Protocol Buffers

https://developers.google.com/protocol-buffers/

manojav@gmail.com

Agenda

- Introduction and history
- What are we trying to solve
- Example
- Software architecture
- Comparison

What are Protocol Buffers ?

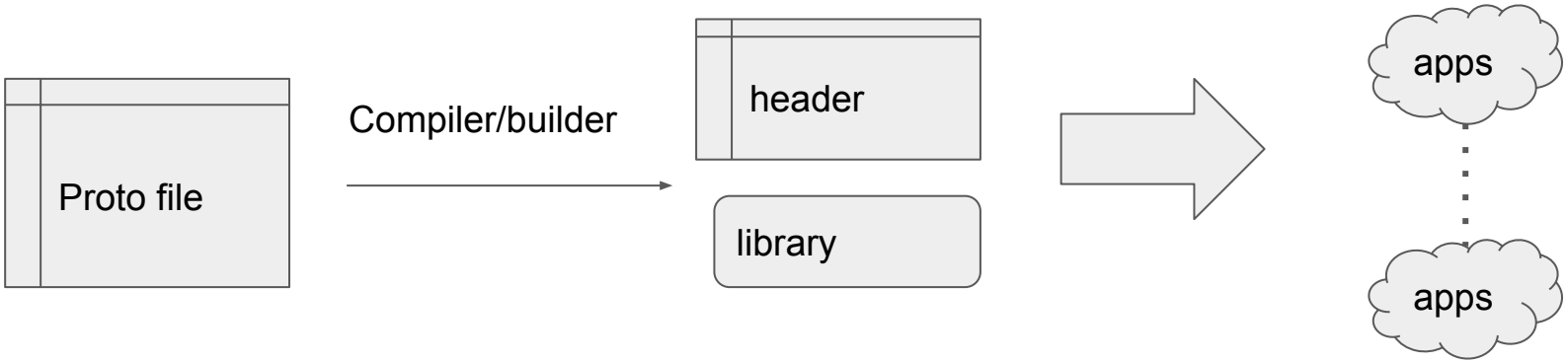
“Data containers” that are:-

- Language neutral
- Platform neutral
- Extensible
- Backward compatible

Example : setup

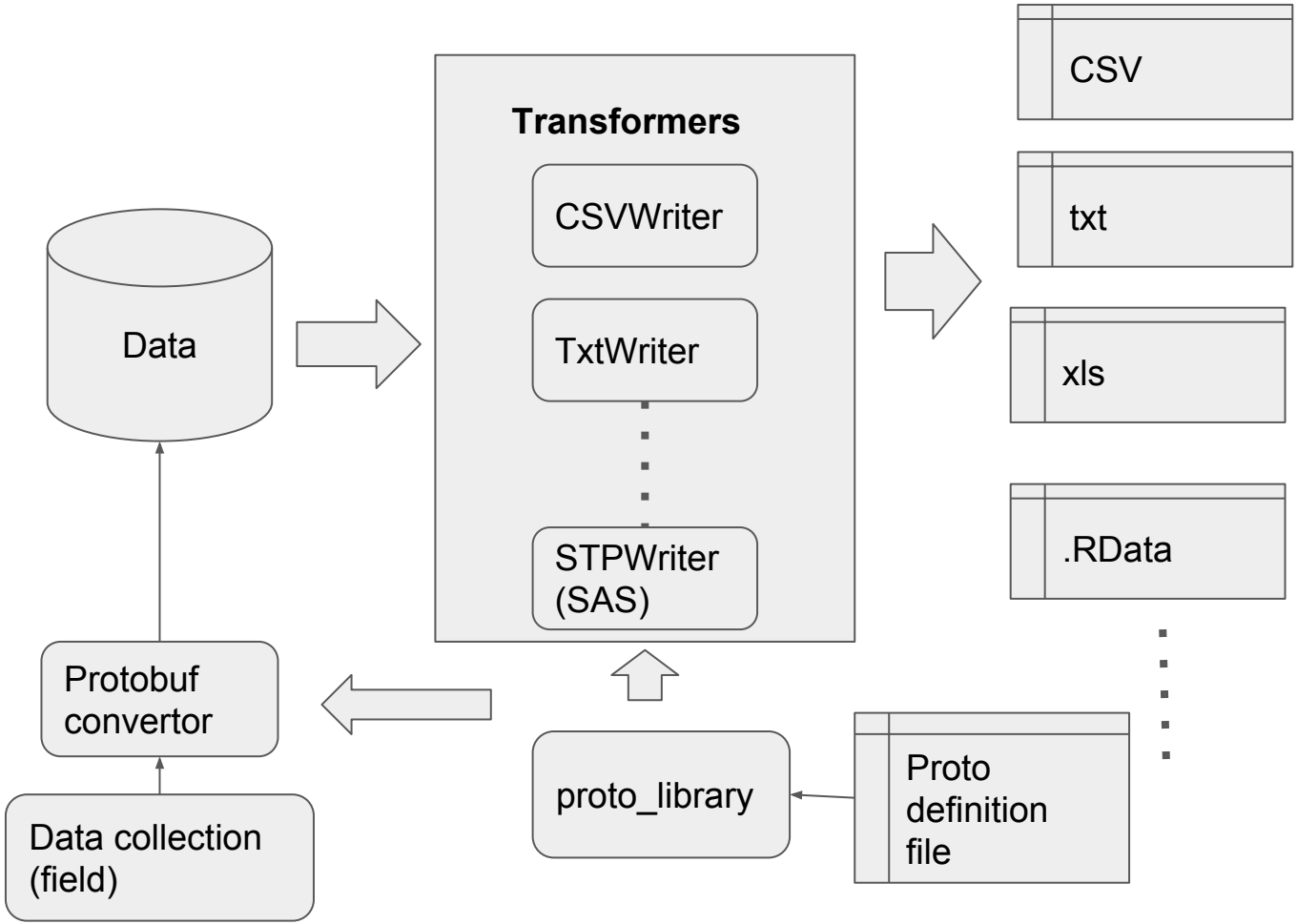
```
package demo;

Message person {
   required string name = 1;
   optional string email = 2;
};
```

Example contd.



https://developers.google.com/protocol-buffers/docs/downloads

https://github.com/google/protobuf/releases


Transformers
CSVWriter
TxtWriter
STPWriter
(SAS)
Data
CSV
txt
xls
.RData
Protobuf
convertor
Data collection
(field)
proto_library
Proto
definition
file

Cross platform access

C++
```
demo::Person person;
person.ParseFromString(...);
std::cout<< person.name();
```

Java
```
Demo.Person.Builder person =
Demo.Person.newBuilder();
…
System.out.println(person.getName());
```

Python
```
person = demo.Person()
 f = open(sys.argv[1], "rb")
person.ParseFromString(f)
print person.name
```

C#
```
Person person = new Person
using (var input
File.OpenRead("person.dat"))
{
    person =
Person.Parser.ParseFrom(input);
...
}
```

Raw data -> model features

```
Class FeatureGenerator {
Public:
 FeatureGenerator(constTrialData& ..)

 Feature1Type GetFeature1() const;
…..

private:
const TrialDataPB raw_data_;
};
```

Usecase : Adverse event definition (CTCAE)

```
Message SOC {
  Enum Grade {
    Grade1 =1;
    Grade2 = 2;
    ….
  };
  optional AnemiaSOCPB anemia;
  optional HemolysisSOCPB hemolysis;
  ….
};
```

SOC::Grade GetGrade(const AnemiaSOCPB& anemia) { …. }

Handling spec evolution

```
Message SOC {
  Enum Grade {
    ….
  };
  optional MyocardialInfaction myocardial_infaction = ..;
  ….
};
```

(Spec_ver4.cc)

```
SOC::Grade GetGrade(const MyocardialInfaction& msg) {
 const string term = CanonicalizeCTCAETerm(msg.term());
 if (msg.term().empty()) return SOC::Grade1;
….
}
```

(Spec_ver5.cc)

```
SOC::Grade GetGrade(const MyocardialInfaction& msg) {
 const string term = CanonicalizeCTCAETerm(msg.term());
 if (term.equals(UNEXPLAINED_FAINTING) ) return SOC::Grade1;
….
```

v/s XML

`<person>`
`  <name>John Doe</name>`
`  <email>jdoe@example.com</email>`
`</person>`
	person {
	  name: "John Doe"
	  email: "jdoe@example.com"
	}

Less ambiguous.

Separates data definition and data - versioning, validation.

Programmatically accessible - test automation, data decoupling.

Structural validation - “strongly” typed format.

Faster and concise.

v/s JSON

“Person”: {
  “name”: "John Doe"
  “email”: "jdoe@example.com"
}
	# Textual representation of a protocol buffer data.
	person {
	  name: "John Doe"
	  email: "jdoe@example.com"
	}

Typed fields - validation

Data definition decoupled from data - robustness.

Extensibility/versioning

Ser-de libraries build with data definition.

Extending and versioning

```
Message person {
   required string name = 1;
   optional string email = 2;
   optional uint32 age = 3;
};
```

Features

- Types
- Message nesting
- Extensions
- Reflections

References

https://developers.google.com/protocol-buffers/

Encoding/serialization

Base128 varints in LSB order

Optional fields

Repeated fields

Field number/position