I465S
セキュリティ関連実践的知識
―ネットワークを中心に―

知念

北陸先端科学技術大学院大学
セキュリティ・ネットワーク領域 サイバーレンジ構成学
Cyber Range Organization and Design,
Security and Networks Area,
Japan Advanced Institute of Science and Technology

オーバービュー

ネットワークのトラヒック観測の基礎を紹介

- TCP/IP、関連プロトコル
- 待ち行列、性能曲線
- ソケットAPI
  サーバ、クライアント、プロキシ

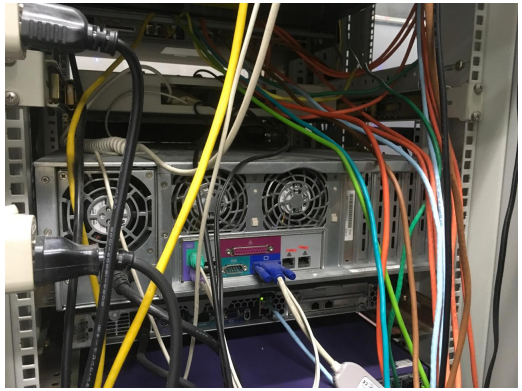
RFC (Request for Comment)

- プロトコルを規定する文章（標準化）
  `https://www.rfc-editor.org/`
- 歴史的経緯で「コメント求む」という名称
- 例）
  - IP (Internet Protocol; RFC 791)
  - TCP (Transmission Control Protocol; RFC 793)
  - UDP (User Datagram Protocol; RFC 768)

IPアドレスとホスト

IPアドレスとホストは 1対1対応ではない

- IPアドレスは ネットワークインターフェース につく



サーバPCでは
複数IFが典型的

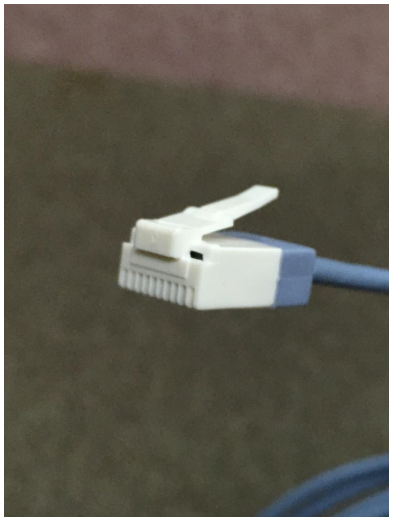
IPアドレスとホスト *(cont.)*

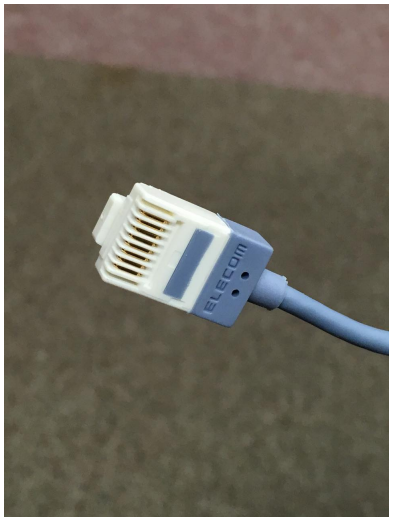
- ルータでは複数インターフェイスが基本
  - ◇ 組織の基幹ルータでは数十 IF も多い
- 最近のノートPCでも複数IF // WiFi, Bluetooth他

典型的用語

ネットワークインターフェース
network interface
ネットワークインターフェースカード
network interface card (NIC)
ネットワークデバイス
network device

有線インターフェース例: RJ-45





10BASE-T
100BASE-TX
1000BASE-T
10GBASE-T　など

インターネット

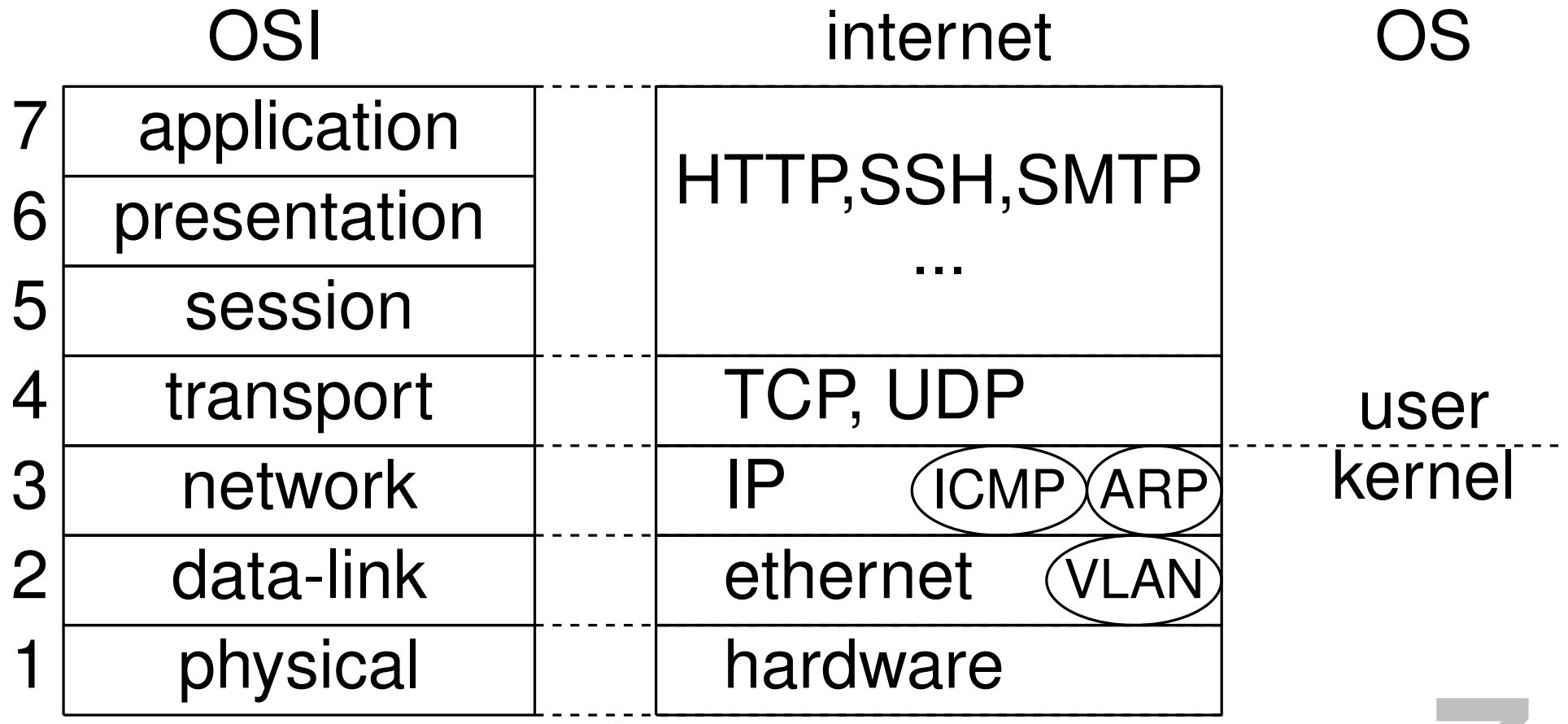
インターネット = ネットワークのネットワーク
the Internet
the net

過去には様々なネットワークがあった

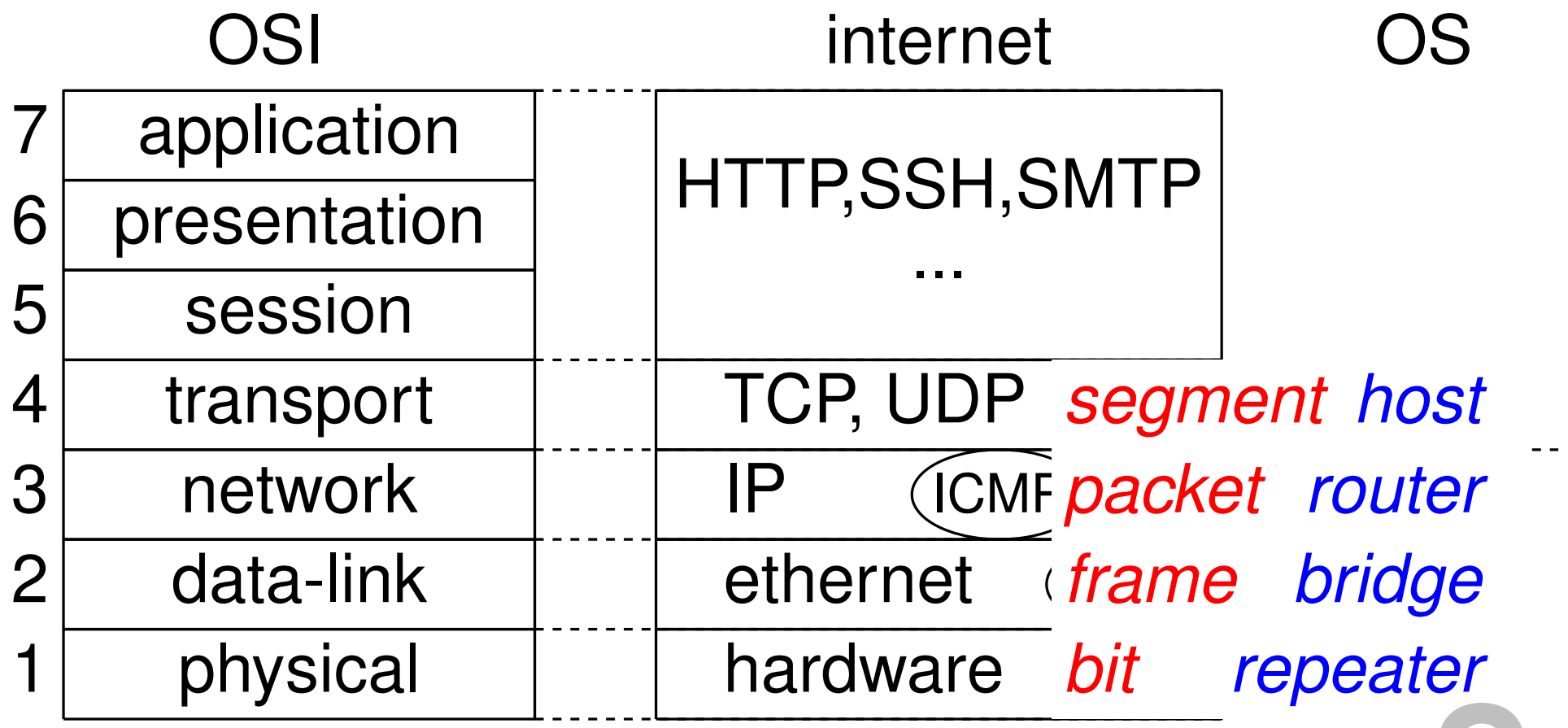
- AppleTalk
- DECnet
- NETBEUI(NETBIOS)
- その他

淘汰されて、TCP/IP が主流に

階層モデル


OSI
internet
OS
7
application
6
presentation
5
session
HTTP,SSH,SMTP
...
4
transport
TCP, UDP
user
3
network
IP
ICMP
ARP
kernel
2
data-link
ethernet
VLAN
1
physical
hardware

階層モデル *(cont.)*


OSI
internet
OS
7
application
6
presentation
5
session
HTTP,SSH,SMTP
...
4
transport
TCP, UDP
segment
host
3
network
IP
ICMP
packet
router
2
data-link
ethernet
frame
bridge
1
physical
hardware
bit
repeater

識別子

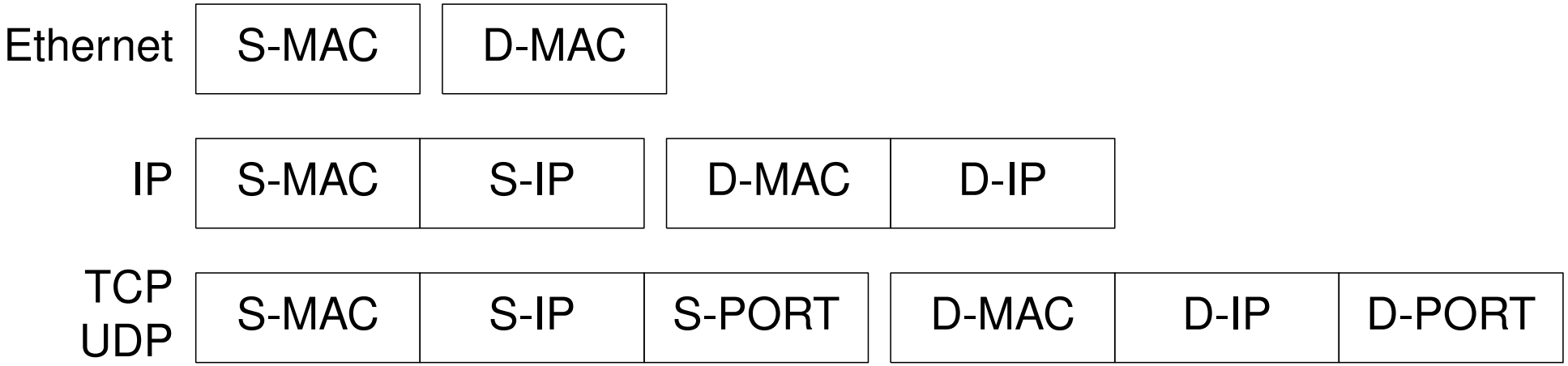
- MACアドレス[6バイト]
  00:1B:21:59:AE:FC （Intel製）
- IPアドレス[4/16バイト] // ホスト名ではない
  IPv4 150.65.74.93
  IPv6 2001:df0:2ed:7201:851:8f3a:1223:cf05
- ポート[2バイト]（いくつかは定まっている）

FTP	21	**BOOTPS(DHCP)**	67
SSH	22	**BOOTPC(DHCP)**	68
TELNET	23	**HTTP**	80
DNS	53	**HTTPS**	443

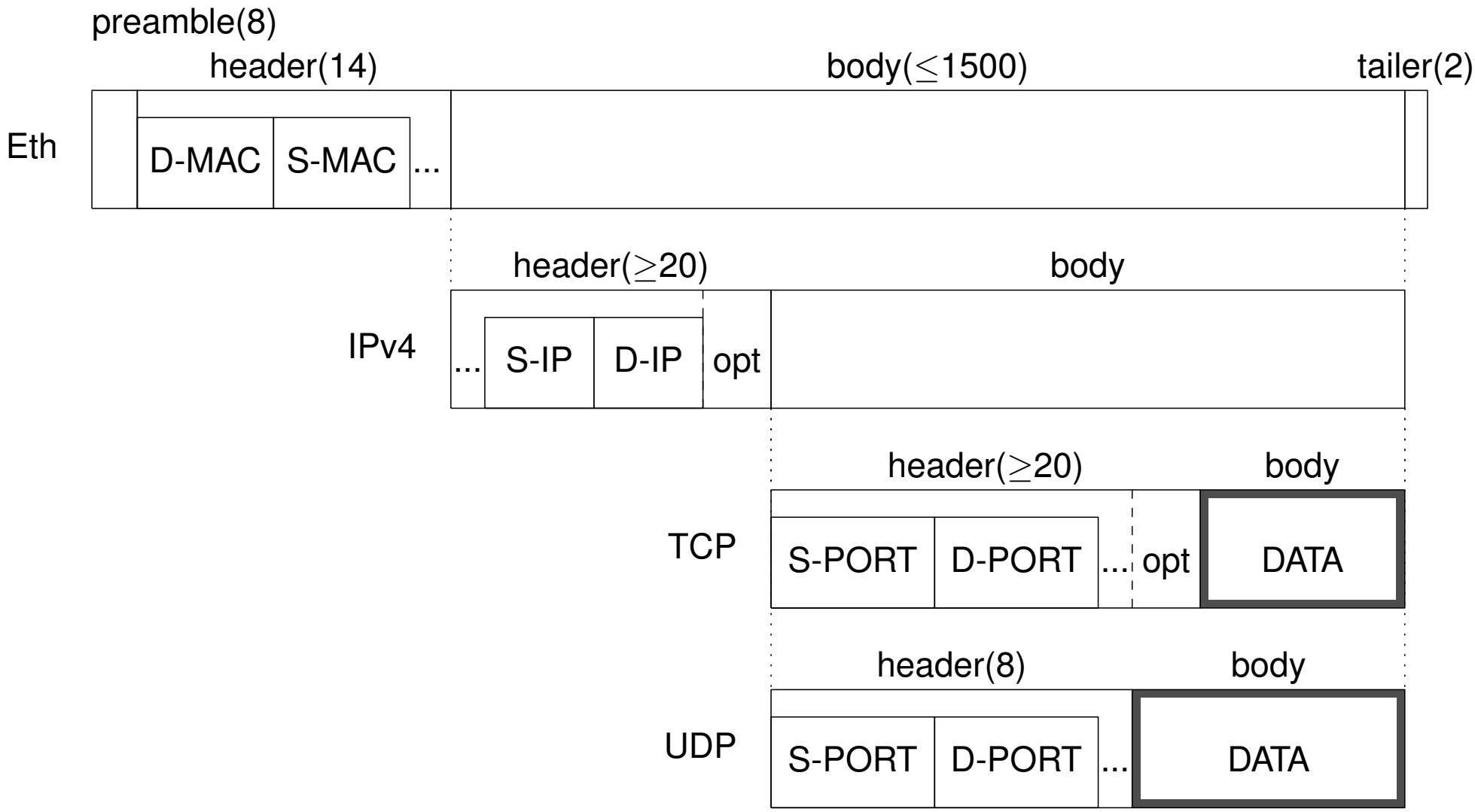
通信の識別

以下のような項目で通信を識別する

※S: 送付元 source ; D: 送付先 destination

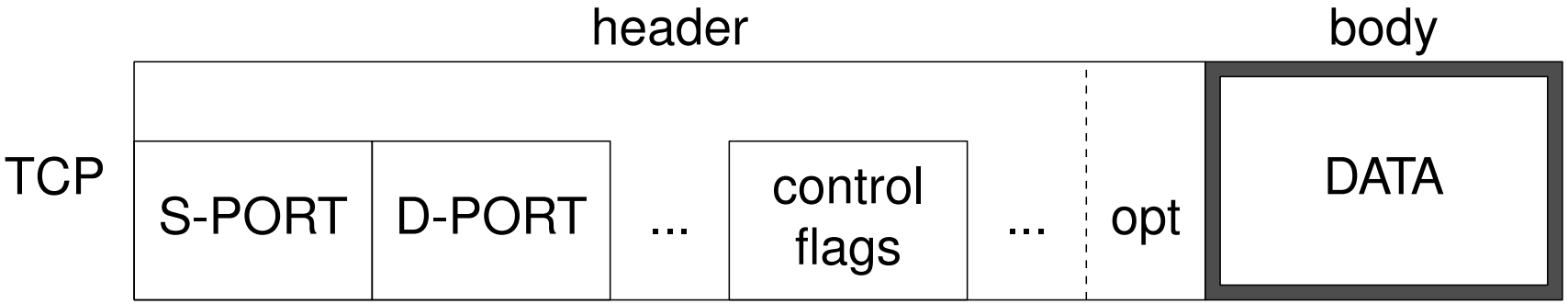


プロトコルフォーマットは上記と異なる


preamble(8)
header(14)
body(≤1500)
tailer(2)
Eth
D-MAC
S-MAC
...
header(≥20)
body
IPv4
...
S-IP
D-IP
opt
header(≥20)
body
TCP
S-PORT
D-PORT
...
opt
DATA
header(8)
body
UDP
S-PORT
D-PORT
...
DATA

TCP

- TCP は状態がある
- 状態は control flags に応じて遷移する



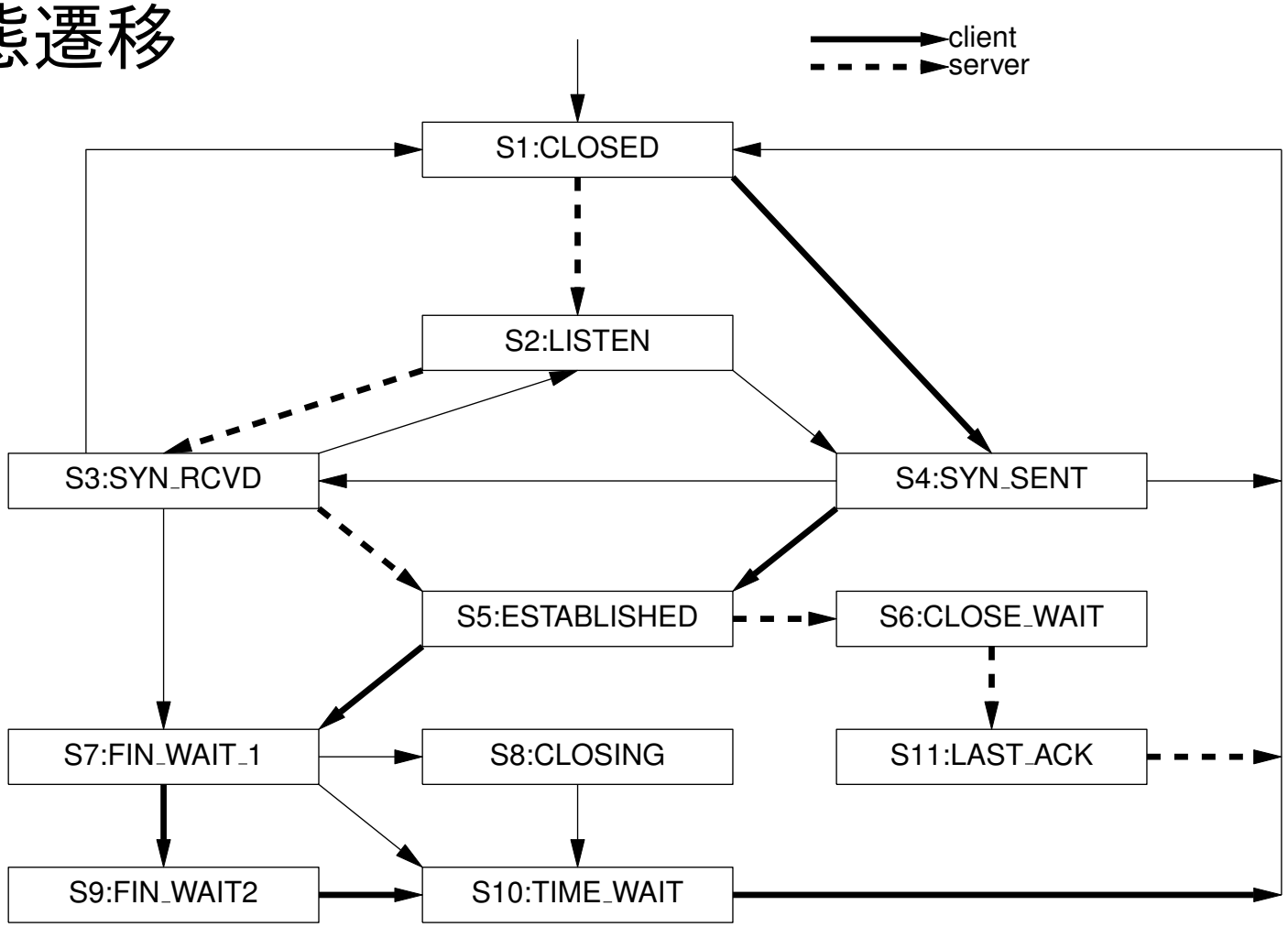
- 具体的なフラグは以下の 6つ
  URG, ACK, PSH, RST, SYN, FIN

TCP *(cont.)*

URG Urgent Pointer field significant
ACK Acknowledgment field significant
PSH Push Function
RST Reset the connection
SYN Synchronize sequence numbers
FIN No more data from sender

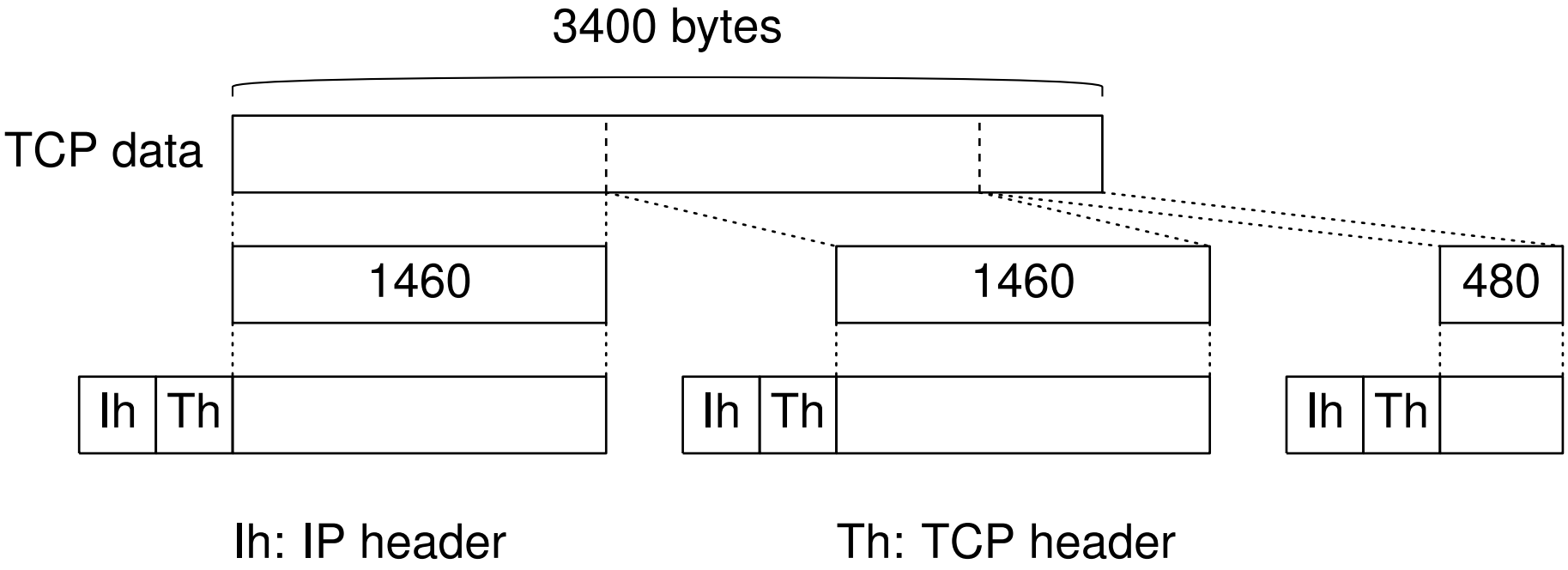
※ RFC793参照

TCP 状態遷移


client
server
S1:CLOSED
S2:LISTEN
S3:SYN_RCVD
S4:SYN_SENT
S5:ESTABLISHED
S6:CLOSE_WAIT
S7:FIN_WAIT_1
S8:CLOSING
S11:LAST_ACK
S9:FIN_WAIT2
S10:TIME_WAIT

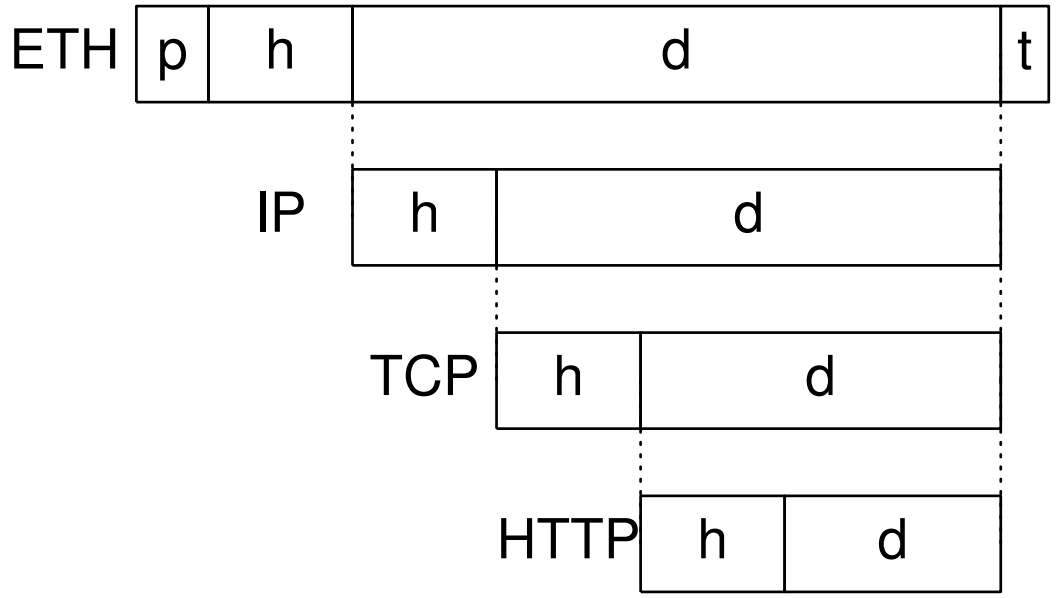
TCP から IP への分割★

TCP のデータは分割され、TCP や IP ヘッダがつく
3400バイトデータの例;　　*3400 = 1460*2 + 480*



アプリケーションプロトコル

TCP や UDP 上のプロトコル。以下は HTTP の例



※ 前述の IP への分割は省略

アプリケーションプロトコル: **HTTP**

HTTP(HyperText Transport Protocol; RFC1945,7230)

- WWW (World Wide Web) 向け
- それ以外にも広く使われる; 今年 30 周年
- TCP 上のプロトコル
- 基本は、1 資源 1 コネクション
- ヘッダはテキスト、ボディは様々

HTTPリクエスト

```
GET△/△HTTP/1.0
Host:△10.1.2.3

```

- 行単位
- 行末は CR (\r ; 0x0d) と LF （\n ; 0x0a）
- 最後は空行; CR LF だけ
- 上の例は 34バイト
- リクエスト最初の行
  メソッド、ターゲット、バージョン

HTTP リクエスト *(cont.)*

メソッド:
- 主に GET, HEAD, POST が使われる

ターゲット:
- 主にパス。プロキシでは URL

バージョン:
- HTTP/1.0 や HTTP/1.1 など
- （付いていない場合は HTTP/0.9）

`https://www.w3.org/Protocols/HTTP/AsImplemented.html`

HTTP リクエスト *(cont.)*

- ヘッダは基本任意、Host ヘッダのみ必須

長い例（Windows Internet Exproler 11 の抜粋）

```
GET△/△HTTP/1.0
Host:△10.1.2.3
Accept:△*/*
User-Agent:△Mozilla/5.0△(Windows△NT△6.3;△WOW64;△Trident/7.0
```

- User-Agent はクライアントのこと
  OS やアプリケーションのバージョンも付加
  Mozilla とあるのは互換性確保のため

HTTP レスポンス

```
HTTP/1.0△200△OK

<HTML>
...
```

- 200 はレスポンスコードで、成功
- この例は HTML なのでテキスト
- 画像などはバイナリ
- ヘッダは任意

www.jaist.ac.jp の例

```
HTTP/1.1△200△OK
Date:△Tue,△12△Mar△2019△09:32:55△GMT
Server:△Apache
Last-Modified:△Tue,△12△Mar△2019△01:33:45△GMT
ETag:△"9a416-e6cf-583dbae1cb2f1"
Accept-Ranges:△bytes
Content-Length:△59087
X-FRAME-OPTIONS:△SAMEORIGIN
Keep-Alive:△timeout=5,△max=100
Connection:△Keep-Alive
Content-Type:△text/html

<!DOCTYPE△html>
<html△class="no-js"△prefix="og:△http://ogp.me/ns#"△lang="ja">
<head>
<meta△charset="UTF-8">
<title>JAIST△北陸先端科学技術大学院大学</title>
<meta△name="viewport"△content="width=device-width,minimum-scale=
```

トラヒック具体例★

```
0x0000:  ac1f 6b1d 7b8a 0030 4885 f5c9 0800 4500
0x0010:  00c0 0000 4000 3f06 e40c ac14 fe01 ac14
0x0020:  0101 cc5b 0050 da6e 5c84 6fa7 ffa7 8018
0x0030:  0805 6159 0000 0101 080a 29fb 95ed 69e0
0x0040:  4eff 4745 5420 2f20 4854 5450 2f31 2e31
0x0050:  0d0a 5573 6572 2d41 6765 6e74 3a20 5767
0x0060:  6574 2f31 2e31 362e 3320 2864 6172 7769
0x0070:  6e31 342e 312e 3029 0d0a 4163 6365 7074
0x0080:  3a20 2a2f 2a0d 0a41 6363 6570 742d 456e
0x0090:  636f 6469 6e67 3a20 6964 656e 7469 7479
0x00a0:  0d0a 486f 7374 3a20 3137 322e 3230 2e31
0x00b0:  2e31 0d0a 436f 6e6e 6563 7469 6f6e 3a20
0x00c0:  4b65 6570 2d41 6c69 7665 0d0a 0d0a
```

トラヒック具体例★ *(cont.)*

tcpdump の解析出力

```
20:02:55.656904 IP (tos 0x0, ttl 63, id 0, offset 0, flags [DF],
 proto TCP (6), length 192)
    172.20.254.1.52315 > 172.20.1.1.http: Flags [P.],
    cksum 0x6159 (correct), seq 1:141, ack 1, win 2053,
    options [nop,nop,TS val 704353773 ecr 1776307967], length 140:
    HTTP, length: 140
       GET / HTTP/1.1
       User-Agent: Wget/1.16.3 (darwin14.1.0)
       Accept: */*
       Accept-Encoding: identity
       Host: 172.20.1.1
       Connection: Keep-Alive
```

トラヒック具体例★ *(cont.)*

テクニック

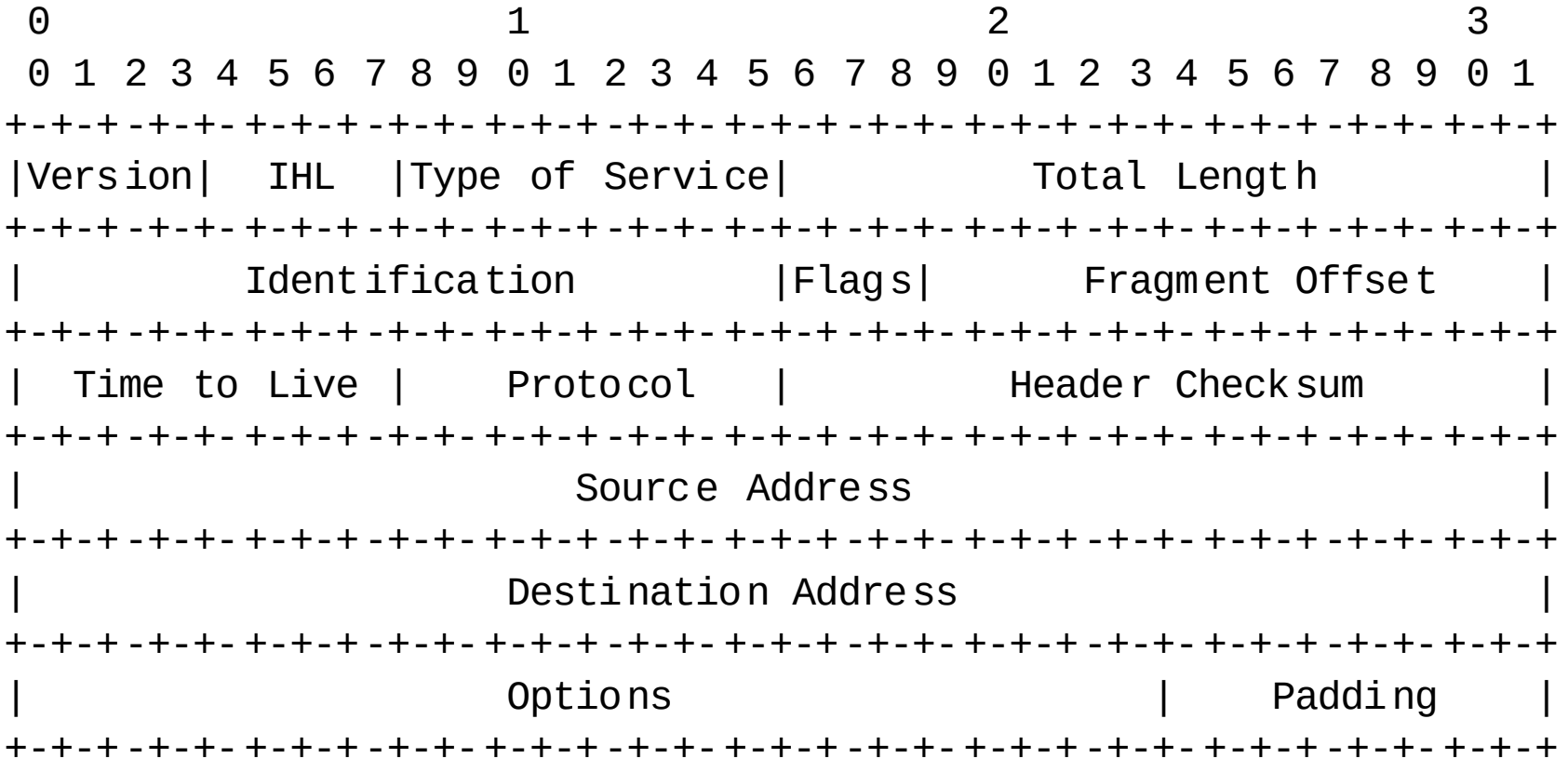
- 45 を探す // IPv4先頭, HTTP GET リクエスト
- 0d 0a を探す // テキストの行末

注意点（ネットワークバイトオーダ）

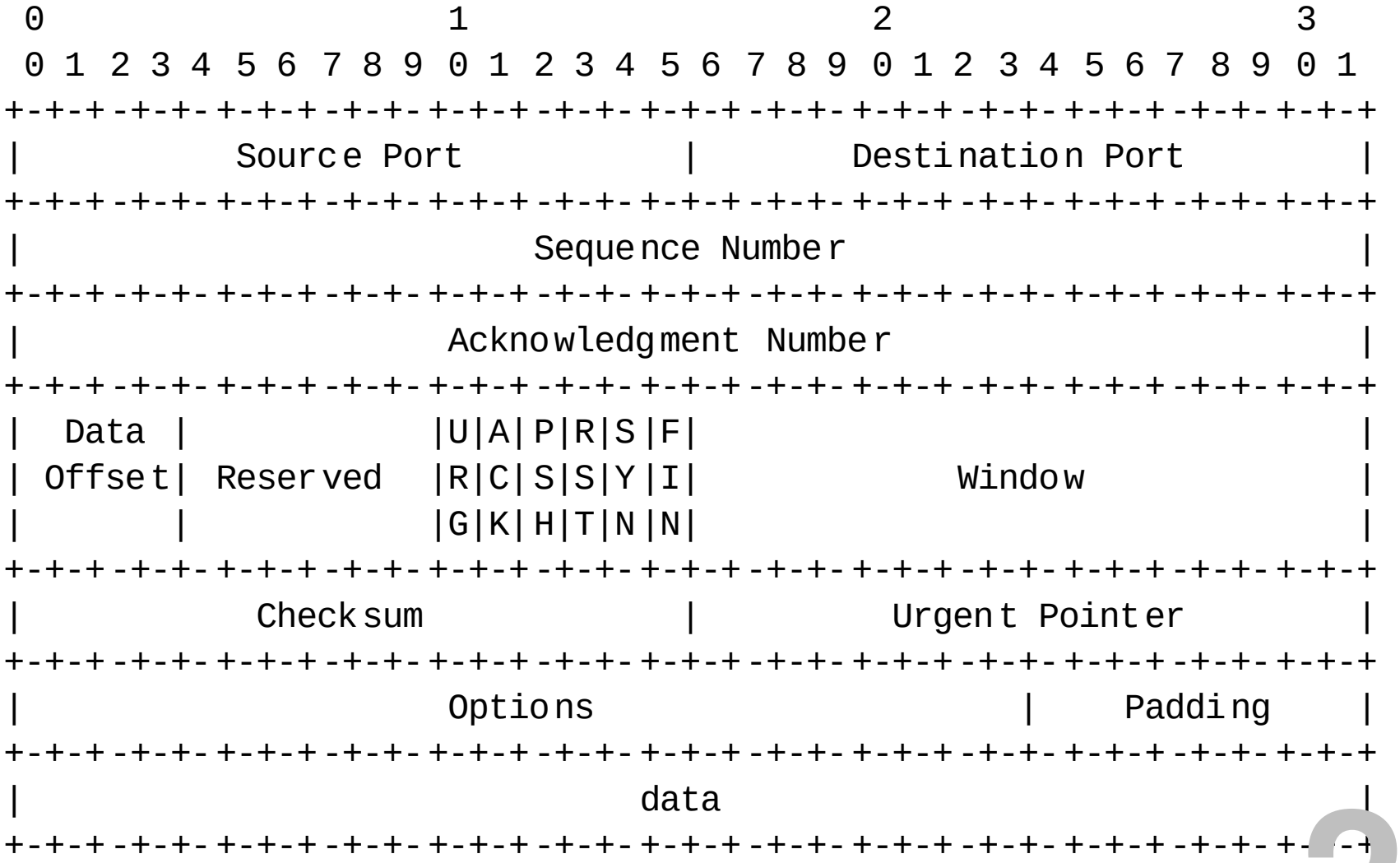
- バイトオーダはハード・ソフト依存
- TCP/IP ではビッグエンディアン

0x1234 → 0x1234 ビックエンディアン
→ 0x3412 リトルエンディアン

The Format of IP Header (RFC791)



The Format of TCP Header (RFC793)



プロトコル番号、ポート番号

EtherType, Protocol, Port の情報（IANA）

```
https://www.iana.org/assignments/
    ieee-802-numbers/
    ieee-802-numbers.xhtml
https://www.iana.org/assignments/protocol-numbers/
    protocol-numbers.xhtml
https://www.iana.org/assignments/
    service-names-port-numbers/
    service-names-port-numbers.xhtml
```

※ /etc/services にも主要なポート番号は載っている

```
0x0000:  ac1f 6b1d 7b8a 0030 4885 f5c9 0800 4500
         D-MAC          S-MAC          IPv4v4
0x0010:  00c0 0000 4000 3f06 e40c ac14 fe01 ac14
         192B           TTLTCP    S-IP      D-IP
0x0020:  0101 cc5b 0050 da6e 5c84 6fa7 ffa7 8018
         D-IP S-Pr HTTP Seq       Ack
0x0030:  0805 6159 0000 0101 080a 29fb 95ed 69e0
                        N N  TSval
0x0040:  4eff 4745 5420 2f20 4854 5450 2f31 2e31
         val  G E  T ▲  / ▲  H T  T P  / 1  . 1
0x0050:  0d0a 5573 6572 2d41 6765 6e74 3a20 5767
         \r\n U s  e r  - A  g e  n t  : ▲  W g
0x0060:  6574 2f31 2e31 362e 3320 2864 6172 7769
         e t  / 1  . 1  6 .  3 ▲  ( d  a r  w i
0x0070:  6e31 342e 312e 3029 0d0a 4163 6365 7074
         n 1  4 .  1 .  0 )  \r\n A c  c e  p t
0x0080:  3a20 2a2f 2a0d 0a41 6363 6570 742d 456e
         : ▲  * /  * \r \nA  c c  e p  t -  E n
0x0090:  636f 6469 6e67 3a20 6964 656e 7469 7479
         c o  d i  n g  : ▲  i d  e n  t i  t y
0x00a0:  0d0a 486f 7374 3a20 3137 322e 3230 2e31
         \r\n H o  s t  : ▲  1 7  2 .  2 0  . 1
0x00b0:  2e31 0d0a 436f 6e6e 6563 7469 6f6e 3a20
         . 1  \r\n C o  n n  e c  t i  o n  : ▲
0x00c0:  4b65 6570 2d41 6c69 7665 0d0a 0d0a
         K e  e p  - A  l i  v e  \r\n \r\n
```

IPオプション付き

▲はスペース

NOTE

- インターフェイス、IPアドレス確認
  ifconfig; ip
- 通信内容確認（管理者権限必要）
  tcpdump; wireshark

---

tcpdump に似たプログラムを作ると理解が深まる
libpcap を利用
https://www.tcpdump.org/

HTTP 認証: Basic (RFC2617)

基本だが認証内容が筒抜け

1) クライアントから資源要求

```
GET /d1/a.html HTTP/1.1
```

2) サーバから未認証と応答

```
HTTP/1.1 401 Unauthorized
Date: Mon, 11 Mar 2019 09:22:29 GMT
Server: Apache/2.4.18 (Ubuntu)
WWW-Authenticate: Basic realm="d1"
```

HTTP 認証: Basic (RFC2617) *(cont.)*

3) 認証情報付けて再度要求

```
GET /d1/a.html HTTP/1.1
＜略＞
Authorization: Basic YWxpY2U6cmFiaXQ=
```

単純な符号化（可逆変換）で読み出し可能

```
$ echo  "YWxpY2U6cmFiaXQ=" |base64 -d
alice:rabit
```

HTTP 認証: Digest

通信回数は同じ、パスワード流れない

```
GET /d2/a.html HTTP/1.1
```

```
HTTP/1.1 401 Unauthorized
<略>
WWW-Authenticate: Digest realm="d2",
 nonce="YYVmvc6DBQA=f3dd78ecafa98e8ab12a09e15e42f31c5debc485",
 algorithm=MD5, domain="/d2/", qop="auth"
```

```
GET /d2/a.html HTTP/1.1
<略>
Authorization: Digest username="bob", realm="d2",
 nonce="YYVmvc6DBQA=f3dd78ecafa98e8ab12a09e15e42f31c5debc485",
 uri="/d2/a.html", algorithm=MD5,
 response="78ddcd5dacf38fdfa422d50d2aab7791", qop=auth,
 nc=00000001, cnonce="35877305543aba06"
```

HTTP 認証: Digest *(cont.)*

- nonce はサーバが作るランダムな文字列
- cnonce はクライアントが作るランダムな文字列
- response は計算結果のハッシュ値（今回は MD5）
- サーバ側で同じように計算
- 計算結果が一致すると認証成功

パスワードが直接流れているわけではない

※ 詳細は割愛

HTTPS — HTTP の暗号化

HTTPS (Hypertext Transfer Protocol Secure)
HTTP over TLS (RFC2616, 2818)
TLS (The Transport Layer Security Protocol, RFC4346,5246)

- HTTP レベルの暗号化 （TCP, IPは平文）
- サーバ側に公開鍵、証明書が必要
- 内容解析困難
- ただし、証明書が適切でない場合は注意
  - ◇ 偽造や設定ミスのおそれ

HTTPS — HTTP の暗号化 *(cont.)*

1 C→S （443ポート; 0x01bb）

```
0x0000:  0000 5e00 0118 88e9 fe51 a03b 0800 4500
0x0010:  0040 0000 4000 4006 85cb 0a1b 1f09 172a
0x0020:  749f d160 01bb 93fb 18cc 0000 0000 b0c2
0x0030:  ffff 8378 0000 0204 05b4 0103 0306 0101
0x0040:  080a 2e7c 4f77 0000 0000 0402 0000
```

2 S→C

```
0x0000:  88e9 fe51 a03b 001a 1e00 c9d0 0800 4500
0x0010:  003c 0000 4000 3906 8ccf 172a 749f 0a1b
0x0020:  1f09 01bb d160 3595 3ba8 93fb 18cd a012
0x0030:  7120 1ecd 0000 0204 05b4 0402 080a 8ece
0x0040:  0532 2e7c 4f77 0103 0307
```

HTTPS — HTTP の暗号化 *(cont.)*

3 C→S

```
0x0000:  0000 5e00 0118 88e9 fe51 a03b 0800 4500
0x0010:  0034 0000 4000 4006 85d7 0a1b 1f09 172a
0x0020:  749f d160 01bb 93fb 18cd 3595 3ba9 8010
0x0030:  080a b69c 0000 0101 080a 2e7c 4f8a 8ece
0x0040:  0532
```

HTTPS — HTTP の暗号化 *(cont.)*

4 S→C

```
0x0000:  0000 5e00 0118 88e9 fe51 a03b 0800 4500
0x0010:  017e 0000 4000 4006 848d 0a1b 1f09 172a
0x0020:  749f d160 01bb 93fb 18cd 3595 3ba9 8018
0x0030:  080a f5a1 0000 0101 080a 2e7c 4f8a 8ece
0x0040:  0532 1603 0101 4501 0001 4103 036a 8d7d
0x0050:  5d09 664b 4125 0d07 8f52 d6da e0b2 4b5b
0x0060:  9f5b d744 dbdc e5b9 6f73 94ee 3e00 00ac
0x0070:  c030 c02c c028 c024 c014 c00a 00a5 00a3
0x0080:  00a1 009f 006b 006a 0069 0068 0039 0038
0x0090:  0037 0036 0088 0087 0086 0085 c032 c02e
0x00a0:  c02a c026 c00f c005 009d 003d 0035 0084
0x00b0:  c02f c02b c027 c023 c013 c009 00a4 00a2
0x00c0:  00a0 009e 0067 0040 003f 003e 0033 0032
0x00d0:  0031 0030 009a 0099 0098 0097 0045 0044
0x00e0:  0043 0042 c031 c02d c029 c025 c00e c004
```

HTTPS — HTTP の暗号化 *(cont.)*

```
0x00f0:  009c 003c 002f 0096 0041 0007 c011 c007
0x0100:  c00c c002 0005 0004 c012 c008 0016 0013
0x0110:  0010 000d c00d c003 000a 00ff 0201 0000
0x0120:  6b00 0000 1200 1000 000d 7777 772e 6173
0x0130:  6168 692e 636f 6d00 0b00 0403 0001 0200
0x0140:  0a00 1c00 1a00 1700 1900 1c00 1b00 1800
0x0150:  1a00 1600 0e00 0d00 0b00 0c00 0900 0a00
0x0160:  2300 0000 0d00 2000 1e06 0106 0206 0305
0x0170:  0105 0205 0304 0104 0204 0303 0103 0203
0x0180:  0302 0102 0202 0300 0f00 0101
```

～キャスト

通信相手を示す表現

ブロードキャスト	broadcast	全員（放送）
マルチキャスト	multicast	何人か
エニキャスト	anycast	誰か
ユニキャスト	unicast	一人

ユニキャスト以外はオーバヘッドが大きいので、広い範囲では多用しない傾向にある

～キャスト *(cont.)*

アドレスで範囲を示す場合が多い（IPの例）

範囲	ブロードキャストアドレス
全体	255.255.255.255（理論上）
150.65.1.1/16	150.65.255.255
150.65.1.1/24	150.65.1.255

150.65.1.1	10010110 01000001 00000001 00000001
150.54.1.255	10010110 01000001 00000001 11111111
150.54.255.255	10010110 01000001 11111111 11111111

NOTE

ビット展開の方法は様々

- 手計算
- 関数電卓
- スクリプト言語

```
% perl -e 'printf "%08b %08b %08b %08b\n",150,65,255,255'
10010110 01000001 11111111 11111111
```

16進数も同様

```
% perl -e 'printf "%02x %02x %02x %02x\n",150,65,255,255'
96 41 ff ff
```

～キャスト *(cont.)*

イーサネットのブロードキャスト

全体 ff:ff:ff:ff:ff:ff

TCP/IP関連プロトコル: Domain

Domain (RFC 1034,1035)

- DNS(Domain Name Server) とも呼ばれる
- 名前から IP アドレスを引くプロトコル
- 名前を引くプログラムは resolver と呼ばれる
  - 多くの OS では設定ファイル /etc/resolv.conf
  - 多くの場合は DHCP で得た設定
- 組織のサーバ、あるいは外部のサーバと通信
  - 再帰的に探索する

TCP/IP関連プロトコル: ARP

ARP (Address Resolution Protocol; RFC826)

- プロトコルアドレス、ハードウェアアドレスの対応を応答するプロトコル

  ※ 実質的にはIPアドレスとMACアドレス
- IPアドレスがわからない時点で使う
  - ◇ イーサネットレベルのブロードキャスト
- IP通信の前に発生する
- 一定期間（伝統的には30秒）だけ記憶する
  - ◇ 通信時間に応じて自動的に発生する

TCP/IP関連プロトコル: ICMP

ICMP (Internet Control Message Protocol; RFC792)

- 制御のためのプロトコル
- エラー報告が中心
  - 未到達
  - TTL 超過
- 疎通チェックの側面も持つ
  - ping や tracreroute

TCP/IP関連プロトコル: DHCP

DHCP (Dynamic Host Configuration Protocol; RFC2131,2132)

- 自動的にネットワークが使えるようになる仕組み
  - OS 起動やネットワーク変化時に動作
- BOOTP (Bootstrap Protocol; RFC951)の延長
- ホストを設定するプロトコル
  - 主に IPアドレス、ゲートウェイ、DNS
  - 他に、ホスト名、ログ、...
- 設定適用範囲はクライアント側が決める

TCP/IP関連プロトコル: DHCP *(cont.)*

- 向きによってポート異なる
  - サーバ発67、クライアント発68

src	*dest*	*desc*
C-host.68	S-host.67	Request
S-host.67	C-host.68	Reply

  - アドレス不明なので、ブロードキャストを利用
- クライアントのサーバ発見(DISCOVER)し
- サーバが設定を提案(OFFER)する

観測のコツ

流れを考えると観測しやすい

- （最初）DHCP が発生

  ...

- （前）Domain, ARP が発生
- （本体）TCP, UDP が発生

  ...

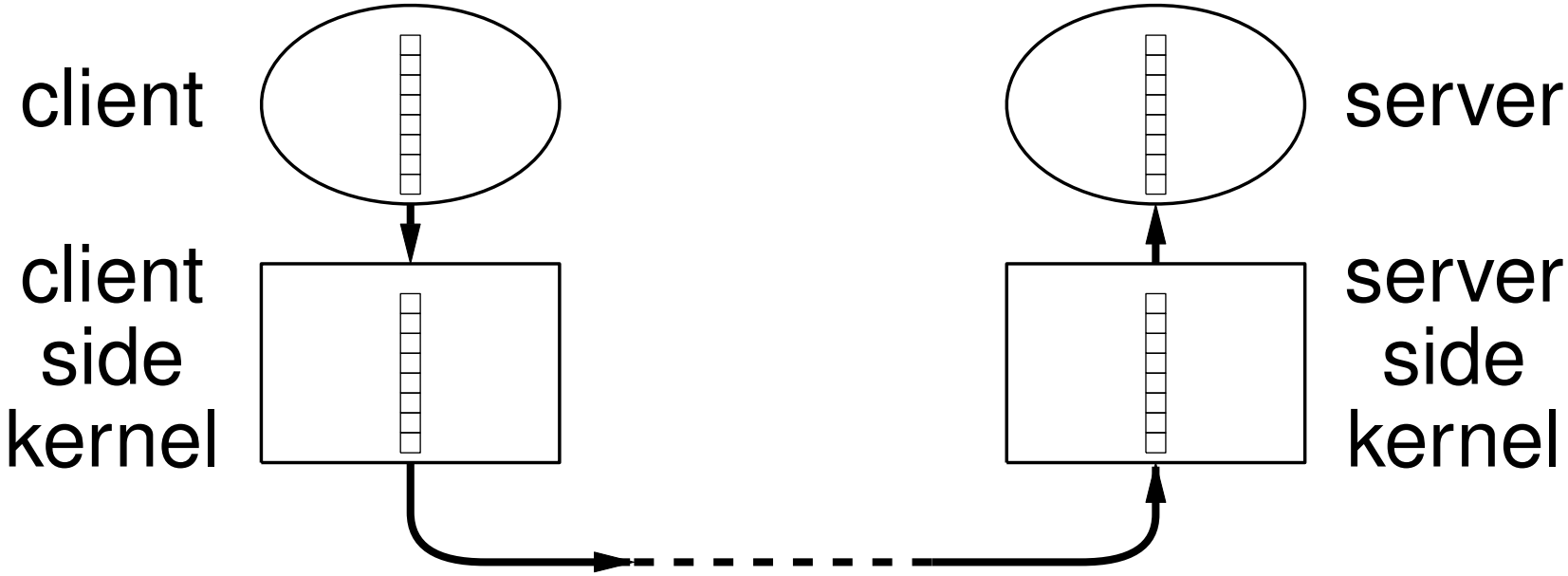
- （脇）ICMP でエラー到達
- （脇）ARP や DHCP で更新

NOTE

- MACアドレステーブルを確認
  arp
- エラーを確認
  ping; traceroute
- 名前からIPアドレス検索
  dig; nslookup

待ち行列（キュー； **Queue**）

様々な箇所に待ち行列がある



※一般論、詳細は実装による

待ち行列（キュー； **Queue**） ***(cont.)***

待ち行列の特徴

- 平滑化:
  - ◇ 処理の瞬間的な集中を緩和（吸収）
  - ◇ 処理ができない場合に一旦保管する
  - ◇ ただし、処理が追いつかない場合は破綻
- FIFO (first in, first out)

飽和攻撃に弱い

- DoS, DDoS, ... SYN attack

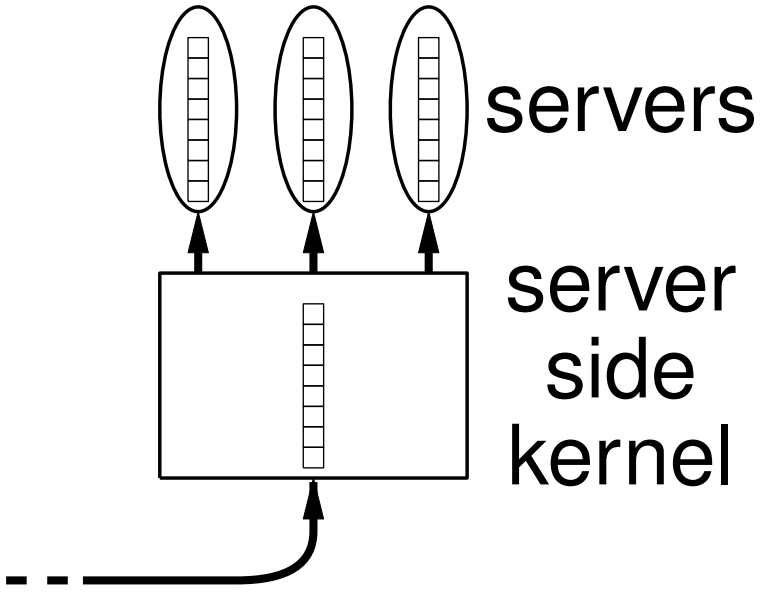
待ち行列（キュー； **Queue**） ***(cont.)***

待ち行列の例

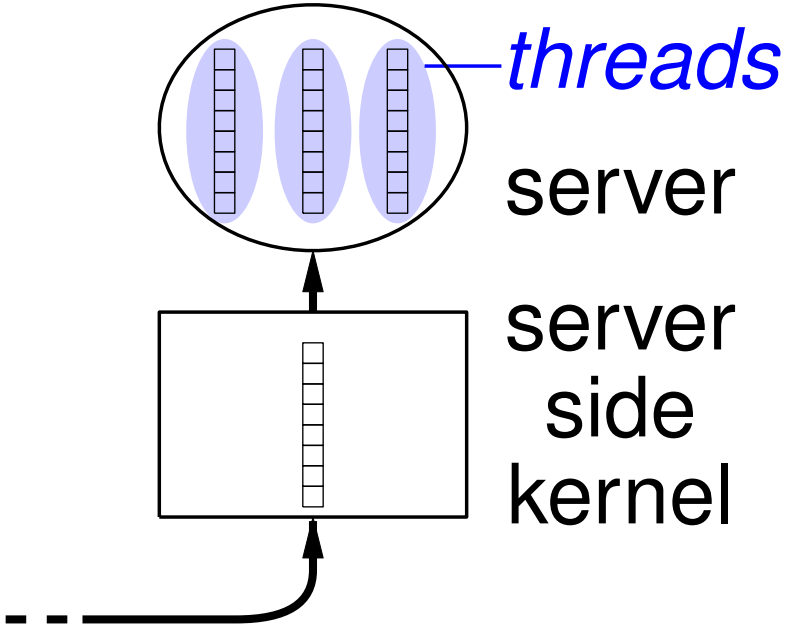
- カーネル
  - connect 待ち（listening）ソケット
  - 確立済ソケット
- サーバ、クライアント
  - 読み込み
  - 書き込み
  - （ある意味、プロセスやスレッドも）

待ち行列（キュー； **Queue**） ***(cont.)***

マルチプロセス



マルチスレッド



別に1コンテキスト複数待ち行列の設計も可

待ち行列（キュー； **Queue**） ***(cont.)***

関連用語

緩衝（バッファ**; buffer**）**:**

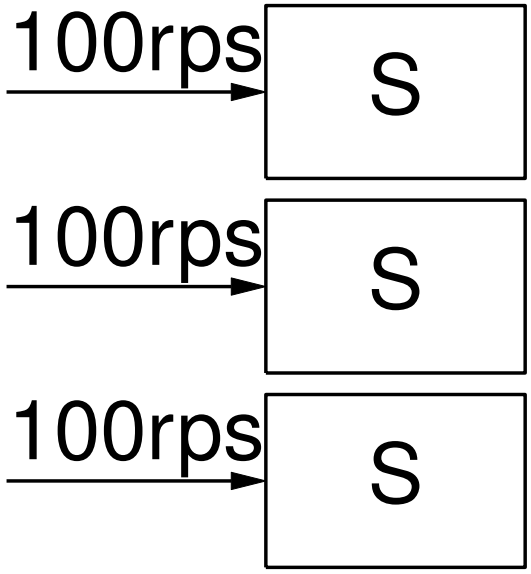
- 平滑化の機構
- 多くの場合 FIFO

余白（マージン**; margin**）**:**

- （性能のコンテキストでは）限界からの余裕
- 装置の限界ギリギリで運用しない
  - 8割、半分などをかけて考える

性能計算例

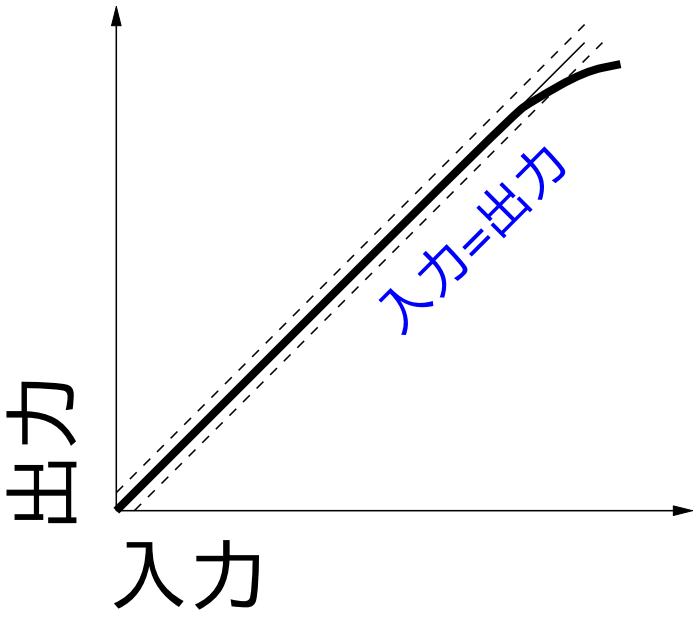
毎秒100リクエスト（100 request/second; 100 rps）
さばけるサーバが 3台



- 全体限界は 300rps
- 300rps 超えたら危険
- 8割運用なら 240rpsまで
- 240rps 超えたら増強
- 1 台故障したら、全体限界は 200rps に低下
- 1台故障＋8割運用 160rps

性能計算例 *(cont.)*

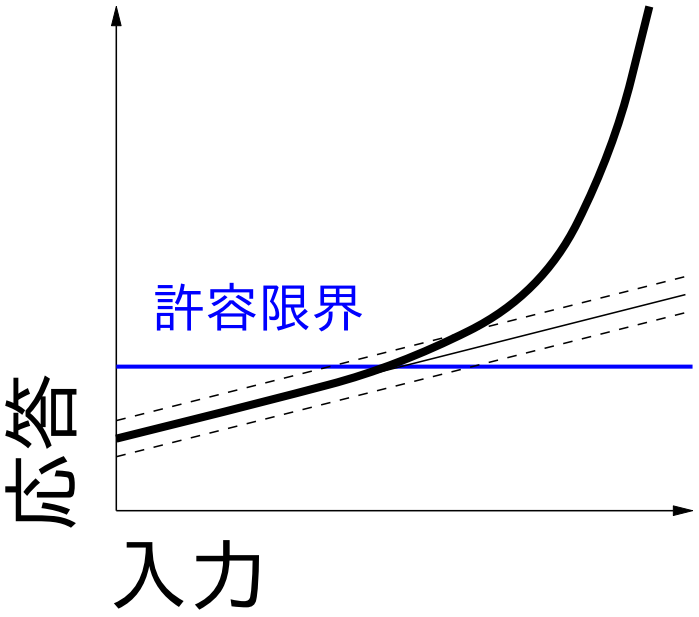
入出力相関曲線



- 入力＝出力 の線から離れたら性能劣化
- 入力＝出力の線を引く
- 幅をもって補助線を引く
  - ◇ ばらつきがあるため
- 限界点を決める
- 運用許容点を決める

性能計算例 *(cont.)*

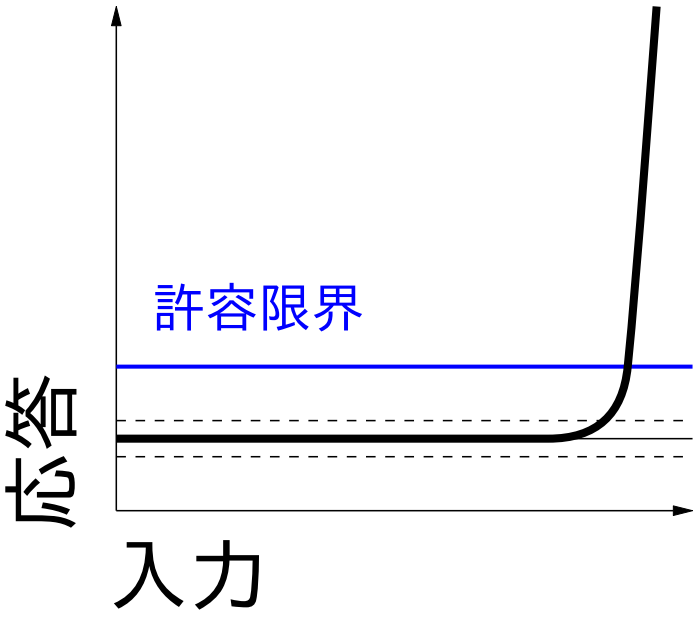
応答は、多くの場合、下に凸の右上がりグラフになる



- 応答が著しく長くなる箇所は避ける
- 接線を引いて考える
- 幅をもって補助線を引く
- （あれば）応答許容限界を引く

性能計算例 *(cont.)*

理想的にはギリギリまで応答一定が望ましい



- ここまで綺麗な曲線はなかなかない
- 変曲点; 曲率が変化する点
  - ◇ 肘（elbow）
  - ◇ 膝（knee）

NOTE

- SYN の待ち行列調査（Linux の場合）

```
$ sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 2048
```

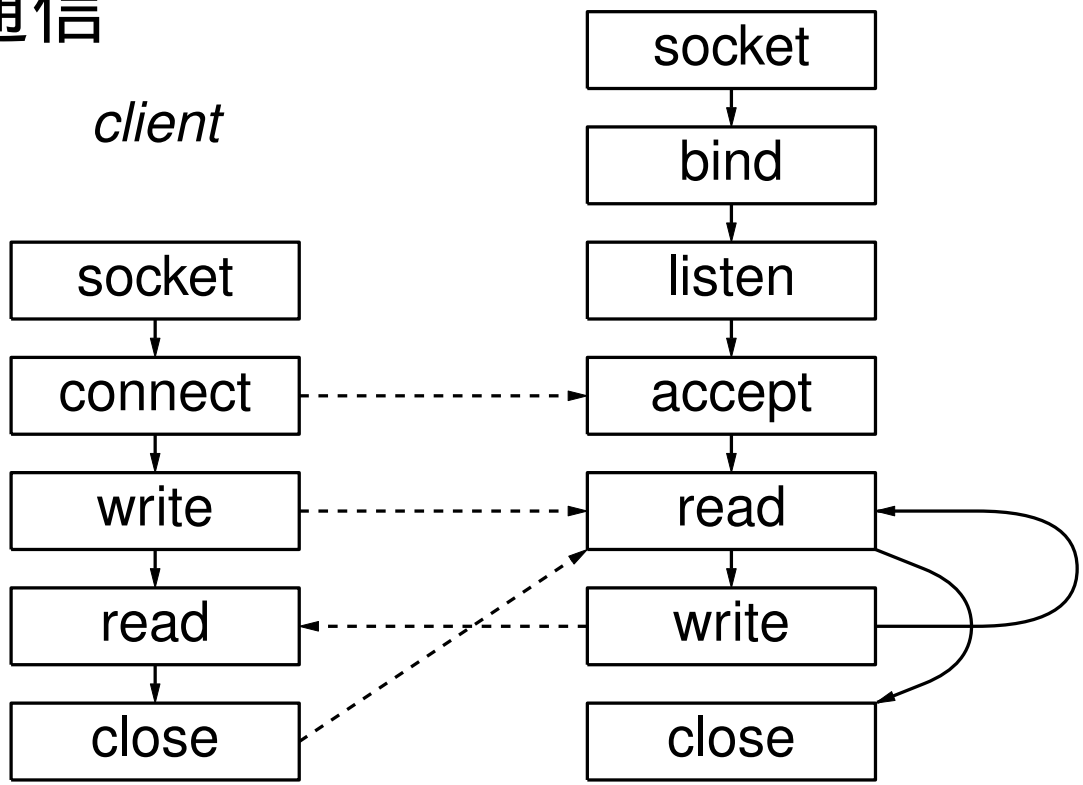
Solaris は ndd などを使う

通信の研究では待ち行列は重要

ソケット **API ; socket API**

システムコールで考える

例: HTTP 通信



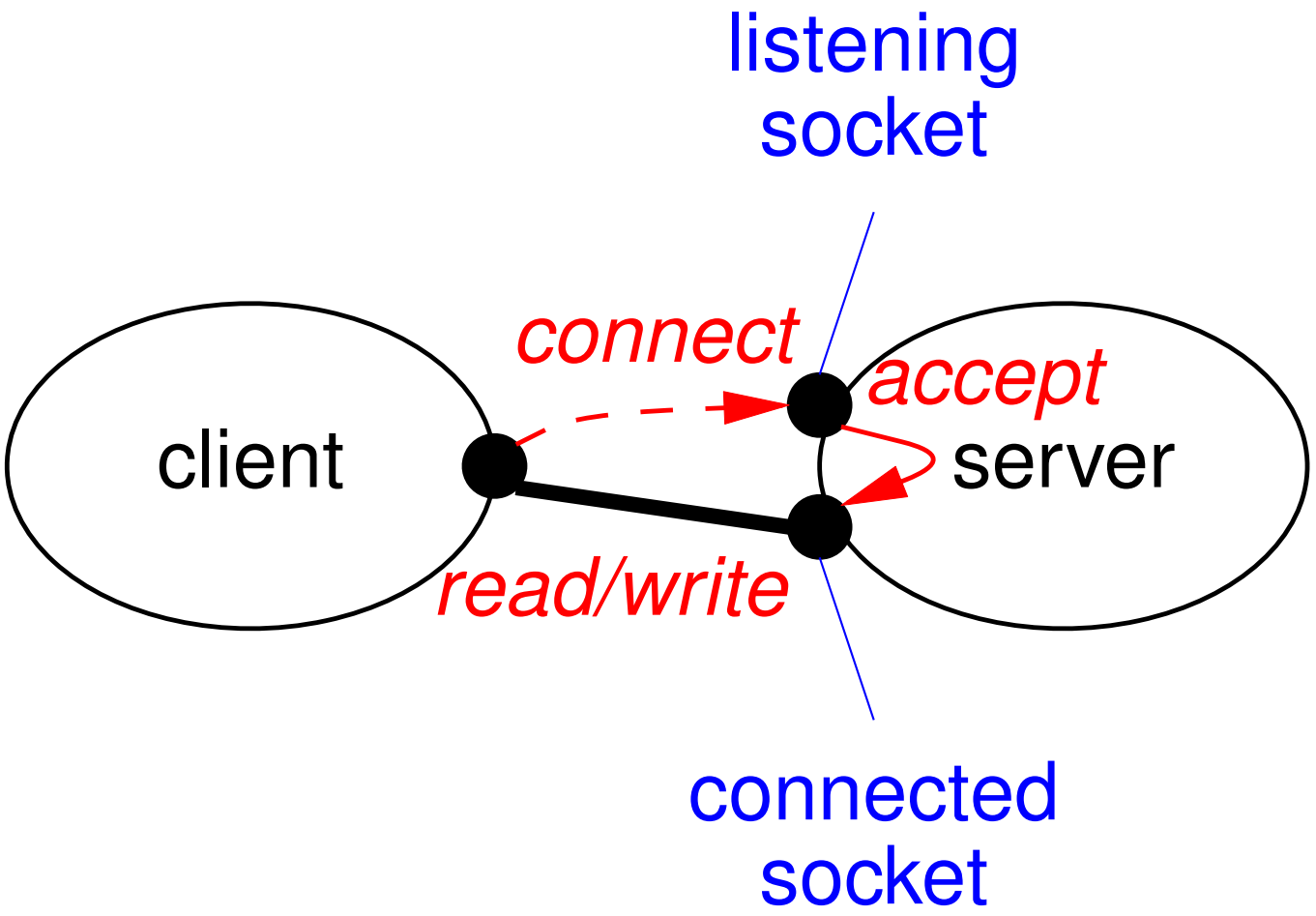
クライアント

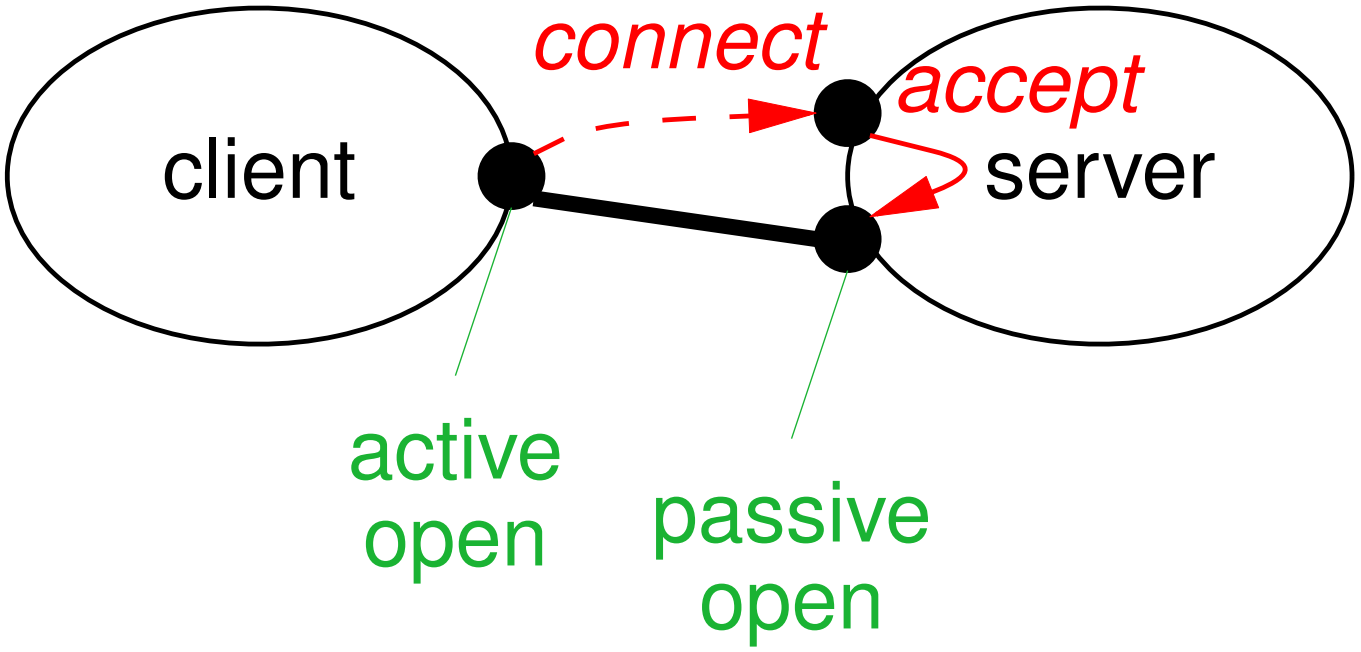
「ソケット」は概念
「システムコール」は OS の機能
　　一部のプラットホームではライブラリ

- socket でソケット作成
- connect で接続
- write, read でデータ交換
  - 順番は対象プロトコルによる
  - 交換内容もプロトコルによる
- close で接続閉じる

サーバ

- socket でソケット作成
- bind でアドレス割り当て
- listen で OS に登録
- accept で接続受付
  - 受け付けると二つ目のソケット生成
- read, write でデータ交換
  - 順番や内容は対象プロトコルによる
- close で接続閉じる


listening
socket
connect
accept
client
server
read/write
connected
socket


connect
accept
client
server
active
open
passive
open

コネクション

- TCP では接続や通信路は「コネクション」と呼ぶ
- 接続する方を「アクティブオープン」
- 受ける方を「パッシブオープン」

★ ★ ★

- ファイルと同様に扱えるよう抽象化
  - ファイルディスクリプター (file descriptor)
  - 0以上の整数

NOTE

- socket (file descriptor) 確認
  lsof; netstat
  - ◇ サーバ側が LISTEN になっているか
  - ◇ 通信を繰り返してサーバ側の変化
    - ⋆ fd番号
    - ⋆ プロセス番号

NOTE

- 便利ツール netcat
  - 簡易サーバ
  - 簡易クライアント

マルチコンテキスト **multi-context**

コンテキスト種類

プロセス **process:**

プログラム実行の単位

メモリは独立

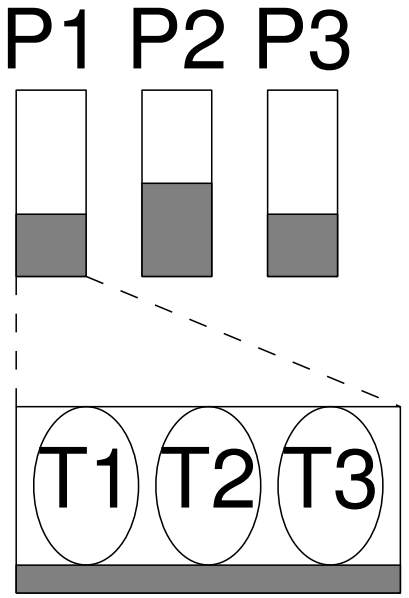
スレッド **thread:**

歴史的にプロセスの後にできた

プロセス内の実行単位

メモリを共有

トラブル発生時は道連れ



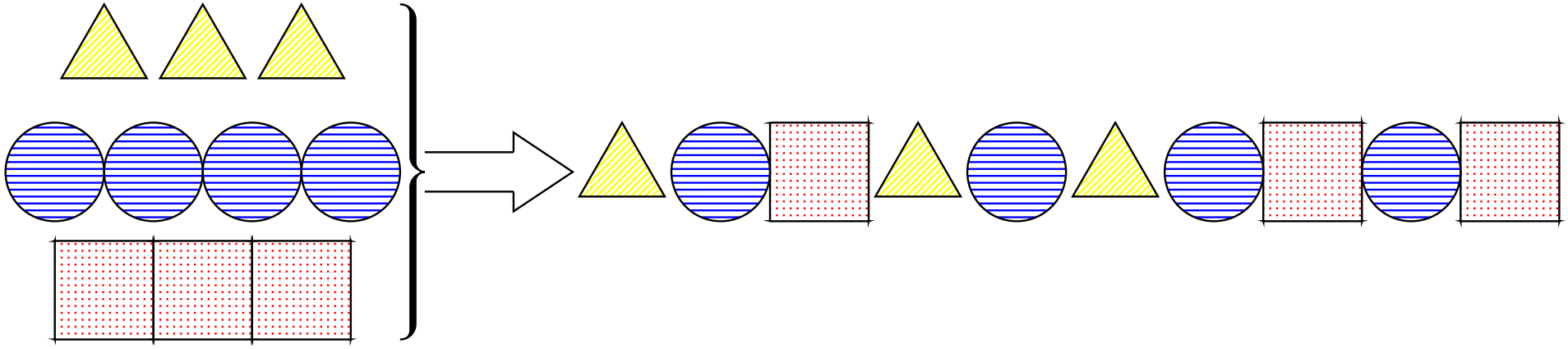
マルチコンテキスト **multi-context** ***(cont.)***

	process	**thread**
create	fork	pthread_create
purge	exit	pthread_exit
wait	wait	pthread_join

マルチコンテキストの通信の追跡は困難

マルチコンテキスト **multi-context** ***(cont.)***

各コンテキストの通信が混ざる



NOTE

- プロセスを調べる

  ps
- スレッドを調べる

  ps

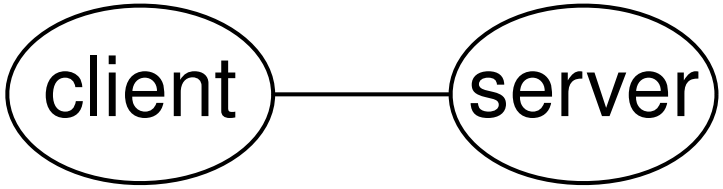
  スレッドに関するオプションは様々、各自が使っている OS で調べること。

  man ps

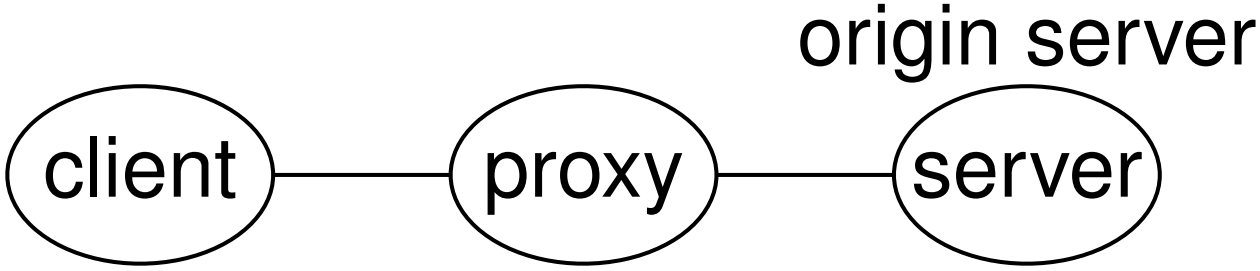
  例えば ps -eLf ; ps -ejH など

プロキシサーバ（**proxy server**）

基本**:** クライアント/サーバ

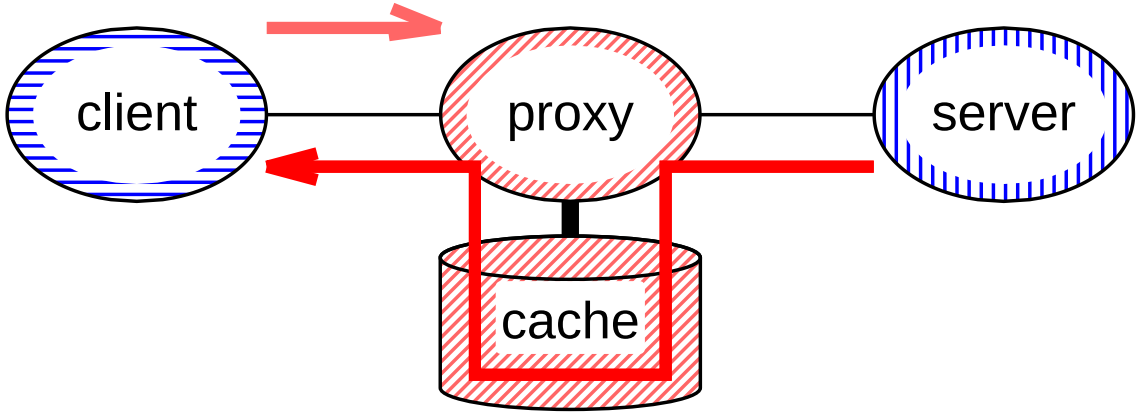


応用**:** クライアント/サーバ＋プロキシ

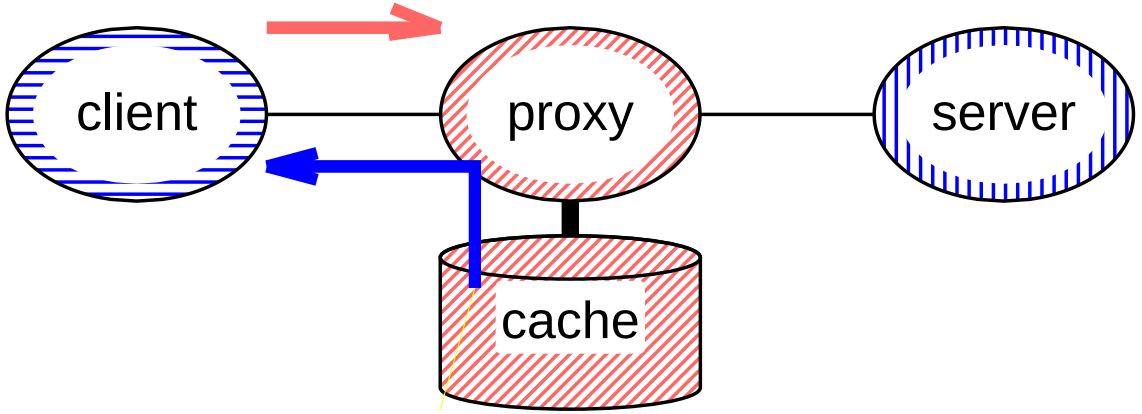


※ 集約や障壁乗り越えのため中継する

プロキシとキャッシュ



ミス



ヒット

キャッシュ

- キャッシュにヒットすると外部への通信削減
  → 帯域節約、応答時間短縮
- メモリやディスクは有限
- インターネットの資源は膨大
  → 全ては保存できない前提で設計する
- 置換アルゴリズムが重要; LFU, LRU, ...
- ただし、キャッシュ不能・無効なコンテンツもある

※ 万能ではない

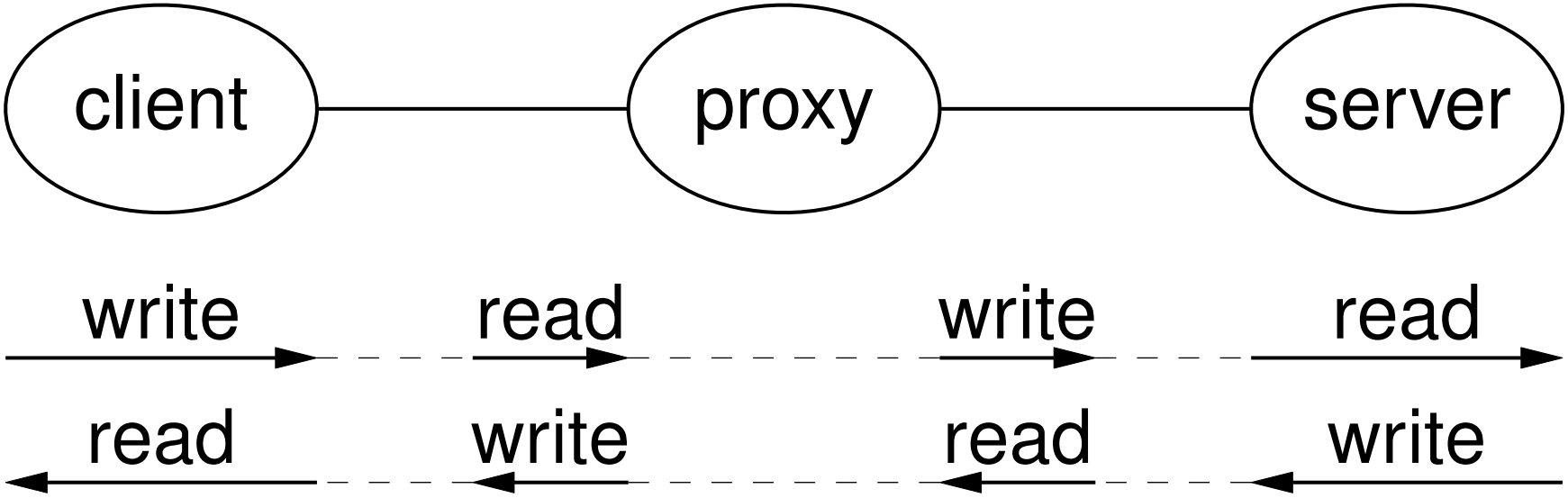
プロキシ **v.s.** 暗号通信

- 単純には暗号通信は中継できない
- 鍵が手に入るとは限らない

  → WWW では HTTPS が増えてプロキシ減少
  → キャッシュの効果が得られない

プロキシと複数通信

クライアント向け、サーバ向けの二方向の通信を扱う



→ マルチコンテキストや非同期通信が必要

マルチコンテキスト

読み込み/書き込みの担当コンテキストを作る
（HTTPなら）

- クライアントからリクエスト読み込む
- サーバに書き込む
- サーバから読み込む
- クライアントに読み込む

一度に一つの操作
並列な操作はできないので、コンテキストを増やす

→ コンテキストが多数発生

同期/非同期通信

同期通信

- 読み込み時データが届くまで待つ（block）
- あるコネクションのデータを待つ際に、他のコネクションのデータが届いても扱えない

非同期通信

- データの届いたコネクションから読み出す
- データの届いたコネクションを探して、届いた分を読み出す

コネクション監視

ポーティング上は select が無難

伝統的	select	POSIX (BSD)
	poll	POSIX (SYS-V)
最新技術	kqueue	BSD
	epoll	Linux
	/dev/poll	Solaris

libevent など網羅的なライブラリも登場している

select システムコール

コネクション集合から、読み込み可能(readble)、書き込み可能（writable）なコネクションを検出

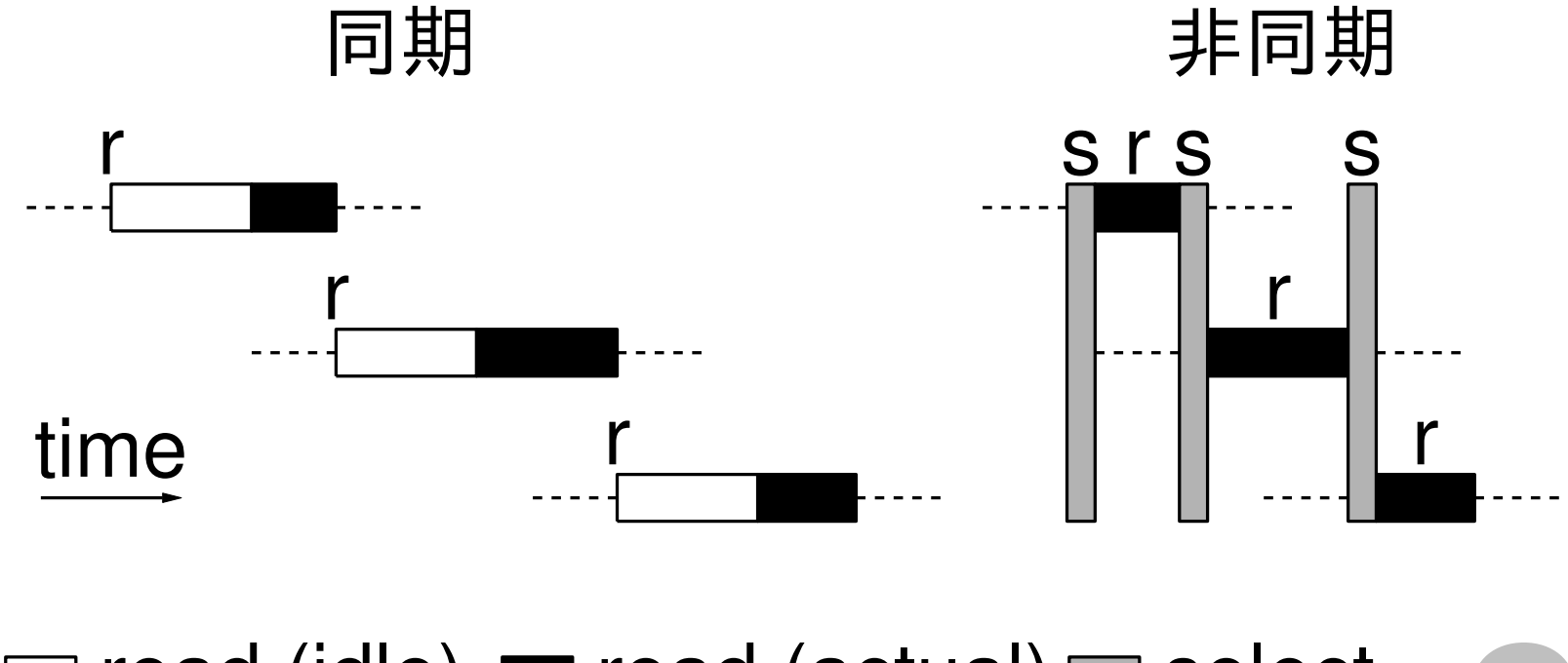
- 読み込み可能 → read, accept
- 書き込み可能 → write, connect

ブロック回避できる
⇒ 1コンテキストで複数コネクション扱える
プロキシなどで有効

※ 他のコネクション監視は、データの扱いや処理の呼び出しなどが異なる

select システムコール ***(cont.)***

ブロックしている時間（待ち時間）の有効利用
（3コネクション例）



待ち時間削減

- 待ち時間 = CPU は暇
  - ◇ read（前述）
  - ◇ connect
  - ◇ ホスト名検索 ; gethostbyname など
- ネットワーク以外でも待ち時間は存在
  - ◇ ユーザの入力待ち
  - ◇ ディスク
- （素直に待たず）その間に別の処理をする
- このような待ち時間削減が高速化に重要

待ち時間削減 *(cont.)*

注意点

- コネクション状態監視をやりすぎてはいけない
  - コネクション監視で CPU を浪費しない
  - busy loop
- 全力疾走だけでなく、息継ぎも必要
  - 変化が出るまで待つ // 待ち時間無限
  - 短い待ち時間を指定

サーバ防御手段（典型的）

リクエスト増加

- 高性能ハード投入
- ハード台数追加
- ロードバランサ投入

- コンテキスト数拡大; ワーカプロセスなど
- TCP パラメータチューニング; タイムアウトなど

- コンテンツ数、量を縮小

サーバ防御手段（典型的） *(cont.)*

サニタイズ（sanitize）

- 機能を整理
  - ◇ 用途不明の箇所作らない
- ネットワークを整理
  - ◇ 外部/内部通信を混ぜない
- ユーザ権限を整理
- 状況把握機構を導入
  - ◇ トラヒックやサーバ稼働を監視