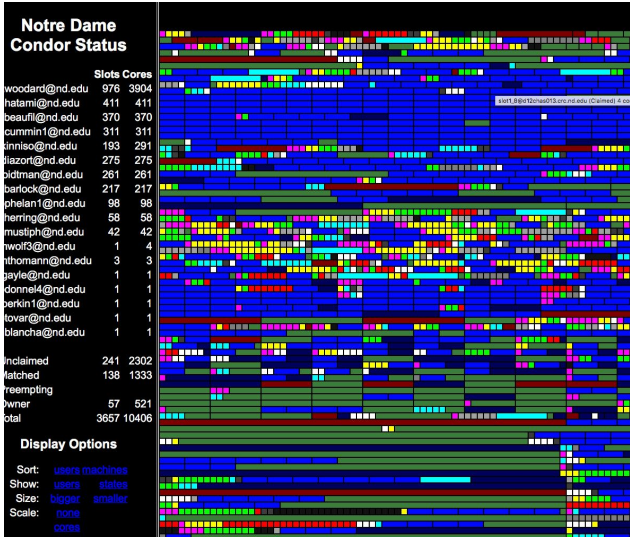
Executing Analysis Workflows at Scale with Coffea+Dask+TaskVine

Ben Tovar
NDCMS and Center for Research Computing and Cooperative Computing Lab
October 2023

UNIVERSITY OF NOTRE DAME

How to scale up computation to clusters?

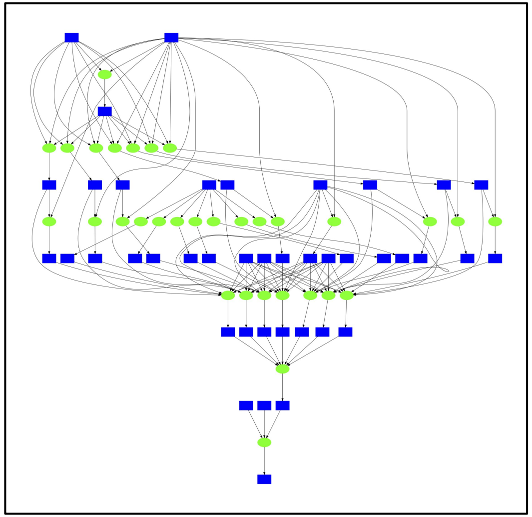
Computing Facility



UNIVERSITY OF NOTRE DAME

Workflows as a Computational Abstraction

A workflow is a collection of existing programs (functions) along with files (data objects) joined together into a large graph expressing dependencies. Allows for parallelism, distribution, and provenance without rewriting everything from scratch.

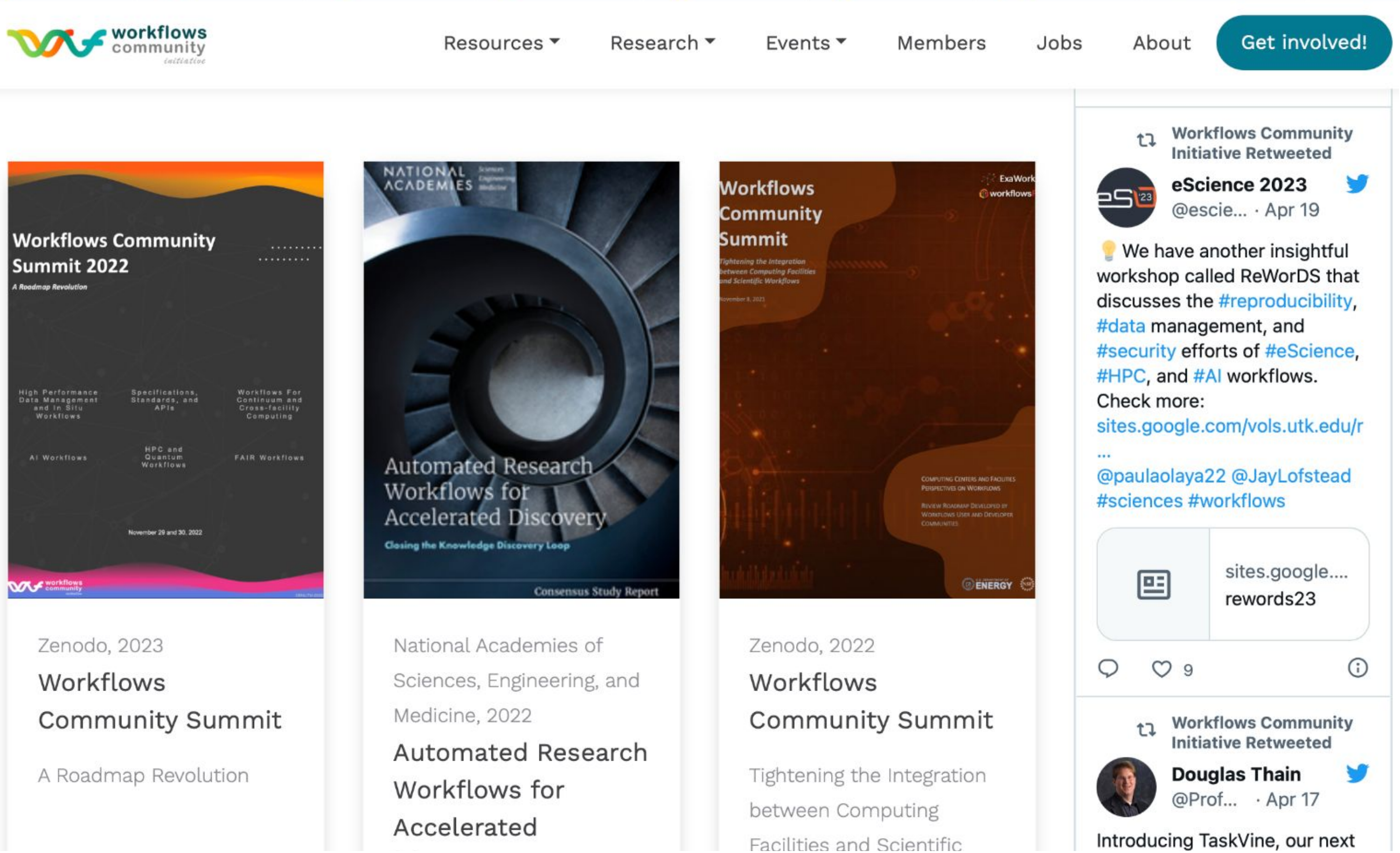


Research and Design Problems:

- Resource Allocation
- Scaling and Performance
- Data Management
- Reliability
- Portability
- Reproducibility

UNIVERSITY OF NOTRE DAME

http://workflows.community

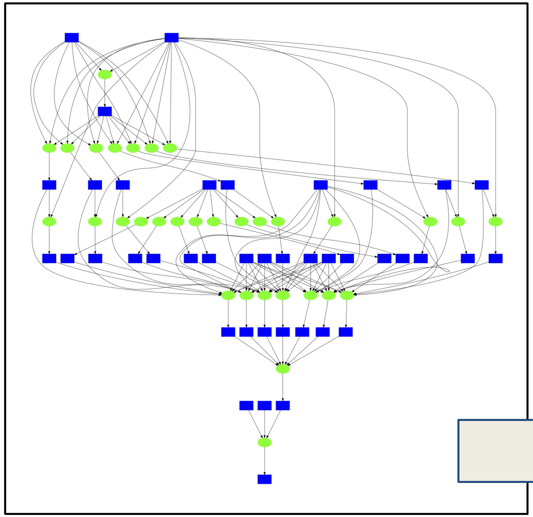


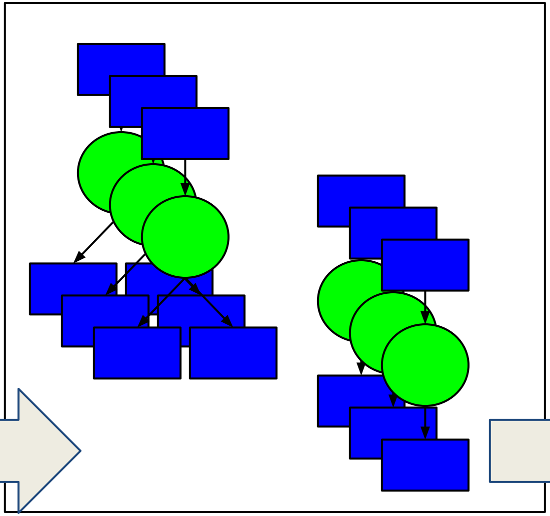
Workflow Management Systems

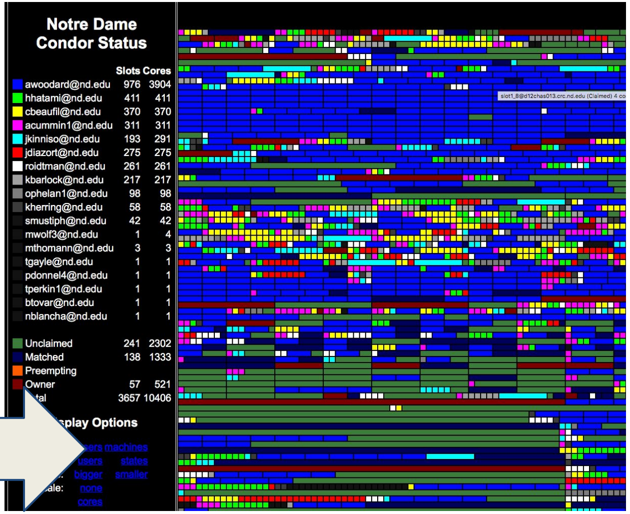
Workflow

Task / Data Scheduler

Computing Facility







Express overall workflow structure, components, constraints, and goals.

Assign ready tasks and data objects to resources in the cluster, subject to runtime constraints.

Execute tasks on computational resources, store and move data between nodes.

Challenges of Workflows on Clusters

- HPC filesystems are optimized for concurrent large-file access for message-passing jobs: bulk load, coordinated checkpoint, final write.
- But workflows tend to behave differently:
  - Traverse deep directory trees of small files. (metadata surge)
  - Access same input file from many nodes at once.
  - Create large intermediate files that are consumed and then deleted.
- Software is an essential part that is not usually integrated into the task dependencies:
  - huge startup times at scale due to metadata
  - Same packages get installed and loaded over and over again with small changes, sometimes intended, sometimes not.

UNIVERSITY OF NOTRE DAME

TaskVine

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

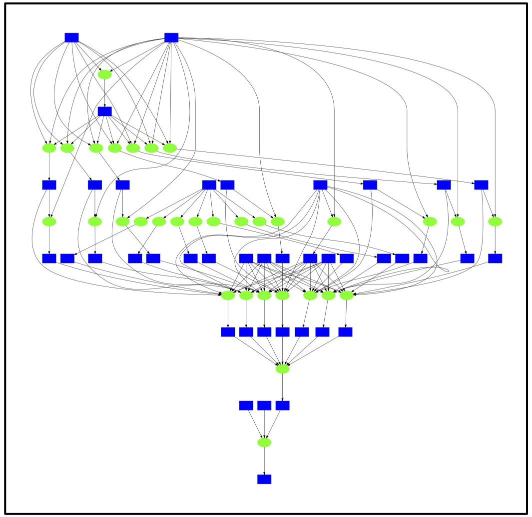
TaskVine controls the computation **and storage** capability of a large number of workers, striving to carefully manage, transfer, and re-use data and software wherever possible.

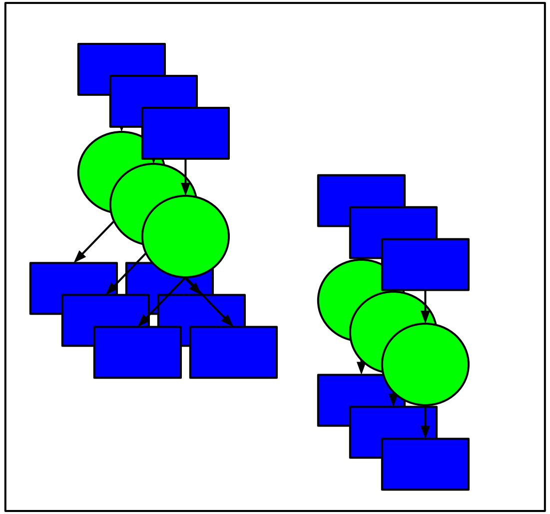
Key Idea: Exploit Storage in Cluster

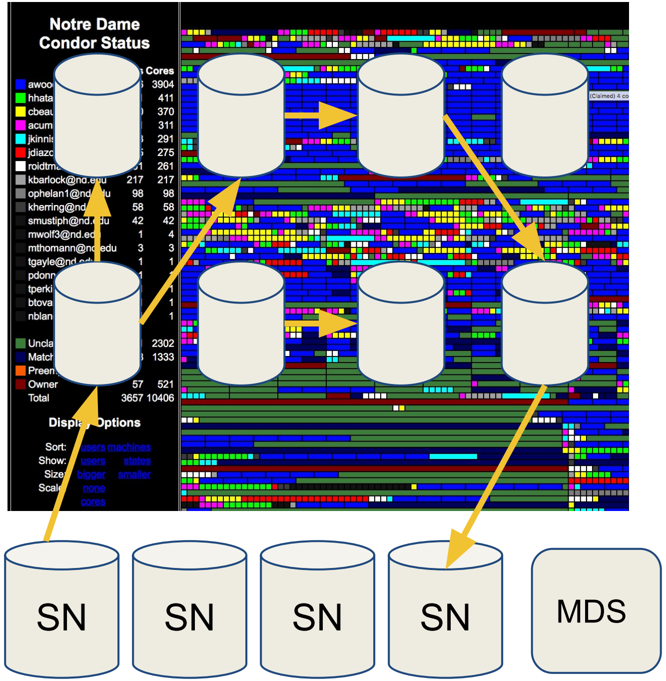
Workflow Creation (Dask)

Task / Data Manager (TaskVine)

Storage Already Embedded in Cluster

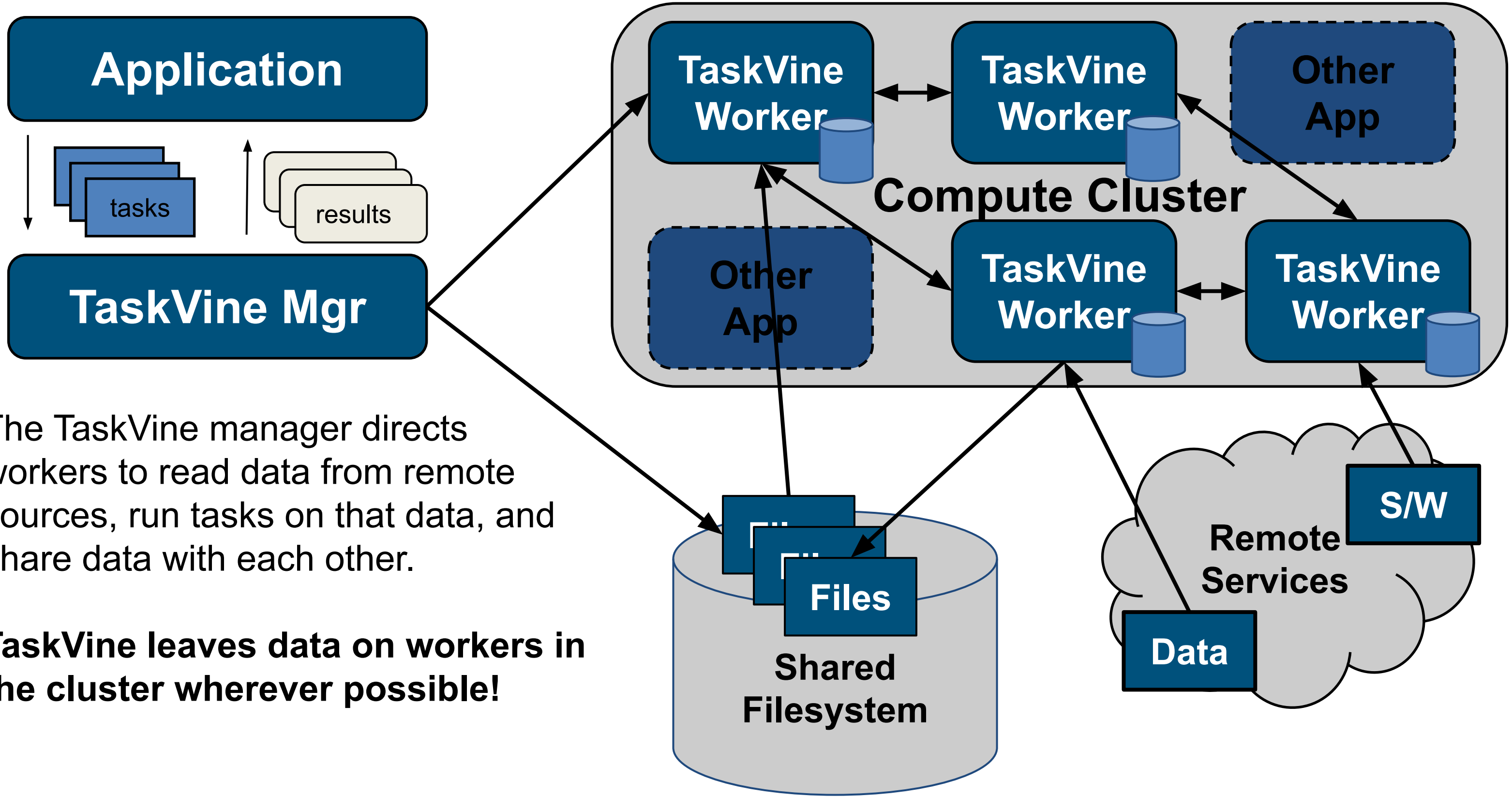






Shared Parallel Filesystem

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!

UNIVERSITY OF NOTRE DAME

Design Goals for TaskVine

Avoid moving data wherever possible: leave data in place until it needs to be moved or duplicated.

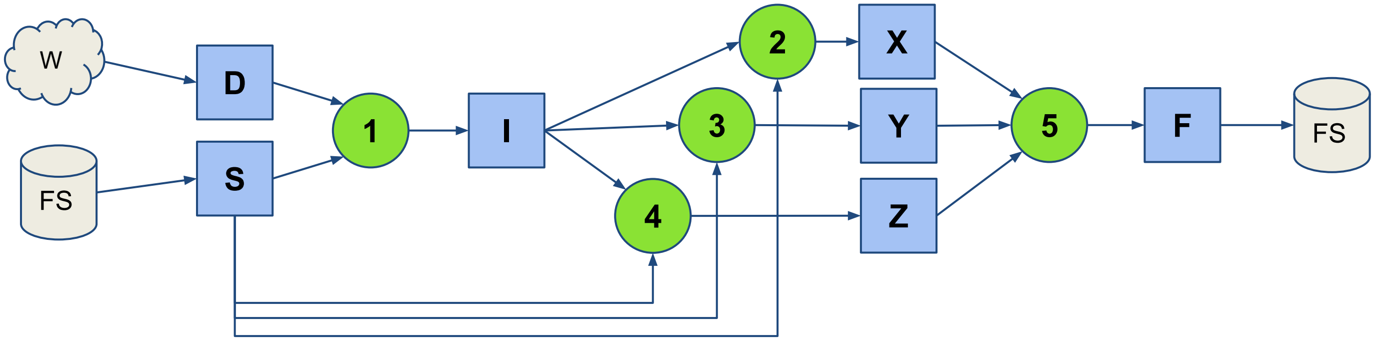
Manage task resources (cpu, gpu, mem, disk) carefully in order to pack in as much as we can (but not too much!) into each worker.

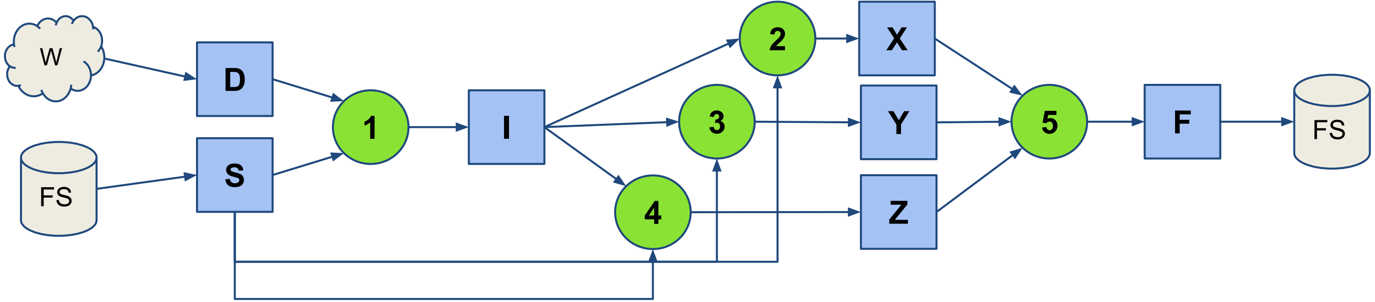
Make it easy to construct dynamic workflows with thousands to millions of tasks running on thousands of cluster nodes.

Handle common failures by detecting and recovering from worker crashes, network failures, and other unexpected events.

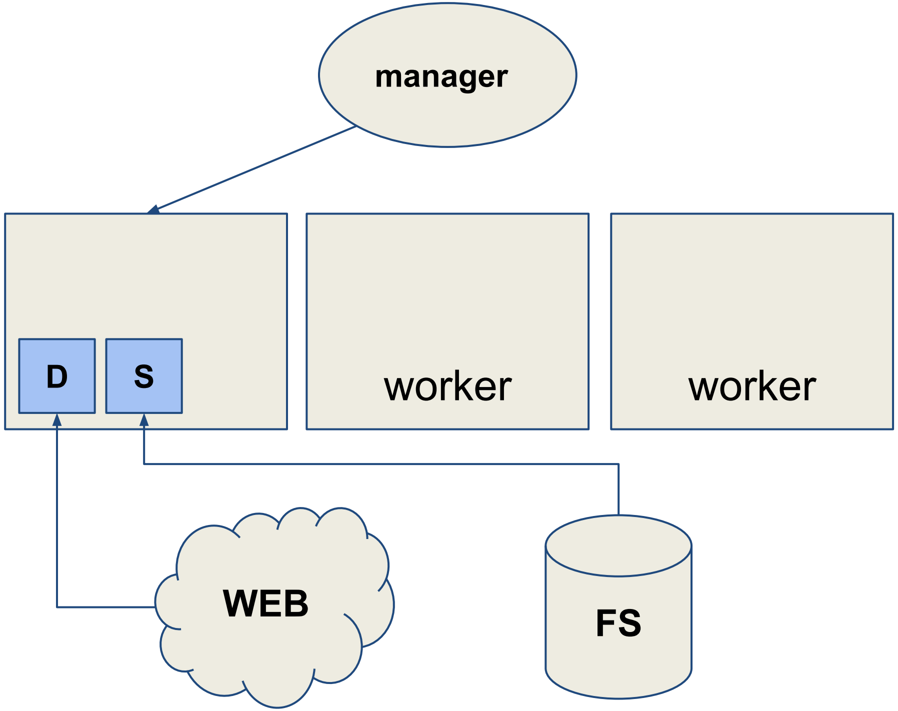
In-Cluster Data Management

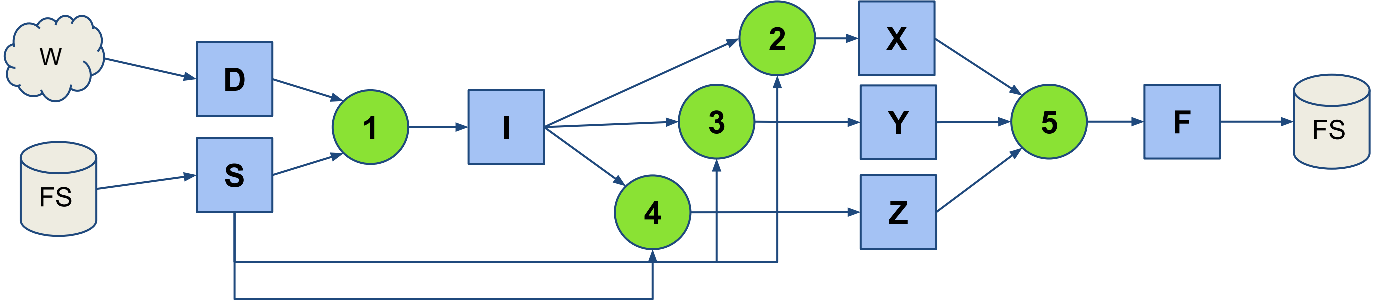
Suppose you have a workflow like this: a dataset D comes from a web repository, a software package S comes from the shared filesystem. After passing through tasks 1-5, the final output F should be written to the filesystem. TaskVine aims to keep all of the data within the cluster, as follows.



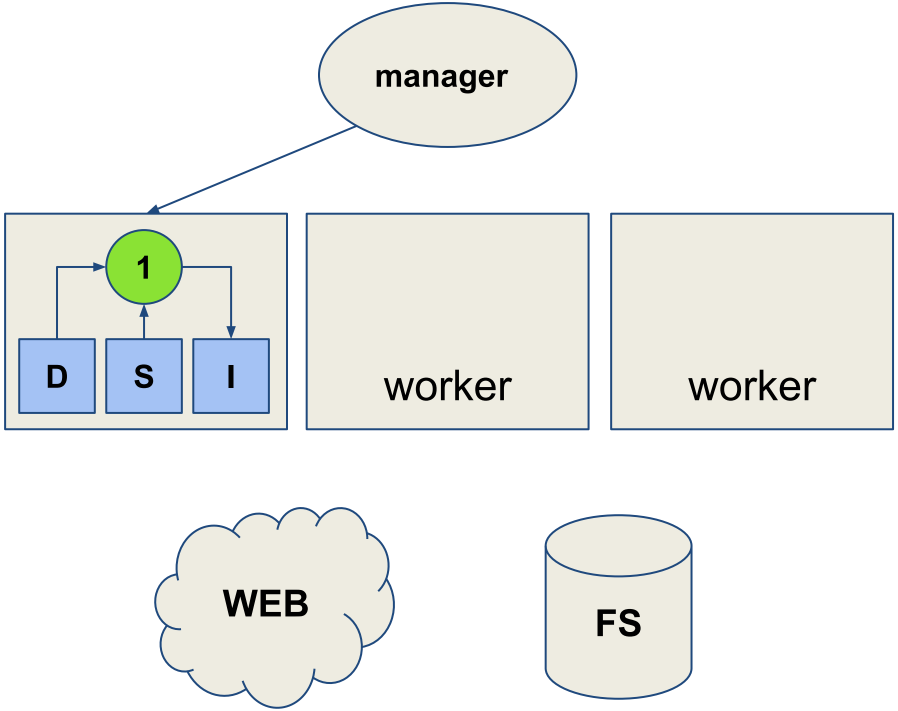


The manager selects a worker for task 1, and then directs dataset D to be downloaded from the web, and software package S to be loaded from the shared filesystem.

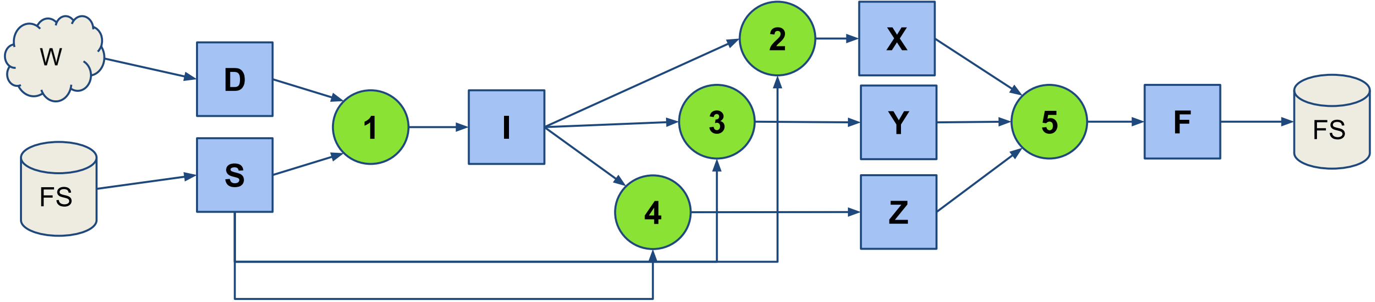




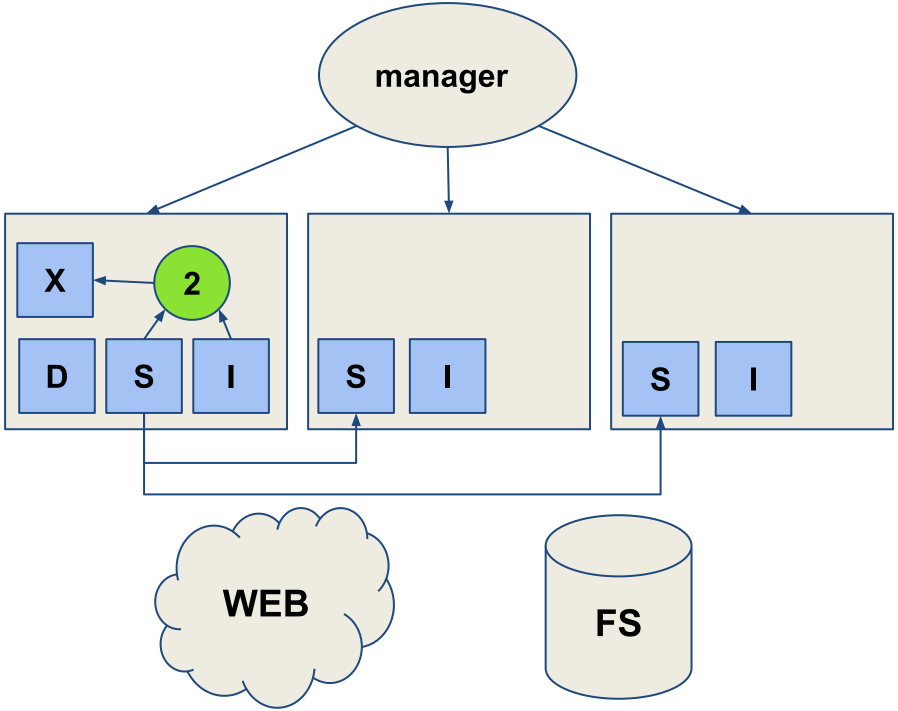
Next, task 1 is dispatched to that worker, where it reads dataset D, runs software package S, and produces file I, which stays where it is created.

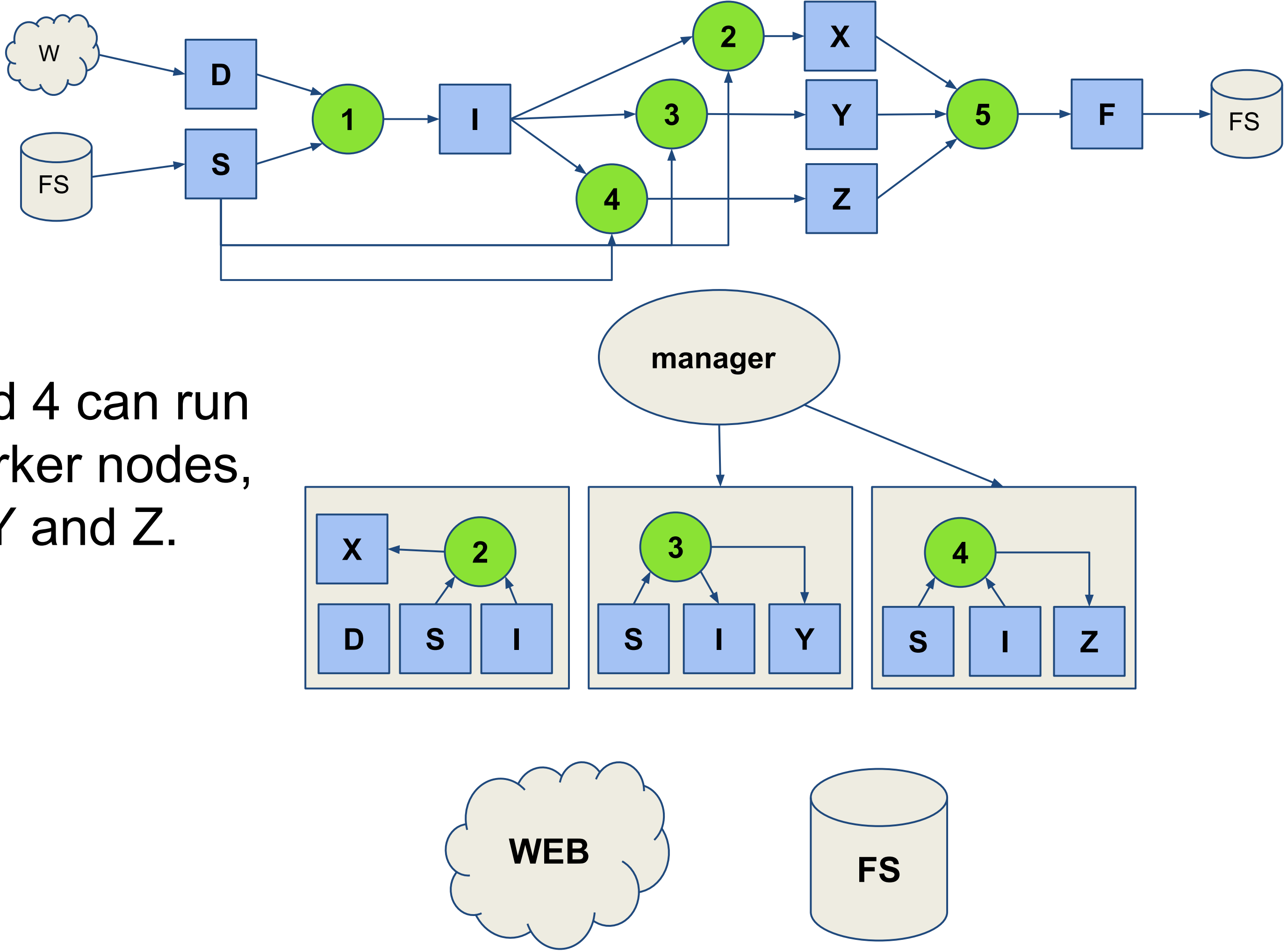


UNIVERSITY OF NOTRE DAME

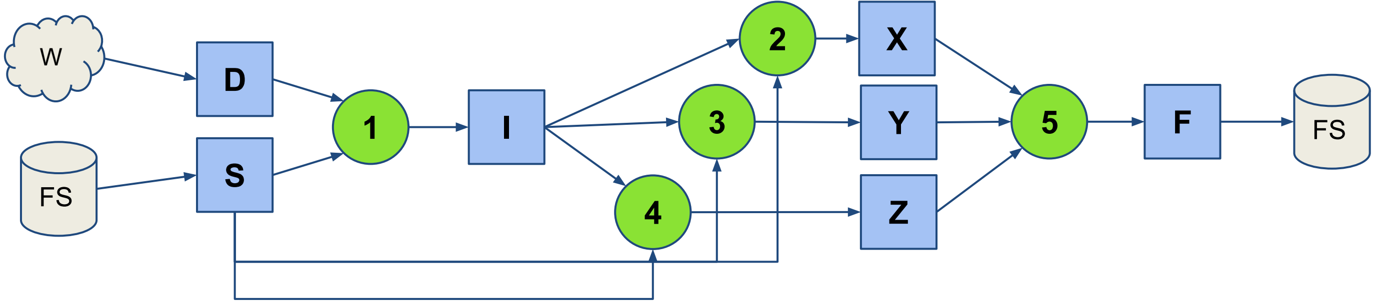


Once file I is created, task 2 can run immediately on that node, producing file X. Software package S and file I are duplicated to the other worker nodes.

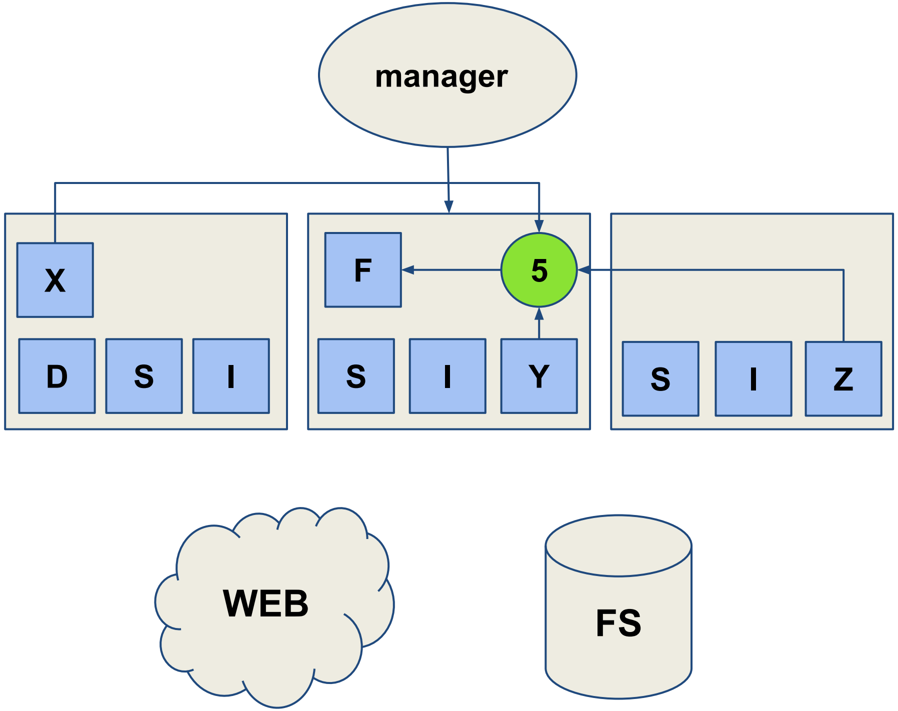


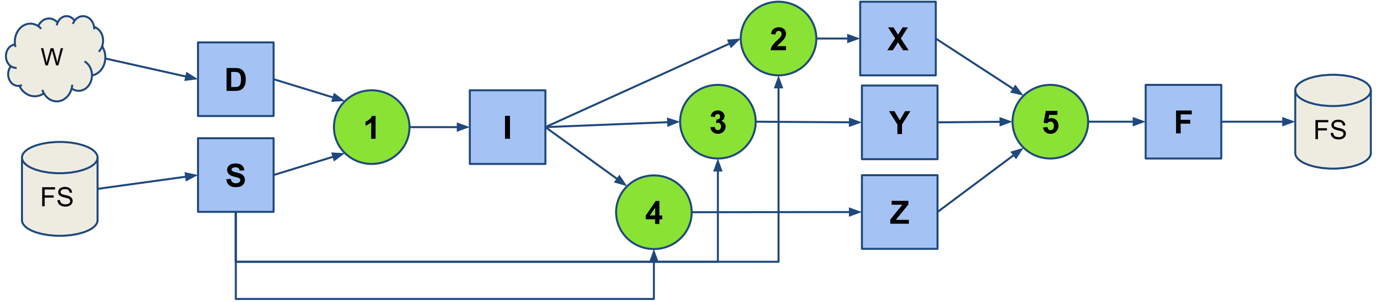


Now tasks 3 and 4 can run on the other worker nodes, producing files Y and Z.

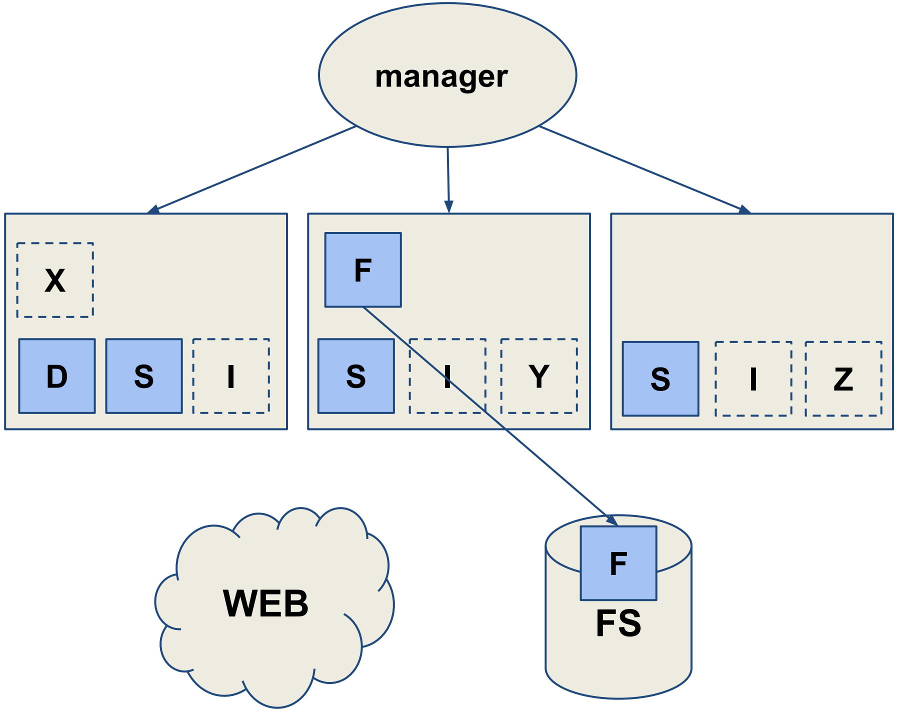


Next, task 5 is dispatched to the middle worker. It consumes files X, Y, and Z, which are pulled in from peer nodes. The output file X is produced on that node.





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

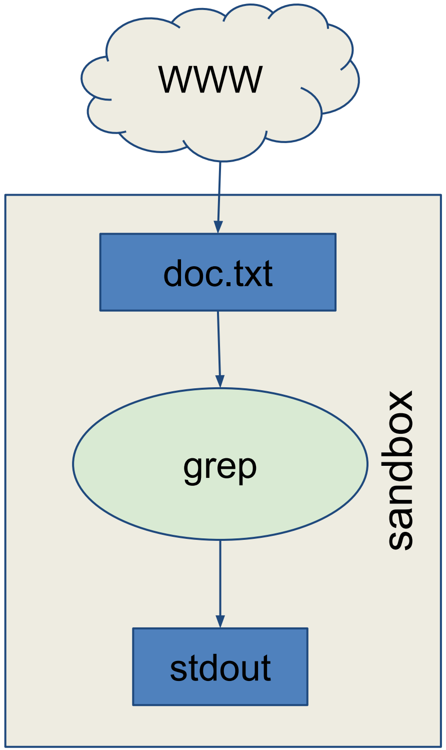


The manager directs the workers to delete any remaining uncacheable files.

Common input files remain to accelerate future workflows.

Defining a Simple Task

```python
import ndcctools.taskvine as vine

m = vine.Manager(9123)

doc = m.declare_url("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()

print(task.output)
```



API: Declare Files Explicitly

```python
import ndcctools.taskvine as vine

m = vine.Manager(9123)

file   = m.declare_file("mydata.txt")
buffer = m.declare_buffer("Some literal data")
url    = m.declare_url("https://somewhere.edu/data.tar.gz")
temp   = m.declare_temp();
```

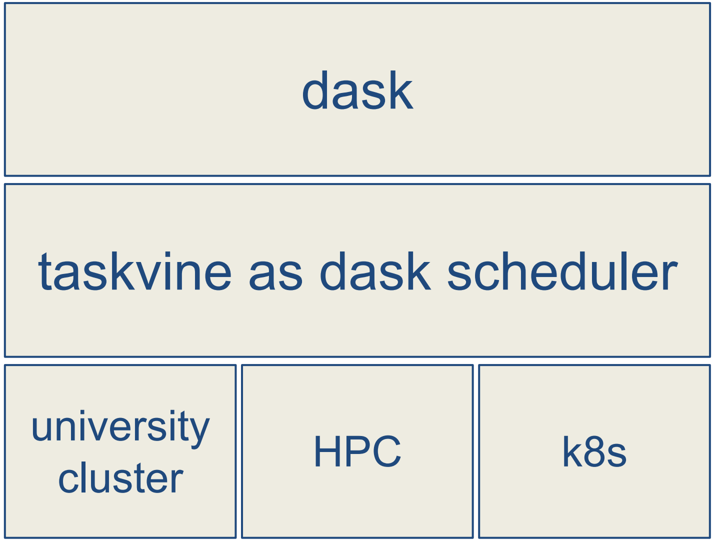
UNIVERSITY OF NOTRE DAME

API: Submit Python Functions

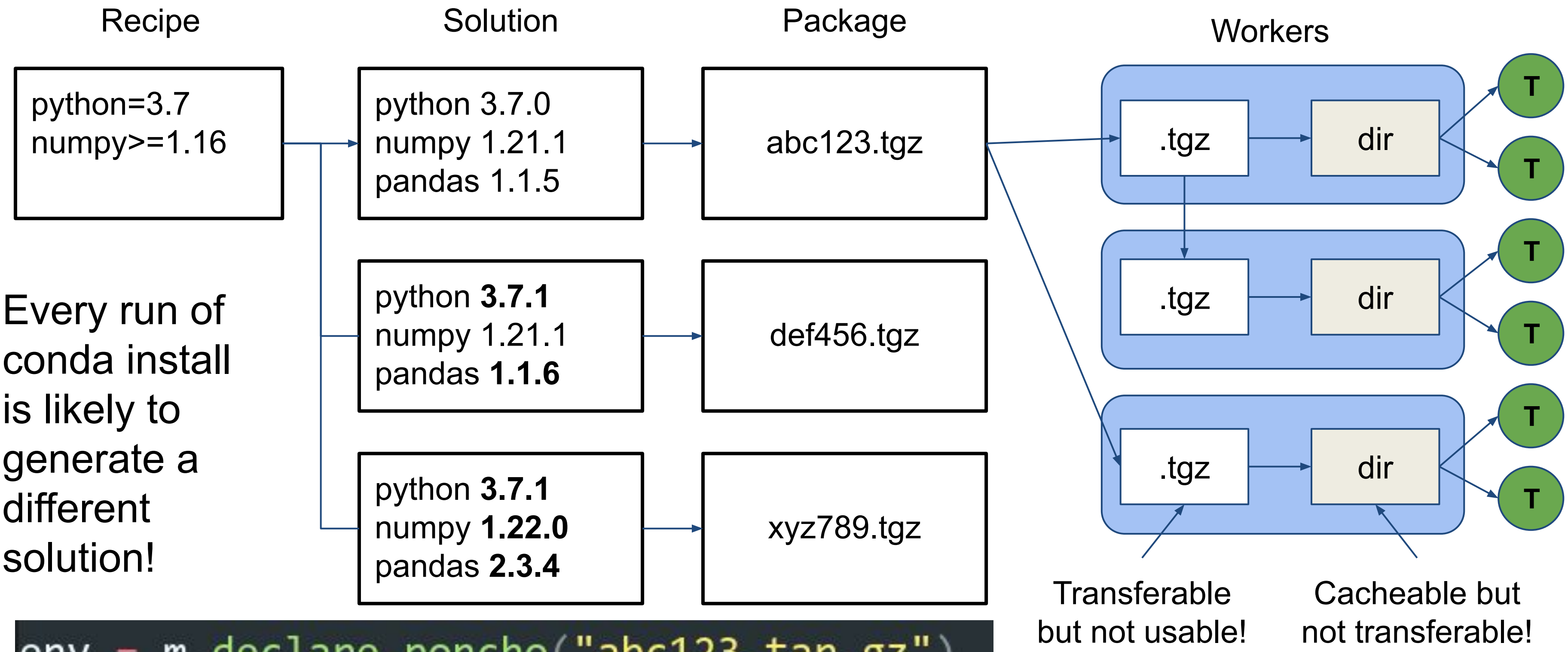
```
t = vine.PythonTask(some_function, event, parameters)

url     = m.declare_url("https://somewhere.edu/data.tar.gz")
temp    = m.declare_temp();

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_category("processing")

taskid = m.submit(t)
```

Computing for Dask Workflows

```python
import ndcctools.taskvine as vine

m = vine.DaskVine(9123)

...

z.compute(scheduler=m.get)
```



Sharing Software Environments



Every run of conda install is likely to generate a different solution!

```
env = m.declare_poncho("abc123.tar.gz")
t.add_environment(env)
```

Analysis: TopEFT Framework

- Use TopEFT analysis to test current framework
- Full Run 2 analysis (~150/fb, HL-LHC~3000/fb)
- Designed to analyze CMS data in order to search for new physics using the framework of Effective Field Theory (EFT) CMS-PAS-22-006
- Built on Coffea framework with columnar approach relying on scientific python ecosystem

https://github.com/TopEFT/topcoffea
https://github.com/TopEFT/topeft

TopEFT overview

- The TopEFT workflow:
  - Inputs are flat n-tuple (CMS NanoAOD) formatted proton-proton collision data from CMS (~2TB)
  - Processing step consists of calculating relevant properties of the events and filling histograms
  - Accumulation function merges together the histograms to produce the final output
- Memory considerations of the histograms produced and accumulated with TopEFT:
  - TopEFT histograms are heavier than conventional histograms
  - Each bin carries an array of 378 numbers for its EFT framework
  - The accumulation step can cause large memory requirements

Previous TopEFT performance at ND Tier-3

(Work Queue, coffea 0.7.x, old coffea-hists)

cpu needs

runtime:	100min
cores:	up to 1000

IO needs

total root data:	1.7 TB
origin:	xrootd local

processing tasks

avg time:	110s
slowest:	318s
largest mem	4 GB
largest disk	0.5 GB

accumulation tasks (20 to 1)

avg time:	6s
slowest:	**141s**
largest mem:	**12 GB**
largest disk:	**20 GB**

Toward Ten-Minute Turnaround in CMS Data Analysis: the View from Notre Dame, CHEP 2023
https://cds.cern.ch/record/2872899

UNIVERSITY OF NOTRE DAME

TopEFT performance today at ND Tier-3

(TaskVine, coffea 0.7.x, new scikit HEP histograms)

cpu needs

runtime:	**30 min**
cores:	up to 4500

IO needs

total root data:	1.7 TB
origin:	vast

processing tasks

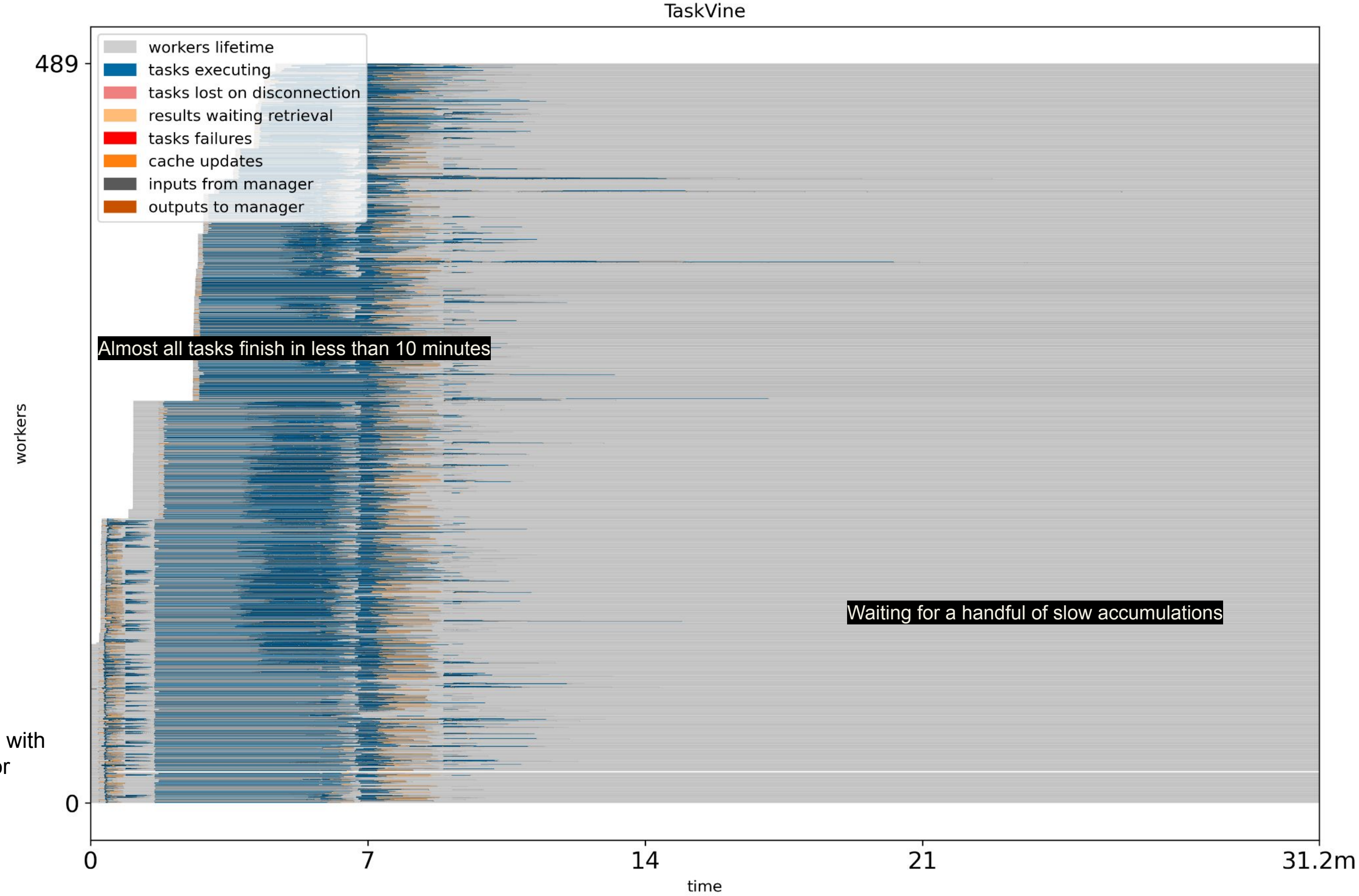
avg time:	105s
slowest:	460s
largest mem	6 GB
largest disk	0.5 GB

accumulation tasks (5 to 1)

avg time:	6s
slowest:	**440s**
largest mem:	**35 GB**
largest disk:	**8 GB**

UNIVERSITY OF NOTRE DAME

Full R2 run



12 core workers

managed with HTCondor

taskvine as an old coffea-style executor

using scikit-hep histograms

UNIVERSITY OF NOTRE DAME

Current Bottleneck: Histograms

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       10    0.000    0.000  101.450   10.145
/var/condor/execute/dir_736567/worker-196886-736585/task.14889/__vine_env_task-rnd-
wjargmxyiltudhp/lib/python3.10/site-packages/coffea/processor/executor.py:191(_deco
mpress)
       10    7.888    0.789  101.435   10.144 {built-in method _pickle.load}
```

topEFT histograms are pretty sparse (only about 15% of entries filled).

Needed to adapt scikit-hep hist to sparse histograms, otherwise we ran out of memory (more than 128GB for some accumulation tasks)

Pickling by writing histogram counts with scipy sparse matrices (saves ¾ disk, ½ of memory)

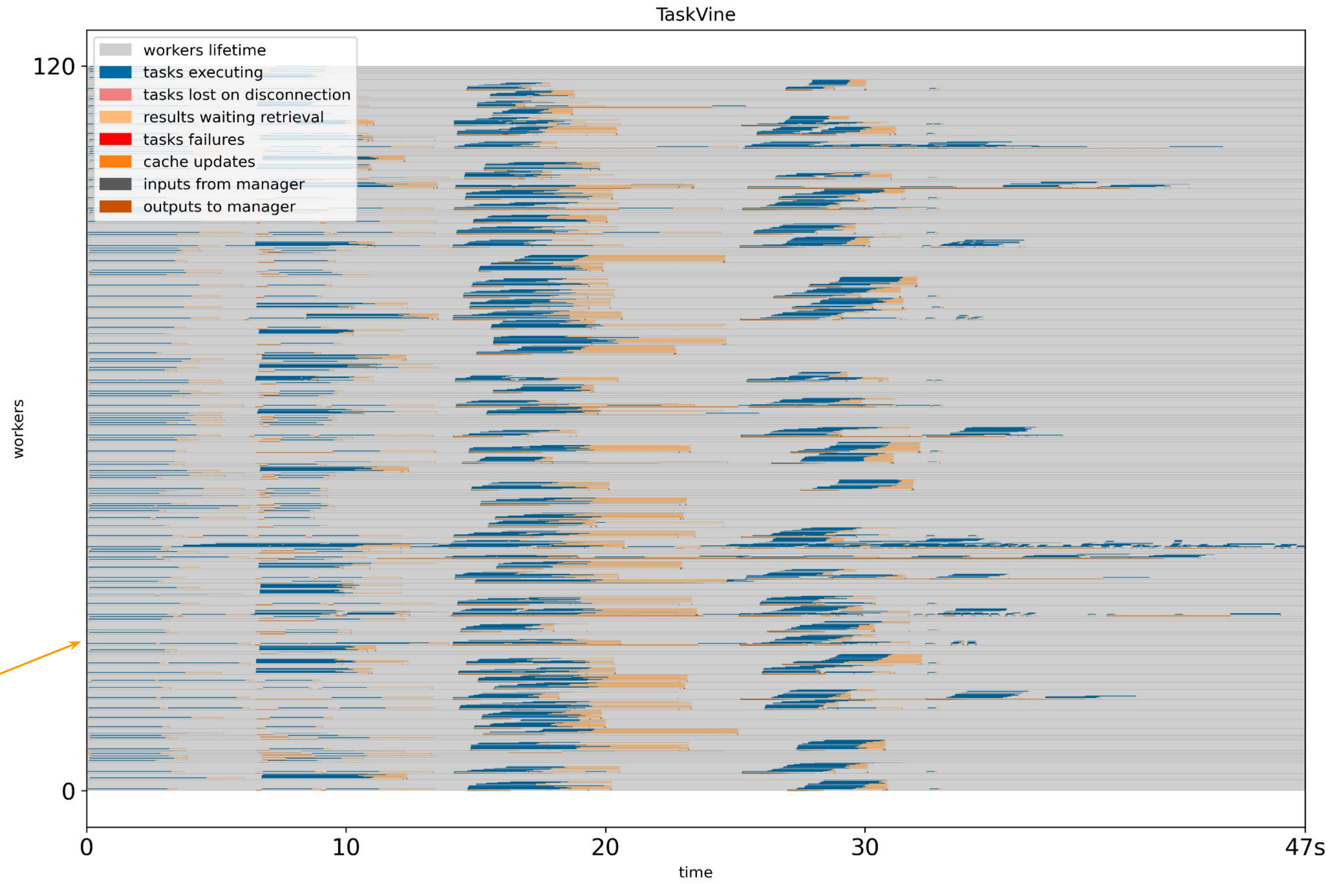
Currently working on adding sparse histograms to scikit-hep hist.

UNIVERSITY OF NOTRE DAME

TaskVine + Coffea + Dask

```python
m = DaskVine(name="my-vine-manager")

# The opendata files are non-standard NanoAOD, so some optional data columns are missing
processor.NanoAODSchema.warn_missing_crossrefs = False

events = NanoEventsFactory.from_root(
    "file:/project01/ndcms/btovar/Run2012B_SingleMu.root",
    treepath="Events",
    chunks_per_file=288,
    permit_dask=True,
    metadata={"dataset": "SingleMu"},
).events()

q1_hist = (
    hda.Hist.new.Reg(100, 0, 200, name="met", label="$E_{T}^{miss}$ [GeV]")
    .Double()
    .fill(events.MET.pt)
)

h = q1_hist.compute(
    scheduler=m.get,
    resources_mode="min waste",
    lazy_transfers=True,
    environment="my-env.tar.gz",
)

dak.necessary_columns(q1_hist)

h.plot1d()
```

TaskVine + Coffea + Dask (q6 coffea benchmark)



TaskVine chooses resources (like cores)

UNIVERSITY OF NOTRE DAME

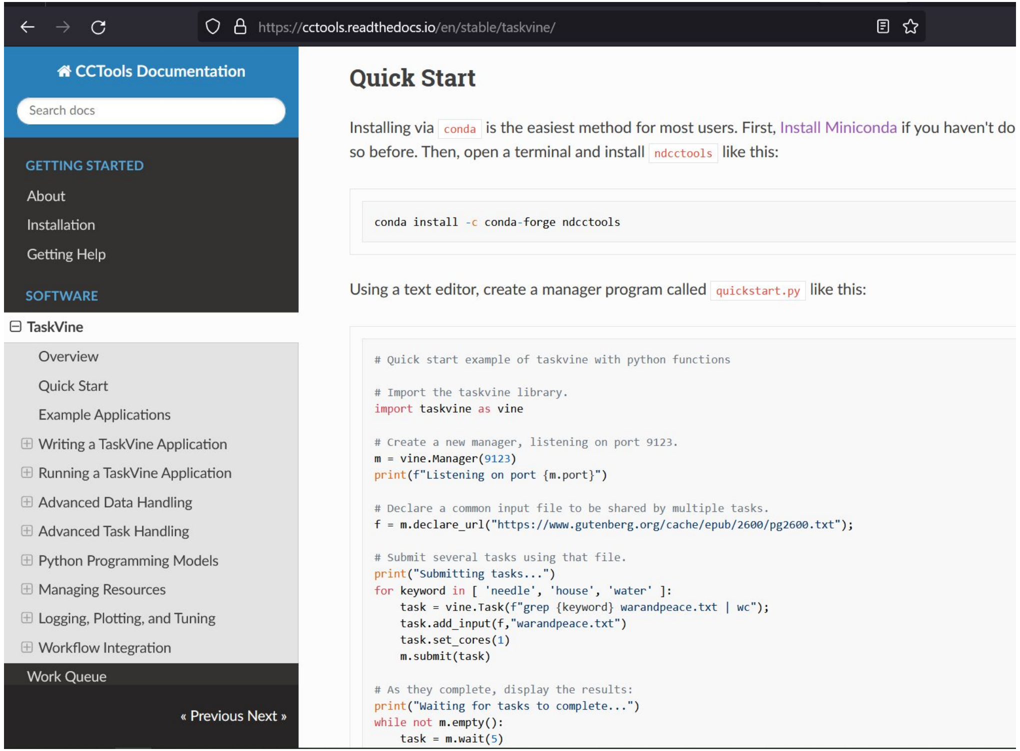
TaskVine Other Features

- Option for cached files at workers to survive workflows executions.
- Python "serverless": install libraries at workers and don't pay interpreter initialization overhead.
- Define custom file types and environments with mini-tasks.
- Measurement and automatic allocation of resources.

UNIVERSITY OF NOTRE DAME

Documentation

https://cctools.readthedocs.io/en/stable/taskvine/



https://cctools.readthedocs.io
https://github.com/cooperative-computing-lab/cctools
`conda install -c conda-forge ndcctools`

Thanks to team and collaborators

ND CMS	CCL
Prof. Kevin Lannon	Prof. Douglas Thain
Prof. Mike Hildreth	Thanh Son Phung
Kelci Mohrman	Barry Sly-Delgado
Brent R. Yates	Colin Thomas
Andrew Wightman	Md Saiful Islam
John Lawrence	Jin Zhou
Andrea Trapote	David Simonetti
Irena Johnson	Andrew Hennessee
Kenyi Hurtado	Jachob Dolak

This work was supported by NSF Award OAC-1931348