Scientific Workflows

Overview

- More background on workflows
- Kepler Details
- Example Scientific Workflows
- Other Workflow Systems

Recap from last time

- Background: What is a scientific workflow?
  - Goals: automate a scientist's repetitive data management and analysis tasks
  - Typical Phases:
    - Data access, scheduling, generation, transformation, aggregation, analysis, visualization
    - Design, test, share, deploy, execute, reuse SWF's
- Overview and demo of Kepler

Adapted from B. Ludaescher

Scientific Workflows: Some Findings

- Very different granularities: from high-level design to lowest level plumbing
- More dataflow than (business control) workflow
- Need for “programming extensions”
  - Iterations over lists (foreach), filtering, functional composition, generic & higher-order operations (zip, map(f))
- Need for abstraction and nested workflows

Adapted from B. Ludaescher

Scientific Workflows: findings (continued)

- Need for data transformations
- Need for rich user interaction and workflow steering
  - Pause/revise/resume
  - Select & branch, e.g., web browser capability at specific steps as part of a coordinated SWF
- Need for high-throughput data transfers and CPU cycles “Grid-enabling”, “streaming”
- Need for persistence of intermediate products and provenance

Adapted from B. Ludaescher

Data-flow vs Control-flow

- Useful for
  - Specification (language, model)
  - Synthesis (scheduling, optimization)
  - Validation (simulation, formal verification)
- Rough classification:
  - Control
    - Don’t know when data arrive (quick reaction)
    - Time of arrival often matters more than value
  - Data
    - Data arrive in regular streams (samples)
    - Value matters most

Adapted from B. Ludaescher

Data-flow vs. Control-flow

- Specification, synthesis, and validation methods tend to emphasize…
- For control:
  - Event/reaction relation
  - Response time
  - (Real time scheduling for deadline satisfaction)
  - Priority among events and processes

Adapted from B. Ludaescher

Data-flow vs. Control-flow

- For Data:
  - Functional dependency between input and output
  - Memory/time efficiency
  - (Dataflow scheduling for efficient pipelining)
  - All events and processes are equal

Adapted from B. Ludaescher

Business Workflows vs. Scientific Workflows

- Business Workflows
  - Task oriented: travel reservations, credit-approval, etc.
  - Tasks, documents, etc undergo modifications (e.g., flight reservation from *reserved* to *ticketed)*, but modified WF objects still identifiable throughout
  - Complex control flow, complex process composition
  - Dataflow and control-flow are often *divorced*

Adapted from B. Ludaescher

Business Workflows vs. Scientific Workflows

- Scientific Workflows
  - Dataflow and data transformations
  - Data problems: volume, complexity, heterogeneity
  - Grid aspects:
    - Distributed computation
    - Distributed data
  - User-interactions/WF steering
  - Data, tool, and analysis integration
  - Dataflow and control-flow are often *married*

Adapted from B. Ludaescher

SWF User Requirements

- Design tools – especially for non-expert users
  - Need to look into how scientists define processes
- Ease of use – fairly simple user interface having more complex features hidden in background
- Reusable generic features
- Generic enough to serve different communities but specific enough to serve one domain
- Extensibility for the expert user – almost a visual programming interface
- Registration and publication of data products and “process products” (workflows); provenance

Adapted from B. Ludaescher

SWF Technical Requirements

- Error detection and recovery from failure
- Logging information for each workflow
- Allow data-intensive and compute-intensive tasks (maybe at the same time)
- Data management/integration
- Allow status checks and on the fly updates
- Visualization
- Semantics and metadata based dataset access
- Certification, trust, security

Adapted from B. Ludaescher

Challenges/Requirements

- Seamless access to resources and services
  - Web services are simple solution but doesn't address harder problems, e.g., web service orchestration, third party transfers
- Service composition & reuse and workflow design
  - How to compose simple services to perform complex tasks
  - Design components that are easily reusable, not application-specific

Adapted from B. Ludaescher

Challenges/Requirements

- Scalability
  - Some workflows require large amounts of data and/or high-end computational resources
  - Require interfaces to Grid middleware components
- Detached execution
  - Allow long running workflows to run in the background on remote server
- Reliability and Fault Tolerance
  - e.g., workflow could fail if web service fails

Adapted from B. Ludaescher

Challenges/Requirements

- User interaction
  - e.g., users may inspect intermediate results
- “Smart” re-runs
  - Changing a parameter after intermediate results without executing workflow from scratch
- “Smart” semantic links
  - Assist in workflow design by suggesting which components might fit together
- Data Provenance
  - Which data products and tools created a derived data product
  - Log sequence of steps, parameter settings,etc.

Adapted from B. Ludaescher

Why is a GUI useful?

- No need to learn a programming language
- Visual representation of what workflow does
- Allows you to monitor workflow execution
- Enables user interaction
- Facilitates sharing workflows

Kepler Details

- Director/Actor metaphor
  - Actors are executable components of a workflow
  - Director controls execution of workflow
- Workflows are saved as XML files
  - Workflows can easily be shared/published

Directors

- Many different models of computation are possible
- Synchronous – Processing occurs one component at a time
- Parallel – One or more components run simultaneously
- Every Kepler workflow needs a director

Actors

- Reusable components that execute a variety of functions
- Communicate with other actors in workflow through *ports*
- Composite actor – aggregation of actors
- Composite actor may have a local director

Parameters

- Values that can be attached to workflow or individual directors/actors
- Accessible by all actors in a workspace
- Facilitate worklflow configuration
- Analogous to global variables

Ports

- Ports used to produce and consume data and communicate with other actors in workflow
  - Input port – data consumed by actor
  - Output port – data produced by actor
  - Input/output port – data both produced and consumed
- Ports can be singular or multiple

Relations

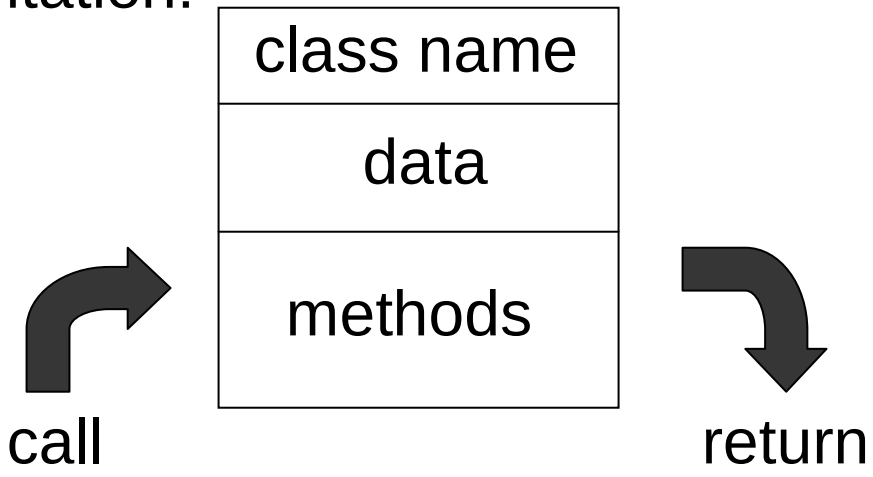
- Direct the same input or output to more than one other port
- Example: direct an output to a display actor to show intermediate results, and an operational actor for further processing

Other Kepler features

- Can call external functions
- Can implement your own actors
- Incremental development for rapid prototyping
  - If inputs and outputs defined, can incorporate actors into workflow
  - Example – “dummy” composite actor
  - Components can be designed and tested separately

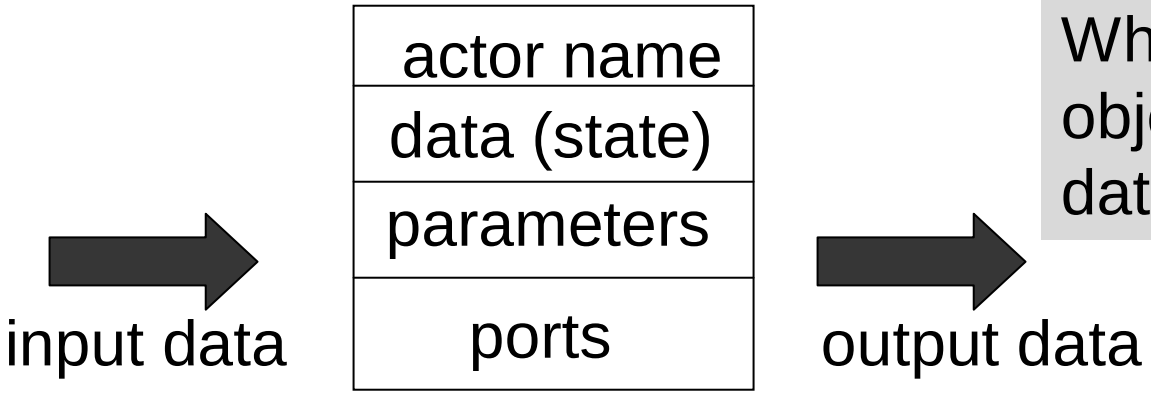
Focus on Actor-Oriented Design

Object orientation:



What flows through object is sequential control

Actor orientation:



What flows through object is streams of data

Adapted from B. Ludaescher

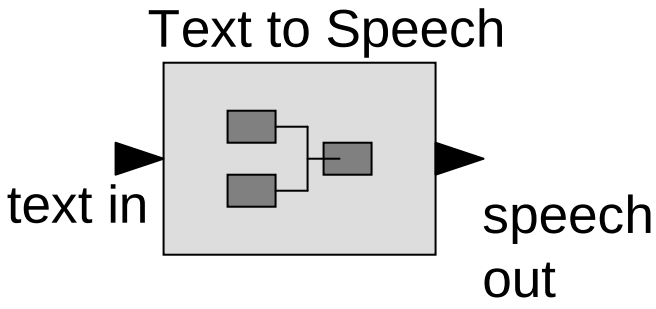
Object-Oriented vs. Actor Oriented Interface Definitions

Object oriented

TextToSpeech

`initialize(); void`
`notify(); void`
`isReady(); boolean`
`getSpeech(); double[]`

Actor oriented



OO interface definition gives procedures that have to be invoked in an order not specified as part of the interface definition

AO interface definition says “Give me text and I’ll give you speech”

Adapted from B. Ludaescher

Models of Computation

- Semantic interpretations of the abstract syntax
- Different models ⇔ Different semantic ⇔ Different execution
- One class: Producer/consumer
- Are actors active? Passive? Reactive?
- Are communications timed? Synchronized? Buffered?

Adapted from B. Ludaescher

Directors: Semantics for Component Interaction

- Some directors:
  - CT – continuous time modeling
  - DE – discrete event systems
  - FSM – finite state machines
  - **PN – process networks**
  - **SDF – synchronous dataflow**

Adapted from B. Ludaescher

Polymorphic Actors: Working Across Data Types and Domains

- Recall the add/subtract actor from last time
- Actor Data Polymorphism:
  - Add *numbers* (int, float, double, complex)
  - Add *strings* (concatenation)
  - Add *complex types* (arrays, records, matrices)
  - Add *user-defined types*

Adapted from B. Ludaescher

Polymorphic Actors (continued)

- Actor behavioral polymorphism
  - In *synchronous dataflow model* (SDF), add when all inputs have data
  - In *process networks*, execute infinite loop in a thread that blocks when reading empty inputs
  - In a *time-triggered model*, add when clock ticks

Adapted from B. Ludaescher

Benefits of Polymorphism

- Some observations:
  - Can define actors without defining input types
  - Can define actors without defining model of computation
- Why is this useful?
  - Increases reusability
  - But need to ensure that actor works in every circumstance

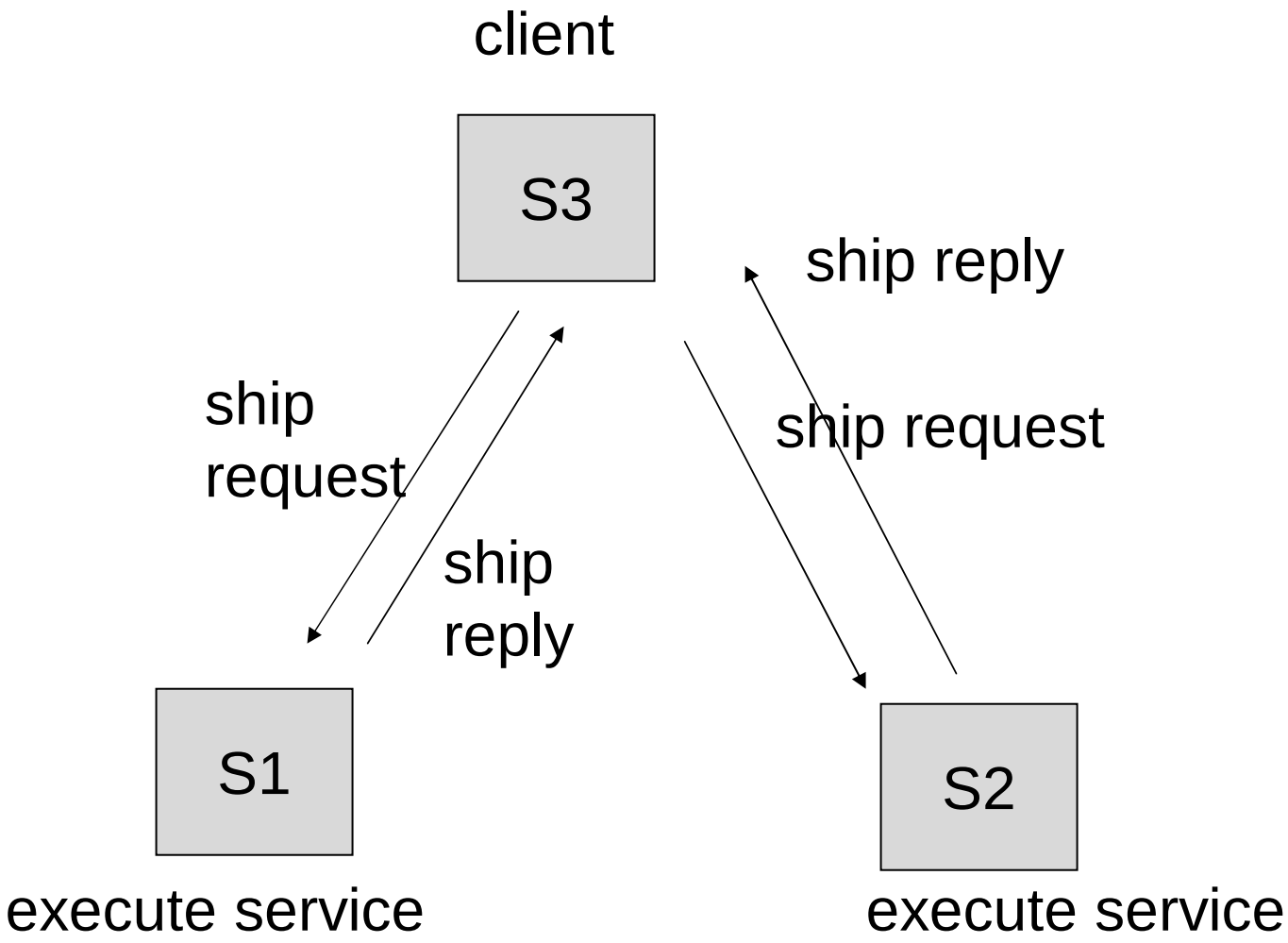
Actor Implementation

- Details beyond the scope of this class
- Idea: each actor implements several methods:
  - initialize() – initializes state variables
  - prefire() – indicates if actor wants to fire
  - fire() – main point of execution
    - Read inputs, produce outputs, read parameter values
  - postfire() – update persistent state, see if execution complete
  - wrapup()
- Each director call these methods according to its model

Third-party transfers

- Problem: Many workflows access data from one web service S1, pass the output on to service S2
- Current web services do not provide mechanism to transfer directly from S1 to S2
- Data is moved around more than necessary

Third party transfers



Handle-oriented approach

- Idea: instead of shipping actual data, web service send handle (pointer to data)
- Web services need support for handles

Scientific Workflow Examples

- Promoter Identification
- Mineral Classification
- Environmental Modeling
- Blast-ClustalW Workflow

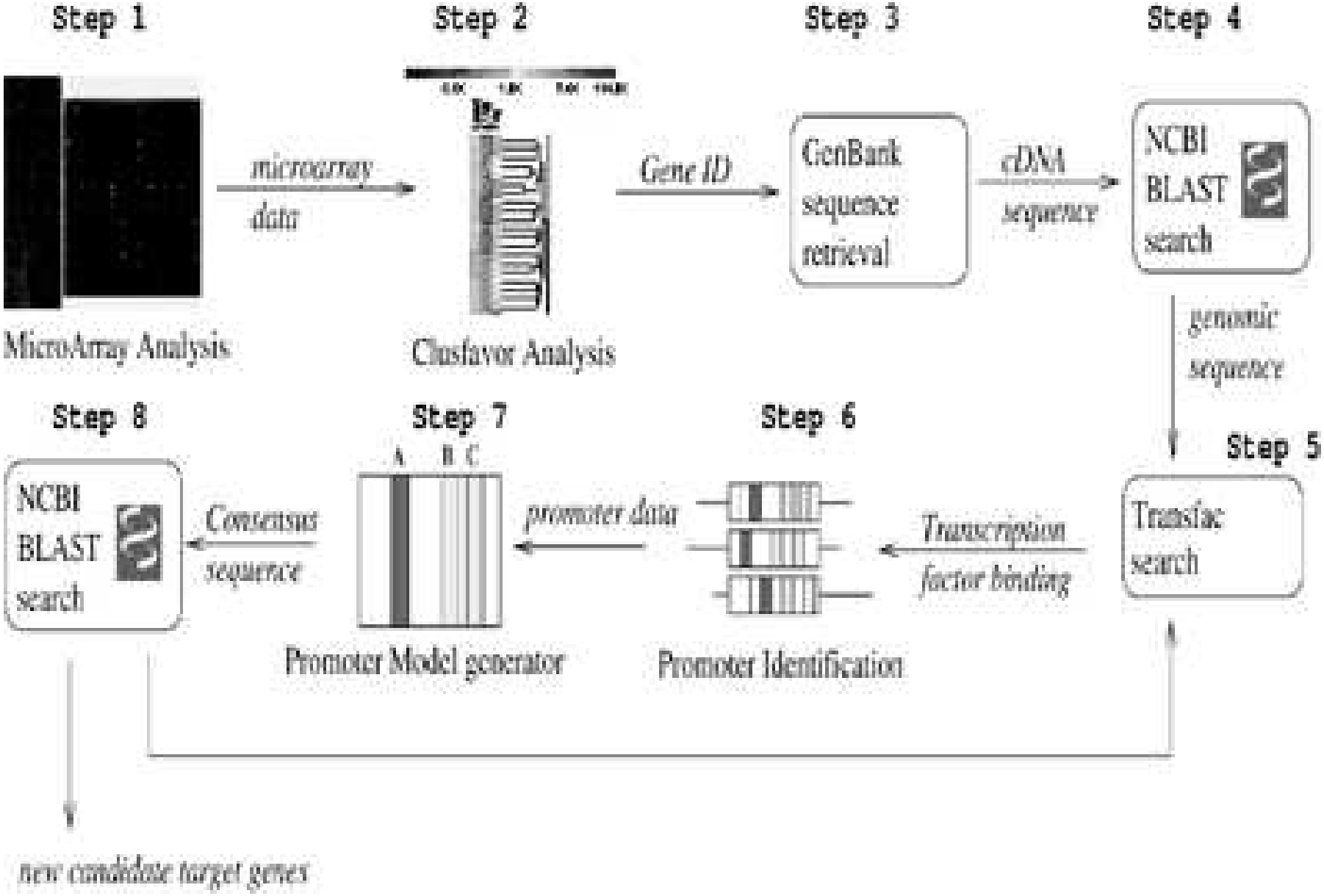
Promoter Identification Workflow

- Designed to help a biologist compare a set of genes that exhibit similar expression levels
- Goal: find the set of promoter modules responsible for this behavior
- Promoter is a subsequence of a chromosome that sits close to a gene and regulates its activity

Promoter Identification Workflow

1. Input – list of gene IDs
2. For each gene, construct likely upstream region by finding sequences that significantly overlap input gene
    1. Use GenBank to get sequence for each gene ID
    2. Use BLAST to find similar sequences
3. Find transcription factor binding sites in each of the sequences
    1. Run a Transfac search on sequence to identify binding sites
4. Align them and display
    1. Use ClustalW to align

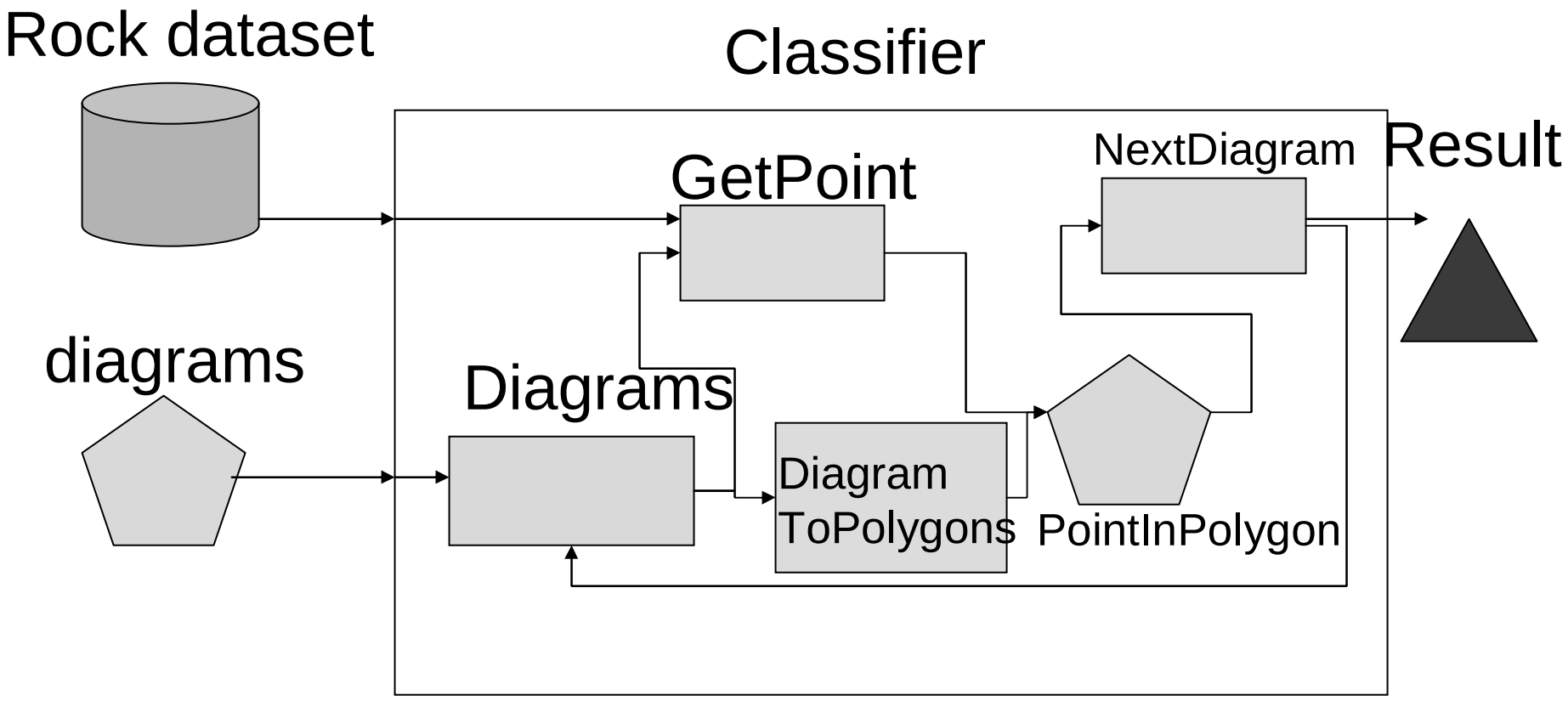
Promoter Identification Workflow



Mineral Classification Workflow

- Samples selected from a database holding mineral compositions of igneous rocks
- This data, along with set of classification diagrams, fed to a classifier
- Process of classifying samples involves determining position of sample values in series of diagrams
- When location of point in diagram of order n is determined, consult corresponding diagram of order n+1
- Repeat until terminal level of diagrams reached

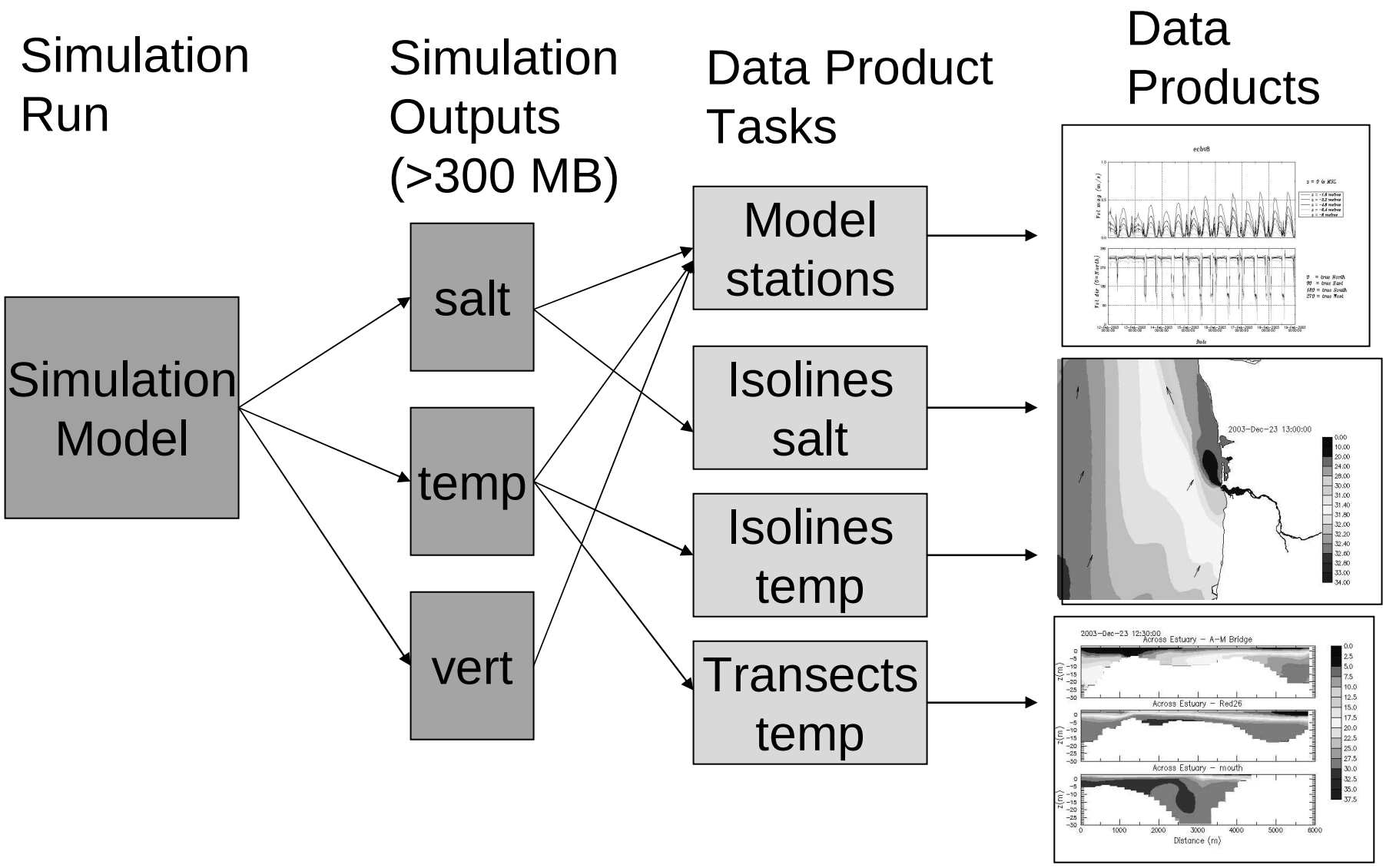
Mineral Classification Workflow



CORIE Environmental Observation and Forecasting System

- Daily forecasts of bodies of water throughout coastal United States
- Simulation program models physical properties of water (e.g., salinity, temperature, velocity)
- Scripts generate images, plots, and animations from raw simulation outputs

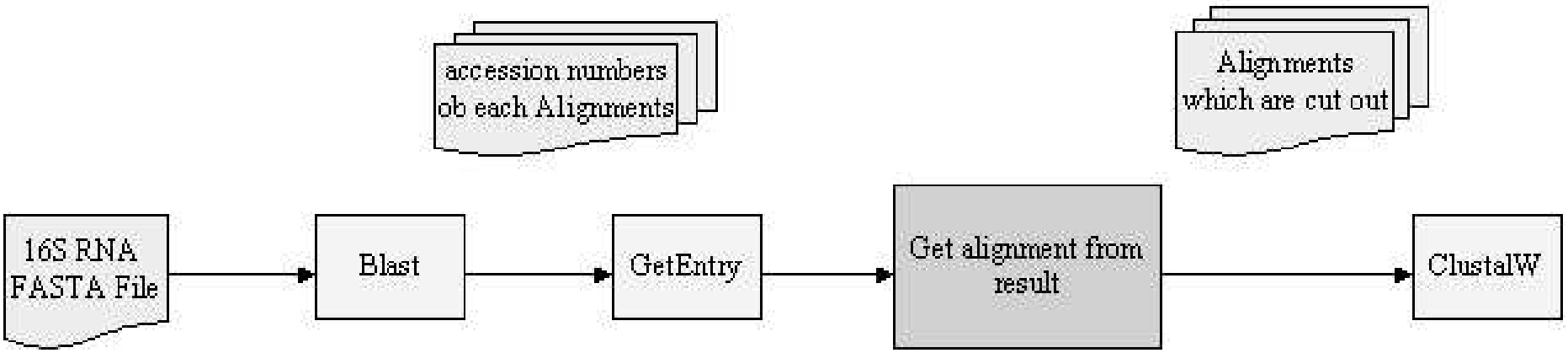
Example CORIE Workflows



Blast-ClustalW Workflow

Goal: Run BLASTN against DDBJ with a given DNA sequence, compare alignment regions of similar sequences using ClustalW

1. Run BLAST service with input sequence
2. Run GetEntry to get sequences of each hit
3. Cut off corresponding area
4. Run ClustalW



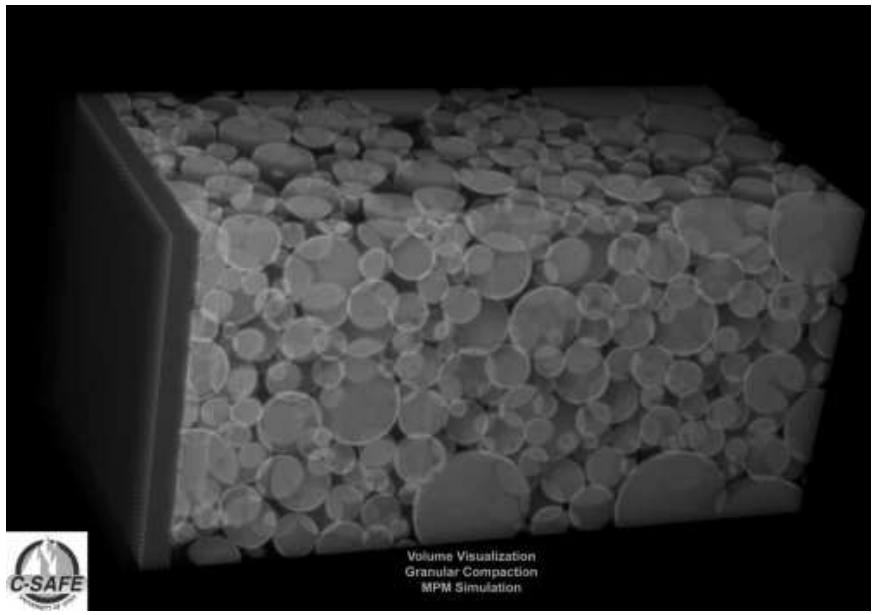
Some Scientific Workflow Tools

- Kepler
- SCIRun
- Triana
- Taverna
- Some commercial tools:
  - Windows Workflow Foundation
  - Mac OS X Automator

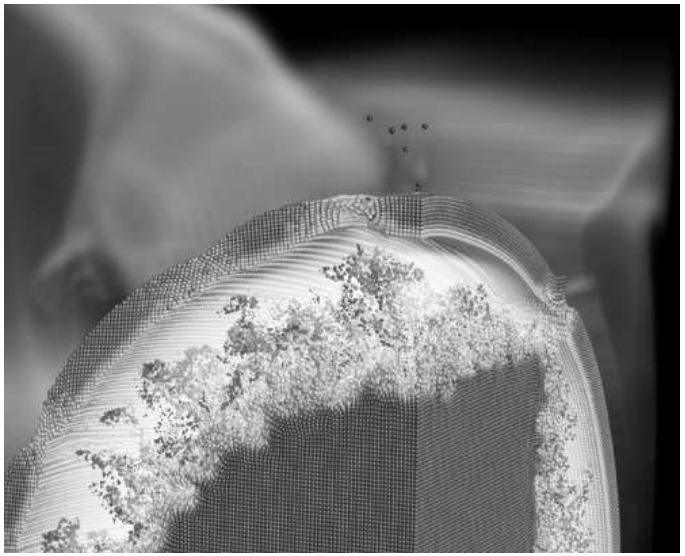
SciRun

- Computational workbench to interactively design and modify simulations
- Emphasis on visualization
- Scientists can interactively change models and parameters
- Fine-grained dataflow to improve computational efficiency

Some SCIRun Images



- Granular compaction simulation



- C-Safe Integrated Fire/Container Simulation

Triana

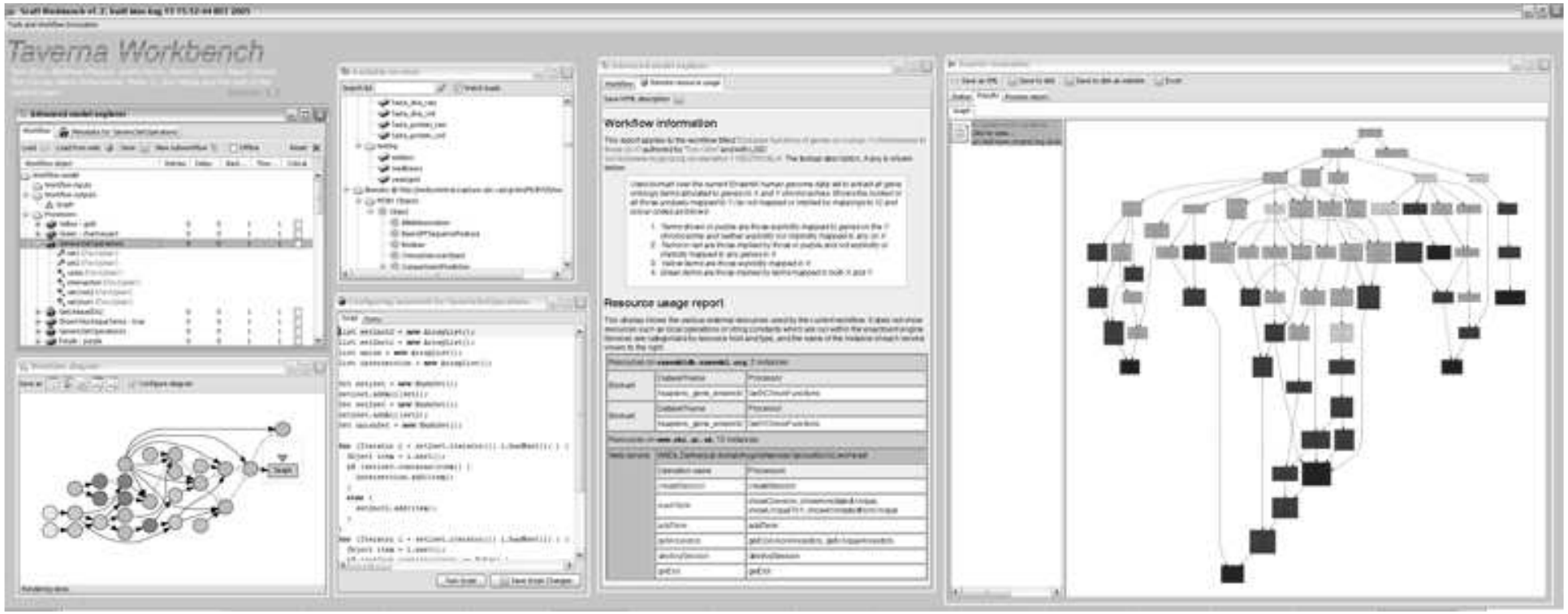
- Problem-solving environment combining visual interface and data analysis tools
- Emphasis on P2P and Grid computing environments
- Distributed functionality

Triana

Taverna

- Emphasis on bioinformatics workflows
- Enables coordination of local and remote resources
- Provides a GUI and access to bioinformatics web services
- Records provenance information

Taverna


Taverna Workbench
Workflow information
Resource usage report

A brief aside: BioMOBY

- **M**odel **O**rganism **B**ring **Y**our own Database
- Messaging standard to automatically discover and interact with biological data and service providers
- Automatic manipulation of data formats

BioMOBY (continued)

- Ontology of bio-informatics data types
- Define data syntax
- Create an open API over this ontology
- Define web service inputs/outputs
- Register services

- Many clients being deployed
- Clients for some workflow tools, e.g., Taverna in development

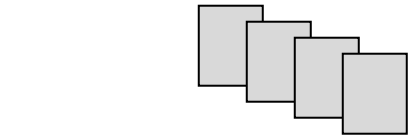
Executing Kepler on the Grid

- 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

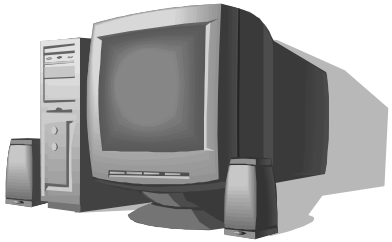
Example Grid Workflow

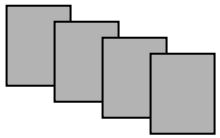
Stage-execute-fetch:

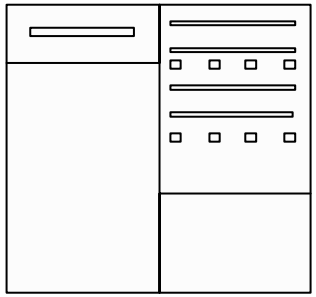
1. Stage local files to remote server



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