SEANERGYS
Energy Efficient Exascale

# D1.1 - Design Workshop

## Document Properties

| | |
|---|---|
| Contract Number | 101177590 |
| Contractual Deadline | M5 (October 31, 2025) |
| Dissemination Level | Public |
| Nature | Report |
| Author(s) | M. Schulz (TUM), H.-C. Hoppe (FZJ), S. Maloney (FZJ), P. Pochelu (LXP), U. Sinha (FZJ) |
| Contributor(s) | Representatives from all partners participating at the workshop |
| Reviewers | M. Ott (BAdW-LRZ), E. Suarez (FZJ) |
| Date | October 31, 2025 |
| Keywords | SEANERGYS, HPC, Exascale, Software, Energy Efficiency |
| Status | Final |
| Release | 1.0 |

*The SEANERGYS project receives funding from the European High Performance Computing Joint Undertaking (JU) under grant agreement no 101177590. The JU receives support from the European Union's Horizon Europe research and innovation programme and Czechia, France, Germany, Greece, Italy, and Spain. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or of the granting authority. Neither the European Union nor the granting authority can be held responsible for them.*

## Document Status Sheet

| Release | Date | Author, Organisation | Description of Changes |
|---|---|---|---|
| 1.0 | 31.10.2025 | | Final version submitted to EuroHPC JU |

# Table of Contents

## List of Figures



## List of Tables

# Executive Summary

SEANERGYS conducted a two-day Design Workshop on September 18 and 19, 2025 at Forschungszentrum Jülich. This event enabled the consortium to agree on and practice a strict GitLab-based process for defining use cases, requirements and architectural decisions, and it made excellent progress in describing first use cases, requirements and architecture decisions. This deliverable describes the structure of the workshop, paraphrases the talks presented and discussions which took place, and enumerates the results created by the event.

# 1 Introduction: The SEANERGYS Design Workshop

As one of its first tasks, WP1 organised the SEANERGYS Design Workshop with the objective of discussing the core use cases for the project, identifying their inner workings, and refining the initial architecture developed during the proposal process. Further, we discussed and agreed on a traceable, GitLab-based process for defining use cases, requirements and architecture decisions, reviewed all components brought into the project, identified overlaps, and started mapping them to the developed architecture.

The project will use the Architecture Decision Record (ADR) concept to document and track decisions on and the evolution of use cases, requirements, and the architecture. A key result of the design workshop was the agreement on a GitLab-based design process, which is fundamental for the continued work and the success of SEANERGYS. To train participants and give them the opportunity to acquaint themselves and practice, the process was used within the workshop to document early decisions and results.

This document takes the reader through the organisation of the workshop, discusses the agreed upon ADR-based decision process, and presents excerpts from the plenary and breakout sessions. It further details the list of use cases and early architecture discussions/decisions, as well as the status of discussion regarding the selection of a scheduling/resource management system to base the SEANERGYS work in WP4 on.

# 2 Design Workshop Organisation and Agenda

This chapter describes organisational aspects of the workshop, the participation, and the agenda.

## 2.1 Organisational Aspects

The SEANERGYS Design workshop was held at the premises of Forschungszentrum Jülich (FZJ) on September 18 and 19, 2025. To host the expected about 40 physical participants and enable holding three-way breakout sessions, two large meeting rooms (one for day 1, the second for day 2) and four breakout rooms were reserved. To let SEANERGYS participants who were unable to attend face–to–face participate in the presentations and discussions, virtual meetings for all plenary and breakout sessions were set up using Microsoft Teams and Zoom. This offer was taken up by on average 10-15 participants.

Minutes of the sessions were kept using the online Hedgedoc markup system, and all results and artefacts were stored and edited using the Jülich Supercomputing Centre's GitLab instance. All participants, whether face–to–face or online, had equal access to these materials.

FZJ provided sustenance to the participants in the form of four coffee breaks, two group lunches, and a social dinner on the evening of the first day. When signing in, participants agreed not to claim any expenses related to sustenance for the workshop days.

## 2.2 Participation

The workshop was attended by 40 physical participants on Day 1, and 41 colleagues on Day 2. All of the SEANERGYS partners were represented by at least one person throughout the workshop. Figure 1 shows a group photo taken during Day 2.

Virtual participation varied across the course of the workshop with an average of 10-15.

## 2.3 Agenda

The WP1 team designed the workshop agenda to be flexible and open for discussions, to best achieve the goals of the workshop. It used both plenary presentations/discussion as well as breakout sessions to cover more specialised topics. During the course of the workshop the agenda was adjusted according to the needs of the ongoing discussions to reflect the actual pace and to accommodate additional topics raised. Tables 1 and 2 depict the final agenda, as actually used during the workshop.


**FIGURE 1:** GROUP PHOTO OF THE DAY 2 PARTICIPANTS.

| Time | Session |
|---|---|
| 08:30–09:00 | Arrive at Meeting venue |
| 09:00–09:15 | Welcome and Organisation |
| 09:15–10:30 | Session 1: ADR Process and GitLab Instructions |
| | – Architecture Decision Records (ADRs) |
| | – Creating and modifying ADRs |
| | – Documenting Use cases and Software Architecture Decisions as ADRs |
| 10:30–11:00 | Break |
| 11:00–11:30 | Session 2: Summary of Use Cases and Categories |
| 11:30–12:30 | Session 3: Breakout Sessions |
| | – Node-Level Use-Cases |
| | – Job-Level Use-Cases |
| | – System-Level Use-Cases |
| 13:00–14:00 | Lunch |
| 14:00–14:30 | Session 4: Report Back and Brainstorming on Missing Use Cases |
| 14:30–16:00 | Session 5: Overview of Software Component List |
| 16:00–16:30 | Break |
| 16:30–17:00 | Session 6: EVIDEN ARGOS System Overview |
| 17:00–18:00 | Session 7a: Continued Discussion on Use Cases, Part 1 |
| 18:00 | Adjourn |

**Table 1:** Agenda for day one (September 18)

| Time | Session |
|---|---|
| 08:15–08:30 | Arrive at Meeting venue |
| 08:30–10:30 | Session 7b: Continued Discussion on Use Cases, Part 2 |
| 10:30–11:00 | Break |
| 11:00–12:30 | Session 8: Breakout Sessions |
| | – Hardware Access Daemon |
| | – Workload Characterisation |
| | – Data Plane Details |
| 12:30–13:30 | Lunch |
| 13:30–14:30 | Report Out from Breakout Sessions |
| 14:30–15:30 | Session 9: Initial Scheduler and Resource Manager Analysis |
| 15:30 | Farewell Coffee Break & Adjourn |

**Table 2:** Agenda for day two (September 19)

# 3 Workshop Sessions and Discussions

The workshop had the goal to document use cases, map parts of the use cases to components, and with that refine the initial architecture from the proposal. For this, the organising team attempted to balance the need for flexibility and open discussions with the need for proper documentation and tracking (as expected from a proper software development project). As a consequence, the workshop started with a presentation of the ADR process to be used for documentation and tracking, followed by brainstorming sessions and breakout discussions. The workshop was concluded with a discussion on one of the key components, the resource manager, as well as a feedback round, fine tuning the ADR process, as well as concrete next steps.

## 3.1 Session 1 (Day 1): ADR Process and GitLab Instructions

For a complex, collaborative software development project, like SEANERGYS, it is of prime importance to institute strict processes for collecting use cases, defining flow-down requirements and coming up with architectural and software design decisions. This not only requires a genuine software development processes, which SEANERGYS will follow (and which will documented in more detail in D5.2 and D8.1), but also traceable design decision and process documentation. For both, SEANERGYS uses GitLab as a central tool.

To introduce the team to this process, WP5 presented the development process, which the project will use both for managing documentation, such as use-cases and ADRs, and for developing the SEANERGYS code base. This process leverages the structured and traceable workflow within GitLab. The presentation outlined how GitLab's integrated tools—such as labels, templates, and permalinks—are used to capture use-cases and document key architectural decisions consistently, and how in the next step also code in SEANERGYS will be managed. The approach aims to achieve several core goals: ensuring traceability by linking features, discussions, and progress through issues and merge requests; fostering collaboration among distributed teams via mentions, assignees, and threaded discussions; maintaining structure by organising dependencies using Epics and linked issues; supporting asynchronous work through notifications, ToDos, and due dates; and enabling automatic publishing of shared documentation via GitLab Pages.

### GitLab Workflow

The GitLab branch and merge request workflow provides a structured approach to collaborative software development, ensuring that all code or document additions, deletions, or changes are reviewed, tested, and approved before being integrated into the main codebase or approved documentation. Figure 2 illustrates the GitLab branch and merge request workflow to be adopted for SEANERGYS. Development work begins in a separate feature branch (e.g. dev-feature-abc), where multiple commits are made, as codes are developed and refined. Once the feature is ready, a merge request is created to integrate it into the protected main branch. The merge request undergoes a review process where reviewers can provide comments, request revisions, and ultimately approve the changes. Continuous Integration (CI) pipelines automatically run to ensure the new code or text passes all tests and quality/consistency checks. Once the merge request

is approved and the CI pipeline succeeds, the changes are merged into the main branch, maintaining a controlled, reviewed, and stable codebase or approved documentation.

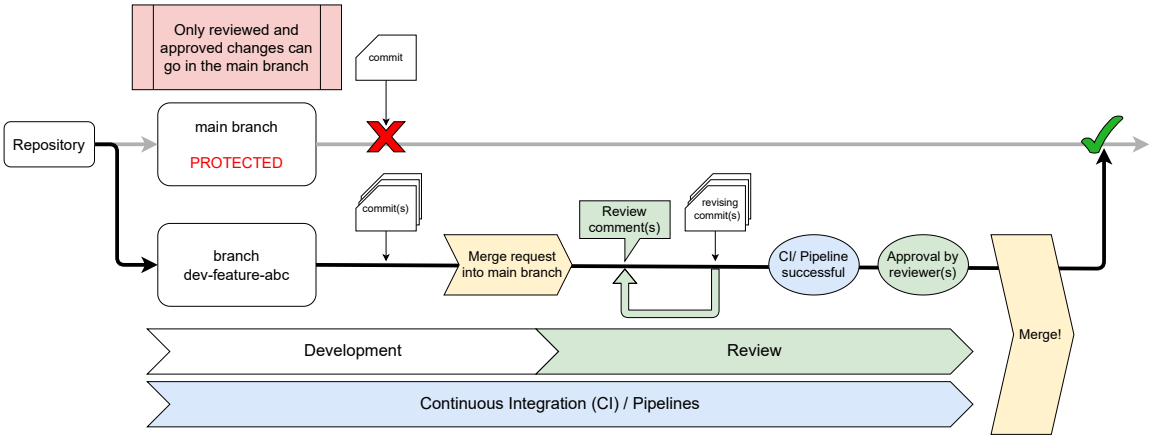


**FIGURE 2:** GIT BRANCH AND MERGE REQUEST WORKFLOW ILLUSTRATING THE PROCESS OF FEATURE DEVELOPMENT, DOCUMENTATION UPDATES, CODE REVIEW, AND CONTROLLED INTEGRATION INTO THE MAIN BRANCH

## Use Case Workflow

Figure 3 illustrates the workflow for managing and automating use-cases within the development process. It begins with a merge request containing an annotated markdown file that defines a use case. A CI job then validates the markdown structure, checking for required fields, scheme compliance, and correct links. After successful validation, the merge request undergoes approval and code review to ensure content quality and accuracy. Once approved, an automated process converts the markdown file into structured data, creates corresponding GitLab epics, applies appropriate tags, and performs bookkeeping tasks to maintain consistency across the project. Finally, the accepted use cases are published as documentation pages, ensuring traceable and well-documented project artifacts.

## ADR Workflow

Following the description of use-case workflow, the WP5 presentation introduced the framework for ADRs that will be used to document key project decisions in a consistent and transparent manner. Each ADR captures important details including the title, decision outcome, context and problem statement, discussion, and the consequences—both positive and negative—of the decision. Once finalised, an ADR becomes immutable to preserve its integrity, but it can be superseded by a new ADR when architectural changes are necessary; in such cases, the new ADR must explicitly reference the previous one to maintain traceability. The creation and management process mirrors that of GitLab use-cases: ADRs are created and tracked in GitLab, linked to related issues and merge requests, and collaboratively reviewed and approved by contributors. This structured and traceable approach ensures that architectural decisions are well-documented, justified, and accessible to all team members, supporting long-term project coherence.

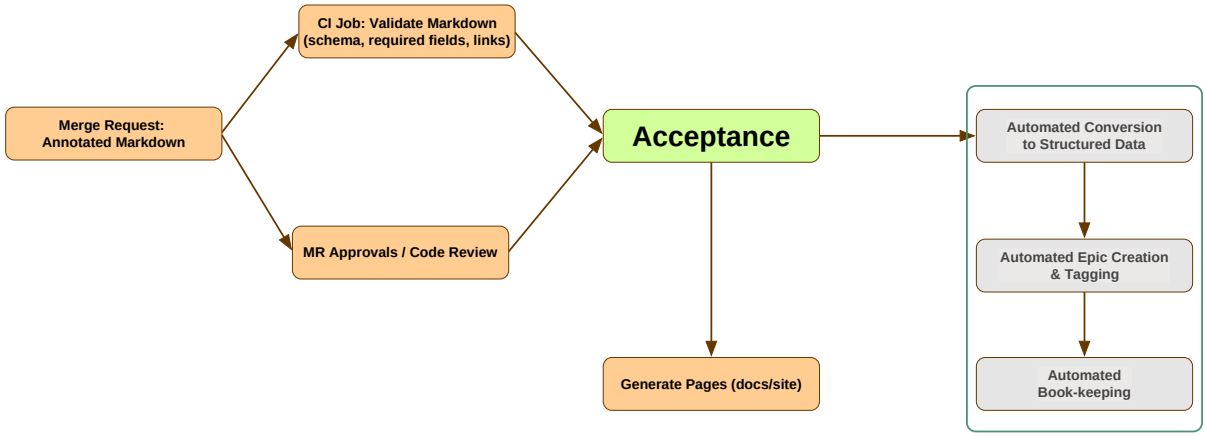


**FIGURE 3:** AUTOMATED USE CASE WORKFLOW SHOWING THE PROCESS FROM ANNOTATED MARKDOWN CREATION AND VALIDATION TO AUTOMATED EPIC GENERATION, BOOKKEEPING, AND DOCUMENTATION PUBLISHING

After evaluating multiple tools for managing ADRs, we decided to adopt Log4brains as the official solution. It provides a well-structured ADR template, integrates seamlessly with Git, and offers both a Command-Line Interface (CLI) and a static site generator that enable easy local previews and clear web-based navigation. This choice aligns with our need for a developer-friendly, lightweight, and accessible system to document and share architectural decisions. While it requires maintaining ADR status in two places and occasionally experiences markdown parsing issues, its overall usability and visualisation capabilities make it the most suitable option for our project's needs.

Once an ADR is defined, we then also use the above GitLab process for ADRs. In the case of ADRs leading directly to code changes, we will further use the GitLab processes to tie the changes triggered by an ADR decision with the ADR itself. This ADR-driven DevOps process is illustrated in Figure 4. The ADR captures one key architectural decisions, and is linked to the original reason for their development, typically overarching use cases or "sister epics." In the ADR, the scope of the ADR in terms of WPs and components is captured and the needed work is broken down into actionable work items for the developers in each WP. The work items are then introduced as GitLab issues associated with source code, leading to concrete implementation plans. Developers create commits and merge requests to contribute changes, which are automatically processed through CI and Testing pipelines. This structure ensures traceability from high-level architecture and design decisions down to key code changes.

## 3.2 Session 2 (Day 1): Summary of Use Cases and Categories

The core agenda point for the workshop was the discussion of use cases and their impact on the architecture. This was done starting with a plenary session upfront, followed by breakouts, of which results were then reported back to the group. This was followed by a general brainstorming on gaps as well as two specific use cases.

The first overview session covered a number of topics:

- Methods used to collect the initial use cases and annotate them

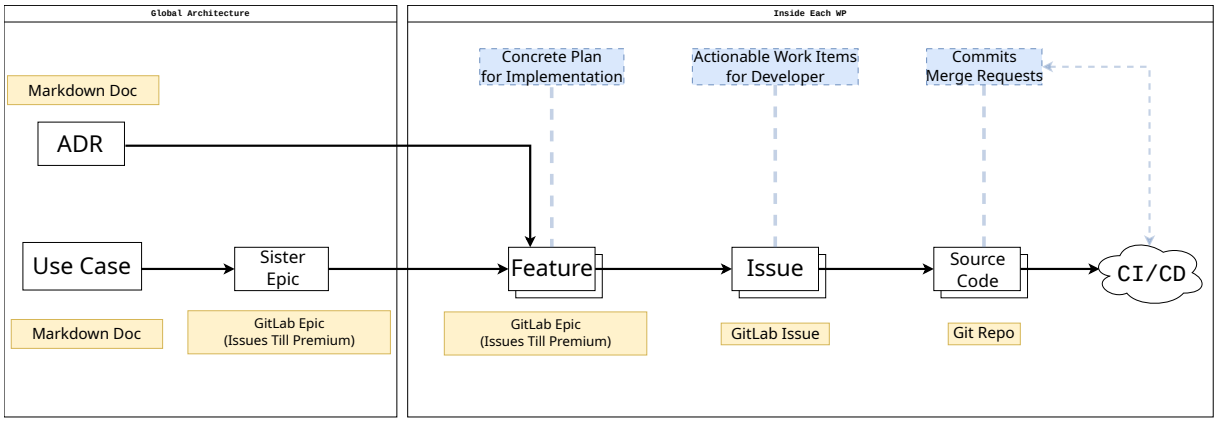


**FIGURE 4:** ADR-BASED DEVOPS WORKFLOW, SHOWING THE DEFINITION AND USE OF ARCHITECTURAL DECISION RECORDS (ADRS). CONNECTED ISSUES IMPLEMENT THE CHANGES AND GITLAB-BASED CI (AND EVENTUALLY CI/CD) PIPELINES PROVIDE END-TO-END ADR DEVELOPMENT AND TRACEABILITY

- Overview of the 39 use cases collected
- Approaches how to group and consolidate the use cases

As a result, a new list of 31 consolidated use cases with proper annotations regarding use case objective, function, level, user, and lead WP was created and entered into GitLab. Section 4.1 provides a summary and a comprehensive list of use cases in Table 4. Following a discussion leading up to the workshop, the group decided to classify use cases into the system level in which they serve and are implemented: on the node, on jobs, or across the entire system. The following breakouts were then structured along these groups with detailed discussions.

## 3.3 Session 3 (Day 1): Breakout Sessions

The next step in the workshop were three focused breakout sessions aimed at exploring different facets of energy optimisation in High Performance Computing (HPC) systems. The sessions were structured to address complementary levels of analysis and strategy development

- **System-level**: focused on energy reduction and optimisation at the level of a complete HPC and Artificial Intelligence (AI) system;
- **Node-level**: focused on data from node components and their potential for energy optimisation; and
- **Job-level**: investigated how historical job submission data could inform energy-efficient scheduling and system management.

Together, these sessions provided a comprehensive framework for understanding and improving energy efficiency across multiple layers of HPC operation.

### 3.3.1 Use Cases at System Level

Clearly, maximising throughput and minimising energy used by a system are competing objectives, i.e. throughput can be maximised by operating all system resources at their highest performance, resulting in maximum energy use. Vice versa, minimising energy use can be easiest achieved by deep throttling of resources and maybe even by switching off parts of a system, which in turn will limit or reduce throughput. Therefore, it is important to find a balance, determined by weights attached to metrics, like throughput and energy. These can differ from centre to centre, following the local policy and optimisation goals.

To ensure safe and affordable operation, it is important to run the complete system within a total power budget and within the centre's cooling capacity, as well as by availability and prices of energy and the available budget; all three limits can vary over time (with the environmental conditions and power caps and prices set by the power provider). To achieve a global system power cap, it has to be broken down to its constituent parts of the system, like modules or racks, forming a system hierarchy. Global limits then need to be propagated down such an established hierarchy to the nodes, switches and storage devices. While it would be easiest to subdivide the global budget and set individual power budgets in the same way for all like resources, a case was made to recognise the characteristics of workloads running and minimise the global effect of the power cap on these. An example is to reduce frequency more aggressively on nodes running memory-bound workloads, since the delivered performance would be impacted less than that for nodes with Central Processing Unit (CPU)-bound workloads.

In the interest of limiting any effect on the workloads, the scheduler would also need to take the global power cap into consideration for creating a compatible schedule. For example, when back filling nodes, it would need information on the predicted energy use and characteristic (memory vs. CPU bound) of jobs.

In cases of extreme reduction of global power budgets, it would become necessary to (temporarily) switch off parts of the system to meet the new limits. This will necessitate terminating running workloads, and the selection of which nodes and workloads to target that way could be aided by node-level data (energy consumption of single nodes) and job-level data and predictions – an obvious example being the time a job has been running (terminating jobs early on will likely lose less "state") or, better still, the use of checkpoints by a job and the time since the last checkpoint. Speculatively, one could also consider the option of consolidating workloads on fewer nodes, provided that "moving" running jobs becomes technically feasible, and that combining jobs on nodes is possible (co-scheduling).

Finally, the general topic of leveraging node sharing (i.e. running multiple independent jobs on the same node/nodes) was raised. Low-hanging fruit here would include small jobs that use only a part of a single node, which could be combined to run on the same node with minimal individual performance impacts and a lower total energy use; combining multi-node applications that do only use part of the capacity/capability of each of the nodes they are running on, might also be possible. In any case, having reliable data or predictions about the suitability of jobs to be run using node sharing is a key requirement.

### 3.3.2 Use Cases at Node Level

The discussion in the Node-Level Architectures breakout session focused on strategies for optimising system throughput under a global power cap by coordinating control ("knobs") of in-node components such as CPUs and Graphics Processing Units (GPUs). The participants explored mechanisms for dynamic power

allocation, intra-node coordination, and integration with higher-level schedulers to achieve energy-efficient yet performance-conscious operation in HPC environments.

The discussion was structured around several core aspects of node-level power management, emphasising both the architectural and operational perspectives. Participants examined how node managers can enforce power limits, handle real-time data, and interact with higher-level control mechanisms. Subsequent sections summarise these discussions, detailing node power management principles, fine-grained power control approaches, and the concept of predefined power profiles designed to balance performance with energy efficiency.

### Node Power Management

A central theme of the discussion was the role of an effective Node Manager in enforcing and coordinating power management at the node level. Key points included:

- **Node-level power limits:** Power constraints should be enforced through a Node Manager capable of managing intra-node power distribution across hardware components such as CPUs and GPUs, maintaining power balance, and integrating monitoring capabilities within its management scope.
- **Live data requirements:** Data specifications must be clearly defined, including parameters such as reaction time, granularity, lifetime, encoding, and database structure. The group emphasised the need to clarify whether a bi-directional data channel or a separate control plane will be necessary for efficient operation.
- **Configurability and dynamic adjustment:** The Node Manager should support flexible configuration and dynamic adjustment of node parameters, enabling adaptive responses to workload and power changes.

### Fine-grained power control:

Effective energy optimisation requires fine-grained power management that allows job-level power caps to aggregate into node-level power caps. The Node Manager's power cap Application Programming Interface (API) should expose resource-level details that can be guided by the system or job scheduler.

### Predefined power profiles:

The concept of predefined power/frequency configurations ("tags") was proposed to streamline management. In this model, the resource manager specifies power profiles per user or job, while the Node Manager provides an API to apply these configurations but does not define them directly.

The group concluded that a robust and flexible Node Manager is central to balancing performance and energy efficiency within power-constrained HPC systems. Such a manager must combine real-time data handling, configurable control mechanisms, and effective communication with higher-level schedulers to achieve coordinated, system-wide optimisation.

### 3.3.3 Job Level Use-Cases

The discussion in the Job-Level Architectures breakout session focused on exploring Machine Learning (ML)-based strategies for predicting and optimising job-level performance and energy consumption within HPC systems. Two primary use cases were discussed in detail: job runtime prediction and job power consumption prediction. Both cases aim to leverage cross-WP collaboration and multi-source data integration to support dynamic, energy-aware scheduling and user feedback mechanisms.

The discussion was organised around how predictive models can evolve from static estimates at schedule time to dynamic updates during queueing and job execution. The models draw on performance data from WP2, job metadata from WP4, scheduler and launcher input supported by WP4, and job script interpretation with support from WP3. The outcomes of these predictive processes feed directly into both the scheduler being developed by WP4 and the visualisation and user-support tools under WP3.

#### Use Case: Job Runtime Prediction

The first use case discussed in the breakout focused on predicting the execution time (runtime) of a job using ML-based approaches. The consensus was to establish a multi-phase prediction framework that operates:

- **At schedule time (static prediction):** Estimating runtime prior to job submission to guide scheduling decisions and resource allocation.
- **While in queue:** Updating predictions dynamically as more contextual and system state information becomes available.
- **During job execution:** Refining predictions in real time based on live performance data to identify deviations or anomalies.

These predictions utilise performance data provided by WP2, job metadata from WP4, scheduler and launcher input (with WP4 support), and analysis of launching scripts (with WP3 support). The system also accounts for the effects of co-running jobs on performance. The predictive outputs are fed into the scheduler under development in WP4 to improve resource utilisation and into the visualisation and user-support performance reports developed in WP3 to enhance user feedback and transparency.

#### Use Case: Job Power Consumption Prediction

The second use case focused on predicting job-level power consumption using ML models capable of estimating both the time series of power usage and the expected power peak during execution. Similar to the runtime prediction use case, the framework supports:

- **At schedule time (static prediction):** Providing early estimates of expected power demands for scheduling optimisation under system-wide power caps.
- **While in queue:** Updating power consumption predictions as system load and job interactions evolve.

- **During job execution:** Continuously refining power usage forecasts using real-time monitoring data.

These predictions also integrate performance data from WP2, job metadata from WP4, scheduler and launcher inputs supported by WP4, and launching script analysis from WP3. The models consider the potential impact of co-running jobs on total power consumption. The resulting power consumption predictions are fed into both the WP4 scheduler for energy-aware decision making and the WP3 visualisation and reporting tools to inform users about expected and actual energy usage patterns.

The session emphasised that job-level predictive modelling for both runtime and power consumption is central to achieving energy-efficient HPC operation. The integration of performance data, metadata, and real-time monitoring enables dynamic adaptation across the job life cycle. Collaboration between WP2, WP3, and WP4 is essential to ensure data availability, model validation, and alignment between predictive insights, scheduling strategies, and user-facing performance reporting.

## 3.4 Session 4 (Day 1): Report Back and Brainstorming on Missing Use Cases

The next session of the workshop focused on brainstorming additional use cases, with the objective of identifying new scenarios that capture system- and node-level architectural impacts while incorporating end-user perspectives on energy use. Participants aimed to explore extensions to the existing SEANERGYS use cases, strengthening the links between architecture, energy management, and user experience.

### 3.4.1 Discussion

During the brainstorming on missing use cases, the team emphasised the importance of understanding the impact on the overall system architecture. Among all use cases, optimising for system throughput emerged as a potential use case (driven based on needs in the telemetry work in WP2), alongside others that ranged from basic to complex requirements. One area of particular interest was the creation of a power profile or footprint for any application (which was driven out of the analysis capabilities in WP3).

In addition to system-level considerations, node-level discussions were also highlighted. These focused on how individual components and jobs interact with energy constraints and performance expectations. A key point raised was the expectation that end users should be aware of their energy consumption, not only to promote transparency but also to enable $CO_2$ footprint estimation. The SEANERGYS framework could play a pivotal role in educating users about actionable measures to reduce their energy usage.

The conversation also explored data centre-level strategies, such as power capping and frequency scaling. Questions arose around the decision-making authority within data centres—specifically, who determines the strategies for throttling jobs based on predicted energy consumption. The possibility of centres power-capping jobs was discussed, with the caveat that such measures must be balanced against user experience. It was noted that clocking down jobs indiscriminately could lead to unacceptable delays for end users, raising the need for indirect representation of user interests in centre-level policies.

Ultimately, the group agreed that future use cases should address both technical feasibility and user impact, ensuring that energy-aware computing does not compromise usability or performance. These insights will guide the refinement of WP1 and inform the next steps in architectural planning.

### 3.4.2 Insights

In summary, the discussion revolved around three major themes: system-level insights, node-level considerations, and end-user perspectives and trade-offs. Each topic contributed to a broader understanding of how future use cases could more comprehensively represent performance, efficiency, and usability dimensions.

#### System-Level Insights

At the system level, participants noted that several existing use cases, especially those covering telemetry and data plane architecture (which typically emerged from discussions in WP2), already outline potential architectural drivers. However, they proposed the inclusion of an additional use case focusing on system throughput optimisation under power constraints to better capture architectural dependencies. The group expressed interest in analysing and integrating power profiles to inform architectural design decisions and to enable better coordination between hardware capabilities and power management strategies.

It was emphasised that both basic and complex system requirements should be represented in the expanded use-case set. The intent is to ensure that architectural insights not only address current operational limits but also support scalability, adaptability, and predictive system management under varying power conditions.

#### Node-Level Considerations

At the node level, the discussion centred on improving end-user awareness of energy consumption and enabling actionable insights. Participants agreed that users should be able to understand their own carbon footprint and receive guidance on minimising energy usage. To support this, the project could develop feedback mechanisms or interactive tools that provide transparent, personalised energy usage data and recommendations.

Additionally, the group discussed exploring data centre-level power management strategies, such as power-capping individual jobs, dynamic throttling based on predicted energy use, and adjusting power limits and CPU/GPU frequencies dynamically in response to workload and system states. These strategies could enhance node-level adaptability and complement higher-level scheduling mechanisms to improve overall energy efficiency.

#### End-User Perspective and Trade-offs

The discussion also emphasised the importance of reflecting the end-user perspective in the design of energy-aware HPC systems. Participants highlighted that data centre policies must balance system-level optimisation with acceptable user performance outcomes. An identified key trade-off was that overly aggressive power capping—while energy efficient—could lead to performance degradation and unacceptable job delays.

The group agreed that user experience metrics should therefore be integrated into energy optimisation frameworks to ensure that system policies remain both efficient and user-centred. Transparent communication of trade-offs through performance reporting and user-support tools was seen as a critical factor in achieving this balance.

### 3.4.3 Brainstorming Conclusions

In conclusion, the brainstorming session underscored the need for new use cases that explicitly link system performance, power management, and user impact. By combining architectural insights with user-focused considerations, SEANERGYS can develop a more holistic set of scenarios that reflect real-world operational priorities. These enhanced use cases will help ensure that future architectural designs support both energy efficiency and usability across all layers of HPC system management.

## 3.5 Session 5 (Day 1): Overview of Software Component List

Following the detailed use case discussion, the group made a change of topics and looked at the components that would be needed to implement the use cases and the resulting architecture. This session highlighted the need for a clearer understanding of the component landscape within SEANERGYS, especially in relation to the defined use cases. One immediate action item was to ensure that the needed authentication levels to run the components (system vs. user) are properly documented and included in the component overview. To support architectural clarity, participants suggested creating a graphical representation of components, which would help visualise their interconnections and dependencies. This will be included in the upcoming Deliverable D1.4, which defines the overall architecture.

### 3.5.1 Mapping and Dependencies

The discussion began with the topic of mapping components to the architecture as well as their dependencies among each other. The group agreed on a systematic process:

1. Define all use cases comprehensively
2. Map existing components against those use cases
3. Identify any missing components required to support them

This mapping, which will be defined in Deliverable D1.4, will provide the base for assessing the completeness of the architecture and identifying integration gaps across the system. Participants also emphasised that system optimisation will rely heavily on performance and throughput considerations, ensuring that each component contributes effectively to overall efficiency.

### 3.5.2 Architecture and Flexibility

The second discussion focused on architectural design principles and flexibility. The group agreed that the virtualisation and transformation layers can play a pivotal role in unifying heterogeneous data sources under shared semantics. This unification would simplify interoperability across the project's components. A key consensus point was that node-local control actions – such as CPU or GPU frequency adjustments – should operate independently of long communication loops to minimise latency and overhead. The session also highlighted the need for a single, powerful API capable of handling both monitoring and control operations, even if implemented with minor variations to accommodate specific use cases or system constraints.

### 3.5.3 Data Plane and API Discussion

The final topic addressed data handling, focusing on the data plane and API architecture. The architecture is illustrated in Figure 5.

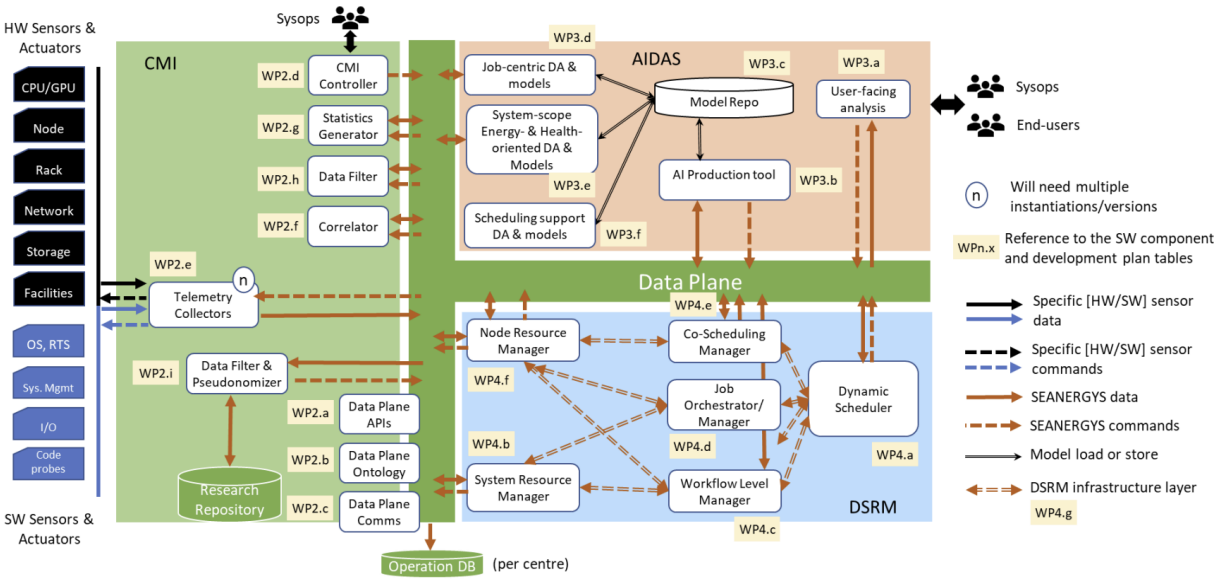


**FIGURE 5:** TOP-LEVEL SOFTWARE ARCHITECTURE

Concerns were raised about the complexity introduced by having multiple data planes. A single data plane was favoured for ensuring compatibility across tools and simplifying integration. The idea of combining a single data plane with a control plane was proposed as a way to manage configuration and activation models more effectively. It was pointed out that much of the current discussion revolves around streaming data, which raises the question of whether activation and configuration models should also be integrated into the same data plane.

One point emphasised was the need for access to *all relevant* data, reinforcing the importance of a unified approach. However, the group acknowledged that security considerations might necessitate separate pipelines, which could simplify certain aspects of implementation. This led to a deeper exploration of what constitutes different data planes—whether it is a matter of communication mechanisms, representation formats, or semantic boundaries.

The consensus leaned toward designing a powerful and semantically rich API that could serve diverse needs. If the API is sufficiently expressive, it would unify different data sources and representations under a common interface. For example, a power counter on a node and a Slurm job database might share the same API if their semantics align, but diverge if their operational requirements differ.

The group also discussed the importance of low-latency data access, particularly for node-local operations requiring fast turnaround like thread scheduling or small-granular frequency scaling. It was agreed that such operations should not require long round trips, and that a generic, flexible API could support both high-performance and configurability. To the question of whether a single API could realistically serve all needs, the group responded optimistically, provided that data annotation and format flexibility are built into the architecture.

In summary, the session concluded with strong agreement on the importance of a unified, semantically consistent API built on a single data plane and, if needed, a complementary control plane. Such a design would simplify component integration, preserve performance, and enable flexible data representation across all project modules. This unified approach was seen as essential to achieving interoperability, scalability, and maintainability within the SEANERGYS software architecture.

Ultimately, the analogy to the Message-Passing Interface (MPI) was invoked: a single API can support a wide range of operations, if designed well. While multiple APIs are possible, the goal should be to converge on a unified interface that supports diverse components and use cases without sacrificing performance or semantic clarity.

## 3.6 Session 6 (Day 1): EVIDEN ARGOS System Overview

In this session, Philippe Couvée from Eviden delivered a presentation highlighting the key features of the BullSequana ARGOS V2.0 architecture. The architecture incorporates two types of data collectors: job-related collectors, which gather data during job execution and index it using the job ID, cluster name, and user ID; and infrastructure-related collectors, which continuously collect system-wide data. These collectors are responsible for transferring data from the source to the consolidation layer and for managing storage within their respective databases. Data access and presentation are handled through standardised REST APIs, integrated with an APISIX-based API gateway that manages user authentication and serves as the main access point for end users.

Collectors expose a simple, standardised API with three main endpoints:

- **Metadata Endpoint:** An optional endpoint that provides job-related metadata based on parameters such as cluster name, job ID, or user ID. It returns information including node lists, job duration, return codes, runtime environment, and the list of active collectors with their attributes.

- **Summary Endpoint:** Provides consolidated metrics for each collector over the duration of a job or infrastructure activity. For job collectors, parameters include cluster name, job ID, and user ID; for infrastructure collectors, the parameters are start time, end time, and device list. Typical outputs include total MPI call counts and average CPU usage.
- **Time-Series Endpoint:** Returns time-series metrics on a per-collector basis using the same parameterisation as the summary endpoint. It provides detailed, temporal performance data such as CPU or GPU usage across all nodes, collected at regular intervals (e.g. every five seconds).

## 3.7 Session 7 (Day 1/2): Continued Discussion on Use Cases

After the exploration of the component landscape and the immediate ties of components to use cases, the group decided to go deeper into more cases to study their impact on the architecture and its components. We ended up discussing two larger key use cases (power capping and power-aware scheduling) and decided to move more use cases to the upcoming regular WP1 meetings.

### 3.7.1 Use Case Details: Power Capping (provided by BAdW-LRZ)

#### Power capping as a constraint for optimisation of energy

This use case focuses on implementing a system-wide power capping mechanism to reduce energy consumption in HPC environments, motivated by the increasing cost of energy. The system should enforce a global power cap, which is then communicated to individual nodes for localised management.

A key constraint is that application performance must not degrade by more than 10 %, ensuring that energy savings do not come at the expense of usability or scientific productivity. The system manager is responsible for distributing power budgets dynamically across the system, while node managers handle the enforcement of local caps. In cases of conflicting constraints, the stricter (larger) constraint takes precedence.

To support this functionality, the system requires a live feed of sensor data integrated into the data plane. This enables real-time monitoring and decision-making based on current power usage and system state. Identifying which sensors to read and how to stream their data efficiently is a critical part of the design.

This use case highlights the need for coordination between system-level and node-level components, and for a robust infrastructure that can balance energy efficiency with performance.

On the architecture side, this use case is well supported, assuming a flexible data plane that allows measurements from the nodes as well as the ability to control frequencies. It also requires a system wide daemon infrastructure, which is already part of the architecture.

### 3.7.2 Use Case Details: Scheduling (provided by LXP)

#### Power-Aware Scheduling

This use case focuses on node-level scheduling strategies that adapt to energy efficiency goals while maintaining fairness across jobs. The aim is to implement power-aware scheduling on homogeneous nodes by classifying them into “red” and “green” zones:

- Red nodes are uncapped and suitable for power-hungry jobs.
- Green nodes are power-capped and designated for energy-efficient jobs that do not require full power.

The classification is managed by system administrators and updated periodically. During job submission, the scheduler evaluates job characteristics—such as requested cores, GPUs, memory, and potentially job scripts—and queries a Characterisation Component (WP3) to generate a tag list describing the job’s energy profile. This tag list is then used by a Red/Green Decision Component to determine the appropriate node type for scheduling.

The entire decision process occurs at submission time, ensuring that scheduling policies are applied early and efficiently. The system relies on a data plane for communication between components, using remote invocation or service calls. Historical and labelled data are essential for training the models that drive both characterisation and decision-making. Further, it relies on proper model building. While the former is already cleanly integrated into the architecture, the latter will require a refinement of data plane usage within WPs (which later led to the data plane discussion, see below) as well as decisions on location on model and raw-data storage.

The overarching goals are:

- Power capping through intelligent node selection.
- Energy savings without compromising fairness or performance.
- Seamless integration of scheduling logic with existing infrastructure via a unified API.

This use case supports the broader SEANERGYS vision of energy-aware HPC operations, leveraging predictive models and system-level coordination to optimise resource usage.

## 3.8 Session 8 (Day 2): Breakout Sessions

In a second set of breakout session, the groups focused on key components from the two use cases and discussed them in more detail.

### 3.8.1 Breakout 1: Hardware Access Daemon

In the discussion on hardware access daemons for energy efficiency, it was proposed that each compute node should run a dedicated daemon responsible for interfacing with both hardware sensors and actuators. These daemons would operate with privileged access, enabling fine-grained control over energy-related parameters. A key function of the daemon is conflict resolution among competing requests, which necessitates a robust system of prioritisation, authorisation mechanisms, and likely the use of session tokens to manage access securely. To function effectively, the daemon must be context-aware, integrating both software and hardware states to make informed decisions.

The conversation also touched on the broader integration of these daemons within the project's architecture, particularly in relation to WP5, which anticipates their deployment. A point of contention was whether the daemons should rely solely on hardware counters or also incorporate software-level events such as MPI activity. It was agreed that such contextual information should ideally be provided by other services, suggesting a need for further coordination across WPs. This highlights the importance of cross-WP collaboration to ensure the daemons are both effective and interoperable within the larger system.

### 3.8.2 Breakout 2: Workload Characterisation

The discussion in the Workload Characterisation breakout session focused on defining the data needs, sources, and mechanisms for effective workload characterisation and model training across the SEANERGYS components. Participants examined how workload data can be collected, processed, and leveraged to enhance model accuracy and enable energy-efficient operation in HPC environments.

The discussion was organised around five key themes: data requirements and sources, model training and evolution, advanced use cases and extensions, architecture and retraining mechanisms, and governance and role clarity. Each of these topics is summarised in the sections below, capturing the main insights and directions identified by the participants.

#### Data Requirements and Sources

The group identified several categories of data essential for workload characterisation:

- **Data types:** Submission-time data (such as user resource requests, job scripts, input data, and loaded libraries); historical data (past job records retrievable through searches or embedded in trained models); and dynamic data (live monitoring information from running jobs).
- **Data sources:** Existing repositories within WP2 and WP3, as well as live site data provided by hosting institutions.
- **Desired outcomes:** Improved characterisation of runtime behaviour, power consumption, and overall energy efficiency.

#### Model Training and Evolution

The discussion emphasised the need for well-defined processes for model training and ongoing refinement. Participants agreed that models should continue self-training beyond the end of the project, with a retraining pipeline established as part of the project infrastructure. Post-mortem data from completed runs will be used to refine submission-time models. The potential for cross-site data sharing was recognised as a means to enhance training, although it may require data anonymisation. Retraining strategies must also consider *ab initio* training approaches when frameworks or architectures undergo significant changes. However, retraining strategies must also take into account the compute and energy costs for such retraining, which will likely be non-negligible, and find a suitable trade-off.

#### Advanced Use Cases and Extensions

Several advanced applications were proposed to extend the impact of workload characterisation. Early dynamic job data could be used to refine predictions and trigger proactive actions such as job consolidation or restarts. Predictive updates (for example, on probable runtime) could support scheduling decisions in real time. Additionally, the group discussed the potential of multi-job or global models to enable coordinated co-scheduling and "digital twin" simulations of system behaviour, while still maintaining the utility of per-job models.

#### Architecture and Retraining Mechanisms

Participants highlighted the need for a dedicated architectural component to oversee model training and retraining activities, particularly through continuous comparison between predictions and actual outcomes. Retraining workflows will require Continuous Integration and Continuous Delivery/Deployment (CI/CD) pipelines capable of handling large data volumes and performing hyperparameter tuning efficiently. Distinct requirements for retraining versus *ab initio* training must also be considered in system design.

#### Governance and Role Clarity

The session concluded with a discussion on governance and the delineation of responsibilities. Close collaboration between WP3 and WP4 will be necessary to define data ownership, access, and functional boundaries. It was clarified that the Artificial Intelligence Data Analytics System (AIDAS) developed in WP3 will concentrate on data submission and analysis activities, without direct involvement in issuing control commands to HPC systems.

### 3.8.3 Breakout 3: Data Plane Details

In this breakout it was decided to propose the development and implementation of a single API to access the Data Plane, encapsulated within a dedicated library. This API will abstract all communication details, allowing component developers to interact with the system without having to manage low-level implementation specifics. It should support both Publish/Subscribe (Pub/Sub) and request/response semantics, ensuring flexibility across different communication patterns.

This approach simplifies development by hiding the complexity of the underlying communication mechanisms, enabling faster integration and reducing the burden on component teams. However, it may also introduce additional overhead due to the extra software layer, which could impact performance if not carefully optimised.

To address performance concerns, the Data Plane will include node-local shortcuts and provide bindings for the relevant programming languages (e.g. C, C++, Python). While request/response could be built on top of Pub/Sub, the team acknowledged potential drawbacks in terms of complexity and efficiency. Therefore, the API may internally leverage different communication frameworks for each semantic type, while maintaining a unified interface. The need for data annotation and consistent semantics across all data streams was also emphasised to ensure interoperability and clarity.

## 3.9 Session 9 (Day 2): Initial Scheduler and Resource Manager Analysis

Two central components in the architecture are the (batch) scheduler and resource manager.[1] To achieve the project goals, the scheduler and resource manager in the SEANERGYS software solution must fully support all identified energy optimisation methods, truly dynamic assignment and use of resources, and adaptive workloads and workflows, which likely requires changes and/or extensions compared to currently supplied schedulers and resource managers. As a consequence, the work in SEANERGYS, in particular in WP4 on the Dynamic Scheduling and Resource Management (DSRM) component, will be based on an existing base scheduler and resource management system, which the project will adapt and extend to match use cases and requirements and to fit seamlessly into the SEANERGYS software suite. Re-implementing a scheduler or resource management framework from scratch will not be possible, given the SEANERGYS timetable and effort available.

The discussion was focused around the two main contenders—Slurm and Flux—and also gave some consideration to the OAR software developed by partner Université Grenoble Alpes (UGA). Topics covered were the previous discussions of FZJ with SchedMD (the company controlling Slurm) and the Flux development team, the support for dynamicity in the candidate systems, and requirements of the participating HPC centres.

### 3.9.1 Slurm

Slurm (formerly the Simple Linux Utility for Resource Management) is currently the dominant batch scheduler in HPC with proven stability and performance on a very wide range of systems, and a range of functional additions that are implemented as plugins. All HPC centres participating in SEANERGYS are using Slurm. In addition to the batch scheduling itself, Slurm implements a protocol and daemon network for resource management and control of compute nodes, and—in a limited way—also non-compute resources.

Slurm source code is available under the GNU General Public License (GPL) open source license, yet developed and maintained by SchedMD, a private company, which offers commercial licenses including

[1] Often, these two components are part of a single software package

support to customers and tightly controls the upstream Slurm source code. While the Slurm code can be freely forked and modified, collaborations with SchedMD—such as contributing changes for inclusion in upstream Slurm versions or integrating with their development roadmap—typically require subcontracting SchedMD to implement the features required by the project. Conversations with the company clearly stated that software contributions by the project participants would not be accepted on the Slurm upstream codebase. This model poses challenges for projects operating under public or restricted funding, as direct collaboration with SchedMD might not be feasible.

Any independent fork of Slurm would be the sole responsibility of the forking party, with no support from SchedMD for third-party modifications, and no certainty that the extensions would be back-ported into the SchedMD Slurm codebase. In effect, a fork would need to track the evolution of Slurm, either applying changes to each release implementing the extensions that were the reason for the fork, or porting significant functional and security improvements from new Slurm releases. This in turn leads to indefinite-term obligations to keep a putative Slurm-based SEANERGYS scheduler and resource manager viable, without any influence on the SchedMD releases or codebase.

Use of the Slurm plugin interface for implementing the extensions required by SEANERGYS does not look like a viable proposition due to the restricted functionality of that interface. In the DEEP-SEA European High Performance Computing Joint Undertaking (EuroHPC JU) project, considerable effort was expended to prototype an extension for MPI application malleability using the plugin interface; this required building a "twin" of several core Slurm modules, which replicated the relevant internal status of the Slurm core, since that status could not be inspected via the plugin interface. In effect, this resulted in a quasi-fork of that core functionality. In addition, the plugin interface was modified repeatedly. Thus, going the route of coupling SEANERGYS scheduler/resource manager extensions to Slurm via the plugin interface will not save effort compared to a straightforward fork of the Slurm system.

### 3.9.2 Flux

Flux is a modern, open-source job scheduler developed as an alternative to Slurm, introducing greater flexibility and modularity in HPC resource management. It has been in development since 2012 at Lawrence Livermore National Laboratory (LLNL) and was greatly bolstered by its inclusion in the Exascale Computing Project (ECP). Flux has reached a stage of maturity allowing it to be used as the main scheduler and resource manager for the El Capitan system (at the top of the Top500 list since November 2024). Like Slurm, it includes resource management functionality, and extension of the system is possible by using published APIs and protocols.

Flux relies on a community-driven approach that aligns well with publicly funded research projects and promotes diversity in the HPC ecosystem. While Flux shows strong potential, and is in operation on several large systems at LLNL, further evaluation is needed to confirm its scalability and maturity relative to Slurm.

Flux software is available under open source licenses, primarily the GNU Lesser General Public License (LGPL) for the core framework. LLNL is working to put Flux under the auspices of the High Performance Software Foundation (HPSF), adopting the HPSF governance policies. Regarding commercial-grade support, the Flux team at LLNL is in discussions with other commercial companies.

In discussions with FZJ on cooperation opportunities in the context of SEANERGYS, the Flux development team expressed enthusiasm and openness to active collaboration, including the prospect of including third-party code in upstream releases. Previous collaborative work with the Flux team at LLNL has already resulted in design changes and external code contributions that have been merged upstream and accepted into the Flux main code base.

However, a key challenge remains: performance and maturity. While Flux shows promise, questions persist about whether it can match Slurm's scalability and robustness, given Slurm's long-standing presence and proven track record in large-scale HPC environments.

### 3.9.3 Slurm vs. Flux

Table 3 provides an initial comparison of Slurm and Flux across key technical, strategic, and organisational aspects to support informed decision-making regarding future scheduler adoption and collaboration models.

Further, it should be noted that workshop participants pointed out that working with Flux promises to offer a strategic advantage by promoting diversity in HPC scheduling tools and helping to counterbalance Slurm's dominance. This aligns with broader community interests in maintaining competition and innovation in the field. From the SEANERGYS project point of view, the Flux codebase is far easier to work with than the Slurm code, and partnering with the Flux effort to upstream key SEANERGYS additions/extensions would reduce the effort on SEANERGYS and its consortium to maintain and support the SEANERGYS scheduler and resource manager in the mid-term.

After the Design Workshop, SEANERGYS WP4 will continue the analysis effort and extend/update the comparison, with the result to be reported in SEANERGYS Deliverable D4.1.

| Aspect | Slurm | Flux |
|---|---|---|
| **General Overview** | Most widely used and established scheduler in HPC environments; industry standard. | Newer open-source scheduler developed as a modern alternative to Slurm; in operational use at LLNL, including on El Capitan. |
| **Developer & Maintenance** | Developed and maintained by SchedMD (private company). Source code available as open source. | Developed collaboratively by an open-source community, with active engagement from the core development team. |
| **Collaboration Model** | • Open-source, SchedMD controls upstream code.<br>• SchedMD will only up-stream code developed by them; implementation of extensions proposed by third parties requires payment.<br>• Independent forks are allowed but unsupported by SchedMD.<br>• Paid partnership model not compatible with SEANERGYS funding rules. | • Open-source and fully open to collaboration without licensing or financial restrictions.<br>• Flux team is enthusiastic and supportive of co-development.<br>• Flux to be moved under the HPSF and to adopt an HPSF-compatible governance model. |
| **Independence & Flexibility** | Forking is possible, but the fork would need to be entirely supported by SEANERGYS. | Forking and independent development are fully supported; no licensing barriers. |
| **Strategic Considerations** | Continuing to rely on Slurm reinforces its dominant market position | Supporting Flux promotes diversity and competition in the HPC ecosystem, reducing dependency on a single vendor and encouraging innovation. |
| **Performance & Maturity** | Proven high performance and scalability, with a long operational history. | Sufficiently mature for use by LLNL, yet not proven on other systems; achieving performance parity with Slurm remains a key challenge. |
| **Sustainability & Maintenance** | Forked versions would require maintenance by SEANERGYS, as SchedMD would not support third-party changes; tracking the stream of Slurm releases will be difficult. | Maintenance of any extensions themselves stays with SEANERGYS, yet open collaboration can help share the burden for the complete system. |
| **Overall Assessment** | Mature, stable, but restrictive collaboration model and potential funding incompatibility. | Flexible, collaborative, and strategically beneficial, though with potential performance and maturity risks. |

**TABLE 3:** COMPARATIVE ANALYSIS OF SLURM AND FLUX

# 4 Use Cases and ADRs

This chapter lists the use cases and ADRs defined as a result of the Design Workshop; these were refined further after the workshop and will bedescribed in full detail in the upcoming SEANERGYS Deliverable D1.2.

## 4.1 Use Cases

### Use Case Collection Methodology

The methodology used to collect the use cases followed a chronological process:

1. Research and preliminary discussions were conducted to review previous EuroHPC projects, talk to operational experts from the participating centres, and inspect the relevant literature. Commonalities and complementarities among use cases were identified, as well as lexical homogenisation of needs and scope.
2. A GitLab repository was opened, where partners submitted their contributions following the rule "one use case is one file". Discussions and clarifications were handled transparently through GitLab issues. During the initial collection activity and in (virtual) meetings, four set of tags have been identified to describe each use case (goal tags, function tags, level tags, user tags).
3. Before the Design Workshop, a total of 39 use case files had been collected. Some tags have been identified and use cases may be grouped into tags. At this stage, the “level” and “function” dimensions allowed us to clearly distinguish three categories: node-level, job or workflow-level, and system-level use cases.
4. During the Design Workshop, partners reviewed and challenged the collected material. Several use cases were merged, refined, or added to ensure consistency and technical soundness. Some descriptions were also edited for clarity. The final, consolidated set of use cases is presented in the following sections.

Each use case was annotated with the following attributes:

- **Goal**: the overarching purpose.
- **Functional objective(s)**: the technical function(s) contributing to that goal.
- **Abstraction level**: the system layer at which the use case operates.
- **User role**: the actor interacting directly with the SEANERGYS software
- **Associated WP**: the SEANERGYS work package(s) primarily responsible.

The goals of SEANERGYS use cases are grouped as follows:

- RES – Improving resource utilisation of HPC and AI infrastructures. Although energy reduction is not always explicitly stated, better utilisation generally leads to shorter time-to-solution and reduced total energy consumption (e.g. `MIN_TIME_TO_SOLUTION` in the Energy Aware Runtime (EAR) software).
- ENE – Minimising energy consumption. Focused on reducing energy-to-solution, idle energy, or carbon footprint.
- CON – Staying under operational or power constraints. Ensuring that system-level or facility-level limits are respected while maintaining performance.

The functional objectives supporting these goals include:

1. MON – Monitoring
2. ALR – Alerting
3. PRE – Predicting
4. OPT – Optimising

Each use case also targets a specific system level:

- JOB – Individual job
- WF – Workflow or workload (set of jobs)
- NOD – Node level. The use case may apply to multiple or even all the nodes, yet does so independently node by node.
- SYS – Computer system. The HPC or AI system where SEANERGYS software is deployed.
- DCT – Data centre or infrastructure. The use case require external information from the data centre facilities or the operating environment (e.g. air temperature, cooling, or power distribution)

Finally, user roles were identified as:

- USR – Application user
- DEV – Application developer
- ADM – System administrator
- AUT – Autonomous or internal service logic

This structured approach ensured that contributions from all partners were consistently described, comparable across sites, and suitable for subsequent consolidation (in SEANERGYS Deliverable D1.2) into a unified framework of use cases and architecture-driven requirements.

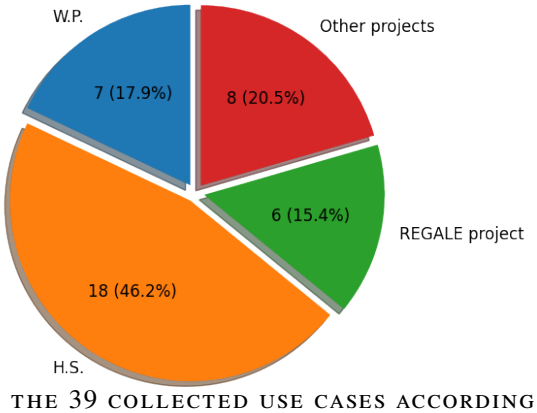


**FIGURE 6:** DISTRIBUTION OF THE 39 COLLECTED USE CASES ACCORDING TO THEIR ORIGIN (W.P. = WORK PACKAGE, H.S. = HPC SITE)

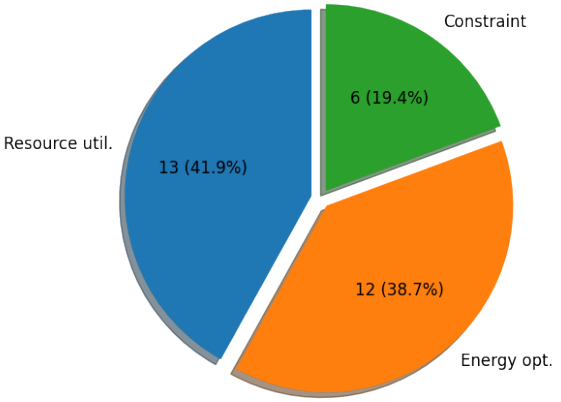


**FIGURE 7:** DISTRIBUTION OF MAIN GOALS AMONG THE 31 UNIQUE USE CASES

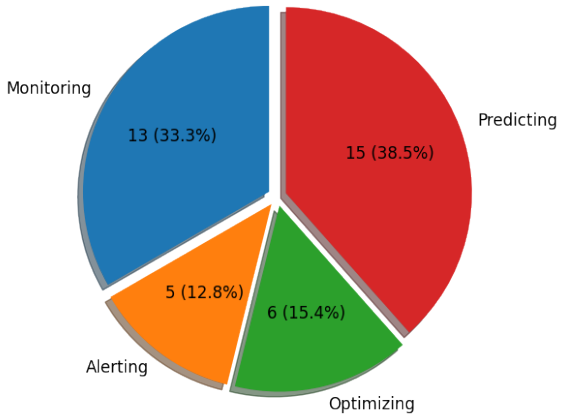


**FIGURE 8:** DISTRIBUTION OF FUNCTIONAL OBJECTIVES ACROSS 31 USE CASES (39 NON-EXCLUSIVE TAGS)

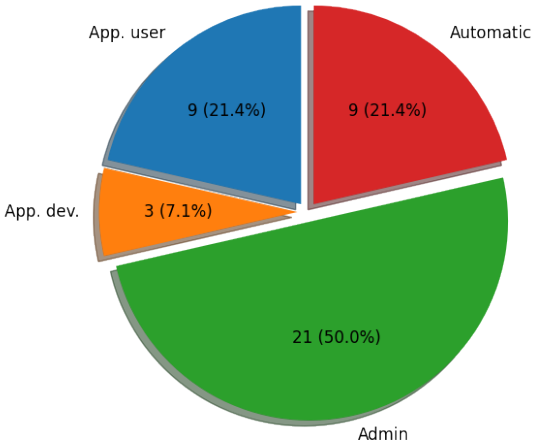


**FIGURE 9:** DISTRIBUTION OF TARGETED USERS AMONG THE 31 UNIQUE USE CASES (42 NON-EXCLUSIVE TAGS)

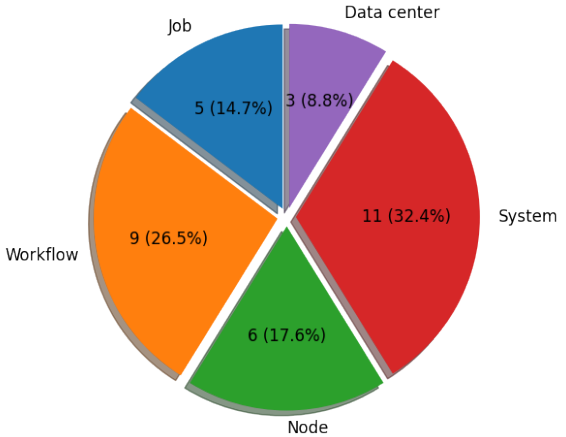


**FIGURE 10:** DISTRIBUTION OF ABSTRACTION LEVELS AMONG THE 31 USE CASES (34 NON-EXCLUSIVE TAGS)

## Overview of Collected Use Cases

The Design Workshop began with 39 use cases gathered from all SEANERGYS partners, as illustrated in Figure 6.

These initial contributions reflect the broad participation of the participating supercomputer centres, research organisations, and technology vendors. While HPC hosting sites contributed most of the initial use cases, research and industrial partners provided complementary perspectives focused on analytics, prediction, and optimisation. Approximately 15 % of the initial cases were adapted from REGALE, the emphasis of which on monitoring and optimising HPC resource utilisation—and thus power efficiency—closely aligns with SEANERGYS. Another 20% comes from other projects and initiatives (e.g., DEEP-SEA, EAR, Data Center Data Base (DCDB), HazardNet, COUNTDOWN), drawing on partners' established expertise and offering a baseline. After the workshop, several use cases were merged, refined, or jointly edited to remove overlaps and clarify their scope. This consolidation resulted in 31 unique use cases. The classification of use cases according to their *goal* and *functional role* is shown in Figures 7 and 8.

The distribution of user roles and system levels is presented in Figures 9 and 10.

Several insights emerge from this analysis. First, the majority of use cases target system administrators as primary users, reflecting SEANERGYS's focus on enhancing the operational efficiency and manageability of production HPC or AI systems. Second, many use cases emphasise autonomous optimisation, aiming to reduce human intervention once mechanisms are properly configured and deployed. Finally, the distribution by abstraction level is weighted toward hardware-oriented cases (node, system, and data centre) at approximately 60 %, with software- and workload-oriented cases (job and workflow) representing the remaining 40 %. This balance underscores the holistic design philosophy of SEANERGYS, addressing both infrastructure management and user-facing optimisation of workload behaviour and performance to achieve integrated efficiency across the HPC stack.

## Collected Use Cases

For completeness, the full list of use cases is provided below. However, we agreed during preparation that discussing such a long table in a single session would not be ideal; therefore, the breakout sessions did focused on three different groups of use cases.

Table 4 summarises the 31 unique use cases – consolidated from 39 initial submissions and refined during the workshop – grouped by goal (RES/ENE/CON) and annotated with function, level, target user, and contributing work package. The column *"uc files"* indicates how many raw submissions (from WPs or hosting sites) were merged into each unique use case.

## Description of Collected Use Cases

Use cases aiming at resource utilisation maximisation:

- **Shared Library Statistics** — Collect statistics on shared library usage via sampling to reveal dependency hotspots and optimisation opportunities.

**TABLE 4:** 31 UNIQUE USE CASES BY GOAL, FUNCTION, LEVEL, USER, AND LEAD WORK PACKAGE

| Use Case name | Goal | uc files | Function | Level | User | WP2 | WP3 | WP4 |
|---|---|---|---|---|---|---|---|---|
| **RES – Improve Resource utilisation** | | | | | | | | |
| Shared Library Statistics | RES | 1 | MON | WF | DEV | X | | |
| User Feedback (job perf/config) | RES | 1 | MON | JOB | USR | X | | |
| Co-Scheduling | RES | 2 | OPT | WF | USR, AUT | | | X |
| Min time under performance threshold | RES | 2 | OPT | WF | USR | | | X |
| Max App Perf under power cap | RES | 2 | OPT | WF | USR | | | X |
| Max throughput under Power cap in 1 node (balance) | RES | 1 | OPT | NOD | ADM, AUT | X | | |
| Max throughput under Power cap Scheduling | RES | 1 | OPT | SYS | ADM, AUT | X | | |
| Monitoring / Dashboard (3D, chatbot) | RES | 2 | MON | DCT | ADM | | X | |
| Alerting anomaly detection hardware failure | RES | 1 | MON, ALR | NOD | ADM | X | | |
| Alerting – node health (zombie nodes) | RES | 1 | MON, ALR | NOD | AUT | X | | |
| Detection user behaviour – underutilisation | RES | 1 | MON, ALR | JOB | USR | X | | |
| Analysis of job submission scripts | RES | 1 | MON | JOB | DEV | | X | |
| Dashboard – user behaviour analysis | RES | 1 | MON | WF | ADM | X | X | |
| **ENE – Minimise Energy** | | | | | | | | |
| Co-optimisation Scheduling | ENE | 1 | PRE, OPT | SYS | USR | | | X |
| Min total system under power cap | ENE | 1 | OPT | SYS | ADM, AUT | X | | |
| Min. Energy-to-Solution | ENE | 3 | OPT | WF | USR | | | X |
| Predefined power config (user/node/island) | ENE | 1 | OPT | NOD | ADM | | | X |
| Dynamic Power Opt. (Idle States) | ENE | 1 | OPT | SYS | AUT | X | | |
| Power Reduction when collective comm. synch. | ENE | 1 | OPT | JOB | AUT | X | | |
| Energy-aware Fair Sharing | ENE | 1 | PRE, OPT | WF | AUT | | | X |
| Dashboard – job/project energy/$CO_2$ budgeting | ENE | 1 | MON | WF | USR, ADM | X | X | |
| Identification of best perf. counters for power | ENE | 1 | MON | NOD | ADM | X | X | |
| Job power prediction | ENE | 1 | PRE | JOB | DEV | | X | |
| System-level power/energy consumption prediction | ENE | 1 | PRE | SYS | ADM | | X | |
| Min. $CO_2$e by capping when no “green” source | ENE | 1 | OPT | DCT | ADM, AUT | | | X |
| **CON – Staying Under Constraints** | | | | | | | | |
| Monitoring / Alerting | CON | 2 | MON, ALR | SYS | ADM | X | X | |
| Predictive Maintenance | CON | 2 | PRE | SYS, WF | ADM | X | X | |
| Dashboards and Visualisation of system status | CON | 1 | MON | SYS | ADM | | X | |
| Monitoring / Predictive (power/heat/anomaly) | CON | 1 | MON, PRE, ALR | DCT | ADM | X | X | |
| Node Power Cap | CON | 1 | OPT | NOD | ADM, AUT | X | | |
| System-level and subsystem-level power cap | CON | 1 | OPT | SYS | ADM, AUT | | | X |

- **User Feedback (job perf/config)** — Identify poor performance or misconfiguration in user jobs and return actionable feedback (e.g. pinning, MPI layout).
- **Co-Scheduling** — Co-locate complementary jobs (CPU/GPU, memory- vs. compute-bound, under-utilised cores) at chip/node level to improve throughput with limited interference.
- **Min time under performance threshold** — Minimise time-to-solution while capping allowable slowdown (e.g. ≤5 %) through runtime tuning.
- **Max App Perf under power cap** — Optimise application performance when a power limit is enforced, including memory/CPU power redistribution for memory-bound jobs.
- **Max throughput under Power cap in 1 node (balance)** — Coordinate per-node knobs to maximise single-node throughput under a local power cap.

- **Max throughput under Power cap Scheduling** — Schedule jobs across the system to maximise total throughput while respecting a global power cap.
- **Monitoring / Dashboard (3D, chatbot)** — Provide rich dashboards (incl. DCDB/ExaMon, 3D views, chatbot) for system status and efficiency insight.
- **Alerting anomaly detection hardware failure** — Detect hardware failures/anomalies and alert administrators proactively.
- **Alerting – node health (zombie nodes)** — Detect zombie processes or sick nodes, run pre-launch health checks, and trigger mitigation (job kill/migrate).
- **Detection user behaviour – underutilisation** — Flag patterns like low core utilisation with high memory use to guide better resource requests.
- **Analysis of job submission scripts** — Analyse job scripts to catch parameter mismatches (oversubscription, pinning conflicts) and propose fixes.
- **Dashboard – user behaviour analysis** — Visualise user/job behaviour (size, walltime, utilisation) to inform training, policy, and incentives.

Use cases aiming at minimise energy:

- **Co-optimisation Scheduling** — Perform multi-objective scheduling (performance/energy) with predictive inputs and optimisation loops.
- **Min total system under power cap** — Reduce aggregate system energy under a cap via power-aware placement and budgeting.
- **Min. Energy-to-Solution** — Minimise job energy with bounded performance degradation using runtime control (e.g. EAR).
- **Predefined power config (user/node/island)** — Apply predefined power configurations ("energy tags") at user, node, or island scope.
- **Dynamic Power Opt. (Idle States)** — Cut idle power at system level via adaptive low-power states and policies.
- **Power Reduction when collective comm. synch.** — Lower CPU power during communication/synchronisation phases (e.g. COUNTDOWN).
- **Energy-aware Fair Sharing** — Adjust fair-share priorities based on job energy efficiency, rewarding "good" jobs.
- **Dashboard – job/project energy/$CO_2$ budgeting** — Expose energy/$CO_2$ metrics and budgets per job/project, beyond core-hours accounting.
- **Identification of best perf. counters for power** — Select performance counters that best explain/predict power and arithmetic intensity.
- **Job power prediction** — Predict per-job power traces/peaks over time to improve feasibility checks and budgeting.

- **System-level power/energy consumption prediction** — Forecast system-wide power/energy to plan caps and anticipate peaks.
- **Min. $CO_2$ by capping when no "green" source** — Reduce available power when low-carbon energy is scarce to minimise $CO_2$e.

Use cases aiming at stay under constraints:

- **Monitoring / Alerting** — Monitor IT/infrastructure for errors, outliers, efficiency and capacity issues, raising timely alerts.
- **Predictive Maintenance** — Detect long-term deviations and error accumulation to predict failures and schedule maintenance.
- **Dashboards and Visualisation of system status** — Provide consolidated visual status of the system for admins (health, capacity, efficiency).
- **Monitoring / Predictive (power/heat/anomaly)** — Combine monitoring with ML to predict power/thermal hazards and anomalies at data-centre scale.
- **Node Power Cap** — Enforce node-level power/thermal caps to keep operation within safe limits.
- **System-level and subsystem-level power cap** — Apply and coordinate power caps at partition/rack/system scope to meet facility constraints.

## 4.2 Architecture Decision Records

Based on the detailed discussions at the workshop, the group created and proposed three ADRs, which will be discussed after the workshop; three additional ADRs were raised in the workshop and need more detailed discussion after the workshop discussions.

### Proposed ADRs

The following ADRs were discussed in detail at the Design Workshop:

1. **Inter-component communication through the data plane:** All interactions between SEANERGYS components must occur via the data plane using a unified API.
2. **Data plane service discovery and component registration:** Introduce a locator/metadata service within the data plane to handle component registration and discovery.
3. **Data plane communication:** Standardise communication to two modes — *Publish/Subscribe* and *Request/Response*.

### Draft ADRs

The following ADRs came up at the Design Workshop and were tabled for detailed discussion after the workshop:

1. **Hardware access daemon:** All hardware interactions that may cause conflicts or require privileged access should be managed by a dedicated hardware access daemon.
2. **Data plane serialisation:** Data serialisation and deserialisation will be encapsulated within the Data Plane API to minimise the impact on other components.
3. **Data plane core functionality:** Develop and implement a library that provides a unified API for accessing the Data Plane.

# 5 Summary

The two-day SEANERGYS Design Workshop on September 18 and 19, 2025 at FZJ was attended by about 40 individuals from all the SEANERGYS partners in person and 10-15 individuals in a virtual setting. These included operational experts from the participating centres and specialists in SW architecture and implementation for monitoring systems, machine-learning based data analytics and resource management and scheduling.

During the two days, nine sessions covered a wide range of key topics for the SEANERGYS project, and the workshop provided a hands-on training and exercise opportunity for using the development infrastructure (at the time still based on regular Gitlab hosted by Jülich Supercomputing Centre (JSC)).

The participants agreed to follow a formalised process for creating and modifying artefacts like use cases and their description, requirements emanating from these, and decisions on the architecture and design of the SEANERGYS SW suite. This process is based on the ADR method of capturing such artefacts, and relies heavily on Gitlab features to discuss, ratify and modify them. ADRs are a structured, text-based format for capturing definitions and decisions together with the rationale underpinning them. To keep track of the gestation of ADRs in the project, discussions pertaining to the creation of each ADR are captured as Gitlab issues (to be arranged into "Epics" in the upcoming Gitlab Premium infrastructure), and agreement on an ADR is formalised as the acceptance of a merge requests. This uses GitLab's collaboration capabilities – such as merge requests, threaded discussions, inline comments, version history, and approval workflows and ensures that all steps in coming up with use cases, requirements and architecture decisions are documented and can be traced.

In the use case related sessions, a way to structure and annotate use cases was agreed upon, use cases collected going into the workshop were analysed and correlated accordingly, and additional use cases were proposed in brainstorming sessions and analysed, leading to a total of 31 use cases going out of the workshop.

In light of the use cases and requirements and of prior SW component analysis performed in SEANERGYS work packages 2 and 4, key architecture decisions to be taken were brought up and discussed; the most important of these are the modus operandi and API of the data plane, the need for and integration of latency-optimised communication (for instance for use by parallel runtimes), and the choice of a framework for scheduling and resource management to base the DSRM work on; for the latter, the discussion of advantages/disadvantages of Slurm and Flux was started.

At the 64,000 feet level, the discussions at Design Workshop have significantly advanced the work in SEANERGYS, in areas like the design/development processes to be used, use cases and requirements, and understanding of important architecture choices.

# Acronyms and Abbreviations

## A

**ADR** *(Architecture Decision Record)* A document that captures a single justified design choice that addresses a functional or non-functional requirement that is architecturally significant, along with its rationale . . . . . 6, 10–12, 30, 36, 38
**AI** *(Artificial Intelligence)* . . . . . 13, 31, 33
**AIDAS** *(Artificial Intelligence Data Analytics System)* . . . . . 25
**API** *(Application Programming Interface)* . . . . . 15, 20, 21, 23, 25–27, 36, 37, 39, 40
**APISIX** An open source, dynamic, scalable, and high-performance cloud native API gateway for APIs and microservices . . . . . 21

## B

**BSC** *(Barcelona Supercomputing Centre)* HPC Centre located in Barcelona, Spain . . . . . 39

## C

**CD** *(Continuous Delivery/Deployment)* A software development best practice where automation ensures that software is always ready for release (Delivery) or is released to production (Deployment), usually in conjunction with CI . . . . . 39
**CI** *(Continuous Integration)* A software development best practice where code changes are integrated frequently and automated builds and tests are run on every change or at regular intervals . . . . . 10–12, 39
**CI/CD** *(Continuous Integration and Continuous Delivery/Deployment)* The combination of CI and Continuous Delivery/Deployment (CD) practices in software development . . . . . 25
**CINECA** A centre of excellence in the Italian and European ecosystem for supercomputing technologies, supporting and developing frontier applications . . . . . 39
**CLI** *(Command-Line Interface)* . . . . . 12
**COUNTDOWN** A run-time library for application-agnostic energy saving in MPI communication primitives, developed by Università di Bologna (UNIBO) and CINECA . . . . . 33, 35
**CPU** *(Central Processing Unit)* . . . . . 14, 15, 18, 20, 22, 34, 35

## D

**DCDB** *(Data Center Data Base)* Monitoring framework for the acquisition of telemetry data developed by Leibniz-Rechenzentrum (LRZ) . . . . . 33, 35
**DEEP-SEA** DEEP – Software for Exascale Architectures . . . . . 27, 33
**DSRM** *(Dynamic Scheduling and Resource Management)* . . . . . 26, 38

## E

**EAR** *(Energy Aware Runtime)* System software for energy management, accounting, and optimisation for super computers developed by Barcelona Supercomputing Centre (BSC) . . . . . 31, 33, 35
**ECP** *(Exascale Computing Project)* A U.S. project operated between 2016 and 2024 responsible for delivering a capable exascale computing ecosystem, including software, applications, and hardware technology . . . . . 27

**EuroHPC JU** *(European High Performance Computing Joint Undertaking)* A legal and funding entity, created in 2018 and located in Luxembourg; Group of EU-state members joining forces to foster HPC in Europe . . . . . 27, 40

**ExaMon** *(Exascale Monitoring)* A highly scalable framework for the performance and energy monitoring of HPC servers, developed by UNIBO . . . . . 35

## F

**Flux** Resource and Job Management System (RJMS) developed by LLNL, USA . . . . . 26–29, 38

**FZJ** *(Forschungszentrum Jülich)* Research Centre located in Jülich, Germany . . . . . 7, 26, 28, 38

## G

**GPL** *(GNU General Public License)* . . . . . 26

**GPU** *(Graphics Processing Unit)* . . . . . 14, 15, 18, 20, 22, 23, 34

## H

**HPC** *(High Performance Computing)* . . . . . 13, 15–17, 19, 22–29, 31, 33, 39, 40

**HPSF** *(High Performance Software Foundation)* A neutral hub for open source high performance software; part of the nonprofit Linux Foundation . . . . . 27, 29

## J

**JSC** *(Jülich Supercomputing Centre)* Jülich Supercomputing Centre GmbH, Jülich, Germany . . . . . 38

## L

**LGPL** *(GNU Lesser General Public License)* . . . . . 27

**LLNL** *(Lawrence Livermore National Laboratory)* . . . . . 27–29, 40

**LRZ** *(Leibniz-Rechenzentrum)* Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften. Computing Centre located in Garching, Germany. Short name of long partner acronym “BADW-LRZ” . . . . . 39

## M

**ML** *(Machine Learning)* . . . . . 16, 36

**MPI** *(Message-Passing Interface)* An API specification typically used in parallel programs that allows processes to communicate with one another by sending and receiving messages . . . . 21, 22, 24, 27, 34, 39

## O

**OAR** RJMS developed by UGA . . . . . 26

## P

**Pub/Sub** *(Publish/Subscribe)* A messaging pattern where publishers send messages to a central topic without knowing which subscribers will receive them . . . . . 25, 26

## R

**REGALE** A EuroHPC JU project to develop an open architecture to equip next generation HPC applications with exascale capabilities . . . . . 33

**RJMS** *(Resource and Job Management System)* Software that tracks and monitors the hardware deployed in a data centre, arbitrates access as users submit work they would like to run, and manages scheduling and placement of that work onto the available hardware . . . . . 40, 41

## S

**Slurm** RJMS developed by SchedMD LLC . . . . . 21, 26–29, 38

## U

**UGA** *(Université Grenoble Alpes)* . . . . . 26, 40
**UNIBO** *(Università di Bologna)* . . . . . 39, 40