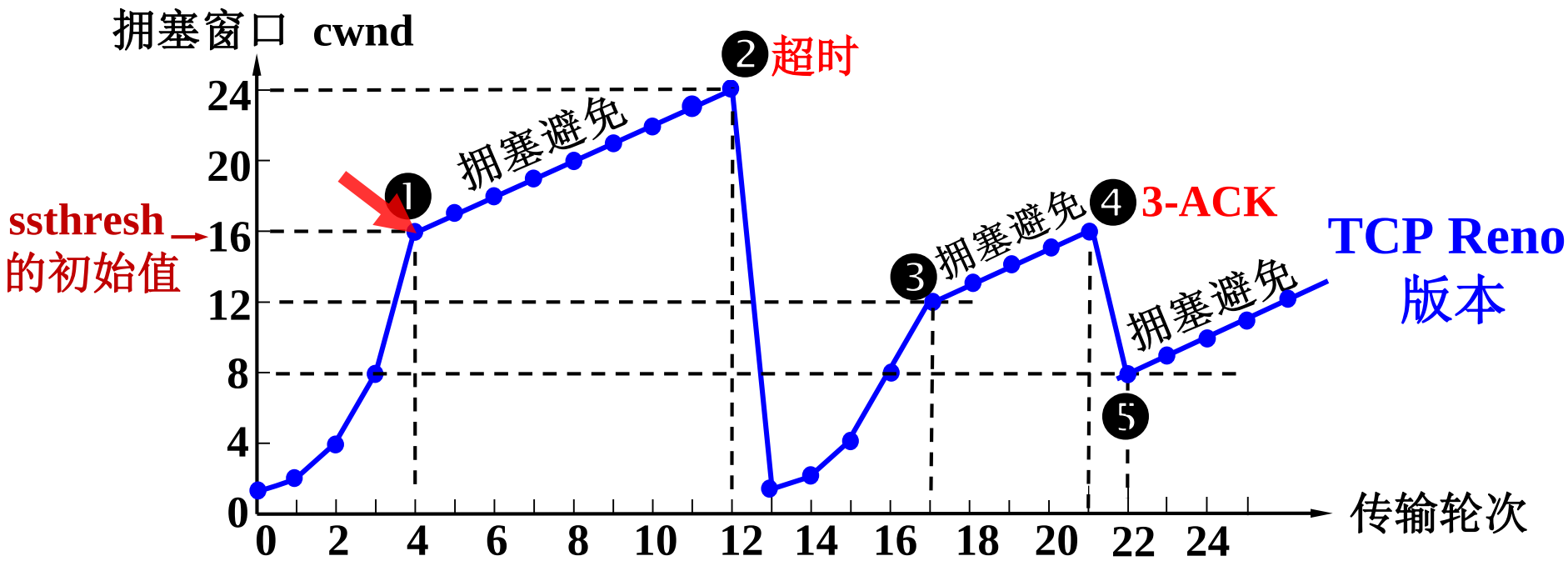
接收端共发回两个确认。发送端每收到一个对新报文段的确认，就把发送端的 cwnd 加 1。现在 cwnd 从 2 增大到 4，并可接着发送后面的 4 个报文段。

慢开始和拥塞避免算法的实现举例



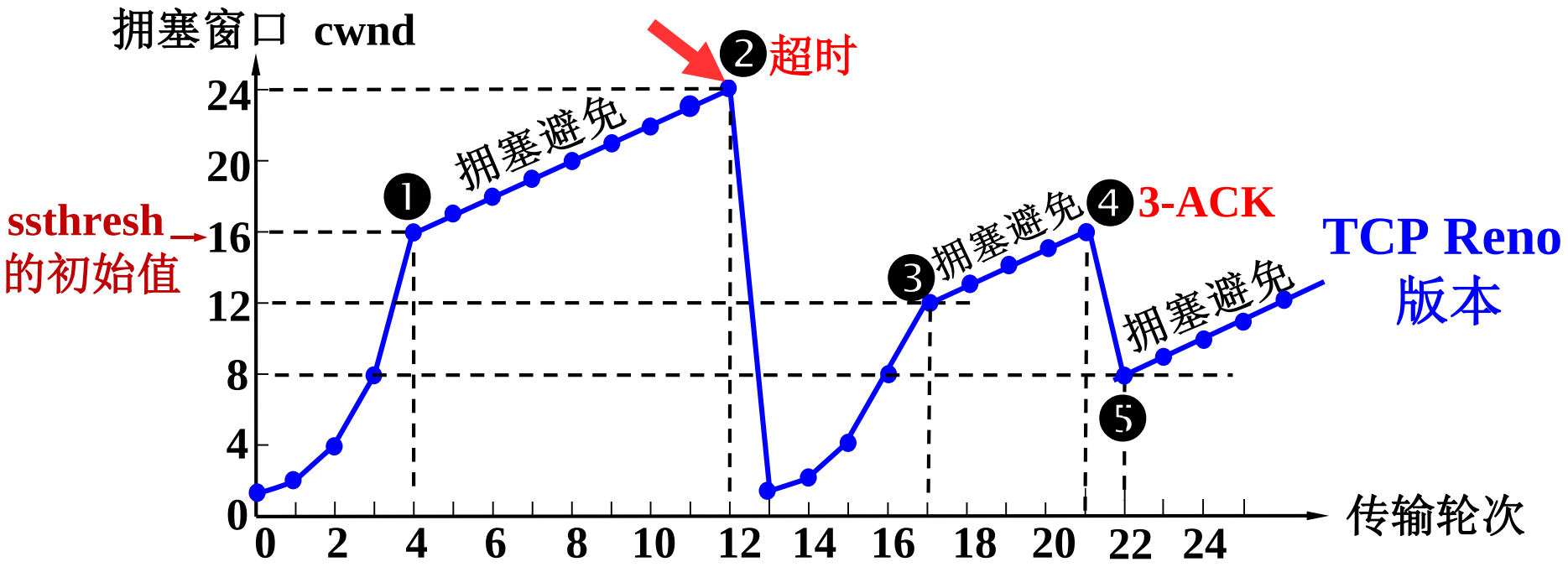
发送端每收到一个对新报文段的确认，就把发送端的拥塞窗口加 1，因此拥塞窗口 cwnd 随着传输轮次按指数规律增长。

慢开始和拥塞避免算法的实现举例



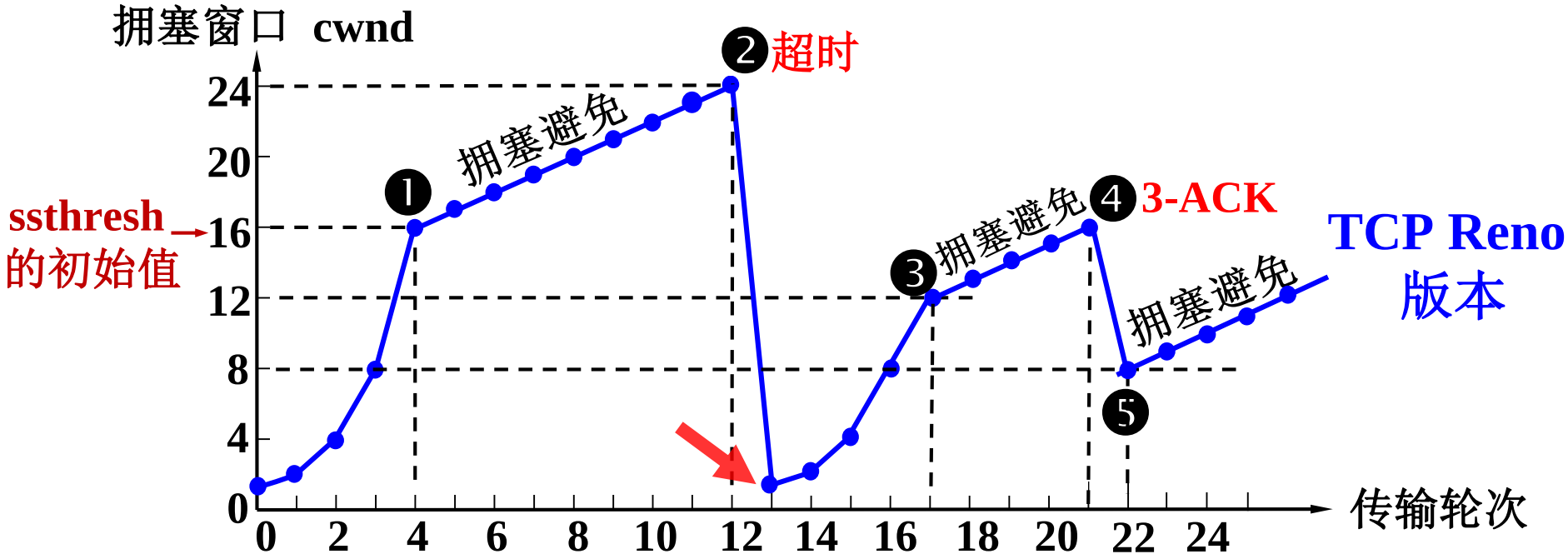
当拥塞窗口 cwnd 增长到慢开始门限值 ssthresh 时（即当 cwnd = 16 时），就改为执行拥塞避免算法，拥塞窗口按线性规律增长。

慢开始和拥塞避免算法的实现举例



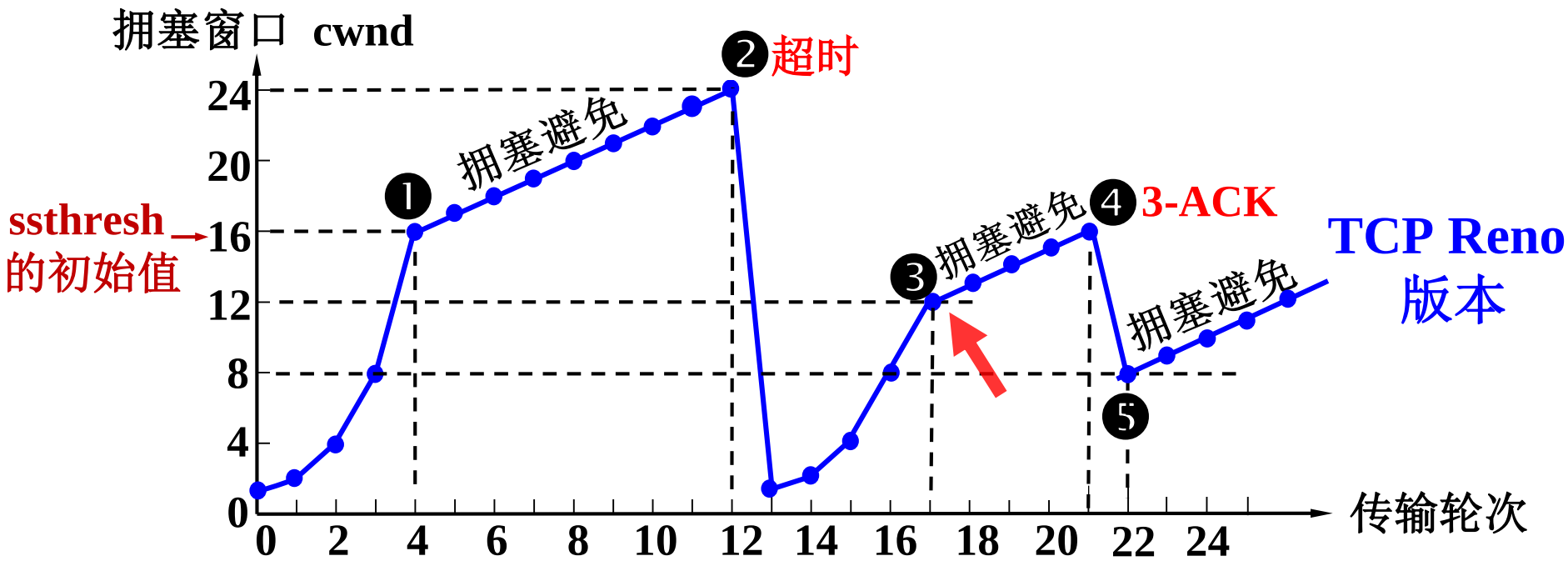
假定拥塞窗口的数值增长到 24 时，网络出现超时，表明网络拥塞了。

慢开始和拥塞避免算法的实现举例



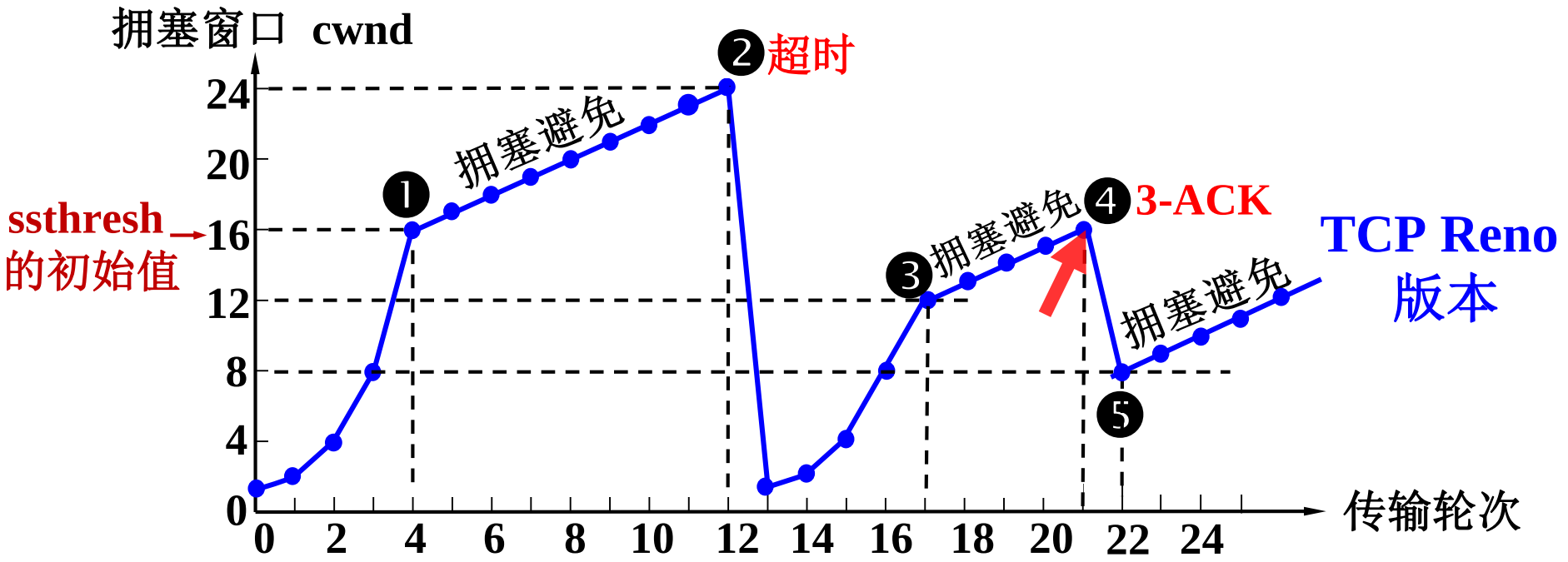
更新后的 ssthresh 值变为 12（即发送窗口数值 24 的一半），拥塞窗口再重新设置为 1，并执行慢开始算法。

慢开始和拥塞避免算法的实现举例



当 cwnd = 12 时改为执行拥塞避免算法，拥塞窗口按按线性规律增长，每经过一个往返时延就增加一个 报文段的大小。

慢开始和拥塞避免算法的实现举例

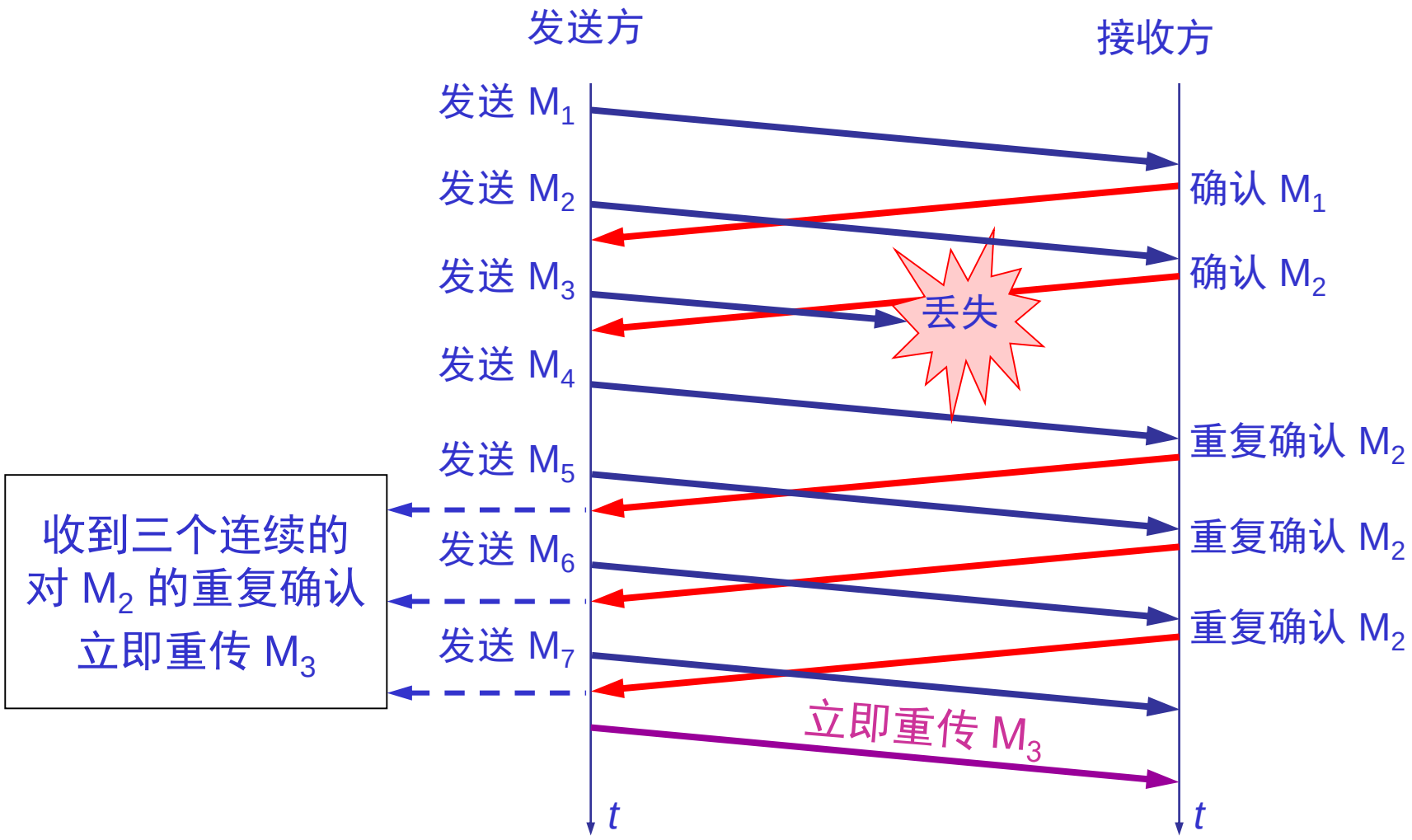


当拥塞窗口cwnd = 16时（图中的点❹），出现了一个新的情况，就是发送方一连收到 3 个对同一个报文段的重复确认（图中记为3-ACK）。发送方改为执行快重传和快恢复算法。

2. 快重传和快恢复

- 快重传算法首先要求接收方每收到一个失序的报文段(丢失)后就立即发出重复确认。这样做可以让发送方及早知道有报文段没有到达接收方。
- 发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段。

快重传举例


发送方
接收方
发送 M_1
发送 M_2
发送 M_3
发送 M_4
发送 M_5
发送 M_6
发送 M_7
确认 M_1
确认 M_2
丢失
重复确认 M_2
重复确认 M_2
重复确认 M_2
立即重传 M_3
收到三个连续的
对 M_2 的重复确认
立即重传 M_3
t
t

快恢复算法

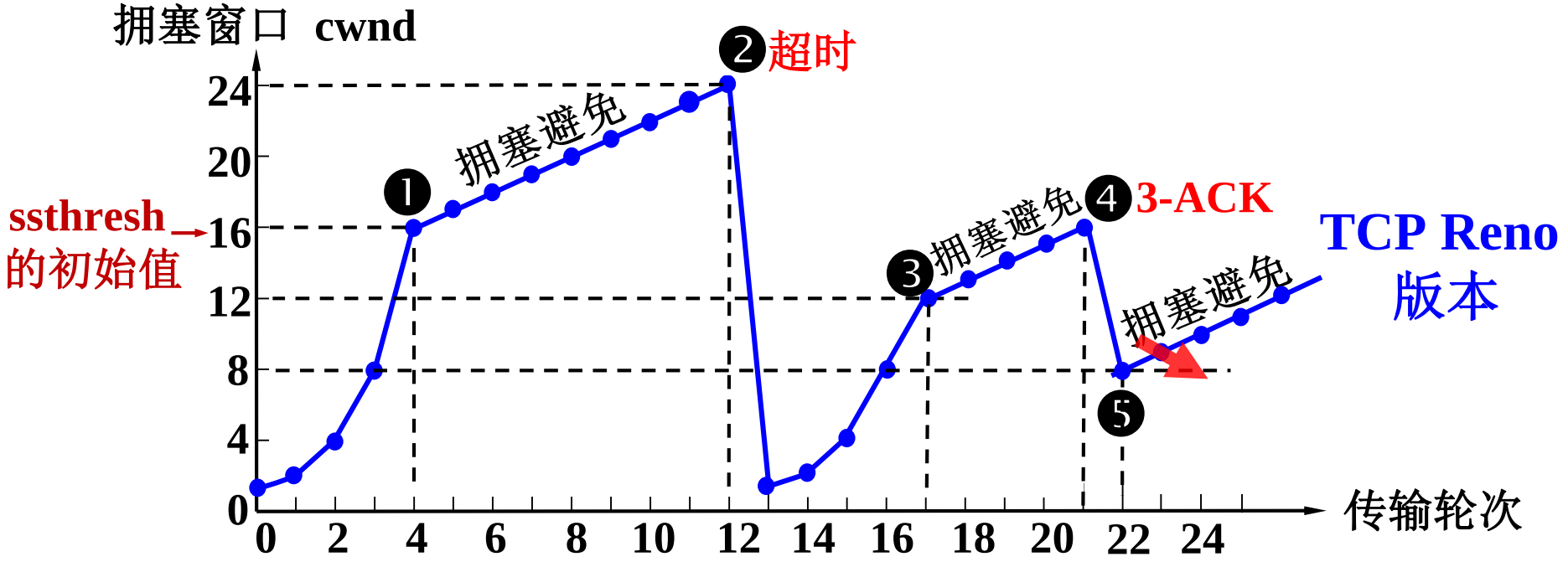
- 当发送端收到连续三个重复的确认时，由于发送方现在认为网络很可能没有发生拥塞，因此现在不执行慢开始算法，而是执行快恢复算法 FR (Fast Recovery) 算法：

(1) 慢开始门限 ssthresh = 当前拥塞窗口 cwnd / 2；

(2) 新拥塞窗口 cwnd = 慢开始门限 ssthresh；

(3) 开始执行拥塞避免算法，使拥塞窗口缓慢地线性增大。

慢开始和拥塞避免算法的实现举例

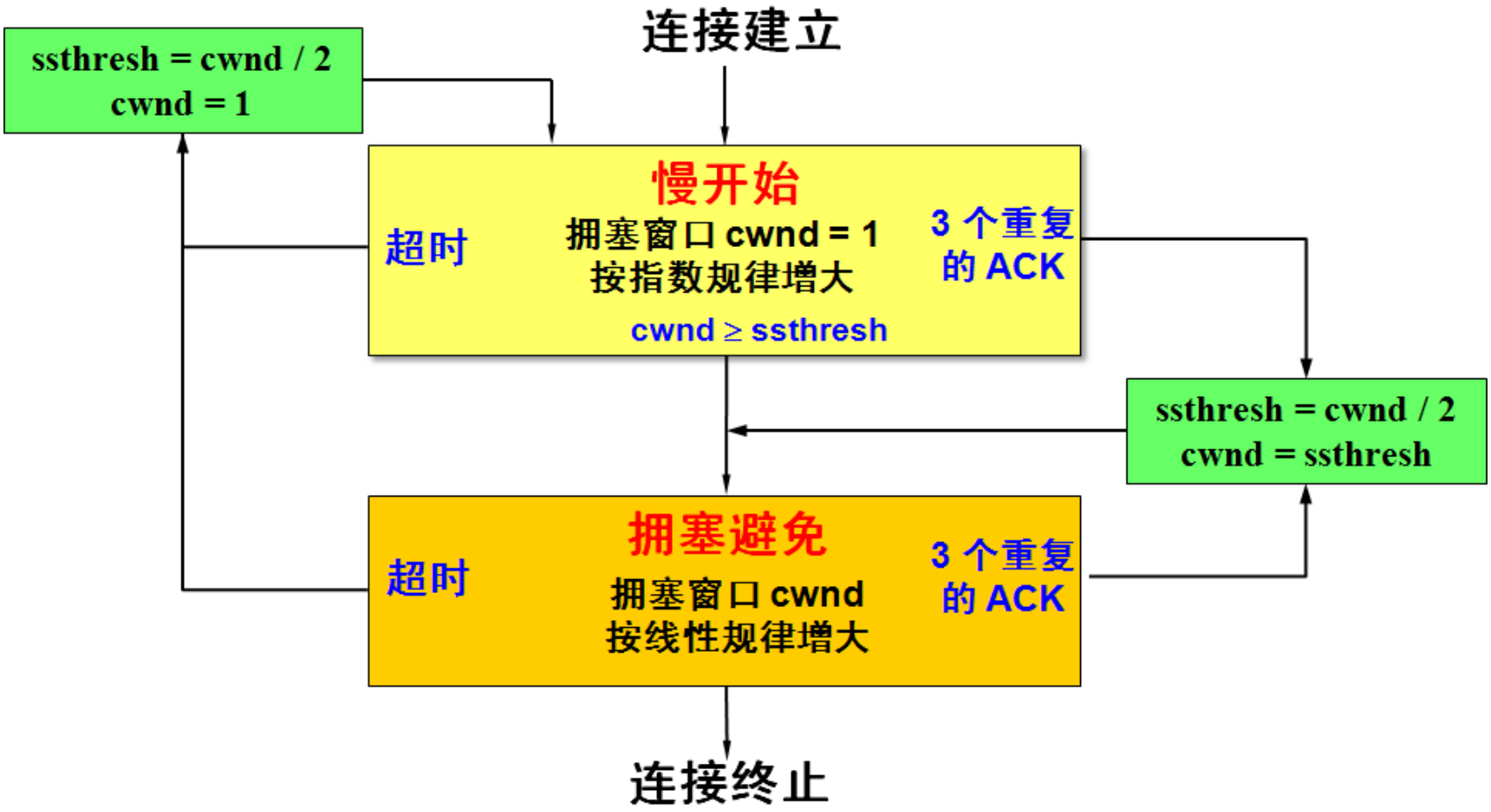


因此，在图的点❹，发送方知道现在只是丢失了个别的报文段。于是不启动慢开始，而是执行快恢复算法。这时，发送方调整门限值ssthresh = cwnd / 2 = 8，同时设置拥塞窗口cwnd = ssthresh = 8（见图中的点❺），并开始执行拥塞避免算法。

加法增大，乘法减小 (AIMD)

- 可以看出，在拥塞避免阶段，拥塞窗口是按照线性规律增大的。这常称为“加法增大” AI (Additive Increase)。
- 当出现超时或3个重复的确认时，就要把门限值设置为当前拥塞窗口值的一半，并大大减小拥塞窗口的数值。这常称为“乘法减小”MD (Multiplicative Decrease)。
- 二者合在一起就是所谓的 AIMD 算法。

TCP拥塞控制流程图


连接建立
ssthresh = cwnd / 2
cwnd = 1
慢开始
拥塞窗口 cwnd = 1
按指数规律增大
cwnd ≥ ssthresh
超时
3 个重复
的 ACK
ssthresh = cwnd / 2
cwnd = ssthresh
拥塞避免
拥塞窗口 cwnd
按线性规律增大
超时
3 个重复
的 ACK
连接终止

发送窗口的上限值

- 发送方的发送窗口的上限值应当取为接收方窗口 rwnd 和拥塞窗口 cwnd 这两个变量中较小的一个，即应按以下公式确定：

发送窗口的上限值 = Min [rwnd, cwnd]

- 当 rwnd < cwnd 时，是接收方的接收能力限制发送窗口的最大值。
- 当 cwnd < rwnd 时，则是网络的拥塞限制发送窗口的最大值。

5-9 TCP 的运输连接管理

1. 运输连接的三个阶段

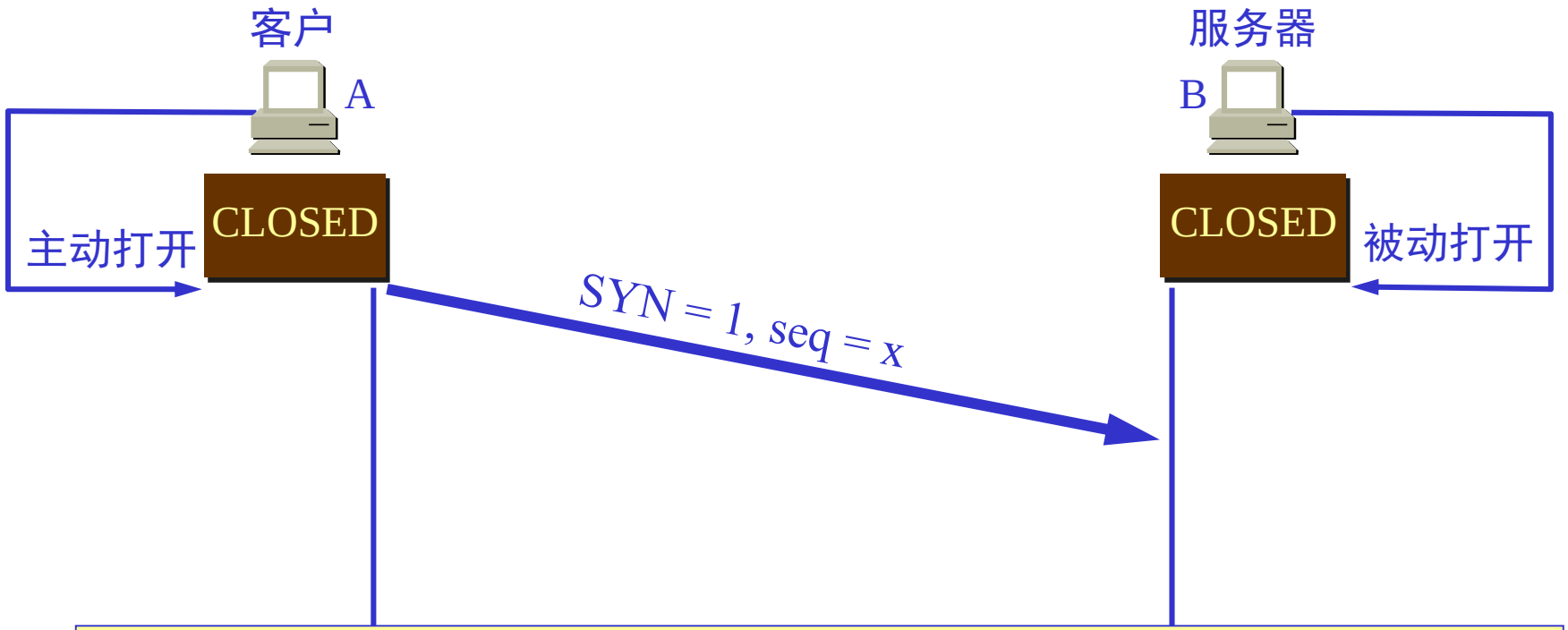
- 运输连接就有三个阶段，即：连接建立、数据传送和连接释放。运输连接的管理就是使运输连接的建立和释放都能正常地进行。

客户服务器方式

- TCP 连接的建立都是采用客户服务器方式。
- 主动发起连接建立的应用进程叫做客户(client)。
- 被动等待连接建立的应用进程叫做服务器(server)。

5.9.1 TCP 的连接建立

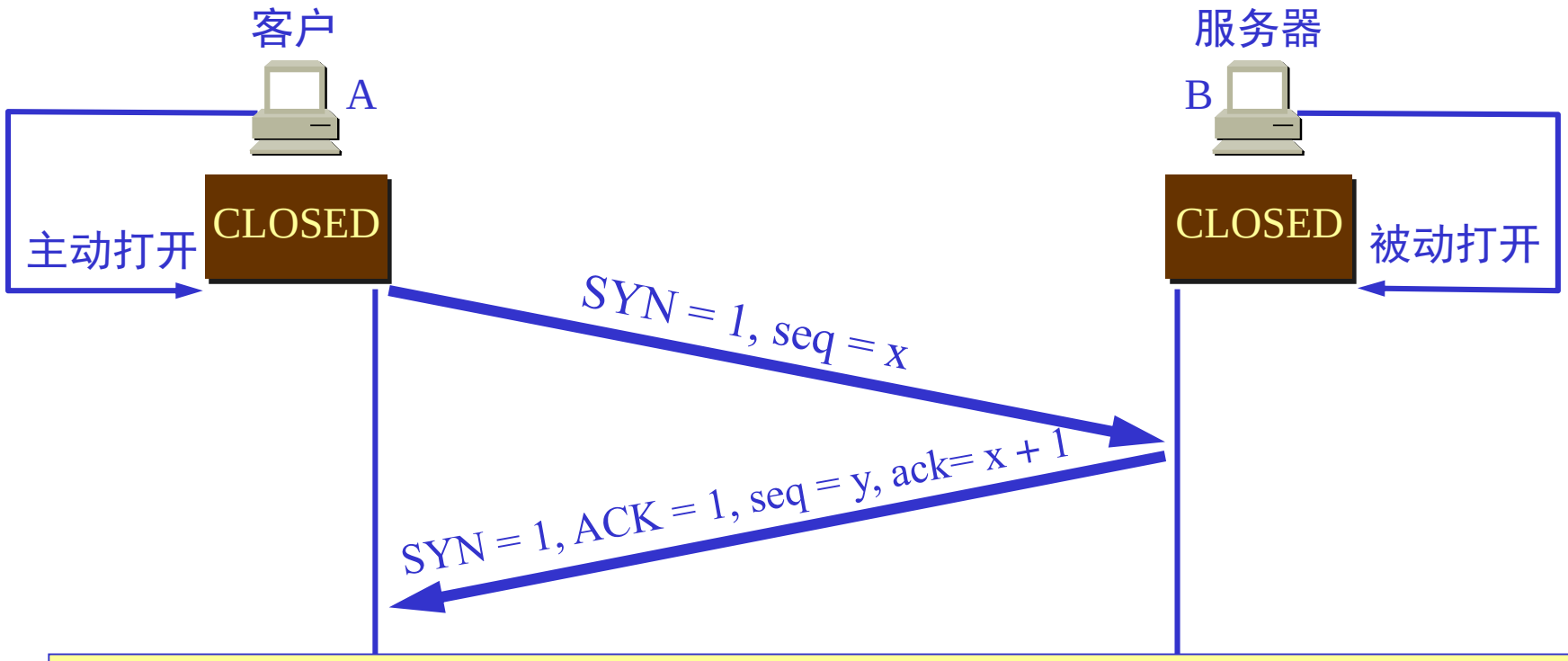
用三报文握手建立 TCP 连接



A 的 TCP 向 B 发出连接请求报文段，其首部中的同步位 SYN = 1，并选择序号 seq = x，表明传送数据时的第一个数据字节的序号是 x。

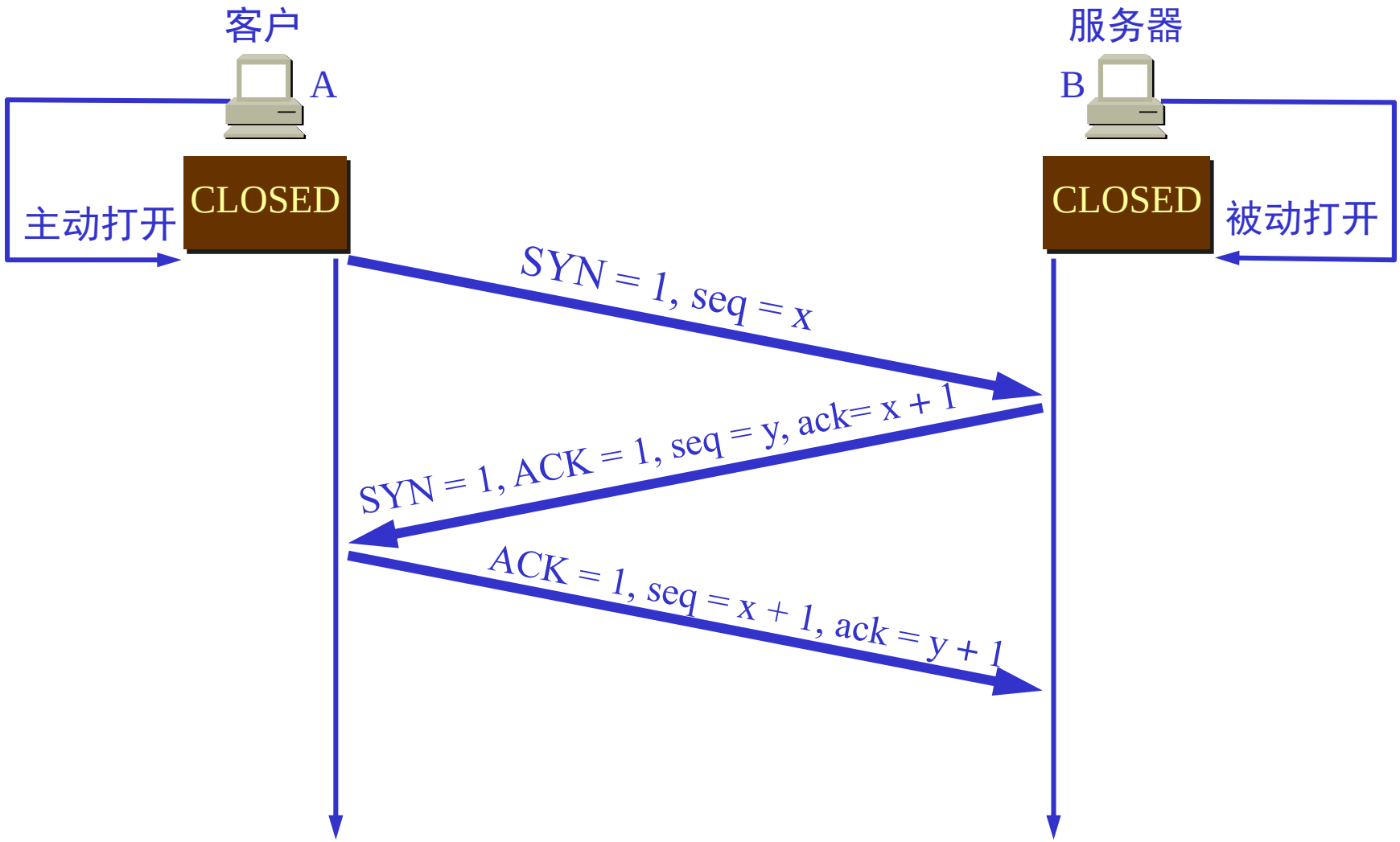
5.9.1 TCP 的连接建立

用三报文握手建立 TCP 连接

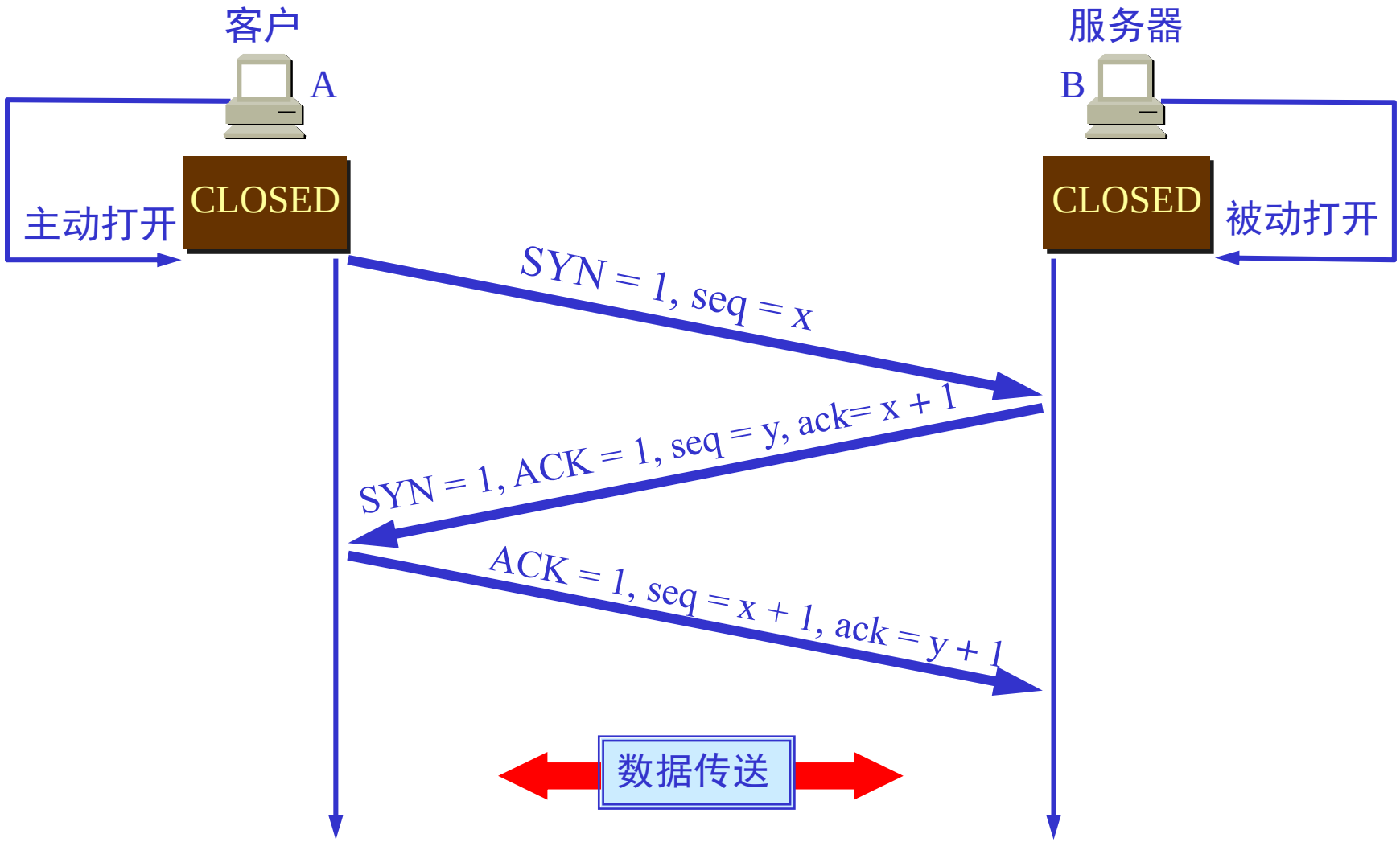


- B 的 TCP 收到连接请求报文段后，如同意，则发回确认。
- B 在确认报文段中应使 SYN = 1，使 ACK = 1，其确认号ack = x + 1，自己选择的序号 seq = y。

- A 收到此报文段后向 B 给出确认，其 ACK = 1，确认号 ack = y + 1。
- A 的 TCP 通知上层应用进程，连接已经建立。

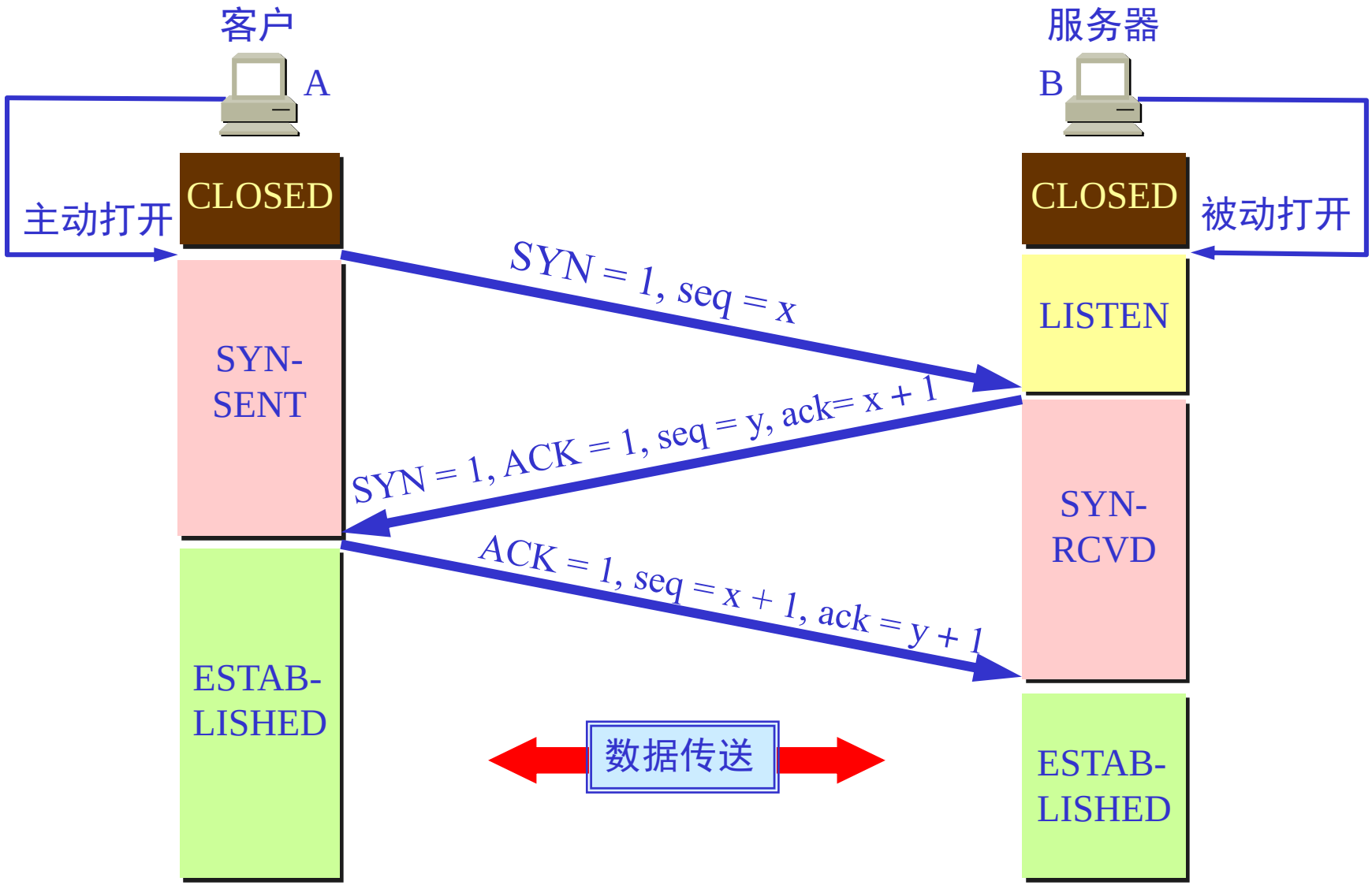


- B 的 TCP 收到主机 A 的确认后，也通知其上层应用进程：TCP 连接已经建立。

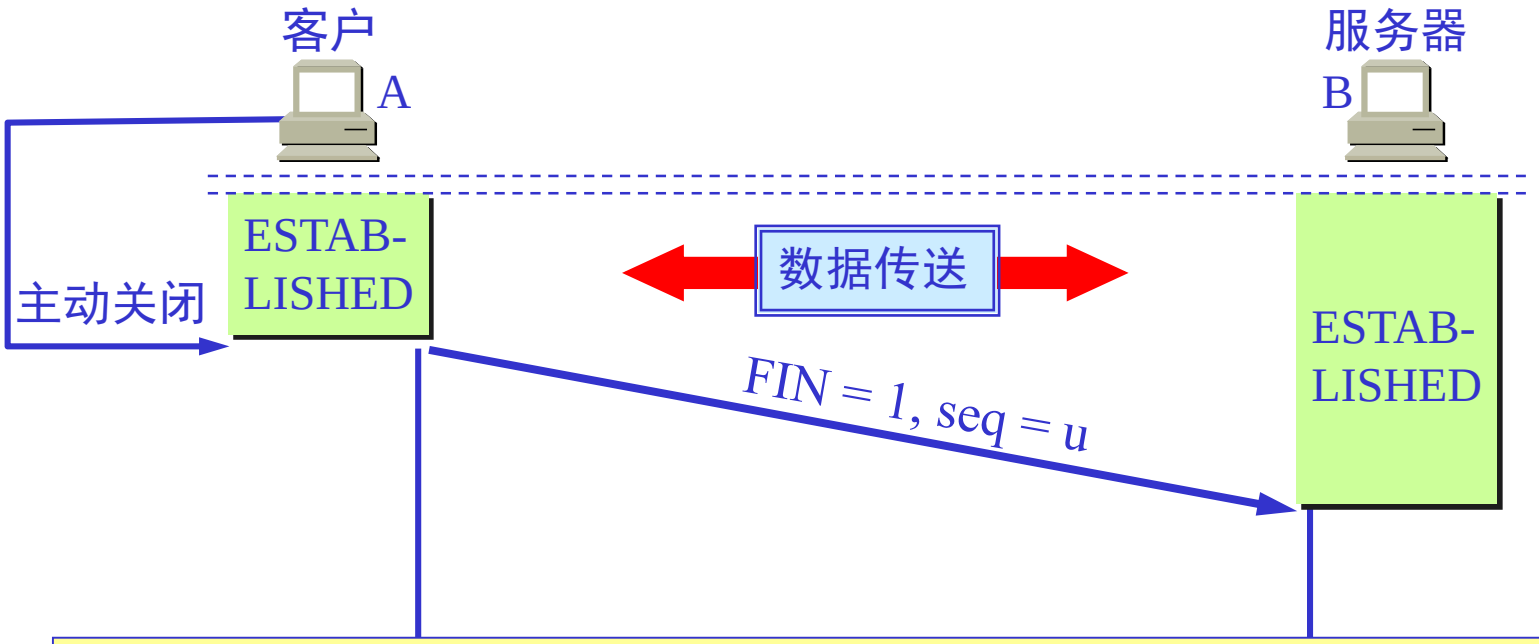


5.9.1 TCP 的连接建立

用三报文握手建立 TCP 连接的各状态



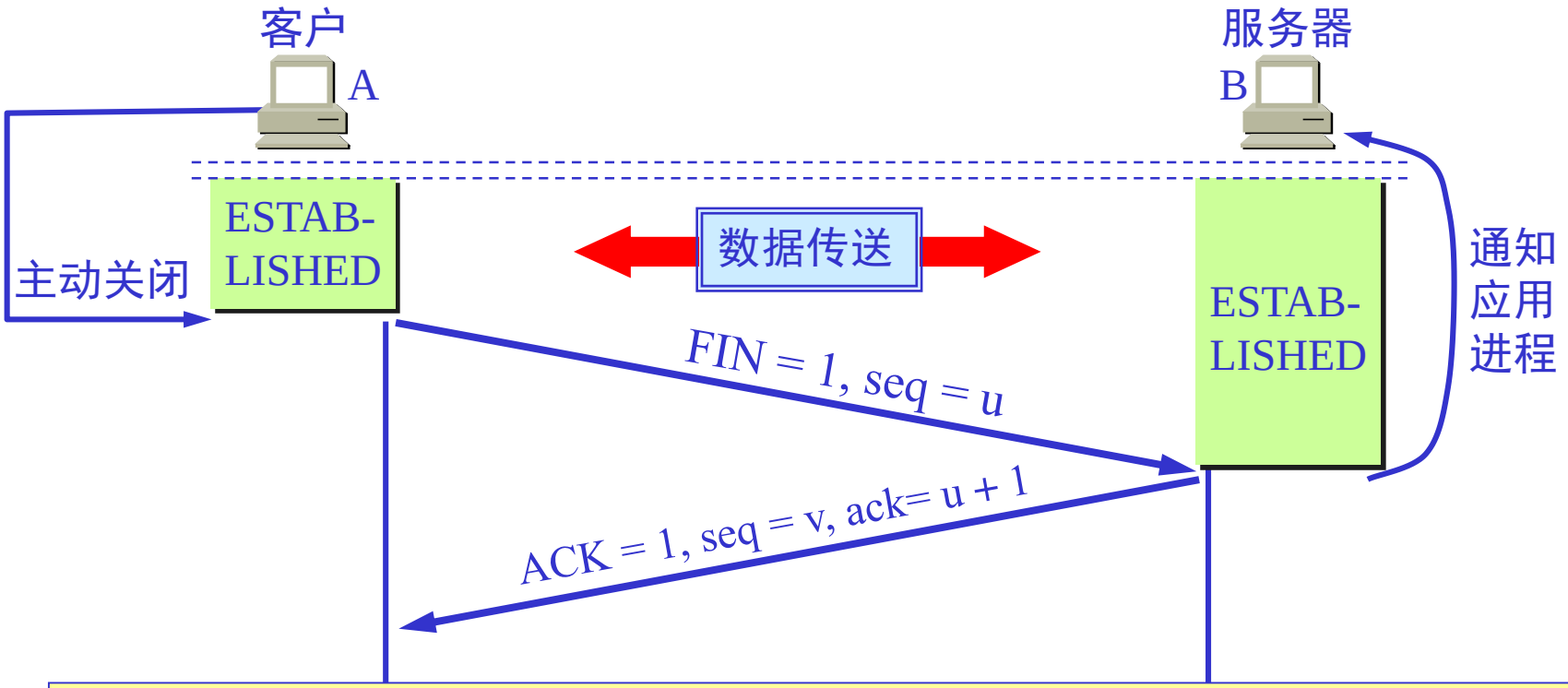
5.9.2 TCP 的连接释放



- 数据传输结束后，通信的双方都可释放连接。现在 A 的应用进程先向其 TCP 发出连接释放报文段，并停止再发送数据，主动关闭 TCP 连接。
- A 把连接释放报文段首部的 FIN = 1，其序号 seq = u，等待 B 的确认。

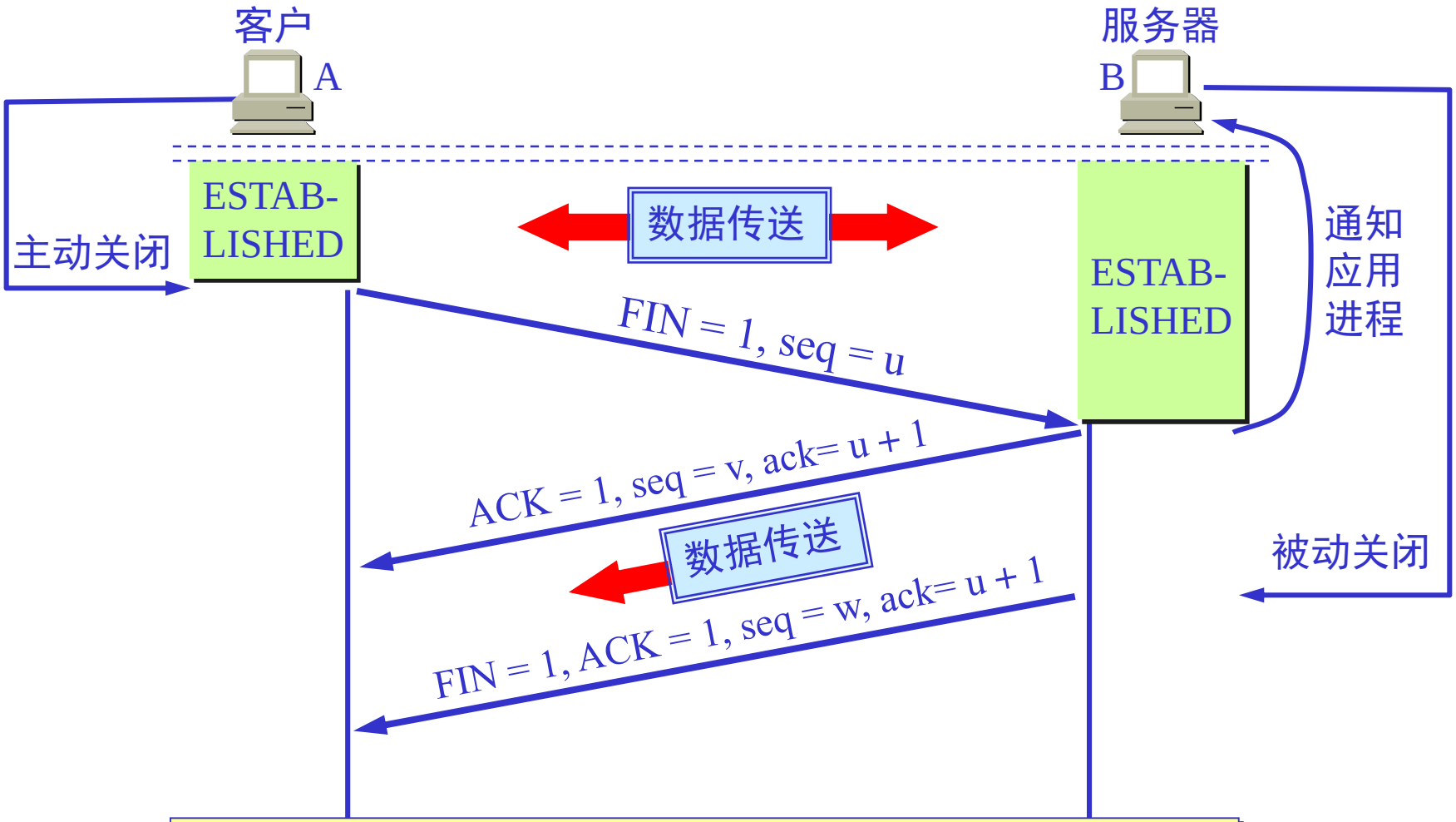
CLOSED

5.9.2 TCP 的连接释放



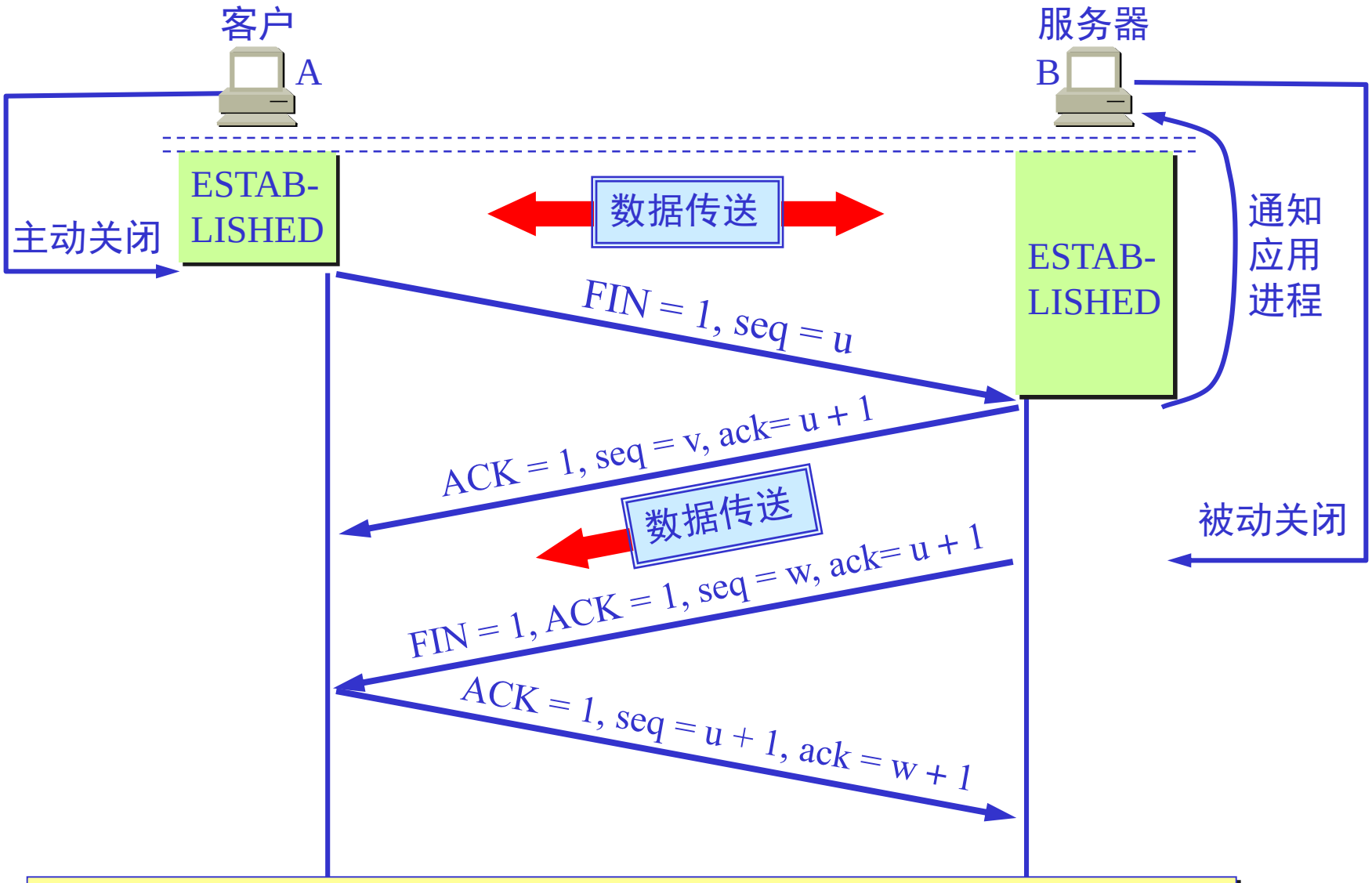
- B 发出确认，确认号 ack = u + 1，而这个报文段自己的序号 seq = v。
- TCP 服务器进程通知高层应用进程。
- 从 A 到 B 这个方向的连接就释放了，TCP 连接处于半关闭状态。B 若发送数据，A 仍要接收。

5.9.2 TCP 的连接释放



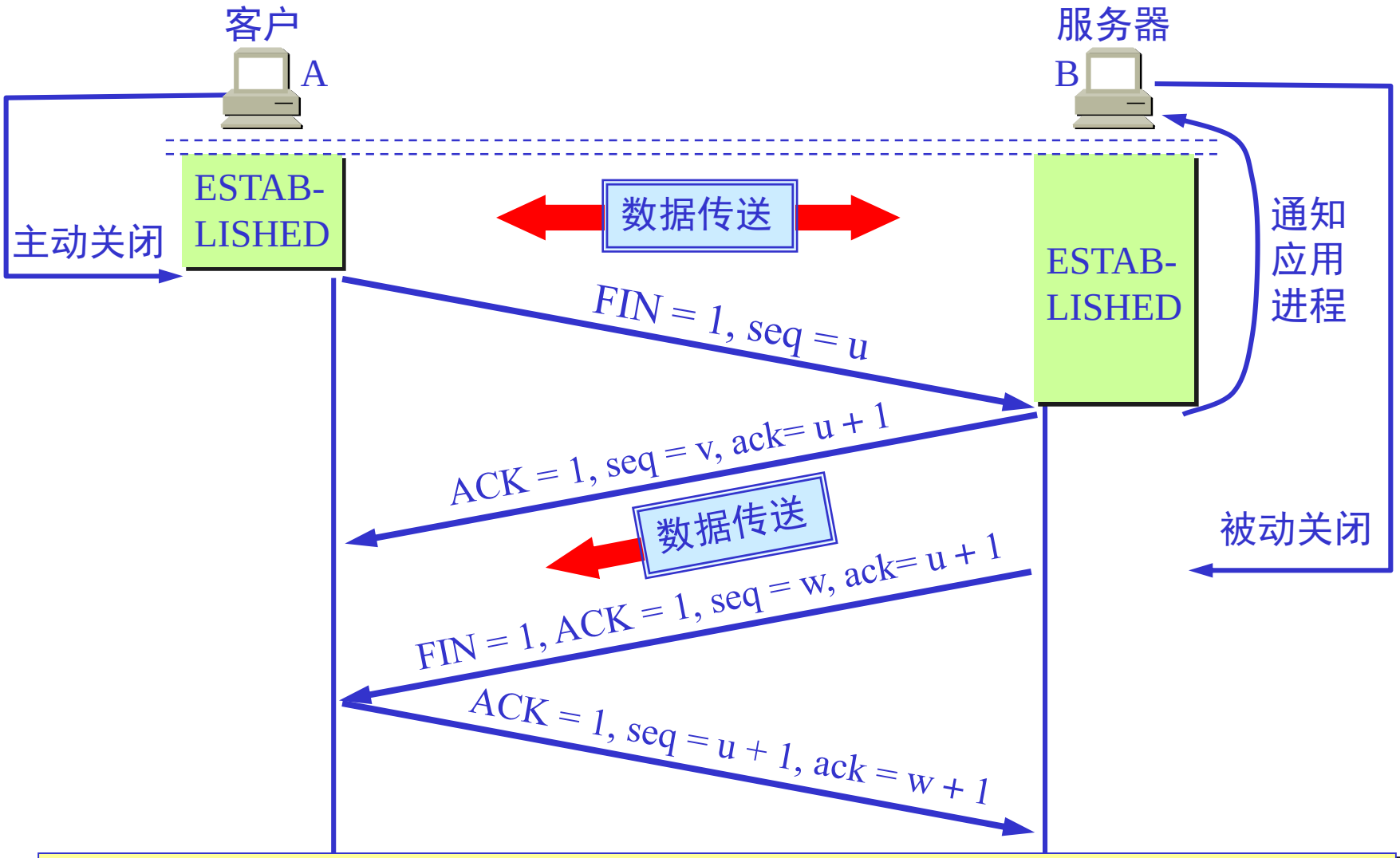
- 若 B 已经没有要向 A 发送的数据，其应用进程就通知 TCP 释放连接。

5.9.2 TCP 的连接释放



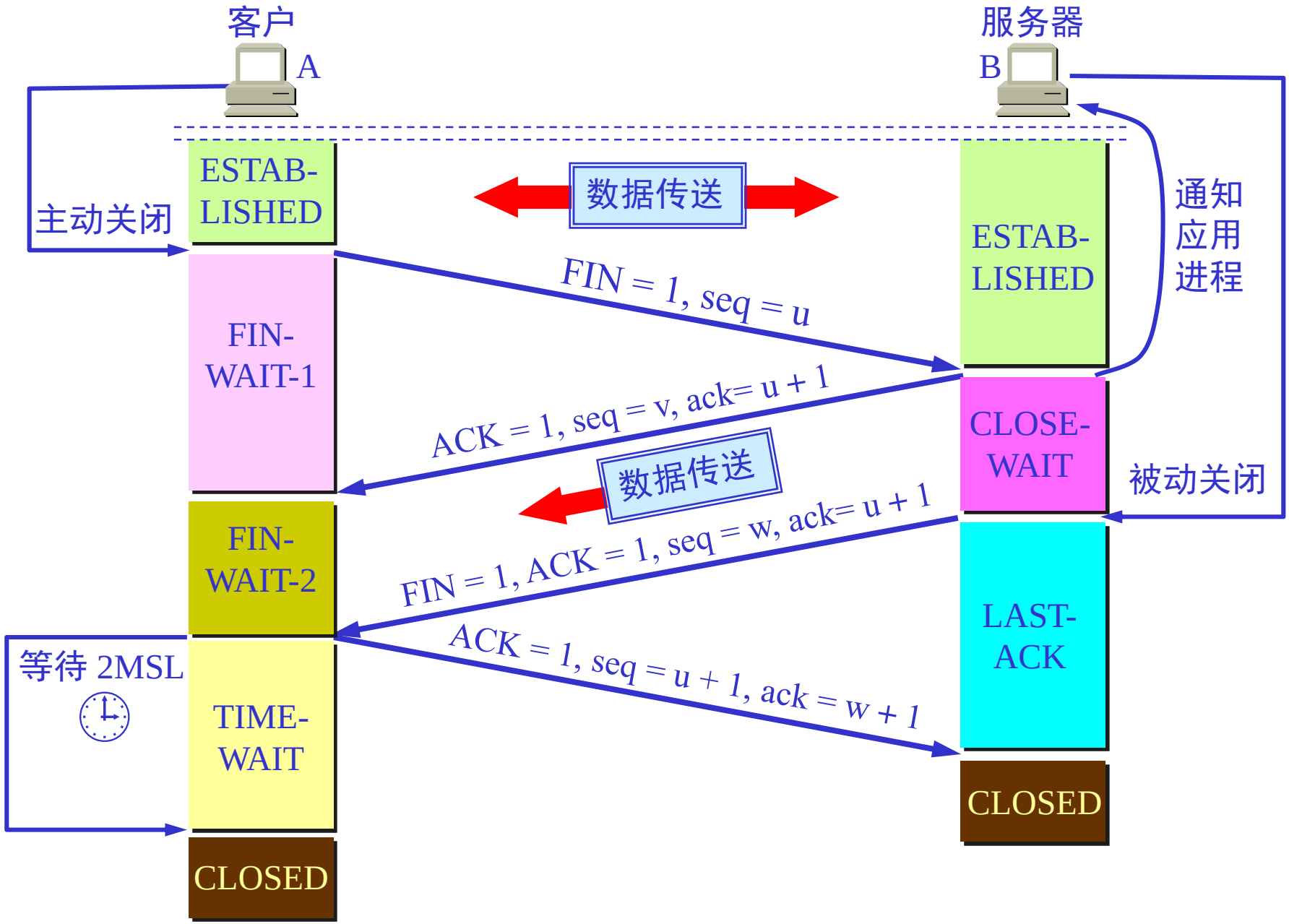
- A 收到连接释放报文段后，必须发出确认。

5.9.2 TCP 的连接释放



- 在确认报文段中 ACK = 1，确认号 ack = w + 1，自己的序号 seq = u + 1。

TCP 连接必须经过时间 2MSL 后才真正释放掉。



A 必须等待 2MSL 的时间

- 第一，为了保证 A 发送的最后一个 ACK 报文段能够到达 B。
- 第二，防止 “已失效的连接请求报文段”出现在本连接中。A 在发送完最后一个 ACK 报文段后，再经过时间 2MSL，就可以使本连接持续的时间内所产生的所有报文段，都从网络中消失。

课后作业

- 习题：5-37,5-38,5-39,5-41,5-46