

DEC 2022

API Styles Fundamentals

by Lukasz Dynowski

linkedin.com/in/ldynia



GitHub Fundamentals—with Interactivity

Published by [O'Reilly Media, Inc.](#)



[**</> Interactive**](#)

[What you'll learn](#) [Is this live event for you?](#) [Schedule](#)

GitHub is a central hub for most open source projects, and it's the biggest hosting platform for storing source code versioned by Git. Yet it still manages to be highly underutilized. Correctly managed, GitHub can be a proven record of your work and interests, a résumé that can be used as leverage in salary negotiations, a hosting platform to promote yourself or your projects, a CI/CD platform to deliver your software to millions, and a fantastic collaborative tool for your team to facilitate communication among developers. And GitHub can also be your go-to place for solving daily programming challenges and fixations.

Join expert Lukasz Dynowski to learn everything you need to know to get started with GitHub. In a little over an hour, you'll build a complete open source project, create a web page and a wiki to promote it, and discover how to maintain and collaborate with others on your project.

YOUR INSTRUCTOR



Lukasz Dynowski

Lukasz Dynowski is an independent consultant who in his career was involved in over 150+ projects. Counting over 10 years of experience as a software engineer Lukasz was doing Full-Stack, DevOps, Software Architecture, as well as...

[Read more](#)



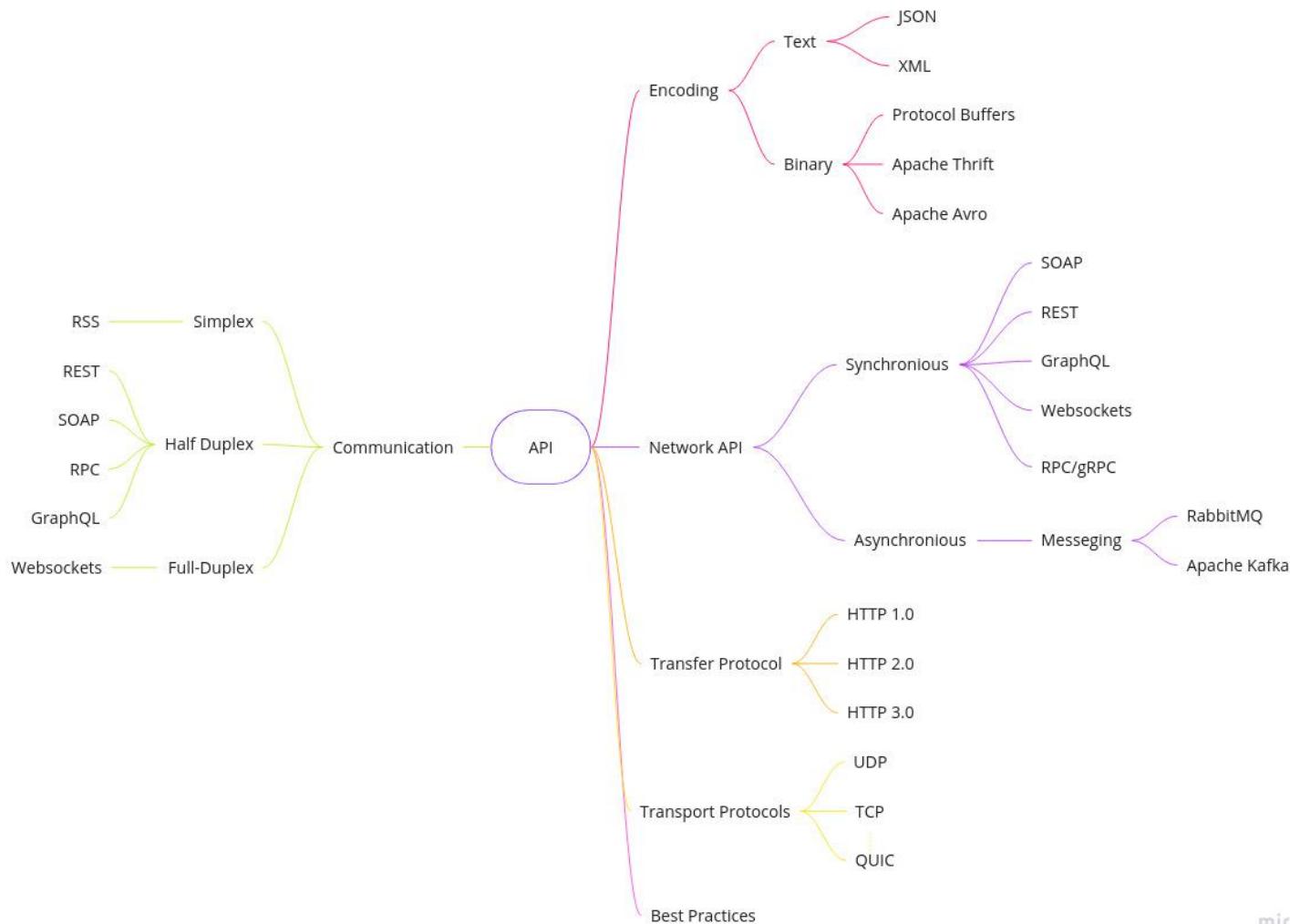
Objectives

Preliminary

- Common API styles
- Offering of each API style and its trade-offs
- The counterpart technology for every API
- When to use appropriate API style
- Characteristics of good API

Secondary

- Historical context that lead to creation of APIs
- The OSI model and telecommunication transmission models
- Common encodings used in the context of APIs



Why API?

75%

of organizations will be completely digitally transformed in the next decade. Those that do not transform will not survive *

up to
30%
of revenues via
digital channels

By 2022

90%

of all new apps will feature microservices architecture; 35% of all production apps will be cloud native **

What is an interface?

Dictionary

Definitions from [Oxford Languages](#) · [Learn more](#)



interface

noun

1. a point where two systems, subjects, organizations, etc. meet and interact.
"the interface between accountancy and the law"
2. **COMPUTING**
a device or program enabling a user to communicate with a computer.
"a graphical user interface"

verb

1. interact with (another system, person, etc.).
"you will interface with counterparts from sister companies"
2. **COMPUTING**
connect with (another computer or piece of equipment) by an interface.
"the hotel's computer system can interface automatically with the booking system"

User Interface



By Evan-Amos - Own work, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=35155956>

Terminal User Interface

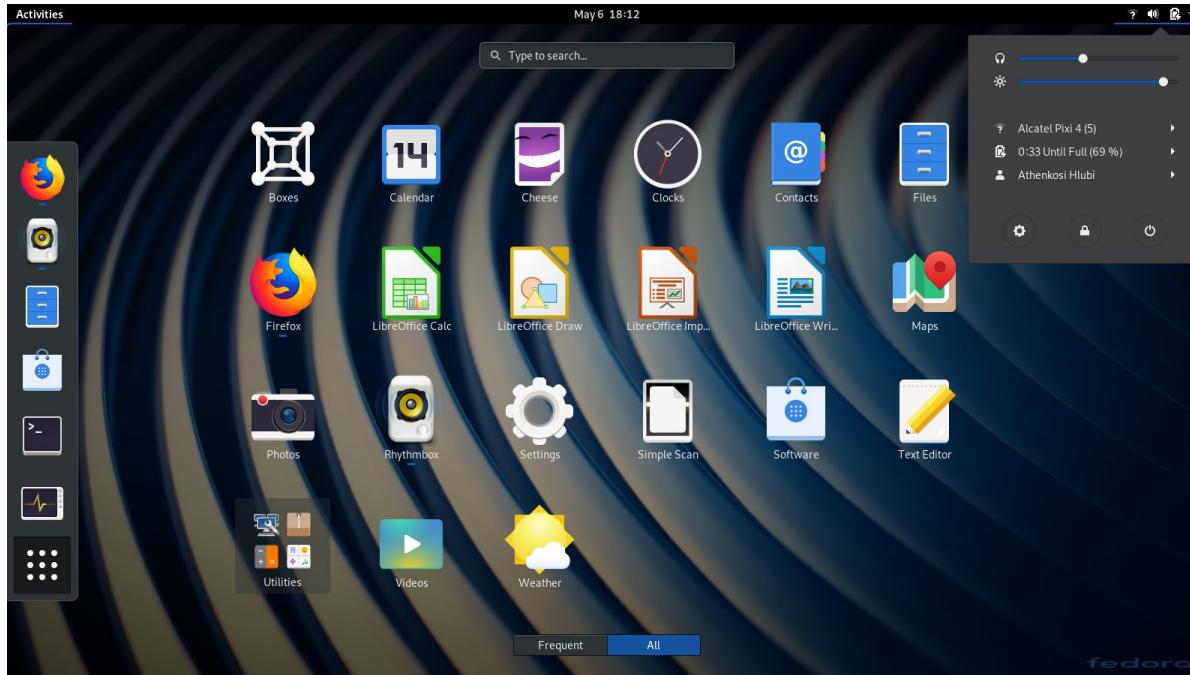
```
61 +-- 3 lines: Is a socket 'connection oriented' ?-----|211 EXPORT_SYMBOL(netpoll_poll_dev);
64 static inline int connection_based(struct sock *sk) |212
65 +-- 3 lines: {-----|213 void netpoll_poll(struct netpoll *np)
68 |214 +-- 3 lines: {-----|217 EXPORT_SYMBOL(netpoll_poll);
69 static int receiver_wake_function(wait_queue_t *wait, unsigned |218
70 |219 static void refill_skbs(void)
71 |220 +-- 14 lines: {-----|234
71 +-- 10 lines: {-----|221 EXPORT_SYMBOL(netpoll_zap_completion);
72 |222 static void zap_completion_queue(void)
73 |223 +-- 26 lines: {-----|235
73 +-- 3 lines: Wait for a packet..|224
74 |225 static struct sk_buff *find_skb(struct netpoll *np, int len, int
75 |226 +-- 24 lines: {-----|236
75 +-- 45 lines: {-----|227
76 |228 static int netpoll_owner_active(struct net_device *dev)
77 |229 +-- 9 lines: {-----|237
77 +-- 2 lines: {-----|238
78 |239 void netpoll_send_skb_on_dev(struct netpoll *np, struct sk_buff *
79 |240 |241 struct net_device *dev)
79 +-- 54 lines: {-----|242
80 |243 EXPORT_SYMBOL(netpoll_send_skb_on_dev);
81 |244 +-- 59 lines: {-----|245
81 +-- 6 lines: net/core/dst.c Protocol independent destination c|246 EXPORT_SYMBOL(netpoll_send_udp);
82 |247 void netpoll_send_udp(struct netpoll *np, const char *msg, int le
83 |248 +-- 116 lines: {-----|249
83 +-- 6 lines: {-----|250 static void arp_reply(struct sk_buff *skb)
84 |251 +-- 90 lines: {-----|252
84 +-- 6 lines: {-----|253 int __netpoll_rx(struct sk_buff *skb)
85 |254 +-- 14 lines: {-----|255
85 +-- 6 lines: {-----|256 void netpoll_print_options(struct netpoll *np)
86 |257 +-- 30 lines: {-----|258 EXPORT_SYMBOL(netpoll_print_options);
86 +-- 30 lines: {-----|259
87 |260
87 +-- 30 lines: {-----|261
88 |262
88 +-- 30 lines: {-----|263
89 |264
89 +-- 30 lines: {-----|265
90 |266
90 +-- 30 lines: {-----|267
91 |268
91 +-- 30 lines: {-----|269
92 |270
92 +-- 30 lines: {-----|271
93 |272
93 +-- 30 lines: {-----|273
94 |274
94 +-- 30 lines: {-----|275
95 |276
95 +-- 30 lines: {-----|277
96 |278
96 +-- 30 lines: {-----|279
97 |280
97 +-- 30 lines: {-----|281
98 |282
98 +-- 30 lines: {-----|283
99 |284
99 +-- 30 lines: {-----|285
100 |286
100 +-- 30 lines: {-----|287
101 |288
101 +-- 30 lines: {-----|289
102 |290
102 +-- 30 lines: {-----|291
103 |292
103 +-- 30 lines: {-----|293
104 |294
104 +-- 30 lines: {-----|295
105 |296
105 +-- 30 lines: {-----|297
106 |298
106 +-- 30 lines: {-----|299
107 |299
107 +-- 30 lines: {-----|300
108 |301
108 +-- 30 lines: {-----|301
109 |302
109 +-- 30 lines: {-----|302
110 |303
110 +-- 30 lines: {-----|303
111 |304
111 +-- 30 lines: {-----|304
112 |305
112 +-- 30 lines: {-----|305
113 |306
113 +-- 30 lines: {-----|306
114 |307
114 +-- 30 lines: {-----|307
115 |308
115 +-- 30 lines: {-----|308
116 |309
116 +-- 30 lines: {-----|309
117 |310
117 +-- 30 lines: {-----|310
118 |311
118 +-- 30 lines: {-----|311
119 |312
119 +-- 30 lines: {-----|312
120 |313
120 +-- 30 lines: {-----|313
121 |314
121 +-- 30 lines: {-----|314
122 |315
122 +-- 30 lines: {-----|315
datagram.c 132,1 8% 300 void netpoll_send_skb_on_dev(struct netpoll *np, struct sk_buff *
1 +-- 6 lines: net/core/dst.c Protocol independent destination c|301 |301 struct net_device *dev)
7 |302 +-- 54 lines: {-----|302
8 #include <linux/bitops.h> |303 EXPORT_SYMBOL(netpoll_send_skb_on_dev);
9 #include <linux/errno.h> |304
10 #include <linux/init.h> |305
11 #include <linux/kernel.h> |306 void netpoll_send_udp(struct netpoll *np, const char *msg, int le
12 #include <linux/workqueue.h> |307 +-- 59 lines: {-----|307
13 #include <linux/mm.h> |308 EXPORT_SYMBOL(netpoll_send_udp);
14 #include <linux/module.h> |309
15 #include <linux/slab.h> |310 static void arp_reply(struct sk_buff *skb)
16 #include <linux/netdevice.h> |311 +-- 116 lines: {-----|311
17 #include <linux/skbuff.h> |312
18 #include <linux/string.h> |313 int __netpoll_rx(struct sk_buff *skb)
19 #include <linux/types.h> |314 +-- 90 lines: {-----|314
20 #include <net/net_namespace.h> |315 void netpoll_print_options(struct netpoll *np)
21 #include <linux/sched.h> |316 +-- 14 lines: {-----|316 EXPORT_SYMBOL(netpoll_print_options);
22 |317
dst.c 1,1 Top netpoll.c 645, 30 39%
```

Command Line Interface

```
[root@localhost ~]# ping -q fa.wikipedia.org
PING text.pmtpa.wikimedia.org (208.80.152.2) 56(84) bytes of data.
^C
--- text.pmtpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 540.528/540.528/540.528/0.000 ms
[root@localhost ~]# pwd
/root
[root@localhost ~]# cd /var
[root@localhost var]# ls -la
total 72
drwxr-xr-x. 18 root root 4096 Jul 30 22:43 .
drwxr-xr-x. 23 root root 4096 Sep 14 20:42 ..
drwxr-xr-x. 2 root root 4096 May 14 00:15 account
drwxr-xr-x. 11 root root 4096 Jul 31 22:26 cache
drwxr-xr-x. 3 root root 4096 May 18 16:03 db
drwxr-xr-x. 3 root root 4096 May 18 16:03 empty
drwxr-xr-x. 2 root root 4096 May 18 16:03 games
drwxrwx--T. 2 root gdm 4096 Jun 2 18:39 gdm
drwxr-xr-x. 38 root root 4096 May 18 16:03 lib
drwxr-xr-x. 2 root root 4096 May 18 16:03 local
lrwxrwxrwx. 1 root root 11 May 14 00:12 lock -> ../run/lock
drwxr-xr-x. 14 root root 4096 Sep 14 20:42 log
lrwxrwxrwx. 1 root root 10 Jul 30 22:43 mail -> spool/mail
drwxr-xr-x. 2 root root 4096 May 18 16:03 nis
drwxr-xr-x. 2 root root 4096 May 18 16:03 opt
drwxr-xr-x. 2 root root 4096 May 18 16:03 preserve
drwxr-xr-x. 2 root root 4096 Jul 1 22:11 report
lrwxrwxrwx. 1 root root 6 May 14 00:12 run -> ../run
drwxr-xr-x. 14 root root 4096 May 18 16:03 spool
drwxrwxrwt. 4 root root 4096 Sep 12 23:50 tmp
drwxr-xr-x. 2 root root 4096 May 18 16:03 yp
[root@localhost var]# yum search wiki
Loaded plugins: langpacks, presto, refresh-packagekit, remove-with-leaves
| rpmfusion-free-updates | 2.7 kB 00:00
| rpmfusion-free-updates/primary_db | 266 kB 00:04
| rpmfusion-nonfree-updates | 2.7 kB 00:00
| updates/metalink | 5.9 kB 00:00
| updates | 4.7 kB 00:00
| updates/primary_db | 2.6 MB 00:15 ETA
[ 73% [=====] ] 62 kB/s
```

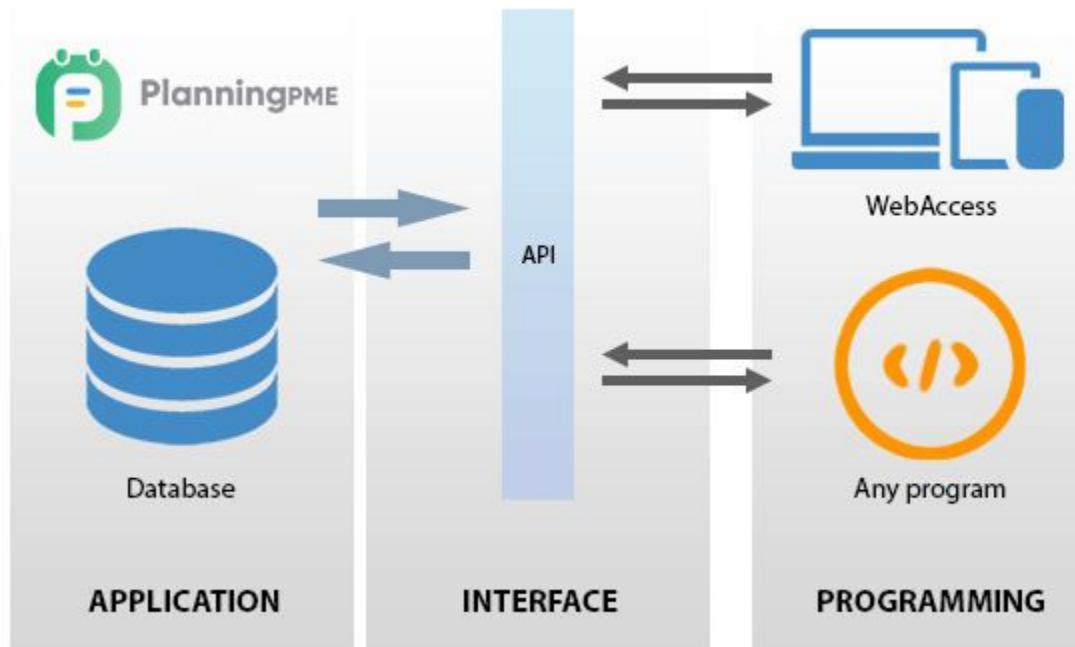
By ZxxZxxZ - Own work, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=16485488>

Graphical User Interface



By Athenkosi Hlubi - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=78751426>

Application Programming Interface

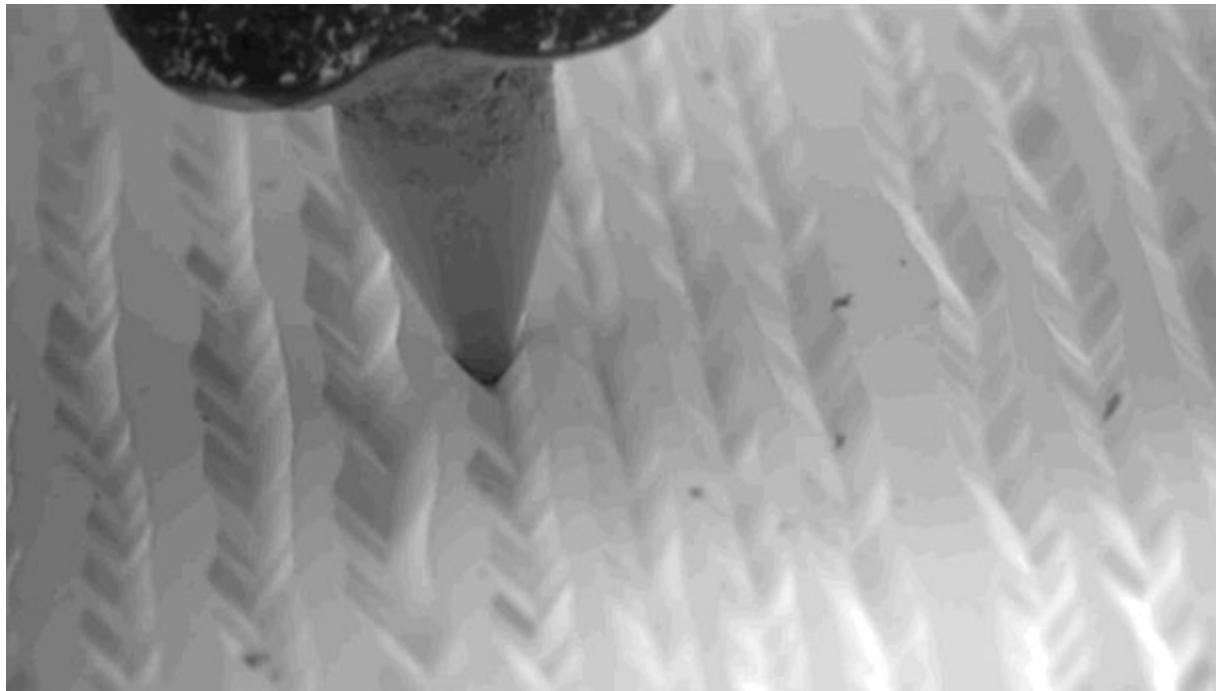


<https://www.planningpme.com/img/planningpme-api.jpg>

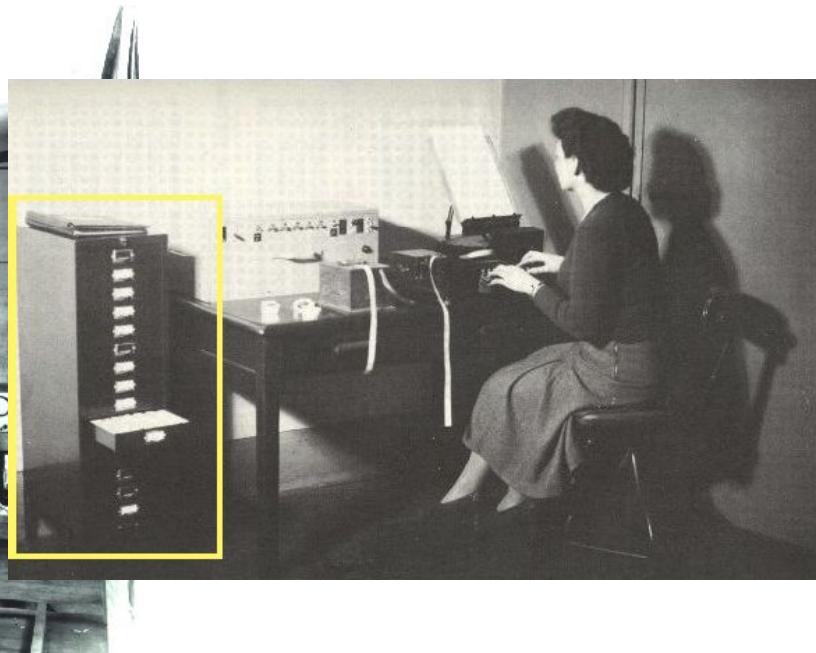
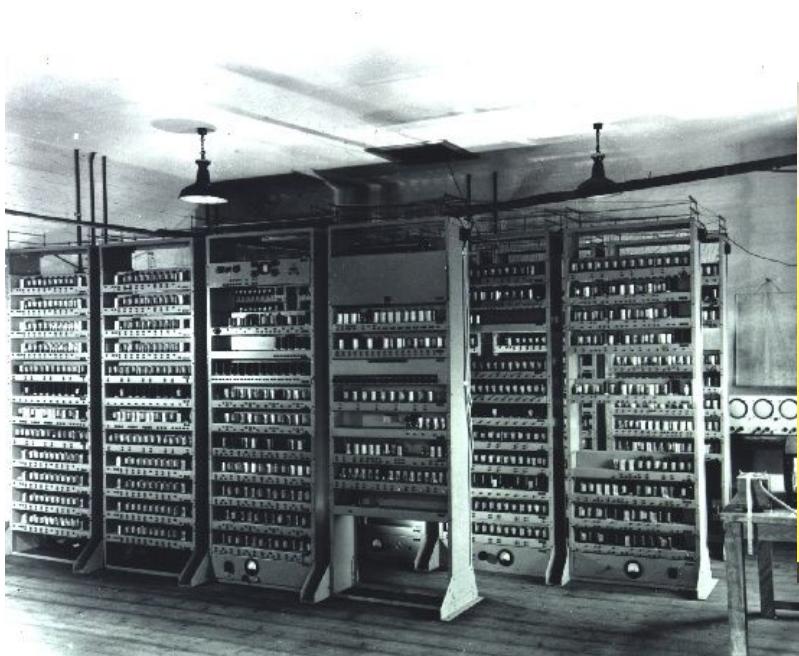
What is the purpose of an API?

- To bridge relevant applications / services.
- To hide the inner workings of a system and exposes only the parts that a programmer will find useful and keep them consistent even if the internal details of that system change later.
- To help create a standard for reusable and easily understood by the users components.
- The defined scope of API helps in the designing, testing, building, managing, and versioning of the component.

Vinyl



History of API - 40s EDSAC



History of API - 60s

Data structures and techniques for remote computer graphics

by IRA W. COTTON

*Sperry Rand Corporation
Philadelphia, Pennsylvania*

and

FRANK S. GREATOREX, JR.

*Adams Associates
Bedford, Massachusetts*

INTRODUCTION

It has been adequately demonstrated^{1,2,3} that computer graphics systems need not require the dedication of a large scale computer for their operation. Computer graphics has followed the trends of computing in general, where remote access, time sharing, and multi-programming have become the key phrases. The prob-

and of relating the appearances of a picture on the tube face to its description in the data structure.

Figures 1 and 2 illustrate a typical configuration as it

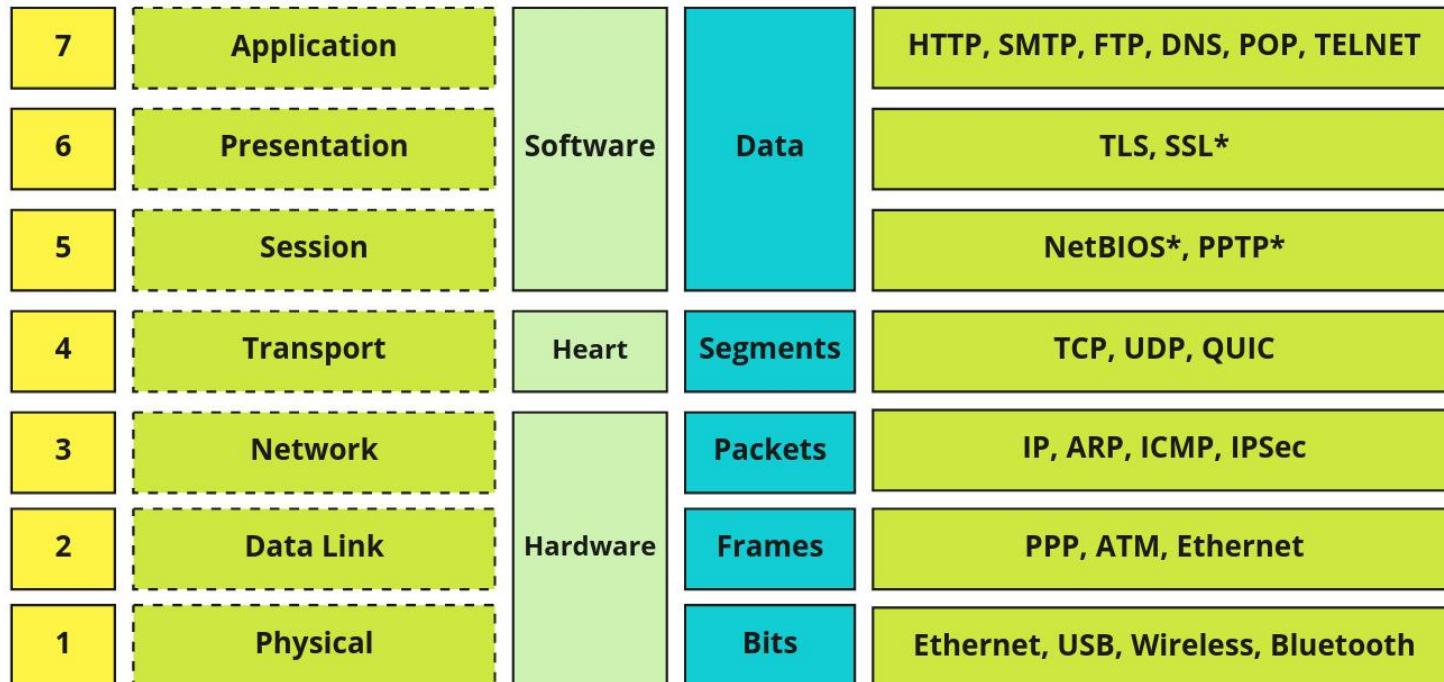


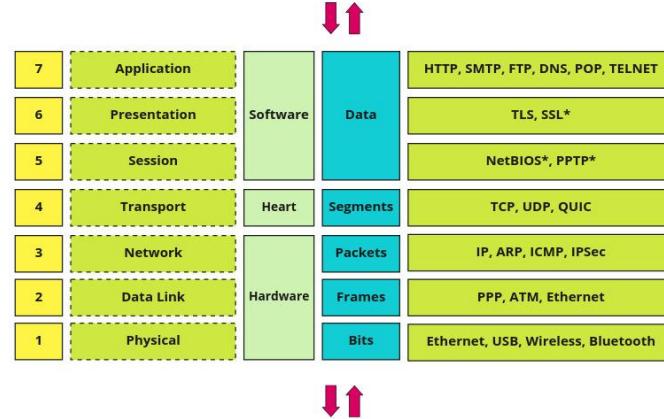
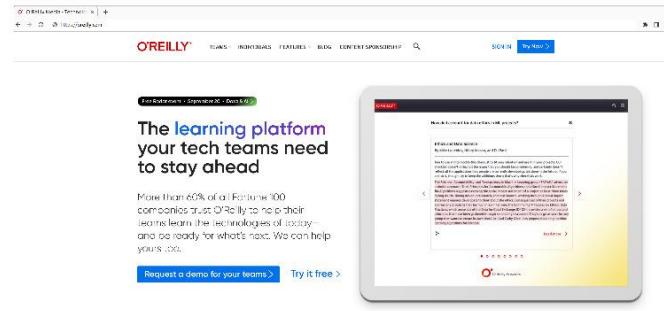
trolled tools comes immediately to mind). Finally, hardware independence at the central computer means that a consistent application program interface could be maintained if that computer were replaced; this also

History of API - 90s and 2000s

- **90s** - Program executed on computer located elsewhere. Expanded the idea of API
- **2000s** - Roy Fielding “network-based Application Programming Interface” - Web API

Open Systems Interconnection (OSI)



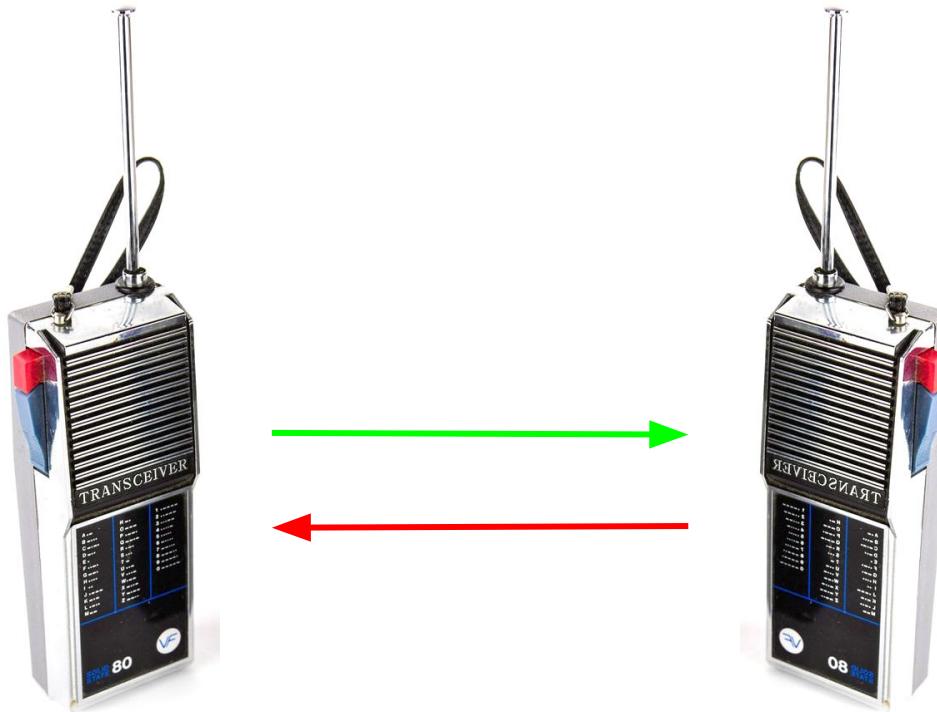


TRANSMISSION MODES

Simplex



Half-Duplex



Full-Duplex



When to use what?

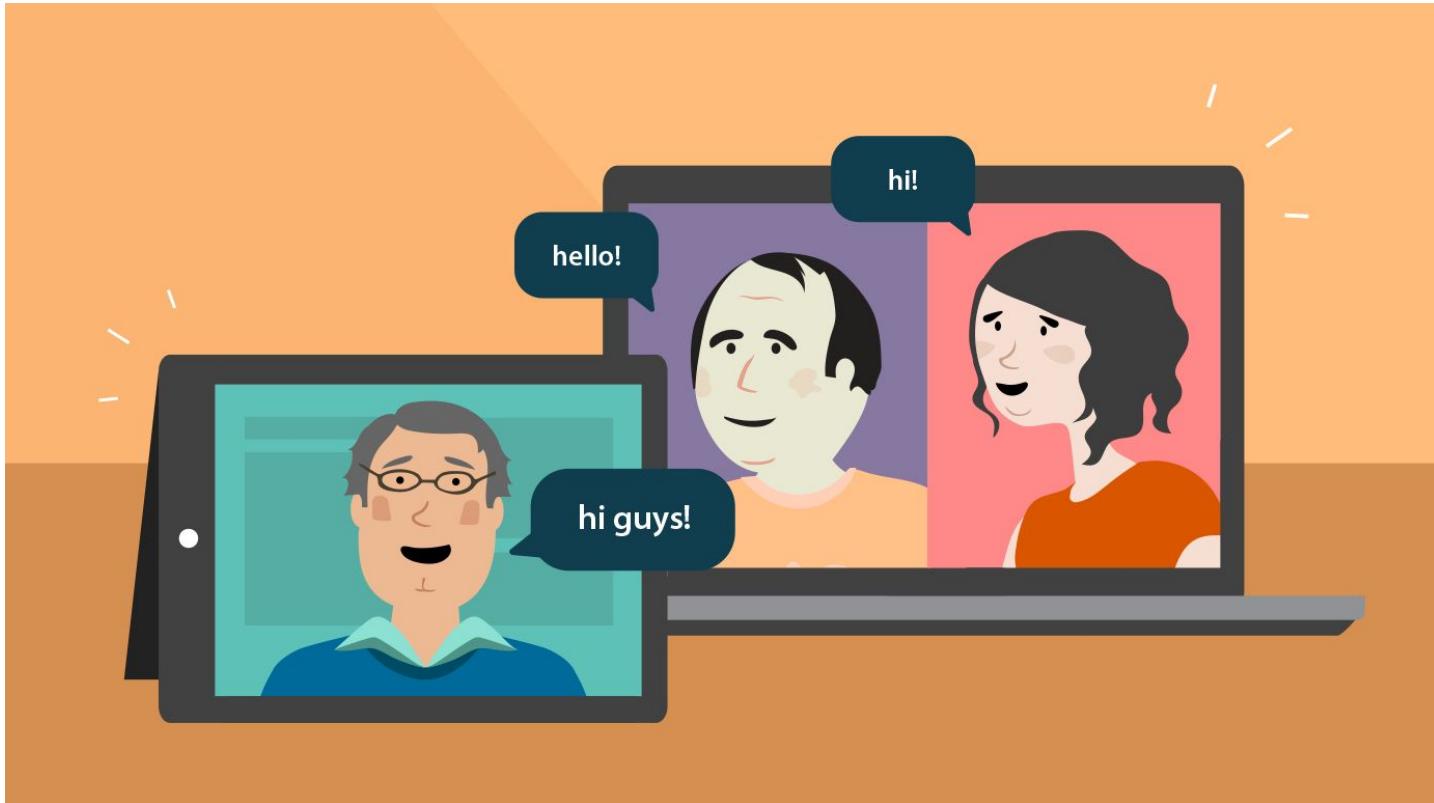
Simplex - pull based broadcasting

Half-Duplex - for system that can operate at worst with large latencies

Full-Duplex - for system that requires real time information exchange

COMMUNICATION STYLES

Synchronous



Asynchronous



When to use what?

Synchronous communication is easy to reason about, but it can lead to lack of scalability, reliability, elasticity and other architectural characteristics.

Asynchronous communication benefits from performance and scale however, at the cost of data synchronization, deadlocks, race conditions, debugging, and so on.

Synchronous by default, asynchronous when needed.

TRANSPORT PROTOCOLS

UDP

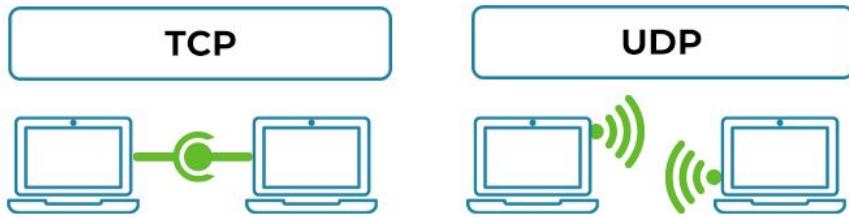
Bits

0	16	31
SOURCE PORT NUMBER		DESTINATION PORT NUMBER
LENGTH		CHECKSUM

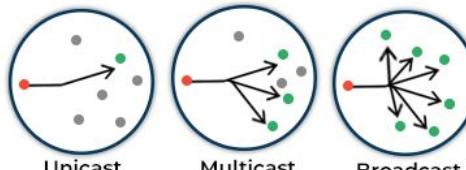
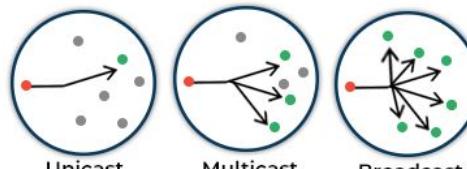
TCP

Bits					
0	8	16	31		
Source Port		Destination Port			
Sequence Number					
Acknowledgment Number					
Data Offset	Reserved	Code	Window		
Checksum		Urgent Pointer			
Options			Padding		
Data					

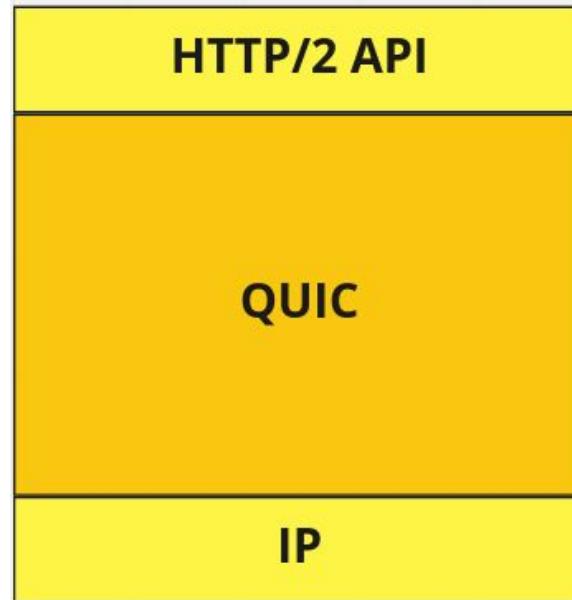
TCP vs UDP



- Slower but more reliable transfers
 - Typical Applications:
 - File Transfer Protocol
 - Web Browsing
 - Email
- Faster but not guaranteed transfer ("Best Effort")
 - Typical Applications:
 - Live streaming
 - Online Games
 - VoIP

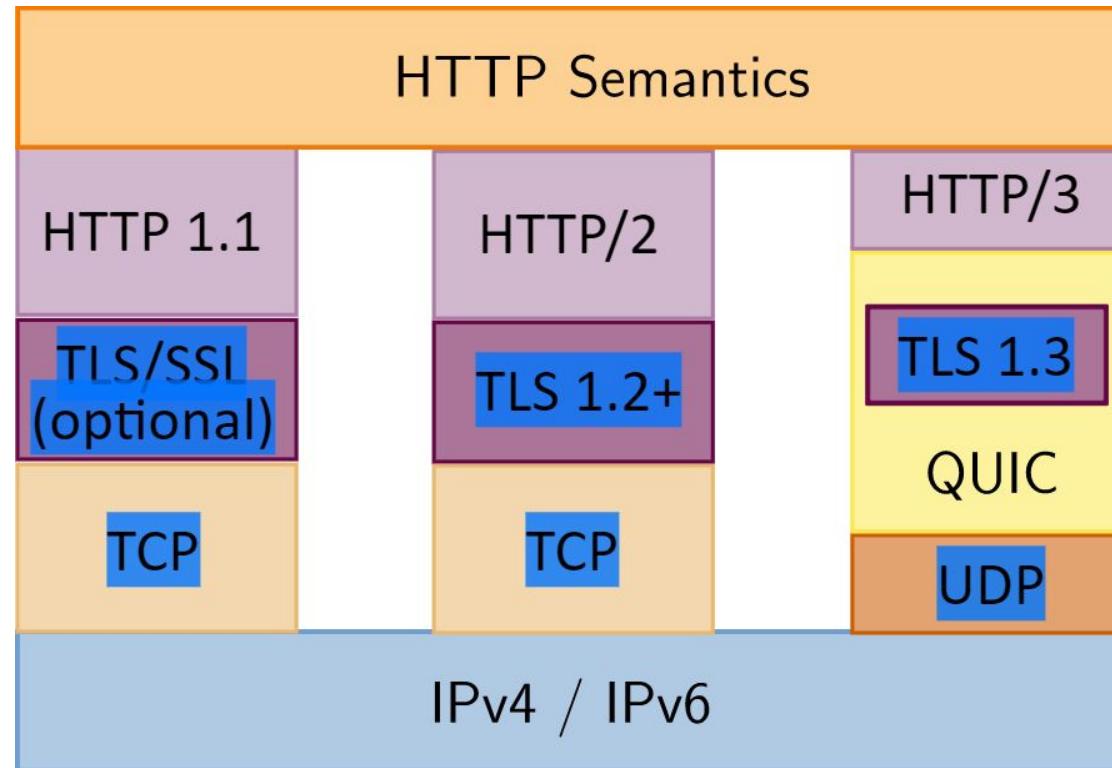


QUIC UDP



APPLICATION PROTOCOLS

HTTP



Page Speed Insights

- HTTP/1 - [Craigslist](#) - [Page Speed Insights](#)
- HTTP/2 - [Twitter](#) - [Page Speed Insights](#)
- HTTP/3 - [Google](#) - [Page Speed Insights](#)

ENCODING AND ENCRYPTING

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
    <calories>650</calories>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>Light Belgian waffles covered with strawberries and whipped cream</description>
    <calories>900</calories>
  </food>
  <food>
    <name>Berry-Berry Belgian Waffles</name>
    <price>$8.95</price>
    <description>Light Belgian waffles covered with an assortment of fresh berries and whipped cream</description>
    <calories>900</calories>
  </food>
  <food>
    <name>French Toast</name>
    <price>$4.50</price>
    <description>Thick slices made from our homemade sourdough bread</description>
    <calories>600</calories>
  </food>
  <food>
    <name>Homestyle Breakfast</name>
    <price>$6.95</price>
    <description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns</description>
    <calories>950</calories>
  </food>
</breakfast_menu>
```

XML

James Webb Space Telescope XML Database: From the Beginning to Today¹

Jonathan Gal-Edd
NASA/ Goddard Space Flight Center
Greenbelt, MD 20771
301-286-2378
jonathan.s.gal_edd@nasa.gov

Curtis C. Fatig
SAIC
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
curtis.fatig@gsfc.nasa.gov

Abstract—The James Webb Space Telescope (JWST) Project has been defining, developing, and exercising the use of a common eXtensible Markup Language (XML) for the command and telemetry (C&T) database structure. JWST is the first large NASA space mission to use XML for databases. The JWST Project started developing the concepts for the C&T database in 2002. The database will need to last at least 20 years since it will be used beginning with flight software development, continuing through Observatory integration and test (I&T) and through operations. Also, a database tool kit has been provided to the 18 various flight software development laboratories located in the United States, Europe, and Canada that allows the local users to create their own databases.

TABLE OF CONTENTS

- 1. INTRODUCTION**
- 2. JWST DATABASE HISTORY**
- 3. XML THE EARLY YEARS**
- 4. JWST XML GROWTH**
- 5. JWST XML TODAY**
- 6. JWST XML COMPATIBILITY**
- 7. REFERENCES**
- 8. BIOGRAPHIES**

JSON

```
{  
    "firstName": "John",  
    "lastName": "Smith",  
    "isAlive": true,  
    "age": 27,  
    "address": {  
        "streetAddress": "21 2nd Street",  
        "city": "New York",  
        "state": "NY",  
        "postalCode": "10021-3100"  
    },  
    "phoneNumbers": [{  
        "type": "home",  
        "number": "212 555-1234"  
    },  
    {  
        "type": "office",  
        "number": "646 555-4567"  
    }]  
},  
    "children": [  
        "Catherine",  
        "Thomas",  
        "Trevor"  
    ],  
    "spouse": null  
}
```

Protocol Buffers - Schema

```
message Person {
    string name = 1;
    int32 id = 2; // Unique ID number for this person.
    string email = 3;

    enum PhoneType {
        MOBILE = 0;
        HOME = 1;
        WORK = 2;
    }

    message PhoneNumber {
        string number = 1;
        PhoneType type = 2;
    }

    repeated PhoneNumber phones = 4;

    google.protobuf.Timestamp last_updated = 5;
}

// Our address book file is just one of these.
message AddressBook {
    repeated Person people = 1;
}
```

Protocol Buffers - Marshaling

```
protoc -I=$SRC_DIR --go_out=$DST_DIR $SRC_DIR/addressbook.proto
```

```
book := &pb.AddressBook{}
// ...

// Write the new address book back to disk.
out, err := proto.Marshal(book)
if err != nil {
    log.Fatalln("Failed to encode address book:", err)
}
if err := ioutil.WriteFile(fname, out, 0644); err != nil {
    log.Fatalln("Failed to write address book:", err)
}
```

Protocol Buffers - Unmarshaling

```
// Read the existing address book.  
in, err := ioutil.ReadFile(fname)  
if err != nil {  
    log.Fatalln("Error reading file:", err)  
}  
book := &pb.AddressBook{}  
if err := proto.Unmarshal(in, book); err != nil {  
    log.Fatalln("Failed to parse address book:", err)  
}
```

Apache Thrift - Service

```
/**  
 * Services just need a name and can optionally inherit from another service.  
 */  
service Calculator extends shared.SharedService {  
  
    void ping(),  
  
    i32 add(1:i32 num1, 2:i32 num2),  
    i32 calculate(1:i32 logid, 2:Work w) throws (1:InvalidOperationException ouch),  
  
    /**  
     * This method has a oneway modifier. The client only makes a request and  
     * does not listen for any response at all. Oneway methods must be void.  
     */  
    oneway void zip()  
  
}
```

Apache Thrift - Client

```
def main():
    # Make socket
    transport = TSocket.TSocket('localhost', 9090)

    # Buffering is critical. Raw sockets are very slow
    transport = TTransport.TBufferedTransport(transport)

    # Wrap in a protocol
    protocol = TBinaryProtocol.TBinaryProtocol(transport)

    # Create a client to use the protocol encoder
    client = Calculator.Client(protocol)

    # Connect!
    transport.open()

    client.ping()
    print('ping()')

    sum_ = client.add(1, 1)
```

Apache Avro - JSON Schema

```
{  
    "namespace": "example.avro",  
    "type": "record",  
    "name": "User",  
    "fields": [  
        {"name": "name", "type": "string"},  
        {"name": "favorite_number", "type": ["null", "int"]},  
        {"name": "favorite_color", "type": ["null", "string"]}  
    ]  
}
```

Apache Avro - Writing

```
import avro.schema
from avro.datafile import DataFileReader, DataFileWriter
from avro.io import DatumReader, DatumWriter

schema = avro.schema.parse(open("user.avsc", "rb").read())

writer = DataFileWriter(open("users.avro", "wb"), DatumWriter(), schema)
writer.append({"name": "Alyssa", "favorite_number": 256})
writer.append({"name": "Ben", "favorite_number": 8, "favorite_color": "red"})
writer.close()
```

Apache Avro - Data

```
$ od -v -t x1z users.avro
0000000 4f 62 6a 01 04 14 61 76 72 6f 2e 63 6f 64 65 63 >obj...avro.codec<
0000020 08 6e 75 6c 6c 16 61 76 72 6f 2e 73 63 68 65 6d >.null.avro.schem<
0000040 61 ba 03 7b 22 74 79 70 65 22 3a 20 22 72 65 63 >a..{"type": "rec<
0000060 6f 72 64 22 2c 20 22 6e 61 6d 65 22 3a 20 22 55 >ord", "name": "U<
0000100 73 65 72 22 2c 20 22 6e 61 6d 65 73 70 61 63 65 >ser", "namespace<
0000120 22 3a 20 22 65 78 61 6d 70 6c 65 2e 61 76 72 6f >": "example.avro<
0000140 22 2c 20 22 66 69 65 6c 64 73 22 3a 20 5b 7b 22 >", "fields": [{"<
0000160 74 79 70 65 22 3a 20 22 73 74 72 69 6e 67 22 2c >type": "string", <
0000200 20 22 6e 61 6d 65 22 3a 20 22 6e 61 6d 65 22 7d > "name": "name"}<
0000220 2c 20 7b 22 74 79 70 65 22 3a 20 5b 22 69 6e 74 >, {"type": ["int<
0000240 22 2c 20 22 6e 75 6c 6c 22 5d 2c 20 22 6e 61 6d >, "null"], "nam<
0000260 65 22 3a 20 22 66 61 76 6f 72 69 74 65 5f 6e 75 >e": "favorite_nu<
0000300 6d 62 65 72 22 7d 2c 20 7b 22 74 79 70 65 22 3a >mber"}, {"type": <
0000320 20 5b 22 73 74 72 69 6e 67 22 2c 20 22 6e 75 6c > ["string", "nul<
0000340 6c 22 5d 2c 20 22 6e 61 6d 65 22 3a 20 22 66 61 >l"], "name": "fa<
0000360 76 6f 72 69 74 65 5f 63 6f 6c 6f 72 22 7d 5d 7d >vorite_color"}]}<
0000400 00 05 f9 a3 80 98 47 54 62 bf 68 95 a2 ab 42 ef >.....GTb.h...B.<
0000420 24 04 2c 0c 41 6c 79 73 73 61 00 80 04 02 06 42 >$.,.Alyssa.....B<
0000440 65 6e 00 10 00 06 72 65 64 05 f9 a3 80 98 47 54 >en....red.....GT<
0000460 62 bf 68 95 a2 ab 42 ef 24 >b.h...B.$<
0000471
```

Apache Avro - Read

```
reader = DataFileReader(open("users.avro", "rb"), DatumReader())
for user in reader:
    print user
reader.close()
```

```
{u'favorite_color': None, u'favorite_number': 256, u'name': u'Alyssa'}
{u'favorite_color': u'red', u'favorite_number': 8, u'name': u'Ben'}
```

When to use what?

Text based encoding when:

- Data must be human readable
- You are at the scale of MiB or approaching a GiB limit
- Data is directly consumed by a web browser
- You aren't prepared to tie the data model to a schema
- To make debugging easier for developers

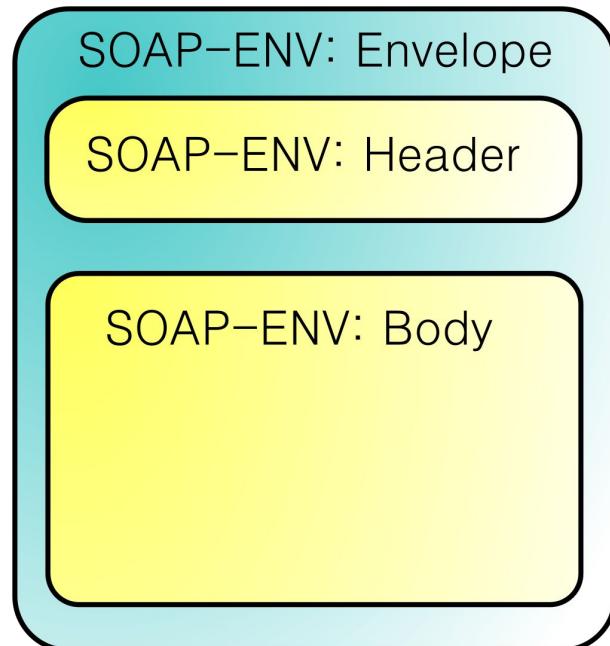
Binary based encoding when:

- You are handling GiB and beyond data
- Latency of moving data back and forth between the systems is a huge cost
- You need efficient service to service communication

API STYLES

Simple Object Access Protocol

SOAP - Message



SOAP - Message

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:m="http://www.example.org">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>T</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Pros and Cons

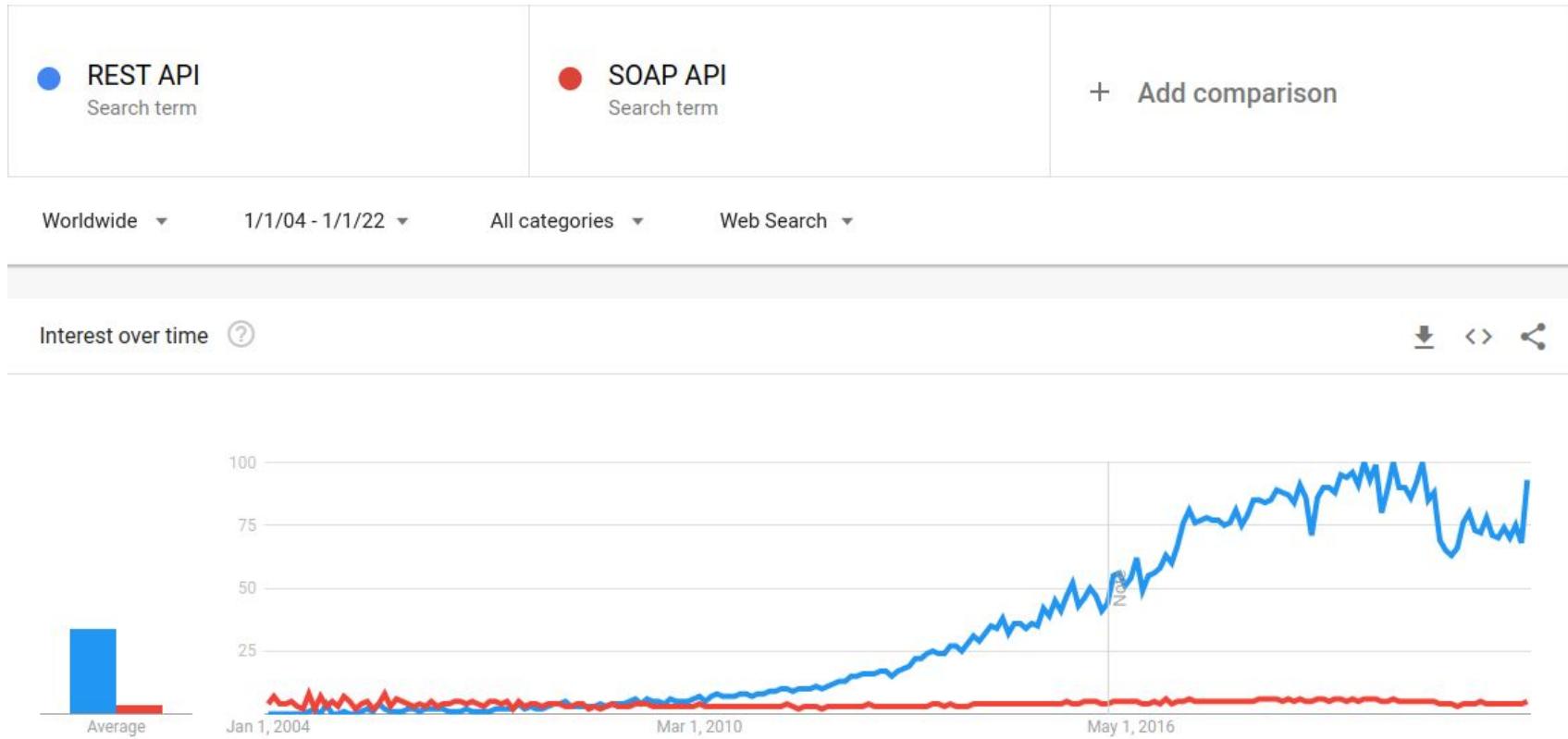
Advantages

- SOAP's neutrality characteristic explicitly makes it suitable for use with any transport protocol. HTTP, SMTP
- SOAP tunnels easily through firewalls and proxies - when used with HTTP protocol

Disadvantages

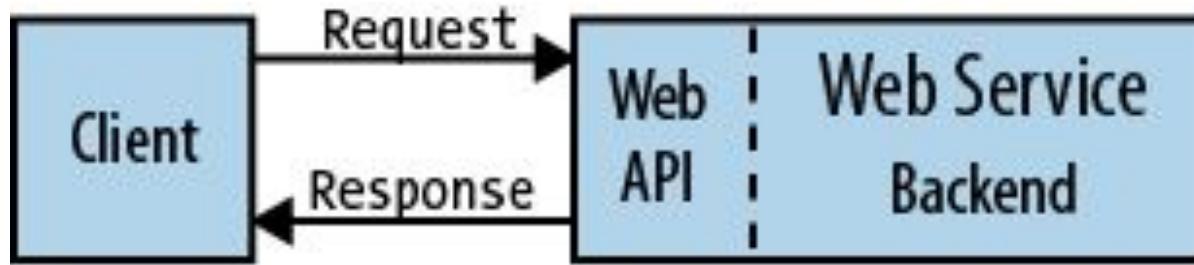
- The verbosity of the protocol
- Slow parsing speed of XML messages
- Lack of a standardized interaction model
- Being protocol-agnostic, SOAP is unable to take advantage of protocol-specific features and optimizations
- When relying on HTTP as a transport protocol and not using Web Services Addressing, the roles of the interacting parties are fixed. That means that only one party (the client) can use the services of the other

SOAP - Message



REpresentational State Transfer

REST - Architecture



REST

UNIVERSITY OF CALIFORNIA, IRVINE

Architectural Styles and the Design of Network-based Software Architectures

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Roy Thomas Fielding

2000

https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Dissertation Committee:

Professor Richard N. Taylor, Chair
Professor Mark S. Ackerman
Professor David S. Rosenblum

REST - IDEMPOTENT

IDEMPOTENCE

WHEN PERFORMING AN OPERATION AGAIN GIVES THE SAME RESULT

HTTP METHOD	IDEMPOTENCE	SAFETY
GET	YES	YES
HEAD	YES	YES
PUT	YES	NO
DELETE	YES	NO
POST	NO	NO
PATCH	NO	NO

REST - Swagger - Open API

The screenshot shows the SWAGGERhub interface for a Petstore API. The top navigation bar includes tabs for Editor, Split, and UI, along with status indicators for PRIVATE, UNPUBLISHED, and OAS3. The main content area displays the API endpoint <https://virtserver.swaggerhub.com/TradeshowDemos/SamplePetstoreAPI/1.0.0>. A search bar is present above the operation list. The operations listed under the `pet` resource are:

- POST /pet** Add a new pet to the store
- PUT /pet** Update an existing pet
- GET /pet/findByStatus** Finds Pets by status
- GET /pet/findByTags** Finds Pets by tags
- GET /pet/{petId}** Find pet by ID
- POST /pet/{petId}** Updates a pet in the store with form data
- DELETE /pet/{petId}** Deletes a pet
- POST /pet/{petId}/uploadImage** uploads an image

A note at the bottom states "store Access to Petstore orders". On the left sidebar, there are icons for Home, Search, Test, and Documentation.

REST - Swagger - Open API

The screenshot shows the Swagger UI interface for a REST API. At the top left, there is a green button labeled "POST". To its right, the endpoint "/pet" is shown, followed by the description "Add a new pet to the store". On the far right of the header is a lock icon.

Below the header, there is a "Parameters" section with a "Cancel" button. The table has two columns: "Name" and "Description".

Name	Description
body * required	Pet object that needs to be added to the store (body)

Under the "body" parameter, there is a "Example Value Model" section containing a JSON object:

```
{  
  "id": 193844,  
  "category": {  
    "id": 0,  
    "name": "string"  
  },  
  "name": "Bentley",  
  "photoUrls": [  
    "string"  
  ],  
  "tags": [  
    {  
      "id": 0,  
      "name": "string"  
    }  
  ]  
}
```

At the bottom of the "body" section is another "Cancel" button.

Below the "body" section, there is a "Parameter content type" dropdown menu set to "application/json".

At the very bottom of the interface is a large blue "Execute" button with a white cursor icon pointing to it.

SOAP vs REST

#	SOAP	REST
1	A XML-based message protocol	An architectural style protocol
2	Uses WSDL for communication between consumer and provider	Uses XML or JSON to send and receive data
3	Invokes services by calling RPC method	Simply calls services via URL path
4	Does not return human readable result	Result is readable which is just plain XML or JSON
5	Transfer is over HTTP. Also uses other protocols such as SMTP, FTP, etc.	Transfer is over HTTP only
6	JavaScript can call SOAP, but it is difficult to implement	Easy to call from JavaScript
7	Performance is not great compared to REST	Performance is much better compared to SOAP - less CPU intensive, leaner code etc.

REST - Advantages

- Simplicity
- REST is almost synonymous with HTTP
- Data Format
- RESTful API are stateless
- Caching
- No need for much bandwidth
- REST API uses entity-oriented models
- Debugging
- Ecosystem

REST - Disadvantage

- Security
- Data exposure
- Lack of state

When to use REST?

- You are interested in decoupled systems which are more adaptable to change.
- The core concern that you are facing is integration between many systems.
- If your API will be consumed by many external systems that you have no control over.
- You need data to be retrieved in many different formats.
- You expect a huge influx of requests.
- Long lived projects.

GraphQL

GraphQL - Query

```
{  
  hero {  
    name  
    height  
  }  
}
```

```
{  
  "hero": {  
    "name": "Luke Skywalker",  
    "height": 1.72  
  }  
}
```

GraphQL - Types

```
{  
  hero {  
    name  
    friends {  
      name  
      homeWorld {  
        name  
        climate  
      }  
      species {  
        name  
        lifespan  
        origin {  
          name  
        }  
      }  
    }  
  }  
}
```

```
type Query {  
  hero: Character  
}  
  
type Character {  
  name: String  
  friends: [Character]  
  homeWorld: Planet  
  species: Species  
}  
  
type Planet {  
  name: String  
  climate: String  
}  
  
type Species {  
  name: String  
  lifespan: Int  
  origin: Planet  
}
```

GraphQL - Docs

The screenshot shows the GraphQL Docs interface with the following components:

- GraphQL**: The title of the tab.
- Play**: A button to execute the query.
- FilmsConnection**: The current connection being viewed.
- Film**: A link to the Film type definition.
- X**: A close button.
- Query Area (Left):** Contains the GraphQL query:

```
1 {  
2   allFilms {  
3     films {  
4       title  
5       openingCrawl  
6     }  
7   }  
8 }
```
- Results Area (Center):** Shows the JSON response from the query:

```
{  
  "data": {  
    "allFilms": {  
      "films": [  
        {  
          "title": "A New Hope",  
          "openingCrawl": "It is a period of  
            peace in the galaxy.",  
        },  
        {  
          "title": "The Empire Strikes Back",  
          "openingCrawl": "It is a dark time  
            in the galaxy.",  
        },  
        {  
          "title": "Return of the Jedi",  
          "openingCrawl": "Luke Skywalker has  
            discovered his true strength.",  
        },  
        {  
          "title": "The Phantom Menace",  
          "openingCrawl": "Turmoil has engulfed  
            the galaxy.",  
        },  
        {  
          "title": "Attack of the Clones",  
        }  
      ]  
    }  
  }  
}
```
- Description (Right):** A single film.
- IMPLEMENTATION (Right):** Node
- FIELDS (Right):**
 - title: String
 - episodeID: Int
 - openingCrawl: String
 - director: String
 - producers: [String]
 - releaseDate: String

GraphQL - Advantages

- Each client retrieves exactly the data it needs. This can be especially beneficial in low-bandwidth environments
- JSON objects can be retrieved and used as is, without any manipulation on the client side
- clients are immune to how the APIs are defined
- GraphQL pairs perfectly with React

GraphQL - Disadvantages

- **Expensive Queries** - GraphQL allows you to query and mutate large amounts of data. A big query could very quickly bring down an entire GraphQL server.
- Ability to traverse a graph can lead to **deep recursions** that could burn valuable resources.
- [GraphQL Rate Limiting](#) - In REST API, you can specify that we allow only this amount of requests in one day", This is difficult to specify in GraphQL
- GraphQL Caching - Caching in GraphQL is very complex because each query can be different, even though it operates on the same entity.
- Files upload/download can be tricky forcing you to implement dedicated REST endpoints.
- The fact that you can allow/disallow fields makes Relational Database joins non-trivial.

GraphQL - Big Queries

```
query lotsOfData {  
    allProducts  
    allSKUs  
    allCategories  
    allCustomers  
    allOrders  
}
```

GraphQL - Recursive Traverse

```
query somethingMalicious {
    allProducts {
        category {
            products {
                category {
                    products {
                        category
                    }
                }
            }
        }
    }
}
```

When to use GraphQL?

- When application bandwidth matters - mobile phones, smartwatches, IoT
- Composite pattern, where application retrieves data from multiple, different APIs
- In short lived project.

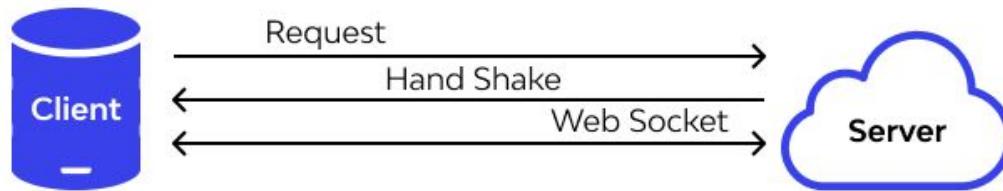
GitHub - REST and GraphQL

- <https://docs.github.com/en/rest/repos/repos#list-public-repositories>
- <https://docs.github.com/en/graphql/overview/explorer>

WebSocket

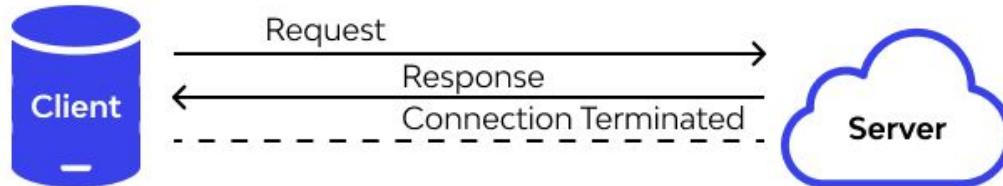
WebSocket - Connection

WebSocket Connection

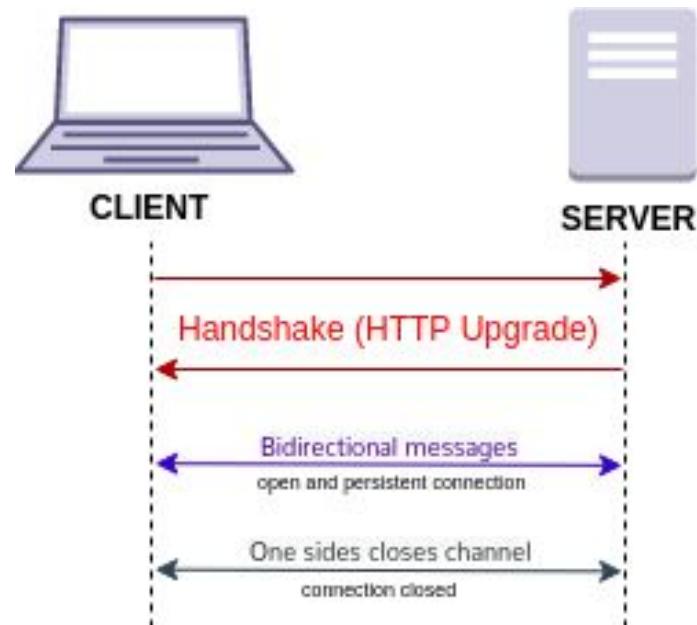


VS

HTTP Connection



WebSocket - Upgrade



WebSocket - Protocol Switch

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Server response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm50PpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

WebSocket - Events

```
// Creates new WebSocket object with a wss URI as the parameter
const socket = new WebSocket('wss://game.example.com/ws/updates');

// Fired when a connection with a WebSocket is opened
socket.onopen = function () {
    setInterval(function() {
        if (socket.bufferedAmount == 0)
            socket.send(getUpdateData());
    }, 50);
};

// Fired when data is received through a WebSocket
socket.onmessage = function(event) {
    handleUpdateData(event.data);
};

// Fired when a connection with a WebSocket is closed
socket.onclose = function(event) {
    onSocketClose(event);
};

// Fired when a connection with a WebSocket has been closed because of an error
socket.onerror = function(event) {
    onSocketError(event);
};
```

WebSocket - Pros and Cons

Advantages

- Real time full-duplex communication
- Platforms agnostic web, desktop, mobile
- WebSocket overhead has only 2-byte compared to HTTP 2000 bytes
- WebSockets data frames can transfer text or binary data

Disadvantages

- A fully HTML5-compliant web browser is required
- Websockets do not provide intermediary caching
- Lost connection require reconnecting mechanisms
- Communication between origins - at the costs of security risks

When to use WebSockets?

- full-duplex communication
- real time constraints

<https://miro.com/app/board/uXjVPZmxNVQ=/>

RPC / gRPC

gRPC - .proto

```
// The greeting service definition.
service Greeter {
    // Sends a greeting
    rpc SayHello (HelloRequest) returns (HelloReply) {}
    // Sends another greeting
    rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
    string name = 1;
}

// The response message containing the greetings
message HelloReply {
    string message = 1;
}
```

gRPC - code generation

```
python -m grpc_tools.protoc \  
-I../../protos \  
--python_out=. \  
--pyi_out=. \  
--grpc_python_out=. \  
../../protos/helloworld.proto
```

- generated **request** and **response** classes
- generated **client** and **server** classes

gRPC - greeter server

```
class Greeter(helloworld_pb2_grpc.GreeterServicer):

    def SayHello(self, request, context):
        return helloworld_pb2.HelloReply(message='Hello, %s!' % request.name)

    def SayHelloAgain(self, request, context):
        return helloworld_pb2.HelloReply(message='Hello again, %s!' % request.name)
```

gRPC - greeter client

```
def run():
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = helloworld_pb2_grpc.GreeterStub(channel)
        response = stub.SayHello(helloworld_pb2.HelloRequest(name='you'))
        print("Greeter client received: " + response.message)
        response = stub.SayHelloAgain(helloworld_pb2.HelloRequest(name='you'))
        print("Greeter client received: " + response.message)
```

gRPC- Pros and Cons

Advantages

- Performance
- gRPC uses HTTP/2 under the hood, but does not expose any of HTTP/2 to the API designer or API user.

Disadvantage

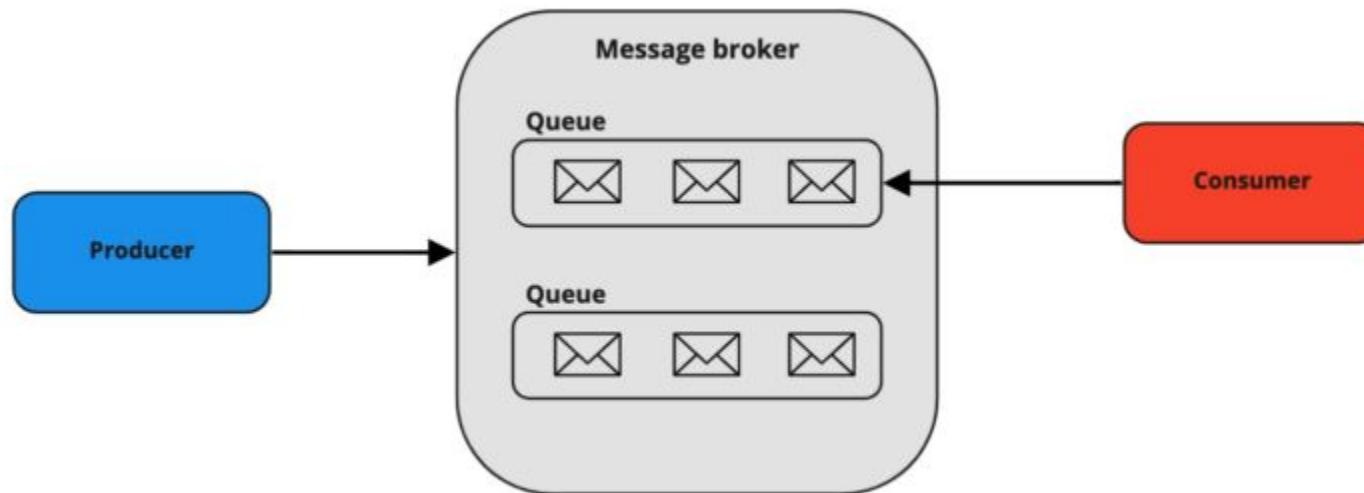
- Client and the server regeneration.
- RPC tends to bloat over time as more procedures are added.

When to use gRPC?

- When every bit matters
- When you need efficient communication between tightly coupled services that you own
- When the consumers of the API is the organization that you have control over (internal APIs)

Messaging

Messaging - Architecture

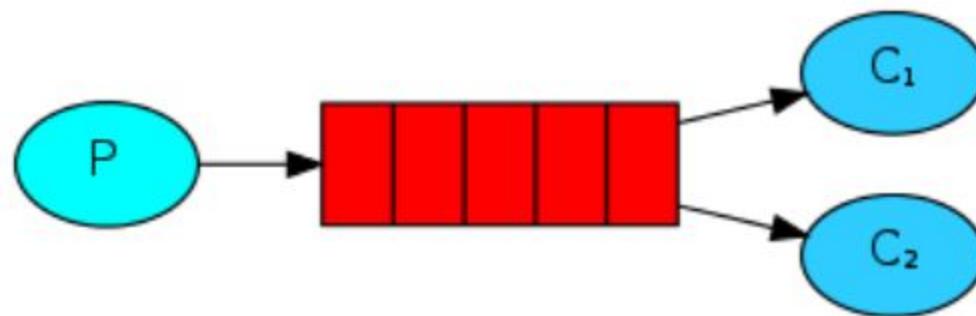


Messaging vs RPC

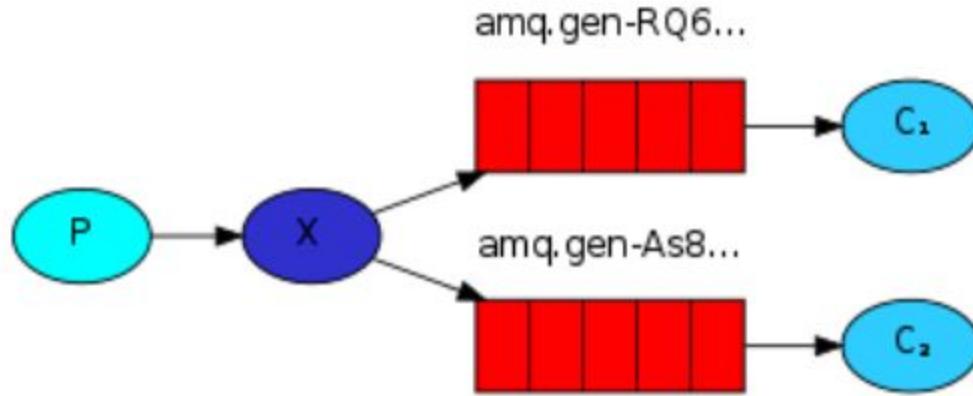
- The sender doesn't need to know the IP address and port number of the recipient.
- If the recipient is unavailable or overloaded then the broker acts as a buffer.
- It can automatically redeliver messages to a process that has crashed.
- It allows one message to be sent to several recipients.
- It logically decouples the sender from the recipient.



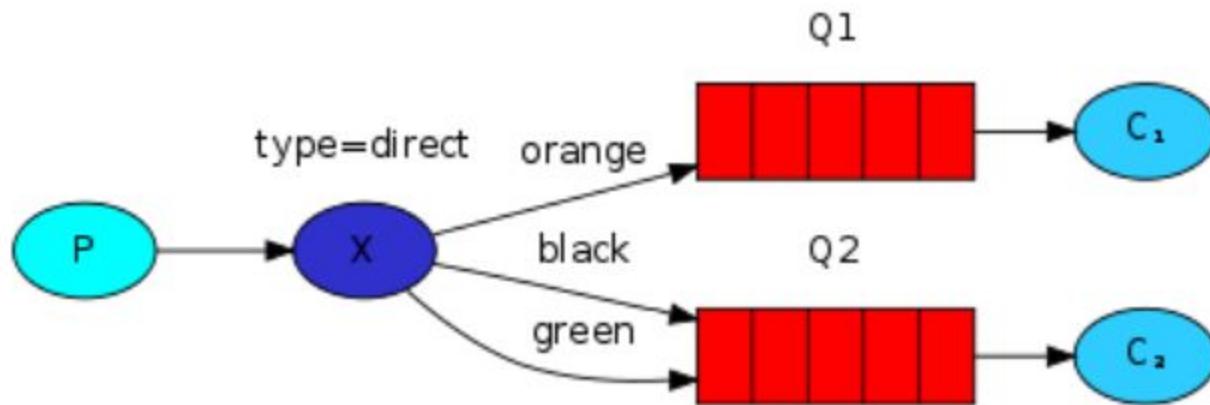
Work Queue



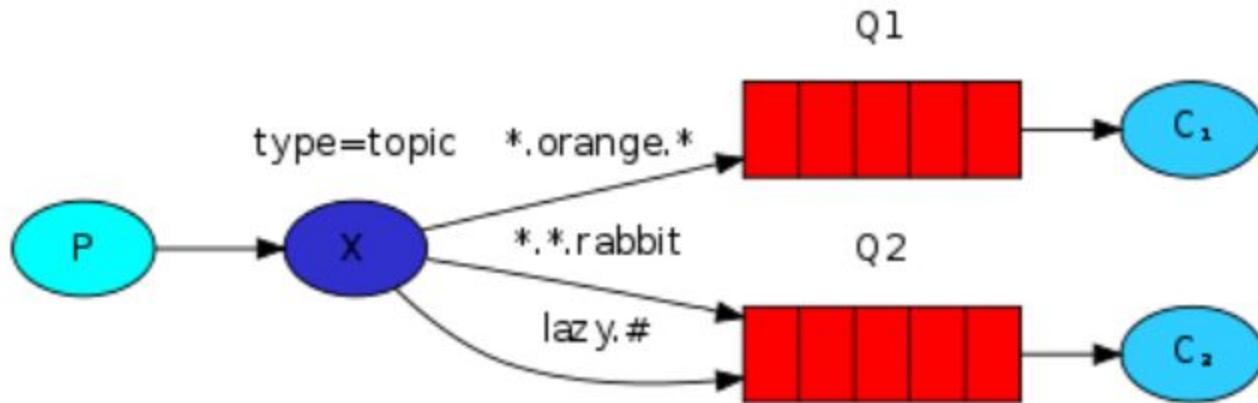
Publish / Subscribe



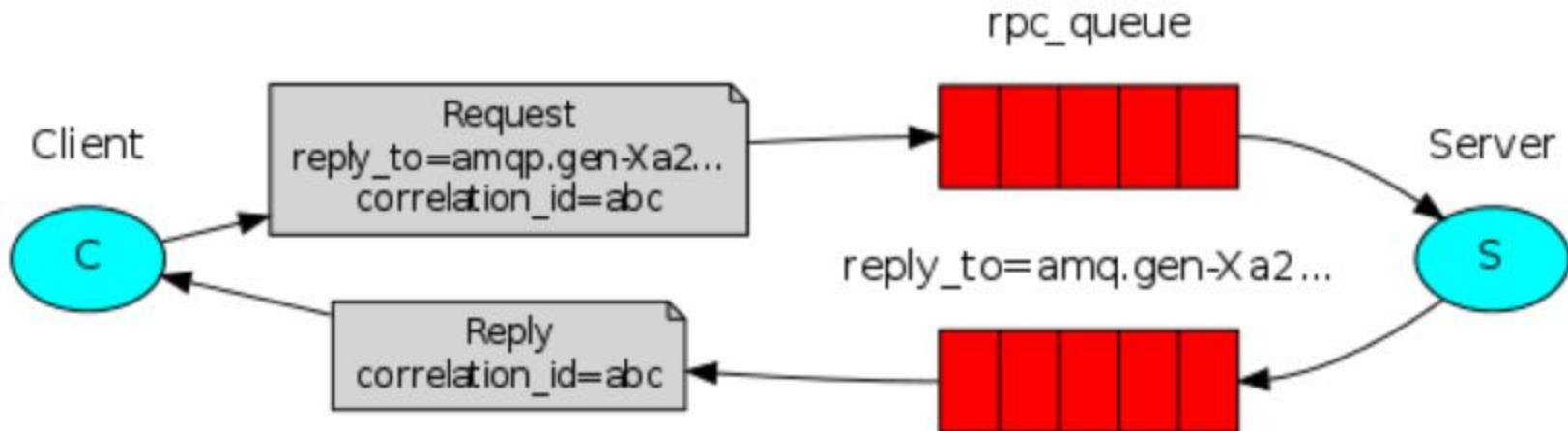
Routing



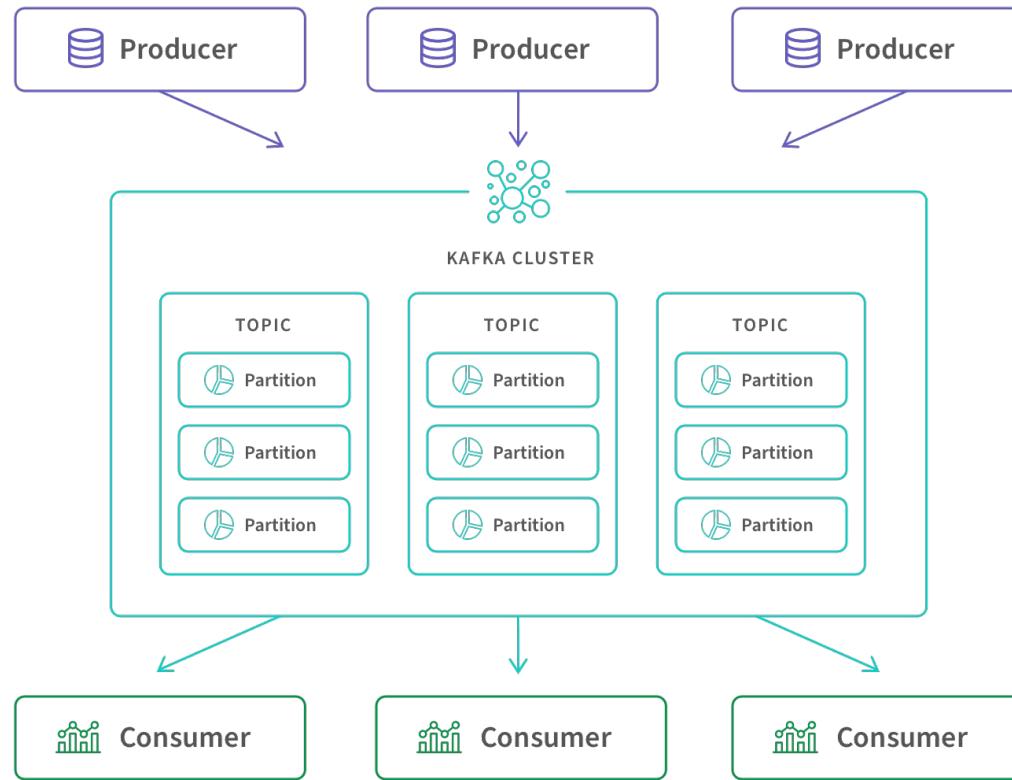
Topics



Request - Reply







Kafka vs RabbitMQ

Kafka

- Users can replay messages
- Pairs well with big data systems such as Elasticsearch
- Allows topics to be split into partitions
- Performance improves with fewer, large batches of data
- Includes four primary security features

RabbitMQ

- Allows users to set sophisticated rules for message delivery
- Supports STOMP, MQTT, WebSockets, and others
- Capability to vary point-to-point, request/reply, and publish/subscribe messaging
- Strong authentication and expression-based authorization

When to use Messaging?

- Synchronous communication is not a concern
- Near real time data feeds
- Data delivery is a concern
- You want to build decoupled systems - interservice communication

Characteristics of Good API

"APIs should be easy to use and hard to misuse. It should be easy to do simple things; possible to do complex things; and impossible, or at least difficult to do wrong things. Documentation matters. No matter how good an API is, it won't get used without good documentation." Joshua Bloch

Good API is:

- Easy to read
- Easy to learn
- Easy to maintain
- Easy to evolve
- Intuitive - easy to use even without documentation
- Hard to misuse
- Appropriate to the audience

Recap

- Historical context that lead to creation of various APIs
- The OSI model
- Transmission modes and communication styles for API
- Encoding and encryption in context of API
- Relevant protocols utilized in network based communication for API
- Understanding of common web API
- API style trade-offs
- API style is appropriate for particular set of problems
- Characteristics of good APIs