QtWS25

Dennis Oberst

Speed Up with
QtGrpc & QtProtobuf

Speed Up Your Remote Communication with QtGrpc & QtProtobuf

Introduction

*"... are Google's language-neutral, platform-neutral, extensible mechanism for **serializing structured data**."*

*"... is a modern, open source, high-performance **remote procedure call** (RPC) **framework** that can run anywhere."*

Protobuf

Language

```
// event.proto

syntax = "proto3";

message Event {
    enum Type {
        UNKNOWN = 0;
        USER = 1;
        SYSTEM = 2;
    }
    Type type = 1;
    string name = 2;
}
```

Protocol Buffer

Compiler

```
protoc
  –java_out
  –python_out
  –cpp_out
  –rust_out
  ...
```

Code Generation

Library

```
#include "event.pb.h"

...

int main(int argc, char *argv[])
{
    Event e1;
    e1.set_type(Event::USER);
    e1.set_name("User.json");

    Event e2;
    e2.set_type(Event::SYSTEM);
    e2.set_name("Sys.proto");

    printSerialized("JSON : "
        + messageToJsonString(e1));
    printSerialized("PROTO: "
        + e2.SerializeAsString());
}
```

Serialization

Protobuf

```
protoc –cpp_out=. event.proto
g++ main.cpp event.pb.cc –o event_proto \
  $(pkg-config --libs protobuf)
```

```
./event_proto
JSON : {"type":"USER","name":"User.json"}
PROTO: \x08\x02\x12\x09Sys.proto
```

```
Bytes used
34
13
```

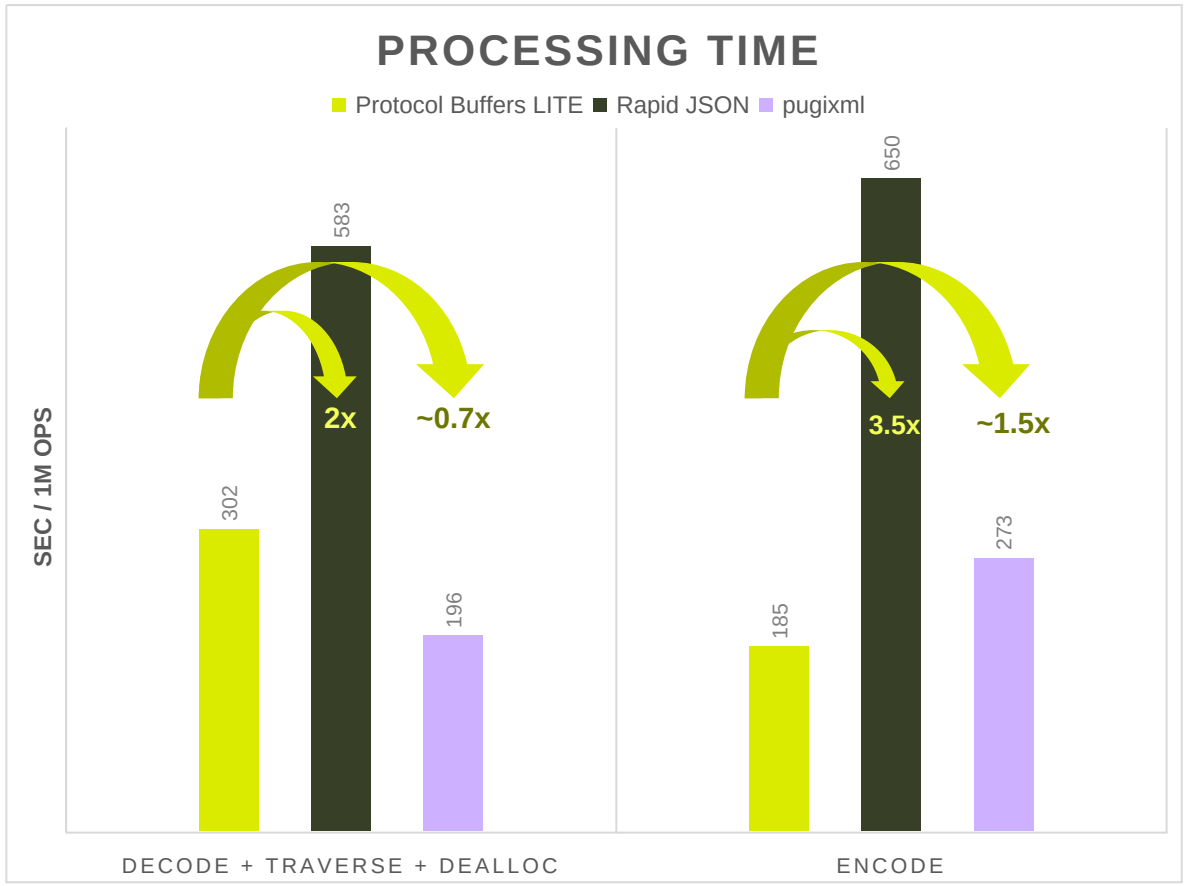
9 bytes

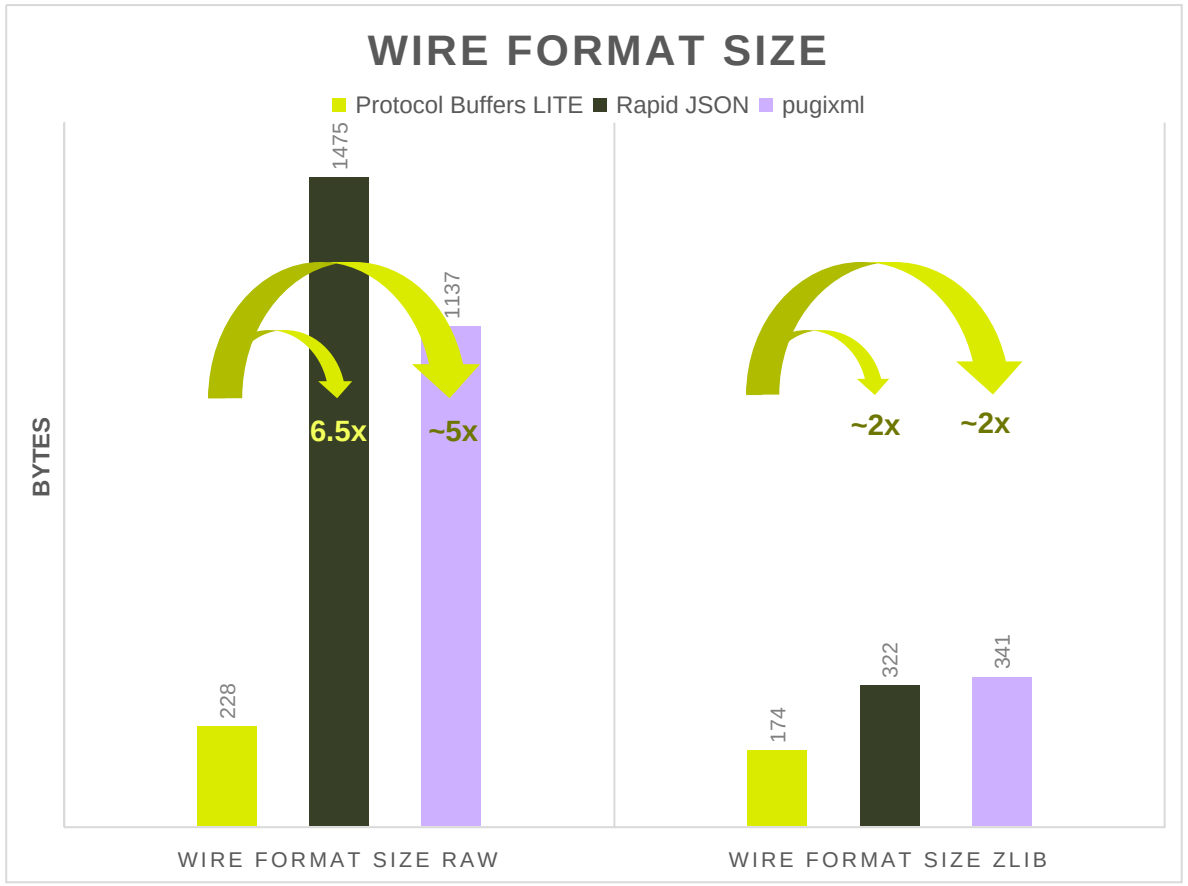
Field 2 (**name**): LEN

SYSTEM

Field 1 (**type**): VARINT

Protobuf


PROCESSING TIME
Protocol Buffers LITE
Rapid JSON
pugixml
SEC / 1M OPS
583
302
196
650
185
273
2x
~0.7x
3.5x
~1.5x
DECODE + TRAVERSE + DEALLOC
ENCODE


WIRE FORMAT SIZE
Protocol Buffers LITE
Rapid JSON
pugixml
BYTES
1475
228
1137
174
322
341
6.5x
~5x
~2x
~2x
WIRE FORMAT SIZE RAW
WIRE FORMAT SIZE ZLIB

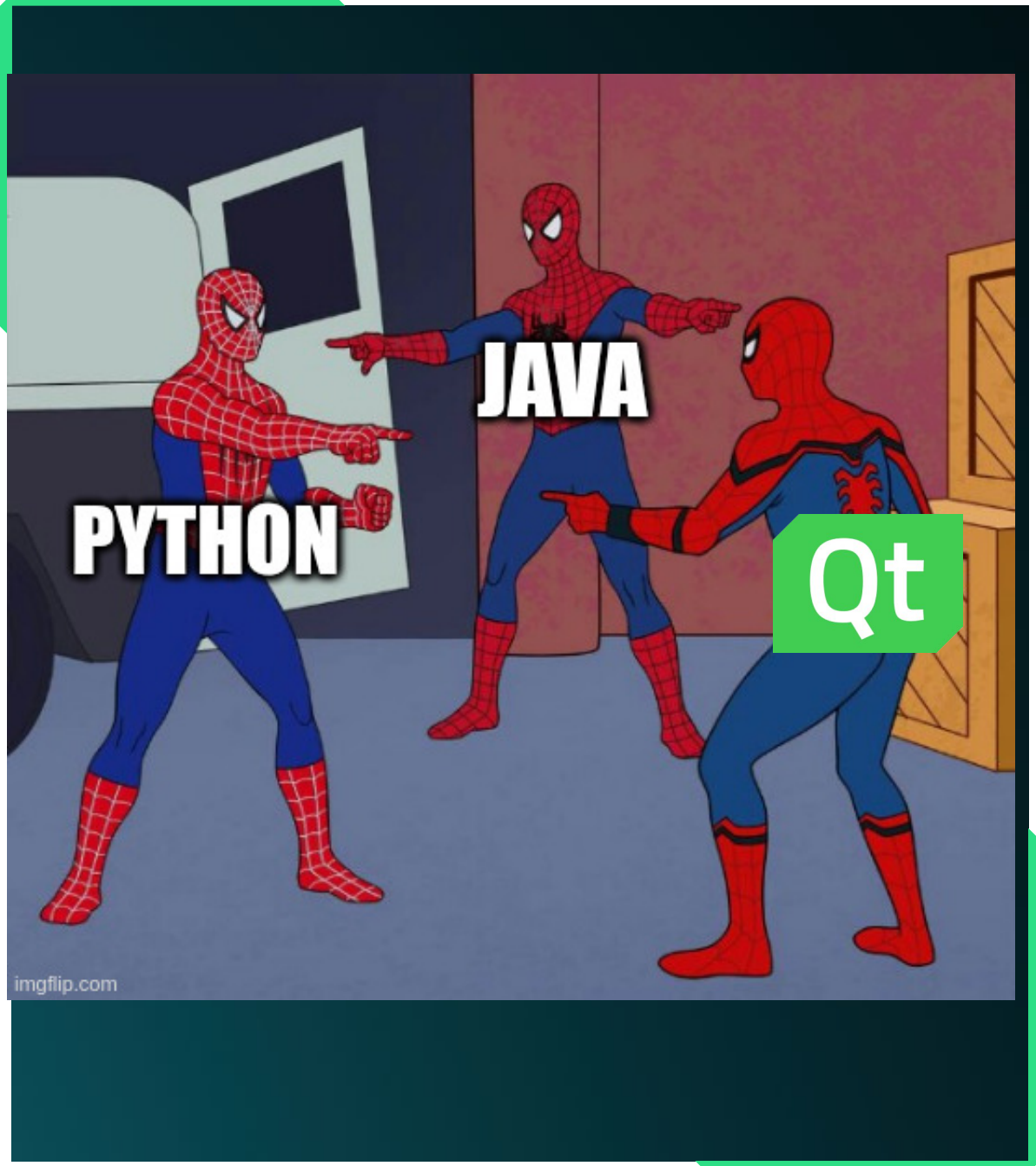
Protobuf

1. Schema-Driven API
   - Language-Independent
   - Strongly typed definitions
   - Safe evolution & versioning
2. Code Generation
   - Optimized for each target language
   - Extensible Plugin Ecosystem
3. Performance
   - Fast serialization & deserialization
   - Compact binary format

- Dependency on schema definition
- Not human-readable
- Integration complexity

Protobuf

QtProtobuf



QtProtobuf

1. Schema-Driven API
   - Language-Independent
   - Strongly typed definitions
   - Safe evolution & versioning
2. Code Generation
   - Optimized for each target language
   - Extensible Plugin Ecosystem
3. Performance
   - Fast serialization & deserialization
   - Compact binary format

- Proto Messages as native Qt C++ types
  - Meta-object system compatible
  - Easy use in models, views and signals
  - Uses familiar Qt types
- Proto Messages as native Qt QML types
- Integrated tooling & build system
- Smooth conversion of QtGui & QtCore types

Protobuf

```
#include "event.pb.h"

...

int main(int argc, char *argv[])
{
    Event e1;
    e1.set_type(Event::USER);
    e1.set_name("User.json");

    Event e2;
    e2.set_type(Event::SYSTEM);
    e2.set_name("Sys.proto");

    std::string e1Serialized;
    google::protobuf::util::MessageToJsonString(
        e1, &e1Serialized);
    printSerialized("JSON : "
        + e1Serialized);
    printSerialized("PROTO: "
        + e2.SerializeAsString());
}
```

QtProtobuf

```
#include "event.qpb.h"

...

int main(int argc, char *argv[])
{
    Event e1;
    e1.setType(Event::Type::USER);
    e1.setName("User.json");

    Event e2;
    e2.setType(Event::Type::SYSTEM);
    e2.setName("Sys.proto");

    QProtobufSerializer protoSer;
    QProtobufJsonSerializer jsonSer;

    printSerialized("JSON : "
        + e1.serialize(&jsonSer));
    printSerialized("PROTO: "
        + e2.serialize(&protoSer));
}
```

.qpb.h prefixed

camelCase

Scoped Enums

Separate Serializer

Protobuf

```
protoc –cpp_out=. event.proto



g++ main.cpp event.pb.cc –o event_proto \
  $(pkg-config --libs protobuf)
```

```
./event_proto
JSON : {"type":"USER","name":"User.json"}
PROTO: \x08\x02\x12\x09Sys.proto
```

QtProtobuf

```
protoc –qtprotobuf_out=. \
  –plugin=protoc-gen-qtprotobuf=\
  $(which qtprotobufgen) event.proto

moc ...

g++ main.cpp event.qpb.cc –o event_qtproto \
  $(pkg-config --libs Qt6Protobuf)
```

```
./event_qtproto
JSON : {"name":"User.json","type":"USER"}
PROTO: \x08\x02\x12\x09Sys.proto
```

QtProtobuf QML

```
find_package(Qt6 REQUIRED COMPONENTS
  Protobuf ProtobufQuick)
qt_standard_project_setup(REQUIRES 6.9)

qt_add_executable(event_qml main.cpp)
qt_add_qml_module(event_qml
    URI Event
    QML_FILES Main.qml
    SOURCES serializer.h
)

add_library(event_qml_proto STATIC)
qt_add_protobuf(event_qml_proto
    QML QML_URI Event.Proto
    PROTO_FILES event.proto
)

target_link_libraries(event_qml PRIVATE
    Qt6::Protobuf Qt6::ProtobufQuick
    event_qml_proto
)
```

QtProtobuf QML

serializer.h

```
#include "event.qpb.h"
...

class Serializer : public QObject
{
    Q_OBJECT
    QML_ELEMENT
    Q_PROPERTY(Type type MEMBER mType NOTIFY typeChanged)

public:
    enum Type { Proto, Json }; Q_ENUM(Type)

    Q_INVOKABLE QByteArray serialize(const Event &msg) {
        return mType == Proto
            ? msg.serialize(&mProtoSerializer)
            : msg.serialize(&mJsonSerializer);
    }

    Q_INVOKABLE QString display(const QByteArray &msg)
    { /* prettify serialized @msg */ }

signals:
    void typeChanged();

private:
    Type mType
    QProtobufSerializer mProtoSerializer;
    QProtobufJsonSerializer mJsonSerializer;
};
```

Main.qml

Generated module

QML_VALUE_TYPE

```
import Event
import Event.Proto

ApplicationWindow {
    id: root

    property event event          // from Event.Proto
    onEventChanged: serializer.update()
    property string data

    Serializer {                  // from Event
        id: serializer
        onTypeChanged: update()
        function update() {
            root.data = serialize(root.event)
        }
    }

    ComboBox {
        model: [
            { text: "UNKNOWN", value: Event.Type.UNKNOWN },
            { text: "USER",    value: Event.Type.USER    },
            { text: "SYSTEM",  value: Event.Type.SYSTEM  }
        ]
        onActivated: root.event.type = currentValue
    }
    TextField {
        text: root.event.name
        onTextEdited: root.event.name = text
     }
    // display serialized event, select serializer type ...
}
```

QtProtobuf QML

serializer.h

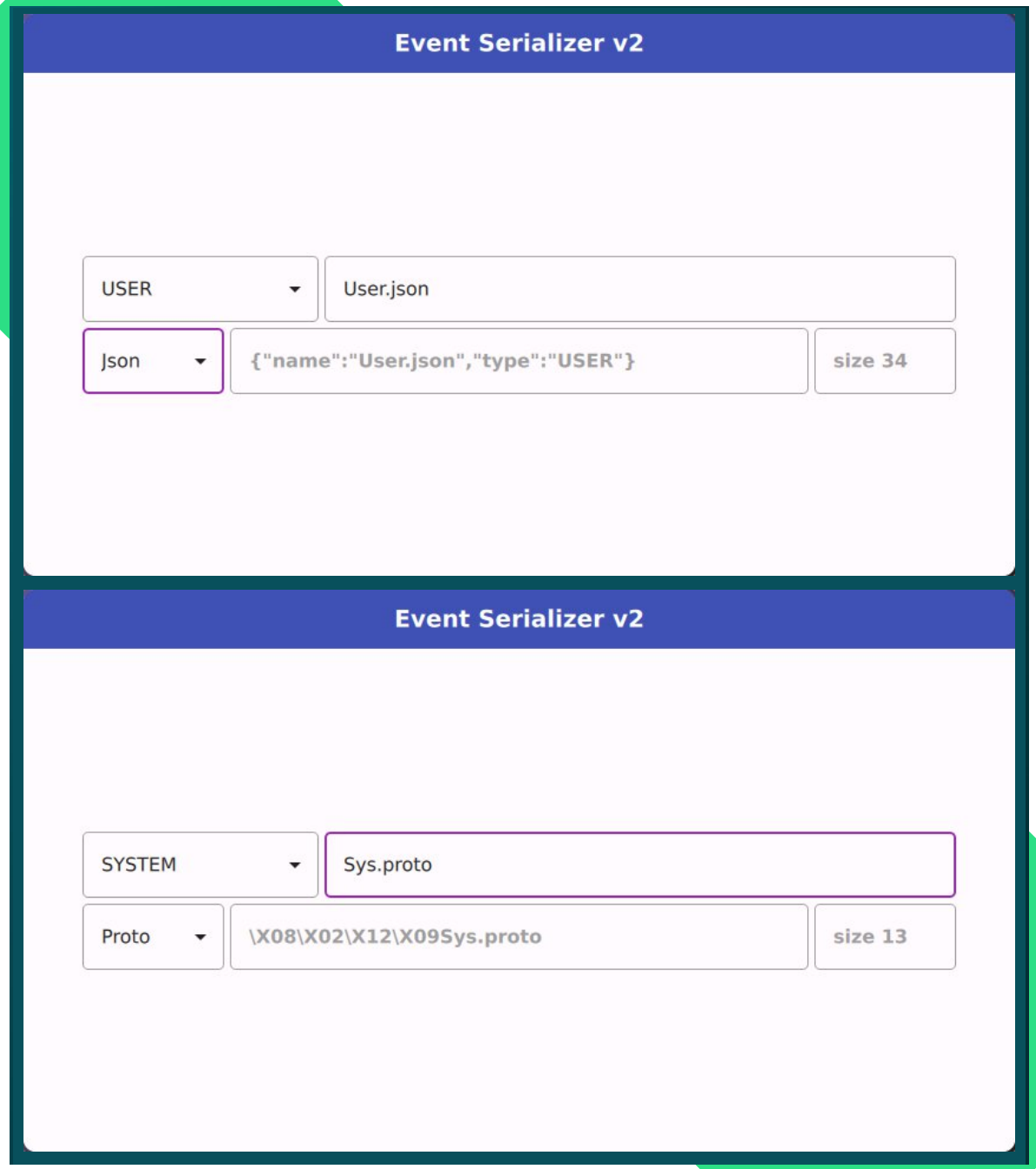
```
class Serializer : public QObject
{
    Q_OBJECT
    QML_ELEMENT
    Q_PROPERTY(Type type MEMBER mType NOTIFY typeChanged)
    Q_PROPERTY(Event event MEMBER mEvent NOTIFY eventChanged)
    Q_PROPERTY(QByteArray data READ data NOTIFY dataChanged)

public:
    enum Type { Proto, Json }; Q_ENUM(Type)

    Serializer(QObject *parent = nullptr)
        : QObject(parent)
    {
        connect(this, &Serializer::typeChanged,
                &Serializer::serialize);
        connect(this, &Serializer::eventChanged,
                &Serializer::serialize);
    }

    void serialize() {
        mData = mType == Proto
            ? mEvent.serialize(&mProtoSerializer)
            : mEvent.serialize(&mJsonSerializer);
        emit dataChanged();
    }

    Q_INVOKABLE QString display(const QByteArray &msg)
    { /* prettify serialized @msg */ }

private:
    Type mType;
    Event mEvent;
    QByteArray mData;
    QProtobufSerializer mProtoSerializer;
    QProtobufJsonSerializer mJsonSerializer;
};
```

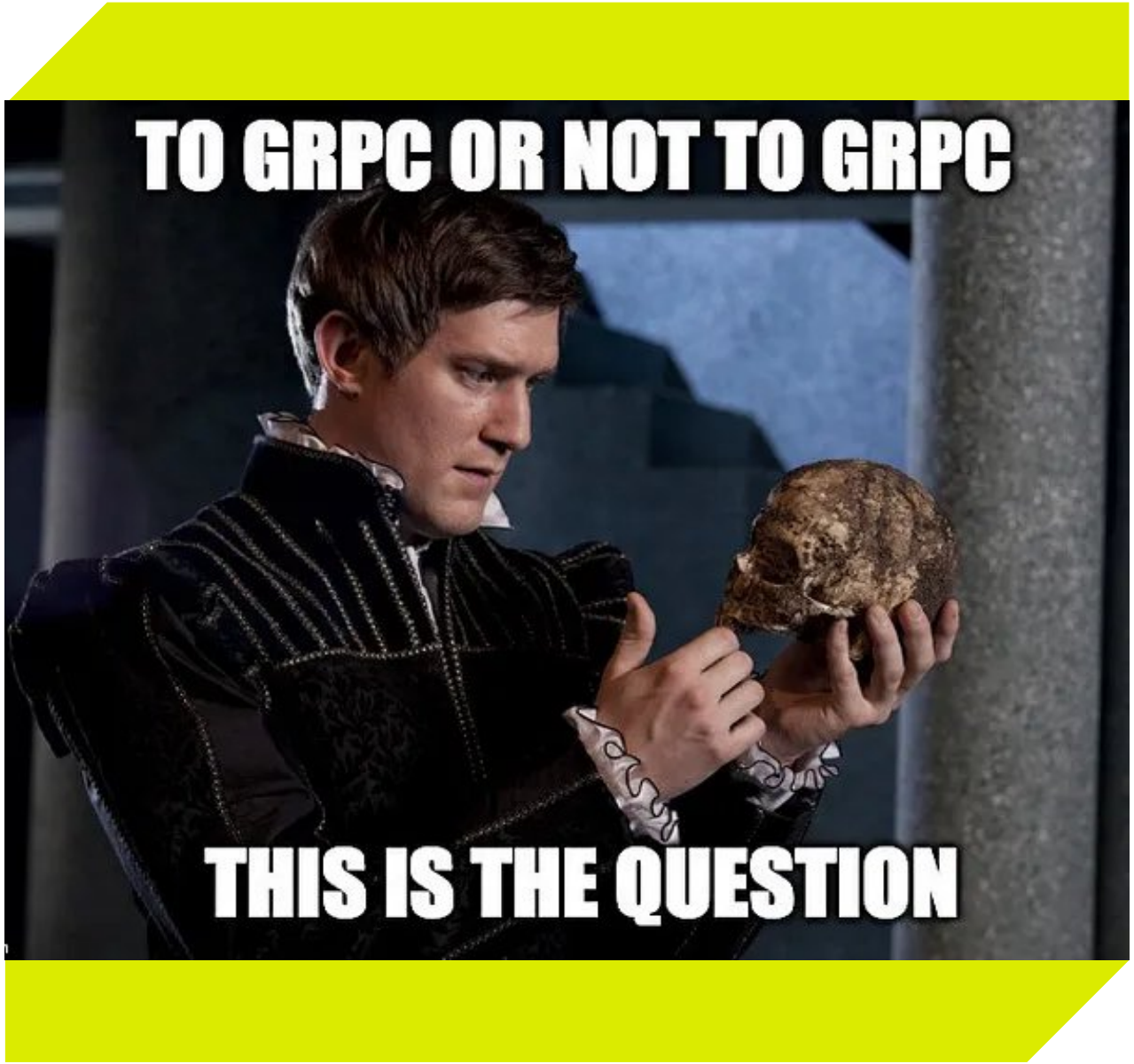
Main.qml

```
import Event
import Event.Proto

ApplicationWindow {
    id: root

    Serializer {
        id: serializer
        type: serializeSelector.currentValue
    }

    ComboBox {
        model: [
            { text: "UNKNOWN", value: Event.Type.UNKNOWN },
            { text: "USER",    value: Event.Type.USER    },
            { text: "SYSTEM",  value: Event.Type.SYSTEM  }
        ]
        onActivated: serializer.event.type = currentValue
    }
    TextField {
        text: serializer.event.name
        onTextChanged: serializer.event.name = text
    }

    ComboBox {
        id: serializeSelector
        model: [
            { text: "Proto", value: Serializer.Proto },
            { text: "Json",  value: Serializer.Json  },
        ]
    }
    TextField { text: serializer.display(serializer.data)) }
    TextField { text: "size " + String(serializer.data).length }
}
```

QtProtobuf QML

DEMO

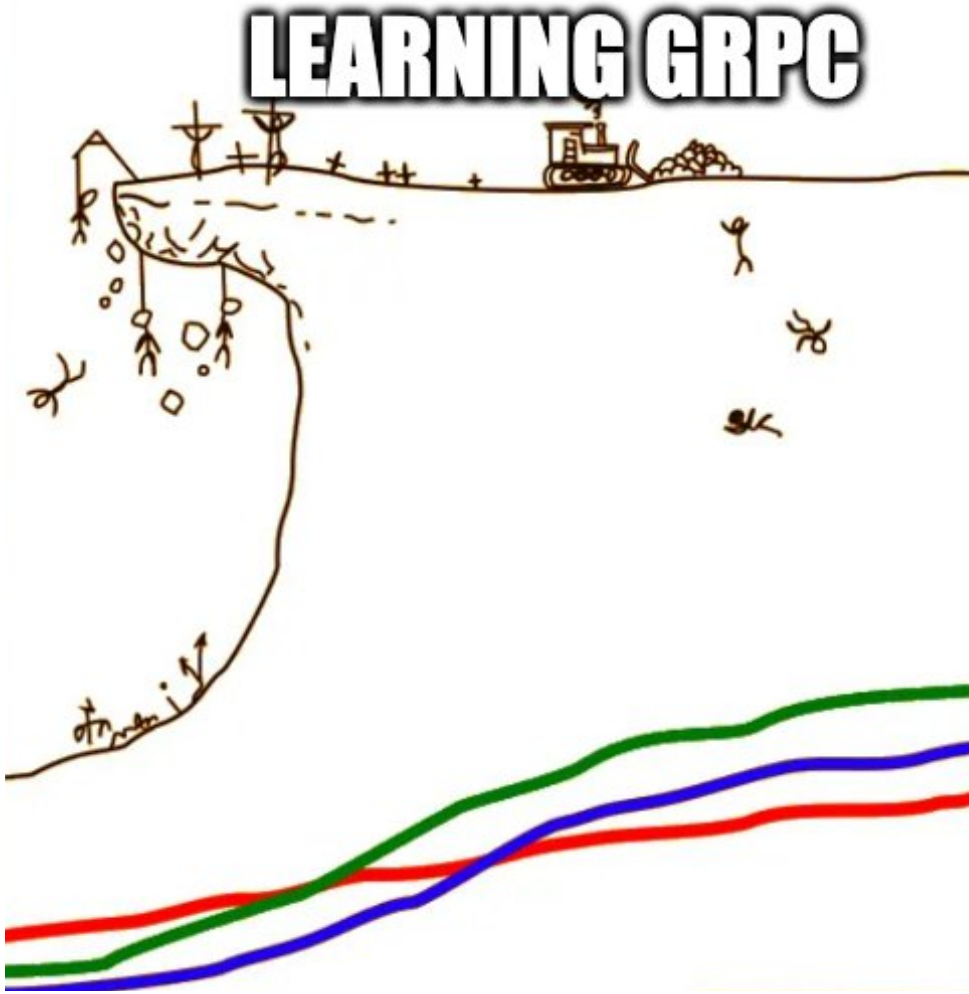



TO GRPC OR NOT TO GRPC
THIS IS THE QUESTION

gRPC

- **R**emote **P**rocedure **C**alls
  - Execute methods on remote systems as if they were local calls.
  - Abstracts network details by hiding complexities

- Where is it used?
  - Microservice Architecture
  - Real-Time & Streaming Apps
  - Cross-Platform & Multi-language Environments



gRPC

```
protoc --plugin=protoc-gen-grpc= \
$(which grpc_cpp_plugin) eventhub.proto
```

Protobuf Definitions

```
syntax = "proto3";                    // eventhub.proto

import "event.proto";

message None {}

service EventHub {
  rpc Push(Event) returns (None) {}
  rpc Subscribe(None) returns(stream Event) {}
  rpc Notify(stream Event) returns (None){}
  rpc Exchange (stream Event) (stream Event) {}
}
```

Service

Interfaces for RPCs

```
// eventhub.grpc.pb.h

class EventHub {
    static std::unique_ptr<Stub> NewStub(channel);      // “Client”
    // Stub interfaces ...

    class Service {                                     // “Server”
        virtual grpc::Status Push(...);
        virtual grpc::Status Subscribe(...);
        virtual grpc::Status Notify(...);
        virtual grpc::Status Exchange(...);
    };
};
```

Synchronous

gRPC

```
protoc --plugin=protoc-gen-grpc= \
$(which grpc_cpp_plugin) eventhub.proto
```

Protobuf Definitions

```
syntax = "proto3";                  // eventhub.proto

import "event.proto";

message None {}

service EventHub {
  rpc Push(Event) returns (None) {}
  rpc Subscribe(None) returns(stream Event) {}
  rpc Notify(stream Event) returns (None){}
  rpc Exchange (stream Event) (stream Event) {}
}
```

Service

Interfaces for RPCs

```
// eventhub.grpc.pb.h

class EventHub {
    static std::unique_ptr<Stub> NewStub(channel);      // “Client”
    // Stub interfaces ...

    class CallbackService {                             // “Server”
        virtual grpc::ServerUnaryReactor* Push(...);
        virtual grpc::ServerWriteReactor<Event>* Subscribe(...);
        virtual grpc::ServerReadReactor<Event>* Notify(...);
        virtual grpc::ServerBidiReactor<Event,Event>* Exchange(...);
    };
};
```

Asynchronous Callback

gRPC

```
protoc --plugin=protoc-gen-grpc= \
$(which grpc_cpp_plugin) eventhub.proto
```

Protobuf Definitions

```
syntax = "proto3";                  // eventhub.proto

import "event.proto";

message None {}

service EventHub {
  rpc Push(Event) returns (None) {}
  rpc Subscribe(None) returns(stream Event) {}
  rpc Notify(stream Event) returns (None){}
  rpc Exchange (stream Event) (stream Event) {}
}
```

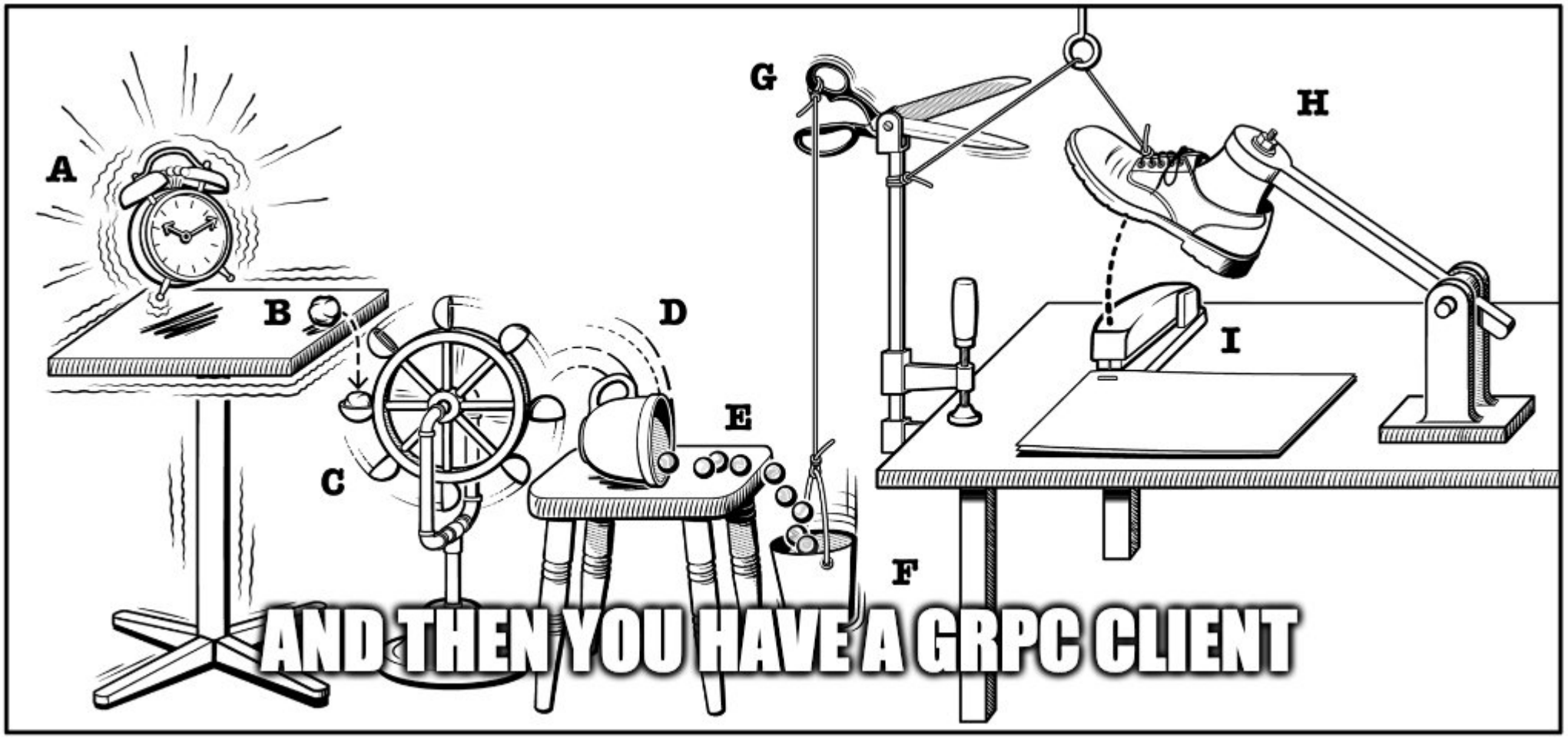
Service

Interfaces for RPCs

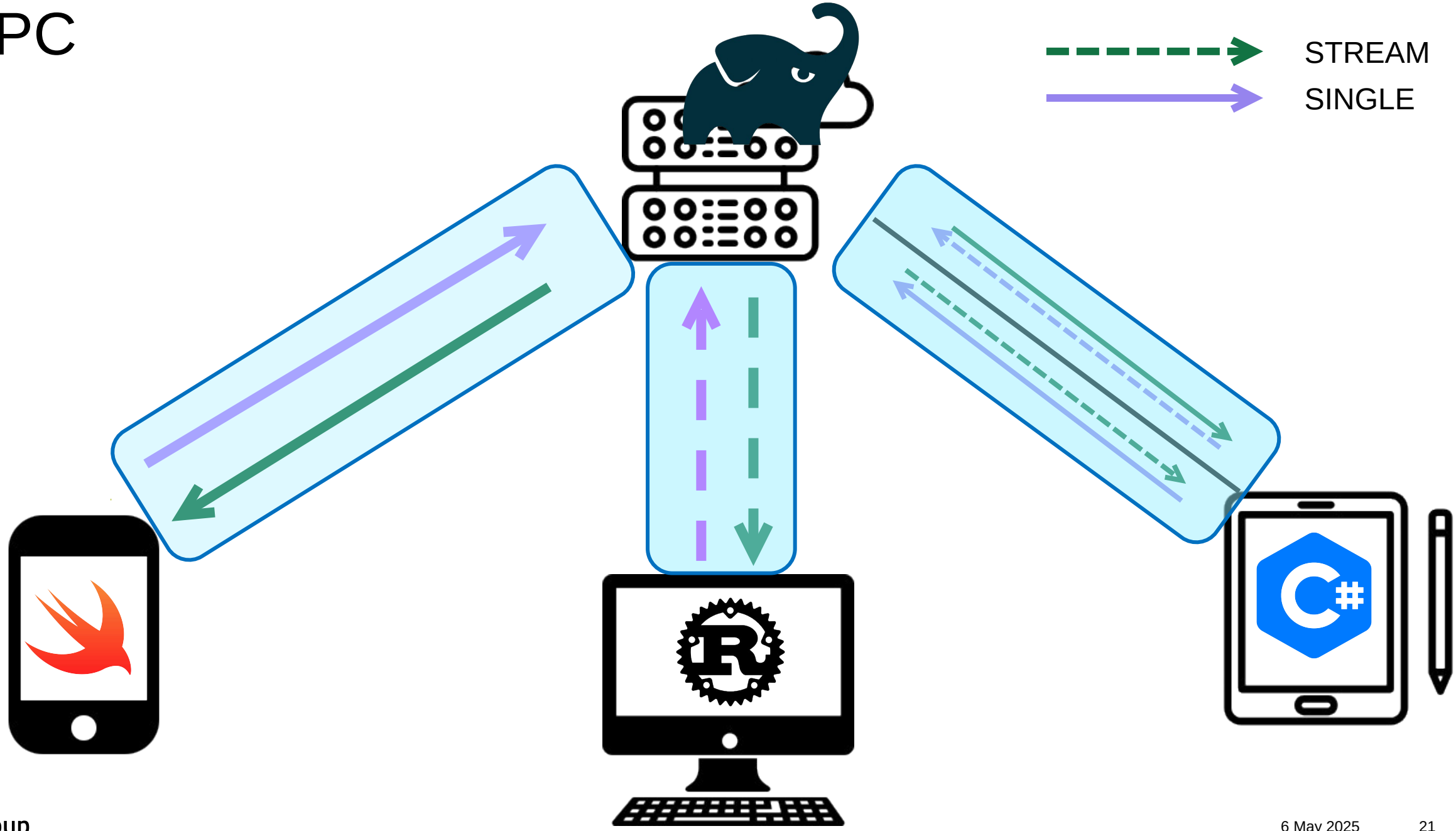
```
// eventhub.grpc.pb.h

class EventHub {
    static std::unique_ptr<Stub> NewStub(channel);      // “Client”
    // Stub interfaces ...

     class AsyncService {                                // “Server”
        void RequestGetEvent(...);
        void RequestSubscribe(...);
        void RequestNotify(...);
        void RequestExchange(...);
    };
};
```

Asynchronous

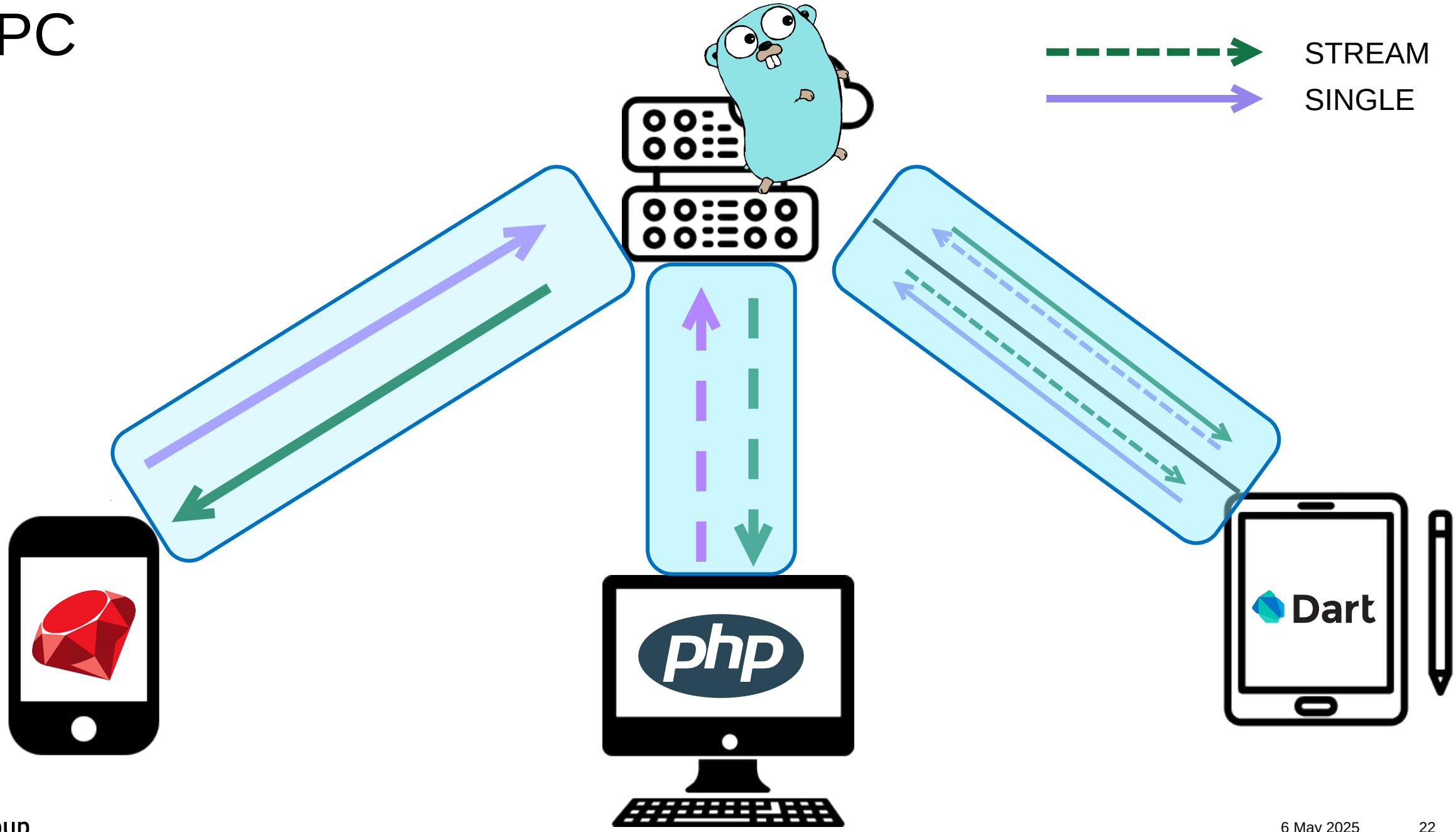
gRPC



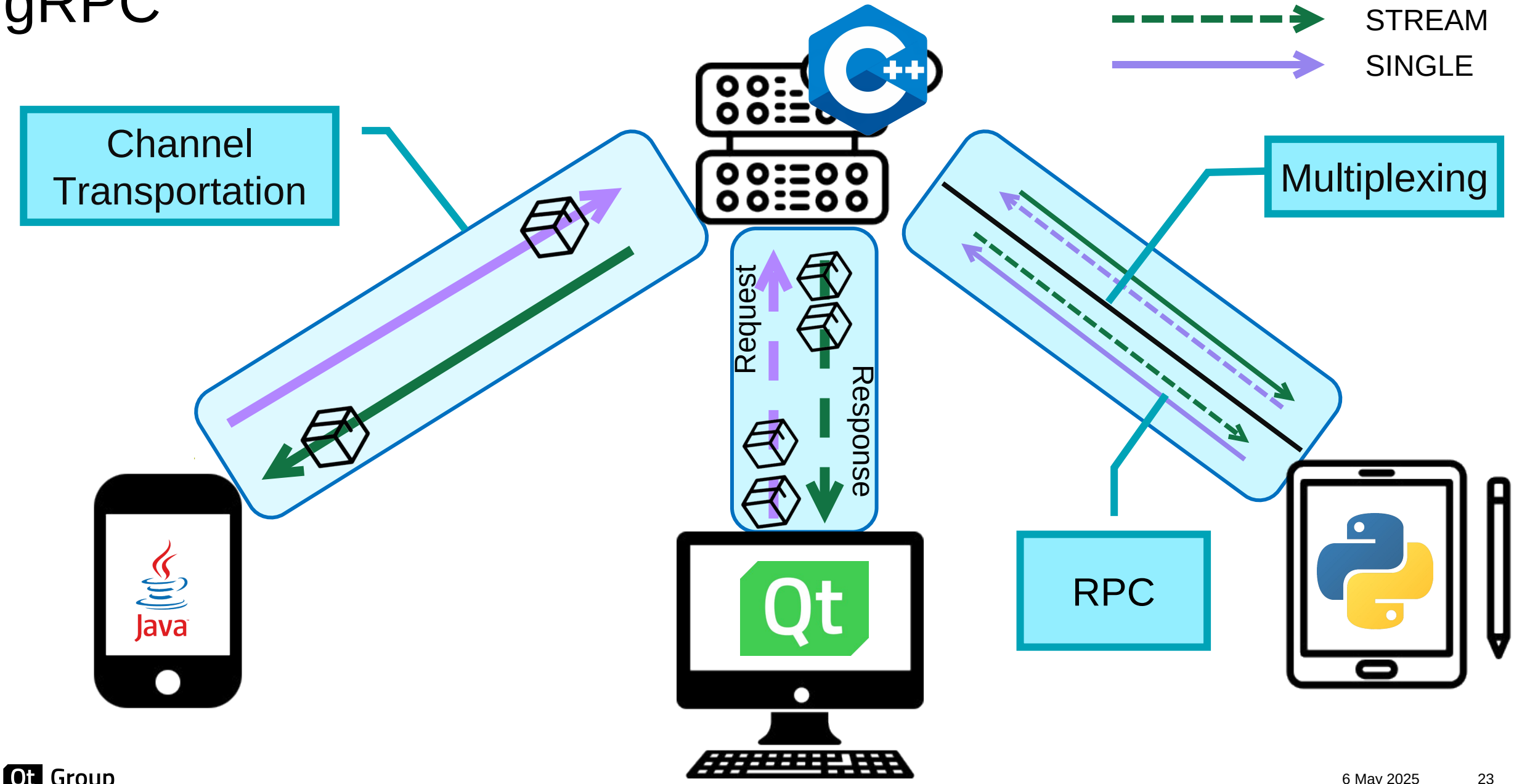
gRPC



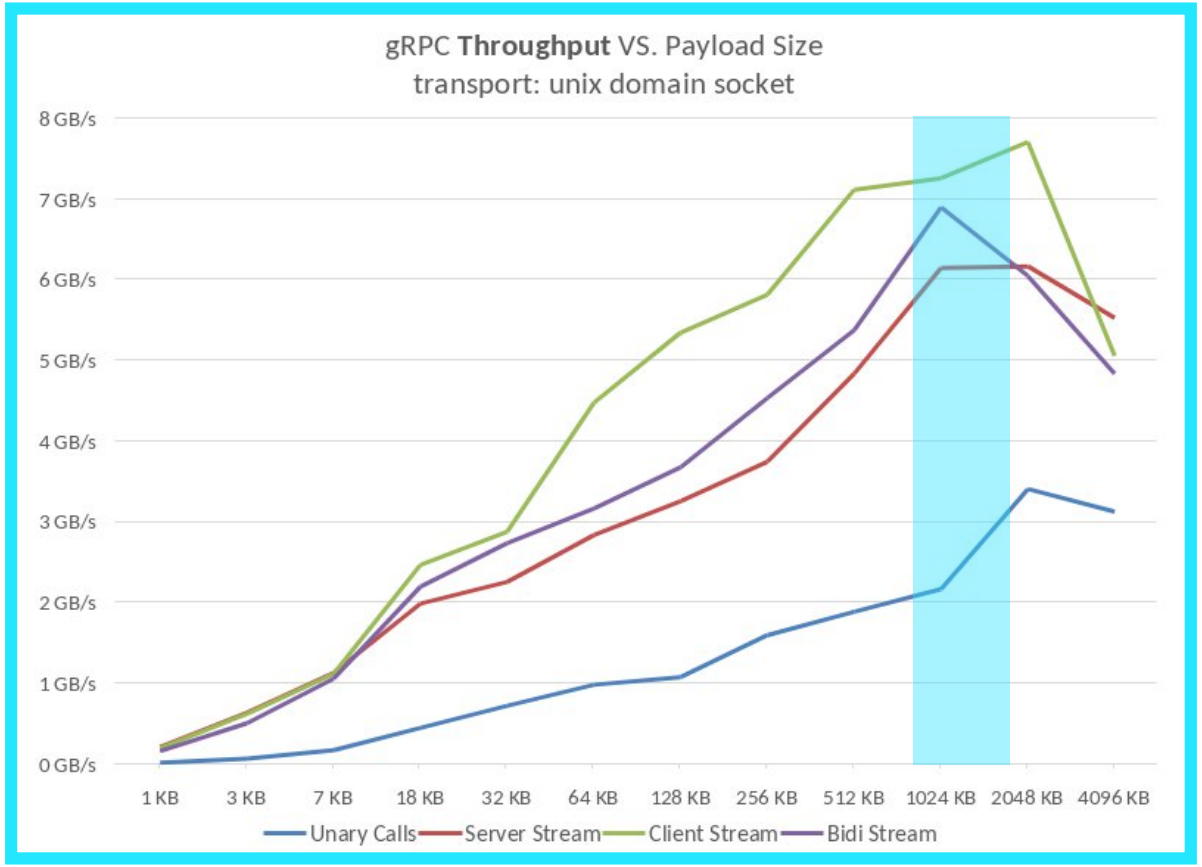
gRPC

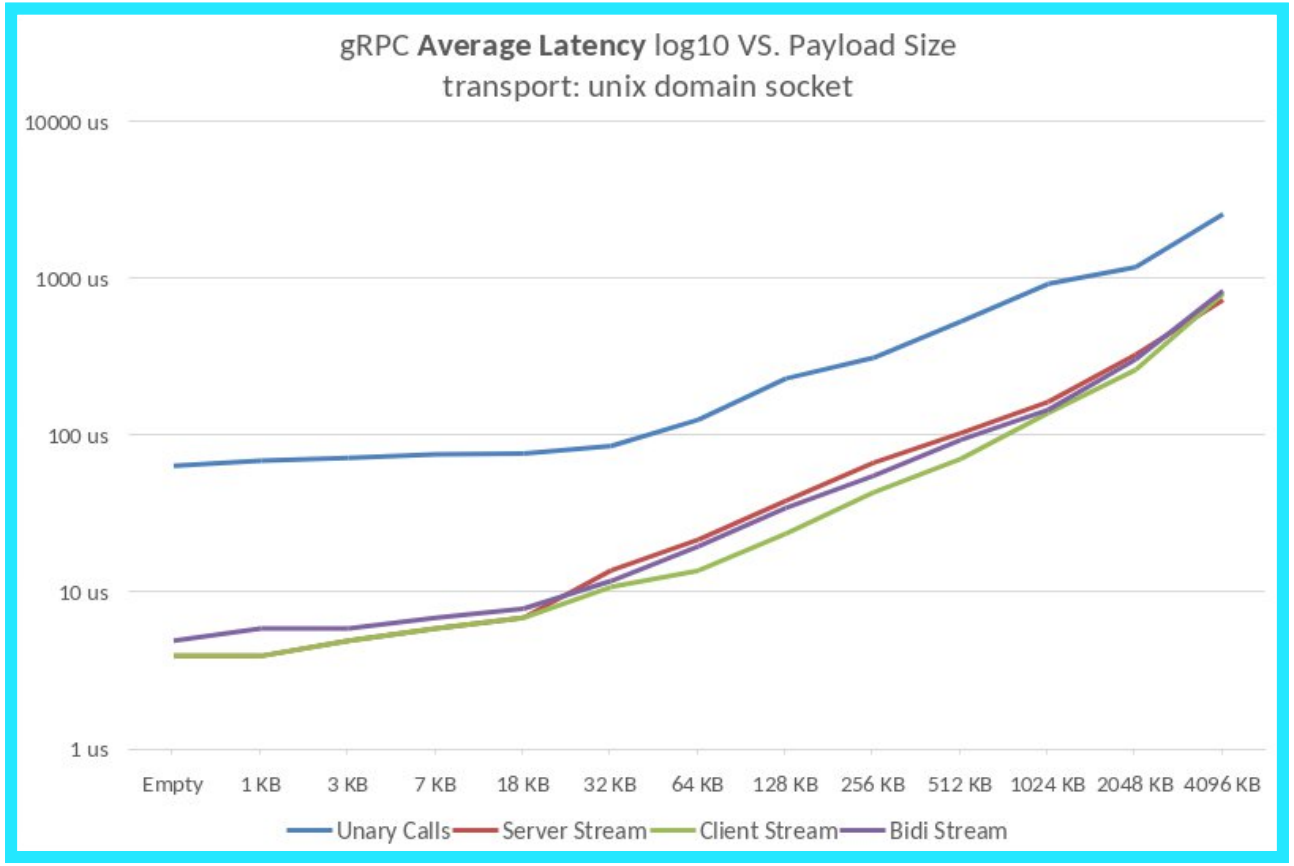


gRPC


STREAM
SINGLE
Channel
Transportation
Multiplexing
Request
Response
Java
Qt
RPC

gRPC





Reference

gRPC

gRPC

Uber

- 45% faster connect latency in P95s

Reference

NETFLIX

- “Incredible reduction in P99s”
- Developer productivity
- Reduced code complexity

Reference

Spotify®

- Stability, Performance, Latency
- Standardization (Integrating new Devs)
- Ecosystem (LB, Caching, 3rdparty tools)

Reference

Dropbox

- Standardization + Uniformity,
- Reduction in boilerplate code

Reference

gRPC

1. Performance
   - ✅ Binary encoding / decoding
   - ✅ Compact & fast transmission
   - ✅ Low latency
2. Schema-Driven API
   - ✅ Language-Independent
   - ✅ Strongly typed definitions
   - ✅ Safe evolution & versioning
3. HTTP/2 Foundation
   - ✅ Multiple parallel Streams
   - ✅ Real-time data exchange
   - ✅ Fine-Grained networking options

- ❌ HTTP/1.x <> HTTP/2 Incompatible
- ❌ No Browser support for HTTP/2
  - ➢ gRPC-Web Proxy
- ❌ Integration Complexity

gRPC

Protobuf

QtProtobuf

QtGrpc

QtGrpc

1. Performance
   - Binary encoding / decoding
   - Compact & fast transmission
   - Low latency
2. Schema-Driven API
   - Language-Independent
   - Strongly typed definitions
   - Safe evolution & versioning
3. Channels (HTTP/2)
   - Multiple parallel Streams
   - Real-time data exchange
   - Fine-Grained networking options

- QtGrpc C++ Integration
  - Signals & Slots
  - Qt Networking
  - Qt Event Loop
- QtGrpc QML Integration
  - Quick RPCs for GUIs
- Tooling & Build Integration
- Great Documentation & Examples

*Since Qt **6.8** ...*

- A “Real” Qt Module
  - Stable & Binary Compatible APIs
- Many performance improvements

QtGrpc

```
protoc --plugin=protoc-gen-qtgrpc= \
$(which qtgrpcgen) eventhub.proto
```

Protobuf Definitions

```
syntax = "proto3";                  // eventhub.proto

import "event.proto";

message None {}

service EventHub {
  rpc Push(Event) returns (None) {}
  rpc Subscribe(None) returns(stream Event) {}
  rpc Notify(stream Event) returns (None){}
  rpc Exchange (stream Event) (stream Event) {}
}
```

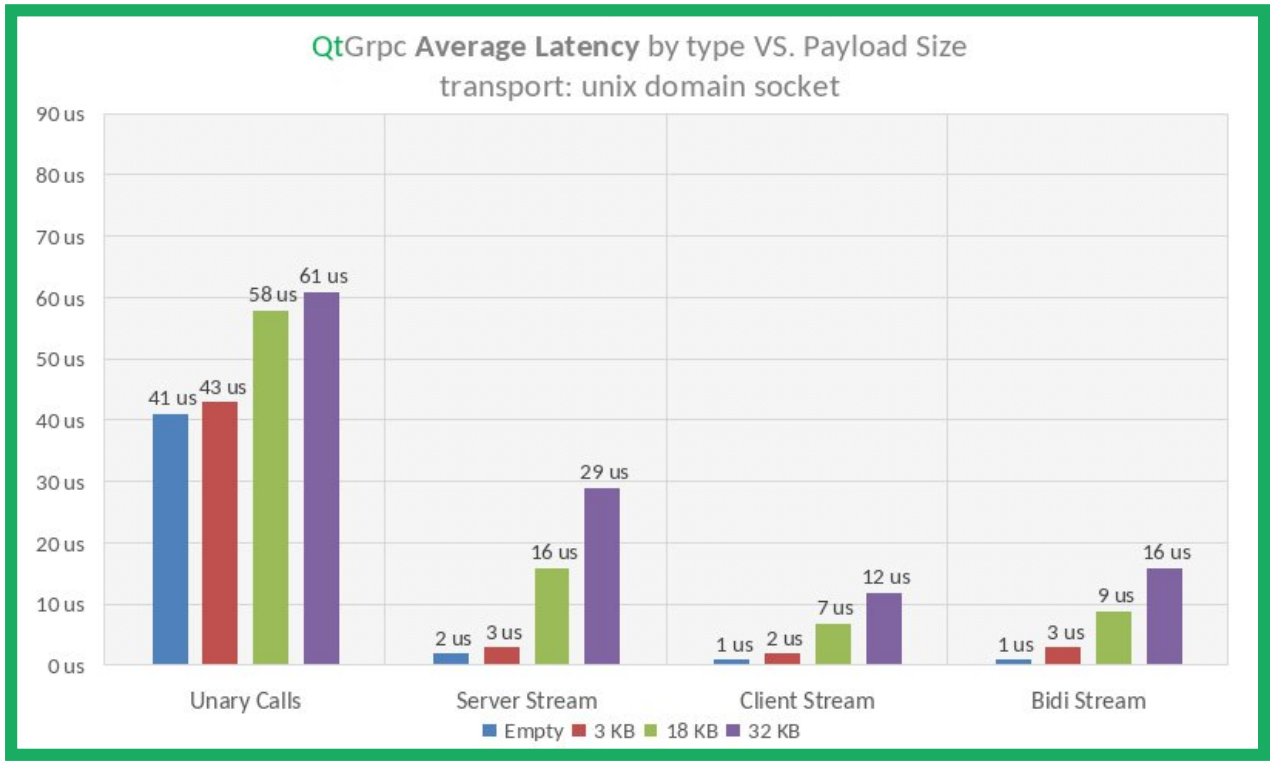
Service

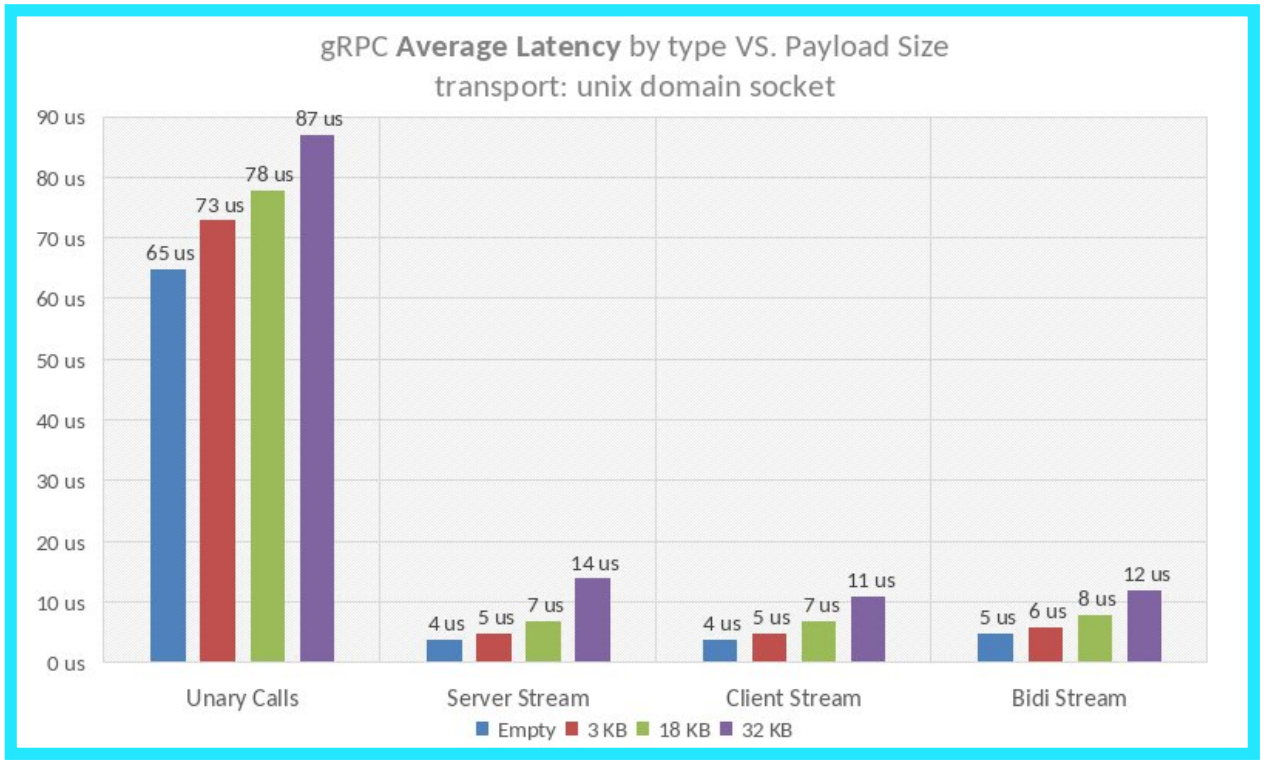
Qt Interfaces for RPCs

```
// eventhub_client.grpc.qpb.h

namespace EventHubService {

class Client : public QGrpcClientBase
{
    ...
    std::unique_ptr<QGrpcCallReply> Push(const Event &arg);
    std::unique_ptr<QGrpcServerStream> Subscribe(const None &arg);
    std::unique_ptr<QGrpcClientStream> Notify(const Event &arg);
    std::unique_ptr<QGrpcBidiStream> Exchange(const Event &arg);
    ...
};

} // namespace EventHubService
```

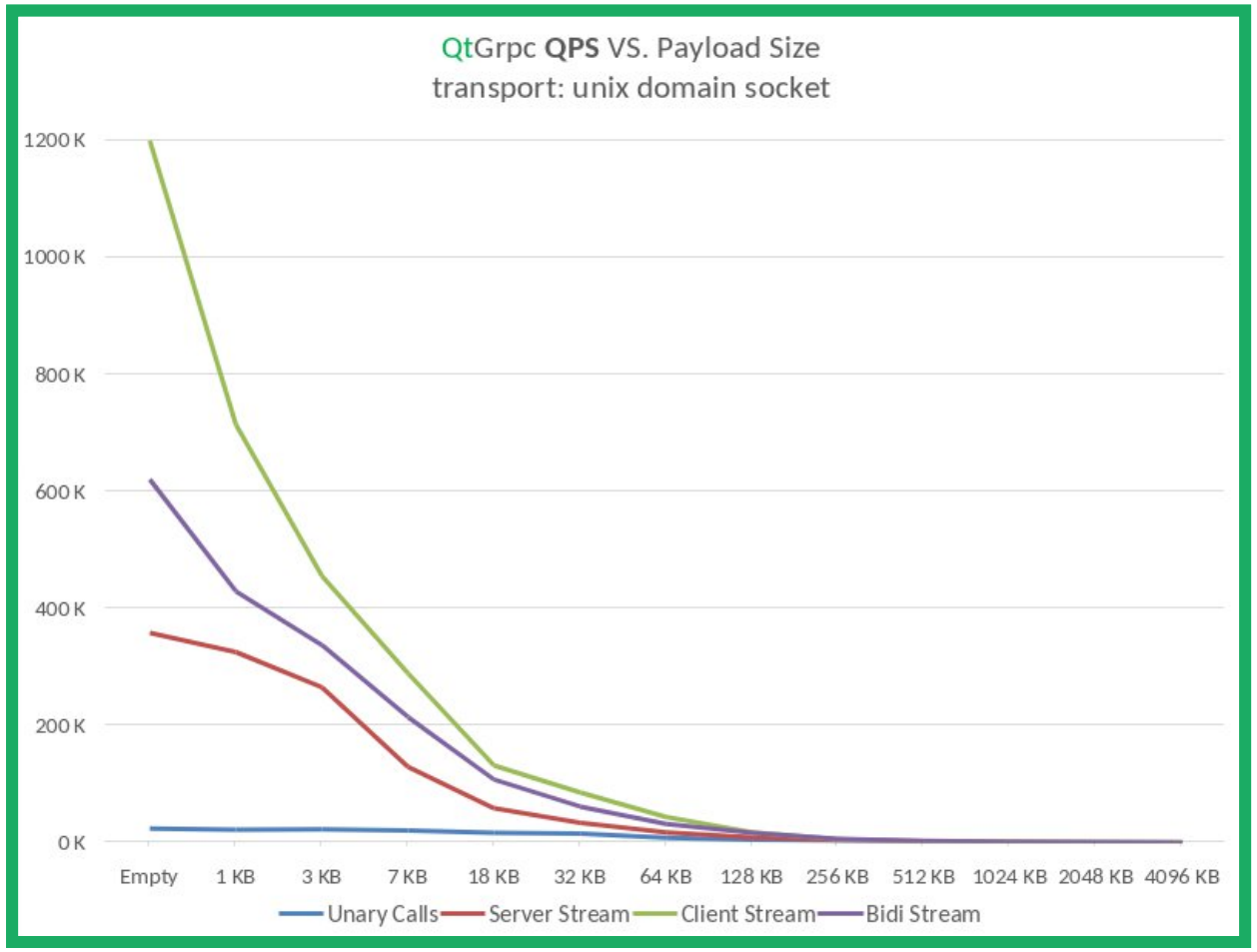
Asynchronous

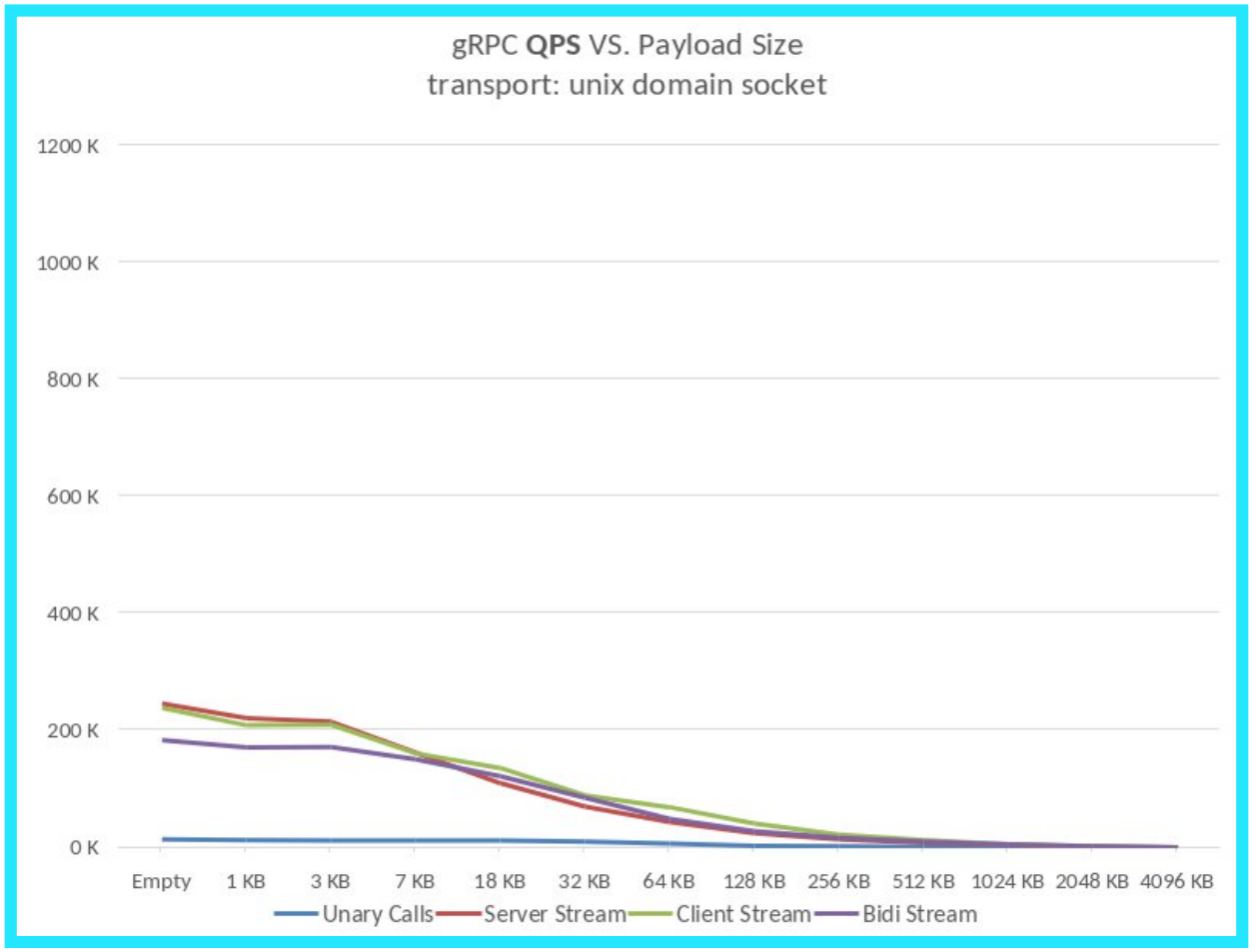
QtGrpc


QtGrpc Average Latency by type VS. Payload Size
transport: unix domain socket
90 us
80 us
70 us
60 us
50 us
40 us
30 us
20 us
10 us
0 us
41 us
43 us
58 us
61 us
2 us
3 us
16 us
29 us
1 us
2 us
7 us
12 us
1 us
3 us
9 us
16 us
Unary Calls
Server Stream
Client Stream
Bidi Stream
Empty
3 KB
18 KB
32 KB

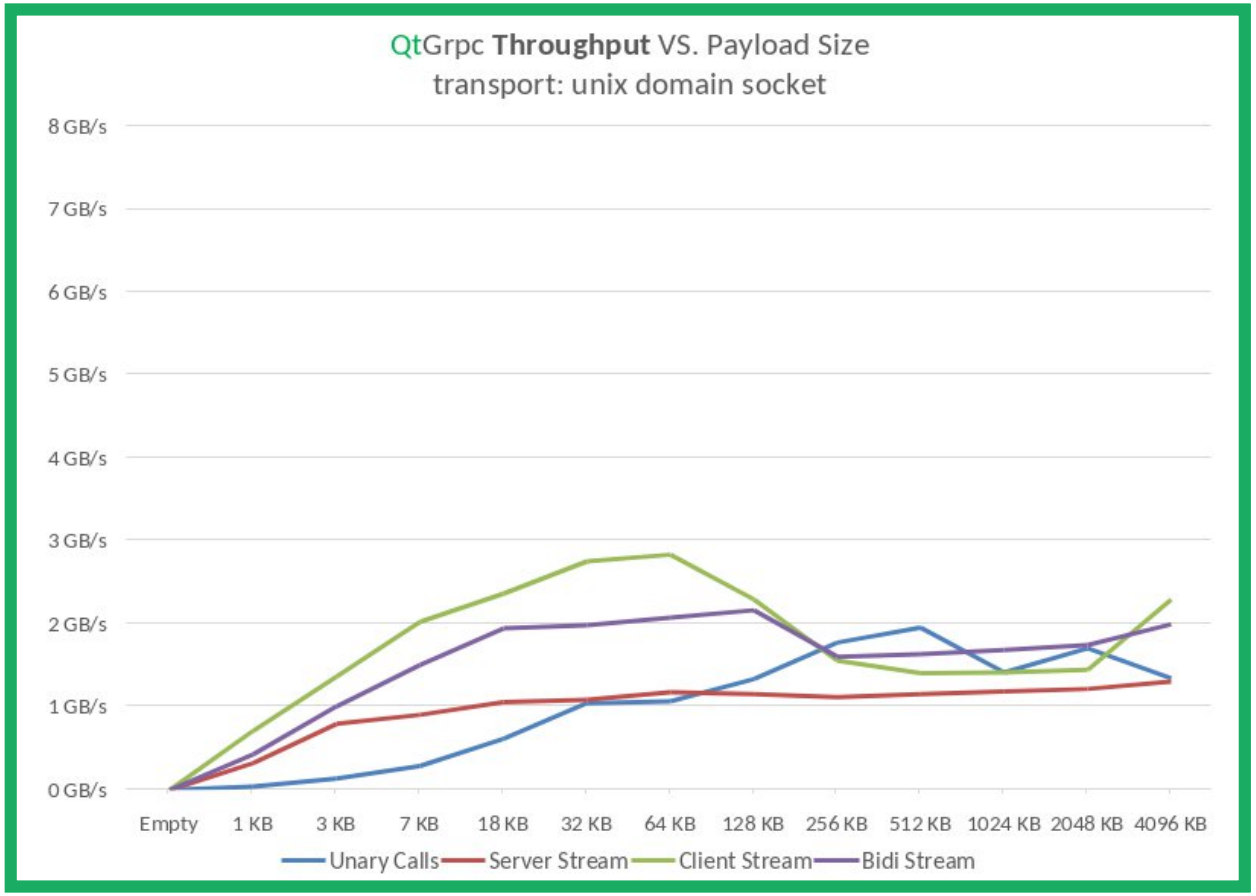

gRPC Average Latency by type VS. Payload Size
transport: unix domain socket
90 us
80 us
70 us
60 us
50 us
40 us
30 us
20 us
10 us
0 us
65 us
73 us
78 us
87 us
4 us
5 us
7 us
14 us
4 us
5 us
7 us
11 us
5 us
6 us
8 us
12 us
Unary Calls
Server Stream
Client Stream
Bidi Stream
Empty
3 KB
18 KB
32 KB

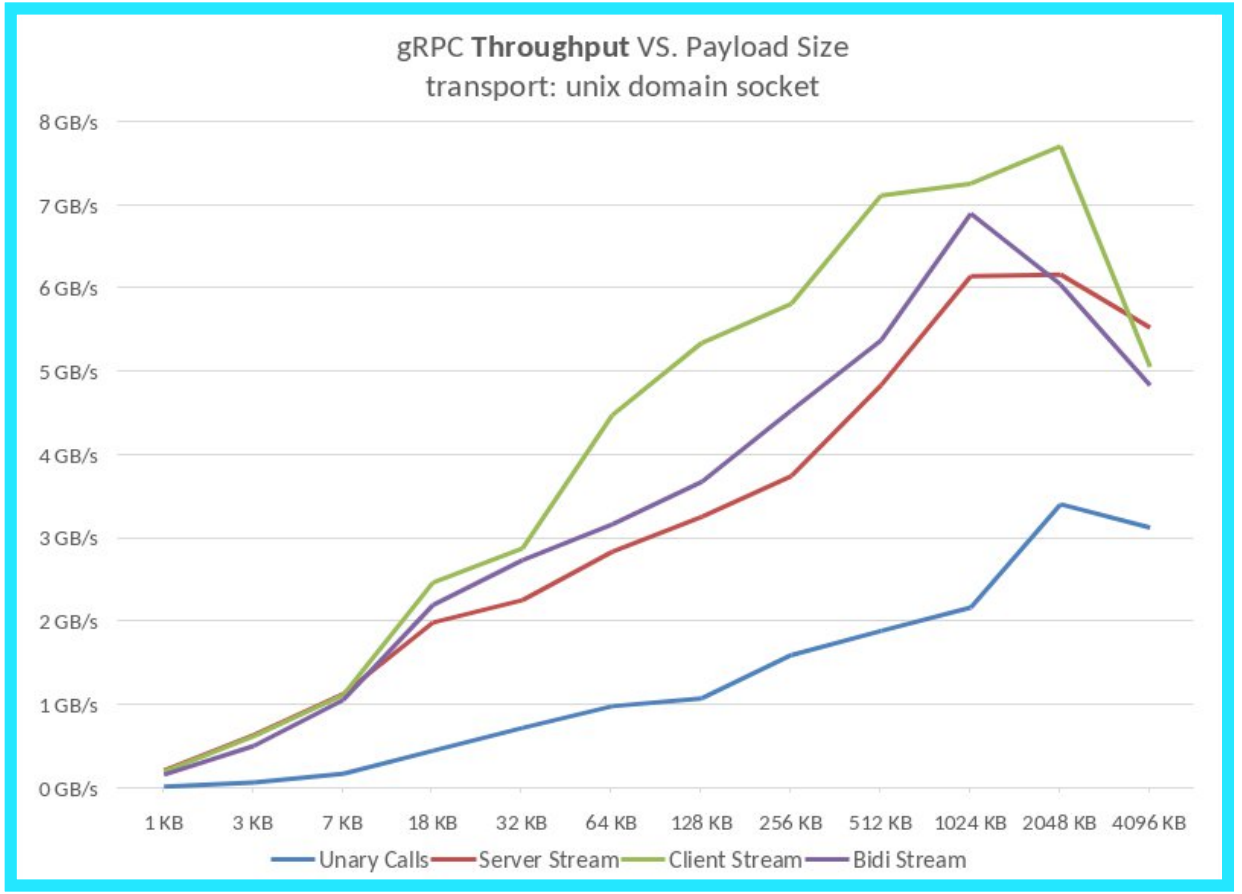
QtGrpc


QtGrpc QPS VS. Payload Size
transport: unix domain socket
1200 K
1000 K
800 K
600 K
400 K
200 K
0 K
Empty
1 KB
3 KB
7 KB
18 KB
32 KB
64 KB
128 KB
256 KB
512 KB
1024 KB
2048 KB
4096 KB
Unary Calls
Server Stream
Client Stream
Bidi Stream


gRPC QPS VS. Payload Size
transport: unix domain socket
1200 K
1000 K
800 K
600 K
400 K
200 K
0 K
Empty
1 KB
3 KB
7 KB
18 KB
32 KB
64 KB
128 KB
256 KB
512 KB
1024 KB
2048 KB
4096 KB
Unary Calls
Server Stream
Client Stream
Bidi Stream

QtGrpc

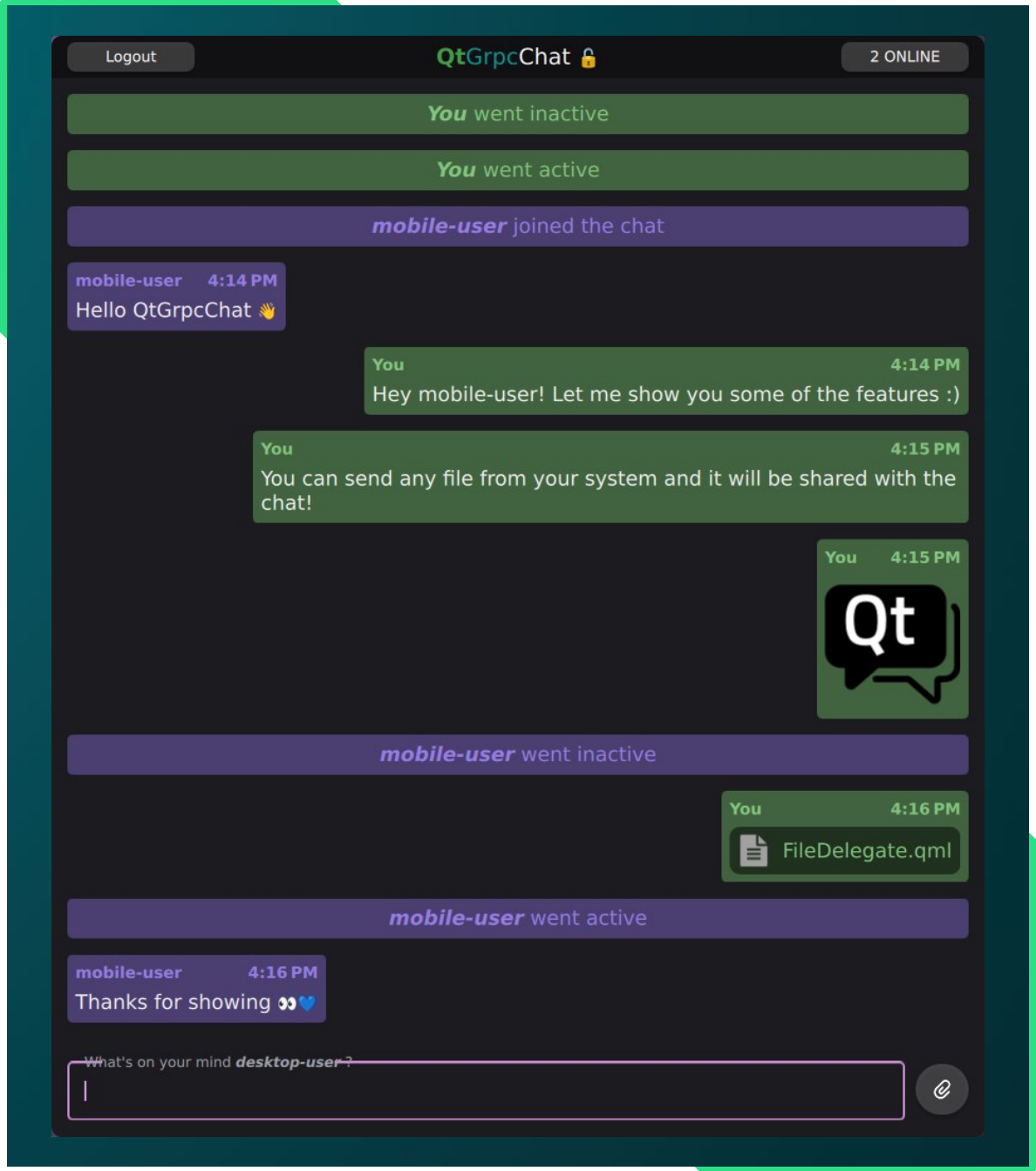




Reference

QtGrpc

DEMO


Logout
QtGrpcChat
2 ONLINE
You went inactive
You went active
mobile-user joined the chat
mobile-user 4:14 PM
Hello QtGrpcChat
You 4:14 PM
Hey mobile-user! Let me show you some of the features :)
You 4:15 PM
You can send any file from your system and it will be shared with the chat!
You 4:15 PM
mobile-user went inactive
You 4:16 PM
FileDelegate.qml
mobile-user went active
mobile-user 4:16 PM
Thanks for showing
What's on your mind desktop-user?

DENNIS OBERST

linkedin.com/in/deeedob

github.com/deeedob

dennis.oberst@qt.io

THANK YOU

QtWS25