# An investigation of the ZeroMQ asynchronous messaging library and its application to a client-server online model and an error message logger for the SwissFEL

Ambrish Rawat

# Overview

- ZeroMQ
- Google Protocol Buffer
- Accelerator Models
- Message Logger
- Performance
- Ensuring Reliability (Majordomo Model)
- Conclusions

# Overview

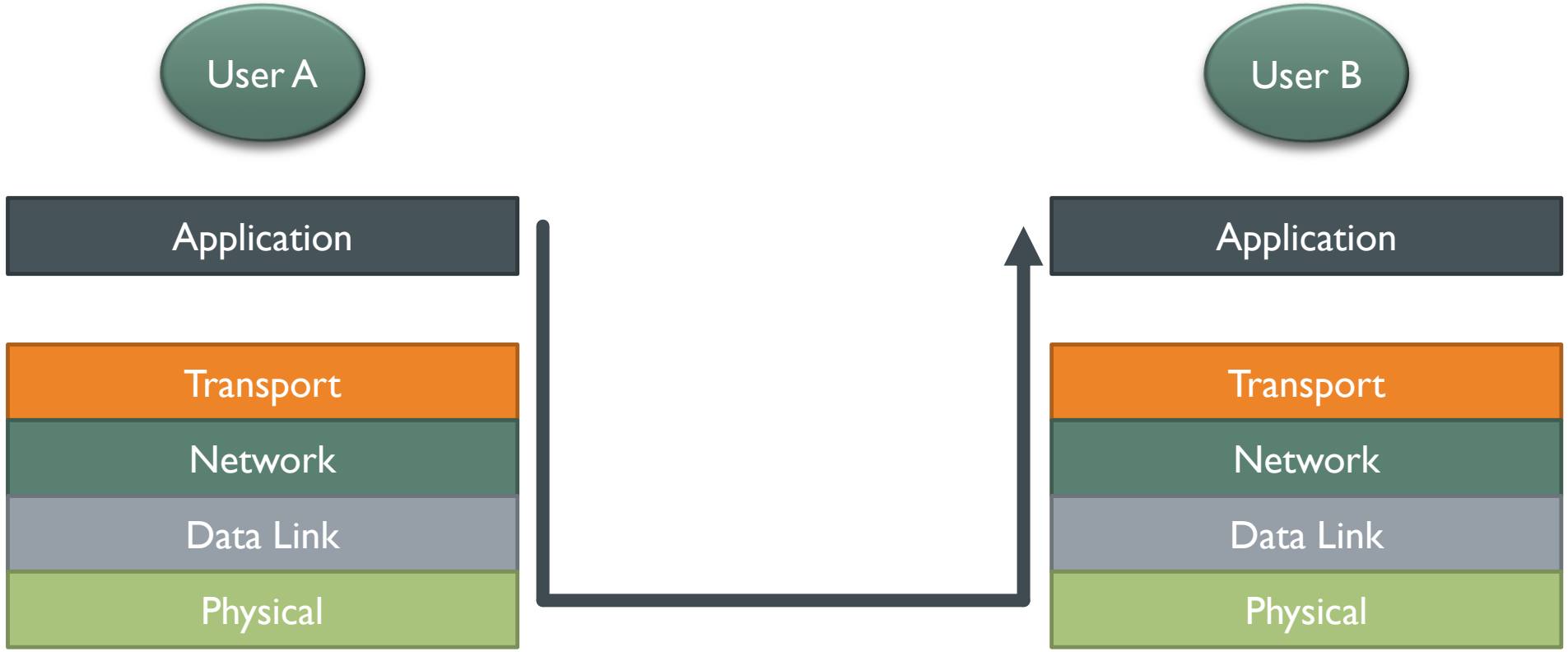
- ZeroMQ
- Google Protocol Buffer
- Accelerator Models
- Message Logger
- Performance
- Ensuring Reliability (Majordomo Model)
- Conclusions

# ZeroMQ

- Why ZeroMQ?
- What is ZeroMQ?
- Messaging Patterns

# Why ZeroMQ?

- Sending data across a network using basic principles



# What-if

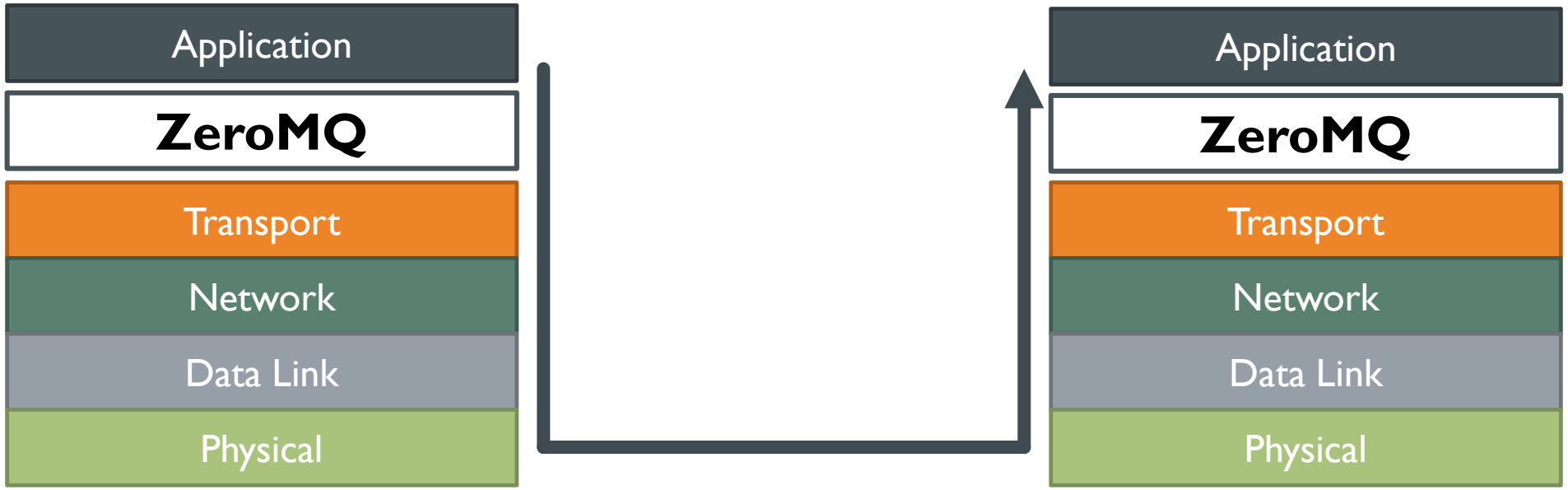
- How to handle dynamic components?
- How to represent a message on the wire?
- What about message that can't be delivered immediately (queues?)?

# What is ZeroMQ?

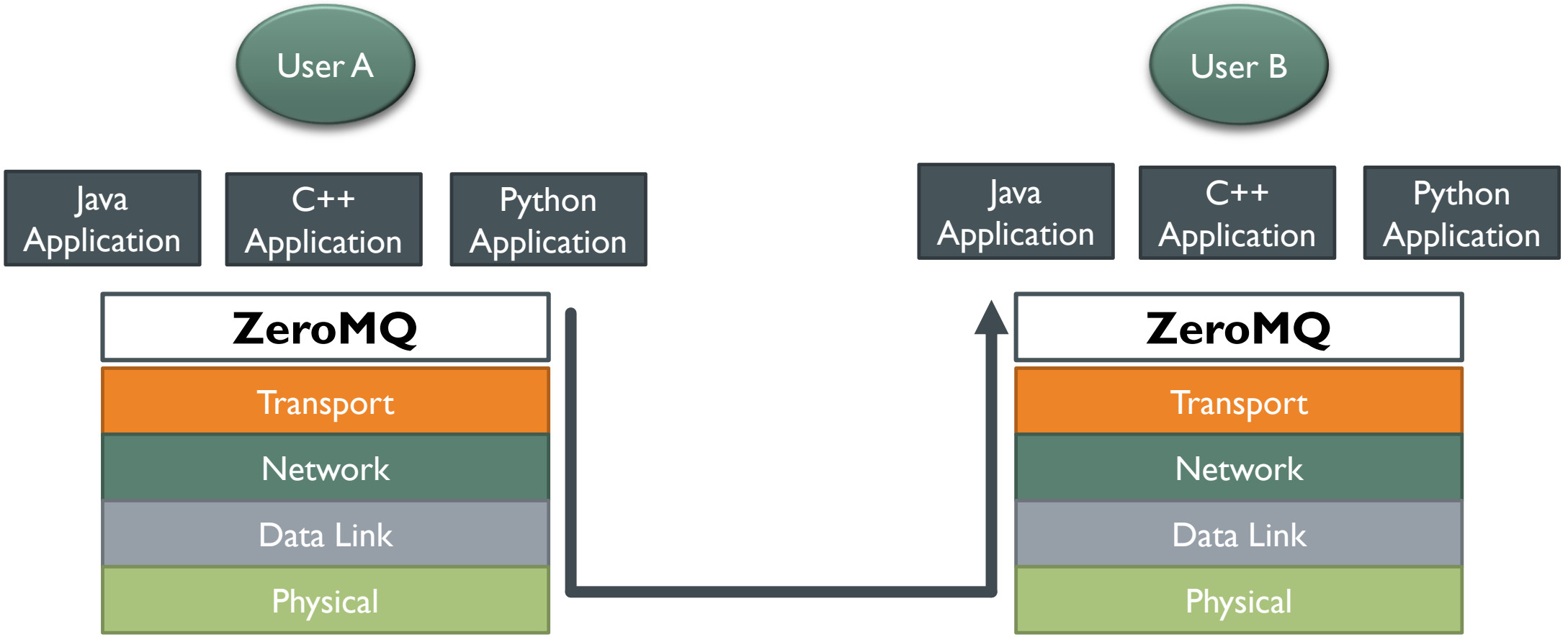
- Message-oriented-middleware (a library) and not a messaging server (an application)

# What is ZeroMQ?

- Message-oriented-middleware (a library) and not a messaging server (an application)



# What is ZeroMQ?



# What is ZeroMQ?

ZeroMQ:

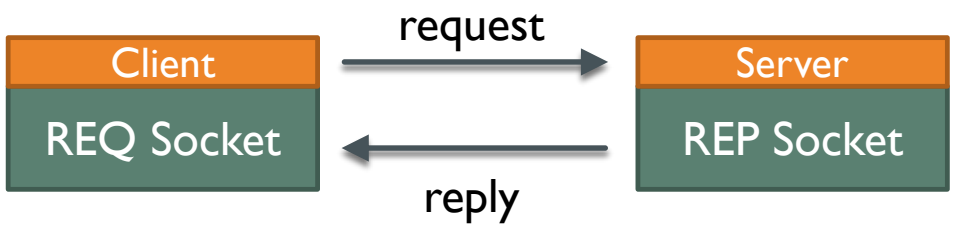
- is a C library with bindings for Java, C++, Python and many other languages
- facilitates easy application creation (invoking a library from in a program)
- sends strings across the wire
- supports multiple messaging patterns
- has framework for sending multipart messages
- automatically reconnects to peers as they come and go

# Messaging Patterns

- Request/Reply
- Publisher/Subscriber
- Push/Pull

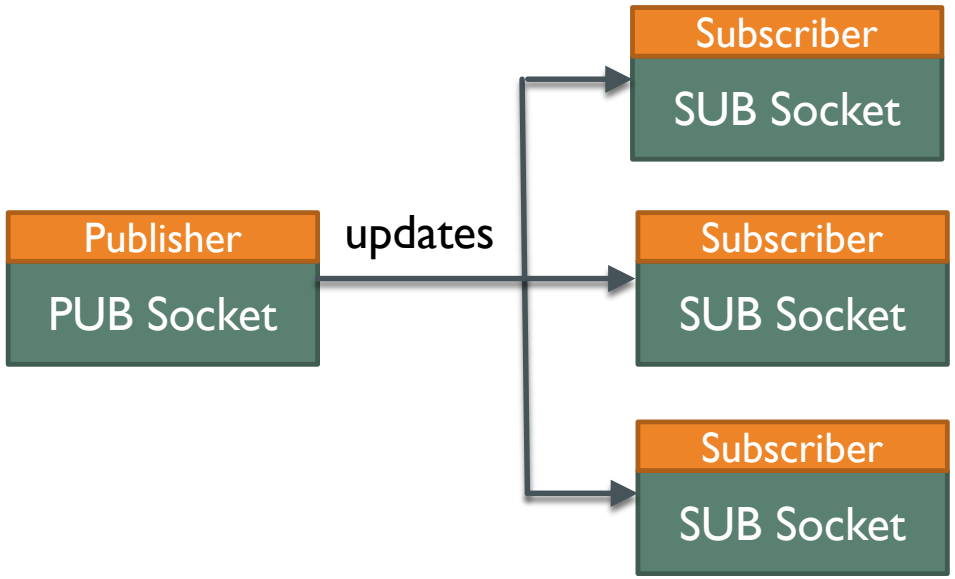
# Messaging Patterns

- Request/Reply
- Publisher/Subscriber
- Push/Pull



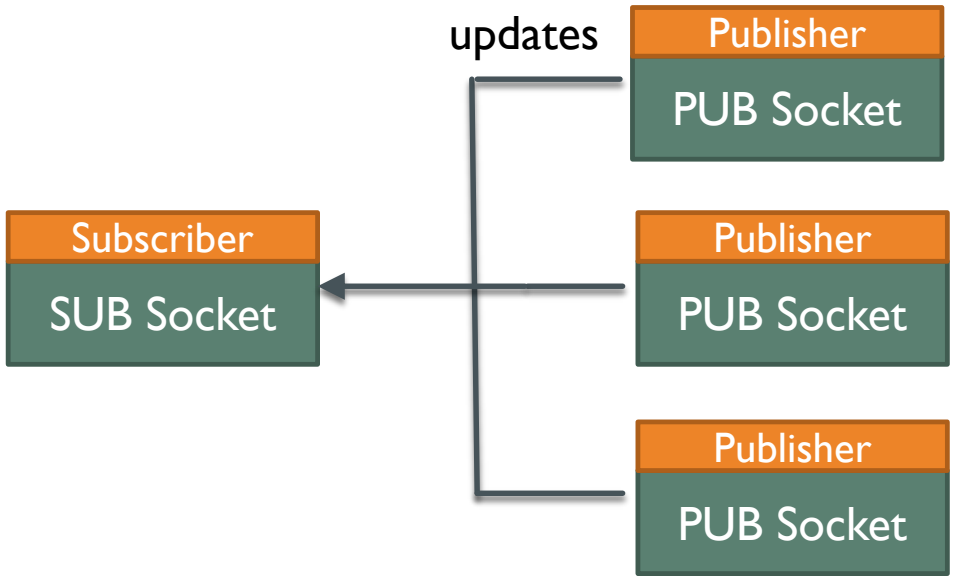
# Messaging Patterns

- Request/Reply
- Publisher/Subscriber
- Push/Pull



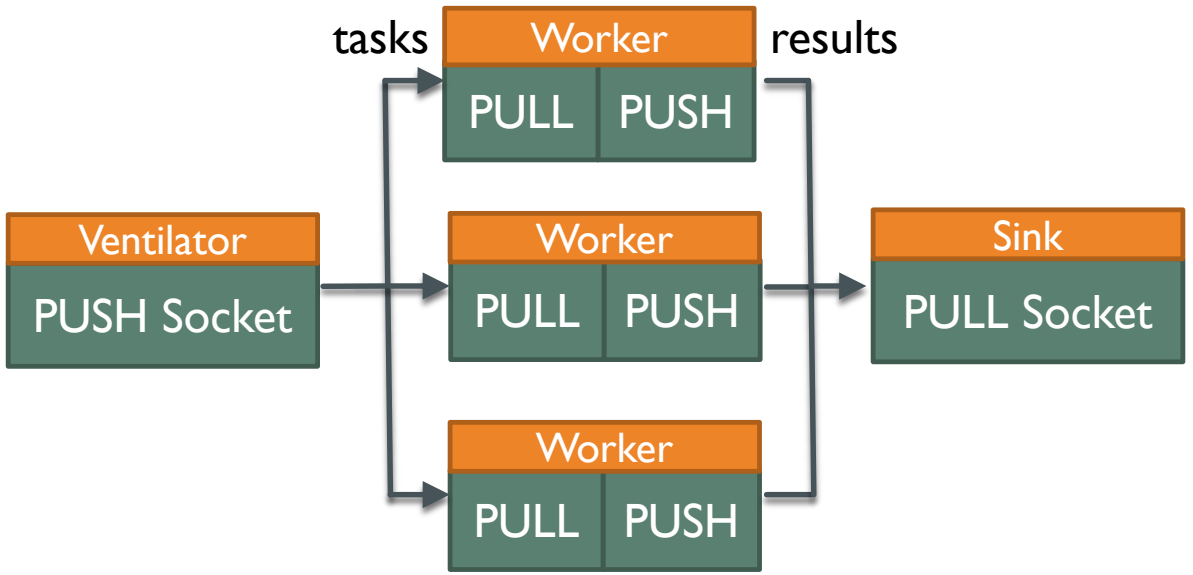
# Messaging Patterns

- Request/Reply
- Publisher/Subscriber
- Push/Pull



# Messaging Patterns

- Request/Reply
- Publisher/Subscriber
- Push/Pull



# Overview

- ZeroMQ
- Google Protocol Buffer
- Accelerator Models
- Message Logger
- Performance
- Majordomo Model
- Conclusions

# Google Protocol Buffer

Why Protocol Buffers?

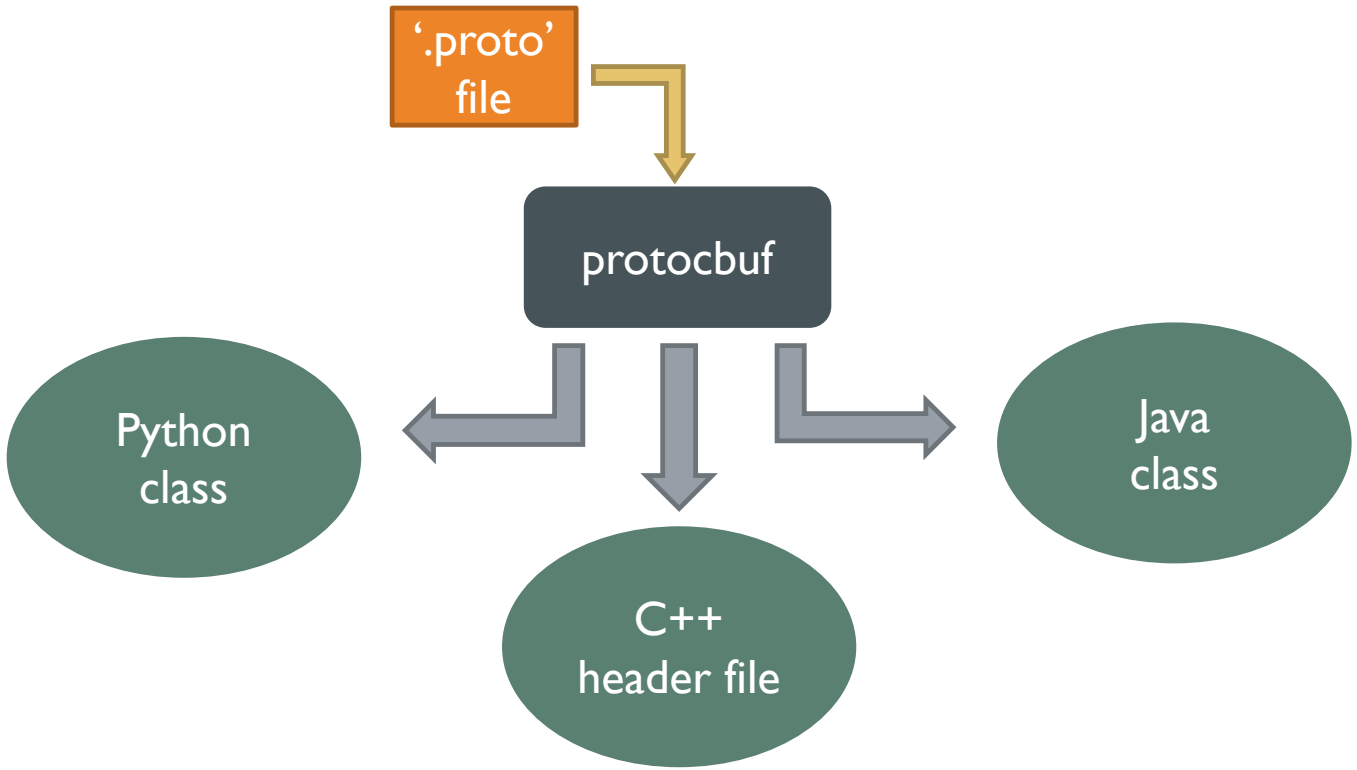
- There is a requirement for sending structured data
- ZeroMQ only handles strings

# Google Protocol Buffer

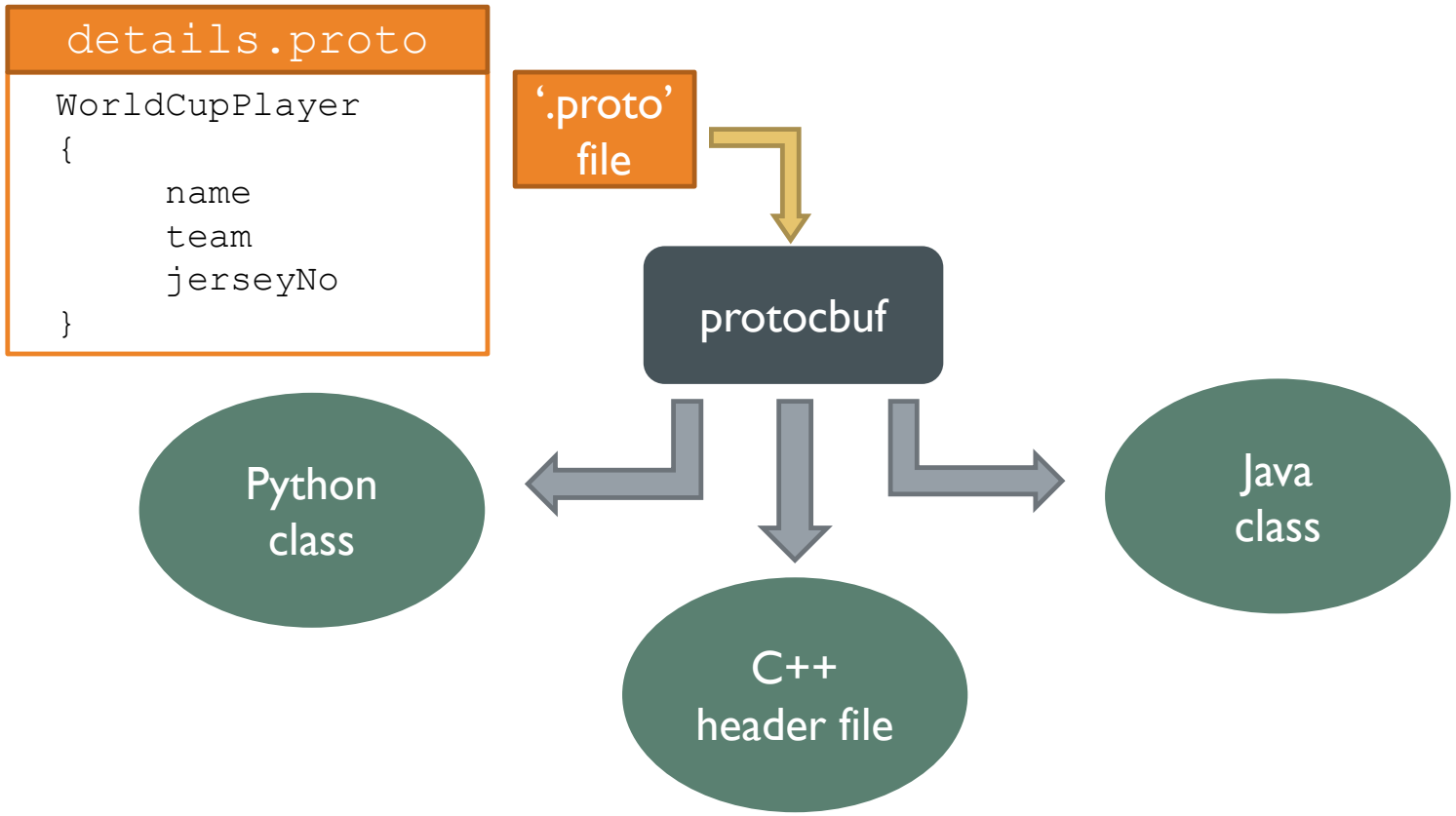
Google Protocol Buffer:

- provides efficient encoding of structured data
- has support for C++, Python and Java (language-neutral and platform-independent)

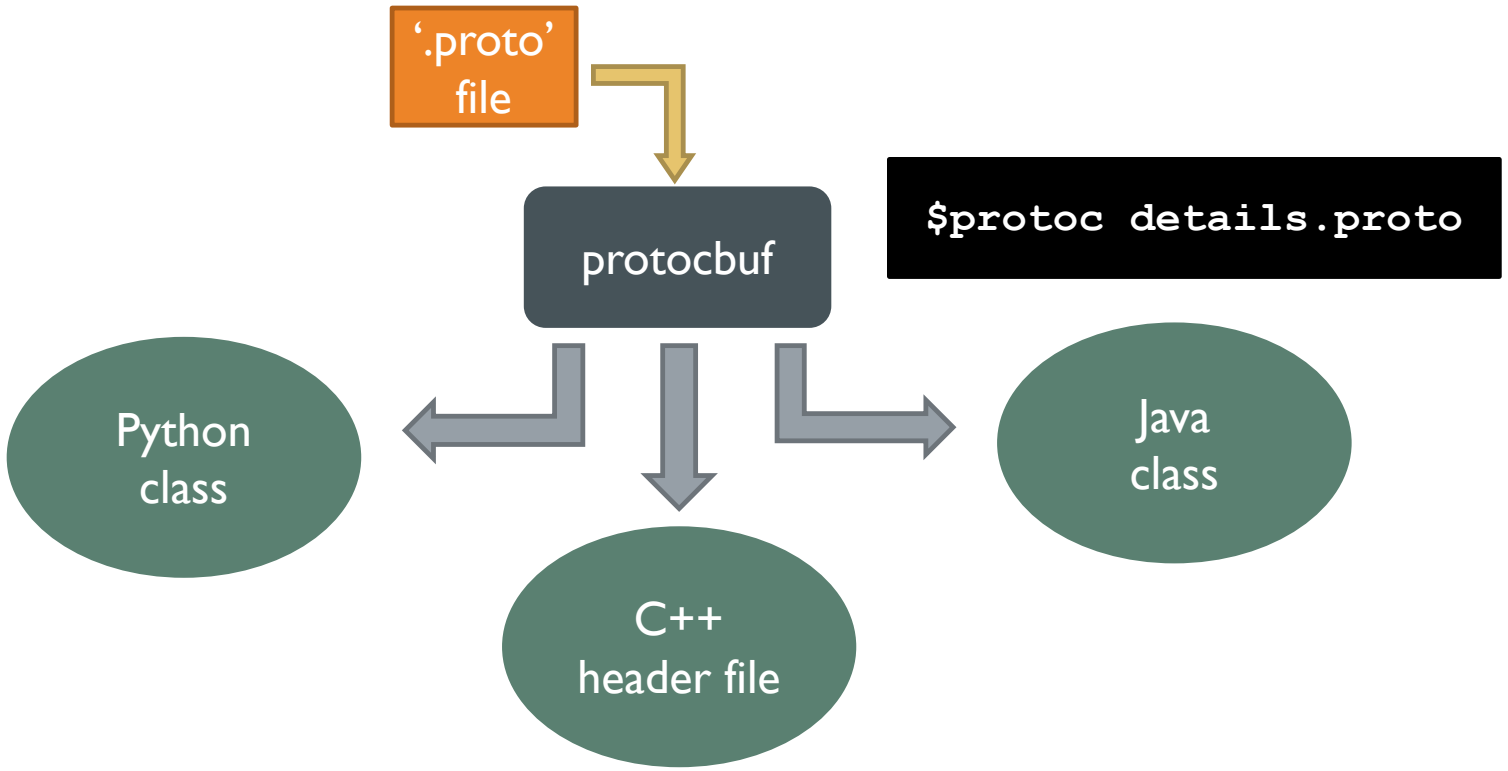
# Google Protocol Buffer



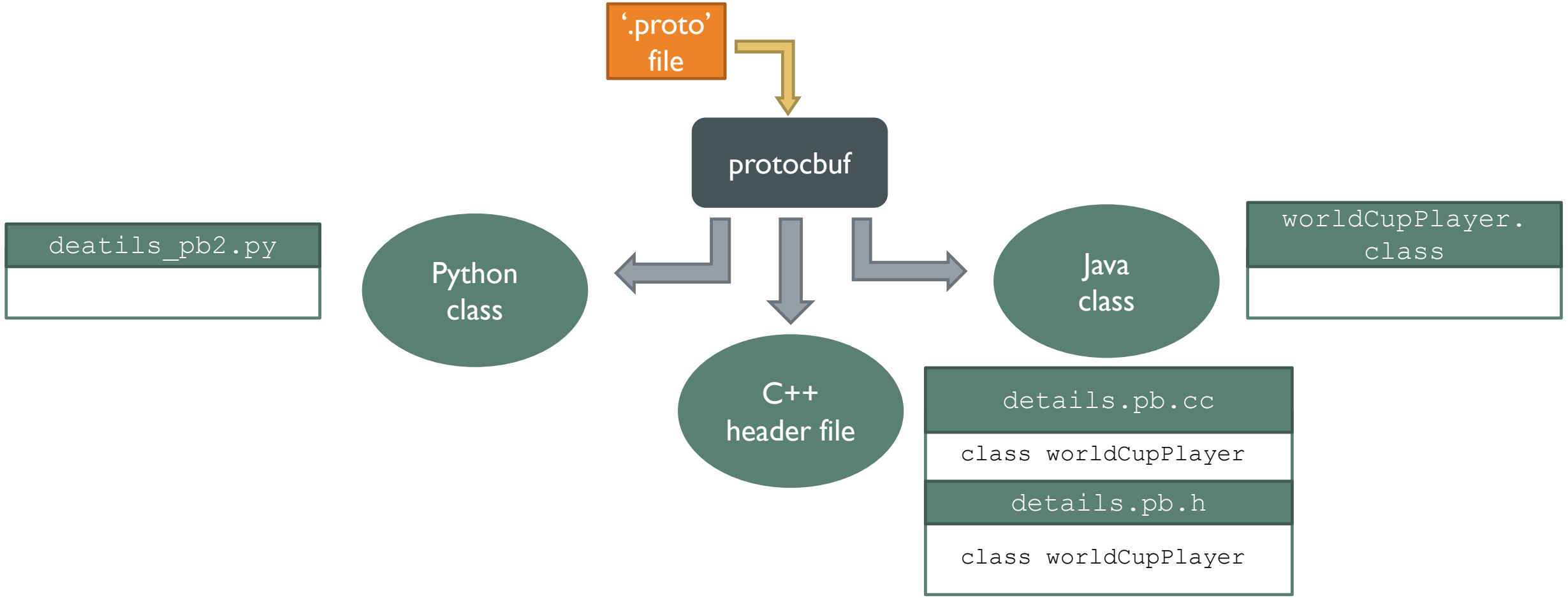
# Google Protocol Buffer



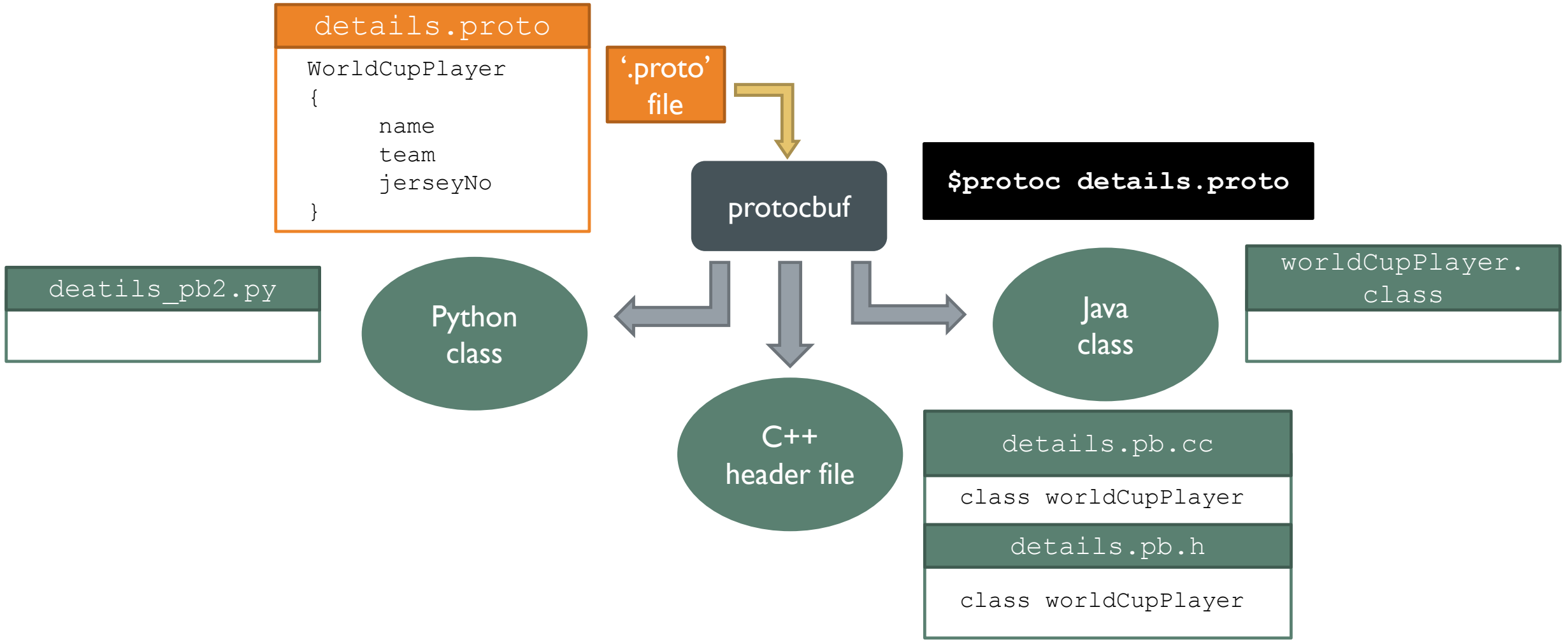
# Google Protocol Buffer



# Google Protocol Buffer



# Google Protocol Buffer



# Overview

- ZeroMQ
- Google Protocol Buffer
- Accelerator Models
- Message Logger
- Performance
- Majordomo Model
- Conclusions

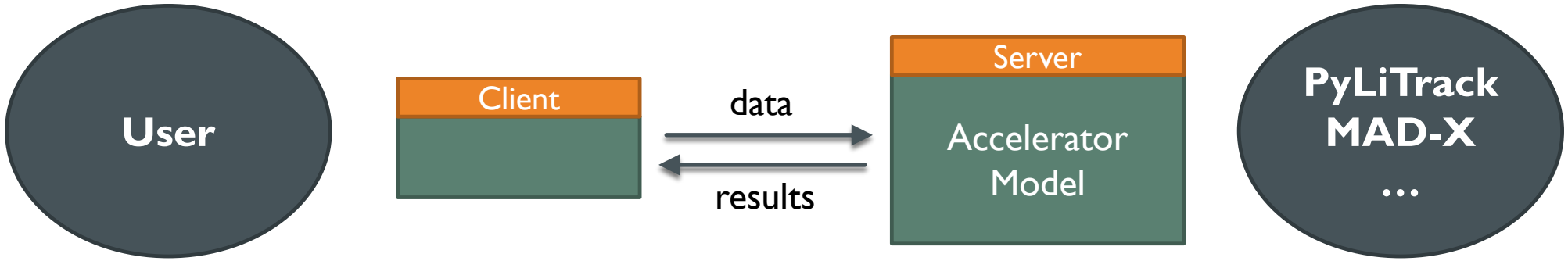
# Accelerator Models

- Objective
- Available Tools
- Proof of Concept
- Clients-side

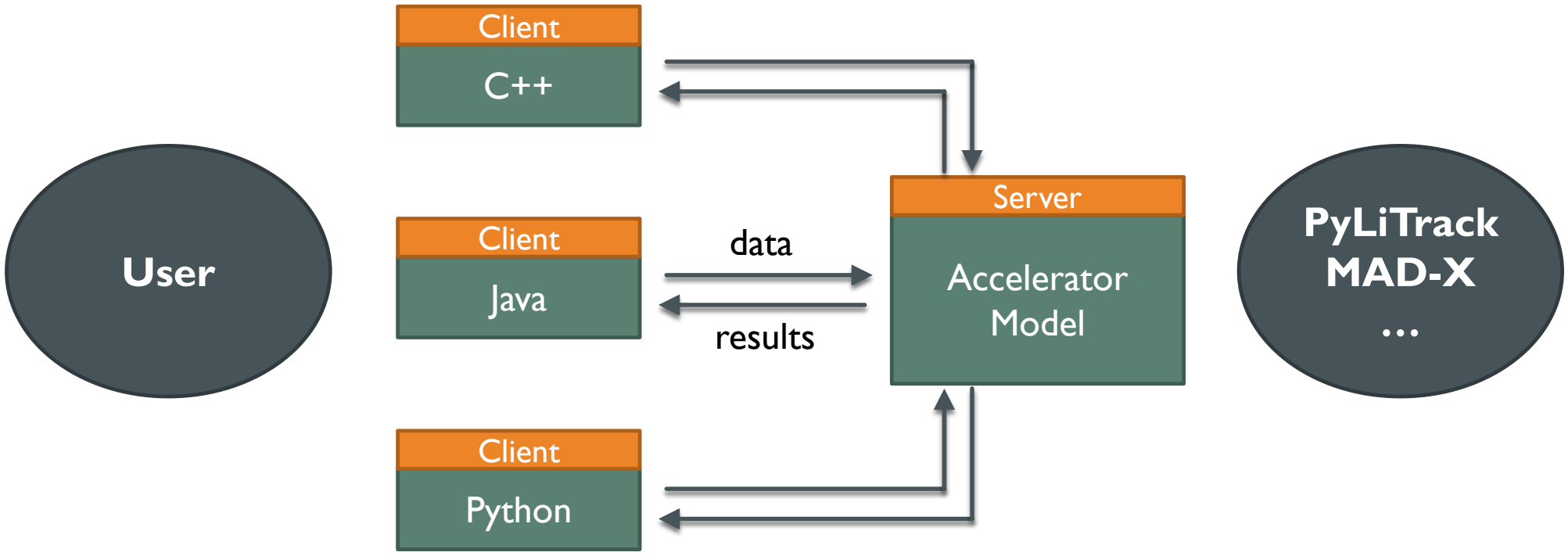
# Accelerator Models

- Objective
- Available Tools
- Proof of Concept
- Clients-side

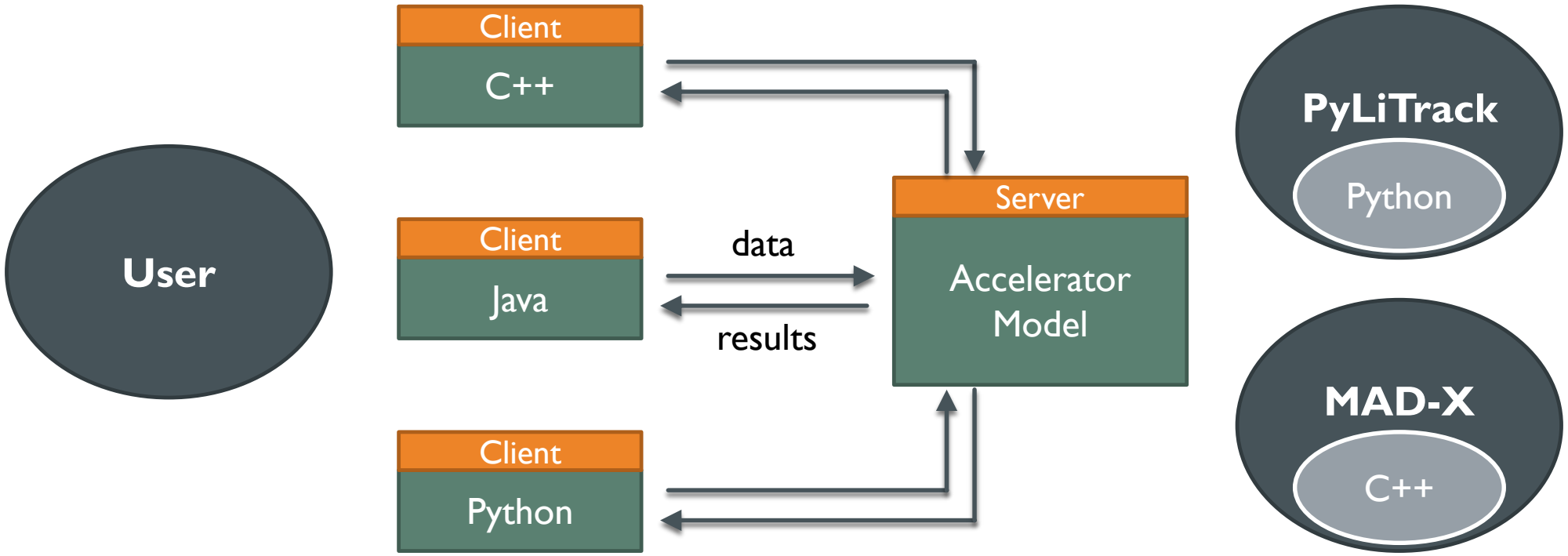
# Objective


User
Client
data
results
Server
Accelerator
Model
PyLiTrack
MAD-X
…

# Objective


User
Client
C++
Client
Java
Client
Python
data
results
Server
Accelerator
Model
PyLiTrack
MAD-X
...

# Objective


User
Client
C++
Client
Java
Client
Python
data
results
Server
Accelerator
Model
PyLiTrack
Python
MAD-X
C++

# Accelerator Models

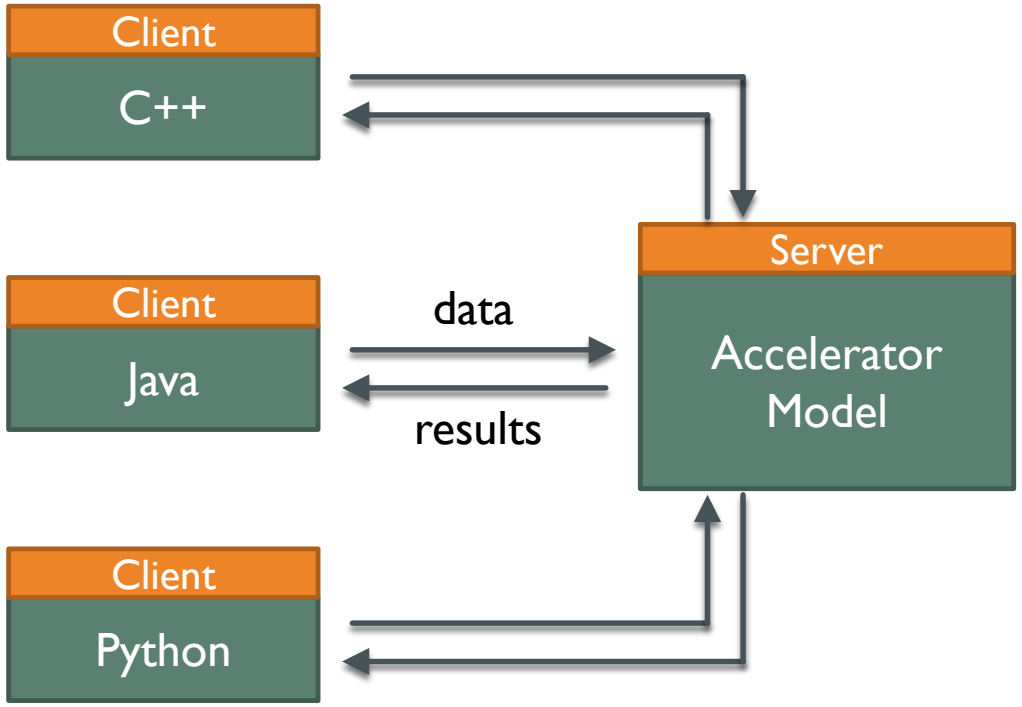
- Objective
- Available Tools
- Proof of Concept
- Clients-side

# Available Tools

**ZeroMQ (REQ/REP)**

**User**

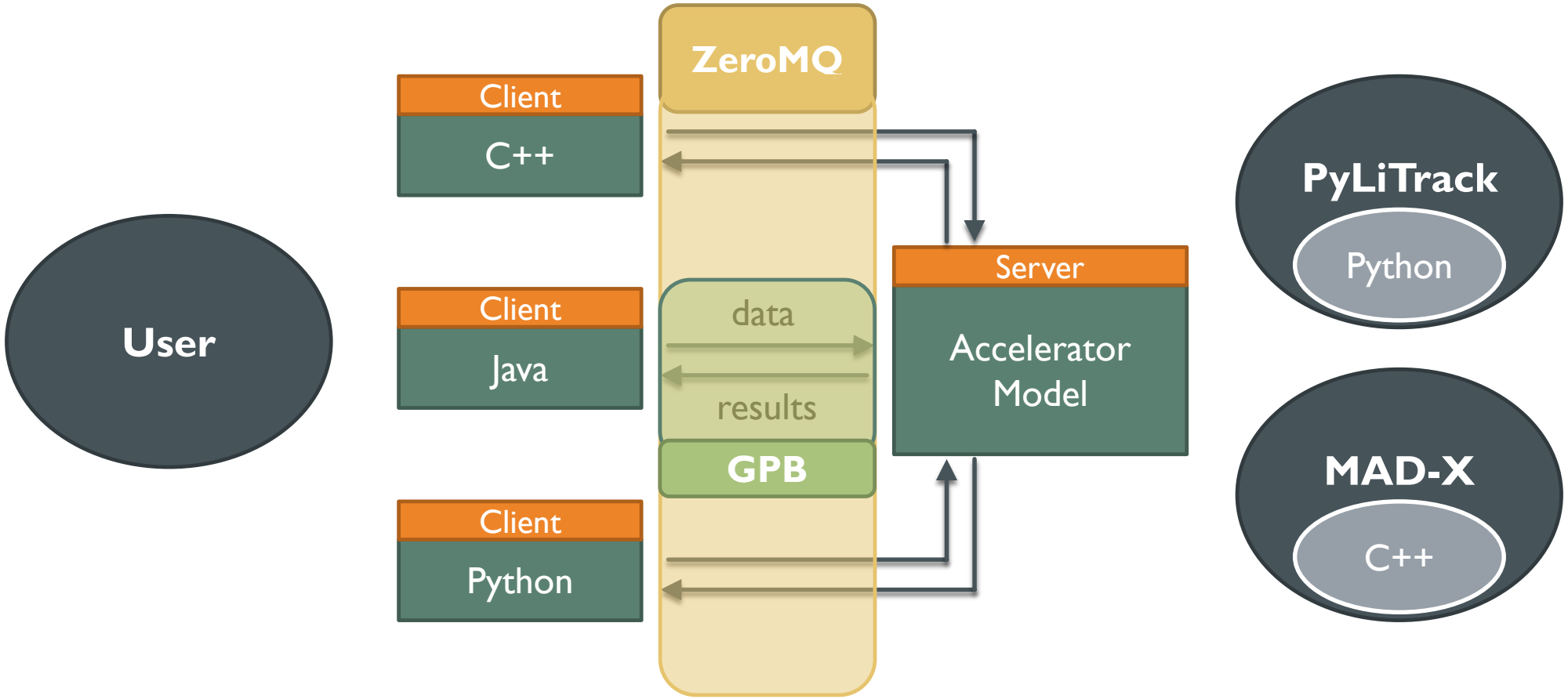
**Google Protocol Buffer**




**PyLiTrack**
Python

**MAD-X**
C++

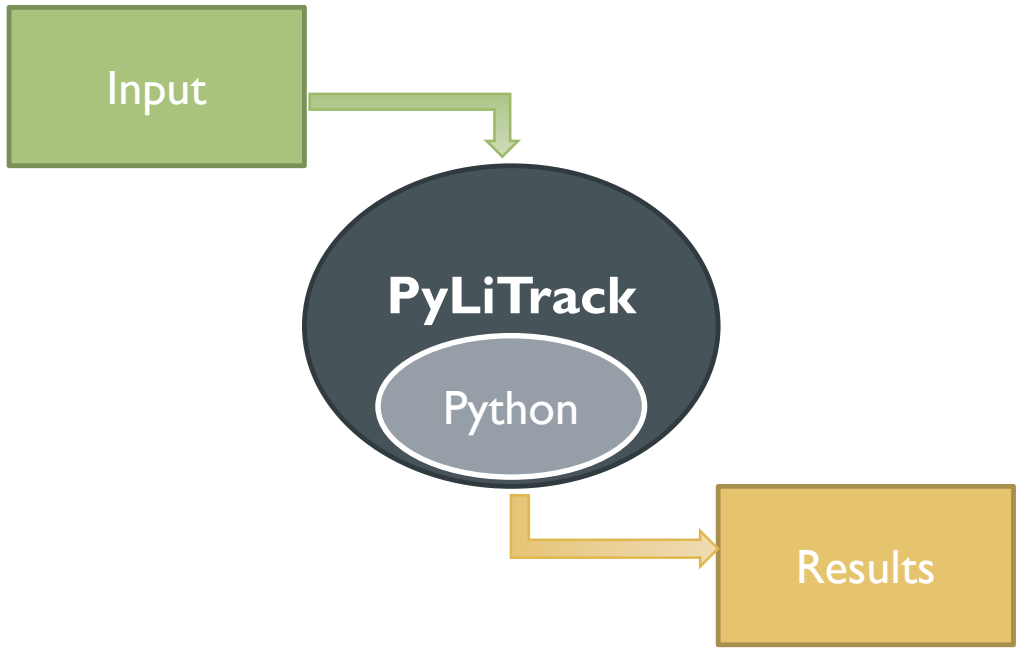
# Available Tools

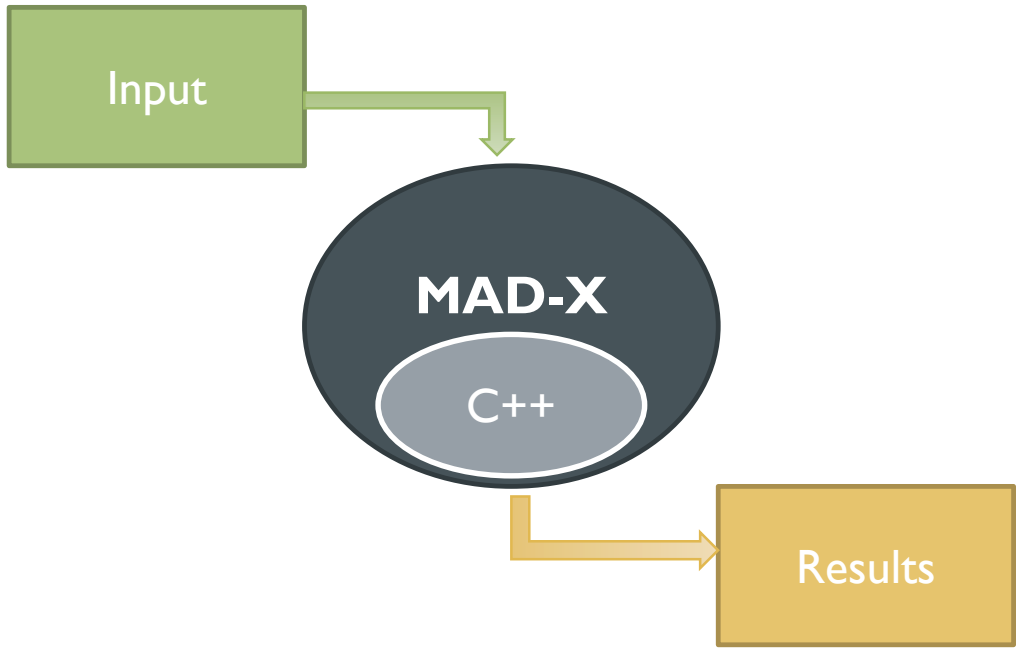


# Accelerator Models

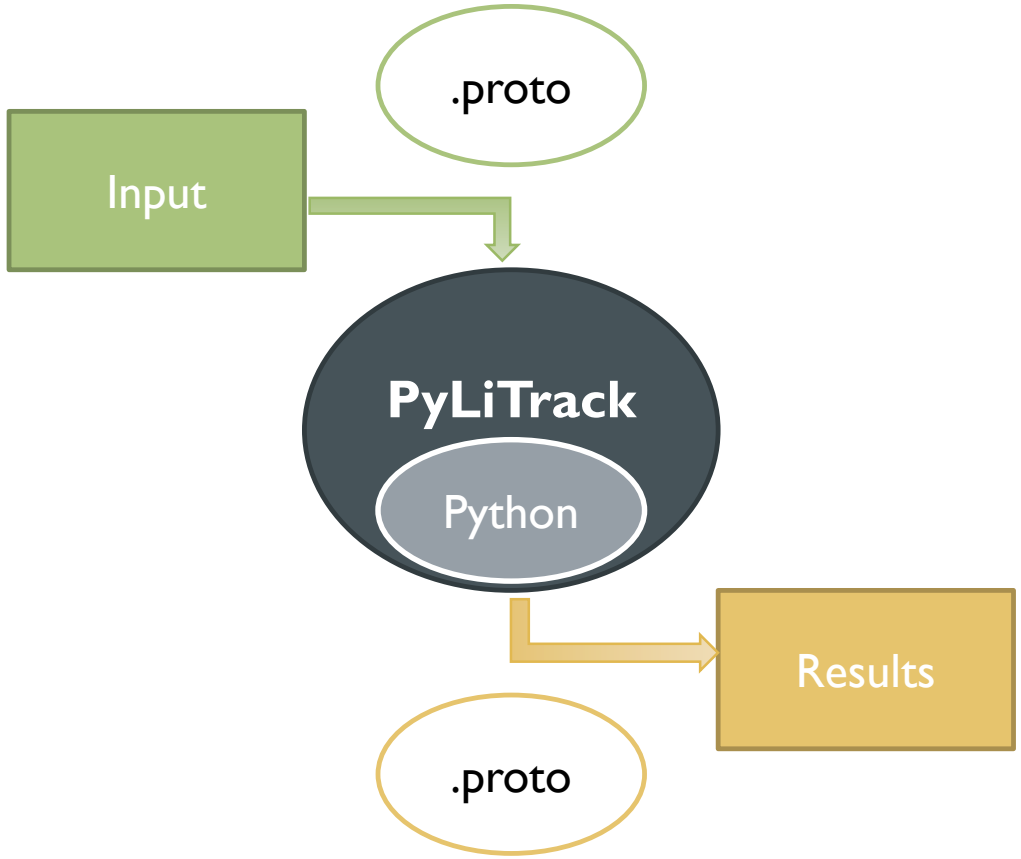
- Objective
- Available Tools
- Proof of Concept
- Clients-side

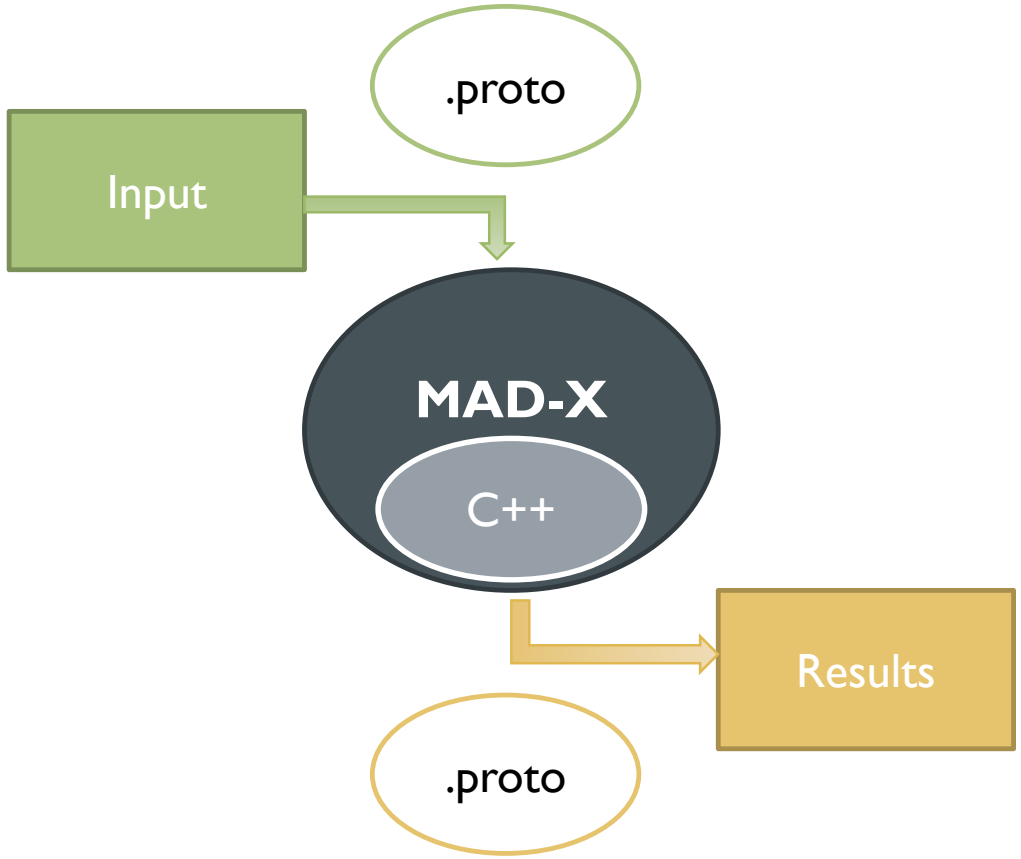
# Proof of Concept



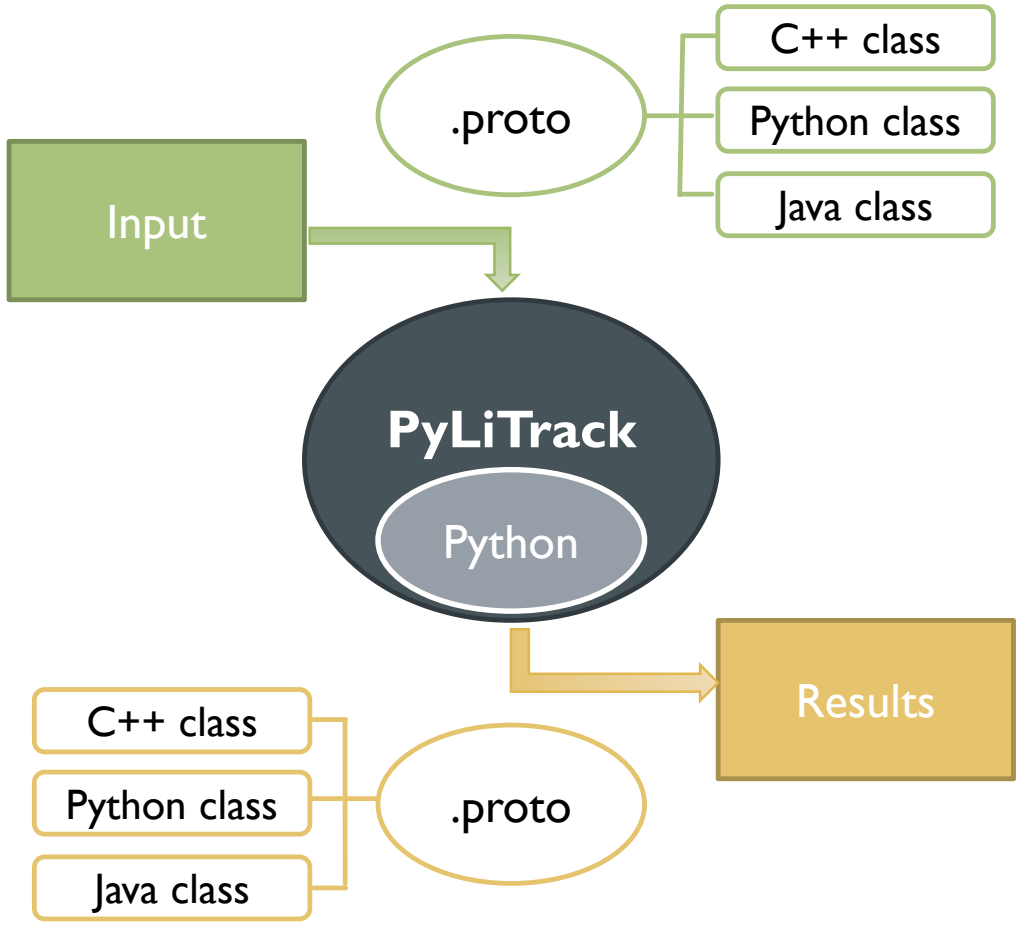



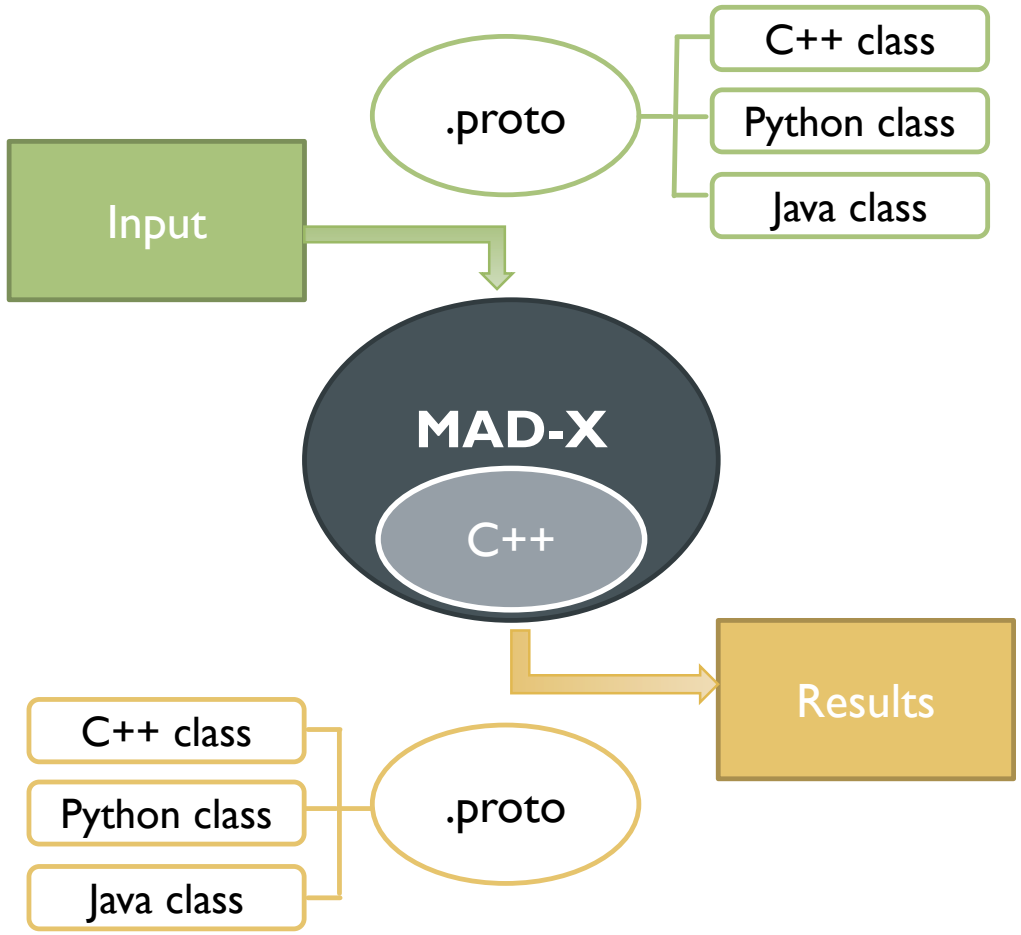
# Proof of Concept

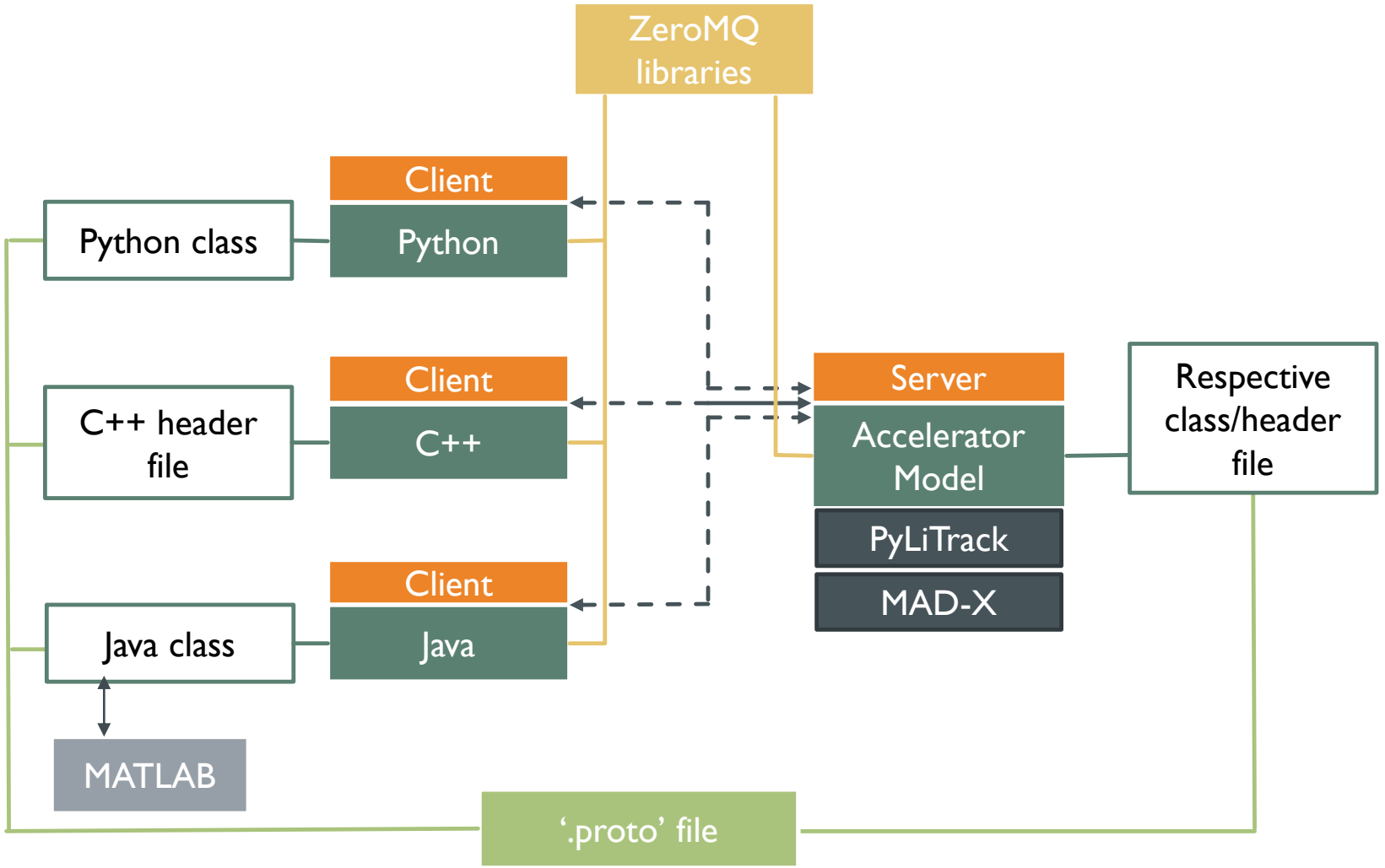


# Proof of Concept




ZeroMQ libraries
Client
Python
Python class
Client
C++
C++ header file
Client
Java
Java class
MATLAB
Server
Accelerator Model
PyLiTrack
MAD-X
Respective class/header file
'.proto' file

# Accelerator Models

- Objective
- Available Tools
- Proof of Concept
- Clients-side

# What is achieved

- An accelerator model for your language of choice
- C++, Python and Java

# PyLiTrack

pilyformat.proto

Input

```
message PBinput {

        repeated string filename = 1;

        message Arrayelement {
                repeated float floatlist = 1;
                repeated int32 intlist = 2;
                required string strtag = 3;
        }

        repeated Arrayelement beamlinedata = 2;

}
```

Output

```
message PBoc {

        message Arrayelement {
                repeated float floatlist = 1;
                repeated int32 intlist = 2;
                required string strtag = 3;
        }

        message floatlist {
                repeated float ele = 1;
        }

        repeated Arrayelement PBBLj = 1;
        repeated floatlist PBzposj = 2;
        repeated floatlist PBdE_Ej = 3;
        repeated float PBEbarj = 4;
        repeated float PBz_barj = 5;
        repeated float PBZFWmmj = 6;
        repeated float PBsigzGj = 7;
        repeated float PBdFWpctj = 8;
        repeated float PBEbarcutsj = 9;
        repeated float PBsigEGj = 10;
        repeated float PBI_pkj = 11;
        repeated float PBI_pkfj = 12;
        repeated float PBfcutj = 13;

}
```

# C++ Client – PyLiTrack

- Connect to the PyLiTrack server

```
zmq::context_t context (1);
zmq::socket_t socket (context, ZMQ_REQ);
std::cout << "Connecting to server..." << std::endl;
socket.connect ("tcp://129.129.145.206:5555");
```

- Input data format – include the respective header file or import the respective class

```
#include "pyliformat.pb.h"
```

```
pily::PBinput in_1;
in_1.add_filename("/afs/psi.ch/project/zeromq/deps/PyLiTrack/SF_Sband.dat");
in_1.add_filename("/afs/psi.ch/project/zeromq/deps/PyLiTrack/SF_Cband.dat");
```

- Send serialized string

```
string in_str;
in_1.SerializeToString(&in_str)
int size_string = in_str.length();
zmq::message_t msg (size_string);
memcpy ((void *) msg.data (), in_str.c_str(), size_string);
std::cout << "Sending data for processing "<<std::endl;
socket.send (msg);
```

- Receive track results

```
zmq::message_t reply;
socket.recv (&reply);
std::string rpl = std::string(static_cast<char*>(reply.data()), reply.size());
pily::PBoc out_1 = ParseFromString(rpl);
```

# Overview

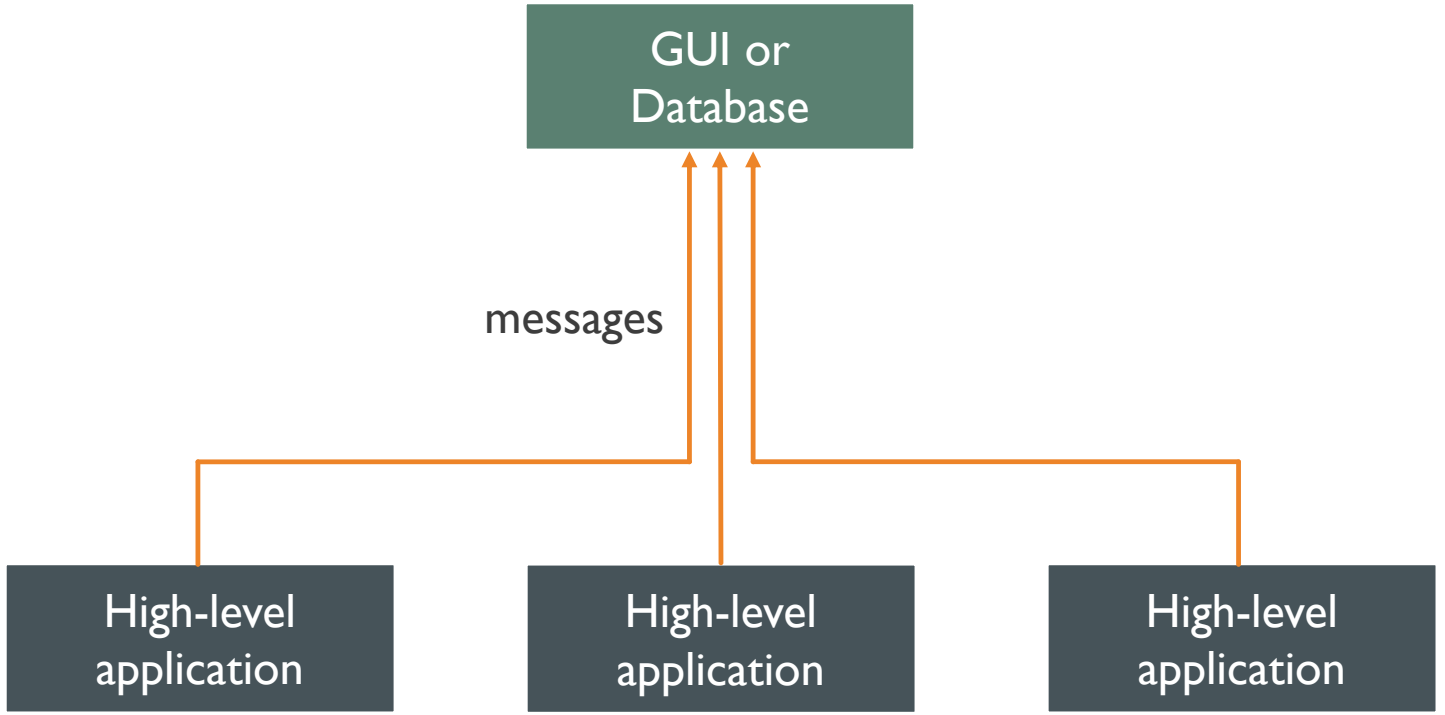
- ZeroMQ
- Google Protocol Buffer
- Accelerator Models
- Message Logger
- Performance
- Ensuring Reliability (Majordomo Model)
- Conclusions

# Message Logger

- Objective
- Log Messages
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept

# Message Logger

- Objective
- Log Messages
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept



# Message Logger

- Objective
- Log Messages
  - Adopted RFC 5424 (syslog protocol)
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept

```
<34>1 2014-07-04T14:15.003Z gfalc6064 4R AYT RYE - Sample
```

# Message Logger

- Objective
- Log Messages
  - Adopted RFC 5424 (syslog protocol)
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept

```
<34>1 2014-07-04T14:15.003Z gfalc6064 4R AYT RYE - Sample
```

Priority value

# Message Logger

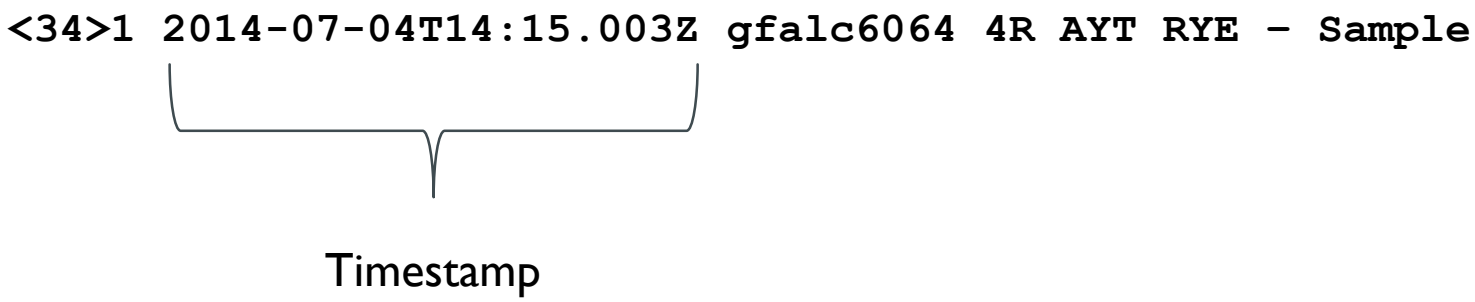
- Objective
- Log Messages
  - Adopted RFC 5424 (syslog protocol)
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept

```
<34>1 2014-07-04T14:15.003Z gfalc6064 4R AYT RYE - Sample
```

Version

# Message Logger

- Objective
- Log Messages
  - Adopted RFC 5424 (syslog protocol)
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept



# Message Logger

- Objective
- Log Messages
  - Adopted RFC 5424 (syslog protocol)
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept

```
<34>1 2014-07-04T14:15.003Z gfalc6064 4R AYT RYE - Sample
```

Hostname

# Message Logger

- Objective
- Log Messages
  - Adopted RFC 5424 (syslog protocol)
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept

```
<34>1 2014-07-04T14:15.003Z gfalc6064 4R AYT RYE - Sample
```

Application name

# Message Logger

- Objective
- Log Messages
  - Adopted RFC 5424 (syslog protocol)
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept

```
<34>1 2014-07-04T14:15.003Z gfalc6064 4R AYT RYE - Sample
```

Process ID

# Message Logger

- Objective
- Log Messages
  - Adopted RFC 5424 (syslog protocol)
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept

```
<34>1 2014-07-04T14:15.003Z gfalc6064 4R AYT RYE - Sample
```

Message ID

# Message Logger

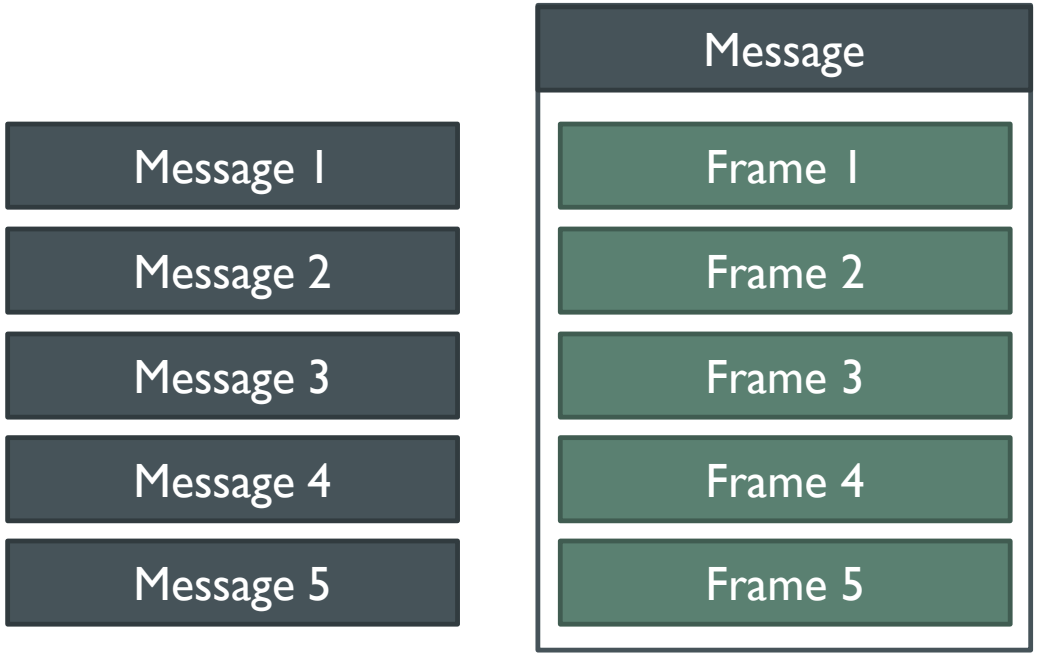
- Objective
- Log Messages
  - Adopted RFC 5424 (syslog protocol)
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept

```
<34>1 2014-07-04T14:15.003Z gfalc6064 4R AYT RYE - Sample
```

Message

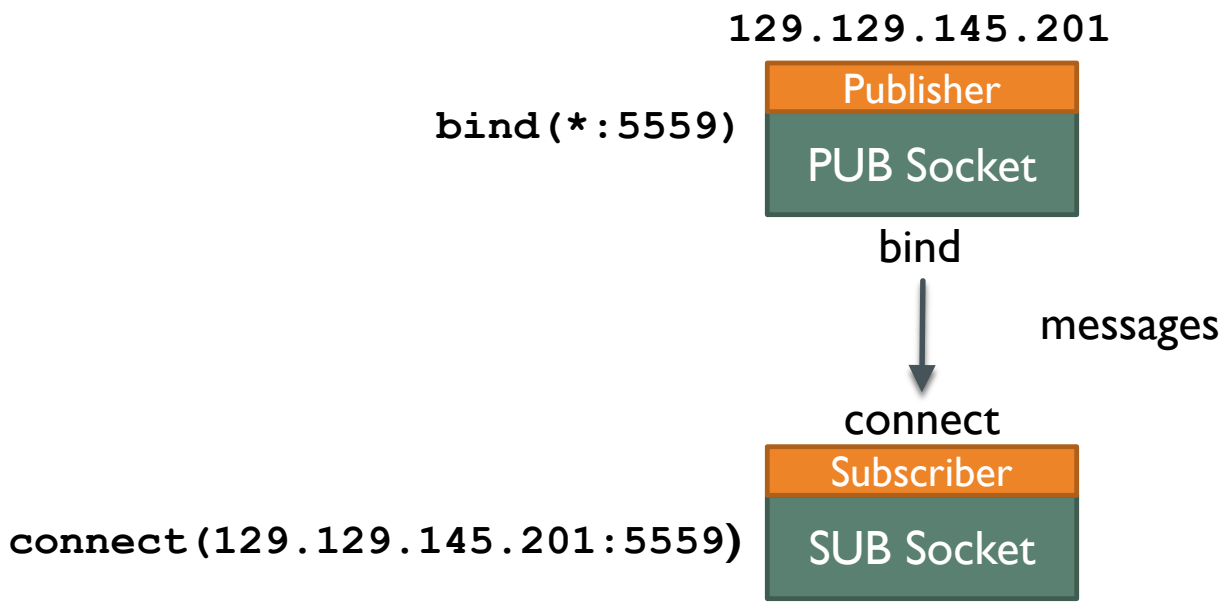
# Message Logger

- Objective
- Log Messages
- Multi-part Messages
  - Several frames in one message
  - No need for protocol buffer
  - Simple serialization (saves processing time)
  - All frames or no frames
- Pub/Sub and Proxy



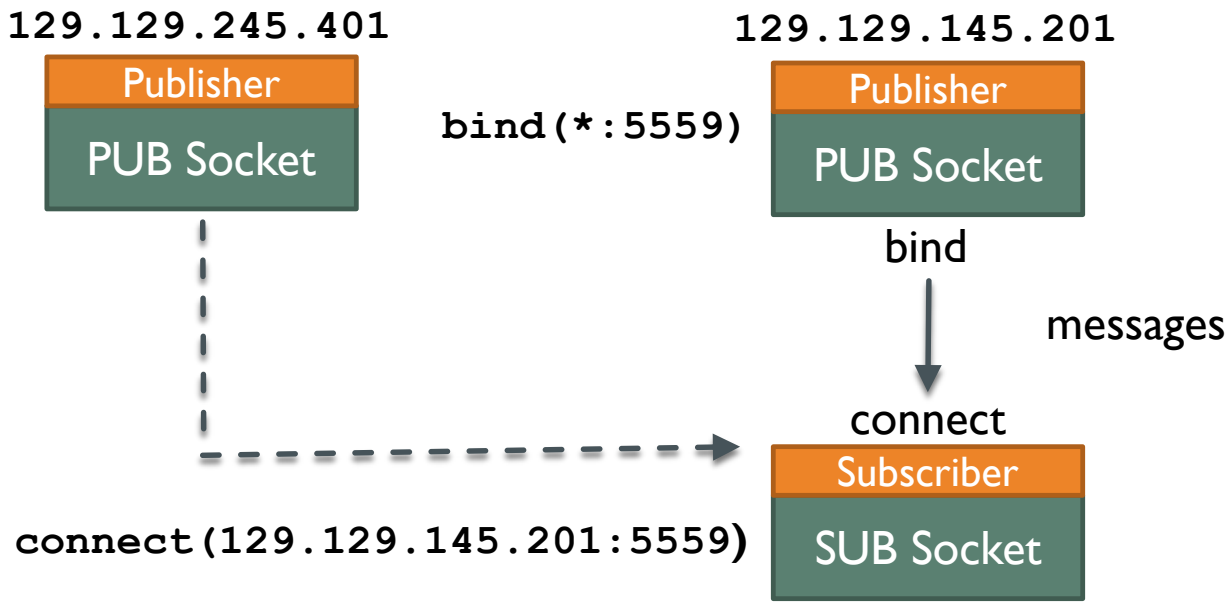
# Message Logger

- Objective
- Log Messages
- Multi-part Messages
- Pub/Sub and Proxy
  - Traditional Pub/Sub Architecture
  - The Dynamic Discovery Problem
  - Proxy
- Proof of Concept



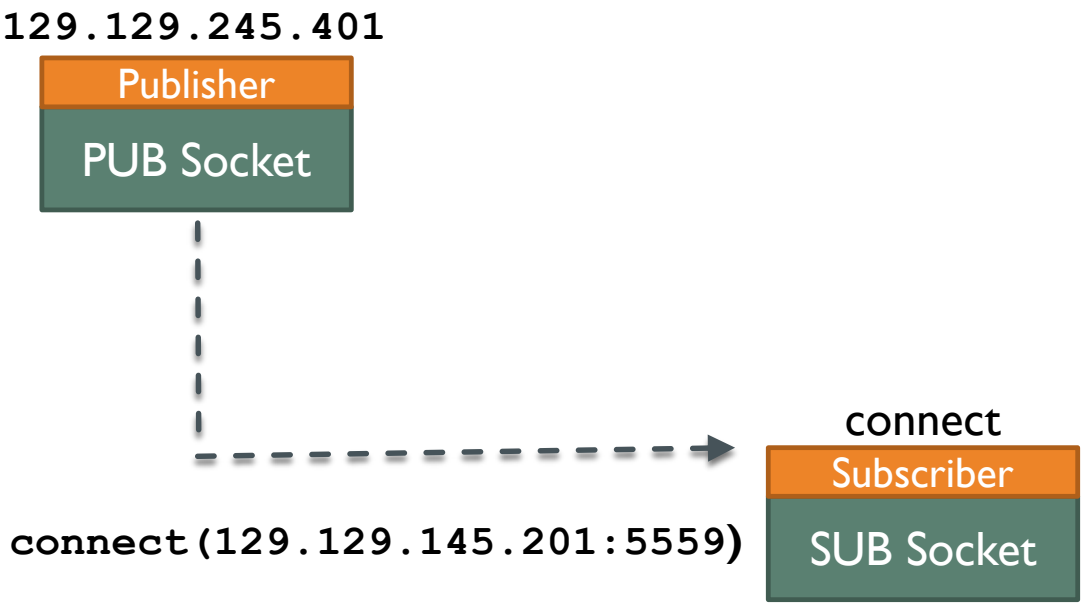
# Message Logger

- Objective
- Log Messages
- Multi-part Messages
- Pub/Sub and Proxy
  - Traditional Pub/Sub Architecture
  - The Dynamic Discovery Problem
  - Proxy
- Proof of Concept



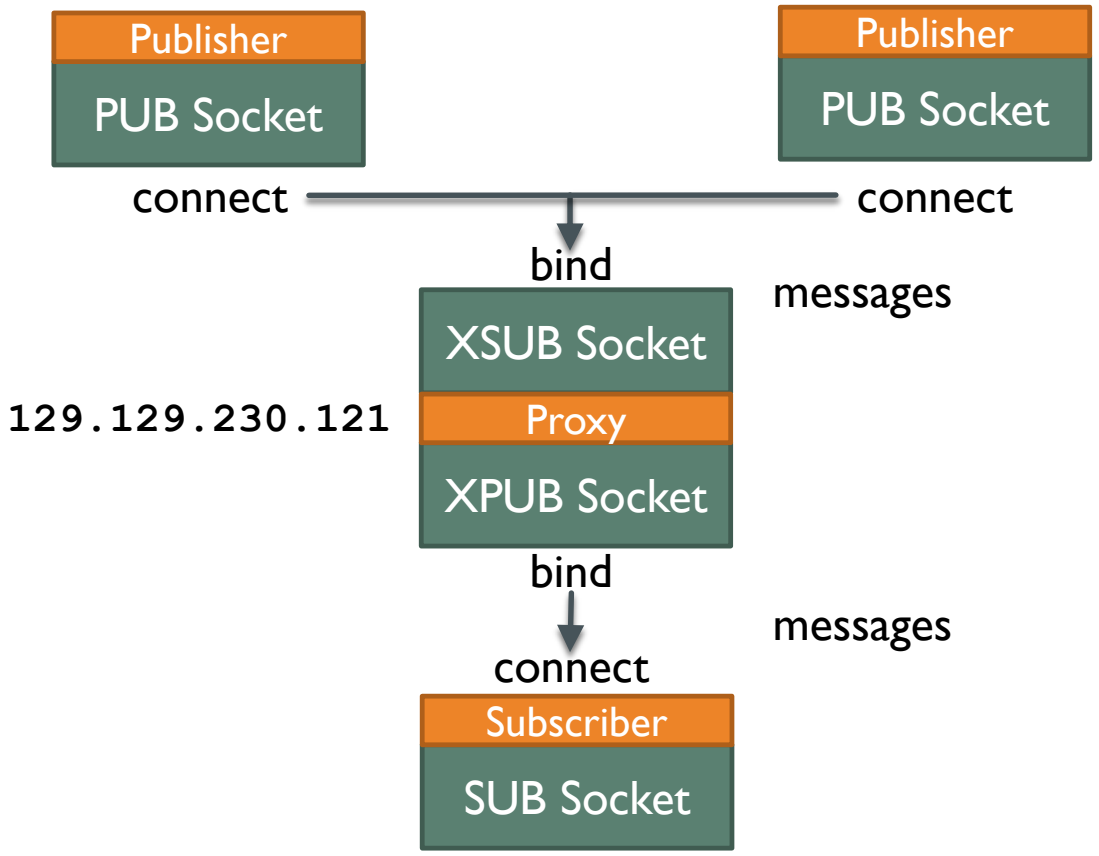
# Message Logger

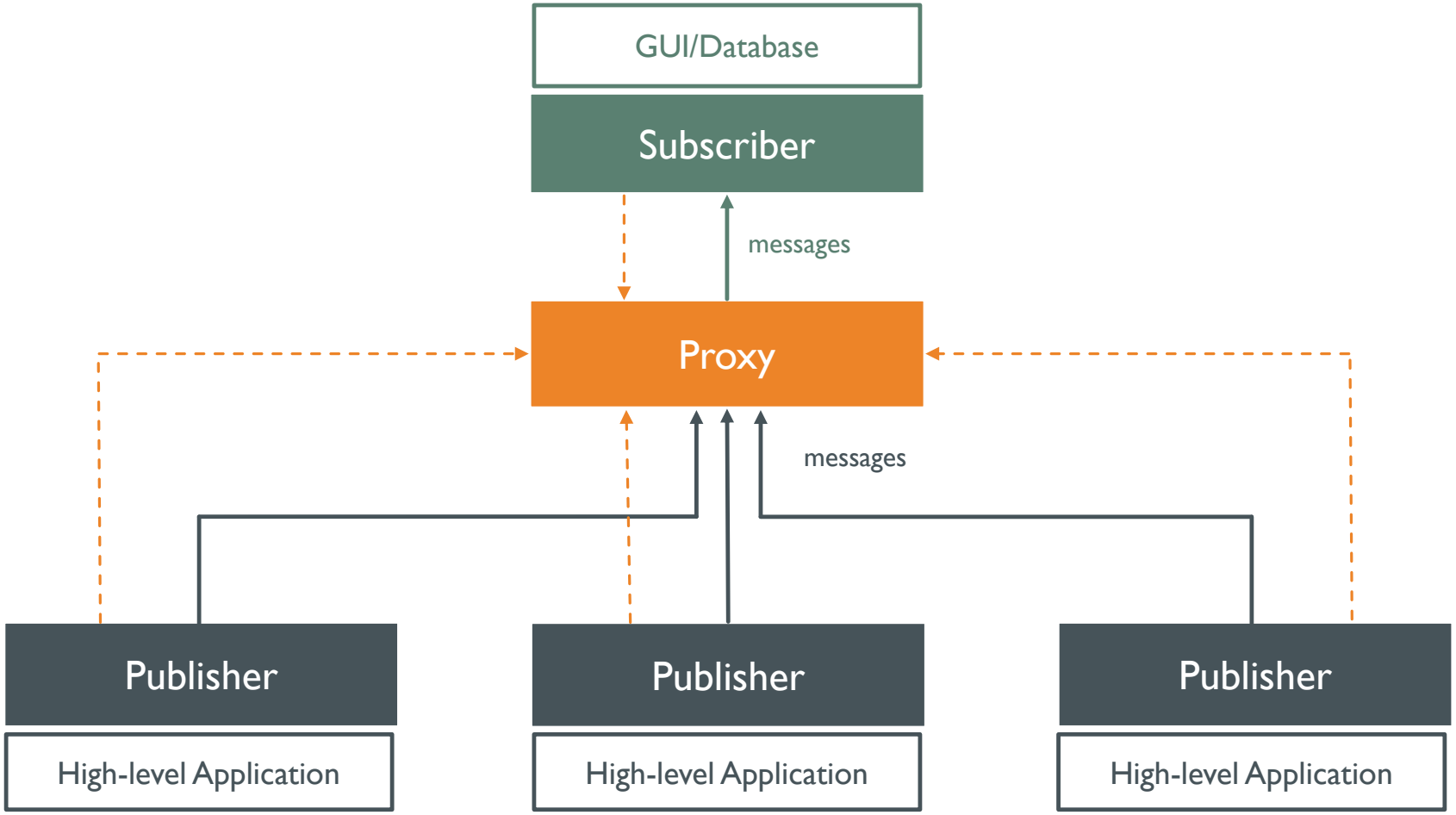
- Objective
- Log Messages
- Multi-part Messages
- Pub/Sub and Proxy
  - Traditional Pub/Sub Architecture
  - The Dynamic Discovery Problem
  - Proxy
- Proof of Concept



# Message Logger

- Objective
- Log Messages
- Multi-part Messages
- Pub/Sub and Proxy
  - Traditional Pub/Sub Architecture
  - The Dynamic Discovery Problem
  - Proxy
- Proof of Concept

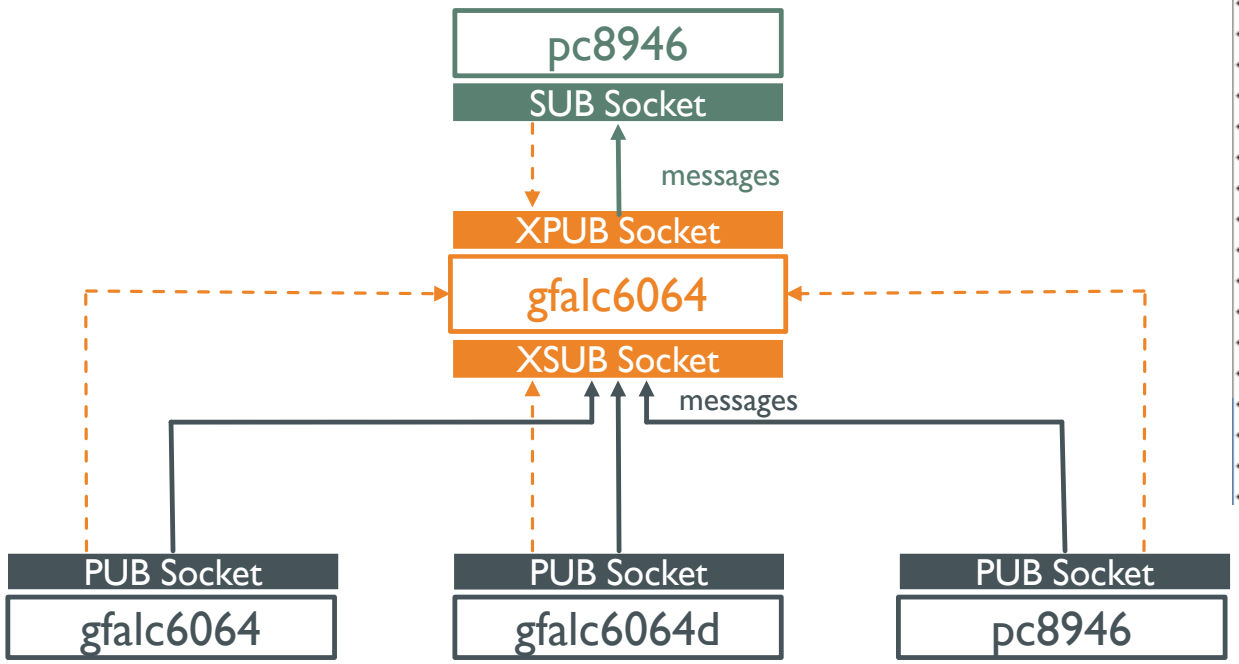



GUI/Database
Subscriber
messages
Proxy
messages
Publisher
Publisher
Publisher
High-level Application
High-level Application
High-level Application

# Message Logger

- Objective
- Log Messages
- Multi-part Messages
- Pub/Sub and Proxy
- Proof of Concept

# Proof of Concept




```
<151>1 2014:07:14T23:16:14.000000Z gfalc6064d D2 Y6HN8 UE01H - BOM1405372574852
<106>1 2014:07:14T23:16:14.000000Z gfalc6064 NK ICFXS XN61L - BOM1405372574830
<159>1 2014:07:14T23:16:14.000000Z pc8946 JW AXLP3 WRTX8 - BOM1405372574750
<161>1 2014:07:14T23:16:14.000000Z gfalc6064d 0J BNRLI 7MFF4 - BOM1405372574852
<159>1 2014:07:14T23:16:14.000000Z gfalc6064 P0 U2FD0 6R4HS - BOM1405372574830
<43>1 2014:07:14T23:16:14.000000Z pc8946 YT SZZVV Q5SFJ - BOM1405372574750
<138>1 2014:07:14T23:16:14.000000Z gfalc6064d F5 868GS QNU6I - BOM1405372574852
<21>1 2014:07:14T23:16:14.000000Z gfalc6064 FO 12U3O QJ0WR - BOM1405372574830
<79>1 2014:07:14T23:16:14.000000Z pc8946 G1 PCWGX 4BGJI - BOM1405372574750
<162>1 2014:07:14T23:16:14.000000Z gfalc6064d OQ A0HZH 2BCPO - BOM1405372574852
<94>1 2014:07:14T23:16:14.000000Z gfalc6064 RM JMZKS 7O9ZO - BOM1405372574830
<148>1 2014:07:14T23:16:14.000000Z pc8946 H9 65BGM T6E52 - BOM1405372574750
<43>1 2014:07:14T23:16:14.000000Z gfalc6064d X6 BELSW CKM0Y - BOM1405372574852
<76>1 2014:07:14T23:16:14.000000Z gfalc6064 58 HNF46 2JQUQ - BOM1405372574830
<8>1 2014:07:14T23:16:14.000000Z pc8946 ME A20JO EP65F - BOM1405372574750
<164>1 2014:07:14T23:16:14.000000Z gfalc6064d FT 6B4JX GDIO9 - BOM1405372574852
<6>1 2014:07:14T23:16:14.000000Z gfalc6064 DU PJ7RS 2BWJ0 - BOM1405372574830
<4>1 2014:07:14T23:16:14.000000Z pc8946 5O 8OC3T Z8OY1 - BOM1405372574750
```

# Overview

- ZeroMQ
- Google Protocol Buffer
- Accelerator Models
- Message Logger
- Performance
- Ensuring Reliability (Majordomo Model)
- Conclusions

# Performance

- Number of roundtrips - $10^6$

| Message Size (Bytes) | Latency (microseconds) |
|---|---|
| 10 | 164.28 |

| Message Size (Bytes) | Throughput (messages/second) |
|---|---|
| 10 | 3394444 |
| 100 | 1121072 |
| 1000 | 115145 |

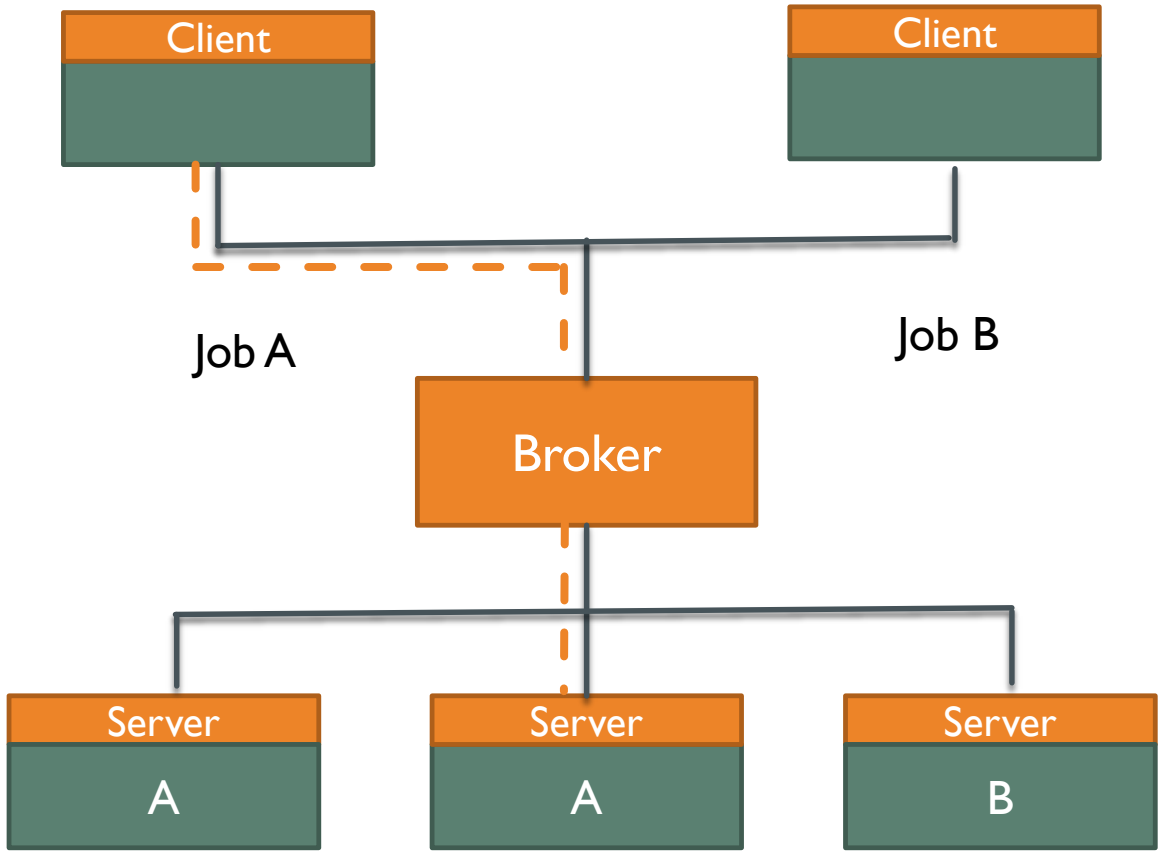
# Overview

- ZeroMQ
- Google Protocol Buffer
- Accelerator Models
- Message Logger
- Performance
- Ensuring Reliability (Majordomo Model)
- Conclusions

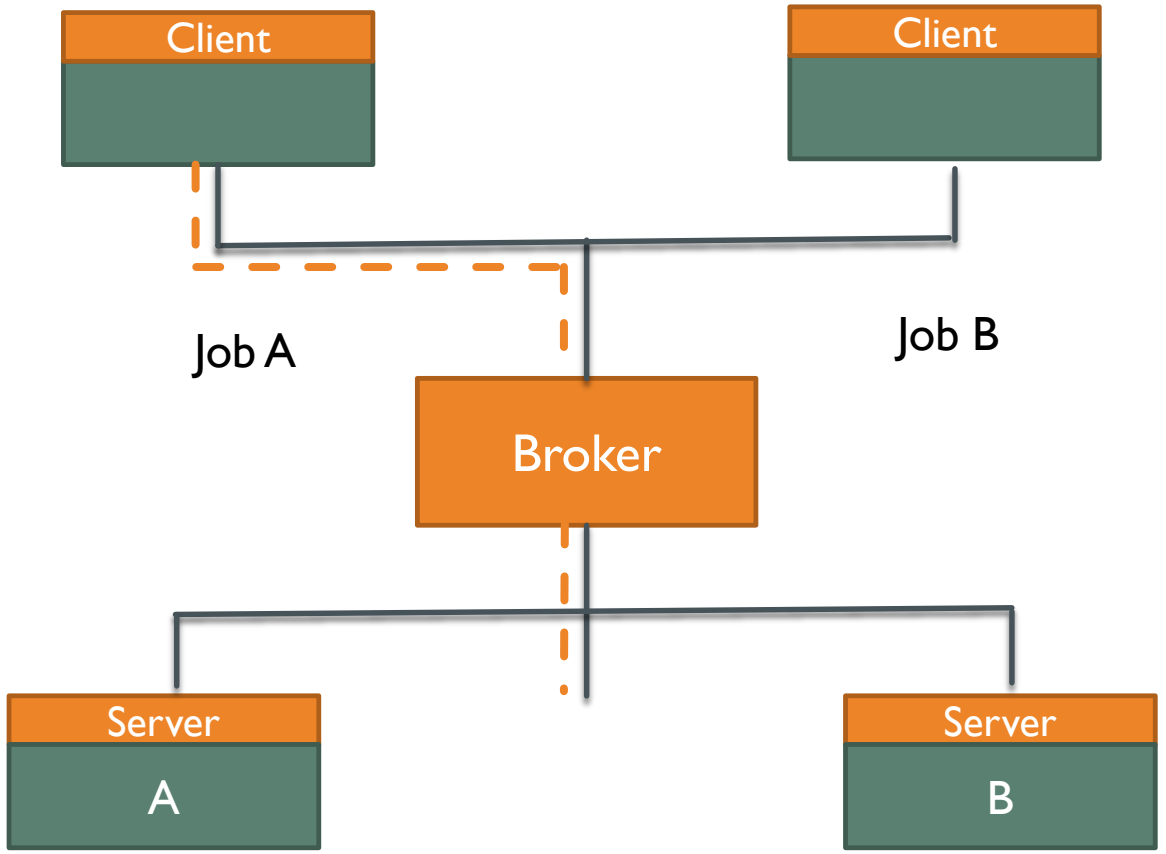
# Ensuring Reliability (Majordomo Model)

- Service-oriented architecture

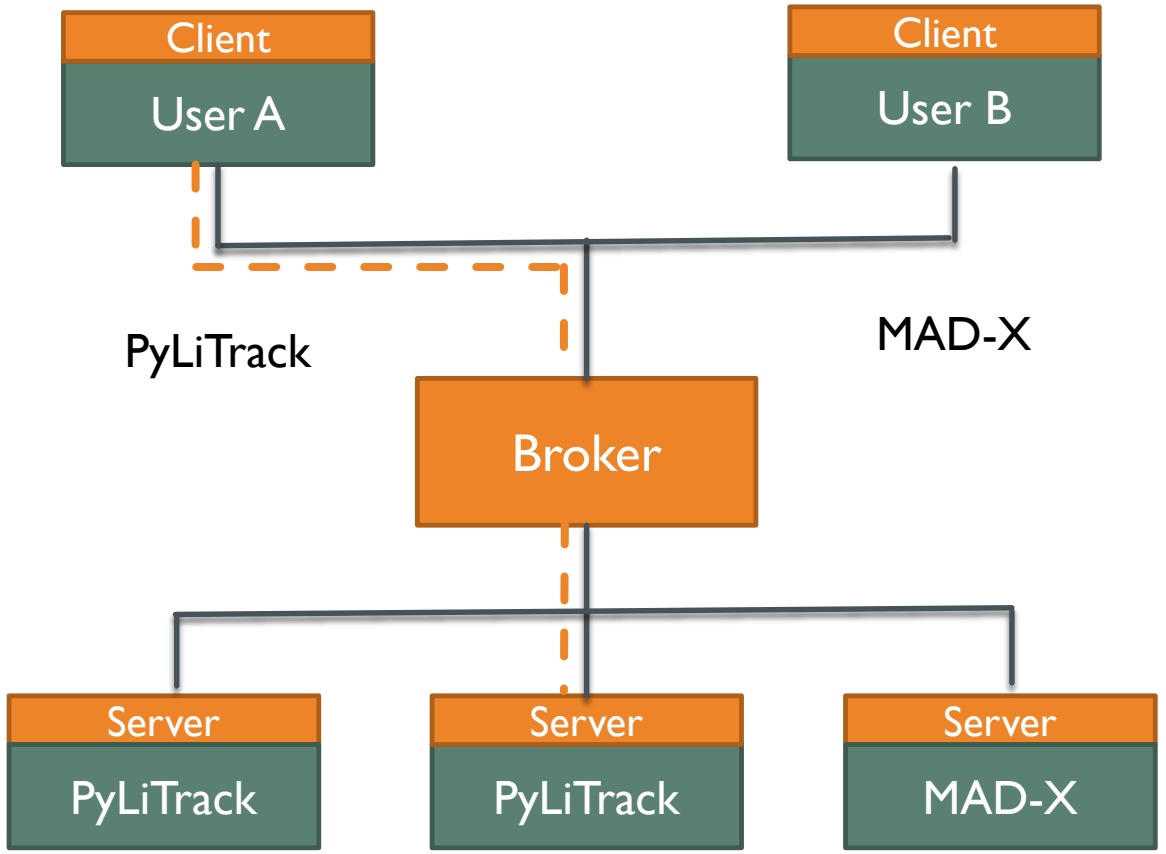
# Ensuring Reliability (Majordomo Model)



# Ensuring Reliability (Majordomo Model)



# Ensuring Reliability (Majordomo Model)



# Ensuring Reliability (Majordomo Model)

It has an ability to detect and recover from a specific set of failures:

- Server applications which crash, run too slowly, or freeze.
- Server applications that are disconnected from the network (temporarily or permanently).
- Client applications that are temporarily disconnected from the network.
- Requests or replies that are lost due to any of these failures.

# Overview

- ZeroMQ
- Google Protocol Buffer
- Accelerator Models
- Message Logger
- Performance
- Ensuring Reliability (Majordomo Model)
- Conclusions

# Conclusions

- Explored various facets of ZeroMQ
  - Messaging Patterns
  - Multipart Messages
  - Majordomo Protocol (in-progress)
- Implemented a framework for accessing accelerator models in different programming languages (C++, Java and Python)
  - PyLiPtrack (Python)
  - MAD-X (CPP)
- Developed a proof of concept for a error message logger

# Acknowledgements


- Jan Chrin
- Hans Braun
- Masamitsu Aiba
- PSI