CARLA 2025

LATIN AMERICA HIGH PERFORMANCE COMPUTING CONFERENCE

- KINGSTON, JAMAICA -

SEPTEMBER 22-26

Job-specific performance monitoring on HPC clusters: Challenges and Solutions

Job-specific performance monitoring on HPC clusters: Challenges and Solutions

presented by
Christian Iwainsky
- RSE expert
- Researcher
- HPC specialist

TECHNISCHE UNIVERSITÄT DARMSTADT

NHR4CES NHR for Computational Engineering Science

christian.iwainsky@tu-darmstadt.de

Hessisches Kompetenzzentrum für Hochleistungsrechnen

brain ware for science

HESSEN
Hessisches Ministerium für Wissenschaft und Kunst

Agenda

- Motivation:
  - Why performance monitoring is important …
  - Compromising between conflicting parties and constraints
- Knowledge gap: admin view vs user view vs performance view
  - Did you know …
- Tools that help:
  - Cluster Cockpit, Patho Jobs
- State of the art:
  - What else is out there
- Where should we go?
- Summary!
- Q&A

S.th. to look at while I talk …

My background …

- Research & Development:
  - Performance-analysis
  - Modelling
  - OpenMP/MPI/C++ tooling
  - Instrumentation Compiler
- Affinity to performance analysis tools:
  - Scalasca, Vampir, LIKWID, HPCToolkit
- Primary research question:
  - What's happening inside this job?
  - How do we make it faster?
  - Still active in research!

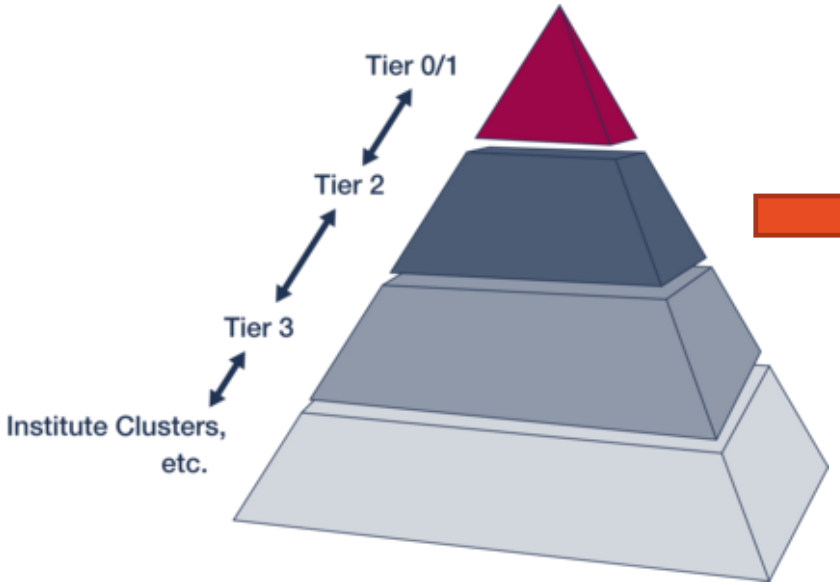
ORCID

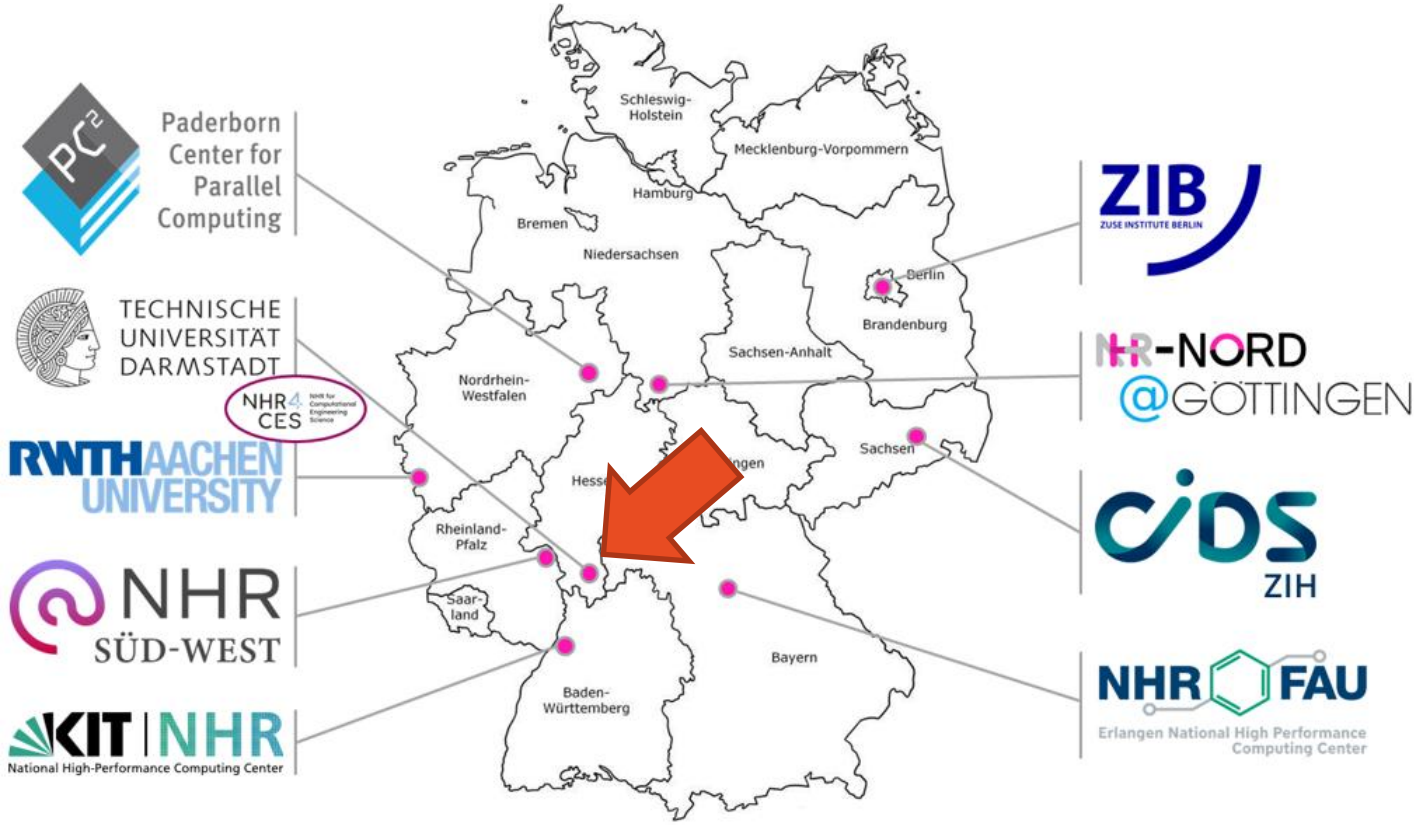
Where I am now?

- Day-job:
  - HPC expert
    - Last-level HPC support
  - RSE engineer
  - Trainer
    - HPC skills & Softwareengineering-coach
- In-between users and system administrators
- Reality:
  - Screen 10.000 Jobs/day
  - ~ 70% outbound interactions
  - Extreme diversity in codes, users and issues
- **"Did you know that..." is how most user conversations start.**

Background of TU Darmstadt

Tier-0 PRACE und EuroHPC
Tier-1 Gauss Center for Supercomputing
Tier-2 NHR
Tier-3 lokale HPC at Universities



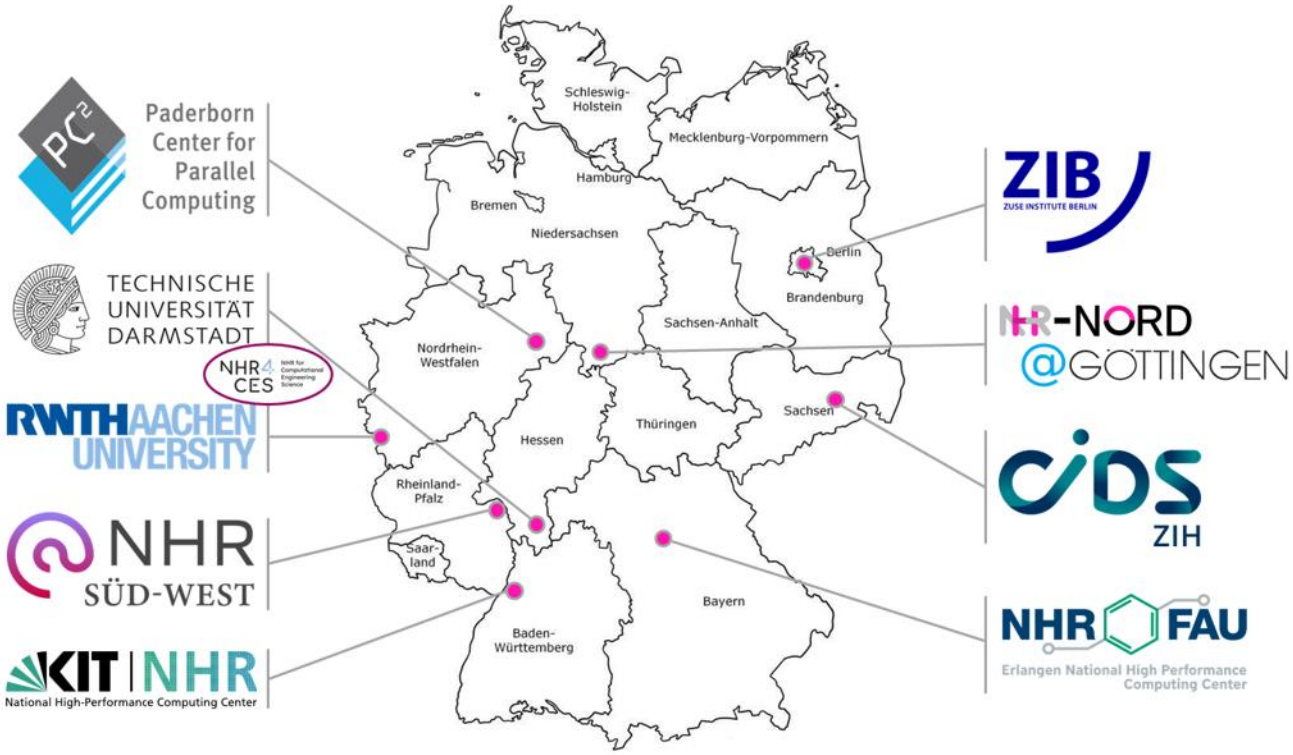


Background of TU Darmstadt

- HPC @ TU Darmstadt

  One of 9 NHR centers in Germany

  - O(1,000) compute nodes
  - O(10,000) total cores @
  - at least 4GB RAM/ core
  - O(500) active users
    - students,
      PHD students,
      post-doc, researchers and
      profs.
  - 2 HPC (user) support staff
  - ~ 10 HPC system admins



What I consider DevOps?

DevOps principle	What I think it means
Observability	Jobs and system behavior observable at runtime
Automation	Measurement and observation is automated
Feedback Loops	Information enables to incrementally improve HPC use
Scalability	Mechanisms scale to all HPC levels
Collaboration	Enables teamwork of admins, support staff and end-users
Tooling	Data and formats allow integration and application of more tools

HPC ain't free, it comes at a price

- Lichtenberg at TU Darmstadt
  - O(10,000) cores (on average for a full system)
  - 15 Mio.€ (cluster incl. storage)
  - 5 years of operation (per installment)
  - 10 full-time staff
  - 5 years of power and cooling
  - Building and infrastructure

Can't share specific numbers

- ~ about 75 cents / core*h
- 72€ / node*h

HPC ain't free, it comes at a price

- Mock-up cluster:
  - Node:
    - CPU: 2x AMD EPYC 9965 (192 cores) ~ 8.000€ / piece
    - Motherboard: Dual Socket Motherboard ~ 1.600€ / piece
    - RAM: 24x 32GB DDR5-4800 (ECC) ~ 270€ / piece
    - Network: InfiniBand HDR100 ~ 850€ / piece
    - ~~GPGPU: NVIDIA H100 ~ 40.000€ / piece~~
    - Misc: Cables, chassis, infrastructure, storage ~ 5000€ / node
  - Operational costs (1000 nodes):
    - Staff: 7 FTE (full time employees) ~ 80k€ / (FTE * y)
    - Power: ~ 3.4 MW ~ 67,70 € / MWh

- Node/h ~ 0,97€ Node/y ~ 8562€
- Cluster/h ~ 977,44€ Cluster/y ~ 8.562 M€

What is the challenge?

Are users

- using all invested ressources?
- fully using all invested ressources?
- being stalled by some resource?

- ~~Understand performance of a code with a dataset on a machine!~~

- Unterstand performance
  - of a **specific** code
  - with a **specific** dataset
  - on a **specific subset** of a machine!

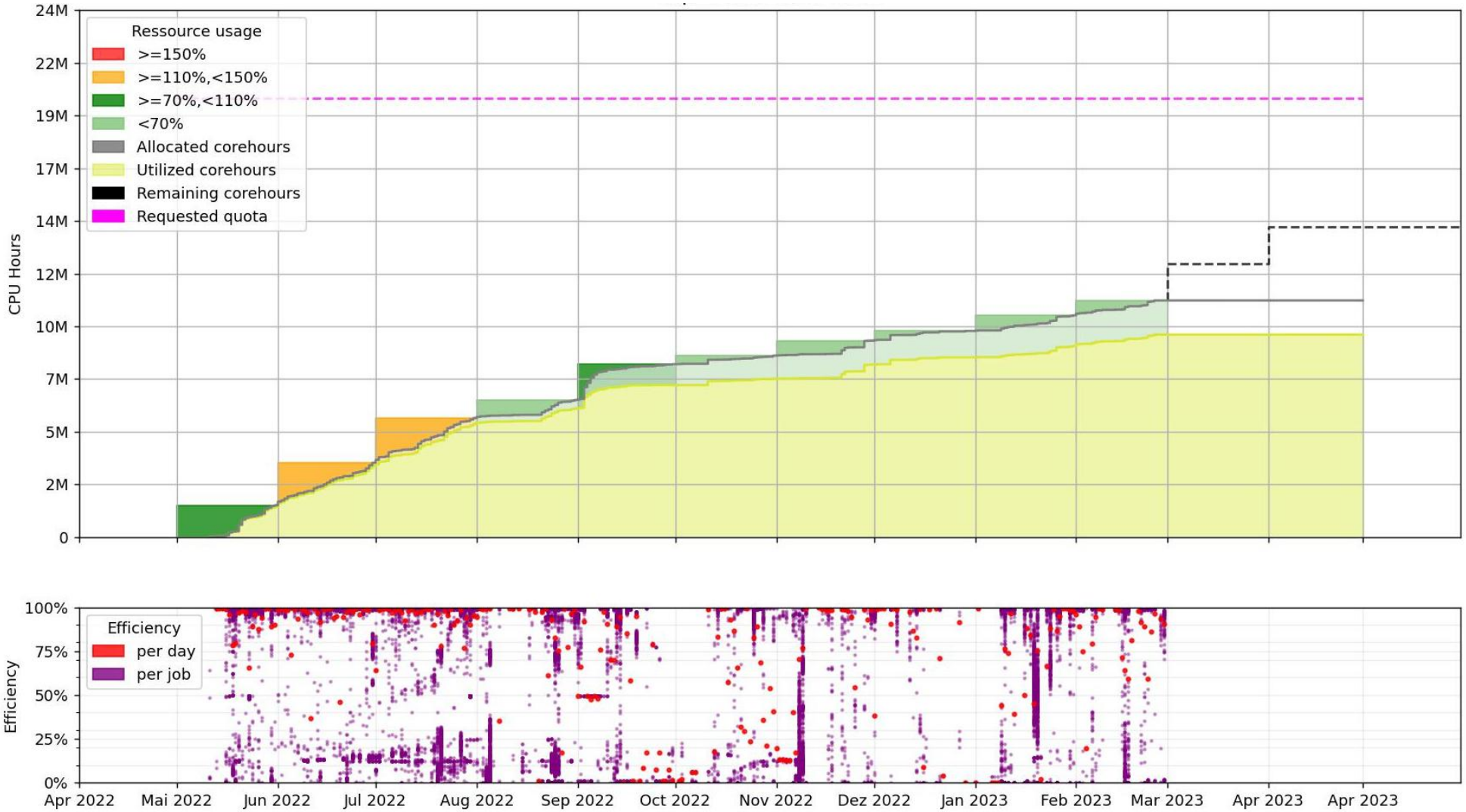
Trust …

Know your performance …

- You can only manage what you observe – and in HPC, our users are often flying blind.
- “Know the performance you get”
- Performance = Work / Time
- We’re not just measuring performance — we’re integrating it into the workflow, at scale, for everyone

Things to catch …

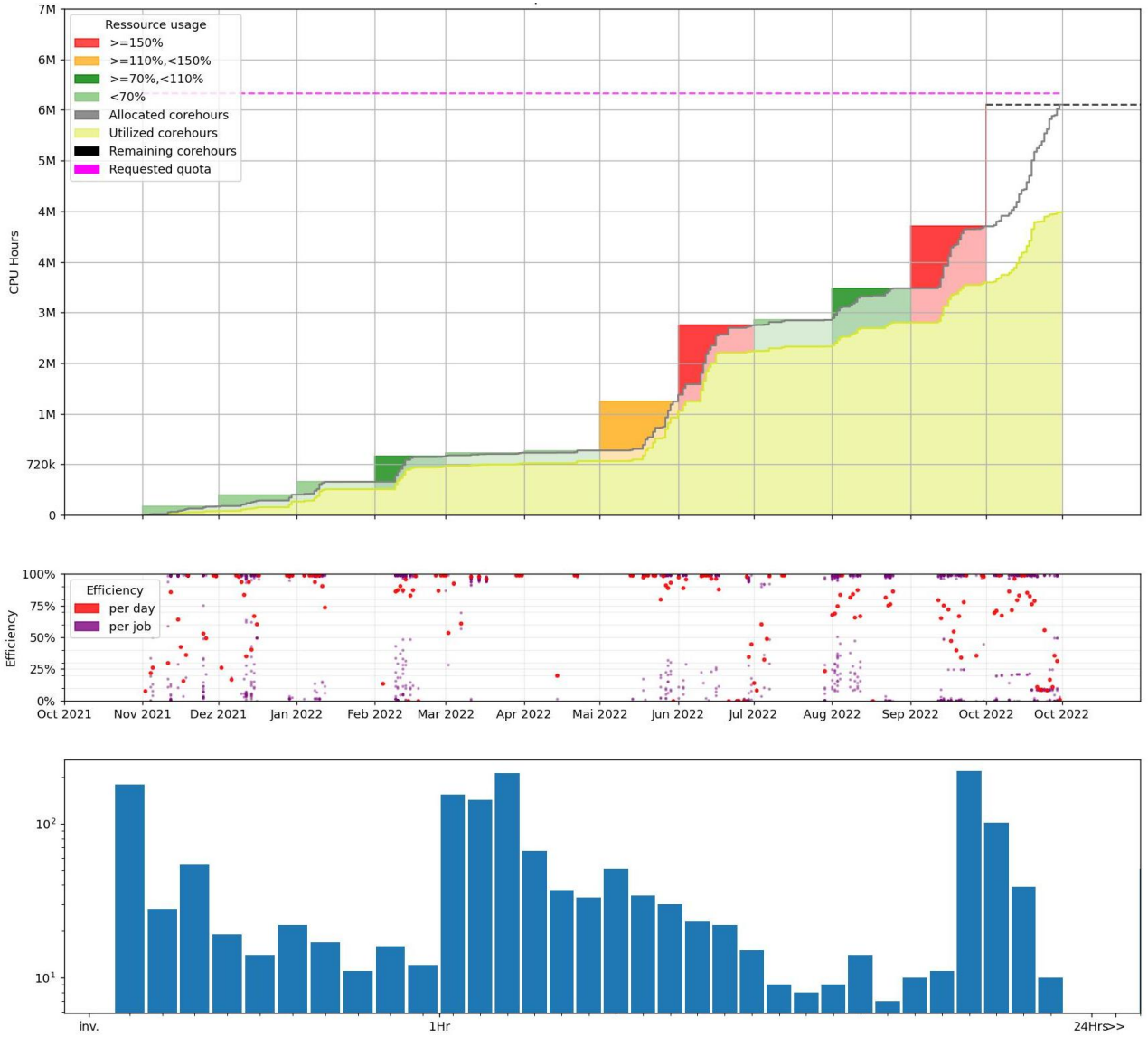
- Utilization
- Usage
- Efficiency



*Deprecated visualization

Things to catch …

- Utilization
- Usage
- Efficiency



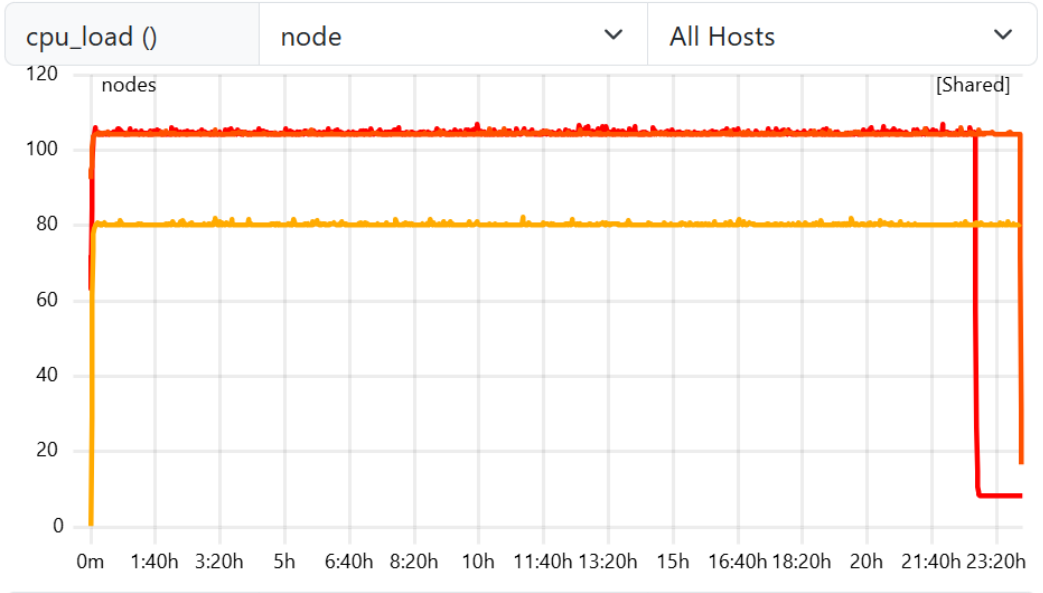
Example: unexpected runtime

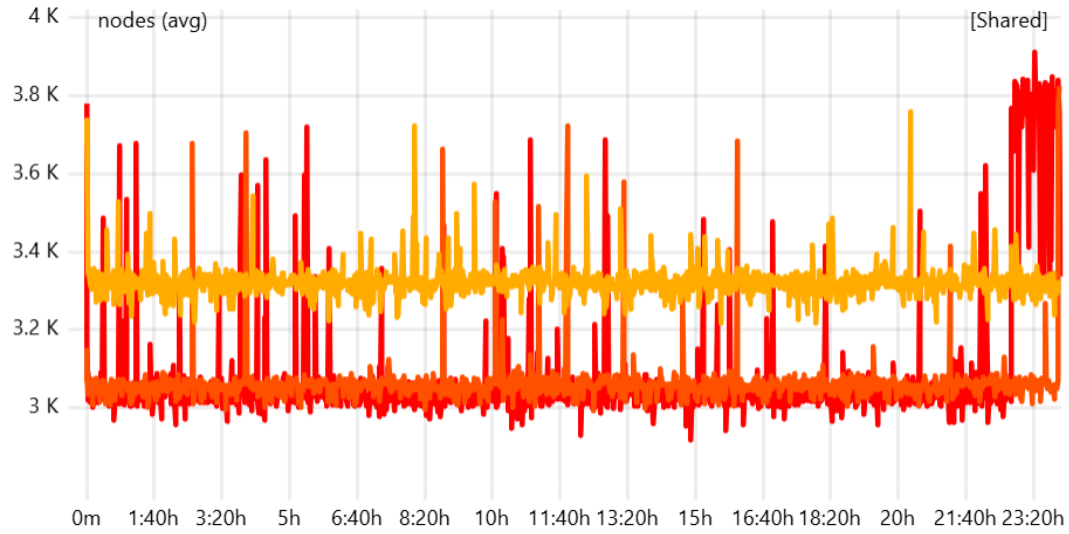
System errors & User mistakes:

Job: Runtime unexpectedly slow

- 96 Processes
- 1 Thread / Core
- Mem-per-cpu=3000 MB

- Runtime use:
  - 3 nodes
  - 48 on node A
  - 24 on node B
  - 24 on node C

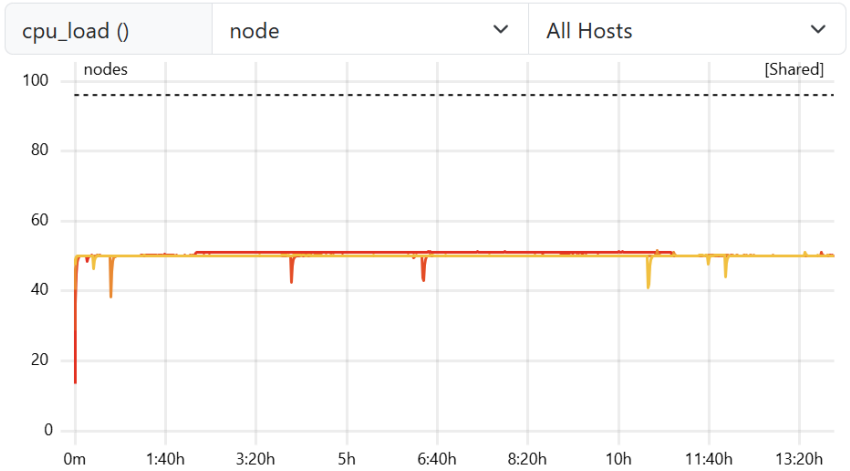


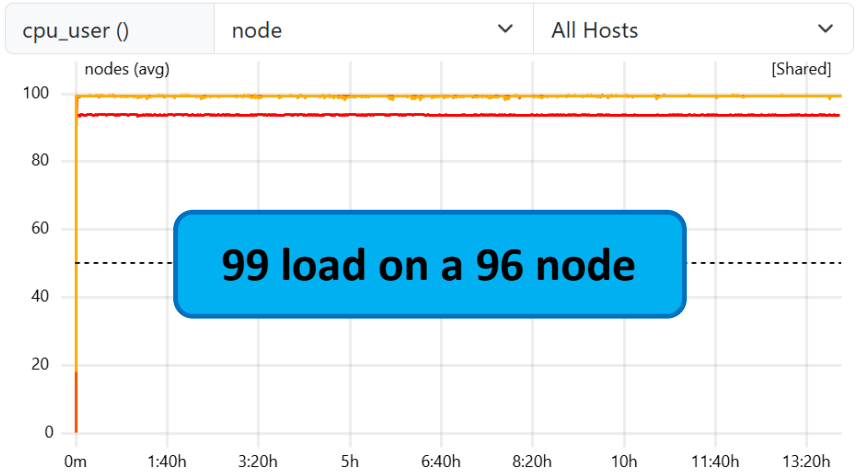


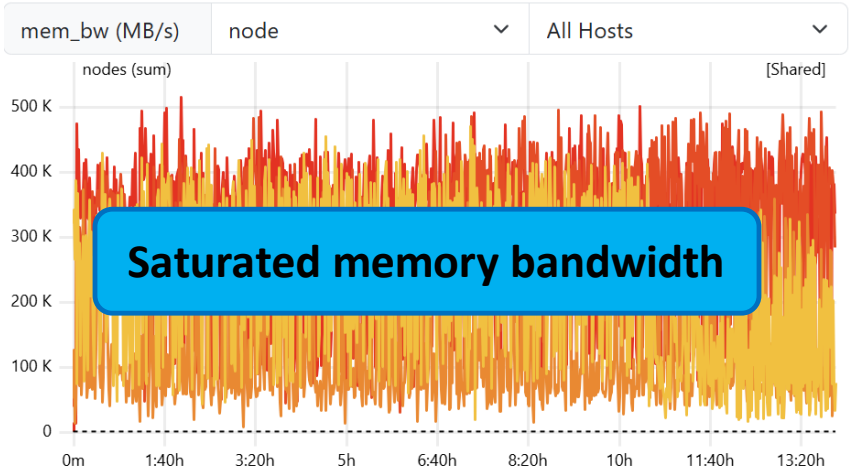
Example: low load

- 96 processes
- 4 nodes
- 7GB / rank
  - ➡ 24 processes / node

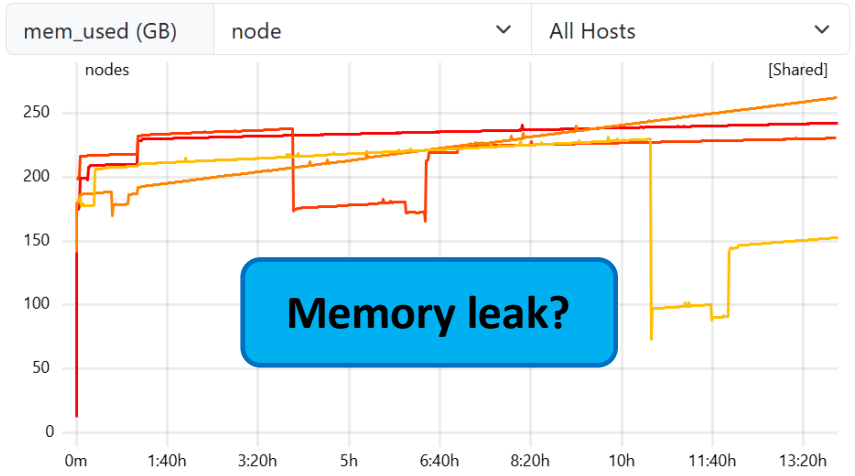
How can job be improved?

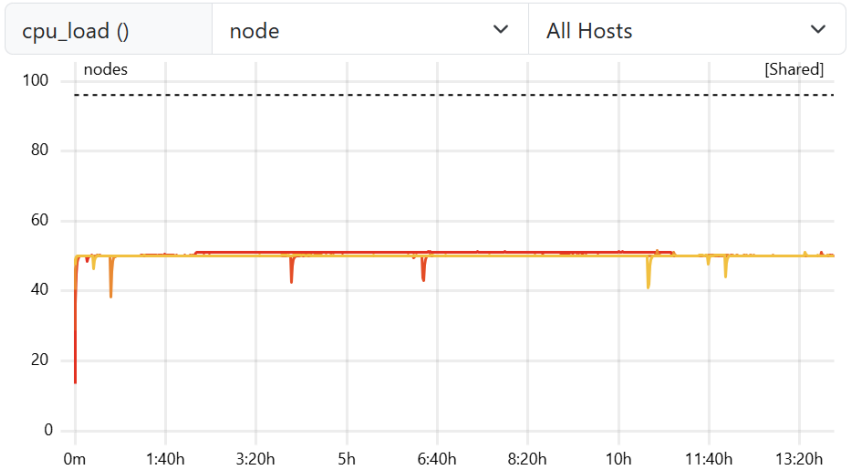


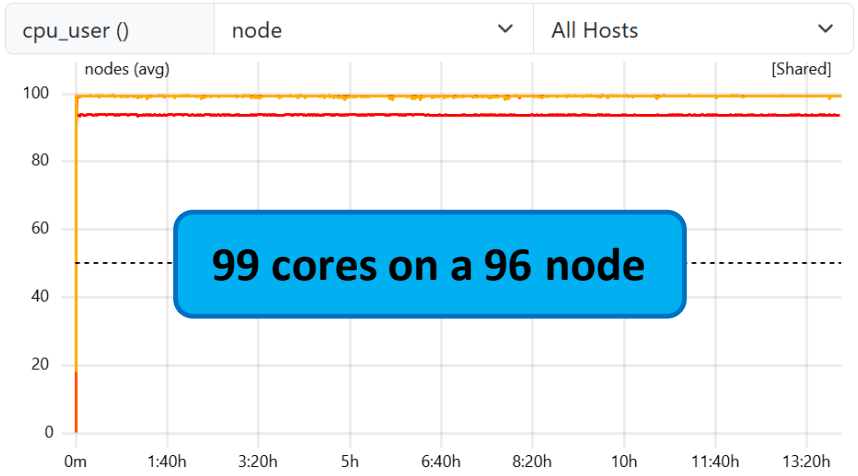


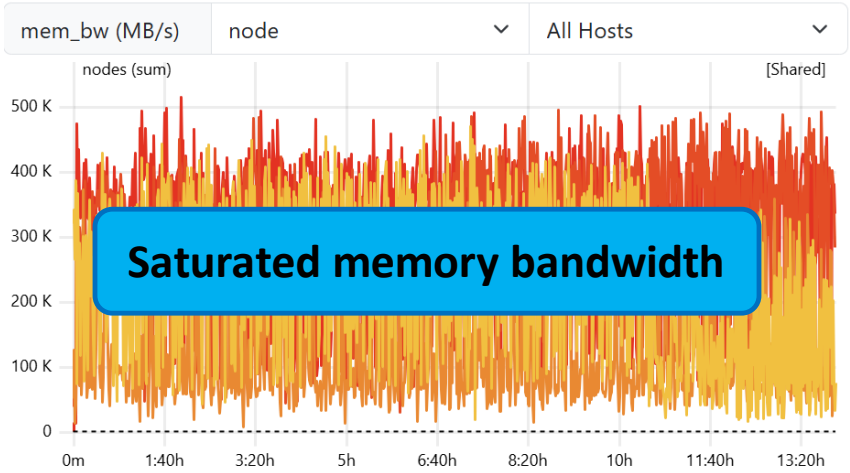


Example: low load









Trust …

But monitor!

What would a monitoring tool need!

- Scheduling information:
  - Which processes ran where?
  - Which ressources were used by a specific job?
    - Ressources per parallel entitiy?
  - What aspects of the hardware was used?
- Hardware information:
  - Which cores were saturated?
  - What are the functional units doing?
  - Which network connection was full?
- Context information:
  - Which mathmatical equations were being solved?
  - Which phase of the code was operational?
- Temporal information:
  - What was the previous state?

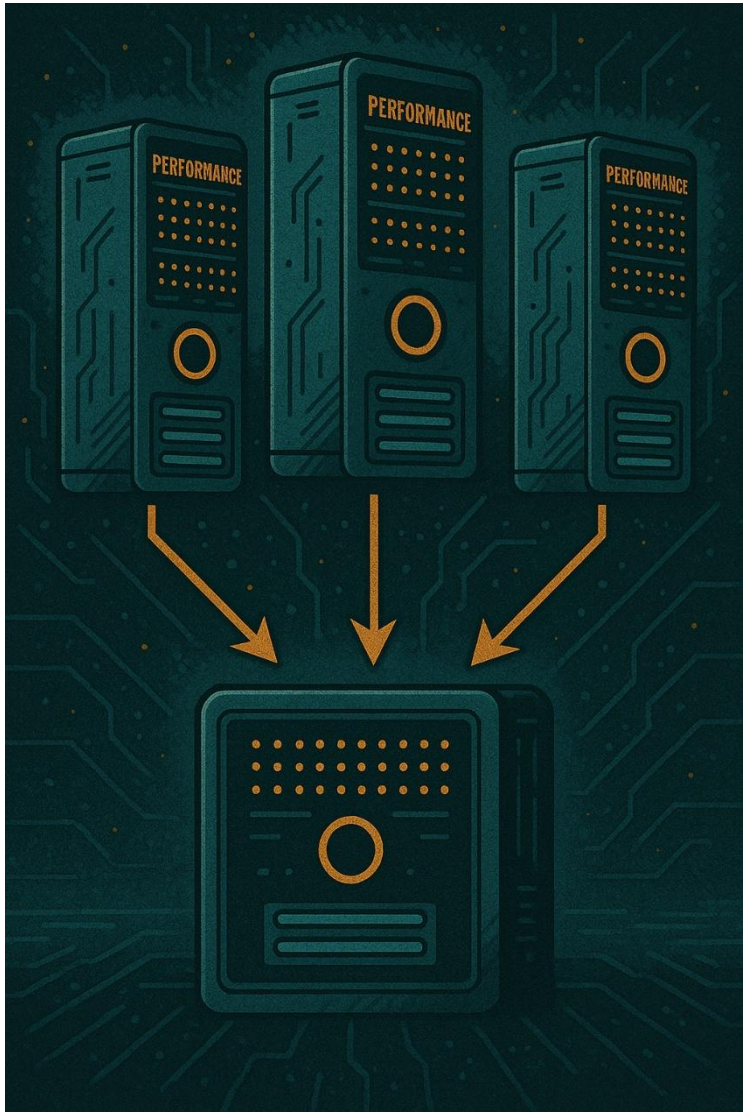
Background: System Level Monitoring

- Goal: track health-status of hardware
  - Zabbix
  - Prometheus
  - Grafana
  - Vendor specific tools
- Operation:
  - Node-level deamon gathers data locally, and sends
  - System-level deamon gathers data remotely
- Criteria:
  - Local overhead
- Data:
  - Operational information:
    - Load, hardware-failurs, energy consumption

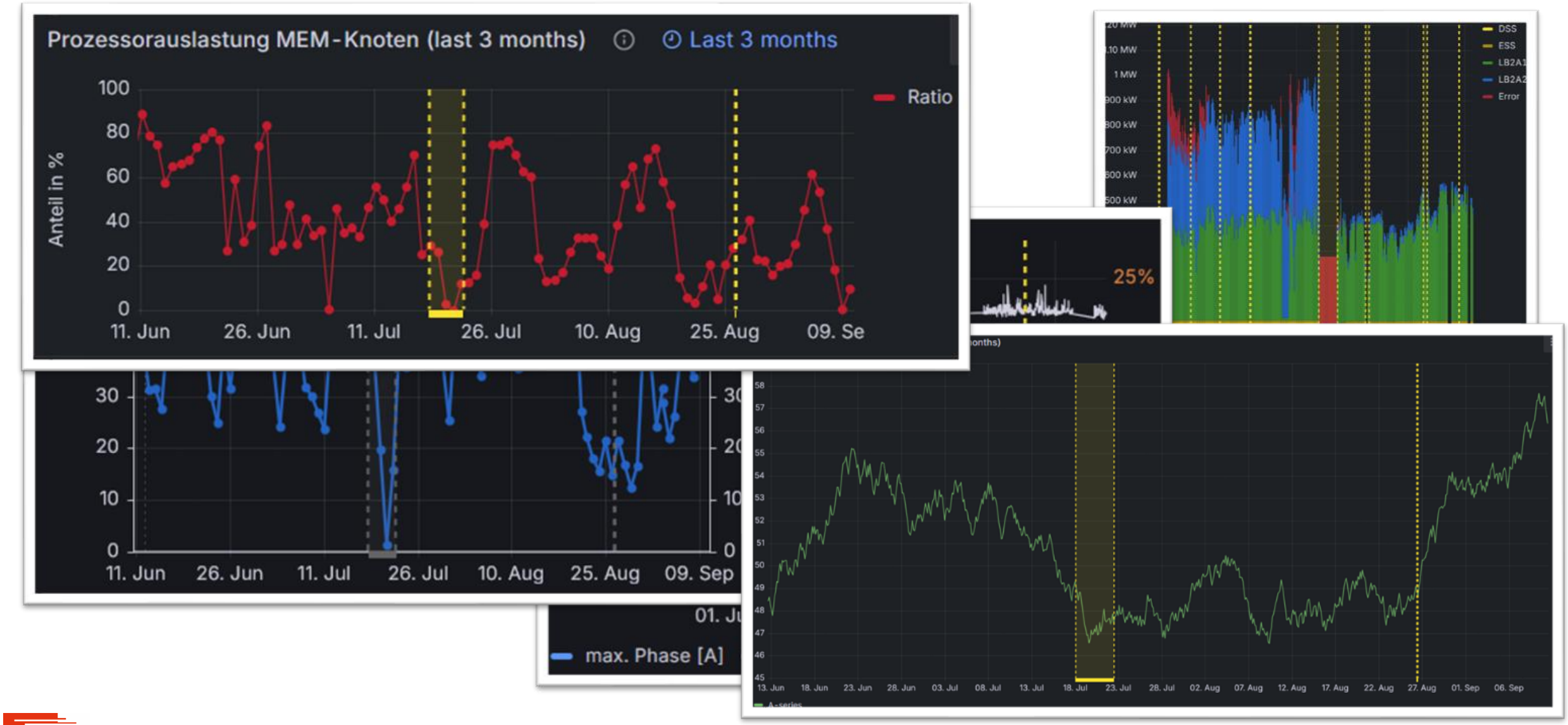
How does it work?

- Node agent gather all data
  - push: active send it to an aggregation agent
  - pull: data gathered from aggregation agent
- Aggregation agent / service
  - Processes data and vizualize

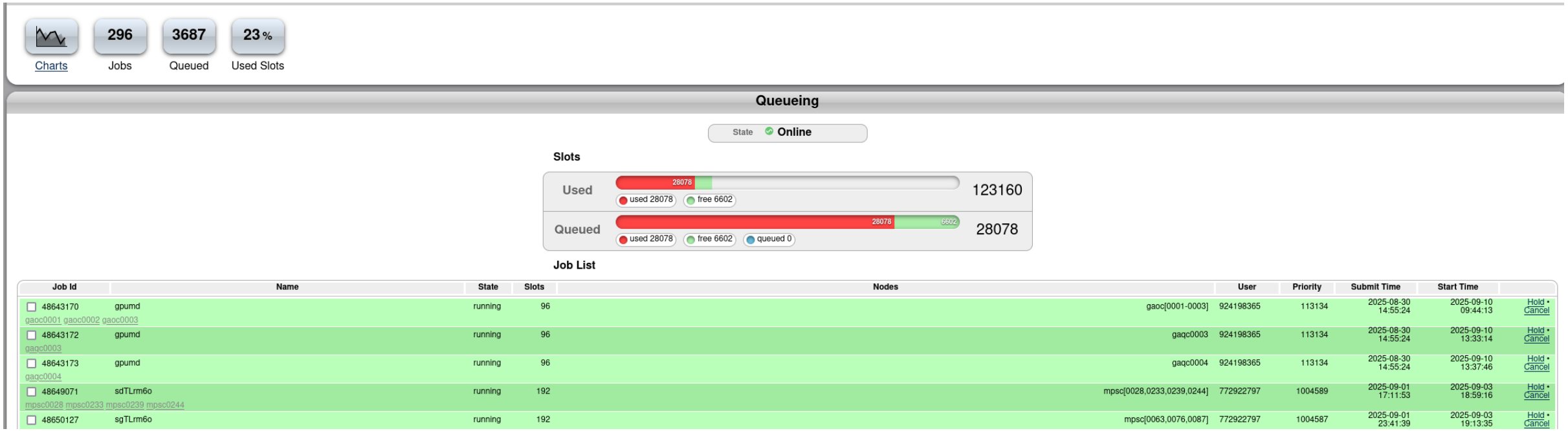
- Challenges:
  - Perturbation by the node agent
    1 ms / core @ 394 cores -> 0.4 s / node / sample
  - Network-traffik
    - 1000 nodes * 394 cores * 1kb = 394MB / sample
  - Aggregation work (DDOS)
    - sychronous delivery: O(1 million) packets per query



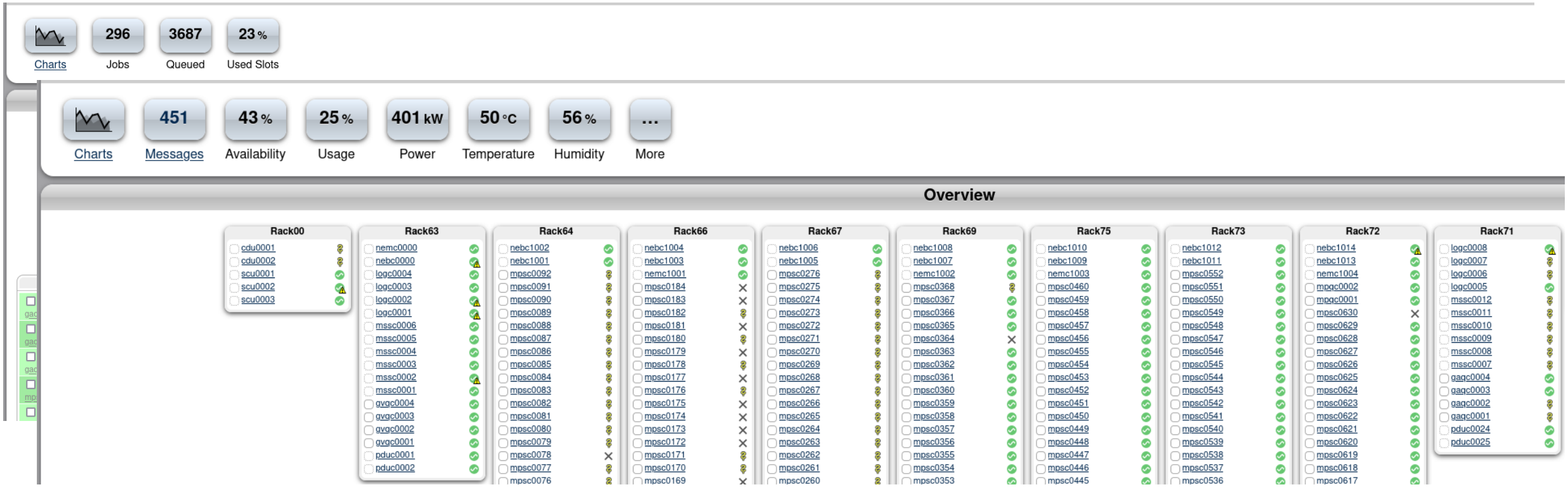
System Level Monitoring:



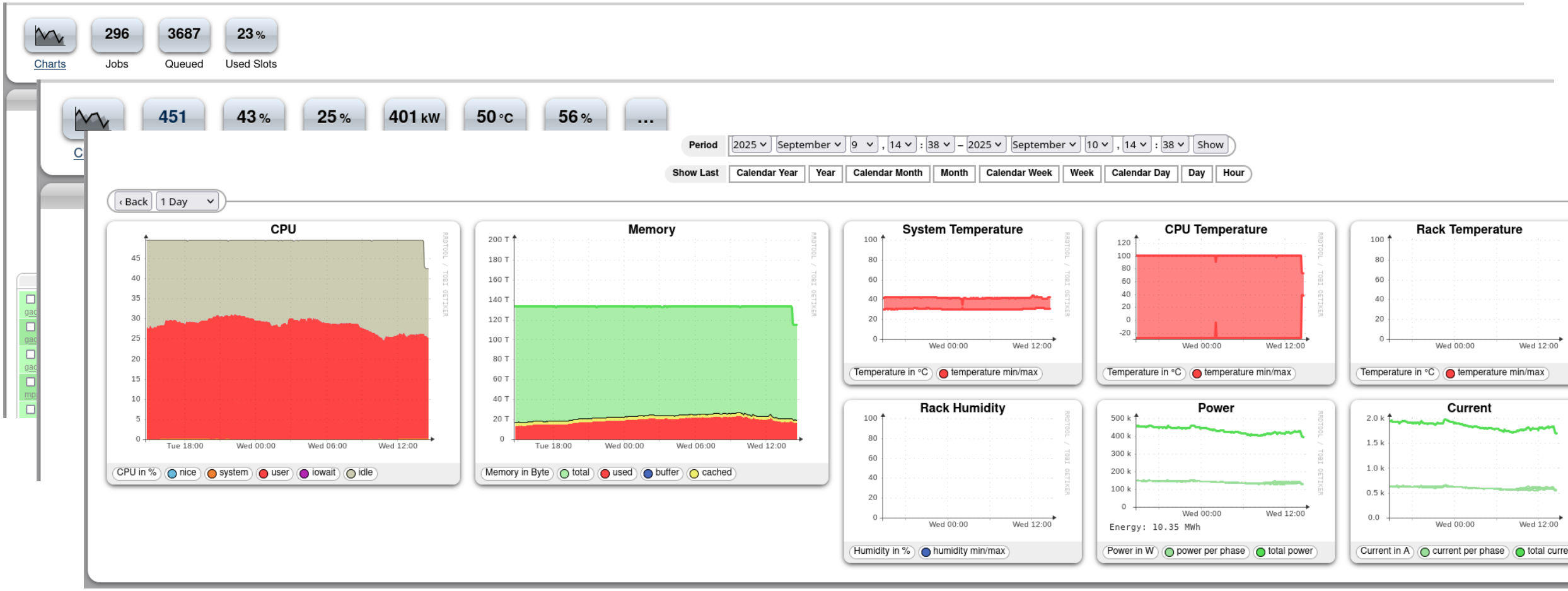
System Level Monitoring:



System Level Monitoring:



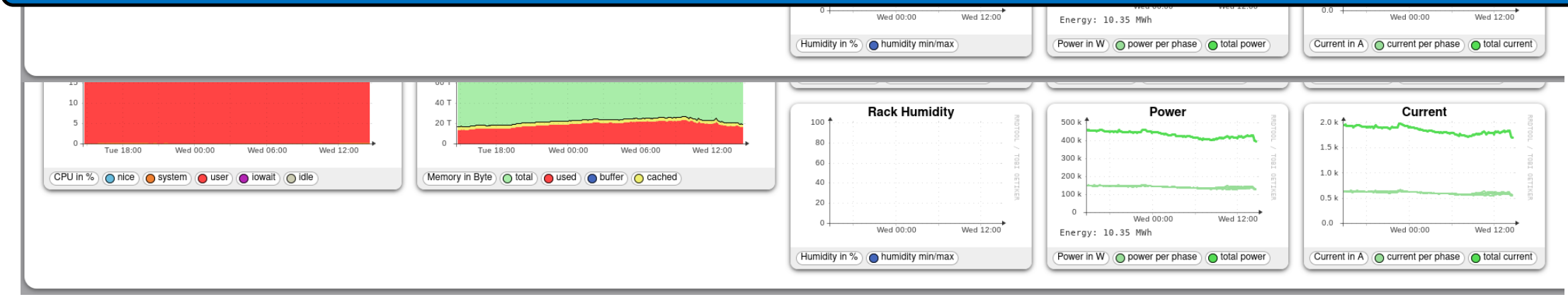
System Level Monitoring:



Manual correlation of data …

sacct (SLURM accounting):

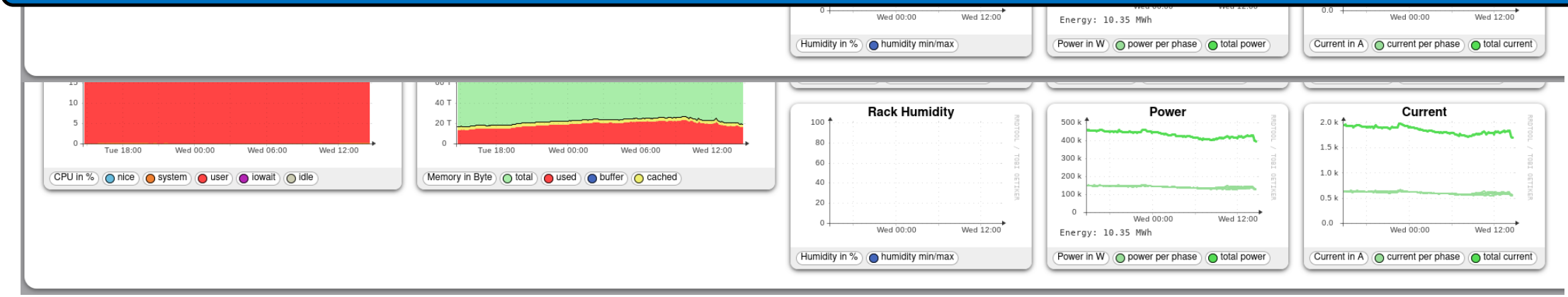
4 1 96 02:01:34 COMPLETED 0:0 336K 0 0 0 594.24K 130.23M mpsc0150 0 130.23M 7.57M mpsc0150 0 7.57M cpu=96,me+ cpu=02:00:34,+ cpu=02:00:34,+ cpu=mpsc0150,ener+ cpu=0,fs/disk=0,m+ cpu=02:00:34,+ cpu=mpsc0150,ener+ cpu=0,fs/disk=0,m+ cpu=02:00:34,+ energy=81,fs/d+ energy=mpsc0150,fs+ fs/disk=0 energy=70,fs/d+ energy=70,fs/d+



Manual correlation of data …

sacct (SLURM accounting):

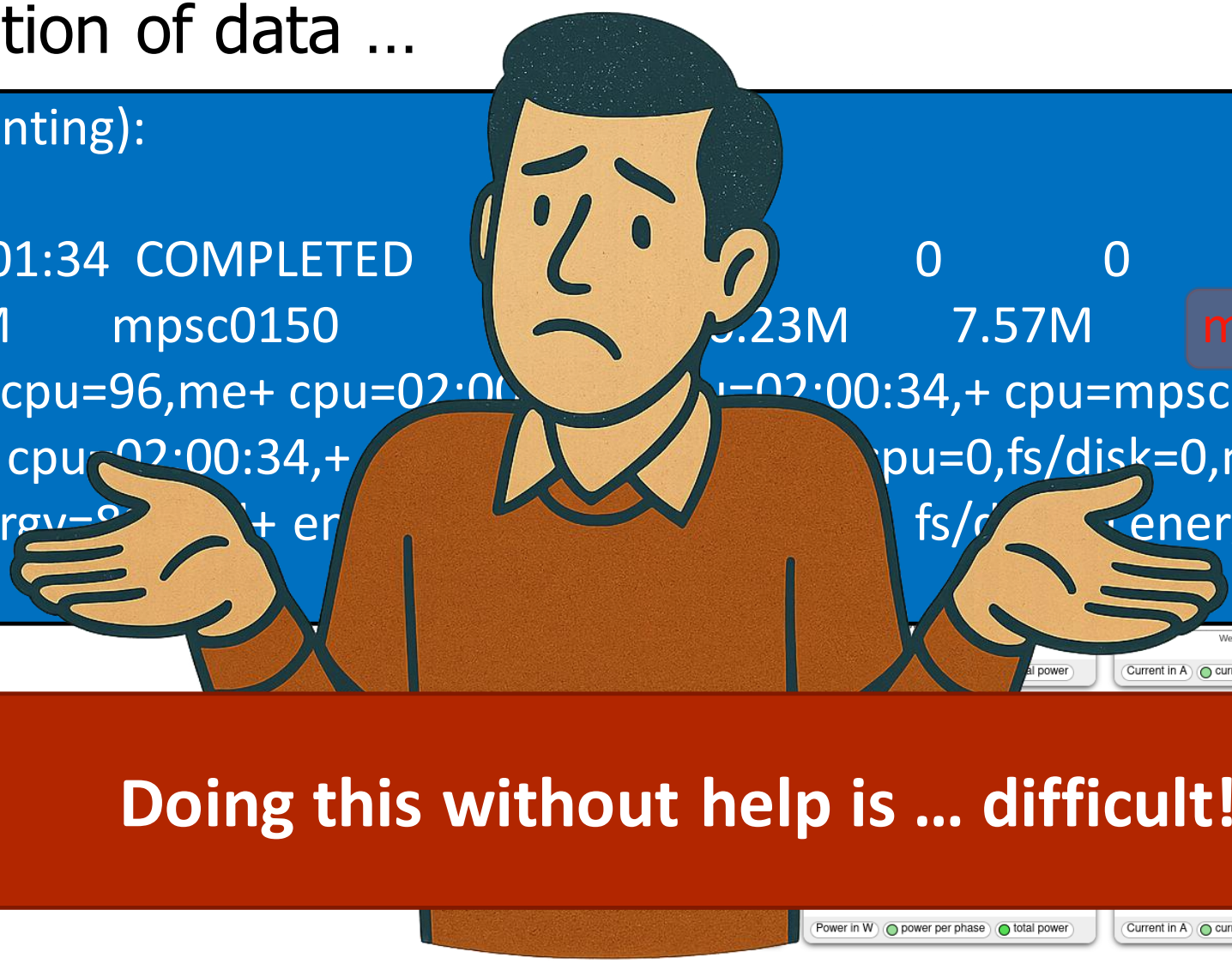
4 1 96 02:01:34 COMPLETED 0:0 336K 0 0 0
594.24K 130.23M mpsc0150 0 130.23M 7.57M mpsc0150
0 7.57M cpu=96,me+ cpu=02:00:34,+ cpu=02:00:34,+ cpu=mpsc0150,ener+
cpu=0,fs/disk=0,m+ cpu=02:00:34,+ cpu=mpsc0150,ener+ cpu=0,fs/disk=0,m+
cpu=02:00:34,+ energy=81,fs/d+ energy=mpsc0150,fs+ fs/disk=0 energy=70,fs/d+
energy=70,fs/d+



Manual correlation of data …

sacct (SLURM accounting):

4 1 96 02:01:34 COMPLETED 0 0 0
594.24K 130.23M mpsc0150 .23M 7.57M mpsc0150
0 7.57M cpu=96,me+ cpu=02:00 =02:00:34,+ cpu=mpsc0150,ener+
cpu=0,fs/disk=0,m+ cpu=02:00:34,+ pu=0,fs/disk=0,m+
cpu=02:00:34,+ energy=8 + er fs/d energy=70,fs/d+
energy=70,fs/d+



Doing this without help is … difficult!

What would we like to see?

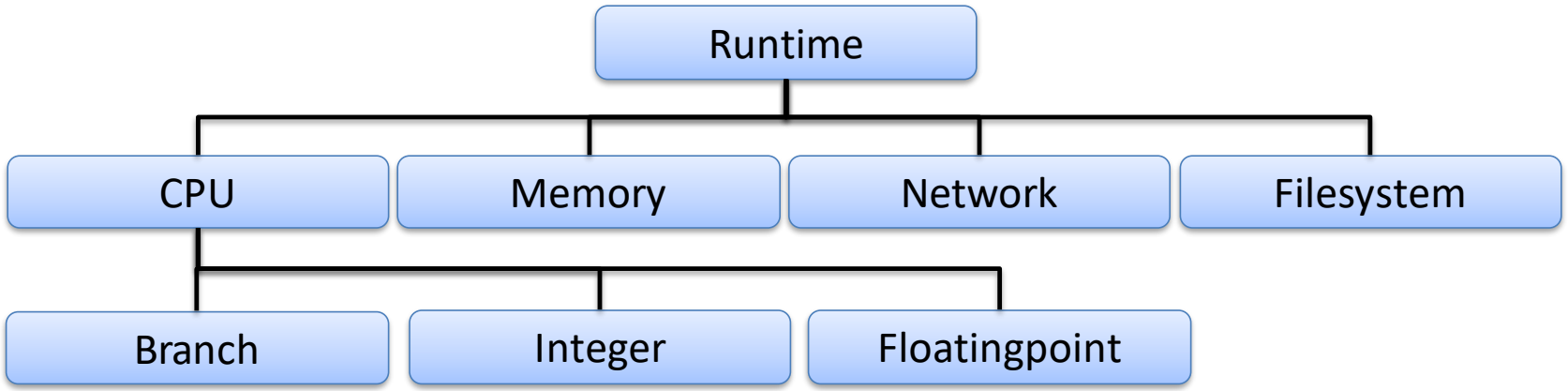
- Domain of job:
  - Which processes ran where?
  - Which ressources were used by a specific job?
    - Ressources per parallel entitiy?
  - What aspects of the hardware was used?
- What was the hardware doing:
  - Which cores were saturated?
  - Which network connection was full?
- What was the software trying to achieve
  - Which mathmatical equations were being solved?
  - Which phase of the code was operational?

What metrics?

Simple model

$$Runtime = \frac{Number\ of\ Instructions}{\frac{Instructions}{Second}} + \frac{\frac{Memory}{Instruction}}{\frac{Memory}{Second}} + \frac{Memory\ to\ Communicate}{\frac{Memory}{Second}} + \cdots$$

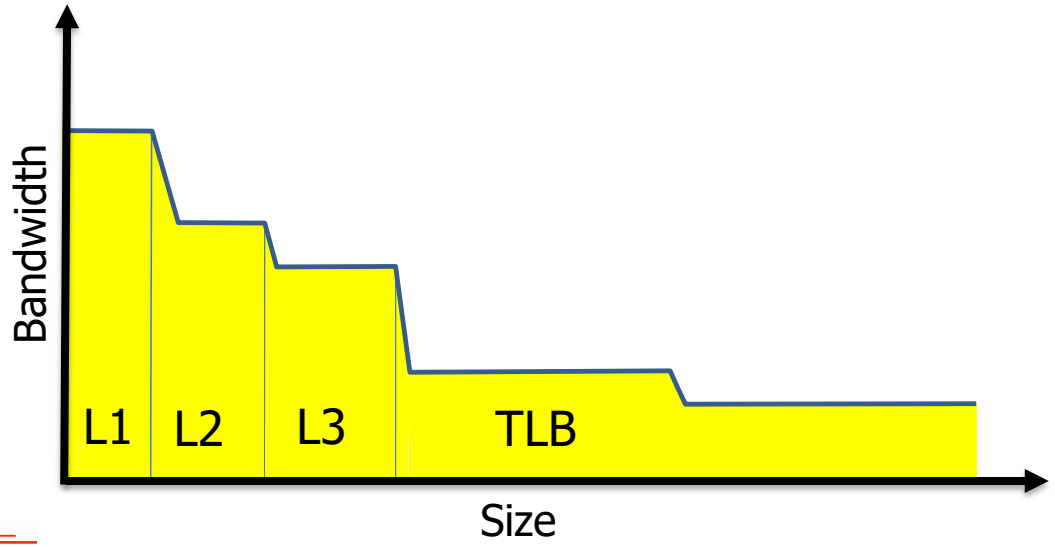
- Metrics aim to capture and overt aspects of this expression
- Each of the terms above can (and should) be split into more precise sub-terms



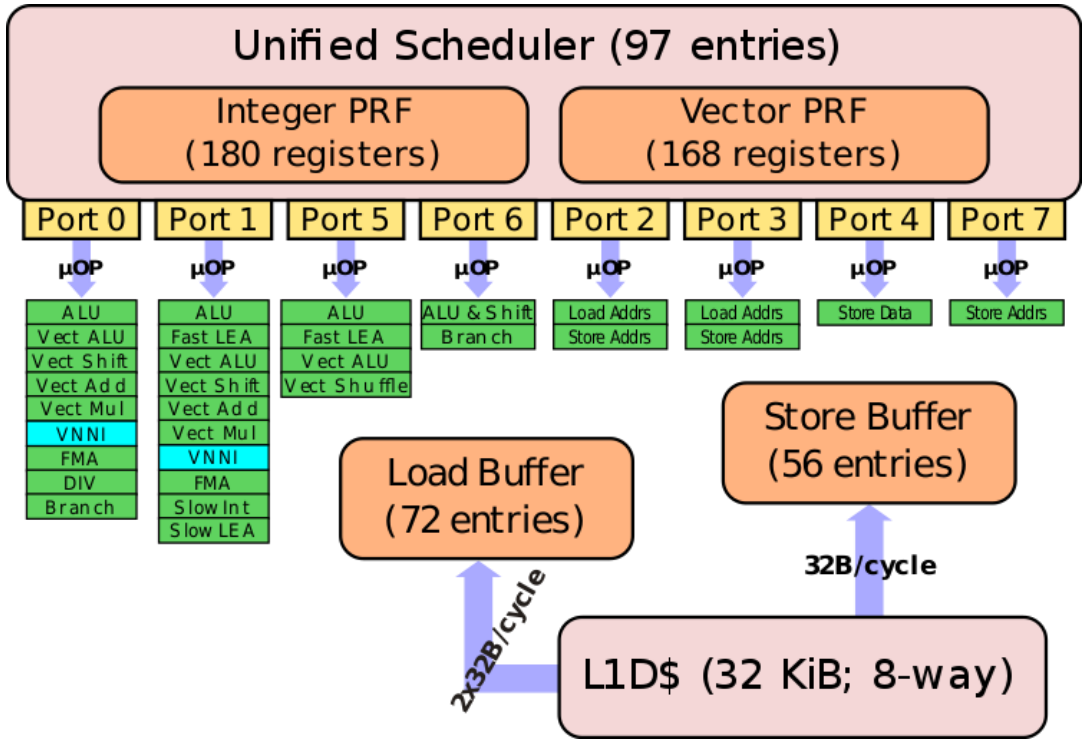
Architecture knowledge

- Understanding of the systems architectural capabilities: everything is complex

Memory



CPU design



Src.: https://en.wikichip.org/wiki/intel/microarchitectures/cascade_lake

Hardware Performance Counters

- More than 2000 events, limited counter registers

Counter-group	Details
CYCLE_STALLS	
FLOPS_DP	Double Precision MFLOP/s
L2	L2 cache bandwidth in MBytes/s
DATA	Load to store ratio
MEM	Main memory bandwidth in MBytes/s
L3CACHE	L3 cache miss rate/ratio
Power	Power and Energy consumption
BRANCH	Branch prediction miss rate/ratio
…	
CACHES	CACHES

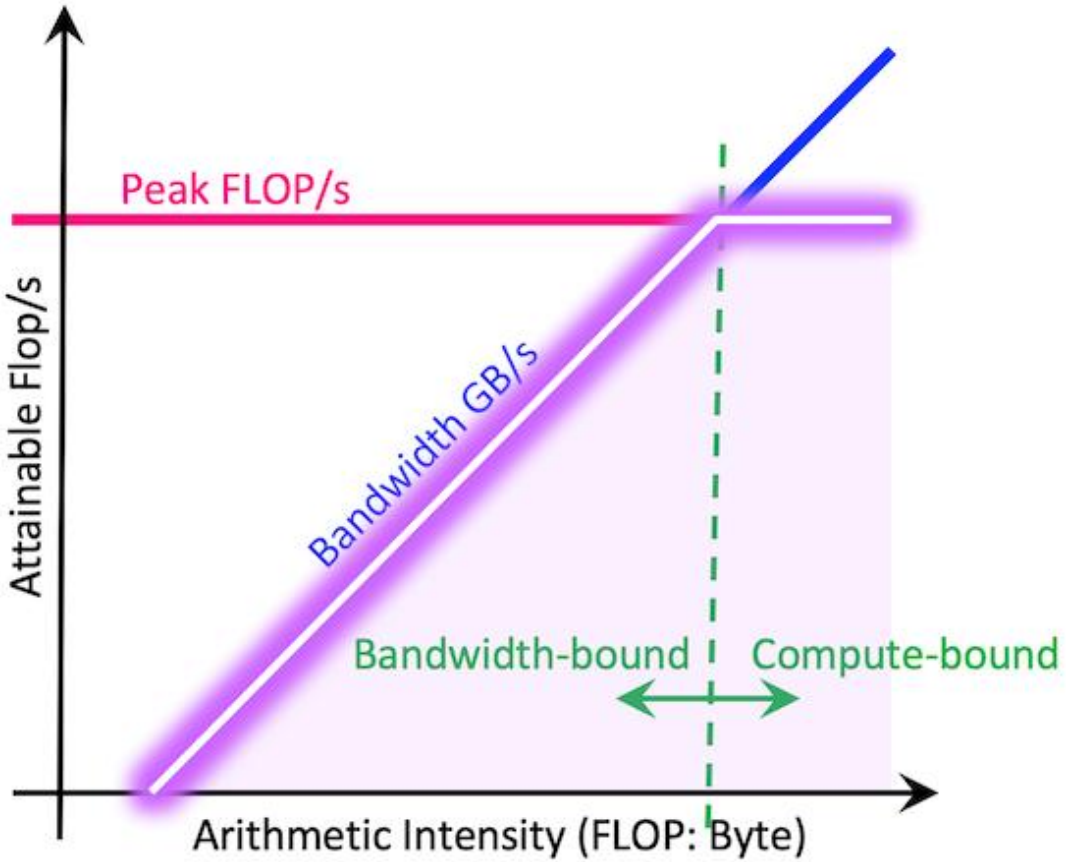
MEM:
- Memory read bandwidth [MBytes/s]
- Memory read data volume [GBytes]
- Memory write bandwidth [MBytes/s]
- Memory write data volume [GBytes]
- Memory bandwidth [MBytes/s]
- Memory data volume [GBytes]

Roof-line model

- Lay-mans summary:

  Performance limited either by
  - compute or
  - memory
- Arithmetic intensity $\frac{Work}{Byte}$
- Goal: Get to the ridge-line
  - Move to the right: do more work with the same data
  - Move to the left: do less work with the same data

Applies to a steady-state

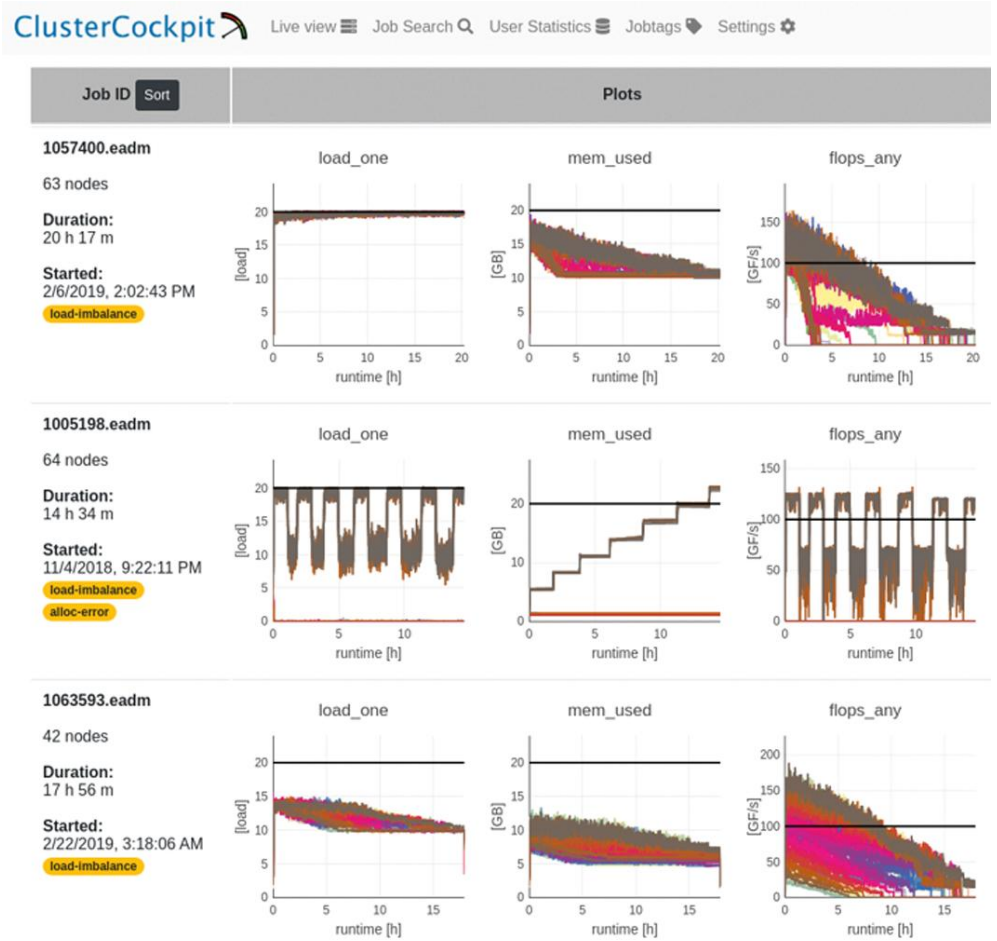


Src. https://docs.nersc.gov/tools/performance/roofline/

Understanding and Context

- Metrics gathered for a „whole run“ have little expressiveness
- Context during which a metric was gathered
- Correlate change in metric to specific code regions

- Requirements:
  - How it „should“ run
    What the hardware can do
  - Observe actual execution



Src: https://www.rrze.fau.de/2020/01/webanwendung-zum-jobspezifischen-performance-monitoring-eine-erfolgsgeschichte-keine-ressourcen-verschwendung-gezielte-optimierung/

Measurement Techniques

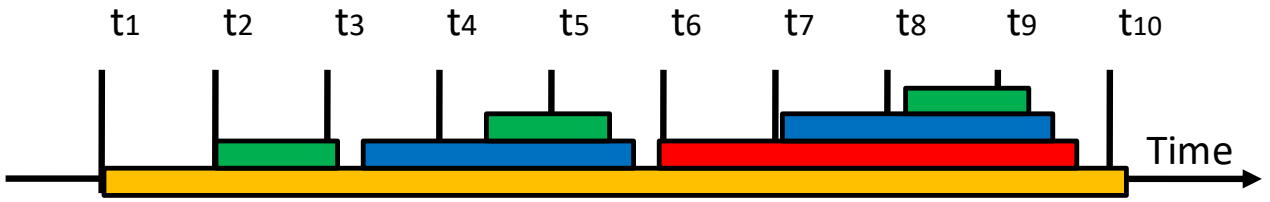
- Process isolation prevents direct observation of a programs state
  - Modify the target program before execution
  - Use a loader to gain control of the application
    - Keyword: LD_PRELOAD mechanism
- Terminology:
  - **Hook**: Facility to potentially measure
  - **Probe**: A measurement device, connecting to a hook
  - **Measurement**: Gathering of data and context

- Resolving the temporal state of the program
  1) Sampling
  2) Instrumentation

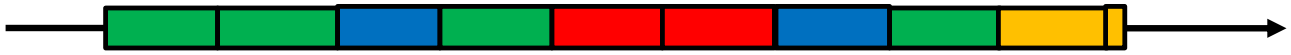
Sampling without unwinding

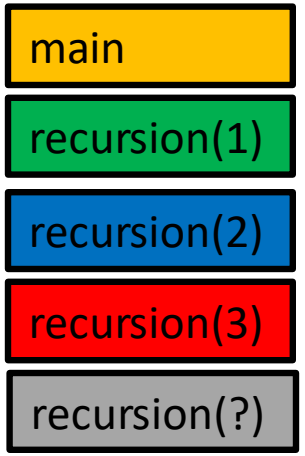
```
void recursion(int depth){
    if (depth>1)
        return recursion(depth-1);
    return;
}

int main(int argc, char ** argv){
    for (int depth=1;depth<4;depth++)
    {
        recursion(depth);
    }
}
```

Call-stack view:



Measurement view:





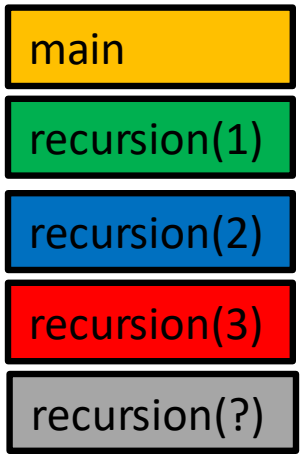
- Unwinding exposes stack
- „Thunking/Trunking" accuarately tracks function exits

Basic Instrumentation Concept

```
void recursion(int depth){
    if (depth>1)
        return recursion(depth-1);
    return;
}

int main(int argc, char ** argv){
    for (int depth=1;depth<4;depth++)
    {
        recursion(depth);
    }
}
```

Call-stack view:

Measurement view:

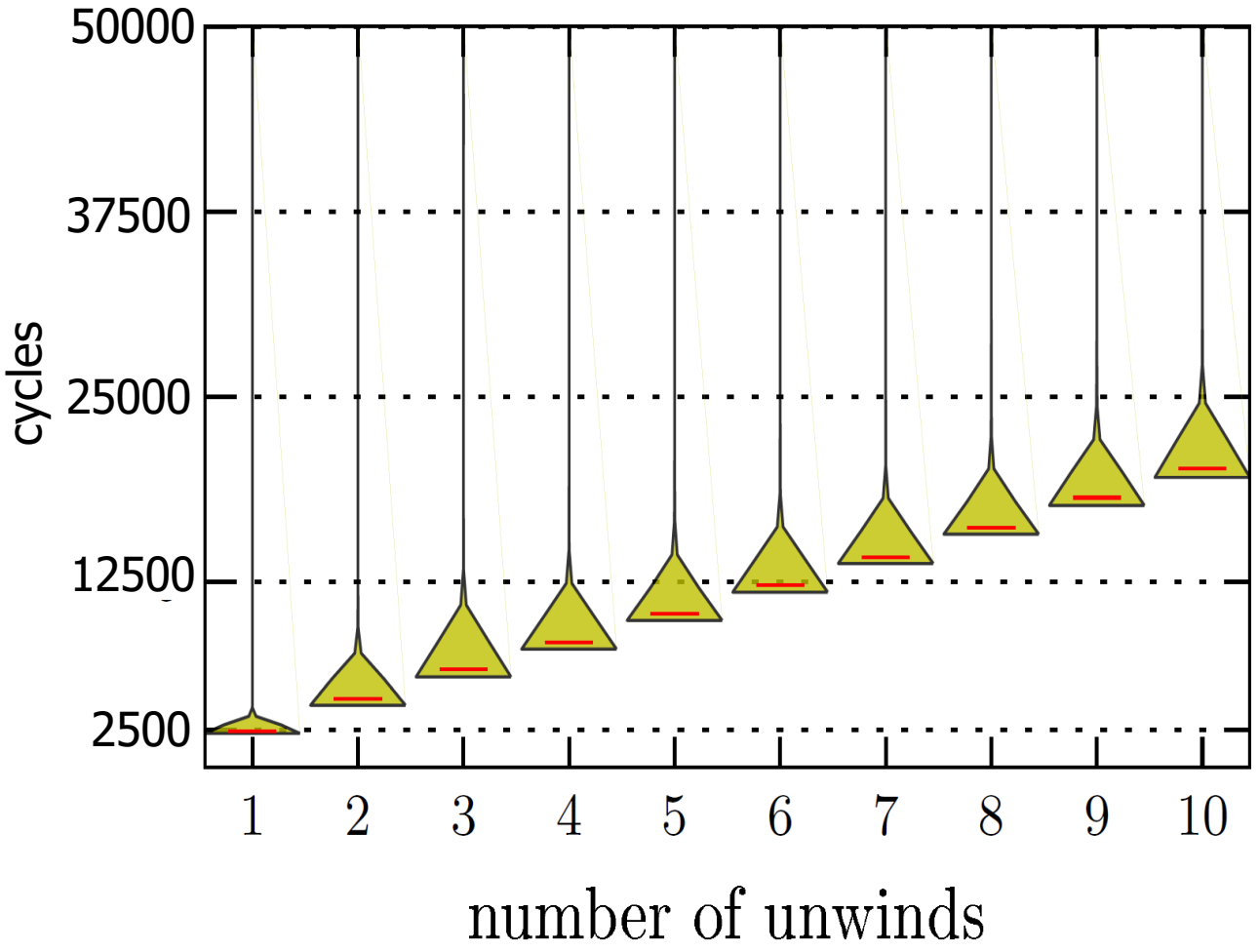


- Instrumentation before or after optimization

Overhead of sampling and unwinding

- $O_{Sampling} = \sum_{locations} O_{Unwind}(loc.) * n(loc.)$
  - » $O_{Unwinding} \approx 4{*}10^{-6}$ s,
  - » $\sum_{locations} n(loc.)$/second ≈ 200Hz
- Unwinding is not for free
- Example:
  - Runtime: 10000s
  - Samplerate: 200Hz
  - Average unwinding-depth: 10

  Overhead:
  100,000s*200Hz*7.5µs = 15s



CPU: 2,5GHz

DOI: 10.1109/ICPPW.2014.33

Instrumentation overhead considerations

Benchmark	Runtime	Number of Functions	Number of Function Calls
444.namd	392	100	1,62E+06
453.povray	155	809	1,41E+06
464.h264ref	62	319	2,55E+08
403.gcc	288	2734	3,40E+08
447.dealII	302	5491	1,17E+08
458.sjeng	488	73	1,11E+10
473.astar	338	129	7,66E+09
433.milc	437	112	1,73E+07
450.soplex	205	769	4,72E+09
482.sphinx3	532	203	1,25E+09
lulesh	102	254	6,50E+10
miniFE	46	552	2,80E+10
DROPS	63	7257	1,70E+11

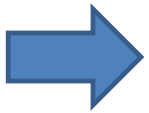
- $O_{Instrumentation} = O_{Probe} * n_{invocations}$
- Cost of a single probe:
  - $O_{Probe} \approx 4 * 10^{-9}$ s
- Number of probe-calls:
  - 1,41 E+06 < $n_{invocations}$ < 170 E+09

Instrumentation overhead considerations

Benchmark	Runtime	Number of Functions	Number of Function Calls	Overhead	Overhead in %
444.namd	392	100	1,62E+06	1,30E-02	0%
453.povray	155	809	1,41E+06	1,13E-02	0%
464.h264ref	62	319	2,55E+08	2,04E+00	3%
403.gcc	288	2734	3,40E+08	2,72E+00	1%
447.dealII	302	5491	1,17E+08	9,36E-01	0%
458.sjeng	488	73	1,11E+10	8,88E+01	18%
473.astar	338	129	7,66E+09	6,13E+01	18%
433.milc	437	112	1,73E+07	1,38E-01	0%
450.soplex	205	769	4,72E+09	3,78E+01	18%
482.sphinx3	532	203	1,25E+09	1,00E+01	2%
lulesh	102	254	6,50E+10	5,20E+02	510%
miniFE	46	552	2,80E+10	2,24E+02	487%
DROPS	63	7257	1,70E+11	1,36E+03	2159%

Take-Away:

- Performance analysis techniques helpful
- Too expensive for job level monitoring
- User work to tailor measurement



Gathering program state without help not likely

Is there a middle ground?

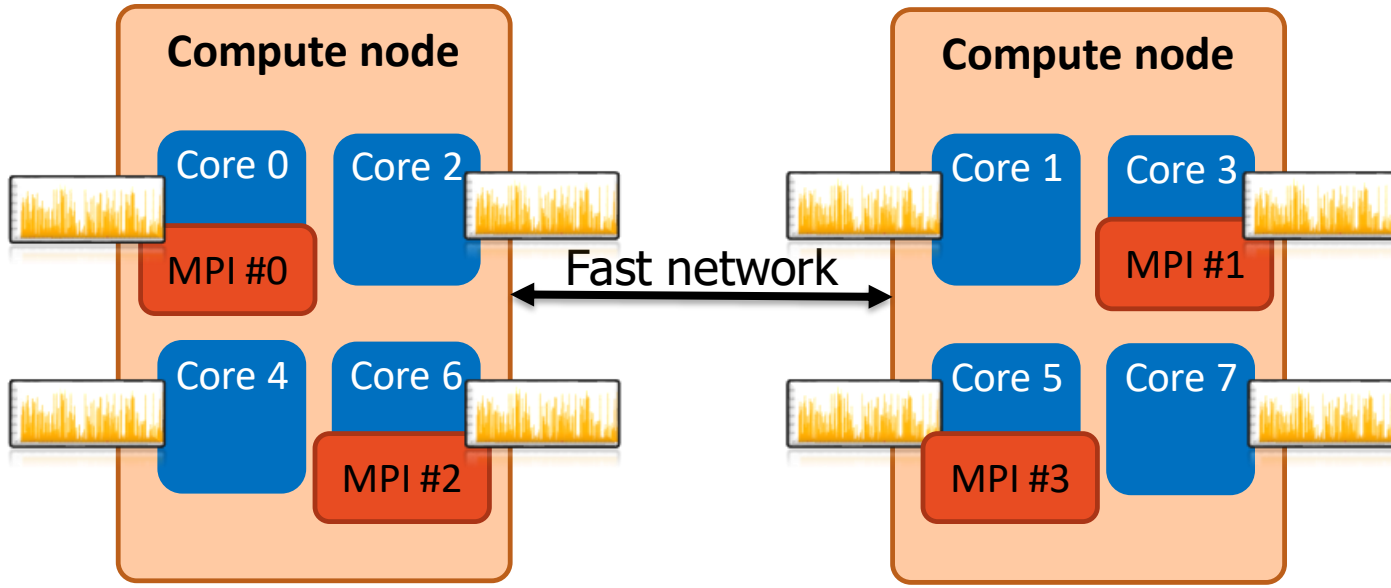
- System operation:
  - system healthy?
  - all hardware operational?
  - providing peak performance?
- Usage monitoring:
  - what ressources are used ?
  - which ressources are the limiting factor?
    (compute units, memory bandwidth, memory capacity …)?
  - are ressouces utilized, or just occupied?
- Performance monitoring / analysis:
  - what is the software doing?
  - why is the software behaving like that?

less context, less data, existing solutions

more context, more data, existing solutions

Correlation game:

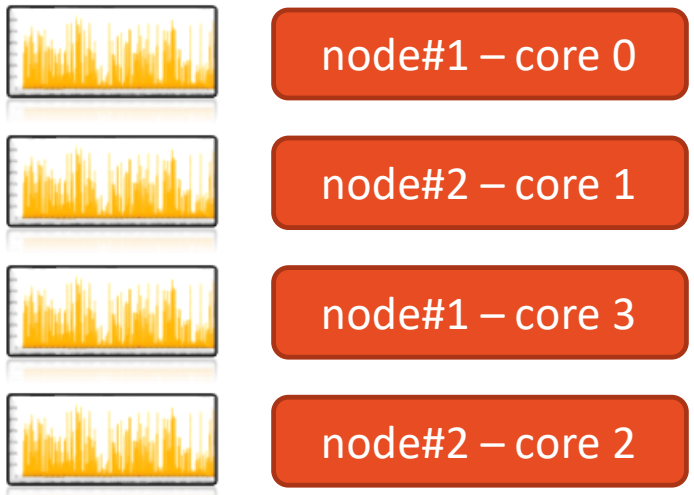
- What can we know:
  - Which software (job) ran when & where
    - Batch systems knows
      - E.g.: Which MPI rank ran when and where?
  - Time-series of metrics at granularity
    - Metrics per scheduled entity

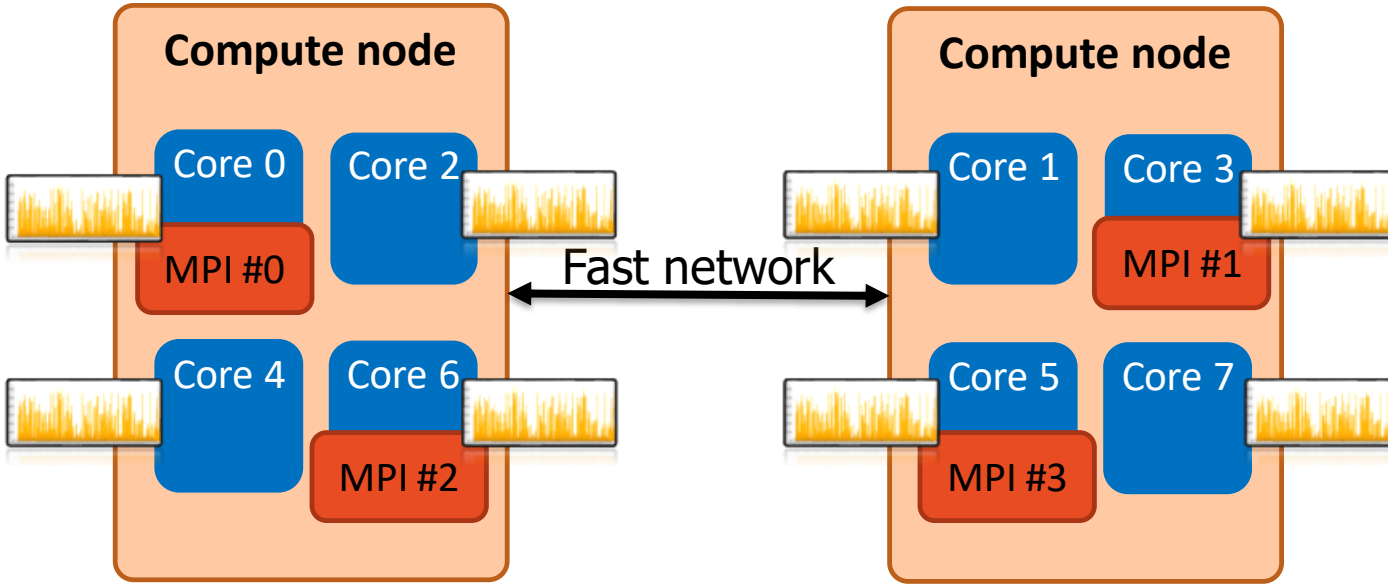


Correlation game:

- What can we know:
  - Which software (job) ran when & where
    - Batch systems knows
      - E.g.: Which MPI rank ran when and where?
  - Time-series of metrics at granularity
    - Metrics per scheduled entity

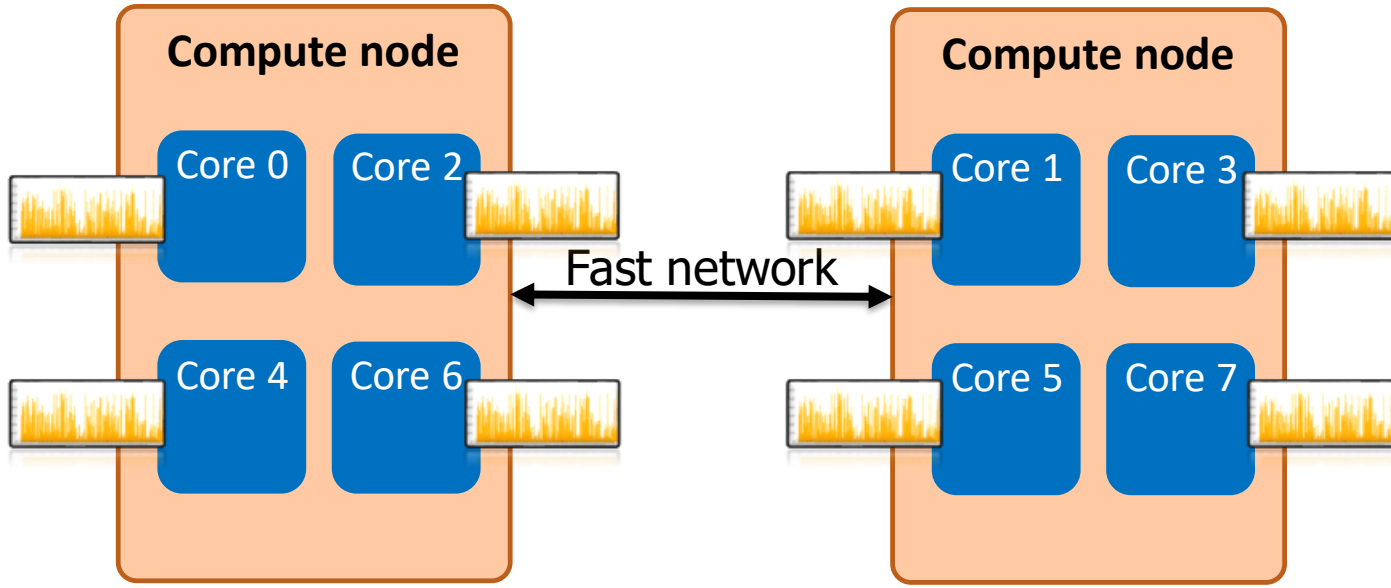
Tool to match and correlate data!





Correlation game:

- What can we know:
  - Which software (job) ran when & where
    - Batch systems knows
      - E.g.: Which MPI rank ran when and where?
  - Time-series of metrics at granularity
    - Metrics per scheduled entity



Tool to match and correlate data!

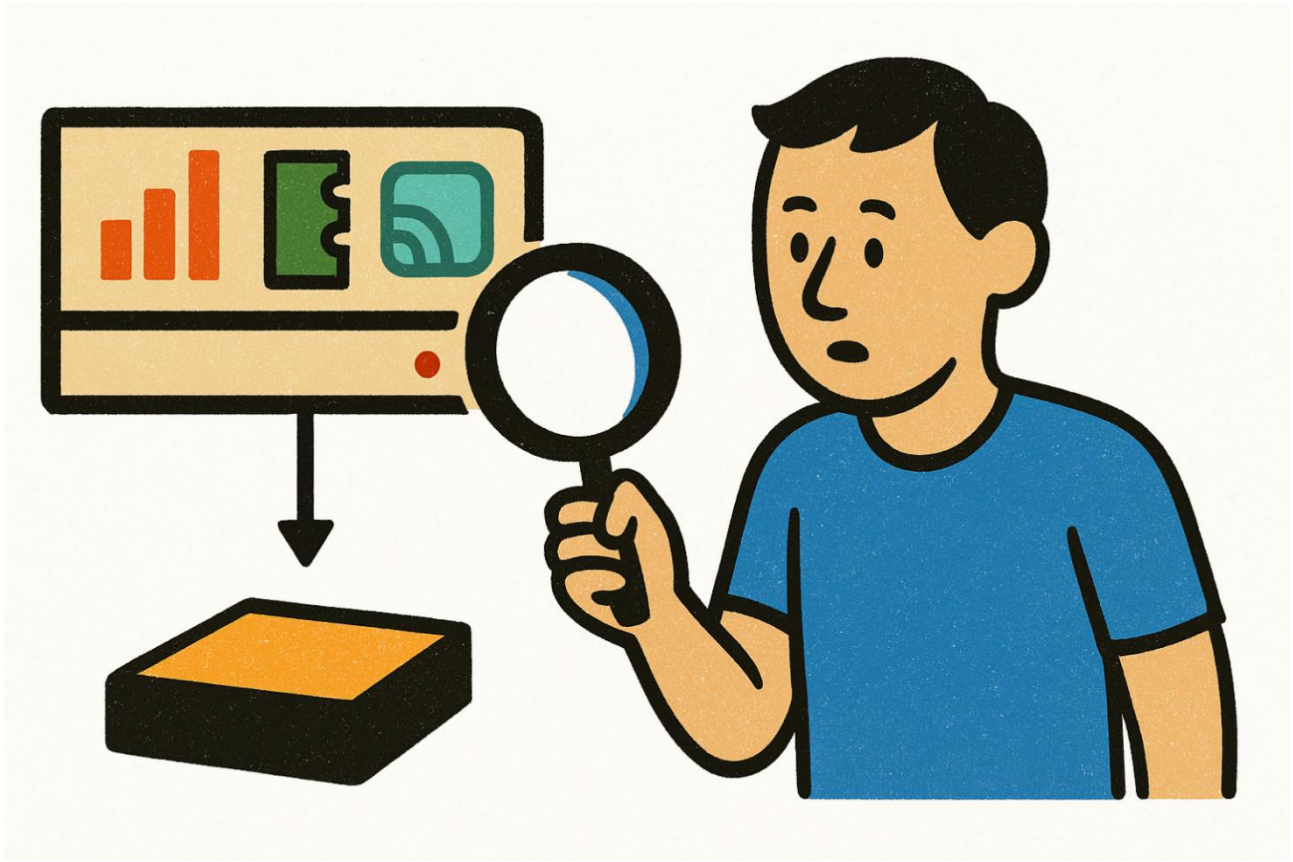
Metrics to measure … ?

- Mock-up cluster:
  - Node:
    - **CPU**: 1 — 2x AMD EPYC 9965 (192 cores) — ~ 8.000€ / piece
    - Motherboard: — Dual Socket Motherboard — ~ 1.600€ / piece
    - **RAM**: 2 — 24x 32GB DDR5-4800 (ECC) — ~ 270€ / piece
    - **Network**: 3 — InfiniBand HDR100 — ~ 850€ / piece
    - ~~GPGPU: NVIDIA H100 ~ 40.000€ / piece~~
    - Misc: — Cables, chassis, infrastructure, **storage** 4 — ~ 5000€ / node
  - Operational costs (1000 nodes):
    - Staff: — 7 FTE (full time employees) — ~ 80k€ / FTE / y
    - Power: — ~ 3.4 MW — ~ 67,70 € / MWh

• Node/h	~ 0,97€	Node/y	~ 8562€
• Cluster/h	~ 977,44€	Cluster/y	~ 8.562 M€

Metrics to measure?

- Any metric, that is related to individual hardware and usual bottlenecks:
  - **Utilization**
    - Allocation vs actual use
  - **CPU**
    - Compute unit utilization
    - Cache use
    - CPI / IPC
  - **Memory**
    - Memory use
    - Memory bandwidth
  - **Network**
    - Saturation
    - Packet size
  - **Storage**
    - IO operations



Cluster Cockpit

CusterCockpit: Concept and Purposes

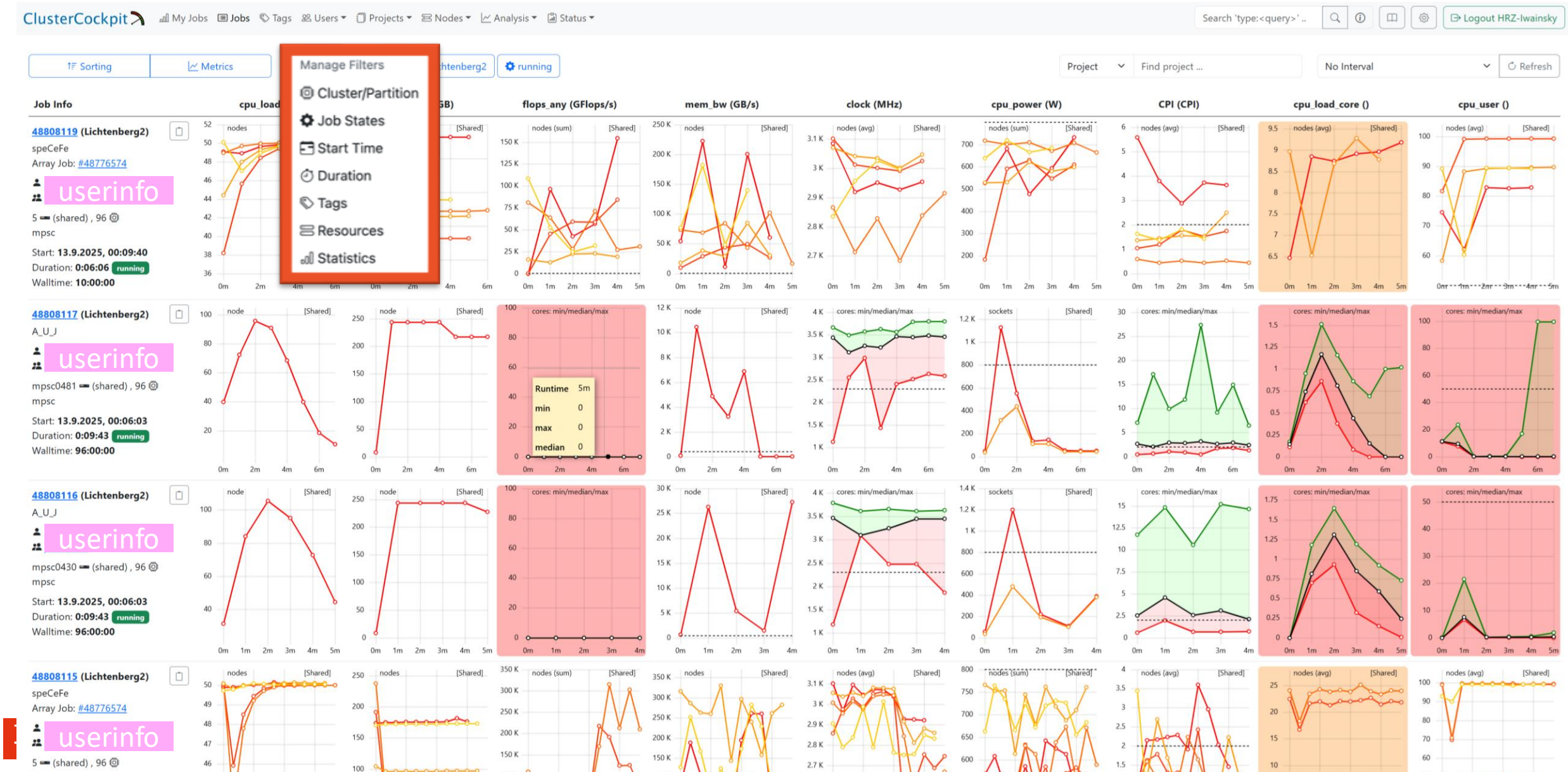
- Cluster-wide continuous monitoring
  - Hardware Performance Monitoring (HPM) metrics: CPI, GFlops/s, Memory bandwidth; load, memory usage, File IO, Network utilization, RAPL energy metrics, GPUs …
  - Fixed frequency measurements for native granularities (core, memory domain, socket, node)
- UI for
  - job monitoring,
  - early detection of performance or runtime issues
  - access to aggregate statistics for users and jobs
  - Web-interface (different views for roles: admin, support, manager, user)
    - Search, filter, tag, and sort jobs, users, and projects
    - Job-specific views with metric plots, job statistics, job meta information
    - Cluster-specific views with overall utilization, job and user statistics

https://github.com/ClusterCockpit/

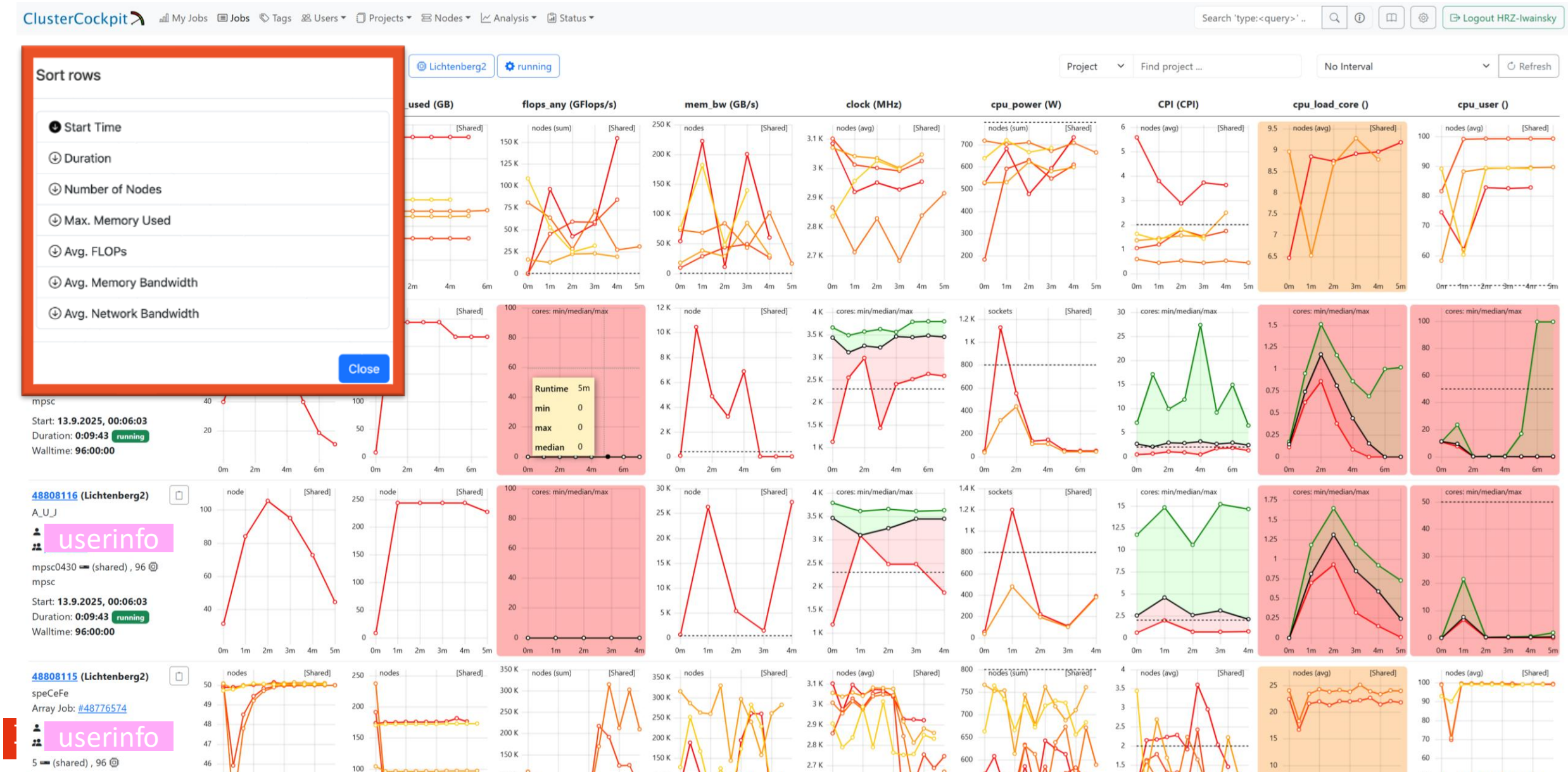
Friedrich-Alexander-Universität Erlangen-Nürnberg

NHR FAU

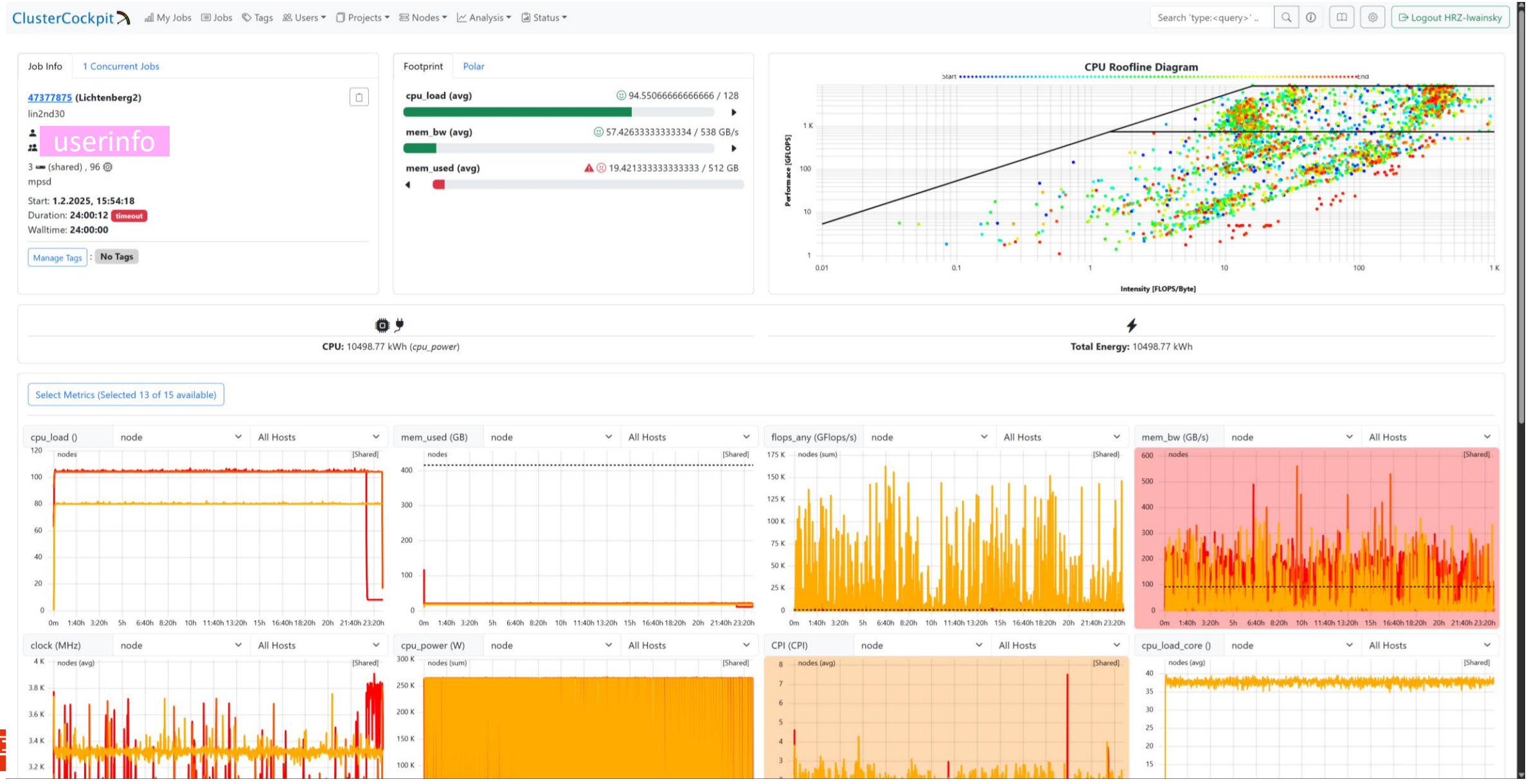
Cluster Cockpit: Job-list



Cluster Cockpit: Job-list

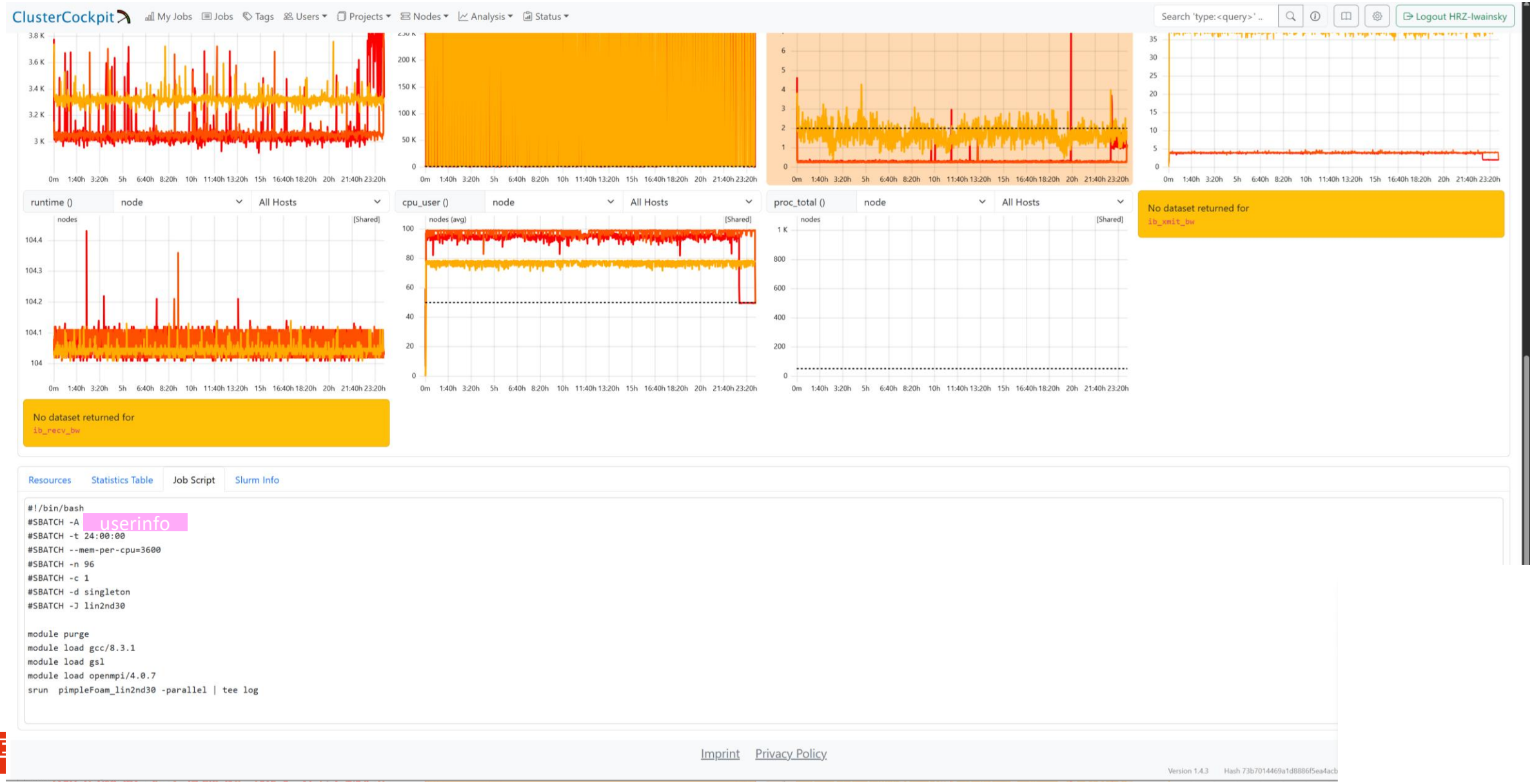


Cluster-Cockpit: Job-view



Cluster-Cockpit: Job-view

Cluster-Cockpit: Job-view

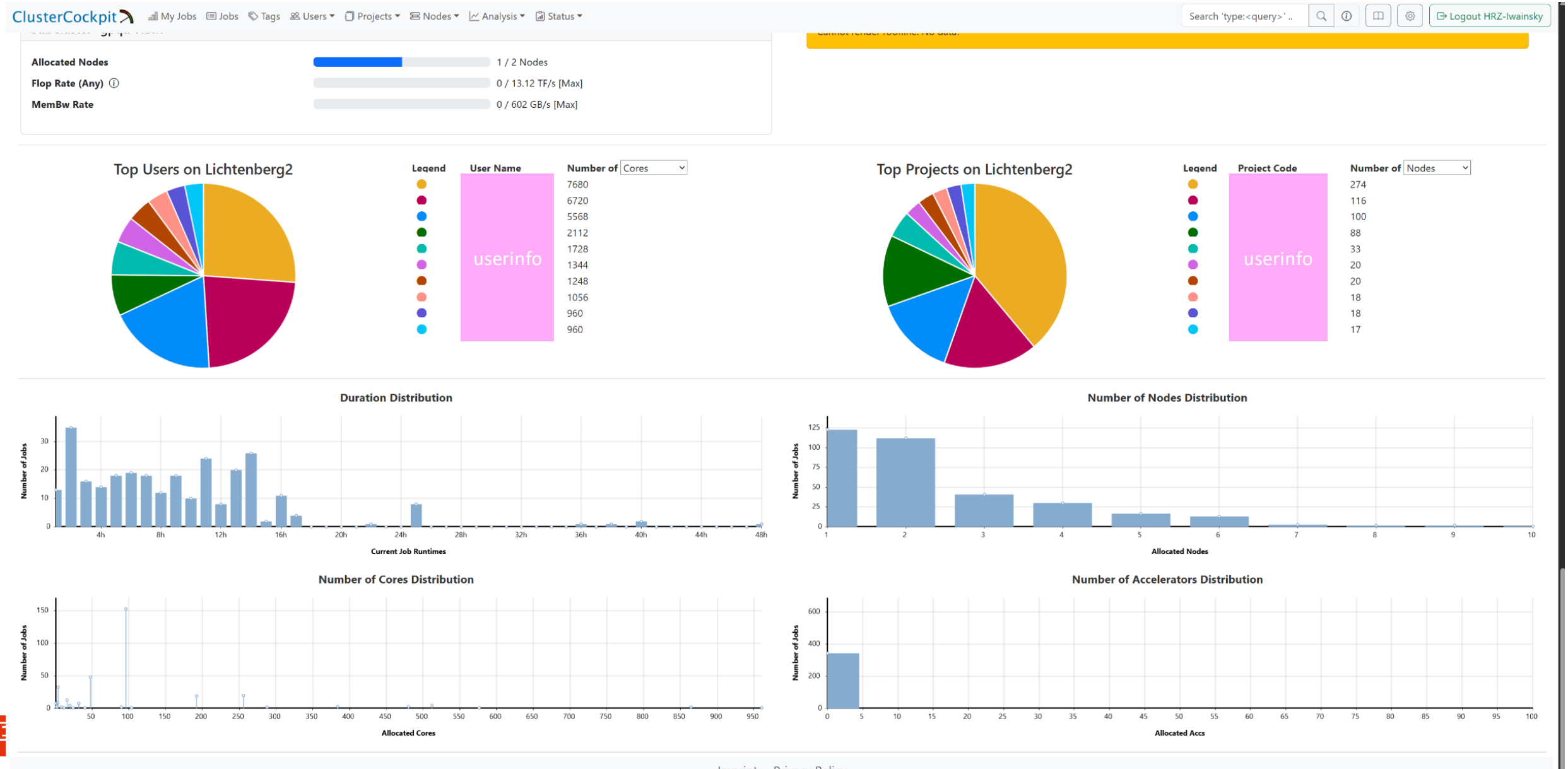


CusterCockpit: User-list (a project list is also available)

Search users | Filter by username | Filters | Last 30 Days

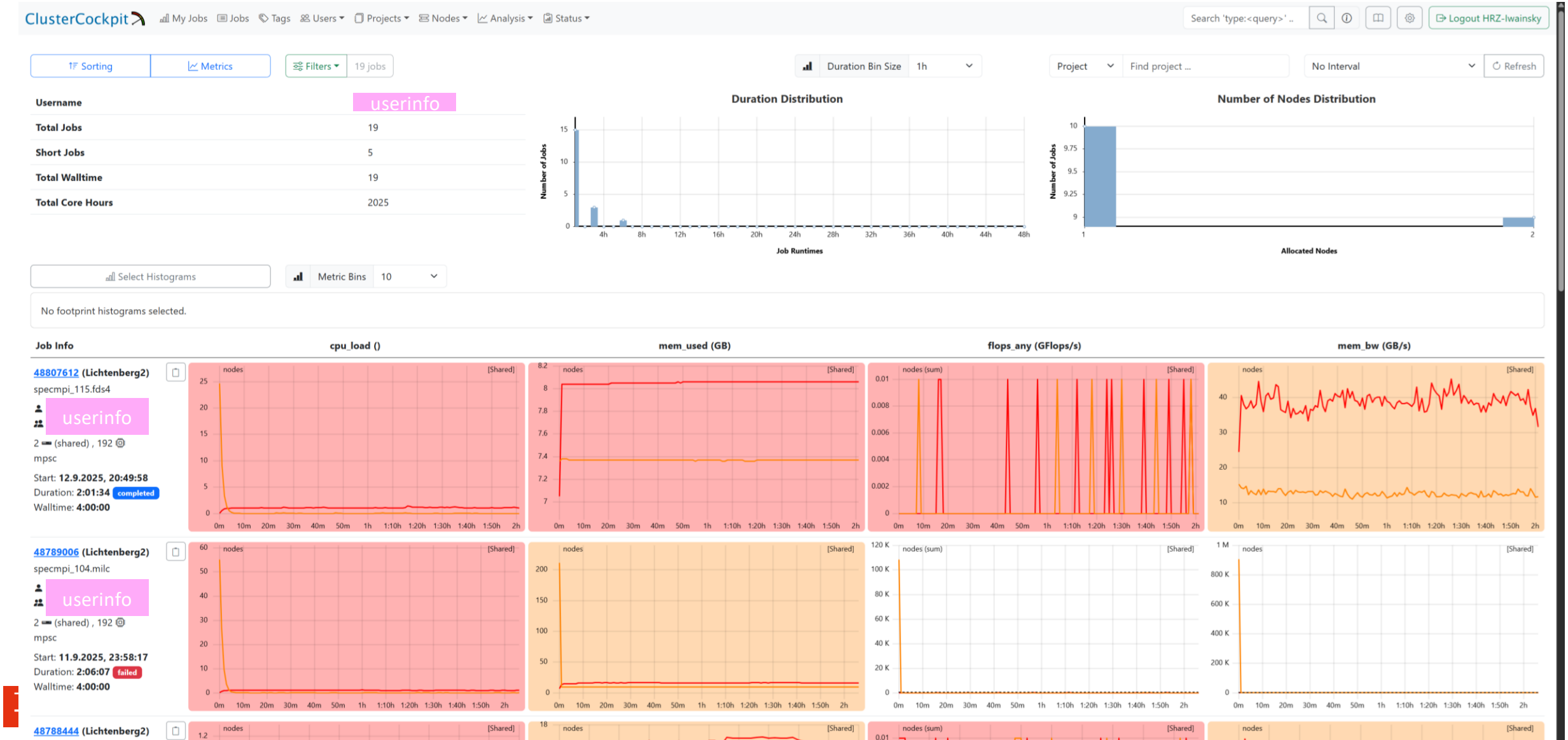
Username	Name	Total Jobs	Total Walltime	Total Core Hours	Total Accelerator Hours
userinfo	userinfo	16664	28376	28376	0
userinfo	userinfo	4159	25052	25052	0
userinfo	userinfo	1838	23682	378909	23681
userinfo	userinfo	119916	19073	32914	0
userinfo	userinfo	11302	18481	295709	18481
userinfo	userinfo	716	13705	986793	0
userinfo	userinfo	4234	13432	967142	0
userinfo	userinfo	1446	12146	194313	12146
userinfo	userinfo	3562	10032	10032	0
userinfo	userinfo	2826	9880	59281	0
userinfo	userinfo	533	9864	631300	0
userinfo	userinfo	763	9373	299900	0
userinfo	userinfo	3573	8197	3541186	0
userinfo	userinfo	1023	8186	130986	8186
userinfo	userinfo	3479	7811	141933	8870
userinfo	userinfo	5967	7431	118900	7431
userinfo	userinfo	761	7318	29274	0
userinfo	userinfo	314	6996	223874	13992

CusterCockpit: Cluster status view (aka management view)



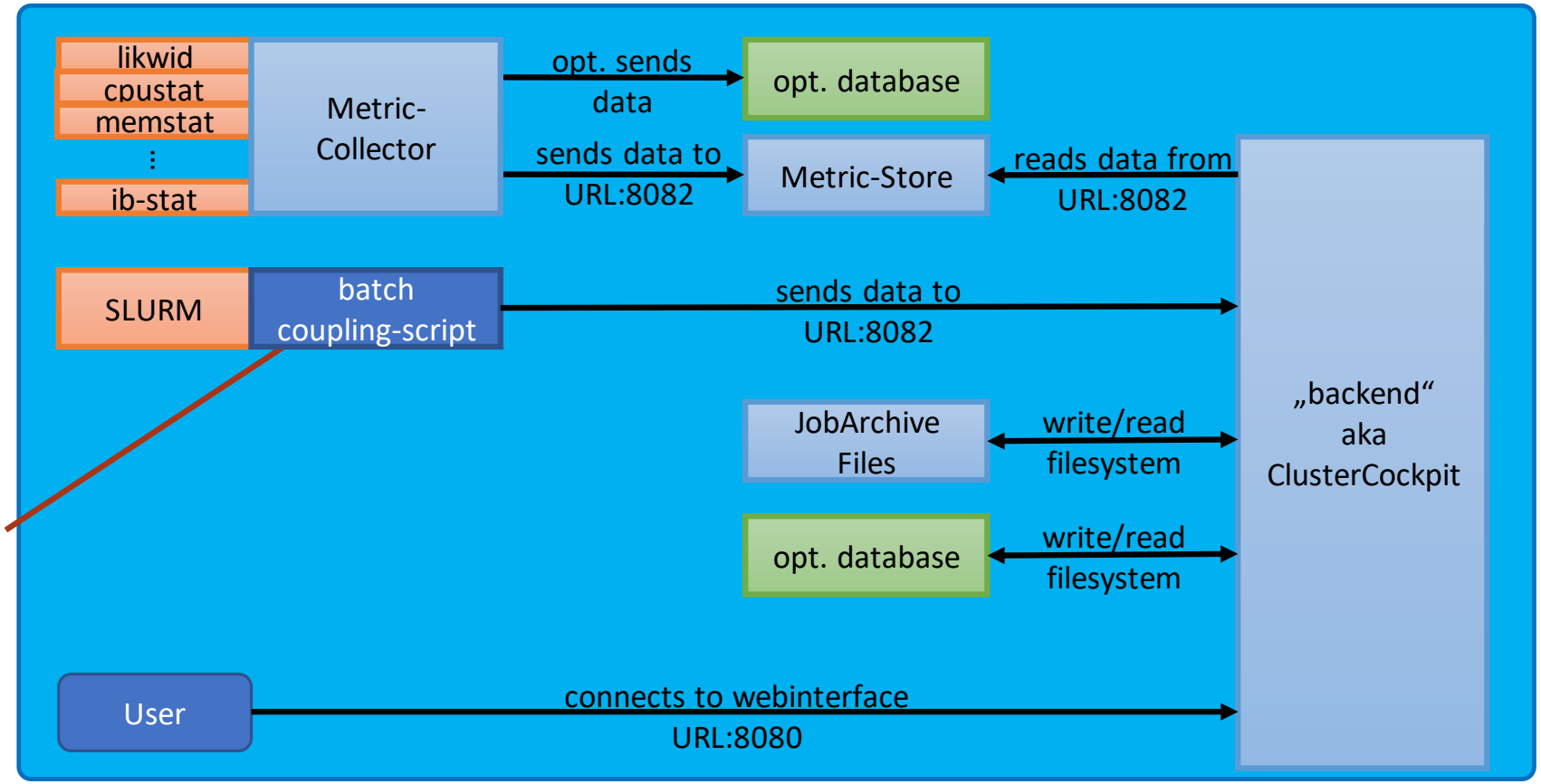
ClusterCockpit: Cluster-view

ClusterCockpit: User-view



ClusterCockpit: Components

- Components
  - cc-backend
  - cc-metric-collector
  - cc-metric-store
- Open-source MIT license

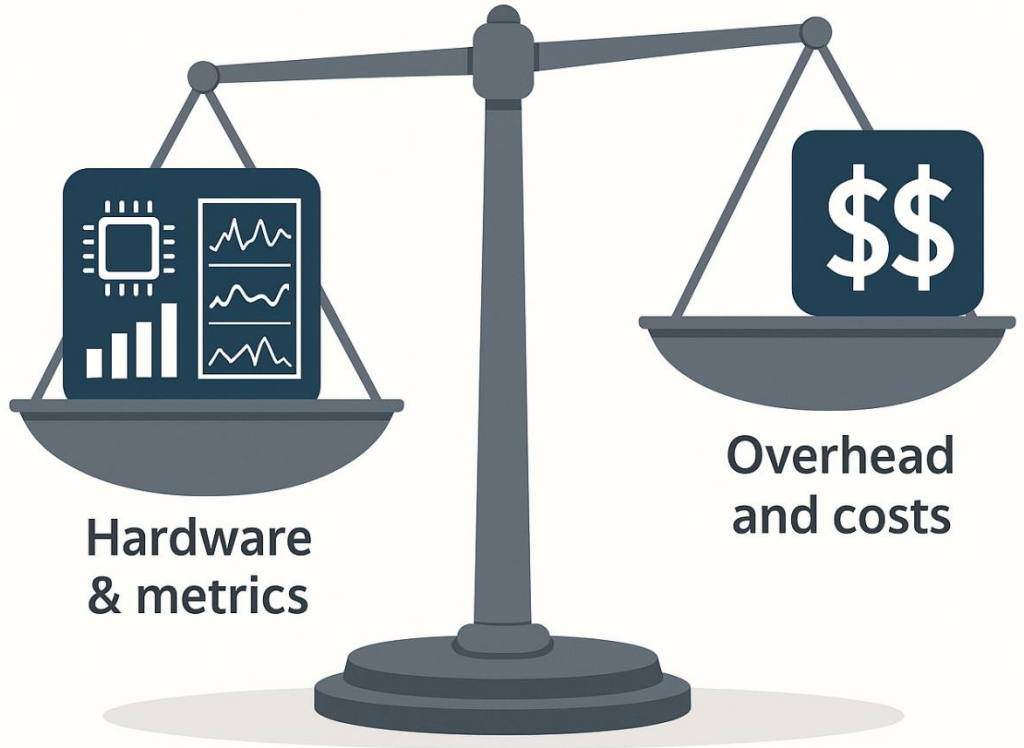


You must provide adapter, e.g.:
- Slurm commands
- Slurm REST API
- PrEp Plugin
- Existing SLURM coupling

https://github.com/ClusterCockpit/

… nice tool, but at what price?

- Metrics collectors at the node level
  - Collector Hardware Performance Counters / Core
    - 96 / 106 / 392 times the base overhead
  - System level metrics
    - Once per system
  - Data-transfer / node
- Metric Store
  - Data-sink for all nodes
  - O(2kB) data per node
  - Data retention for max job length
- Backend / Visualization

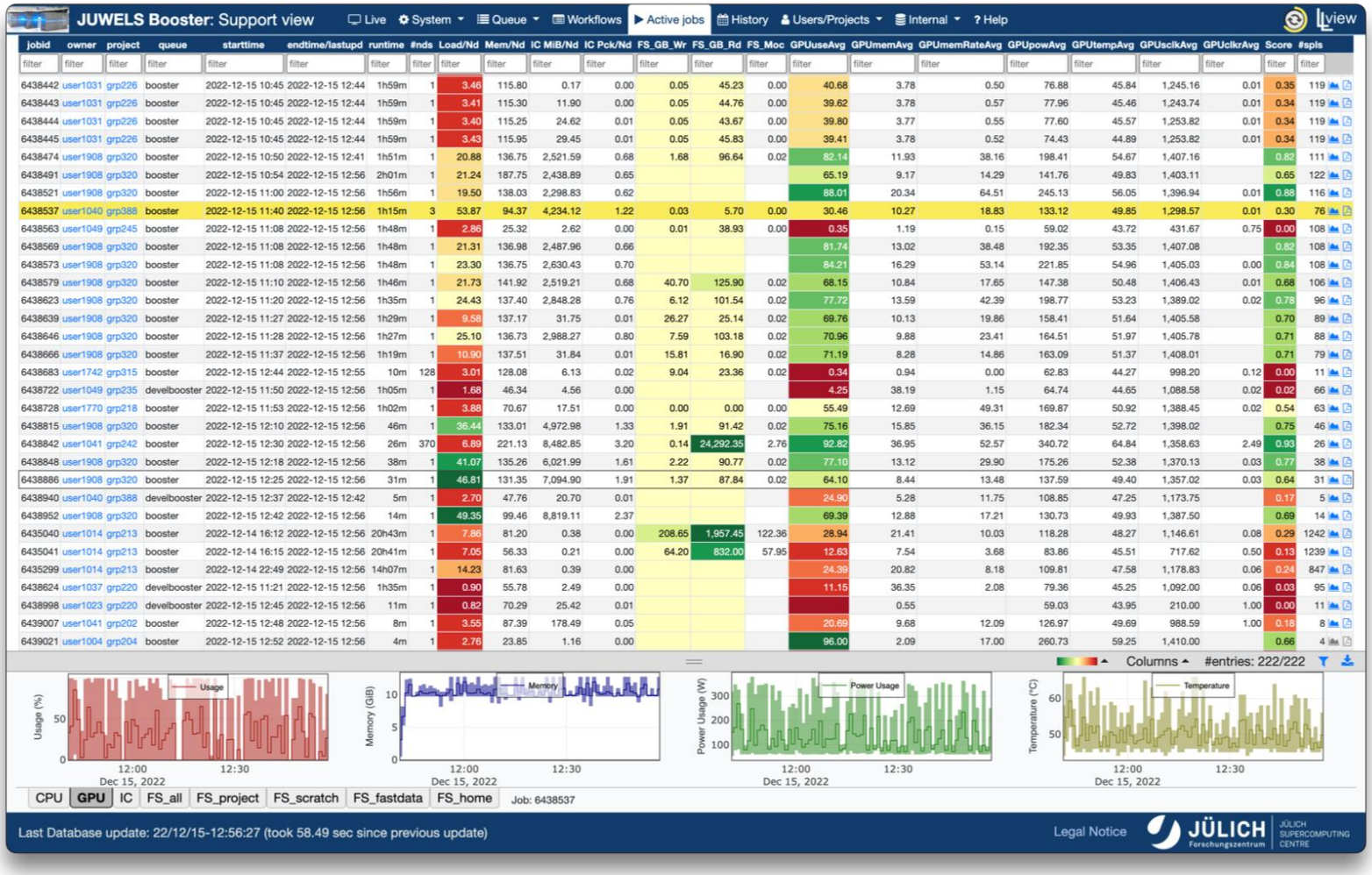


Alternatives?

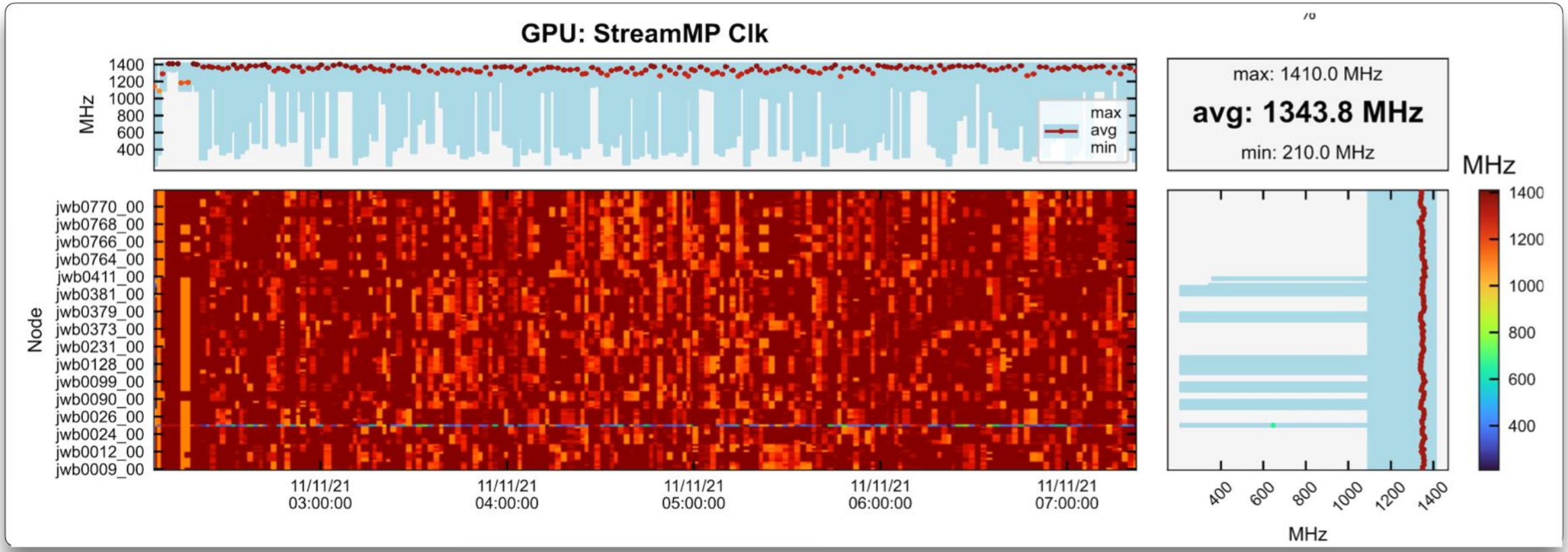
Other tools?

- Many tools
- Highlight:
  LLView by Jülich Research Center
  https://github.com/FZJ-JSC/LLview
  - System level views
  - Per Job statistics
  - No additional hardware counters
  - Not suited for shared nodes

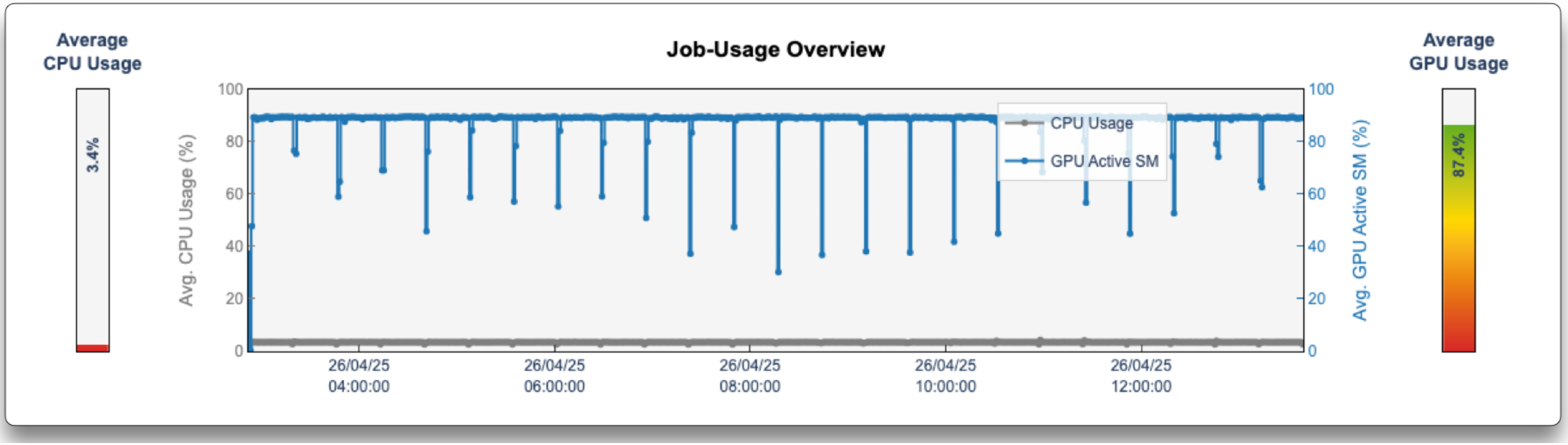
LLView


JUWELS Booster: Support view
Live
System
Queue
Workflows
Active jobs
History
Users/Projects
Internal
Help
lview
jobid owner project queue starttime endtime/lastupd runtime #nds Load/Nd Mem/Nd IC MiB/Nd IC Pck/Nd FS_GB_Wr FS_GB_Rd FS_Moc GPUuseAvg GPUmemAvg GPUmemRateAvg GPUpowAvg GPUtempAvg GPUsclkAvg GPUclkrAvg Score #spls
6438442 user1031 grp226 booster 2022-12-15 10:45 2022-12-15 12:44 1h59m 1 3.46 115.80 0.17 0.00 0.05 45.23 0.00 40.68 3.78 0.50 76.88 45.84 1,245.16 0.01 0.35 119
6438443 user1031 grp226 booster 2022-12-15 10:45 2022-12-15 12:44 1h59m 1 3.41 115.30 11.90 0.00 0.05 44.76 0.00 39.62 3.78 0.57 77.96 45.46 1,243.74 0.01 0.34 119
6438444 user1031 grp226 booster 2022-12-15 10:45 2022-12-15 12:44 1h59m 1 3.40 115.25 24.62 0.01 0.05 43.67 0.00 39.80 3.77 0.55 77.60 45.57 1,253.82 0.01 0.34 119
6438445 user1031 grp226 booster 2022-12-15 10:45 2022-12-15 12:44 1h59m 1 3.43 115.95 29.45 0.01 0.05 45.83 0.00 39.41 3.78 0.52 74.43 44.89 1,253.82 0.01 0.34 119
6438474 user1908 grp320 booster 2022-12-15 10:50 2022-12-15 12:41 1h51m 1 20.88 136.75 2,521.59 0.68 1.68 96.64 0.02 82.14 11.93 38.16 198.41 54.67 1,407.16 0.82 111
6438491 user1908 grp320 booster 2022-12-15 10:54 2022-12-15 12:56 2h01m 1 21.24 187.75 2,438.89 0.65 65.19 9.17 14.29 141.76 49.83 1,403.11 0.65 122
6438521 user1908 grp320 booster 2022-12-15 11:00 2022-12-15 12:56 1h56m 1 19.50 138.03 2,298.83 0.62 88.01 20.34 64.51 245.13 56.05 1,396.94 0.01 0.88 116
6438537 user1040 grp388 booster 2022-12-15 11:40 2022-12-15 12:56 1h15m 3 53.87 94.37 4,234.12 1.22 0.03 5.70 0.00 30.46 10.27 18.83 133.12 49.85 1,298.57 0.01 0.30 76
6438563 user1049 grp245 booster 2022-12-15 11:08 2022-12-15 12:56 1h48m 1 2.86 25.32 2.62 0.00 0.01 38.93 0.00 0.35 1.19 0.15 59.02 43.72 431.67 0.75 0.00 108
6438569 user1908 grp320 booster 2022-12-15 11:08 2022-12-15 12:56 1h48m 1 21.31 136.98 2,487.96 0.66 81.74 13.02 38.48 192.35 53.35 1,407.08 0.82 108
6438573 user1908 grp320 booster 2022-12-15 11:08 2022-12-15 12:56 1h48m 1 23.30 136.75 2,630.43 0.70 84.21 16.29 53.14 221.85 54.96 1,405.03 0.00 0.84 108
6438579 user1908 grp320 booster 2022-12-15 11:10 2022-12-15 12:56 1h46m 1 21.73 141.92 2,519.21 0.68 40.70 125.90 0.02 68.15 10.84 17.65 147.38 50.48 1,406.43 0.01 0.68 106
6438623 user1908 grp320 booster 2022-12-15 11:20 2022-12-15 12:56 1h35m 1 24.43 137.40 2,848.28 0.76 6.12 101.54 0.02 77.72 13.59 42.39 198.77 53.23 1,389.02 0.02 0.78 96
6438639 user1908 grp320 booster 2022-12-15 11:27 2022-12-15 12:56 1h29m 1 9.58 137.17 31.75 0.01 26.27 25.14 0.02 69.76 10.13 19.86 158.41 51.64 1,405.58 0.70 89
6438646 user1908 grp320 booster 2022-12-15 11:28 2022-12-15 12:56 1h27m 1 25.10 136.73 2,988.27 0.80 7.59 103.18 0.02 70.96 9.88 23.41 164.51 51.97 1,405.78 0.71 88
6438666 user1908 grp320 booster 2022-12-15 11:37 2022-12-15 12:56 1h19m 1 10.90 137.51 31.84 0.01 15.81 16.90 0.02 71.19 8.28 14.86 163.09 51.37 1,408.01 0.71 79
6438683 user1742 grp315 booster 2022-12-15 12:44 2022-12-15 12:55 10m 128 3.01 128.08 6.13 0.02 9.04 23.36 0.02 0.34 0.94 0.00 62.83 44.27 998.20 0.12 0.00 11
6438722 user1049 grp235 develbooster 2022-12-15 11:50 2022-12-15 12:56 1h05m 1 1.68 46.34 4.56 0.00 4.25 38.19 1.15 64.74 44.65 1,088.58 0.02 0.02 66
6438728 user1770 grp218 booster 2022-12-15 11:53 2022-12-15 12:56 1h02m 1 3.88 70.67 17.51 0.00 0.00 0.00 0.00 55.49 12.69 49.31 169.87 50.92 1,388.45 0.02 0.54 63
6438815 user1908 grp320 booster 2022-12-15 12:10 2022-12-15 12:56 46m 1 36.44 133.01 4,972.98 1.33 1.91 91.42 0.02 75.16 15.85 36.15 182.34 52.72 1,398.02 0.75 46
6438842 user1041 grp242 booster 2022-12-15 12:30 2022-12-15 12:56 26m 370 6.89 221.13 8,482.85 3.20 0.14 24,292.35 2.76 92.82 36.95 52.57 340.72 64.84 1,358.63 2.49 0.93 26
6438848 user1908 grp320 booster 2022-12-15 12:18 2022-12-15 12:56 38m 1 41.07 135.26 6,021.99 1.61 2.22 90.77 0.02 77.10 13.12 29.90 175.26 52.38 1,370.13 0.03 0.77 38
6438886 user1908 grp320 booster 2022-12-15 12:25 2022-12-15 12:56 31m 1 46.81 131.35 7,094.90 1.91 1.37 87.84 0.02 64.10 8.44 13.48 137.59 49.40 1,357.02 0.03 0.64 31
6438940 user1040 grp388 develbooster 2022-12-15 12:37 2022-12-15 12:42 5m 1 2.70 47.76 20.70 0.01 24.90 5.28 11.75 108.85 47.25 1,173.75 0.17 5
6438952 user1908 grp320 booster 2022-12-15 12:42 2022-12-15 12:56 14m 1 49.35 99.46 8,819.11 2.37 69.39 12.88 17.21 130.73 49.93 1,387.50 0.69 14
6435040 user1014 grp213 booster 2022-12-14 16:12 2022-12-15 12:56 20h43m 1 7.86 81.20 0.38 0.00 208.65 1,957.45 122.36 28.94 21.41 10.03 118.28 48.27 1,146.61 0.08 0.29 1242
6435041 user1014 grp213 booster 2022-12-14 16:15 2022-12-15 12:56 20h41m 1 7.05 56.33 0.21 0.00 64.20 832.00 57.95 12.63 7.54 3.68 83.86 45.51 717.62 0.50 0.13 1239
6435299 user1014 grp213 booster 2022-12-14 22:49 2022-12-15 12:56 14h07m 1 14.23 81.63 0.39 0.00 24.39 20.82 8.18 109.81 47.58 1,178.83 0.06 0.24 847
6438624 user1037 grp220 develbooster 2022-12-15 11:21 2022-12-15 12:56 1h35m 1 0.90 55.78 2.49 0.00 11.15 36.35 2.08 79.36 45.25 1,092.00 0.06 0.03 95
6438998 user1023 grp220 develbooster 2022-12-15 12:45 2022-12-15 12:56 11m 1 0.82 70.29 25.42 0.01 0.55 59.03 43.95 210.00 1.00 0.00 11
6439007 user1041 grp202 booster 2022-12-15 12:48 2022-12-15 12:56 8m 1 3.55 87.39 178.49 0.05 20.69 9.68 12.09 126.97 49.69 988.59 1.00 0.18 8
6439021 user1004 grp204 booster 2022-12-15 12:52 2022-12-15 12:56 4m 1 2.76 23.85 1.16 0.00 96.00 2.09 17.00 260.73 59.25 1,410.00 0.66 4
Columns
#entries: 222/222
Usage
Usage (%)
Memory
Memory (GiB)
Power Usage
Power Usage (W)
Temperature
Temperature (°C)
12:00
12:30
Dec 15, 2022
CPU
GPU
IC
FS_all
FS_project
FS_scratch
FS_fastdata
FS_home
Job: 6438537
Last Database update: 22/12/15-12:56:27 (took 58.49 sec since previous update)
Legal Notice
JÜLICH
Forschungszentrum
JÜLICH SUPERCOMPUTING CENTRE

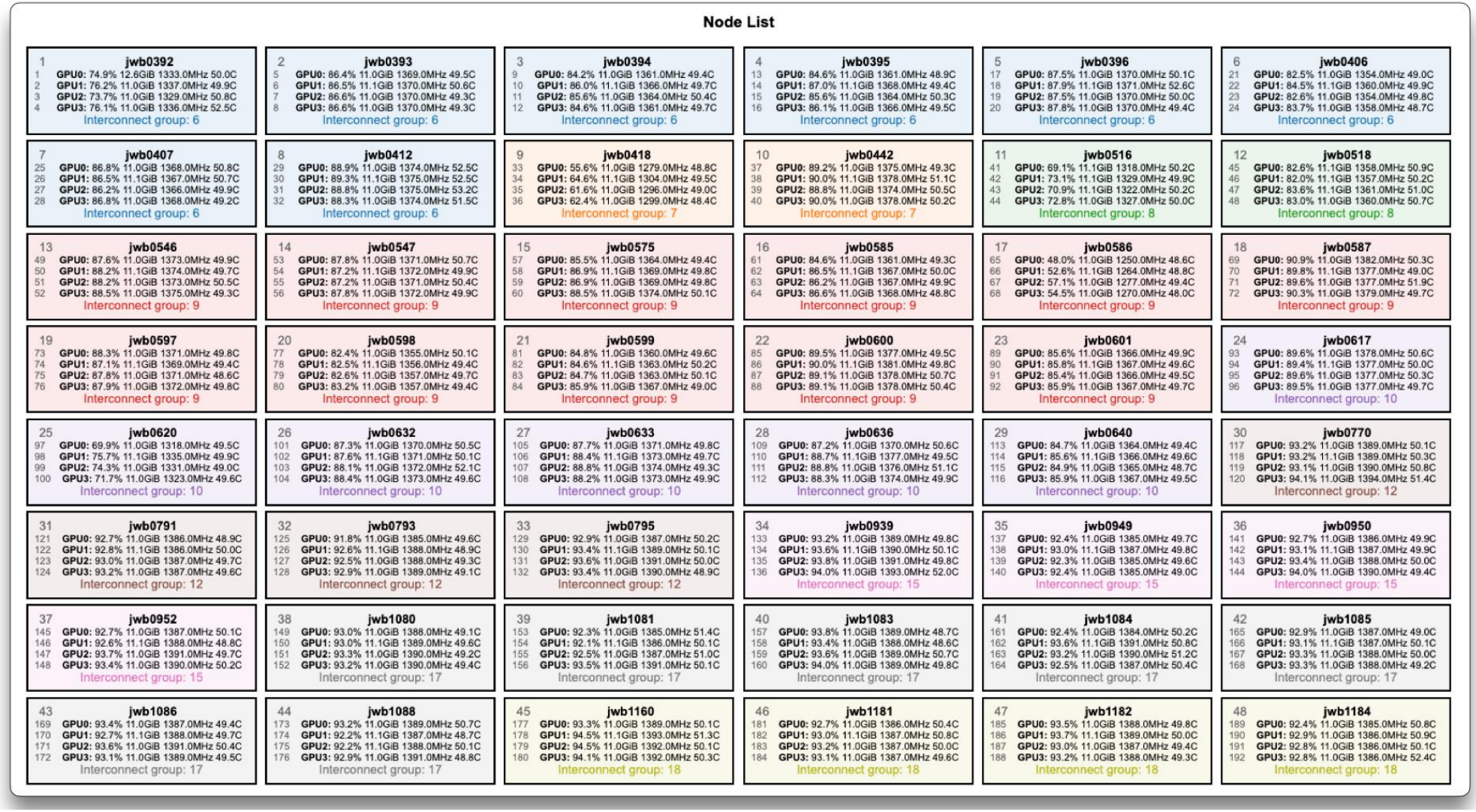
LLView: GPU stats



LLView: Job Usage



LLView: Node list



Other tools?

- LLView by Jülich Research Center
  https://github.com/FZJ-JSC/LLview
  - Sophisticated system level breakdown
    - Not for shared nodes
    - No hardware performance counters

- Different choices available, pick your solution

PathoJobs

Patho-Jobs

- Goal:
  - Automated rule-based detection system
  - Detection of pathological HPC jobs
    - jobs with previously encountered inefficiencies
  - Target audience: cluster operators and users

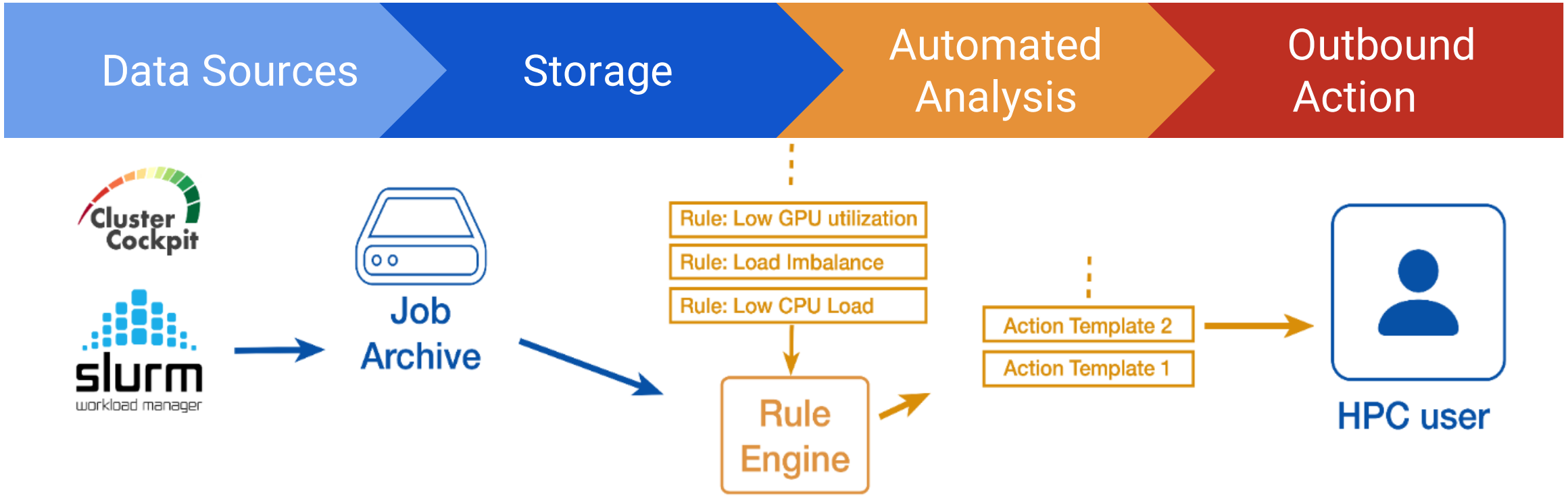
Collaboration with:

Paderborn Center for Parallel Computing

NHR FAU

ZIB ZUSE INSTITUTE BERLIN

https://git-ce.rwth-aachen.de/pathojobs/



- Cluster Cockpit generates a job-archive for each job;
  - alternative sources experimented with, but not ready to use
- Rule-Engine applies anti-pattern and triggers appropriate response for each job
- Detected anti-patterns trigger action templates (Email or job-archive tags)

PathoJobs: Prototype antipatterns

- Low CPU load
- Load-Imbalance
- Job Payload Overhead
- Excessive CPU load
- Problematic memory usage
- Uncoordinated multi-process GPU-usage
- Low GPU utilization
- Unnecessary job distribution
- Unnecessary PFS read

Patho-Jobs: **Low-Load** Pattern

```
{
 "name":"Low CPU load",
 "tag":"lowload",
 "parameters": ["threshold_factor"],
 "metrics": ["cpu_load, job"],
 "terms":[
 { "load_mean":     "cpu_load.mean('all')"},
 { "load_threshold":"job.numHwthreads  * threshold_factor"},
 { "lowload_nodes": "load_mean < load_threshold"},
 { "lowload":       "lowload_nodes.any('all')"},
 { "load_perc":     "1.0 - (load_mean / load_threshold)"}],
  "output":"lowload",
  "output_scalar":"load_perc",
  "template":"Job ({{job.jobId}}) was detected as the mean cpu
  load {{load_mean}} falls below {{load_threshold}}."
}
```

Tag for use in ClusterCockpit

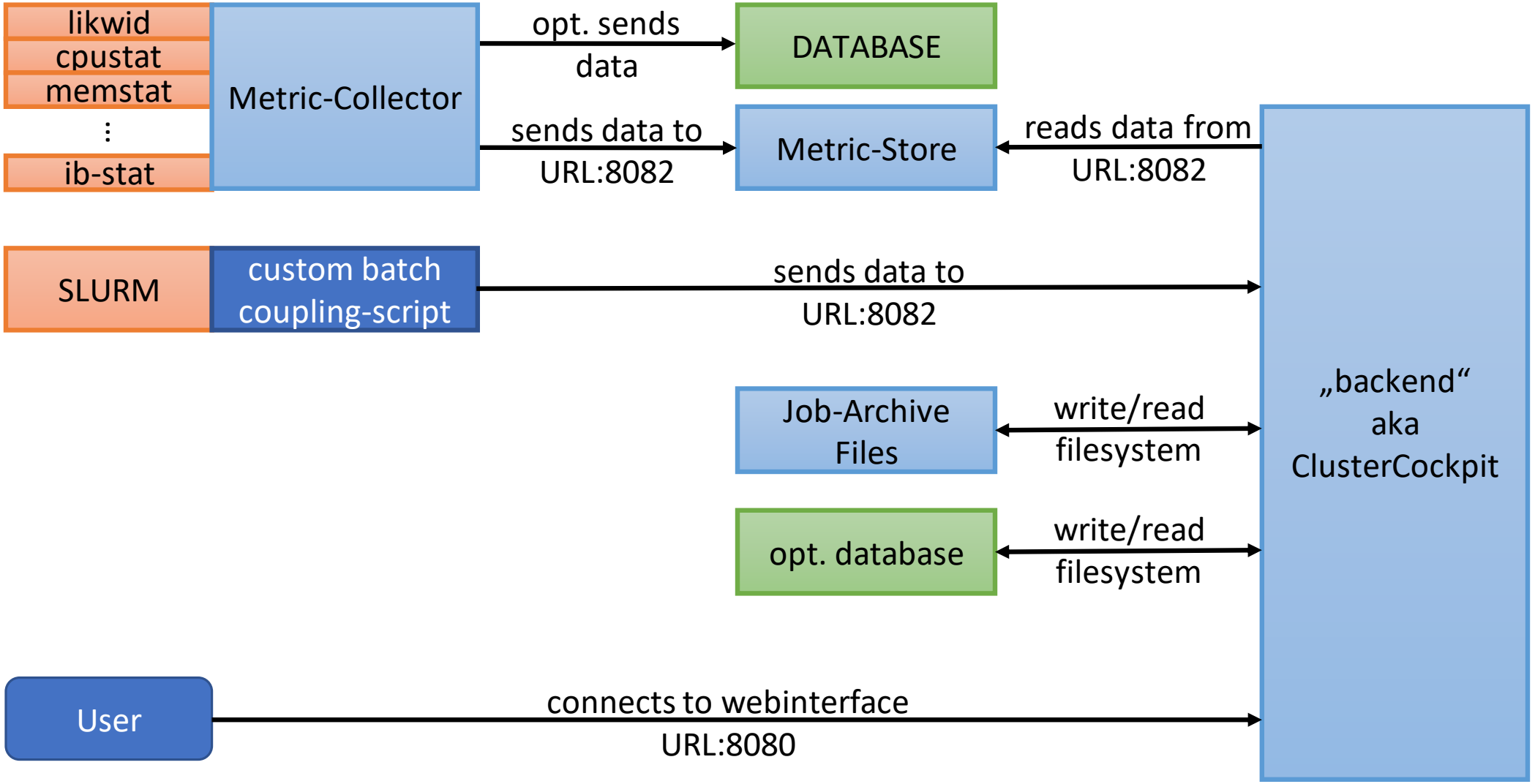
External configurable thrshold

Which datapoints to pull from job-archive

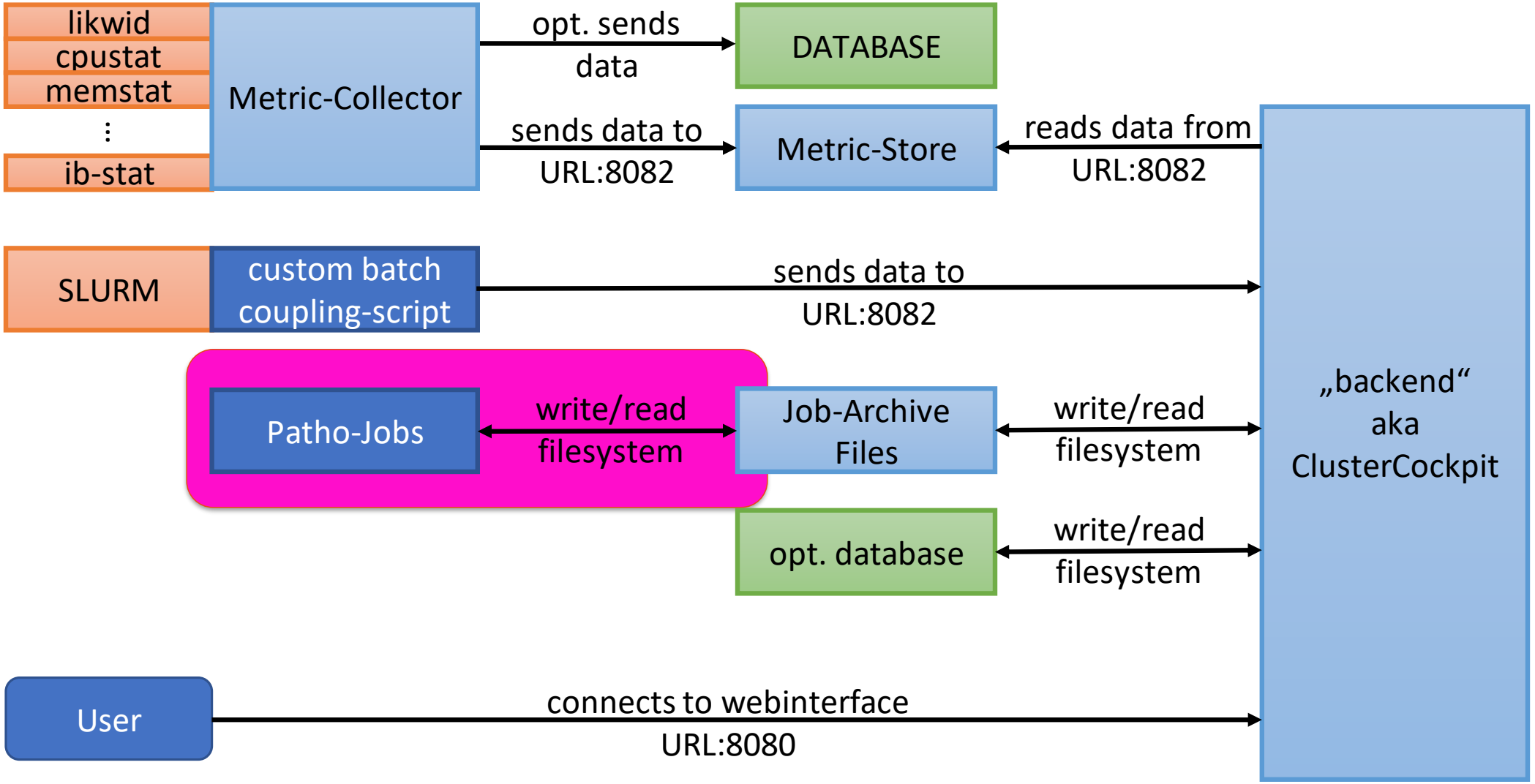
Variables initialized with in sequence evaluation of terms

Pattern output, variables can be used

Classic CusterCockpit:



Patho-Jobs extension:



- Deployment:
  - The "PathoJobs" system is deployed by partners on production HPC systems.
  - Each processes job-archives from hundred-thousands to millions of jobs per month.
  - Current performance anti-patterns are evaluated for all HPC jobs.

- First Experience & Insights:
  - Rule evaluation does not exhibit noticeable extra-load on the Cluster Cockpit hosts
  - Number of jobs with detected inefficiencies exceeds acceptable rates
  - System is able to detect explicitly malformed test-jobs

Location	Jobs / month	Jobs with issues	Rules with highest hit-rate	Analysis cost
TU Darmstadt	~ 250k	598 / 1000	Low CPU load Load-imbalance	230 ms / job
Paderborn	~ 190k	180 / 1000	Low CPU load CPU oversubscription	560 ms / job
FAU	~ 255k	326 / 1000	Low CPU load Low resource utilization	n.a.

Experience …

- Software can be directly deployed from binaries:
  - Did not attempt to compile on HPC system
  - 3 types of services:
    - metric-collector per compute node,
    - at least one metric store
    - backend/UI service
- Basic setup:
  - cluster configuration: specification of the cluster for the backend; **tools help**
  - metric collector: select which metrics are of interest, **challenging**
    - managing CPU registers, trade-off available metric sources, manage access priviliges
  - metric store: holds metrics; must match with metric collector and backend; assertaining peak capacity is not trivial

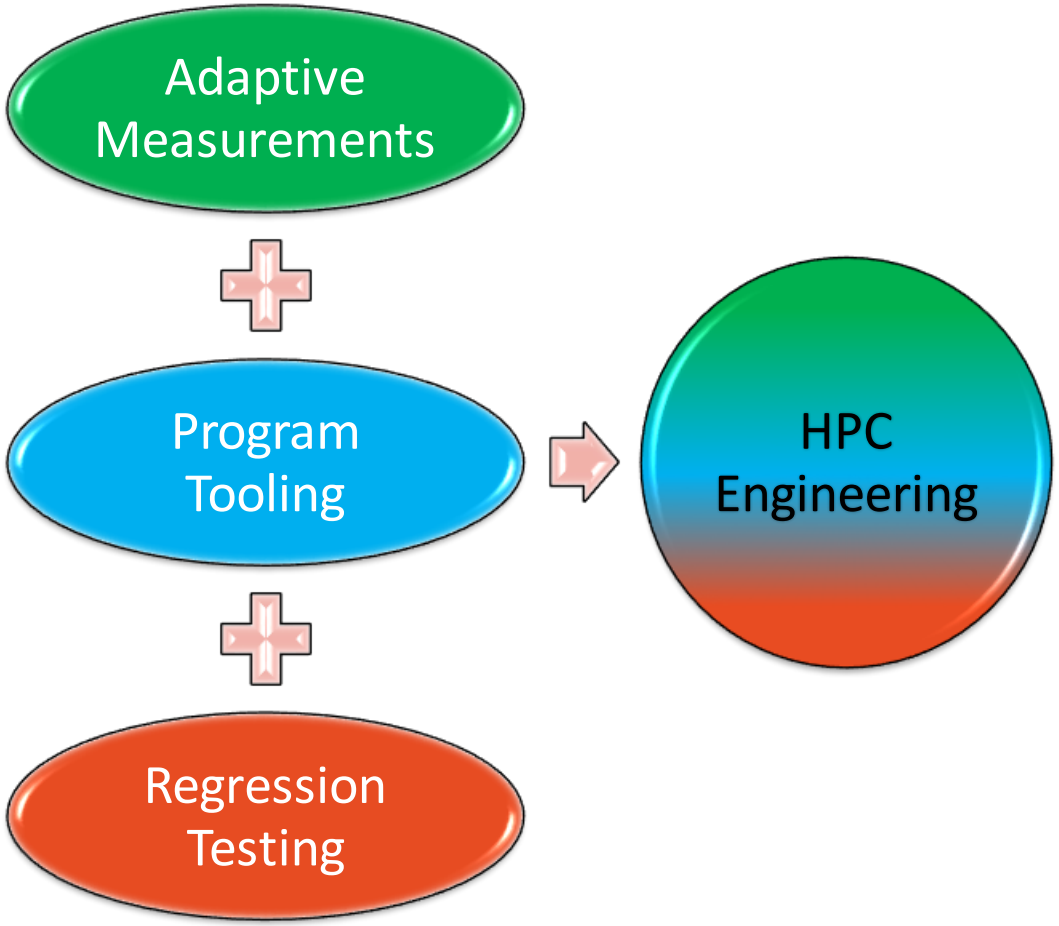
Why is that DevOps?

DevOps principle	How It applies in this talk
Observability	Job-level monitoring with ClusterCockpit and hardware counters
Automation	PathoJobs automatically detects inefficient jobs using rule engines
Feedback Loops	Tagged jobs, performance summaries, and user-facing metrics
Scalability	Designed for 10,000+ jobs/day with minimal overhead
Collaboration	Shared infrastructure for admins, users, and support staff
Open Tooling	Open-source, API-driven, SLURM-integrated - DevOps-ready

Key Takeaways – Boosting HPC with Job-Specific Monitoring

- **HPC systems require more than just system monitoring**
  → Insightful resource usage must happen at the job level, not just the node.
- **Instrumentation is powerful, but doesn't scale**
  → We need lightweight, non-invasive observability for real-world workloads.
- **ClusterCockpit enables scalable job-aware monitoring**
  → Tied to users, jobs, and projects — not just hosts and hardware.
- **PathoJobs adds automation and rule-based diagnosis**
  → Proactive detection of inefficiencies from 10,000+ jobs/day.
- **HPC DevOps is not just CI/CD — it's feedback, insight, and collaboration**
  → Observability and automation must become part of everyday HPC operations.

Where are things going …



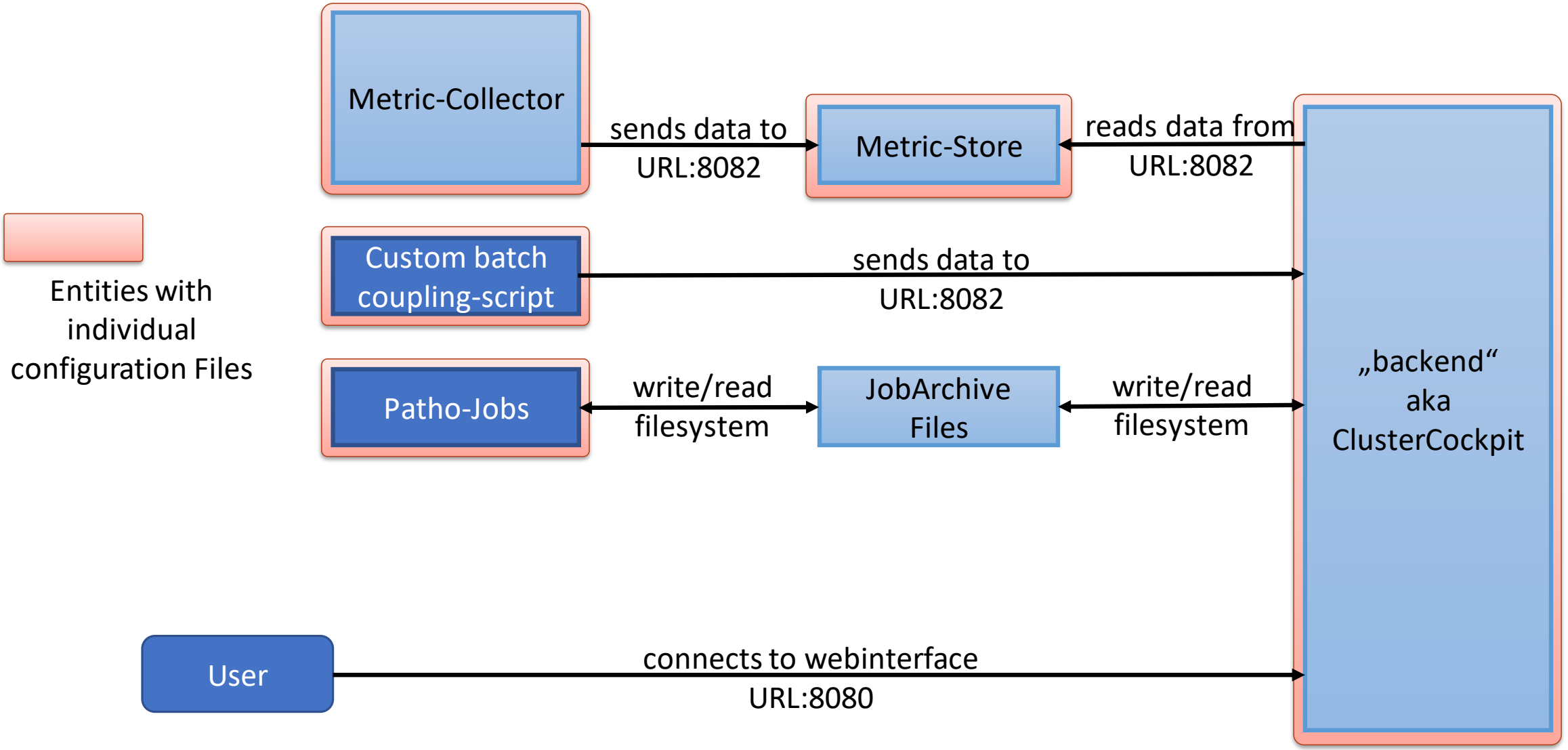
- Adaptive measurements
  - InstRO, CAPI, XRAY
    - All applications come pre-tooled for performance analysis, but inactive
    - When more data is required, it detailed measurements are seamless activated
- Program tooling
  - CALIPER style integration
    - Application reports what it is doing
- Performance regressions tests
    - Integrate „self reflection“ into programs and communicate results to external viewers
    - Keyword: Performance assertions/active program documentation
    - Extend PathoJobs approch

Key Takeaways – Boosting HPC with Job-Specific Monitoring

- **HPC systems require more than just system monitoring**
- **Instrumentation is powerful, but doesn't scale**
- **ClusterCockpit enables scalable job-aware monitoring**
- **PathoJobs adds automation and rule-based diagnosis**
- **HPC DevOps is not just CI/CD — it's feedback, insight, and collaboration**

Questions?

Individual configuration files:



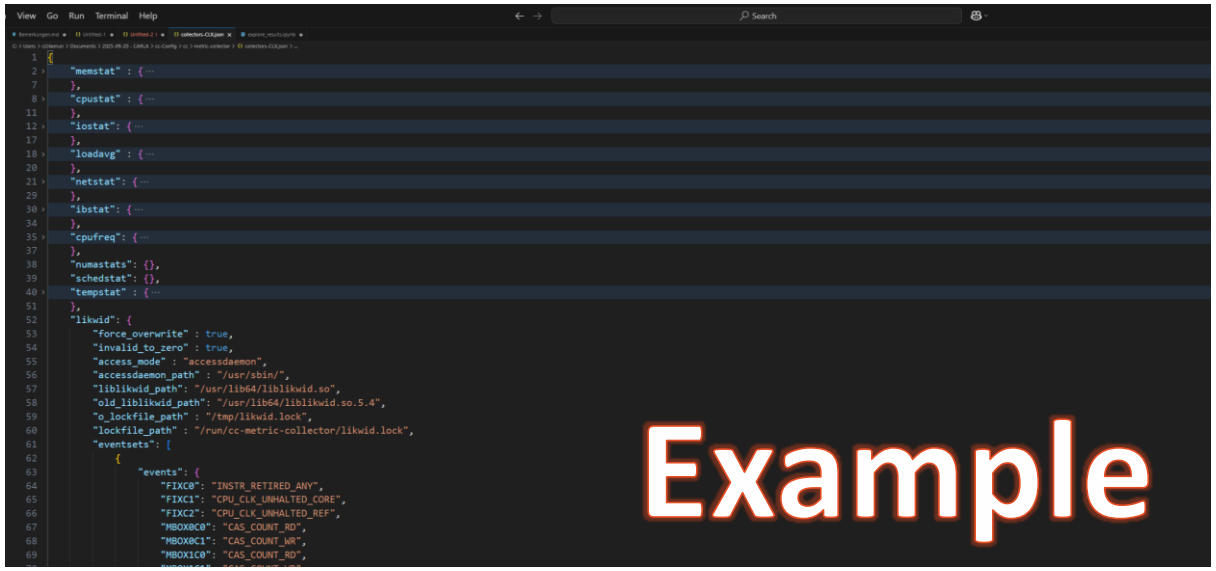
Metric Collector

- Collector config:

```
{
  "sinks": "sinks-http.json",
  "collectors" : "collectors-CLX.json",
  "receivers" : "receivers.json",
  "router" : "router.json",
  "interval": "60s",
  "duration": "1s"
}
```

Example

- Collectors:



Metric-Store

- Metric-Store config:

```
{
  "metrics": {
    "clock":           {"frequency": 60, "aggregation": "avg"},
    "cpu_user":        {"frequency": 60, "aggregation": "avg"},
    "acc_utilization":{"frequency": 60, "aggregation": "avg"},
    "acc_mem_used":    {"frequency": 60, "aggregation": "sum"},
    "mem_used":        {"frequency": 60, "aggregation": null }
  },
  "checkpoints": {
    "interval": "12h","directory": "/work/home/hrzcc/var/metric-store/checkpoints",
    "restore": "168h"
    },
  "archive": {
    "interval": "168h",
    "directory": "/work/home/hrzcc/var/metric-store/archive"
    },
  "http-api": {
    "address": "0.0.0.0:8082",
    "https-cert-file": null,
    "https-key-file": null
    },
  "retention-in-memory": "168h",
  "nats": null,
  "jwt-public-key":"use-your-own-key"
}
```

Example

Backend

2+ Config-files:

- Cluster-configurations:
  - Helper: generate-subcluster.pl
- Backend-configuration:

Cluster-config

Backend-config