Grids and Workflows

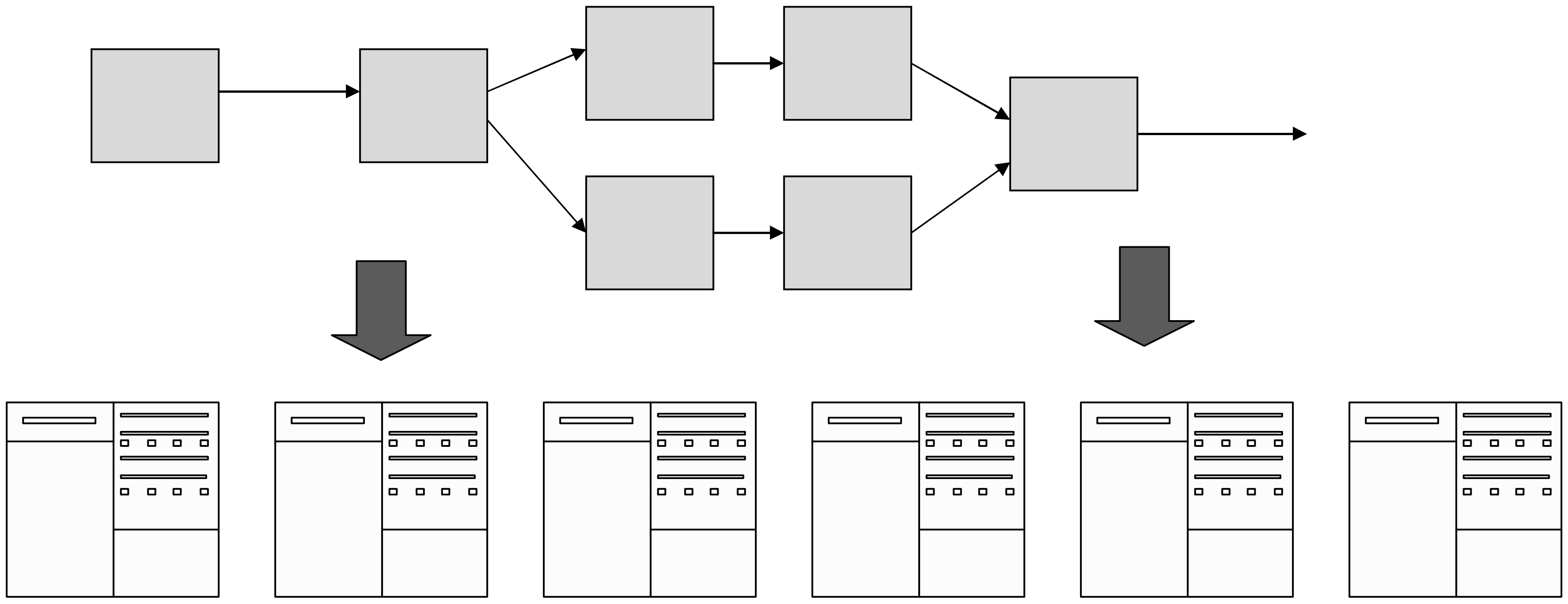
Overview

- Scientific workflows and Grids
  - Taxonomy
  - Example systems
- Kepler revisited
- Data Grids
  - Chimera
  - GridDB

Workflows and Grids

•Given a set of workflow tasks and a set of resources, how do we map them to Grid resources?

•What are some other challenges?



Executing Scientific Workflows on Grids

- Grids can address many challenges of scientific workflow execution
  - Scalability
  - Detached execution
- Many systems have been developed to aid in design and execution of Grid workflows

Taxonomy

- Classifies 4 elements of workflow systems in context of Grid computing
  - Workflow design
  - Workflow scheduling
  - Fault Tolerance
  - Data Movement

Workflow Design

- *Workflow structure* indicates temporal relationship between tasks
  - Can be Directed Acyclic Graph (DAG) or non-DAG
- DAG-based
  - Sequence (ordered series of tasks)
  - Parallel (tasks that run concurrently)
  - Choice (task executed at runtime if all conditions are true)
- Non-DAG
  - Iteration (sections of workflow can be repeated)

Workflow Design

- *Workflow Model/Specification* defines workflow including task definition and structure definition
- Abstract model
  - Workflow specified without referring to specific resources
- Concrete model
  - Bind workflow tasks to specific resources
- Applications that use abstract can generate concrete model before or during execution

Workflow Design

- *Workflow Composition System* enables users to assemble components into workflows
- User-directed
  - – Users edit workflows directly
  - – Language-based (e.g., XML)
  - – Graph-based (e.g., Kepler)
- Automatic
  - – Generate workflows from higher-level requirements, e.g., data products, input values
  - – Difficult to capture functionality of components

Workflow Scheduling

- Scheduling architecture can be *centralized, hierarchical,* or *decentralized*
- Centralized- one central scheduler makes decisions for all tasks in a workflow
- Hierarchical- central manager assigns sub-workflows to lower-level schedulers
- Decentralized- multiple schedulers that can communicate with each other and balance load
- Optimality/scalability tradeoff

Workflow Scheduling

- How to map workflows onto resources?
- Decisions can be based on current task or subworkflow (local) or entire workflow (global)
- Global decisions may produce better results, but high overhead

Workflow Scheduling

- How to translate abstract models to concrete models?
- Static – concrete models generated before execution
  - User directed or simulation based
- Dynamic – make decisions at runtime
  - Prediction-based or just in time

Workflow Scheduling

- Scheduling workflow applications in distributed system is NP-complete
- Use heuristics to match users Quality of Service constraints (deadline, budget)
- *Performance-driven* –minimize overall execution time
- *Market-driven* –minimize usage price
- *Trust-driven*- select resources based on trust properties (security, reputation, site vulnerability, etc)

Fault Tolerance

- Failures may occur for a variety of reasons: network failure, overloaded resource conditions, non-availability of components
- Failure handling: task-level and workflow-level
  - Task-level – mask the effects of the failure
  - Workflow-level – manipulate workflow structure

Fault Tolerance

- Task level
  - Retry
  - Alternate resource
  - Checkpoint/restart
  - Replication
- Workflow level
  - Alternate task
  - Redundancy
  - User-defined exception handling
  - Rescue workflow

Intermediate Data Movement

- Input files of tasks need to be staged at remote site before processing tasks
- Output files may be required by child tasks processed on other resources
- User directed – movement specified as part of workflow
- Automatic – system does it automatically
- Approaches can be centralized, mediated, or peer-to-peer

Intermediate Data Movement

- Centralized
  - Easy to implement
  - Good when large-scale data flow not required
- Mediated
  - Intermediate data managed by distributed data management system
  - Good when want to keep data for later use
- Peer-to-Peer
  - Good for large-scale data transfer
  - But more difficulties to deployment

Some examples

- Kepler
- Taverna
- Triana
- GrADS
- Pegasus

Kepler Classification

- Structure – non-DAG
- Graph-based
- Centralized architecture
- Many user-defined features
  - Scheduling
  - Fault tolerance
  - Data movement

Taverna

- Workflow management system of the myGrid project
- Workflow can be expressed either graphically (Kepler-like GUI) or XML-based language (SCUFL)
- Allows implicit iteration over incoming datasets
- Allows multithreading to speed up interation
- Good for services capable of simultaneous processing, e.g., those backed by a cluster

Triana

- Visual workflow-oriented data analysis environment
- Clients can log in to Triana Controlling Service (TCS)
- TCS can execute locally or distribute based on distribution policy
  - Parallel – no host-based communication
  - Peer-to-peer – intermediate data passed between hosts
- Resources dynamically allocated

GrADS

- **Gr**id **A**pplication **D**evelopment **S**oftware
- Application-level task scheduling
- Goal: minimize overall job completion time (makespan) – performance driven
- Scheduler maps tasks to resources using heuristics
  - Weighted sum of expected execution time on resource and expected cost of data movement
  - Monitors performance of executing tasks and reschedules as needed

Pegasus

- Workflow manager in GriPhyN
- Maps abstract workflow to available Grid resources and generates executable workflow
- DAG structure
- Two methods for resource selection:
  - Random allocation
  - Performance prediction
- Intermediate data registered with replica service (mediated approach)

Summary and Challenges

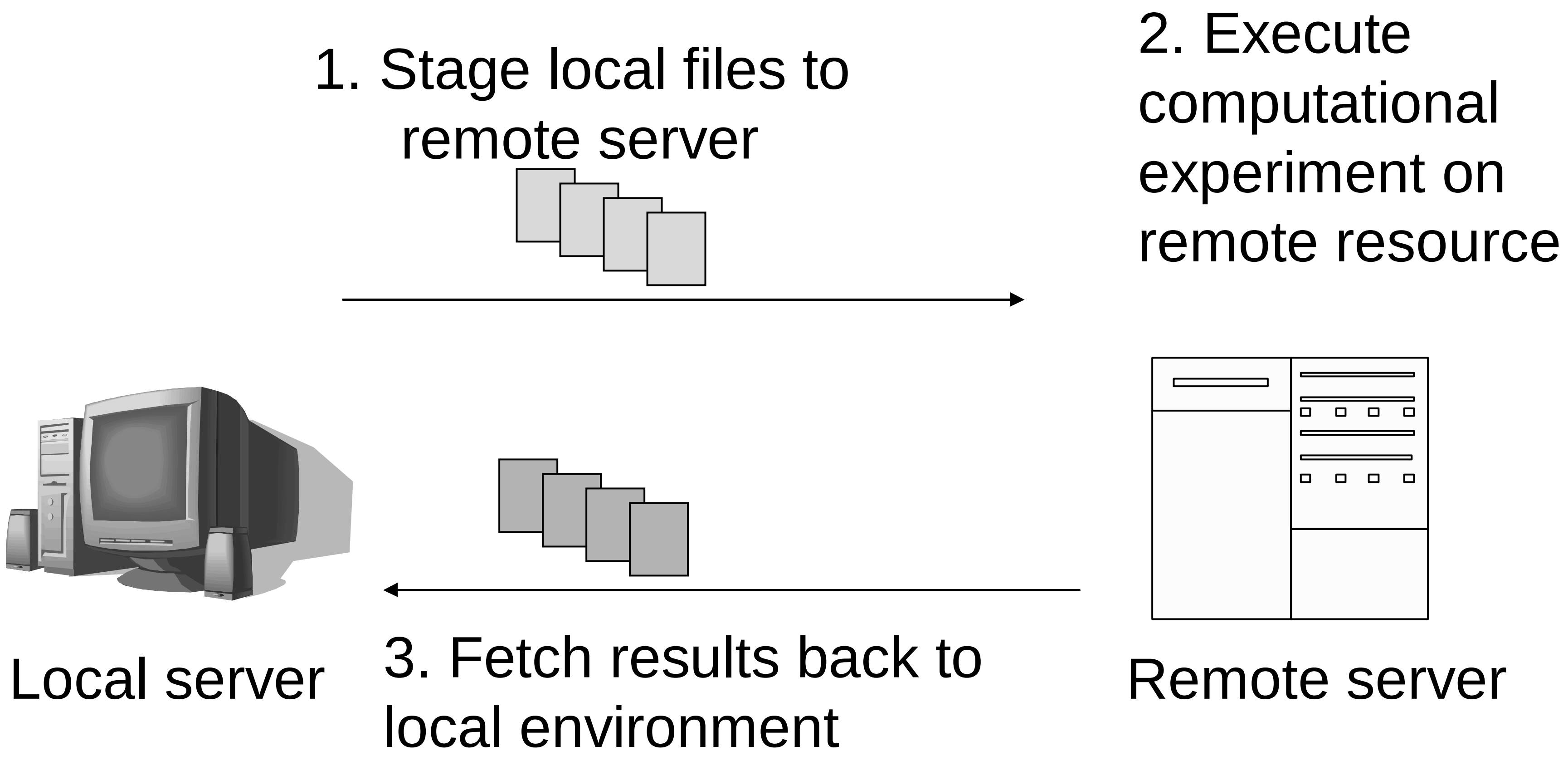
- Many projects have graphical workflow modeling language
  - Standardization needed
- Quality of Service (QoS) not well addressed
  - QoS needed at both specification and execution level
  - Market-driven strategies will become increasingly important
- Optimal schedule requires estimates of task execution time
  - Analytical models (GrADS) or historical performance (Pegasus)
- Better fault tolerance needed

Executing Kepler on the Grid

- Many challenges to Grid workflows, including:
  - Authentication
  - Data movement
  - Remote service execution
  - Grid job submission
  - Scheduling and resource management
  - Fault tolerance
  - Logging and provenance
  - User interaction
- May be difficult for domain scientists

Example Grid Workflow

Stage-execute-fetch:


1. Stage local files to remote server
2. Execute computational experiment on remote resource
Local server
3. Fetch results back to local environment
Remote server

Why not use a script?

- Script does not specify low-level task scheduling and communication
- May be platform-dependent
- Can’t be easily reused

Some Kepler Grid Actors

- Copy – copy files from one resource to another during execution
  - Stage actor – local to remote host
  - Fetch actor - remote to local host
- Job execution actor – submit and run a remote job
- Monitoring actor – notify user of failures
- Service discovery actor – import web services from a service repository or web site

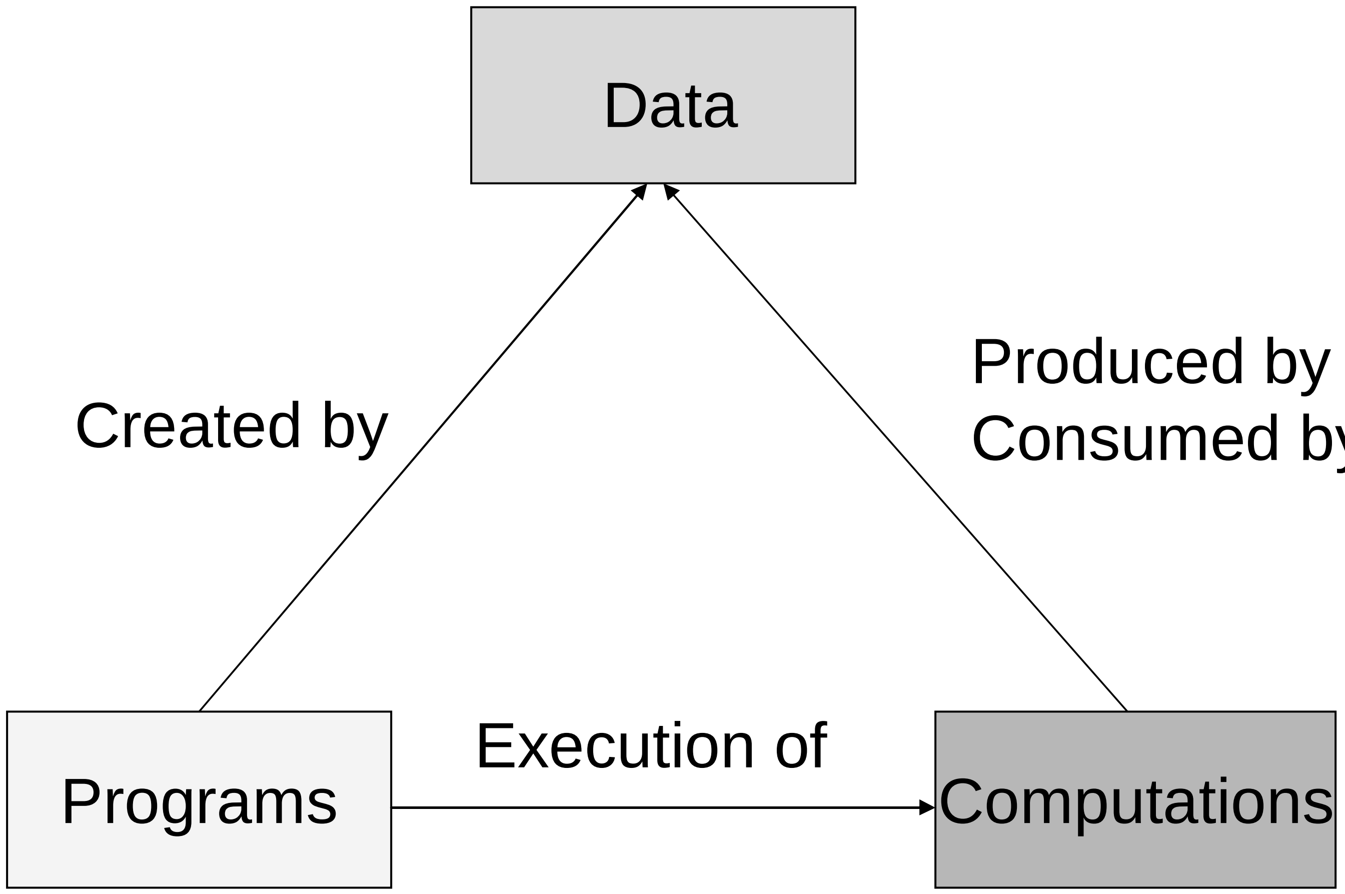
Data Grids

- Chimera
- GridDB

Data Grids

- Communities collaboratively construct collections of *derived data*
  - Flat files, relational tables, persistent object structures
- Relationships between data objects corresponding to computational procedures used to derive one from the other

Relationships among Programs,Computations, Data



Challenges

- “I’ve come across some interesting data, but I need to understand how it was constructed before I can trust it for my purposes.”
- “I want to search an astronomical database for galaxies with certain characteristics. If a program that does this exists, I won’t need to write one from scratch.”
- “I want to apply an astronomical analysis program to millions of objects. If the program has already been run and the results stored, I’ll save weeks of computation.”
- “I’ve detected a calibration error in an instrument and want to know which derived data to recompute”.

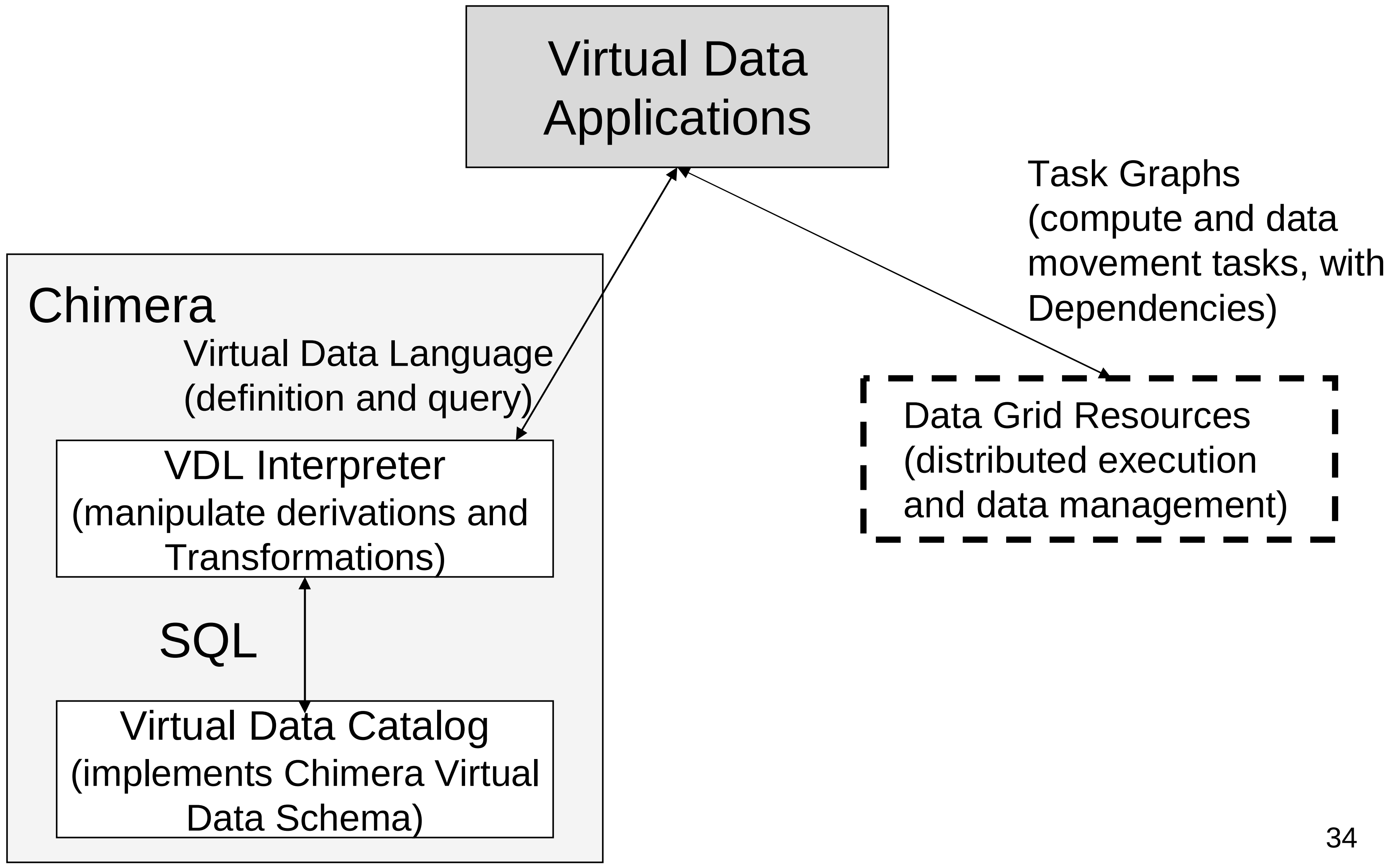
Virtual Data

- Track how data products are derived
- Ability to create and/or recreate products using this knowledge
- “Virtual data management” operations
  - “Re-materialize” deleted data products
  - Generate data products defined but not created
  - Regenerate data when dependencies or programs change
  - Create replicas at remote locations when cheaper than transfer

Chimera (Foster et al., 2002) (now GriPhyN VDS)

- Virtual data system
- Two main components:
  - – Virtual data catalog (VDC)
    - Implements virtual data schema
  - – Virtual data language interpreter
    - Implements tasks to call VDC operations
- Queries can return a representation of tasks that will generate a specified data product

Chimera Architecture


Virtual Data Applications
Task Graphs (compute and data movement tasks, with Dependencies)
Chimera
Virtual Data Language (definition and query)
VDL Interpreter (manipulate derivations and Transformations)
SQL
Virtual Data Catalog (implements Chimera Virtual Data Schema)
Data Grid Resources (distributed execution and data management)

Some definitions

- *Transformation* – an executable program
- *Derivation* – an execution of a transformation
- *Data object* – named entity that may be consumed or produced by a derivation
  - – Logical file name
  - – Replica catalog maps logical name to physical location
  - – Data objects can also be relations or objects

Chimera Virtual Data Language

```
TR t1 ( output a2, input a1,
        none env="100000",
        none pa = "500") {
    app vanilla="/usr/bin/app3";
    app parg = "-p "${none:pa};
    app farg  = "-x –y ";
    arg stdout = ${output:a2};
    profile env.MAXMEM = ${none:env};
}
```

t1 reads input file a1 and produces a2

app is application to run (/usr/bin/app3)

args are default argument values

stdout redirects output to a2

Chimera VDL

```
DV t1 (
  a2=@{output:run1.exp15.T1932.summary},
  a1=@{input:run1.exp15.T1932.raw},
   env =“20000”, pa=“600” );
```

String after DV indicates transformation to be invoked (t1)

Corresponding invocation:

```
export MAXMEM=20000
/usr/bin/app3 –p 600 \
     -f run1.exp15.T1932.raw –x –y \
     > run1.exp15.T1932.summary
```

Queries

- VDL implemented in SQL
- Queries allow one to search for transformations by name, application name, input LFN(s), output LFN(s), argument matches, or other metadata
- Query results indicate if desired transformations already exist in data grid
  - Retrieve them if they do
  - Create them if they do not

Example: SDSS Galactic Structure Detection

- Applied virtual data to locating galactic clusters in image collection
- Sky tiled into set of “fields”
- For each field, search for clusters in that field and some set of neighbors
- Use “brightest cluster galaxy” (BCG) and “brightest red galaxy” (BRG) to determine cluster candiates

SDSS Galactic Structure Detection

1. fieldPrep – extract required measurements from galaxies and produce files with this data (~40x smaller than original files)
2. brgSearch – unweighted BCG likelihood for each galaxy
3. bcgSearch – weighted BCG likelihood (most expensive step
4. bcgCoalesce – determine whether a galaxy is most likely galaxy in the neighborhood
5. getCatalog – remove extraneous data and store result in compact format

SDSS Galactic Structure Detection

- *getCatalog* is a function that can invoke the four prior dependent steps
- Generate “virtual” results for entire sky by defining one derivation of *getCatalog* for each field

Virtual Data Summary

- Performs “bookkeeping” to track large scale productions
- Can be thought of as paradigm for management of batch job production scripts or a “makefile” for data production
- Data production can be performed interactively in parallel by users
  - Virtual data grid acts as a “cache”

Pegasus and Chimera

- Pegasus can construct an abstract workflow using Chimera
- Before mapping tasks to resources, Pegasus reduces abstract workflow by eliminating materialized data products
  - Assumes more costly to reproduce dataset than to access existing results
- Pegasus can automatically generate a workflow using metadata description of desired data product using AI planning

GridDB (Liu and Franklin, 2004)

- Data-centric overlay for scientific grid data analysis
- Manage data entities rather than processes
- Idea: provide interactive database interface to Grid computing

GridDB: Background

- Assumptions:
  - Scientific analysis programs can be abstracted as typed functions, and invocations as function calls
  - While most scientific data is not relational, there is a subset with relational characteristics

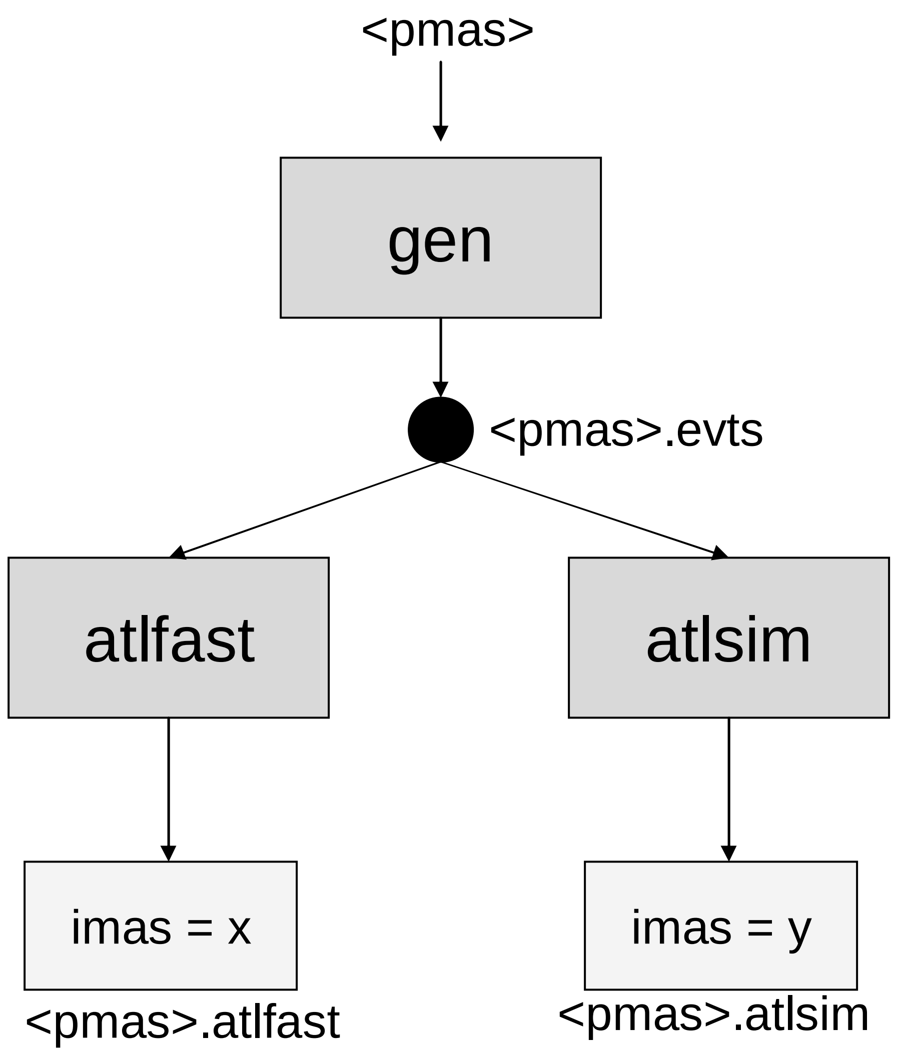
Benefits of GridDB

- Declarative interface
- Type checking
- Interactive Query Processing
- Memoization support
- Data provenance
- Co-existence with process-centric middleware

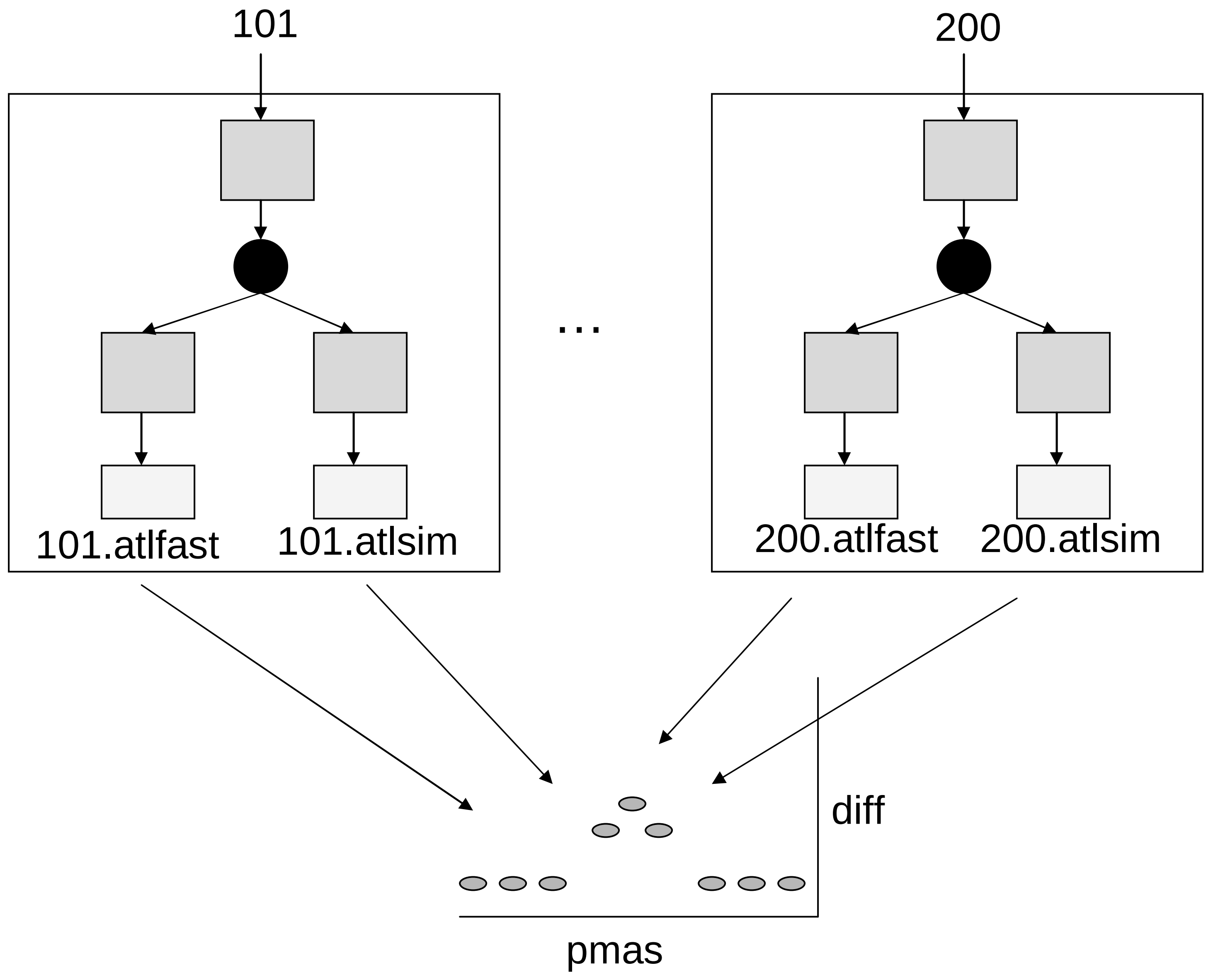
High-Energy Physics Example

- Scientists want to replace a slow but trusted detector simulation with faster, less precise one
- To ensure soundness of new simulation, need to compare response of new and old simulation for various physics events

High Energy Physics Abstract Workflow



Grid Invocation



GridDB Modeling Principles

- Programs and workflows can be represented as *functions*
- An important subset of data in workflow can be represented as relations – *relational cover*
- Represent inputs and outputs to workflows as relational tables

High Energy Physics Example

Input

gRn

gID	pmas
g00	101
…	…
g99	200

Output

fRn

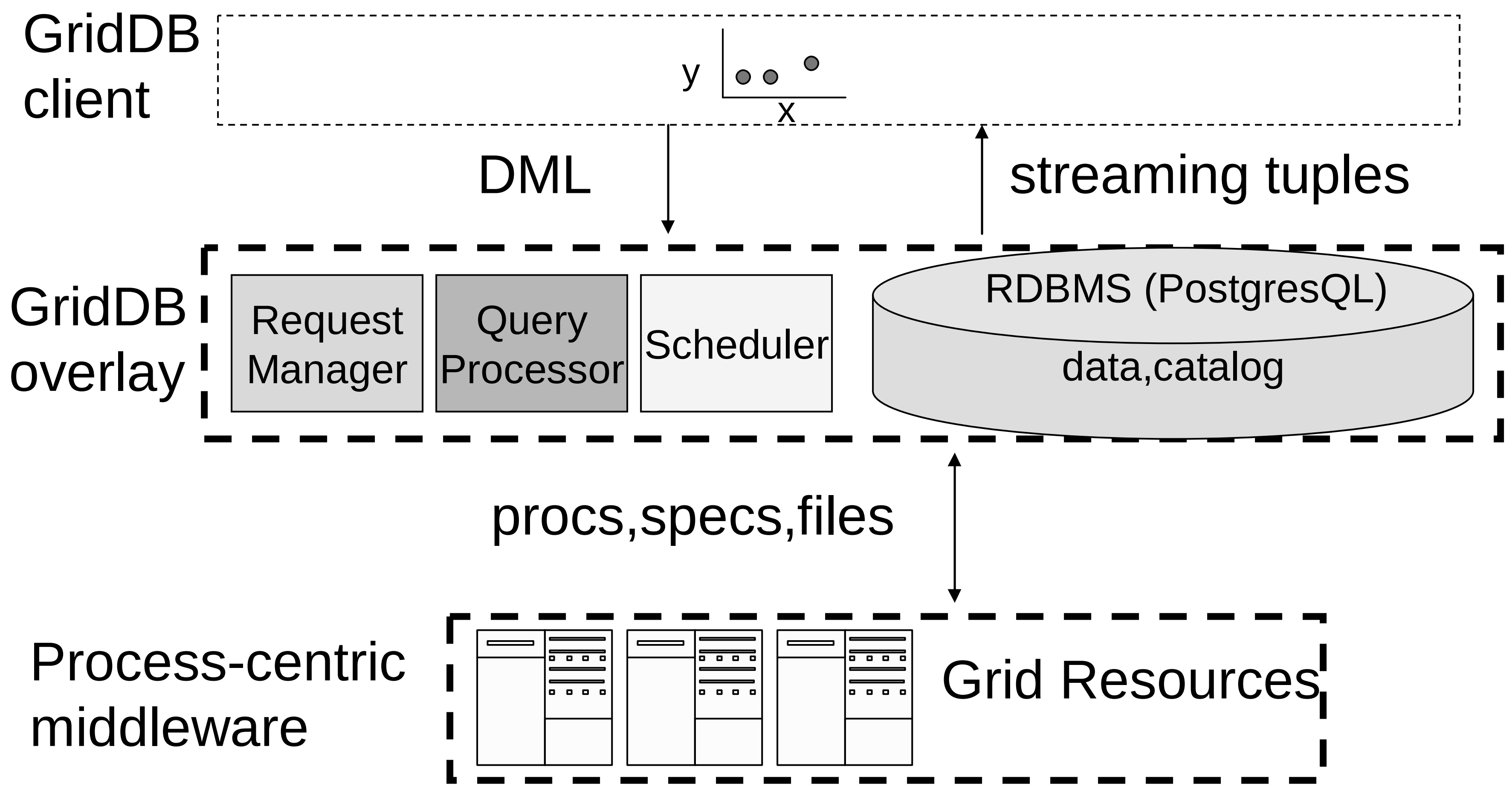
fID	fImas
f00	102
…	…
f99	…

Output

sRn

sID	sImas
s00	100
…	…
s99	…

GridDB Architecture



Basic actions

- Workflow setup – create sandbox entity-sets and connect as inputs/outputs
- Data procurement – submission of inputs to workflow, triggering function evaluations to create output entities
- Automatic views – for streaming partial results

Basic actions

1. gRn:set(g); fRn:set(f); sRn:set(s);
2. (fRn,sRn) = simCompareMap(gRn);
3. INSERT INTO gRn VALUES pmas= {100, …,200};
4. SELECT * FROM autoview(gRn, fRn,sRn);

Summary

- GridDB can leverage relational database functionality
- Provides interactive data-centric interface
- What are some challenges/limitations?