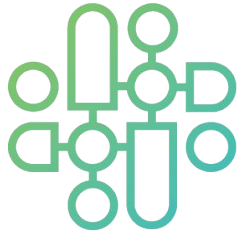


SC23
Denver, CO | i am hpc.

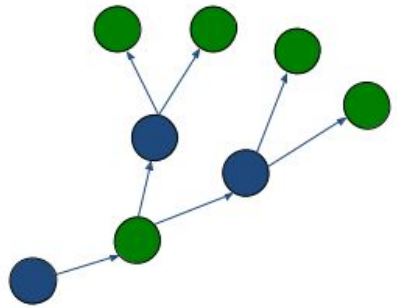
TaskVine: Managing In-Cluster Storage for High-Throughput Data-Intensive Workflows

Barry Sly-Delgado, Thanh Son Phung, Colin Thomas, David Simonetti, Andrew Hennesee, Ben Tovar, Douglas Thain
University of Notre Dame

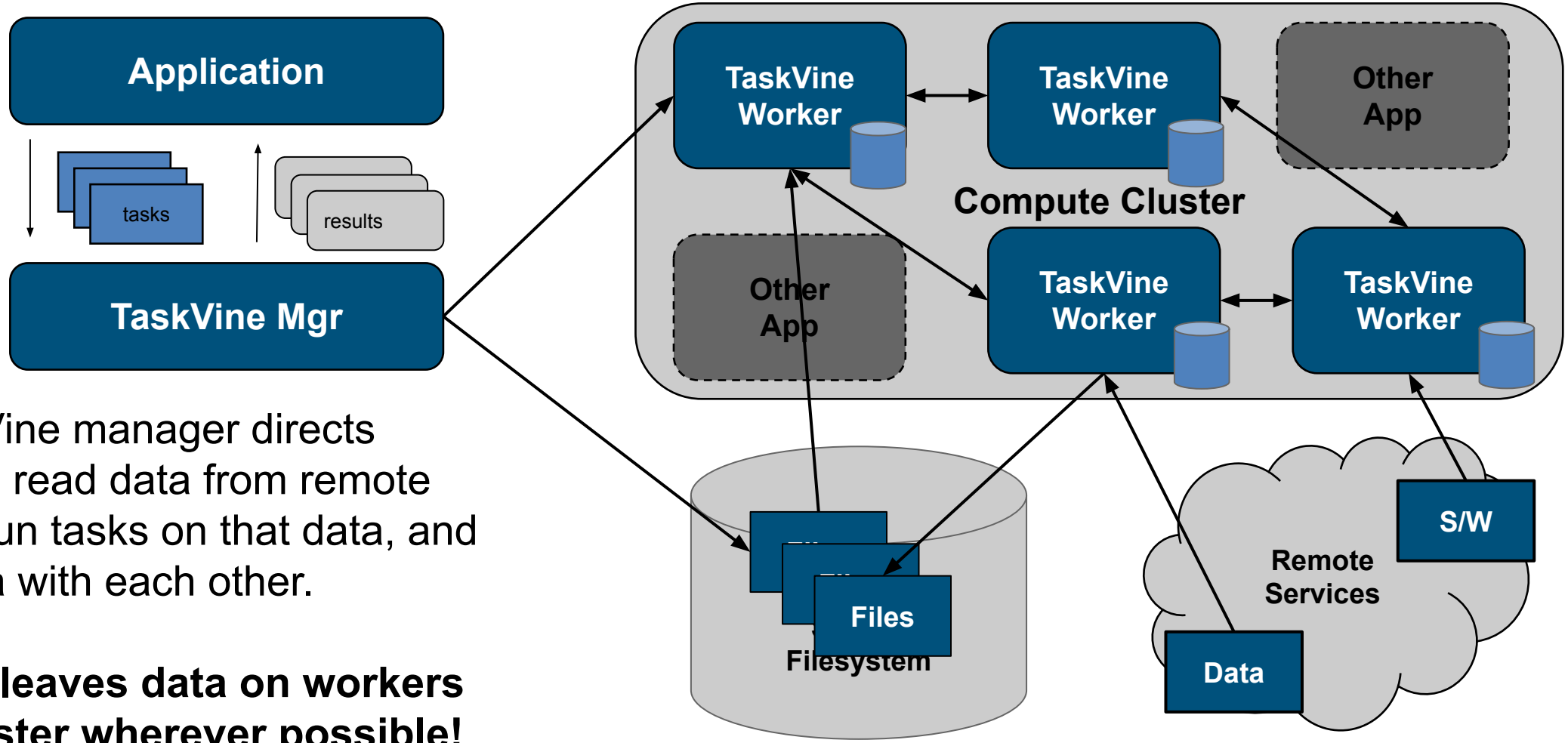
TaskVine Overview

TaskVine is a workflow executor for data intensive workflows that carefully manages data dependencies and resource utilization by utilizing in-cluster storage and bandwidth

TaskVine



TaskVine Architecture Overview

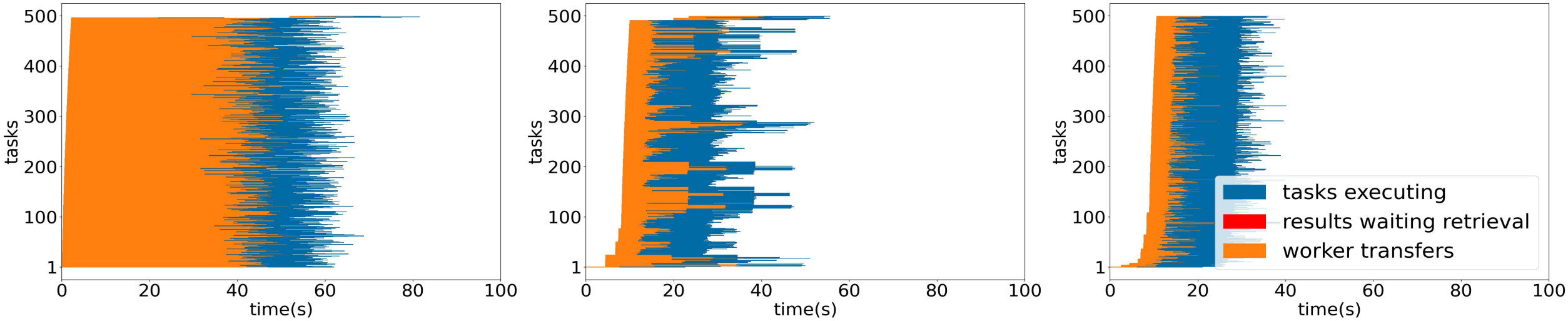


The TaskVine manager directs workers to read data from remote sources, run tasks on that data, and share data with each other.

TaskVine leaves data on workers in the cluster wherever possible!

Evaluation: Managing Transfers

Experiment: Introducing remote data into cluster. Left: each task independently downloads file. Middle: Uncontrolled peer transfers between workers. Right: Limited peer transfers between workers.



Limiting peertransfer concurrency improves performance.

TaskVine Program

Three paradigms can be combined:

- **Dynamic Workflow:** submit multiple tasks, wait for completion, consider results, submit more.
- **Static Workflow:** define entire workflow up front, wait for complete results to be returned.
- **Serverless Computing:** define functions as services, then submit lightweight invocations of those services.

Task Definitions:

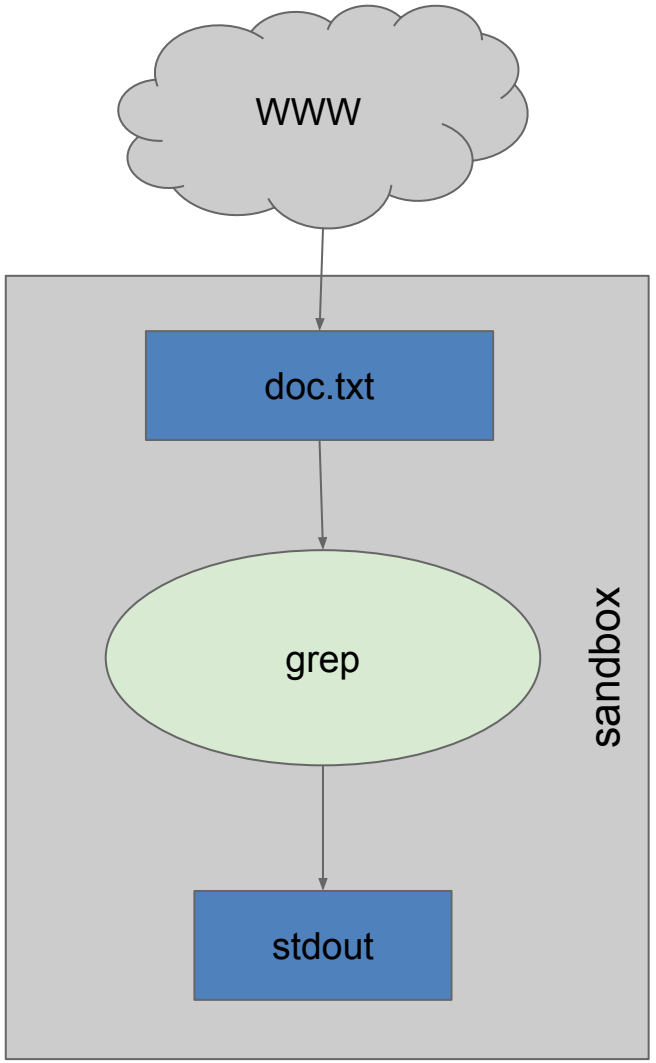
- **Regular Task**
- **Python Task**
- **Mini Task**
- **Serverless Task**

Data Definitions:

- **LocalFile**
- **TempFile**
- **BufferFile**
- **URLFile**
- **Mini Task***

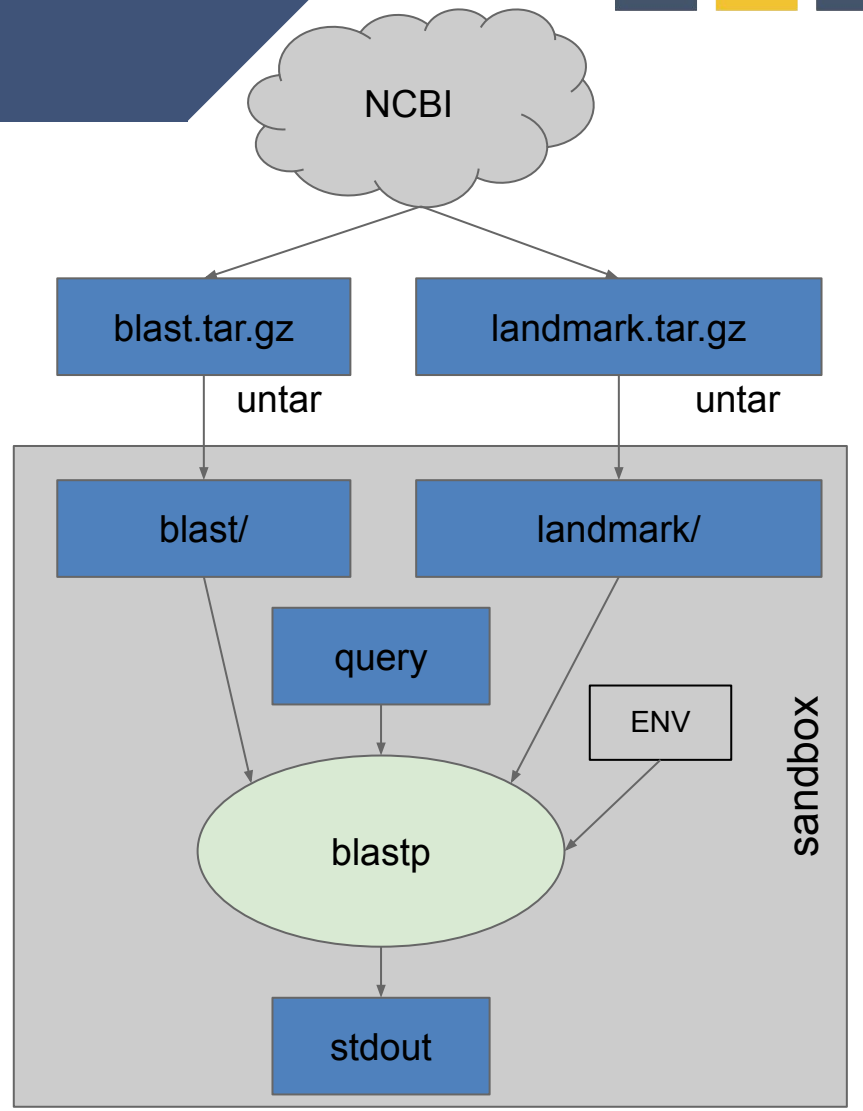
Regular Task Definition

```
import taskvine as vine

m = vine.Manager(9123)


doc = m.declareURL("https://www.gutenberg.org/files/1960/1960.txt")

task = vine.Task("grep chair doc.txt")
task.add_input(doc,"doc.txt")

taskid = m.submit(task)
task = queue.wait(VINE_FOREVER)

print task.output
```



TaskVine Application

```
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)
```



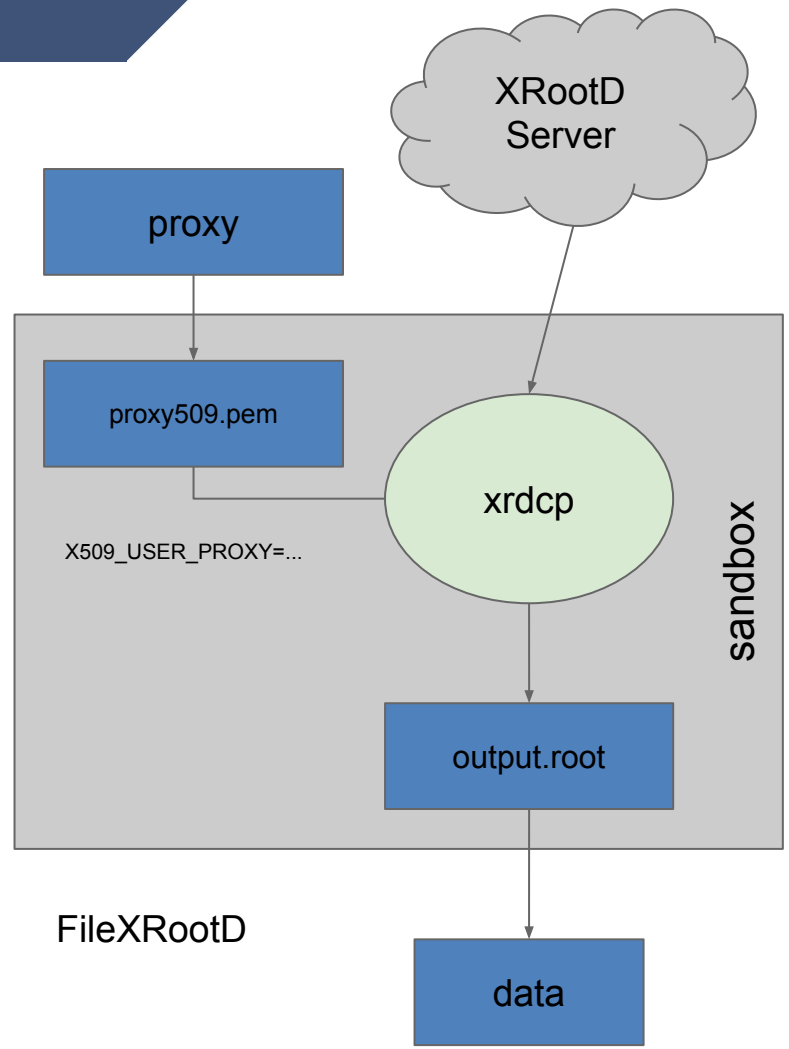
Mini-Tasks

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" )
```

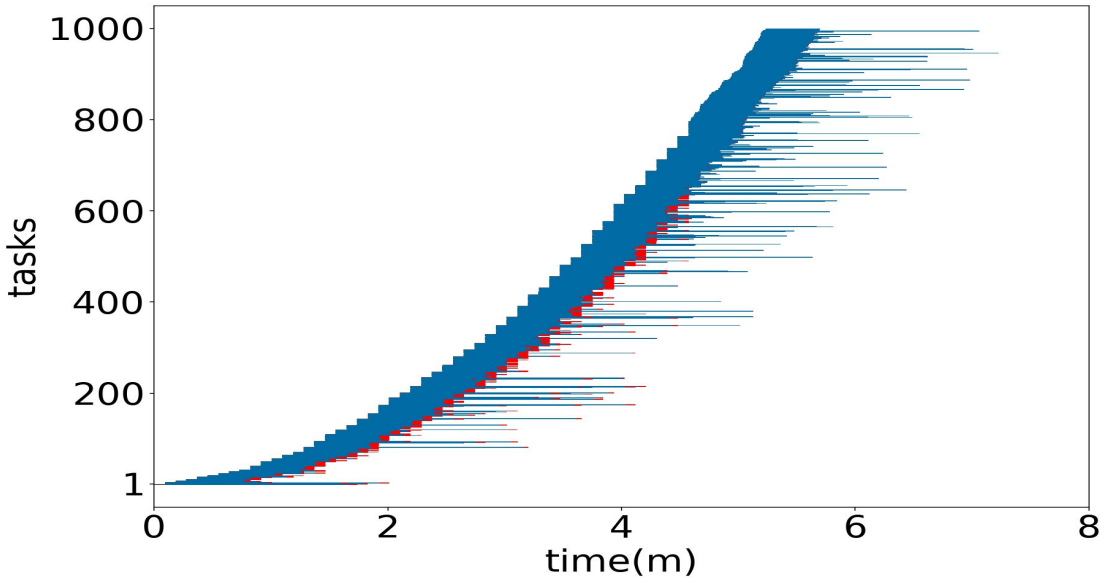
Which is defined as a mini-task like this:

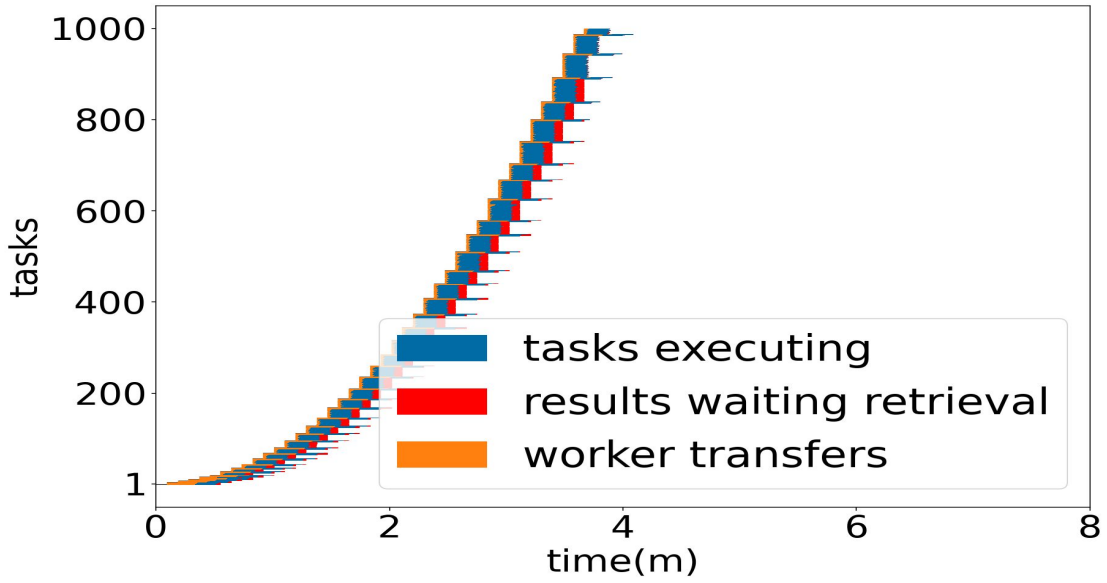
```
t = vine.Task("xrdcp {} output.root".format(url));
t.add_input(proxy,"proxy509.pem")
t.set_env_var("X509_USER_PROXY","proxy509.pem")
data = m.declareMiniTask(t,"output.root")
```



Evaluation: Mini Tasks

Experiment:Mini Tasks enable tasks to share staged data, even when it requires some transformation following transfer. Left: Each task expands the environment itself as part of its own task definition. Right: Each task shares an expanded environment defined by a shared mini-task.

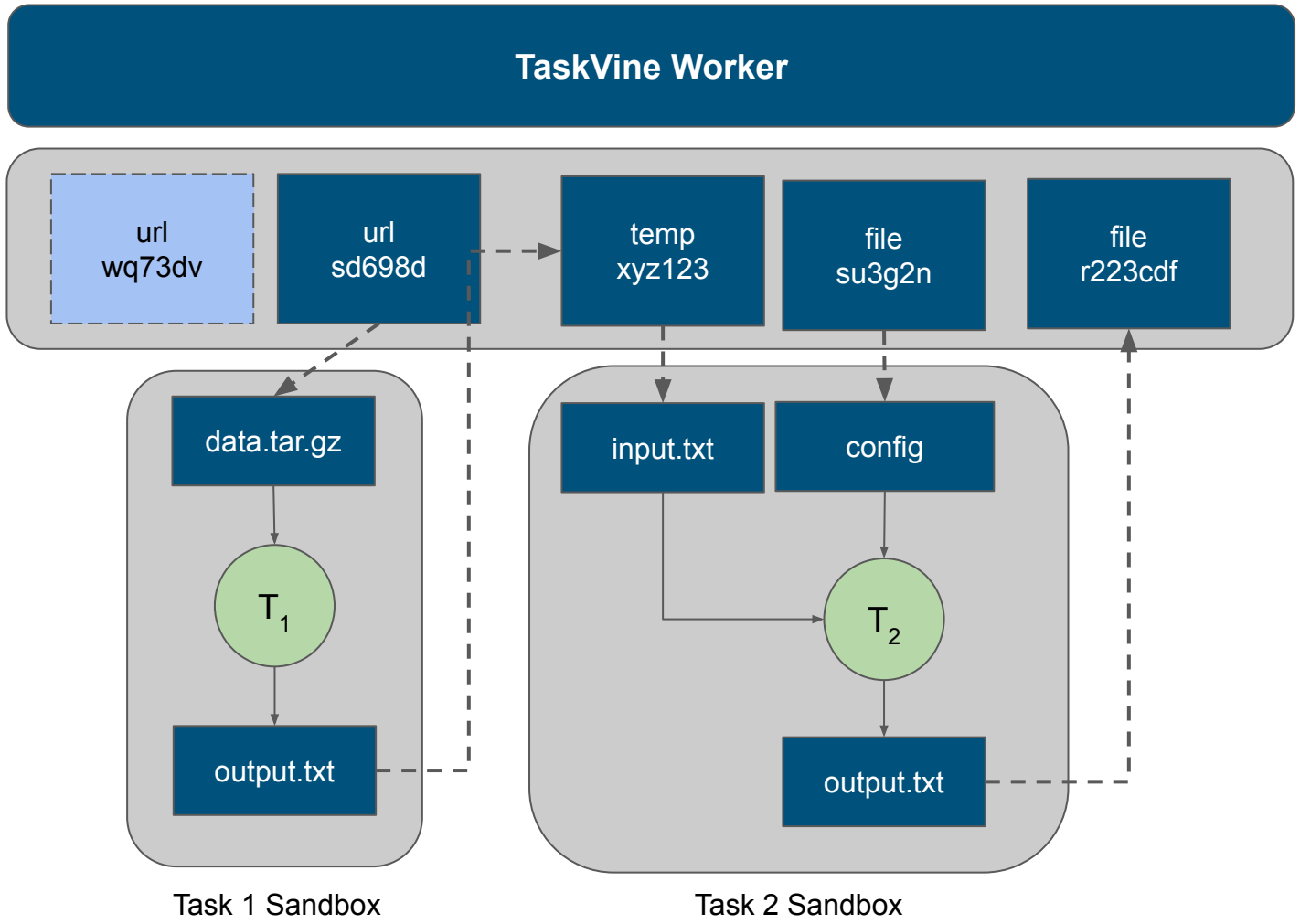




Environments can be staged and re-used among tasks

TaskVine Implementation

- Environment Management
- Storage Management and Naming
- Transfer Management
- Serverless Computing

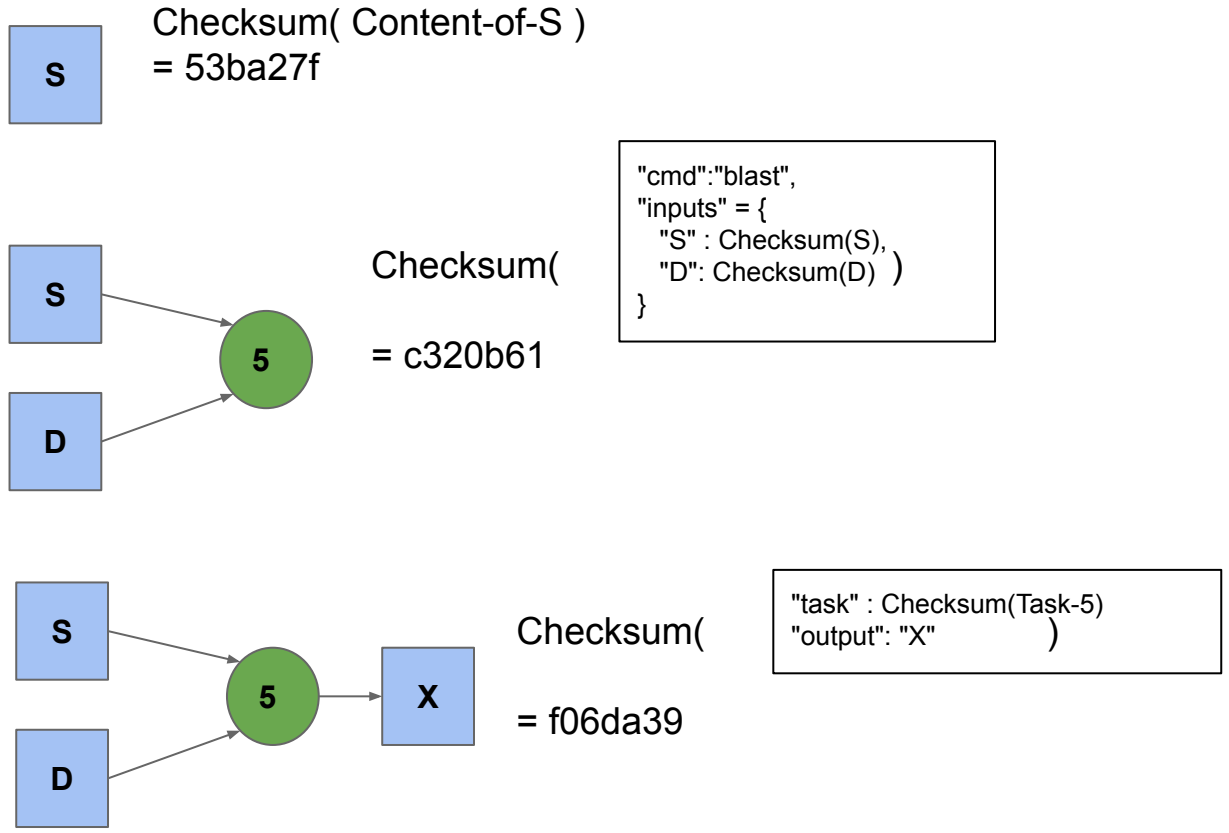


Consistent Naming

Files have one of three lifetimes:

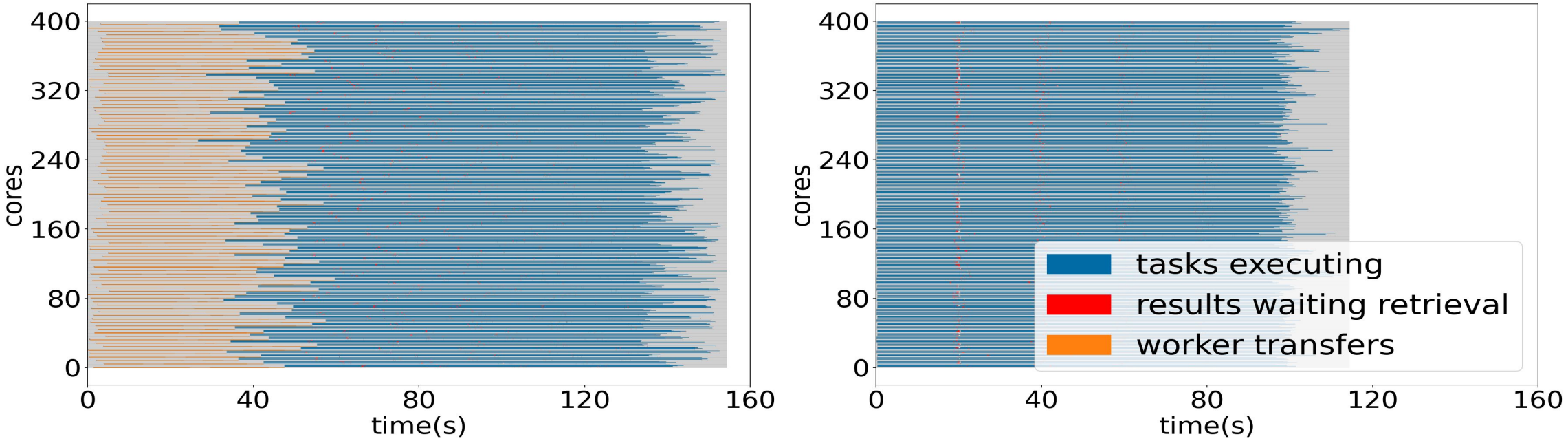
- single-task
- workflow (default)
- forever

"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.



Evaluation: Caching Through Consistent Naming

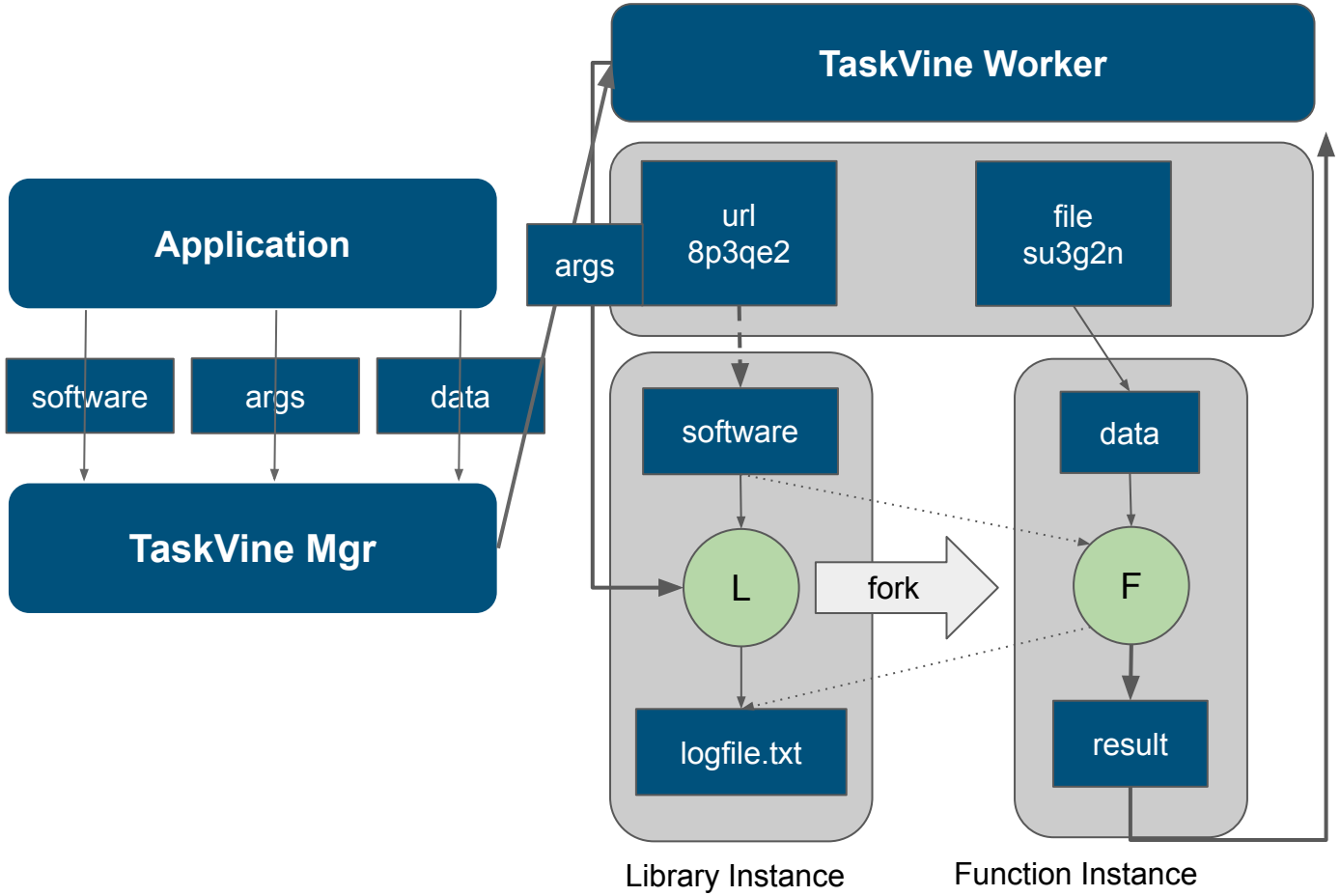
Experiment: Execution a workflow from a worker's perspective. During a cold start, there is substantial overhead due to transferring and staging data. This overhead is removed on subsequent runs.



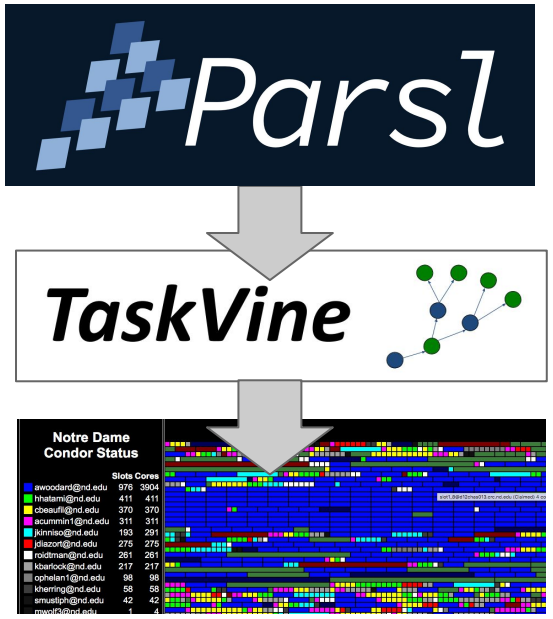
NOTE: consistent naming for various data types make caching possible across workflows!

Serverless Execution

```
# 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)

f = FunctionCall('my_library', 'my_mul', 2, 17)
m.submit(f)
```

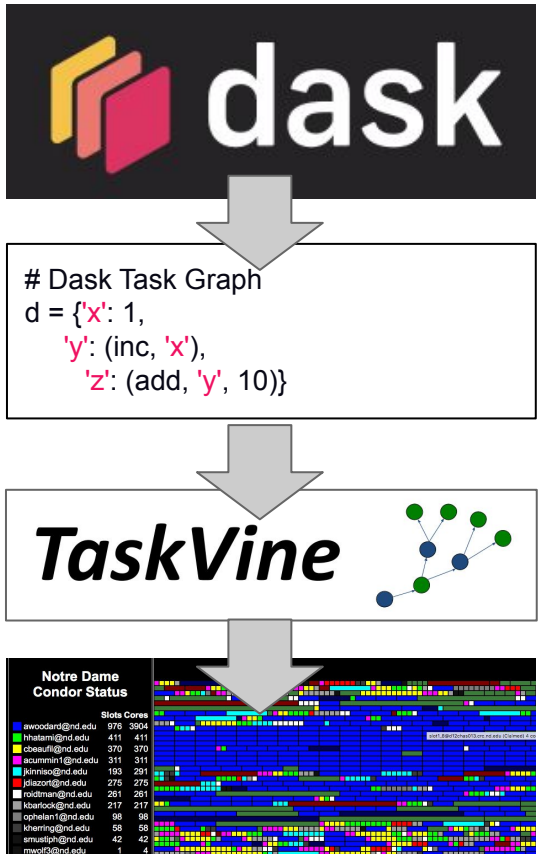


TaskVine Integration: Parsl




```python
import parsl
from parsl import python_app
from parsl.configs.vineex_local import config


parsl.load(config)

@python_app
def double(x):
    return x*2

future = double(1)
assert future.result() == 2
```

TaskVine Integration: Dask




```
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);
```

Thanks!

- **TaskVine** is a component of the Cooperative Computing Tools (cctools) from Notre Dame alongside Makeflow, Work Queue, Resource Monitor, etc.
- Latest release in October 2023.

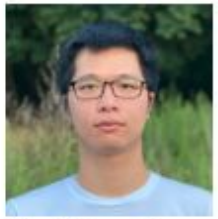
bslydelg@nd.edu

https://cctools.readthedocs.io

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

Douglas Thain
Director

Benjamin Tovar
Research Soft. Engineer



Thanh Son Phung
Ph.D. Student

Barry Sly Delgado
Ph.D. Student

Colin Thomas
Ph.D. Student

David Simonetti
Undergraduate

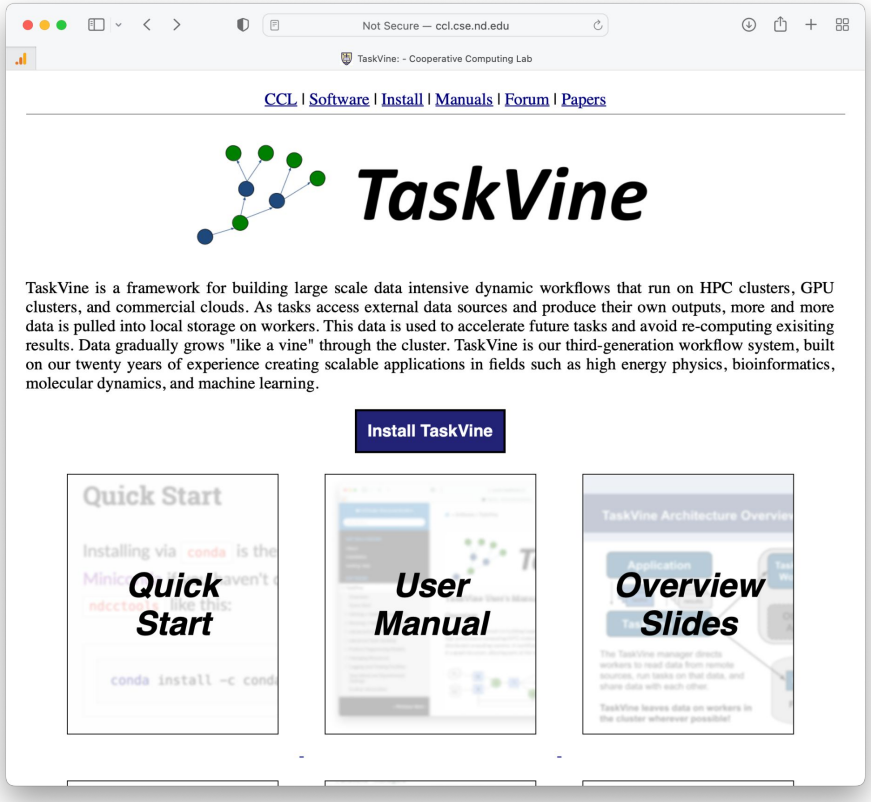
Joe Duggan
Undergraduate

Andrew Hennessee
Undergraduate

Matt Carbonaro
Undergraduate

Jachob Dolak
Undergraduate

This work was supported by NSF Award OAC-1931348



conda install -c conda-forge ndcctools