USCMS ANALYSIS FACILITIES MEETING

The vine-cms-analysis-stack: Use Case at Notre Dame

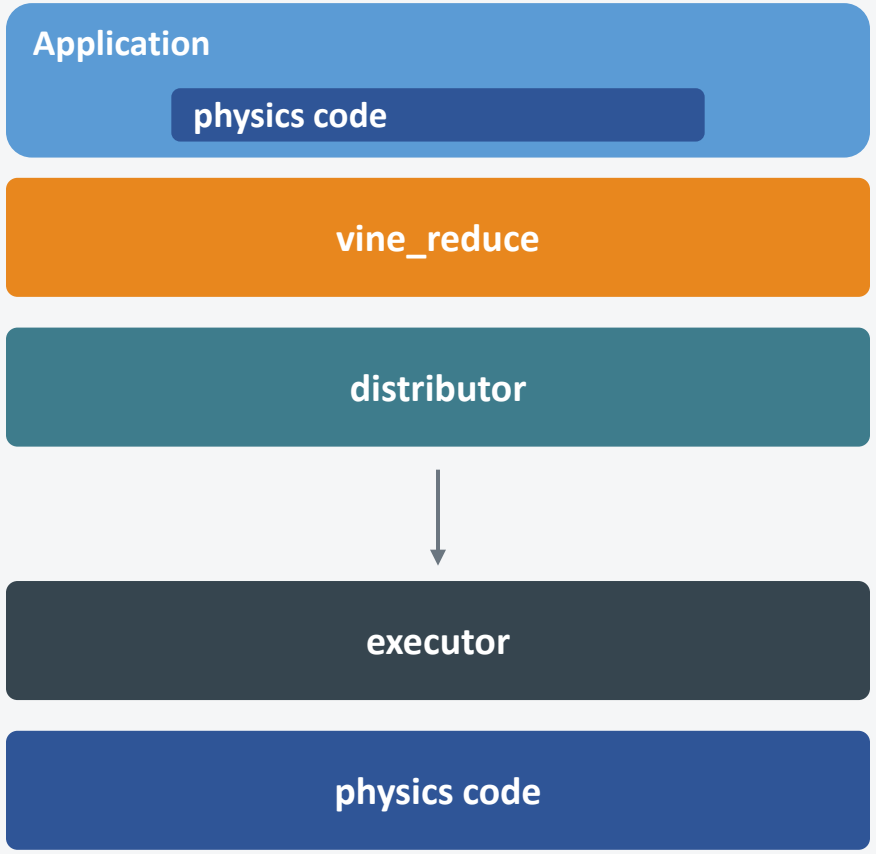
Coffea + TaskVine + vine_reduce

Benjamin Tovar btovar@nd.edu Cooperative Computing Lab • University of Notre Dame

github.com/cooperative-computing-lab/vine-cms-analysis-stack

github.com/cooperative-computing-lab/vine_reduce

THE VINE CMS ANALYSIS STACK



local × 1

runs once, front-end process

how the work is split into chunks and accumulated back together

where each call runs

remote × N

once per chunk, on a worker
perhaps many per worker

how one call runs on the machine that got it

what to compute invoked by the executor

Each is chosen on its own, and none needs to know what the others contain.

Framework

Dynamic Data Reduction

A MapReduce in with commutative reductions. Chunks are processed independently, and their partial results are accumulated in whatever order and grouping the cluster happens to have available. Make two concepts separate:

Execution

What happens on one chunk: read the events, run the processing function, produce a partial result.

- One chunk, one call without a view of the cluster.
- Any local tool: a plain function, Coffea, Dask, RDataFrame.
- Nothing above it needs to know which one was picked other than dispatch.

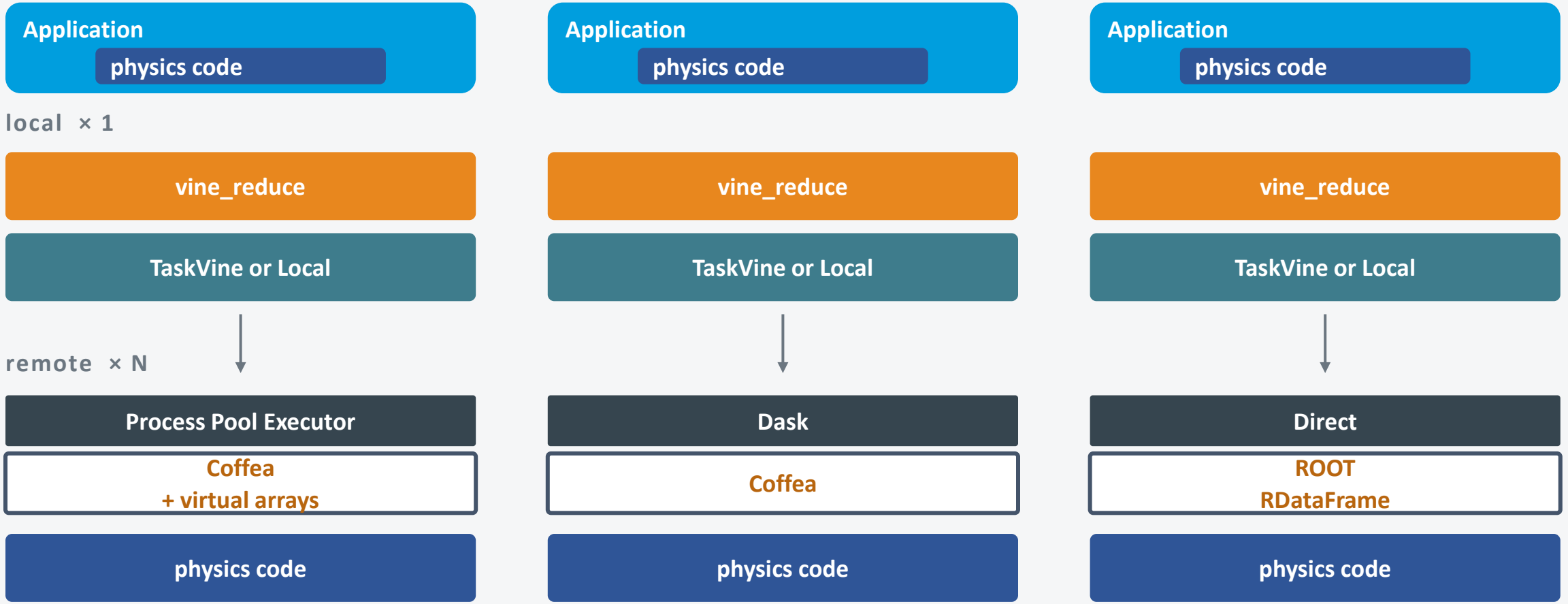
Distribution

How work spreads across the cluster: which chunks run where, in what order, at what size, what happens on failure.

- A cluster and workflow-shape view.
- Owns priorities, resource exhaustion, checkpointing, restart.
- Can exploit associativity, commutativity, distributivity, orthogonality of common analysis workflows.

vine_reduce keeps them apart — which is why one distribution layer serves every analysis we run at ND unchanged, whatever each of them runs on a chunk. Neither vine_reduce, its executors nor its distributor know any physics.

THE VINE CMS ANALYSIS STACK



what we run at ND

no Coffea in the stack at all

Only the outlined slot differs. It decides how the physics code beneath it is invoked; the distribution logic above it is identical in all three. The distributor slot swaps the same way — LocalDistributor on laptop cores, TaskVineDistributor on the cluster, nothing else edited.

How work is distributed

1 No global task graph

Processor and reducer tasks are submitted opportunistically as work becomes ready, throttled by what the cluster can currently take. No need to wait for the graph to be built. (xN mini graphs)

→ *before: nothing ran until everything was planned*

2 Size adapts at runtime

A task that reports resource exhaustion halves the chunk or reduction size for that (processor, dataset).

→ *before: one bad chunk size would kill the workflow*

3 Checkpoint at partial reductions

Crossing an event count, execution-time or size in memory threshold creates a checkpoint and keeps a record on a db. Progress is persistent: a restart resumes from the last checkpoints, not from the beginning.

→ *before: a crash cost the whole run*

4 Per-dataset priority and isolation

Datasets are orthogonal, so they never cross-reduce; earlier-listed datasets are scheduled first and finish sooner. A failure is scoped to one chunk* of one dataset on one processor. (*And its file.)

→ *before: long tails affected all datasets*

How work is distributed

Why reordering the work is safe

Reductions are **associative** $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ and **commutative** $a \oplus b = b \oplus a$, and **distributive** $f([a,b]) = f([a]) \oplus f([b])$

Stating it explicitly is what lets the distribution layer to dynamically manipulate order, grouping, and size as the run progresses.

Opportunistic scheduling is correct

Reducing chunks in whatever order and grouping the cluster makes available gives the same answer.

A partial reduction is a valid checkpoint

A partially-reduced result is an exact intermediate value of the accumulation, not an approximation.

Chunk size is a free variable

Modifying the chunk mid-run is a resource decision unrelated to the physics computation.

HOW IT IS PUT TOGETHER

vine_reduce architecture: one manager, n workers

LOCAL — the vine_reduce process (×1)

`input_to_datasets(json) → datasets_to_chunks() | is_checkpoint() → is_result()`

chunk generation throttled by distributor.capacity(); It is possible to have many results per (processor, dataset)

submit(priority, category, kind, func) ↓

↑ *wait() → Outcome(resources, ...)*

REMOTE — a worker node (×N), one instance per submitted call

`kind = "processor"`

`executor_wrapper(chunk, ...)`

→ `chunk_to_args(chunk) → events`

→ `executor(processor, events)`

→ `Outcome{file | traceback}`

`kind = "reducer"`

`reducer_wrapper(reducer, results, is_result)`

→ `reducer(a, b) → merged`

→ `if is_result(n, mem, time): result_postprocess()`

→ `Outcome{file | traceback}`

Distribution interface

Interface vine_reduce needs from a distributor

```
class Distributor():
    def submit(self, priority, category, kind, priority, checkpoint_p, func, *args) -> uuid:
        ...  # kind is "processor" or "reducer"
        ...  # category is a label per processor, dataset, kind
    def wait(self, timeout=None) -> Outcome | None:
        ...  # blocks for one finished call's Outcome, or None on timeout
    def capacity(self) -> int:
        ...  # how many more chunks + reducers this distributor could take right now

    def release(self, result_id) -> None: ...              # free resources (checkpointed or canceled)
    def retrieve(self, result_id, dest_path) -> None: ... # copy result to dest_path
    def add_file(self, local_path) -> None: ...            # ship this file with every task
    def set_env_var(self, name, value) -> None: ...        # set this env var for every task
```

What a user writes

From examples/trijet (ADL benchmark Q6) — no vine_reduce, TaskVine, or distributor code appears in the processor.

```python
def trijet_processor(events):
    jets = ak.zip({k: getattr(events.Jet, k)
        for k in ["x","y","z","t","btag"]},
        with_name="LorentzVector")
    trijet = ak.combinations(jets, 3,
        fields=["j1","j2","j3"])
    trijet["p4"] = (trijet.j1 + trijet.j2
        + trijet.j3)
    dm = abs(trijet.p4.mass - 172.5)
    best = ak.argmin(dm, axis=1)
    trijet = ak.flatten(
        trijet[ak.singletons(best)])
    return {"trijetpt": hist.Hist.new
        .Reg(100, 0, 200, name="pt3j")
        .Double().fill(trijet.p4.pt)}
```

```python
distributor = TaskVineDistributor(
    name="btovar-vine",
    resources_processor={"cores": 2}, # max
    resources_reducer={"memory_mb": 4000},
    checkpoints_dir="checkpoints",
)
vr = VineReduceCoffea(
    processors={
        "trijet": trijet_processor,
    },
    input=datasets,  # coffea preprocess()
    distributor=distributor,
    results_dir="results",
    environment=get_environment()  # conda
)
vr.compute()
```

WHAT WE RUN AT ND

Three analyses on the stack today

ttbarEFT

Hannah Nelson's thesis

50 × 4-core

~15 min

turnaround

TopEFT/ttbarEFT: histogram-filling processors per lepton channel, with Wilson-coefficient

MDV5

Connor Moore skimming on a dozen criteria

450 x 4-core

51 datasets

~3 hrs

turnaround

Large skim over many orthogonal datasets —per-dataset independence and checkpointing matter a lot, output is parquet files.

Cortado

Kelci Mohrman general skimmer

200 x 8-core

419 datasets

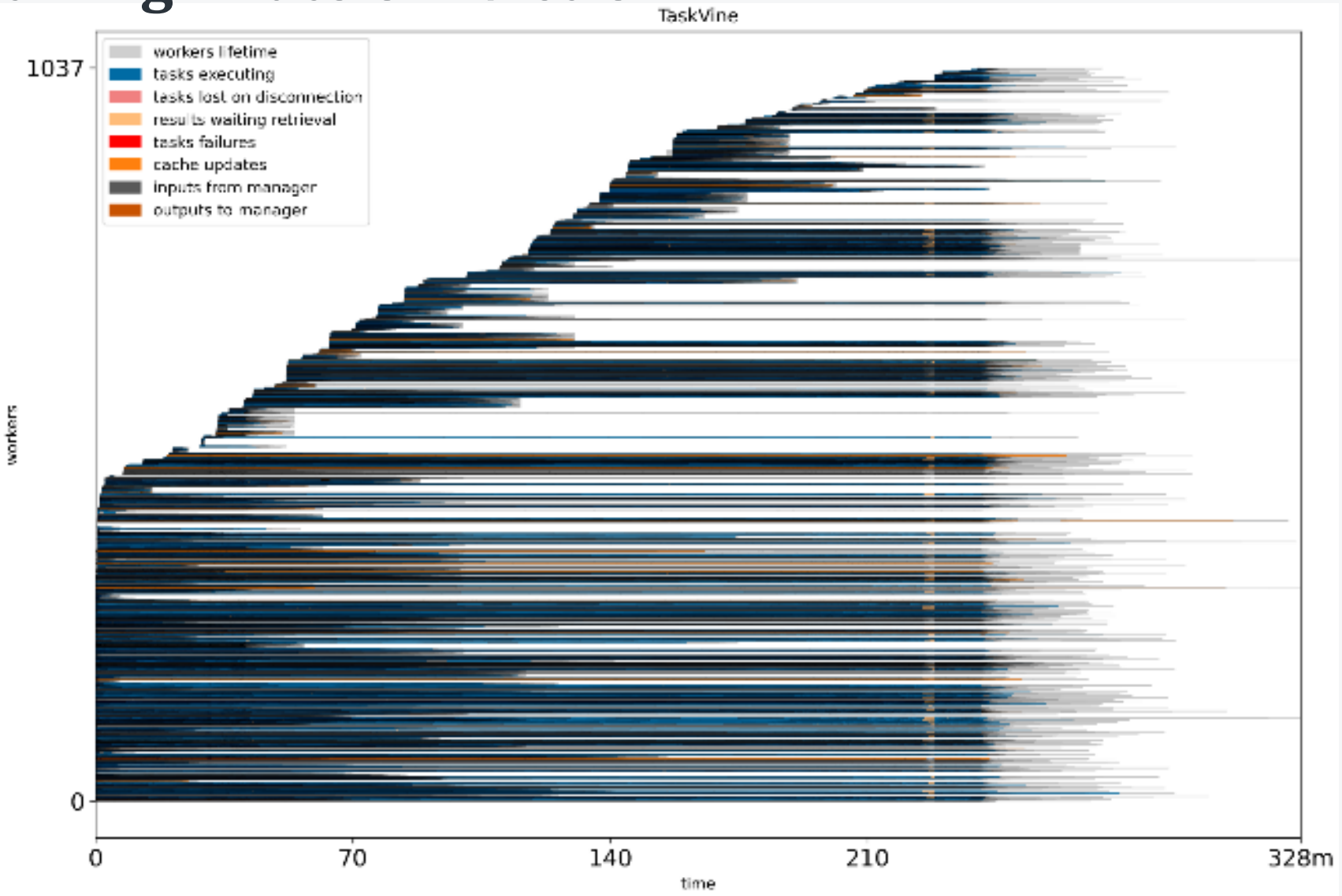
~5 hrs

turnaround

Output is parquet files.

Same distribution layer, three very different processors and output types — none of them required changing anything outside the analysis code.

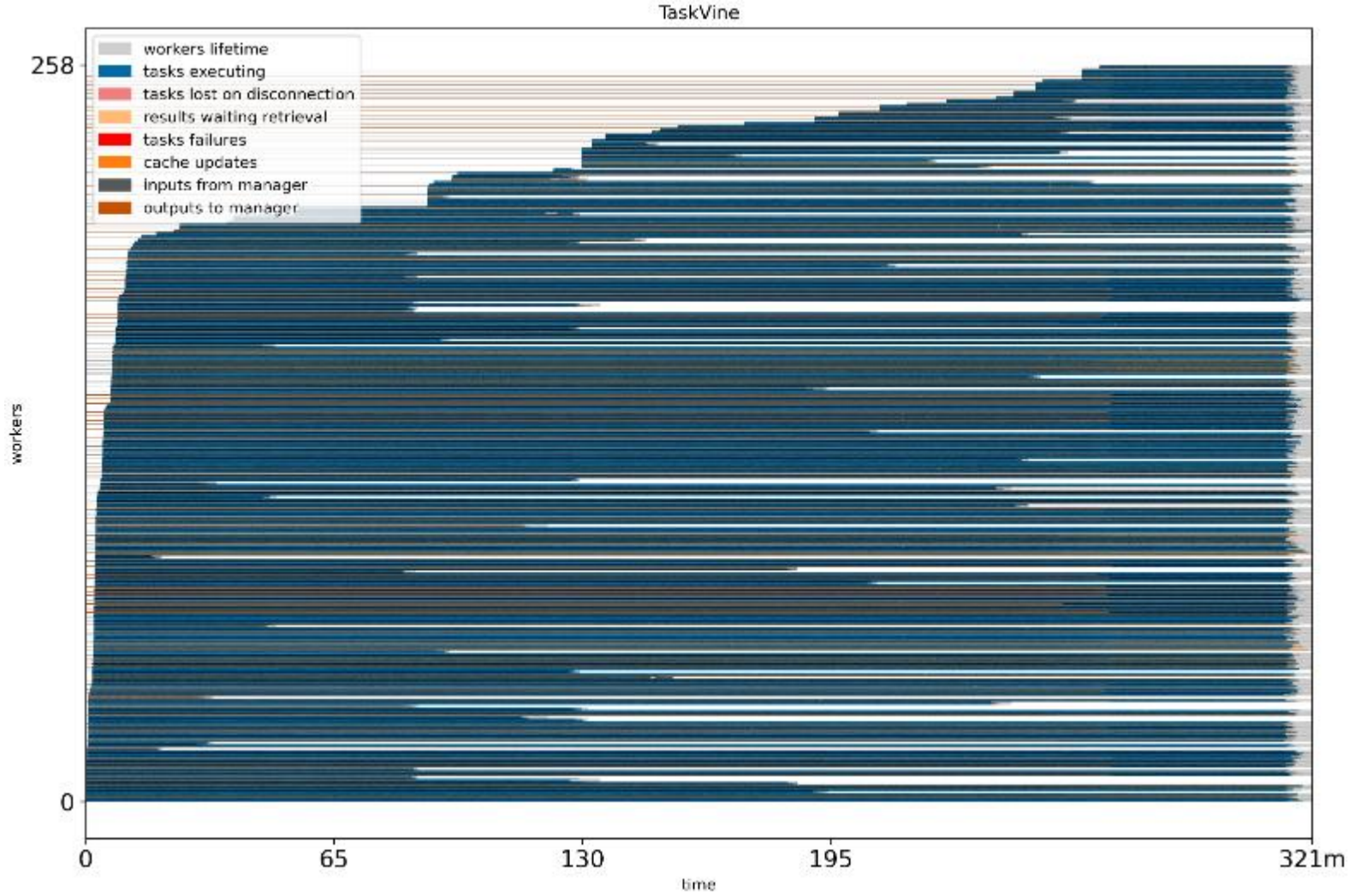
Run with High Rate of Eviction



1037 4-core workers connected total, but only 450 max at any one time
@ ND HTCondor opportunistic pool

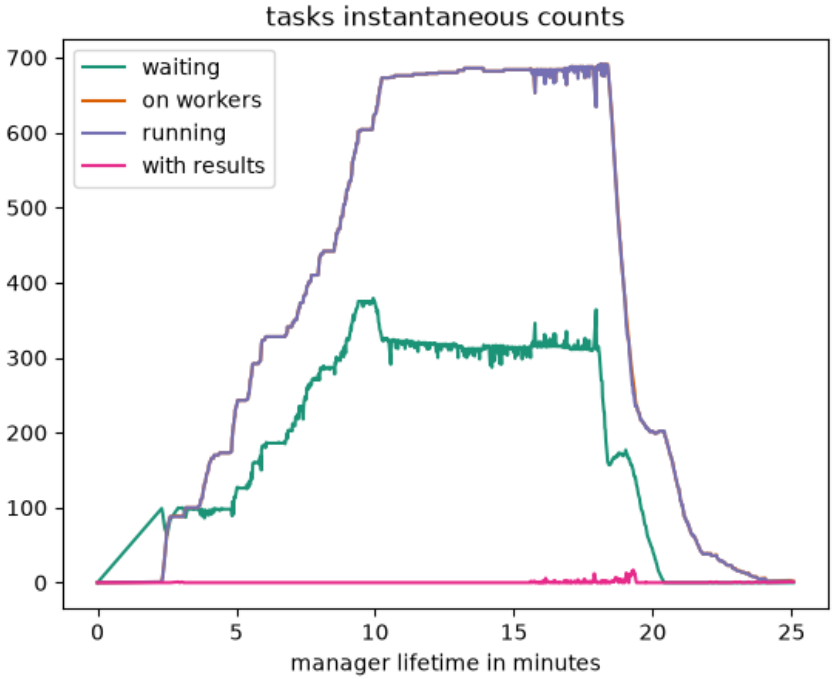
Data lives on vast

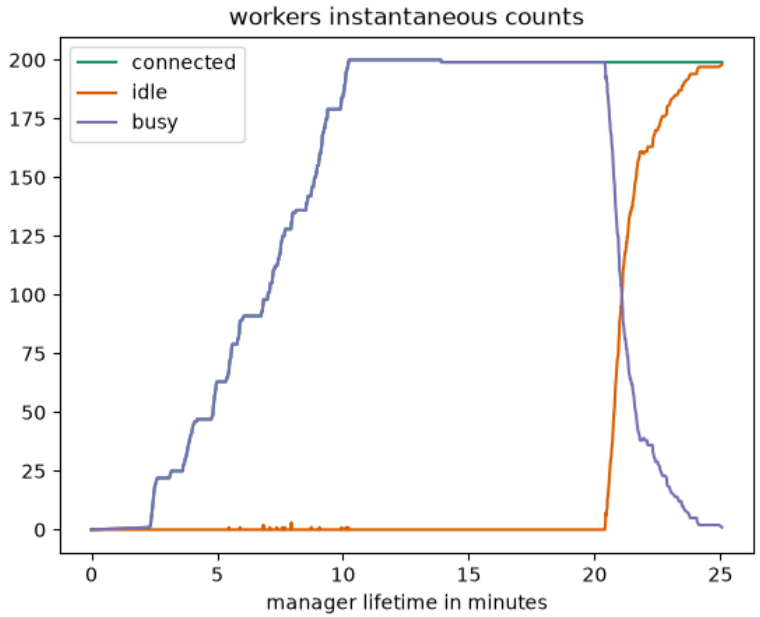
Run with High Rate of Eviction



258 8-core workers connected total, but only 200 max at any one time
@ ND HTCondor opportunistic pool

Bottleneck: xrootd timeouts





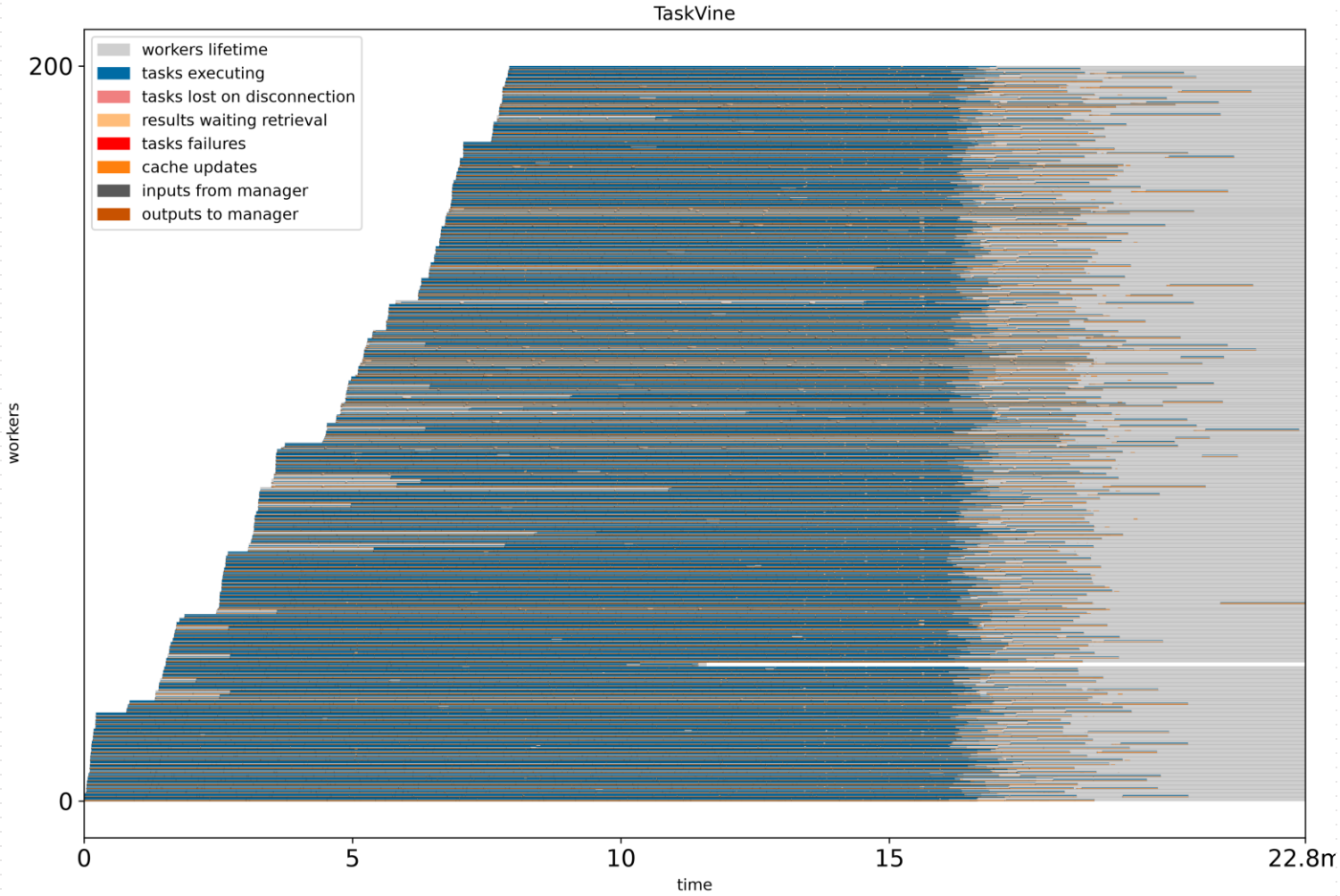
CORTADO (reduced SKIMMING WORKFLOW)

4 processors

30 datasets

0.5 TB

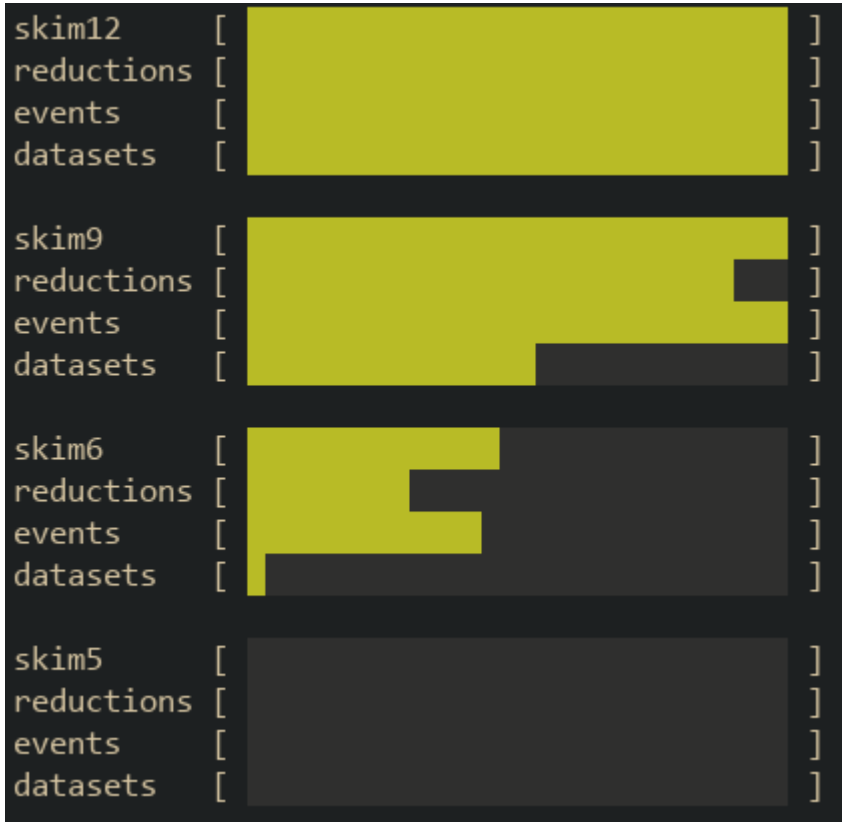
max 200 4-core workers



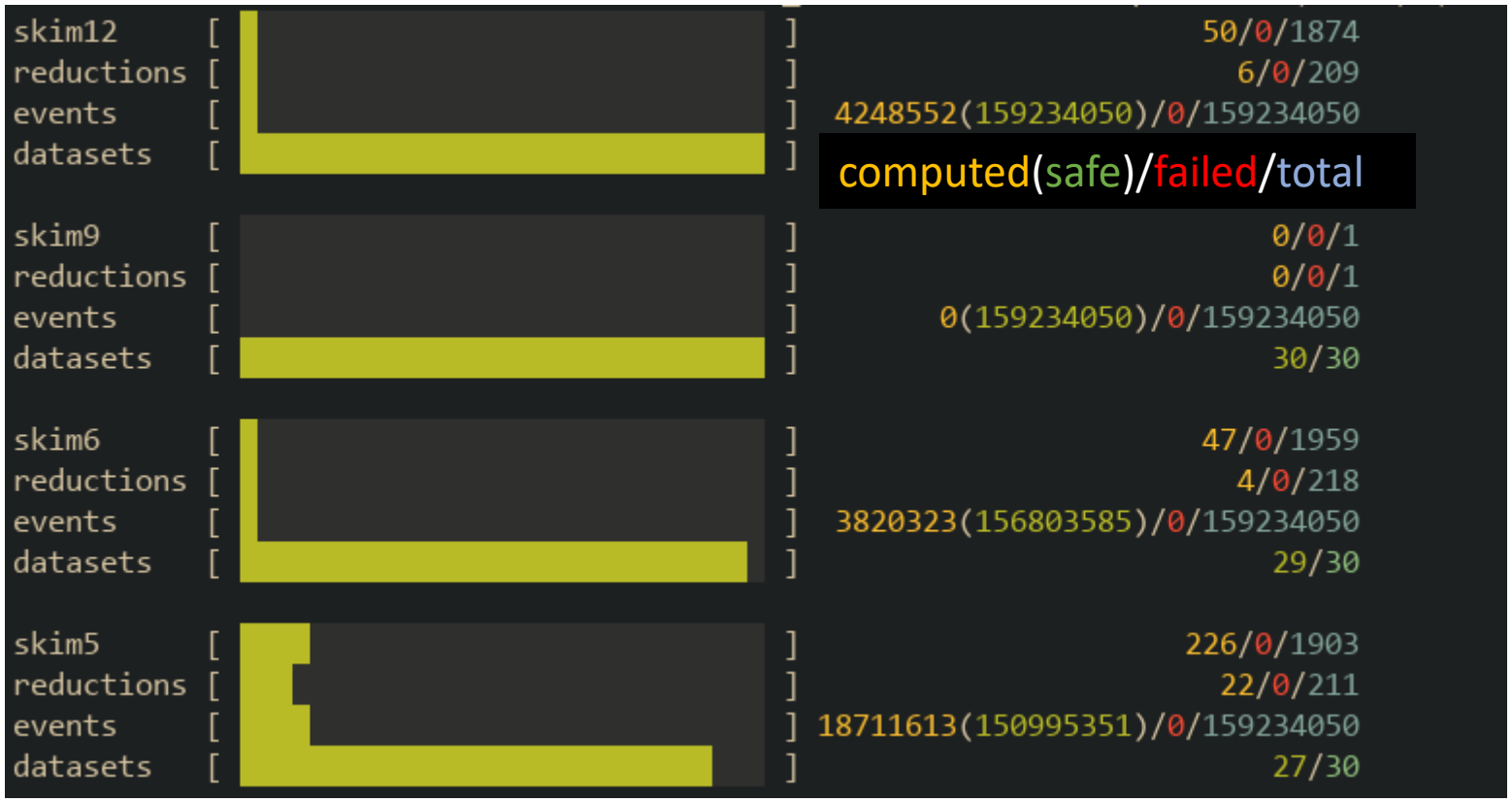
Example Run

Cortado Runs

One processor after another and one dataset after another with overlap



Recovery after workflow interruption



Automatic Resource Allocations / Adaptive Task Sizes

```
distributor = TaskVineDistributor(
    name="vine-reduce-btovar",
    port=[9120,9129],
    resources_processor={"memory_mb": 1000},
    resources_reducer={"memory_mb": 8000},
    checkpoint_dir=checkpoint_dir,
)
```

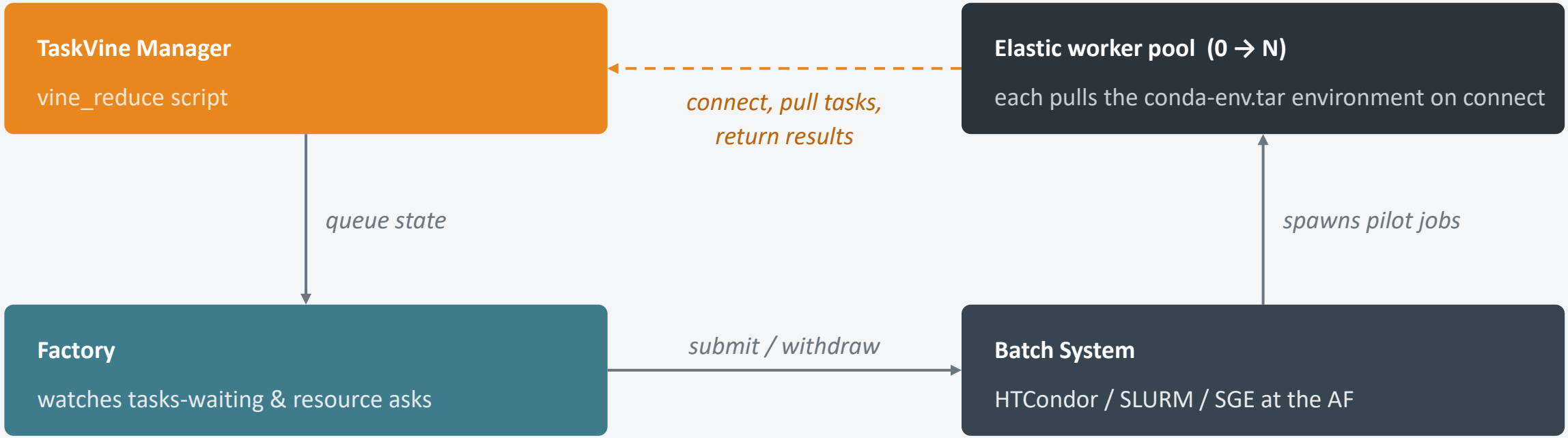
resource=measured(allocated)

```
processor skim12/dataset_00 ...taset_00/file_5.root[0:1590] (id=f5c1c3d7) cores=1(4) memory_mb=305(8064)
processor skim12/dataset_00 ...taset_00/file_2.root[0:3629] (id=00715cf0) cores=1(4) memory_mb=308(8064)
processor skim12/dataset_00 ...taset_00/file_1.root[0:81459] (id=24a40ac5) cores=1(4) memory_mb=428(8064)
processor skim12/dataset_00 ...taset_00/file_3.root[0:37776] (id=42dae82a) cores=1(4) memory_mb=366(8064)
processor skim12/dataset_00 ...taset_00/file_6.root[0:4101] (id=3835c401) cores=1(1) memory_mb=309(1911)
processor skim12/dataset_00 ...taset_00/file_7.root[0:53757] (id=f41336fa) cores=1(1) memory_mb=394(1911)
processor skim12/dataset_00 ...taset_00/file_8.root[0:74050] (id=9afbcaf0) cores=1(1) memory_mb=421(1911)
processor skim12/dataset_00 ...aset_00/file_15.root[0:51936] (id=9a0e87e2) cores=1(1) memory_mb=392(500)
processor skim12/dataset_00 ...aset_00/file_21.root[0:28135] (id=3deef33b) cores=1(1) memory_mb=348(500)
processor skim12/dataset_00 ...aset_00/file_19.root[0:24274] (id=d9b66a84) cores=1(1) memory_mb=343(500)
processor skim12/dataset_00 ...aset_00/file_13.root[0:53612] (id=ba141a61) cores=1(1) memory_mb=394(500)
processor skim12/dataset_00 ...aset_00/file_22.root[0:61694] (id=3c309e89) cores=1(1) memory_mb=408(500)
```

```
success             final     skim5/dataset_25 reduction of 4 partials (final) (id=5d6d352a) cores=1(2) memory_mb=1257(5250) wall_time_s=8(0)
                    processor skim5/dataset_28 ...aset_28/file_20.root[0:131716] (id=8ec09c33) cores=1(1) memory_mb=474(1000) wall_time_s=105(0)
resource_exhaustion result    skim5/dataset_29 reduction of 10 partials (final) (id=a43a2402) cores=1(1) memory_mb=9055(3000) wall_time_s=1(0)
2026/09/07 15:56:38.4 cctools-monitor[2240711]notice: sending kill signal to process 2240712.-1801786016

success             result    skim5/dataset_16 reduction of 10 partials (final) (id=7c6cdd6f) cores=1(3) memory_mb=5874(8000) wall_time_s=44(0)
success             processor skim5/dataset_24 ...aset_24/file_66.root[0:162696] (id=8d7b81a4) cores=1(1) memory_mb=505(1000) wall_time_s=127(0)
```

What runs on the facility



The loop is self-regulating: queued tasks make the factory ask for more pilot jobs, and an empty queue lets idle workers drain and exit.

- Factories can find managers through a catalog server.
- The TaskVine Manager can also report to Prometheus.

Nothing *is** ND specific.

RUNNING IT AT AN AF

Automatic worker factories: elastic by default

1	**Demand-driven scaling**	A factory process watches the manager's queue — tasks waiting, resources requested per category — and submits worker pilot jobs to the facility's batch system to match.
2	**Max, min bounds**	Workers scale within operator-set min / max bounds; when there's no work left, idle workers drain and exit instead of sitting on shared resources.
3	**No pre-installed dependencies**	Each worker pulls the conda-packaged Python environment on connect, so facility nodes don't need Coffea, vine_reduce, or any analysis code pre-staged.
4	**Resource categories pass straight through**	Separate core / memory / disk requests for processor and reducer tasks go to the batch system's own scheduler unchanged — no manual per-job sizing.

RUNNING IT AT ND AF

What an operator actually has to provide

1 User points the factory at existing pool

Same HTCondor / SLURM / SGE credentials, queues, and fairshare policy as any other job at the facility

2 Scratch space

Intermediate and staging files may reach O(10) GB.

3 Open ports

One port to talk to the manager. (Which can use SNI).
One port per worker so workers can talk to each other. (It works without it, but less efficient.)

4 Allow custom conda-forge / pixi envs

TaskVine (ndcctools) and vine_reduce are ordinary Python-ecosystem packages — no Notre Dame-specific components anywhere in the stack.

Thank you!

REPOSITORY

github.com/cooperative-computing-lab/vine-cms-analysis-stack

to start clone this repo

CONTACT

btovar@nd.edu