# Clas12 Reconstruction and Analysis Framework

## SOA based physics data processing (PDP)

V. Gyurjyan†, S. Paul‡, S. Heddle‡

†JLAB data acquisition and analysis group, ‡Christopher Newport University

6/13/2010

# PDP environment

- Large user base
  - Deployment
  - Scalability
  - Maintenance
  - Propagation of updates
  - Short response time to bugs
- Long life time
  - Dynamic user base
  - Aging technologies
  - Author-drop rate
- Evolving
  - Drop inefficient/unsatisfying software module(s)
  - Integrate new module(s)
  - Diversification

# PDP Application basic Requirements

- Agility
- Scalability
- Maintainability
- Easy deployment


- Modularity
- Loose coupling
- Distribution


Divide and conquer

# SOA Defined

- An architecture (NOT a technology) based on well defined, reusable components: services.
- Services are loosely coupled.
- Services with functional context that are agnostic to a composed application logic.
- Agnostic services participate in multiple algorithmic compositions.
- Application design based on available services.

# Service

- Atomic unit of an SOA.
- Encapsulates a logic, or a process.
- Autonomous
- Location Transparent.
- It is defined by the messages it can accept and the responses it can give.
- Implements standard contract/interface.
- Composable
  - They can be integrated to provide higher-level, complex services.
- Reusable
- Stateless
- Discoverable
- Loose coupling between services.

# ClaRA

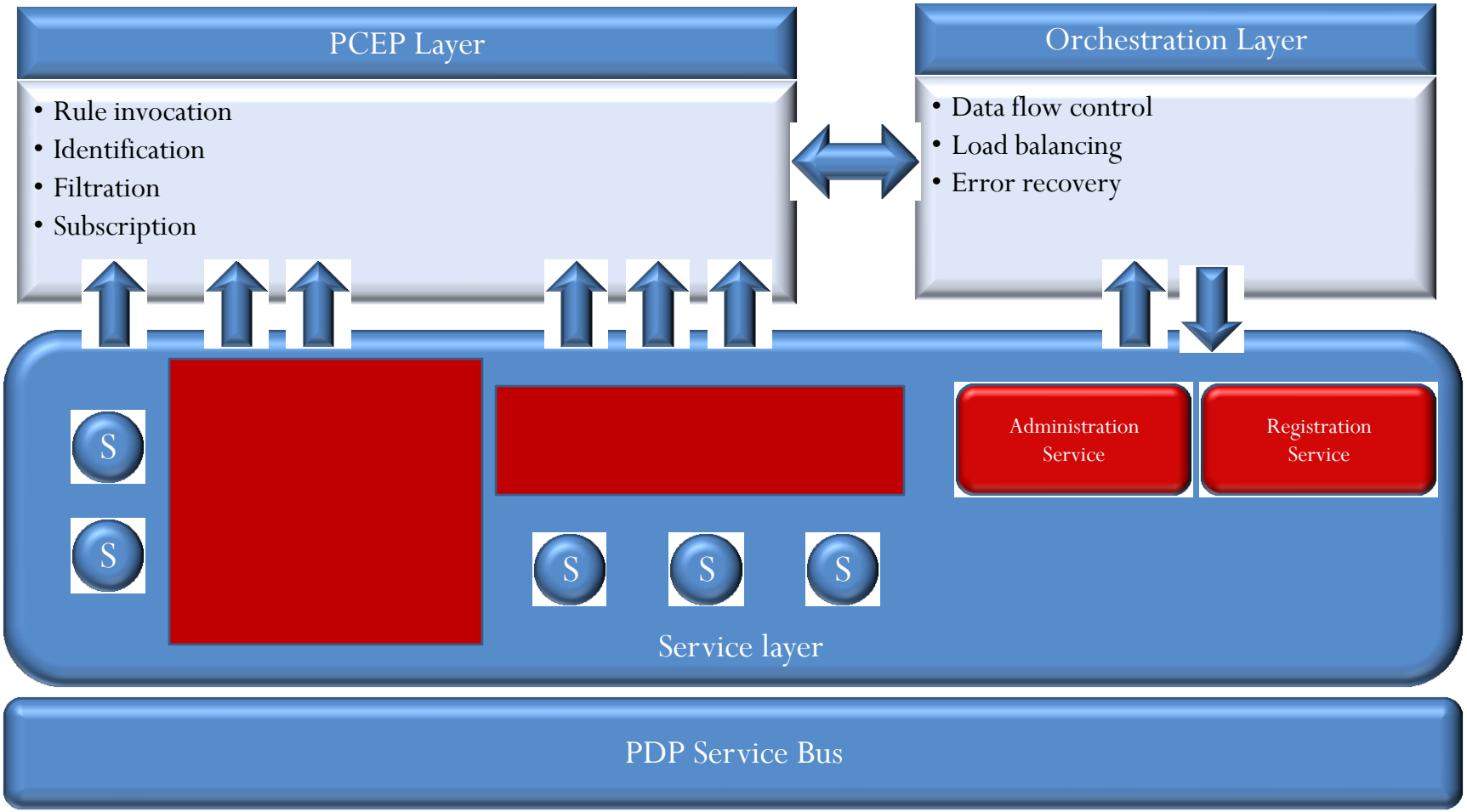
- SOA based physics data production application development framework, written in pure Java.
- Service development environment.
- Increase intrinsic interoperability of services by standardizing data exchange interface.
- Complex service composition.
- Clear separation between PDP application designer and service programmer.
  - Build and run PDP applications without an access to the source code of individual services.
- Increase federation.
  - Services and ClaRA based applications are united while maintaining their individual autonomy and self governance.
- Multi-Threaded event processing.
- Distributed event processing.
- Ease of application deployment.
- PDP application diversification and agility.

# PDP Application Design Data Centric Approach

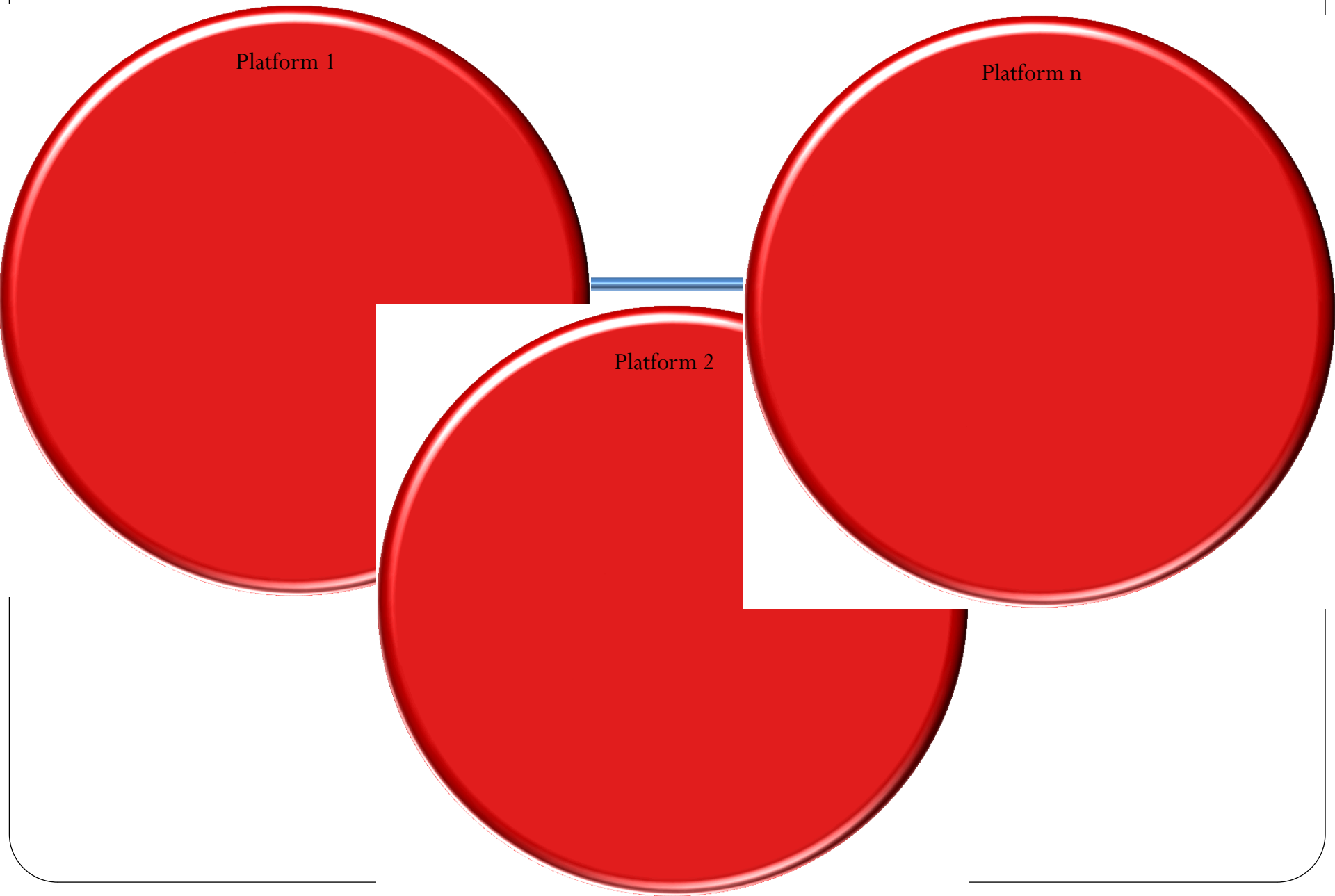
- Focus on data that is moving and transforming in the system.
- Data flow defines the essential aspect of an application.


| Object Centric | Data Centric |
|---|---|
| Data encapsulation | Data exposure |
| Object/method exposure | Object/method encapsulation |
| Intermix of data and algorithm | Separate data and algorithm |
| Tightly coupled | Loosely coupled |

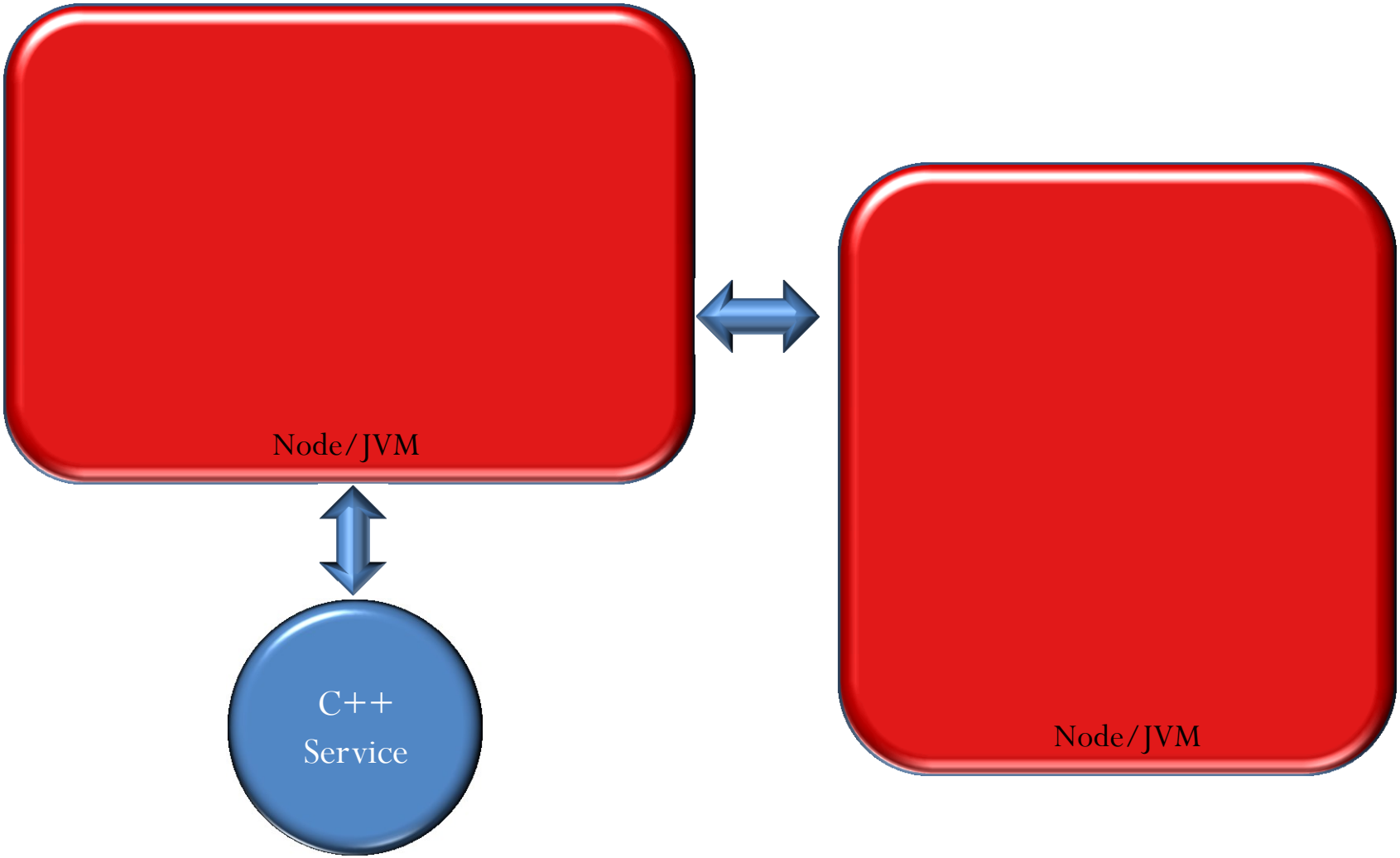
# ClaRA Design Architecture



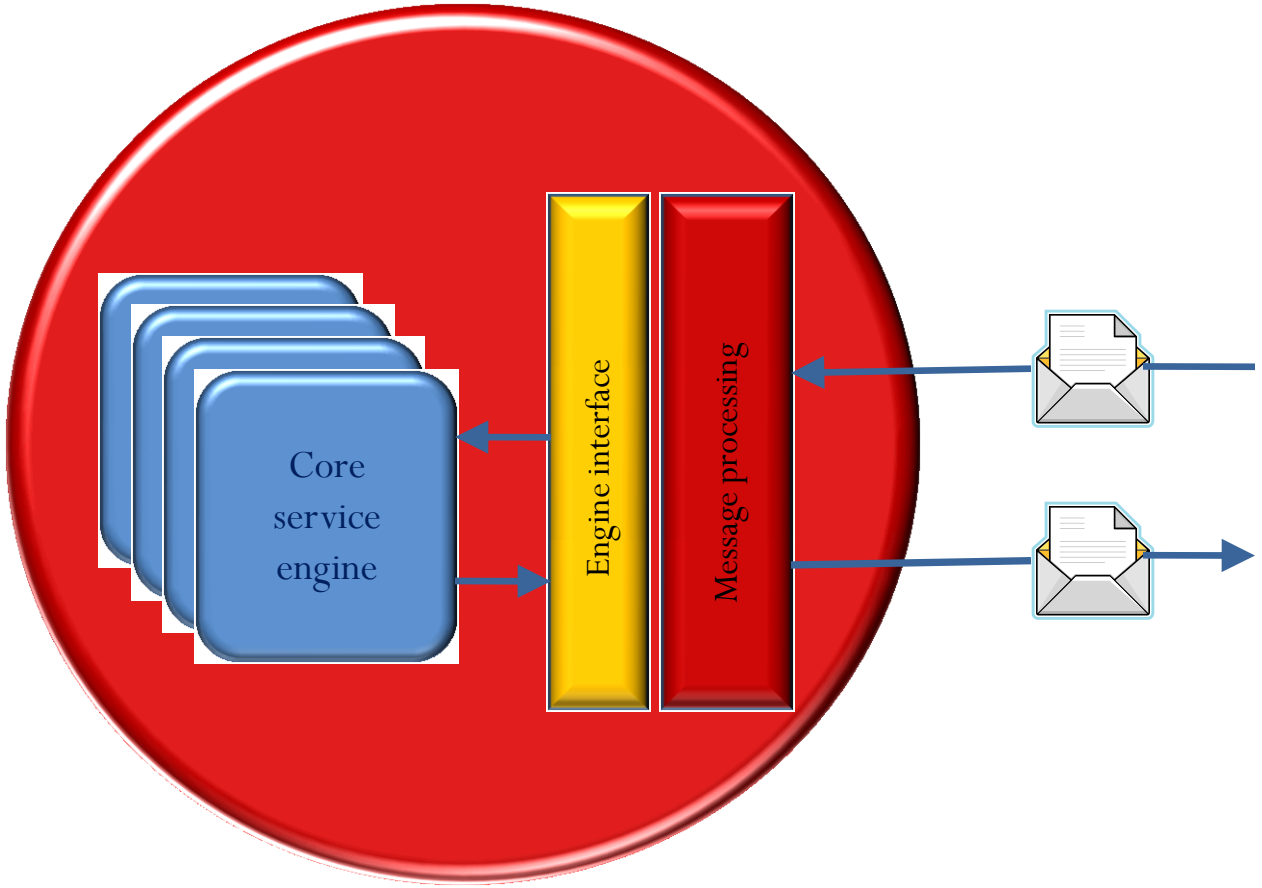
# ClaRA Cloud



# ClaRA Containers



# ClaRA Java Service Container



# Service Coupling

- A functional service context is independent from the outside logic.
- Contract-To-Functional coupling between services.
  - Service has a name.
  - Service receives data and sends data.
- Consumer-To-Contract coupling between consumers and services.
  - User/consumer sends data to the named service.
  - Sending the data will trigger execution of the receivers service.

# Service Interface

- Simple, decoupled from technology and implementation details.
- Input/output data types
  - Object
  - EvIo
  - Primitive types
  - Arrays of primitive types
- Limited  semantic/metadata information
  - Description
  - Author
  - Version

# Service abstraction

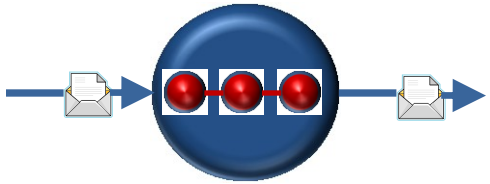
- Technology information (hidden)
- Programmatic logic information (hidden)
- Functional information (exposed through service contract meta data)
- Quality of service information (available from the platform registration services)

# Service types

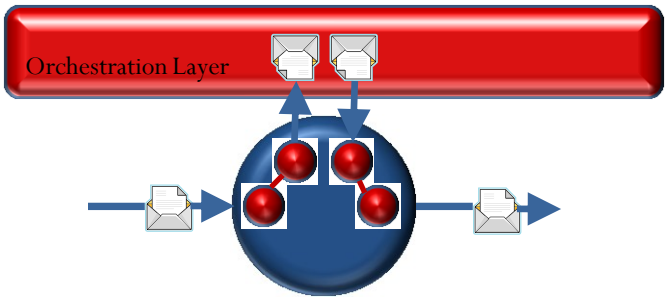
- Entity services
  - Generic
  - Highly reusable
- Utility services
  - Self contained
  - Legacy systems
- Composite services
  - Primitive compositions
    - Self governing service compositions
  - Complex compositions
    - Controlled aggregate services

# Service Composition

- Primitive composition
  - Represents message exchanges across two or more services.
  - Requires composition initiator.
  - Task services are examples of primitive composition.

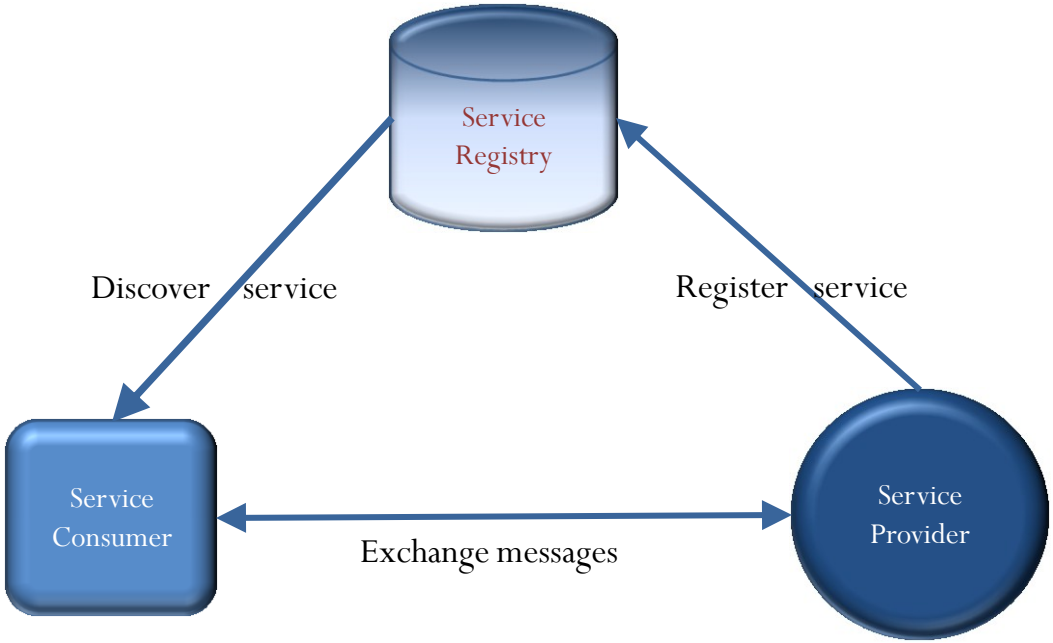


- Complex composition
  - Orchestrated task services seen as a single service.



# Service Discovery

- Design time
- Runtime



# Multi-Threading

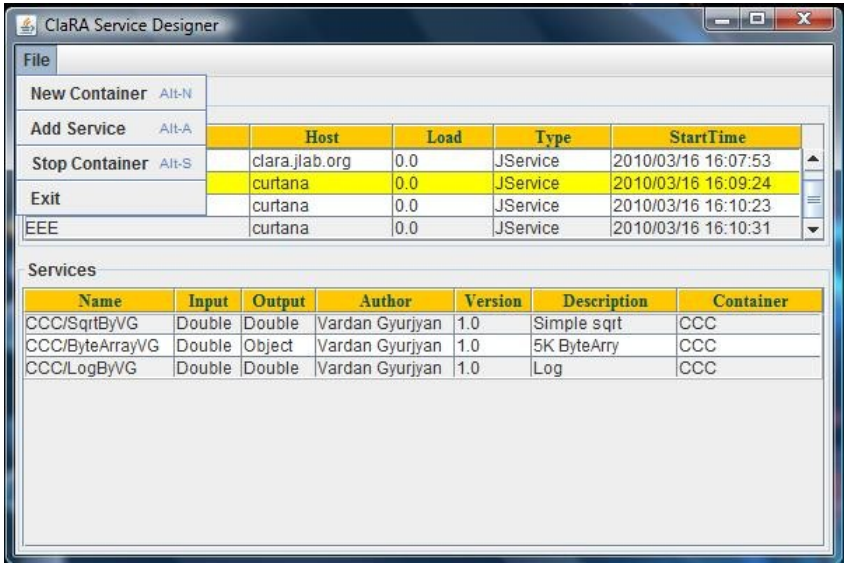
- Only one process is active on the ClaRA platform node.
- Single ClaRA container (JVM) on a node.
- Service containers run in their own threads.
- A single service container executes  contained service engines in separate threads.
- A service engine must be thread safe if it is planned to run in a multi-threaded mode.
- ClaRA based PDP application gets inherent multi-threading and distributed processing, with the relative processing time :

$$\Delta T_R = (\Delta T_A + \Delta t_n) / N_c * N_n$$

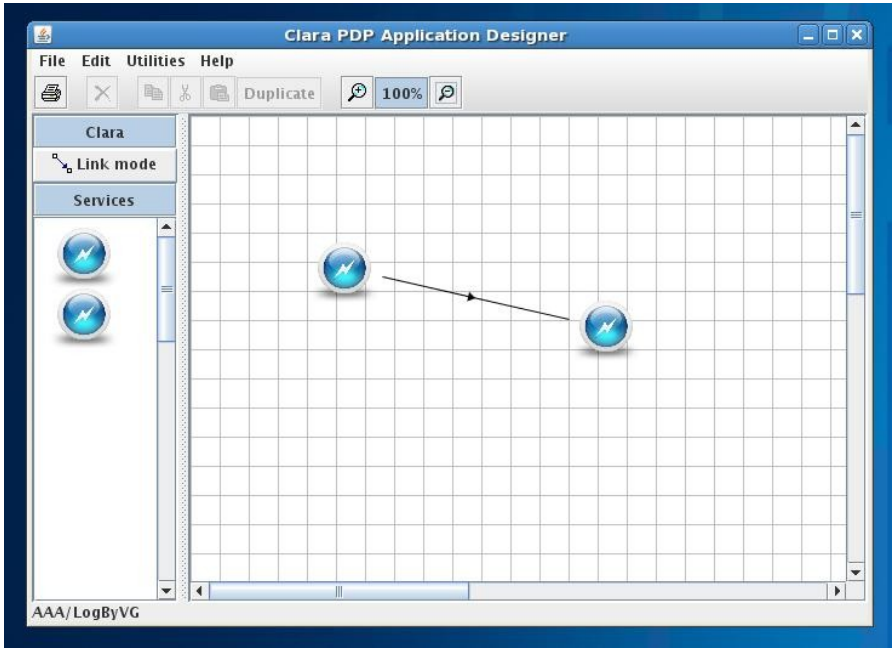
# Advantages Multi-Threading over Multi-Processing

- Small memory footprint, less L1/L2 caching, lower probability for missing a cache, less RAM access, better performance.
- Multi-Processing bookkeeping complexity.
  - Users must keep track of input/output data distribution and data reconsolidation.

# Service deployment and monitoring Interfaces

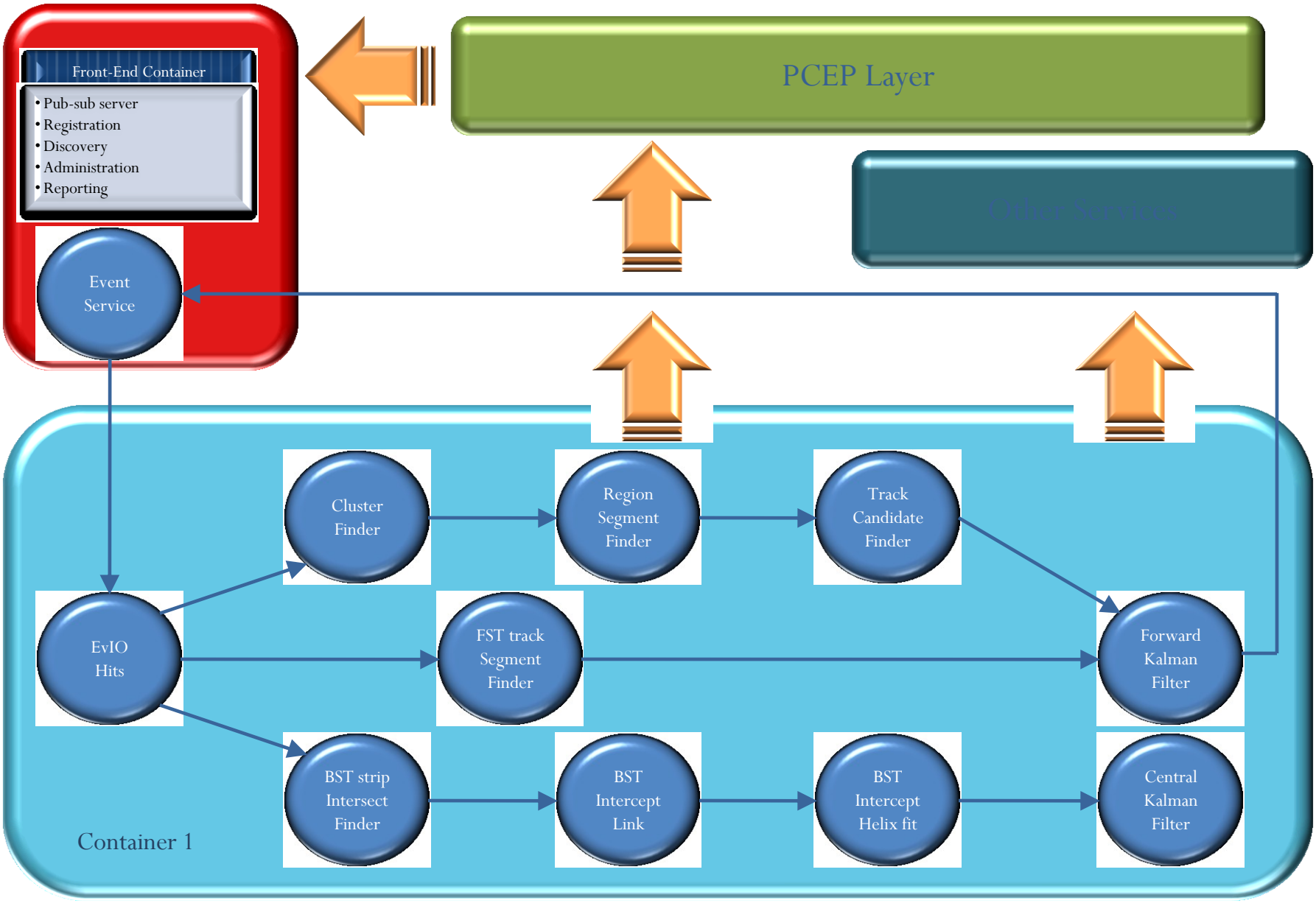


# PDP Application Designer Interfaces

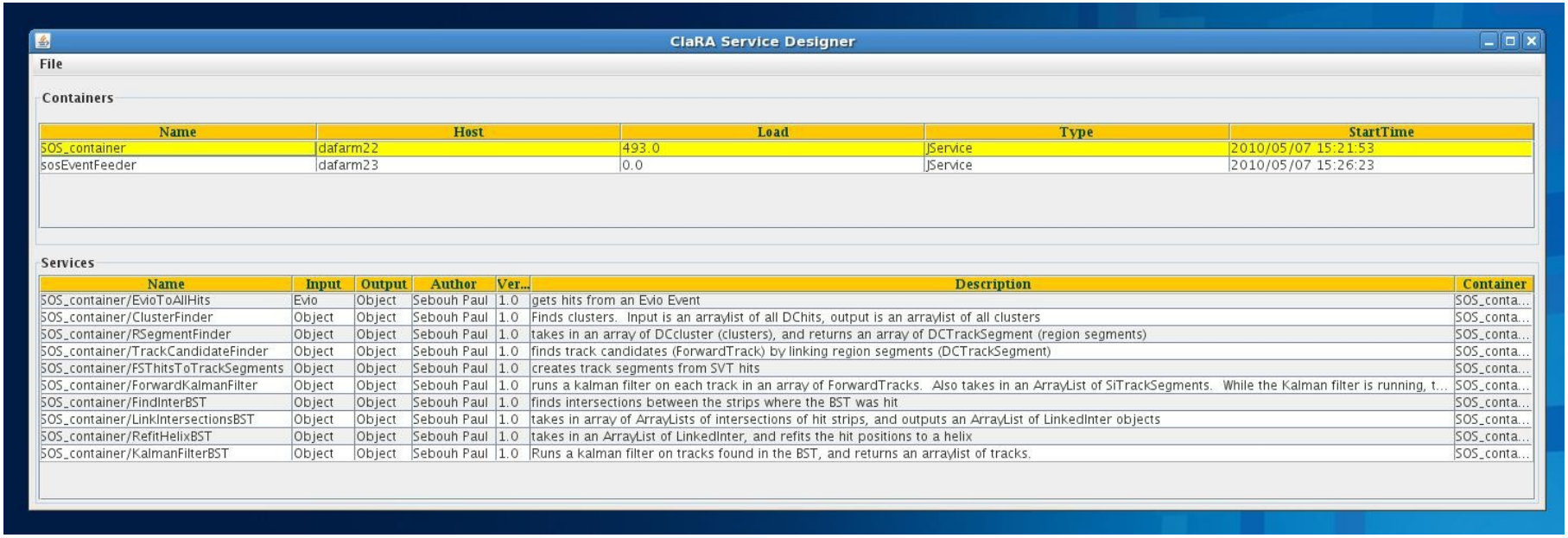


# Clas12 Reconstruction

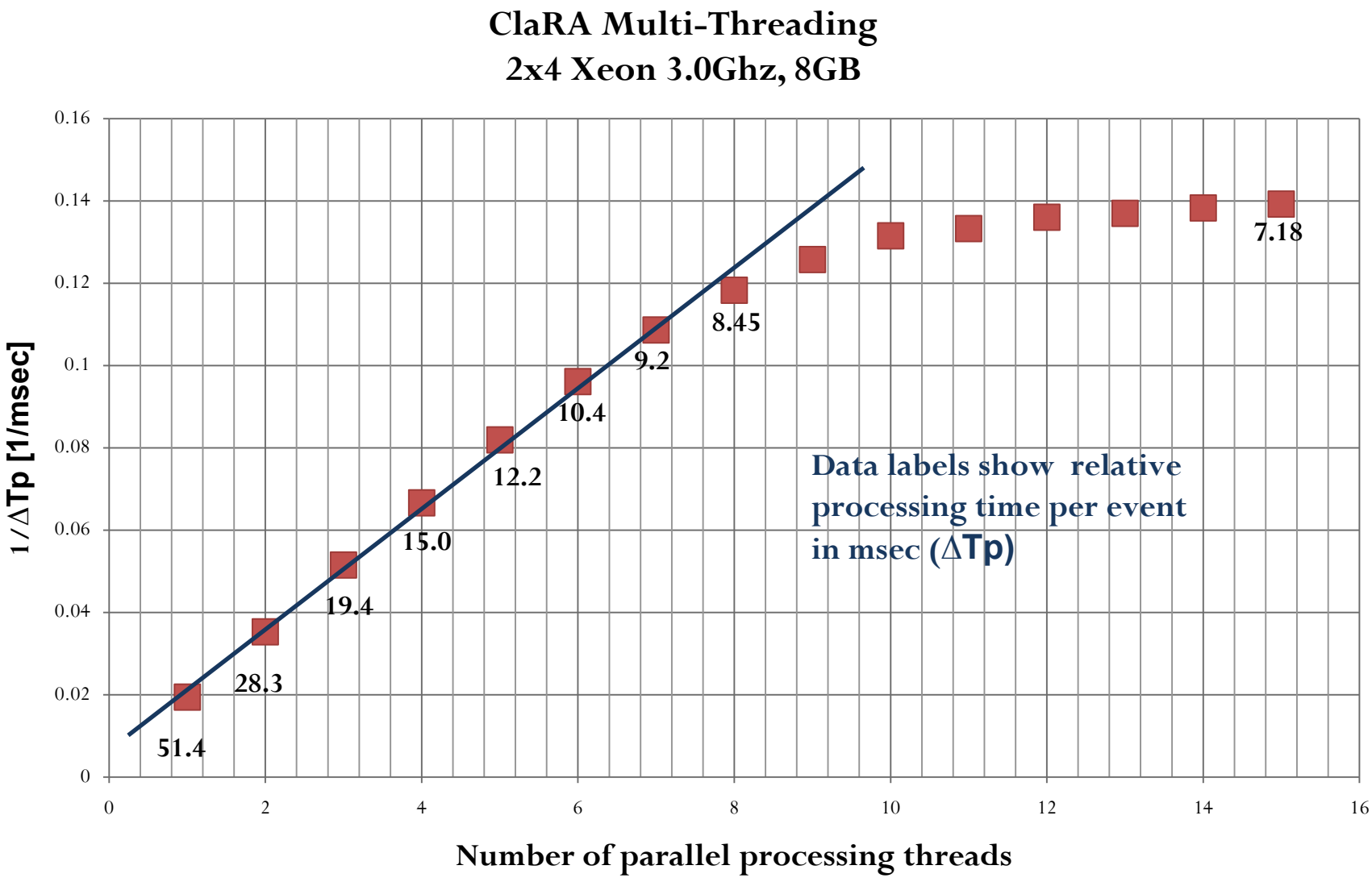
# Clas12 Track Reconstruction Services, and Tracking Application Design



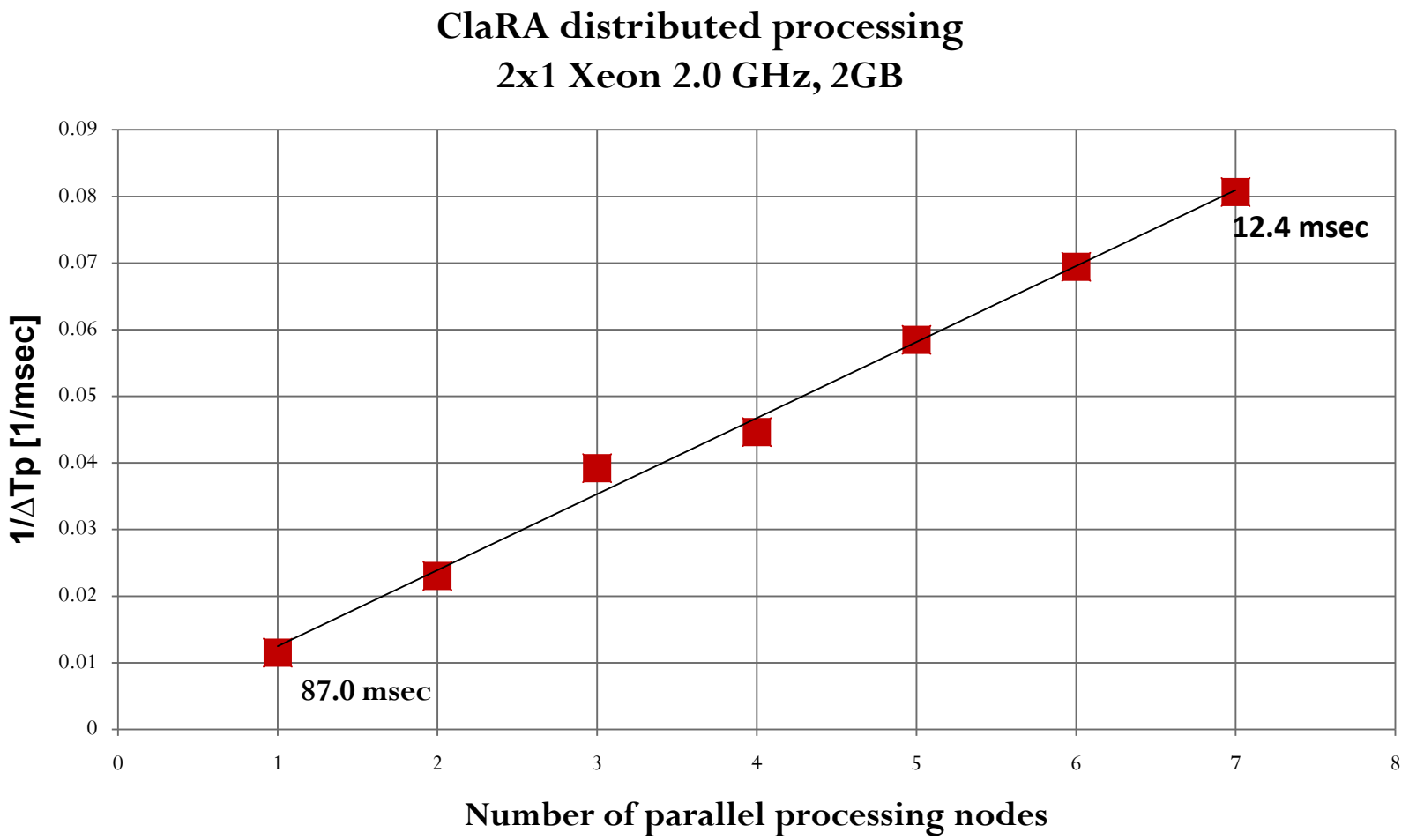
# Service deployment monitoring



# Composite Reconstruction Service Multi-Threading



# Composite Reconstruction Service Distributed Processing



# Conclusion

- SOA based physics data production application development framework, written in pure Java.
- Separation between PDP application designer and service programmer.
- Service development environment.
- Increase intrinsic interoperability of services by standardizing data exchange interface.
- Increase federation.
- Multi-Threaded event processing.
- Distributed event processing.
- Ease of application deployment.
- Increase application diversification and agility.
- Designed and deployed service based Clas12 full track reconstruction application, showing ~750micor second per event relative processing time on the ClaRA platform using 10 JLAB farm nodes.

# Tutorial
# Building a service

# Java

```
public class SqrtEngine implements ICService {
    public Object executeService(int type,Object input) {
        if(type==DOUBLE){
            return Math.sqrt((Double)input);
        } else {
            return null;
        }
    }
    public String getName() {
        return "SqrtByVG";
    }
    public String getDescription() {
        return  "Simple sqrt";
    }
    public String getAuthor() {
        return "Vardan Gyurjyan";
    }
    public int getInputType() {
        return DOUBLE;
    }
    public int getOutputType() {
        return DOUBLE;
    }
    public String getversion() {
        return "1.0";
    }
}
```

# C++

```
#include <string>
#include "CService.h"
using namespace std;

static string name = "";
static string phost = "";
static string pname = "";

class Average : public CService {
public:
    // constructor
    Average(string name, string platformHost, string platformName) :
    CService(name, platformHost, platformName) {
    };

    // service engine
    Cio* executeService(int type, Cio* o) {
        if (type == DOUBLE_ARRAY) {
            Cio* output = new Cio();
            double avg;
            double** data = static_cast<double**> (o->getData());
            for (int i = 0; i < o->getLength(); i++) {
                avg = *data[i] + avg;
            }
            avg = avg / o->getLength();
            output->setData(&avg);
            delete(data);
            return output;
        }

int main(int argc, char** argv) {
    string description = "Simple average calculation";
    string author = "gurjyan";
    string version = "1.0";
    int inputType = DOUBLE_ARRAY;
    int outputType = DOUBLE;

    // create an instance
    Average* a = new Average(name, phost, pname);

    // register service
    a->registerService(name, description, author, version,
inputType, outputType);
    while (1) {
        sleep(1);
    }
    return (EXIT_SUCCESS);
};
```

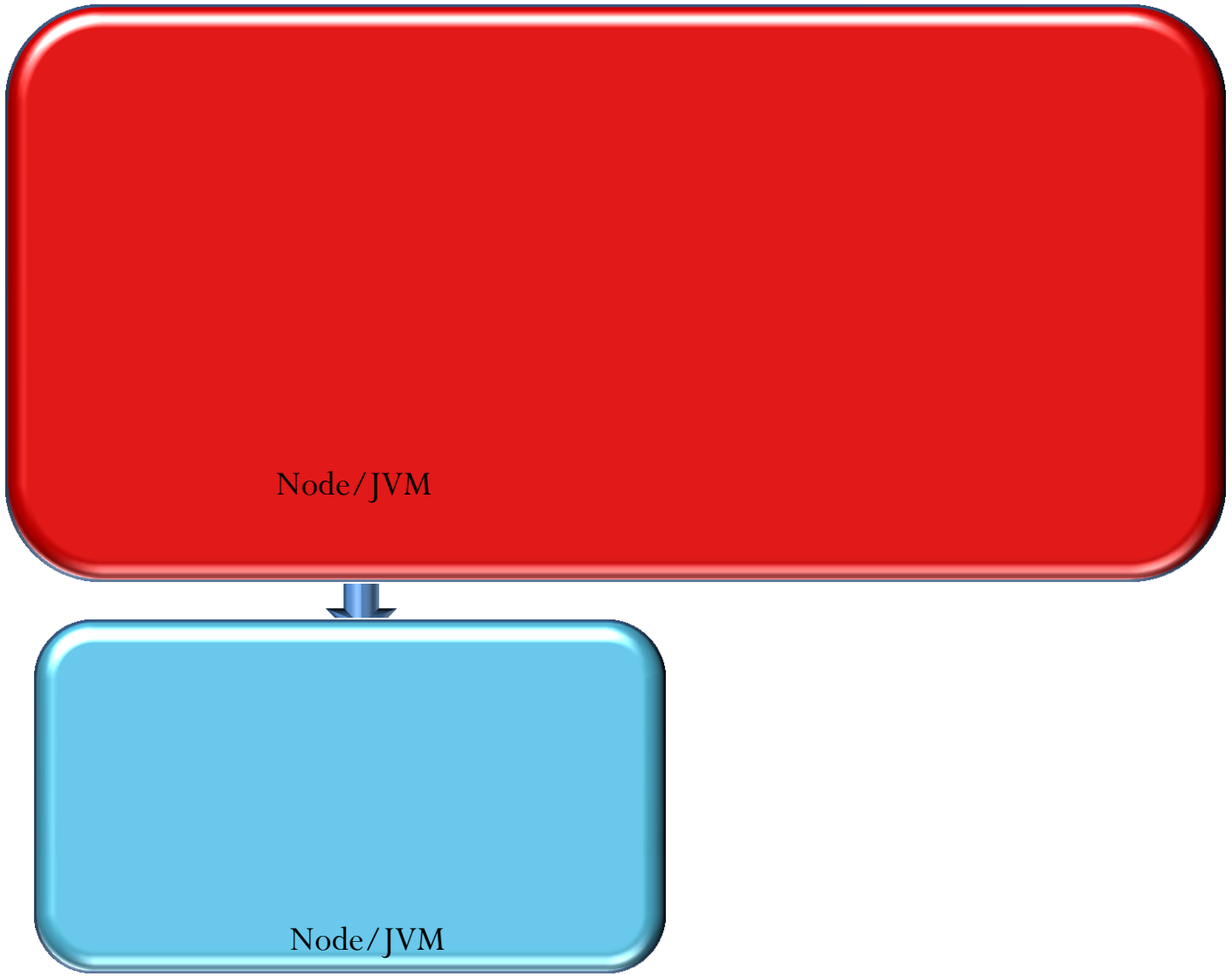
# C++

```
#ifndef _CIO_H
#define                 _CIO_H

class Cio {
public:
   Cio() {};
   void* getData(){return data;};
   void  setData(void* d){data = d;};
   int   getLength(){return length;};
   void  setLength(int d){length = d;};
   int   getEndean(){return indean;};
   void  setEndean(int d){indean = d;};
   int   getOffset(){return offset;};
   void  setOffset(int d){offset = d;};
private:
   void* data;
   int   length;
   int   indean;
   int   offset;
};

#endif                  /* _CIO_H */
```

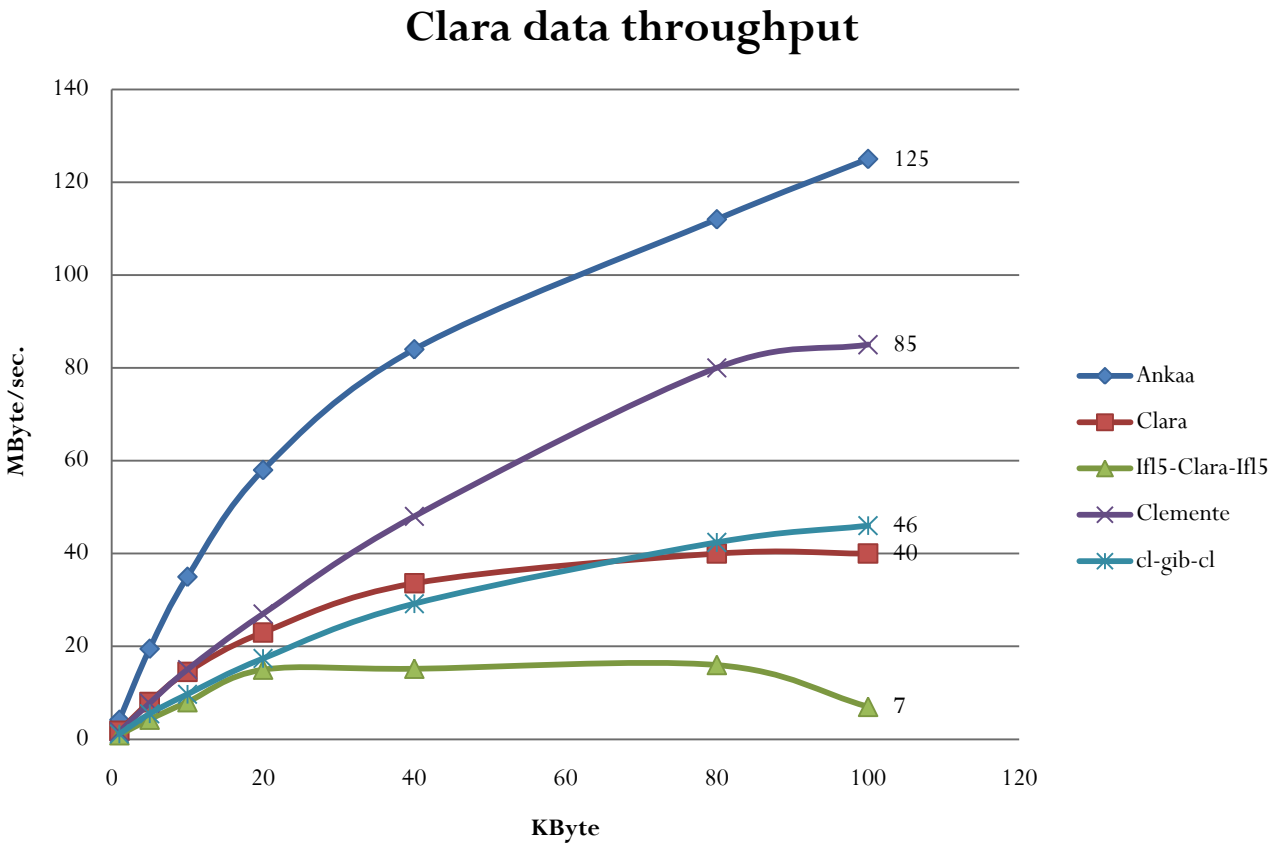
```
#ifndef _CCONST_H
#define                 _CCONST_H
   static const int   EVIO                 = 199;
   static const int   OBJECT               = 200;
   static const int   BYTE                 = 201;
   static const int   SHORT                = 202;
   static const int   INT                  = 203;
   static const int   FLOAT                = 204;
   static const int   DOUBLE               = 205;
   static const int   STRING               = 206;
   static const int   BYTE_ARRAY           = 207;
   static const int   SHORT_ARRAY          = 208;
   static const int   INT_ARRAY            = 209;
   static const int   FLOAT_ARRAY          = 210;
   static const int   DOUBLE_ARRAY         = 211;
   static const int   STRING_ARRAY         = 212;
   static const int   OBJECT_ARRAY         = 213;
#endif                  /* _CCONST_H */
```

# Tutorial
# Simple deployment

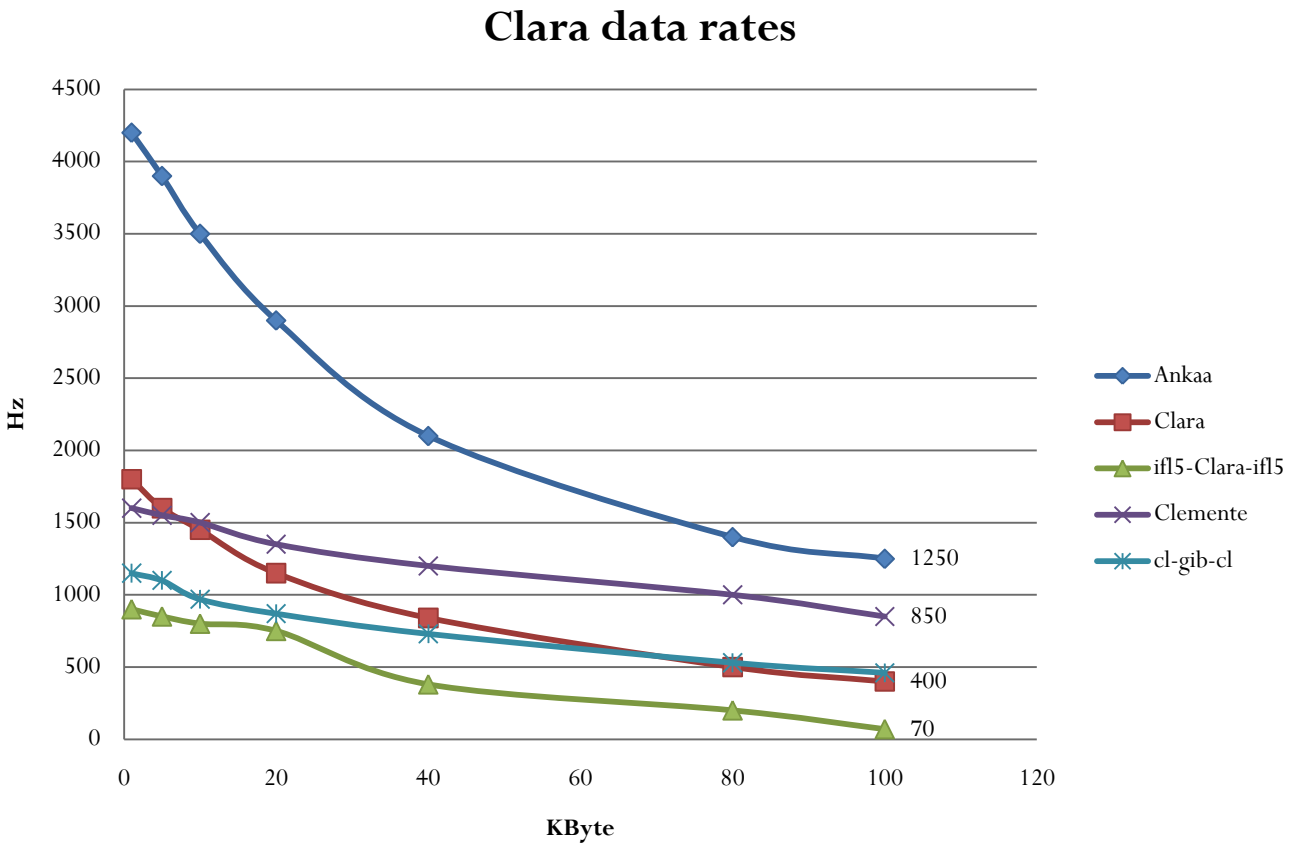
# Exercise



# Data Throughput



# Data Rate


Clara data rates
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Hz
0
20
40
60
80
100
120
KByte
1250
850
400
70
Ankaa
Clara
ifl5-Clara-ifl5
Clemente
cl-gib-cl