Under the Hood of HPC

Analyzing Cluster Processes Through Data

Aaron Küsters

2026-05-12

NHR4CES NHR for Computational Engineering Science

P D A S Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

Outline

1. A short tour of (object-centric) process mining
2. Turning live SLURM state into object-centric event data
3. One week of CLAIX, through different lenses
4. Predicting job outcomes (and where it breaks down)
5. Where this is going

Process mining 101: from log to traces

Event log (source)

Case	Activity	Time
A	Submit	09:00
A	Check	09:15
A	Approve	09:30
A	Pay	09:35
B	Submit	09:10
B	Check	09:25
B	Reject	09:40
...	...	...

Process mining 101: from log to traces

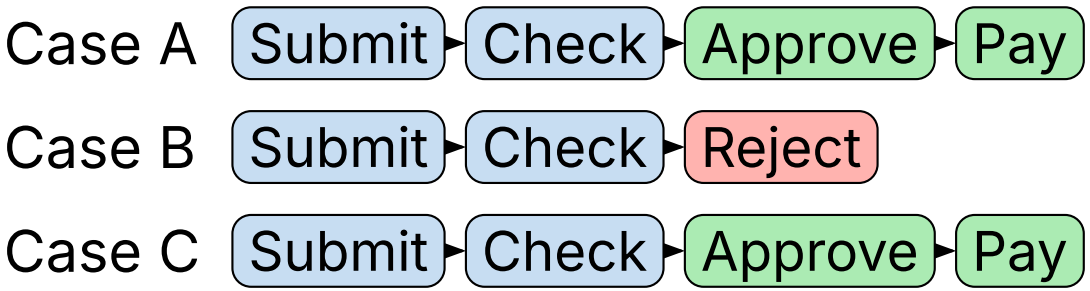
Event log (source)

Case	Activity	Time
A	Submit	09:00
A	Check	09:15
A	Approve	09:30
A	Pay	09:35
B	Submit	09:10
B	Check	09:25
B	Reject	09:40
...	...	...

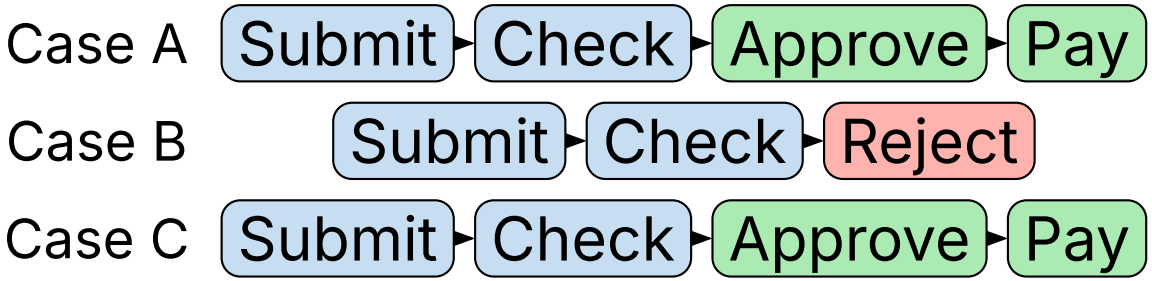
→

group by case

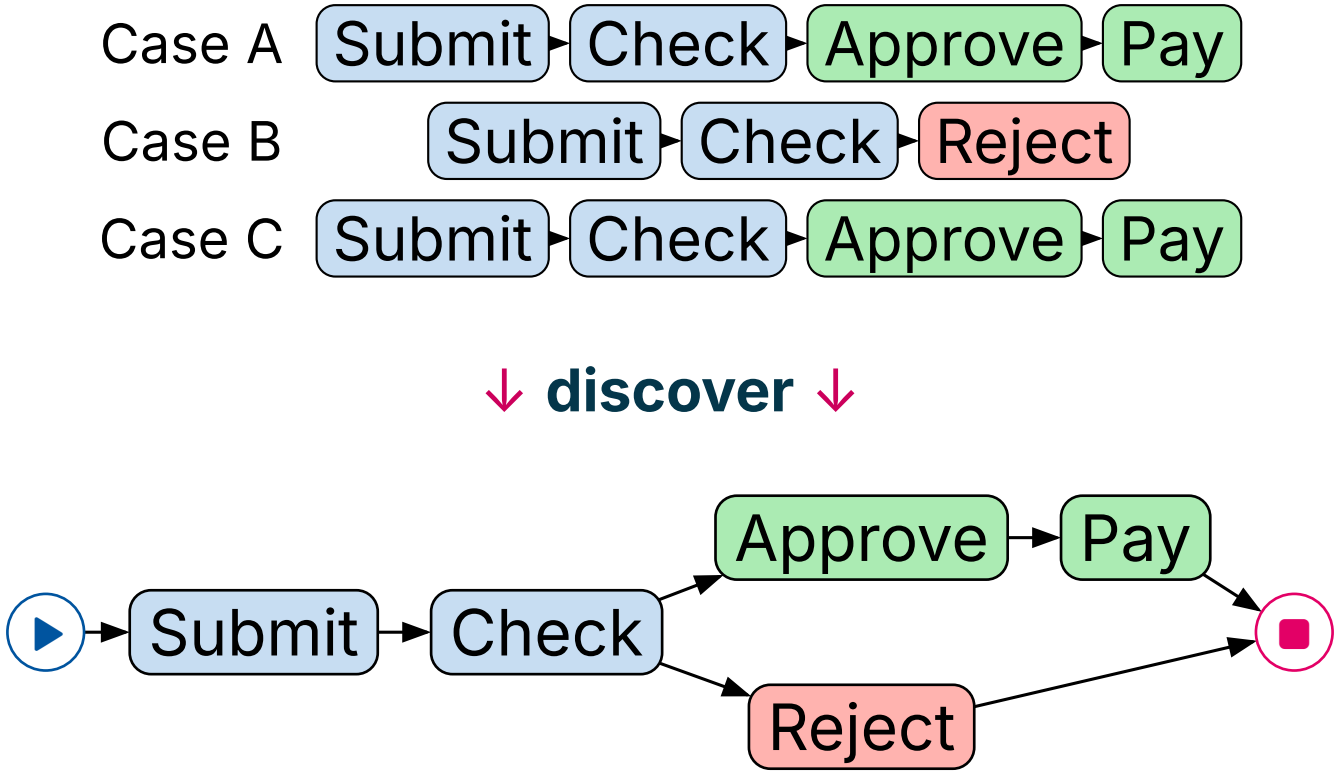
Per-case traces



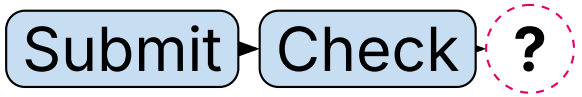
Process mining 101: from traces to a model



Process mining 101: from traces to a model



Process mining 101: predicting from a partial trace

Case D, so far: 

Process mining 101: predicting from a partial trace

Case D, so far: Submit → Check → ?

↓ predict ↓

from trace prefix, time, case attributes, history aggregates

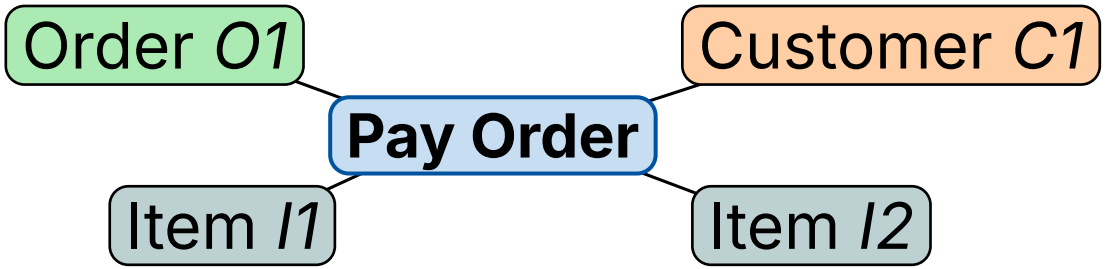
Outcome	Next activity	Remaining time
Approve or Reject?	What is the next step?	How long until done?
↑ this talk		↑ this talk

One event, many objects

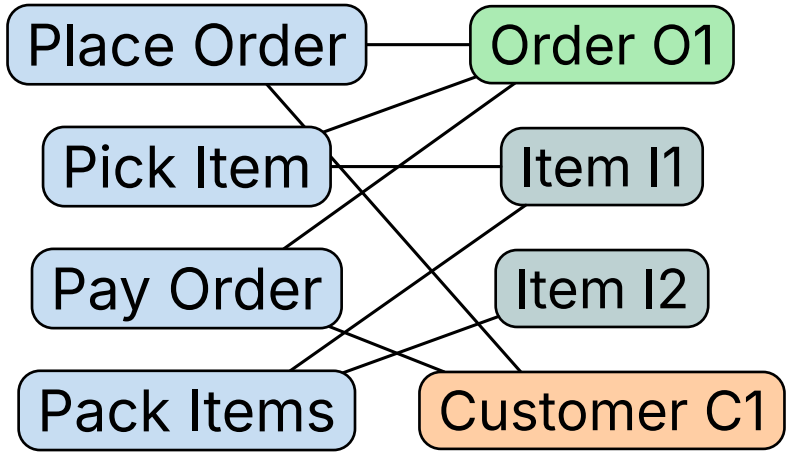
Real events rarely belong to a single case. A **Pay Order** event involves an **order**, several **items**, and a **customer**.

Flatten on order and you lose the customer.
Flatten on customer and you lose the items.

OCPM keeps all object types at once.

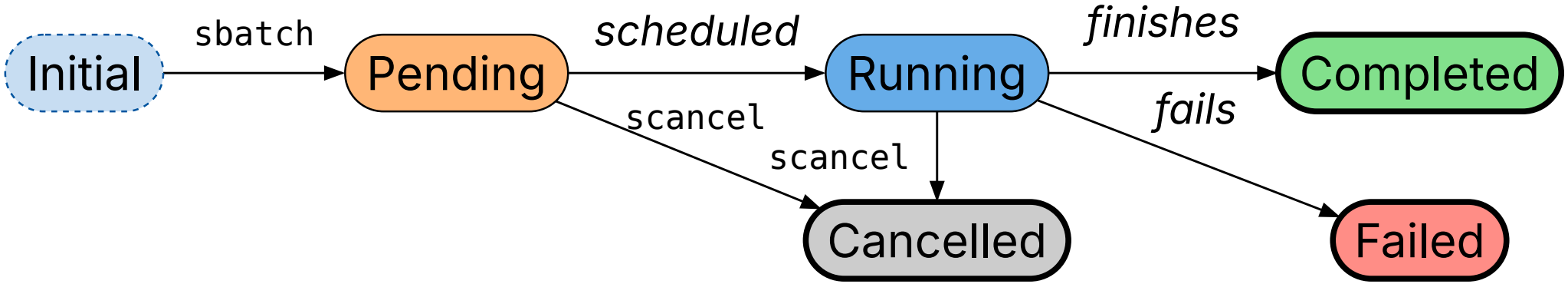


Object-centric event logs (OCEL)



Events on the left, objects on the right. Each event links to **any number** of objects of **any types**. Object types here: Order, Item, Customer.

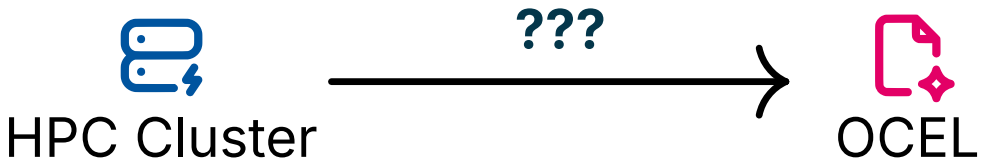
A SLURM job, as a process



SLURM **defines** the process. Our job is to capture each transition *and* enrich it with the context around it: host load, account, partition, …

Turning a live cluster into an event log

The gap



- SLURM doesn’t ship an event log. It exposes **snapshots** of the current state.
- Two paths in: query the **internal database**, assuming it contains historical data, or repeatedly **poll the current state** every few seconds.
- We chose polling. It is portable, reproducible, and works without admin rights.

Slurry: poll, diff, emit events

`slurry` is a small Rust crate that wraps the polling loop:

- fetch the current cluster state every ∼5 s
- parse it into typed structs
- **diff** against the previous snapshot
- emit one OCEL event per state change

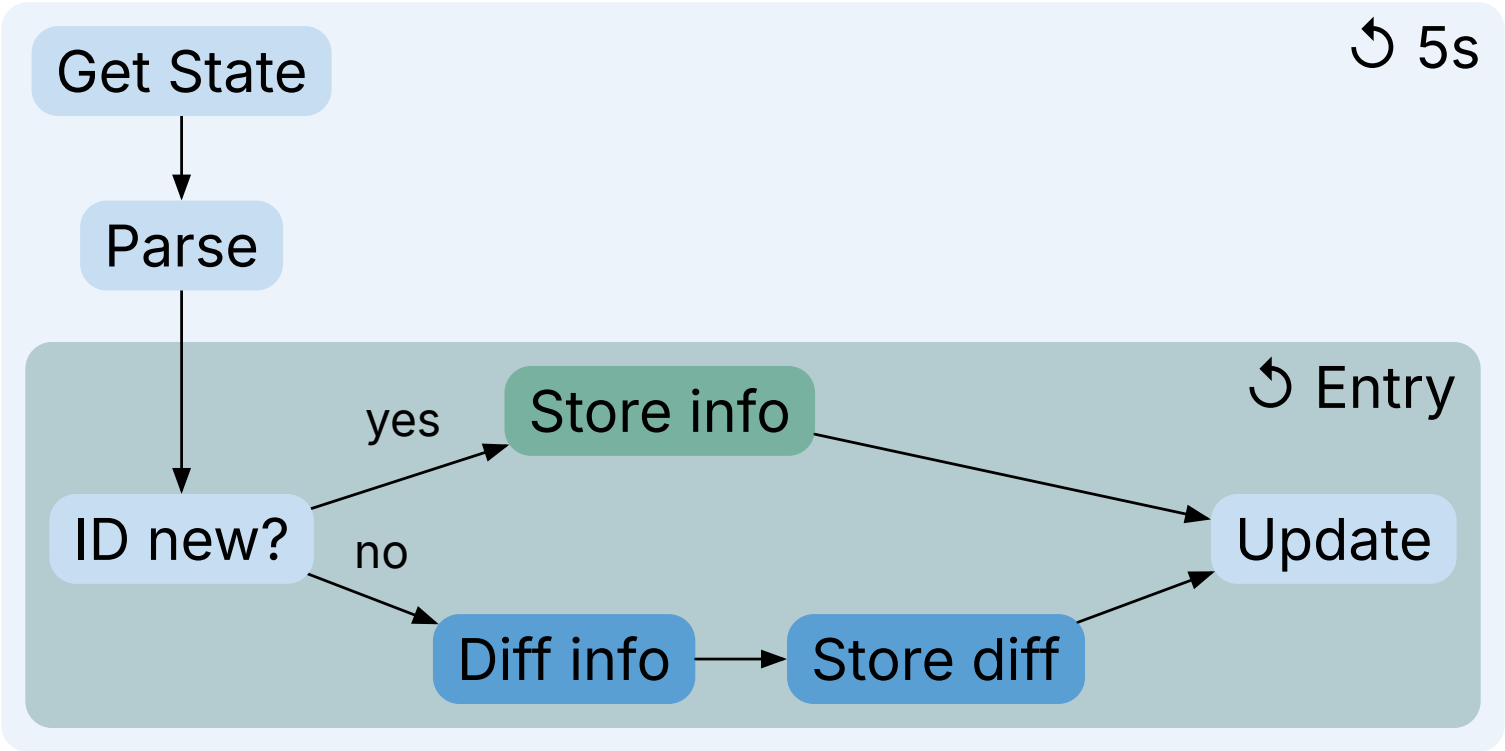
Same crate also handles job submission and SSH port forwarding.

crates.io/crates/slurry · docs.rs/slurry

Credits to Uy Sa Huynh for the original implementation (bachelor thesis).

Extraction Loop

Idea: Periodically extract data from SLURM and look for changes



Extraction Approach

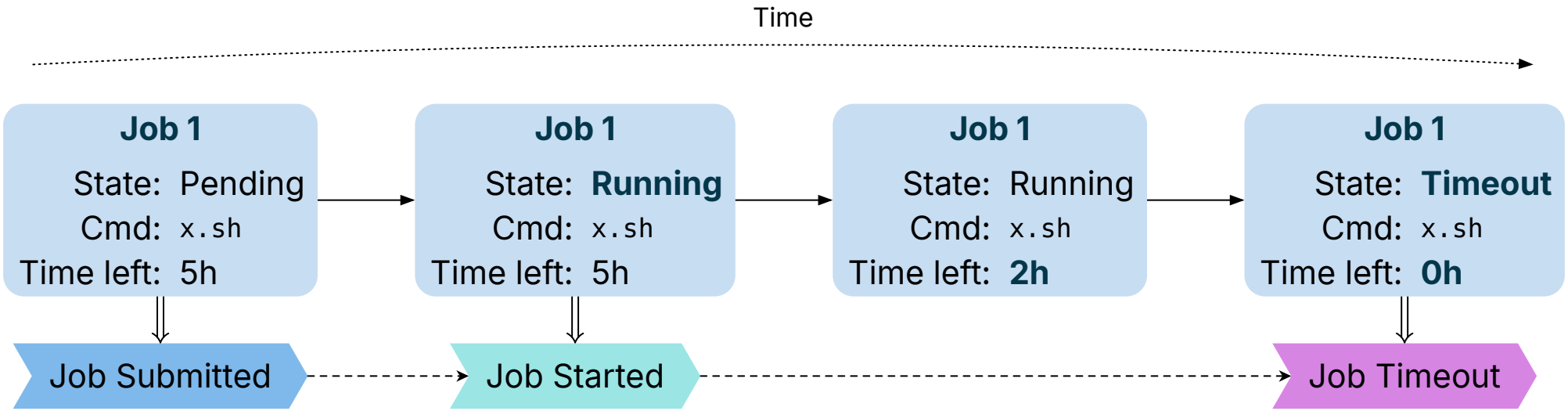
Example: Job state changes

- Get the current job state, command, time left, etc. from SLURM
- Periodically read this state and parse it

Extraction Approach

Example: Job state changes

- Get the current job state, command, time left, etc. from SLURM
- Periodically read this state and parse it



Extraction Approach

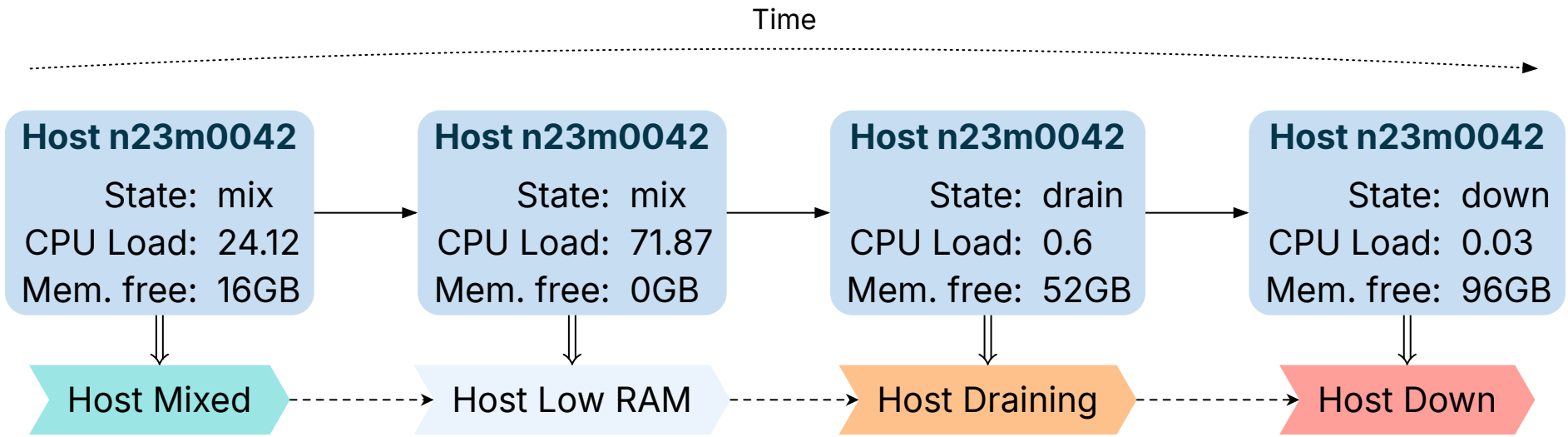
Example: Host state changes

- Get the current host state, CPU and memory usage, etc. from SLURM
- Periodically read this state and parse it

Extraction Approach

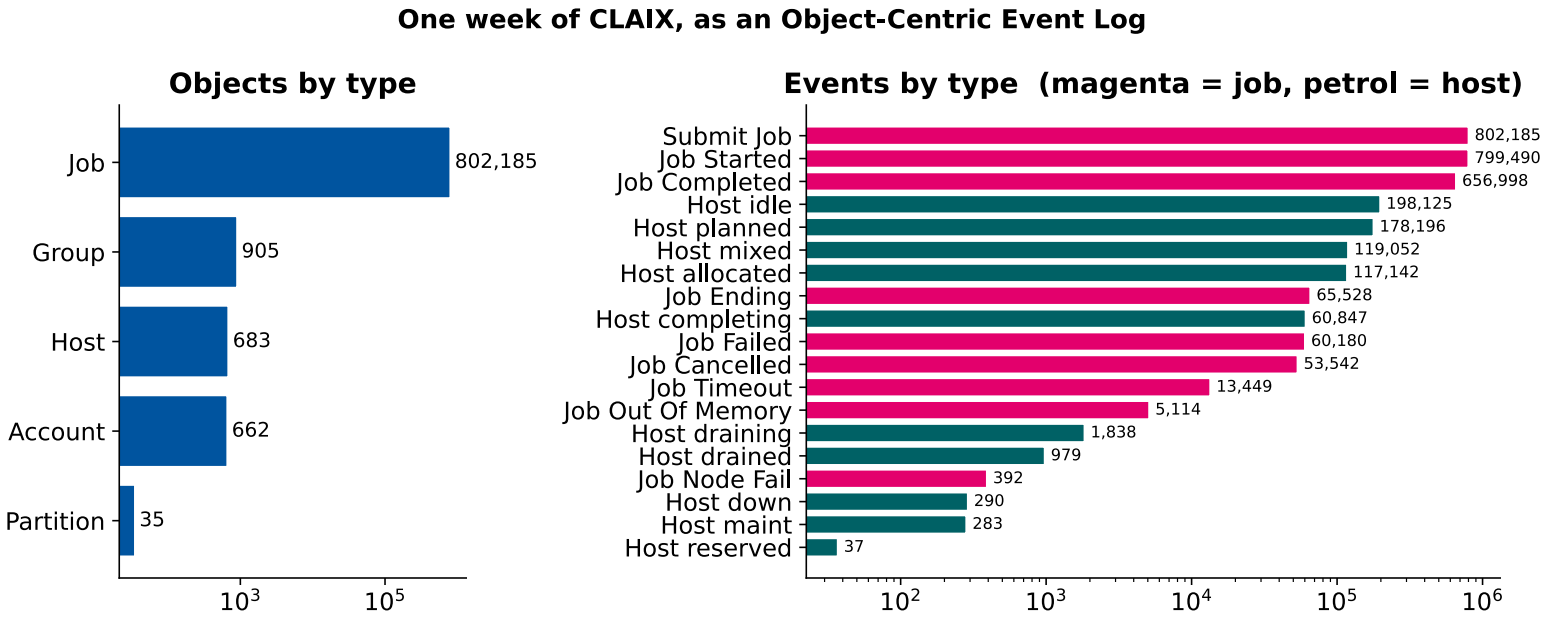
Example: Host state changes

- Get the current host state, CPU and memory usage, etc. from SLURM
- Periodically read this state and parse it



One week of CLAIX, as data

The shape of the dataset



3.13M events across 19 activity types

802k jobs, 905 groups, 683 hosts, 662 accounts, 35 partitions.

Most jobs follow the same trace

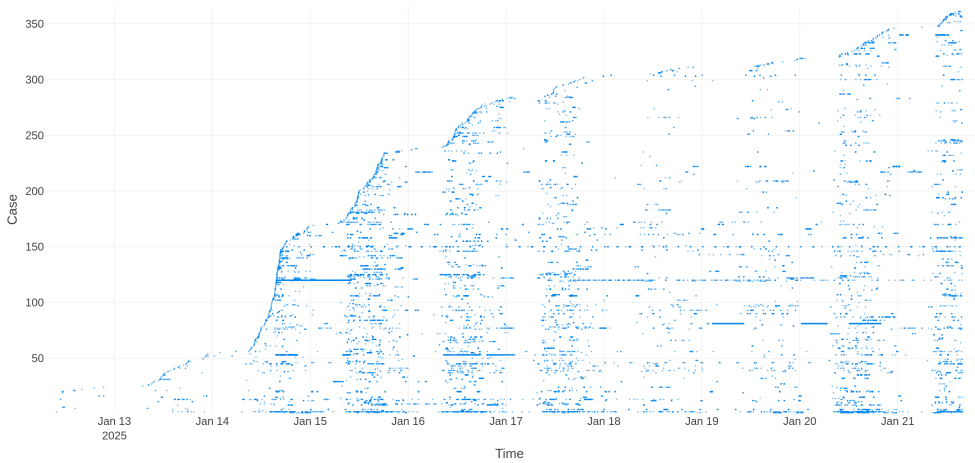
Variant	Count	Percentage
Submit Job → Job Started → Job Compl...	99,439	76.54%
Submit Job → Job Started → Job Failed	13,248	10.20%
Submit Job → Job Started → Job Ending → Job Compl...	4,197	3.23%
Submit Job → Job Started	3,616	2.78%
Submit Job → Job Started → Job Cance...	2,378	1.83%
Submit Job → Job Started → Job Ending → Job Cance...	2,188	1.68%
Submit Job → Job Started → Job Ending → Job Timeout	1,615	1.24%

Variant analysis on the **job** perspective.

The dominant trace is just Submit → Start → Complete.

Same data, different lens

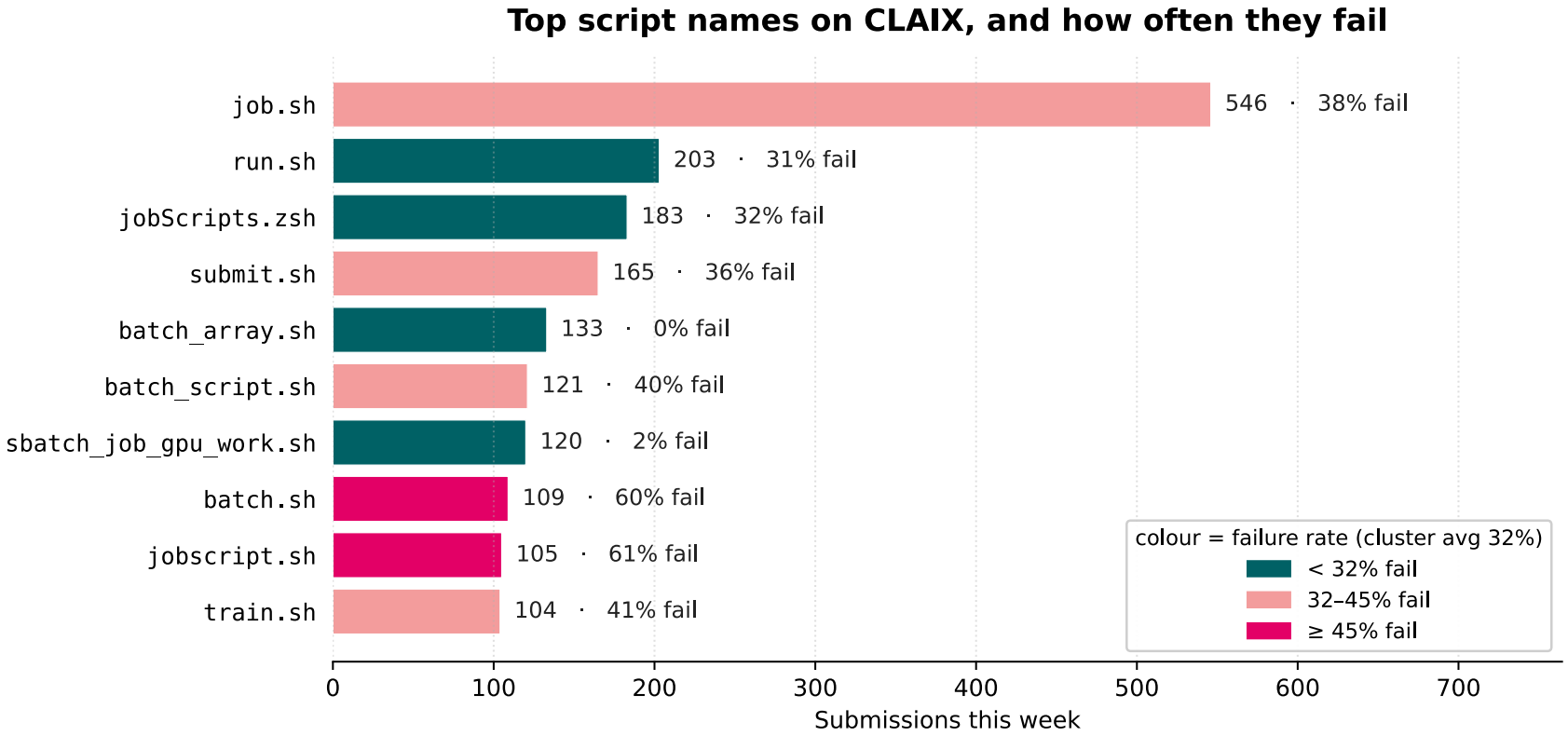
Cases = accounts



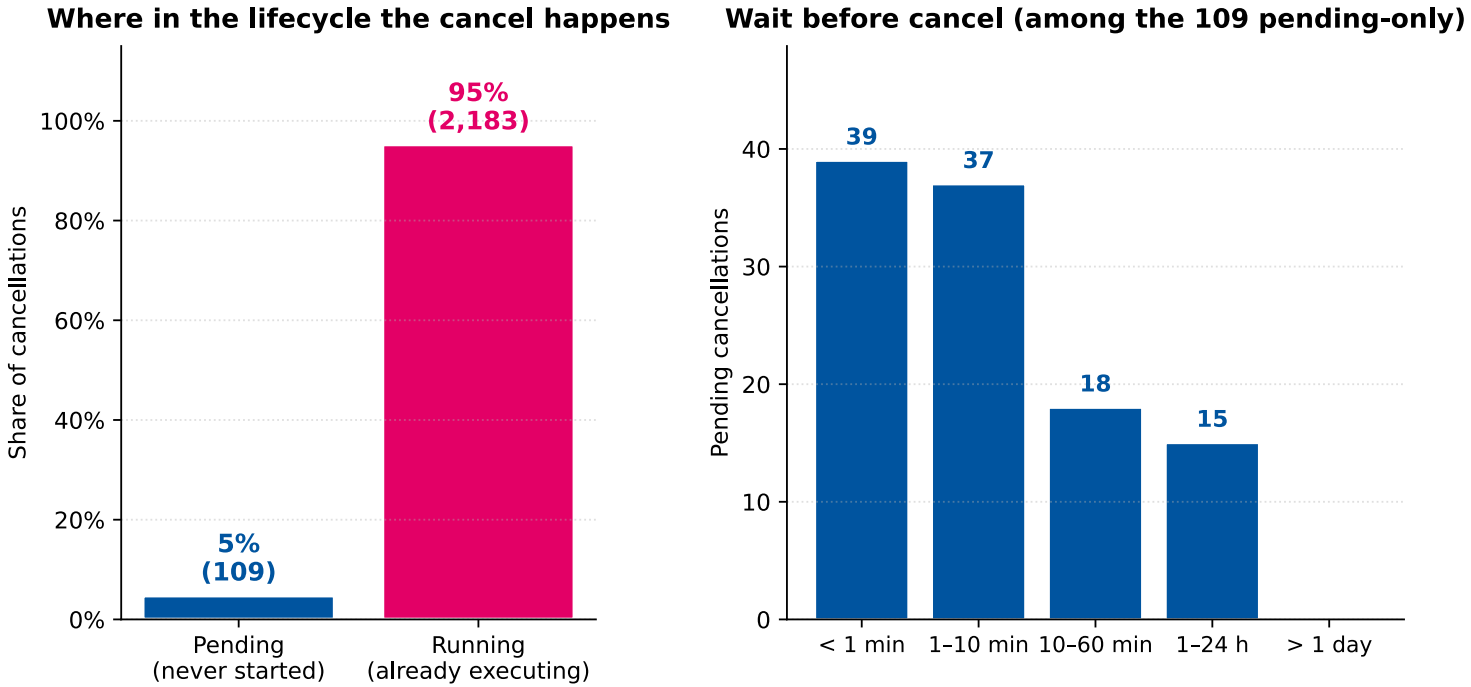
Cases = hosts

A **dotted chart**: one row per case, time on x-axis, one dot per event. Same week, two stories.

What people call their scripts



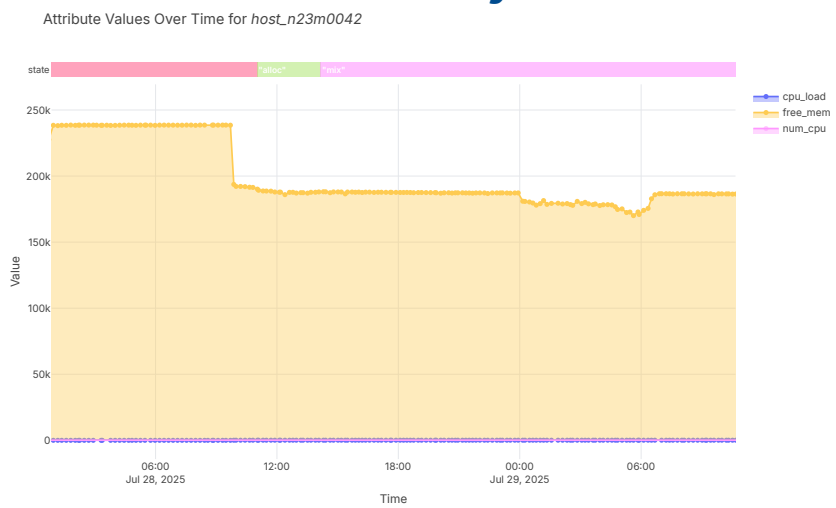
When users actually cancel



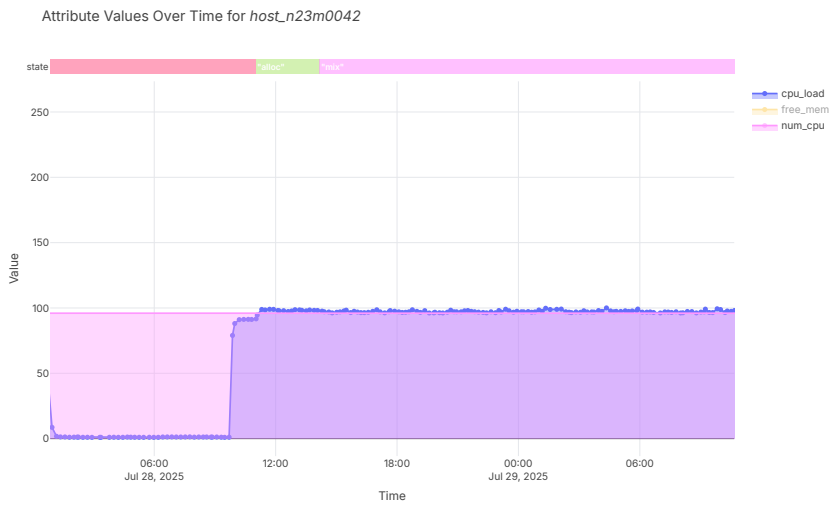
Of 2,292 cancelled jobs, **95% were cancelled after the job had already started running**.

Hosts have a process too

Free memory (MB)



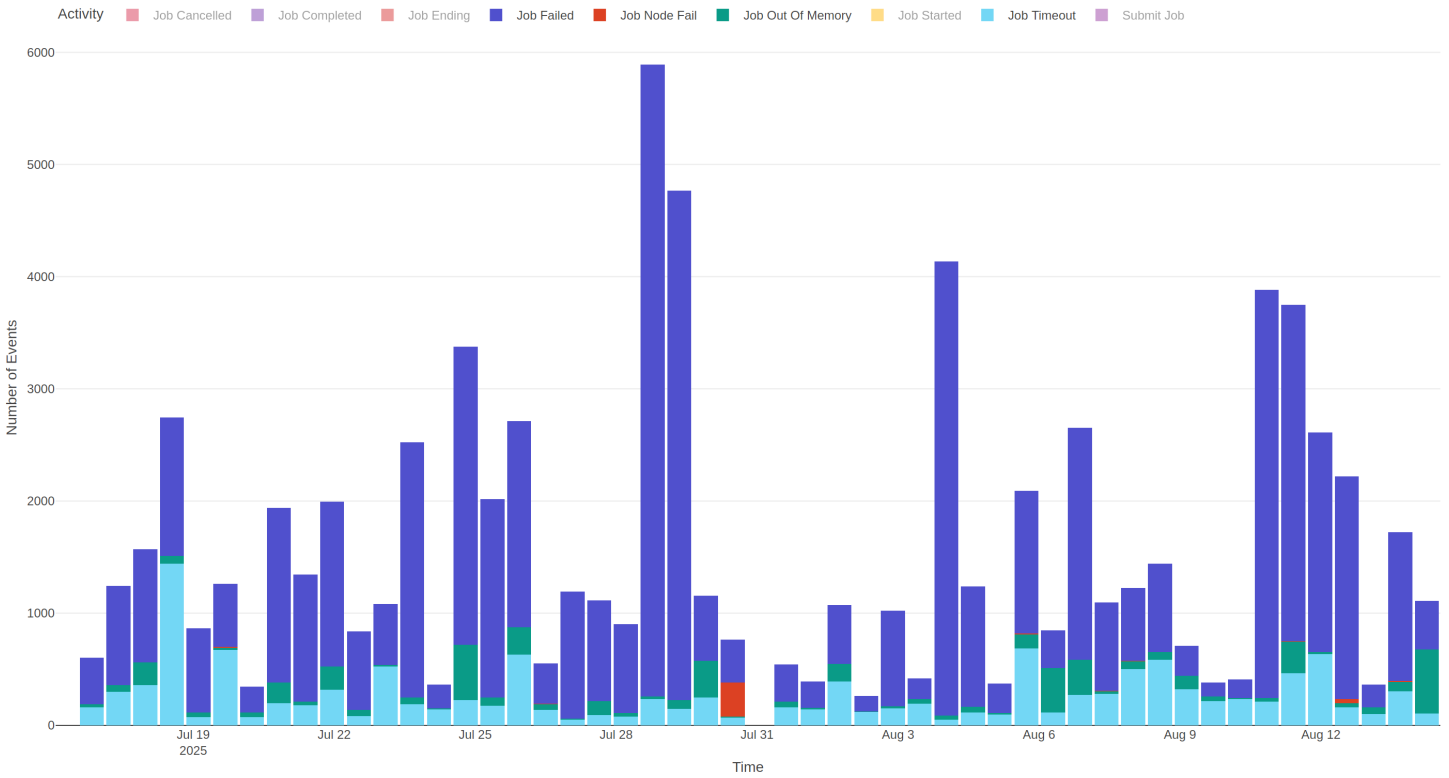
CPU load



Same host (`n23m0042`), 24h. Left: free memory drops in steps as jobs land. Right: CPU load climbs to capacity. Every change is an OCEL event on the **host** object.

One week of CLAIX, as data

Emergency shutdown



Predicting what jobs will do

From event log to predictive model

The task: given everything the cluster has seen so far, predict what this job will do.

10,455 jobs from accounts with ≤ 100 jobs. Two targets: outcome (6-class) and runtime. Submit-time features only.

Account history

- prior jobs / failures / failure rate
- avg. prior runtime
- distinct cmds & dirs so far
- mem / cpu vs user's running avg
- is-first-job flag (no prior jobs)

From event log to predictive model

The task: given everything the cluster has seen so far, predict what this job will do.

10,455 jobs from accounts with ≤ 100 jobs. Two targets: outcome (6-class) and runtime. Submit-time features only.

Account history

- prior jobs / failures / failure rate
- avg. prior runtime
- distinct cmds & dirs so far
- mem / cpu vs user's running avg
- is-first-job flag (no prior jobs)

Submission timing

- Δt since last submission
- Δt since last failure

From event log to predictive model

The task: given everything the cluster has seen so far, predict what this job will do.

10,455 jobs from accounts with ≤ 100 jobs. Two targets: outcome (6-class) and runtime. Submit-time features only.

Account history

- prior jobs / failures / failure rate
- avg. prior runtime
- distinct cmds & dirs so far
- mem / cpu vs user's running avg
- is-first-job flag (no prior jobs)

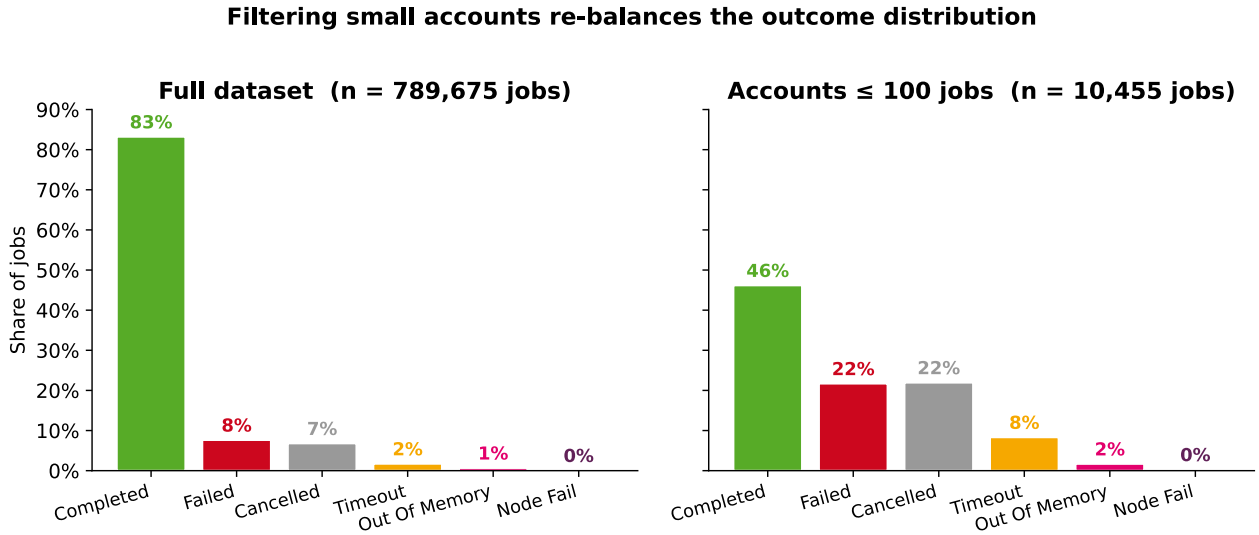
Submission timing

- Δt since last submission
- Δt since last failure

Job static

- `cpus`, `min_memory`
- mem-per-CPU, partition
- hour, weekday
- script ext, work-dir info

Filtering accounts

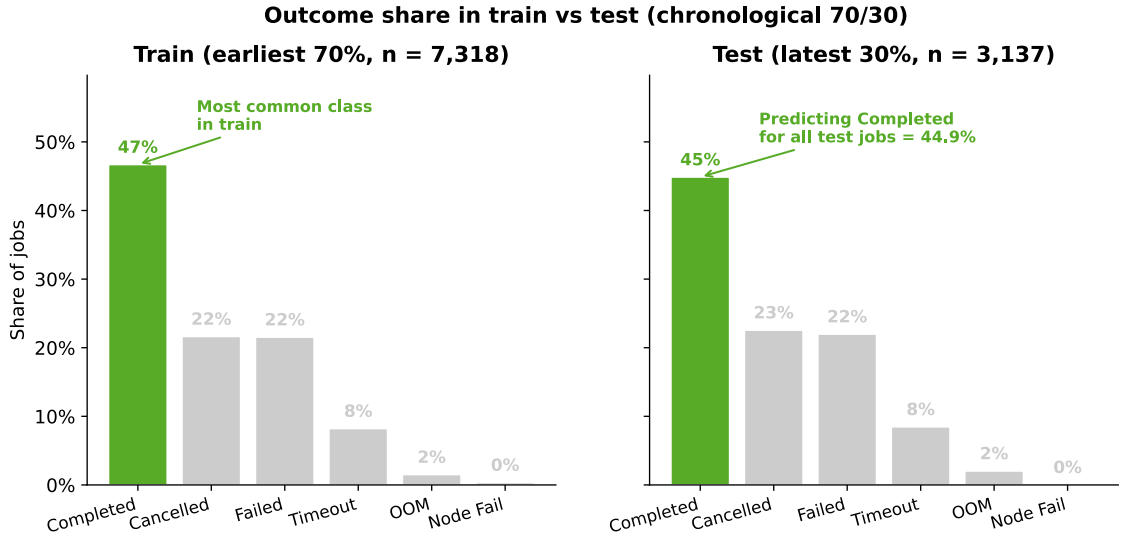


The full dataset is 83% Completed, some accounts have thousands of jobs.
To filter out organizational noise, automated scripts, and outliers, we focus on the 466 accounts with $\leq$ 100 jobs (amounting to 10,455 jobs).

Evaluating: train on the past, test on the future

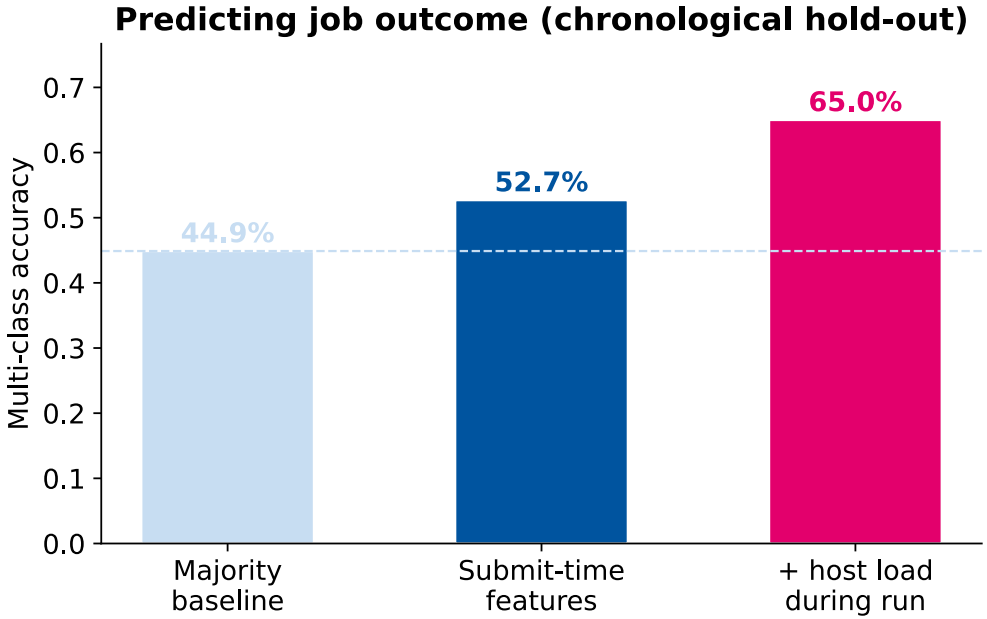
Sort by `submit_time`, train on the earliest 70% (up to Aug 5), test on the latest 30%.
No shuffle: a random split would let the model peek at a user's future.

Outcome share in train vs test



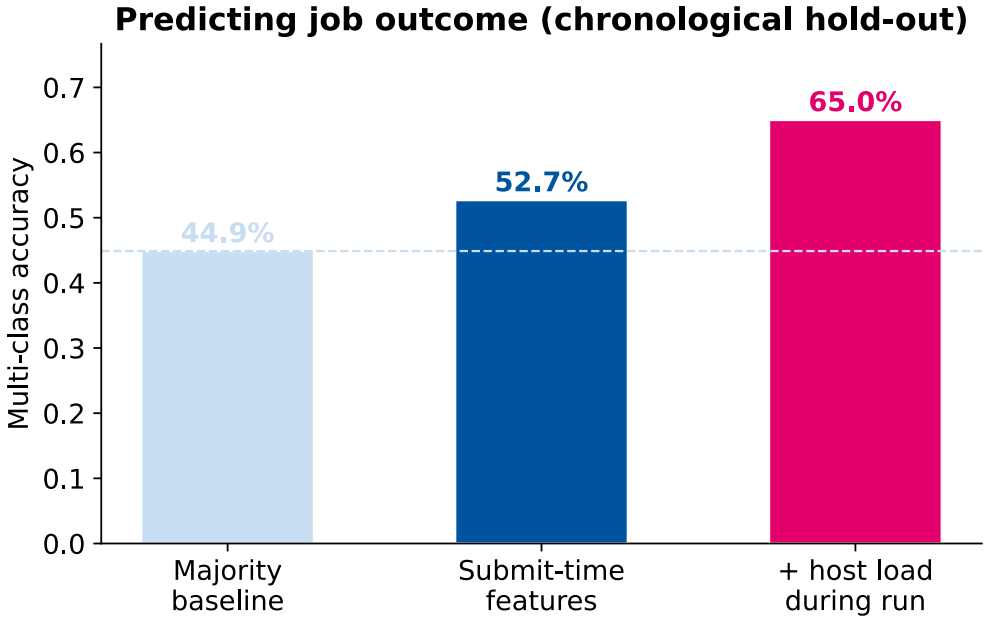
Completed dominates train, so it is the natural majority-vote baseline.
Always predicting *Completed* yields an accuracy of **44.9%** for the test window.

Predicting the outcome at submit time



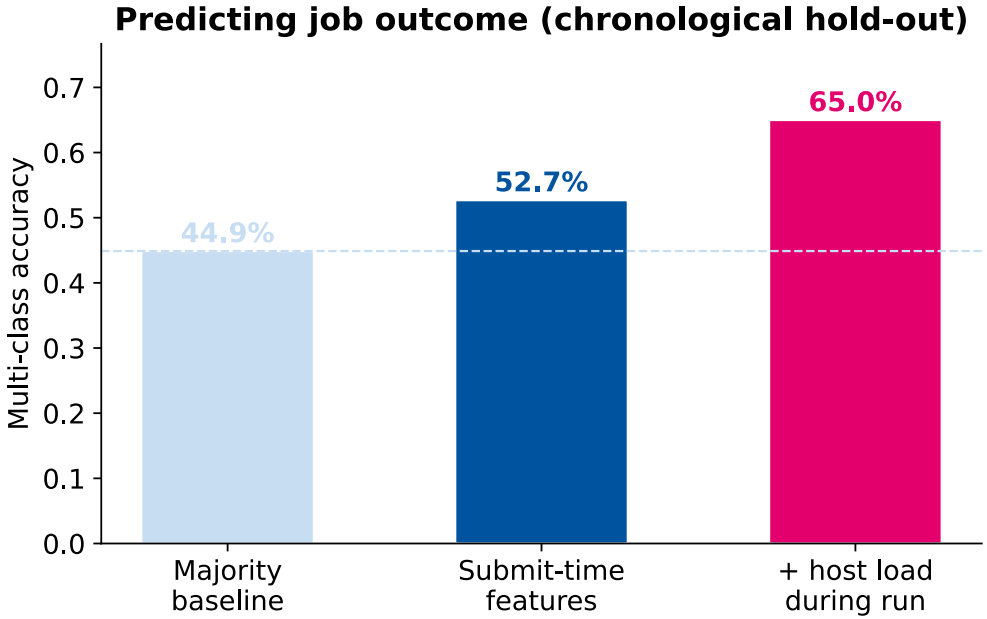
- **Baseline 44.9%**: the share of Completed jobs in the test window (Always guess Completed).

Predicting the outcome at submit time



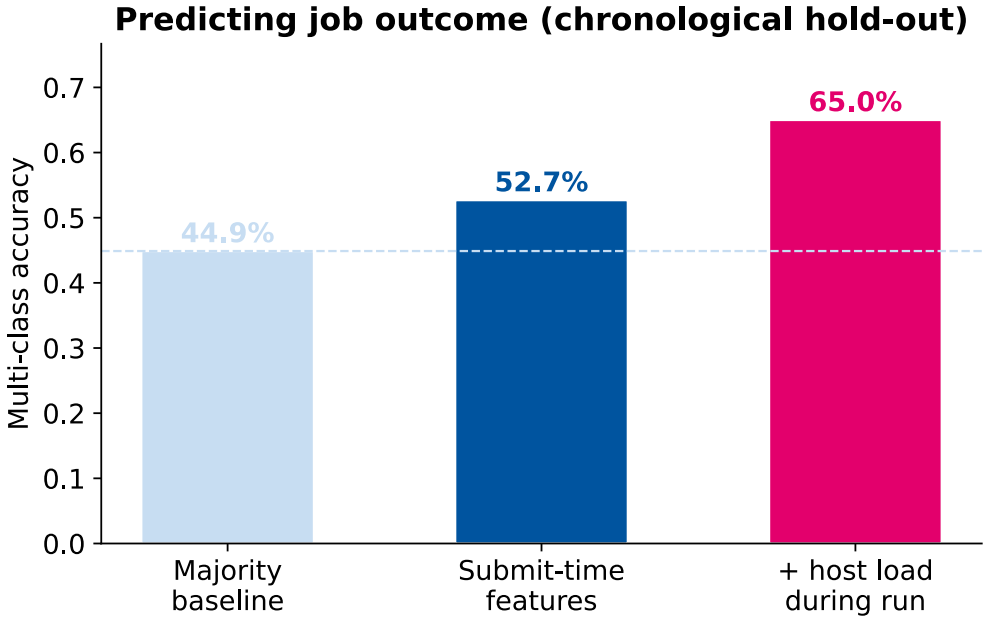
- **Baseline 44.9%**: the share of Completed jobs in the test window (Always guess Completed).
- **Submit-time features**: ∼+8 pp. What the scheduler sees at `sbatch` time.

Predicting the outcome at submit time



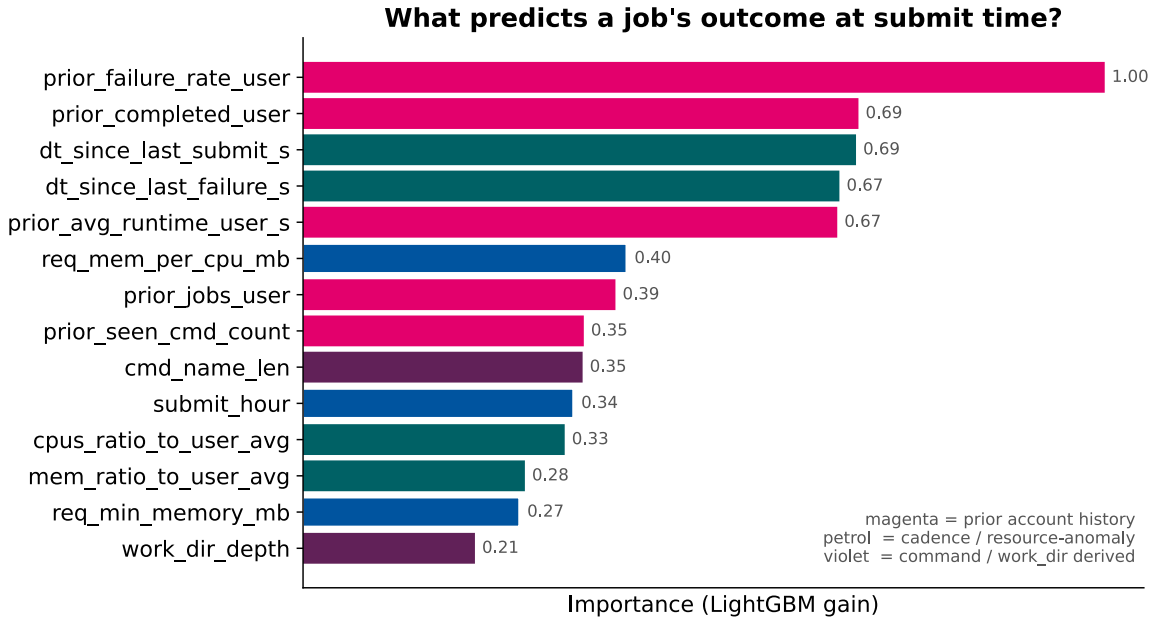
- **Baseline 44.9%**: the share of Completed jobs in the test window (Always guess Completed).
- **Submit-time features**: ∼+8 pp. What the scheduler sees at `sbatch` time.
- **+ host load during run**: avg/peak CPU load and avg/min free memory sampled on the node. Another ∼+12 pp. Post-hoc.

Predicting the outcome at submit time



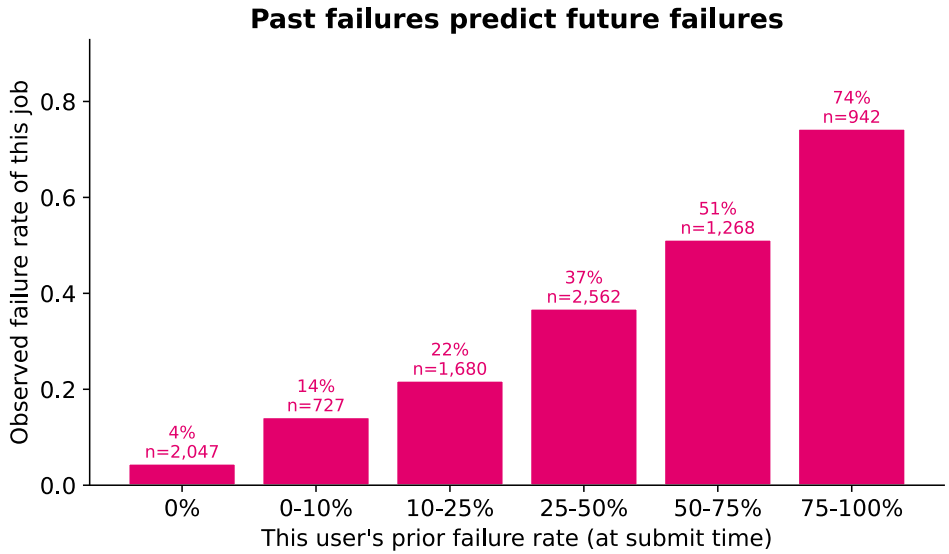
- **Baseline 44.9%**: the share of Completed jobs in the test window (Always guess Completed).
- **Submit-time features**: ∼+8 pp. What the scheduler sees at `sbatch` time.
- **+ host load during run**: avg/peak CPU load and avg/min free memory sampled on the node. Another ∼+12 pp. Post-hoc.
- **Binary "will it fail at all":** 70%.

What the model leans on



Top predictors are based on **this user's prior jobs** (e.g., previous failure rate).
Requested resources rank lower.

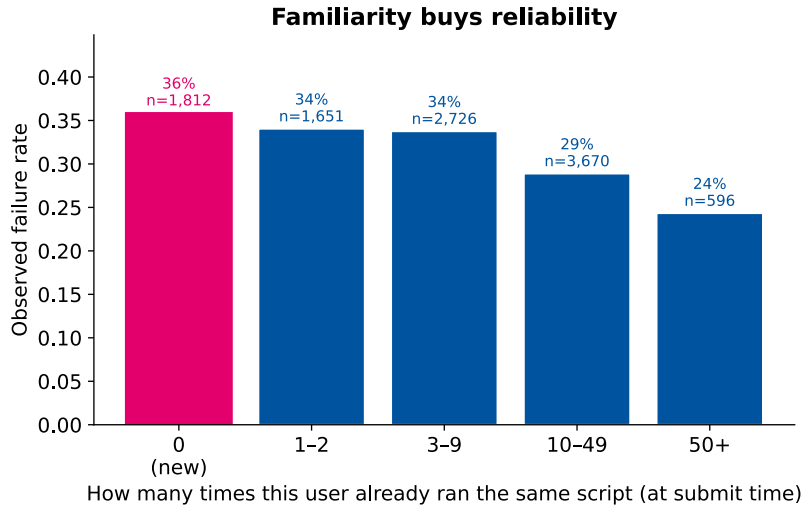
Past failures predict future failures



Each bar groups jobs by the user's running failure rate at submit time.
The ≥75% bucket fails 74% of the time vs 4.5% for the cleanest users, a 16× lift.

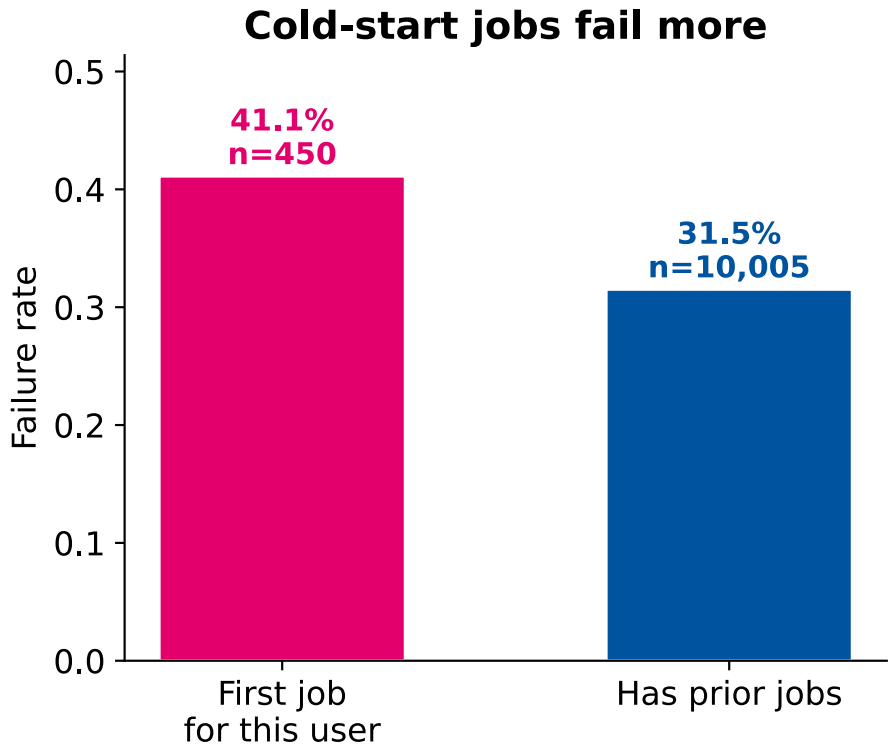
Predicting what jobs will do

Familiarity buys reliability



Bucketed by how often this user previously submitted a script with the **same filename**. First-time scripts fail 36%, scripts seen ≥50 times fail 24%. About 1.5× improvement.

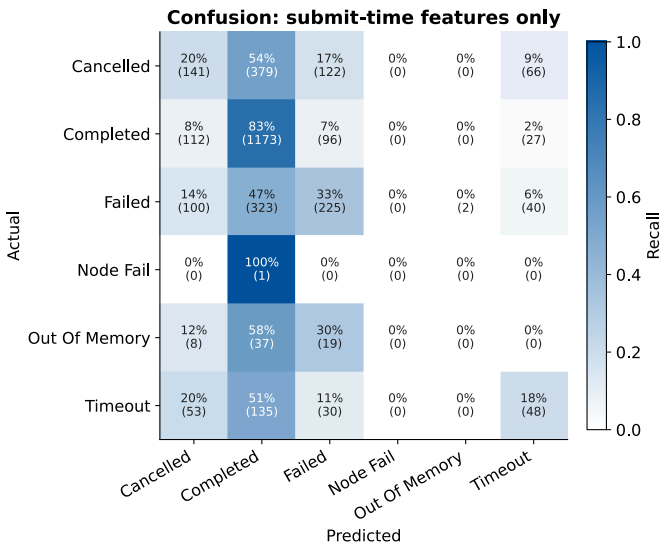
Cold start: the first job is the riskiest



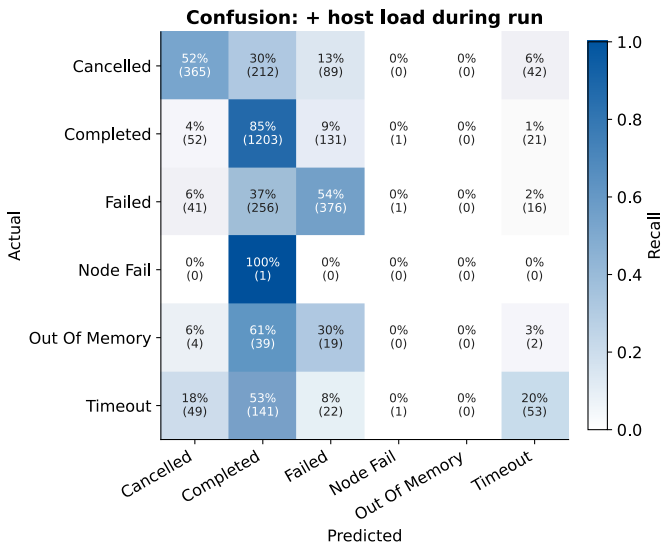
- A user's **first job** in this dataset fails 41% of the time, vs 32% for later jobs.
- Operationally actionable: new accounts deserve extra attention.
- Same effect for **first time on a partition** and **first time with a new script**.

Where the model gets confused

Submit-time features only



+ host load during run



Cancelled vs Completed is the persistent confusion. OOM and Node Fail are too rare to learn well.

Time of day

Hypothesis: users submitting late at night make more mistakes. Their jobs fail more often.

Time of day

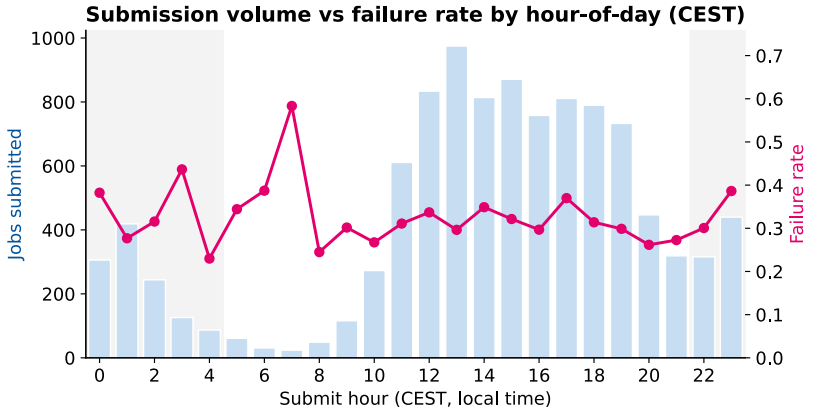
Hypothesis: users submitting late at night make more mistakes. Their jobs fail more often.

Test: bucket every submission by its local hour (CEST). Plot volume and failure rate.

Time of day

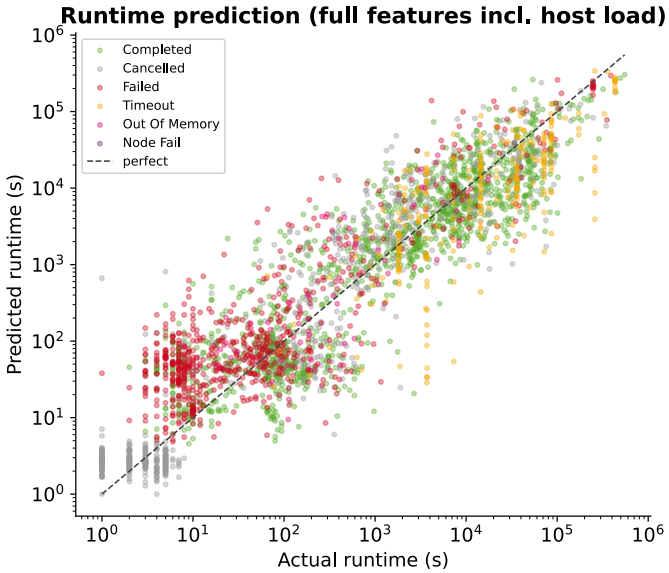
Hypothesis: users submitting late at night make more mistakes. Their jobs fail more often.

Test: bucket every submission by its local hour (CEST). Plot volume and failure rate.



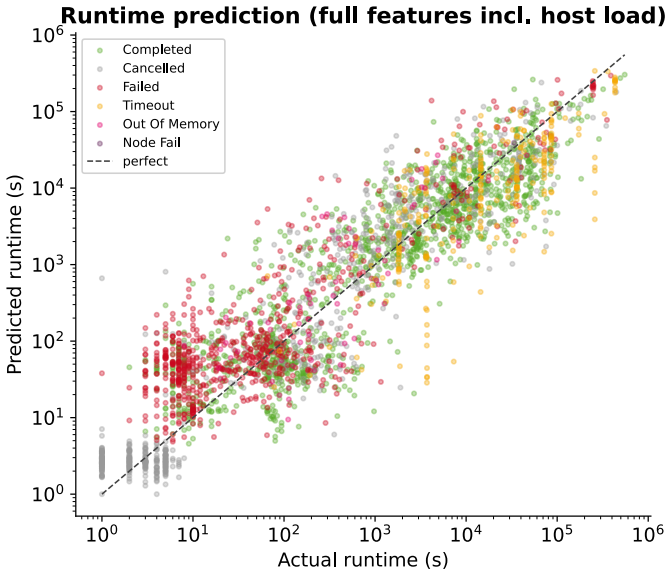
Result: hypothesis not supported. Late night (22-04 CEST, shaded): 33.5% fail. Daytime (08-20 CEST): 31.6% fail. Difference 1.9 pp, within sampling noise.

Predicting **runtime** is much harder



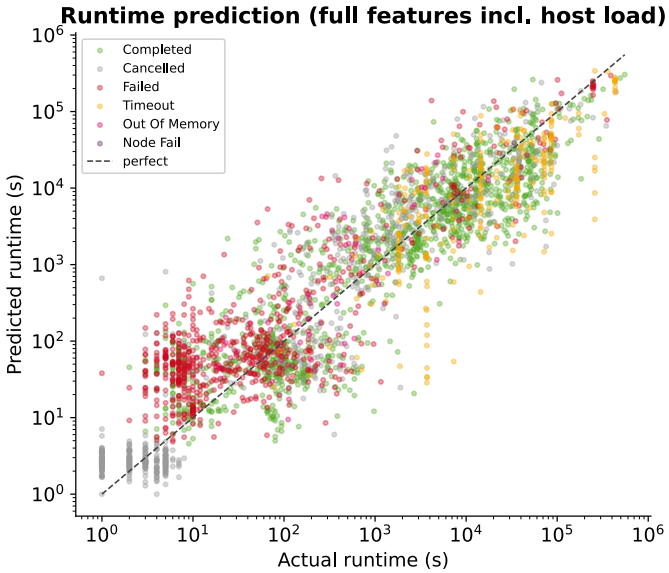
Submit-time only: $R^2 \approx 0$. Runtimes span up to 5 orders of magnitude per user (median: 4).

Predicting runtime is much harder



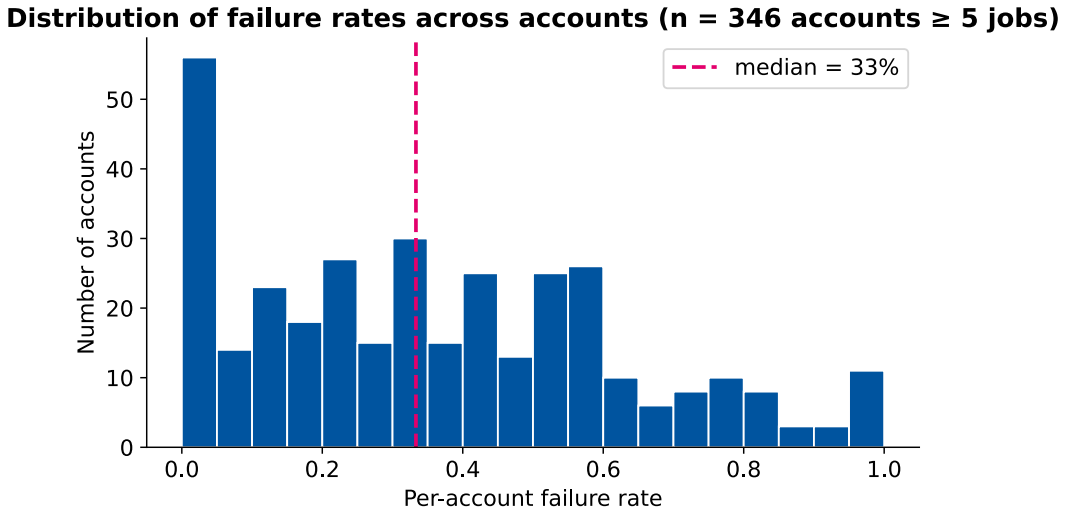
Submit-time only: $R^2 \approx 0$. Runtimes span up to 5 orders of magnitude per user (median: 4). **+ host load:** $R^2 \approx 0.84$ on the log scale. Once we observe the node, we can extrapolate.

Predicting **runtime** is much harder



Submit-time only: $R^2 \approx 0$. Runtimes span up to 5 orders of magnitude per user (median: 4). **+ host load:** $R^2 \approx 0.84$ on the log scale. Once we observe the node, we can extrapolate. **So:** a-priori runtime estimation does not work, but **online** estimation does.

Users are not average



Per-account failure rates are bimodal: many users rarely fail, many fail constantly. A global average sits between the modes and describes neither, **so personalised features pay off**.

Jobs share hosts: ask about the neighbours

A job's *neighbours* on host h: other jobs that recently ran on h or are still running when this one starts.

- 657 hosts. 1 to 14 jobs run concurrently per host.
- 57% of hosts saw $\geq$2 concurrent jobs at peak. Top host: 56 users.
- 19% of jobs have an immediate-overlap neighbour on the same host.
- So we can build features over a job's neighbours, not just its user.

Jobs share hosts: ask about the neighbours

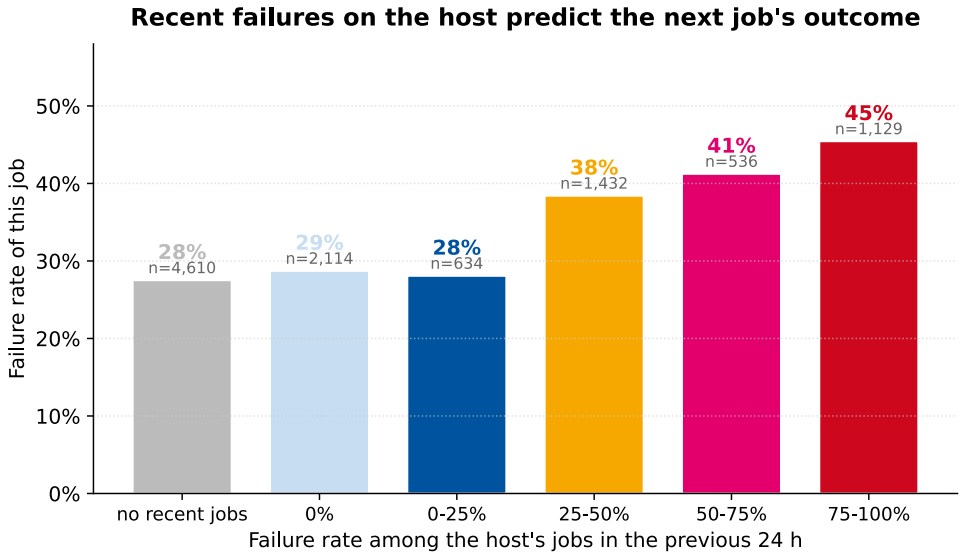
A job's *neighbours* on host h: other jobs that recently ran on h or are still running when this one starts.

- 657 hosts. 1 to 14 jobs run concurrently per host.
- 57% of hosts saw $\geq$2 concurrent jobs at peak. Top host: 56 users.
- 19% of jobs have an immediate-overlap neighbour on the same host.
- So we can build features over a job's neighbours, not just its user.

New per-job features (submit time, once host is assigned):

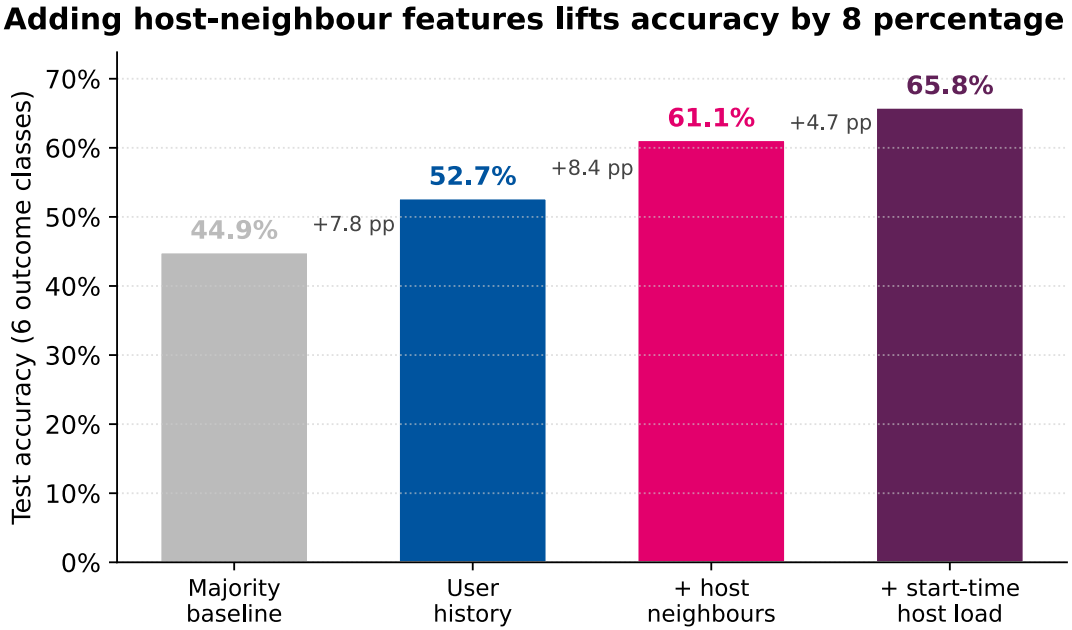
- `host_jobs_24h_before` → neighbours that ended on host in last 24h
- `host_failure_rate_24h` → fraction of those neighbours that failed
- `host_concurrent_at_submit` → neighbours still running now
- `host_dt_since_last_fail_s` → seconds since the last burn on host

Recent host failures predict the next failure



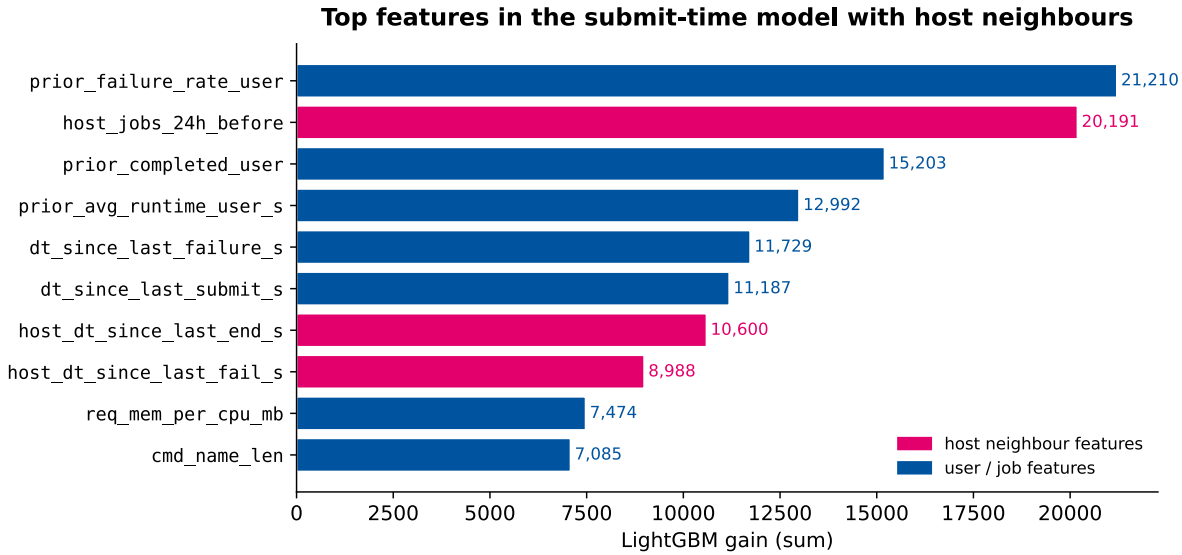
Host's last 24h ≥75% failure: next job fails 45%. Clean last 24h: 29%. Pure concurrency does not move the rate, recent outcomes do.

Adding host neighbours lifts accuracy by 8 points



- User history alone: 52.7%.
- Add host's recent neighbours: **61.1%** (+8.4 pp).
- Add host load during the run: 65.8%.
- Neighbour features need only the **host assignment**.

Top features, after adding host neighbours



`host_jobs_24h_before` ranks 2 by gain, just behind `prior_failure_rate_user`. Three of the top eight features are host-neighbour signals.

Many jobs wait longer than they run

Of 9,853 jobs with both queue + run:

- **33%** waited ≥ as long as they ran
- **16%** waited 10× longer
- **8%** waited 100× longer

Many jobs wait longer than they run

Of 9,853 jobs with both queue + run:

- **33%** waited ≥ as long as they ran
- **16%** waited 10× longer
- **8%** waited 100× longer

Across the wait-heavy jobs:
3,650 days in queue,
128 days of compute.
A **28**× ratio of queue to work.

Where this is going

Two audiences, two payoffs

For operators

- Spot users / hosts drifting toward failure **before** they cost capacity
- Predicted outcome as a scheduling signal
- Anomaly detection: shutdowns, drained nodes, runaway loads
- Abuse detection: crypto miners, DoS, …
- Combination with support tickets and user feedback

Two audiences, two payoffs

For operators

- Spot users / hosts drifting toward failure **before** they cost capacity
- Predicted outcome as a scheduling signal
- Anomaly detection: shutdowns, drained nodes, runaway loads
- Abuse detection: crypto miners, DoS, …
- Combination with support tickets and user feedback

For users

- Personalised “is this job likely to fail?” hint at submission
- Runtime estimates that sharpen as the job runs
- Anomaly nudges in their own session (“this is the kind of script that fails”)

Open questions

- **SLURM-perceived outcome ≠ user-perceived outcome**
  - Return code of a job might be zero, but the user's script might not have done what they wanted.
  - Users let their jobs run out of time and then pick up from a checkpoint. SLURM sees a timeout, users see a success.
- **More signal for free.** Extract `--time`, `--nodes`, GPU flags, dependency chains.
- **Online prediction.** Stream state deltas through the model in real time.
- **Object-centric features.** Connections between jobs, hosts, accounts, and users.

Thanks!

Questions, ideas, or HPC datasets to look at next?