Simplify Your Science with Workflow Tools

International HPC Summer School
June 4, 2014

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
- 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
```

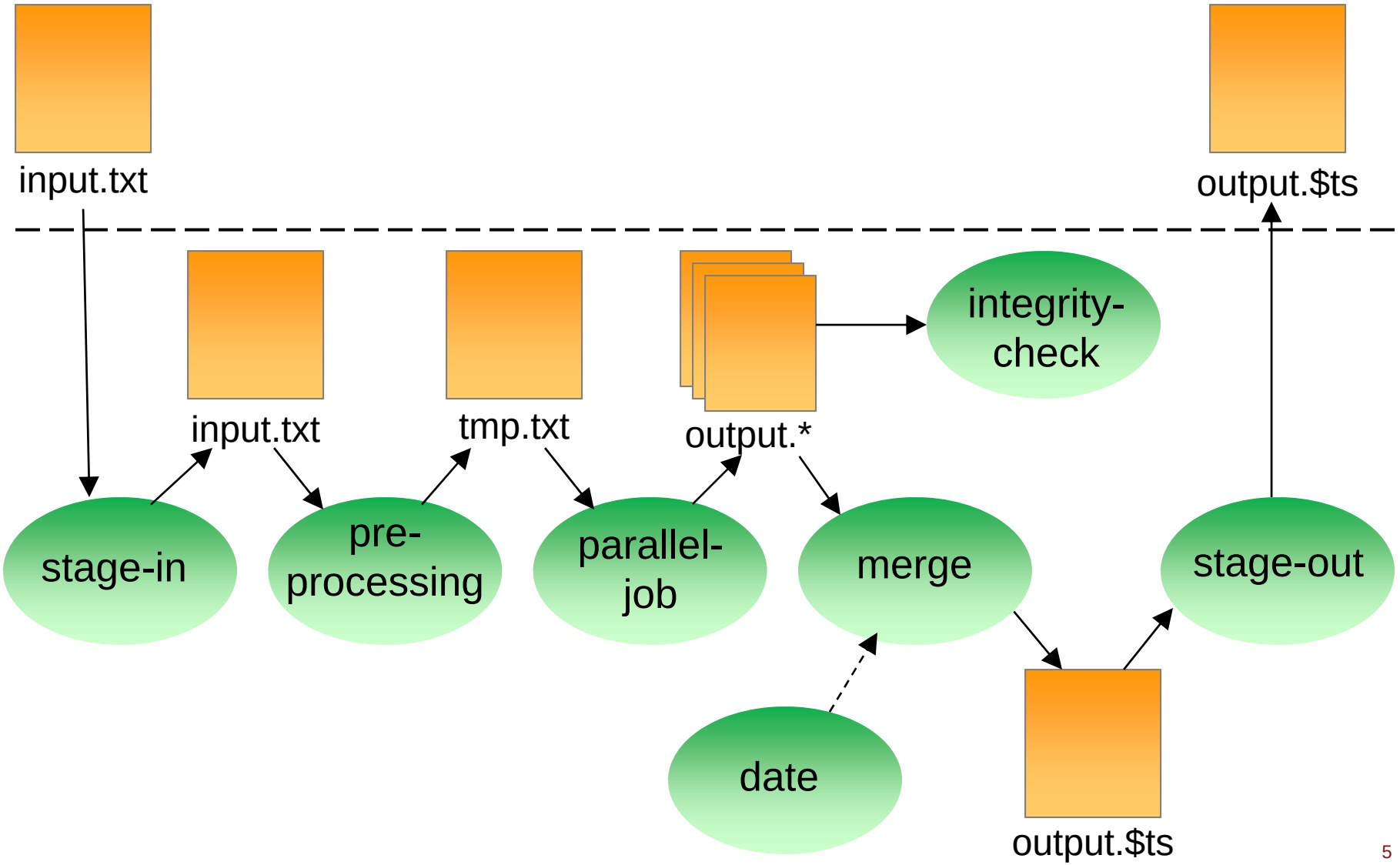
5) While those are running, get metadata and run another serial job

```
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
```

Could think of shell script as a workflow



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
- Can support all kinds of workflows
- Provide many kinds of fancy features and capabilities
  - Flexible but can be complex
- Will focus on Pegasus, but concepts are shared

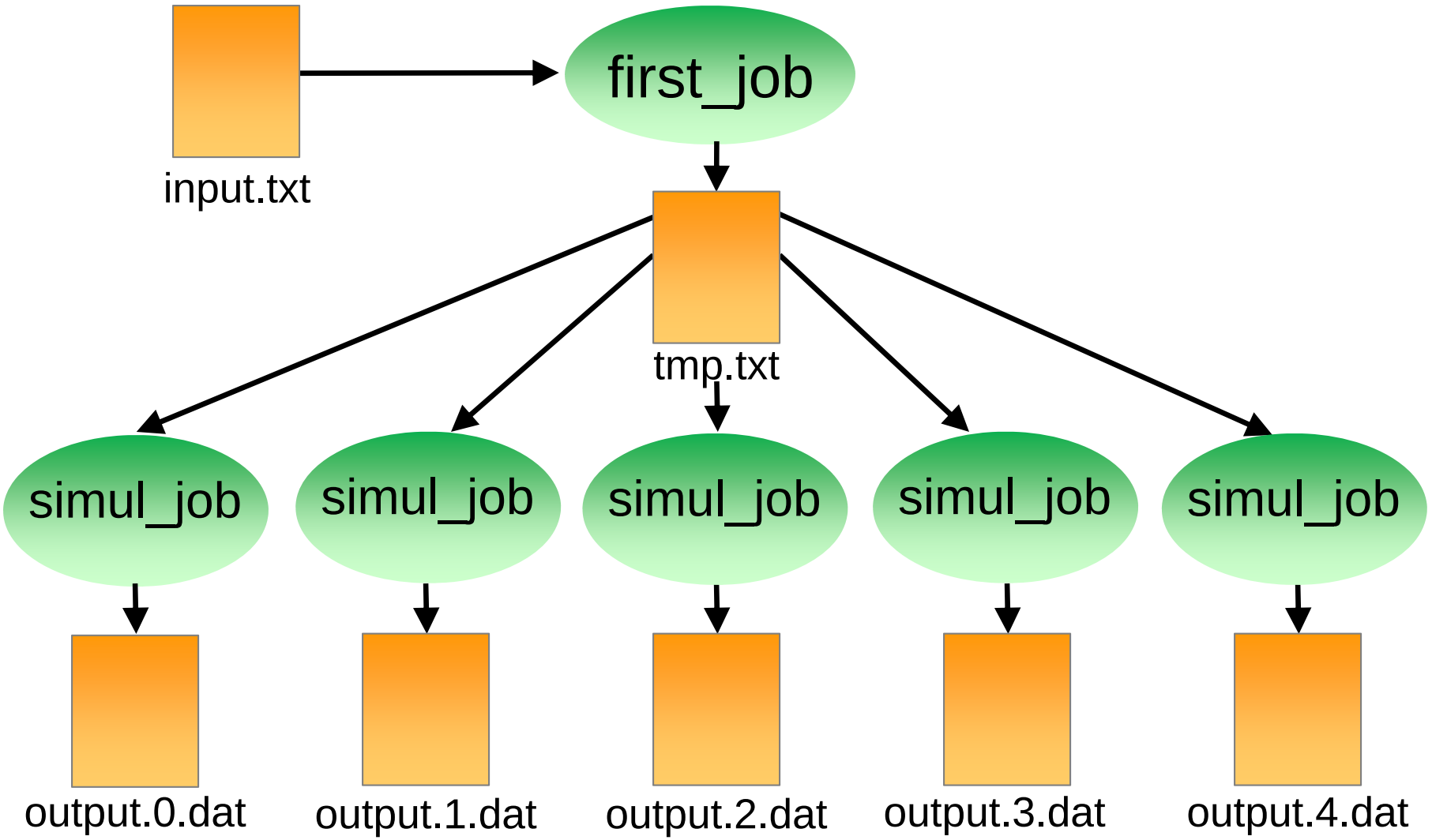
Pegasus-WMS

- Developed at USC's Information Sciences Institute
- 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



Sample DAX Generator

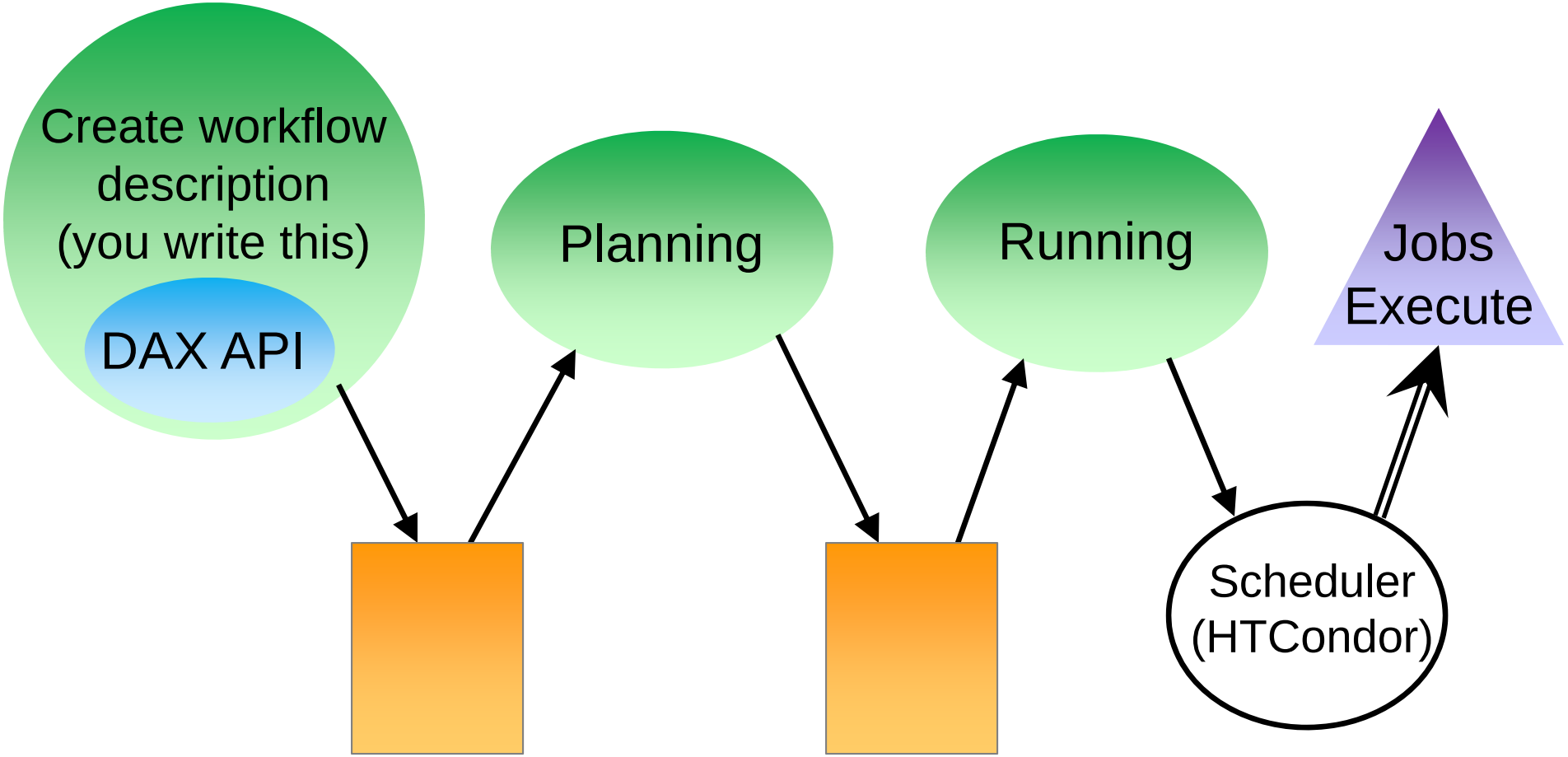
```
//Create DAX object
dax = ADAG("test_dax")
//Define first job
firstJob = Job(name="first_job")
//Input and output files to first job
firstInputFile = File("input.txt")
firstOutputFile = File("tmp.txt")
//Arguments to first_job (first_job input=input.txt output=tmp.txt)
firstJob.addArgument("input=input.txt", "output=tmp.txt")
//Role of the files for the job
firstJob.uses(firstInputFile, link=Link.INPUT)
firstJob.uses(firstOutputFile, link=Link.OUTPUT)
//Add the job to the workflow
dax.addJob(firstJob)
```


```
for i in range(0, 5):
    //Create simulation job
    simulJob = Job(id="%s" % (i+1), name="simul_job")
    //Define files
    simulInputFile = File("tmp.txt")
    simulOutputFile = File("output.%d.dat" % i)
    //Arguments to job
    //simulJob parameter=<i> input=tmp.txt output=output<i>.dat
    simulJob.addArgument("parameter=%d" % i, "input=tmp.txt",
                "output=%s" % simulOutputFile.getName())
    //Role of files
    simulJob.uses(simulInputFile, link=Link.INPUT)
    simulJob.uses(simulOutputFile, line=Link.OUTPUT)
    //Add job to dax
    dax.addJob(simulJob)
    //Dependency on firstJob
    dax.depends(parent=firstJob, child=simulJob)
//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
    - Stages 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



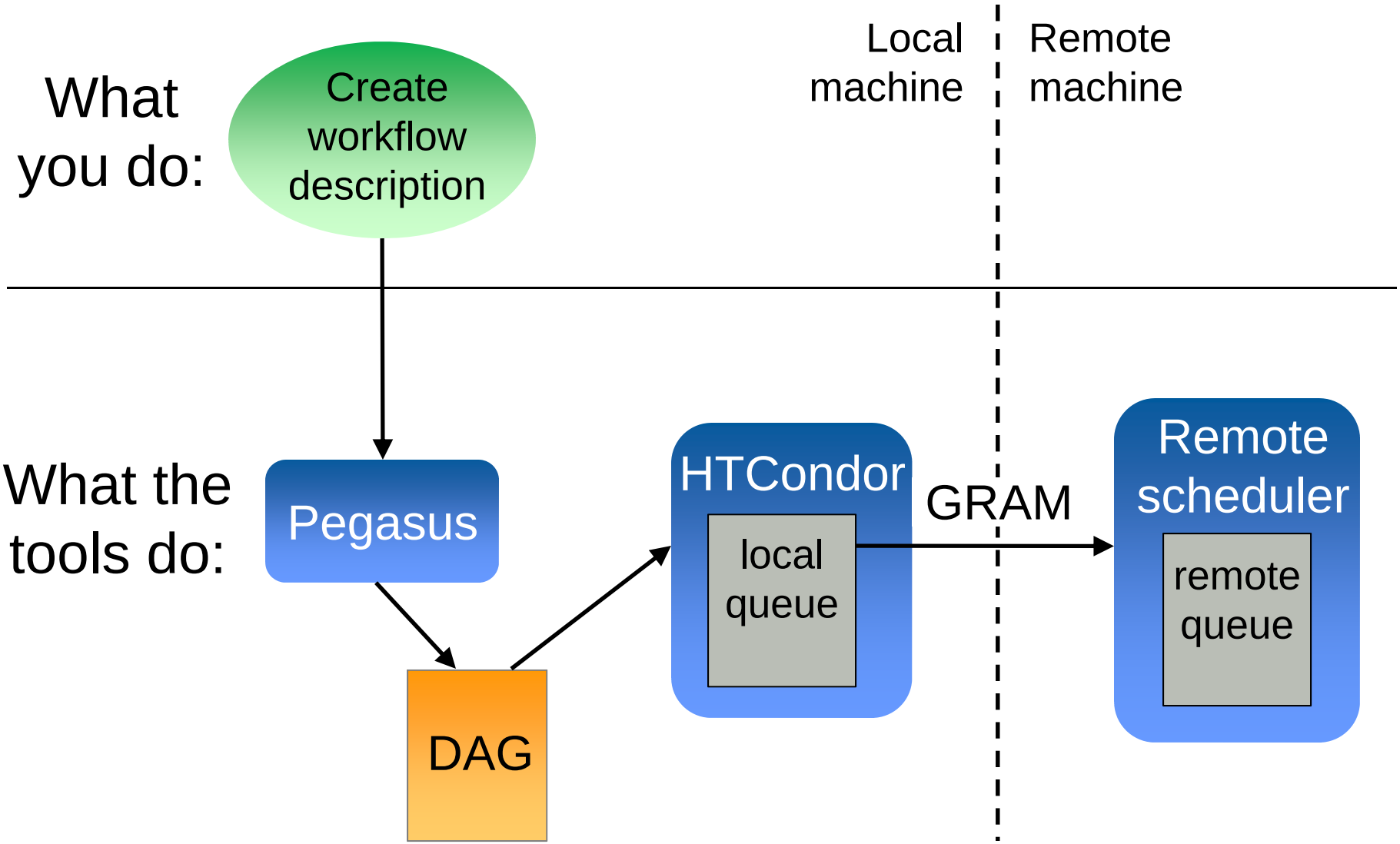
Running with HTCondor

- Developed by HTCondor group at U of Wisconsin
- Pegasus “submits” workflow to HTCondor DAGMan
  - Contains local queue of jobs
  - Monitors dependencies
  - Schedules jobs to resources
  - Automatically retries failed jobs
    - Writes rescue DAG to restart if job keeps failing
  - Updates status (jobs ready, complete, failed, etc.)
- Can run jobs locally or remotely (cluster, cloud)
  - HTCondor-G uses GRAM to submit jobs to remote scheduler

GRAM

- Part of the Globus Toolkit
- Uses certificate-based authentication
  - Like gsissh, GridFTP, Globus Online
  - Requires X509 certificate and account on remote machine
- Enables submission of jobs into a remote queue
- Supported by many HPC resources
  - Stampede, Blue Waters, SuperMUC

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 other available tools

Other Workflow Tools: Swift

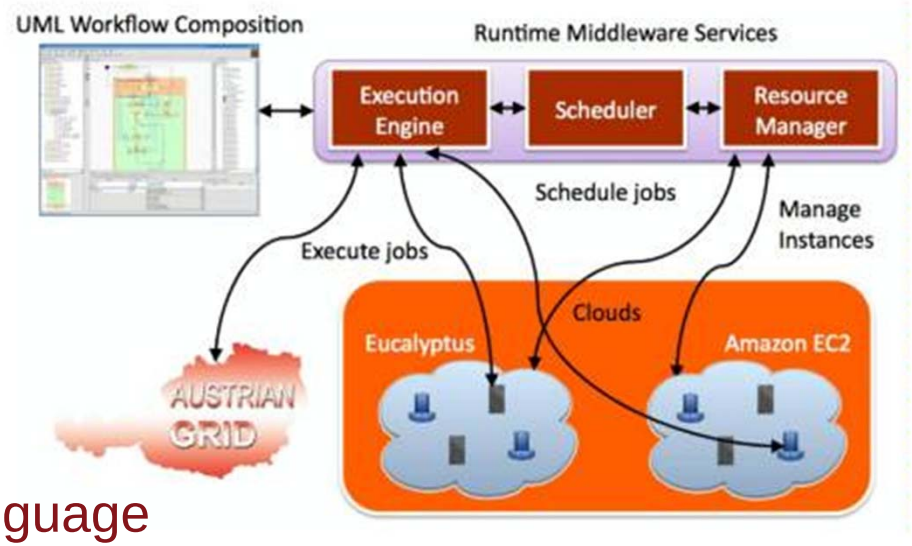
- Similar, but workflow defined via scripting language
- Developed at the University of Chicago

```
//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();
```

- Catalogs used to resolve executables and resources
- Workflow compiled internally and executed

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

- WS-PGRADE/gUSE
  - Developed at the Hungarian Academy of Sciences
  - WS-PGRADE is GUI interface to gUSE services
  - Supports “templates”, like OOP inheritance
  - Describe workflow, then configure it for execution
- UNICORE
  - Maintained by Jülich Supercomputing Center
  - GUI interface to describe workflow
  - Submit workflow to Gateway which manages execution
  - Gateway interfaces with schedulers over the Grid

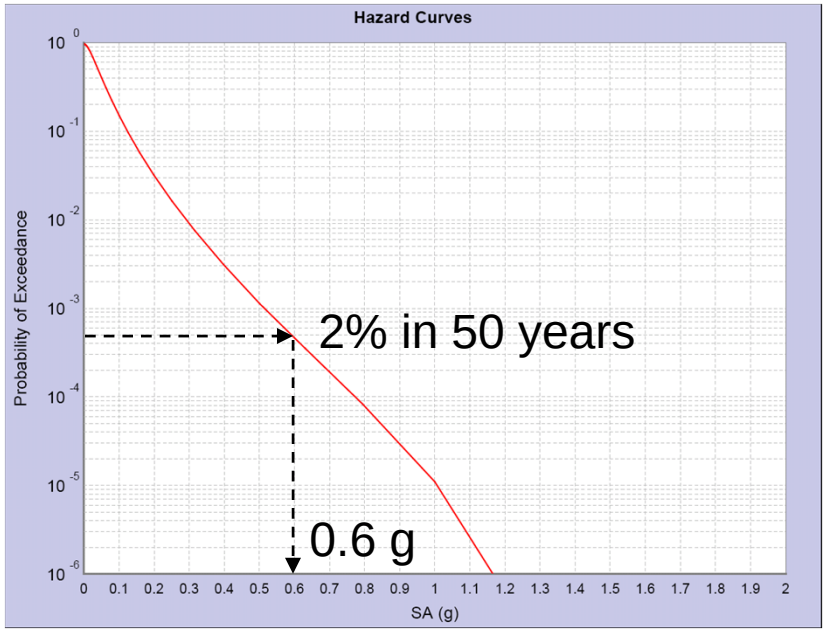
Other Remote Job Tools: SAGA

- These tools only submit remote jobs
  - You must check jobs for success and manage dependencies
- Developed at Rutgers University
- Python API for job submission and data transfer
- Interfaces with many scheduler types

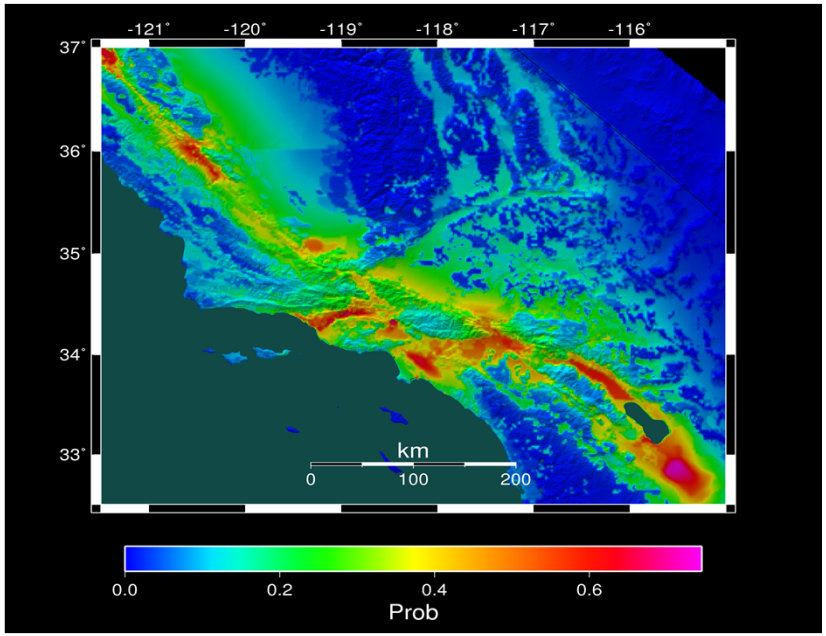
```
ctx = saga.Context("ssh“) # Use SSH for authentication
ctx.user_id = "your_username“ # Your identity
session = saga.Session() # Create new session
session.add_context(ctx) # Link session with authentication
# Use a PBS adaptor
js = saga.job.Service("pbs+ssh://%s“ % REMOTE_HOST,session=session)
jd = saga.job.Description() # Create description
jd.executable = '/bin/ls‘ # Could include environment, input, output
myjob = js.create_job(jd) # Create job with description and service
myjob.run() # Submit job
myjob.wait() # Wait for completion or error
```

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



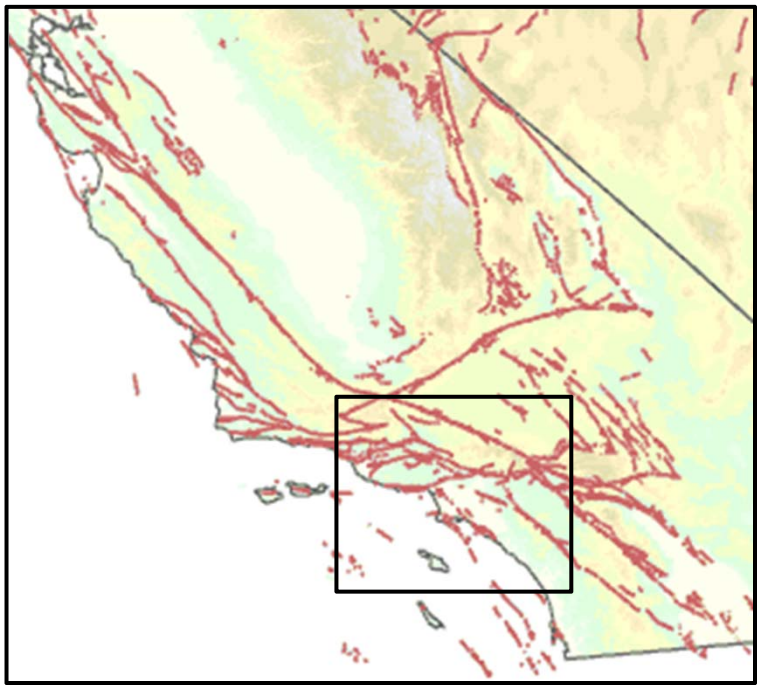
Hazard curve for downtown LA

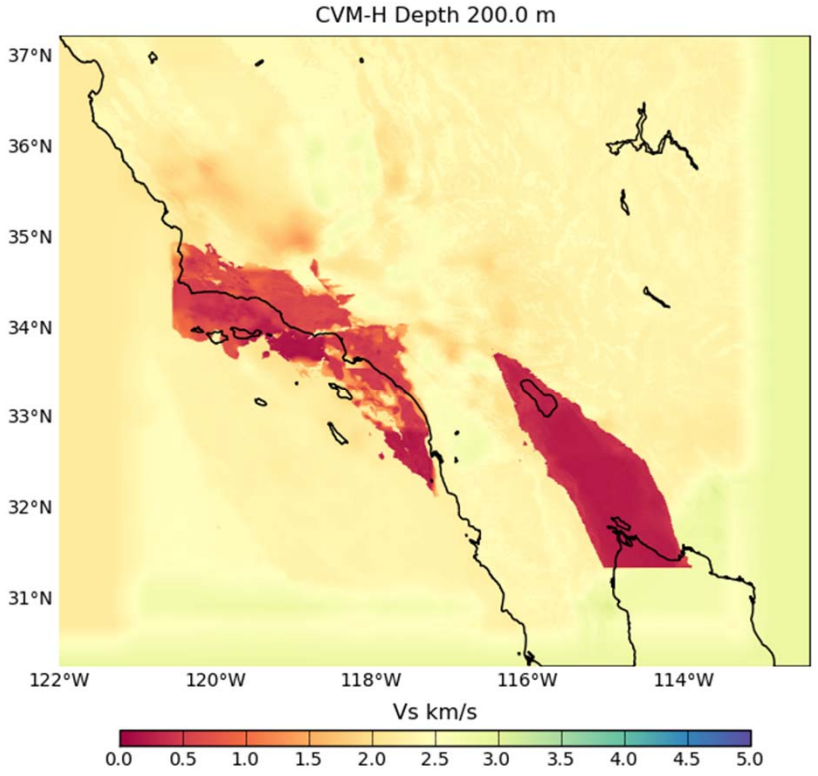


Probability of exceeding 0.1g in 50 yrs

Seismic Hazard Analysis Calculation

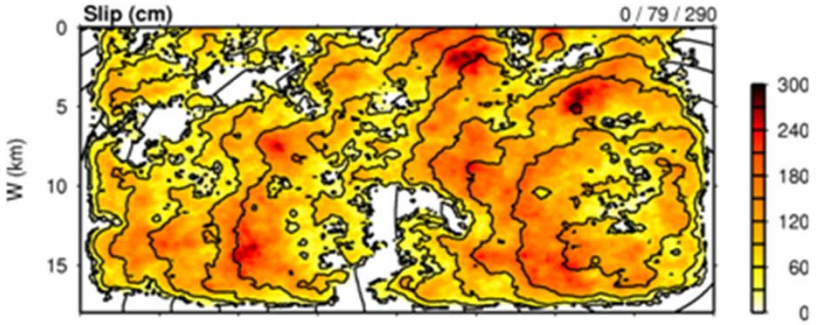
- Tensor simulation
  - Create 1.5 billion point mesh with material properties
  - Generate tensors across volume
  - Parallel, ~8,000 CPU-hrs

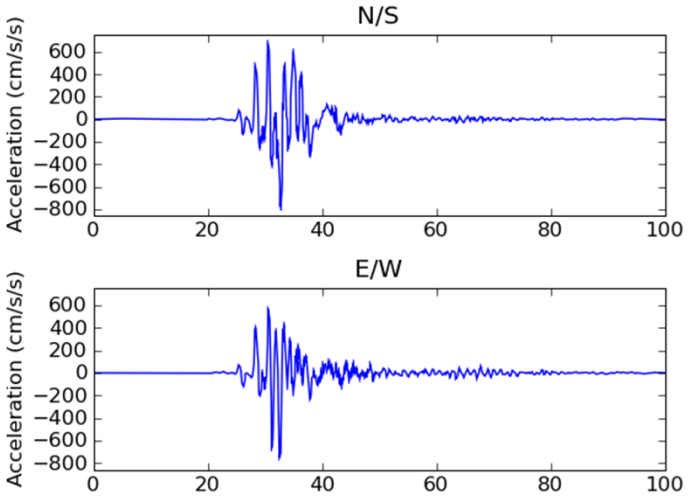




Post-Processing

- Individual earthquake contributions
  - Get list of earthquakes of interest (~415,000)
  - Simulate seismograms for each earthquake
  - Loosely-coupled, short-running serial jobs
- Combine the levels of shaking with probabilities to produce a hazard curve.





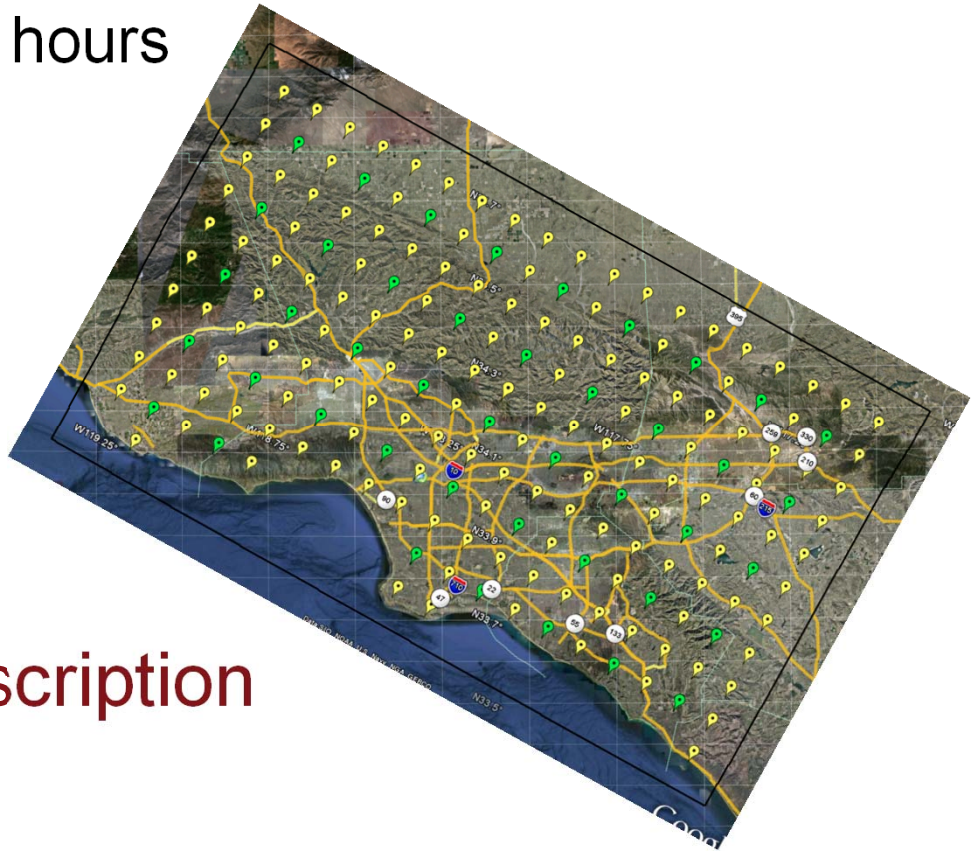
Computational Requirements

	Component	Data	Executions	Cores/exec	Core hours
Tensor Creation	Mesh generation	15 GB	1	320	50
	Tensor simulation	40 GB	2	10,000 CPU 100 GPU	16,000 CPU 1,200 GPU
Post Processing	Tensor extraction	690 GB	6	256	275
	Seismogram synthesis	12 GB	**415,000**	1	2,300
	Curve generation	1 MB	1	1	< 1
	Total	757 GB	415,000		18,625

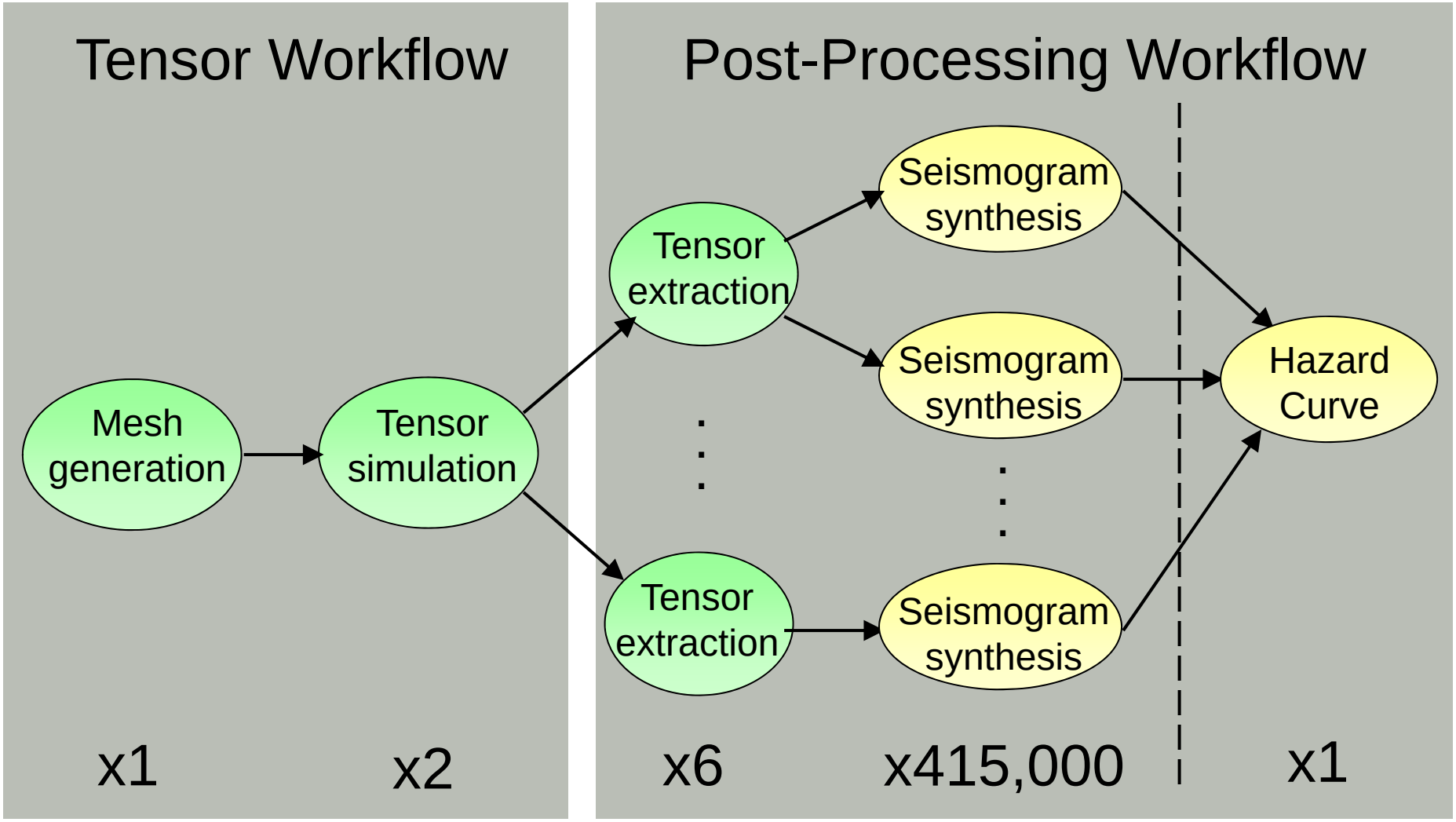
This is for **one** location of interest; we wanted to run ~1000

Why Scientific Workflows?

- Large-scale, heterogeneous, high throughput
  - Parallel and many (~415,000) serial tasks
  - Task duration 100 ms – 2 hours
- Automation
- Data management
- Error recovery
- Resource provisioning
- Scalable
- System-independent description



CyberShake workflows



Challenge: Resource Provisioning

- In tensor workflow, submit job to remote scheduler
  - GRAM puts jobs in remote queue
  - Runs like a normal batch job
  - Can specify either CPU or GPU nodes
- For post-processing workflow, 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
  - Locality preferences under development

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

Challenge: File System Load

- Seismogram tasks cause heavy I/O load
  - Reads an earthquake description
  - Writes a seismogram file
- Reduce reads
  - Generate earthquake description on the fly, from geometry
  - Added memcached to cache rupture geometry
    - Local memory cache on compute node
    - Pegasus-mpi-cluster hook for custom startup script
- Reduce writes
  - Pegasus-mpi-cluster supports “pipe forwarding”
  - Workers write to pipes, master aggregates to fewer files

CyberShake Study 14.2

- Hazard curves for 1144 sites
- 46,720 CPUs + 225 GPUs for 14 days (Blue Waters)
  - Peak of 295,040 CPUs, 1100 GPUs
- 99.8 million tasks executed
  - 81 tasks/sec
  - Only 31,463 jobs in Blue Waters queue
- On average, 26.2 workflows running concurrently
- Managed 830 TB of data
  - 57 TB output files
  - 12.3 TB staged back to local disk (~16M files)
- Workflow tools scale!

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

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

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, 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/
- WS-PGRADE: https://guse.sztaki.hu/liferay-portal-6.0.5/
- UNICORE: http://www.unicore.eu/
- SAGA: http://saga-project.github.io/saga-python/
- CyberShake: http://scec.usc.edu/scecpedia/CyberShake

Questions?

SCEC
an NSF+USGS center

ISI
Information Sciences Institute

XSEDE
Extreme Science and Engineering Discovery Environment

National Science Foundation

SDSC
SAN DIEGO SUPERCOMPUTER CENTER

NICS

TACC

USGS
science for a changing world

OLCF
Oak Ridge Leadership Computing Facility

NCSA

USCGRID