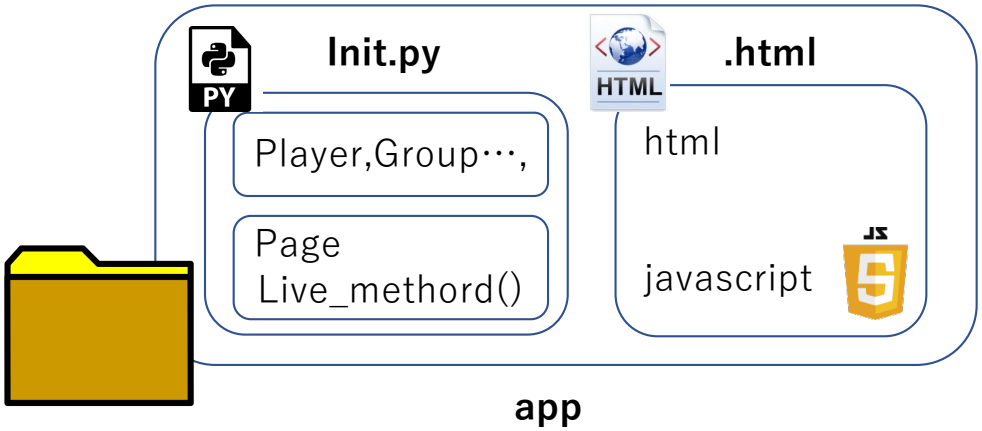
otree (live page機能)を用いた動的なプログラミング

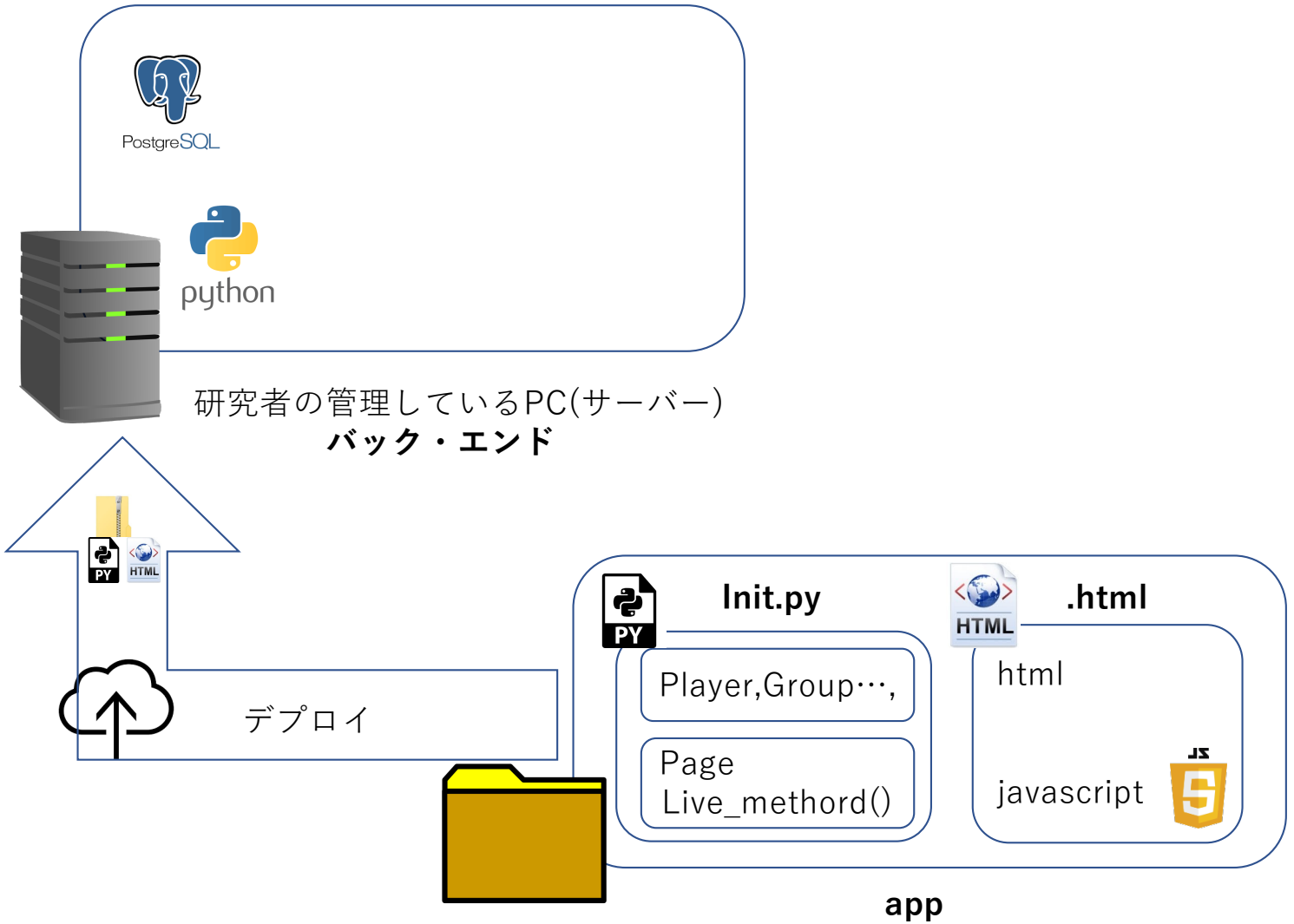
Otree講習会

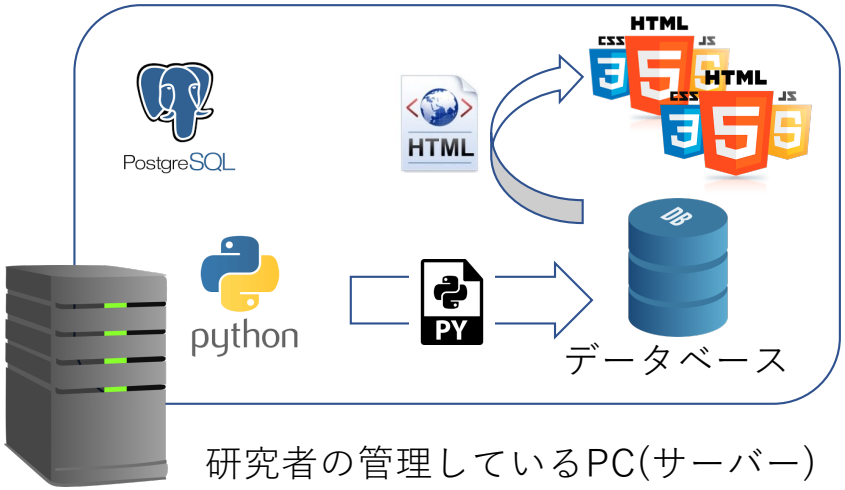
2022/7/14

三上亮

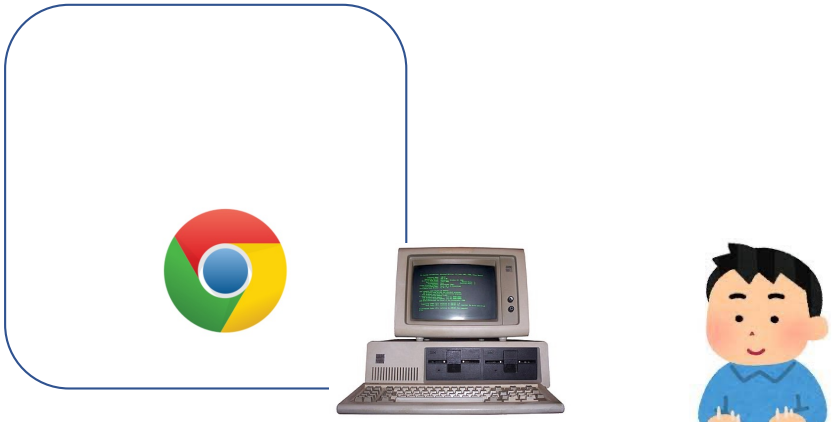
Otreeのイメージ


PY
Init.py
HTML
.html
Player,Group…,
Page
Live_methord()
html
javascript
app

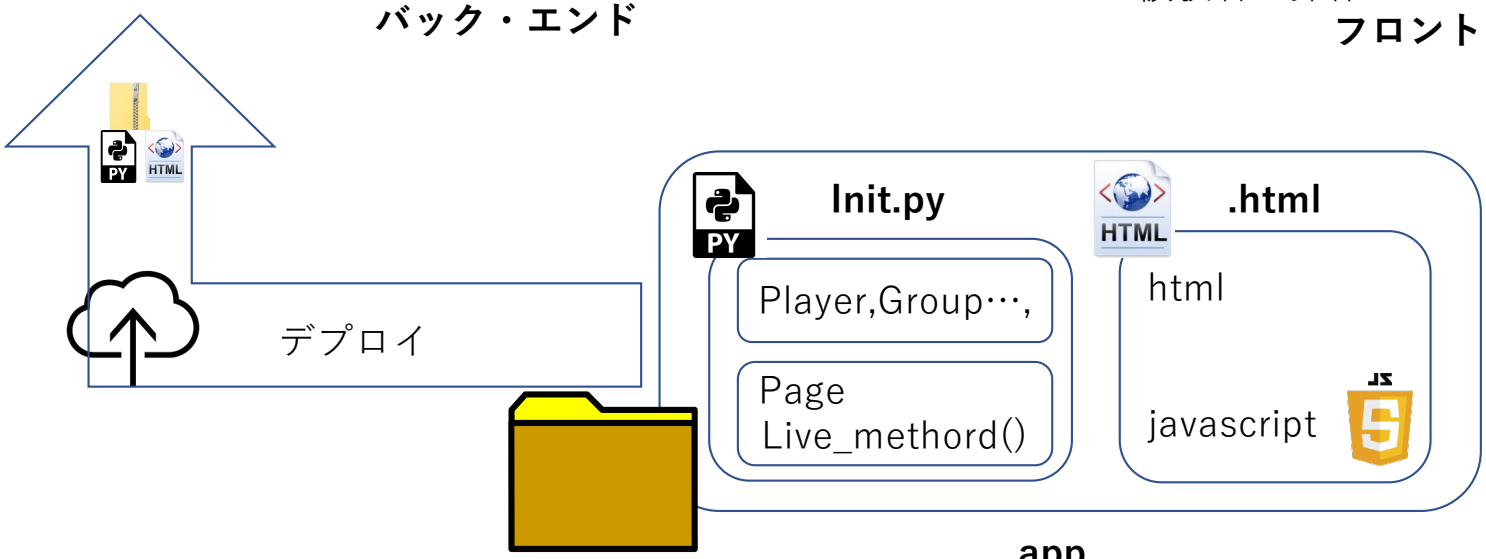

PostgreSQL
python
研究者の管理しているPC(サーバー)
バック・エンド
PY
HTML
デプロイ
PY
Init.py
Player,Group…,
Page
Live_methord()
HTML
.html
html
javascript
app



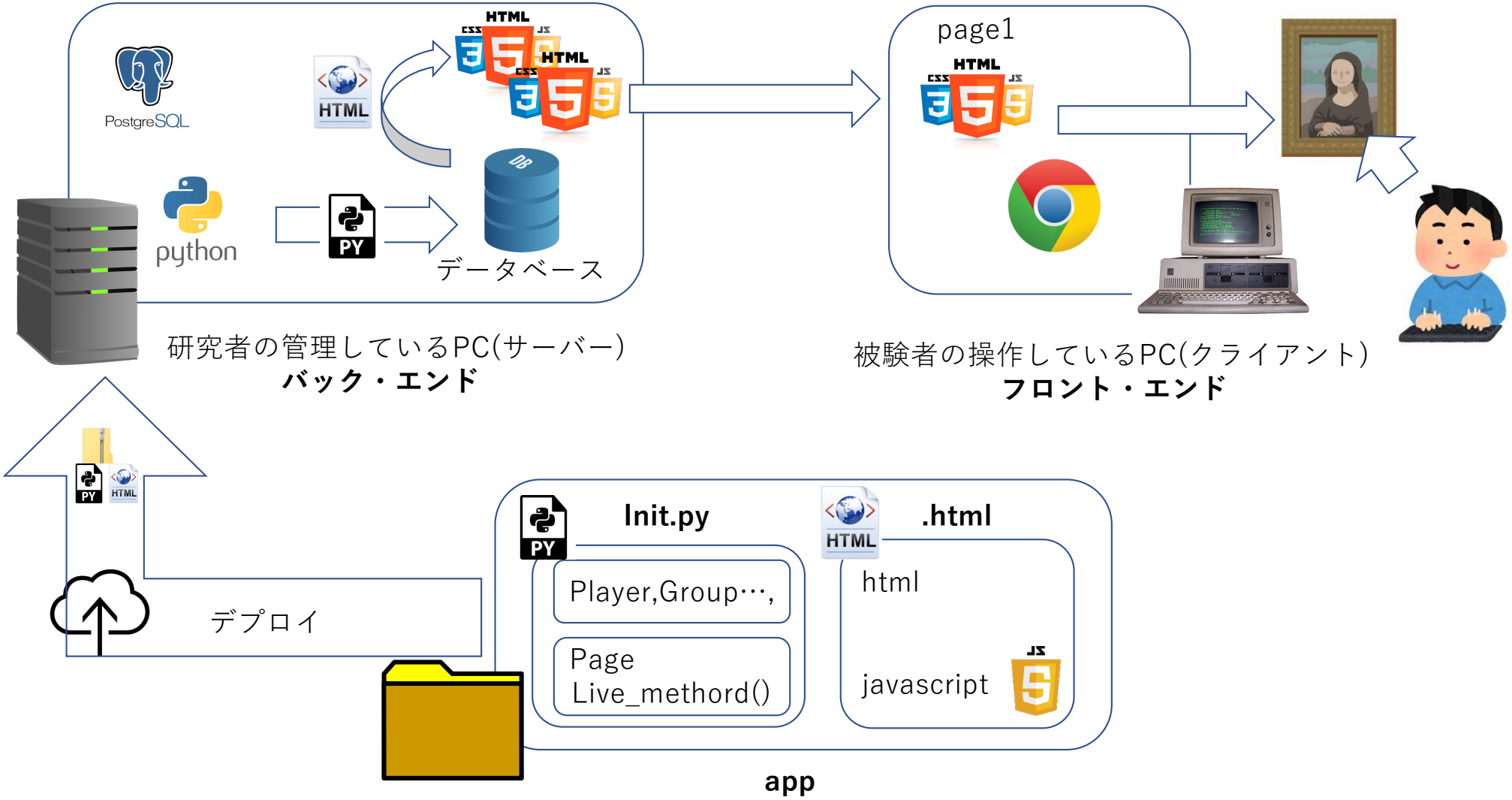
研究者の管理しているPC(サーバー)
バック・エンド

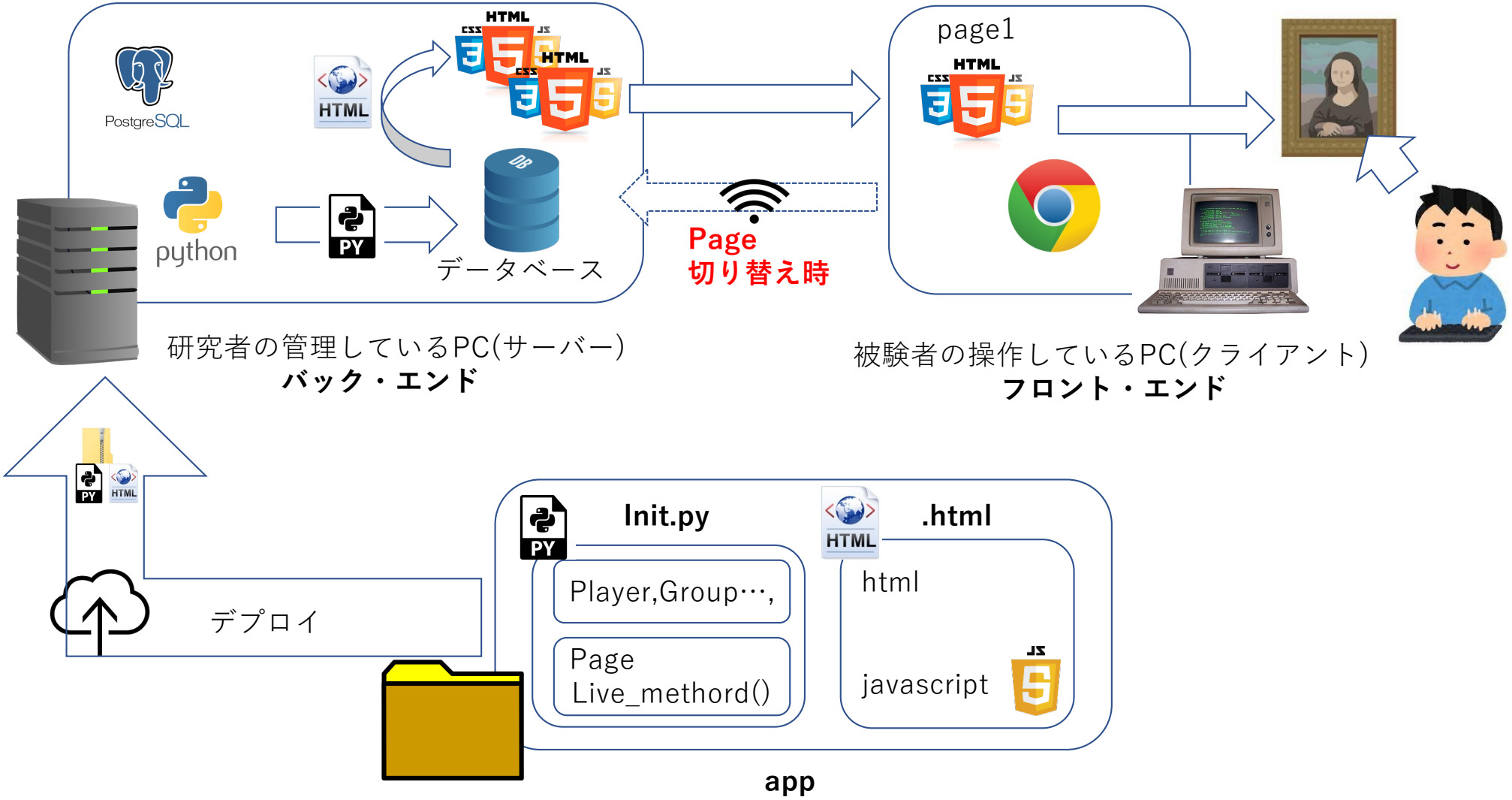


被験者の操作しているPC(クライアント)
フロント・エンド



app


PostgreSQL
HTML
HTML
python
PY
DB
データベース
page1
HTML
研究者の管理しているPC(サーバー)
バック・エンド
被験者の操作しているPC(クライアント)
フロント・エンド
PY
HTML
デプロイ
PY
Init.py
Player,Group…,
Page
Live_methord()
HTML
.html
html
javascript
app


PostgreSQL
HTML
python
PY
データベース
Page
切り替え時
page1
研究者の管理しているPC(サーバー)
バック・エンド
被験者の操作しているPC(クライアント)
フロント・エンド
デプロイ
Init.py
.html
Player,Group…,
Page
Live_methord()
html
javascript
app

動的なプログラミング

- 実験中（=init.pyがrunした後=サーバーにデプロイ後）に決定（もしくは変化）するような変数が必要な場合。

→事前にhtmlやDBを定義できない。

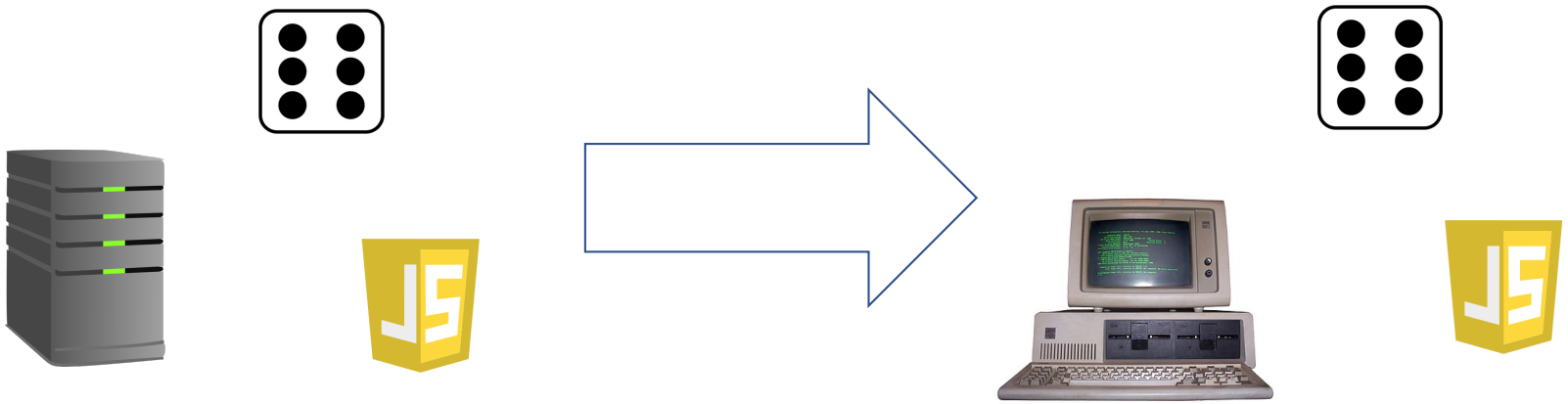
例1:　対戦相手が選んだ最新の選択肢を常に表示し続ける。

→　実験中にクライアントが持っているhtmlフォイル(+ DB) を書き換え続ける。

例2:　個人レベルで、ある条件を満たすまでラウンドが継続する。

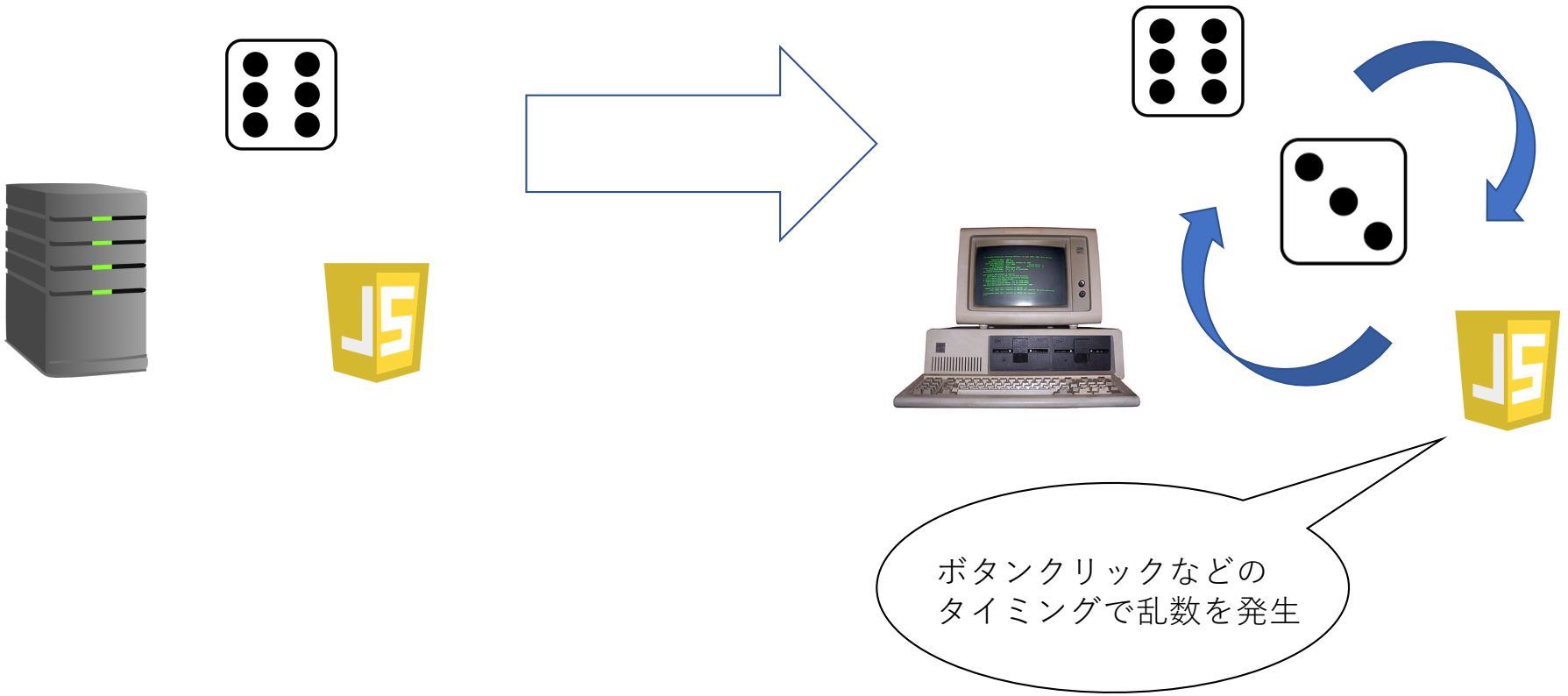
→　被験者によって、playerクラスに保存するデータが異なる。

例1
Javascript （クライアント側）のみで制御

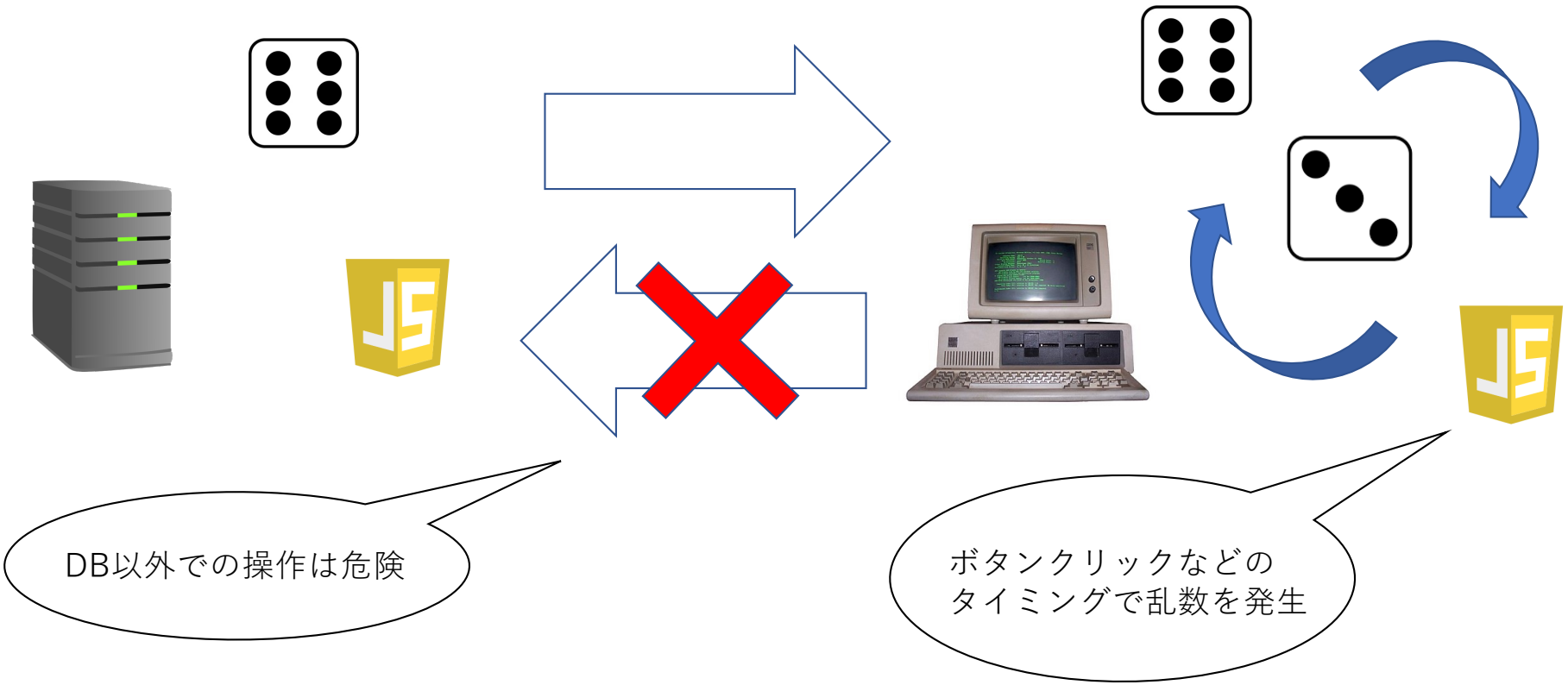


動的な変数（乱数）を、 Javascriptを使いクライアント側で制御する様な仕様の設計を考える。

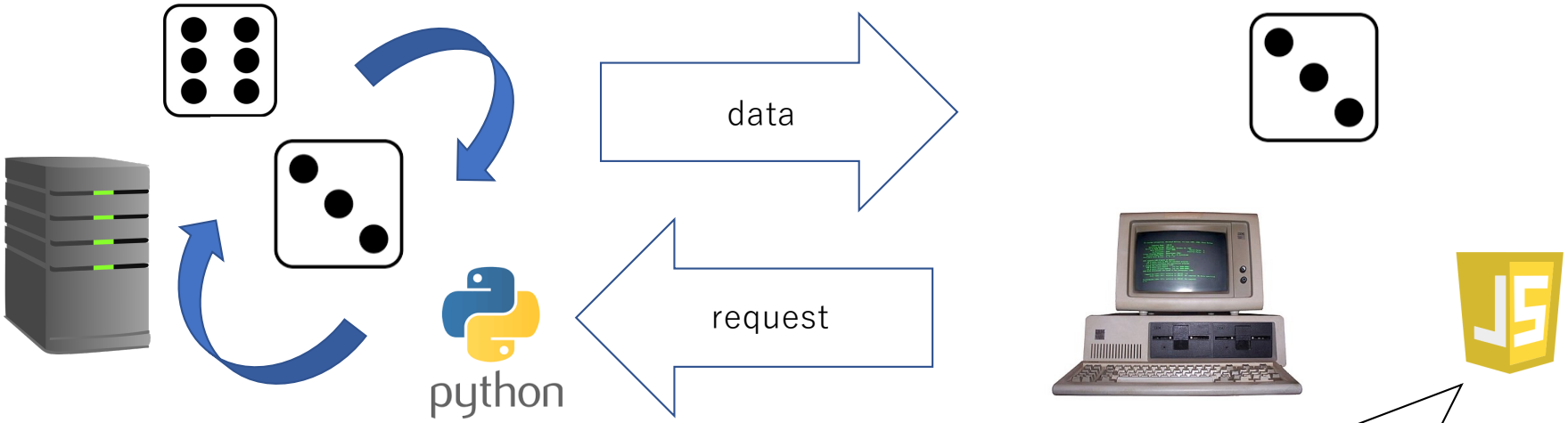
例1
Javascript （クライアント側）のみで制御



例1
Javascript（クライアント側）のみで制御



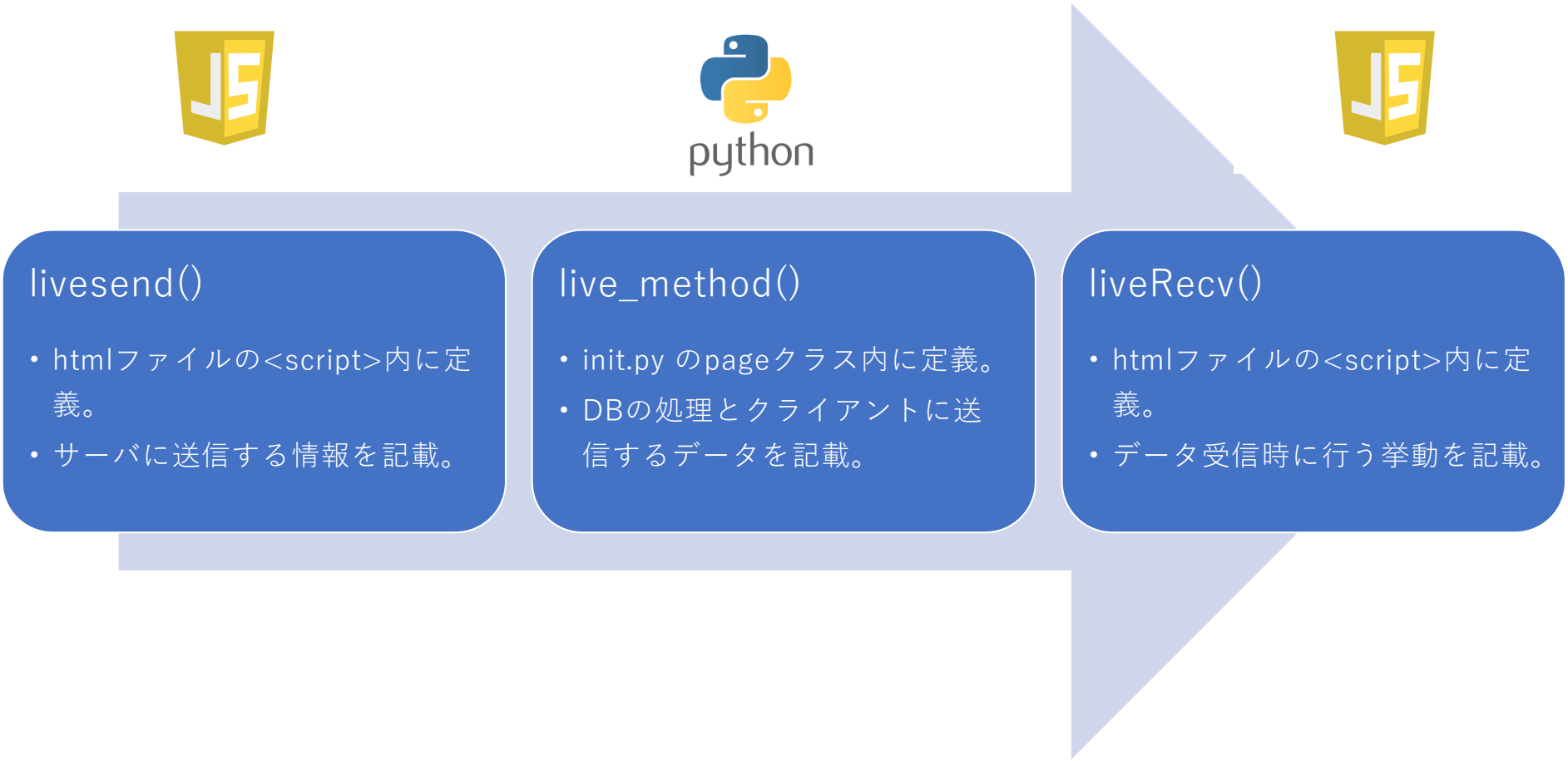
データは全てDB側(python)で管理すべき



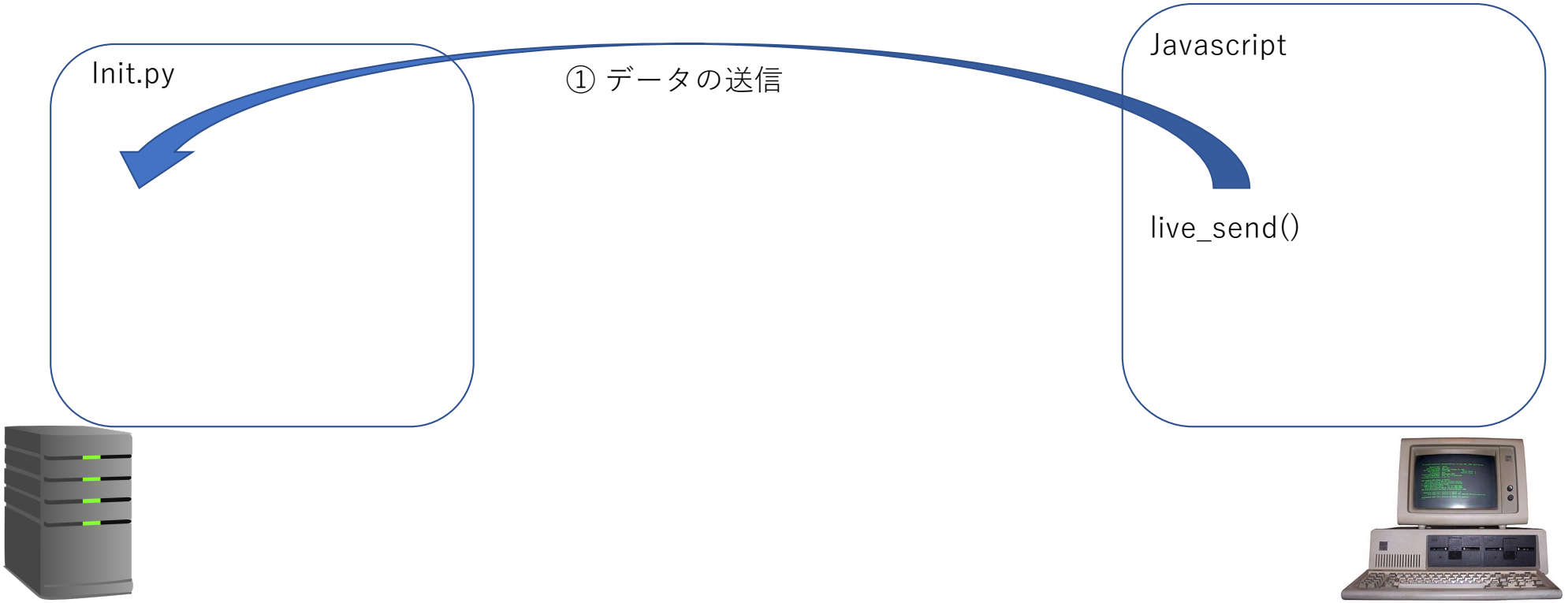
- requestに従い、乱数を発生。
- 結果を送信。

- resuestのタイミング制御。
- 新しく受け取ったデータに表示を更新。

live_method()の挙動の順序



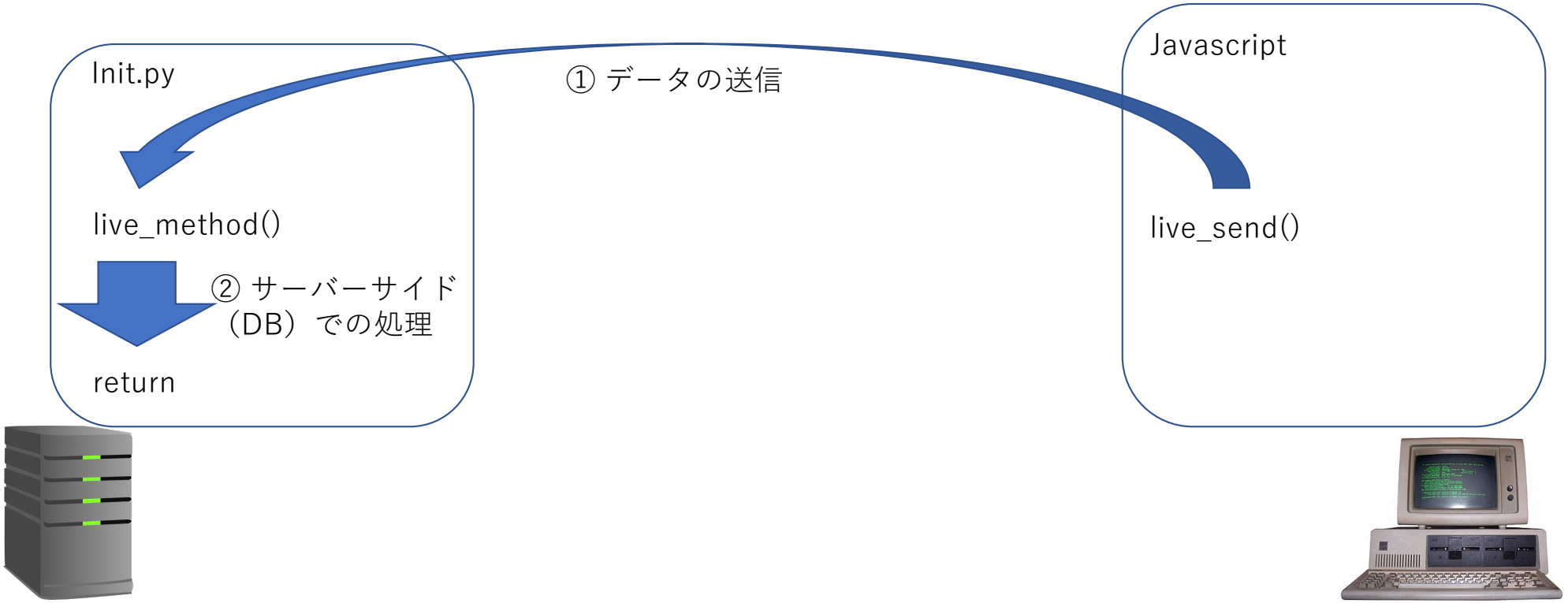
live_method()の挙動の順序



研究者の管理しているPC(サーバー)
バック・エンド

被験者の操作しているPC(クライアント)
フロント・エンド

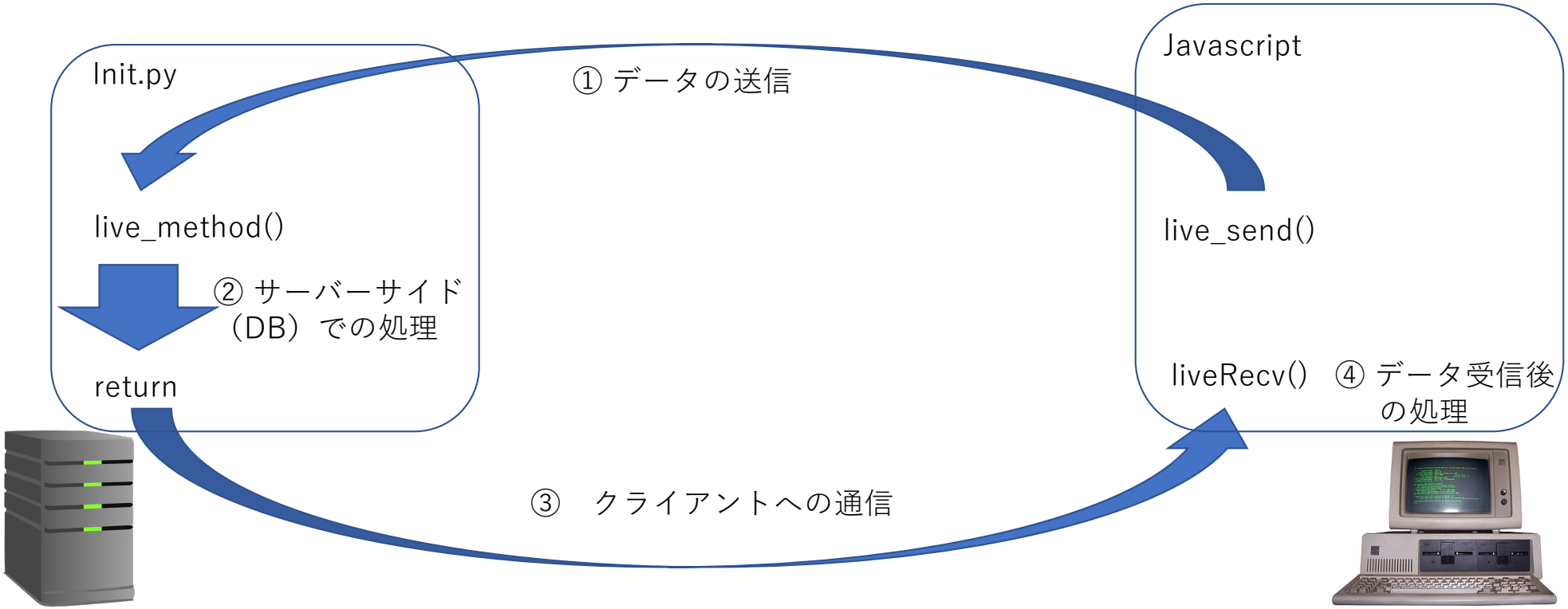
live_method()の挙動の順序



研究者の管理しているPC(サーバー)
バック・エンド

被験者の操作しているPC(クライアント)
フロント・エンド

live_method()の挙動の順序



double_auctionプログラム (more demos)

https://www.otreehub.com/projects/otree-more-demos/
からダウンロード可能

デフォルトの設定では

- 4人1グループ（買手2、売手2）のゲーム。
- 売手は初期に3つの財を保有している。
- 毎回、買手→売手のオファーを**順に**全通りチェック。
- 買手のオファー> 売手のオファーが見つかった段階で、**買手の提示価格で**取引が行われる。

DBの構造

Playerクラス

- `is_buyer = models.BooleanField()`
- `current_offer = models.CurrencyField()`
- `break_even_point = models.CurrencyField()`
- `num_items = models.IntegerField()`

Groupクラス:

- `start_timestamp = models.IntegerField()`

ページの構成

0. creating_session()
 役割を振り分け

1. WaitToStart
2. Trading　←メイン画面
3. ResultsWaitPage
4. Results

自作関数

find_match(buyers, sellers):

- buyers　：　オファーを提供しているplayr（買手）のリスト
- sellers　：　オファーを提供しているplayr（売手）のリスト
- 出力　　：　条件を満たす[player1, player2]というリスト

otree側の組み込み関数

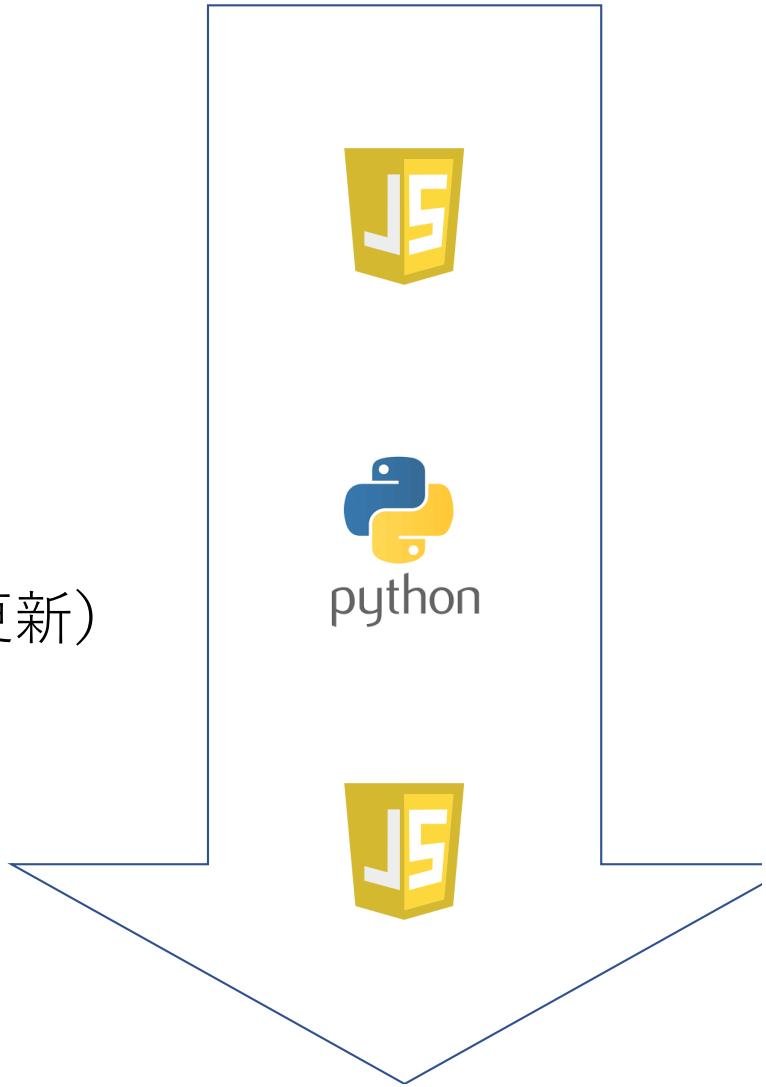
- live_method(player: Player, data)
- Transaction(ExtraModel):

Trade.html

liveSend():
- サーバーにオファーの額を送信

live_method():
- 受け取ったオファーをDBに更新。
- テーブル（未確定のオファーなど一覧を更新）
- 一覧情報を全参加者に送信

liveRecv(data) :
- 自分の取引が成立の場合、結果を表示。
- 画面の更新。



Trade

Time left to complete this page: **0:46**

You sold to player 1 for 3 points

Your role	**seller**
Your break-even point (you should sell for more than)	**48 points**
Items in your possession	**2**
Your current offer	**111 points**
Profits	**-45 points**

3 Make offer

Bids

Asks

Trade history

Trade

Time left to complete this page: **0:46**

You sold to player 1 for 3 points

```
<p id="news" style="color: green"></p>
```

Your role	**seller**
Your break-even point (you should sell for more than)	**48 points**
Items in your possession	**2**
Your current offer	**111 points**
Profits	**-45 points**

```
<input type="number" id="my_offer">
```

3 Make offer

```
<button type="button" onclick="sendOffer()"
id="btn-offer">Make offer</button>
```

Bids

```
<h4>Bids</h4>
<table id="bids_table"></table>
```

Asks

```
<h4>Asks</h4>
<table id="asks_table"></table>
```

Trade history

Trade

Time left to complete this page: **0:46**

You sold to player 1 for 3 points

Your role	**seller**
Your break-even point (you should sell for more than)	**48 points**
Items in your possession	**2**
Your current offer	**111 points**
Profits	**-45 points**

3 Make offer

Bids

Asks

```
<input type="number" id="my_offer">

                <button type="button" onclick="sendOffer()" id="btn-offer">Make offer</b
```

Trade history

ExtraModel

- メインのDBとは別に柔軟な（階層的な）DBを作成する。
- 実験中に自由に更新（データの追加）ができる。最終的に保存したい場合はfilter()で呼び出して、create()で作成した変数をDBに書き込めば良い。

- 今回の場合、取引の履歴をDBに格納するのが難しい。

→playerごとに取引の回数が異なる＆事前には未知

ID	player.data1	player.data2	player.data3	player.data4
001	a	b		
002	c	d	e	f

リレーショナル・DB

- **非リレーショナルデータベース（NoSQL）は階層構造を持つデータに構造に弱い**

世帯id	名前	人数	地域
1	三上	1	長野
2	竈門	6	東京
3	虎杖	2	東京

親テーブル

主キー

世帯id	名前	家族内id
2	炭治郎	1
2	禰豆子	2
2	六太	3

子テーブル

外部キー

ExtraModel(Transaction)の構造

- `group = models.Link(Group)`
- `buyer = models.Link(Player)`
- `seller = models.Link(Player)`
- `price = models.CurrencyField()`
- `seconds = models.IntegerField()`

DB（親）

主キー

候補キー

player	group	player.data2
001	1	b
002	1	d

Extramodel（子）

外部キー

group	player1	player2	price	seconds
1	001	004	60	10
1	001	002	50	20