The Virtual Data System – *a workflow toolkit for TeraGrid science applications*

TeraGrid '06
Indianapolis, IN
June 12, 2006

Ben Clifford[1] benc@mcs.anl.gov
Gaurang Mehta[3] gmehta @isi.edu
Karan Vahi[3] vahi @isi.edu
Michael Wilde[1,2] wilde@mcs.anl.gov

[1]Computation Institute, University of Chicago
[2]Mathematics and Computer Science Division,
Argonne National Laboratory
[3]Center for Grid Technologies,
USC Information Sciences Institute

VDS Tutorial Outline

- Part 1: The concept of Virtual Data
  - 1:30 PM: 30 minutes
- Part 2: Basics of VDL
  - 2:00 PM: 45 minutes
- *Break: 2:45 – 3:00*
- Part 3: Pegasus: Grid Workflow Planning
  - 3:00 PM: 90 Minutes
- Part 4 : VDS in the Science Process
  - Summary and Conclusion, Q & A
  - 4:30 PM: 30 minutes

VDS Tutorial Lab Exercises

- Part 1: Virtual Data Concept
  - Ex 1.1: Hello World in VDL
  - Ex 1.2: Test the Cosmic Ray example code (locally)
- Part 2: Basics of VDL
  - Ex 2.1: Running Single Transformations (simplevdl)
  - Ex 2.2: Chaining Derivations (bigervdl)
  - Ex 2.3: Compound VDL (compoundvdl)
- Part 3: Pegasus and Grid Workflow Planning
  - EX 3.1: Cataloging data files in the RLS replica system
  - Ex 3.2: Setting up the site and transformation catalogs
  - Ex 3.3: Running the planner and starting the workflow
  - Ex 3.4: Watching workflow progress and debugging

The Virtual Data System Team

- Argonne and The University of Chicago:
  Ben Clifford, Ian Foster, Yong Zhao, Mike Wilde
- The USC Information Sciences Institute:
  Ewa Deelman, Carl Kesselman, Gaurang Mehta,
  Gurmeet Singh, Mei-Hui Su, Karan Vahi,
  Jens Voeckler
- Contributions by Jed Dobson, fMRI Data Center,
  Dartmouth College, and Luiz Meyer, UFRJ-Brazil,
  Doug Scheftner, University of Chicago

VDS Tutorial Outline

- Part 1: Concept & applications of Virtual Data
- Part 2: VDL - The Virtual Data Language
- Part 3: Pegasus: Grid Workflow Planning
- Part 4 : VDS in the Science Process
- Summary and Conclusion

Virtual Data Concept
Developed by GriPhyN Project – The Grid Physics Network

Enhance scientific productivity through:

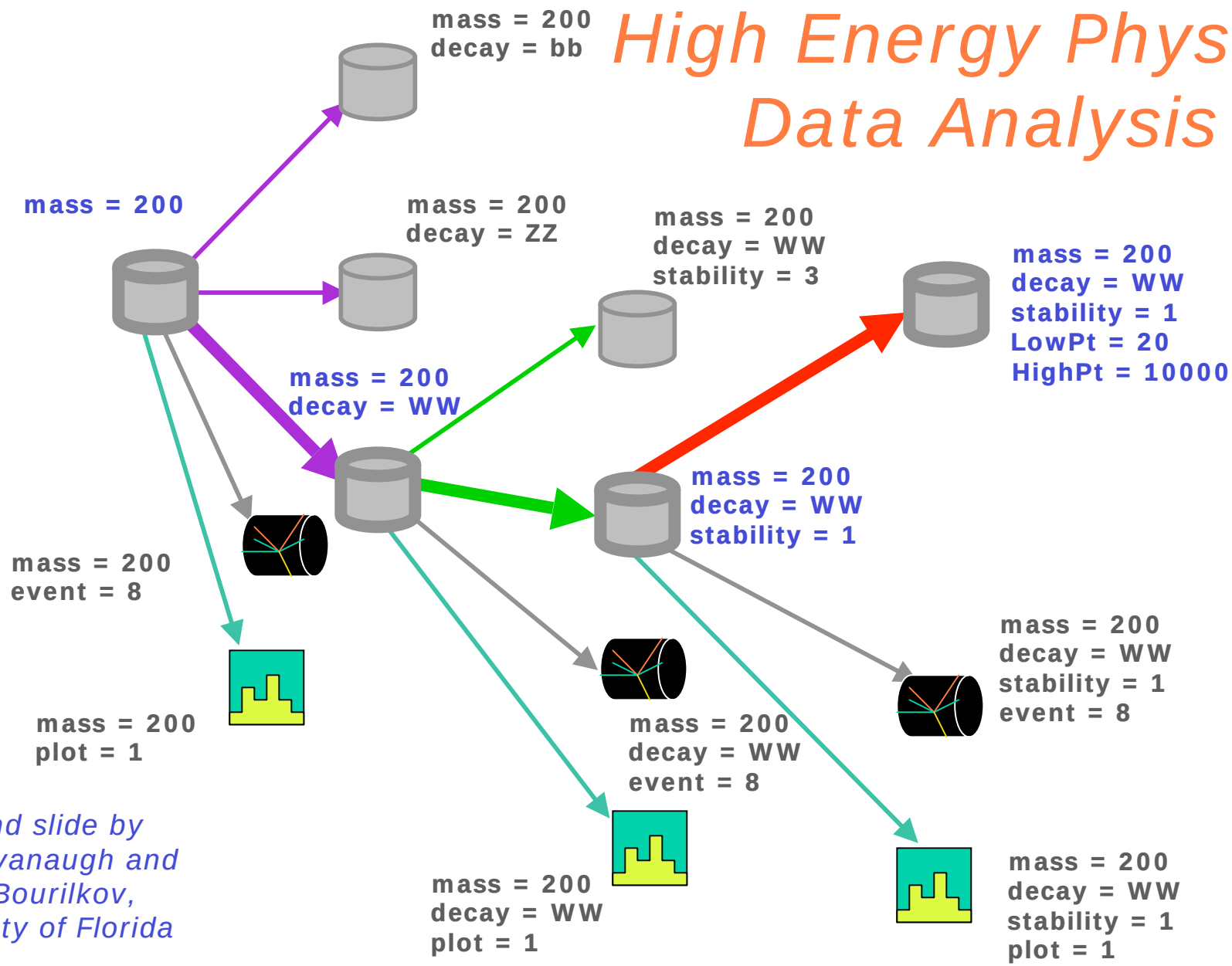
- Discovery and application of datasets and programs at petabyte scale
- Enabling use of a worldwide data grid as a scientific workstation

Virtual Data enables this approach by creating datasets from workflow "recipes" and recording their provenance.

Virtual Data and Workflows

- Challenge is managing and organizing the vast computing and storage capabilities provided by Grids
- **Workflow** expresses computations in a form that can be readily mapped to Grids
- **Virtual data** keeps accurate track of data derivation methods and **provenance**
- Grid tools **virtualize** location and caching of data, and recovery from failures

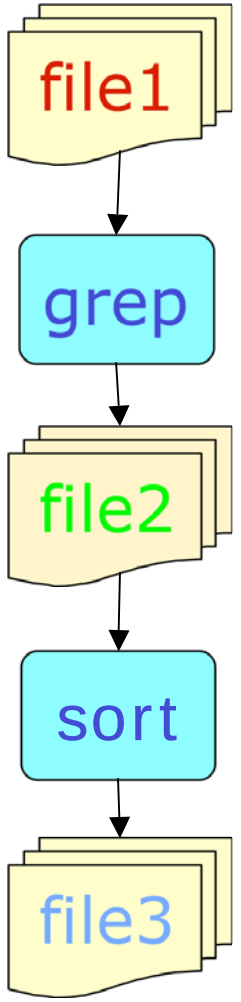
Virtual Data Application: *High Energy Physics Data Analysis*



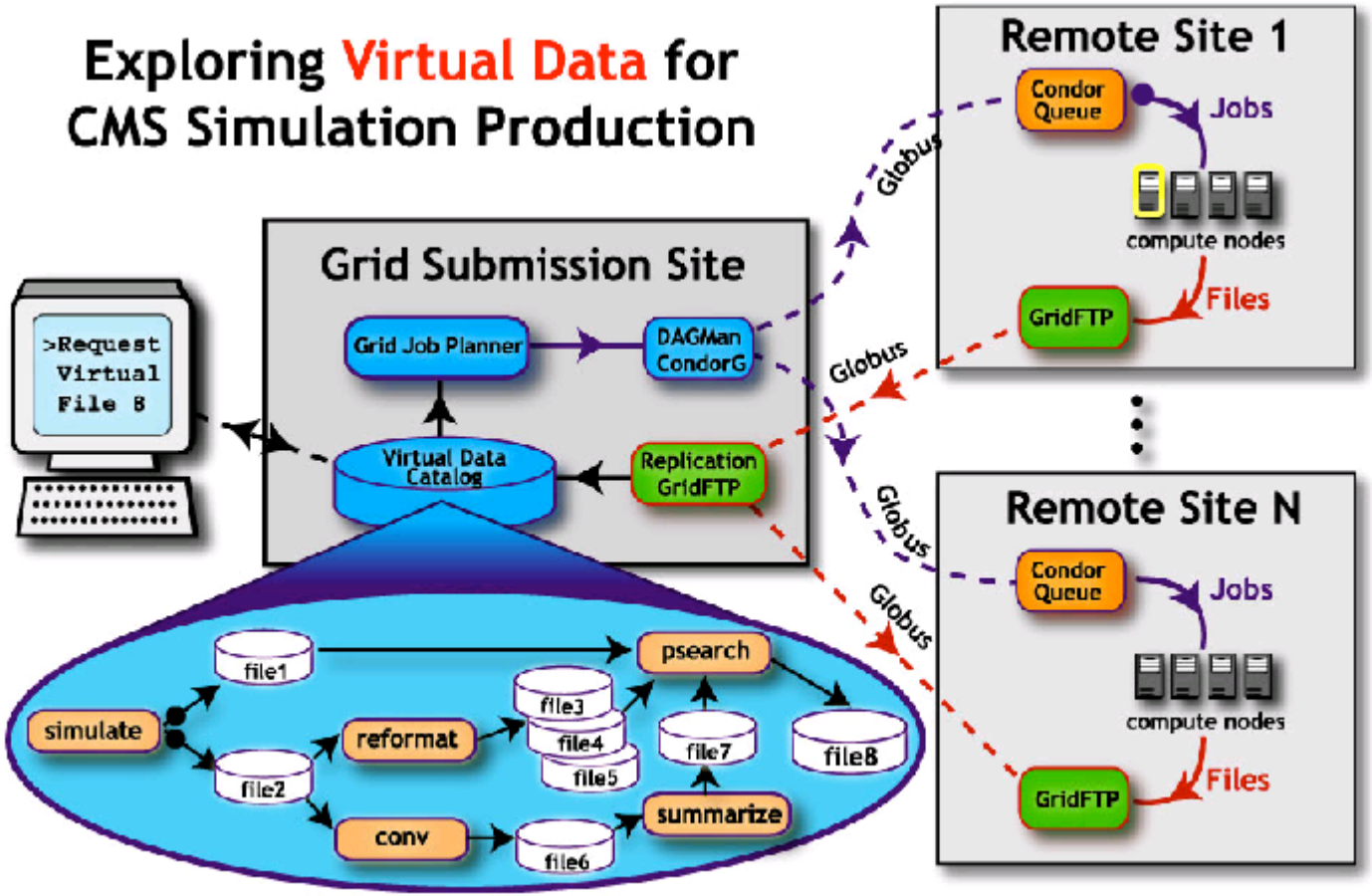
Work and slide by Rick Cavanaugh and Dimitri Bourilkov, University of Florida

Expressing Workflow in VDL

```
TR grep (in a1, out a2) {
    argument stdin = ${a1};
    argument stdout = ${a2}; }

TR sort (in a1, out a2) {
    argument stdin = ${a1};
    argument stdout = ${a2}; }

DV grep (a1=@{in:file1}, a2=@{out:file2});
DV sort  (a1=@{in:file2}, a2=@{out:file3});
```



Virtual Data Workflow Abstracts Grid Details

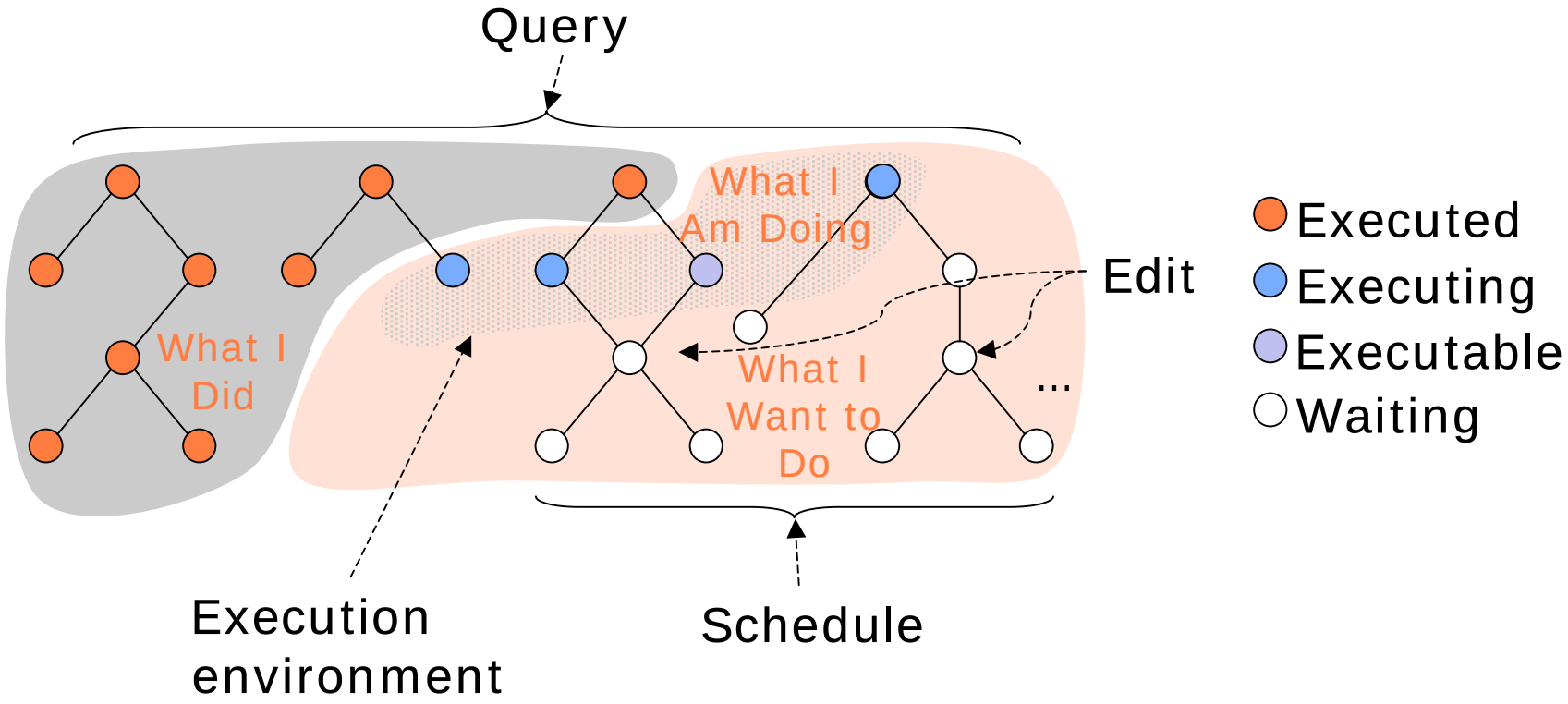


What must we “virtualize” to compute on the Grid?

- Location-independent computing: represent all workflow in abstract terms
- Declarations not tied to specific entities:
  - sites
  - file systems
  - schedulers
- Failures – automated retry for data server and execution site un-availability

How does Workflow Relate to Provenance?

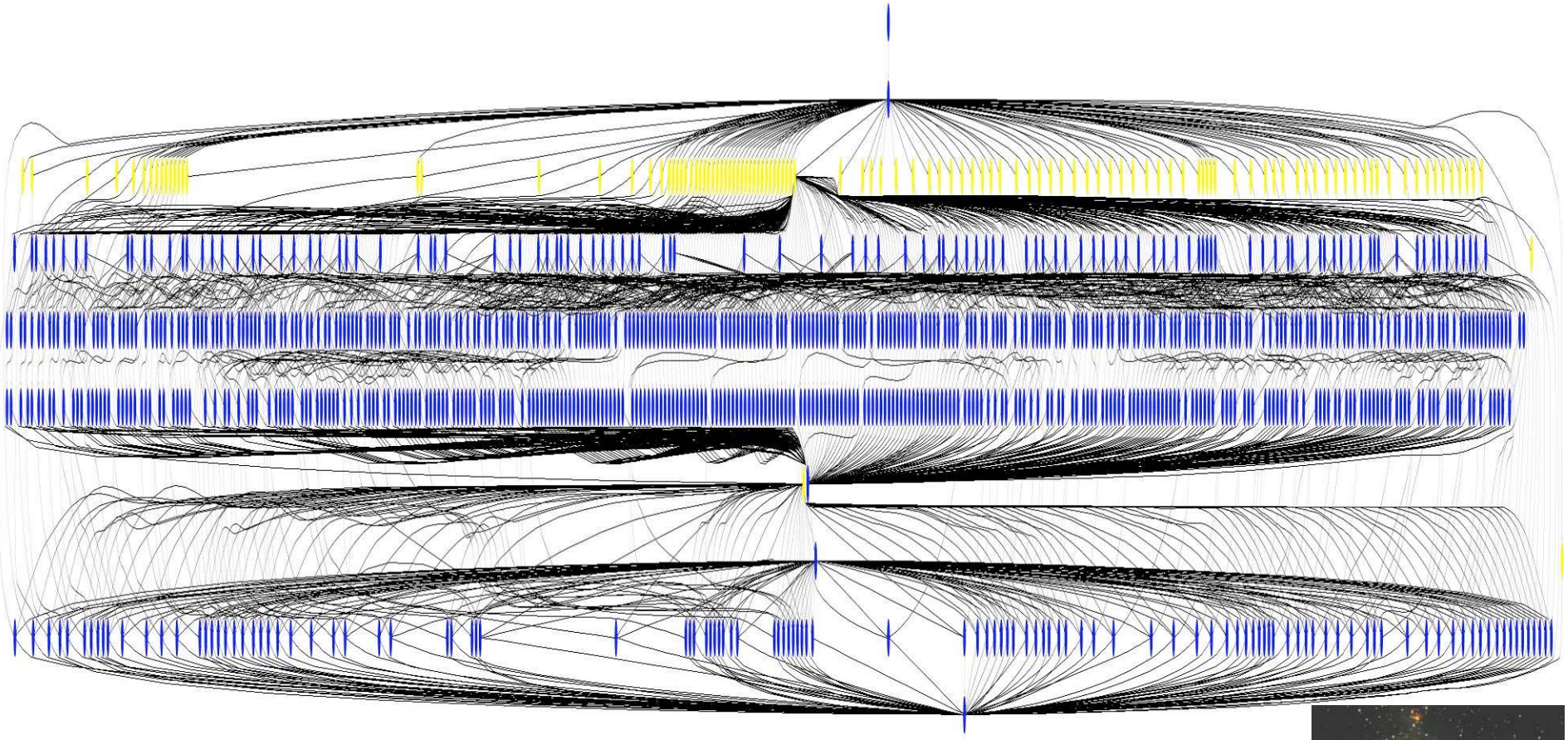
- Workflow – specifies what to do
- Provenance – tracks what was done
- Virtual Data integrates these capabilities



VDS Applications

Application	Jobs / workflow	Levels	Status
ATLAS HEP Event Simulation	500K	1	In Use
LIGO Inspiral/Pulsar	~700	2-5	Inspiral In Use
NVO/NASA Montage/Morphology	1000s	7	In Use
GADU Genomics: BLAST,...	40K	1	In Use
fMRI AIR, freeSurfer prepro	100s	12	In Devel
QuarkNet CosmicRay science	<10	3-6	In Use
SDSS Coadd; Cluster Search	40K 500K	2 8	CS Research
FOAM Ocean/Atmos Model	2000 (core app runs 250 8-CPU jobs)	3	In use
GTOMO Image proc	1000s	1	In Devel
SCEC Earthquake sim	1000s		In use

Small Montage Workflow

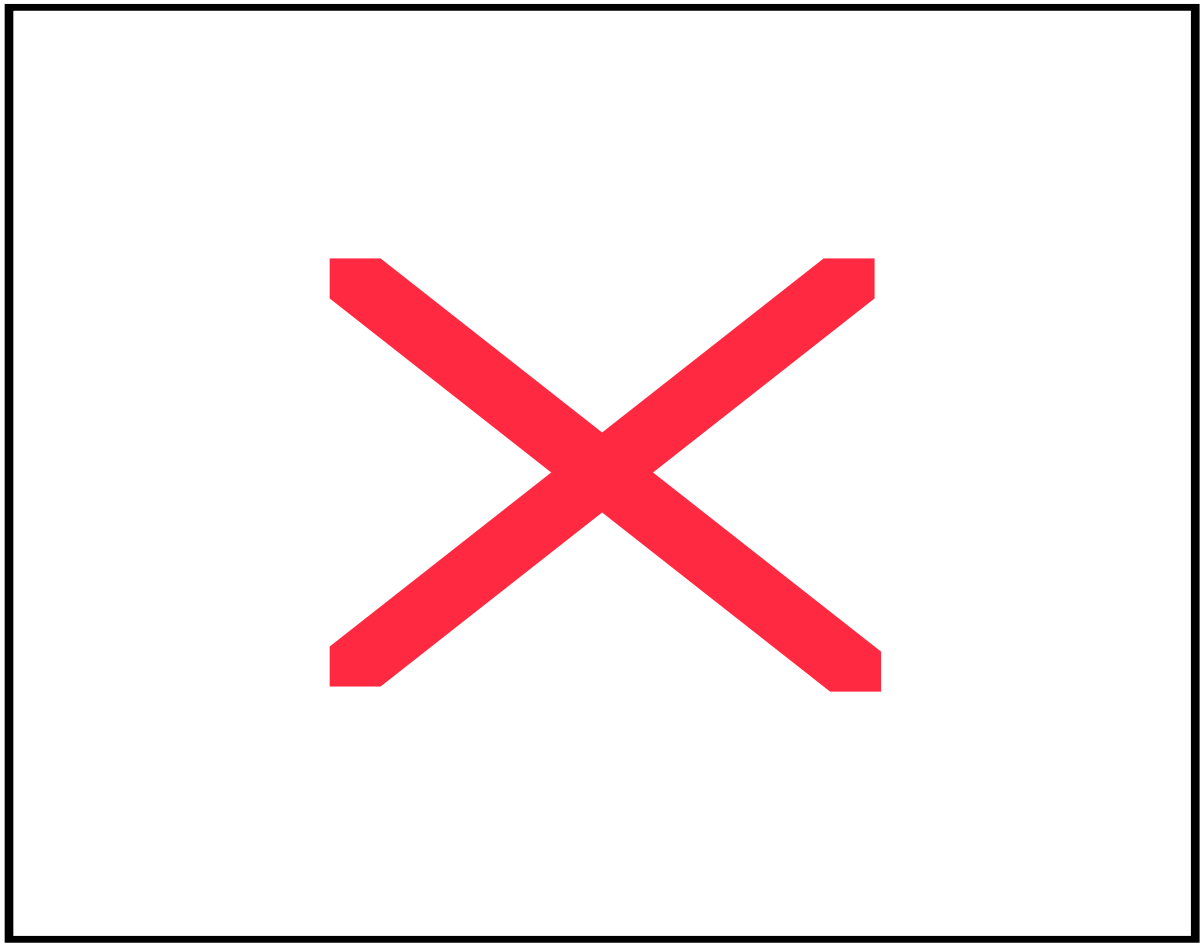


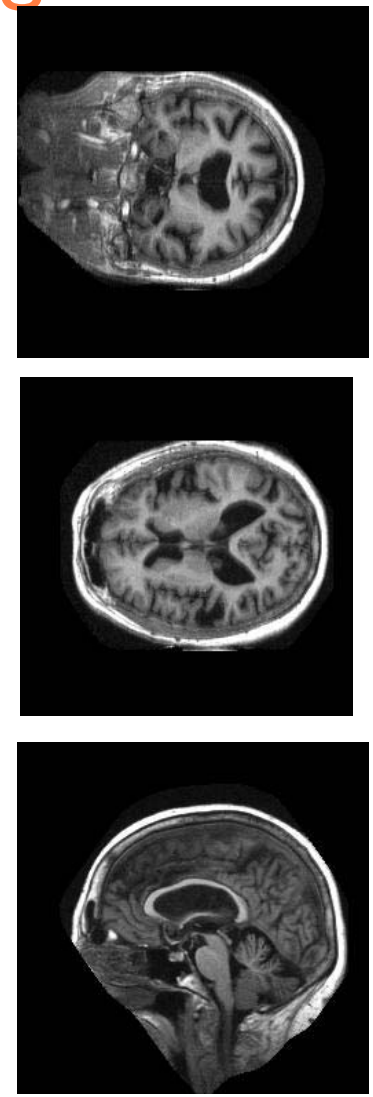
~1200 node workflow, 7 levels

Mosaic of M42 created on the Teragrid using Pegasus

http://montage.ipac.caltech.edu/

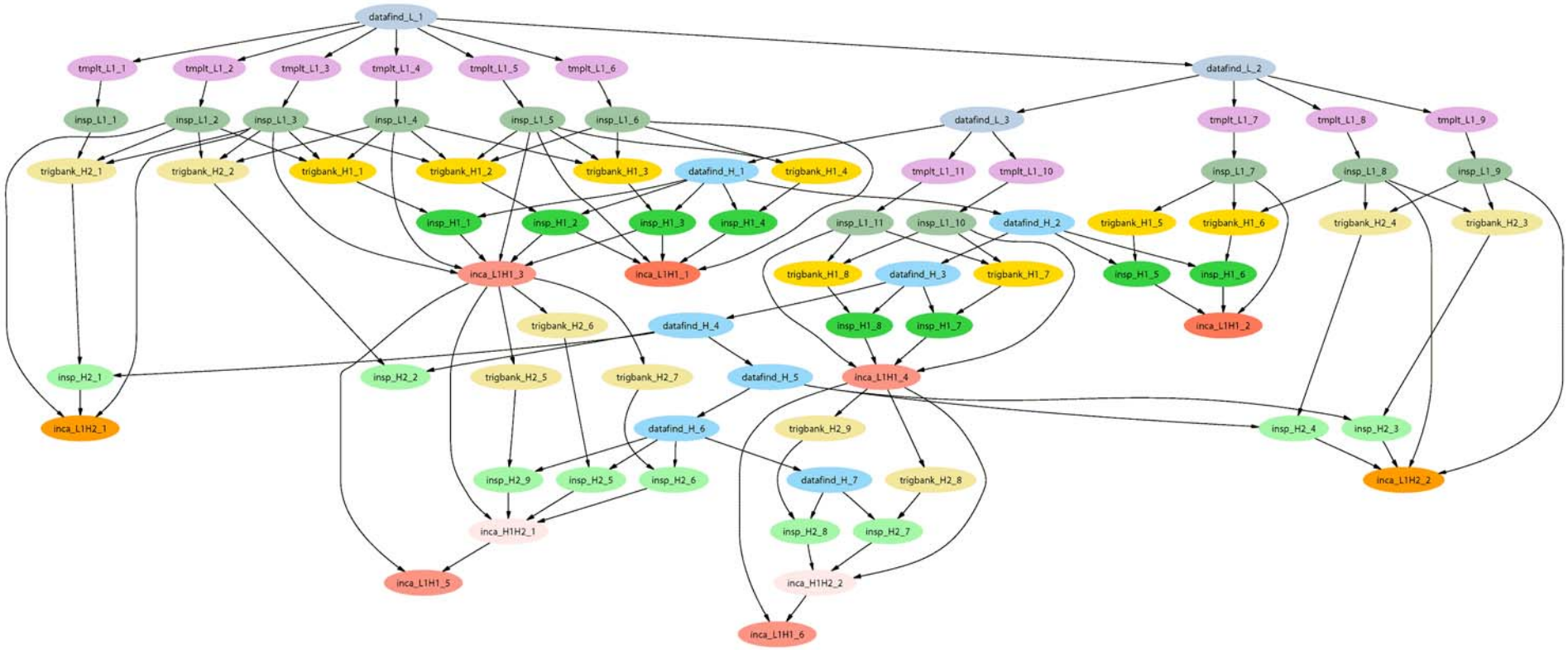
Functional MRI Analysis





Workflow courtesy James Dobson, Dartmouth Brain Imaging Center

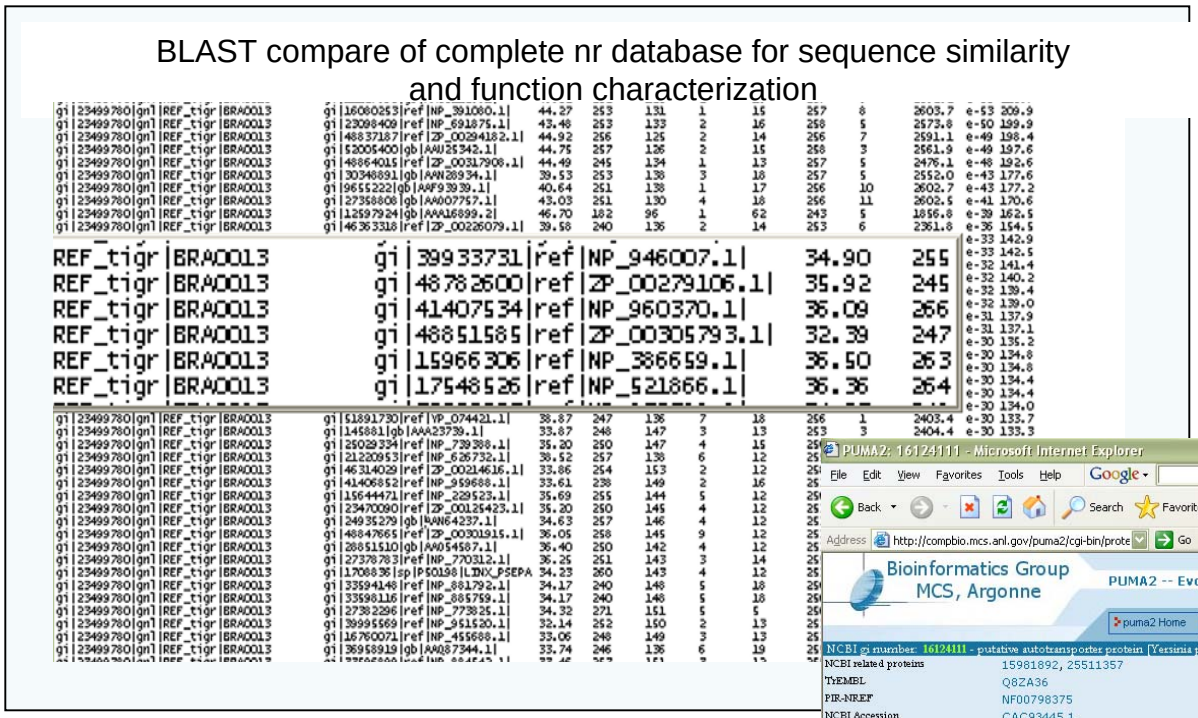
LIGO Inspiral Search Application



Inspiral workflow application is the work of Duncan Brown, Caltech, Scott Koranda, UW Milwaukee, and the LSC Inspiral group

Blasting for Protein Knowledge

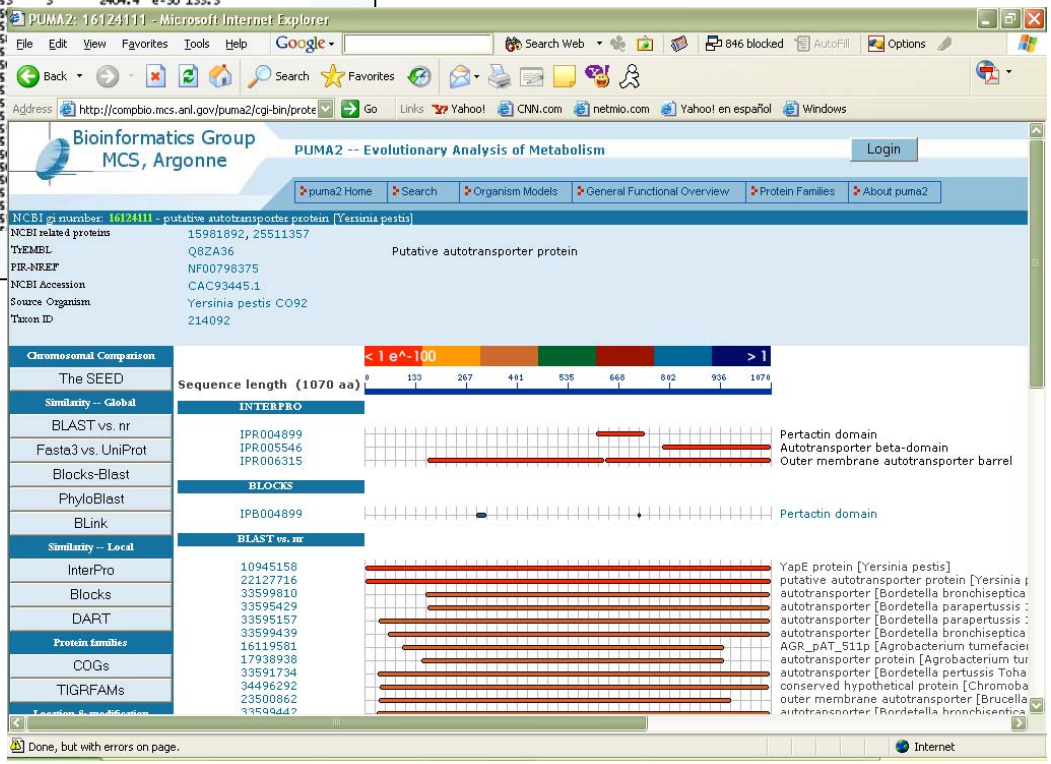
BLAST compare of complete nr database for sequence similarity and function characterization



REF_tigr|BRA0013	gi|39933731|ref|NP_946007.1|	34.90	255							e-32	142.5
REF_tigr|BRA0013	gi|48782600|ref|ZP_00279106.1|	35.92	245							e-32	141.4
REF_tigr|BRA0013	gi|41407534|ref|NP_960370.1|	36.09	266							e-32	140.2
REF_tigr|BRA0013	gi|48851585|ref|ZP_00305793.1|	32.39	247							e-32	139.4
REF_tigr|BRA0013	gi|15966306|ref|NP_386659.1|	36.50	263							e-32	139.0
REF_tigr|BRA0013	gi|17548526|ref|NP_521866.1|	36.36	264							e-31	137.9
										e-31	137.1
										e-30	136.2
										e-30	134.8
										e-30	134.8
										e-30	134.4
										e-30	134.4
										e-30	134.0
										e-30	133.7
										e-30	133.3

Knowledge Base

PUMA is an interface for the researchers to be able to find information about a specific protein after having been analyzed against the complete set of sequenced genomes (nr file ~ approximately 3 million sequences)

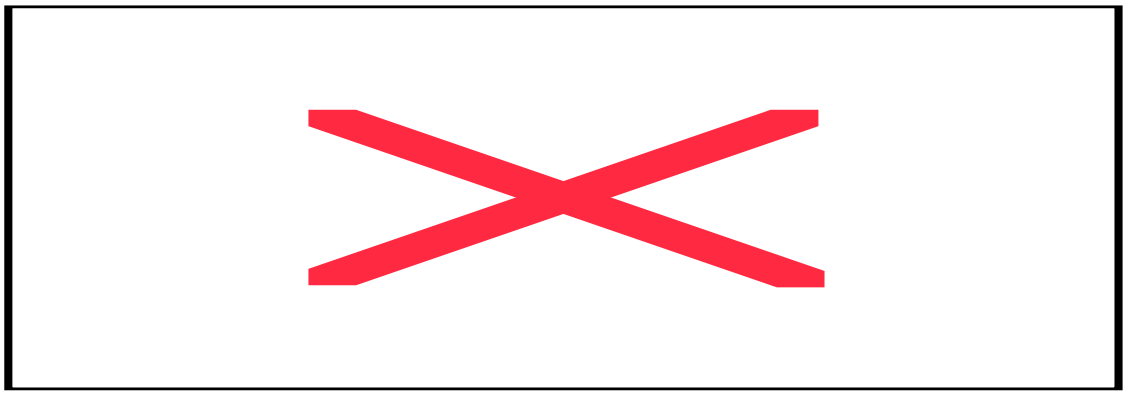


Analysis on the Grid

The analysis of the protein sequences occurs in the background in the grid environment. Millions of processes are started since several tools are run to analyze each sequence, such as finding out protein similarities (BLAST), protein family domain searches (BLOCKS), and structural characteristics of the protein.

Southern California Earthquake Center

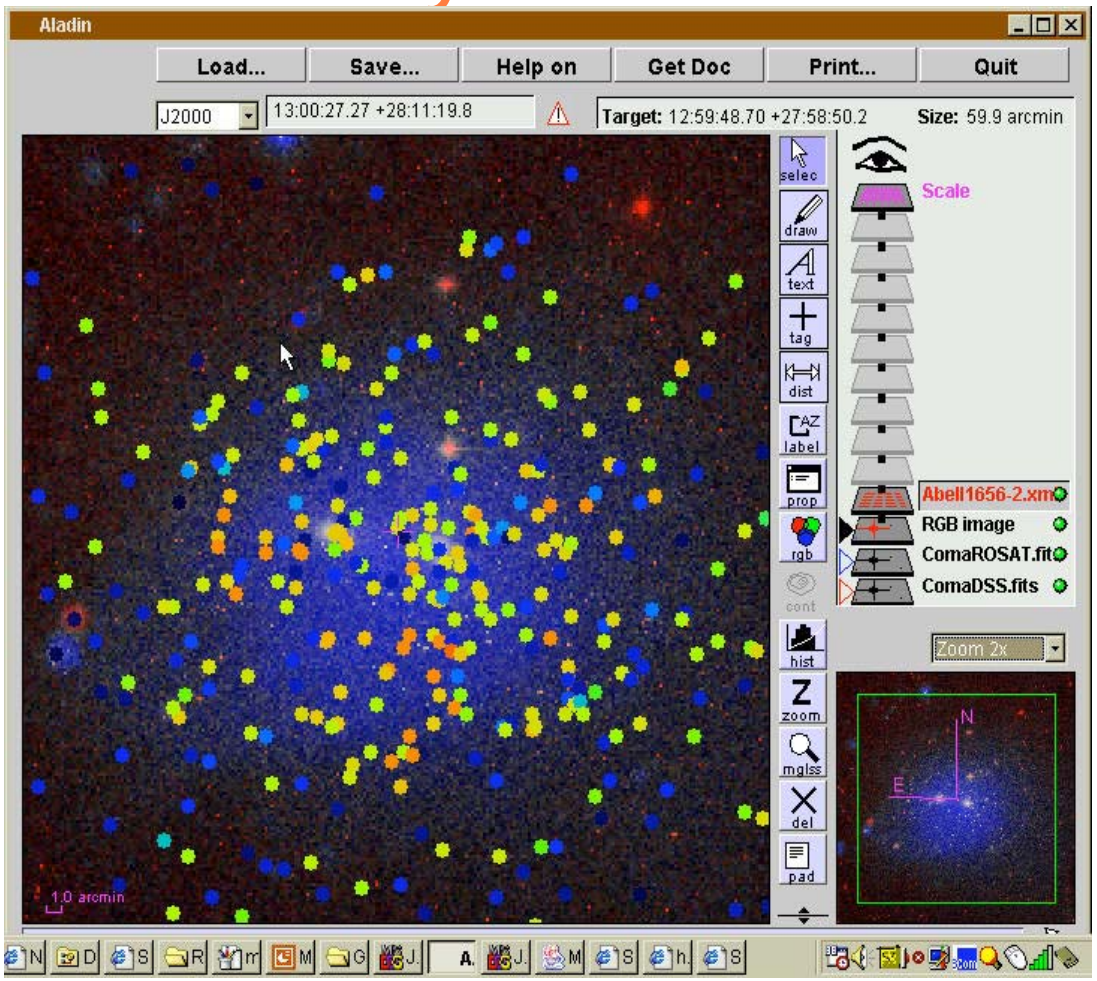
- SCEC is a collaboration of USC ISI, SDSC, USGS, and the Incorporated Research Institutions for Seismology
- Community Modeling Environment (SCEC/CME) automates selecting, configuring, and executing models of earthquake systems.
- Creates full 3D simulations of fault-system dynamics.
- Can assess and mitigating earthquake risks through Seismic Hazard Analysis



Work of Philip Maechling and Vipin Gupta, University Of Southern California

Astronomy

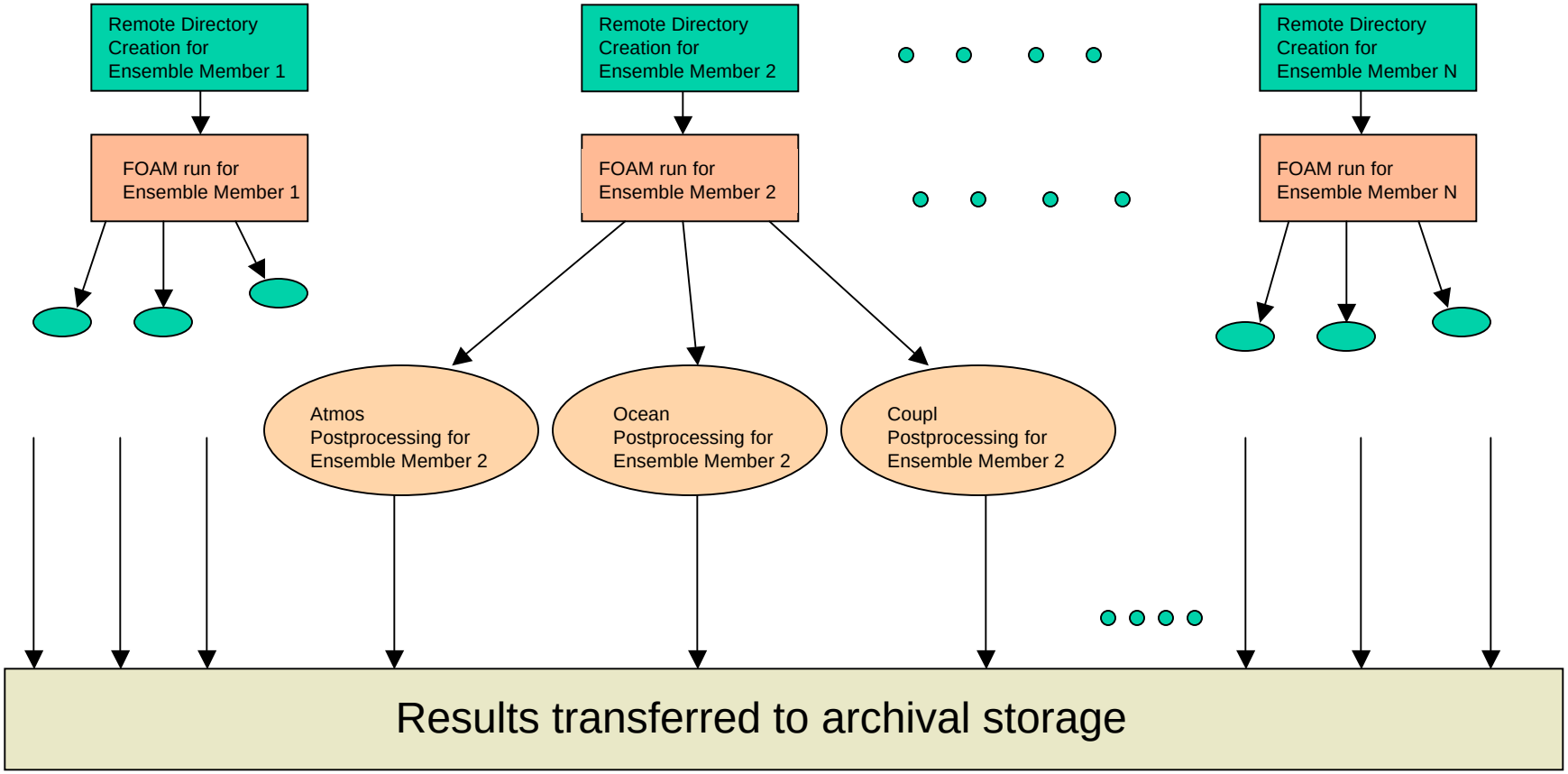
- Galaxy Morphology (National Virtual Observatory)
  - Investigates the dynamical state of galaxy clusters
  - Explores galaxy evolution inside the context of large-scale structure.
  - Uses galaxy morphologies as a probe of the star formation and stellar distribution history of the galaxies inside the clusters.
  - Data intensive computations involving hundreds of galaxies in a cluster



The x-ray emission is shown in blue, and the optical mission is in red. The colored dots are located at the positions of the galaxies within the cluster; the dot color represents the value of the asymmetry index. Blue dots represent the most asymmetric galaxies and are scattered throughout the image, while orange are the most symmetric, indicative of elliptical galaxies, are concentrated more toward the center.

People involved: Gurmeet Singh, Mei-Hui Su, many others

FOAM:
Fast Ocean/Atmosphere Model
250-Member Ensemble
Run on TeraGrid under VDS

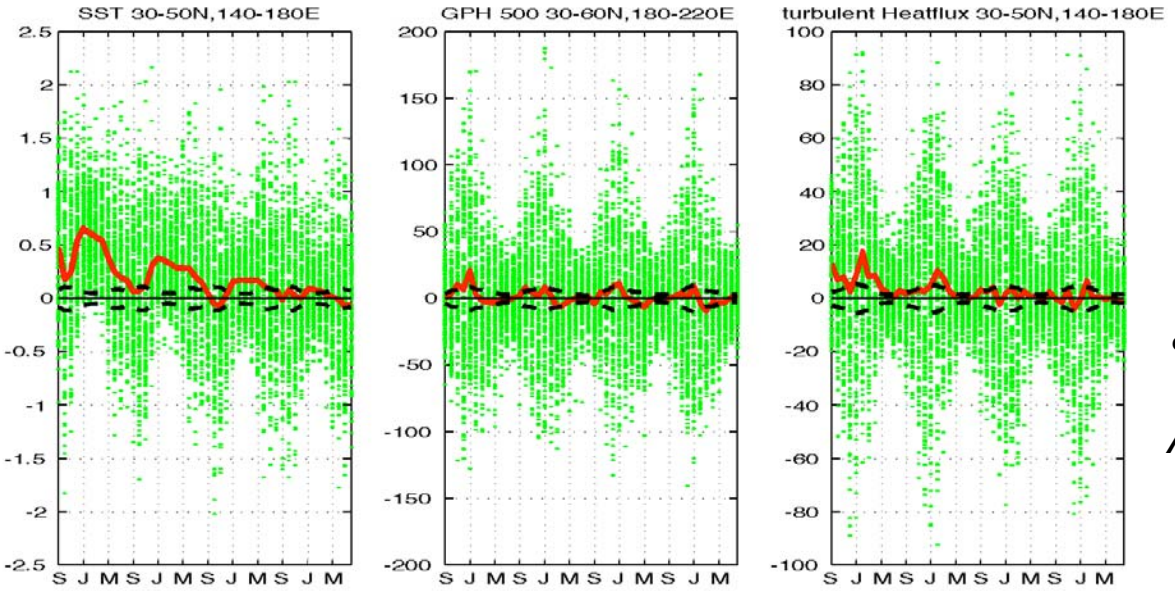


Work of: Rob Jacob (FOAM), Veronica Nefedova (Workflow design and execution)

FOAM: TeraGrid/VDSBenefits

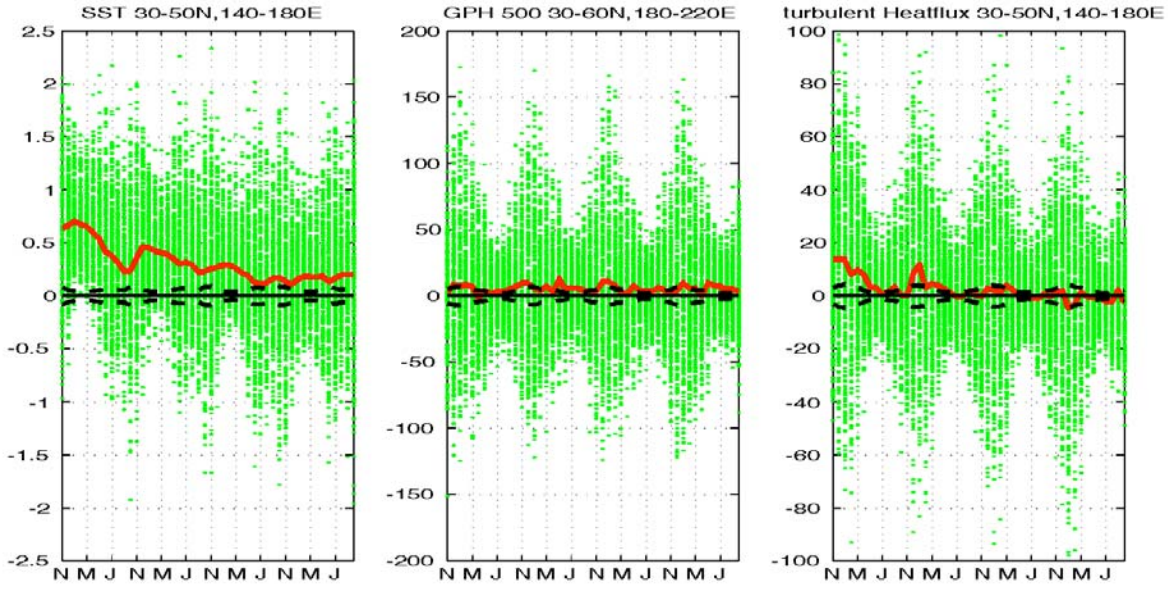
Climate Supercomputer

160 ensemble members = 2.5 months to run



TeraGrid with NMI and VDS

250 ensemble members = 4 days to run

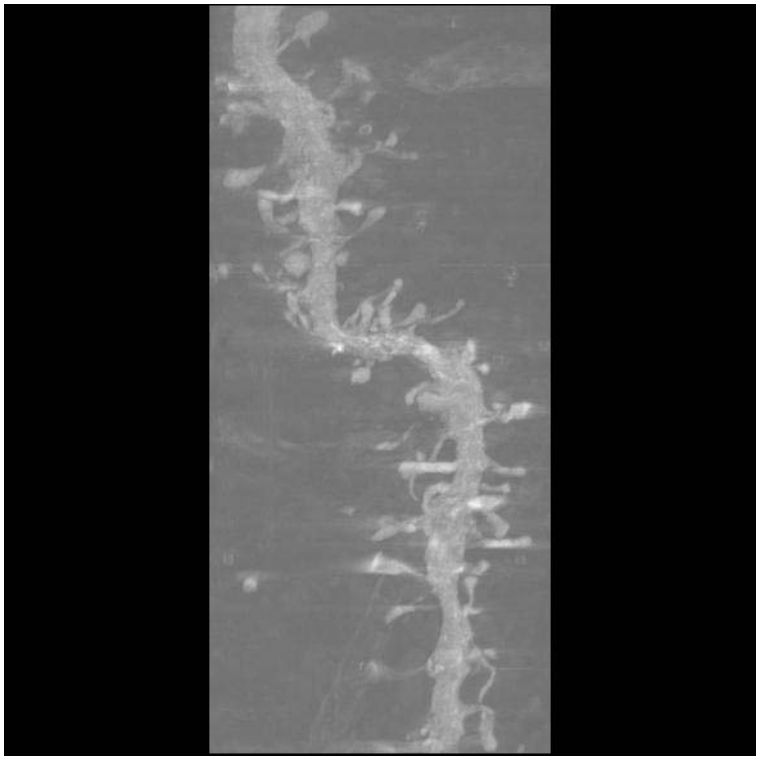


FOAM application by Rob Jacob, Argonne; VDS workflow by Veronika Nefedova, Argonne Visualization courtesy Pat Behling and Yun Liu, UW Madison..

Biomedical Imaging Applications

Tomography *(NIH-funded work)*

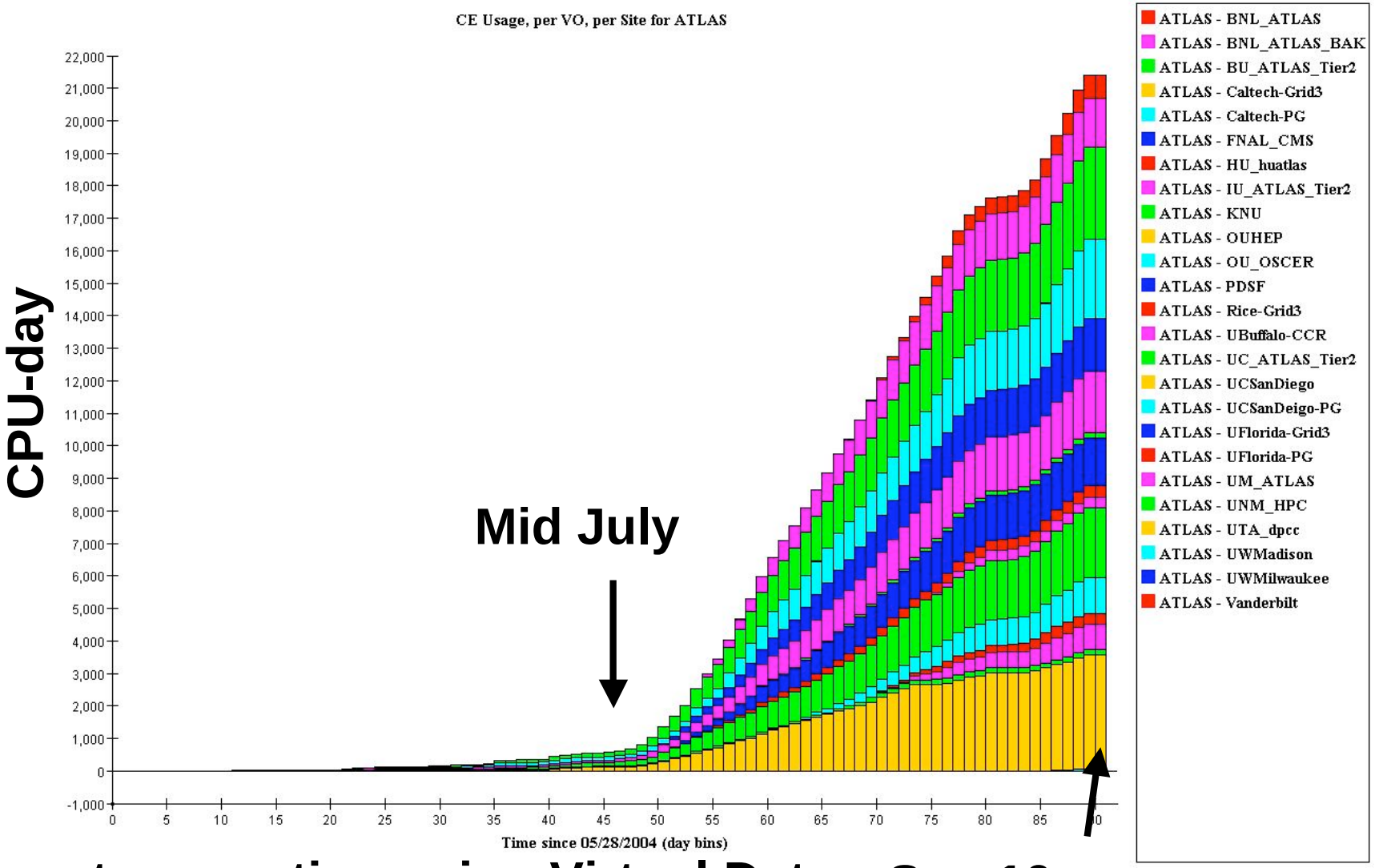
- Derivation of 3D structure from a series of 2D electron microscopic projection images,
- Reconstruction and detailed structural analysis
  - complex structures like synapses
  - large structures like dendritic spines.
- Acquisition and generation of huge amounts of data
- Large amount of state-of-the-art image processing required to segment structures from extraneous background.



Dendrite structure to be rendered by Tomography

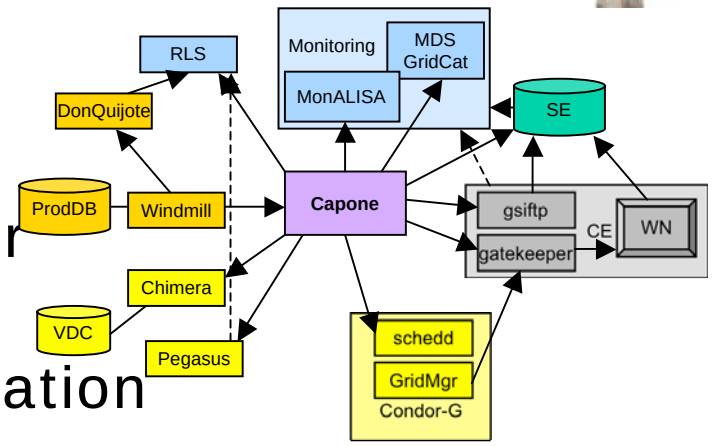
Work by Mei-Hui Su (ISI), Mark Ellisman, Steve Peltier, Abel Lin, Thomas Molina (SDSC)

US-ATLAS
Data Challenge 2



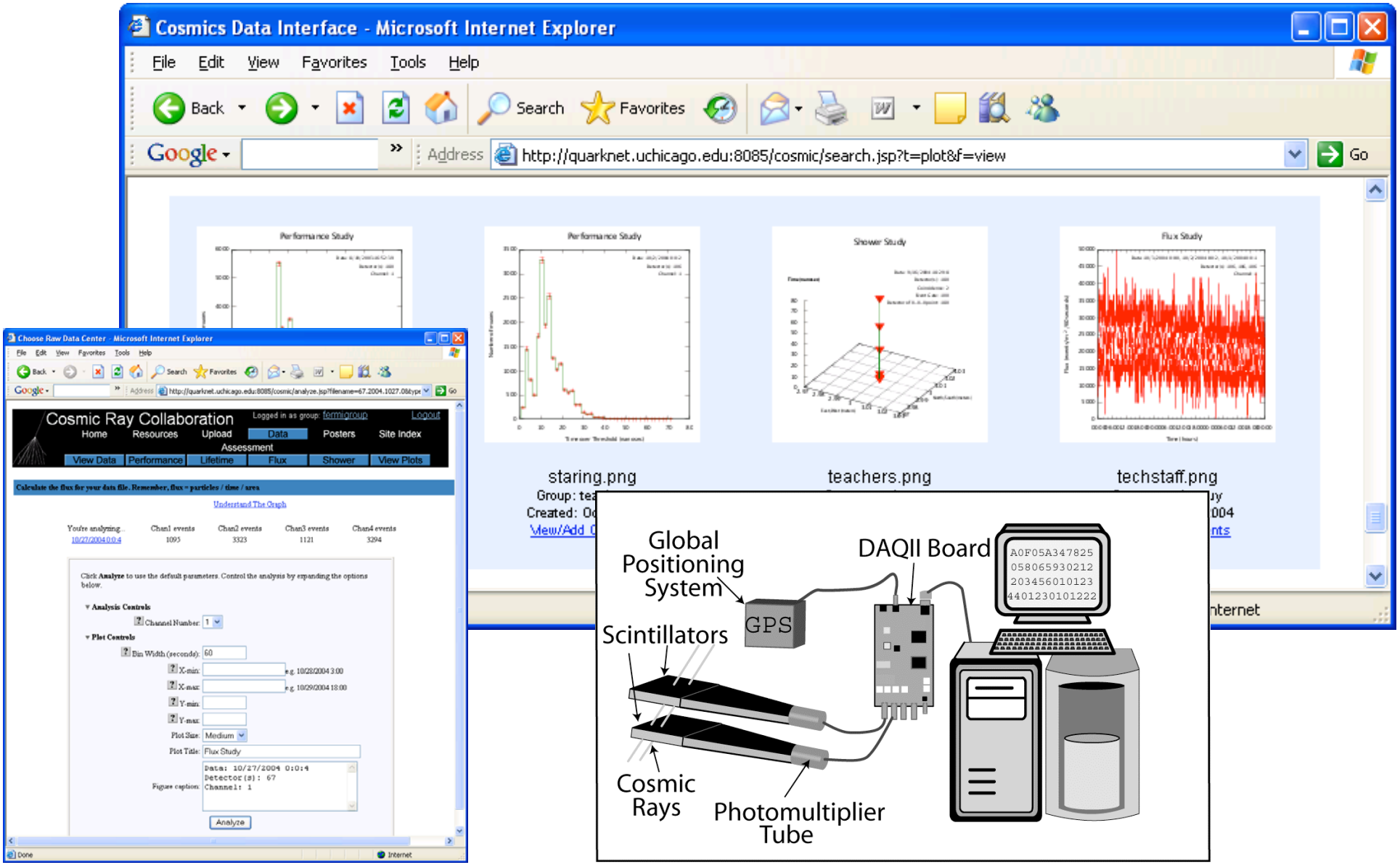
Event generation using Virtual Data Sep 10

ATLAS “Capone”
Event Simulation Production Executor



- Reception
  - Job received from work distributor
- Translation
  - Un-marshalling, ATLAS transformation
- DAX generation
  - VDL tools generate abstract DAG
- Input file retrieval from RLS catalog
  - Check RLS for input LFNs (retrieval of GUID, PFN)
- Scheduling: CE and SE are chosen
- Concrete DAG generation and submission
  - Pegasus creates Condor DAGman submit files

QuarkNet - Leveraging Virtual Data for Science Education



Cosmic Ray Data Analysis

a tutorial science application example

- Raw Datasets are time series files
  - File names of form:
    - > Data/180.2005.0126.1.raw
    - > Data/180.2005.0926.0.raw
- Set of processing applications in Perl
  - ThresholdTimes.pl
  - WireDelay.pl
  - Combine.pl
  - Sort.pl
  - Lifetime.pl
  - ExtraFunctions.pl
  - Plot.pl

Part 1 - Virtual Data Concept
Lab Exercises

10 Minutes

- Ex 1.1: Run the Cosmic Ray example science-code locally
- Ex 1.2: Hello World in VDL (local)

VDS Tutorial Outline

- Part 1: The concept of Virtual Data
- Part 2: Basics of VDL
- Part 3: Pegasus: Grid Workflow Planning
- Part 4 : VDL Details
- Summary and Conclusion

Virtual Data Process

- Describe data derivation or analysis steps in a high-level workflow language (VDL)
- VDL is cataloged in a database for sharing by the community
- Grid workflows are generated from VDL
- Provenance of derived results stored in database for assessment or verification

Essence of VDL

- Elevates specification of computation to a logical, location-independent level
- Acts as an “interface definition language” at the shell/application level
- Can express composition of functions
- Codable in textual and XML form
- Often machine-generated to provide ease of use and higher-level features
- Preprocessor provides iteration and variables

Expressing Workflow in VDL

```
TR grep (in a1, out a2) {
    argument stdin = ${a1};
    argument stdout = ${a2};
}

TR sort (in a1, out a2) {
    argument stdin = ${a1};
    argument stdout = ${a2};
}

DV grep (a1=@{in:file1}, a2=@{out:file2});
DV sort (a1=@{in:file2}, a2=@{out:file3});
```

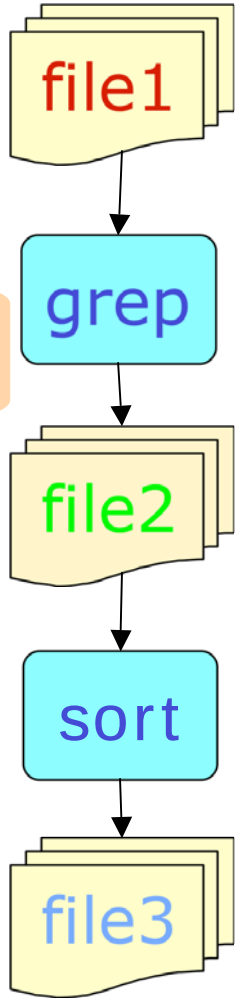
Define a "function" wrapper for an application

Define "formal arguments" for the application

Connect applications via output-to-input dependencies

Define a "call" to invoke application

Provide "actual" argument values for the invocation



Using VDL

- Generated directly for low-volume usage
- Generated by scripts for production use
- Generated by application tool builders as wrappers around scripts provided for community use
- Generated transparently in an application-specific portal (e.g. quarknet.fnal.gov/grid)
- Generated by drag-and-drop workflow design tools such as Triana

Terminology

- Virtual data
  - defining data by the *logical* workflow needed to create it *virtualizes* it with respect to location, existence, failure, and representation
- VDL – Virtual Data Language
  - A language (text and XML) that defines the functions and function calls of a workflow
- VDC – Virtual Data Catalog
  - The database and schema that store VDL definitions
- VDS – Virtual Data System
  - The tools to define, store, manipulate and execute virtual data workflows and query data provenance

Basic VDL Toolkit

- Convert between text and XML representation
- Insert, update, remove definitions from a virtual data catalog
- Attach metadata annotations to defintions
- Search for definitions
- Generate an abstract workflow for a data derivation request
- Multiple interface levels provided:
  - Java API, command line, web service

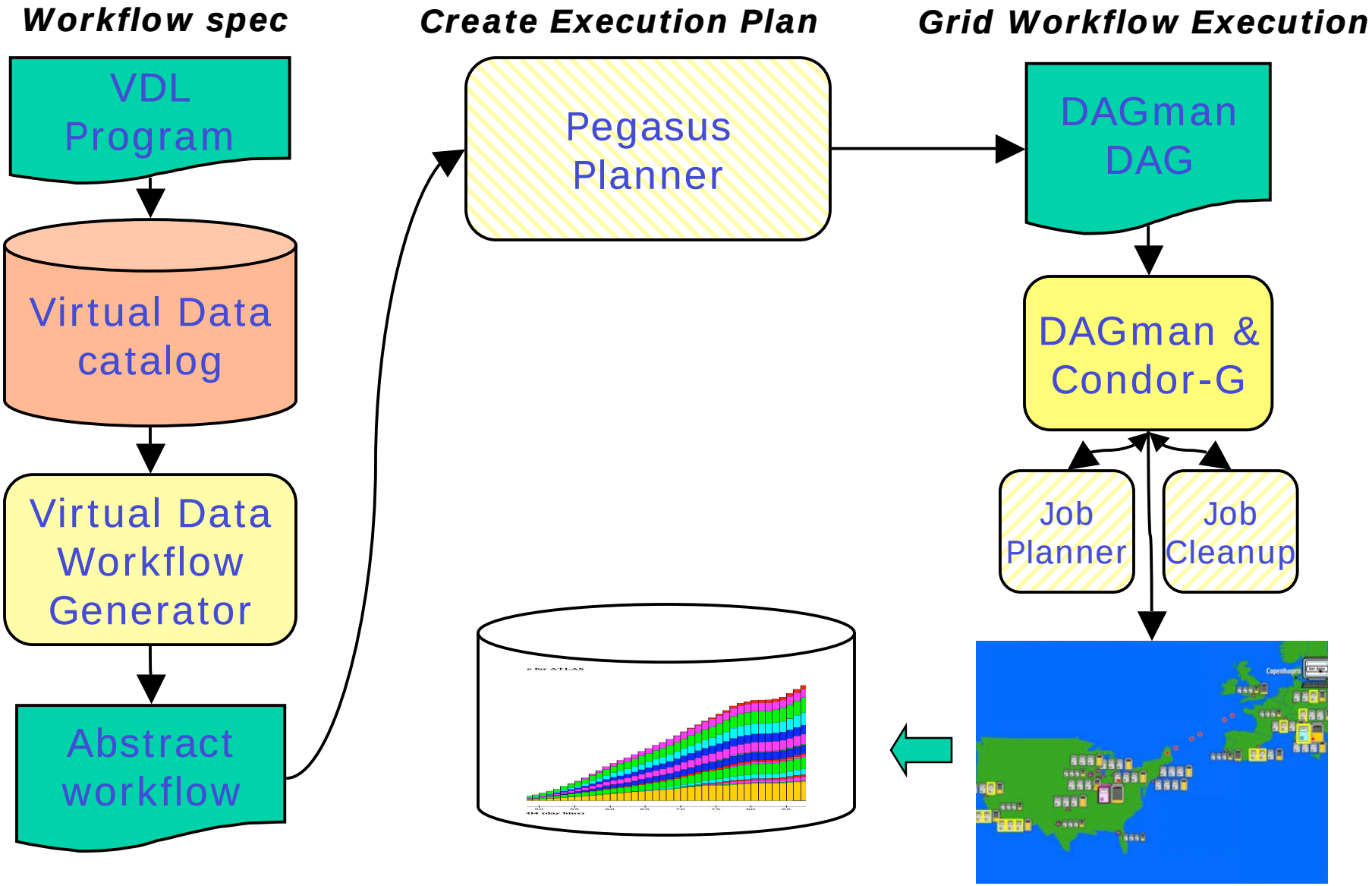
Representing Workflow

- Specifies a set of activities and control flow
- Sequences information transfer between activities
- VDS uses XML-based notation called “DAG in XML” (DAX) format
- VDC Represents a wide range of workflow possibilities
- DAX document represents steps to create a specific data product

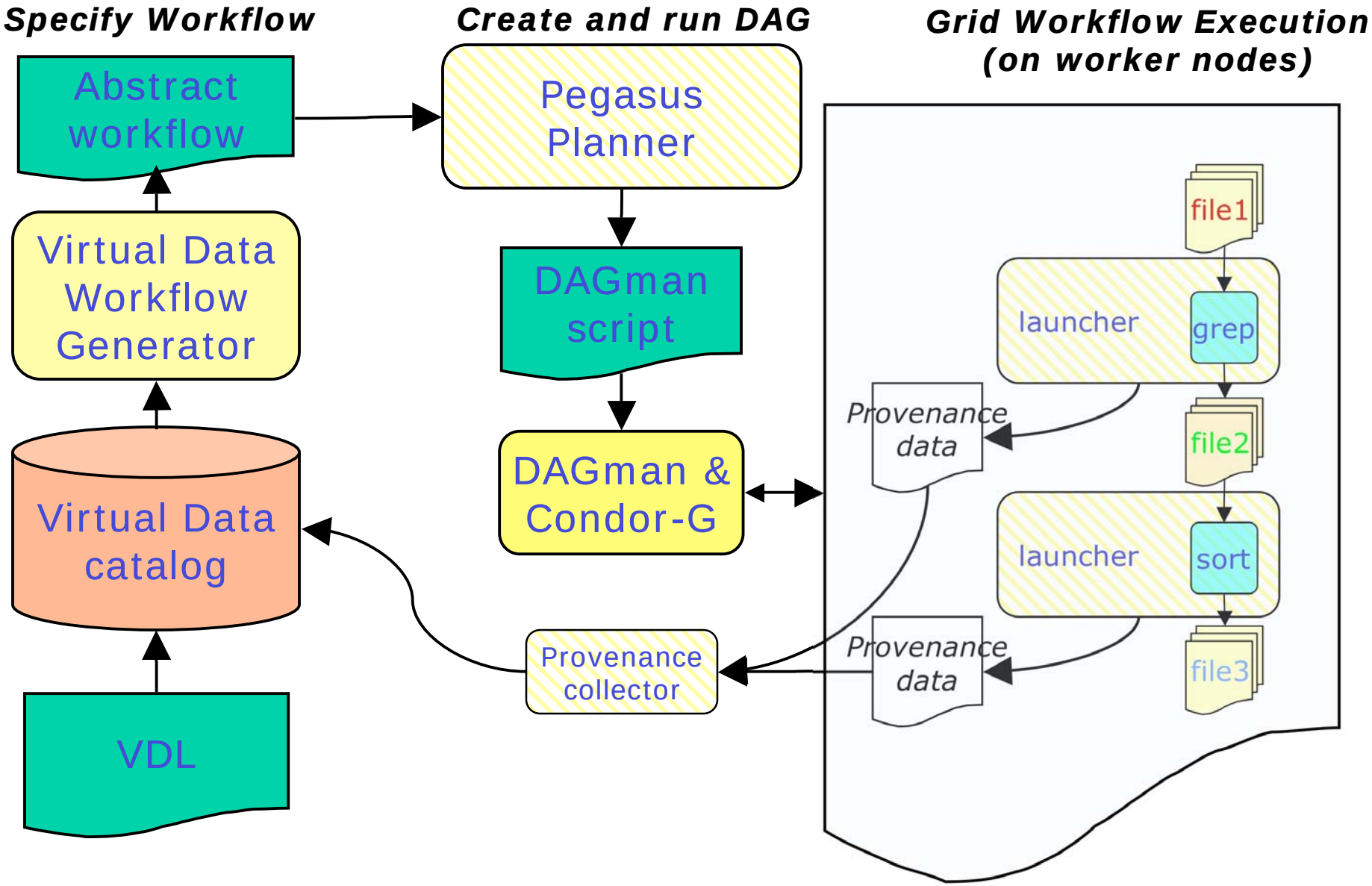
Abstract and Concrete Workflow

- Abstract Workflow (DAX)
  - Expressed in terms of logical entities
  - Specifies all logical files required to generate the desired data product from scratch
  - Dependencies between the jobs
  - Analogous to build style dag
- Concrete Workflow
  - Expressed in terms of physical entities
  - Specifies the location of the data and executables
  - Analogous to a make style dag

Executing VDL Workflows



Run Time Environment and Provenance Collection

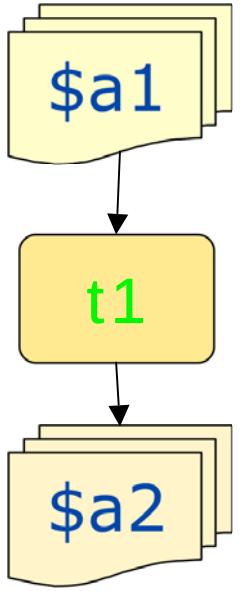


VDL: Virtual Data Language Describes Data Transformations

- Transformation - “TR”
  - Abstract template of program invocation
  - Similar to "function definition"
- Derivation – “DV”
  - “Function call” to a Transformation
  - Store past and future:
    - > A record of how data products were generated
    - > A recipe of how data products can be generated
- Invocation
  - Record of a Derivation execution

Example Transformation

```
TR t1( out a2, in a1, none pa = "500", none
   env = "100000" ) {
  argument = "-p "${pa};
  argument = "-f "${a1};
  argument = "-x –y";
  argument stdout = ${a2};
  profile env.MAXMEM = ${env};
}
```

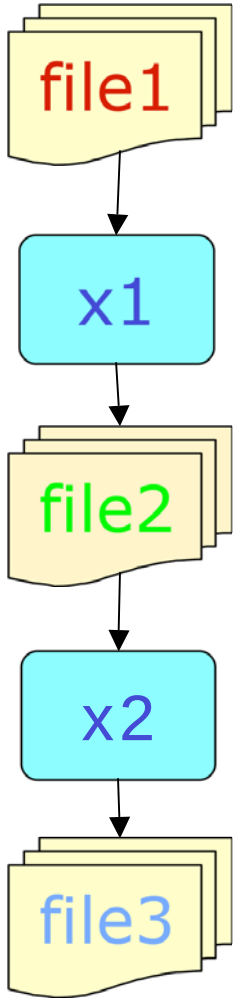


Example Derivations

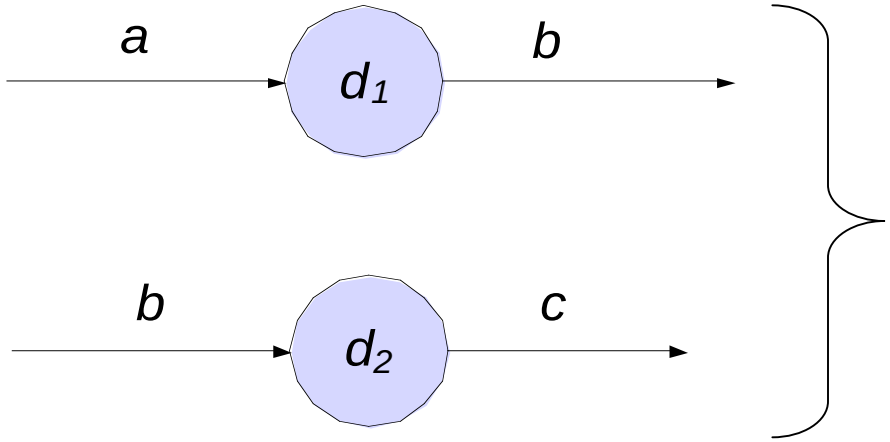
```
DV d1->t1 (
  env="20000", pa="600",
  a2=@{out:run1.exp15.T1932.summary},
  a1=@{in:run1.exp15.T1932.raw},
);

DV d2->t1 (
  a1=@{in:run1.exp16.T1918.raw},
  a2=@{out.run1.exp16.T1918.summary}
);
```

Workflow from File Dependencies

```
TR tr1(in a1, out a2) {
    argument stdin = ${a1};
    argument stdout = ${a2}; }

TR tr2(in a1, out a2) {
    argument stdin = ${a1};
    argument stdout = ${a2}; }

DV x1->tr1(a1=@{in:file1}, a2=@{out:file2});
DV x2->tr2(a1=@{in:file2}, a2=@{out:file3});
```



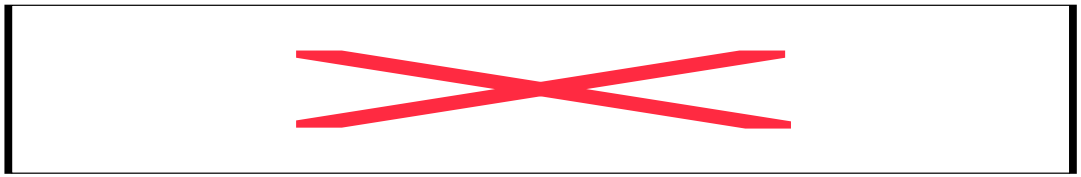
VDL and Abstract Workflow



VDL descriptions

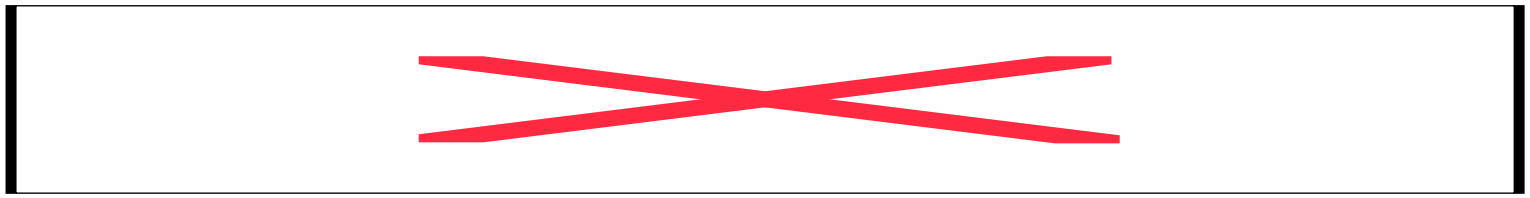
User request data file “c”

Abstract Workflow

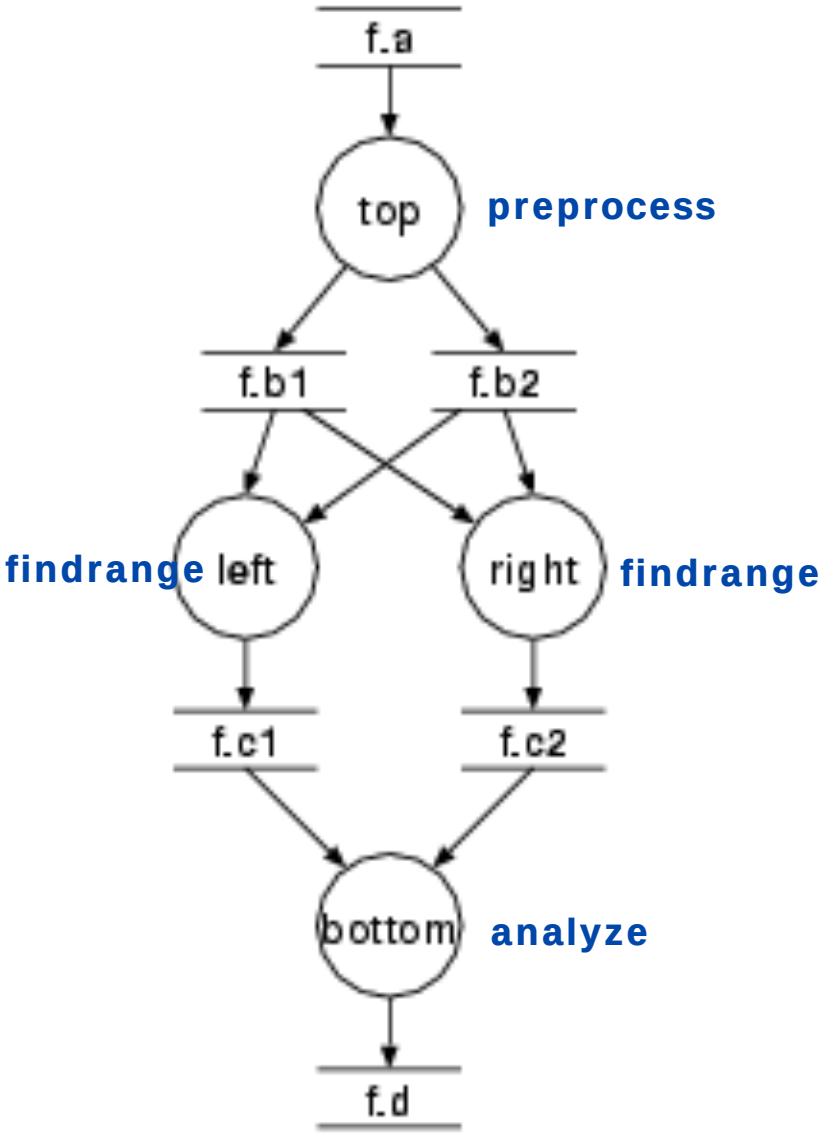


Executable Workflow Construction

- VDL tools used to build an abstract workflow based on VDL descriptions
- Planners (e.g. Pegasus) take the abstract workflow and produce an executable workflow for the Grid or other environments
- Workflow executors (“enactment engines”) like Condor DAGMan execute the workflow



Example Workflow



- Graph structure
  - Fan-in
  - Fan-out
  - "left" and "right" can run in parallel
- Uses input file
  - Register with RLS
- Complex file dependencies
  - Glues workflow

Workflow step "preprocess"

- TR preprocess turns f.a into f.b1 and f.b2

```
TR preprocess( output b[], input a ) {
  argument = "-a top";
  argument = " –i "${input:a};
  argument = " –o " ${output:b};
}
```

- Makes use of the "list" feature of VDL
  - Generates 0..N output files.
  - Number file files depend on the caller.

Workflow step "findrange"

- Turns two inputs into one output

```
TR findrange( output b, input a1, input a2,
  none name="findrange", none p="0.0" ) {
  argument = "-a "${name};
  argument = " –i " ${a1} " " ${a2};
  argument = " –o " ${b};
  argument = " –p " ${p};
}
```

- Uses the default argument feature

Can also use list[] parameters

```
TR findrange( output b, input a[],
  none name="findrange", none p="0.0" ) {
  argument = "-a "${name};
  argument = " –i " ${" "|a};
  argument = " –o " ${b};
  argument = " –p " ${p};
}
```

Complete VDL workflow

- Generate appropriate derivations

```
DV top->preprocess( b=[ @{out:"f.b1"}, @{
  out:"f.b2"} ], a=@{in:"f.a"} );
DV left->findrange( b=@{out:"f.c1"},
  a2=@{in:"f.b2"}, a1=@{in:"f.b1"},
  name="left", p="0.5" );
DV right->findrange( b=@{out:"f.c2"},
  a2=@{in:"f.b2"}, a1=@{in:"f.b1"},
  name="right" );
DV bottom->analyze( b=@{out:"f.d"}, a=[
  @{in:"f.c1"}, @{in:"f.c2"} );
```

Compound Transformations

- Using compound TR
  - Permits composition of complex TRs from basic ones
  - Calls are independent
    - unless linked through LFN
  - A Call is effectively an anonymous derivation
    - Late instantiation at workflow generation time
  - Permits bundling of repetitive workflows
  - Model: Function calls nested within a function definition

Compound Transformations (cont)

- TR diamond bundles black-diamonds

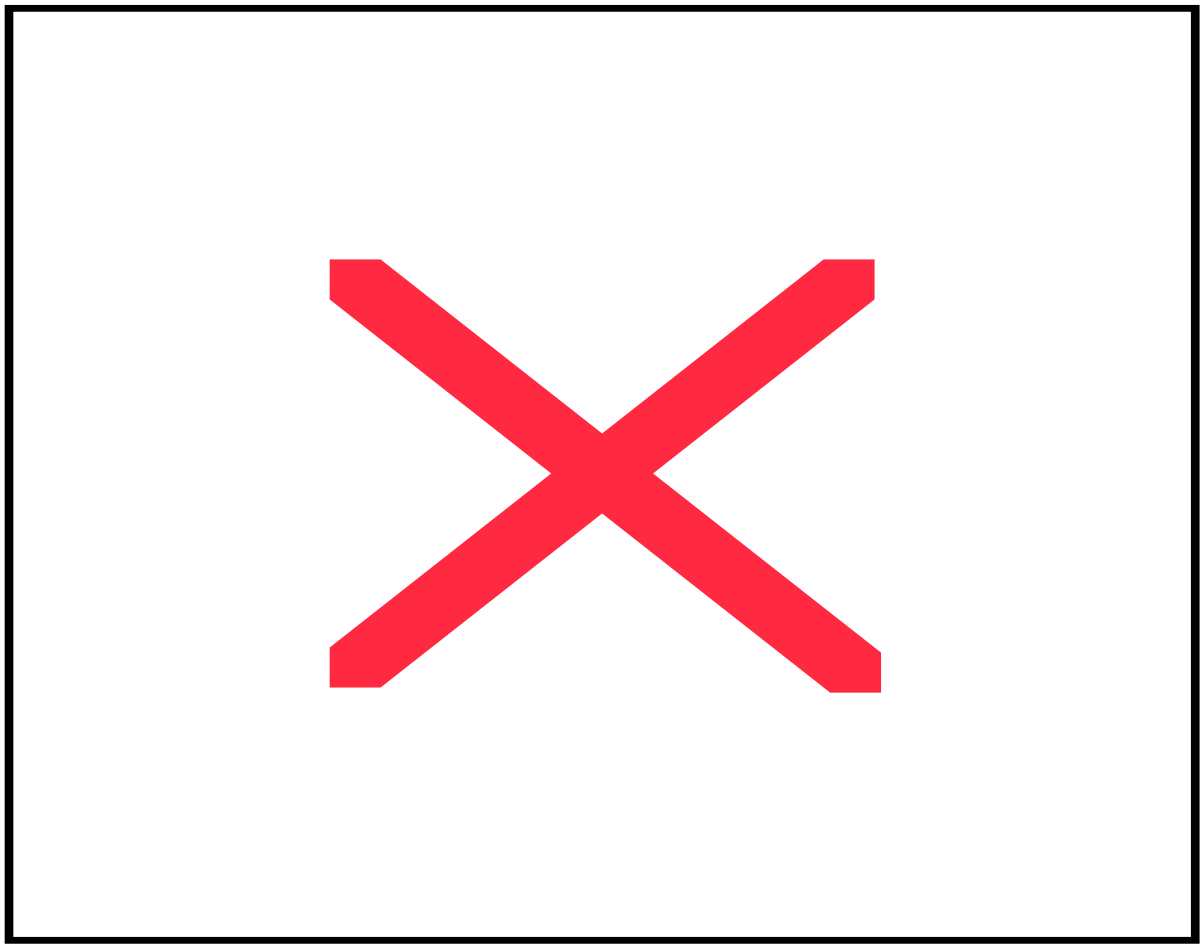
```
TR diamond( out fd, io fc1, io fc2, io fb1, io fb2, in fa, p1,
  p2 ) {
 call preprocess( a=${fa}, b=[ ${out:fb1}, ${out:fb2} ]
  );
 call findrange( a1=${in:fb1}, a2=${in:fb2},
  name="LEFT", p=${p1}, b=${out:fc1} );
 call findrange( a1=${in:fb1}, a2=${in:fb2},
  name="RIGHT", p=${p2}, b=${out:fc2} );
 call analyze( a=[ ${in:fc1}, ${in:fc2} ], b=${fd} );
}
```

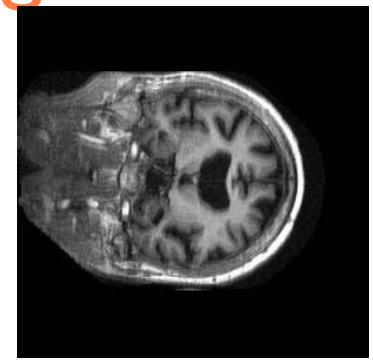
Compound Transformations (cont)

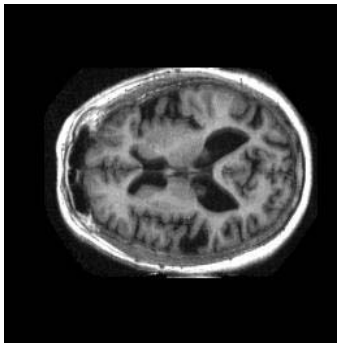
- Multiple DVs allow easy generator scripts:

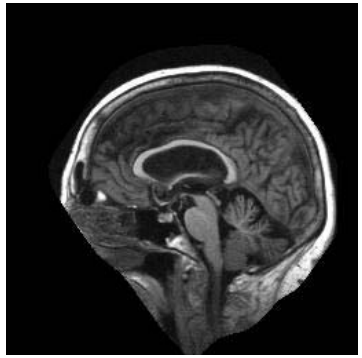
```
DV d1->diamond( fd=@{out:"f.00005"},
   fc1=@{io:"f.00004"}, fc2=@{io:"f.00003"},
   fb1=@{io:"f.00002"}, fb2=@{io:"f.00001"},
   fa=@{io:"f.00000"}, p2="100", p1="0" );
DV d2->diamond( fd=@{out:"f.0000B"},
   fc1=@{io:"f.0000A"}, fc2=@{io:"f.00009"},
   fb1=@{io:"f.00008"}, fb2=@{io:"f.00007"},
   fa=@{io:"f.00006"}, p2="141.42135623731", p1="0" );
...
DV d70->diamond( fd=@{out:"f.001A3"},
   fc1=@{io:"f.001A2"}, fc2=@{io:"f.001A1"},
   fb1=@{io:"f.001A0"}, fb2=@{io:"f.0019F"},
   fa=@{io:"f.0019E"}, p2="800", p1="18" );
```

Functional MRI Analysis









fMRI Example: AIR Tools

```
TR air::warp_n_slice(
        in reg_img, in reg_hdr, in sub_img, in sub_hdr, m = "12",
        io warp, sliced, out sliced_img, out sliced_hdr )
{
    call air::align_warp( reg_img=${reg_img}, reg_hdr=${reg_hdr},
                          sub_img=${sub_img}, sub_hdr=${sub_hdr},
                          m=${m},
                          warp = ${out:warp} );
    call air::reslice( warp=${in:warp}, sliced=${sliced},
        sliced_img=${sliced_img}, sliced_hdr=${sliced_hdr} );
}
TR air::softmean( in sliced_img[], in sliced_hdr[], arg1 = "y",
        arg2 = "null", atlas, out atlas_img, out atlas_hdr )
{
    argument = ${atlas};
    argument = ${arg1} " " ${arg2};
    argument = ${sliced_img};
}
```

fMRI Example: AIR Tools

```
DV air::a3472_3->air::softmean(
    sliced_img = [
        @{in:"3472-3_anonymized.sliced.img"},
        @{in:"3472-4_anonymized.sliced.img"},
        @{in:"3472-5_anonymized.sliced.img"},
        @{in:"3472-6_anonymized.sliced.img"} ],
    sliced_hdr = [
        @{in:"3472-3_anonymized.sliced.hdr"},
        @{in:"3472-4_anonymized.sliced.hdr"},
        @{in:"3472-5_anonymized.sliced.hdr"},
        @{in:"3472-6_anonymized.sliced.hdr"} ],
    atlas = "atlas",
    atlas_img = @{out:"atlas.img"},
    atlas_hdr = @{out:"atlas.hdr"}
);
```

“DAX” Abstract Workflow

list of jobs and files

```
<job id="ID000001" namespace="Quarknet.HEPSRCH" name="ECalEnergySum" level="5"
                   dv-namespace="Quarknet.HEPSRCH"  dv-name="run1aesum">
 <argument><filename file="run1a.event"/> <filename file="run1a.esm"/></argument>
 <uses file="run1a.esm" link="output" dontRegister="false" dontTransfer="false"/>
 <uses file="run1a.event" link="input" dontRegister="false“ dontTransfer="false"/>
</job>
...
<job id="ID000014" namespace="Quarknet.HEPSRCH" name="ReconTotalEnergy" level="3"…
 <argument><filename file="run1a.mis"/> <filename file="run1a.ecal"/> …
 <uses file="run1a.muon" link="input" dontRegister="false" dontTransfer="false"/>
 <uses file="run1a.total" link="output" dontRegister="false" dontTransfer="false"/>
 <uses file="run1a.ecal" link="input" dontRegister="false" dontTransfer="false"/>
 <uses file="run1a.hcal" link="input" dontRegister="false" dontTransfer="false"/>
 <uses file="run1a.mis" link="input" dontRegister="false" dontTransfer="false"/>
</job>

<!--list of all files used -->
 <filename file="ecal.pct" link="inout"/>
 <filename file="electron10GeV.avg" link="inout"/>
 <filename file="electron10GeV.sum" link="inout"/>
 <filename file="hcal.pct" link="inout"/>
 ...
```

(excerpted for display)

“DAX” Abstract Workflow

control flow graph

```
<child ref="ID000003">    <parent ref="ID000002"/>   </child>
<child ref="ID000004">    <parent ref="ID000003"/>   </child>
<child ref="ID000005">    <parent ref="ID000004"/>
                          <parent ref="ID000001"/>...
<child ref="ID000009">    <parent ref="ID000008"/>   </child>
<child ref="ID000010">    <parent ref="ID000009"/>
                          <parent ref="ID000006"/>...
<child ref="ID000012">    <parent ref="ID000011"/>   </child>
<child ref="ID000013">    <parent ref="ID000011"/>   </child>
<child ref="ID000014">    <parent ref="ID000010"/>
                          <parent ref="ID000012"/>...
                          <parent ref="ID000013"/>...</child>…
```

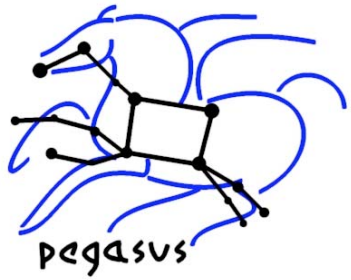
(excerpted for display…)

Part 2 - VDL
Lab Exercises

30 Minutes

- Ex 2.1: Running Single Transformations (simplevdl)
- Ex 2.2: Chaining Derivations (biggervdl)
- Ex 2.3: Compound VDL (compoundvdl)

VDS Tutorial Outline

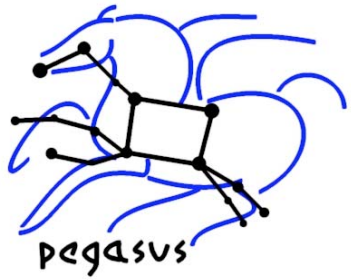


- Part 1: Concept & applications of Virtual Data
- Part 2: VDL – The Virtual Data Language
- Part 3: Pegasus – Grid Workflow Planning
- Part 4 : VDS in the Science Process
- Summary and Conclusion

What we're going to do...

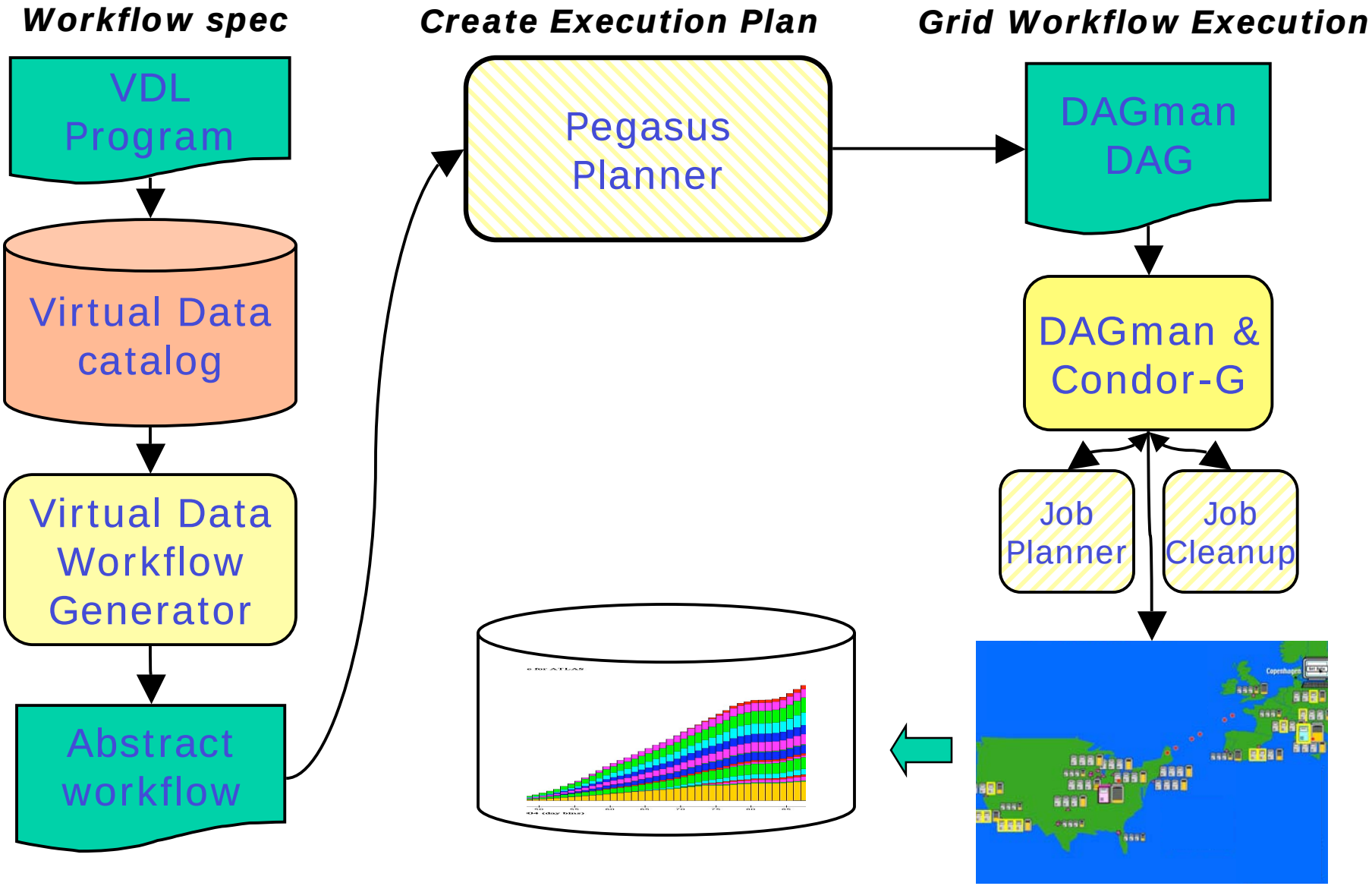
- Learn how to run workflows on the grid.
- What do we need for it ?
  - An abstract workflow description (e.g QuarkNet Workflow)
  - Application executables (QuarkNet, here)
  - A Planner to generate a concrete workflow (like "compiled code" for the grid).
  - ...and, of course, a Grid (TeraGrid, here)

Pegasus Section Outline

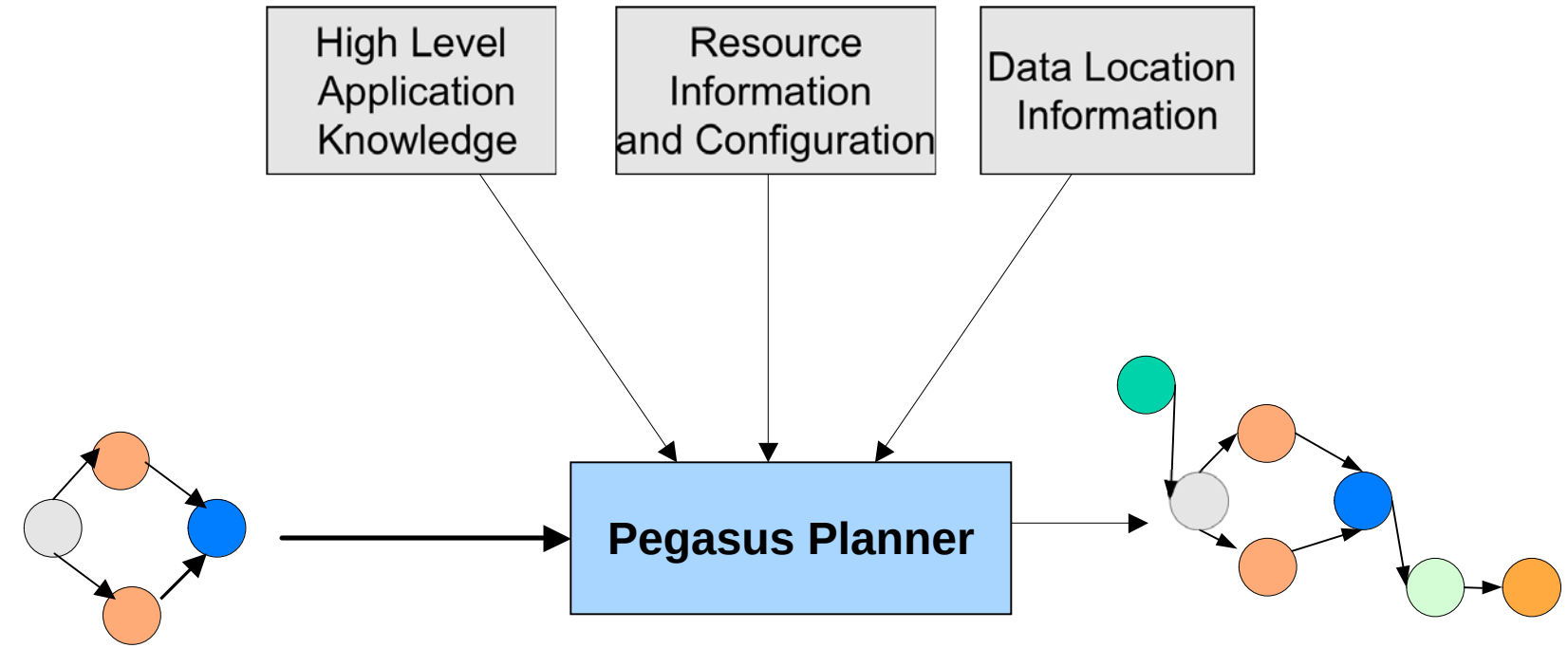


- Planning with Pegasus
  - Catalogs used By Pegasus
  - Pegasus Components
- Executing workflows on the Grid
  - Debugging Workflows
- Pegasus Advanced Features and Optimizations
- Further Reading

Executing VDL Workflows



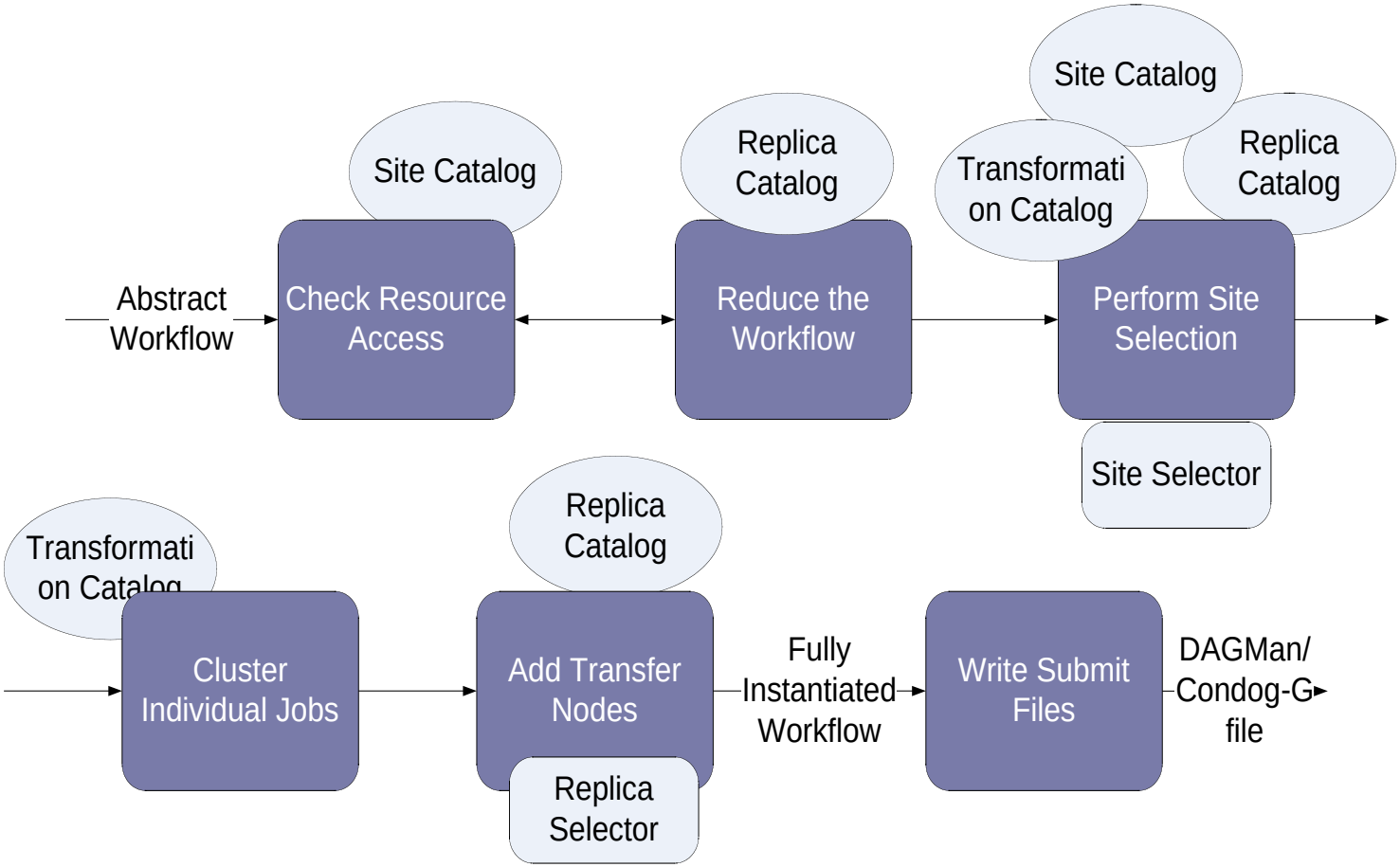
Planning with Pegasus



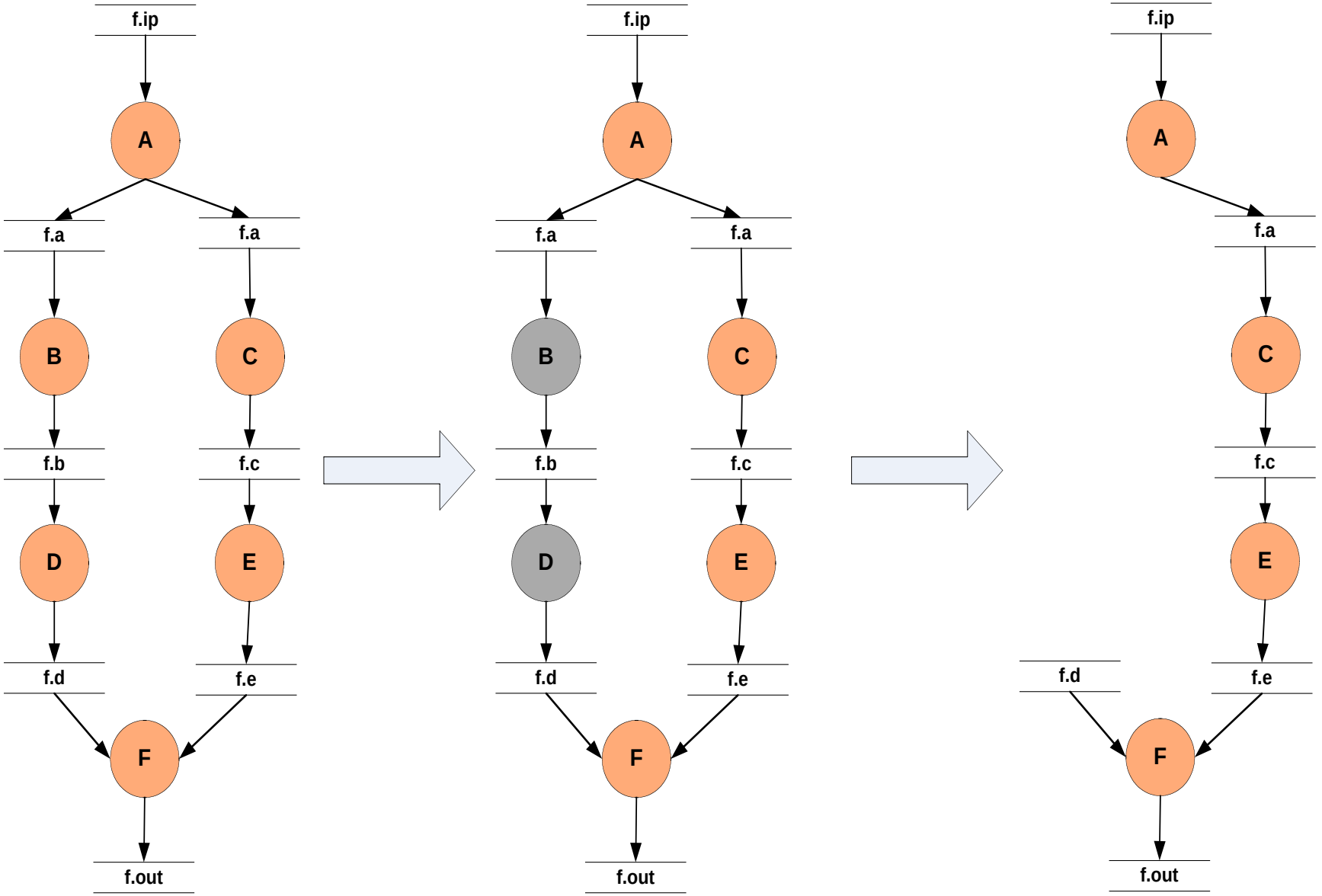
Abstract Workflow (DAX) generated from VDL specification

Plan to be submitted to the grid (e.g condor submit files)

Refinement Pipeline



Abstract to Concrete, Step 1: Workflow Reduction

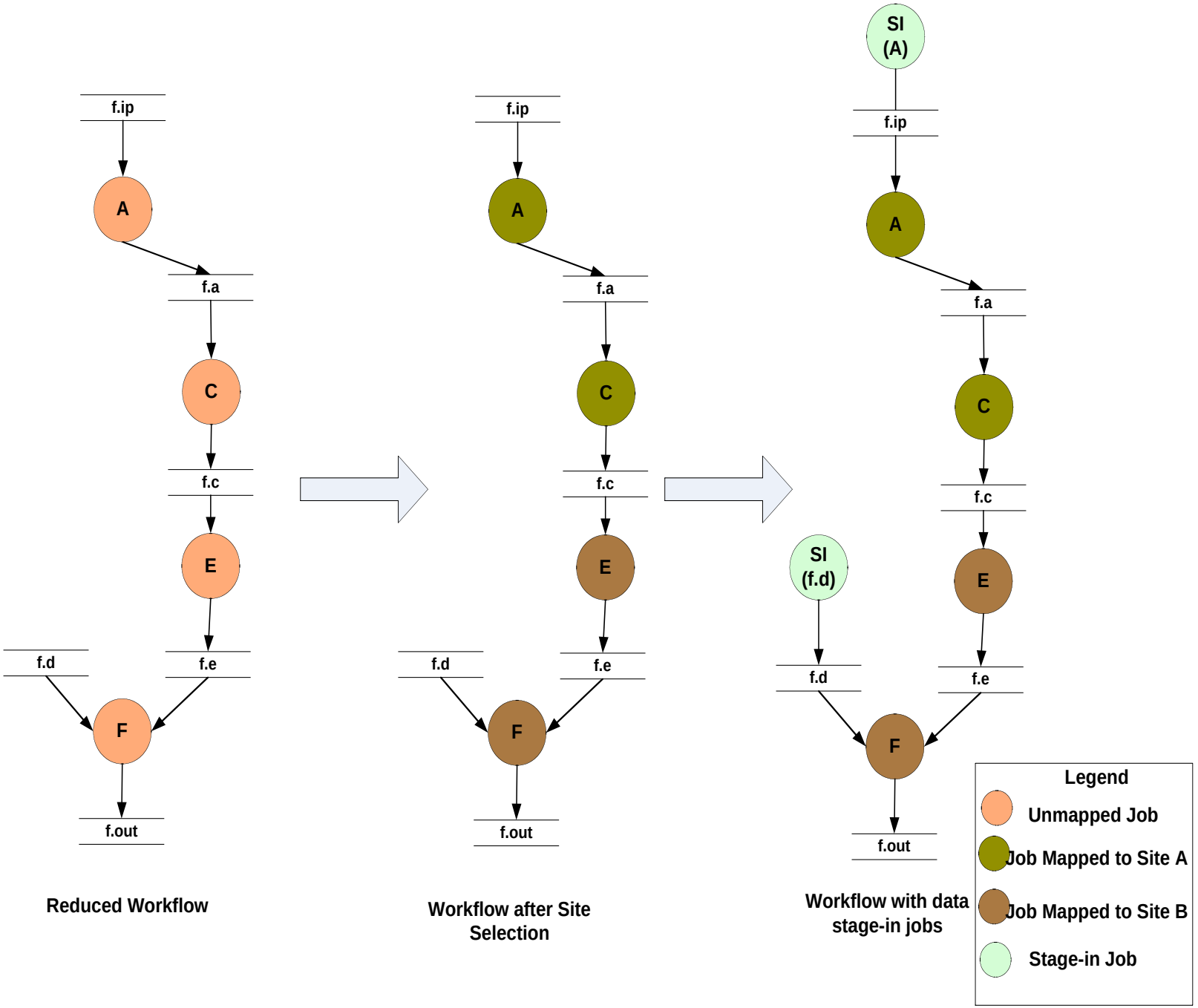


Abstract Workflow

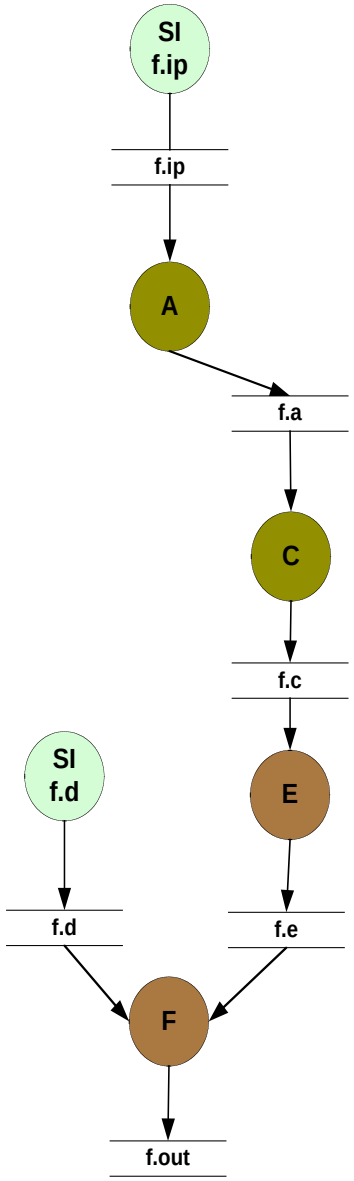
File f.d exists somewhere.
Reuse it.
Mark Jobs D and B to delete

Delete Job D and Job B

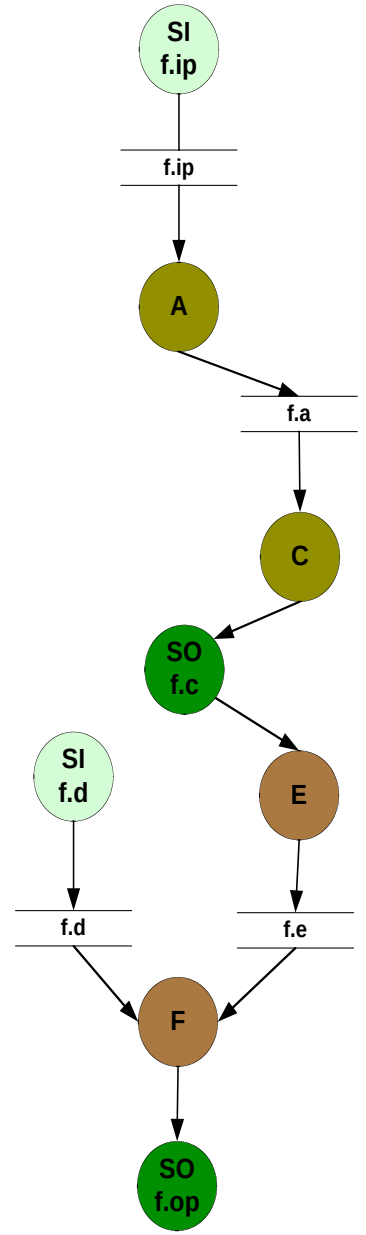
Step 2: Site Selection & Addition of Data Stage-in Nodes



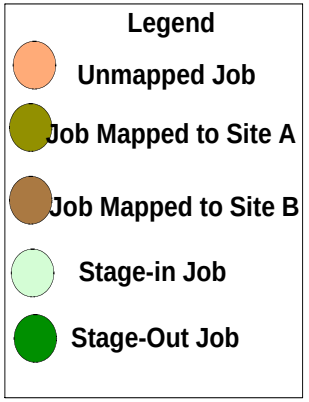
Step 3: Addition of Data Stage-out Nodes



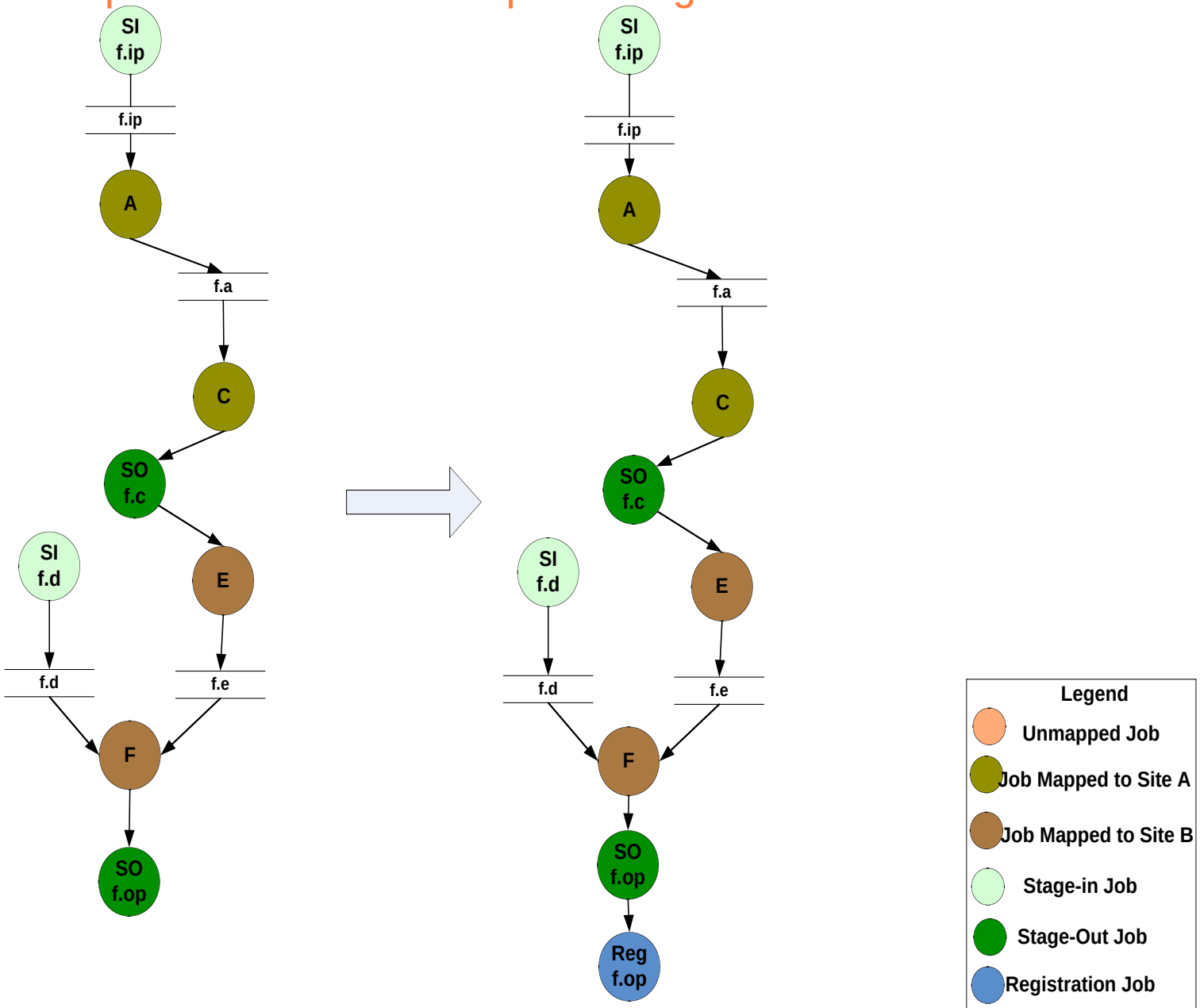
Workflow with Data Stage in jobs



Workflow with Data Stage out jobs to final output site



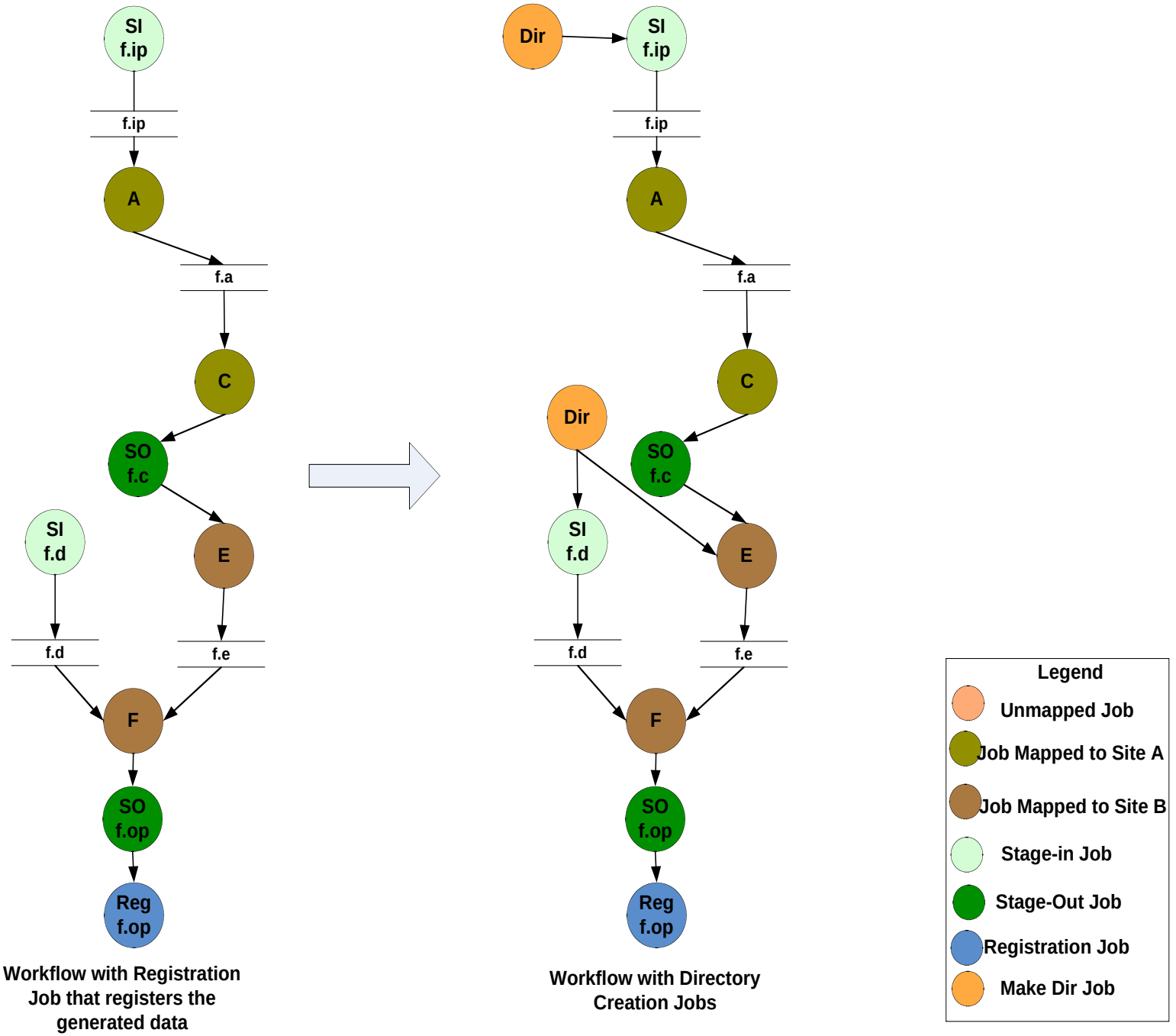
Step 4: Addition of Replica Registration Jobs



Workflow with Data Stage out Jobs to final output site

Workflow with Registration Job that registers the generated data

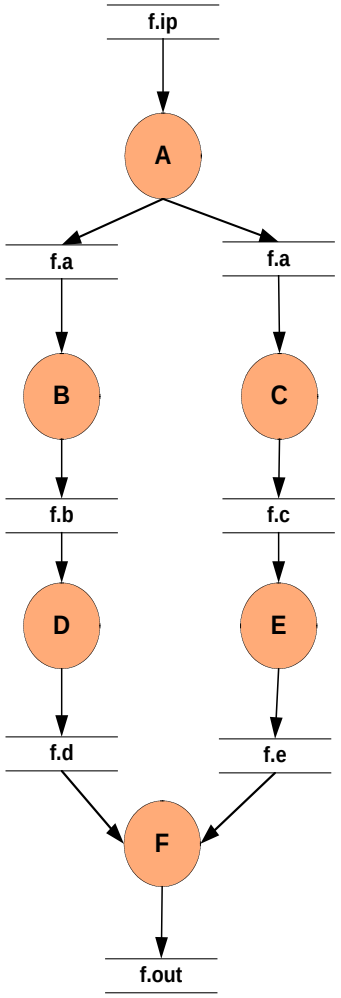
Step 5: Addition of Job-Directory Creation



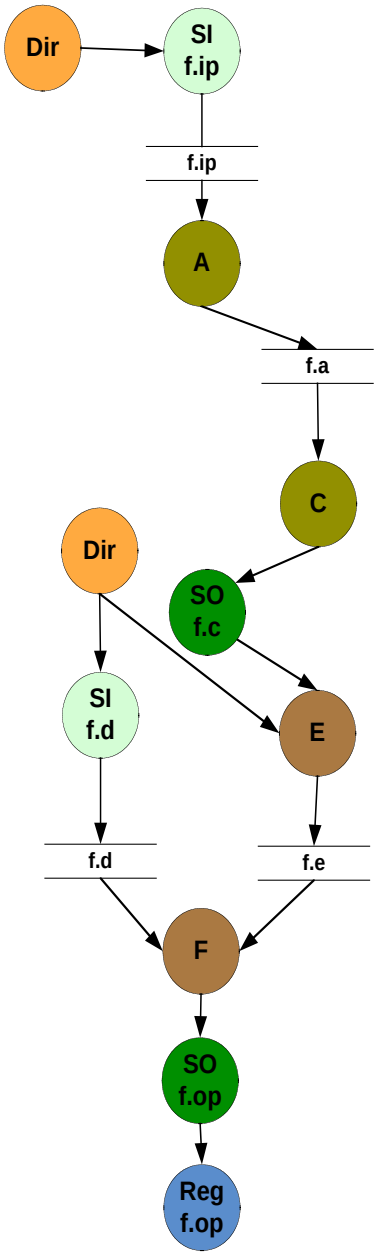
Workflow with Registration Job that registers the generated data

Workflow with Directory Creation Jobs

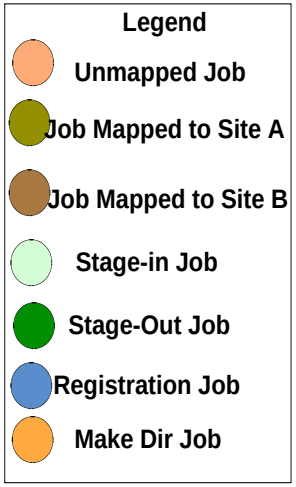
Final Result of Abstract-to-Concrete Process



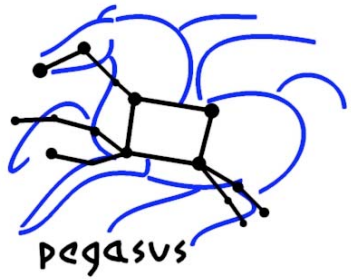
Abstract Workflow



Final Concrete Workflow



Pegasus Section Outline



- Planning with Pegasus
  - Catalogs used By Pegasus
  - Pegasus Components
- Executing workflows on the Grid
  - Debugging Workflows
- Pegasus Advanced Features and Optimizations
- Further Reading

Replica Catalog

- Data is replicated for scalability, reliability and availability
- Replica Catalog stores mappings between logical files and their target locations.
- Pegasus uses RLS as a replica catalog to locate already existing data (raw input data and data products generated from previous runs).

Replica Catalog Exercise (Ex. 3.1 10 minutes)

- The VDS rc-client is a command line tool to interact with RLS.

- Practical exercise (Refer Exercise 3.1):
  - Use the rc-client to
    - > Populate the RLS
    - > Query the RLS
    - > Remove entries (Offline exercise)

Site Catalog

- Contains information about various sites on which workflows may execute.
- For each site following information is stored
  - Installed job-managers for different types of schedulers
  - Installed GridFTP servers
  - Local Replica Catalogs where data residing in that site has to be catalogued
  - Site Wide Profiles like environment variables
  - Work and storage directories

Site Catalog Exercise (Ex 3.1 10 minutes)

- Two clients for generating a site catalog
- vds-get-sites
  - Allows you to generate a site catalog
    - for OSG grid sites by querying GridCAT
    - for ISI skynet, Teragrid, UC SofaGrid by querying a SQLLite2 database
- genpoolconfig
  - Allows you to generate a site catalog
    - By specifying information about a site in a textual format in a file.
    - One file per site

Site Catalog Entry

- <pool handle="**isi_skynet**" sysinfo="**INTEL32::LINUX**" gridlaunch="**/nfs/software/vds/vds/bin/kickstart**">
  <profile namespace="**vds**" key="**grid**">**isi**</profile>
  <lrc url="**rlsn://smarty.isi.edu**" />
  <gridftp url="**gsiftp://skynet-data.isi.edu**" storage="**/nfs/storage01**" major="**2**" minor="**4**" patch="**3**" />
  <gridftp url="**gsiftp://skynet-2.isi.edu**" storage="**/nfs/storage01**" major="**2**" minor="**4**" patch="**3**" />
  <jobmanager universe="**vanilla**" url="**skynet-login.isi.edu/jobmanager-pbs**" major="**2**" minor="**4**" patch="**3**" total-nodes="**93**" />
  <jobmanager universe="**transfer**" url="**skynet-login.isi.edu/jobmanager-fork**" major="**2**" minor="**4**" patch="**3**" total-nodes="**93**" />
  <workdirectory>**/nfs/scratch01**</workdirectory>
  </pool>

Transformation Catalog

- Transformation Catalog maps logical transformations to their physical locations

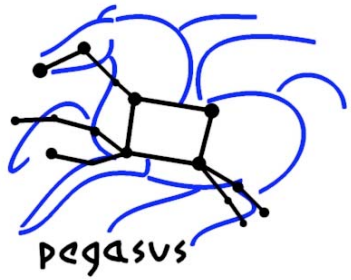
- For each transformation following are stored
  - logical name of the transformation
  - Type of transformation (INSTALLED or STATIC_BINARY)
  - Architecture, OS, Glibc version
  - the resource on the which the transformation is available
  - the URL for the physical transformation
  - Profiles that associate runtime parameters like environment variables, scheduler related information

- Also the logical name of the transformation is same as the ones specified in the VDL . The same name also appears in the DAX

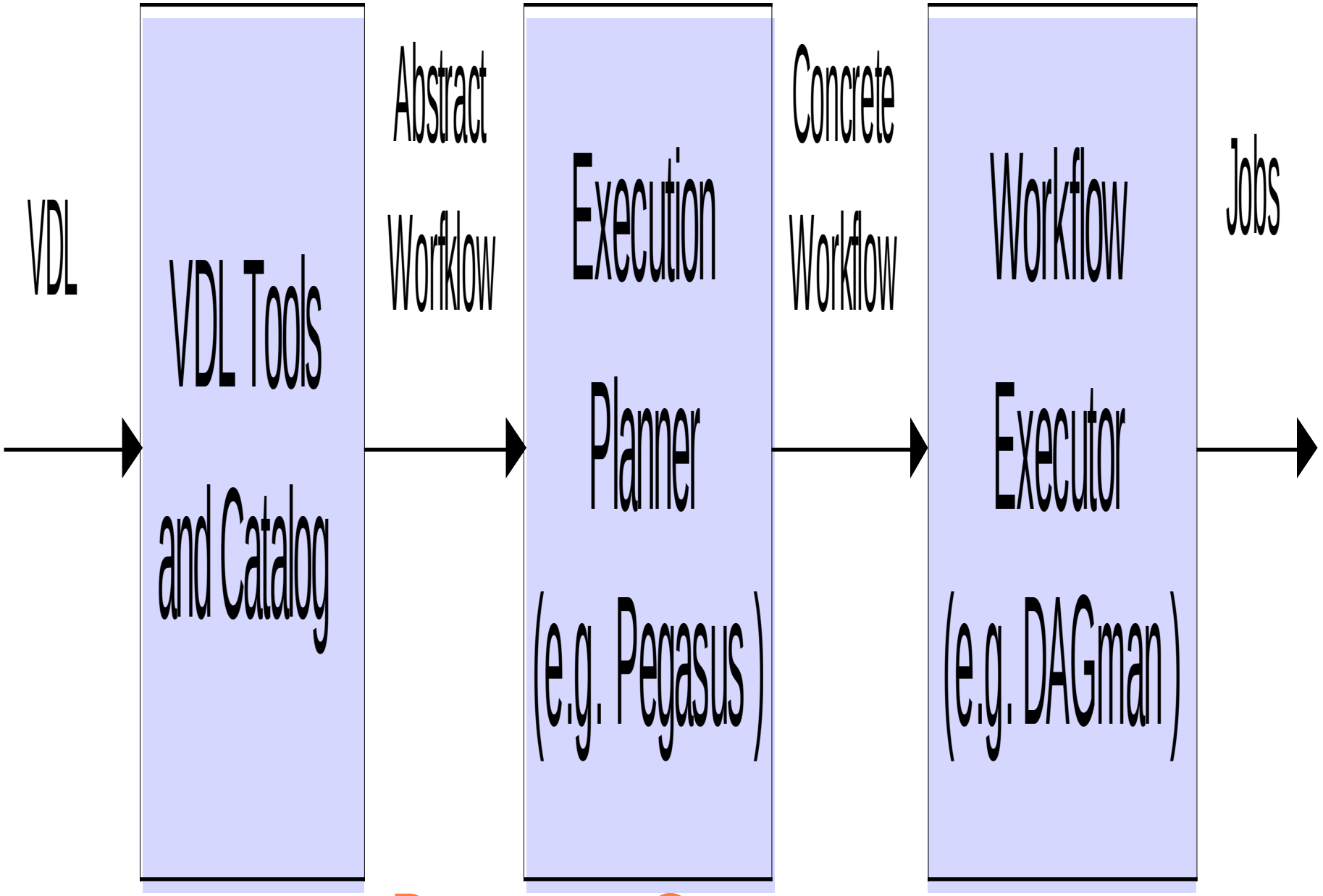
Transformation Catalog Exercise (Offline)

- tc-client is a command line client that is primarily used to configure the database TC
- Works even for file based transformation catalog.

Pegasus Section Outline



- Planning with Pegasus
  - Catalogs used By Pegasus
  - Pegasus Components
- Executing workflows on the Grid
  - Debugging Workflows
- Pegasus Advanced Features and Optimizations
- Further Reading

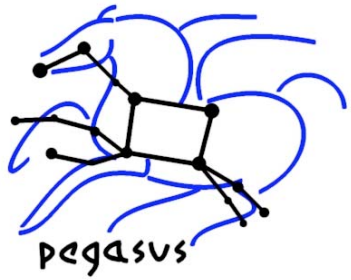


Pegasus Components

Component Configuration using Properties File

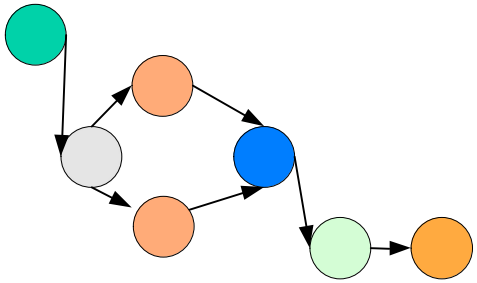
- Most of the configuration of VDS is done by properties.
- Properties can be specified
  - On the command line
  - In $HOME/.vdsrc file
  - In $VDS_HOME/etc/properties
- All properties are described in $VDS_HOME/doc/properties.pdf
- For the tutorial the properties are configured in the $HOME/.vdsrc file

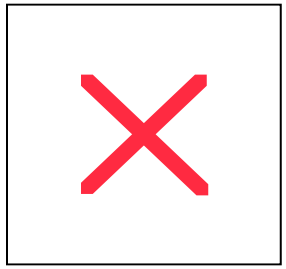
Pegasus Section Outline



- Planning with Pegasus
  - Catalogs used By Pegasus
  - Pegasus Components
- Executing workflows on the Grid
  - Debugging Workflows
- Pegasus Advanced Features and Optimizations
- Further Reading

Grid Job Submission





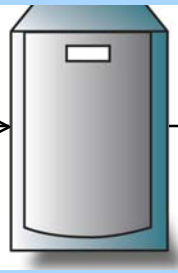
Submit Node

(skynet-login.isi.edu)

Condor DAGMAN/CondorG

Gram jobs

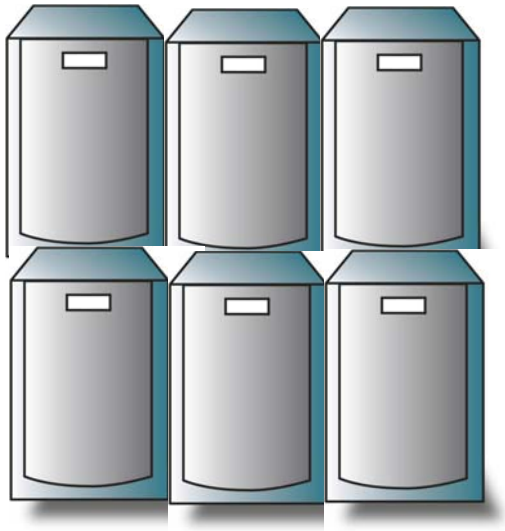
Head Node
(tg-login.ncsa.teragrid.org)



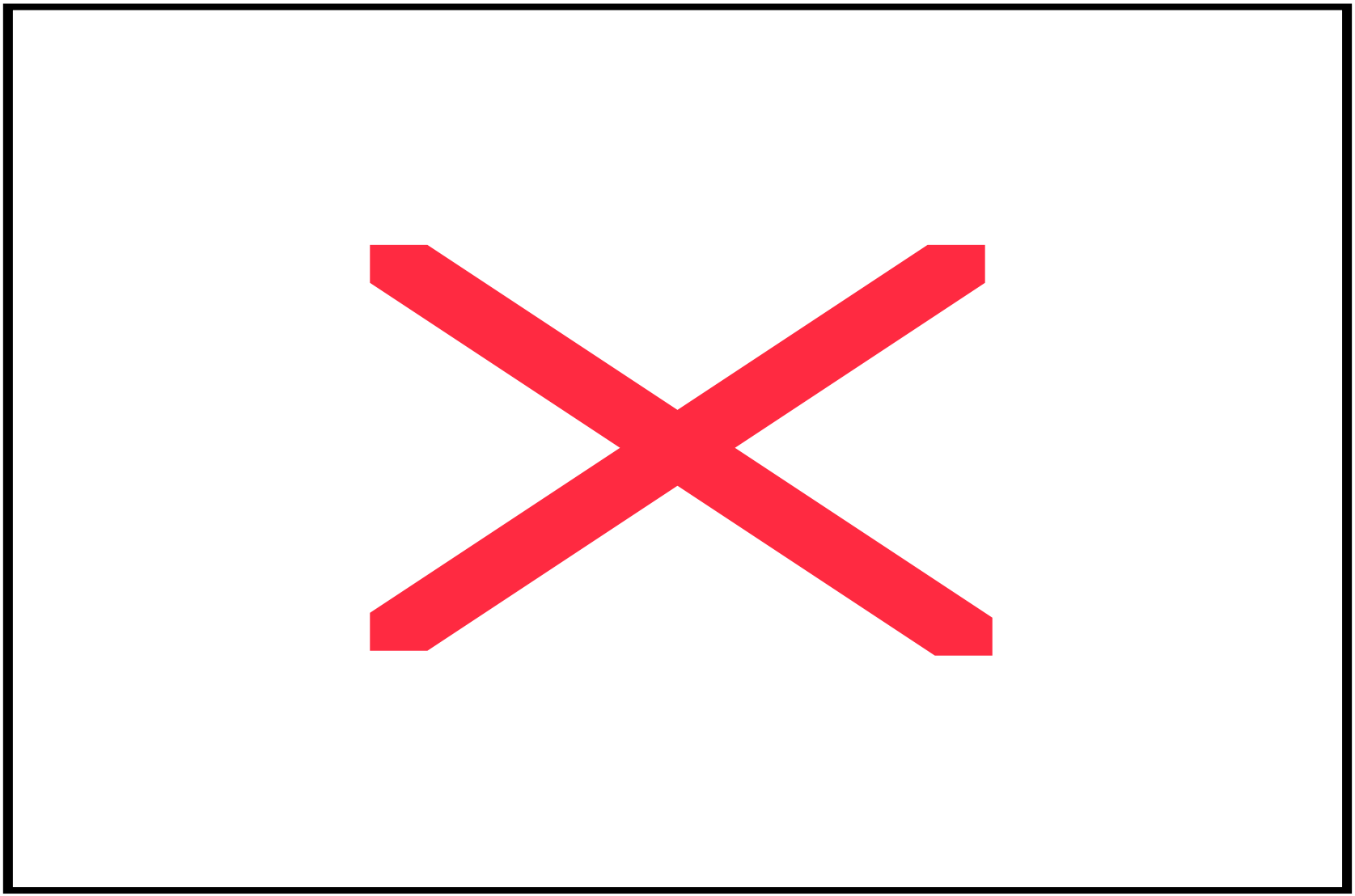
GRAM, PBS

PBS Jobs

Teragrid Cluster



Tutorial Grid Components



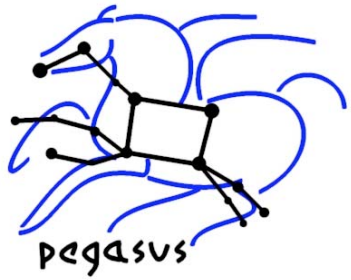
Plan (vds-plan Exercise 3.3)

- As a recap (the whole VDS process is as follows: VDLC -> VDS-PLAN -> VDS_RUN)
- Invokes Pegasus to generate a concrete workflow.
- Generates some default options like the name of the remote work directories that are created for each workflow on the remote grid sites
- An input file (braindump.txt) for the monitoring daemon

Run (vds-run Exercise 3.3)

- Submits the workflow to Condor DAGMAN/ CondorG for remote job submissions
- Starts the monitoring daemon (tailstatd) in the directory containing the condor submit files
- Tailstatd parses the condor output and updates the status of the workflow to a database
- Tailstatd updates job status to a text file jobstate.log in the direcotry containing the condor submit files.

Pegasus Section Outline



- Planning with Pegasus
  - Catalogs used By Pegasus
  - Pegasus Components
- Executing workflows on the Grid
  - Debugging Workflows
- Pegasus Advanced Features and Optimizations
- Further Reading

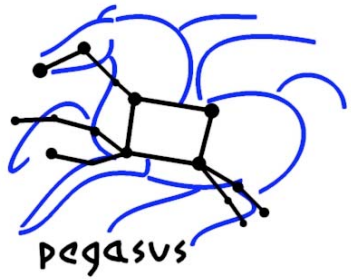
Debugging (Exercise 3.4)

- The status of the workflow can be determined by
  - Looking at the jobstate.log if tailistatd was invoked
  - Or looking at the dagman out file (with suffix .dag.dagman.out)

- All jobs in VDS are launched by a wrapper executable kickstart. Kickstart generates provenance information including the exit code, and part of the remote application ‘s stdout.

- In case of job failure look at kickstart output of the failed job.

Pegasus Section Outline



- Planning with Pegasus
  - Catalogs used By Pegasus
  - Pegasus Components
- Executing workflows on the Grid
  - Debugging Workflows
- Pegasus Advanced Features and Optimizations
- Further Reading

Job Clustering (1)

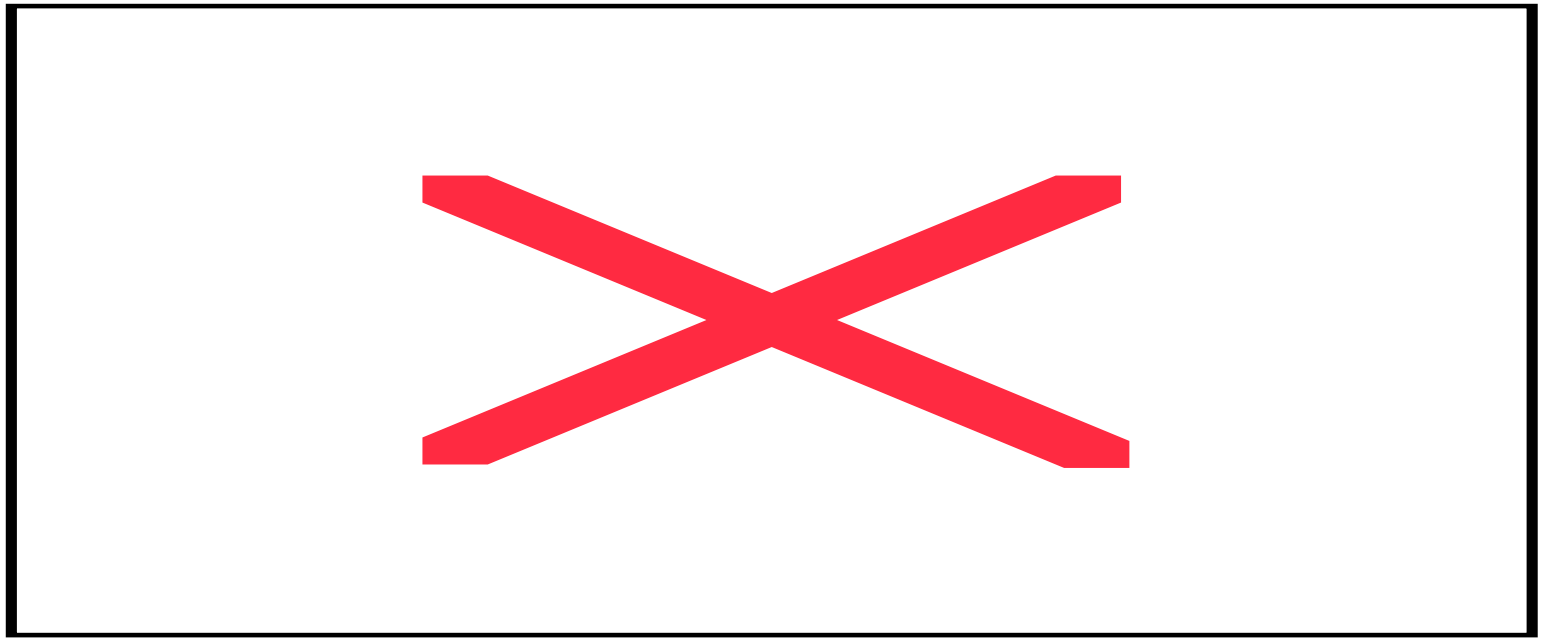
- Cluster small running jobs together to achieve better performance.

- Why?
  - Each job has scheduling overhead
  - Need to make this overhead worthwhile.
  - Ideally users should run a job on the grid that takes at least 10 minutes to execute

More at http://vds.uchicago.edu/vds/doc/userguide/html/H_PegasusJobClustering.html

Or $VDS_HOME/doc/userguide/VDSUG_PegasusJobClustering.xml

Job Clustering(2)



- Horizontal Clustering
  - Jobs on the same level are clustered into larger jobs
  - Clustering parameters can be configured by associating profiles in Transformation Catalog or Site Catalog.
- Vertical Clustering (Soon)
- The clustered job can be run on the remote site
  - Sequentially using VDS tool seqexec.
  - In Parallel using using VDS MPI wrapper mpiexec

Transfer Configurations

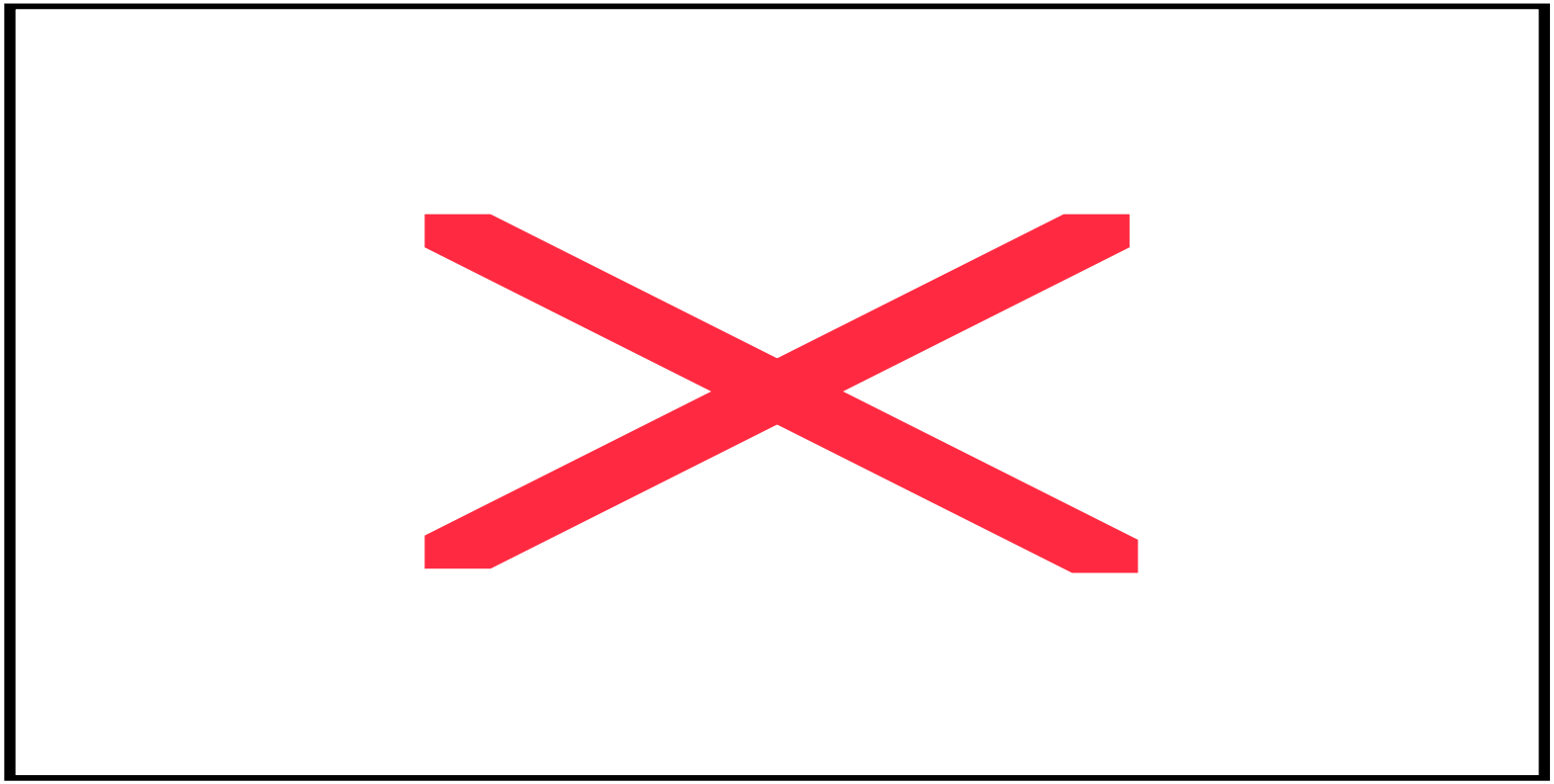
- Variety of transfer clients may be used
  - Set vds.transfer.*.implementation property
  - Support for clients like
    - RFT
    - Stork
    - T2 (VDS client that retries in case of failures)
    - Transfer (VDS client wrapper around g-u-c)
- Variety of refinement strategies maybe used for adding transfer nodes
  - Set vds.transfer.refiner property.
- Varying third party transfer settings
  - Set vds.transfer.*.thirdparty.sites
  - Allows you to specify for which compute sites you want to use for third party party staging.

Explained in more detail at $VDS_HOME/doc/properties.pdf

Transfer Throttling

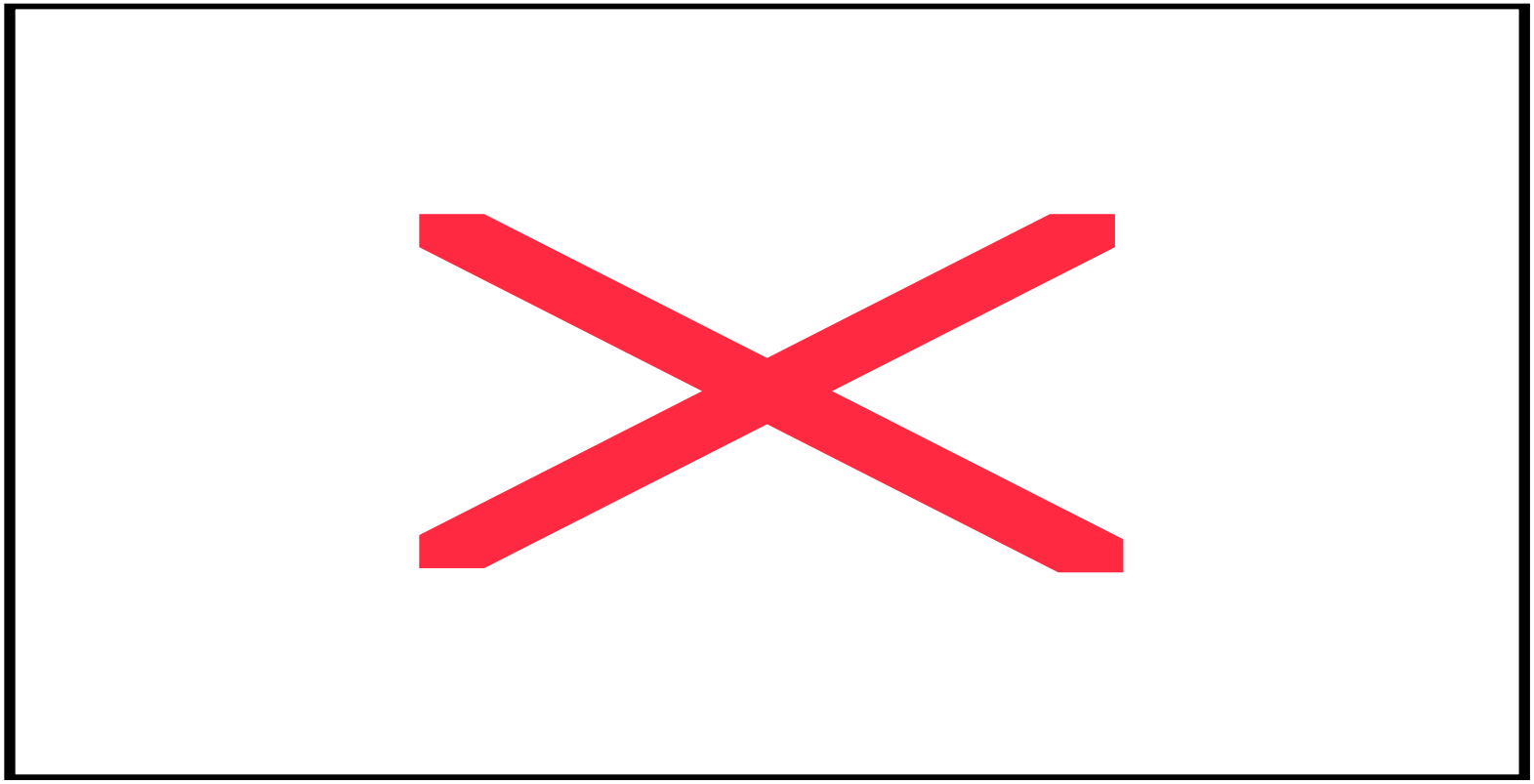
- Large Sized Workflows result in large number of transfer jobs being executed at once. Results in
  - Grid FTP server overload (connection refused errors etc)
  - May result in a high load on the head node if transfers are not configured for being executed as third party transfers
- Need to throttle transfers
  - Set vds.transfer.refiner property.
  - Allows you to create chained transfer jobs or bundles of transfer jobs

Transfer Throttling by Chaining



Explained in more detail at $VDS_HOME/doc/properties.pdf

Transfer Throttling by Bundling



Explained in more detail at $VDS_HOME/doc/properties.pdf

Transfer of Executables

- Allows the user to dynamically deploy scientific code on remote sites
- Makes for easier debugging of scientific code.
- The executables are transferred as part of the workflow
- Currently, only statically compiled executables can be transferred
- Selection of what executable to transfer
  - Set vds.transformation.selector property.

More at "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems" Scientific Programming Journal,January 2005

Also explained in the properties file at $VDS_HOME/doc/properties.pdf

Replica Selection

- Default replica selection
  - Always prefer data present at the compute site, else select randomly a replica
- Restricted Replica Selection
  - Can specify preferred sites from which to stage in data per compute site.
  - Can specify sites to ignore for staging in data per compute site.
- Properties to Set (* in name replaced by site name. * means all sites)
  - vds.replica.selector
  - vds.replica.*.ignore.stagein.sites
  - vds.replica.*.ignore.stagein.sites

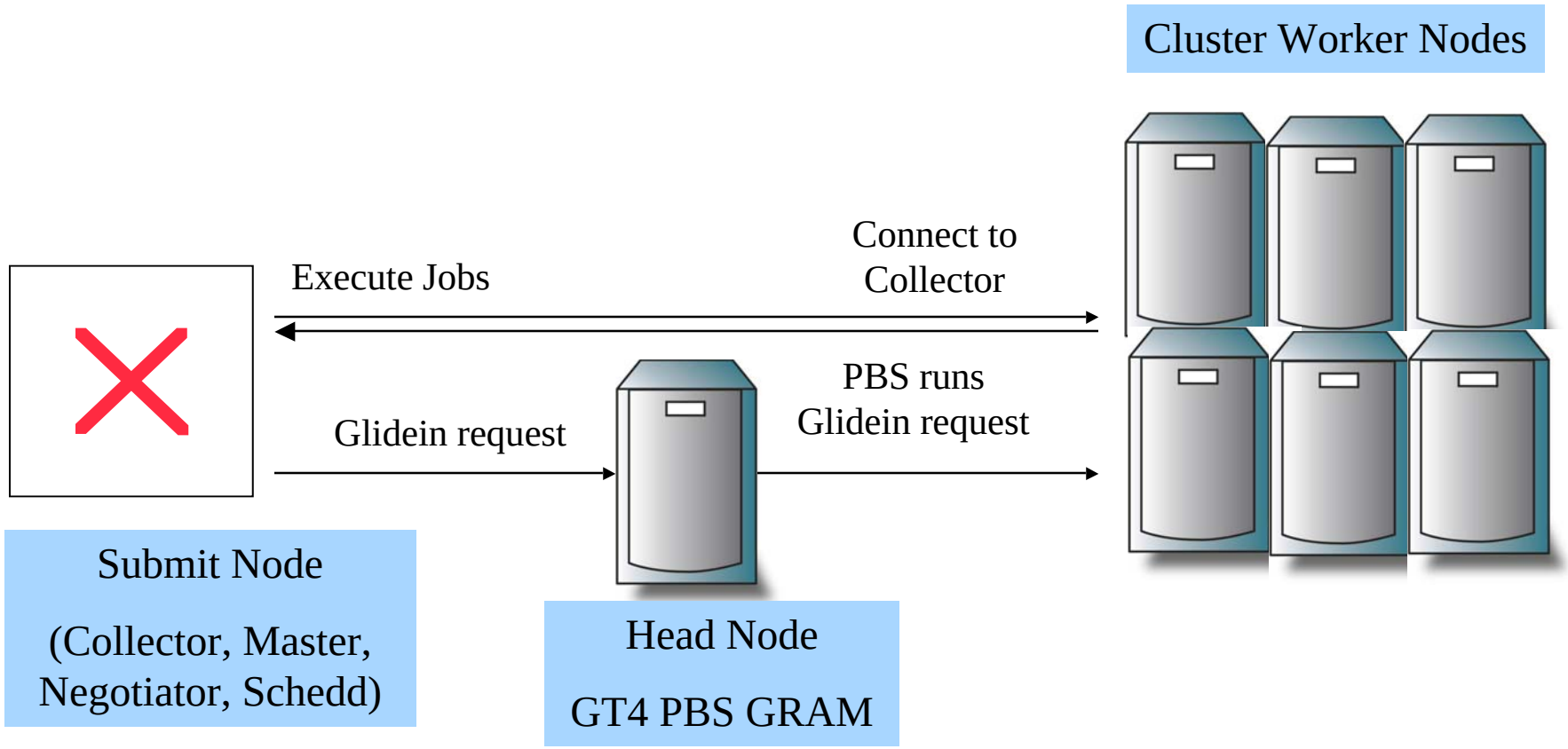
Explained in more detail at $VDS_HOME/doc/properties.pdf

Running in different grid setups

- Need to specify vds namespace profile keys with the sites in the site catalog.
- Submitting directly to condor pool
  - The submit host is a part of a local condor pool
  - Bypasses CondorG submissions avoiding Condor/GRAM delays.
- Using Condor GlideIn
  - User glides in nodes from a remote grid site to his local pool
  - Condor is deployed dynamically on glided in nodes for e.g. you glide in nodes from the teragrid site running PBS.
  - Only have to wait in the remote queue once when gliding in nodes.

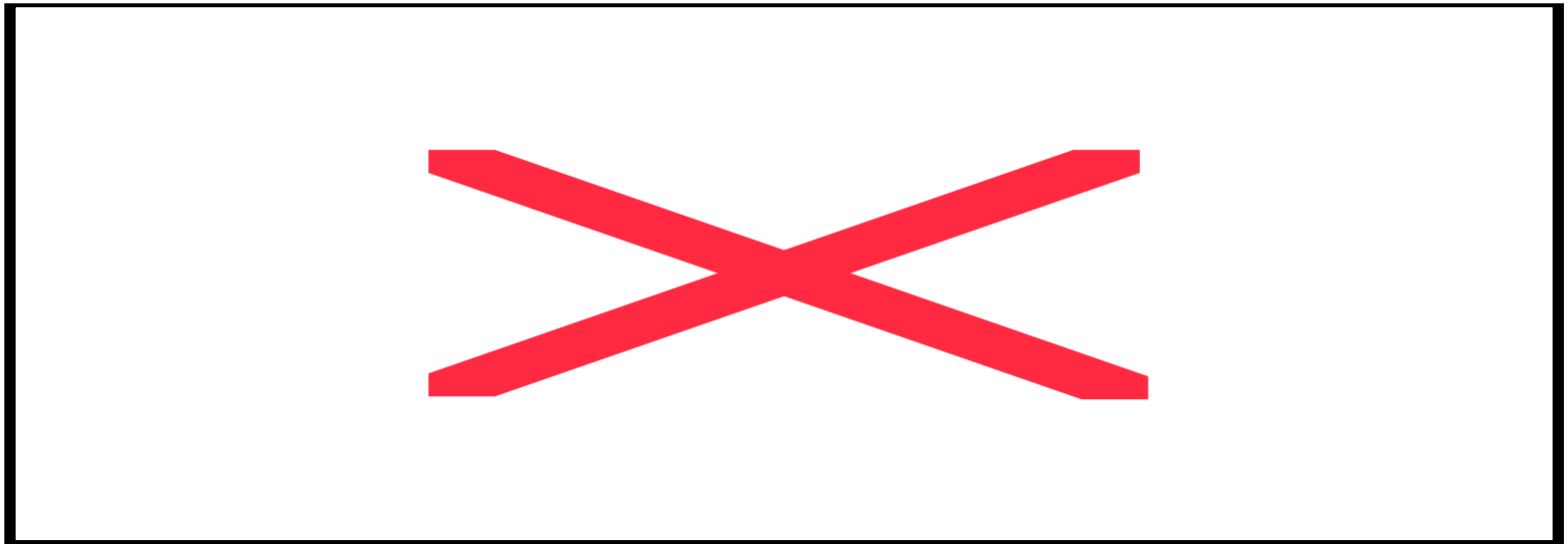
More at http://vds.uchicago.edu/vds/doc/userguide/html/H_RunningPegasus.htmlOr
$VDS_HOME/doc/userguide/VDSUG_RunningPegasus.xml

Condor GlideIn



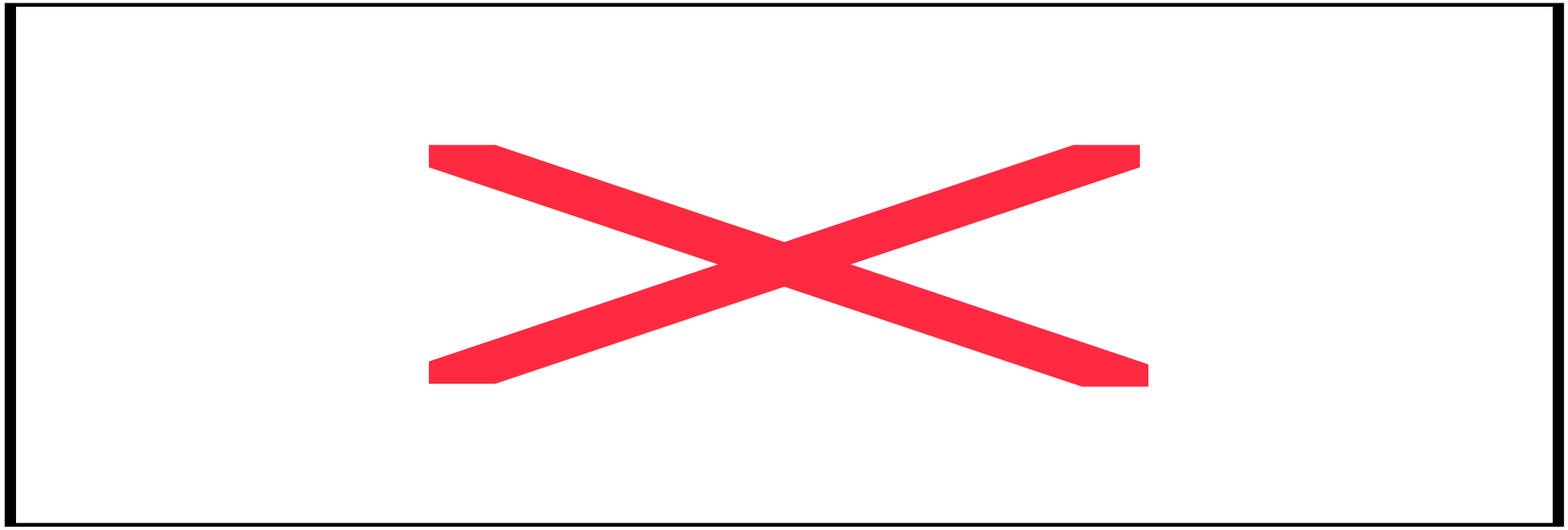
Cluster on a public network

Original Pegasus configuration



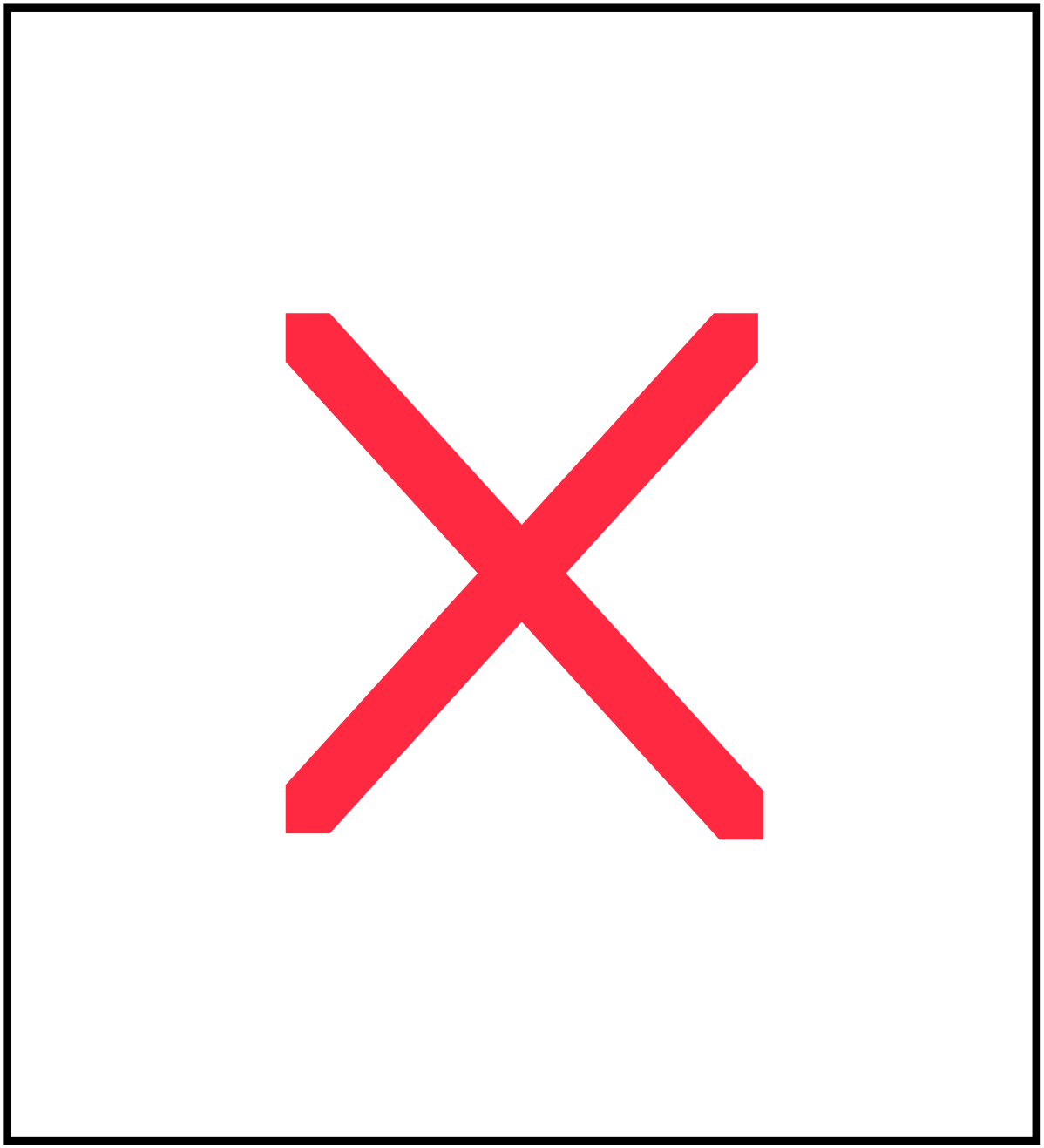
Simple scheduling: random or round robin
using well-defined scheduling interfaces.

Deferred Planning through Partitioning



A variety of partitioning algorithms can be implemented

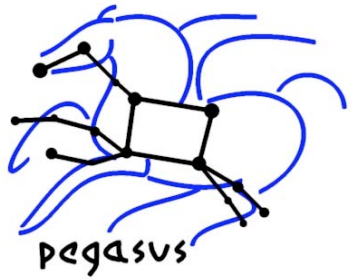
Mega D
by Pega
submitt



Pegasus - Further Reading

- VDS Documents in VDS distribution in $VDS_HOME/doc directory
  - configuration via properties $VDS_HOME/doc/properties.pdf
  - Userguide in $VDS_HOME/doc/userguide directory
- On the web (often lags latest release)
  - http://vds.uchicago.edu/twiki/bin/view/VDSWeb/VDSDocs
  - VDL Reference:

Pegasus Papers



- Papers on Pegasus (more at http://pegasus.isi.edu)
  - "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems" Scientific Programming Journal,January 2005
  - Mapping Abstract Complex Workflows onto Grid Environments, Ewa Deelman , James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard Cavanaugh, and Scott Koranda, Journal of Grid Computing, Vol.1, no. 1, 2003, pp. 25-39.
  - "Artificial Intelligence and Grids: Workflow Planning and Beyond," Yolanda Gil, Ewa Deelman, Jim Blythe, Carl Kesselman, and Hongsuda Tangmurarunkit. *IEEE Intelligent Systems*, January 2004
  - "Transparent Grid Computing: a Knowledge-Based Approach", Jim Blythe, Ewa Deelman, Yolanda Gil, Carl Kesselman, IAAI 2003
  - "The Montage Architecture for Grid-Enabled Science Processing of Large, Distributed Datasets," J. C. Jacob, D. S. Katz, T. Prince, G. B. Berriman, J. C. Good, A. C. Laity, E. Deelman, G. Singh, and M.-H. Su, *Proceedings of the Earth Science Technology Conference (ESTC) 2004*, June 2004.

For further information

- VDS and Pegasus:
  - http://vds.isi.edu
  - http://pegasus.isi.edu
- Mailing Lists
  - vds-support@griphyn.org
  - vds-discuss@griphyn.org
- Workflow Management research group in GGF:
  - www.isi.edu/~deelman/wfm-rg
- Workshops
  - Works06 (http://www.isi.edu/works06/) in conjunction with HPDC 2006.
  - NSF Workflow Workshop (http://vtcpc.isi.edu/wiki/index.php/Main_Page)

Outline

- Part 1: Concept & applications of Virtual Data
- Part 2: VDL – The Virtual Data Language
- Part 3: Pegasus – Grid Workflow Planning
- Part 4 : VDS in the Science Process
- Summary and Conclusion

Mapping the Science Process to VDS

- Start with a single workflow
- Automate the generation of workflow for sets of files (datasets)
- Replicate workflow to explore many datasets
- Change Parameters
- Change code – add new transformations
- Build new workflows
- Leverage availability of provenance info

fMRI Dataset processing

```
FOREACH BOLDSEQ
DV reorient (# Process Blood O2 Level Dependent Sequence
  input =  [ @{in: "$BOLDSEQ.img"},
             @{in: "$BOLDSEQ.hdr"}  ],
  output = [@{out: "$CWD/FUNCTIONAL/r$BOLDSEQ.img"}
            @{out: "$CWD/FUNCTIONAL/r$BOLDSEQ.hdr"}],
  direction = "y", );
END

DV softmean (
  input = [ FOREACH BOLDSEQ
     @{in:"$CWD/FUNCTIONAL/r$BOLDSEQ.img"}
  END ],
  mean = [ @{out:"$CWD/FUNCTIONAL/mean"} ]
);
```

Query Examples

Which TRs can process a "subject" *image?*

- Q: xsearchvdc -q tr_meta dataType subject_image input
- A: fMRIDC.AIR::align_warp

Which TRs can create an "ATLAS"*?*

- Q: xsearchvdc -q tr_meta dataType atlas_image output
- A: fMRIDC.AIR::softmean

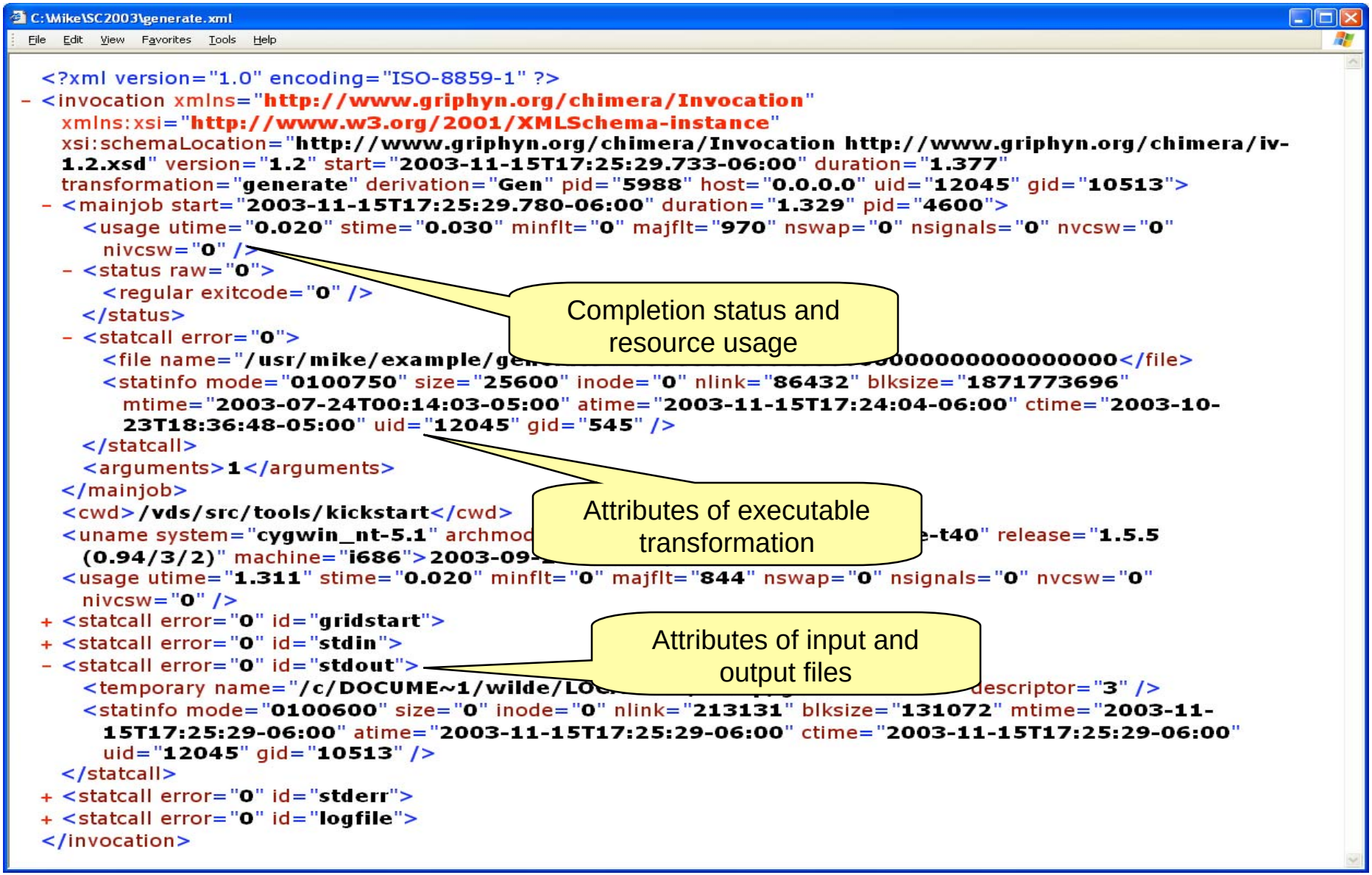
Which TRs have output parameter "warp" and a parameter "options"

- Q: xsearchvdc -q tr_para warp output options
- A: fMRIDC.AIR::align_warp

Which DVs call TR "slicer"?

- Q: xsearchvdc -q tr_dv slicer
- A: fMRIDC.FSL::s3472_3_x->fMRIDC.FSL::slicer
  fMRIDC.FSL::s3472_3_y->fMRIDC.FSL::slicer
  fMRIDC.FSL::s3472_3_z->fMRIDC.FSL::slicer

Invocation Provenance



Outline

- Part 1: Concept & applications of Virtual Data
- Part 2: VDL - The Virtual Data Language
- Part 3: Pegasus: Grid Workflow Planning
- Part 4 : VDS in the Science Process
- Summary and Conclusion

Virtual Data and Workflows

- Addresses the challenge of managing and organizing the vast computing and storage capabilities provided by Grids
- **VDS** enables **workflow** to be expressed in form that can be readily mapped to Grids
- **Virtual data** keeps accurate track of data derivation methods and **provenance**
- VDS **virtualizes** location of applications and data, and recovery from failures

Benefits of the Virtual Data approach

- Virtual Data System provides location-independent computing: represents all workflow in abstract terms
- Declarations not tied to specific entities:
  - sites
  - file systems
  - schedulers
- Failures – automated retry for data server and execution site un-availability

For further information

- VDS software, documentation, papers:
  - www.griphyn.org/vds
- Pegasus planner:
  - http://pegasus.isi.edu
- GGF Workflow Management research group:
  - www.isi.edu/~deelman/wfm-rg

Acknowledgements

The technologies and applications described here were made possible by the following projects and support:

GriPhyN, iVDGL, the Globus Alliance and QuarkNet, supported by
The National Science Foundation

The Globus Alliance, PPDG, and QuarkNet, supported by the US Department of Energy, Office of Science