

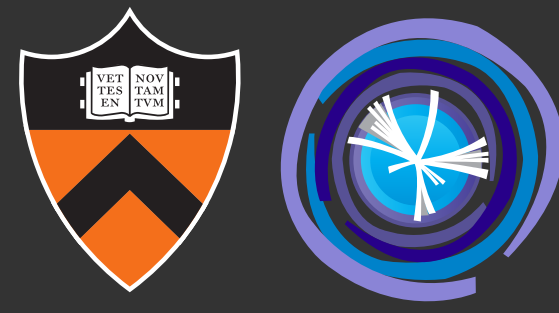


coffea 0.7 ↔ coffea 2025

Peter Fackeldey

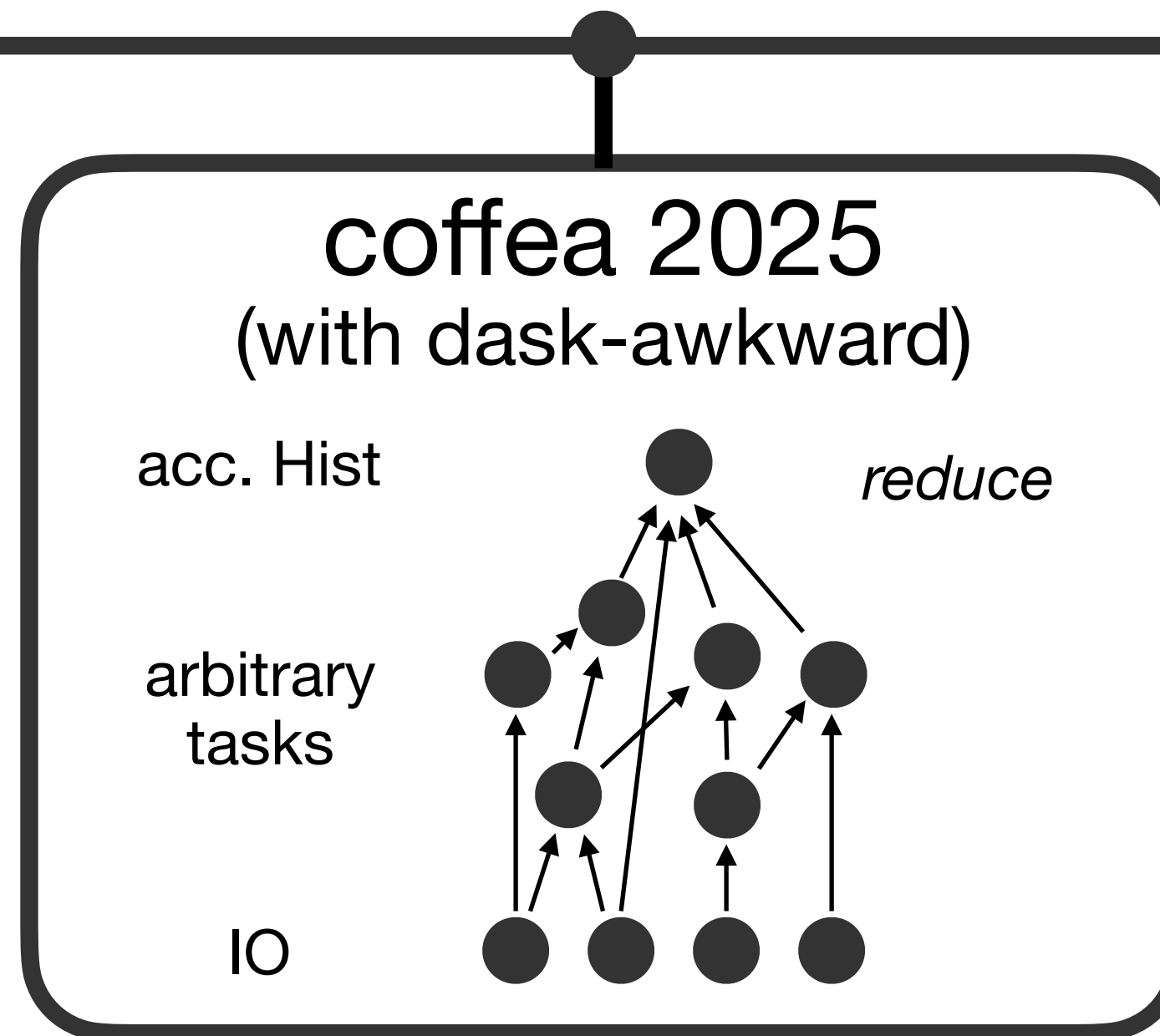
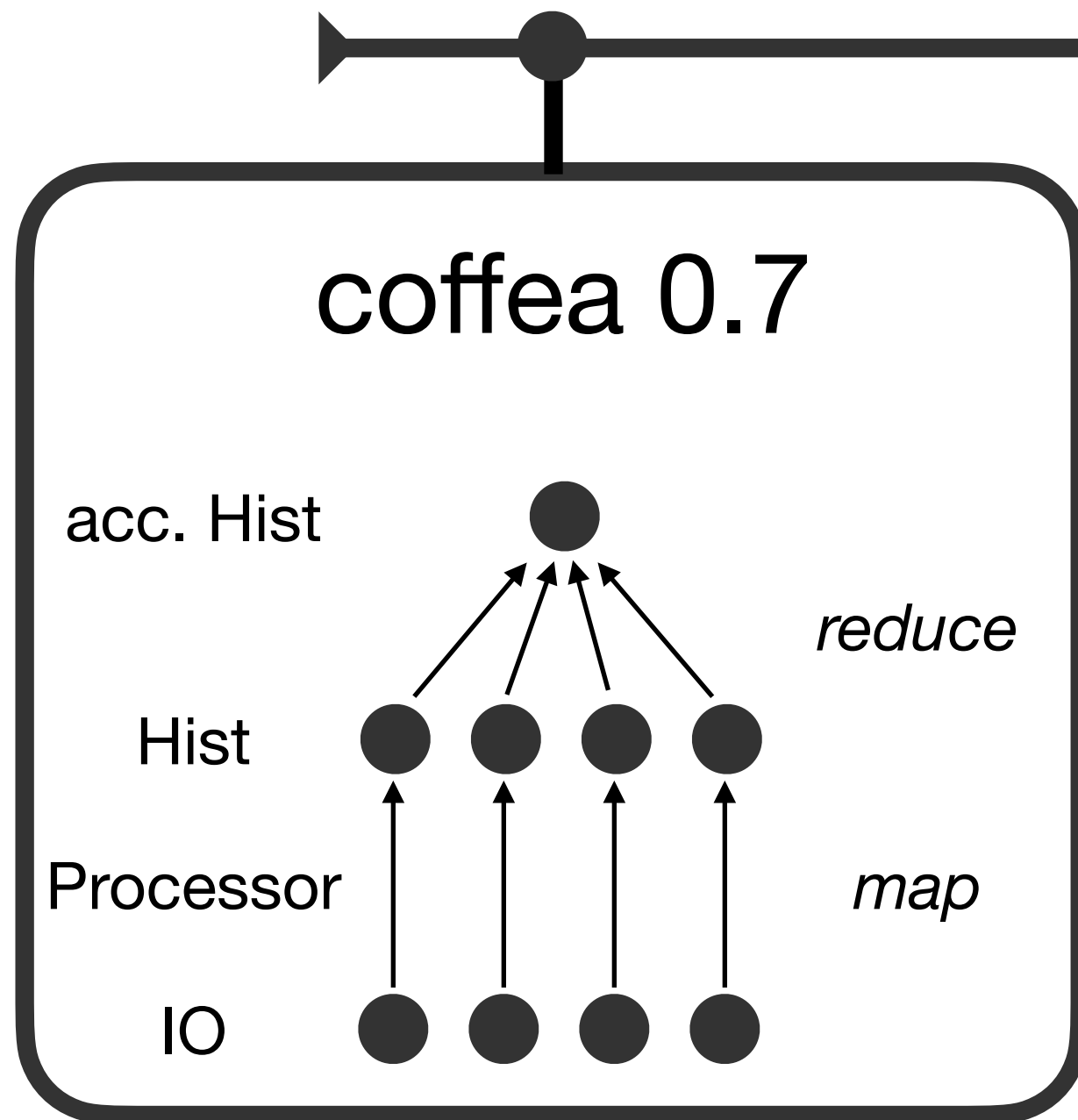
2/14/25

2 task graph complexity



simple graphs

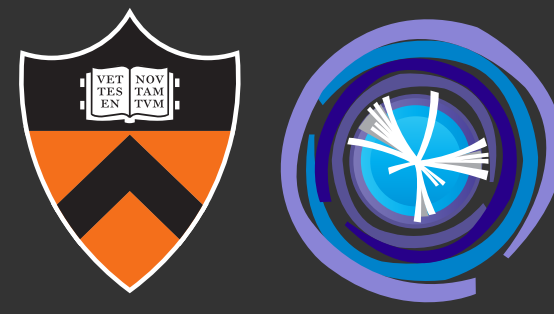
complex graphs



If we'd make every operation on each awkward buffer a dask task...

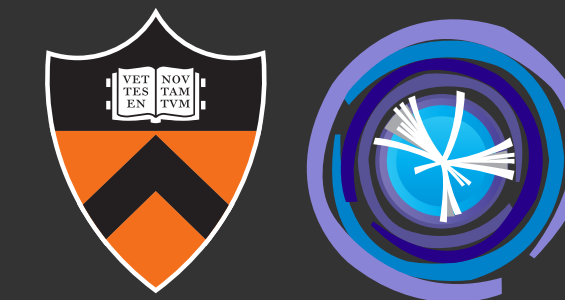
Is that the only difference?

3 coffea 0.7 ↔ coffea 2025



Category	coffea 0.7	coffea 2025
Graph size	# chunks	# chunks · # ops
Graph complexity	low	medium - high
Graph opt. potential	None	there is, but hard to quantify
Graph opt. time	None	O(10)min - O(1)h
Memory management	Python GC within Processor	Python GC within a task dask.distributed memory management
IO optimization	VirtualArrays	Awkward's typetracing
Software stack	awkward 1, uproot 4	awkward 2, uproot 5, dask-awkward

4 coffea 0.7 ↔ coffea 2025: Graph size

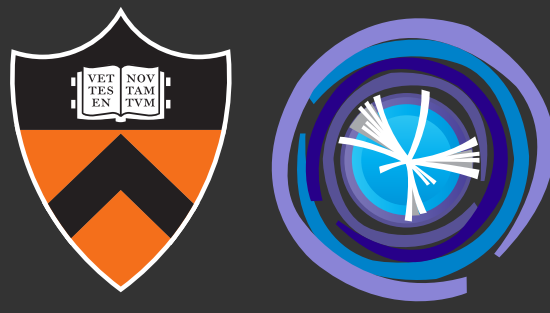


Category	coffea 0.7	coffea 2025
Graph size	# chunks $\sim O(10)k$	# chunks \cdot # ops $> O(10)k$

Notes:

- dask.distributed (single-threaded Python) has $\sim 1ms$ overhead for scheduling 1 task
→ large graphs $\sim O(10)min - O(1)h$ scheduling overhead
- scheduler has to manage all tasks and their state
- taskvine (C++) seems to be much more efficient than dask.distributed
- dask.distributed scheduler starts to choke at $\sim 50k$ tasks (personal experience)

5 coffea 0.7 ↔ coffea 2025: Graph opt. time

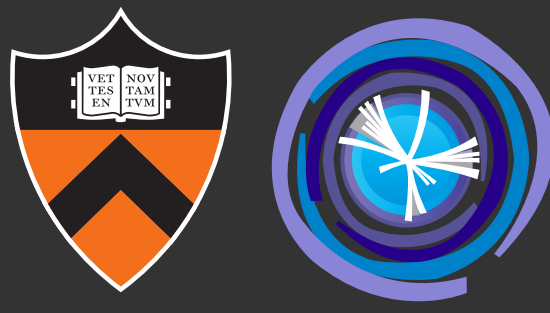


Category	coffea 0.7	coffea 2025
Graph opt. time	None	O(10)min - O(1)h

Notes:

- Some dask opt. algorithms already improved from $O(N^2) \rightarrow O(N)$
- Most of the optimization time spent in awkward's typetracing
- We have ideas to reduce this time significantly (reusing tracing information)
- However, full-scale analysis are likely experiencing >1 h optimization times (without any feedback/output...it's an unpleasant user experience)

6 coffea 0.7 ↔ coffea 2025: Memory management

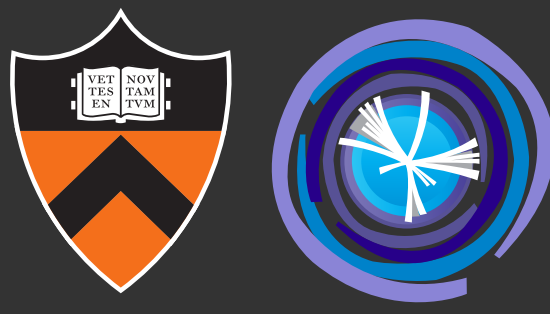


Category	coffea 0.7	coffea 2025	
Memory management	Python GC within Processor	Python GC within a task	dask.distributed memory management

Notes:

- Dask's memory management (likely) often the root cause of too much memory consumption
- We're suspecting that dask sometimes decides to hold onto arrays longer than necessary
- We found that tree reductions are a way to convince Dask's memory management to free chunks of arrays (basically avoiding these memory issues)
- General: when is it better to recompute arrays instead of holding them in memory?

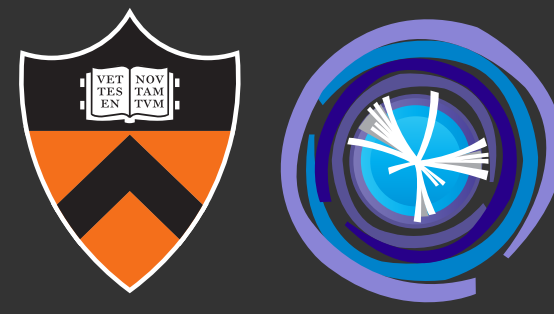
7 coffea 0.7 ↔ coffea 2025: IO optimization



Category	coffea 0.7	coffea 2025
IO optimization	VirtualArrays	Awkward's typetracing

Notes:

- Tracing is more restrictive than on-demand loading, e.g. only one if-branch can be traced (same for libraries such as JAX)
- Tracing only works inside the “awkward-world”, can't easily switch to e.g. NumPy
- Tracing flow can't be interrupted → requires users to sometimes mock arrays
- However, tracing let's us find out all needed branches ahead of time → can be much more efficient than multiple read requests
- VirtualArrays are currently in “re-implementation” phase in awkward 2

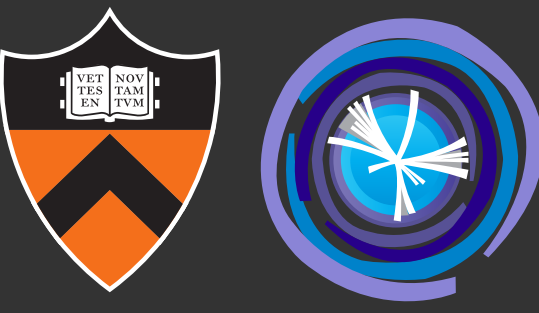


8 coffea 0.7 ↔ coffea 2025: Software stack

Category	coffea 0.7	coffea 2025
Software stack	awkward 1, uproot 4	awkward 2, uproot 5, dask-awkward

Notes:

- awkward 2:
 - ***much*** faster, especially together with vector v1.6.1 and newer
 - new features, such as named axes, and plenty of bug fixes
- uproot 5:
 - supports RNTuple reading
 - stable memory profile for consecutive reads from the same opened file



9 What are we currently doing?

- Adding back VirtualArrays in awkward 2:
 - allows to run coffea 0.7-like analysis
 - avoids “sharp bits” of tracing
 - lets users migrate to newer awkward, uproot, etc versions
- Adding a way to manually tell dask-awkward:
“Hey, I need branches X, Y, Z, please load them ahead of time”
 - Let’s us skip tracing → much faster graph opt. time
 - can be more efficient than loading on-demand (i.e. single network request)
- Educating physicists about **dak.map_partitions** to simplify graphs
→ **dak.map_partitions** produces a single node no matter what the mapped function does!

10 How much dask can we have?

