Simplify Your Science with Workflow Tools

International HPC Summer School
June 28, 2016

Scott Callaghan
Southern California Earthquake Center
University of Southern California
scottcal@usc.edu

SCEC an NSF+USGS center

Overview

- What are scientific workflows?
- What problems do workflow tools solve?
- Overview of available workflow tools
- CyberShake (seismic hazard application)
  - Computational overview
  - Challenges and solutions
- Ways to simplify your work
- Goal: Help you figure out if this would be useful

Scientific Workflows

- Formal way to express a scientific calculation
- Multiple tasks with dependencies between them
- No limitations on tasks
  - Short or long
  - Loosely or tightly coupled
- Capture task parameters, input, output
- Independence of workflow process and data
  - Often, run same workflow with different data
- You use workflows all the time…

Sample Workflow

```
#!/bin/bash
```

1) Stage-in input data to compute environment

```
scp myself@datastore.com:/data/input.txt /scratch/input.txt
```

2) Run a serial job with an input and output

```
bin/pre-processing in=input.txt out=tmp.txt
```

3) Run a parallel job with the resulting data

```
mpiexec bin/parallel-job in=tmp.txt out_prefix=output
```

4) Run a set of independent serial jobs in parallel – scheduling by hand

```
for i in `seq 0 $np`; do
      bin/integrity-check output.$i &
done
```

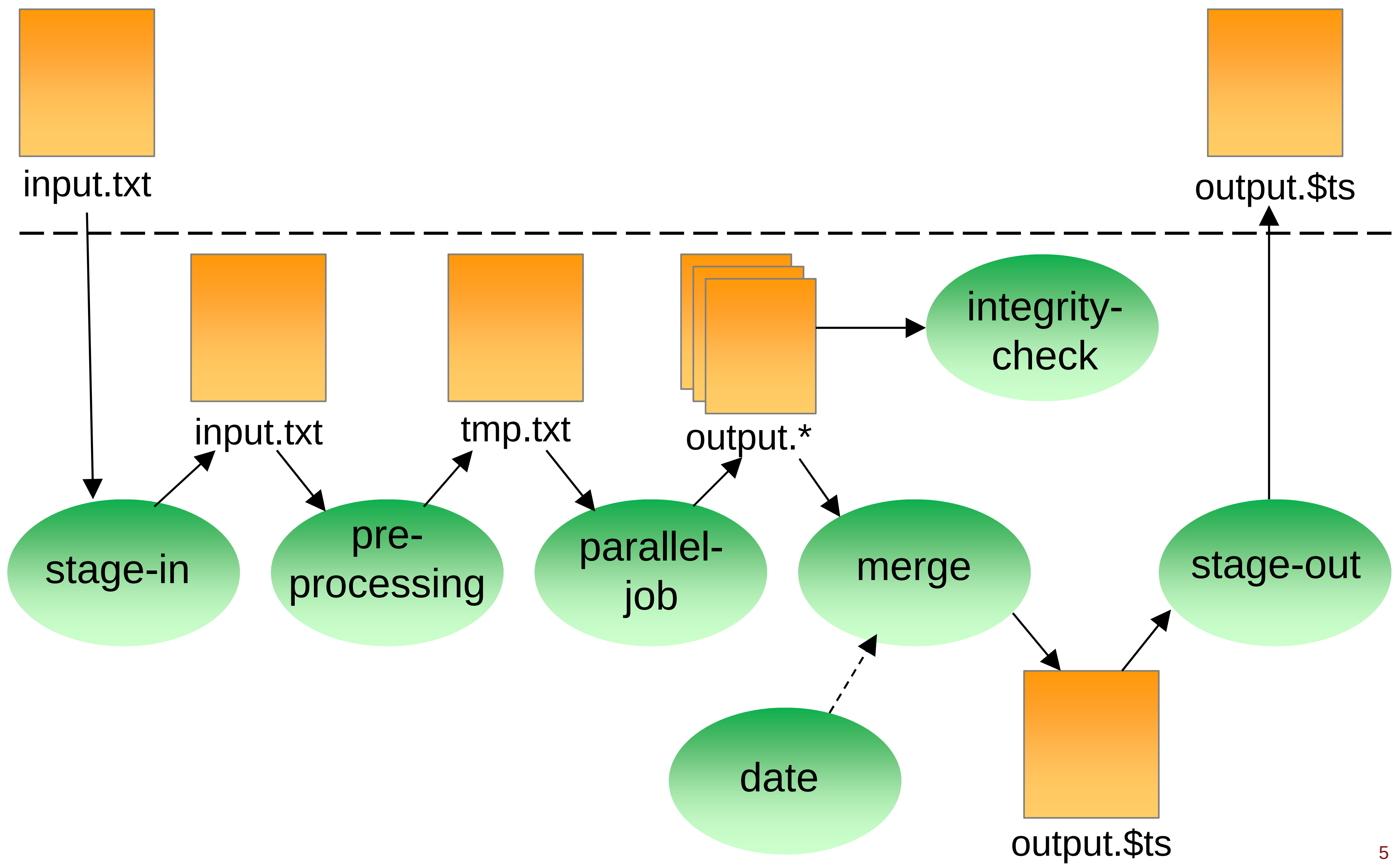
5) While those are running, get metadata and run another serial job

```
ts=`date +%s`
bin/merge prefix=output out=output.$ts
```

6) Finally, stage results back to permanent storage

```
scp /scratch/output.$ts myself@datastore.com:/data/output.$ts
```

Workflow schematic of shell script



Workflow Components

- Task executions
  - Specify a series of tasks to run
- Data and control dependencies between tasks
  - Outputs from one task may be inputs for another
- Task scheduling
  - Some tasks may be able to run in parallel with other tasks
- File and metadata management
  - Track when a task was run, key parameters
- Resource provisioning (getting cores)
  - Computational resources are needed to run jobs on

What do we need help with?

- Task executions
  - What if something fails in the middle?
- Data and control dependencies
  - Make sure inputs are available for tasks
  - May have complicated dependencies
- Task scheduling
  - Minimize execution time while preserving dependencies
- Metadata
  - Automatically capture and track
- Getting cores

Workflow Tools

- Define workflow via programming or GUI
- Can support all kinds of workflows
- Use existing code (no changes)
- Automate your pipeline
- Provide many kinds of fancy features and capabilities
  - Flexible but can be complex
- Will discuss one set of tools (Pegasus) as example, but concepts are shared

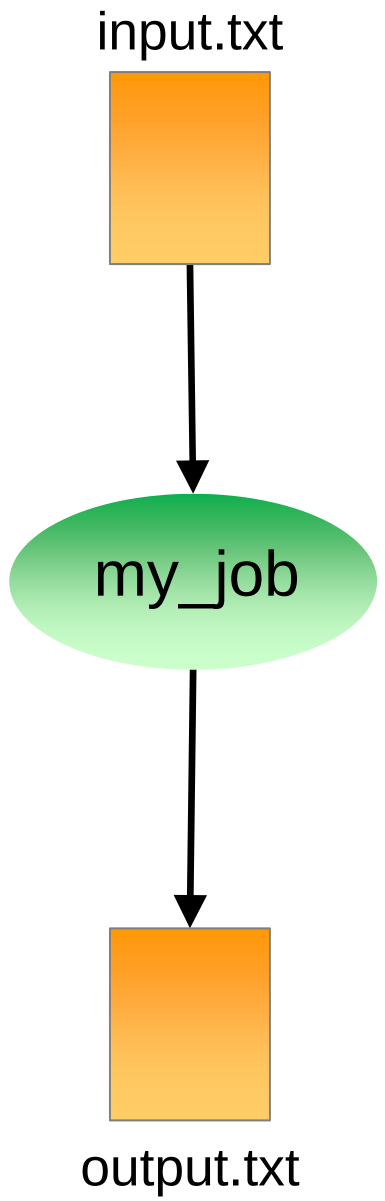
Pegasus-WMS

- Developed at USC’s Information Sciences Institute
- Used for many domains, including LIGO project
- Designed to address our earlier problems:
  - Task execution
  - Data and control dependencies
  - Data and metadata management
  - Error recovery
- Uses HTCondor DAGMan for
  - Task scheduling
  - Resource provisioning

Pegasus Concepts

- Separation of "submit host" and "execution site"
  - Create workflow using code on your local machine
  - Can run on local machine or on distributed resources
- Workflow represented with directed acyclic graphs
- You use API to write code describing workflow
  - Python, Java, Perl
  - Tasks with parent / child relationships
  - Files and their roles
  - Can have nested workflows
- Pegasus creates XML file of workflow called a DAX

Sample Workflow

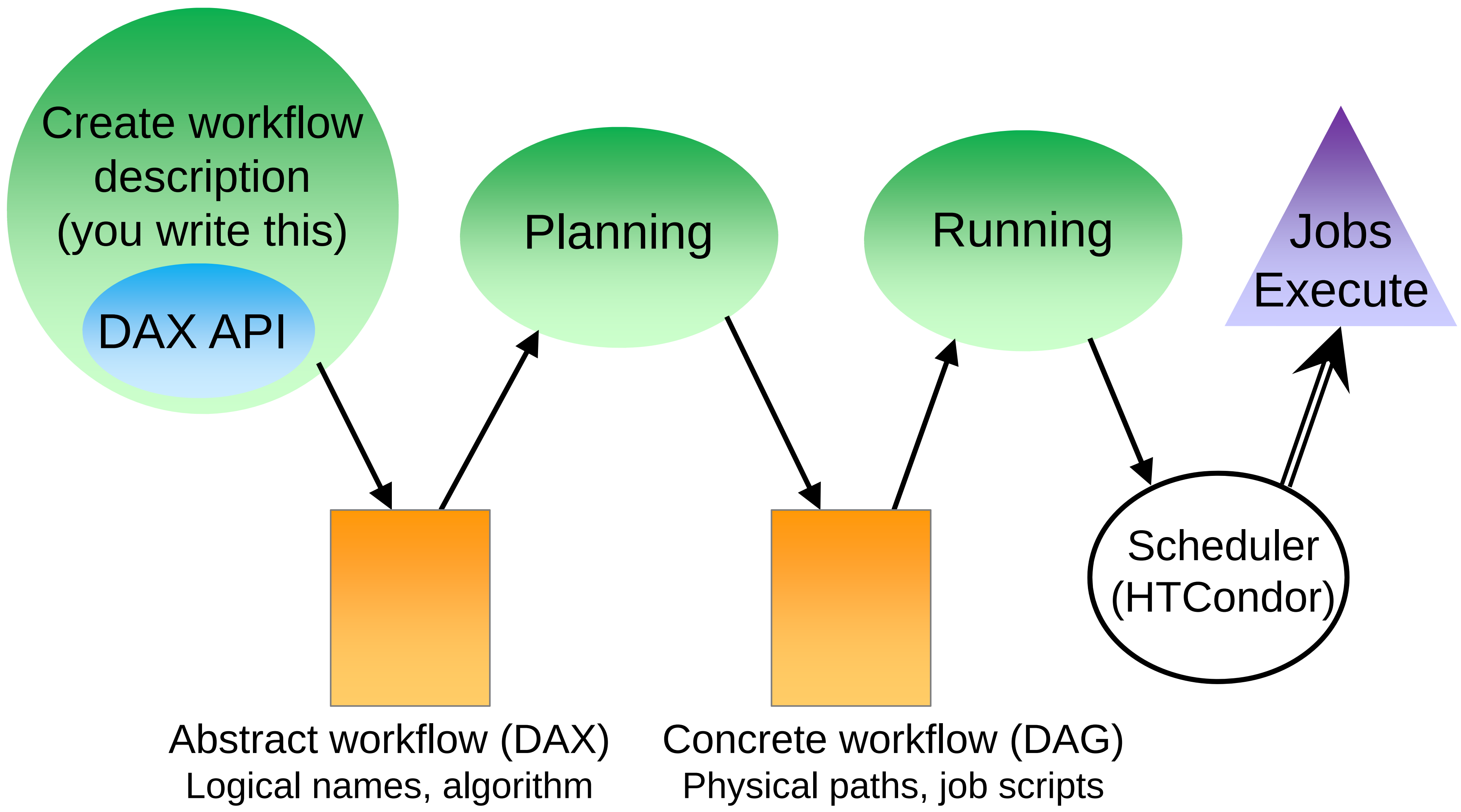


```
//Create DAX object
dax = ADAG("test_dax")
//Define my job
myJob = Job(name="my_job")
//Input and output files to my job
inputFile = File("input.txt")
outputFile = File("output.txt")
//Arguments to my_job (./my_job input=input.txt
   output=output.txt)
myJob.addArgument("input=input.txt",
   "output=output.txt")
//Role of the files for the job
myJob.uses(inputFile, link=Link.INPUT)
myJob.uses(outputFile, link=Link.OUTPUT)
//Add the job to the workflow
dax.addJob(myJob)
//Write to file
fp = open("test.dax", "w")
dax.writeXML(fp)
fp.close()
```

Planning

- DAX is "abstract workflow"
  - Logical filenames and executables
  - Algorithm description
- Prepare workflow to execute on a certain system
- Use Pegasus to "plan" workflow
  - Uses catalogs to resolve logical names, compute info
  - Pegasus automatically augments workflow
    - Staging jobs (if needed) with GridFTP or Globus Online
    - Registers output files in a catalog to find later
    - Wraps jobs in pegasus-kickstart for detailed statistics
  - Generates a DAG
    - Top-level workflow description (tasks and dependencies)
    - Submission file for each job (HTCondor format)

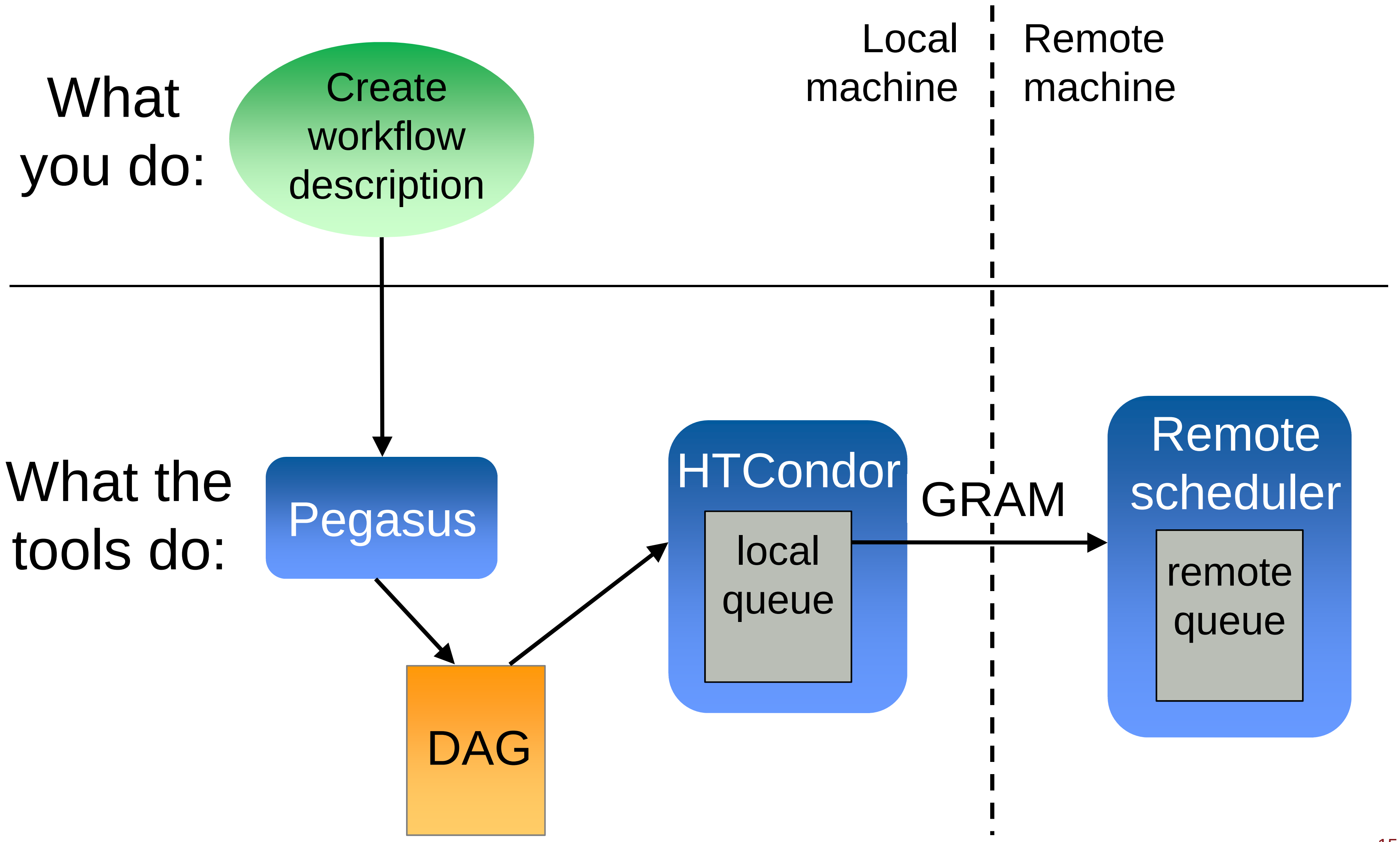
Pegasus Workflow Path



Other tools in stack

- HTCondor (UW Madison)
  - Pegasus ‘submits’ workflow to HTCondor
  - Supervises runtime execution of DAG files
    - Maintains queue
    - Monitors dependencies
    - Schedules jobs
    - Retries failures
    - Writes checkpoint
- GRAM (Globus Toolkit)
  - Uses certificate-based authentication for remote job submission
  - Supported by many HPC resources

Pegasus/HTCondor/GRAM stack



Other Workflow Tools

- Regardless of the tool, trying to solve same problems
  - Describe your workflow (Pegasus “Create”)
  - Prepare your workflow for the execution environment (Pegasus “Plan”)
  - Send jobs to resources (HTCondor, GRAM)
  - Monitor the execution of the jobs (HTCondor DAGMan)
- Brief overview of some other available tools

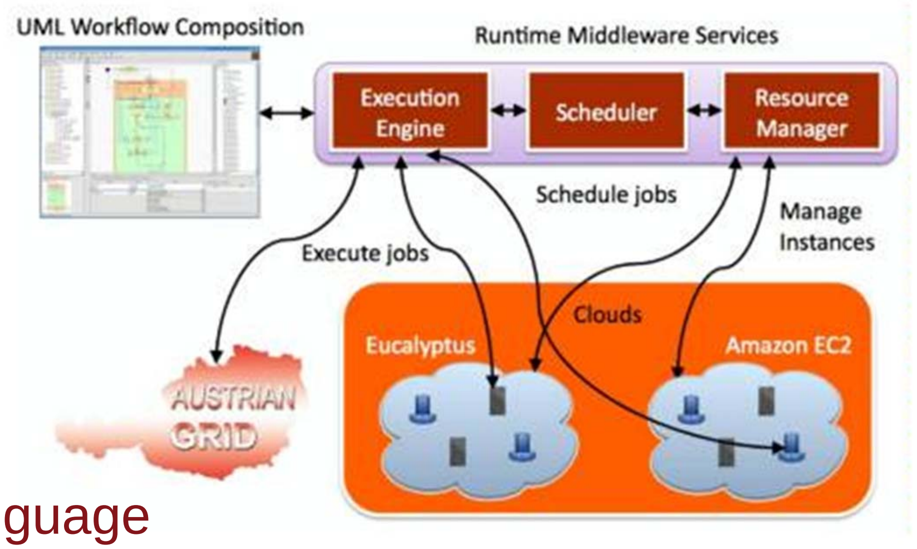
Other Workflow Tools: Swift

- Similar, but workflow defined via scripting language
- Developed at the University of Chicago

```
//Create new type
type messagefile;
//Create app definition, returns messagefile
app (messagefile t) greeting() {
    //Print and pipe stdout to t
    echo “Hello, world!” stdout=@filename(t);
}
//Create a new messagefile, linked to hello.txt
messagefile outfile <“hello.txt”>
//Run greeting() and store results
outfile = greeting();
```

- Workflow compiled internally and executed
- Focus on large data, many tasks

Other Workflow Tools: Askalon



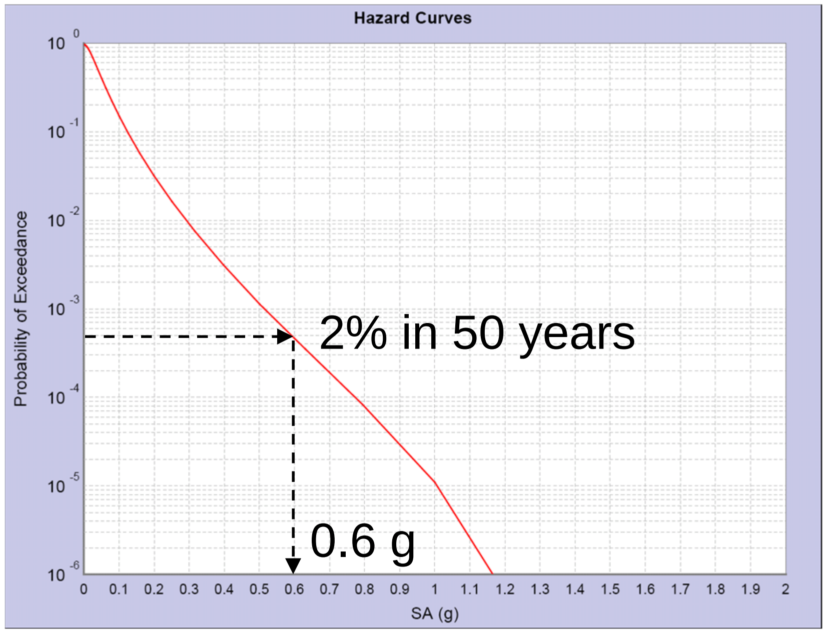
- Developed at University of Innsbruck
- Similar approach to Pegasus/HTCondor
  - Create workflow description
    - Either program in workflow language
    - Or use UML editor to graphically create
  - Conversion: like planning, to bind to specific execution
  - Submit jobs to Enactment Engine, which distributes jobs for execution at remote grid or cloud sites
  - Provides monitoring tools

Other Workflow Tools

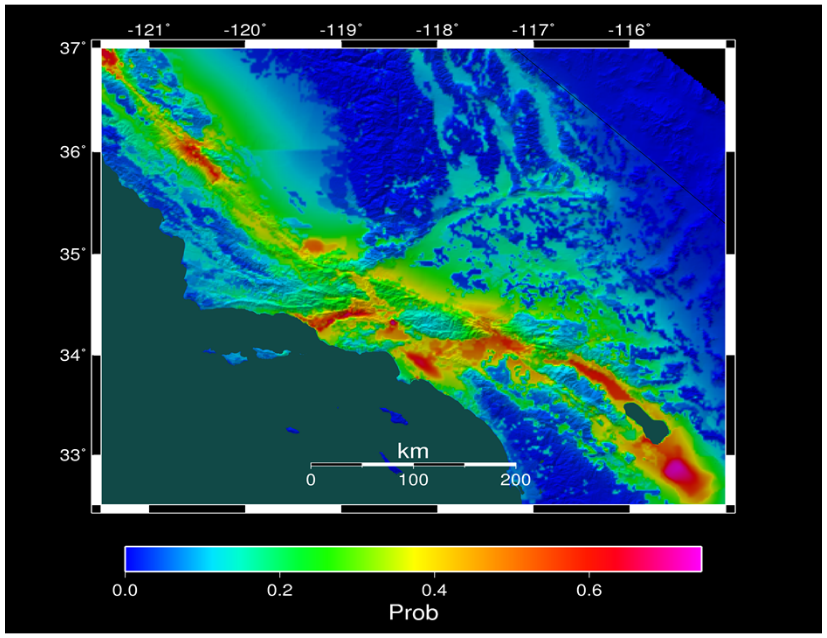
- Kepler (diverse US collaboration)
  - GUI interface
  - Many models of computation (‘actors’)
  - Many built-in components (tasks) already
- WS-PGRADE/gUSE (Hungarian Academy of Sciences)
  - WS-PGRADE is GUI interface to gUSE services
  - Supports “templates”, like OOP inheritance, for parameter sweeps
  - Interfaces with many architectures
- UNICORE (Jülich Supercomputing Center)
  - GUI interface to describe workflow
  - Branches, loops, parallel loops
- Many more: ask me about specific use cases

Workflow Application: CyberShake

- What will peak ground motion be over the next 50 years?
  - Used in building codes, insurance, government, planning
  - Answered via Probabilistic Seismic Hazard Analysis (PSHA)
  - Communicated with hazard curves and maps



Hazard curve for downtown LA



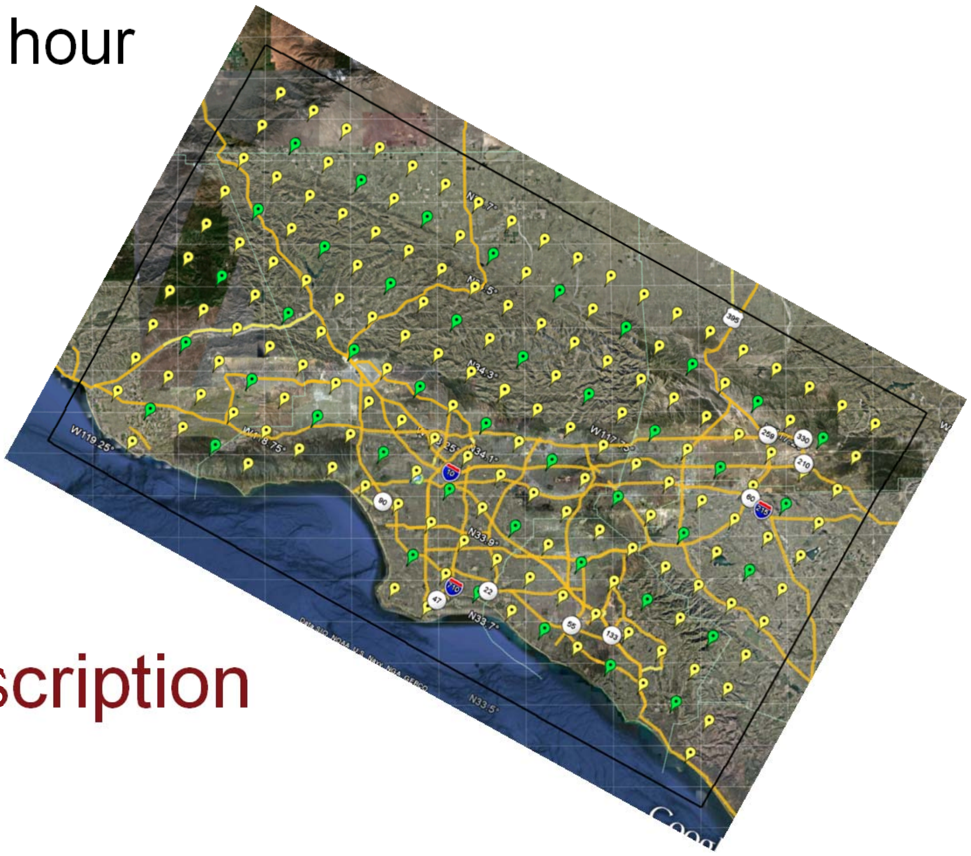
Probability of exceeding 0.1g in 50 yrs

CyberShake Computational Requirements

- Determine shaking due to ~500,000 earthquakes per site of interest
- Large parallel jobs
  - 2 GPU wave propagation jobs, 800 nodes x 1 hour
  - Total of 1.5 TB output
- Small serial jobs
  - 500,000 seismogram calculation jobs, 1 core x 4.7 minutes
  - Total of 30 GB output
- Need ~300 sites for hazard map

Why Scientific Workflows?

- Large-scale, heterogeneous, high throughput
  - Parallel and many serial tasks
  - Task duration 100 ms – 1 hour
- Automation
- Data management
- Error recovery
- Resource provisioning
- Scalable
- System-independent description



Challenge: Resource Provisioning

- For large parallel jobs, submit to remote scheduler
  - GRAM puts jobs in remote queue
  - Runs like a normal batch job
  - Can specify either CPU or GPU nodes
- For small serial jobs, need high throughput
  - Putting lots of jobs in the batch queue is ill-advised
    - Scheduler isn’t designed for heavy job load
    - Scheduler cycle is ~5 minutes
    - Policy limits too
- Solution: Pegasus-mpi-cluster (PMC)

Pegasus-mpi-cluster

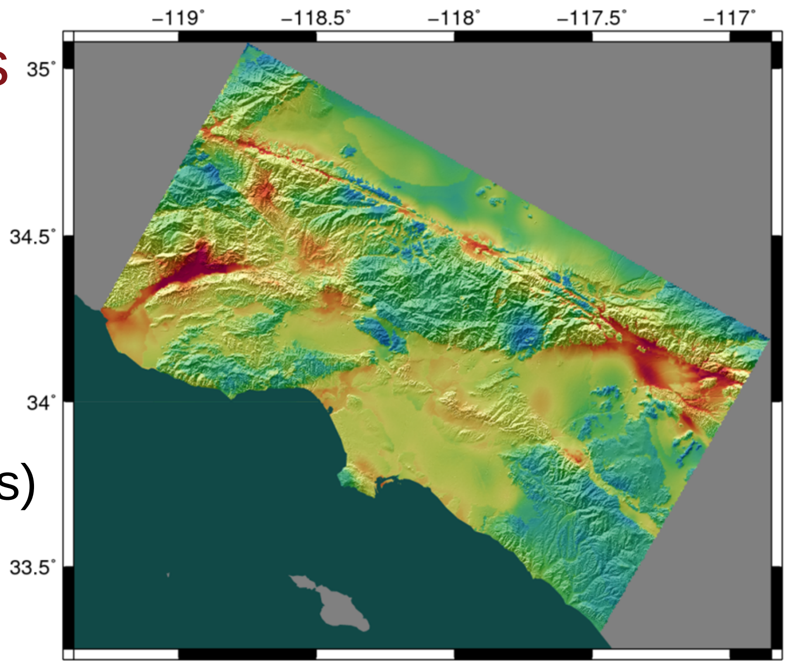
- MPI wrapper around serial or thread-parallel jobs
  - Master-worker paradigm
  - Preserves dependencies
  - HTCondor submits job to multiple nodes, starts PMC
  - Specify jobs as usual, Pegasus does wrapping
- Uses intelligent scheduling
  - Core counts
  - Memory requirements
  - Priorities
- Can combine writes
  - Workers write to master, master aggregates to fewer files

Challenge: Data Management

- Millions of data files
  - Pegasus provides staging
    - Symlinks files if possible, transfers files if needed
    - Supports running parts of workflows on separate machines
  - Transfers output back to local archival disk
  - Pegasus registers data products in catalog
  - Cleans up temporary files when no longer needed
- Directory hierarchy to reduce files per directory
- Added automated checks to check integrity
  - Correct number of files, NaN, zero-value checks
  - Included as new jobs in workflow

CyberShake Study 15.4

- Hazard curves for 336 sites
- Used OLCF Titan and NCSA Blue Waters
  - Pegasus transferred 408 TB of intermediate data
- Averaged 1962 nodes (CPUs and GPUs) for 35 days
  - Max of 20% of Blue Waters, 80% of Titan
- Generated 170 million seismograms
  - 4372 jobs in queues
- On average, 10 site workflows running concurrently
- Managed 1.1 PB of data
  - 7.7 TB staged back to local disk (~7M files)
- Workflow tools scale!



Problems Workflows Solve

- Task executions
  - Workflow tools will retry and checkpoint if needed
- Data management
  - Stage-in and stage-out data
  - Ensure data is available for jobs automatically
- Task scheduling
  - Optimal execution on available resources
- Metadata
  - Automatically track runtime, environment, arguments, inputs
- Getting cores
  - Whether large parallel jobs or high throughput

Should you use workflow tools?

- Probably using a workflow already
  - Replaces manual hand-offs and polling to monitor
- Provides framework to assemble community codes
- Scales from local computer to large clusters
- Provide portable algorithm description independent of data
- Does add additional software layers and complexity
  - Some development time is required

Final Thoughts

- Automation is vital
  - Eliminate human polling
  - Get everything to run automatically if successful
  - Be able to recover from common errors
- Put ALL processing steps in the workflow
  - Include validation, visualization, publishing, notifications
- Avoid premature optimization
- Consider new compute environments (dream big!)
  - Larger clusters, XSEDE/PRACE/RIKEN/CC, Amazon EC2
- Tool developers want to help you!

Links

- SCEC: http://www.scec.org
- Pegasus: http://pegasus.isi.edu
- Pegasus-mpi-cluster: http://pegasus.isi.edu/wms/docs/latest/cli-pegasus-mpi-cluster.php
- HTCondor: http://www.cs.wisc.edu/htcondor/
- Globus: http://www.globus.org/
- Swift: http://swift-lang.org
- Askalon: http://www.dps.uibk.ac.at/projects/askalon/
- Kepler: https://kepler-project.org/
- WS-PGRADE: https://guse.sztaki.hu/liferay-portal-6.0.5/
- UNICORE: http://www.unicore.eu/
- CyberShake: http://scec.usc.edu/scecpedia/CyberShake

Questions?

SCEC
an NSF+USGS center

ISI
Information Sciences Institute

XSEDE
Extreme Science and Engineering Discovery Environment

NSF National Science Foundation

SDSC
SAN DIEGO SUPERCOMPUTER CENTER

TACC

USGS
science for a changing world

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

NCSA

USCGRID