Java Interface For the Trajectory Synthesizer

DANISH VAID
UNIVERSITIES SPACE RESEARCH ASSOCIATION
SEPTEMBER 4TH, 2015

What Is the Trajectory Synthesizer (TS)?

- Used for CTAS to generate trajectories for:
  - Scheduling
  - Conflict Prediction/Resolution
- Generates Trajectories from input files
- Written and currently accessible in C/C++

Task and Motivation

- Create a JAVA interface to access the TS
- Allow JAVA Research Software Platforms (ex. ACES, FACET, etc) to use the TS as an alternative Trajectory Generator
- Research approaches for JAVA to access C++

Task Stages

- Stage 1: Research different tools/library
- Stage 2: Test research results, learn, and determine best option
- Stage 3: Initial prototyping
  - Sub – Stage 3.1: Pass TsInput file name and process
  - Sub – Stage 3.2: Protoype different data structure types
- Stage 4: Design and implement

  Create TsInput Java Object and pass it into C++ TS
  - Sub – Stage 4.1: TS Class to Struct conversion
  - Sub – Stage 4.2: Java (JNA) Declaration and Linking
  - Sub – Stage 4.3: Struct to Class Constructor
  - Sub – Stage 4.4: Return results to Java side

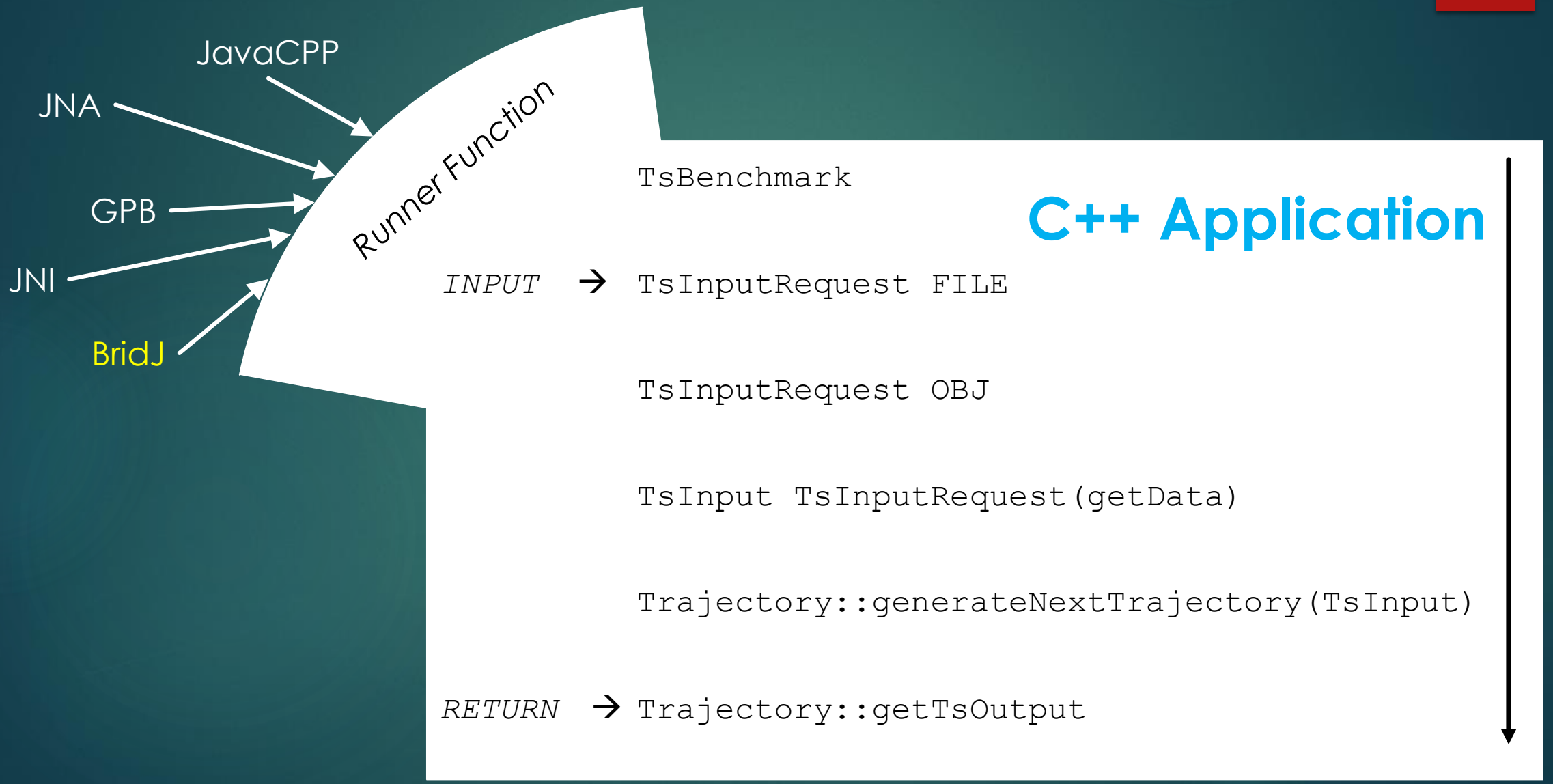
Stage 1: Possible Libraries/Tools

- Google Protocol Buffers (GPB)
- BridJ
- Java Native Interface (JNI)
- JavaCPP
- Java Native Access (JNA)

Implementation Ease: JNA > GPB > JavaCPP/JNI

Performance/Speed: JNI > JavaCPP > JNA > GPB

Data Flow Chart



JNI vs JNA

JNI

- Framework enables code running in Java Virtual Machine
- Allows Native method access
- Mapped through (machine generated) header file

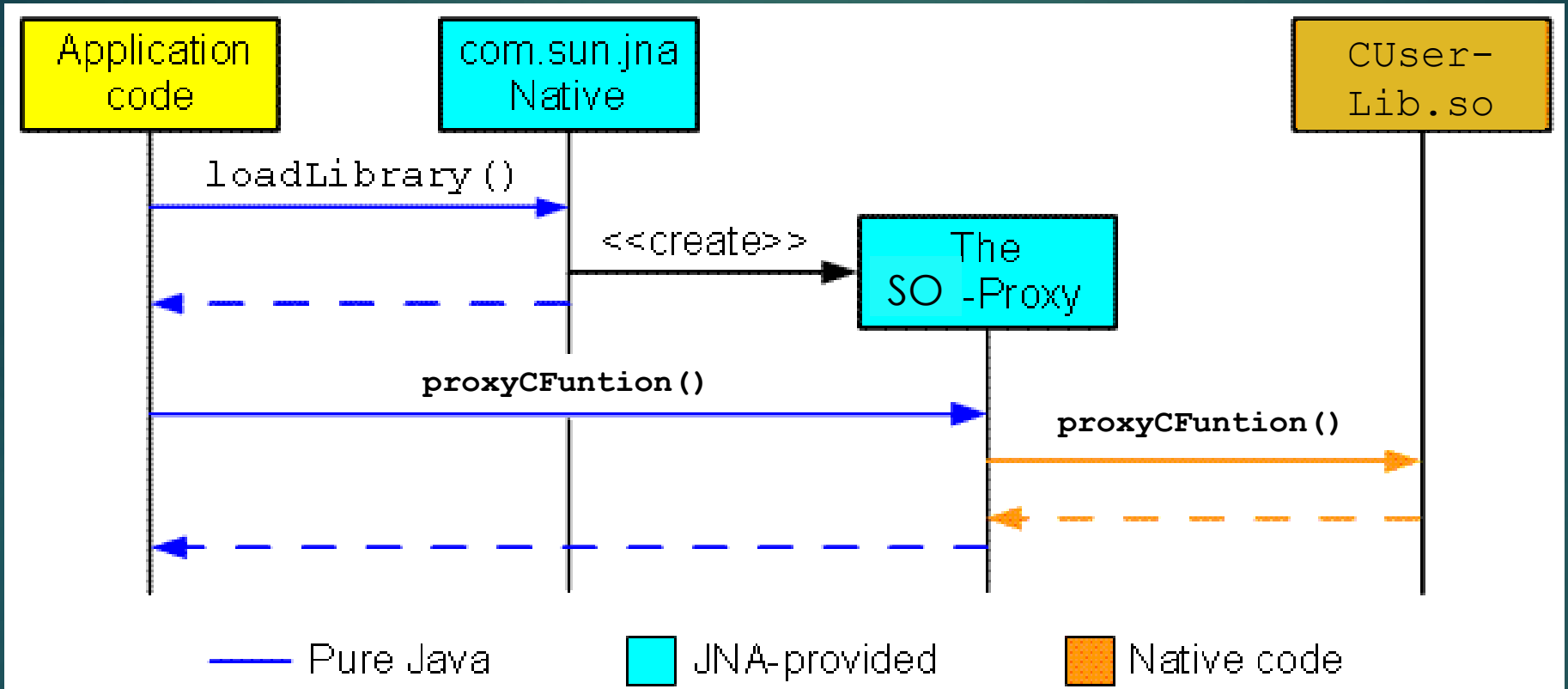
JNA

- Community developed layover to JNI
- Uses foreign function native interface for *dynamic* invocation using proxying
- Allows structure development and passing
- Mapping handled automatically during declaration
- Some reports say JNA is 10x slower than JNI

What Is A Wrapper?

- Encapsulated the functionality of another component
- Provides a level of abstraction from underlying application
- Acts as a “Bridge”

How Does JNA work?



- Uses proxy pattern
- Obtains proxy-ed object/methods from SO (shared object file)
- Automatically handles all run-time aspects
- Code must extend: com.sun.jna.Library

Stage 2: JNA Basic Example

C:

```
int  example1(int val)
{
    return val * 2;
}
```

JAVA:

```
Public interface Clibrary extends Library{
    public int example1(int val);
}
Clibrary clib = (Clibrary)Native.loadLibrary("testlib", CLibrary.class);
int newVal = clib.example1(23);
System.out.println("example 1: " + newVal);
```

- Compile and Execute

gcc – o *libName* –shared *fileName.c* → Create shared object

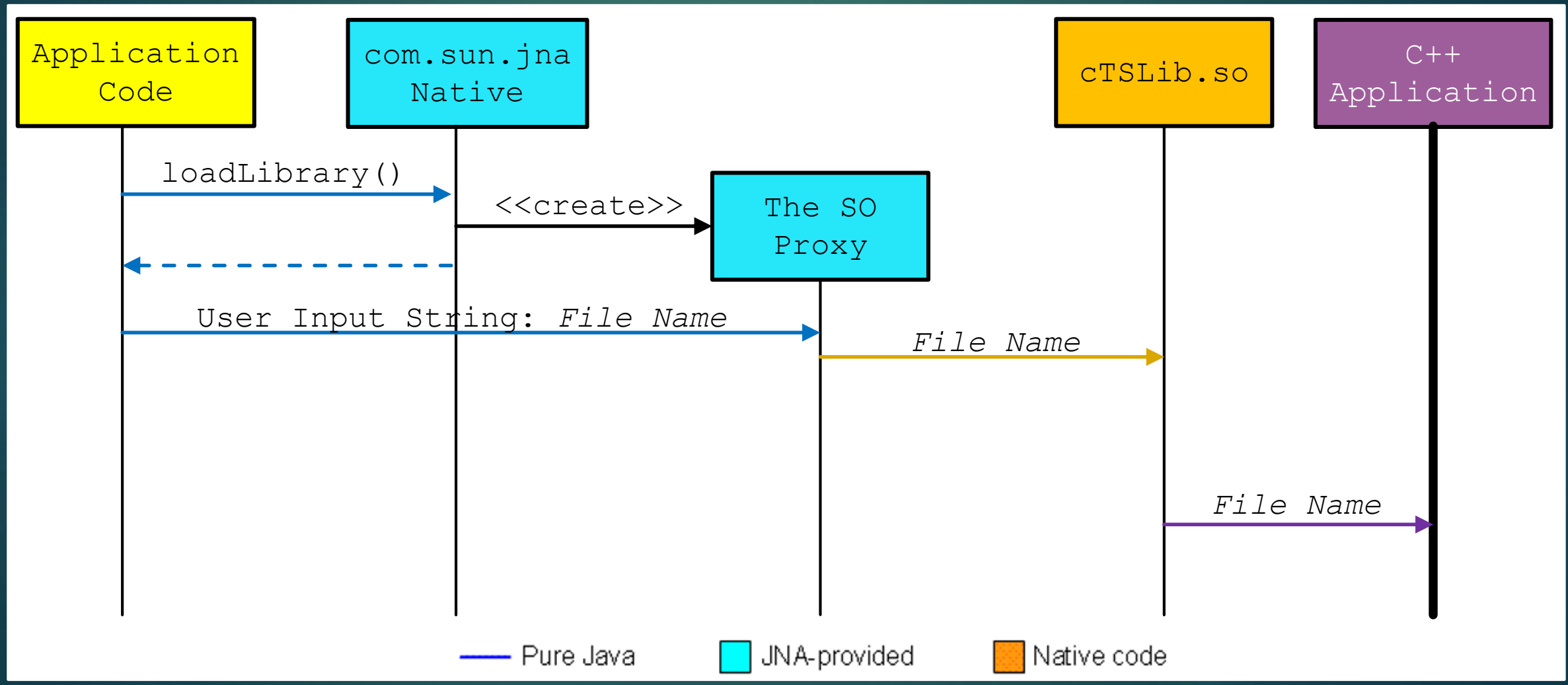
Javac –classpath jna.jar *fileName*.java → Compile .java files

Javac –classpath jna.jar:. *Filename* → Run java class

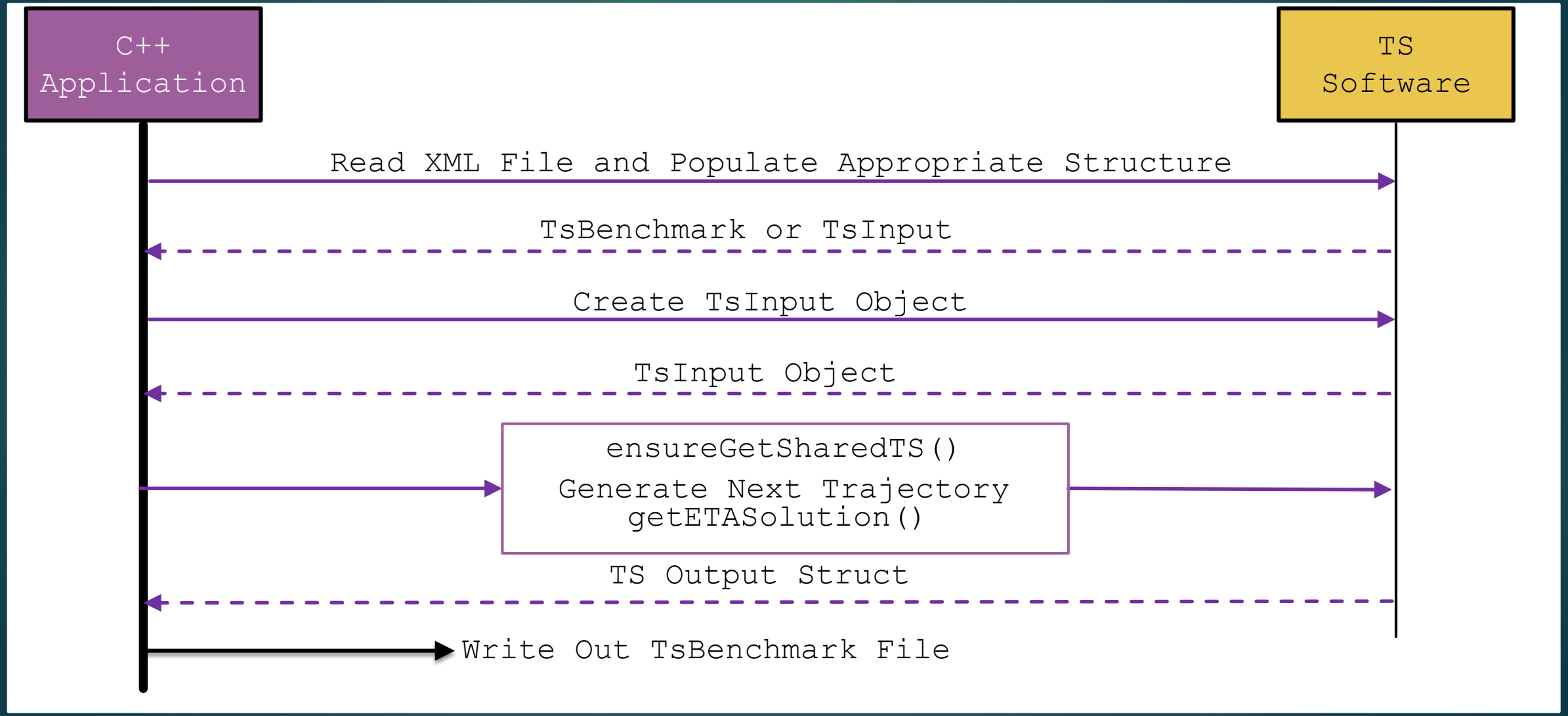
Bash File Written For .SO linking and Running

- Automated Bash file using TS shared library
- Imports JNA packages in libTS.so (shared object file)
- Compiles JAVA files and links to .so
- Runs classes

Stage 3: Develop JAVA Wrapper Passing TsInput File Name String to TS



Stage 3: Develop JAVA Wrapper Passing TsInput File Name String to TS

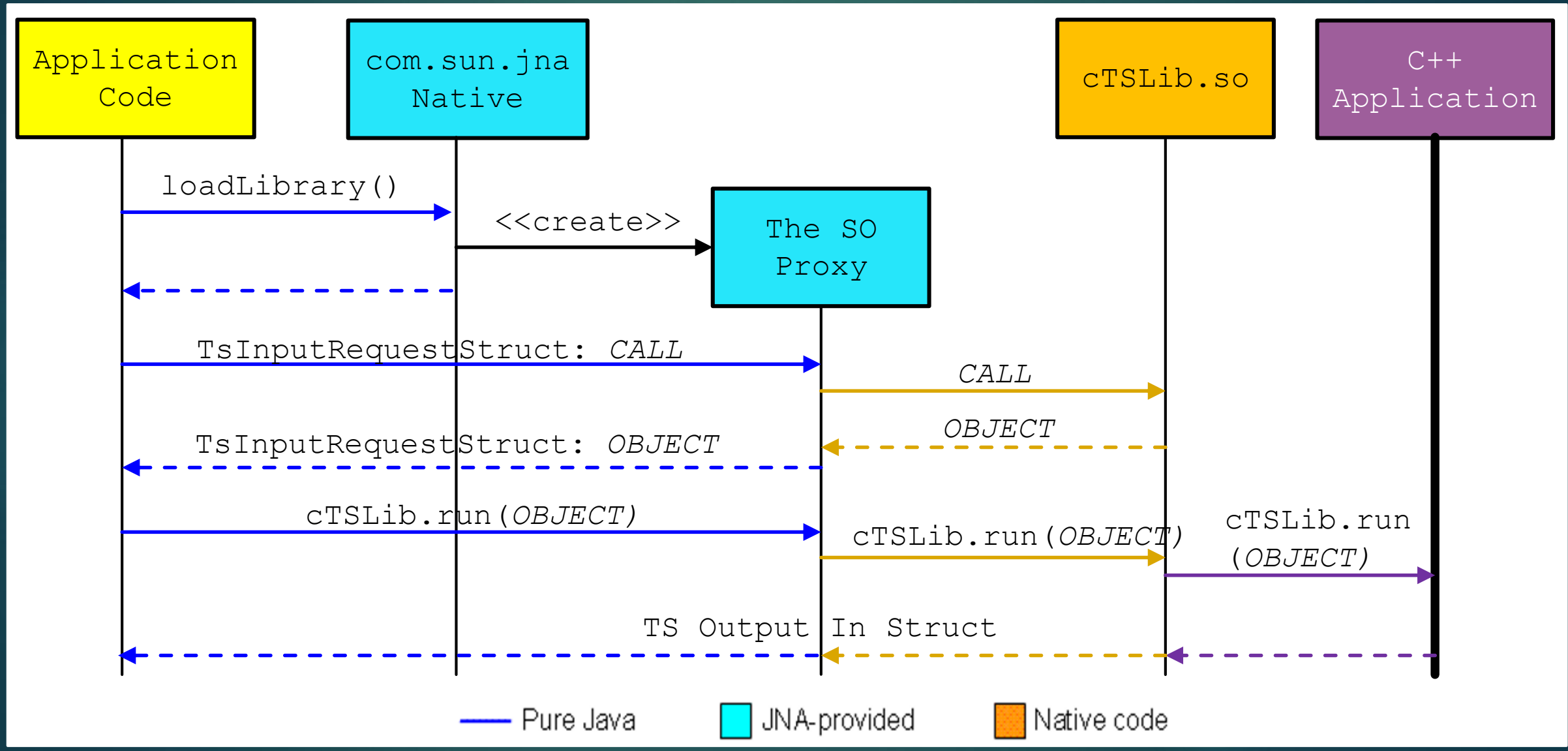


Stage 3.2: Develop JAVA Wrapper Protoype different data structure types

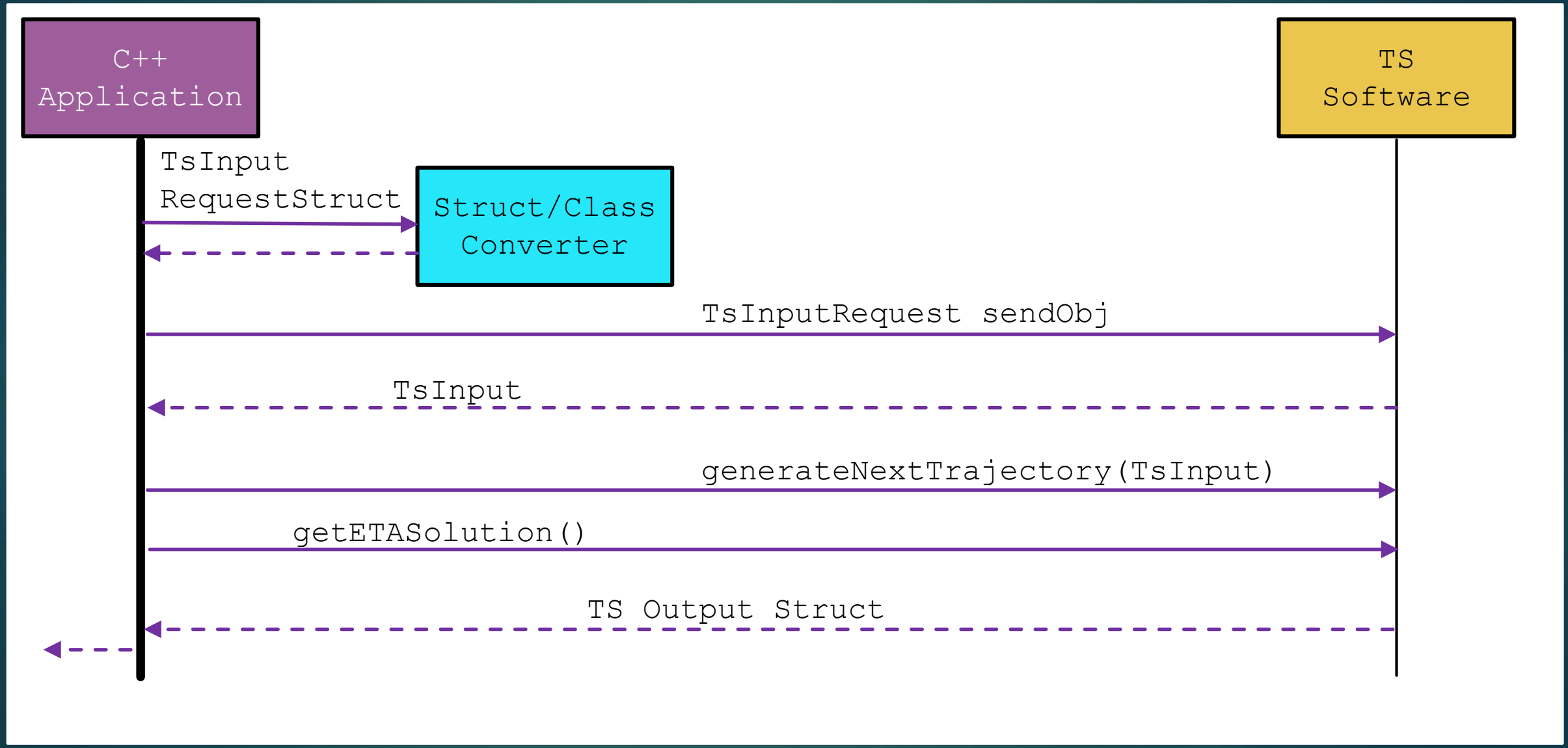
Tested Passing and returning of:

- Plain old data types (int, float, double, etc)
- C Strings
- Arrays of PODs
- Arrays of Structs
- Unions
- Arrays of Unions
- Arrays of Structures containing Unions

Stage 4: Create TsInput Java Object and pass it into C++ TS

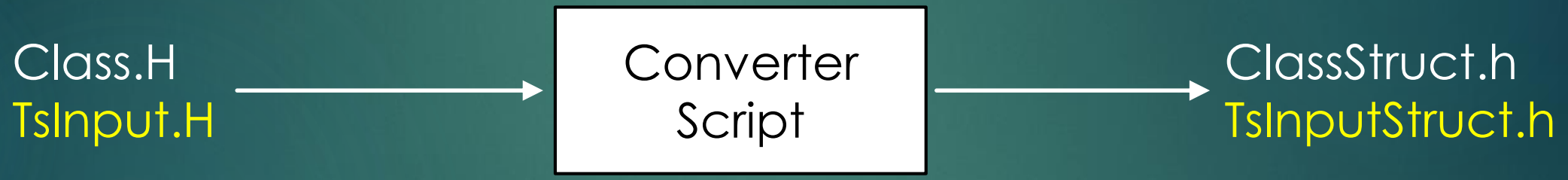


Stage 4: Create TsInput Java Object and pass it into C++ TS



Stage 4.1: TS Class to Struct conversion

- TS in C++ classes to C Structs
- Created a script



- File instantiation
- Regular expressions module

Stage 4.2: Java (JNA) Declaration and Linking

- Write all structs (data structures we created) in JNA Java format
- Example:

C:

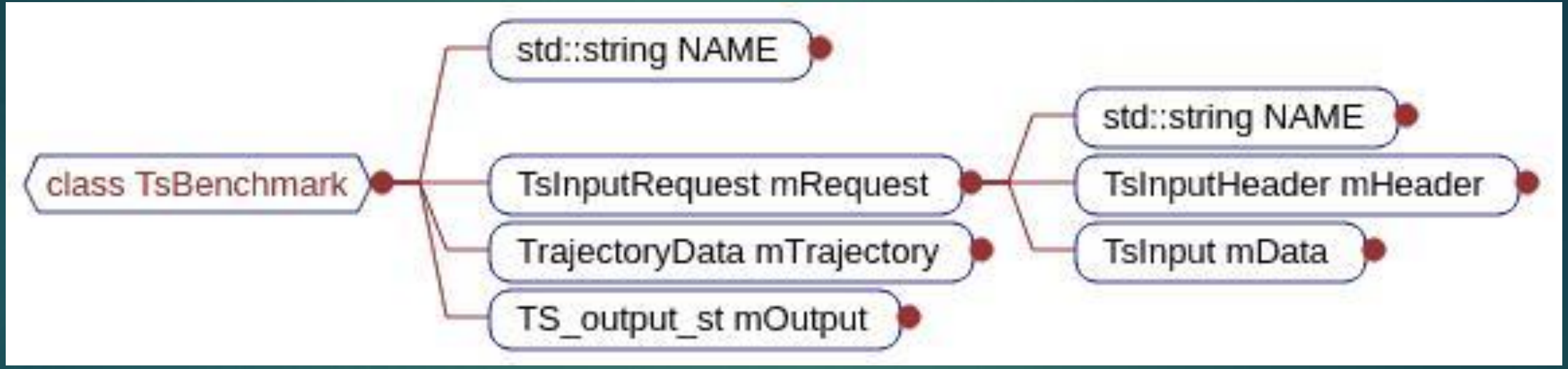
```
Struct TSInputRequestStruct{
    static const std::string NAME;
    TsInputHeaderStruct mHeader;
    TsInputStruct mData;
};
```

JAVA:

```
Public static class TsInputRequestStruct extends Structure{
   public static class ByValue extends TsInputRequestStruct implements Structure.ByValue{}

   public static String Name;
   public TsInputHeaderStruct mHeader;
   public TsInputStruct mData;
}
```

TS Data Members Breakdown



Stage 4.3: Struct to Class Constructor

- Initializes TS established class using passed in Struct

C/C++:

Struct Defined as:

```
Struct TsInputRequestStruct{
    static const std::string NAME;
    TsInputHeaderStruct mHeader;
    TsInputStruct mData;
};
```

Class Constructor:

```
TsInputRequest structToClass(TsInputRequestStruct inp){
    return  TsInputRequest():
            NAME(inp.NAME)
            mHeader(inp.mHeader)
            mData(inp.mData);
}
```

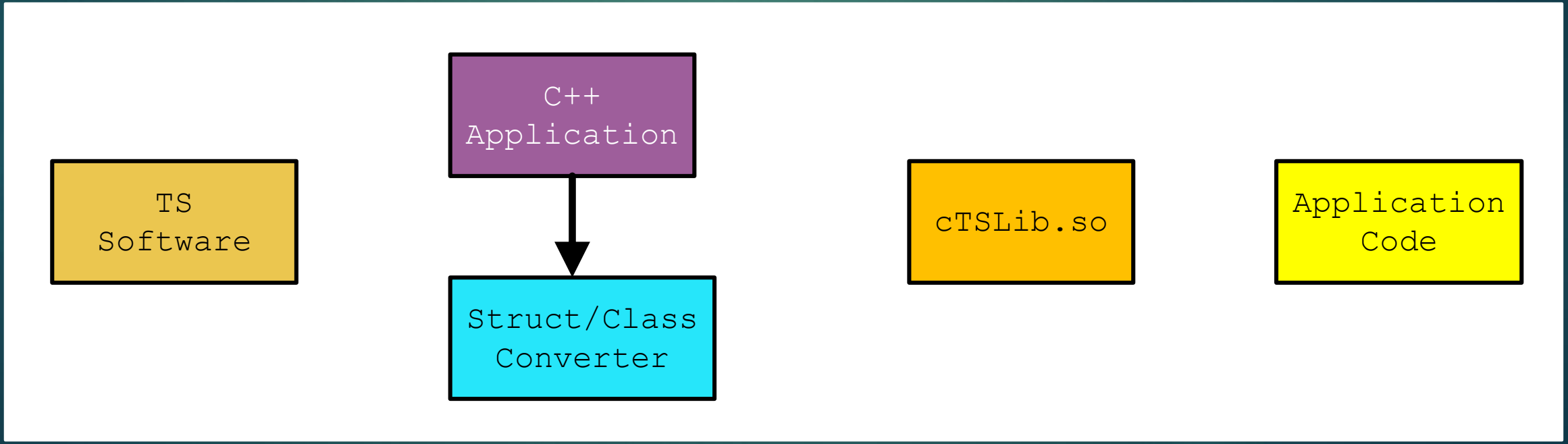
Current Progress

- ✔ Stage 1: Research different tools/library
- ✔ Stage 2: Test research results, learn, and determine best option
- ✔ Stage 3: Initial prototyping
  - ✔ Sub – Stage 3.1: Pass TsInput file name and process
  - ✔ Sub – Stage 3.2: Protoype different data structure types
- Stage 4: Design and implement chosen option

  Create TsInput Java Object and pass it into C++ TS and return results
  - ✔ Sub – Stage 4.1: TS Class to Struct conversion
  - ✔ Sub – Stage 4.2: Java Declaration and Linking
  - ✔ Sub – Stage 4.3: Struct to Class Constructor
  - ✔ Sub – Stage 4.4: Return results to Java side

Code Maintenance

- Stages to follow through for any changes to data structures
  - Sub – Stage 4.1: TS Class to Struct conversion
  - Sub – Stage 4.2: Java Declaration and Linking
  - Sub – Stage 4.3: Struct to Class Constructon



What Did I Learn?

- Scripting
- Data Member Management
- JNA
- Makefiles
- Creating/Using SO
- Linux tools and GIT
- Documentation, Documentation, *Documentation!*
- Professional/Research Environment
- Professional Etiquette

THANK YOU FOR YOUR TIME

Thanks to:
Michael
Gilbert
Charles
Lingmei
Saugata

QUESTIONS?

danishvaid@umail.ucsb.edu