

A Pull-Based Task Distribution System for HEP Analysis

Coffea integration and validation of hq (hep-queue) on the AGC

Michael Noamesi

Gettysburg College

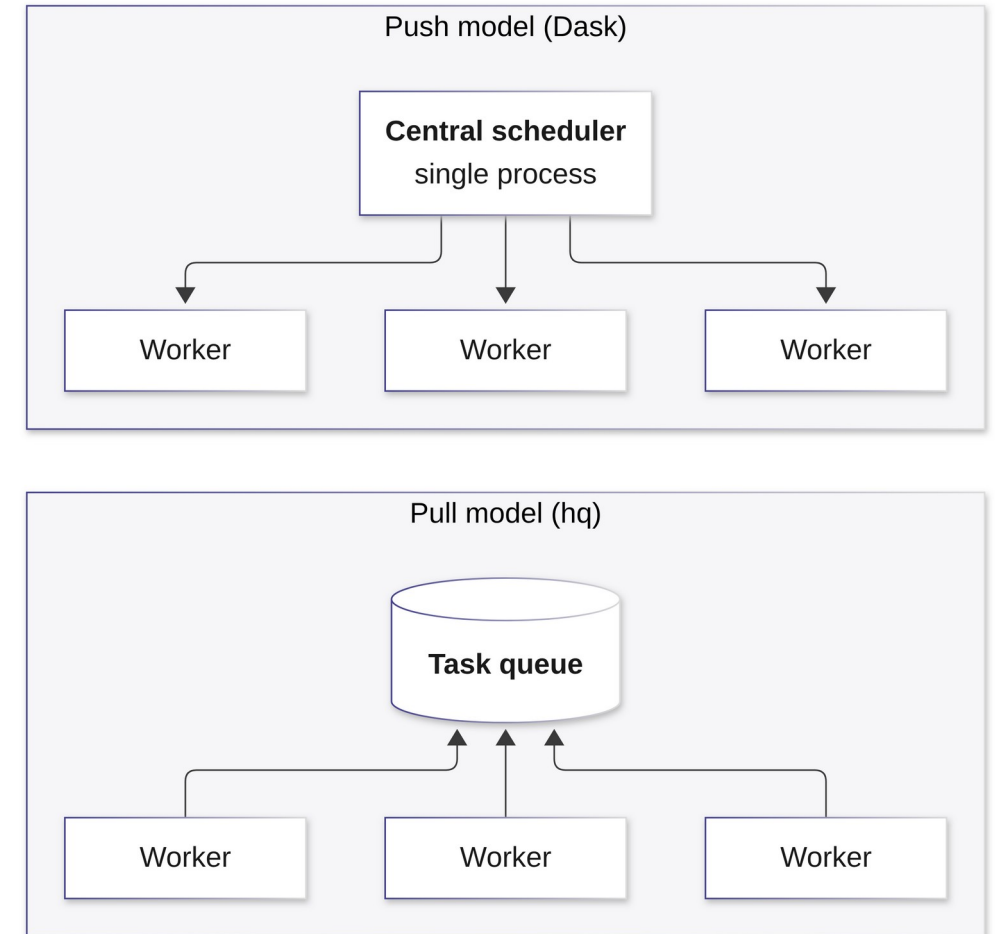
Mentors: Peter Fackeldey (Princeton University),
Oksana Shadura (University of Nebraska-Lincoln)

IRIS-HEP Fellows Final Presentations, October 2026



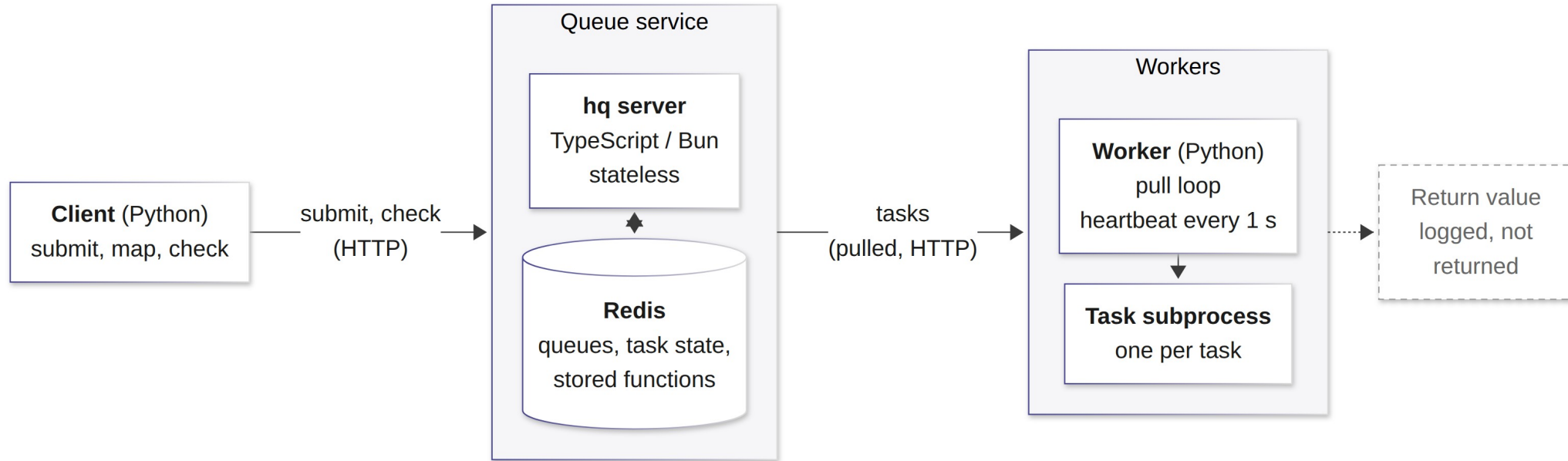
1. Motivation
2. Background: the hq prototype
3. Project objectives
4. Implementation
5. Validation on the AGC $t\bar{t}$ analysis
6. Summary and outlook

- Columnar analyses in coffea divide a dataset into thousands of independent chunks
- Dask, the standard distributed backend, uses a central scheduler that assigns every task to a worker
- At the 200 Gbps Challenge on coffea-casa, scaling the scheduler beyond ~2000 workers and its memory use were identified as open issues [1]
- A pull-based queue inverts this model: idle workers request the next task, so load balancing follows from worker speed



[1] S. Albin et al., "Tuning the CMS Coffea-casa facility for 200 Gbps Challenge", arXiv:2503.12991

Background: the hq prototype



- Stateless HTTP server (TypeScript, Bun); all queue and task state is held in Redis, so the server can be restarted without losing queued tasks
- Workers pull tasks over HTTP and execute each one in a separate subprocess
- Heartbeat every 1 s; the tasks of a worker silent for 30 s are marked *lost*
- `map ( )` stores the function once; each task carries only its argument

Design and implementation: P. Fackeldey, January–February 2026

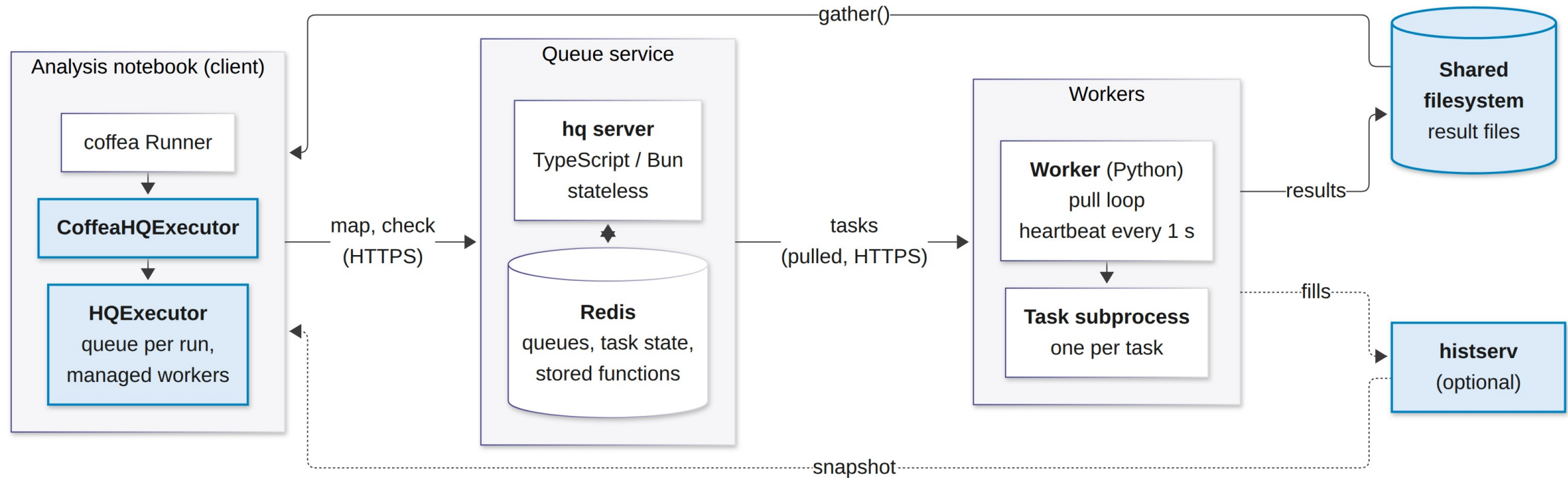
Proposal: harden hq for deployment on coffea-casa (TLS, worker telemetry, per-task software environments, failure handling, validation at scale)

Objectives addressed in this work

End-to-end execution of a coffea analysis on hq

1. Secure transport between client, workers and server (TLS)
2. Return of task results to the client
3. Integration with the coffea Runner as a standard executor
4. Validation of the physics output against coffea's reference executor

Architecture after this project



 Added during the fellowship

Client and worker connections verify the server certificate (TLS)

Tasks notebook → queue service → workers (pulled over HTTPS)

Results workers → shared filesystem → notebook (gather ())

Histograms optional: workers → histserv → notebook (one snapshot at the end)

Contributions to hq



Period	Contribution	Details
June	TLS on client and worker	Certificate verification; HTTPS end-to-end test script; fix for a server start-up failure with TLS enabled. <i>Merged upstream (pfackeldey/hq #1)</i>
July	Result retrieval	Workers write each result to a shared filesystem; the status carries its path; <code>client.gather()</code>
July	HQExecutor	Private queue per run, managed local workers, process-group termination
August	CoffeaHQExecutor	Implements coffea's ExecutorBase; outputs merged with coffea's <code>accumulate</code>
August	histserv integration	Optional server-side histogram filling with idempotent fills
August	agc-hq, documentation	Full AGC ttbar pipeline on hq; architecture notes, 8 design records, operations guide

20 commits, ≈1,260 lines added to hq (≈690 in the library); separate agc-hq and hq_docs repositories

```
from hq.coffea import CoffeaHQExecutor

executor = CoffeaHQExecutor(
    host="http://localhost", port=3000, n_workers=8,
    pickle_modules=(utils, ttbar_processor),    # notebook-local modules
)

run = processor.Runner(executor=executor, schema=NanoAODSchema, chunksize=200_000)
out, metrics = run(fileset, processor_instance=TtbarAnalysis(...), treename="Events")
```

- Replaces processor.FuturesExecutor or DaskExecutor; processor, fileset and post-processing are unchanged
- One hq task per chunk; outputs merged with coffea's accumulate
- pickle_modules serialises notebook-local modules by value, as the AGC already does for Dask

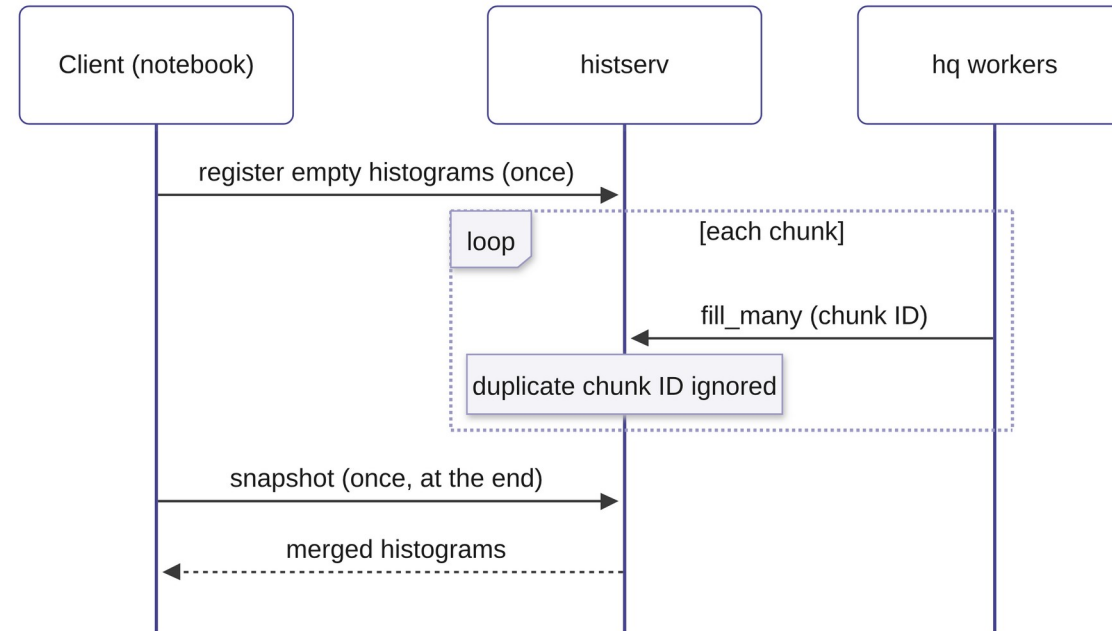
Issues identified with a production workload



Symptom	Cause	Resolution
Worker failed to parse task reports	NanoAOD writes warnings to stderr; the worker parsed all of stderr as JSON	Parse only the final line
Large jobs failed to launch	Pickled task passed as a command-line argument, exceeding the OS limit	Pass the payload through a temporary file
Workers continued polling after shutdown	Terminating a worker left its child processes running	One process group per worker; terminate the group
Distinct functions could share a cache key	Key derived from Python's <code>id()</code> , which is reused	Key = SHA-256 of the pickled function

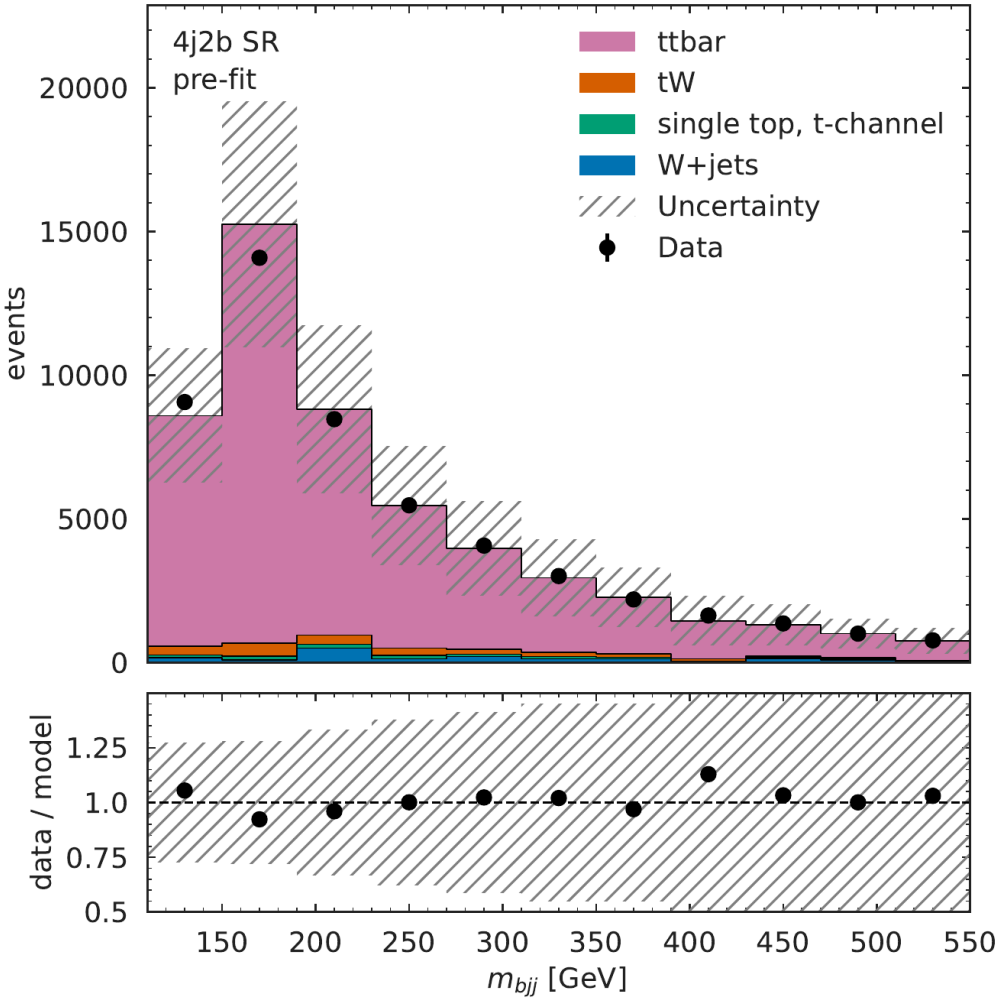
None of these occurred with the prototype's example tasks.

Server-side histogram aggregation (histserv)



- Default: each task returns all of its histograms (MB per chunk), and the client merges them
- With histserv (scikit-hep/histserv): fills are sent to a gRPC histogram server and merged on arrival
- Each flush is tagged with its chunk identity; a repeated chunk is ignored, so a retry cannot double-count
- No change to the processor's `.fill()` calls; enabled by the optional notebook flag `USE_HISTSERV`

Validation: AGC ttbar analysis



4j2b signal region, pre-fit, produced by this run (hq, histserv, cabinetry)

Run parameters

Dataset	9 samples, CMS 2015 Open Data (1 file each)
Events	9.8 M (1.3 GB read via XRootD)
Tasks	9 preprocessing + 50 processing
Workers	8, single node (AMD Ryzen 5 7235HS)
Processing wall time	115 s
Failed tasks	0
Worker utilisation	≈ 80 %

Correctness: nominal histograms identical bin by bin to coffea's FuturesExecutor (automated comparison, with and without histserv)

Single-node performance comparison

Executor	Description	Processing time
FuturesExecutor	coffea's single-machine process pool	91 s
DaskExecutor	local Dask cluster (scheduler and workers on one node)	98 s
hq	queue service and pulling workers	156 s
hq + histserv	as above, histograms filled on a server	133 s

Median of 3 runs. Same node, 8 single-threaded workers each, full AGC ttbar input (9.8 M events) read from UNL over XRootD.

- Process pools start their workers once; hq starts a new Python process for every task, and starting Python and importing coffea alone takes 1.7 s
- Running each task in its own process is deliberate:
 - it allows a software environment per task; a Dask worker's environment is fixed when the worker starts
 - a blocking task cannot delay the heartbeat; in Dask, tasks run in the worker process and can stall its event loop
- histserv removes the file-based return of results: faster than plain hq in every round, by 11–23 s
- To be explored: running tasks in threads instead of new processes, avoiding the start-up cost

Summary

- hq executes a complete coffea analysis (AGC ttbar) through the standard Runner interface
- Physics output agrees bin by bin with coffea's reference executor
- Added: TLS verification, result retrieval, HQExecutor, CoffeaHQExecutor, histserv integration, documentation
- Integration with a production workload exposed four defects in the prototype, all resolved

Outlook

- Automatic requeueing of lost tasks, with a retry limit
- Lower per-task overhead: reuse task processes where isolation is not needed
- Worker telemetry and per-task software environments
- Deployment on coffea-casa with HTCondor or Kubernetes workers
- Scale tests against Dask and TaskVine (hundreds to thousands of workers)
- Upstream the remaining contributions to pfackeldey/hq

Acknowledgements



I thank my mentors Peter Fackeldey (Princeton University) and Oksana Shadura (University of Nebraska-Lincoln) for their guidance, and the IRIS-HEP Fellows Program for its support.

This work was supported by the National Science Foundation under Cooperative Agreement PHY-2323298.

Resources

Code	github.com/ijohnkojo/hq
Analysis	github.com/ijohnkojo/agc-hq
Documentation	github.com/ijohnkojo/hq_docs
Upstream	github.com/pfackeldey/hq



Thank you for your attention

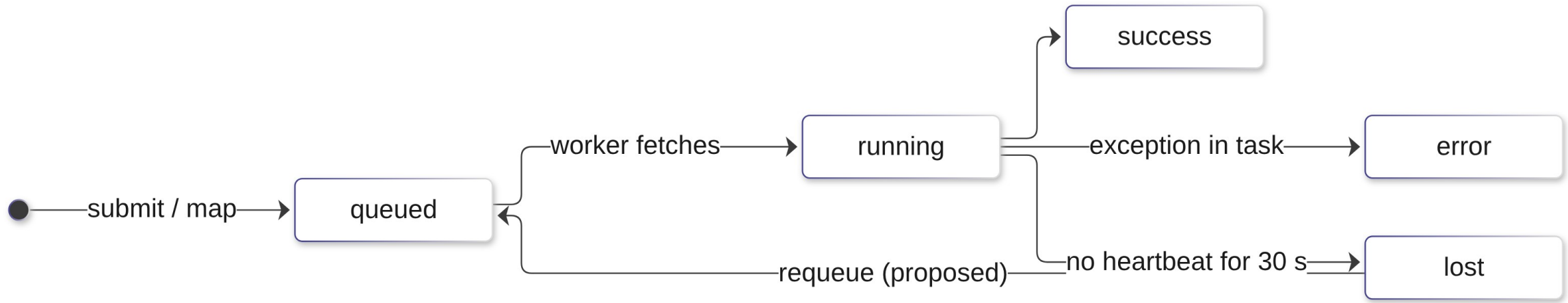
Questions?

michael.john.kojo.noamesi@cern.ch



Backup

Backup: task lifecycle and worker failure



- **Current:** the running tasks of a failed worker become *lost* after 30 s, and the client receives an error
- **Proposed:** requeue them with a maximum number of attempts; the server already records which worker owns which task (`workers:running:{workerId}`)
- **Consequence:** at-least-once execution, so side effects must be idempotent; histserv fills already are

Backup: projected control-plane throughput

Component	Throughput (single instance)
Asynchronous Python HTTP server, single thread	~10k requests/s
Bun HTTP server (hq server)	~150k requests/s
Redis	~150k operations/s

- About 15× control-plane headroom on one server; the stateless server also scales horizontally
- Component benchmarks by P. Fackeldey, not an end-to-end AGC measurement; measuring this at facility scale is future work

Backup: timing methodology

Executor	Median	Min	Max
FuturesExecutor	91 s	90 s	105 s
DaskExecutor (local)	98 s	96 s	107 s
hq	156 s	130 s	157 s
hq + histserv	133 s	109 s	146 s

- All four configurations from the same AGC notebook; kernel restarted between runs; order rotated over 3 rounds
- 8 single-threaded workers each; one file per sample; chunk size 200k events; no ML inference
- The timer covers the processing step only; the Dask cluster is started before it, while hq and FuturesExecutor start their workers within it
- Input is read from UNL over XRootD, so run-to-run variance is dominated by the network

Backup: additional issues



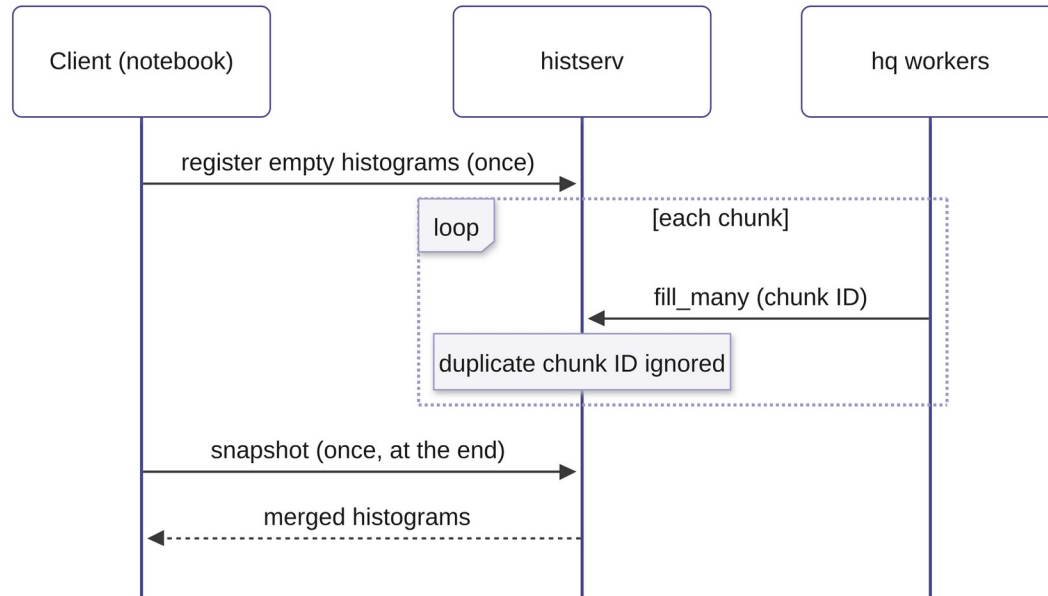
Symptom	Cause	Resolution
coffea not importable inside tasks	Task subprocesses ran python from PATH, not the worker's interpreter	Use <code>sys.executable</code>
Server failed at start-up with TLS enabled	Circular import between the configuration and logging modules	Logger reads its level from the environment, breaking the cycle

Backup: result storage design



- Results are megabytes per chunk (histograms and metrics)
- Storing them in the task status would place every result in Redis and pass it through the HTTP server twice, for data that one client reads once
- Workers therefore write `{HQ_RESULT_DIR}/{queue}/{task_id}.pkl`; the status carries only the path; `gather()` reads it
- With histserv, histograms leave this path entirely; only small counts remain
- *Open question for coffea-casa: which shared storage the workers can access*

Backup: histserv message sequence



- Time runs from top to bottom; solid arrows are requests, the dashed arrow is the reply
- Not shown: the hq server and Redis; tasks still reach the workers through the hq queue
- Without histserv, each task writes its histograms to the shared filesystem and the notebook merges them at the end

- **register empty histograms (once):** before processing, the notebook sends the histogram definitions (axes and bins, no content); the handles it gets back are packed into every hq task
- **loop, fill_many (chunk ID):** for each chunk, the task collects its `.fill()` calls locally and sends them in one call when the chunk is done, tagged with the file and entry range; the server adds them to its totals on arrival
- **duplicate chunk ID ignored:** a chunk the server has already received is rejected, so a retried task cannot count twice; a task that fails sends nothing
- **snapshot, merged histograms (once):** after all tasks finish, the notebook fetches the merged histograms; plotting and the fit run unchanged