石家庄铁道大学
SHIJIAZHUANG TIEDAO UNIVERSITY

# □算机网□

## 第 18 讲　运输层（IV）

# 上讲内容回顾

- TCP可靠传输的实现

  滑动窗口，超时重传时间的选择，选择确认

- TCP的流量控制

  利用滑动窗口实现流量控制，传输效率

# 本讲内容

- TCP的拥塞控制

  拥塞控制的原理，拥塞控制方法，随机早期检测

- TCP的运输连接管理

  TCP的连接建立，连接释放，有限状态机

# TCP的拥塞控制

## 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：

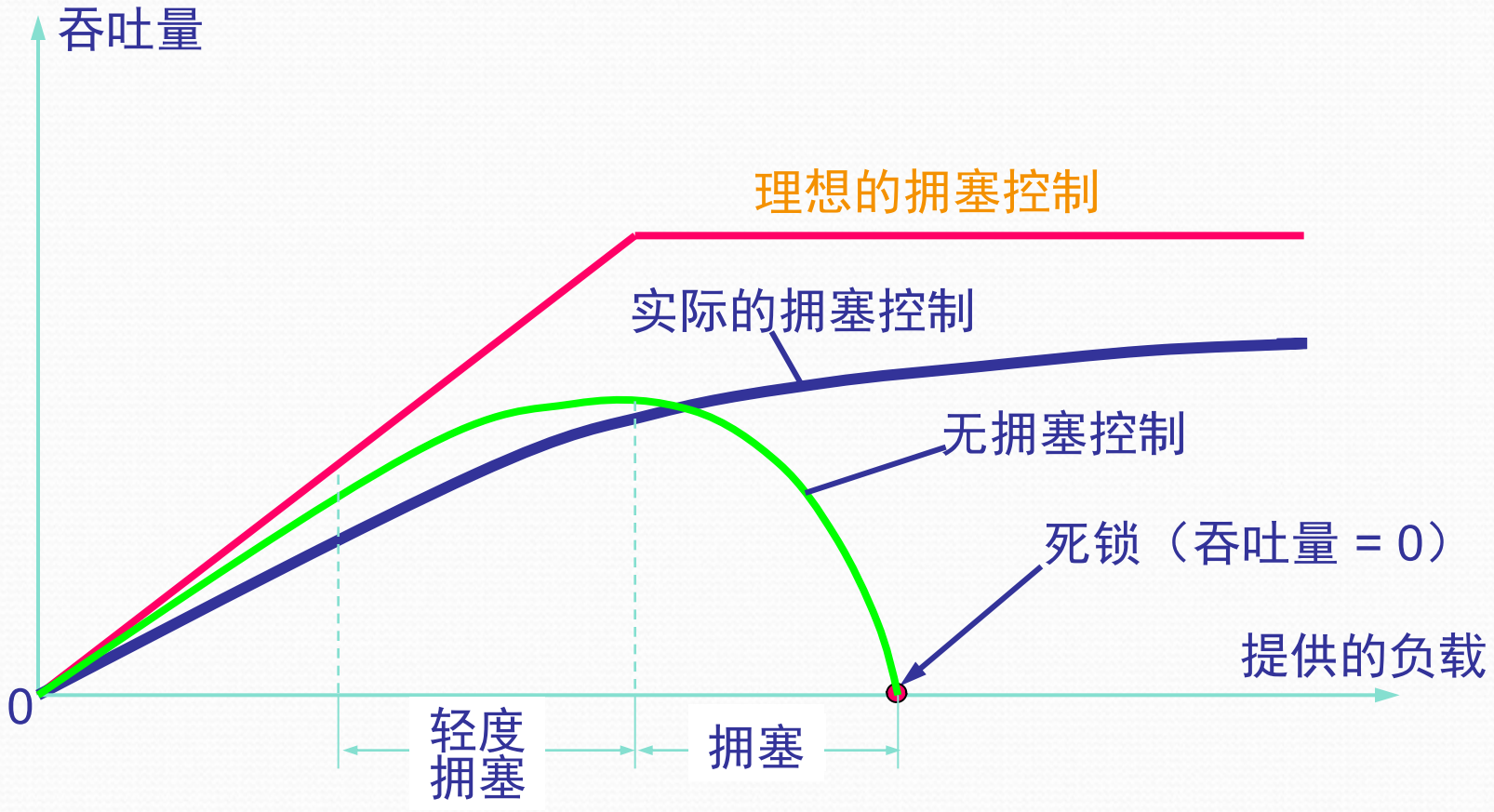
  对资源需求的总和 > 可用资源　　(5-7)

- 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。

# 拥塞控制所起的作用



# 拥塞控制的一般原理

- 拥塞控制是很难设计的，因为它是一个动态的（而不是静态的）问题。
- 当前网络正朝着高速化的方向发展，这很容易出现缓存不够大而造成分组的丢失。但分组的丢失是网络发生拥塞的征兆而不是原因。
- 在许多情况下，甚至正是拥塞控制本身成为引起网络性能恶化甚至发生死锁的原因。这点应特别引起重视。

# 开环控制和闭环控制

- 开环控制方法就是在设计网络时事先将有关发生拥塞的因素考虑周到，力求网络在工作时不产生拥塞。
- 闭环控制是基于反馈环路的概念。属于闭环控制的有以下几种措施：
  - 监测网络系统以便检测到拥塞在何时、何处发生。
  - 将拥塞发生的信息传送到可采取行动的地方。
  - 调整网络系统的运行以解决出现的问题。

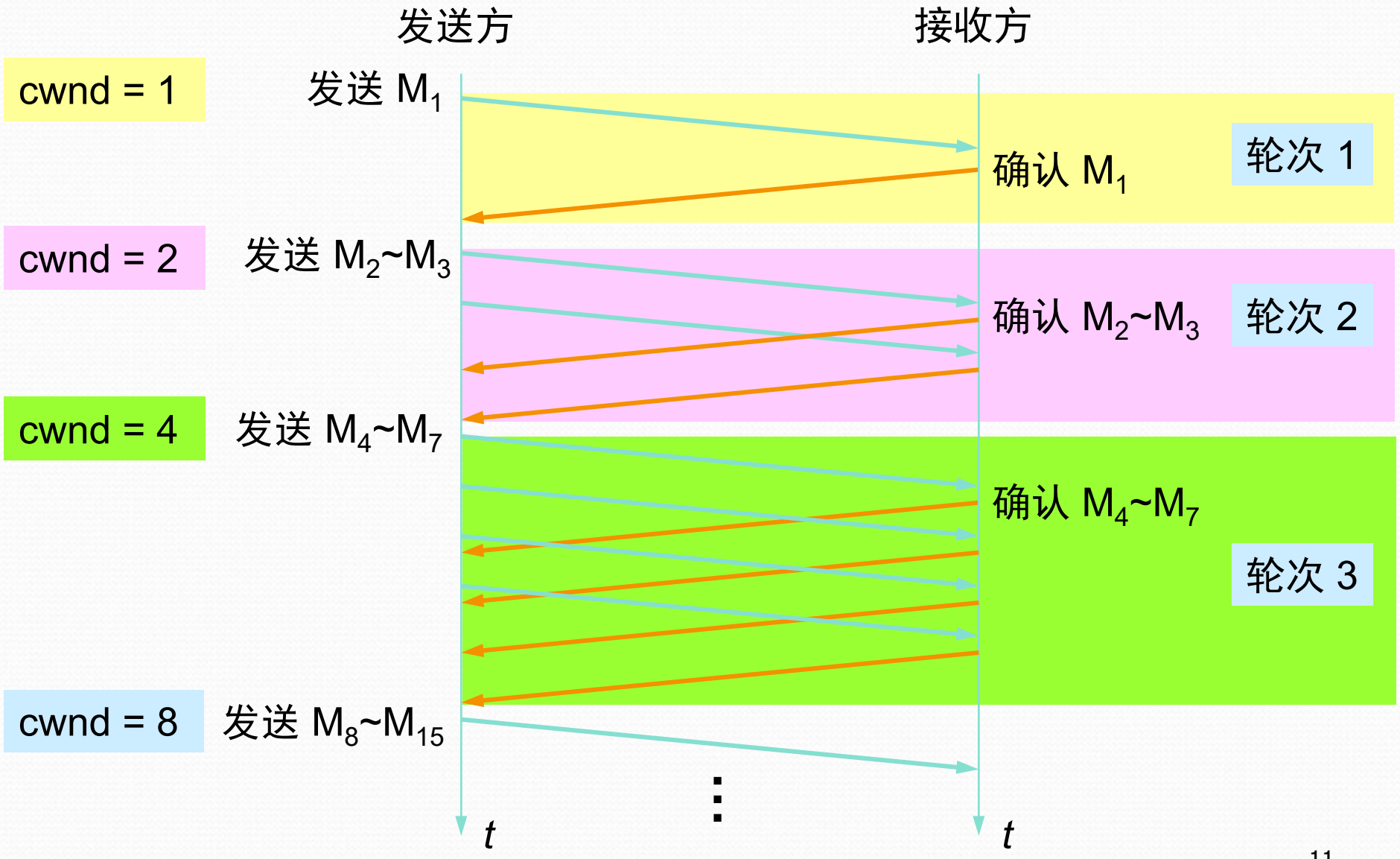
# 几种拥塞控制方法
# 1. 慢开始和拥塞避免

- 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。如再考虑到接收方的接收能力，则发送窗口还可能小于拥塞窗口。
- 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。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，即设置为一个最大报文段 MSS 的数值。
- 在每收到一个对新的报文段的确认后，将拥塞窗口加 1，即增加一个 MSS 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。



# 传输轮次 (transmission round)

- 使用慢开始算法后，每经过一个传输轮次，拥塞窗口 cwnd 就加倍。
- 一个传输轮次所经历的时间其实就是往返时间 RTT。
- “传输轮次”更加强调：把拥塞窗口 cwnd 所允许发送的报文段都连续发送出去，并收到了对已发送的最后一个字节的确认。
- 例如，拥塞窗口 cwnd = 4，这时的往返时间 RTT 就是发送方连续发送 4 个报文段，并收到这 4 个报文段的确认，总共经历的时间。

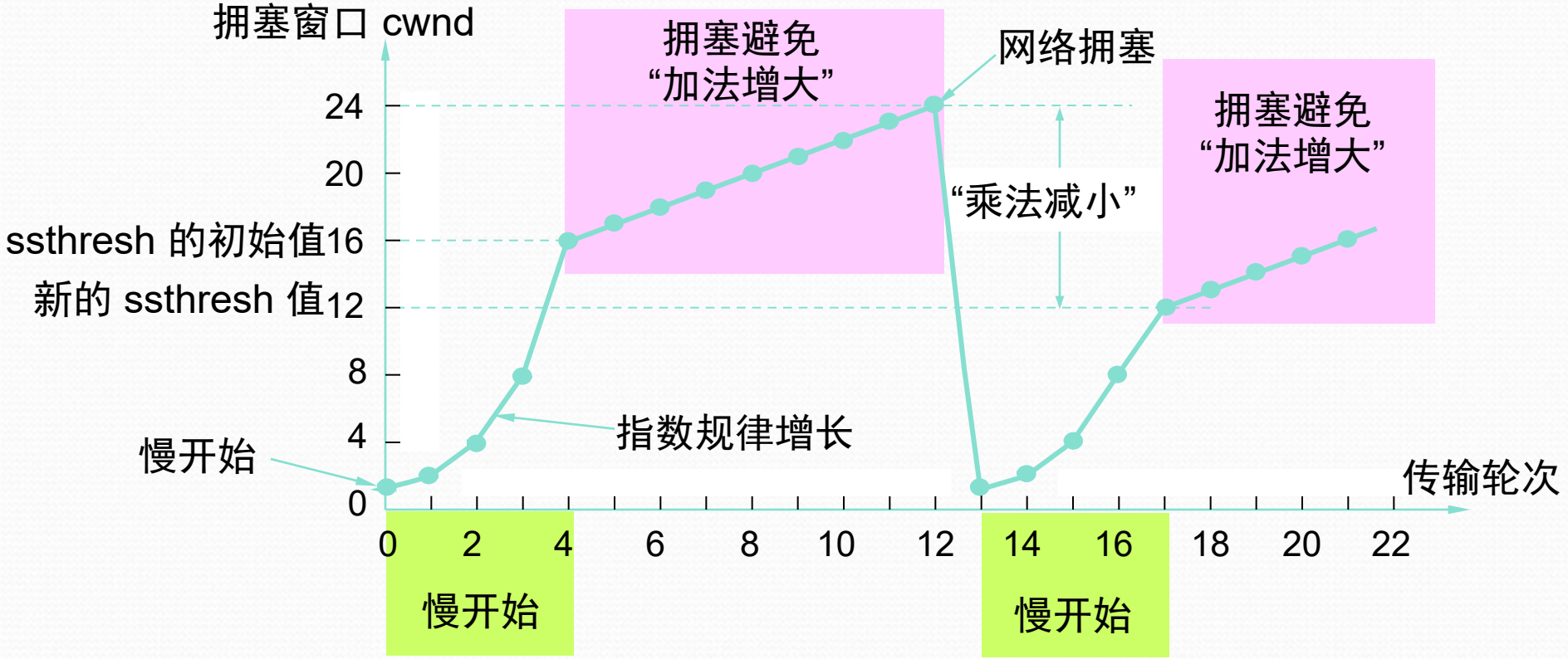
# 设置慢开始门限状态变量 ssthresh

- 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（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。
- 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。

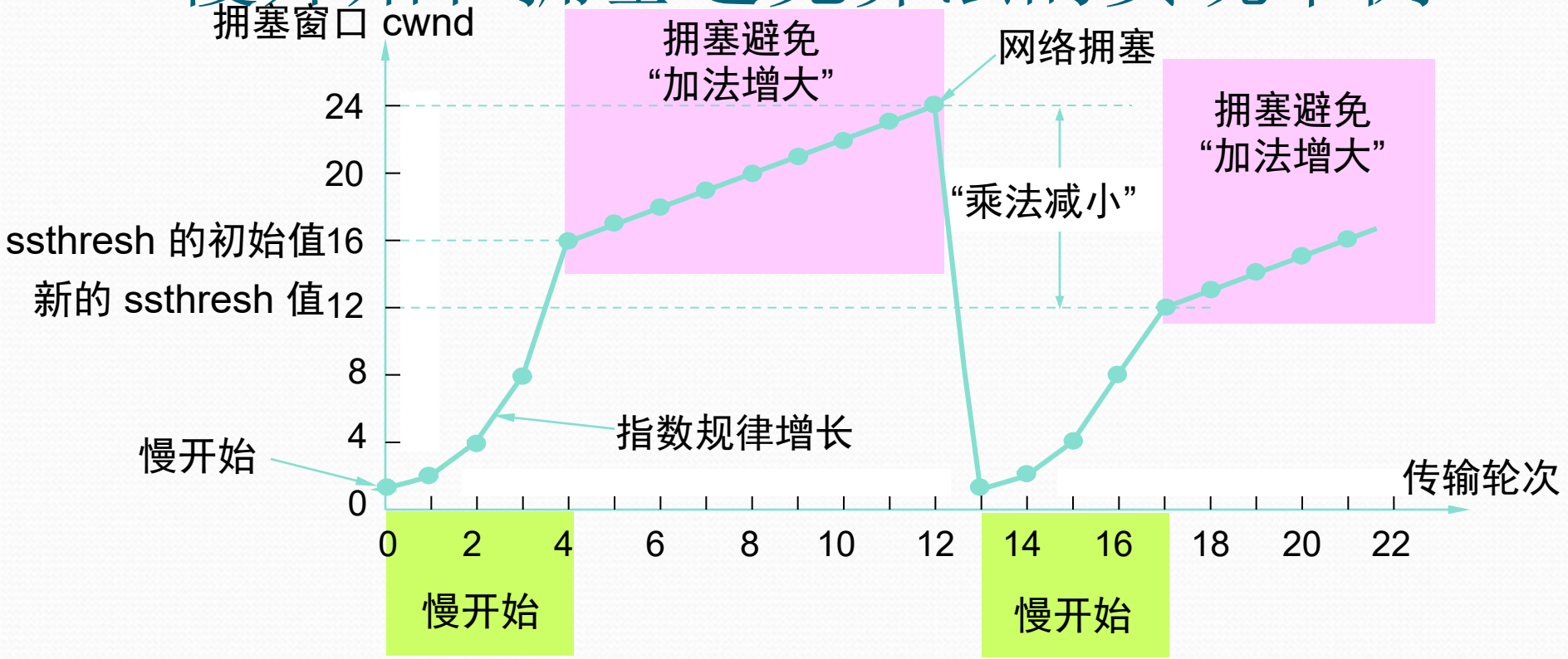
# 慢开始和拥塞避免算法的实现举例




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。

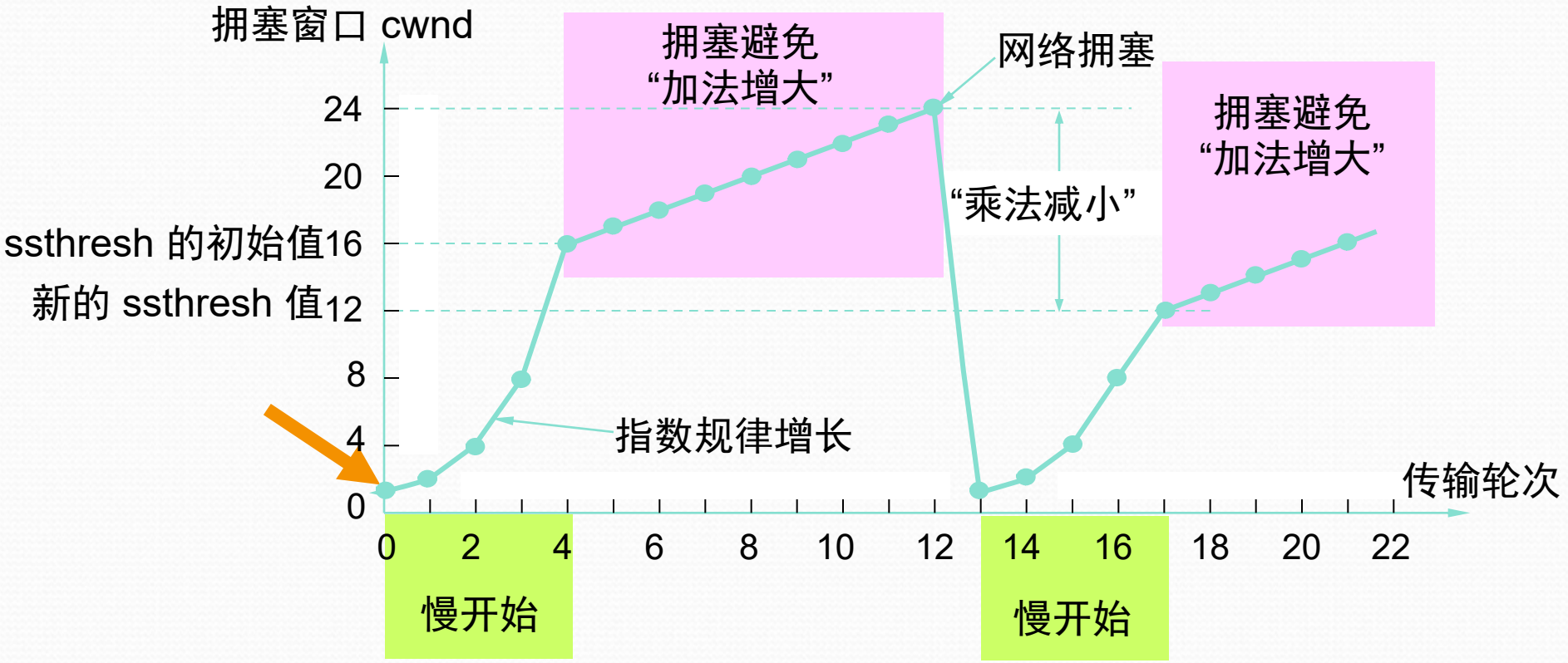
慢开始门限的初始值设置为 16 个报文段，即 ssthresh = 16。

# 慢开始和拥塞避免算法的实现举例



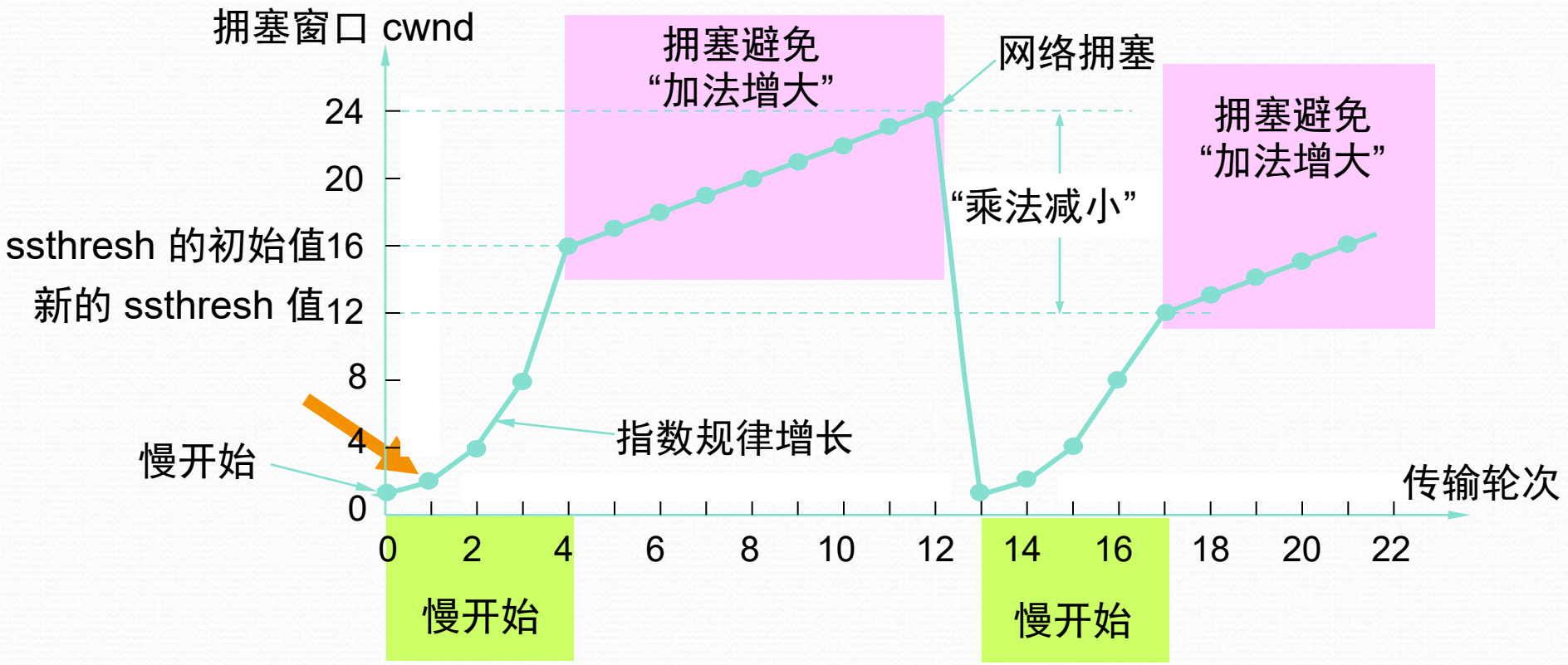

发送端的发送窗口不能超过拥塞窗口 cwnd 和接收端窗口 rwnd 中的最小值。我们假定接收端窗口足够大，因此现在发送窗口的数值等于拥塞窗口的数值。

# 慢开始和拥塞避免算法的实现举例



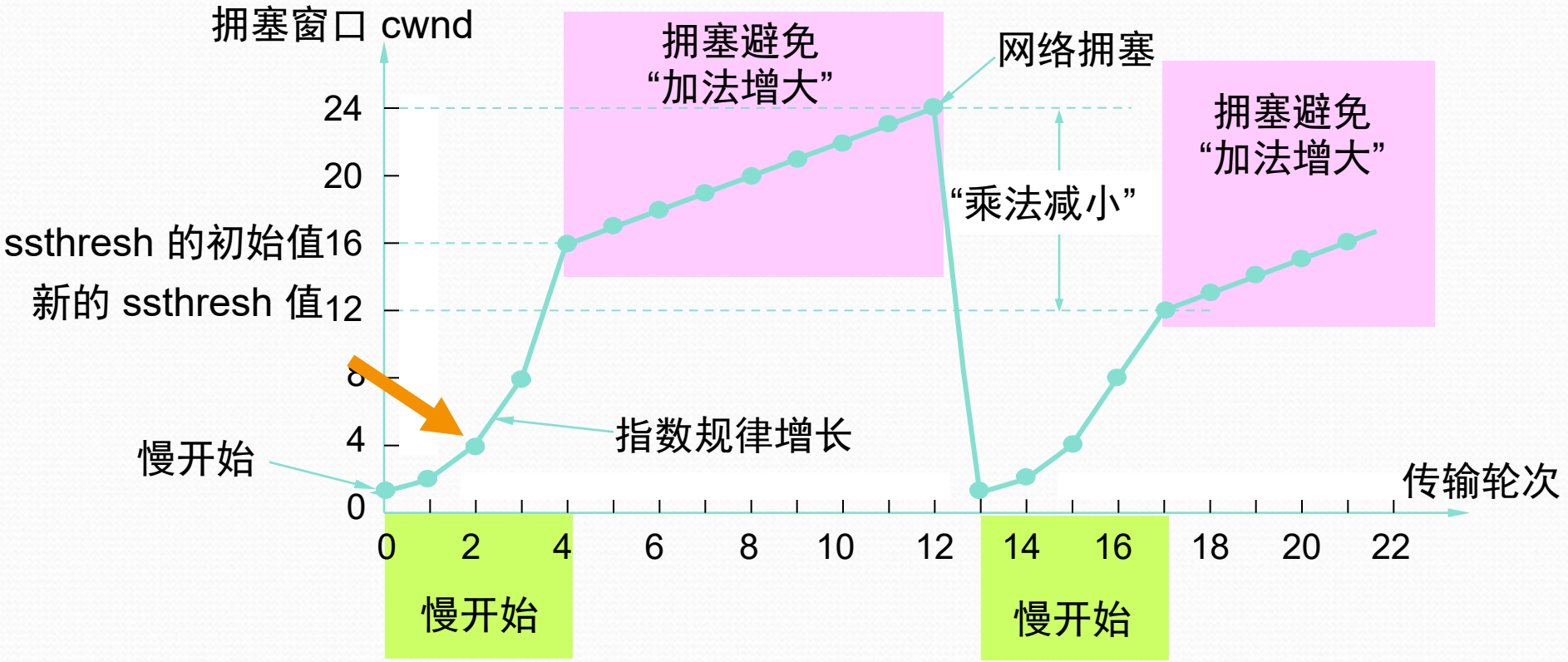

在执行慢开始算法时，拥塞窗口 cwnd 的初始值为 1，发送第一个报文段 $M_0$。

# 慢开始和拥塞避免算法的实现举例



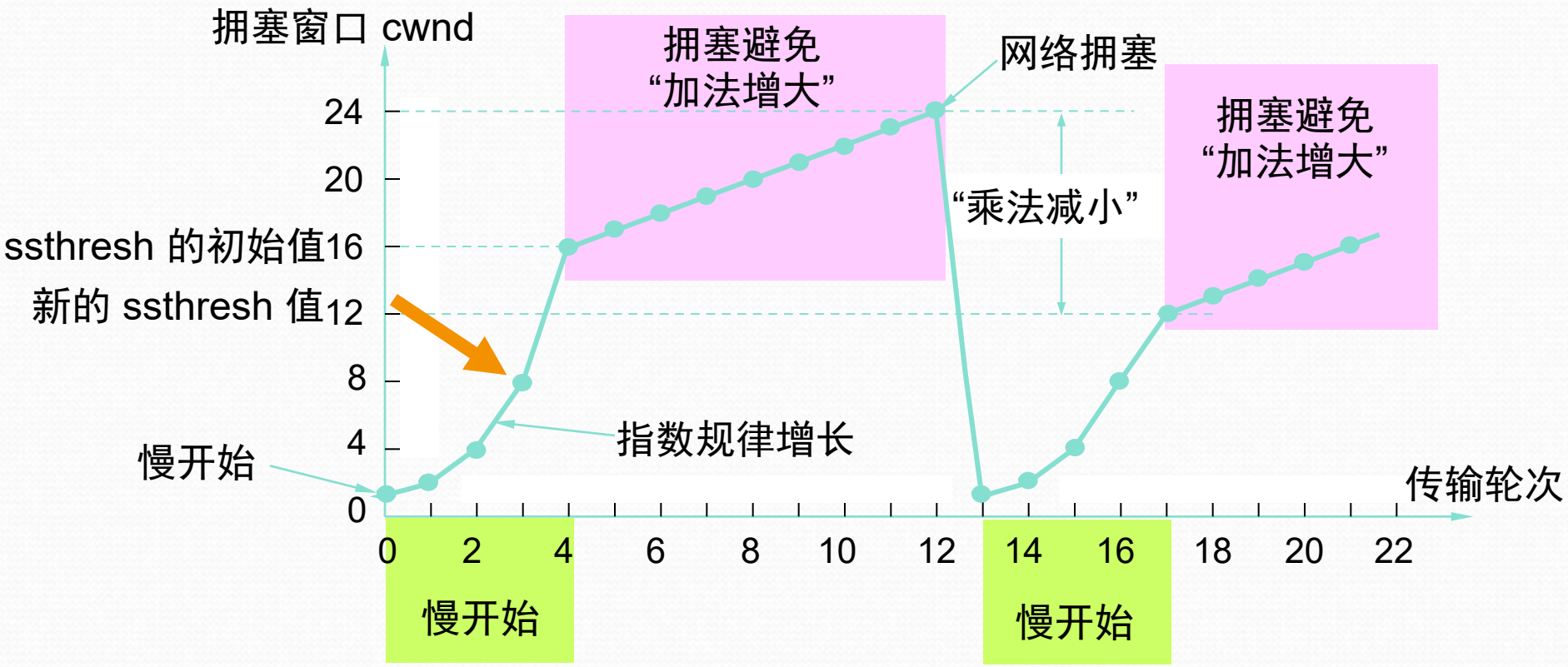

发送端每收到一个确认，就把 cwnd 加 1。于是发送端可以接着发送 $M_1$ 和 $M_2$ 两个报文段。

# 慢开始和拥塞避免算法的实现举例



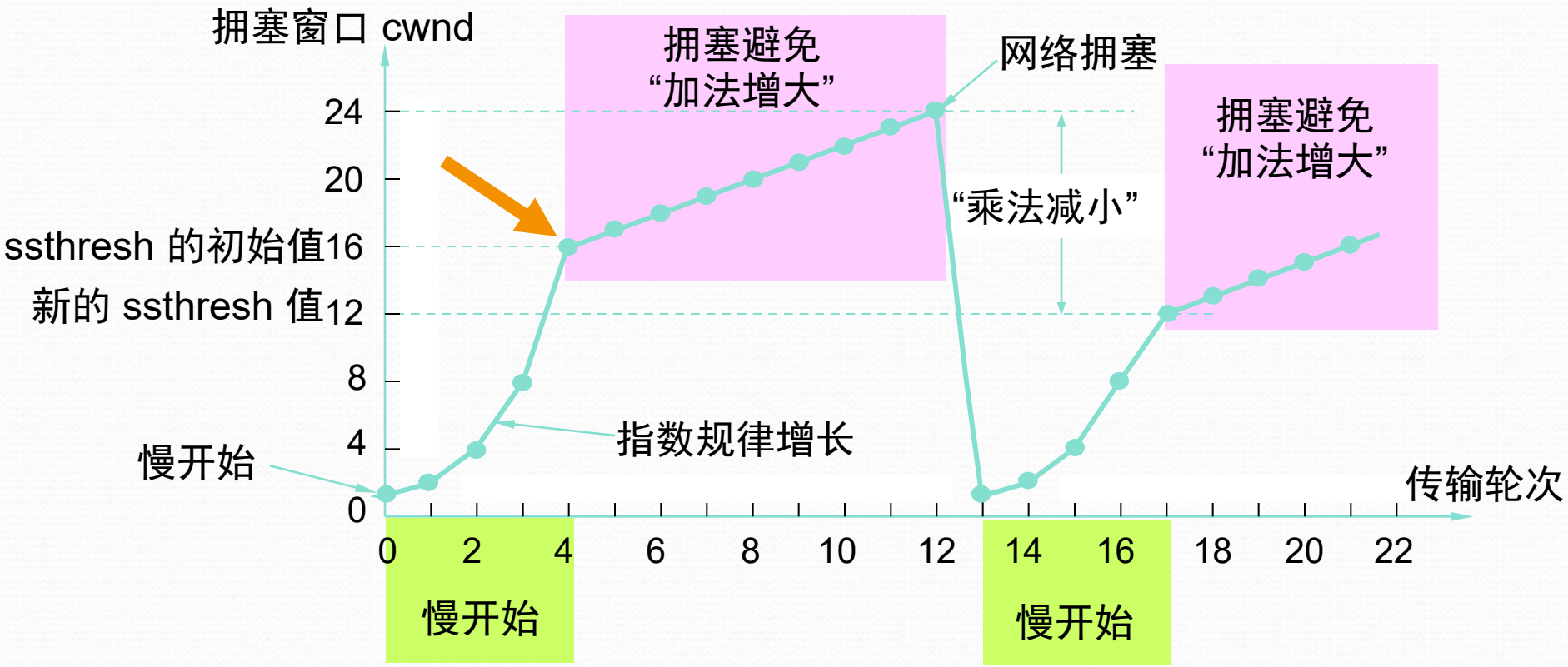

接收端共发回两个确认。发送端每收到一个对新报文段的确认，就把发送端的 cwnd 加 1。现在 cwnd 从 2 增大到 4，并可接着发送后面的 4 个报文段。

# 慢开始和拥塞避免算法的实现举例



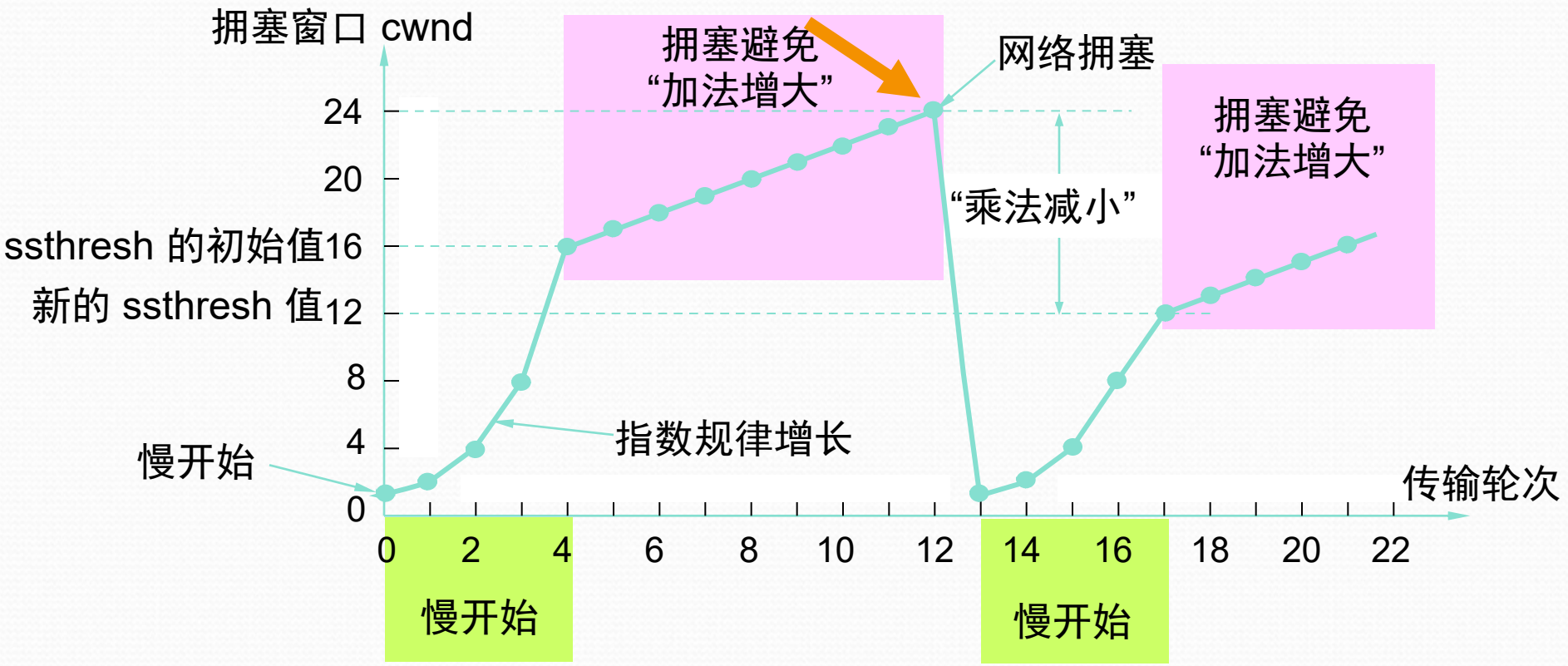

发送端每收到一个对新报文段的确认，就把发送端的拥塞窗口加 1，因此拥塞窗口 cwnd 随着传输轮次按指数规律增长。

# 慢开始和拥塞避免算法的实现举例



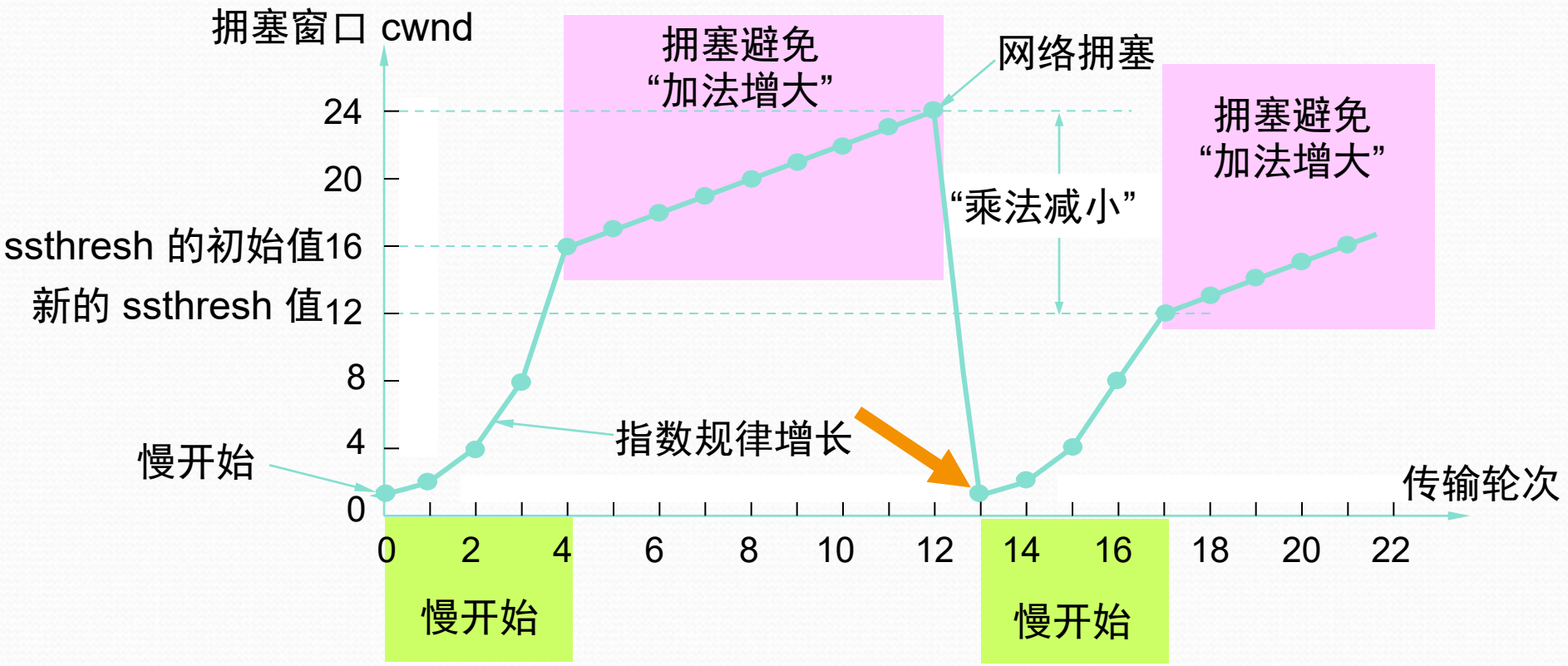

当拥塞窗口 cwnd 增长到慢开始门限值 ssthresh 时（即当 cwnd = 16 时），就改为执行拥塞避免算法，拥塞窗口按线性规律增长。

# 慢开始和拥塞避免算法的实现举例



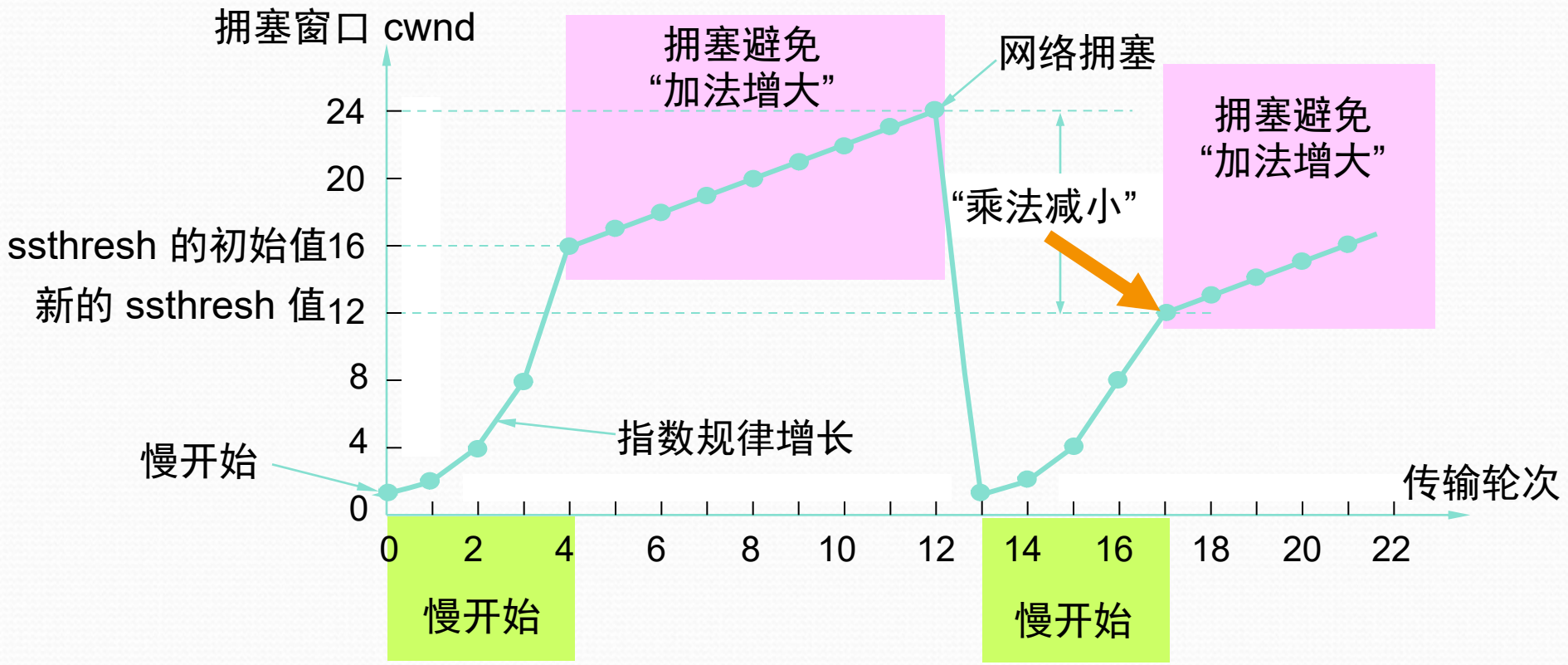

假定拥塞窗口的数值增长到 24 时，网络出现超时，表明网络拥塞了。

# 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 MSS 的大小。

# 乘法减小 (multiplicative decrease)

- “乘法减小“是指不论在慢开始阶段还是拥塞避免阶段，只要出现一次超时（即出现一次网络拥塞），就把慢开始门限值 ssthresh 设置为当前的拥塞窗口值乘以 0.5。
- 当网络频繁出现拥塞时，ssthresh 值就下降得很快，以大大减少注入到网络中的分组数。

# 加法增大 (additive increase)

- “加法增大”是指执行拥塞避免算法后，在收到对所有报文段的确认后（即经过一个往返时间），就把拥塞窗口 cwnd增加一个 MSS 大小，使拥塞窗口缓慢增大，以防止网络过早出现拥塞。

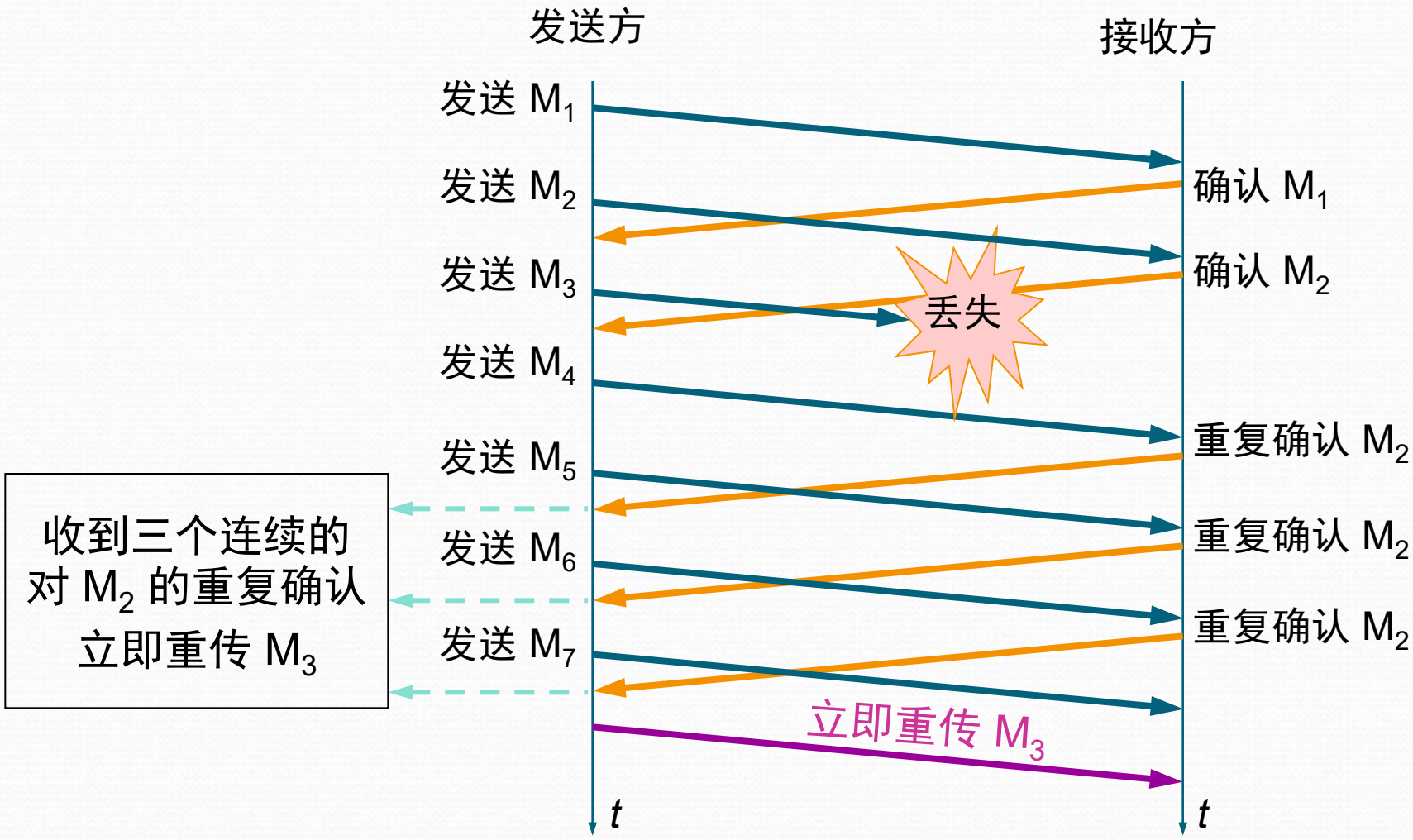
# 必须强调指出

- “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。

# 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。
- 不难看出，快重传并非取消重传计时器，而是在某些情况下可更早地重传丢失的报文段。

# 快重传举例

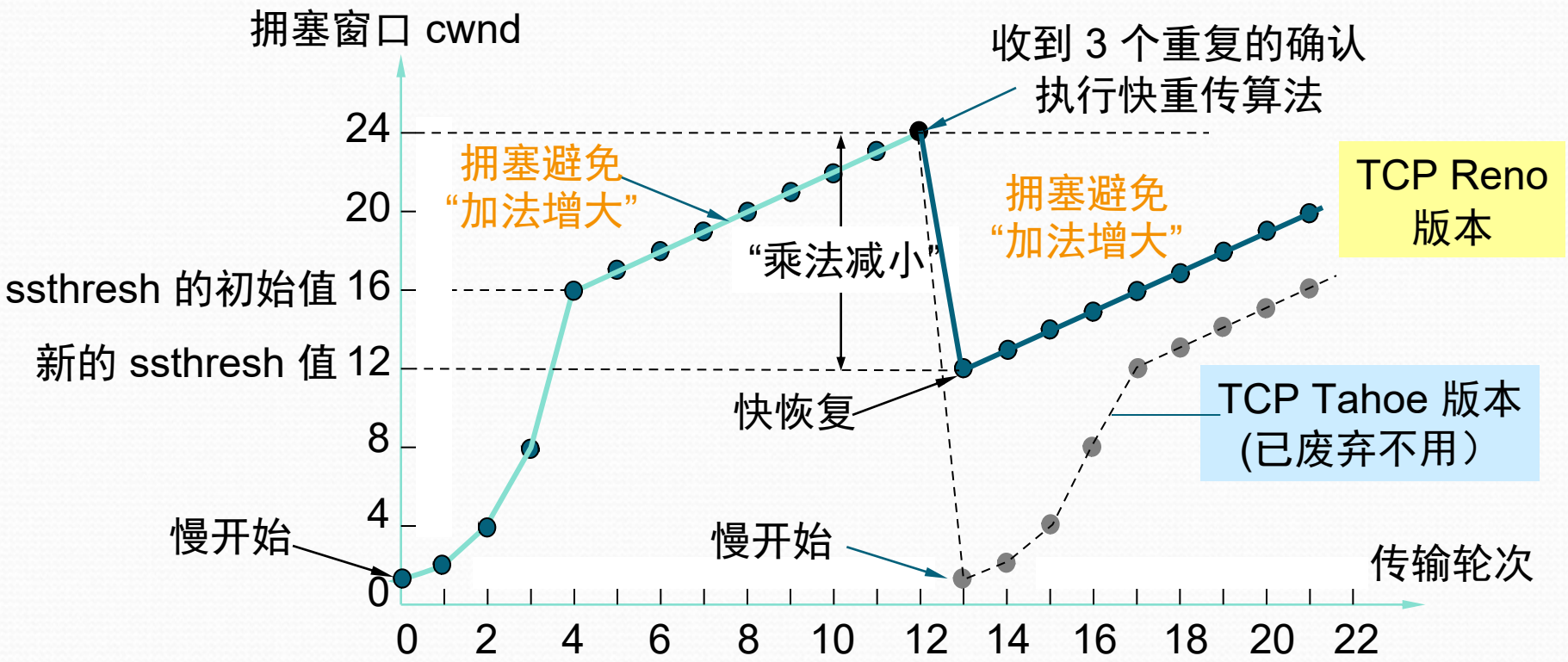

发送方
接收方
发送 $M_1$
发送 $M_2$
发送 $M_3$
发送 $M_4$
发送 $M_5$
发送 $M_6$
发送 $M_7$
确认 $M_1$
确认 $M_2$
丢失
重复确认 $M_2$
重复确认 $M_2$
重复确认 $M_2$
立即重传 $M_3$
t
t
收到三个连续的
对 $M_2$ 的重复确认
立即重传 $M_3$

# 快恢复算法

(1) 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。但接下去不执行慢开始算法。

(2)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，即拥塞窗口 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。

# 从连续收到三个重复的确认转入拥塞避免



# 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：

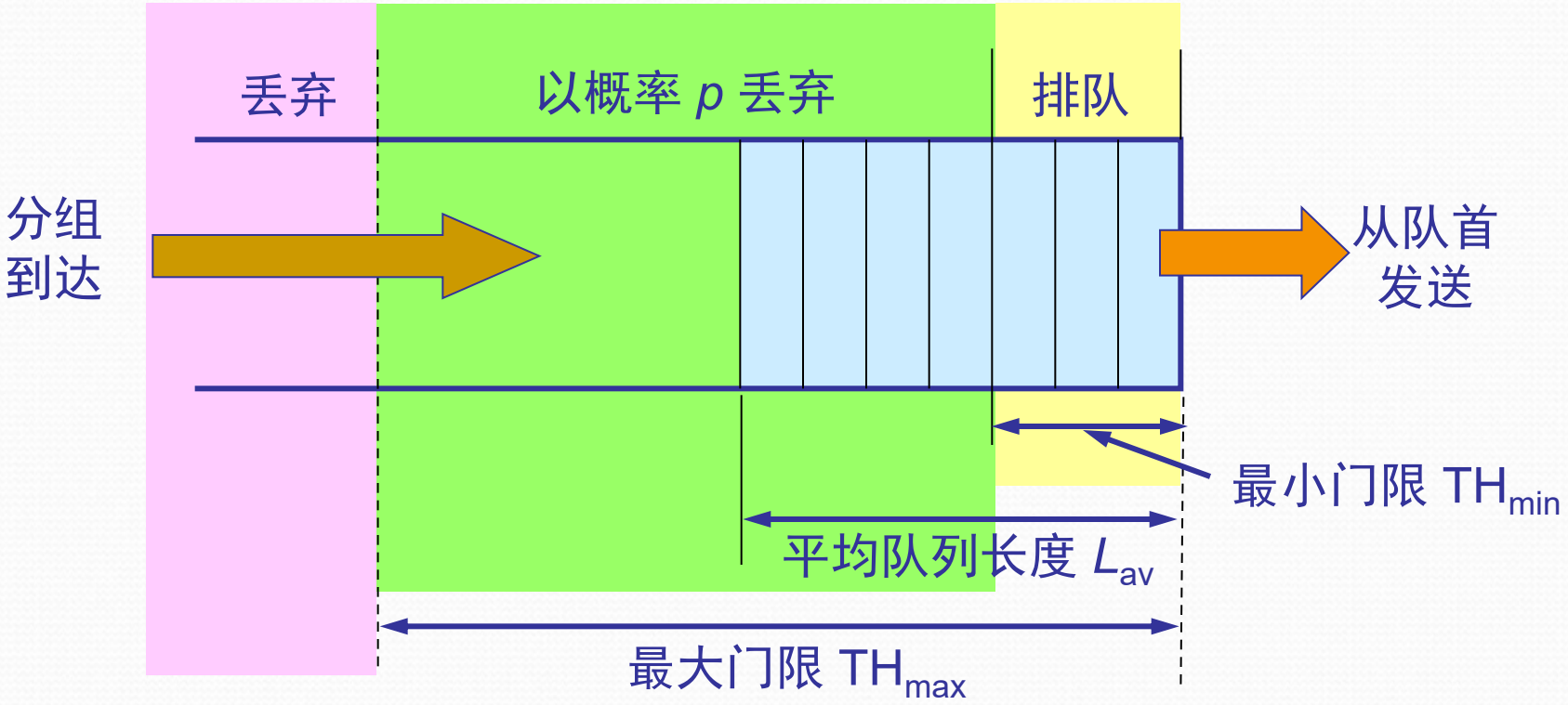
发送窗口的上限值 = Min [rwnd, cwnd]　　(5-8)

- 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。

# 随机早期检测 RED (Random Early Detection)

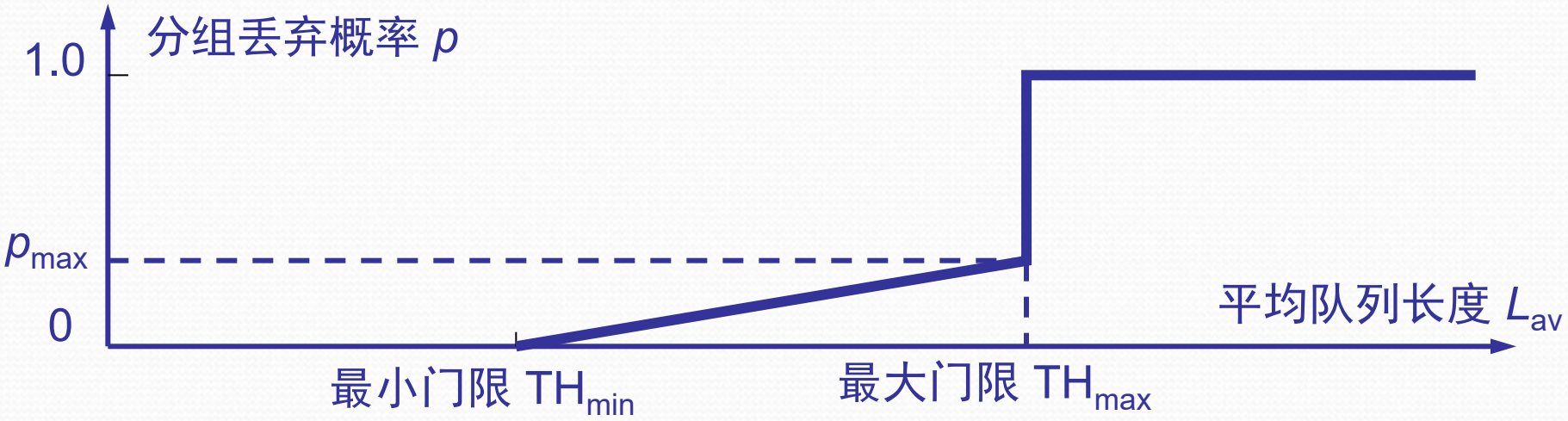
- 使路由器的队列维持两个参数，即队列长度最小门限 $TH_{min}$ 和最大门限 $TH_{max}$。
- RED 对每一个到达的数据报都先计算平均队列长度 $L_{AV}$。
- 若平均队列长度小于最小门限 $TH_{min}$，则将新到达的数据报放入队列进行排队。
- 若平均队列长度超过最大门限 $TH_{max}$，则将新到达的数据报丢弃。
- 若平均队列长度在最小门限 $TH_{min}$ 和最大门限$TH_{max}$ 之间，则按照某一概率 $p$ 将新到达的数据报丢弃。

# RED 将路由器的到达队列划分成为三个区域



# 丢弃概率 $p$ 与 $TH_{min}$ 和 $Th_{max}$ 的关系

- 当 $L_{AV} < Th_{min}$ 时，丢弃概率 $p = 0$。
- 当 $L_{AV} > Th_{max}$ 时，丢弃概率 $p = 1$。
- 当 $TH_{min} < L_{AV} < TH_{max}$ 时，$0 < p < 1$。
  例如，按线性规律变化，从 0 变到 $p_{max}$。



# TCP 的运输连接管理

## 1. 运输连接的三个阶段

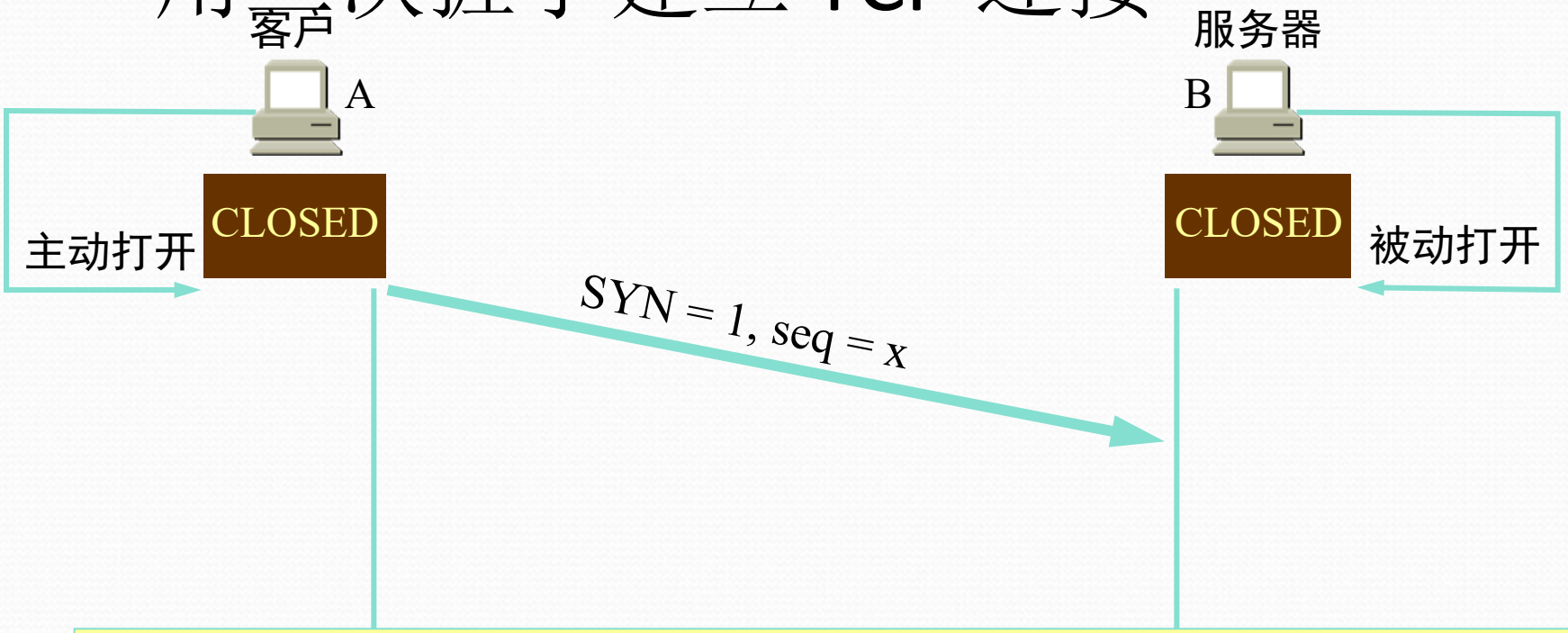
- 运输连接就有三个阶段，即：连接建立、数据传送和连接释放。运输连接的管理就是使运输连接的建立和释放都能正常地进行。
- 连接建立过程中要解决以下三个问题：
  - 要使每一方能够确知对方的存在。
  - 要允许双方协商一些参数（如最大报文段长度，最大窗口大小，服务质量等）。
  - 能够对运输实体资源（如缓存大小，连接表中的项目等）进行分配。

# 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)。

# TCP 的连接建立

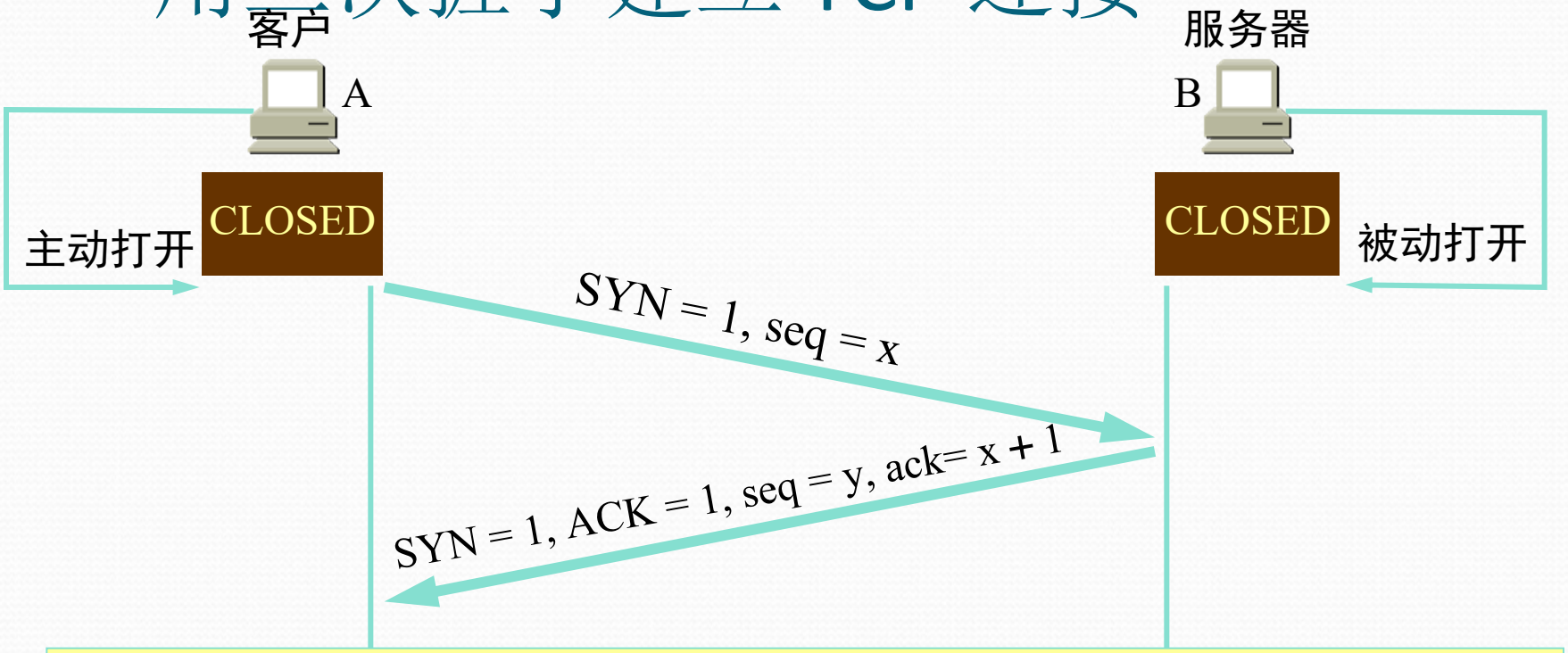
## 用三次握手建立 TCP 连接




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，表明本报文段要发送的数据的第一个字节的序号是 x。

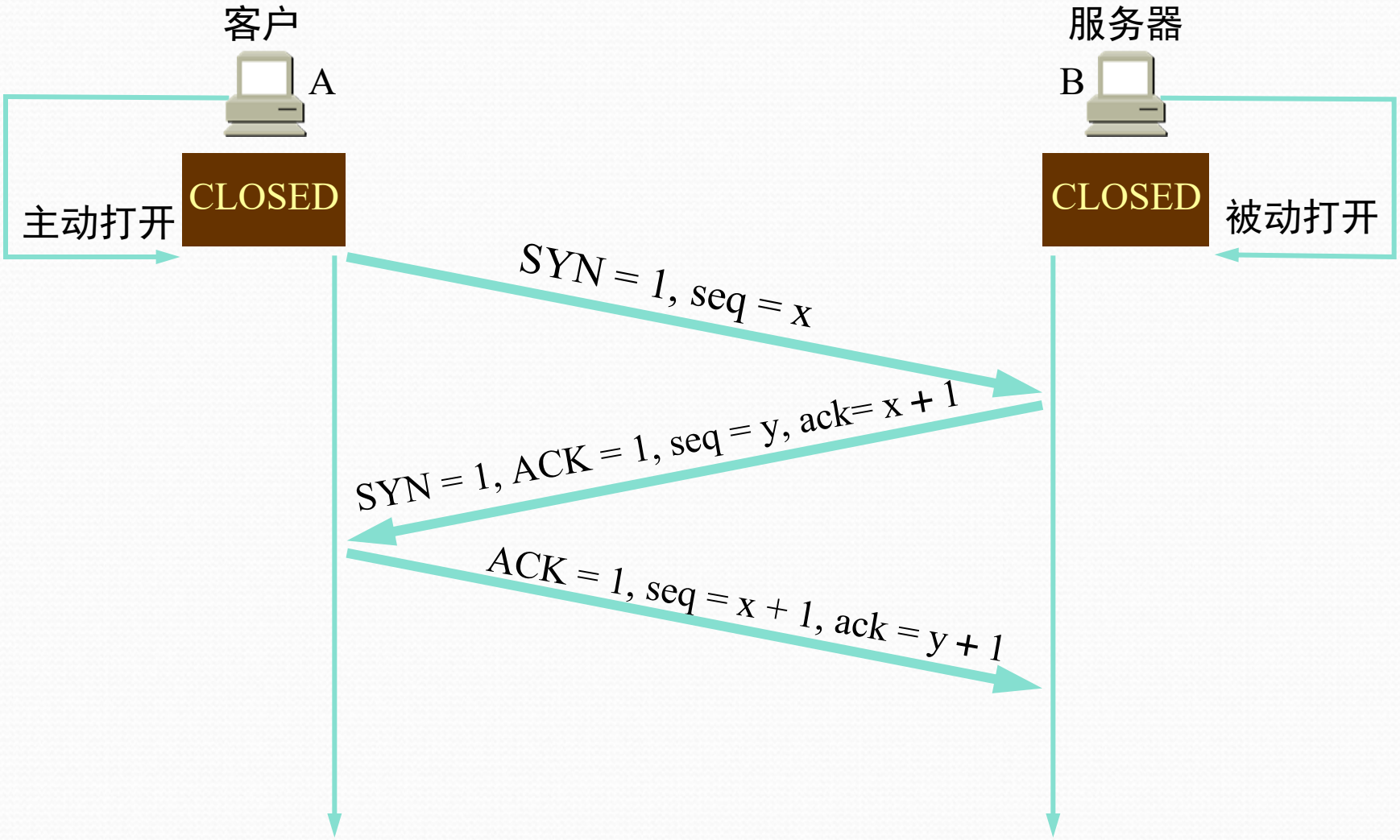
# TCP 的连接建立

## 用三次握手建立 TCP 连接

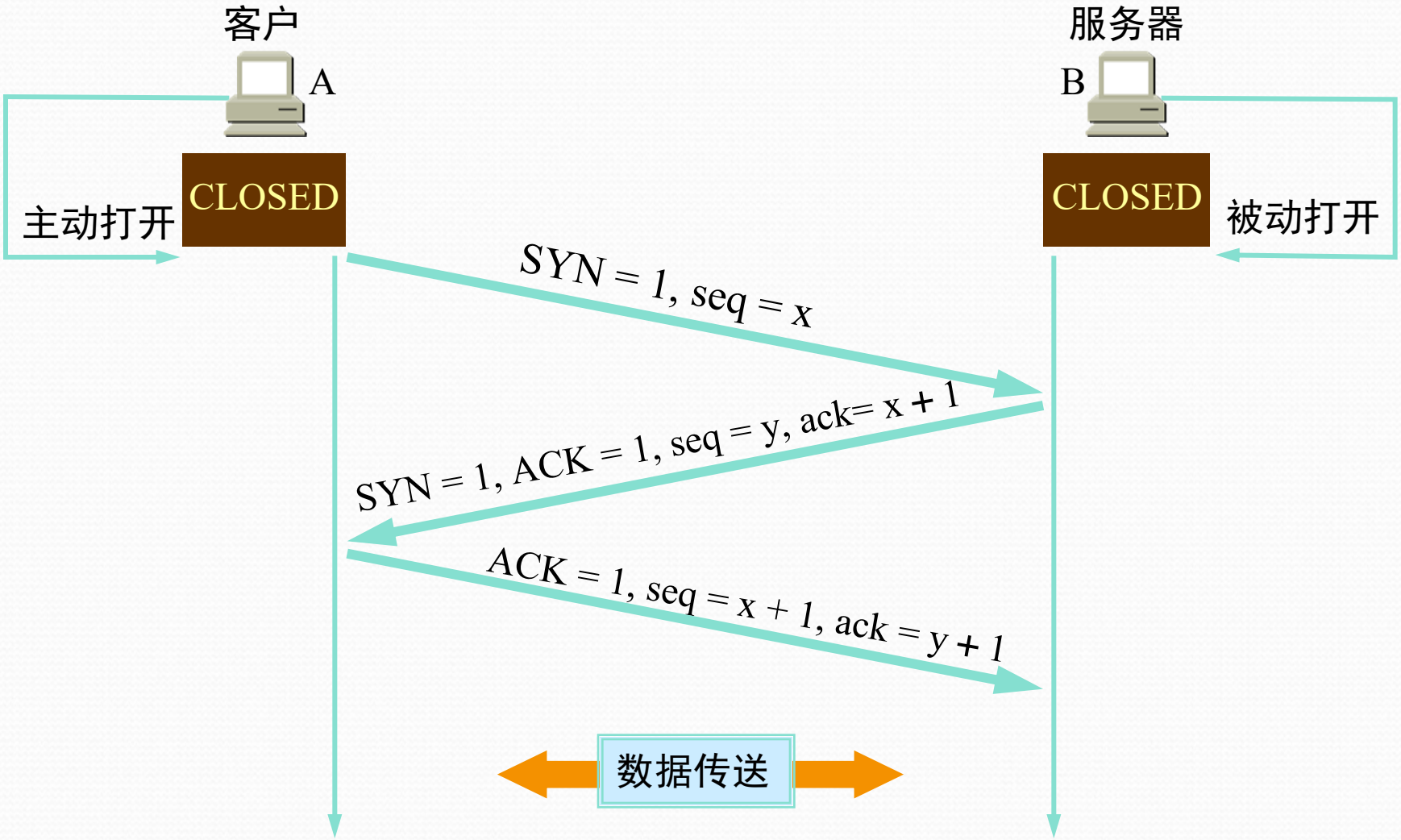



- B 的 TCP 收到连接请求报文段后，如同意，则发回确认。
- B 在确认报文段中应使 SYN = 1，使 ACK = 1，其确认号ack = x + 1，自己选择的序号 seq = y。

- 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。

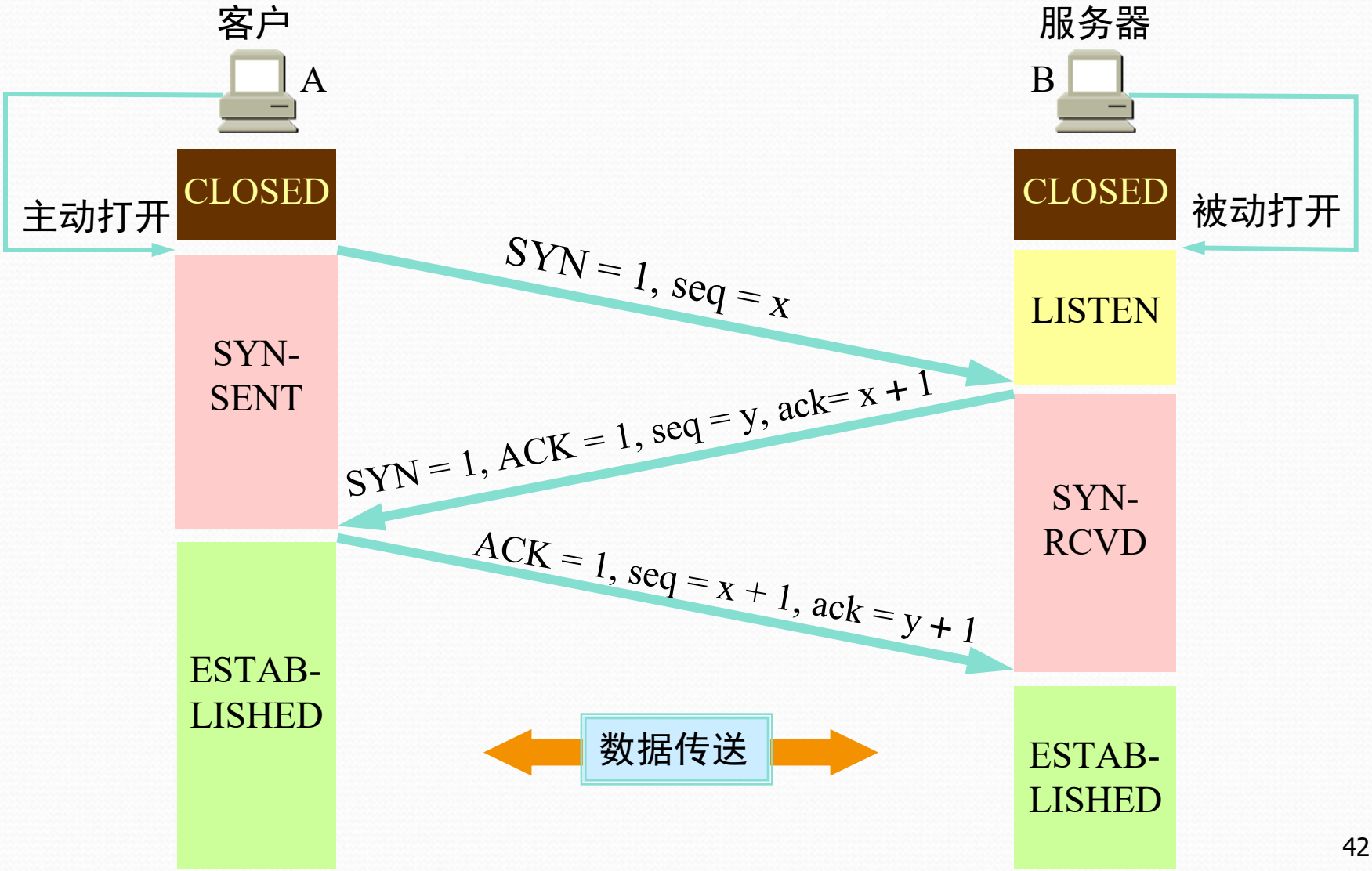


- B 的 TCP 收到主机 A 的确认后，也通知其上层应用进程：TCP 连接已经建立。

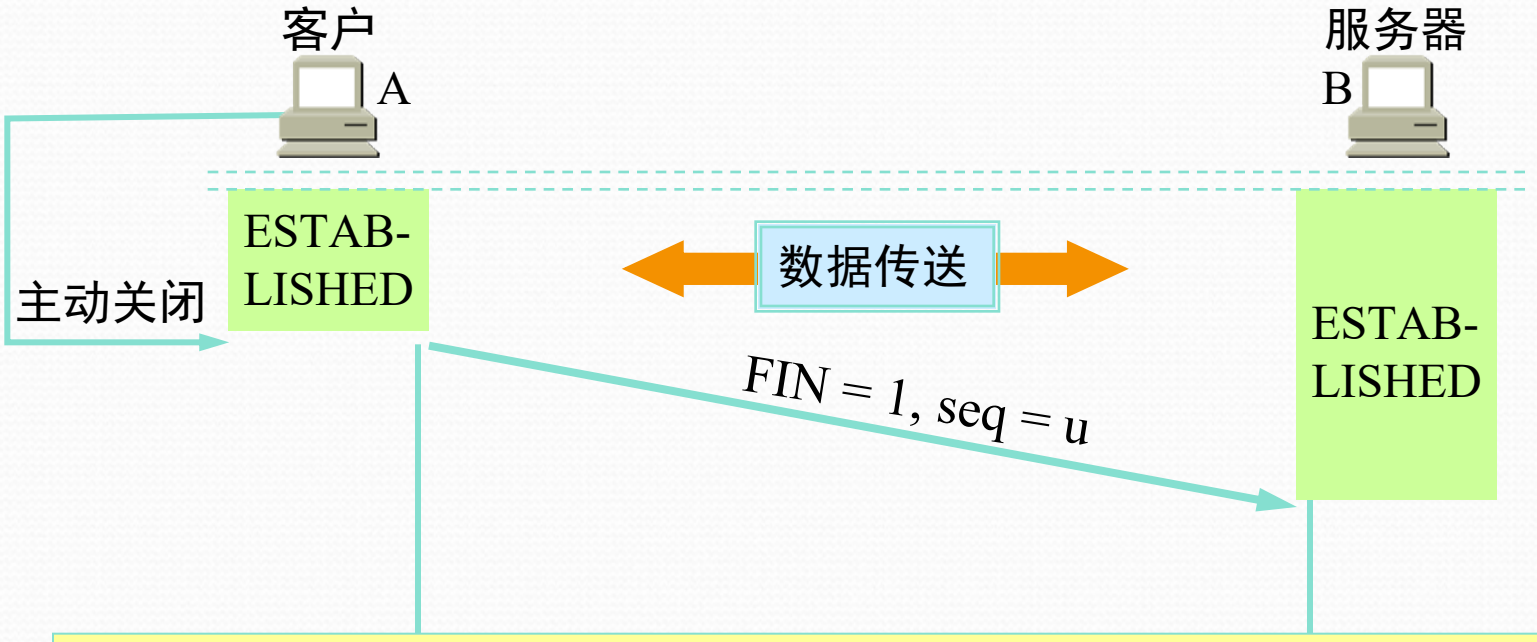


# TCP 的连接建立

## 用三次握手建立 TCP 连接的各状态



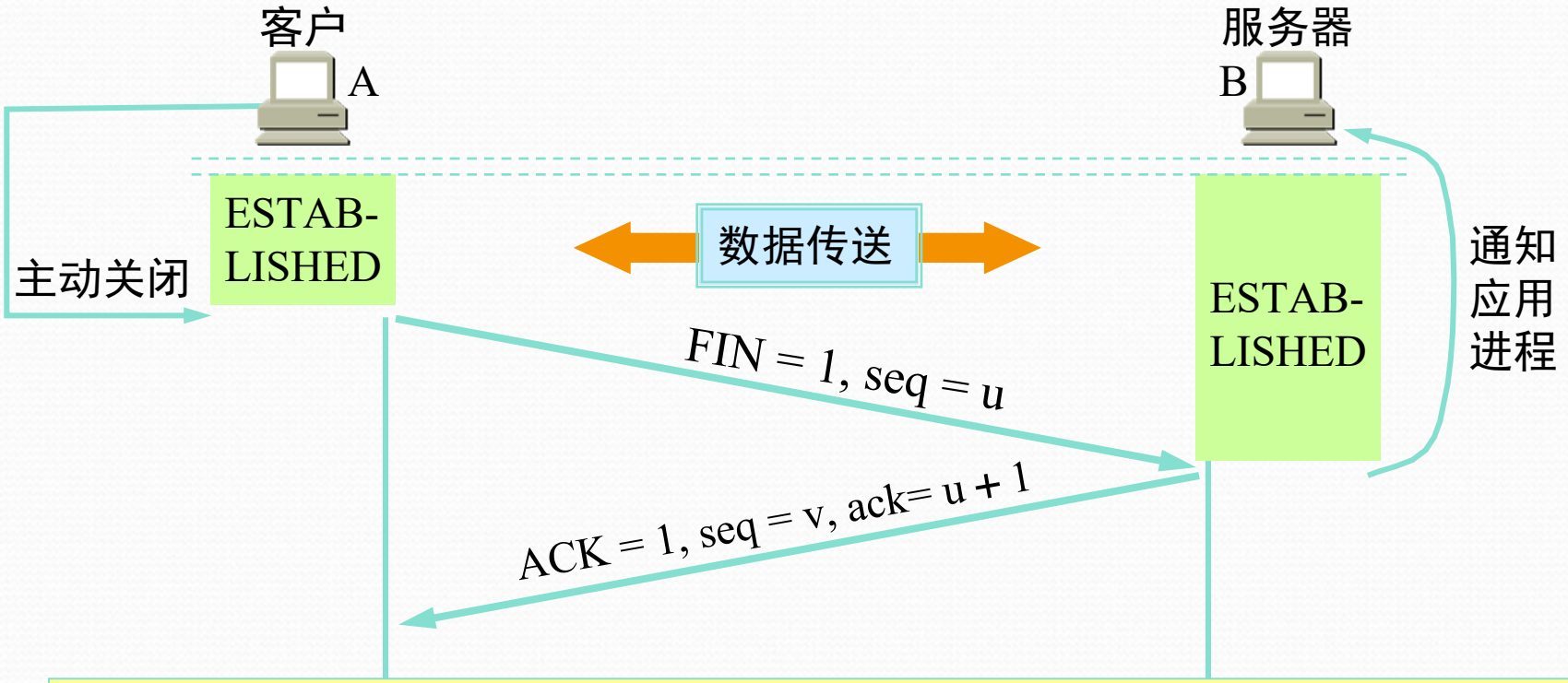
# TCP 的连接释放




- 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。

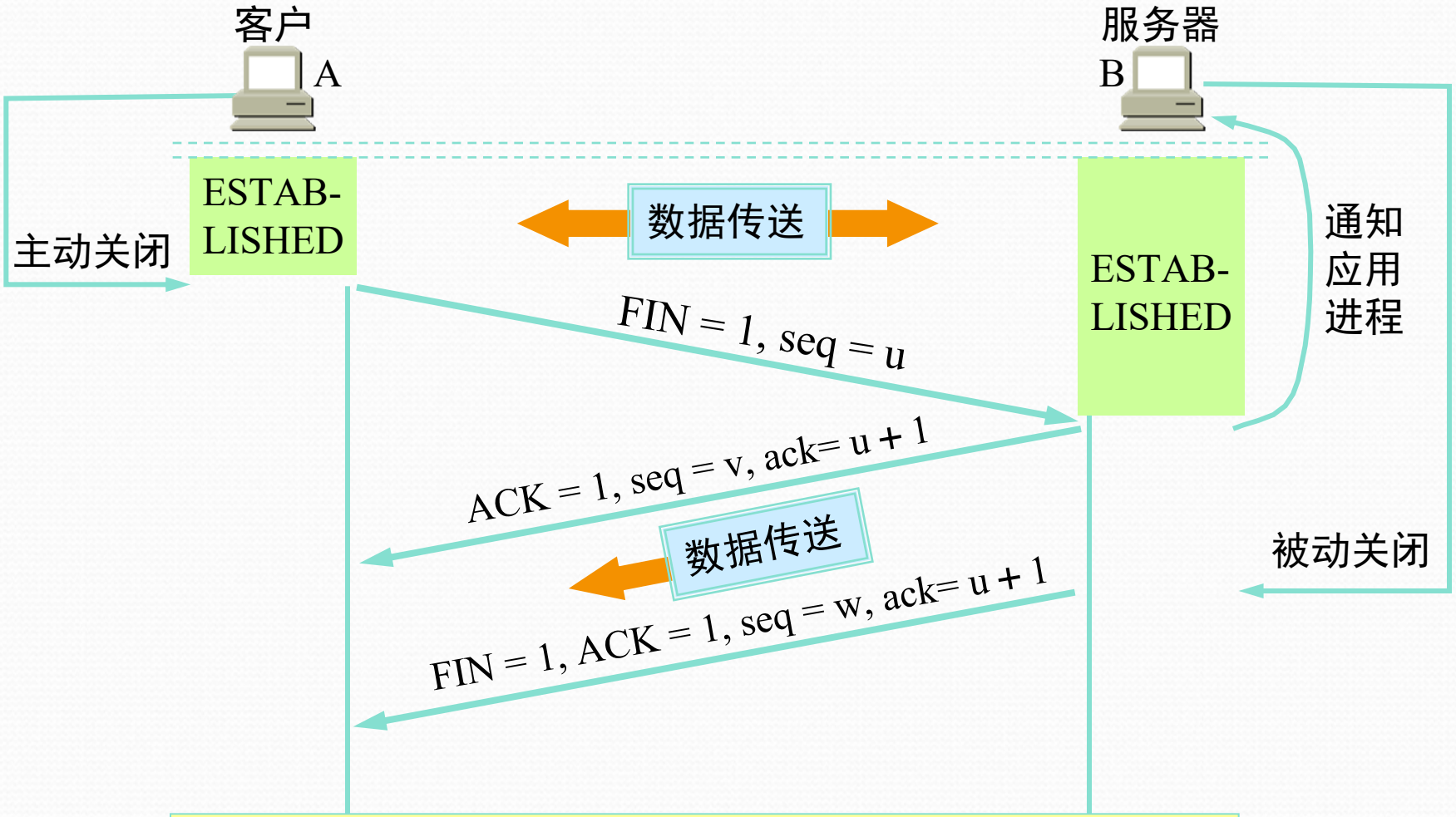
CLOSED

# TCP 的连接释放



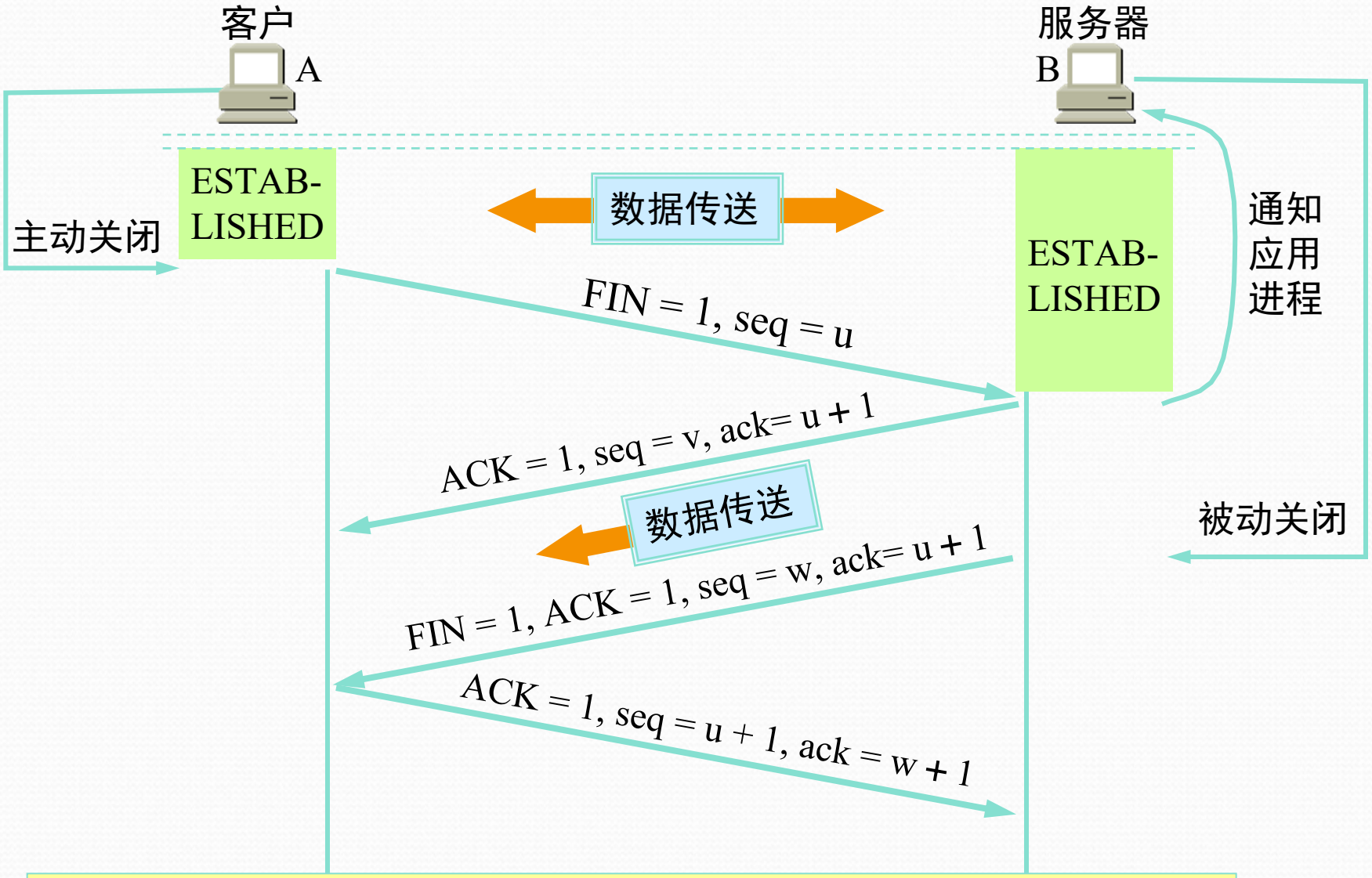

- B 发出确认，确认号 ack = u + 1，而这个报文段自己的序号 seq = v。
- TCP 服务器进程通知高层应用进程。
- 从 A 到 B 这个方向的连接就释放了，TCP 连接处于半关闭状态。B 若发送数据，A 仍要接收。

# TCP 的连接释放



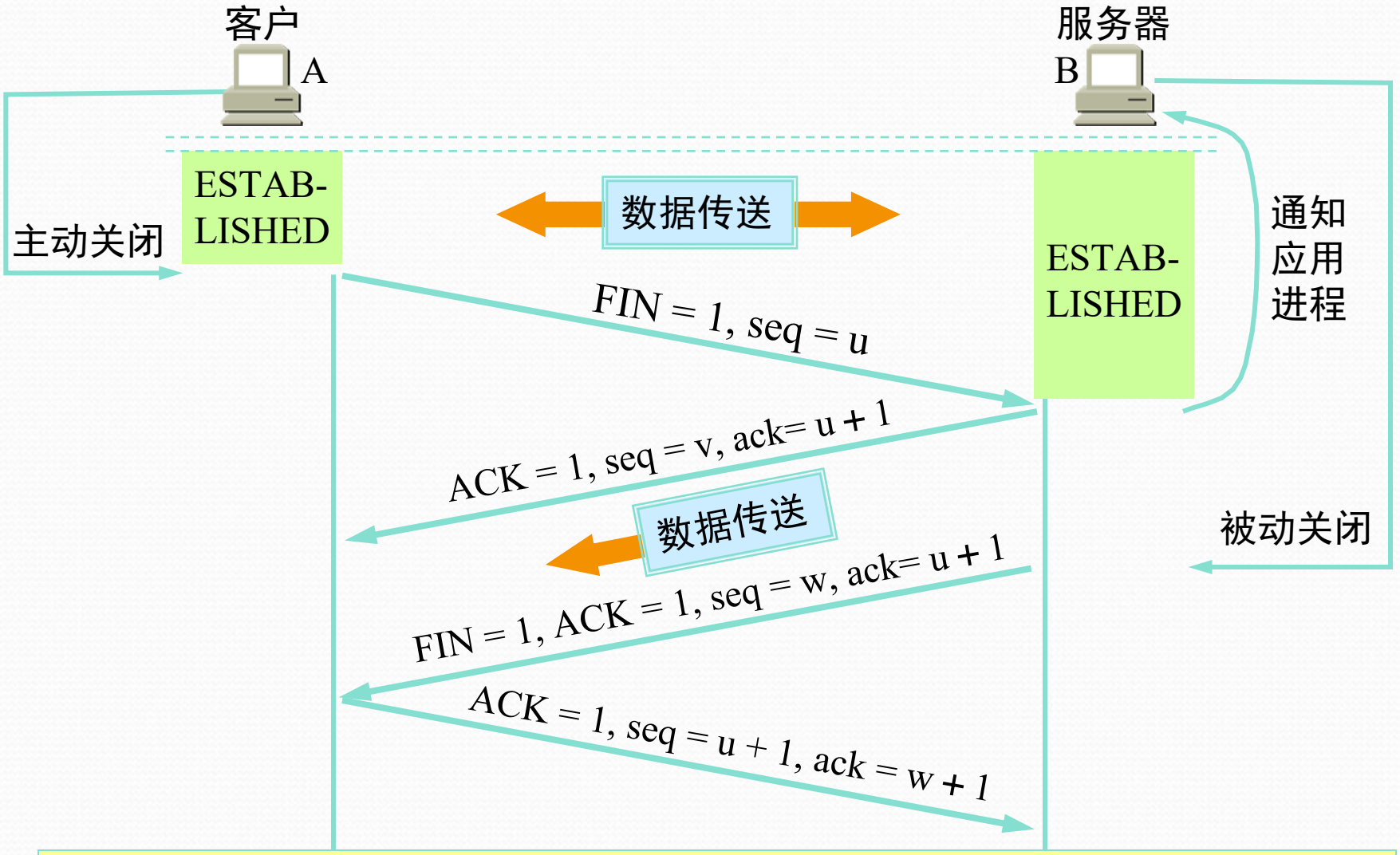

- 若 B 已经没有要向 A 发送的数据，其应用进程就通知 TCP 释放连接。

# TCP 的连接释放



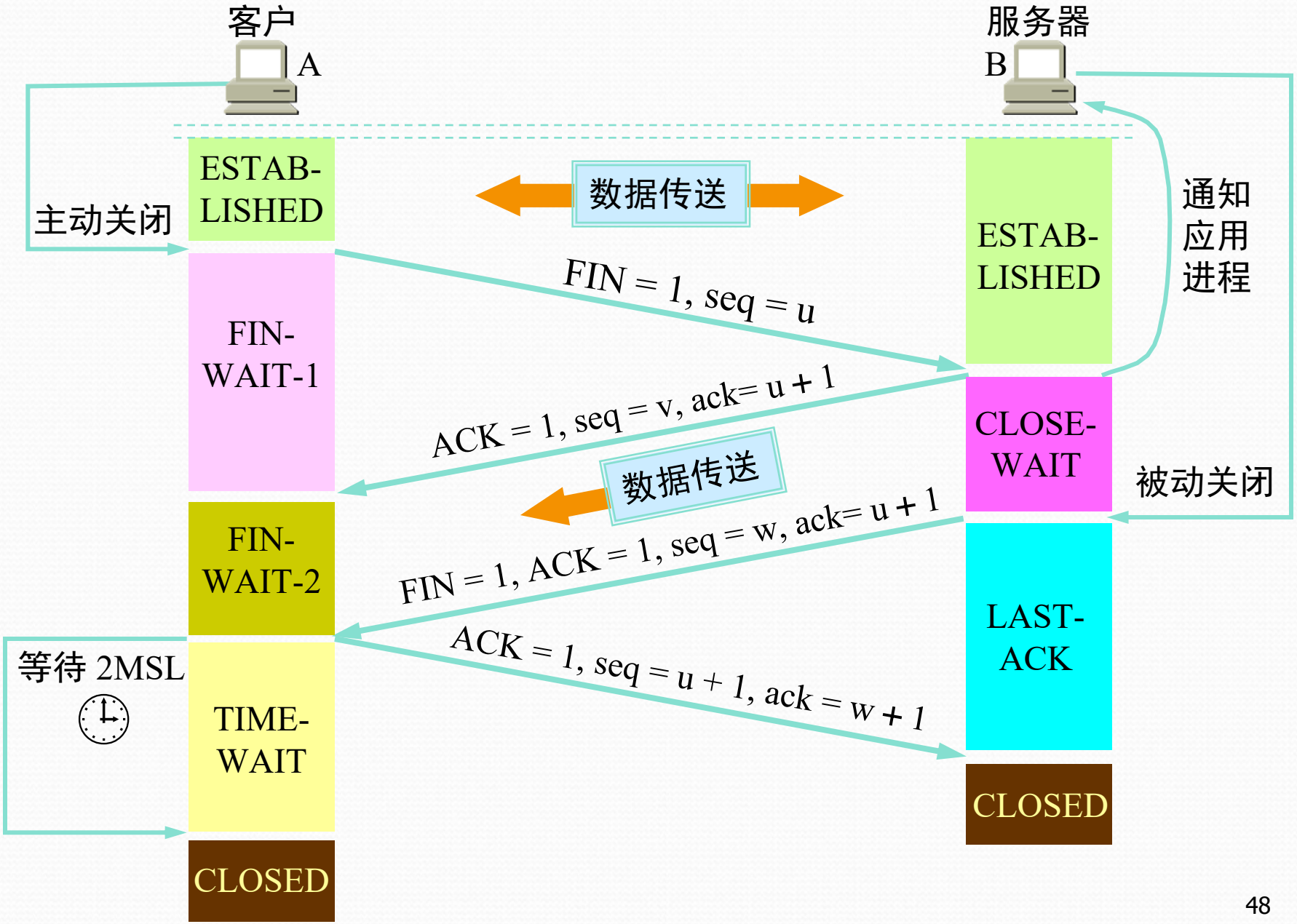

- A 收到连接释放报文段后，必须发出确认。

# TCP 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。



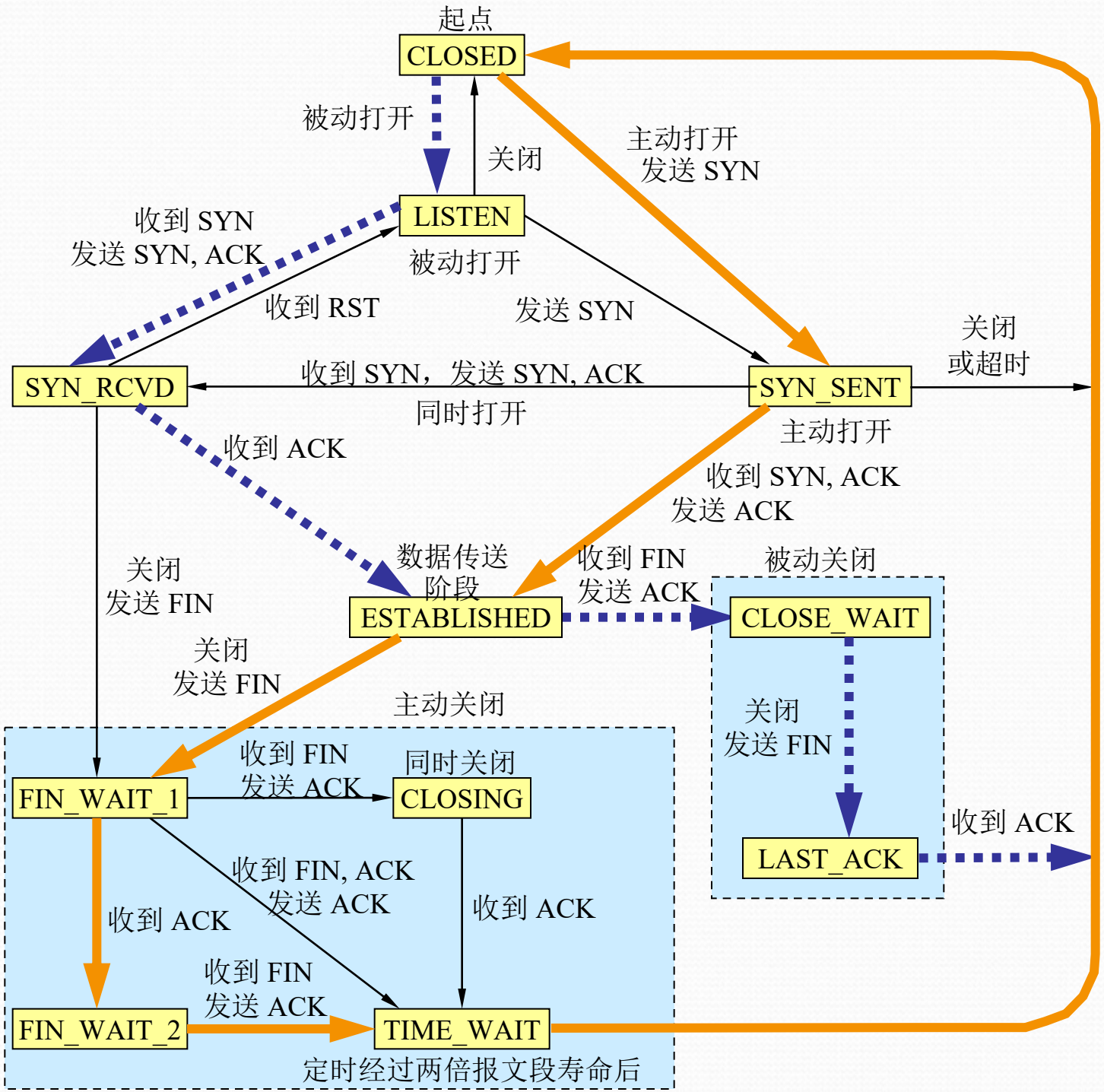
# A 必须等待 2MSL 的时间

- 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。这样就可以使下一个新的连接中不会出现这种旧的连接请求报文段。

# TCP 的有限状态机

- TCP 有限状态机的图中每一个方框都是 TCP 可能具有的状态。
- 每个方框中的大写英文字符串是 TCP 标准所使用的 TCP 连接状态名。状态之间的箭头表示可能发生的状态变迁。
- 箭头旁边的字，表明引起这种变迁的原因，或表明发生状态变迁后又出现什么动作。
- 图中有三种不同的箭头。
  - 粗实线箭头表示对客户进程的正常变迁。
  - 粗虚线箭头表示对服务器进程的正常变迁。
  - 另一种细线箭头表示异常变迁。

# TCP 的有限状态机


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
LAST_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□□□

TCP的拥塞控制

拥塞控制的原理，拥塞控制方法，随机早期检测

TCP的运输连接管理

TCP的连接建立，连接释放，有限状态机

# 作业

- 5-37，5-46