第5章 运输层

河北师范大学软件学院
Software College of Hebei Normal University

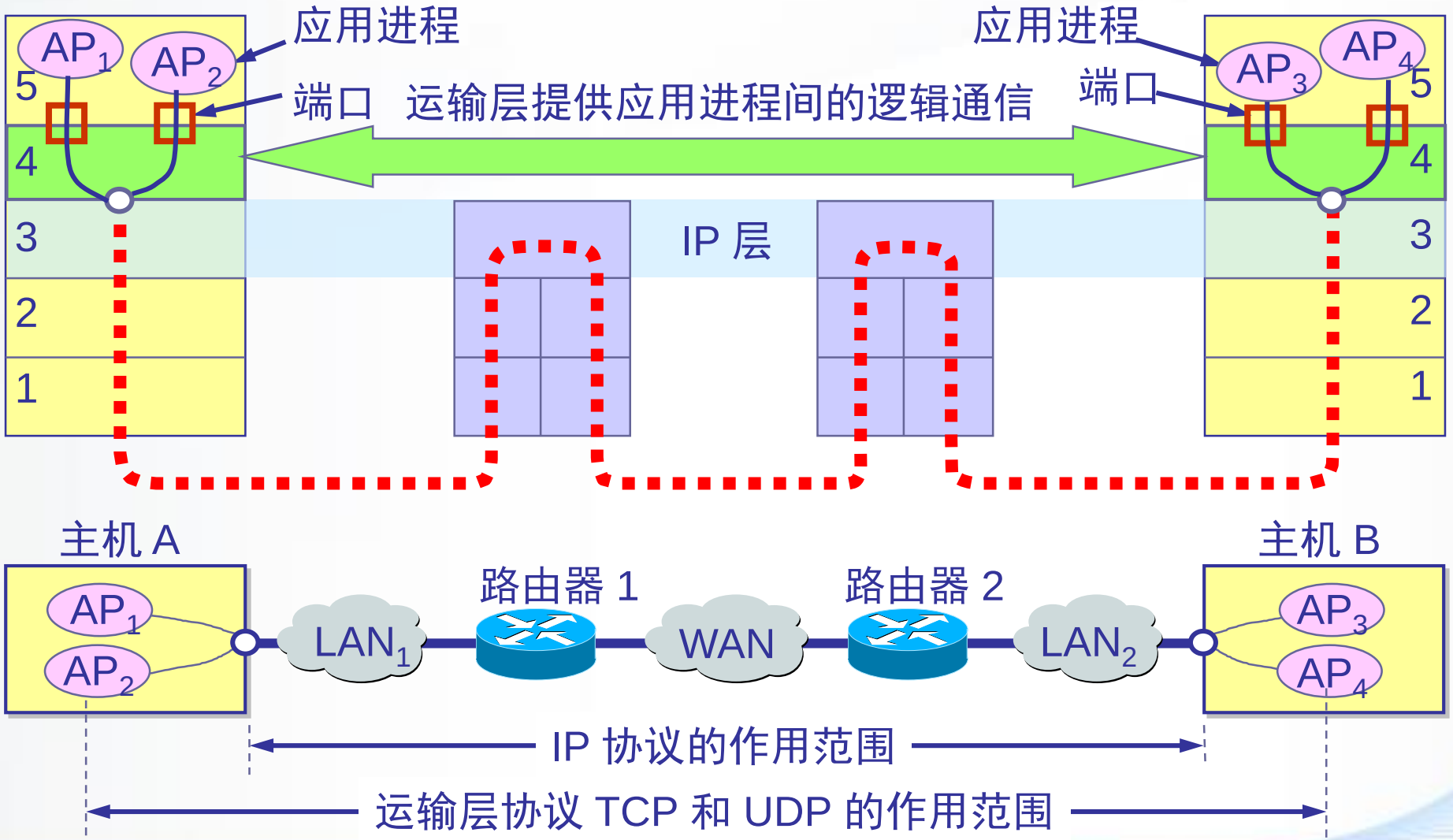
回顾

- 数据报 虚电路 虚拟互连网络
- 分类IP地址 子网划分 构成超网
- IP 地址与物理地址的关系
- ARP RARP IP数据报格式
- RIP OSPF BGP工作原理
- IGMP和多播
- VPN NAT

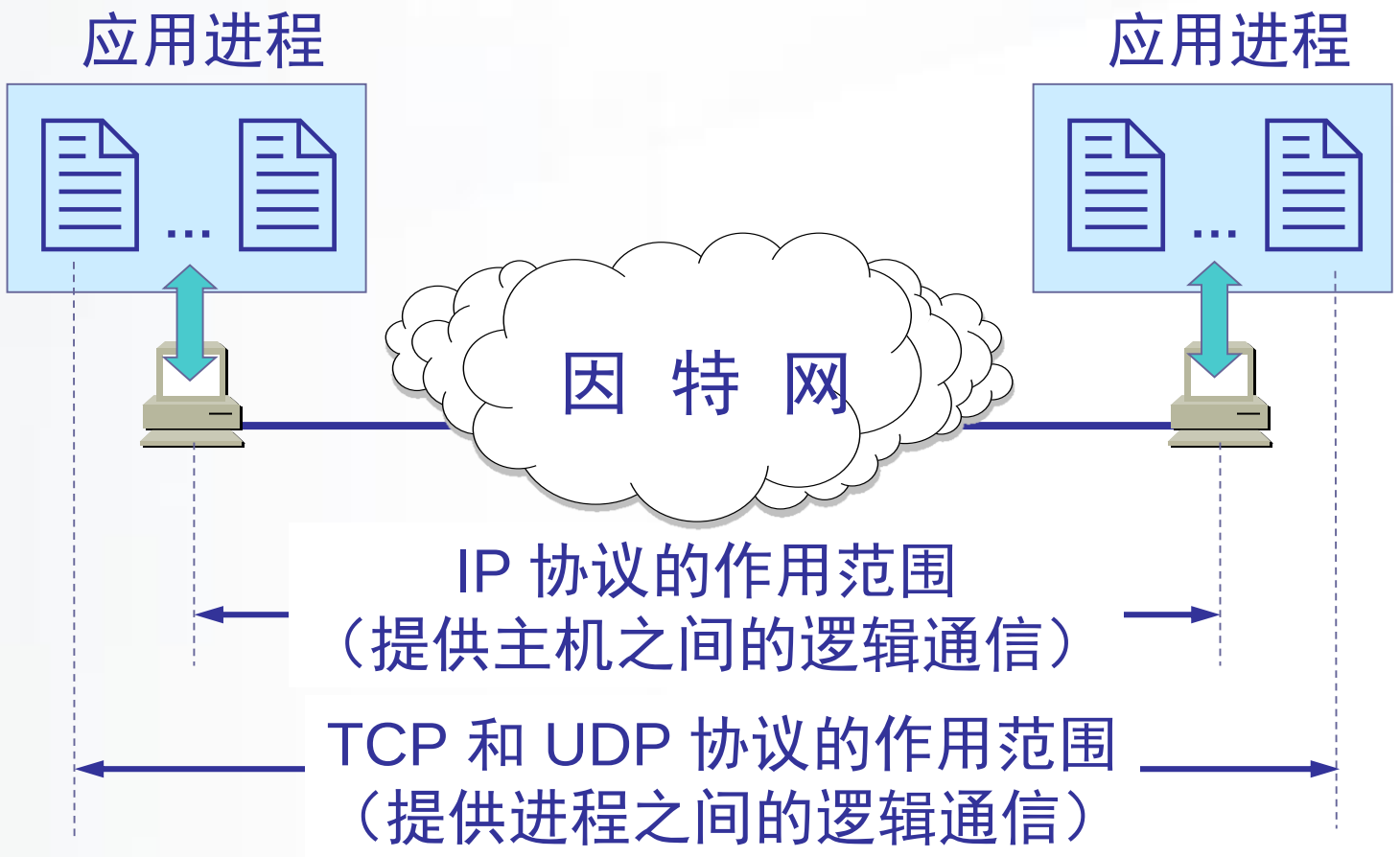
指引

- 运输层的功能
- 运输层协议UDP和TCP
- TCP可靠传输的实现
- TCP的流量控制
- TCP的拥塞控制
- TCP的运输连接管理

为相互通信的应用进程提供逻辑通信



运输层协议和网络层协议的主要区别



运输层的主要功能

- 运输层为应用进程之间提供端到端的逻辑通信（但网络层是为主机之间提供逻辑通信）。
- 运输层还要对收到的报文进行差错检测。
- 运输层提供面向连接和无连接的服务。

指引

- 运输层的功能
- 运输层协议UDP和TCP
- TCP可靠传输的实现
- TCP的流量控制
- TCP的拥塞控制
- TCP的运输连接管理

运输层的两个主要协议

- TCP/IP 的运输层有两个不同的协议：
  - 用户数据报协议 UDP (User Datagram Protocol)
  - 传输控制协议 TCP (Transmission Control Protocol)

TCP 与 UDP

- 两个对等运输实体在通信时传送的数据单位叫作运输协议数据单元 TPDU (Transport Protocol Data Unit)。
  - TCP 传送的协议数据单元是 TCP 报文段(segment)
  - UDP 传送的协议数据单元是 UDP 报文或用户数据报

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 则提供面向连接的服务。TCP 不提供广播或多播服务。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。这不仅使协议数据单元的首部增大很多，还要占用许多的处理机资源。

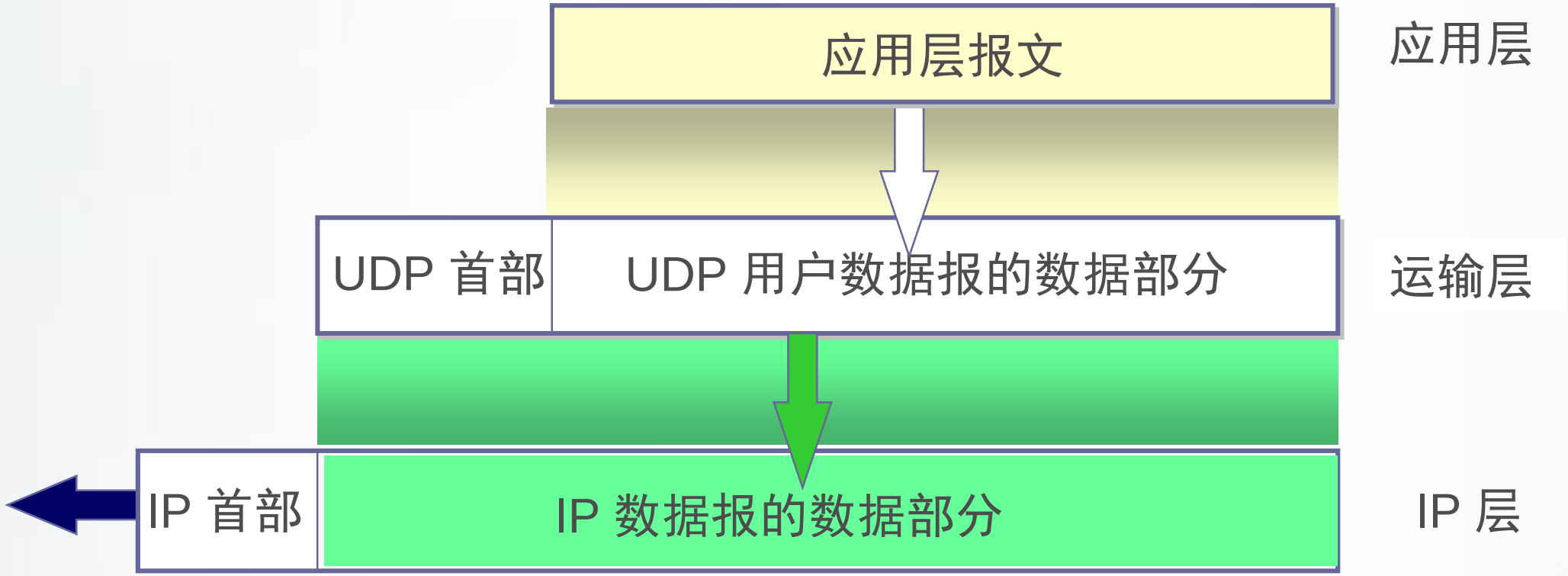
软件端口与硬件端口

- 在协议栈层间的抽象的协议端口是软件端口。
- 路由器或交换机上的端口是硬件端口。
- 传输层端口用一个 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。在因特网中不同计算机的相同端口号是没有联系的。

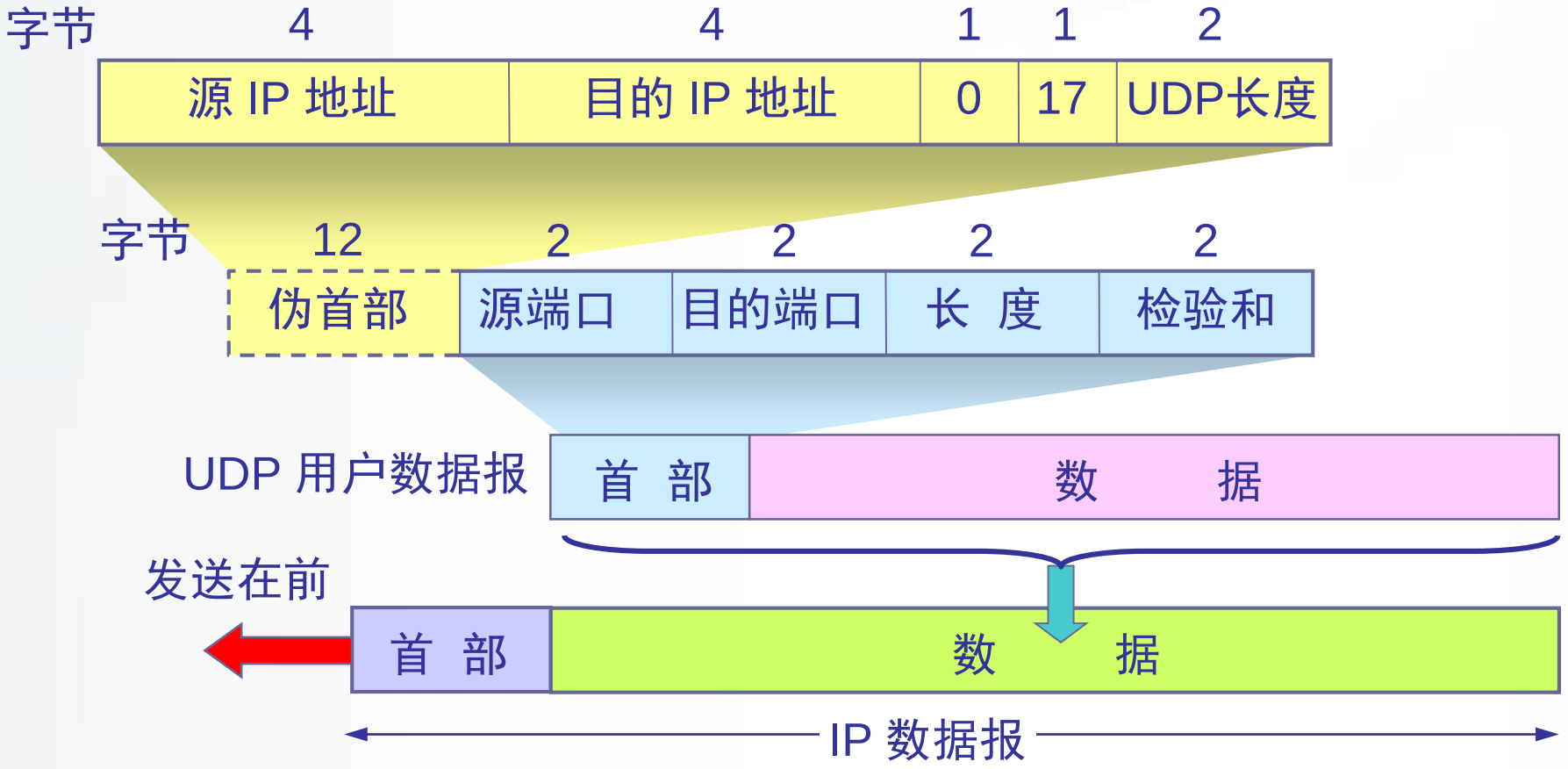
三类端口

- 熟知端口，数值一般为 0～1023。
  - FTP:21
  - TELNET:23
  - SMTP:25
  - DNS:53
  - HTTP:80
- 登记端口号，数值为1024～49151
- 客户端口号，数值为49152～65535

用户数据报协议 UDP



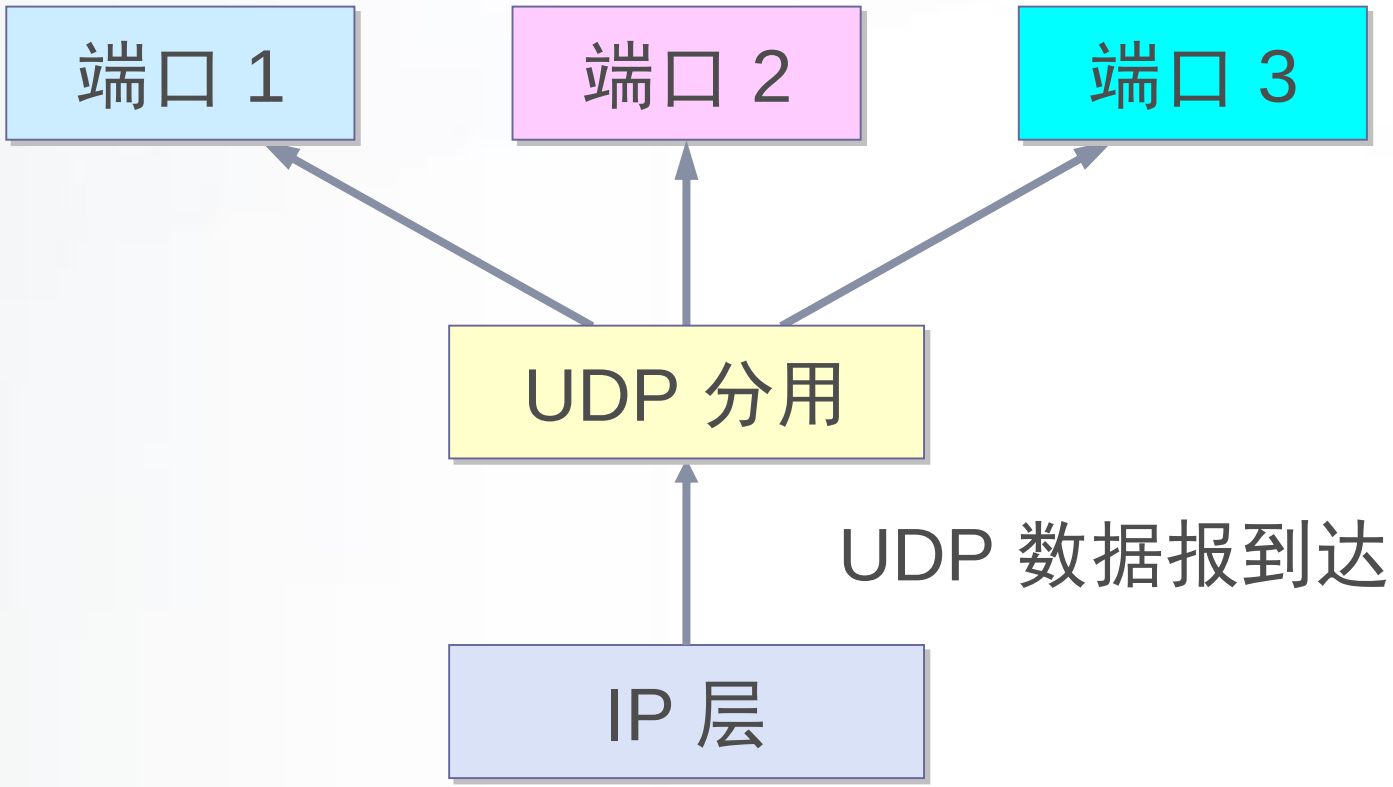
UDP 的首部格式


字节
4
4
1
1
2
源 IP 地址
目的 IP 地址
0
17
UDP长度
字节
12
2
2
2
2
伪首部
源端口
目的端口
长 度
检验和
UDP 用户数据报
首 部
数 据
发送在前
首 部
数 据
IP 数据报

UDP 的主要特点

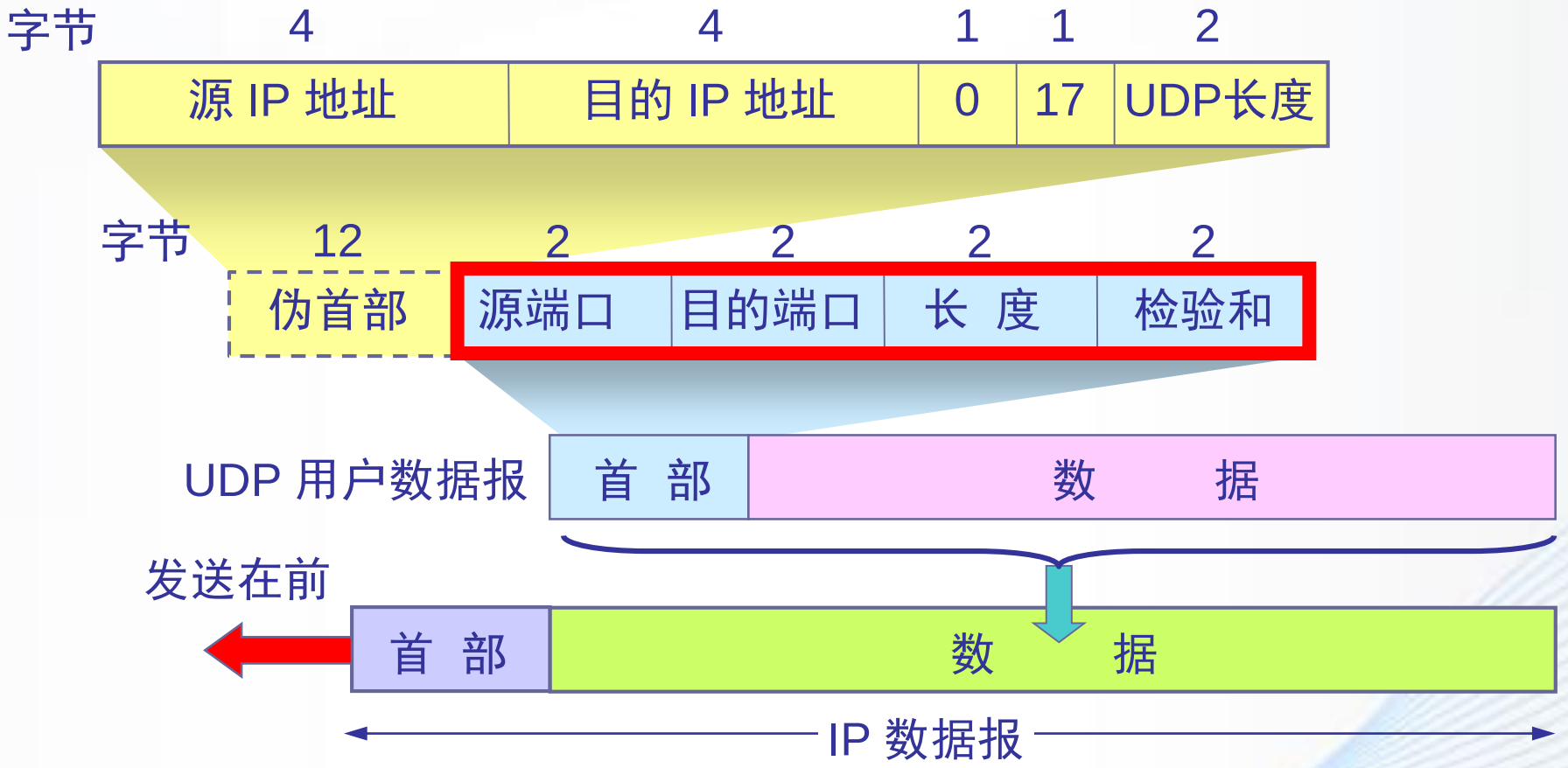
- UDP 是无连接的，即发送数据之前不需要建立连接。
- UDP 使用尽最大努力交付，即不保证可靠交付，同时也不使用拥塞控制。
- UDP 是面向报文的。UDP 没有拥塞控制，很适合多媒体通信的要求。
- UDP 支持一对一、一对多、多对一和多对多的交互通信。
- UDP 的首部开销小，只有 8 个字节。

UDP 基于端口的分用



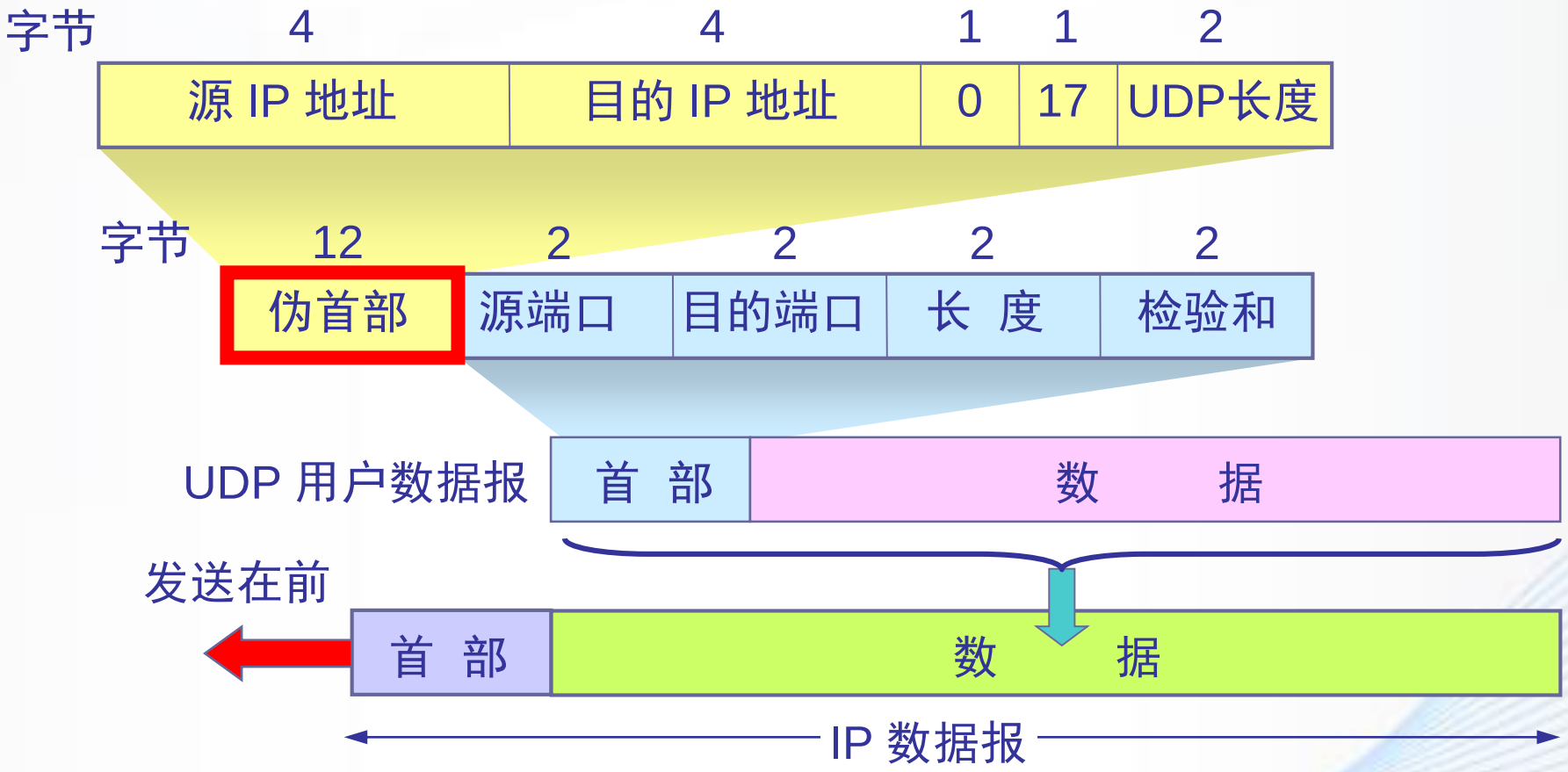
用户数据报 UDP

- UDP 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。

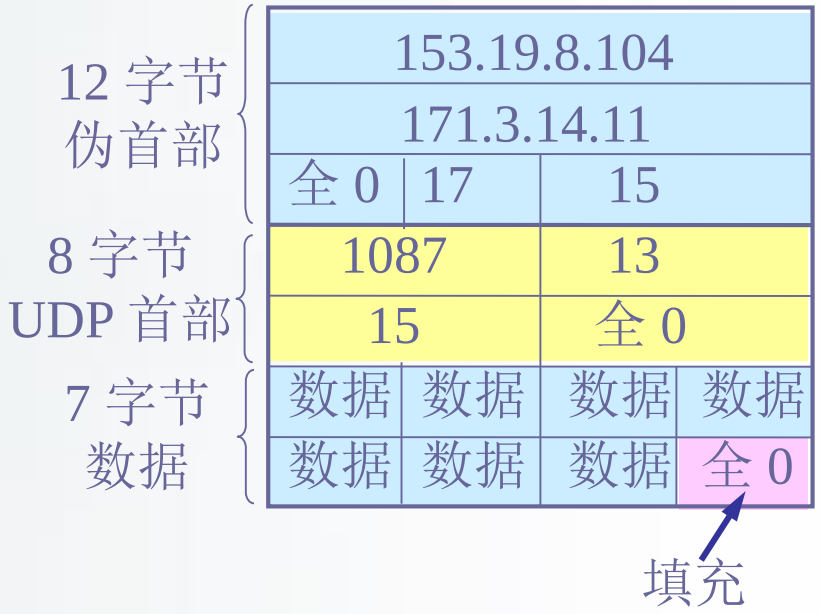


伪首部的作用

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。



计算 UDP 检验和的例子



10011001 00010011 → 153.19
00001000 01101000 → 8.104
10101011 00000011 → 171.3
00001110 00001011 → 14.11
00000000 00010001 → 0 和 17
00000000 00001111 → 15
00000100 00111111 → 1087
00000000 00001101 → 13
00000000 00001111 → 15
00000000 00000000 → 0（检验和）
01010100 01000101 → 数据
01010011 01010100 → 数据
01001001 01001110 → 数据
01000111 00000000 → 数据和 0（填充）

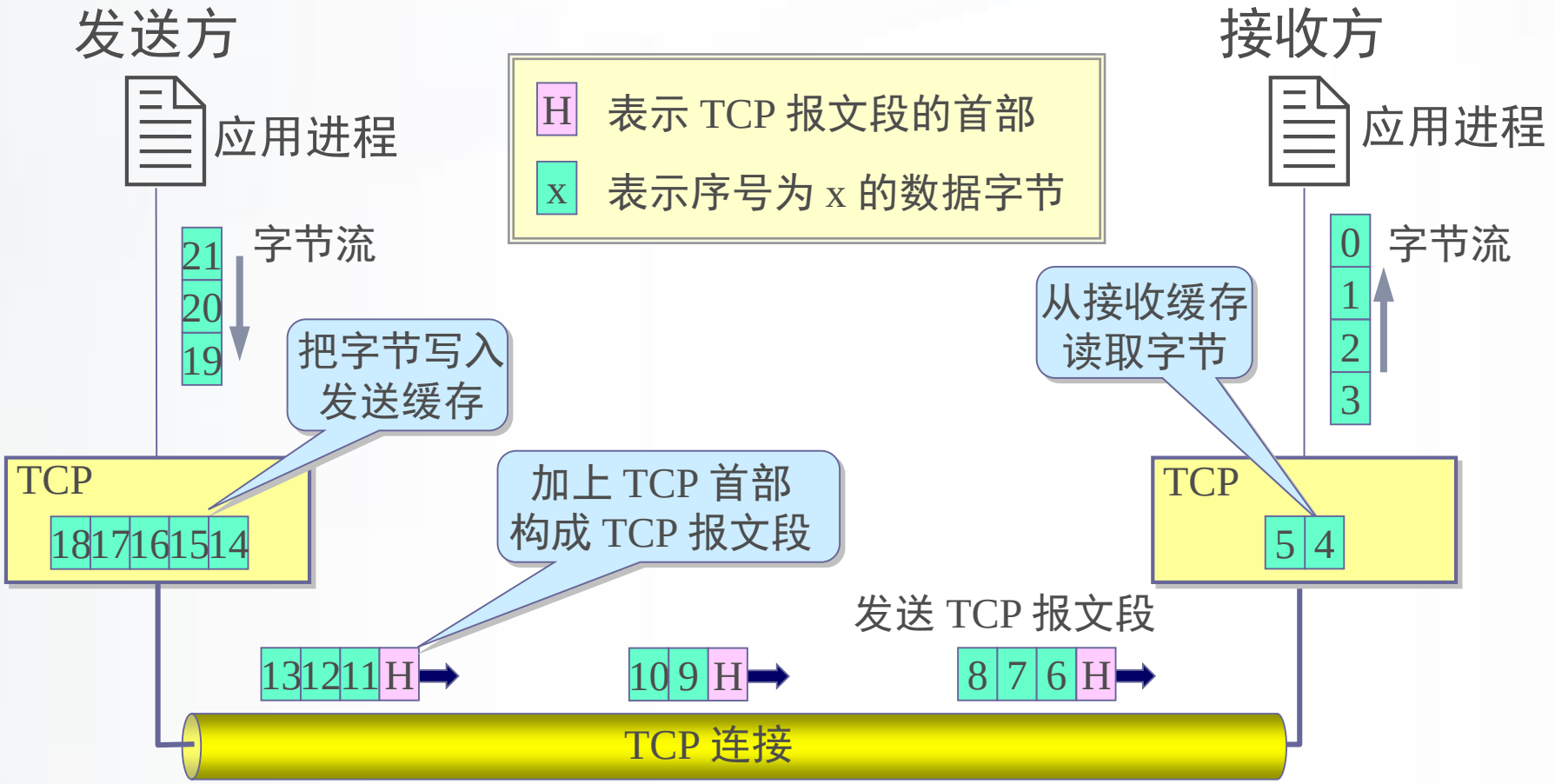
按二进制反码运算求和 10010110 11101101 → 求和得出的结果

将得出的结果求反码 01101001 00010010 → 检验和

传输控制协议TCP概述

- 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。
- TCP 提供全双工通信。
- 面向字节流。

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
发送 TCP 报文段
8 7 6 H
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

TCP 的连接

- TCP 把连接作为最基本的抽象。
- 每一条 TCP 连接有两个端点。
- TCP 连接的端点不是主机，不是主机的IP 地址，不是应用进程，也不是运输层的协议端口。TCP 连接的端点叫做套接字(socket)或插口。
- 端口号拼接到IP 地址即构成了套接字。

TCP 的连接

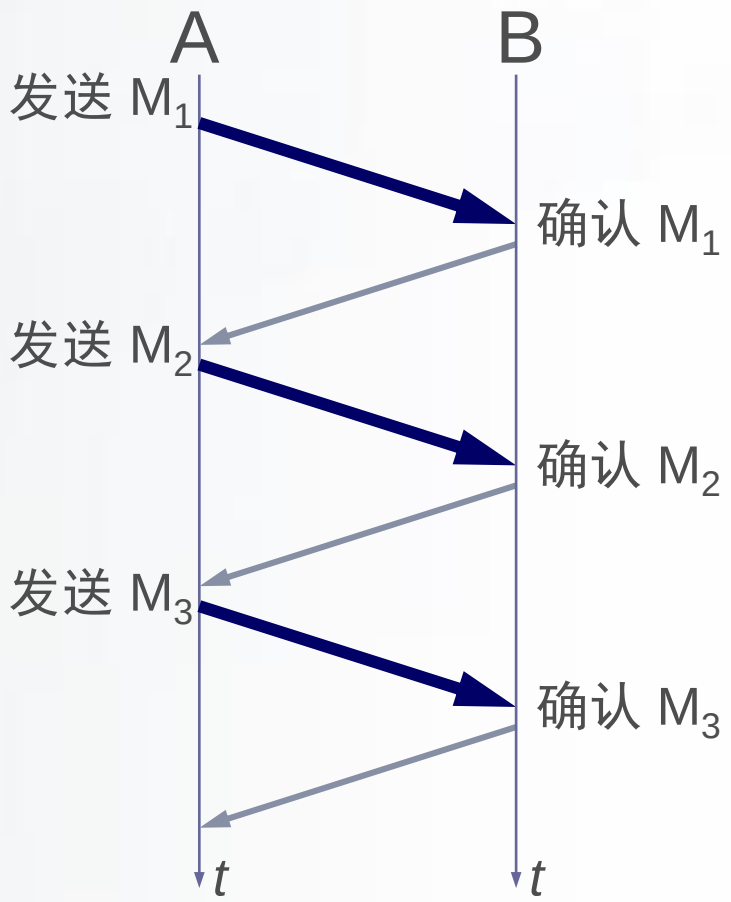
套接字 socket = (IP地址: 端口号)

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。即：

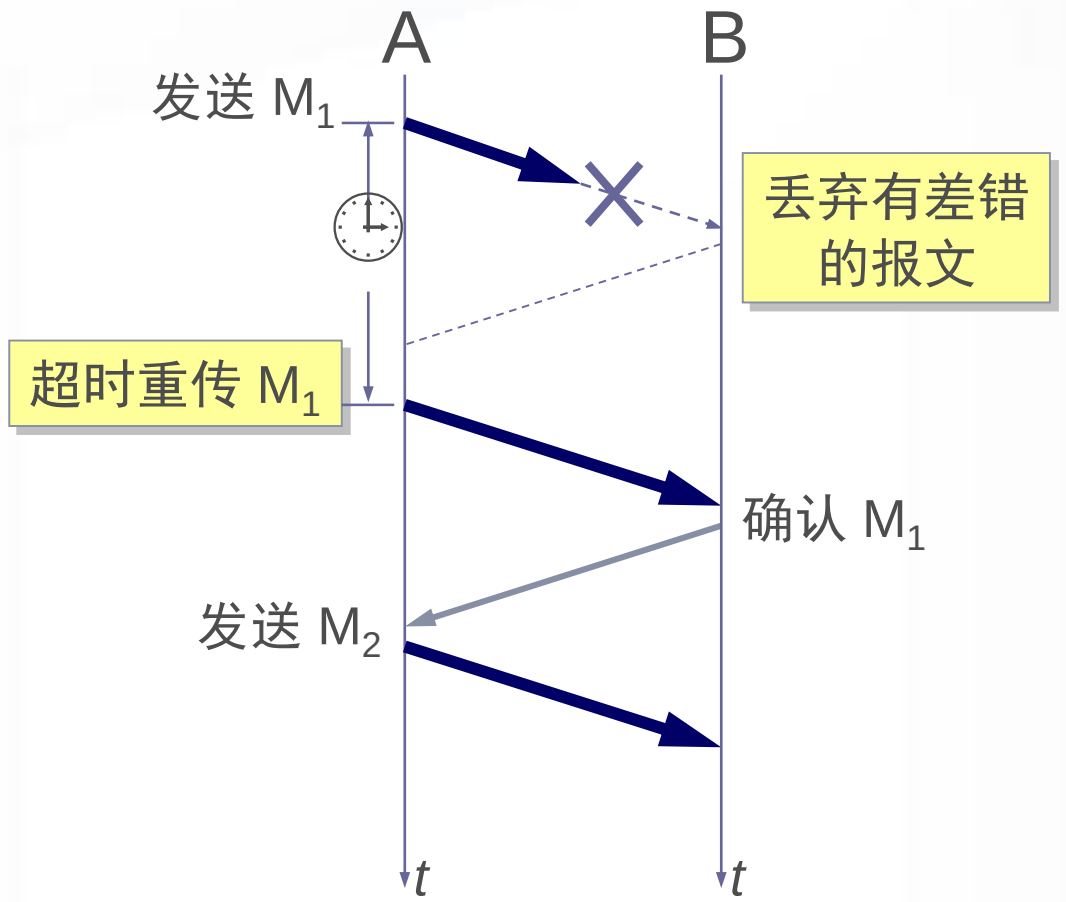
TCP 连接 ::= {socket1, socket2}

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

可靠传输工作原理--停止等待协议



(a) 无差错情况

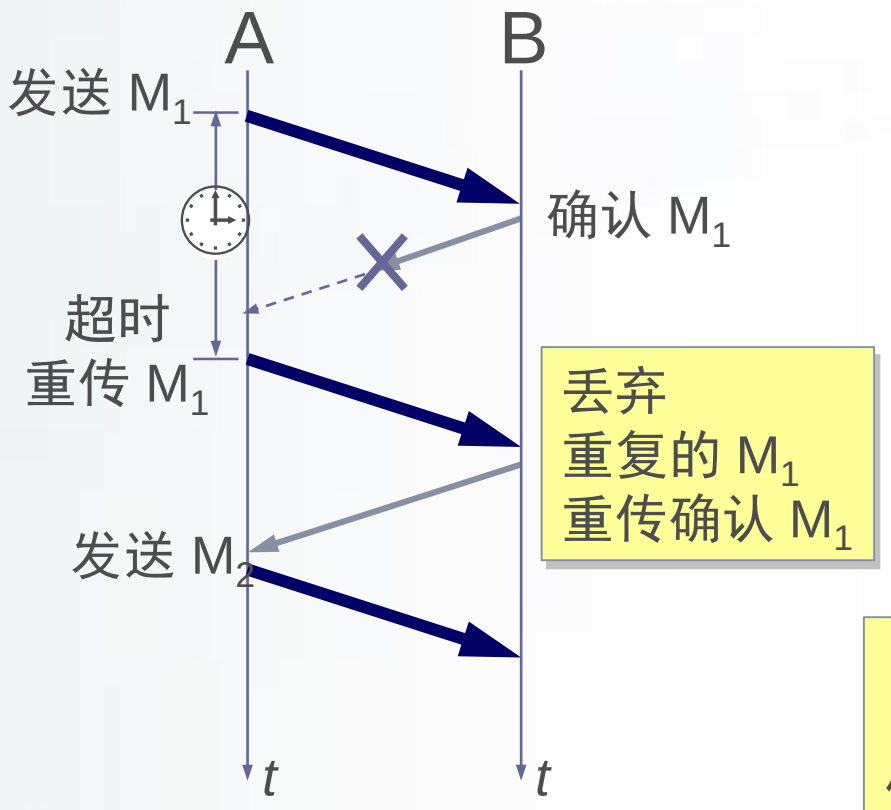


(b) 超时重传

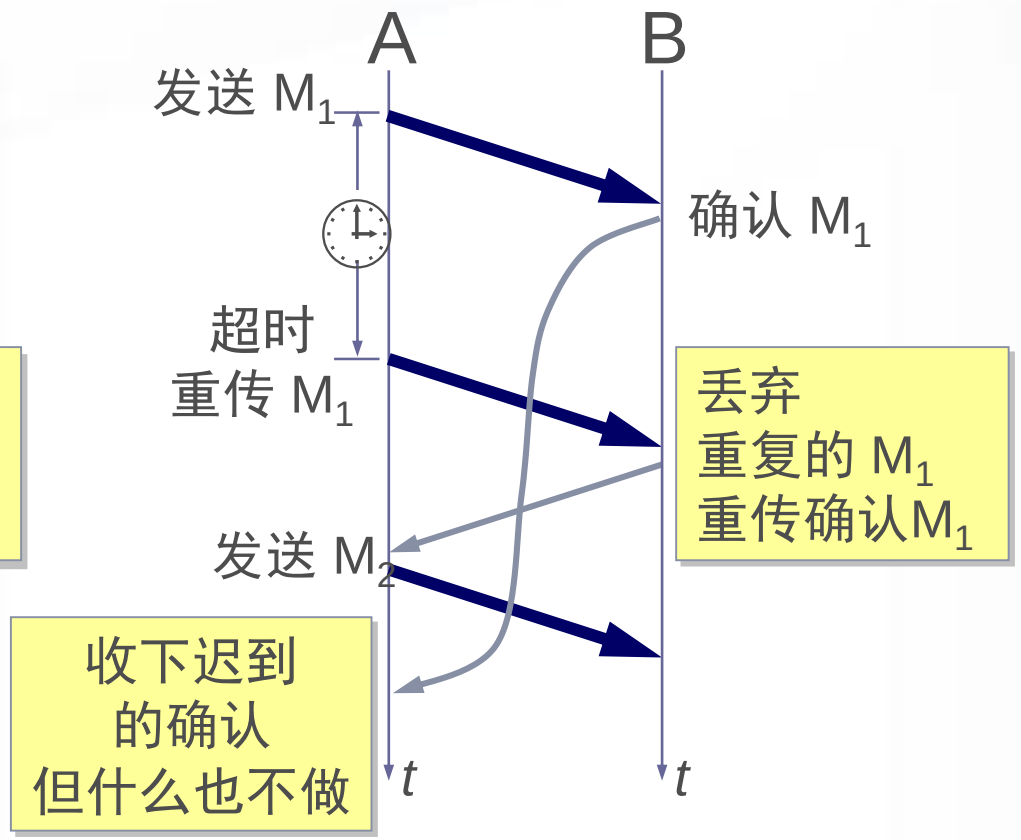
注意

- 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



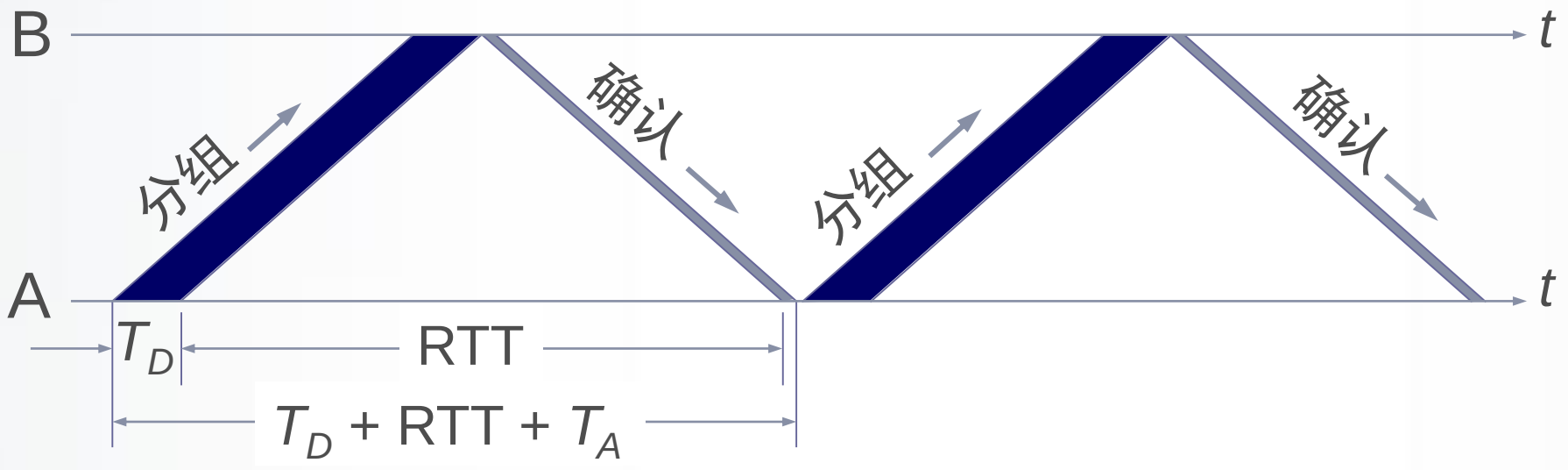
(b) 确认迟到

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。

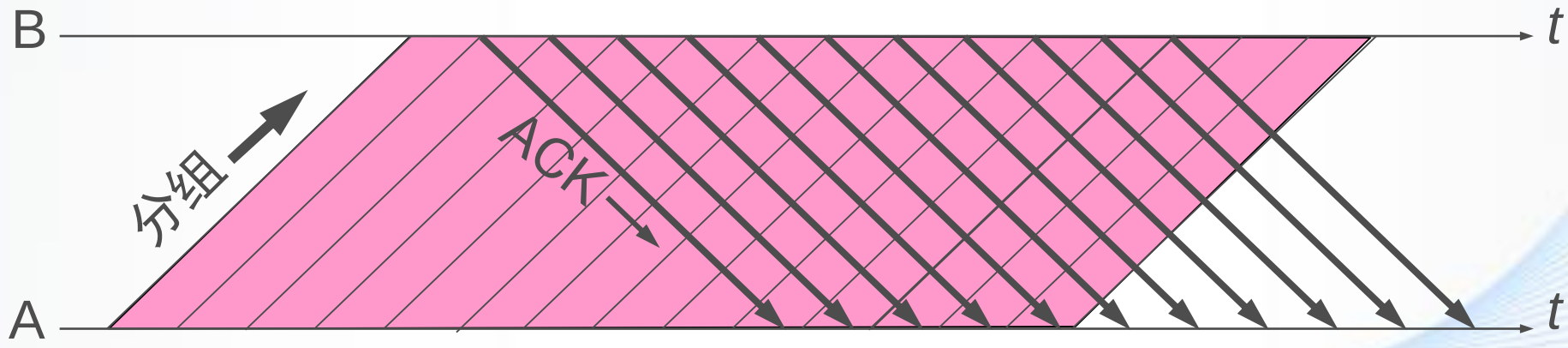


信道的利用率 U

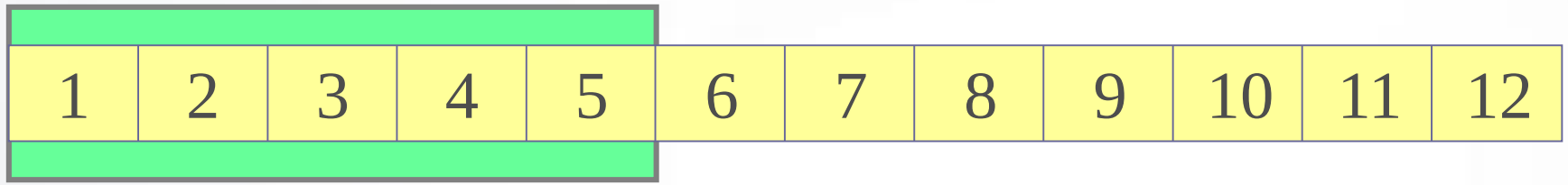
$$U = \frac{T_D}{T_D + \mathrm{RTT} + T_A} \qquad (5\text{-}3)$$

流水线传输

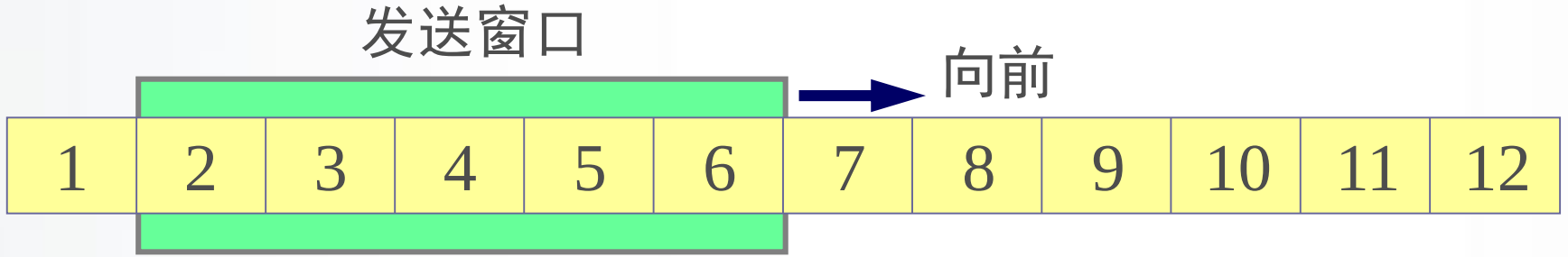
- 发送方可连续发送多个分组，不必每发完一个分组就停顿下来等待对方的确认。
- 由于信道上一直有数据不间断地传送，这种传输方式可获得很高的信道利用率。



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

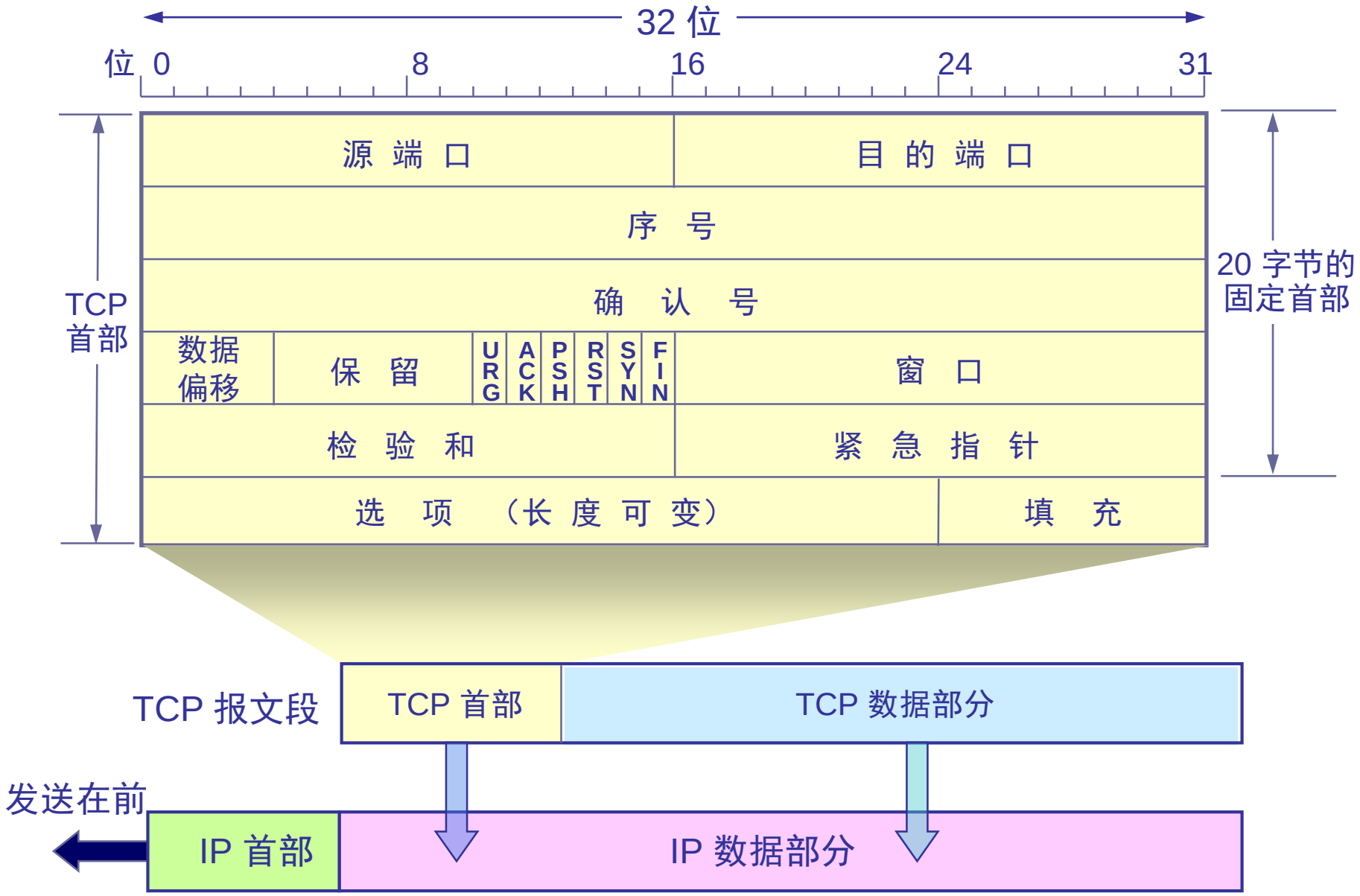
累积确认

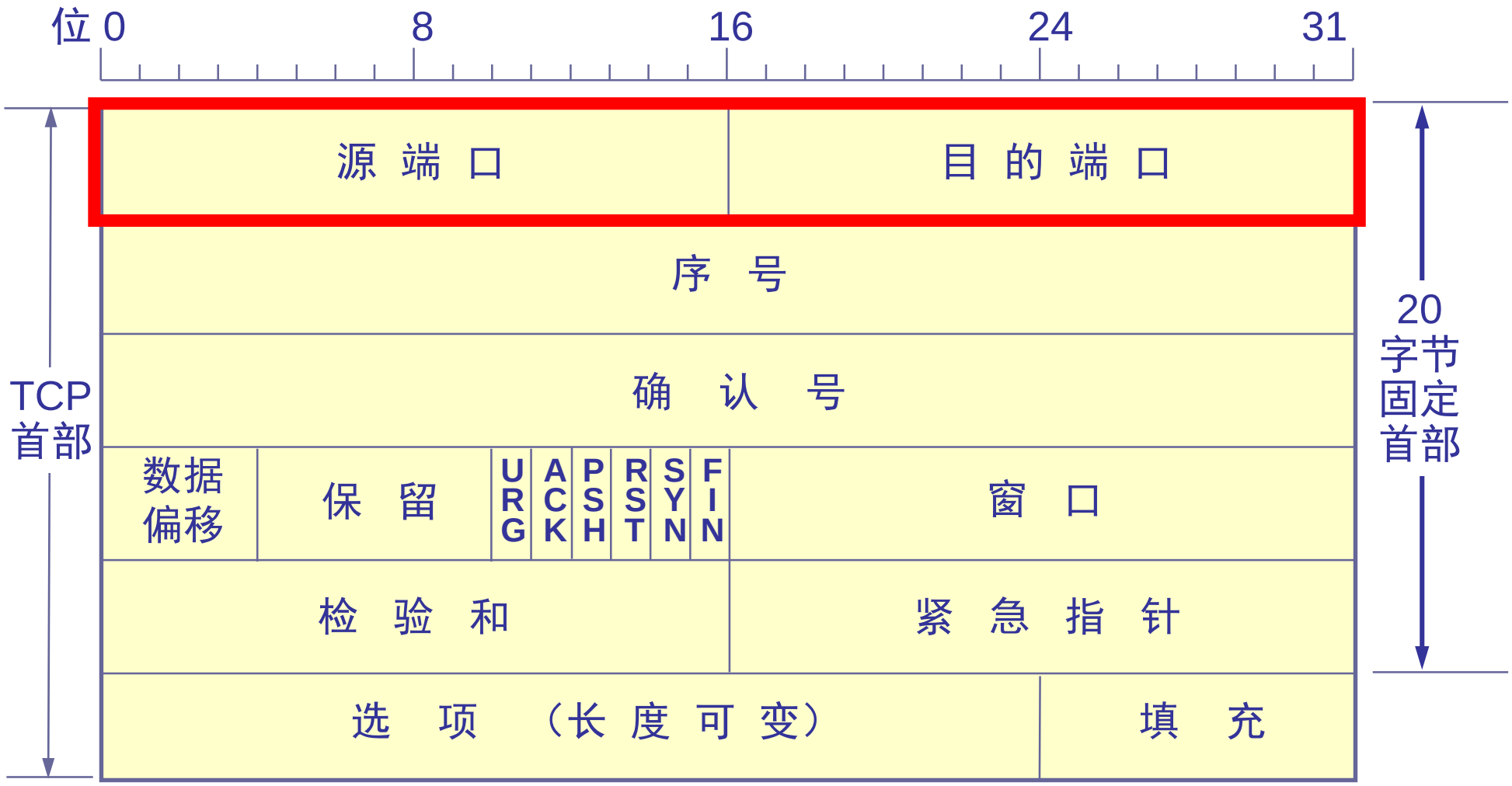
- 接收方一般采用累积确认的方式。
- 优点是：容易实现，即使确认丢失也不必重传。
- 缺点是：不能向发送方反映出接收方已经正确收到的所有分组的信息。

TCP 可靠通信的具体实现

- TCP 连接的每一端都必须设有两个窗口——一个发送窗口和一个接收窗口。
- TCP 的可靠传输机制用字节的序号进行控制。
- TCP 两端的四个窗口经常处于动态变化之中。
- TCP连接的往返时间 RTT 也不是固定不变的。需要使用特定的算法估算较为合理的重传时间。

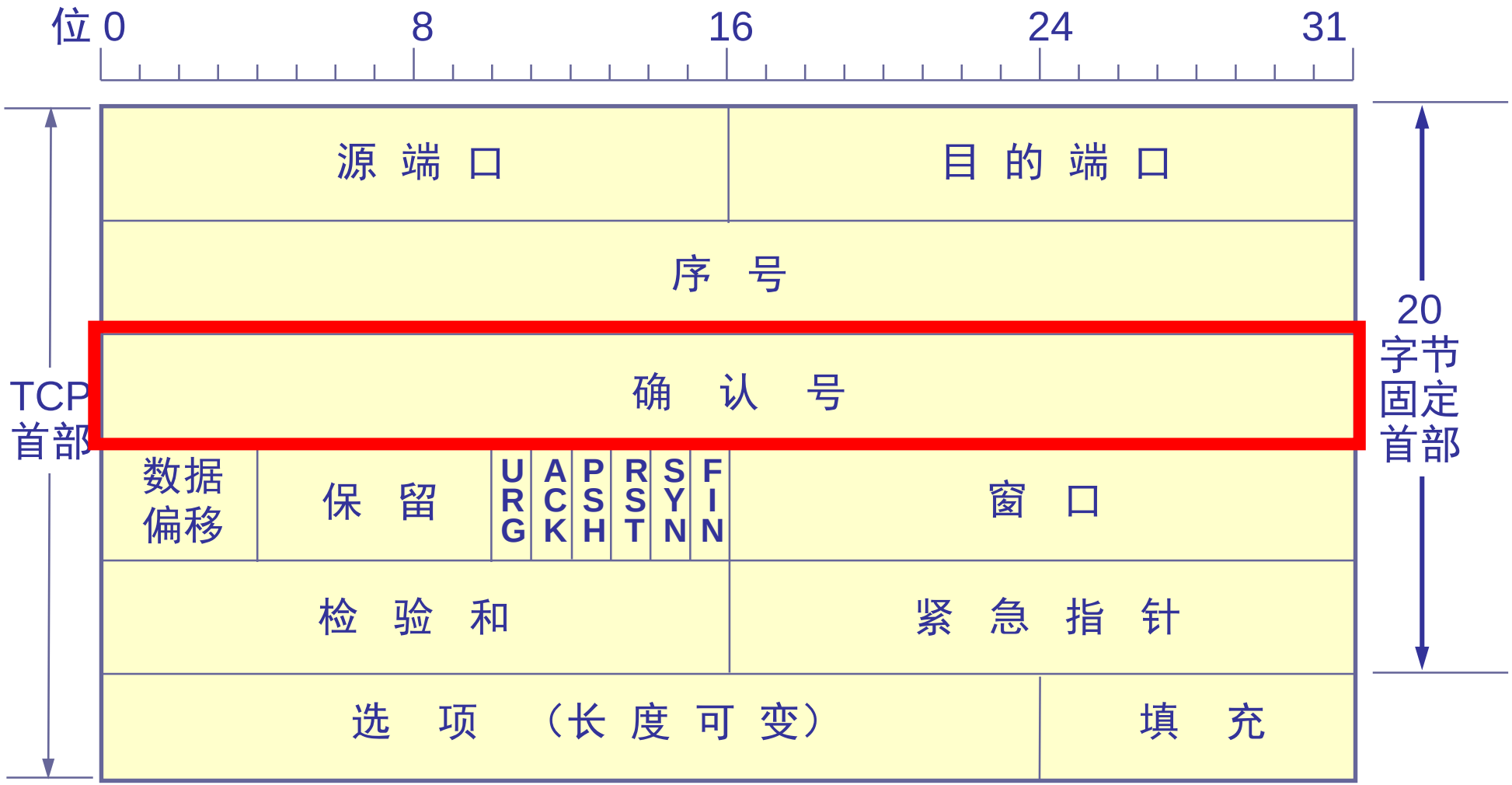
TCP 报文段的首部格式





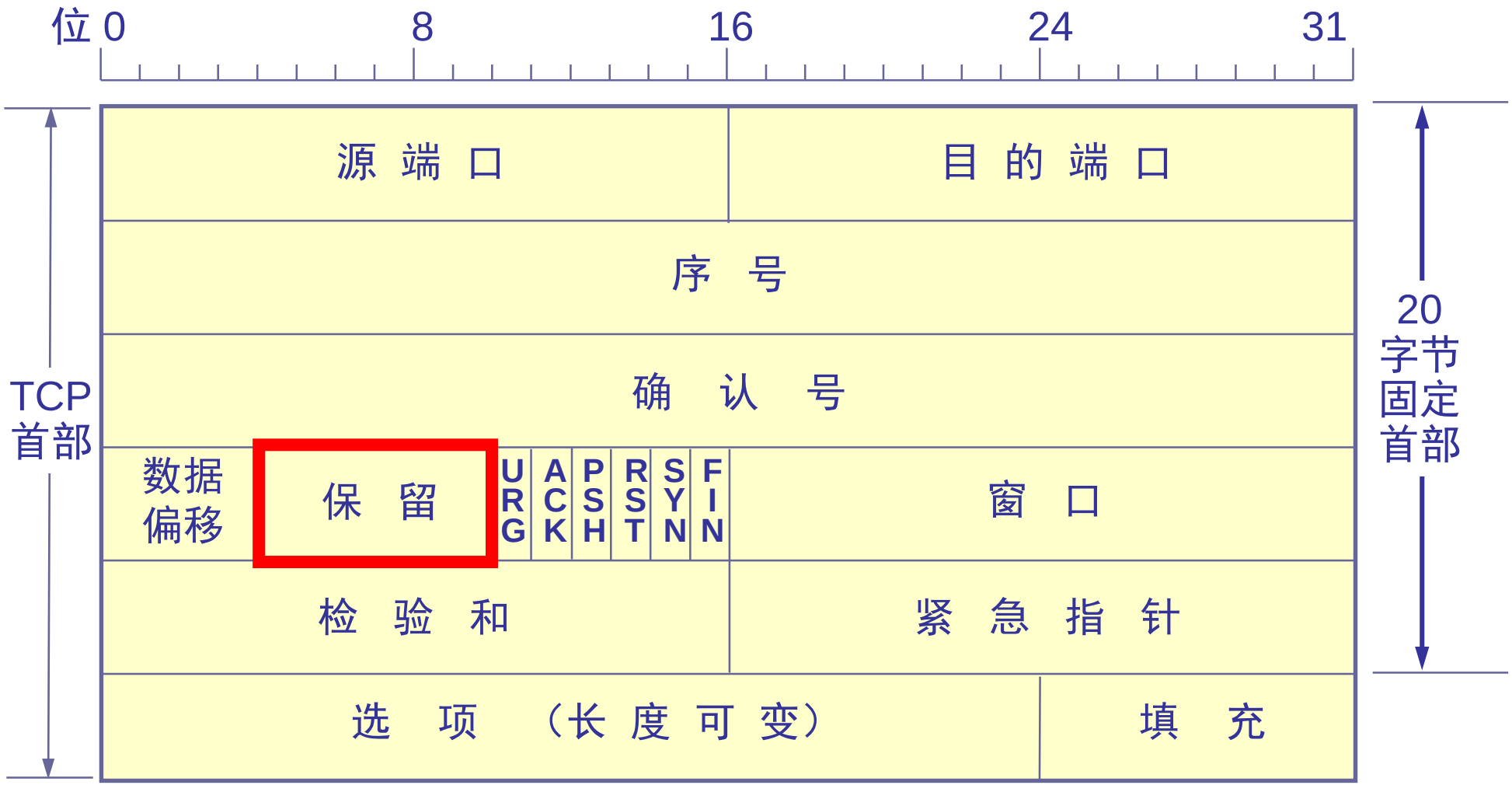
源端口和目的端口字段——各占 2 字节。端口是运输层与应用层的服务接口。运输层的复用和分用功能都要通过端口才能实现。

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)。

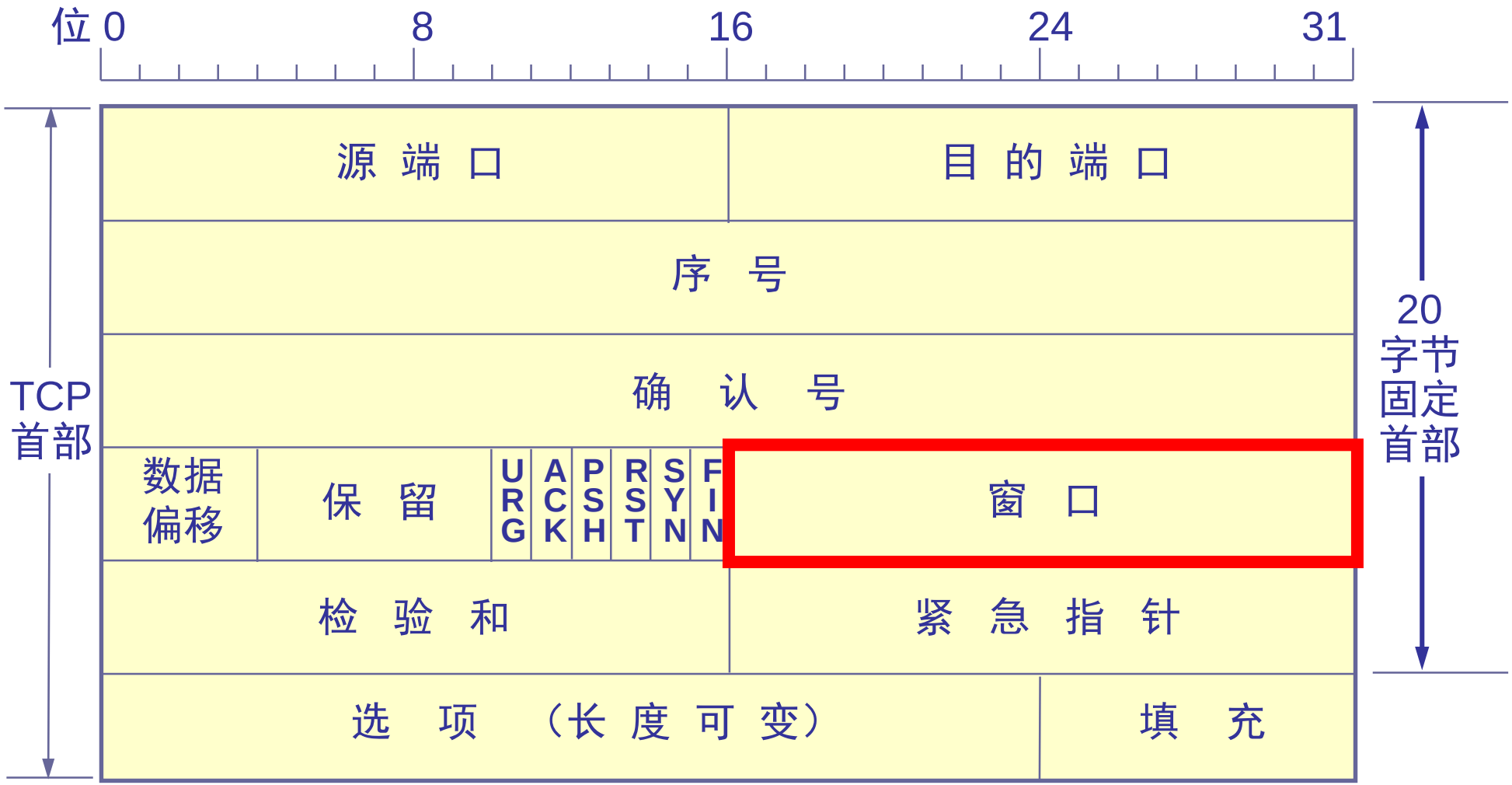
确认 ACK —— 只有当 ACK = 1 时确认号字段才有效。当 ACK = 0 时，确认号无效。

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。

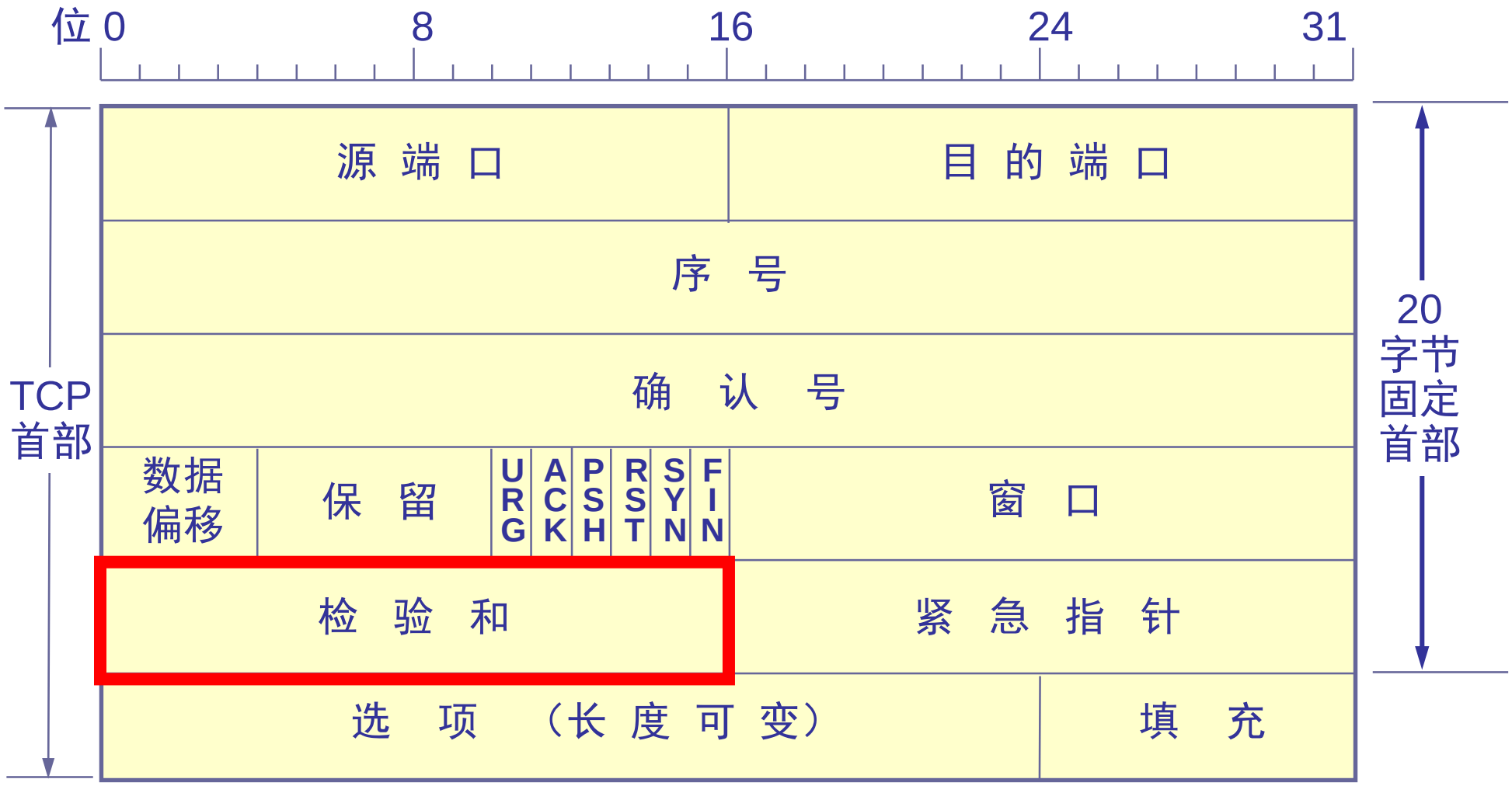
复位 RST (ReSeT) —— 当 RST = 1 时，表明 TCP 连接中出现严重差错（如由于主机崩溃或其他原因），必须释放连接，然后再重新建立运输连接。

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

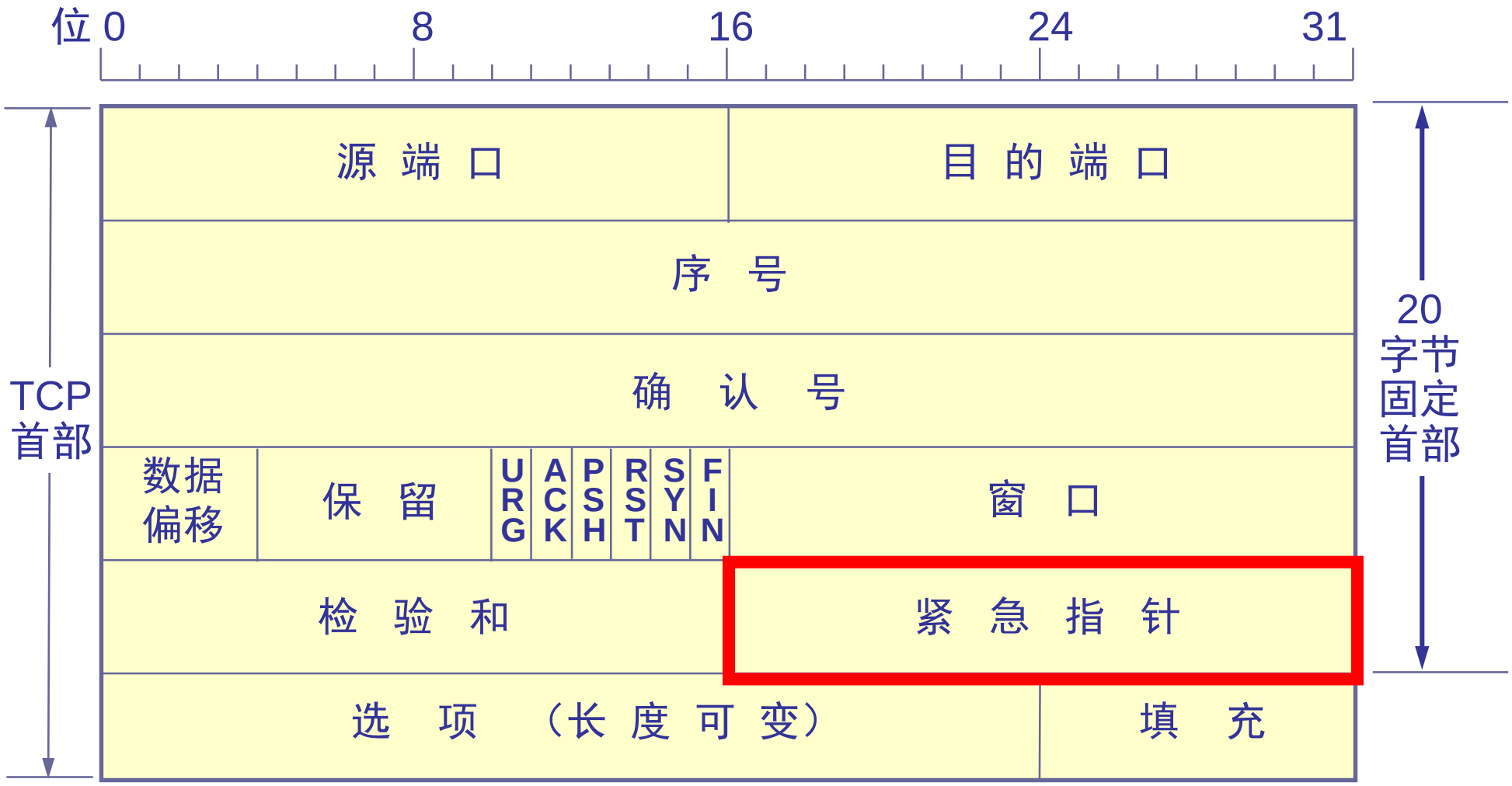
终止 FIN (FINish) —— 用来释放一个连接。FIN = 1 表明此报文段的发送端的数据已发送完毕，并要求释放运输连接。



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。

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 告诉对方 TCP：“我的缓存所能接收的报文段的数据字段的最大长度是 MSS 个字节。”

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。

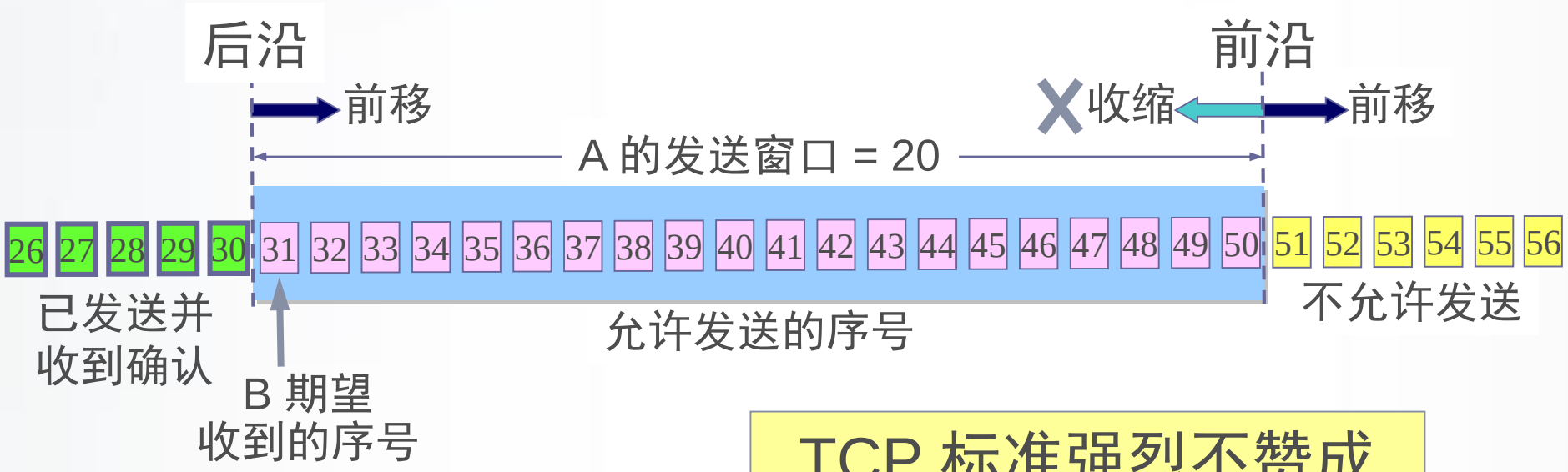
填充字段 —— 这是为了使整个首部长度是 4 字节的整数倍。

指引

- 运输层的功能
- 运输层协议UDP和TCP
- TCP可靠传输的实现
- TCP的流量控制
- TCP的拥塞控制
- TCP的运输连接管理

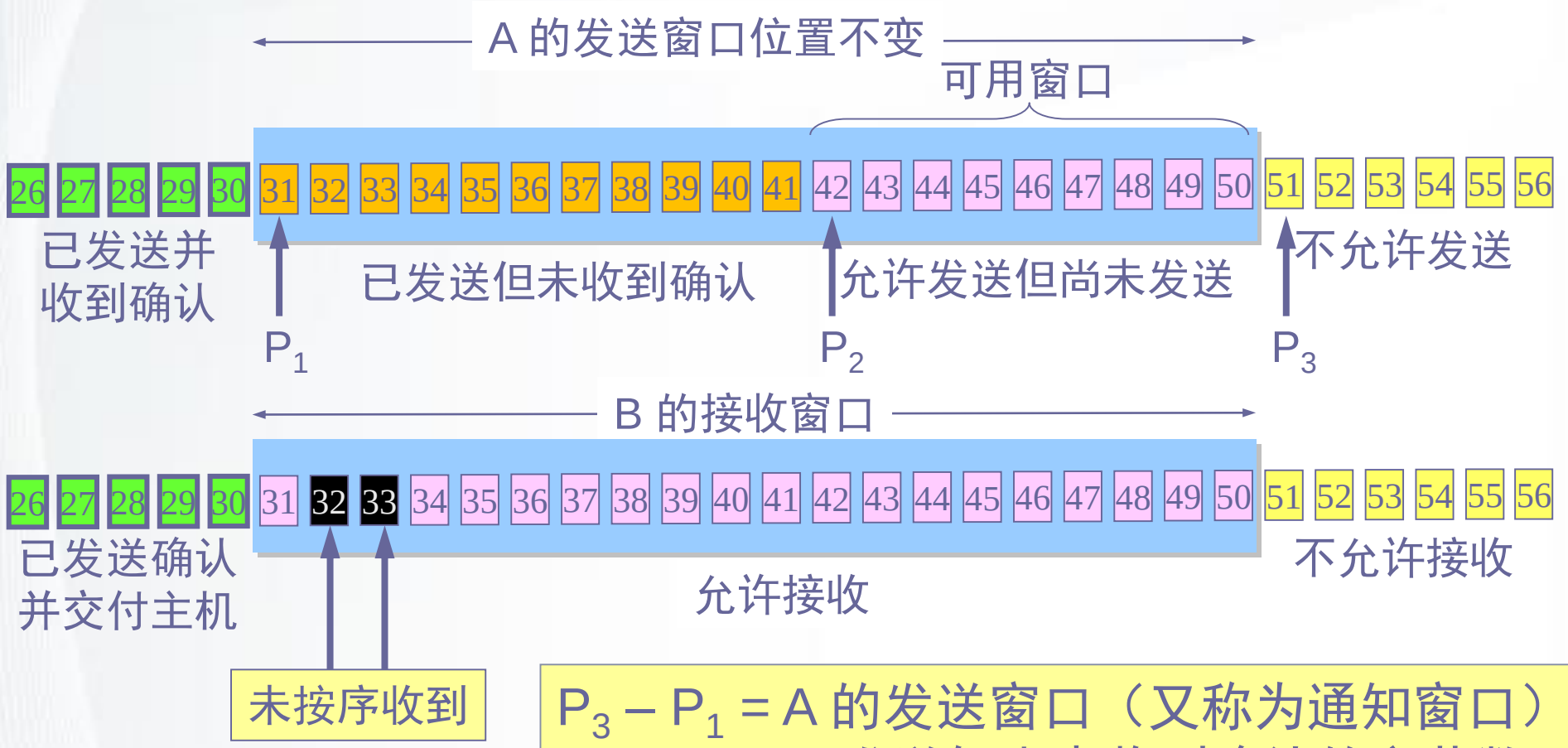
以字节为单位的滑动窗口

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

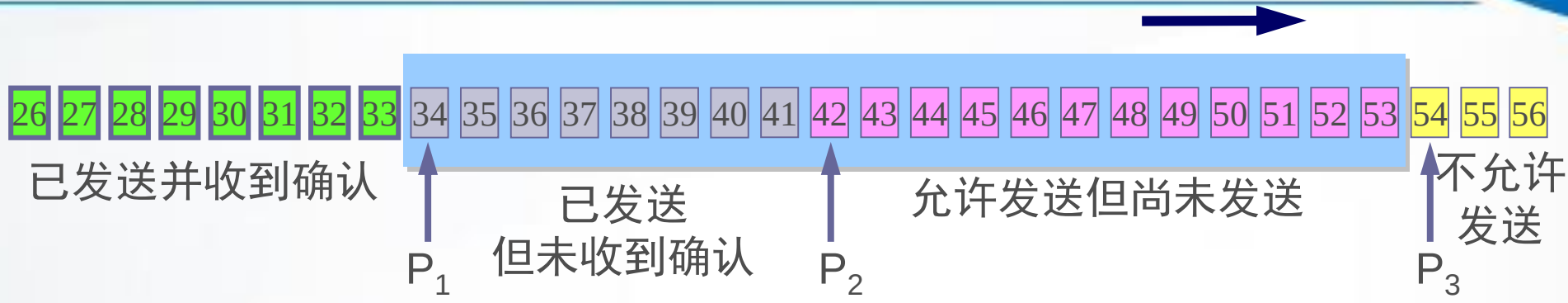


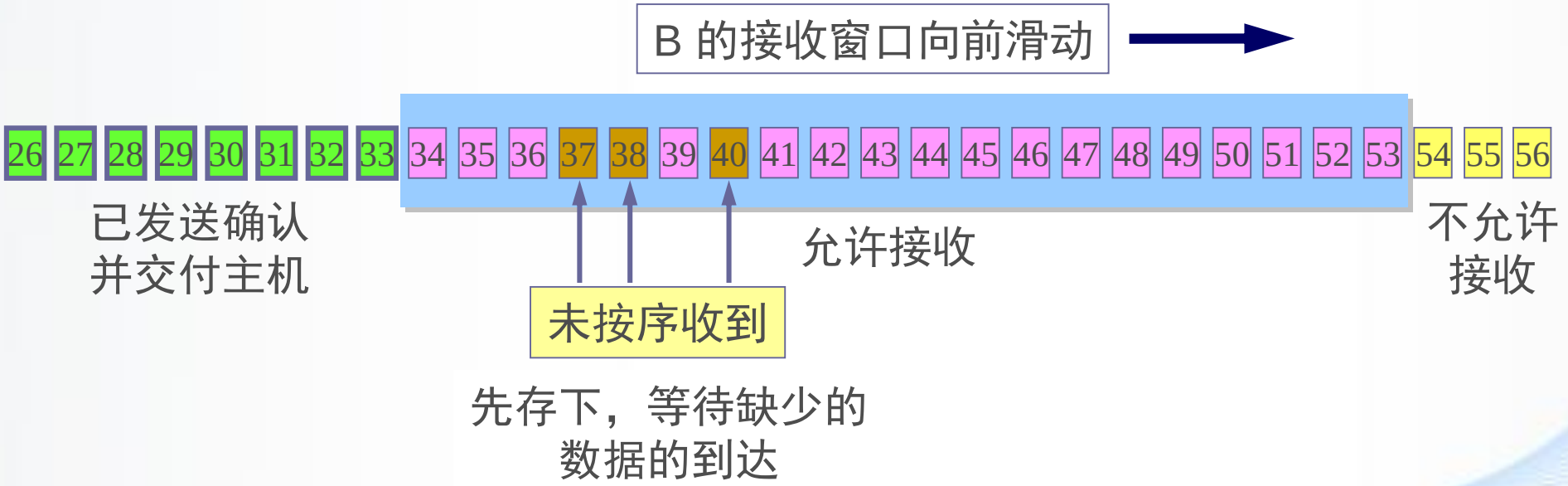
$P_3 - P_1$ = A 的发送窗口（又称为通知窗口）

$P_2 - P_1$ = 已发送但尚未收到确认的字节数

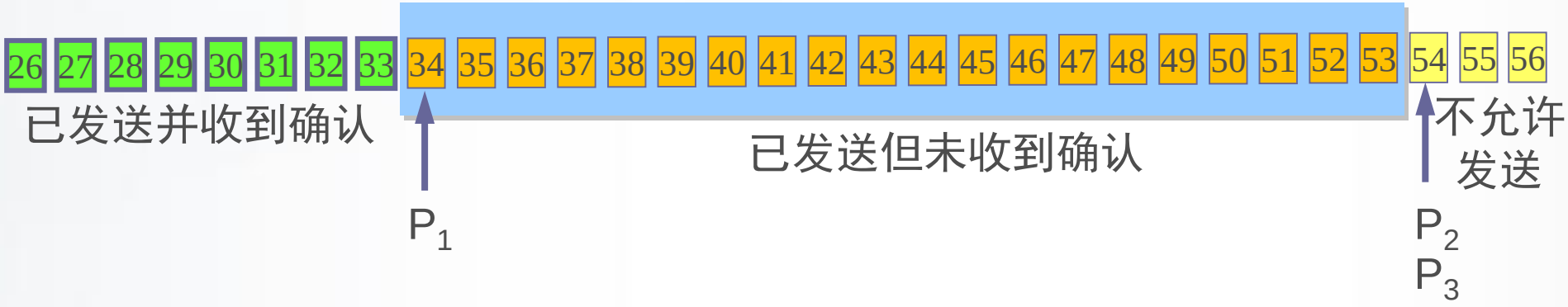
$P_3 - P_2$ = 允许发送但尚未发送的字节数（又称为可用窗口）

A收到新的确认号，发送窗口向前滑动



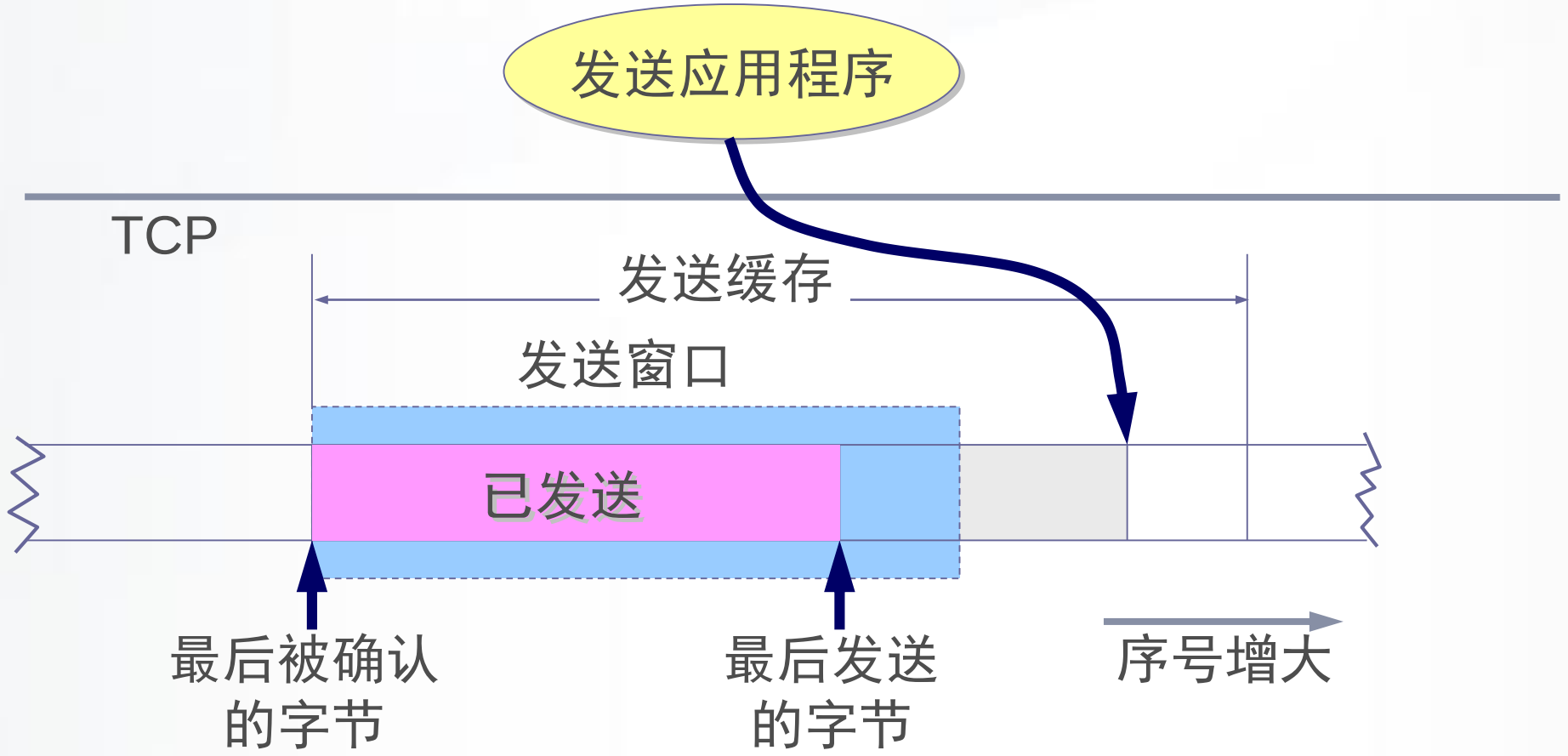


A 的发送窗口已满，有效窗口为零

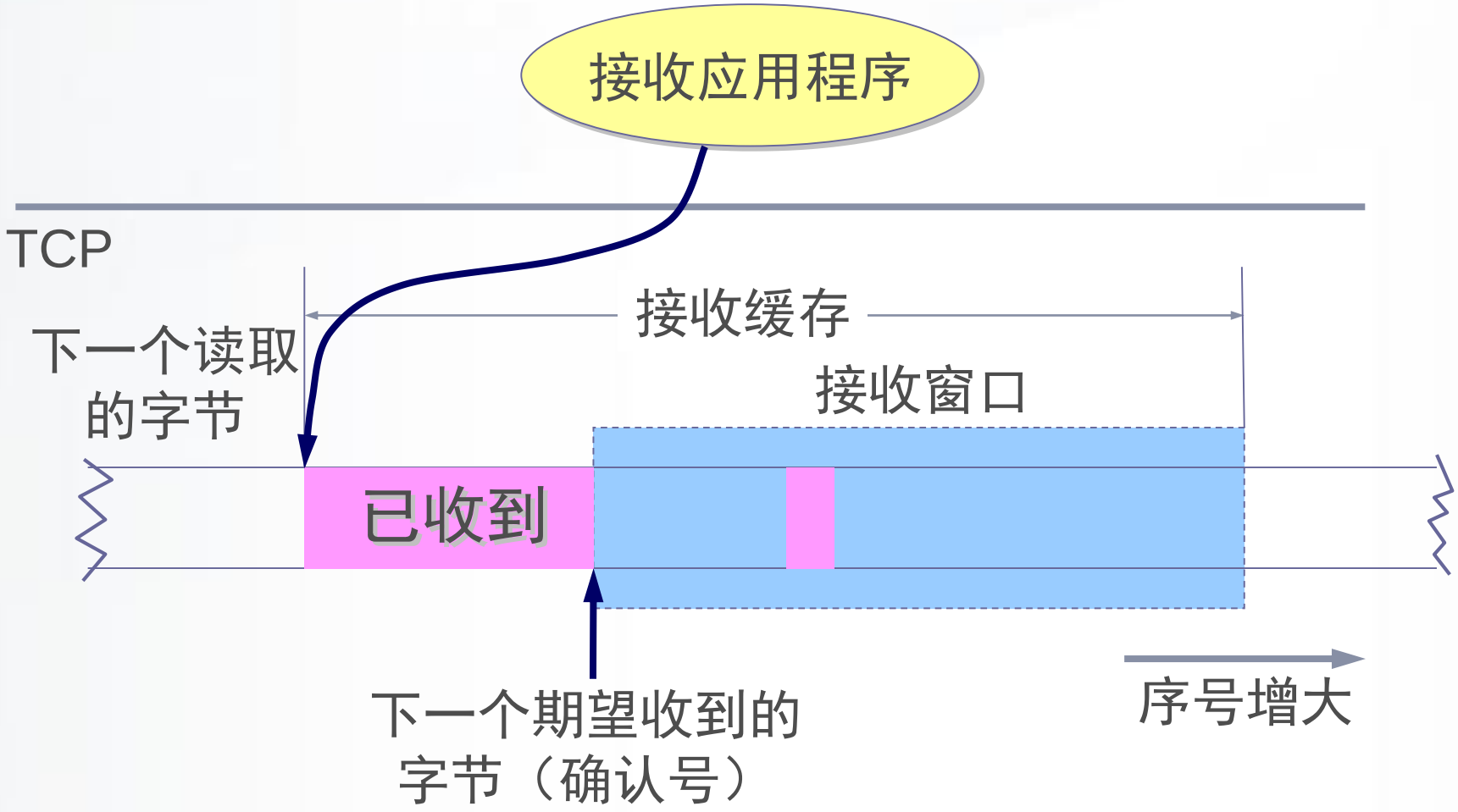


A 的发送窗口内的序号都已用完，
但还没有再收到确认，必须停止发送。

发送缓存



接收缓存



超时重传时间的选择

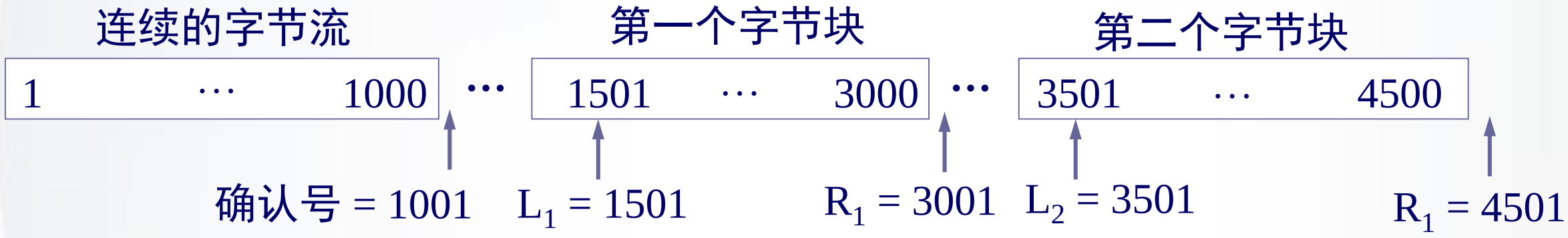
- TCP 每发送一个报文段，就对这个报文段设置一次计时器。只要计时器设置的重传时间到但还没有收到确认，就要重传这一报文段。

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

超时重传时间应略大于上面得出的**加权平均往返时间** RTT_S

选择确认 SACK

- 接收到的字节流序号不连续



- 和前后字节不连续的每一个字节块都有两个边界：左边界和右边界。图中用四个指针标记这些边界。
- 第一个字节块的左边界 L1 = 1501，但右边界 R1 = 3001。
- 左边界指出字节块的第一个字节的序号，但右边界减 1 才是字节块中的最后一个序号。
- 第二个字节块的左边界 L2 = 3501，而右边界 R2 = 4501。

指引

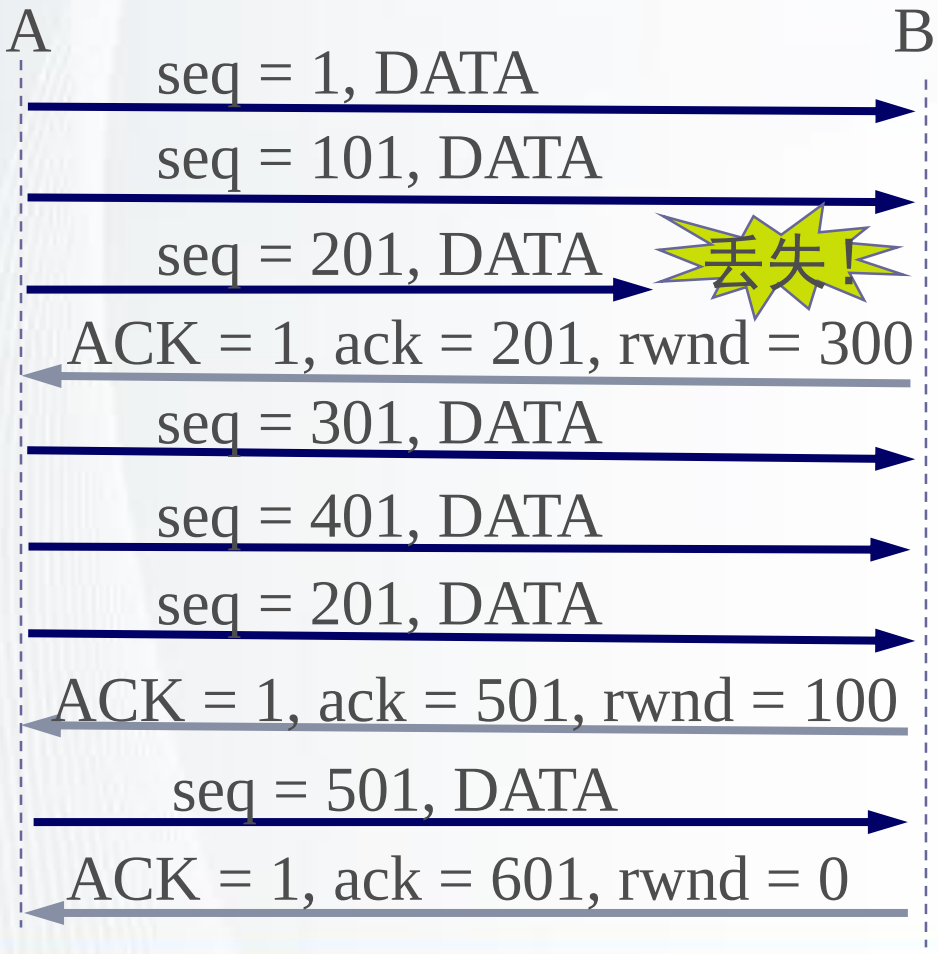
- 运输层的功能
- 运输层协议UDP和TCP
- TCP可靠传输的实现
- TCP的流量控制
- TCP的拥塞控制
- TCP的运输连接管理

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(persistence timer)

- 发送方和接收方死锁局面的产生
- TCP 为每一个连接设有一个持续计时器
- 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
- 若窗口仍然是零，则收到这个报文段的一方就重新设置持续计时器。
- 若窗口不是零，则死锁的僵局就可以打破了。

指引

- 运输层的功能
- 运输层协议UDP和TCP
- TCP可靠传输的实现
- TCP的流量控制
- TCP的拥塞控制
- TCP的运输连接管理

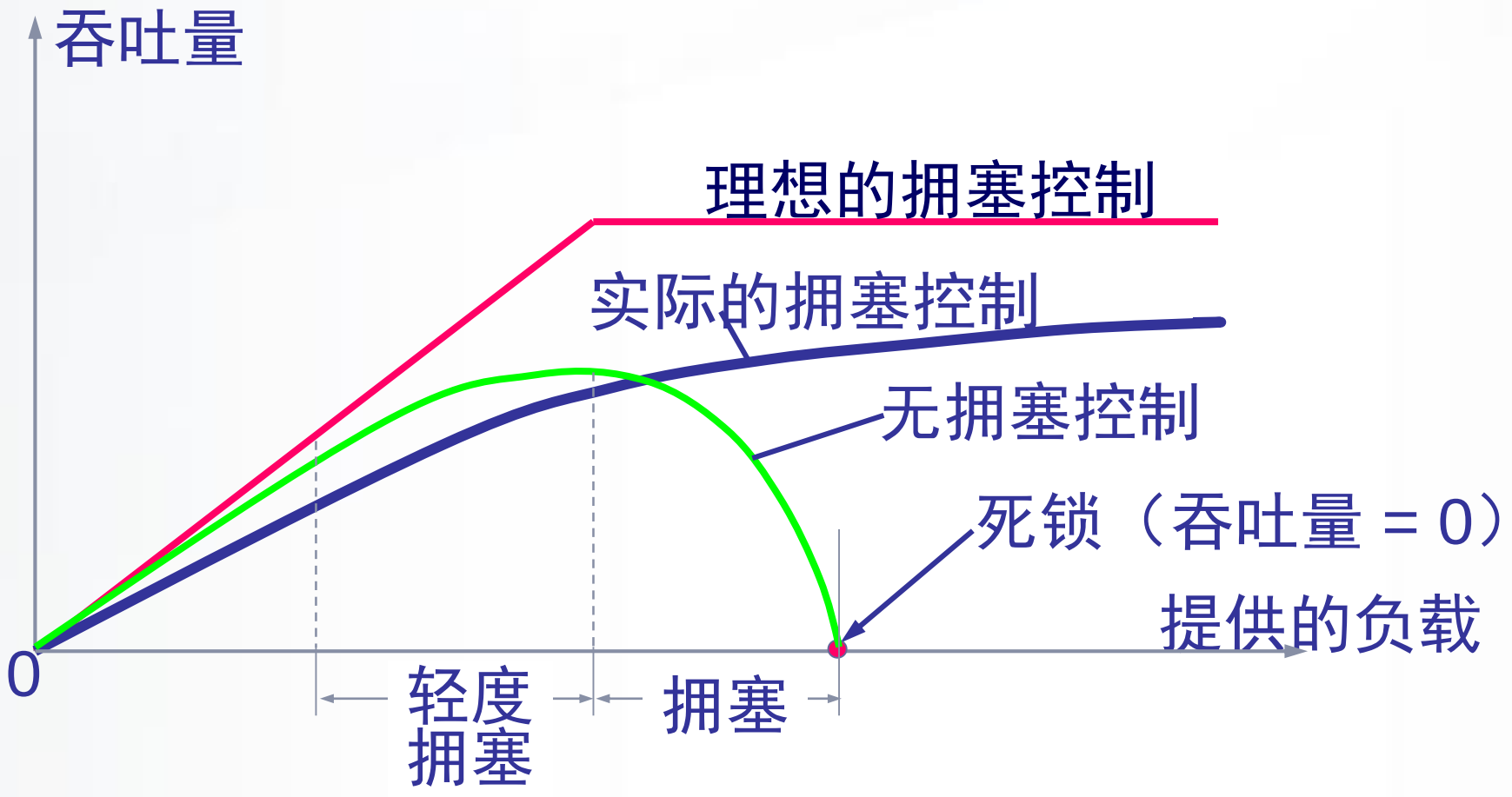
拥塞控制的一般原理

- 出现资源拥塞的条件：

  对资源需求的总和 > 可用资源

- **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，它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。

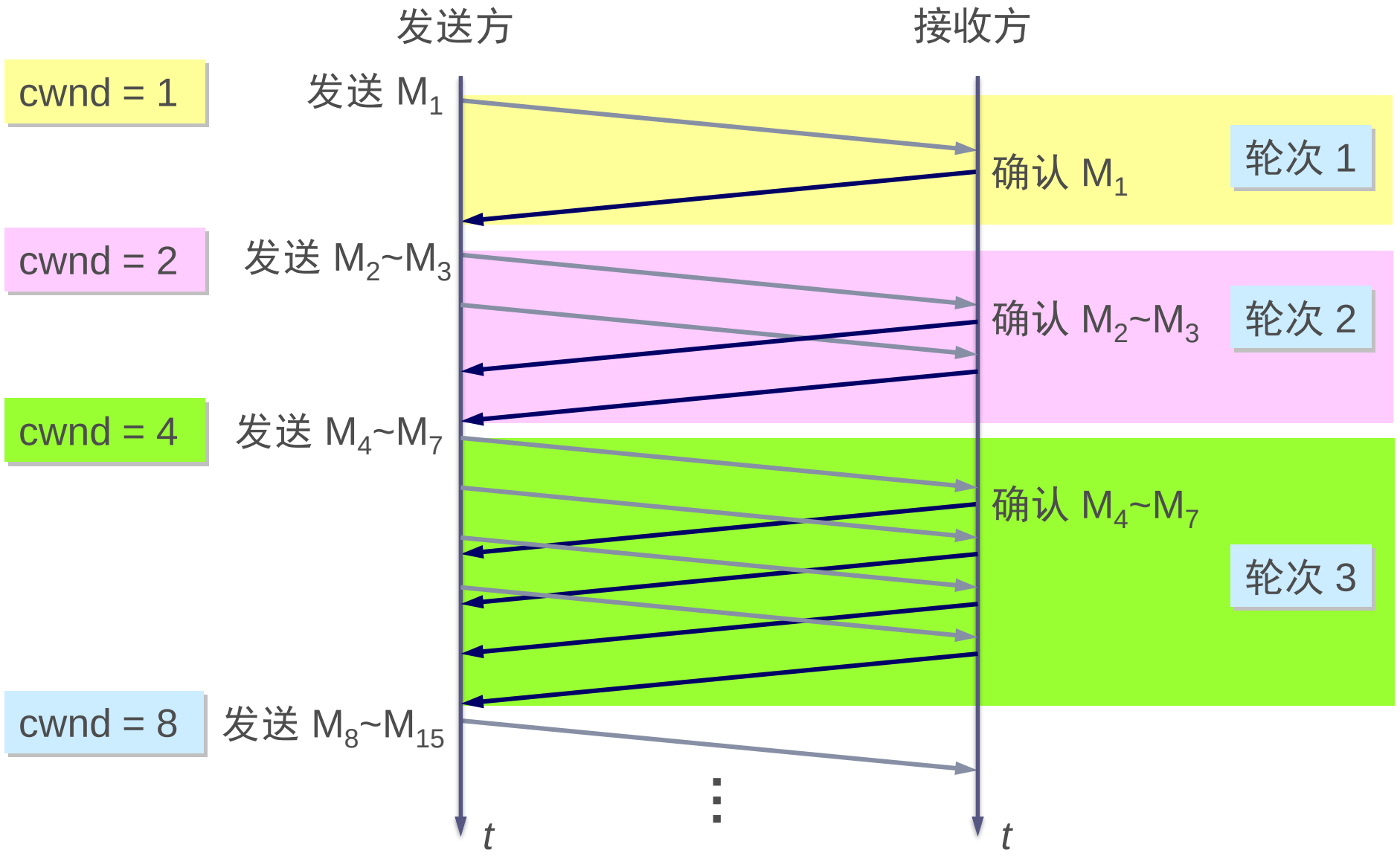
拥塞控制所起的作用



慢开始和拥塞避免

- 发送方维持拥塞窗口 cwnd (congestion window)
- 发送方控制拥塞窗口的原则是：
  - 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。
  - 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


发送方
接收方
cwnd = 1
发送 M1
确认 M1
轮次 1
cwnd = 2
发送 M2~M3
确认 M2~M3
轮次 2
cwnd = 4
发送 M4~M7
确认 M4~M7
轮次 3
cwnd = 8
发送 M8~M15
t
t

设置慢开始门限状态变量ssthresh

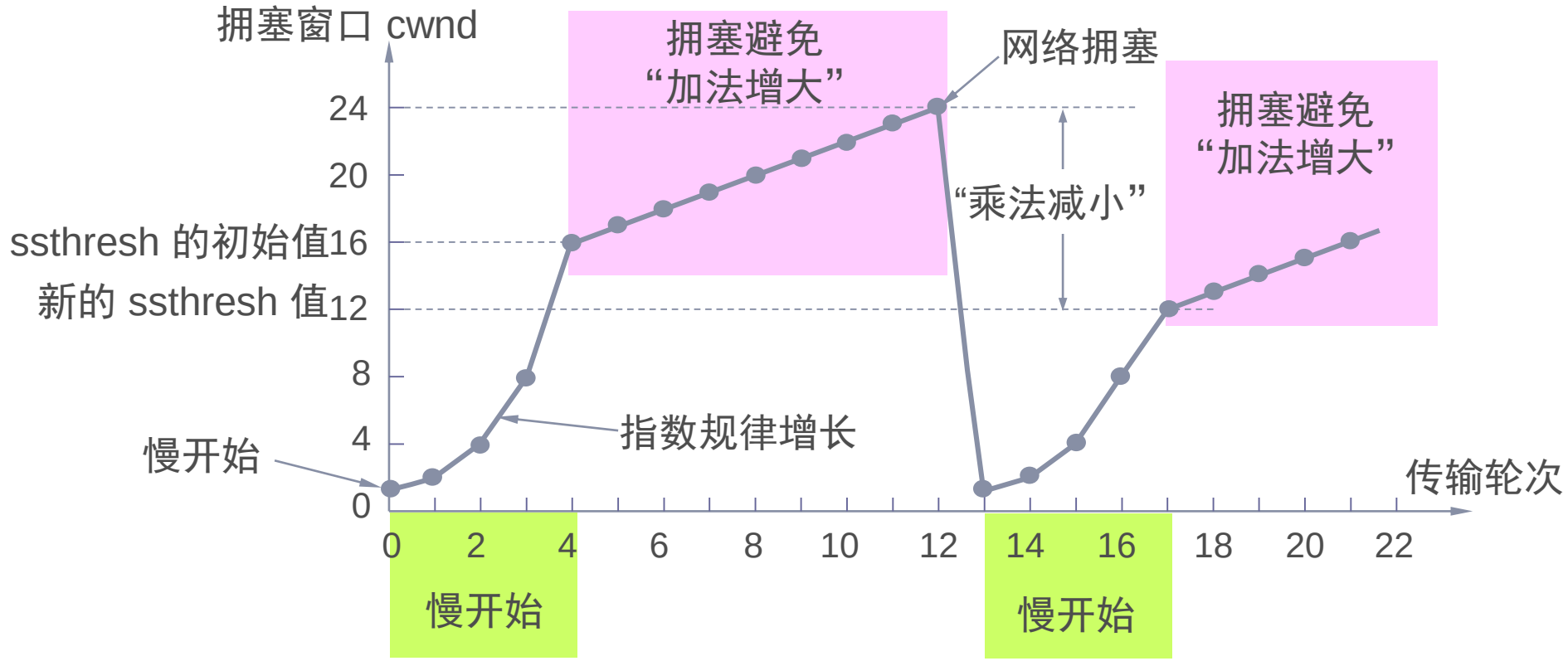
- 慢开始门限 ssthresh 的用法如下：
  - 当 cwnd < ssthresh 时，使用慢开始算法。
  - 当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。
  - 当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞避免算法。
- 拥塞避免算法的思路是让拥塞窗口 cwnd 缓慢地增大，即每经过一个往返时间 RTT 就把发送方的拥塞窗口 cwnd 加 1，而不是加倍，使拥塞窗口 cwnd 按线性规律缓慢增长。

当网络出现拥塞时

- 无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（没有按时收到确认），就要把慢开始门限 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。

这样做的目的就是要迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。

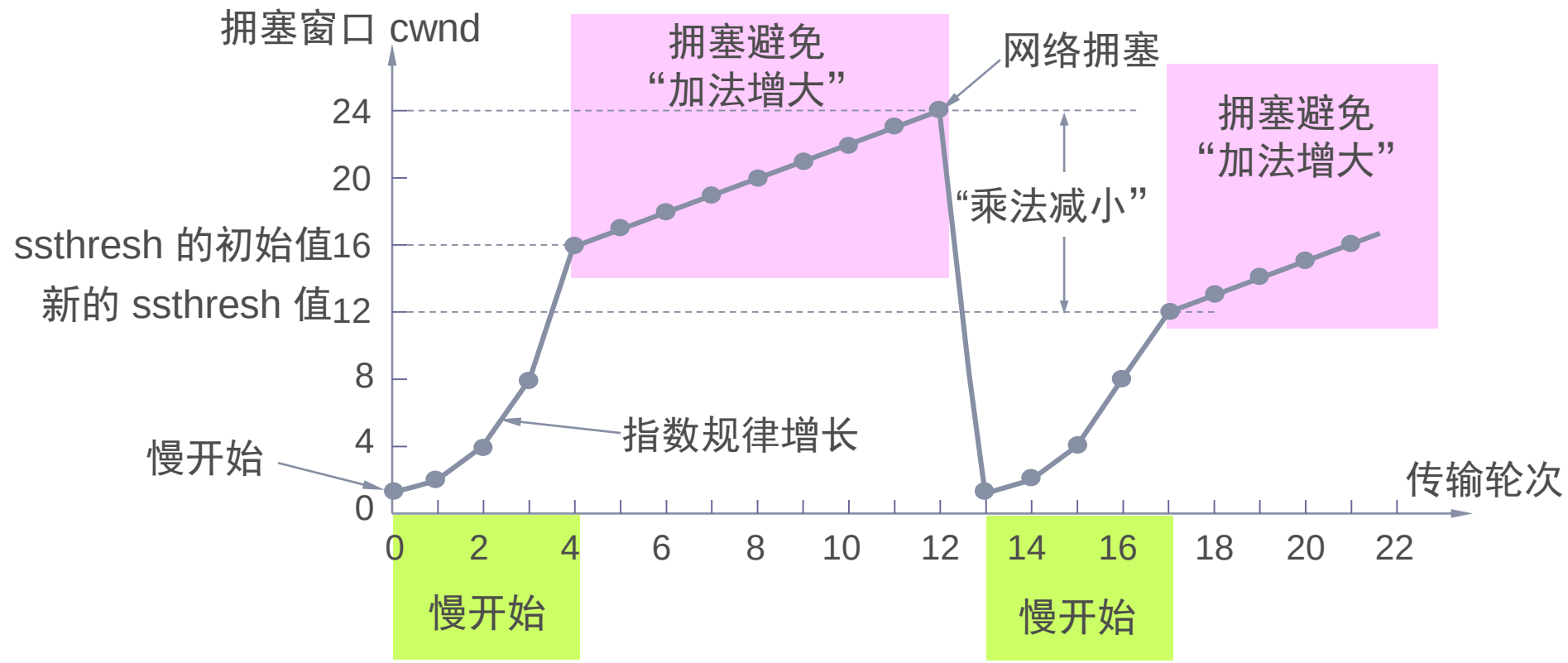
慢开始和拥塞避免



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。

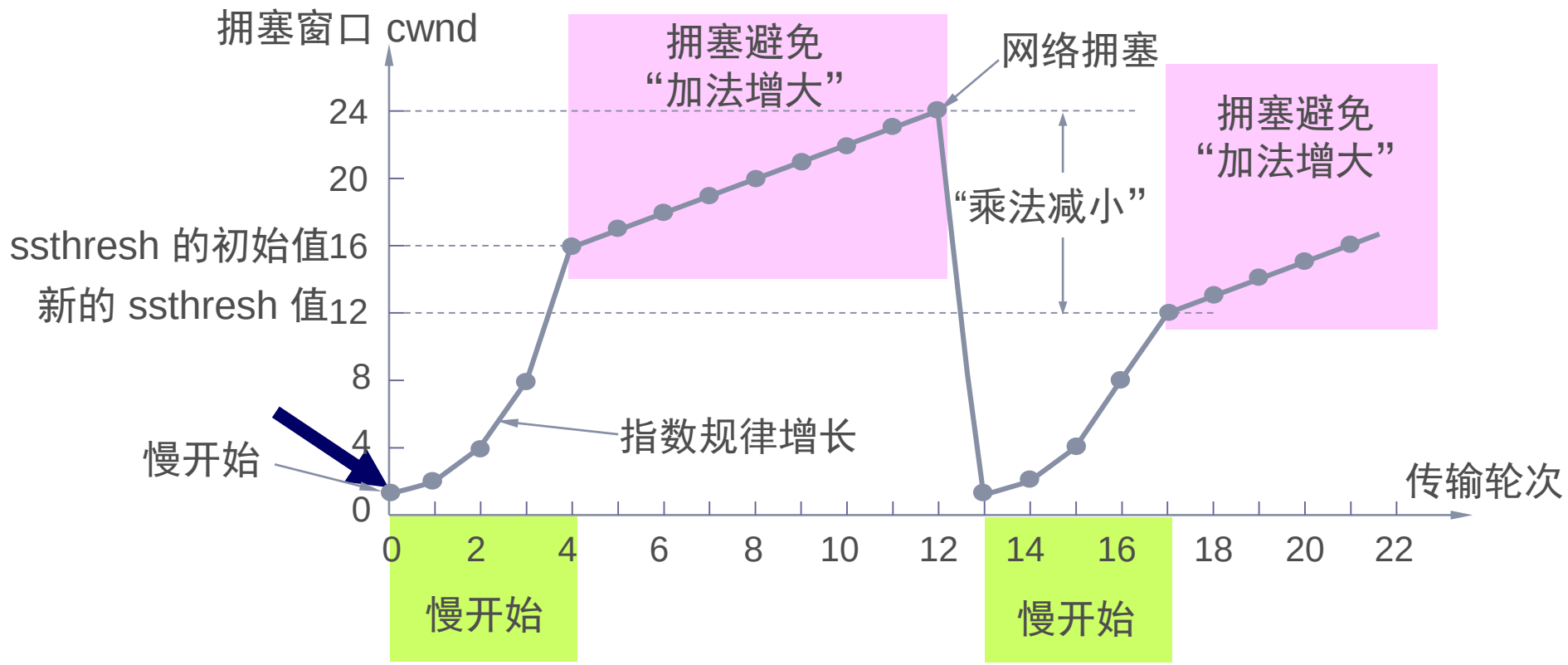
慢开始门限的初始值设置为 16 个报文段，即 ssthresh = 16。

慢开始和拥塞避免算法的实现举例



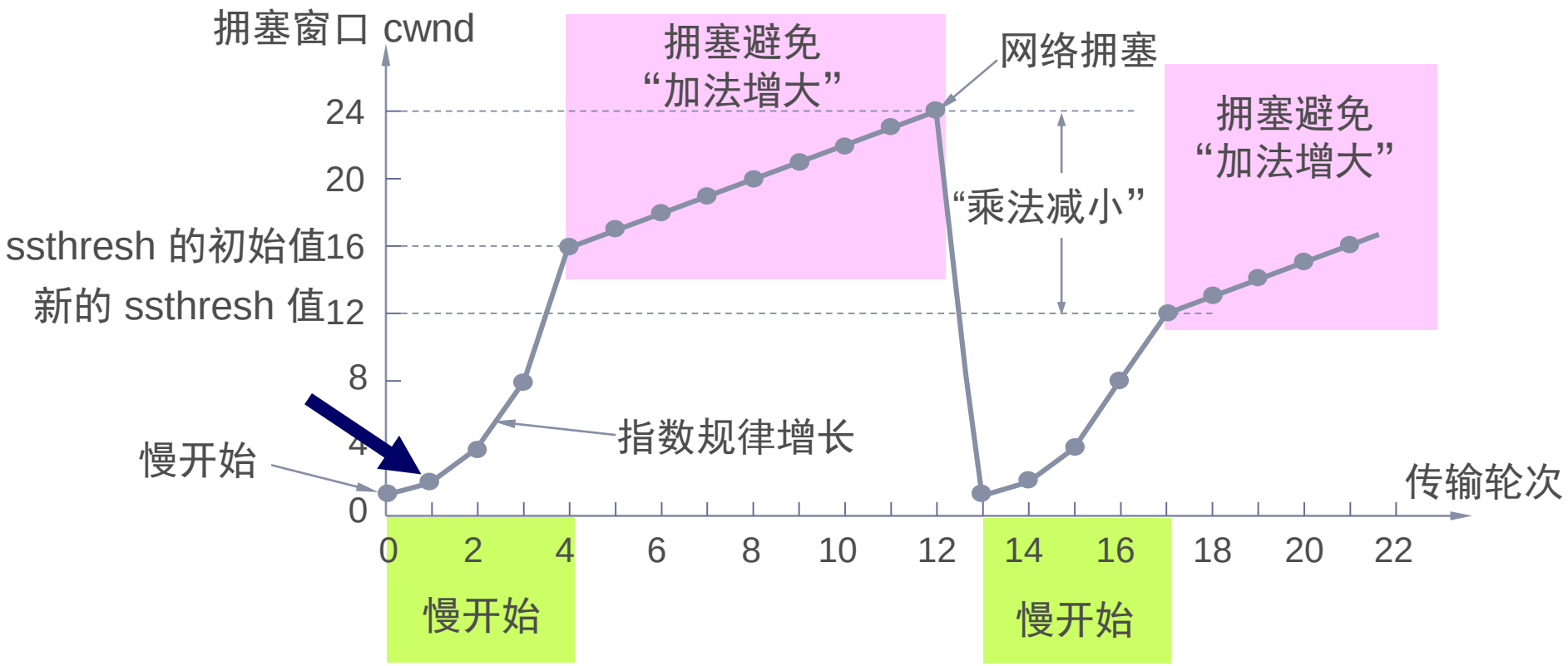
发送端的发送窗口不能超过拥塞窗口 cwnd 和接收端窗口 rwnd 中的最小值。我们假定接收端窗口足够大，因此现在发送窗口的数值等于拥塞窗口的数值。

慢开始和拥塞避免算法的实现举例



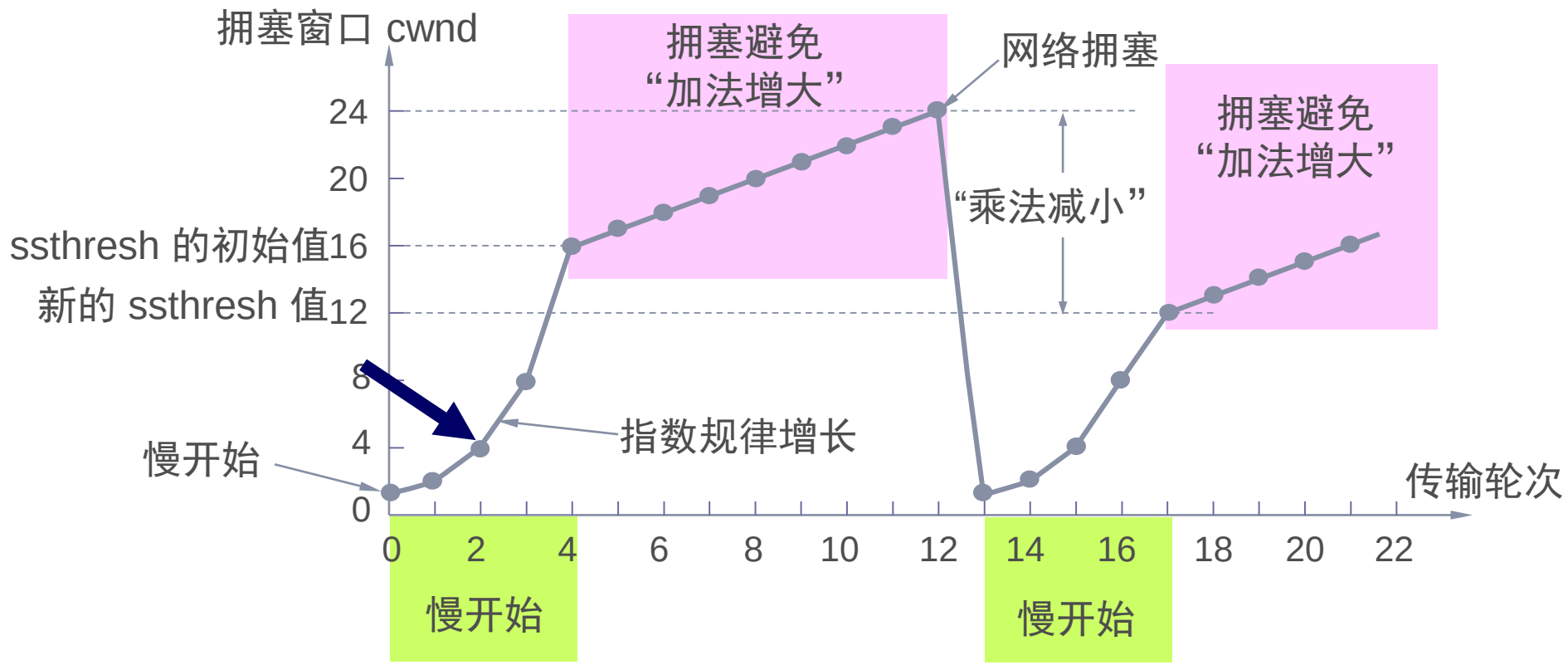
在执行慢开始算法时，拥塞窗口 cwnd 的初始值为 1，发送第一个报文段 M0。

慢开始和拥塞避免算法的实现举例



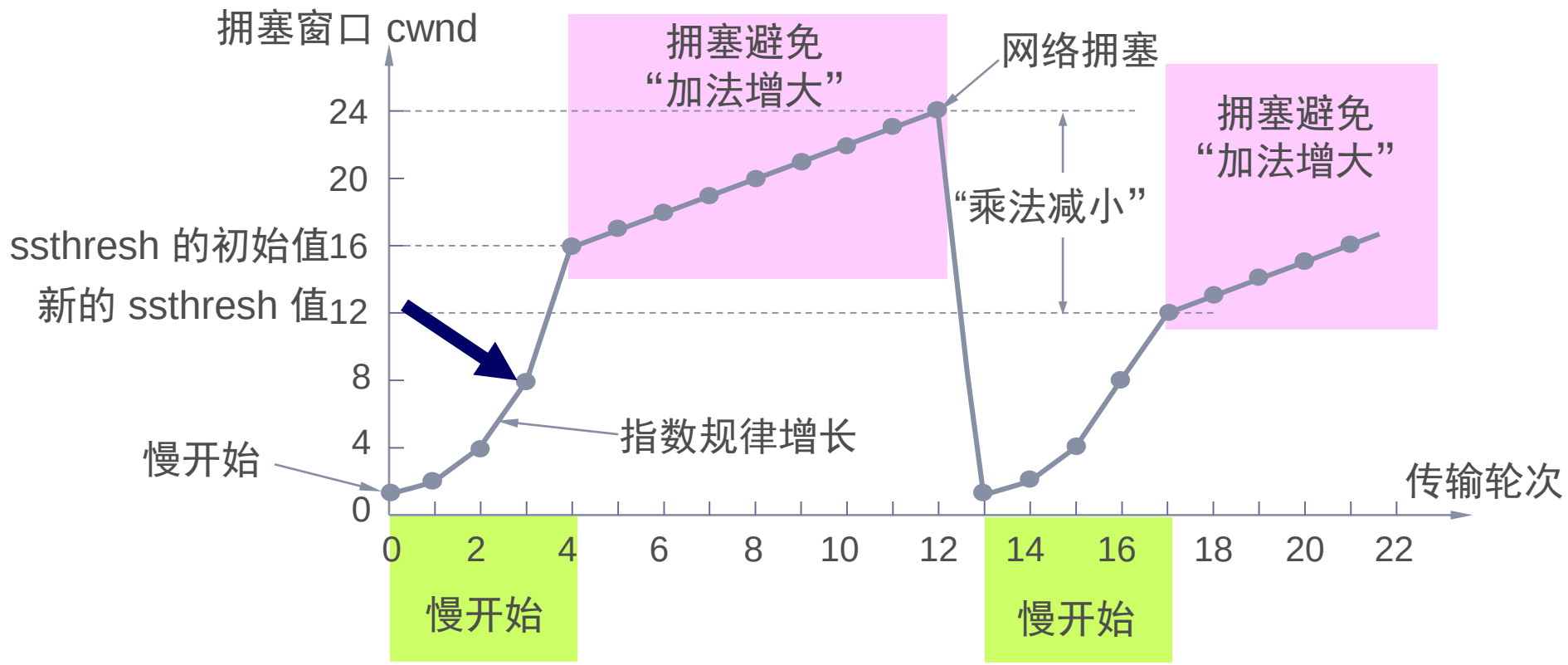
发送端每收到一个确认，就把 cwnd 加 1。于是发送端可以接着发送 M1 和 M2 两个报文段。

慢开始和拥塞避免算法的实现举例



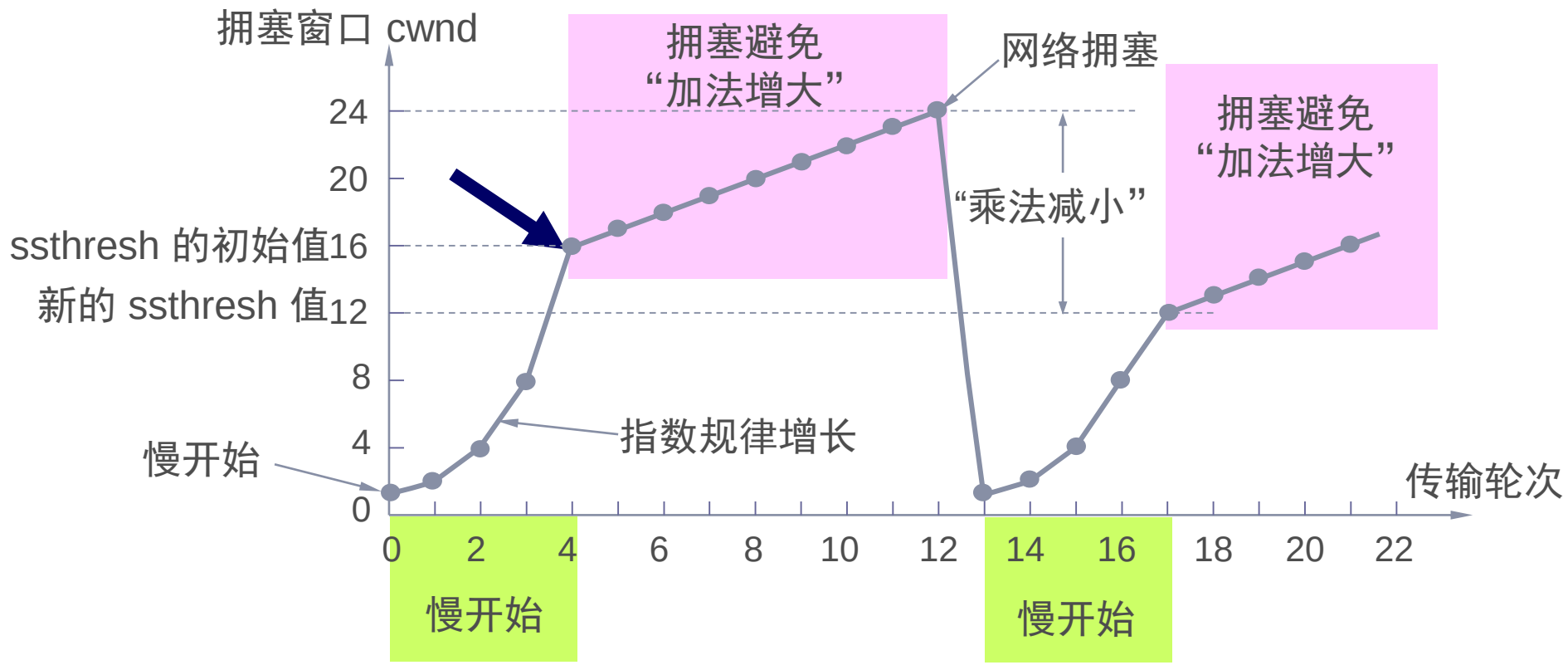
接收端共发回两个确认。发送端每收到一个对新报文段的确认，就把发送端的 cwnd 加 1。现在 cwnd 从 2 增大到 4，并可接着发送后面的 4 个报文段。

慢开始和拥塞避免算法的实现举例



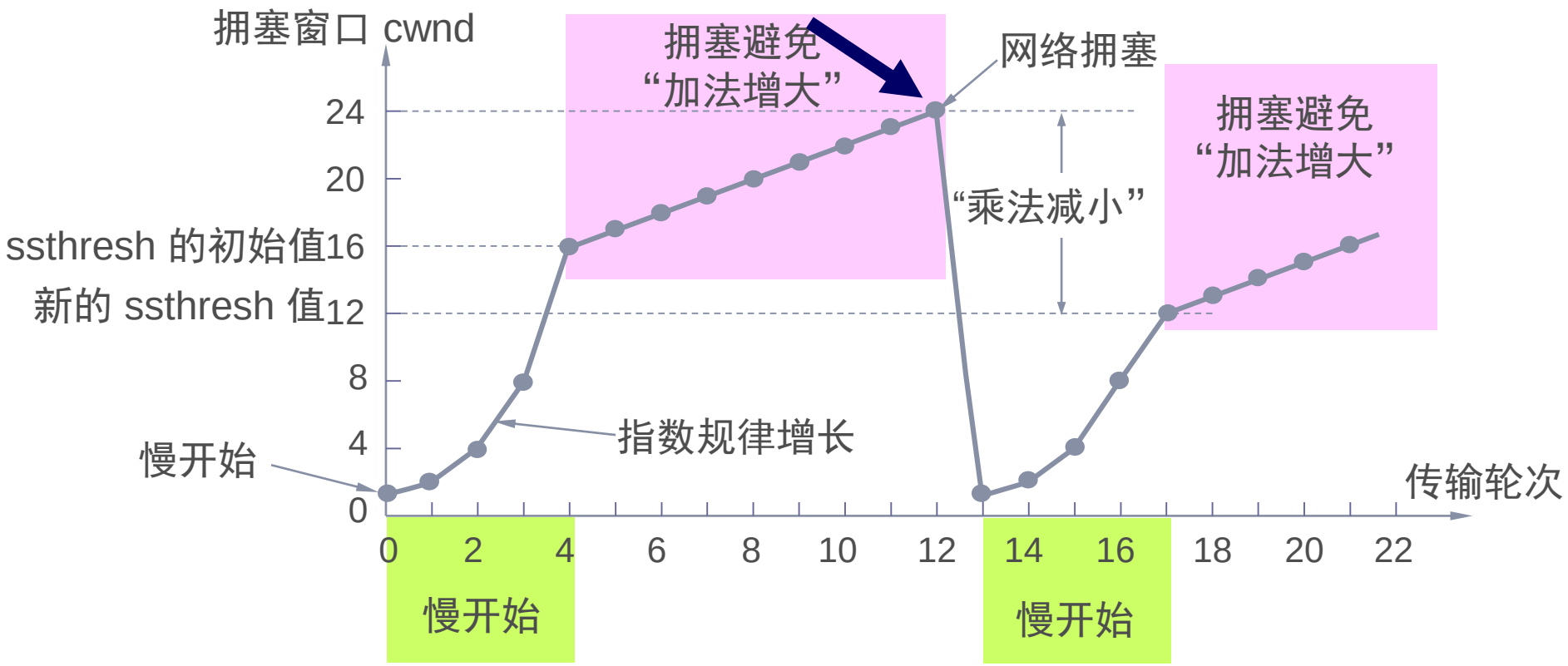
发送端每收到一个对新报文段的确认，就把发送端的拥塞窗口加 1，因此拥塞窗口 cwnd 随着传输轮次按指数规律增长。

慢开始和拥塞避免算法的实现举例



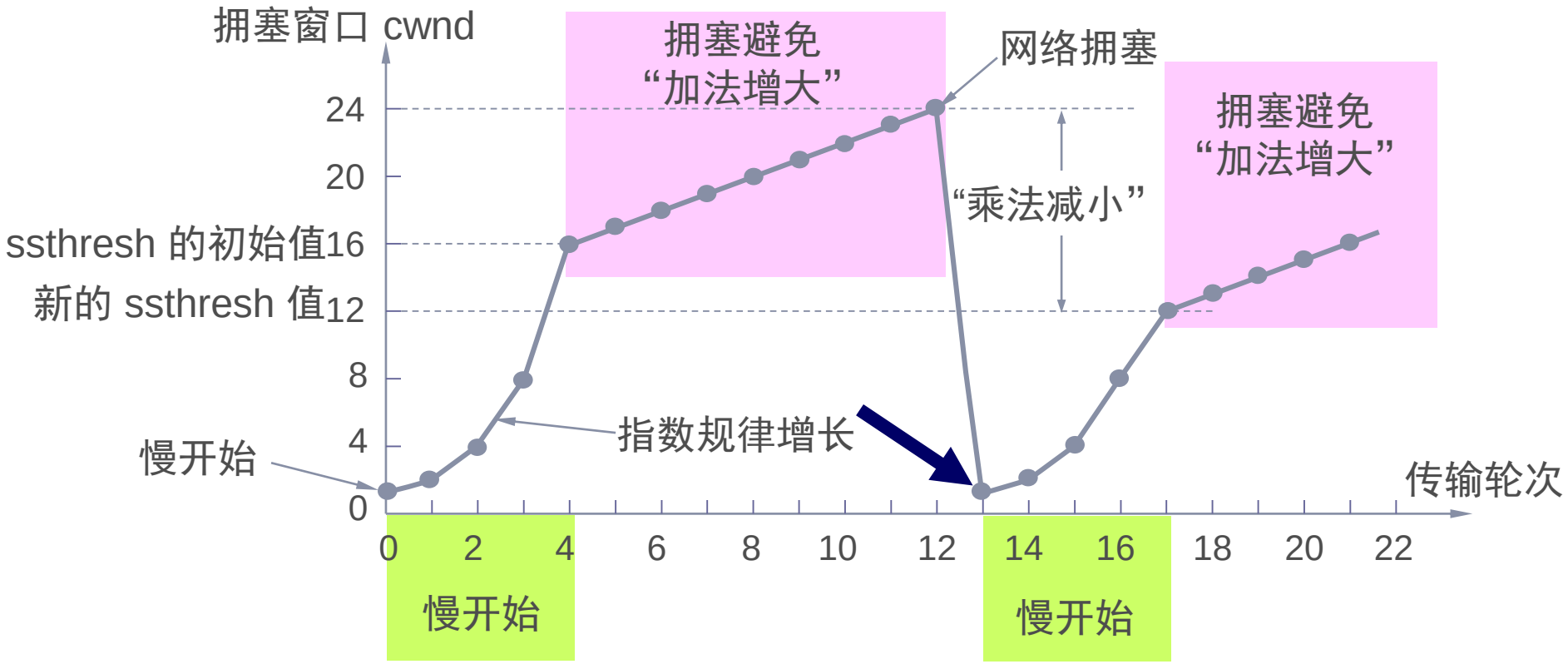
当拥塞窗口 cwnd 增长到慢开始门限值 ssthresh 时（即当 cwnd = 16 时），就改为执行拥塞避免算法，拥塞窗口按线性规律增长。

慢开始和拥塞避免算法的实现举例



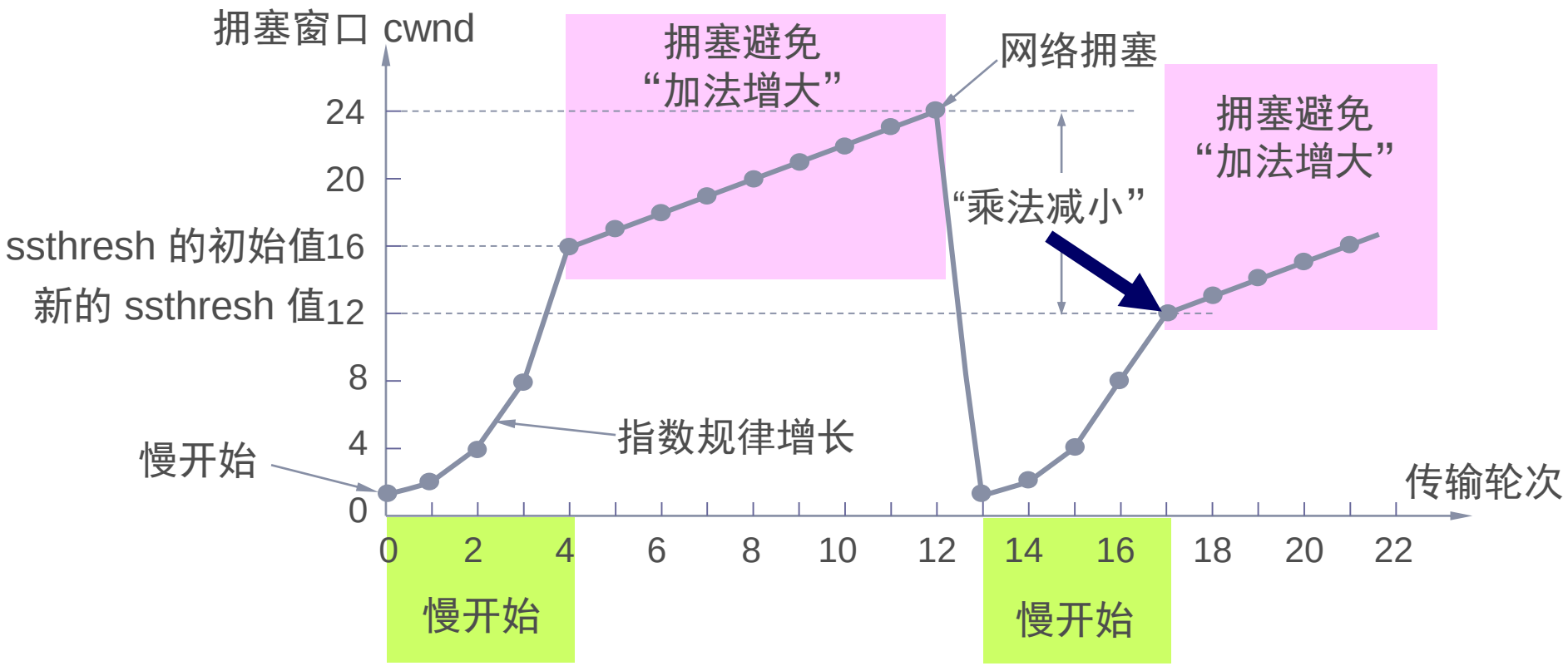
假定拥塞窗口的数值增长到 24 时，网络出现超时，表明网络拥塞了。

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 MSS 的大小。

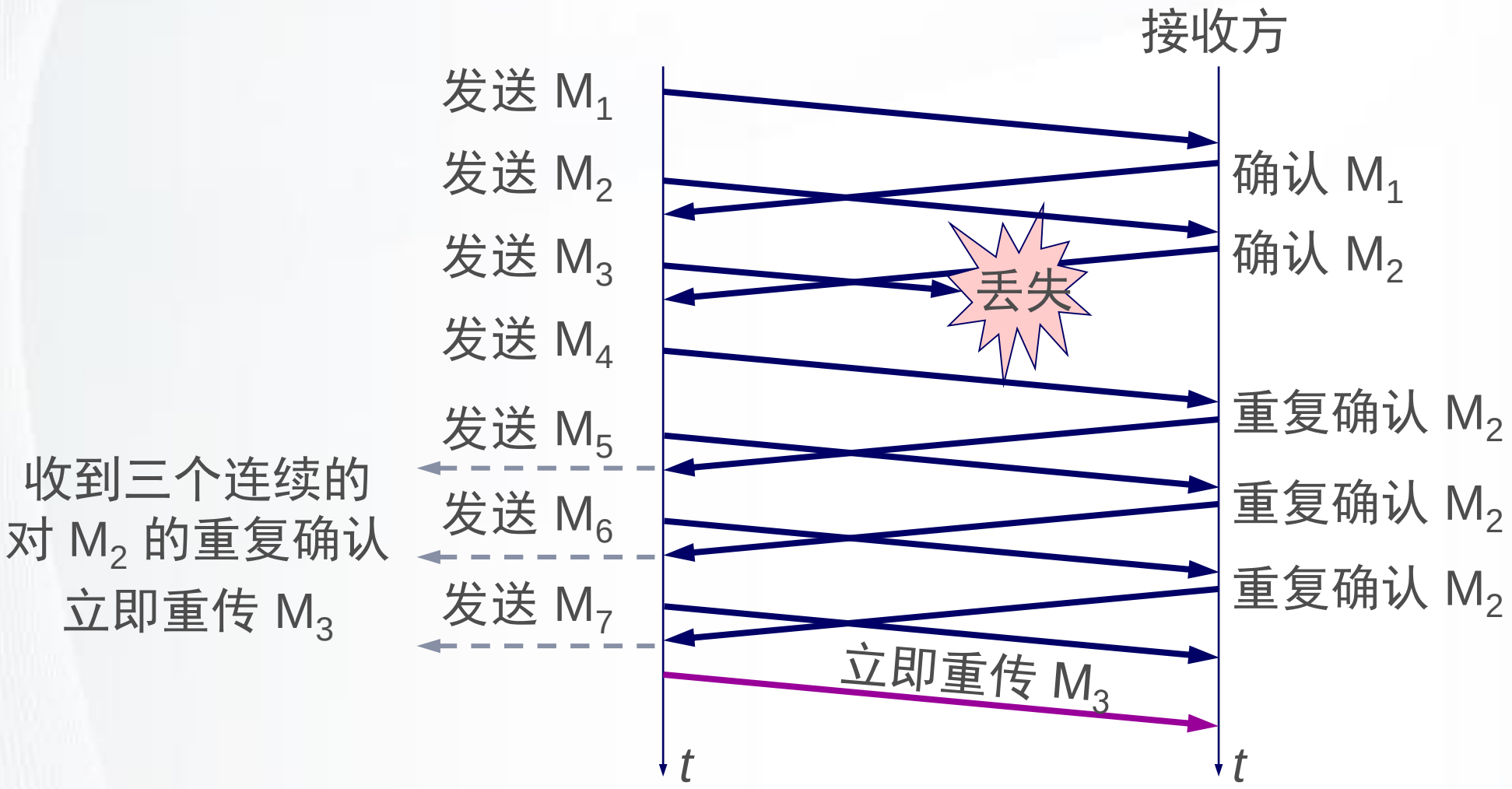
必须强调指出

- “拥塞避免”并非指完全能够避免了拥塞。利用以上的措施要完全避免网络拥塞还是不可能的。
- “拥塞避免”是说在拥塞避免阶段把拥塞窗口控制为按线性规律增长，使网络比较不容易出现拥塞。

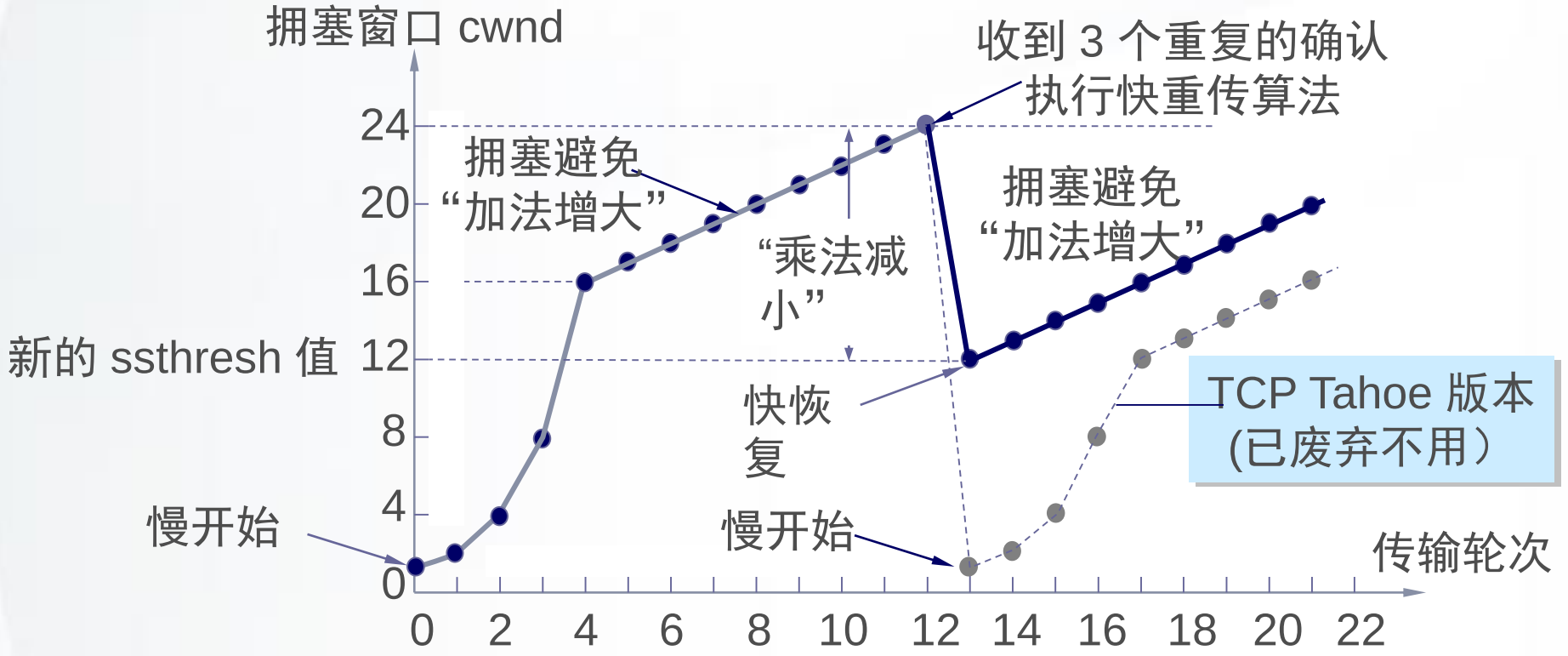
快重传和快恢复

- 快重传算法首先要求接收方每收到一个失序的报文段后就立即发出重复确认。
- 当发送端收到连续三个重复的确认时，就执行“乘法减小”算法，把慢开始门限 ssthresh 减半，但拥塞窗口 cwnd 现在不设置为 1，而是设置为慢开始门限 ssthresh 减半后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。

快重传举例


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
收到三个连续的
对 M_2 的重复确认
立即重传 M_3
立即重传 M_3
t
t

快恢复举例



从连续收到三个重复的确认转入拥塞避免

发送窗口的实际上限值

- 发送方的发送窗口的上限值应当取为接收方窗口 和拥塞窗口 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]

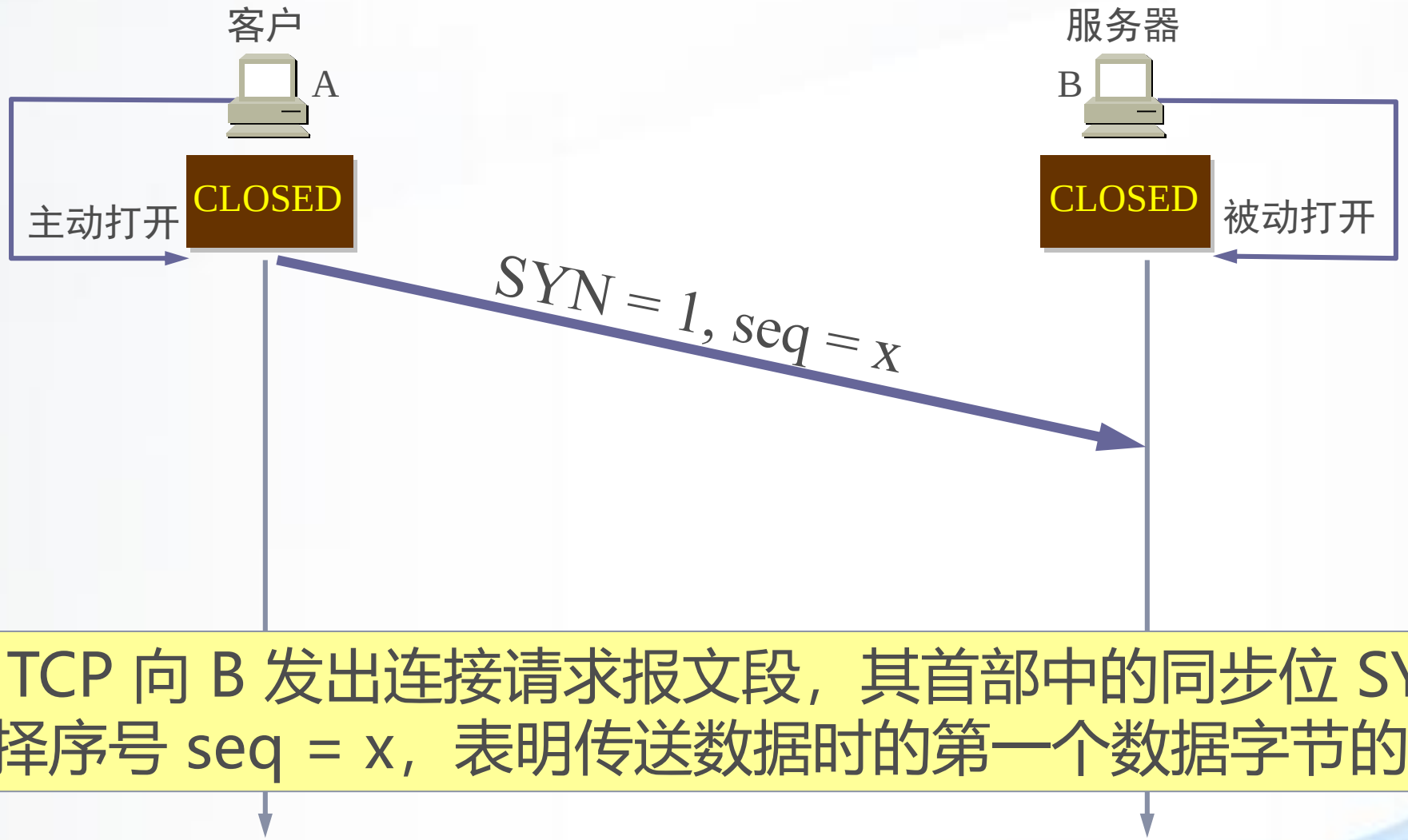
指引

- 运输层的功能
- 运输层协议UDP和TCP
- TCP可靠传输的实现
- TCP的流量控制
- TCP的拥塞控制
- TCP的运输连接管理

TCP的运输连接管理

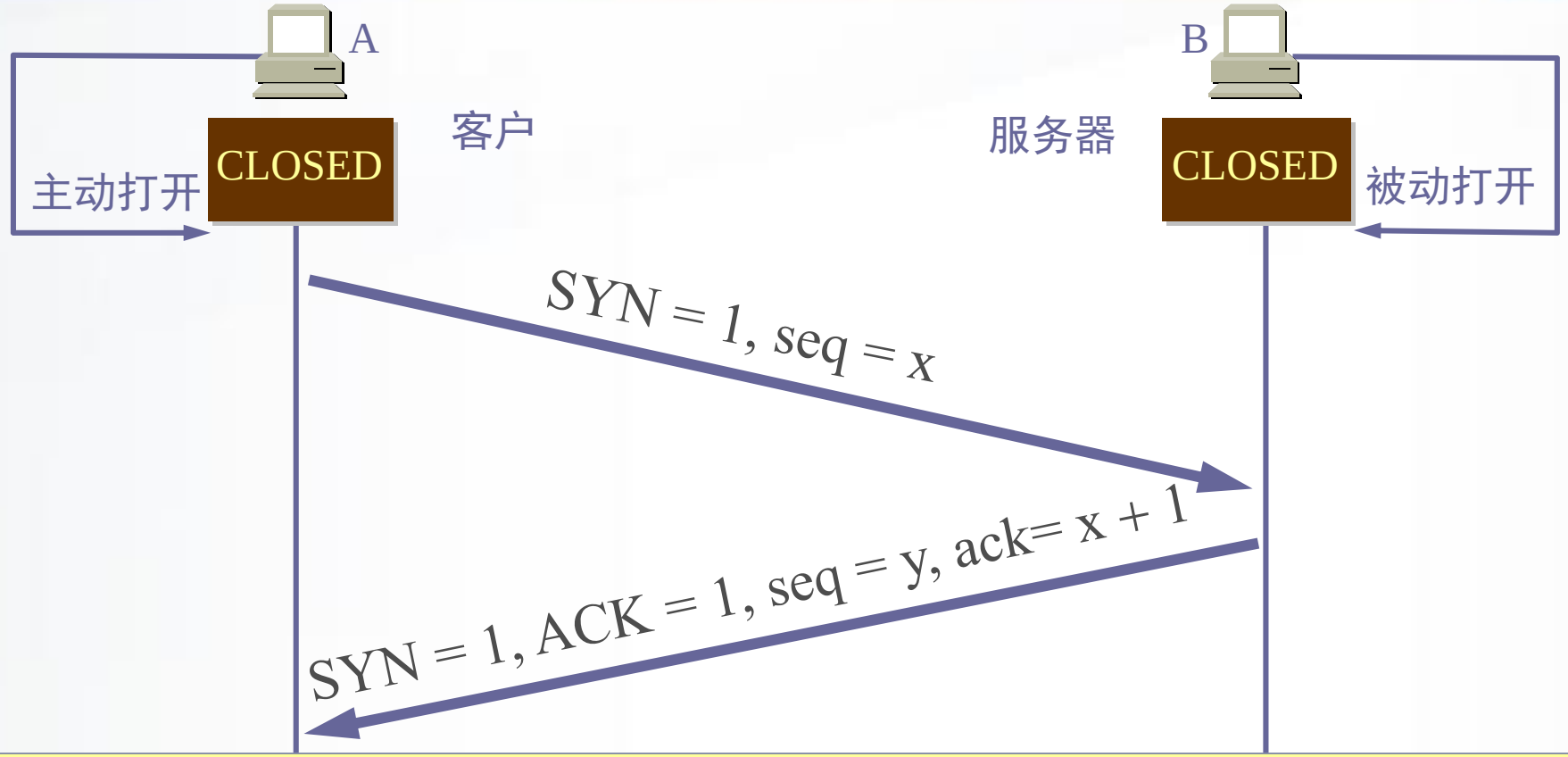
- 运输连接有三个阶段，即：连接建立、数据传送和连接释放。
- TCP 连接的建立都是采用客户服务器方式。
- 主动发起连接建立的应用进程叫做客户(client)。
- 被动等待连接建立的应用进程叫做服务器(server)。

TCP的连接建立



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。

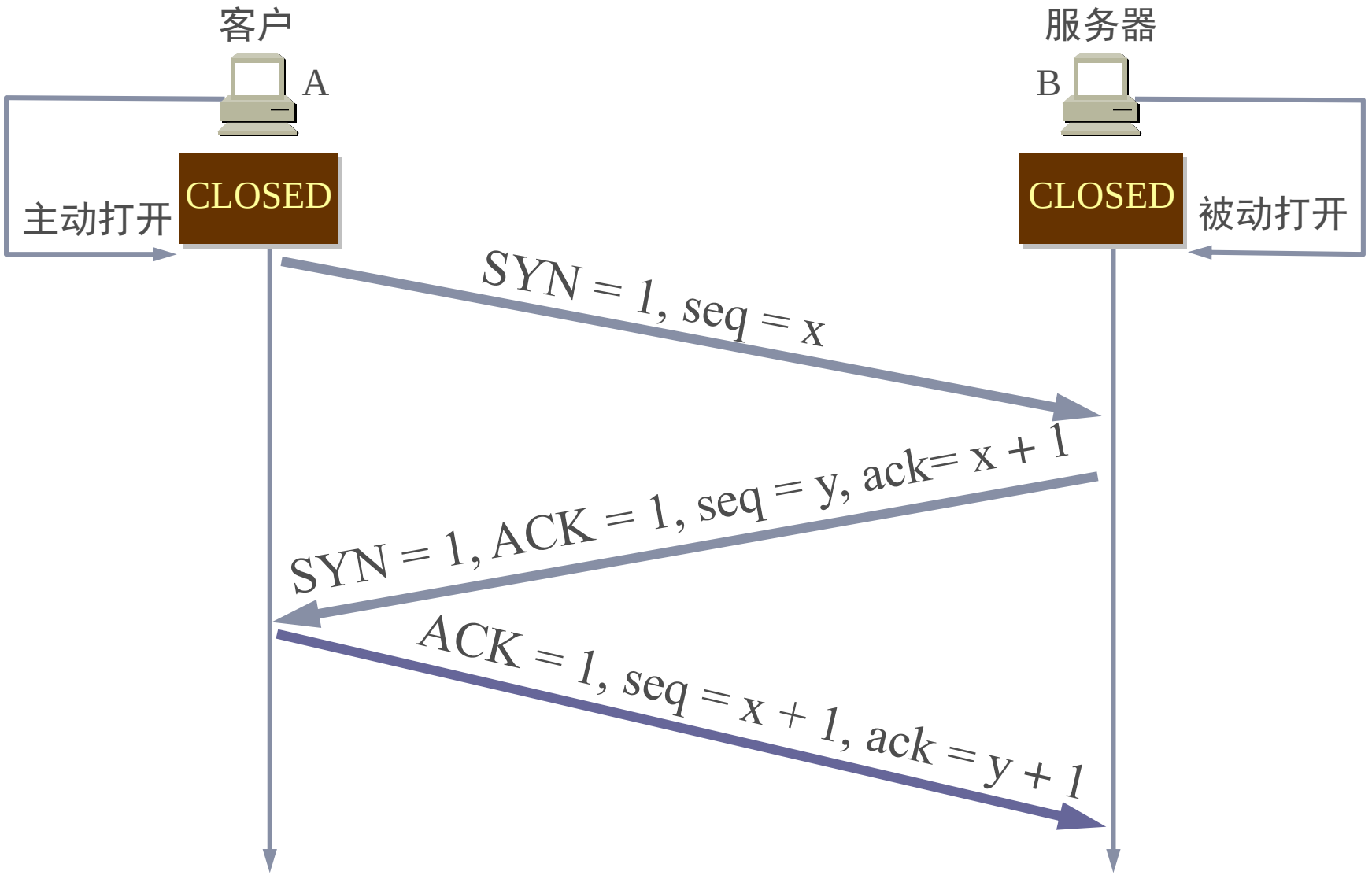
TCP 的连接建立

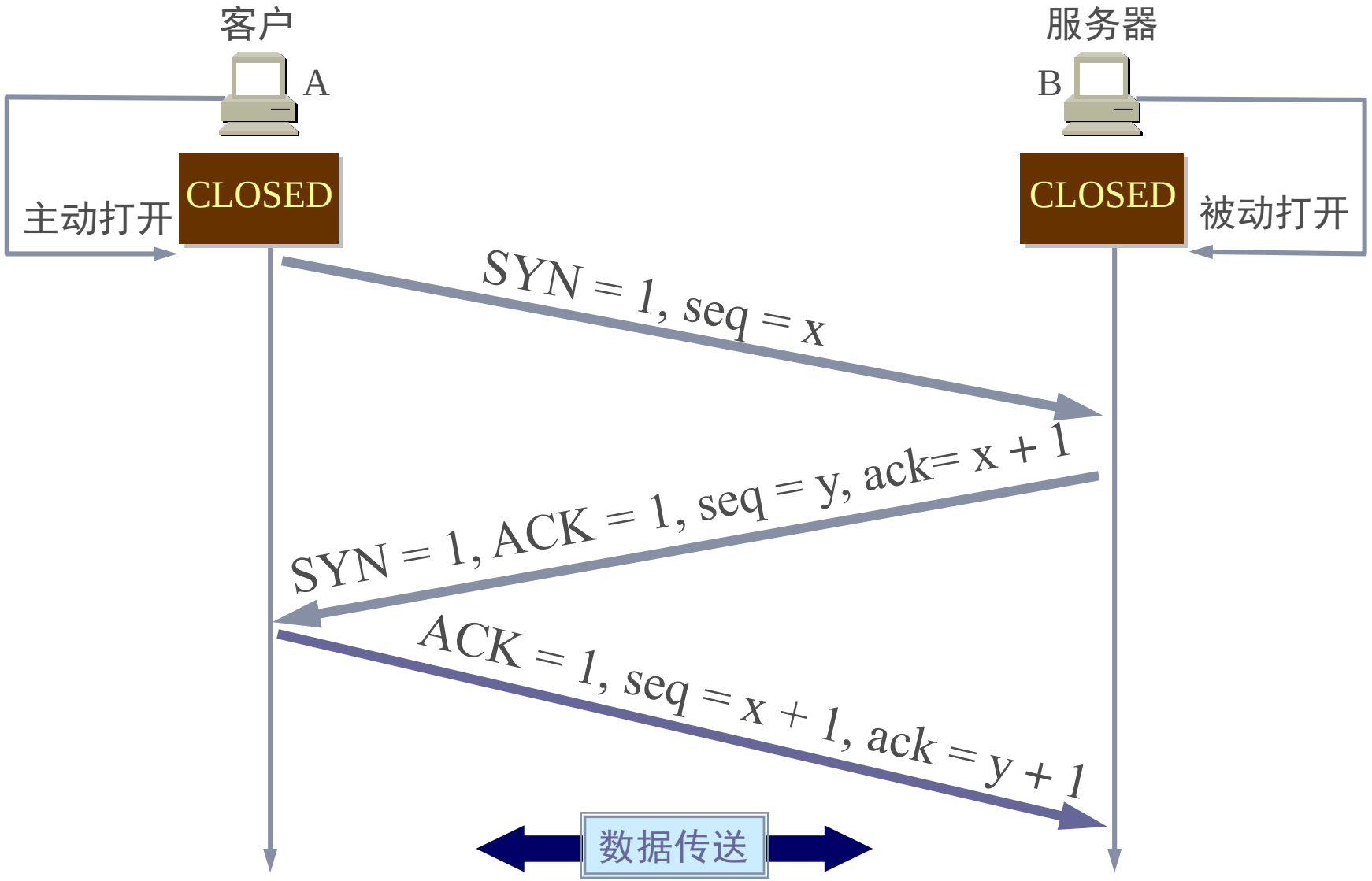


B 的 TCP 收到连接请求报文段后，如同意，则 发回确认。

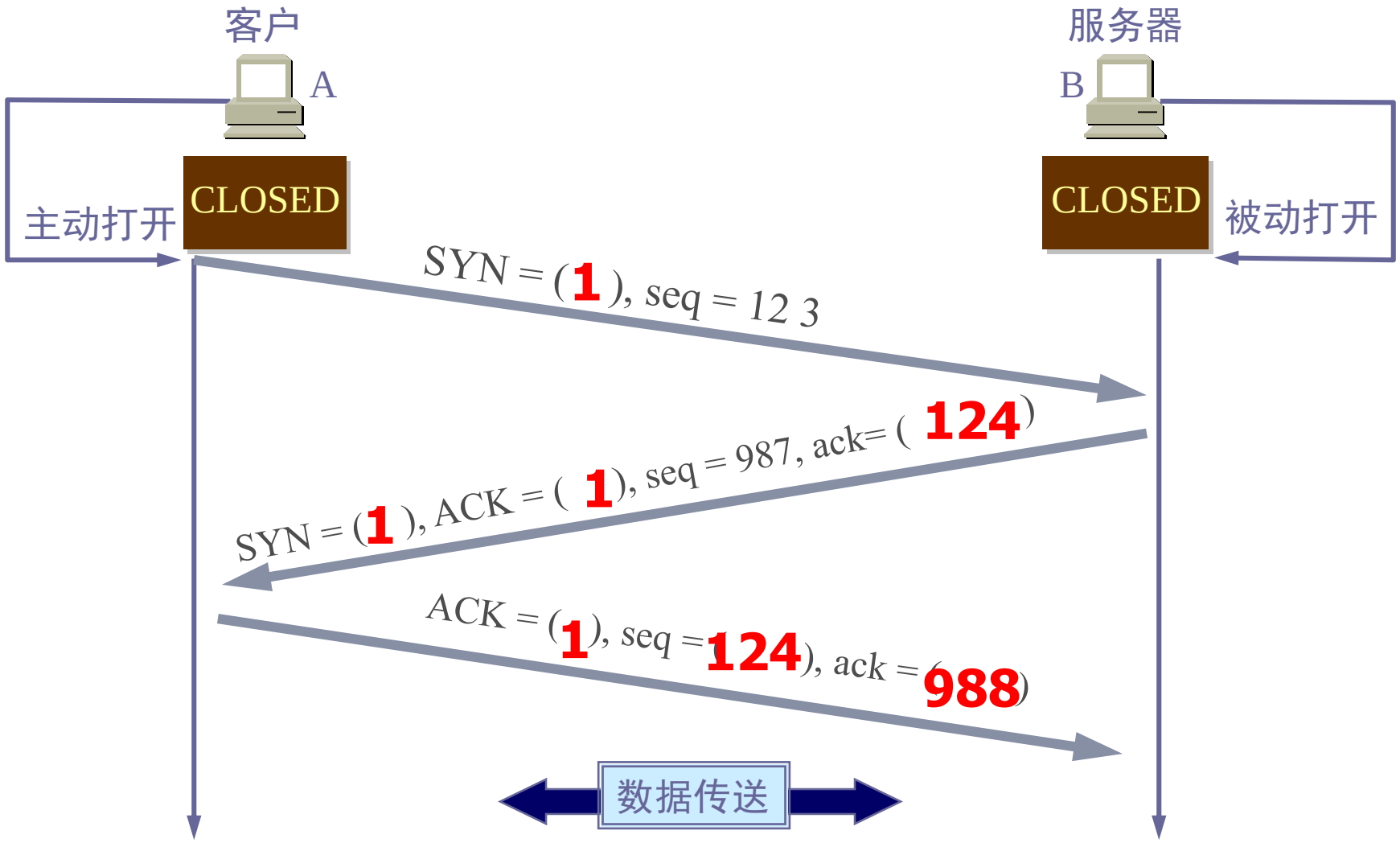
B 在确认报文段中应使 SYN = 1，使 ACK = 1， 其确认号ack = x + 1，自己选择的序号 seq = y。

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。

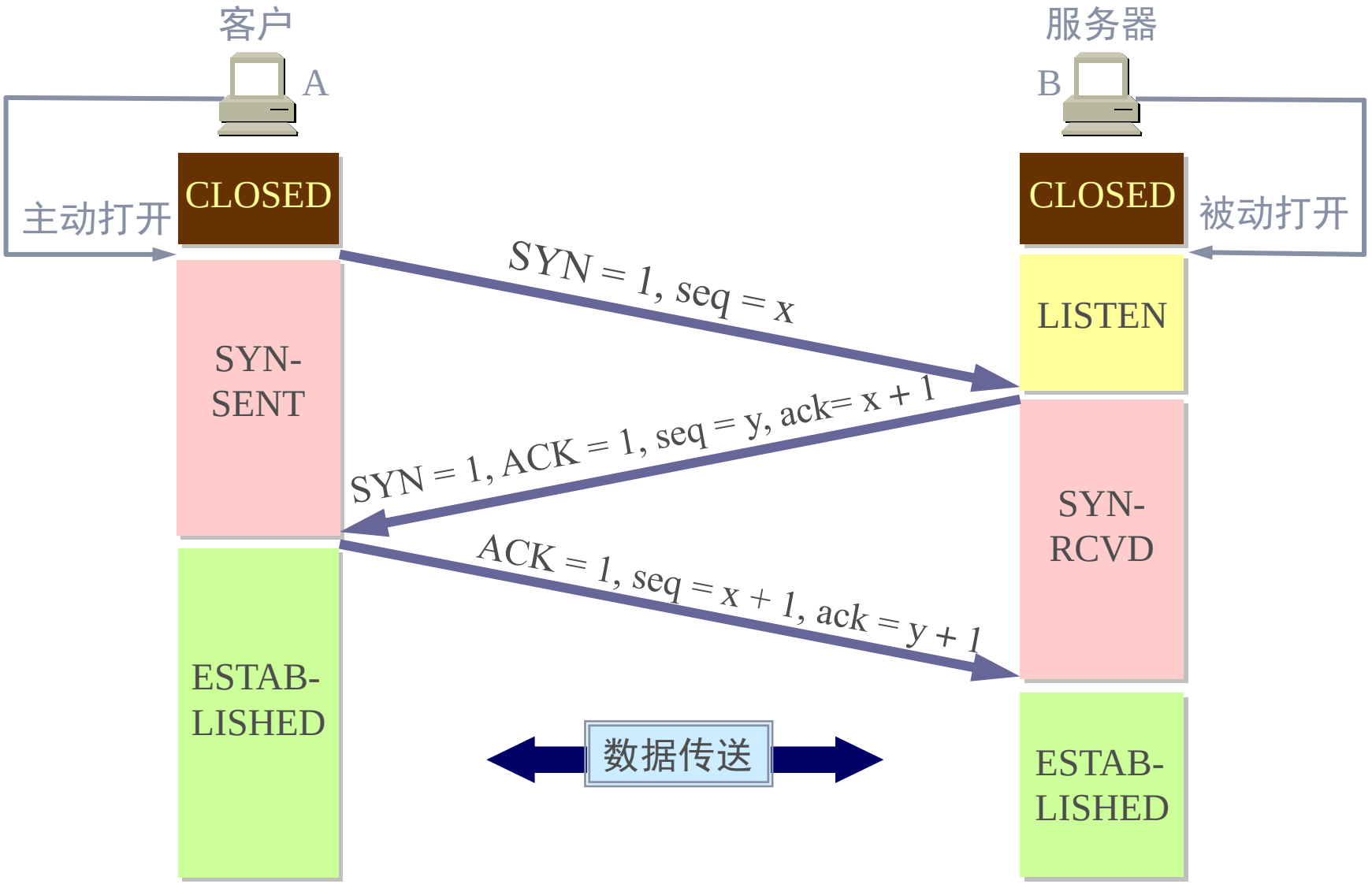




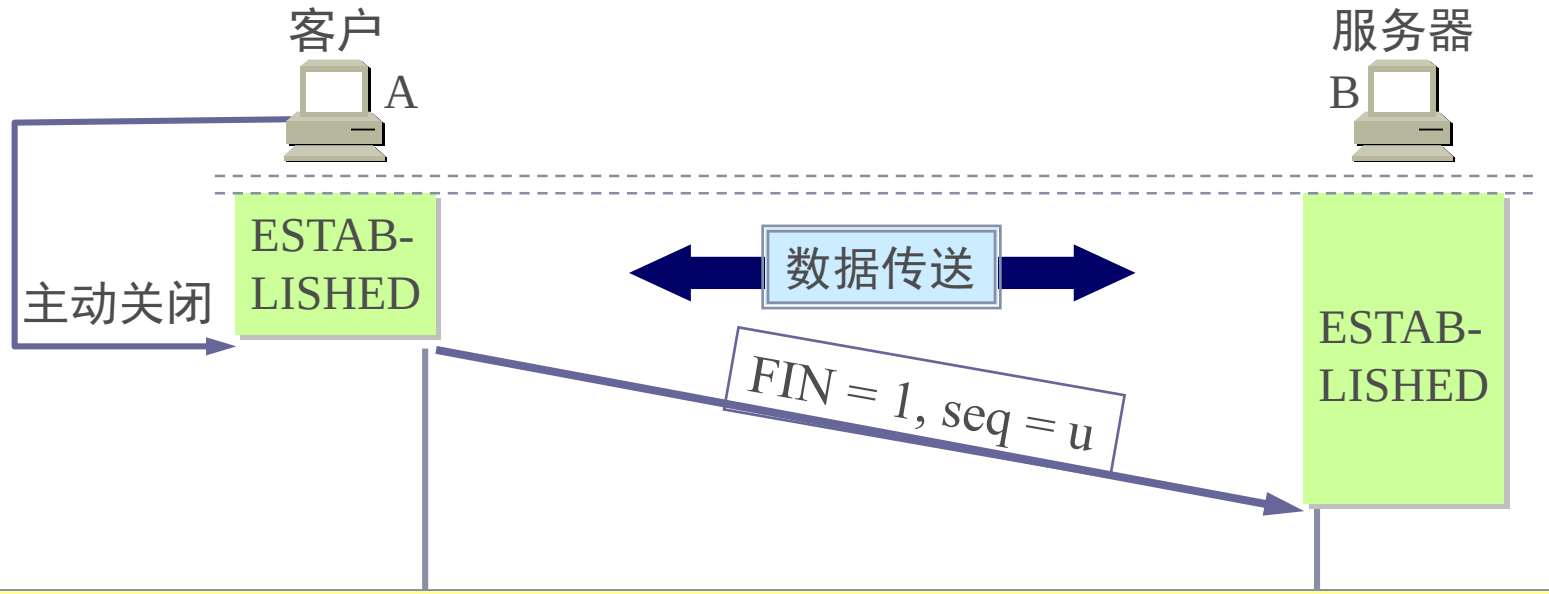
- B 的 TCP 收到主机 A 的确认后，也通知其上层应用进程：TCP 连接已经建立。


客户
A
服务器
B
主动打开
CLOSED
CLOSED
被动打开
SYN = (1), seq = 12 3
SYN = (1), ACK = (1), seq = 987, ack= (124)
ACK = (1), seq = 124), ack = 988)
数据传送

TCP 的连接建立


客户
A
服务器
B
主动打开
CLOSED
CLOSED
被动打开
SYN = 1, seq = x
LISTEN
SYN-
SENT
SYN = 1, ACK = 1, seq = y, ack= x + 1
SYN-
RCVD
ACK = 1, seq = x + 1, ack = y + 1
ESTAB-
LISHED
数据传送
ESTAB-
LISHED

TCP 的连接释放

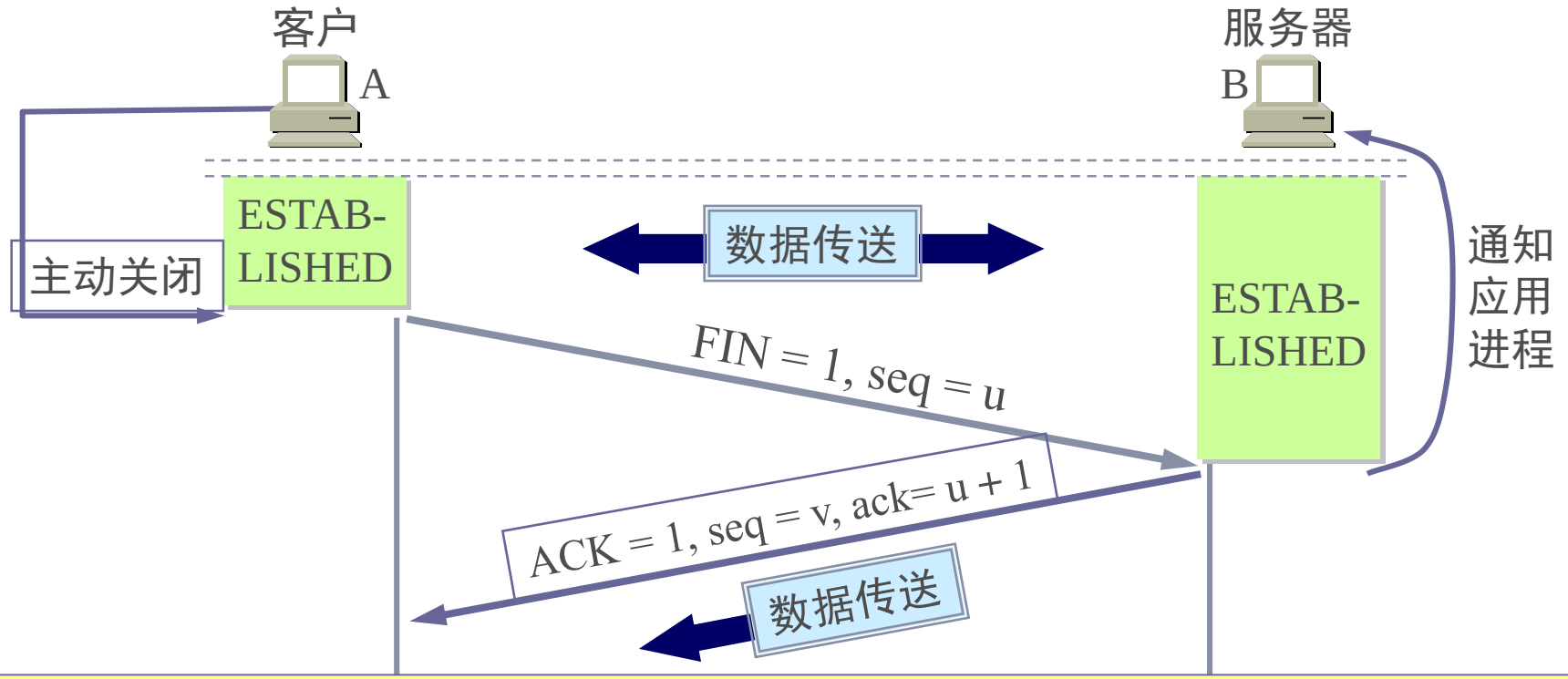


- 数据传输结束后，通信的双方都可释放连接。

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。

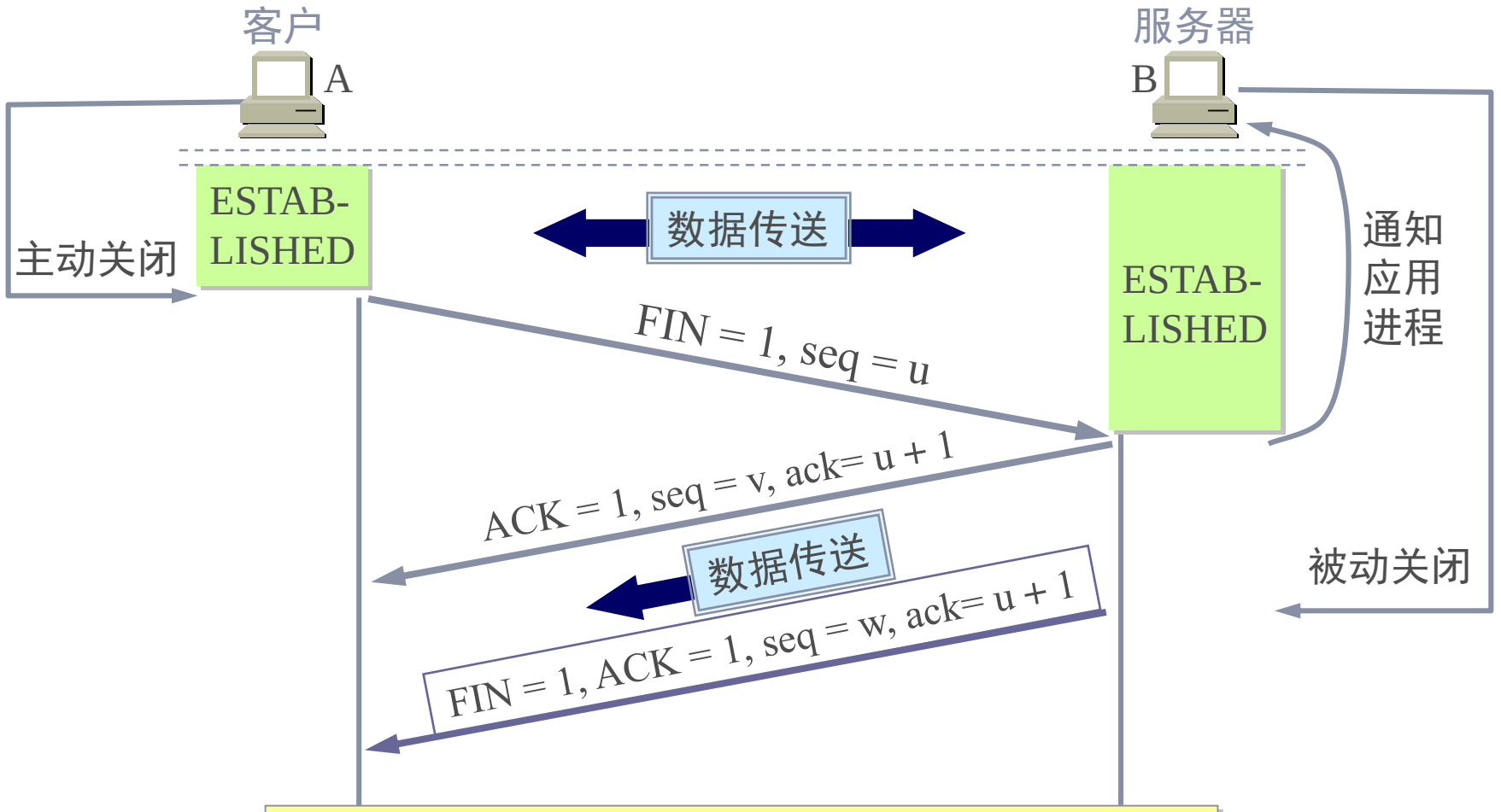
- A 把连接释放报文段首部的 FIN = 1，其序号seq = u，等待 B 的确认。

TCP 的连接释放



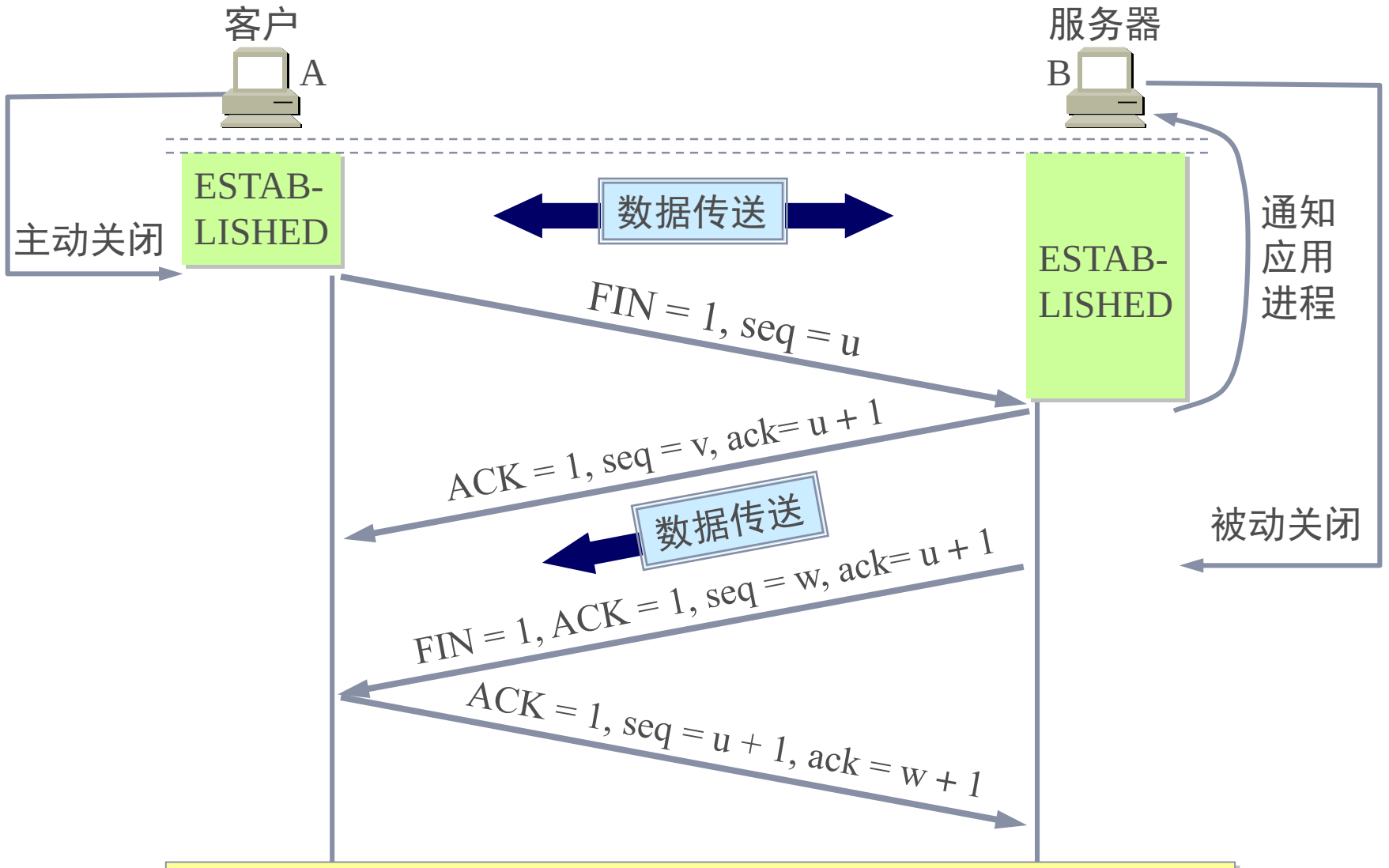
- B 发出确认，确认号 ack = u + 1，而这个报文段自己的序号 seq = v。
- TCP 服务器进程通知高层应用进程。
- 从 A 到 B 这个方向的连接就释放了，TCP 连接处于半关闭状态。B 若发送数据，A 仍要接收。

TCP 的连接释放



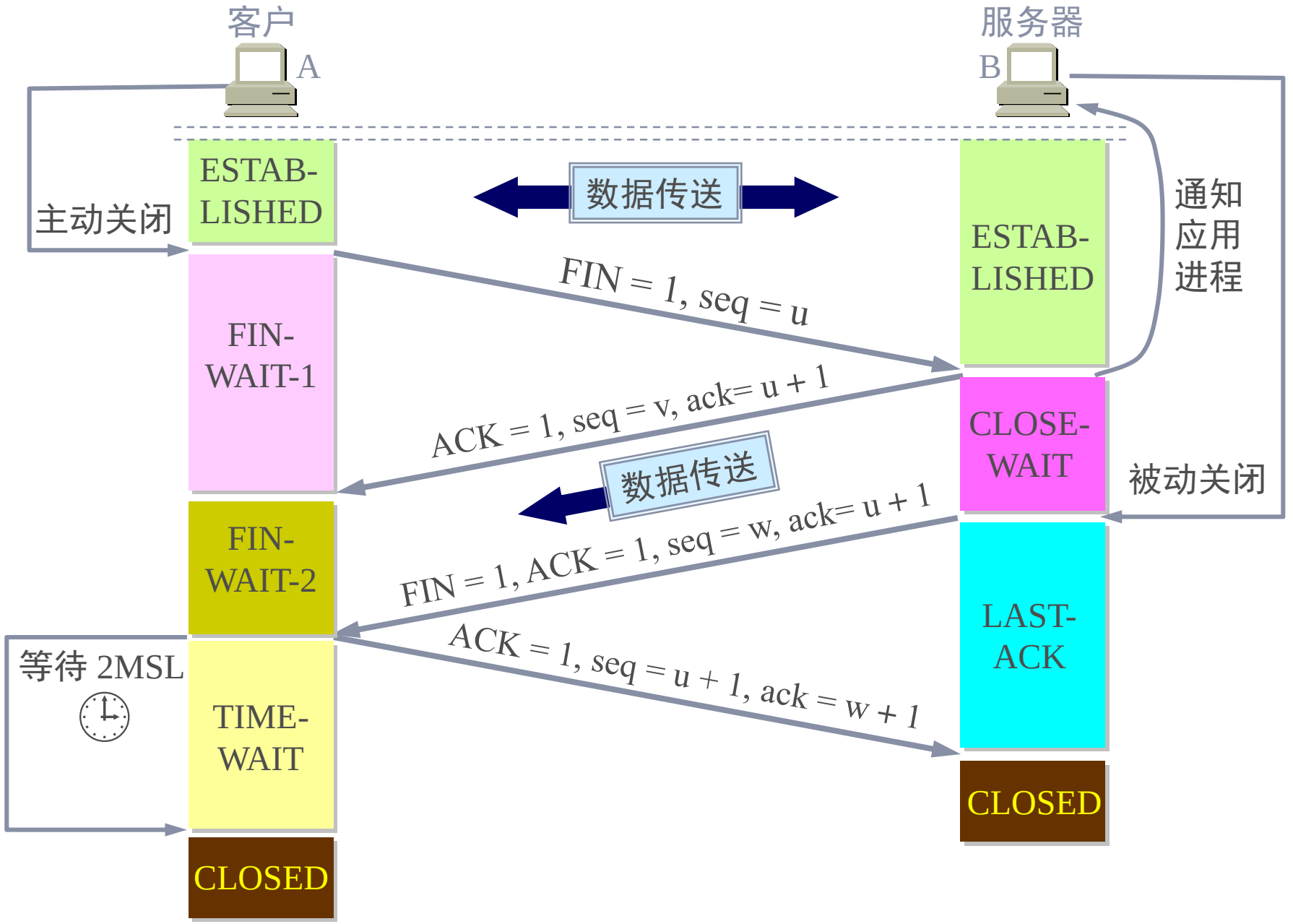
- 若 B 已经没有要向 A 发送的数据，其应用进程就通知 TCP 释放连接。

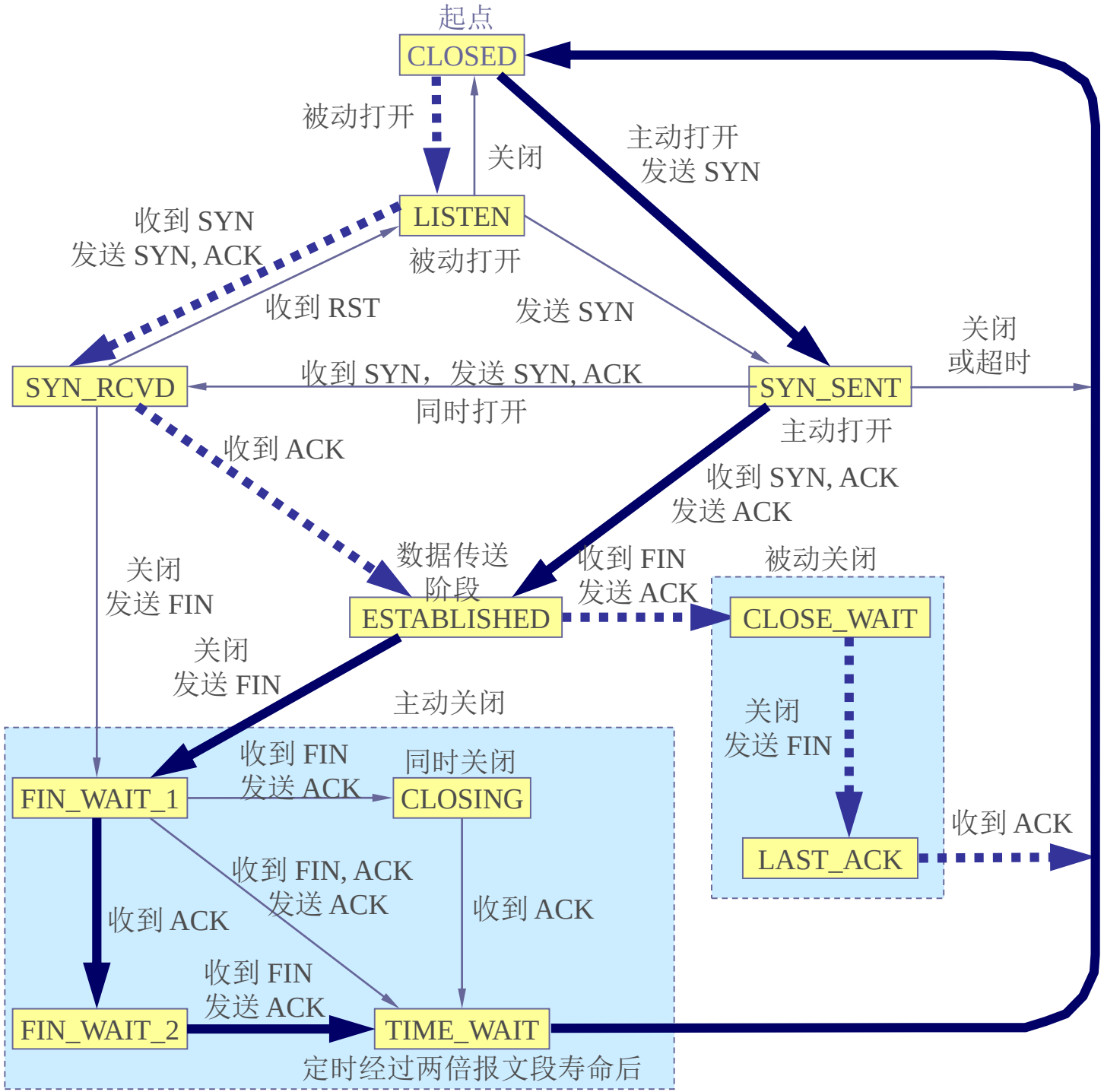
TCP 的连接释放



- A 收到连接释放报文段后，必须发出确认。

TCP 连接必须经过时间 2MSL 后才真正释放掉。




起点
CLOSED
被动打开
关闭
主动打开
发送 SYN
LISTEN
收到 SYN
发送 SYN, ACK
被动打开
收到 RST
发送 SYN
关闭
或超时
SYN_RCVD
收到 SYN，发送 SYN, ACK
同时打开
SYN_SENT
主动打开
收到 ACK
收到 SYN, ACK
发送 ACK
关闭
发送 FIN
数据传送
阶段
收到 FIN
发送 ACK
被动关闭
ESTABLISHED
CLOSE_WAIT
关闭
发送 FIN
主动关闭
关闭
发送 FIN
收到 FIN
发送 ACK
同时关闭
FIN_WAIT_1
CLOSING
关闭
发送 FIN
收到 ACK
收到 FIN, ACK
发送 ACK
收到 ACK
收到 ACK
收到 FIN
发送 ACK
FIN_WAIT_2
TIME_WAIT
定时经过两倍报文段寿命后

本章小结

- TCP
- UDP
- 端口
- 流量控制
- 拥塞控制
- 三次握手

Thank You!
Any Questions?