ORACLE

APIs, Stop Polling Let's Go Streaming

Phil Wilkins

OCI

Feb 2023

Speaker

http://mng.bz/jmzV
Code: **ctwdevweek23**

Phil Wilkins
Cloud Developer Evangelist

Philip.Wilkins@Oracle.com
http://bit.ly/odevrel_slack @Phil Wilkins - ORACLE

mp3monster.org / cloud-native.info / oracle-integration.cloud
linkedin.com/in/philwilkins
github.com/mp3monster
@mp3monster

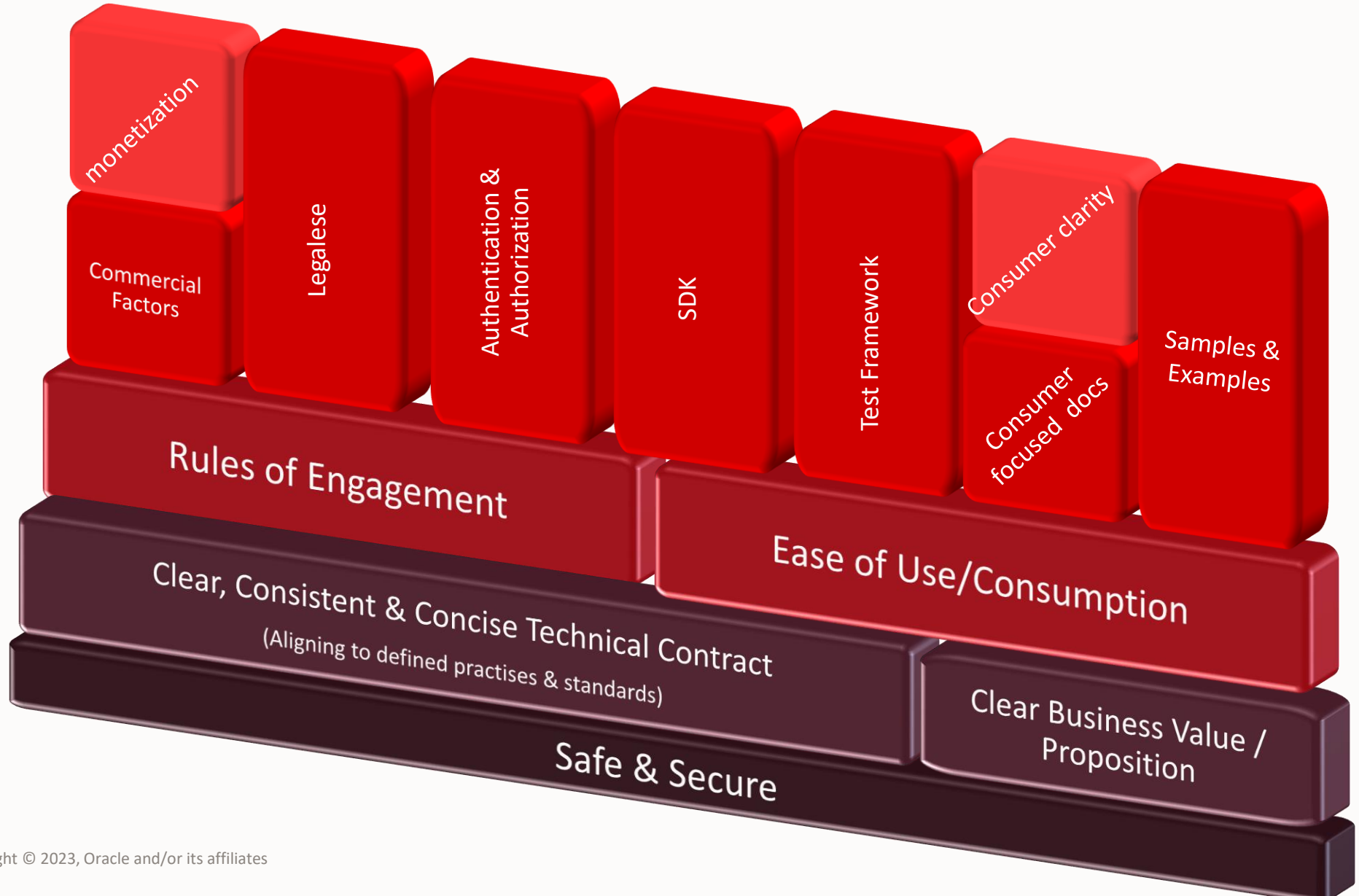
Downsides of API Polling

- Too frequent …
  - Unnecessary server effort repeating the same queries
    - Too much load and risk of service degradation
    - Each API call carries an overhead of initiating the exchange
  - Network bandwidth consumption transmitting duplicated data
  - If content refresh frequency can impact user experience – try to do something and the data has already changed
- Too infrequent …
  - Data received too late to be actionable
  - User experience – application content not refreshing quickly enough, and users start to force app refreshes – typically more costly!
  - Amount of data that may need to be cached is a function of the polling interval

Before we go streaming, we need to consider …

- Security…
  - Know who is getting what data
  - Is data going to the requestor
  - Satisfying consumer security needs (assurance of legitimate origin when pushing events)
- Is the client consuming data…
  - Recognizing consumer connection loss
  - Consumer coping with data volume (back pressure)
  - Handling out-of-sequence or missing events
  - Only receiving data they're allowed to get (events & attributes)
- API documentation…
  - Open API Specification – not optimal for Async / Streaming API specifications
  - Consumer enablement e.g. tech availability - libraries, SDKs, etc.
- Cross charging / Monetization of APIs…
  - How might the charging model work if we're pushing events?
  - Controlling data serving costs e.g. not sending events that aren't needed/wanted
- Ease of development & maintenance
  - How well is the technology understood
  - Is the solution maintainable?

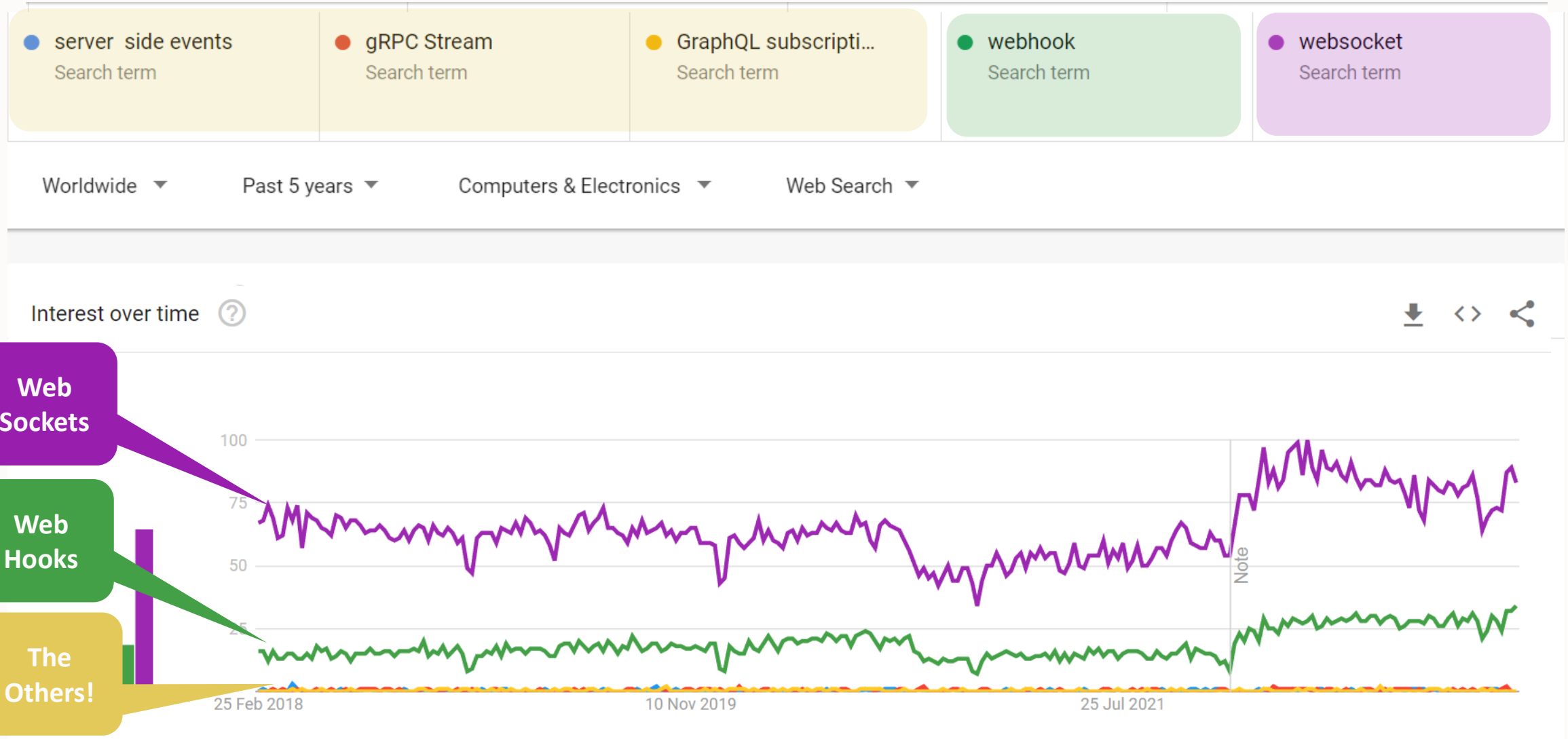
The Make-Up of a Good API



Common ‘Streaming’ API options …

- Web Hooks (Inverted APIs)
- Web Sockets
- Server Side Events
- GraphQL Subscriptions
- gRPC Streams

Trends on different Stream Techniques



Trends on different Stream Techniques

server side events
Search term

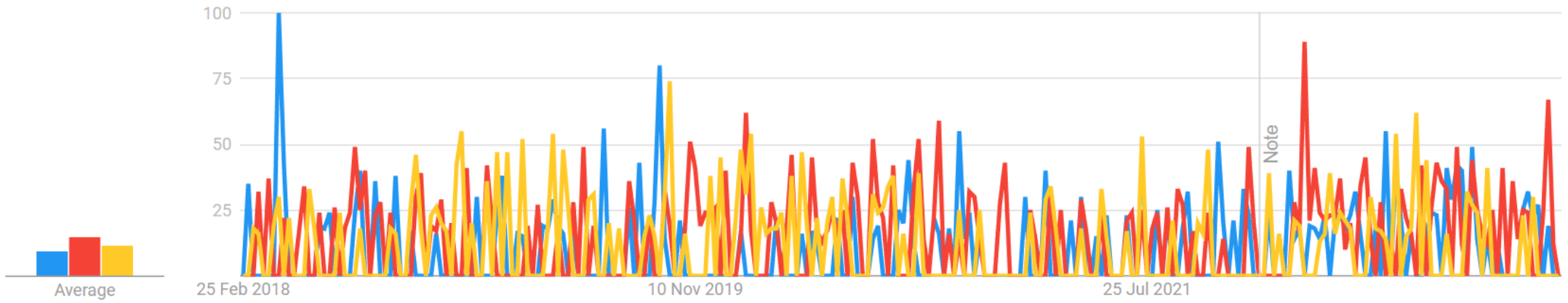
gRPC Stream
Search term

GraphQL subscript...
Search term

+ Add comparison

Worldwide | Past 5 years | Computers & Electronics | Web Search

Interest over time



Trends on different Parent (Stream) Techniques

server side events
Search term

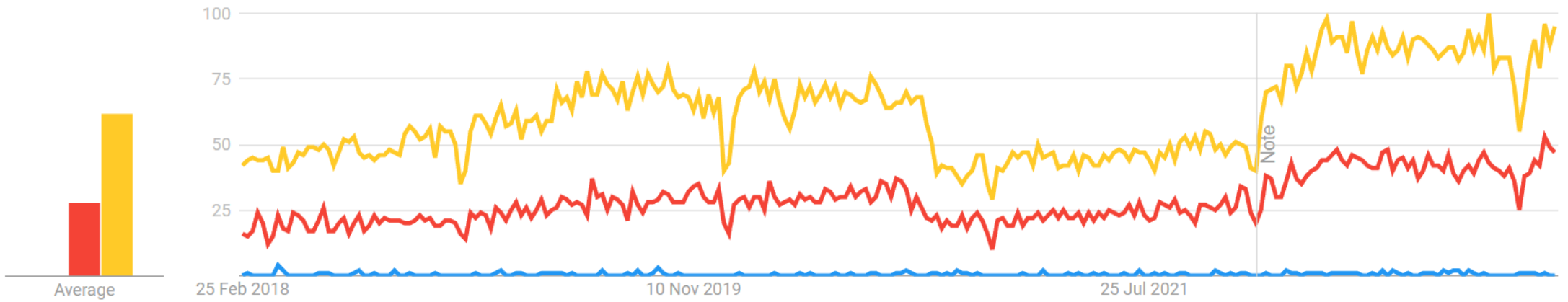
gRPC
Search term

GraphQL
Search term

+ Add comparison

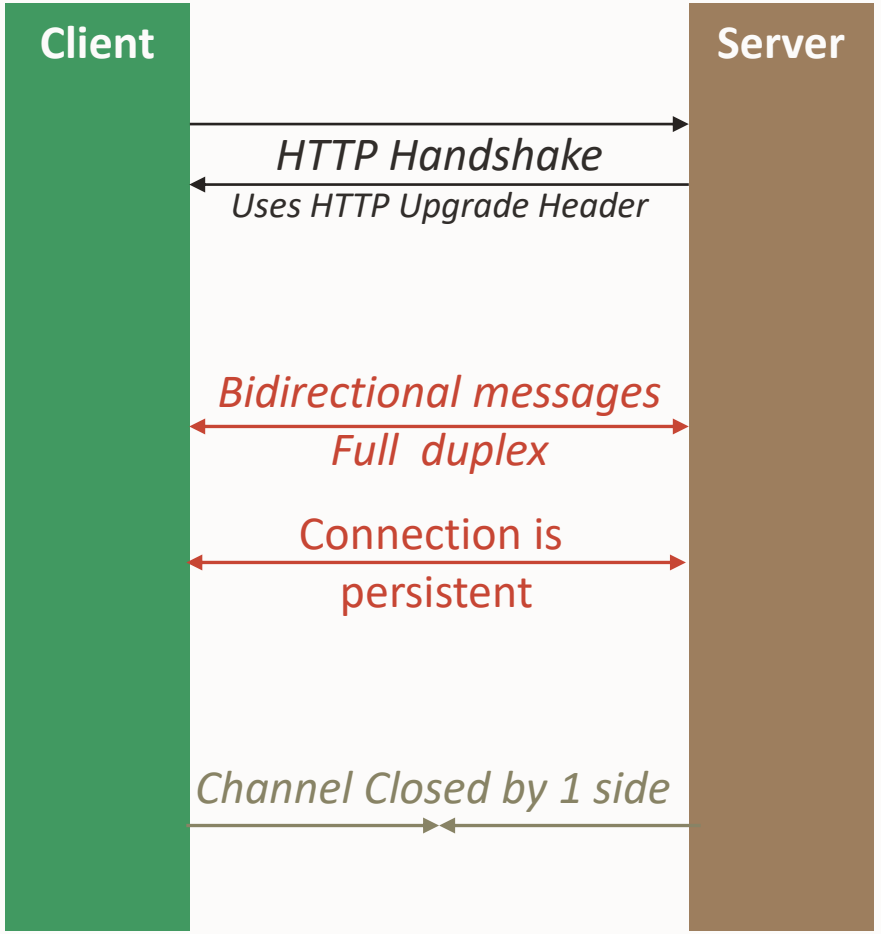
Worldwide | Past 5 years | Computers & Electronics | Web Search

Interest over time



Web Sockets

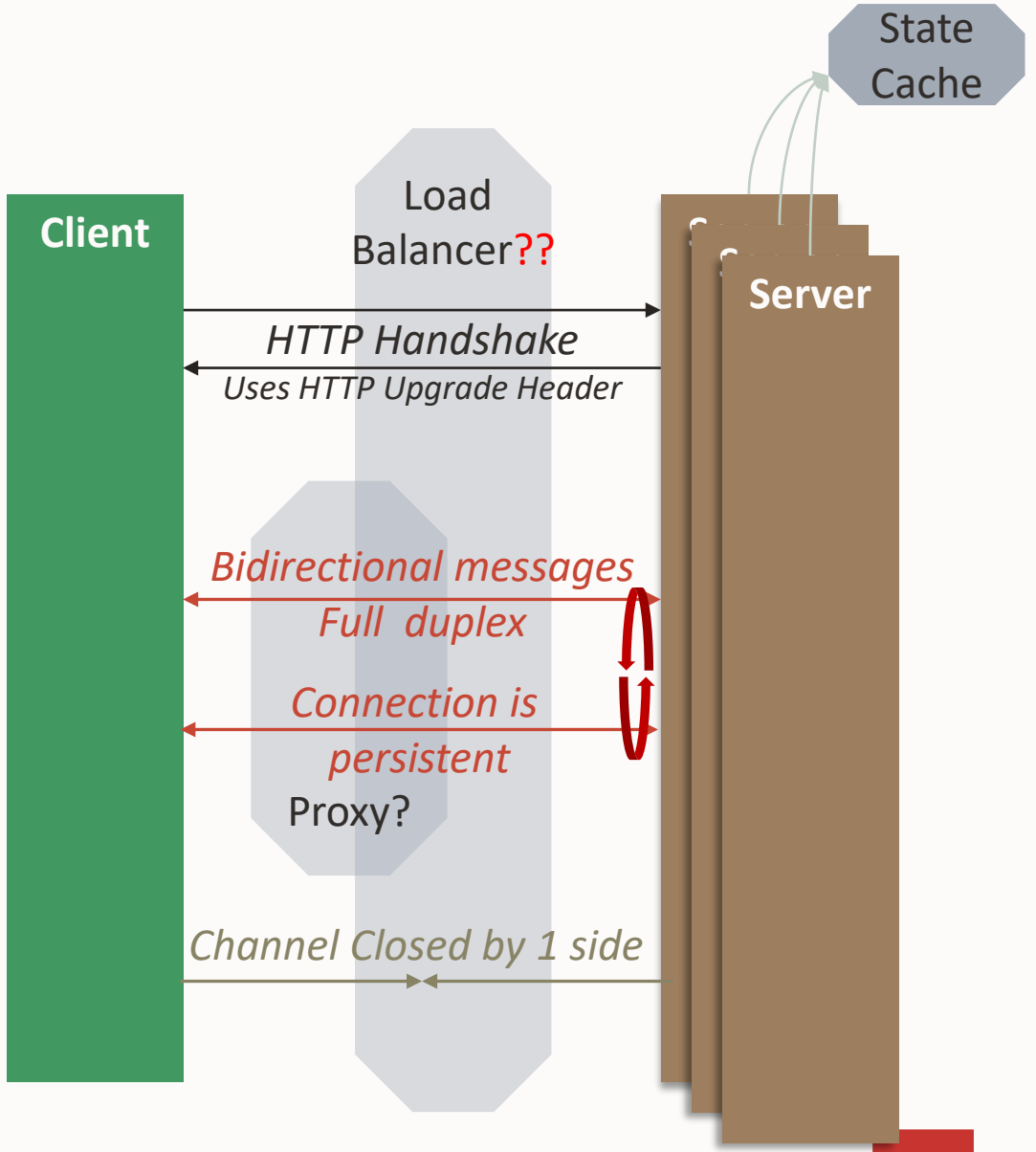
- Web Sockets (WS) have been around 10-15 years and formalized through IETF's RFC6455
  - There are a variety sub-protocols/specializations
    - Some recognized by IANA[2] e.g. STOMP & MQTT
    - Custom sub-protocols – not recognized by IANA e.g. something created yourself
- WS does have some challenges …
  - It works at a lower level than REST (emphasis on TCP rather than HTTP for traffic)
  - Some organizations choose to prevent sockets – the bidirectional nature means the risk of data egress.
  - Not same origin restrictions enforced like HTTP
  - Resource hungry – the socket is not multiplexed between requests but dedicated to 1 client
  - Need to recognize when the client has failed to close properly to release resources.



[1] https://caniuse.com/websockets
[2] https://www.iana.org/assignments/websocket/websocket.xml#subprotocol-name

Web Sockets

- More challenges …
  - Some web proxies can't differentiate between a WS connection and a normal HTTP request in a 'limbo' state
  - The conversation is stateful
    - therefore, impact on managing load balancing etc.
    - Depending on how data is exchanged over the socket – may need to track conversation state
- The benefits …
  - It is transient, so the client doesn't have a continuously open network port
  - About 98% of browsers support WS today[1]
  - Plenty of library implementations to ease the workload
  - Reduced overhead – 1 handshake until the communication completes



[1] https://caniuse.com/websockets

Server side

```
var WebSocketServer = require('ws').Server;
const wssPort = process.env.PORT || 8080;
const wss = new WebSocketServer({port: wssPort});
var clients = new Array;
```

```
function handleConnection(client, request) {
    clients.push(client);
    // add this client to the clients array

   function endClient() {
     var position = clients.indexOf(client);
     clients.splice(position, 1);
     console.log("connection closed");  }

   function clientResponse(data)  {
      console.log(data); }

 // set up client event listeners:
 client.on('message', clientResponse);
 client.on('close', endClient);
}
```

```
wss.on('connection', handleConnection);
```

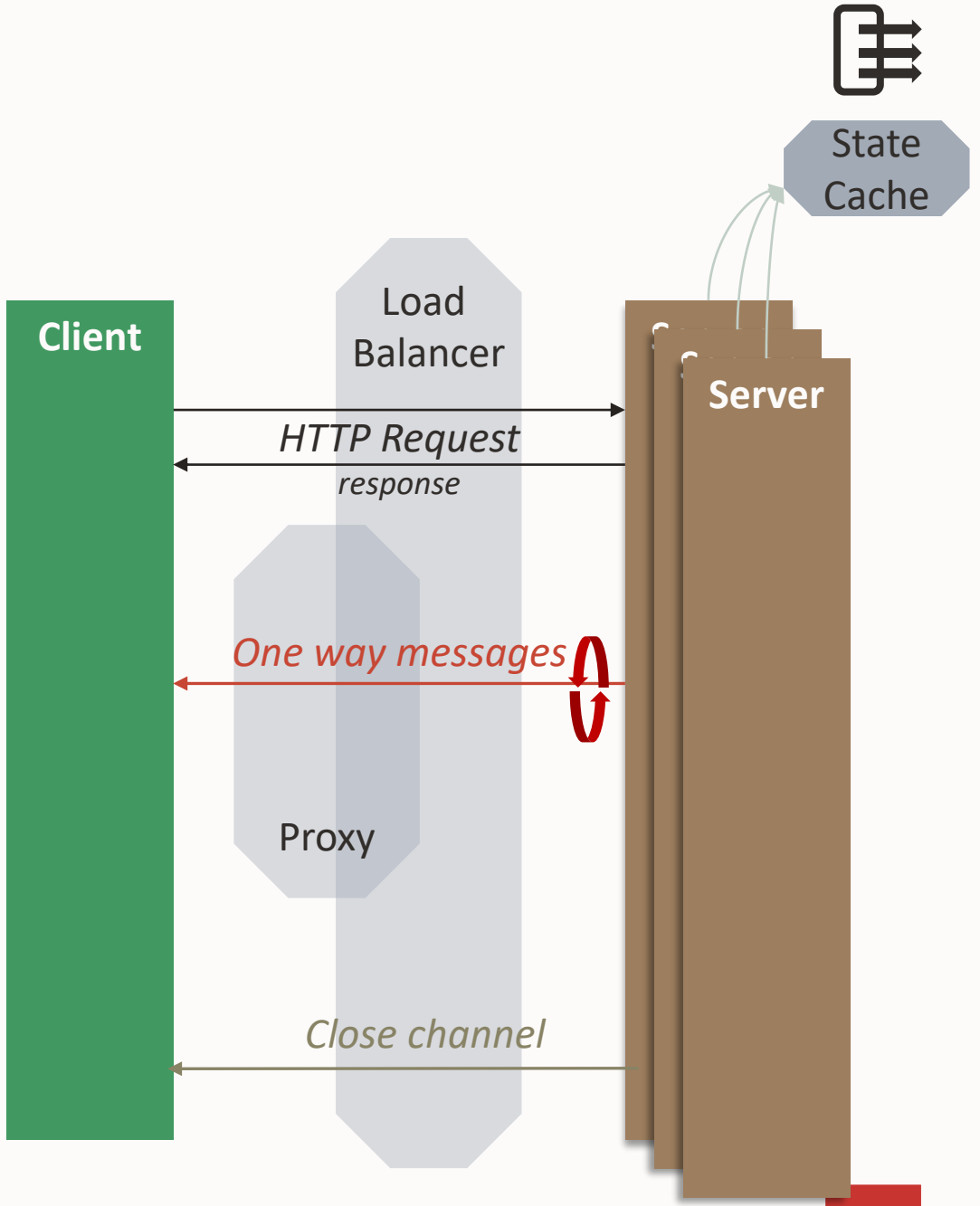
Client side

```
var WebSocket = require('ws');
var ws = new WebSocket('ws://localhost:8992/');

ws.on('open', function open() {
   data = …
  // something happens & prep data
  ws.send(data);
});

ws.on('error', function(error) {console.log(error);});

ws.on('message', function(data, flags) {
  console.log('Server said: ' + data);});
```

- Example uses Node.js with Web Socket library

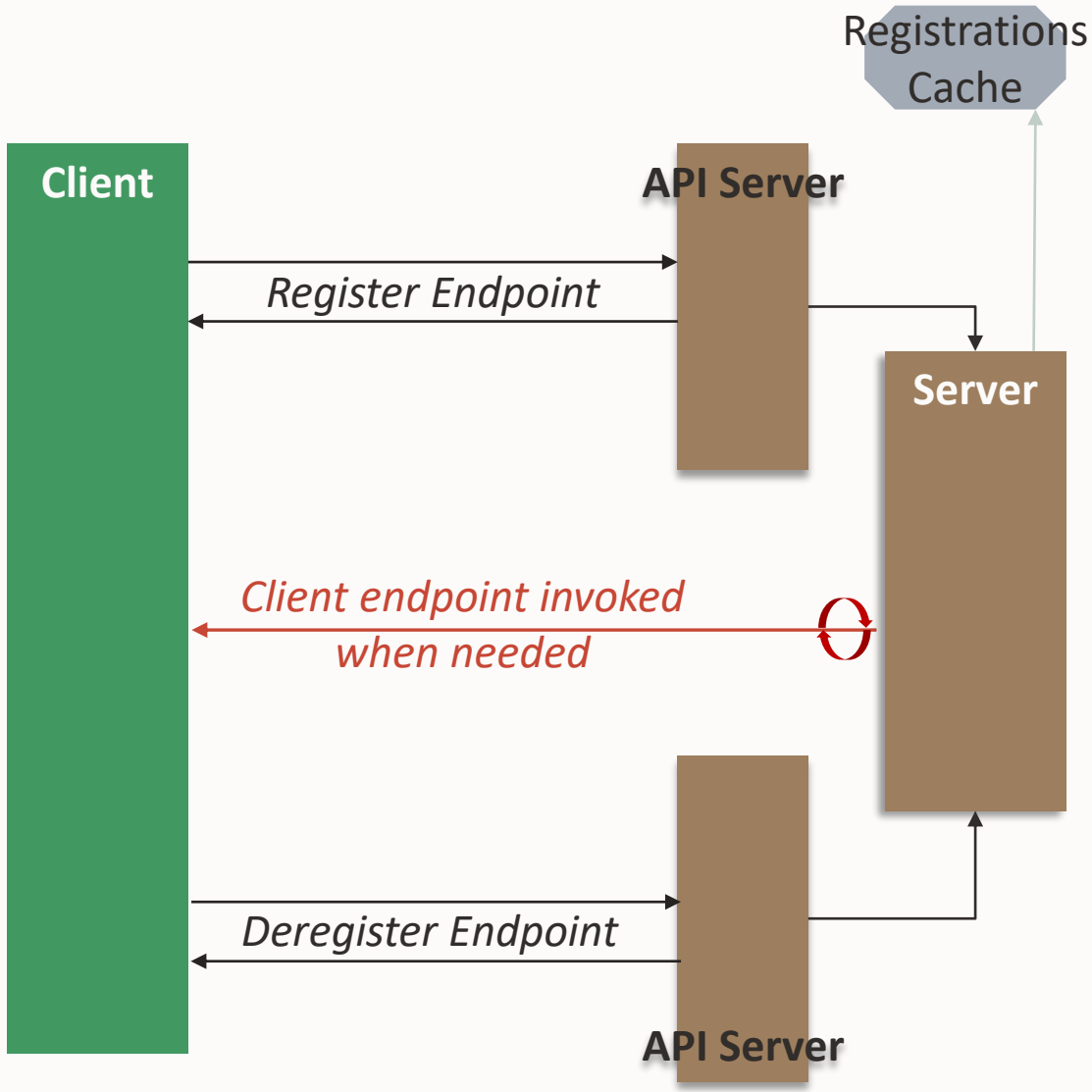
Server Sent Events

- Developed around 2006 - EventSource API is standardized as part of HTML5
  - Supported by all major browsers
- Process follows
  - Client supplies the server with the URL
  - Server calls the URL provided and sends a stream of events.
  - Once the server decides it is finished it closes the connection.
- Unlike Sockets – is only 1 direction and only closed by the server.
  - No elegant means to perform heartbeat or event ack
- Does focus on HTTP level exchanges rather than TCP – and gains the security restrictions
- More efficient than using long polling (call and wait for an event)



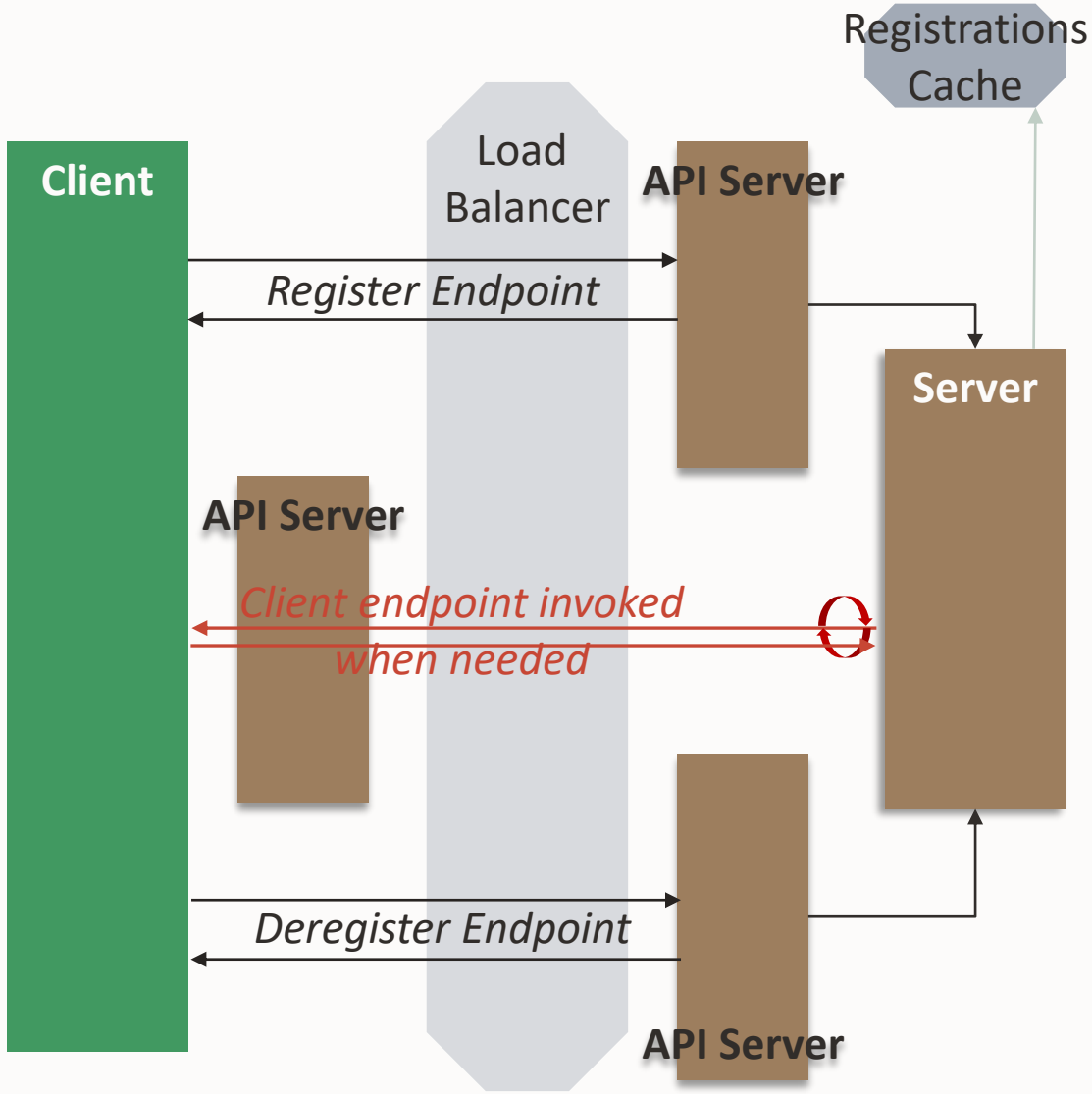
Web Hook

- Web Hooks (WH) is half duplex (i.e. 1 end communicating at a time)
- Client provides URI to be called on when something happens – just like any other API call
- Some challenges …
  - Client has a discoverable endpoint
  - Security is better when information is pulled NOT pushed
  - If clients are transient, risk of someone else getting the API call
  - Expose endpoint for URL
- Some benefits …
  - Simple to implement
  - Security management approaches can help protect clients, e.g., client registers with a key to be used when called
  - Easier to load balance, exploit common OOTB services such as an API Gateway for outbound to track data (audit, attach security creds etc)



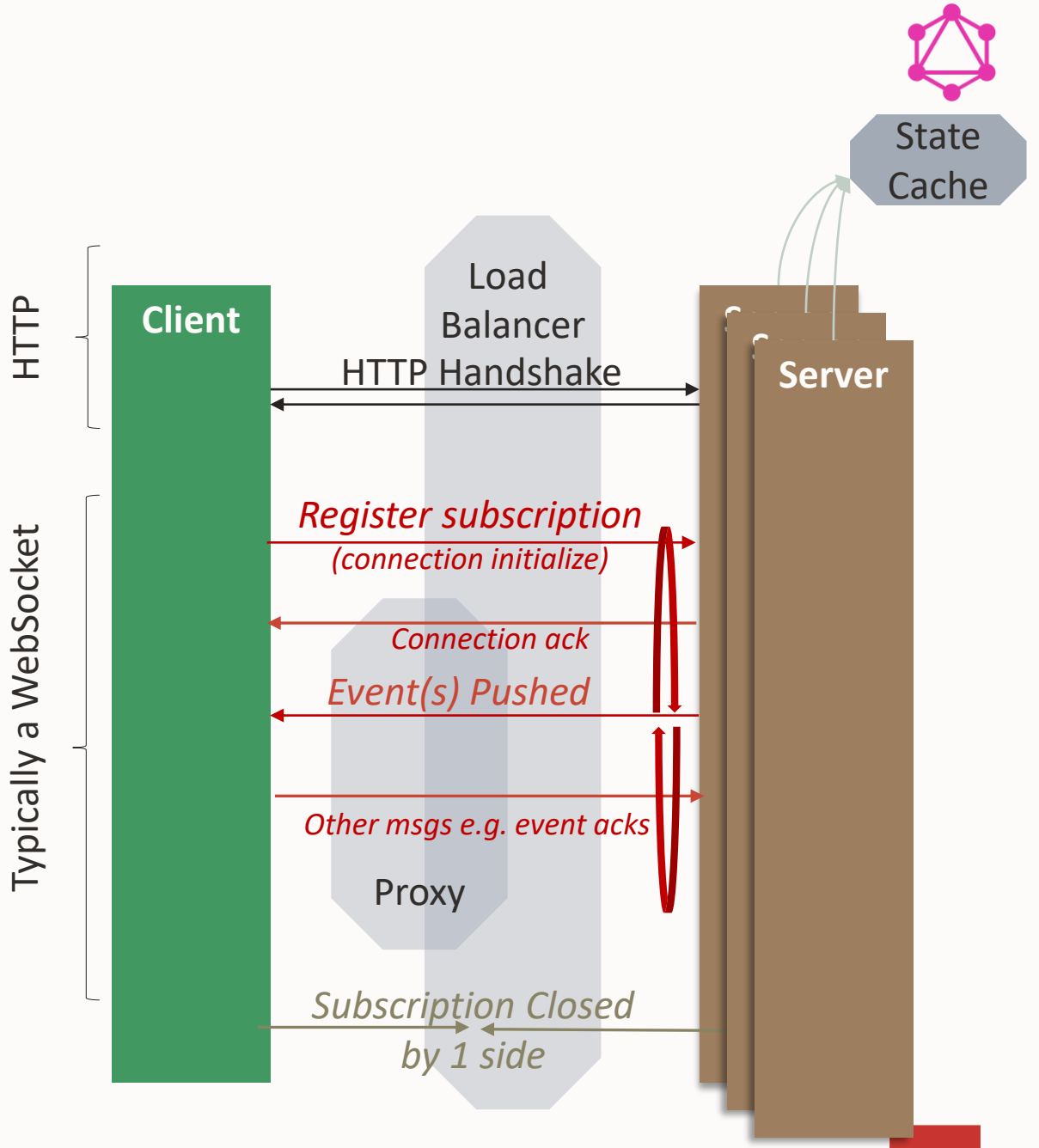
Web Hook

- Improve security through API Gateway
  - Audit outbound traffic
  - Authenticate transmission request (egress authorization)
  - Attach client-provided credentials to outbound traffic
  - Out of the box features



GraphQL Subscriptions

- GraphQL Subscriptions (GQL Subs) defines how subscriptions should functionally work – but not how to implement
  - Different implementations may use different strategies, but WS is the most common
- Benefits of GQL Subs…
  - Same benefits as a GQL request (tailored data set, option for attribute level access controls, etc.)
  - Lower-level mechanisms abstracted
- Challenges of GQL Subs …
  - Lot of work on the server – the ability to build cached answers for all harder
  - Need to consider potential variance or custom sub-protocols imposing client constraints



GraphQL – Basic Query

```
interface Character
{ id: ID!
  name: String!
  friends:
[Character]
  appearsIn:
[String]!
}
```

```
Droid implements
Character
{ id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  primaryFunction:
String
}
```

```
type Query {droid(id: ID!): Droid }
type Mutation {deleteDroid(id: ID!)
type Mutation (addDroid(newDroid: Droid!}
```

```
Query
{
 droid(id: "2000")
 {
  name,
    primaryFunction
 }
}
```

```
{
 "data":
 {
  "droid":
  {
   "name": "C-3PO“
  “primaryFunction”:

“Interpreter”
  }
 }
}
```

- Schemas with strong typing
- Schemas can define multiple entities
- Schemas support the idea of abstraction through interfaces
- Different entities can be linked via common attributes
- Schemas can define different types of operations
  - Query → get
  - Mutations → insert / update / delete
  - Subscriptions → live query
- Operations can then be used
- Operations can define the attributes to use/retrieve

Based on:https://github.com/graphql/swapi-graphql

GraphQL – Subscription

```
interface Character
{ id: ID!
  name: String!
  friends:
[Character]
  appearsIn:
[String]!
}
```

```
Droid implements
Character
{ id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  primaryFunction:
String
}
```

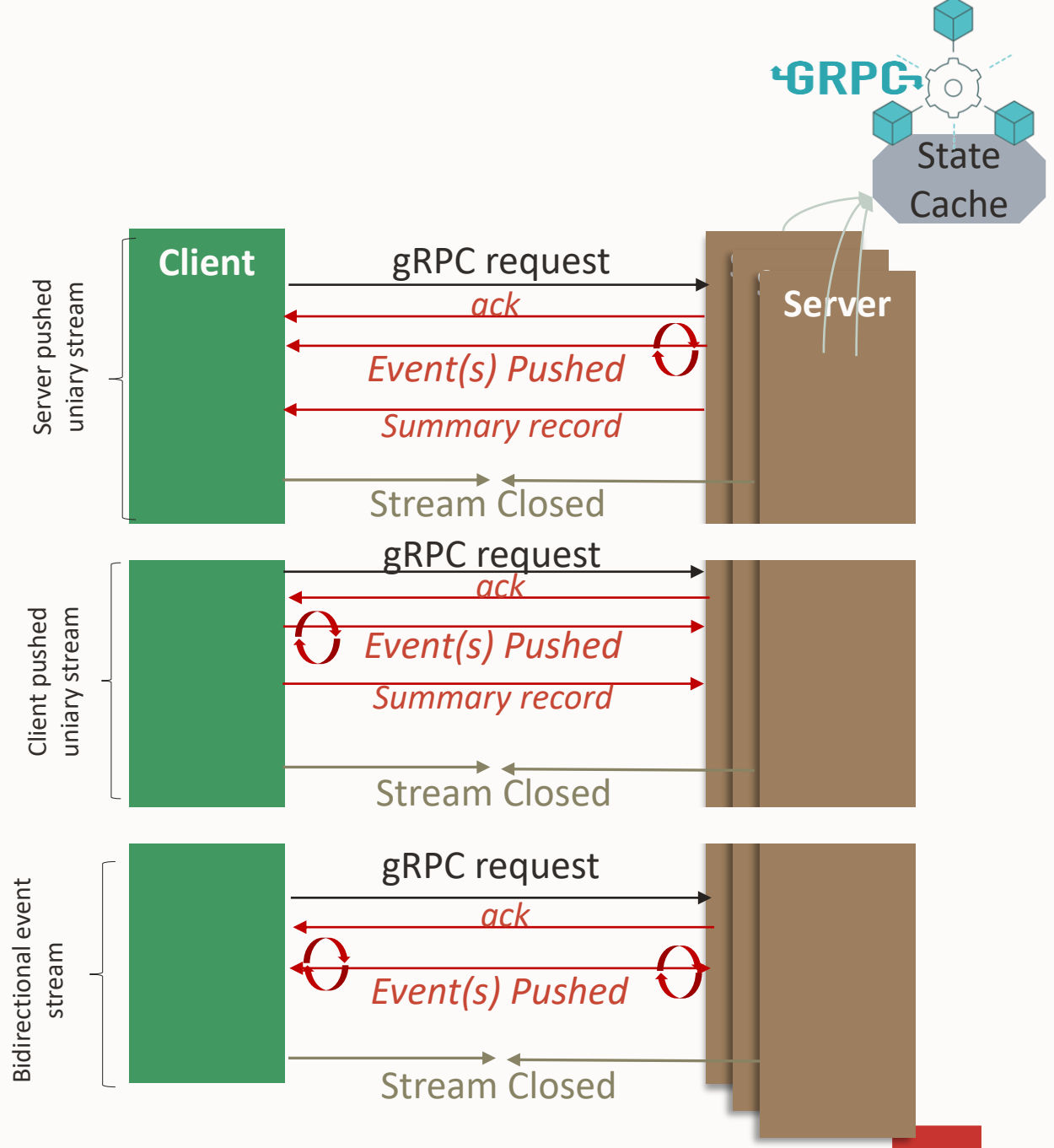
```
type Query {droid(id: ID!): Droid }
type subscription {characterUpdate(id: ID!}
{onCharacterUpdated(id : ID)
    {
      id: ID!
      name: String!
   }
}
```

- Subscription much like a query
- Primary difference is we're also stipulating what we want as the query when data changes.

Based on:https://github.com/graphql/swapi-graphql

gRPC Streams

- HTTP/2 foundation escape constraints of HTTP 1.x
  - Multiple streams per connection
  - Enabled by different flow control implementation
  - Greater network level efficiencies
  - Header compression & security mechanisms
  - Parallelism can be achieved
- Ability to have half or full duplex depending on needs
- HTTP/2 does have challenges
  - Does require infrastructure/software stack to support HTTP/2
  - Monitoring of HTTP/2 is harder to implement
- Streams can't be load balanced, so connectivity needs to be intelligently distributed when establishing the handshake/request



gRPC Expression

```
syntax = "proto3";

message Id {
  uint32 id = 1;}

message Character {
   Id id =1;
  string name =2;
  repeated string
     appearsIn =3;
  repeated Character
     friends =4;
}
```

```
message Droid
{
  Id id = 1;
  string
primaryFunction =2;
  Character character =
3;
}

message Ids {
  repeated uint32 id =
1;}
```

```
service GetService
{
  rpc getDroidById (Id) returns (Droid) {}
  rpc setCharacter (Character)  {}
  rpc getCharacters () returns  (stream
Character) {}
  rpc shareCharacters (stream Character)
    returns  (stream Character) {}
}
```

- The same considerations of normal data structure definitions still apply – streaming or otherwise
- The RPC definitions in Protobuf 2 & 3 define whether the invocation will stream

- The stream initiation can include metadata controlling the life of the stream (e.g. use of setting a deadline to receive the data)
- API design needs to consider whether the stream uses strategies like *ping-pong* to manage the delivery of messages
- The only difference between single calls and streams is the keyword ***stream*** in the rpc definition
- Position of ***stream*** in the rpc will dictate half or full duplex

Based on:https://github.com/graphql/swapi-graphql

Summary / Recommendations

	Web Hooks	Web Sockets	Server Sent Events (SSE)	GraphQL Subscriptions	gRPC Streams
Pros	• Technically simple & proven • Lowest common denominator • Each message can be ack'd to server in HTTP response	• Well supported • Same connection for bidirectional traffic • Should consider SDK to mask serialization	• Single direction calls • More efficient than webhooks for multiple events	• All the power of selective data from GraphQL • Often (not always) implemented using WebSockets • Bi-directional	• Very efficient • Easy to express once you know gRPC • Exploits HTTP/2 performance • Single or bidirectional flow
Cons	• Not very efficient • Client exposed endpoint for inbound calls	• More work as having to (de)serialize payloads • Lose some HTTP level security	• Not so commonly used • Client exposed endpoint for server to call • No means to ack each message	• Potential for differences in implementation – ideally provide client with additional info or SDK	• Client needs to have correct code frame • Needs HTTP/2

Useful Background resources

Streaming API Application

- Why Oracle Hospitality adopted streaming APIs - https://bit.ly/OHIPStreamingWhy
- How Oracle Hospitality decided on their streaming technology - https://bit.ly/OHIPStrategy
- JavaScript and Oracle Database subscribe to data changes - https://bit.ly/DBChangeSubscriptions
- Oracle Content Management uses GraphQL https://bit.ly/GraphQLOCM

Technology Resources

- Web Socket Examples using Node.js frameworks - https://tigoe.github.io/websocket-examples/
- Documentation on different Streaming mechanisms https://ably.com/
- Async API specification - https://www.asyncapi.com/docs

Oracle Cloud Free Tier – Special Promo

Try Always Free. No Time Limits.

Always Free

Services you can use for unlimited time

+

30-Day Free Trial

Free credits you can use for additional OCI services

~~300$~~ 500$ in Oracle Cloud Credits

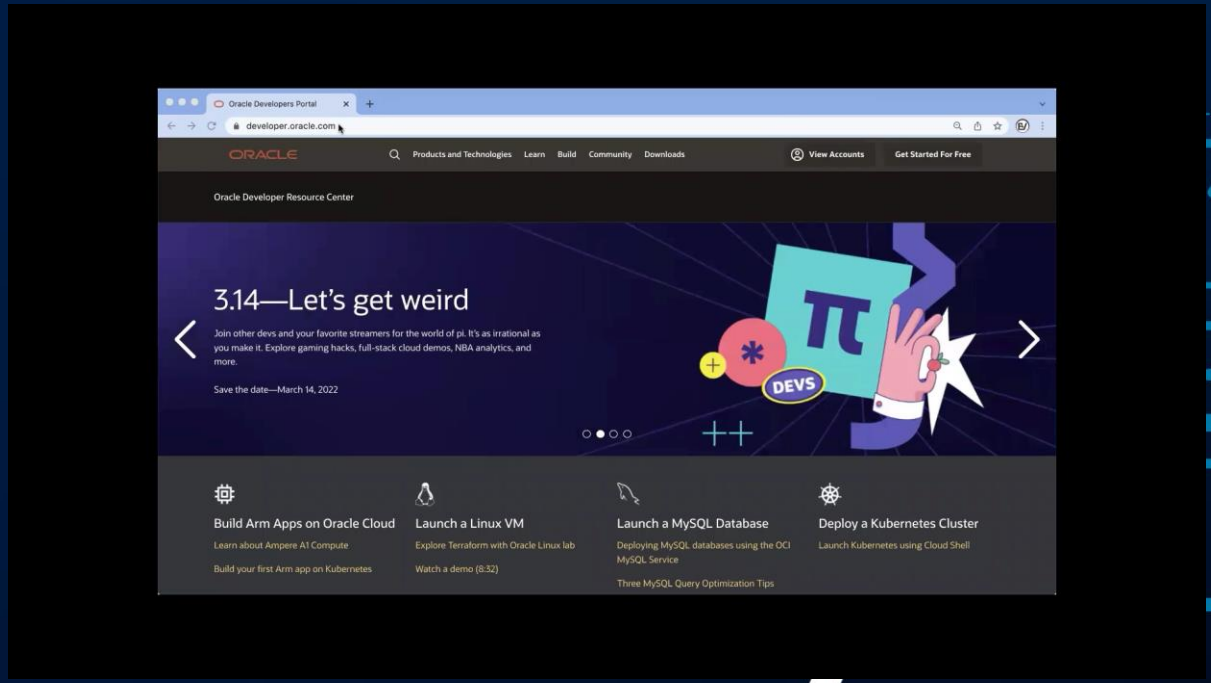
To be activated for this special promo:

- **Join our Public Slack Workspace and contact me**

Join our public Oracle DevRel Workspace

slack

oracledevrel.slack.com



Join the dedicated Slack channel to be part of the conversation and raise your questions to our Experts:

Step 1: Access the Slack OracleDevRel Workspace following this link:
http://bit.ly/odevrel_slack

Step 2: Search for Phil Wilkins - ORACLE
philip.wilkins@oracle.com

OCI Architecture Center -- Free Content & More

URLS are https://oracle.com/goto/...

Reference Architectures /ref-archs	Playbooks /playbooks	Built & Deployed /deployed	Live Labs /labs	Tutorials /tutorial	GitHub - DevRel /gh-devrel
GitHub - Samples /gh-samples		Developer /dev	Open Source /open		GitHub - Oracle /gh-oracle
Oracle Community /community	Learning Videos /youtube	Blogs /blog	Apex /apex	PaaS Community /paas	Cloud Customer Connect /connect

URLS are https://oracle.com/goto/...

or https://blog.mp3monster.org/oracle-resources/

Questions / Thank you

Phil Wilkins

Cloud Developer Evangelist

Philip.Wilkins@Oracle.com
bit.ly/devrel-slack-emea @Phil Wilkins

mp3monster.org / cloud-native.info / oracle-integration.cloud
linkedin.com/in/philwilkins
github.com/mp3monster
@mp3monster


ORACLE