第三章 数据链路层（二）

袁华，hyuan@scut.edu.cn
华南理工大学计算机科学与工程学院
广东省计算机网络重点实验室

本节目的

- 掌握**6**种基本的**DLL**协议
- 掌握滑动窗口技术
- 掌握**ARQ**（自动重复请求）技术
- 掌握捎带确认技术

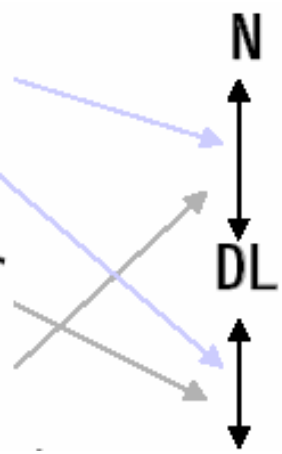
基本数据链路协议

- 由浅入深地通过六个协议的介绍，了解数据链路层的主要工作，这些工作原理和概念是网络通信的基础，其运行机理在网络层和传输层也会采用。
- 首先介绍最简单的三个协议：
  - 无限制的单工协议
  - 单工仃—等协议
  - 有噪声信道的单工协议

几点说明（1/2）

- 六个协议共同使用的数据类型、调用的函数过程定义见图**3.9**（**protocol.h**）。
- 与网络层、物理层之间的数据传送

发送：　from_network_layer
　　to_physical_layer
接收：　from_physical_layer
　　to_network_layer



几点说明（2/2）

- **Wait_for_event** 等待某个事件发生

  event：fram_arrive, cksum_err, timeout

- **Timer** 计时器的作用
  - **start_timer, stop_timer, start_ack_timer，stop_ack_timer**

帧结构：P169

- typedef struct{

  frame_kind kind;

  seq_nr seq;

  seq_nr ack;

  packet info;

  }frame;

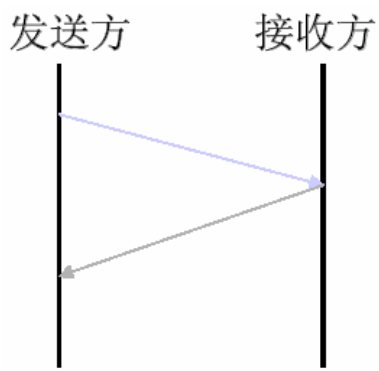
无限制的单工协议(协议1) P171

- 单向传送，不考虑流量，接收方允许无限量地接收。
- 收发双方的网络层都处于就绪状态
- 假设DLL之间的信道永远不会损坏或者丢失帧
- “乌托邦”

```
typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                      /* buffer for an outbound frame */
  packet buffer;                /* buffer for an outbound packet */

  while (true) {
      from_network_layer(&buffer);                    /* go get something to send */
      s.info = buffer;          /* copy it into s for transmission */
      to_physical_layer(&s);    /* send it on its way */
  }                             /* Tomorrow, and tomorrow, and tomorrow,
                                   Creeps in this petty pace from day to day
                                   To the last syllable of recorded time.
                                       - Macbeth, V, v */
}


void receiver1(void)
{
  frame r;
  event_type event;             /* filled in by wait, but not used here */

  while (true) {
      wait_for_event(&event);   /* only possibility is frame_arrival */
      from_physical_layer(&r);  /* go get the inbound frame */
      to_network_layer(&r.info);/* pass the data to the network layer */
  }
}
```

单工的停—等协议（协议2）P172

- 解决如何避免收方被涌入的数据淹没，即取消“接收方允许无限量接收”的假设
- 解决方法：收方回发一个哑帧，接收方收到哑帧，表明收方允许接收数据，此时再次发送下一帧数据。
- 实际上是半双工协议



```
void sender2(void)
{
  frame s;                              /* buffer for an outbound frame */
  packet buffer;                        /* buffer for an outbound packet */
  event_type event;                     /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);      /* go get something to send */
      s.info = buffer;                  /* copy it into s for transmission */
      to_physical_layer(&s);            /* bye bye little frame */
                                        /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                           /* buffers for frames */
  event_type event;                     /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event);           /* only possibility is frame_arrival */
      from_physical_layer(&r);          /* go get the inbound frame */
      to_network_layer(&r.info);        /* pass the data to the network layer */
                                        /* send a dummy frame to awaken sender */
  }
}
```

有噪声信道的单工协议（协议3）P174

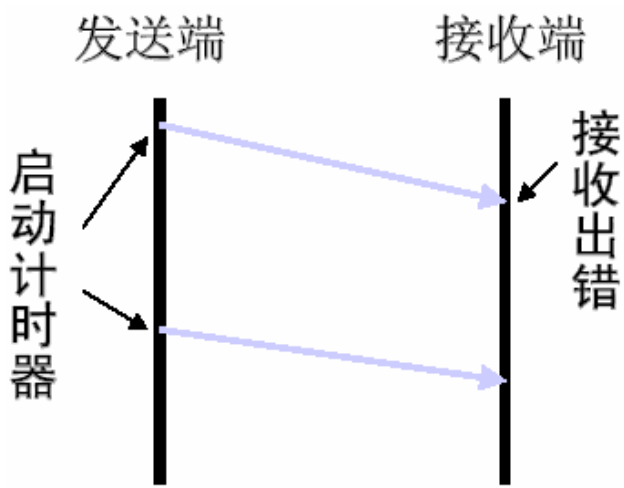
- 有噪声就会产生差错，有差错就会引起以下一些问题：
  - 接收方检测到错误帧，如何通知发送方？如何恢复正确帧？
  - 数据帧或确认帧在途中丢失将如何解决？
  - 有可能收到重复帧，如何解决？

解决以上问题的方法（1/2）

- 确认；重发；计时器；帧序号。
- 确认帧
  - 接收端进行差错检查后，发确认帧送回发送端，告诉接收是否正确。
  - 为了减少网络内的流量，只在接收无差错时才发确认帧，出错时不发确认帧。发送端等候计时器给定的时间后仍未收到确认帧，则知道接收出错，此时发送端重发原来的帧。

主动确认重发P175

- 接收方无误，回发确认帧。
- **ARQ:automatic repeat request P175**
- **PAR:positive acknowl-edgement with retransmission**



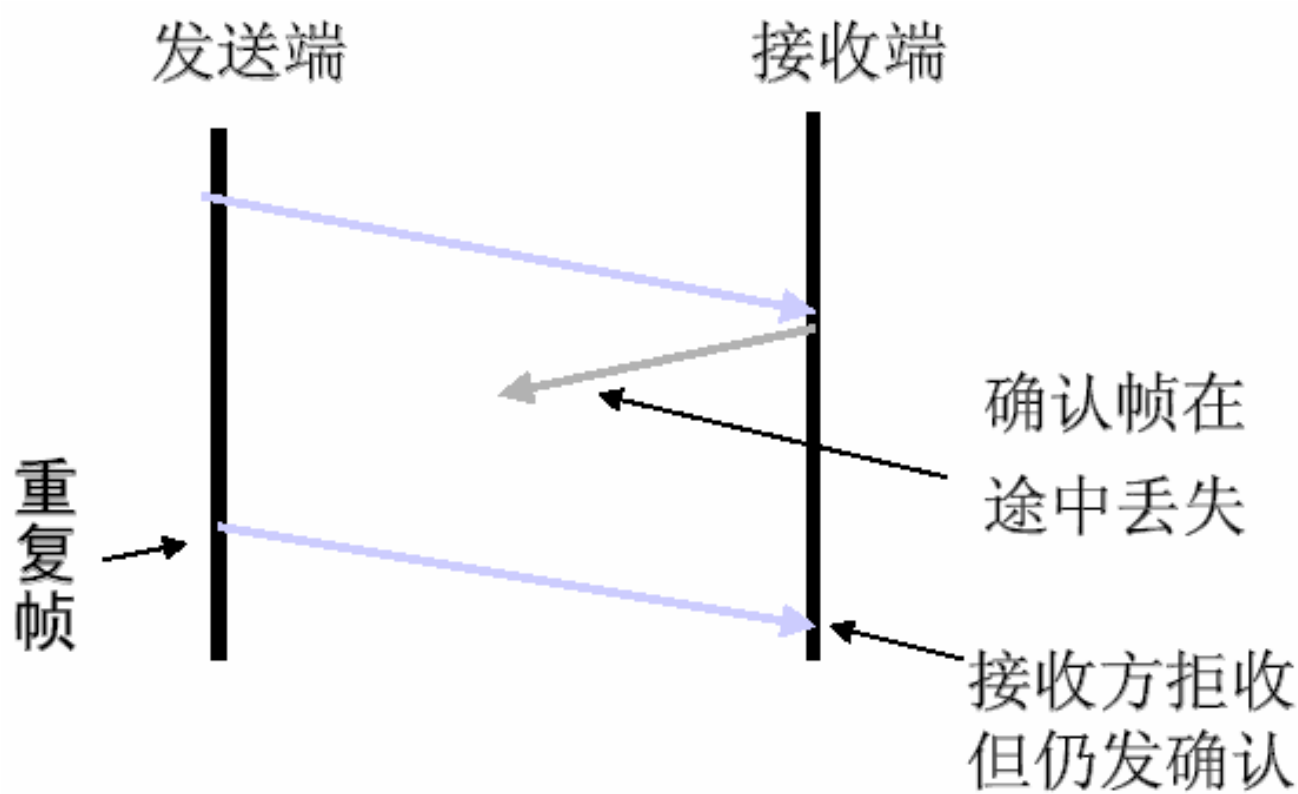
解决以上问题的方法（2/2）

- 重发
  - 网络中采用检错码，无法纠正错误，由重发原来帧的方式来恢复正确的帧。
- 计时器
  - 控制何时重发。
- 帧序号
  - 防止重发时接收端收到重复的帧，序号还用于接收时排序。
  - 保证送给网络层的都是按序无重复的分组

计时器的作用

- 通过发送方计时器（超时）解决。
  - 超时（**TimeOut**）：在传输过程中，所发送的帧丢失，接收方根本没有收到，不可能发送确认帧（包括否定性确认），而发送方正在等待接收方的确认帧，当然也不可能等到接收方的确认。所以，发送方一旦发送一帧，就启动一计时器，在规定的时间内，一般都应收到确认，如收不到确认，则在计时器溢出时，再重发此帧。
  - 确认超时：捎带确认超时，单发确认帧

接收端为什么会收到重复帧？

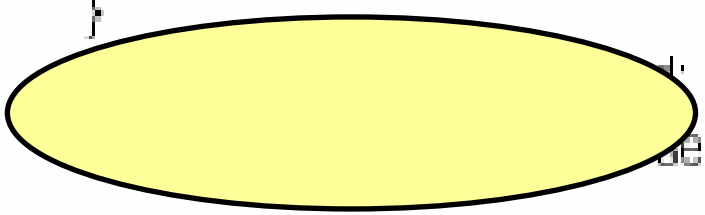


重复帧

- 重发机制也包括当接收方发送的确认帧丢失而导致发送方的重发
- 由于接收方确认帧的丢失，导致发送方多次发送同一帧，接收方也将多次收到同一帧，要能区别是否是同一帧，则每帧都应有一个帧的编号
- 支持重传的肯定确认协议

```
void sender3(void)
{
  seq_nr next_frame_to_send;       /* seq number of next outgoing frame */
  frame s;                         /* scratch variable */
  packet buffer;                   /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;          /* initialize outbound sequence numbers */
  from_network_layer(&buffer);     /* fetch first packet */
  while (true) {
       s.info = buffer;            /* construct a frame for transmission */
       s.seq = next_frame_to_send; /* insert sequence number in frame */
       to_physical_layer(&s);      /* send it on its way */
                                   /* if answer takes too long, time out */
       wait_for_event(&event);     /* frame_arrival, cksum_err, timeout */
       if (event == frame_arrival) {
            from_physical_layer(&s);  /* get the acknowledgement */
            if (s.ack == next_frame_to_send) {
                                   /* turn the timer off */
                 from_network_layer(&buffer);  /* get the next one to send */
                 inc(next_frame_to_send);      /* invert next_frame_to_send */
```

```
void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
       wait_for_event(&event);          /* possibilities: frame_arrival, cksum_err */
       if (event == frame_arrival) {    /* a valid frame has arrived. */
            from_physical_layer(&r);    /* go get the newly arrived frame */
            if (r.seq == frame_expected) {/* this is what we have been waiting for. */
                 to_network_layer(&r.info);    /* pass the data to the network layer */
                 inc(frame_expected); /* next time expect the other sequence nr */
            }
                                                 /* tell which frame is being acked */
                                            end acknowledgement */
       }
}
```



帧格式

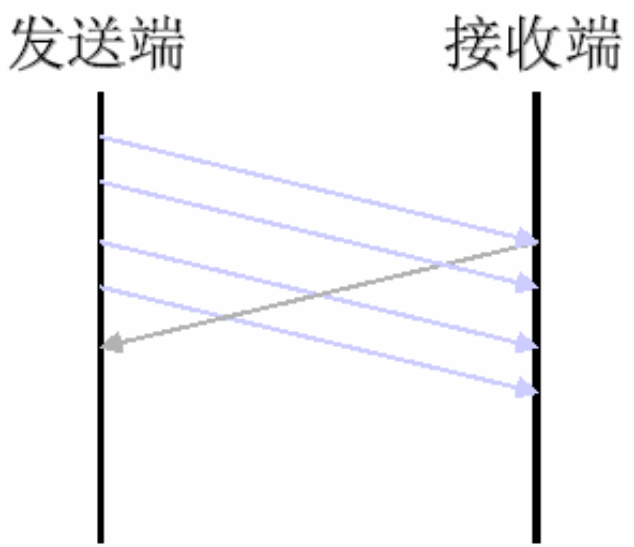
一般意义上的帧的格式

捎带确认（piggyBacking P177）

- 捎带确认的作用
  - 进一步减少确认帧。
- 捎带确认：将确认暂时延迟以便可以钩到一个外发数据帧
- 如无法“捎带”，当一个控制捎带确认的计时器超时后，单独发确认帧。

批发数据

- 仃—等协议是每收到一个确认才能发送下一帧，发送端等待时间太长，网络通信效率不高。为了提高效率，可以在等待的时间发送数据帧，这样大大减少了浪费的时间。



问题

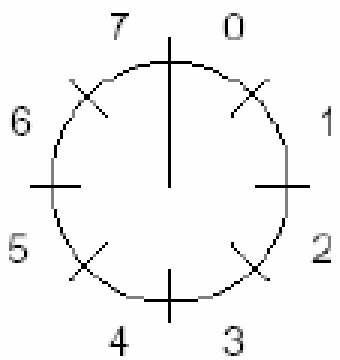
- 如果发送端可以连续发送一批数据帧，必须考虑接收端是否来得及接纳与处理这么多的帧，这里就提出了网络流量控制问题。
- 发送窗口**P178**
- 接收窗口**P178**

一位的滑动窗口协议

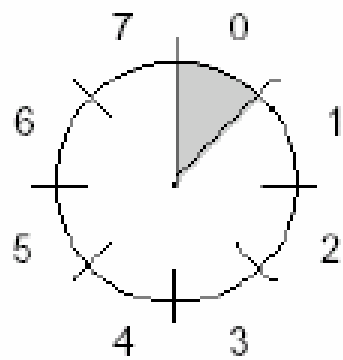
- 滑动窗口用于控制网络中的流量。
- 发送方和接收方接收能力的匹配即流量控制。
- 如接收方的处理能力低于发送方，接方收的缓冲区可能被“占满”，所以通常在接方收的缓冲区到达一定量时，应及时通知发送方，暂停发送，等候通知，这就是流量控制机制。

大小为1，有3位序列号的滑动窗口

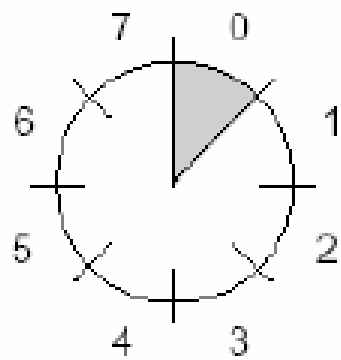
Sender



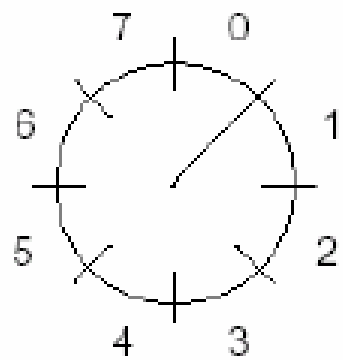
初始时



发送了第一帧

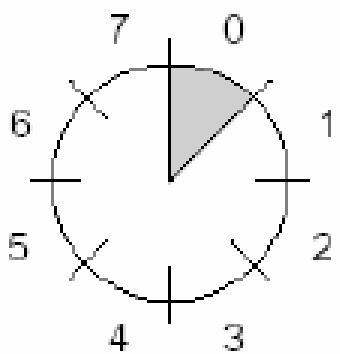


接收了第一帧

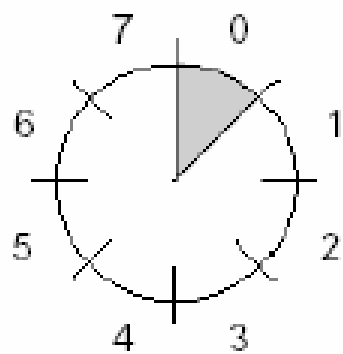


收到第一个确认帧

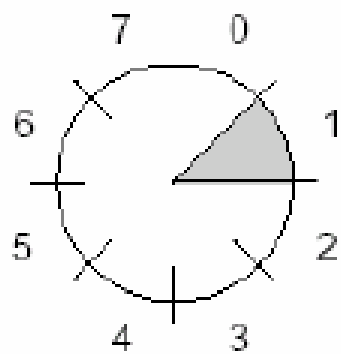
Receiver



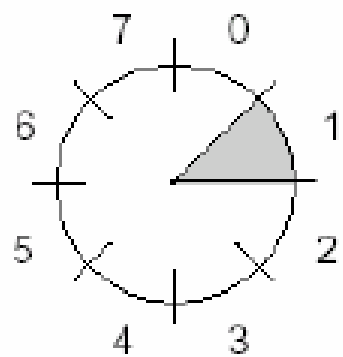
(a)



(b)



(c)



(d)

原理

- 接收方收到帧后，首先核对是否为预期帧号**(frame_expected)**，如果是的，则接收并**frame_expected+1**，即移动接收窗口。
- 发送端收到回答响应帧，核对响应帧号**next_frame_to_send**，核对无误后，从网络层取新的帧，并执行**next_frame_to_send+1**，即移动发送窗口。如核对帧号不正确，则不移动窗口

```
void protocol4 (void)
{
  seq_nr next_frame_to_send;
  seq_nr frame_expected;
  frame r, s;
  packet buffer;
  event_type event;

  next_frame_to_send = 0;
  frame_expected = 0;
  from_network_layer(&buffer);
  s.info = buffer;
  s.seq = next_frame_to_send;
  s.ack = 1 – frame_expected;
  to_physical_layer(&s);
  start_timer(s.seq);
```

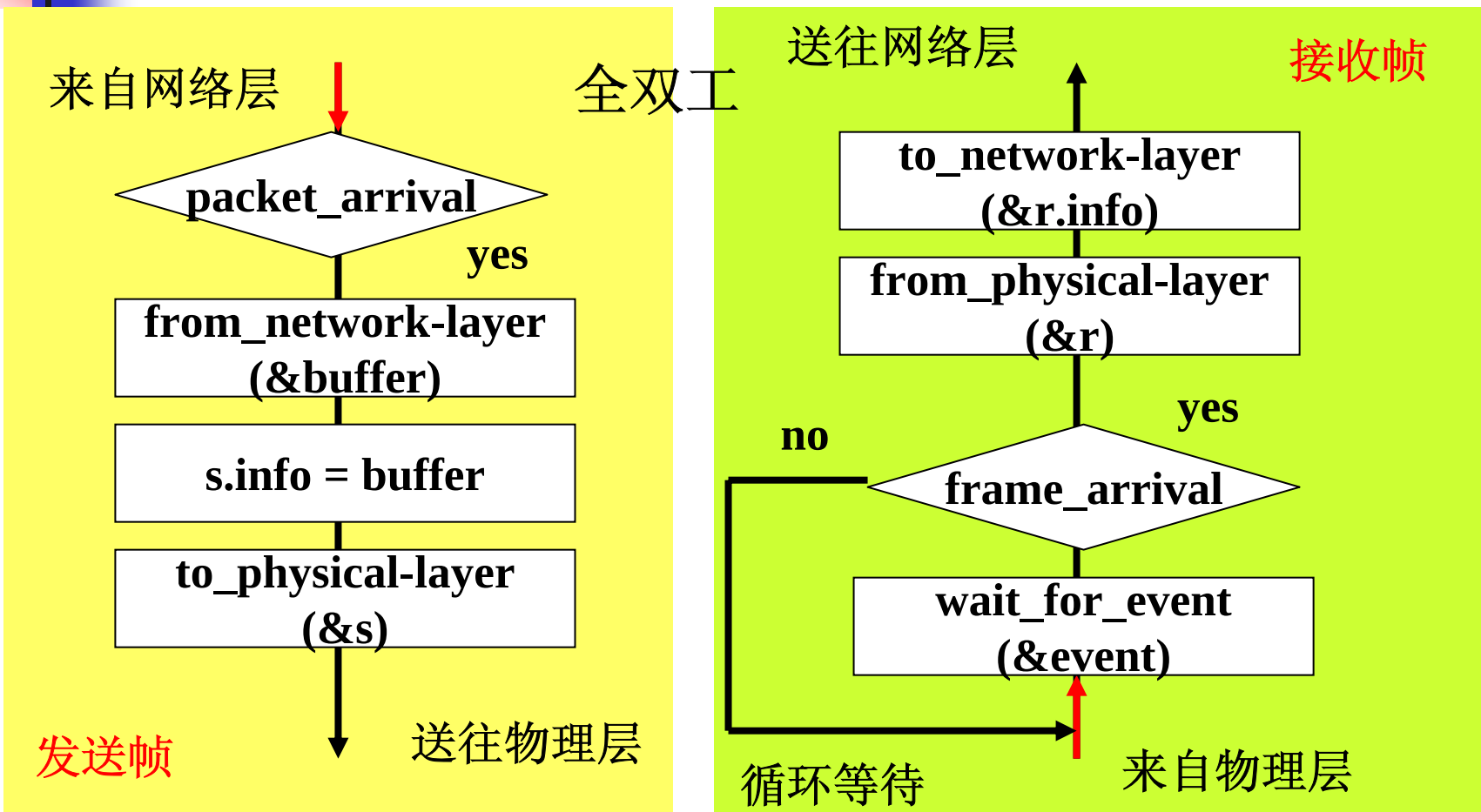
```
while (true) {
    wait_for_event(&event);
    if (event == frame_arrival) {
        from_physical_layer(&r);
        if (r.seq == frame_expected) {
            to_network_layer(&r.info);
            inc(frame_expected);
        }
        if (r.ack == next_frame_to_send) {
            stop_timer(r.ack);
            from_network_layer(&buffer);
            inc(next_frame_to_send);
        }
    }
    s.info = buffer;
    s.seq = next_frame_to_send;
    s.ack = 1 – frame_expected;
    to_physical_layer(&s);
    start_timer(s.seq);
}
```

捎带确认

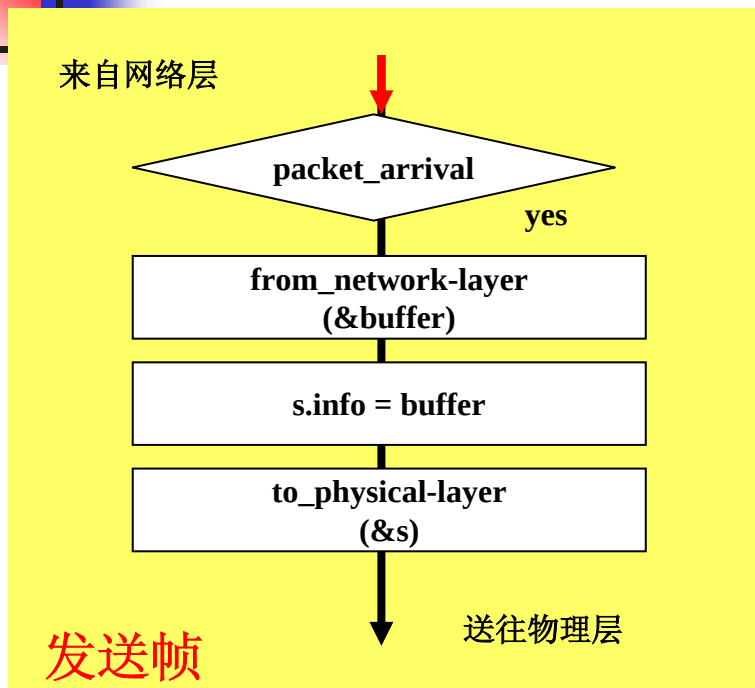
滑动窗口协议

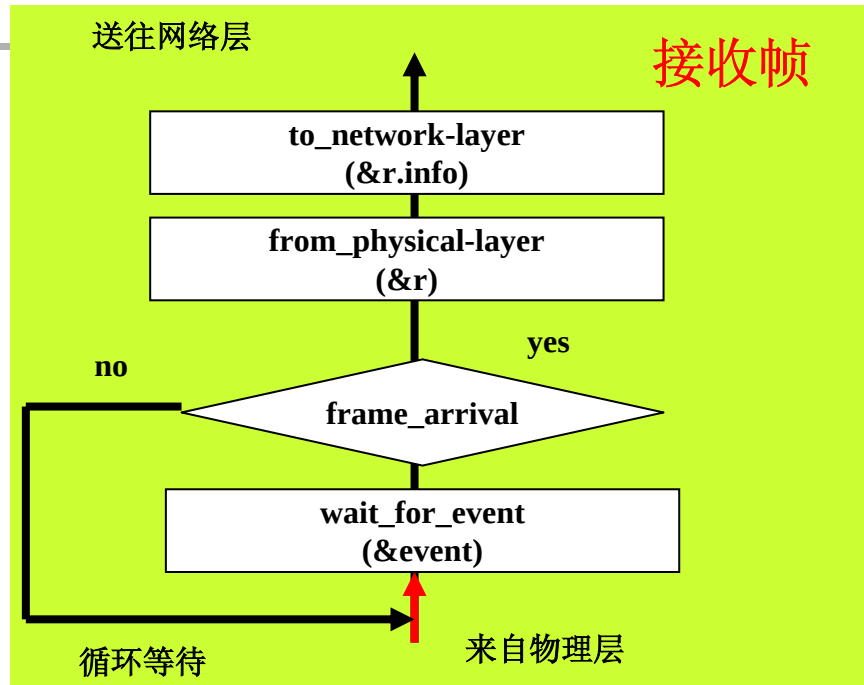
- 提高效率的方法
  - 传输方式：全双工（**full-duplex**）
  - 确认方式：捎带（**piggyback**）
  - 发送/接收方式：滑动窗口（**sliding window**）
- 滑动窗口协议
  - 协议**4**：**n=1**——引出滑动窗口的基本概念
  - 协议**5**：回退**n**帧（**Go Back n**）
  - 协议**6**：选择重传（**Select Repeat**）

传输方式：同时运行发/收进程



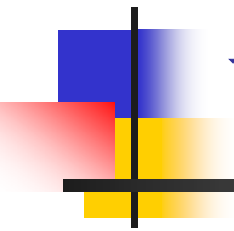
确认方式：在数据帧中捎带确认





A send s to B **A get r from B**

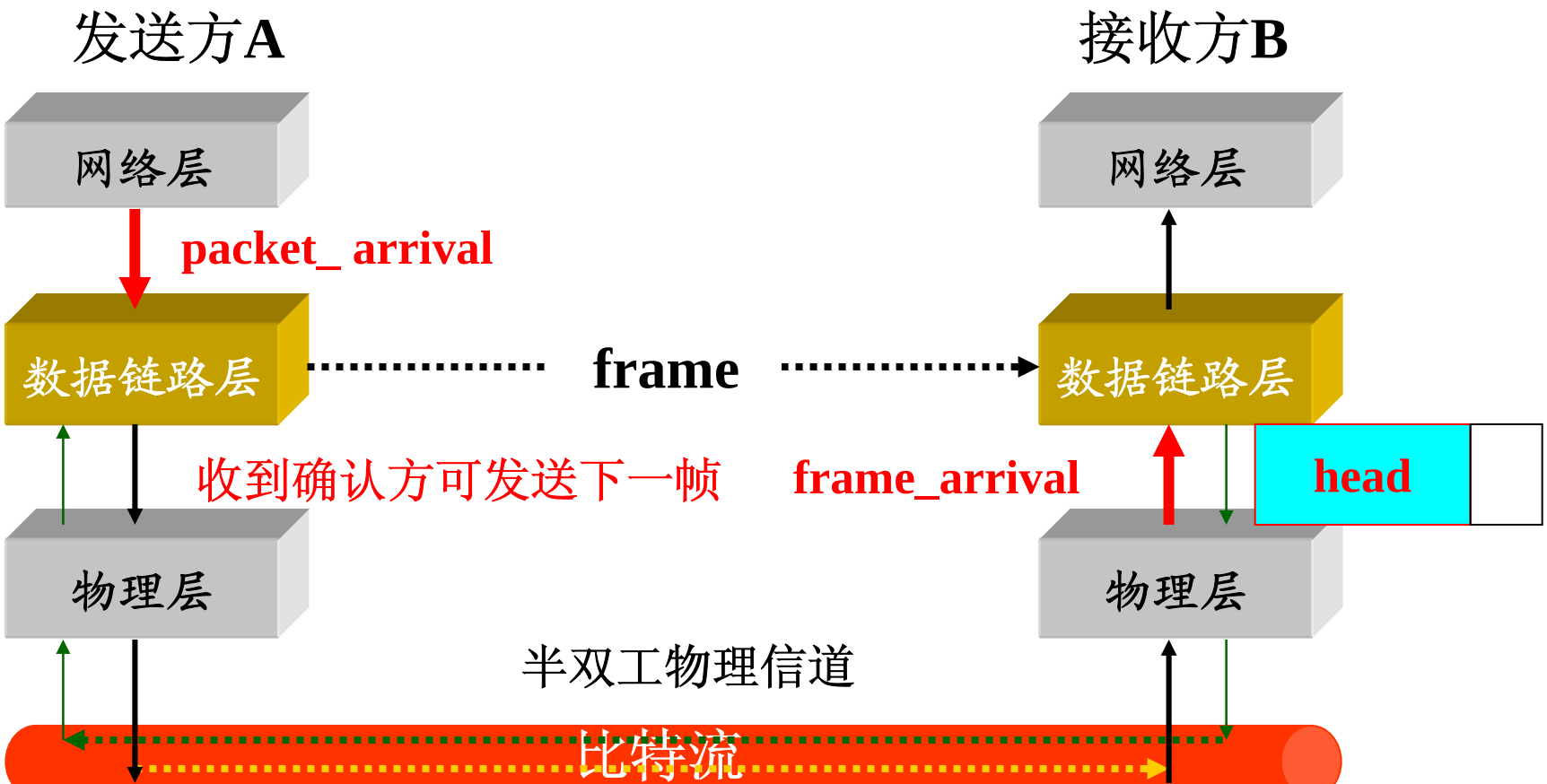
类型	序列号	确认号	数据
data	**seq fromA**	**ack toB**	**packet from A L3**



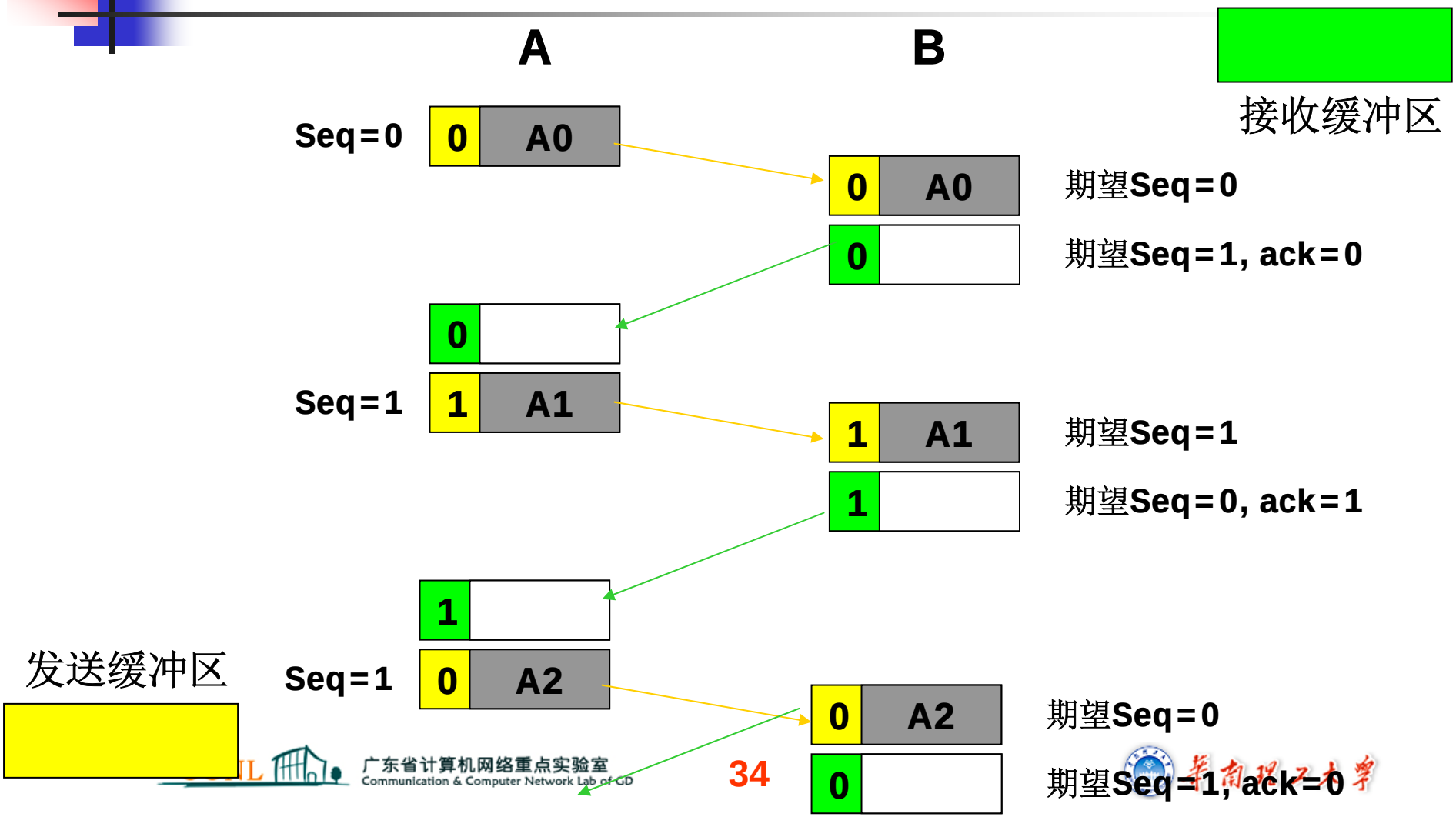
协议4：n=1

从单工的停-等协议2出发，引入双向传输、捎带确认、滑动窗口功能， 得到滑动窗口为1的协议4，并得出滑动窗口的一般概念。

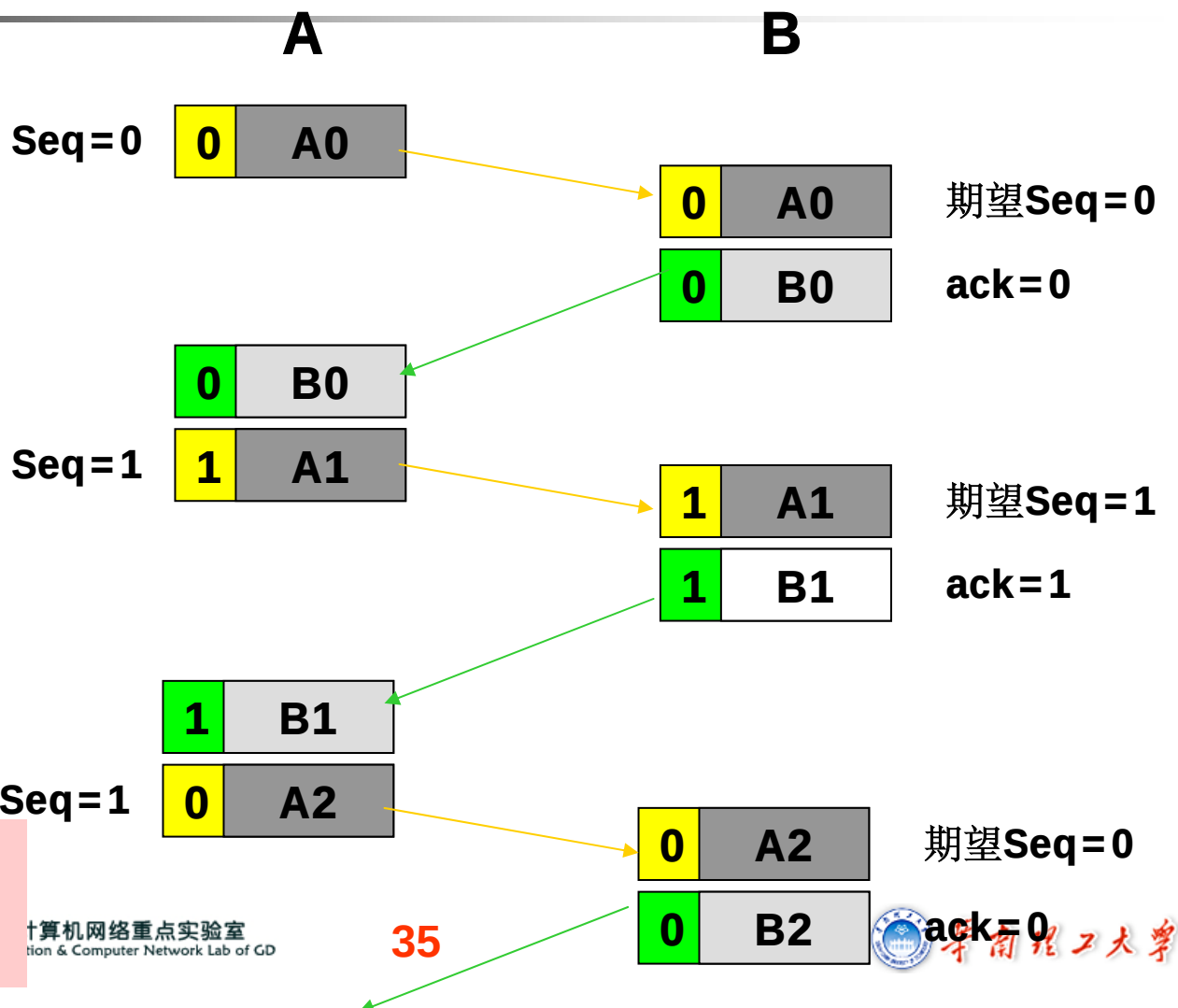
单工的停-等协议（协议2）


发送方A
接收方B
网络层
网络层
packet_ arrival
数据链路层
frame
数据链路层
收到确认方可发送下一帧
frame_arrival
head
物理层
物理层
半双工物理信道
比特流

对数据帧编号，建立缓冲区



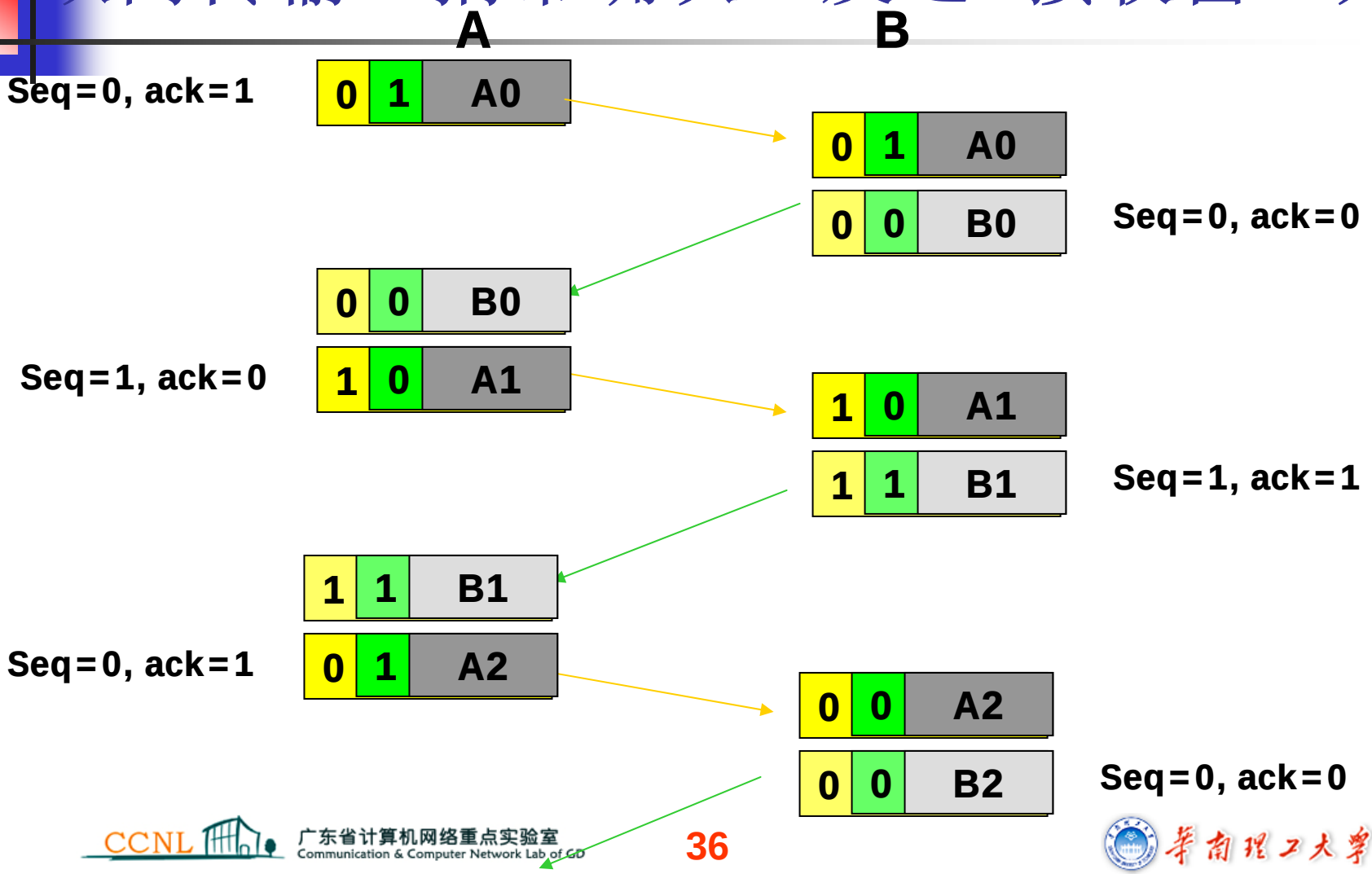
B在发送确认的同时也发送数据



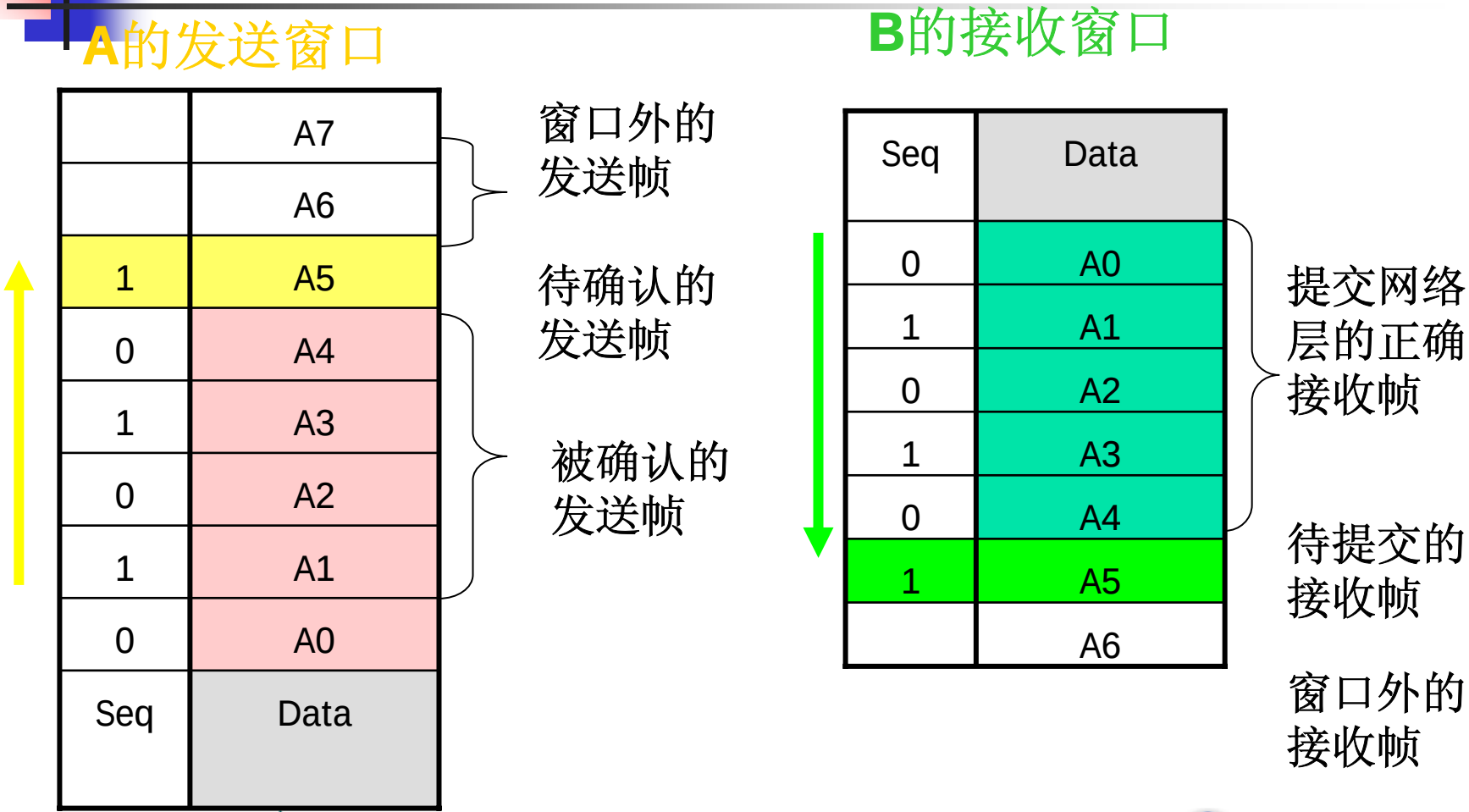
B需要对数据帧编号

A需要对接收帧确认

双向传输、捎带确认、发送/接收窗口为1



n=1滑动窗口概念

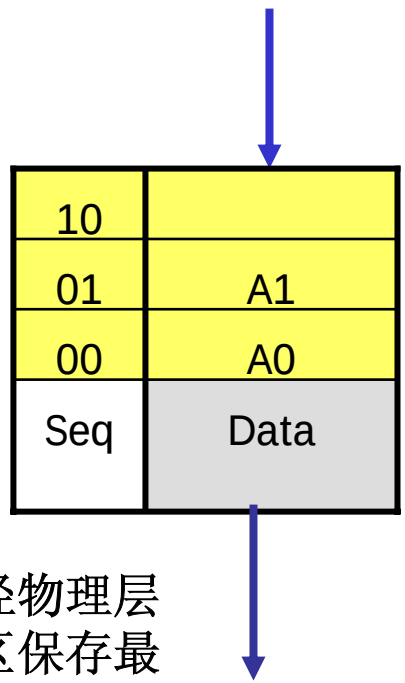

A的发送窗口
B的接收窗口
A7
A6
1
A5
0
A4
1
A3
0
A2
1
A1
0
A0
Seq
Data
窗口外的发送帧
待确认的发送帧
被确认的发送帧
Seq
Data
0
A0
1
A1
0
A2
1
A3
0
A4
1
A5
A6
提交网络层的正确接收帧
待提交的接收帧
窗口外的接收帧

滑动窗口的基本概念

- 每个待发送帧被赋予一个序列号**seq**
  - **seq**的取值范围是 **0 ~ 2^n-1**（**n**位字段）
- 建立缓冲区
  - 发送窗口：缓存已发送、待确认的帧
    - 顺序接收来自网络层的分组，成帧，赋予序列号
    - 最多保存**W**个已经发送、等待确认的帧
    - 窗口达到最大值**W**时强制关闭网络层
  - 接收窗口：缓存已正确接收、待上交网络层的帧
    - 对进入窗口的帧顺序提交网络层，产生确认
    - 落在窗口外的帧被丢弃

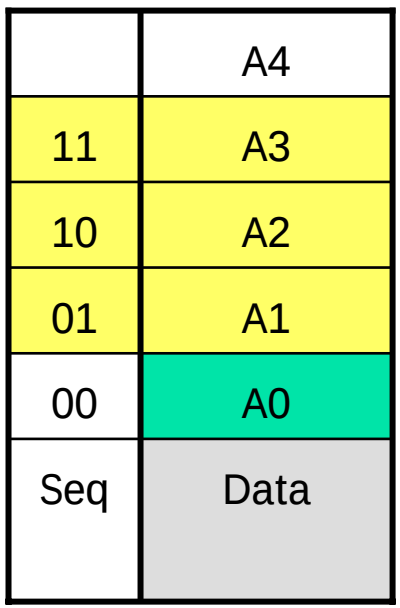
发送窗口的滑动（w=3）

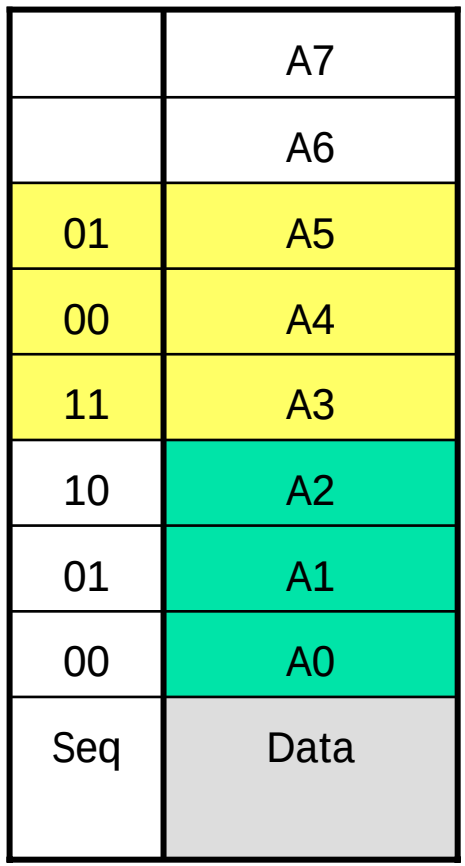
顺序接收来自网络层的分组，成帧



赋予序列号，经物理层发送并在缓冲区保存最多W个待确认帧。

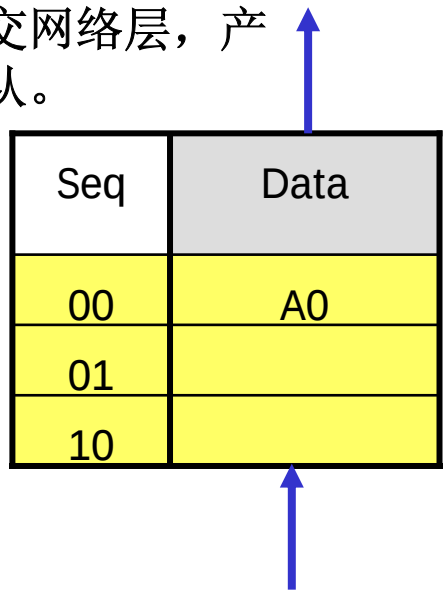
收到确认帧后向后滑动，至窗口满。





接收窗口的滑动（w=3）

对进入窗口的帧顺序提交网络层，产生确认。



Seq	Data
00	A0
01	
10	

接收来自物理层的帧

接收窗口向后滑动，最多保存待上交帧W个。

Seq	Data
00	A0
01	A1
10	A2
11	

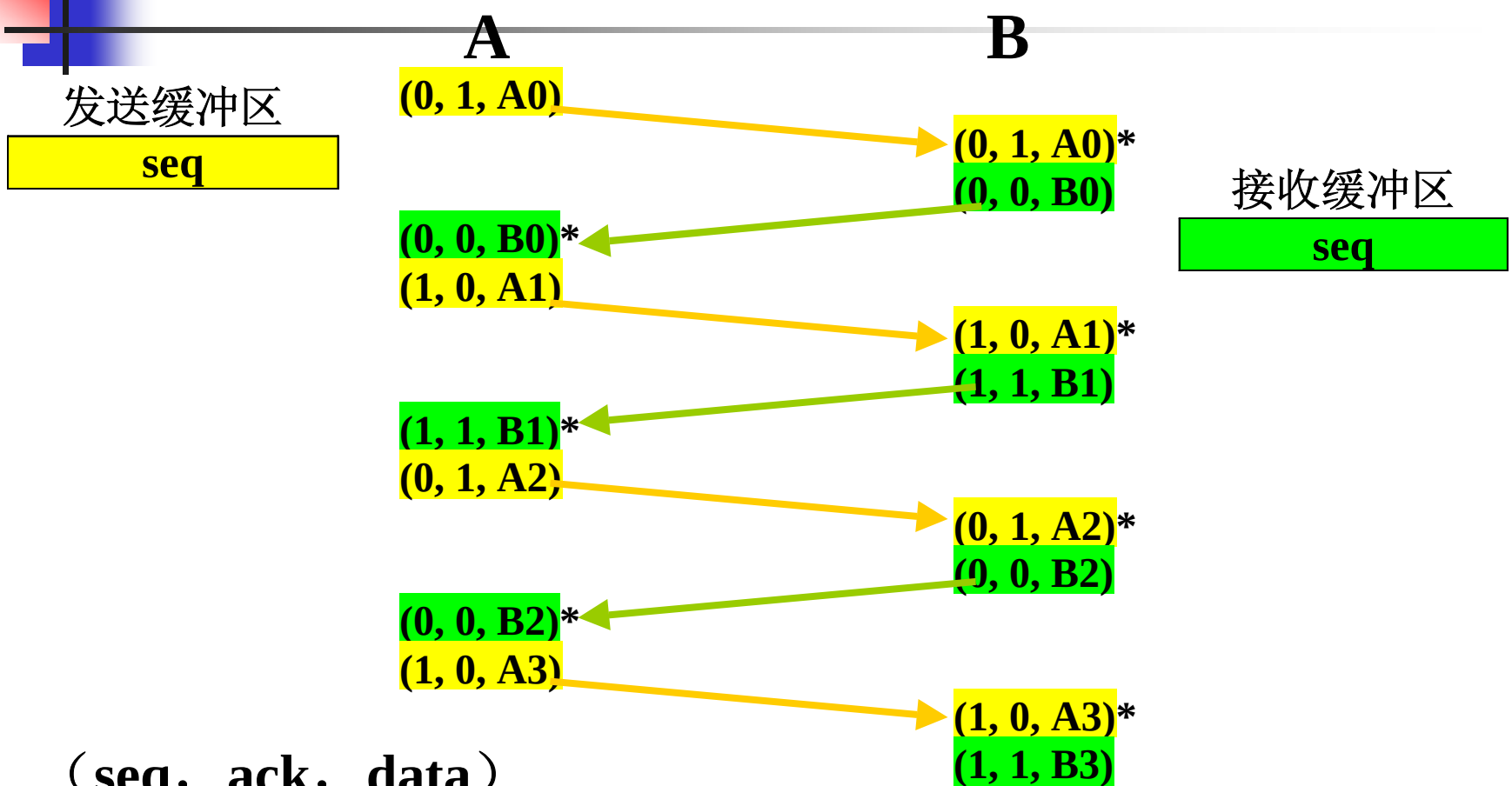
接收窗口满，落在窗口外的帧被丢弃。

Seq	Data
00	A0
01	A1
10	A2
11	A3
00	A4
01	A5
	A6

协议4的滑动窗口基本工作原理

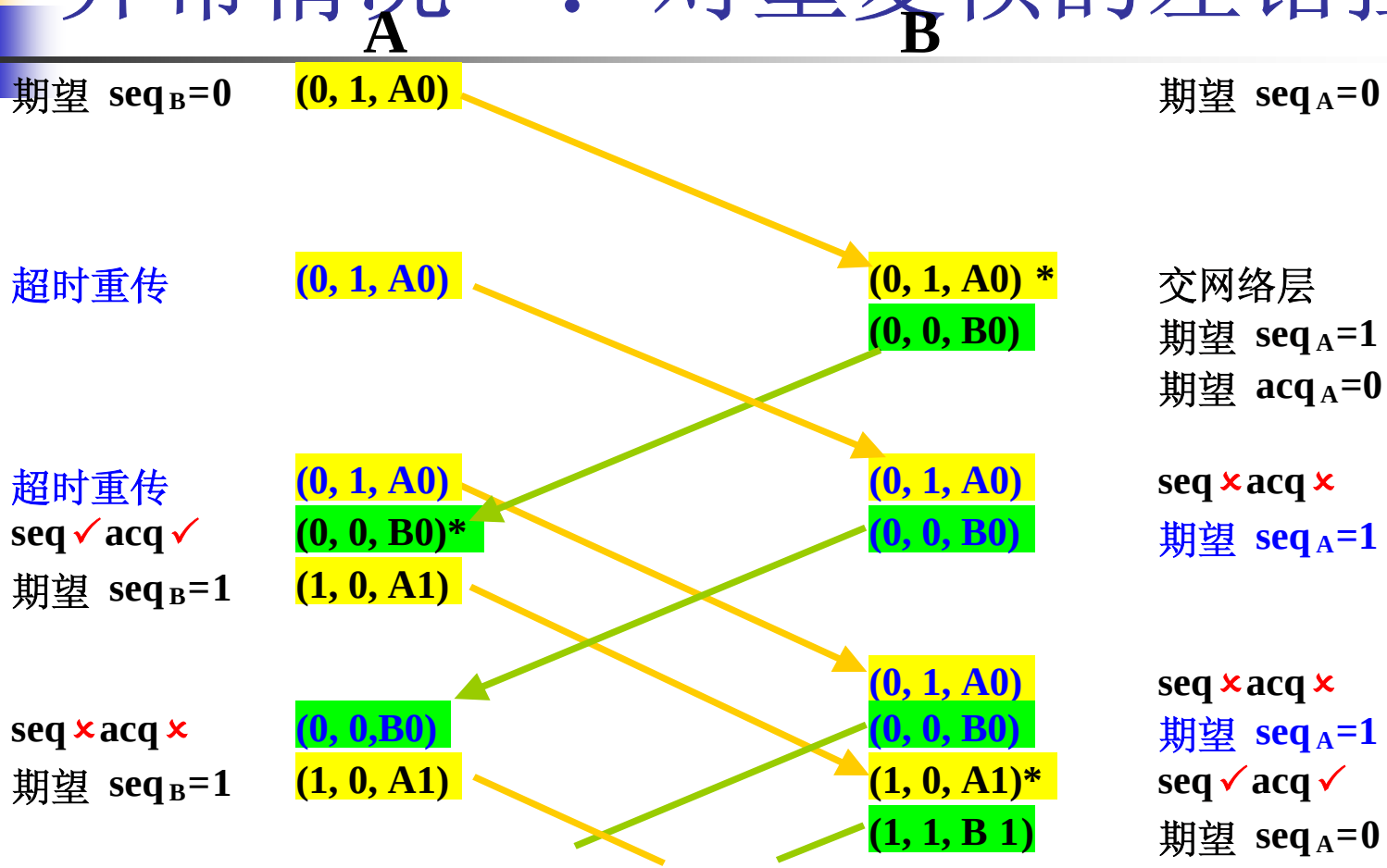
- 窗口设置
  - 滑动窗口最大值：**MAX_SEQ = 1**
  - 通信双方初始值：**seq =0, ack=1**（期待接收**seq=0**）
- 窗口滑动机制
  - **A**首先发送数据帧（**seq=0, ack=1, A0**）
  - **B**收到**A0**，发送捎带确认帧（**seq=0, ack=0, B0**）
  - **A**收到对**A0**的确认，滑动窗口，发送帧（**seq=1, ack=0, A1**）
- 特点
  - 序列号**seq**和确认值**ack“0”“1”**交替
  - 滑动窗口长度**W=1**，收到确认才移动窗口
  - 保证按顺序将接收到的正确帧只一次上交网络层

正常情况下发送窗口滑动机制

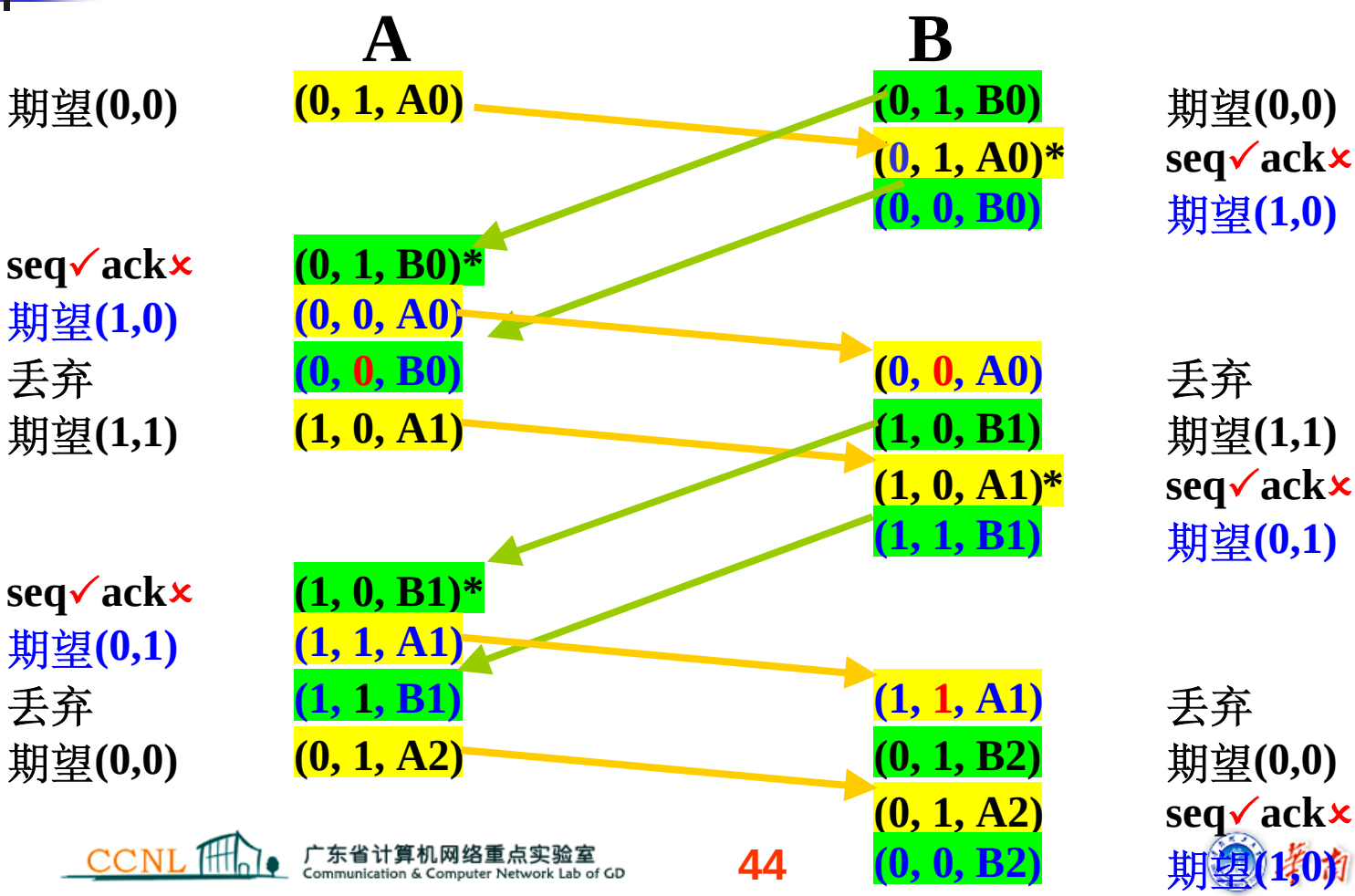


（seq，ack，data）

异常情况一：对重复帧的差错控制



异常情况二：同步开始发送过程的差错控制



协议4的信道利用率P181~182

通过例题分析协议4（双向停-等协议）存在的信道利用率低问题，其解决办法为管道化技术（pipelining），并提出差错控制面临的新问题。

例题P181

- 已知：
  - 信道容量 **b = 50 kbps**
  - 传输延迟 **R = 500 ms**（双程）
  - 数据帧的长度 **L = 1000 bit**
  - 设接收方收到数据帧后马上回送确认短帧，没有延时
- 求：信道利用率

例题解答

- 在源端发送数据帧过程需要的时间

  $T_f = L/b = 20\ ms$

- 从发送完毕到确认帧返回需要的时间（双程延迟）

  $R = 500\ ms$

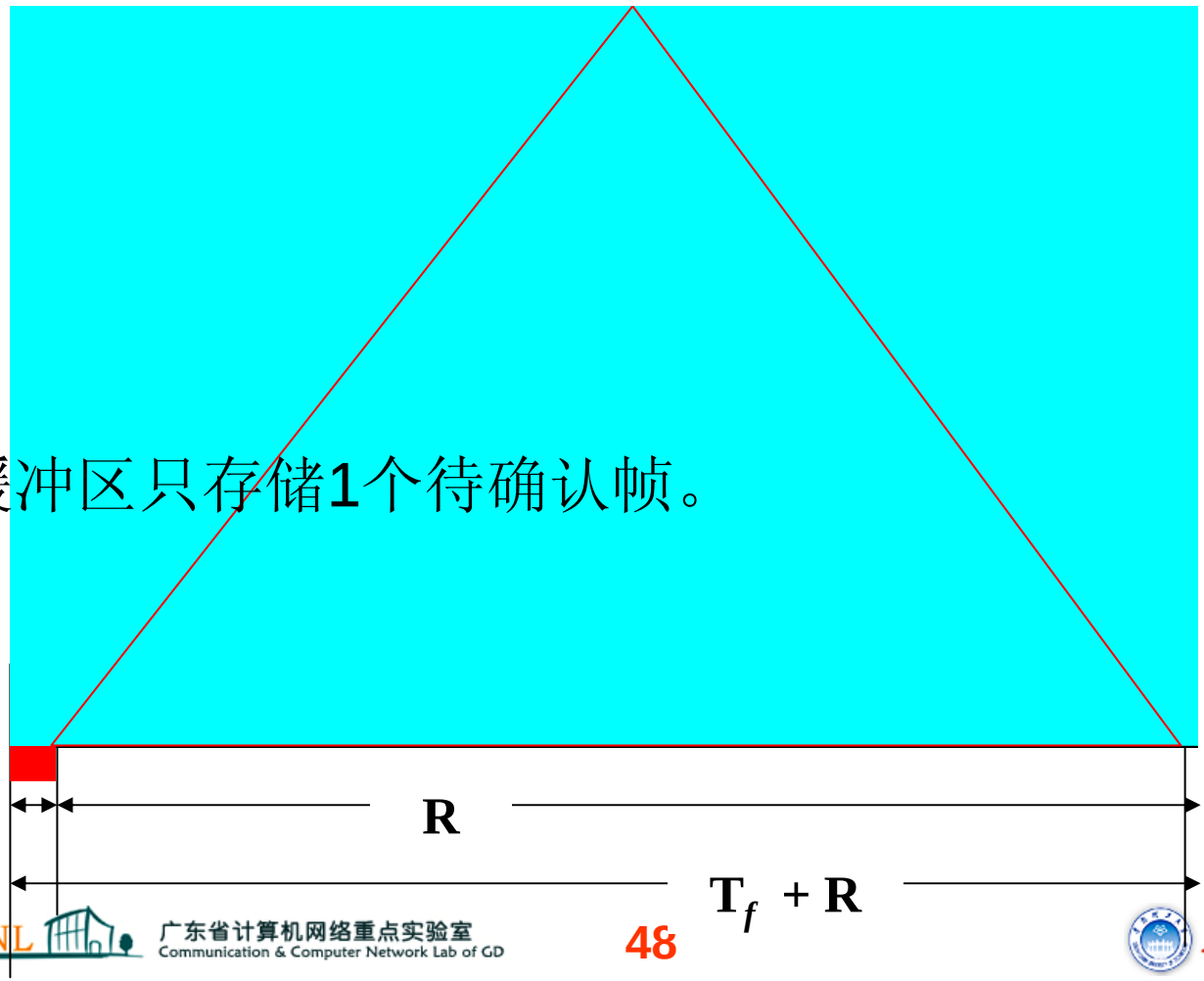
- 从开始发送到确认返回总共需要的时间

  $(T_f + R) = 20 + 500 = 520\ ms$

- 线路的利用率

  $T_f/(T_f + R) = 20/520 = 3.85\ \%$

信道利用率不足4%



W = 1

在发送缓冲区只存储1个待确认帧。

提高信道利用率的方法

- 增加滑动窗口最大长度**W**

  信道利用率 $= W*T_f / (T_f + R)$

  $= W* L / (L + bR)$

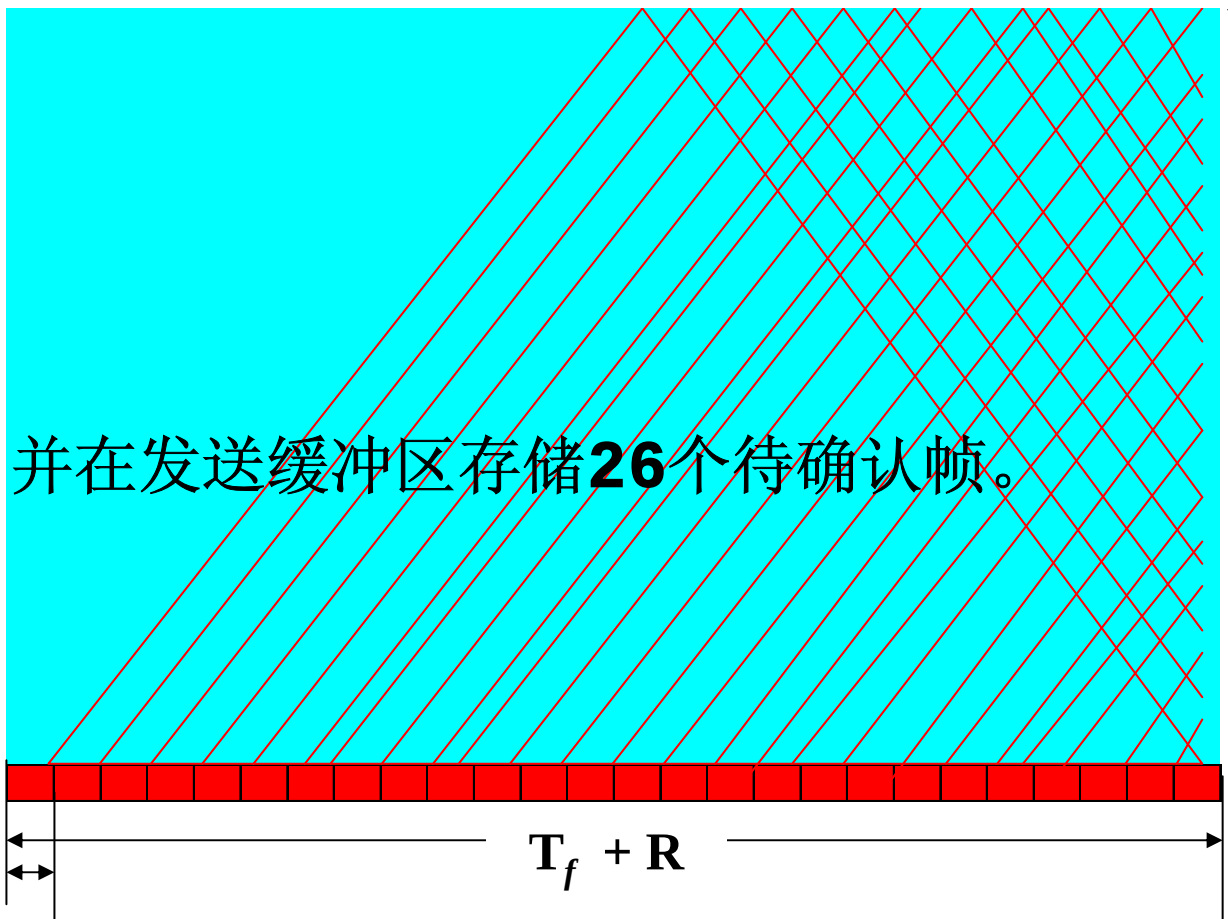
- 理想情况下，使例题信道利用率达到**100%**，则滑动窗口最大长度为：

  $W = (T_f + R) / T_f = 520 / 20 = 26$

使信道利用率达到100％

W = 26

连续发送、并在发送缓冲区存储**26**个待确认帧。



管道化（**pipeling**）

用管道化技术发送帧面临的新问题

- 出错情况
  - 连续发送**W**个数据帧，其中有一帧出错，但其后续帧被成功发送
- 接收方的接收策略选择
  - 丢弃错帧，其后续帧因不是期望接收帧也被丢弃
  - 丢弃错帧，缓存后续正确接收帧
- 对应的发送方的重传策略选择
  - 缓存在发送窗口中的出错帧以及其后续帧全部重发
  - 只重发出错帧

协议5：回退n帧

接收方的接收策略选择：

丢弃错帧，其后续帧因不是期望接收帧也被丢弃（接收窗口为1）。

发送方的重传策略选择：

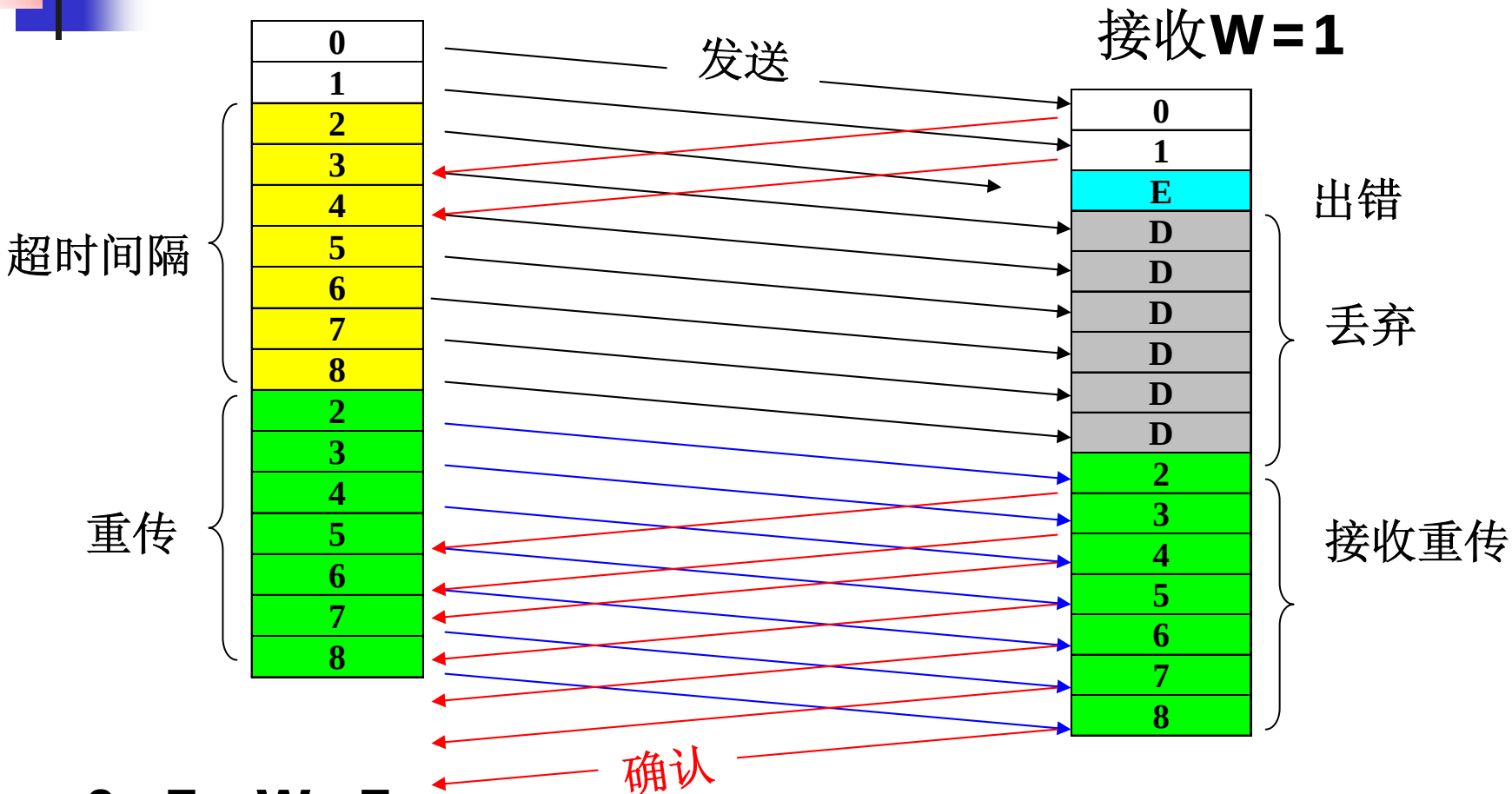
缓存在发送窗口中的出错帧以及其后续帧全部重发。

回退n帧协议的基本概念

- 定义序列号**seq**的取值范围和滑动窗口长度**W**
- 发送方连续发送至发送窗口满
- 接收窗口为**1**，对丢弃帧不确认
- 发送方超时重传，从未被确认帧开始

举例（**MAX_SEQ = 7)**

回退n帧协议举例



Seq=0~7；W=7

回退n帧协议的工作原理分析

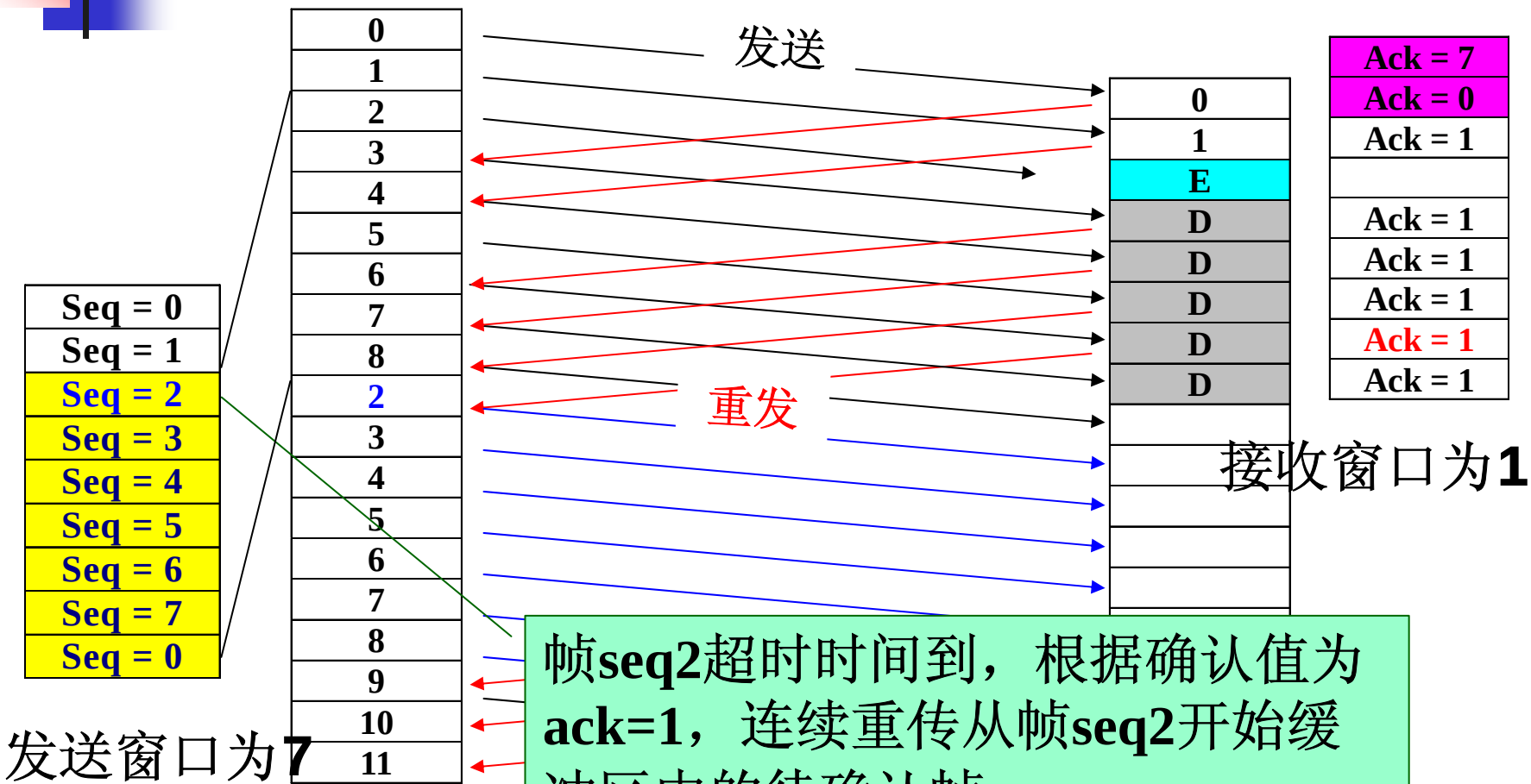
发送方

- 正常发送
  - 对帧编号，待确认帧缓存
- 收到确认
  - 释放确认帧所占缓冲区，滑动发送窗口
- 差错帧超时时间到
  - 回退到超时帧（出错帧），顺序重传最后被确认帧以后的缓冲区中缓存的帧

接收方

- 收到每一个期望的正确帧
  - 上交网络层、回送确认
- 收到出错帧或非期望帧
  - 丢弃，回送对接收的最后正确帧的确认
- 收到重传帧（即为一个期望的正确帧）

关键步骤：帧seq2超时，回退7帧重传



协议6：选择重传

接收方的接收策略选择：

丢弃错帧，缓存后续正确接收帧；

发送方的重传策略选择：

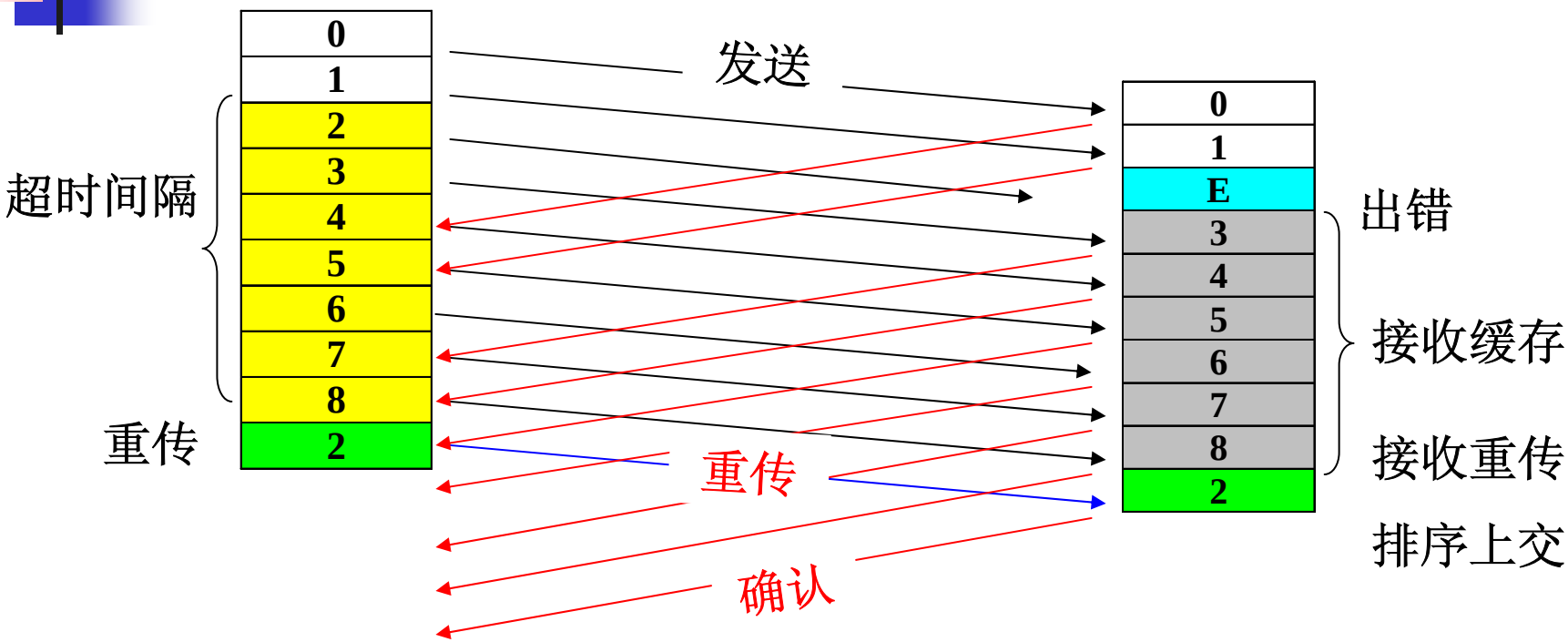
只重发出错帧。

选择重传协议的基本概念

- 接收窗口存储差错帧后继的所有正确帧
- 发送方只重传差错帧
- 接收方接收重传帧，按正确顺序将分组提交网络层

举例（**MAX_SEQ = 15**）

选择重传协议举例



Seq=0~15；W=8

选择重传协议的工作原理分析

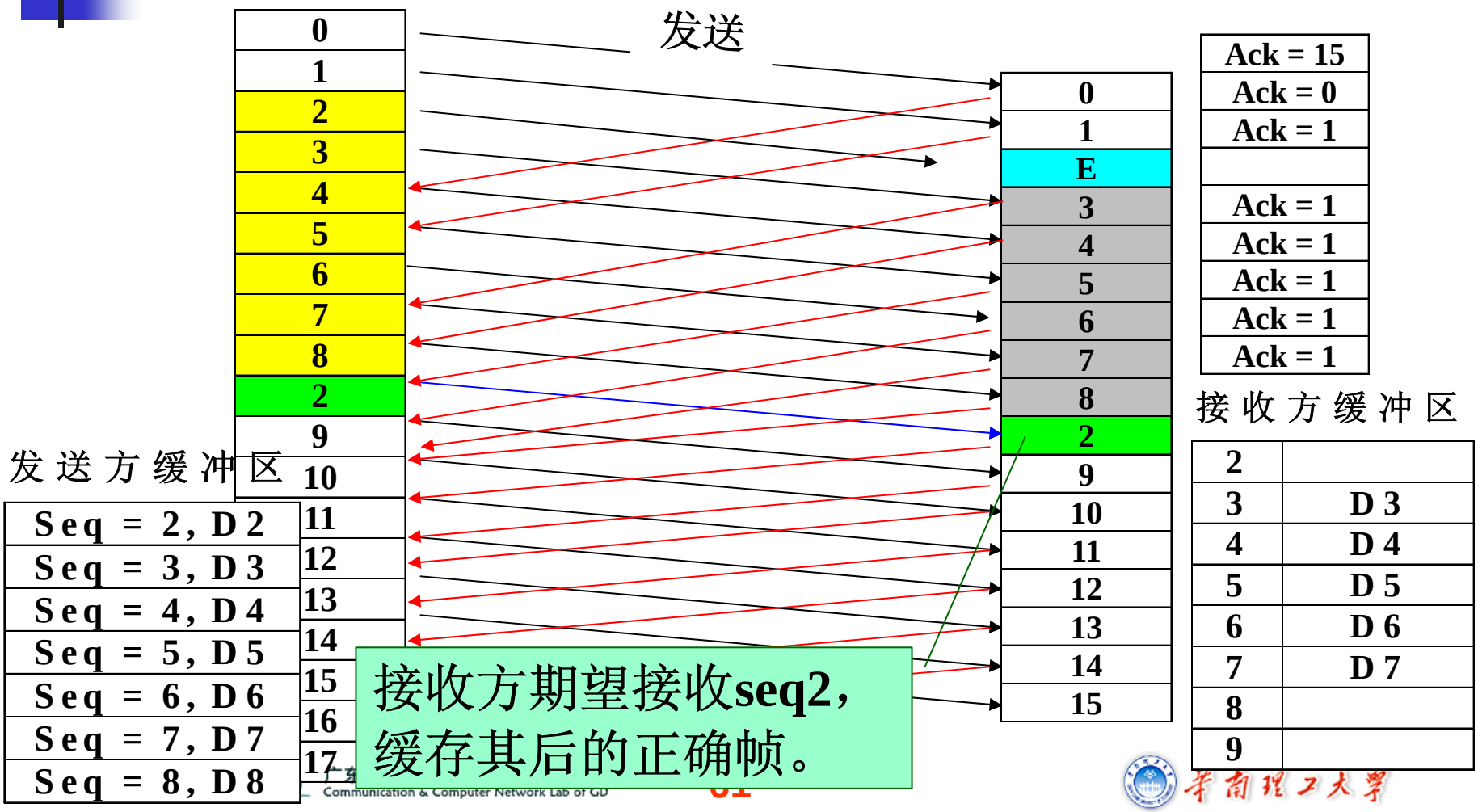
发送方

- 正常发送
  - 对帧编号，待确认帧缓存
- 收到确认
  - 释放确认帧所占缓冲区，滑动发送窗口
- 差错帧超时时间到
  - 重传缓存的最后被确认帧以后的那一帧

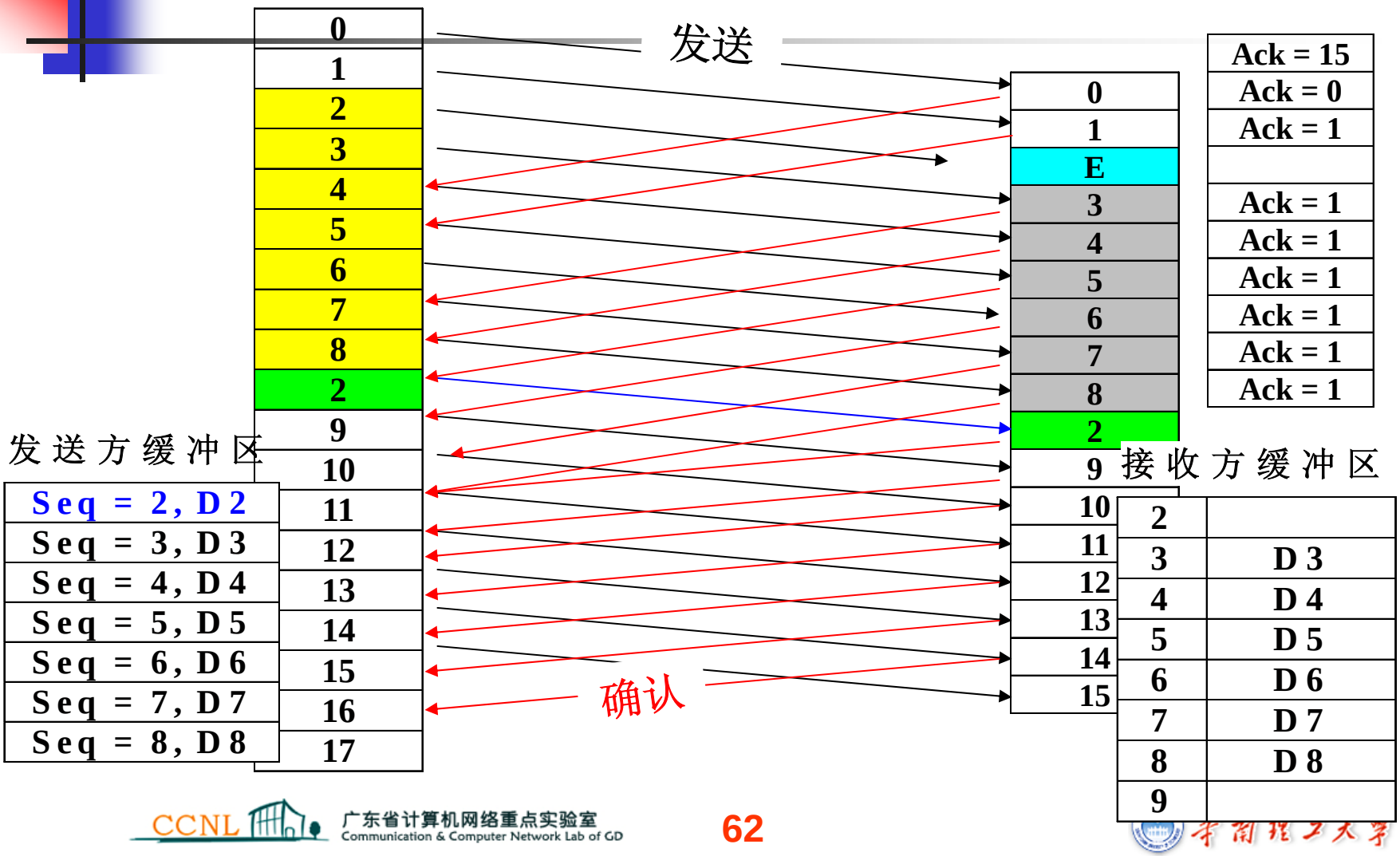
接收方

- 正常接收
  - 上交网络层、回送确认，滑动接收窗口
- 收到非期望的正确帧
  - 缓存，回送对接收的最后正确帧的确认
- 收到重传帧
  - 将缓存帧排序上交，回送确认，滑动接收窗口

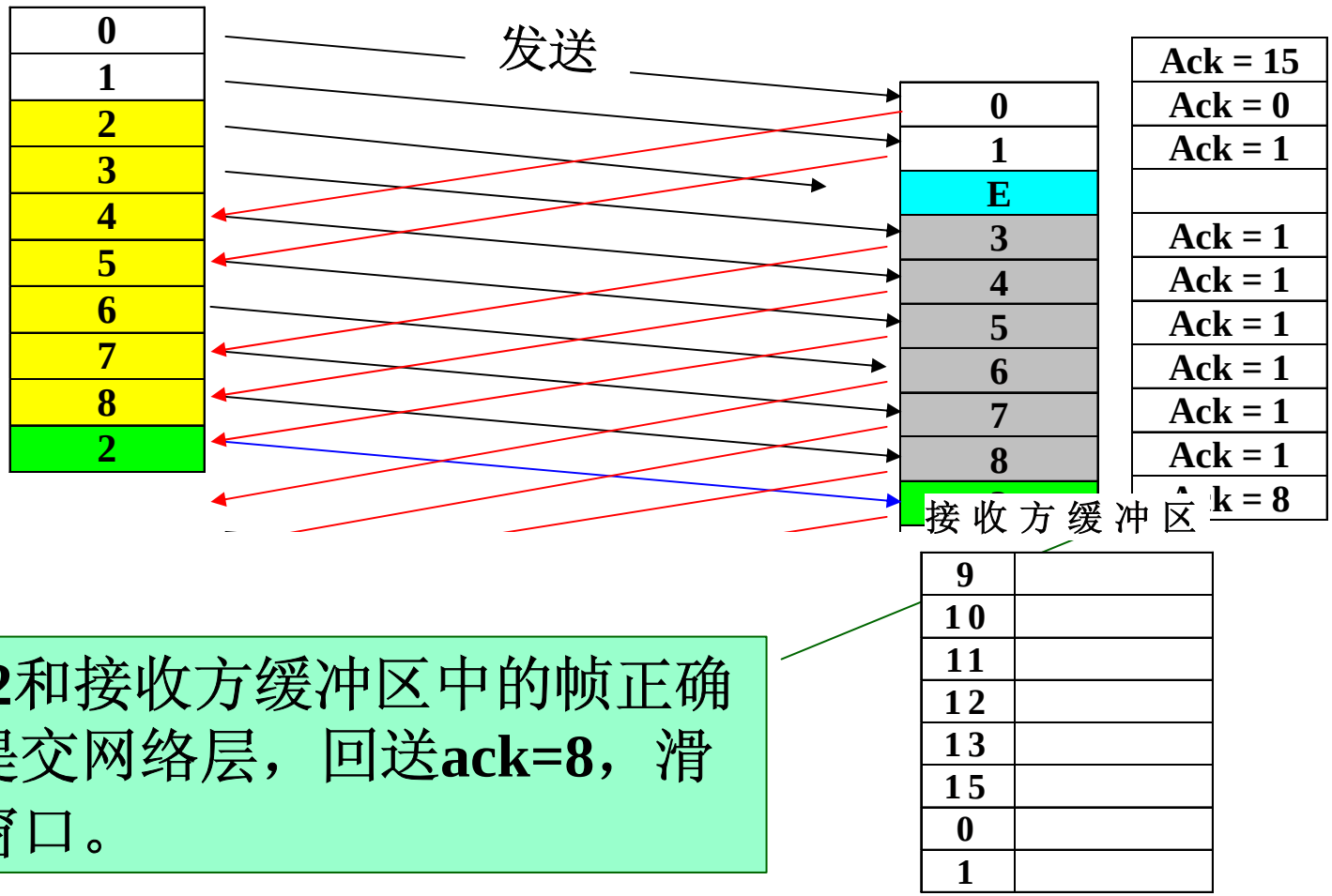
关键步骤：
接收方收到非期望的正确帧—缓存



关键步骤：
发送方选择帧seq2重传



关键步骤：
接收方收到重传帧seq2－排序上交



将帧seq2和接收方缓冲区中的帧正确排序，提交网络层，回送ack=8，滑动接收窗口。

NAK的作用 P183

- 加快出错帧的重传
- 对出错帧回送否定确认，使发方不再等到超时再重传

差错控制策略比较和滑动窗口大小的选择

差错控制策略比较

- 回退n帧
  - 发送方需要较大的缓冲区，以便重传
  - 重传帧数多，适于信道出错率较少的情况
- 选择重传
  - 接收方也需要较大的缓冲区，以便按正确顺序将分组提交网络层
  - 重传帧数少，适于信道质量不好的情况

滑动窗口长度w的选择

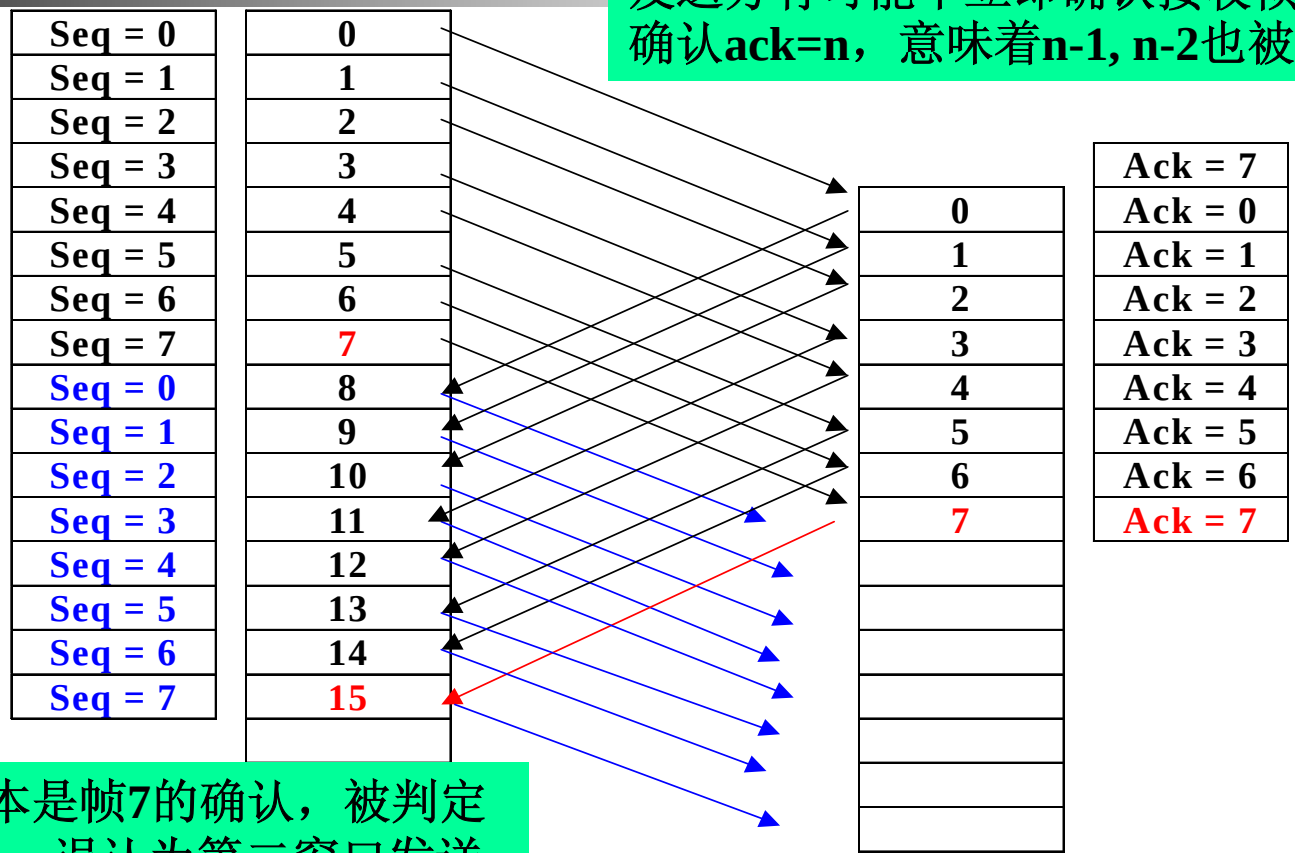
- 协议5（回退n帧）
  - MAX_SEQ = 7（ Seq=0～ MAX_SEQ ）
  - W = 7

  W = MAX_SEQ
- 协议6（选择重传）
  - 课堂举例
    - MAX_SEQ =15（ Seq=0～ MAX_SEQ ）
    - W = 8
  - 教材举例
    - MAX_SEQ =7（ Seq=0～ MAX_SEQ ）
    - W = 4

    W= (MAX_SEQ + 1) / 2

协议5：W = 8，异常情况P186

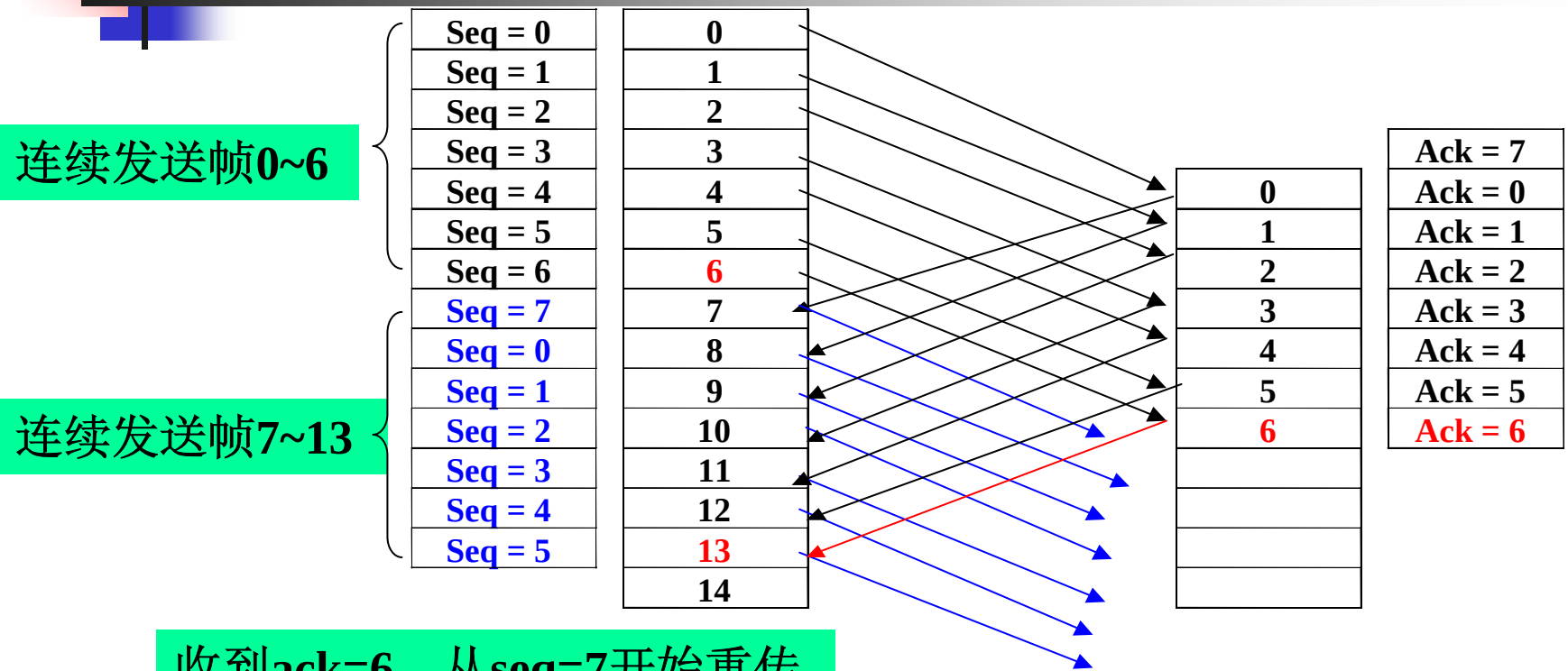
发送方有可能不立即确认接收帧，捎带确认ack=n，意味着n-1, n-2也被确认。



收到ack=7，本是帧7的确认，被判定为帧15的确认，误认为第二窗口发送成功，开始发送后续帧。

第二个窗口发送的数据帧全部丢失。

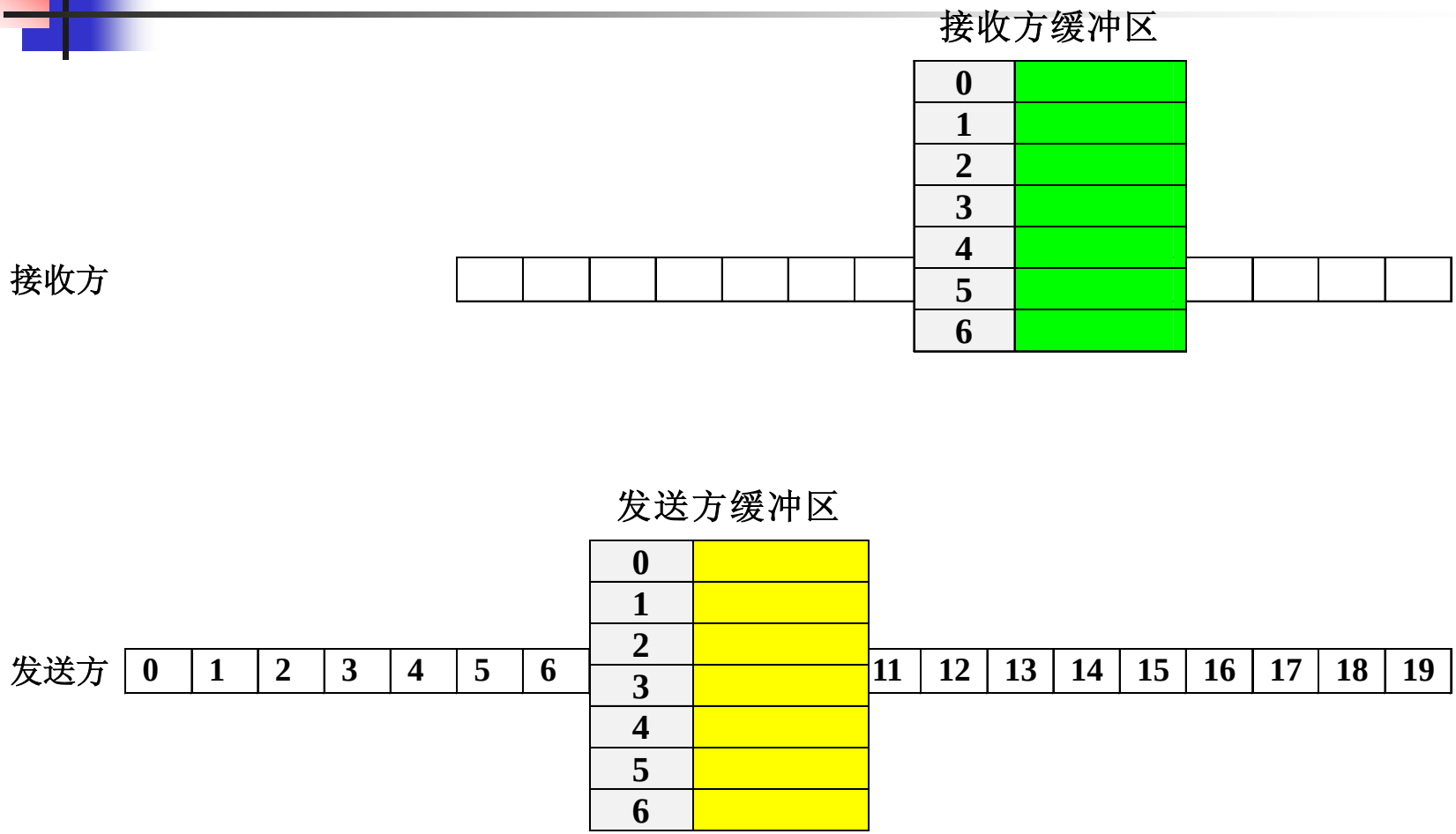
协议5：W = 7，异常情况



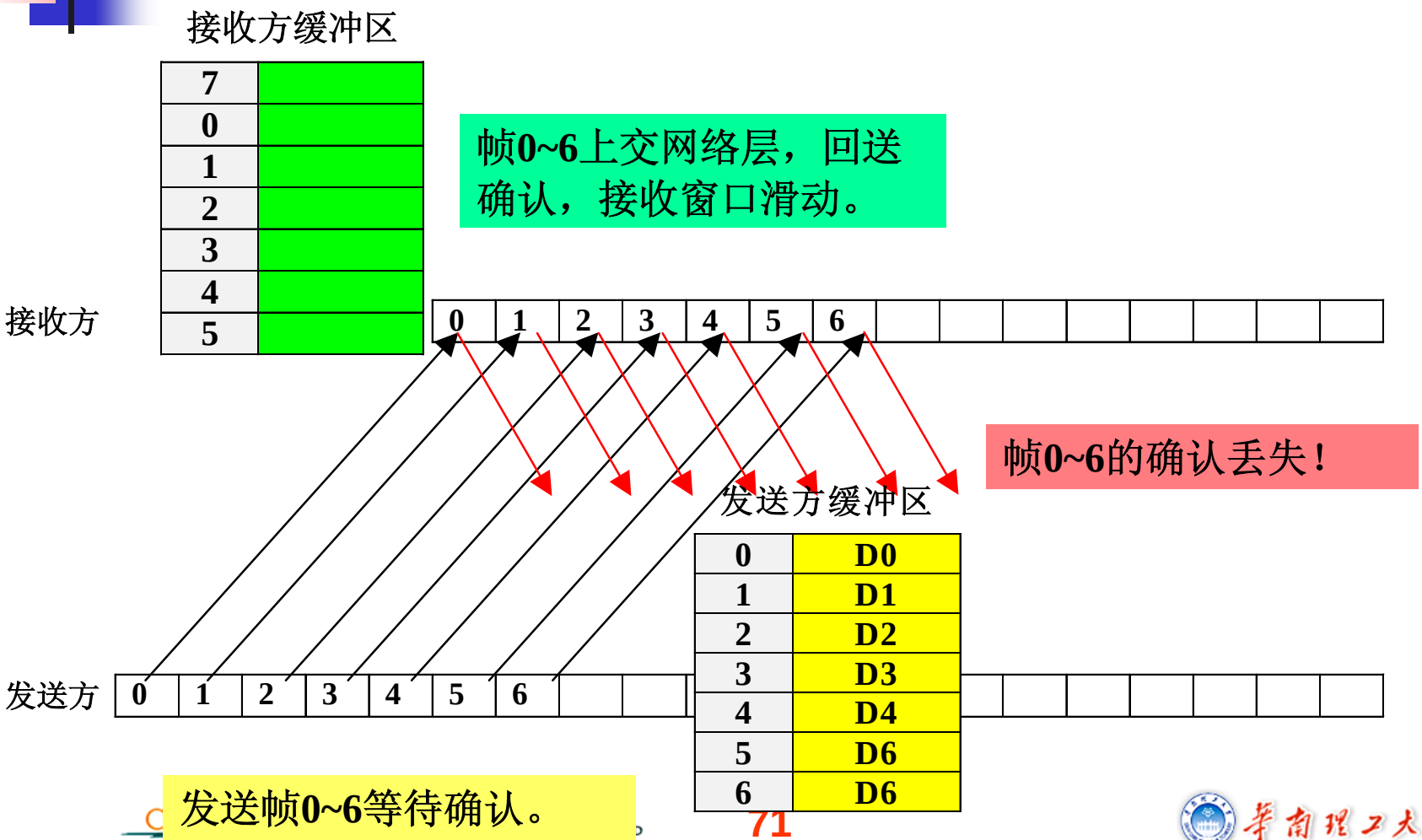
收到ack=6，从seq=7开始重传第二窗口的数据帧，不会误认为第二窗口发送成功。

第二个窗口发送的数据帧全部丢失。

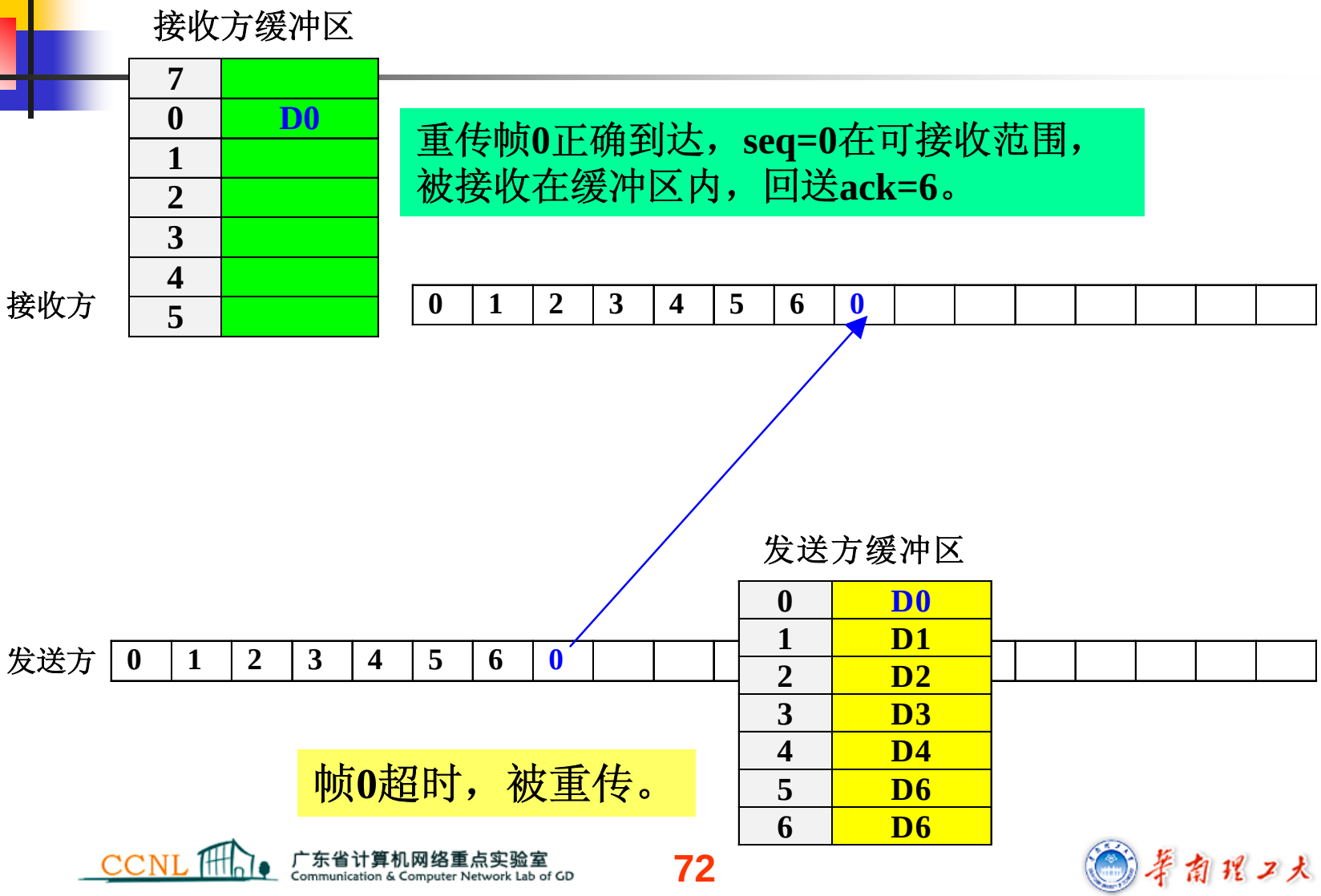
协议6：W=7，初始缓冲区空



协议6：帧0~6发送成功、正确接收



协议6：帧0超时重传并被正确接收

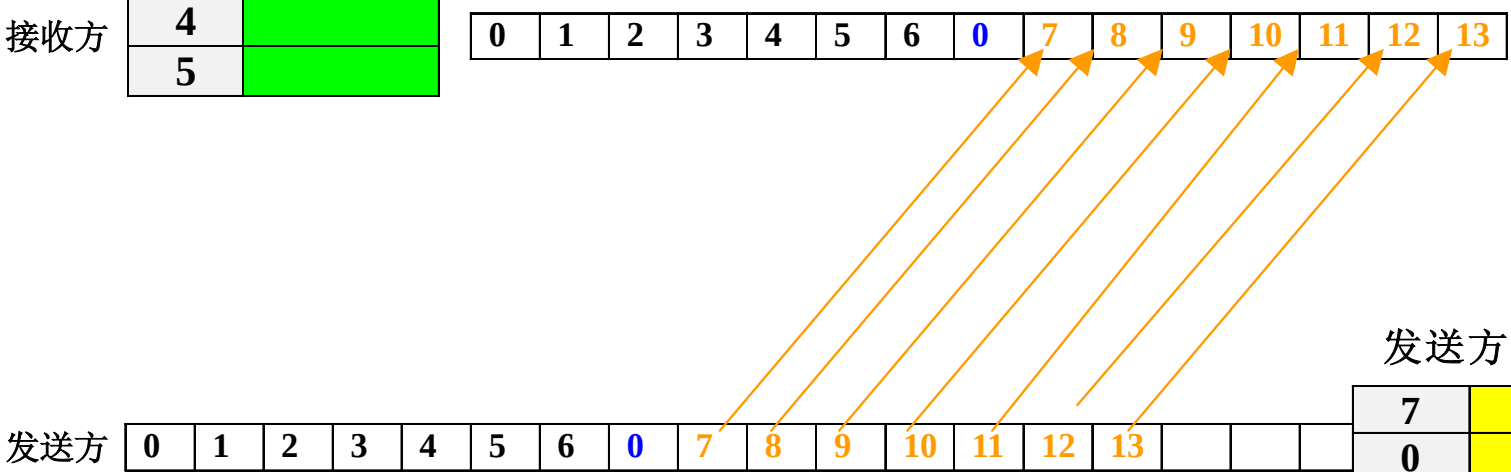


协议6：帧0被重复提交

接收方缓冲区

7	D7
0	D0
1	
2	
3	
4	
5	

第7帧正确提交，重传帧0被认为是正确帧提交，出现重复提交错误。

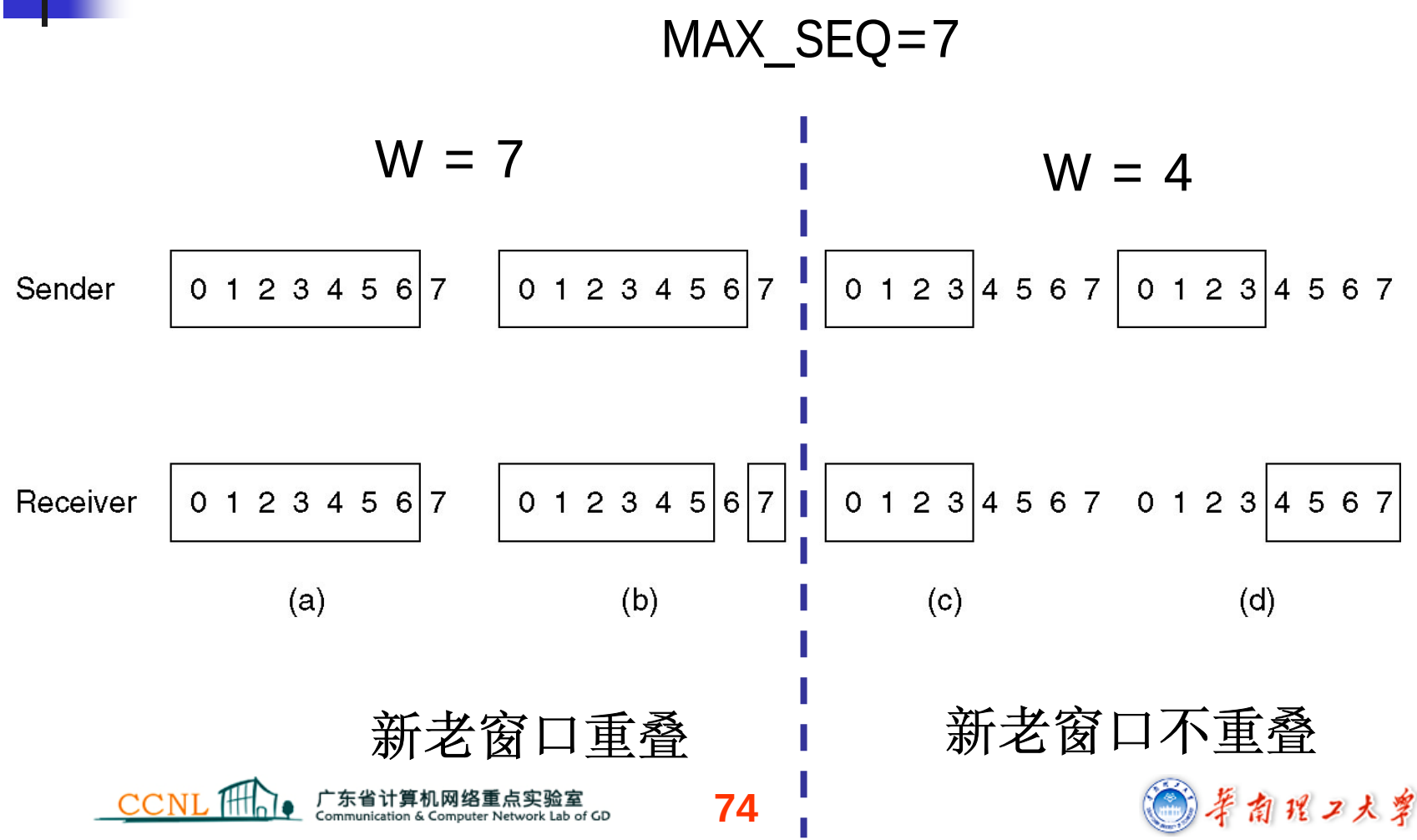


发送方缓冲区

7	D7
0	D8
1	D9
2	D10
3	D11
4	D12
5	D13

收到第6帧的确认，认为0～6被正确接收，发送第7～13帧。。

解决办法：保证新老窗口不重叠



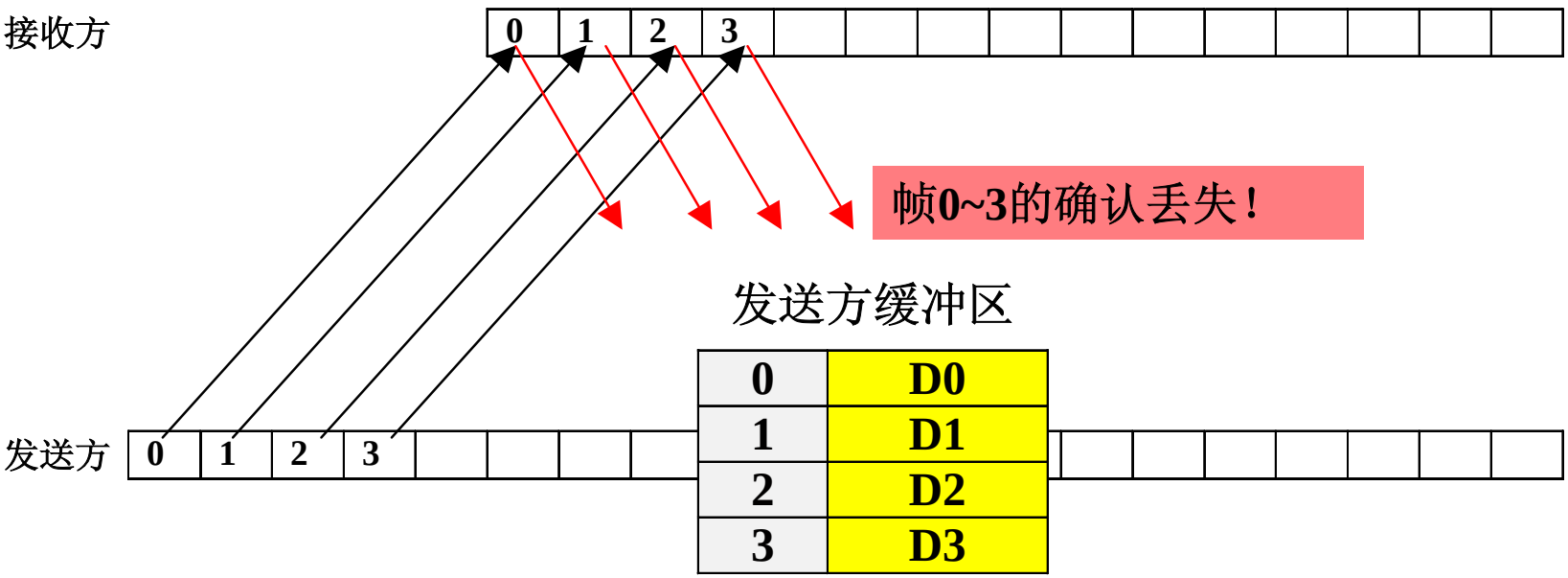
协议6：W=(MAX_SEQ+1)/2

帧0～3的重传帧落在接收窗口外，被拒绝，不会出现重复提交错误。

接收方缓冲区

4	
5	
6	
7	

帧0~3上交网络层，回送确认，接收窗口滑动。



发送帧0~3等待确认。

本节小结

- 理解掌握**6**种基本的**DLL**协议
- 理解涉及到的相关问题，例如肯定确认重传技术、捎带确认、滑动窗口技术、管道技术

谢 谢！