

On Component Models to Deploy Application on Clouds

Christian Perez

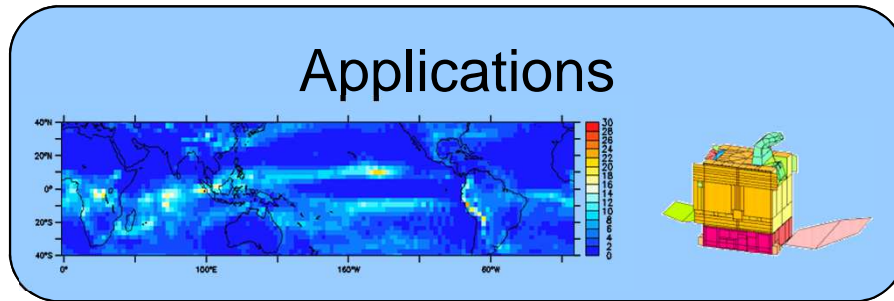
Avalon Team

LIP, ENS Lyon, France

9th workshop of the
Joint Laboratory for Petascale Computation

Lyon 2013, June 12-14

Deploying an Application



?

CPU/data-intensive Scientific Applications

- From “simple” to code coupling
 - Structure complexity
 - “New” forms of interactions (MR)

Computing platforms

- Different characteristics
 - Performance, energy, size, cost, reliability, QoS, etc.
- Hybridization
 - Sky computing, HPC@Cloud, Exascale, Spot instance

Objectives

- Expressiveness simplicity
- Application portability
- Resource specific optimizations
 - Elastic resource management
 - Energy consumption

**Super-computers
(Exascale)**

Large scale

**Grids
(EGI)**

Heterogeneity

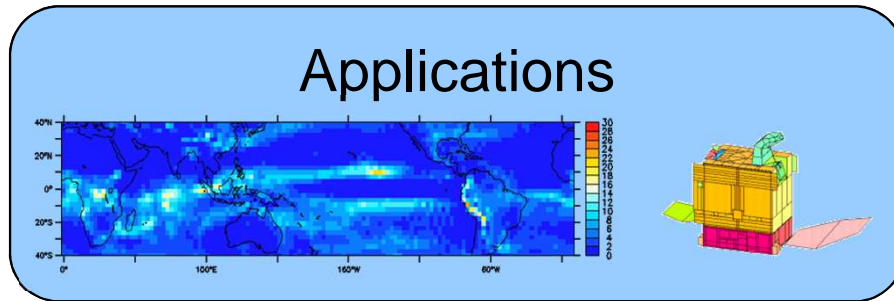
**Desktop
Grids**

Volatility

**Clouds
(IaaS, PaaS)**

On demand

Deploying an Application



Resource “independent” model

“Compilation”

+ Resource model
+ Objective function

Resource dependent model

Super-
computers
(Exascale)

Large scale

Grids
(EGI)

Heterogeneity

Desktop
Grids

Volatility

Clouds
(IaaS, PaaS)

On demand

CPU/data-intensive Scientific Applications

- From “simple” to code coupling
 - Structure complexity
 - “New” forms of interactions (MR)

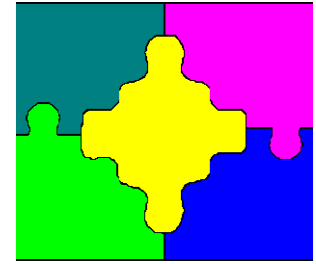
Computing platforms

- Different characteristics
 - Performance, energy, size, cost, reliability, QoS, etc.
- Hybridization
 - Sky computing, HPC@Cloud, Exascale, Spot instance

Objectives

- Expressiveness simplicity
- Application portability
- Resource specific optimizations
 - Elastic resource management
 - Energy consumption

Software Component



Technology that advocates for composition

- Old idea (late 60's)
- *Assembling* rather than *developing*

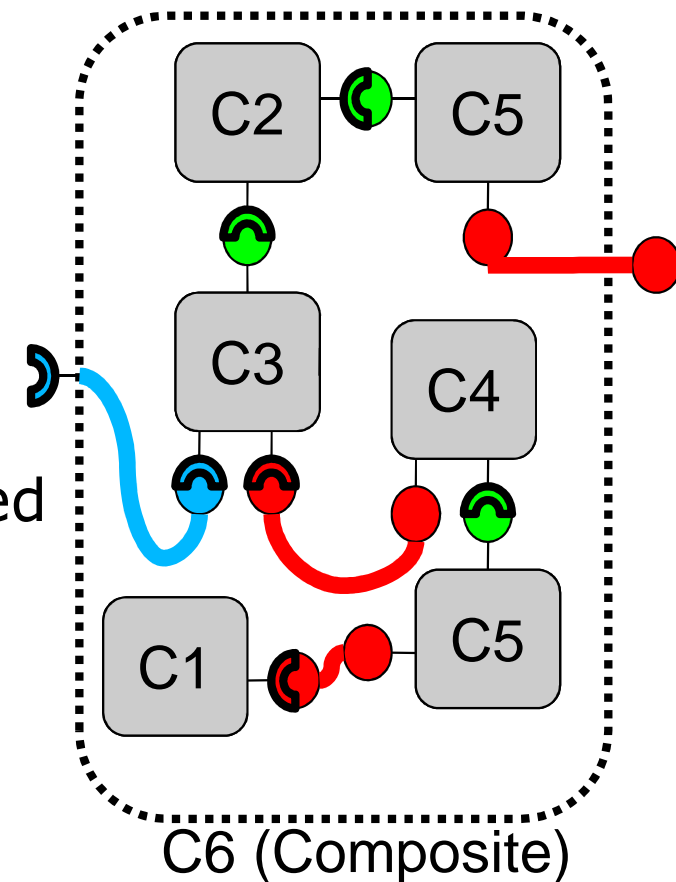
Many models

- CCA, CCM, Fractal, OGSi, SCA, ...

Pre-defined set of interactions

- Usually function/method invocation oriented

➤ *Programming model vs execution model*



Outline of the talk

Context

FP7 PaaSage Project (2012-2016)

- Overview of the project
- CPIM / CPSM

ANR MapReduce Project (2010-2014)

- Overview of HLCM
- Defining MapReduce Skeletons in HLCM

Conclusions & Perspectives



Define your application once —
deploy it at the full spectrum
of the Clouds

PaaSage: Model-based Cloud Platform Upperware

48 months, Oct 2012-Sep 2016

The Consortium



Cloud technology
providers



Application
developers



Research

Technology
Transfer



EDB ErgoGroup
Creating a New IT Experience



Lufthansa Systems



The “pain”

Many different Cloud platforms

- Heterogeneous
- API and architecture are not standardized
- Interdependence between the client application and the cloud platform
- Porting an existing application to one of the Cloud platform is still a challenging task
- Lacking support/tools for analysing and porting existing applications

“Developing once and deploying on many Cloud” is not the reality today

- Lacking support/tools for deployment and execution without vendor lock-in
- ICT businesses (and also ALL businesses) need:

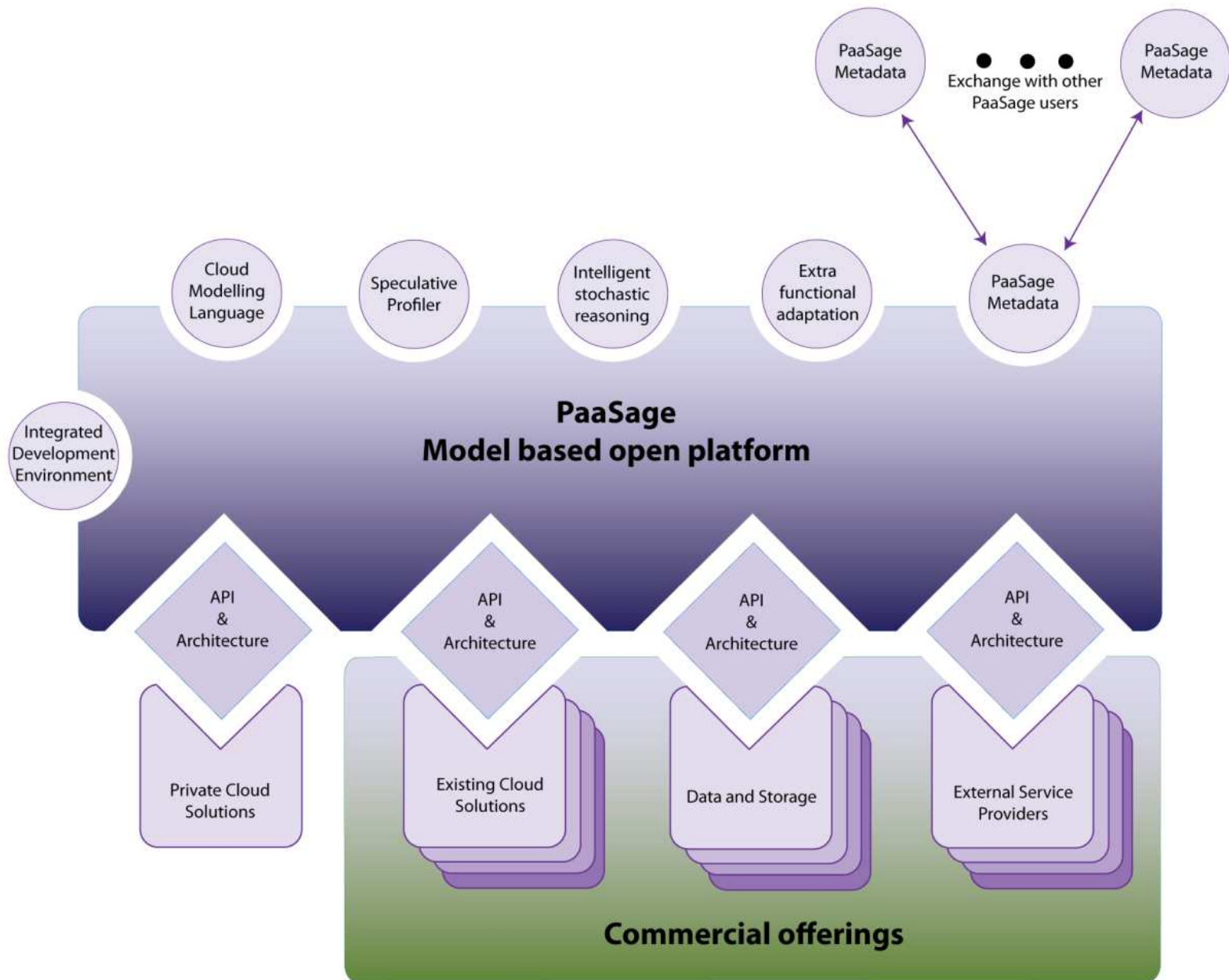
- A development and deployment platform
- An appropriate methodology

for a technology-neutral approach while targeting best performance and abstracting technical specific details

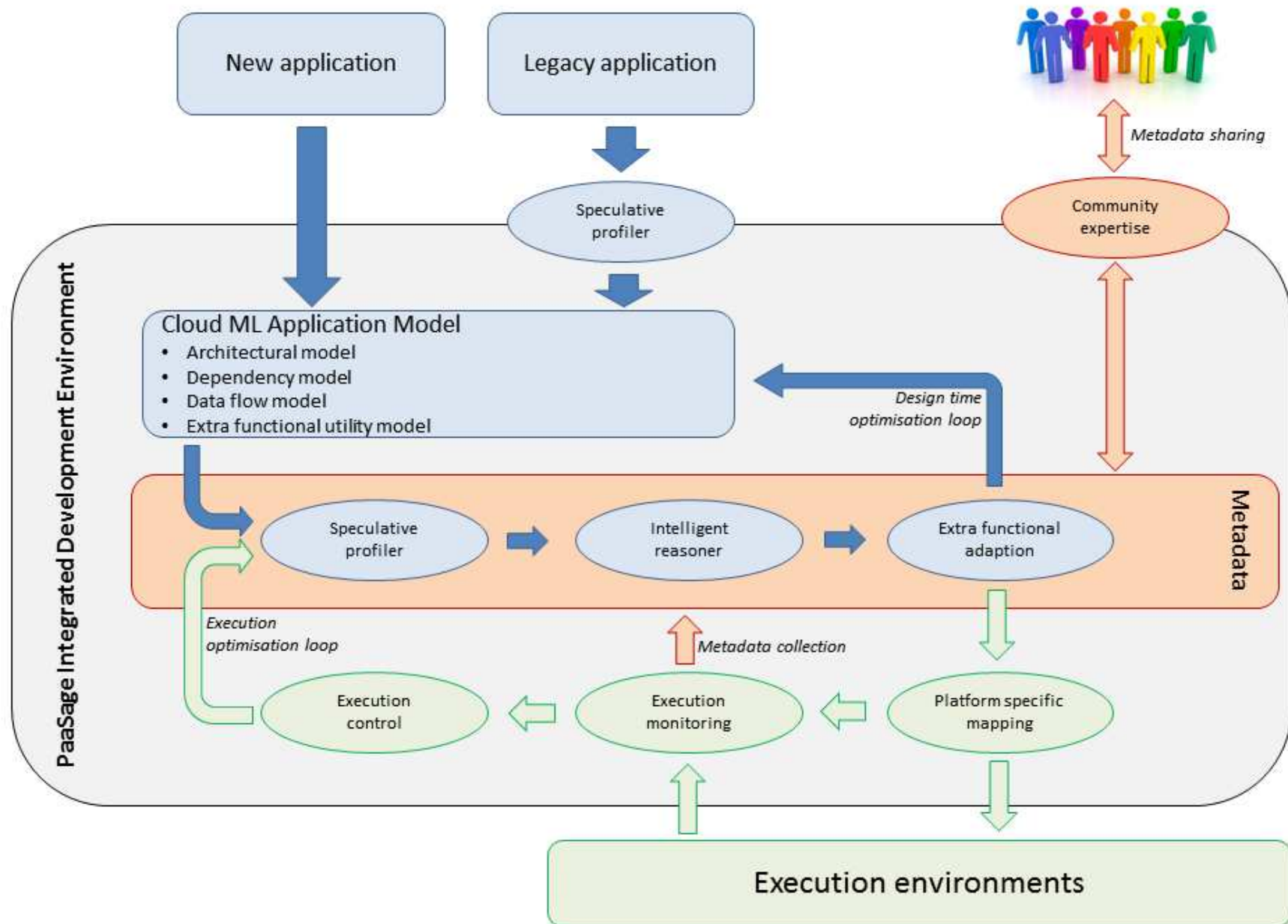
Why - Challenges

- Accessing Cloud solutions in a seamless and efficient way.
- Analysing and managing architectural changes.
- Integration with other applications in and outside the Cloud, or in different Clouds.
- Specifying Key Performance Indicators for Cloud applications in a solution-independent way and monitoring these.
- Understanding resource and cost models of Clouds and mapping their needs to these models.
- Increased complexity for legacy applications.

The Architecture



The Workflow



CloudML / PaaSage

<http://www.cloudml.org/>

- Developed by SINTEF (Norway)
- Domain-specific language (DSL) for modeling the provisioning and deployment of multi- cloud systems at design-time
- Overview
 - Cloud Provider Independent Model (CPIM)
 - Node type (VM), Component type (any deployable artifact)
(Web Server, DB, WAR Container, WAR, etc)
 - Cloud Provider Specific Model (CPSM)
 - Node instance, component instance
(constraints on almost all elements)
 - “Connector” Use/Provide ID
(apt/ssh, dbref, warref, etc)

MapReduce Component Based Skeletons

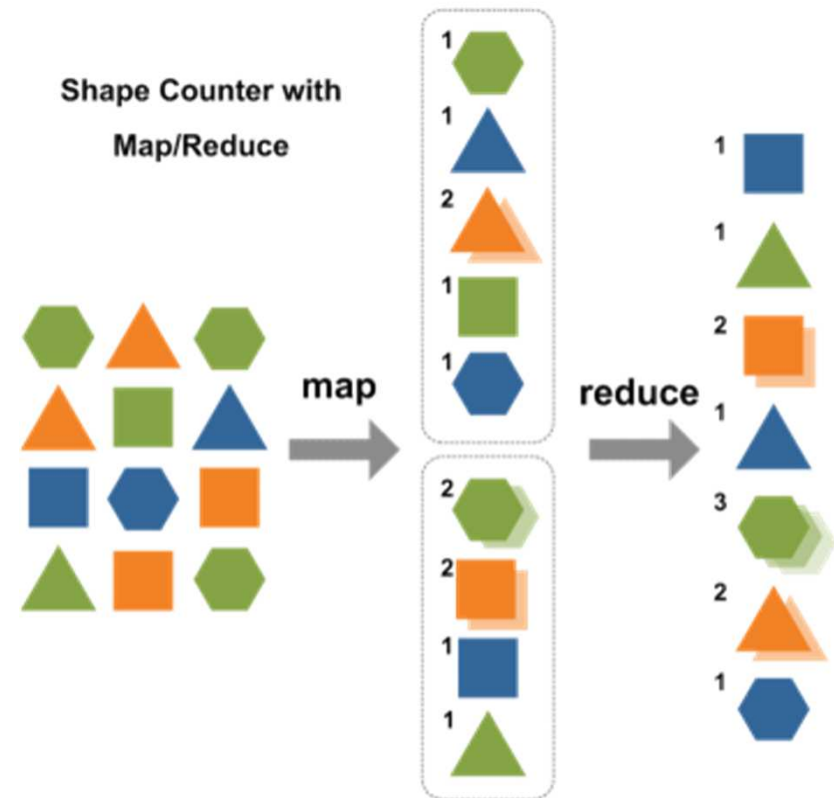
Scalable Map-Reduce Processing

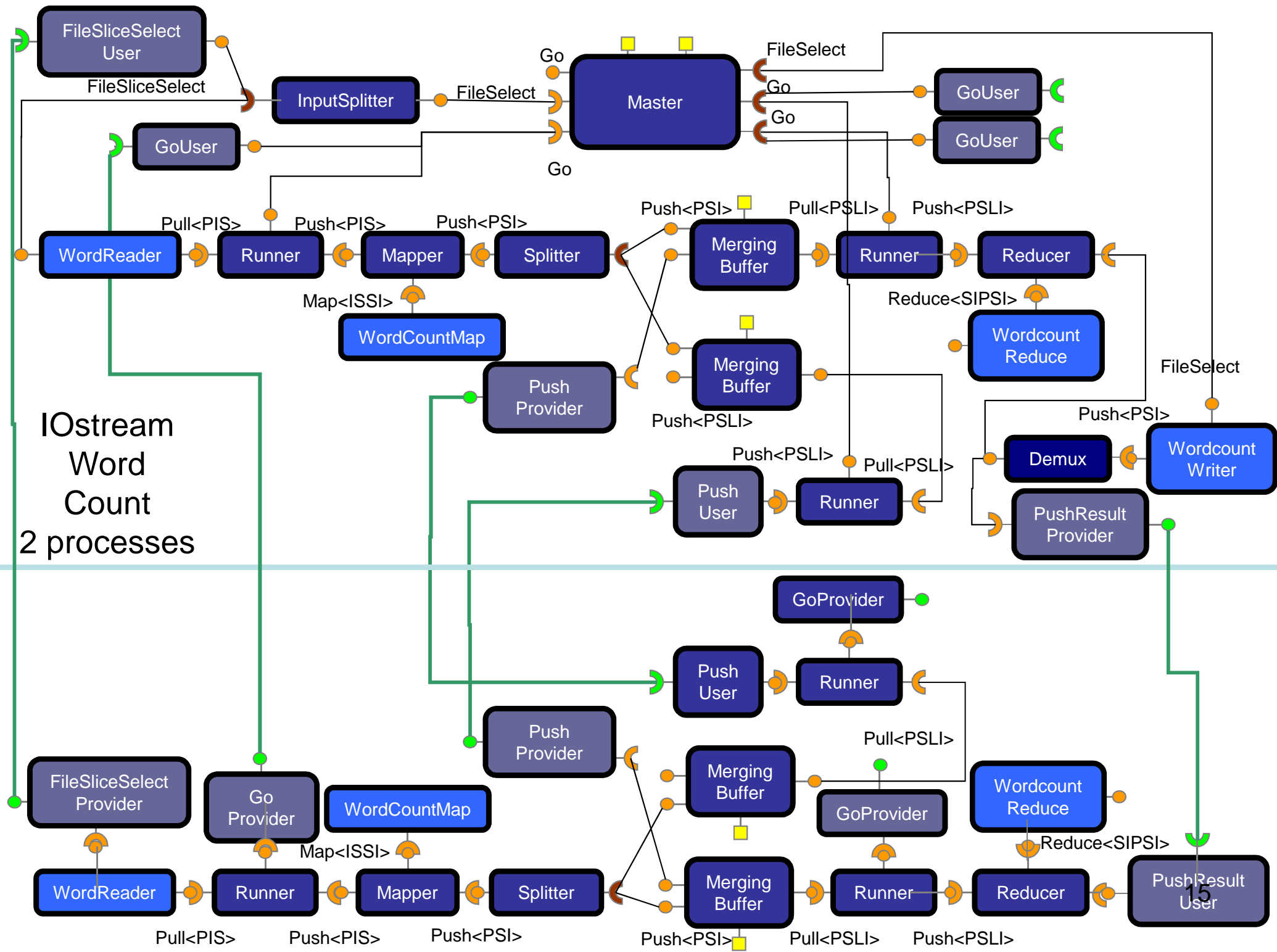
ANR Project Map-Reduce
(ARPEGE, 2010-2014)

Partners: INRIA (KerData - leader, Avalon, Grand Large), ANL, UIUC, JLPC, IBM, IBCP, MEDIT

Goal: High-performance Map-Reduce processing through concurrency-optimized data processing

URL: mapreduce.inria.fr





bs_page_size bs_replica_count
bs_input_block_size

BSImport

ImporterBS

bs_cfg
bs_id

BSInputSplitter

Master

FileSelect

GoUser

GoUser

BSSliceSelect

FileSelect

Go

Go

GoUser

Pull<PIS>

Push<PIS>

Push<PSI>

WordReader
BS

Runner

Mapper

Splitter

Merging
Buffer

Runner

Reducer

Map<ISSI>

Reduce<SIPSI>

WordCountMap

Push
Provider

Merging
Buffer

Wordcount
Reduce

Demux

Wordcount
Writer

BSSliceSelect
User

Blobseer
Word
Count

2 processes

Push
User

Runner

PushResult
Provider

GoProvider

Push
User

Runner

Push
Provider

Merging
Buffer

Pull<PSLI>

Wordcount
Reduce

Reduce<SIPSI>

BSSliceSelect
Provider

Go
Provider

WordCountMap

Map<ISSI>

WordReader
BS

Runner

Mapper

Splitter

Merging
Buffer

Runner

Reducer

PushResult
User

Pull<PIS>

Push<PIS>

Push<PSI>

Push<PSI>

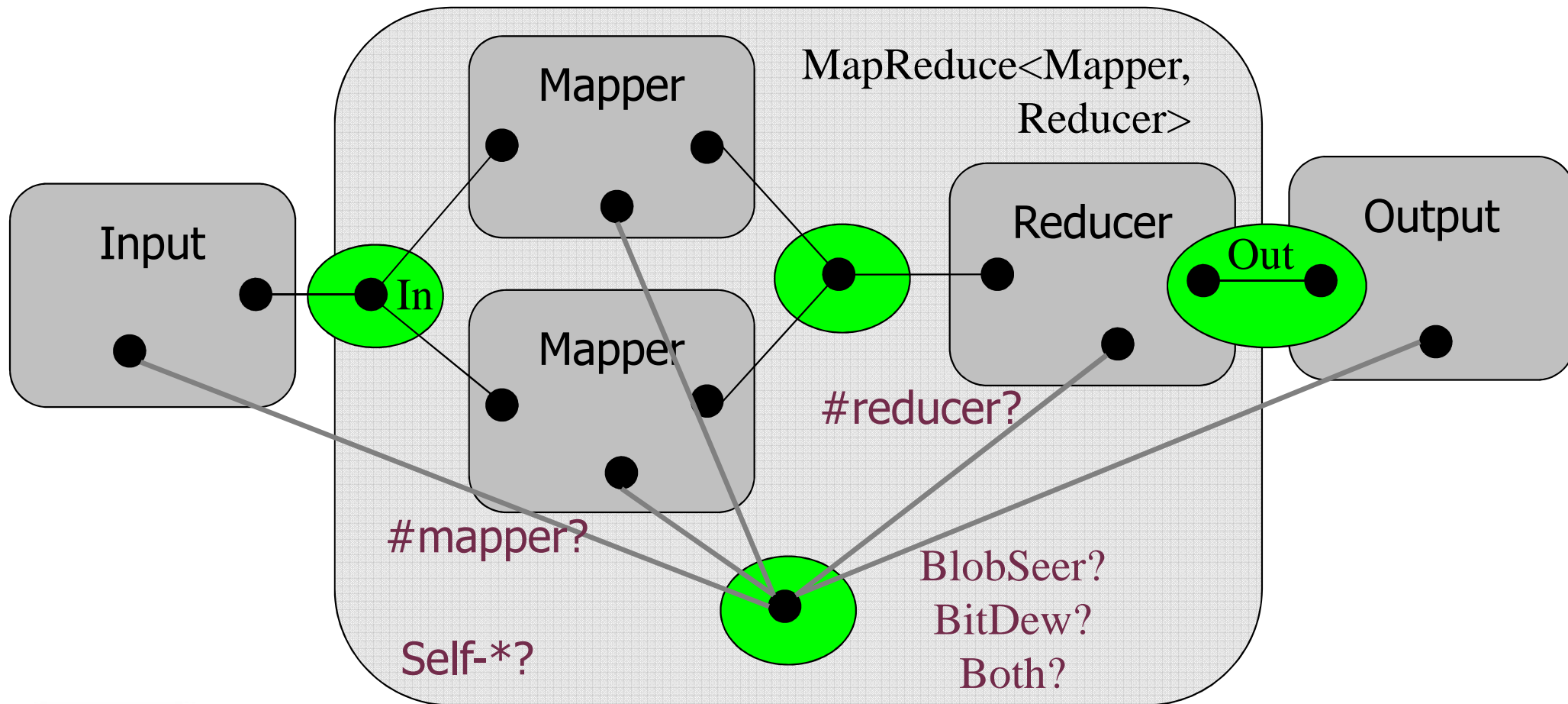
Pull<PSLI>

Push<PSLI>

16

Towards a MapReduce Skeleton

Component `MapReduce<Component Map, Component Reduce>`
exposes `{ In, Out }`



Objectives

Enable code-reuse

- E.g. mapper or reducer code
- Let expert develop a piece of code not tied to a framework

Enable *adaptation* when re-using code

- E.g. reducer “sum” not specific to a particular type of data
- Let re-use code with parameterization options

Enable any kind of composition operators

- E.g. mapper or reducer may interact with a DB
- Do not impose any communication models (framework)

Enable efficient implementation of composition operators

- E.g. enable resource specific optimization

How to Achieve Those Objectives?

Enable code-reuse

- Software Component
 - Primitive component for re-using implementation code
 - Composite component for re-using assemblies of components

Enable *adaptation* when re-using code

- Genericity

Enable any kind of composition operators

- Connectors

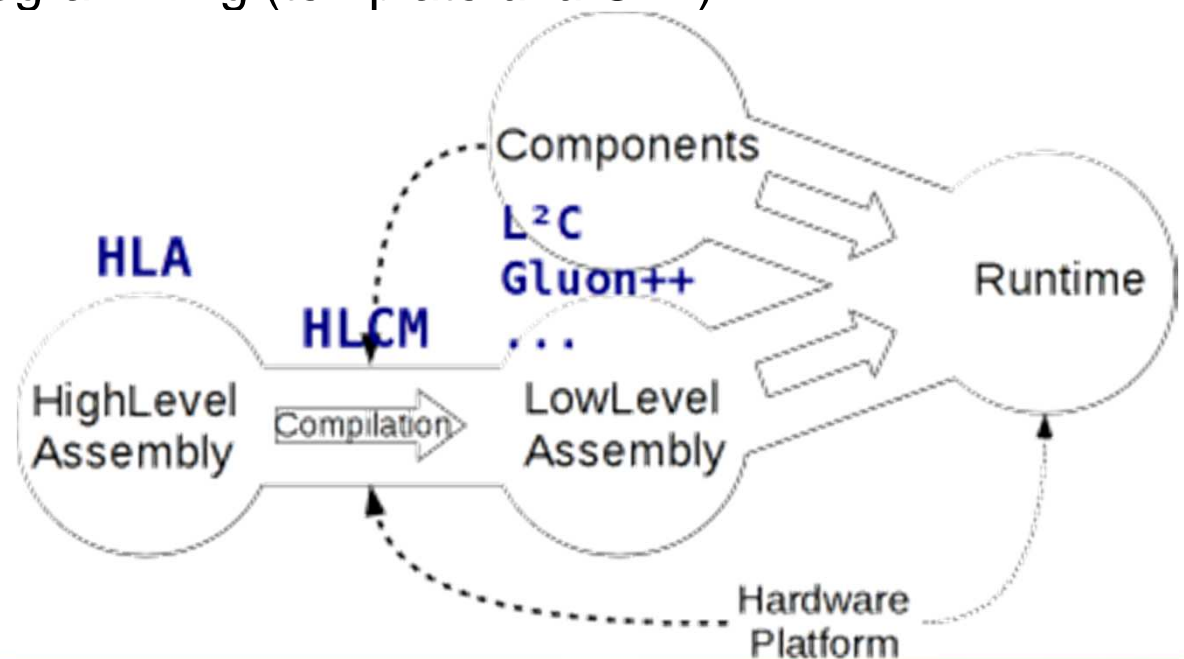
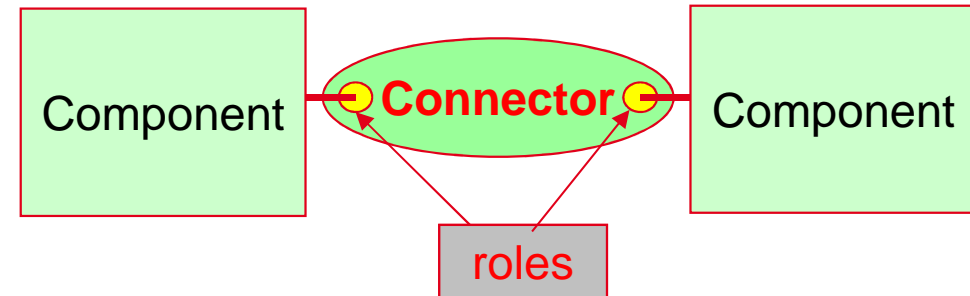
Enable efficient implementation of composition operators

- Open connection

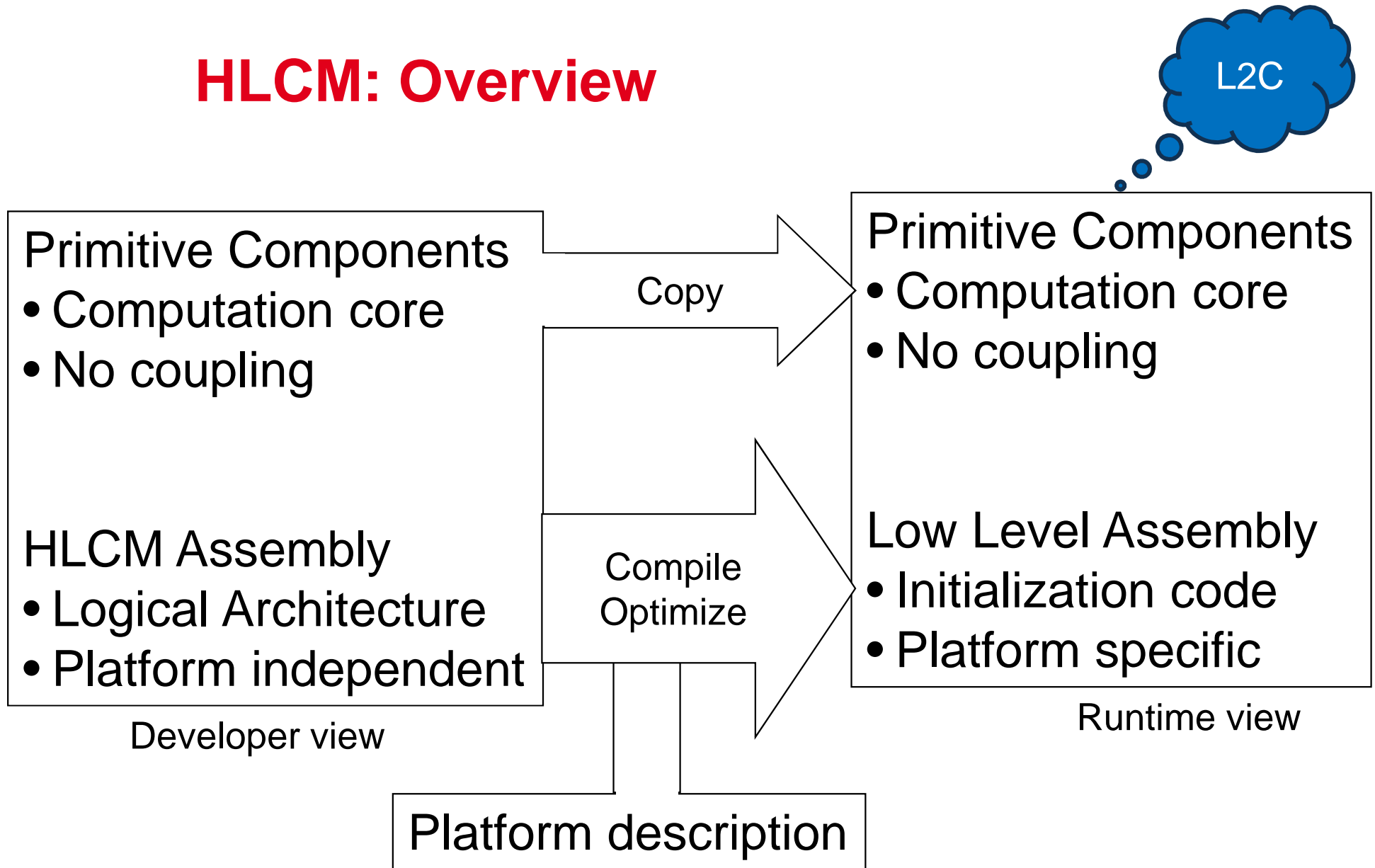
HLCM: High Level Component Model

Major concepts

- Component model (hierarchical)
 - Primitive and composite
- Connector based
 - Primitive and composite
- Generic model
 - Support meta-programming (template à la C++)
- Currently static



HLCM: Overview

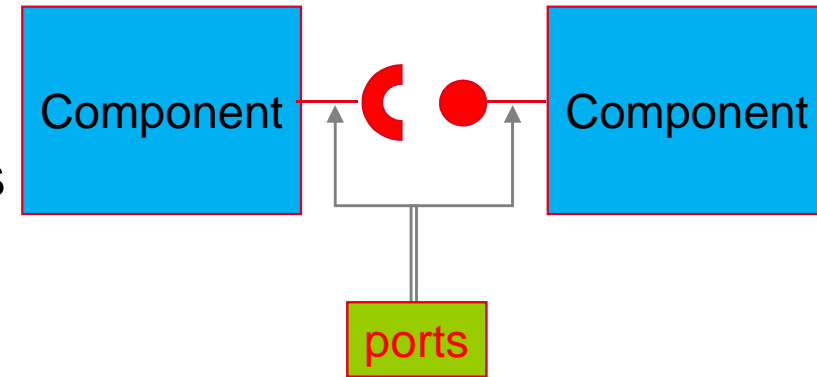


HLCM compiler (hla to l2c) GPL

Connectors

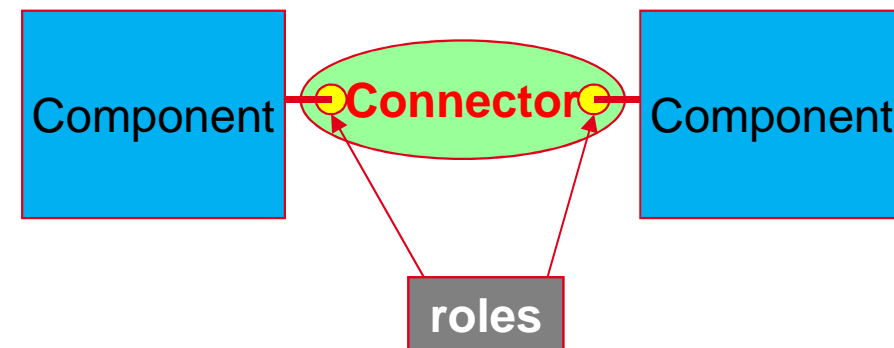
Without connectors

- Direct connection between ports through model provided interactions

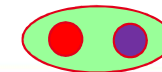


With connectors

- Originally defined in ADLs
- Connectors reify connections
 - A name
 - A set of roles
- Any number of roles
- Can be 1st class entities
 - Provided by the underlying model
 - User implemented



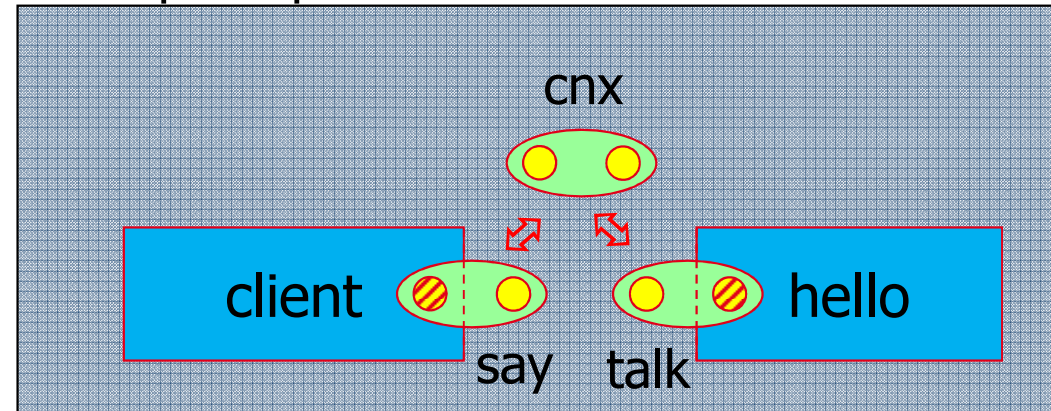
```
connector UseProvide  
< role user, role provider >;
```



HLCM: Composite Component

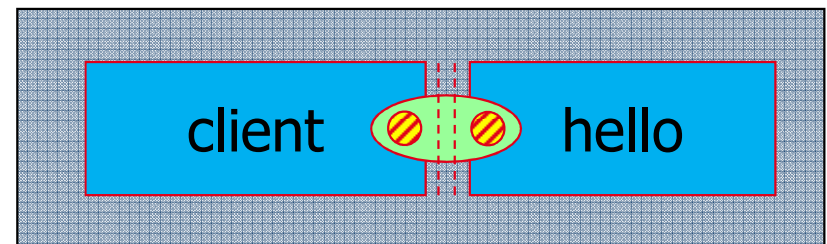
```
component Example { }  
  
composite ExampleImpl  
implements Example  
{  
  components.  
    HelloComponent hello;  
    ClientComponent client;  
  
  connections.  
    merge(hello.talk, client.say);  
}
```

ExampleImpl



Results in

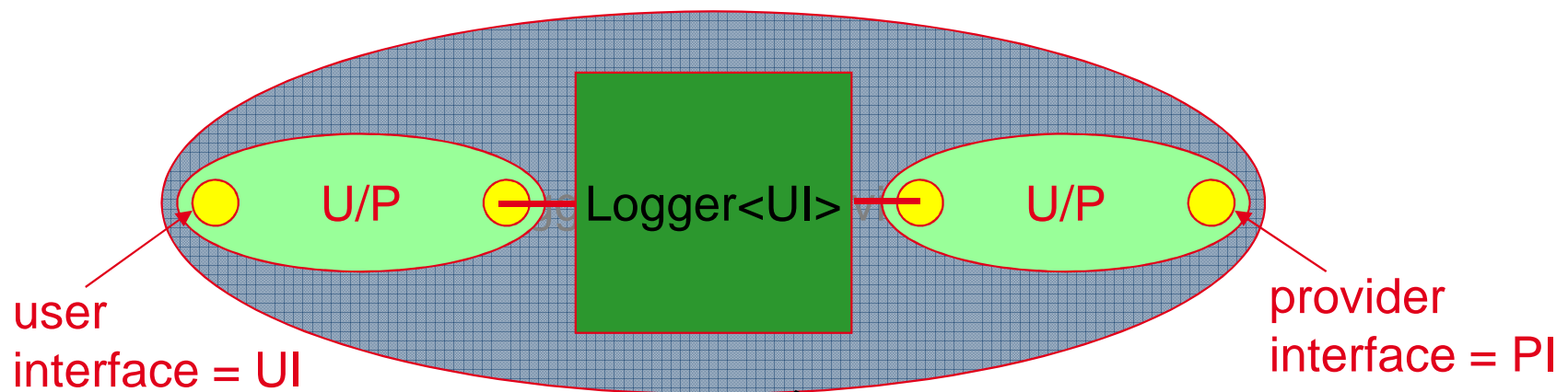
ExampleImpl



HLCM: User Implemented Connector

```
connector Logger{ provider : L_CPP_PROVIDE<PI>  
                  user   : L_CPP_USE<UI>      }
```

```
generator LoggingUP<datatype UI, datatype PI>  
with { SuperInterface(UI, PI) }  
implements Logger<UI, PI>  
{  
  proxy : Logger<UI>  
  merge(proxy.clientSide.user, this.user);  
  merge(proxy.serverSide.provider, this.provider);  
}
```



When PI subtype of UI and
user.host = provider.host

WordCount and HLCM/L2C

HOMR: Home-made MapReduce

HLCM WordCount Components

```
component Map<datatype T>  
exposes {  
    map: Provide<T>;  
}
```

```
component Reduce<datatype C, datatype R>  
exposes {  
    combine: Provide<C>;  
    reduce: Provide<R>;  
}
```

WordCount Map Example

```
#include <l1cmcpp/l1cmcpp3.h>
#include <mapreduce/map.h>

typedef Map< int, string, string, int > Map_int_string_string_int;

class WordCountMap: virtual public Map_int_string_string_int {
public:
    void map ( const int&, const string& word,
               list< pair< string, int> >& result )
    {
        result.push_back(make_pair(word, 1)); }
};

// #implements=Map<int_string_string_int>
LCMP(WordCountMap)
    L_CPP_PROVIDE(Map_int_string_string_int, map);
LEND

// component Map<datatype T> exposes {
//     map: Provide<T>;
// }
```

WordCount Reduce Example

```
typedef Combine<string, int > Combine_string_int;
typedef Reduce<string, int, pair<string, int > > Reduce_string_int_pair_string_int;

class WordCountReduce: public virtual Combine_String_int {
public:
    void reduce (const string& key, const list< int >& values,
                 list< pair< string, int > >& result ) {
        result.push_back(make_pair(string(key),
                                     accumulate(values.begin(), values.end(), 0)));
    }
};

//#implements=Reduce<Combine_string_int, Reduce_string_int_pair_string_int>
LCMP(WordCountReduce)
    L_CPP_PROVIDE(Combine_string_int, combine);
    L_CPP_PROVIDE(Reduce_string_int_pair_string_int, reduce);
LEND

// component Reduce<datatype C, datatype R> exposes {
// combine: Provide<C>;
// reduce: Provide<R>;
// }
```

MapReduce Local M mappers N Reducers

```
component MRFullLocalNM<Integer M, Integer N, String ifile, String ofile>  
exposes { go: Provide<Go>; }
```

```
composite MRFullLocalNMimp<Integer M, Integer N, String ifile, String ofile>  
implements MRFullLocalNM<M,N, ifile, ofile> {
```

components:

```
    master: MRMaster<ifile, ofile>;  
    input_splitter: MRInputSplitter;  
    demux_writer: MRDemux<Push_pair_string_int>;  
    writer: MRWordcountWriter<Push_pair_string_int>;  
    mr: MRFullMapReduceNM<M,N>;
```

connections:

```
    merge(master.input_file_selector, input_splitter.split);  
    merge(master.output_file_selectors, writer.write);  
    merge(demux_writer.outputs, writer.input);
```

```
    ...
```

exposes:

```
    go=master.main;
```

```
}
```

Shuffle Part (no localization constraints)

composite MRFullMapReduceNMimpl<**Integer nm, Integer nr**> implements MRFullMapReduceNM<**nm, nr**> {
components:

```
lrm:    [each ( i | [ 1 .. nm ] ) {IReadMap<FileSliceSelect, Push_pair_string_int>} ];  
splitter: [each ( i | [ 1 .. nm ] ) {MRSplitter<Push_pair_string_int>} ];  
mbm:    [each ( j | [ 1 .. nm ] ) { [each ( i | [ 1 .. nr ] ) {  
      MRMergingBuffer<3, Push_pair_string_int, Push_pair_string_list_int, Pull_pair_string_list_int> }]}];  
  
transfer_runner: [each ( j | [ 1 .. nm ] ) { [each ( i | [ 1 .. nr ] ) {  
      MRRunner<Pull_pair_string_list_int, Push_pair_string_list_int> }]}];  
  
mbr: [each ( j | [ 1 .. nr ] ) {  
      MRMergingBuffer<3, Push_pair_string_int, Push_pair_string_list_int, Pull_pair_string_list_int> }];  
lrr: [each ( i | [ 1 .. nr ] ) {IReduce} ];
```

connections:

```
each ( im | [ 1 .. nm ] ) { merge(lrm[im].mapper_output, splitter[im].input); } // mapper => splitter  
each (im| [ 1 .. nm] ) { // splitter => mbm  
  each (ir | [ 1 .. nr] ) { merge(splitter[im].outputs, mbm[im][ir].input); }  
}  
each ( im | [ 1 .. nm] ) { each ( ir | [ 1 .. nr ] ) {  
  merge(mbm[im][ir].output, transfer_runner[im][ir].input); // mbm => transfer  
  merge(transfer_runner[im][ir].output, mbr[ir].list_input); // transfert => mbr  
} }  
each ( ir | [ 1 .. nr] ) { // mbr => reducer  
  merge(mbr[ir].output, lrr[ir].reduce_runner_input);
```

}

Automatic C++/Corba Connection Resolution

HLCM/L2C

- Connector < user, provider>
- 2 primitive implementations: C++ & Corba

Rules

- Privilege C++ connection over Corba connection
- C++ connections only possible when on **SameProcess**
- Corba connections when on **NotSameProcess**

Example

```
connector ProxyConnector<datatype T>
{
    provider: L_CPP_PROVIDE<T>    user: [ L_CPP_USE<T> ] }
generator ProxyGenCorbaHello with { NotSameProcess }
implements ProxyConnector<CppHello> {
    pu: hwcorbaproxyclient; pp: hwcorbaproxyserver;
    merge( { user=this.user} , pu.greeter);
    merge( pu.corba_greetservice, pp.corba_greeter);
    merge( pp.greetservice, { provider=this.provider} );
}
```

HLCM: Mapping Constraints (In progress)

HLCM/L2C

- Components mapped to a vector of ID (string)
 - No semantic associated
 - In this talk, only process id

Mapping Constraints

- Design a process: `Process("P"), Process(c1, "P")`
- Same process: `SameProcess(c1, c2, c3)`
- Distinct process: `DistinctProcess(c1, c2, c3)`

Example

composite I

*with { **Process(s, "S"), SameProcess(s,r), DistinctProcess(c)** }*

implements ComponentType {

s: server; c: client; r: runner;

}

Shuffle Part (Constrained Version)

```
composite lReadMapLocImpl<datatype SS, datatype PsPSI, String sloc, Integer loc>
with { Process(""+sloc+loc)} implements lReadMapLoc<SS, PsPSI, sloc, loc> {
components:
    lrm: lReadMap<SS, PsPSI>;
    ...
}
```

```
composite MRFullMapReduceLocNMimpl<Integer nm, Integer nr>
implements MRFullMapReduceLocNM<nm, nr> {
components:
    lrm:      [each ( i | [ 1 .. nm ] ) {lReadMapLoc<FSS, Push_pair_string_int, "PM_", i>} ];
    splitter: [each ( i | [ 1 .. nm ] ) {MRSplitterLoc<Push_pair_string_int, "PM_", i>} ];
    mbm:      [each ( i | [ 1 .. nm ] ) { [each ( j | [ 1 .. nr ] ) {
                                                MRMergingBufferLoc<3, Push_pair_string_int, Push_pair_string_list_int,
                                                Pull_pair_string_list_int, "PM_", i>}}]];
    transfer_runner: [each ( i | [ 1 .. nm ] ) { [each ( j | [ 1 .. nr ] ) {
                                                MRRunnerLoc<Pull_pair_string_list_int, Push_pair_string_list_int, "PM_", i>
                                                }]]];
    mbr: [each ( j | [ 1 .. nr ] ) {
        MRMergingBufferLoc<3, Push_pair_string_int, Push_pair_string_list_int,
        Pull_pair_string_list_int, "PR_", j> }];
    lrr: [each ( j | [ 1 .. nr ] ) {lReduceLoc<"PR_", j> } ];
Connections: // identical to Local Version
}
```

Architecture Variants

Shuffle

- 1 buffer between a Mapper and a Reducer
- 1 output buffer per Mapper +
1 input buffer per Reducer

Localization

- “Sequential components” (Master, Writer)
- Mappers & Reducers

To do

- Transfer scheduling algorithm (S. Gault)
- Interfaces as parameter
- Combining operator
- etc

Conclusion

Application Specialization to Resources

- Component model as an architectural application description

PaaSage

- Simplify cloud utilization through artifact mapping to VMs
- Relies on CPIM/CPSM

HLCM

- General purpose (Hierarchical, connector based, generic)
- Transformation more challenging
- Initial working example for MapReduce

Future Work

- Add (cloud) resource model to HLCM
- Develop mapping algorithms (heuristics?)
- What about dynamicity?