

Clas12 Reconstruction and Analysis Framework

SOA based physics data processing (PDP)

V. Gyurjyan[†], S. Paul[‡], S. Heddle[‡]

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

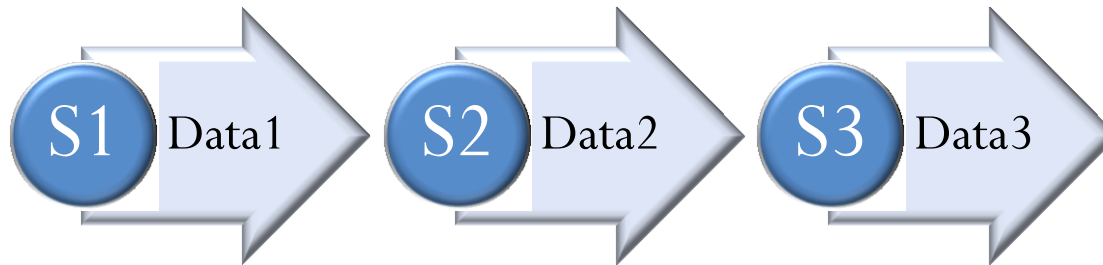
- Atomic unit of an SOA.
- Encapsulates a logic, or a process.
- Autonomous
- Location Transparent.
- It is defined by the messages it can accept and the responses it can give.
- Implements standard contract/interface.
- Composable
 - They can be integrated to provide higher-level, complex services.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

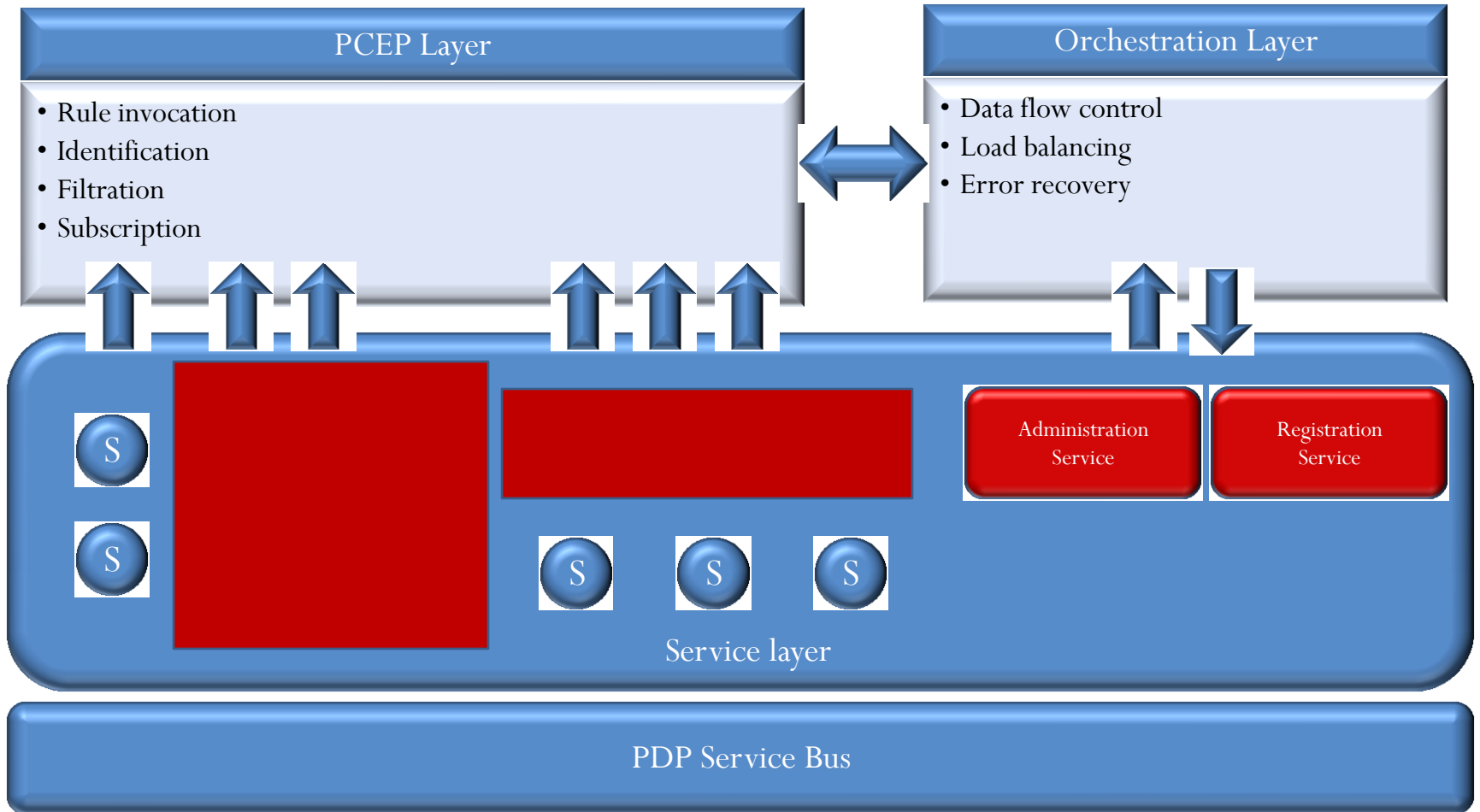
PDP Application Design Data Centric Approach

- 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

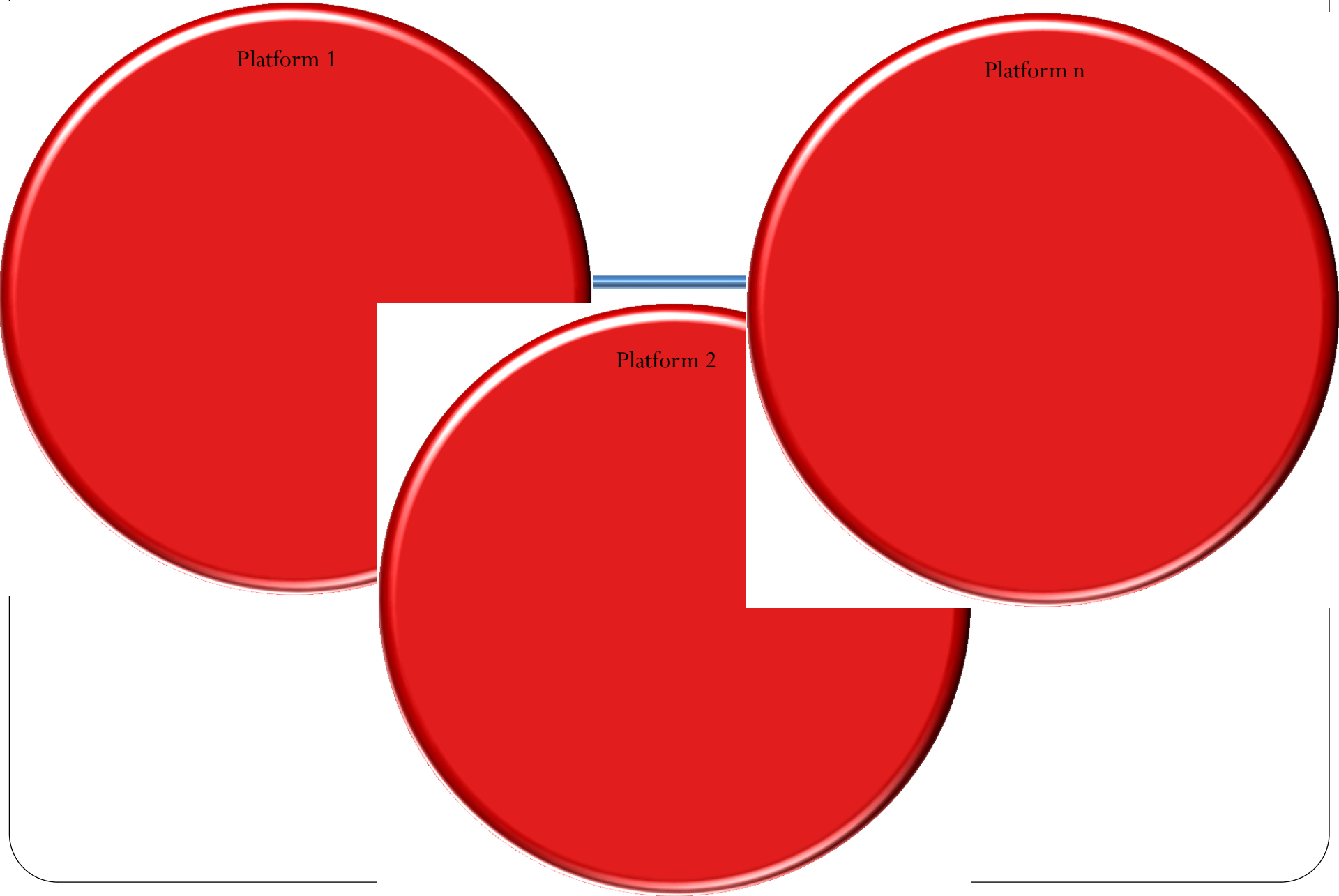


ClaRA Cloud

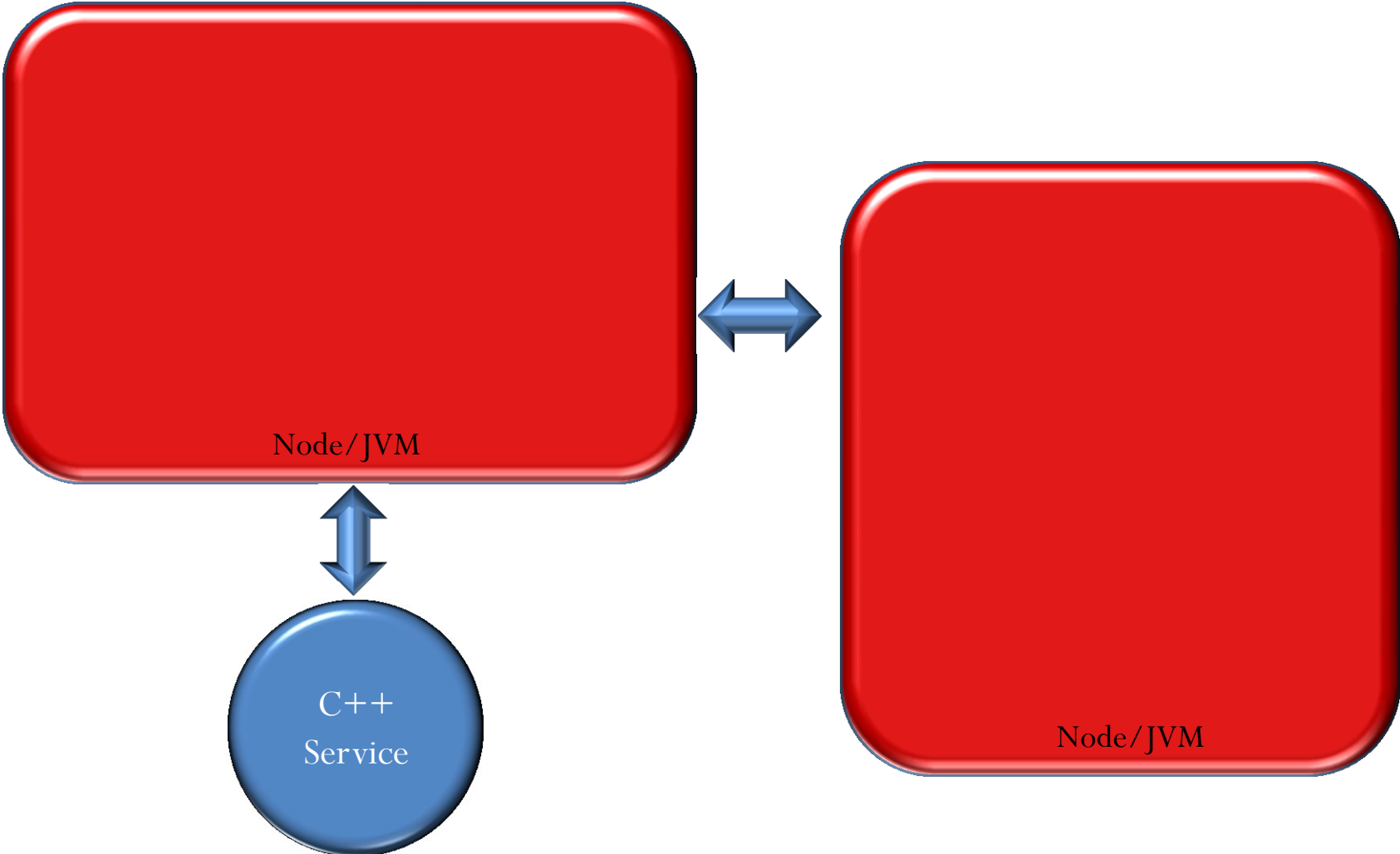
Platform 1

Platform n

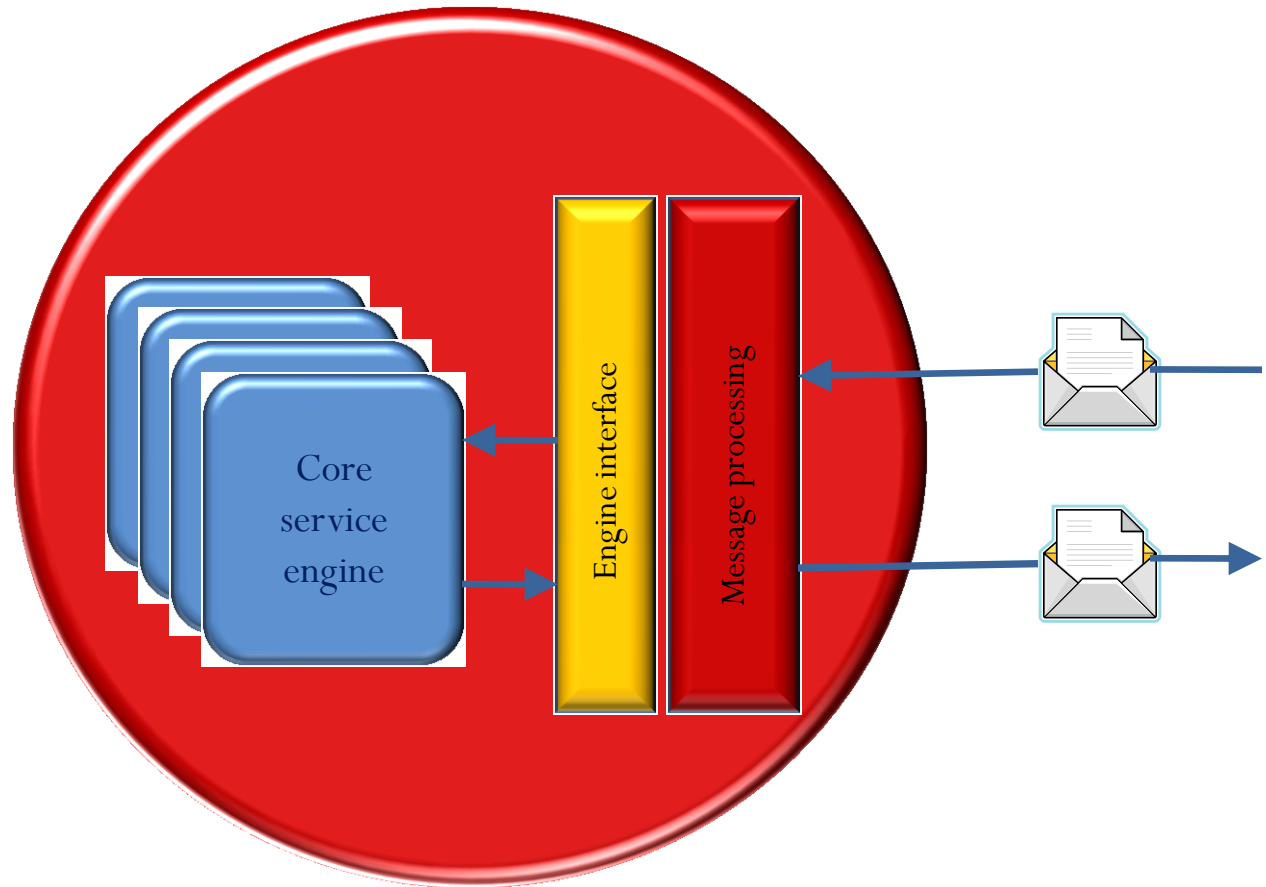
Platform 2



ClaRA Containers

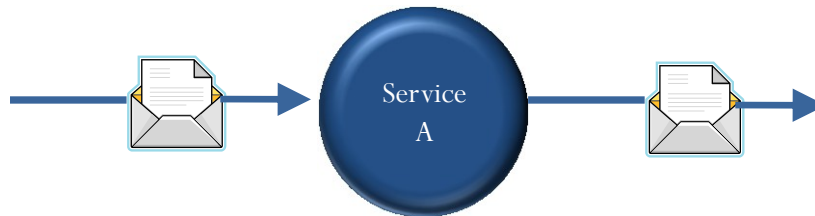


ClaRA Java Service Container



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

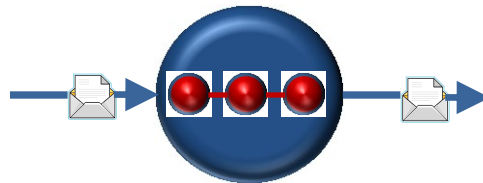
- Technology information (hidden)
- Programmatic logic information (hidden)
- Functional information (exposed through service contract meta data)
- Quality of service information (available from the platform registration services)

Service types

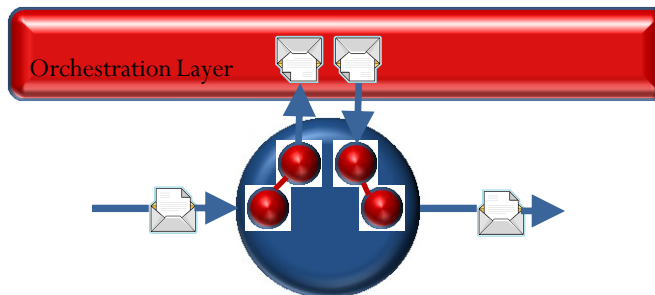
- Entity services
 - Generic
 - Highly reusable
- Utility services
 - Self contained
 - Legacy systems
- Composite services
 - Primitive compositions
 - Self governing service compositions
 - Complex compositions
 - Controlled aggregate services

Service Composition

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

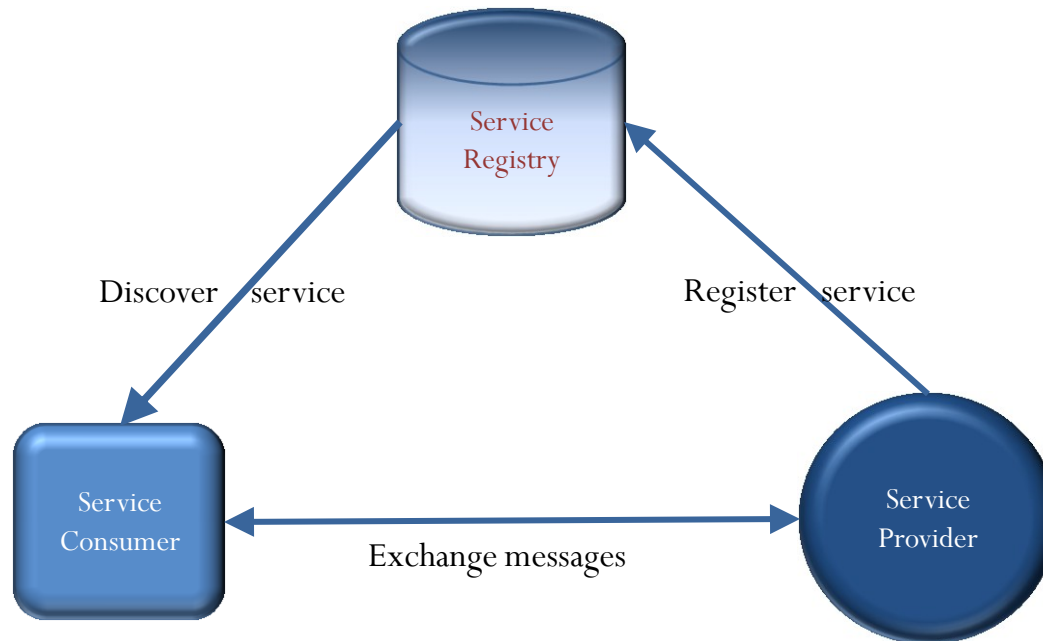


- Complex composition
 - Orchestrated task services seen as a single service.



Service Discovery

- Design time
- Runtime



Multi-Threading

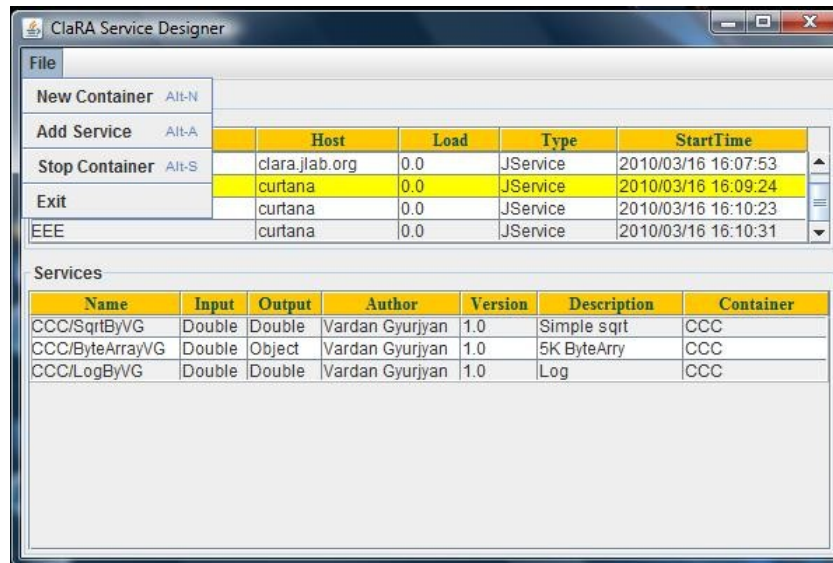
- Only one process is active on the ClaRA platform node.
- Single ClaRA container (JVM) on a node.
- Service containers run in their own threads.
- A single service container executes contained service engines in separate threads.
- A service engine must be thread safe if it is planned to run in a multi-threaded 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



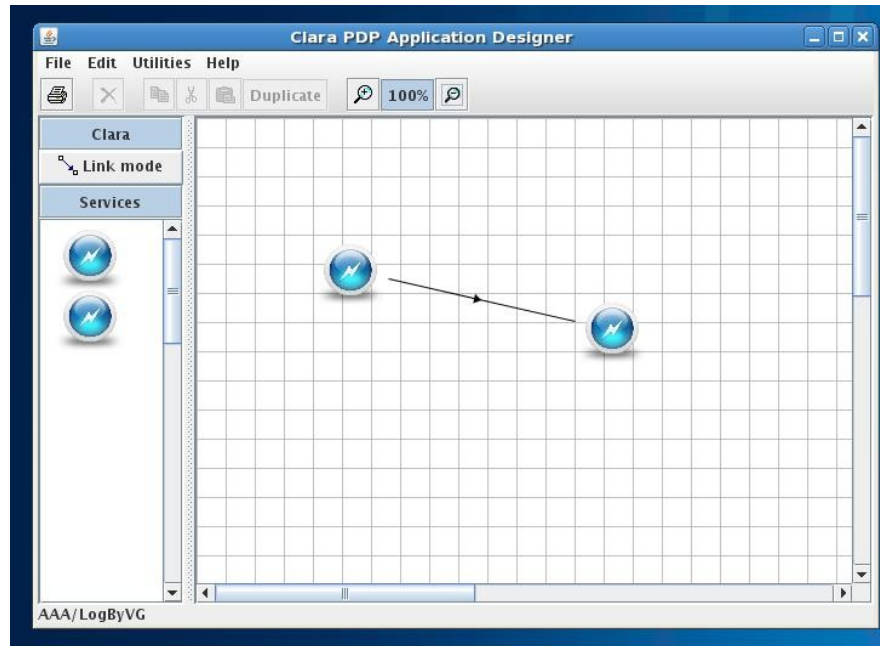
The screenshot displays the 'ClaRA Service Designer' application window. It features a menu bar with the following options: File, New Container (Alt+N), Add Service (Alt+A), Stop Container (Alt+S), Exit, and EEE. Below the menu is a table with columns: Host, Load, Type, and StartTime. The table contains four rows of data, with the second and third rows highlighted in yellow.

| Host | Load | Type | StartTime |
|----------------|------|----------|---------------------|
| clara.jlab.org | 0.0 | JService | 2010/03/16 16:07:53 |
| curtana | 0.0 | JService | 2010/03/16 16:09:24 |
| curtana | 0.0 | JService | 2010/03/16 16:10:23 |
| curtana | 0.0 | JService | 2010/03/16 16:10:31 |

Below this table is a section titled 'Services' containing another table with columns: Name, Input, Output, Author, Version, Description, and Container. This table lists three services.

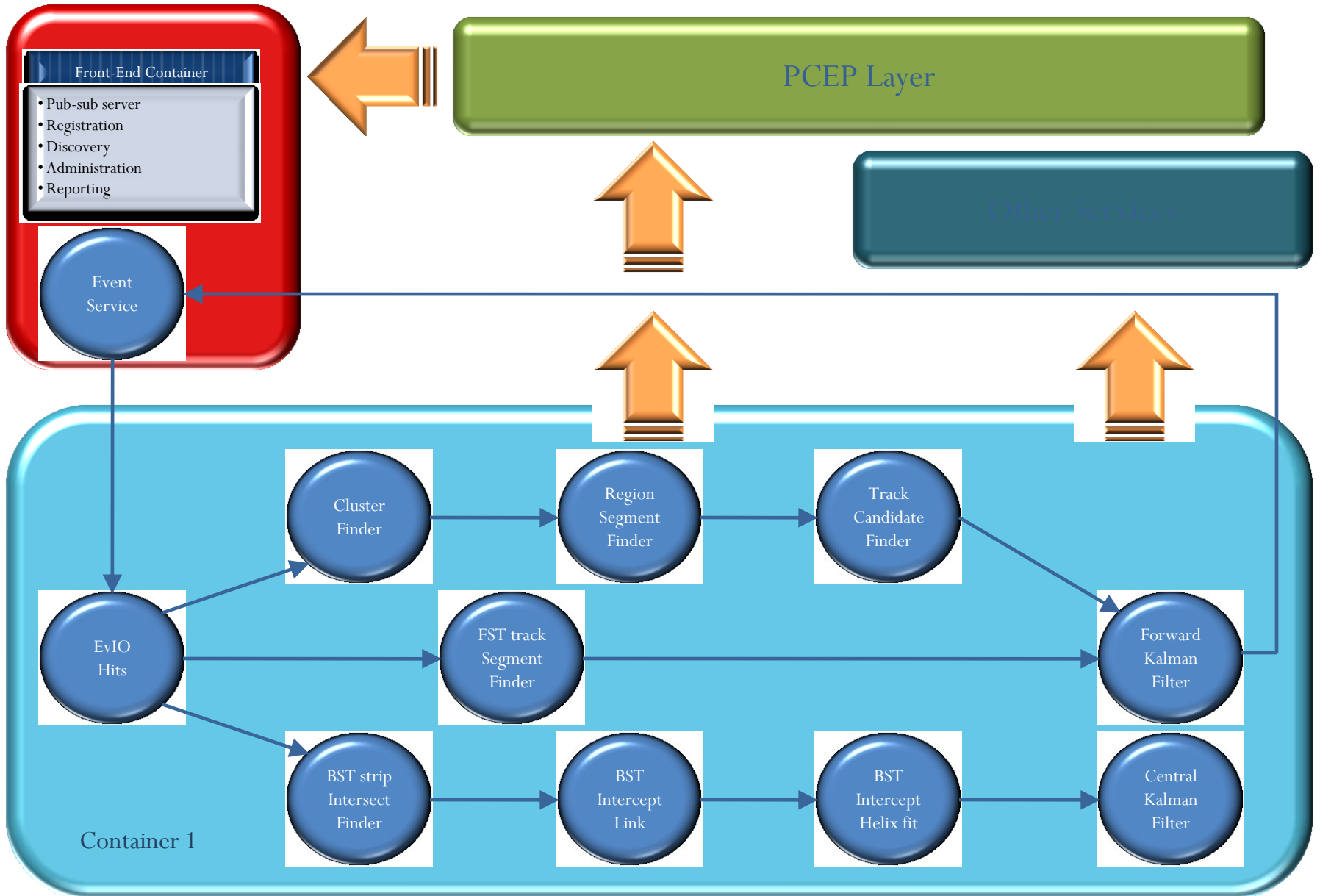
| Name | Input | Output | Author | Version | Description | Container |
|-----------------|--------|--------|-----------------|---------|--------------|-----------|
| CCC/SqrtByVG | Double | Double | Vardan Gyurjyan | 1.0 | Simple sqrt | CCC |
| CCC/ByteArrayVG | Double | Object | Vardan Gyurjyan | 1.0 | 5K ByteArray | CCC |
| CCC/LogByVG | Double | Double | Vardan Gyurjyan | 1.0 | Log | CCC |

PDP Application Designer Interfaces



Clas12 Reconstruction

Clas12 Track Reconstruction Services, and Tracking Application Design



Service deployment monitoring

ClARA Service Designer

File

Containers

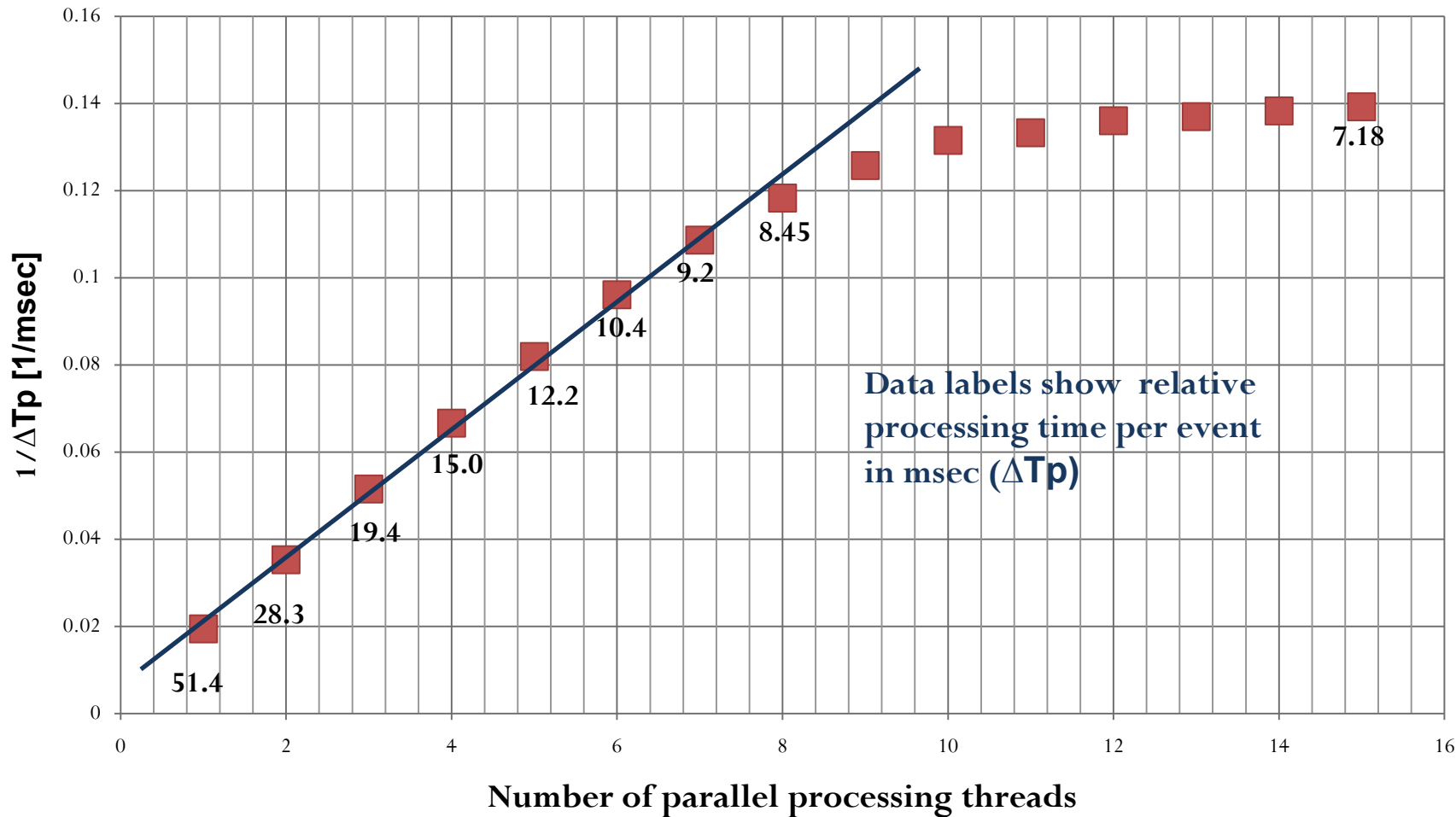
| Name | Host | Load | Type | StartTime |
|----------------|----------|-------|----------|---------------------|
| SOS_container | dafarm22 | 493.0 | JService | 2010/05/07 15:21:53 |
| sosEventFeeder | dafarm23 | 0.0 | JService | 2010/05/07 15:26:23 |

Services

| Name | Input | Output | Author | Ver... | Description | Container |
|--------------------------------------|--------|--------|-------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| SOS_container/EvioToAllHits | Evio | Object | Sebouh Paul | 1.0 | gets hits from an Evio Event | SOS_conta... |
| SOS_container/ClusterFinder | Object | Object | Sebouh Paul | 1.0 | Finds clusters. Input is an arraylist of all DChits, output is an arraylist of all clusters | SOS_conta... |
| SOS_container/RSegmentFinder | Object | Object | Sebouh Paul | 1.0 | takes in an array of DCcluster (clusters), and returns an array of DCTrackSegment (region segments) | SOS_conta... |
| SOS_container/TrackCandidateFinder | Object | Object | Sebouh Paul | 1.0 | finds track candidates (ForwardTrack) by linking region segments (DCTrackSegment) | SOS_conta... |
| SOS_container/FSThitsToTrackSegments | Object | Object | Sebouh Paul | 1.0 | creates track segments from SVT hits | SOS_conta... |
| SOS_container/ForwardKalmanFilter | Object | Object | Sebouh Paul | 1.0 | runs a kalman filter on each track in an array of ForwardTracks. Also takes in an ArrayList of SiTrackSegments. While the Kalman filter is running, t... | SOS_conta... |
| SOS_container/FindInterBST | Object | Object | Sebouh Paul | 1.0 | finds intersections between the strips where the BST was hit | SOS_conta... |
| SOS_container/LinkIntersectionsBST | Object | Object | Sebouh Paul | 1.0 | takes in array of ArrayLists of intersections of hit strips, and outputs an ArrayList of LinkedInter objects | SOS_conta... |
| SOS_container/RefitHelixBST | Object | Object | Sebouh Paul | 1.0 | takes in an ArrayList of LinkedInter, and refits the hit positions to a helix | SOS_conta... |
| SOS_container/KalmanFilterBST | Object | Object | Sebouh Paul | 1.0 | Runs a kalman filter on tracks found in the BST, and returns an arraylist of tracks. | SOS_conta... |

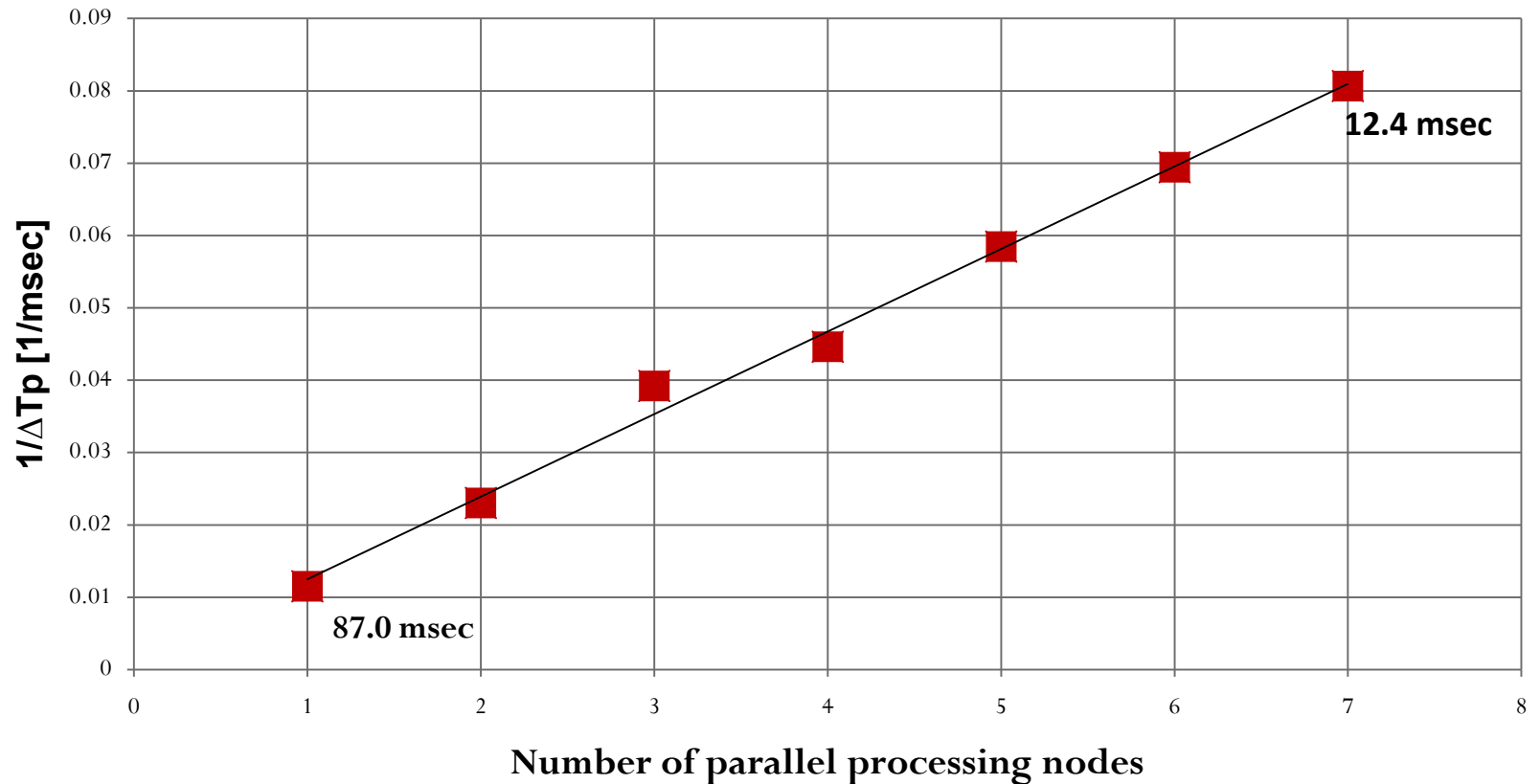
Composite Reconstruction Service Multi-Threading

ClaRA Multi-Threading
2x4 Xeon 3.0Ghz, 8GB



Composite Reconstruction Service Distributed Processing

ClARA distributed processing
2x1 Xeon 2.0 GHz, 2GB



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";
    }
}
```



```
#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 */
```

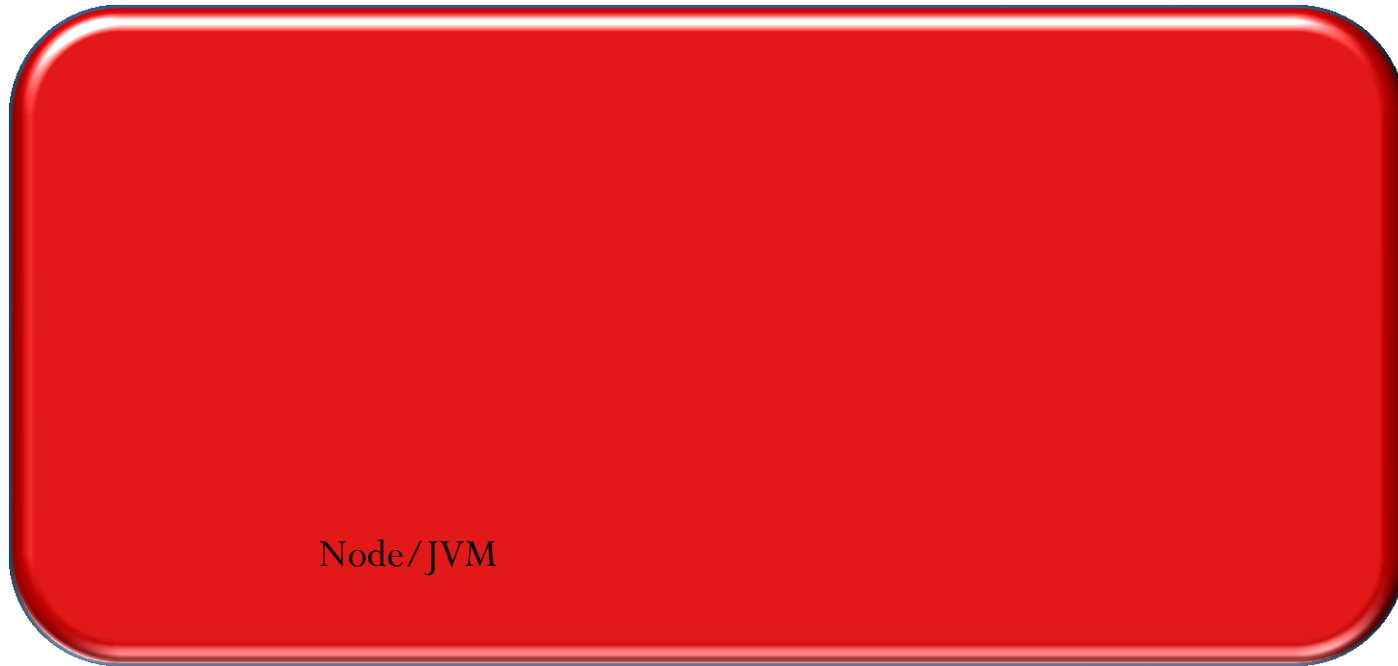
```
#ifndef _CCONST_H
#define _CCONST_H

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 */
```

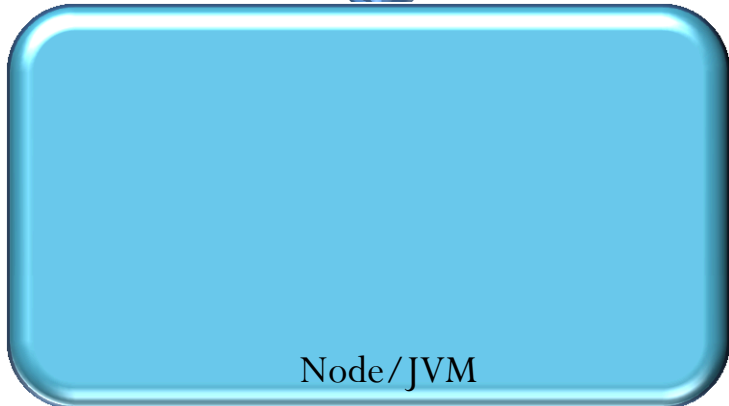
Tutorial

Simple deployment

Exercise



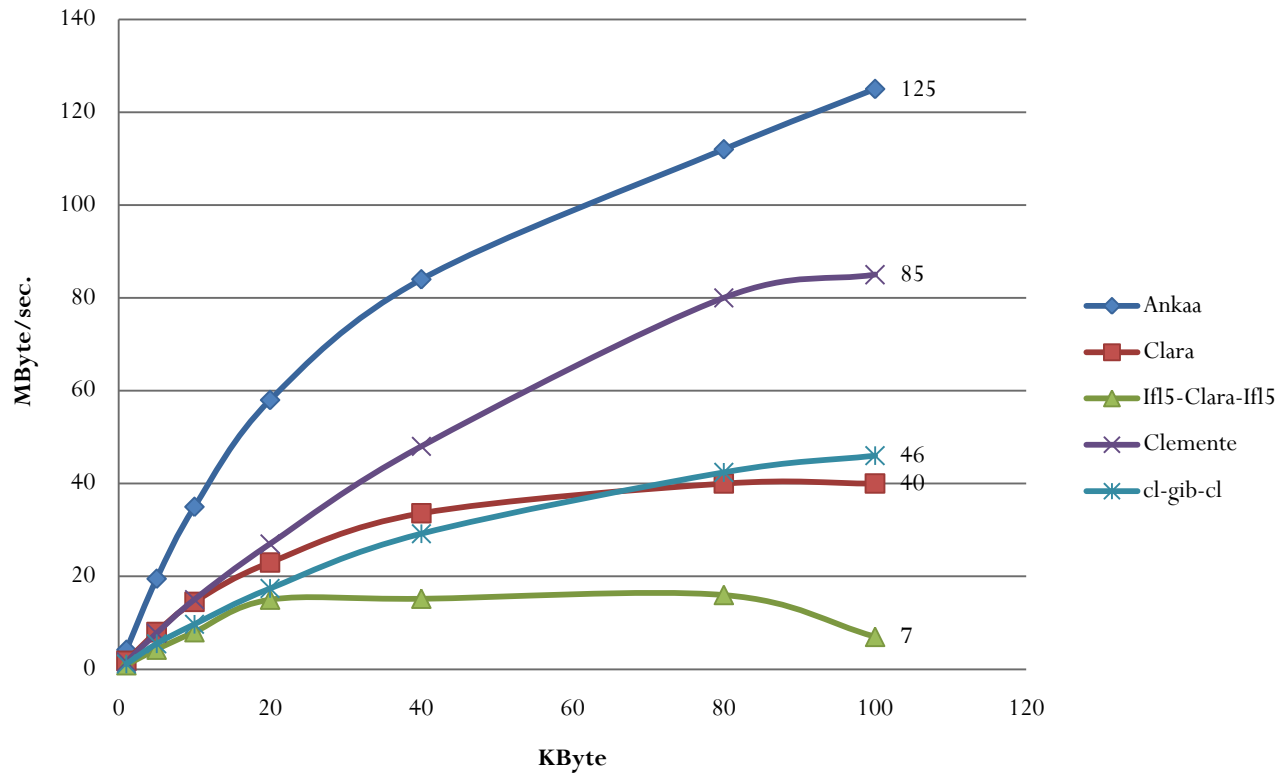
Node/JVM



Node/JVM

Data Throughput

Clara data throughput



Data Rate

Clara data rates

