GRPC

gRPC

- Multi-language RPC for Web Services
- PL-independent IDL
- Generate PL-specific bindings
- Protocol buffers

Principles (1/2)

- Services not Objects,
  Messages not References
  - Fallacies of ignoring network
  - Pitfalls of distributed objects
- Interoperable
- Layered
  - Ex: authentication layer
- Payload agnostic
  - JSON, XML, Thrift, protocol buffers, etc

92

92

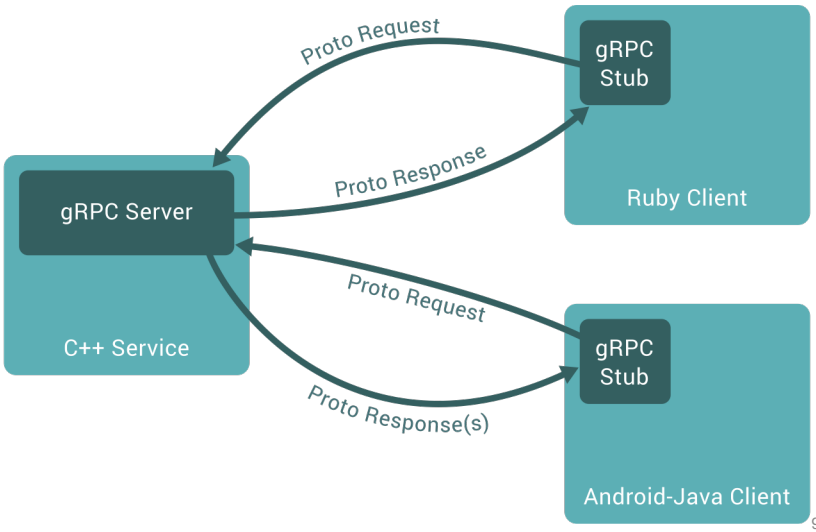
Principles (2/2)

- Streaming
- Blocking and Non-blocking
- Lameducking
- Flow Control
- Pluggable
  - security, load-balancing, failover, health-checking
- Meta-data Exchange
- Standardized Status Codes

93

93

gRPC



94

94

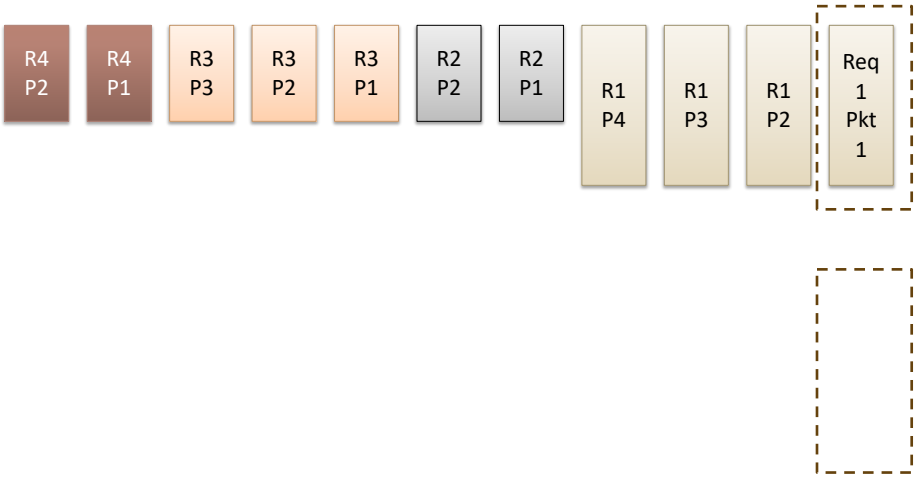
HTTP/1.1

- Connections vs requests
- Keep connections alive for further requests
- Head-of-line blocking

95

95

HTTP/1.1: Head of Line Blocking



96

96

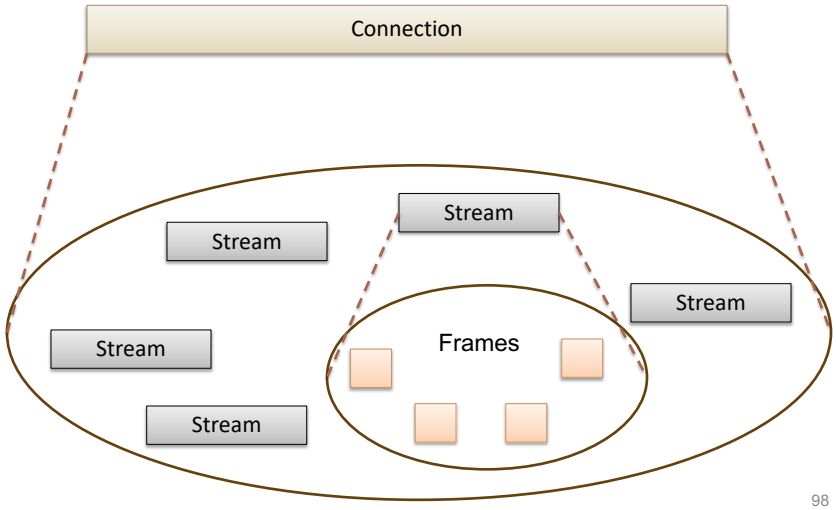
HTTP/2

- Semantic layer above connections
- Streams and frames
- Concurrent streams over a connection

97

97

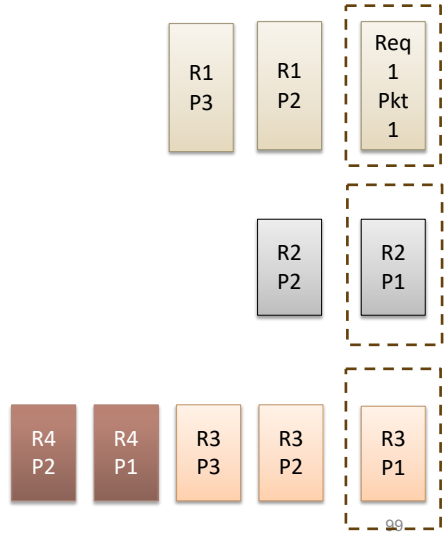
HTTP/2: Streams and Frames



98

98

HTTP/2: Concurrency via Streaming



99

99

HTTP/2

- Semantic layer above connections
- Streams and frames
- Concurrent streams over a connection
- Flow Control
  - Clients sharing connection
  - Client gives sender a "budget" (buffer space)
- Smarter Proxies
  - Create min # of connections based on BDP
  - Load balancing across streams, etc

100

100

HTTP/2

- Semantics around graceful close
  - GOAWAY control frame
- Header compression
- Server push
- Pinging
- Stream priority

101

101

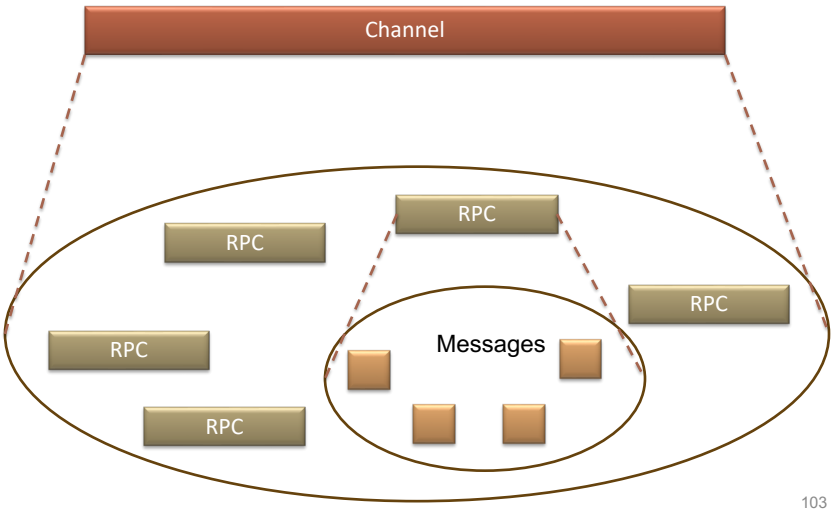
gRPC Semantics

- Channels
- Remote Procedure Calls (RPCs)
- Messages

102

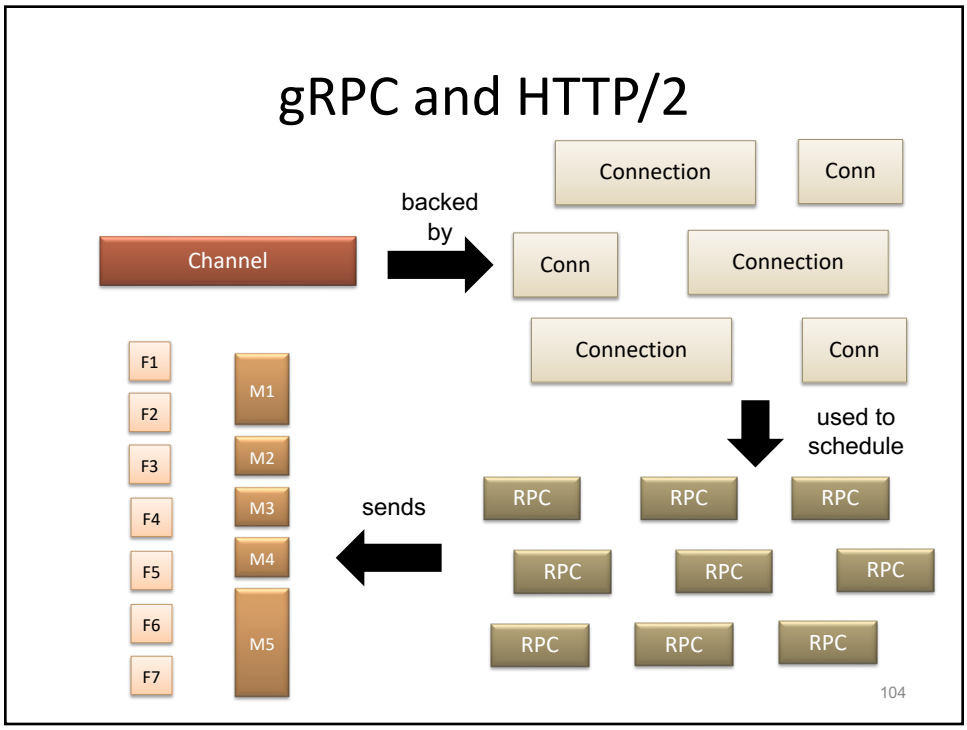
102

gRPC: RPCs and Messages

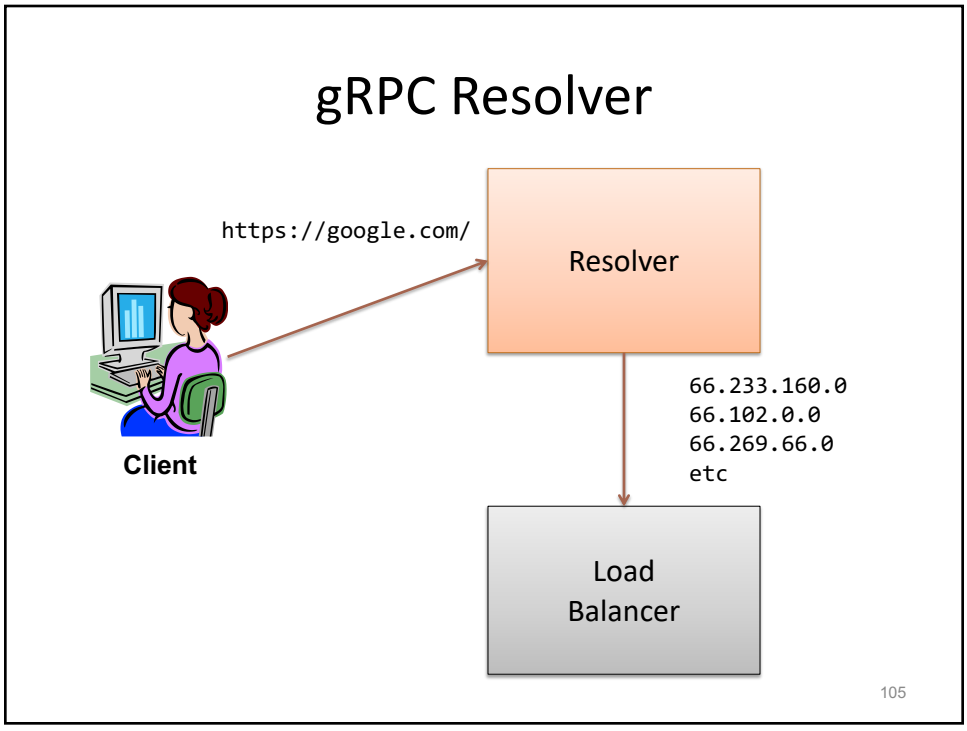


103

103

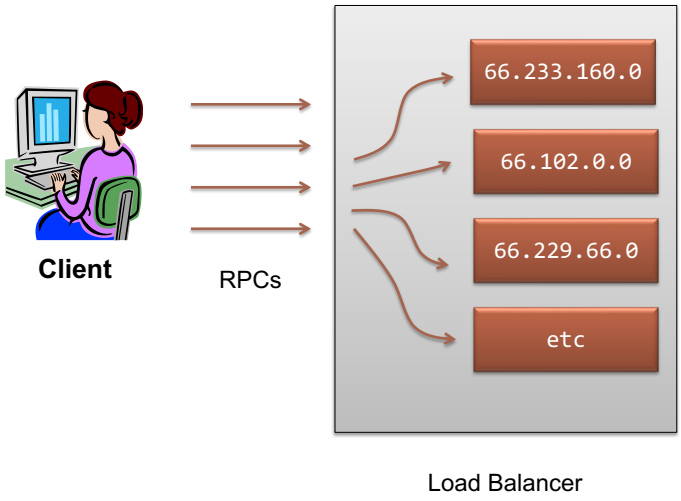

gRPC and HTTP/2
Connection
Conn
backed by
Channel
Conn
Connection
Connection
Conn
used to schedule
F1
F2
F3
F4
F5
F6
F7
M1
M2
M3
M4
M5
sends
RPC
RPC
RPC
RPC
RPC
RPC
RPC
RPC
RPC
104

104


gRPC Resolver
https://google.com/
Resolver
66.233.160.0
66.102.0.0
66.269.66.0
etc
Client
Load Balancer
105

105

gRPC Load Balancer



106

106

gRPC Connection Management

- Multiple streams over shared connection
- Stream multiplexed over several connections
- Connection failure:
  - Load balancer reconnects using known addresses
  - Resolver re-resolves addresses
- Detecting failed connections
  - Clean failure: TCP FIN
  - Endpoint failure: HTTP/2 PING frames
  - PING frames keep connection alive for proxies

107

107

Messages

- Define message format:

```
message Person {
   string name = 1;
   int32 id = 2;
   bool has_ponycopter = 3;
}
```

- Compile using **proto** compiler

108

108

Service

```
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

message HelloRequest {
  string name = 1;
}

message HelloReply {
  string message = 1;
}
```

109

109

RPC Styles

- Unary RPC

```
rpc SayHello(HelloRequest)
returns (HelloResponse);
```

- Server Streaming

```
rpc LotsOfReplies(HelloRequest)
returns (stream HelloResponse);
```


RPC Styles

- Client Streaming

```
rpc LotsOfGreetings(stream HelloRequest)
returns (HelloResponse);
```

- Bidirectional Streaming

```
rpc BidiHello(stream HelloRequest)
returns (stream HelloResponse);
```

gRPC Features

- Synchronous vs Asynchronous
- Deadlines/Timeouts
  - timeouts (duration of time)
  - deadlines (points in time)
- Termination
  - Independent on client and server
- Cancellation
  - No rollback
- Metadata
  - Name-value pairs
- Channels
  - Channel state (idle, running), compression

112

112

PROTOCOL BUFFERS

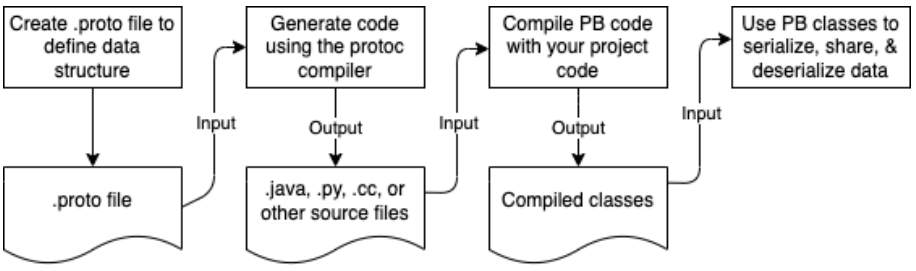
113

113

Protocol Buffers

- Alternative to JSON
  - Protocol buffer data definition language
  - Compiler
  - Runtime library
  - Runtime data formats

Protocol Buffers Workflow



Example

- Proto file

```
syntax = "proto3";
message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}
```

- Usage

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args[0]);
john.writeTo(output);
```

116

116

Protocol Buffer Definitions

- Field numbers (unique)
- Field properties:
  - optional
  - repeated
- Field types:
  - scalar (int32, int64, float, double, bool, string, etc)
  - enumeration
  - message (nested)
  - map

117

117

Protocol Buffer Advantages

- Language independence
- Compact storage
- Fast parsing

- Forward and backward compatibility
  - Addition of fields
  - Deletion of fields

118

118

Protocol Buffer Limitations

- Entire message fits in memory
- Different serialization formats
- Messages are not compressed
- Not efficient for scientific
  - multi-dimensional arrays of floating point
- Not self-describing
  - use reflection with .proto file
- Not an industry standard

119

119

USING GRPC

120

120

Service

```
service Greeter {
  // Sends a greeting
  rpc sayHello (HelloRequest) returns (HelloReply) {}
}

message HelloRequest {
  string name = 1;
}

message HelloReply {
  string message = 1;
}
```

121

121

Server

```
class ServerStub extends GreeterGrpc.GreeterImplBase {

  @Override
  public void sayHello(HelloRequest req,
                       StreamObserver<HelloReply> obs) {









  }
}
```

122

122

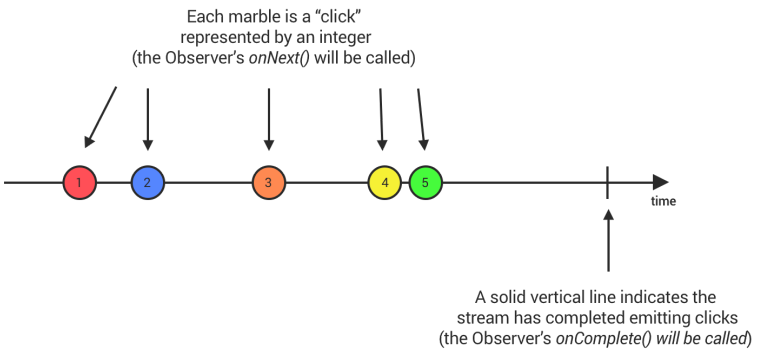
Stream Observer

```
public interface StreamObserver<T> {
   void onNext(T value);
   void onComplete();
   void onError(Throwable e);
}
```

123

123

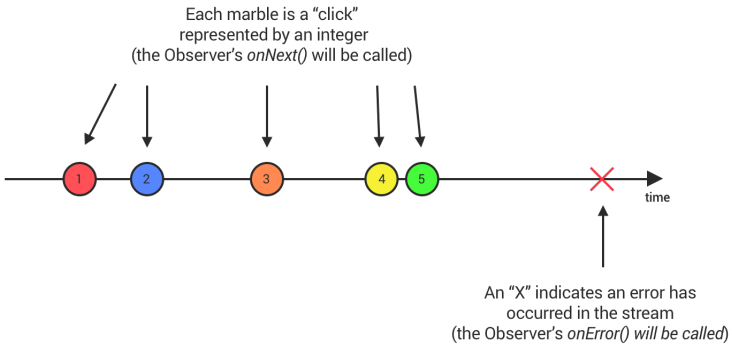
Marble Diagrams



124

124

Marble Diagrams



125

125

Server

```
class ServerStub extends GreeterGrpc.GreeterImplBase {

  @Override
  public void sayHello(HelloRequest req,
                       StreamObserver<HelloReply> obs) {
    HelloReply reply =
       HelloReply.newBuilder()
                 .setMessage("Hello " + req.getName())
                 .build();
    responseObserver.onNext(reply);
    responseObserver.onCompleted();
  }
}
```

126

126

Server

```
ServerCredentials cr = InsecureServerCredentials.create();
Server server = Grpc.newServerBuilderForPort(port, cr)
  .addService(new ServerStub())
  .build()
  .start();

Runtime.getRuntime().addShutdownHook(new Thread() {() -> {
  try {
    server.shutdown().awaitTermination(30, TimeUnit.SECONDS);
  } catch (InterruptedException e) {
    e.printStackTrace(System.err);
  }
});

server.awaitTermination();
```

127

127

Client

```
public class ClientStub {

  GreaterRpc.GreeterBlockingStub blockingStub;

  public ClientStub(Channel channel) {
    blockingStub = GreeterRpc.newBlockingStub(channel);
  }

  public String greet(String name) {
    HelloRequest request = HelloRequest
                           .newBuilder()
                           .setName(name).build();
    HelloReply response = blockingStub.sayHello(request);
    return response.getMessage();
  }
}
```

128

128

Client

```
ChannelCredentials cr = InsecureChannelCredentials.create();
ManagedChannel channel = Grpc.newChannelBuilder(..., cr)
  .build();

ClientStub client = new ClientStub(channel);

System.out.println(client.greet("..."));

channel.shutdownNow().awaitTermination(5, TimeUnit.SECONDS);
```

129

129