Go駆動開発で超速Pushエンジンを作った話

@plan9user

GoCon 2014 spring

Goを使いたい一心で受託業務へ持ち込んだ時の話

自己紹介

- 門多恭平
- 趣味は”Plan 9”です！
    - Go好きならPlan 9の思想も好むはず
- 2011年からフェンリルで働いてます

フェンリル株式会社

- こないだSleipnir 6が出ました！

- 共同開発部
  - 企業様からの受託開発を行う部署
  - 実績紹介 - http://biz.fenrir-inc.com

- Sleipnirへのご意見は公式サイトから

業務内容

- スマートフォンアプリ開発やってます
    - iOS, Android
    - Windows 8 (若干)
- 昨年末あたりからXamarinはじめました

go?

- プッシュ通知
  - 国内サービスでなければ×　とか
  - 自分で全体をコントロールしたい　とか
  - 買い切りたい　とか

push通知のしくみ

1. アプリトークンをサーバに送信

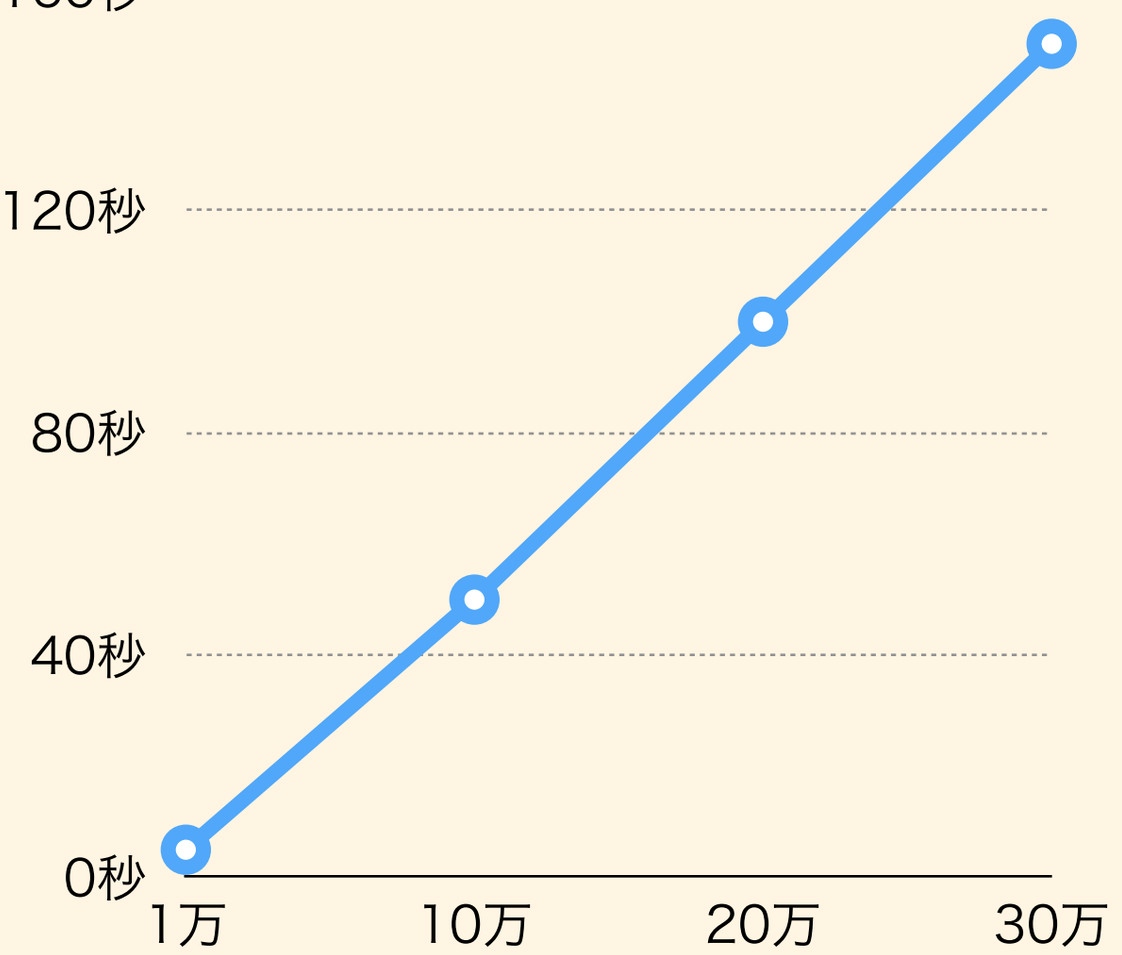
2. サーバは保存しているトークンとメッセージをセットにしてAPNs/GCMへ送信

3. メッセージが各端末へ届く

決して難しくはないんだけど...

performance problems...

- 最初はPHPで実装
- 水準を満たせない
- 数十万通知に耐えられない
- 設計がまずかった
- PHP悪くない



だめだったところ

problem?

- PHPなので並列化が難しかった
- 二重送信防止のためDBをキューにしていた
- 通信エラー等でリトライすべき場合や途中で落ちた場合を想定
- 完了マークもDBに持っていた

goal?

- 処理の待ち時間をなくして性能を上げる
- 端末数が多いときは分散して送信する
- 複数マシンでひとつのシステムにする
- DBへの依存をなくす

おや？

これGoの出番じゃね？

Goを好む理由(当時)

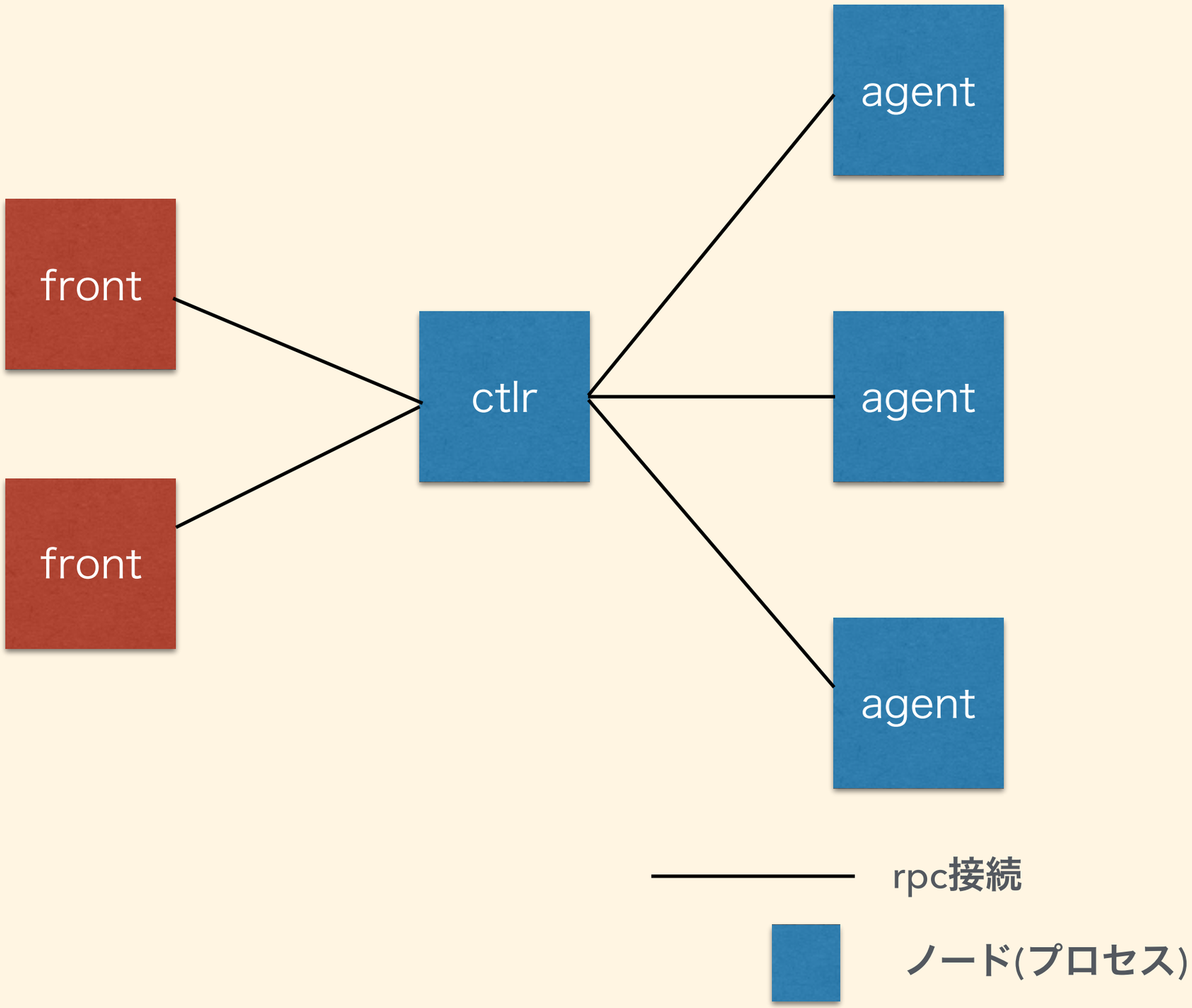
- Rob PikeやRuss Coxが参加してる
    - Plan 9にも関わっておられました
- Alef, Limboから続くCSP良い
- 標準パッケージも充実してそう(雰囲気)

GoとXamarin同時投入決定

- よく通ったなと思います...
- 運が良く期間限定のアプリだった
    - 長期運用なら保守も考える必要がある
- 開発期間が比較的長く取れた
    - 最初は調べながら書くので効率悪い

Goで書く！

overview

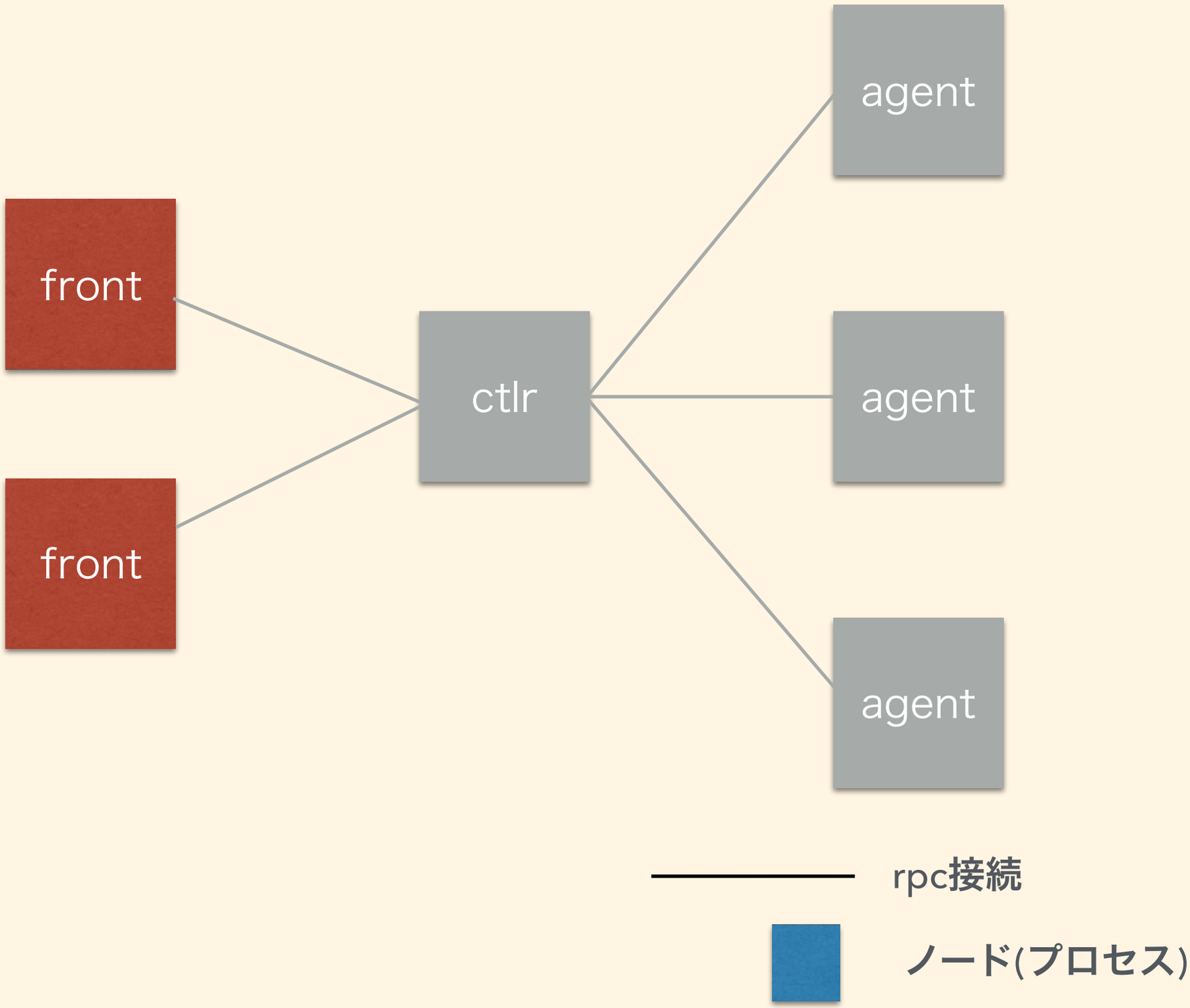


net/rpcパッケージ

```
// 決まった形のメソッドをエクスポートしておいて
fund (agent *Agent) Broadcast(r *Request, res *Response) error {
    return nil
}

// 別のノードからrpcをコールする
c, _ := rpc.Dial(“net”, “localhost:17030”)
err := c.Call(“Broadcast”, r, &res)
```

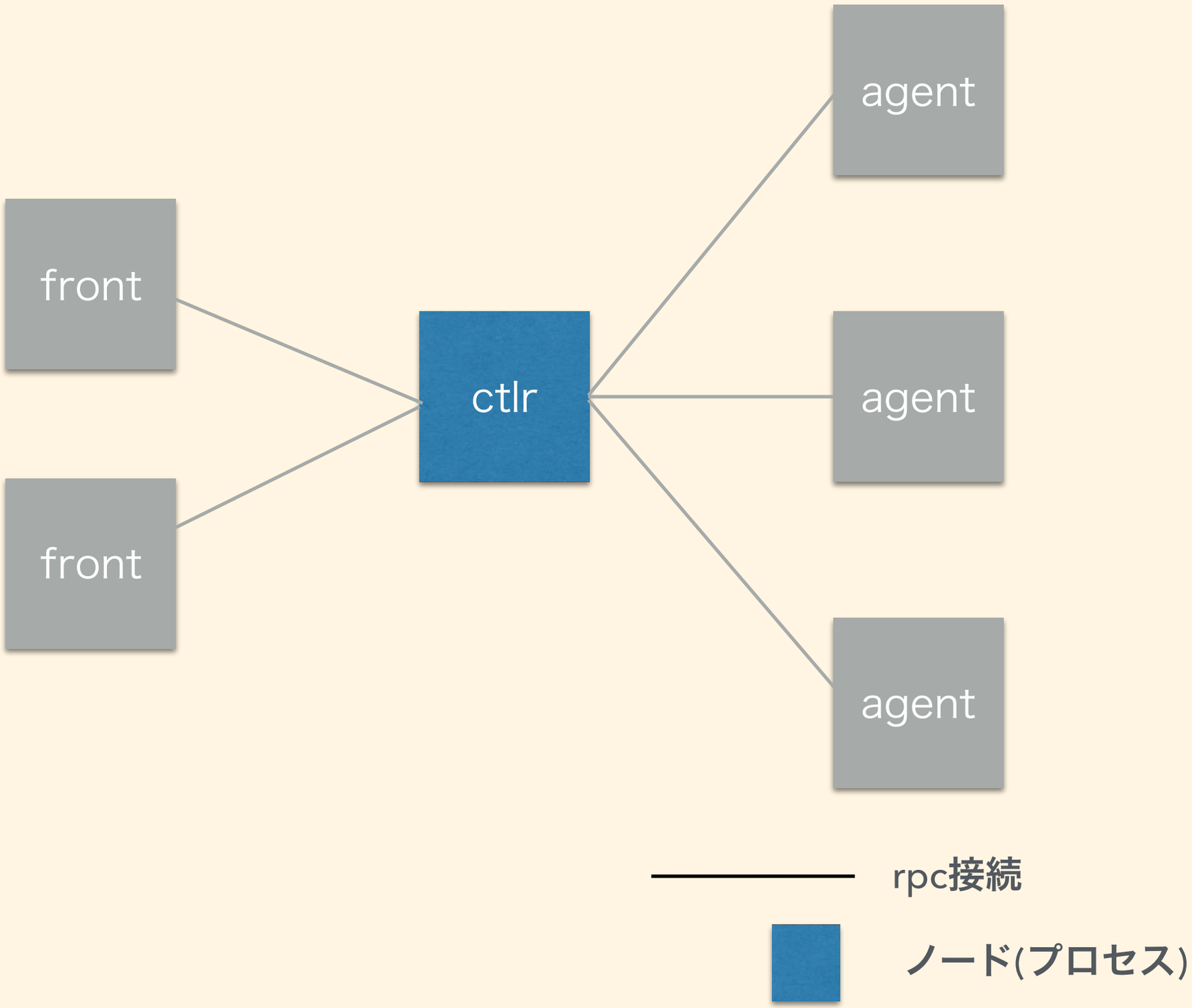
front



front

- 案件固有のロジックを担当
  - ほとんどがJSONを受信して、ctlrへリクエスト送信
- 案件固有のロジックなので都度実装する
- DB関係の処理もこのノードが対応する

controller



controller

- frontからのrpcリクエストを受け付けるノード
- 複数のエージェントを管理する
- 1リクエスト内の通知数が多すぎる場合は複数エージェントへ分割転送
- エージェントの選別は空いているものを選ぶ *

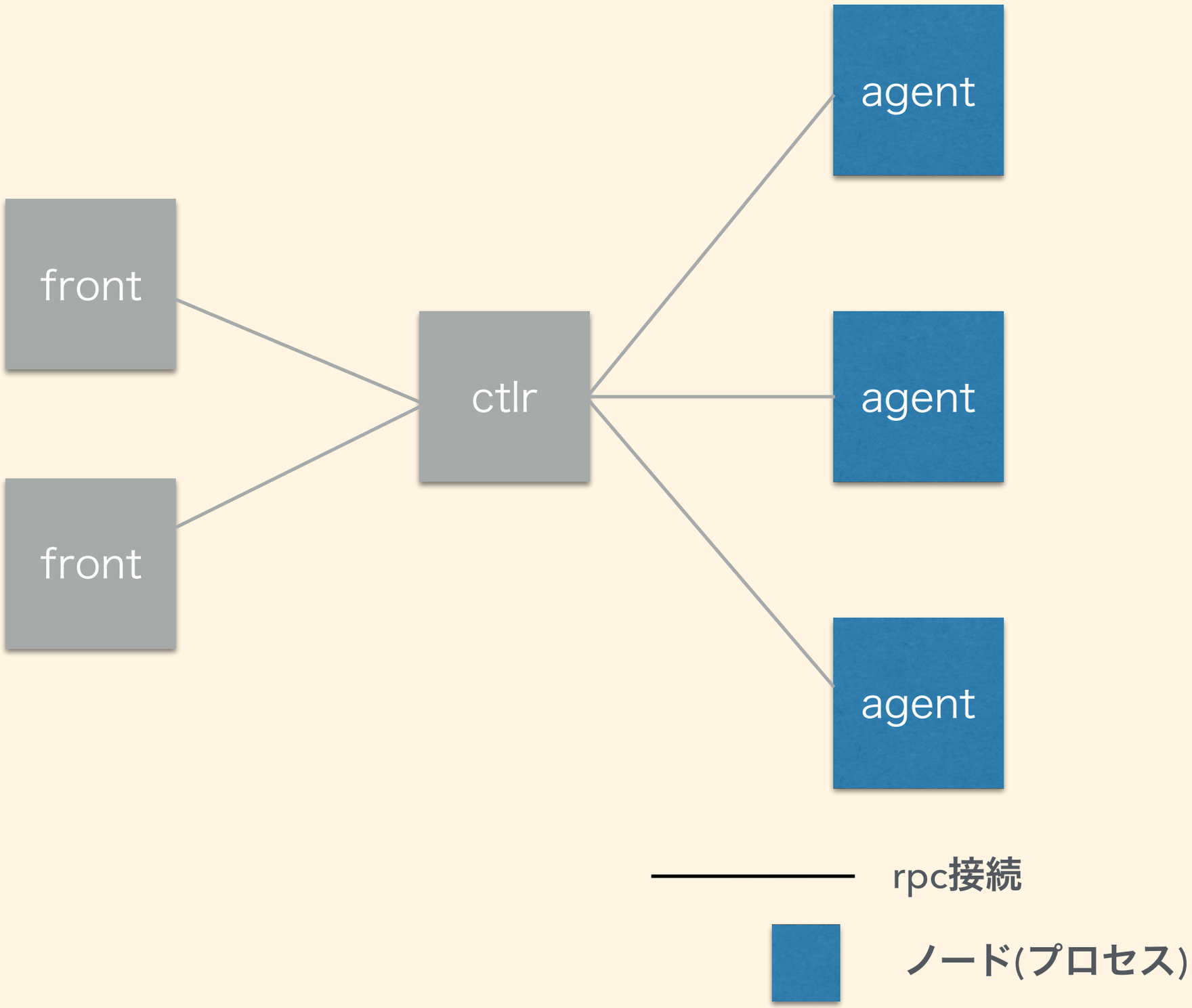
開いているagentを選択

```
// リクエスト用チャネルを1本だけ用意
req := make(chan *Request, bufSize)

// 管理するノード分、リクエスト用チャネルから受信するゴルーチンを起動しておく
for agent := range agents {
        go func(c chan *Request){
                _ = <-c
        }(req)
}

// チャネルにリクエストを投げると待機しているゴルーチンのどれかが拾う
select {
case req <- newReq:
        ....
case <-time.After(timeoutInterval):
        ...
```

agent



agent

- コントローラからのリクエストを実際にAPNs/GCMへ送信する
- 複数リクエストは可能なら1つにまとめる *

複数まとめて送信

```
que := make(chan *Message, bufSize)
...
// ctlrからのリクエスト受信
    req := <-que
    msgs = append(msgs, req)
    for len(que) > 0 {
        msgs = append(msgs, <-que)
    }
    conn.WriteMessages(msgs)
```

性能測定

性能測定

- 実際のAPNs/GCMを使って性能試験は難しい
  - 試験に耐えうる量の端末を持っていない
    - GCMは1端末へ数万件送るとInternalServerError
- Fake APNs/GCMを作ってベンチマーク
  - 少なくとも前回より良くなったのかは分かる

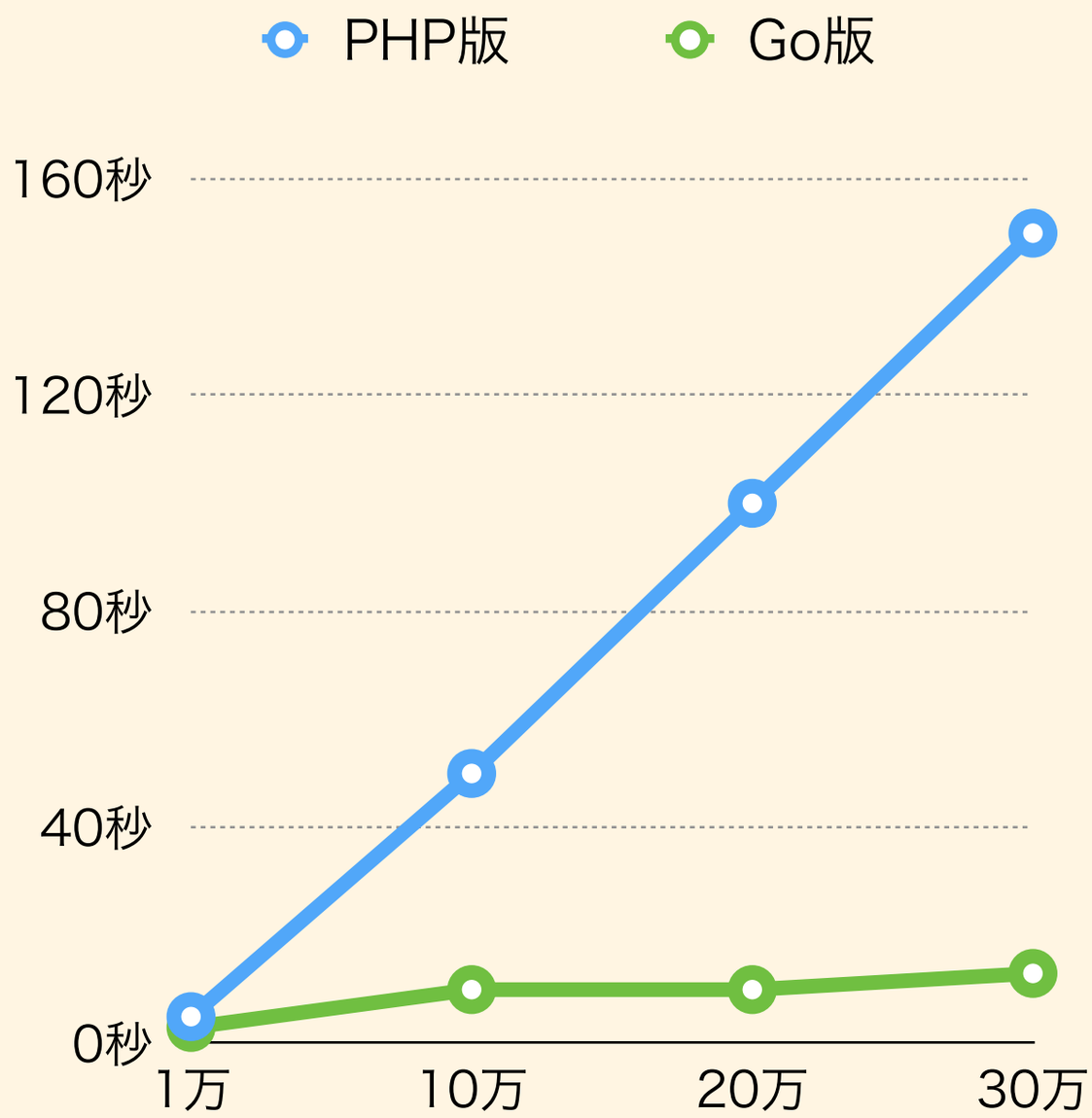
テストコード

```
// テスト用Fake APNsサーバ(本物はgateway.push.apple.com:2195)を起動
s := apnstest.NewServer(func(w io.Writer, msg *apns.Message) {
    // wに何かWriteすればエラーとしてクライアントに返す
    // 何もしなければ正常に通知完了
})
cli := NewClient(s.Addr)
msg := newMessage()

que := make([]<-chan error, b.N)
for i := 0; i < b.N; i++ {
    que[i] = cli.Go(msg)
}
```

result.

- 並列数によるが10〜30秒でスケールするようになった
- グラフは15並列程度
- 2〜300万あたりならおそらく同じ程度



受託開発とGo

Goって大丈夫？

- 受託開発の場合、Go導入は難しくみえる
    - 技術者数が少ない
        - 規模の大きな開発に耐えられない
- 2年後3年後もサポートされているか不透明
- 実績が少ない

だけどメリットも

- 環境構築が簡単
    - バイナリだけで動作する
        - →本番環境でライブラリ管理が不要
- WindowsでもLinuxでも動作する
- 今年流行る言語にノミネート
    - 流行った頃に自慢できる、かも？

Goで良かったことはあっても

Goだから困ったことは無い

受託でもGoを流行らせていこう

ありがとうございました