Survey of data formats and conversion tools

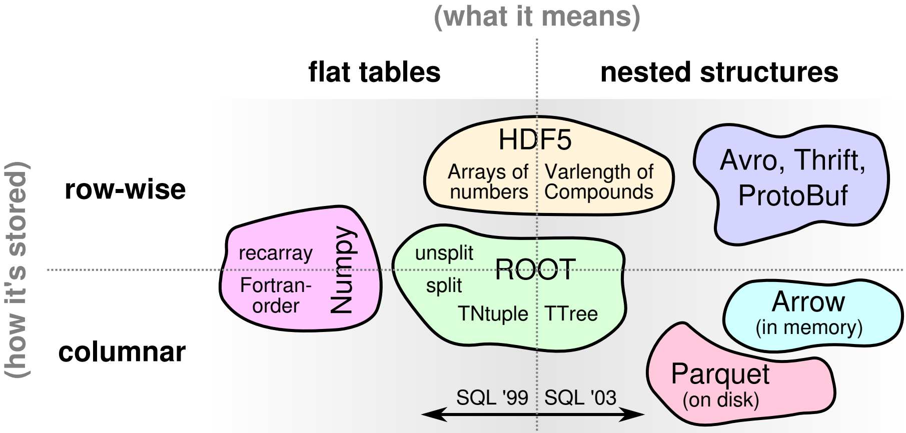
Jim Pivarski

Princeton University – DIANA

May 23, 2017

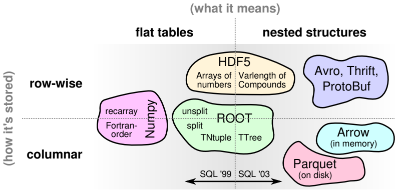
The landscape of generic containers

By "generic," I mean file formats that define general structures that we can specialize for particular kinds of data, like XML and JSON, but we're interested in binary formats with schemas for efficient numerical storage.



The landscape of generic containers

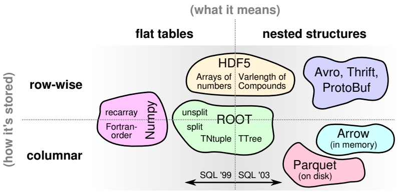
ROOT: can do anything with the right settings



The landscape of generic containers

ROOT: can do anything with the right settings

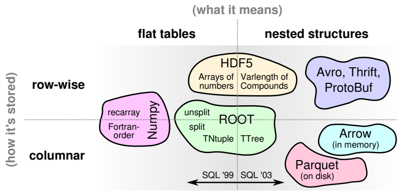
HDF5: stores block-arrays well, good for flat ntuples; can use variable-length arrays of compounds to store e.g. lists of particles, but not in an efficient, columnar way (?)



ROOT: can do anything with the right settings

HDF5: stores block-arrays well, good for flat ntuples; can use variable-length arrays of compounds to store e.g. lists of particles, but not in an efficient, columnar way (?)

Numpy: usually in-memory, also has an efficient file format; only for block-arrays (can hold Python objects, but not efficiently)



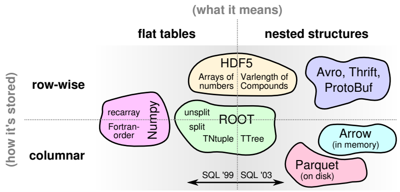
The landscape of generic containers

ROOT: can do anything with the right settings

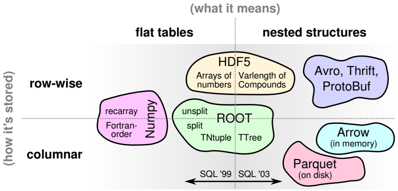
HDF5: stores block-arrays well, good for flat ntuples; can use variable-length arrays of compounds to store e.g. lists of particles, but not in an efficient, columnar way (?)

Numpy: usually in-memory, also has an efficient file format; only for block-arrays (can hold Python objects, but not efficiently)

Avro et al: interlingual, binary, deep structures with schemas, row-wise storage is best for streaming and RPC



ROOT:	can do anything with the right settings
HDF5:	stores block-arrays well, good for flat ntuples; can use variable-length arrays of compounds to store e.g. lists of particles, but not in an efficient, columnar way (?)
Numpy:	usually in-memory, also has an efficient file format; only for block-arrays (can hold Python objects, but not efficiently)
Avro et al:	interlingual, binary, deep structures with schemas, row-wise storage is best for streaming and RPC
Parquet:	extension of Avro et al with *columnar* storage, intended for databases and fast querying



The landscape of generic containers

ROOT: can do anything with the right settings

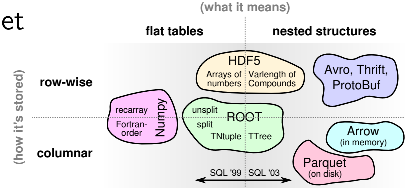
HDF5: stores block-arrays well, good for flat ntuples; can use variable-length arrays of compounds to store e.g. lists of particles, but not in an efficient, columnar way (?)

Numpy: usually in-memory, also has an efficient file format; only for block-arrays (can hold Python objects, but not efficiently)

Avro et al: interlingual, binary, deep structures with schemas, row-wise storage is best for streaming and RPC

Parquet: extension of Avro et al with *columnar* storage, intended for databases and fast querying

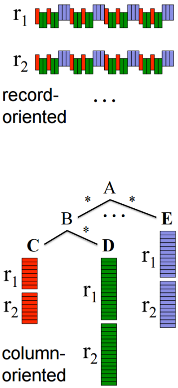
Arrow: in-memory extension of Parquet intended for zero-copy communication among databases, query servers, analysis frameworks, etc.



Is there a performance penalty?

- Formats differ most in how *nested* structure is represented:

  Avro: whole records are contiguous
  ROOT: each leaf is contiguous with list sizes in a separate array
  Parquet: each leaf is contiguous with depth in "repetition levels"



Is there a performance penalty?

- Formats differ most in how *nested* structure is represented:

  Avro: whole records are contiguous
  ROOT: each leaf is contiguous with list sizes in a separate array
  Parquet: each leaf is contiguous with depth in "repetition levels"

- Nevertheless, differences (with gzip/deflate) are only ∼15%. *Use-cases* may have more variation than *choice of format.*

47 407 $t\bar{t}$ Monte Carlo events in `TClonesArrays` or variable-length lists of custom classes.

ROOT 6.06, Avro 1.8.1, Parquet 1.8.1.

format	MB	rel.
ROOT none	399	1.96
ROOT gzip 1	204	1.00
ROOT gzip 2	208	1.02
ROOT gzip 9	202	0.99
Avro none	237	1.16
Avro snappy	198	0.97
Avro deflate	180	0.88
Avro LZMA	169	0.83
Parquet none	210	1.03
Parquet snappy	200	0.98
Parquet gzip	176	0.86

I don't have a grand study of all formats.

Is there a performance penalty?

- Formats differ most in how *nested* structure is represented:

  Avro: whole records are contiguous
  ROOT: each leaf is contiguous with list sizes in a separate array
  Parquet: each leaf is contiguous with depth in "repetition levels"

- Nevertheless, differences (with gzip/deflate) are only $\sim$15%. *Use-cases* may have more variation than *choice of format.*

- Speed depends more on runtime representation than file format.

  E.g. Avro's C library loads into its custom C objects in 113 sec; Avro's Java library in 8.3 sec! But if Avro's C library reads through the same row-wise data and fills *minimalist* objects, it's 5.4 sec.

47 407 $t\bar{t}$ Monte Carlo events in `TClonesArrays` or variable-length lists of custom classes.

ROOT 6.06, Avro 1.8.1, Parquet 1.8.1.

format	MB	rel.
ROOT none	399	1.96
ROOT gzip 1	204	1.00
ROOT gzip 2	208	1.02
ROOT gzip 9	202	0.99
Avro none	237	1.16
Avro snappy	198	0.97
Avro deflate	180	0.88
Avro LZMA	169	0.83
Parquet none	210	1.03
Parquet snappy	200	0.98
Parquet gzip	176	0.86

I don't have a grand study of all formats.

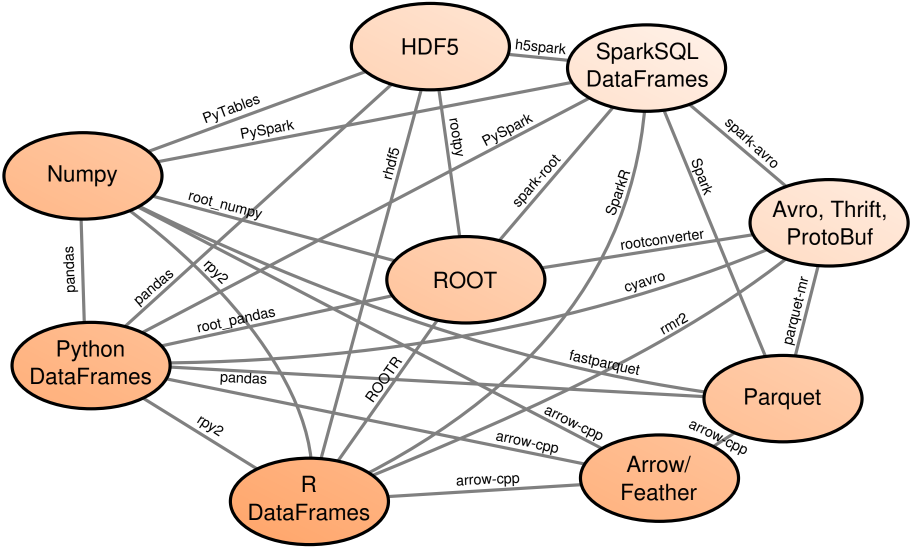
What matters is what you'll use it with

ROOT	is the best way to access petabytes of HEP data and use tools developed in HEP
HDF5	is the best way to use tools developed in other sciences, particuarly R, MATLAB, HPC
Numpy	is the best way to use the scientific Python ecosystem, particularly recent machine learning software
Avro et al	is the best way to use the Hadoop ecosystem, particularly streaming frameworks like Storm
Parquet	is the best way to use database-like tools in the Hadoop ecosystem, such as SparkSQL
Arrow	is in its infancy, but is already a good way to share data between Python (Pandas) DataFrames and R DataFrames

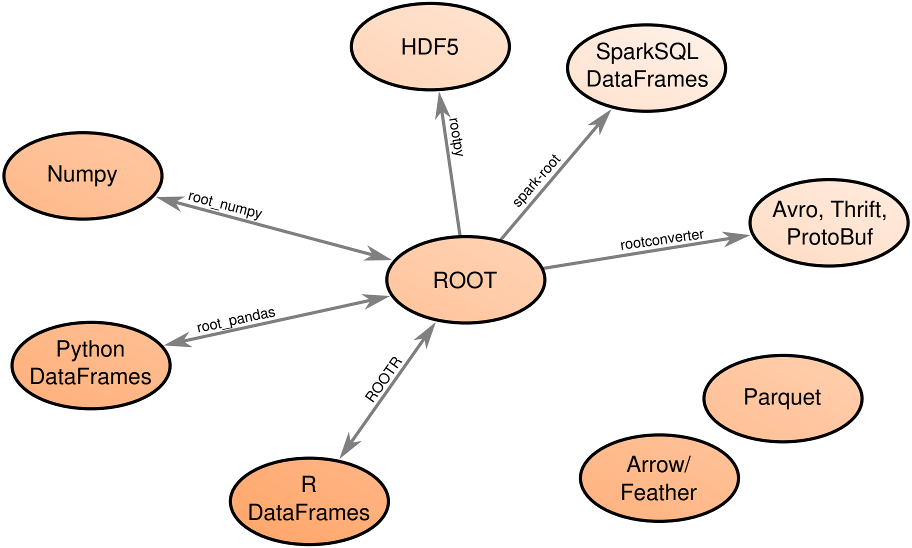
What matters is what you'll use it with

ROOT	is the best way to access petabytes of HEP data and use tools developed in HEP
HDF5	is the best way to use tools developed in other sciences, particuarly R, MATLAB, HPC
Numpy	is the best way to use the scientific Python ecosystem, particularly recent machine learning software
Avro et al	is the best way to use the Hadoop ecosystem, particularly streaming frameworks like Storm
Parquet	is the best way to use database-like tools in the Hadoop ecosystem, such as SparkSQL
Arrow	is in its infancy, but is already a good way to share data between Python (Pandas) DataFrames and R DataFrames

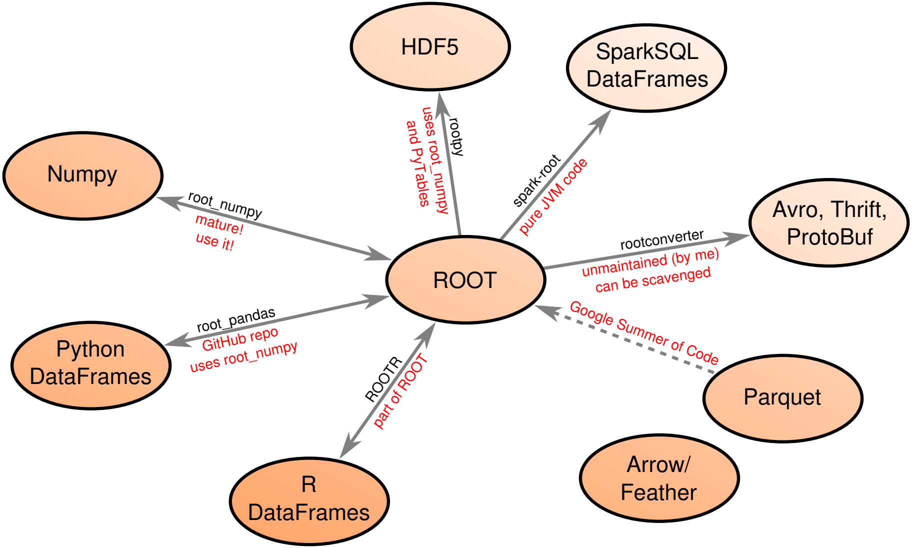
Conversions: getting from here to there



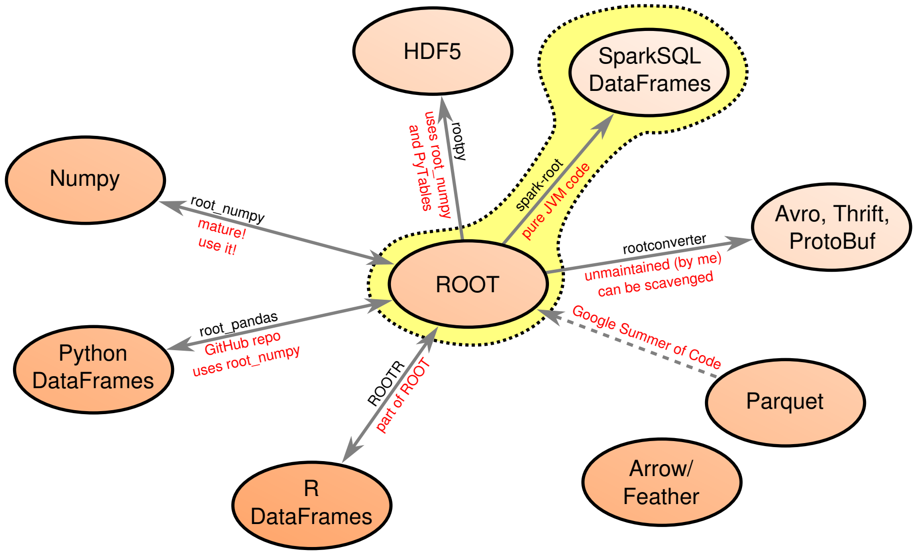
Conversions: getting from here to there



Conversions: getting from here to there



Conversions: getting from here to there



ROOT is now a Spark DataFrame format

Launch Spark with JARs from Maven Central (zero install).

```
pyspark --packages org.diana-hep:spark-root_2.11:0.1.11
```

Access the ROOT files as you would any other DataFrame.

```
df = sqlContext.read \
       .format("org.dianahep.sparkroot") \
       .load("hdfs://path/to/files/*.root")

df.printSchema()
root
 |-- met: float (nullable = false)
 |-- muons: array (nullable = false)
 |    |-- element: struct (containsNull = false)
 |    |    |-- pt: float (nullable = false)
 |    |    |-- eta: float (nullable = false)
 |    |    |-- phi: float (nullable = false)
 |-- jets: array (nullable = false)
 |    |-- element: struct (containsNull = false)
 |    |    |-- pt: float (nullable = false)
```

ROOT is now a Spark DataFrame format

Launch Spark with JARs from Maven Central (zero install).

```
pyspark --packages org.diana-hep:spark-root_2.11:0.1.11
```

Access the ROOT files as you would any other DataFrame.

```
df = sqlContext.read \
       .format("org.dianahep.sparkroot") \
       .load("hdfs://path/to/files/*.root")
```

```
df.printSchema()
root
 |-- met: float (nullable = false)
 |-- muons: array (nullable = false)
 |    |-- element: struct (containsNull = false)
 |    |    |-- pt: float (nullable = false)
 |    |    |-- eta: float (nullable = false)
 |    |    |-- phi: float (nullable = false)
 |-- jets: array (nullable = false)
 |    |-- element: struct (containsNull = false)
 |    |    |-- pt: float (nullable = false)
```

abstract type system!

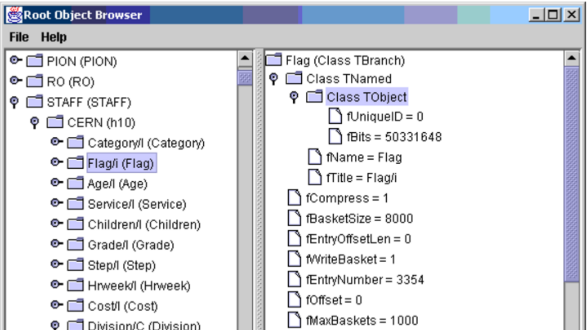
FreeHEP ROOTIO

Root Object Browser

As an illustration of the use of the Java interface, we have built a sample application which is a simple Root Object Browser. It can be used to open any Root file and look at all the objects inside the file. If you already have Java 2 installed (JDK 1.3), you can download the root.jar file containing the application, and run it using the command:

```
java -jar root.jar
```

(on Windows you can just double-click on the root.jar file). A screen shot of the application is show below. The pane on the left shows the directory structure of the file. The object browser knows how to navigate directories (TDirectories), trees (TTrees and TBranches) and these will all be shown in the left pane. Clicking on any object in the left pane will cause the details of the object to be shown in the right pane. The right pane knows how to follow embedded pointers to other objects.



FreeHEP ROOTIO

Last Published: 2013-03-01 | Version: 2.2.1 FreeHEP | JAS | WIRED

General
- **Introduction**
- License
- Team

User Info
- Summary
- API Doc
- Jar File(s)
- Dependencies
- Forum
- Bug Reports

Developer Info
- Source Code

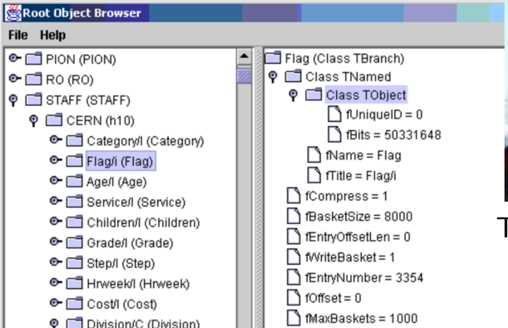
Built by: maven

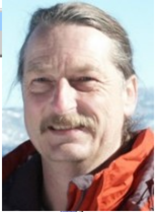
Root Object Browser

As an illustration of the use of the Java interface, we have built a sample application which is a simple Root Object Browser. It can be used to open any Root file and look at all the objects inside the file. If you already have Java 2 installed (JDK 1.3), you can download the root.jar file containing the application, and run it using the command:

```
java -jar root.jar
```

(on Windows you can just double-click on the root.jar file). A screen shot of the application is show below. The pane on the left shows the directory structure of the file. The object browser knows how to navigate directories (TDirectories), trees (TTrees and TBranches) and these will all be shown in the left pane. Clicking on any object in the left pane will cause the details of the object to be shown in the right pane. The right pane knows how to follow embedded pointers to other objects.





Tony Johnson
SLAC

This repository Search Pull requests Issues Gist

diana-hep / **root4j**

Watch 10 | Star 2 | Fork 2

Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

A fork of http://java.freehep.org/freehep-rootio/ with hooks for Spark DataFrames

Edit

Add topics

45 commits | **2** branches | **2** releases | **2** contributors | LGPL-2.1

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

vkhristenko making hadoop as provided dependency — Latest commit 2a7bd47 on Mar 15

src	fixing issues with string and other minor updates	3 months ago
.gitignore	updating gitignore	6 months ago
DATAFORMATS.md	updating data format description	4 months ago
LICENSE	Initial commit	6 months ago
README.md	updated readme	6 months ago
pom.xml	making hadoop as provided dependency	2 months ago

README.md

ROOT4J

A fork of http://java.freehep.org/freehep-rootio/

This repository Search Pull requests Issues Gist

diana-hep / **root4j**

Watch 10 | Star 2 | Fork 2

Code | Issues 1 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

A fork of http://java.freehep.org/freehep-rootio/ with hooks for Spark DataFrames

Edit

Add topics

45 commits | **2** branches | **2** releases | **2** contributors | LGPL-2.1

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

vkhristenko making hadoop as provided dependency — Latest commit 2a7bd47 on Mar 15

src	fixing issues with string and other minor updates	3 months ago
.gitignore	updating gitignore	6 months ago
DATAFORMATS.md	updating data format description	4 months ago
LICENSE	Initial commit	6 months ago
README.md	updated readme	6 months ago
pom.xml	making hadoop as provided dependency	2 months ago

README.md

ROOT4J

A fork of http://java.freehep.org/freehep-rootio/

Viktor Khristenko
University of Iowa

ROOT I/O implementations

As far as I'm aware, root4j is one of only five ROOT-readers:

standard ROOT	C++	
JsRoot	JavaScript	for web browsers
root4j	Java	can't link Java and ROOT
RIO in GEANT	C++	to minimize dependencies
go-hep	Go	to use Go's concurrency

ROOT I/O implementations

As far as I'm aware, root4j is one of only five ROOT-readers:

standard ROOT	C++	
JsRoot	JavaScript	for web browsers
root4j	Java	can't link Java and ROOT
RIO in GEANT	C++	to minimize dependencies
go-hep	Go	to use Go's concurrency

Most file formats are fully specified in the abstract and then re-implemented in dozens of languages.

But ROOT is much more complex than most file formats.

Dreaming. . .

Serialization mini-languages:

DSLs that transform byte sequence → abstract data types

- Construct: http://construct.readthedocs.io
- PADS: http://pads.cs.tufts.edu
- PacketTypes, DataScript, the "Power of Pi" paper. . .

Dreaming...

Serialization mini-languages:

DSLs that transform byte sequence $\to$ abstract data types

- Construct: http://construct.readthedocs.io
- PADS: http://pads.cs.tufts.edu
- PacketTypes, DataScript, the "Power of Pi" paper...

Even if these languages are too limited to implement ROOT I/O, could it be implemented in a *translatable* subset of a common language?

Dreaming...

Serialization mini-languages:

DSLs that transform byte sequence → abstract data types

- Construct: http://construct.readthedocs.io
- PADS: http://pads.cs.tufts.edu
- PacketTypes, DataScript, the "Power of Pi" paper...

Even if these languages are too limited to implement ROOT I/O, could it be implemented in a *translatable* subset of a common language?

That is, a disciplined use of minimal language features, allowing for automated translation to C, Javascript, Java, Go...?