


# 入門 HTTP

上岡輔乃

# 自己紹介

---

- 上岡 輔乃（うえおか ほない）
- CAMPHOR- 運営メンバー
- Webフロントが好き
- 好きな果物:みかん 

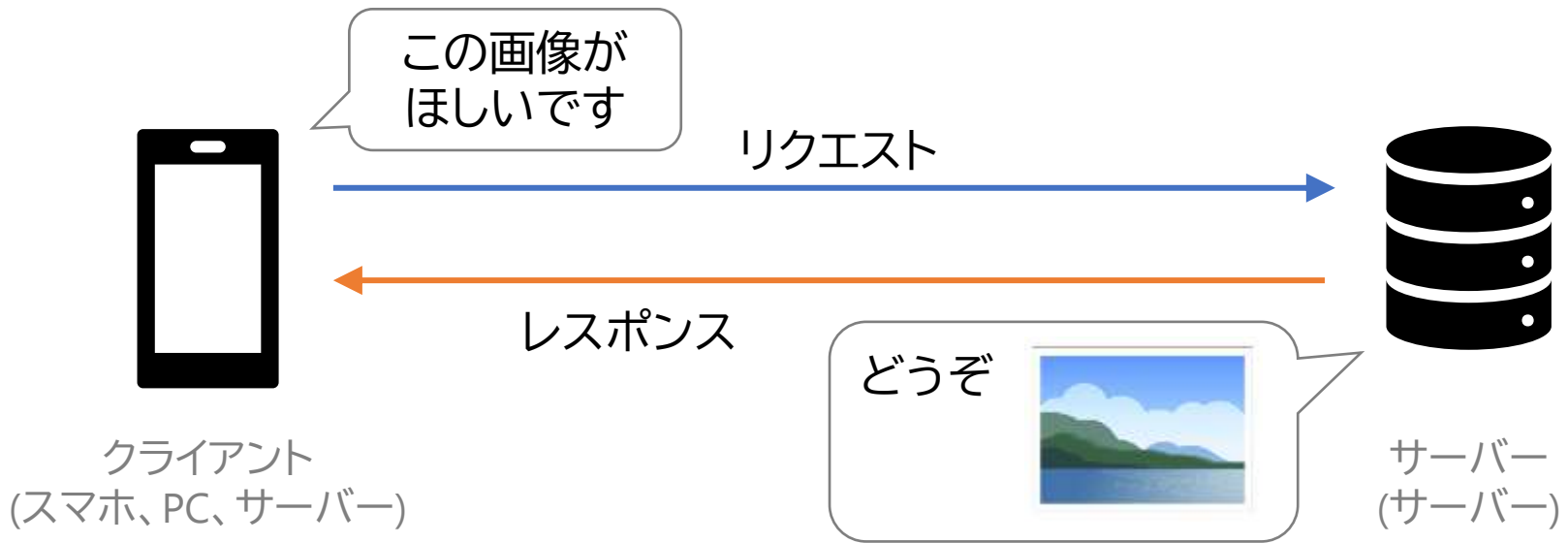


@\_honai

HTTP、使ってますか

---

# HTTP: Hyper Text Transfer Protocol



HTML文書に限らず、Web上の多様なコンテンツを  
転送するためのプロトコル

<https://youtube.com/>

# HTTPの歴史

---

- (1991年 HTTP/0.9)
- 1996年 HTTP/1.0
- 1997年 HTTP/1.1 ← 現役！
- 2010年 ChromeでSPDY(HTTP/2の前身)が有効に
- 2015年 HTTP/2、ChromeでQUICが有効に
- 2020年 (HTTP/3 draft-27)

# Webアプリをつくる人にとってのHTTP

---

- 基本概念は20年前にほぼ完成され、  
変わっていない
- ブラウザやフレームワーク、言語の標準機能で  
十分に抽象化されている
- HTTPの何が進化しているのかは  
あまり見えてこない

# “再”入門 HTTP

---

- 抽象化されたHTTPの背後にどんな処理があるのか
- HTTPの進化
  - 高速化
  - レイテンシの削減
  - セキュリティ向上
  - 変化するインターネット事情への対応

Webを支えるHTTPの進化を知ってほしい！

# トークの内容

---

- HTTPのおさらい
- HTTP/1.x
  - テキストベース
  - Keep Alive
- TLSとHTTP
  - TLSによるセキュアな通信
  - TLS1.3のハンドシェイク
- HTTP/2
  - バイナリベース
  - 多重化
  - ヘッダー圧縮
- QUIC と HTTP/3
  - UDP上の新しいプロトコル
  - TLSとの統合、0-RTT
  - Connectino Migration



# HTTPの基本要素 — リクエスト

```
GET /index.html HTTP/1.1
Host: www.example.com
Accept: */*
```

メソッドいろいろ

- GET
- HEAD
- POST
- PUT
- DELETE
- CONNECT
- OPTIONS

GET   /index.html   HTTP/1.1  
メソッド   パス

Host: www.example.com  
フィールド名1   値1  
Accept: \*/\*  
フィールド名2   値2

ヘッダー

# HTTPの基本要素 — レスポンス

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Length: 1256
```

```
<html>
  <head>
    <title>Example Domain</title>
    ...
```

HTTP/1.1 200 OK  
ステータス

```
<html>
  <head>
    <title>...
  ] ボディ
```

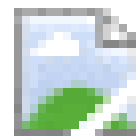


このページは動作していません

localhost では現在このリクエストを処理できません。

HTTP ERROR 500

500 Internal Server Error



404 Not Found

# HTTP/1.x

---

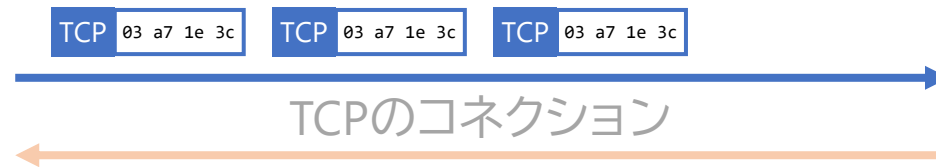
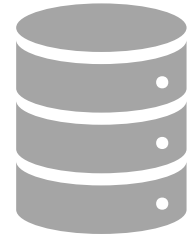
TCPを使ったテキストベースのシンプルなプロトコル

# HTTP/1.0 と HTTP/1.1

---



# HTTP/1.0 と HTTP/1.1



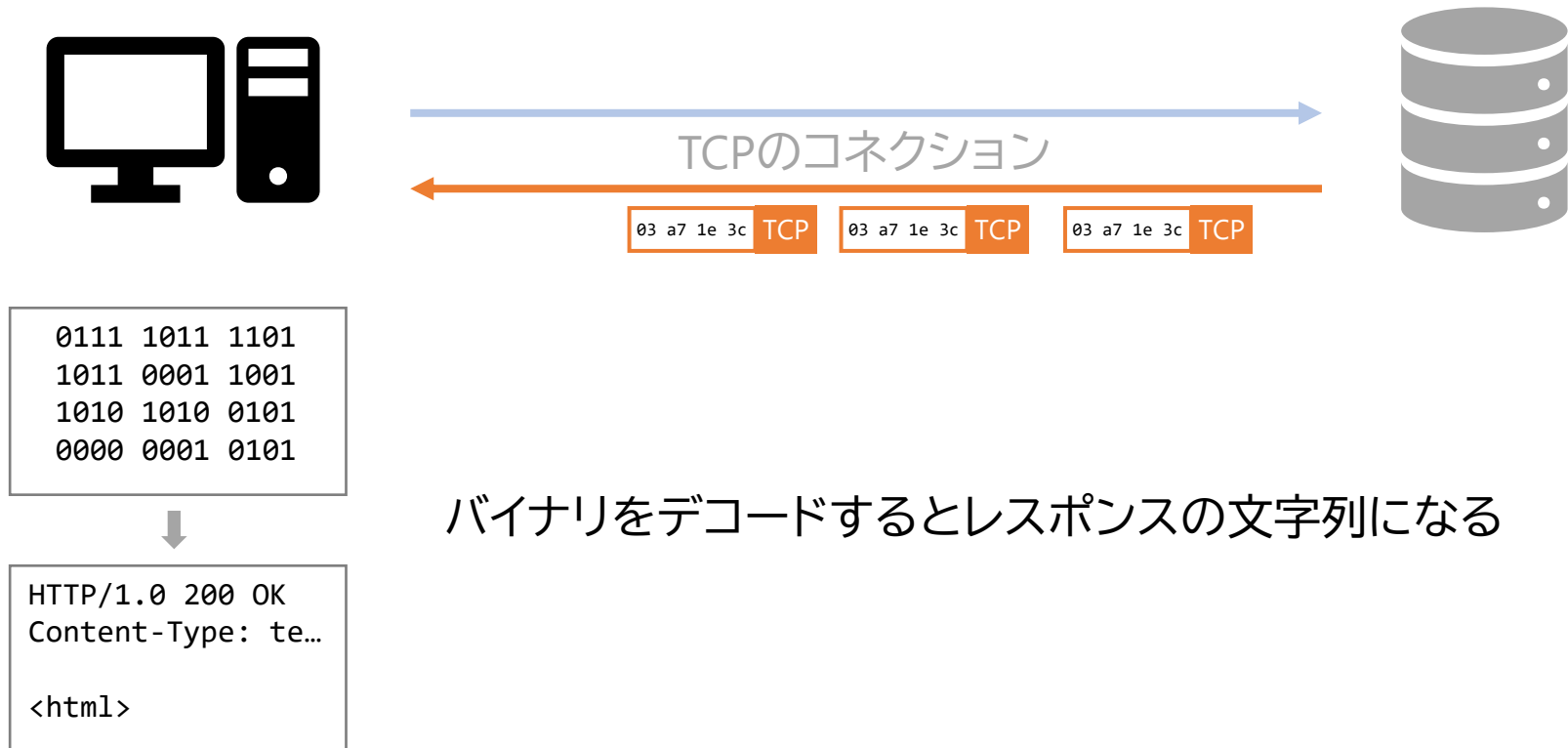
```
0111 1011 1101
1011 0001 1001
1010 1010 0101
0000 0001 0101
```



文字列をエンコード

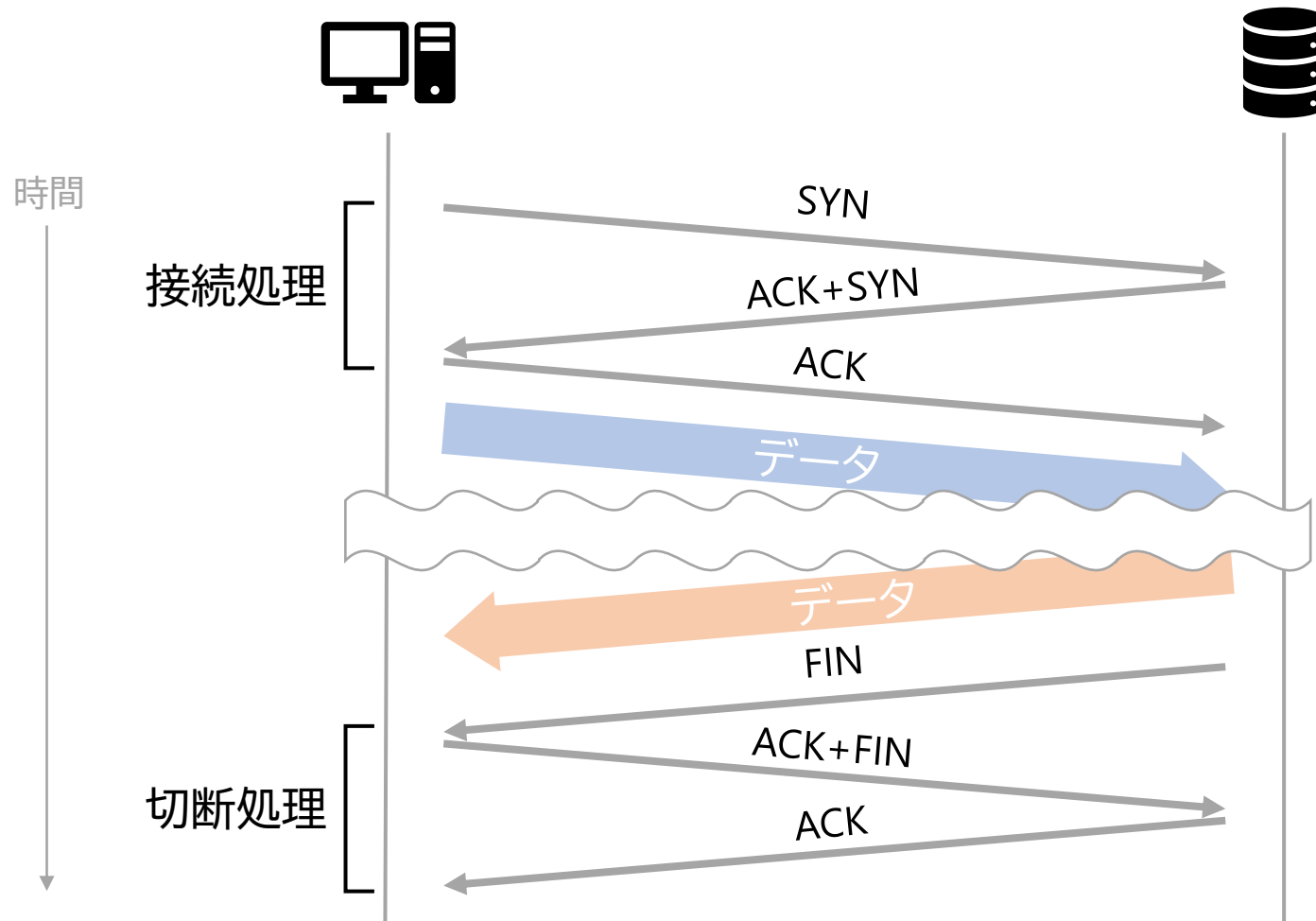
```
GET / HTTP/1.0
_____
_____
```

# HTTP/1.0 と HTTP/1.1



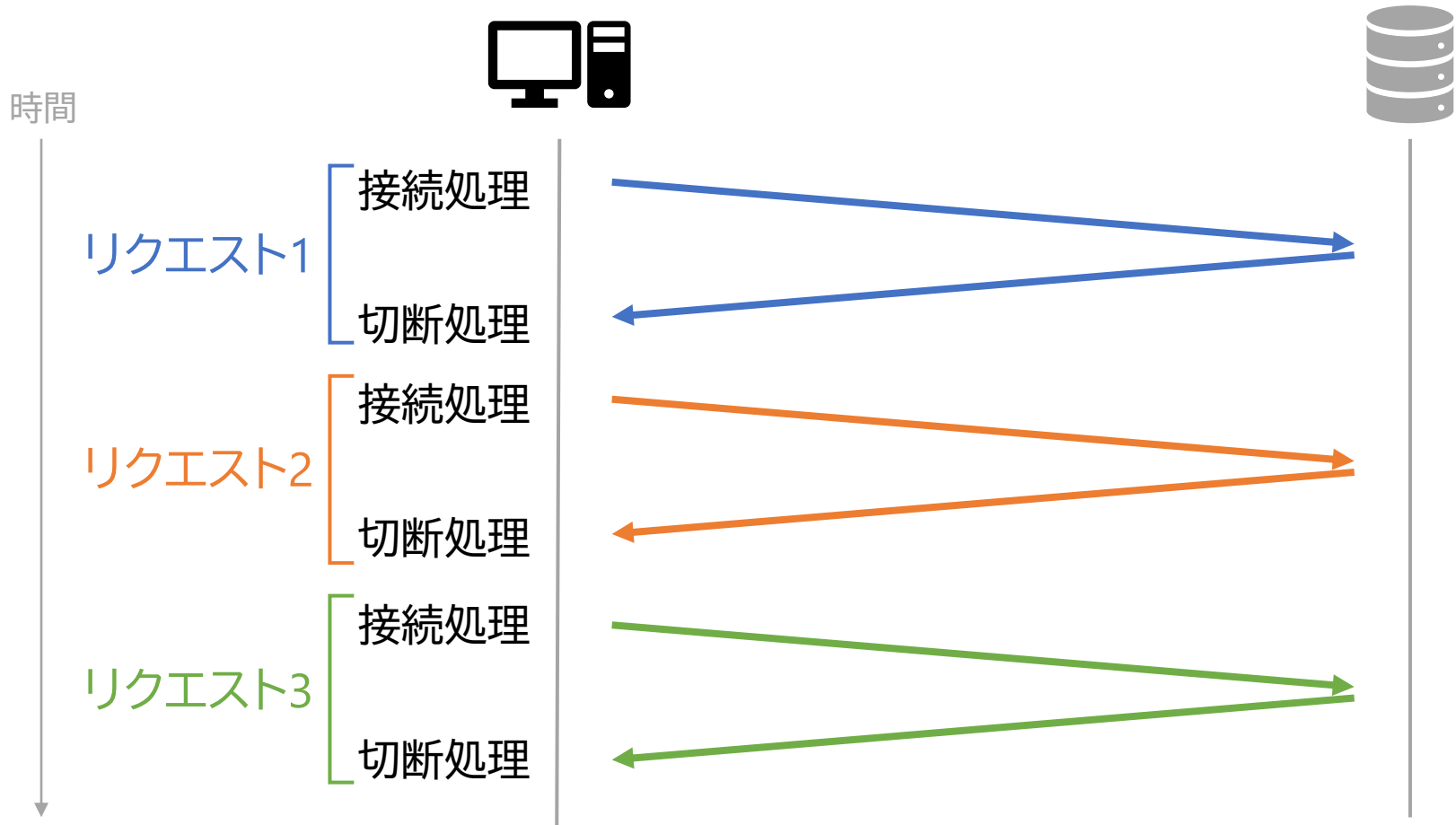
テキストベースのシンプルなプロトコル

# HTTP/1.x の課題 — 毎回接続すると時間がかかる



**TCP通信は接続/切断に1RTTかかる**

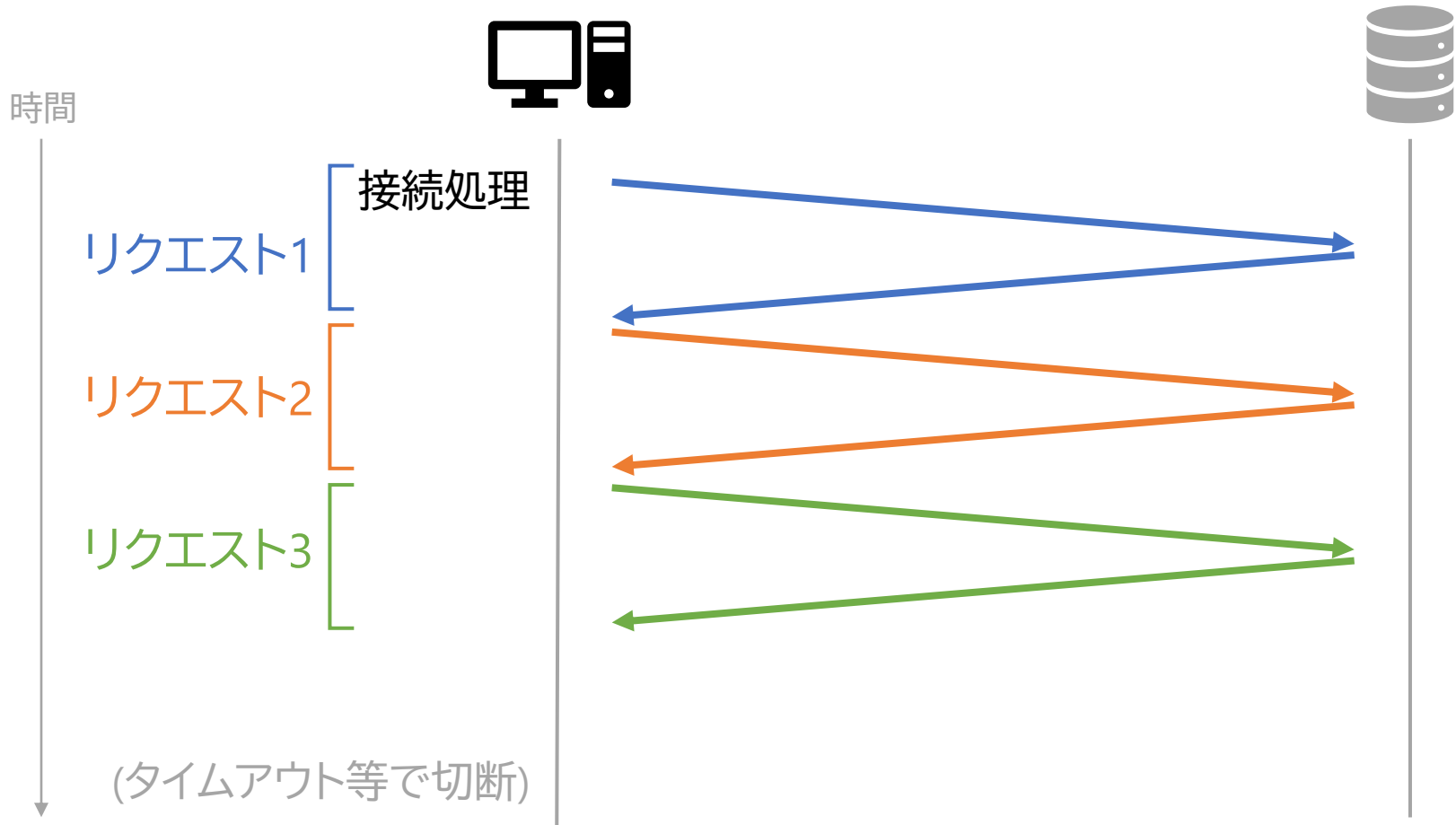
# HTTP/1.x の課題 — 毎回接続すると時間がかかる



リクエスト毎に接続/切断するので遅い



# Keep Alive



TCPの接続をつなぎっぱなしにして高速化

# Keep Aliveの効果を見てみましょう

```
// nginx.conf
server_1 {
    listen          8001;
    keepalive_timeout 0;
}

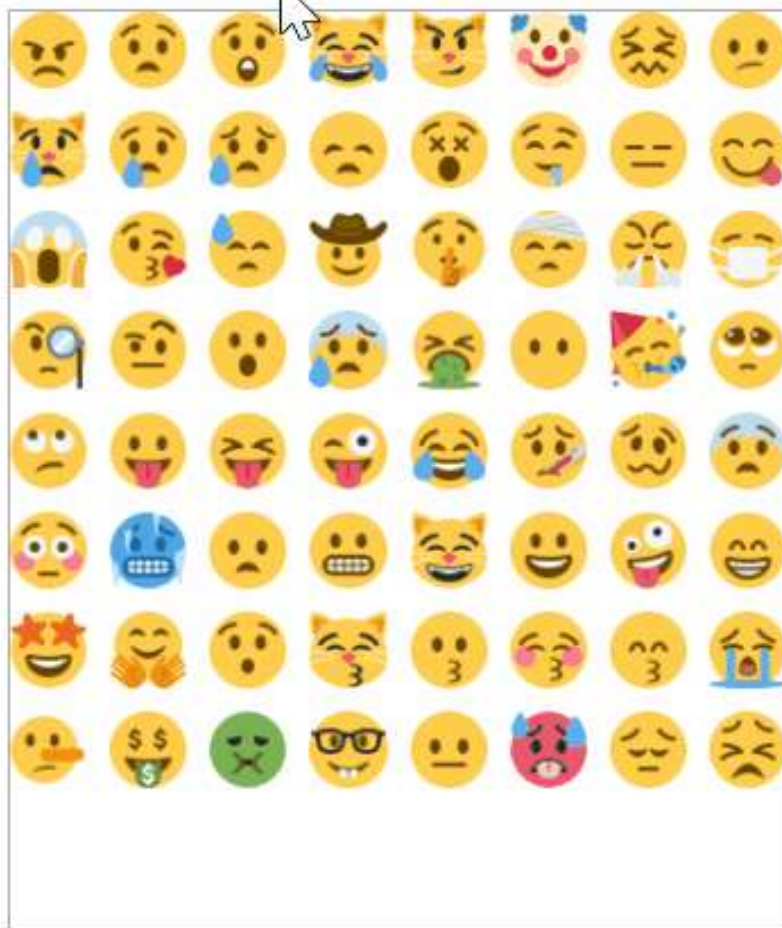
server_2 {
    listen          8002;
    keepalive_timeout 65;
}
```

nginxでサーバーを2つ立てる

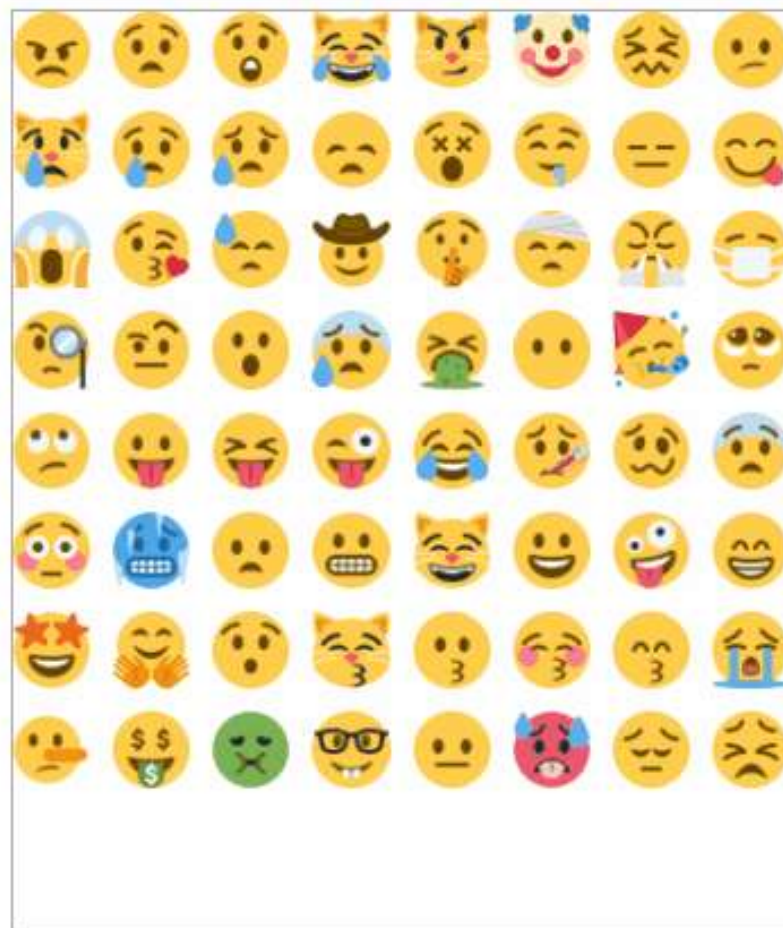
```
// index.html
<iframe src="http://localhost:8001" />
<iframe src="http://localhost:8002" />
```

iframeで両方を読み込む

## Keep Alive OFF



## Keep Alive ON

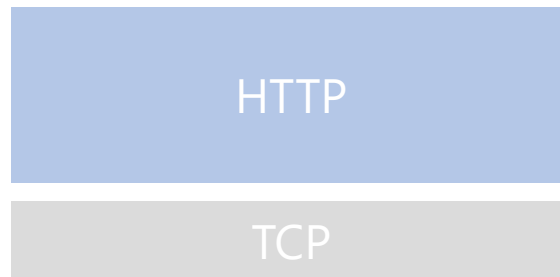
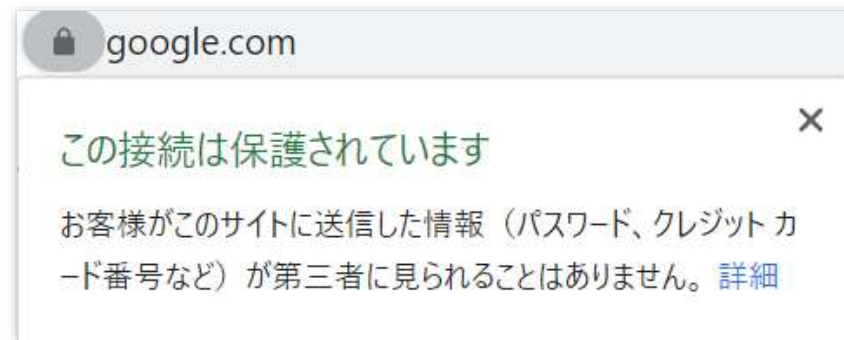


# TLS と HTTP

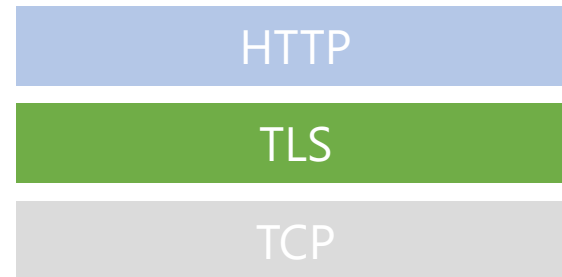
---

# TLS: Transport Layer Security

- 通信における、盗聴・改ざん・成りすましを防ぐ
- HTTPとTCPの間で利用することで、セキュアな通信を実現

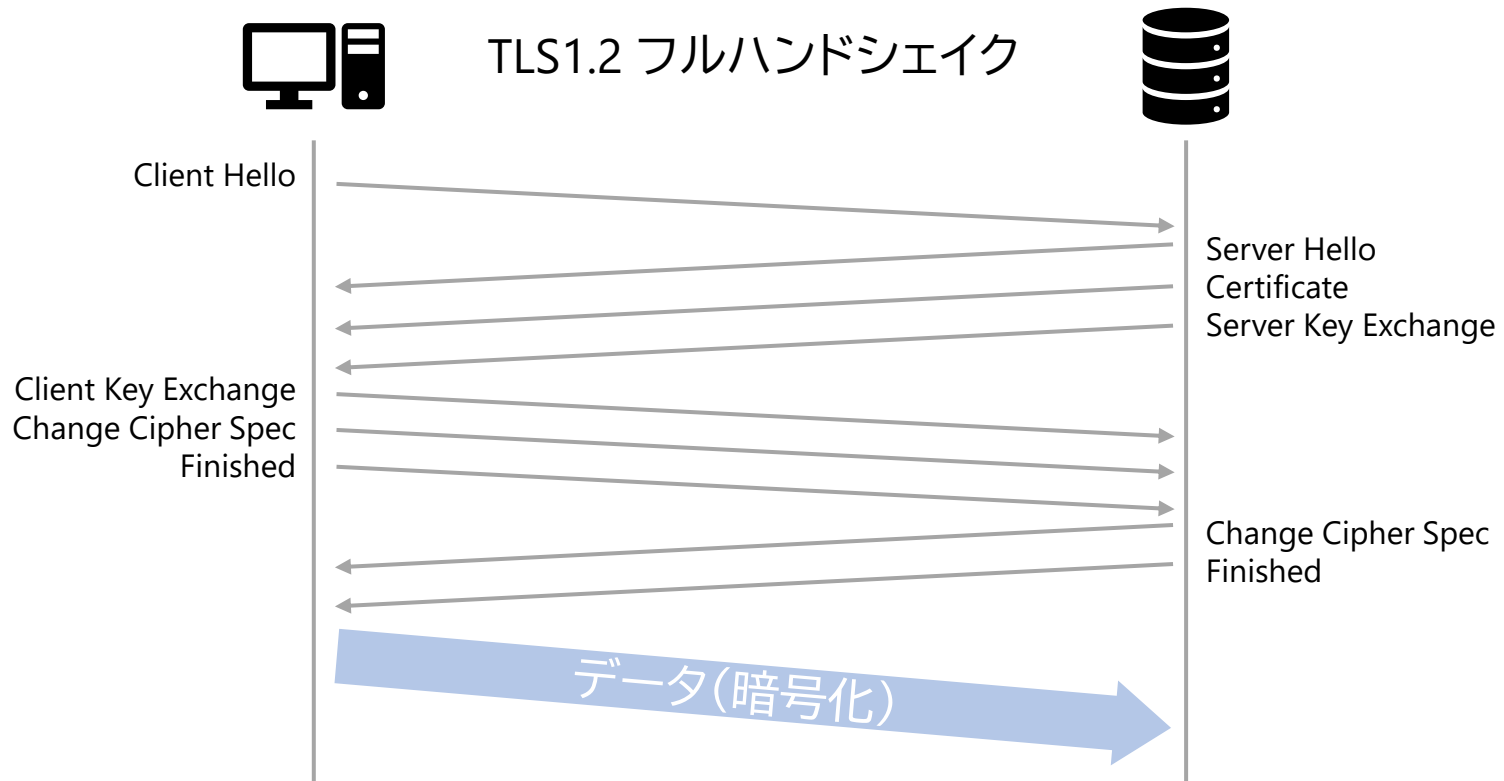


http://



https://

# TLSとハンドシェイク

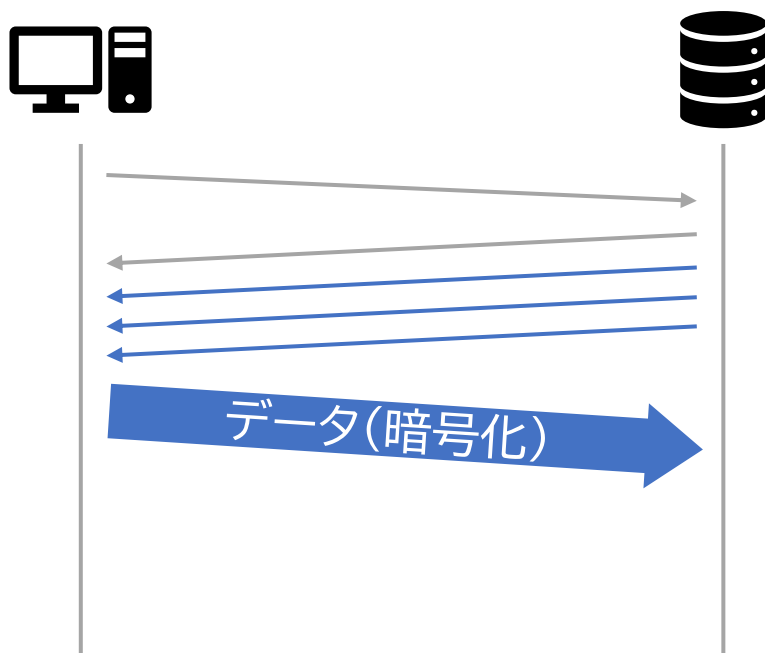


初回接続: 2RTT(上図)、再接続: 1RTT

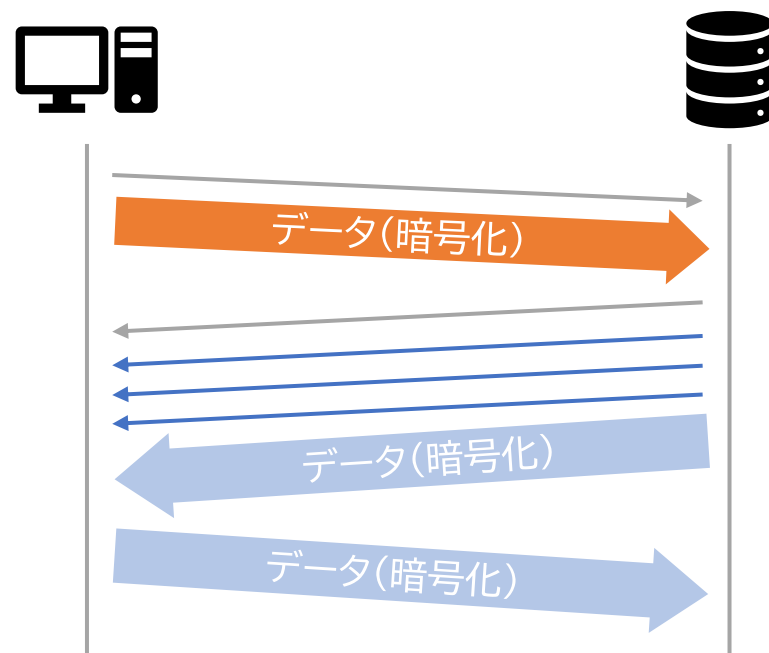
TLSで通信するにはハンドシェイクが必要

# TLS 1.3 — 2018年にRFC 8446として公開

フルハンドシェイク: 1RTT



再接続: 0RTT



HTTPのレイテンシ削減につながった

# HTTP/2

---

仮想的に多重化されたバイナリベースのプロトコル



# HTTP/1.x の課題 — 輻輳制御の効率が悪い

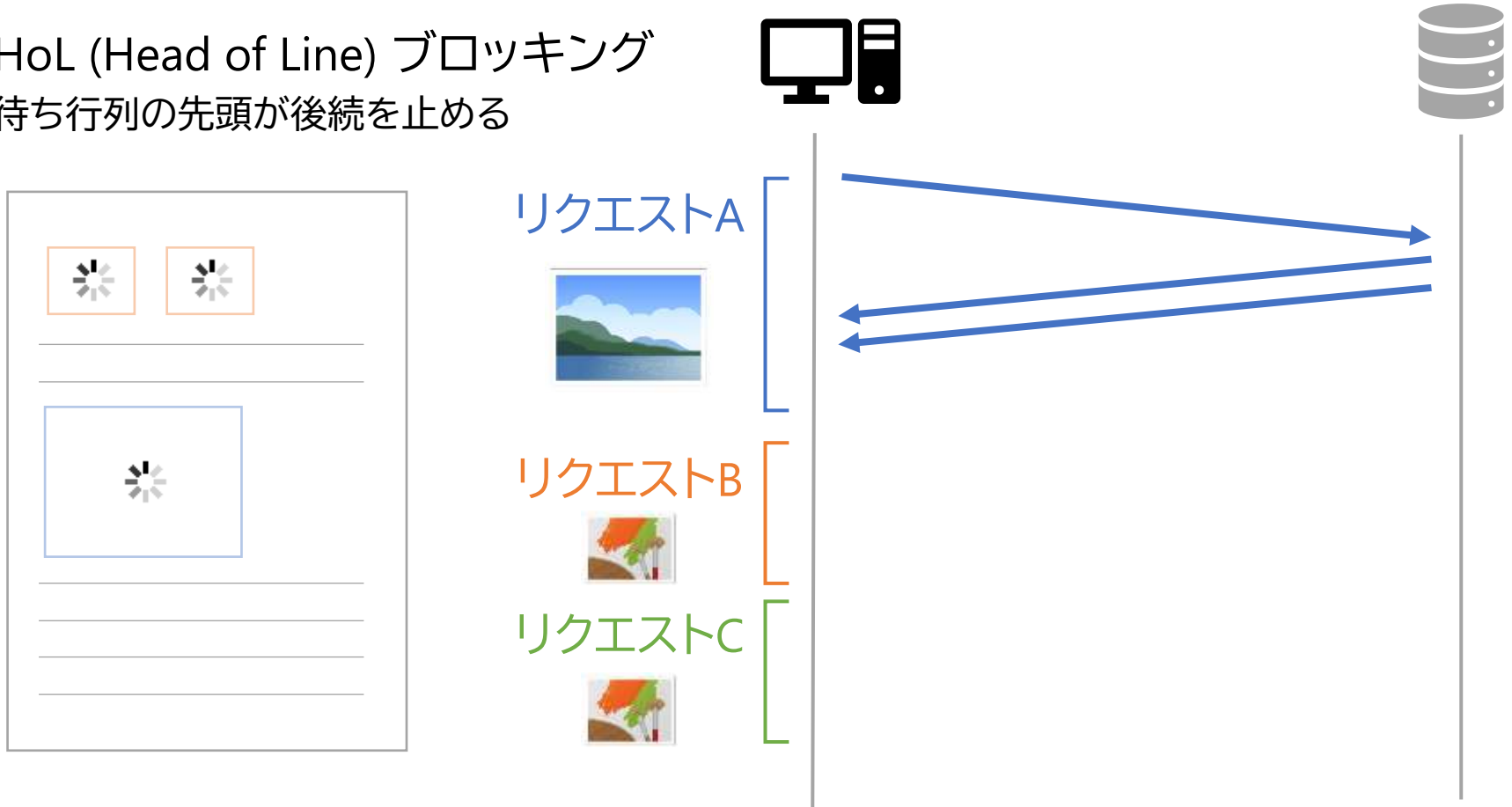
---

- 1つのドメインに対して6本ほどのTCP接続
- それぞれがばらばらに輻輳制御を行うので、効率が悪い

引用した図を削除しています

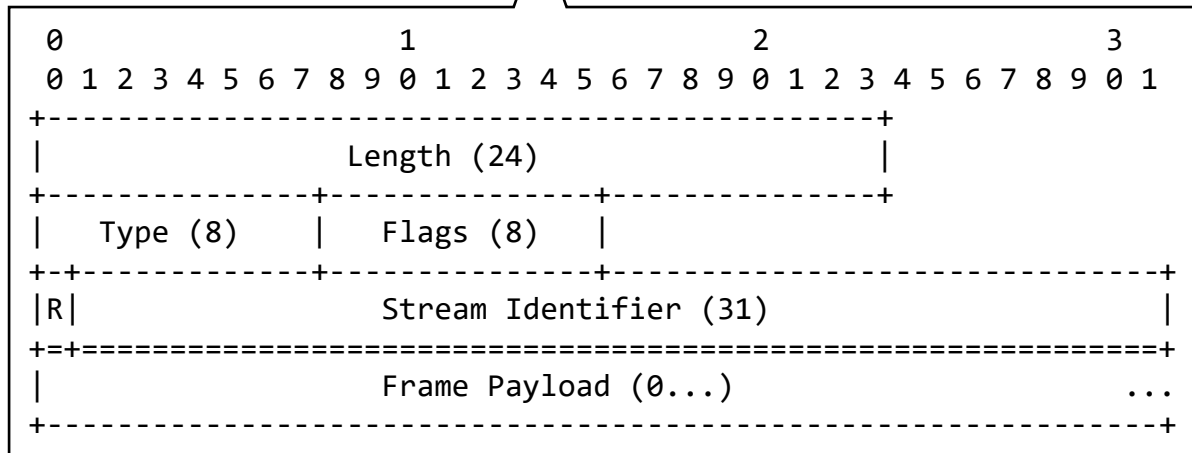
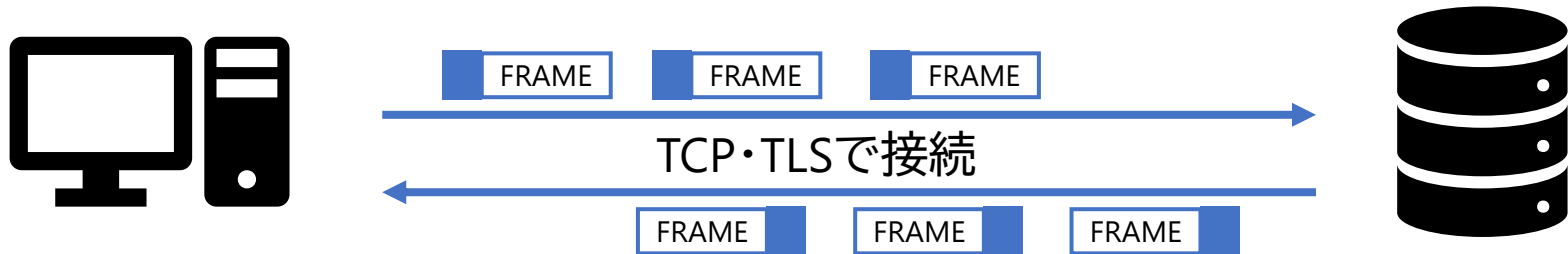
# HTTP/1.x の課題 — HTTP HoLブロッキング

HoL (Head of Line) ブロッキング  
待ち行列の先頭が後続を止める



1つの(重い)リクエストが  
他のリクエストをブロックしてしまう

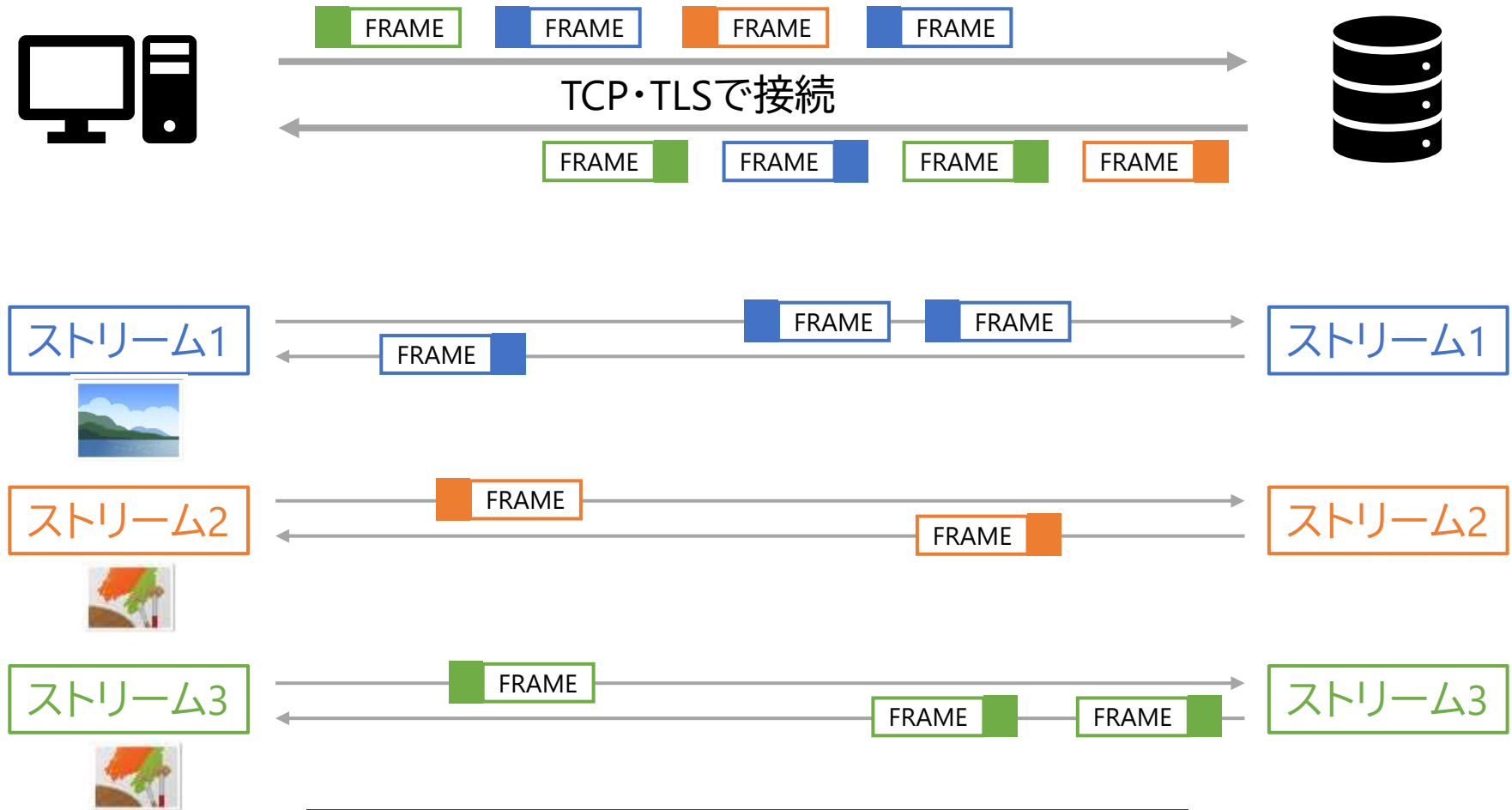
# HTTP/2 — バイナリベース



HTTP/2 Frame Layout

HTTPをフレームに分解して送受信する

# HTTP/2 — ストリームによる多重化



ストリームIDを振ることで  
仮想的に通信路を多重化する

# HTTP/2 — 輻輳制御が効率的

---

- 1本のTCP接続
- 輻輳制御がはたらいて効率的に帯域を使える

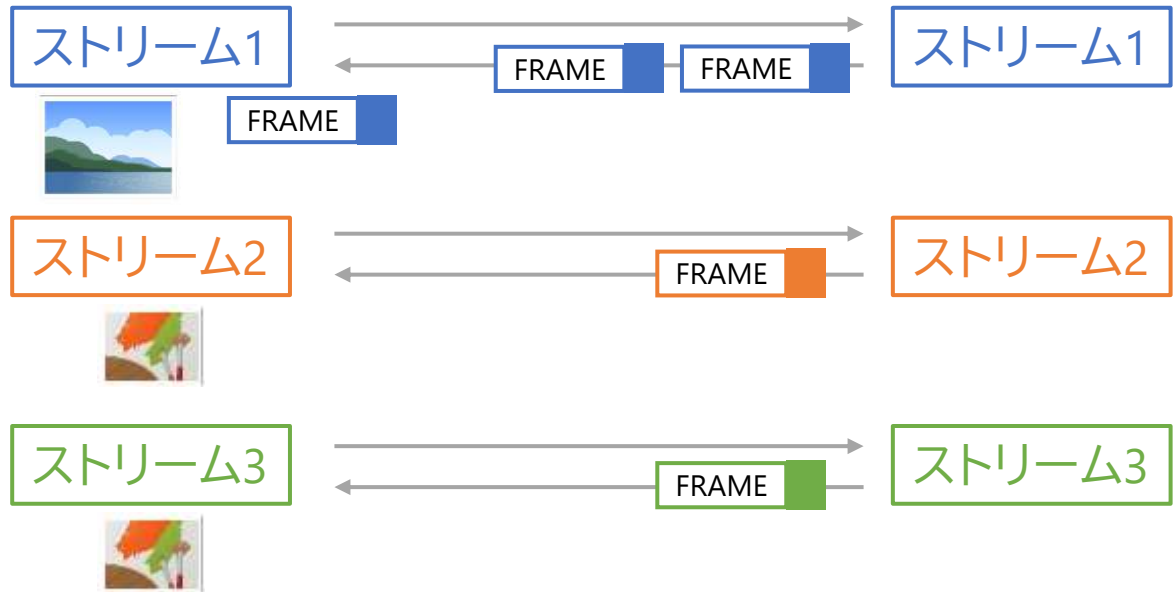
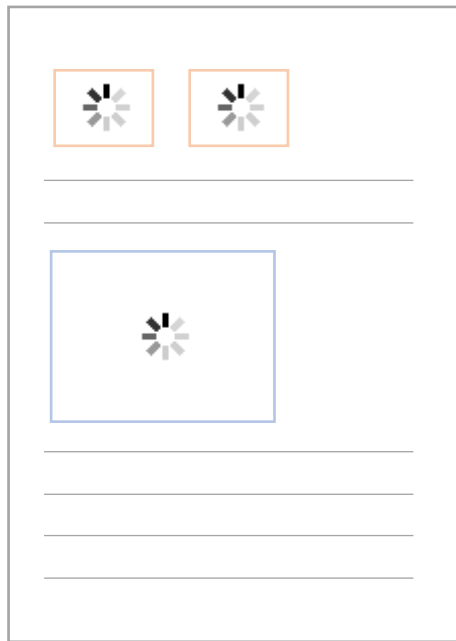
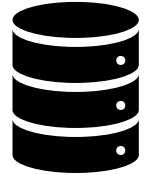
引用した図を削除しています

1本のTCP接続で帯域を最大限に使える

# HTTP/2 — HTTP HoLブロッキングを解消



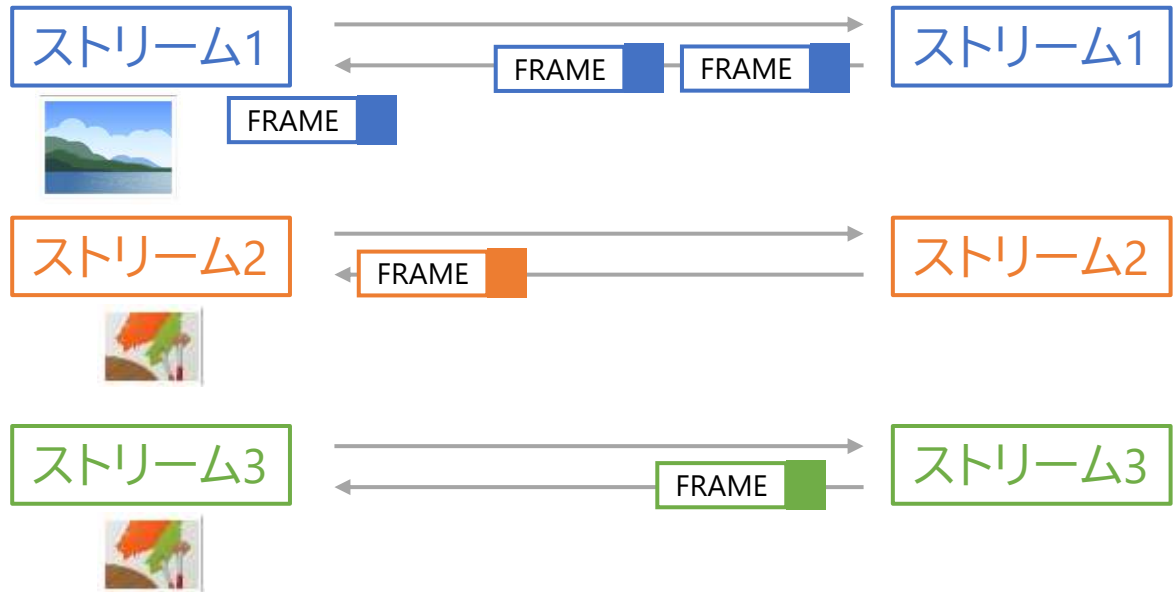
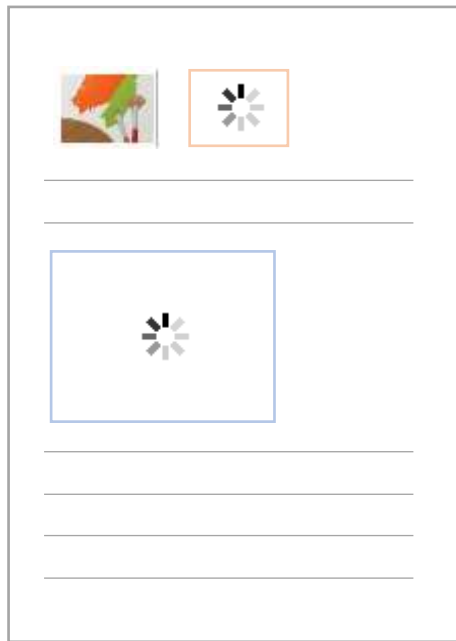
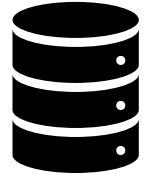
ストリームで並列化  
非同期



# HTTP/2 — HTTP HoLブロッキングを解消



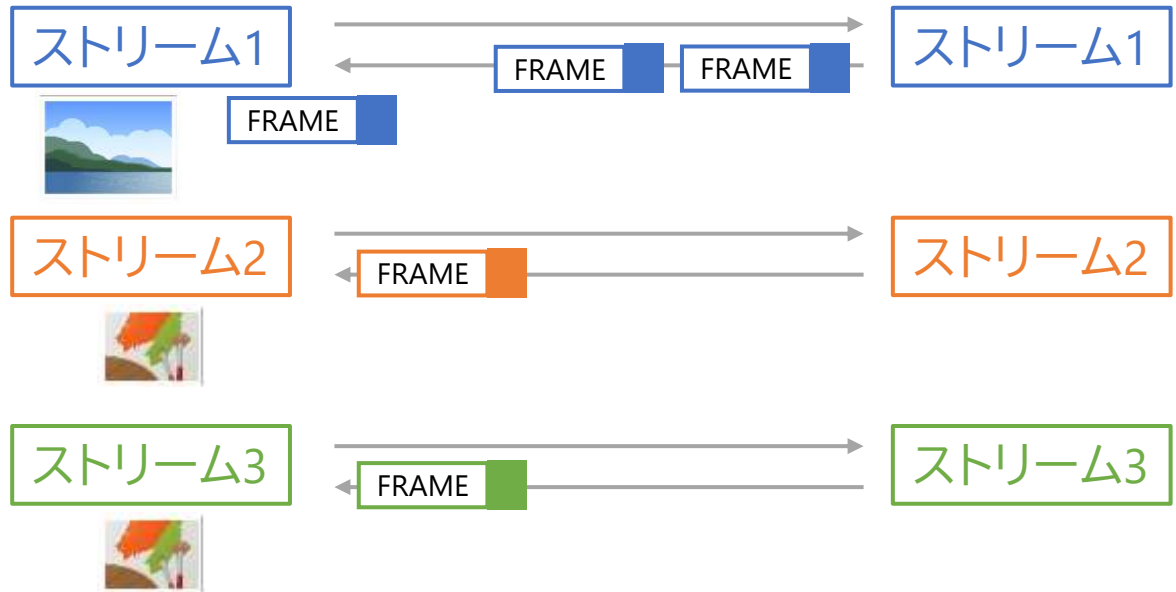
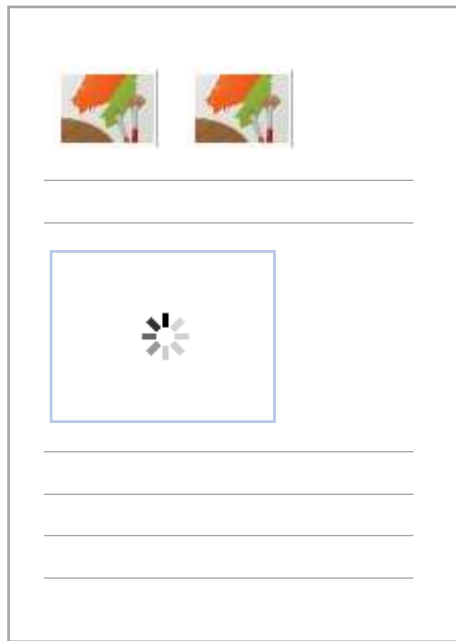
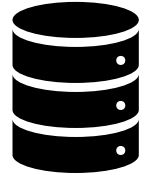
ストリームで並列化  
非同期



# HTTP/2 — HTTP HoLブロッキングを解消



ストリームで並列化  
非同期



リクエストは他のリクエストに  
ブロックされない

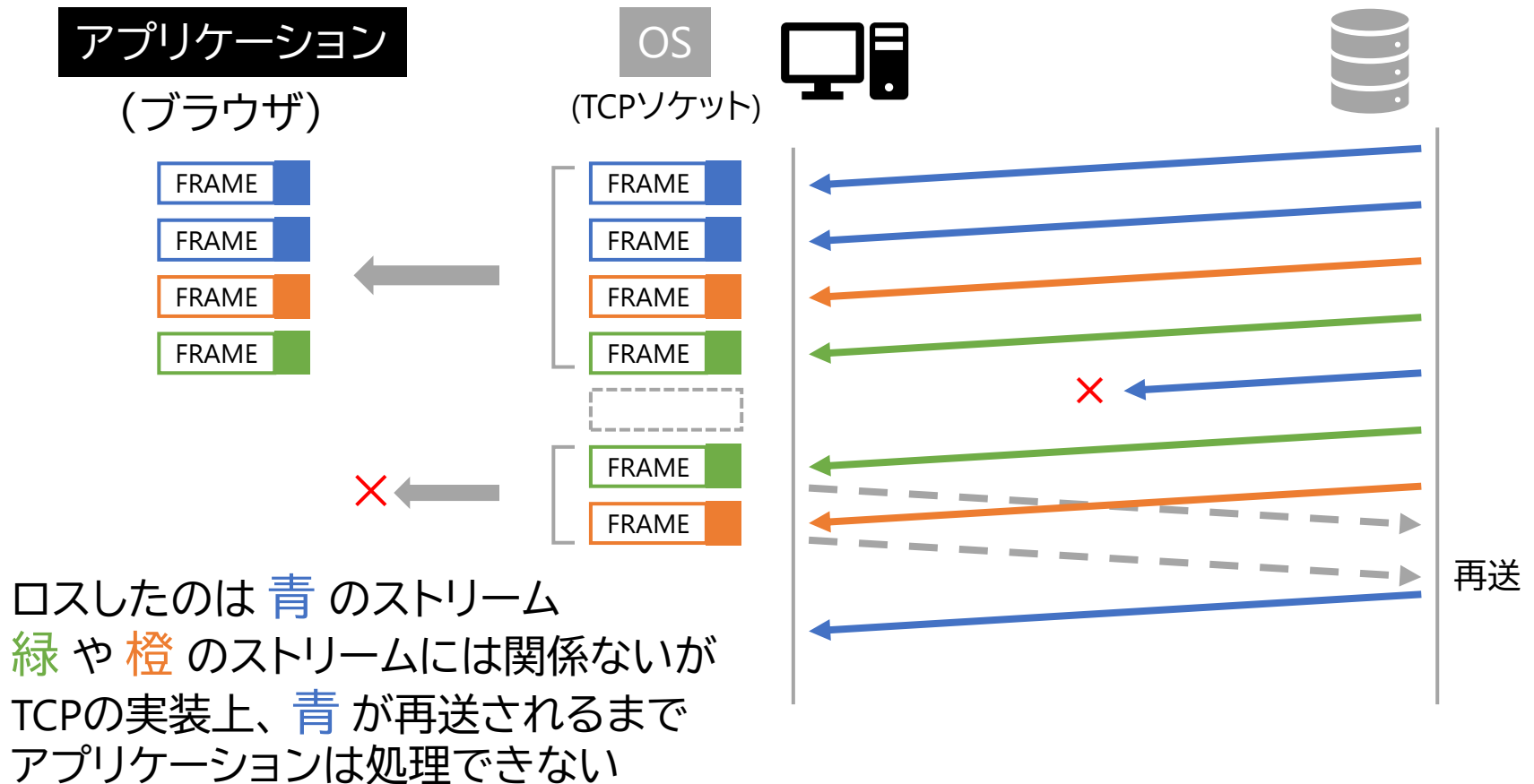


# HTTP/2 — HTTPヘッダーの圧縮

---

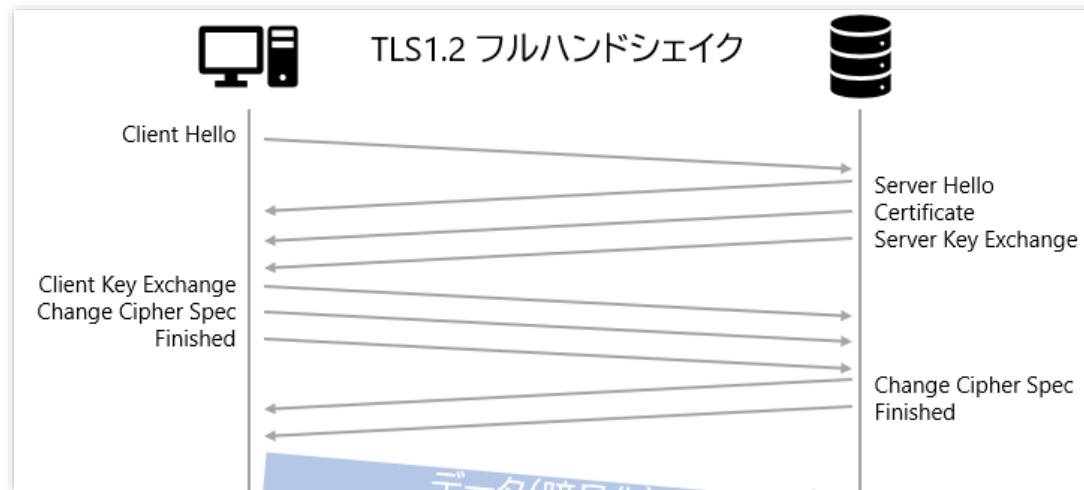
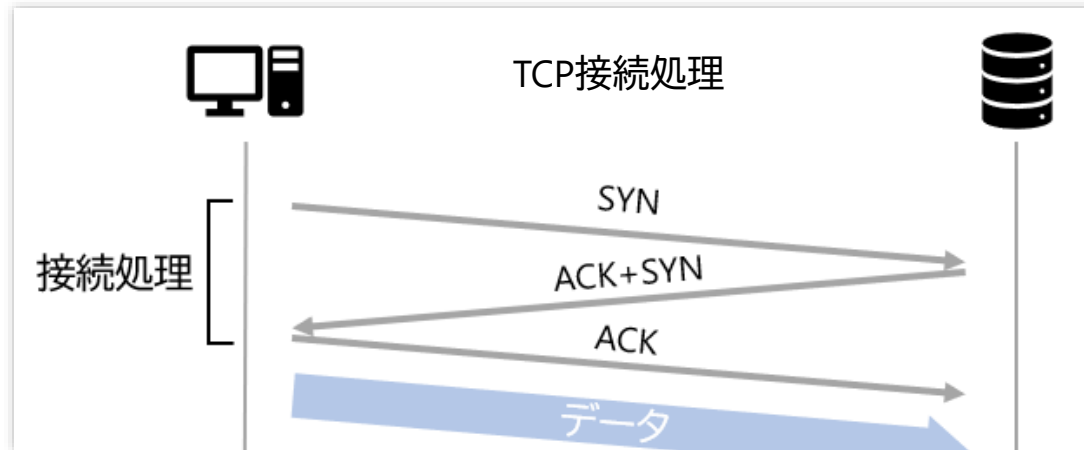
- HTTP/1.x ではヘッダーはプレーンテキスト
  - 圧縮できていなかった
- HTTP/2ではバイナリになった
- HPACKという圧縮形式を導入
- ハフマン符号化
  - 規格で定められた表
  - 前回までのリクエスト/レスポンスでの値を再利用

# HTTP/2の課題 — TCP HoLブロッキング



TCPのパケットロス/再送が  
すべてのストリームをブロックしてしまう

# HTTP/2の課題 — 接続までの時間



TCPとTLSのハンドシェイクで時間がかかる

# HTTP/2の課題

---

- TCP HoLブロッキング
- TCPとTLSの接続時間の短縮

HTTPのレイヤーではどうにもならない！

Google「TCPやめて新しいプロトコル作るわ」

# QUIC

---

UDPベースの多重化されたセキュアな転送プロトコル

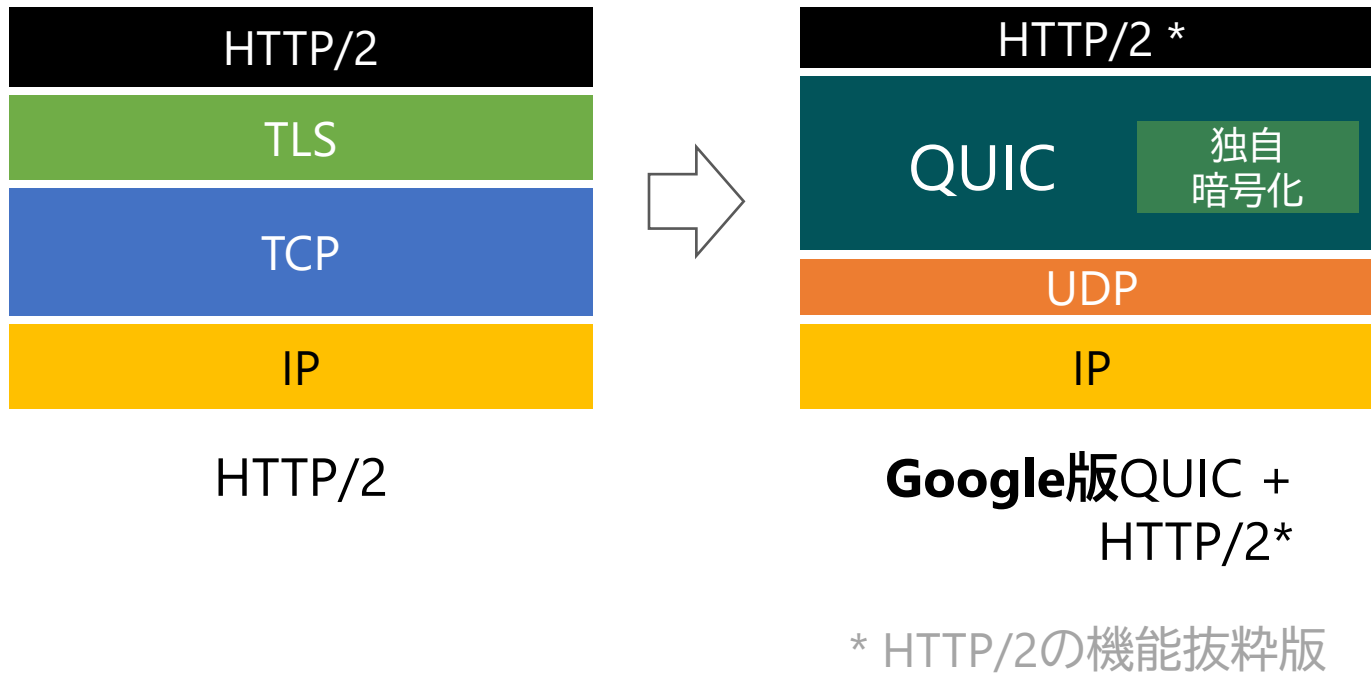
The image shows the Google homepage on the left and the Chrome DevTools Network tab on the right. The Network tab displays a list of requests, with a red box highlighting the 'Protocol' column, which shows 'http/2+quic/46' for all requests. Below the Network tab, a table summarizes the first five requests.

Name	Status	Protocol	Type
www.goo...	200	http/2+quic/46	document
googlelo...	200	http/2+quic/46	img
i2_2ec824...	200	http/2+quic/46	img
photo.jpg	200	http/2+quic/46	img
googlemi...	200	http/2+quic/46	img

Below the table, the text 'Google 検索は次の言語でもご利用いただけます: English' is visible.

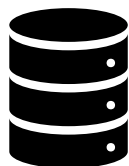
ChromeでGoogleのサービスにアクセス

# QUIC: UDPベースの多重化されたセキュアな転送プロトコル



- Googleが開発し自社サービスとChromeで採用
- (IETF版のQUICについては後ほど説明します)

# なぜUDPを使うか



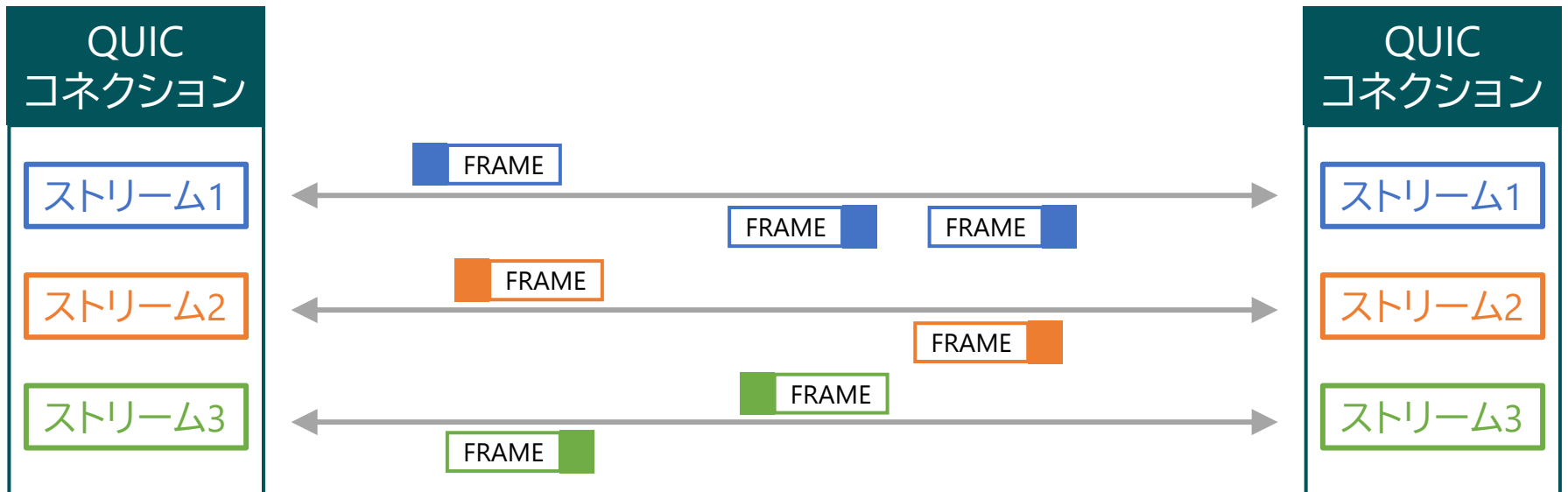
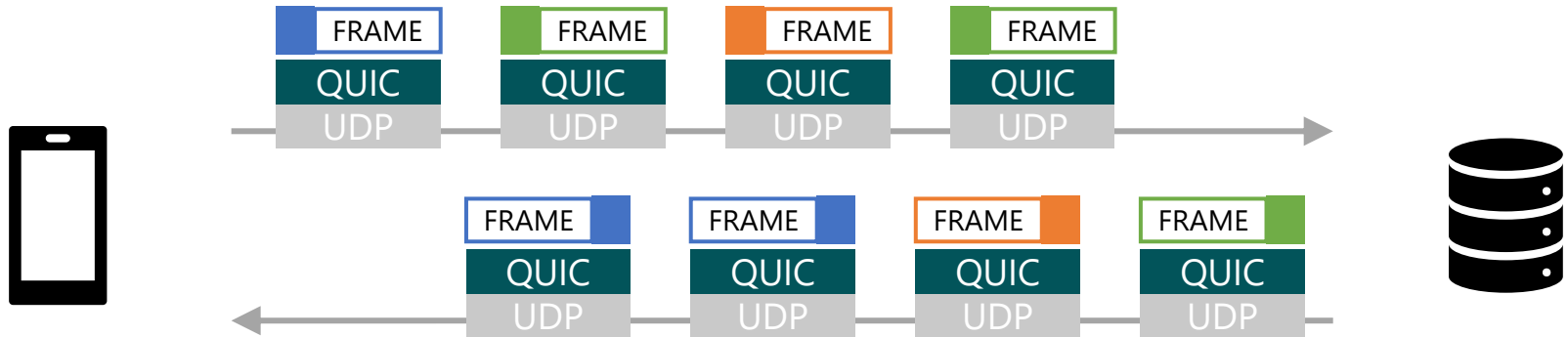
	TCP	UDP
通信の方向	双方向	相手に送るだけ
誤り/不達の検出	できる	できない
データの順序	保証される	保証されない
輻輳制御	ある	ない

機能がシンプルな代わりに、(届けば) **到達速度がはやい**  
UDPはTCPと同じように**普及している**

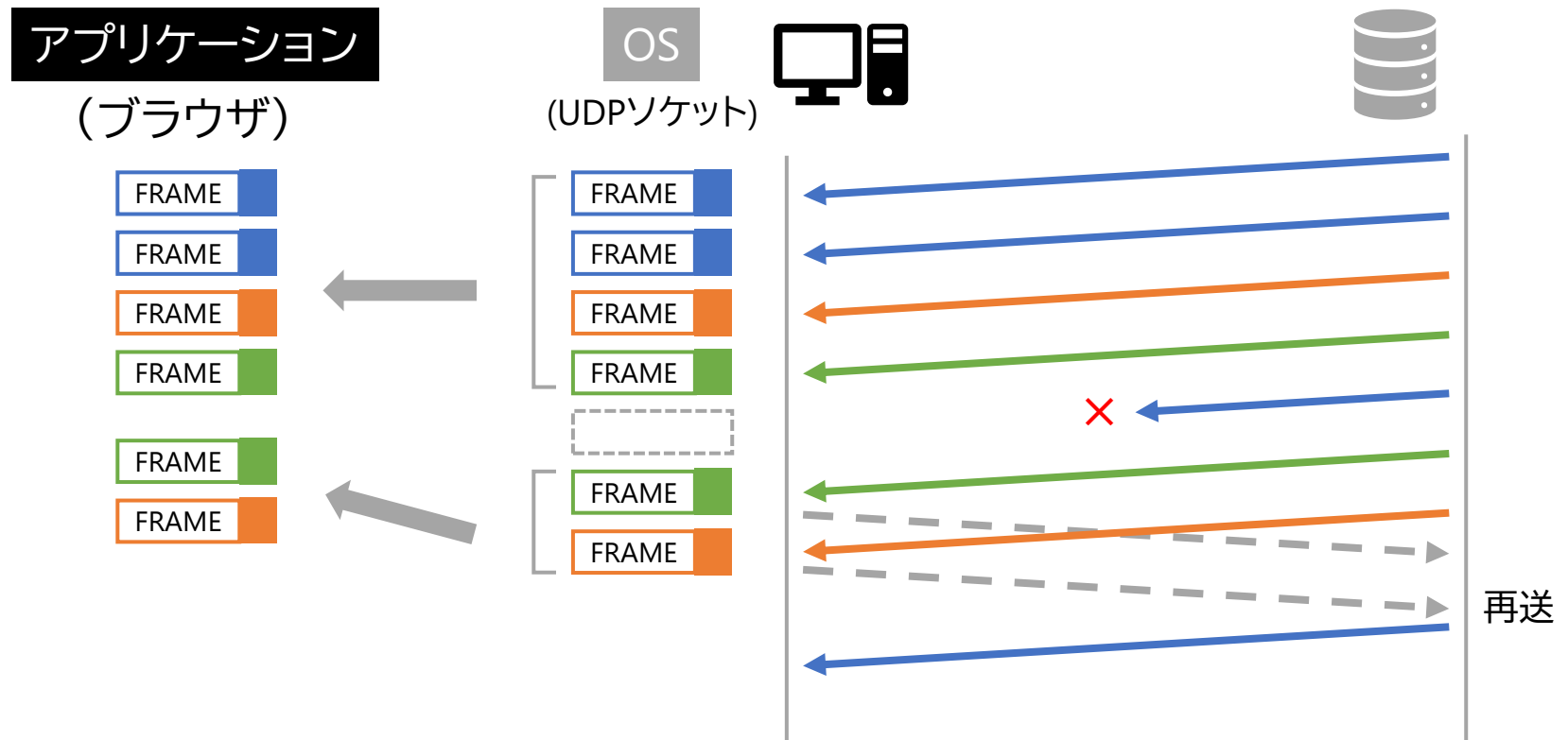
UDPを利用し、TCPよりも適したプロトコルを  
(ユーザー空間に)構築する



# QUIC: UDPベースの多重化されたセキュアな転送プロトコル



# QUIC — HoLブロッキングがない



パケットロスが起きても  
他のストリームの処理をブロックしない

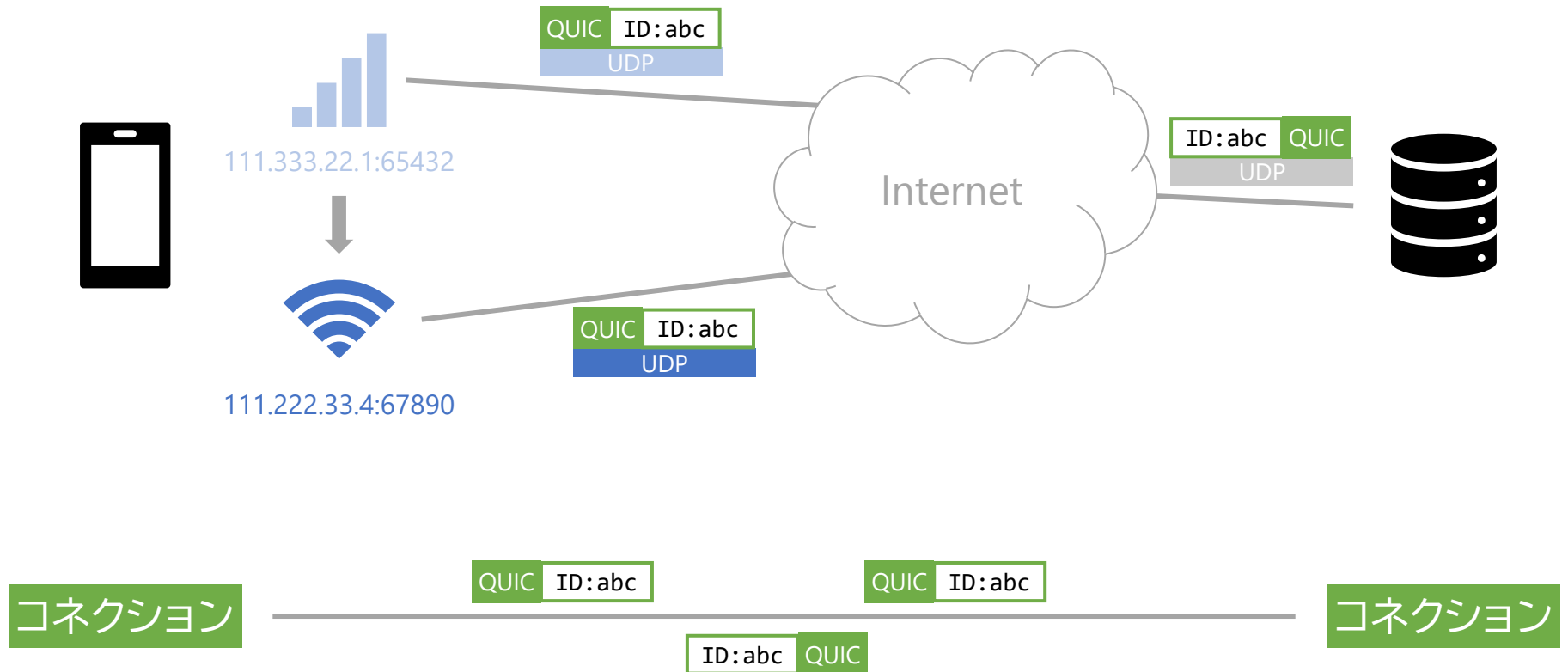
# QUIC — 0-RTT



TCP+TLSの場合は、それぞれにハンドシェイクが必要  
QUICはトランスポートとセキュリティを統合したプロトコルであり、  
再接続の場合、真に0-RTTで暗号化されたデータを送信可能

**最短 0-RTT でデータを送れる**

# QUIC — Connection Migration



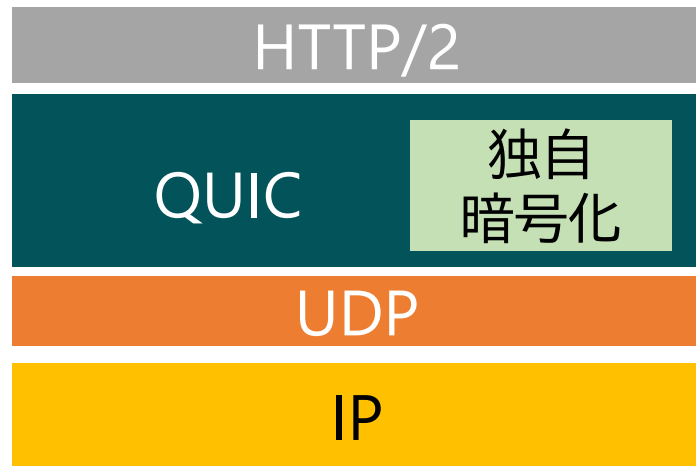
## IPやポートが変わっても処理を継続できる

# HTTP/3

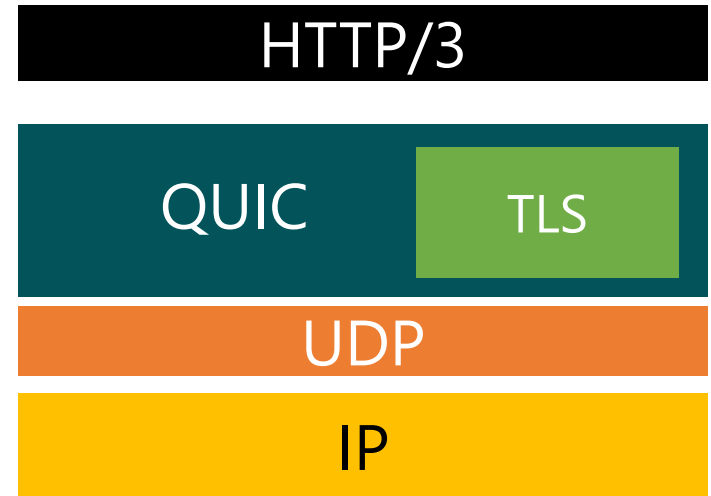
---

QUICを利用した新しいHTTP

# 2つのQUIC



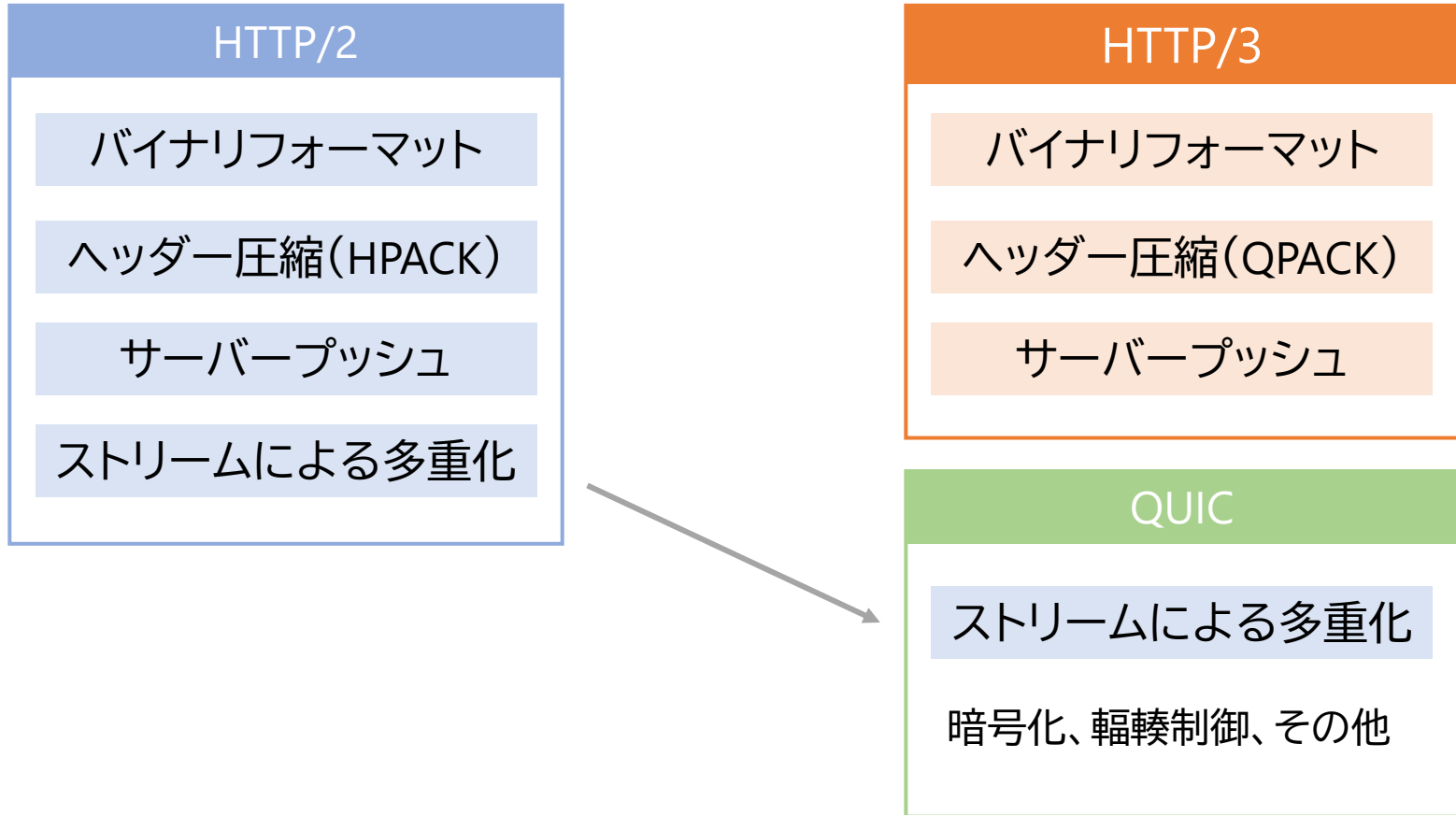
Google版QUIC +  
HTTP/2



IETF版QUIC +  
HTTP/3

- Google版QUICの独自暗号化をTLSにし、HTTP部分をHTTP/3としたものがIETFで策定中

# HTTP/3 — QUICを利用した新しいHTTP



HTTP/2からQUICと重複する機能を除いたもの

QUICとともにIETFで議論が進んでいる

まとめ

---



# HTTPの歴史(再掲)

---

- (1991年 HTTP/0.9)
- 1996年 HTTP/1.0
- 1997年 HTTP/1.1
- 2010年 ChromeでSPDY(HTTP/2の前身)が有効に
- 2015年 HTTP/2、ChromeでQUICが有効に
- 2020年 (HTTP/3 draft-27)

20年以上Webを支えているHTTP/1.1(もちろん現役)

HTTPのセマンティックは1.x系から変わっていない

# HTTPの進化は「水面下」で

- テキストベースからバイナリベースへ
- 多重化・並列化
  - HTTP/TCP HoLとの戦い
  - TCPからの脱却、新しいトランスポートプロトコルへ
- セキュリティ
  - TLSの利用、QUICでのTLSとの統合
- レイテンシーの削減
  - Keep Alive、サーバープッシュ、0-RTT
- マルチパスへの対応(モバイルネットワークなどに対応)
  - Connection Migration

Webを支えるHTTPのこれからの注目！

# 参考資料

最後までご視聴ありがとうございました。  
質問: #CAMPHOR\_Day, YouTubeコメント

- 渋谷よしき, Real World HTTP ミニ版, オライリー・ジャパン, 2019
- Hypertext Transfer Protocol Version 3 (HTTP/3), IETF, 2020  
<https://tools.ietf.org/html/draft-ietf-quic-http-27>
- QUIC: A UDP-Based Multiplexed and Secure Transport, IETF, 2020  
<https://tools.ietf.org/html/draft-ietf-quic-transport-27>
- Using TLS to Secure QUIC, IETF, 2020  
<https://tools.ietf.org/html/draft-ietf-quic-tls-27>
- Hypertext Transfer Protocol Version 2 (HTTP/2), IETF, 2015  
<https://tools.ietf.org/html/rfc7540>
- Ilya Grigorik, High Performance Browser Networking, O'Reilly Media, 2013  
<https://hpbnp.co/>
- 大津繁樹, HTTP/2からQUICへ続くWebプロトコルの進化, IJ, 2015  
[https://www.ij.ad.jp/dev/tech/techweek/pdf/151111\\_4.pdf](https://www.ij.ad.jp/dev/tech/techweek/pdf/151111_4.pdf)
- Martin Thomson, QUIC Security, 2020  
<https://docs.google.com/presentation/d/1OASDYIJlgSFg6hRkUjqdKfYTK1ZUk5VMGP3lv2zQCI8>