応用プログラミング

ex2

本日のレシピ

1. TCP/IP のしくみ
2. Socket（ソケット・インターフェイス）
3. echoClient.py
4. バイト(bytes)型と文字列(str)型
5. echoServer.py
6. コマンドライン引数

プロトコル

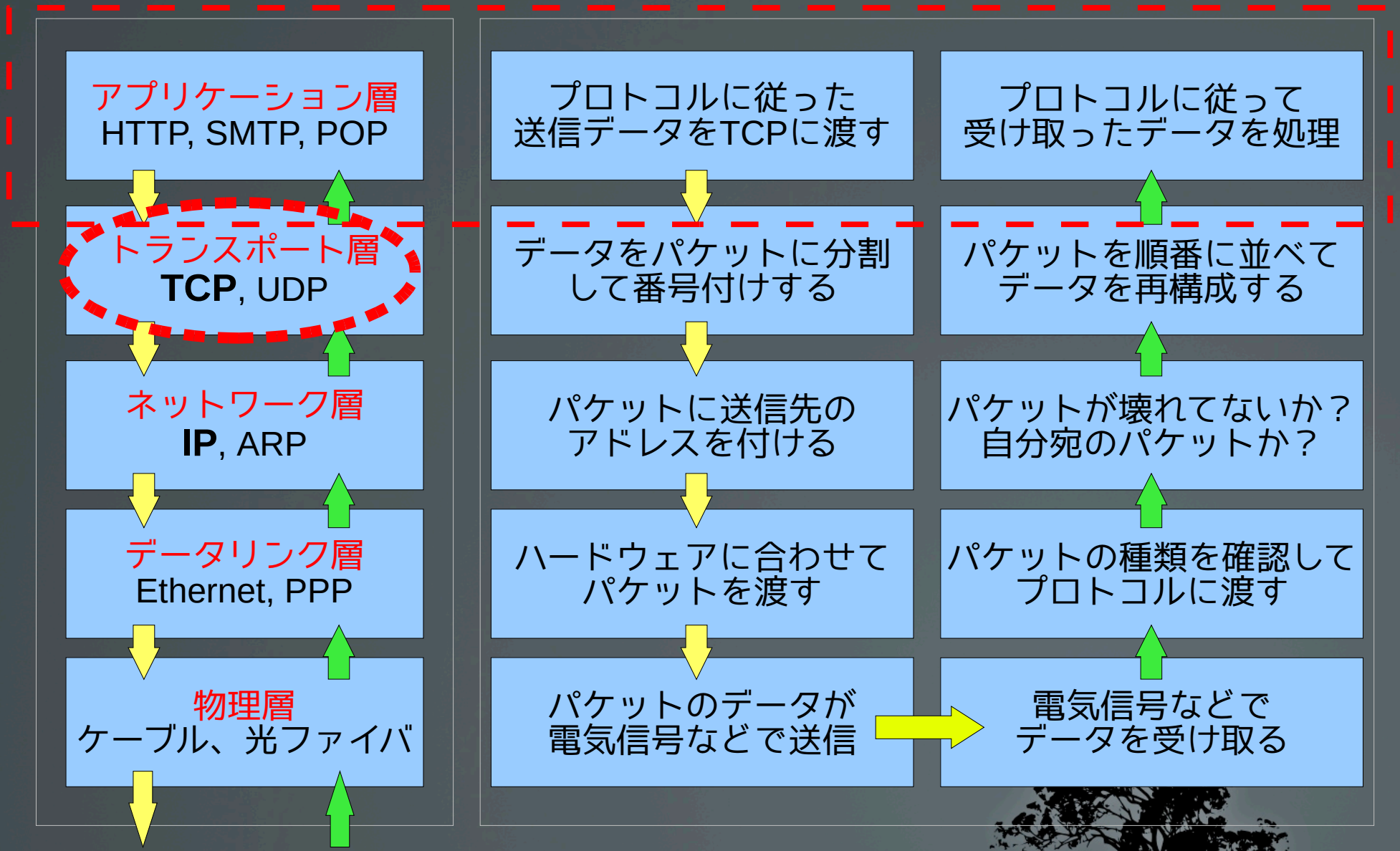
- プロトコル = 約束事、（通信）規約
  - 情報伝達における曖昧さを無くすための手続き方法
  - 情報伝達や手続きの方法・通信手順など
  - 使われる言葉や記号の意味、それの伴う動作など
  - TCP/IP プロトコル、京都議定書(*Kyoto Protocol*)

- この応用プログラミング（後半戦）では、

  TCP/IP プロトコルを用いたネットワークプログラミング

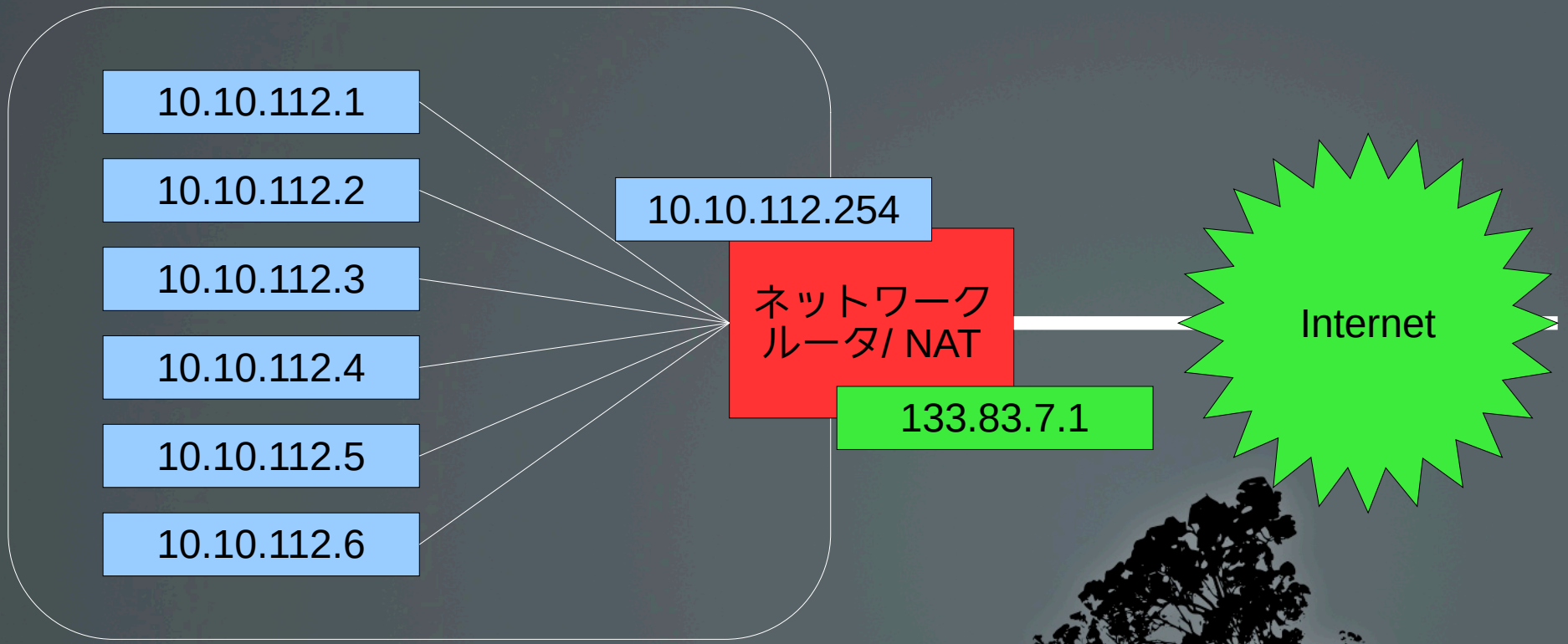
  を扱う。

プロトコル階層


アプリケーション層
HTTP, SMTP, POP
トランスポート層
TCP, UDP
ネットワーク層
IP, ARP
データリンク層
Ethernet, PPP
物理層
ケーブル、光ファイバ
プロトコルに従った
送信データをTCPに渡す
データをパケットに分割
して番号付けする
パケットに送信先の
アドレスを付ける
ハードウェアに合わせて
パケットを渡す
パケットのデータが
電気信号などで送信
電気信号などで
データを受け取る
パケットの種類を確認して
プロトコルに渡す
パケットが壊れてないか？
自分宛のパケットか？
パケットを順番に並べて
データを再構成する
プロトコルに従って
受け取ったデータを処理

グローバルIPとプライベートIP

- グローバルIPアドレス　：Internet上で一意
- プライベートIPアドレス：ローカルネット上で一意



ポート番号

- IPアドレス
  - ネットワーク上の計算機を識別
- ポート番号
  - 計算機上の通信プログラムを識別
  - 0～1023: Well-known Port Number
  - /etc/services
- 通信識別に必要な番号(association)
  - 送信元IPアドレス、送信先IPアドレス
  - 送信元ポート番号、送信先ポート番号
  - プロトコル番号(TCP=6, UDP=17)

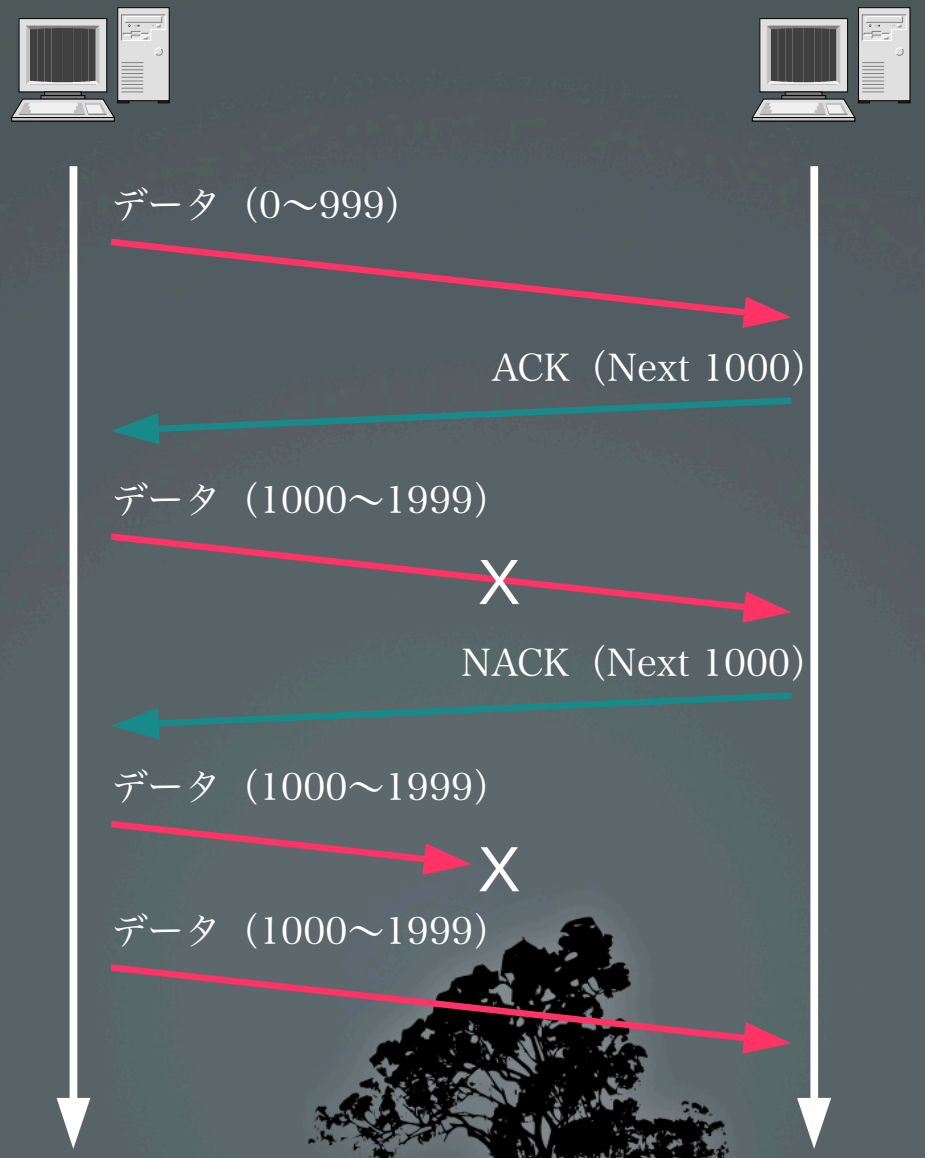
TCPとUDP

- TCP (Transmission Control Protocol), RFC793
  - ストリーム型（区切りが無い、10bytes×100 = 1000bytes）
  - 順序制御、再送制御、流量制御、輻輳制御
  - 信頼性

- UDP (User Datagram Protocol), RFC768
  - データグラム型（10bytes×100 = 10bytes×100）
  - 細かい制御はアプリケーション任せ
  - リアルタイム性、高速性（動画配信、IP電話など）

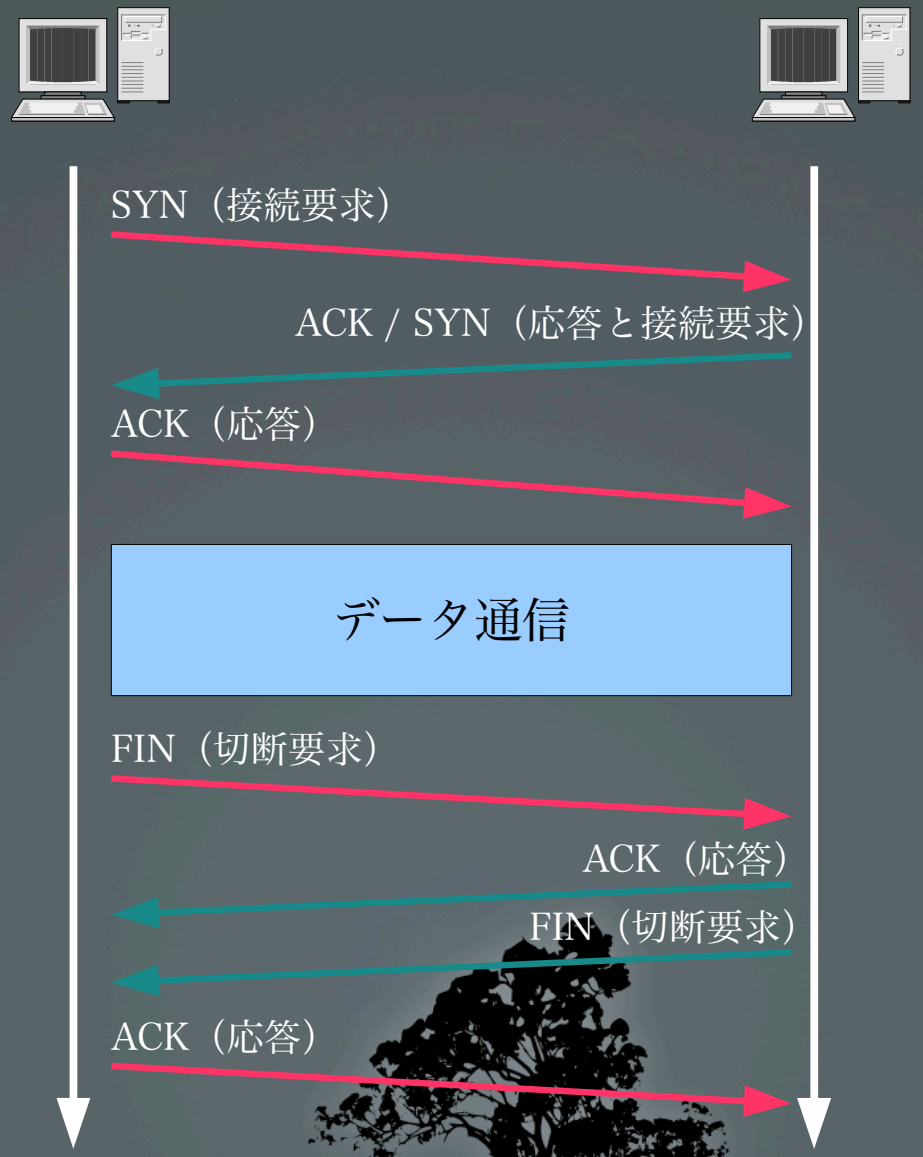
TCPの通信手順

- シーケンス番号
- 応答確認
  - ACK, NACK

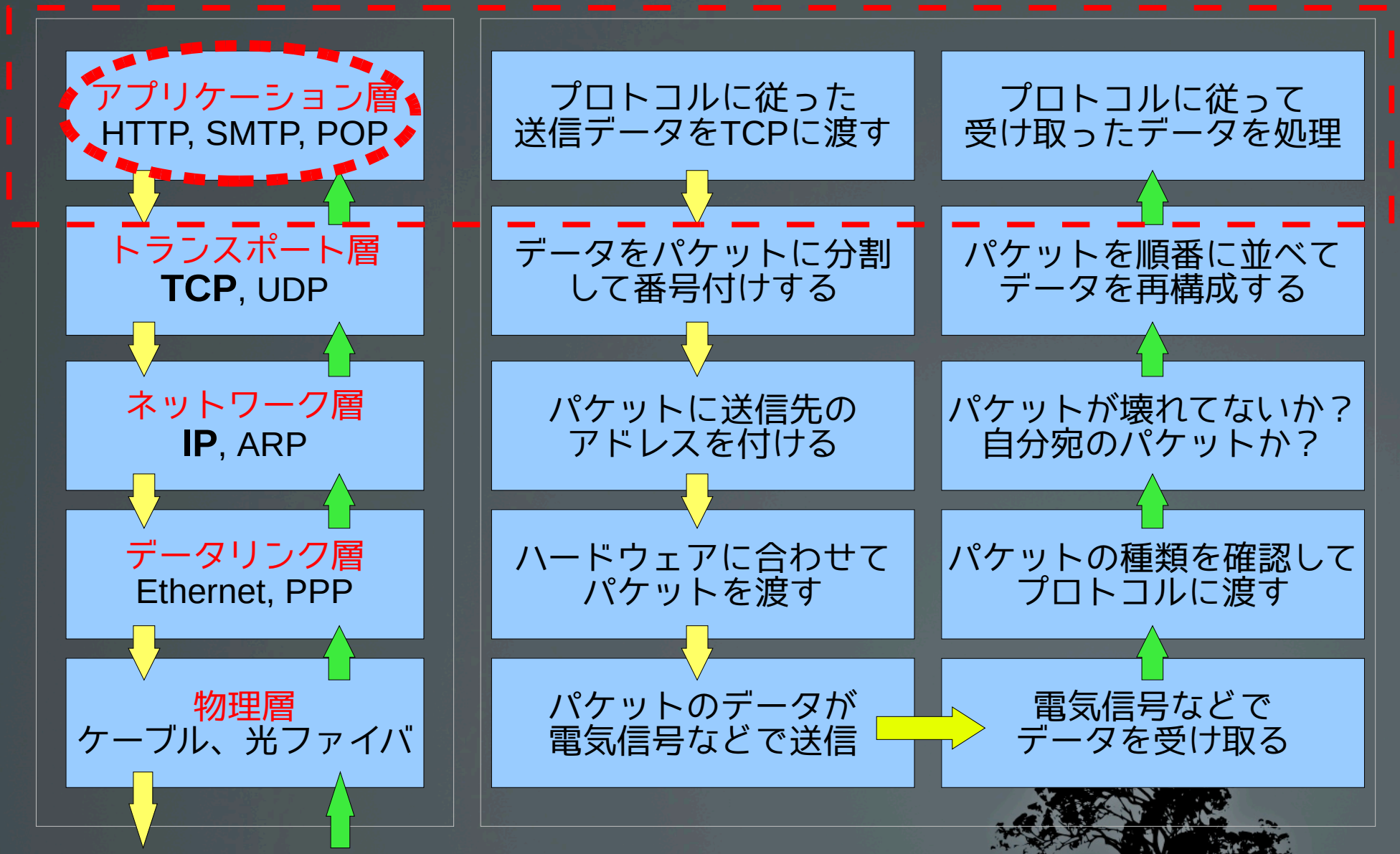


TCPのコネクション管理

- 通信の開始準備
- 通信の切断
- コネクション要求
  - SYN, FIN



プロトコル階層


アプリケーション層
HTTP, SMTP, POP
トランスポート層
TCP, UDP
ネットワーク層
IP, ARP
データリンク層
Ethernet, PPP
物理層
ケーブル、光ファイバ
プロトコルに従った
送信データをTCPに渡す
データをパケットに分割
して番号付けする
パケットに送信先の
アドレスを付ける
ハードウェアに合わせて
パケットを渡す
パケットのデータが
電気信号などで送信
電気信号などで
データを受け取る
パケットの種類を確認して
プロトコルに渡す
パケットが壊れてないか？
自分宛のパケットか？
パケットを順番に並べて
データを再構成する
プロトコルに従って
受け取ったデータを処理

HTTP プロトコル

- HyperText Transfer Protocol
- HTTP/0.9 (1991)
- HTTP/1.0 RFC1945(1996)
- HTTP/1.1 RFC2616(1999)
- HTTP/2　RFC7540(2015)
- 80番ポート
- URL/URI (Uniform Resource Locator/Identifier)

  http://www.math.ryukoku.ac.jp/index.html
- 送信要求の例

  > GET /index.html HTTP/1.0

  > [HTTP ヘッダ]

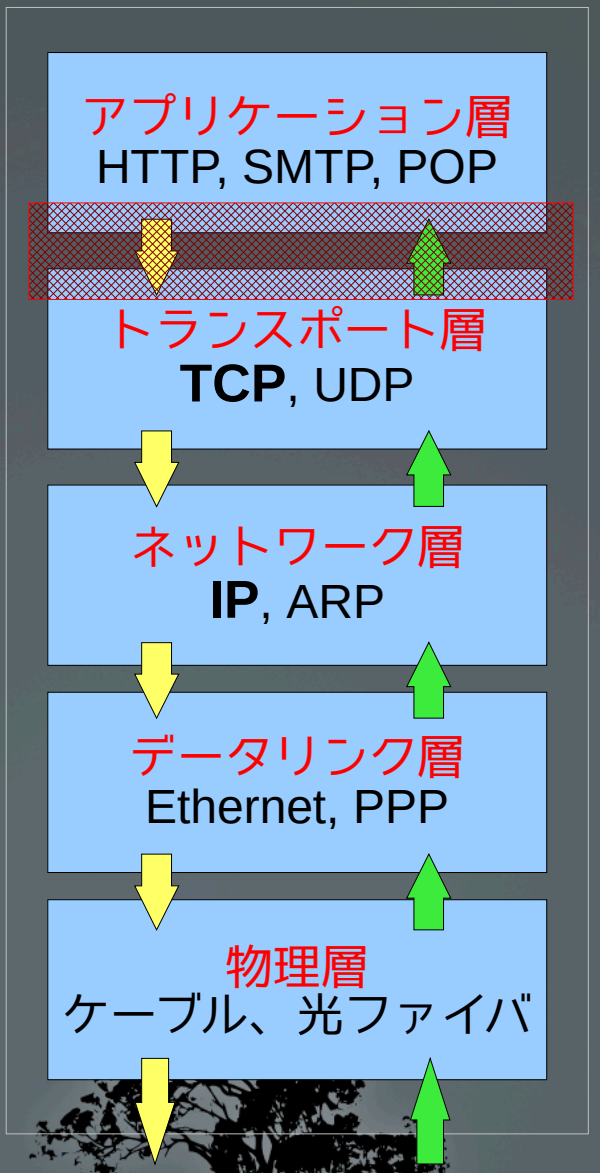
本日のレシピ

1. TCP/IP のしくみ
2. Socket（ソケット・インターフェイス）
3. echoClient.py
4. バイト(bytes)型と文字列(str)型
5. echoServer.py
6. コマンドライン引数

Socket（ソケットインターフェイス）

- アプリケーションの通信を抽象化
- ネットワーク通信用の汎用API (Application Programing Interface)
- ファイルの入出力と類似
  - 送信／書き込み、受信／読み出し

- import socket
  - socket()　　ソケットを作成
  - connect()　　通信先と接続
  - sendall()　　データを送る
  - recv()　　データを受け取る



本日のレシピ

1. TCP/IP のしくみ
2. Socket（ソケット・インターフェイス）
3. echoClient.py
4. バイト(bytes)型と文字列(str)型
5. echoServer.py
6. コマンドライン引数

echoClient.py

```
# Echo client program
import socket               # 低水準ネットワークインターフェイス

HOST = 'localhost'          # リモート（サーバ）ホスト名
PORT = 50007                # リモート（サーバ）ホストのポート番号

# IPv4(AF_INET)、TCP(SOCK_STREAM)用のソケットを作成
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    # ソケット s を使って HOST, POST に接続
    s.connect((HOST, PORT))         # 引数はタプル(ペア)として渡す
    send= b'Hello, world'

    # バイト型（文字列）send を s で送信
    s.sendall(send)
    print('Send', repr(send))

    # s から受信
    data = s.recv(1024)             # 受信する最大長は1024バイト
    print('Received', repr(data))

# with 構文は、前半6回・教科書 pp.340
```

本日のレシピ

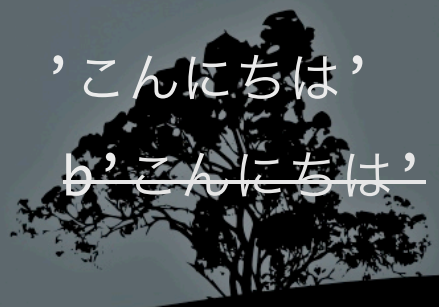
1. TCP/IP のしくみ
2. Socket（ソケット・インターフェイス）
3. echoClient.py
4. バイト(bytes)型と文字列(str)型
5. echoServer.py
6. コマンドライン引数

バイト(bytes)型：Python3

- これ → b'Hello World'
- これは文字列型 → 'Hello World'

- 文字列(str)型：文字コードに従って表示
  - 文字を表すデータ ＋ 文字コード（例: "A" = 65 + UTF-8コード）
- バイト(bytes)型：1バイトを1文字として表示
  - 文字を表すデータ ＋~~文字コード~~（1バイトのASCII文字のみ正しく表示される）

- 定数リテラル
  - 文字列型は基本的にUTF-8： 'Hello World'　'こんにちは'
  - バイト型はASCII文字のみ： b'Hello World'　~~b'こんにちは'~~



バイト型と文字列型の変換

- 文字列型 → バイト型
  - str.encode()
    - 'Hello World'.encode() → b'Hello World'
    - 'こんにちは'.encode() → b'\xe3\x81\x93\xe3\x82\x93\xe3\x81\xab\xe3\x81\xa1\xe3\x81\xaf'
  - bytes.decode() （ = bytes.decode('utf-8') ）
    - b'Hello World'.decode() → 'Hello World'
    - b'\xe3\x81\x93\xe3\x82\x93\xe3\x81\xab\xe3\x81\xa1\xe3\x81\xaf'.decode() → 'こんにちは'
- バイト(bytes)型
  - 文字を表すデータ + ~~文字コード~~

ネットワークとバイト型

- ネットワークを流れるデータは単なるバイト列
- socket.recv(1024) のよう(1024はバイト長)にバイト列として扱われる

- 文字列型に限らず、整数型や実数型もすべてバイト列に変換して送受信
- 送受信されたバイト列を元に戻す（バイト列がどんなデータなのかを解釈する）のはプログラムの仕事

- とはいえ、いろいろな型が混ざるとややこしいので、通常は「文字列型 ⇔ バイト型」変換して通信する

本日のレシピ

1. TCP/IP のしくみ
2. Socket（ソケット・インターフェイス）
3. echoClient.py
4. バイト(bytes)型と文字列(str)型
5. echoServer.py
6. コマンドライン引数

echoServer.py

```
# Echo server program
import socket

HOST = ''                 # サーバホスト名（'' とすると実行マシン上の接続可能な全てのホスト名）
PORT = 50007              # サーバポート番号

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    # ソケット s を指定されたホスト名とポート番号に紐付ける
    s.bind((HOST, PORT))
    # s を「接続待ち受け状態」に設定する
    s.listen(1)                           # 引数は接続待ちできるクライアントの最大数
    # s でクライアントから接続を待つ（プログラムは休止）、受信すると再開
    conn, addr = s.accept()               # 返り値は相手先ソケットとアドレスのタプル(ペア)
    # ソケットが存在していれば...
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)        # データを最大1024バイト繰り返し受け取る
            if not data: break            # 新たなデータが無ければ終了
            print('Received', repr(data))
            conn.sendall(data)
            print('Send', repr(data))
```

echoServer / echoClient 接続手順まとめ

本日のレシピ

1. TCP/IP のしくみ
2. Socket（ソケット・インターフェイス）
3. echoServer.py
4. バイト(bytes)型と文字列(str)型
5. echoClient.py
6. コマンドライン引数

コマンドライン引数

- たとえば

  > ls /home/t070000

  > cc program.c -o program -lm
- 実行時にプログラムへ値を渡す一手段
- Python:

  import sys

  sys.argv

- C : int main(int argc, char *argv[])

argv.py

```
# argv.py: sample code for sys.argv
import sys

args = sys.argv        # sys.argv がコマンド引数の文字列のリスト

argc = len(args)       # 引数の数はリストの長さでわかる

print('args =',args)
print('len(args) =', argc)

# 各引数はリストの要素として参照する
# リスト要素は全て文字列なので、数値として扱うためには int() などが必要
for i in range(argc):
    print('args[{0}] = {1}'.format(i, args[i]))
```

コマンドライン引数で echoServer.py のポート番号を指定できるようにする

- 引数が無ければポート番号は 50007
  python echoClient.py → port = 50007
- 引数があればポート番号を指定
  python echoClient.py 1234 → port = 1234