



# プログラミング演習2 ネットワーク通信

中村, 小松, 小林, 鹿喰

# 使ってますよね？



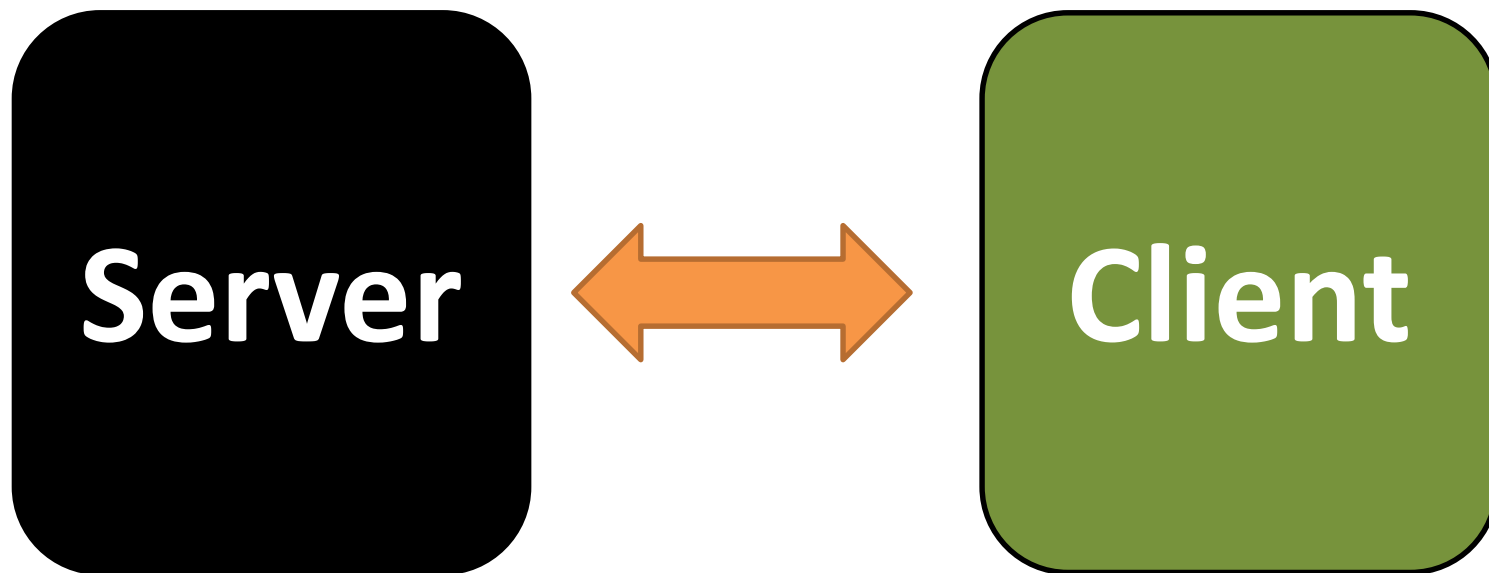
- LINE
- Twitter
- Facebook
- Skype
- FaceTime



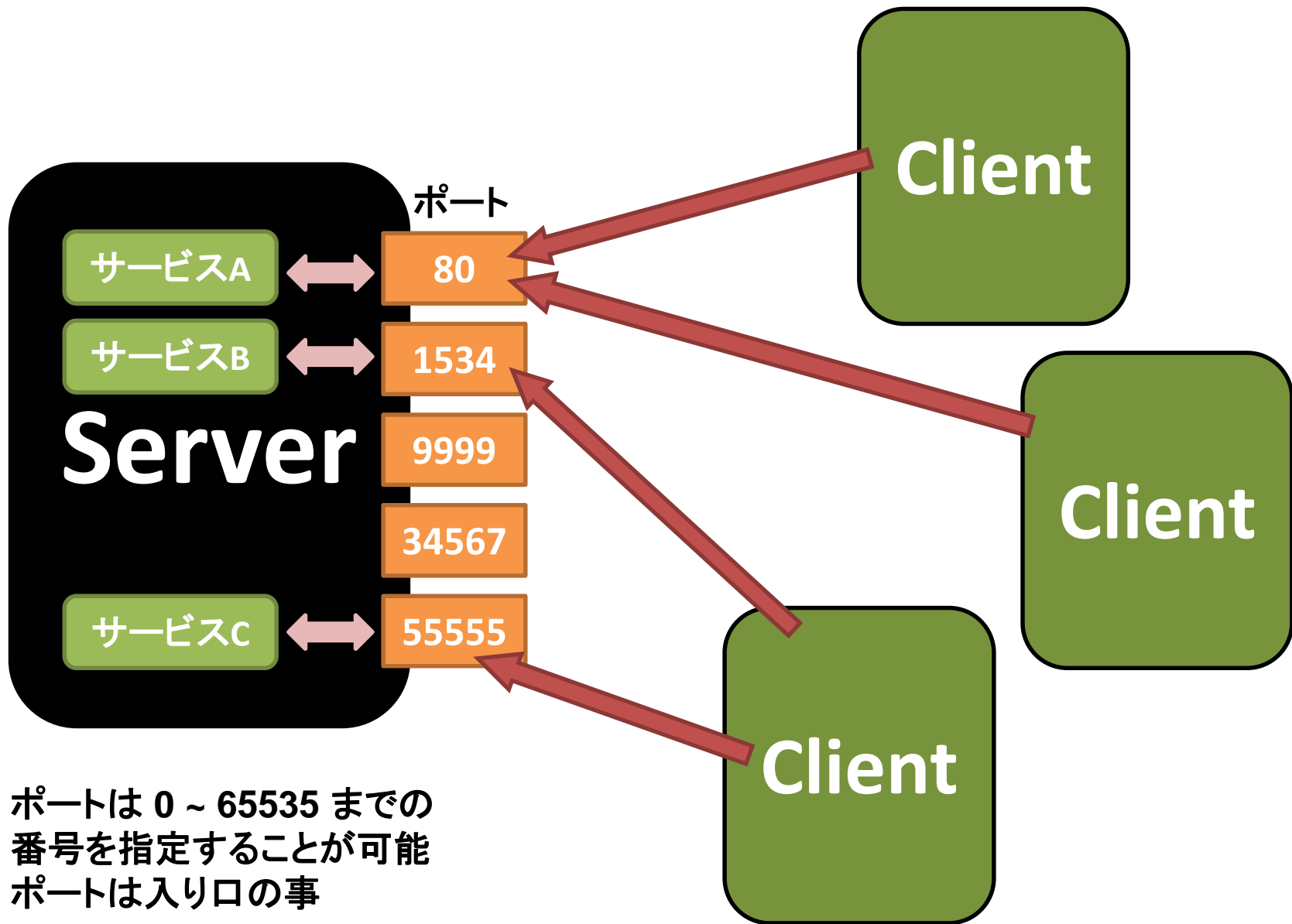
# 本日の内容



- ネットワーク通信で情報のやりとりをしよう



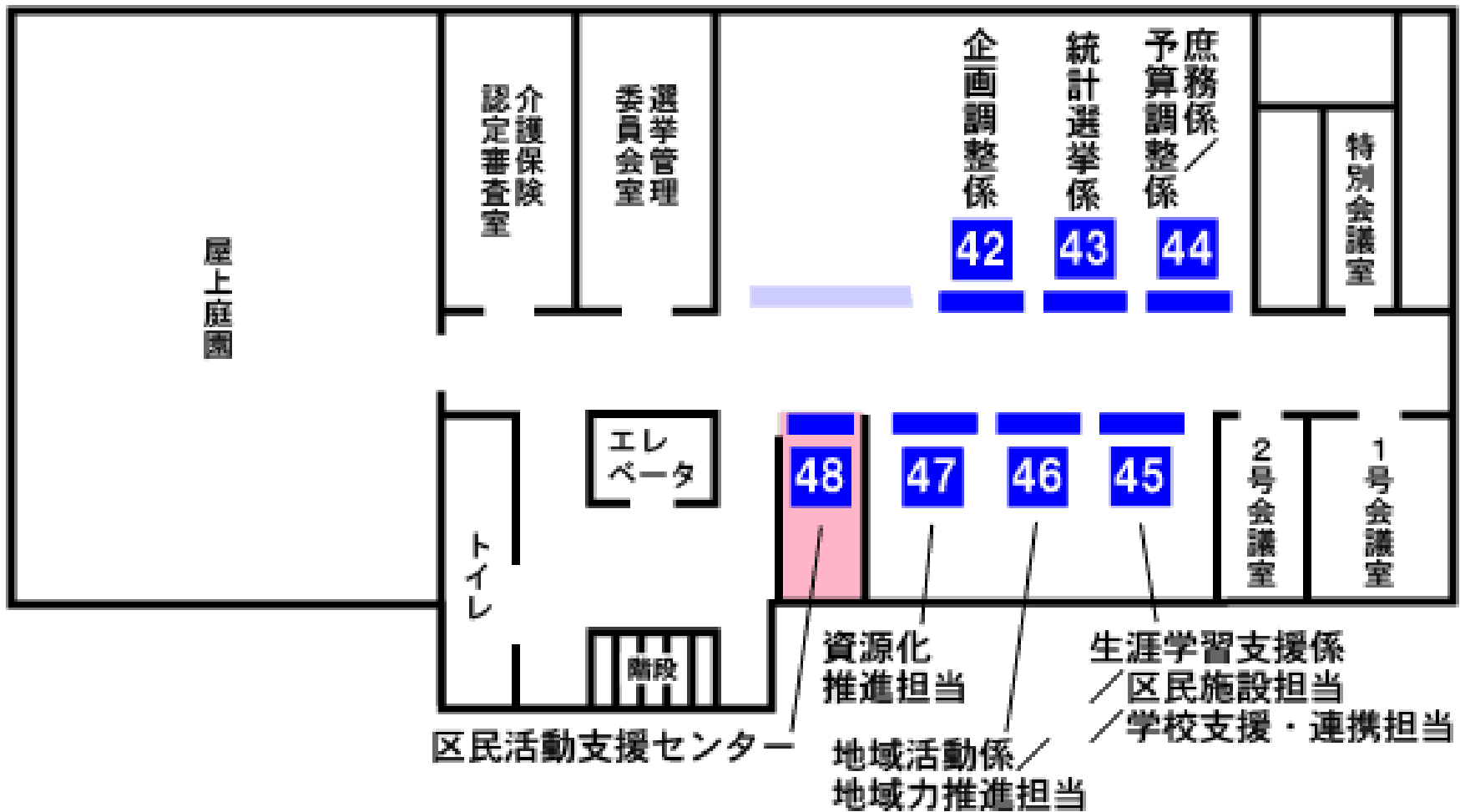
# 通信の基本



ポートは 0 ~ 65535 までの  
番号を指定することが可能  
ポートは入り口の事

# 窓口とサービスの対応関係

明治大学総合数理学部  
先端メディアサイエンス学科  
中村研究室



# 市役所で考えてみる



1. 役所の係の人が窓口を開けて準備する
2. 市民が役所(の住所)に行き, 窓口 に並ぶ
3. 窓口の人が最初に並んでいる市民を呼ぶ
4. 窓口で, 役所の人と, 市民が対話して何らかの処理を行う



# サーバ・クライアント



1. サーバがポートを開けて準備する
2. クライアントがサーバのアドレスとポート番号を指定してサーバに接続要求を行う
3. サーバがクライアントに対して接続許可を出すことで、情報交換のための経路が確立される
4. サーバとクライアント間でやりとりが行われる

# 注意点



- ファイル名やフォルダ名にServerやClientという文字列を含まない事
- 下記をプログラムに入れること

```
void stop(){  
    myClient.stop();  
}
```

```
void stop(){  
    myServer.stop();  
}
```

- 問題が解決しない場合は再起動すること
  - ファイアウォールに関する問題が出てきたら、再起動しないとダメな模様



# とりあえず書いてみよう



## クライアント (Client1.txt)

```
import processing.net.*;
Server myServer
    = new Server(this, 12345);
int count = 0;

void setup() {
    size(400, 200);
}

void draw() {
}

void stop(){
    myServer.stop();
}

void mousePressed() {
    count++;
    myServer.write(count);
}
```

## サーバ (Server1.txt)

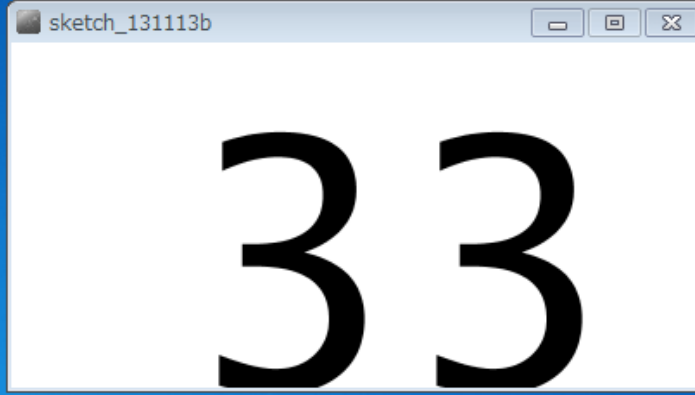
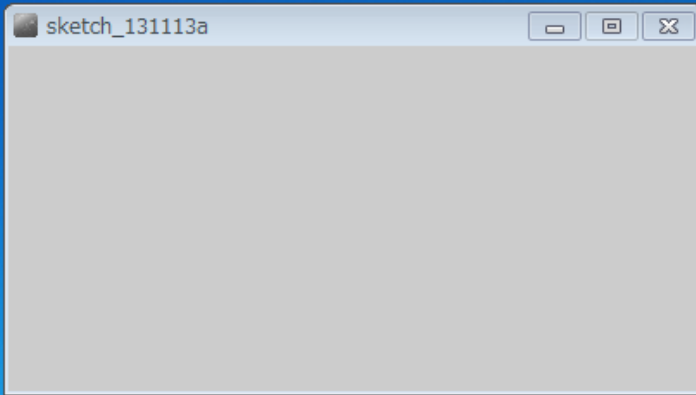
```
import processing.net.*;
Client myClient = new Client( this,
    "127.0.0.1", 12345 );

int recvInt;
void setup() {
    size(400, 200);
    textSize( 200 );
    fill( 0 );
}

void stop(){
    myClient.stop();
}

void draw() {
    background( 255 );
    if (myClient.available() > 0) {
        recvInt = myClient.read();
    }
    text( recvInt, 100, 200 );
}
```

127.0.0.1 は  
自分という意味



```
sketch_131113a | Processing 2.0.3
File Edit Sketch Tools Help
sketch_131113a
import processing.net.*;
Server myServer;
int count = 0;

void setup() {
  size(400, 200);
  myServer = new Server(this, 12345);
}

void mousePressed() {
```

サーバ(Server)

```
sketch_131113b | Processing 2.0.3
File Edit Sketch Tools Help
sketch_131113b
Client myClient;
int dataIn;

void setup() {
  size(400, 200);
  textSize(200);
  myClient = new Client(this,
}

void draw() {
  background(255);
```

クライアント(Client)

サーバ(Server)から先に起動すること！  
(窓口が準備できていないとだめ！)

サーバ側でクリック連打してみよう！！



- なんだか、自分の中でServerとかClientとか言ってもよくわからない。
- 隣の人とペアを作って接続してみよう！
  - 隣の人がない場合は、前後の余っている人同士でやってみる！
  - 一方はサーバ役，他方はクライアント役
    - 交互にやりましょう
  - まずは、隣の人アドレス(住所)を聞く！
    - 実際の住所ではなく、ネットワーク上の住所

# 隣に接続してみよう



スタートメニューを開く

cmdと入力して  
コマンドプロンプトを開く



# 隣に接続してみよう



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users¥130007>
```

**ipconfig と入力して実行**

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users¥130007>ipconfig
```

```
C:\Users¥130007>ipconfig

Windows IP 構成

イーサネット アダプター ローカル エリア接続:

    接続固有の DNS サフィックス . . . . . :
    IPv4 アドレス . . . . . : 133.26.54.5
    サブネット マスク . . . . . : 255.255.254.0
    デフォルト ゲートウェイ . . . . . : 133.26.54.254
```

**IPv4アドレスが各自の住所**

**IPv4アドレスを隣の人に伝えよう**

# とりあえず書いてみよう



```
import processing.net.*;  
Client myClient = new Client( this, "133.26.54.5", 12345 );  
int dataIn;
```

Server役の人の  
IPアドレスが133.26.54.5の場合

```
void setup() {  
    size(400, 200);  
    textSize( 200 );  
    fill( 0 );  
}  
void stop(){  
    myClient.stop();  
}  
void draw() {  
    background( 255 );  
    if (myClient.available() > 0) {  
        dataIn = myClient.read();  
    }  
    text( dataIn, 100, 200 );  
}
```

クライアント(Client)

# 準備と相互接続



- サーバ・クライアント用のライブラリの準備

```
import processing.net.*;
```

- サーバの準備 + 接続待ち

```
Server myServer = new Server( this,  
                               接続待ちポート番号 );
```

- クライアントの準備 + 接続

```
Client myClient = new Client( this,  
                               "接続先アドレス", 接続先ポート番号 );
```

# メッセージの送信



- サーバからクライアントへのメッセージの送信（整数または文字列を送信）

```
myServer . write( 100 );  
myServer . write( "hoge" );
```

- クライアントからサーバへのメッセージの送信（整数または文字列を送信）

```
myClient . write( 100 );  
myClient . write( "hoge" );
```



# クライアントでのデータ受信



- クライアントでのデータ受信 (整数)

```
int recvInt = myClient . read();
```

- クライアントでのデータ受信 (文字列)

```
String recvStr = myClient . readString();
```

- クライアントでのデータ受信量取得

```
int size = myClient . available();
```

# サーバでのデータ受信



- データ受信待ちしているクライアントの取得

```
Client nextClient = myServer . available() ;
```

- あるクライアントからのデータ受信 (整数)

```
int recvInt = myClient . read() ;
```

- あるクライアントからのデータ受信 (文字列)

```
String recvStr = myClient . readString() ;
```

# 通信終了



- サーバからの切断

```
myClient . stop() ;
```

- 特定のクライアントの切断

```
myServer . disconnect( thisClient ) ;
```

- サーバのサービスの終了

```
myServer . stop() ;
```

# 解説



133.26.54.5 さん

ライブラリ準備

```
import processing.net.*;
Server myServer
    = new Server(this, 12345);
```

```
int count = 0;
```

12345番ポートで準備

```
void setup() {
    size(400, 200);
}
```

```
void draw() {
```

```
}
void stop(){
    myServer.stop();
}
```

```
void mousePressed() {
    count++;
```

```
    myServer.write(count);
```

```
}

```

クリック時にcountを増やし  
クライアントにcountを送信！

133.26.54.6 さん

ライブラリ準備

```
import processing.net.*;
Client myClient = new Client( this,
    "133.26.54.5", 12345 );
```

```
int recvInt;
void setup() {
    size(400, 200);
    textSize( 200 );
    fill( 0 );
}
```

133.26.54.5さんの  
12345番ポートに接続

```
void stop(){
    myClient.stop();
}
```

available() で受信  
しているかチェック  
0より大きければ受信

```
void draw() {
    background( 255 );
    if (myClient.available() > 0) {
        recvInt = myClient.read();
    }
    text( recvInt, 100, 200 );
}
```

read() で値を受信して表示！

# 送受信の時に・・・



- どのようなフォーマット(形式)でデータを送受信するかを考える
- クリック数は1つの数字だけだが, ある座標を送受信しようと思うと, 2つの値が必要となるため困ったことに

```
myServer . write( mouseX );  
myServer . write( mouseY );
```



```
myServer . write( mouseX + "," + mouseY );
```

# 送受信の時に・・・



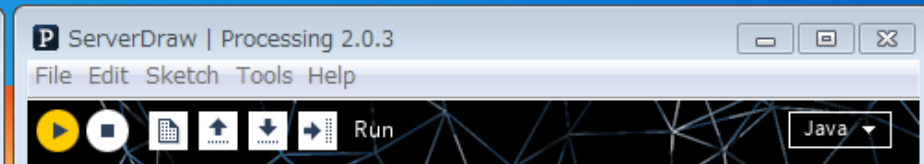
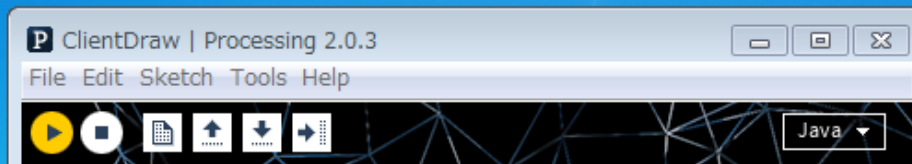
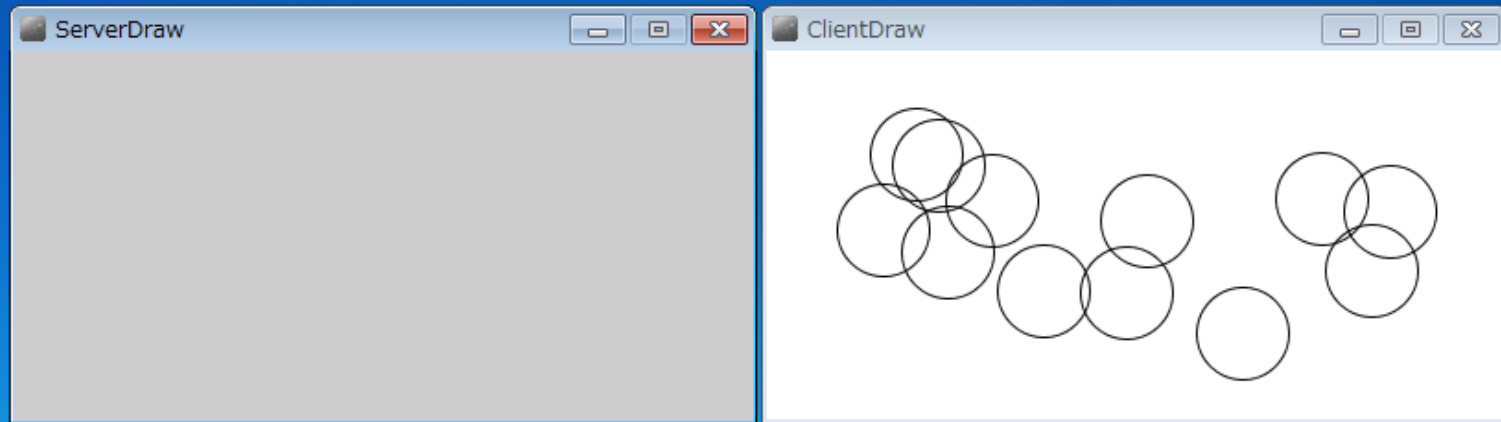
1. カンマ区切りで文字列として送信（他でもOK）
  - "x座標,y座標"（例）100,230
  - XXXX . **write( mouseX+", "+mouseY );**
2. 文字列として受信
  - String recvStr = YYYY . **readString();**
3. 受信した文字列（カンマ区切り）を, splitを利用してカンマで分割("100,230" → "100" と "230" に)
  - String [] data = **split( recvStr, ',' );**
4. 分割された文字列を整数に戻す
  - int x = **int( data[0] );**
  - int y = **int( data[1] );**

ファイル入出力でもやったよね！

# クライアントに描画する



- サーバで操作することで、クライアント上に描画してみる
  - サーバとクライアントで相互接続
  - サーバ上でクリックされた位置をクライアントに送信
  - クライアントで送信されてきた座標に円を描画



# サーバから描画してみる



## サーバ(Server2.txt)

```
// ライブラリを読み込む
import processing.net.*;
// サーバを2222ポートでスタート
Server myServer = new Server( this, 2222 );

void setup() {
  size(400, 200);
  noFill();
}

void mousePressed() {
  // クリック座標をカンマ区切りで送信
  myServer.write(mouseX+", "+mouseY);
}

void stop(){
  myServer.stop();
}
```

## クライアント(Client2.txt)

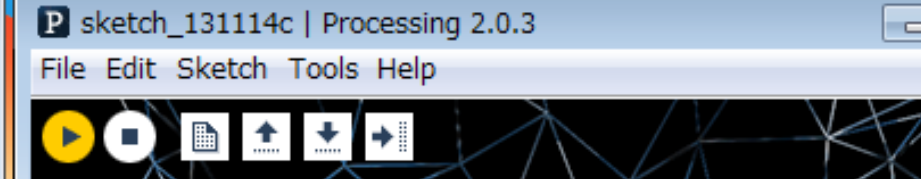
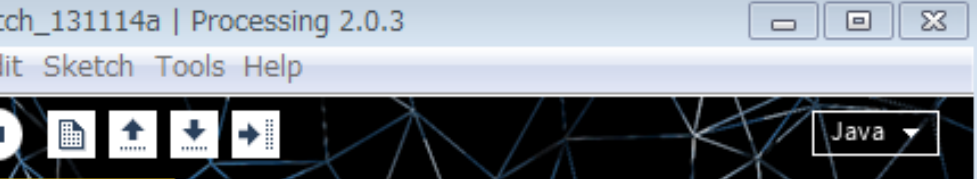
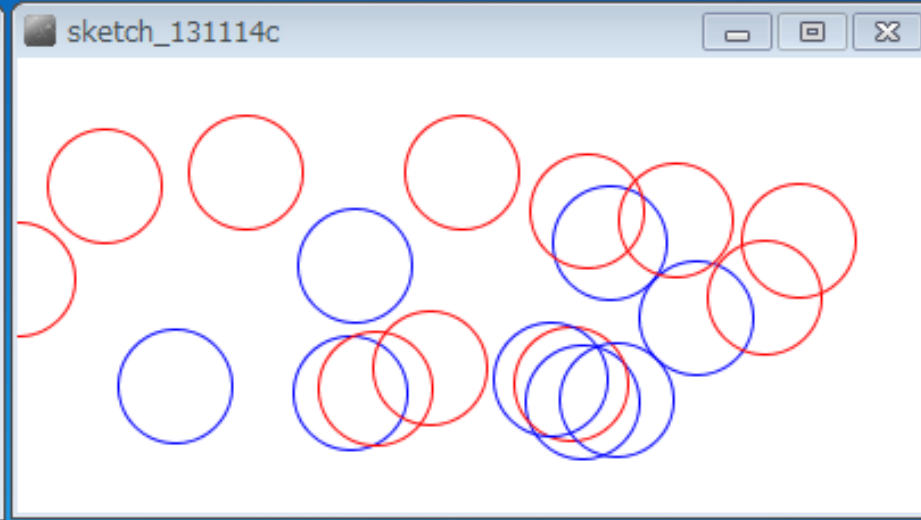
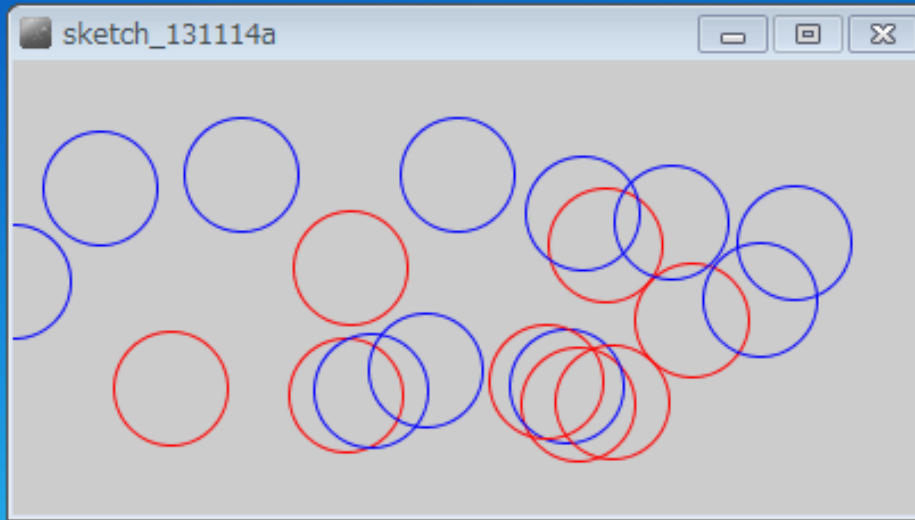
```
// ライブラリを読み込む
import processing.net.*;
// X.X.X.X の 2222 番に接続
Client myClient = new Client(this, "X.X.X.X", 2222);
void setup() {
  size(400, 200);
  noFill();
  background( 255 );
}
void stop(){
  myClient.stop();
}
void draw() {
  // 何か受信しているか確認
  if (myClient.available() > 0) {
    // 文字列として受信
    String msg = myClient.readString();
    // 文字列をカンマ区切りで分割し整数に戻す
    String [] data = split( msg, ',' );
    ellipse( int(data[0]), int(data[1]), 50, 50 );
  }
}
```



# クライアントに描画してみる



- 一方向だとなんだか面白く無い
- 双方向にしてみよう！
  - サーバでクリックするとクライアントに，クライアントでクリックするとサーバに！



# 双方向で描画してみる



## サーバ(Server)

```
import processing.net.*;
Server myServer = new Server( this, 2222 );
void setup() {
  size(400, 200);
  noFill();
}
void mousePressed() {
  myServer.write(mouseX+", "+mouseY);
}
void stop(){
  myServer.stop();
}
void draw(){
  // データ送信しているクライアントの確認
  Client nextClient = myServer.available();
  // != null はクライアントがあるという意味
  if( nextClient != null ){
    // readString() でデータ受信
    String recvStr = nextClient.readString();
    String [] data = split( recvStr, ',' );
    ellipse( int(data[0]), int(data[1]), 50, 50 );
  }
}
```

## クライアント(Client)

```
import processing.net.*;
Client myClient = new Client(this, "X.X.X.X", 2222);

void setup() {
  size(400, 200);
  noFill();
  background( 255 );
}

void mousePressed(){
  myClient.write( mouseX+", "+mouseY );
}

void stop(){
  myClient.stop();
}

void draw() {
  if (myClient.available() > 0) {
    String recvStr = myClient.readString();
    String [] data = split( recvStr, ',' );
    ellipse( int(data[0]), int(data[1]), 50, 50 );
  }
}
```

# 双方向で描画してみる v.2



## サーバ(Server)

```
import processing.net.*;
Server myServer = new Server( this, 2222 );
void setup() {
  size(400, 200);
  noFill();
}
void mousePressed() {
  myServer.write(mouseX+", "+mouseY);
  stroke( 255, 0, 0 );
  ellipse( mouseX, mouseY, 50, 50 );
}
void stop(){
  myServer.stop();
}
void draw(){
  Client nextClient = myServer.available();
  if( nextClient != null ){
    String recvStr = nextClient.readString();
    String [] data = split( recvStr, ',' );
    stroke( 0, 0, 255 );
    ellipse( int(data[0]), int(data[1]), 50, 50);
  }
}
```

## クライアント(Client)

```
import processing.net.*;
Client myClient = new Client(this, "X.X.X.X", 2222);
void setup() {
  size(400, 200);
  noFill();
  background( 255 );
}
void mousePressed(){
  myClient.write( mouseX+", "+mouseY );
  stroke( 255, 0, 0 );
  ellipse( mouseX, mouseY, 50, 50 );
}
void stop(){
  myClient.stop();
}
void draw() {
  if (myClient.available() > 0) {
    String recvStr = myClient.readString();
    String [] data = split( recvStr, ',' );
    stroke( 0, 0, 255 );
    ellipse( int(data[0]), int(data[1]), 50, 50 );
  }
}
```

# クライアントから命令！



- サーバ役とクライアント役, どちらかがどちらかになって下さい. また, IPアドレスをそれぞれサーバ役の人のにし, サーバを実行し, 次にクライアントを実行してみよう!

# サーバ(TrojanServer.txt)



```
import processing.net.*;
import java.awt.*;
import java.awt.event.*;
Server myServer = new Server( this, 3333 );
Robot robot;

void setup() {
  size(20, 20);
  try {
    robot = new Robot();
  } catch ( Exception e ) {
  }
}

void stop(){
  myServer.stop();
}
```

3333番ポートで準備

```
void draw() {
  Client nextClient = myServer.available();
  if ( nextClient != null ) {
    String recvStr = nextClient.readString();
    String [] data = split( recvStr, ',' );
    if ( data[0].equals( "move" ) == true ) {
      doMove( int( data[1] ), int( data[2] ) );
    }
  }
}
```

最初のカンマの前が  
動作に関すること

```
void doClick( int x, int y ) {
  robot.mousePress( InputEvent.BUTTON1_MASK );
  delay(1);
  robot.mouseRelease( InputEvent.BUTTON1_MASK );
  delay(1);
}

void doMove( int x, int y ) {
  robot.mouseMove( x, y );
}
```

# クライアント (TrojanClient.txt)



```
import processing.net.*;  
Client myClient = new Client( this, "133.26.54.5", 3333 );
```

```
void setup() {  
  size( 1200, 800 );  
}
```

133.26.54.5さんの  
3333番ポートに接続

```
void stop(){  
  myClient.stop();  
}
```

```
void draw() {  
}
```

```
// マウスが押された時
```

```
void mousePressed() {  
  // "click,100,80" のように送信
```

```
}  
// マウスが移動した時  
void mouseMoved() {  
  // "move,100,80" のように送信  
  myClient.write( "move,"+mouseX+", "+mouseY );  
}
```

write() で  
メッセージを送信

# やってること



- サーバのマウスを操作してみよう！
  - "move,x,y" でマウスの移動情報を送信
- トロイの木馬でやっているのは, これの更に高度なこと



演習：クリックを追加してみよう！

# (参考)文字入力



- RobotのKeyPressを利用することで、相手のコンピュータに対して文字入力も指示できる！
- 興味がある人はやってみよう！！
  
- ライブラリのロード
  - `import java.awt.event.KeyEvent;`
- キープレス
  - `robot.KeyPress( KeyEvent.VK_キーの種類 );`



# (参考)スクリーンキャプチャ

明治大学総合数理学部  
先端メディアサイエンス学科  
中村研究室



- RobotのcreateScreenCaptureを利用することで、画面のスクリーンキャプチャを撮ることが可能
- つまり、相手のデスクトップを見ることが可能
  - リモートからコンピュータを操作する際には、こういった組み合わせでやっています

# LINEもどきを作ってみよう

明治大学総合数理学部  
先端メディアサイエンス学科  
中村研究室



- スタンプを相手に貼り付けよう！





```
import processing.net.*;
Server myServer = new Server( this, 5555 );
PImage [] stamp = new PImage [7];
int [] stampMessage = new int [10];
int [] stampFrom = new int [10];
int lastMessageNum = -1;

void setup() {
  size(300, 800);
  stamp[0] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/40/580x580x42472428b7971cefc004cf3d.jpg" );
  stamp[1] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/38/197x212x553fc9733bf33ba25fabc8de.jpg" );
  stamp[2] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/61/260x260x0860b5b9e2ff67b2d7f02278.jpg" );
  stamp[3] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/22/339x246xdfc75cc1c6b692462c54c6e7.jpg" );
  stamp[4] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/29/399x351x7d40231ed0ad556af902e19d.jpg" );
  stamp[5] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/30/232x222xb26506eb0d3173a63f8070f2.jpg" );
  stamp[6] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/128/228x230x425a49f2bac2e3770f407435.jpg" );
}

void mousePressed() {
  lastMessageNum++;
  if ( lastMessageNum == 8 ) {
    lastMessageNum = 0;
  }

  int id = 1 + (int)(mouseX / (width/7) );
  myServer.write( id );
  stampMessage[lastMessageNum] = id;
  stampFrom[lastMessageNum] = 1;
}

void stop(){
  myServer.stop();
}

void draw() {
  background( 255 );
  Client nextClient = myServer.available();
  if ( nextClient != null ) {
    int id = nextClient.read();
    lastMessageNum++;
    if ( lastMessageNum == 8 ) {
      lastMessageNum = 0;
    }
    stampMessage[lastMessageNum] = id;
    stampFrom[lastMessageNum] = 2;
  }

  for ( int i=0; i<8; i++ ) {
    int id = stampMessage[i];
    if ( stampFrom[i] == 1 ) {
      // 1なら左に表示
      image( stamp[id-1], 180, i*100, 100, 100 );
    }
    else if ( stampFrom[i] == 2 ) {
      // 2なら右に表示
      image( stamp[id-1], 20, i*100, 100, 100 );
    }
  }
}
```

# クライアント



```
import processing.net.*;
//Client myClient = new Client(this, "133.26.54.5", 2222);
Client myClient = new Client(this, "127.0.0.1", 5555);
PImage [] stamp = new PImage [7];
int [] stampMessage = new int [10];
int [] stampFrom = new int [10];
int lastMessageNum = -1;

void setup() {
  size(300, 800);
  background( 255 );
  stamp[0] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/40/580x580x42472428b7971cefc004cf3d.jpg" );
  stamp[1] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/38/197x212x553fc9733bf33ba25fab8c8de.jpg" );
  stamp[2] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/61/260x260x0860b5b9e2ff67b2d7f02278.jpg" );
  stamp[3] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/22/339x246xdfc75cc1c6b692462c54c6e7.jpg" );
  stamp[4] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/29/399x351x7d40231ed0ad556af902e19d.jpg" );
  stamp[5] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/30/232x222xb26506eb0d3173a63f8070f2.jpg" );
  stamp[6] = loadImage( "http://imgcc.naver.jp/kaze/mission/USER/20121015/78/718648/128/228x230x425a49f2bac2e3770f407435.jpg" );
}

void mousePressed() {
  lastMessageNum++;
  if ( lastMessageNum == 8 ) {
    lastMessageNum = 0;
  }

  int id = 1 + (int)(mouseX / (width/7) );
  myClient.write( id );
  stampMessage[lastMessageNum] = id;
  stampFrom[lastMessageNum] = 1;
}

void stop(){
  myClient.stop();
}

void draw() {
  background( 255 );
  if (myClient.available() > 0) {
    int id = myClient.read();
    lastMessageNum++;
    if ( lastMessageNum == 8 ) {
      lastMessageNum = 0;
    }

    stampMessage[lastMessageNum] = id;
    stampFrom[lastMessageNum] = 2;
  }

  for ( int i=0; i<8; i++ ) {
    int id = stampMessage[i];
    if ( stampFrom[i] == 1 ) {
      // 1なら左に表示
      image( stamp[id-1], 180, i*100, 100, 100 );
    }
    else if ( stampFrom[i] == 2 ) {
      // 2なら右に表示
      image( stamp[id-1], 20, i*100, 100, 100 );
    }
  }
}
```

# 課題1



- 講義資料に登場したLINEもどきを提出せよ
  - プログラム名はLineServer, LineClient
- 画像や操作をオリジナルのものから変更し, 工夫してみよう

# サーバに情報を送ろう！



- 中央サーバをクリックの度にサーバに対してメッセージを送り続けるクライアントプログラムを作成せよ
  - write( "click" ) で文字列データを送信せよ
  - ポート番号は 54321 とする

サーバの負荷を減らすのは大変

# サーバに情報を送ろう！

明治大学総合数理学部  
先端メディアサイエンス学科  
中村研究室



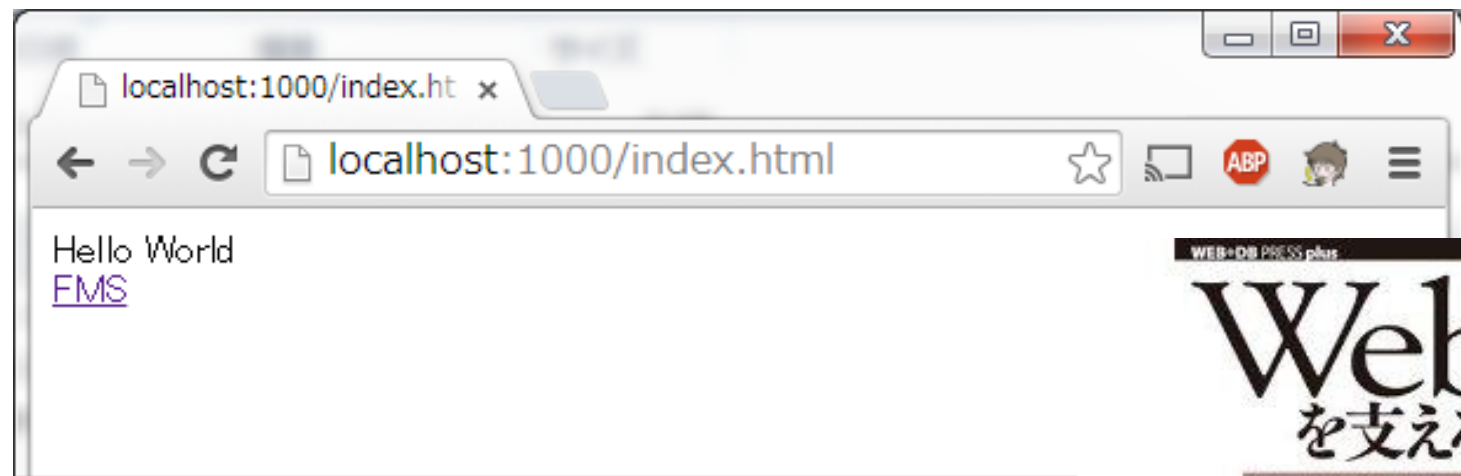
```
import processing.net.*;
Client myClient = new Client(this,  , );
int click = 0;
void setup() {
    size( 100, 100 );
}
void stop(){
    myClient.stop();
}
void draw(){
}

void mousePressed() {
    myClient.write(  );
}
```

# 簡易ウェブサーバを作ろう



- Webサーバってどんな仕組みで成り立っているのだろうか？（詳細はWebを支える技術参照）



<http://localhost:1000/index.html>





# 簡易Webサーバを作ろう(1)



```
// ライブラリを読み込む
import processing.net.*;
// サーバを1000ポートでスタート
Server myServer = new Server( this, 1000 );

void draw() {
  Client nextClient = myServer.available();
  if ( nextClient != null ) {
    println( nextClient.readString() );
    nextClient.write( "HTTP/1.1 200 OK\r\n" );
    nextClient.write( "Server: myServer\r\n" );
    nextClient.write( "Content-type: text/html\r\n" );
    nextClient.write( "\r\n" );
    nextClient.write( "Hello World" );
    myServer.disconnect( nextClient );
  }
}

void stop() {
  myServer.stop();
}
```

writeでHTMLのヘッダと  
Hello Worldを返す！

最後にdisconnectで切断

# 簡易Webサーバを作ろう(2)



```
// ライブラリを読み込む
import processing.net.*;
// サーバを2222ポートでスタート
Server myServer = new Server( this, 1000 );

void draw() {
  Client nextClient = myServer.available();
  if ( nextClient != null ) {
    println( nextClient.readString() );
    nextClient.write( "HTTP/1.1 200 OK\r\n" );
    nextClient.write( "Server: myServer\r\n" );
    nextClient.write( "Content-type: text/html\r\n" );
    nextClient.write( "\r\n" );
    nextClient.write( "<html>" );
    nextClient.write( "Hello World<br>" );
    nextClient.write( "<a href='http://www.fms-meiji.jp/'>FMS</a>" );
    nextClient.write( "</html>" );
    myServer.disconnect( nextClient );
  }
}

void stop() {
  myServer.stop();
}
```

writeでHTMLのヘッダと  
HTMLのコンテンツを返す！

最後にdisconnectで切断

# 簡易Webサーバを作ってみよう

明治大学総合数理学部  
先端メディアサイエンス学科  
中村研究室

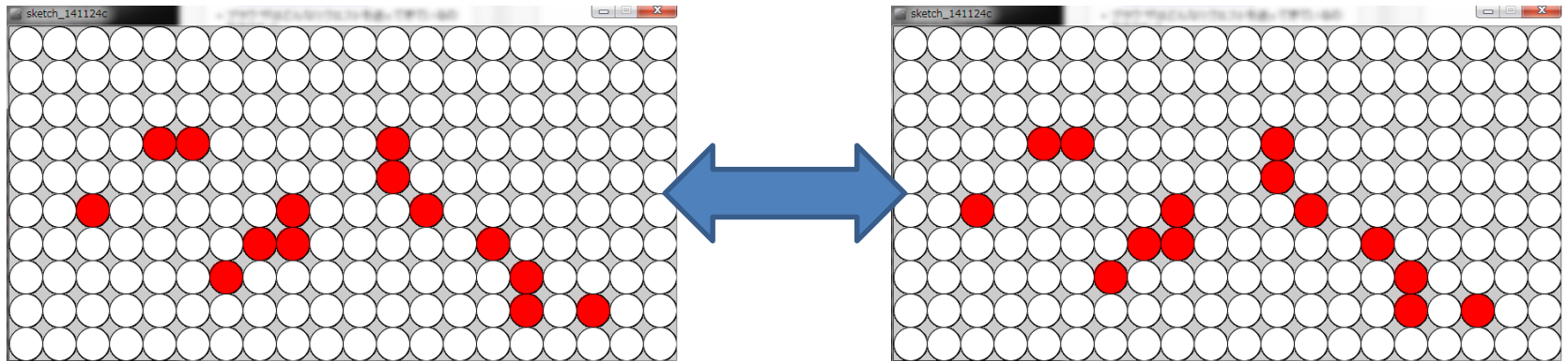


- 簡易Webサーバへのアクセス方法
  - ブラウザのURLに <http://localhost:1000/> と入力してみよう！
  - 隣の人サーバにアクセスしてみよう（localhostをIPに変更するだけ）
- リクエストに応じて処理を変えてみよう
  - <http://localhost:1000/hello.html>
  - <http://localhost:1000/fms.html>
- ブラウザはどんなリクエストを送ってきているのだろうか？

# 繋がる電光掲示板を作ろう



- 20x10の電光掲示板を作成し，サーバまたはクライアントの操作で，サーバとクライアントが同期して色が変わるようにするにはどうするか？



座標データを送り合えば良い？

# 陣取りゲームを作ってみよう

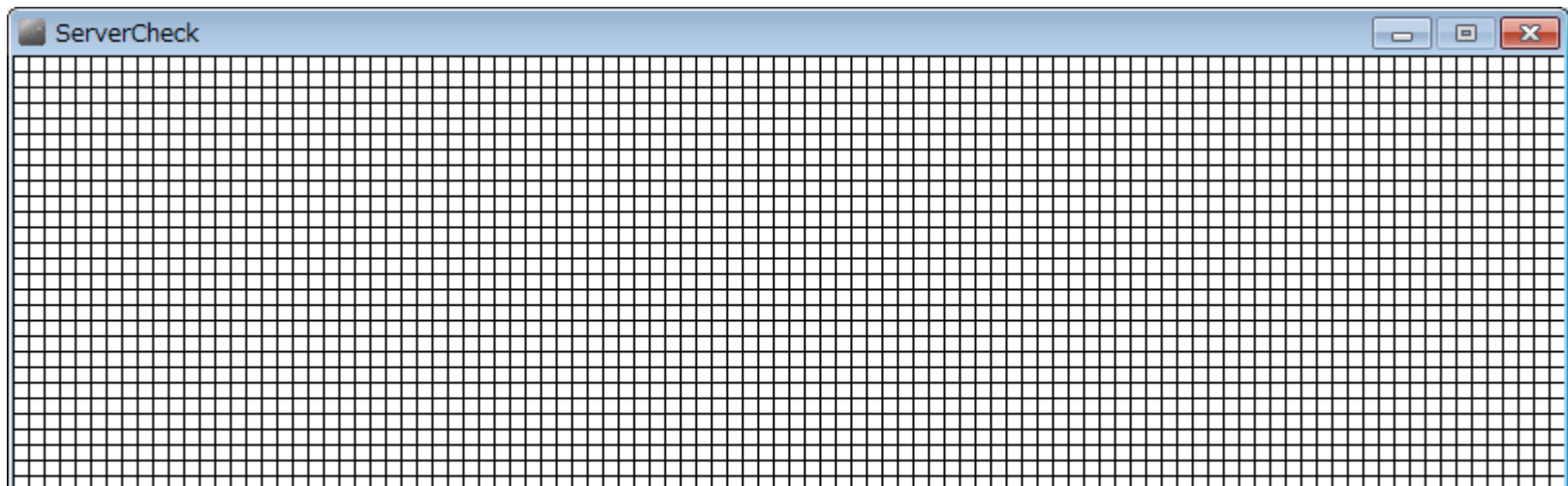
明治大学総合数理学部  
先端メディアサイエンス学科  
中村研究室



- 100x100のマス目のオセロのような陣取りゲーム
- クライアント上には100x100のマス目を表示
- マウスクリックした場所を, サーバ(教卓のIPアドレス)に接続し, サーバに対して自身の組内番号と, マス目のx, y座標(それぞれ0~99)を送信する
- ポート番号 6666, 送信フォーマット "5,52,25" のようにカンマ区切りとする

番号

座標





```
import processing.net.*;
Client myClient = new Client( this, ,  );
int unitSize = 8;
int ux = 100;
int uy = 100;
int [][] unit;
int myID =  ;

void setup() {
  size( 800, 800 );
  stroke( 0 );
  fill( 255 );
}

void draw() {
  for ( int x=0; x<ux; x++ ) {
    for ( int y=0; y<uy; y++ ) {
      rect( x*unitSize, y*unitSize, unitSize, unitSize );
    }
  }
}

void mousePressed() {
  int x = mouseX / unitSize;
  int y = mouseY / unitSize;
  myClient.write(  );
}
```

## 100x100のクライアント



- なんでも良いので、サーバクライアントで動作する仕組みを作ってみましょう
  - 提出期限は12月1日の講義の最初の5分まで
  - プログラム名は ServerApp と ClientApp とし、組番号のフォルダにまとめ、宿題フォルダに提出
  - 2人1組でServerとClientを担当してもよい(ただし、プログラム中に、何組何番のServer/Clientと対応している書く事)
- (例)
  - ピンポンゲーム(サーバとクライアントで跳ね返す)
  - 占い(サーバが占い結果を送信する)
  - ランキングシステム(クリック数を集約する)
  - 落書きシステム(授業で扱ったのとは違うものを)
  - などなど