Apache Thrift
Introduction & Tutorial

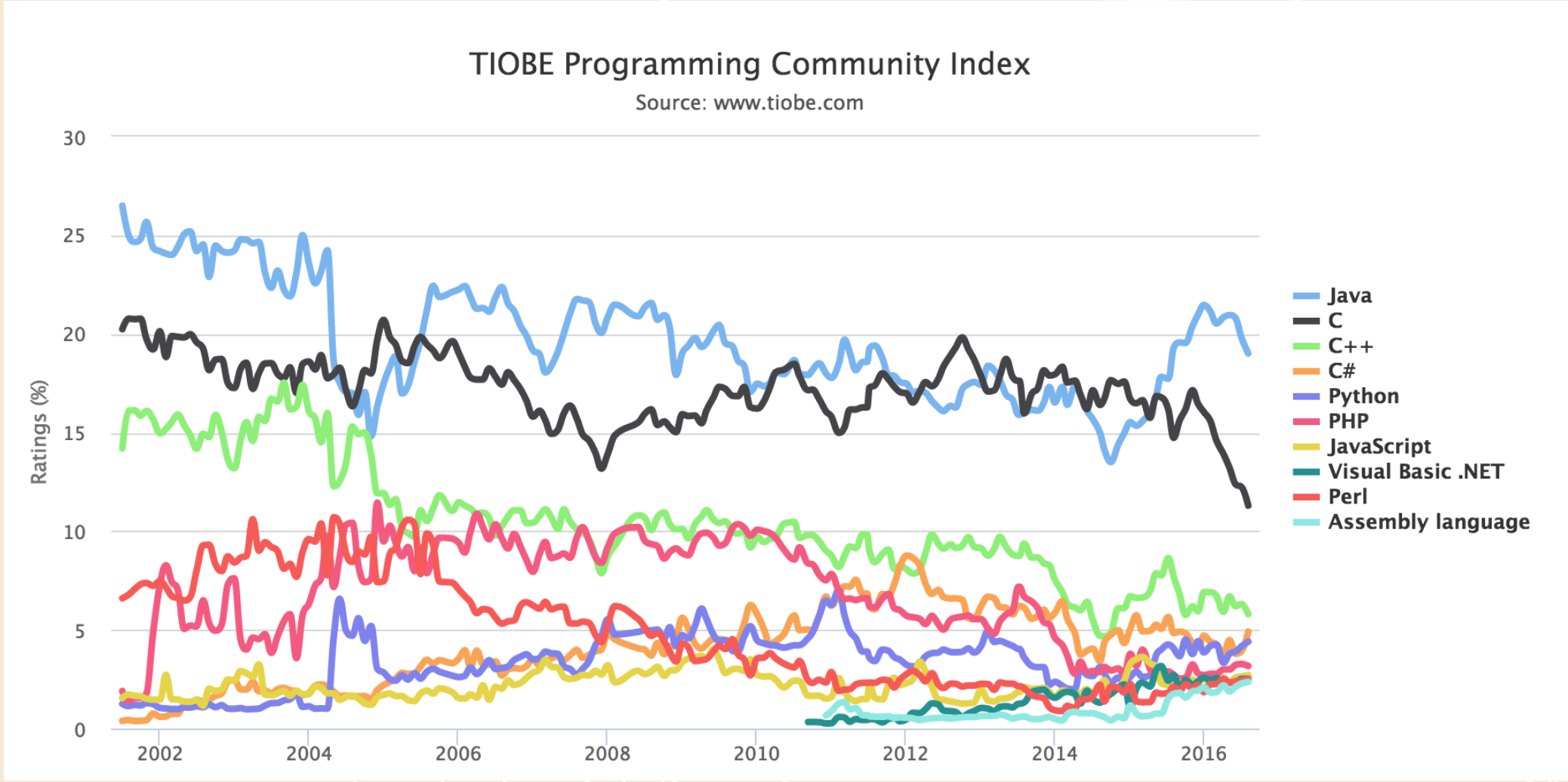
Marlon Pierce, Suresh Marru

CSCI-B 649 Science Gateway Architectures

INDIANA UNIVERSITY BLOOMINGTON
SCHOOL OF INFORMATICS AND COMPUTING

Q & A on Assignment 1

TIOBE Index


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
Java
C
C++
C#
Python
PHP
JavaScript
Visual Basic .NET
Perl
Assembly language

Programming Language "polyglotism"

- Modern distributed applications are rarely composed of modules written in a single language.
- Weaving together innovations made in a range of languages is a core competency of successful enterprises.
- Cross language communications are a necessity, not a luxury.
- In your projects you need to demonstrate this by using three or more languages.

Protocol Buffers

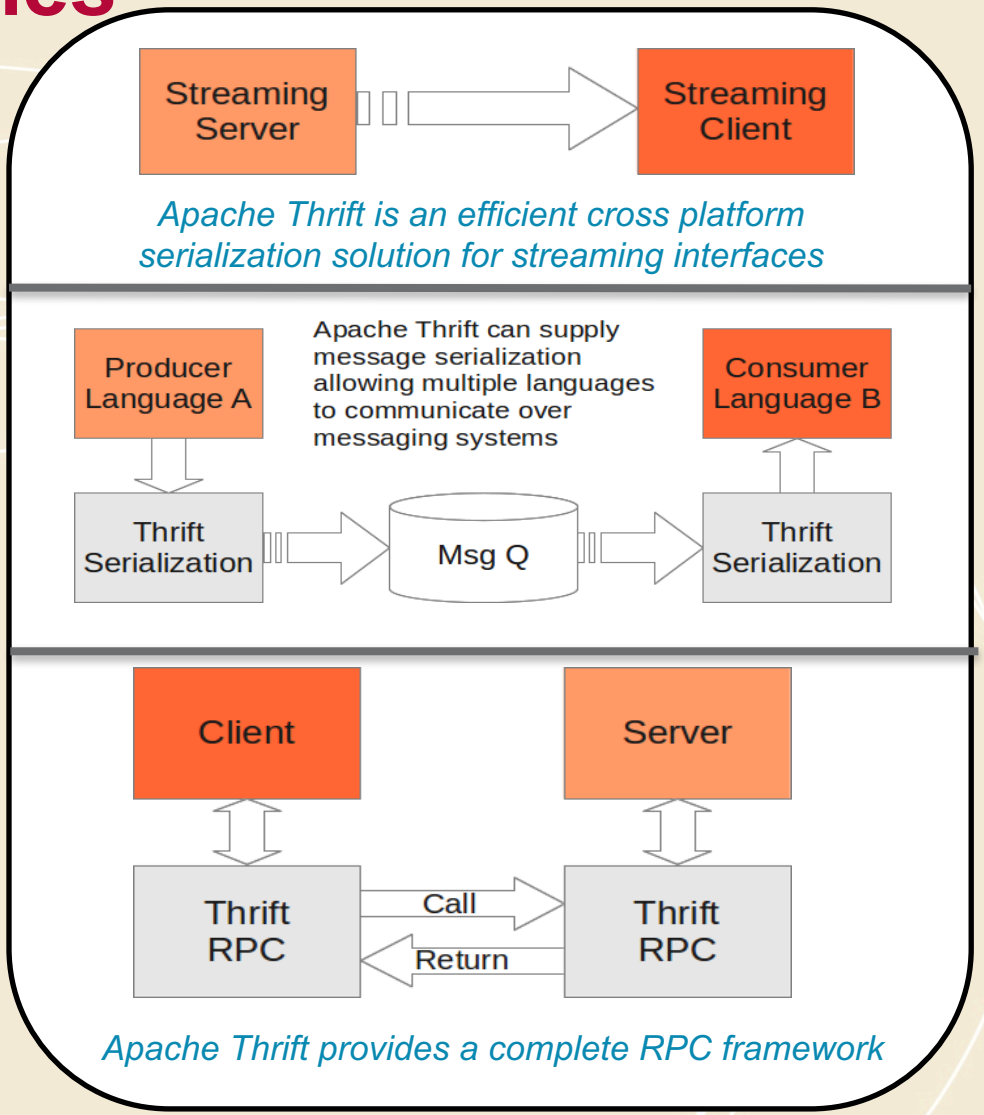
- “a language-neutral, platform-neutral, extensible way of serializing structured data for use in communications protocols, data storage, and more.”
- “Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data – think XML, but smaller, faster, and simpler. ”
  - https://developers.google.com/protocol-buffers/docs/overview
- Started internally within Google in 2001 and Opened in 2008.

Apache Thrift

- Thrift is Facebook's implementation of Proto Buff open sourced under Apache.
- A high performance, scalable cross language serialization and RPC framework.
- Provides a full RPC Implementation with generated clients, servers, everything but the business logic.
- Thrift is is fast and efficient, solutions for minimal parsing overhead and minimal size.

Thrift: Multiple Communication Schemes

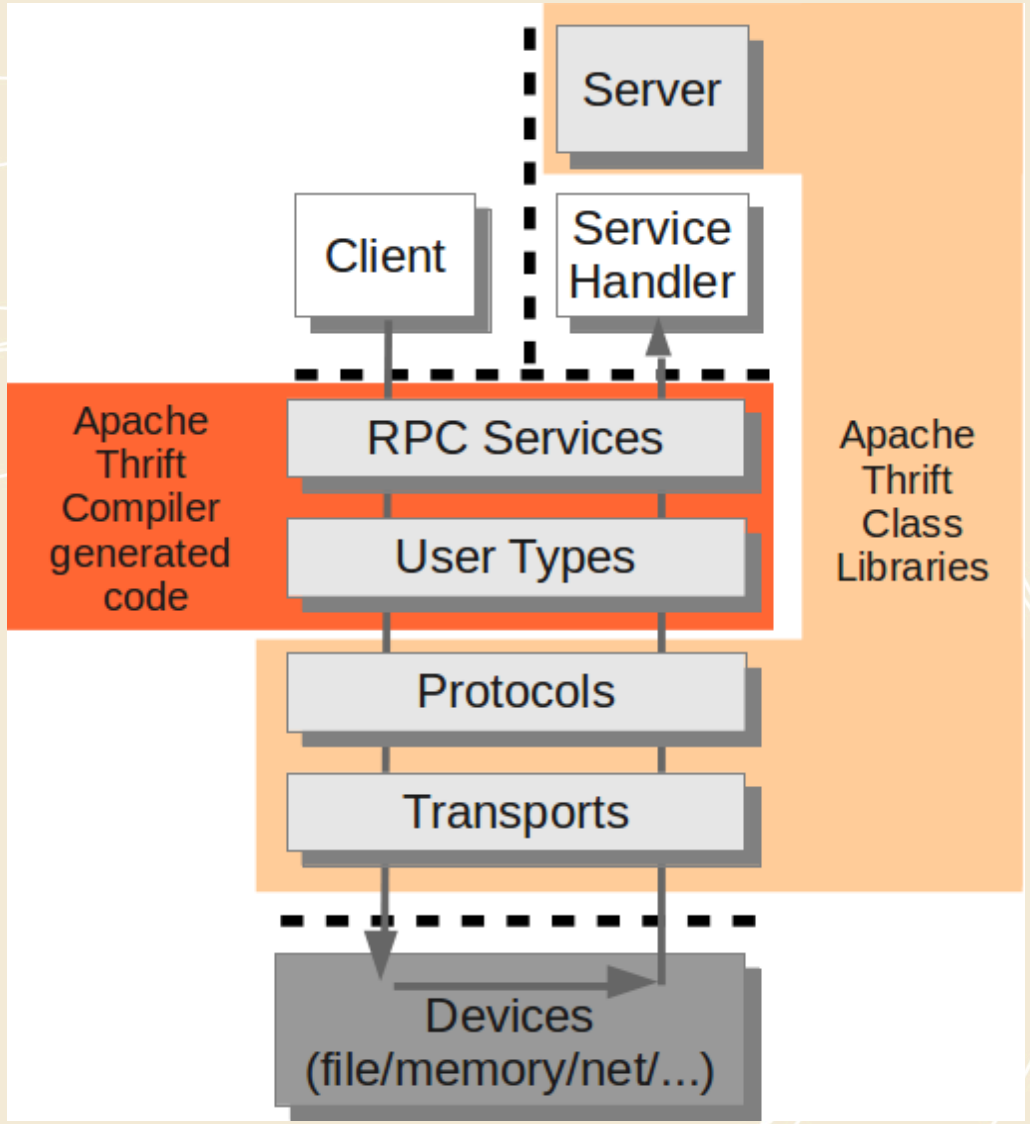
- Streaming – Communications characterized by an ongoing flow of bytes from a server to one or more clients.
  - Example: An internet radio broadcast where the client receives bytes over time transmitted by the server in an ongoing sequence of small packets.
- Messaging – Message passing involves one way asynchronous, often queued, communications, producing loosely coupled systems.
  - Example: Sending an email message where you may get a response or you may not, and if you do get a response you don't know exactly when you will get it.
- RPC – Remote Procedure Call systems allow function calls to be made between processes on different computers.
  - Example: An iPhone app calling a service on the Internet which returns the weather forecast.



Source: Randy Abernethy. The Programmer's Guide to Apache Thrift, Manning Publications Co.

Thrift for RPC Services

- User Code
  - client code calls RPC methods and/or [de]serializes objects
  - service handlers implement RPC service behavior
- Generated Code
  - RPC stubs supply client side proxies and server side processors
  - type serialization code provides serialization for IDL defined types
- Library Code
  - servers host user defined services, managing connections and concurrency
  - protocols perform serialization
  - transports move bytes from here to there



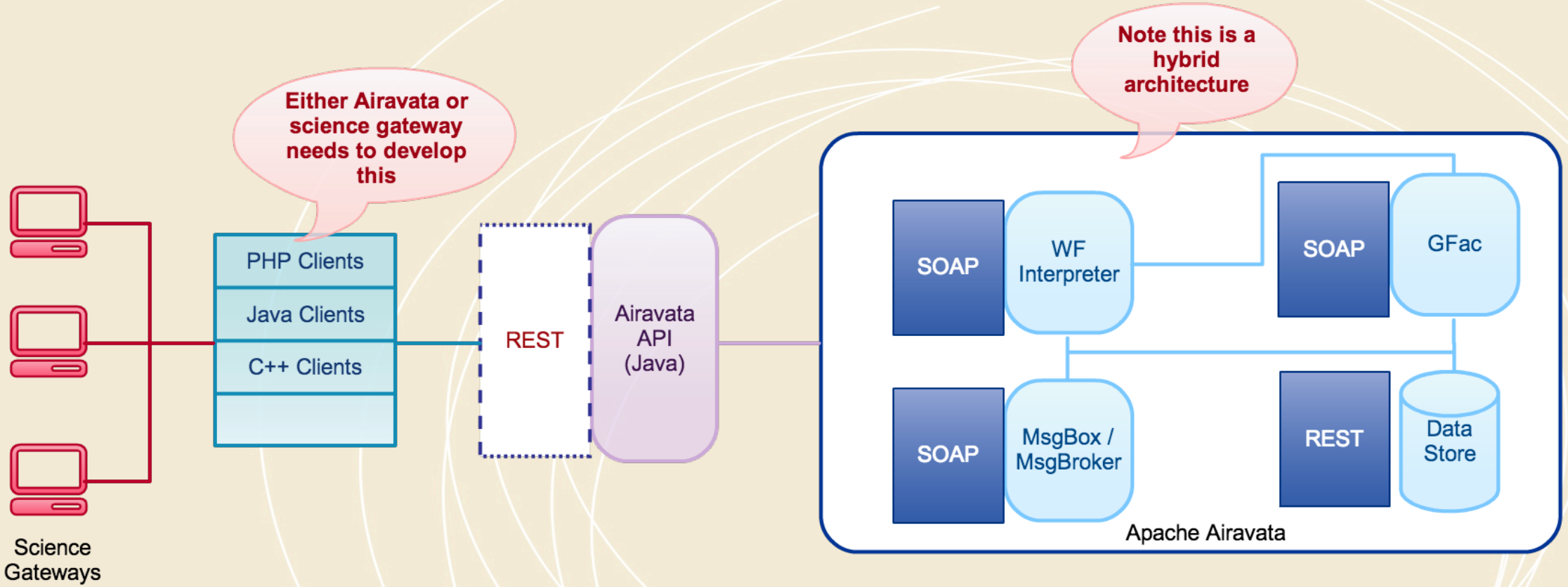
Source: Randy Abernethy. The Programmer's Guide to Apache Thrift, Manning Publications Co.

Thrift Resources

- Web
  - thrift.apache.org
  - github.com/apache/thrift
- Mail
  - Users: user-subscribe@thrift.apache.org
  - Developers: dev-subscribe@thrift.apache.org
- Chat
  - #thrift

Thrift Experiences within Apache Airavata

SOAP/REST based Airavata



(a simplified view - evolved over 10 years)

RPC using SOAP

- Pros
  - ability to separate out context and the payload
  - Already proven tools and techniques (XML, WSDL, WS-Security etc)
  - Clients can easily generate stubs using WSDLs
- Cons
  - Heavy weight
  - Data schema changes cannot be handled easily
  - Could not support broad range of clients

RPC using REST

- Pros
  - Flexible for data representation (JSON or XML).
  - Light weight.
  - Better performance.
- Cons
  - Multiple object layers.
  - No standard way to describe the service to the client.

Note: A detailed lecture on REST is forthcoming.

RPC using Thrift

- Integrating a language specific client library is easier than an open API like REST.
- Light framework
  - No multiple dependencies and servlet containers.
- Easy learning curve to get started
- If carefully crafted, the IDLs and framework support backward and forward compatibility

Note: These arguments will be similar for ProtoBuff and Avro

IDL's with Richer Data Structures

- Experiment data model is a **complex data model**
- Data strucutures : string, type-defs, integers, lists, sets
- Can refer other structs, enums

```
struct Experiment {
    1: required string experimentID = DEFAULT_ID,
    2: required string projectID = DEFAULT_PROJECT_NAME
    3: optional i64 creationTime,
    4: required string userName,
    5: required string name,
    6: optional string description,
    7: optional string applicationId,
    8: optional string applicationVersion,
    9: optional string workflowTemplateId,
    10: optional string workflowTemplateVersion,
    11: optional UserConfigurationData userConfigurationData,
    12: optional string workflowExecutionInstanceId,
    13: optional list<DataObjectType> experimentInputs,
    14: optional list<DataObjectType> experimentOutputs,
    15: optional ExperimentStatus experimentStatus,
    16: optional list<WorkflowNodeStatus> stateChangeList,
    17: optional list<WorkflowNodeDetails> workflowNodeDetailsList,
    18: optional list<ErrorDetails> errors
}
```

Defining a struct

- Can send such **complex data model** over the wire
- Also note that Thrift support **exception handling** too.

```
string createExperiment(1: required experimentModel.Experiment experiment)
  throws (1: airavataErrors.InvalidRequestException ire,
          2: airavataErrors.AiravataClientException ace,
          3: airavataErrors.AiravataSystemException ase)

/**
```

Defining a service method

Thrift Experiences Summary

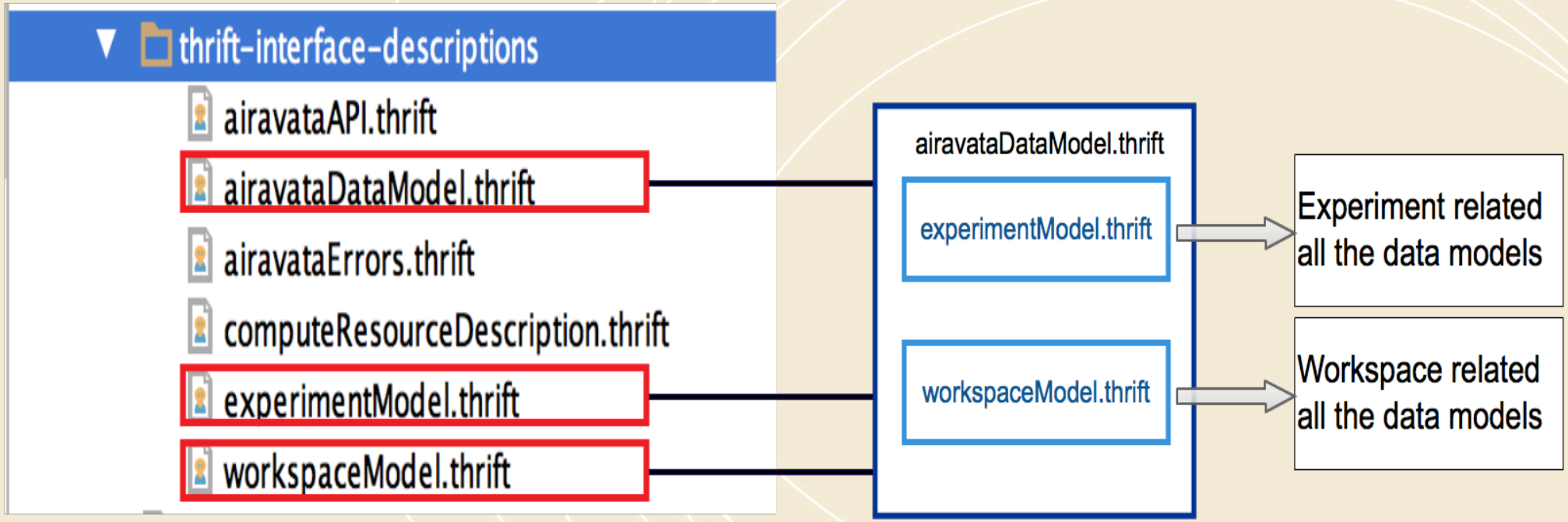
- Auto generated PolyGot clients.
- Clean design
- Multiple servers to choose from:
  - TSimpleServer - Simple single threaded server
  - TThreadPoolServer - Uses Java's built in ThreadPool management
  - TNonblockingServer - non-blocking TServer implementation
  - THsHaServer - extension of the TNonblockingServer to a Half-Sync/Half-Async server

Experiences Contd..

- No explicit need for marshalling / unmarshalling.
- Data models can also be used internally.
- Servers are very robust - able to handle large number of concurrent requests.
- Easy to do modifications to the models, since code is auto-generated.
- Convenient way to achieve backward compatibility.

Lessons Learned

Modularize the data models in order to ease the maintenance



Lessons Learned Contd..

```
/**
 * A structure holding the experiment metadata and its child models.
 *
 * userName:
 *   The user name of the targeted gateway end user on whose behalf the experiment is being created.
 *     the associated gateway identity can only be inferred from the security hand-shake so as to avoid
 *     authorized Airavata Clients mimicking an unauthorized request. If a gateway is not registered with
 *     Airavata, an authorization exception is thrown.
 *
 * experimentName:
 *   The name of the experiment as defined by the user. The name need not be unique as uniqueness is enforced
 *      by the generated experiment id.
 *
 * experimentDescription:
 *    The verbose description of the experiment. This is an optional parameter.
*/
struct Experiment {
    1: required string experimentID = DEFAULT_ID,
    2: required string projectID = DEFAULT_PROJECT_NAME
    3: optional i64 creationTime,
    4: required string userName,
    5: required string name,
    6: optional string description,
    7: optional string applicationId,
    8: optional string applicationVersion,
    9: optional string workflowTemplateId,
    10: optional string workflowTemplateVersion,
    11: optional UserConfigurationData userConfigurationData,
    12: optional string workflowExecutionInstanceId,
    13: optional list<DataObjectType> experimentInputs,
    14: optional list<DataObjectType> experimentOutputs,
    15: optional ExperimentStatus experimentStatus,
    16: optional list<WorkflowNodeStatus> stateChangeList,
    17: optional list<WorkflowNodeDetails> workflowNodeDetailsList,
    18: optional list<ErrorDetails> errors
}
```

Add proper documentation to structs and services

Use "set" if you want to avoid duplicates

```
/**
 * Terminate a running experiment.
 *
 * @param airavataExperimentId
 *    The identifier for the requested experiment. This is returned during the create experiment step.
 *
 * @return
 *   This method call does not have a return value.
 *
 * @throws org.apache.airavata.api.error.InvalidRequestException
 *    For any incorrect forming of the request itself.
 *
 * @throws org.apache.airavata.api.error.ExperimentNotFoundException
 *    If the specified experiment is not previously created, then an Experiment Not Found Exception is thrown.
 *
 * @throws org.apache.airavata.api.error.AiravataClientException
 *    The following list of exceptions are thrown which Airavata Client can take corrective actions to resolve:
 *
 *      UNKNOWN_GATEWAY_ID - If a Gateway is not registered with Airavata as a one time administrative
 *         step, then Airavata Registry will not have a provenance area setup. The client has to follow
 *         gateway registration steps and retry this request.
 *
 *      AUTHENTICATION_FAILURE - How Authentication will be implemented is yet to be determined.
 *         For now this is a place holder.
 *
 *      INVALID_AUTHORIZATION - This will throw an authorization exception. When a more robust security hand-shake
 *         is implemented, the authorization will be more substantial.
 *
 * @throws org.apache.airavata.api.error.AiravataSystemException
 *    This exception will be thrown for any Airavata Server side issues and if the problem cannot be corrected by the c
 *      rather an Airavata Administrator will be notified to take corrective action.
 *
 */
void terminateExperiment(1: string airavataExperimentId)
  throws (1: airavataErrors.InvalidRequestException ire,
          2: airavataErrors.ExperimentNotFoundException enf,
          3: airavataErrors.AiravataClientException ace,
          4: airavataErrors.AiravataSystemException ase)
}
```

Lessons Learned Contd..

- Thrift has limited support to handle null values.
  - Models can be complex with lot of other structs inside.
  - Enums cannot be passed as null over the wire even they are specified as optional in the struct
- Thrift has limited documentation and samples
  - airavata can be used as a reference
  - https://github.com/apache/airavata

Apache Thrift Tutorial

4 Simple Steps to Create a RPC microservice

1. Define the service in a language neutral “Interface Description Language”.
2. Compile the IDL to generate Server and Client “stubs” in desired programming languages.
3. Plug the server implementation in the pre-generated server stub.
4. Call the remote services as if they are making local method calls.

Thank You!

Marlon Pierce, Suresh Marru

{marpierc, smarru}@iu.edu