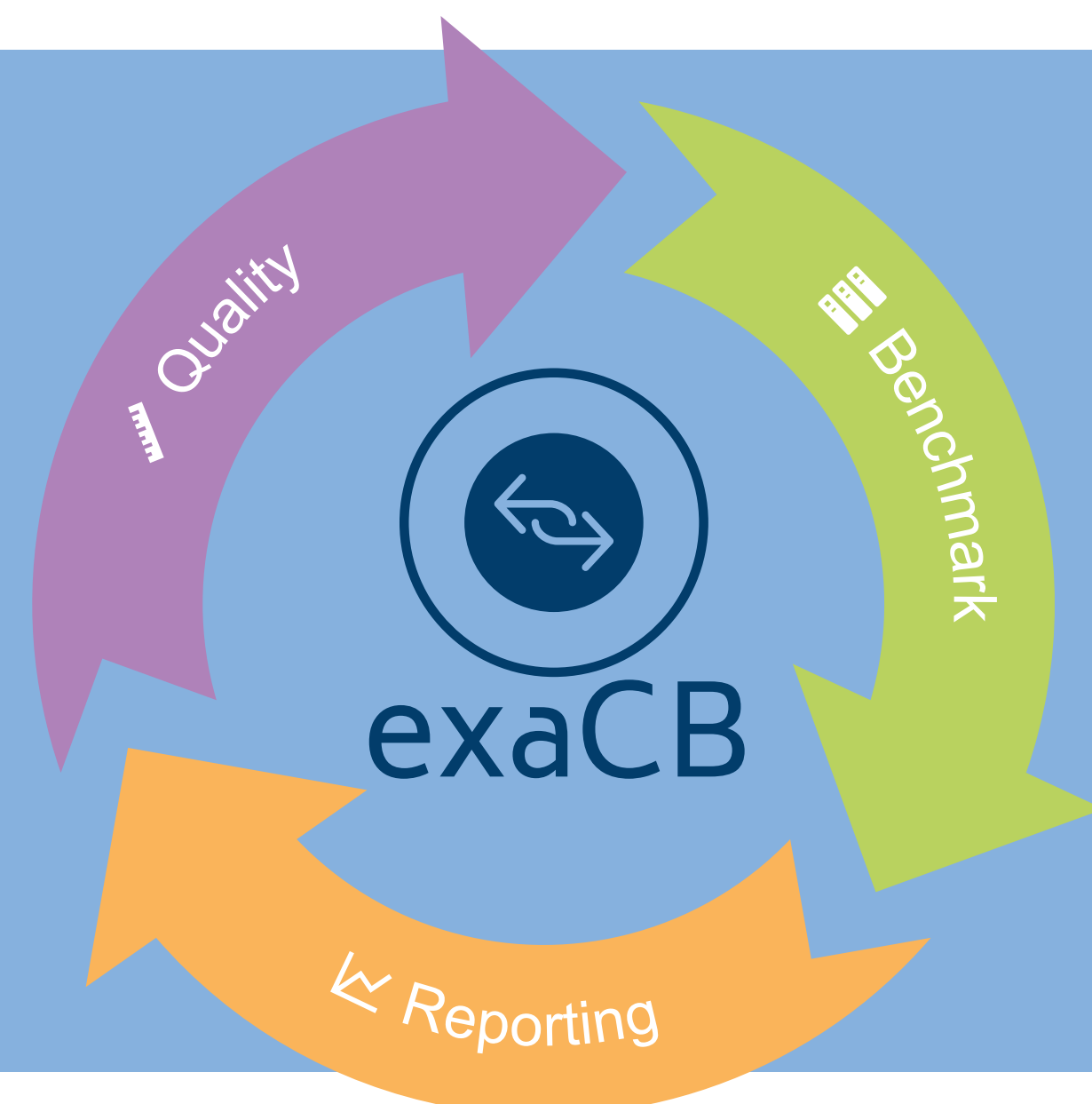


A BENCHMARKER'S JOURNEY

AN INTEGRATED APPROACH FROM ONBOARDING TO BENCHMARK VISUALIZATION

Michał Paweł Rajski · Filipe Souza Mendes Guimarães · Jayesh Badwaik · Andreas Herten
Jülich Supercomputing Centre, Forschungszentrum Jülich

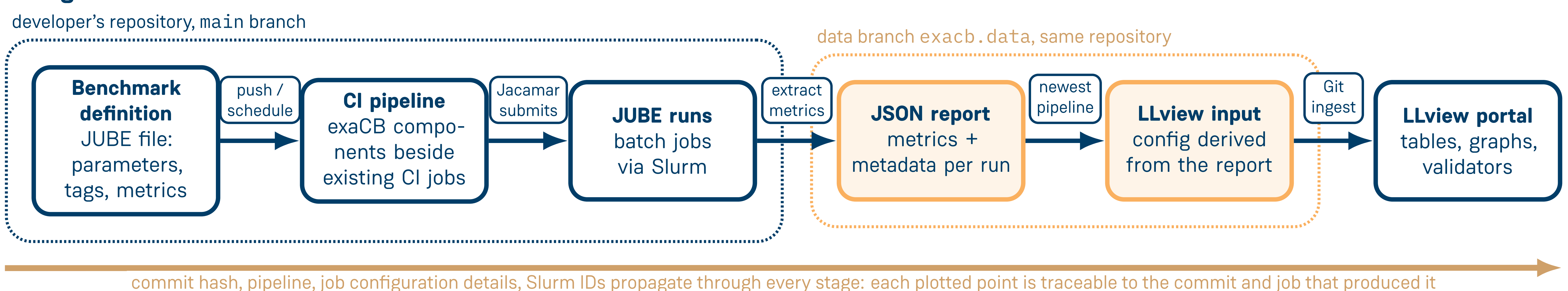


Why continuous benchmarking?

Benchmarking is a recurring workflow: integrate, execute, measure, collect, visualise. In practice, this process is fragmented and manual, and every newly onboarded application pays the same manual cost again. Our solution builds on extending good software engineering practices to performance-critical execution by considering following:

- Benchmarks live in the developer's main repository
- Easy performance measurement via known CI/CD interface
- Every result is machine-readable and traceable to commit and job
- Dashboards are derived from the results, not written by hand
- Adoption is incremental: runnability → instrumentation → reproducibility

The integrated workflow



Benchmarking inside your CI exaCB

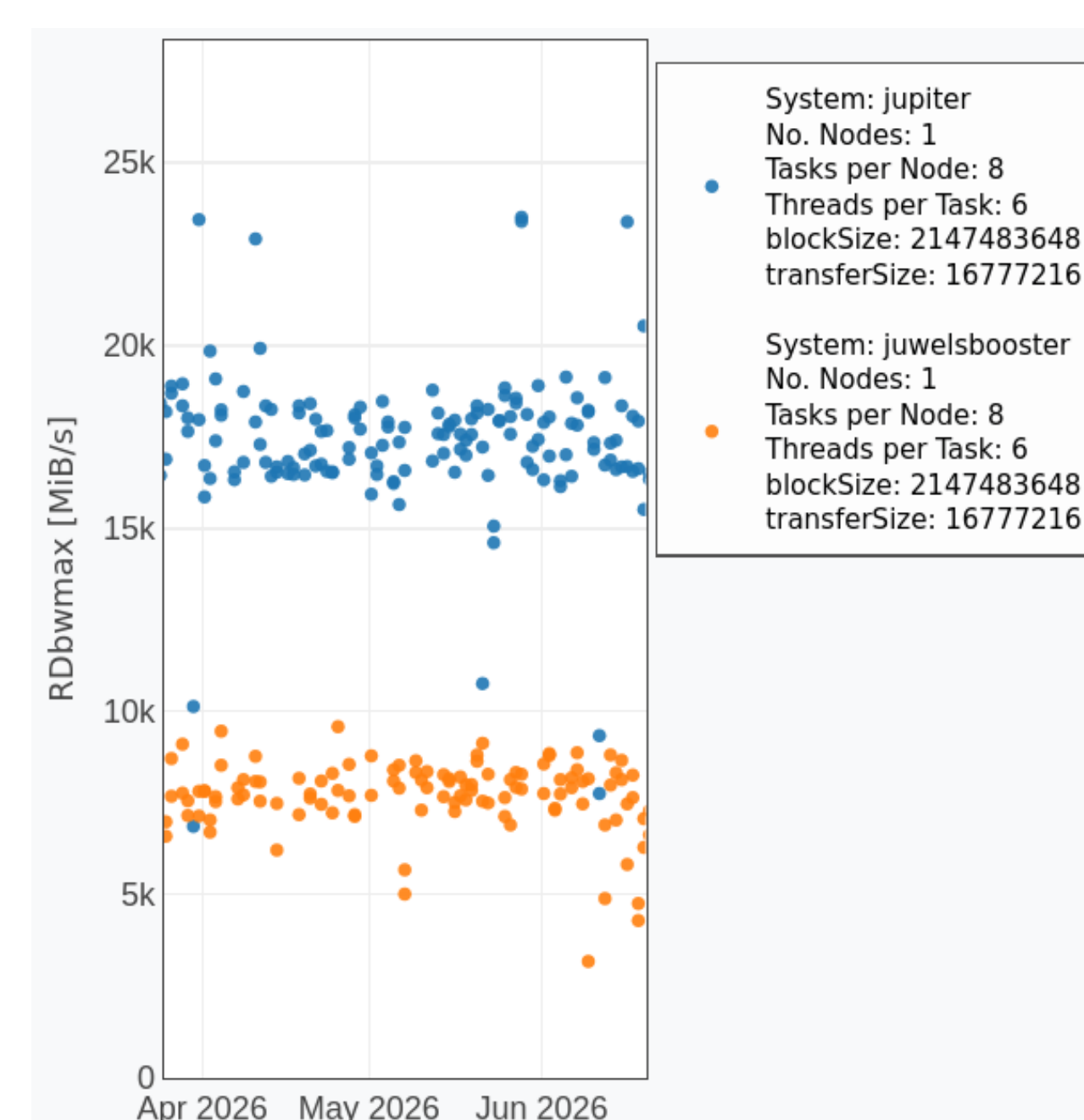
- Reusable GitLab CI/CD components orchestrate **execution, post-processing and feature injection**
- Shared **protocol**: every run yields a self-describing JSON report of the same shape, so results compare across applications and systems is straightforward
- **Harness-agnostic**: JUBE² today, Re-Frame and Ramble potential extension
- **Ready-made analyses**: machine comparison, scalability, time series; possibility of injecting commands without touching the benchmark
- Proven at scale in **JUREAP** (JUPITER early access): more than 70 applications

```
include:
- component: <exacb>/execution@v3
  inputs:
    prefix:
      ↪ "babelstream.jupiter.single"
    jube_file: "default.yml"
    usecase: "CUDA"
    variant: "single"
    machine: "jupiter"
    queue: "booster"
    project: "exalab"
    budget: "benchmark"
```

Minimal pipeline for the BabelStream benchmark: one component, a few inputs.

Declare the parameter space JUBE

- One YAML/XML file declares the **parameter space**: node counts, tasks per node, problem sizes, compilers, modules
- JUBE expands it, generates and submits the batch jobs, waits for them, and extracts the metrics from the output
- **Tags** select system and variant: the same file runs on JUWELS, JURECA and JUPITER



Same recipe, two systems: IOR⁵ write bandwidth on JUPITER and JUWELS Booster

Reproducibility

- Definition, execution recipe, results and visualisation configuration are versioned in **one repository**: checking out the recorded commit re-runs the exact experiment
- System state (firmware, hardware issues, congestion) is outside the recipe: an identical recipe with a different result isolates the cause to the system side

References

- [1] J. Badwaik, M. Bode, M. Rajski, A. Herten: *exaCB: Reproducible Continuous Benchmark Collections at Scale Leveraging an Incremental Approach*, arXiv:2603.22251, 2026.
- [2] T. Breuer, J. Wellmann, F. Souza Mendes Guimarães, C. Himmels, S. Lührs: *JUBE*, software, doi:10.5281/zenodo.11394333, apps.fz-juelich.de/jsc/jube.
- [3] Y. Müller, F. Souza Mendes Guimarães, C. Karbach, W. Frings: *LLview*, software, doi:10.5281/zenodo.10221407, github.com/FZJ-JSC/LLview.

Dashboards derived from the report LLview

One component, one required input to start: the prefix of the exaCB¹ report. Everything else is derived from it: metric names and units from the headers, timestamps from the result directories. Optional inputs refine selection, presentation and validation.

```
- component: <exacb>/llview@v1
  inputs:
    prefix:
      ↪ "babelstream.jupiter.single"
```

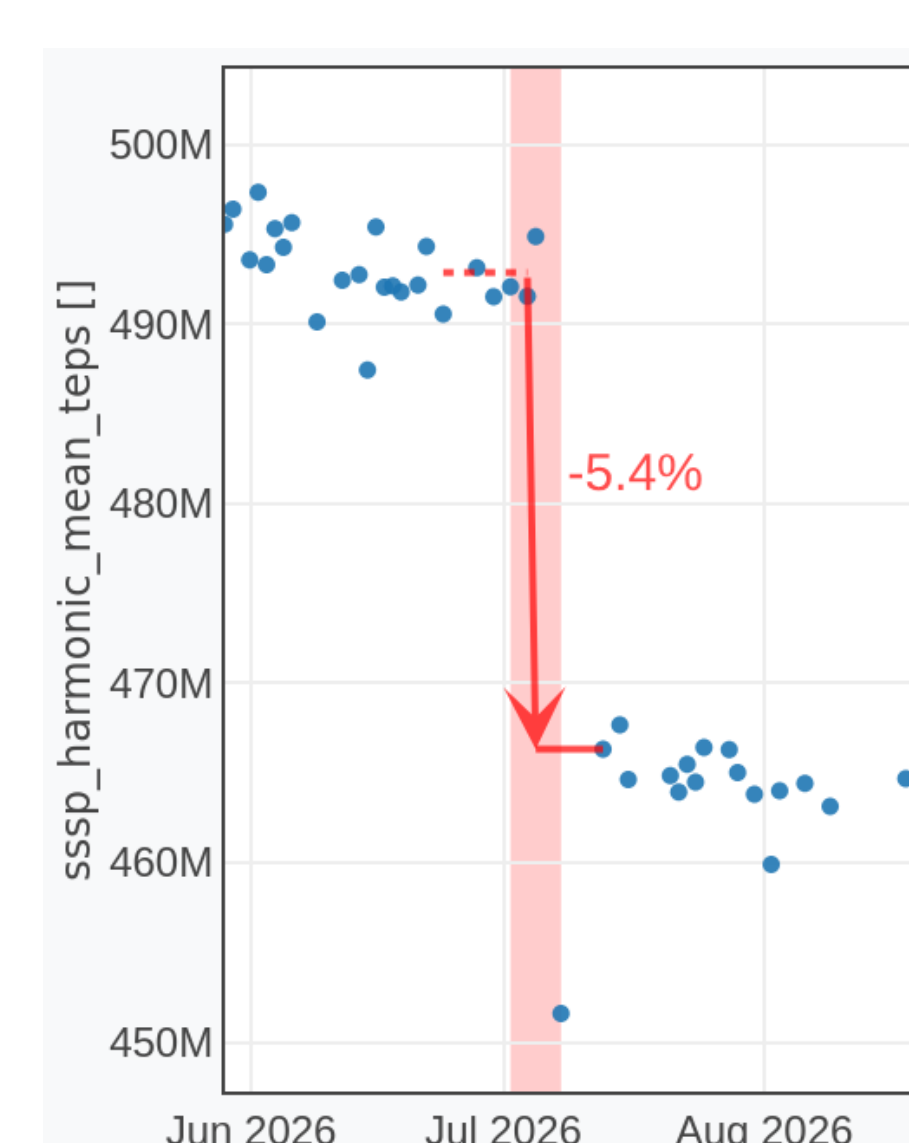
Complete LLview³ job for BabelStream: the prefix names the report to plot.

Validators turn the dashboard from a record into a check

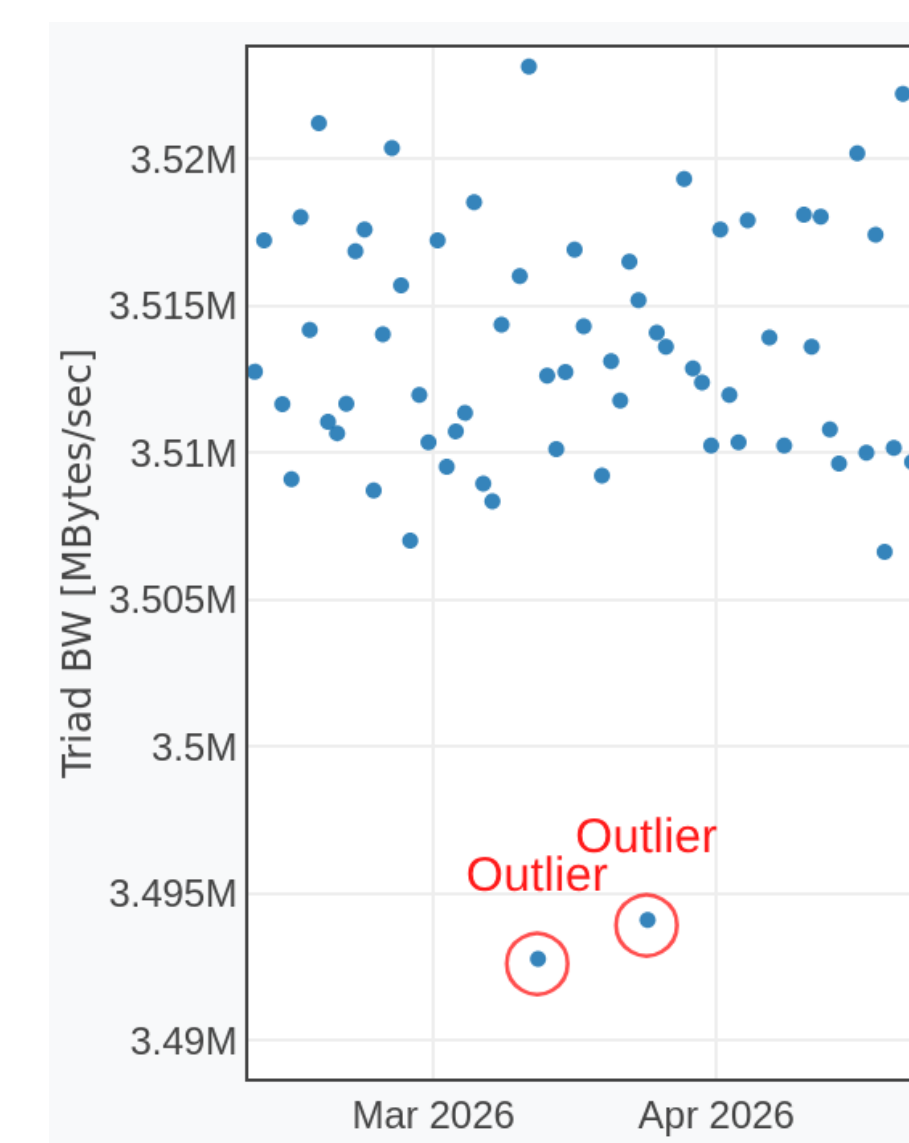
Regression sustained shift against the curve's own history

Outlier single run departing from a rolling window, above a noise floor

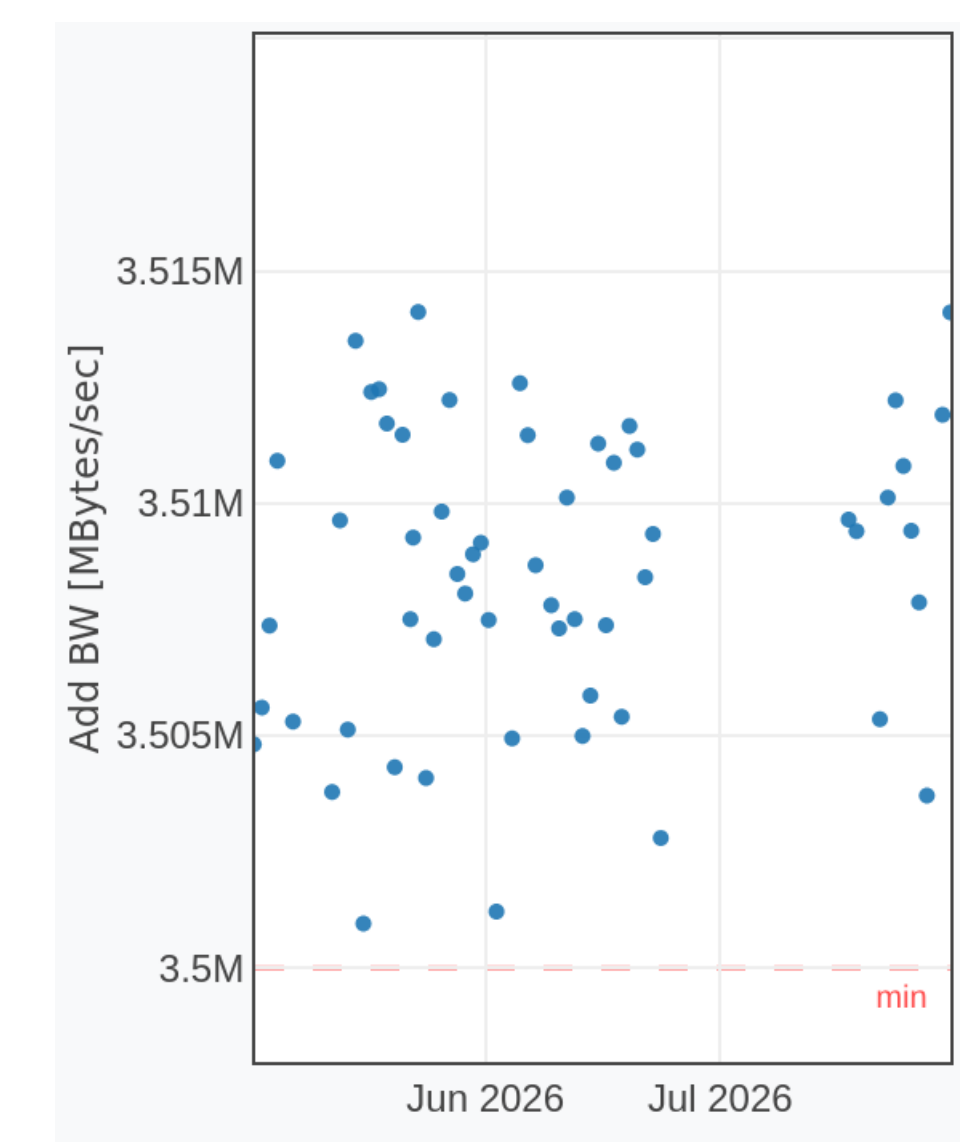
Range static min/max bounds, meaningful from the first run



Regression: Graph500⁶ SSSP, -5.4 % sustained



Outlier: BabelStream⁴ Triad, single runs flagged



Range: BabelStream⁴ Add against a static min

Three lessons that generalise beyond our setting

1. A low entry bar is what makes a workflow effective: onboarding effort is small, simple smoke test with runnable code can be easily implemented
2. Deriving the visualisation from the measurement is removed; the book-keeping effort is low and no requirement of hand-maintained data repositories
3. Git as the transport for definitions, results and visualisation makes reproducibility a property of the design, not an enforced requirement

- [4] T. Deakin, J. Price, M. Martineau, S. McIntosh-Smith: *Evaluating attainable memory bandwidth of parallel programming models via BabelStream*, Int. J. Comput. Sci. Eng. 17(3), 2018.
- [5] Lawrence Livermore National Laboratory: *IOR: HPC I/O benchmark repository*, github.com/hpc/ior; Exascale Computing Project: *Jacamar CI*, gitlab.com/ecp-ci/jacamar-ci.
- [6] R. C. Murphy, K. B. Wheeler, B. W. Barrett, J. A. Ang: *Introducing the Graph 500*, Cray User Group (CUG), 2010, graph500.org.