ONOS gRPC Brigade

Jian Li

ONOS/CORD Ambassador Steering Team, Open Networking Foundation (ONF), US

ONOS/CORD Working Group, SDN/NFV Forum, Korea

PIRL, POSTECH, Korea

jian@opennetworking.org

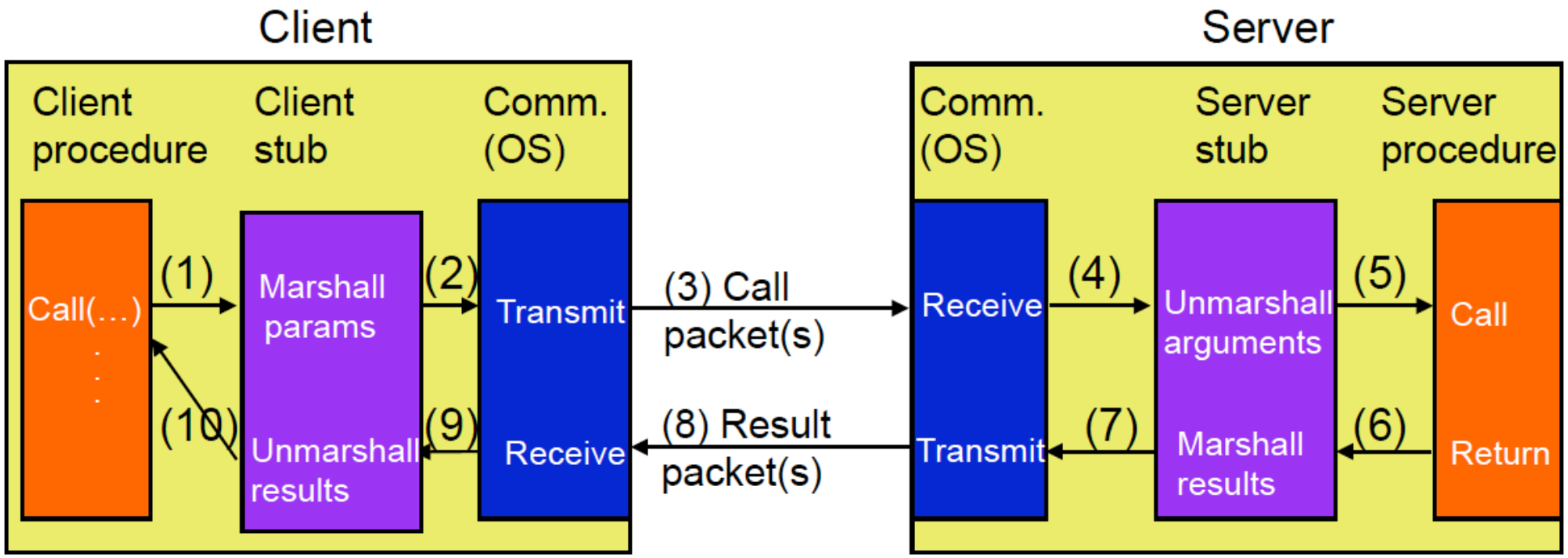
ONOS Build 2017

Agenda

- Background
- gRPC and Protocol Buffer
- gRPC with ONOS
- Connection Variants and Uses
- Challenges
- Roadmap

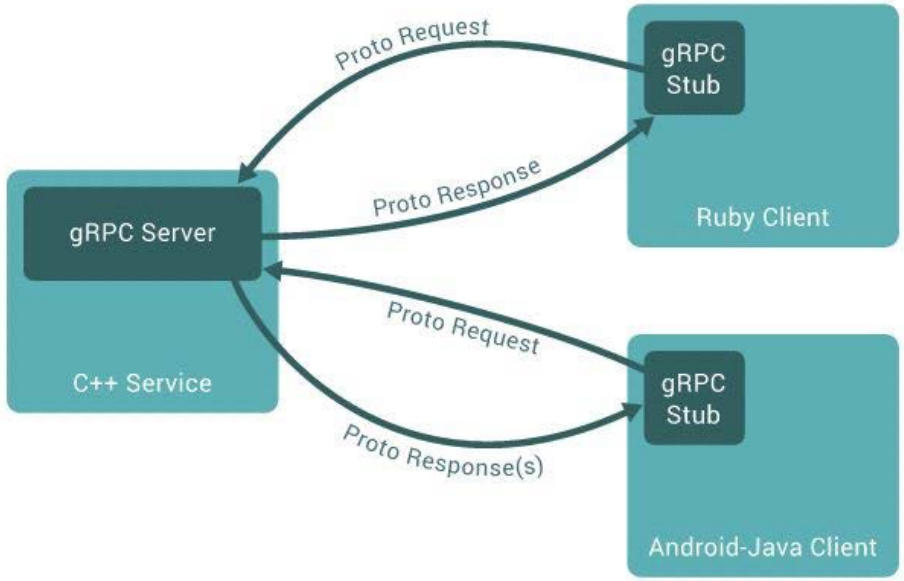
Background

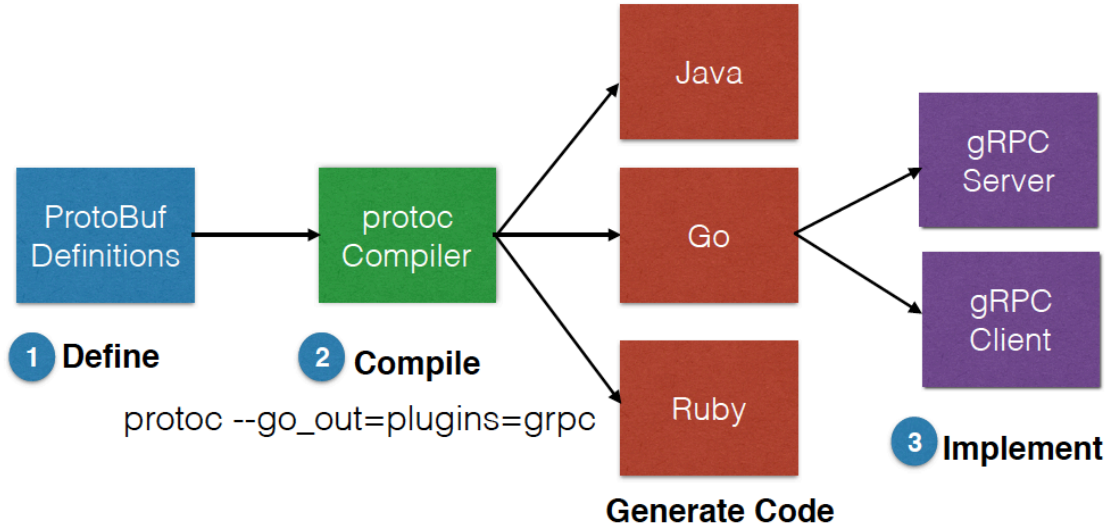
- Remote Procedure Call (RPC)
  - Objective: make inter-process communication among remote processes similar to local ones
  - RPC achieves its transparency in a way analogous to the read operation in a traditional single processor system



What is gRPC

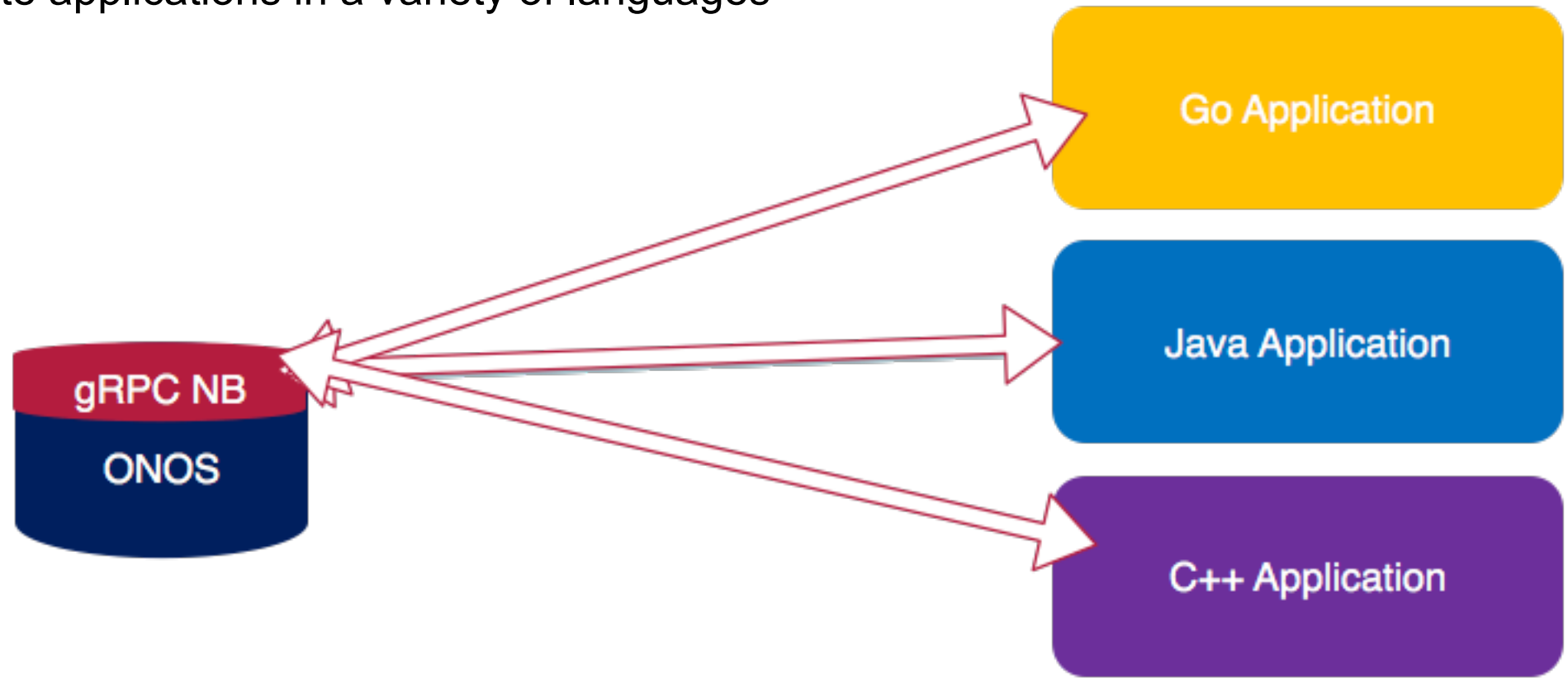
- gRPC
  - A google RPC provides high-performance, open source universal RPC framework
- Two Components
  - Protocol Buffer (protobuf)
    - Provides efficient encoding of data structures
  - gRPC servers
    - Provide efficient easy to use and secure transport for protobuf messages





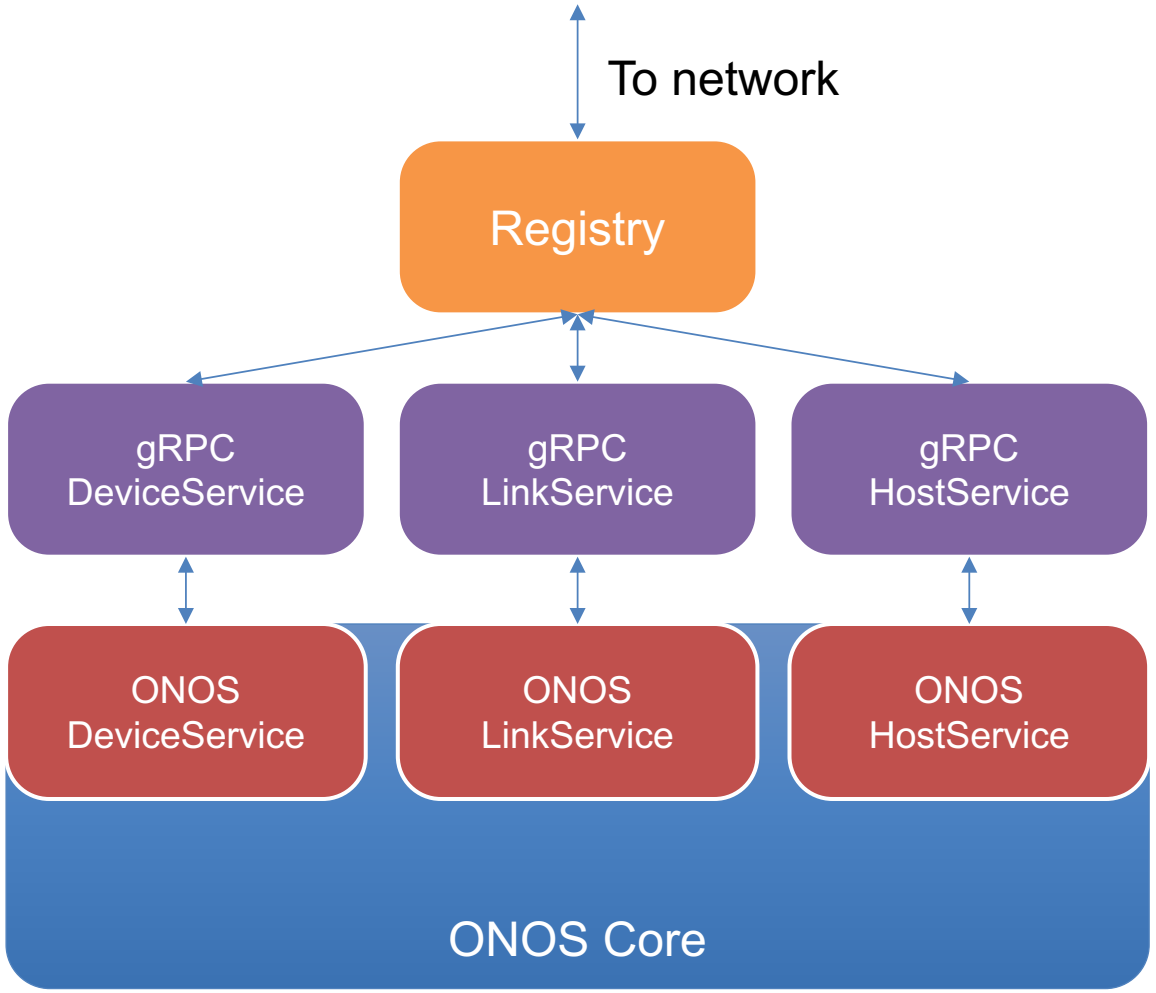
gRPC with ONOS

- gRPC with ONOS
  - gRPC provides efficient, compact and granular communication between the core ONOS system and remote applications in a variety of languages

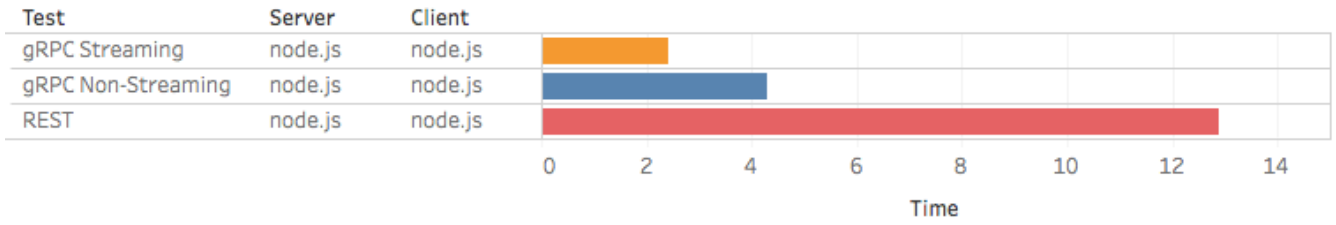


gRPC with ONOS

- gRPC with ONOS
  - gRPC connects to the core ONOS and passes through requests and serves up responses
  - Each gRPC service maps to a single ONOS core service
    - There can only be one instance of each service active at a time
  - Provide higher performance compared to existing REST API

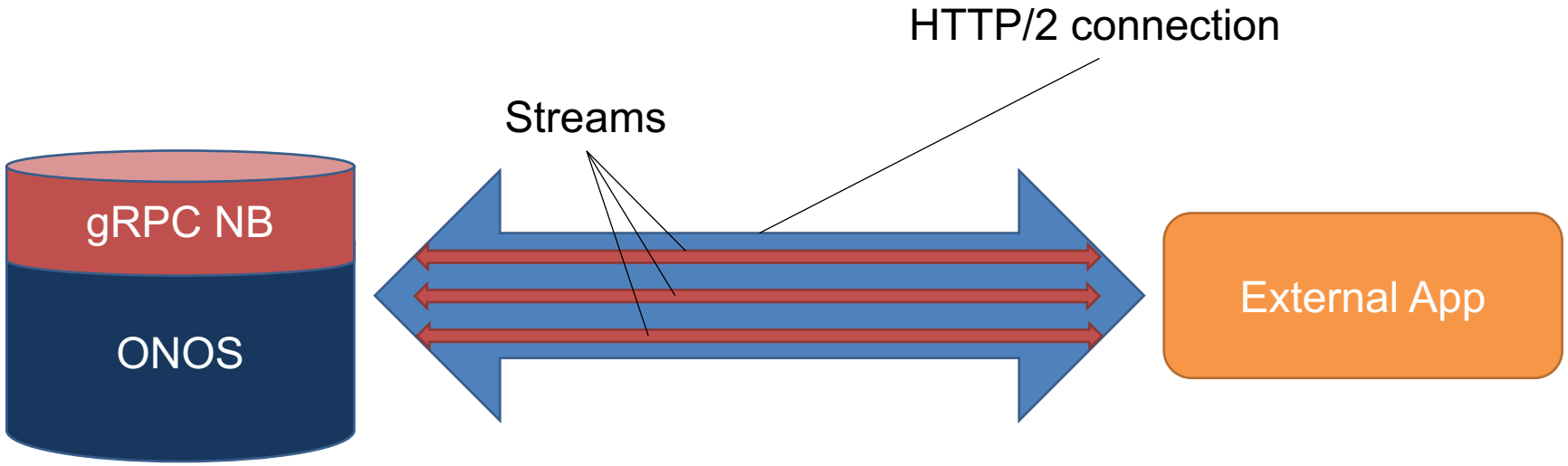


10,000 ping-pongs over the “same” connection



gRPC with ONOS

- gRPC with ONOS
  - gRPC communicates with external applications via http2
  - Serialized message formats are extremely compact
  - Connections are multiplexed and avoid blocking

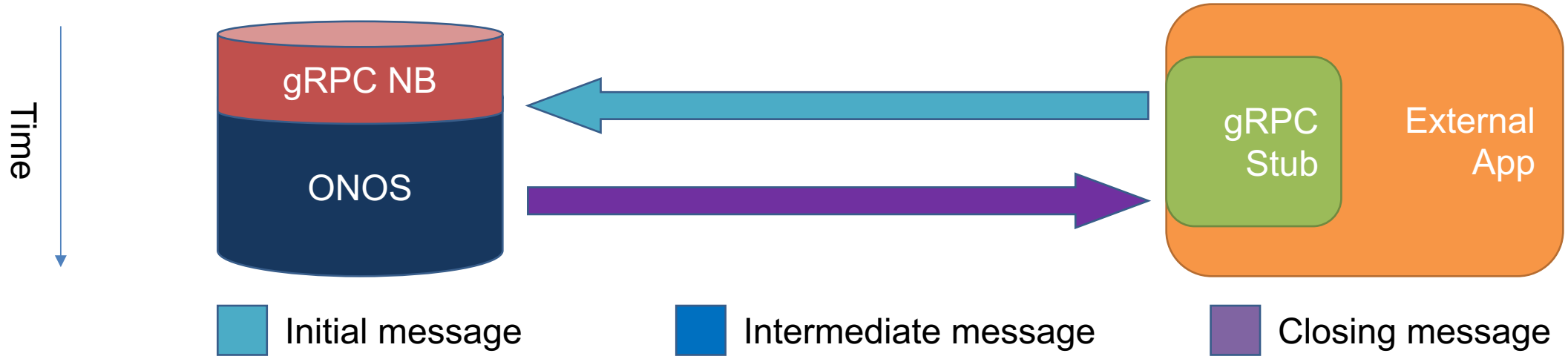


gRPC with ONOS

- gRPC with ONOS
  - By compiling the provided gRPC models an off platform developer can produce a stub in their (supported) language of choice
  - A stub provides the basic functionality for sending requests to and receiving requests from the ONOS gRPC NB
    - Synchronous or blocking stubs can have one outstanding call at a time
    - Asynchronous stubs return immediately and allow calls to carry on in parallel, in cases like UI these are strongly preferred and are the only stubs capable of supporting streaming from client to server

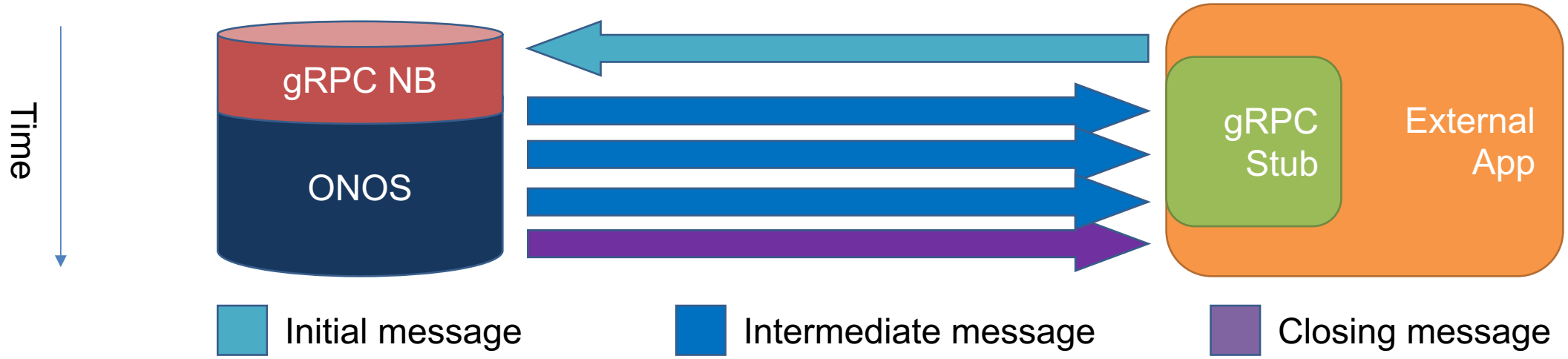
Connection Variants and Uses

- One Call, One Response
  - Simplest case, client sends a query and receives a response resolving the query
  - Most calls fall into this category, calls such as requesting the number of devices, element type or link type
  - Easily implemented synchronously or asynchronously



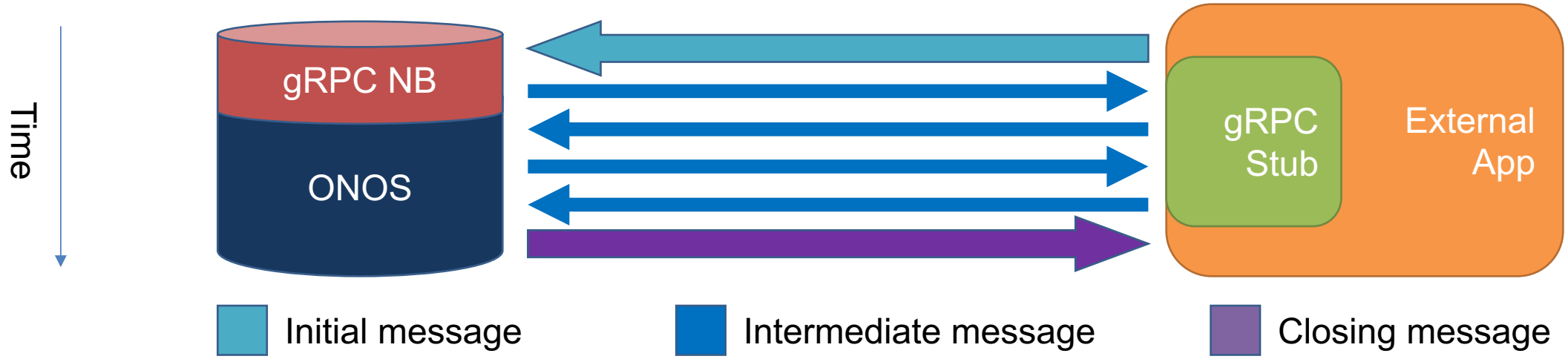
Connection Variants and Uses

- One Call, Many Responses
  - Client sends a single message and receives a series of responses
  - Used for the transmission of large datasets, the client initiates an exchange and receives a series of relatively small responses until the requested data has been transmitted



Connection Variants and Uses

- Many calls, Many Response
  - Client and server exchange numerous messages in both directions, they do not need to be pairwise as in the example
  - A client requests and receives periodic updates throughout the active life of the application
    - E.g., maintains a cache



Brigade Goal

- Goal
  - Offering off-platform application performance and granularity of control
    - Previously only available to local (on-platform) applications
  - Language compatibility
    - Enables applications in languages other than Java to interact effortlessly with the core
  - Moving functionality to off-platform
    - Make it possible to improve the isolation of the ONOS core services from applications
    - Reduce the likelihood of crashes
    - Increase the percentage of system resources that can be dedicated to core ONOS

Benefits

- Speed
  - Dramatically increases message throughput when compared to systems like REST by providing support for multiplexed connections
- Size
  - Protobuf encoding reduces the size of messages by omitting fields with default values
  - Binary formatted encoding can further reduce the size compared text
  - Efficient content-dependent encoding
    - Eliminate the need to transmit delimiters like the XML tags
- Simplicity
  - Protobuf models are reusable across languages
  - Minimize the effort an off-platform developer write an new off-platform ONOS app
- Compatibility
  - Effortless between clusters running different versions of ONOS

First Step

- BUCK Build for gRPC
  - Added a bucklet for gRPC
    - Download the necessary dependencies → compile protobuf model → build entire Java stub with ONOS
- Models
  - Created standards document
  - Converted a significant set of ONOS's internal model objects into protobuf model
  - The building blocks are ready for who wants to develop their own gRPC services
- Translation
  - Translating protobuf model into ONOS model is tedious, and vice versa
  - Provided translation utilities to easily jump between the ONOS model and protobuf model
- Starter Services
  - Implemented a selected set of services which expose Java-like API's to off-platform apps

Challenges

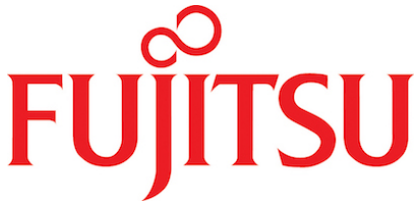
- Dynamic Registration of Services
  - Problem: when a new service is registered, the entire server needs to restart
  - Working to provide a setup where services can be gracefully added and removed without disturbance
- Handling of Large Datasets
  - Large dataset like topology info is currently sent in a large chunk → heavy load
- Stale Data & Caching
  - Problem: for large dataset, client risk performing calculations on stale data
  - Finding a way to provide notifications for dataset updates allows clients to maintain a local cache at reduced cost to the network

Roadmap

- Magpie Release
  - More services and supporting models
  - DEMO applications including sample client code
  - Registry service

- N Release
  - More services and supporting models
  - Create registry capable of dynamically updating the set of services
  - AAA to allow for the addition of administrative functionality to the current gRPC services
  - Stress testing and performance assessment

Acknowledgement



ONOS

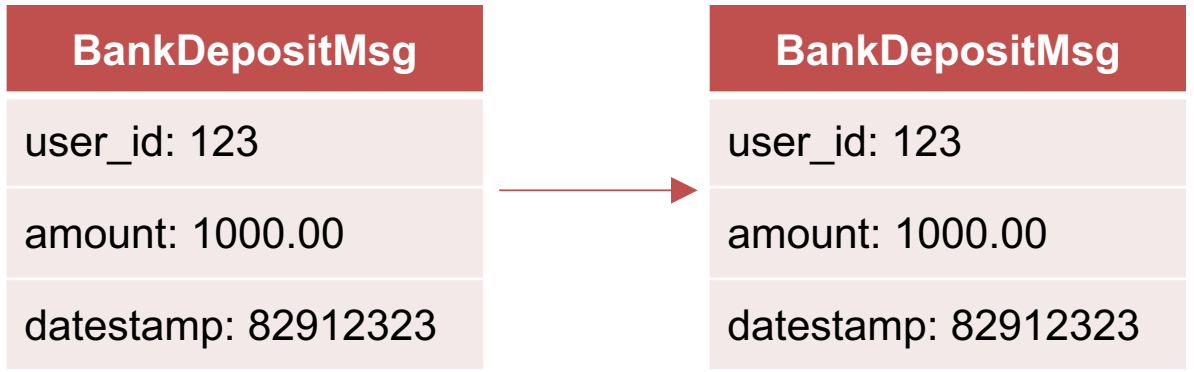
Open Network Operating System

http://onosproject.org/

Forward and Backward Compatibility (1/2)

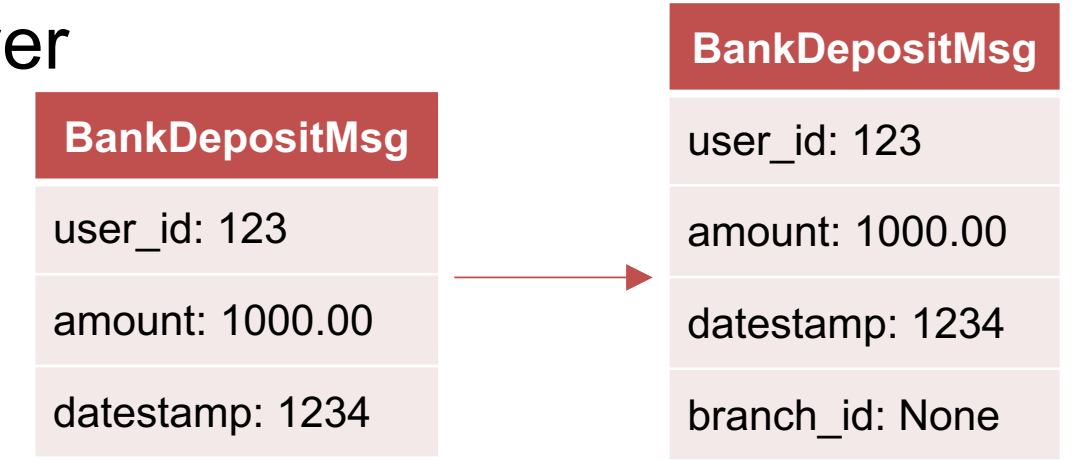
- Four Cases
  - Added field: old client ←→ new server
  - Removed field: old client ←→ new server
  - Added field: new client ←→ old server
  - Removed field: new client ←→ old server

- No Field Changes
  - Client sends a message to a server
  - Operated properly without any issue!



Forward and Backward Compatibility (2/2)

- Added Field: Old Client → New Server
- Removed Field: New Client → Old Server
  - Server recognizes the field is not set, and implements default behavior for Out-Of-Date requests



- Added Field: New Client → Old Server
- Removed Field: Old Client → New Server
  - The server simply ignores the newly added fields and processes as normal