トランスポートレイヤ技術

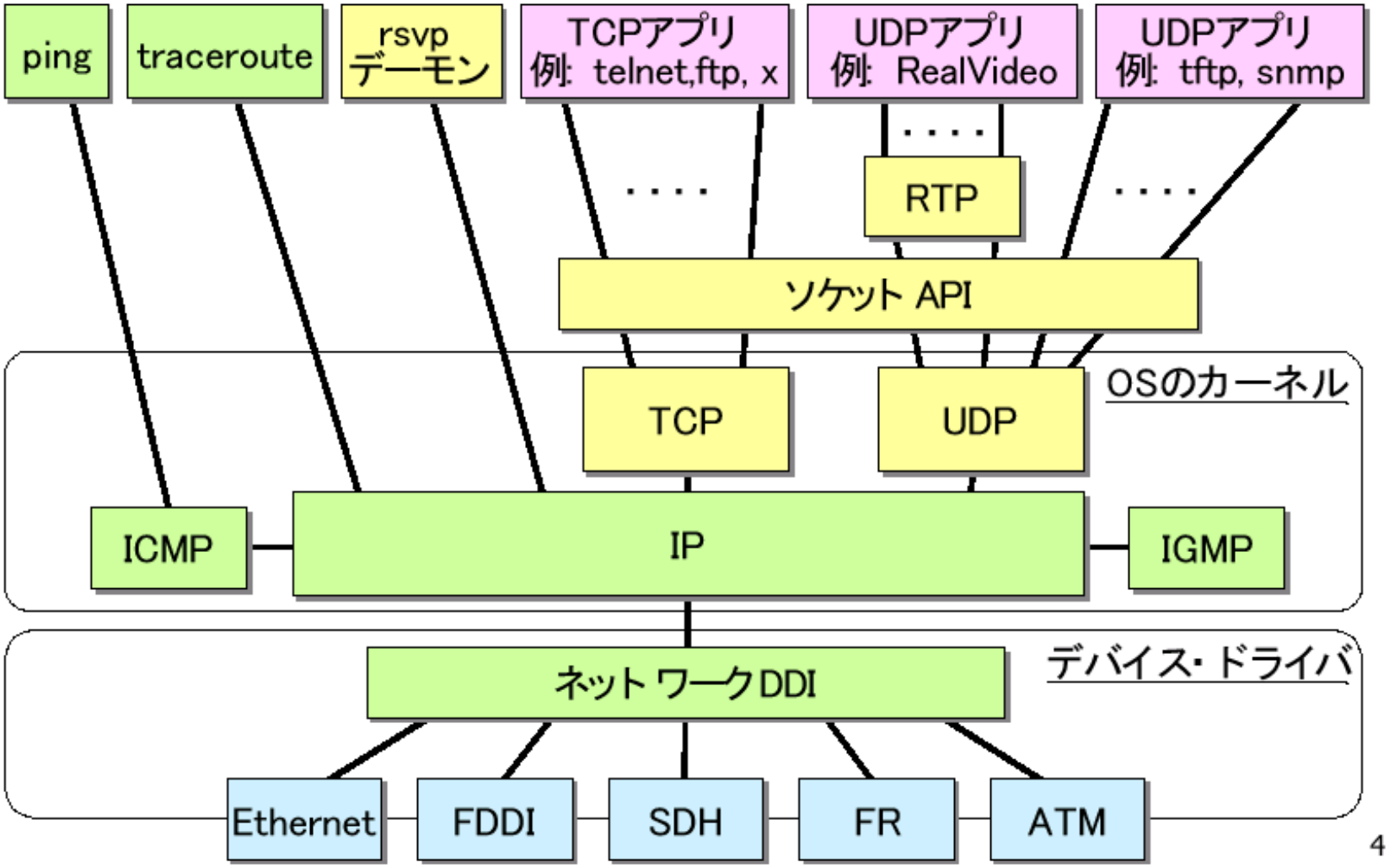
- TCP; Transmission Control Protocol -

トランスポートレイヤの仕事

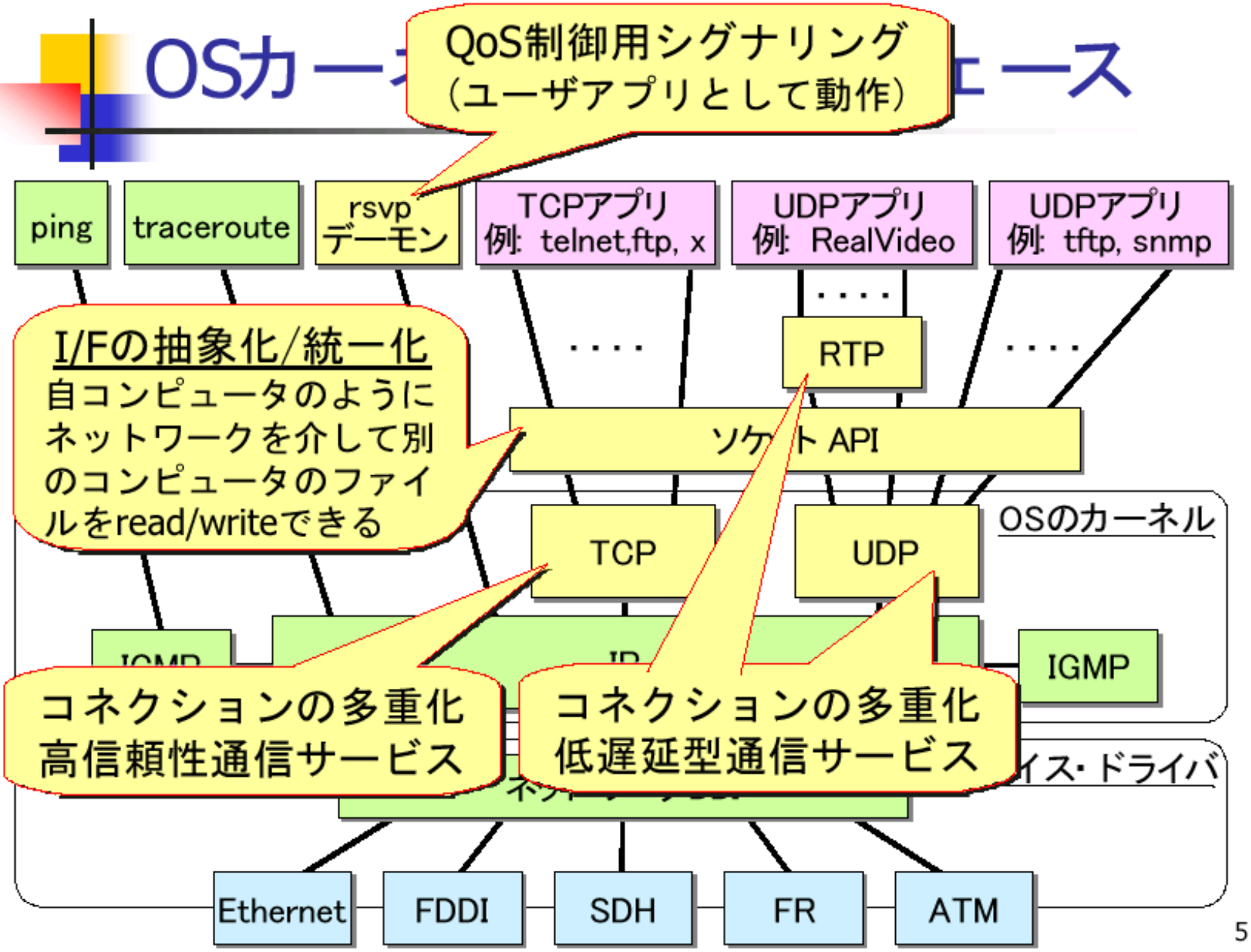
- 計算機のインターフェース(Socket)間での 良好なデータのやり取りを実現する。
  - 誤りがないように
    - 再送
    - パリティー情報による自動再生(FEC; Forward Error Correction)
  - データを取りこぼさないように

  (*) ファイルアクセスと同じ インターフェース を提供
- それ以外に欲しくなる機能
  - 並列データ転送
  - ネットワークに “やさしく”
    - 道が混まないように
    - ネットワークは単純化、エンドホストが賢く

OSカーネルとインターフェース

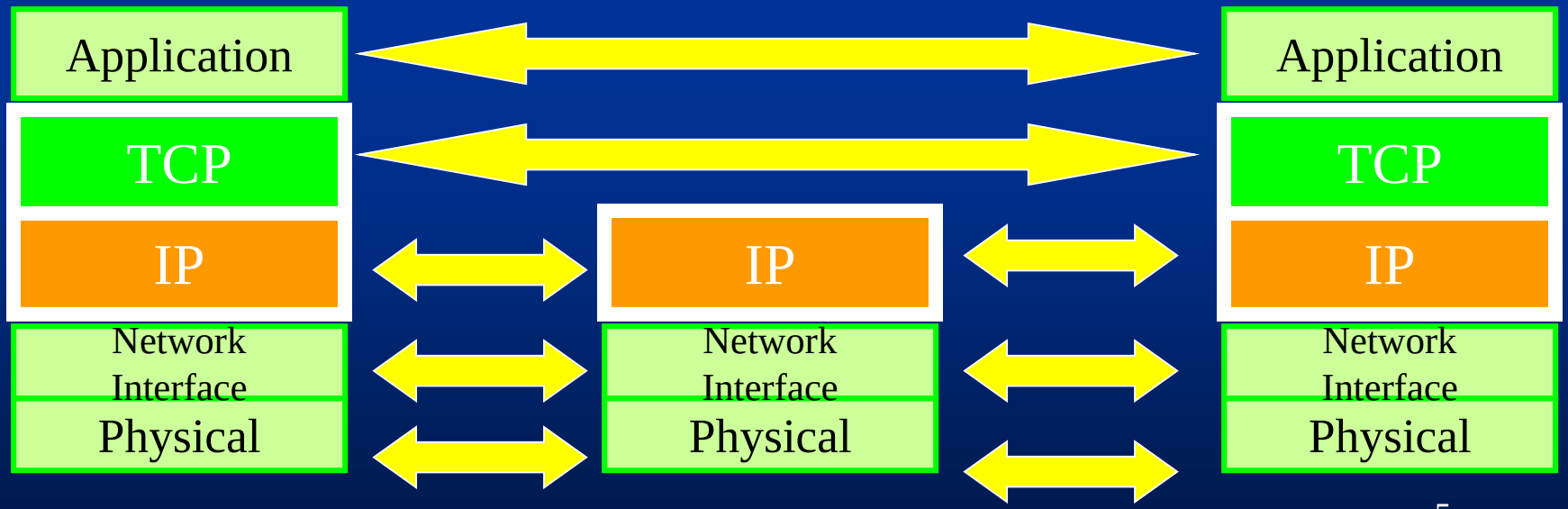


OSカーネルインタフェース



インターネットアーキテクチャ
- TCP : Transmission Control Protocol -

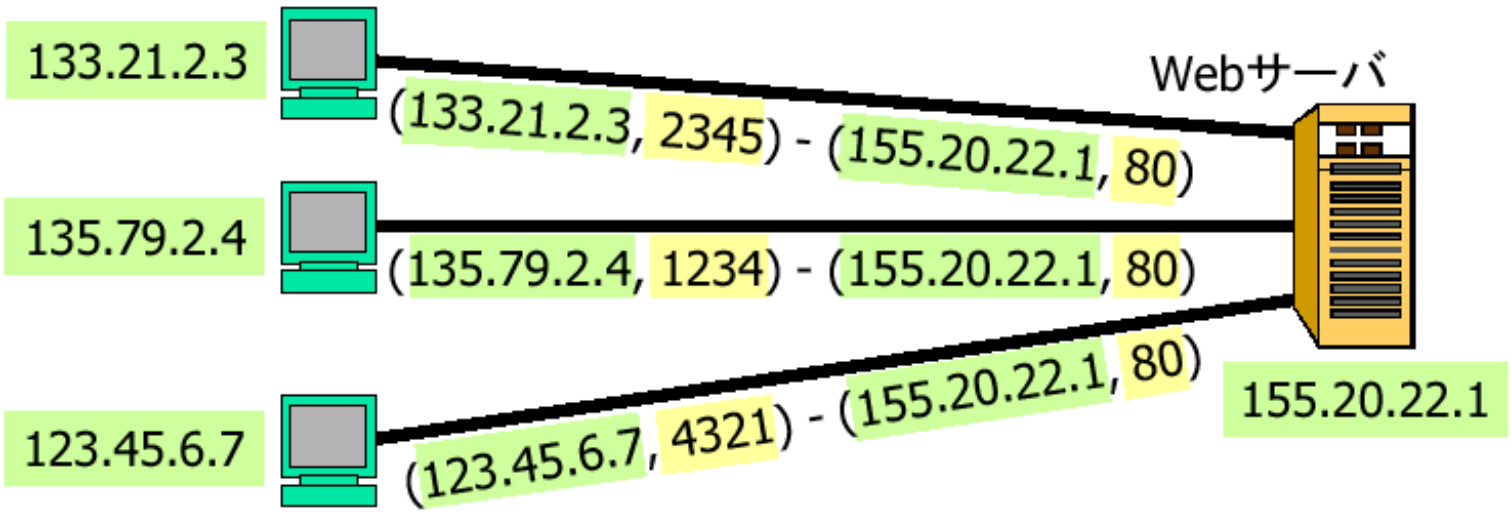
- TCP (Transmission Control Protocol)；end-to-end
  - フロー制御
  - エラー制御 / 再送制御
  - コネクション管理
  - セッションの多重化



TCP Features

- *"Stream"* Oriented Data Transmission
  → Connection確立(*Three-way-handshake*)
- Connection ("Stream") Identifier ＝ *"Socket"*
  {dst_IP_addr, dst_port, src_IP_addr, src_port}
- *"Sequence Number"* ; 32 bits
  → バイト番号：0 ー (2^32-1)
  → 2^32 でSequence NumberがWrapされる
- *"Full-Duplex"*での通信
- Acknowledgement (ACK) ;
  → 次に受信すべきバイト番号(SN)の通知
- エラー回復: セグメント再送(Segment retransmission)
  by *Time-out, Dupilicated-ACK*
- *"Sliding Window Control"* を用いたデータ転送制御
  (*) Window_size ≦ 65,535 Bytes

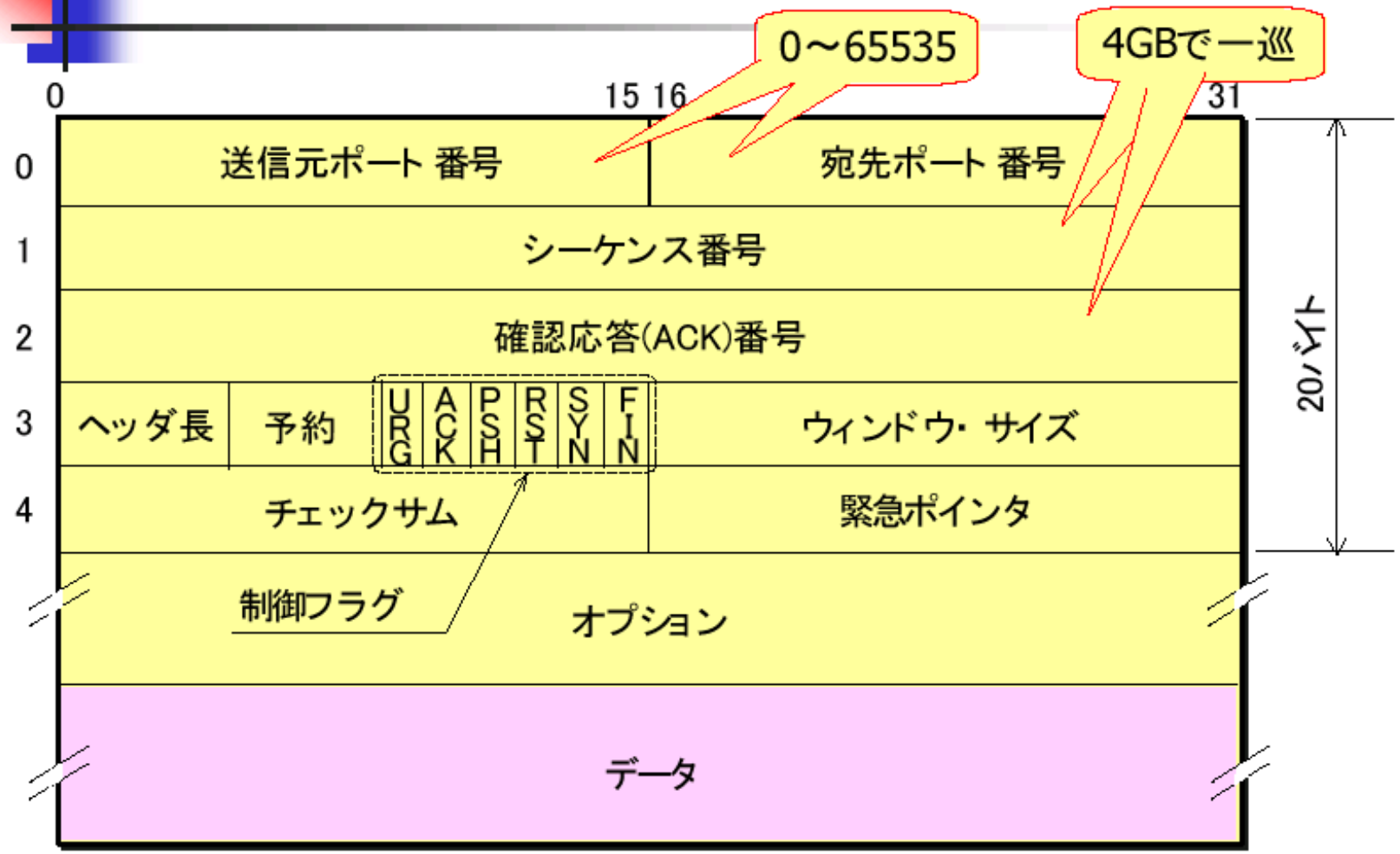
TCPコネクションの多重化と識別



Well-known Portの例

ポート 番号	用途(アプリケーション層プロトコル)
20, 21	FTP: ファイル転送プロトコル(20:データ，21:制御)
25	SMTP: 簡易メール配送プロトコル
53	Domain Name Service (DNS)
80	HTTP: ハイパーテキスト 転送プロトコル

TCPパケット の構成



URG: 緊急ポインタフィールド
ACK: 確認応答フィールド
PSH: 強制転送機能
RST: コネクションリセット
SYN: コネクション確立要求
FIN: 転送データ終了

TCP Header Format

UR : Urgent Pointer Field Significant (URG)
AK : Acknowledgement Field Significant (ACK)
PH : Push Function
RT : Reset the Connection
SY : Synchronize Sequence Numbers (SYN)
FN : No More Data From Sender (FIN)

TCP Port Allocation（RFC1700)

1. Well-Known Ports ; 0 - 1,023
2. Registered Ports ; 1,024 - 49,151
3. Dynamic and/or Private Ports ; 49,152 - 65,535

最新情報：
ftp://ftp.isi.edu/in-notes/iana/assignments/port-numbers

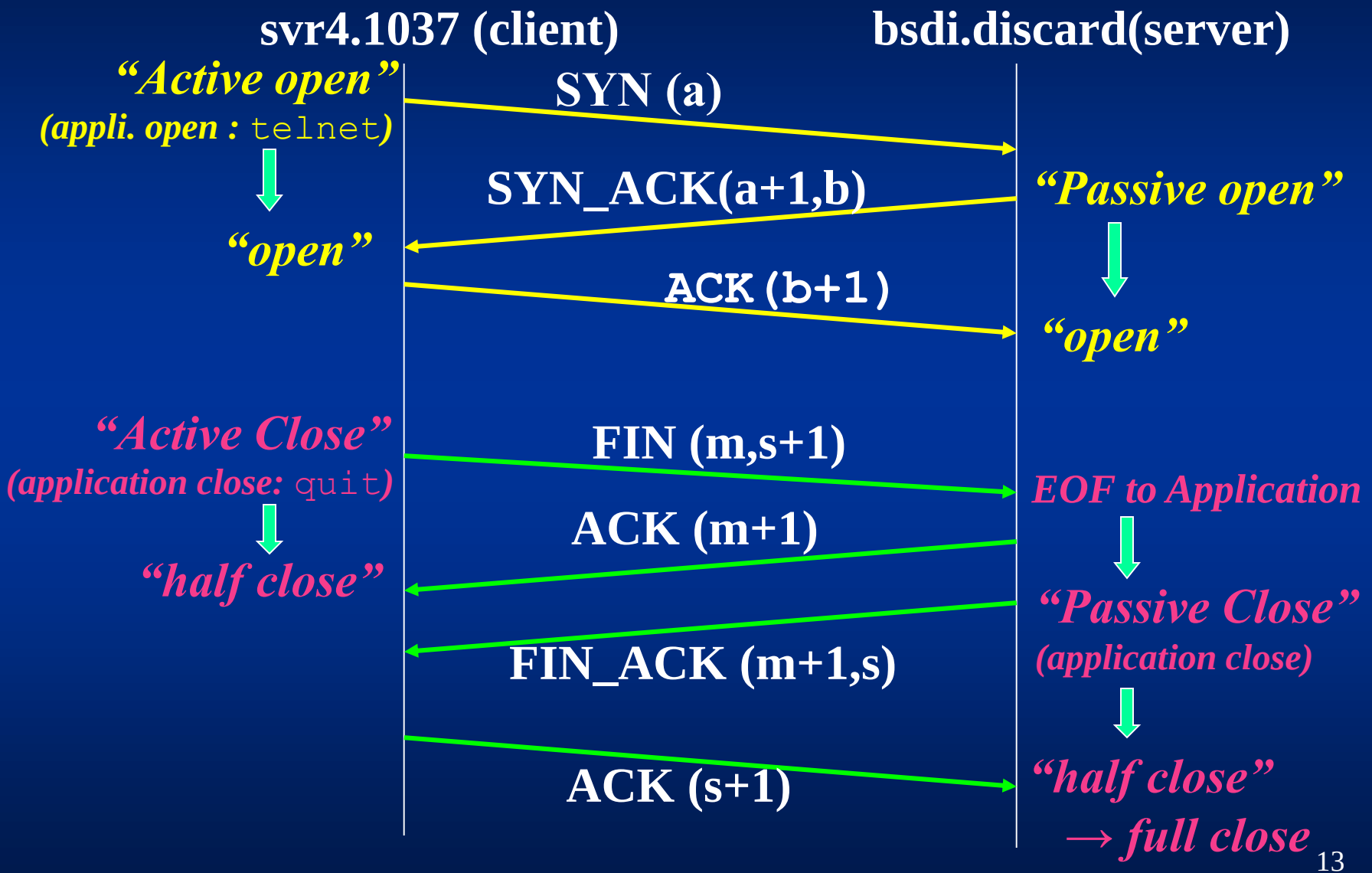
TCP Well-Known Ports

Port Number	Keyword	Application
5	rje	Remote Job Entry
20	ftp-data	File Transfer [Default data]
21	ftp	File Transfer [Control]
23	telnet	Telnet
25	smtp	Simple Management Protocol
39	rlp	Resource Location Protocol
53	domain	Domain Name Server
63	whois++	Whois++
67	bootp	Bootstrap Protocol Server
69	tftp	Trivial File Transfer
70	gopher	Gopher
79	finger	Finger
80	http	World Wide Web HTTP
110	pop3	Post Office Protocol - Version 3
111	sunrpc	SUN Remote Procedure Call
119	nntp	Network News Transfer Protocol

TCP Well-Known Ports

Port Number	Keyword	Application
123	ntp	Network Time Protocol
137	netbios-ns	NetBIOS Name Service
138	netbios-dgm	NetBIOS Datagram Service
139	netbios-ssn	NetBIOS Session Service
179	bgp	Border Gateway Protocol (BGP)
202	at-nbp	AppleTalk Name Binding Protocol
213	ipx	IPX
220	imap3	IMAP3 (Interactive Mail Access Protocol)
396	netware-ip	Novell Netware over IP
540	uucp	uucp daemon
546	dhcpv6-client	DHCPv6 Client
547	dhcpv6-server	DHCPv6 Server
560	rmonitor	remote monitor daemon

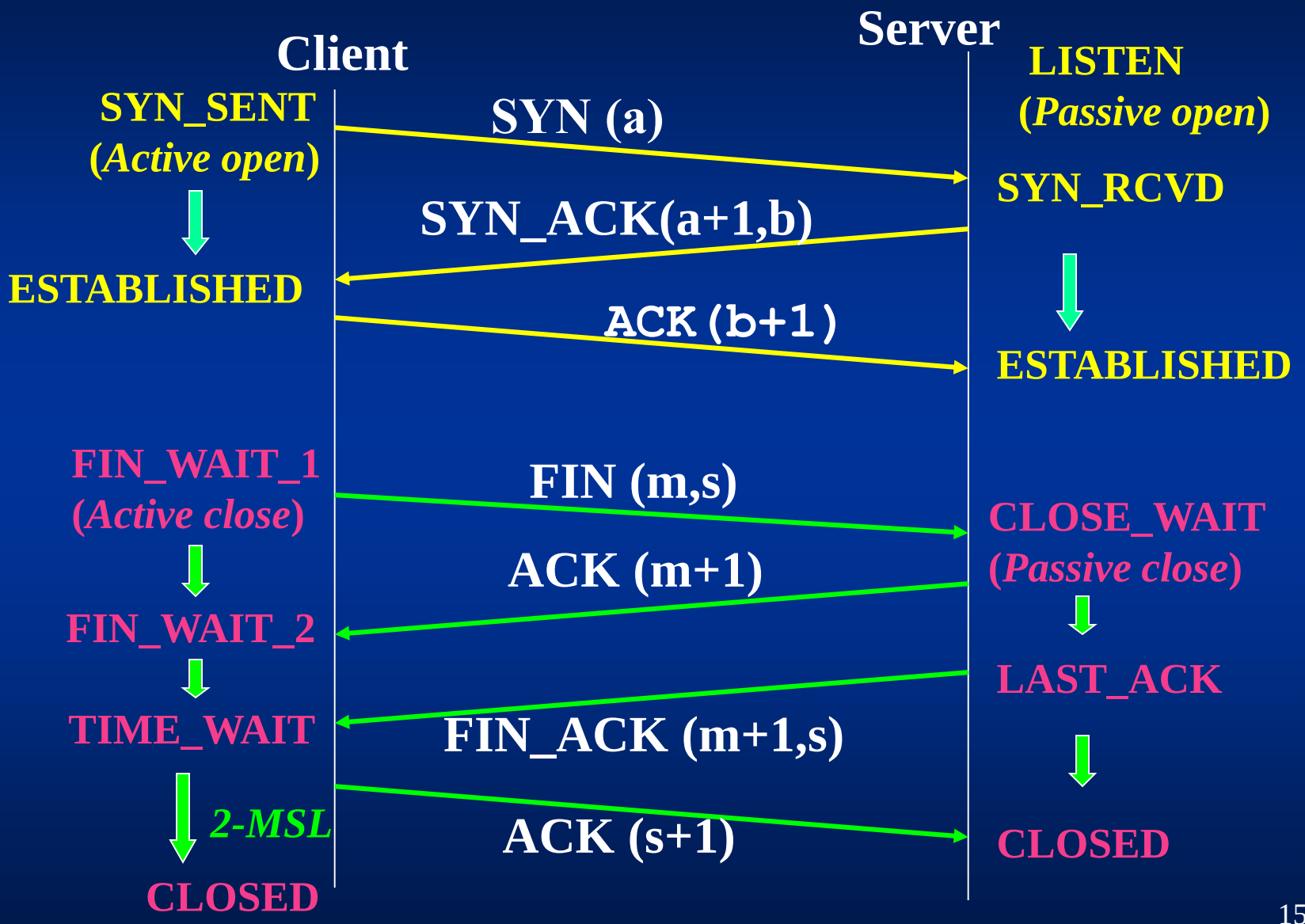
TCP Connection確立/解放



Three Way Handshalking

- エラー が発生する可能性が存在する。
- 2つの ノードの間での それぞれのノードが持つ 情報の「同期」を確立させたい。
- 3つのパケット(＝メッセージ)の 送受信で、情報の同期が確保・保証される(必要十分条件)。

TCP Connection確立/開放



TCP Connection確立/開放

Log on the console;

```
svr4% telnet bsdi discard     #port="9" (server discard packet)
Trying 140.252.13.35
Connected to bsdi.
Escape character is '^]'.
^]
telnet> quit
Connection closed.
```

tcpdump output

```
1 0.0                 svr4.1037 > bsdi.discard: S 14155.14155(0)
                                                win 4096 <mss 1024>
2 0.024 (0.0024)      bsdi.discard > svr4.1037: S 18239.18239(0)
                                  ack 14156 win 4096 <mss 1024>
3 0.007 (0.0048)      svr4.1037 > bsdi.discard: . ack 18240 win 4096
4 4.155 (4.1482)      svr4.1037 > bsdi.discard: F 14156:14156(0)
                                  ack 18240 win 4096
5 4.158 (0.0013)      bsdi.discard > svr4.1037: . ack 14157 win 4096
6 4.159 (0.0014)      bsdi.discard > svr4.1037: F 18240.18240(0)
                                  ack 14157 win 4096
7 4.189 (0.0225)      svr4.1037 > bsdi.discard: . ack 18241 win 4096
```

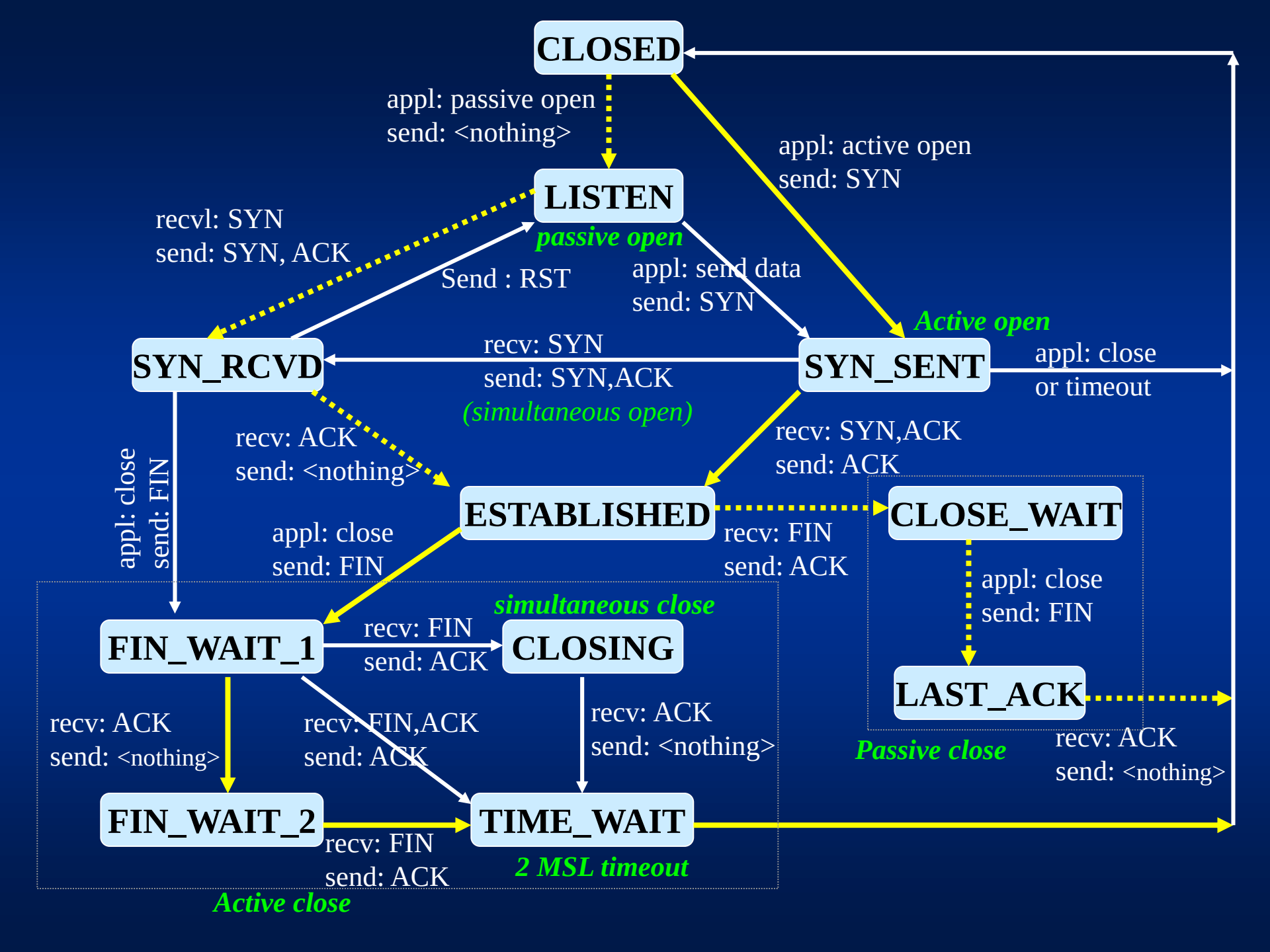
TCP Connection確立/開放

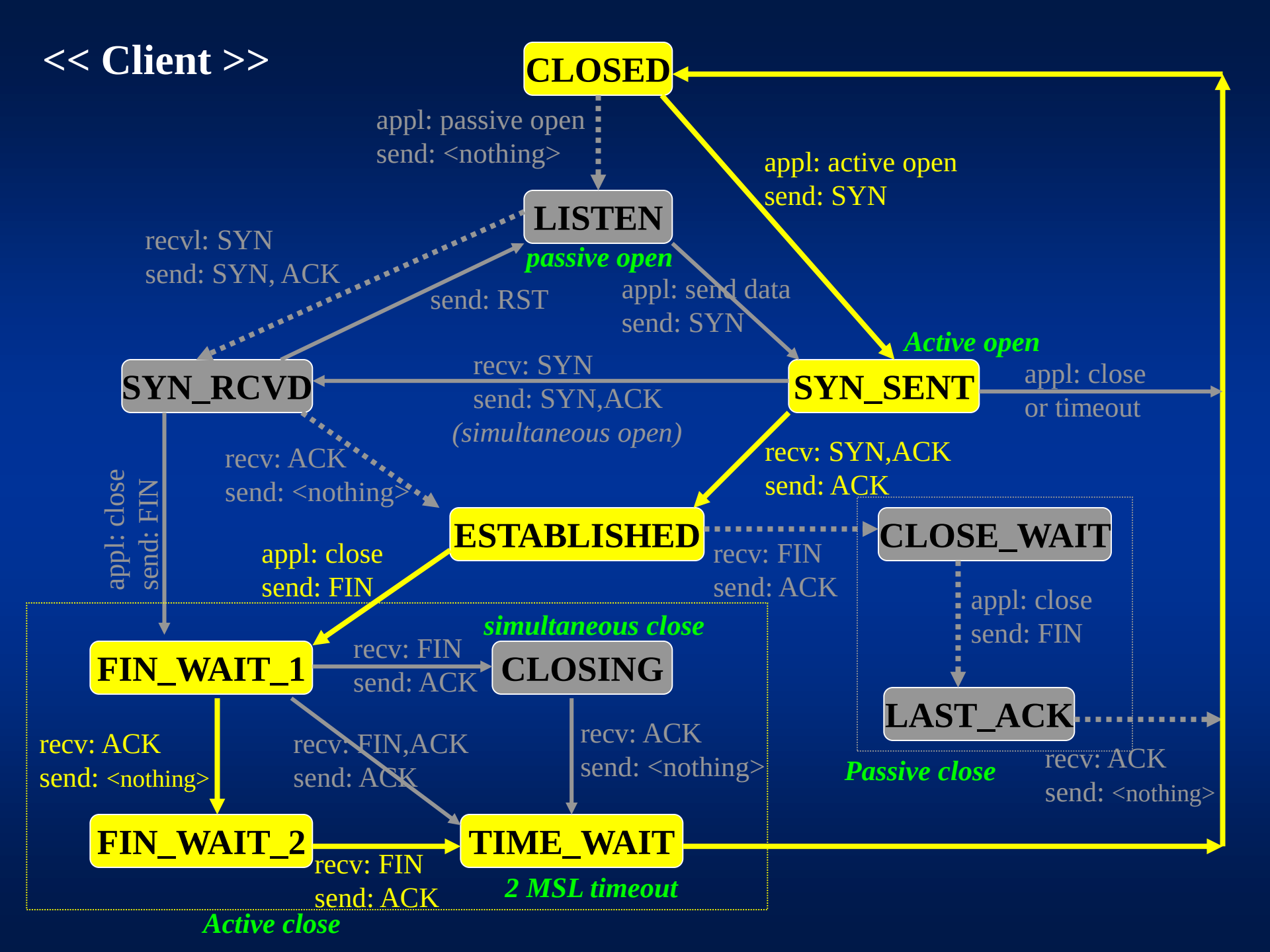
tcpdump output

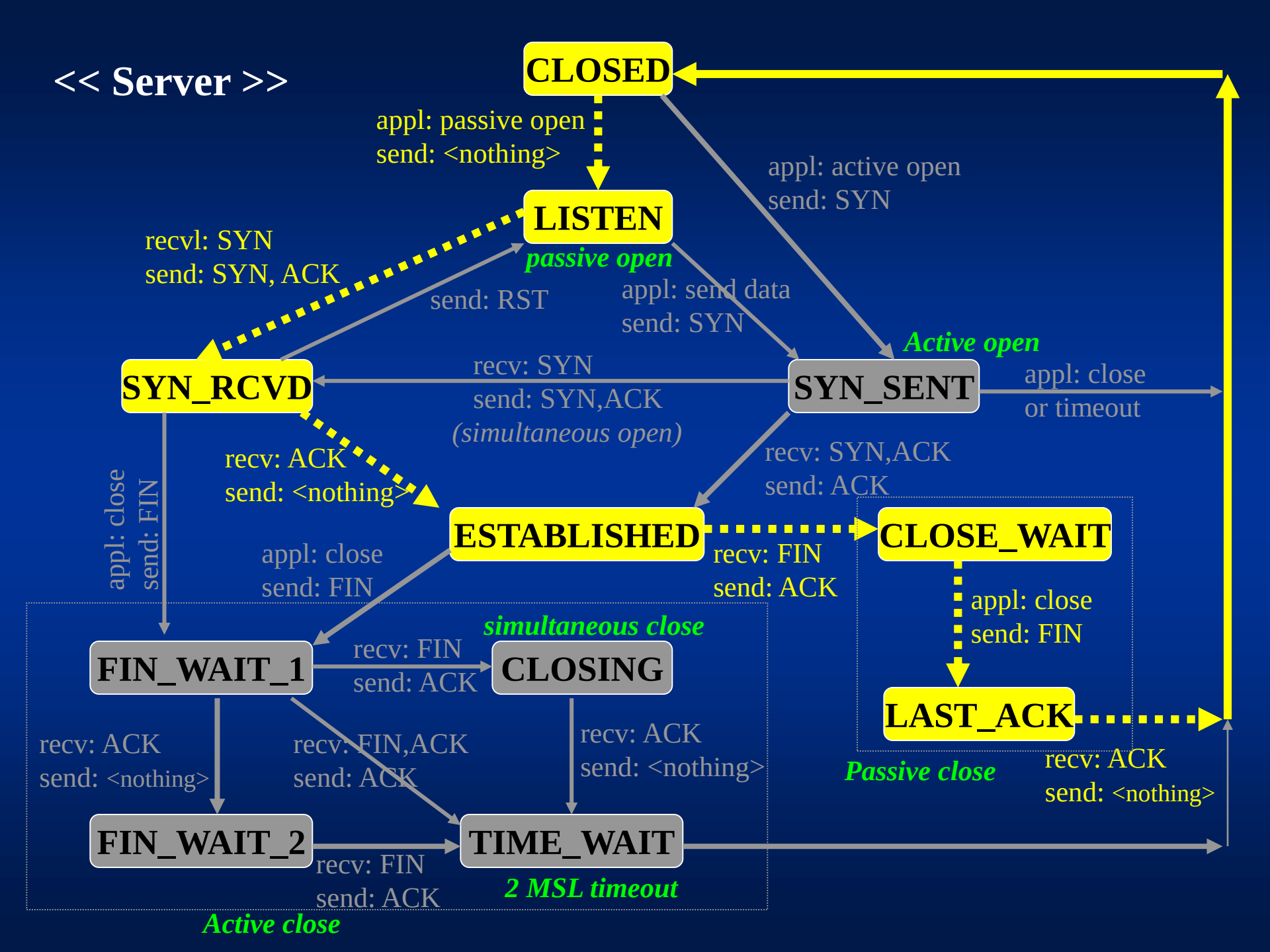
```
 1 0.0                  svr4.1037 > bsdi.discard: S 14155.14155(0)
                                                  win 4096 <mss 1024>
 2 0.024 (0.0024)       bsdi.discard > svr4.1037: S 18239.18239(0)
                                                 ack 14156 win 4096 <mss 1024>
 3 0.007 (0.0048)       svr4.1037 > bsdi.discard: . ack 18240 win 4096
 4 4.155 (4.1482)       svr4.1037 > bsdi.discard: F 14156:14156(0)
                                                 ack 18240 win 4096
 5 4.158 (0.0013)       bsdi.discard > svr4.1037: . ack 14157 win 4096
 6 4.159 (0.0014)       bsdi.discard > svr4.1037: F 18240.18240(0)
                                                 ack 14157 win 4096
 7 4.189 (0.0225)       svr4.1037 > bsdi.discard: . ack 18241 win 4096
```

[意味]

```
source.port > destination.port : flags SN_begin.SN_end(data_size)
     flags :  S = SYN ; Synchronize sequence_number(SN)
              F = FIN ; Finish data transmission
              R = RST ; Reset connection
              P = PSH ; push data to receiving process asap
              . =     ; none of above four flags is on
     SN_end = SN_begin + data_size
win 4096 ; window size is 4096
mss 1024 ; maximum segment size is 1024 bytes
```


CLOSED
appl: passive open
send: <nothing>
appl: active open
send: SYN
LISTEN
passive open
recvl: SYN
send: SYN, ACK
Send : RST
appl: send data
send: SYN
Active open
SYN_RCVD
recv: SYN
send: SYN,ACK
(simultaneous open)
SYN_SENT
appl: close
or timeout
recv: ACK
send: <nothing>
recv: SYN,ACK
send: ACK
appl: close
send: FIN
ESTABLISHED
CLOSE_WAIT
appl: close
send: FIN
recv: FIN
send: ACK
simultaneous close
FIN_WAIT_1
recv: FIN
send: ACK
CLOSING
LAST_ACK
recv: ACK
send: <nothing>
recv: FIN,ACK
send: ACK
recv: ACK
send: <nothing>
Passive close
recv: ACK
send: <nothing>
FIN_WAIT_2
TIME_WAIT
recv: FIN
send: ACK
2 MSL timeout
Active close


<< Client >>
CLOSED
appl: passive open
send: <nothing>
appl: active open
send: SYN
LISTEN
passive open
recvl: SYN
send: SYN, ACK
send: RST
appl: send data
send: SYN
Active open
SYN_RCVD
recv: SYN
send: SYN,ACK
(simultaneous open)
SYN_SENT
appl: close
or timeout
recv: ACK
send: <nothing>
recv: SYN,ACK
send: ACK
appl: close
send: FIN
ESTABLISHED
CLOSE_WAIT
appl: close
send: FIN
recv: FIN
send: ACK
appl: close
send: FIN
simultaneous close
FIN_WAIT_1
recv: FIN
send: ACK
CLOSING
LAST_ACK
recv: ACK
send: <nothing>
recv: FIN,ACK
send: ACK
recv: ACK
send: <nothing>
recv: ACK
send: <nothing>
Passive close
FIN_WAIT_2
TIME_WAIT
recv: FIN
send: ACK
2 MSL timeout
Active close


<< Server >>
CLOSED
appl: passive open
send: <nothing>
LISTEN
passive open
appl: active open
send: SYN
recvl: SYN
send: SYN, ACK
send: RST
appl: send data
send: SYN
Active open
SYN_RCVD
recv: SYN
send: SYN,ACK
(simultaneous open)
SYN_SENT
appl: close
or timeout
recv: ACK
send: <nothing>
recv: SYN,ACK
send: ACK
appl: close
send: FIN
ESTABLISHED
CLOSE_WAIT
appl: close
send: FIN
recv: FIN
send: ACK
simultaneous close
FIN_WAIT_1
recv: FIN
send: ACK
CLOSING
LAST_ACK
recv: ACK
send: <nothing>
recv: ACK
send: <nothing>
recv: FIN,ACK
send: ACK
recv: ACK
send: <nothing>
Passive close
FIN_WAIT_2
TIME_WAIT
recv: FIN
send: ACK
2 MSL timeout
Active close

フロー制御

- バッファオーバーフロー
  - 受信側アプリが読み込む以上の速度で，送信側がデータを送信

  ⇒ウィンドウ制御：受信側がウィンドウサイズを通知
  - ルータにパケットが集中(輻輳)

  ⇒輻輳ウィンドウ：送信側がウィンドウサイズを決定
  - 小さい方のウィンドウサイズを採用

誤りのないデータ転送

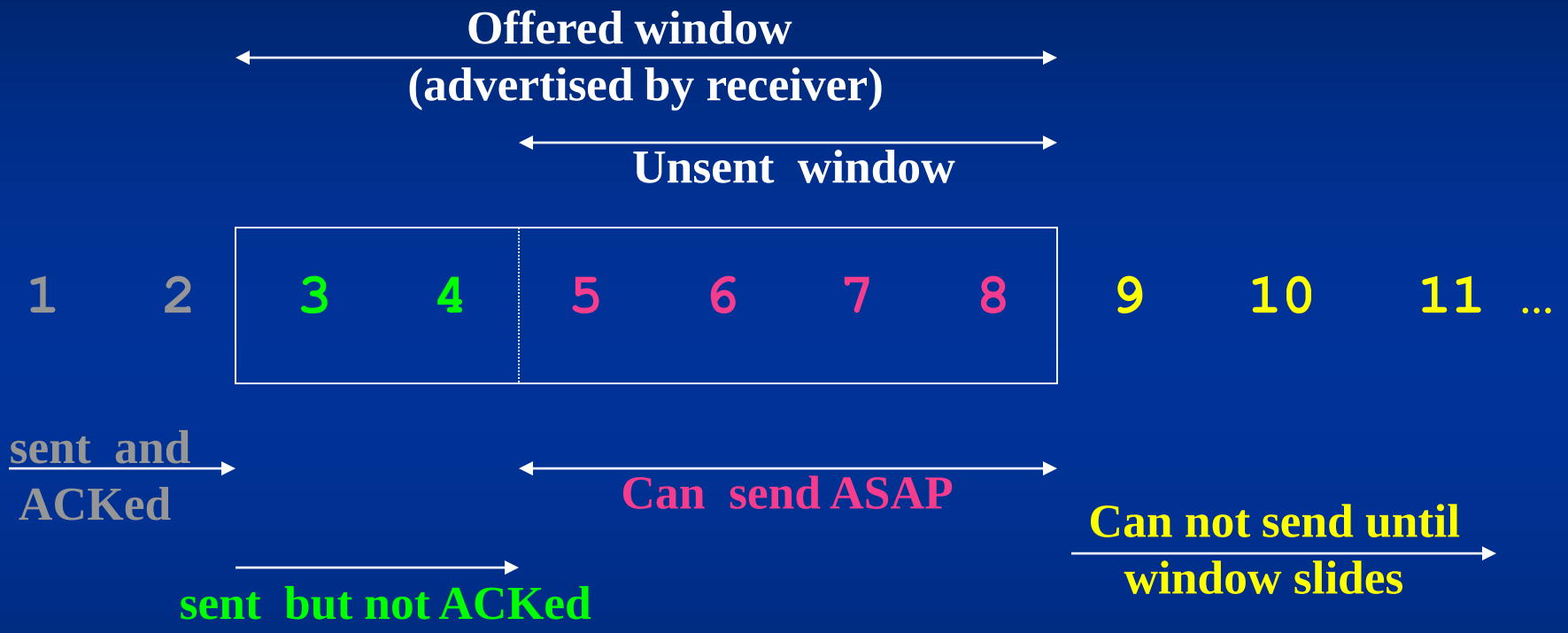
- パケットが紛失したり誤ったりしたら
  - 再送(Resend)して、もとにもどす。
- 正しく受信できたかの確認のメッセージ(ACK; Acknowledge)を送信(From dst➔src)
  - とても原始的な手順では、、、、速度が出ない。。
  - 2つの改善手法
    - 大きなパケット長: 最大でも 帯域幅の 1/3 まで。。。
    - パイプラインで パケットを転送

TCP Bulk Data Transmission
- Sliding Window -

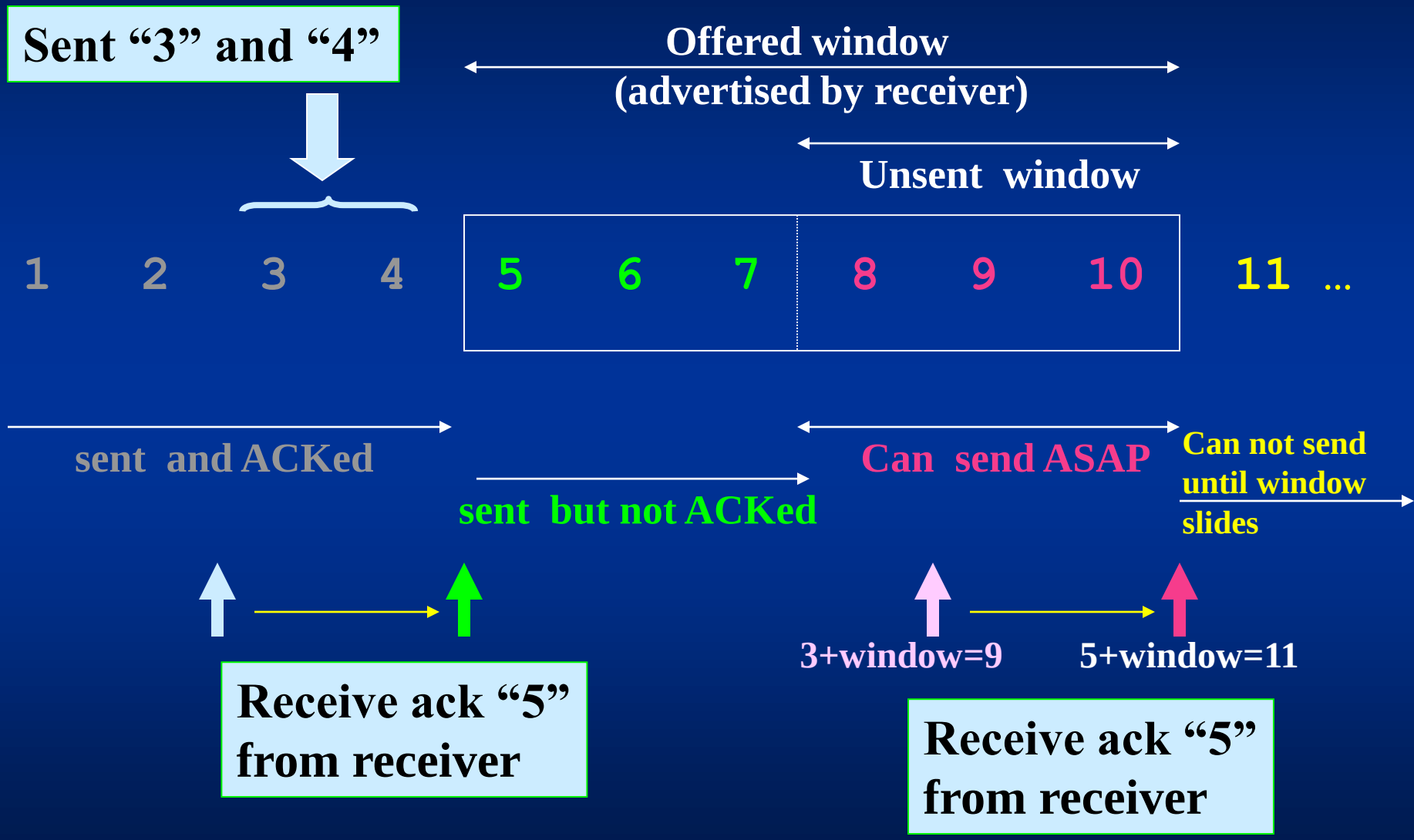
- Window制御を用いたパケット転送
  - ①Sliding Window (Receiver設定)
  - ②Congestion Window(Sender設定)

(1) ACKなしにwindow数のパケットを転送
(2) ACKのAggregation(ACKパケットの減少)
(3) Receiver側によるwindow幅の制御
(4) ACK受信でwindowをスライドさせる

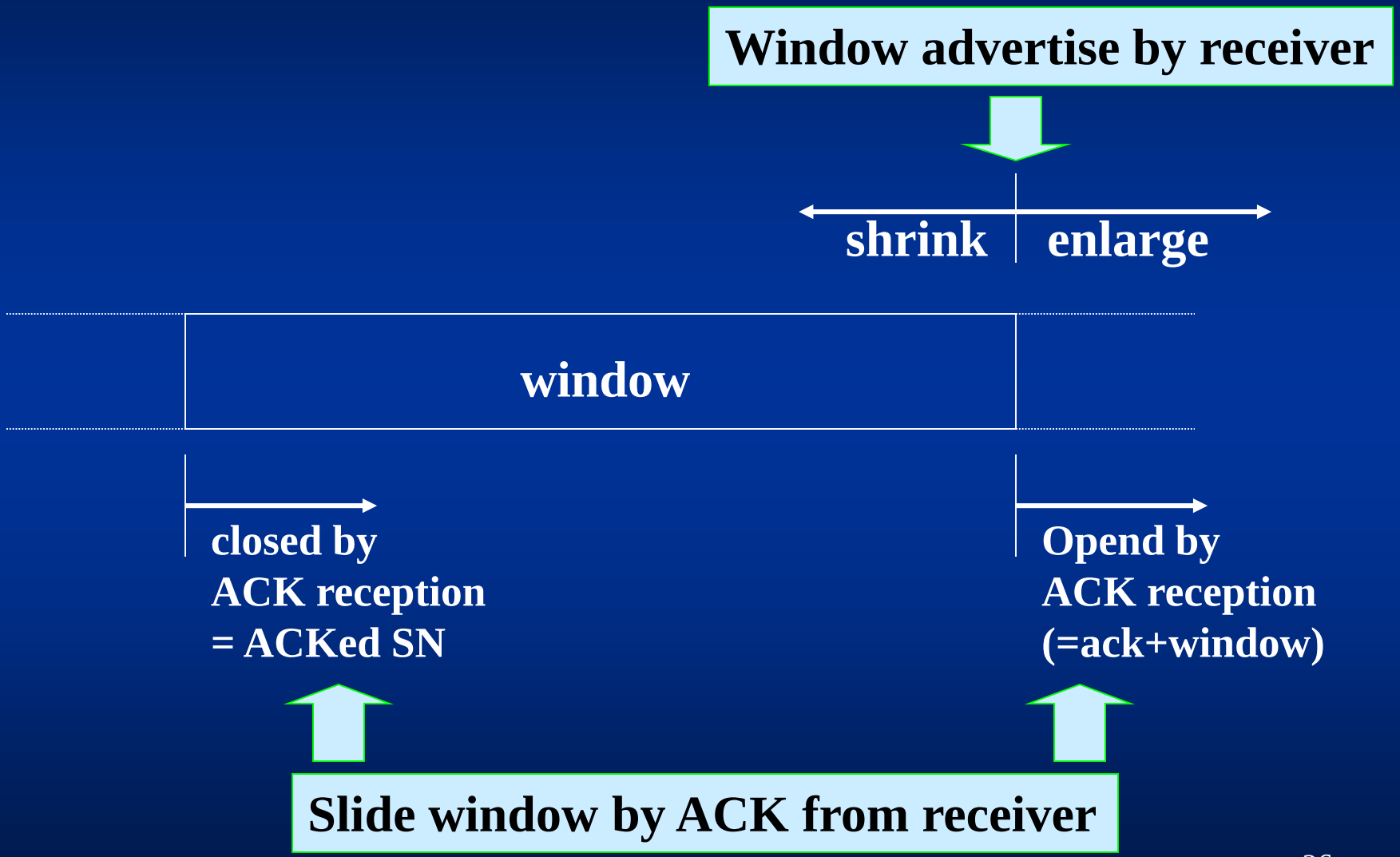
TCP Sliding Window



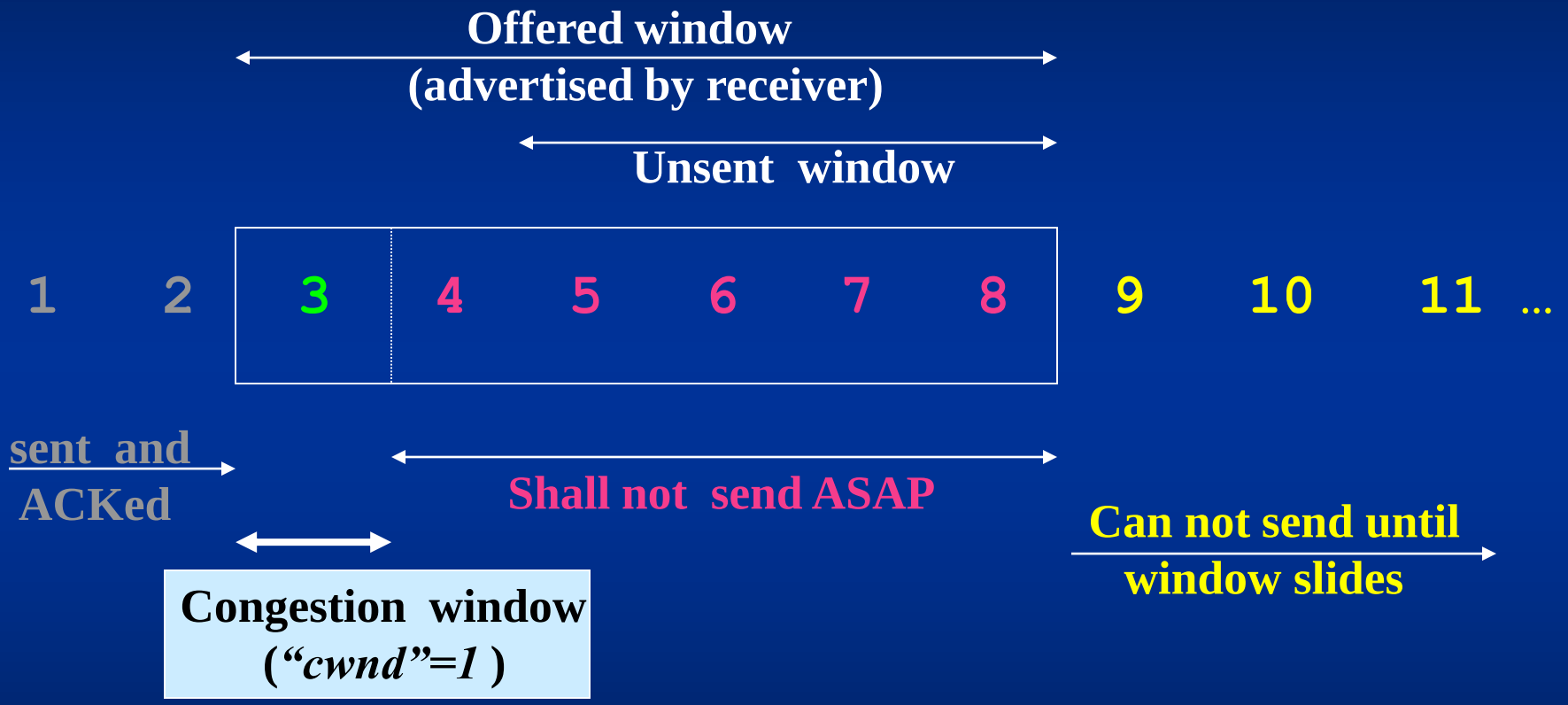
TCP Sliding Window



TCP Sliding Window

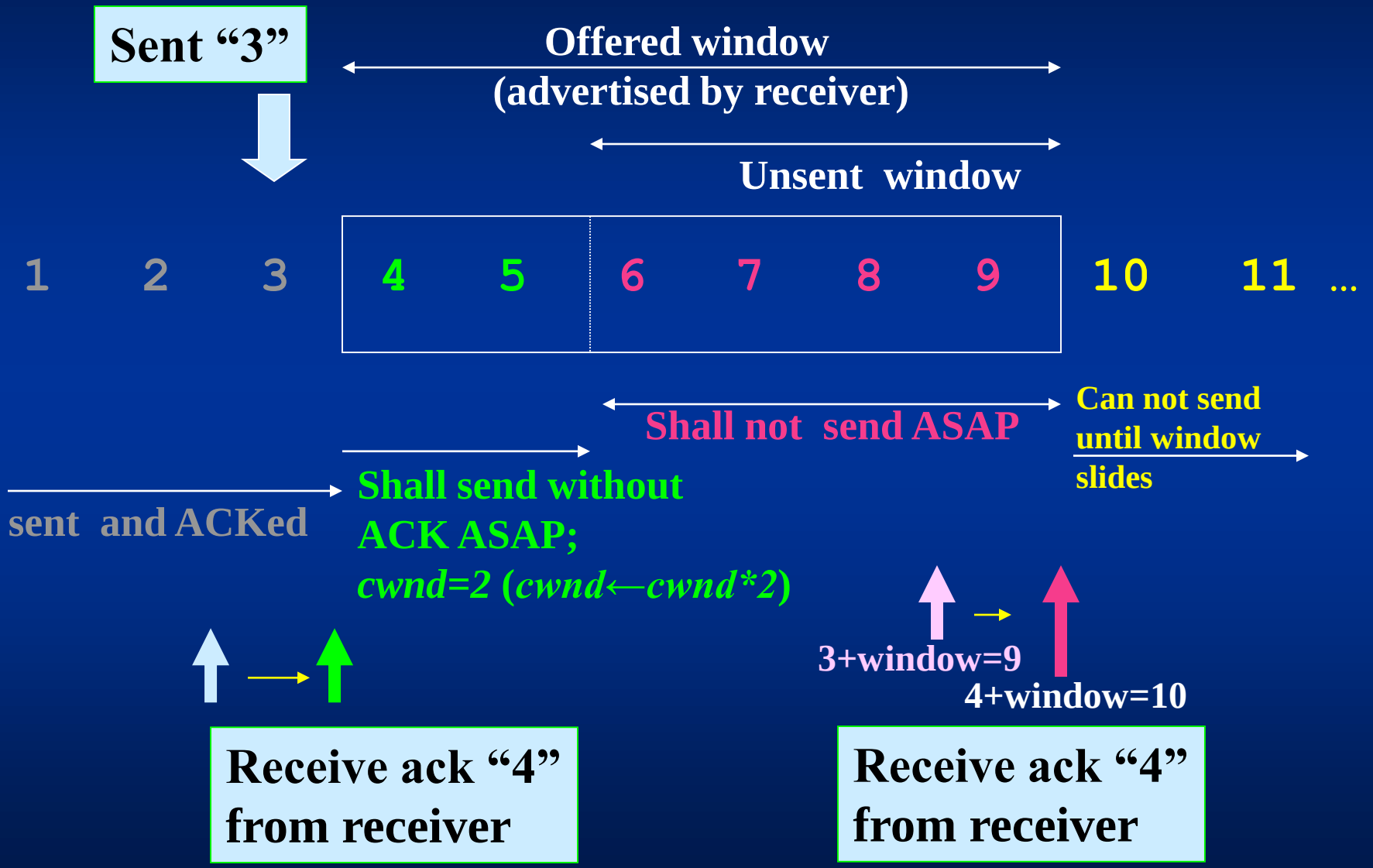


TCP Congestion Window



→ sent but not ACKed

TCP Congestion Window



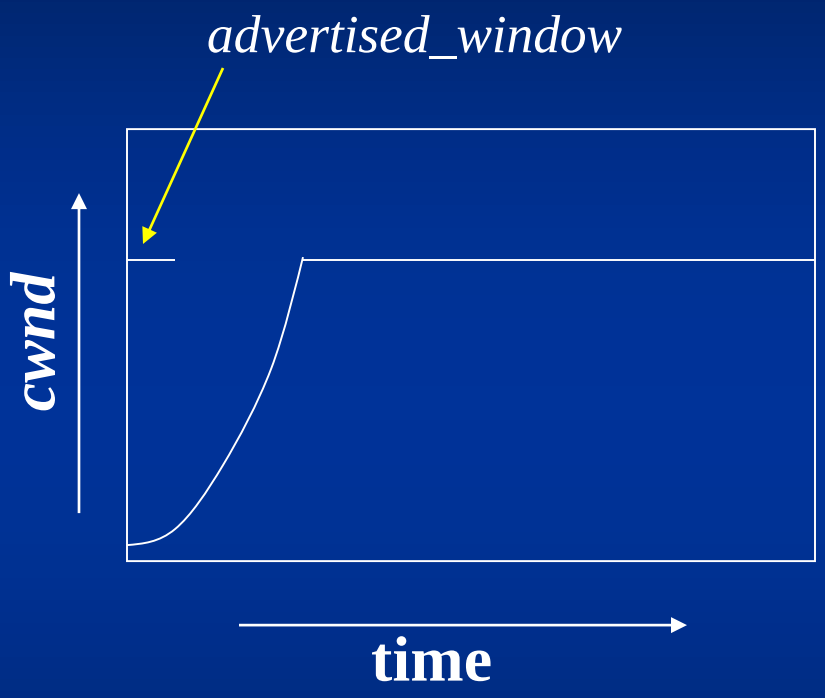
TCP Congestion Window

- ***Slow Start*** **Policy** ***(cwnd*** **; exponential increase)**

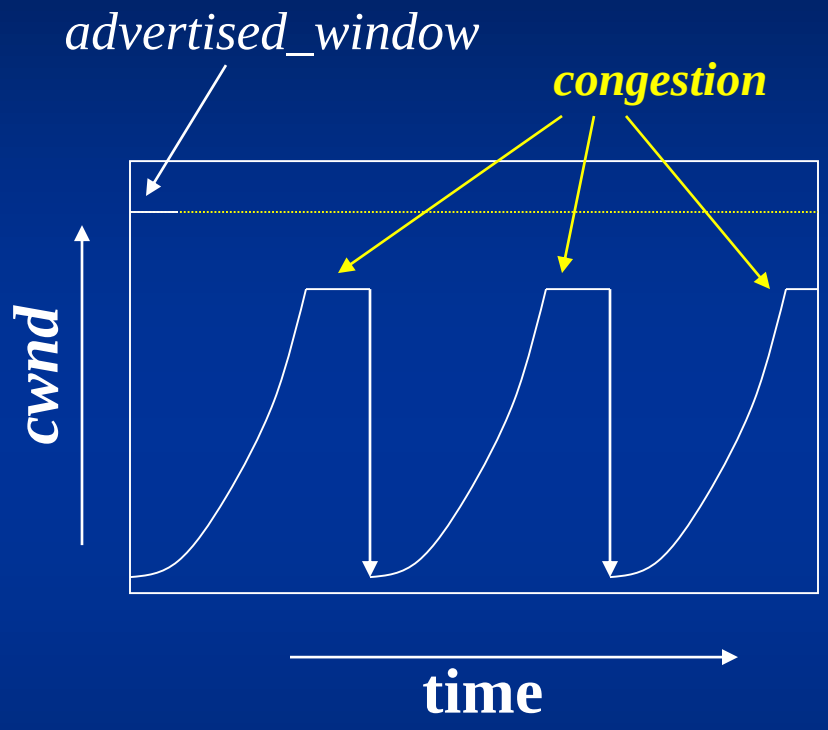
```
cwnd = 1 ;
for (セグメント転送)
 {
  for (not congestion)
    {
      if (セグメント転送ACK受信)
        { cwnd = cnwd +1 }
 cwnd = 1
 }
```

(*)注意：Congestion Avoidance では若干異なる。
SenderがLocal に制御することなので、変えることが容易に可能

TCP Congestion Window



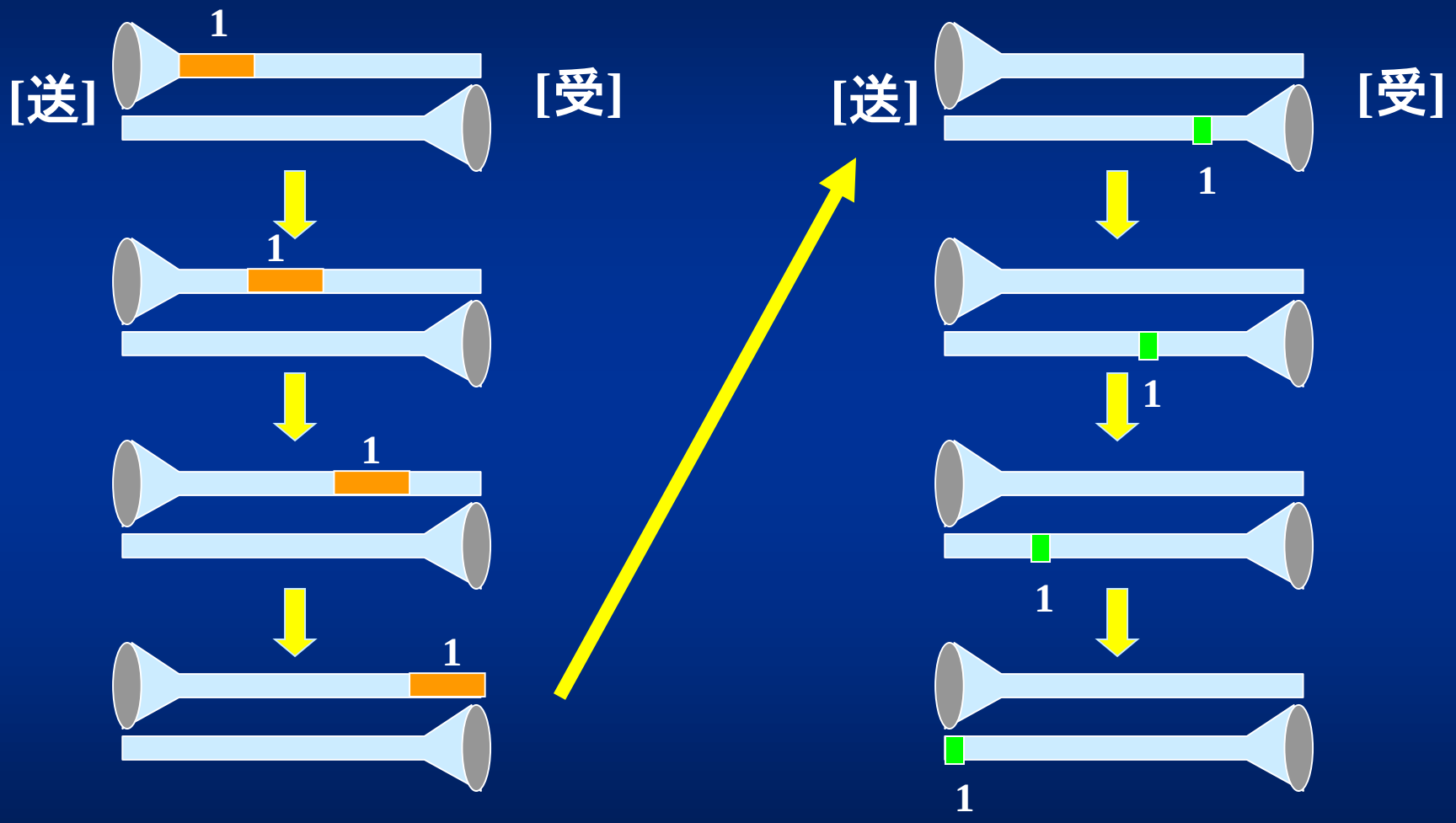
< Congestionなしの場合 >



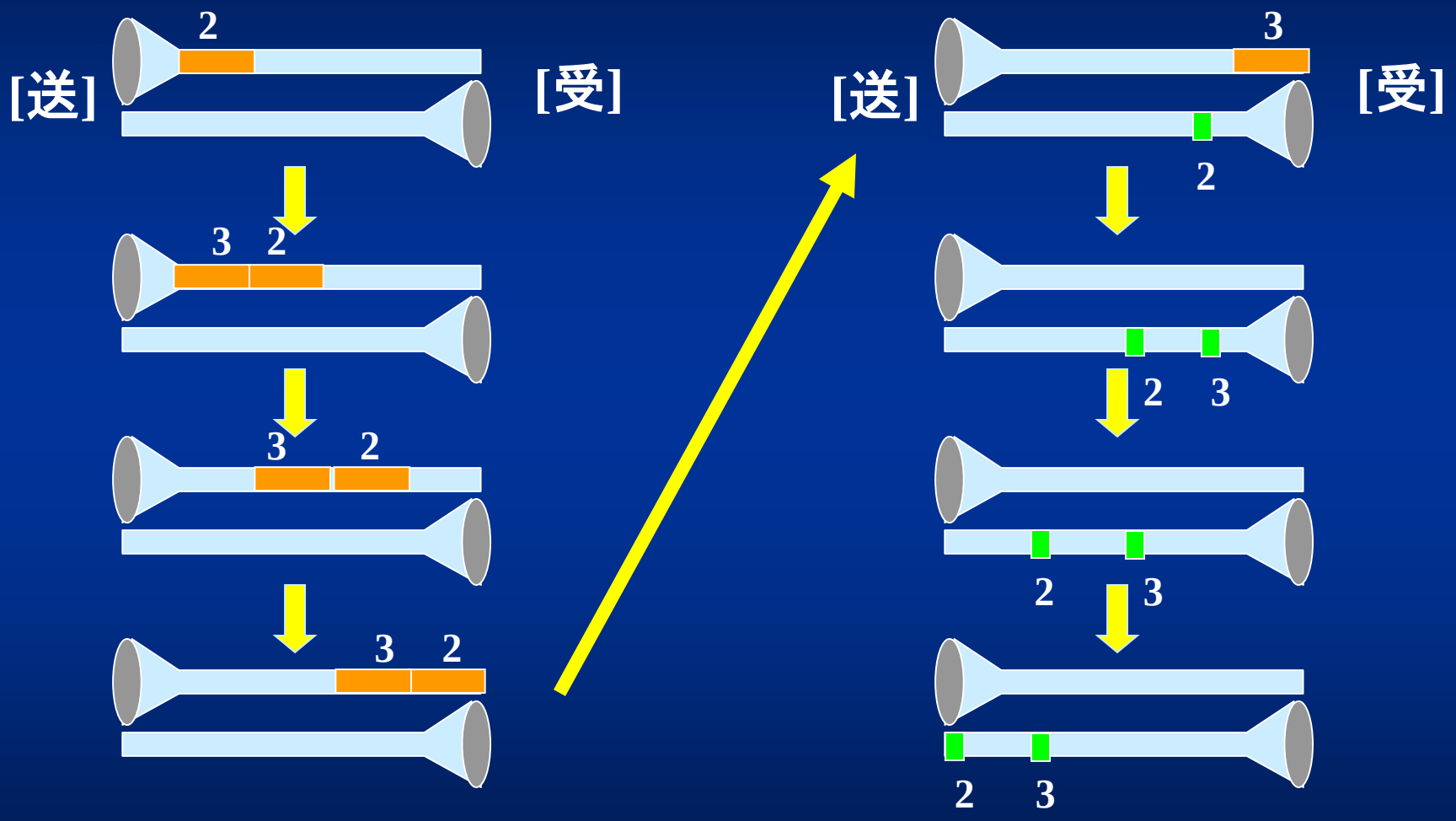
< Congestion経験の場合 >

(*) Duplicated ACKを使用せず

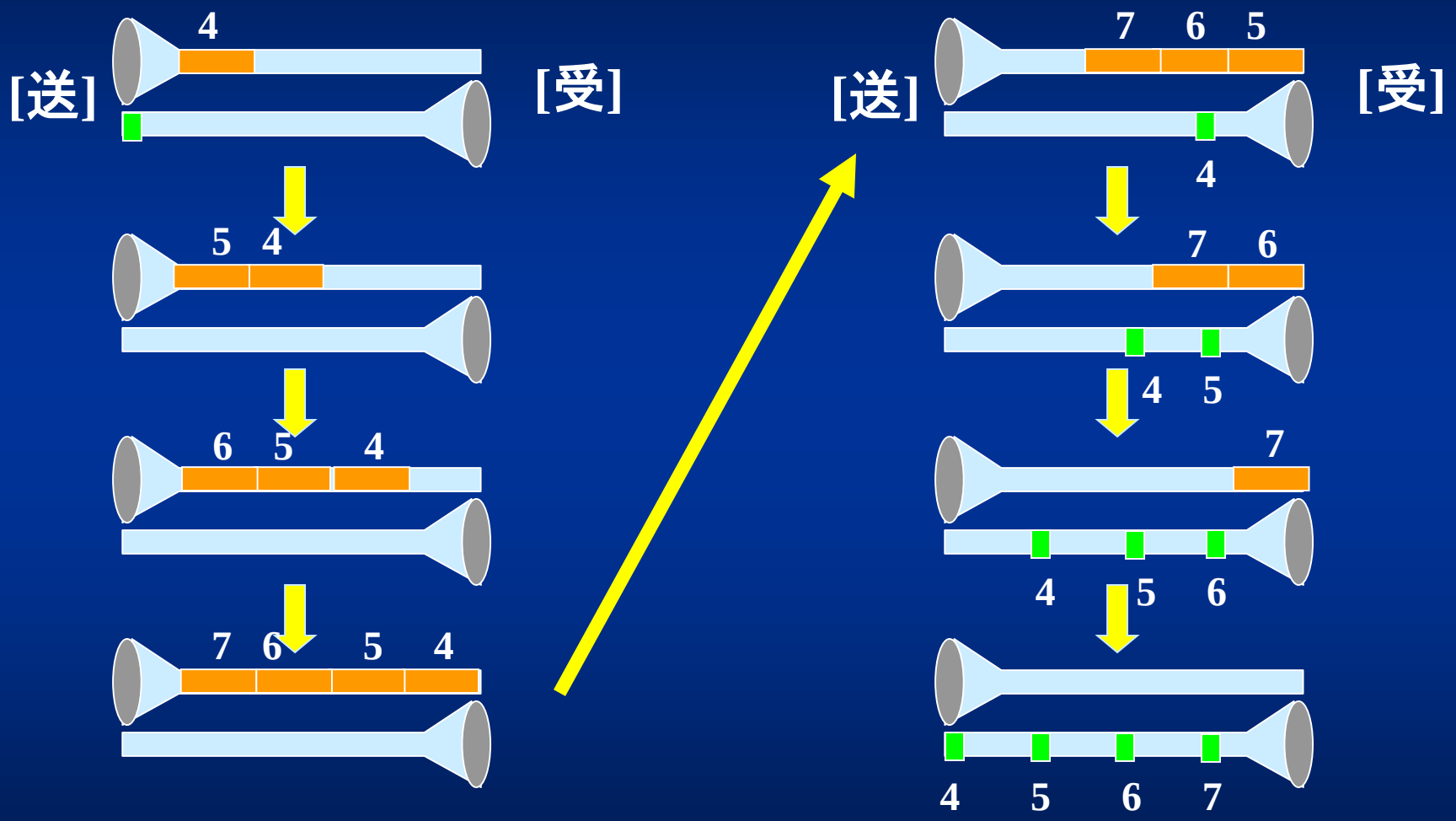
TCP Congestion Window(1)



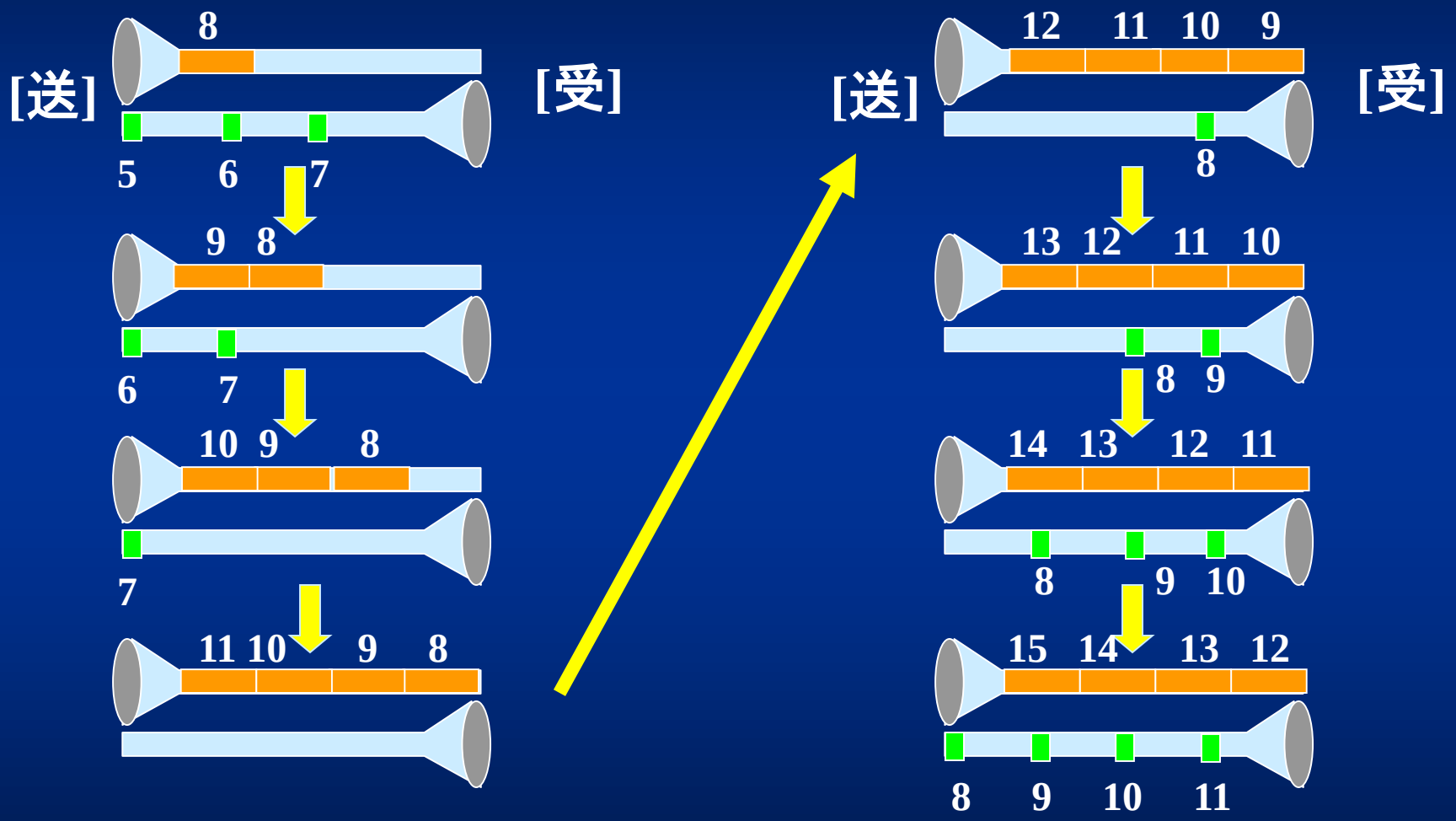
TCP Congestion Window(2)



TCP Congestion Window(3)



TCP Congestion Window(4)



必要なウィンドー幅 ≧ BWxRTT

Congestion Window Control

[目的]
cwndの大きな振動を防ぎ、
適切なcwndで運用する

[1] cwndの制御
(i) ssthresh以下のcwndサイズ
→ Exponential increase
(slow start)
(ii) ssthresh以上のcwndサイズ
→ Liner increase
(congestion avoidance)
[2] ssthreshの制御
(i) Timeout；goto “1”
(ii) Duplicated-ACK；1/2

```
cwnd=1;  ssthresh=64KB;
for ()
{
 if (“Timeout”)
  { cwnd=1;
    ssthresh = cwnd/2;  }
 if (“duplicated ACK”)
  { ssthresh=cwnd / 2;
    cwnd=ssthresh;       }

 if (cwnd ≦ ssthresh)
  { slow_start;
    /* exponential */    }
 else
  { congestion_avoidance;
    /* liner */          }
}
```

Congestion Window Control (続)

- ICMP 制御メッセージ

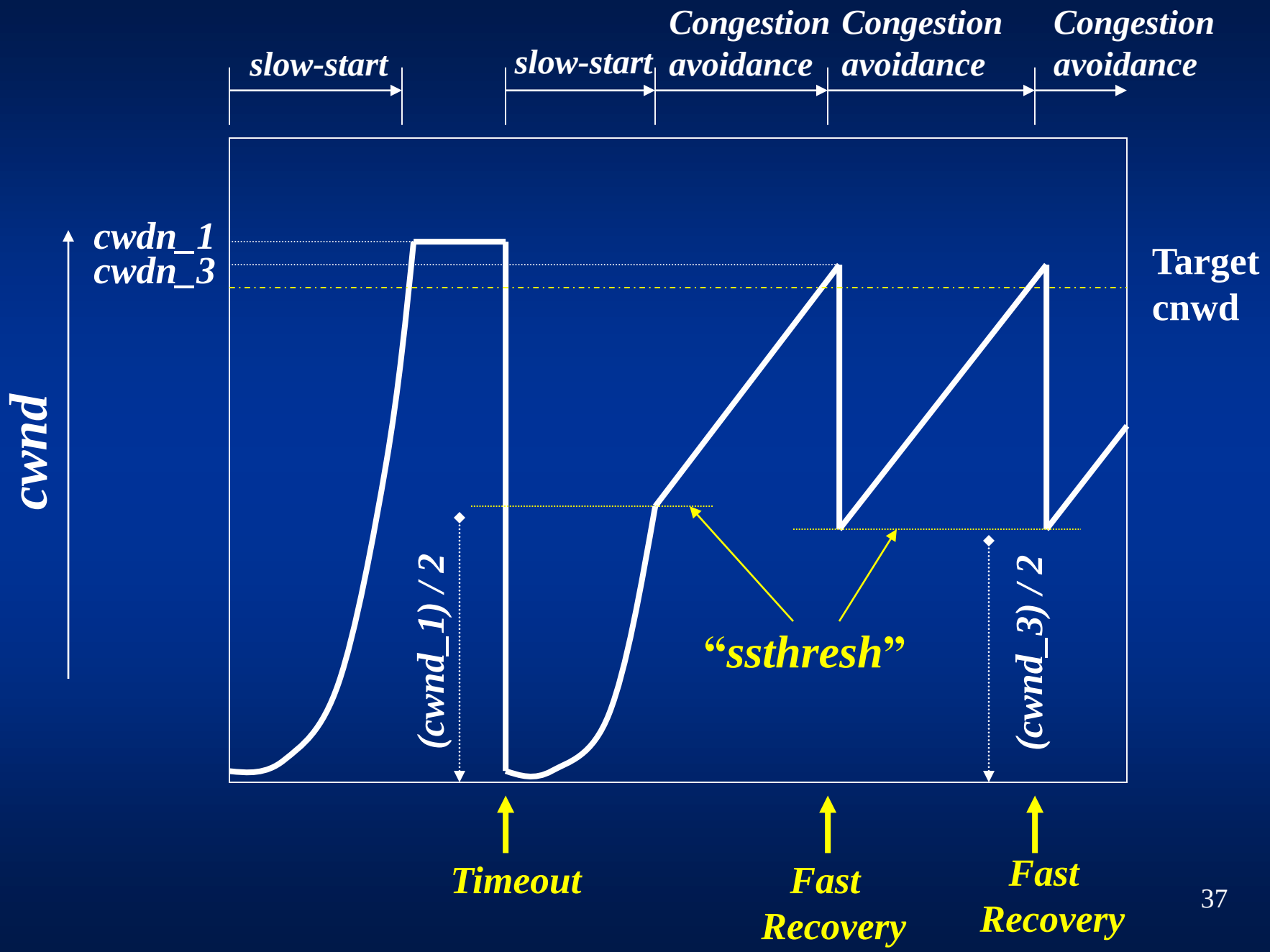
(1) ICMP Source Quench

→ cwnd = 1 ;

ssthresh = as is ;

(2) Host unreachable

→ No Action ;


slow-start
slow-start
Congestion avoidance
Congestion avoidance
Congestion avoidance
cwdn_1
cwdn_3
cwnd
Target cnwd
(cwnd_1) / 2
(cwnd_3) / 2
"ssthresh"
Timeout
Fast Recovery
Fast Recovery

Window Scaling for Long Fat Pipe
- RFC1323 -

Network	Bandwidth（bps）	RTT（ms）	BWxRTT（B）
Ethernet	10.000 M	3	3,750
T1（大陸間）	1.544 M	60	11,580
T1（衛星）	1,544 M	500	96,500
T3（大陸間）	45,000 M	60	337,500
OC12（大陸間）	2,400,000 M	60	7,500,000

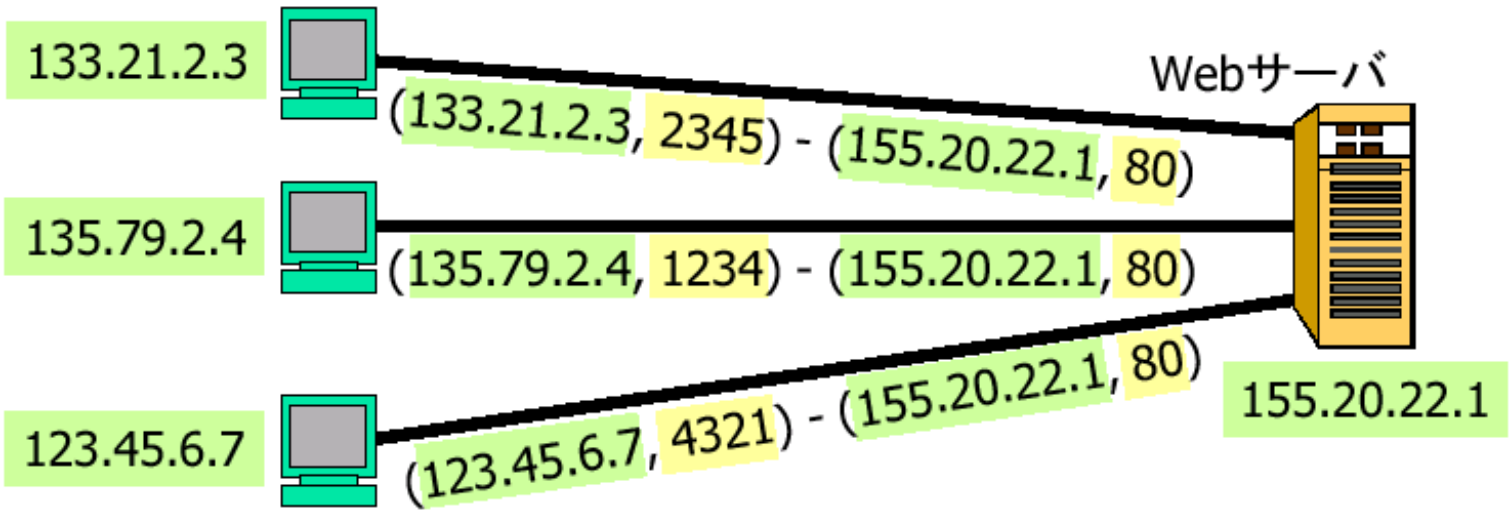
- Max. Window Size ； 2^(16) Bytes = 64KB
  → Window Scaling ; “wscale”
  wscale=n → 64 x 2^(n) windowサイズ

64KB だと、どのくらいになる？

RTT (msec)	Throughput
200	2.56Mbps
20	25.60 Mbps
2	256.00 Mbps

- どうすれば、高速化できるか?
  1. Window-Scale Option (前述)
  2. 複数のTCPパイプを利用
     - ✓ TCP port-ID を複数利用する。

TCPコネクションの多重化と識別

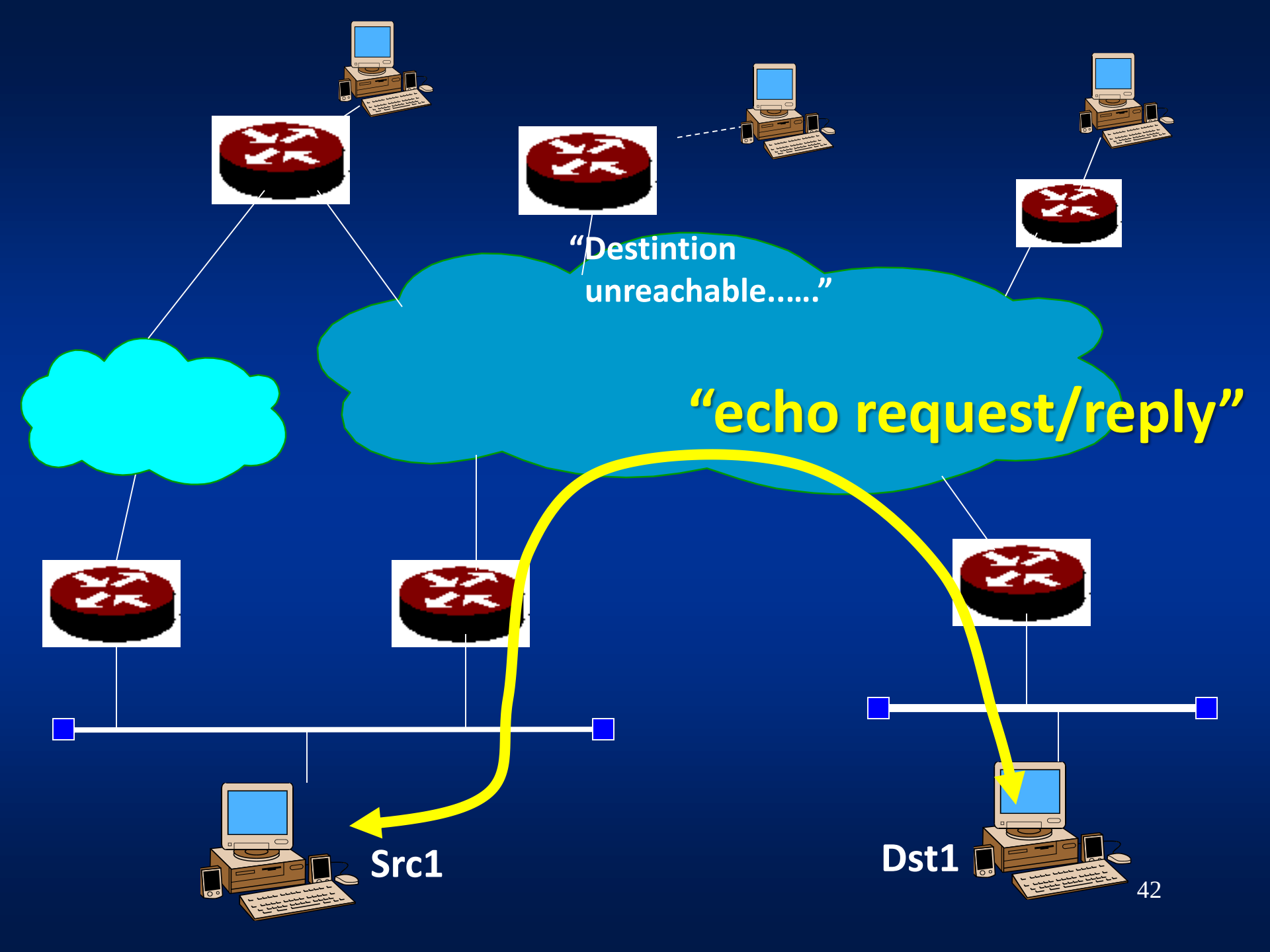


Well-known Portの例

ポート 番号	用途(アプリケーション層プロトコル)
20, 21	FTP: ファイル転送プロトコル(20:データ，21:制御)
25	SMTP: 簡易メール配送プロトコル
53	Domain Name Service (DNS)
80	HTTP: ハイパーテキスト 転送プロトコル

BW x RTT を測定できないか？

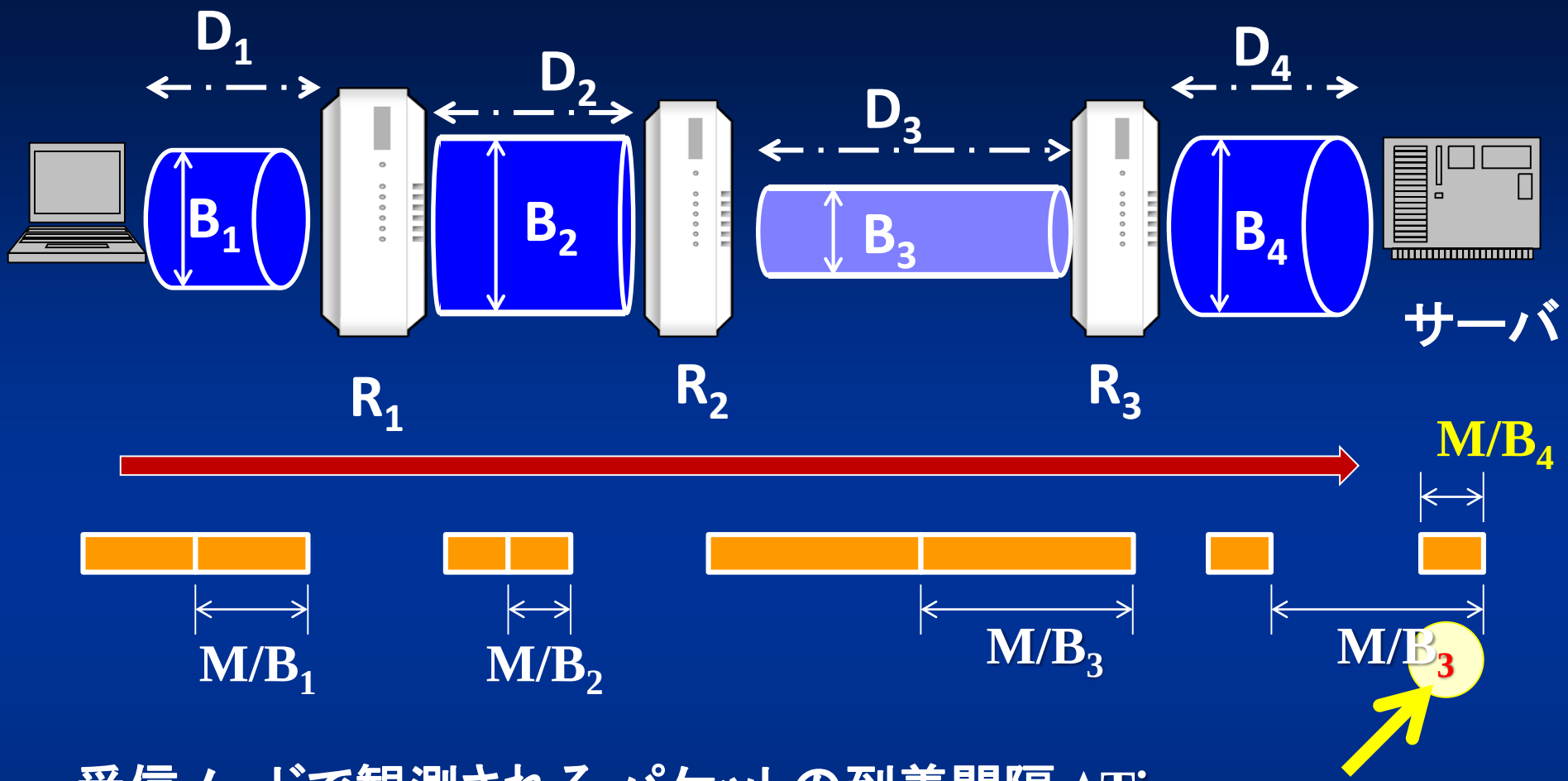
1. RTT
   - ✓ ping (by ICMP echo request/reply) で 可能


"Destintion unreachable....."
"echo request/reply"
Src1
Dst1

BW x RTT を測定できないか？

1. RTT
   - ✓ ping (by ICMP echo request/reply) で 可能

2. BW
   - ✓ Packet-Pair で可能
   - ✓ IPパケットを、連続して転送した時の、IPパケットの到着間隔 で 利用可能な 帯域幅の最大値が 分かる。



受信ノードで観測される パケットの到着間隔 ΔTi

$\min\{\Delta Ti\} = M/B_3$

つまり、サーバでの IPパケットの到着間隔の最小値で、経路上での 利用可能な最大帯域幅が分かる(図では、B_3)。

RFC 1379 ; T/TCP
- Transaction TCP -

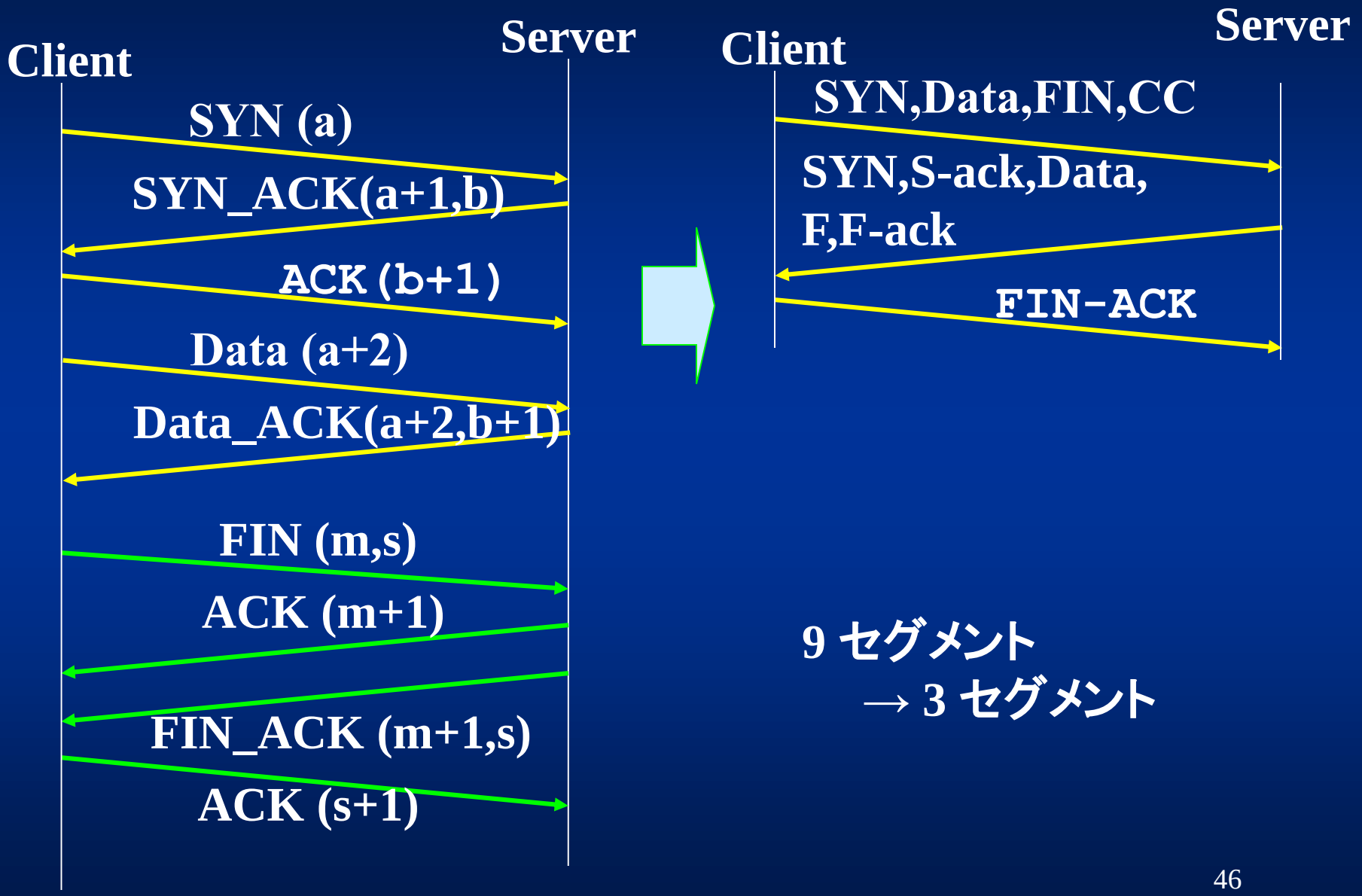
[目的]

TCPコネクションの確立・開放手続きの速度アップ

[方法]

- *CC (Connection Count)* Option
- SYNへのPiggy-back ; “*half-synchronization*”
  - (1) SYN, Data, FIN, CC
  - (2) SYN, SYN-ACK, Data, FIN, FIN-ACK, CC, CC-Echo
  - (3) FIN-ACK

RFC 1379 ; T/TCP



9 セグメント
→ 3 セグメント

再送制御

- 再送タイマーがゼロになったとき

  RTO = {平均RTT}+4{標準偏差}

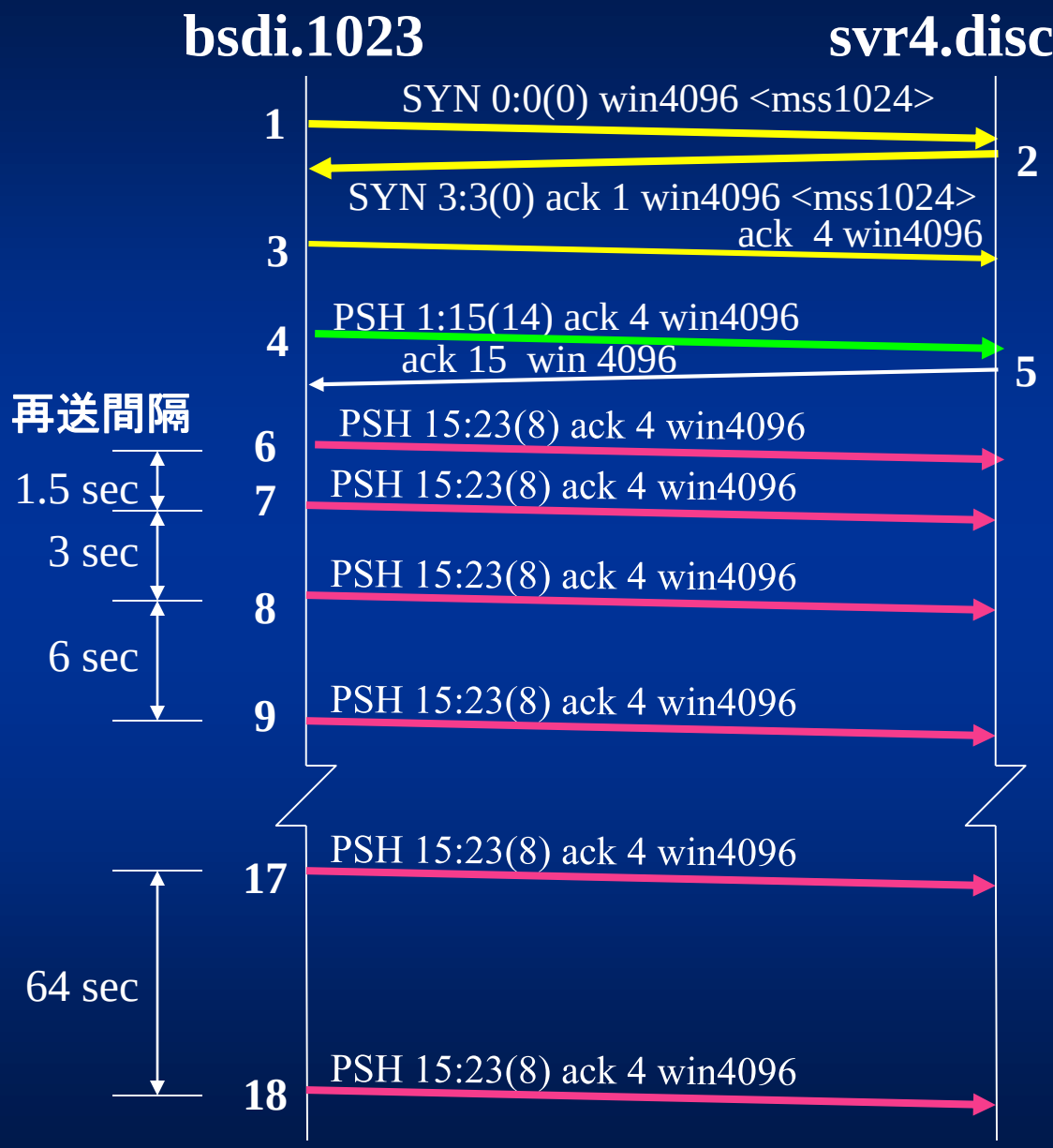
  再送ごとに2倍に(RTO ＜ 64秒)

  9分間ACKパケットを受信しないとコネクションを解放

- 同一ACKパケットを複数回受信したとき

  Fast Retransmission, Fast Recovery

RTO Expired Retransmission



再送トライ (RTO; 再送タイマ)

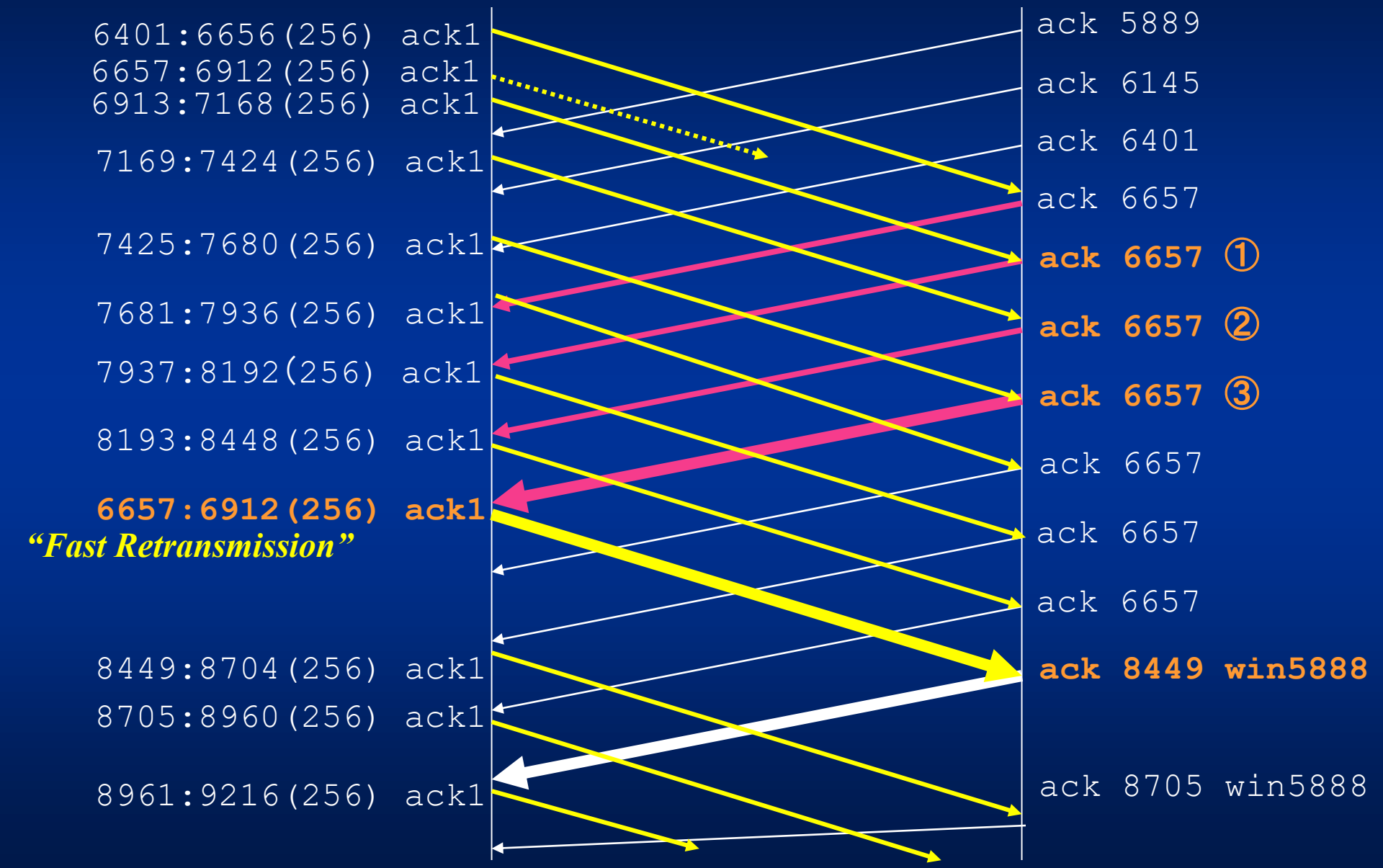
```
RTO = 1.5 sec /* 変更可能*/
for ( 9 minutes)
 {
 if ( RTO expired)
  {
   retransmission;
   RTO=RTO x 2;
   RTO=min{64sec, RTO};
  }
 }
end /* 諦める */
```

Retransmission by Duplicated ACK

(2) Reception of Duplicated ACK
- Fast Retransmission / Fast Recovery

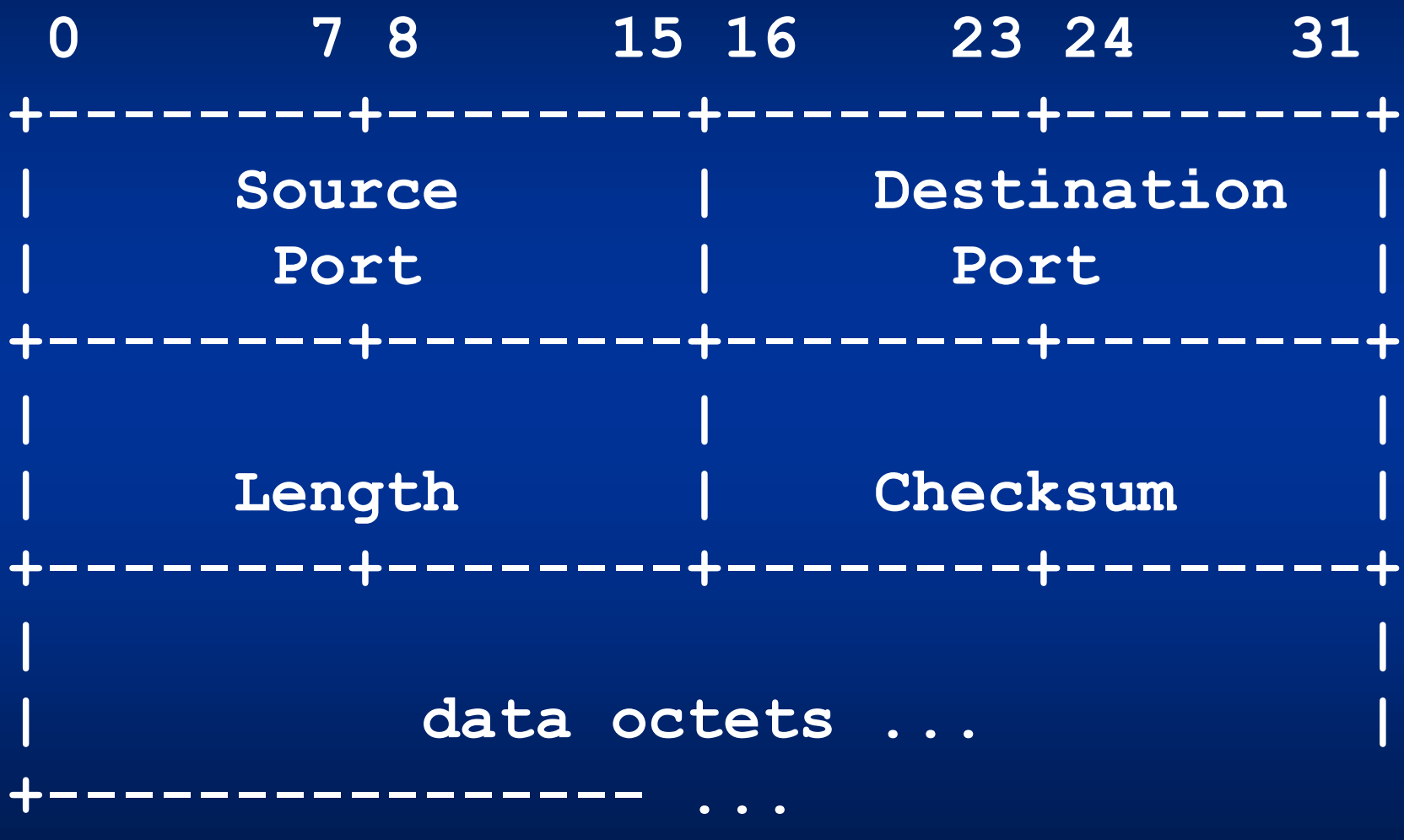
Segment廃棄特性；
→ “single (or few) segment(S)” あるいは連続多数。
→ 未ACKの同一ACK Segmentsを複数(3回)受信したら、再送。

Fast Retransmission by Duplicated ACK

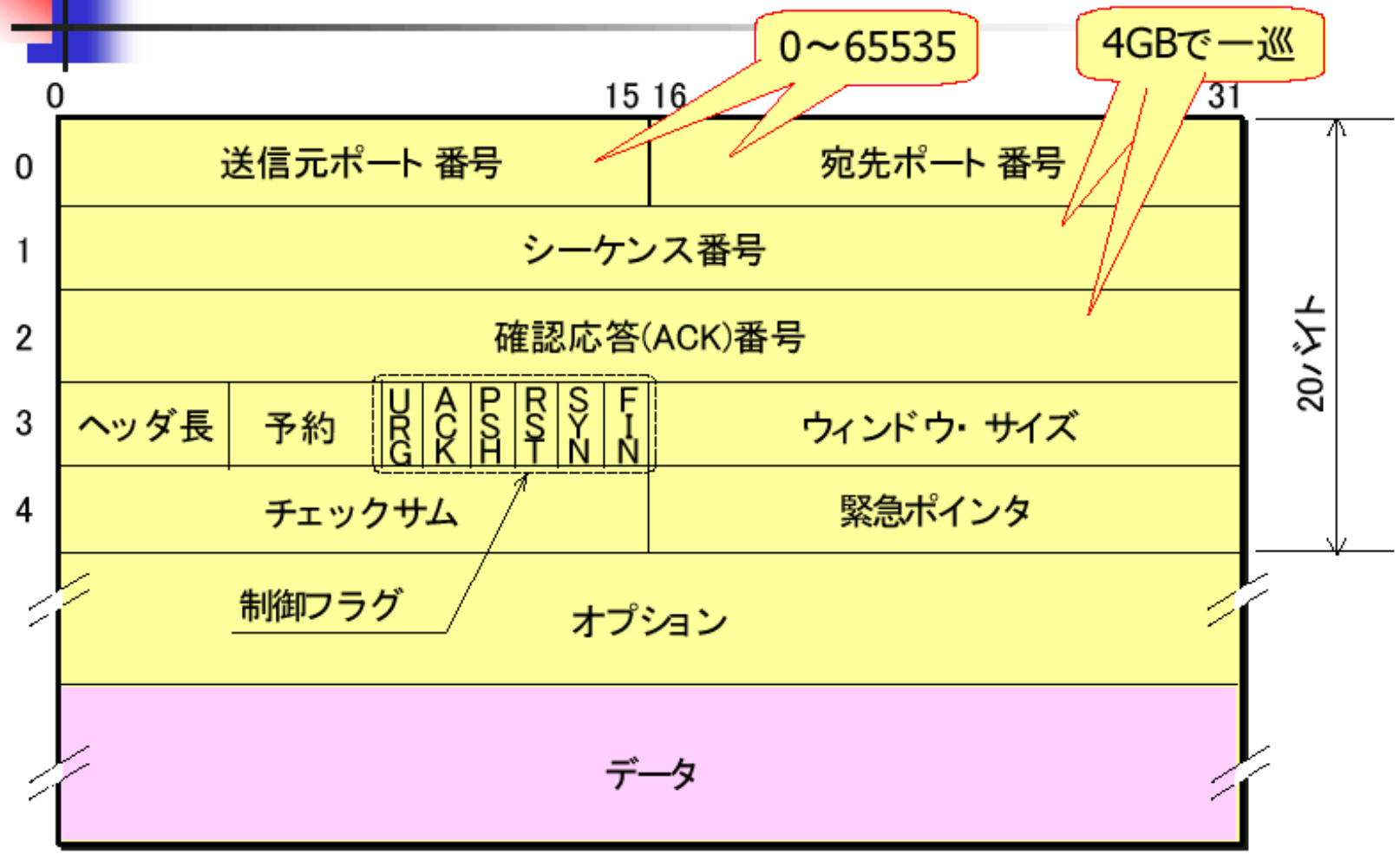


UDP

UDP Header format

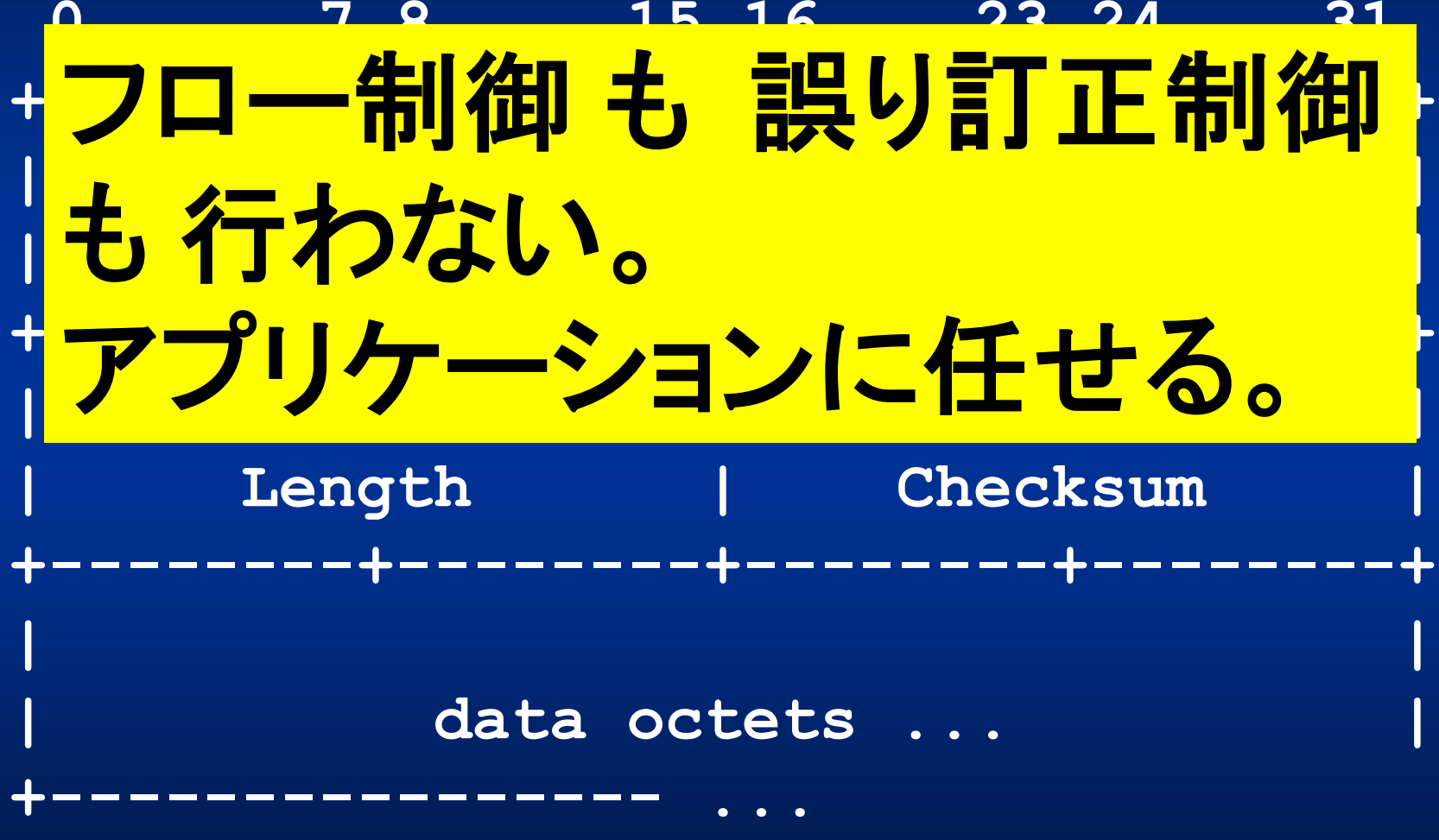


TCPパケット の構成



URG: 緊急ポインタフィールド
ACK: 確認応答フィールド
PSH: 強制転送機能
RST: コネクションリセット
SYN: コネクション確立要求
FIN: 転送データ終了

UDP Header format



RTP

RTP

- RTP; Real-time Transport Protocol
- RTPはEnd-Hostでのみ適用される
  (*) ルータでの通信品質はOut-of-Focus
- 基本仕様; RFC1889, RFC1890
- Playbackタイミングの再生
  - Payload Type
  - Sequence Number
  - Time-Stamp
- 2対のUDP Portを使用
  - User Data
  - Control Data
- ContentごとにRTP Payload Formatを規定

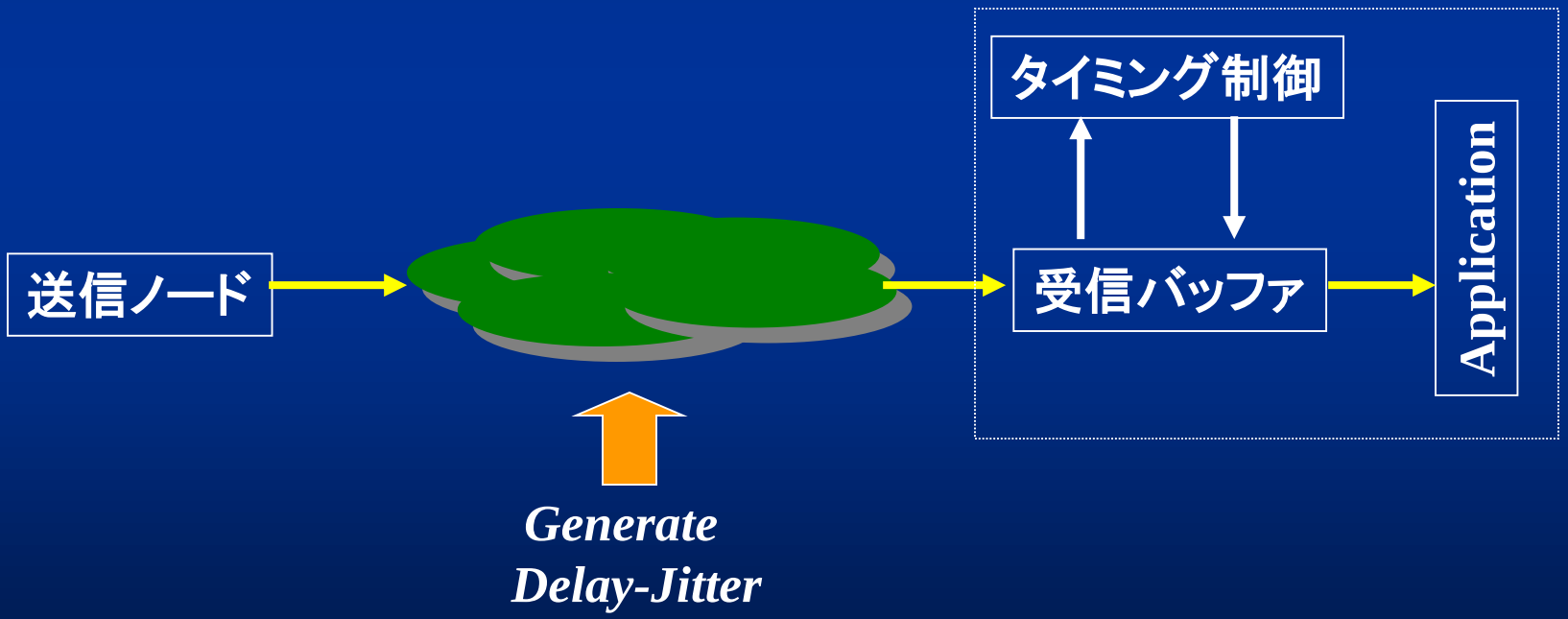
RTP

- RTP Payload Format 仕様
  RFC2029 ; CellB Video Encoding (for SUN)
  RFC2032 ; H.261 Video Stream
  RFC2035 ; JPEG-compressed Video
  RFC2250 ; MPEG1/MPEG2 Video

- Control Protocol
  RTCP ; RTP Control Protocol
- 通信品質監視機能
  - 通信受信/送信ノード
  - 品質監視ノード

RTP

- RTPの仕事;
  「受信ノードにおいて、送信側から送信される
  データの出力タイミングを再生する。」

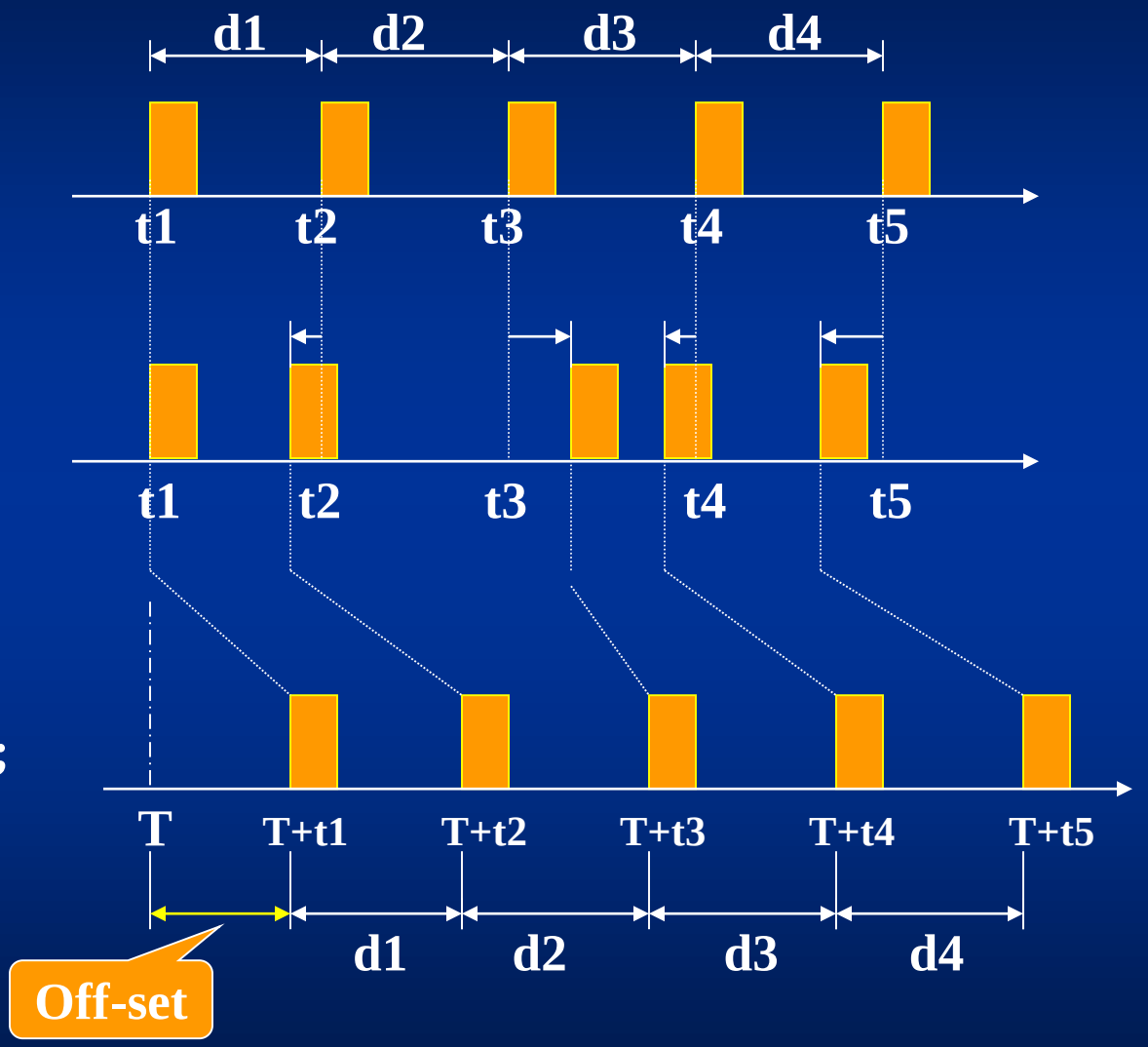


RTP

・送信側タイミング;

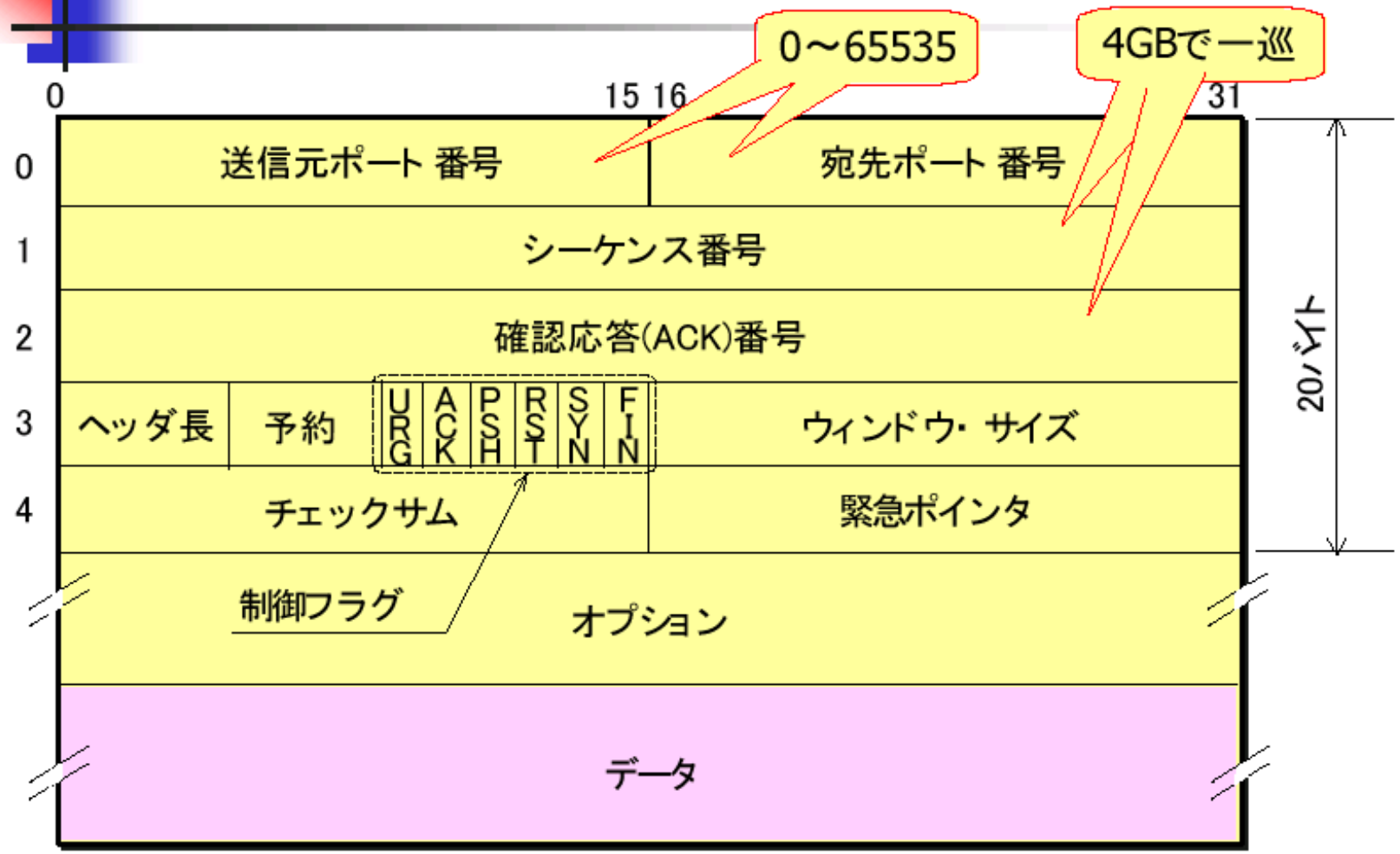
・受信側入力タイミング;

・受信側出力タイミング;



NAT

TCPパケット の構成



URG: 緊急ポインタフィールド
ACK: 確認応答フィールド
PSH: 強制転送機能
RST: コネクションリセット
SYN: コネクション確立要求
FIN: 転送データ終了

NAT(Network Address Translation)

- 受信パケットのIPアドレス(src_IP)およびポート番号の(src_port)変換テーブルを持ちIPヘッダの変換。(RFC1631)

  (1) Private → Global

  - DNS：宛先ノードのIPアドレスが解決される。
  - 受信パケット(dst_IP)
    → 送信パケットの(src_IP, src_port)の書換え

  (2) Global → Private

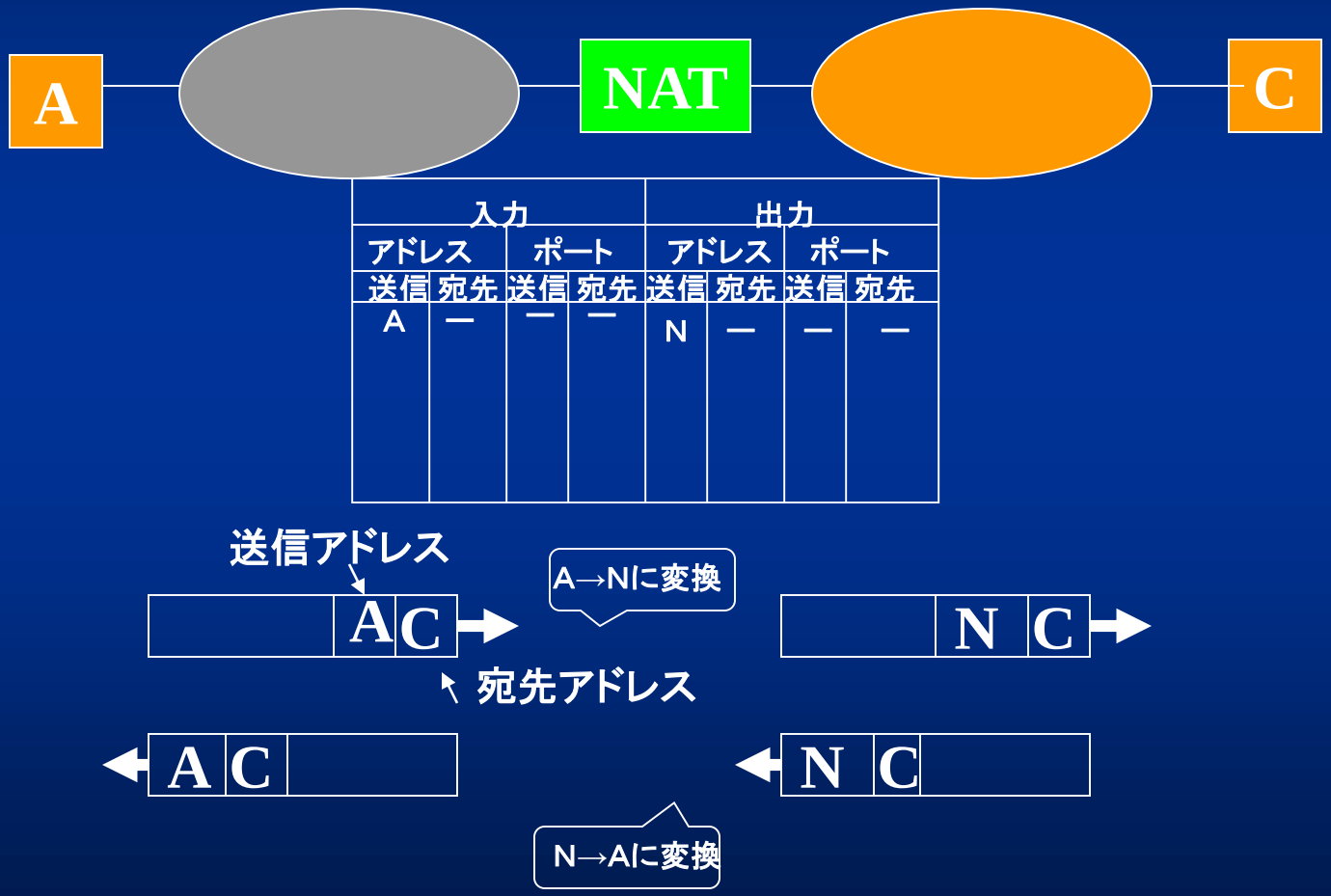
  - 受信パケット(src_IP, src_port)
    → 送信パケットの(dst_IP)の書換え

(*) ポート番号(src_port)の機能

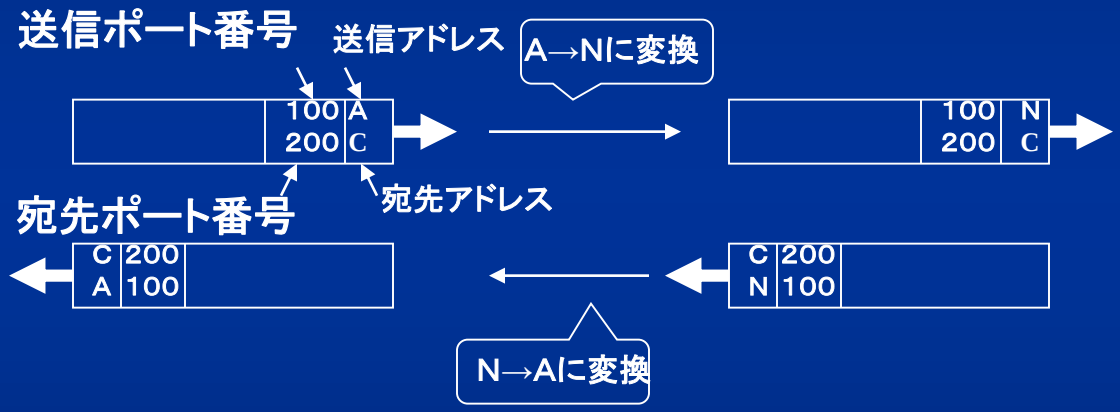
(i) src_IPの多重化

(ii) dst_IPのマッピング

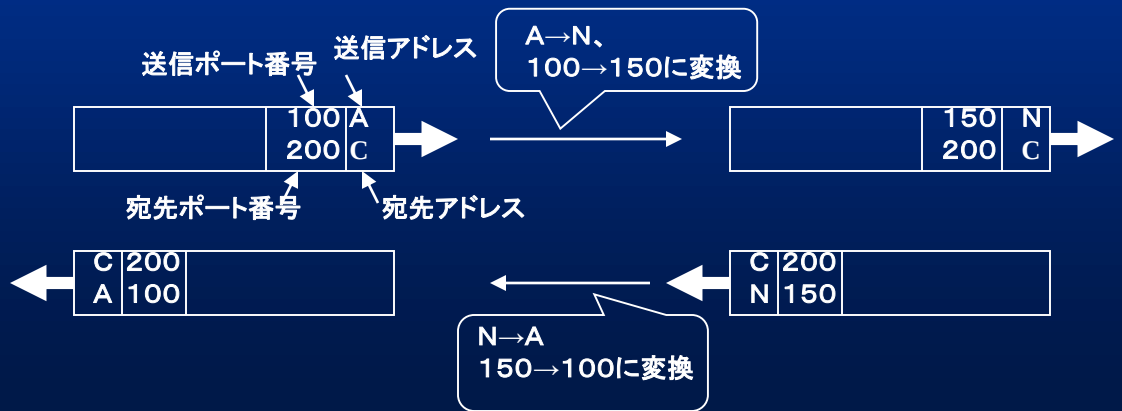
NAT


A
NAT
C
入力
出力
アドレス
ポート
アドレス
ポート
送信
宛先
送信
宛先
送信
宛先
送信
宛先
A
ー
ー
ー
N
ー
ー
ー
送信アドレス
A
C
A→Nに変換
N
C
宛先アドレス
A
C
N
C
N→Aに変換

Traditional NAT

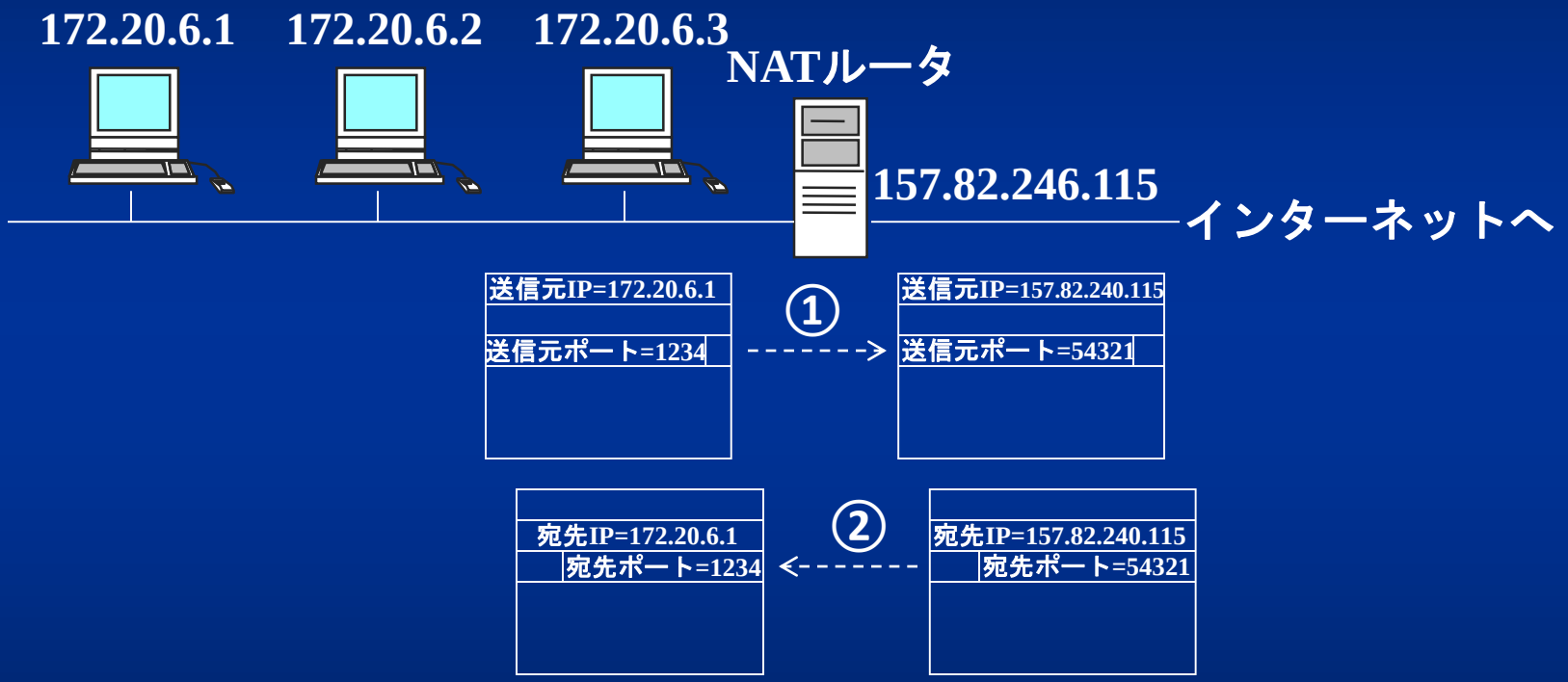


Basic NAT



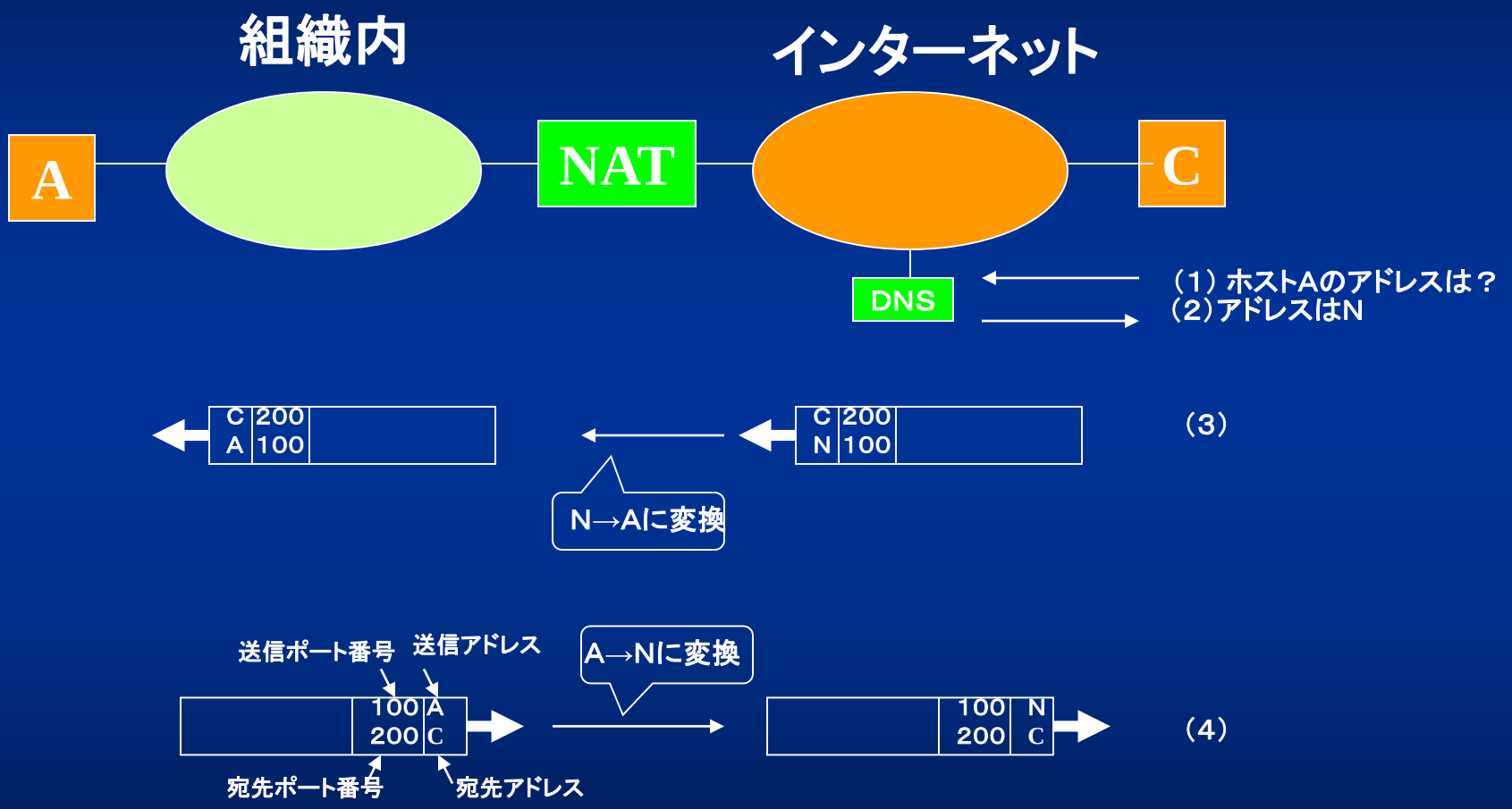
NAPT

NAPT (Network Address and Port Translation)

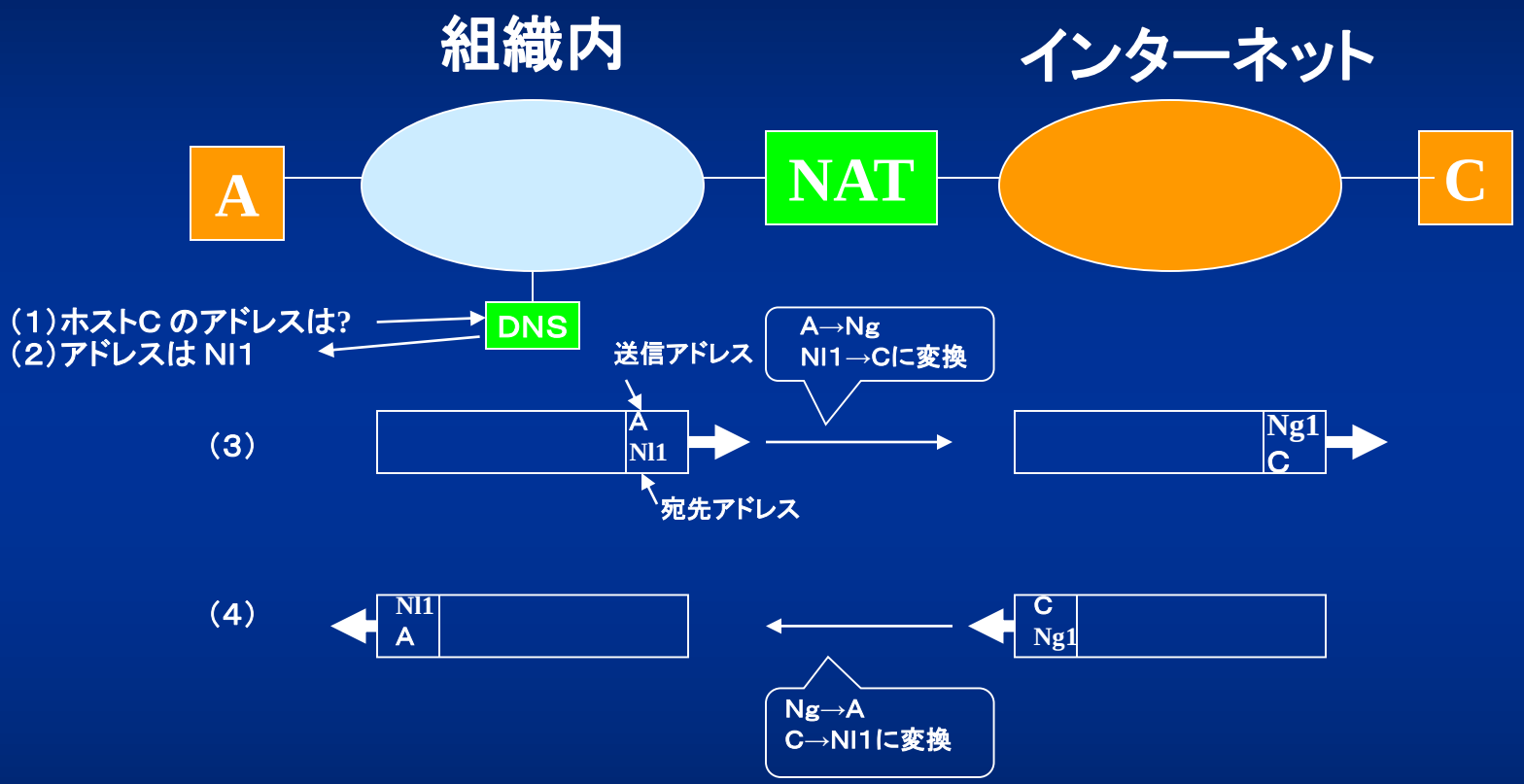


① 内から外に向かうパケットがあるとNATルータはポート番号を割当
② その後外から来るパケットについてもIPアドレスとポート番号を変換

Bi-directional NAT



Twice NAT


組織内
インターネット
A
NAT
C
DNS
(1)ホストC のアドレスは?
(2)アドレスは NI1
送信アドレス
A→Ng
NI1→Cに変換
(3)
A
NI1
Ng1
C
宛先アドレス
(4)
NI1
A
C
Ng1
Ng→A
C→NI1に変換

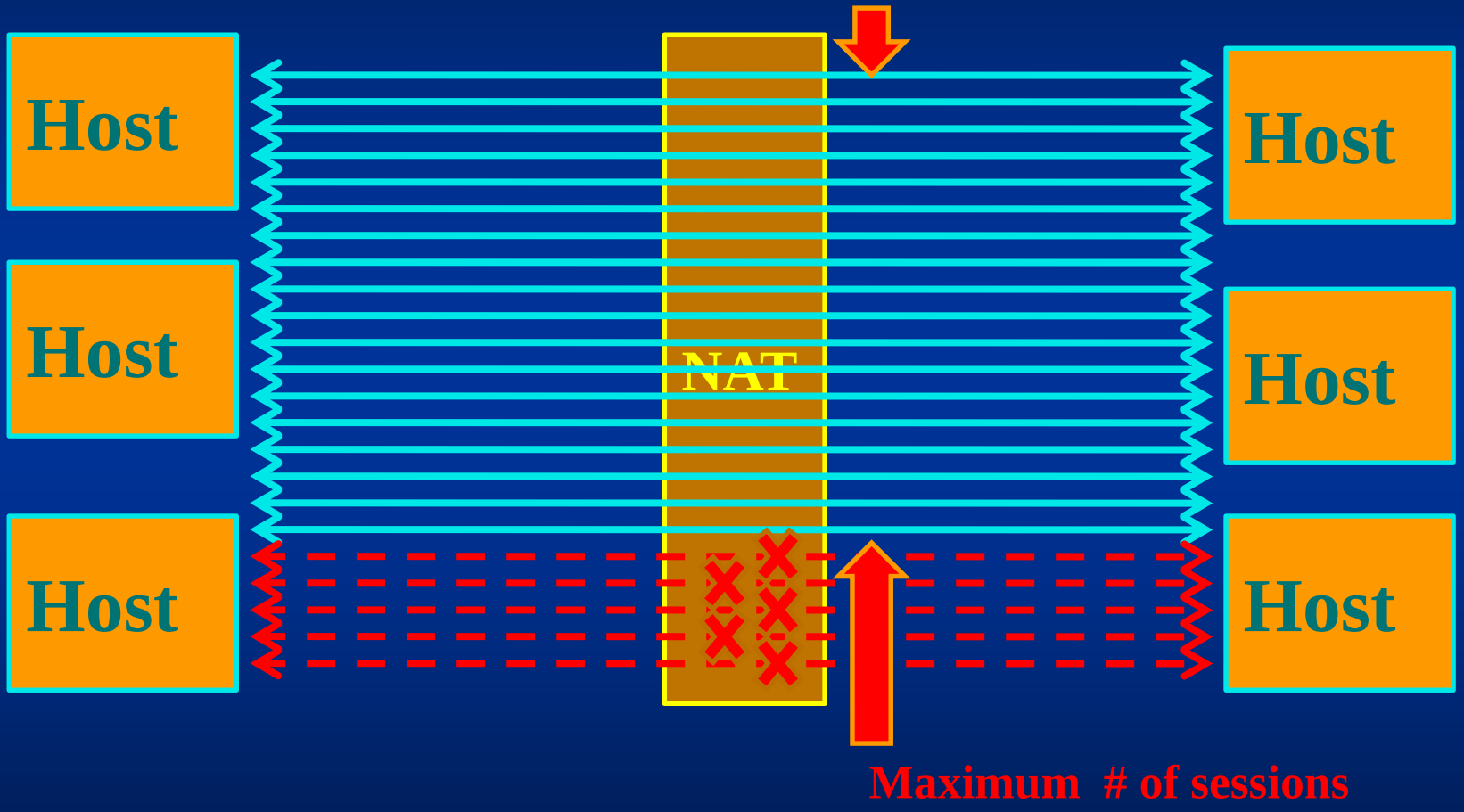
However……

- **Limitation** on the number of session states for NAT operation
- Each user could use certain number of sessions
  - How many sessions ?
  - Even as the best case, **65,536** is the maximum number of sessions, shared by customers accommodated into a single IPv4 address

    ➔ When the number of users is **2,000**, it will be **only 30 sessions**

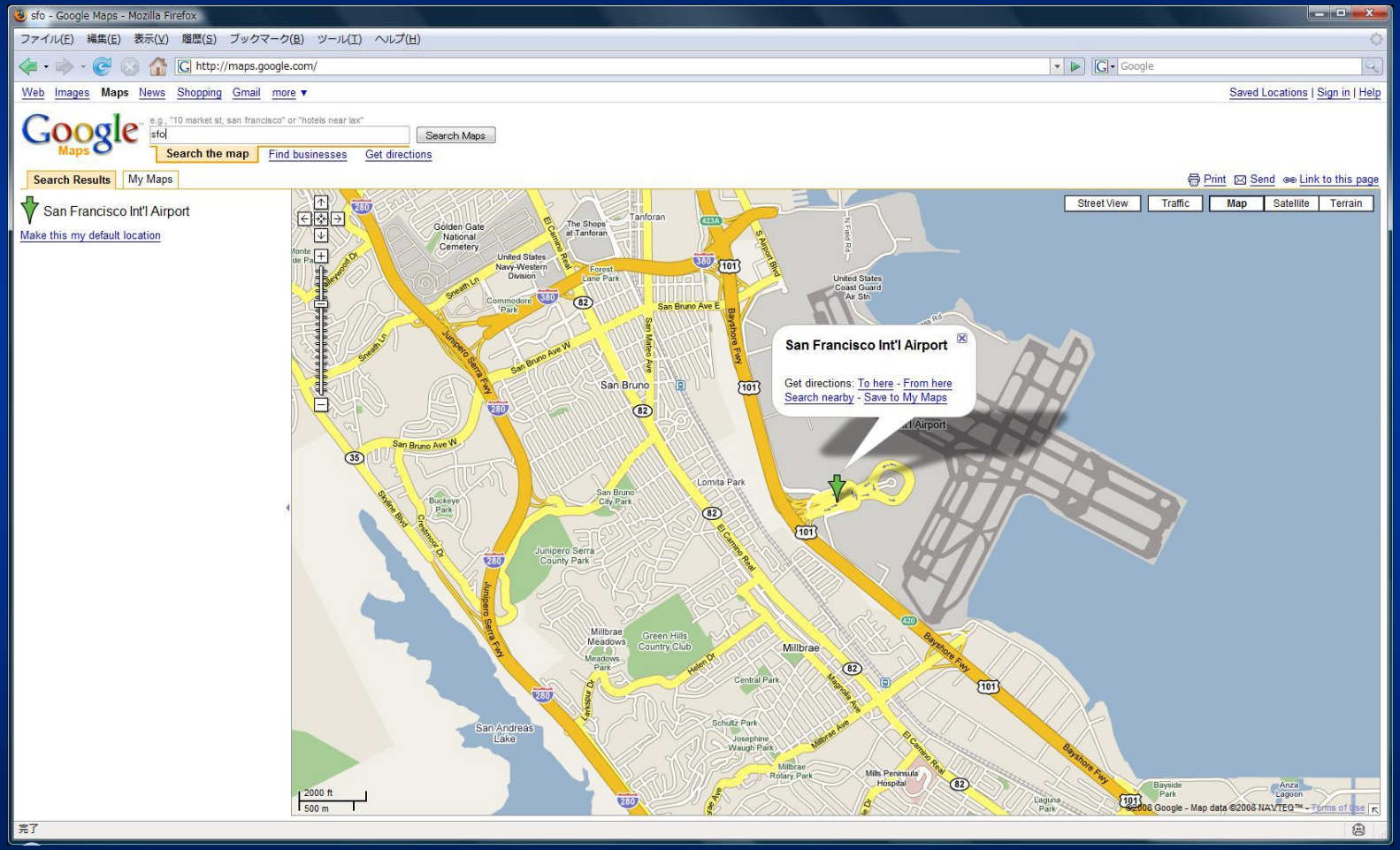
➔ This means……..

Limitation of NAT Solution

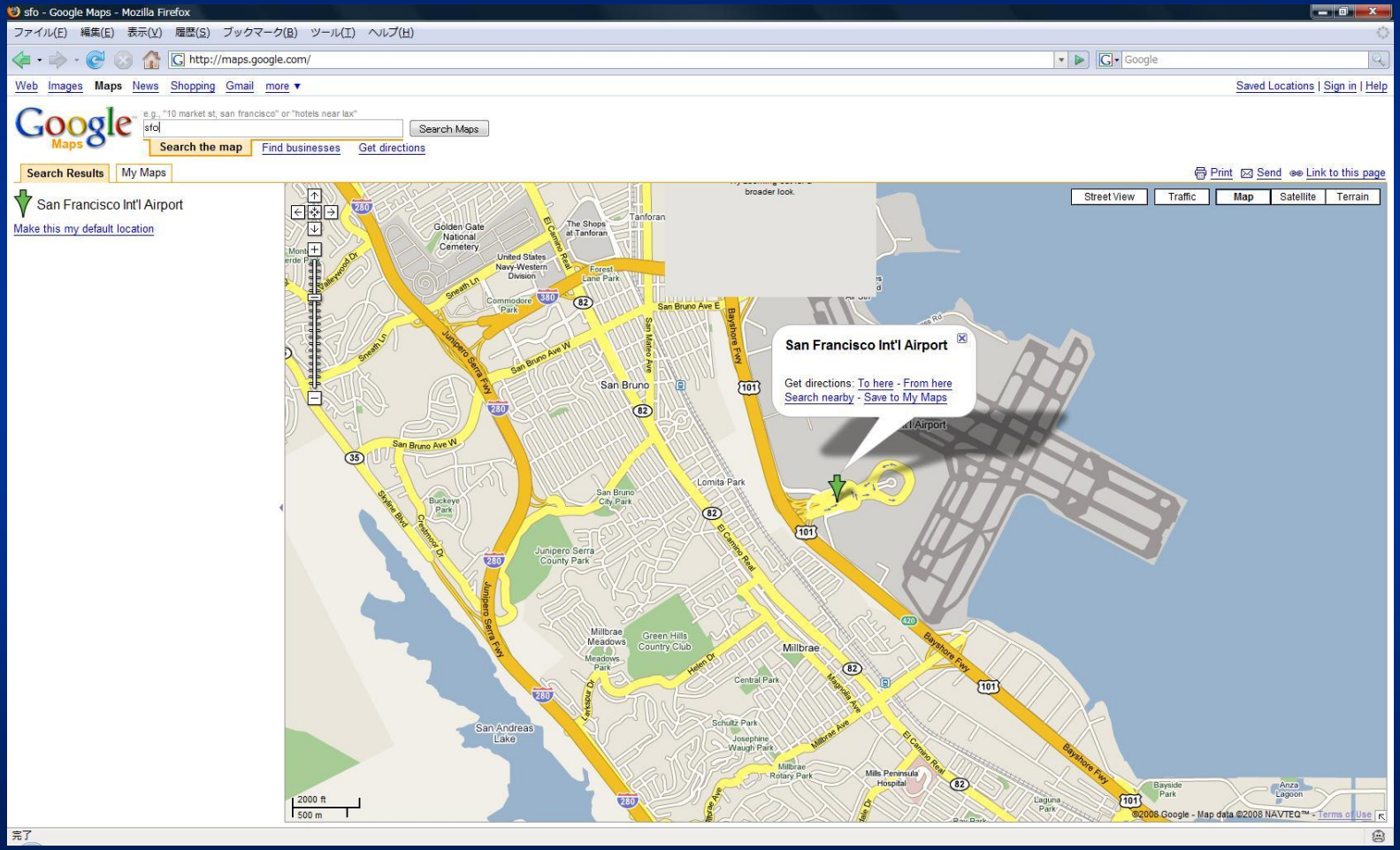


Limitation of NAT Solution

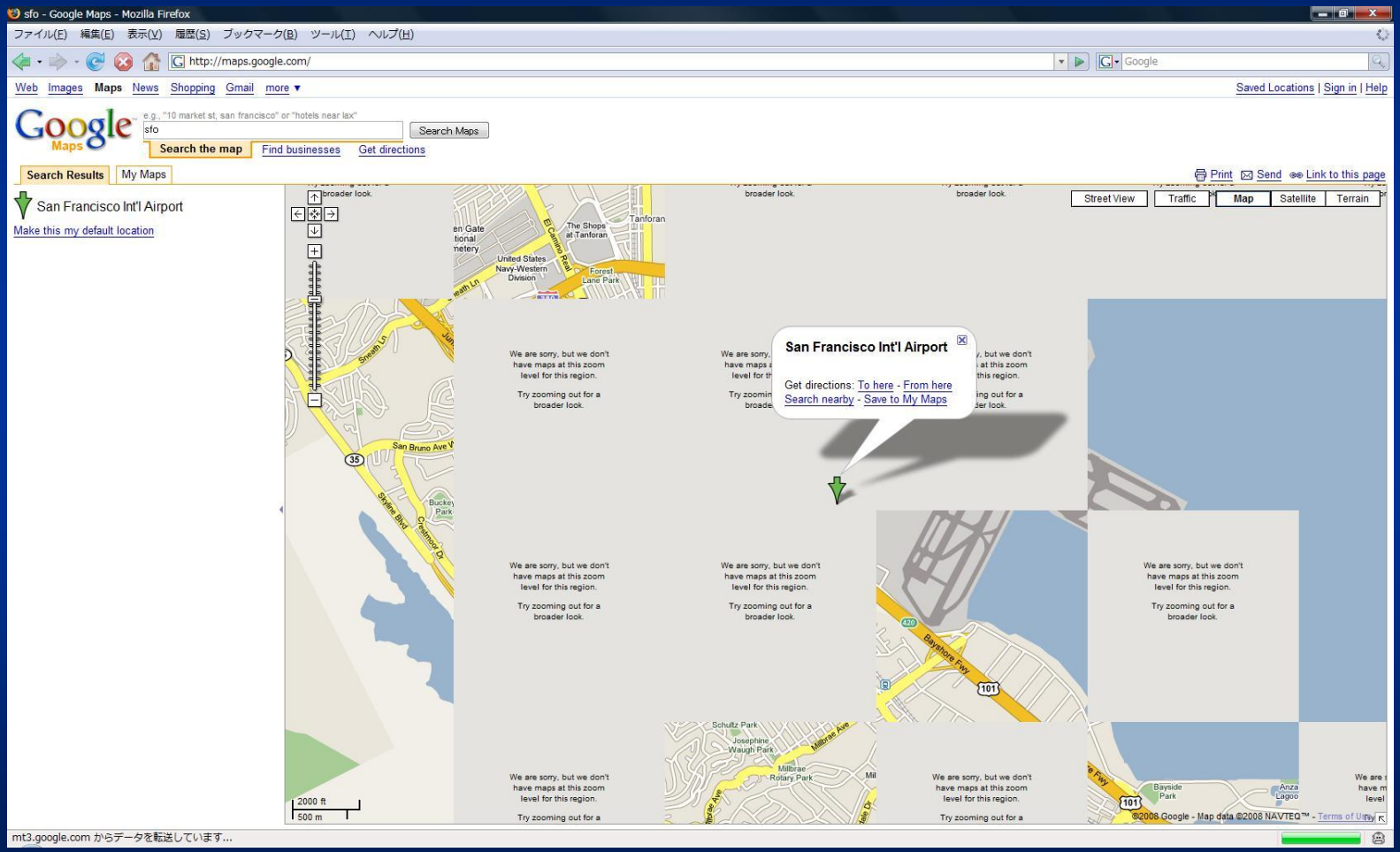
Max 30 Connections



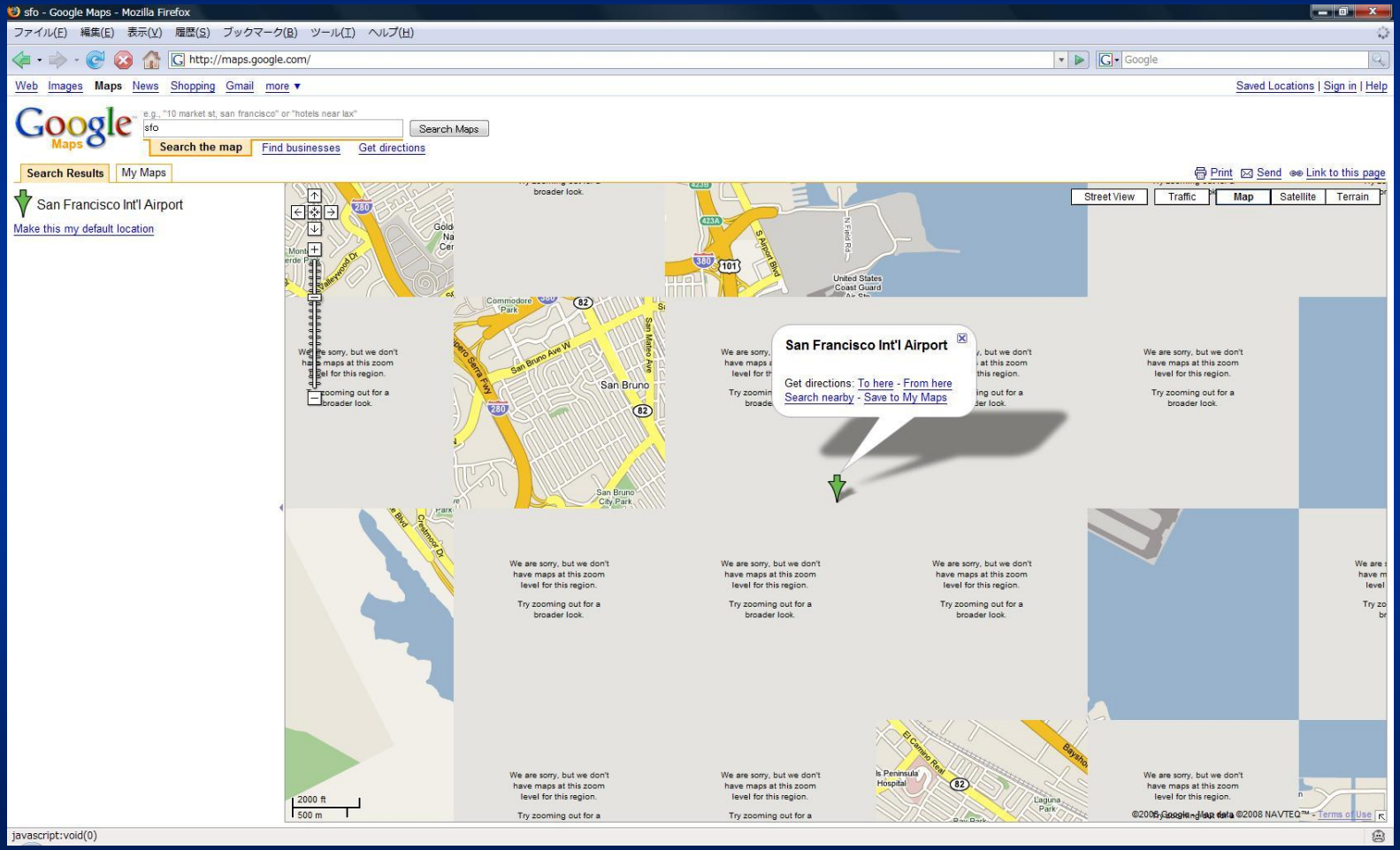
Max 20 Connections



Max 15 Connections



Max 10 Connections



Max 5 Connections

Some examples of major Web site

Application	# of TCP sessions
No operation	5～10
Yahoo top page	10～20
Google image search	30～60
ニコニコ動画	50～80
OCN　photo friend	170～200+
iTunes	230～270
iGoogle	80～100
楽天(Rakuten)	50～60
Amazon	90
HMV	100
YouTube	90

新しい方向性

- マルチパス化
  - MTCP (Multi-Pah TCP)
    - ➔ 複数のIPアドレスを利用
- UDP ＋ アプリケーション での実装
  - **QUIC(Quick UDP Internet Connection)**

    (*) Proposed and implemented by Google

    1. 初期設定のための Hand-Shake を高速化
    2. 前方誤り訂正機能(FEC : Forward Error Correction)
    3. 暗号化
    4. 自由なフロー制御