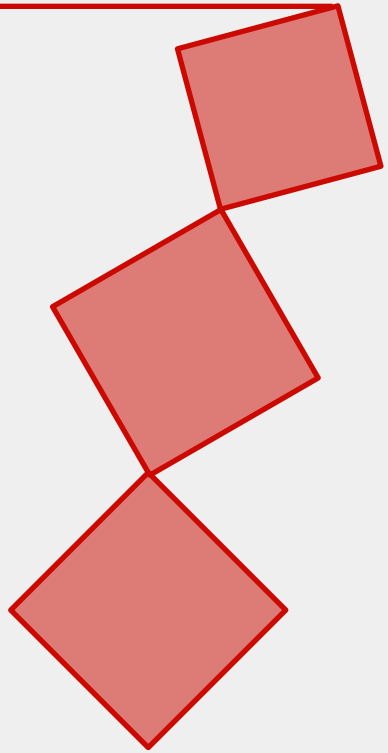
WebSocketを見てみよう

@Kuruma

本スライドは下記URIで公開しています:

http://bit.ly/ws-kuruma

http://kuruman.org/diary/2011/05/28/file/websocket_study_kuruma.pdf



whoami

- **Kuruma**
  - しがないOpera Browser使い
  - twitter.com/Kuruma
  - kuruman.org

もくじ

- 概要
- 実装状況
- はじめの一歩
  - 大まかな決まり事と流れ

概要

WebSocketとは

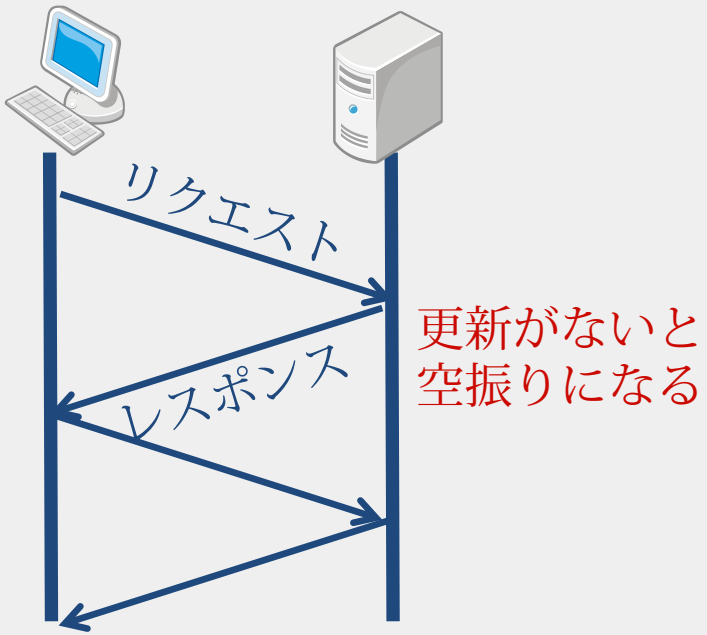
この技術の目的は、サーバとの双方向通信を必要とするブラウザベースのアプリケーションのために、**複数のHTTP接続を開くことなく双方向通信を実現する**ための仕組みを提供すること。

draft-ietf-hybi-thewebsocketprotocol-07より
Abstract最後の一文を意訳

HTTPを用いた強引な双方向通信

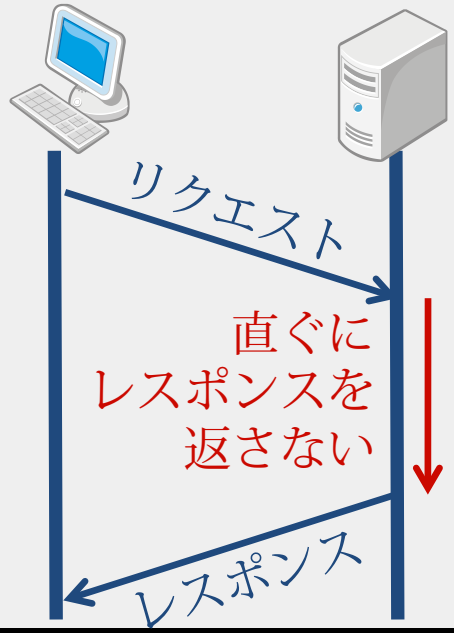
Ajax (Polling)

- 問い合わせによるPull
  - 非同期通信による情報取得
  - 無駄な通信が多い



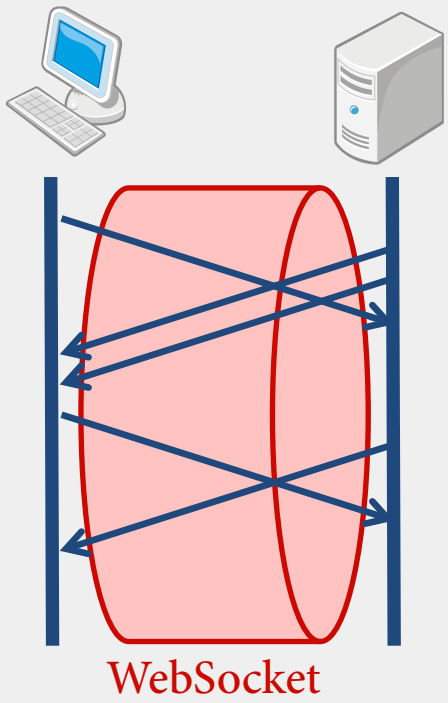
Comet (Long Polling)

- サーバからの疑似Push
  - 予めリクエストを送信
  - 依然無駄な通信が残る



WebSocketによる双方向通信

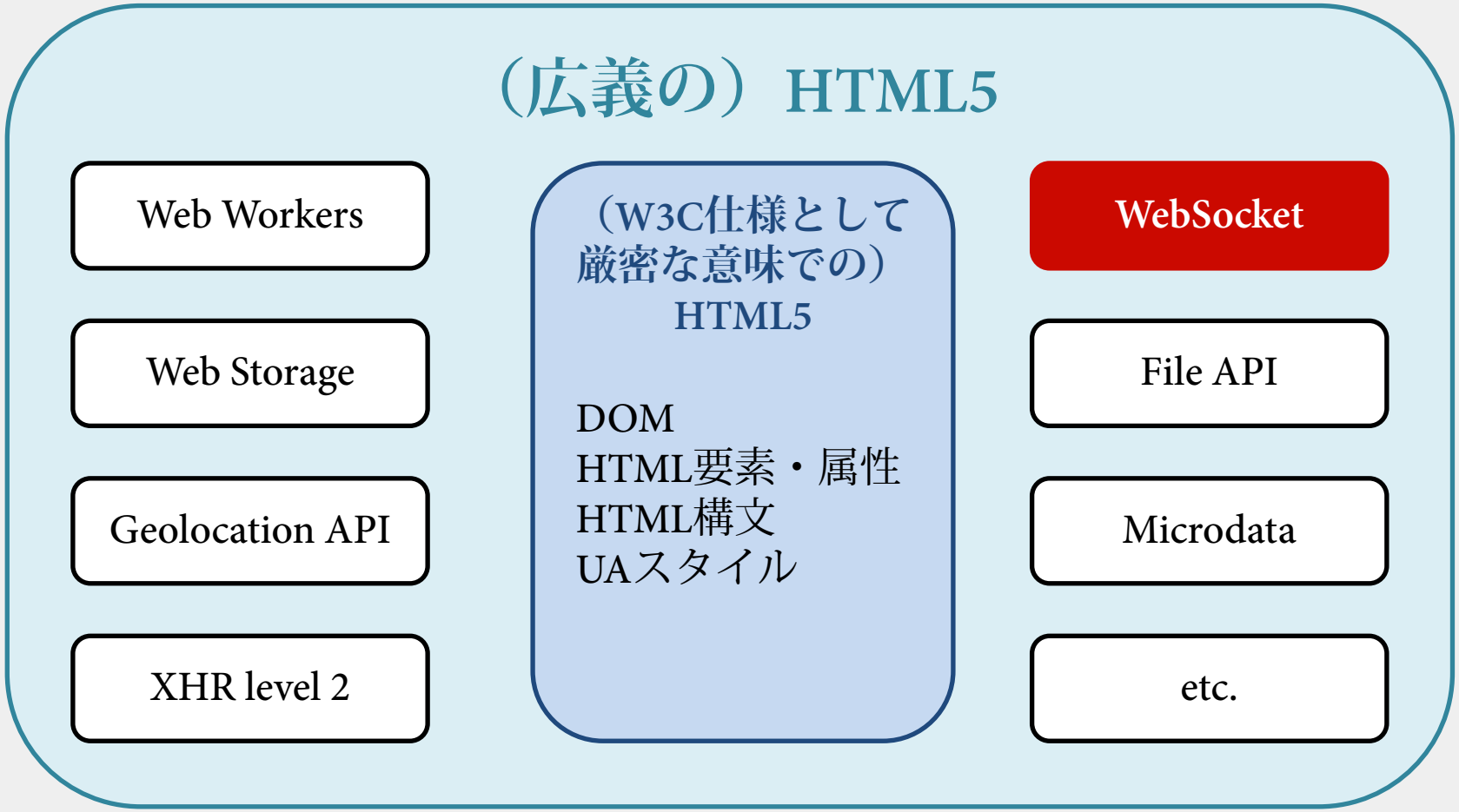
- 全二重の双方向通信が可能
  - UTF-8の文字列、バイナリデータ
- 生のTCPソケットではない
  - TCPベースの独立したプロトコル
  - 初期は”TCP for the Web”とも



WebSocketの標準化

- 標準化は現在進行形
  - The WebSocket protocol
    - IETF HyBi WG
    - http://tools.ietf.org/wg/hybi/draft-ietf-hybi-thewebsocketprotocol/
  - The WebSocket API
    - W3C WebApps WG
    - http://www.w3.org/TR/websockets/

WebSocket ∉ HTML5



WebSocket ∈ HTML5

- **Semantics:** RDFa, microdata, microformats
- **Offline & Storage:** Local Storage, Indexed DB, File API
- **Device Access:** Geolocation API, audio/video input
- **Connectivity:** WebSocket, Server-Sent Event
- **Multimedia:** audio/video
- **3D, Graphics & Effects:** SVG, Canvas, WebGL, CSS
- **Performance & Integration:** WebWorkers, XHR2
- **CSS3:** WOFF

既存のWebとの親和性

- 既存のWeb-proxyを通過できるように工夫
  - 最初のハンドシェイクをHTTPで行う
  - 80番ポートを使う (TLS使用時は443番)
- もちろんUser Agentの対応は必須

実装状況

WebSocketの変遷概要

	認証	バイナリ	圧縮	拡張	切断時 Status Code	データ フレーム
hixie-75	なし	MIME	不可	不可	なし	0x0 文字列 0xff
hixie-76 (hybi-00)	認証 もどき					
hybi-01		そのまま				バイナリ
hybi-02						
hybi-03			Deflate			
hybi-04	あり			あり		+mask
hybi-05				+negotiate		
hybi-06					あり	
hybi-07						

細かなフレーム形式の変更等は多々有り

hixie-**, hybi-**はそれぞれ
draft-hixie-thewebsocketprotocol-**, draft-ietf-hybi-thewebsocketprotocol-**を示す

潤沢なサーバ環境

- 一例として
  - Python: pywebsocket
    - hixie-75, hybi-00, hybi-07に対応
    - http://code.google.com/p/pywebsocket/
  - Java: Jetty
    - hybi-06, hybi-07にも対応
    - http://jetty.codehaus.org/jetty/
  - JavaScript: Socket.IO-node
    - https://github.com/LearnBoost/Socket.IO-node
  - PHP: phpwebsocket, Erlang: shirasu, etc.

Webブラウザ環境はこれから

	Internet Explorer	Firefox	Chrome	Safari	Opera
hybi-00		(4 RC+)	6	5.0.1	(11.00)
hybi-06	Lab版	Dev版			
hybi-07		6 Alpha			

実装差（バージョンや実装有無）を吸収するライブラリ等も存在
e.g. https://github.com/LearnBoost/Socket.IO

括弧で記載のものは標準で無効に設定(hybi-00にはセキュリティ上の問題有)
via. http://en.wikipedia.org/wiki/WebSockets

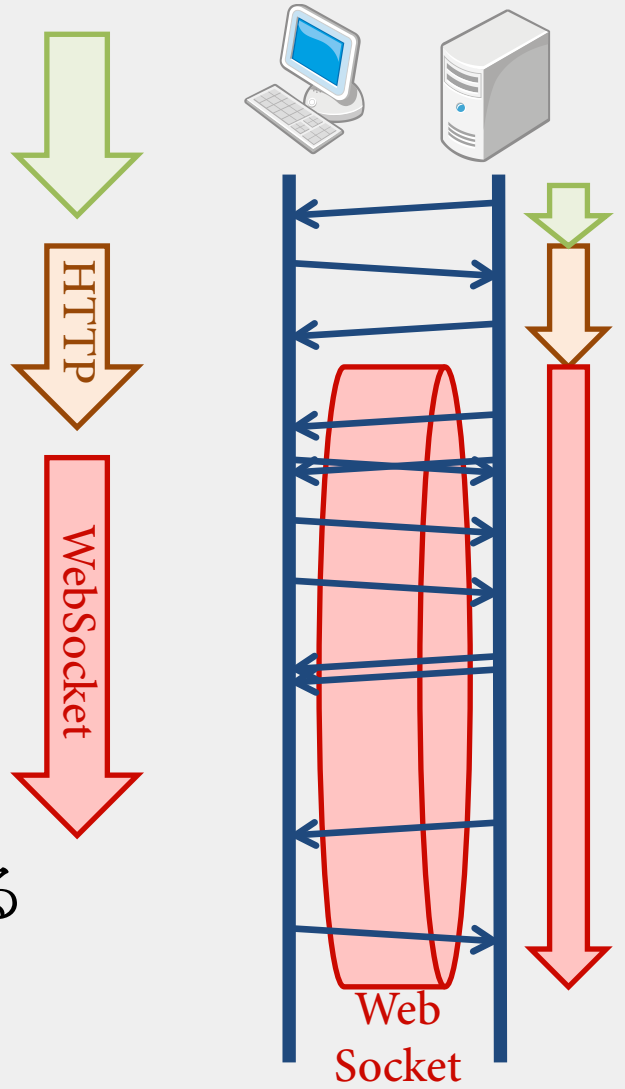
はじめの一歩

おことわり

- 今後大幅に変わる可能性もあります
- 2011-05-25時点でIETF/W3Cの公開しているドラフト文書に基づいた説明です

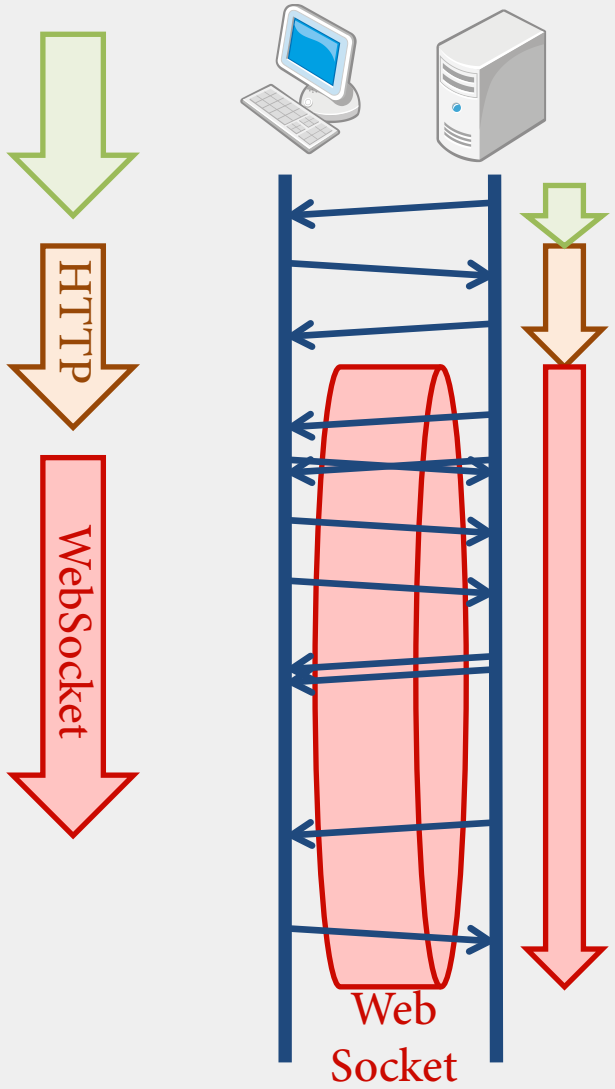
ブラウザから見た流れ

1. サーバAからScriptを取得
2. Scriptを実行
3. サーバAへWebSocket接続を要求
4. サーバAから接続が承認される
5. **サーバAとWebSocket通信**
6. WebSocket接続切断
   - サーバAから切断要求を受ける
   - 接続要求を送信する
   - その他なんらかの理由で切断される



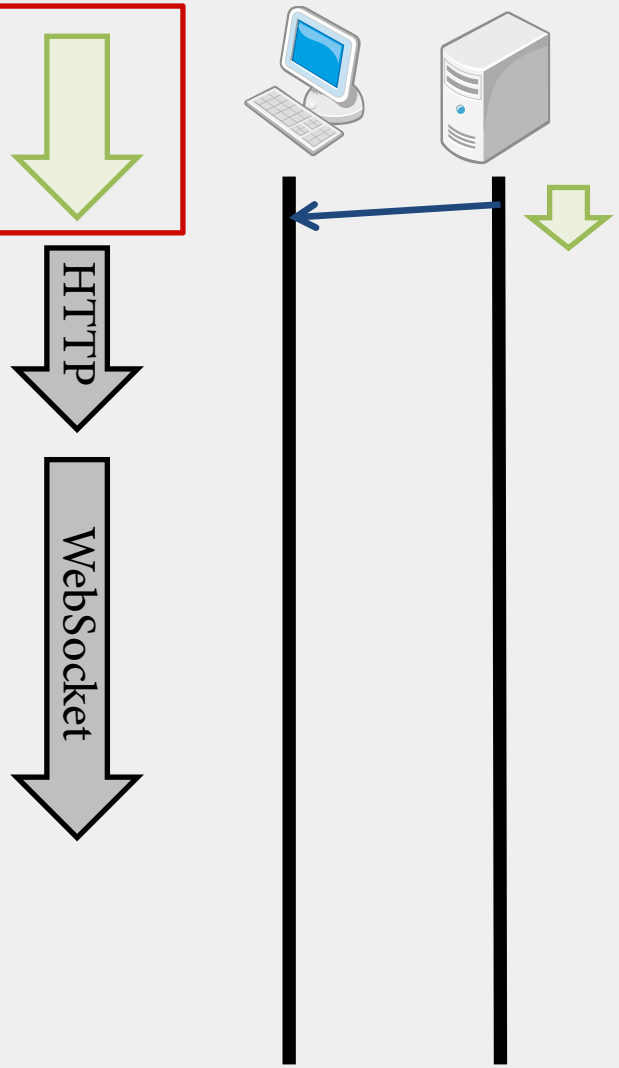
もっと大まかな流れ

1. Scriptを実行
2. HTTPでハンドシェイク
3. WebSocketでデータ転送



1. Scriptを実行

1. Scriptを実行
2. HTTPでハンドシェイク
3. WebSocketでデータ転送



1. Scriptを実行 (WebSocket API)

- WebSocket(url [, protocol])
- 4つのイベントハンドラ
  - onopen:　　接続確立時
  - onmessage:　メッセージ受信時
  - onerror:　　エラー発生時
  - onclose:　　接続終了時
- 2つのメソッド
  - boolean send(data)
  - void close()

cf. WebSocket URIs (protocol)

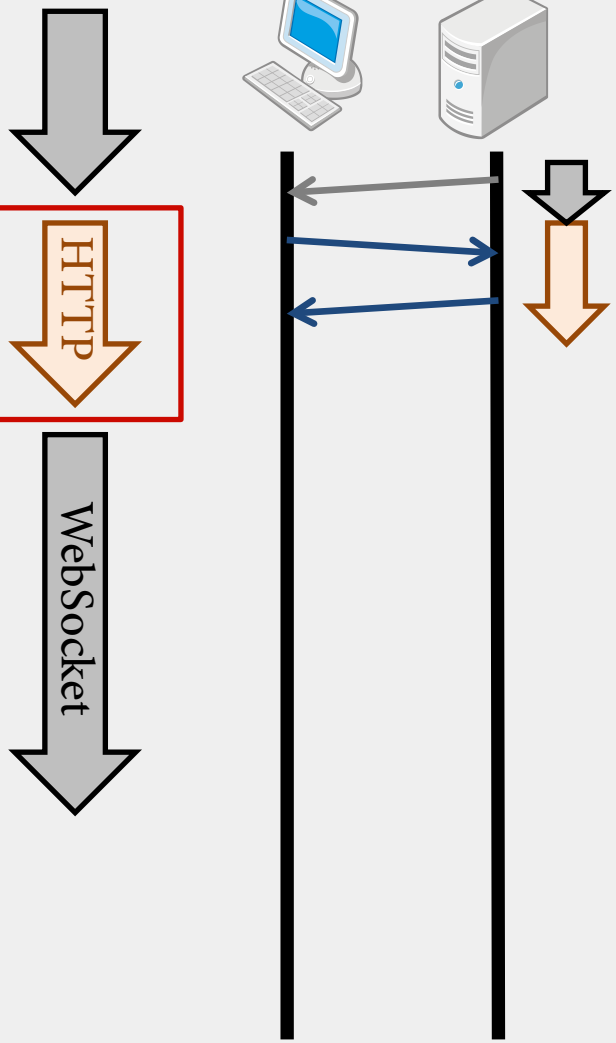
- <scheme>は下のいずれか:
  - ws:　　非セキュア通信
  - wss:　　セキュア通信
- <port>を含むことができる
  - 省略された場合、80(ws)または443(wss)
- “<path>[?<query>]”がWebSocketの端点のID
  - <path>が空の時は、<path>をU+002F(/)とする
- <fragment>を含んではならない

cf. コンストラクタの例

- 下の2つは等価
  - new WebSocket(“wss://example.com:5984”)
  - new WebSocket(“wss://example.com:5984/”)
- リソース名でソケットを使い分けられる
  - new WebSocket(“ws://example.com/ctrl”)
  - new WebSocket(“ws://example.com/chat?room=1”)
  - new WebSocket(“ws://example.com/chat?room=2”)
- フラグメントはダメ
  - new WebSocket(“ws://example.com/#ng”)

2. HTTPでハンドシェイク

1. Scriptを実行
2. **HTTPでハンドシェイク**
3. WebSocketでデータ転送



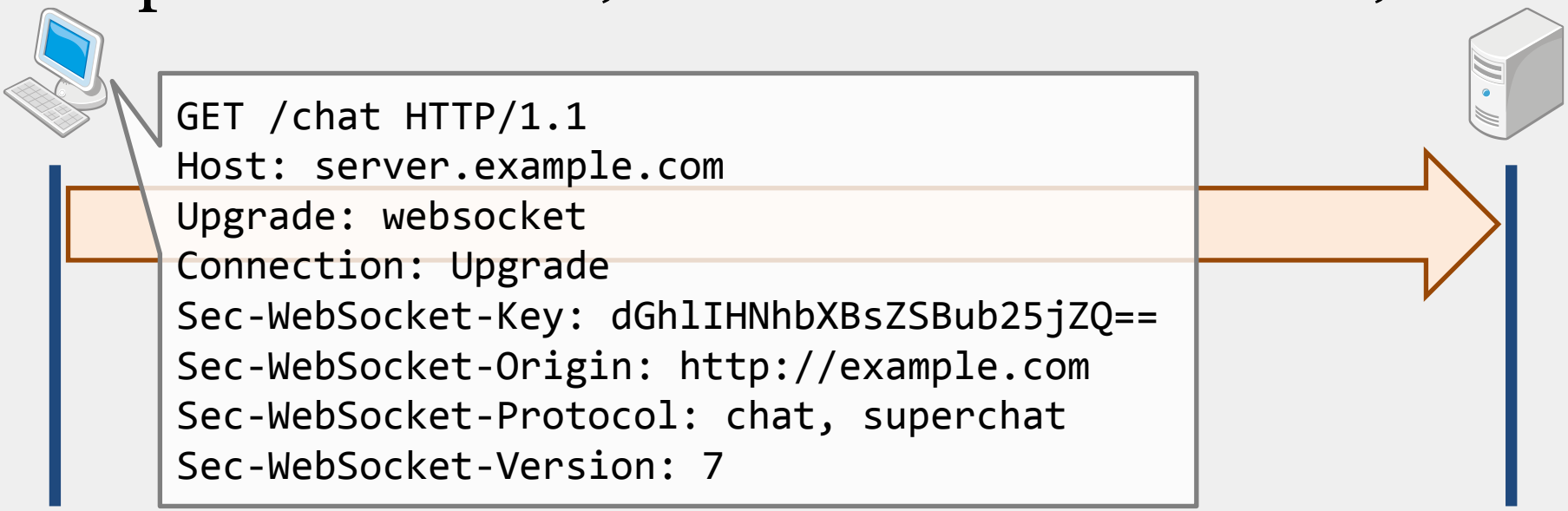
2. HTTPでハンドシェイク

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 7
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

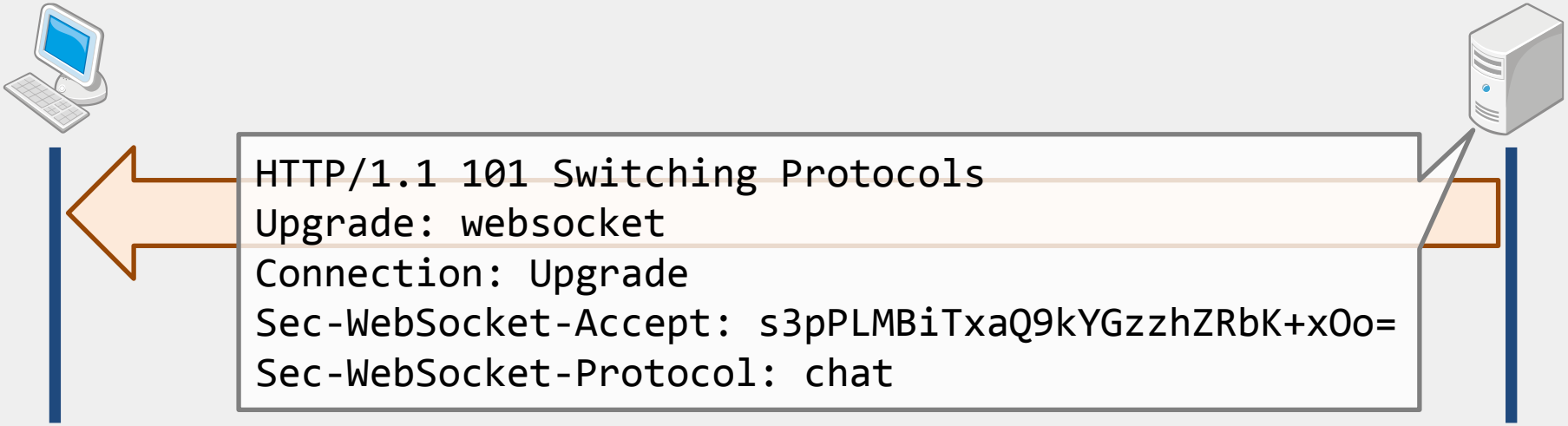
2-1.リクエスト

- HTTP/1.1以上でGET
- Upgrade: websocket
- Sec-WebSocket-Key: 任意の文字列
- Optional: Cookie, Sec-WebSocket-Protocol, etc.



2-2.レスポンス

- 101 Switching Protocols
- Sec-WebSocket-Accept: KEYに応じた文字列
  - base64(sha1(KEY+GUID*))
    *258EAFA5-E914-47DA-95CA-C5AB0DC85B11
- Optional: Sec-WebSocket-Protocol, etc.

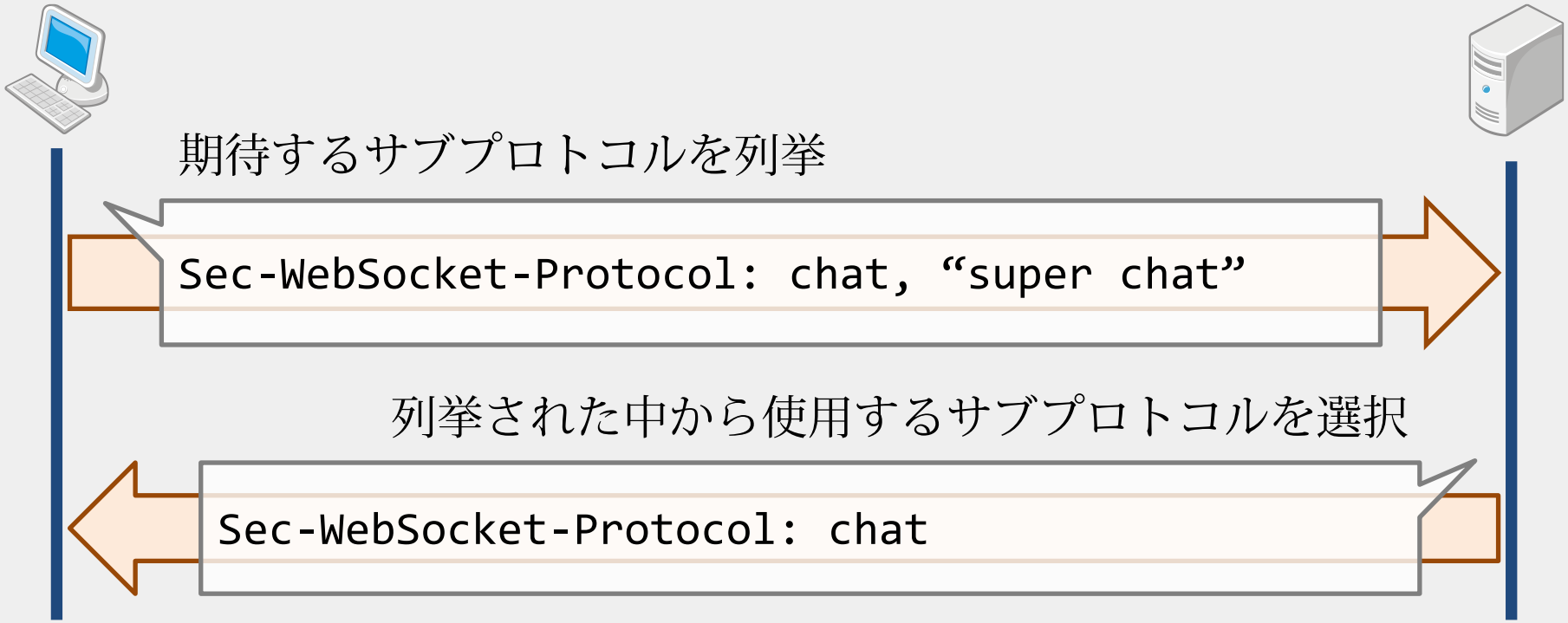


2-3. 接続確立

- Sec-WebSocket-Acceptの値で検証
- ブラウザから接続を確立

cf. サブプロトコル

- アプリケーションレベルのプロトコル
  - WebSocketの上でデータ転送を行う
- Sec-WebSocket-Protocolでネゴシエーション



cf. 拡張 (Extensions)

- Sec-WebSocket-Extensionsヘッダを用いる
  - プライベートな拡張はx-で始まる名称
- 定義済みの拡張は圧縮(deflate-stream)のみ

拡張と引数（拡張でのみ使われる）を適用希望順に列挙
※他のHTTPヘッダ同様に行が分割されうる

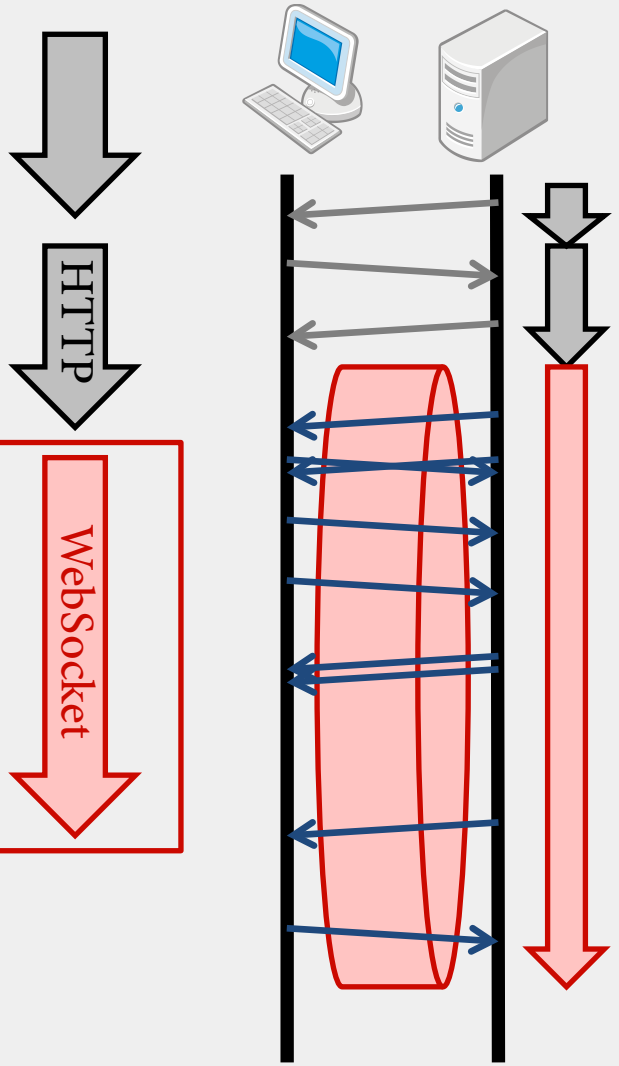
```
Sec-WebSocket-Extensions: hoge
Sec-WebSocket-Extensions: x-fuga; q=0.9
```

使用する拡張を適用順に列挙
※この場合はx-fuga(hoge(DATA))

```
Sec-WebSocket-Extensions: hoge, x-fuga
```

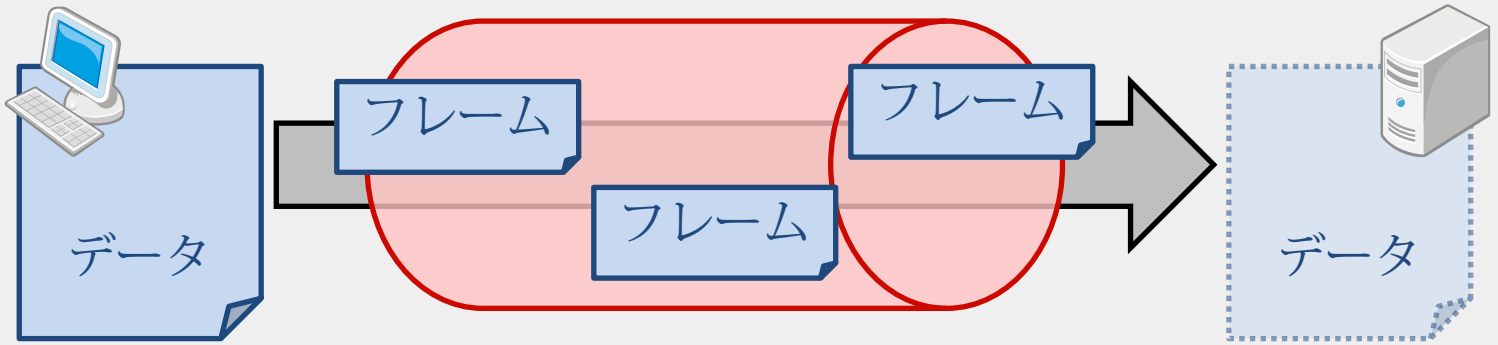
3. WebSocketでデータ転送

1. Scriptを実行
2. HTTPでハンドシェイク
3. **WebSocketでデータ転送**

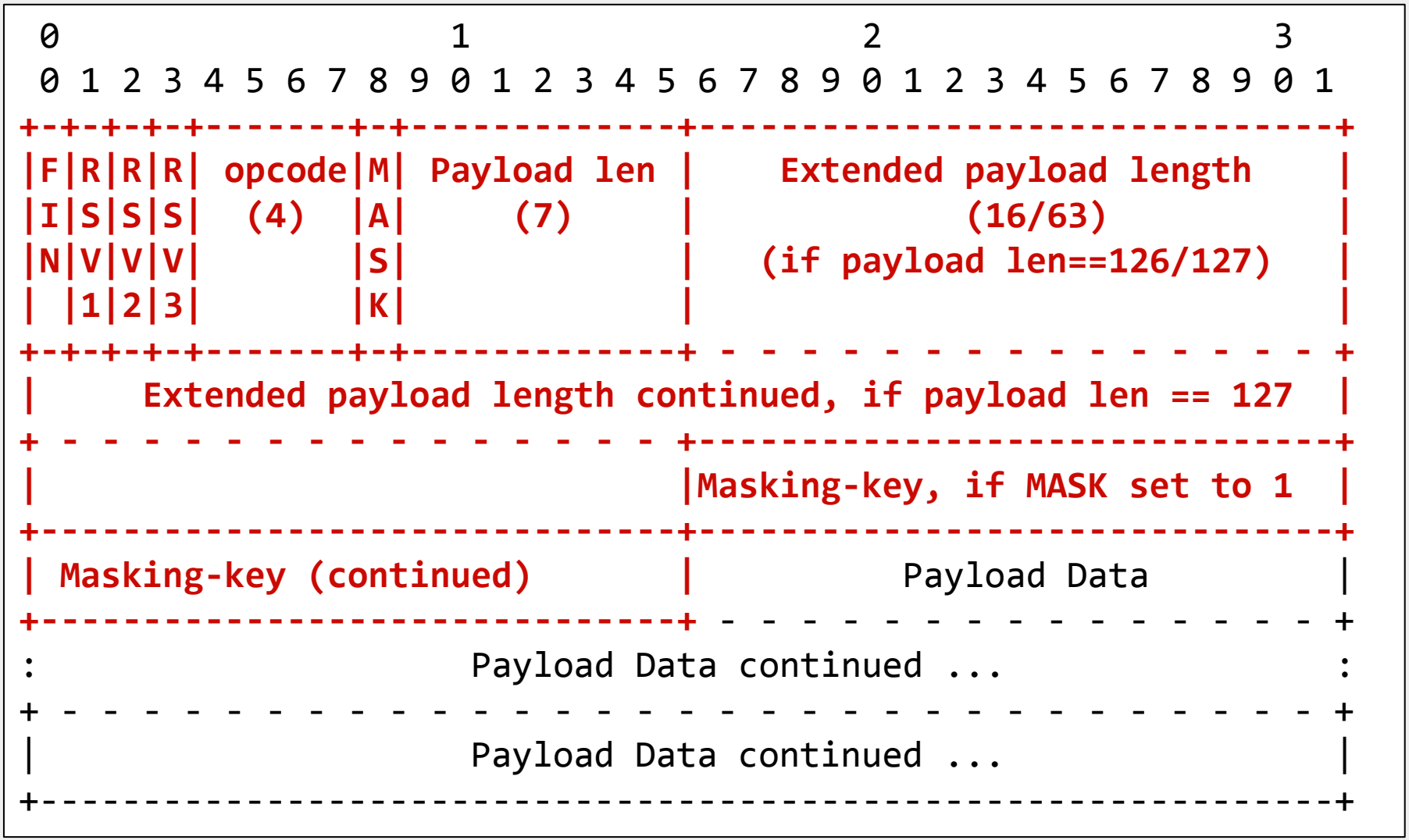


データとフレーム

- WebSocketはデータを転送する際、1つまたはそれ以上のフレームの連続として、データを転送する。
  - フレームの順序等は下のレイヤで保証(TCP)
  - フレームは可変長(最大2^{63}バイト)のバイナリ
  - ペイロードはマスクできる



フレームの構造

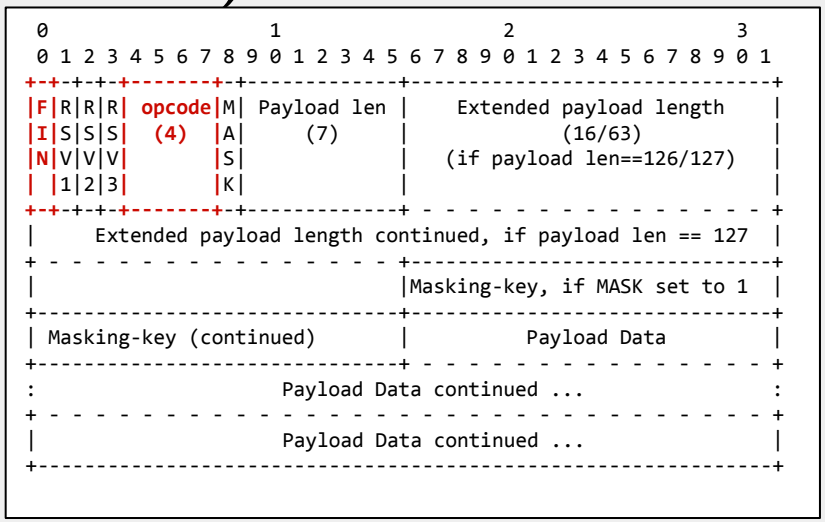

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
FIN RSV1 RSV2 RSV3 opcode (4) MASK Payload len (7)
Extended payload length (16/63)
(if payload len==126/127)
Extended payload length continued, if payload len == 127
Masking-key, if MASK set to 1
Masking-key (continued)
Payload Data
Payload Data continued ...
Payload Data continued ...

フレームの種類 (opcode)

- 非制御用フレーム (non-control frame)
  - ペイロードが文字列型 (text frame)
  - ペイロードがバイナリ型 (binary frame)
  - 続きのペイロード (continuation frame)
- 制御用フレーム (control frame)
  - 切断 (connection close)
  - ping / pong

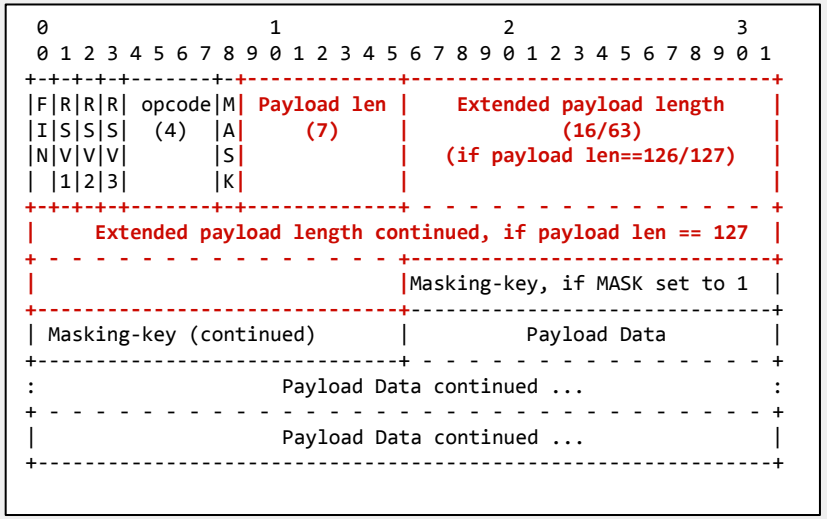
※分割送信禁止

※125バイト以下



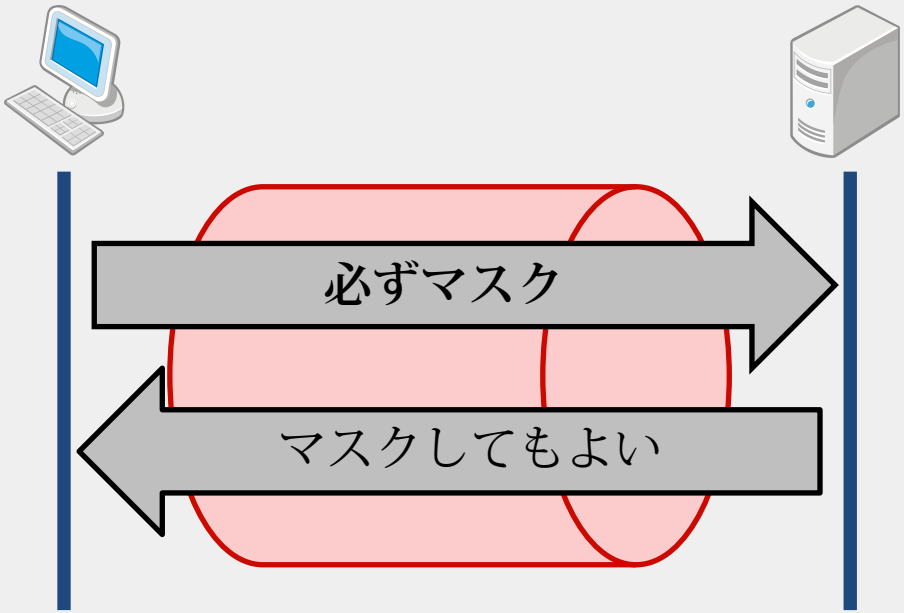
ペイロード長の定義

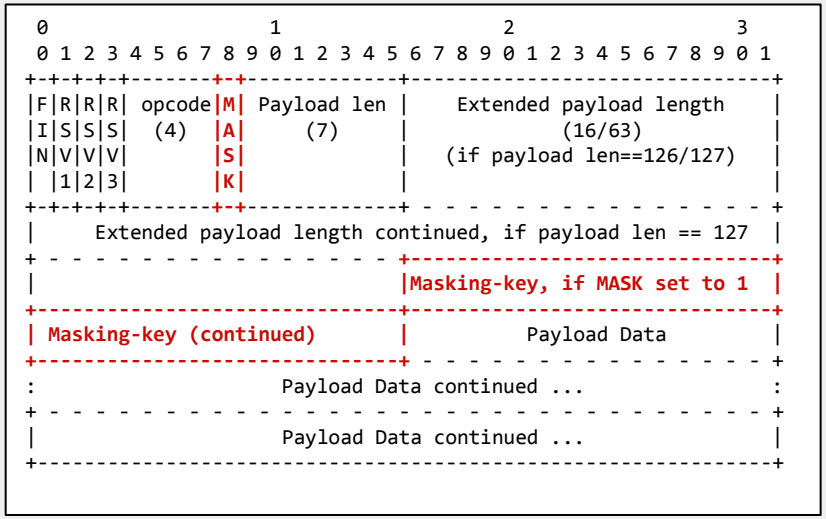
ペイロード長	Payload length (7)	Extended payload length (16)	Extended payload length continued (48)
～125バイト	unsigned int	-	-
～2^16バイト	126	unsigned int	-
～2^63バイト	127	unsigned int（但し最上位ビットは0）	



ペイロードのマスク

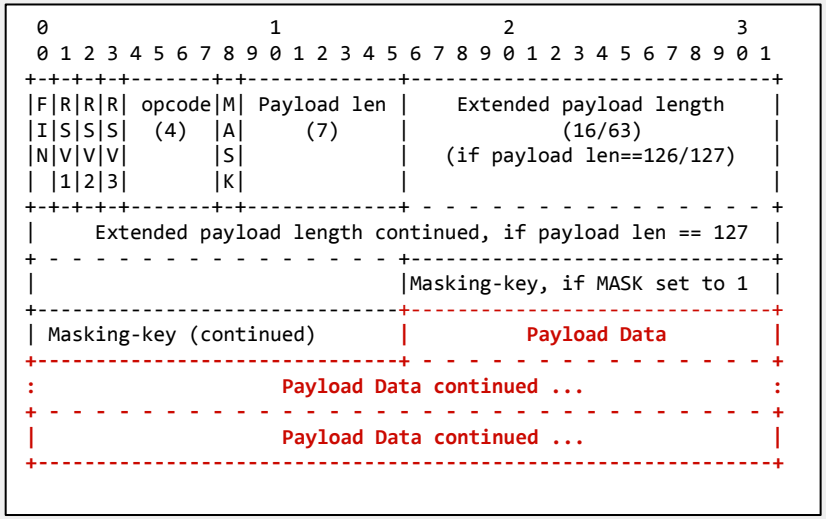
- フレーム中のMasking-keyを用いる
- ブラウザが送信する際は、必ずマスクする





あとはよしなに送信するのみ

- 非制御用フレームのペイロードは任意
  - 必要に応じてサブプロトコルで規定
- 拡張はペイロードの先頭に情報追加しうる



pingとpong

- WebSocketでも定義
- Pingのペイロードは任意
- Pongはpingと同一のペイロードを返す

WebSocketの切断

- closeの送信と受信をそれぞれ1回以上行う
  - 同時に切断が始まっても問題ない
  - WebSocket切断後は下のTCP接続も切断すべき
- ステータスコードで切断理由を示せる
  - closeのPayload Dataの先頭に数値で格納

切断時のステータスコード

- **0-999**: 未使用
- **1xxx**: プロトコルのための予約領域
  - 1000: 通常の切断（用済み等）
  - 1001: 端末の消失（電源断やページ遷移等）
  - 1002: プロトコルエラー
  - 1003: データ型に未対応
  - 1004: 容量超過
- **2xxx**: 拡張のための予約領域
- **3xxx**: ライブラリやフレームワークが使用
- **4xxx**: アプリケーションが使用

もっと深く知りたい方は

- 仕様書を読みましょう！

まとめ

- 効率的な双方向通信を実現
- 標準化に向けて今も議論中
- 通信内容も決して複雑ではない

ご清聴ありがとうございました