先端エレクトロニクスDAQセミナー2020

～ ソフトウェア技術 ～

データ収集システム化技術（ネットワーク技術）

広島工業大学

長坂 康史

nagasaka@cc.it-hiroshima.ac.jp

目的と達成目標

- 目的
  - データ収集システムを構築するためのシステム化技術、特に、ネットワーク技術を修得する
- 達成目標
  - データ収集システムを構築するために必要なネットワーク技術を理解し、活用することができる

目次

- ネットワーク技術
  - 通信プロトコル
    - OSI基本参照モデル
    - TCP/IPプロトコル
    - TCPの機能
  - インターネットの標準

ネットワーク技術
通信プロトコル

通信プロトコルアーキテクチャ

- 通信プロトコル（Communication Protocol）
  - 通信規約・通信手順
  - 通信を実現するための詳細な決まり

- 通信プロトコルアーキテクチャ（プロトコル階層）
  - OSI基本参照モデル
    - OSI (Open System Interconnection)
    - 開放型システム間相互接続
  - 特徴
    - ネットワークを介した通信を実現するための枠組み
    - 階層化されたプロトコル構造
    - プロトコルは各層ごとに独立
    - 開放型システムのアプリケーションプロセスを結ぶ

7	アプリケーション層
6	プレゼンテーション層
5	セッション層
4	トランスポート層
3	ネットワーク層
2	データリンク層
1	物理層

OSI基本参照モデルの階層（１）

- 第１層（物理層）
  - 伝送路の物理的接続の確立や維持管理
  - 電気的条件や手順に関する特性の提供
  - ビット列伝送の保証
- 第２層（データリンク層）
  - ケーブル内での複数ホストからのデータの競合（輻輳）の制御
  - 通信制御とエラー制御
- 第３層（ネットワーク層）
  - データ交換の中継処理と経路制御
- 第４層（トランスポート層）
  - 端末間同士の通信の保証
  - フロー制御
    - 下位階層でのデータが輻輳しないように

7	アプリケーション層
6	プレゼンテーション層
5	セッション層
4	トランスポート層
3	ネットワーク層
2	データリンク層
1	物理層

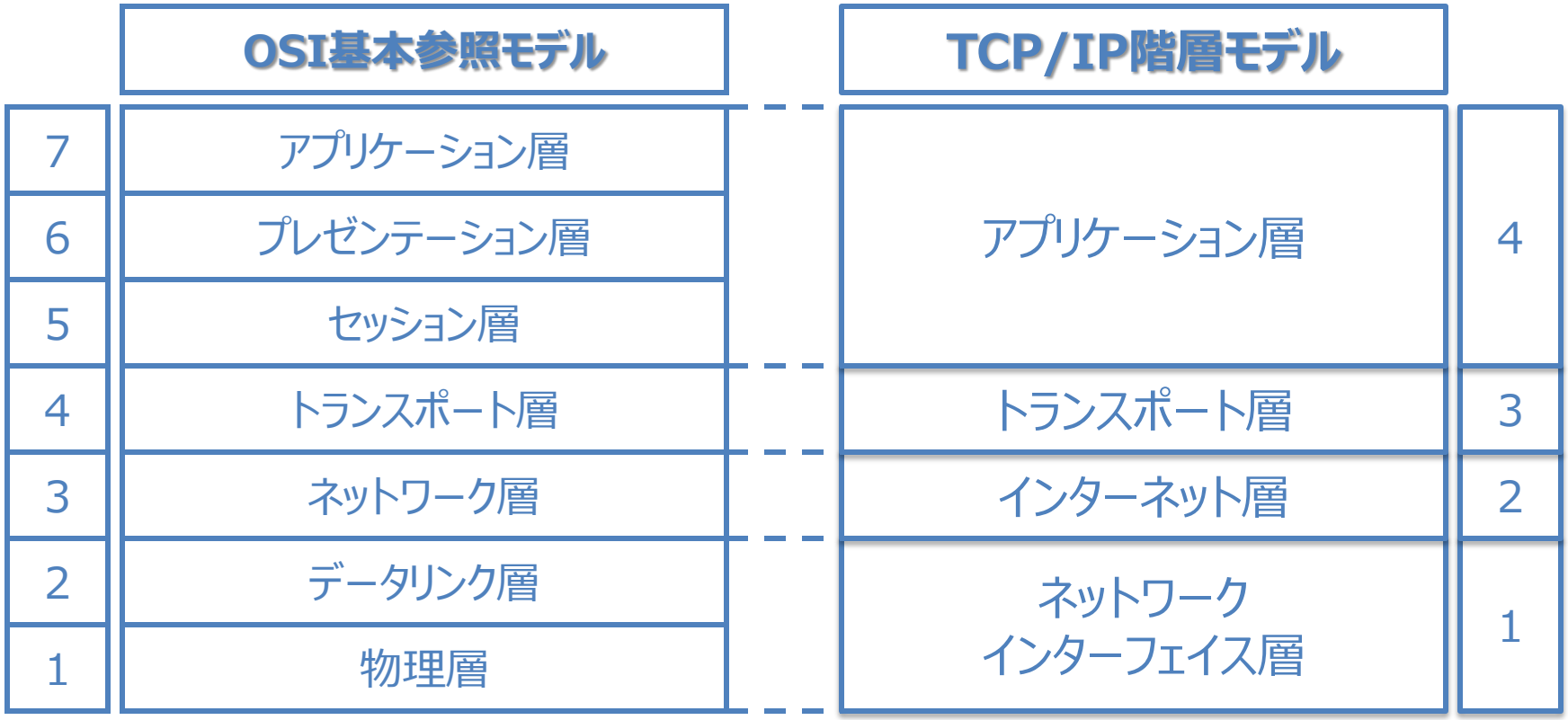
OSI基本参照モデルの階層（２）

- 第５層（セッション層）
  - 情報の伝送方法（半二重、全二重）の制御
  - 同期処理
- 第６層（プレゼンテーション層）
  - コード変換、データの圧縮や暗号化などの情報処理
- 第７層（アプリケーション層）
  - アプリケーション プログラムに対応するデータの処理単位を定義
  - 各種情報サービスの実現

7	アプリケーション層
6	プレゼンテーション層
5	セッション層
4	トランスポート層
3	ネットワーク層
2	データリンク層
1	物理層

TCP/IPの構成

- OSI基本参照モデルとTCP/IP階層モデル



通信プロトコル
TCP/IPプロトコル

インターネット層

- インターネット層（レイヤ２）
  - Internet Layer
  - OSIのネットワーク層（第３層）に対応
  - データの送受信者間の接続に関する規格
    - コンピュータの認識
    - 通信経路の選択
- プロトコル
  - IP（Internet Protocol）
  - ICMP（Internet Control Message Protocol）
  - ARP（Address Resolution Protocol）
  - OSPF（Open Shortest Path First）

トランスポート層

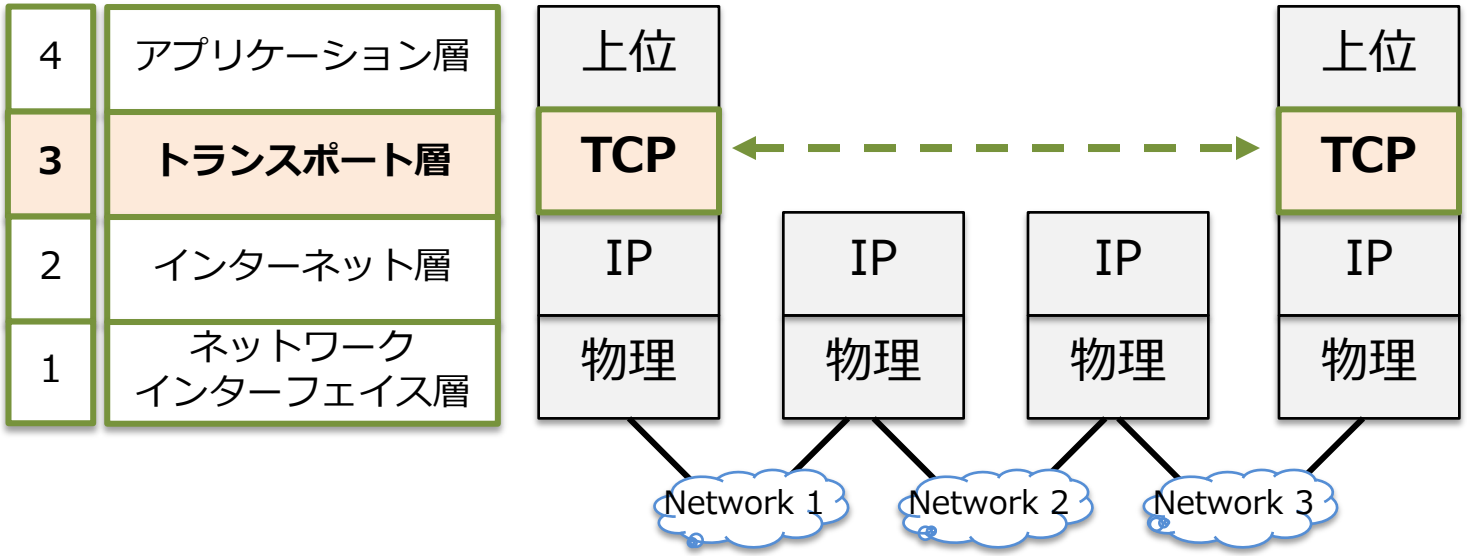
- トランスポート層（レイヤ３）
  - Transport Layer
  - OSIのトランスポート層（第４層）に対応
  - ホスト間でのデータ通信に関わる規格
    - エンドツーエンド（End-to-End 通信）
- プロトコル
  - TCP
    - Transmission Control Protocol
    - コネクション型通信プロトコル
  - UDP
    - User Datagram Protocol
    - コネクションレス型通信プロトコル

TCP

- TCP（Transmission Control Protocol）
  - コネクション型ストリーム転送
  - インターネット層のIP上で
    信頼性　と　**安全性**
    が確保された通信の実現
  - エンドツーエンド(End-to-End)のプロセス間通信

IP通信はコネクションレス型
→ 信頼性・安全性に劣る

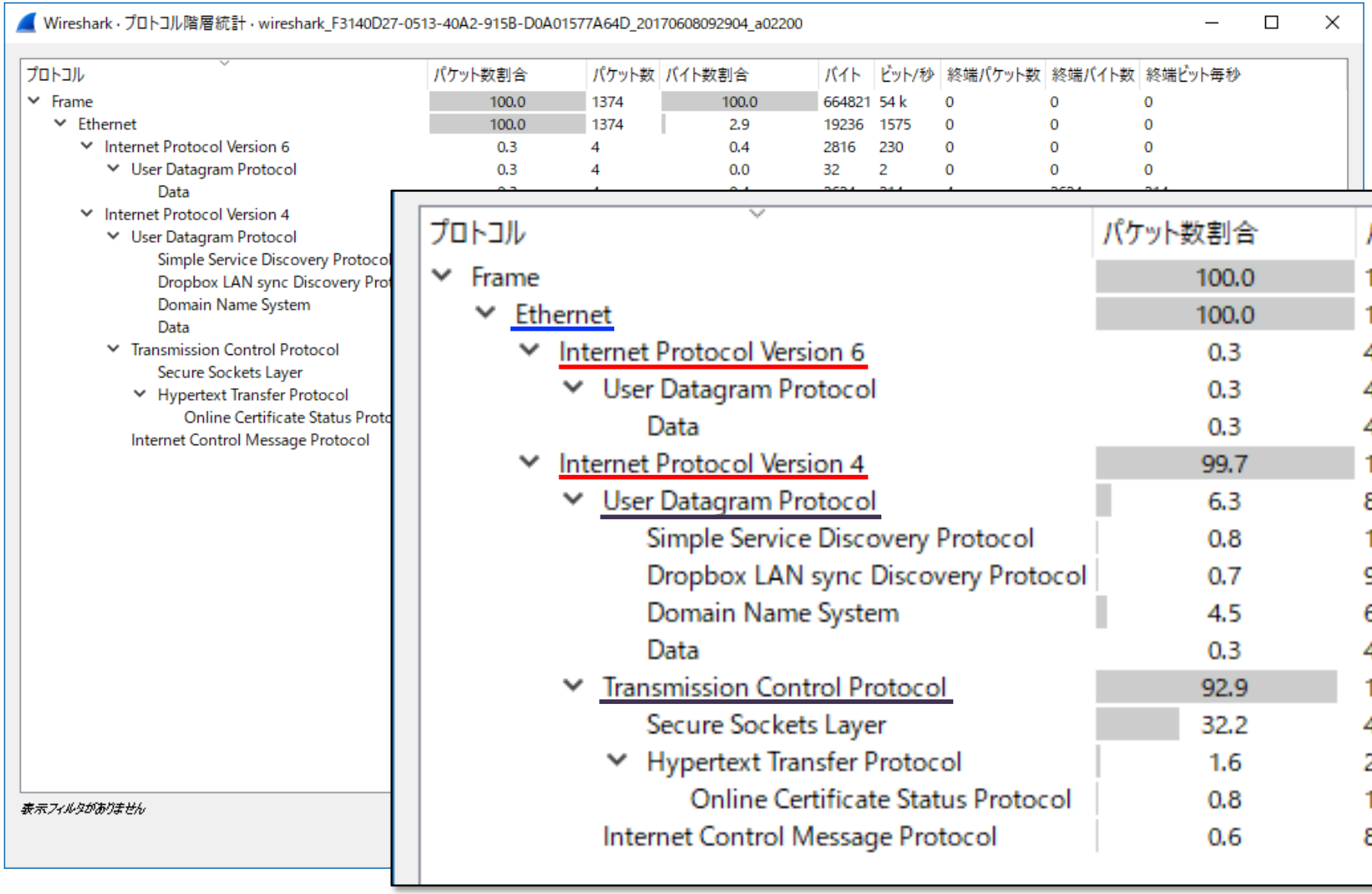
（注）　レイヤ２：　隣接ノード間のデータグラム転送サービス



UDP

- UDP（User Datagram Protocol）
  - コネクションレス型データグラム転送
    - 転送開始時に接続手続きを行なわない
  - 伝送の簡易制御方式
    - データ到着の順序制御などのシーケンス制御やエラー制御を行なわない
    - 上位層からのデータは分割せずに送信
  - 低負荷伝送
    - 構造が簡単、かつ、コネクションレス型転送なので、転送効率に優れる

Wireshark -プロトコル階層統計



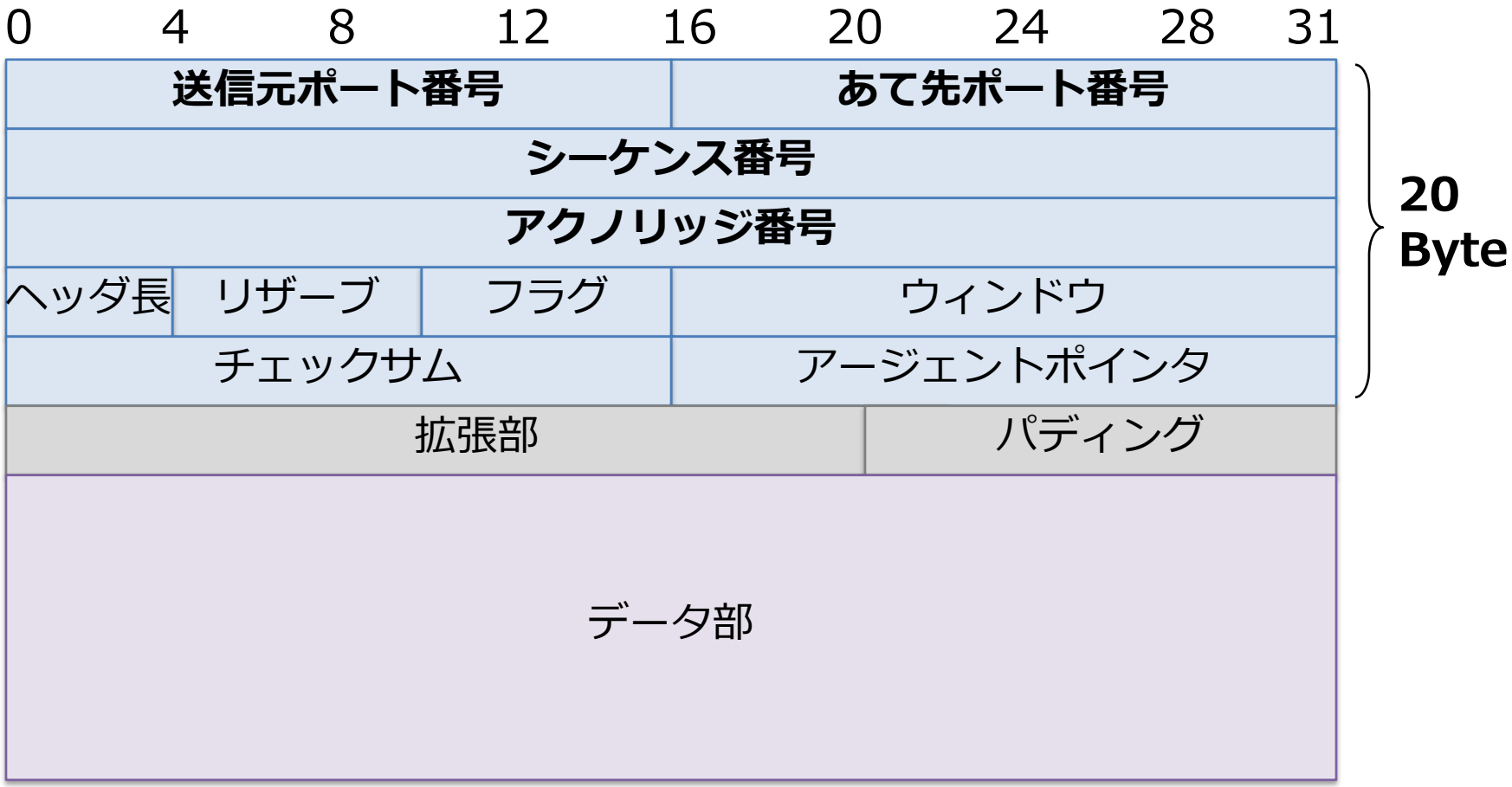
TCPの機能

- TCPの５つの機能
  1. **基本データ送信**（Basic Data Transfer）
     - セグメント単位でのデータ伝送
  2. **信頼性**（Reliability）
  3. **フロー制御**（Flow Control）
  4. **多重化**（Multiplexing）
  5. **コネクション**（Connection）

  – 高信頼性・高安全性通信の実現

TCPの機能： 1.セグメント

- TCPセグメント = ヘッダ部（最低20バイト）＋データ部



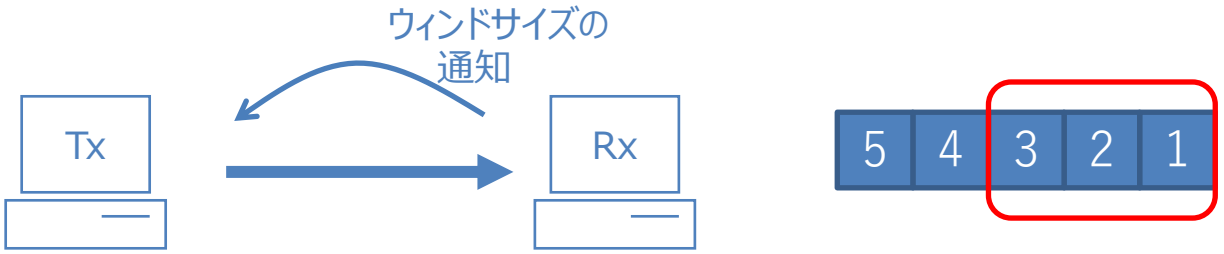
TCPの機能： 2.信頼性

- **信頼性**
  - データへの**通し番号(Sequential Number)**の添付
  - **肯定応答(Positive Acknowledge)**方式を利用したデータの送受信
  - **エラー回復**
    - データの紛失・重複の障害回復
    - データ送信後、一定時間内に応答がない場合は再送
  - **データ転送の信頼性の保証**
    - 転送シーケンス制御によりデータ抜けのチェックを行なう
    - 双方向データ転送（ストリーム型）をサポート

TCPの機能： 3.フロー制御

- **ウィンドウによるフロー制御**
  - **ウィンドウサイズ**を利用したフロー制御

  – **フロー制御**（データ流量の制御）
    - 送信データ量を調整・送受信側の同期
  – **スライディングウィンドウ利用のフロー制御**
    - ウィンドウのうち、Ack が返信された分のみデータを新たに送信



TCPの機能： 4.多重化

- **多重化**
  - ポートを利用した通信
  - ソケット通信の導入

  – ソケット通信（利用者インターフェイス）
    - アプリケーション・プログラム・インターフェイス
    - ソケットを利用し、TCP/IPの通信をプログラム
    - ポート番号によってアプリケーションを識別

  – ポート番号
    – RFC1700 (ASSIGNED NUMBERS) で割り当て
    - Well Known Ports: 1 - 1023
      – UNIX Standard Service : 256 - 1023
    - Registered Ports: 1024 - 49151
    - Dynamic and/or Private Ports: 1025 - 65535
      – 利用者が任意に利用可能

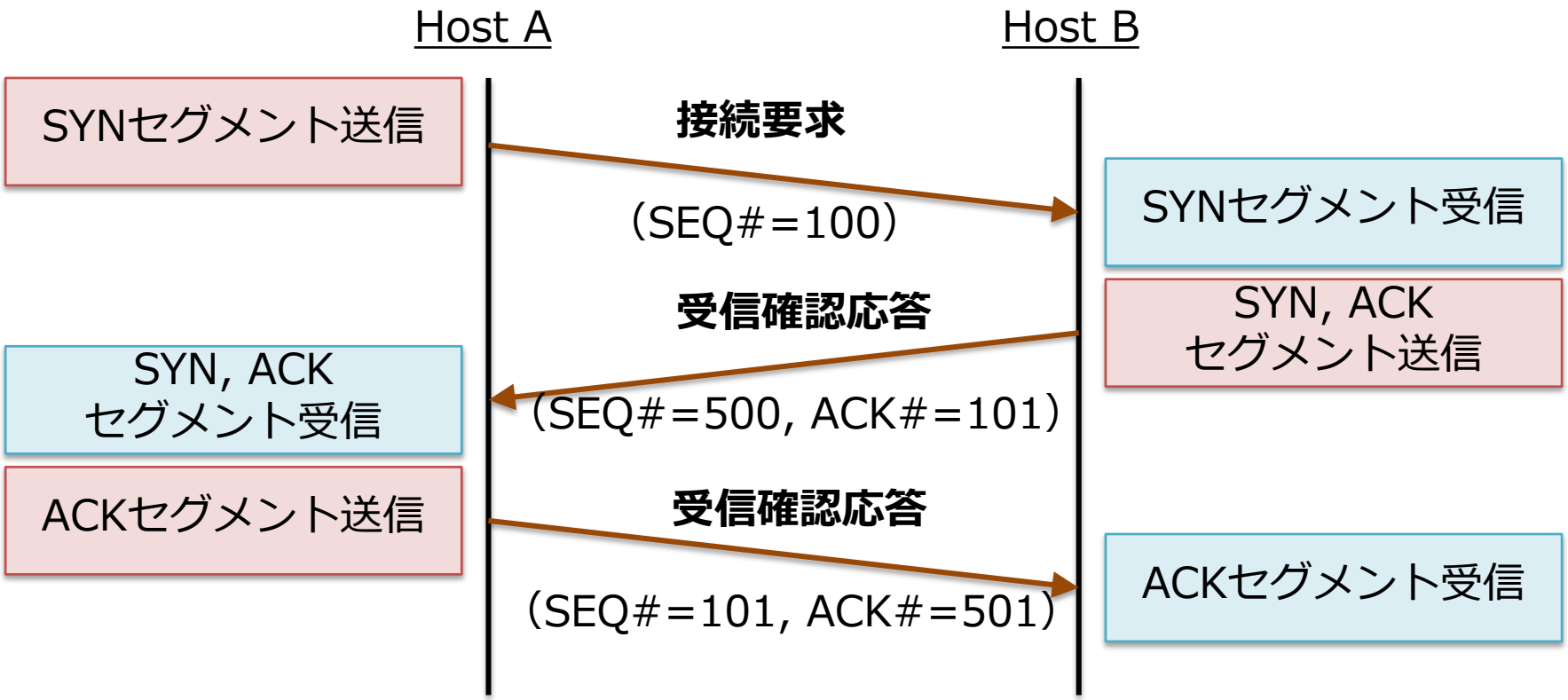
http://www.iana.org/
assignments/port-numbers

TCPの機能： 5.コネクション

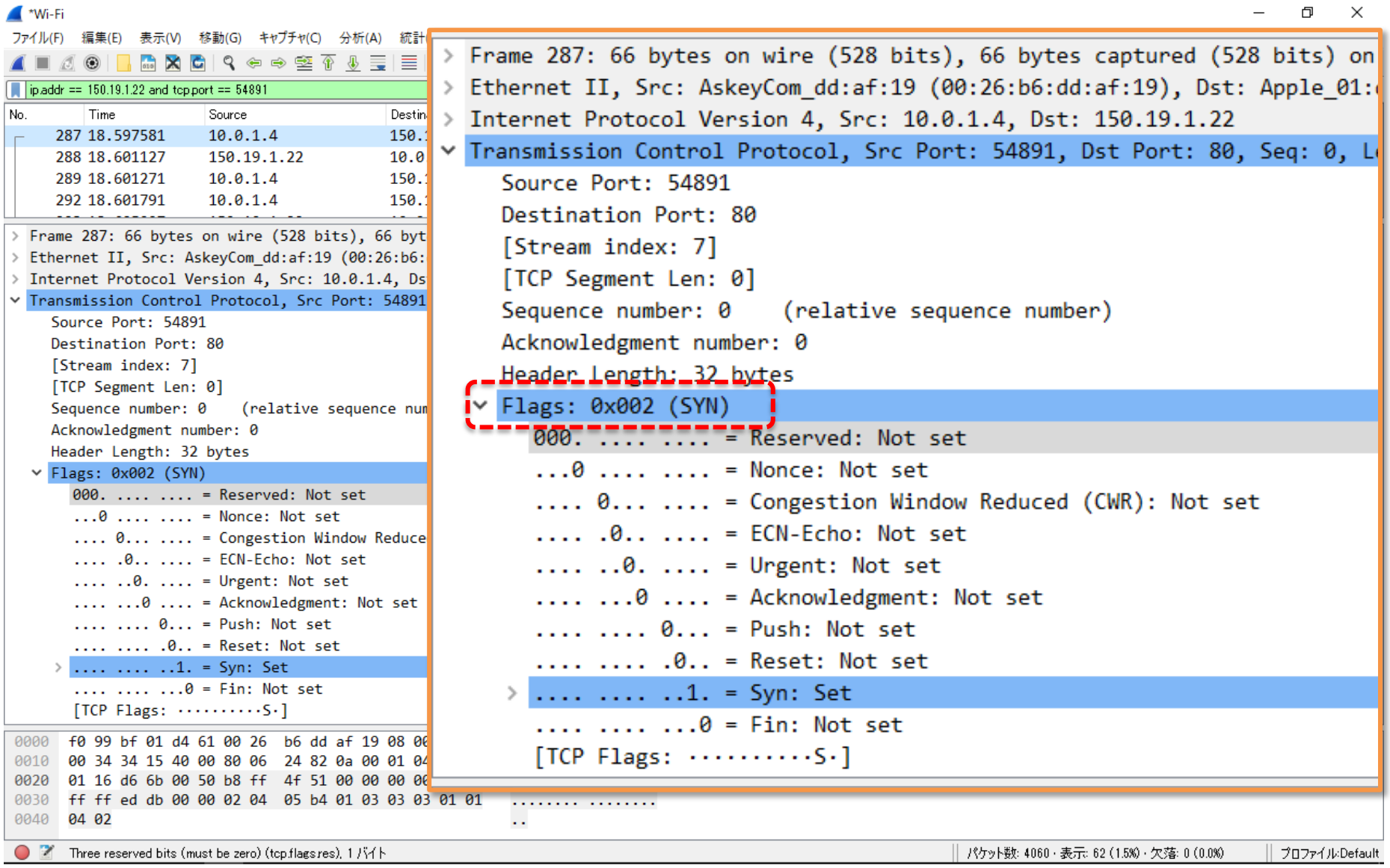
- **コネクション**
  - 通信をするために確立された仮想の経路
    - 仮想通信路
  - **ハンドシェイク（Handshake）方式**を利用したコネクションの確立と解除（切断）
    - **確立**：通信開始時に送信側と受信側との間に論理的な通信路（コネクション）を設定
    - **解除**：通信終了時に設定されていた通信路を解除（切断）

コネクションの確立

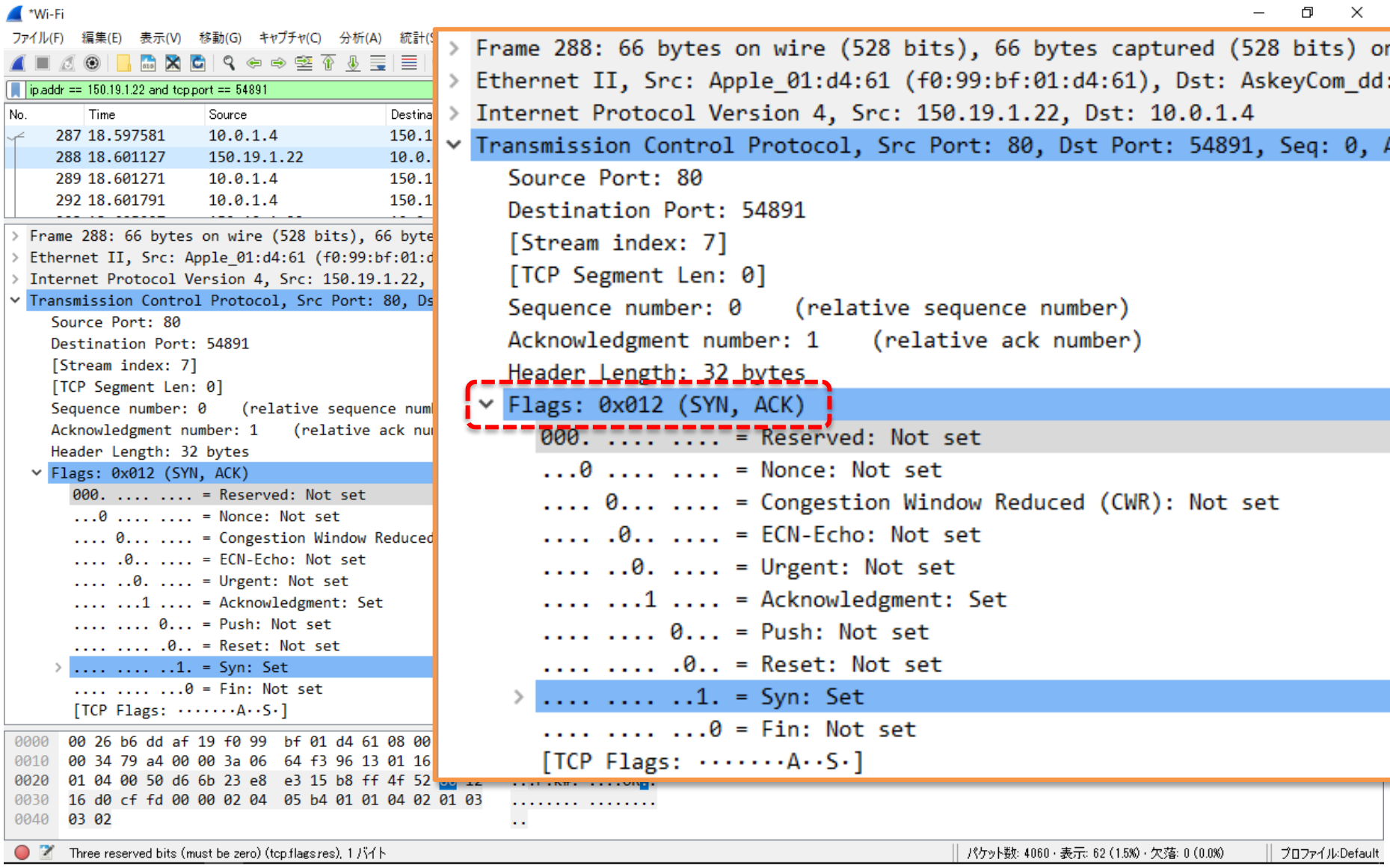
- **Three-way handshake**



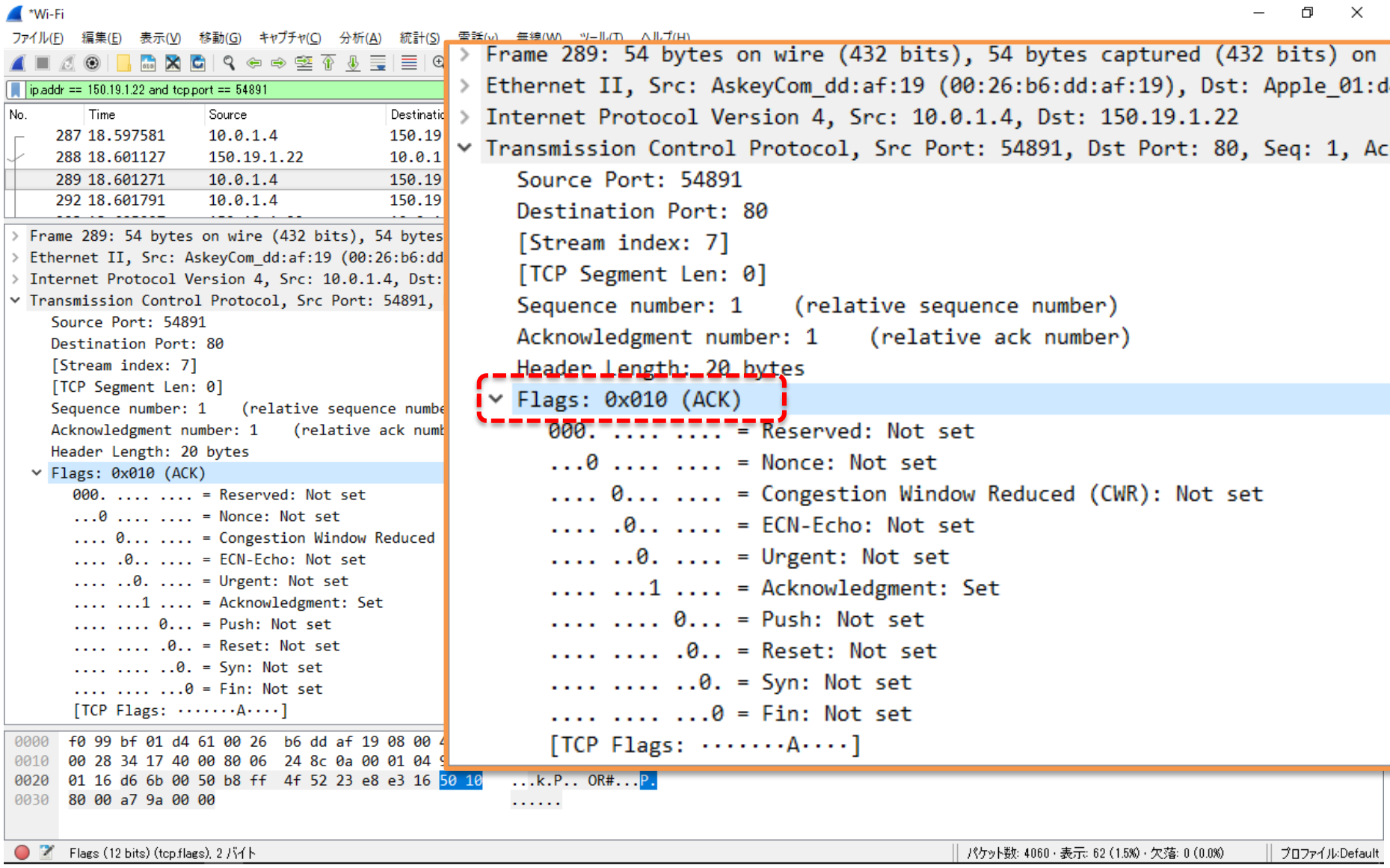
コネクションの確立：SYN



コネクションの確立：SYN+ACK



コネクションの確立：ACK

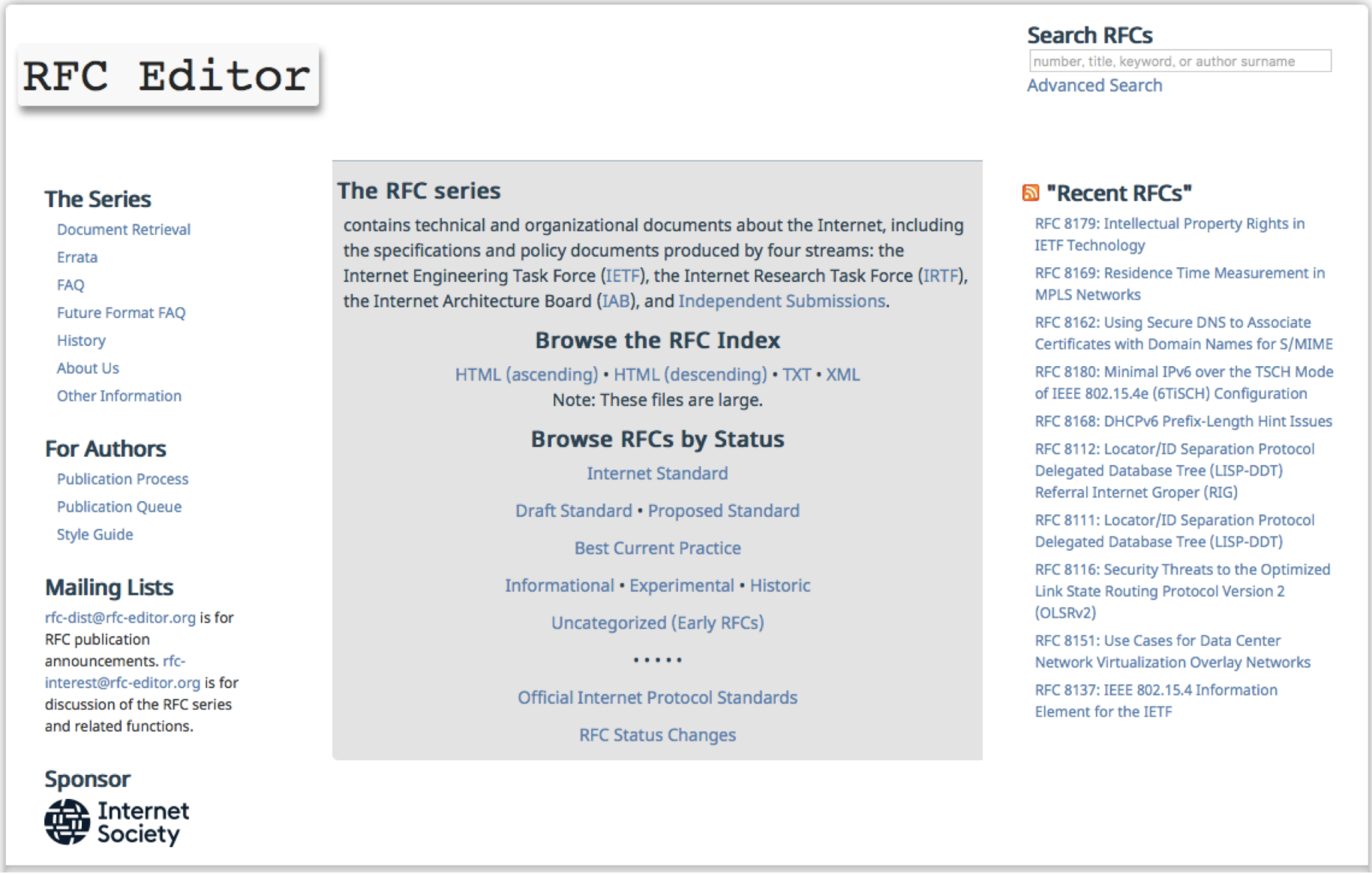


インターネットの標準

インターネットの標準

- RFC（Request For Comments）
  - 発行・維持・管理
    - IETF（Internet Engineering Task Force）
  - Standard Track：　標準文書であることを示す
    - Proposed Standard：　標準の提案段階
    - Draft Standard：　標準の最終仕様の段階
    - Standard：　標準として認められた段階
  - Best Current Practice：　手続きなどの文書
  - Informational：　参考文書
  - Experimental：　試験的な技術文書

RFC Editor http://www.rfc-editor.org/



RFC Editor http://www.rfc-editor.org/

RFC Editor

About this page

RFC Number (or Subseries Number):

Title/Keyword: Transmission Control Protocol

Show Abstract Show Keywords

Additional Criteria

Search Clear all

117 results (Show **25** | All)

Number	Files	Title	Authors	Date	More Info	Status
RFC 761	ASCII, PDF	DoD standard Transmission Control Protocol	J. Postel	January 1980	Obsoleted by RFC 793, RFC 7805	Historic (changed from Unknown April 2016)
RFC 793 a.k.a. STD 7	ASCII, PDF	**Transmission Control Protocol**	J. Postel	September 1981	**Errata**, Obsoletes RFC 761, Updated by RFC 1122, RFC 3168, RFC 6093, RFC 6528	Internet Standard
RFC 964	ASCII, PDF	**Some problems with the specification of the Military Standard Transmission Control Protocol**	D.P. Sidhu, T. Blumer	November 1985		Informational (changed from Unknown April 2016)
RFC 1379	ASCII, PDF	Extending TCP for Transactions -- Concepts	R. Braden	November 1992	Obsoleted by RFC 6247, Updated by RFC 1644	Historic (changed from Informational May 2011)
RFC 1613	ASCII, PDF	**cisco Systems X.25 over TCP (XOT)**	J. Forster, G. Satz, G. Glick, R. Day	May 1994		Informational
RFC 1644	ASCII, PDF	T/TCP -- TCP Extensions for Transactions Functional Specification	R. Braden	July 1994	Obsoleted by RFC 6247, Updates RFC 1379	Historic (changed from Experimental March 2011)
RFC 1693	ASCII, PDF	An Extension to TCP : Partial Order Service	T. Connolly, P. Amer, P. Conrad	November 1994	Obsoleted by RFC 6247	Historic (changed from Experimental March 2011)
RFC 1791	ASCII, PDF	**TCP And UDP Over IPX Networks With Fixed Path MTU**	T. Sung	April 1995		Experimental

RFC 793 – Transmission Control Protocol

RFC: 793
Replaces: RFC 761
IENs: 129, 124, 112, 81,
55, 44, 40, 27, 21, 5

TRANSMISSION CONTROL PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

1. INTRODUCTION

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.

This document describes the functions to be performed by the Transmission Control Protocol, the program that implements it, and its interface to programs or users that require its services.

代表的なSTD

RFC 791 (STD 5) - Internet Protocol (IP)
RFC 792 (STD 5) - Internet Control Message Protocol (ICMP)
RFC 768 (STD 6) - User Datagram Protocol (UDP)
RFC 793 (STD 7) - Transmission Control Protocol (TCP)
RFC 854 (STD 8) - Telnet Protocol Specification (TELNET)
RFC 859 (STD 9) - File Transfer Protocol (FTP)
RFC 821 (STD10) - Simple Mail Transfer Protocol (SMTP)
RFC1058 (STD34) - Routing Information Protocol (Historic) (RIP)
RFC 826 (STD37) - Ethernet Address resolution Protocol (ARP)
RFC1939 (STD53) - Post Office Protocol - Version3 (POP3)
RFC2328 (STD54) - OSPF Version 2 (OSPF)
RFC2453 (STD56) - RIC Version2 (RIP)

まとめ

- 情報システムを構築するために必要な技術である情報システム化技術のうち、
  - ネットワーク技術

  の知識を概観

- ネットワーク技術
  - 通信プロトコル（ OSI と TCP/IP ）
  - TCPの５つの機能
    - セグメント・信頼性・フロー制御・多重化・コネクション
      - 通し番号と肯定応答、そして、再送
      - ウィンドウによるフロー制御
      - コネクションの確立と切断
  - コンピュータのアドレス
    - MACアドレス・IPアドレス・ドメイン名
      - プライベートアドレス
      - CIDRによるアドレス割り当て

先端エレクトロニクスDAQセミナー2020

～ ソフトウェア技術 ～

データ収集システム化技術
（ネットワークプログラミング）

広島工業大学

長坂 康史

nagasaka@cc.it-hiroshima.ac.jp

目的と達成目標

- 目的
  - データ収集システムを構築するためのシステム化技術、特に、ネットワークプログラミングに関する技術を修得する
- 達成目標
  - データ収集システムを構築するために必要なネットワークプログラミングに関する技術を理解し、活用することができる

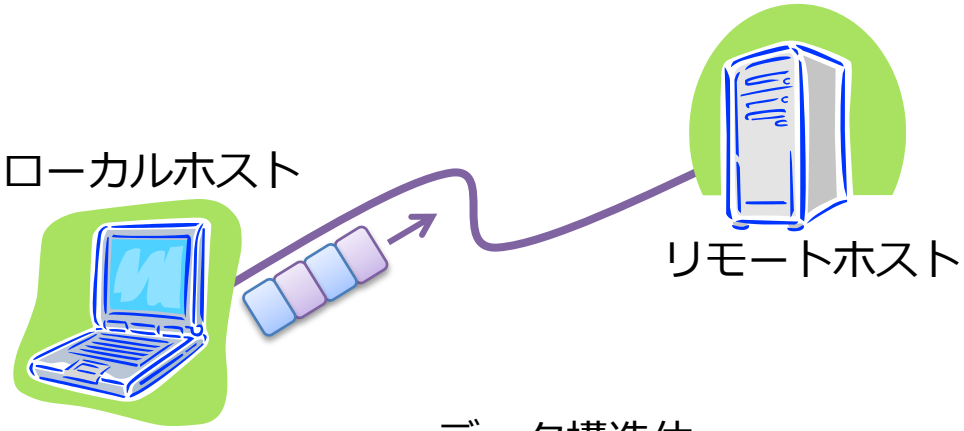
目次

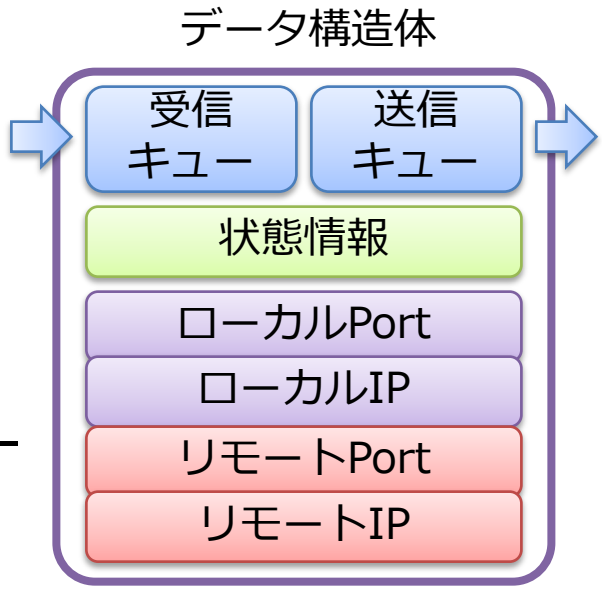
- データ収集システム化技術
  ネットワークプログラミング
  - ソケット通信
  - ソケットAPI
  - TCPクライアントとサーバ
- データ収集システムフレームワーク
  - DAQ-MW

ネットワーク通信
ソケット通信

ソケットに関するデータ構造体

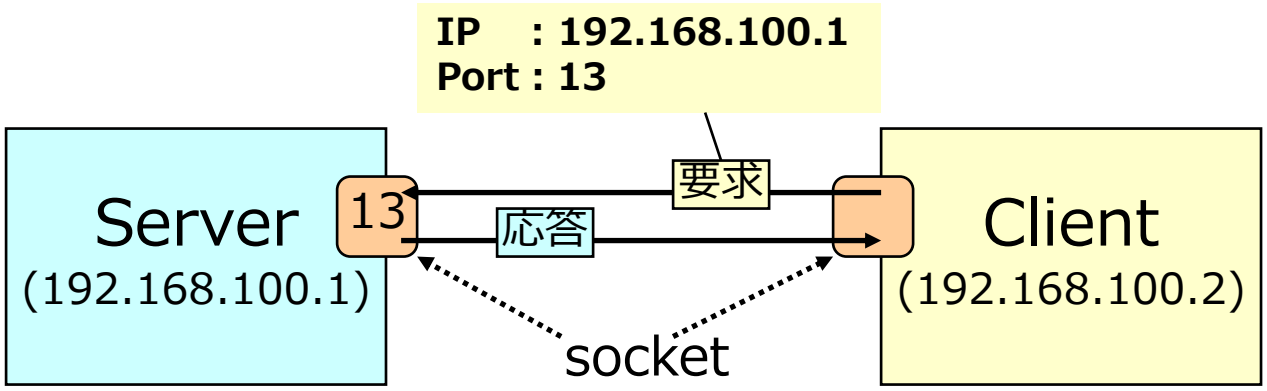
- ソケット構造体
  - ホストの情報
    - ローカルホスト
      - IPアドレス
      - ポート番号
    - リモートホスト
      - IPアドレス
      - ポート番号
  - キュー
    - 受信データのFIFOキュー
    - 送信を待つデータのFIFOキュー
  - プロトコル状態情報





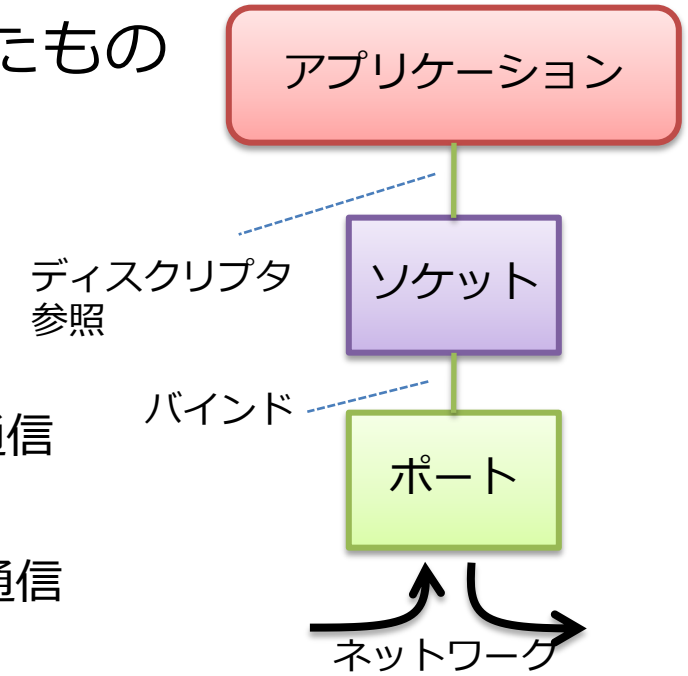
ソケット通信

- **Socket Communication**
  - クライアント（Client）-サーバ（Server）間での通信
  - 通信はソケットと呼ばれる通信ポートを利用して実現
    - ソケット：通信用の出入り口
  - アプリケーションごとにソケットを作成（割り当て）
    - ポート番号によってソケットを識別　（16 bit）



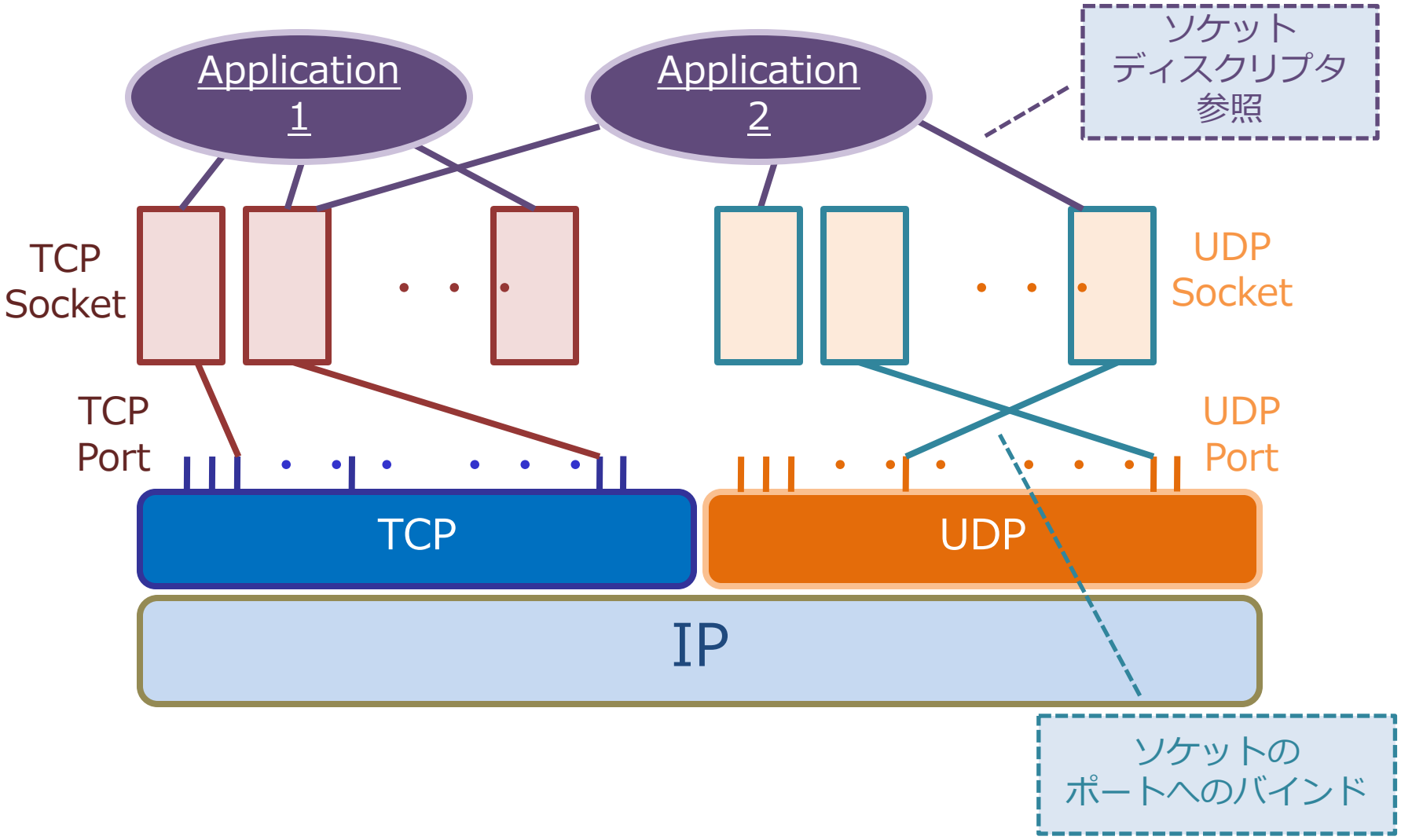
ソケットとポート番号

- ソケット
  - データ送受信の仕組みを抽象化したもの
  - エンドツーエンド通信の仕組み



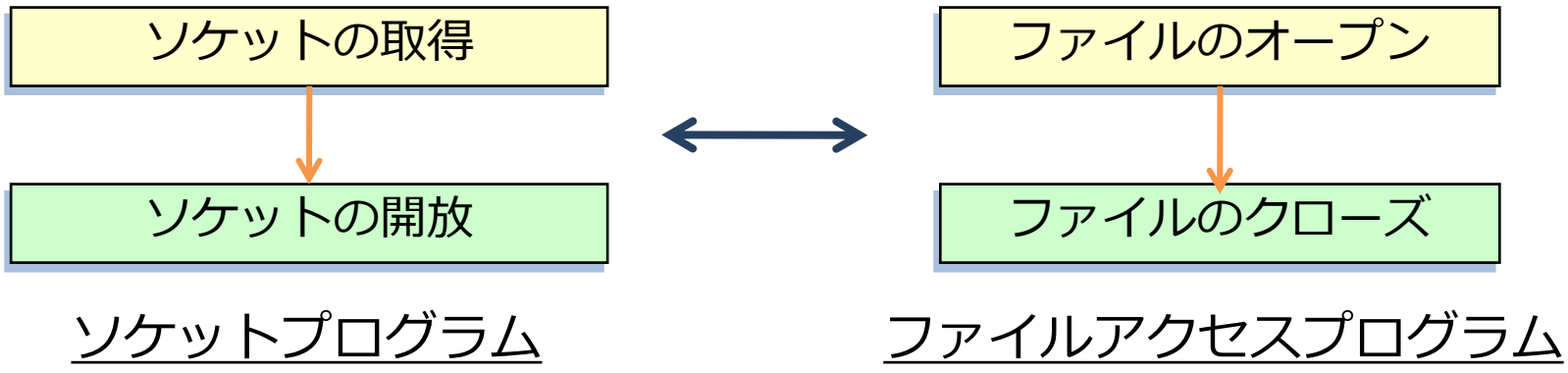
  - ストリームソケット
    - TCPの利用　→　信頼性の高いソケット通信
  - データグラムソケット
    - UDPの利用　→　信頼性の低いソケット通信
  - ポート番号
    - IANA (Internet Assigned Numbers Authority)がポート番号の割振り
      - » http://www.iana.org/assignments/port-numbers

ソケットディスクリプタとポート



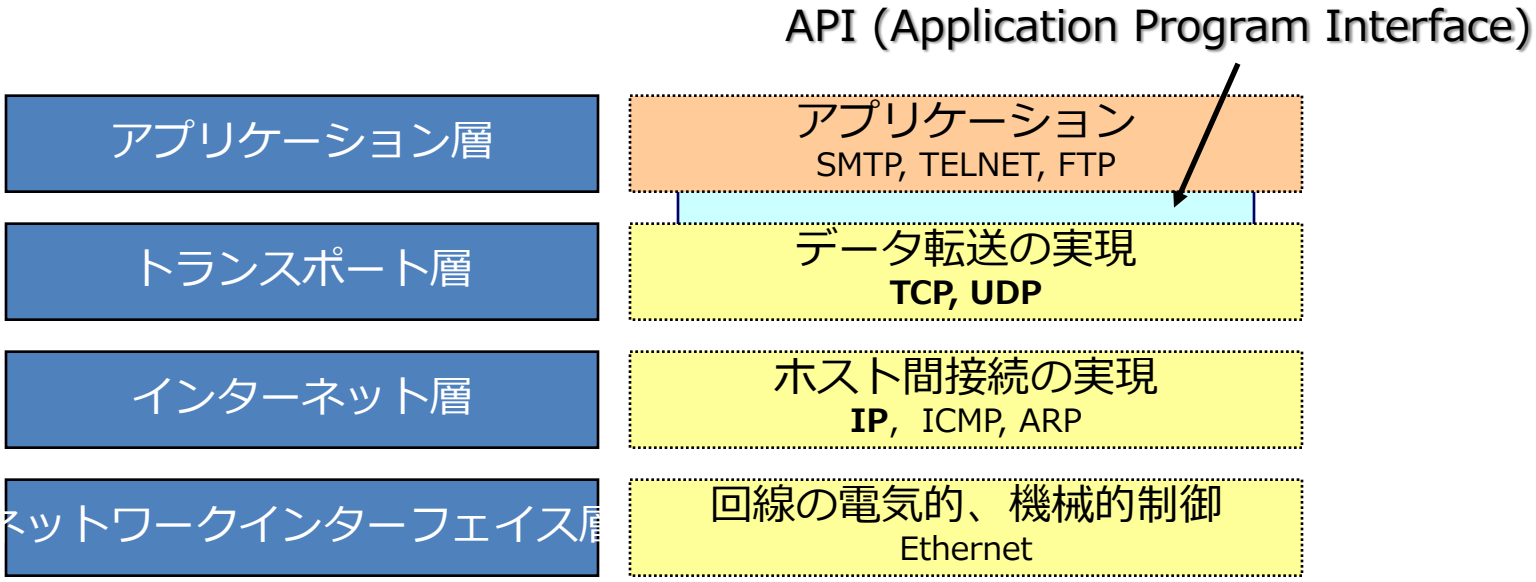
ソケットAPIとソケット通信

- ソケットAPI
  - サーバとクライアントやTCP/UDPで利用する種類が異なる
- ソケット通信
  - ファイル入出力に似ている部分あり
  - ソケット通信プロトコルの流れ（共通）



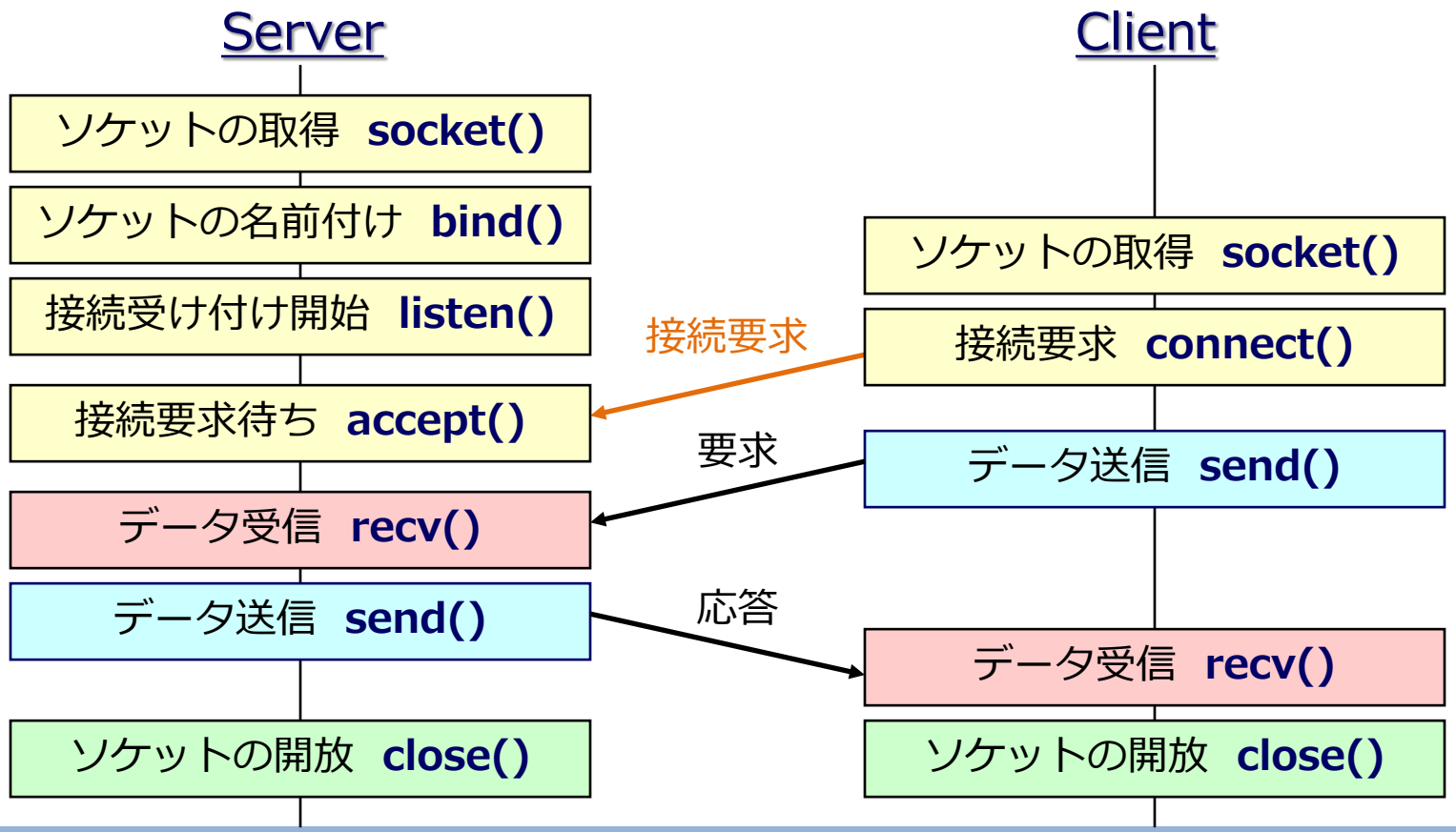
ソケットAPI（ネットワークAPI）

- ネットワークを利用してデータを送受信するためのAPI
- UNIX系OSのソケットシステムコール
  - socket, bind, listen, connect, accept, read, write, recv, send, recvfrom, sendto, shutdown, close, …



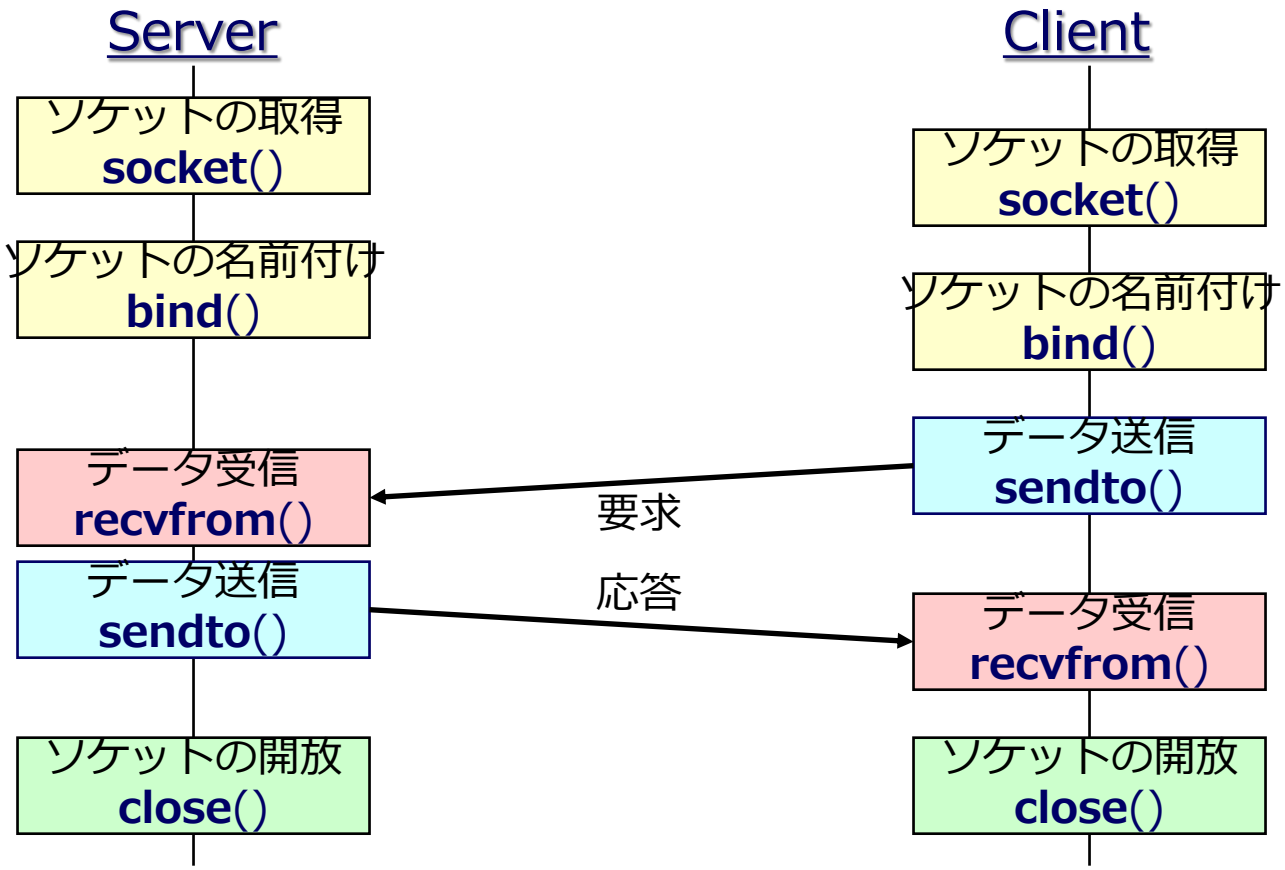
TCP

- Transmission Control Protocol
  - コネクション型通信の実現



UDP

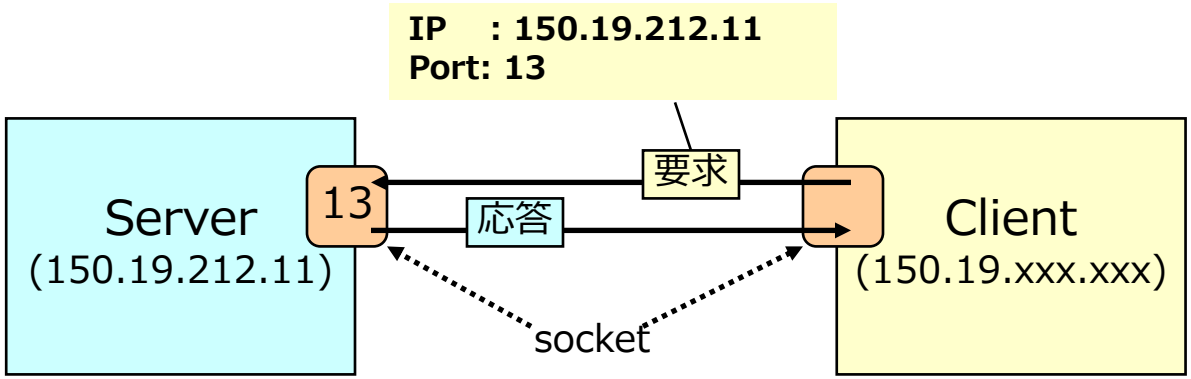
- User Datagram Protocol
  - コネクションレス型通信の実現



簡単なクライアントの例：時刻取得

- 時刻提供サーバに接続し、現在の時刻を得る
  - TCP/IPのTimeサービスポートを利用　（13）
    - daytime (13) ポートの使用

```
$ telnet 192.168.100.10 daytime
Trying 192.168.100.10...
Connected to 192.168.100.10.
Escape character is '^]'.
01 JUL 2018 15:40:07 JST
Connection closed by foreign host.
```



プロトコルファミリ
ソケット
アドレスの指定

ソケットの基礎

プロトコルファミリの種類

- プロトコルファミリ（プロトコルスイート）
  - プロトコルの集合体

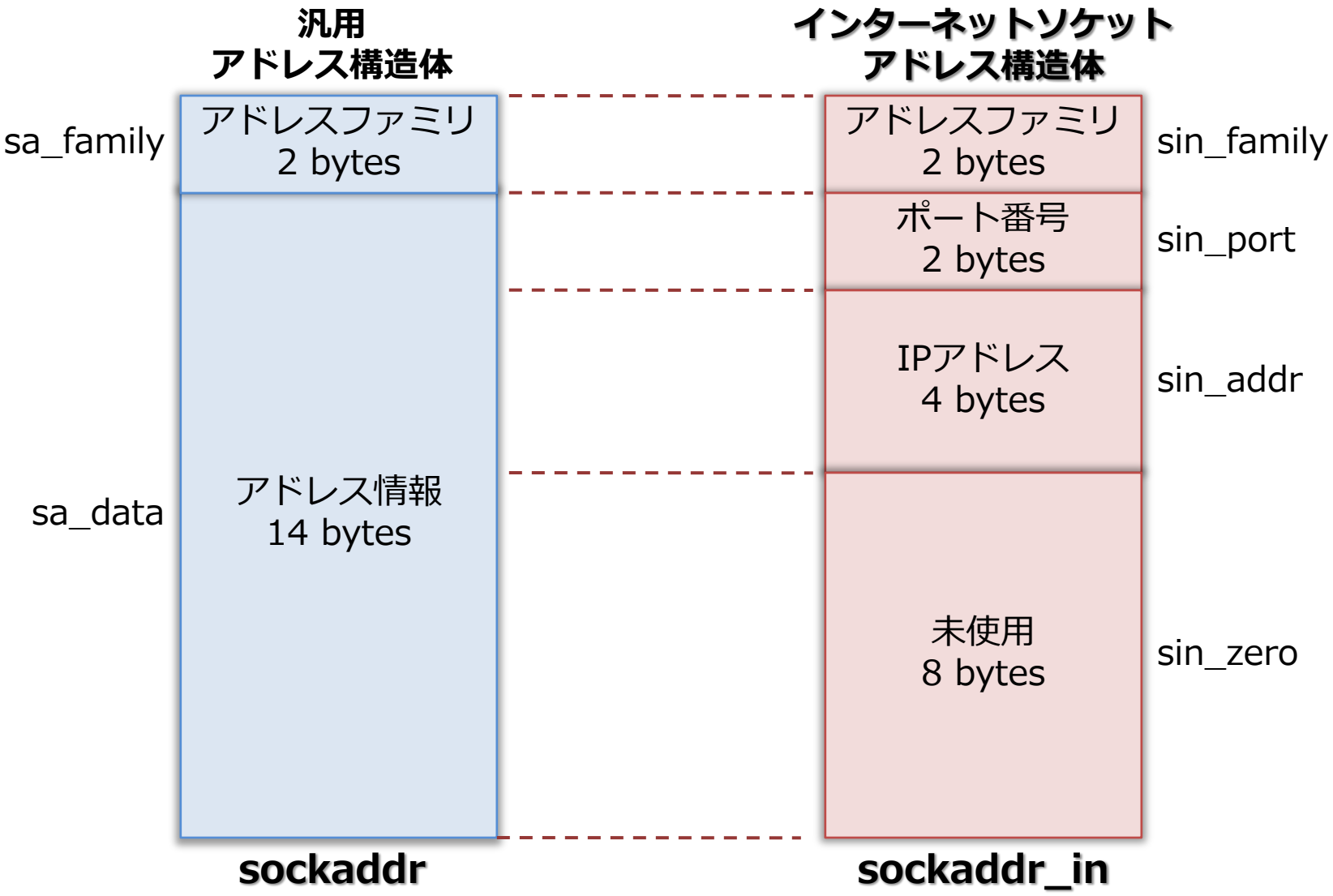
- Linux Socket API で利用できるプロトコルファミリ（PF）
  - **PF_INET** IPv4 インターネット・プロトコル
  - **PF_UNIX, PF_LOCAL** ローカル通信
  - PF_INET6 IPv6 インターネット・プロトコル
  - PF_IPX IPX（Novell）プロトコル
  - PF_NETLINK カーネル・ユーザ・デバイス
  - PF_X25 ITU-T X.25 / ISO-8208 プロトコル
  - PF_AX25 アマチュア無線 AX.25 プロトコル
  - PF_ATMPVC ATM PVC にアクセス
  - PF_APPLETALK アップルトーク
  - PF_PACKET 下層のパケットインターフェース

ソケットの種類

- **SOCK_STREAM**
  - 順序性と信頼性の保証されたコネクション型の通信ソケット
    - バイト・ストリームの提供
      - TCP（Transmission Control Protocol）
- **SOCK_DGRAM**
  - 順序性と信頼性の保証されないコネクションレス型の通信ソケット
    - データグラムの提供
      - UDP（User Datagram Protocol）

- SOCK_SEQPACKET
  - 順序性と信頼性のあるコネクション型の通信ソケット
- SOCK_RAW
  - 生のネットワーク・プロトコルに直接アクセスする通信ソケット
- SOCK_RDM
  - 信頼性は保証するが、順序性は保証しないデータグラム通信ソケット

アドレスの指定（構造体）


汎用
アドレス構造体
インターネットソケット
アドレス構造体
sa_family
アドレスファミリ
2 bytes
アドレスファミリ
2 bytes
sin_family
ポート番号
2 bytes
sin_port
IPアドレス
4 bytes
sin_addr
sa_data
アドレス情報
14 bytes
未使用
8 bytes
sin_zero
sockaddr
sockaddr_in

アドレスの指定 – sockaddr

- sockaddr 構造体 -汎用のアドレス構造体

```
struct  sockaddr {
    u_short   sa_family;     /* アドレスファミリ(AF_INET)*/
    char      sa_data[14];          /* アドレス情報  */
};
```

  - アドレス情報はOSやネットワークの種類の違いを吸収するために用意
  - ソケットアドレス
    - IP address – 32 bits
    - Port number – 16 bits

- ソケットアドレス用に特化したもの
  - sockaddr_in 構造体

アドレスの指定 – sockaddr_in

- sockaddr_in - インターネットソケットのアドレス構造体

```
struct sockaddr_in {
    short           sin_family;   /* アドレスファミリ(AF_INET)*/
    u_short         sin_port;     /* ポート番号(16 bits)   */
    struct in_addr  sin_addr;     /* IPアドレス(32 bits)   */
    char            sin_zero[8];  /* 未使用                */
};
```

- in_addr - アドレス構造体

```
struct in_addr {
    u_long          s_addr;       /* アドレス              */
};
```

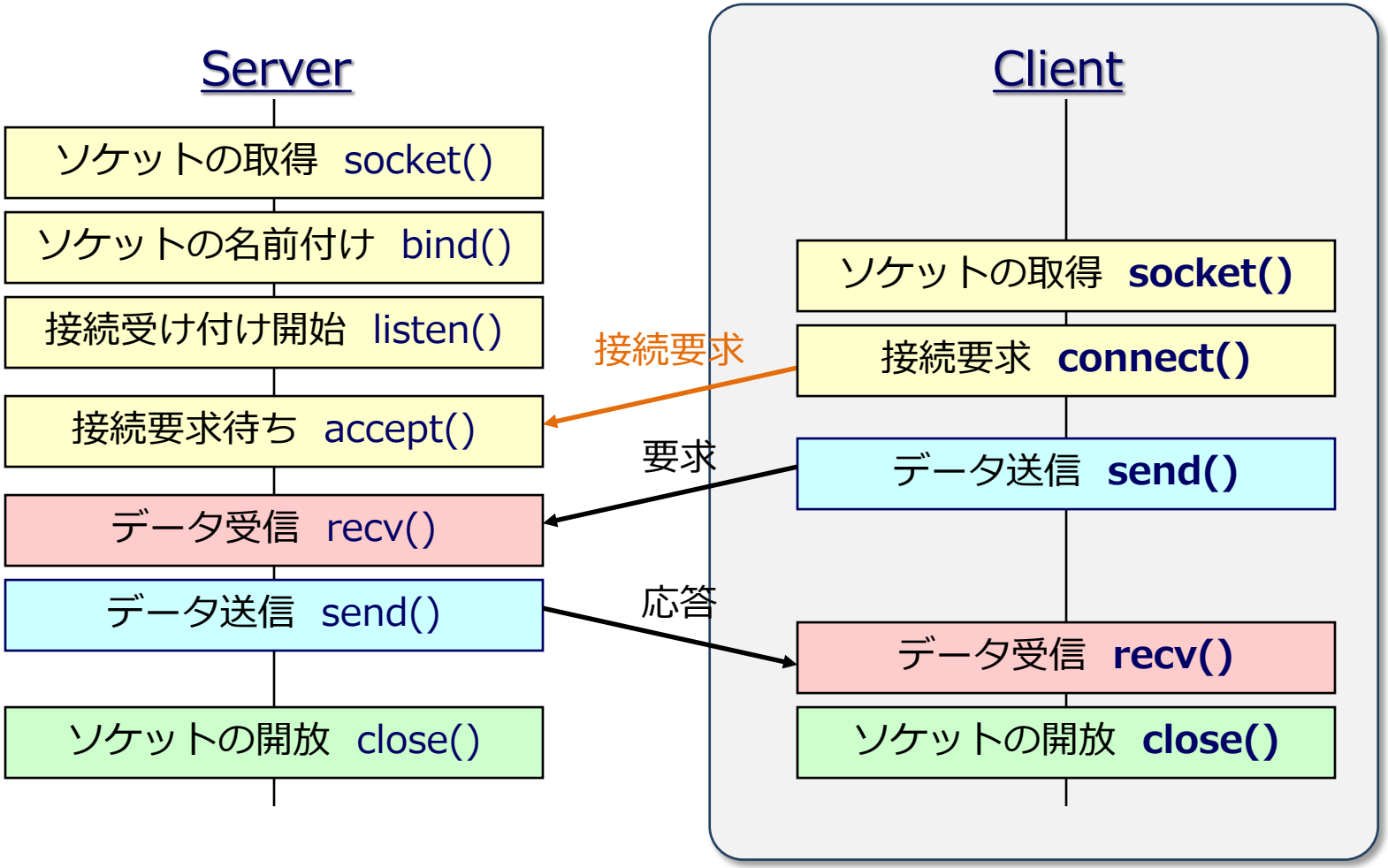
TCPクライアント

エコープログラム（TCP）

- TCPを利用したエコープログラム
  - クライアントから送った文字列が、そのままサーバから送り返されるプログラム

  - プログラム構成
    - TCPEchoClient.c：メインプログラム
    - DieWithError.c：エラー処理プログラム

TCPEchoClient概要



TCPEchoClient.c（1）

```
/*
 * filename: TCPEchoClient.c
 * date    : 2018.07.01
 * author  : Yasushi Nagasaka
*
 */

#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define RCVBUFSIZE 32

void DieWithError(char *errorMessage);
```

TCPEchoClient.c（2）

```
int main(int argc, char **argv) {
  int sock;
  struct sockaddr_in echoServAddr;
  unsigned short echoServPort;
  char *servIP;
  char *echoString;
  char echoBuffer[RCVBUFSIZE];
  unsigned int echoStringLen;
  int bytesRcvd, totalBytesRcvd;
```

TCPEchoClient.c（3）

```
if ( (argc<3) || (argc>4) ) {
  fprintf(stderr, "Usage: %s <Server IP> <Echo Word> [<Echo Port>]¥n",
          argv[0]);
  exit(1);
}

servIP = argv[1];
echoString = argv[2];

if ( argc == 4 ) {
  echoServPort = atoi(argv[3] );
} else {
  echoServPort = 7;
}
```

TCPEchoClient.c（4）

```
 if ( (sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0 ) {
  DieWithError("socket() failed");
}
```

```
memset(&echoServAddr, 0, sizeof(echoServAddr));
echoServAddr.sin_family = AF_INET;
echoServAddr.sin_addr.s_addr = inet_addr(servIP);
echoServAddr.sin_port = htons(echoServPort);
```

```
if ( connect(sock, (struct sockaddr *)&echoServAddr,
                                sizeof(echoServAddr)) < 0 ) {
  DieWithError("connect() failed");
}
```

```
echoStringLen = strlen(echoString);
```

TCPEchoClient.c（5）

```
if ( send(sock, echoString, echoStringLen, 0) != echoStringLen ) {
  DieWithError("send() sent a different number of bytes than expected");
}

totalBytesRcvd = 0;
printf("Received: ");

while ( totalBytesRcvd < echoStringLen ) {
  if ( (bytesRcvd = recv(sock, echoBuffer, RCVBUFSIZE-1, 0 )) <= 0 ) {
    DieWithError("recv() failed or connection closed prematurely");
  }

  totalBytesRcvd += bytesRcvd;
  echoBuffer[bytesRcvd] = '¥0';

  printf("%s", echoBuffer);
}
```

TCPEchoClient.c（6）

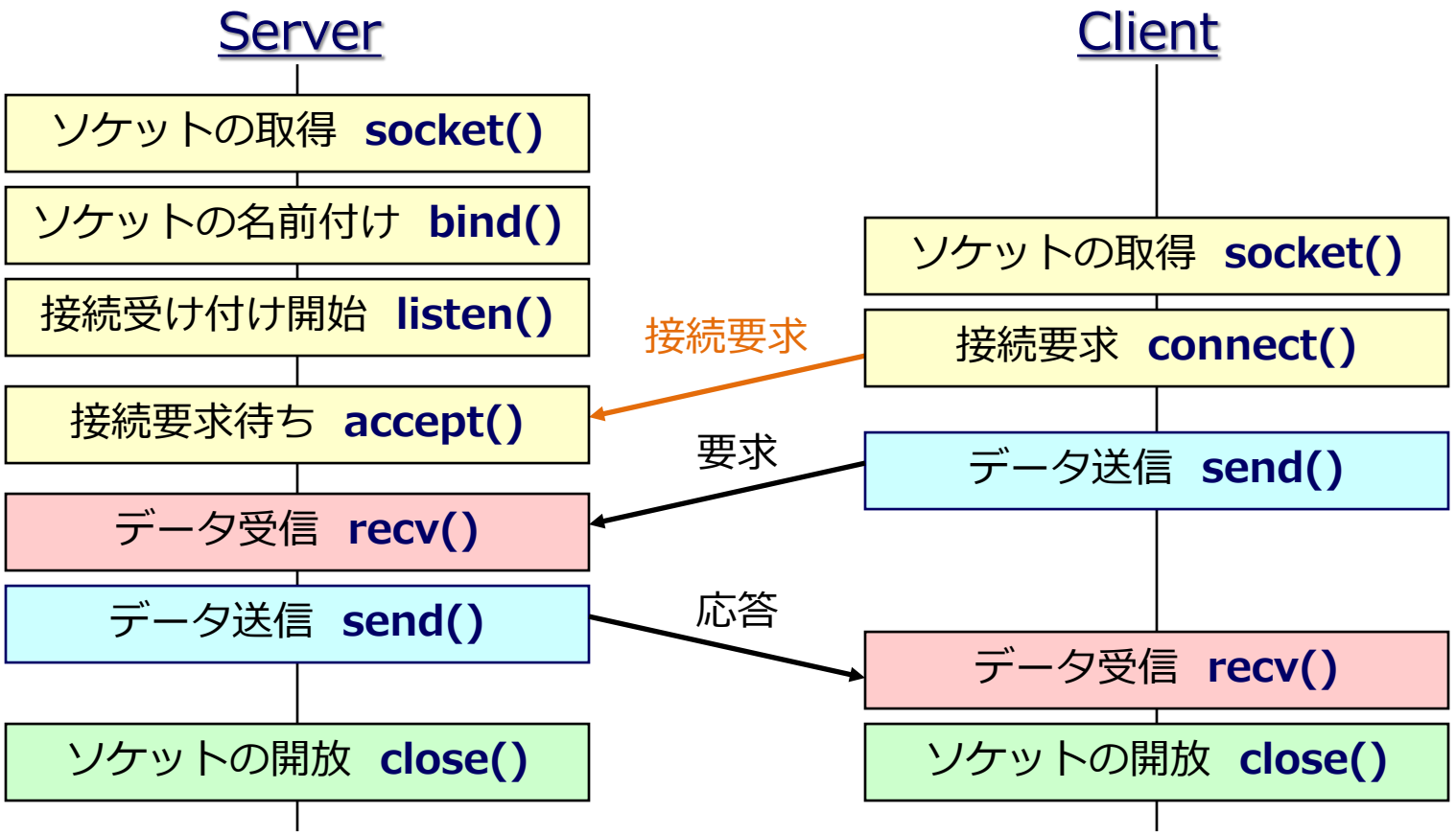
```
  printf("¥n");

   close(sock);
  exit(0);
}
```

DieWithError.c

```
/*
 * filename: DieWithError.c
 * date    : 2018.07.01
 * author  : Yasushi Nagasaka
*
 */

#include <stdio.h>
#include <stdlib.h>

void DieWithError(char *errorMessage) {
  perror(errorMessage);
  exit(1);
}
```

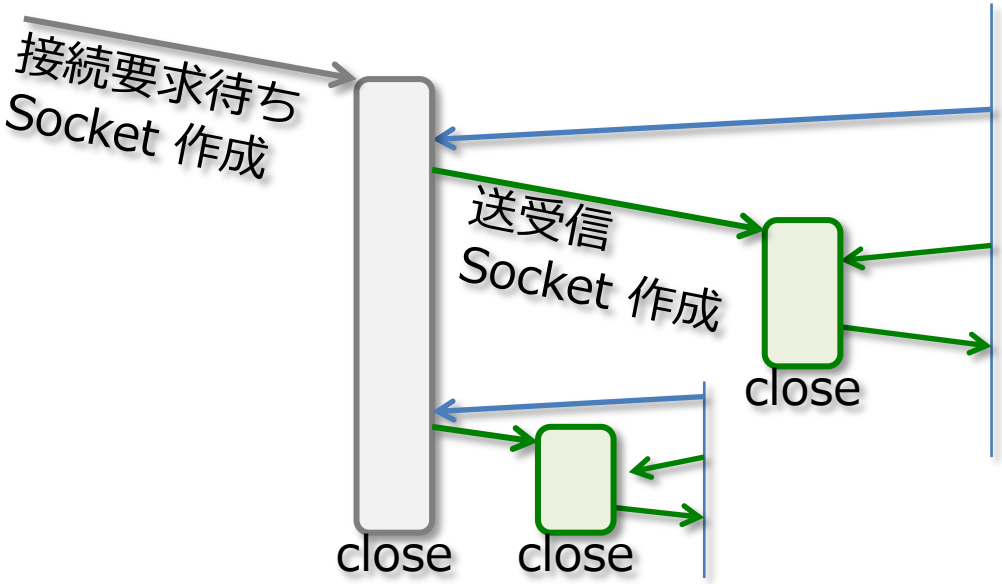
TCPサーバ

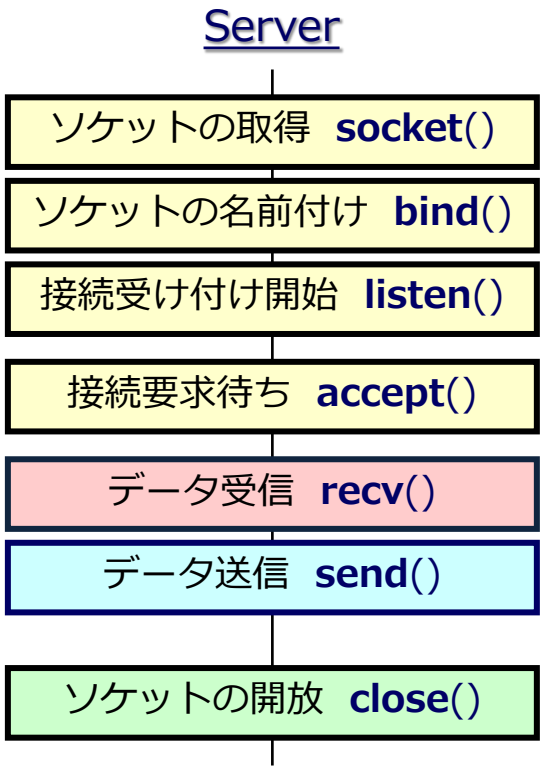
TCPEcho システム



サーバでの処理（１）

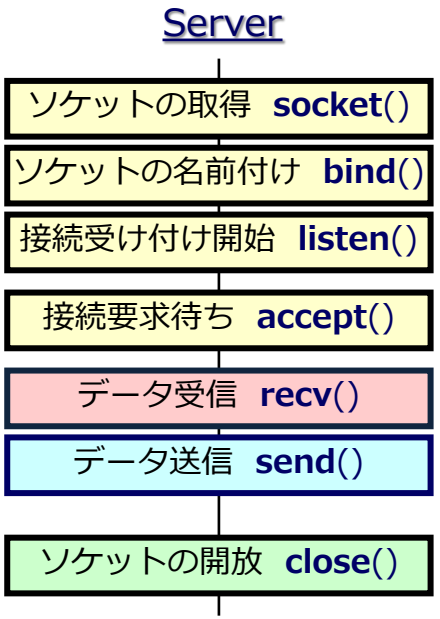
- ２つのサーバソケット
  - 接続要求待ち用ソケット
    - socket関数で作成
    - socket関数の戻り値として取得
  - データ送受信用ソケット
    - accept関数で作成
    - accept関数の戻り値として取得



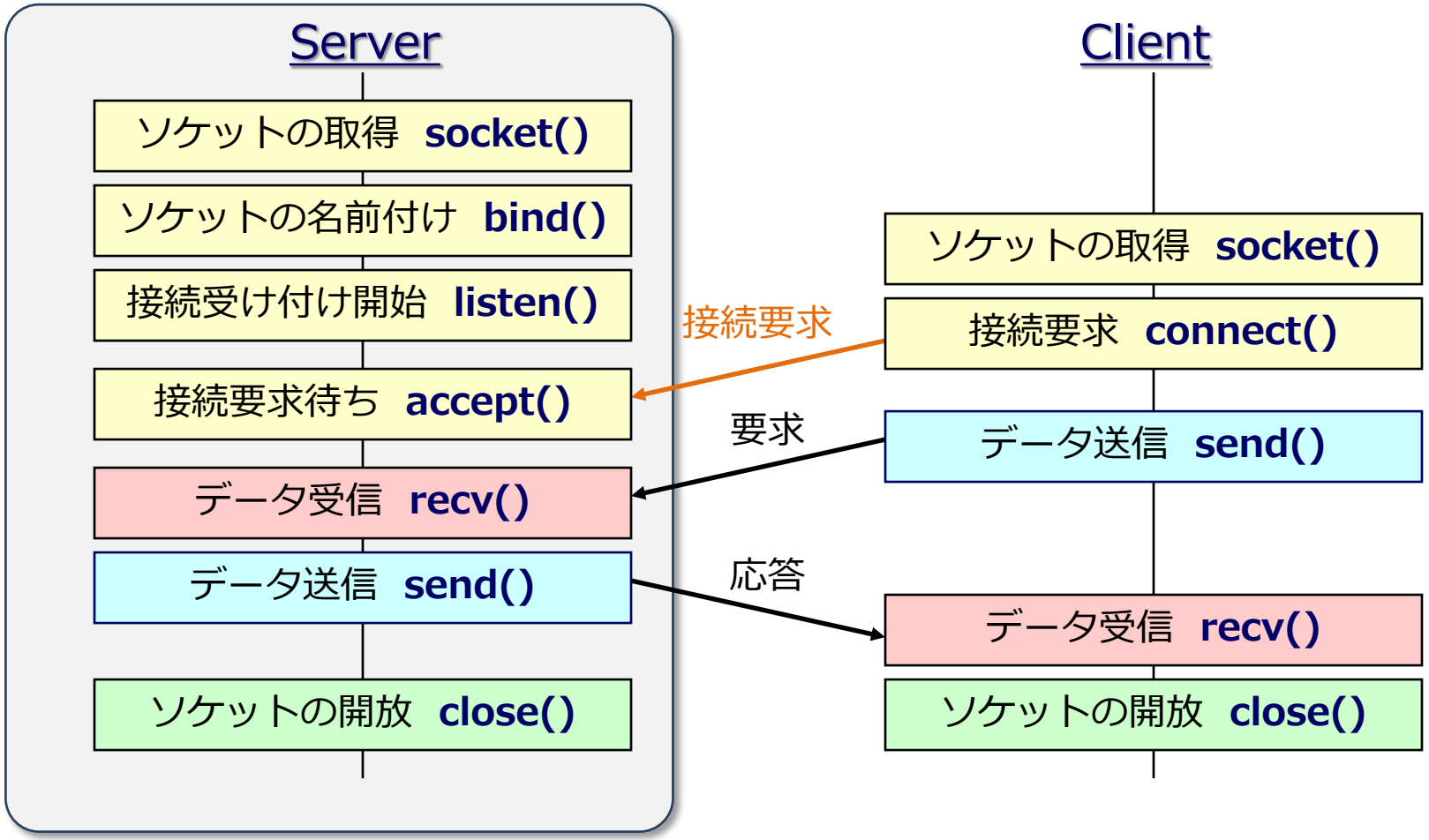


サーバでの処理（２）

- **socket( )**
  - ソケットの取得：TCPソケットの作成
- **bind( )**
  - ソケットの名前付け：ソケットにポート番号の割り当て
- **listen( )**
  - 接続受け付け開始：ポート番号へ接続できることをシステムへ通知
- **accept( )**
  - 接続要求待ち：要求された接続に対してソケットを作成
- **recv( ) / send ( )**
  - データ送受信：必要なデータの送受信
- **close( )**
  - ソケットの開放：２種類あることに注意



TCPEcho Server



TCPEchoServer（1）

```
/*
 * filename: TCPEchoServer.c
 * date    : 2018.07.01
 * author  : Yasushi Nagasaka
 *
 */

#include  <stdio.h>
#include  <sys/socket.h>
#include  <arpa/inet.h>
#include  <stdlib.h>
#include  <string.h>
#include  <unistd.h>

#define  MAXPENDING 5

void DieWithError(char *errorMessage);
void HandleTCPClient(int clntSocket);
```

TCPEchoServer（2）

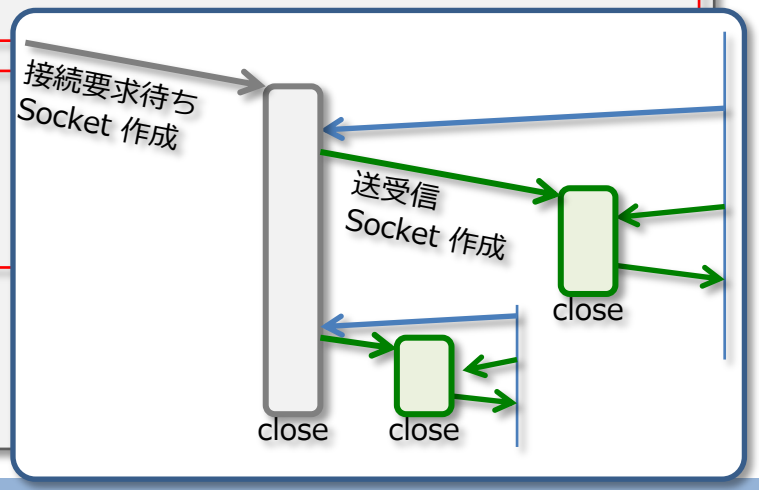
```
int main(int argc, char **argv)  {
  int servSock;
  int clntSock;
  struct sockaddr_in  echoServAddr;
  struct sockaddr_in  echoClntAddr;
  unsigned shorte  choServPort;
  unsigned  int clntLen;

  if (argc != 2) {
    fprintf(stderr,  "Usage: %s <Server Port>¥n", argv[0]);
    exit(1);
  }

  echoServPort  =  atoi(argv[1]);

  if ((servSock=socket(PF_INET,  SOCK_STREAM,  IPPROTO_TCP)) < 0) {
    DieWithError("socket() failed");
  }
```

TCPEchoServer（3）

```
memset(&echoServAddr,  0,  sizeof(echoServAddr));
echoServAddr.sin_family  =  AF_INET;
echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY);
echoServAddr.sin_port = htons(echoServPort);
```

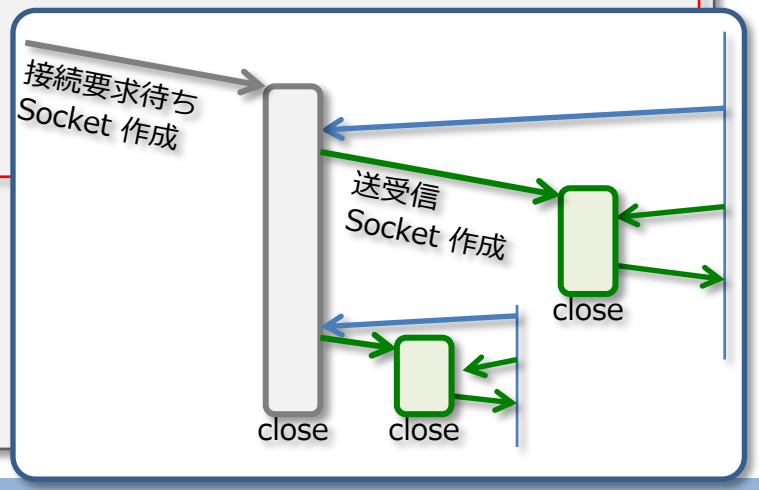
```
if( bind(servSock, (struct sockaddr *) &echoServAddr,
                                       sizeof(echoServAddr) ) <0) {
  DieWithError("bind() failed");
}
```

```
if ( listen(servSock, MAXPENDING) < 0 ) {
  DieWithError("listen() failed");
}
```



TCPEchoServer（4）

```
for ( ;; ) {
  clntLen = sizeof(echoClntAddr);

  if ((clntSock = accept(servSock, (struct sockaddr *)&echoClntAddr,
       &clntLen) ) < 0) {
    DieWithError("accept() failed");
  }

  printf("Handling client %s¥n", inet_ntoa(echoClntAddr.sin_addr));

  HandleTCPClient(clntSock);
}

}
```



HandleTCPClient（1）

```
/*
 * filename: HandleTCPClient.c
 * date    : 2017.05.11
 * author  : Yasushi Nagasaka  #b115501
 *
 */

#include <stdio.h>
#include <sys/socket.h>
#include <unistd.h>

#define RCVBUFSIZE 32

void DieWithError( char  *errorMessage);
```

HandleTCPClient（2）

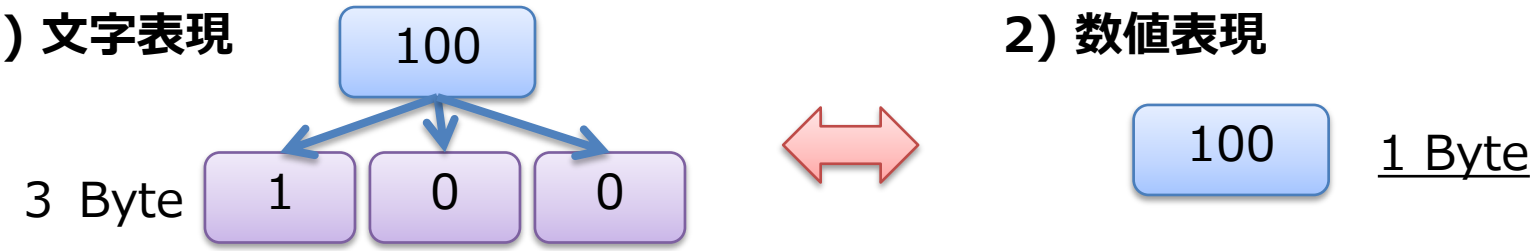
```
void HandleTCPClient(int clntSocket)
{
  char  echoBuffer[RCVBUFSIZE];
  int  recvMsgSize;

  if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0 ) {
    DieWithError("recv() failed");
  }
  while(recvMsgSize > 0) {
    if ( send(clntSocket, echoBuffer, recvMsgSize, 0 ) != recvMsgSize) {
      DieWithError("send()  failed");
    }
    if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0))  < 0){
      DieWithError("recv() failed");
    }
  }
  close(clntSocket);
}
```

メッセージの作成

データのエンコード

- エンコード（符号化）
  - 情報源などから発生した情報を一定の規則に則ってデジタルデータに変換すること
- デコード（復号化）
  - エンコードされたデータを元の情報に変換すること

- 数字のエンコード
  - 数字をそれぞれ、文字（ASCII文字）として表現
  - 数字をそれぞれ、バイナリデータとして表現
    - アクセスのしやすさと数値計算のしやすさ

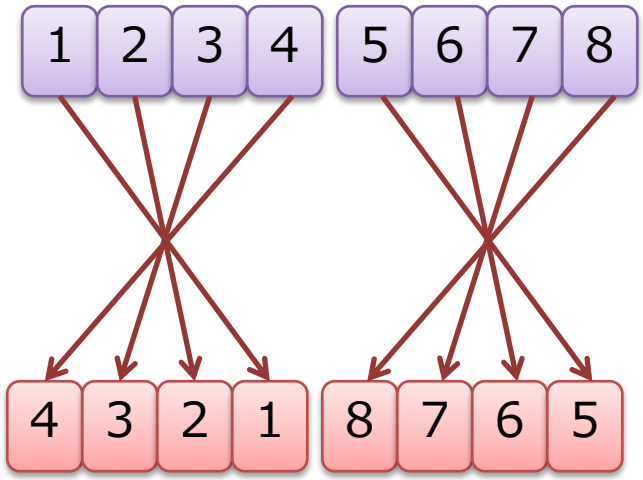


バイトオーダ（バイト順）

- バイトオーダ
  - CPUアーキテクチャによって違うバイトの表現順序のこと
    - アーキテクチャにより起こる情報の表現方法の違い

- **ビッグエンディアン**
  - 最上位バイトが最下位アドレスに
  - Motorola社68000ファミリ・SUN社Sparc
  - ネットワーク バイト オーダ

- **リトルエンディアン**
  - 最下位バイトが最下位アドレスに
  - Intel社x86系



8バイトのデータの場合

バイトオーダ変換関数

- バイトオーダを変換する関数

```
long int htonl(long int hostLong);
```

```
long int ntohl(long int netLong);
```

```
short int htons(short int hostShort);
```

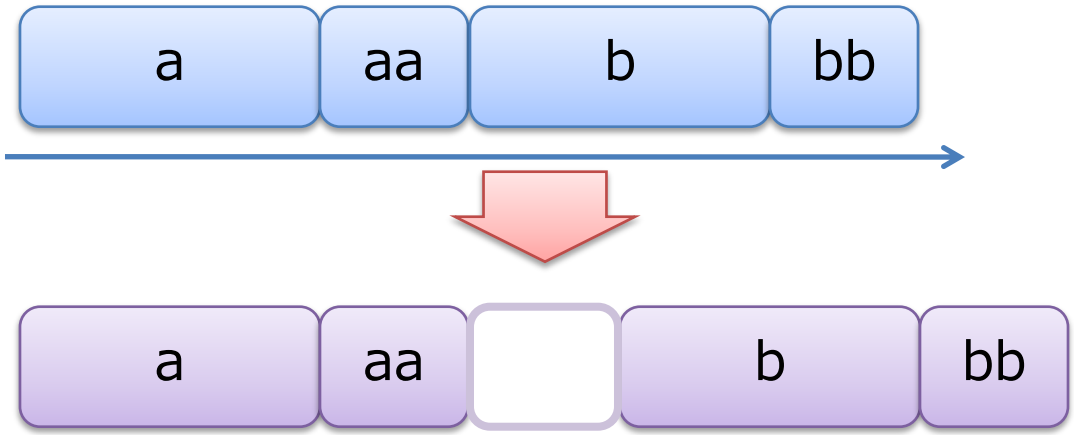
```
short int ntohs(short int netShort);
```

  - ホストバイトオーダとネットワークバイトオーダが同一の場合
    - これらの関数では、データの順番はそのままで、素通りする関数

整列とパディング

- 整列
  - それぞれのデータを扱いやすい境界（ここでは４バイト）に合わせてデータを配置
- パディング
  - 整列で空いてしまった場所に入れる空の領域

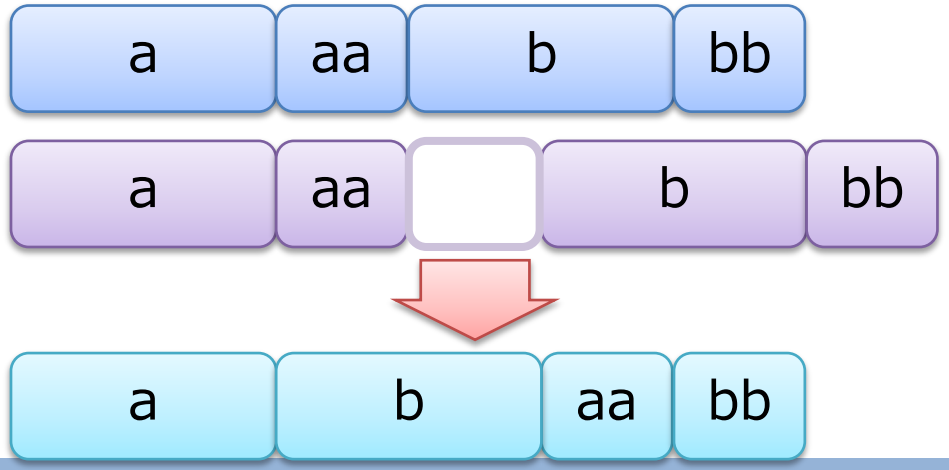
```
struct {
    int  a;
    u_short  aa;
    int  b;
    u_short  bb;
} msgBuf;
```



整列とパディング（2）

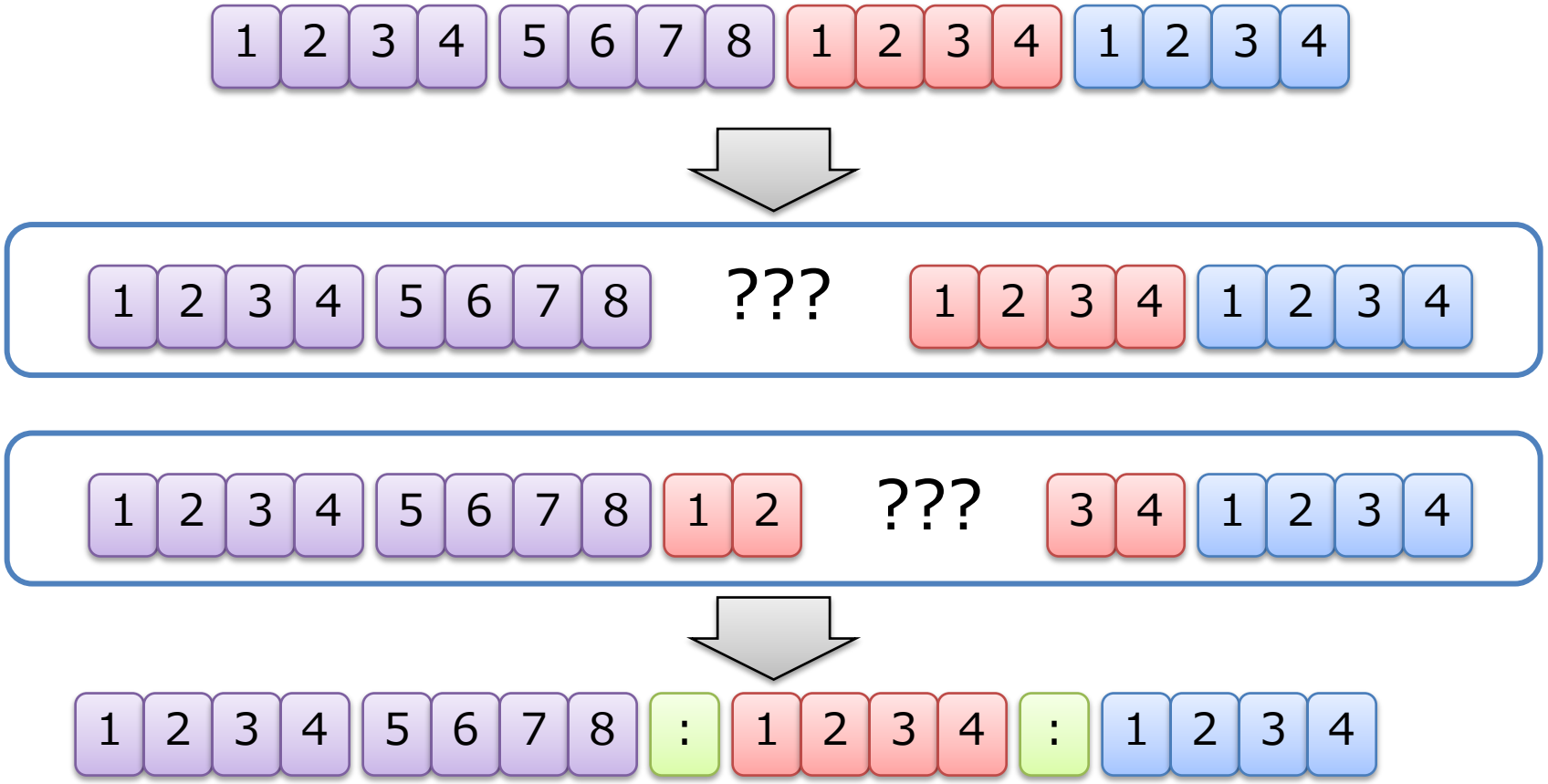
- 整列
  - 構造体はできるだけ大きな境界にあわせて整列
  - 構造体以外のフィールドはそれぞれのサイズに合わせて整列
- コンパイラによるパディングの挿入を避けるために
  - メッセージの定義にパディングを明示的に挿入
  - 各フィールドが正しく整列されるように構造体を編成

```
struct {
    int  a;
    int  b;
    u_short  aa;
    u_short  bb;
} msgBuf2;
```



フレーミングと解析

- フレーミング
  - メッセージの先頭や末尾、およびフィールド間の区切りを特定できるように、データのフォーマットを行うこと



ソケットオプション

- **ソケット通信**
  - ソケットを通してデータの送受信を実施
  - ソケットの実装およびパラメータ設定は汎用型

- **ソケットオプション関数**
  - ソケットの動作の特性を決めるパラメータへのアクセス
    - ソケットオプションの値を得る関数 - **getsockopt( )**
    - ソケットオプションの値を設定する関数 - **setsockopt( )**

ソケットオプション関数

- getsockopt( )：ソケットオプションの値を得る関数

```
int getsockopt(int socket, int level, int optName,
                                    void *optVal, u_int *optLen)
```

- setsockopt( )：ソケットオプションの値を設定する関数

```
int setsockopt(int socket, int level, int optName,
                                    const void *optVal, u_int optLen)
```

- ソケットオプションの指定
  - 関数の第二引数および第三引数で指定
    1. レベル（ int level ）　　- 関数の第二引数
    2. 種類（ int optName ）　　- 関数の第三引数

ソケットオプションの指定：レベル

○ **レベル（ int　level ）**

- **SOL_SOCKET**
  - ソケットレイヤ自体で処理されるオプション
    - プロトコルに依存しない
- **IPPROTO_TCP**
  - トランスポート層プロトコル（TCP）で処理されるオプション
- **IPPROTO_IP**
  - ネットワーク層プロトコル（IP）で処理されるオプション

TCP/IP 階層	
4	アプリケーション層
3	トランスポート層
2	インターネット層
1	ネットワーク インターフェイス層

ソケットオプションの指定（種類）

○ 種類（ int optName ）

- **SOL_SOCKET**
  - **SO_BROADCAST**
  - SO_KEEPALIVE
  - SO_LINGER
  - **SO_RCVBUF**
  - SO_RCVLOWAT
  - **SO_REUSEADDR**
  - SO_SNDLOWAT
  - **SO_SNDBUF**
- **IPPROTO_TCP**
  - TCP_MAX
  - **TCP_NODELAY**
- **IPPROTO_IP**
  - IP_TTL
  - IP_MULTICAST_TTL
  - IP_MULTICAST_LOOP
  - IP_ADD_MEMBERSHIP
  - IP_DROP_MEMBERSHIP

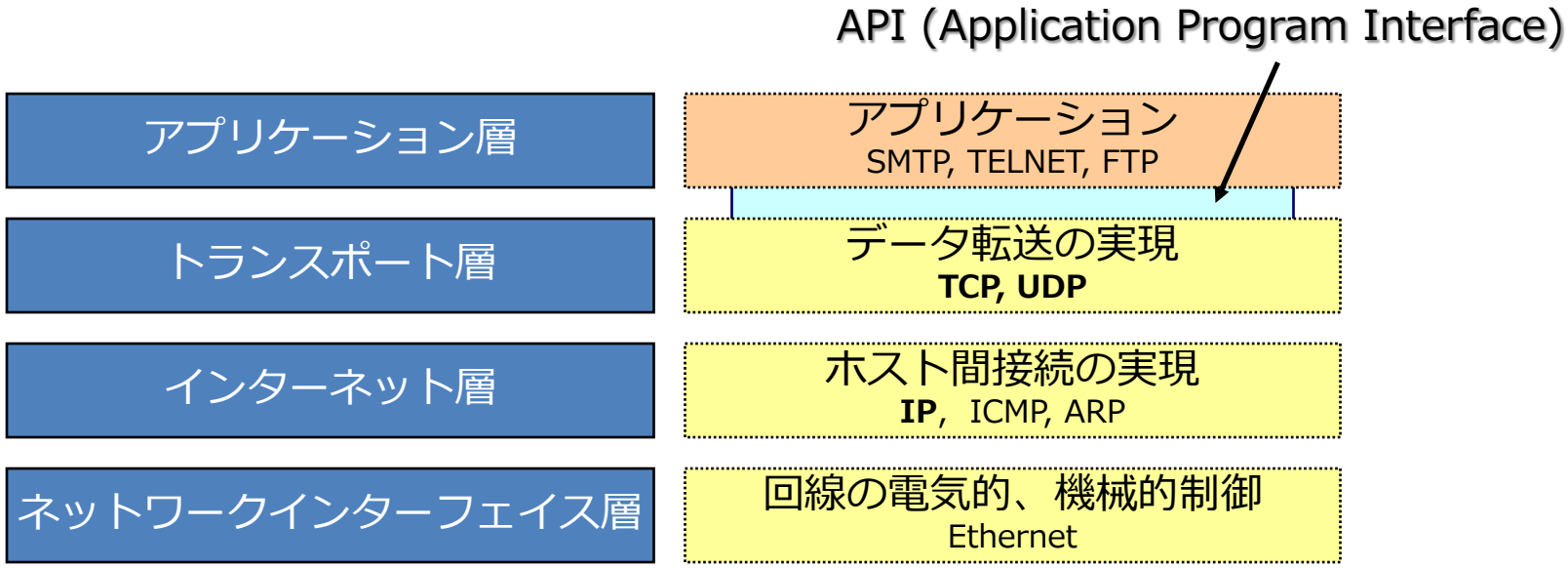
ソケットオプションの例

```
sockOptSize = sizeof(rcvBufferSize);

if ( getsockopt(sock, SOL_SOCKET, SO_RCVBUF,
                        &rcvBufferSize, &sockOptSize) < 0 ) {
   DieWithError("getsockopt() failed");
}
printf("Initial Recive Buffer Size: %d¥n", rcvBufferSize);

rcvBufferSize *= 2;
if ( setsockopt(sock, SOL_SOCKET, SO_RCVBUF,
                        &rcvBufferSize, sockOptSize) < 0 ) {
   DieWithError("setsockopt() failed");
}
```

ネットワークAPI

ネットワークAPI

- ネットワークを利用してデータを送受信するためのAPI
- UNIX系OSのソケットシステムコール
  - socket, bind, listen, connect, accept, read, write, recv, send, recvfrom, sendto, shutdown, close, …

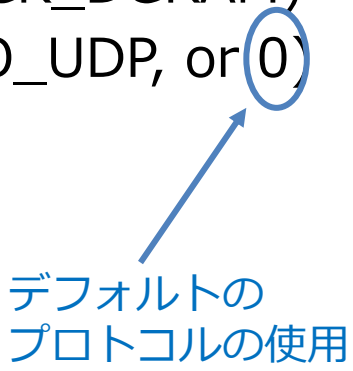


socket関数

- socket 関数 – ソケットの生成（取得）

```
int socket(int family, int type, int protocol);
```

  - 引数
    - family ：プロトコルファミリ　(PF_INET)
    - type　：ソケットタイプ　(SOCK_STREAM or SOCK_DGRAM)
    - protocol: プロトコル　(IPPROTO_TCP, IPPROTO_UDP, or 0)
  - 戻り値
    - 成功時　：　取得されたソケットディスクリプタ
    - エラー時: -1



close関数

- close関数 – ソケットの破棄（開放）

```
int close(int socket);
```

  - 引数
    - socket ：ソケットディスクリプタ
      - socket 関数の戻り値
  - 戻り値
    - 成功時　　：　0
    - エラー時　　：　-1

connect関数

- connect関数 – 接続要求の送信

```
int connect( int socket,
             sturct sockaddr *foreignAddress,
             u_int addressLength);
```

  - 引数
    - socket ：ソケット・ディスクリプタ（ socket 関数の戻り値 ）
    - foreignAddress：接続先のアドレス
      - sockaddr へのポインタ　⇒　sockaddr_in　へのポインタ
    - addressLength：接続先アドレスの長さ
      - sizeof(struct sockaddr_in) で求めらる長さ
  - 戻り値
    - 成功時　：　0
    - エラー時:　-1

send関数

- send 関数 – データの送信

```
int send(int socket, const void *msgBuffer,
                    u_int msgLength, int flags);
```

  - 引数
    - socket ：ソケットディスクリプタ（socket関数の戻り値）
    - msgBuffer：送信データへのポインタ
    - msgLength：送信データの大きさ（バイト数）
    - flags：設定の変更（0でデフォルト）
  - 戻り値
    - 成功時　：　実際に送信したデータの長さ（バイト数）
    - エラー時:　 -1

recv関数

- recv 関数 – データの受信

```
int recv(int socket, void *rcvBuffer,
                        u_int bufferLength, int flags);
```

  - 引数
    - socket ：ソケットディスクリプタ（socket関数の戻り値）
    - rcvBuffer：受信データの格納場所（バッファ）へのポインタ
    - bufferLength：一度に受信可能な最大データ長（バイト数）
    - flags：設定の変更（0でデフォルト）
  - 戻り値
    - 成功時　：　実際に受信したデータの長さ（バイト数）
    - エラー時:　 -1

bind関数

- bind関数 – ソケットとアドレスを結び付ける

```
int bind(int sockfd,
         struct sockaddr *addr, socklen_t len);
```

  - 引数
    - sockfd : ソケット・ディスクリプタ（socketの戻り値）
    - addr : サーバのアドレス
    - len : アドレスの長さ
  - 戻り値
    - 成功時 : 0
    - エラー時 : -1
  - Include
    - #include <sys/types.h>
    - #include <sys/socket.h>

listen関数

- listen関数 – 接続待ちのための準備

```
int listen(int sockfd, int backlog);
```

  - 引数
    - sockfd　：　ソケット・ディスクリプタ（socketの戻り値）
    - backlog　：　ソケットに対する待ち行列の大きさ
  - 戻り値
    - 成功時　：　0
    - エラー時：　-1
  - Include
    - #include <sys/socket.h>

accept関数

- accept関数 – 接続待ち状態に入り、接続を受付ける

```
int accept(int sockfd,
           struct sockaddr *addr, socklen_t *len);
```

  - 引数
    - sockfd : ソケット・ディスクリプタ（socketの戻り値）
    - addr : 接続されたクライアントのアドレス
    - len : アドレスの長さ
  - 戻り値
    - 成功時 : クライアントとの送受信に使うソケット ディスクリプタ
    - エラー時 : -1
  - Include
    - #include <sys/types.h>
    - #include <sys/socket.h>

sendto 関数

- sendto関数 – データの送信

```
int sendto(int socket, const void *msg,
    u_int msgLength, int flags,
    struct sockaddr *destAddr,  u_int addrLen);
```

  - 引数
    - socket　　：ソケットディスクリプタ （socket関数の戻り値）
    - msgBuffer　：送信データへのポインタ
    - msgLength　：送信データの大きさ（バイト数）
    - flags　　：設定の変更（0でデフォルト）
    - destAddr　　： 送信先のアドレス（sockaddr_in 構造体のポインタ）
    - addrLen　　： destAddr のアドレス長
  - 戻り値
    - 成功時　　：送信したデータの長さ（バイト数）
    - エラー時　：-1

recvfrom 関数

- recvfrom関数 – データの受信

```
int recvfrom(int socket, void *msg,
      u_int msgLength, int flags,
      struct sockaddr *srcAddr, u_int *addrLen);
```

  - 引数
    - socket　　：ソケットディスクリプタ （socket関数の戻り値）
    - Msg　　：受信データの格納場所（バッファ）へのポインタ
    - msgLength　：一度に受信可能な最大データ長（バイト数）
    - flags　　：設定の変更（0でデフォルト）
    - srcAddr　　：受信元のアドレス （sockaddr_in 構造体のポインタ）
    - addrLen　　：srcAddr のアドレス長が格納されている変数のアドレス
  - 戻り値
    - 成功時　：　受信したデータの長さ（バイト数）
    - エラー時: -1

memset 関数

- memset 関数 – バイト単位のデータを書き込む

```
void *memset(void *dst, int c, size_t nbytes);
```

  - 引数
    - dst　　：書き込み先（コピー先）の先頭アドレス
    - c　　：書き込む数値
    - nbytes　：書き込む（コピーする）大きさ（バイト）
  - 戻り値
    - dst へのポインタ

inet_addr 関数

- inet_addr 関数 – IPv4アドレスをネットワークバイトオーダーのバイナリデータに変更する

```
in_addr_t inet_addr(const char *cp);
```

  - 引数
    - cp　：ピリオドで区切ったIPv4アドレス文字列の先頭アドレス
  - 戻り値
    - アドレス情報（ in_addr_t (uint32_t)型）

htons 関数

- htons 関数 – ネットワーク・バイト・オーダーへ変換

```
u_short htons(u_short value);
```

  - 引数
    - value ：16 bit (unsigned short int) のホスト・バイト・オーダー値
  - 戻り値
    - 成功時：16 bit (unsigned short int) のネットワーク・バイト・オーダー値

Network Byte Order： Big-Edian

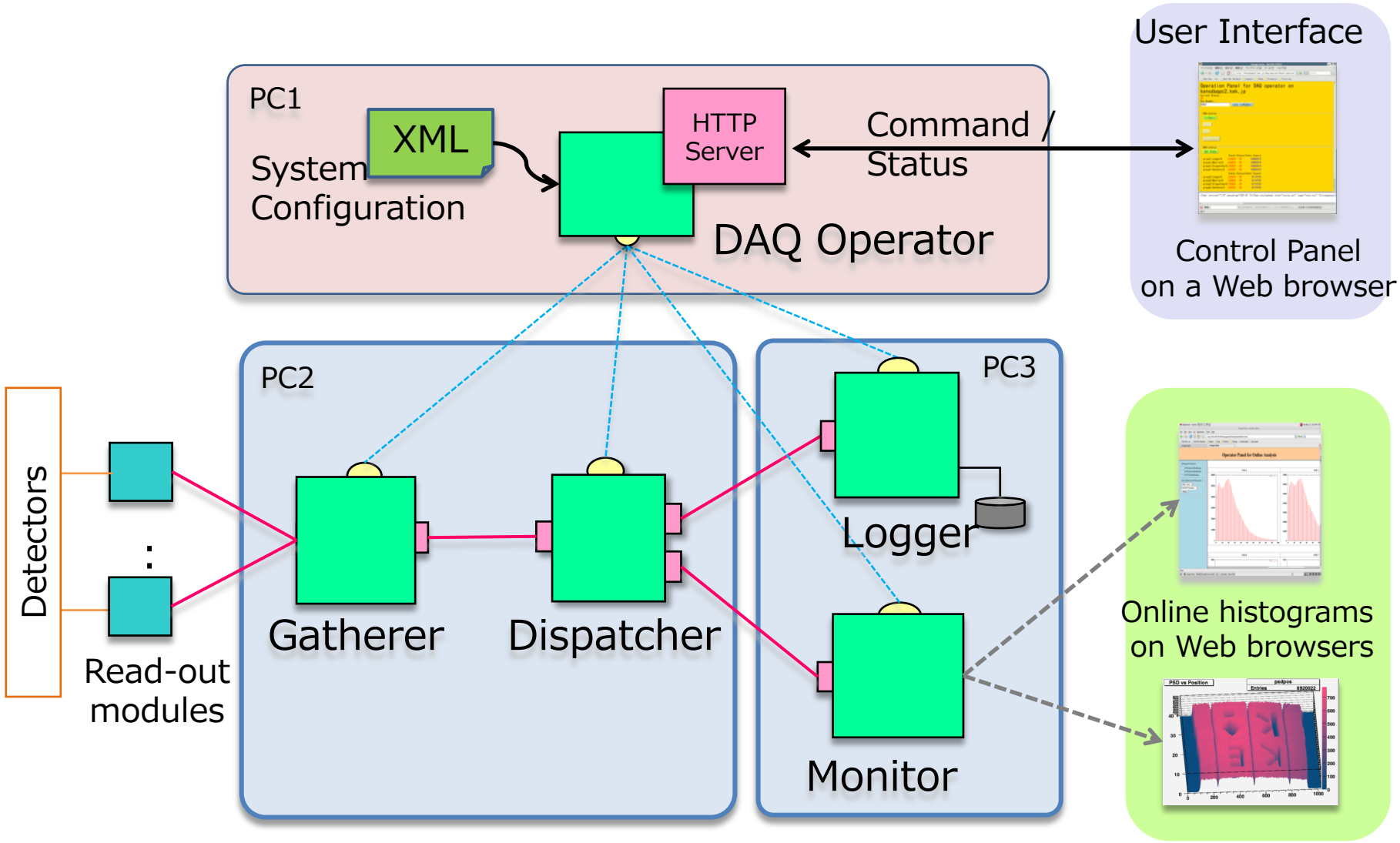
- Endian
  - Big-endian (MSB(Most Significat Byte) value first)
    - Ex. 0x12 0x34 (4660 decimal) – Motorola, and others
  - Little-endian (LSB(Least Significant Byte) value first)
    - Ex. 0x34 0x12 (4660 decimal) - Intel

データ収集フレームワーク
DAQ-MW

フレームワークとテンプレート

- **フレームワーク**
  - カスタマイズが可能なように作成された「半完成品ソフトウェア」
    - オブジェクト指向型クラスとして提供
    - 課題
      - 設計の自由度の低下
      - フレームワークの設計および選択が重要
      - フレームワークへの理解が必要
- **テンプレート**
  - 典型的な業務を標準化した「ひな型ソフトウェア」
    - 必要に応じてカスタマイズして再利用
    - ユーザが使用時にテンプレートを修正可能
      - サブルーチン部品がブラックボックス化されていない

DAQ-Middleware の例

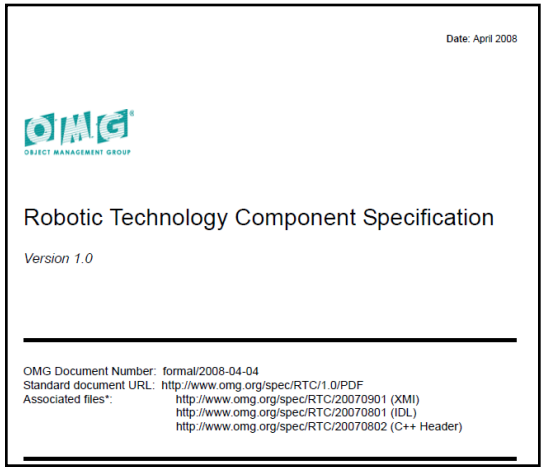


DAQ-Middleware（1）

- 目標
  - 再利用が容易な、柔軟性がある汎用のネットワークベースデータ収集（DAQ）ソフトウェアフレームワークを開発すること
  - 簡単に開発、設定、使用できること
- 適用領域
  - 中小規模実験
  - テストベッド　(測定器、エレクトロニクス等)

DAQ-Middleware（2）

- RT(Robot Technology)-Middlewareをデータ収集用に拡張
- RT-Middleware
  - ネットワークロボットシステムの構築のためのソフトウェア共通プラットフォーム
  - 産総研知能システム研究部門・タスクインテリジェンス研究グループが開発
  - 複数のコンポーネントが通信してひとつの機能を実現する
  - そのソフトウェアコンポーネントの仕様は国際標準規格（OMG）
  - 2006年から産総研と共同研究を行っている


Date: April 2008

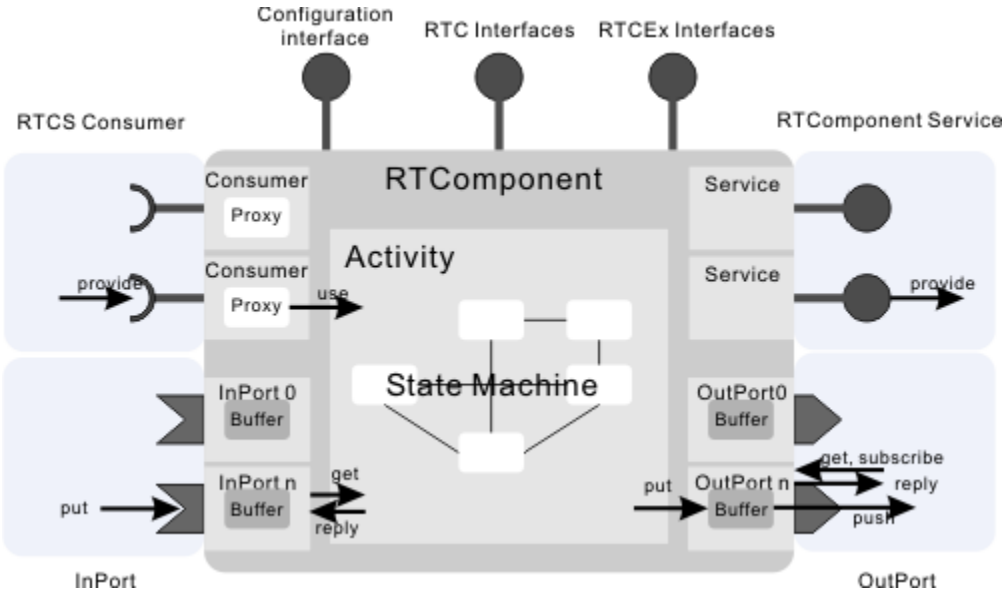
OMG

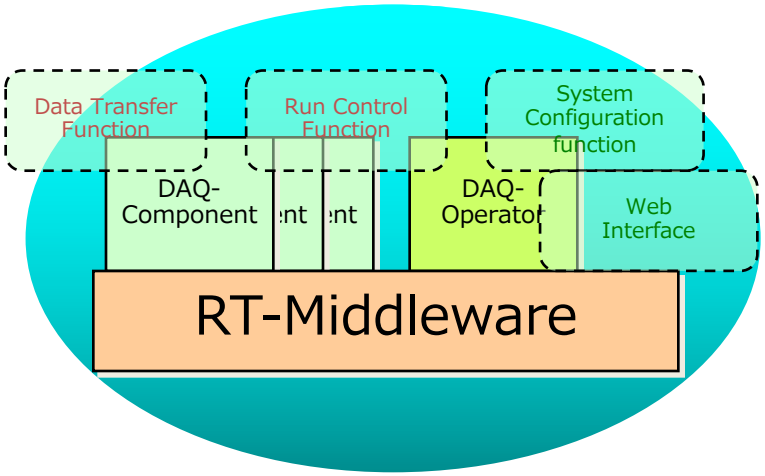
OBJECT MANAGEMENT GROUP

Robotic Technology Component Specification

Version 1.0

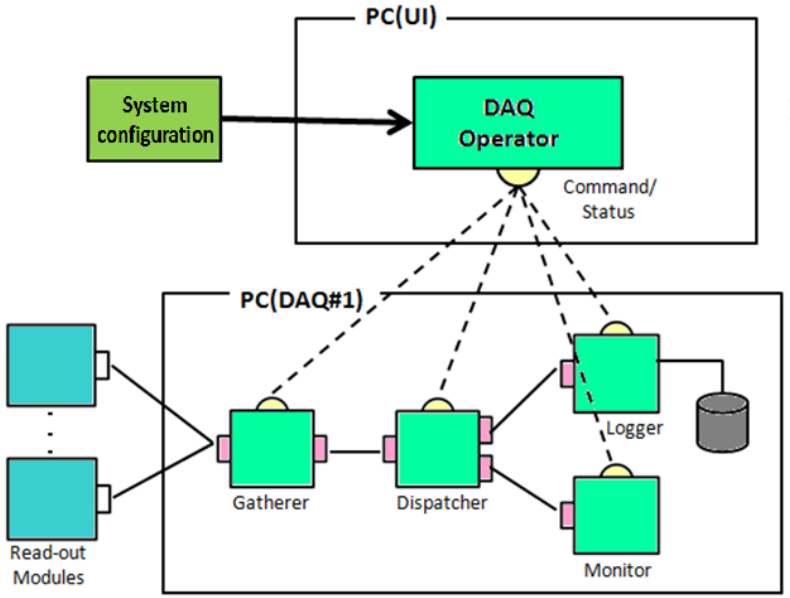
OMG Document Number: formal/2008-04-04
Standard document URL: http://www.omg.org/spec/RTC/1.0/PDF
Associated files*: http://www.omg.org/spec/RTC/20070901 (XMI)
http://www.omg.org/spec/RTC/20070801 (IDL)
http://www.omg.org/spec/RTC/20070802 (C++ Header)

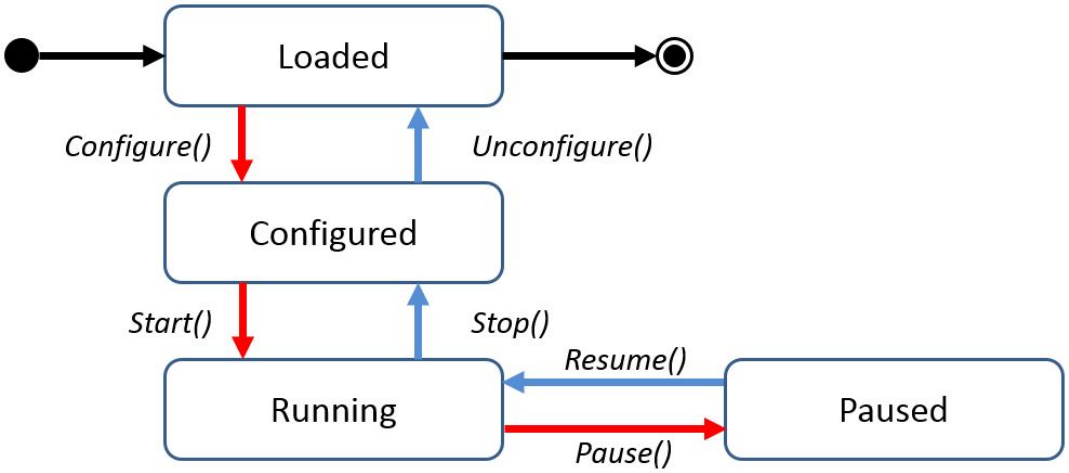




DAQ Operator

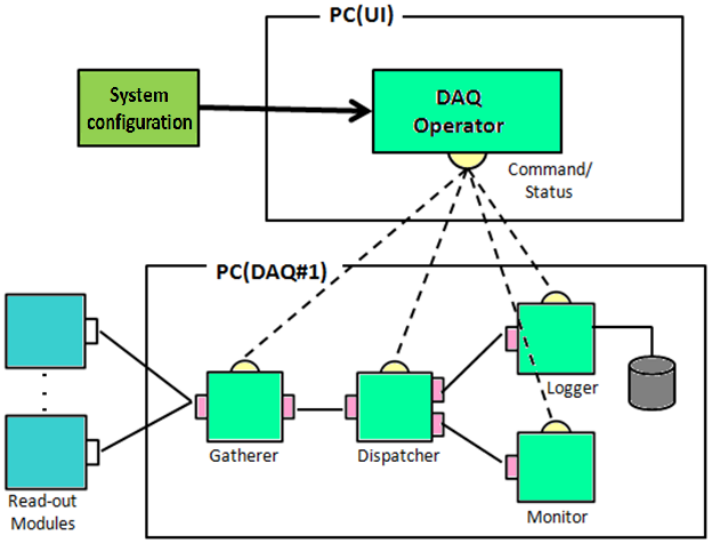
- コンフィグレーションファイルの読み込み
- 各コンポーネントの初期設定の制御
- コマンド送信によりシステム全体の制御
  - State（状態）の管理

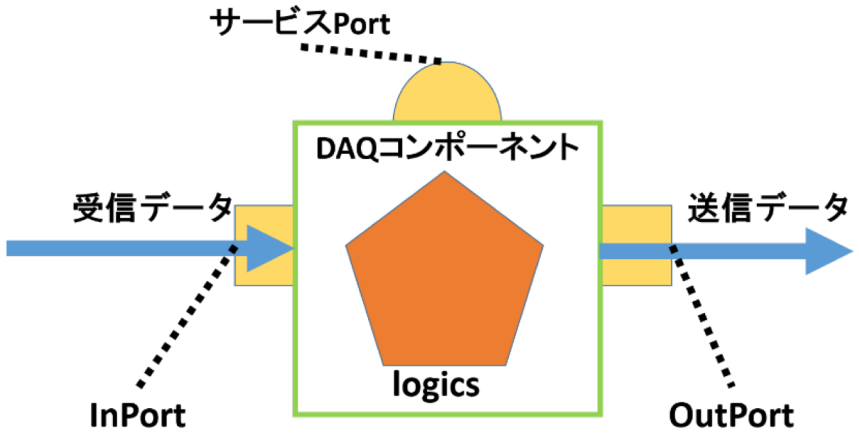




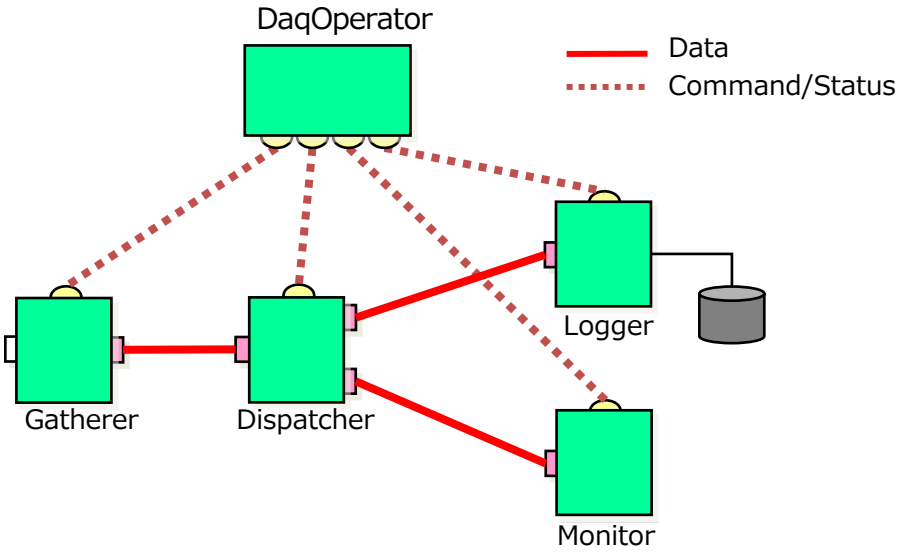
DAQ Component

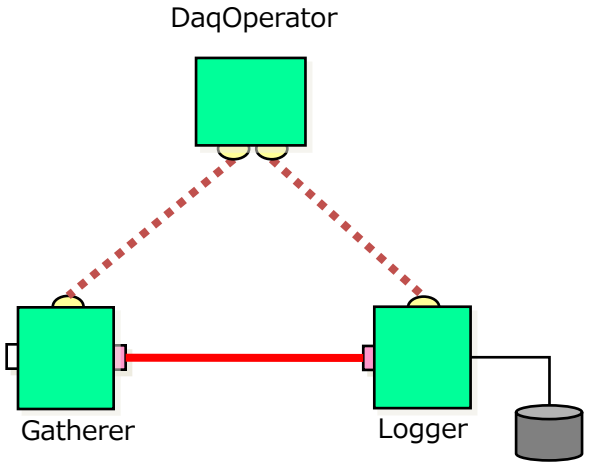
- ２種類のポート
  - Service Port：制御情報の送受信
  - InPort, OutPort：入出力ポート（複数可能）
- Logicに任意の機能を実装することが可能



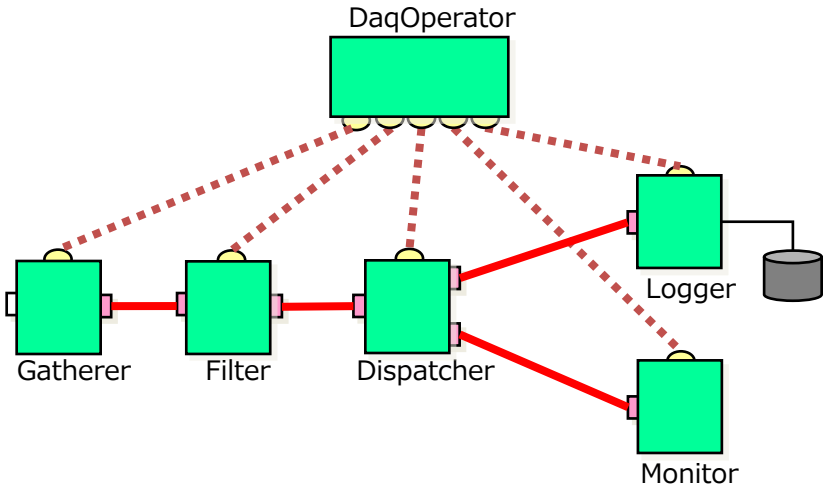


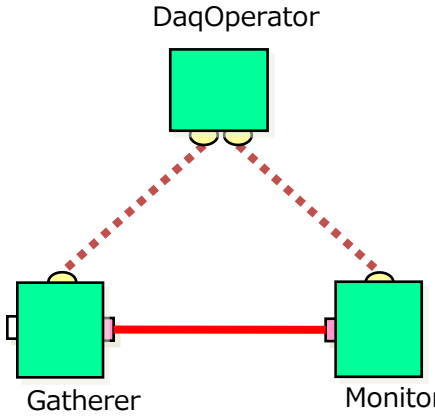
DAQコンポーネント 構成例 (1)





モニターなしでデータをディスクにセーブする

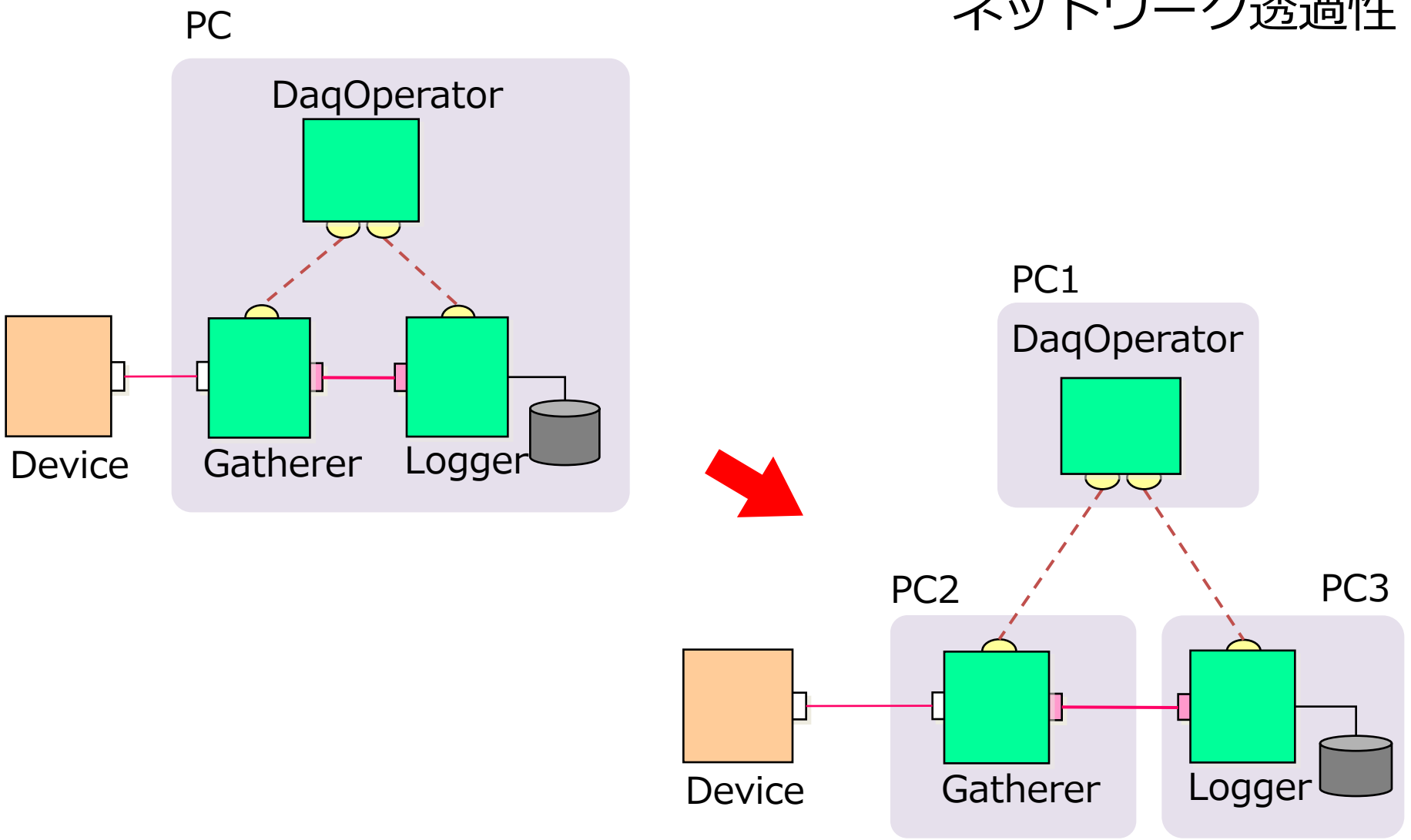




データセーブなしでオンラインモニターする

DAQコンポーネント 構成例 (2)

ネットワーク透過性



DAQ-Middleware 開発体制

- **2010年４月**
  DAQ-Middleware Core グループ結成

- メンバー
  - 千代、濱田 (KEK)
  - 長坂 (広島工業大学)
  - 味村 (大阪大学)
  - 神徳、安藤 (産業技術総合研究所)
  - 和田 ((株) Bee Beans Technologies)

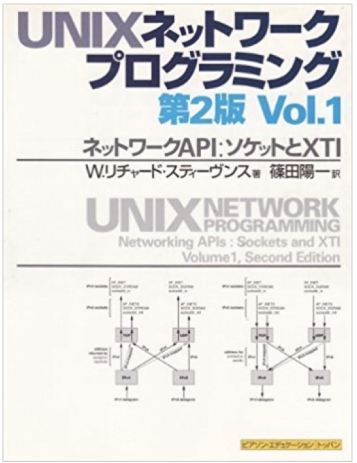
  - 仲吉(KEK, 2011年4月まで)
  - 安(KEK, 2012年3月まで)
  - 井上(KEK, 2015年3月まで)

DAQ-MW WebPage

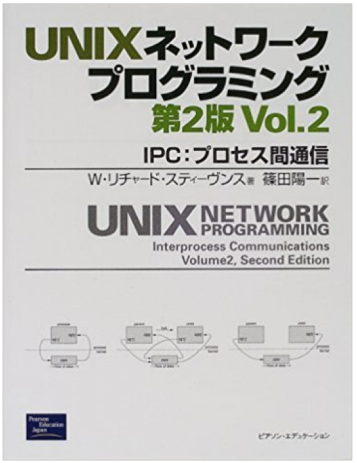
- http://daqmw.kek.jp/

ネットワークプログラミング 参考書

- **UNIXネットワークプログラミング〈Vol.1〉 ネットワークAPI・ソケットとXTI**
  - 著者：W.リチャード スティーヴンス, 篠田 陽一 (翻訳)
  - 出版社: ピアソンエデュケーション; 第2版 (1999/07)
  - ISBN-10: 4894712059



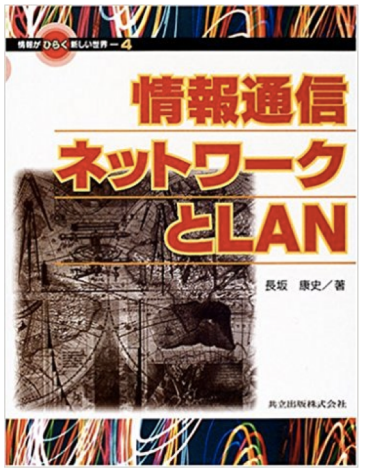
- **UNIXネットワークプログラミング〈Vol.2〉 IPC:プロセス間通信**
  - 著者：W.リチャード スティーヴンス, 篠田 陽一 (翻訳)
  - 出版社: ピアソンエデュケーション; 第2版 (2000/08)
  - ISBN-10: 4894712571



ネットワークプログラミング 参考書

- **TCP/IPソケットプログラミング C言語編**
  - 著者：Michael J. Donahoo, Kenneth L. Calvert, 小高 知宏 (翻訳)
  - 出版社: オーム社 (2003/5/1)
  - ISBN-10: 4274065197

- **情報通信ネットワークとLAN**
  - 著者：長坂康史
  - 出版社: 共立出版 (2001/06)
  - ISBN-10: 4320029666



まとめ

- 情報システムを構築するために必要な技術である情報システム化技術のうち、
  - ネットワークプログラミグ

  の知識を概観

- データ収集システム・フレームワークのDAQ-MWを概観
  - **http://daqmw.kek.jp/**