Google Summer of Code 2016

Project Milestones

Project Milestone 3: Due February 29th

- In this milestone, you will provide a simple user interface that will allow users to submit jobs to Karst. All gateways should support the same application: LAMMPS.
- Can be either Web or GUI interface.
  - Allow the user to log in, run and monitor multiple LAMMPS jobs (following https://kb.iu.edu/d/beus (Links to an external site.)).
  - Users should be able specify the number of nodes, wall time, input file.
  - Users should be able to download outputs.
- Submitting the Assignment
  - Make a named branch of your master branch (milestone-3).
  - Put all instructions in README-MILESTONE3.md. You could also use GitHub's wiki feature.
  - Submit the assignment in Canvas when you are ready. Include pointers to the appropriate GitHub branch and the documentation.
- Simplifying assumptions:
  - Do not attempt to maintain the user identity from UI server to backend. It is OK to run jobs under a common account. Do not attempt to implement sophisticated user identity management, single sign-on, etc.
  - TLS security between UI server and Application server is enough for the milestone. Don't try to implement OAuth.

Project Milestone 4: Due March 11th

- Enhance the middleware provide information services and track user metadata.
  - Users should be able to see a job history in the user interface
  - Events should be intercepted by the portal.
- Enhance the User Interface to support LAMMPS submission on Big Red II as well as Karst
- Design and publish schema in GitHub for your metadata system.
  - Computing, applications, and experiments must be described.
- Design and publish a description of your information management system.
- Submitting the Assignment
  - Make a named branch of your master branch (milestone-4).
  - Put all instructions in README-MILESTONE4.md. You could also use GitHub's wiki feature.
  - Submit the assignment in Canvas when you are ready. Include pointers to the appropriate GitHub branch and the documentation.

Milestones, Generally

- Submissions
  - Late submissions are OK
  - But when semester ends, its over. Don't get behind.
  - Milestones build on each other, but I will not accept "combined" milestone submissions.
- I must be able to compile and run everything from a blank GitHub checkout.
- I am happy to install Tomcat, open source databases, etc as needed **if you provide the instructions.**
  - But "run it in Eclipse" is not acceptable.
  - Your project must be deployable onto a VM as a service.
- Hopefully you see where this is going.
  - You will be required to automate your entire deployment stack in upcoming milestones.

March 8th Special Guests

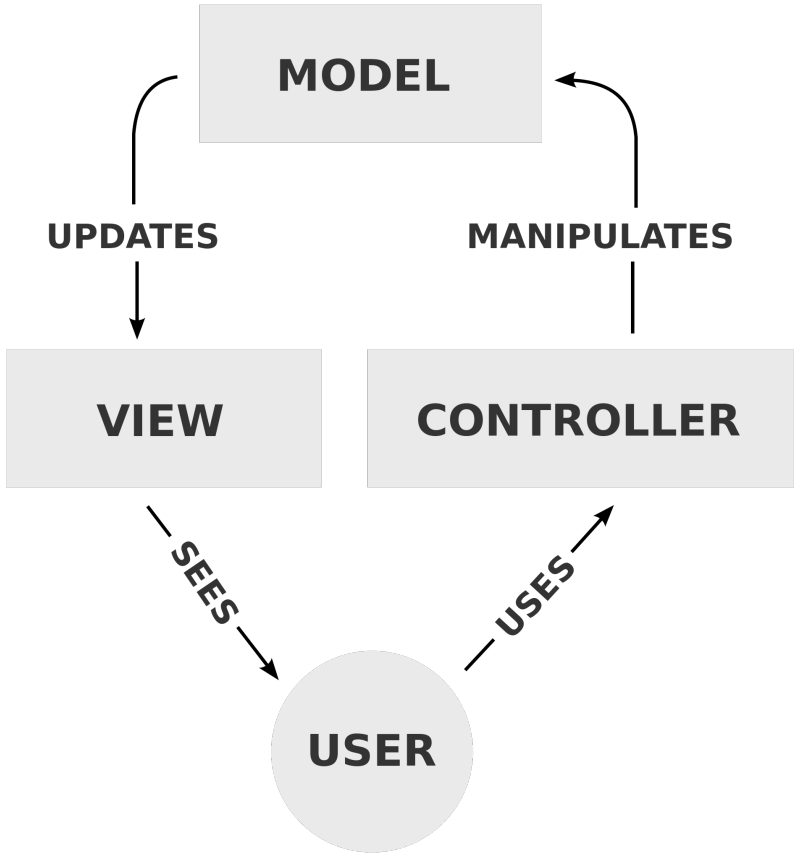
- George Turner from the IU Jetstream team will give an overview of Jetstream
- Rich Bowen
  - Apache Software Foundation Executive Vice President
  - Openstack Community Liaison, Red Hat inc.
- Jen Krieger
  - RedHat Agile Coach
  - DevOps, Continuous Integration and Continuous Delivery experiences

Science Gateways and Metadata Management

Designing and prototyping information services for your science gateway.

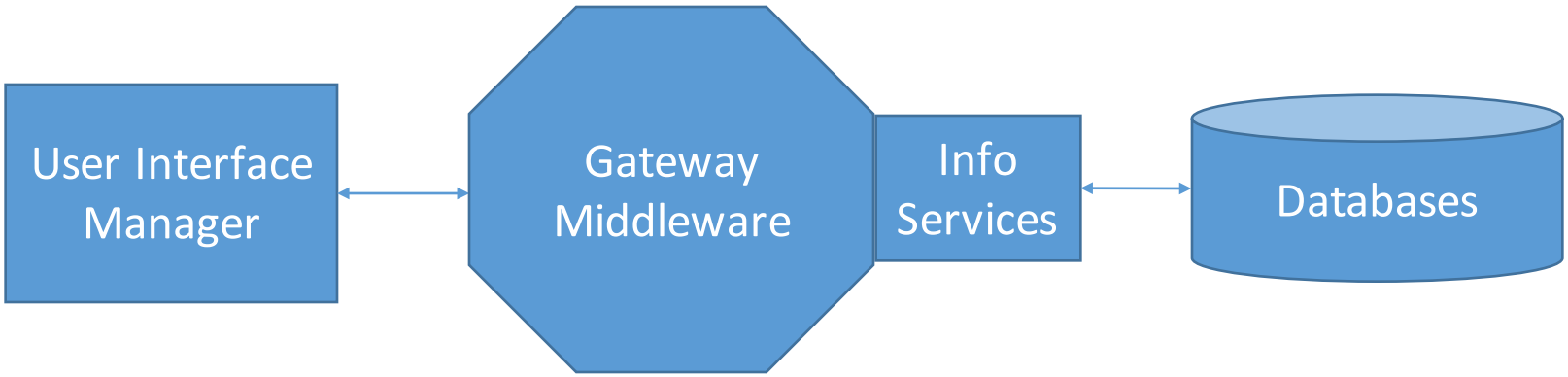
Basic Information Service Design

- Variation on well-known MVC pattern
- Think “client” instead of “user”
- Substitute “service” or “middleware component” for “controller”.
- View can be abstract
  - HATEOAS
  - Language bindings usable by other components



https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/2000px-MVC-Process.svg.png

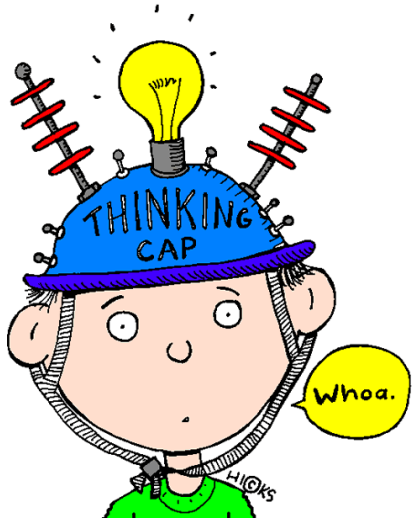
Basic Implementation



What's so hard about that?

Implementation Considerations

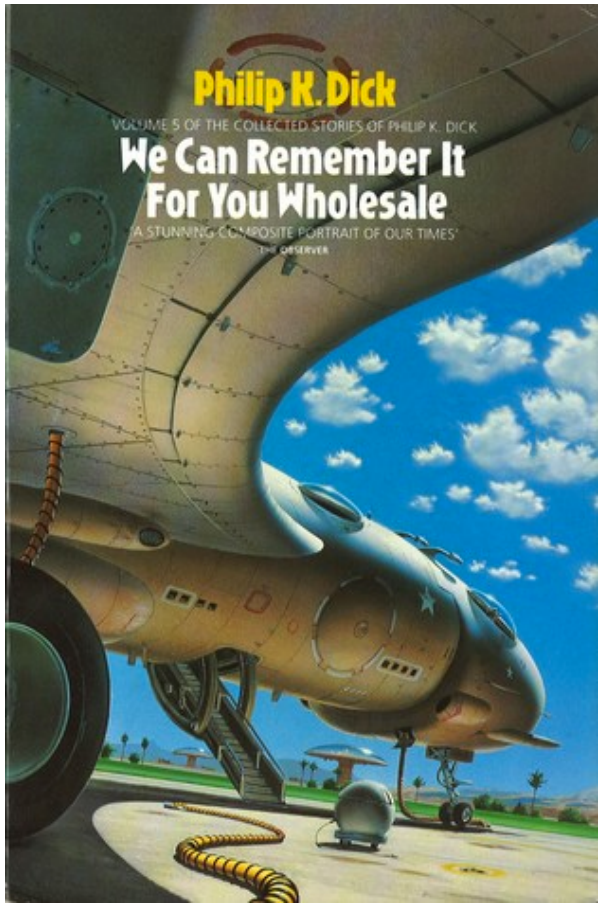
- Metadata can be both stateless and stateful
  - Stateful metadata➔ state of user's job
- Other middleware components can read and write to your metadata services.
  - You will need thought-out internal APIs
- Metadata models will change.
  - Novel resources, unplanned requirements, etc
- Middleware can be decoupled into microservices
- Lots of choices for how to implement the data stores
- Others?



Bootstrapping Your Information Service Designs

Recall REST and Thrift Lectures

- Think of your system as a collection of resources.
  - Your API methods are operations on these resources.
- How do you describe these resources?
  - That is your metadata.
- How do you interact with your metadata?
  - These are your information services.



http://d.gr-assets.com/books/1355133379l/1118429.jpg

This Is an Enormous Design Undertaking

- The **complete problem** is to describe **all the possible ways** of describing **all possible applications** on **all possible resources.**
- You don't need to do this for the project milestone.
- You instead will design around a limited subset.
  - LAMMPS on Karst and LAMMPS on Big Red II

http://cdn.silodrome.com/wp-content/uploads/2013/02/Endeavour-Shuttle-Cockpit.jpg

```
#/bin/bash

#PBS -l nodes=2:ppn=8,walltime=2:30:00
#PBS -m ae
#PBS -N job_myprog

cd $PBS_O_WORKDIR
mpirun -np 16 lmp_openmpi -suffix omp < in.cmdfile
```

LAMMPS on Karst

```
#!/bin/bash

#PBS -l nodes=4:ppn=16,walltime=3:00:00
#PBS -q gpu
#PBS -o out.log
#PBS -e err.log

cd $PBS_O_WORKDIR
aprun -n 1 -N 1 lmp_xe6 -cuda off -suffix gpu < in.cmdfile
```

LAMMPS on Big Red II

Generating these scripts is a key capability for science gateways.

General Advice

- Break the problem down into doable pieces
- Work from concrete problems that you understand
- But remember that your solution will need to evolve
  - Or be rewritten entirely
- Taking this advice is a challenge
  - Information services are one of the pillars of your gateway
  - They can easily become monolithic and hard to change

What Models Do We Need?

You tell me

My Short List of Models

Stateless

- Computers to run applications
- Applications
- Users
- Storage

Stateless: write operations are typically human driven

Stateful

- Jobs that users run
- State of computing resources

Stateful: changes to the metadata are made by other components.

Modeling Computing Resources

```
#/bin/bash

#PBS -l nodes=2:ppn=8,walltime=2:30:00
#PBS -m ae
#PBS -N job_myprog

cd $PBS_O_WORKDIR
mpirun -np 16 lmp_openmpi -suffix omp < in.cmdfile
```

LAMMPS on Karst

```
#!/bin/bash

#PBS -l nodes=4:ppn=16,walltime=3:00:00
#PBS -q gpu
#PBS -o out.log
#PBS -e err.log

cd $PBS_O_WORKDIR
aprun -n 1 -N 1 lmp_xe6 -cuda off -suffix gpu < in.cmdfile
```

LAMMPS on Big Red II

Generating these scripts is a key capability for science gateways.

Machine Metadata Descriptions

What do we know about Karst and Big Red II?

Computing Resources Critical Metadata

Name of resource	• A canonical name.
Unique identifier	• Internal UID for this resource.
Access points	• IP addresses, DN
Methods for submitting jobs	• SSH, other remote submission methods.
Scheduler, queuing system used	• May be none
Important paths	• Where is qsub?
Important directories	• $HOME, $WORK or $SCRATCH
Available applications	• Hmm…we'll probably want to link this

Describing Applications

Describing Applications

- Airavata found it useful to treat application interfaces and application deployments on resources as distinct metadata
- **Application Interface Metadata:** universally true information about an application, regardless of where it runs
  - Inputs and outputs
- **Application Deployment Metadata:** Details about how to run an application on a specific resource
  - Modules you need to load
  - Specific command lines
  - Consider LAMMPS on Karst, LAMMPS on Big Red 2

```
#/bin/bash

#PBS -l nodes=2:ppn=8,walltime=2:30:00
#PBS -m ae
#PBS -N job_myprog

cd $PBS_O_WORKDIR
mpirun -np 16 lmp_openmpi -suffix omp < in.cmdfile
```

LAMMPS on Karst

```
#!/bin/bash

#PBS -l nodes=4:ppn=16,walltime=3:00:00
#PBS -q gpu
#PBS -o out.log
#PBS -e err.log

cd $PBS_O_WORKDIR
aprun -n 1 -N 1 lmp_xe6 -cuda off -suffix gpu < in.cmdfile
```

LAMMPS on Big Red II

What are the common abstractions for the last line?

Application Interface Metadata

What do we need?

Application Interface Metadata for Inputs

- Name of the application and description
- Unique identifier
- Inputs
  - Zero or more
  - Inputs can be files or command line parameters
  - Inputs can be standard input (”<“) or not.
  - Inputs may have flags
  - Order may or may not be important
  - Inputs can be implicit
  - Can be typed
  - Optional or required

Application Interface Outputs

- Applications can have zero or more outputs
- Outputs can be explicitly named on the command line
- Can also be implicit: generated but not named
- Outputs can be conditional on other parameters
- Outputs can be required on the command line or not.

Application Deployment Metadata

Describe how a code runs on a particular resource

```
module swap PrgEnv-cray PrgEnv-gnu
module load fftw
module load cudatoolkit
module load lammps/gnu/gpu/15May15
```

One module combination for LAMMPS on Big Red II

- **GCC:** To load all three required packages, on the command line, enter these commands in this order:

```
module load gcc
module load openmpi/gnu
module load lammps
```

Two module combinations for LAMMPS on Karst

- **Intel:** To load all three required packages, on the command line, enter these commands in this order:

```
module load intel
module load openmpi/intel
module load lammps
```

Application Deployment Metadata

You tell me

Application Deployment Metadata Examples

- Pointer to the application interface description.
- Pointer to the machine description
- Modules to load
  - Alternatively, environment variables.
- Path the application's binary on the resource.

Job Metadata

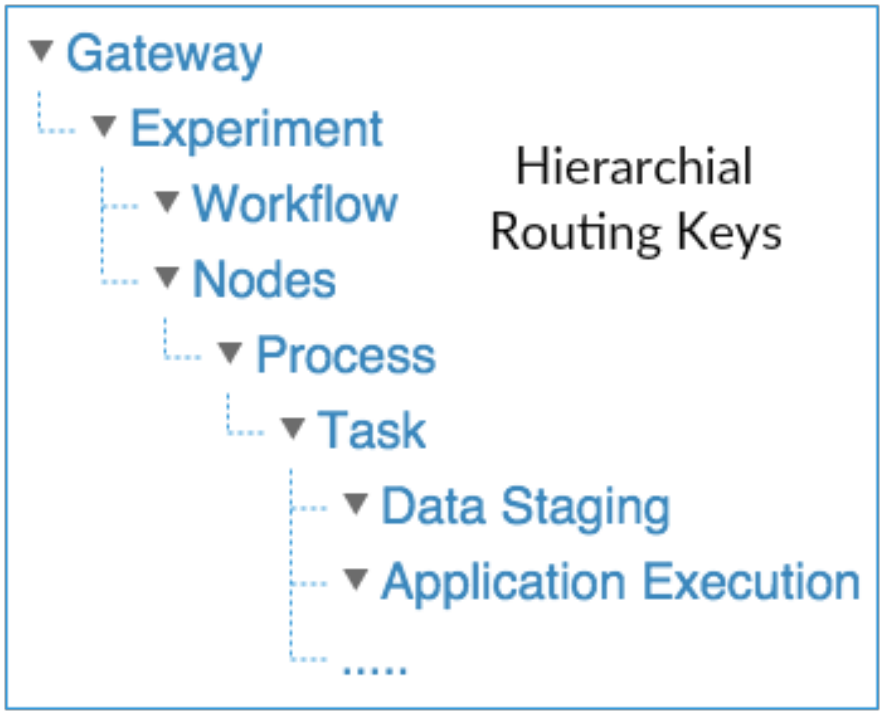
What do we need?

Let's Call Them "Experiments"

- Each invocation of the gateway by a user creates metadata.
  - The code used, the host used, the inputs, etc
- Each submission is a pipeline: stage in data, run code, stage out data
  - So it is useful to think of an experiment as containing this pipeline.
- Why? Failures
  - A job could run correctly but the stage out operation could fail.

Experiments in Airavata

- Airavata's design takes into account advanced execution patterns.
- Workflows
  - Dependent application in a directed acyclic graph
- Multi-part independent executions
  - Parameter sweeps



Experiment Metadata

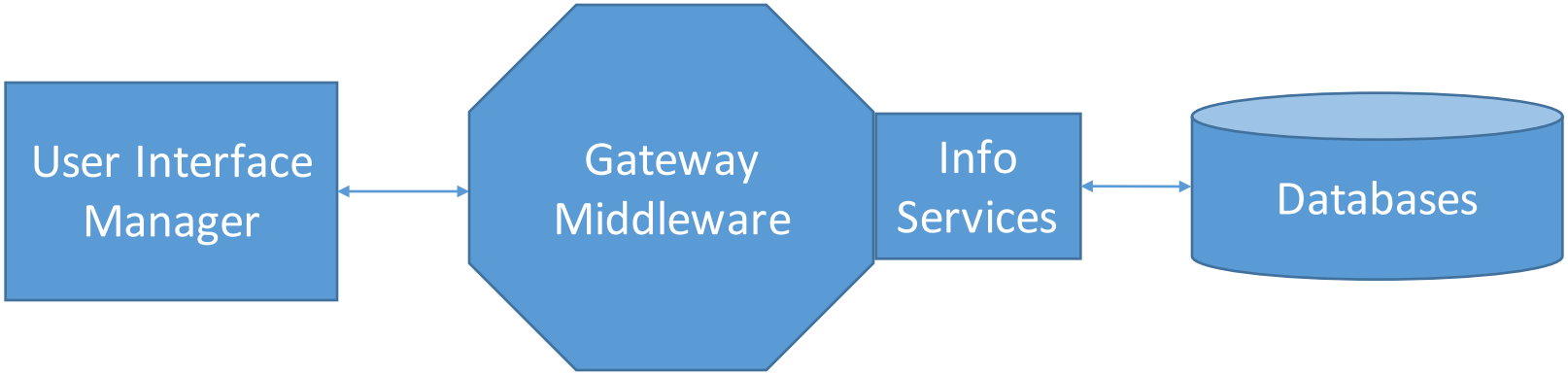
What do we need? You tell me.

Implementing Metadata Services

Metadata Services

- Metadata needs to be stored somewhere.
- Now that you have data models, you need to choose a backing technology.
  - Relational DBs: Postgres, MariaDB
  - Document DBs: MongoDB, CouchDB
  - Graph DBs (including Triplestores)
  - Object DBs

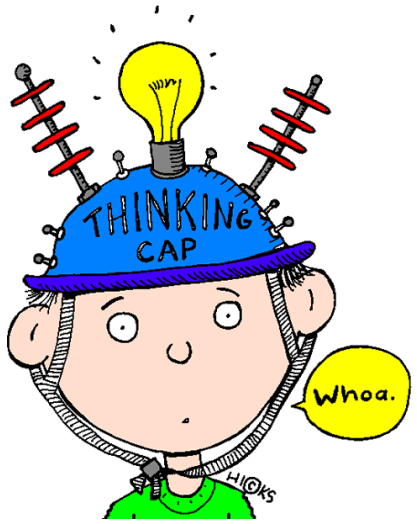
Basic MVC-Like Implementation



What's so hard about that?

Implementation Considerations

- Metadata can be both stateless and stateful
  - Stateful metadata➔ state of user's job
- Other middleware components can read and write to your metadata services.
  - You will need thought-out internal APIs
- Metadata models will change.
  - Novel resources, unplanned requirements, etc
- Middleware can be decoupled into microservices
- Lots of choices for how to implement the data stores



Database Considerations

- Building monolithic databases, regardless of the technology, can lead to problems.
  - Can you think of some?

My List

- Entangled tables make changes very difficult.
- Backward-forward compatibility
- Fragility, inertia
  - “Don’t touch it. It works.”
- Hard to modify code bases
- Hard to change DB technologies if you have made some bad choices.
  - “Oracle sure is expensive!”
  - “Why did we waste time on that NoSQL DB?”
  - “ORM overheads are killing performance”

Microservices

- Microservice design patterns alleviate these problems
- Will lead to others.
- More about this in future lectures

Are there standards?

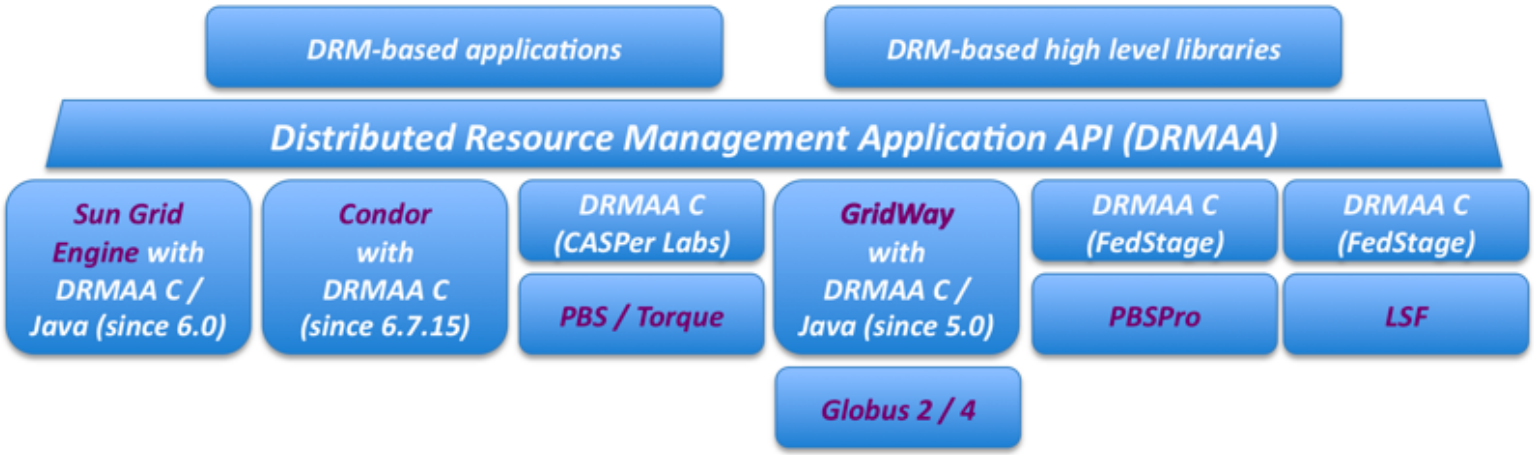
Why don’t we have standard solutions for all of these known challenges?

A Brief Survey of Standards Work

Distributed Resource Management Application API (DRMAA)

http://www.drmaa.org/drmaastack.png

- Portable programmatic access to cluster, grid, and cloud systems.
- Vendors can provide a standardized access to their product through a DRMAA implementation.
- API designers, meta-scheduler architects and end users: unified access to execution resources.
- Focused on job submission, job control, reservation management, and retrieval of job and machine monitoring information.



DRMAA v2 Job Info

DRMAA
Distributed Resource Management Application API — www.drmaa.org

```
interface JobInfo {
      readonly attribute Dictionary resourceUsage;
      readonly attribute boolean hasExited;
      readonly attribute long exitStatus;
      ... [old DRMAA1 job information] ...
      readonly attribute JobState jobState;
      readonly attribute string jobSubState;
      readonly attribute string masterMachine;
      readonly attribute string[] slaveMachines;
      readonly attribute string submissionMachine;
      readonly attribute string jobOwner;
      // amount of time since job was started
      readonly attribute long wallclockTime;
      // amount of time remaining until the job will be terminated
      readonly attribute long wallclockLimit;
      // amount of CPU seconds consumed
      readonly attribute long cpuTime;
      // and so on for submission time, dispatch time, start time, finish time,
      // memory usage and limits
      ...
};
```

Open Archives Initiative Object Reuse and Exchange (OAI-ORE)

Defines standards for the description and exchange of aggregations of Web resources.

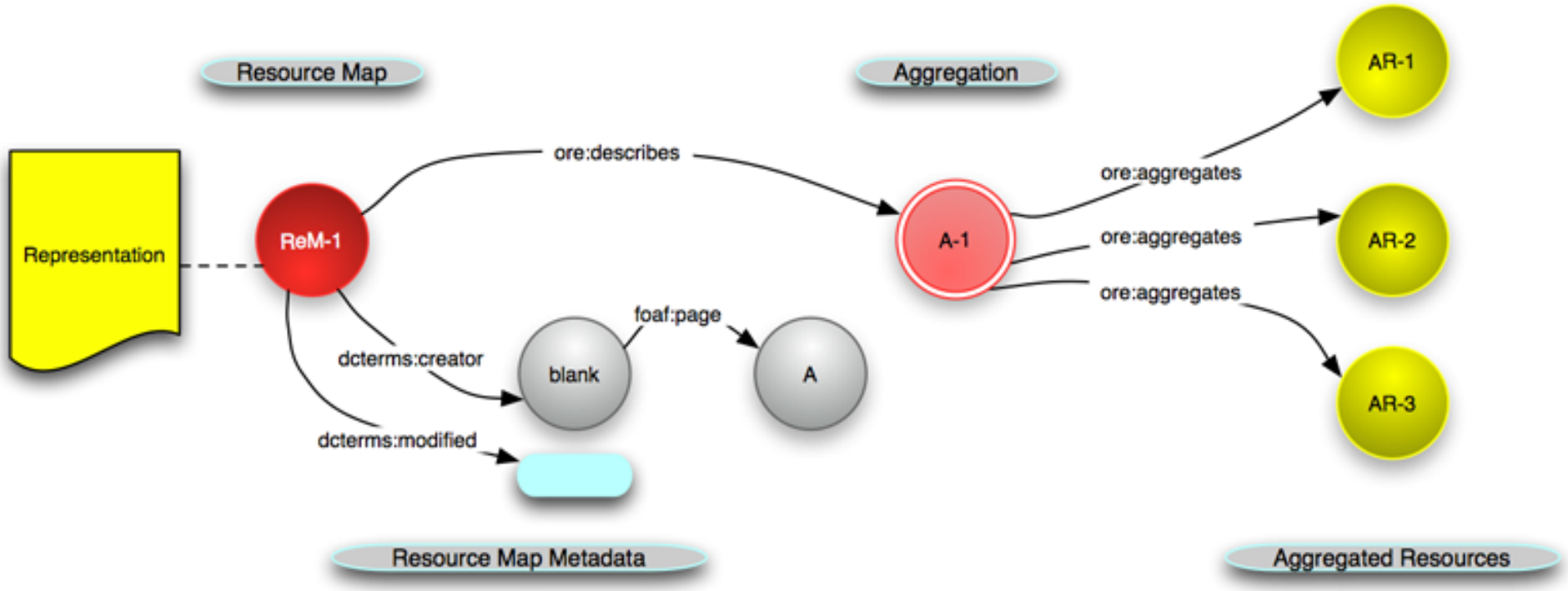
- Aggregations are compound digital objects
- For example, scientific papers on the Web

May combine distributed resources with multiple media types

- Text, Images, Data, Video
- These are links in an ORE document.

https://www.openarchives.org/ore/

ORE Illustrative Data Model



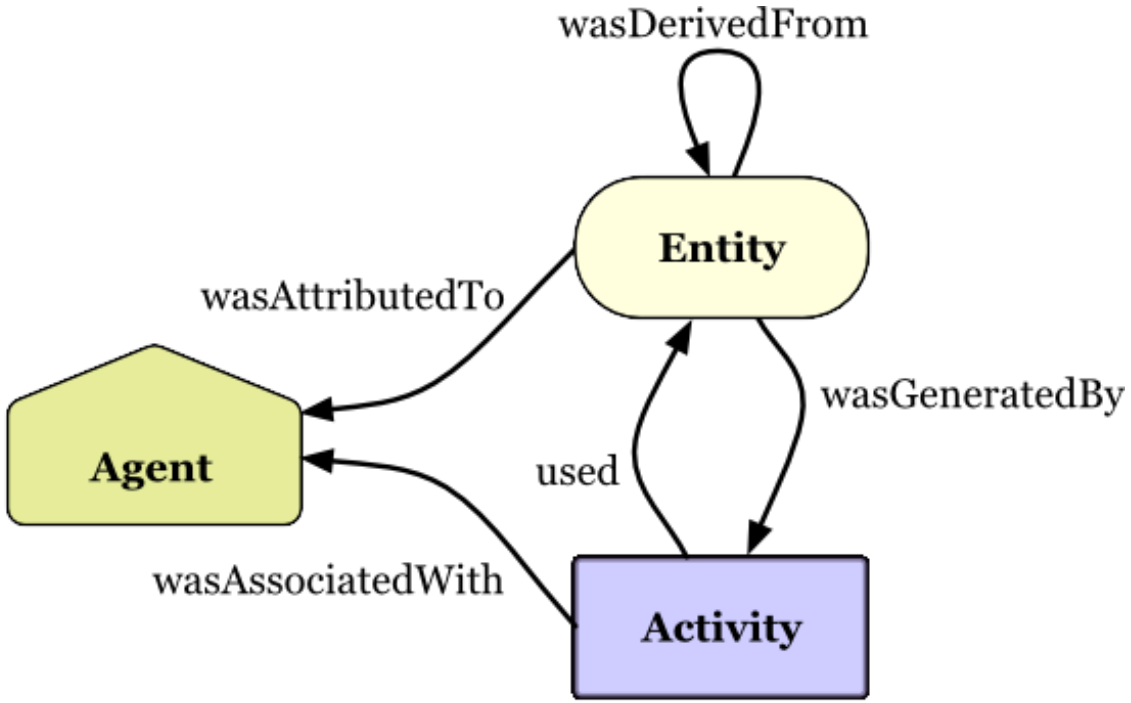
ORE data models can be expressed in RDF, Atom, and other formats.

```
<rdf:Description rdf:about="http://arxiv.org/aggregation/astro-ph/0601007">
    <!-- The Resource is an ORE Aggregation  -->
    <rdf:type rdf:resource="http://www.openarchives.org/ore/terms/Aggregation"/>
    <!-- The Aggregation aggregates ... -->
    <ore:aggregates rdf:resource="http://arxiv.org/abs/astro-ph/0601007"/>
    <ore:aggregates rdf:resource="http://arxiv.org/ps/astro-ph/0601007"/>
    <ore:aggregates rdf:resource="http://arxiv.org/pdf/astro-ph/0601007"/>
    <!-- Metadata about the Aggregation: title and authors -->
    <dc:title>Parametrization of K-essence and Its Kinetic Term</dc:title>
    <dcterms:creator rdf:parseType="Resource">
        <foaf:name>Hui Li</foaf:name>
        <foaf:mbox rdf:resource="mailto:lihui@somewhere.cn"/>
    </dcterms:creator>
    <dcterms:creator rdf:parseType="Resource">
        <foaf:name>Zong-Kuan Guo</foaf:name>
    </dcterms:creator>
    <dcterms:creator rdf:parseType="Resource">
        <foaf:name>Yuan-Zhong Zhang</foaf:name>
    </dcterms:creator>
</rdf:Description>
```

Example RDF/XML serialization of a scientific paper, from the ORE primer.

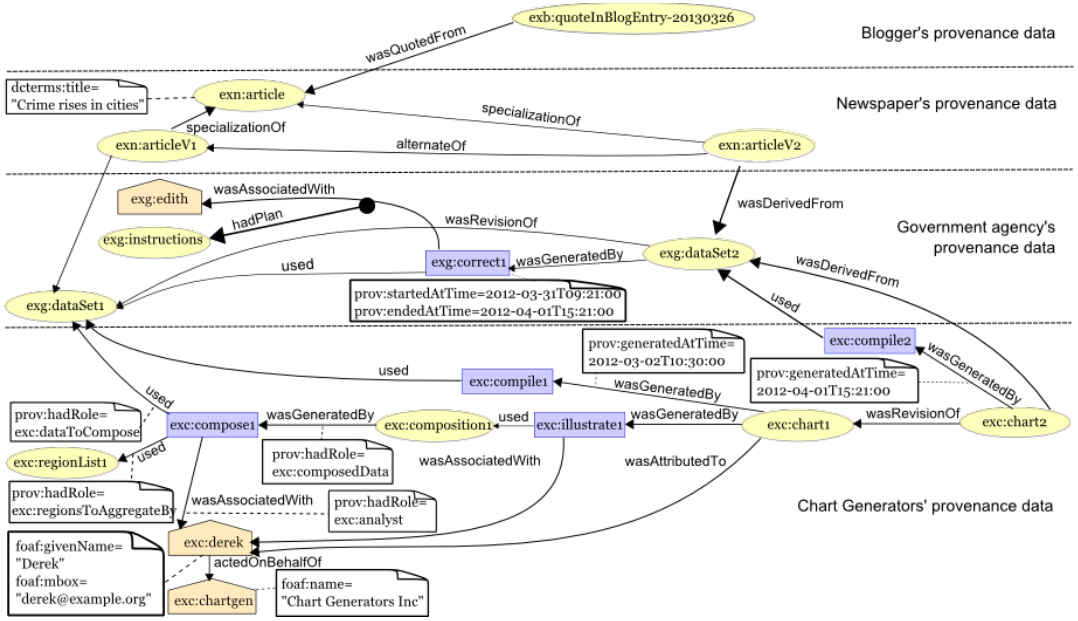
W3C-PROV

- General purpose standard
- Can be used to describe provenance of anything.
- Entity: the thing being described
- Activity: the action taken on the entity by an agent.
- Agent: instigates the activity



https://www.w3.org/TR/prov-overview/

The W3C-Primer has a full example, if you are interested.



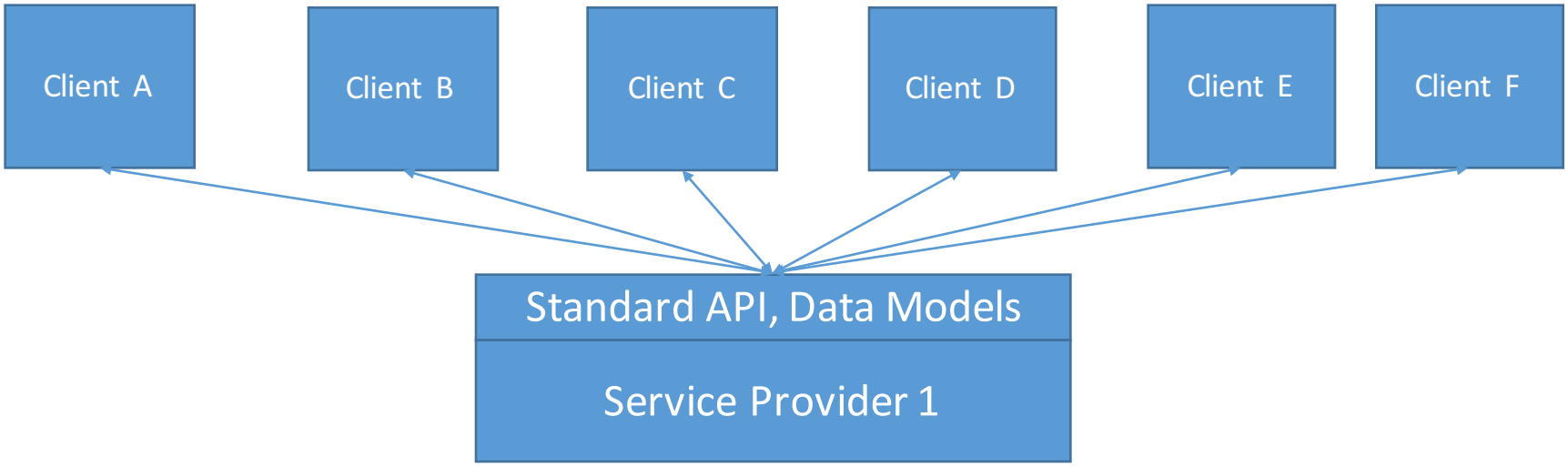
https://www.w3.org/TR/prov-primer/primer-prov-xml-examples.xml

Opinions to Follow

Should Science Gateways Use These?

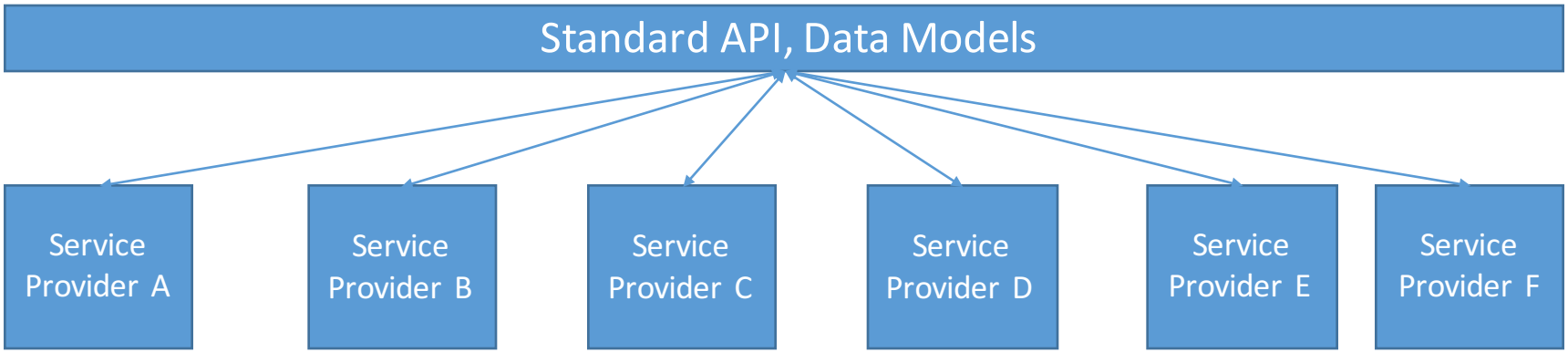
- We chose not to in Apache Airavata
- Standardization has a lot of problems
  - Leads to least common denominator solutions
  - Space for applications and resources is very large and evolves more rapidly than standards
  - Vertical versus horizontal standardization drivers
- We prefer instead open reference implementations
  - Collaborate on common solutions instead of specifications.
- Better to expand outward from things that work than to try to find universal patterns that may not exist

Vertical (Upstream/Downstream) Standards



- One or more service providers and multiple clients negotiate a common standard.
- Service consumers as well as providers are stakeholders in the standards.

Horizontal Standards



- Competing service providers agree to a set of standards on common problems.
- Goal: interoperability
- Most likely to work if the service providers agree to a common reference implementation of the standards
- Anti-pattern: multiple implementations of the standard.