



UNIVERSITY OF  
NOTRE DAME

# TaskVine: Workflow for Data Intensive and Serverless Applications

Douglas Thain and the CCL Team  
University of Notre Dame  
Throughput Computing 2023  
Madison, WI July 2023



# Throughput Computing 2023

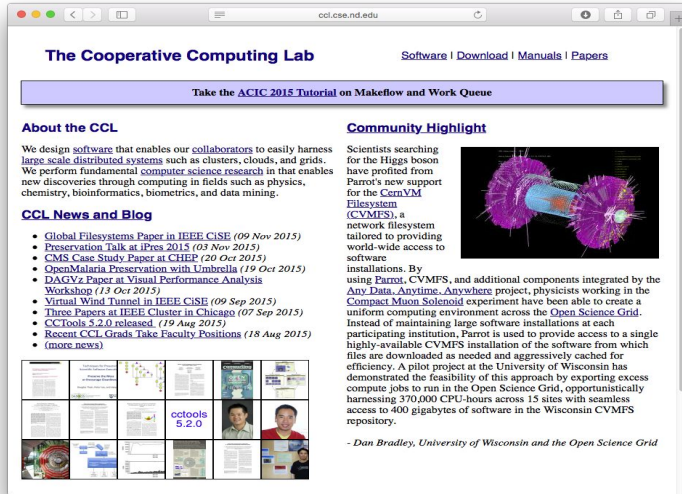
OSG All-Hands Meeting



HTCondor Week



# The Cooperative Computing Lab



The screenshot shows the homepage of the Cooperative Computing Lab (CCL) website. The browser address bar displays [ccl.cse.nd.edu](http://ccl.cse.nd.edu). The page features a navigation menu with links for 'Software', 'Download', 'Manuals', and 'Papers'. A prominent banner reads 'Take the ACIC 2015 Tutorial on Makeflow and Work Queue'. Below this, there are sections for 'About the CCL', 'CCL News and Blog', and 'Community Highlight'. The 'About the CCL' section describes the lab's focus on large-scale distributed systems. The 'CCL News and Blog' section lists recent publications and events. The 'Community Highlight' section features a 3D visualization of a particle detector and text describing the work of scientists at the University of Wisconsin and the Open Science Grid.

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 *develop open source software* for large scale distributed computing.

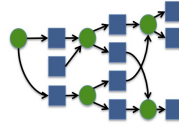
<http://ccl.cse.nd.edu>

# How do I organize my work to use HTCondor?

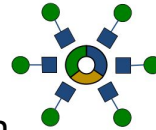
<https://condor.cse.nd.edu>



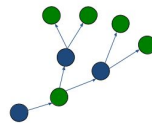
**Makeflow**  
Unix-Oriented DAGs



**Work Queue**  
Dynamic Task Creation



**TaskVine**  
Dynamic Data Sharing

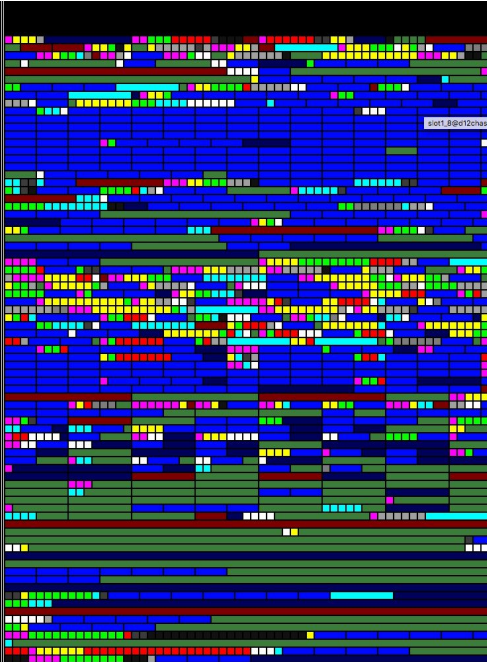


**Notre Dame Condor Status**

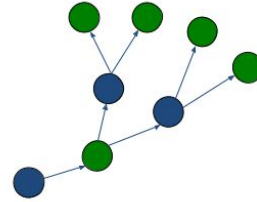
	Slots	Cores
woodard@nd.edu	976	3904
hhatami@nd.edu	411	411
cbeaufl@nd.edu	370	370
acummin1@nd.edu	311	311
jkinniso@nd.edu	193	291
jdiazort@nd.edu	275	275
roidtman@nd.edu	261	261
kbarlock@nd.edu	217	217
ophelan1@nd.edu	98	98
kherring@nd.edu	58	58
smustiph@nd.edu	42	42
rmwolf3@nd.edu	1	4
nthomann@nd.edu	3	3
gayle@nd.edu	1	1
pdonnel4@nd.edu	1	1
tperkin1@nd.edu	1	1
btovar@nd.edu	1	1
nbiancha@nd.edu	1	1
Unclaimed	241	2302
Matched	138	1333
Preempting		
Owner	57	521
Total	3657	10406

**Display Options**

Sort: users machines  
 Show: users states  
 Size: bigger smaller  
 Scale: none cores



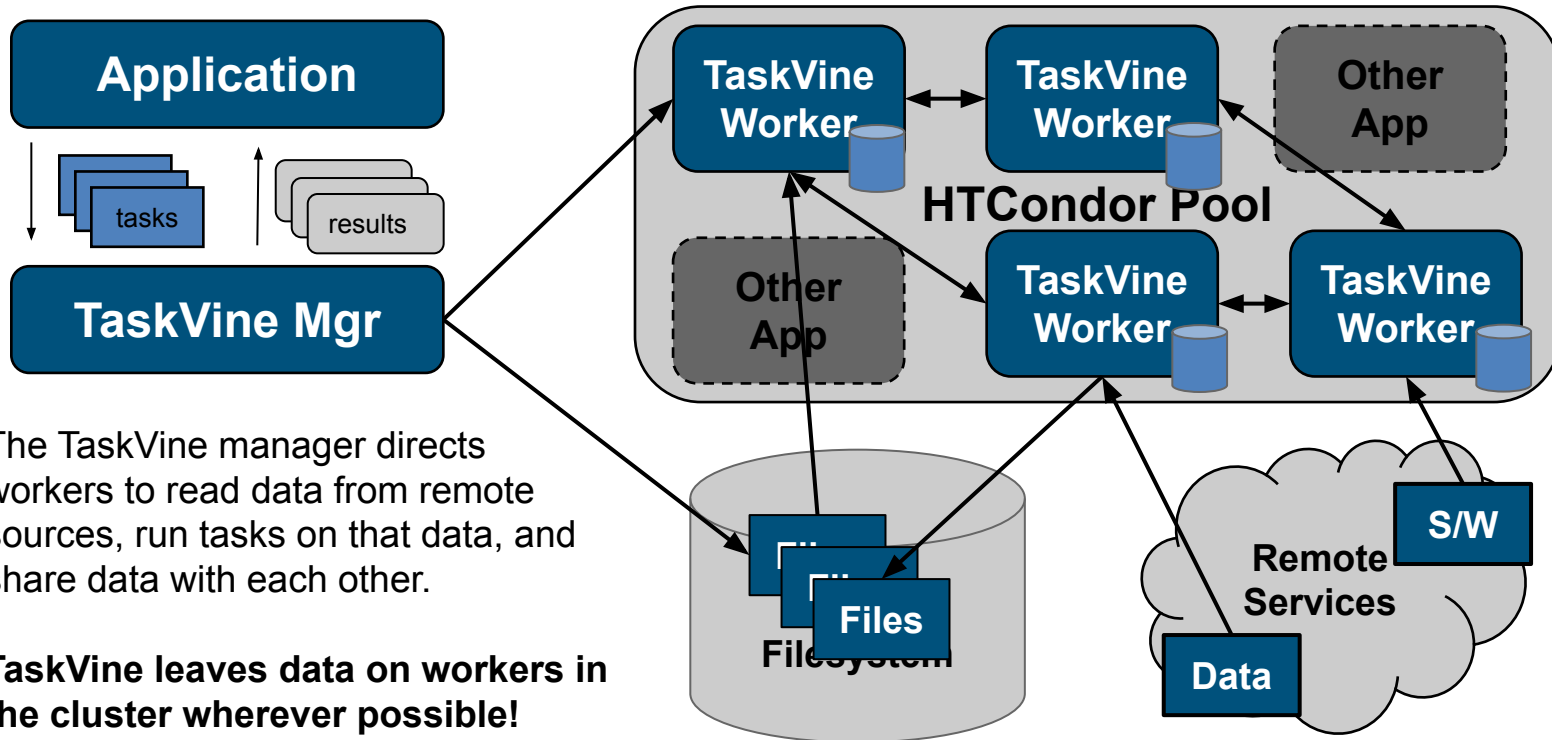
# *TaskVine*



TaskVine is a system for executing **data intensive** scientific workflows on clusters, clouds, and grids from very small to massive scale.

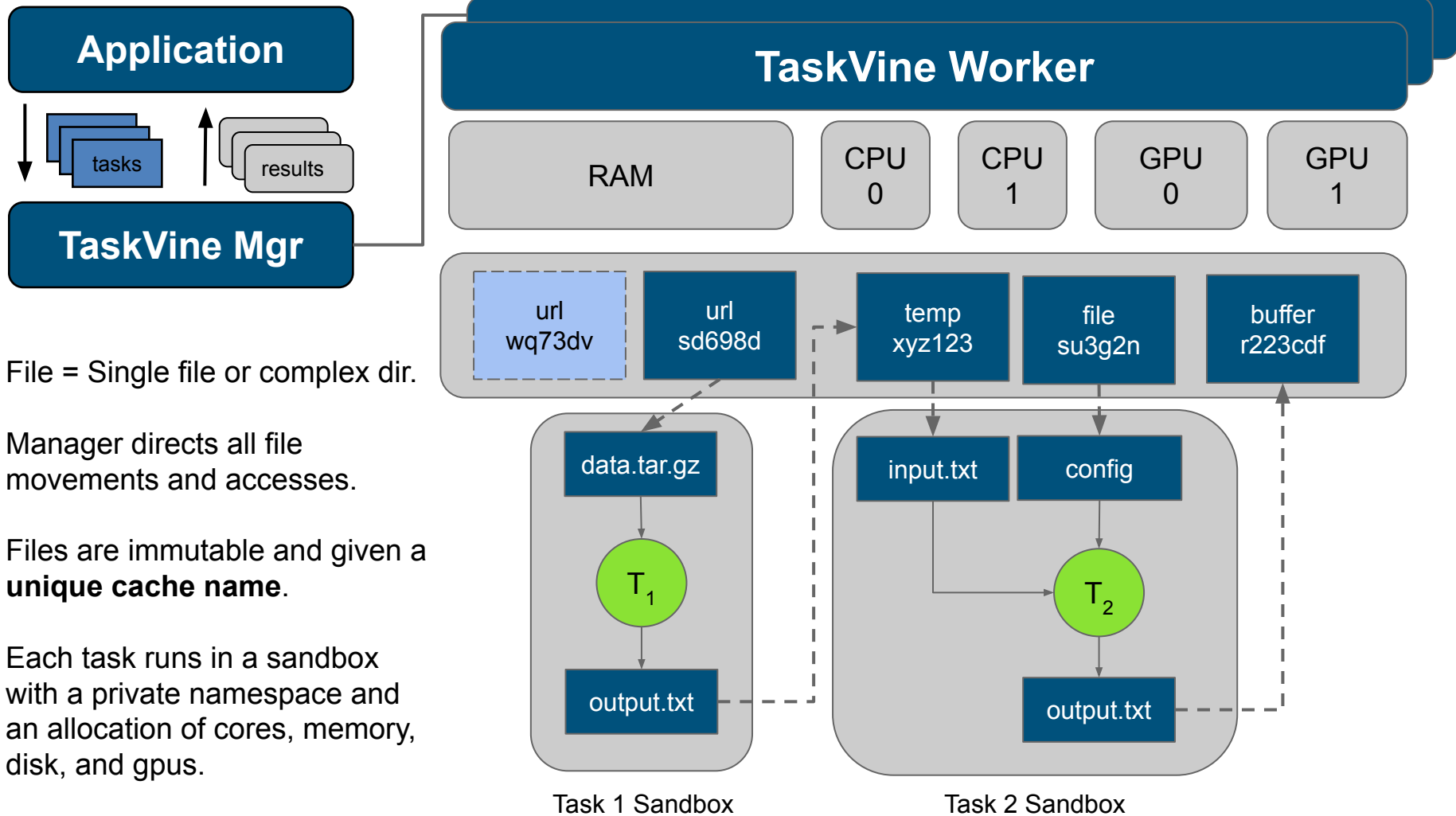
TaskVine controls the computation **and storage** capability of a large number of workers, striving to carefully manage, transfer, and re-use data and software wherever possible.

# TaskVine Architecture Overview



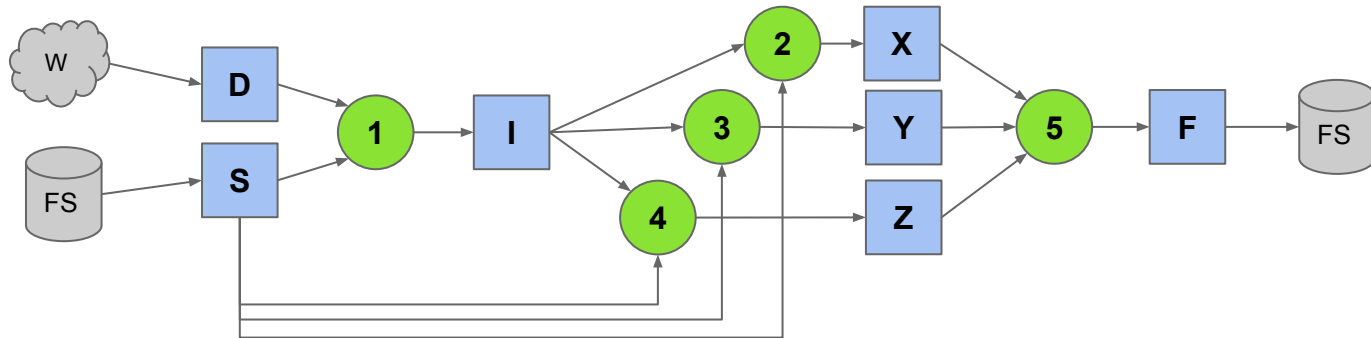
## Design Goals for TaskVine

- **Make it easy** to construct *dynamic* workflows with millions of tasks running on thousands of cluster nodes.
- **Handle common failures** by detecting and recovering from worker crashes, network failures, and other unexpected events.
- **Avoid moving data** wherever possible: leave data in place until it needs to be moved or duplicated.
- **Re-use data objects** within and across workflows by tracking provenance from original sources all the way to final outputs.
- **Manage task resources** (cpu, gpu, mem, disk) carefully in order to pack in as much as we can (but not too much!) into each worker.
- **Support complex software** environments built from package managers by explicitly naming dependencies of tasks.



# In-Cluster Data Management

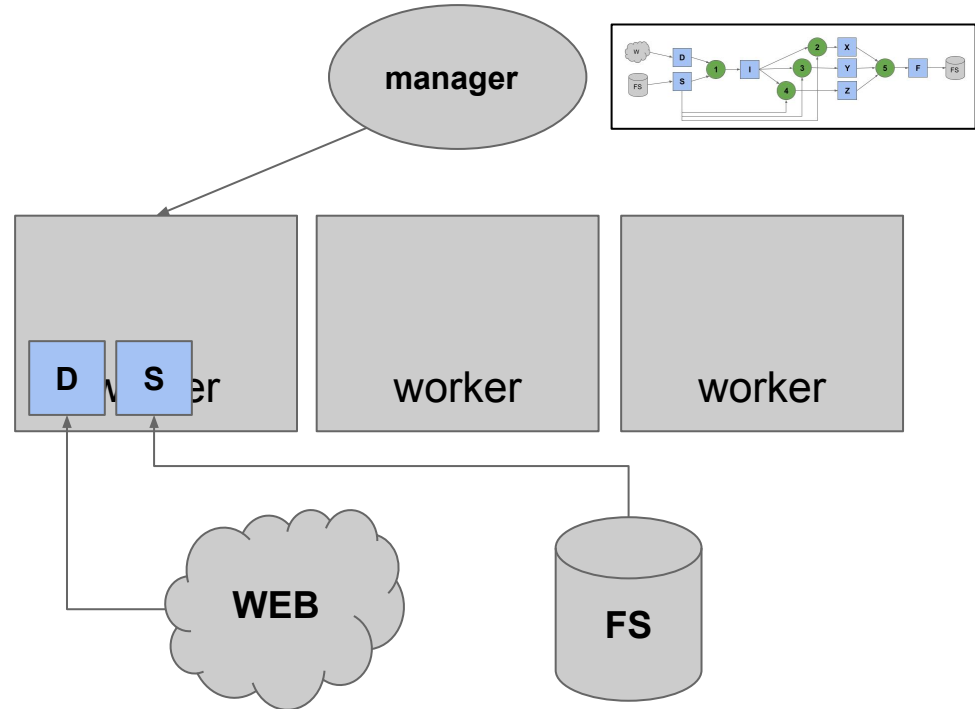
Suppose you have a workflow like this: a dataset D comes from a web repository, a software package S comes from the shared filesystem. After passing through tasks 1-5, the final output F should be written to the filesystem. TaskVine aims to keep all of the data within the cluster, as follows.





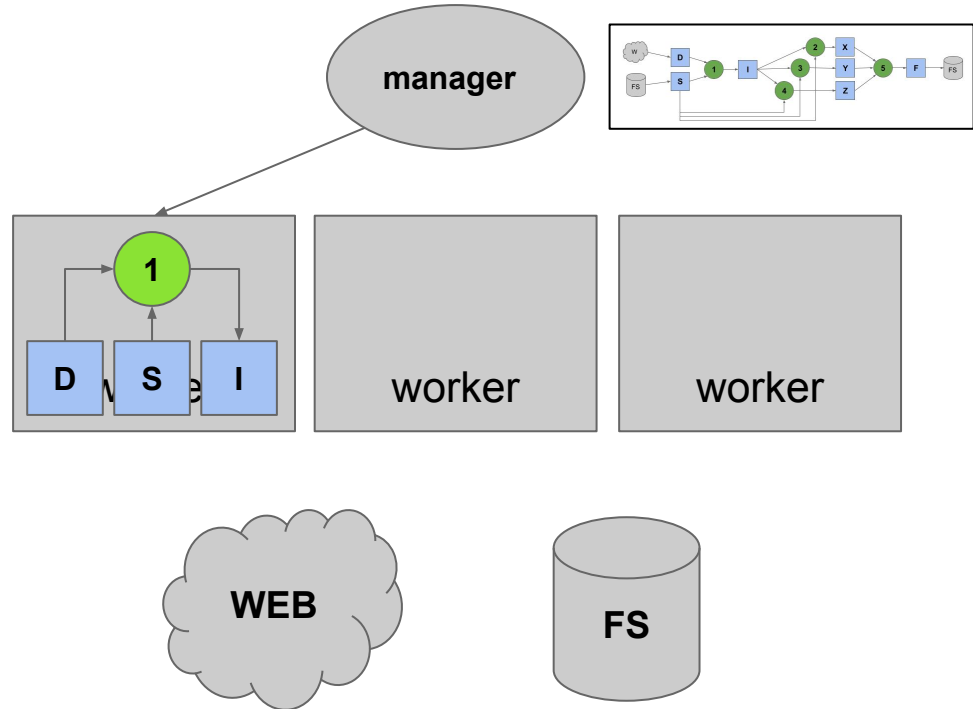
# In-Cluster Data Management

The manager selects a worker for task 1, and then directs dataset D to be downloaded from the web, and software package S to be loaded from the shared filesystem.



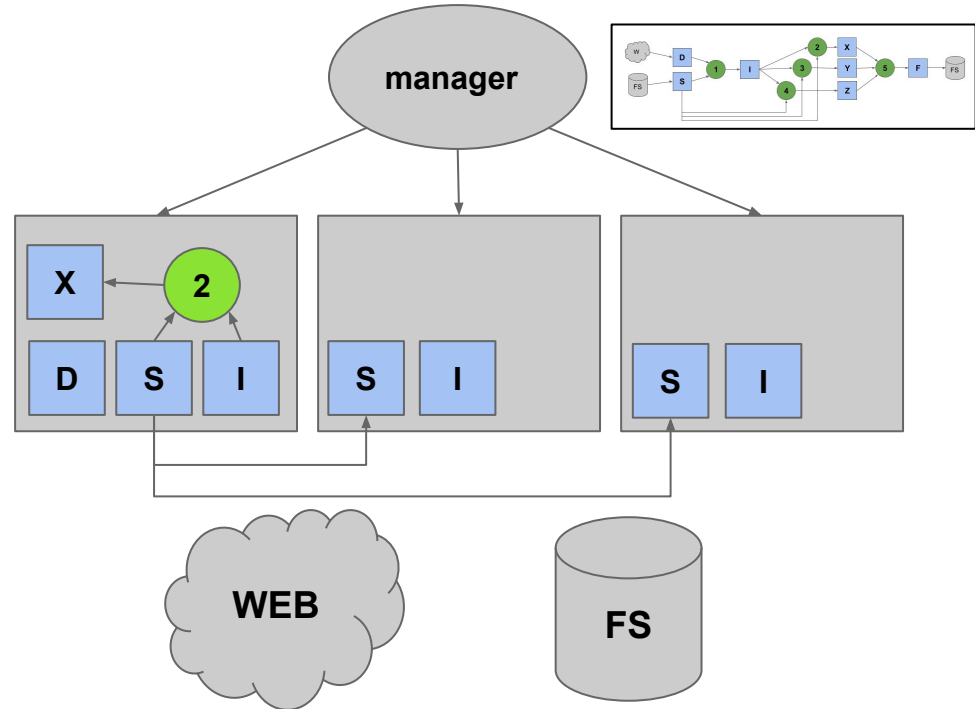
# In-Cluster Data Management

Next, task 1 is dispatched to that worker, where it reads dataset D, runs software package S, and produces file I, which stays where it is created.



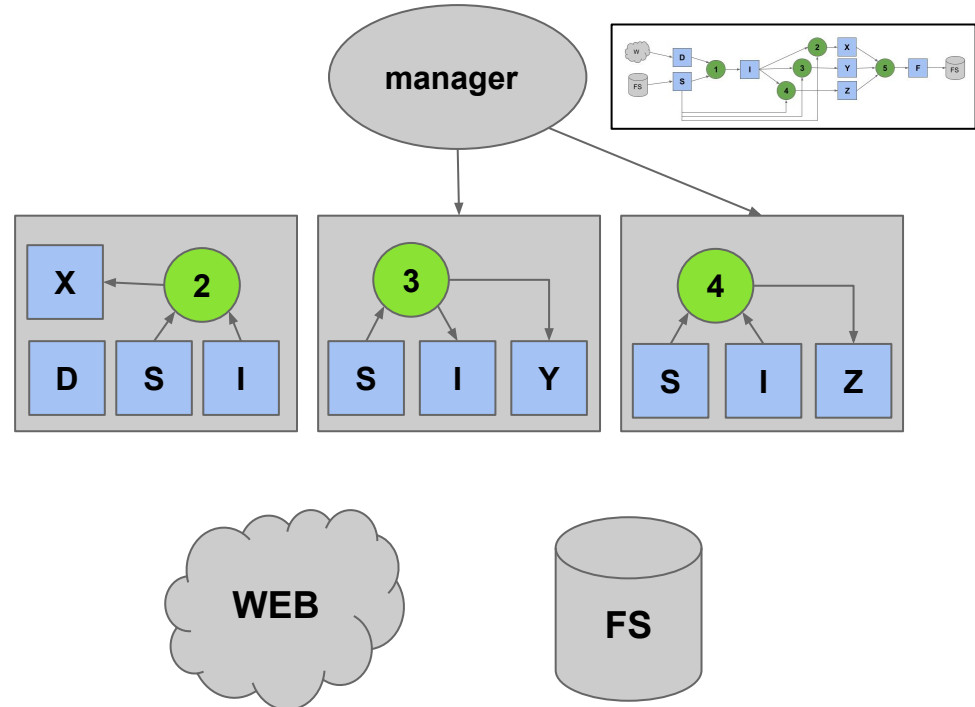
# In-Cluster Data Management

Once file I is created, task 2 can run immediately on that node, producing file X. Software package S and file I are duplicated to the other worker nodes.



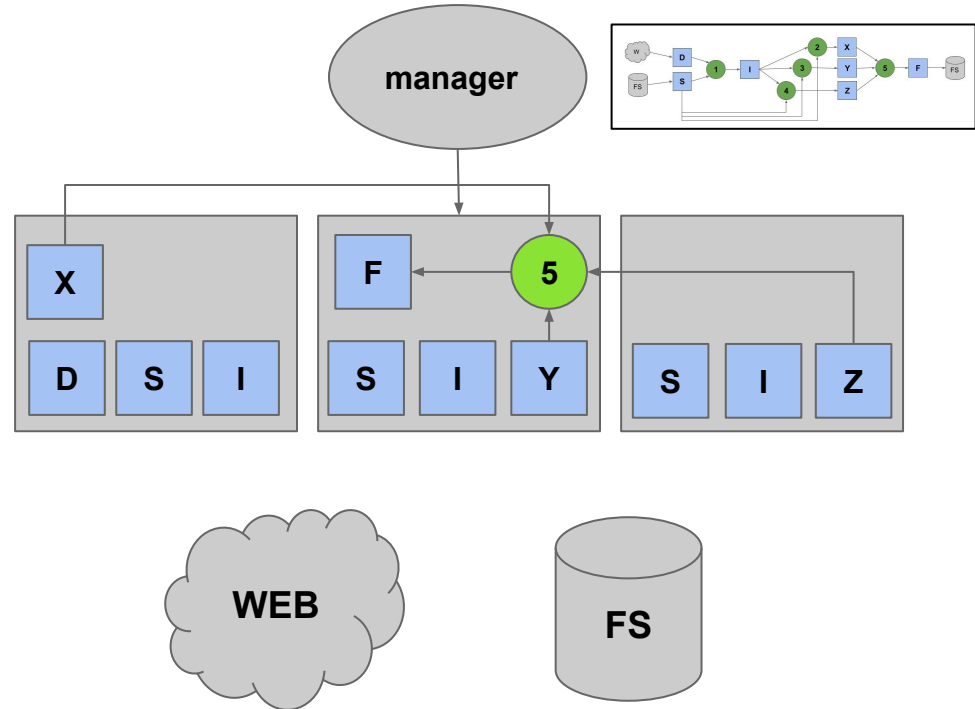
# In-Cluster Data Management

Now tasks 3 and 4 can run on the other worker nodes, producing files Y and Z.



# In-Cluster Data Management

Next, task 5 is dispatched to the middle worker. It consumes files X, Y, and Z, which are pulled in from peer nodes. The output file X is produced on that node.

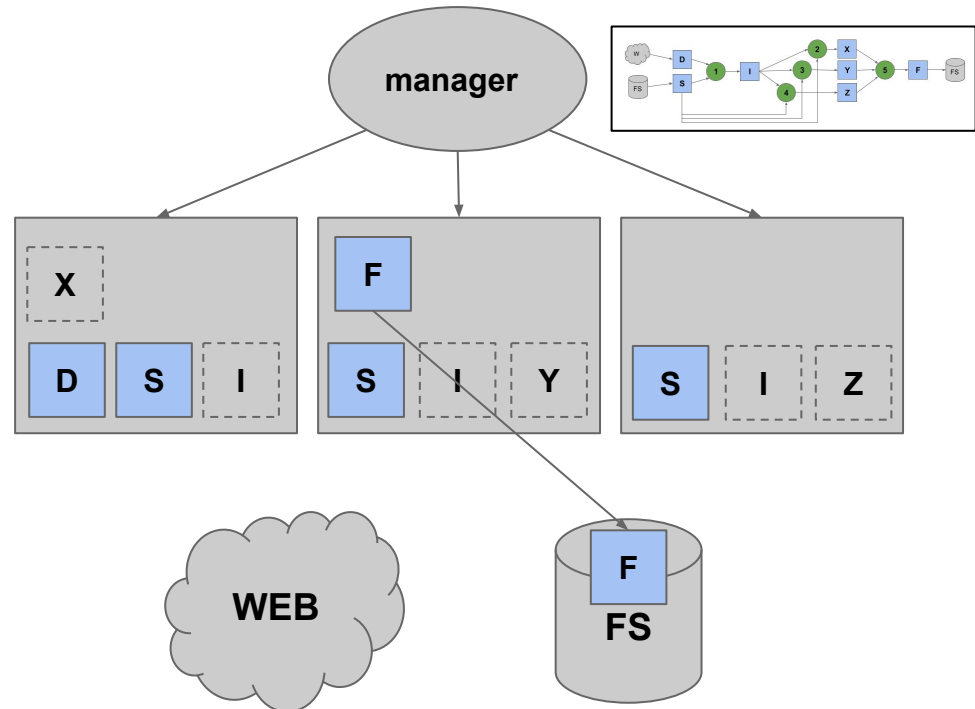


# In-Cluster Data Management

Finally, output file F is written back to the shared filesystem, as the ultimate output of the workflow.

The manager directs the workers to delete any remaining uncacheable files.

Common input files remain to accelerate future workflows.



## API: Declare Files Explicitly

```
import ndcctools.taskvine as vine

m = vine.Manager(9123)

file      = m.declareFile("mydata.txt")
buffer    = m.declareBuffer("Some literal data")
url       = m.declareURL("https://somewhere.edu/data.tar.gz")
temp      = m.declareTemp();

data      = m.declareUntar( url )
package   = m.declareStarch( executable )
```

## API: Connect Tasks to Files

```
task = vine.Task("mysim.exe -p 50 input.data -o output.data")

t.add_input(url, "input.data")
t.add_output(temp, "output.data")

t.set_cores(4)
t.set_memory(2048)
t.set_disk(100)
t.set_tag("simulator")

taskid = m.submit(t)
```



## API: Execute Python Function

```
task = vine.PythonTask(simulate_func, molecule, parameters)

t.set_cores(4)
t.set_memory(2048)
t.set_disk(100)
t.set_tag("simulator")

taskid = m.submit(t)

. . .

print(t.result)
```

# Sample Application: NCBI Blast

```

blast_url="https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+
/LATEST/ncbi-blast-2.13.0+-x64-linux.tar.gz"

landmark_url =
"https://ftp.ncbi.nlm.nih.gov/blast/db/landmark.tar.gz"

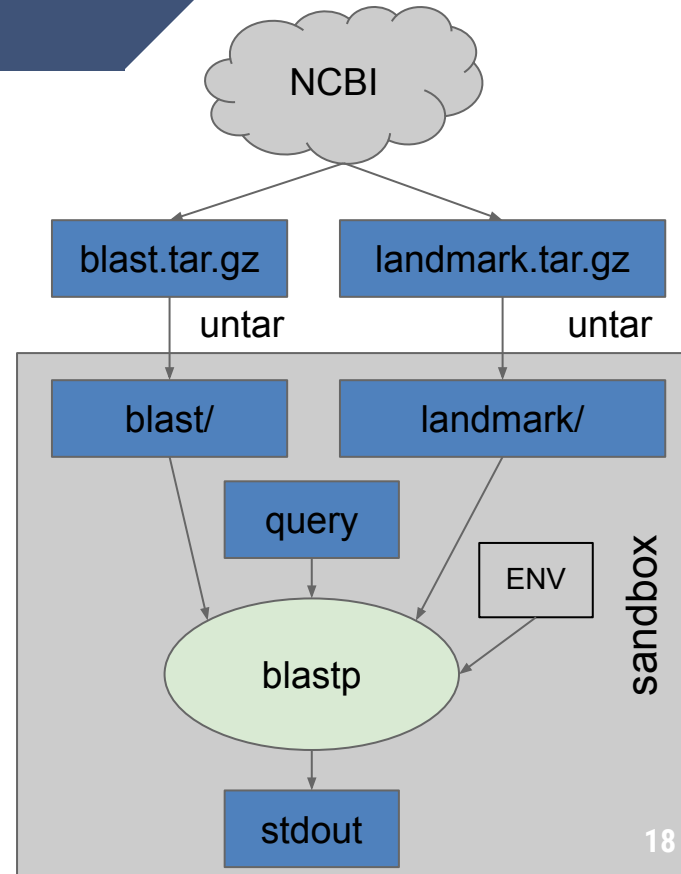
query_string = "GCTAATCCA..."

software = m.declareUntar(m.declareURL(blast_url))
landmark = m.declareUntar(m.declareURL(landmark_url))

task = vine.Task("blastp -db landmark -query query.file")
task.add_input(software, "blastdir")
task.add_input(database, "landmark")
task.add_input_buffer(query_string, "query.file")
task.set_env_var("BLASTDB", value="landmark")

m.submit(task)

```



# Mini-Tasks: FileUntar

```
blast_url="https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+
/LATEST/ncbi-blast-2.13.0+-x64-linux.tar.gz"

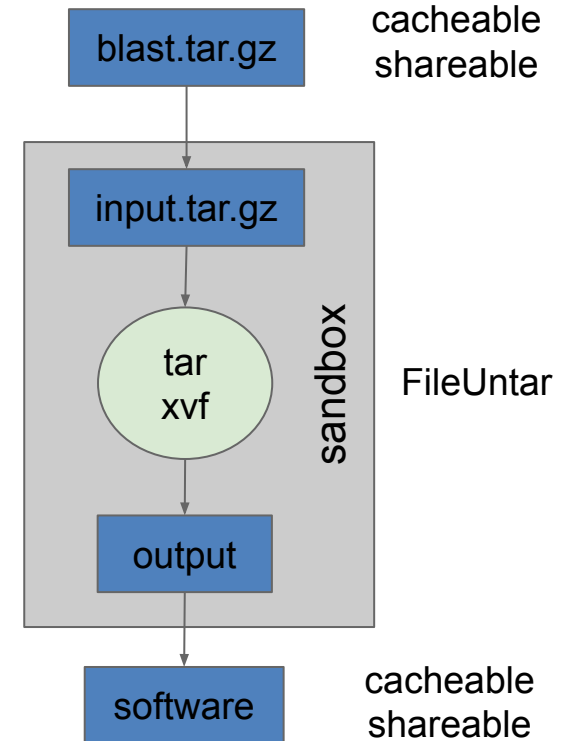
landmark_url =
"https://ftp.ncbi.nlm.nih.gov/blast/db/landmark.tar.gz"

query_string = "GCTAATCCA..."

software = m.declareUntar(m.declareURL(blast_url))
landmark = m.declareUntar(m.declareURL(landmark_url))

task = vine.Task("blastp -db landmark -query query.file")
task.add_input(software, "blastdir")
task.add_input(database, "landmark")
task.add_input_buffer(query_string, "query.file")
task.set_env_var("BLASTDB", value="landmark")
```

Upshot: Common data prep done once for many tasks on a node.



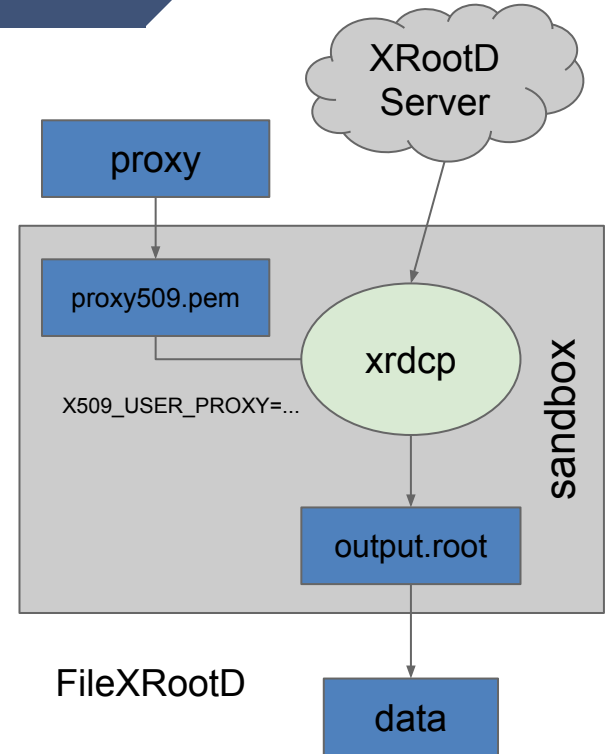
# Mini-Task: FileXRootD

New capabilities are added to the system by defining mini-tasks that use the same task infrastructure to define dependencies and execute them reproducibly:

```
data = m.declareXRootD( "xrootd://host/path", "proxy" )
```

Which is defined as a mini-task like this:

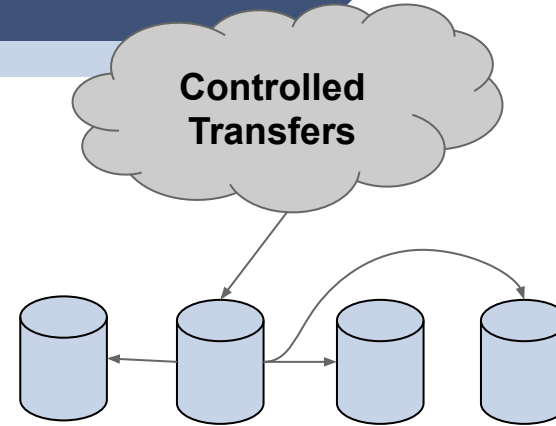
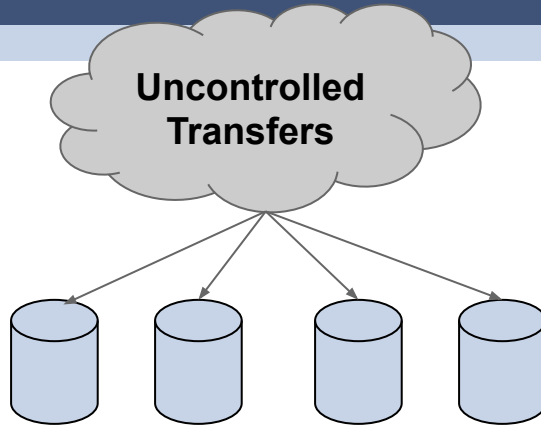
```
t = vine.Task("xrscp {} output.root".format(url));
t.add_input(proxy, "proxy509.pem")
t.set_env_var("X509_USER_PROXY", "proxy509.pem")
data = m.declareMiniTask(t, "output.root")
```



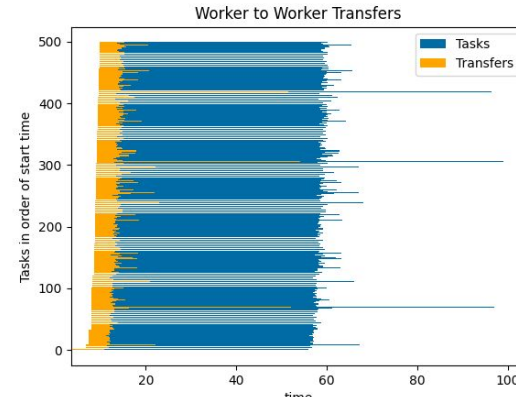
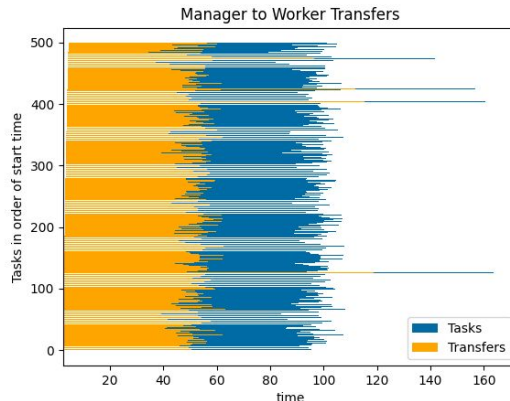
# Manager Schedules Transfers



Colin Thomas

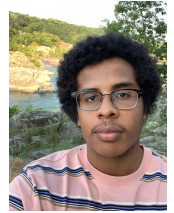


500 task BLAST workflow (from above) requires both software and data from NCBI.



Obtaining and deploying assets is part of the workflow itself!

# Naming Objects for Persistent Storage

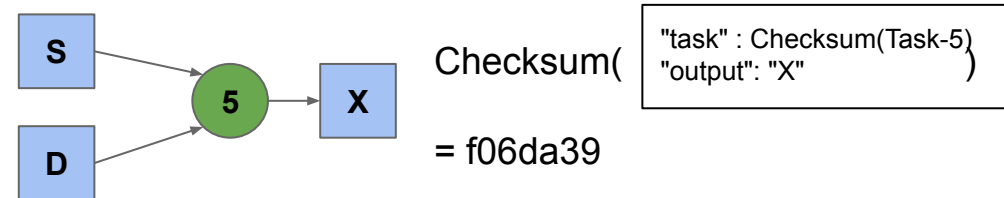
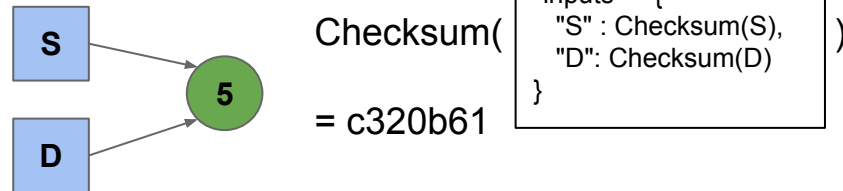
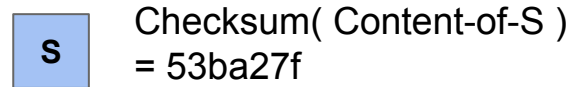


Barry Sly-Delgado

Files have one of three lifetimes:

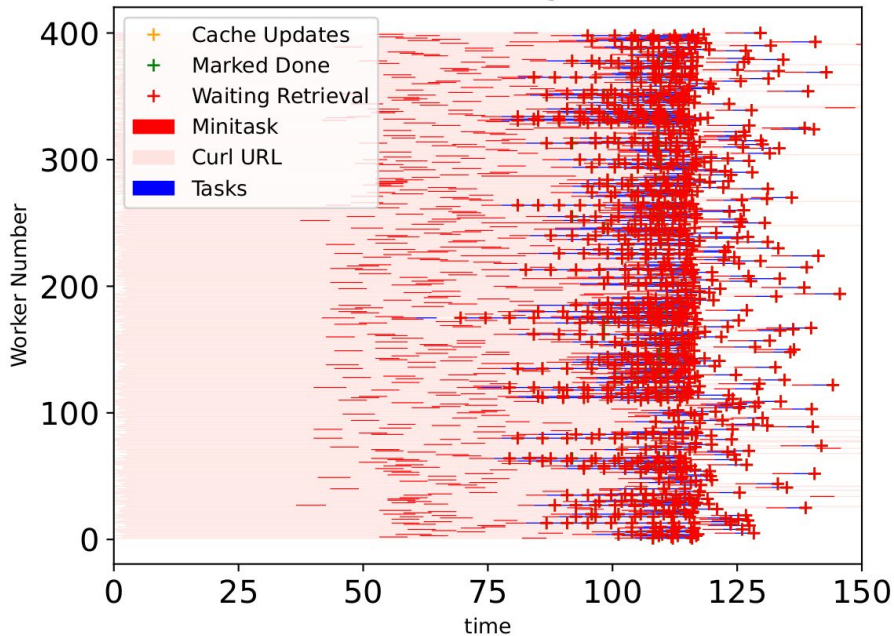
- single-task
- workflow (default)
- forever

"forever" cached objects are given content addressable names from a **Merkle Tree** of the file's provenance. If any inputs change, then so does the name of the output, and it's not the same file.

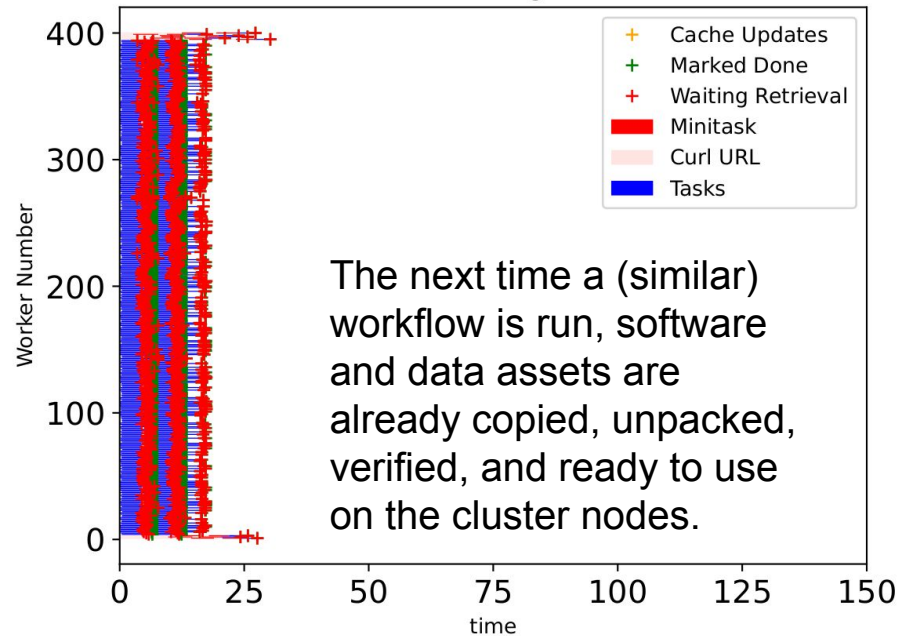


# Eliminating Startup Costs

No Caching



Caching



The next time a (similar) workflow is run, software and data assets are already copied, unpacked, verified, and ready to use on the cluster nodes.

# Functions as a Service - Install Library

```

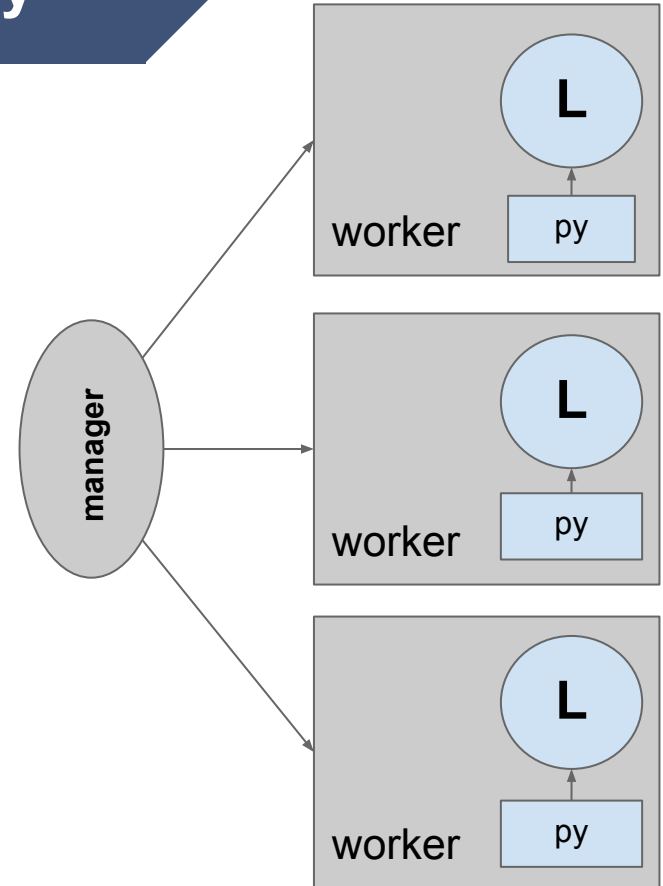
# Define ordinary Python functions
def my_sum(x, y):
    return x+y

def my_mul(x, y):
    return x*y

# Create a library object from functions
L = m.create_library_from_functions(
    "my_library", my_sum, my_mul)

# Install the library on all workers.
m.install_library(L)

```





# Functions as a Service - Invoke It

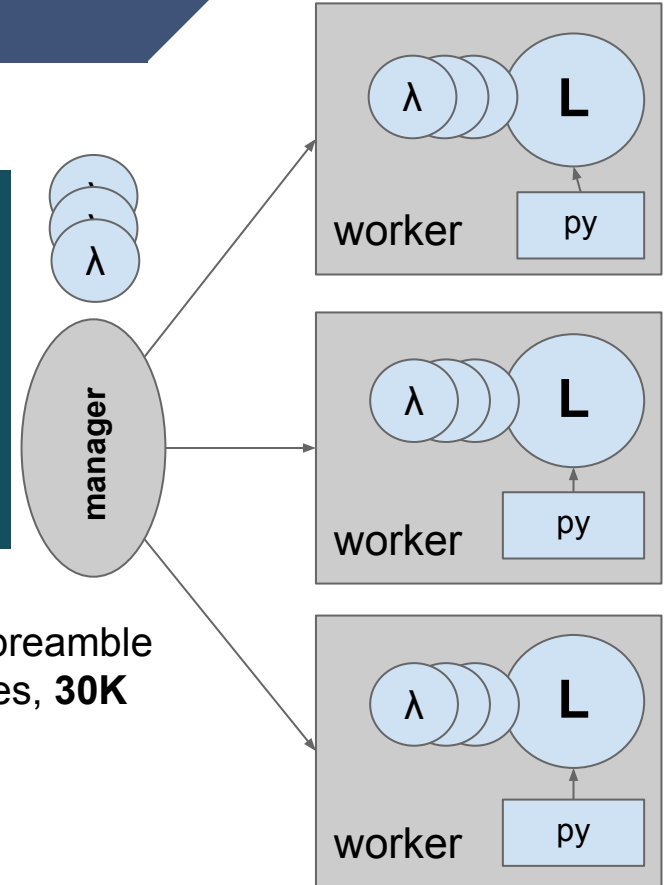
```
# Define a function invocation and submit it
```

```
for i in range(1,100):
    t = vine.FunctionCall("my_library", "my_sum", 10, i)
```



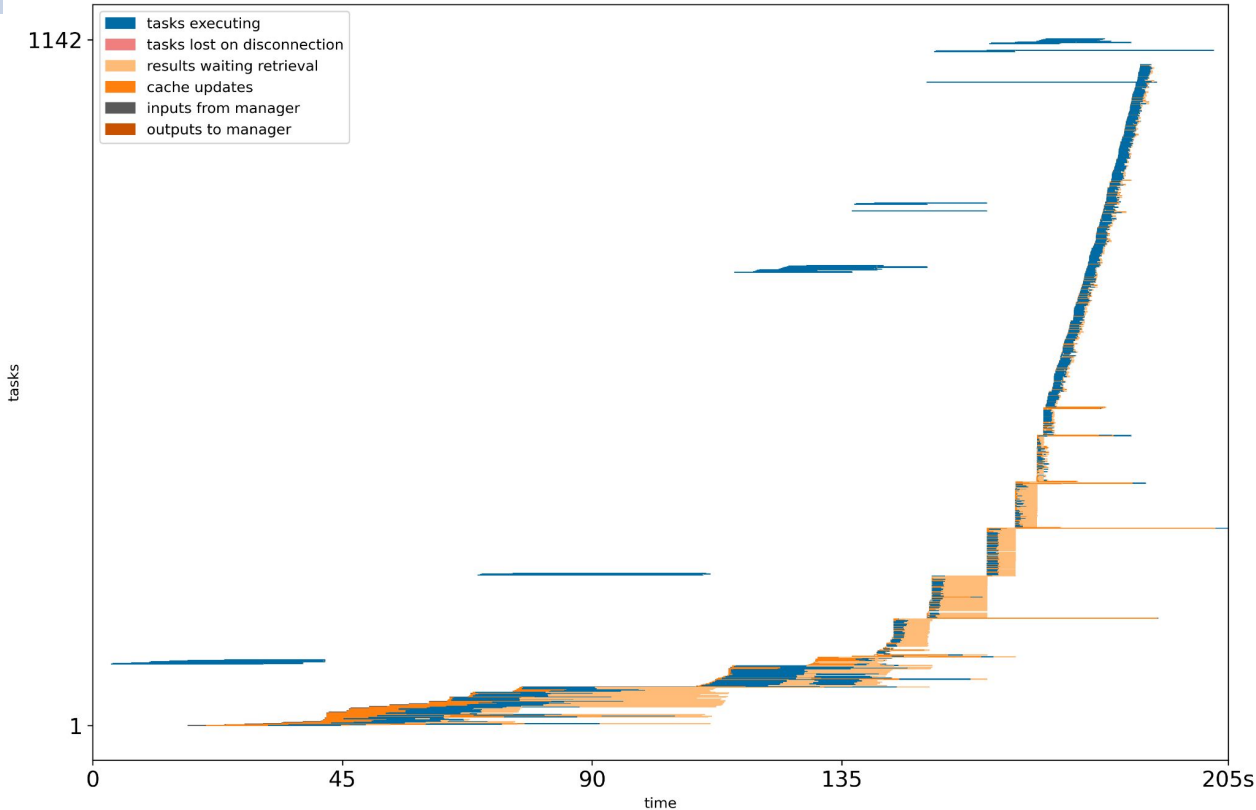
David Simonetti

Simply converting "import tensorflow" into the preamble of a Library task saves **1.2GB** of Python libraries, **30K** metadata system calls, and **5-10s** latency per FunctionCall.



# Multi-Modal Workflows

TaskVine



100x Standard Tasks  
Build model from MNIST data.

For each produced model:  
Deploy LibraryTask for inference.

Submit 10x FunctionCalls that  
invoke each LibraryTask.

Application gradually accelerates  
as standard tasks produce data  
that define libraries that can then  
be invoked.

# Application: TopEFT in WQ

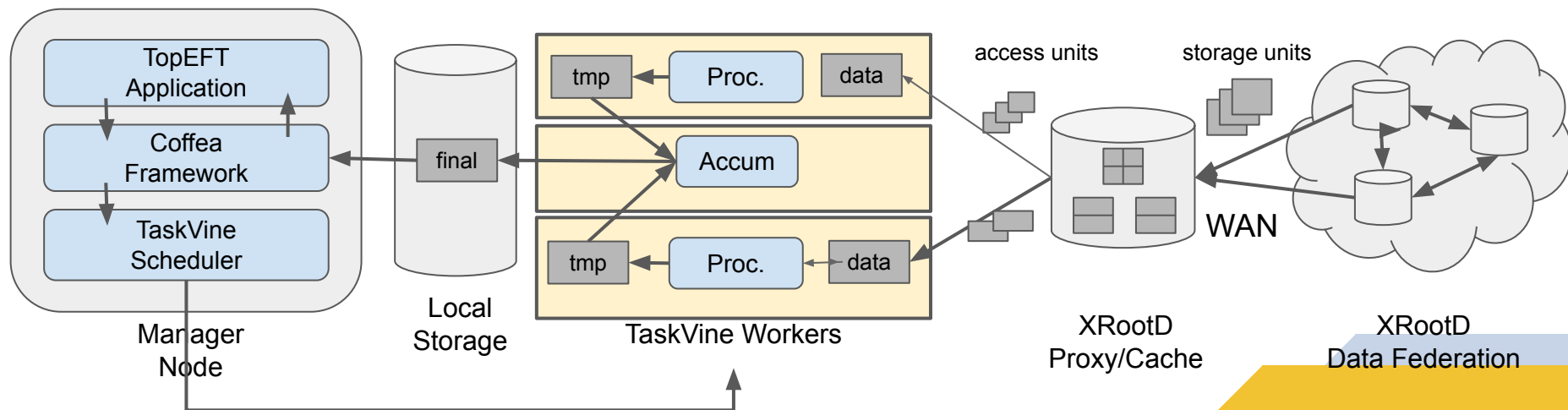


**Kelci  
Mohrman**

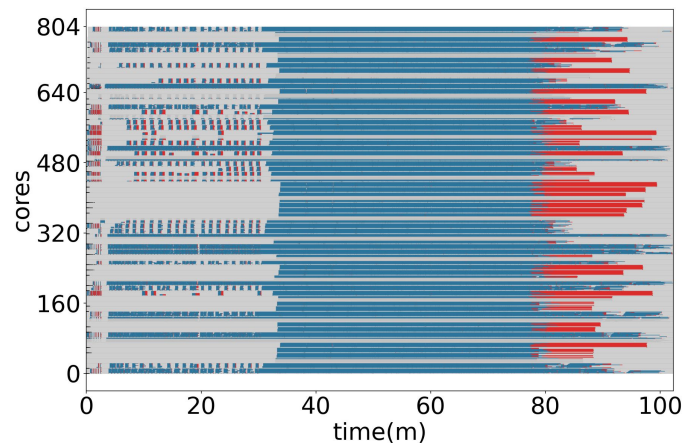
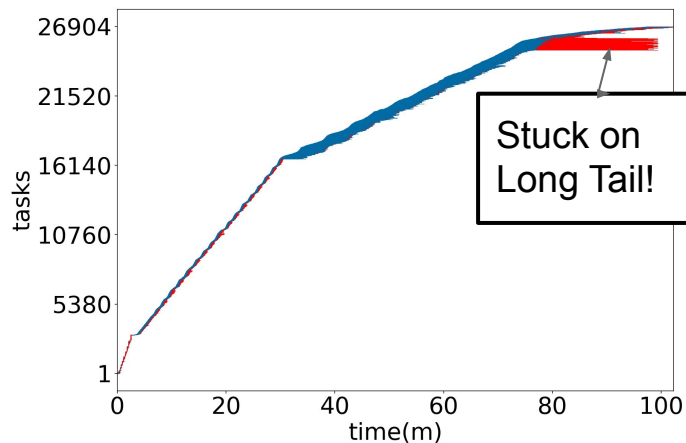


**Kevin  
Lannon**

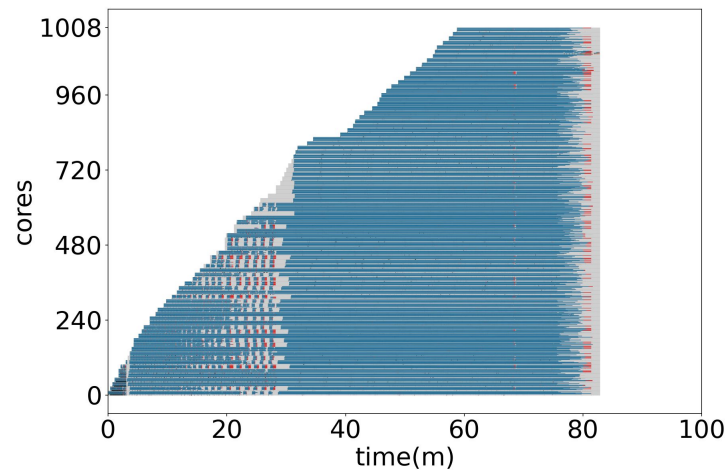
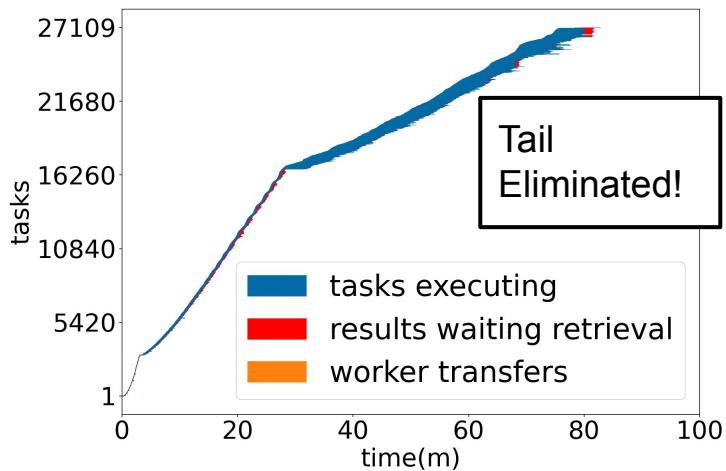
- Late stage data analysis for LHC CMS experiment. Search for new physics impacting associated top quark production using the framework of effective field theory (EFT). TopEFT uses Coffea HEP framework and scientific python components.



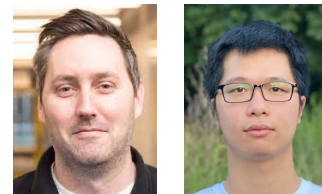
**Old:**  
**Accumulation**  
**Data Returned**  
TopEFT  
+ Work Queue



**New:**  
**In-Cluster**  
**Accumulation**  
TopEFT  
+ TaskVine



# WIP: Parsl + TaskVine



Kyle Chard U. Chicago  
Thanh Phung

```
# Estimate three values for pi
a, b, c = pi(10**6), pi(10**6), pi(10**6)

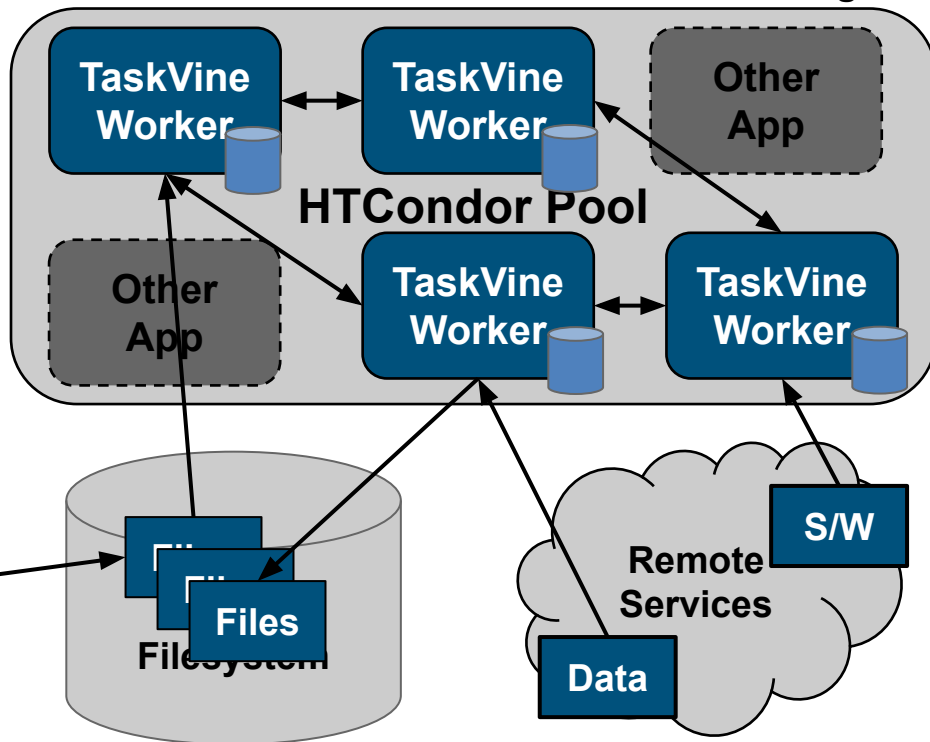
# Compute the mean of the three estimates
mean_pi = mean(a, b, c)

# Print the results
print("a: {:.5f} b: {:.5f} c: {:.5f}".format(a.result(), b.r
print("Average: {:.5f}".format(mean_pi.result()))
```

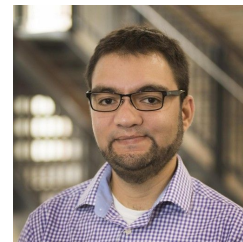
Parsl Data Flow Kernel

Parsl + TaskVine Exec

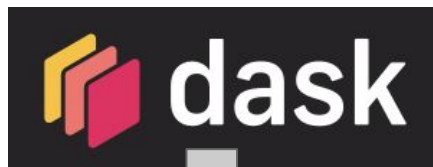
TaskVine Mgr



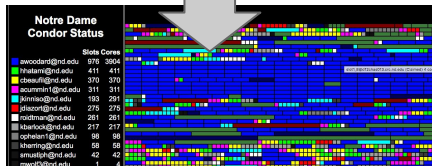
# WIP: TaskVine and Dask



Ben Tovar



```
# Dask Task Graph
d = {'x': 1,
     'y': (inc, 'x'),
     'z': (add, 'y', 10)}
```



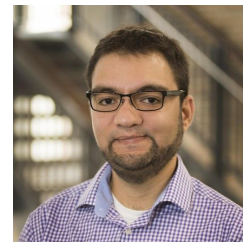
```
import dask
import dask.array as da
```

```
x = da.random.random((10000,10000), chunks=5000)
y = x + x.T
z = y[:, :2, 500:].mean(axis=1)
```

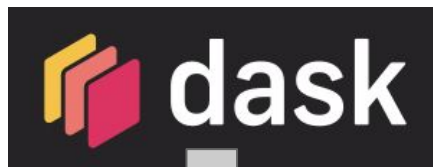
```
result = z.compute()
```

```
print(result);
```

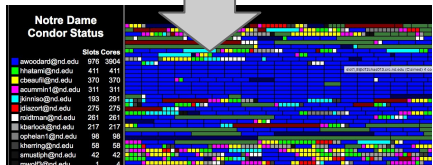
# WIP: TaskVine and Dask



Ben Tovar



```
# Dask Task Graph
d = {'x': 1,
     'y': (inc, 'x'),
     'z': (add, 'y', 10)}
```



```
import ndcctools.taskvine as vine
import dask
import dask.array as da
```

```
# Create a new manager listening on port 9123
manager = vine.DaskVine(9123)
```

```
x = da.random.random((10000,10000), chunks=5000)
```

```
y = x + x.T
```

```
z = y[:, :2, 500:].mean(axis=1)
```

```
result = z.compute(manager.get())
```

```
print(result);
```

# Research Challenges

- Decomposing DAGs of Short Tasks
  - Dask and Parsl can produce  $O(1M)$  function evals that may be less than one second each.
- Automatically Identifying Serverless Candidates
  - Can we recover the cost of deployment?
- Dynamic Resource Management
  - How to choose resources for raw functions?
- Dependency Management Challenges
  - Do you know what your code depends upon?
  - Do you want to use what others depend upon?
  - What are your expectations regarding updates?





# Current Status of TaskVine

This work was supported by  
NSF Award OAC-1931348

- **TaskVine** is a component of the Cooperative Computing Tools (cctools) from Notre Dame alongside Makeflow, Work Queue, Resource Monitor, etc.
- Second release made in July 2023.
- Research software with an engineering process: issues, tests, manual, examples.
- We are eager to collaborate with new users on applications and challenges!

```
conda install -c conda-forge ndcctools
```

<https://cctools.readthedocs.io>

The screenshot shows a web browser displaying the TaskVine User's Manual page. The page features a navigation sidebar on the left with sections for 'GETTING STARTED' (About, Installation, Getting Help), 'SOFTWARE' (TaskVine, Overview, Quick Start, Example Applications, Writing a TaskVine Application, Running a TaskVine Application, Advanced Data Handling, Advanced Task Handling, Python Programming Models, Managing Resources, Logging and Plotting Facilities, Specialized and Experimental Settings), 'Work Queue', 'Makeflow', 'JX Workflow Language', 'Resource Monitor', and 'Parrot'. The main content area is titled 'TaskVine User's Manual' and includes an 'Overview' section. The overview text states: 'TaskVine is a framework for building large scale data intensive dynamic workflows that run on high performance computing (HPC) clusters, GPU clusters, cloud service providers, and other distributed computing systems. A workflow is a collection of programs and files that are organized in a graph structure, allowing parts of the workflow to run in a parallel, reproducible way:'. Below the text is a diagram of a workflow graph with nodes labeled D, T, X, Y, Z, F, and FS, connected by arrows. The diagram shows a sequence of tasks: D (with input W) leads to T (with input S), which leads to X (with input 2), Y (with input 3), and Z (with input 4). X, Y, and Z all lead to F, which leads to FS. The diagram also shows a feedback loop from FS back to D. The page footer includes a 'v: stable' dropdown menu and 'Previous Next' navigation links.

# For more information...



This work was supported by  
NSF Award OAC-1931348

<https://ccl.cse.nd.edu/software/taskvine>

<https://dthain.github.io>



Douglas Thain  
Director



Benjamin Tovarr  
Research  
Soft. Engineer



Thanh Son Phung  
Ph.D. Student



Barry Sly Delgado  
Ph.D. Student



Colin Thomas  
Ph.D. Student



David Simonetti  
Undergraduate



Joe Duggan  
Undergraduate



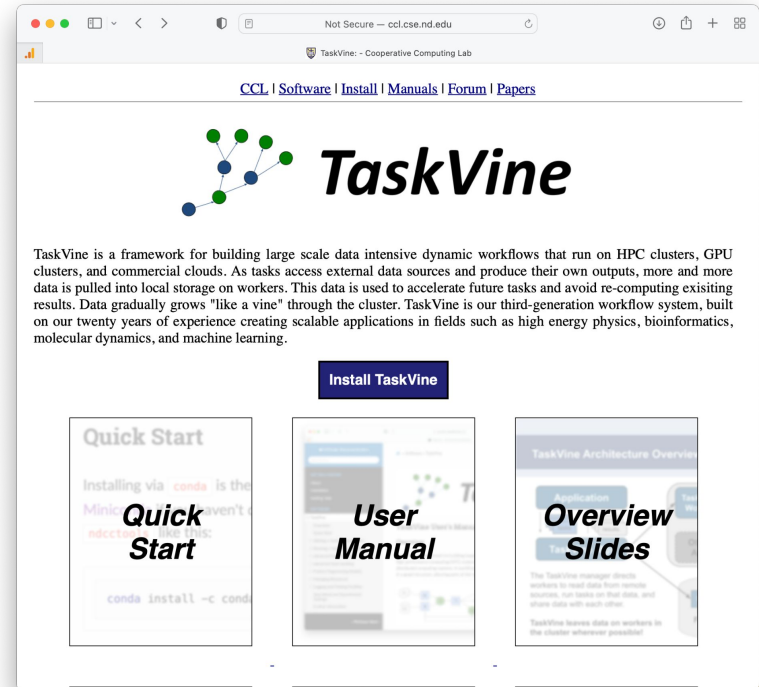
Andrew Hennessee  
Undergraduate



Matt Carbonaro  
Undergraduate



Jacob Dolak  
Undergraduate



The screenshot shows a web browser window displaying the TaskVine website. The browser's address bar shows "Not Secure - ccl.cse.nd.edu". The website header includes navigation links: "CCL | Software | Install | Manuals | Forum | Papers". The main heading is "TaskVine" with a logo of a network graph. Below the heading is a paragraph describing TaskVine as a framework for building large scale data intensive dynamic workflows that run on HPC clusters, GPU clusters, and commercial clouds. A blue button labeled "Install TaskVine" is positioned above three content cards: "Quick Start", "User Manual", and "Overview Slides".

# Extra Slides



# TaskVine Worker

RAM

CPU  
0

CPU  
1

GPU  
0

GPU  
1

url  
sd698d

temp  
xyz123

file  
f19xa2

url  
c03rd5

data.tar.gz

T

output.txt

data

F

result

software

L

logfile.txt

fork

Standard Task

FunctionCall

LibraryTask

Simply converting "import tensorflow" into the preamble of a Library task saves **1.2GB** of Python libraries, **30K** metadata system calls, and **5-10s** latency per FunctionCall. We can mix standard Tasks, Libraries, and FunctionCalls in the same workflow:



David  
Simonetti