PyCon JP 2018

ひろがる Python

Djangoだって
カンバンつくれるもん

Django Channelsで作ろうカンバンアプリケーション

PyConJP 2018

@denzowill or denzow

タイトルを書いてたときは酔ってた。
今は後悔している

お前誰よ？

- でんぞう (@denzowill, denzow)
- scouty,inc シニアエンジニア
  - Scrapy, Djangoあたり
- DBスペシャリスト(PostgreSQLが好き、●racleは得意だけど苦手)
- (元?)StartPythonClubスタッフ

お前誰よ？

https://speakerdeck.com/denzow/imasarazhen-rifan-ru-django-migration

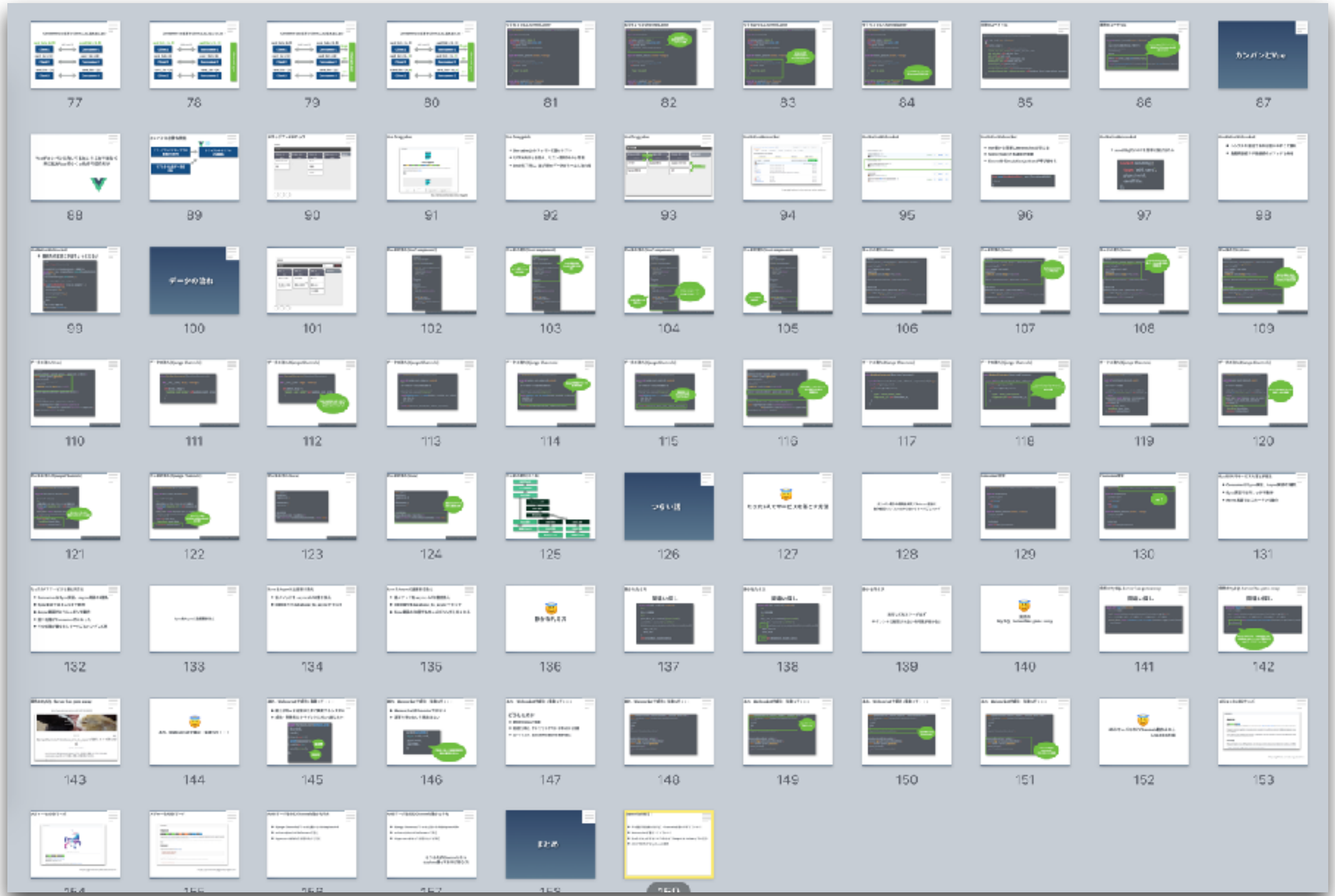
https://qiita.com/denzow/items/77df4b45cfbbf2f0df92

全90Pは50分には長すぎた

https://speakerdeck.com/denzow/imasarazhen-rifan-ru-django-migration

https://qiita.com/denzow/items/77df4b45cfbbf2f0df92

お前誰よ？


77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
データの流れ
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
つなぐ話
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
まとめ

お前誰よ？

人工知能が、天職を探し出す。

日本初のAIヘッドハンティングサービス

scoutyとは

高度なAIで、能力に最適な企業を自動分析。
エンジニアのための、登録不要の新しい転職サービスです。

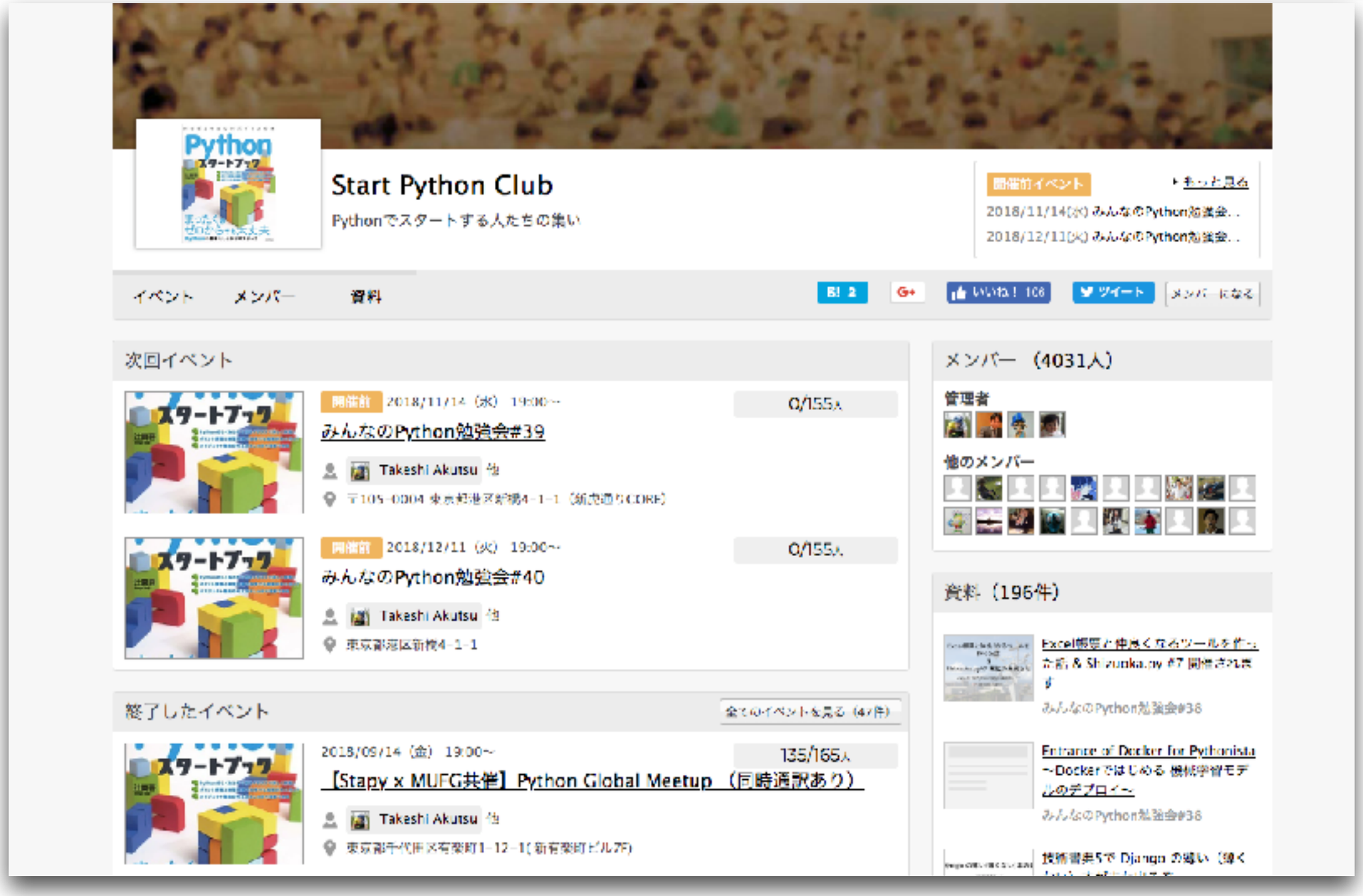
scouty のミッション

「世の中のミスマッチを無くす」

自分のまわりには、自分でも気づいていないたくさんの可能性や偶然性が存在するはずなのに、
人はいつもそれに巡り会えるとは限りません。
そしてその結果、仕事や人材におけるミスマッチに悩む人も少なくはないでしょう。

scoutyは、インターネット上にあふれるデータと最先端の人工知能技術を使って情報と機会を適切にお届けすることで、偶然を必然に変え、世の中のミスマッチをなくしていくことを目指します。 そして、それは結果として、個人の市場価値や生活の質を高め、企業の競争力を高めることにつながると考えています。

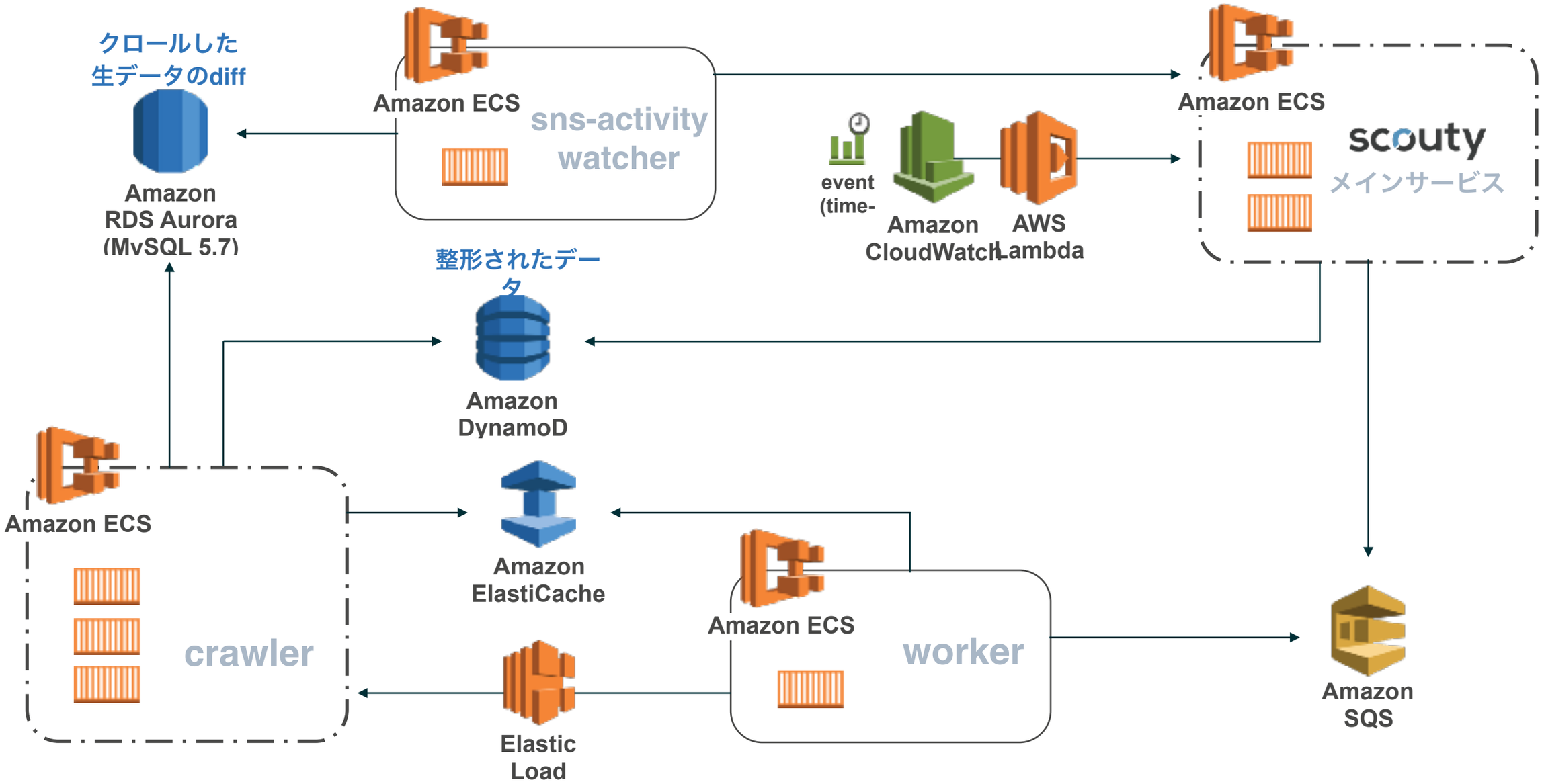
月1で麹町あたりで
100人くらいの勉強会やってました
11月からは新橋



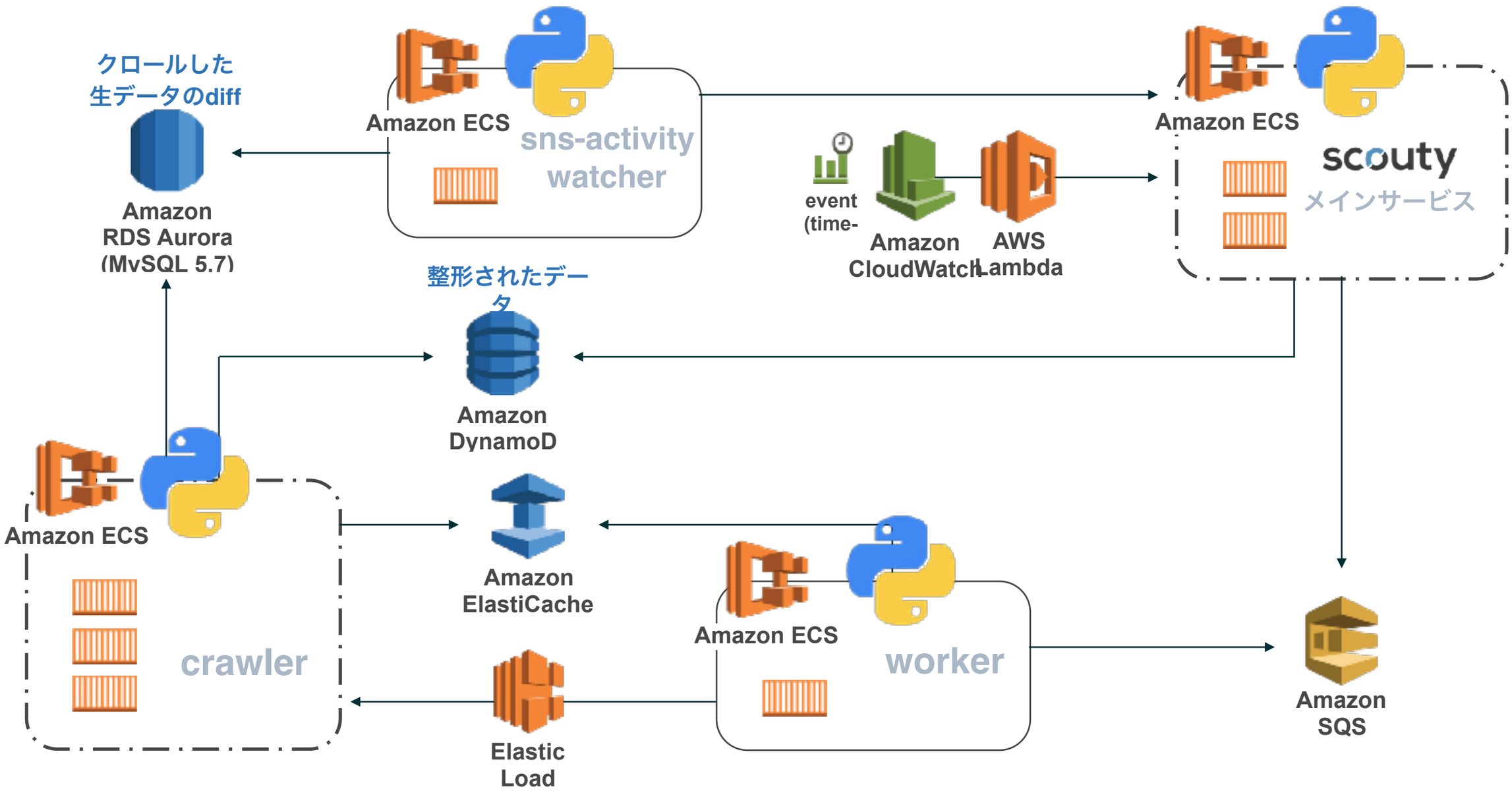
https://startpython.connpass.com/

Pythonと scouty

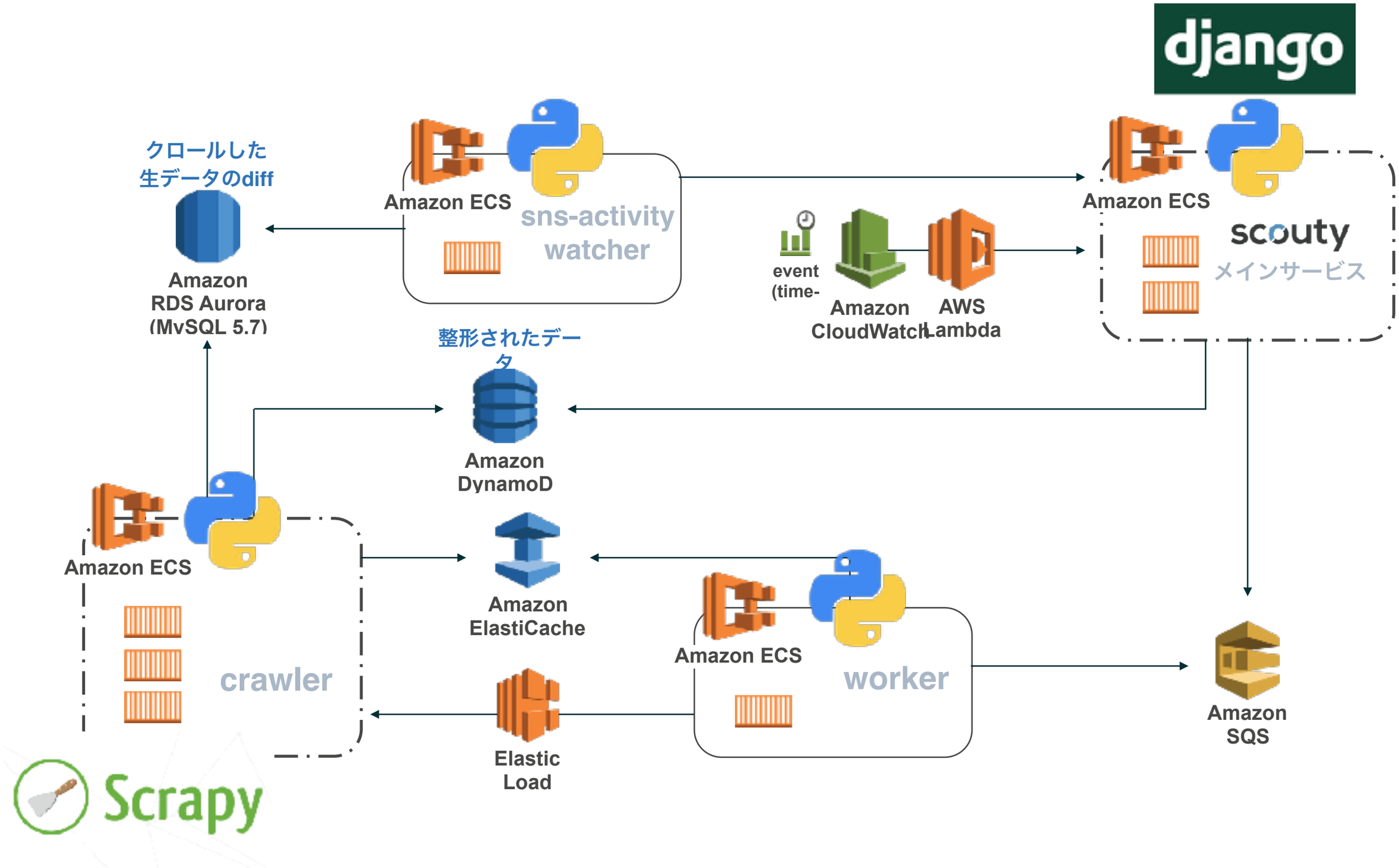
scouty インフラ構成図



scouty インフラ構成図



scouty インフラ構成図



お詫び

2018-9-18 13:30　Talk(45min)　特別会議室

発表言語 : 日本語　資料言語 : 日本語　Level : Beginner

Djangoだってカンバンつくれるもん(Django Channels + Vue)

Djangoのライブラリであるdjango Channelsを使用して、Websocketを使ったアプリケーションの構築についてお話します。Trell
作成した際の話が中心になります。

denzow

scoutyという会社でDBやバックエンドを中心に活動している、Djangoが好きなエンジニアです。

発表詳細

Django Channels 2.1系とVue.js + Vuexを使用して複数ブラウザでリアルタイムに同期するいわゆるカンバンアプリケーションを作
との組み合わせの実践的な内容をコードを含めて紹介します。

* Django Channelsの概要
* Django Channelsの基本概念
  - asgi
  - routing
  - consumer
* 本番でのアプリケーションサーバ選定(not uWSGI, use daphne)

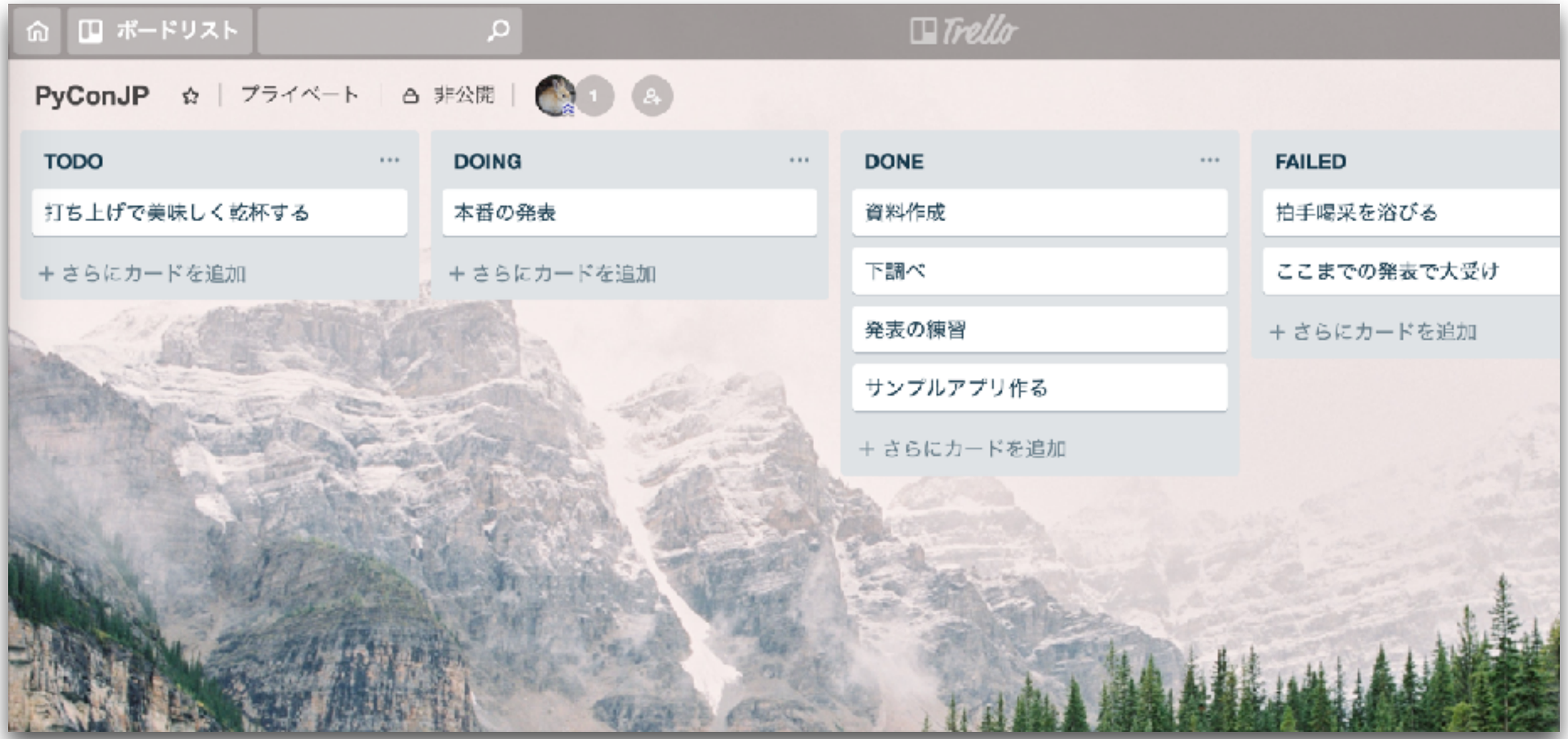
Beginnerの定義読み違えた

Beginner向けにチュートリアル書いてくので許して

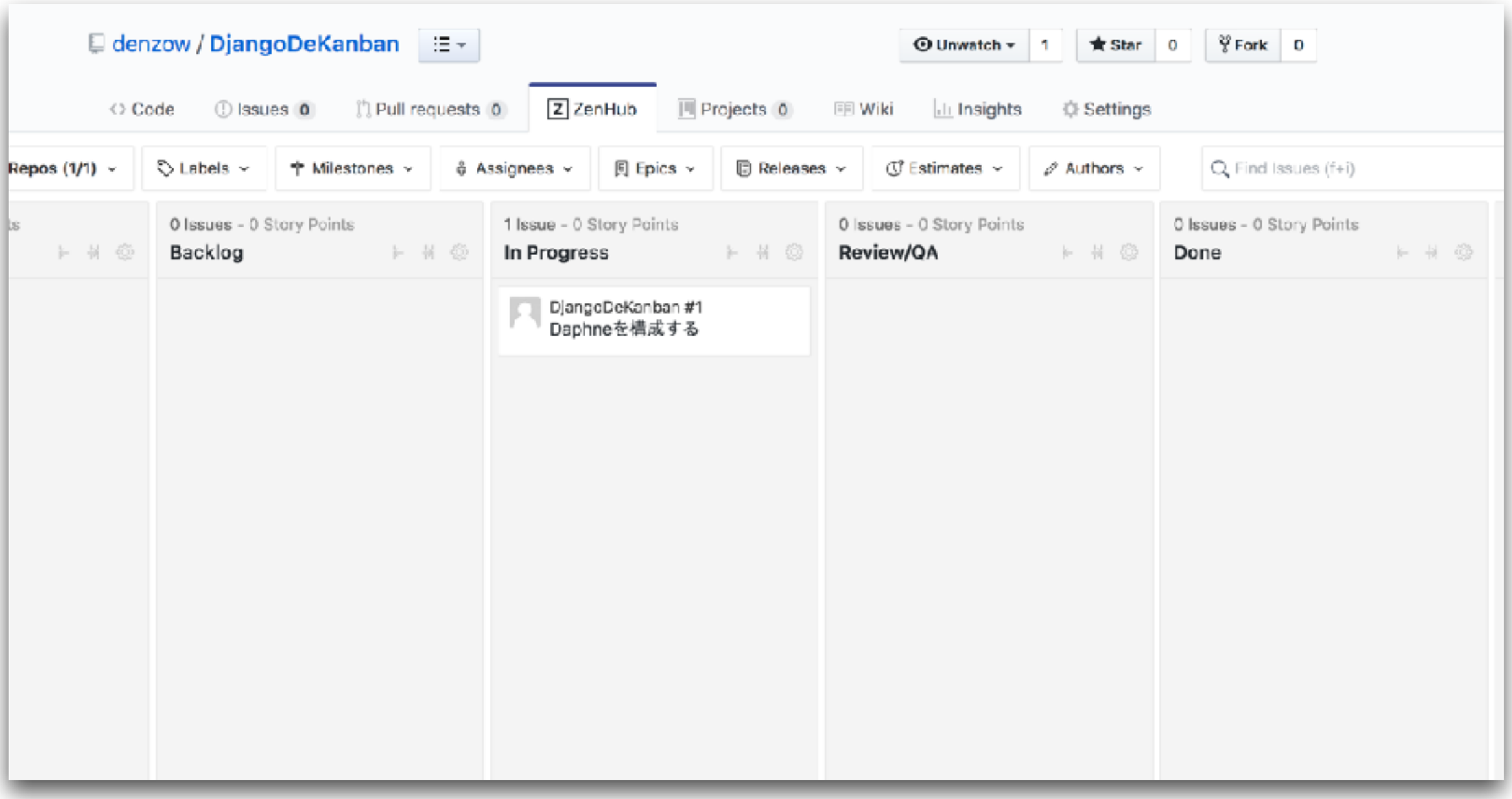
https://qiita.com/denzow/items/046f3c8b9bd8d3378eb4

本題

Trello


ボードリスト
Trello
PyConJP
プライベート
非公開
1
TODO
打ち上げで美味しく乾杯する
＋ さらにカードを追加
DOING
本番の発表
＋ さらにカードを追加
DONE
資料作成
下調べ
発表の練習
サンプルアプリ作る
＋ さらにカードを追加
FAILED
拍手喝采を浴びる
ここまでの発表で大受け
＋ さらにカードを追加

ZenHub


denzow / DjangoDeKanban
Unwatch 1
Star 0
Fork 0
Code
Issues 0
Pull requests 0
ZenHub
Projects 0
Wiki
Insights
Settings
Repos (1/1)
Labels
Milestones
Assignees
Epics
Releases
Estimates
Authors
Find issues (f+i)
0 Issues - 0 Story Points
Backlog
1 Issue - 0 Story Points
In Progress
DjangoDeKanban #1
Daphneを構成する
0 Issues - 0 Story Points
Review/QA
0 Issues - 0 Story Points
Done

カンバンっていいよね

Djangoでカンバンつくりたい

Djangoでカンバンつくろう！

カンバンに必要な機能

ドラッグアンドドロップでの直感的な操作

リアルタイムなデータの反映

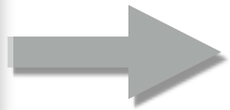
複数人でのコラボレーション

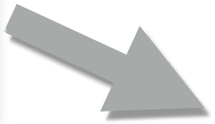
カンバンに必要な機能

ドラッグアンドドロップでの
直感的な操作

リアルタイムなデータの
反映

複数人での
コラボレーション





クライアントサイドの
JS頑張る

WebSocketで
リアルタイム処理をする

カンバンに必要な機能

ドラッグアンドドロップでの
直感的な操作

リアルタイムなデータの
反映

複数人での
コラボレーション

クライアントサイドの
JS頑張る

WebSocketで
リアルタイム処理をする

カンバンに必要な機能

ドラッグアンドドロップでの
直感的な操作

クライアントサイドの
JS頑張る

リアルタイムなデータの
反映

複数人での
コラボレーション

WebSocketで
リアルタイム処理をする

DjangoでWebSocketの何が辛いのか

WSGI とともにデプロイするには

Django の主要なデプロイプラットフォームは、 Web サーバと Web アプリケーション に関して Python の標準である WSGI です。

Django の **startproject** 管理用コマンドはシンプルなデフォルトの WSGI 設定をセットアップします。必要に応じて、あなたのプロジェクトと WSGI 準拠の Web サーバに合わせて微調整することができます。

Django には以下の WSGI サーバのために、手引きとなるドキュメントが用意されています:

- Django を Apache と **mod_wsgi** とともに使うには?
- Django のユーザーデータベースに対する Apache からの認証
- How to use Django with Gunicorn
- How to use Django with uWSGI

WSGI とともにデプロイするには

Django の主要なデプロイプラットフォームは、Web サーバと Web アプリケーション に関して Python の標準である WSGI です。

Django の **startproject** 管理用コマンドはシンプルなデフォルトの WSGI 設定をセットアップします。必要に応じて、あなたのプロジェクトと WSGI 準拠の Web サーバに合わせて微調整することができます。

Django には以下の WSGI サーバのために、手引きとなるドキュメントが用意されています:

- Django を Apache と **mod_wsgi** とともに使うには?
- Django のユーザーデータベースに対する Apache からの認証
- How to use Django with Gunicorn
- How to use Django with uWSGI

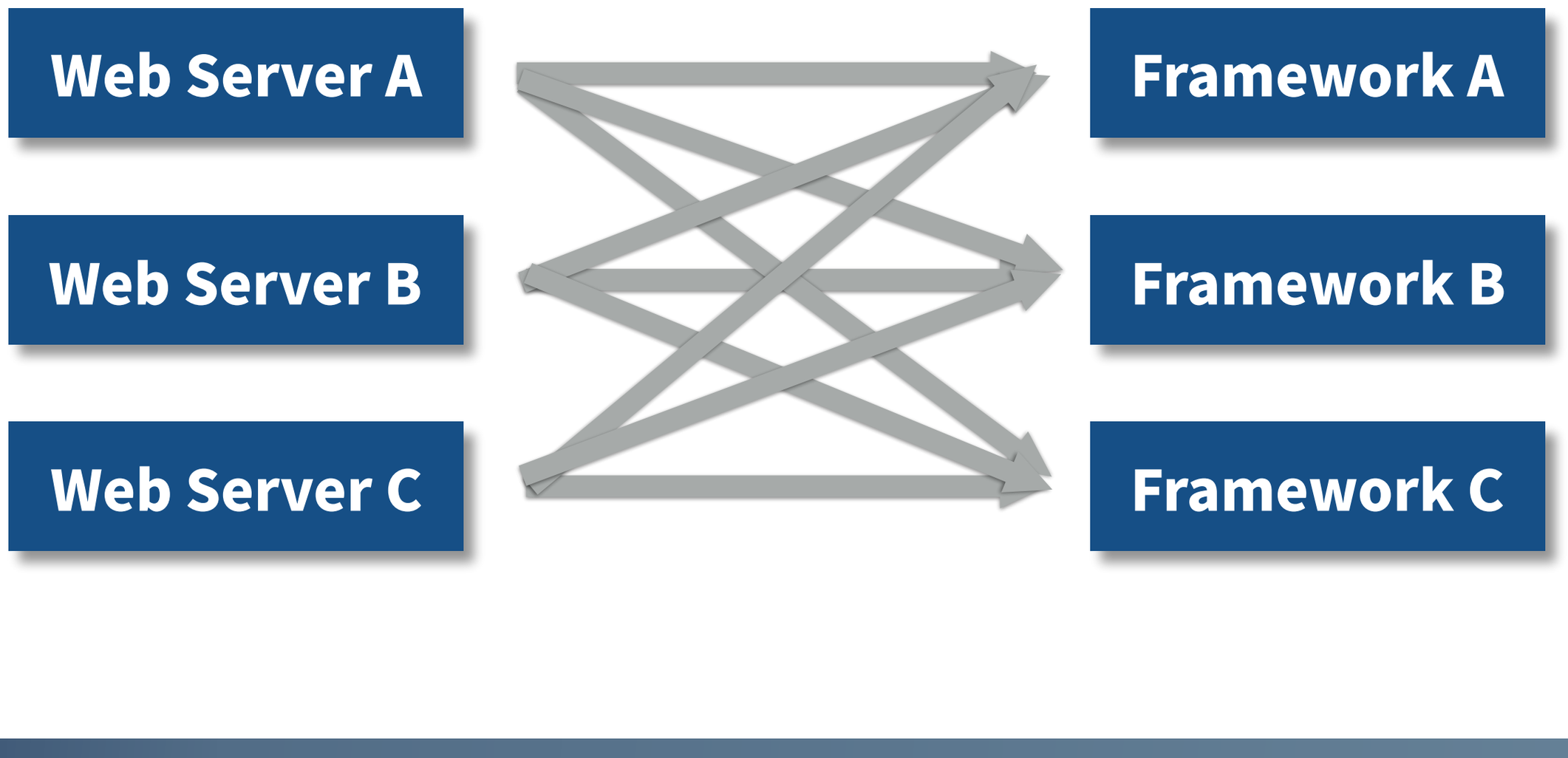
Djangoは普通WSGIで動かす

WSGI?

- アプリケーションとフレームワーク間の規格
- PEP333(Python2)/ PEP3333(Python3)

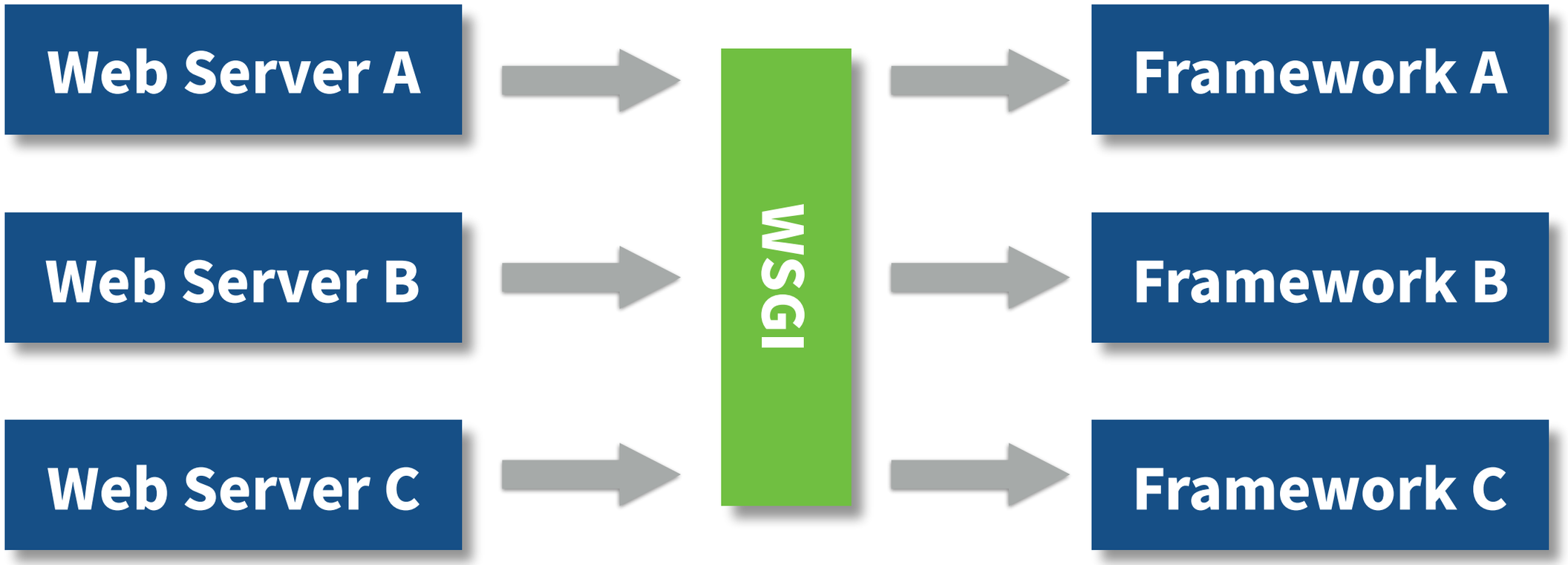
WSGI?

- アプリケーションとフレームワーク間の規格
- PEP333(Python2)/ PEP3333(Python3)



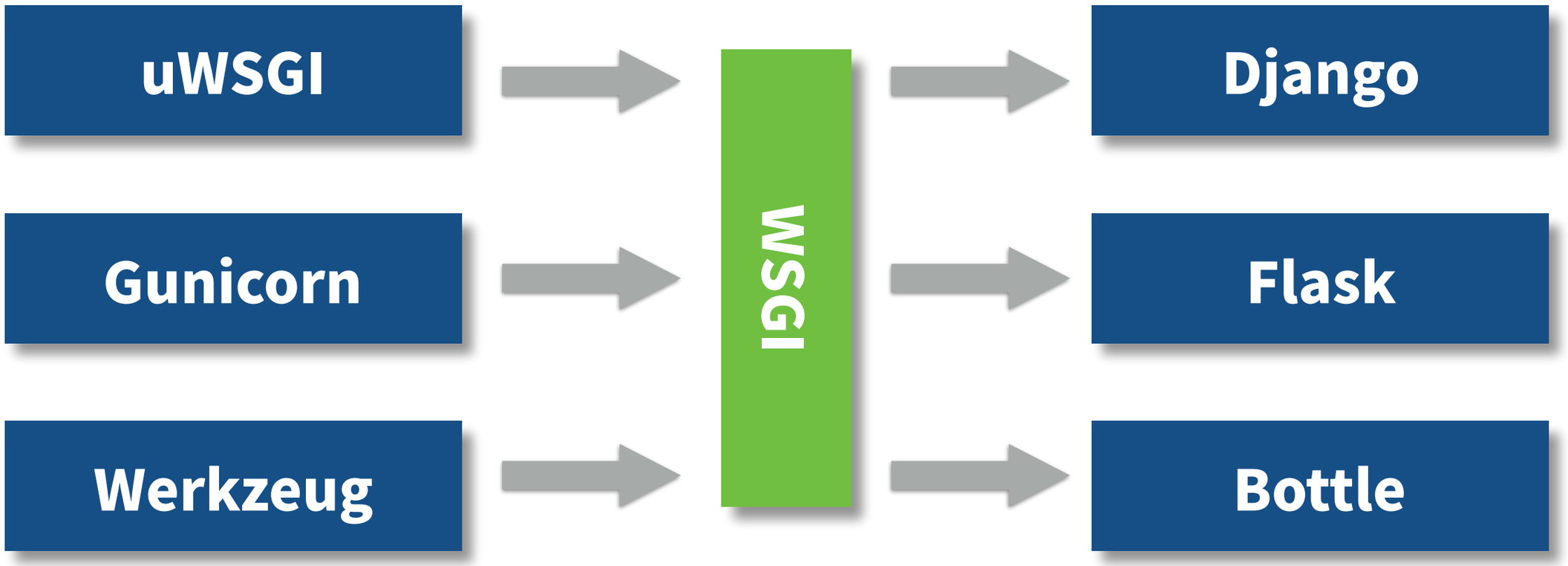
WSGI?

- アプリケーションとフレームワーク間の規格
- PEP333(Python2)/ PEP3333(Python3)



WSGI?

- アプリケーションとフレームワーク間の規格
- PEP333(Python2)/ PEP3333(Python3)



WSGI?

PEP 333の提案日は2003-12-07

PEP 333 -- Python Web Server Gateway Interface v1.0

PEP:	333
Title:	Python Web Server Gateway Interface v1.0
Author:	Phillip J. Eby <pje at telecommunity.com>
Discussions-To:	Python Web-SIG <web-sig at python.org>
Status:	Final
Type:	Informational
Created:	07-Dec-2003
Post-History:	07-Dec-2003, 08-Aug-2004, 20-Aug-2004, 27-Aug-2004, 27-Sep-2010
Superseded-By:	3333

https://www.python.org/dev/peps/pep-0333/

WSGI?

websocketが生まれたのは2011頃

歴史的経緯 [編集]

プロトコルは2011年5月の完成を目標に進められていたが[4]、その期日を過ぎても仕様の改訂は続けられ、2011年7月11日に最終草案のdraft-ietf-hybi-thewebsocketprotocol-10が勧告[5]されたが、さらにその後も改訂は続き、2011年9月30日に draft-ietf-hybi-thewebsocketprotocol-17がリリースされ、それが2011年12月11日にRFC 6455のproposed standard（標準化への提唱）となった。

2010年11月26日にdraft-ietf-hybi-thewebsocketprotocol-03やそれ以前のWebSocketのプロトコルにセキュリティホールが発見され[6]、2010年12月に、一時的に、Firefox 4とOpera 11のWebSocketが無効になり、Chromeはプロトコル改訂よりも先に攻撃コードが出た場合は無効にするとしていた。Opera 11は opera:config#Enable%20WebSockets を開き、設定を有効にすると利用可能。その後、2011年1月11日にdraft-ietf-hybi-thewebsocketprotocol-04が発表され、サーバにアップロード通信する際はプロキシを混乱させないために、通信内容をXORでマスキングさせる方法となった。2011年8月16日に再度WebSocketに対応させた、Firefox 6 がリリースされたが、まだ、仕様の改訂が続くという理由から、Firefox 10までは、MozWebSocketと頭にMozがつく形となった[7]。Firefox for Mobileは7から対応[8]。

ウェブブラウザで動作するJavaScript用のAPIの策定は、当初W3Cで行われていた。その後、HTML5とともにWHATWGに移ることとなった。

https://ja.wikipedia.org/wiki/WebSocket

NO WebSocket

WSGIはWebSocketより前に生まれた

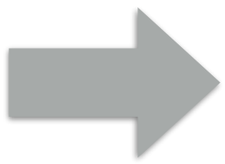
- サーバがwsgiのエンドポイントのcallableを呼び出す
- Callableはenvironとstart_responseを受け取る
- start_responseでステータスやヘッダを送信
- レスポンス本体をiterableな戻り値として返送

```
def application(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/plain')])
    return [b'Hello World\n']
```

WSGIはWebSocketより前に生まれた

- サーバがwsgiのエンドポイントのcallableを呼び出す
- Callableはenvironとstart_responseを受け取る
- start_responseでステータスやヘッダを送信
- レスポンス本体をiterableな戻り値として返送

```
def application(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/plain')])
    return [b'Hello World\n']
```



1リクエスト毎にCallbleが呼ばれ終了する

Websocketのライフサイクル

- 初回接続時にセッションを確立
- セッションは切断処理がされるまで確立されたまま
- 確立したセッションでメッセージの送受信をする
- サーバからもプッシュ形式でメッセージが届く

Websocketのライフサイクル

- 初回接続時にセッションを確立
- セッションは切断処理がされるまで確立されたまま
- 確立したセッションでメッセージの送受信をする
- サーバからもプッシュ形式でメッセージが届く

Websocket の
ライフサイクルは長い

WSGIでWebsocket

つらい

ASGI誕生

Asynchronous Server Gateway Interface

https://asgi.readthedocs.io/en/latest/introduction.html

What's wrong with WSGI?

You may ask "why not just upgrade WSGI"? This has been asked many times over the years, and the problem usually ends up being that WSGI's single-callable interface just isn't suitable for more involved Web protocols like WebSocket.

WSGI applications are a single, synchronous callable that takes a request and returns a response; this doesn't allow for long-lived connections, like you get with long-poll HTTP or WebSocket connections.

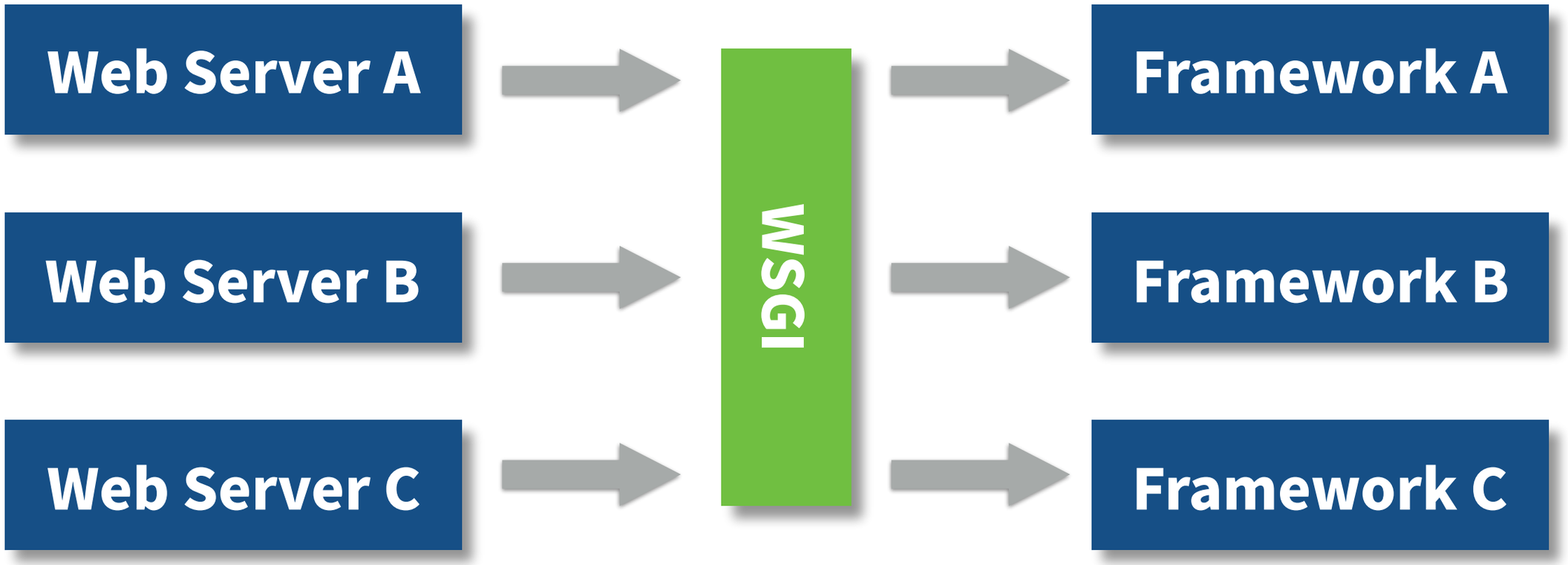
Even if we made this callable asynchronous, it still only has a single path to provide a request, so protocols that have multiple incoming events (like receiving WebSocket frames) can't trigger this.

ASGI?

- WSGIで対応が難しいWebSocket等のための規格
- WSGIのスーパーセットになるようにする

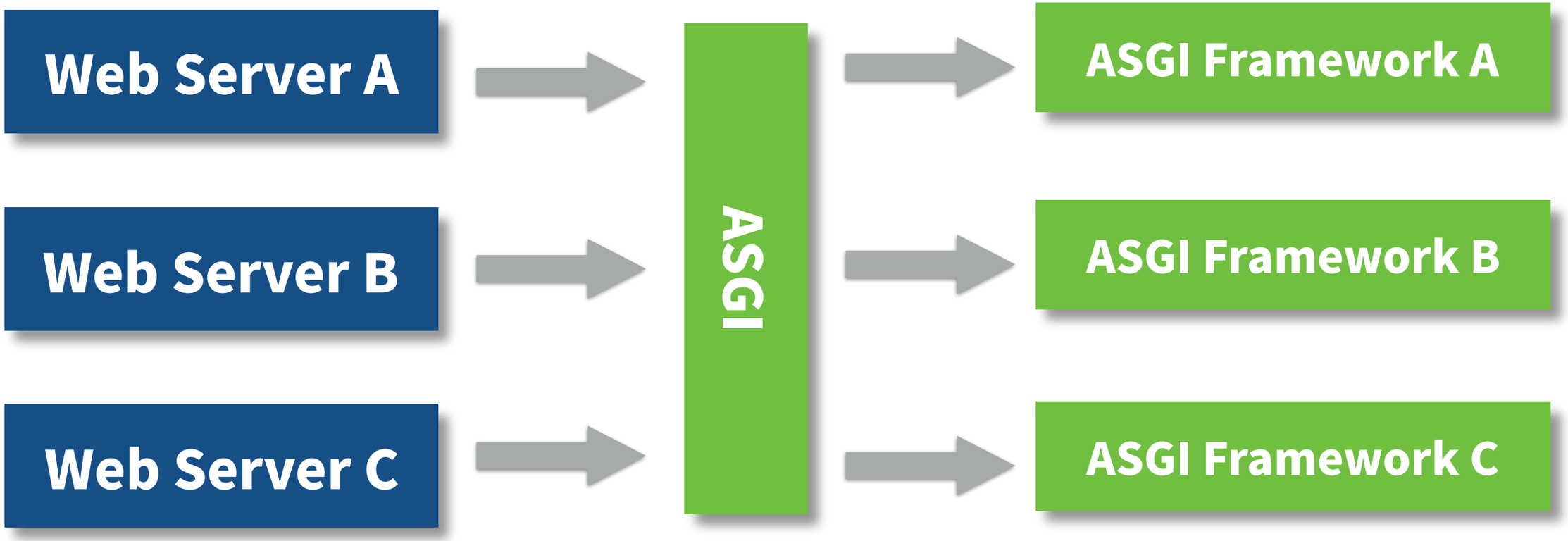
ASGI?

- WSGIで対応が難しいWebSocket等のための規格
- WSGIのスーパーセットになるようにする



ASGI?

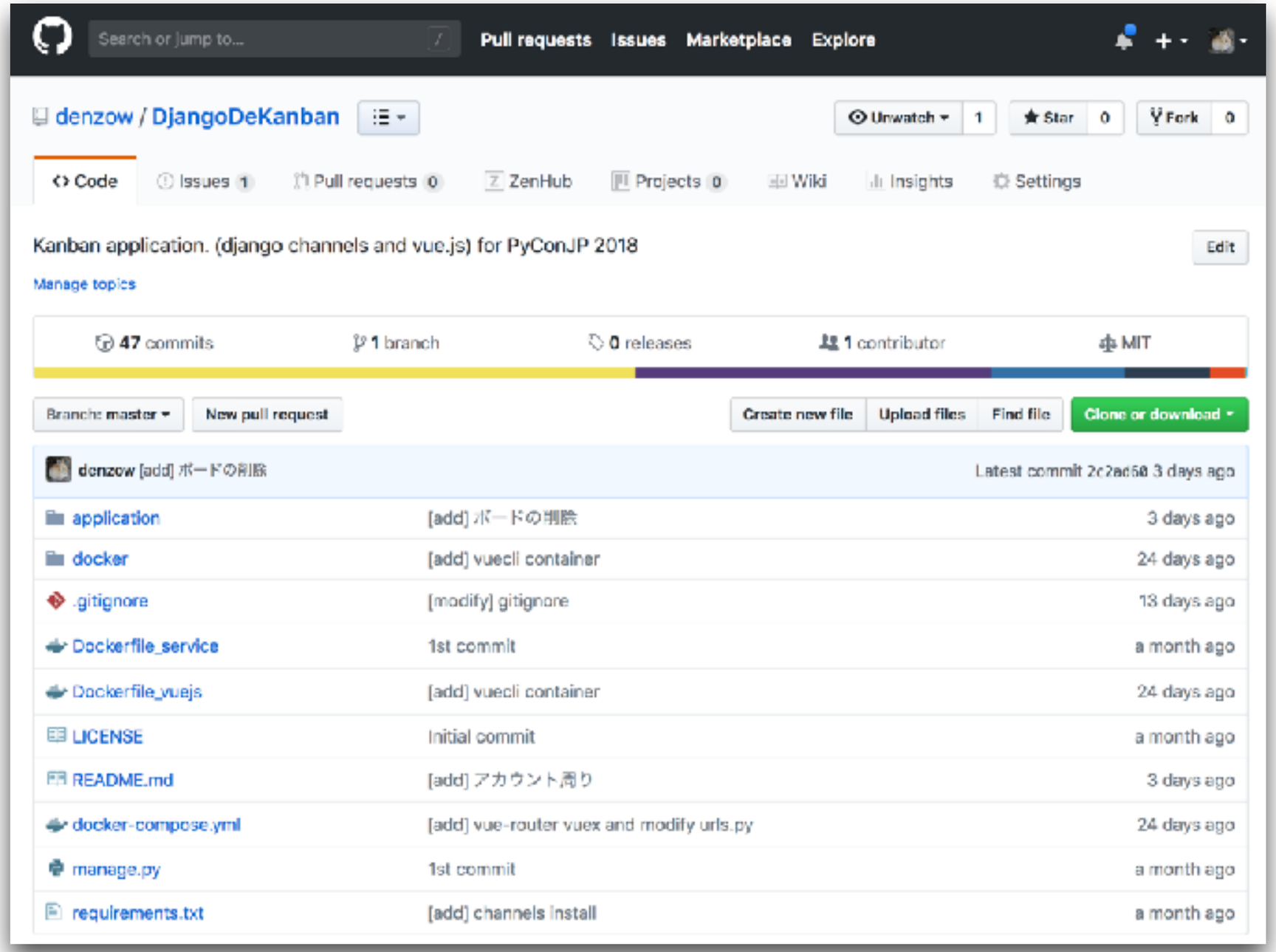
- WSGIで対応が難しいWebSocket等のための規格
- WSGIのスーパーセットになるようにする



ひろがる Python

実際にカンバン
つくってみた

DjangoDeKanban



https://github.com/denzow/DjangoDeKanban

DjangoDeKanban

- ログイン管理
- ボード作成
- パイプライン(リスト)追加
- パイプライン(リスト)の並び替え
- カード追加
- カード並び替え
- キーワードでのカード絞り込み

DjangoDeKanban

- Django 2.1
- Django Channels 2.1

- Vue/Vuex
- VueNativeWebSocket
- VueDraggable

- Django 2.1
- Django Channels 2.1

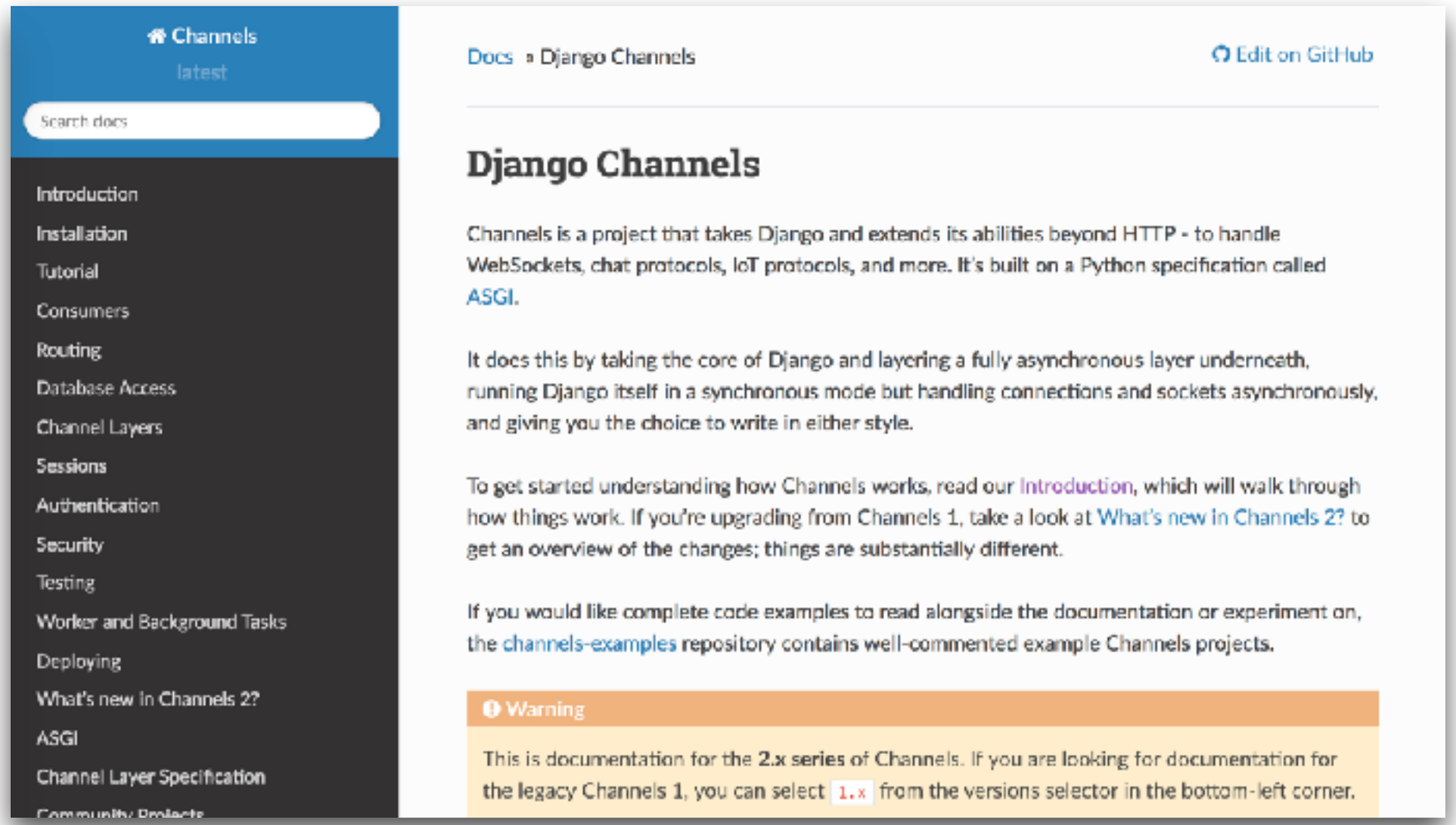
- Vue/Vuex
- VueNativeWebSocket
- VueDraggable

Django Channels 2

Django Channels 1.xの話はしません

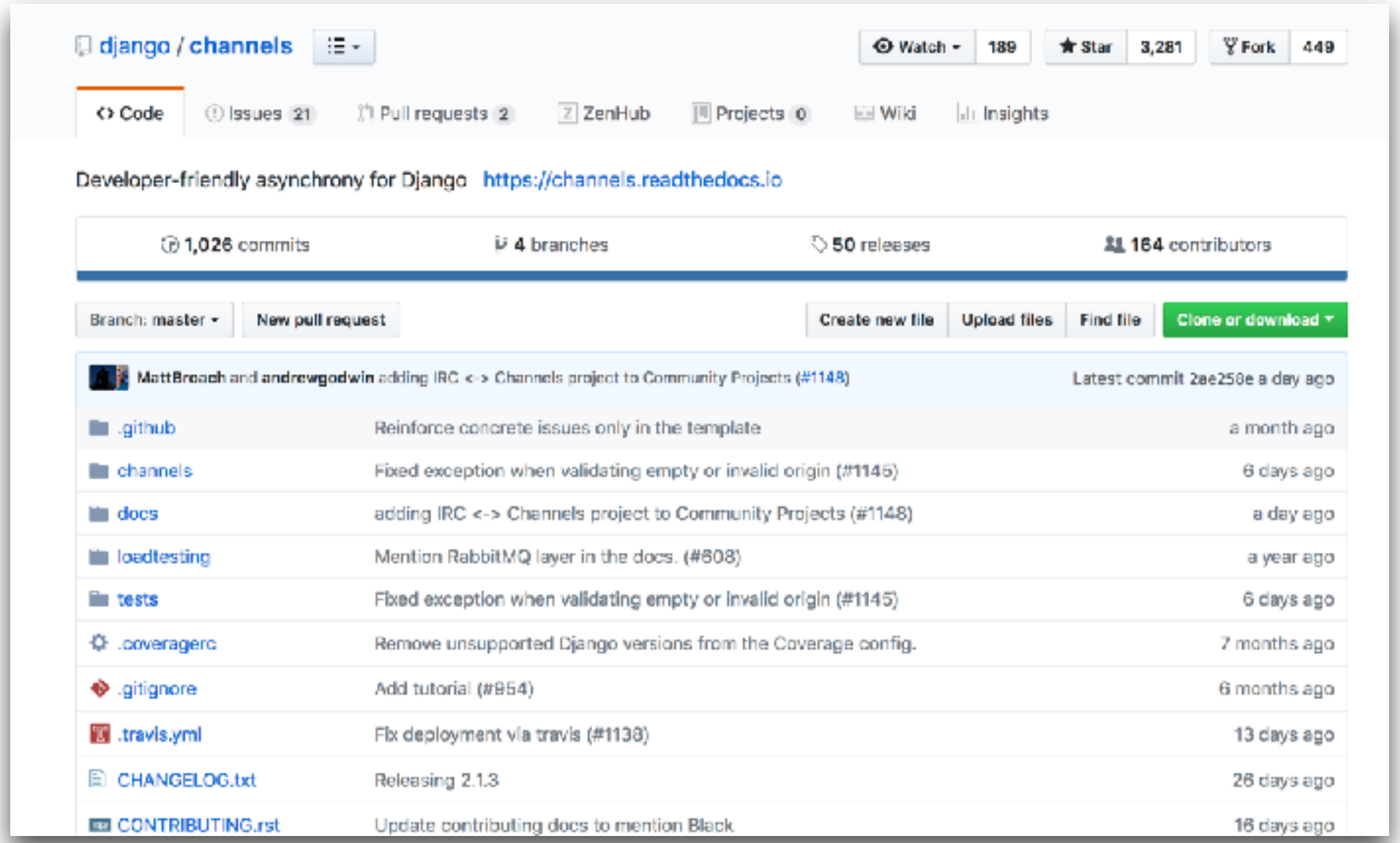
2.xと1.xはPython 2とPython 3くらい違います

Django Channelsは DjangoをASGI対応にするライブラリ



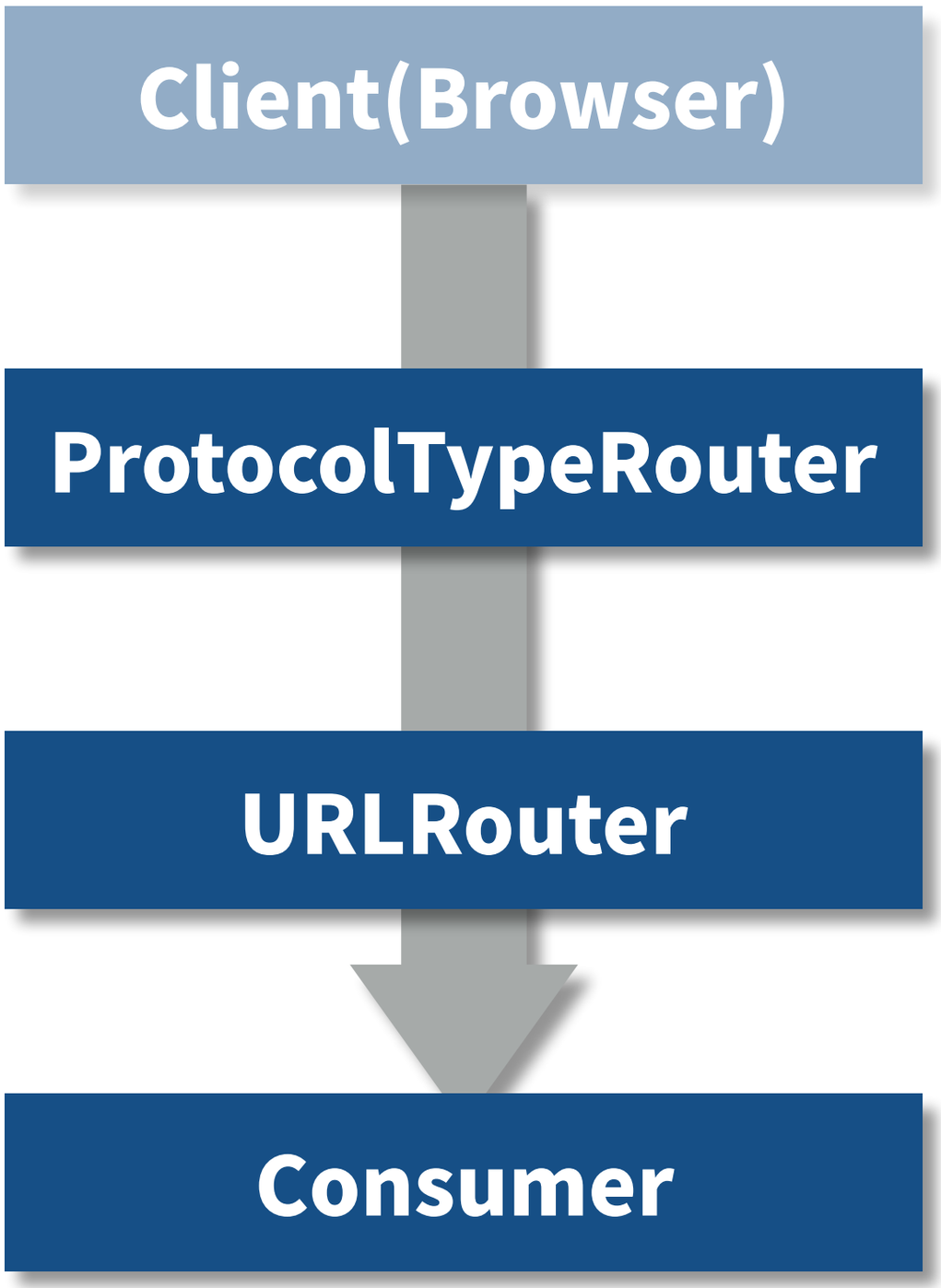
https://channels.readthedocs.io/en/latest/

Djangoグループが開発している



https://github.com/django/channels

Django Channelsでの流れ



Django Channelsでの流れ(ProtocolTypeRouter)

application/settings/base.py

```
# ASGIの起点を指定
ASGI_APPLICATION = 'views.routing.application'
```

application/views/routing.py

```
from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter

# Childのルーティングルールに分割
from .ws.routing import urlpatterns

application = ProtocolTypeRouter({
    # (http->django views is added by default)
    'websocket': AuthMiddlewareStack(
        URLRouter(
            urlpatterns
        )
    ),
})
```

プロトコル毎の
処理振り分け

Django Channelsでの流れ(ProtocolTypeRouter)

application/settings/base.py

```
# ASGIの起点を指定
ASGI_APPLICATION = 'views.routing.application'
```

application/views/routing.py

```
from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter

# Childのルーティングルールに分割
from .ws.routing import urlpatterns

application = ProtocolTypeRouter({
    # (http->django views is added by default)
    'websocket': AuthMiddlewareStack(
        URLRouter(
            urlpatterns
        )
    ),
})
```

httpは書かなくても
勝手にurls.pyをもとに
したものが追加される

Websocketの場合

Django Channelsでの流れ(ProtocolTypeRouter)

application/settings/base.py

```
# ASGIの起点を指定
ASGI_APPLICATION = 'views.routing.application'
```

application/views/routing.py

```
from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter

# Childのルーティングルールに分割
from .ws.routing import urlpatterns

application = ProtocolTypeRouter({
    # (http->django views is added by default)
    'websocket': AuthMiddlewareStack(
        URLRouter(
            urlpatterns
        )
    ),
})
```

Djangoの認証と同じものを使えるようにする

Django Channelsでの流れ(URLRouter)

application/settings/base.py

```
# ASGIの起点を指定
ASGI_APPLICATION = 'views.routing.application'
```

application/views/routing.py

```
from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter

# Childのルーティングルールに分割
from .ws.routing import urlpatterns

application = ProtocolTypeRouter({
    # (http->django views is added by
    'websocket': AuthMiddlewareSta
        URLRouter(
            urlpatterns
        )
    ),
})
```

どのようなURLにアクセス
してきたかでの振り分け
(別に直接書いてもいい)

Django Channelsでの流れ(URLRouter)

基本的にはDjangoのurls.pyと同じ

application/views/ws/routing.py

```
from django.urls import path
from .consumers import kanban_consumer

urlpatterns = [
    path('ws/boards/<int:board_id>', kanban_consumer.KanbanConsumer)
]
```

Django Channelsでの流れ(URLRouter)

基本的にはDjangoのurls.pyと同じ

application/views/ws/routing.py

```
from django.urls import path
from .consumers import kanban_consumer

urlpatterns = [
    path('ws/boards/<int:board_id>' kanban_consumer.KanbanConsumer)
]
```

ws/boards/1 のような
URLにマッチさせる

Django Channelsでの流れ(URLRouter)

基本的にはDjangoのurls.pyと同じ

application/views/ws/routing.py

```
from django.urls import path
from .consumers import kanban_consumer

urlpatterns = [
    path('ws/boards/<int:board_id>', kanban_consumer.KanbanConsumer)
]
```

該当したときの処理

Django Channelsでの流れ(Consumer)

Viewsみたいなもん

application/views/ws/consumers/kanban_consumer.py

```
:
:
class KanbanConsumer(BaseJsonConsumer):
    :
    async def connect(self):
        if not self.scope['user'].is_authenticated:
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
        :
```

Django Channelsでの流れ(Consumer)

Viewsみたいなもん

```
application/views/ws/consumers/kanban_consumer.py
:
:
class KanbanConsumer(BaseJsonConsumer):
    :
    async def connect(self):
        if not self.scope['user'].is_authenticated:
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
        :
```

AuthMiddlewareStackを使うと、scope['user']にいれてくれる

Django Channelsでの流れ(Consumer)

Viewsみたいなもん

```
application/views/ws/consumers/kanban_consumer.py
:
:
class KanbanConsumer(BaseJsonConsumer):
    :
    async def connect(self):
        if not self.scope['user'].is_authenticated:
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
        :
```

URLで <int:board_id> と
定義してた部分にマッチした情報が
取り出せる

Django Channelsでの流れ(Consumer)

application/views/ws/routing.py

```
from django.urls import path
from .consumers import kanban_consumer

urlpatterns = [
    path('ws/boards/<int:board_id>' kanban_consumer.KanbanConsumer)
]
```

これ

Django Channelsでの流れ(Consumer)

Viewsみたいなもん

```
application/views/ws/consumers/kanban_consumer.py
:
:
class KanbanConsumer(BaseJsonConsumer):
    :
    async def connect(self):
        if not self.scope['user'].is_authenticated:
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
        :
```

URLで <int:board_id> と
定義してた部分にマッチした情報が
取り出せる

Django Channelsでの流れ(Consumer)

Viewsみたいなもん

application/views/ws/consumers/kanban_consumer.py

```
:
:
class KanbanConsumer(BaseJsonConsumer):
    :
    async def connect(self):
        if not self.scope['user'].is_authenticated:
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
        :
```

ChannelLayerのgroup_addを
実行し、他のConsumerとの
通信ができるようにする

Django Channelsでの流れ(Consumer)

Viewsみたいなもん

application/views/ws/consumers/kanban_consumer.py

```
:
:
class KanbanConsumer(BaseJsonConsumer):
    :
    async def connect(self):
        if not self.scope['user'].is_auth
            await self.close()
            return
        self.user = self.scope['user']
        self.board_id = self.scope['url_route']['kwargs']['board_id']
        self.room_group_name = self.user.username

        await self.channel_layer.group_add(
            self.room_group_name,
            self.channel_name
        )
        await self.accept()
        :
```

接続を拒否する

接続を受け入れる

もうちょっとConsumer


```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        # accept or close
        if YourCondition:
            await self.accept()
        else:
            await self.close()

    async def receive_json(self, content, **kwargs):
        # receive message
        print(content)

        # echo back
        await self.send_json(content)
```

もうちょっとConsumer


```python
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer)

    async def connect(self):
        # accept or close
        if YourCondition:
            await self.accept()
        else:
            await self.close()

    async def receive_json(self, content, **kwargs):
        # receive message
        print(content)

        # echo back
        await self.send_json(content)
```

(Async)?JsonWebsocketConsumerを
継承して実装する

もうちょっとConsumer


```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        # accept or close
        if YourCondition:
            await self.accept()
        else:
            await self.close()

    async def receive_json(self, content, **kwargs):
        # receive message
        print(content)

        # echo back
        await self.send_json(content)
```

初回接続時に
呼ばれる
Accept or close を
呼び出す

もうちょっとConsumer


```python
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        # accept or close
        if YourCondition:
            await self.accept()
        else:
            await self.close()

    async def receive_json(self, content, **kwargs):
        # receive message
        print(content)

        # echo back
        await self.send_json(content)
```

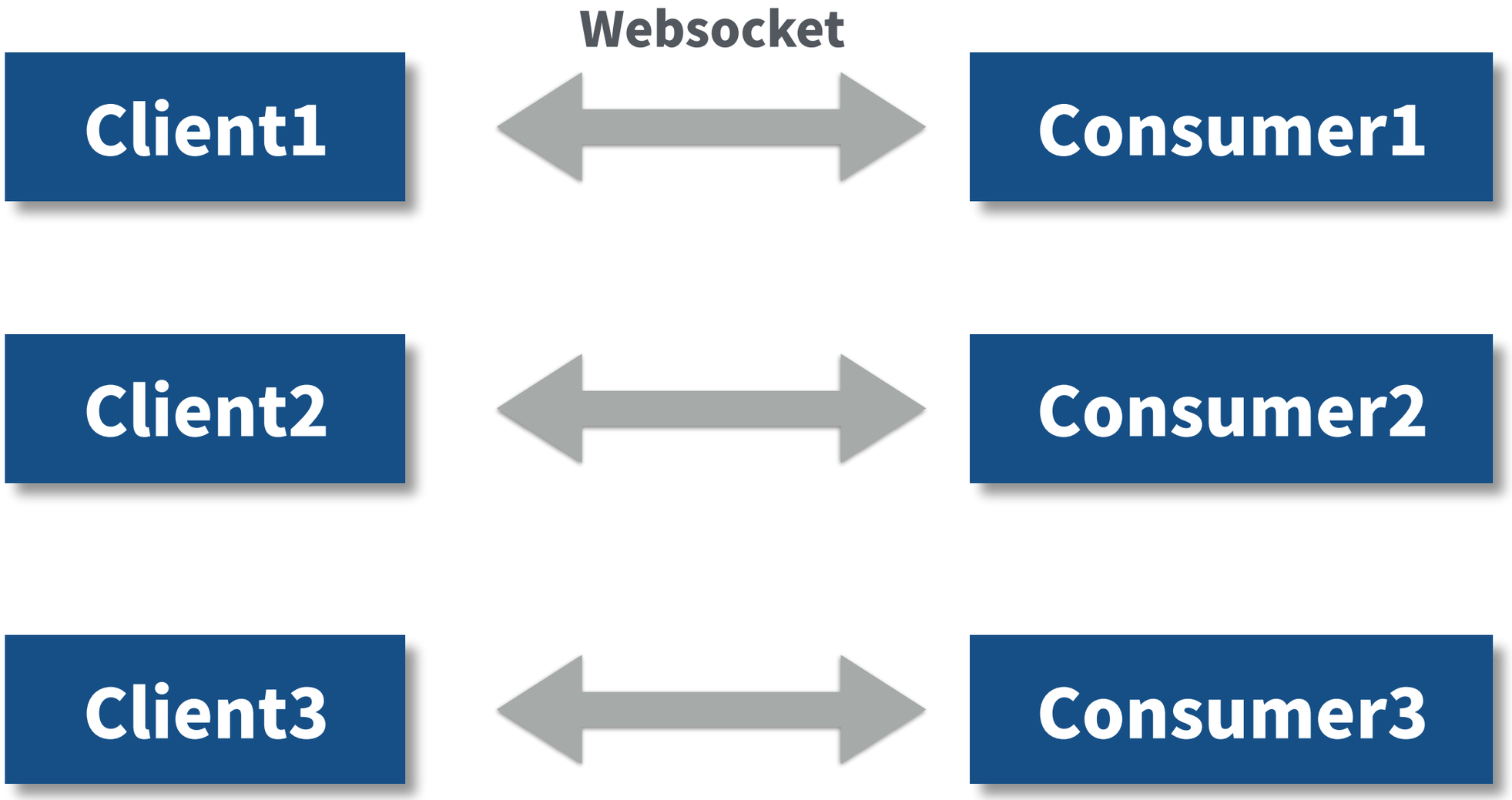
クライアントからの
メッセージ受信時に呼ばれる

もうちょっとConsumer

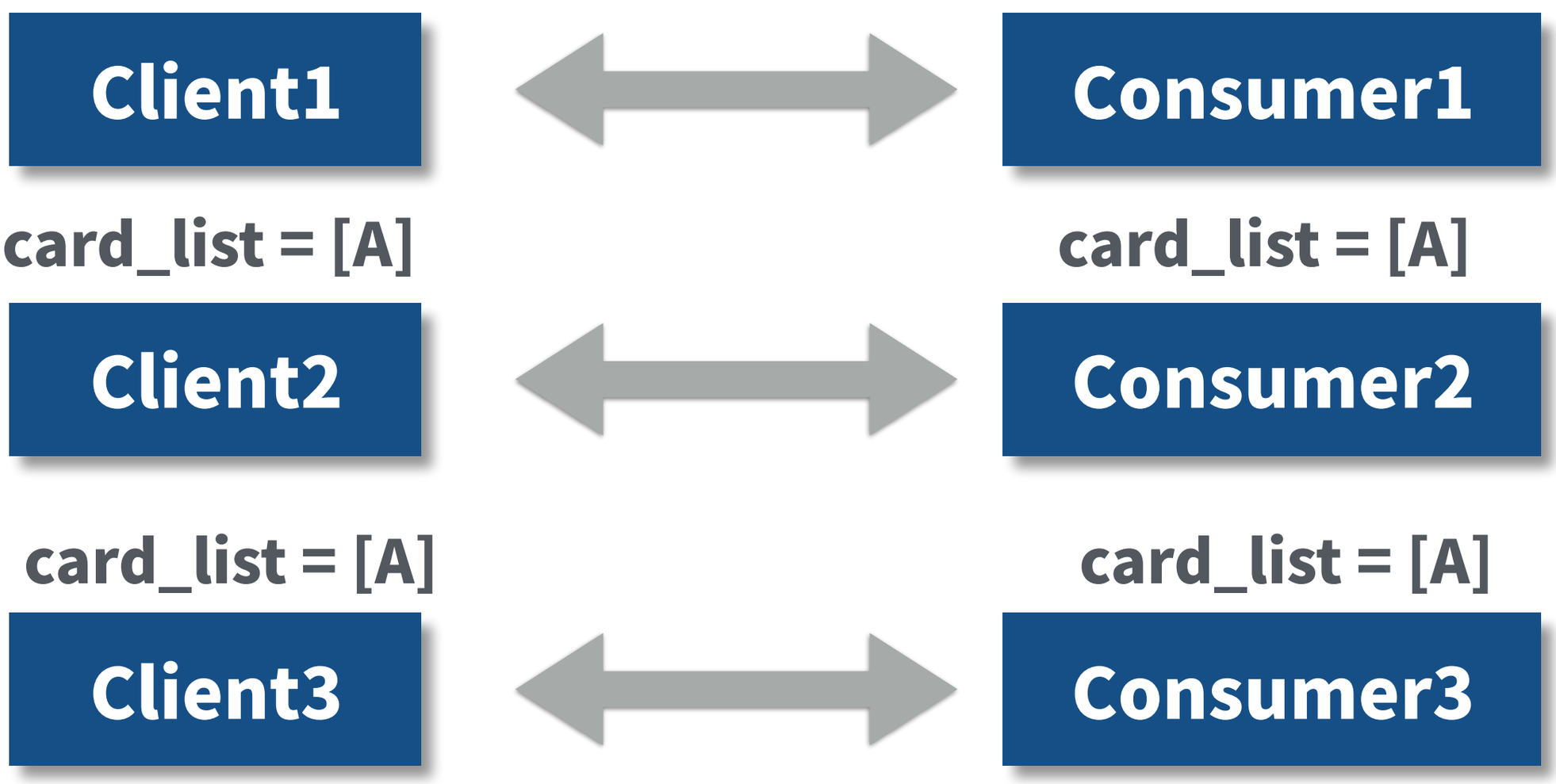

```python
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        # accept or close
        if YourCondition:
            await self.accept()
        else:
            await self.close()

    async def receive_json(self, content, **kwargs):
        # receive message
        print(content)

        # echo back
        await self.send_json(content)
```

クライアントへ
メッセージを返送する

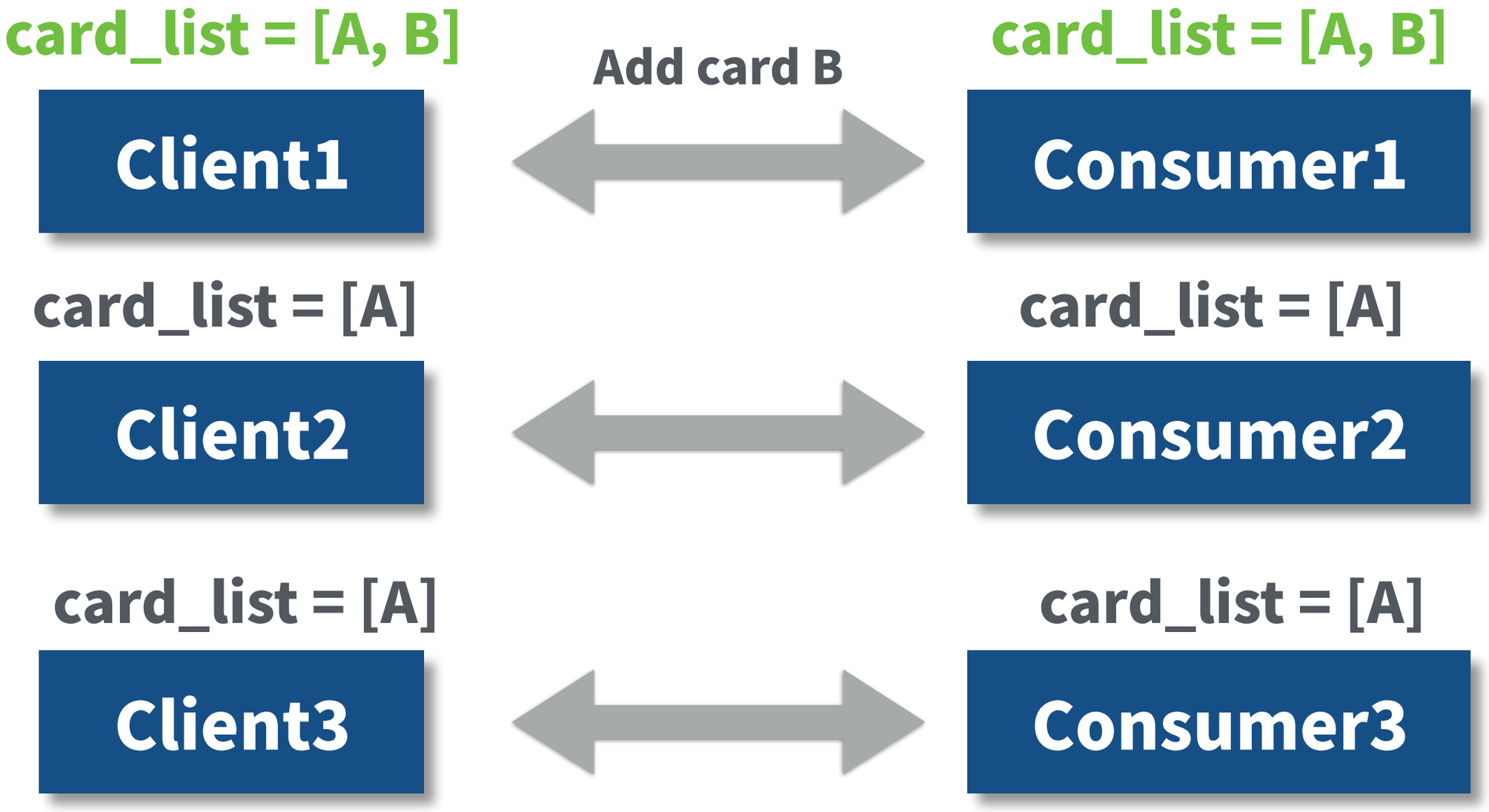
ClientとConsumer間はWebsocketで送受信ができるようになった



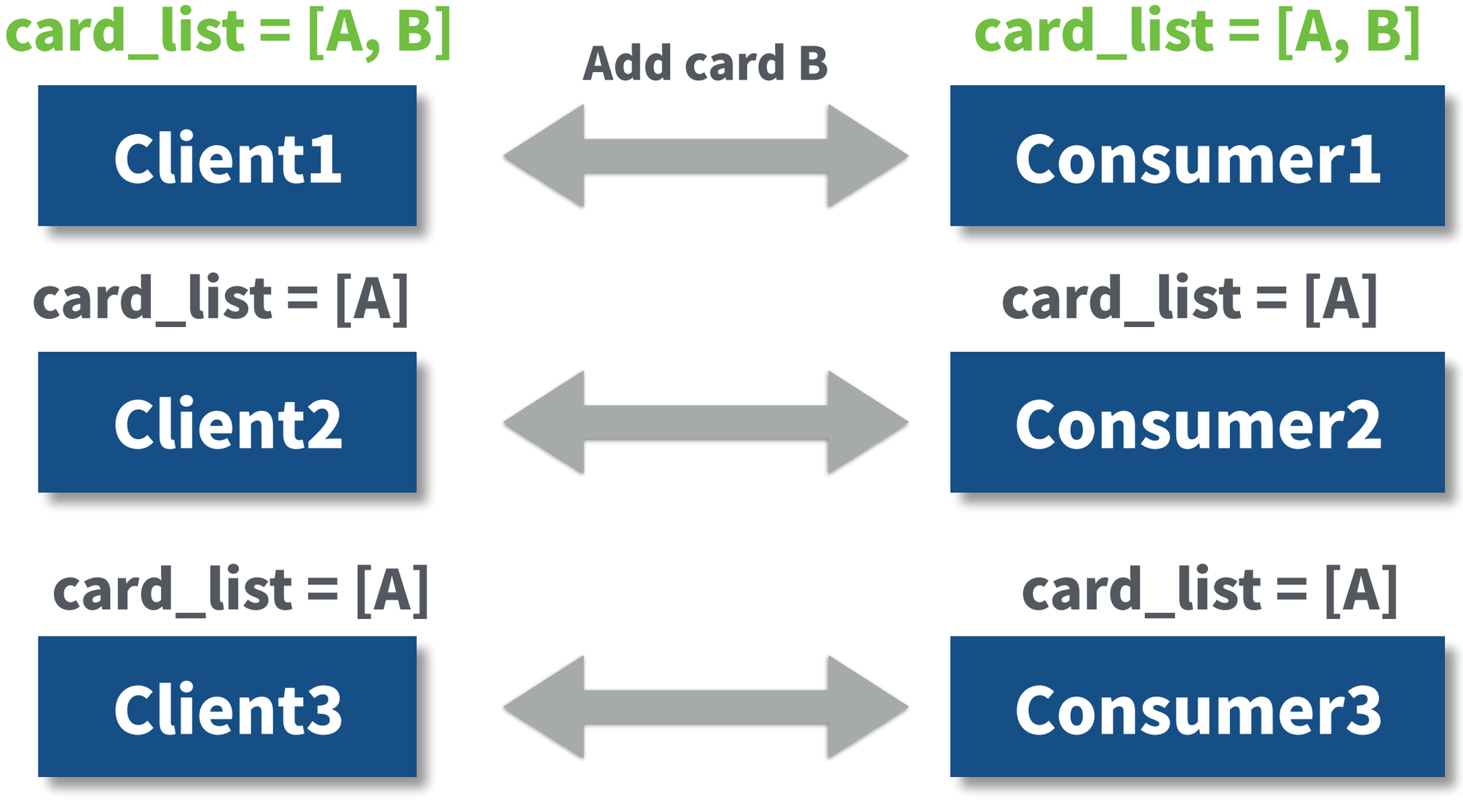
Consumer1の変更をClient2,3に伝えるには？



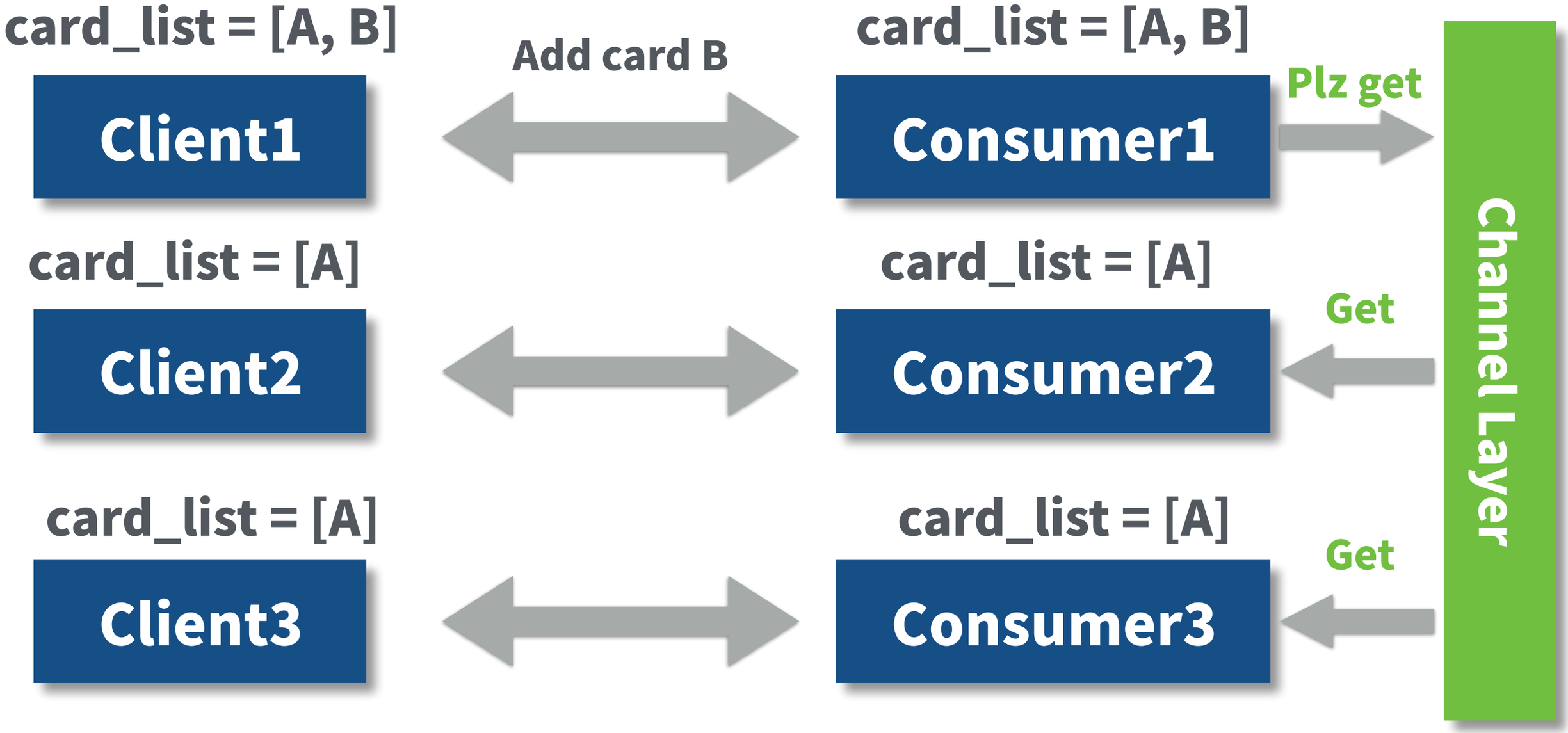
Consumer1の変更をClient2,3に伝えるには？



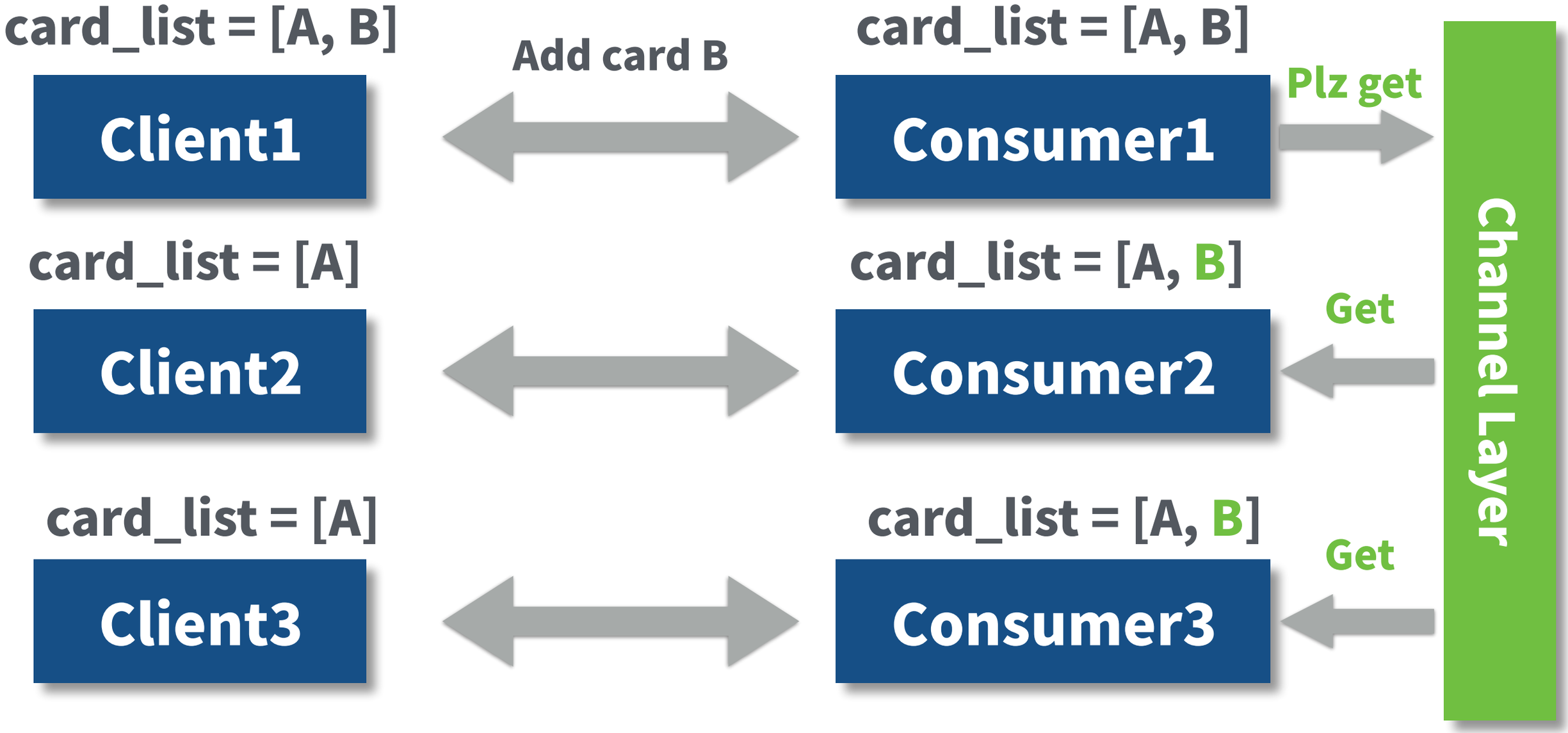
Consumer1の変更をClient2,3に伝えるには？



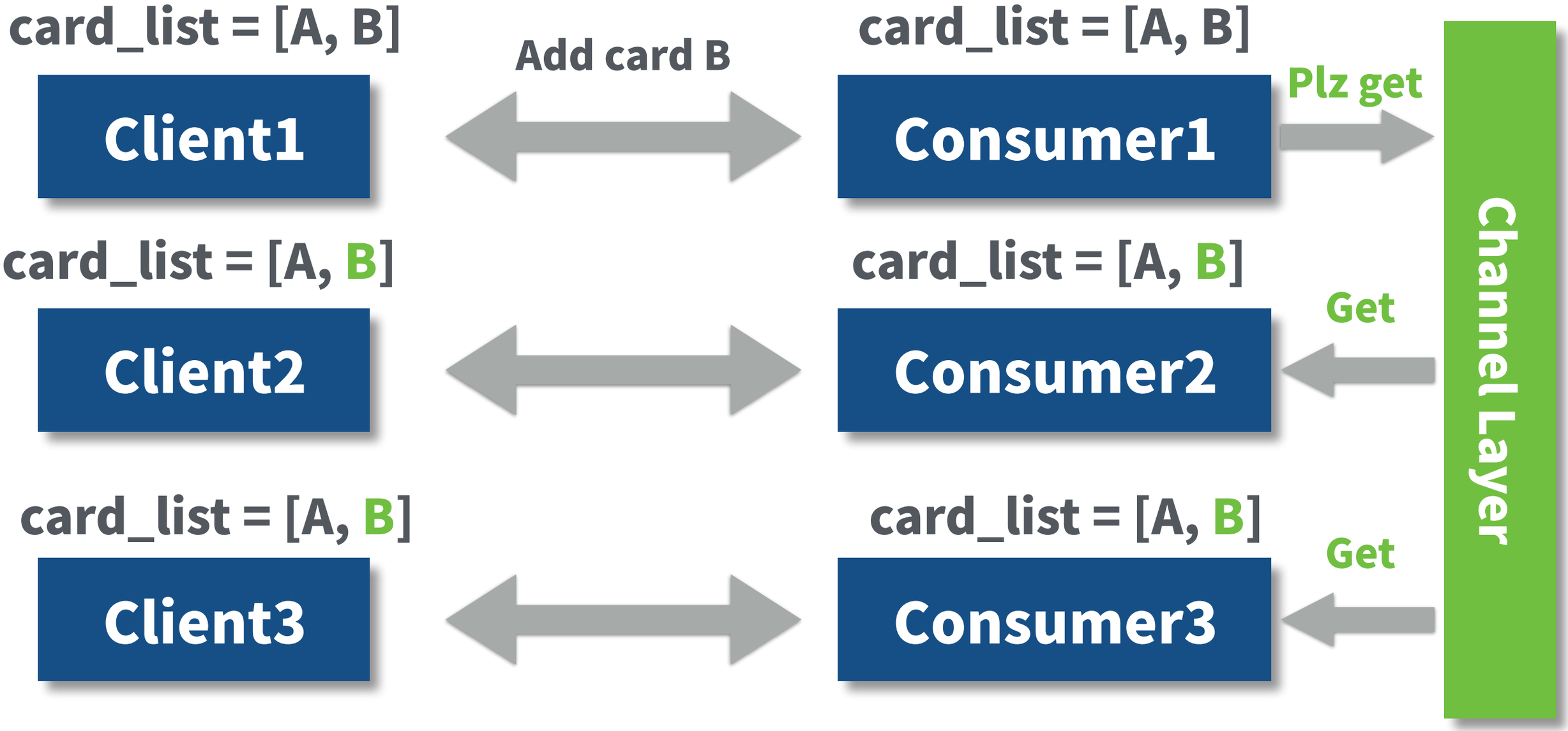
Consumer1の変更をClient2,3に伝えるには？



Consumer1の変更をClient2,3に伝えるには？



Consumer1の変更をClient2,3に伝えるには？



もうちょっとChannelLayer


```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

  async def connect(self):
    :
    self.group_name = 'group_1'
    await self.channel_layer.group_add(
      self.group_name,
      self.channel_name  # auto injection
    )

  async def receive_json(self, content, **kwargs):
    :
    # broadcast other consumers
    await self.channel_layer.group_send(
      self.group_name,
      {
        'type': 're_send',
      }
    )

  async def re_send(self, *args, **kwargs):
    await self.send_json('new_content')
```

もうちょっとChannelLayer


```python
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        :
        self.group_name = 'group_1'
        await self.channel_layer.group_add(
            self.group_name,
            self.channel_name  # auto injection
        )

    async def receive_json(self, content, **kwargs):
        :
        # broadcast other consumers
        await self.channel_layer.group_send(
            self.group_name,
            {
                'type': 're_send',
            }
        )

    async def re_send(self, *args, **kwargs):
        await self.send_json('new_content')
```

connect時に
処理内容を共有する
グループへ登録

もうちょっとChannelLayer


```python
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        :
        self.group_name = 'group_1'
        await self.channel_layer.group_add(
            self.group_name,
            self.channel_name  # auto injection
        )

    async def receive_json(self, content, **kwargs)
        :
        # broadcast other consumers
        await self.channel_layer.group_send(
            self.group_name,
            {
                'type': 're_send',
            }
        )

    async def re_send(self, *args, **kwargs):
        await self.send_json('new_content')
```

group_sendで
ChannelLayerを
通じて他のConsumerに
リクエストする

もうちょっとChannelLayer


```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

  async def connect(self):
    :
    self.group_name = 'group_1'
    await self.channel_layer.group_add(
      self.group_name,
      self.channel_name  # auto injection
    )

  async def receive_json(self, content, **kwargs):
    :
    # broadcast other consumers
    await self.channel_layer.group_send(
      self.group_name,
      {
        'type': 're_send',
      }
    )

  async def re_send(self, *args, **kwargs):
    await self.send_json('new_content')
```

typeで指定したメソッドが
各Consumerで実行される

実際のコードでは

application/views/ws/consumers/kanban_consumer.py

```
  def __init__(self, *args, **kwargs):
    :
    self.action_map = {
      'update_card_order': self.update_card_order,
      'update_pipe_line_order': self.update_pipe_line_order,
      'add_pipe_line': self.add_pipe_line,
      'add_card': self.add_card,
      'rename_pipe_line': self.rename_pipe_line,
      'delete_pipe_line': self.delete_pipe_line,
      'delete_board': self.delete_board,
      'rename_board': self.rename_board,
      'broadcast_board_data': self.broadcast_board_data,
      'broadcast_board_data_without_requester': self.broadcast_board_data_without_requester,
    }
```

実際のコードでは

application/views/ws/consumers/base_consumer.py

```
async def receive_json(self, content, **kwargs):
    """
    Typeに応じた処理を呼び出して実行する
    :param dict content:
    :param kwargs:
    :return:
    """
    action = self.action_map.get(content['type'])
    if not action:
        raise ConsumerException('{} is not a valid action_type'.format(content['type']))
    await action(content)
```

content.typeに応じたメソッドを
呼び出すようにreceive_jsonを
オーバライド

カンバンとVue

Vueがカンバンに向いてるということではなく
単に私がVueでつくったので紹介だけ

カンバンに必要な機能

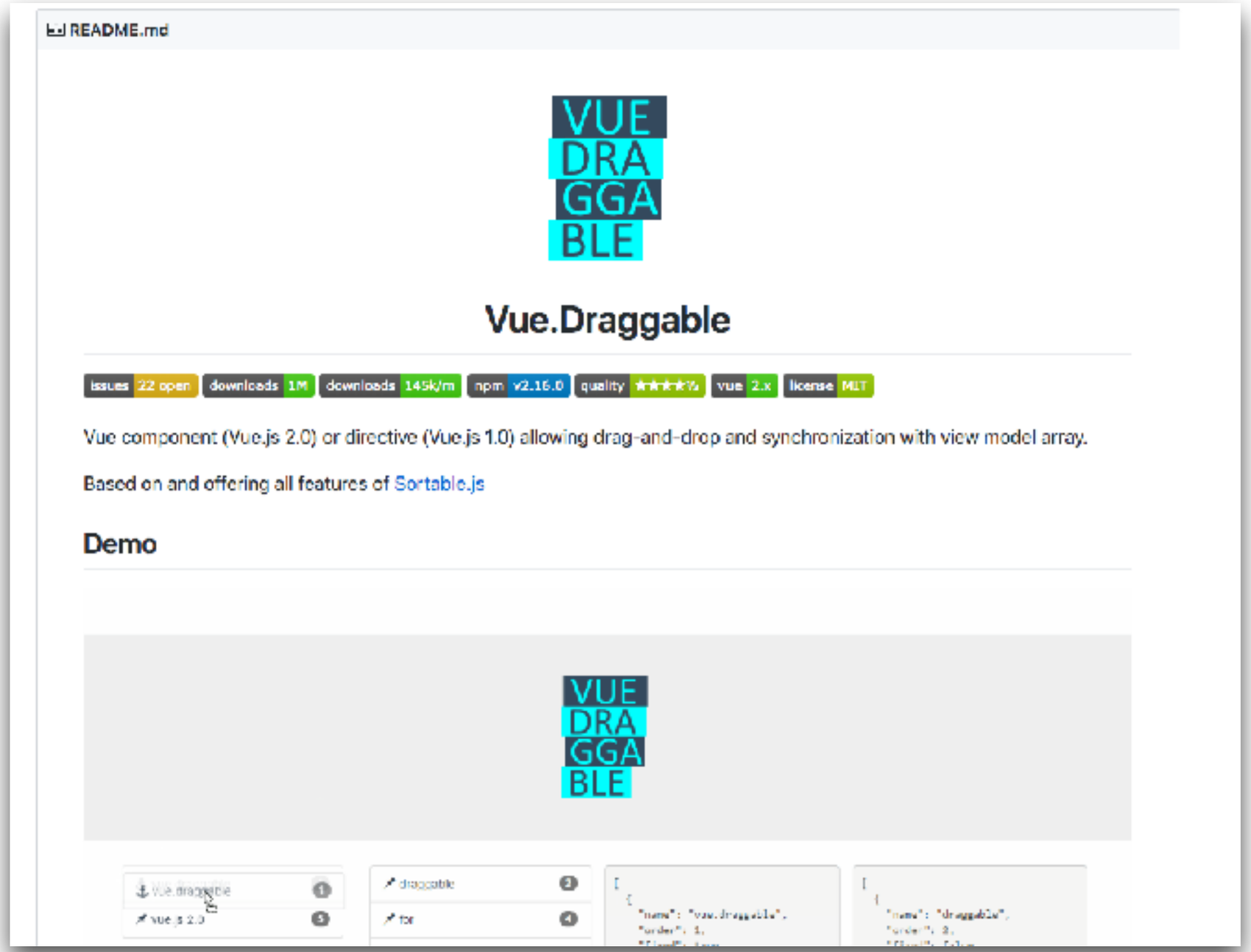
ドラッグアンドドロップでの
直感的な操作

クライアントサイドの
JS頑張る

リアルタイムなデータの
反映

ドラッグアンドドロップ

Vue Draggable

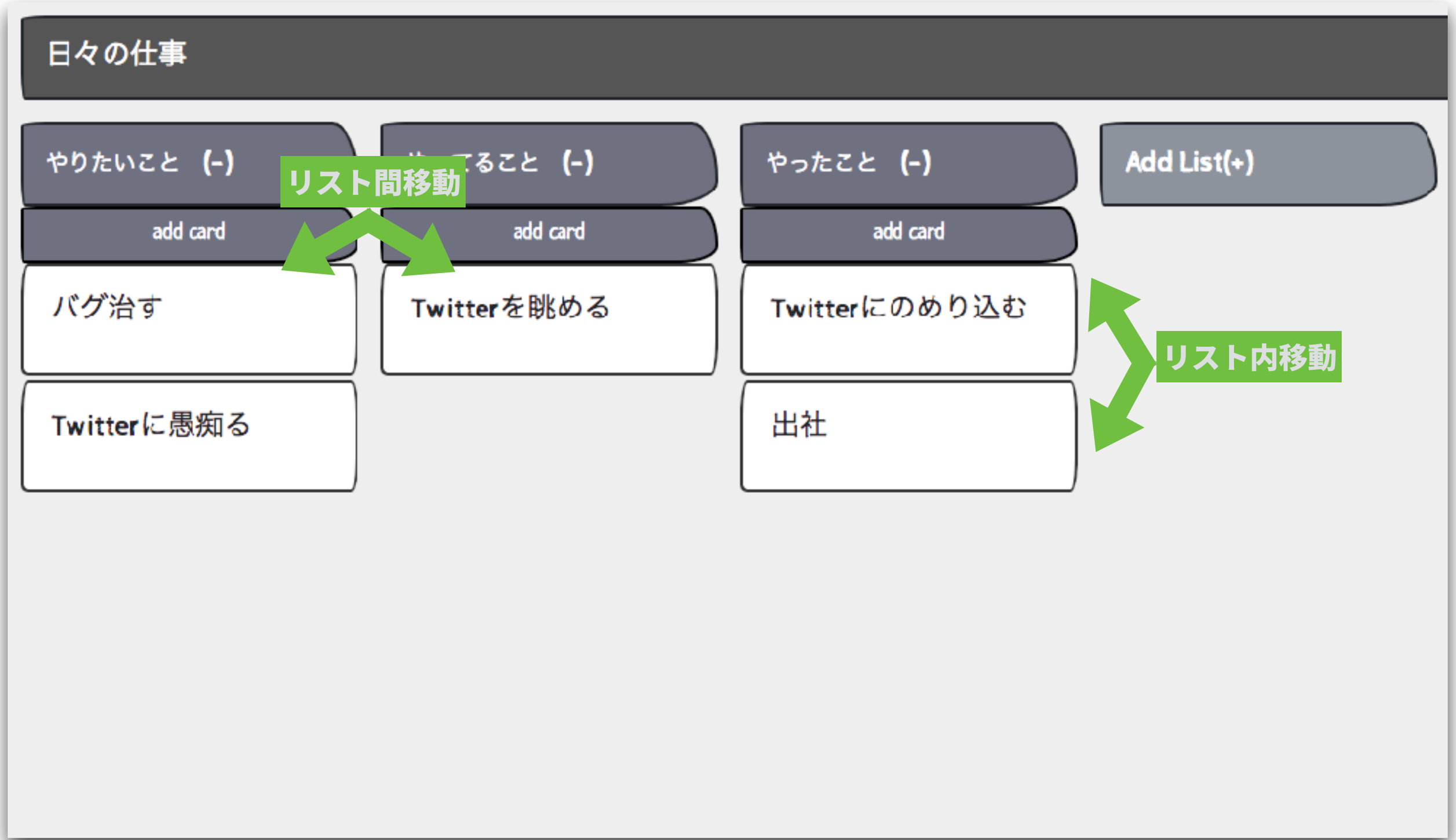


https://github.com/SortableJS/Vue.Draggable

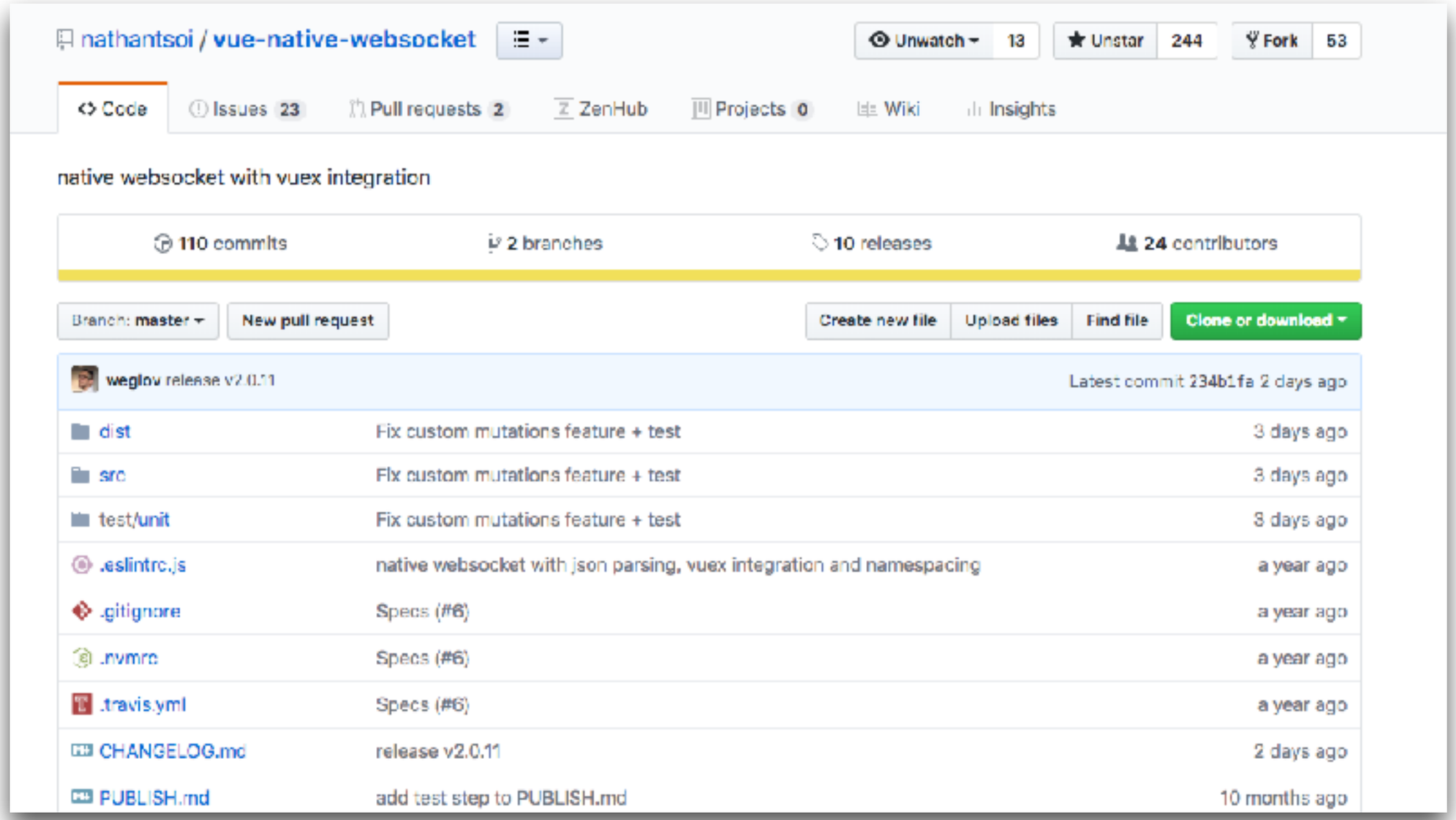
Vue Draggable

- Sortable.jsのラッパーで使いやすい
- リスト内はもちろん、リスト間のD&Dも容易
- D&D完了時に、並び順のデータがちゃんと取れる

Vue Draggable



VueNativeWebsocket



https://github.com/nathantsoi/vue-native-websocket

VueNativeWebsocket

Commits on May 31, 2018

Update README.md ···

justerror committed on 31 May ✓

Verified | 4f9acd0 | <>

Commits on May 7, 2018

Feature handle skip scheme ws url ···

denzow committed on 7 May ✓

c0f0209 | <>

Commits on Apr 18, 2018

update build

weglov committed on 18 Apr ✓

649485e | <>

Merge pull request #48 from saspallow/master ···

weglov committed on 18 Apr ✓

Verified | 45c763f | <>

VueNativeWebsocket

Commits on May 31, 2018

Update README.md ···

justerror committed on 31 May ✓

Verified | 4f9acd0 | <>

Commits on May 7, 2018

Feature handle skip scheme ws url ···

denzow committed on 7 May ✓

c0f0209 | <>

Commits on Apr 18, 2018

update build

weglov committed on 18 Apr ✓

649485e | <>

Merge pull request #48 from saspallow/master ···

weglov committed on 18 Apr ✓

Verified | 45c763f | <>

VueNativeWebsocket

- Vue側から簡単にWebsocketが使える
- Store(Vuex)とも連携が簡単
- ServerからMutation/Actionが呼び出せる

```
Vue.use(VueNativeSock, 'ws://localhost:9090', {
 store,
})
```

- sendObjでJSONを簡単に投げられる

```
socket.sendObj({
    type: 'add_card',
    pipeLineId,
    cardTitle,
});
```

VueNativeWebsocket

- シンプルな用途であれば使いやすくて便利
- 自動再接続や手動接続のメソッドもある

- 接続先の変更とかはちょっとだるい

```
// 切断
Vue.prototype.$disconnect();

// InstallされているVueNativeSockを一旦削除する
const index = Vue._installedPlugins.indexOf(VueNativeSock);
if (index > -1) {
  Vue._installedPlugins.splice(index, 1);
}
// 新しいURLで再インストールする
Vue.use(VueNativeSock, `/new_ws_endpoint/`, {
    connectManually: true,
    reconnection: true,
    reconnectionAttempts: 5,
    reconnectionDelay: 3000,
    format: 'json',
    store,
});
// 新しいURLへ接続する
Vue.prototype.$connect();
```

データの流れ


KANBAN
PyConJP
Search
delete
TODO (-)
DOING (-)
DONE (-)
Add List(+)
add card
add card
add card
参加ブログ書く
自分の発表
朝ごはん
早く終わって気楽になりたい
緊張した面持ち
遅刻しない
十分な睡眠

データの流れ(VueComponent)


```
<Draggable
 class="card-container"
 :options="options"
 v-model="wrappedCardList"
>
 <Card v-for="card in wrappedCardList"
     class="item"
     v-show="card.isShown"
     :card="card"
     :key="card.cardId"
 />
</Draggable>
:
computed: {
 wrappedCardList: {
  get() {
   return this.pipeLine.cardList;
  },
  set(value) {
   console.log(value);
   this.updateCardOrder({
    pipeLineId: this.pipeLine.pipeLineId,
    cardList: value,
   });
  },
 },
```

application/vuejs/src/pages/Board/components/BoardArea/PipeLine.vue

データの流れ(VueComponent)

Draggableで
囲んだ要素(Card)が
D&D可能に

```
<Draggable
class="card-container"
:options="options"
v-model="wrappedCardList"
>
<Card v-for="card in wrappedCardList"
    class="item"
    v-show="card.isShown"
    :card="card"
    :key="card.cardId"
/>
</Draggable>
:
computed: {
 wrappedCardList: {
  get() {
   return this.pipeLine.cardList;
  },
  set(value) {
   console.log(value);
   this.updateCardOrder({
    pipeLineId: this.pipeLine.pipeLineId,
    cardList: value,
   });
  },
 },
```

D&Dの更新内容は
V-modelに同期される
作り

application/vuejs/src/pages/Board/components/BoardArea/PipeLine.vue

データの流れ(VueComponent)


```
<Draggable
 class="card-container"
 :options="options"
 v-model="wrappedCardList"
>
 <Card v-for="card in wrappedCardList"
     class="item"
     v-show="card.isShown"
     :card="card"
     :key="card.cardId"
 />
</Draggable>
:
computed: {
 wrappedCardList: {
  get() {
   return this.pipeLine.cardList;
  },
  set(value) {
   console.log(value);
   this.updateCardOrder({
    pipeLineId: this.pipeLine.pipeLineId,
    cardList: value,
   });
  },
 },
```

D&D完了時にsetが呼ばれる

```
[
  {cardId: 1, title: ....},
  {cardId: 2, title: ....},
  {cardId: 3, title: ....},
]
```

application/vuejs/src/pages/Board/components/BoardArea/PipeLine.vue

データの流れ(VueComponent)


```
<Draggable
 class="card-container"
 :options="options"
 v-model="wrappedCardList"
>
 <Card v-for="card in wrappedCardList"
    class="item"
    v-show="card.isShown"
    :card="card"
    :key="card.cardId"
 />
</Draggable>
:
computed: {
 wrappedCardList: {
  get() {
   return this.pipeLine.cardList;
  },
  set(value) {
   console.log(value);
   this.updateCardOrder({
    pipeLineId: this.pipeLine.pipeLineId
    cardList: value,
   });
  },
 },
```

Vuexに処理を
依頼する

application/vuejs/src/pages/Board/components/BoardArea/PipeLine.vue

データの流れ(Vuex)


```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {
 const socket = getters.getSocket;
 socket.sendObj({
   type: 'update_card_order',
   pipeLineId,
   cardIdList: cardList.map(x => x.cardId),
 });
 commit('updateCardOrder', { pipeLineId, cardList });
},

// MUTATION
updateCardOrder(state, { pipeLineId, cardList }) {
 const targetPipeLine = state.boardData.pipeLineList
                  .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
 targetPipeLine.cardList = cardList;
},
```

application/vuejs/src/store/pages/board.js

データの流れ(Vuex)


```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {
 const socket = getters.getSocket;
  socket.sendObj({
    type: 'update_card_order',
    pipeLineId,
    cardIdList: cardList.map(x => x.cardId),
  });
  commit('updateCardOrder', { pipeLineId, cardList });
},

// MUTATION
updateCardOrder(state, { pipeLineId, cardList }) {
  const targetPipeLine = state.boardData.pipeLineList
                      .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
  targetPipeLine.cardList = cardList;
},
```

VueComponentから
ここが呼ばれる。

application/vuejs/src/store/pages/board.js

データの流れ(Vuex)


```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineI
 const socket = getters.getSocket;
  socket.sendObj({
   type: 'update_card_order',
   pipeLineId,
   cardIdList: cardList.map(x => x.cardId),
  });
  commit('updateCardOrder', { pipeLineId, cardList });
},

// MUTATION
updateCardOrder(state, { pipeLineId, cardList }) {
  const targetPipeLine = state.boardData.pipeLineList
                    .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
  targetPipeLine.cardList = cardList;
},
```

VueNativeWebsocketを使って、Django側に更新を依頼(後述)

application/vuejs/src/store/pages/board.js

データの流れ(Vuex)


```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {
 const socket = getters.getSocket;
  socket.sendObj({
    type: 'update_card_order',
    pipeLineId,
    cardIdList: cardList.map(x => x.cardId),
  });
  commit('updateCardOrder', { pipeLineId, cardList });
},

// MUTATION
updateCardOrder(state, { pipeLineId, cardList }) {
  const targetPipeLine = state.boardData.pipeLineList
                        .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
  targetPipeLine.cardList = cardList;
},
```

投げると同時に投げた
クライアント側では更新が
行われた体で表示を更新

application/vuejs/src/store/pages/board.js

データの流れ(Vuex)


```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {
 const socket = getters.getSocket;
  socket.sendObj({
    type: 'update_card_order',
    pipeLineId,
    cardIdList: cardList.map(x => x.cardId),
  });
  commit('updateCardOrder', { pipeLineId, cardList });
},

// MUTATION
updateCardOrder(state, { pipeLineId, cardList }) {
  const targetPipeLine = state.boardData.pipeLineList
                      .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
  targetPipeLine.cardList = cardList;
},
```

application/vuejs/src/store/pages/board.js

データの流れ(Django Channels)


```
class KanbanConsumer(BaseJsonConsumer):

    def __init__(self, *args, **kwargs):
        :
        self.action_map = {
            'update_card_order': self.update_card_order,
        :
    }
    :
```

application/views/ws/consumers/kanban_consumer.py

データの流れ(Django Channels)


```
class KanbanConsumer(BaseJsonConsumer):

    def __init__(self, *args, **kwargs):
        :
        self.action_map = {
            'update_card_order': self.update_card_order,
            :
        }
        :
```

type: 'update_card_order'は
self.update_card_orderを
呼び出しするようにマップ

application/views/ws/consumers/kanban_consumer.py

データの流れ(Django Channels)


```python
class KanbanConsumer(BaseJsonConsumer):

    async def update_card_order(self, content):
        """
        ボード内のカードの並び順を更新する
        """
        pipe_line_id = content['pipeLineId']
        card_id_list = content['cardIdList']
        await database_sync_to_async(kanban_sv.update_card_order)(
            pipe_line_id,
            card_id_list
        )
        await self.broadcast_board_data_without_requester()
```

application/views/ws/consumers/kanban_consumer.py

データの流れ(Django Channels)


```python
class KanbanConsumer(BaseJsonConsumer):

    async def update_card_order(self, content):
        """
        ボード内のカードの並び順を更新する
        """
        pipe_line_id = content['pipeLineId']
        card_id_list = content['cardIdList']
        await database_sync_to_async(kanban_sv.update_card_order)(
            pipe_line_id,
            card_id_list
        )
        await self.broadcast_board_data_without_requester()
```

DjangoORM経由でカードの
並び順を更新

application/views/ws/consumers/kanban_consumer.py

データの流れ(Django Channels)


```python
class KanbanConsumer(BaseJsonConsumer):

    async def update_card_order(self, content):
        """
        ボード内のカードの並び順を更新する
        """
        pipe_line_id = content['pipeLineId']
        card_id_list = content['cardIdList']
        await database_sync_to_async(kanb
            pipe_line_id,
            card_id_list
        )
        await self.broadcast_board_data_without_requester()
```

更新処理をした
Consumer以外に
ブロードキャスト

application/views/ws/consumers/kanban_consumer.py

データの流れ(Django Channels)


```python
class KanbanConsumer(BaseJsonConsumer):

    async def update_card_order(self, content):
        """
        ボード内のカードの並び順を更新する
        """
        pipe_line_id = content['pipeLineId']
        card_id_list = content['cardIdList']
        await database_sync_to_async(kanban_sv.update_
            pipe_line_id,
            card_id_list
        )
        await self.broadcast_board_data_without_requester()
```

application/views/ws/consumers/kanban_consumer.py

```
// ACTION
updateCardOrder({ commit, getters }, { pipeLineId, cardList }) {
 const socket = getters.getSocket;
  socket.sendObj({
    type: 'update_card_order',
    pipeLineId,
    cardIdList: cardList.map(x => x.cardId),
  });
  commit('updateCardOrder', { pipeLineId, cardList });
},

// MUTATION
updateCardOrder(state, { pipeLineId, cardList }) {
  const targetPipeLine = state.boardData.pipeLineList
                        .find(pipeLine => pipeLine.pipeLineId === pipeLineId);
  targetPipeLine.cardList = cardList;
},
```

処理を依頼したクライアントは
すでに更新できた体で
再描画済み

application/vuejs/src/store/pages/board.js

データの流れ(Django Channels)


```
class KanbanConsumer(BaseJsonConsumer):
    :
    async def broadcast_board_data_without_requester(self, *args):
        await self.group_send(
            self.room_group_name,
            {
                'type': 'send_board_data',
                'requester_id': self.consumer_id,
            }
        )
```

application/views/ws/consumers/kanban_consumer.py

データの流れ(Django Channels)


```
class KanbanConsumer(BaseJsonConsumer):
    :
    async def broadcast_board_data_without_re
        await self.group_send(
            self.room_group_name,
            {
                'type': 'send_board_data',
                'requester_id': self.consumer_id,
            }
        )
```

channel_layerのgroup_sendで
他のConsumerにsend_board_dataの
実行を依頼

application/views/ws/consumers/kanban_consumer.py

データの流れ(Django Channels)


```python
class KanbanConsumer(BaseJsonConsumer):
    :
    async def send_board_data(self, event):
        """
        ボードのデータを送る
        """
        # 自身が発火したブロードキャストなら無視する
        if event.get('requester_id') == self.consumer_id:
            return
        board_data = await database_sync_to_async(
            kanban_sv.get_board_data_board_id
        )(self.board_id)
        await self.send_data({
            'boardData': board_data,
        }, mutation='setBoardData')
```

application/views/ws/consumers/kanban_consumer.py

データの流れ(Django Channels)


```python
class KanbanConsumer(BaseJsonConsumer):
    :
    async def send_board_data(self, event):
        """
        ボードのデータを送る
        """
        # 自身が発火したブロードキャストなら無視する
        if event.get('requester_id') == self.consumer_id:
            return
        board_data = await database_sync_to_async(
            kanban_sv.get_board_data_board_id
        )(self.board_id)
        await self.send_data({
            'boardData': board_data,
        }, mutation='setBoardData')
```

Django ORM経由でボードの構成データを再取得

application/views/ws/consumers/kanban_consumer.py

データの流れ(Django Channels)


```
class KanbanConsumer(BaseJsonConsumer):
    :
    async def send_board_data(self, event):
        """
        ボードのデータを送る
        """
        # 自身が発火したブロードキャストなら無視する
        if event.get('requester_id') == self.consumer_id:
            return
        board_data = await database_sync
            kanban_sv.get_board_data_boa
        )(self.board_id)
        await self.send_data({
            'boardData': board_data,
        }, mutation='setBoardData')
```

新しいボードデータを
対応するクライアントに送信

データの流れ(Django Channels)


```
class KanbanConsumer(BaseJsonConsumer):
  :
  async def send_board_data(self, event):
    """
    ボードのデータを送る
    """
    # 自身が発火したブロードキャストなら無視する
    if event.get('requester_id') == self.consumer_id:
      return
    board_data = await database_sync_to_async(
      kanban_sv.get_board_data_board_id
    )(self.board_id)
    await self.send_data({
      'boardData': board_data,
    }, mutation='setBoardData')
```

クライアント側の
mutation:setBoardDataに
引き渡す

application/views/ws/consumers/kanban_consumer.py

データの流れ(Vuex)


```
const state = {
  boardData: {
    pipeLineList: [],
  },
  focusedCard: {},
  searchWord: '',
};

// MUTATION
setBoardData(state, { boardData }) {
  state.boardData = camelcaseKeys(boardData, { deep: true });
},
```

application/vuejs/src/store/pages/board.js

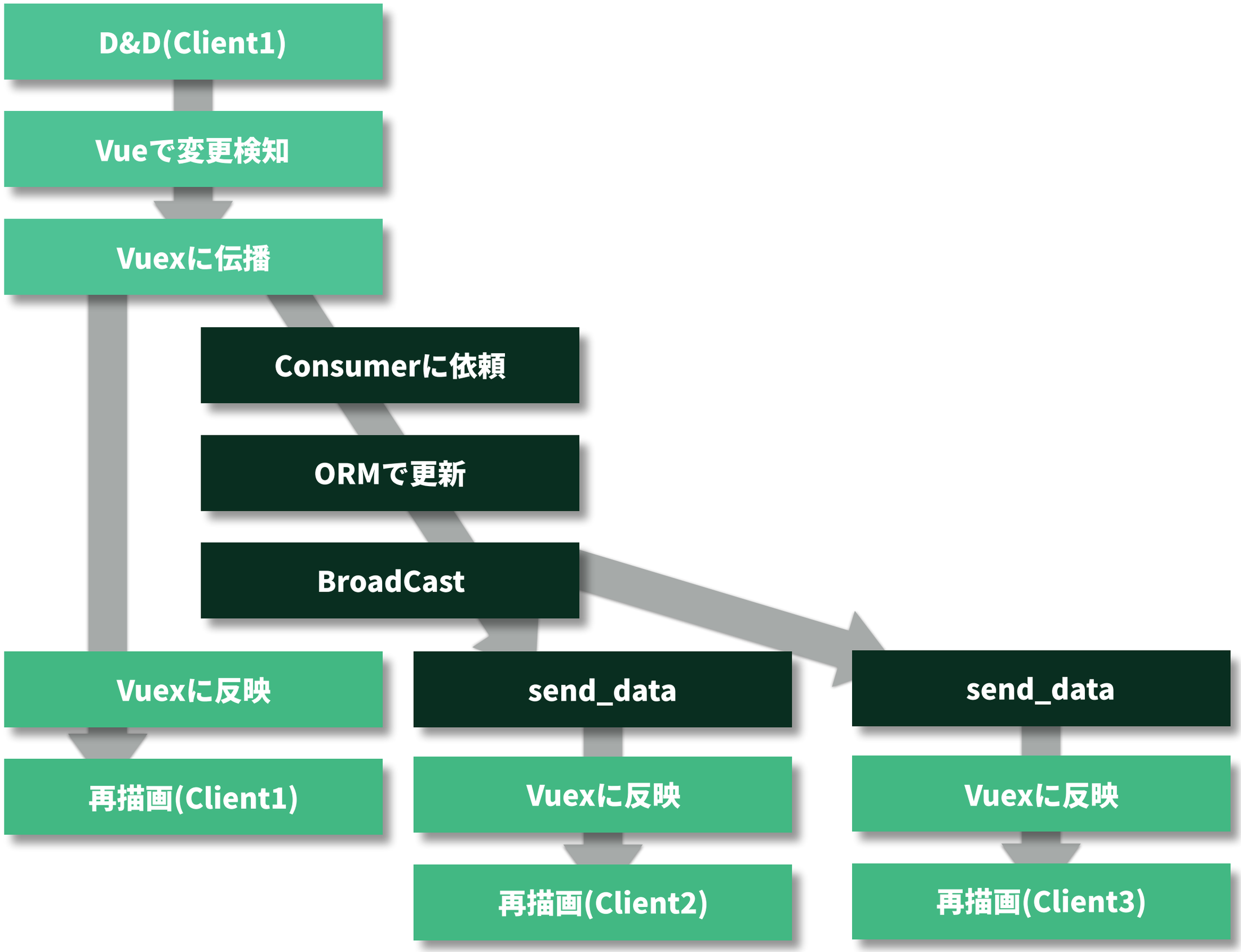
データの流れ(Vuex)


```
const state = {
  boardData: {
    pipeLineList: [],
  },
  focusedCard: {},
  searchWord: '',
};

// MUTATION
setBoardData(state, { boardData }) {
  state.boardData = camelcaseKeys(boardData, { deep: true });
},
```

返送されたデータで
更新し画面に反映

application/vuejs/src/store/pages/board.js

データの流れ(まとめ)


D&D(Client1)
Vueで変更検知
Vuexに伝播
Consumerに依頼
ORMで更新
BroadCast
Vuexに反映
再描画(Client1)
send_data
Vuexに反映
再描画(Client2)
send_data
Vuexに反映
再描画(Client3)

つらい話

たった5人でサービスを落とす方法

カンバン周りの開発を終えてRelease直前に
動作確認をしてたら5人くらいでサーバごとハング

Consumer復習


```
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        # accept or close
        if YourCondition:
            self.accept()
        else:
            self.close()

    async def receive_json(self, content, **kwargs):
        # receive message
        print(content)

        # echo back
        await self.send_json(content)
```

Consumer復習


```python
class MyConsumer(AsyncJsonWebsocketConsumer or JsonWebsocketConsumer):

    async def connect(self):
        # accept or close
        if YourCondition:
            self.accept()
        else:
            self.close()

    async def receive_json(self, content, **kwargs):
        # receive message
        print(content)

        # echo back
        await self.send_json(content)
```

or ?

たった5人でサービスを落とす方法

- ConsumerはSync実装、Async実装の2種類
- Sync実装ではスレッドで動作
- Async実装ではコルーチンで動作

たった5人でサービスを落とす方法

- ConsumerはSync実装、Async実装の2種類
- Sync実装ではスレッドで動作
- Async実装ではコルーチンで動作
- 重い処理がConsumer内にあった
- その処理が重なるとサーバごとハングして死

SyncをAsyncに全面書き換え

SyncをAsyncに全面書き換え

- 各メソッドを async defに書き換え
- ORM周りをdatabase_to_asyncでラップ

SyncをAsyncに全面書き換え

- 各メソッドを async defに書き換え
- ORM周りをdatabase_to_asyncでラップ
- Sync実装の20倍でもサーバがハングしなくなる

静かな凡ミス

間違い探し

```
async def add_card(self, content):
    """
    カードの追加
    """
    pipe_line_id = content['pipeLineId']
    card_title = content['cardTitle']
    database_sync_to_async(kanban_sv.add_card)(
        pipe_line_id,
        card_title
    )
    self.broadcast_board_data()
```

間違い探し

```
async def add_card(self, content):
    """
    カードの追加
    """
    pipe_line_id = content['pipeLineId']
    card_title = content['cardTitle']
    await database_sync_to_async(kanban_sv.add_card)(
        pipe_line_id,
        card_title
    )
    await self.broadcast_board_data()
```

実行してもエラーが出ず

サイレントに処理されないので気が付かない

突然の
My SQL Server has gone away

間違い探し

```
async def hoge(self, event):
    # カードを取得
    card = await database_sync_to_async(Card.objects.get)(id=1)
    # カードが所属するボードを取得して、その全体のデータを取得
    board_data = await database_sync_to_async(kanban_sv.get_board_data)(
        card.board
    )
```

間違い探し

```
async def hoge(self, event):
    # カードを取得
    card = await database_sync_to_async(Card.objects.get)(id=1)
    # カードが所属するボードを取得して、その全体のデータを取得
    board_data = await database_sync_to_async(kanban_sv.get_board_data)(
        card.board
    )
```

FKをたどるだけでも、ORMの操作になる。
database_sync_to_asyncで囲わないと
接続がリークして、やがて死ぬ

突然のMySQL Server has gone away

http://www.denzow.me/entry/2018/07/30/000955

2018-07-30

編集

Django Channelsのdatabase_sync_to_asyncを理解しなくて死んだ話

Channels Django Python

Django Channelsの `@database_sync_to_async` デコレータを正しく理解しなくてえらい目('MySQL server has gone away')にあったので、確認して理解した内容を残しておきます。

あれ、Websocketで成功・失敗って・・・

あれ、Websocketで成功・失敗って・・・

- 例えばカード追加がたまに失敗するシステム
- 成功・失敗毎にクライントにAlert出したい

```
await KanbanClient.deleteCard({
  boardId,
  cardId,
}).then(res => {
  alert('success');
}).catch(e => {
  alert('error');
});
```

成功時

失敗時

あれ、Websocketで成功・失敗って・・・

- WebsocketはPromiseではない
- 返答を待つという概念はない

```
socket.sendObj({
  type: 'add_card',
  pipeLineId,
  cardTitle,
});
```

このメッセージの処理状況を
容易に管理できない

どうしたのか

- 更新部分はAjaxで処理
- 処理完了時に、クライアントがブロードキャストを依頼
- ローディング、成功失敗時の通知等が実装可能に

あれ、Websocketで成功・失敗って・・・


```
async updateCardTitle({ commit, dispatch }, { boardId, cardId, title }) {
 const cardData = await KanbanClient.updateCardData({
  boardId,
  cardId,
  title,
 });
 dispatch('broadcastBoardData');
},
:
broadcastBoardData({ getters }) {
 console.log('call broadcastBoardData');
 const socket = getters.getSocket;
 socket.sendObj({
  type: 'broadcast_board_data',
 });
},
```

あれ、Websocketで成功・失敗って・・・


```
async updateCardTitle({ commit, dispatch }, { boardId, cardId, title }) {
 const cardData = await KanbanClient.updateCardData({
  boardId,
  cardId,
  title,
 });
 dispatch('broadcastBoardData');
},
:
broadcastBoardData({ getters }) {
 console.log('call broadcastBoardData');
 const socket = getters.getSocket;
 socket.sendObj({
  type: 'broadcast_board_data',
 });
},
```

Ajaxでカードを
更新(await)

あれ、Websocketで成功・失敗って・・・


```
async updateCardTitle({ commit, dispatch }, { boardId, cardId, title }) {
 const cardData = await KanbanClient.updateCardData({
  boardId,
  cardId,
  title,
 });
 dispatch('broadcastBoardData');
},
:
broadcastBoardData({ getters }) {
 console.log('call broadcastBoardData');
 const socket = getters.getSocket;
 socket.sendObj({
  type: 'broadcast_board_data',
 });
},
```

Ajaxが終わったら
別の処理を発火

あれ、Websocketで成功・失敗って・・・


```
async updateCardTitle({ commit, dispatch }, { boardId, cardId, title }) {
 const cardData = await KanbanClient.updateCardData({
  boardId,
  cardId,
  title,
 });
 dispatch('broadcastBoardData');
},
:
broadcastBoardData({ getters }) {
 console.log('call broadcastBoardData');
 const socket = getters.getSocket;
 socket.sendObj({
  type: 'broadcast_board_data',
 });
},
```

websocket経由で
consumerに
ブロードキャストを依頼

ASGIサーバなのにChannels動かんやん (2018年6月頃)

メジャーなASGIサーバ

README.rst

daphne

build passing pypi v2.2.2

Daphne is a HTTP, HTTP2 and WebSocket protocol server for ASGI and ASGI-HTTP, developed to power Django Channels.

It supports automatic negotiation of protocols; there's no need for URL prefixing to determine WebSocket endpoints versus HTTP endpoints.

Note: Daphne 2 is not compatible with Channels 1.x applications, only with Channels 2.x and other ASGI applications. Install a 1.x version of Daphne for Channels 1.x support.

Running

Simply point Daphne to your ASGI application, and optionally set a bind address and port (defaults to localhost, port 8000):

```
daphne -b 0.0.0.0 -p 8001 django_project.asgi:application
```

https://github.com/django/daphne

メジャーなASGIサーバ

README.md

The lightning-fast ASGI server.

build passing codecov 91% pypi package 0.3.5

Documentation: https://www.uvicorn.org

Requirements: Python 3.5, 3.6, 3.7

Uvicorn is a lightning-fast ASGI server implementation, using uvloop and httptools.

https://github.com/encode/uvicorn

メジャーなASGIサーバ

README.rst

Hypercorn

pipeline passed | docs passing | pypi v0.2.4 | http 1.0,1.1,2 | python 3.6 | 3.7 | license MIT

Hypercorn is an ASGI web server based on the sans-io hyper, h11, h2, and wsproto libraries and inspired by Gunicorn. Hypercorn supports HTTP/1, HTTP/2, and websockets and the ASGI 2 specification.

Hypercorn was initially part of Quart before being separated out into a standalone ASGI server. Hypercorn forked from version 0.5.0 of Quart.

Quickstart

Hypercorn can be installed via pipenv or pip,

```
$ pipenv install hypercorn
$ pip install hypercorn
```

and requires Python 3.6.1 or higher.

With hypercorn installed ASGI frameworks (or apps) can be served via Hypercorn via the command line,

```
$ hypercorn module:app
```

https://gitlab.com/pgjones/hypercorn

ASGIサーバなのにChannels動かんやん

- Django Channelsがちゃんと動いたのはdaphneのみ
- uvicornはAuthMiddlewareで死亡
- Hypercornはなんでか忘れたけど死亡

ASGIサーバなのにChannels動かんやん

- Django Channelsがちゃんと動いたのはdaphneのみ
- uvicornはAuthMiddlewareで死亡
- Hypercornはなんでか忘れたけど死亡

とりあえずChannelsなら
daphne使っておけば安心(?)

まとめ

Channels大好き！

- 辛い話が最後続いたけど、Channelsは使いやすくていい！
- Websocketが書きやすくていい!
- 2.xのナレッジ少ないのでみんなでOutput & Followしていこう
- ASGIでひろがるPythonに期待

Pythonやりたい人

https://bit.ly/2NPWCc8