Overview: Day 1

- **September 03, Day 1:**
- 13:00 - 13:15: Introductions
- 13:15 - 14:30: Developing Distributed Applications Using SAGA (Jha)
- 14:15 - 15:30: Introducing the API (Merzky)
- **BREAK**
- 16:00 - 16:30: Introducing the API -- II (Merzky)
- 16:40 - 17:30: Installing and Deploying SAGA (Ole)

Overview: Day 2

Day 2:

- 09:00 - 10:30: SAGA: Simple Examples, Programming Manual, SAGA-Shell (Hartmut)
- **BREAK**
- 11:00 - 12:30: SAGA Programming Exercise (Hartmut)
- 12:30 - 13:00: SAGA-based frameworks and Tools (Jha)
- **LUNCH**
- 14:00 - 15:00: Using SAGA For your Application Needs I
- 15:00 - 16:00: Using SAGA for your Application Needs II
- 16:00 - 16:30: Closing

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

e-Science Institute

SAGA: An Introductory Lecture

The SAGA Team

http://saga.cct.lsu.edu

NGS
National Grid Service

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

e-Science Institute

Understanding the Landscape of Distributed Applications

Developing Distributed Applications Using SAGA

Outline (1)

- Simple API for Grid Applications (SAGA)
  - Introduction to Simple API
    - Role of adaptors
    - SAGA as an emerging Standard
- Understanding Distributed Applications (DA)
  - Challenges of DA ? Differ from HPC or || App?
  - Rough Taxonomy of Distributed Applications
  - Using SAGA to develop DA
    i. Distributed Execution Mode Legacy (Implicit)
    ii. Explicit coordination and distribution
    iii. Frameworks: support characteristics or patterns
- Understanding the SAGA Landscape
  - Interface/Specification, C++ Engine, Adaptors

CENTER FOR COMPUTATION & TECHNOLOGY

Outline (2)

e-Science Institute

- Applications Developed Using SAGA:
  - Replica-Exchange, $C0_2$ Sequestration (EnKF)
  - MapReduce
    - Interoperabilty across Infrastructure: Grids, Grids-Clouds
- Other SAGA Projects (Advantage of Standards)
  - gLite, GANGA-SAGA project
  - JSAGA and XtreemOS
  - Service Discovery (SD) and uptake gLite
  - DESHL (DEISA) -- Tool or Application?
  - SAGA Condor project
  - SAGA Gateways project

A take on "What are Grids?"

Production and non-production Grids are:

- Heterogenous:
  - Operating Systems, Libraries, software stack
  - Middleware service versions and semantics
  - Administrative policies – access, usage, upgrade
- Complex:
  - Production Grid Service with high QoS non-trivial
  - Derived from above as well as inherently
- (Can be) Dynamic
  - Depending on relevant time-scales

Not only are Grids difficult to operate etc., it is also difficult to develop, deploy & execute applications for such environments

SAGA: A Quick Tour

- There exists a lack of:
  - Programmatic approaches that provide common grid functionality at the correct level of abstraction for applications
  - Ability to hide underlying complexity of infrastructure, varying semantics, heterogeneity and changes from the application-developer
- Simple, integrated, stable, uniform and high-level interface
  - Simple: 80:20 restricted scope
  - Integrated: Similar semantics & style across commonly used distributed functional requirements
  - Stable: Standard
  - Uniform: Same interface for different distributed systems
- SAGA: Provides the high-level abstraction that application developers need that will work across different distributed systems
  - Shields details of lower level middle-ware and system issues
  - Enables the details of distribution to be left out

CENTER FOR COMPUTATION & TECHNOLOGY

SAGA Example: Copy a File

Application Level Interface

e-Science Institute

```
#include <string>
#include <saga/saga.hpp>

void copy_file(std::string source_url, std::string target_url)
{
  try {
    saga::file f(source_url);
    f.copy(target_url);
  }
  catch (saga::exception const &e) {
    std::cerr << e.what() << std::endl;
  }
}
```

The interface is simple and the actual function calls remain the same

CENTER FOR COMPUTATION & TECHNOLOGY

SAGA Interface
Job Submission API

e-Science Institute

```
01: // Submitting a simple job and wait for completion
02: //
03: saga::job_description jobdef;
04: jobdef.set_attribute ("Executable", "job.sh");
05:
06: saga::job_service js;
07: saga::job job = js.create_job ("remote.host.net", jobdef);
08:
09: job.run();
10:
11: while( job.get_state() == saga::job::Running )
12: {
13:   std::cout << "Job running with ID: "
14:             << job.get_attribute("JobID") << std::endl;
15:   sleep(1);
16: }
```

The interface is simple and the actual function calls remain the same

CENTER FOR COMPUTATION & TECHNOLOGY

File API Example

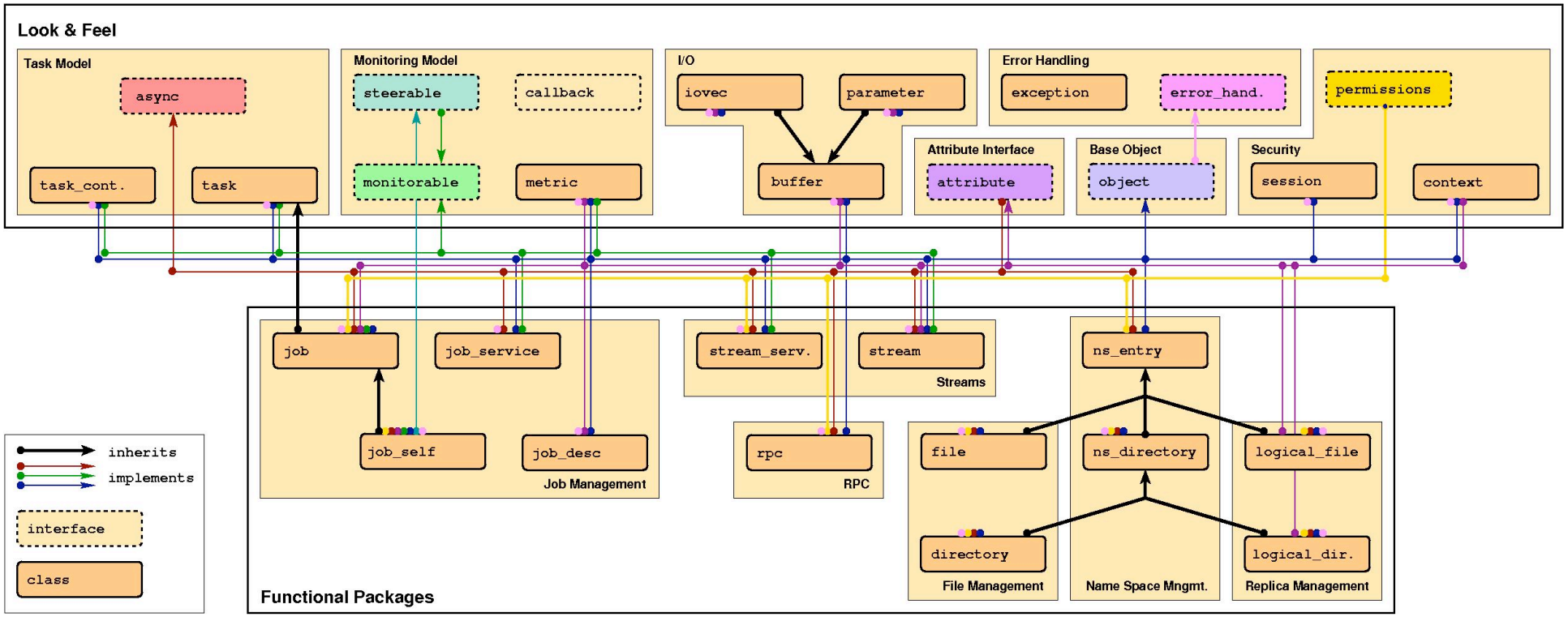
e-Science Institute

```
01: // Read the first 10 bytes of a file if file size > 10 bytes
02: //
03: saga::file my_file ("griftp://gridhub/~/result.dat);
04:
05: off_t size = my_file.get_size ();
06:
07: if ( size > 10 )
08: {
09:   char buffer[11];
10:   long bufflen;
11:
12:   my_file.read (10, buffer, &bufflen);
13:
14:   if ( bufflen == 10 )
15:   {
16:     std::cout << buffer << std::endl;
17:   }
18: }
```

CENTER FOR COMPUTATION & TECHNOLOGY

SAGA: Class Diagram

e-Science Institute



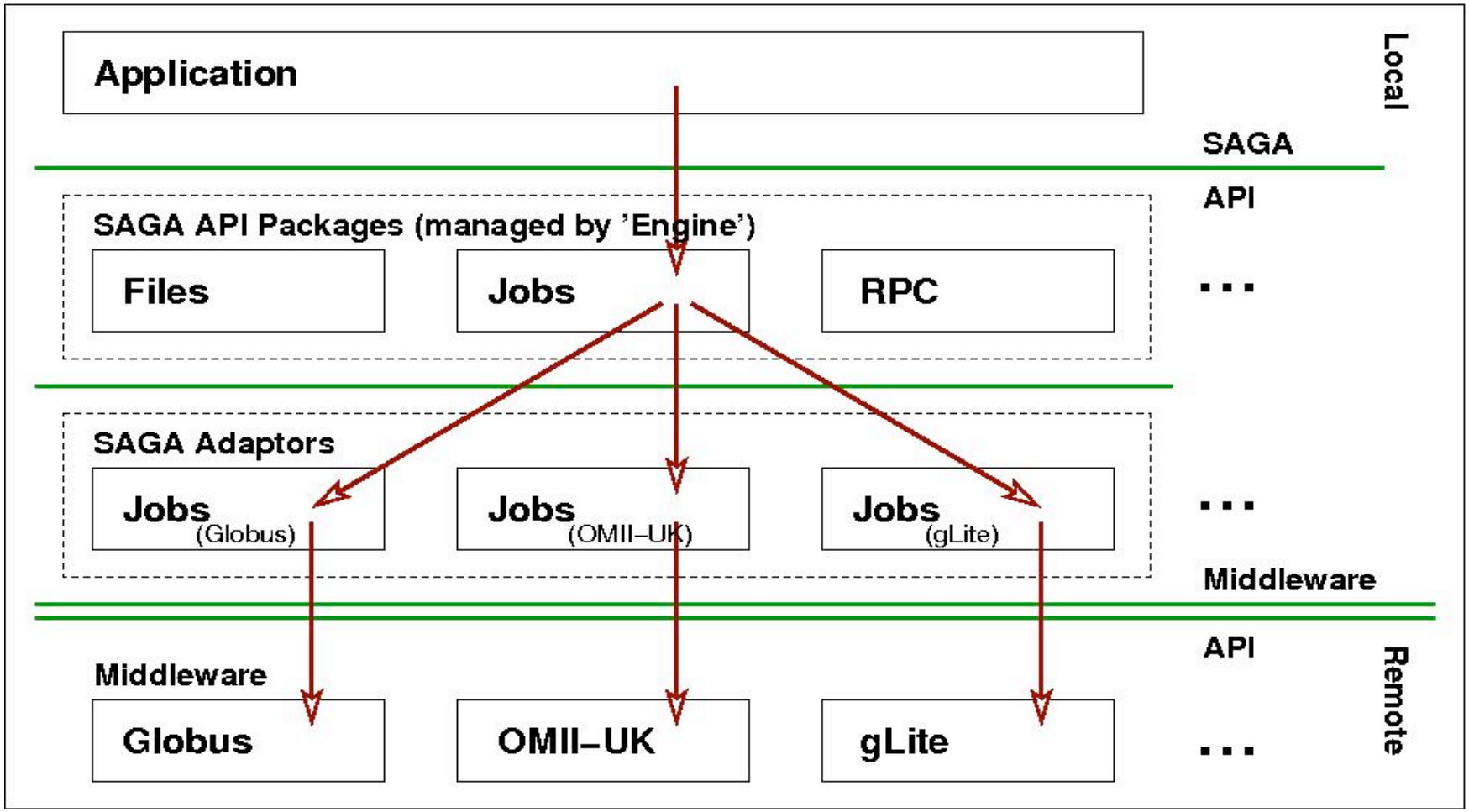
In the works: CPR, Information Services, Messaging, DAI,...

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

SAGA: Job Submission

Role of Adaptors (middleware binding)

e-Science Institute



CENTER FOR COMPUTATION & TECHNOLOGY

SAGA API: Towards a Standard

Standards promote Interoperability

e-Science Institute

- The need for standard programming interface
  - Trade-off “Go it alone” versus “Community” model
  - Reinventing the wheel again, yet again, & then again
  - MPI a useful analogy of community standard
    - Vendors (Resource Provider), Software developers, users..
    - Social/historic parallels also important
      - Time to adoption, after specification ....
  - OGF the natural choice (SAGA-RG, SAGA-WG)
    - Spin-off of the Applications Research Group
    - Driven by EU (German/Dutch), UK, US
    - Design derived from 23 Use Cases
      - different projects, applications and functionality
      - biological, coastal modelling, visualization
  - Will discuss the advantage of SAGA as a standard specification

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Developing DA with SAGA: Parallel Programming Analogy

e-Science Institute

- Distributed Development today, || programming pre-MPI
  - Bewildering # of ways of doing the basic stuff (e.g., job submission)
- MPI was a “success” in that it helped many new applications
  - MPI is simple (most things can be done with 6-8 calls)
  - MPI was a standard (stable and portable code!)
- SAGA conception & trajectory similar to MPI
  - SAGA focusses on simplicity (common vs corner case)
  - OGF specification; on path to becoming a standard
- Therefore, SAGA's Measure(s) of success:
    - Does SAGA enable “new” distributed applications?
    - Does it enable effective development of “old” distributed application

CENTER FOR COMPUTATION & TECHNOLOGY

Outline (1)

e-Science Institute

- Simple API for Grid Applications (SAGA)
  - Introduction to Simple API
  - SAGA as an emerging Standard
- Understanding Distributed Applications (DA)
  - Challenges of DA ? Differ from HPC or || App?
  - Rough Taxonomy of Distributed Applications
  - Using SAGA to develop DA
    i. Distributed Execution Mode Legacy (Implicit)
    ii. Explicit coordination and distribution
    iii. Frameworks: support characteristics or patterns
- Understanding the SAGA Landscape
  - Interface/Specification, C++ Engine, Adaptors

Understanding Distributed Applications (1)

How do they differ from traditional HPC applications?

- Performance Models:
  - Not “peak utilization” e.g., # of jobs
  - Manage scalability, faults, heterogeneity
- Usage Modes:
  - The same application has multiple usage modes
  - How applications are run, deployed and utilized is often determined by the infrastructure
  - Three vertex: Application, Infrastructure, Usage Mode
- Static versus Dynamic Execution:
  - Varying resource conditions; distributed applications should be dynamic

CENTER FOR COMPUTATION & TECHNOLOGY

Developing Distributed Applications: Barriers

e-Science Institute

- .. Underlying infrastructure characteristics:
  - Dynamical and Heterogeneous resources
  - Control (or lack thereof) on remote systems
- Computational Models of Distributed Computing not as well understood as parallel computing
  - Parallel Computing: Focus on performance! For DA?
  - Greater range of DA; no clear taxonomy
  - Underlying model of distributed infrastructure complex..
- Programming Systems for DA:
  - Incomplete? Customization? Extensibility?
  - What should end-user control? Must control? Should not? What should the system support? Role of tools?
  - No universal answers! Simplification and distillation

Infrastructure-Usage Mode: HPC vs HTC

Usage Mode	Number of Users
Batch Computing on Individual Resource	850
Exploratory and Application Porting	650
Science Gateway Access	500
Workflow, Ensemble and Paramater Sweep	250
Remote Interactive Steering and Visualization	35
Tight-Coupled Distributed Computation	10

TABLE I

TeraGrid Usage Mode distribution for 2006, the latest year for which data is available. Notice the small proportions of users/applications that utilize multiple resources collectively – concurrently or otherwise

Application Type	Characteristics & Examples
Simulation	CPU-intensive, Large number of independent jobs. e.g., Physics Monte Carlo event simulation
Production Processing	Significant I/O of data from remote sources e.g, Processing of physics raw event data
Complex Workflow	Use of VO specific higher-level services; Dependencies between tasks e.g., Analysis, Text mining
Real Time Response	Short runs; Semi-guaranteed response times, e.g., Grid operations and monitoring
Small-scale Parallelism	Allocation of multiple CPUs simultaneously; Use of MPI libraries, e.g., Protein analysis, MD

TABLE II

TYPES OF APPLICATION RUNNING ON THE OPEN SCIENCE GRID

Understanding Distributed Applications (2)

- The Future is Distributed:
  - Nature of Scientific Applications, Discovery/Research
    - Parallelism is important, but exclusively for ~20%
    - Working examples of where if you can distribute effectively, you can choose methods not critical on ||
  - Basic Trends in Computing
    - Decoupling & Delocalization of Data Production-Consumption
    - Components of computation
    - Distributed services/people

Understanding Distributed Applications(3)

- Therefore develop applications that require:
  - Distribution of compute-data-people (manage)
  - Coordination over Multiple & Distributed sites:
    - Logically or physically distributed
    - Scale-up and Scale-out
    - Peta/Exa/Atta - Scientific Applications requiring multiple-runs, ensembles, workflows etc.
  - Ability to develop simple, novel or effective distributed applications lags behind

Understanding Distributed Applications (4) Development Objective

- Interoperability: Ability to work across multiple distributed resources
- Scale-Out: The ability to utilize multiple distributed resources concurrently
- Extensibility: Support new patterns/abstractions, different programming systems and Infrastructure
- Adaptivity: Response to fluctuations in dynamic resource and availability of dynamic data
- Simplicity: Accommodate above distributed concerns at different levels *easily…*

Developing Distributed Applications (2) Challenges

- Challenge: How to develop DA effectively and efficiently, with extensibility, scalability, adaptivity and interoperabiltiy and simplicity as first-class concerns
  - Understand what can be done? What are the gaps?
  - Understand characteristics and requirements DA
- Vectors: Axes representing application characteristics, the value of which help us understand the application and what influences the design/constraints of solutions, tools & programming systems that can be used
  - What makes distributed applications hard to develop?
  - What makes distributed infrastructure hard to use

Developing Distributed Applications (3)
What makes a DA a DA?

- Vectors: Axes representing application characteristics, the values of which help us understand:
  - The application requirements, and
  - Design and Constraints of solutions, tools
- Application Vectors:
  - Executable Unit
  - Communication
  - Coordination
  - Execution Environment

Taxonomy of Distributed Applications (1)
How are DA developed?

- Distributing a *legacy* Application, or invoking simple functions
  - Focus mechanisms to support distributed execution
    - Science Gateways, Condor..
    - No-coupling or coordination between tasks
      - E.g., Replica Ensemble
- Distributed Applications with Multiple Components
  - *Naturally* decomposed; aggregate (coordination)
      - E.g. DDDAS, sensor-based application, DAG-based
  - Decompose a Large problem into sub-components
      - Degree-of-coupling (e.g freq/vol of comm)
      - Flexibility (scheduling, ordering)

Taxonomy of Distributed Applications (2) Coordination

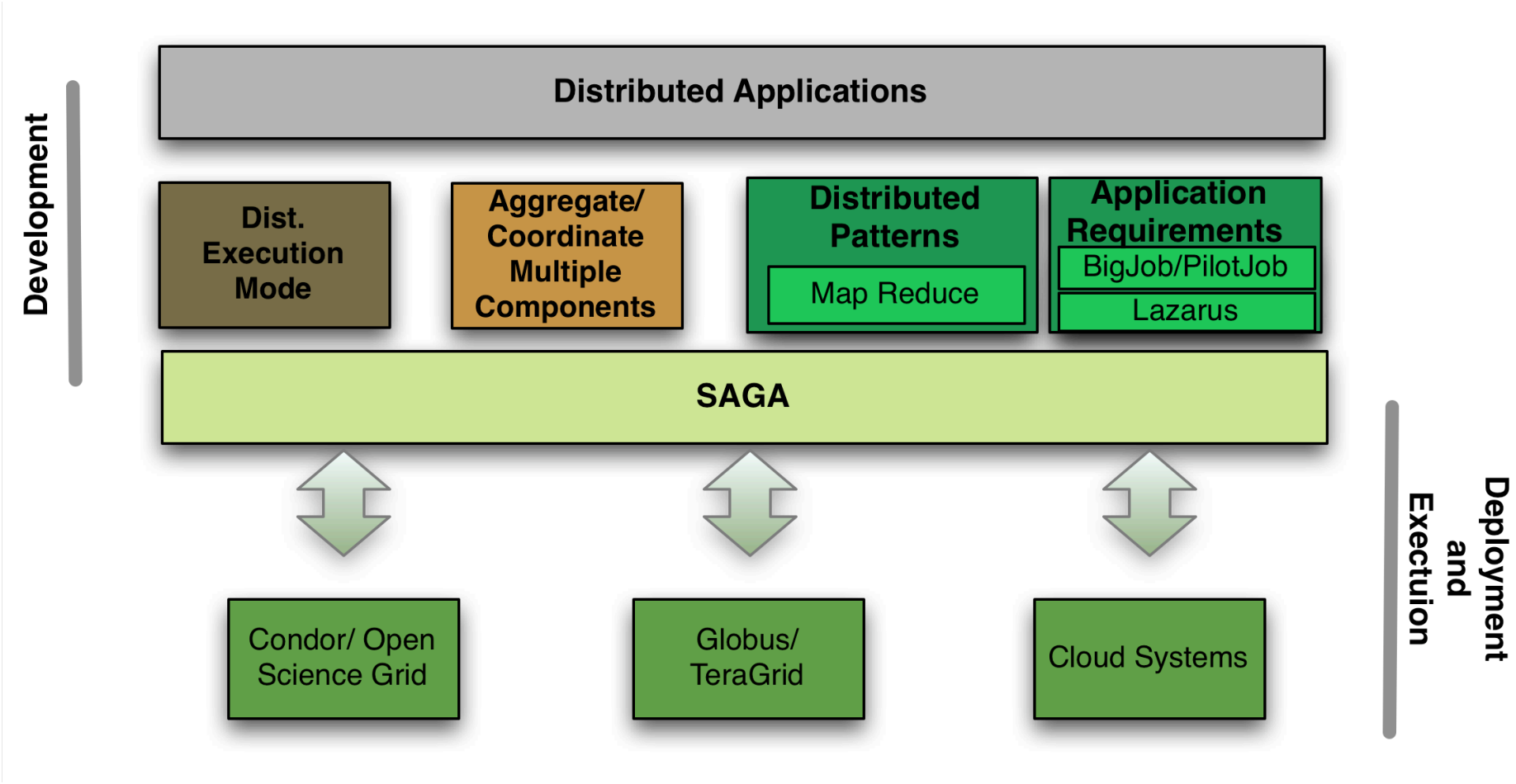
- Challenge: Coordination of multiple-components
  - Decompose a large problem into sub-problem
  - Designing *De Novo* Distributed Application
    - E.g. GridSAT (Wolski), Distributed Reasoning (Bal)
- Coupling components set coordination strategy
  - Coupling is high:
    - Low flexibility in placement, ordering, resource selection; or Communication (e.g. MPIg)
  - Coupling is low:
    - High flexibility in placement, ordering..

      E.g. (Identical?) Replica-Exchange

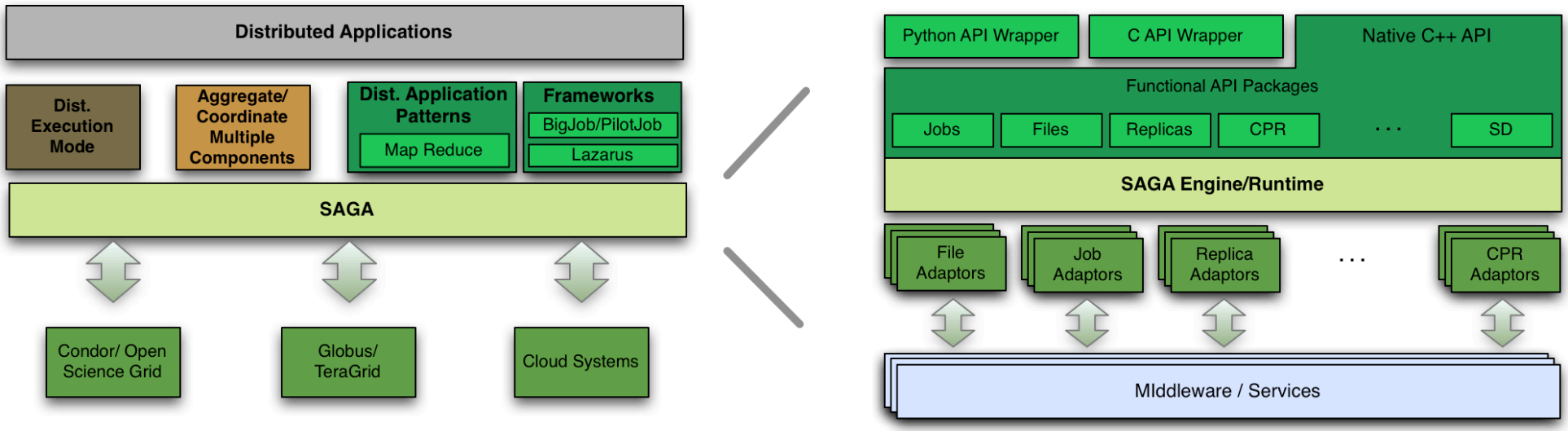
Frameworks: Capturing Application Requirements, Characteristics & Patterns

- Pattern: Commonly recurring modes of computation
- Abstraction: Mechanism to support patterns and application characteristics
  - Development, Deployment and Execution
- Frameworks:
  i. Support Patterns:
     - MapReduce, Master-Worker, Hierarchical Job-Submission
  ii. Provide the abstractions and support the requirements & characteristics

SAGA and Distributed Applications



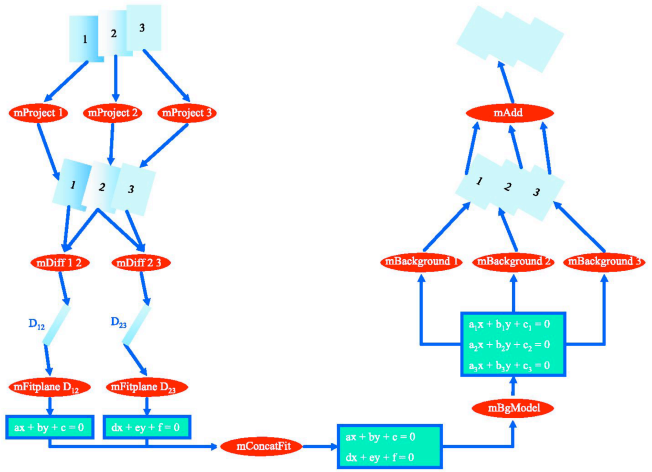
SAGA: Bridging the Gap between Infrastructure and Applications

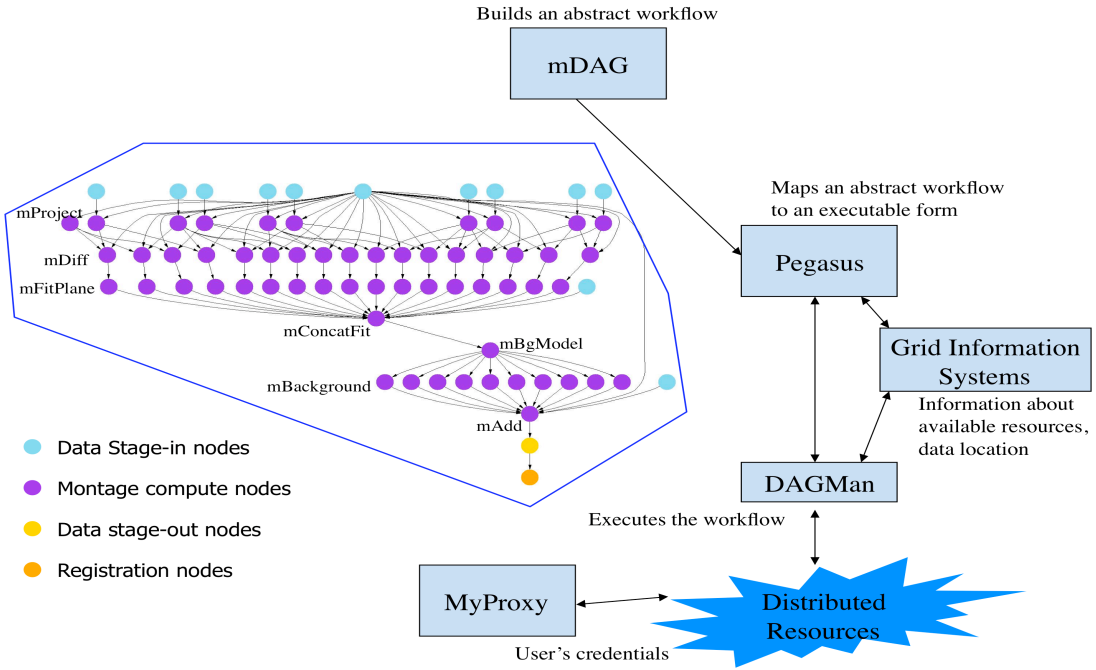


SAGA: Application Types

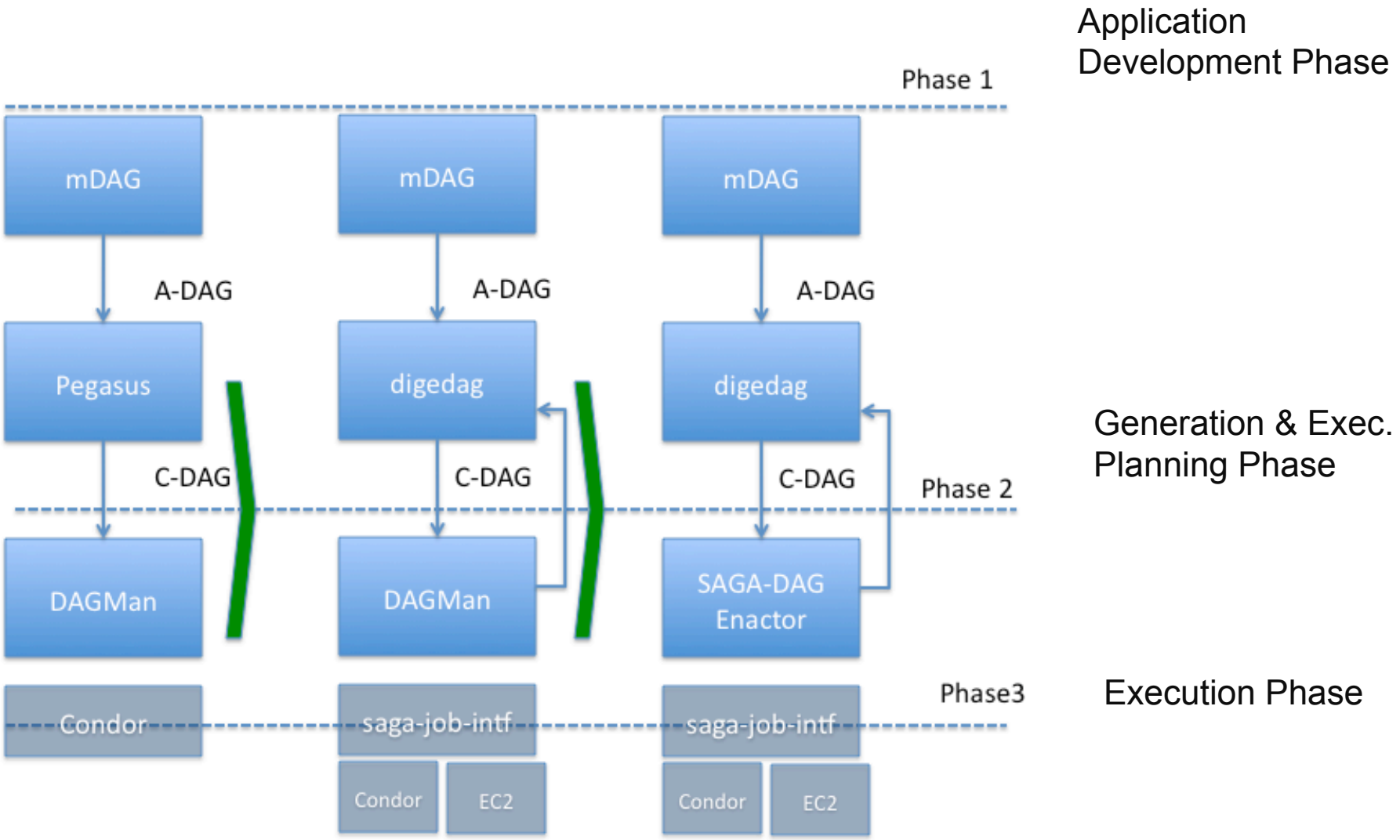
- Example of Distributed Execution Mode:
  - Implicitly Distributed
  - SAGA can be used to script 800 job submissions on the TeraGrid for GridChem
  - SAGA shell example/tutorial
- Example of Explicit Coordination and Distribution
  - Explicitly Distributed
  - EnKF-HM application in which application controls the work-load and determines where & when the execution of a task is to take place

Montage: DAG-based Workflow Application Exemplar





DAG based Workflow Applications

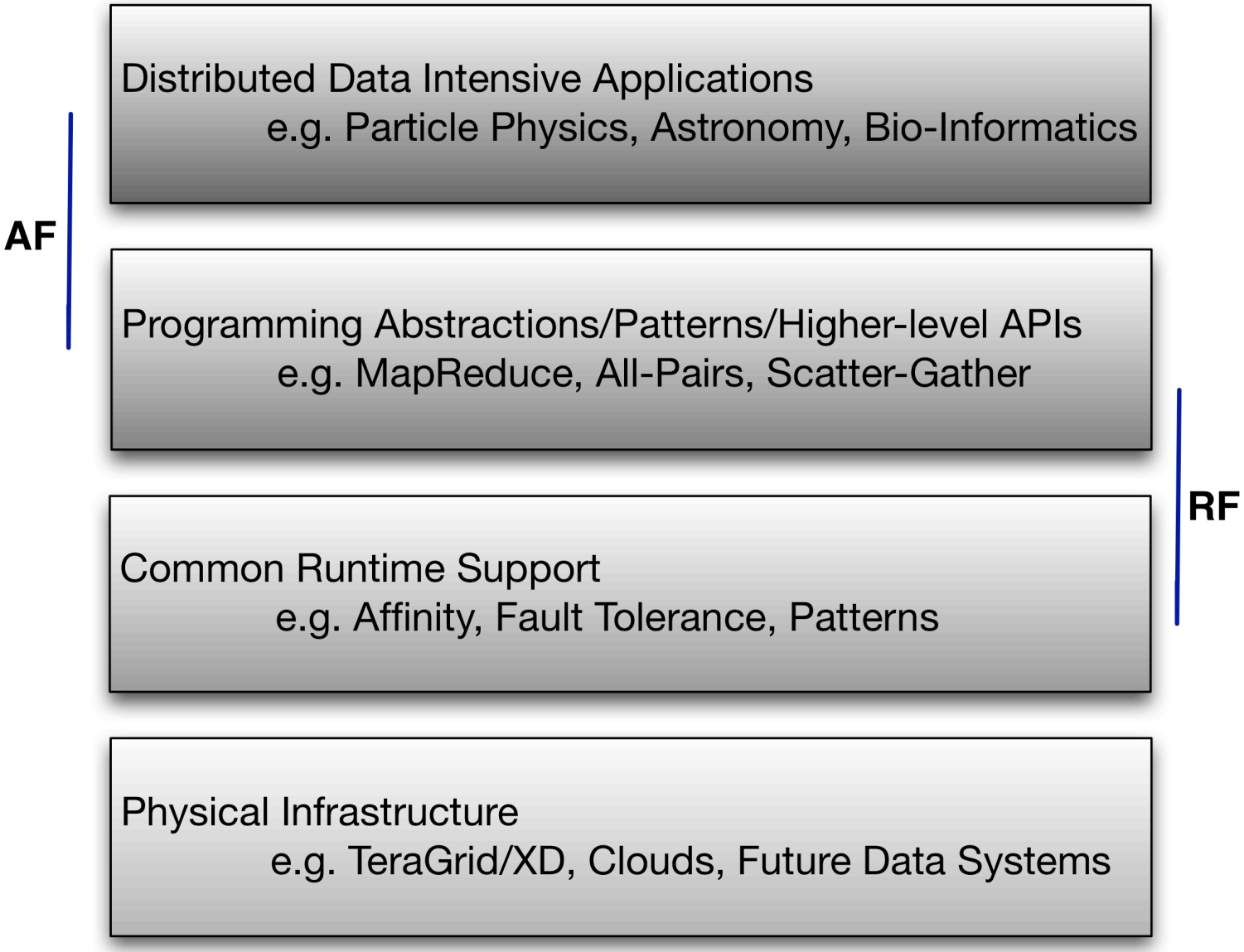


SAGA: Frameworks

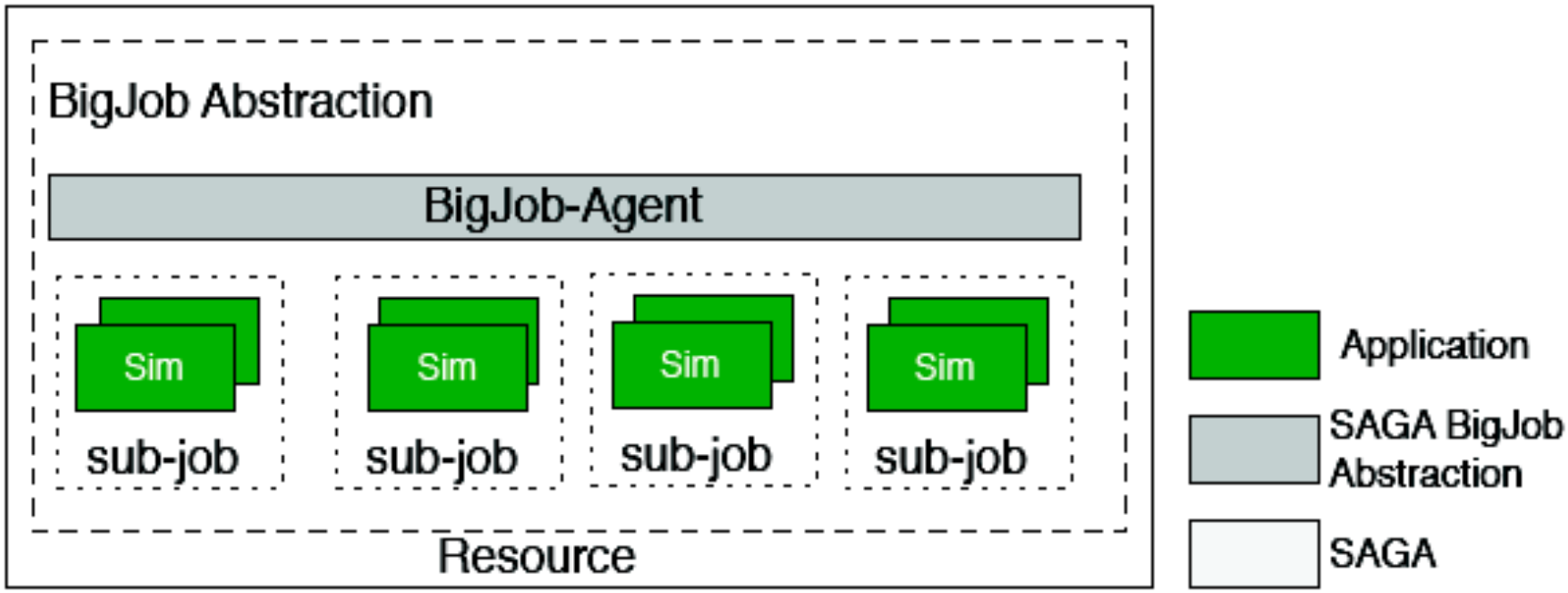
- Frameworks can either support a specific programming pattern or run-time pattern, or a commonly occuring application characteristic, i.e. provides an implementation of the abstraction

- Three types don’t have to be mutually exclusive… Can have a framework that helps in the explicit coordination

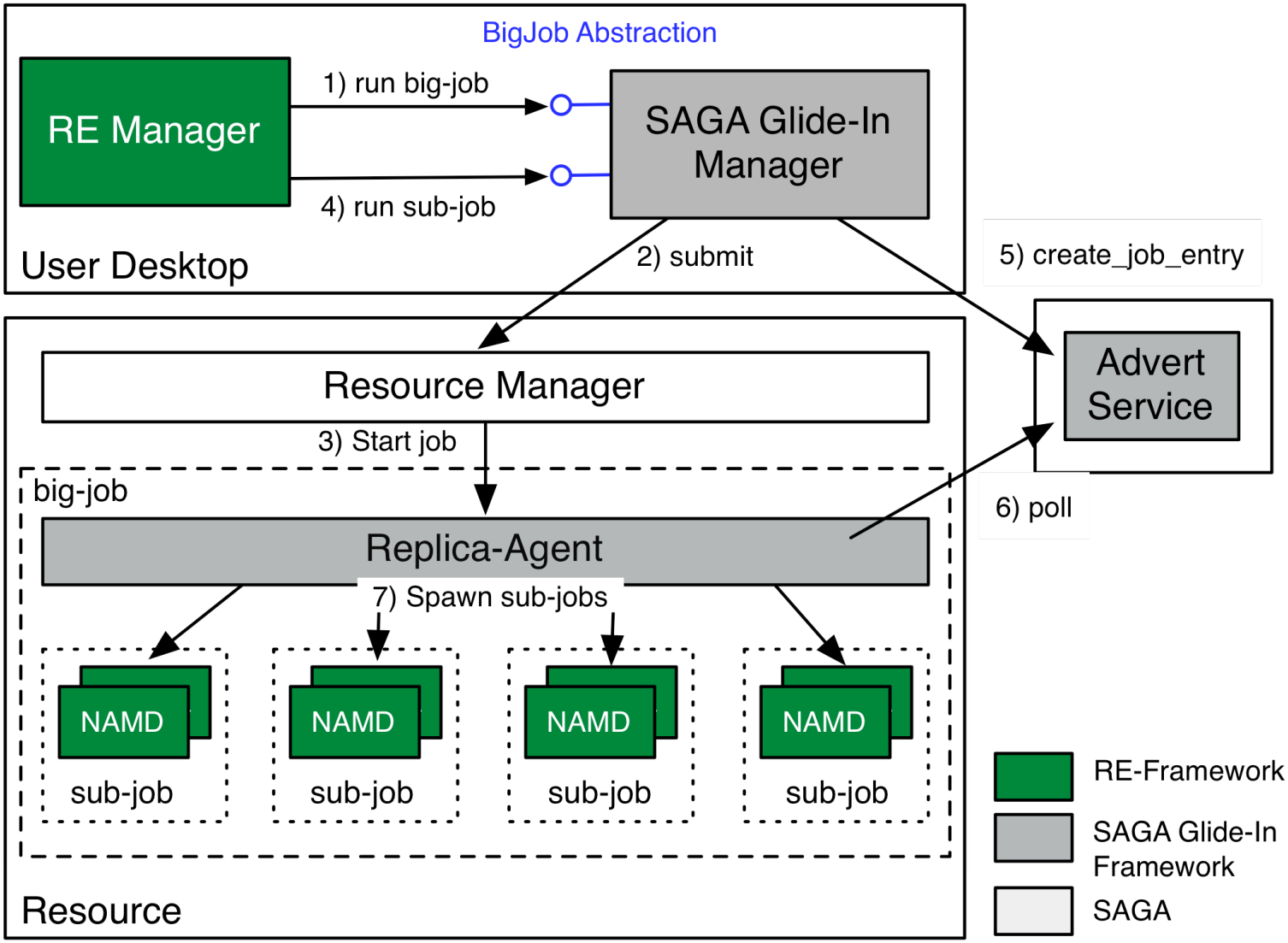
Logical ordering of SAGA-based frameworks



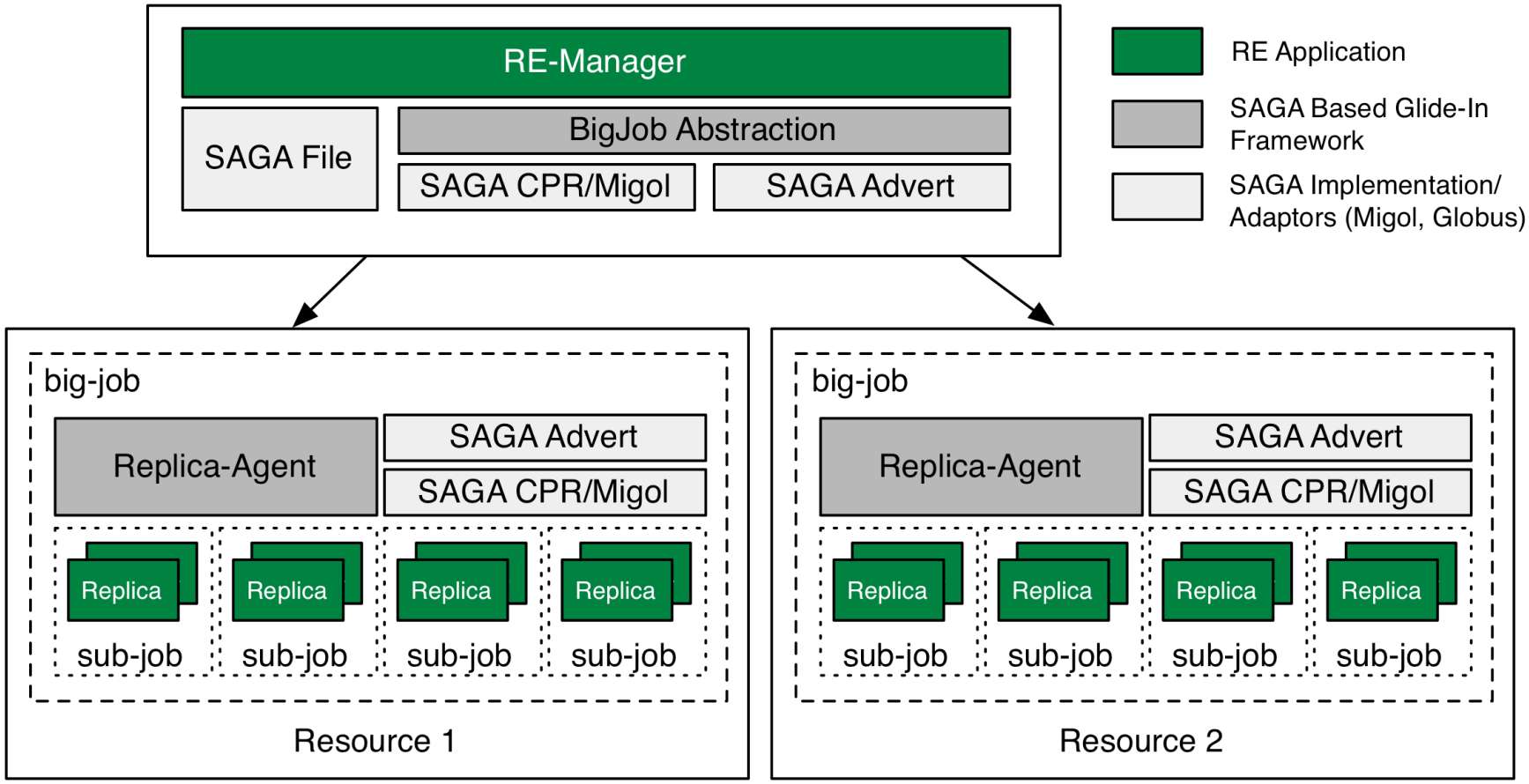
Abstractions for Distributed Computing (1)
BigJob: Container Task



Abstractions for Distributed Computing (2)
SAGA Pilot-Job (Glide-In)

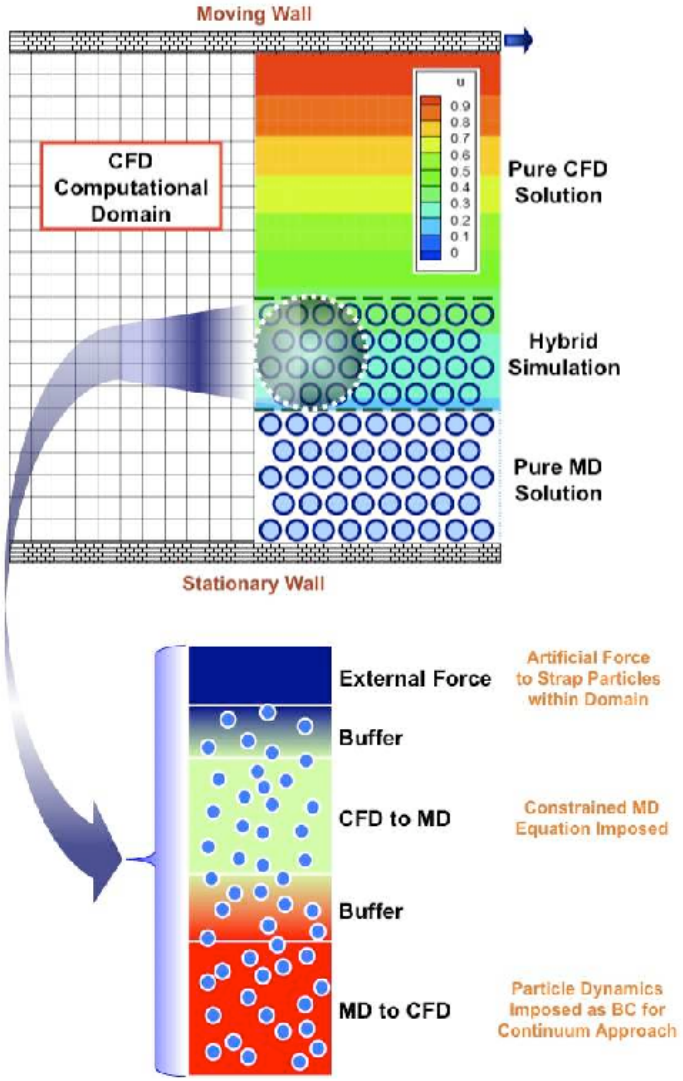


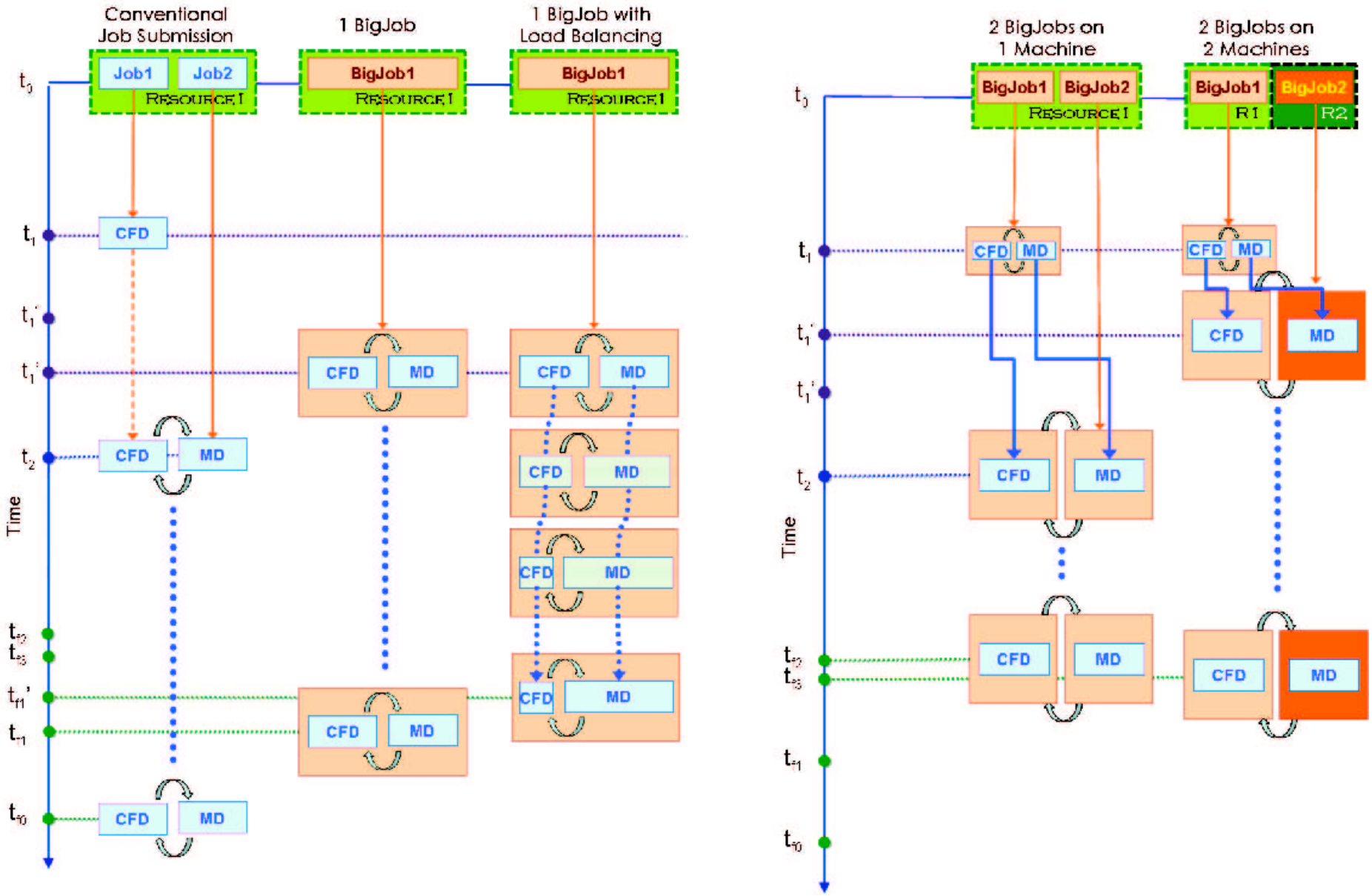
Coordinate Deployment & Scheduling of Multiple Pilot-Jobs



Multi-Physics Runtime Frameworks

- Coupled Multi-Physics require two distinct, but concurrent simulations
- Can co-scheduling be avoided?
  - Adaptive execution model the answer is yes
- Load-balancing required. Capability comes for free!
- First demonstrated multi-platform Pilot-Job:
  - TG(MD) – Condor (CFD)



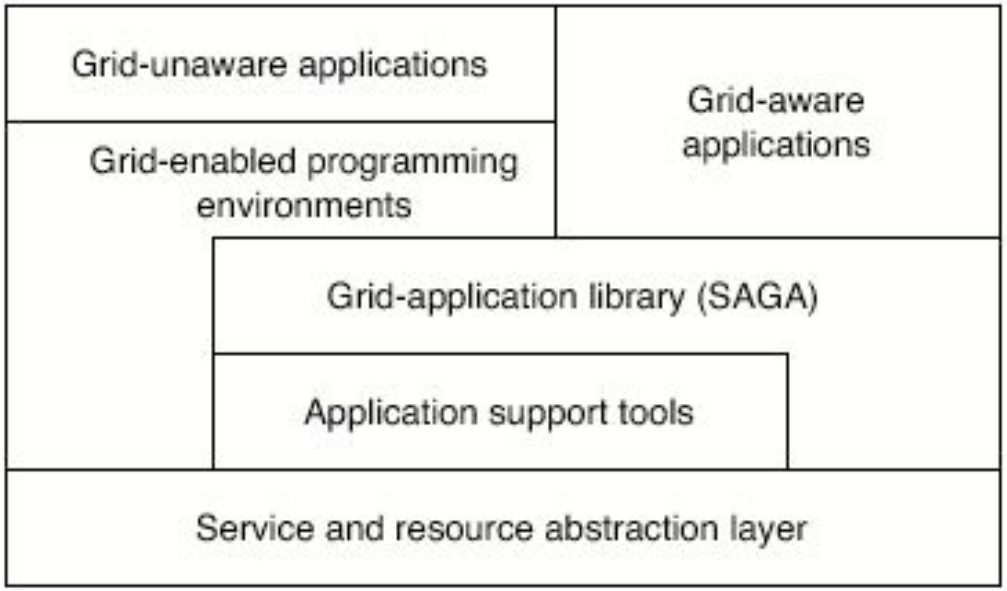

Conventional Job Submission
1 BigJob
1 BigJob with Load Balancing
Job1
Job2
BigJob1
BigJob1
RESOURCE I
RESOURCE I
RESOURCE I
CFD
MD
Time
2 BigJobs on 1 Machine
2 BigJobs on 2 Machines
BigJob1
BigJob2
BigJob1
BigJob2
R1
R2

Distributed: Implicit vs Explicit ?

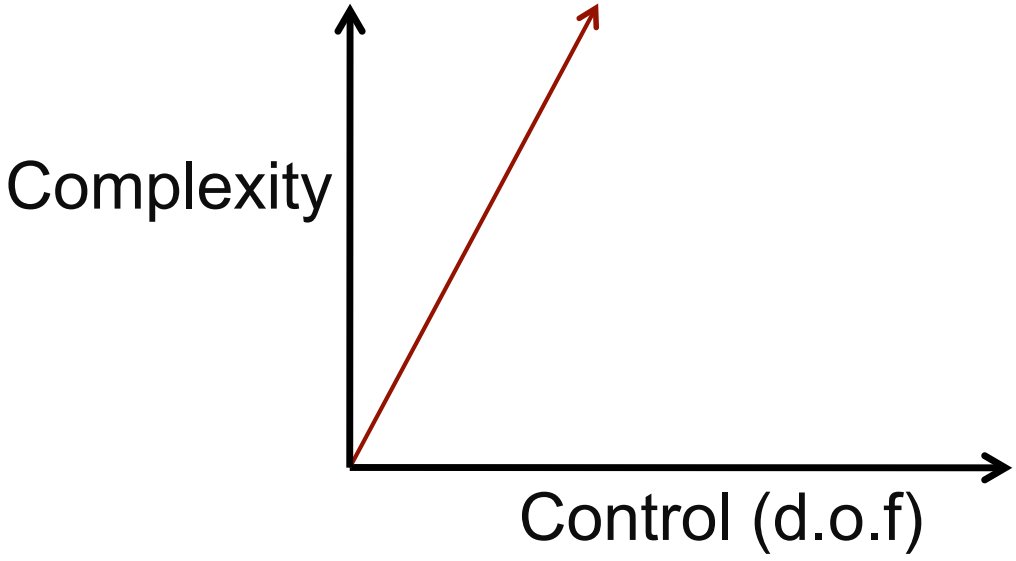
- Most applications/application classes can be either. Which approach (implicit vs explicit) is used depends:
  - How the application is used?
    - Need to control/marshall more than one resource?
  - Why distributed resources are being used?
  - How much can be kept out of the application?
    - Can't predict in advance?
    - Not obvious what to do, application-specific metric
- *Unless necessary, applications should not be explicitly distributed*

Application Class vs Type

Application Class (Cluster)	Properties (Values of Application Vectors)	Application Examples	General Concepts
Loosely Coupled Applications	Independent execution units; loosely coupled; data transfer through files.	Montage	Each execution unit is independently developed, and only exposes an I/O. Only an executable instance of each execution unit may be available. This application class involves a loose coupling between such executable instances.
Tightly Coupled Homogeneous Applications	Execution units interact via messaging or other mechanisms; strong dependency between units	Single Num. Rel. Simulation	
Tightly Coupled Heterogeneous Applications	Execution units interact via files; strong dependency between units; units may be implemented using different programming libraries	Molecular Dynamics Simulation (with separate components implementing different capabilities – such as force and velocity calculations, position of particles, etc)	Trivial generalization of MPI to WAN using MPICH-G2 and variants thereof, should also be considered here.
Loose Coupling of Tightly Coupled Applications	Many applications need to be used in conjunction with other applications; they may not necessarily have been designed for this *ab initio*. Alternatively, certain commonly used algorithms call for replacing single long-time running simulations with multiple shorter time-duration simulations – but with possible infrequent communication between the individual simulations.	Replica Exchange simulations for protein folding; multi-physics scientific applications	There are two-levels of communication; the first is internal to a single job/task (think monolithic MPI job), the second level is communication between the jobs/talks. The latter is less frequent and thereby more tolerant of latency and delays. Coordination of the many tasks/jobs is challenging.
Event-Oriented Applications	Execution units employ a publish/subscribe mechanism to coordinate; generally loose coupling between units	Distributed database search	



- No unique mapping between Application Class & Type.. .
- The same application can *often* be either explict or implicitly distributed



LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Outline (1)

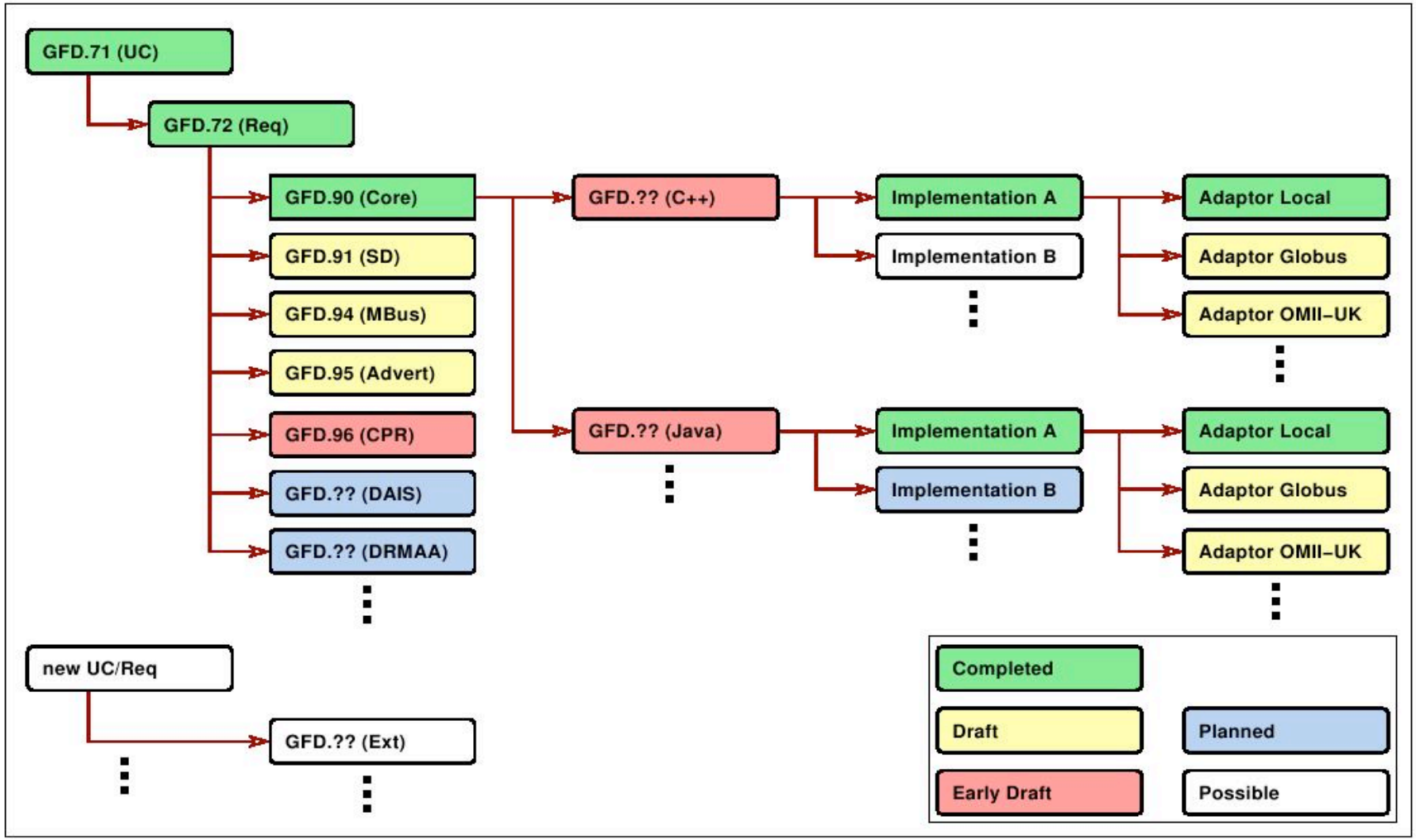
e-Science Institute

- Simple API for Grid Applications (SAGA)
- Understanding Distributed Applications (DA)
  - Challenges of DA ? Differ from HPC or || App?
  - Rough Taxonomy of Distributed Applications
  - Using SAGA to develop DA
    i. Distributed Execution Mode Legacy (Implicit)
    ii. Explicit coordination and distribution
    iii. Frameworks: support characteristics or patterns
- Understanding the SAGA Landscape
  - Interface/Specification
  - C++ Engine
  - Adaptors

CENTER FOR COMPUTATION & TECHNOLOGY

SAGA: The Standard Landscape

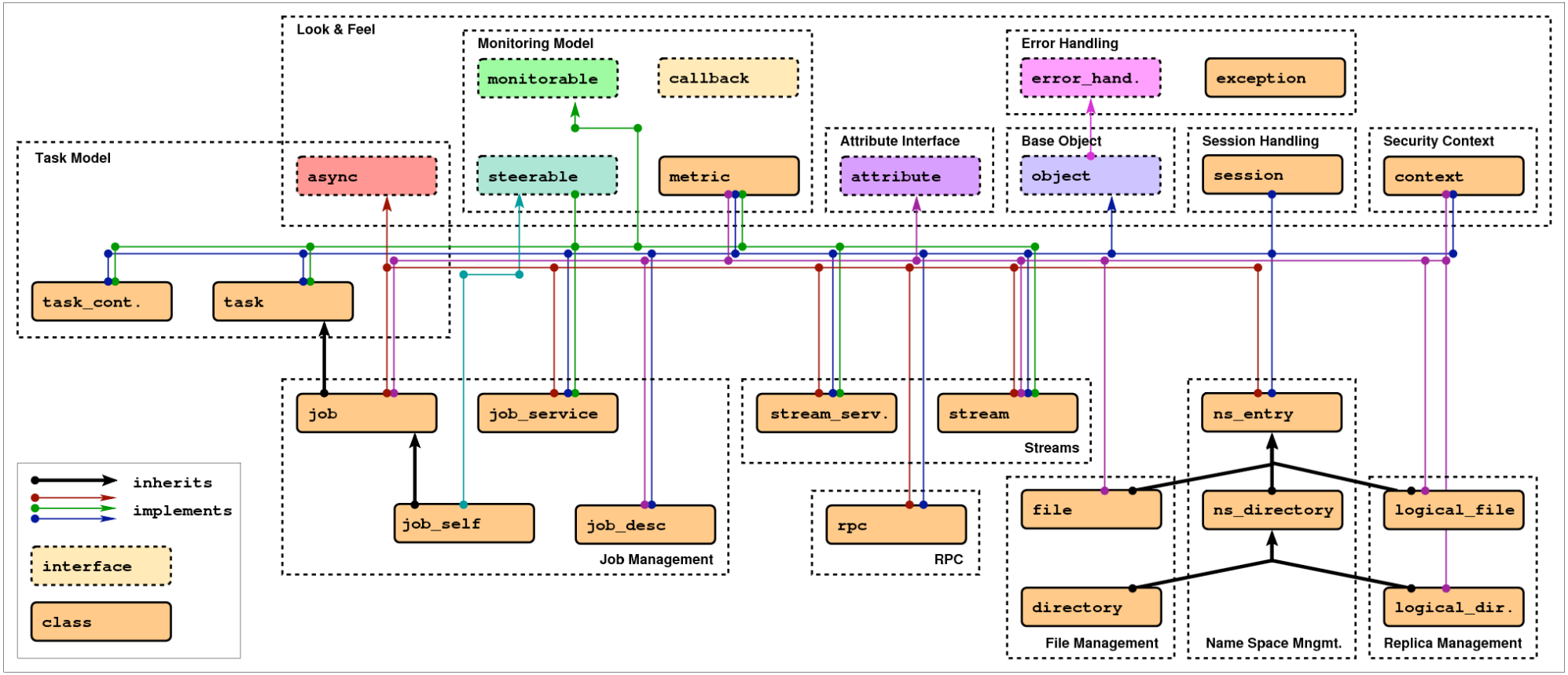
e-Science Institute



CENTER FOR COMPUTATION & TECHNOLOGY

SAGA v1.0 Hierarchy - Full Glory

e-Science Institute

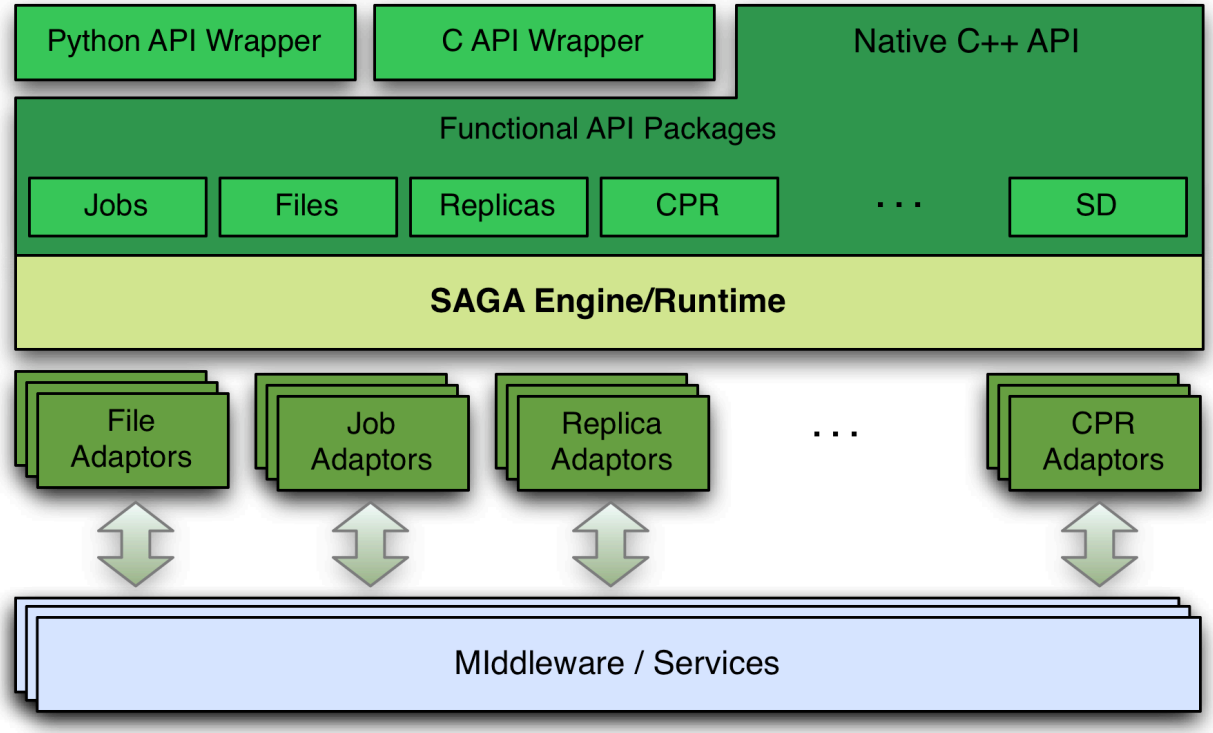


GridRPC - A rendering of *GridRPC*

CENTER FOR COMPUTATION & TECHNOLOGY

And in pictures..

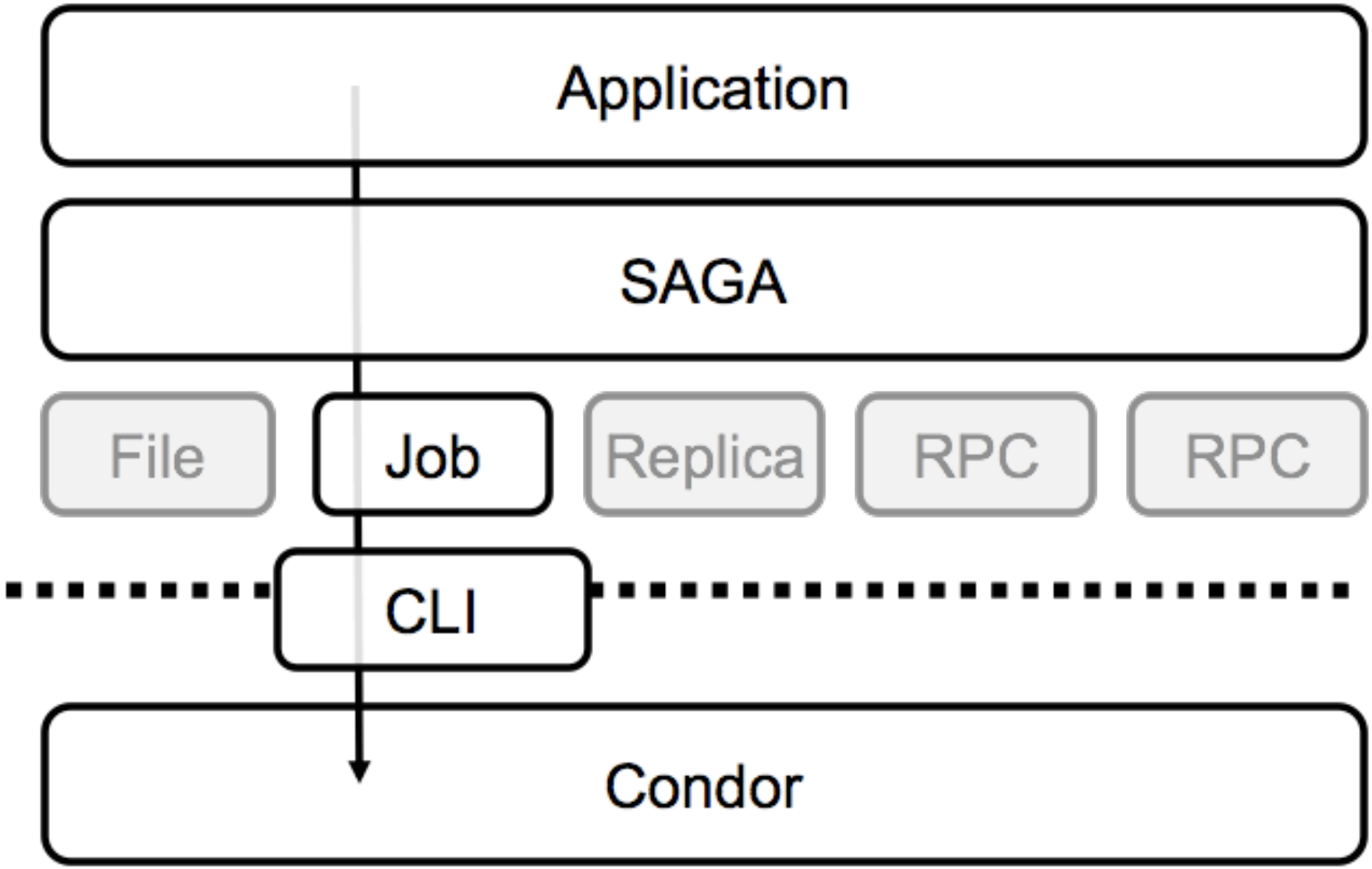
e-Science Institute



CENTER FOR COMPUTATION & TECHNOLOGY

SAGA-Condor: Campus Grids

e-Science Institute



SAGA Implementation: Requirements

- Non-trivial set of requirements:
  - Allow heterogenous middleware to co-exist
  - Cope with evolving grid environments; dyn resources
  - Future SAGA API extensions
  - Portable, syntactically and semantically platform independent; permit latency hiding mechanisms
  - Ease of deployment, configuration, multiple-language support, documentation etc.
  - Provide synchronous, asynchronous & task versions

  Portability, modularity, flexibility, adaptabilty, extensibility

SAGA Implementation: Extensibility

- Horizontal Extensibility – API Packages
  - Current packages:
    - file management, job management, remote procedure calls, replica management, data streaming
    - Steering, information services, checkpoint in pipeline
- Vertical Extensibility – Middleware Bindings
  - Different adaptors for different middleware
  - Set of ‘local’ adaptors
- Extensibility for Optimization and Features
  - Bulk optimization, modular design

CENTER FOR COMPUTATION & TECHNOLOGY

Is SAGA Simple?

e-Science Institute

- It depends: It is certainly not simple to implement!
  - Grids are complex and the complexity needs to be addressed somewhere, by someone!
  - Pain using the middleware goes into the SAGA engine and adaptors.
- But it is simple to use!!
  - Functional Packages (specific calls), Look & Feel
  - Somewhat like MPI - most users only need a very small subset of calls

CENTER FOR COMPUTATION & TECHNOLOGY

Implementations

- OGF Standard: Two independent implementations of a specification are required
- LSU: C++
  - V1.0 release Sep 15
  - Uptake by NAREGI/KEK, gLite (SD)
- Java
  - VU (Amsterdam):
    - Part of the OMII-UK Project
  - JSAGA
  - DEISA (DESHL)

CENTER FOR COMPUTATION & TECHNOLOGY

SAGA: C++ Status and Timelines

http://saga.cct.lsu.edu

e-Science Institute

- C++: Currently at v1.3.1 (July 2009)
  - Monthly release cycles for 2008; moving to Qtr cycle
  - Release V1.0 September 15 2008 (OGF24)
    - Both (OMII) JAVA and C++
      - Will mirror V1.0 of the specification
      - Cleaned up Build system for Linux, Mac & Win
  - Adaptors:
    - Globus RLS, GRAM, GridSAM, Gridftp (available)
    - ssh, libcurl (in progress)
    - Condor, LSF
    - CloudStore (KFS), HDFS
  - Extension packages:
    - Service Discovery (document in public comment)
    - Checkpoint and Recovery (Migol)
- Python bindings to the C++ available
- User Manual and Programmer's Guide (beta version available)

CENTER FOR COMPUTATION & TECHNOLOGY

Outline (2)

e-Science Institute

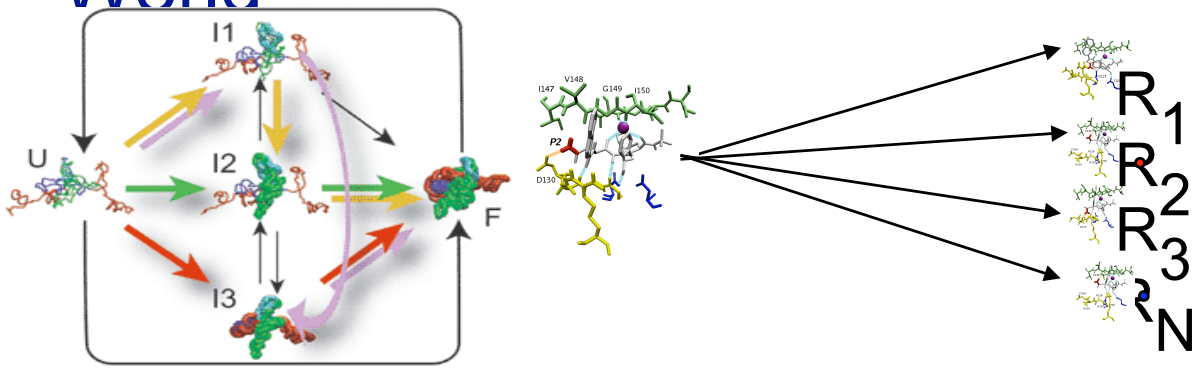
- Applications Developed Using SAGA:
  - Replica-Exchange, $C0_2$ Sequestration (EnKF)
  - MapReduce
    - Interoperabilty across Grids, Grids-Clouds
- Other SAGA Projects (Advantage of Standards)
  - Uptake by JSAGA and XtreemOS
  - Service Discovery (SD) and uptake gLite
  - DESHL (DEISA) -- Tool or Application?
- Session II: SAGA Hands-on
  - Using mini-examples, become familiar with the API
  - A few Application examples
  - Time to play with the code programmer's manual

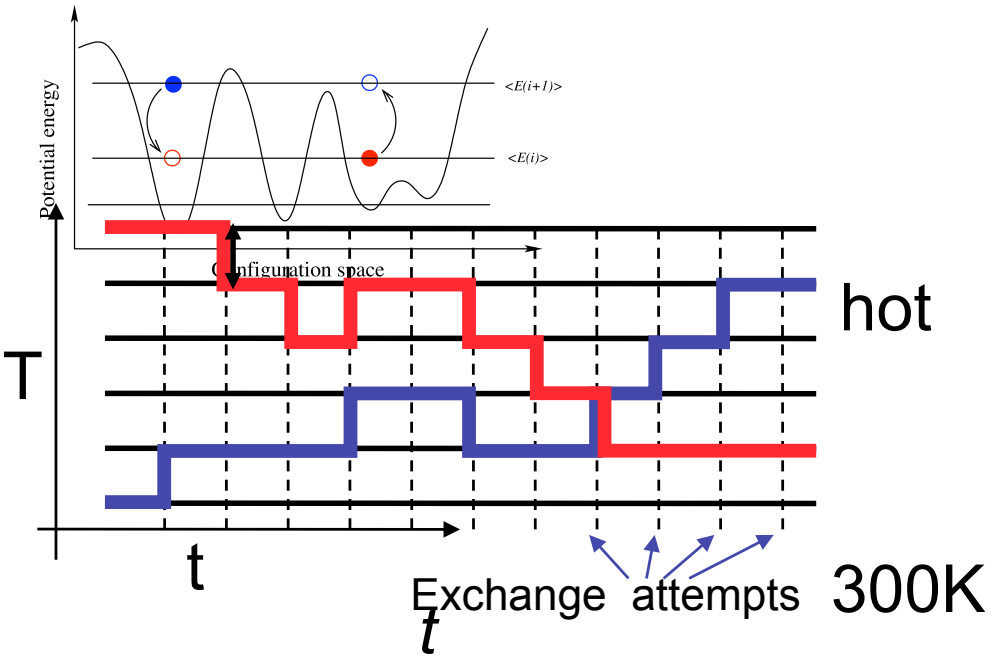
CENTER FOR COMPUTATION & TECHNOLOGY

Replica Exchange: Hello Distributed World

e-Science Institute

- Task Level Parallelism
  - Embarrassingly distributable!
  - Loosely coupled
- Create replicas of initial configuration
- Spawn 'N' replicas over different machine
- Run for time *t ;* Attempt configuration swap
- Run for further time t; Repeat till finish



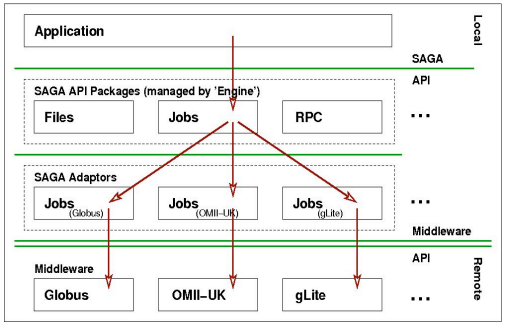


CENTER FOR COMPUTATION & TECHNOLOGY

RE: Programming Requirements

e-Science Institute

- RE can be implemented using following “primitives”
  - Read job description
    - # of processors, replicas, determine resources
  - Submit jobs (local and remote)
    - Move files, job launch
  - Checkpoint and re-launch simulations
    - Exchange, RPC (logic to swap or not)



- Implement above using “Grid primitives” provided by SAGA
- Separated “distributed” logic from “simulation” logic
  - Independent of underlying code/engine
  - Science kernel is independent of details of distributed resource mgmt
  - Need to coordinate resources integrated from Desktop to Supercomputers!!

Scale-Out: Dynamic Execution & Resource Aggregation

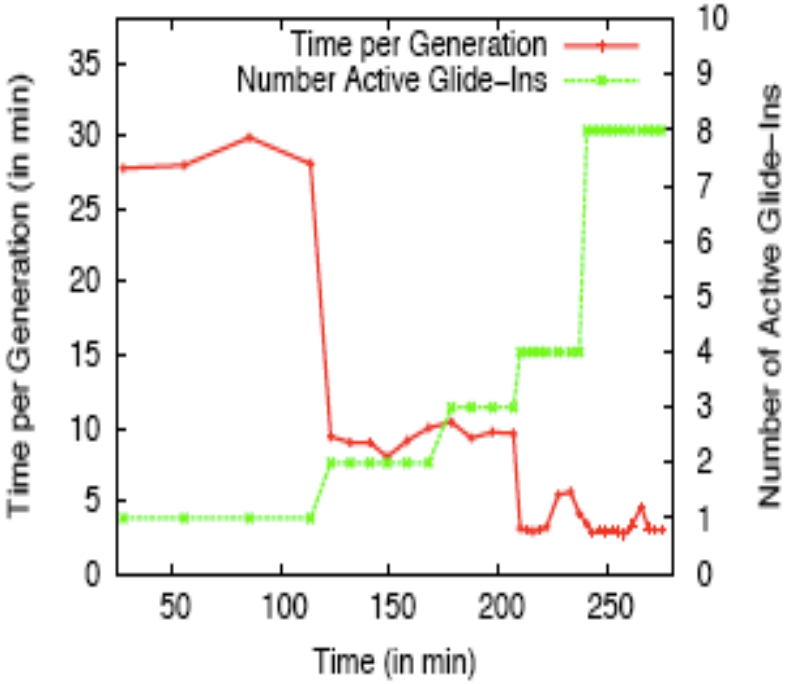


Figure 5: **The plots show the time-series of the average times between exchange attempts (in red and using the left-hand y axis) and the number of active Glide-Ins over a six-hour run on the TeraGrid.**

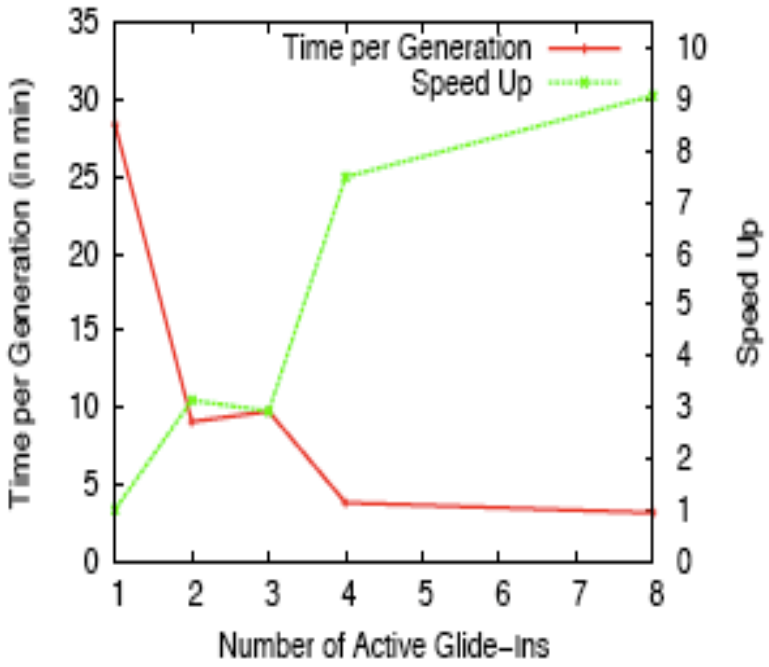


Figure 6: **The plot in red (using the left-hand y axis) illustrates how the average time between exchange attempts decreases as the number of Glide-Ins increases. The plot in green shows the speedup.**

Using Multiple Resources Performance Enhancements

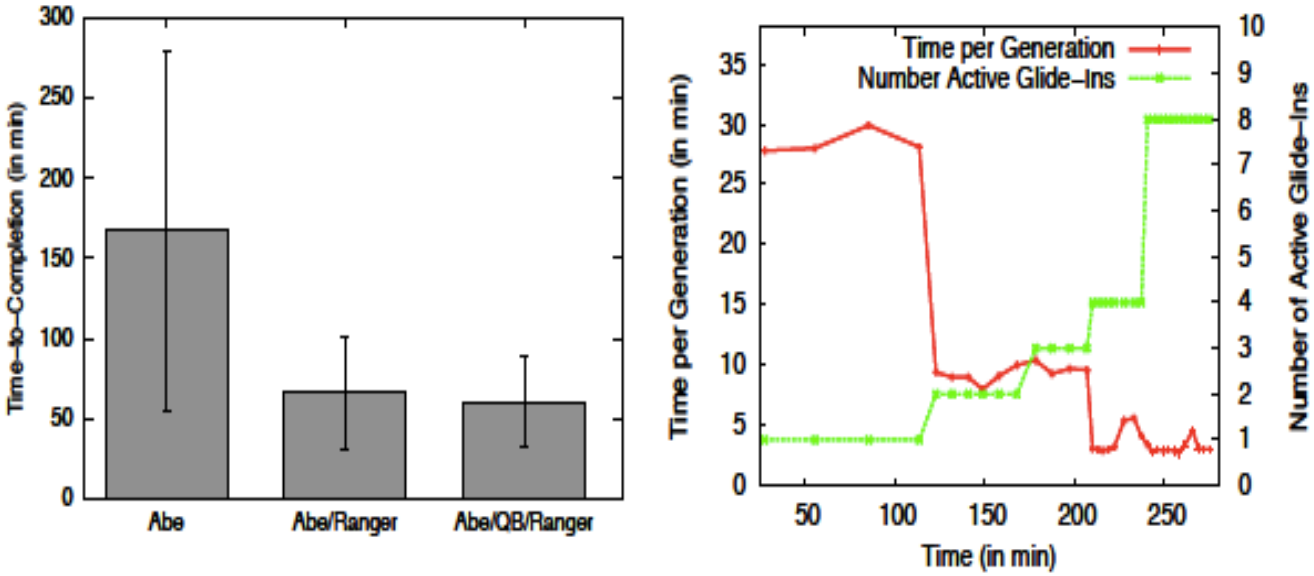
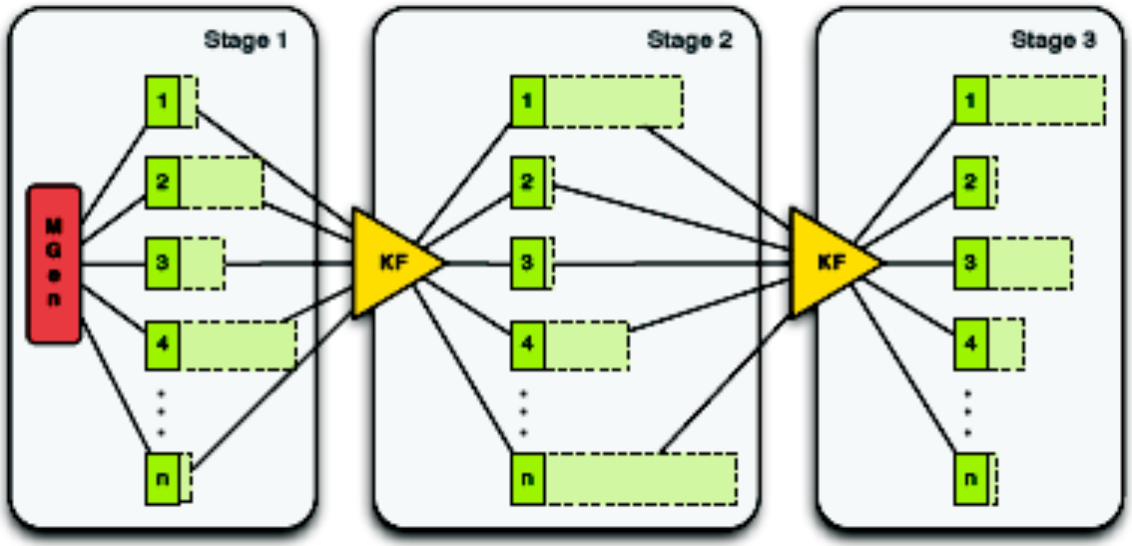


Figure 3: Performance data providing conclusive evidence that SAGA can be used to lower the time-to-solution, as the number of resources that can be used increases. SAGA provides the ability to use multiple resources in a simple and scalable fashion. The total number of computer-cycles used do not necessarily increase, i.e., time-to-completeness is not at the expense of efficiency. The lower figure contains plots which show the time-series of the average times between exchange attempts (upper line using the left-hand y axis) and the number of active Glide-Ins over a 6hr run.

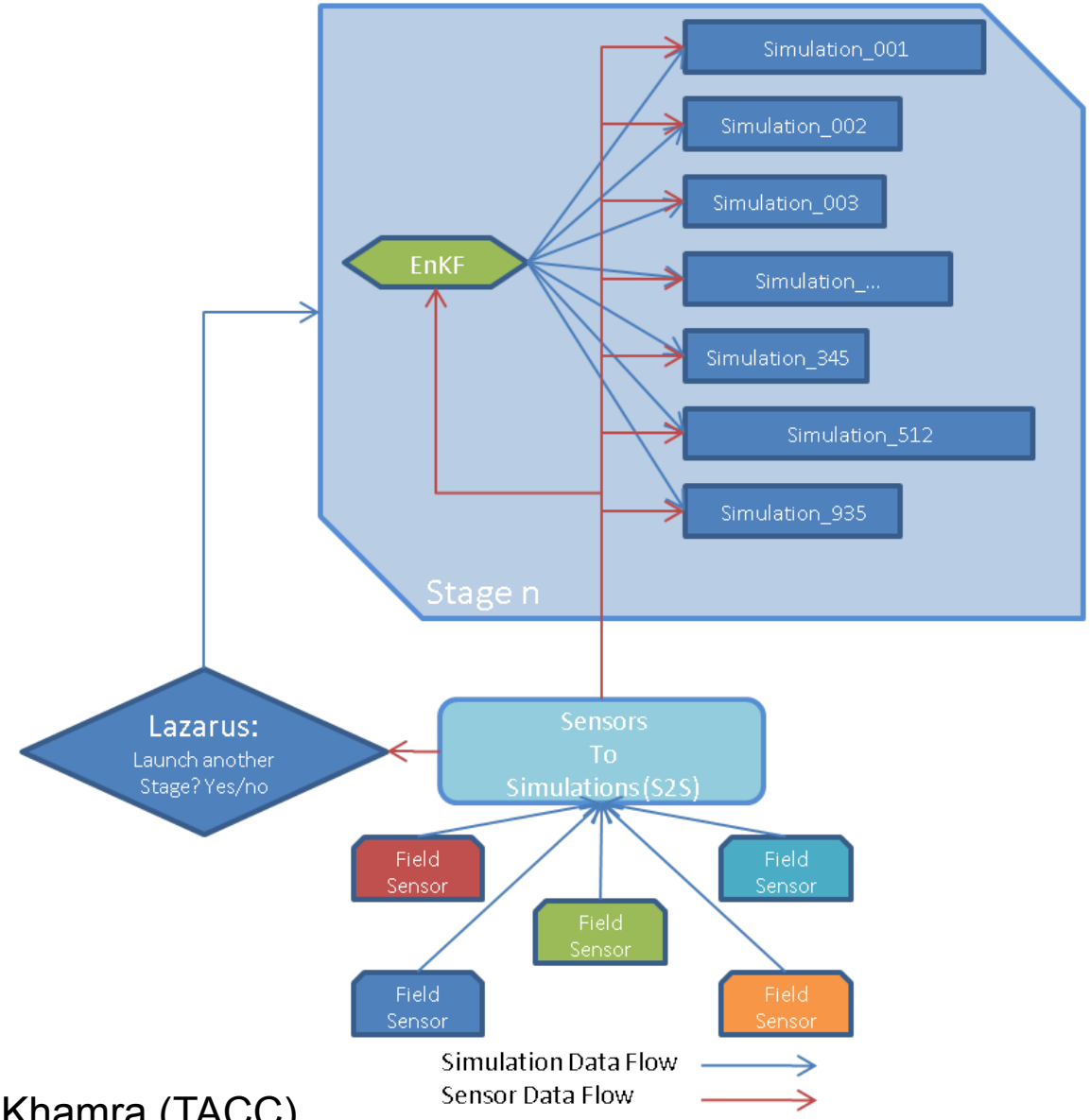
Ensemble Kalman Filters

Heterogeneous Sub-Tasks

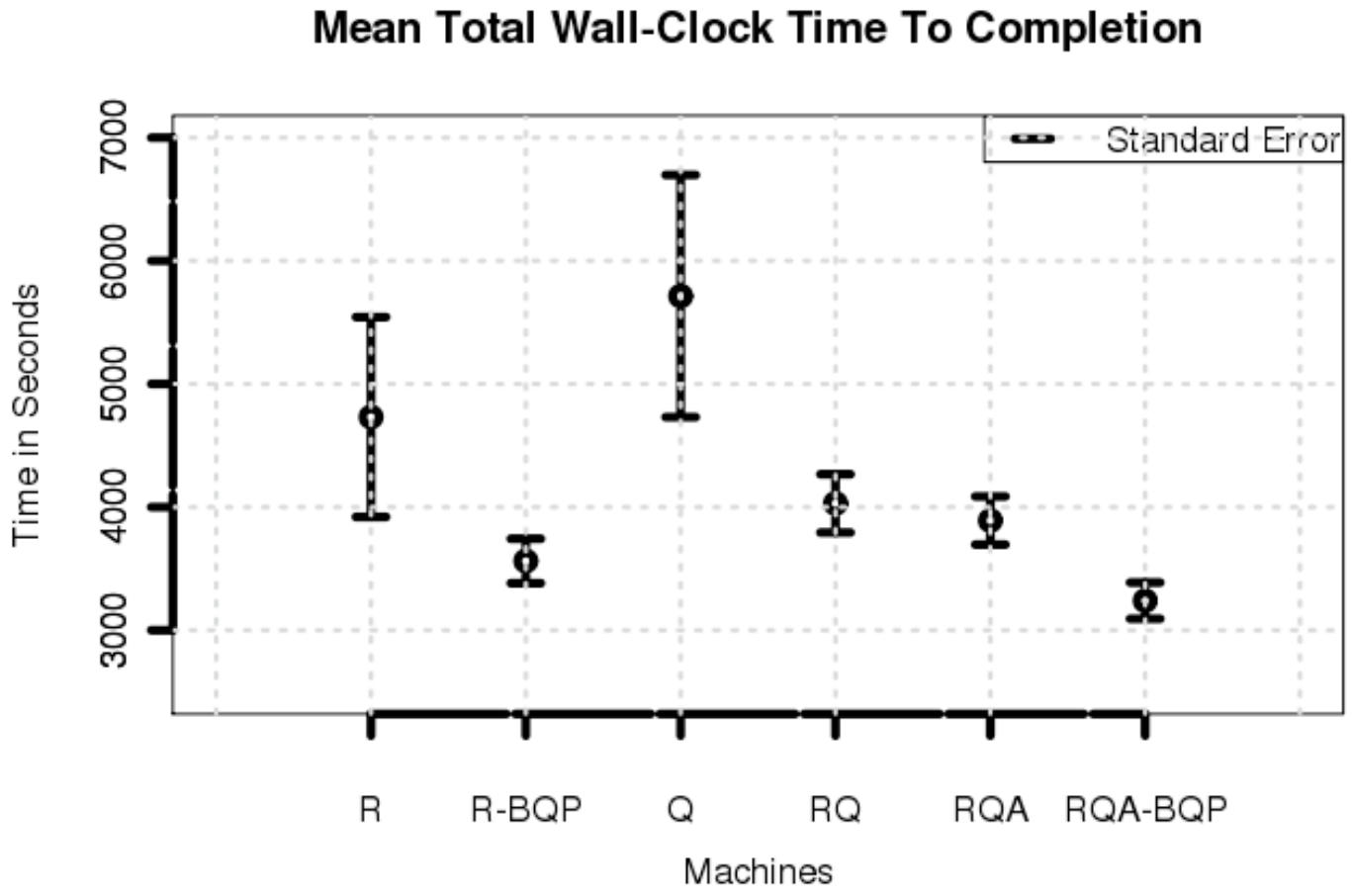
- Ensemble Kalman filters (EnKF), are recursive filters to handle large, noisy data; use the EnKF for history matching and reservoir characterization
- EnKF is a particularly interesting case of irregular, hard-to-predict run time characteristics:



EnKF C02 Sequestration: Data-Driven (Sensor) Distributed HPC



work with Yaakoub El-Khamra (TACC)



But Why does BQP Help?

Forward Vs Reverse Scheduling
Adv of Ensemble of Loosely-Coupled

- Distributed Applications: When formulated using Loosely-Coupled Tasks:
  - Application keeps work ready to be fired, when resource becomes available (ideal case)
  - Match resources to computational requirement (Forward)
    - Take a workload; find a set of resources
  - Find a set of resources (first-to-run); tune workload
    - Reverse Scheduling: Support for Dynamic Execution of Applications is underlying paradigm
    - Late binding to resource and optimal resource configuration (not just resource selection)
    - Unbalanced, irregular workload

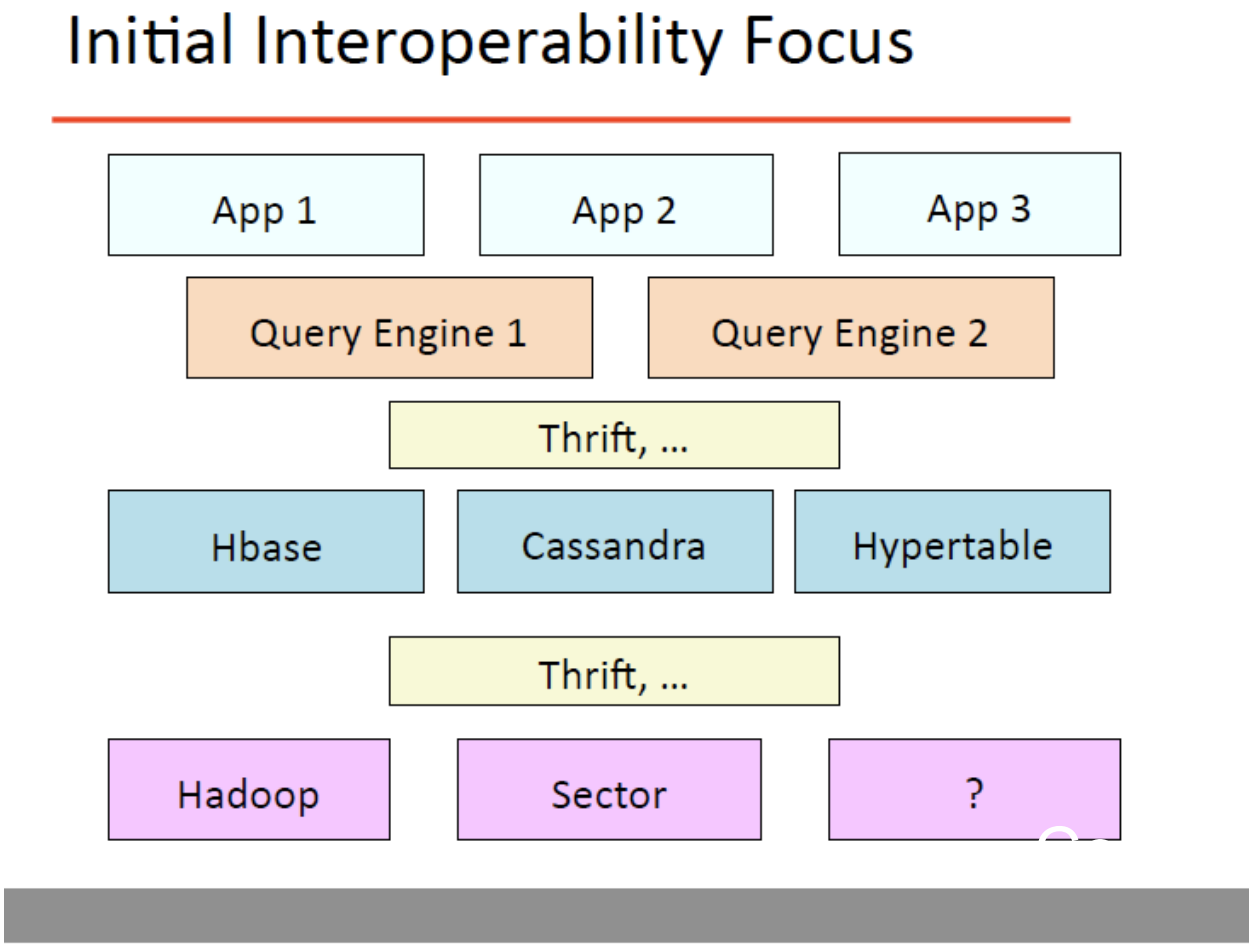
CENTER FOR COMPUTATION & TECHNOLOGY

SAGA for your Distributed Application?

e-Science Institute

- Should you develop using SAGA or not? (As always) it depends:
  - Advantages:
    - Simple yet powerful API, which supports many:
      - Applications types, but not all kinds of applications
      - Programming patterns, models (eg superscalar)
      - Usage modes (eg parameter sweep, task-farming..)
    - Application are Infrastructure independent
    - Provides extensibility (eg Condor, fault-tolerance)
    - Interfaces well with other software (eg BQP)
  - Disadvantages:
    - Overhead (trivial IMHO); “script it *vs* just do it”
    - Deployment of SAGA is currently non-trivial
      - sagalite and work with RP's to ease deployment
    - Stable & Simple interface, but restricted functional scope

Some Motivation for Interoperabilty (1)



Application-level Interoperability

Cloud-Cloud; Cloud-Grid

- Application-level (ALI) vs. System-level Interoperability (SLI)
- Infrastructure Independence is Pre-requisite for ALI
  - Programming Model should be Infrastructure & SLA agnostic:
    - Same application, priced differently, for same performance
    - Same application, priced same, for different performance
  - Availability zone
  - VM motion
  - Data-transfer cost vs. no cost
- Grids AND Clouds:
  - Hybrid & Heterogeneous workload: data-compute affinity differ
  - Complex data-flow dependency: need runtime determination

Power of Google's MapReduce?

- MapReduce an important development but..
  - Currently tied to infrastructure
    - MapReduce + BigTable + GFS or
    - MapReduce + Hbase + HDFS (Yahoo)
  - Limited number of scientific applications that use this simple (though powerful) pattern
    - Other patterns?

Is it possible to provide the power of MapReduce and other patterns in an infrastructure independent

SAGA: Role of Adaptors
Use over different infrastructure

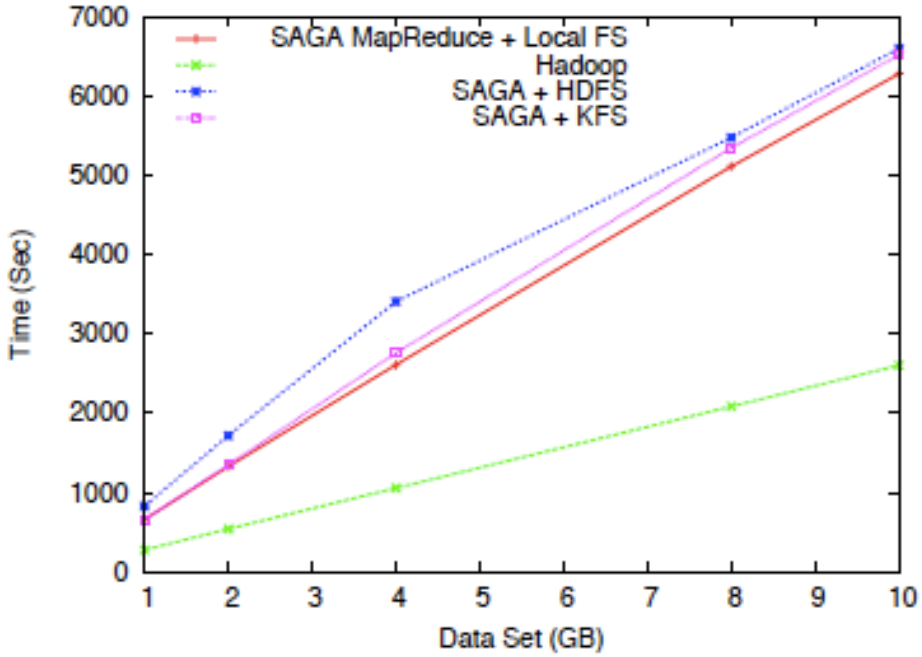


Fig. 5: T_c for SAGA-MapReduce using one worker (local to the master) for different configurations. The label "Hadoop" represents Yahoo's MapReduce implementation; T_c for Hadoop is without chunking, which takes several hundred sec for larger data-sets. The "SAGA MapReduce + Local FS" corresponds to the use of the local FS on Linux clusters, while the label "SAGA + HDFS" corresponds to the use of HDFS on the clusters. Due to simplicity, of the Local FS, its performance beats distributed FS when used in local mode.

Controlling Relative Compute-Data Placement

Configuration		data size	work-load/worker	T_c
compute	data	(GB)	(GB/W)	(sec)
local	local-FS	1	0.1	466
distributed	local-FS	1	0.1	320
distributed	DFS	1	0.1	273.55
local	local-FS	2	0.25	673
distributed	local-FS	2	0.25	493
distributed	DFS	2	0.25	466
local	local-FS	4	0.5	1083
distributed	local-FS	4	0.5	912
distributed	**DFS**	**4**	**0.5**	**848**

TABLE I: Table showing T_c for different configurations of compute and data. The two compute configurations correspond to the situation where all workers are either placed locally or workers are distributed across two different resources. The data configurations arise when using a single local FS or a distributed FS (KFS) with 2 data-servers. It is evident from performance figures that an optimal value arises when distributing both data and compute.

SAGA Job Launch via GRAM gatekeeper

```
{ // contact a GRAM gatekeeper                                          1
  saga::job::service     js;                                           2
  saga::job::description jd;                                           3
  jd.set_attribute (``Executable'', ``/tmp/my_prog'');                 4
  // translate job description to RSL                                  5
  // submit RSL to gatekeeper, and obtain job handle                   6
  saga:job::job j = js.create_job (jd);                                7
  j.run ():                                                            8
  // watch handle until job is finished                                9
  j.wait ();                                                           10
 } // break contact to GRAM                                           11
```

SAGA create a VM instance on a Cloud

```
{// create a VM instance on Eucalyptus/Nimbus/EC2                     1
 saga::job::service     js;                                           2
 saga::job::description jd;                                           3
 jd.set_attribute (``Executable'', ``/tmp/my_prog'');                 4
 // translate job description to ssh command                          5
 // run the ssh command on the VM                                     6
 saga:job::job j = js.create_job (jd);                                7
 j.run ():                                                            8
 // watch command until done                                          9
 j.wait ();                                                           10
} // shut down VM instance                                            11
```

Interoperability

Number-of-Workers			Size	T_s	T_{spawn}	$T_s - T_{spawn}$
TG	AWS	Eucalyptus	(MB)	(sec)	(sec)	(sec)
-	1	1	10	5.3	3.8	1.5
-	2	2	10	10.7	8.8	1.9
-	1	1	100	6.7	3.8	2.9
-	2	2	100	10.3	7.3	3.0
1	-	1	10	4.7	3.3	1.4
1	-	1	100	6.4	3.4	3.0
2	2	-	10	7.4	5.9	1.5
3	3	-	10	11.6	10.3	1.6
4	4	-	10	13.7	11.6	2.1
5	5	-	10	33.2	29.4	3.8
10	10	-	10	32.2	28.8	2.4

TABLE II: Performance data for different configurations of worker placements on TeraGrid, Eucalyptus-Cloud and EC2. The first set of data establishes Cloud-Cloud interoperability. The second set (rows 5- 11) represent interoperability between Grids-Clouds (EC2). The experimental conditions and measurements are similar to Table 1.

SAGA: Extension to Clouds

- Is the API perfect, that nothing really changes?
- Minor change:
  - Grids: Job_service() is not coupled to app lifetime
  - Clouds: Job_service() is coupled to the VM lifetime
    - When job_service() terminates, what happens to VM?
    - When VM terminates, what happens to job_service() ?
    - Changes to the SAGA Resource Discovery and Management Package will address these shortcomings

Grids vs Clouds?

- Its not Grids vs. Clouds:
  - *"…..such a good job of understanding parallel programming in the 90's … very well prepared for multi-core.." (Ken Kennedy)*
    - Hard problems in distributed applications & computing, e.g., Coordination of dist. Data & computation
    - These remain, even if we change names..
  - Its about (scientific) Applications: HLA & Support for Usage modes -- then Clouds vs. Grids is *less critical*
  - Interop of SAGA-MapReduce shows how – once correct abstractions and interfaces are in place -- much of work in easily translates to managing different back-end (adaptors)

Providing Fundamental Distributed Functionality for Applications

- Interop of SAGA-MapReduce shows how – once correct abstractions and interfaces are in place -- much of work in easily translates to managing different back-end (adaptors)

- Its not Grids vs Clouds: How can scientific applications be developed to utilize a broad range of DS, without vendor lock-in, or disruption, yet with the flexibility and performance that scientific applications demand?

- *SAGA is the first comprehensive application programming system to attempt to address these issues.*

Distributed Application Concerns

- Interoperability: Ability to work across multiple distributed resources
- Scale-Out: The ability to utilize multiple distributed resources concurrently
- Extensibility: Support new patterns/abstractions, different programming systems and Infrastructure
- Adaptivity: Response to fluctuations in dynamic resource and availability of dynamic data
- Simplicity: Accommodate above distributed concerns at different levels *easily…*

CENTER FOR COMPUTATION & TECHNOLOGY

SAGA: Other Projects

e-Science Institute

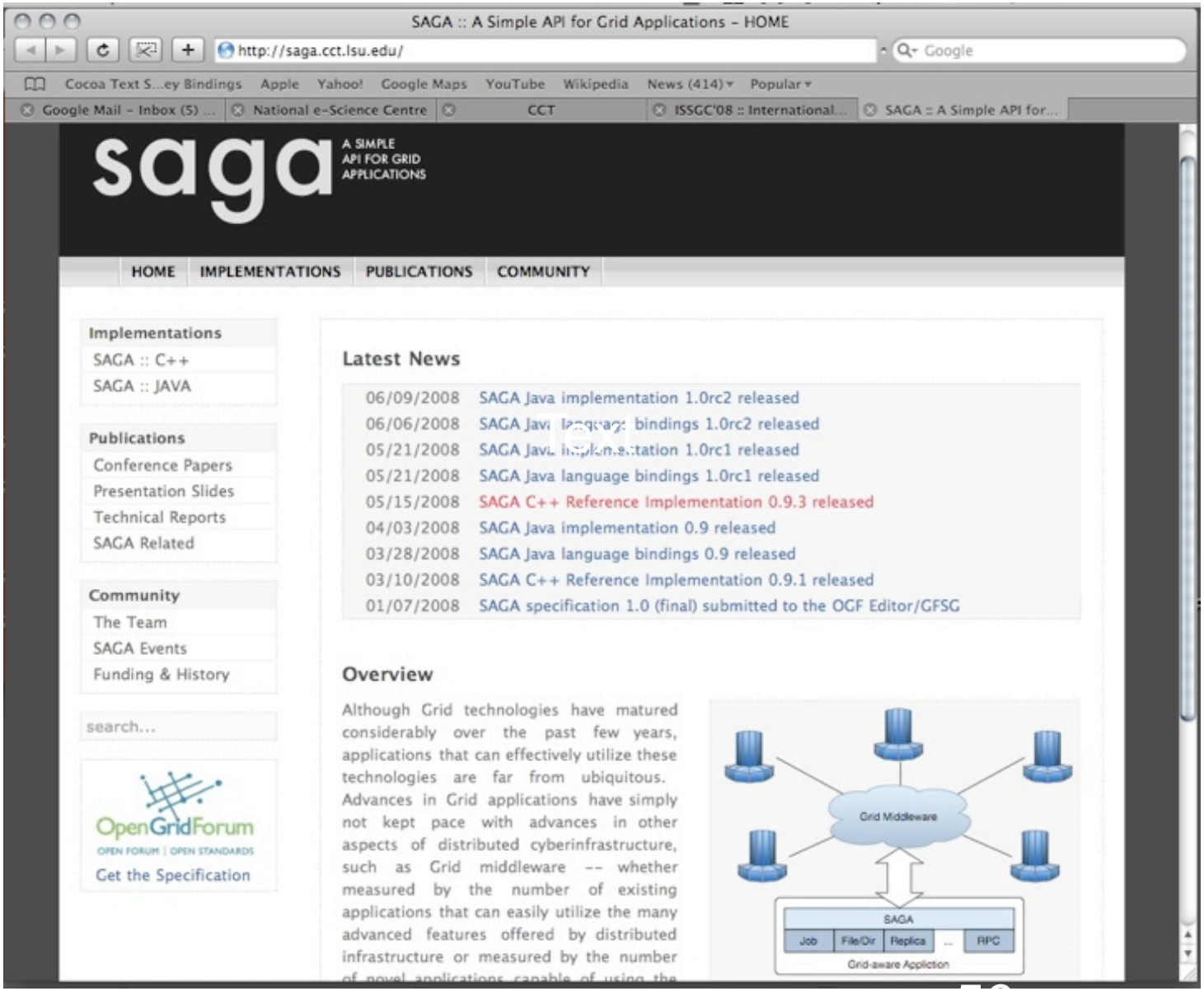
- XtreemOS
- DEISA: Java library
  - JRA7 DEISA (UNICORE) files and jobs
- JSAGA from IN2P3 (Lyon)
  - http://grid.in2p3.fr/jsaga/index.html
- SD Specification
  - With gLite adaptors
- NAREGI, KEK

Advantage of Standards

CENTER FOR COMPUTATION & TECHNOLOGY

http://saga.cct.lsu.edu

e-Science Institute



Acknowledgements

Funding Agencies:

UK EPSRC: DPA, OMII-UK , OMII-UK PAL

US NSF: Cybertools, HPCOPS (TeraGrid), GridChem

NIH: LaCATS, INBRE-LBRN

CCT Internal Resources

People:

SAGA D&D: Hartmut Kaiser, Ole Weidner, Andre Merzky, Joohyun Kim, Lukasz Lacinski, João Abecasis, Chris Miceli, Bety Rodriguez-Milla

Special Users: Andre Luckow, Yaakoub el-Khamra, Kate Stamou, Cybertools (Abhinav Thota, Jeff, N. Kim), Owain Kenway

Google SoC: Michael Miceli, Saurabh Sehgal, Miklos Erdelyi

Collaborators and Contributors: Steve Fisher & Group, Thilo Kielman & Co, Sylvain Renaud (JSAGA), Go Iwai & Yoshiyuki Watase (KEK)

Some Relevant Links

- http://saga.cct.lsu.edu
- http://www.cct.lsu.edu/~sjha/select_publications
- http://saga.cct.lsu.edu/projects

  (out of date; to be updated)
- TeraGrid-DEISA Interoperabilty Project
  - http://www.teragridforum.org/mediawiki/index.php?title=LONI-TeraGrid-DEISA_Interoperabilty_Project