

第8回

# JavaScriptから始める プログラミング2016

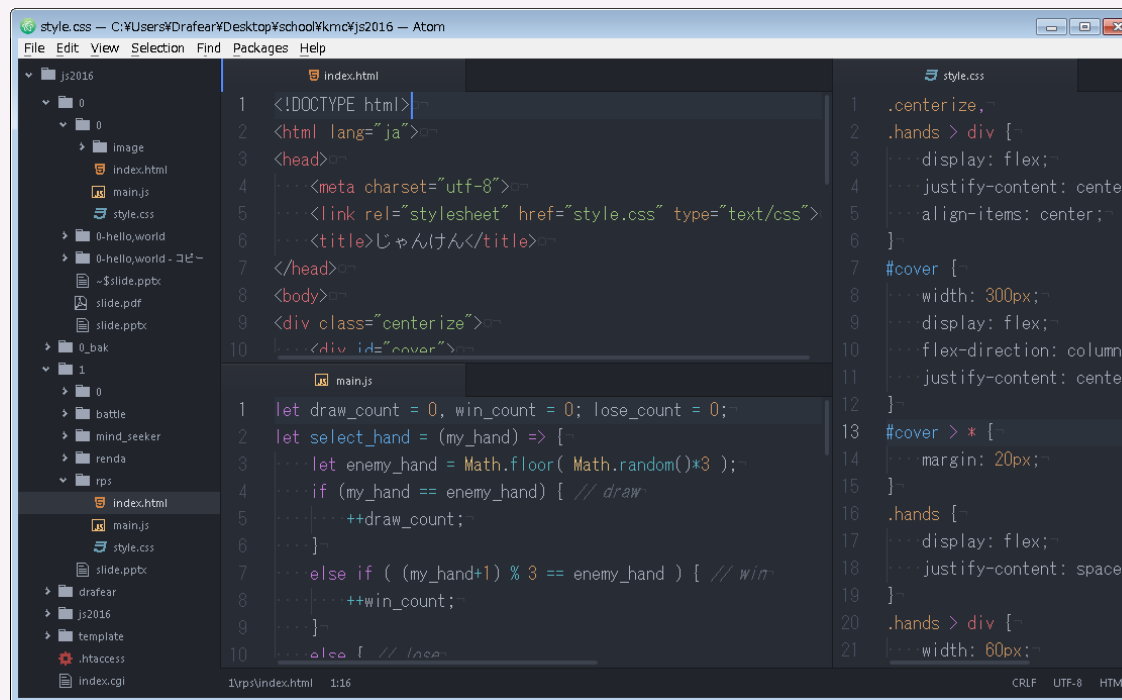
京都大学工学部情報学科  
計算機科学コース3回  
KMC2回 drafear



@drafear

# この講座で使用するブラウザとエディタ

- Google Chrome 
  - <https://chrome.google.com>
- Atom 
  - <https://atom.io/>



The screenshot shows the Atom text editor interface. On the left is a file explorer showing a project structure with folders like '0', '1', 'battle', 'mind\_seeker', 'renda', 'rps', and files like 'index.html', 'main.js', 'style.css', and 'slide.pptx'. The main editor area is split into two panes. The left pane shows the 'index.html' file with the following code:

```
1 <!DOCTYPE html>
2 <html lang="ja">
3 <head>
4   <meta charset="utf-8">
5   <link rel="stylesheet" href="style.css" type="text/css">
6   <title>じゃんけん</title>
7 </head>
8 <body>
9   <div class="centerize">
10    <div id="cover">
```

The right pane shows the 'style.css' file with the following code:

```
1 .centerize,
2 .hands > div {
3   display: flex;
4   justify-content: center;
5   align-items: center;
6 }
7 #cover {
8   width: 300px;
9   display: flex;
10  flex-direction: column;
11  justify-content: center;
12 }
13 #cover > * {
14   margin: 20px;
15 }
16 .hands {
17   display: flex;
18   justify-content: space-around;
19 }
20 .hands > div {
21   width: 60px;
```

# 本日の内容

- 環境構築 + 復習 (60分)
- UNIXコマンドの基礎 (30分)
- オンラインゲームの仕組みとネットワークの基礎 (10分)
- socket.ioを用いたリアルタイム通信 (40分)
- 自由コーディングタイム (40分)
- (できていれば)公開！ (次回)



# 環境構築

# 開発環境

node.js を用いてサーバサイドを開発する

1. ターミナルmsys2を導入
2. msys2の初期設定 (update-core, pacman -Su)
3. 必要最低限の機能の導入 (pacman -S git wget sed ...)
4. node.jsの導入 (windows以外の方はここから ?)
5. pathを通す
6. 確認
7. 日本語の文字化け対策

# 環境構築

ダウンロードやインストールには時間がかかるので、  
その間に次章の復習問題をしましょう！

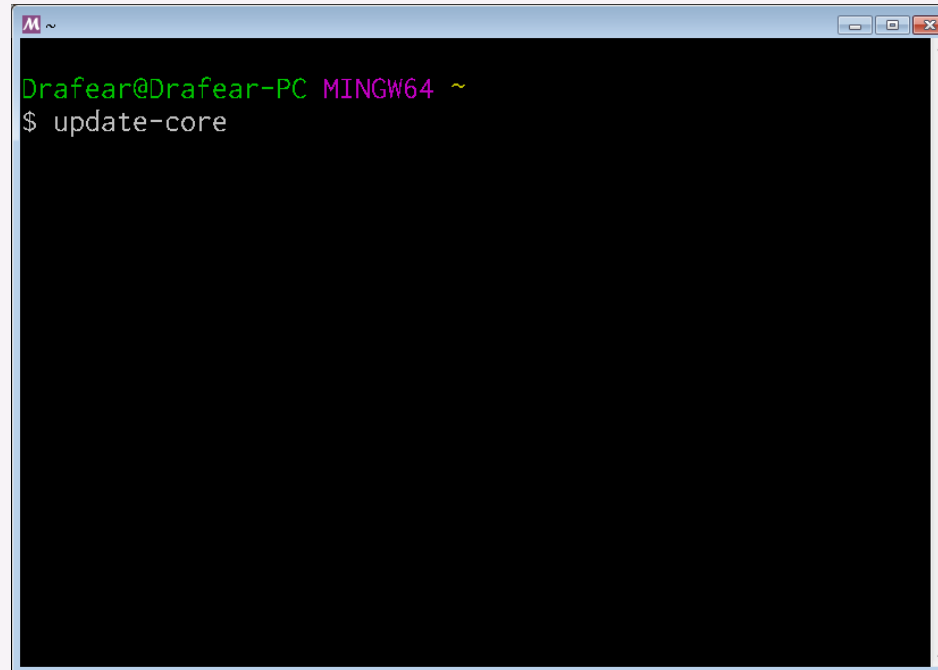
# 環境構築 (windowsの場合)

1. ターミナルmsys2を導入
  - <http://msys2.github.io/>
  - 32bit) i686
  - 64bit) x86\_64

# 環境構築 (windowsの場合)

## 2. msys2の初期設定

- i. msys2ターミナル(端末)を起動
  - すべてのプログラムでmsys2などで検索し, 「MSYS2 Shell」 を起動
- ii. 端末に `update-core` と入力してEnterキーを押し  
msys2を最新バージョンに更新



```
Drafeear@Drafeear-PC MINGW64 ~  
$ update-core
```



# 環境構築 (windowsの場合)

## 2. msys2の初期設定

- iii. 更新が完了したら[端末を再起動](#)
- iv. `pacman -Su` を実行し, パッケージリストを更新
- v. 更新が完了したら端末を再起動
  - 更新後に端末が起動できない場合は, ショートカットのリンク先を確認  
64bitの場合 `"C:¥msys64¥msys2_shell.cmd -mingw64"`  
32bitの場合 `"C:¥msys64¥msys2_shell.cmd -mingw32"`

# 環境構築 (windowsの場合)

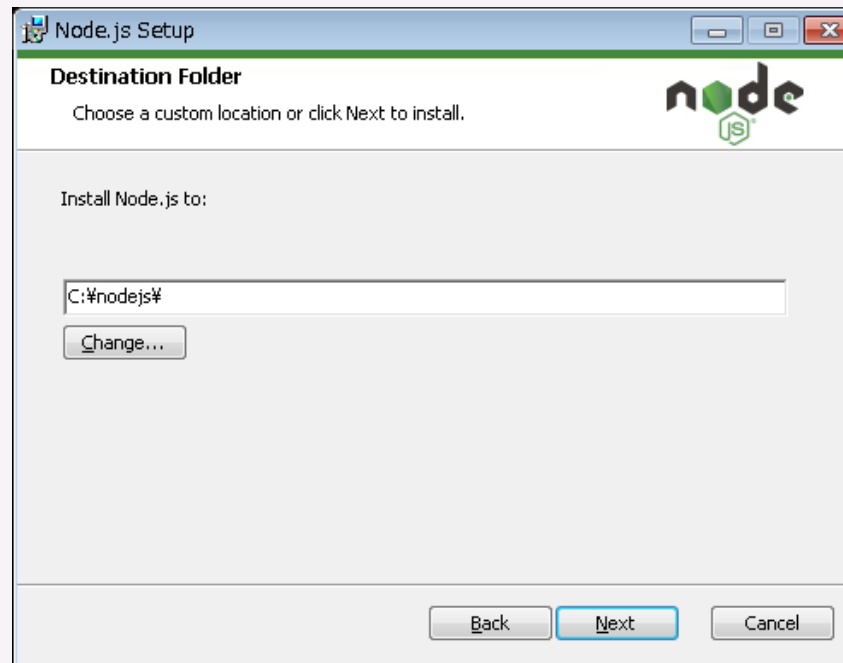
## 3. 必要最低限の機能の導入

- 端末で以下を実行して下さい
- \$ は元から入っているので入力する必要はありません
- \$ ~~ を実行することを ~~**コマンド**を実行するといひます

```
$ pacman -S git wget sed diffutils grep nano tree
```

# 環境構築 (windowsの場合)

4. node.jsの導入 (pacman -S ...)
- <https://nodejs.org/>
  - **v6.2.2 Current** の方をダウンロード
  - **Cドライブ直下**にインストールして下さい  
インストールパス: **C:%nodejs%**



# 環境構築 (windowsの場合)

## 5. pathを通す

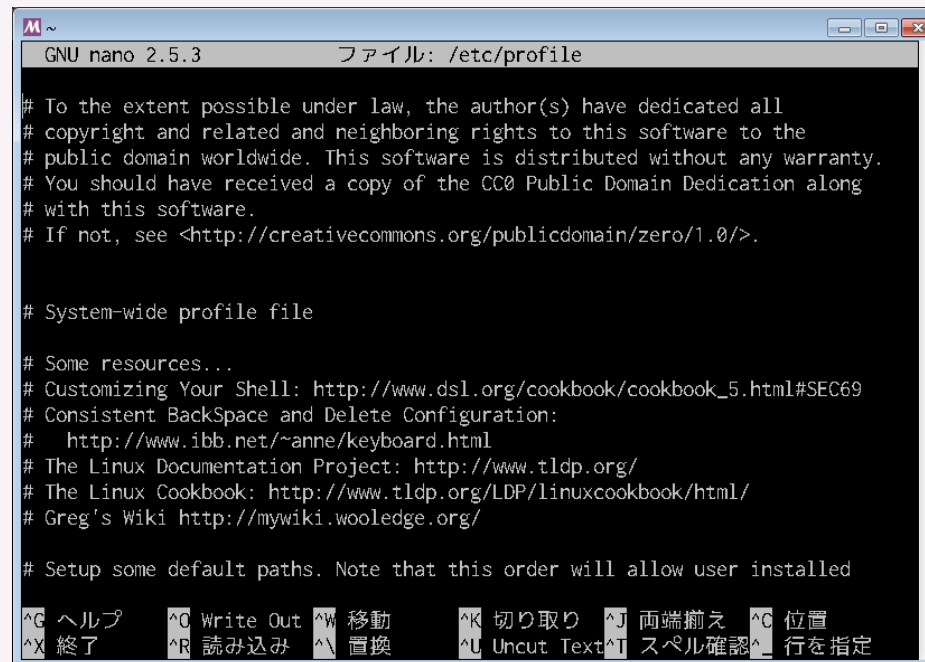
- 端末で次のコマンドを実行して下さい
- エラーが出る場合は, タイプミスしていないか確認してください
- それでもエラーがでる場合は 手順3 をもう一度行って下さい

```
$ nano /etc/profile
```

# 環境構築 (windowsの場合)

## 5. pathを通す

- nanoエディタが立ち上がります
- profile というファイルを編集しようとしています
- 画面に文字が表示されていない場合は打ち間違えた可能性があるので、CTRL + X を押してエディタを終了してやり直してください



```
GNU nano 2.5.3      ファイル: /etc/profile

# To the extent possible under law, the author(s) have dedicated all
# copyright and related and neighboring rights to this software to the
# public domain worldwide. This software is distributed without any warranty.
# You should have received a copy of the CC0 Public Domain Dedication along
# with this software.
# If not, see <http://creativecommons.org/publicdomain/zero/1.0/>.

# System-wide profile file

# Some resources...
# Customizing Your Shell: http://www.dsl.org/cookbook/cookbook_5.html#SEC69
# Consistent BackSpace and Delete Configuration:
#   http://www.ibb.net/~anne/keyboard.html
# The Linux Documentation Project: http://www.tldp.org/
# The Linux Cookbook: http://www.tldp.org/LDP/linuxcookbook/html/
# Greg's Wiki http://mywiki.woledge.org/

# Setup some default paths. Note that this order will allow user installed
```

^G ヘルプ ^O Write Out ^W 移動 ^K 切り取り ^J 両端揃え ^C 位置  
^X 終了 ^R 読み込み ^\ 置換 ^U Uncut Text ^T スペル確認 ^\_ 行を指定

# 環境構築 (windowsの場合)

## 5. pathを通す

- 少し下にスクロールすると次の記述があるので  
以下のように変更して下さい ( :/c/nodejs を追記, **コロン忘れずに** )

profile

```
MSYS2_PATH="/usr/local/bin:/usr/bin"  
MANPATH="/usr/local/man:/usr/share/man:/usr/man:/share/man:${MANPATH}"  
INFOPATH="/usr/local/info:/usr/share/info:/usr/info:/share/info:${INFOPATH}"
```



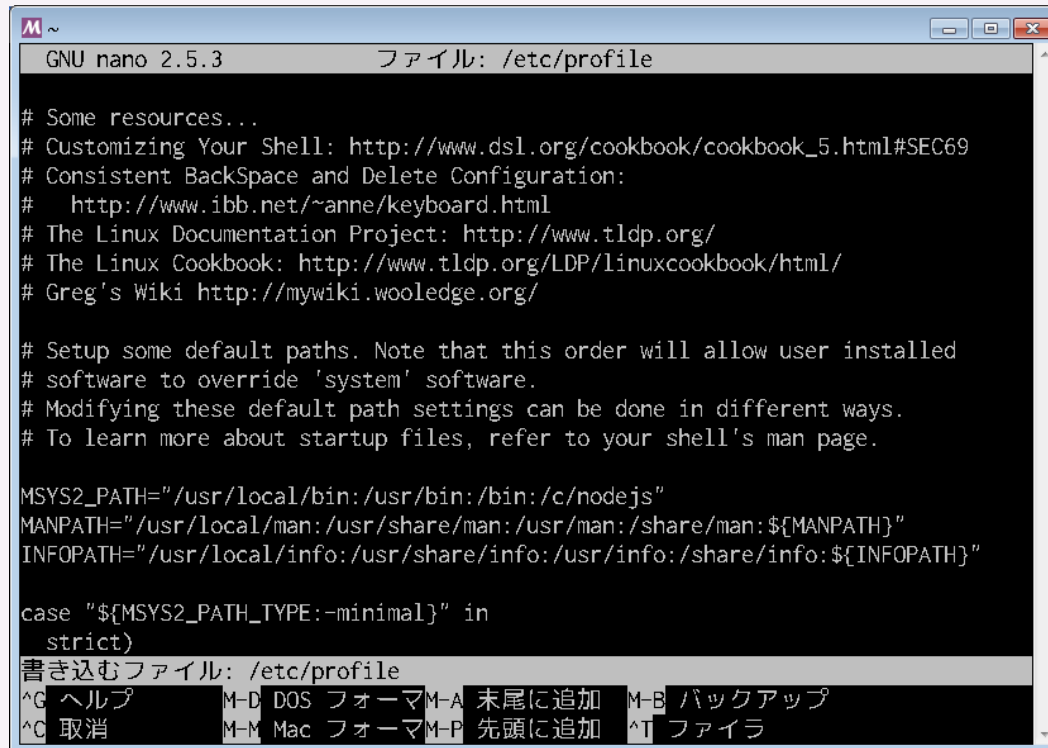
profile

```
MSYS2_PATH="/usr/local/bin:/usr/bin:/bin:/c/nodejs"  
MANPATH="/usr/local/man:/usr/share/man:/usr/man:/share/man:${MANPATH}"  
INFOPATH="/usr/local/info:/usr/share/info:/usr/info:/share/info:${INFOPATH}"
```

# 環境構築 (windowsの場合)

## 5. pathを通す

- CTRL + O の後 Enter で保存
- CTRL + X で終了
- 端末を再起動



```
GNU nano 2.5.3      ファイル: /etc/profile

# Some resources...
# Customizing Your Shell: http://www.dsl.org/cookbook/cookbook_5.html#SEC69
# Consistent BackSpace and Delete Configuration:
#   http://www.ibb.net/~anne/keyboard.html
# The Linux Documentation Project: http://www.tldp.org/
# The Linux Cookbook: http://www.tldp.org/LDP/linuxcookbook/html/
# Greg's Wiki http://mywiki.woledge.org/

# Setup some default paths. Note that this order will allow user installed
# software to override 'system' software.
# Modifying these default path settings can be done in different ways.
# To learn more about startup files, refer to your shell's man page.

MSYS2_PATH="/usr/local/bin:/usr/bin:/bin:/c/nodejs"
MANPATH="/usr/local/man:/usr/share/man:/usr/man:/share/man:${MANPATH}"
INFOPATH="/usr/local/info:/usr/share/info:/usr/info:/share/info:${INFOPATH}"

case "${MSYS2_PATH_TYPE:-minimal}" in
    strict)
書き込むファイル: /etc/profile
^G ヘルプ      M-D DOS フォーマット M-A 末尾に追加 M-B バックアップ
^C 取消      M-M Mac フォーマット M-P 先頭に追加 ^T ファイル
```

# 環境構築 (windowsの場合)

## 6. 確認

- 次のコマンドを実行して結果(バージョン)が表示されるか確認して下さい
- \$ に続く文字列がコマンドで, その下は結果(例)です

```
$ node -v  
v6.2.2
```

```
$ npm -v  
3.9.5
```



# 環境構築 (windowsの場合)

## 7. 日本語の文字化け対策

- 次のコマンドを実行し, .bashrc ファイルを編集します
- .bashrc ファイルには, 端末起動時に自動的に実行されるコマンド列を記述します

```
$ nano ~/.bashrc
```

# 環境構築 (windowsの場合)

## 7. 日本語の文字化け対策

- i. # To the extent ... の前行を挿入して以下を追記  
(中クリックで端末上に貼り付けできます)
- ii. CTRL + O の後 Enter で保存
- iii. CTRL + X でエディタを終了
- iv. 再起動

```
function wincmd() {  
    CMD=$1  
    shift  
    $CMD $* 2>&1 | iconv -f cp932 -t utf-8  
}
```

# 環境構築 (windowsの場合)

## 7. 日本語の文字化け対策

- 以下のコマンドを実行して  
正しく日本語が表示されるか確認してください
- エラーが出る場合は前スライドの手順を再確認して下さい

```
$ wincmd ipconfig
```

# 環境構築 (windowsの場合)

お疲れ様でした！！！！

環境構築は終了です！



# 復習問題

# 演習

- 1番多く解いた人にビックル1/4日分贈呈
- いらなかったら2番目の人



# 演習



# UNIXコマンドの基礎



# 用語

ディレクトリ (復習)	ファイルを格納するもの フォルダ
カレントディレクトリ (ワーキングディレクトリ)	今居るディレクトリ 作業ディレクトリ 今どこに居るかという概念がある 相対パスはここを基準とする
ホームディレクトリ	ユーザごとに作られるディレクトリ 端末起動時のカレントディレクトリとなる場所 ~ で表される
ルートディレクトリ	最上位のディレクトリ これの親ディレクトリは存在しない / で表される

# コマンドの実行方法

- コマンド名を書いてEnterキーを押す
- 引数を渡す場合は, コマンド名の直後に半角スペースを設けて  
その後に半角スペース区切りで引数を並べる

```
$ コマンド名
```

```
$ コマンド名 引数1 引数2 ...
```

# コマンド

- cd (change directory)

cd ディレクトリへのパス

指定したディレクトリに移動する  
つまり、カレントディレクトリを指定したパスに変更する

## 例

```
$ # シャープの後はコメント
$ cd hoge      # hogeディレクトリに移動
$ cd ..        # 親ディレクトリに移動
$ cd "C:¥Users¥Drafeare¥Desktop"  # デスクトップに移動
$ cd ~         # ホームディレクトリに移動
$ cd /         # ルートディレクトリに移動
$ cd           # ホームディレクトリに移動
```

# コマンド

- pwd (print working directory)

pwd

カレントディレクトリの絶対パスを標準出力に出力する

例

```
$ pwd  
/c/Users/Drafear/Desktop
```

# 標準入力・標準出力

- 標準出力とは？
  - コマンドは引数と標準入力から入力を受け取り, 標準出力に出力する



# 標準入力・標準出力

- 引数はコマンド入力時に指定する



# 標準入力・標準出力

- 標準入力はデフォルトではキーボードに割り当てられている
- 標準出力はデフォルトでは端末の画面に割り当てられている
- (実は標準エラー出力というものもあるが今回は割愛)



# リダイレクション

- リダイレクション機能により  
標準入力元や出力先を切り替えられる
  - 「> file」で標準出力を fileファイル に割り当て
  - ファイルに上書き. ファイルがなければ新規作成.
  - 「< file」で標準入力を fileファイル に割り当て
  - (追加書き込み ">>" や 複数行読み込み "<<" もある)
- ※ リダイレクションは引数ではない



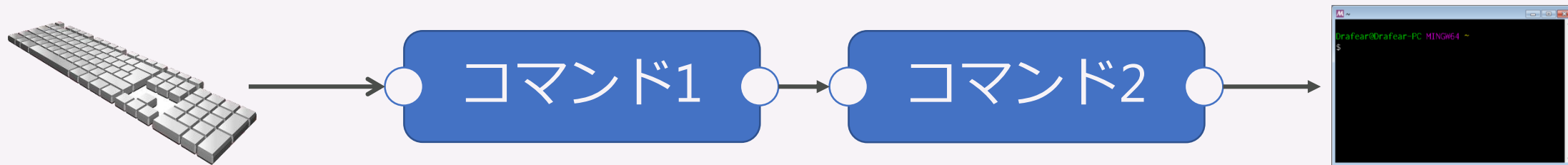
例

```
$ pwd > out.txt    # out.txtにカレントディレクトリの絶対パスを出力
```



# パイプ

- パイプ機能により, あるコマンドの標準出力を別のコマンドの標準入力に割り当てられる
- 詳しくは後述



# コマンド

- コマンド紹介の続き
- echo

echo 引数1 引数2 ...

引数を半角スペースで連結して標準出力に出力  
最後に改行される

## 例

```
$ echo hello
```

```
hello
```

```
$ echo "www"
```

```
www
```

```
$ echo xxx > o # oファイルに"xxx"を書き込み
```

```
$ echo dra > o2 fe      ar # "dra fe ar" とo2ファイルに書き込み
```

# コマンド

- cat (concatenate)

cat ファイル1 ファイル2 ...	ファイルの内容を連結して標準出力に出力 特に, ファイルを1つだけ指定した場合, ファイルの内容を出力
cat	標準入力をそのまま標準出力に出力

## 例

```
$ cat # キーボードから入力した内容がそのまま表示される  
[戻るには CTRL+D]  
$ cat o o2  
xxx  
dra fe ar  
$ cat < o  
xxx
```

# コマンド

- ls (list)

ls	カレントディレクトリのファイルやディレクトリ一覧を標準出力に出力
ls -l	ファイルやディレクトリの情報を詳細に表示
ls -a	隠しファイルも表示

## 例

```
$ ls
o o2
$ ls -l
合計 2.0K
-rw-r--r-- 1 Drafeear None  4 6月  22 21:10 o
-rw-r--r-- 1 Drafeear None 10 6月  22 21:10 o2
$ ls -a
$ ls -a
. .. .bash_history .bash_logout .bash_profile .bashrc .config .inputrc .minttyrc .profile o o2
```

# オプション

- ls (list)
  - "-hoge" をオプションという
  - オプションを書く順番は大抵の場合は決まっていない
  - "ls --help" とすると, オプション一覧が表示される
  - 他にも, "cat --help" とすると, catコマンドの使い方などが表示される

ls	カレントディレクトリのファイルやディレクトリ一覧を標準出力に出力
ls -l	ファイルやディレクトリの情報を詳細に表示
ls -a	隠しファイルも表示

# まとめと残りの紹介

cd ディレクトリへのパス	指定したディレクトリに移動する つまり, カレントディレクトリを指定したパスに変更する
pwd	カレントディレクトリの絶対パスを標準出力に出力する
echo 引数1 引数2 ...	引数を半角スペースで連結して標準出力に出力 最後に改行される
cat ファイル1 ファイル2 ...	ファイルの内容を連結して標準出力に出力 特に, ファイルを1つだけ指定した場合, ファイルの内容を出力
cat	標準入力をそのまま標準出力に出力
ls	カレントディレクトリのファイルやディレクトリ一覧を 標準出力に出力
ls pattern	lsの結果patternにマッチするもののみ出力 pattern には * (0文字以上の任意の文字列), ? (任意の1文字) が 使える (正規表現でないことに注意) 例) ls *.txt

# まとめと残りの紹介

<code>mkdir dir1 dir2 ...</code>	dir1, dir2, ... の名前のディレクトリを作る
<code>touch file1 file2 ...</code>	file1, file2, ... の名前のファイルを作る
<code>mv fname_from fname_to</code>	fname_from ファイルを fname_to ファイルにリネーム fname_from や fname_to はパスで指定し, 親ディレクトリが違えば移動する
<code>mv file1 file2 ... fileN dir</code>	file1, file2, ..., fileN ファイルを dir ディレクトリに移動 file1, ... はディレクトリでも良い
<code>rm file1 file2 ...</code>	file1, file2, ... ファイルを削除する rオプション(-r)でディレクトリも削除できる
<code>cp from to</code>	from ファイルを to ファイルにコピー rオプションでディレクトリもコピーできる
<code>cp file1 ... fileN dir</code>	file1, file2, ..., fileN ファイルを dir ディレクトリ内にコピー
<code>cat ファイル1 ファイル2 ...</code>	ファイルの内容を連結して標準出力に出力 特に, ファイルを1つだけ指定した場合, ファイルの内容を出力
<code>cat</code>	標準入力をそのまま標準出力に出力

# まとめと残りの紹介

コマンド > file	コマンドの標準出力をfileファイルに切り替える
コマンド < file	コマンドの標準入力をfileファイルに切り替える



# 演習

1. 以下のディレクトリ構造を作ろう (ファイルの中身は空でok)

```
$ tree
.
├── dir1
│   ├── A
│   │   ├── a
│   │   └── c
│   ├── B
│   │   ├── d
│   │   ├── e
│   │   └── f
│   ├── j
│   └── k
├── dir2
│   └── z
└── file
```

# 演習

2. file を dir1/B 内に移動しよう
3. dir1 と dir2 を削除しよう
4. デスクトップに移動しよう

# 演習

1. 以下のディレクトリ構造を作ろう (ファイルの中身は空でok)

```
$ mkdir dir1 dir2
$ cd dir1
$ mkdir A B
$ cd A
$ touch a c
$ cd ../B
$ touch d e f
$ cd ..
$ touch j k
$ cd ../dir2
$ touch z
$ cd ..
$ touch file
$ tree
```

```
$ tree
.
├── dir1
│   ├── A
│   │   ├── a
│   │   └── c
│   ├── B
│   │   ├── d
│   │   ├── e
│   │   └── f
│   ├── j
│   └── k
├── dir2
│   └── z
└── file
```

# 演習

2. file を dir1/B 内に移動しよう

```
$ mv file dir1/B  
$ tree
```

2. dir1 と dir2 を削除しよう

```
$ rm -r dir1 dir2  
$ ls
```

3. デスクトップに移動しよう

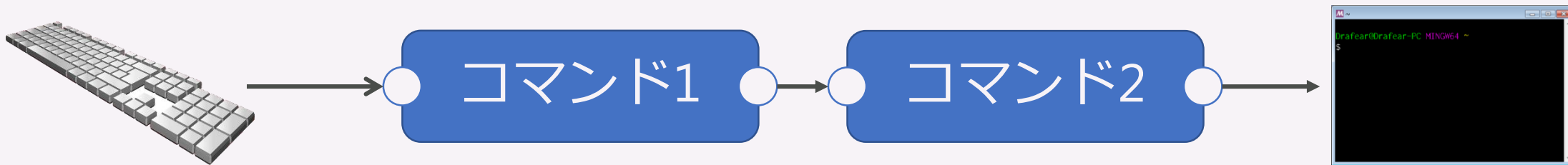
```
$ cd "C:¥Users¥Drafear¥Desktop" # 一例
```

# 他の機能

<code>file filepath</code>	filepath ファイルが何ファイルかを標準出力に出力する ディレクトリも調べられる
<code>nano file</code>	fileファイルをnanoエディタで編集
<code>sed s/正規表現/もじれつ/</code>	標準入力の中で、正規表現にマッチした文字列を もじれつに置換して標準出力に出力する
<code>grep 正規表現</code>	標準入力の中で、正規表現にマッチした行を標準出力に出力する
<code>find dir -name tgt</code>	dir以下全てのファイル、ディレクトリを検索し、tgtにマッチした ものを標準出力に出力する tgtは正規表現ではなく、* 単体で0文字以上、? で1文字を表す 例) <code>find . -name "*.exe"</code>

# パイプ

- パイプ機能により, あるコマンドの標準出力を別のコマンドの標準入力に割り当てられる
- コマンド1 | コマンド2 と記述
- コマンド1 | コマンド2 | ... | コマンドn と幾つも並べられる



例

```
$ echo "Hello" | sed s/[a-z]/o/g
Hoooo
$ ls -l | grep ^d    # ディレクトリのみ表示される
drwxr-xr-x 1 Drafeard None    0 10月 22  2015 c/
...
```

# ショートカットキー

↑ / ↓	以前使ったコマンドを見てそのまま使える
CTRL + R	以前使ったコマンドを検索できる
Tab	入力中のコマンドやファイル名を補完する
Tab Tab	補完候補を一覧表示する
CTRL + C	プログラムを強制終了させる
CTRL + D	標準入力を与え終える または, 端末からログアウトする
CTRL + L	端末をクリアする (実際にはスクロールしているだけ)
CTRL + K	行末まで文字を削除
CTRL + E	行末までカーソルを移動
CTRL + A	行頭までカーソルを移動

# まとめ

- UNIXコマンドはたくさんある
- 使えたらかっこよさそう
- 積極的に補完などの便利機能を使っていこう
- とりあえず `cd` と `ls` と `cat` だけ使えたらおk





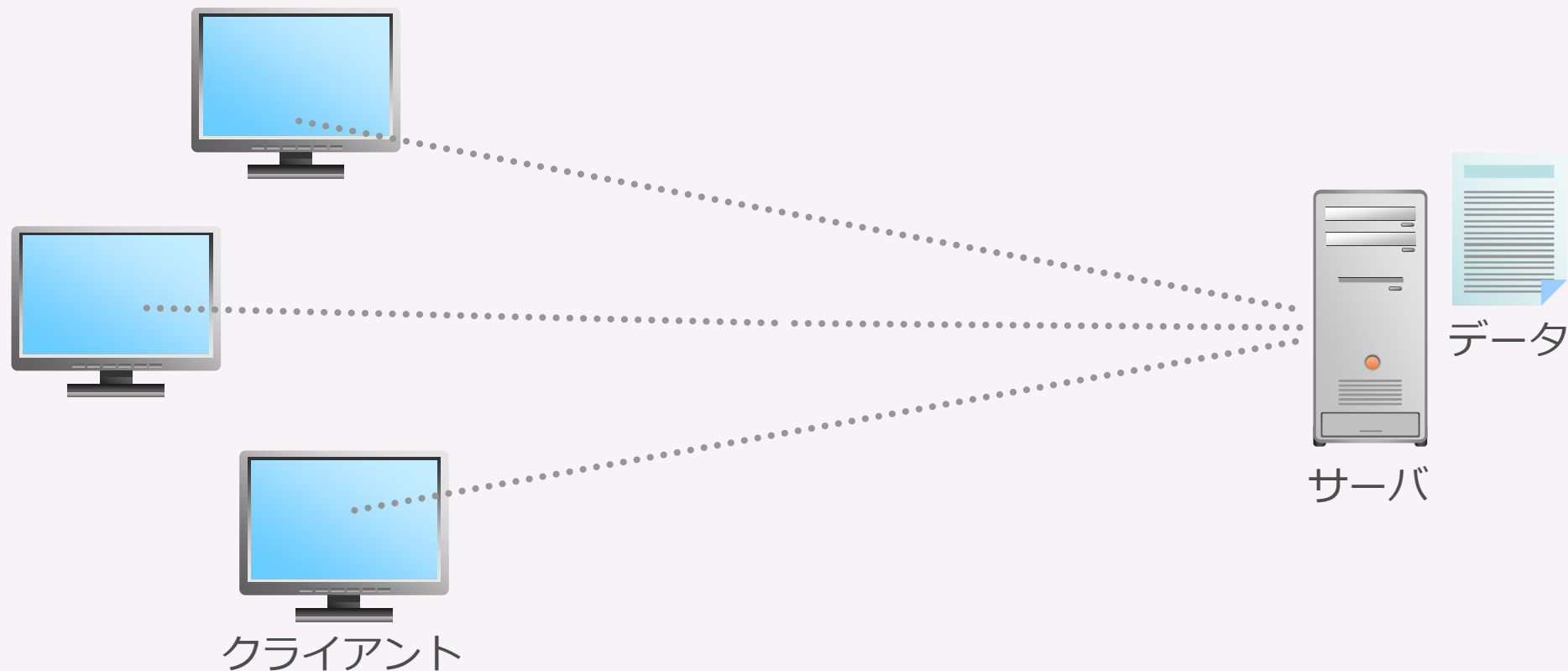
# オンラインゲームの仕組み

## と

# ネットワークの基礎

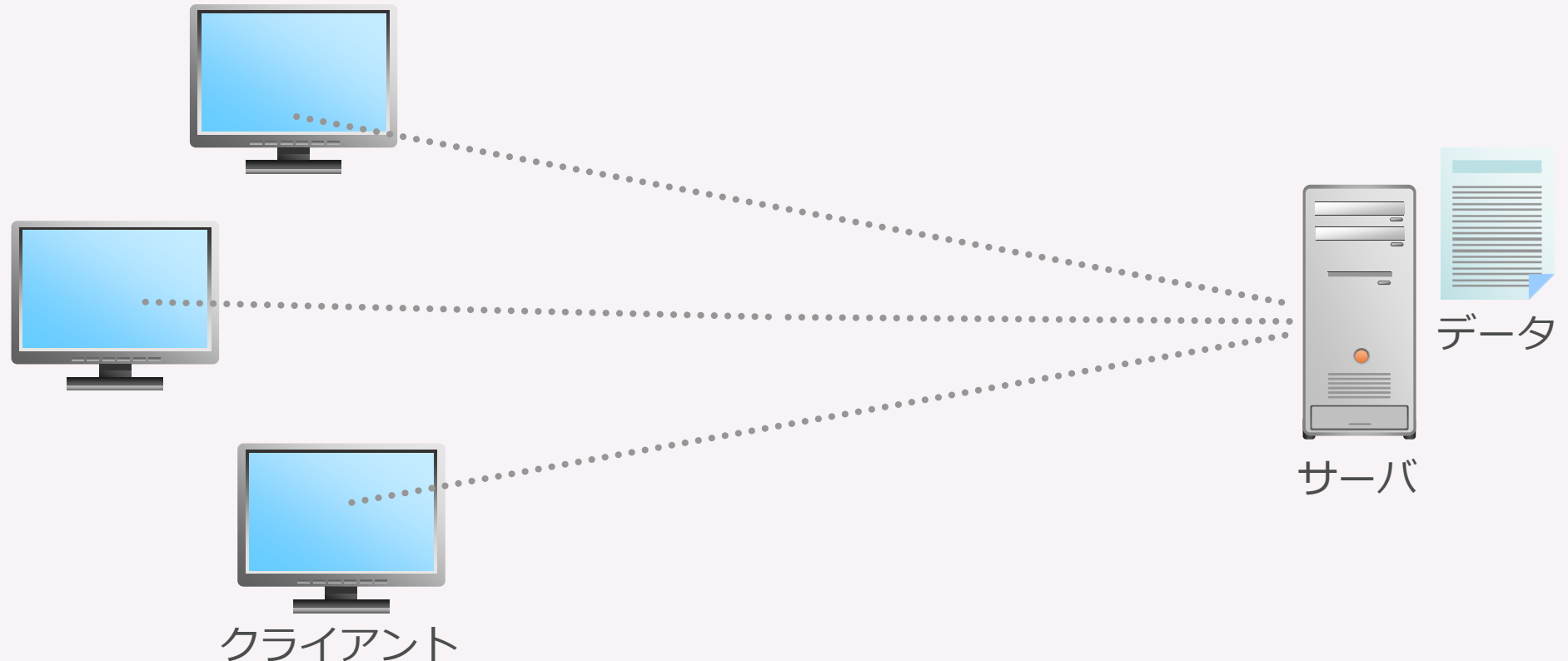
# オンラインゲームの仕組み

- クライアント間でデータを共有できれば良い



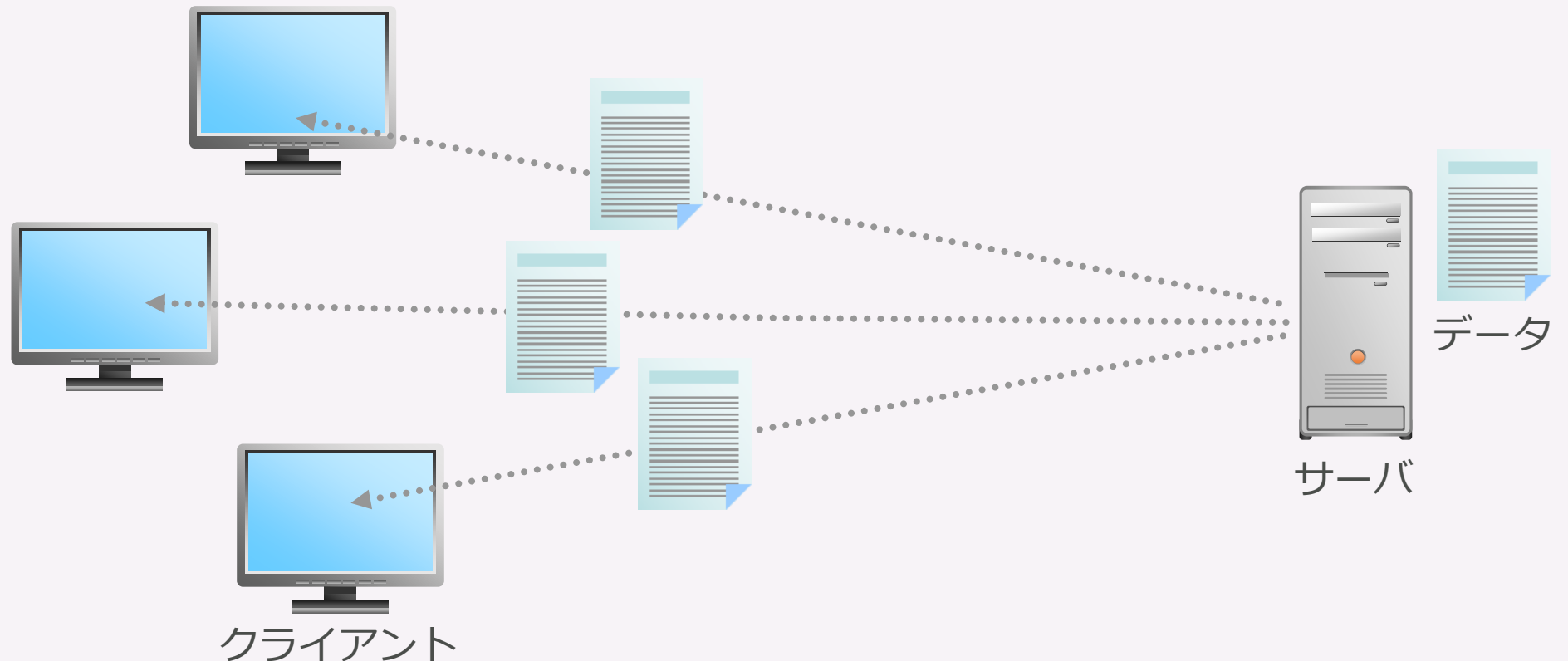
# オンラインゲームの仕組み

- クライアントはサーバとのみ接続している
- サーバにデータが保管され, 各ユーザがサーバに要求してそのデータを取得・更新する



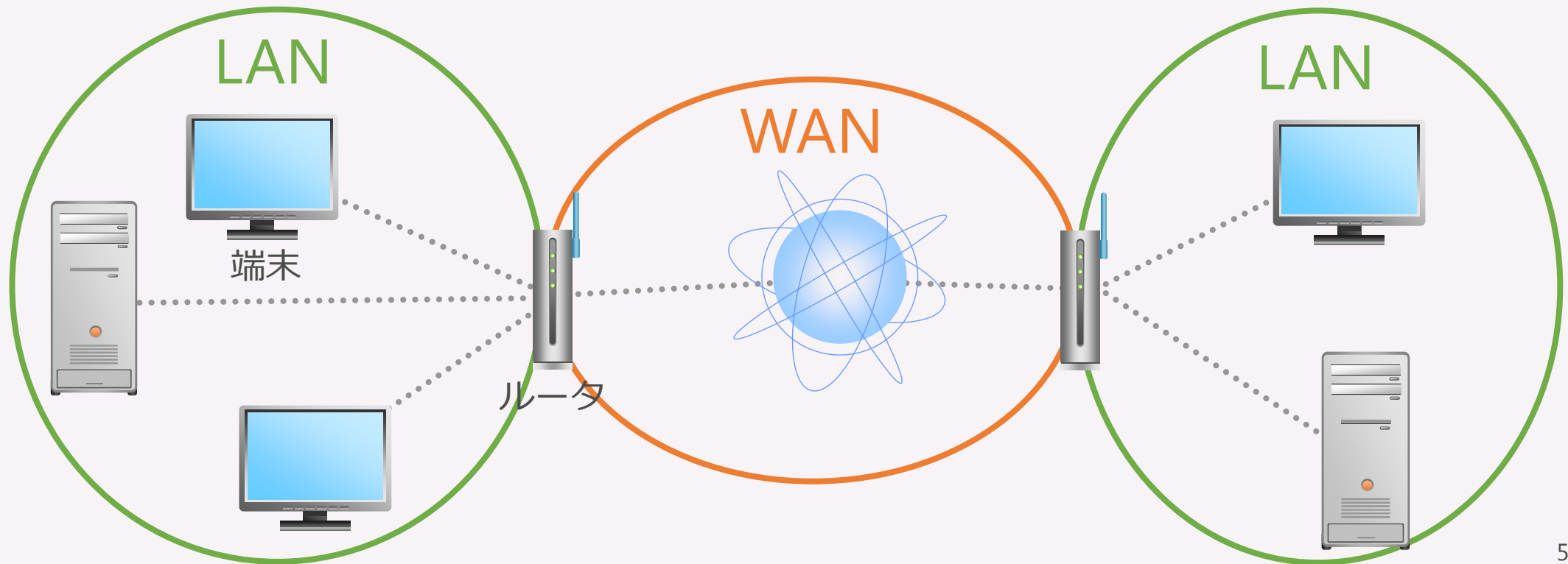
# オンラインゲームの仕組み

- または, いずれかのクライアントの要求によってデータに更新が生じたとき, サーバは全クライアントに更新情報をブロードキャストする



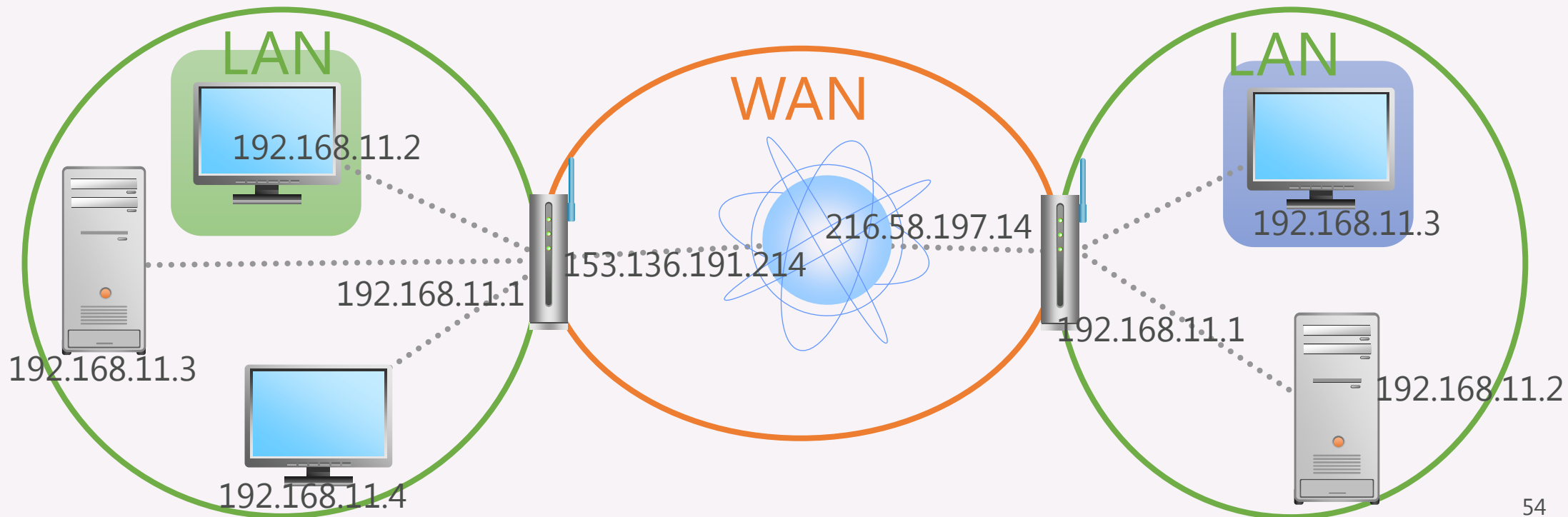
# ネットワークの基礎

- 各端末はルータを介して外の広いネットワーク(WAN)に接続している
- ルータの内側も1つの社内や自宅内の小さなネットワーク(LAN)が形成されている



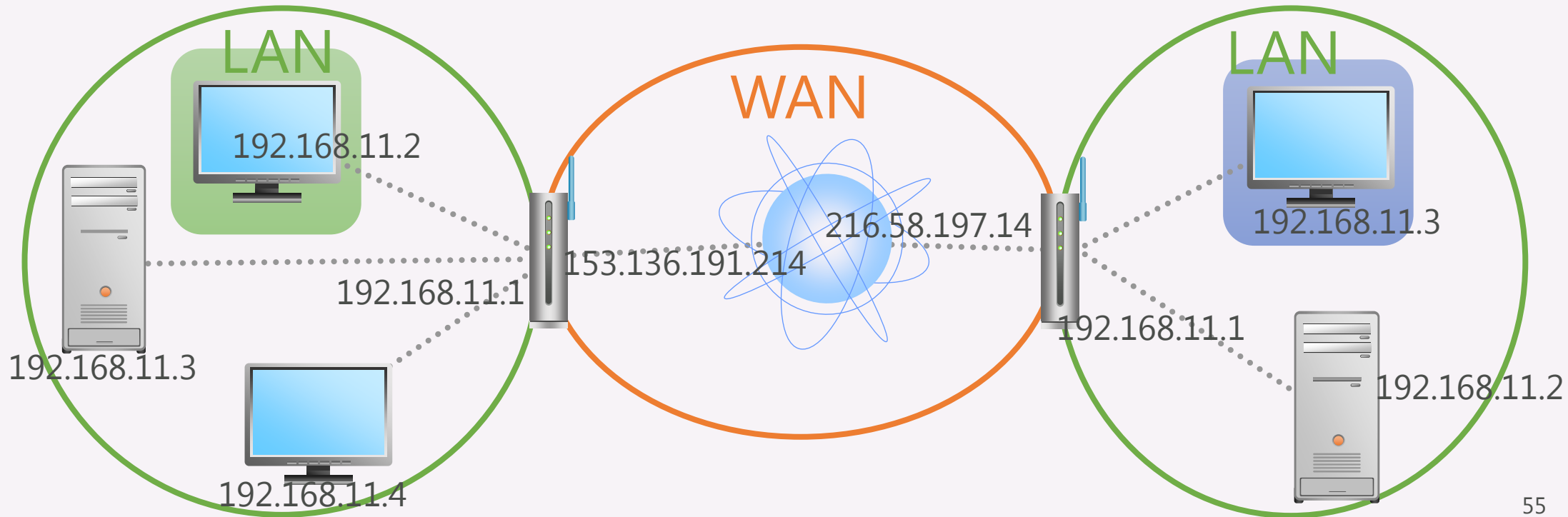
# ネットワークの基礎

- 例えば, 下の緑色の端末が青色の端末にアクセスすることを考える
- 実際には, 「ある端末にアクセスする」ではなく  
「ある端末内の特定のアプリケーションにアクセスする」である



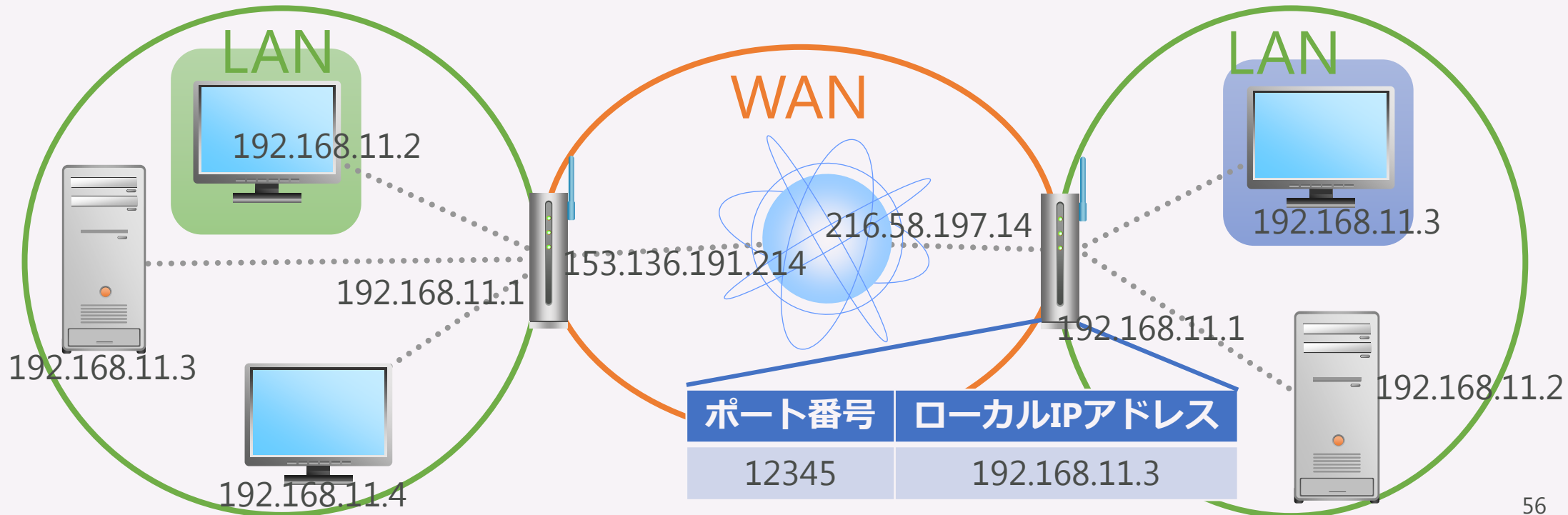
# ネットワークの基礎

- つまり, 緑の端末が青の端末のあるアプリケーションに要求を送ることを考える
- 要求には宛先のグローバルIPアドレスが書かれている  
(この場合216.58.197.14)



# ネットワークの基礎

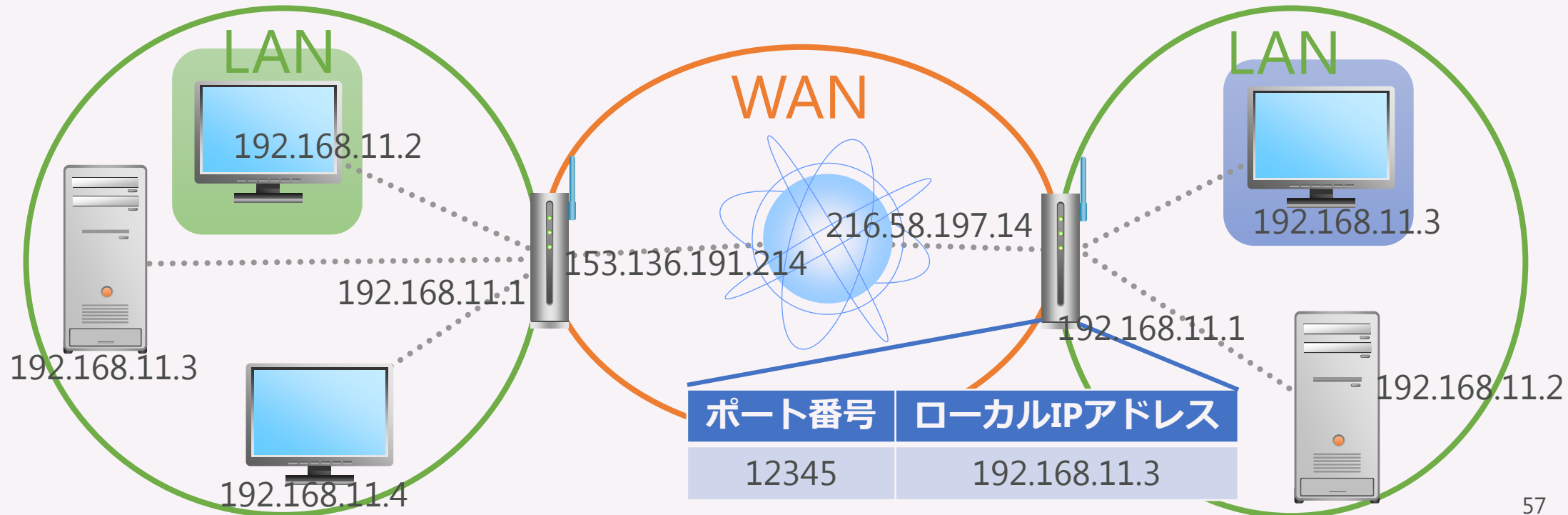
- しかし, これだけだと青の端末への要求なのかその下のサーバへの要求なのか分からない
- そこで, アプリケーションを識別するポート番号という概念がある
- 予めポート番号とローカルIPアドレスの変換表をルータに与えておく





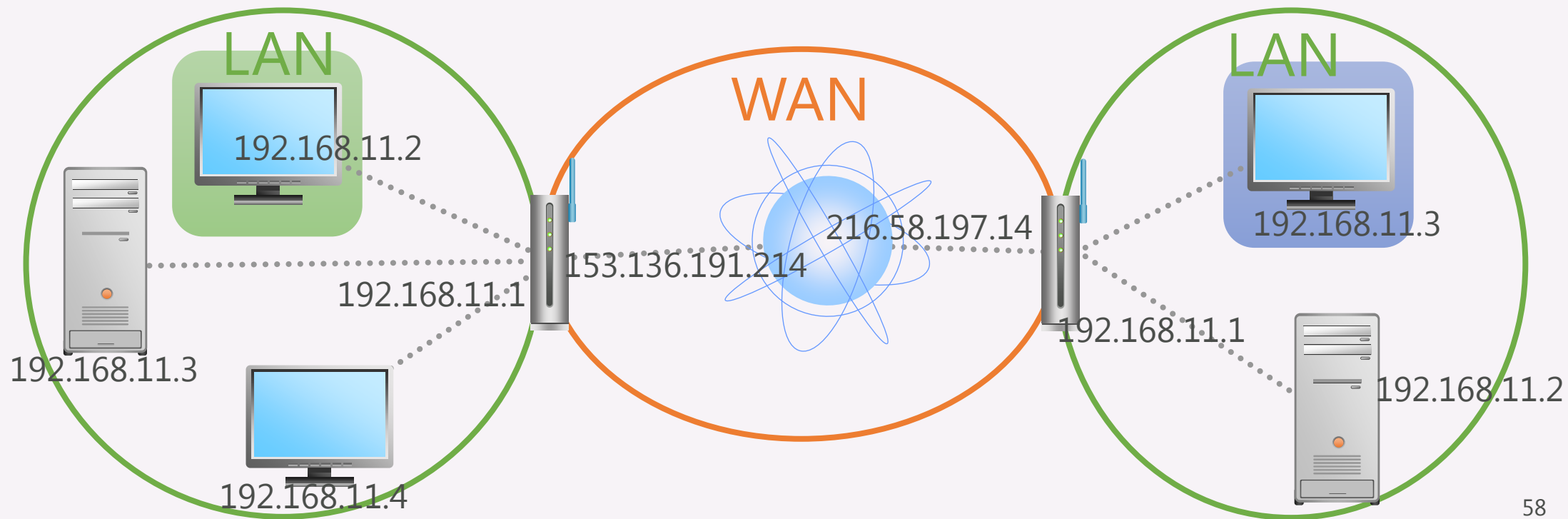
# ネットワークの基礎

- 送信側は要求にグローバルIPアドレスとポート番号を送る
- 要求を受け取ったルータは宛先IPアドレスを変換する(NAPTという)
- 要求を受け取った端末は, ポート番号に基づき特定のアプリケーションに要求を渡して解決する



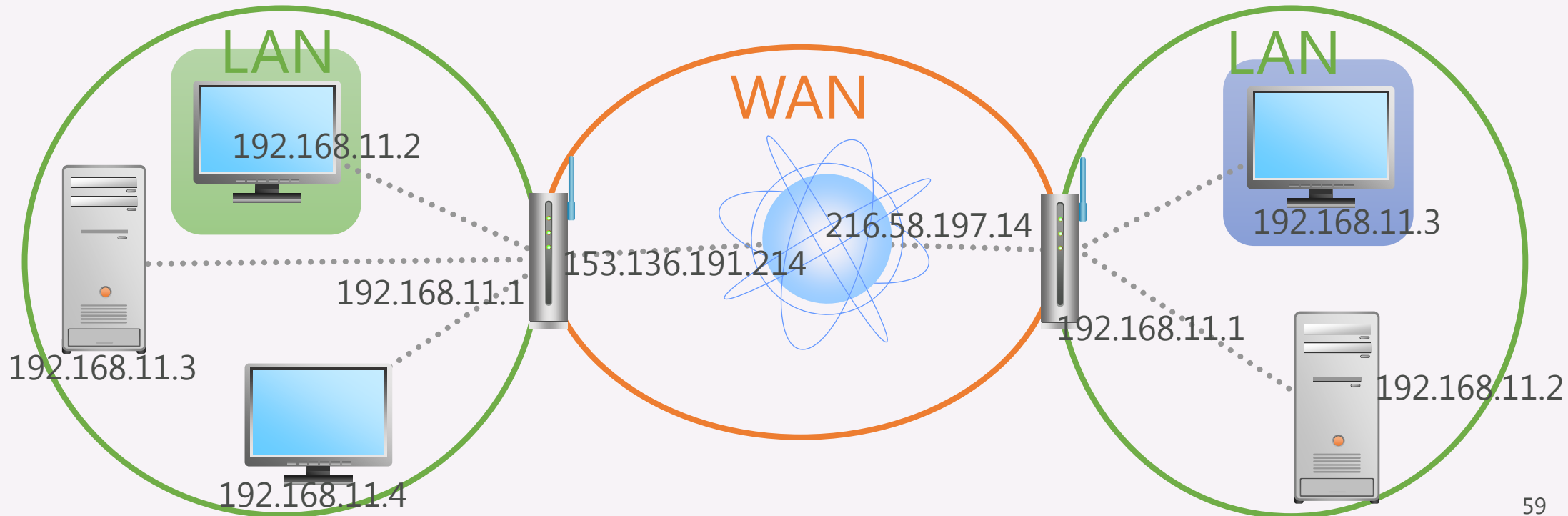
# ネットワークの基礎

これが要求を送る仕組みである



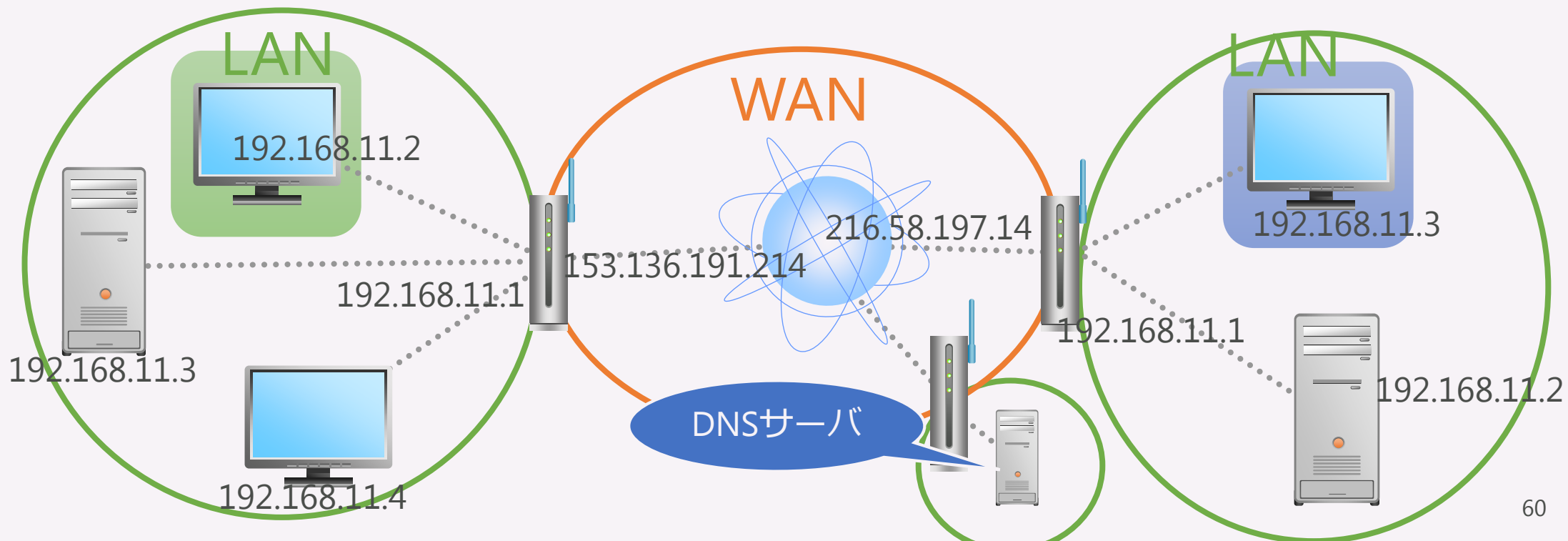
# ネットワークの基礎

- 実際には, 最初から 216.58.197.14 に送るのではなく, ドメイン(例えば google.com)で送ろうとする
- ドメインからIPアドレスへの変換を DNS(Domain Name System)サーバが行う



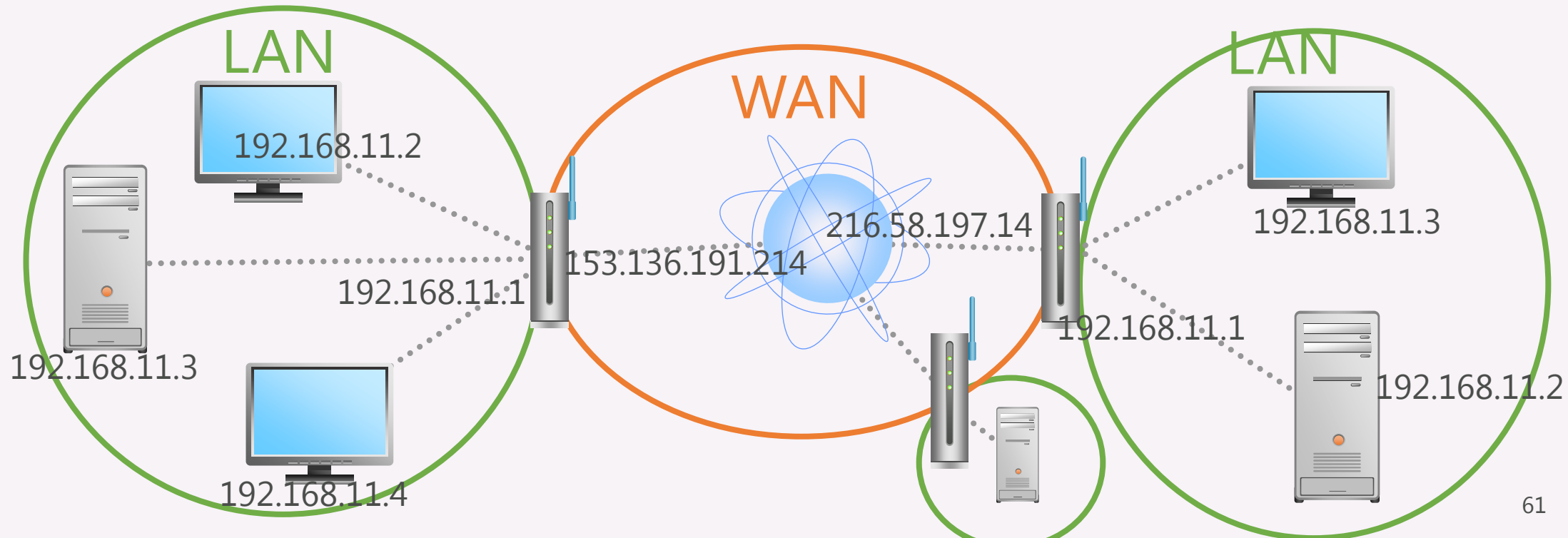
# ネットワークの基礎

- 実際には, 最初から 216.58.197.14 に送るのではなく, ドメイン(例えば google.com)で送ろうとする
- ドメインからIPアドレスへの変換を DNS(Domain Name System)サーバが行う



# ネットワークの基礎

- 例えば google.com は 216.58.197.14 (など)である
- ちなみに, `http://hoge.fuga/` にアクセスしたときのポート番号は 80 である
- 明示的に示す場合は `http://hoge.fuga:8080/` とする



# ネットワークの基礎

- 自分のプライベートIPアドレスは ipconfig で確認できる

```
$ wincmd ipconfig
...
Wireless LAN adapter ワイヤレス ネットワーク接続:
  接続固有の DNS サフィックス . . . . :
  リンクローカル IPv6 アドレス . . . . : fe80::7caf:d141:a78e:704d%14
  IPv4 アドレス . . . . . : 192.168.11.4
  サブネット マスク . . . . . : 255.255.255.0
  デフォルト ゲートウェイ . . . . . : 192.168.11.1
...
```

# ネットワークの基礎

- DNSで変換した後のIPアドレスを知りたい場合  
nslookup でできる

```
$ wincmd nslookup google.com
```

権限のない回答:

サーバー: apcce1d5b8b97a

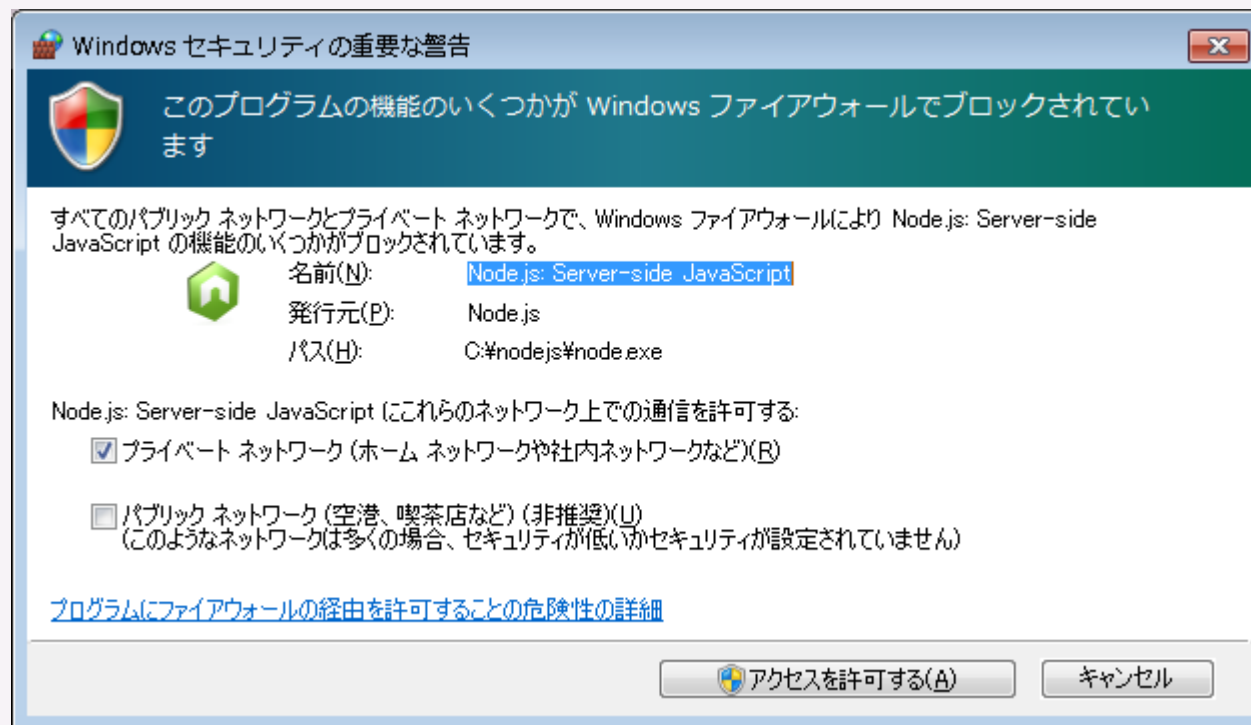
Address: 192.168.11.1

名前: google.com

Address: 216.58.221.174

# ネットワークの基礎

- ファイアウォール(防火壁)
  - 端末のすぐ外にファイアウォールがあり  
特定のポートへのアクセスの許可や拒否を行う







# Node.js 入門

socket.io を用いたリアルタイム通信

# Node.js入門

- Node.jsでできること
  - デスクトップアプリケーション (次々回?)
  - ゲームサーバ (今回)
  - Webサーバ (次回)
  - その他のサーバ

# Node.js入門

- 今回は [socket.io](#) を用いてリアルタイム通信を実現します！
- socket.io で接続や切断, 要求や応答, ブロードキャストなどが簡単にできる！！
- ではまずインストール (またかw)

```
$ cd "[siotemplateディレクトリまでのパス]"  
$ npm install socket.io --save
```

# Node.js入門

- サーバを動かしてみよう

```
$ node app.js  
Server Running!
```

# Node.js入門

- サーバの停止
  - CTRL + C を押す

```
$ node app.js  
Server is Running!
```

```
$
```

# Node.js入門

- ボタンをクリックすると全クライアントに草が生えるものを作ってみる

index.html

```
<div id="mes"></div>  
<button id="btnAlert">通知</button>
```

# Node.js入門

- ボタンをクリックすると全クライアントに草が生えるものを作ってみる

main.js (client側)

```
'use strict'
const socketio = io.connect('http://localhost:8080');
// alertメッセージを受け取ったら
socketio.on("alert", (data) => {
  // 草を生やす
  document.getElementById("mes").textContent += "w";
});
// btnAlertがクリックされたら
document.getElementById("btnAlert").addEventListener("click", (e) => {
  // alertメッセージをサーバに送信
  socketio.emit("alert");
});
```

# Node.js入門

app.js (server側)

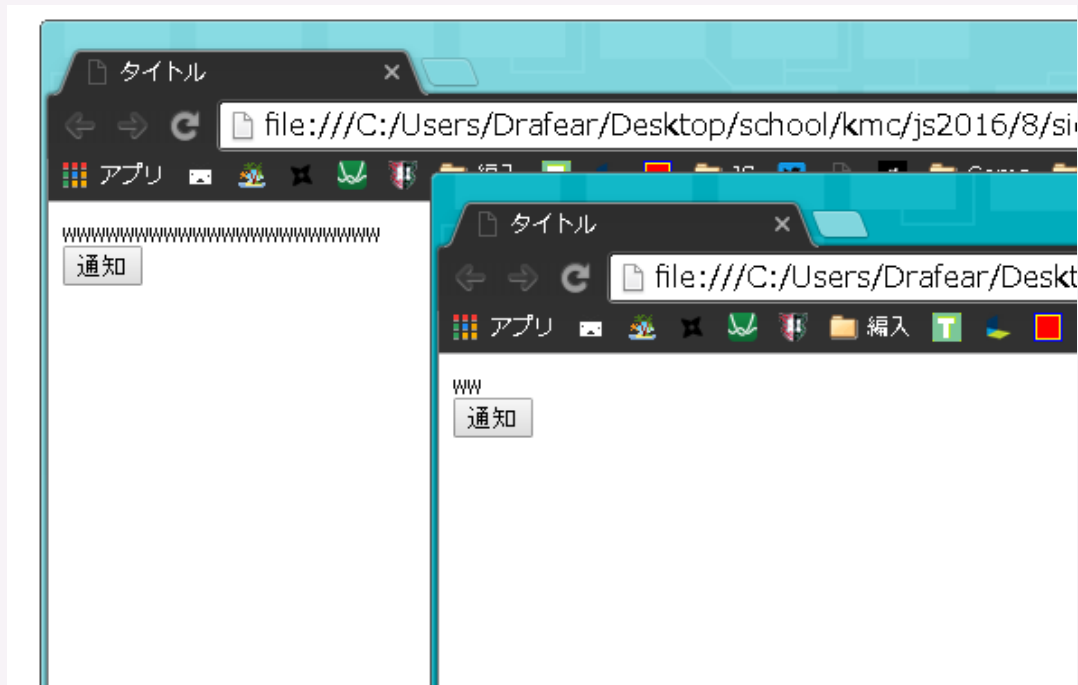
```
'use strict'
const http = require('http');
const server = http.createServer().listen(8080); // 8080ポートで受け付け
const io = require("socket.io").listen(server);
console.log('Server is running!');
// クライアントからの接続があったら
io.sockets.on("connection", (socket) => {
  console.log(`Connected!! id: ${socket.id}`);
  // そのクライアントからalertメッセージが来たら
  socket.on("alert", (data) => {
    // 全クライアントにalertメッセージをブロードキャストする
    io.sockets.emit("alert");
  });
  socket.on("disconnect", () => {
    console.log(`Disconnected id: ${socket.id}`);
  });
});
```



# Node.js入門

- サーバを動かして確認しよう

```
$ node app.js  
Server Running!
```



# Node.js入門

- 誰かのところに接続してみよう

被接続側

```
$ wincmd ipconfig  
...  
IPv4 アドレス .....: 192.168.11.4  
...
```

接続側

main.js (client側)

```
'use strict'  
const socketio = io.connect('http://192.168.11.4:8080');  
...
```



# 要求回数をカウント

- 値を表示する
- 「通知」ボタンを押す度に表示する値をインクリメント

main.js (client側)

```
'use strict'
const socketio = io.connect('http://localhost:8080');
socketio.on("alert", (data) => {
  document.getElementById("mes").textContent = data; // 変更
});
document.getElementById("btnAlert").addEventListener("click", (e) => {
  socketio.emit("alert");
});
```

# Node.js入門

app.js (server側)

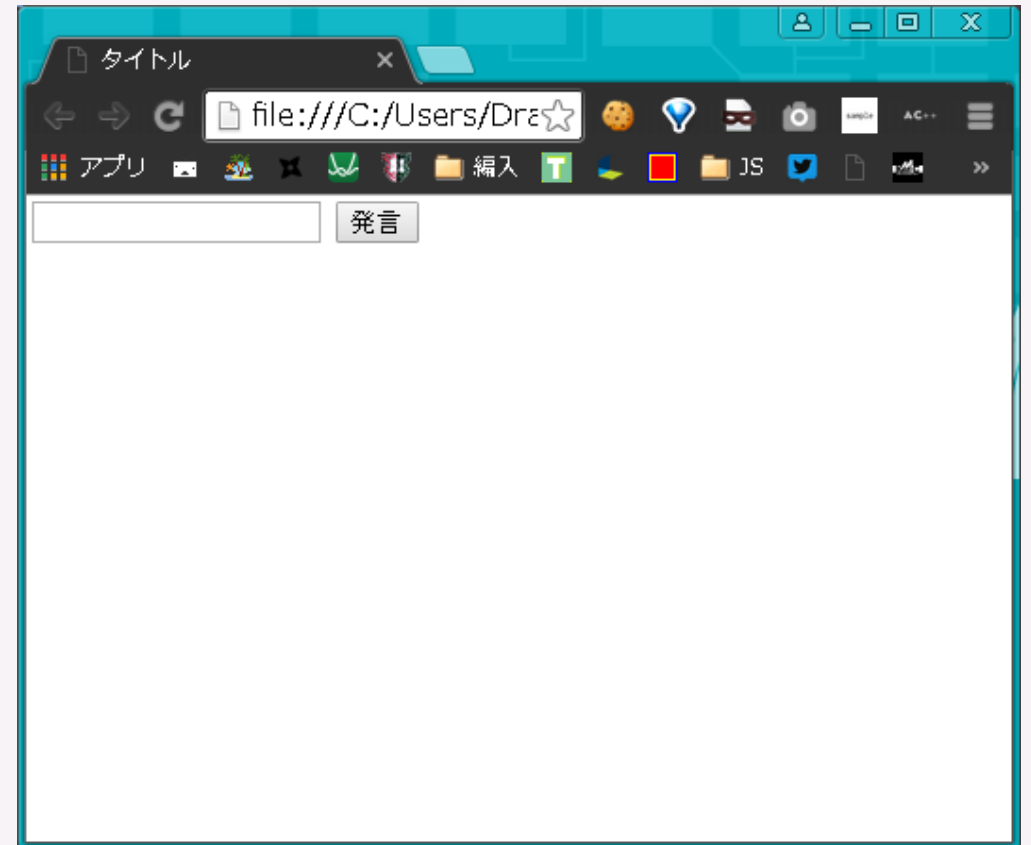
```
'use strict'
const http = require('http');
const server = http.createServer().listen(8080);
const io = require("socket.io").listen(server);
console.log('Server running!');
const global = { cnt: 0 }; // 追加
io.sockets.on("connection", (socket) => {
  console.log(`Connected!! id: ${socket.id}`);
  socket.on("alert", (data) => {
    ++global.cnt; // 追加
    io.sockets.emit("alert", global.cnt); // 変更
  });
  socket.on("disconnect", () => {
    console.log(`Disconnected id: ${socket.id}`);
  });
});
```

# 演習

- 匿名リアルタイムチャットを作ってみよう
  - 時間が余ったら発言者も入力させて表示してみよう

index.html

```
<div id="log"></div>  
<input type="text" id="tbMes">  
<button id="btnSay">発言</button>
```



# 演習

- 匿名リアルタイムチャットを作ってみよう
  - 時間が余ったら発言者も入力させて表示してみよう

main.js (client側)

```
'use strict'
const socketio = io.connect('http://localhost:8080');
socketio.on("addLog", (data) => {
  document.getElementById("log").innerHTML += `<p>${data}</p>`;
});
document.getElementById("btnSay").addEventListener("click", (e) => {
  const eMes = document.getElementById("tbMes");
  socketio.emit("say", eMes.value);
  eMes.value = "";
});
```

# 演習

- これで完成
  - 誰かから発言があれば(発言者含め)全員にブロードキャストするだけ

app.js (server側)

```
'use strict'
const http = require('http');
const server = http.createServer().listen(8080);
const io = require("socket.io").listen(server);
console.log('Server running!');
io.sockets.on("connection", (socket) => {
  socket.on("say", (data) => {
    io.sockets.emit("addLog", data);
  });
});
```

# 演習

- 拡張例 (入退室ログを出力)

app.js (server側)

```
...
io.sockets.on("connection", (socket) => {
  io.sockets.emit("addLog", "匿名さんが入室しました"); // 追加
  socket.on("say", (data) => {
    io.sockets.emit("addLog", data);
  });
  socket.on("disconnect", () => { // 追加
    io.sockets.emit("addLog", "匿名さんが退室しました"); // 追加
  }); // 追加
});
```



# 演習

- 発言者も入力させて表示してみる

index.html

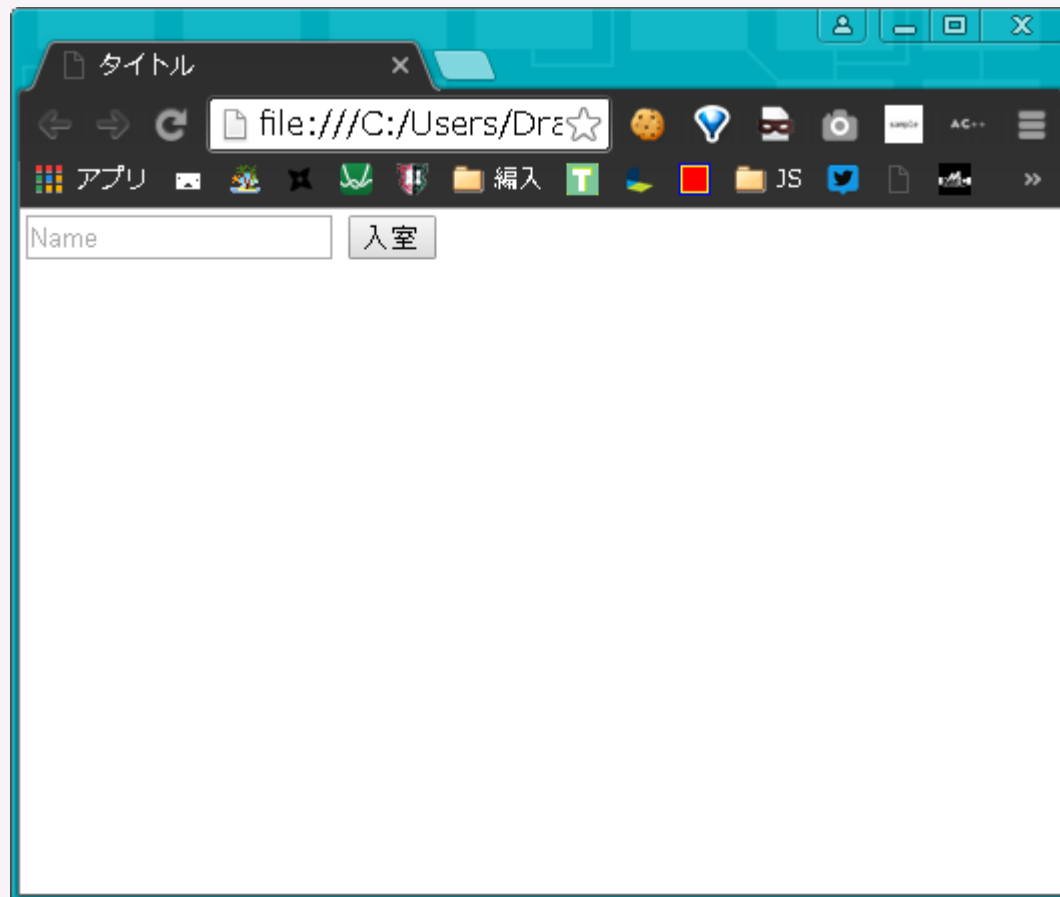
```
<div id="screen">
  <div id="enter">
    <input type="text" id="name" placeholder="Name">
    <button id="btnEnter">入室</button>
  </div>
  <div id="room" class="hide">
    <div id="log"></div>
    <input type="text" id="tbMes">
    <button id="btnSay">発言</button>
  </div>
</div>
```

style.css

```
body {
  margin: 0;
}
#screen > div {
  box-sizing: border-box;
  width: 100vw;
  height: 100vh;
  padding: 3px;
}
.hide {
  display: none;
}
```

# 演習

- 発言者も入力させて表示してみる



# 演習

- 発言部分は同じ
  - 前のプログラムに以下をそのまま追記すればok

main.js (一部, 入室部分)

```
const setName = (name) => {  
    socketio.emit("setName", name);  
};  
const enterRoom = () => {  
    document.getElementById("enter").classList.add("hide");  
    document.getElementById("room").classList.remove("hide");  
    socketio.emit("enter");  
};  
document.getElementById("btnEnter").addEventListener("click", (e) => {  
    const name = document.getElementById("name").value;  
    setName(name === "" ? "匿名さん" : name);  
    enterRoom();  
});
```

# 演習

- `console.log("Server is running!")` 以下を変更. これで完成.

app.js

```
const global = {};  
io.sockets.on("connection", (socket) => {  
  global[socket.id] = { name: "" };  
  socket.on("say", (mes) => {  
    io.sockets.emit("addLog", `${global[socket.id].name}: ${mes}`);  
  });  
  socket.on("setName", (name) => { global[socket.id].name = name; });  
  socket.on("enter", () => {  
    io.sockets.emit("addLog", `${global[socket.id].name}が入室しました`);  
  });  
  socket.on("disconnect", () => {  
    io.sockets.emit("addLog", `${global[socket.id].name}が退室しました`);  
    delete global[socket.id];  
  });  
});
```

# 演習

- これでもok

app.js

```
io.sockets.on("connection", (socket) => {  
  const me = {};  
  socket.on("say", (mes) => {  
    io.sockets.emit("addLog", `${me.name}: ${mes}`);  
  });  
  socket.on("setName", (name) => {  
    me.name = name;  
  });  
  socket.on("enter", () => {  
    io.sockets.emit("addLog", `${me.name}が入室しました`);  
  });  
  socket.on("disconnect", () => {  
    io.sockets.emit("addLog", `${me.name}が退室しました`);  
  });  
});
```

# 部屋

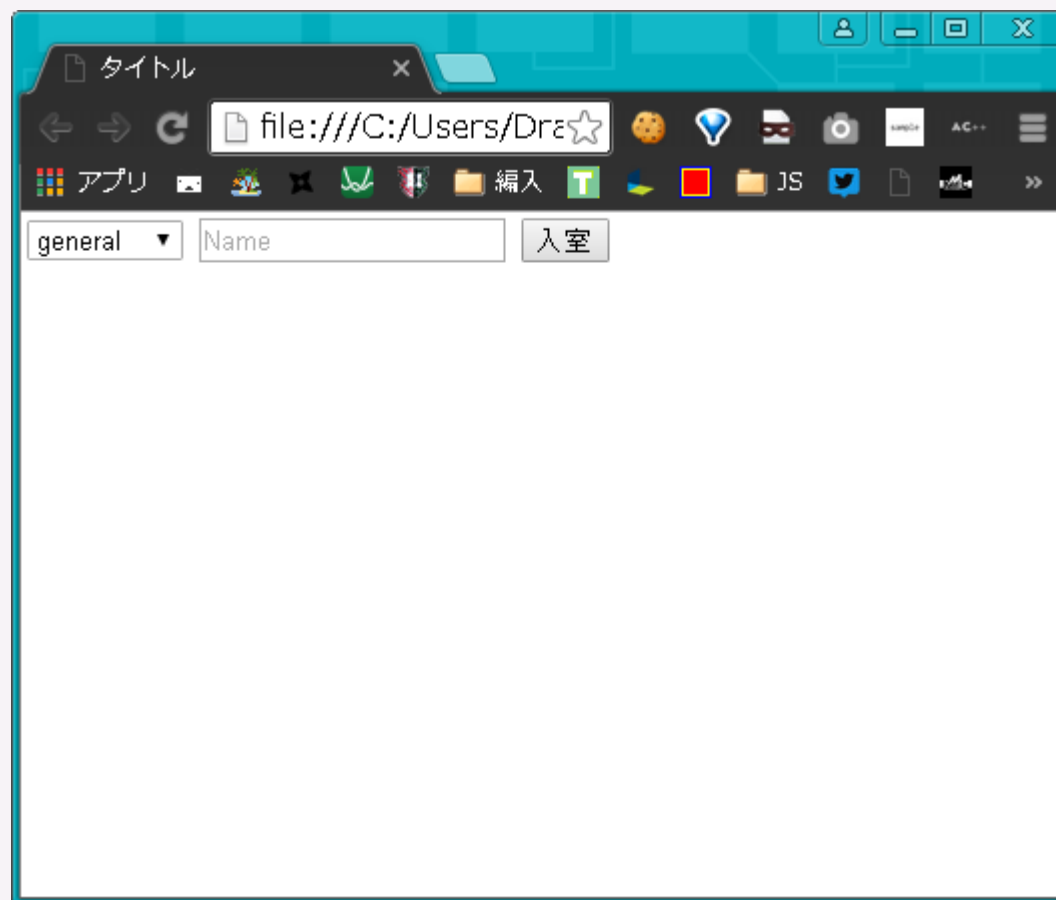
- サーバからクライアントにemit(メッセージを送信)するとき  
"一人" もしくは "全員" に送信できる
  - 一人: `socket.emit(...)`
  - 全員: `io.sockets.emit(...)`
- 特定のグループにemitするにはどうすればよいか
  - 特定のグループのsocketを配列で管理してforでまわす??
  - 遅そう & disconnect時に該当socketをremoveする処理も必要
  - 良い方法があります

# 部屋

- `socket.join("部屋名")`
- `socket.leave("部屋名")`
- `io.sockets.in("部屋名").emit(...)`

# 部屋

- さっきのチャットアプリに部屋機能を実装してみる





# 部屋

index.html

```
<div id="screen">
  <div id="enter">
    <select id="roomId">
      <option value="general">general</option>
      <option value="active">active</option>
      <option value="dream">dream</option>
      <option value="javascript">javascript</option>
    </select>
    <input type="text" id="name" placeholder="Name">
    <button id="btnEnter">入室</button>
  </div>
  <div id="room" class="hide">
    <div id="log"></div>
    <input type="text" id="tbMes">
    <button id="btnSay">発言</button>
  </div>
</div>
```

# 部屋

main.js

```
...  
const enterRoom = (roomId) => { // 変更  
  document.getElementById("enter").classList.add("hide");  
  document.getElementById("room").classList.remove("hide");  
  socketio.emit("enter", roomId); // 変更  
};  
document.getElementById("btnEnter").addEventListener("click", (e) => {  
  const name = document.getElementById("name").value;  
  setName(name === "" ? "匿名さん" : name);  
  // setName(name || "匿名さん"); でもOK  
  enterRoom(document.getElementById("roomId").value); // 変更  
});
```

# 部屋

app.js

```
const global = {};  
io.sockets.on("connection", (socket) => {  
  global[socket.id] = { name: "" };  
  socket.on("say", (mes) => {  
    io.sockets.in(global[socket.id].roomId).emit("addLog", `${global[socket.id].name}: ${mes}`);  
  });  
  socket.on("setName", (name) => { global[socket.id].name = name; });  
  socket.on("enter", (roomId) => {  
    global[socket.id].roomId = roomId;  
    socket.join(roomId);  
    io.sockets.in(roomId).emit("addLog", `${global[socket.id].name}が入室しました`);  
  });  
  socket.on("disconnect", () => {  
    io.sockets.in(global[socket.id].roomId)  
      .emit("addLog", `${global[socket.id].name}が退室しました`);  
    delete global[socket.id];  
  });  
});
```



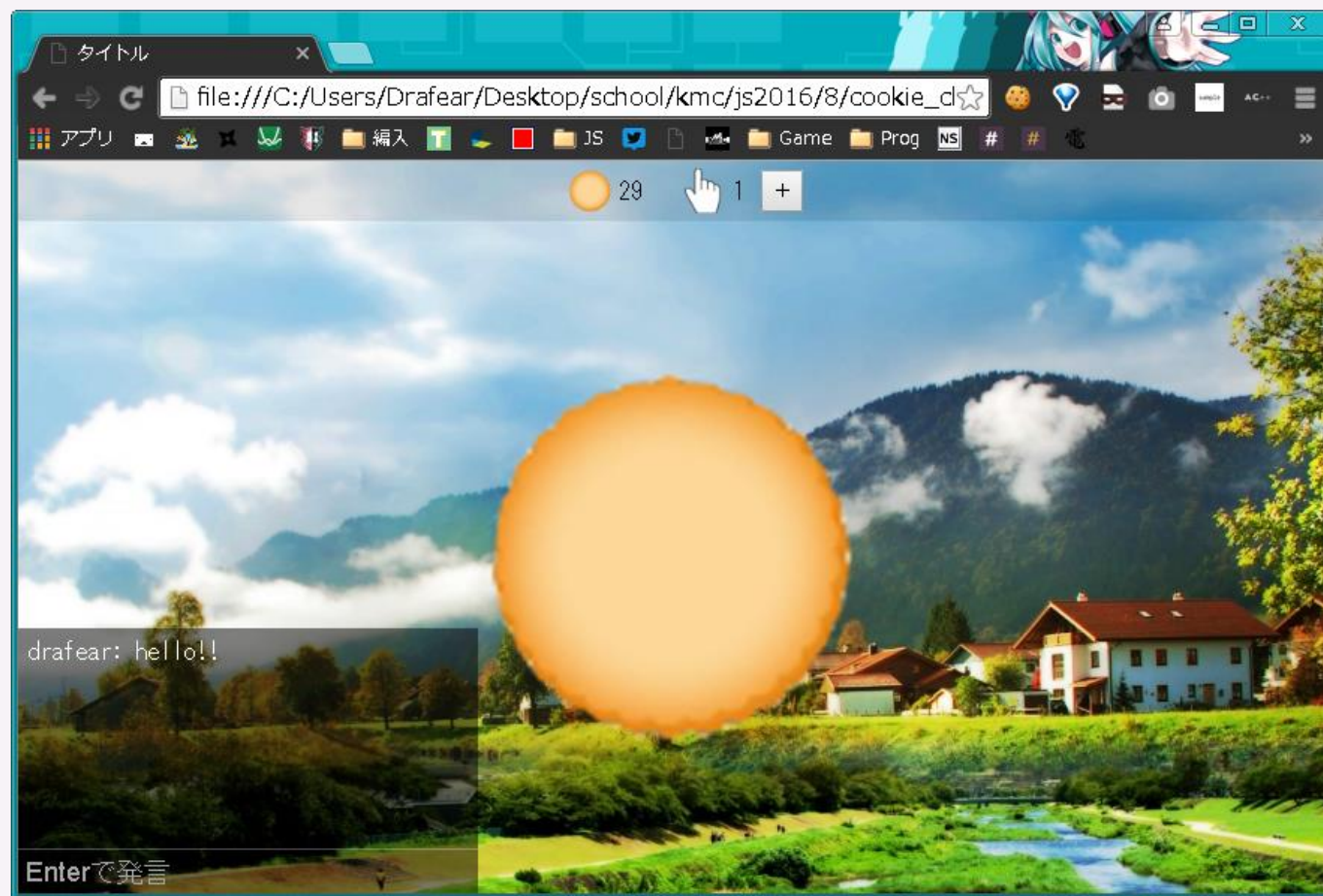
何かを作ろう

# 何かを作ろう

- socket.ioを用いて何かを作ってみよう
- 来週ネット上に公開します
- サーバを再起動してもデータを残す方法も来週  
(今はグローバル変数でデータを管理しよう)

# 例

- ビスケットをみんなで叩いて増やすゲーム



# 何かを作ろう

- 分からないことがあればどんどん聞いて下さい
  - エラーやバグ
  - これやりたいんだけどどうするんだっけ？  
(昔のスライドから漁るの大変なので聞いてもらえたら早いです！)
  - こんなことしたいんだけどどうすれば良さそう？
  - パソコンが壊れた！



# おすすめツール



# Qonsole

- 上から降りてくるコンソール. 便利.
  - <https://github.com/joedf/Qonsole/releases/tag/v1.4.2>

# Qonsole

- 設定

- 以下を qonsole.bat として適当な場所に保存  
(msys2\_shell.cmd へのパスはご自身の環境に合わせて変更して下さい)

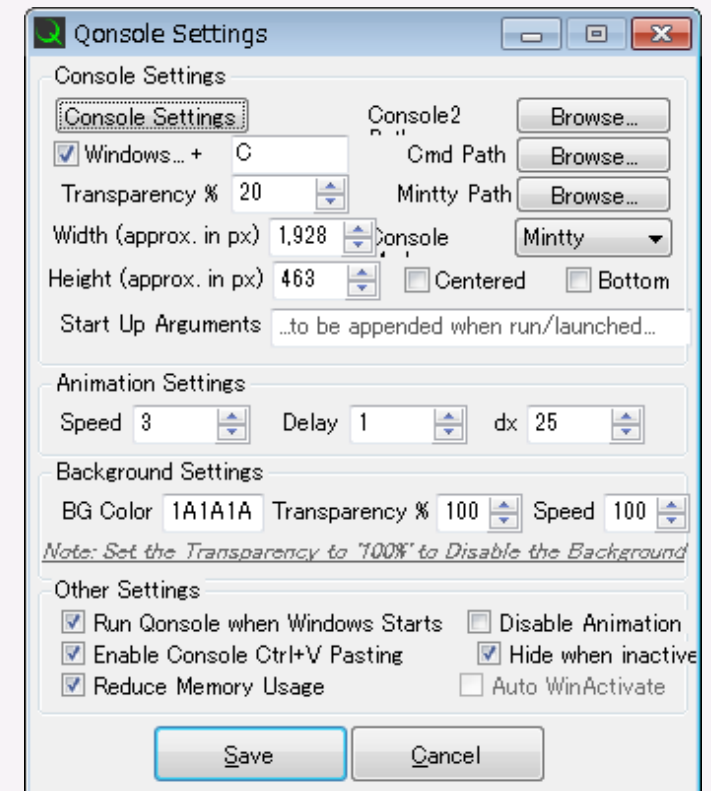
qonsole.bat

```
call C:¥msys64¥msys2_shell.cmd -mingw64  
timeout /T 1
```

# Qonsole

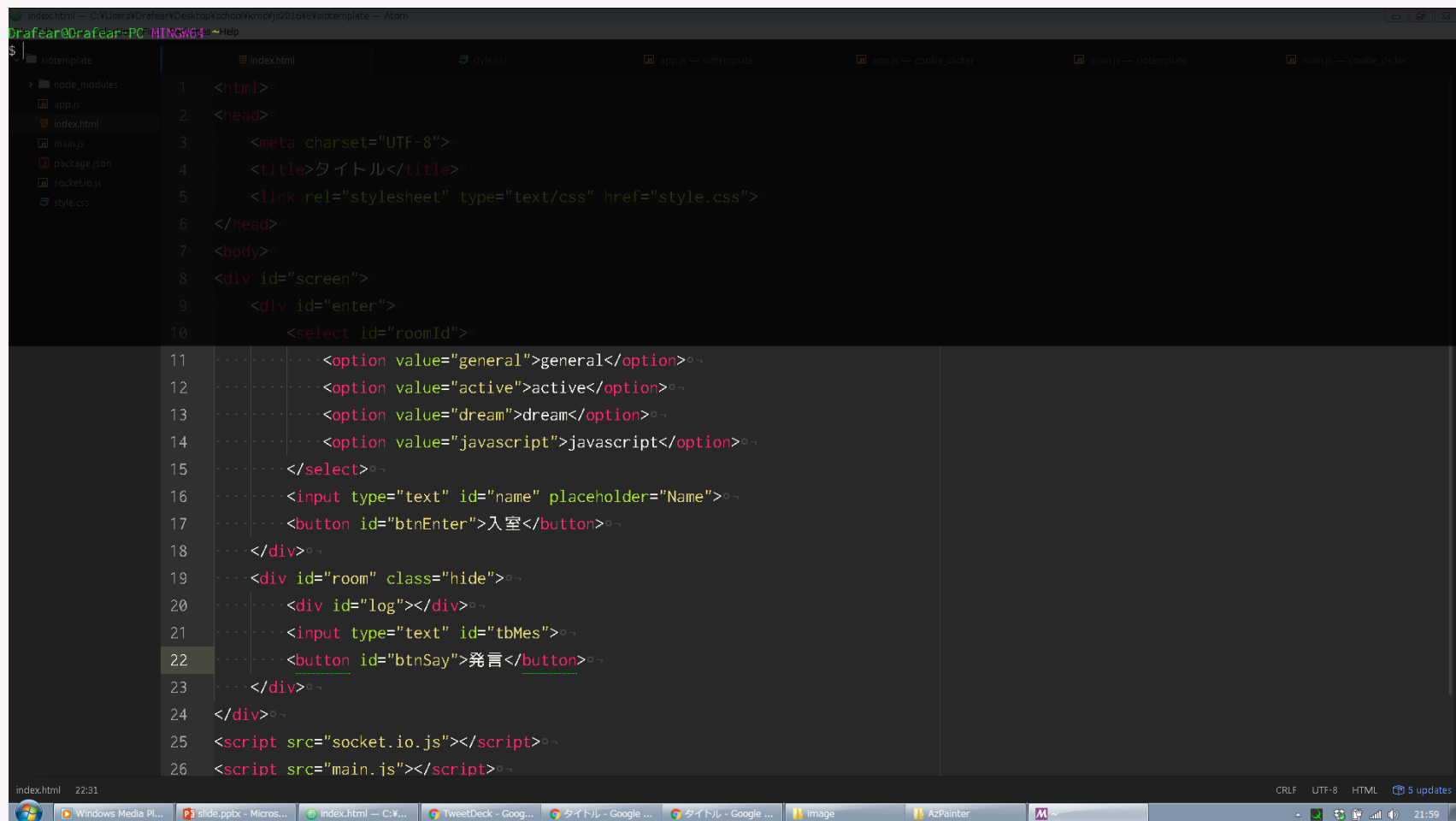
- 設定

- Qonsoleを起動し, タスクトレイのQonsoleを右クリックしてSettings
- Mintty Path の Browse から先ほど生成した qonconsole.bat を選択
- Mintty Path のすぐ下の項目を Mintty に
- 他は好きに設定して下さい (設定例→)



# Qonsole

- Windows + C (デフォルト) で快適なQonsoleライフを。



```
index.html — C:\Users\drafean\Desktop\school\kmo\js2015\k\template — Atom
drafean@drafean-PC: MINGW64 ~
$ |
nodetemplate
├─ node_modules
├─ app.js
├─ index.html
├─ main.js
├─ package.json
├─ socket.io.js
└─ style.css
1 <html>
2 <head>
3   <meta charset="UTF-8">
4   <title>タイトル</title>
5   <link rel="stylesheet" type="text/css" href="style.css">
6 </head>
7 <body>
8   <div id="screen">
9     <div id="enter">
10       <select id="roomId">
11         <option value="general">general</option>
12         <option value="active">active</option>
13         <option value="dream">dream</option>
14         <option value="javascript">javascript</option>
15       </select>
16       <input type="text" id="name" placeholder="Name">
17       <button id="btnEnter">入室</button>
18     </div>
19     <div id="room" class="hide">
20       <div id="log"></div>
21       <input type="text" id="tbMes">
22       <button id="btnSay">発言</button>
23     </div>
24   </div>
25 <script src="socket.io.js"></script>
26 <script src="main.js"></script>
```

# Qonsole

- 名前の由来は Quakeってオングのコンソールが元らしい
- ちなみに Quake は高速で動くFPSゲーム
- 面白かったけどプログラマーが集い無理ゲーすぎたのでやめた
- 動画
  - <https://www.youtube.com/watch?v=yIFl4cTPxzA>