スレッドプログラミングによるHTTP/2の実装

山本和彦

@kazu_yamamoto

概要

- HTTP/2
- スレッドとイベント駆動(コールバック)
- 軽量スレッドによるHTTP/2の実装
- ストリームの優先順位

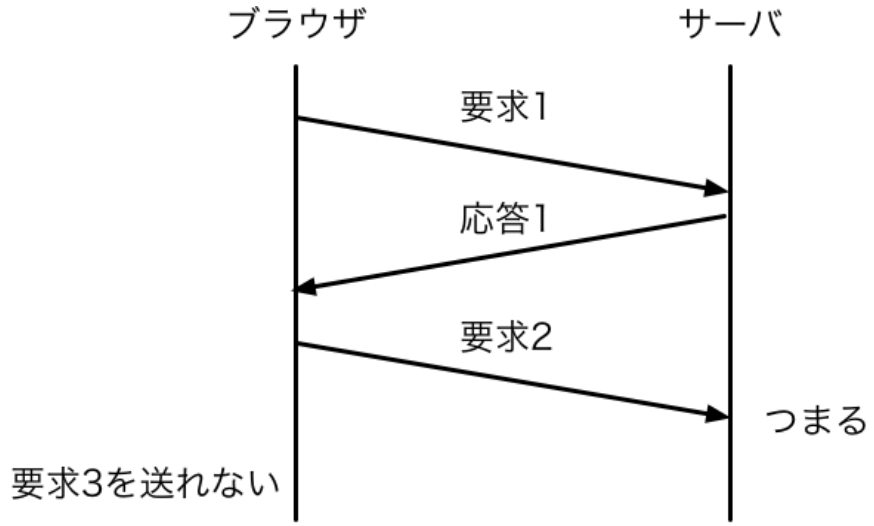
HTTP/2

HTTP/1.1 の問題点

同期性	Head-of-line ブロッキング
低い並列性	ドメインシャーディング
非効率なヘッダ	帯域の浪費

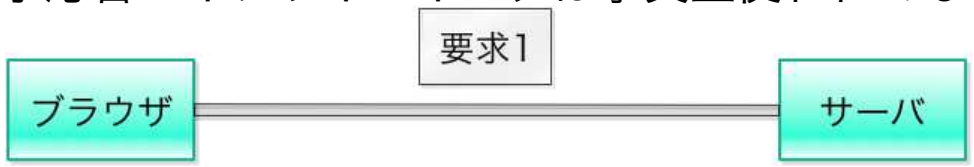
同期性

- HTTP/1.1 は同期的なプロトコル
  - サーバ：　　ある要求を処理した後、応答を返す
  - クライアント：応答が返ってきたら、次の要求を出す
- Head-of-line ブロッキング
  - ある要求への応答に時間がかかると、すべての処理が止まる

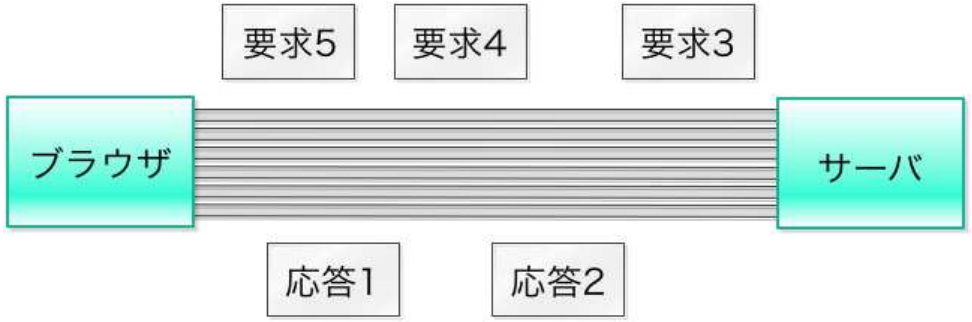


低い並列性

- 1つのコネクション上では高々1つの仕事
  - 要求が1つ、応答が1つ、なにもない
  - 要求応答パイプラインイングは事実上使われてない

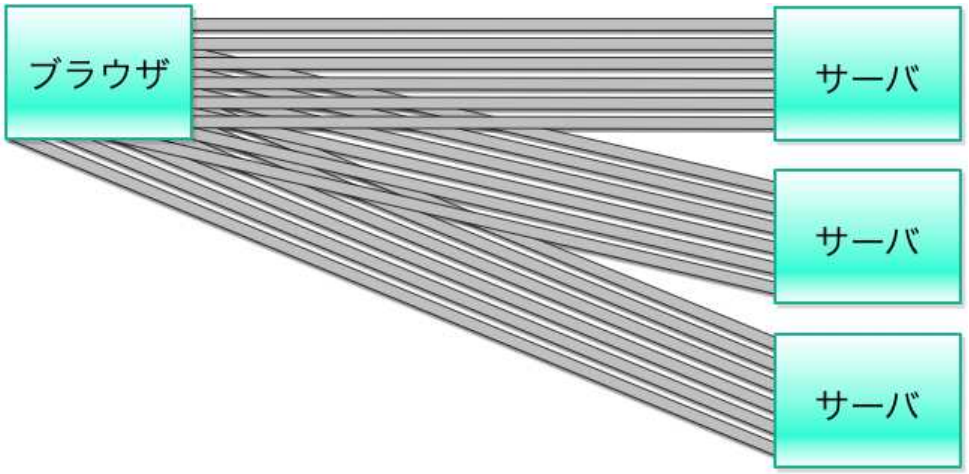


- 仕様上作れるコネクションの数はオリジンごとに6つ



ドメインシャーディング

- オリジンを増やして並列性を上げる
  - コンテンツが分散し、管理が困難になる



非効率なヘッダ

- 帯域の浪費
  - 要求ヘッダの長さの平均は800バイト
  - 似通った要求ヘッダが毎回送られる

```
GET /roversync/ HTTP/1.1
Host: rover.ebay.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8;
 rv:16.0) Gecko/20100101 Firefox/16.0
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.ebay.com/
Cookie: ebay=%5Esbf%3D%23%5E; dp1=bpbf/%2380000000000055
 276504d^u1p/QEBfX0BAX19AQA**5276504d^; cssg=c67883f113a
 0a56964e646c6ffaa1abe; s=CgAD4ACBQlm5NYzY3ODgzZjExM2EwY
 TU2OTY0ZTY0NmM2ZmZhYTFhYmUBSgAYUJZuTTUwOTUxY2NkLjAuMS4z
 LjE1MS4zLjAuMeN+7JE*; nonsession=CgAFMABhSdlBNNTA5NTFjY
 2QuMC4xLjEuMTQ5LjMuMC4xAMoAIFn7Hk1jNjc4ODNmMTEzYTBhNTY5
 NjRlNjQ2YzZmZmFhMWFjMQDLAAFQlSPVMX8u5Z8*
```

HTTP/2

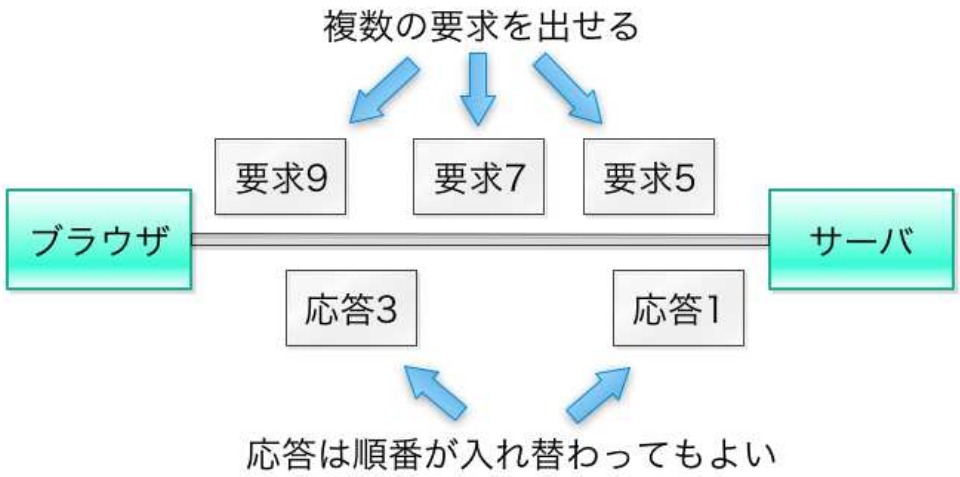
- 2012年9月からIETF で標準化がスタート
- 4つの提案の中から SPDY がベースに選ばれた
- 3年間の議論
- 2015年5月にRFCになった
  - RFC7540: Hypertext Transfer Protocol Version 2 (HTTP/2)
  - RFC7541: HPACK: Header Compression for HTTP/2
- 特徴
  - HTTP ヘッダなどの意味は変わらない
  - トランスポートのみが変更され効率的になった

HTTP/2による解決

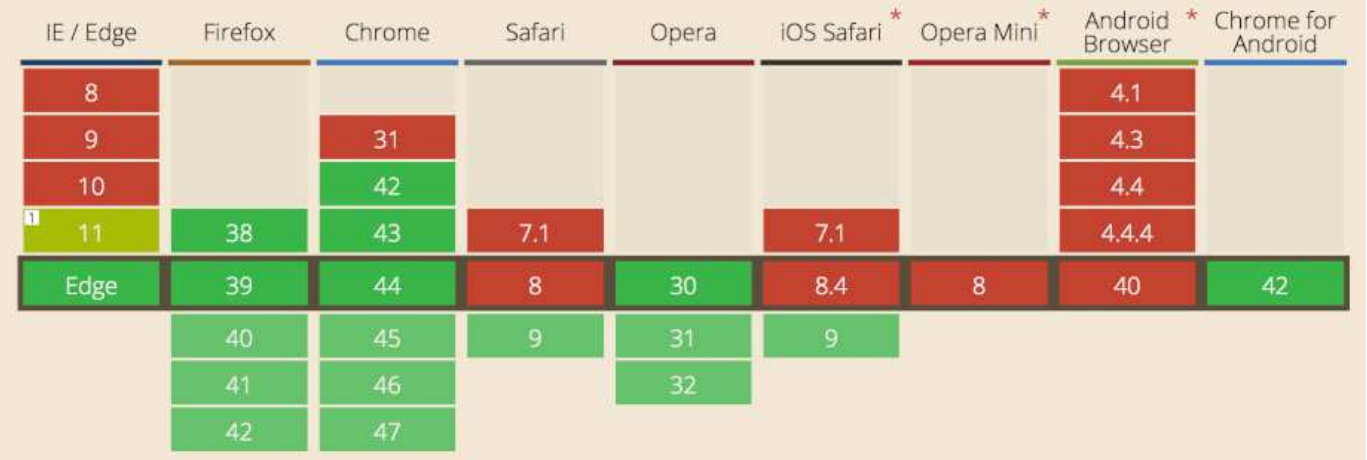
HTTP/1.1	HTTP/2
同期性	フレームによる非同期性
低い並列性	フレームによる 高い並列性
非効率なヘッダ	ヘッダ圧縮

フレームの非同期性と高い並列性

- 高い並列性
  - 利用されるコネクションは1つ
  - 複数のフレームが多重化される
- 非同期性
  - 準備ができた応答から返してよい



ブラウザでのサポート



http://caniuse.com/#feat=http2

サービスでのサポート

- Google や Twitter はすでに HTTP/2 をサポート

Firefox の開発ツールで Google との通信を表示

スレッドとイベント駆動

お題

HTTP/1.1 サーバの実装を考える

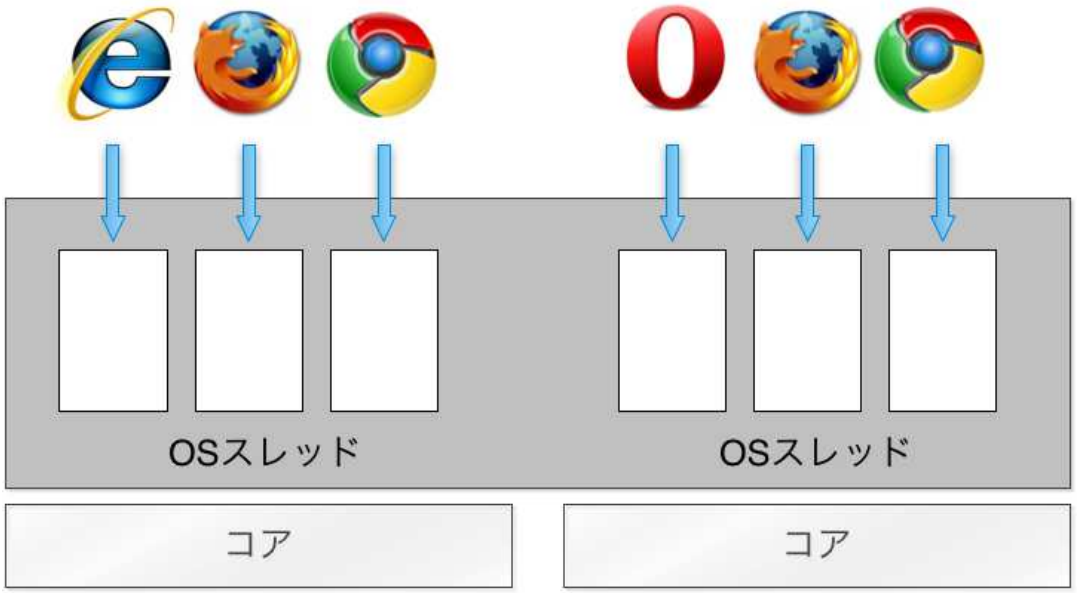
クライアントが1つの場合

- 見通しのよい一直線のコードが書ける

"On the duality of operating system structures", 1978

イベント(メッセージパッシング)と
スレッド(プロセスベース)は双対である

OSスレッドを使って複数の場合に対応



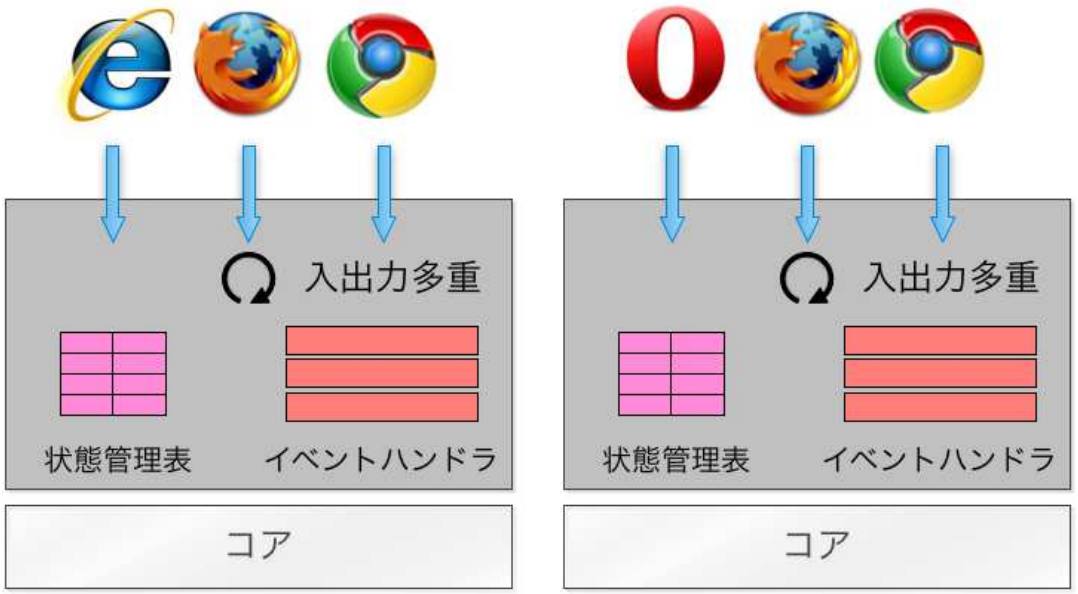
- 利点：見通しのよいコードを再利用できる
  複数のコアを活用できる
- 欠点：OSスレッドの切り替えが多発し遅い

"Why Threads Are A Bad Idea", 1996

OS スレッドはデッドロックに
陥りやすく性能も低い

代替品はイベントである

イベント駆動を使って複数の場合に対応



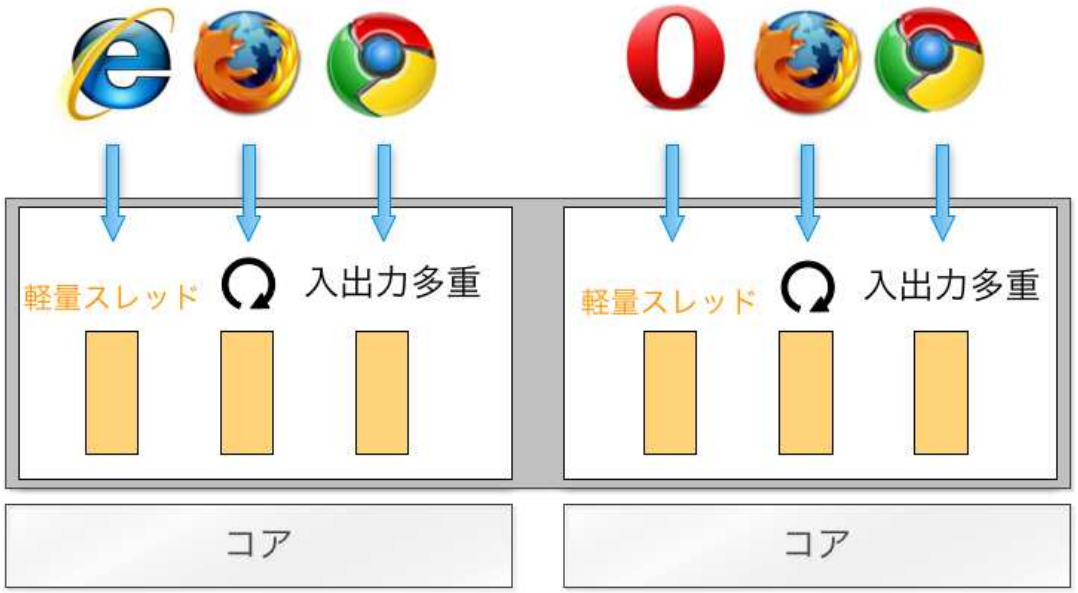
- 利点：コアの性能を引き出せる
  - 複数のコアを利用するには prefork してポートを共有
- 欠点：見通しの悪いコードへ作り直し
  - preforkの欠点：共有メモリがない (レート制御などの実現が難しい)

"Why Events Are A Bad Idea", 2003

イベント駆動は制御構造と状態管理が困難

軽量スレッドは言語のサポートが鍵

軽量スレッドを使って複数の場合に対応



- 利点：見通しのよいコードを再利用できる
- 利点：コアの性能を引き出せる

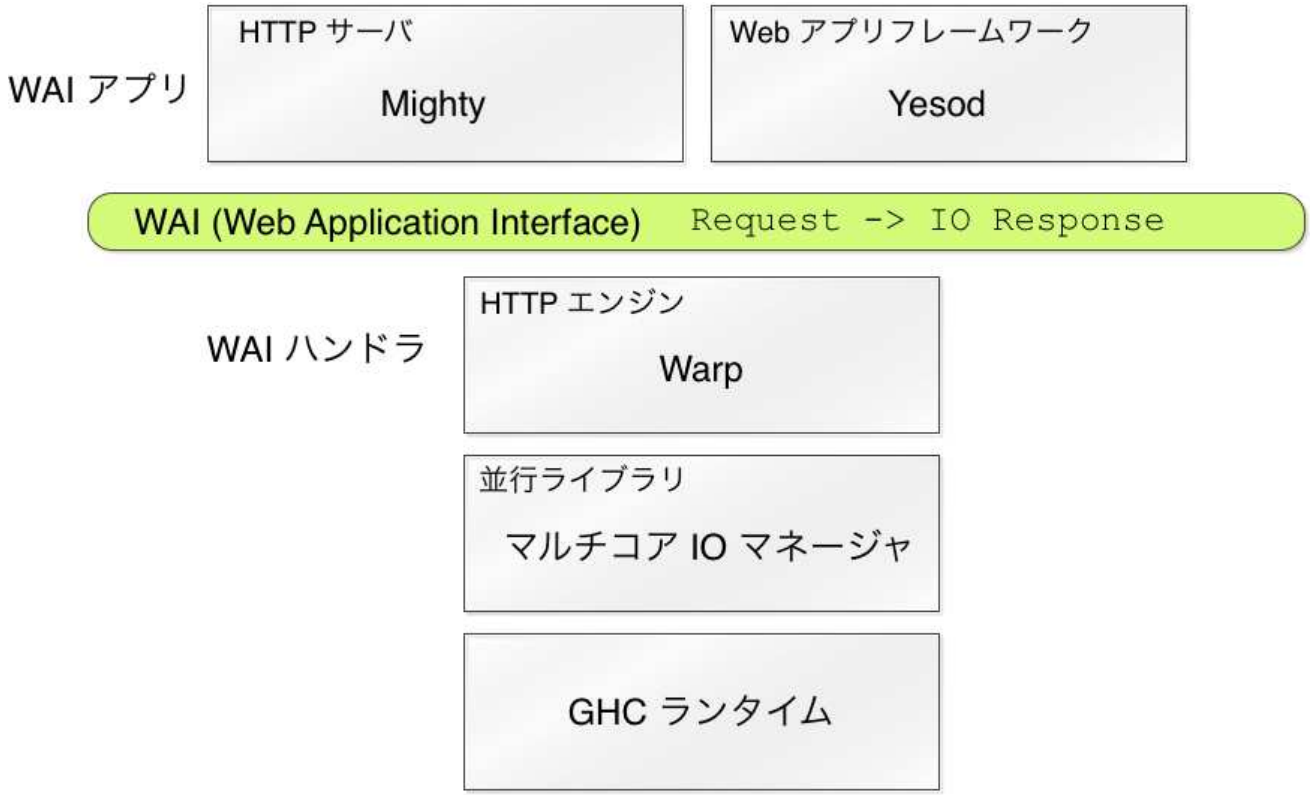
複数のクライアントを扱う技術のまとめ

OS スレッド	クライアント一つ用と複数用のコードが同じ 一直線のコードを書ける 低速
イベント駆動	クライアント一つ用と複数用のコードが異なる コードを分断し状態を管理 高速
軽量スレッド グリーンスレッド	クライアント一つ用と複数用のコードが同じ 一直線のコードを書ける 高速

軽量な構成要素とプログラミング言語

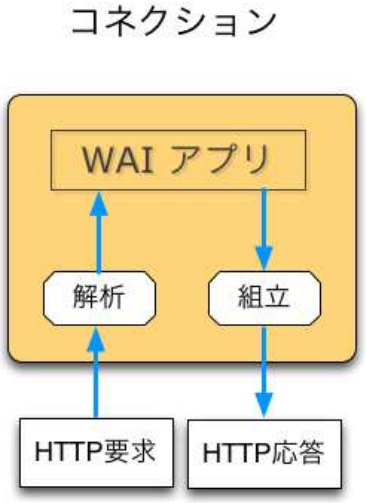
Haskell 軽量スレッド	Go goroutine(高機能コルーチン)
Erlang 軽量プロセス	Rust タスク(軽量プロセス)

Haskell の WAI と Warp



Warp での HTTP/1.1 の実装

- HTTP/1.1ではコネクションごとに軽量スレッドを起動する




```
loop {
  req = receiveRequest();
  rsp = application(req);
  sendResponse(rsp);
}
```

- 軽量スレッドは同期的なアプリケーションプロトコルと相性がよい

疑問

軽量スレッドは
非同期なトランスポートも含む
プロトコルにうまく適用できるか？

スレッドプログラミングと HTTP/2

実装のスケルトン

- Warp を HTTP/2 へ拡張する

要求

- 各HTTP/2ストリームを軽量スレッドで処理する
  - HTTP/2ストリーム = クライアントからの要求 + サーバからの応答
- 最大並列ストリーム数
  - SETTINGS_MAX_CONCURRENT_STREAMS
  - クライアントは同時にこの値までの要求を送信できる
  - 100以上にすることが推奨されている
- 軽量スレッドの生成数に上限を持たせたい
  - 出力キューへの競合を小さくしたい
- 軽量スレッドを生成するオーバヘッドをなくしたい
  - 生成コストは 0 ではない

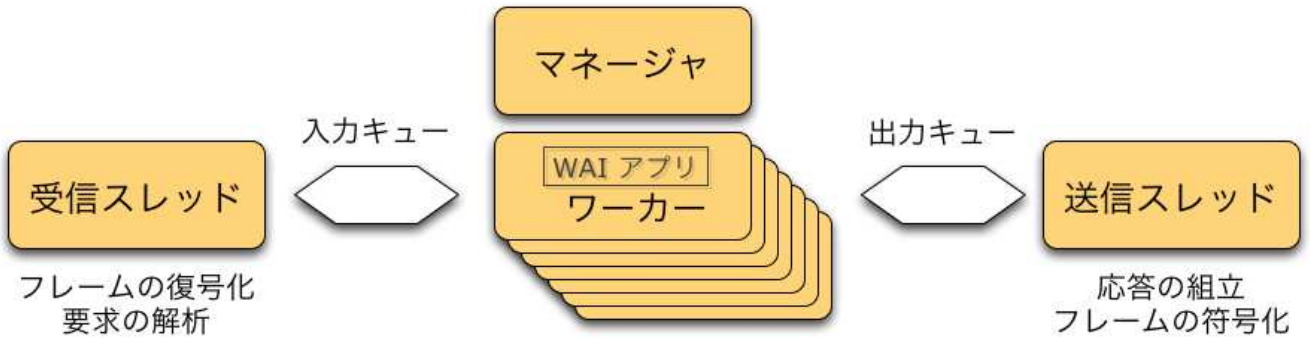
アイディア

Haskell ではあまり利用されてない

ワーカースレッドプールを導入する

Warpでの HTTP/2 の実装

- ワーカープール
  - コネクション確立時に、受信スレッド、送信スレッド、ワーカープールのマネージャを生成
  - マネージャは 10 個のワーカーを生成

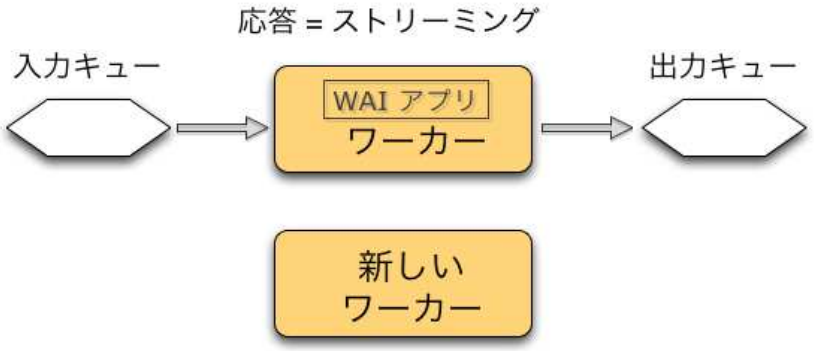


ワーカーを占有しない応答

- 以下を繰り返す
  - 入力キューから要求を取り出す
  - WAIアプリを起動
  - 応答を出力キューへ入れる

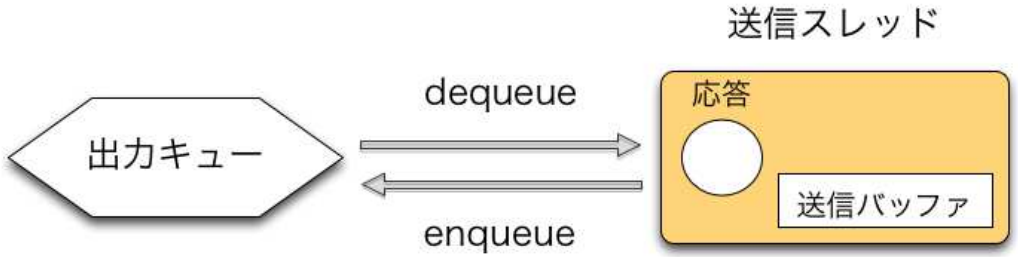
ワーカーを占有する WAI アプリ

- 応答がストリーミングだった場合
  - マネージャに新しいワーカーの作成を依頼
  - ストリーミングの処理
  - 終了



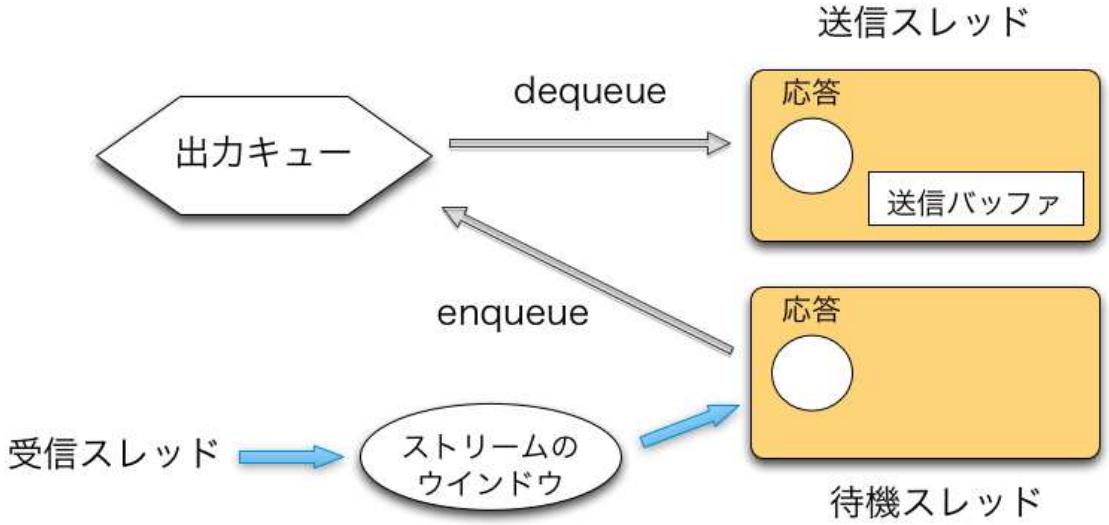
送信スレッド

- 以下を繰り返す
  - 出力キューから応答を取り出す
  - 送信バッファに応答を詰め込んで送信する
  - 残りの作業を出力キューに入れる



ウインドウと待機スレッド

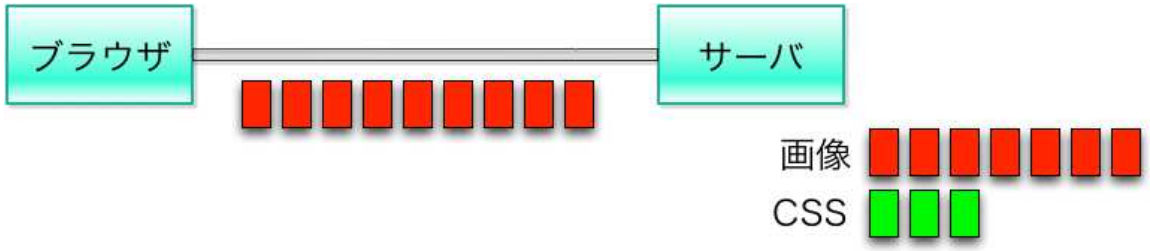
- フロー制御
  - 送信スレッドは出力キューから応答を取り出す
  - ストリームのウインドウが0の場合、
    送信スレッドは待機スレッドを生成する
  - 待機スレッドは、受信スレッドからの通知を待ち、
    ウインドウが開いたら応答を出力キーに入れる



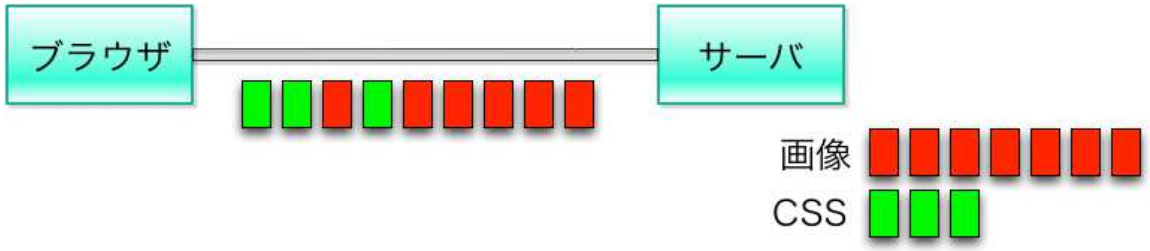
ストリームの優先順位

優先順位が必要な理由

- 優先順位がないと大きなコンテンツが
  コネクションを占有する危険性がある

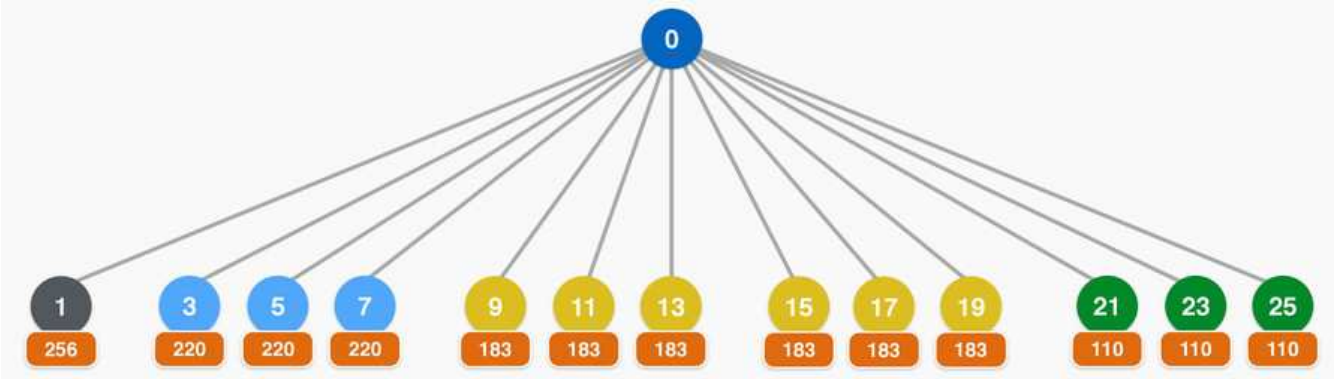


- 優先順位を付けレンダリングに必要な
  コンテンツを優先的に受け取りたい



優先順位と重み

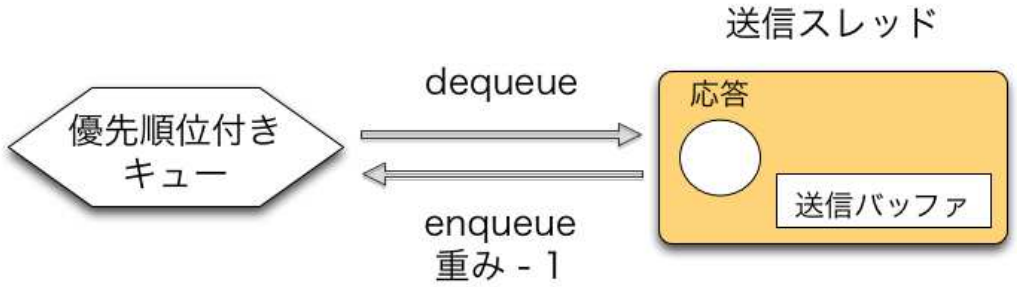
- ストリームは、他のストリームに依存する
  - ストリーム 0 は、ルートストリーム
- 子供のストリームは、親のストリームの資源を重みに応じて分けあう
  - 重み 1〜256
- Chrome が指定する優先順位の例
  - ストリーム3は、220/2088 = 55/522 の資源が与えられる



https://speakerdeck.com/summerwind/2-deep-dive-priority-and-server-push

優先順位付きキューと重み

- 重みを優先順位として優先順位付きキューを使うと公平性がない



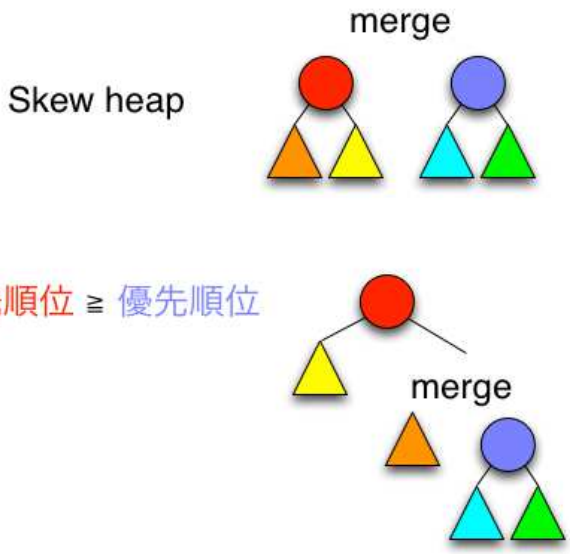
- 重み A:201 と B:101 の場合
  - A:201, A:200, A:199, ..., A:101, B:101, A:100, B:100, ...

優先順位付きキューへの要求

- 資源 = 送信機会とする
- 公平性の担保
  - 重みに応じて出力機会を得る
  - A の重みが W_A、重みの合計が W_{tatal} のとき、A は W_A / W_{total} の確率で処理されるべき
- 性能：操作のオーダーは O(log n)以下

Skew ヒープ

- 優先順位付きキューとしてヒープが利用できる
- Skew ヒープ
  - 平衡情報を持たないがよく平衡するヒープ
  - 2つの子を FIFO として使う



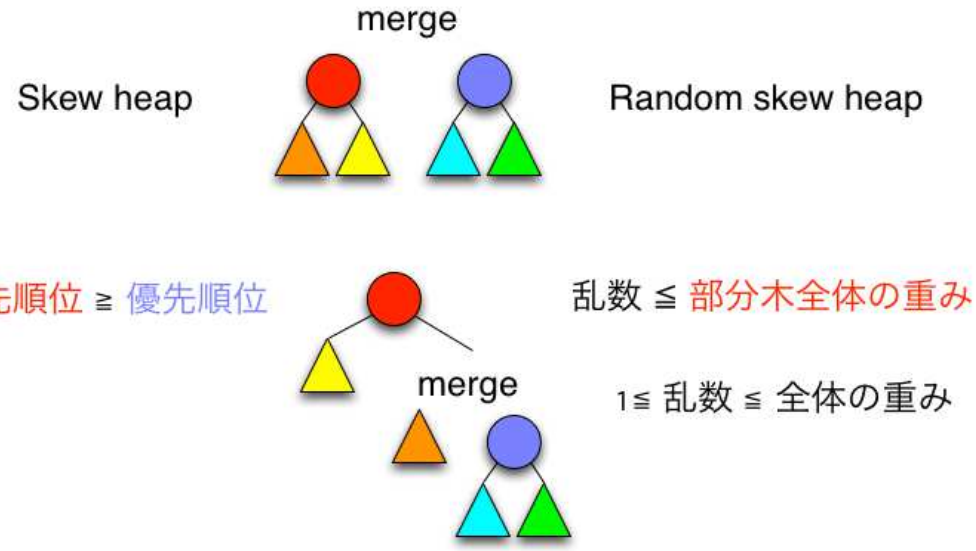
Skew ヒープのままでは公平性がない

アイディア

乱数を使う

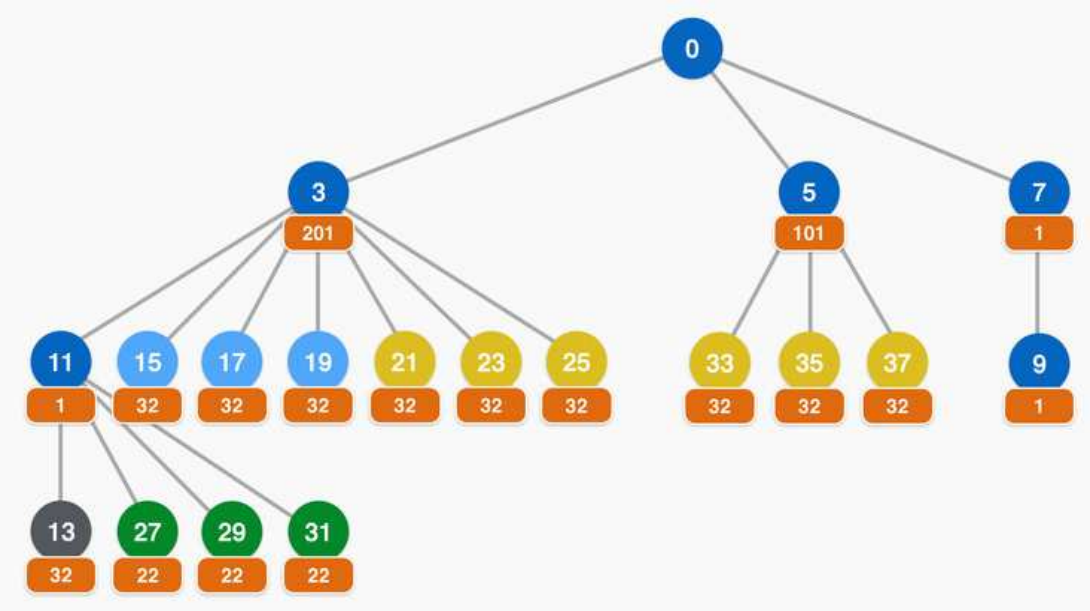
乱択ヒープ

- 乱択平衡探索木
  - 平衡情報を持たないがよく平衡する探索木
  - 大きさNの木に要素を挿入するとき、1/N の確率で根とする
- Skewヒープの乱択化



優先順位付きキューの階層

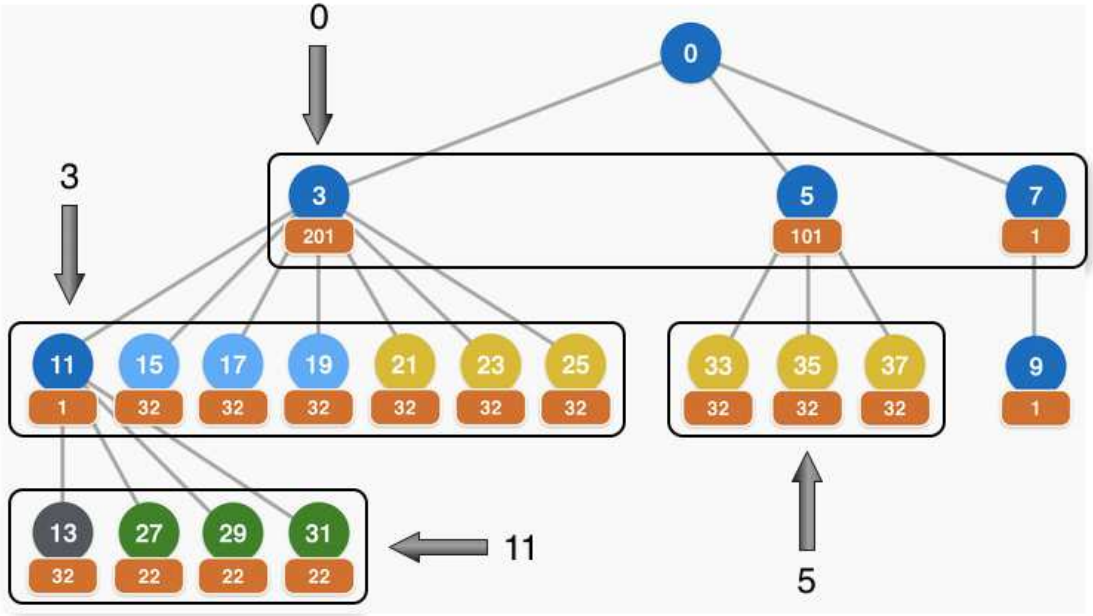
- 依存関係は階層化できる
- Firefox が指定する優先順位の例
  - Firefox は番号の小さいストリームを階層を作るためだけに利用



https://speakerdeck.com/summerwind/2-deep-dive-priority-and-server-push

階層の実装

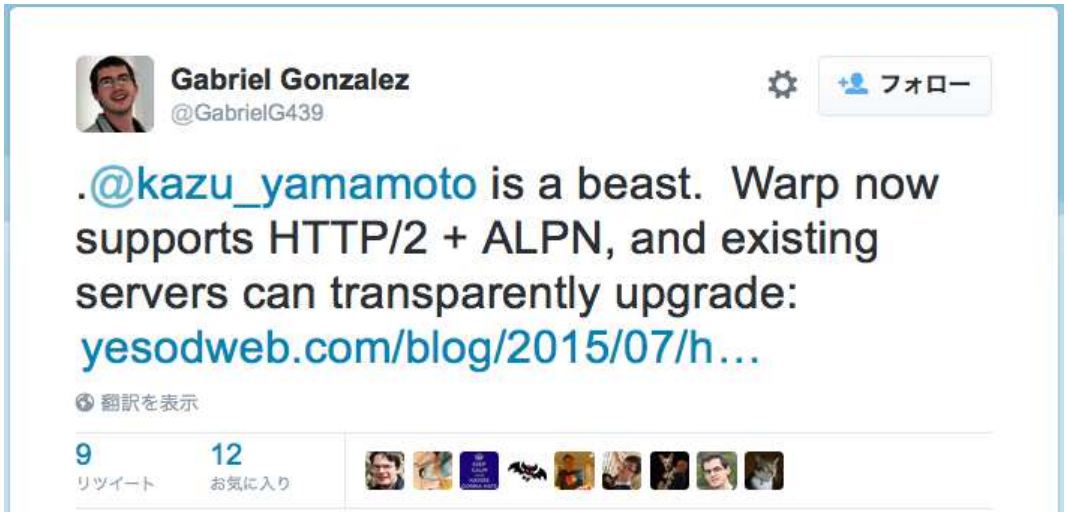
- 共通の親を持つ子供たちを同じ乱択ヒープに入れる
- 階層は有限マップで管理



成果物

http2 ライブラリ	HPACK フレームの符号/復号 優先順位付きキュー
tls ライブラリ 拡張	ALPN ECDHE AES GCM
warp ライブラリ 拡張	入力の二段階のバッファリング 送信時のパケット詰め込み フロー制御、優先順位

反響



[–] **bos** **12ポイント** 17日前

Kazu knocks it out of the park yet again. Very impressive work.

固定リンク

まとめと今後の課題

- 内容
  - HTTP/2の概要
  - 軽量スレッドによる HTTP/2 の実装
  - 優先順位付きキューとしての乱拓ヒープ
- 課題
  - コードの高速化
    - TLS
  - ユーザ体験の高速化
    - サーバプッシュ
    - 優先順位