

# Toward a Common Model for Highly Concurrent Applications

Douglas Thain  
University of Notre Dame

MTAGS Workshop  
17 November 2013

# Overview

- **Experience with Concurrent Applications**
  - Makeflow, Weaver, Work Queue
- Thesis: Convergence of Models
  - Declarative Language
  - Directed Graphs of Tasks and Data
  - Shared Nothing Architecture
- Open Problems
  - Transaction Granularity
  - Where to Parallelize?
  - Resource Management
- Concluding Thoughts

# The Cooperative Computing Lab

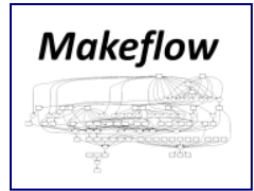
*University of Notre Dame*



<http://www.nd.edu/~ccl>

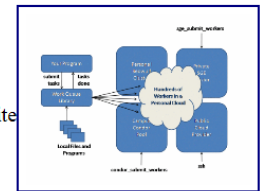
## Makeflow

Makeflow is a workflow system for parallel and distributed computing that uses a language very similar to Make. Using Makeflow, you can write simple scripts that easily execute on hundreds or thousands of machines.



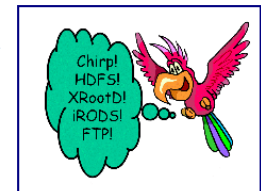
## Work Queue

Work Queue is a system and library for creating and managing scalable master-worker style programs that scale up to thousands machines on clusters, clouds, and grids. Work Queue programs are easy to write in C, Python or Perl.



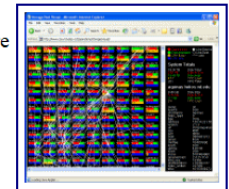
## Parrot

Parrot is a transparent user-level virtual filesystem that allows any ordinary program to be attached to many different remote storage systems, including HDFS, iRODS, Chirp, and FTP.



## Chirp

Chirp is a personal user-level distributed filesystem that allows unprivileged users to share space securely, efficiently, and conveniently. When combined with Parrot, Chirp allows users to create custom wide-area distributed filesystems.

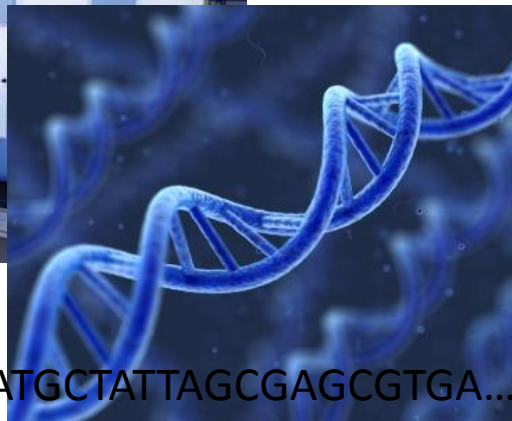
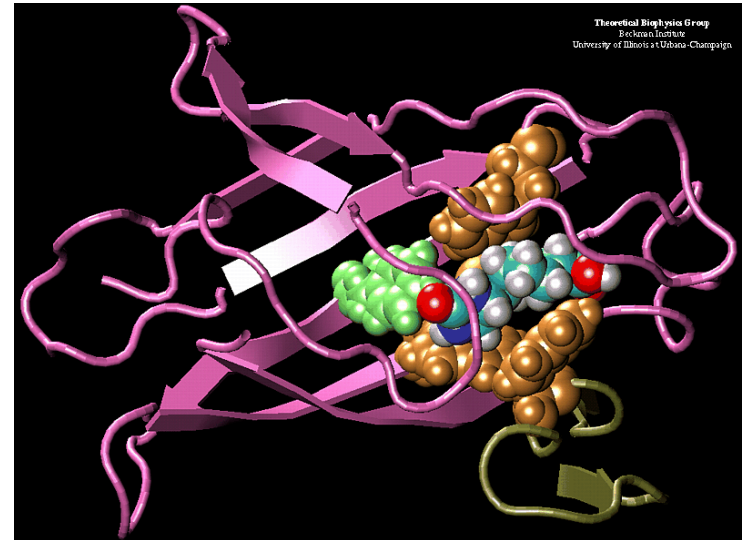


# The Cooperative Computing Lab

- We *collaborate with people* who have large scale computing problems in science, engineering, and other fields.
- We *operate computer systems* on the O(10,000) cores: clusters, clouds, grids.
- We *conduct computer science* research in the context of real people and problems.
- We *release open source software* for large scale distributed computing.

<http://www.nd.edu/~ccl>

# Our Collaborators

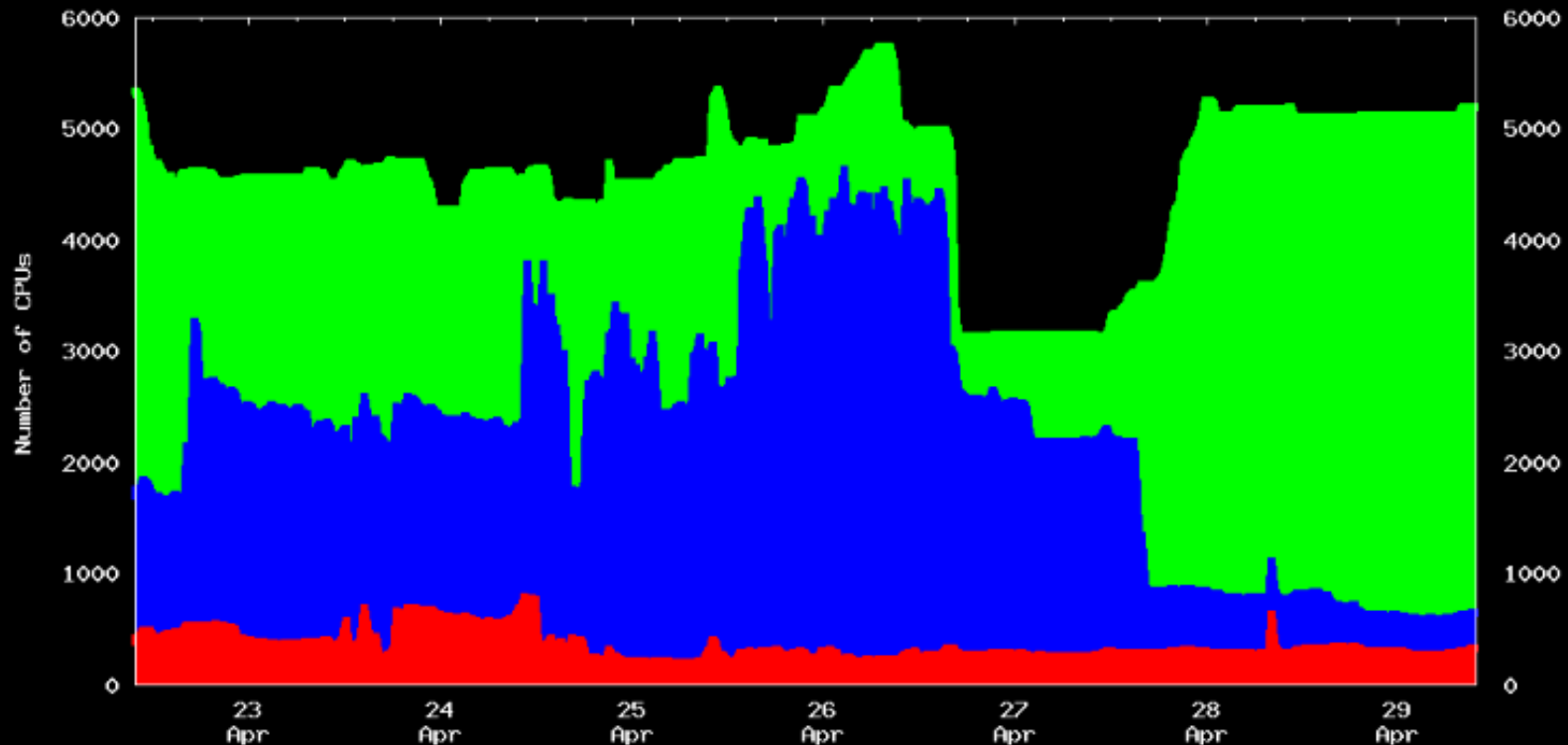


AGTCCGTACGATGCTATTAGCGAGCGTGA...

A screenshot of the Biometric Research Grid (BYGRID) website. The page displays a search interface for "3D Face Scans" with various filters and options. Below the search results, there is a table showing a grid of face images for validation. The table has columns for "Unvalidated", "Metadata Action", and four "Valid" columns (Valid 1, Valid 2, Valid 3, Valid 4). Each row contains a face image and a set of buttons for "Validate", "Problem", and "Unvalidate". The website interface includes a navigation menu on the left and a search bar at the top.

Good News:  
Computing is Plentiful

# CPU Utilization for the Last Week



404855 (51%) CPU-Hours Unused

328960 (41%) CPU-Hours Used by Condor

58935 (7%) CPU-Hours Used by Owner

792750 (100%) CPU-Hours Total



# Superclusters by the Hour

The screenshot shows a web browser window with the address bar containing the URL [http://arstechnica.com/business/news/2011/09/\\$1,279-per-hour, 30,000-core-clus...](http://arstechnica.com/business/news/2011/09/$1,279-per-hour, 30,000-core-clus...). The main content area displays a news article titled "\$1,279-per-hour, 30,000-core cluster built on Amazon EC2 cloud" by Jon Brodtkin, published a day ago. Below the article is a blue banner for CycleServer, Chef, and Ganglia. A green heading reads "Show: All converges over the last hour". Below this is a monitoring table with columns for Host Name, Instance, Cluster, Status, Total Converges, Last Completed Convergence, and Longest Convergence. To the right of the table is a bar chart titled "# Completed Converges" showing a vertical stack of green bars representing the number of completed converges over time, with a timestamp of 3:29 PM.

Host Name	Instance	Cluster	Status	Total Converges	Last Completed Convergence	Longest Convergence
ip-10-36-126-161.ec2.internal	i-b33abc2	412	Green	2	2011-07-30 15:27:42	3:50.787
ip-10-36-125-99.ec2.internal	i-591d9b38	412	Green	4	2011-07-30 15:36:43	3:31.503
ip-10-36-125-91.ec2.internal	i-e522a484	412	Red	4	2011-07-30 15:34:07	2:58.296
ip-10-36-125-137.ec2.internal	i-f088e9e	412	Green	3	2011-07-30 15:24:56	4:54.988
ip-10-35-9-95.ec2.internal	i-5d36b03c	412	Green	2	2011-07-30 15:31:47	4:28.892
ip-10-35-3-63.ec2.internal	i-d70d8bb6	412	Green	4	2011-07-30 15:28:33	4:10.597
ip-10-35-2-10.ec2.internal	i-e13c8a80	412	Green	2	2011-07-30 15:21:27	3:36.483
ip-10-35-14-209.ec2.internal	i-a51492c4	412	Green	3	2011-07-30 15:20:09	3:47.043
ip-10-35-10-207.ec2.internal	i-37399f56	412	Green	2	2011-07-30 15:27:13	4:23.522



The Bad News:  
It is inconvenient.

# End User Challenges

- System Properties:
  - Wildly varying resource availability.
  - Heterogeneous resources.
  - Unpredictable preemption.
  - Unexpected resource limits.
- User Considerations:
  - Jobs can't run for too long... but, they can't run too quickly, either!
  - I/O operations must be carefully matched to the capacity of clients, servers, and networks.
  - Users often do not even have access to the necessary information to make good choices!



I have a standard, debugged, trusted application that runs on my laptop.

A toy problem completes in one hour.

A real problem will take a month (I think.)

Can I get a single result faster?

Can I get more results in the same time?



Last year,  
I heard about  
this grid thing.

This year,  
I heard about  
this cloud thing.

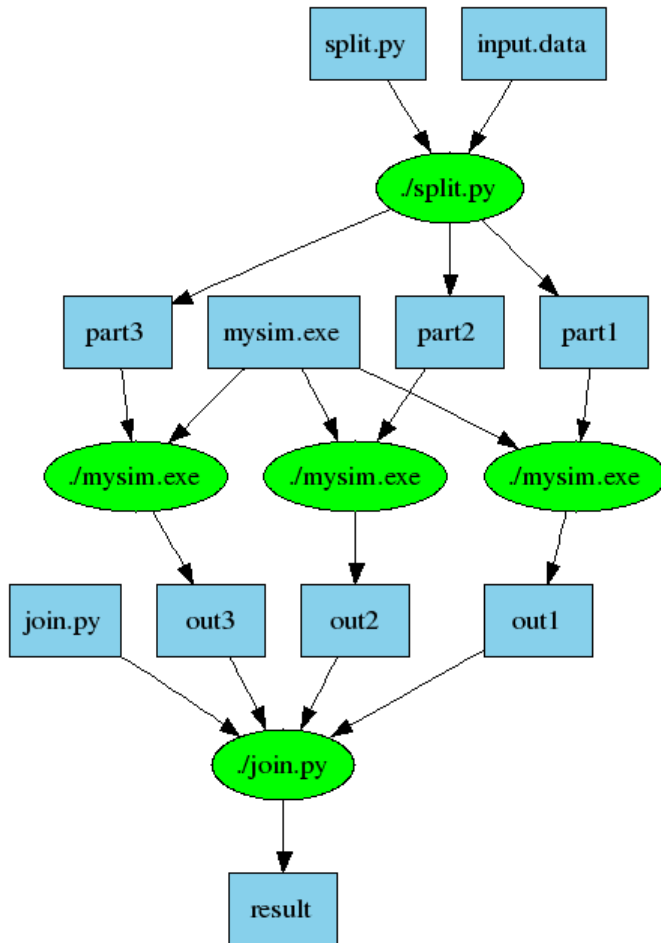


***What do I do next?***

# Our Philosophy:

- Harness all the resources that are available: desktops, clusters, clouds, and grids.
- Make it easy to scale up from one desktop to national scale infrastructure.
- Provide familiar interfaces that make it easy to connect existing apps together.
- Allow portability across operating systems, storage systems, middleware...
- Make simple things easy, and complex things possible.
- ***No special privileges required.***

# An Old Idea: Makefiles



part1 part2 part3: input.data split.py  
./split.py input.data

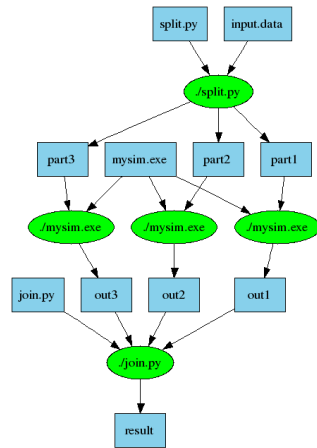
out1: part1 mysim.exe  
./mysim.exe part1 >out1

out2: part2 mysim.exe  
./mysim.exe part2 >out2

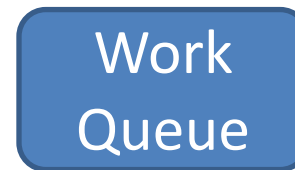
out3: part3 mysim.exe  
./mysim.exe part3 >out3

result: out1 out2 out3 join.py  
./join.py out1 out2 out3 > result

# Makeflow = Make + Workflow



- Provides portability across batch systems.
- Enable parallelism (but not too much!)
- Fault tolerance at multiple scales.
- Data and resource management.



<http://www.nd.edu/~ccl/software/makeflow>



# Makeflow Applications

The image illustrates a Makeflow application for eye biometrics. It consists of three main components:

- Workflow Diagram (Left):** A Makeflow graph showing the execution flow. It starts with an 'input3' node leading to a 'blast' node. The 'blast' node outputs to 'output.1' through 'output.5', 'error.1' through 'error.5', and 'total.1'. These outputs are collected by 'collect' nodes, which then feed into 'output', 'error', and 'total' nodes. Finally, these nodes lead to 'finalize' and then 'complete'.
- BioCompute Web Interface (Top Right):** A screenshot of the BioCompute portal. The 'My Data' section shows a folder named 'BXORID - BIOMETRICS RESEARCH ORID'. The 'Action' section shows 'Step 1 - Select Input File' and 'Step 2 - Title, Algorithm, and Privacy'. The 'My Queue' section shows a list of jobs, all with a status of 'Complete'.
- File Browser Window (Bottom Right):** A screenshot of a file browser showing BLAST results. The results include:
  - Query: 152 FLAGFYSKDILEMMLSLNLMNMIFFLFVSTGLMFIYIR--LLMYIMINDYN---LLII 207
  - Score = 121 bits (303), Expect = 2e-28
  - Identities = 76/139 (54%), Positives = 96/139 (69%), Gaps = 7/139 (5%)
  - Query: 208 YNLYD---ENYTMKSNFILLNMSVITGSMLSWFISFYFYIPLNMLKMLVYVSPGL 264
  - Score = 48.1 bits (113), Expect = 2e-06
  - Identities = 37/175 (21%), Positives = 92/175 (52%)

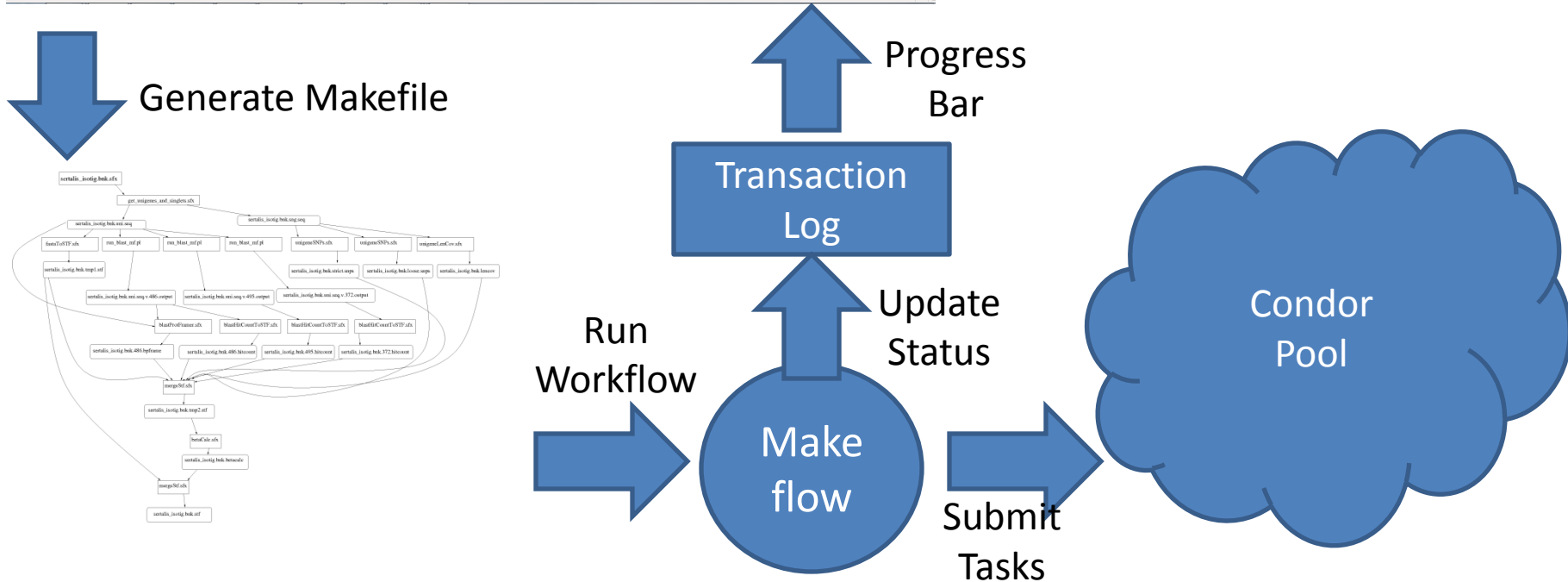
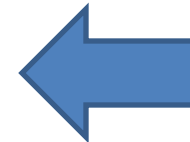
# Example: Biocompute Portal

The screenshot shows the Biocompute Portal interface for user 'athrash1'. It includes a navigation bar with 'Home', 'Data', 'Action', 'Queue', 'Admin', and 'More'. The main content is divided into three sections:

- My Data:** Shows 'View Others' Public Files' (athrash1) and 'Private Files' with a list of files and their sizes (e.g., 1\_assembled.unigenes.f., 1.ref, 1.TCA.clean\_1.fasta).
- Action:** Features a 'Submit a BLAST Job' button and a multi-step configuration process:
  - Step 1 - Select Input File: Includes 'Select Folder' (set to /athrash1) and 'Select File' (set to None).
  - Step 2 - Title, Algorithm, and Privacy: Includes 'Job Title' (untitled), 'Privacy' (Make this job public), and 'Algorithm' (BLASTp).
  - Step 3 - Choose BLAST Databases: (Not fully visible in the screenshot).
- My Queue:** A table showing the status of submitted jobs.
 

Title	Status	Username
test	Complete	athrash1
test	Complete	athrash1
test	Complete	athrash1
test4	Complete	athrash1
test3	Complete	athrash1
test2	Complete	athrash1
sorghum-test	Complete	athrash1
testing - input fl.	Complete	athrash1
debug test	Complete	athrash1
test	Complete	athrash1
test	Complete	athrash1
test - query(fle)	Complete	athrash1
test6	Complete	athrash1

BLAST  
SSAHA  
SHRIMP  
EST  
MAKER  
...

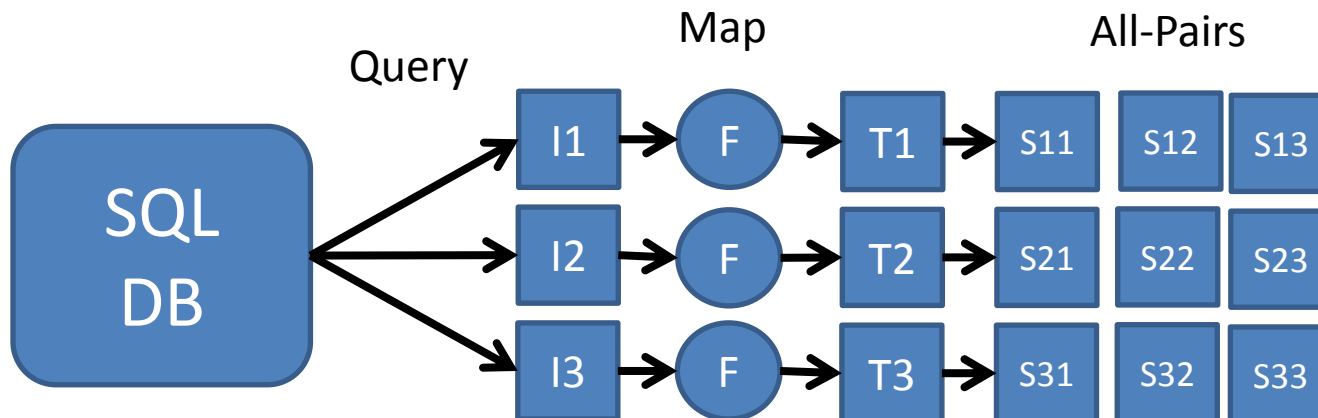


# Generating Workflows with Weaver

```
db      = SQLDataSet('db', 'biometrics', 'irises');  
irises = Query(db,color=='Blue')
```

```
iris_to_bit  = SimpleFunction('convert_iris_to_template')  
compare_bits = SimpleFunction('compare_iris_templates')
```

```
bits = Map(iris_to_bit, irises)  
AllPairs(compare_bits, bits, bits, output='scores.txt')
```

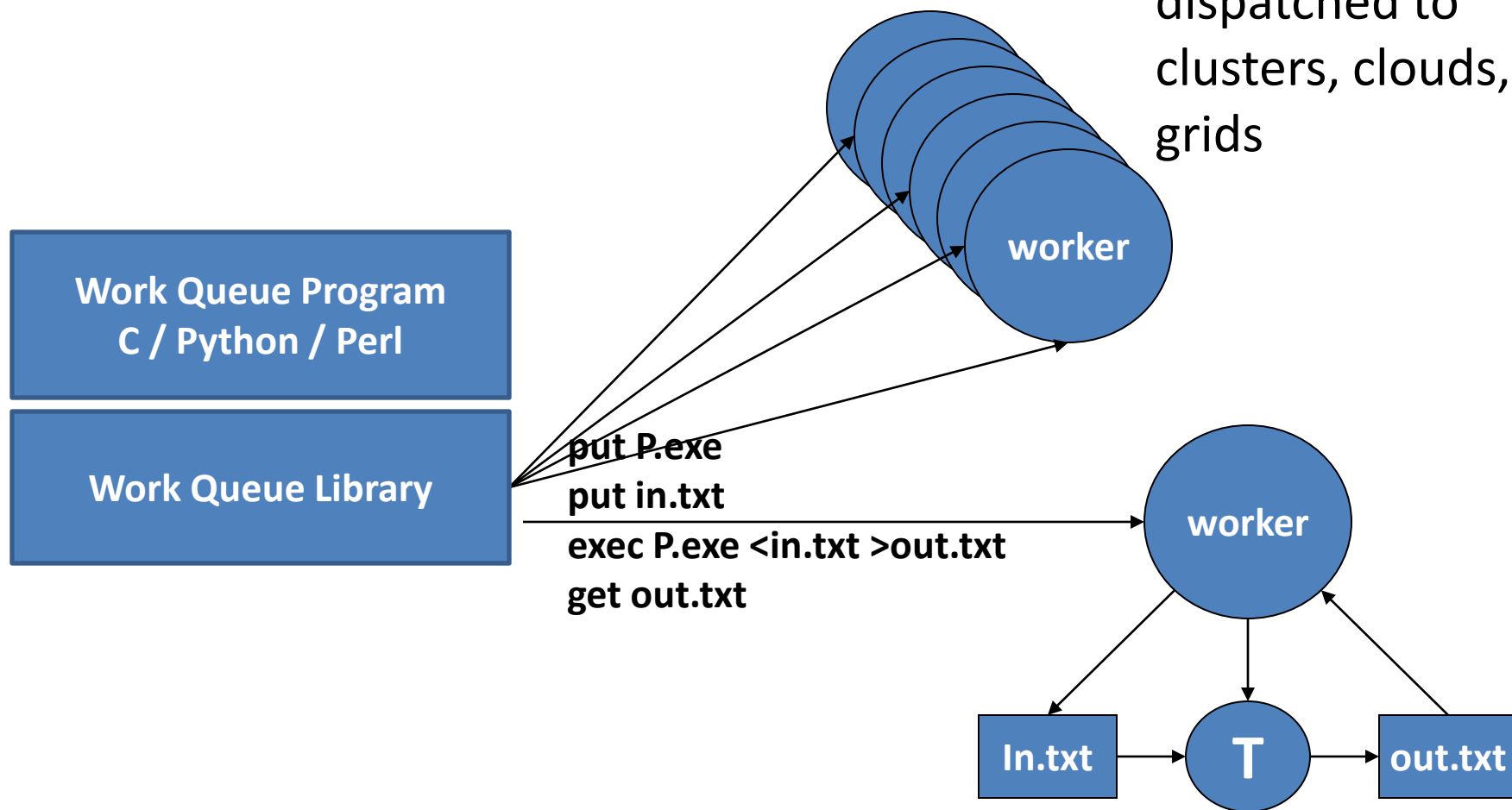


# Weaver + Makeflow + Batch System

- A good starting point:
  - Simple representation is easy to pick up.
  - Value provided by DAG analysis tools.
  - Easy to move apps between batch systems.
- But, the shared filesystem remains a problem.
  - Relaxed consistency confuses the coordinator.
  - Too easy for Makeflow to overload the FS.
- And the batch system was designed for large jobs.
  - Nobody likes seeing 1M entries in qstat.
  - **30-second rule** applies to most batch systems

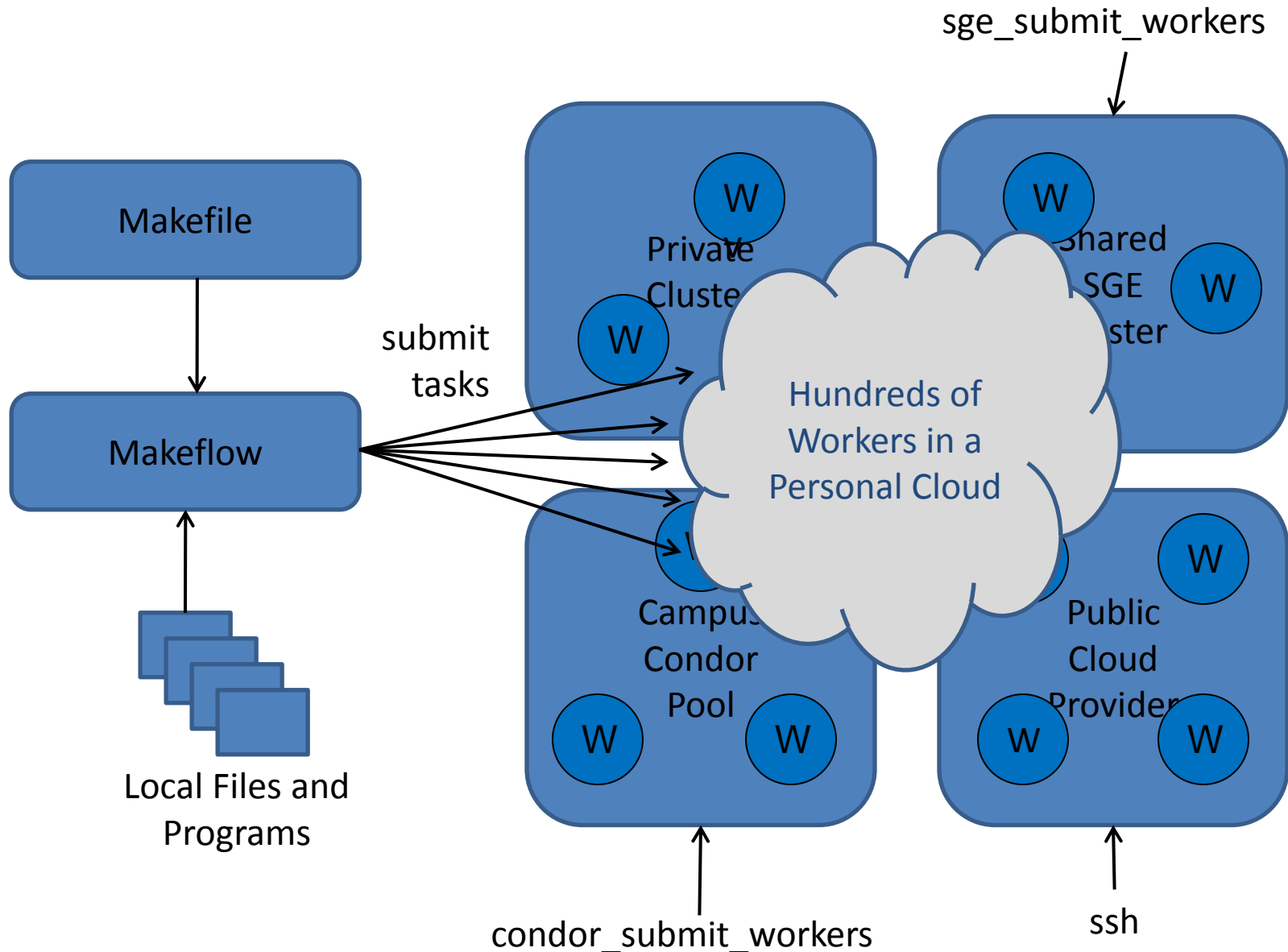
# Work Queue System

1000s of workers  
dispatched to  
clusters, clouds, and  
grids



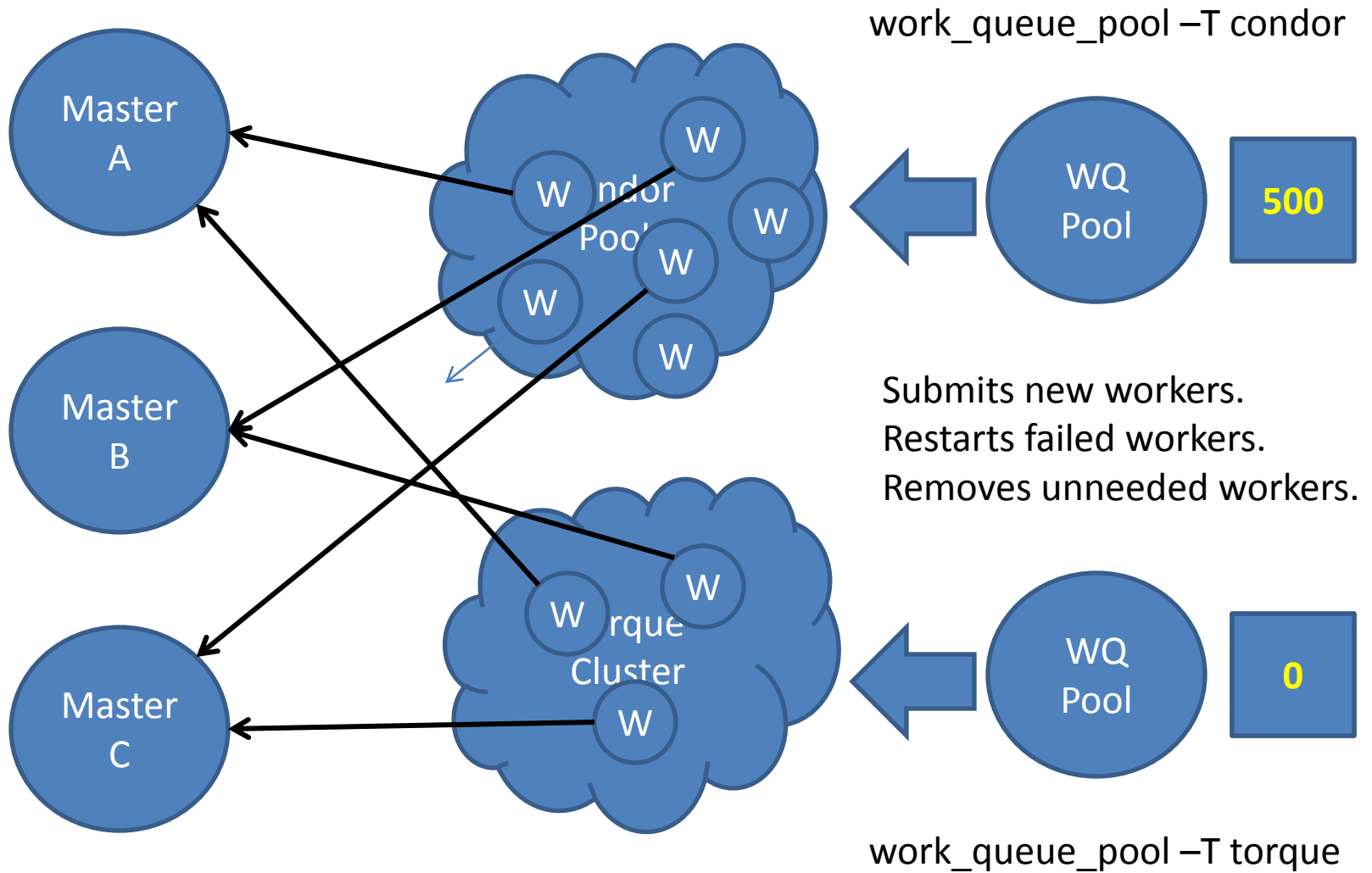
<http://www.nd.edu/~ccl/software/workqueue>

# Makeflow + Work Queue

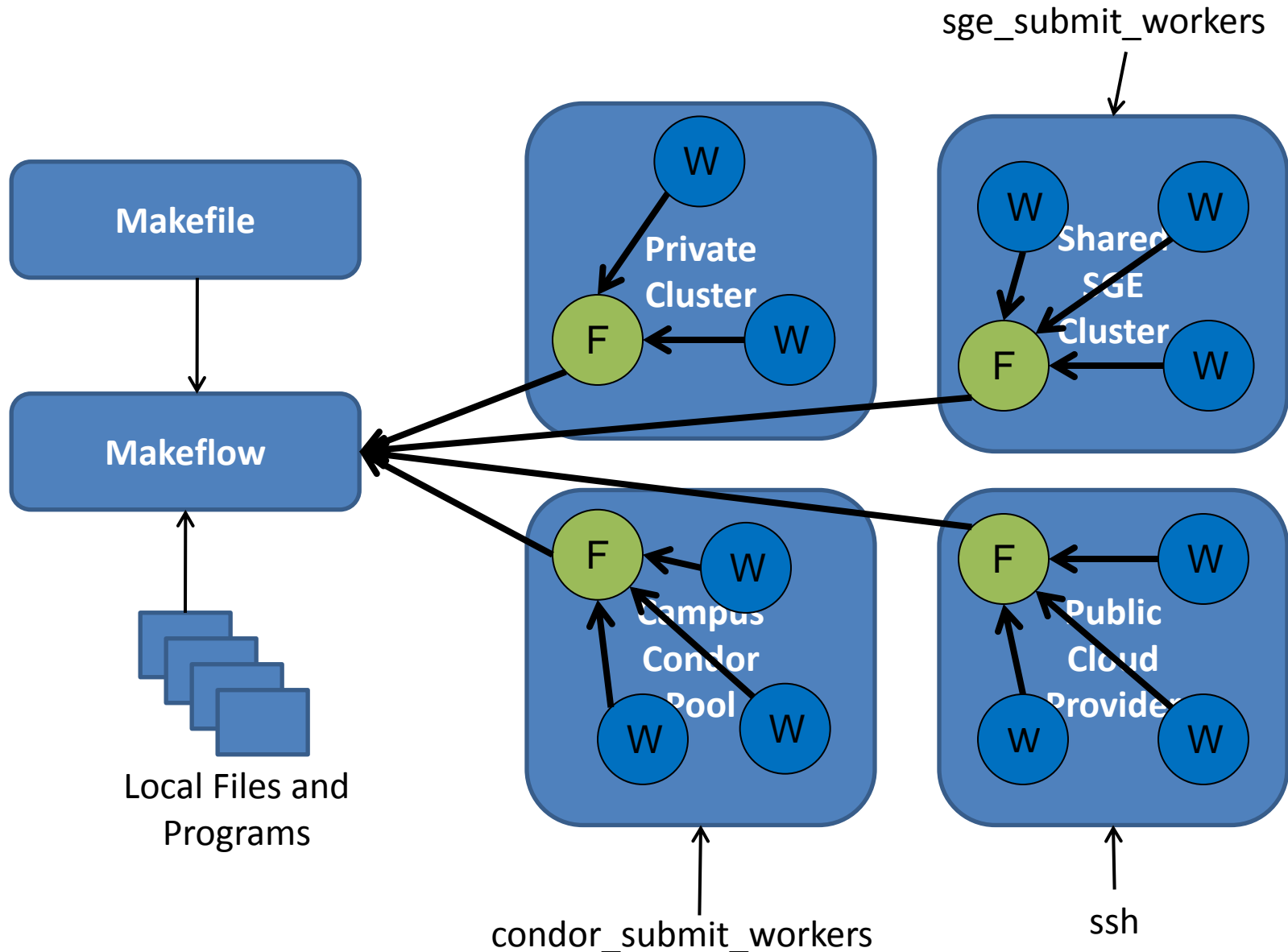




# Managing Your Workforce



# Hierarchical Work Queue



# Work Queue Library

```
#include "work_queue.h"

while( not done ) {

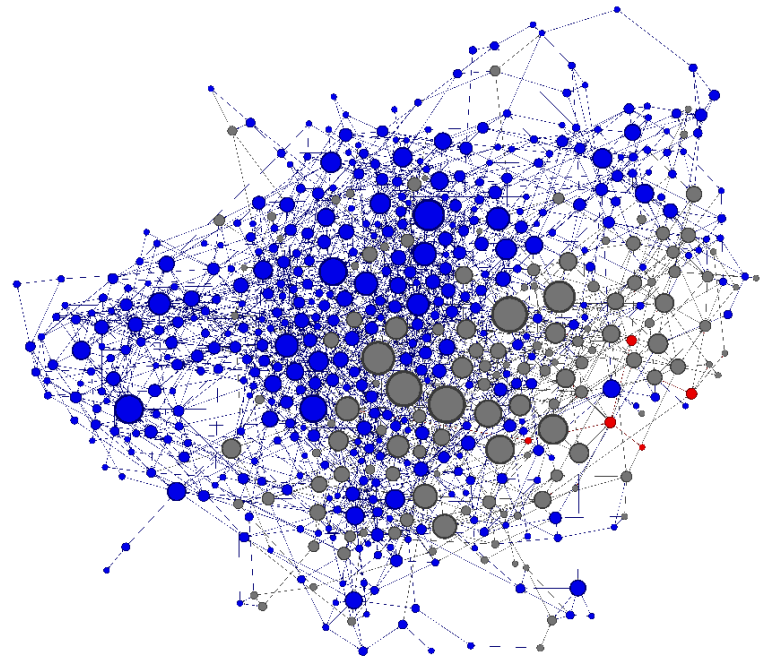
    while (more work ready) {
        task = work_queue_task_create();
        // add some details to the task
        work_queue_submit(queue, task);
    }

    task = work_queue_wait(queue);
    // process the completed task
}
```

<http://www.nd.edu/~ccl/software/workqueue>

# Adaptive Weighted Ensemble

Proteins fold into a number of distinctive states, each of which affects its function in the organism.

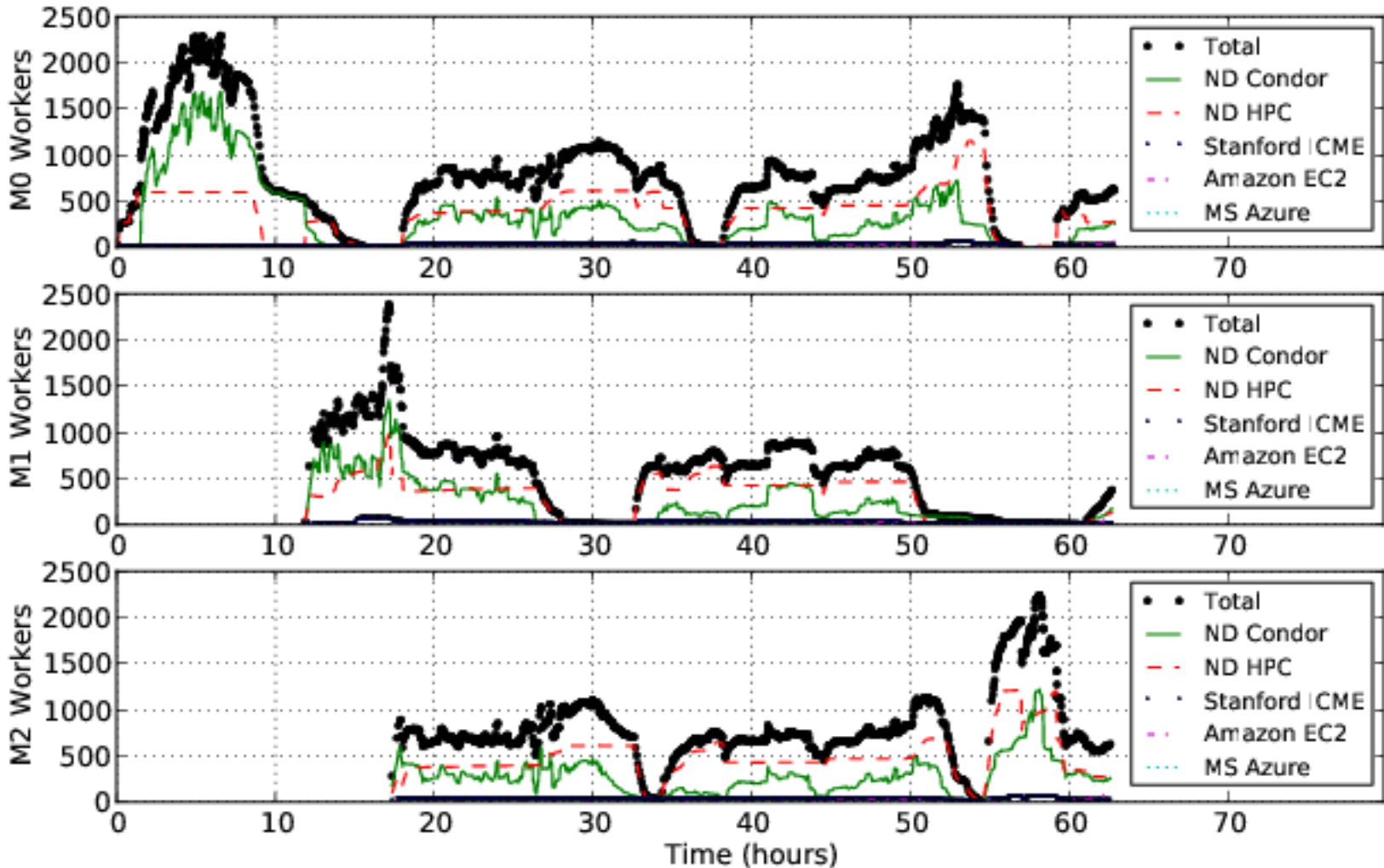


How common is each state?  
How does the protein transition between states?  
How common are those transitions?

# AWE Using Work Queue

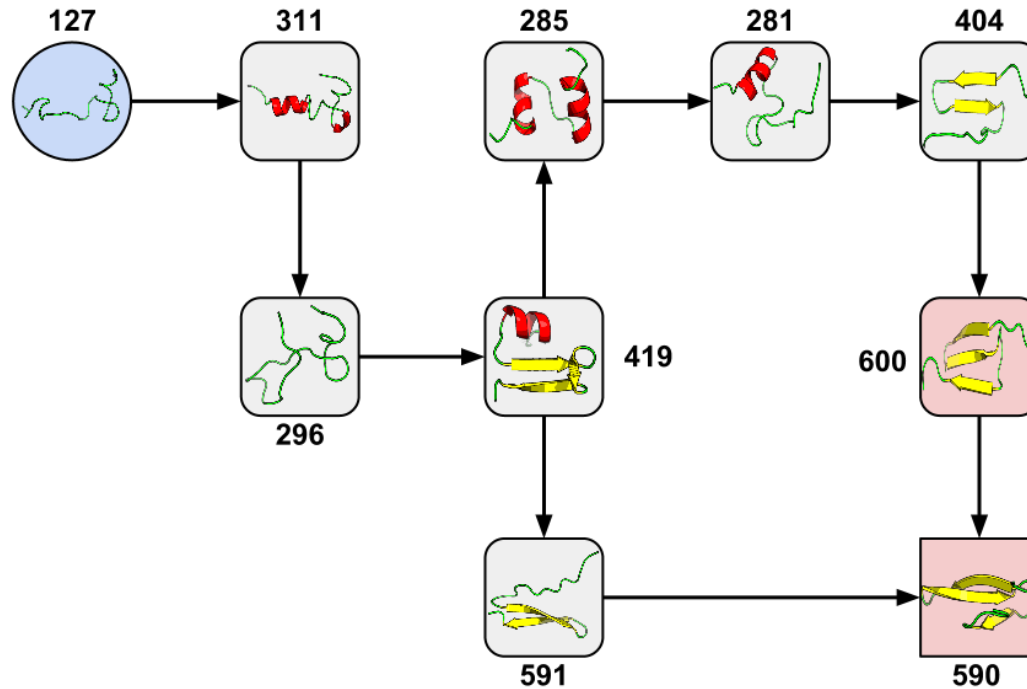
- **Simplified Algorithm:**
  - **Submit** N short simulations in various states.
  - **Wait** for them to finish.
  - When done, record all state transitions.
  - If too many are in one state, redistribute them.
  - Stop if enough data has been collected.
  - Continue back at step 2.

# AWE on Clusters, Clouds, and Grids





# New Pathway Found!



Credit: Joint work in progress with Badi Abdul-Wahid, Dinesh Rajan, Haoyun Feng, Jesus Izaguirre, and Eric Darve.

# Software as a Social Lever

- User and app accustomed to a particular system with standalone executables.
- Introduce Makeflow as an aid for expression, debugging, performance monitoring.
- When ready, use Makeflow + Work Queue to gain more direct control of I/O operations on the existing cluster.
- When ready, deploy Work Queue to multiple systems across the wide area.
- When ready, write new apps to target the Work Queue API directly.

# Overview

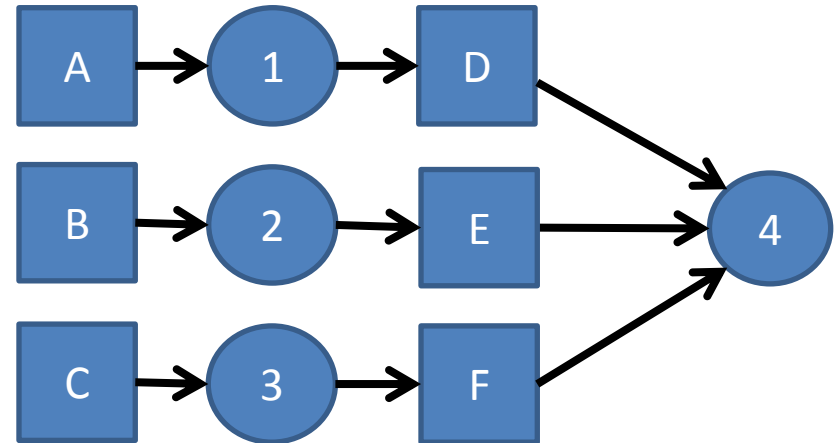
- Experience with Concurrent Applications
  - Makeflow, Weaver, Work Queue
- **Thesis: Convergence of Models**
  - **Declarative Language**
  - **Directed Graphs of Tasks and Data**
  - **Shared Nothing Architecture**
- Open Problems
  - Transaction Granularity
  - Where to Parallelize?
  - Resource Management
- Concluding Thoughts

# Scalable Computing Model

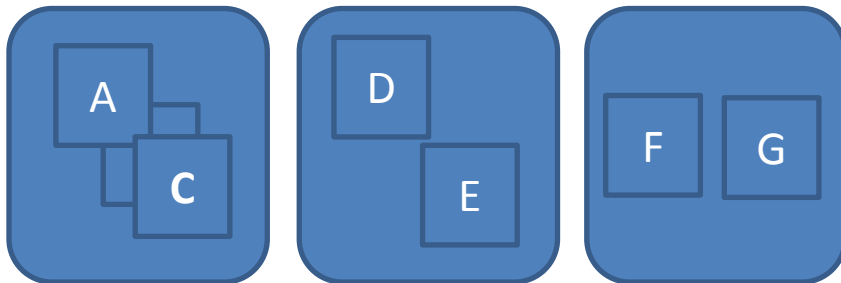
**Weaver**

for x in list f(g(x))

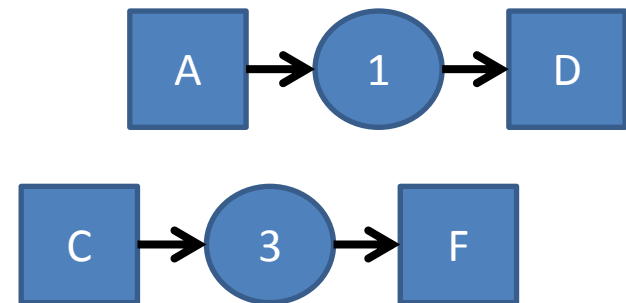
**Makeflow**



**Shared-Nothing Cluster**



**Work Queue**

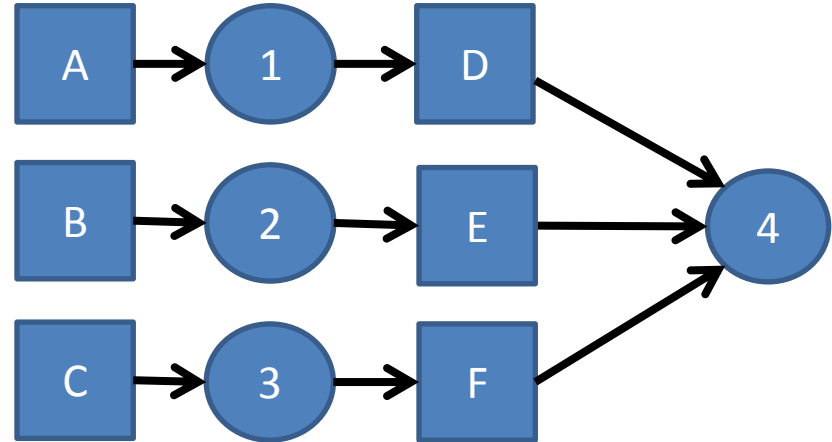


# Scalable Computing Model

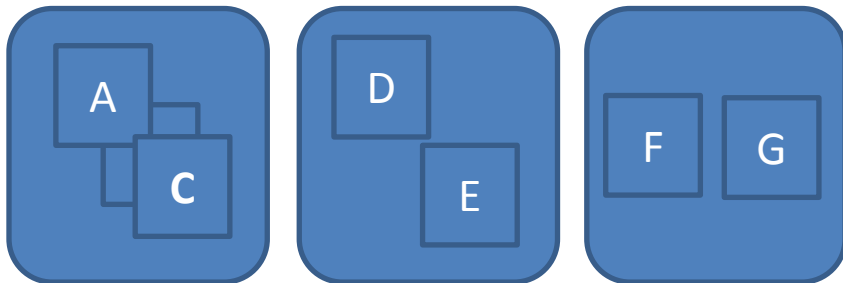
## Declarative Language

for x in list f(g(x))

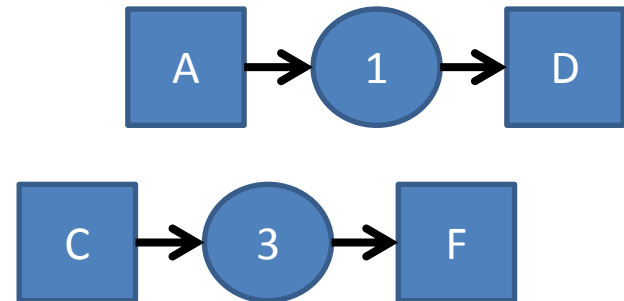
## Dependency Graph



## Shared-Nothing Cluster



## Independent Tasks



# Convergence of Worlds

- Scientific Computing
  - Weaver, Makeflow, Work Queue, Cluster
  - Pegasus, DAGMan, Condor, Cluster
  - Swift-K, (?), Karajan, Cluster
- High Performance Computing
  - SMPSS->JDF->DAGue->NUMA Architecture
  - Swift-T, (?), Turbine, MPI Application
- Databases and Clouds
  - Pig, Map-Reduce, Hadoop, HDFS
  - JSON, Map-Reduce, MongoDB, Storage Cluster
  - LINQ, Dryad, Map-Reduce, Storage Cluster



# Thoughts on the Layers

- Declarative languages.
  - Pros: Compact, expressive, easy to use.
  - Cons: Intractable to analyze in the general case.
- Directed graphs.
  - Pros: Finite structures with discrete components are easily analyzed.
  - Cons: Cannot represent dynamic applications.
- Independent tasks and data.
  - Pros: Simple submit/wait APIs, data dependencies can be exploited by layers above below.
  - Cons: In most general case, scheduling is intractable.
- Shared-nothing clusters.
  - Pros: Can support many disparate systems. Performance is readily apparent.
  - Cons: requires knowledge of dependencies.

# Common Model of Compilers

- Scanner detects single tokens.
  - Finite state machine is fast and compact.
- Parser detects syntactic elements.
  - Grammar + push down automata. LL(k), LR(k)
- Abstract syntax tree for semantic analysis.
  - Type analysis and high level optimization.
- Intermediate Representation
  - Register allocation and low level optimization.
- Assembly Language
  - Generated by tree-matching algorithm.

# Overview

- Experience with Concurrent Applications
  - Makeflow, Weaver, Work Queue
- Thesis: Convergence of Models
  - Declarative Language
  - Directed Graphs of Tasks and Data
  - Shared Nothing Architecture
- **Open Problems**
  - **Transaction Granularity**
  - **Where to Parallelize?**
  - **Resource Management**
- Concluding Thoughts

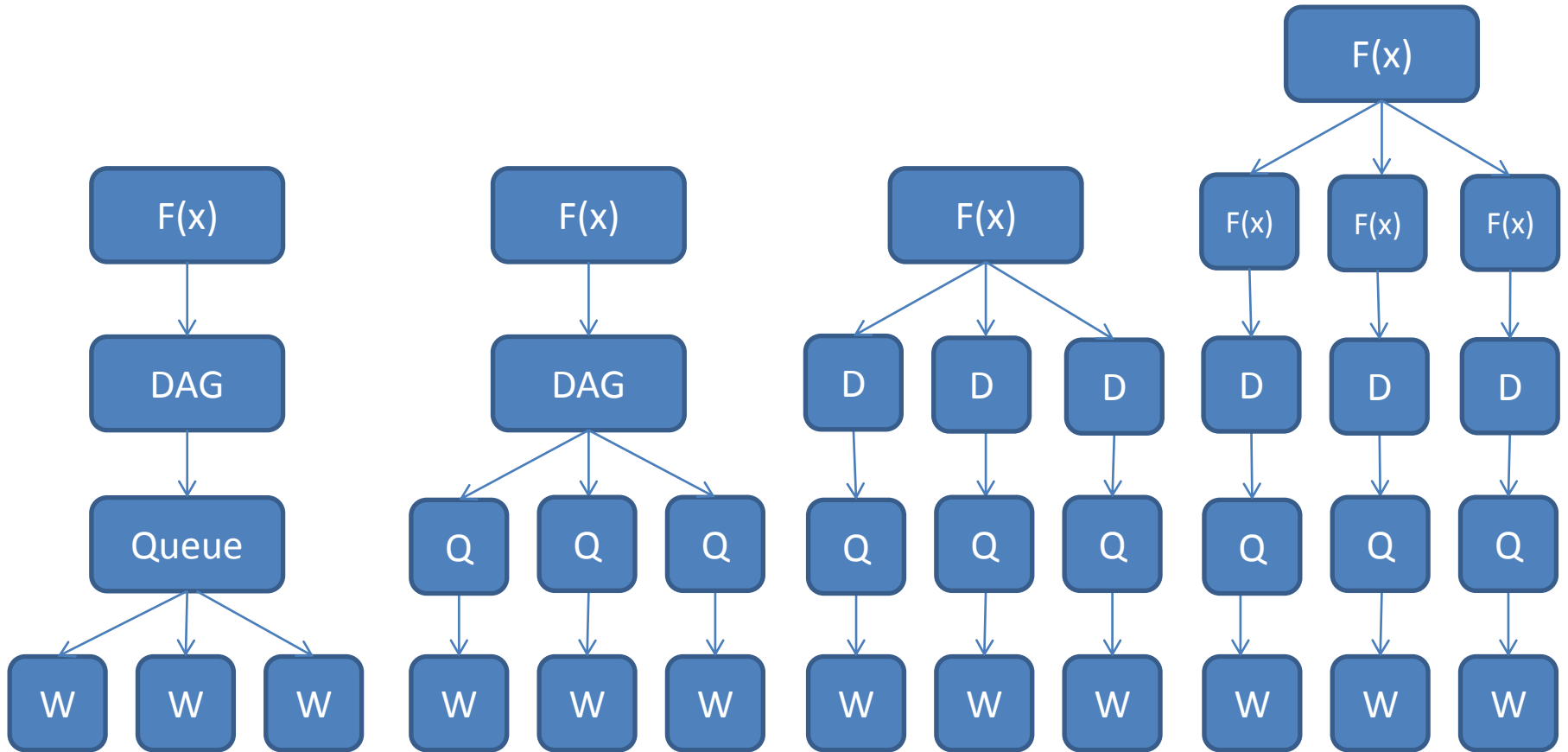
Observation:

Generating parallelism is easy but  
making it *predictable* is hard!

# Challenge: Transaction Granularity

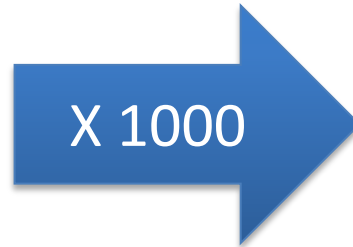
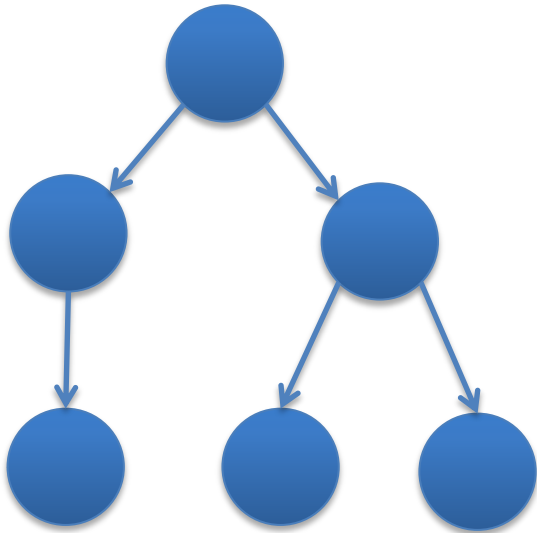
- Commit every action to disk. (Condor)
  - + Makes recovery from any point possible.
  - Significant overhead on small tasks.
- Commit only completed tasks to disk. (Falcon)
  - Cannot recover tasks in progress after a failure.
  - + Fast for very small tasks.
- Extreme: Commit only completed DAG.
- Problem: Choice changes with workload!

# Challenge: Where to Parallelize?



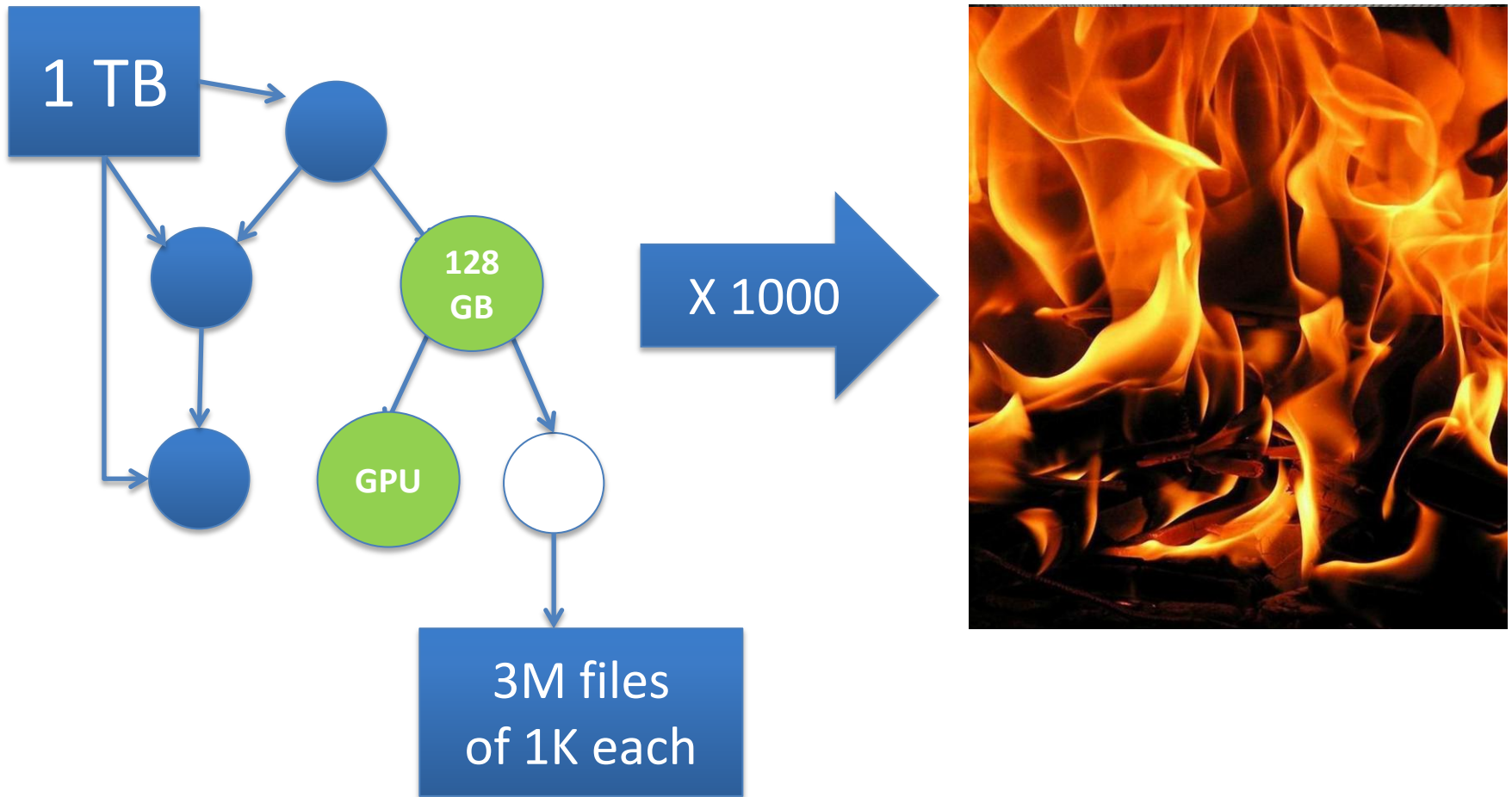
# Challenge: Resource Management

# The Ideal Picture





# What actually happens:



# Some reasonable questions:

- Will this workload **at all** on machine X?
- How many workloads can I run simultaneously without running out of storage space?
- Did this workload actually behave as expected when run on a new machine?
- How is run X different from run Y?
- If my workload wasn't able to run on this machine, where can I run it?

End users have **no idea** what resources  
their applications actually need.

and...

Computer systems are **terrible** at  
describing their capabilities and limits.

and...

They don't know when to say **NO**.

# **dV/dt : Accelerating the Rate of Progress Towards Extreme Scale Collaborative Science**

**Miron Livny (UW), Ewa Deelman (USC/ISI), Douglas Thain (ND),  
Frank Wuerthwein (UCSD), Bill Allcock (ANL)**

*... make it easier for scientists to conduct large-scale computational tasks that use the power of computing resources they do not own to process data they did not collect with applications they did not develop ...*

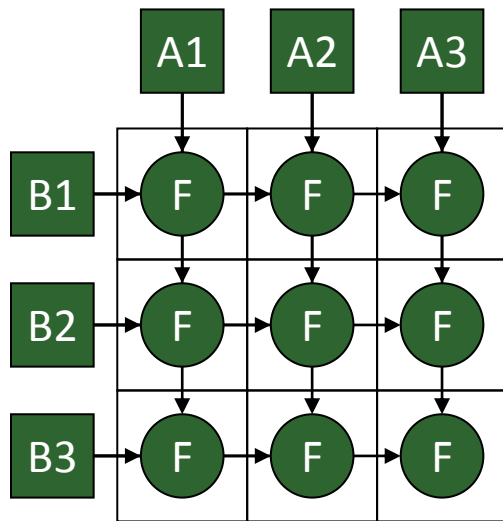
# Categories of Applications

## Concurrent Workloads

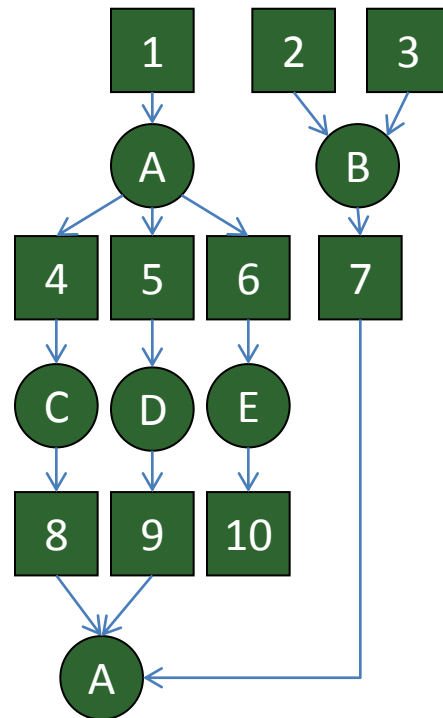
### Static Workloads

### Dynamic Workloads

#### Regular Graphs



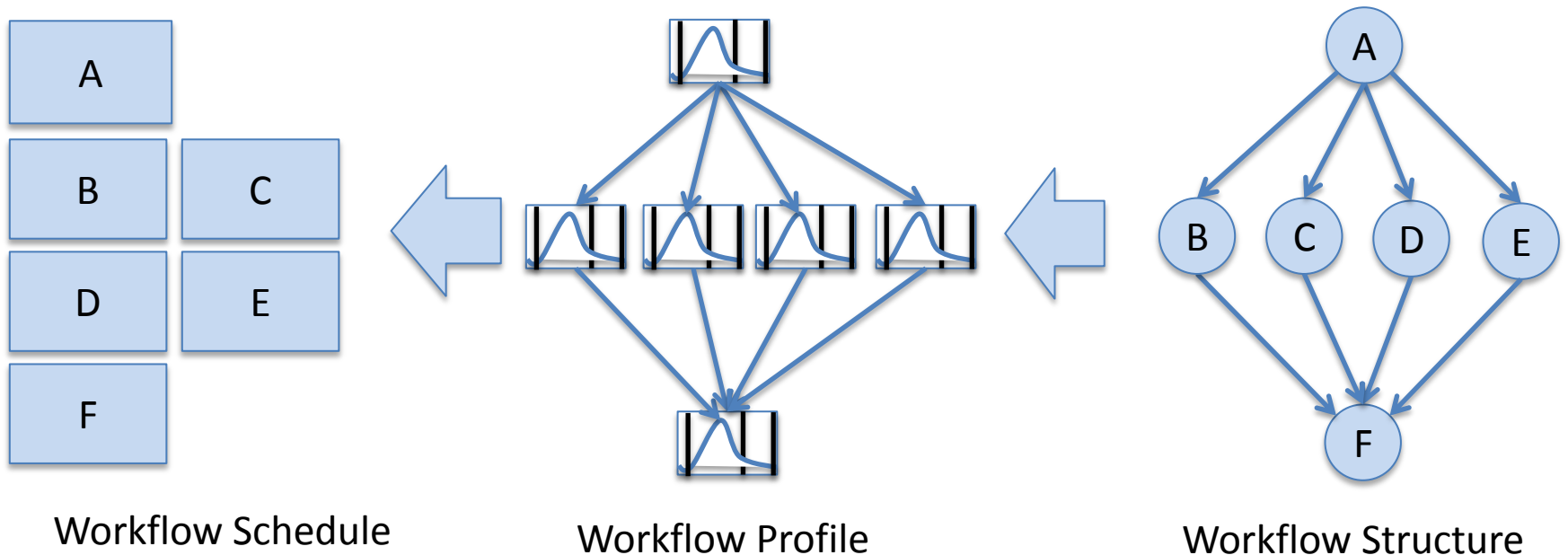
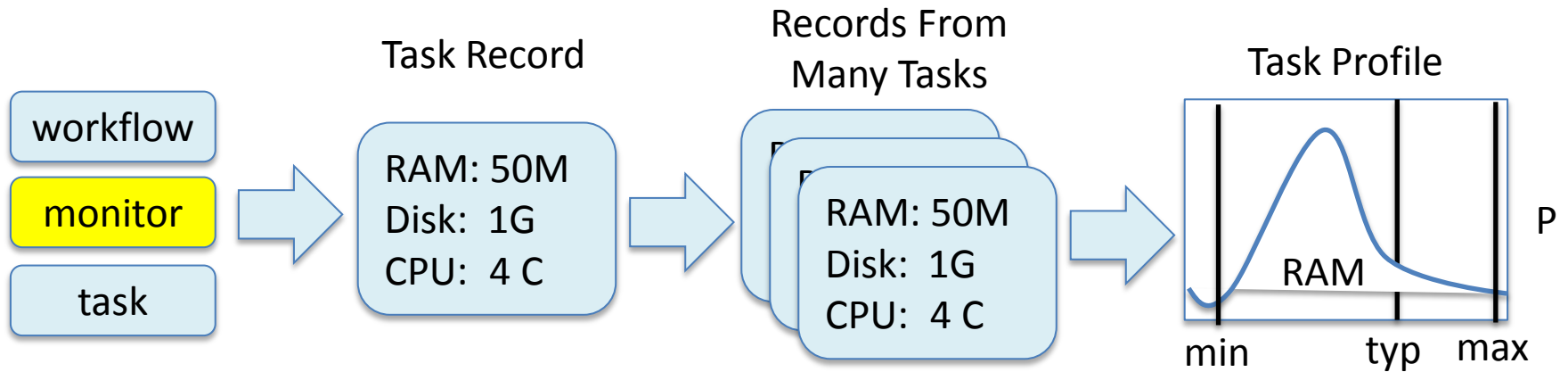
#### Irregular Graphs



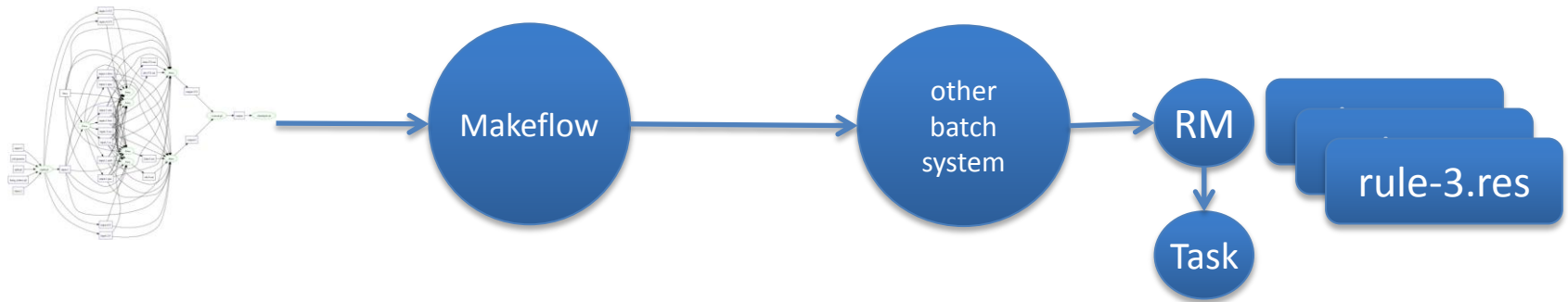
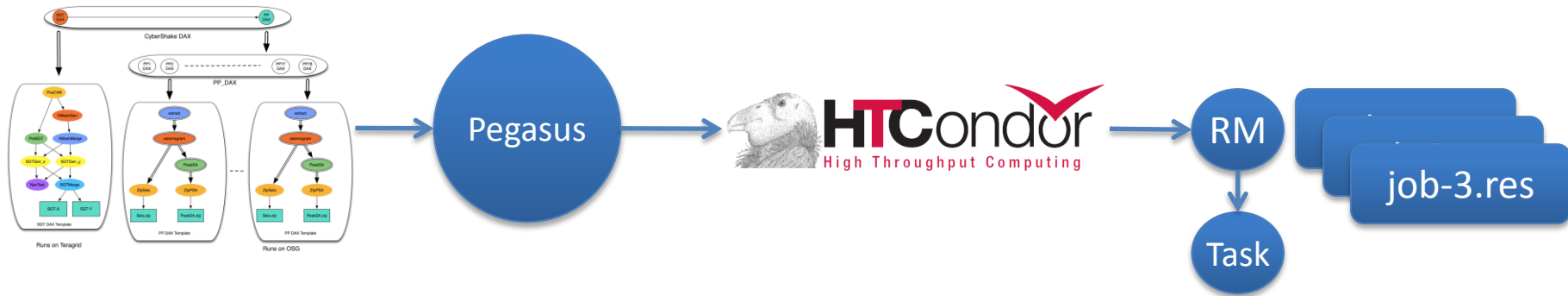
```
while( more work to do)
{
    foreach work unit {
        t = create_task();
        submit_task(t);
    }

    t = wait_for_task();
    process_result(t);
}
```

# Data Collection and Modeling



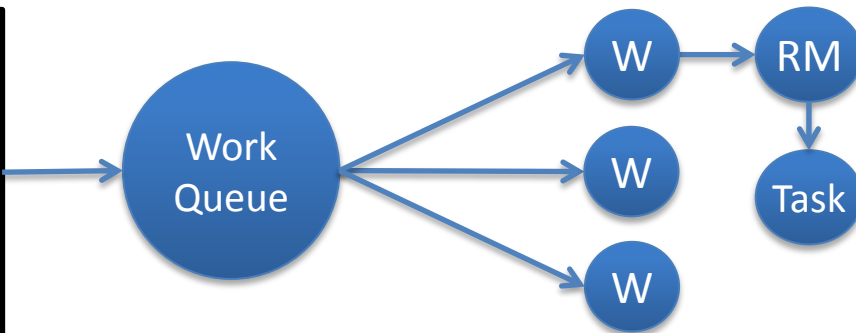
# Portable Resource Management



```

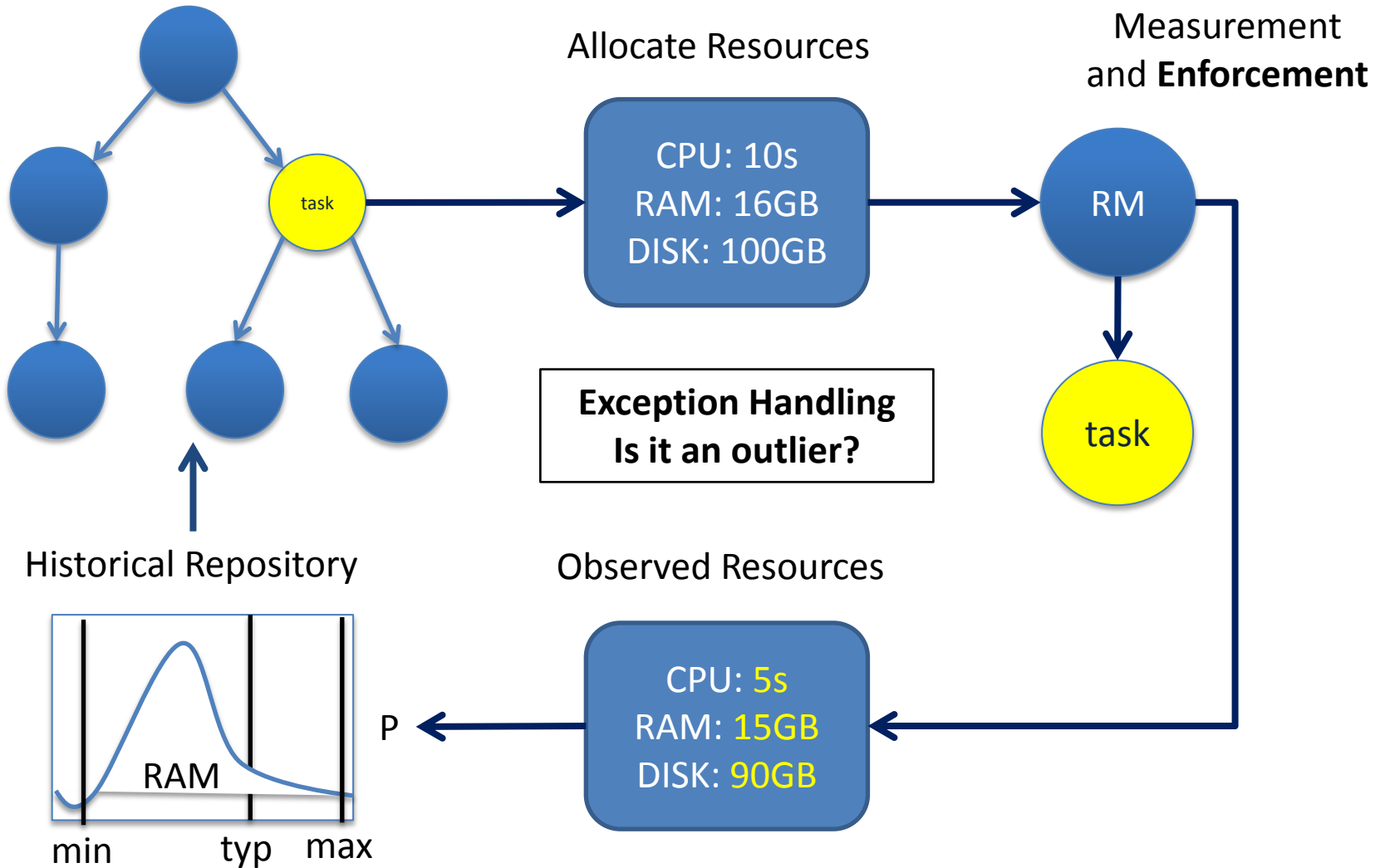
while( more work to do ) {
  foreach work unit {
    t = create_task();
    submit_task(t);
  }

  t = wait_for_task();
  process_result(t);
}
    
```



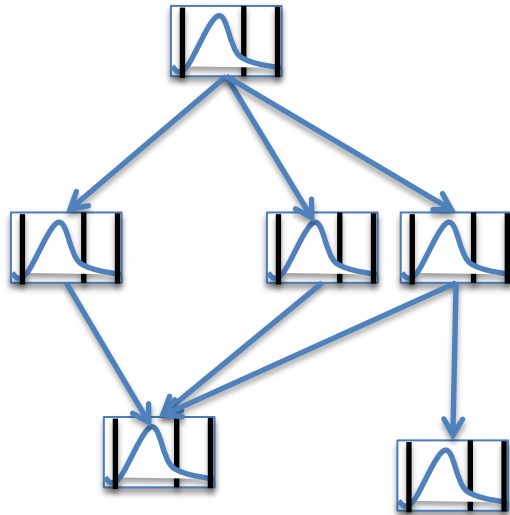
task 1 details:  
cpu, ram, disk  
task 2 details:  
cpu, ram, disk  
task 3 details:  
cpu, ram, disk

# Completing the Cycle





# Complete Workload Characterization



128 GB  
32 cores

X 1

128 GB  
32 cores

X 1

16 GB  
4 cores

X 10

16 GB  
4 cores

X 100

12 hours  
500 Gb/s I/O

1 hour  
5 Tb/s I/O

X 1000

We can approach the question:  
Can it run on this particular machine?  
What machines could it run on?

At what levels of the model can resource management be done?

# Overview

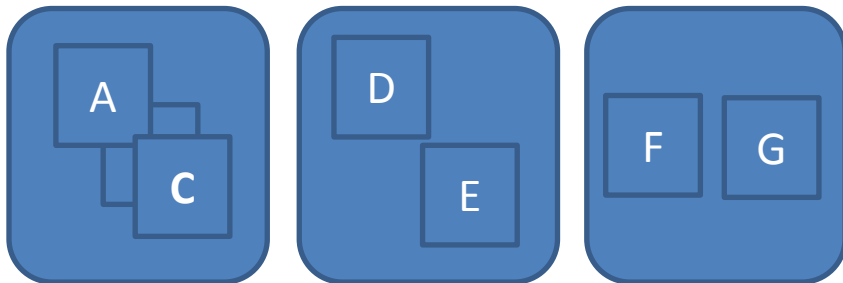
- Experience with Concurrent Applications
  - Makeflow, Weaver, Work Queue
- Thesis: Convergence of Models
  - Declarative Language
  - Directed Graphs of Tasks and Data
  - Shared Nothing Architecture
- Open Problems
  - Transaction Granularity
  - Where to Parallelize?
  - Resource Management
- **Concluding Thoughts**

# Scalable Computing Model

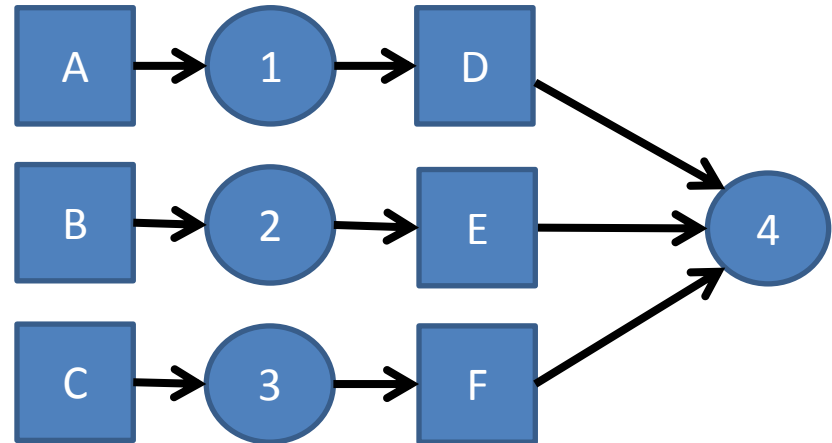
## Weaver

for x in list f(g(x))

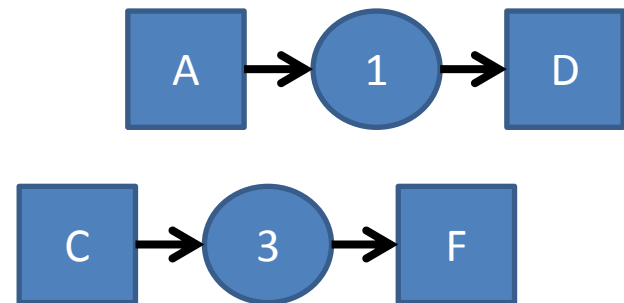
## Work Queue Workers



## Makeflow



## Work Queue Master



An exciting time to work  
in distributed systems!

# Talks by CCL Students This Weekend

- Casey Robinson,  
**Automated Packaging of Bioinformatics Workflows for Portability and Durability Using Makeflow,**  
WORKS Workshop, 4pm on Sunday.
- Patrick Donnelly,  
**Design of an Active Storage Cluster File System for DAG Workflows,**  
*DISCS Workshop on Monday.*

# Acknowledgements

## dV/dT Project PIs

- Bill Allcock (ALCF)
- Ewa Deelman (USC)
- Miron Livny (UW)
- Frank Weurthwein (UCSD)



## CCL Staff

- Ben Tovar

## CCL Graduate Students:

- Michael Albrecht
- Patrick Donnelly
- Dinesh Rajan
- Casey Robinson
- Peter Sempelinski
- Nick Hazekamp
- Haiyan Meng
- Peter Ivie

# The Cooperative Computing Lab

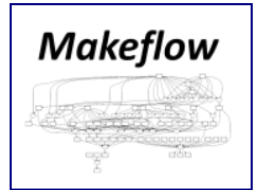
*University of Notre Dame*



<http://www.nd.edu/~ccl>

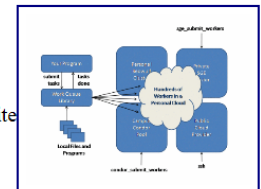
## Makeflow

Makeflow is a workflow system for parallel and distributed computing that uses a language very similar to Make. Using Makeflow, you can write simple scripts that easily execute on hundreds or thousands of machines.



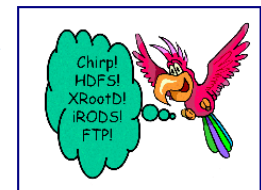
## Work Queue

Work Queue is a system and library for creating and managing scalable master-worker style programs that scale up to thousands machines on clusters, clouds, and grids. Work Queue programs are easy to write in C, Python or Perl.



## Parrot

Parrot is a transparent user-level virtual filesystem that allows any ordinary program to be attached to many different remote storage systems, including HDFS, iRODS, Chirp, and FTP.



## Chirp

Chirp is a personal user-level distributed filesystem that allows unprivileged users to share space securely, efficiently, and conveniently. When combined with Parrot, Chirp allows users to create custom wide-area distributed filesystems.

