Data Processing for CLAS12

Gagik Gavalian (Jefferson National Laboratory)

Introduction

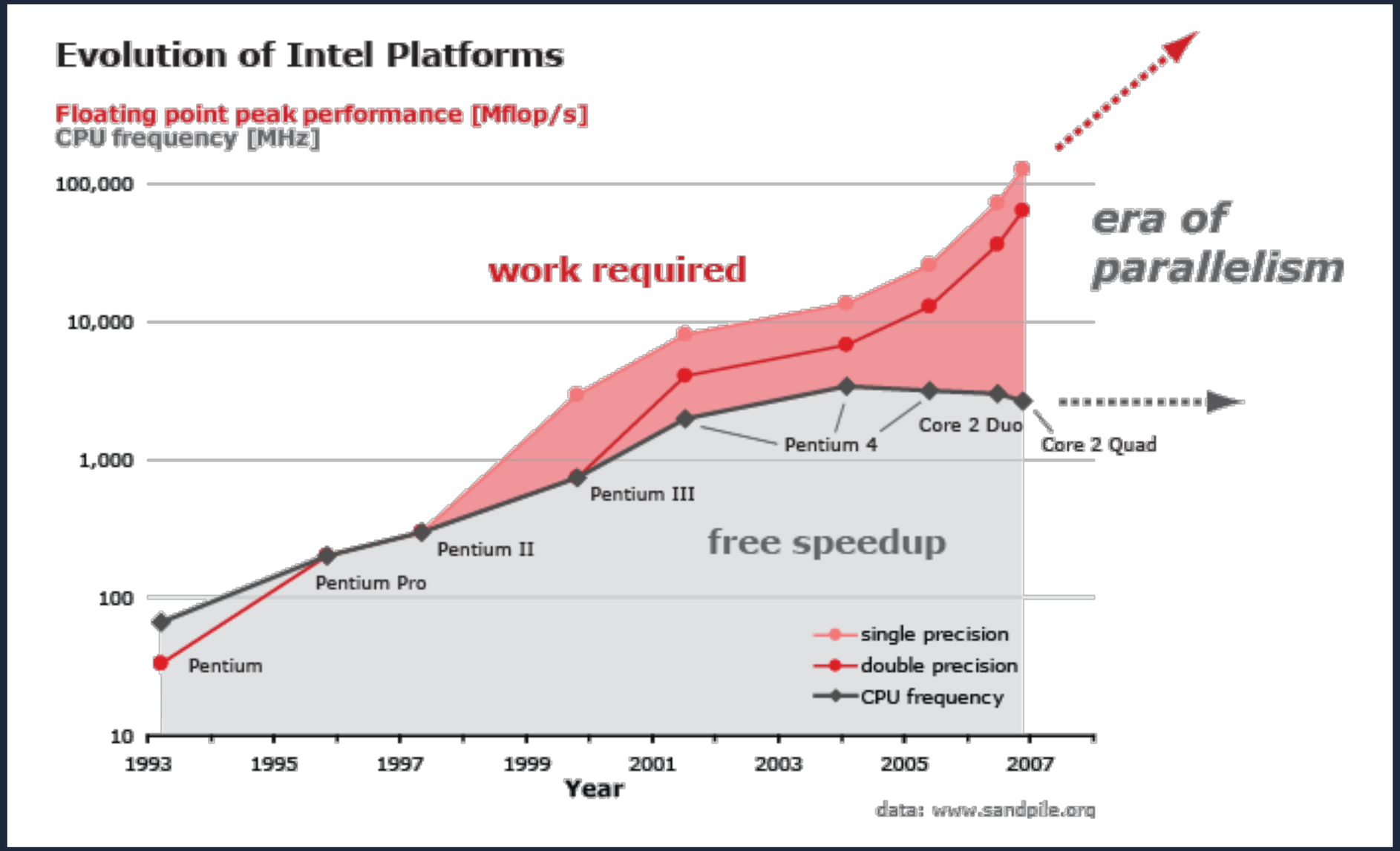
CONVENTIONAL WISDOM

- If things worked well 20 years ago, we should follow the same path.
- Analysis tools that we used for past 30 years are just fine, why reinvent ?
- The data we are writing we can analyze with our existing tools, so what if it's 1000 times larger data ?

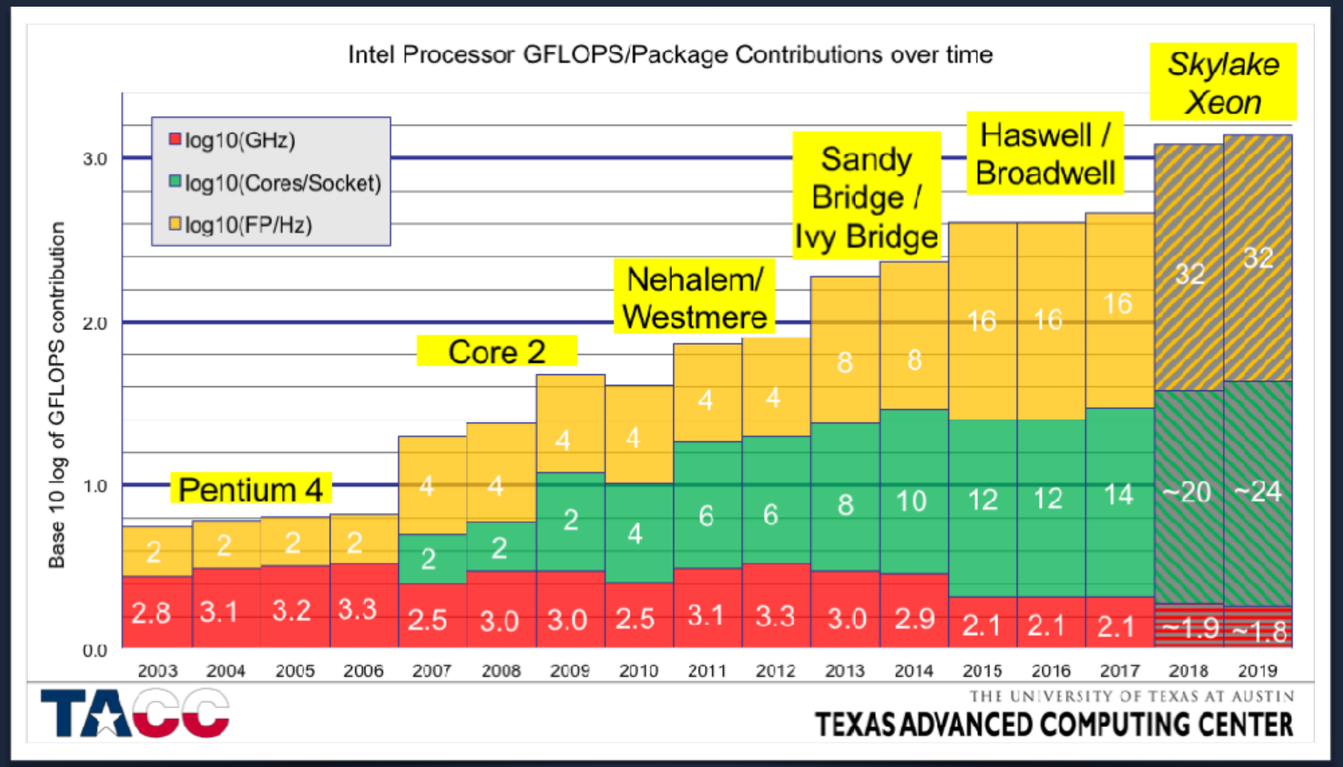
QUESTIONS

- Is computing evolving the same way it was 20 years ago ?
- Software architecture models have to change ?
- Do we need new approaches to data storage and distribution ?

Evolution of Computing



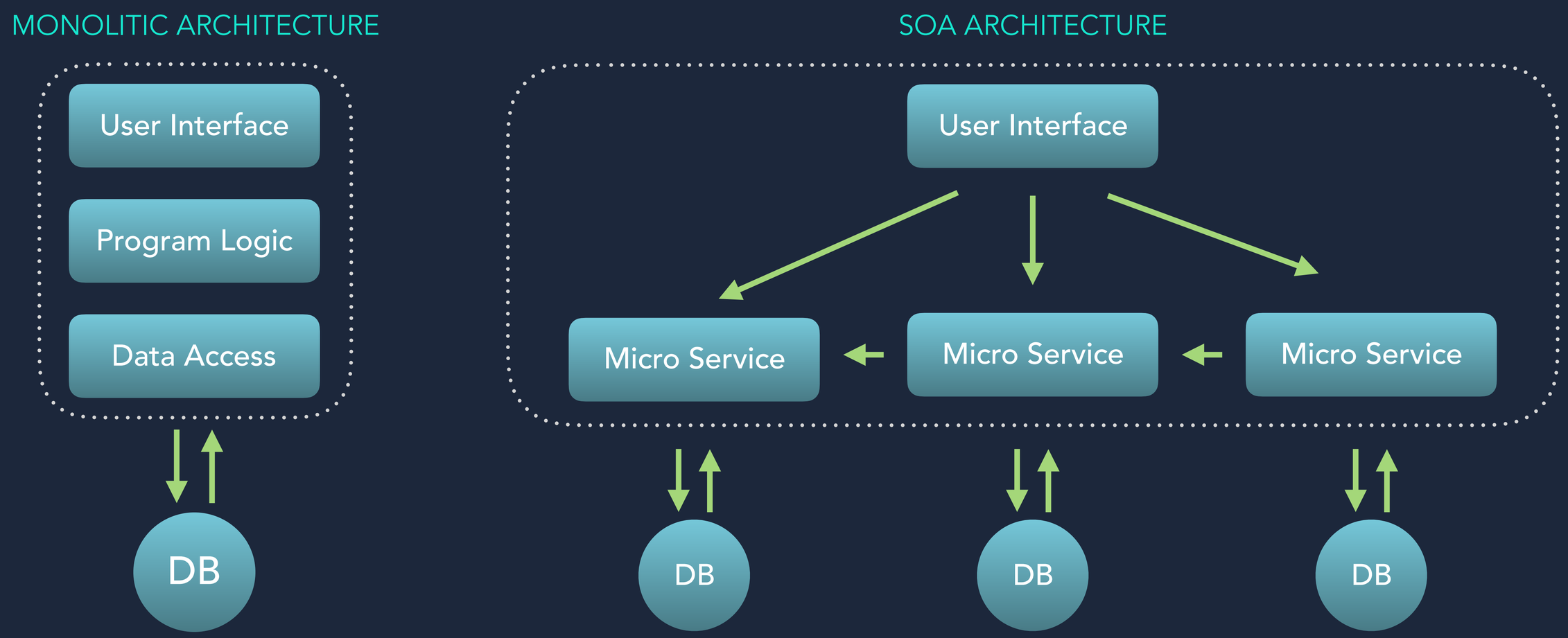
Evolution of Computing



- Major software engineering methods and tools currently focus on sequential software development.
- However, every developer is now confronted with parallel programming applications such as server applications.
- If application is not parallel performance can not be improved by additional cores.
- As more cores are integrated on the same chip, clock rates may decrease, sequential applications will be slower with new processor generation.

- The whole spectrum of software engineering – from design over testing to maintenance – has to be revisited in the light of parallelism. Nondeterminism adds a new dimension of complexity.
- Many parts of the existing software stack require revisions, each providing challenges and opportunities for parallelism exploration.

Service Oriented Architecture (SOA)



Monolitic Approach

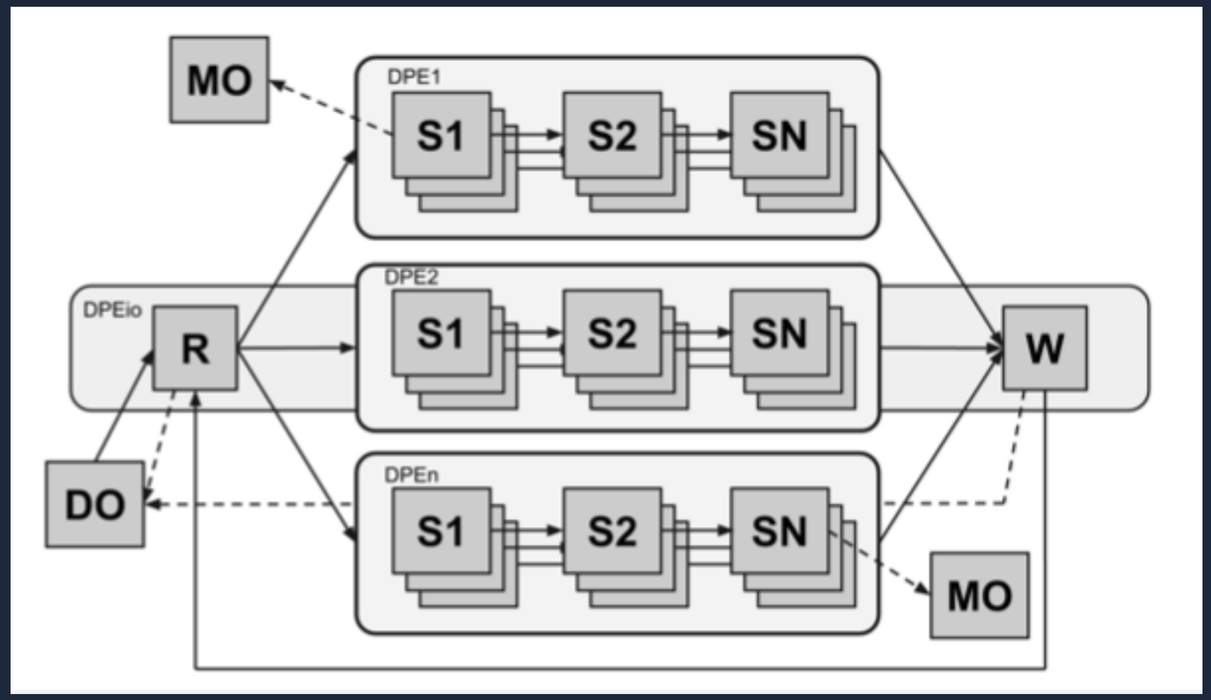
Provides a program based on the libraries (API) to accomplish a specific task. If different task had to be performed one has to re-write the program.

SOA Approach

Provides Services that can do one specific task. User can build and application by chaining services, and passing data through the chain and accomplishing different tasks.

Service Oriented Architecture (SOA) (https://claraweb.jlab.org/clara/)

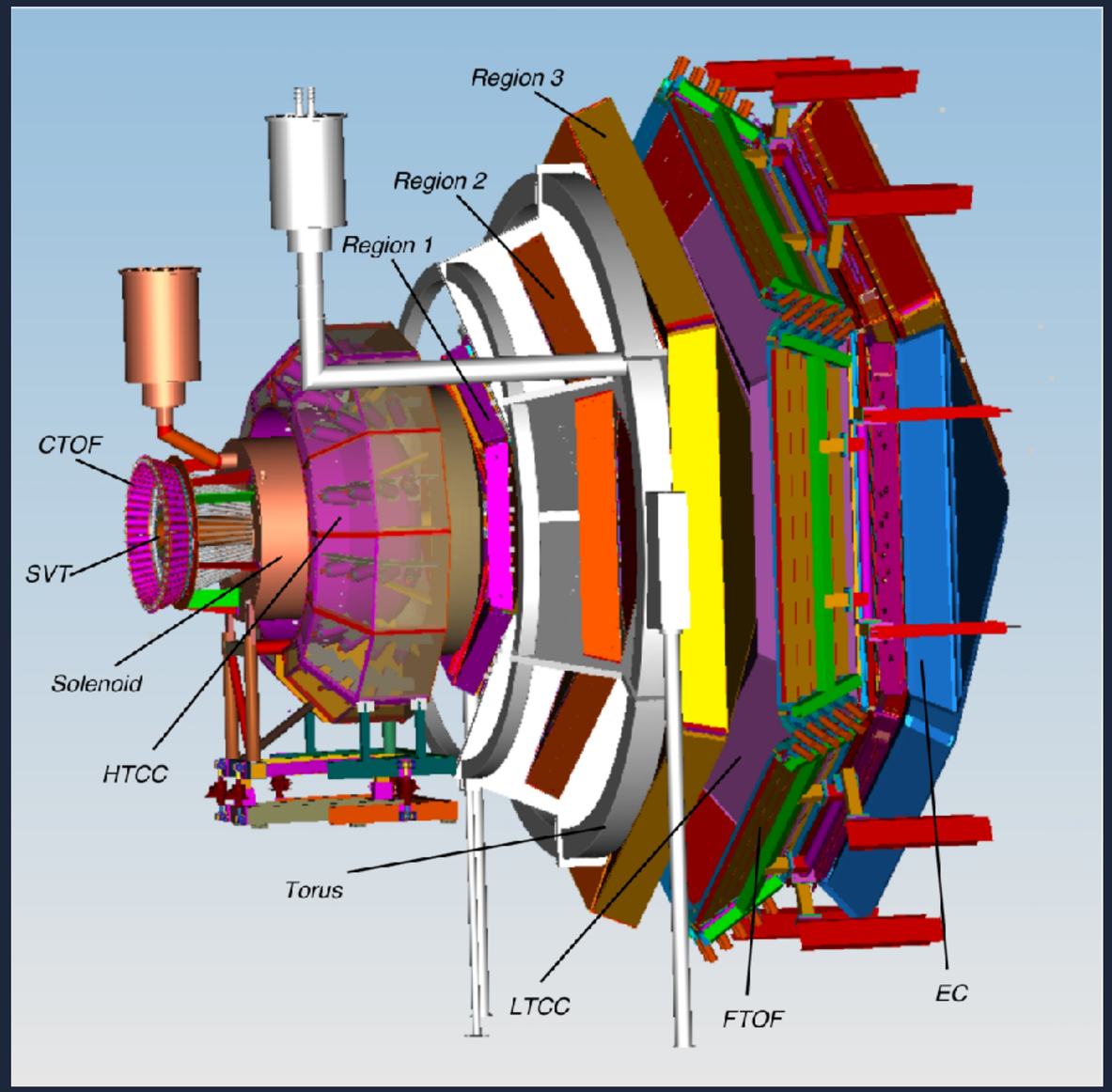
- CLARA Framework aims to enhance the efficiency agility and productivity PDP processes.
- CLARA is an approach to develop data processing applications based on the concept of multiple asynchronous processes (Services)
- The Application in CLARA is system of data streams being transformed by multithreaded services.



- Service Composition is comprised of services that have been assembled to provide functionality required to accomplish a specific data processing task:
  - Data Reconstruction
  - Event filtering
  - Kinematic Fitter
- CLARA makes a clear separation between the service programmer and the data application designer.

- The Application is essentially asynchronous data passing through selected services.
- The data transforms depending on the service.
- Application Orchestrator takes care of data flow optimization depending on how many cores on the system.
- Application can also run on several machines and pass data through network.
- Data passing is based on P2P technology and is agnostic where the services run.

CLAS 12 Detector



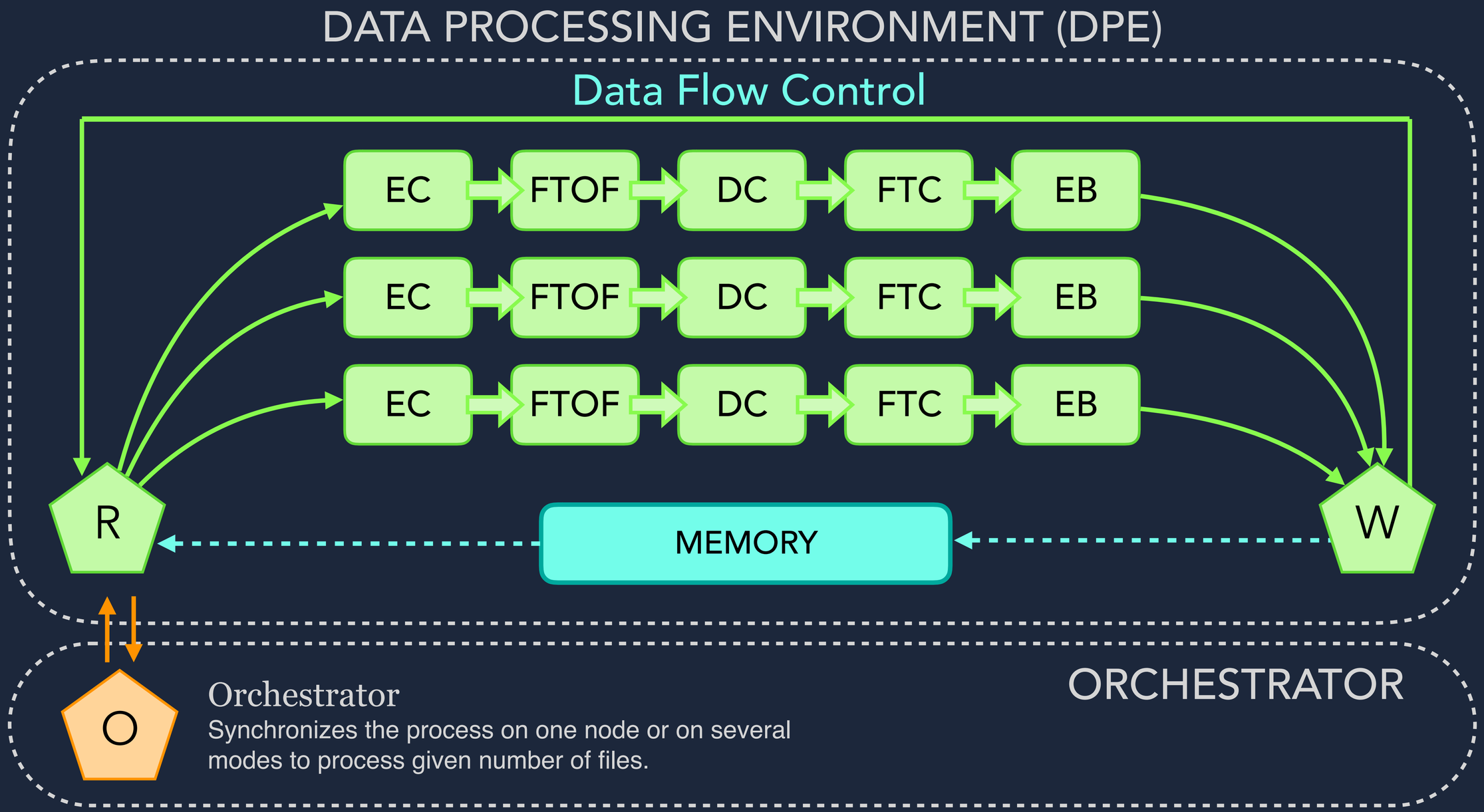
DETECTOR COMPOSITION:

- Drift Chamber inside Toroidal field for forward tacks.
- Electromagnetic Calorimeter for electron identification and neutral particle detector.
- Time of Flight system for particle identification.
- High Threshold Cherenkov Detector for electron pion rejection.
- Silicon tracker for central detector charged particle tracking in Solenoidal Filed.
- Central Neutron Detector for neutron identification.

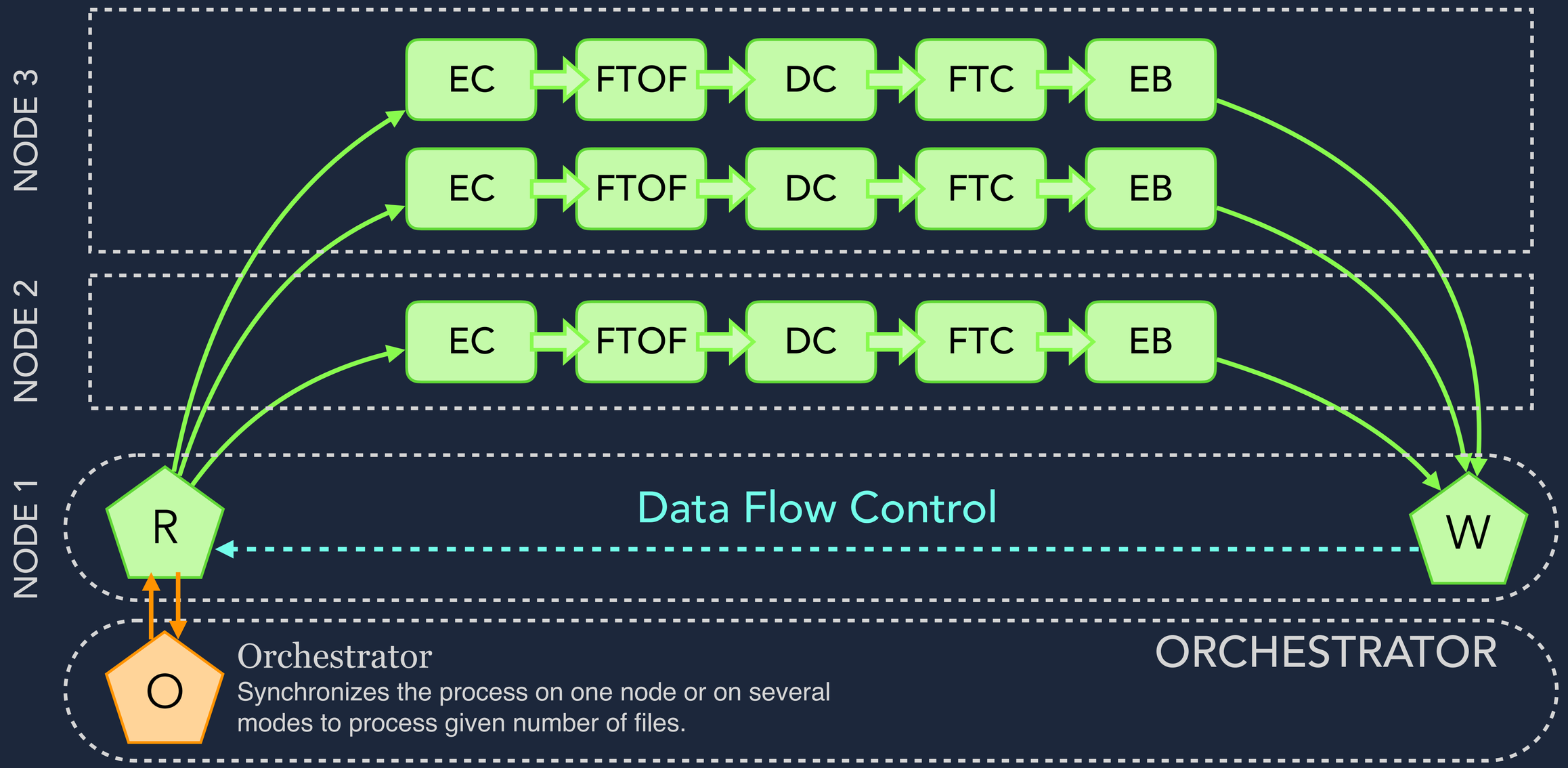
DATA ACQUISITION:

- DAQ data rate 12 kHz,
- Data rate 400 Mb/sec
- Up-to-Date collected ~1.2 Pb

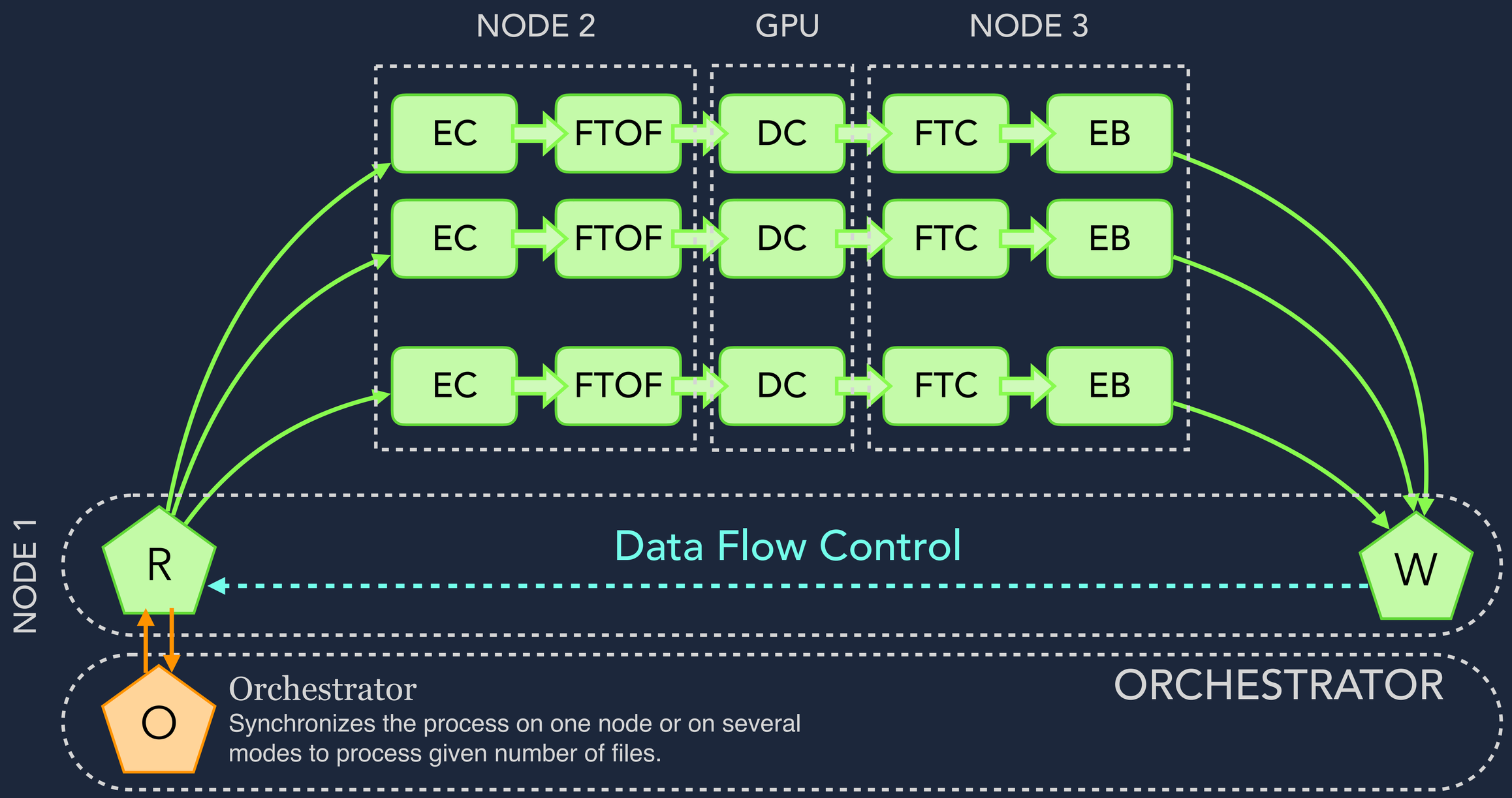
CLARA Data Processing



CLAS12 Data Processing (Vertical Scaling)



CLAS12 Data Processing (Multi-Platform Scaling)



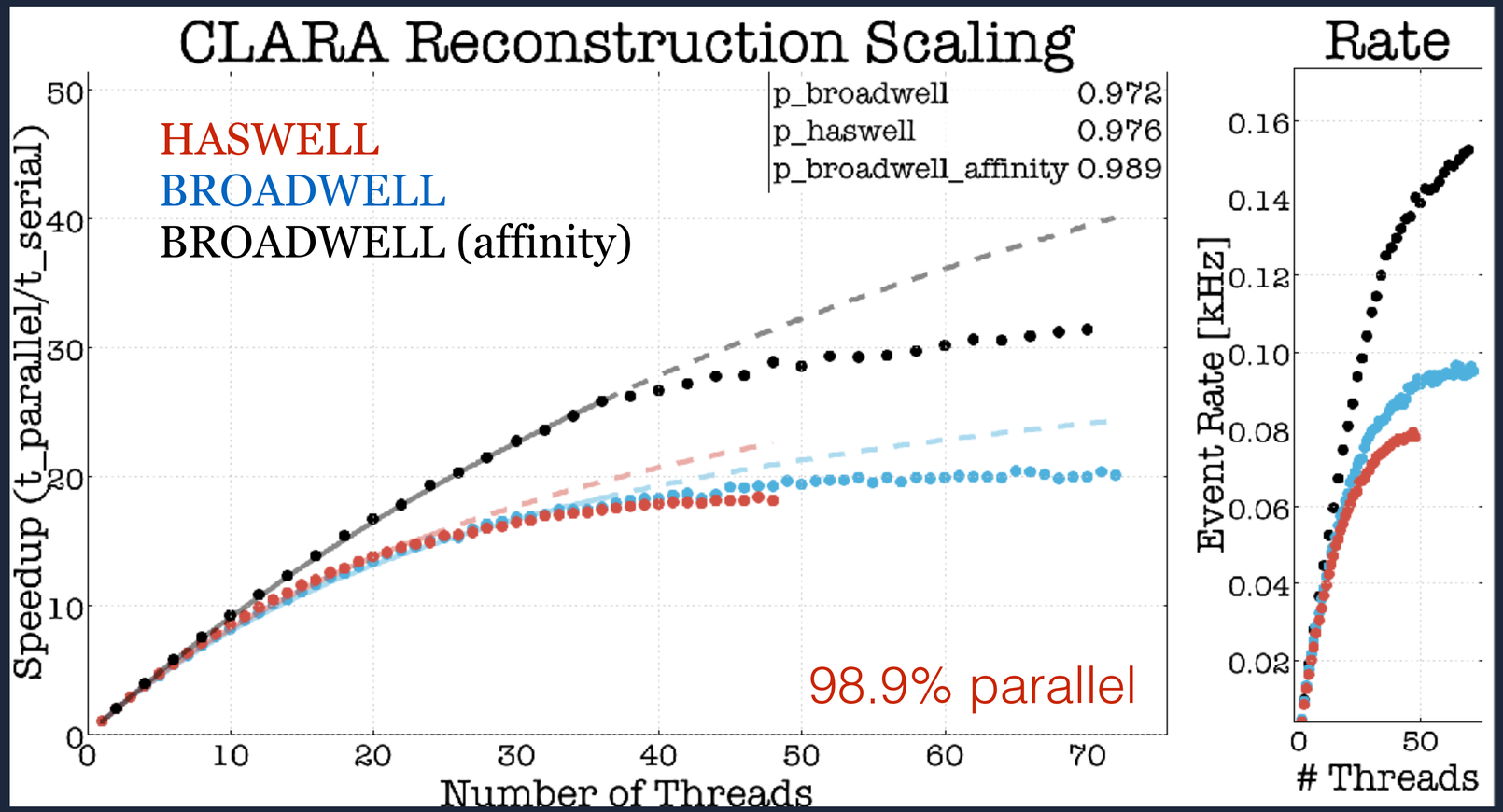
CLAS12 Data Processing

Amdahl's law

gives the theoretical speedup in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved.

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

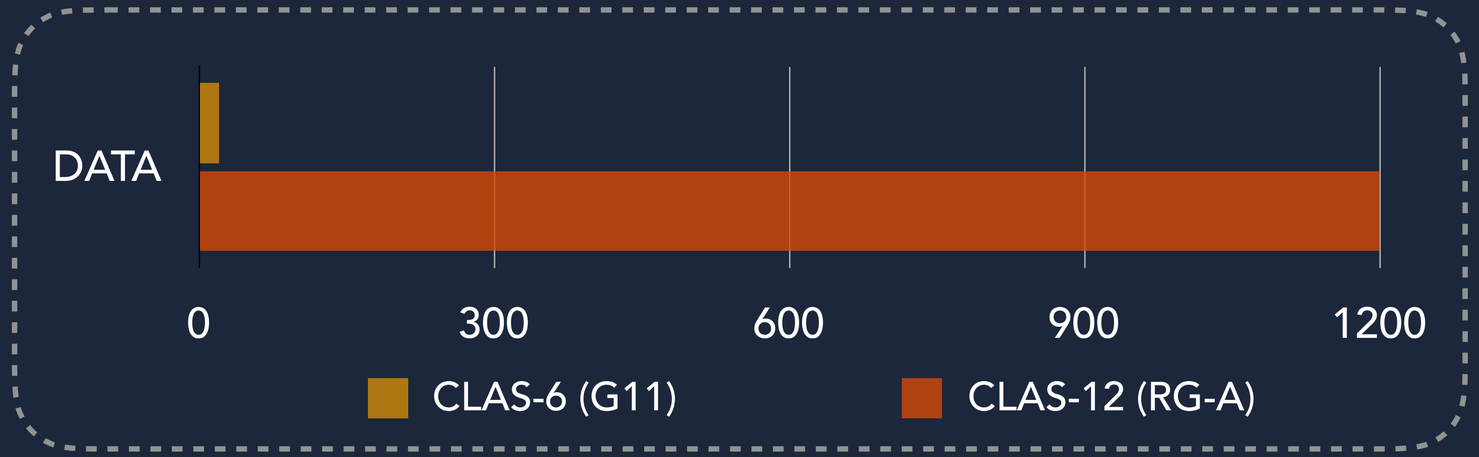
p - is the percentage of the execution time of the whole task concerning the part that benefits from the improvement of the resources of the system before the improvement.
s - is the speedup in latency of the execution of the part of the task that benefits from the improvement of the resources of the system

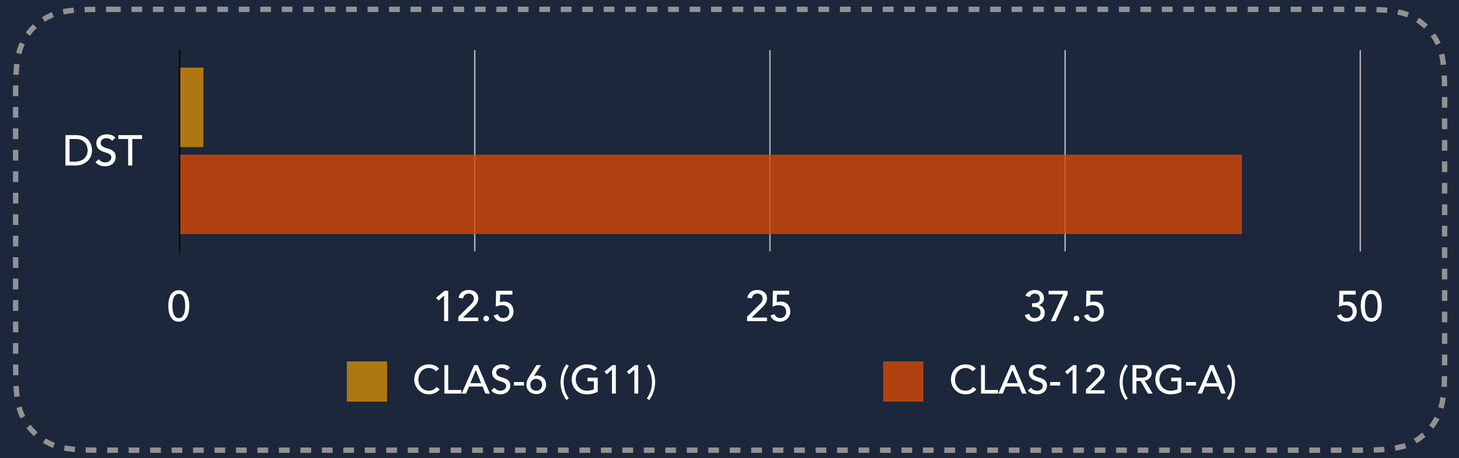


Thread Affinity

Operating system forces the thread to run on one specific core without changing to the available idle cores. Switching requires data to move along with the process causing slow down.

CLAS 12 Detector





DATA SIZES:

- CLAS12 First Experiment collected ~50x more data than longest experiment in CLAS6 era (**~1.12 PB**)
- Data Summary tapes are **~100x** larger (45 TB) than G11 experiment (on the graph compressed size is shown)

Data Formats (Column Store vs Row Store)

- In Row store data are stored on the disk tuple by tuple (Apache Avro, port Buffer)
- In Column store data are stored on the disk column by column (ROOT, Apache Parquet)

- Most of the queries does not process all the attributes of a particular relation.
- For example the query

  Select c.name and c.address
  From CUSTOMER as c
  Where c.region=Mumbai;
- Only process three attributes of the relation CUSTOMER. But the customer relation can have more than three attributes.
- Column-stores are more I/O efficient for read-only queries as they read, only those attributes which are accessed by a query.

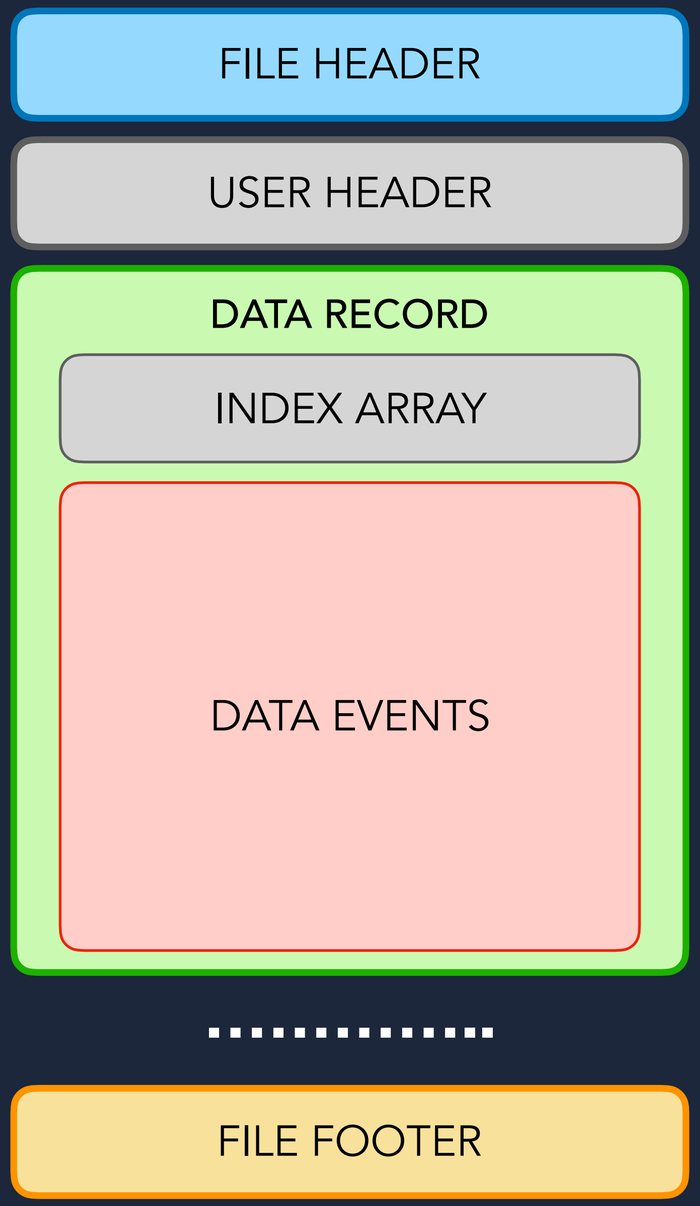
Why Column Stores ?

- Can be significantly faster than row stores for some applications
  - Fetch only required columns for a query
  - Better cache effects
  - Better compression (similar attribute values within a column)

- Physics Applications:
  - Need to read entire tuple to do analysis
  - Not all events are used in the analysis
  - New approach is needed to group tuples that have similar properties
  - Ability to index the data inside the file for accessing only certain tuples

- Row Stores:
  - Easier to create chunks of data (Data Frames)
  - Better suited to multi-threaded applications

HIPO Data Format (File Format)



User Header

Contains information about the record dictionary, format. User specified parameters related to conditions of the experiment.

Data Record

Compressed buffer of data consisting of events and index. Record header provides number of events and the TAG for the record. Data records are typically ~8 MB.

Index Array

Array of event offsets inside the event buffer. Dynamically creates event random access table.

FILE FOOTER

Contains positions of every record in the file with number of events for fast random access. Also has tags for each Data Record.

Table Relations (One-to-One)

- Reconstruction results are written in tables for each detector system
- This information is then referenced from other “parent” banks.

pid	px	py	pz	d-15	d-16	d-17	d-18	d-19
11	0.723	1.567	2.419	0	1	2	0	0
211	1.103	0.891	2.725	0	3	0	0	0
2212	0.923	0.456	1.982	0	4	0	5	0
22	1.082	1.324	1.792	0	0	6	0	0

index	detector	x	y	z	time	energy
1	16	234.5	456.8	725.2	29.56	0.124
1	17	10.4	618.2	645.9	32.65	0.236
2	16	376.9	0.456	549.4	30.12	0.892
3	16	420.5	132.6	398.1	28.74	0.512
3	18	103.6	238.9	654.6	29.14	0.398
4	17	217.8	251.6	708.2	31.89	0.457

- In One-to-One relation one index in parent table points to one row in the child table.
- If the child table is dropped from the data files the parent table is left with redundant information.

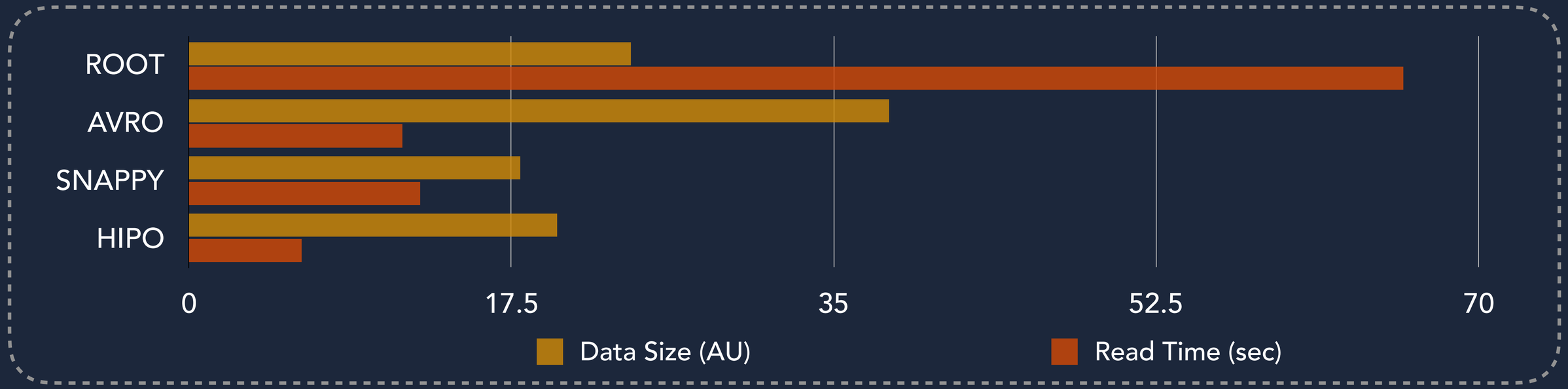
Table Relations (Many-to-One)

- In Many-to-one scheme many rows from dependent table point to one entry in parent table
- By dropping dependent table, no redundant information is kept.
- In case of One-to-One by dropping dependent banks redundant pointers in the parent table are still there.

pid	px	py	pz
11	0.723	1.567	2.419
211	1.103	0.891	2.725
2212	0.923	0.456	1.982
22	1.082	1.324	1.792

index	detector	x	y	z	time	energy
1	16	234.5	456.8	725.2	29.56	0.124
1	17	10.4	618.2	645.9	32.65	0.236
2	16	376.9	0.456	549.4	30.12	0.892
3	16	420.5	132.6	398.1	28.74	0.512
3	18	103.6	238.9	654.6	29.14	0.398
4	17	217.8	251.6	708.2	31.89	0.457

CLAS 12 Detector



HIGH PERFORMANCE OUTPUT (HIPO):

- HIPO data format was implemented for CLAS12
- Dictionary driven data format with Schema evolution.
- Fast compression algorithm (LZ4)
- Chunked data frame implementation to support multithreaded applications.
- Event chunk tagging and indexing to allow reading selective types of events from Data Summary Tapes.
- Deserialization performance better than most commonly used data formats in Nuclear Physics.
- Faster than industry standard data types.

Data Formats

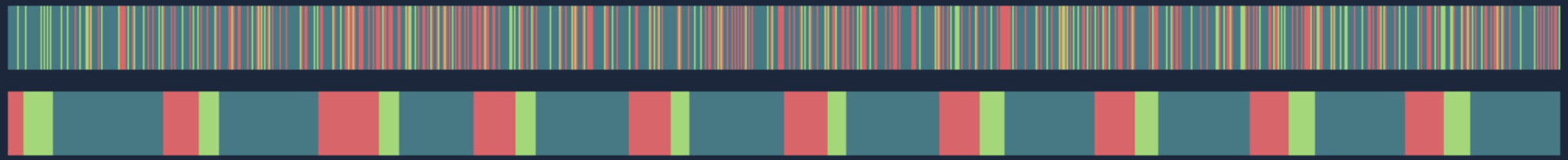
CONVENTIONAL APPROACH

- Conventional data formats write data event after event as they come out from reconstruction.
- Each event has different topology of final state.
- Not all events are useful for given analysis.
- User has to read entire data set to determine which events are useful for his analysis.
- Not suitable for big data.

NEW APPROACH

- HIPO data format is record based format with full indexed file structure.
- Records are tagged according user given criteria.
- Reading specific tags from data file is possible.
- Filtering data does not require parsing events in the records, so each skimming operation is on the level of disk IO.

EVENT TOPOLOGY FROM CLAS12 (RUN #3856)



59.7% Trigger particle is not an electron. No electron Forward Tagger.

25.6% Electron trigger. Forward Detector

14.7% Forward Tagger No Electron in ECAL

API vs FRAMEWORK

API (application programming interface)
is a term meaning the functions/methods in a library that you can call to ask it to do things for you - the interface to the library.

FRAMEWORK
is a big library that provides many services (rather than perhaps only one focussed ability as most libraries do).

Common Approach
if we put together enough API libraries we got a framework.

API (application programming interface)
Write HOW to do it.

Framework
Write WHAT to do.

API vs FRAMEWORK

```
provider = CcdbPackage.createProvider(address);
 Assignment asgmt = provider.getData(table_name);
 int ncolumns = asgmt.getColumnCount();
 Vector<TypeTableColumn> typecolumn = asgmt.getTypeTable().getColumns();

 String[] format = new String[ncolumns-nindex];

 for(int loop = nindex; loop < ncolumns; loop++){
     //System.out.println("COLUMN " + typecolumn.get(loop).getName()
     //        + "  " + typecolumn.get(loop).getCellType().name());
     if(typecolumn.get(loop).getCellType().name().compareTo("DOUBLE")==0){
         format[loop-nindex] = typecolumn.get(loop).getName() + "/D";
     } else {
         format[loop-nindex] = typecolumn.get(loop).getName() + "/I";
     }
     //format[loop-3] =
 }

 IndexedTable  table = new IndexedTable(nindex,format);
 for(int i = 0; i < nindex; i++){
     table.setIndexName(i, typecolumn.get(i).getName());
 }
 //table.show();


 List< Vector<String> >  tableRows = new ArrayList< Vector<String> >();

 for(int loop = 0; loop < ncolumns; loop++){
     String name = typecolumn.get(loop).getName();
         Vector<String> column = asgmt.getColumnValuesString(name);
         tableRows.add(column);
 }

 int nrows = tableRows.get(0).size();

 for(int nr = 0 ; nr < nrows; nr++){
     String[] values = new String[ncolumns];
     for(int nc = 0; nc < ncolumns; nc++){
         values[nc] = tableRows.get(nc).get(nr);
     }
     /*System.out.println("\n LENGTH = " + values.length);
     for(String item : values){
         System.out.print(item + "  :  ");
     }*/
     //table.addRow(values);
     table.addEntryFromString(values);
 }
```

- This code was present in every micro-Service.
- Each user code either reconstruction or calibration has to have entire parsing of database query results.
- Some users forgot to close database connection.
- Some mistakes while copy pasting code.

API vs FRAMEWORK

```
provider = CcdbPackage.createProvider(address);
 Assignment asgmt = provider.getData(table_name);
 int ncolumns = asgmt.getColumnCount();
 Vector<TypeTableColumn> typecolumn = asgmt.getTypeTable().getColumns();

 String[] format = new String[ncolumns-nindex];

 for(int loop = nindex; loop < ncolumns; loop++){
     //System.out.println("COLUMN " + typecolumn.get(loop).getName()
     //        + "  " + typecolumn.get(loop).getCellType().name());
     if(typecolumn.get(loop).getCellType().name().compareTo("DOUBLE")==0){
         format[loop-nindex] = typecolumn.get(loop).getName() + "/D";
     } else {
         format[loop-nindex] = typecolumn.get(loop).getName() + "/I";
     }
     //format[loop-3] =
 }

 IndexedTable  table = new IndexedTable(nindex,format);
 for(int i = 0; i < nindex; i++){
     table.setIndexName(i, typecolumn.get(i).getName());
 }
 //table.show();


 List< Vector<String> >  tableRows = new ArrayList< Vector<String> >();

 for(int loop = 0; loop < ncolumns; loop++){
     String name = typecolumn.get(loop).getName();
         Vector<String> column = asgmt.getColumnValuesString(name);
         tableRows.add(column);
 }

 int nrows = tableRows.get(0).size();

 for(int nr = 0 ; nr < nrows; nr++){
     String[] values = new String[ncolumns];
     for(int nc = 0; nc < ncolumns; nc++){
         values[nc] = tableRows.get(nc).get(nr);
     }
     /*System.out.println("\n LENGTH = " + values.length);
     for(String item : values){
         System.out.print(item + "  :  ");
     }*/
     //table.addRow(values);
     table.addEntryFromString(values);
 }
```

- This code was present in every micro-Service.
- Each user code either reconstruction or calibration has to have entire parsing of database query results.
- Some users forgot to close database connection.
- Some mistakes while copy pasting code.

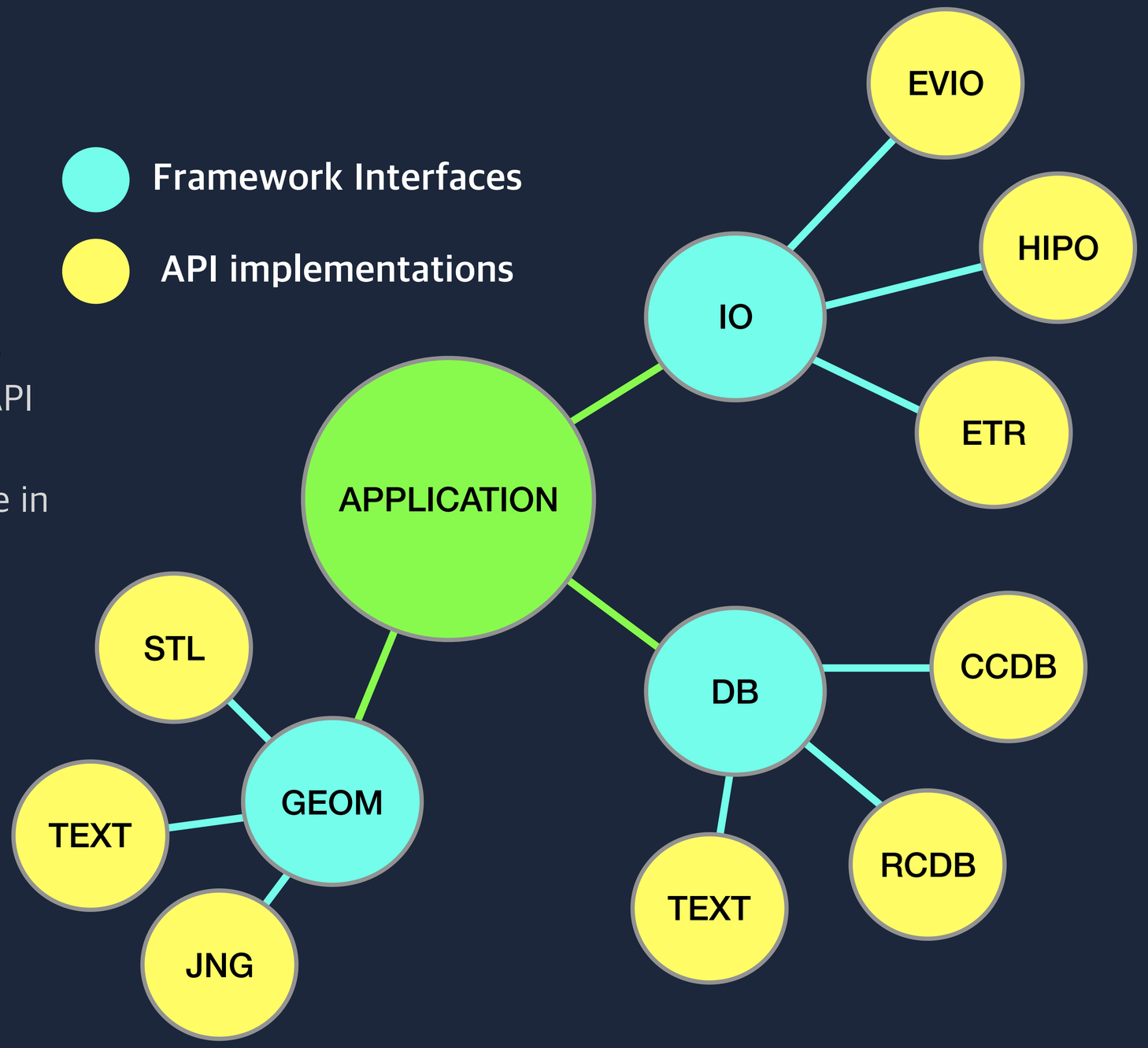
Framework:

```
public void init(){
 requireConstants("/calib/ftof/attenuation");
}

public void processEvent(DataEvent event){
 IndexedTable table = getConstants("/calib/ftof/attenuation");
}
```

- Easy to change database implementation
- Framework puts lock for the multithreaded processes
- Readable and maintainable code.

API vs FRAMEWORK

- Framework provides interfaces of behavior.
- Several implementation are used through API to provide behavior.
- Mistakes and bugs can be fixed in one place in the code base.
- User application code is readable and maintainable.
- Extending functionality is easy.



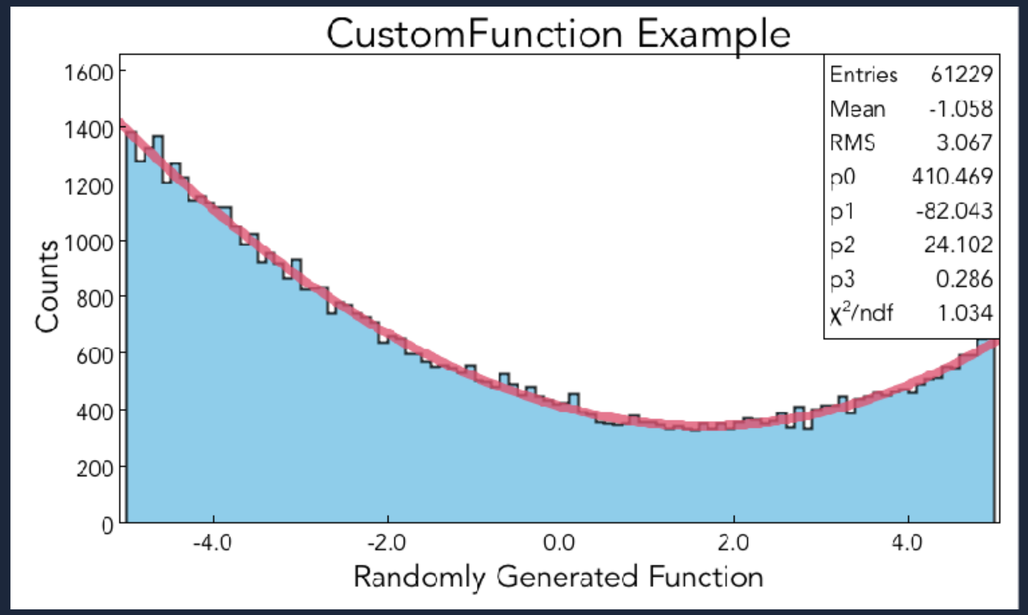
CLAS12 Framework

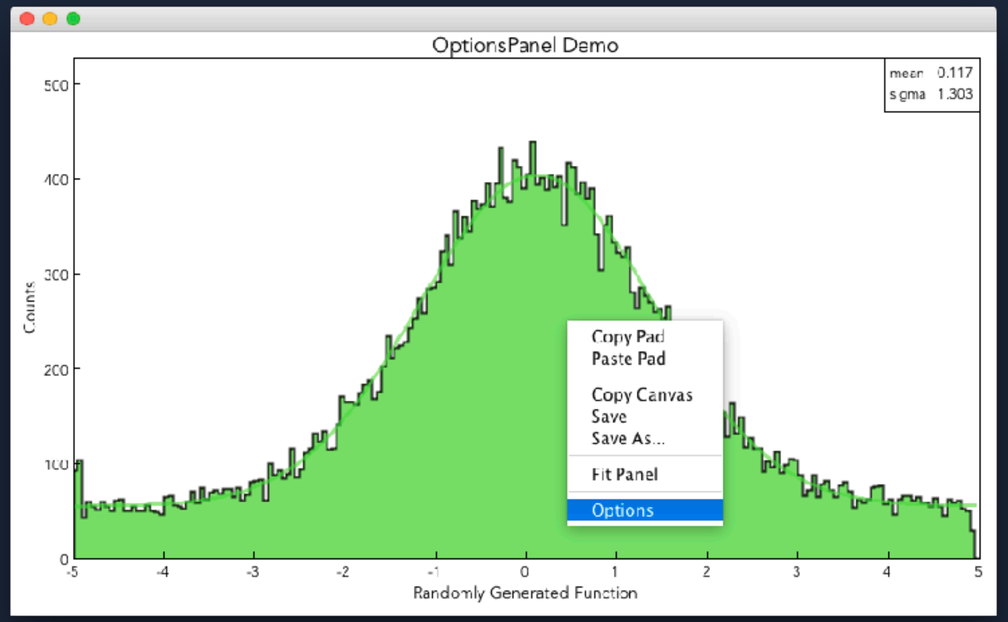
- CLAS12 reconstruction, calibration and monitoring software based on CLARA
- Software is written in JAVA, which represents some challenges:
  - No Data Analysis tools for Nuclear Physics (like ROOT)
  - No Data Visualization (plotting) packages (there are some for specific applications)
  - No familiar data I/O (like ROOT Ntuples)

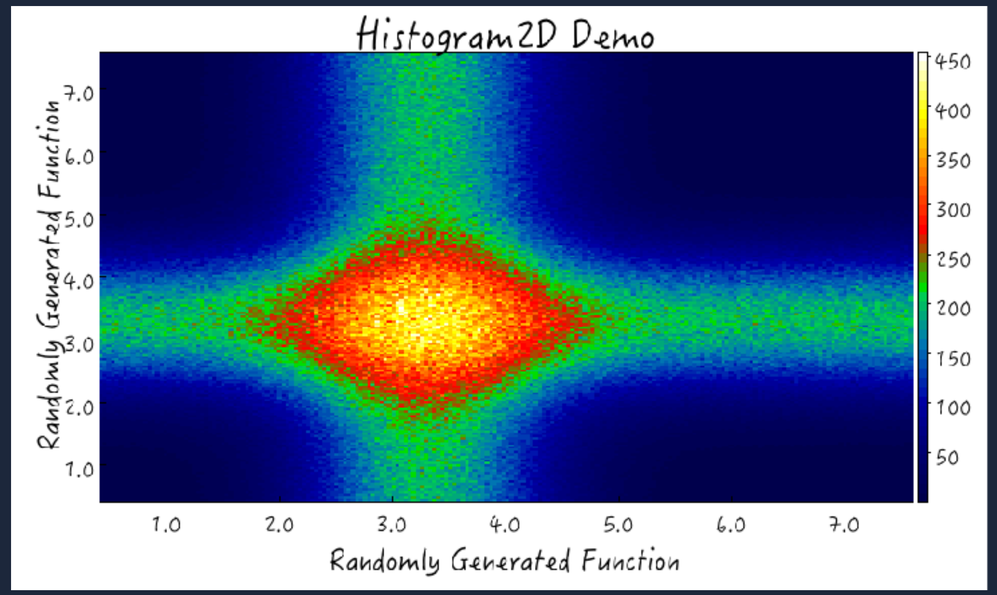
- We had to implement software for familiar data plotting and analysis software (GROOT).
- Includes data fitting (with MUNUIT)
- Data storage software was implemented which has implementations in JAVA, C++, FORTRAN
- Interactive data plotting environment was developed for quick data analysis in form of tuples.
- Monitoring and calibration software suites use GROOT
- Majority of data analysis are done in JAVA, using HIPO and GROOT

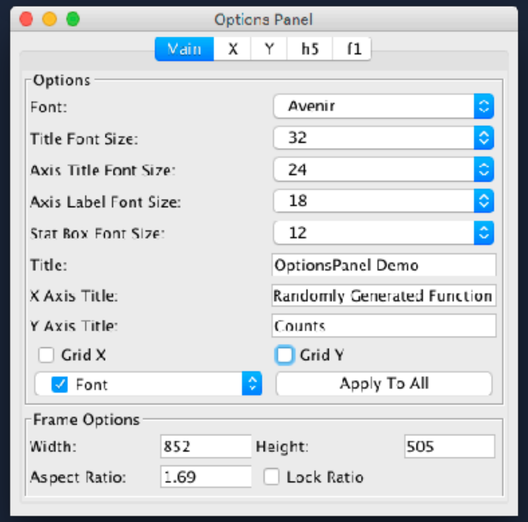
https://github.com/gavalian/groot

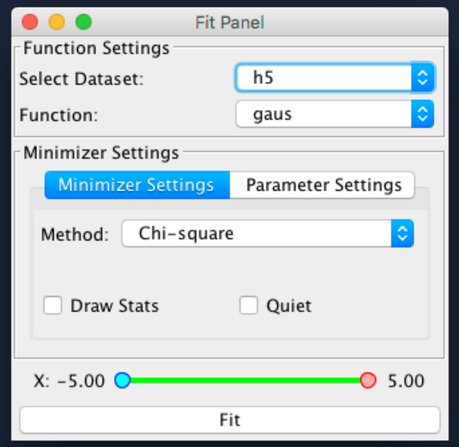
Data Analysis and Visualization







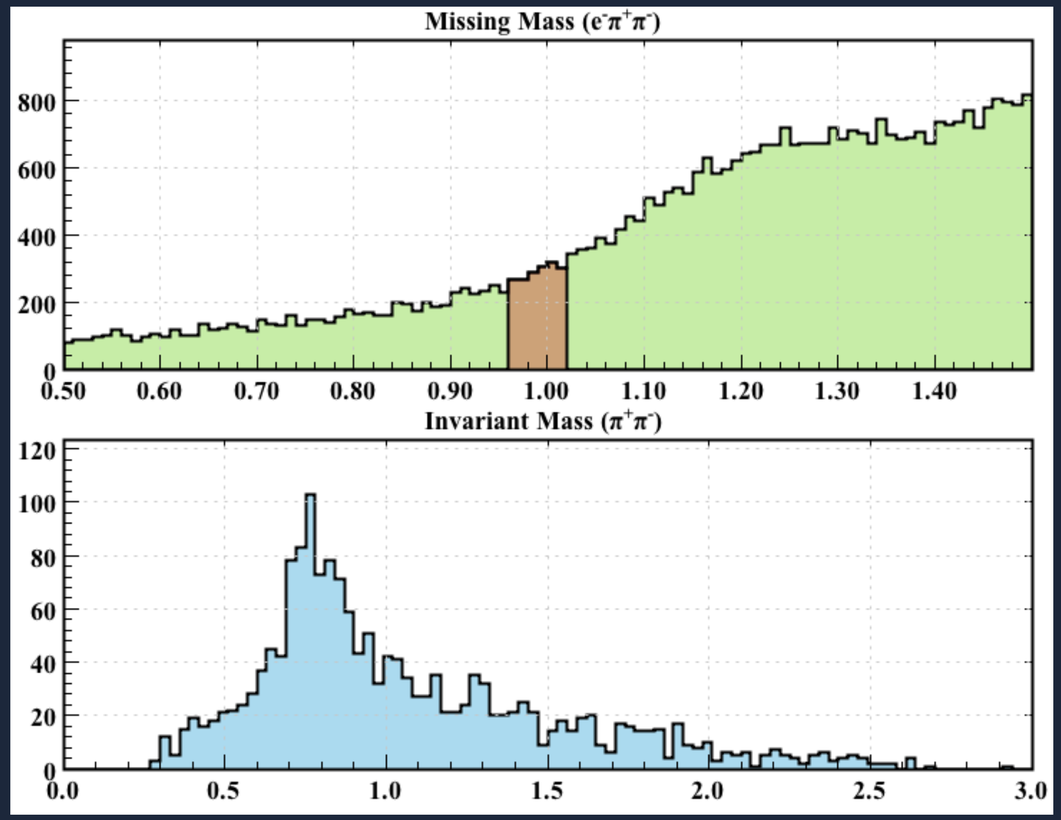




JAVA for Scientists

- JAVA has few appealing features that makes it useful for our field:
  - Portability, once compiled can be used on any machine
  - Binds naturally to Jython (Python for JAVA), no need to write special classes
  - Has scripting language (Groovy) that can be used off the shelf

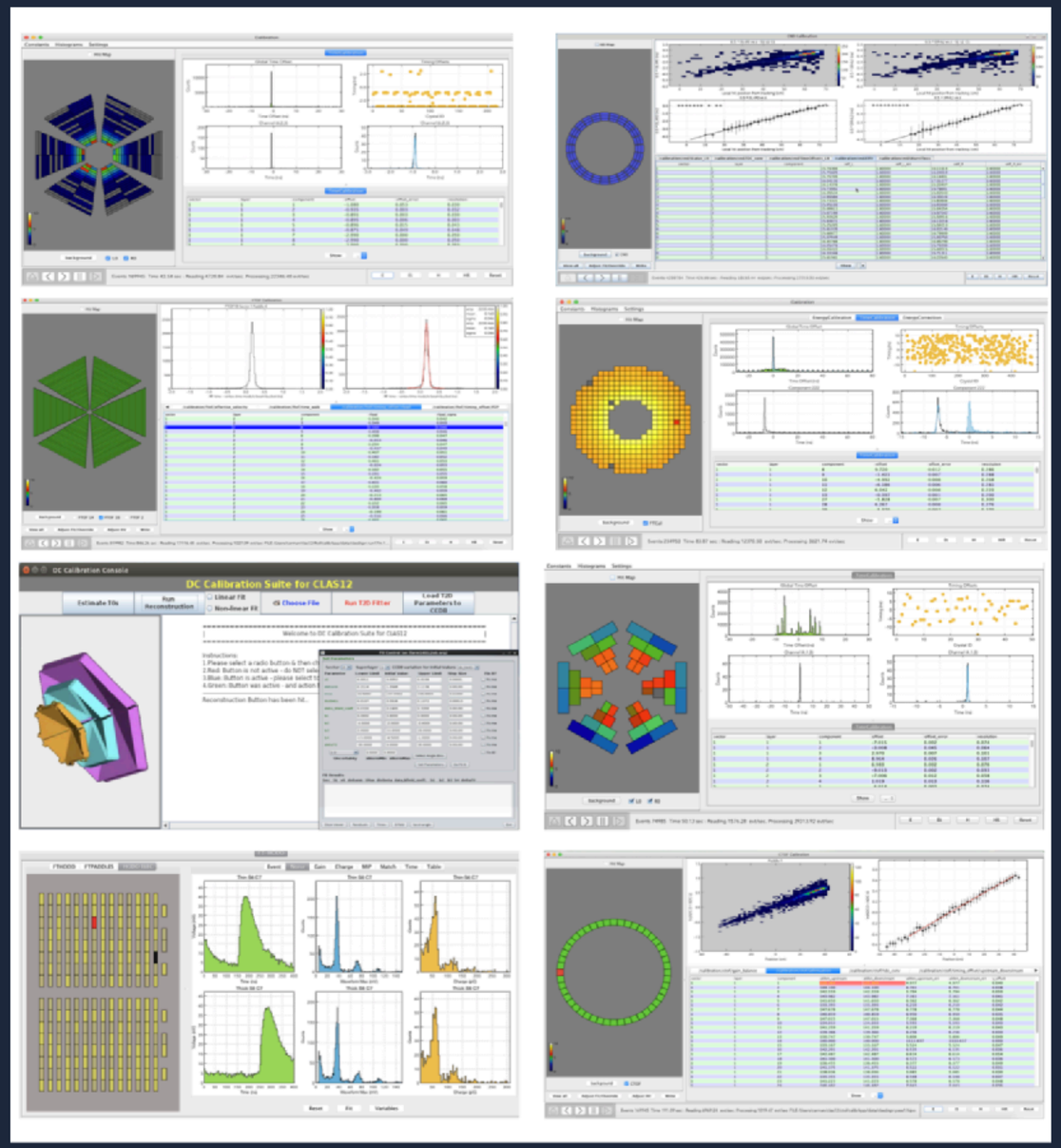
```
import org.jlab.jnp.hipo.io.*;
import org.jlab.jnp.hipo.data.*;
import org.jlab.jnp.reader.*;
import org.jlab.jnp.physics.*;
HipoReader reader = new HipoReader();
reader.open(inputFile);

EventFilter filterT = new EventFilter("11:211:-211:X+:X-:Xn");

while(reader.hasNext()){

  HipoEvent event = reader.readNextEvent();
  PhysicsEvent physEvent = DataManager.getPhysicsEvent(beamEnergy,event);
  Particle mxpipi = physEvent.getParticle("[b]+[t]-[11]-[211]-[-211]");
  Particle impipi = physEvent.getParticle("[211]+[-211]");
  h100.fill(mxpipi.mass());

  if(mxpipi.mass()>0.85&&mxpipi.mass()<1.05){
    h101.fill(mxpipi.mass());
    h102.fill(impipi.mass());
  }
}
```



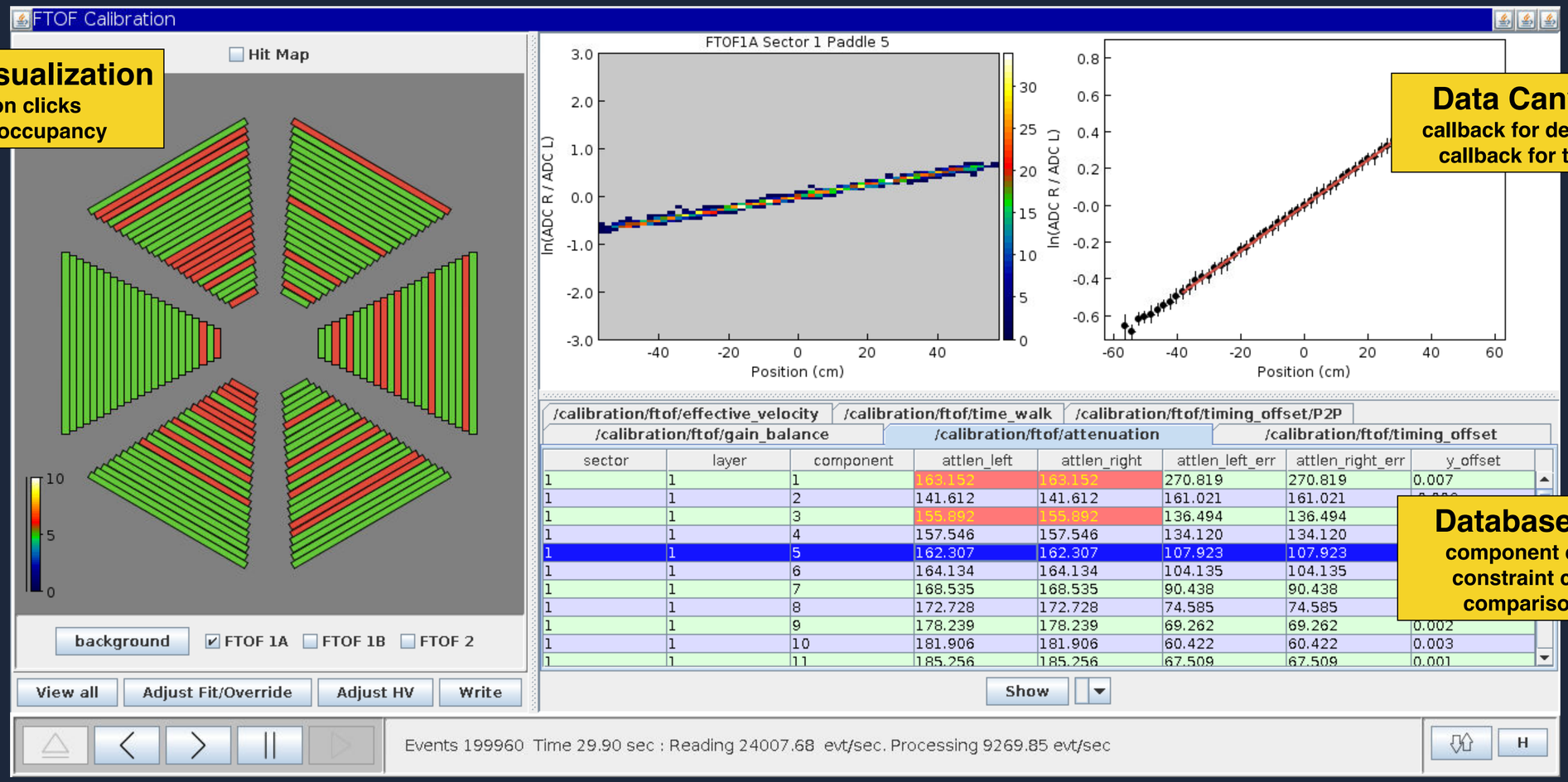
Calibration Software

UNIFIED APPROACH

- Calibration Software is designed within a consistent framework.
- Entire suite is composed of preprogrammed components:
  - Detector View
  - Calibration Constants table
  - Plotting Canvas
  - File processing panel
- User programs only callbacks to the program:
  - What to do for each event (filling histograms)
  - What do do when detector component is clicked
  - What to plot when a row in a table is clicked
- There are predefined numbering scheme to detector components and calibration table structures.



Service Oriented Architecture (SOA)



Detector Visualization
callback on clicks
coloring by occupancy

Data Canvas
callback for detector
callback for table

Database Table
component callback
constraint coloring
comparison utils

Event Processing Pane
event by event,or whole
opens: File, ET ring, CLOUD

Conclusion

PARRALEL DATA PROCESSING

- CLARA modernized the reconstruction software, provided tools for parallelization.
- The reconstruction code is now flexible, maintainable and easily debuggable. Users can develop their own versions of reconstruction code and run along with entire software.
- User analysis codes such as filtering events by event topology and kinematic fitting are also done using CLARA service oriented approach.

DATA FORMATS

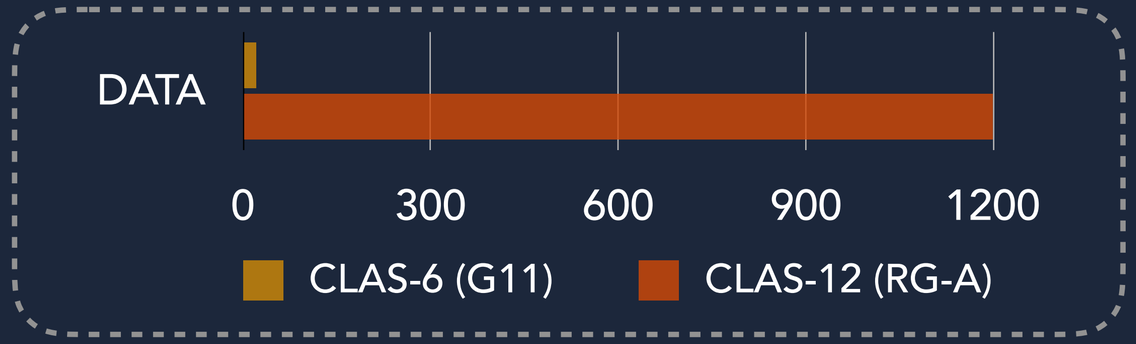
- HIPO data format improved the data storage for CLAS12, providing flexibility to have large files.
- Indexing and tagging mechanism in HIPO allows users to read only events of interest, without putting strain on file system, and made skimming into separate files thing of the past.

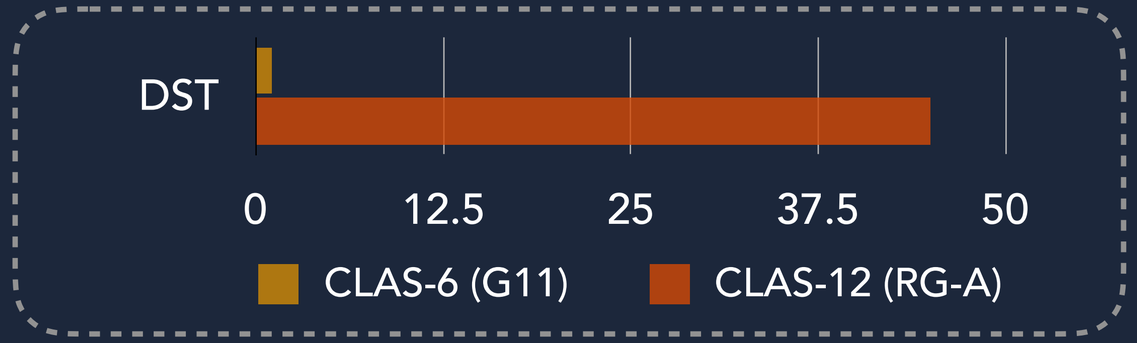
ANSWERS:

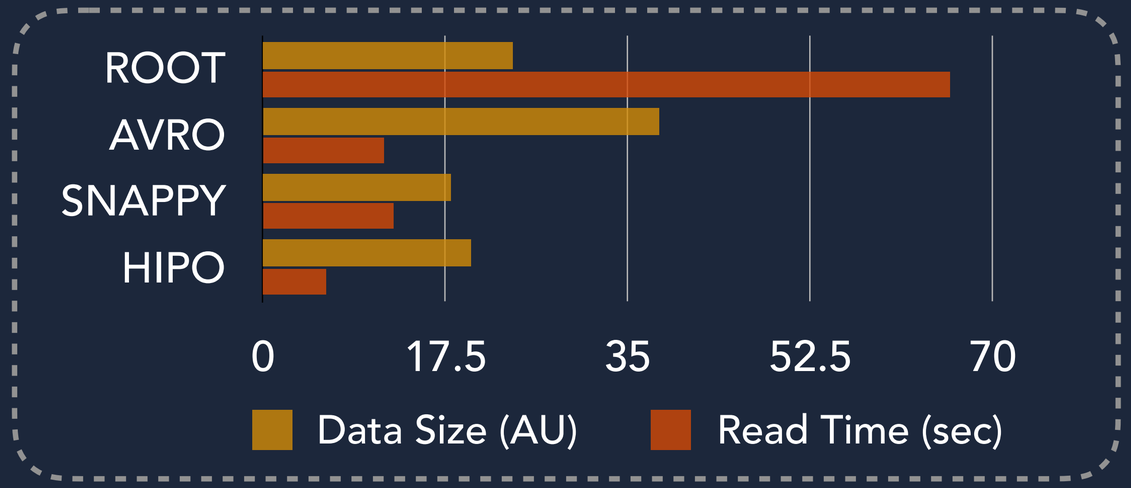
- Computing development took a different route towards parallelization.
- We need more modern approaches to software architecture.
- With increasing data sizes of nuclear physics experiments, we have to re-think data storage and formats.

BACKUP SLIDES

CLAS 12 Detector







DATA SIZES:

- CLAS12 First Experiment collected ~50x more data than longest experiment in CLAS6 era (**~1.12 PB**)
- Data Summary tapes are **~100x** larger (45 TB) than G11 experiment (on the graph compressed size is shown)

HIGH PERFORMANCE OUTPUT (HIPO):

- HIPO data format was implemented for CLAS12
- Dictionary driven data format with Schema evolution.
- Fast compression algorithm (LZ4)
- Chunked data frame implementation to support multithreaded applications.
- Event chunk tagging and indexing to allow reading selective types of events from Data Summary Tapes.
- Deserialization performance better than most commonly used data formats in Nuclear Physics.
- Faster than industry standard data types.