Ergonomics and efficiency of workflows on HPC clusters

Jakub Beránek

Supervisor
Ing. Jan Martinovič, Ph.D.

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Ergonomics and efficiency of workflows on HPC clusters

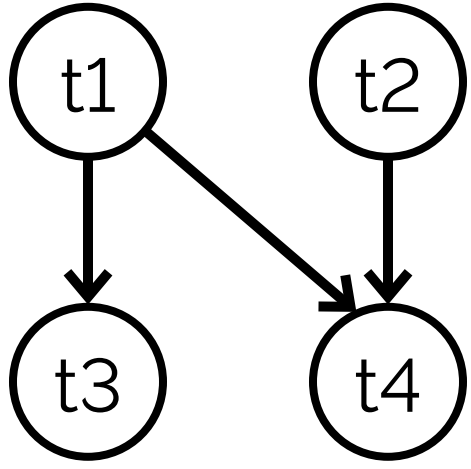
Goal

Help users execute task workflows
on HPC clusters in an easy & efficient way

Ergonomics and efficiency of workflows on HPC clusters

Goal

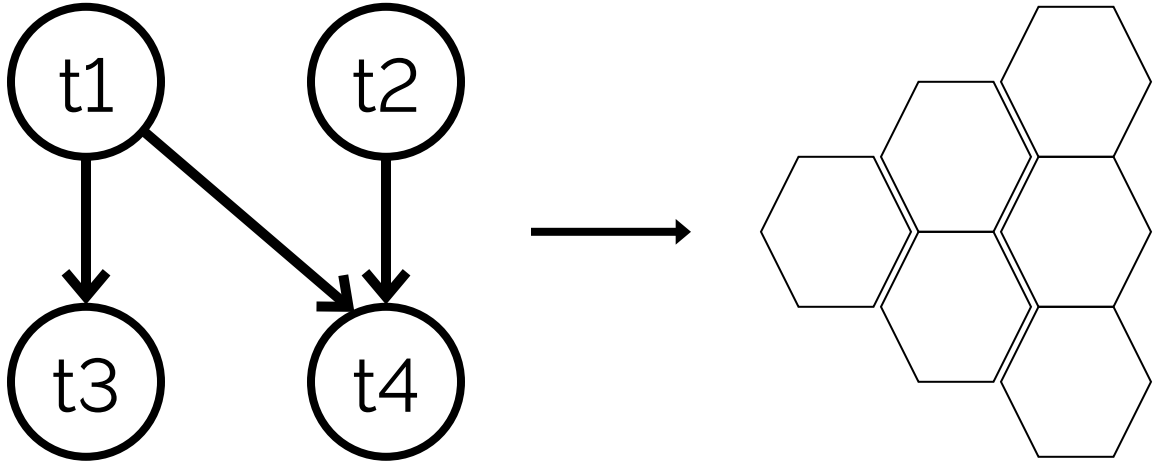
Help users execute task workflows
on HPC clusters in an easy & efficient way



Ergonomics and efficiency of workflows on HPC clusters

Goal

Help users execute task workflows
on HPC clusters in an easy & efficient way



Objectives

1. Identify HPC workflow challenges

Objectives

1. Identify HPC workflow challenges
2. Design approaches for overcoming them

Objectives

1. Identify HPC workflow challenges
2. Design approaches for overcoming them
3. Implement them in a task runtime

Objectives

1. Identify HPC workflow challenges
2. Design approaches for overcoming them
3. Implement them in a task runtime
4. Analyze results on real use-cases

Motivation

Ergonomics and efficiency
of **workflows on HPC clusters**

Workflows

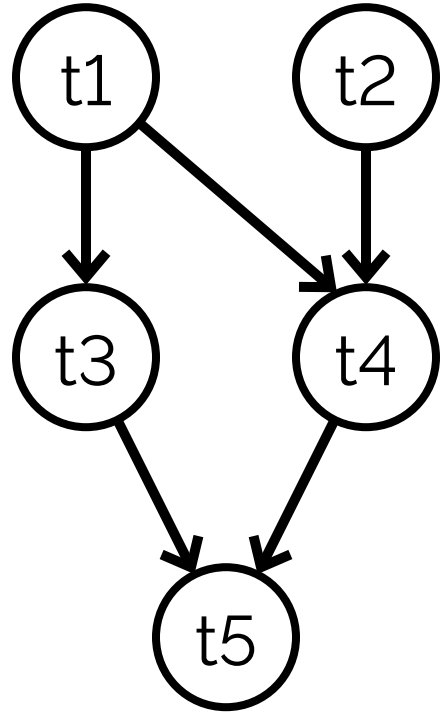
= task graphs, pipelines, task DAGs*, …

* Directed Acyclic Graph

Workflows

= task graphs, pipelines, task DAGs*, …

Popular programming model

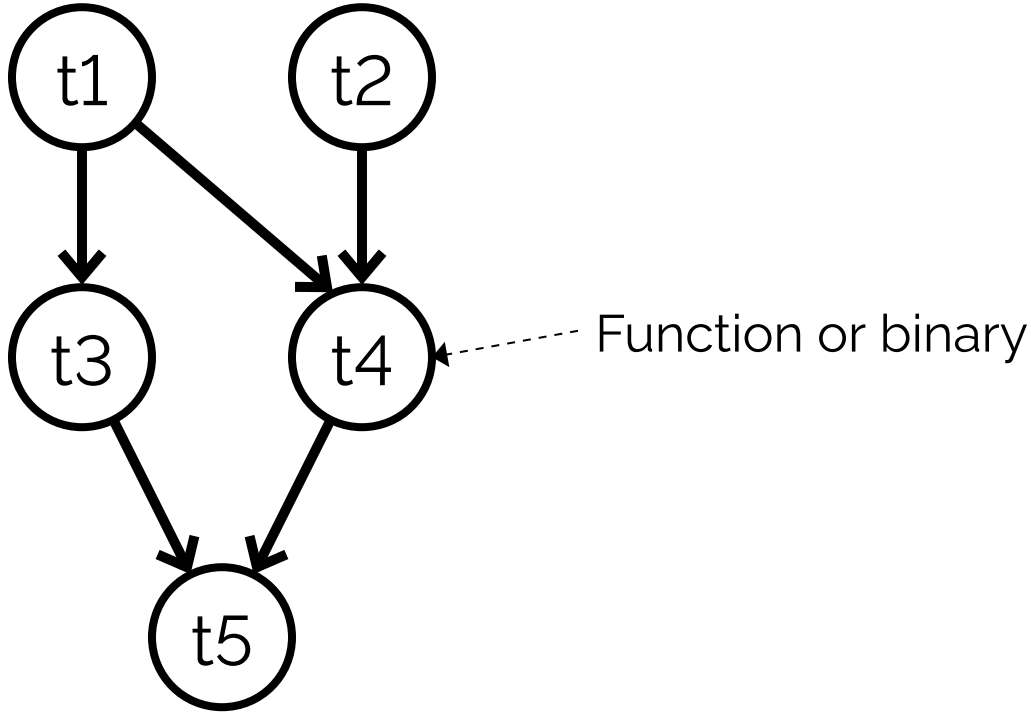


* Directed Acyclic Graph

Workflows

= task graphs, pipelines, task DAGs*, …

Popular programming model

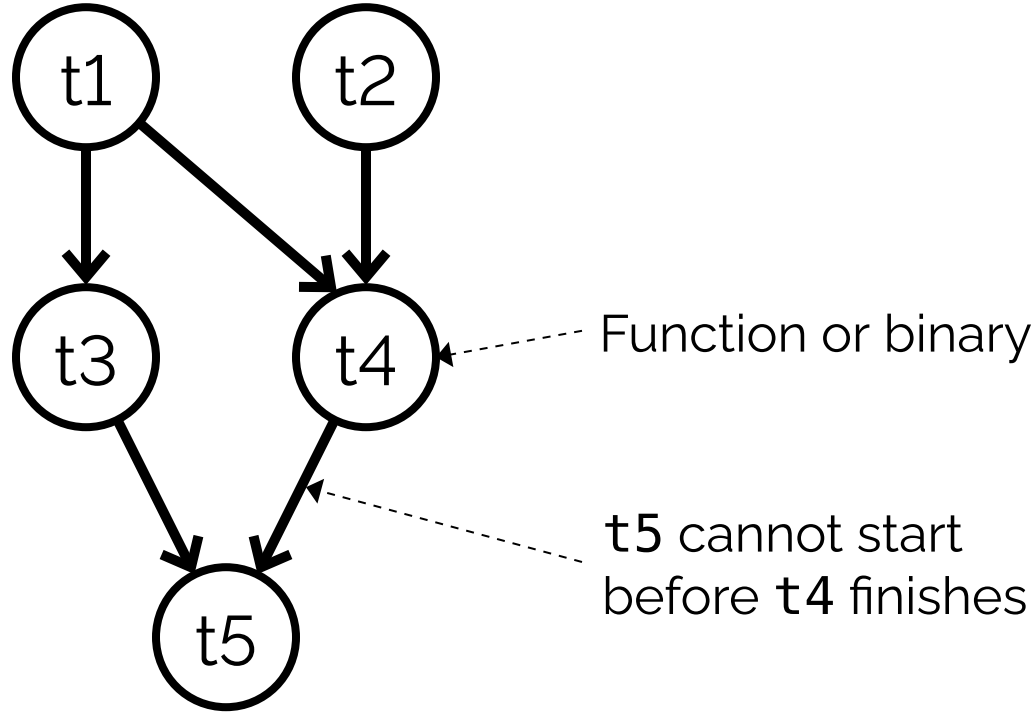


* Directed Acyclic Graph

Workflows

= task graphs, pipelines, task DAGs*, …

Popular programming model



* Directed Acyclic Graph

Main workflow benefits

- **Implicit parallelism**
  - Build a DAG vs. use MPI

Main workflow benefits

- **Implicit parallelism**
  - Build a DAG vs. use MPI
  - Extracted by a task runtime

Main workflow benefits

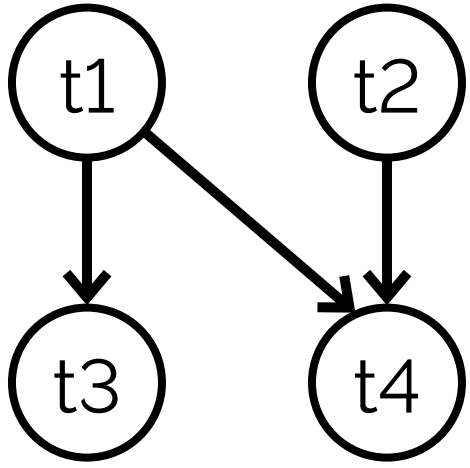
- **Implicit parallelism**
  - Build a DAG vs. use MPI
  - Extracted by a task runtime
- High-level description
  - Python/DSL vs. C/C++

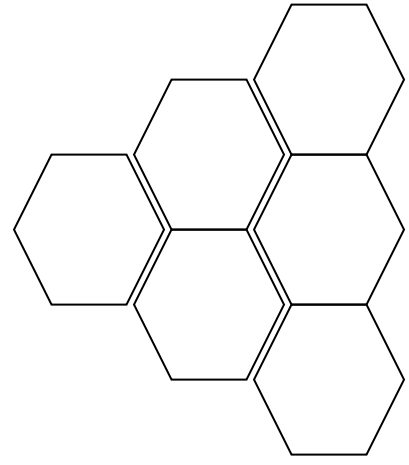
Main workflow benefits

- **Implicit parallelism**
  - Build a DAG vs. use MPI
  - Extracted by a task runtime
- High-level description
  - Python/DSL vs. C/C++
- Portability
  - Use-cases, hardware

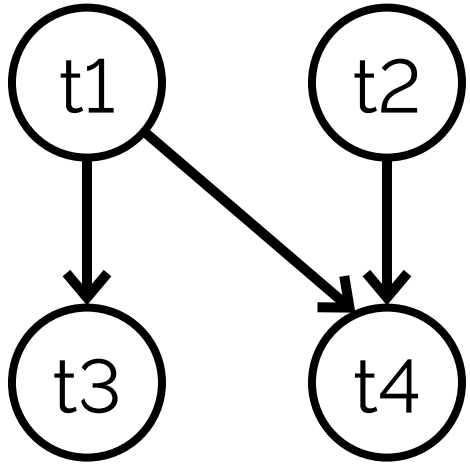
Task graph challenges on HPC clusters

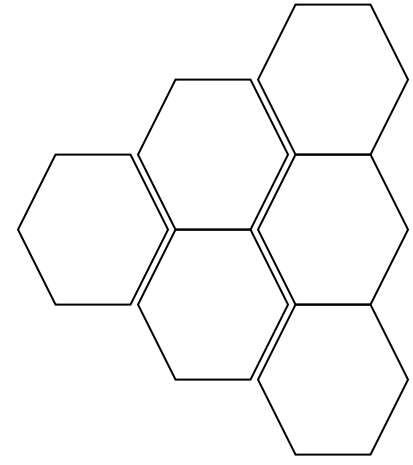
Job manager



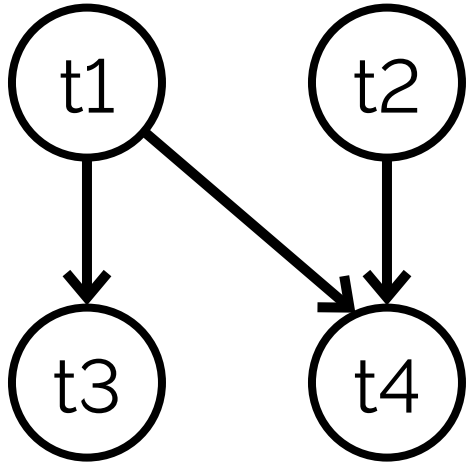


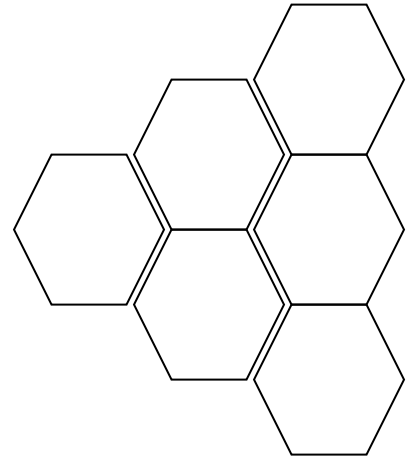
Job manager





Job manager



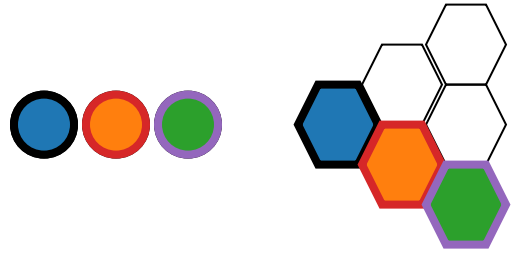


How to map tasks to PBS jobs?

Submitting task graphs into PBS

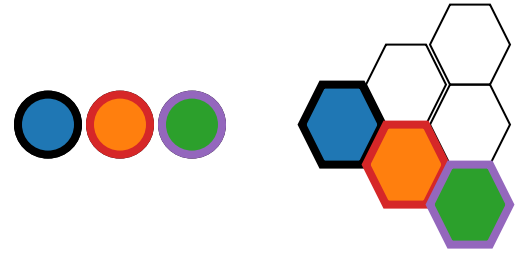
Submitting task graphs into PBS

- Submit each task as a job



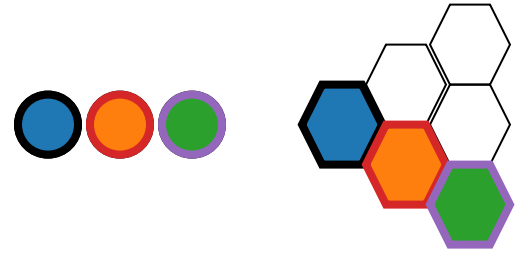
Submitting task graphs into PBS

- Submit each task as a job
  - Massive overhead (millions of jobs)



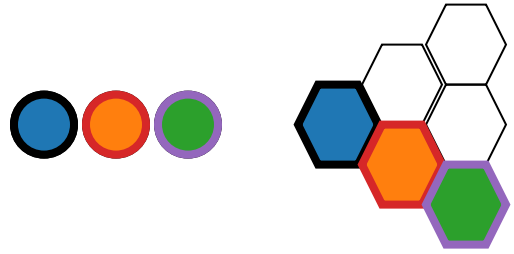
Submitting task graphs into PBS

- Submit each task as a job
  - Massive overhead (millions of jobs)
  - Job count limits



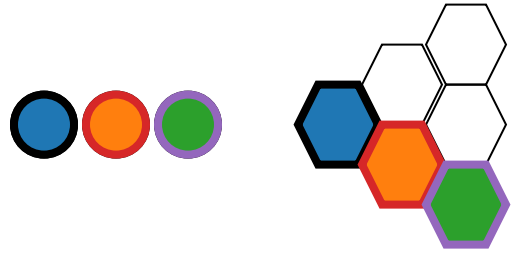
Submitting task graphs into PBS

- Submit each task as a job
  - Massive overhead (millions of jobs)
  - Job count limits
  - **Node granularity**



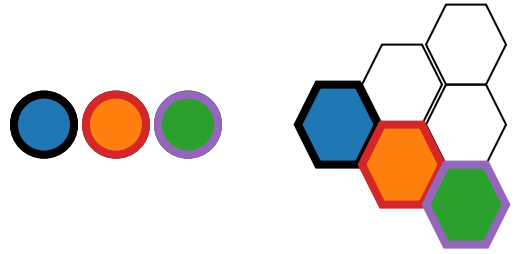
Submitting task graphs into PBS

- Submit each task as a job
  - Massive overhead (millions of jobs)
  - Job count limits
  - **Node granularity**
  - Difficult with dependencies

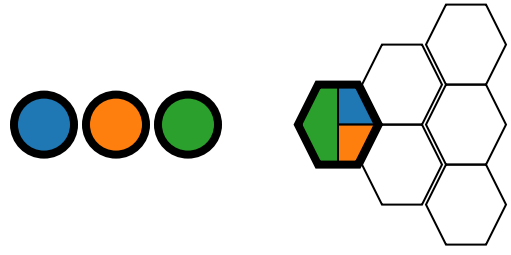


Submitting task graphs into PBS

- Submit each task as a job
  - Massive overhead (millions of jobs)
  - Job count limits
  - **Node granularity**
  - Difficult with dependencies

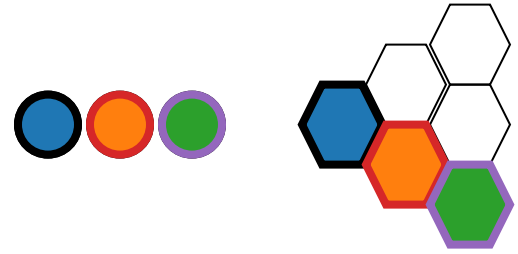


- Submit task graph as a single job
  - Only for small-ish task graphs

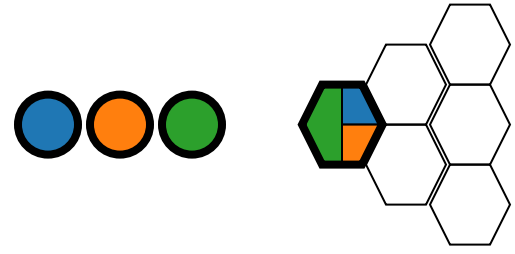


Submitting task graphs into PBS

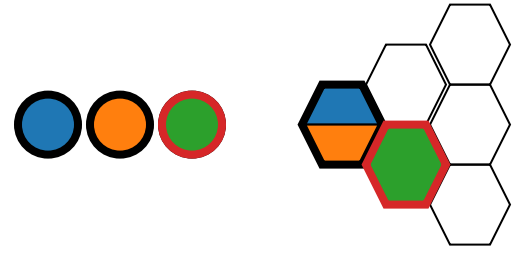
- Submit each task as a job
  - Massive overhead (millions of jobs)
  - Job count limits
  - **Node granularity**
  - Difficult with dependencies



- Submit task graph as a single job
  - Only for small-ish task graphs

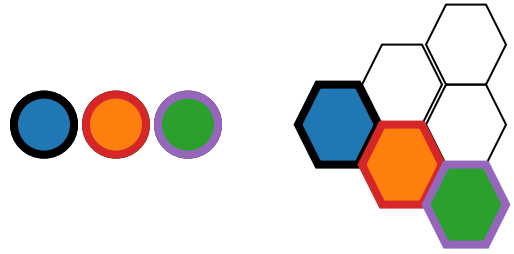


- Split task graph into multiple jobs
  - Lot of work

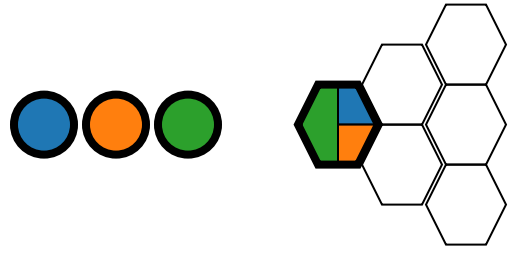


Submitting task graphs into PBS

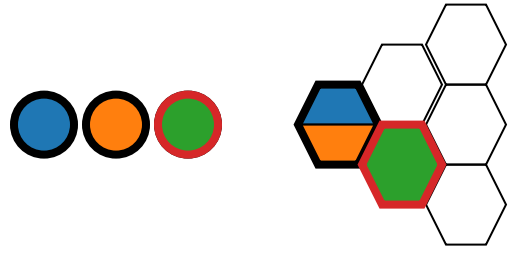
- Submit each task as a job
  - Massive overhead (millions of jobs)
  - Job count limits
  - **Node granularity**
  - Difficult with dependencies



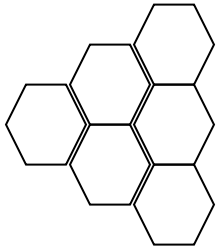
- Submit task graph as a single job
  - Only for small-ish task graphs



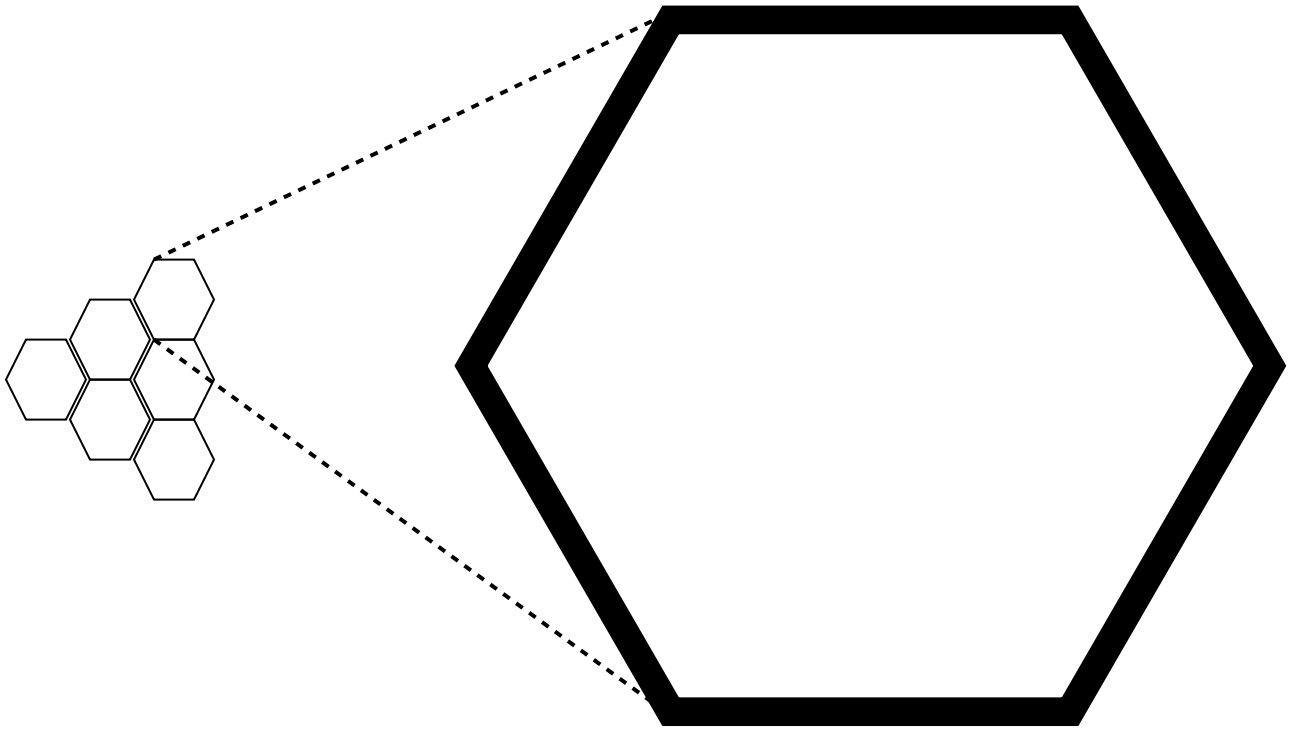
- Split task graph into multiple jobs
  - Lot of work
  - No load balancing across jobs



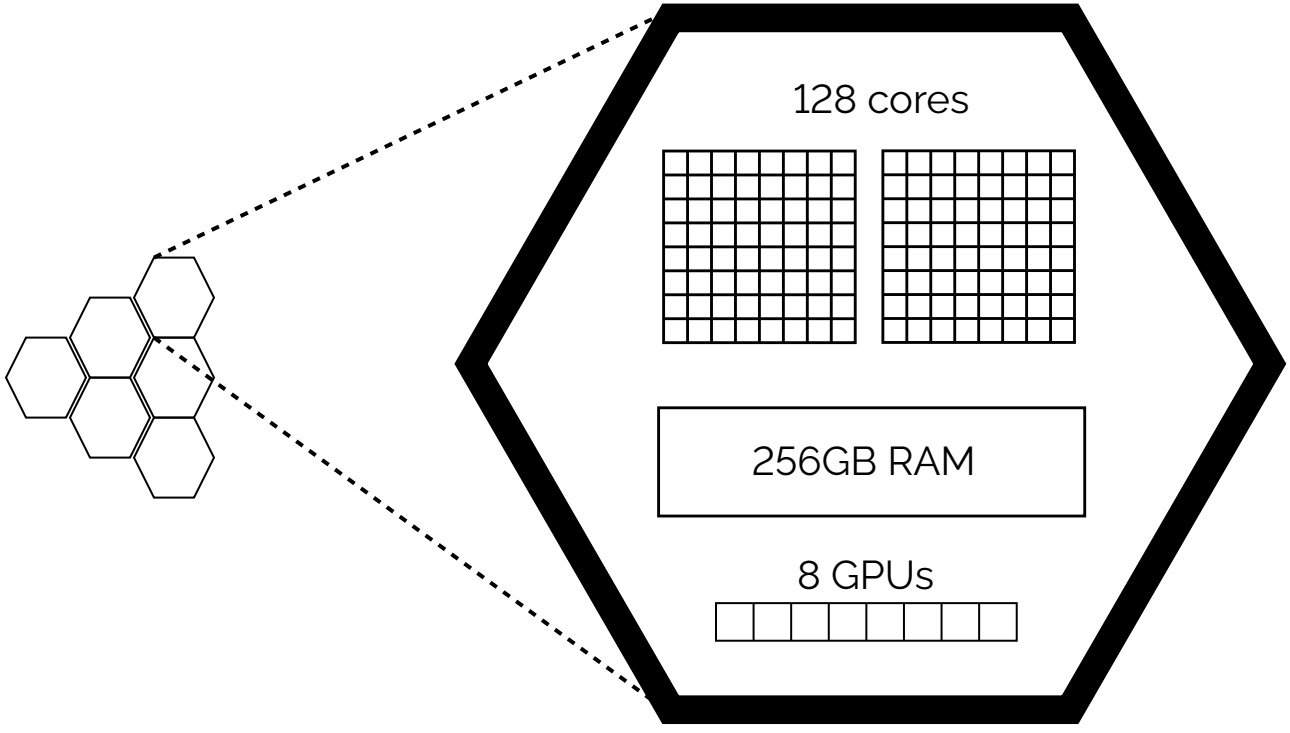
Resource management



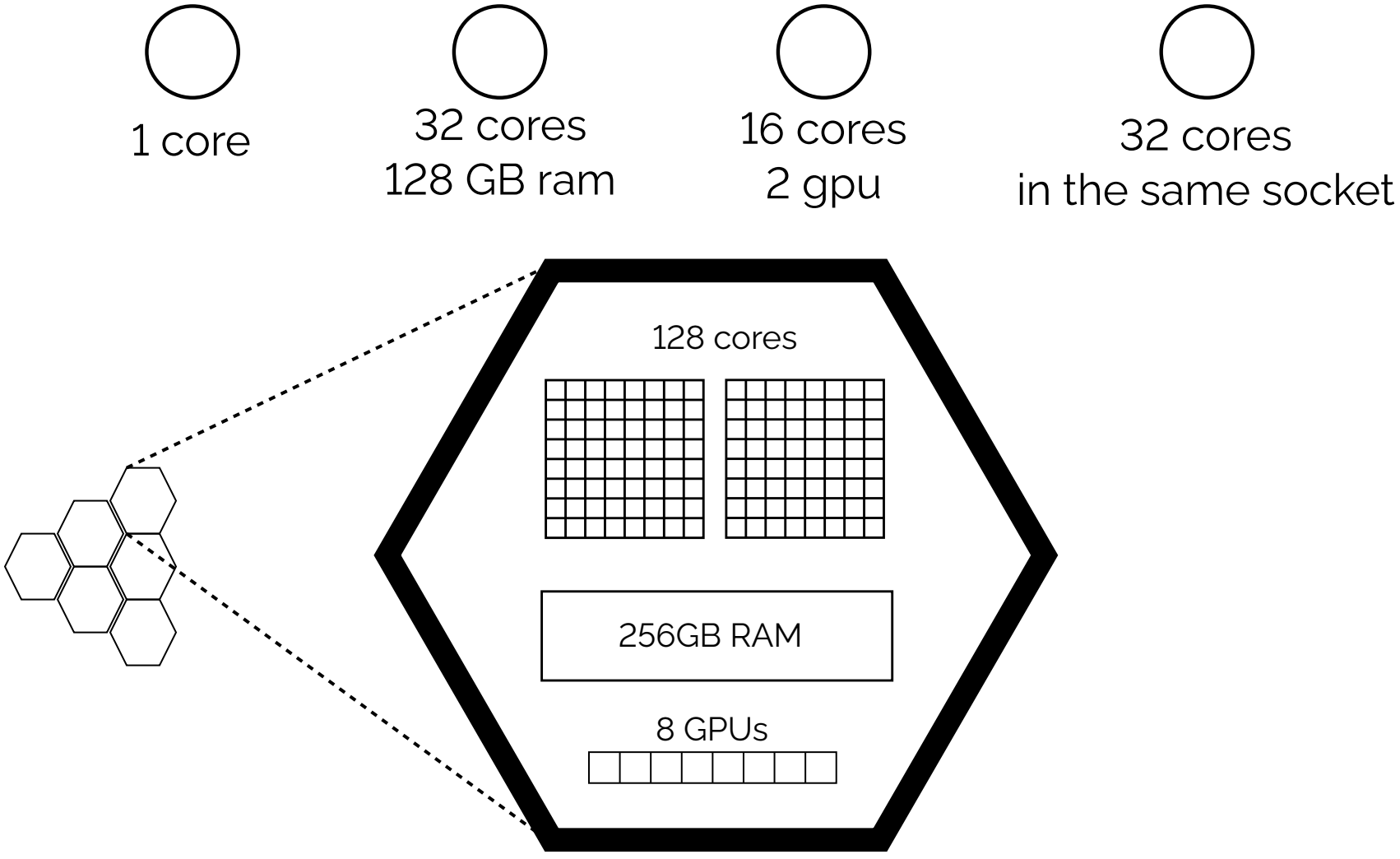
Resource management



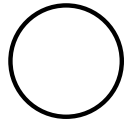
Resource management



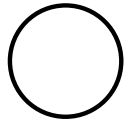
Resource management



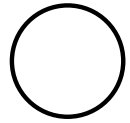
Resource management



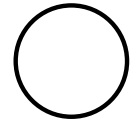
1 core



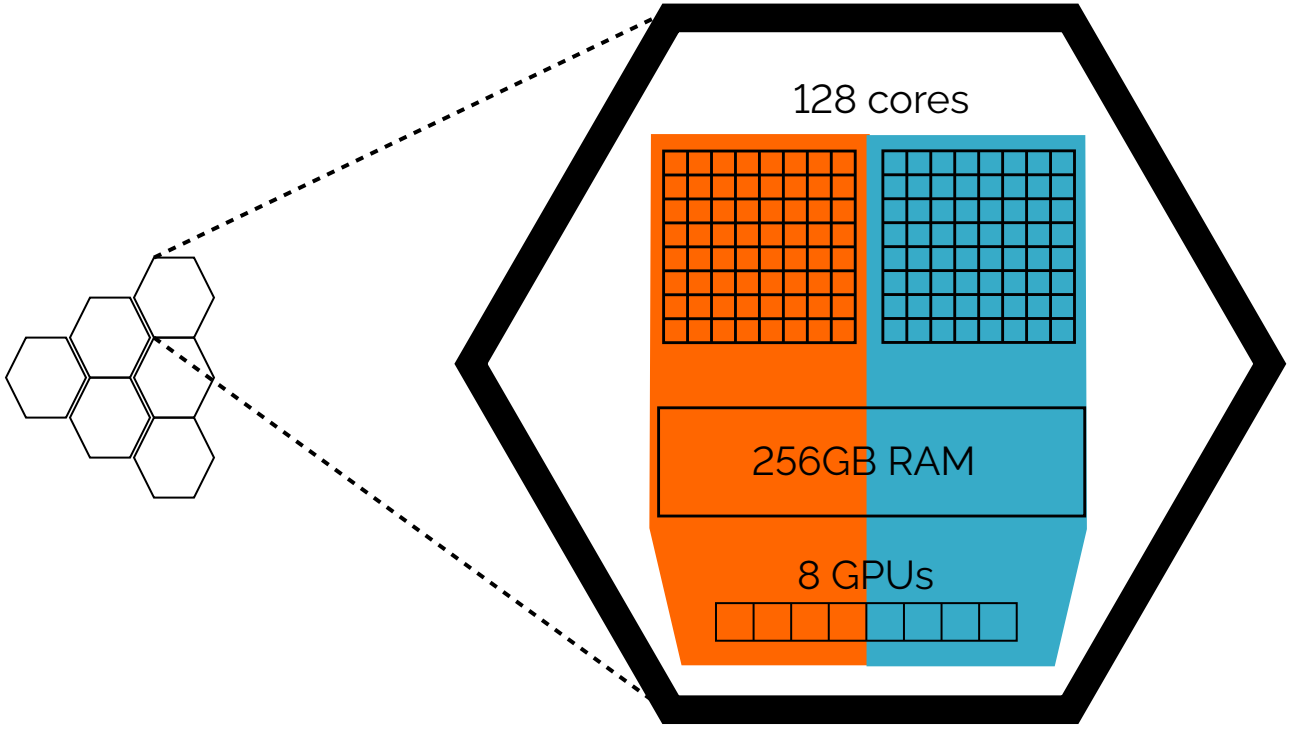
32 cores
128 GB ram



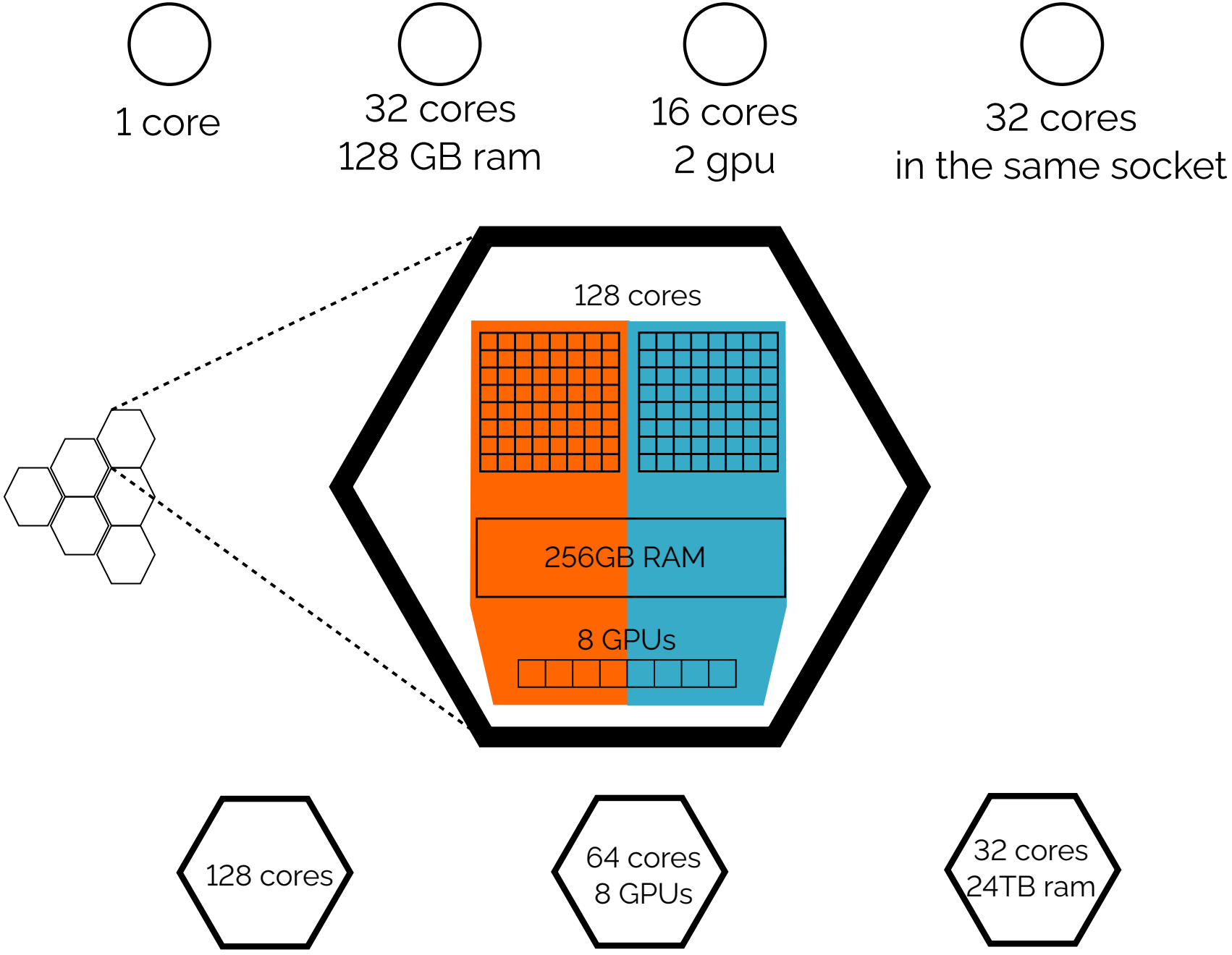
16 cores
2 gpu



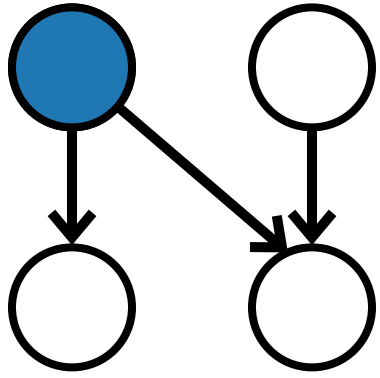
32 cores
in the same socket

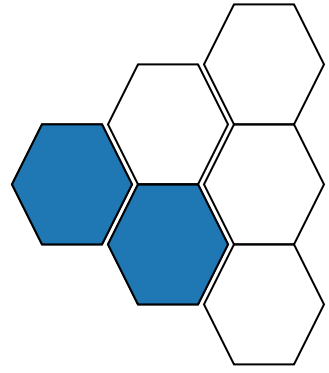


Resource management

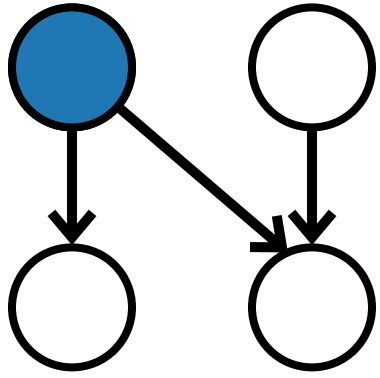


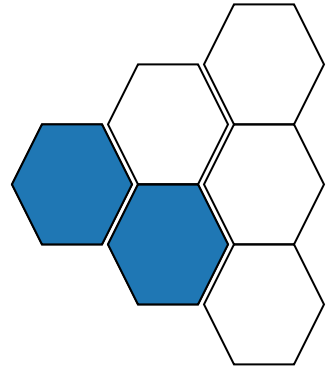
Multi-node tasks





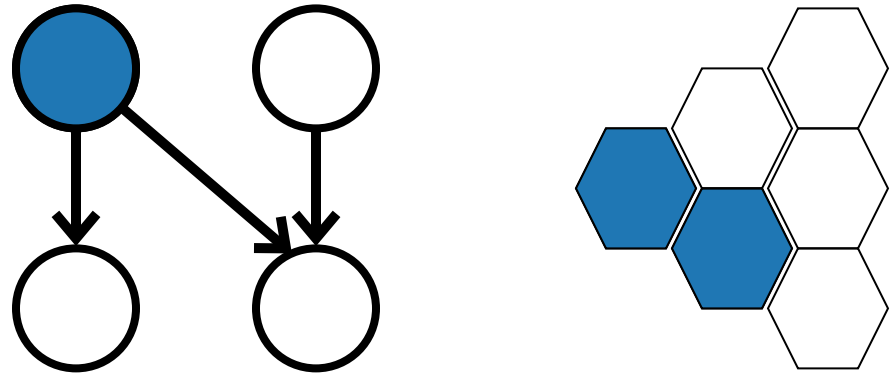
Multi-node tasks





- MPI application within a task

Multi-node tasks



- MPI application within a task
- Complicates scheduling and data transfers

Other workflow challenges on HPC

- Scalability

Other workflow challenges on HPC

- Scalability
  - Millions of tasks, hundreds of nodes

Other workflow challenges on HPC

- Scalability
  - Millions of tasks, hundreds of nodes
  - Communication bottleneck

Other workflow challenges on HPC

- Scalability
  - Millions of tasks, hundreds of nodes
  - Communication bottleneck
- Fault tolerance

Other workflow challenges on HPC

- Scalability
  - Millions of tasks, hundreds of nodes
  - Communication bottleneck
- Fault tolerance
- Iterative computation

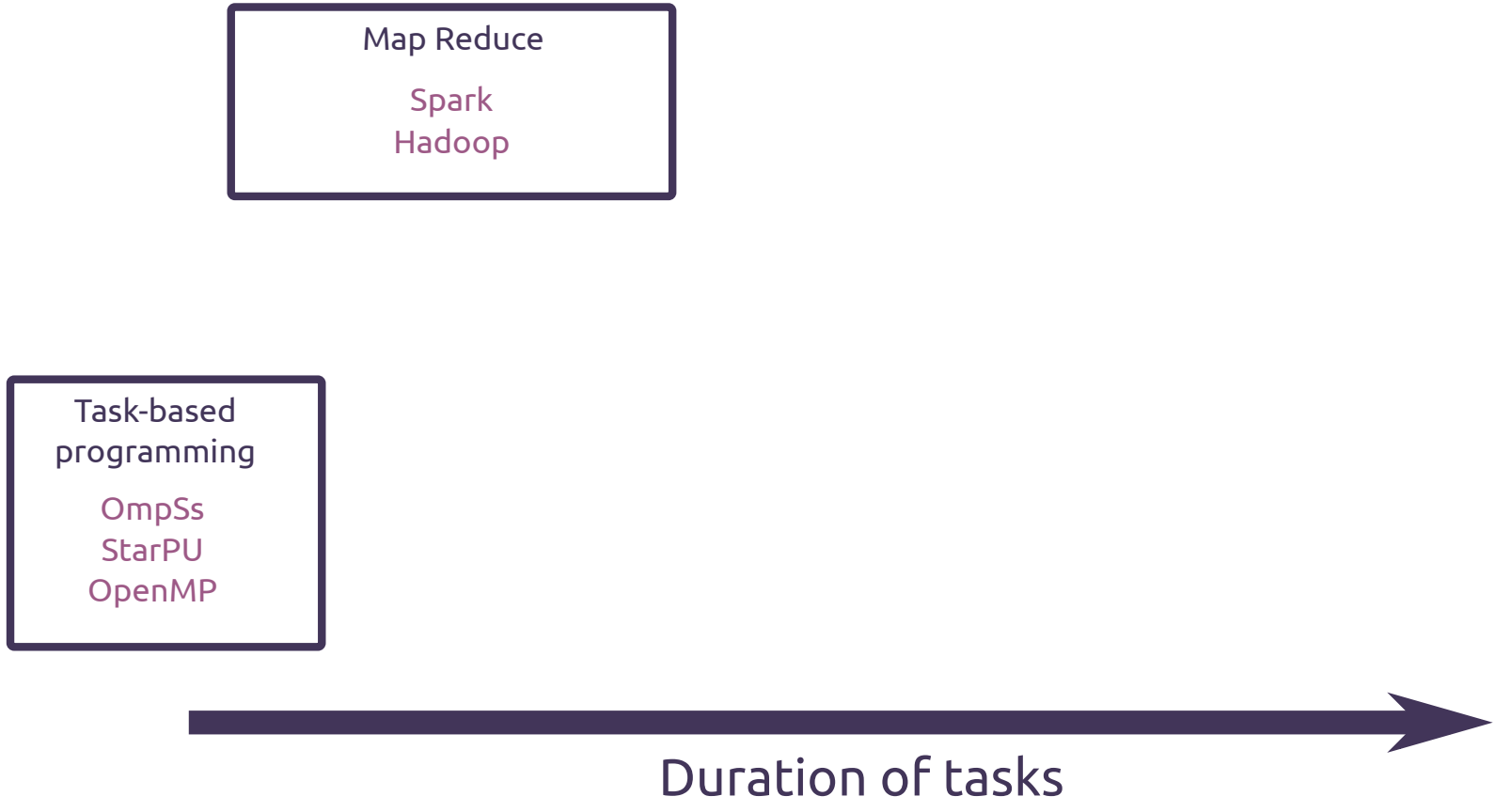
Other workflow challenges on HPC

- Scalability
  - Millions of tasks, hundreds of nodes
  - Communication bottleneck
- Fault tolerance
- Iterative computation
- …

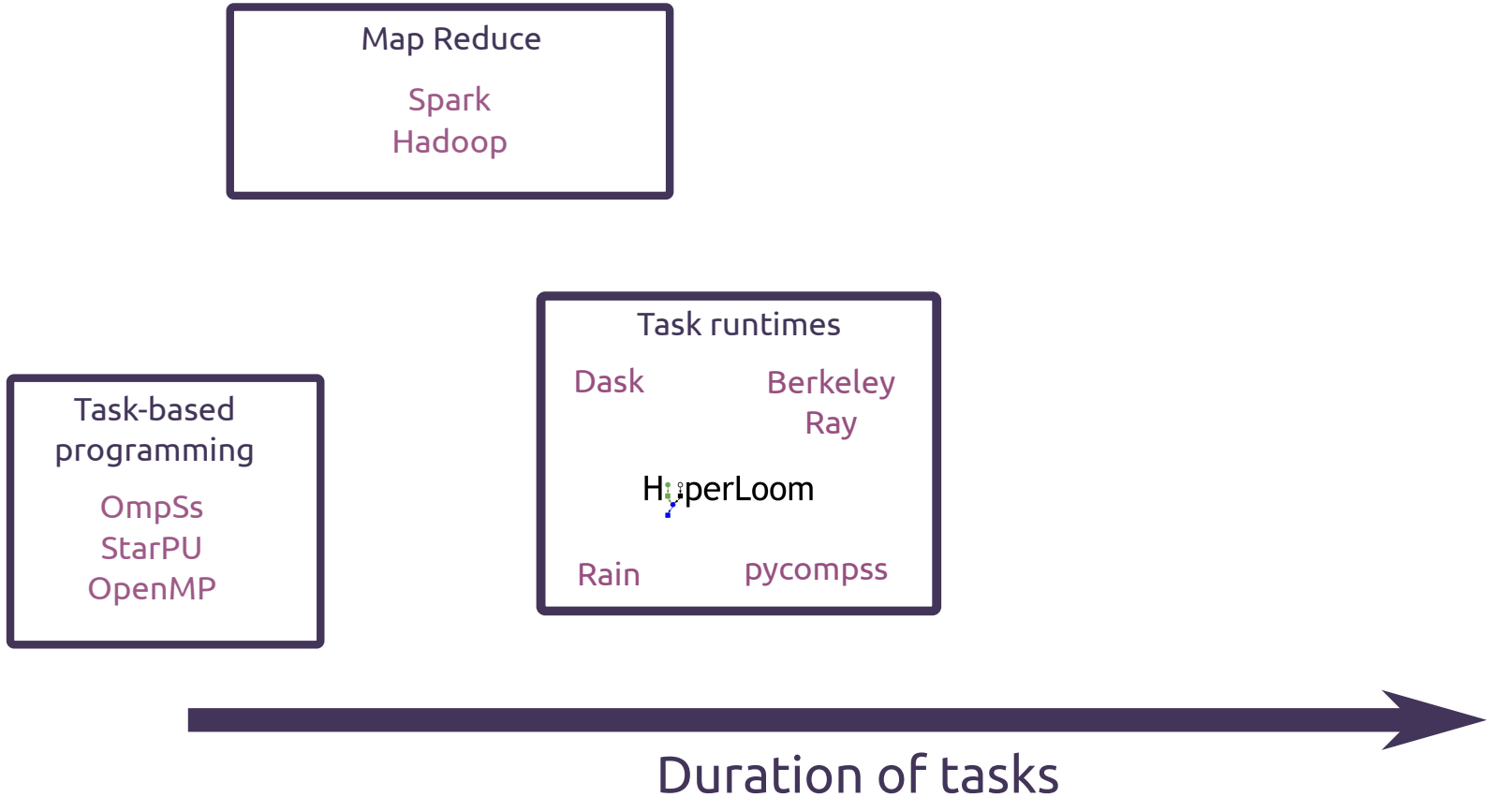
State of the art: task duration

State of the art: task duration

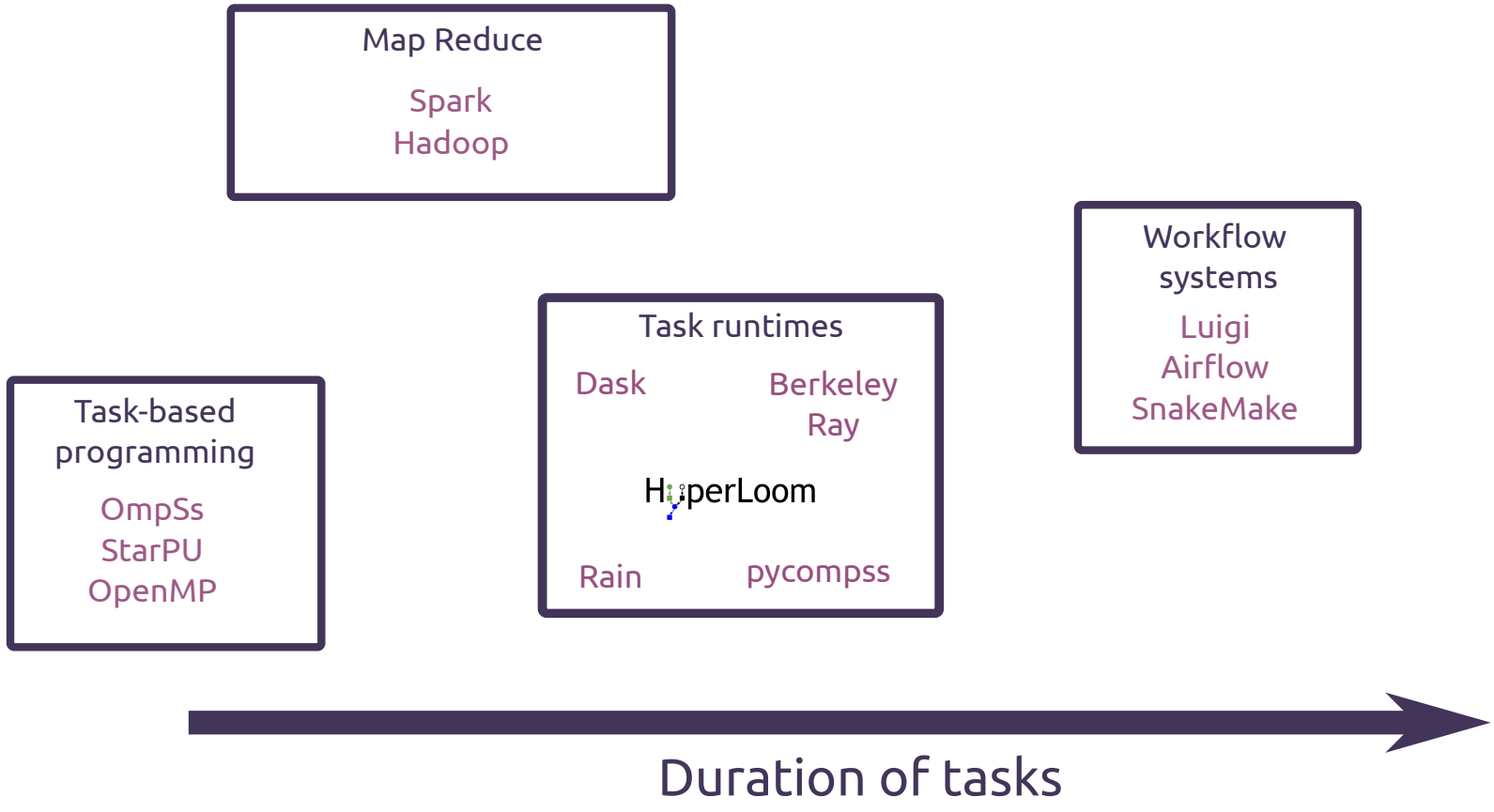
State of the art: task duration



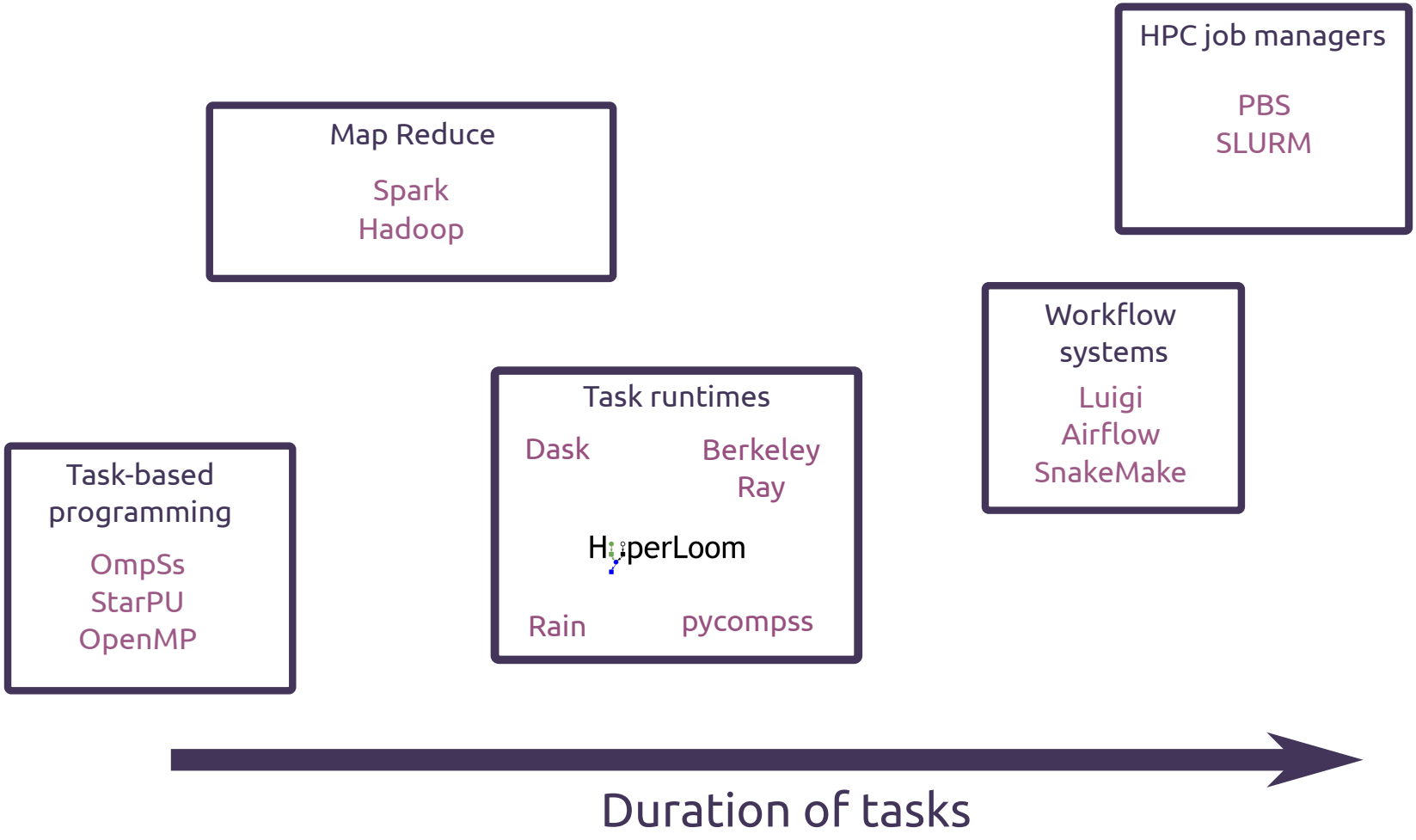
State of the art: task duration



State of the art: task duration

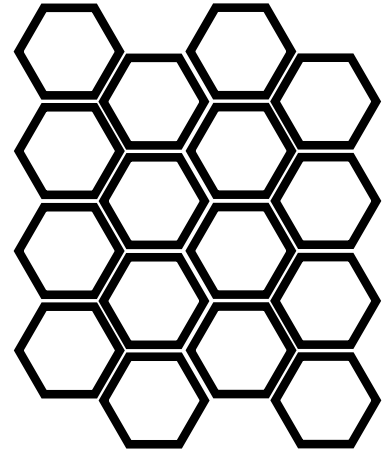


State of the art: task duration



State of the art: job manager

PBS/Slurm



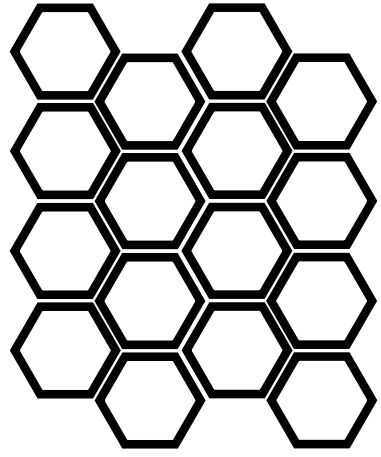
Login nodes

Compute nodes

State of the art: job manager

SnakeMake
Nextflow
Airflow

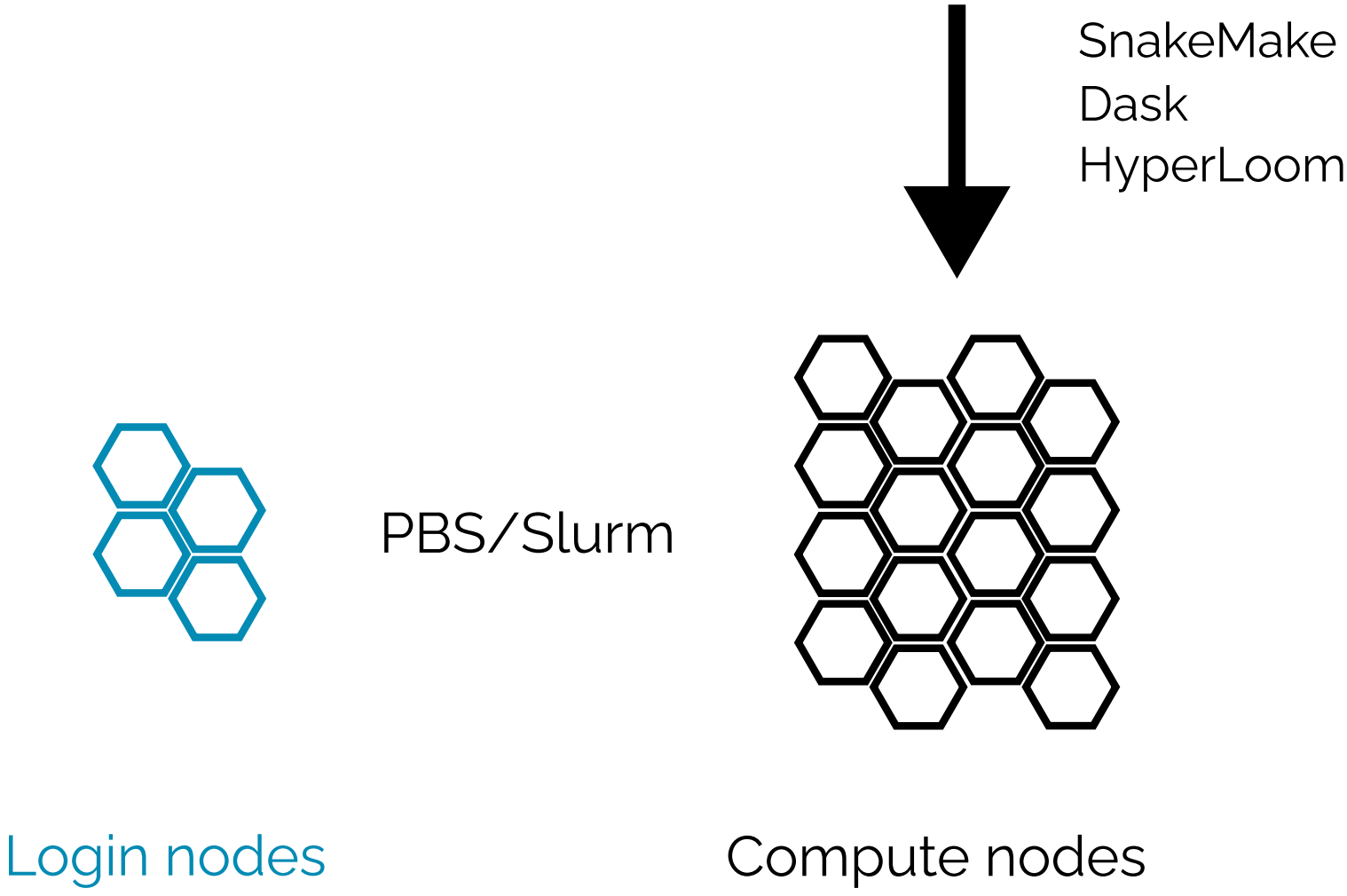
PBS/Slurm



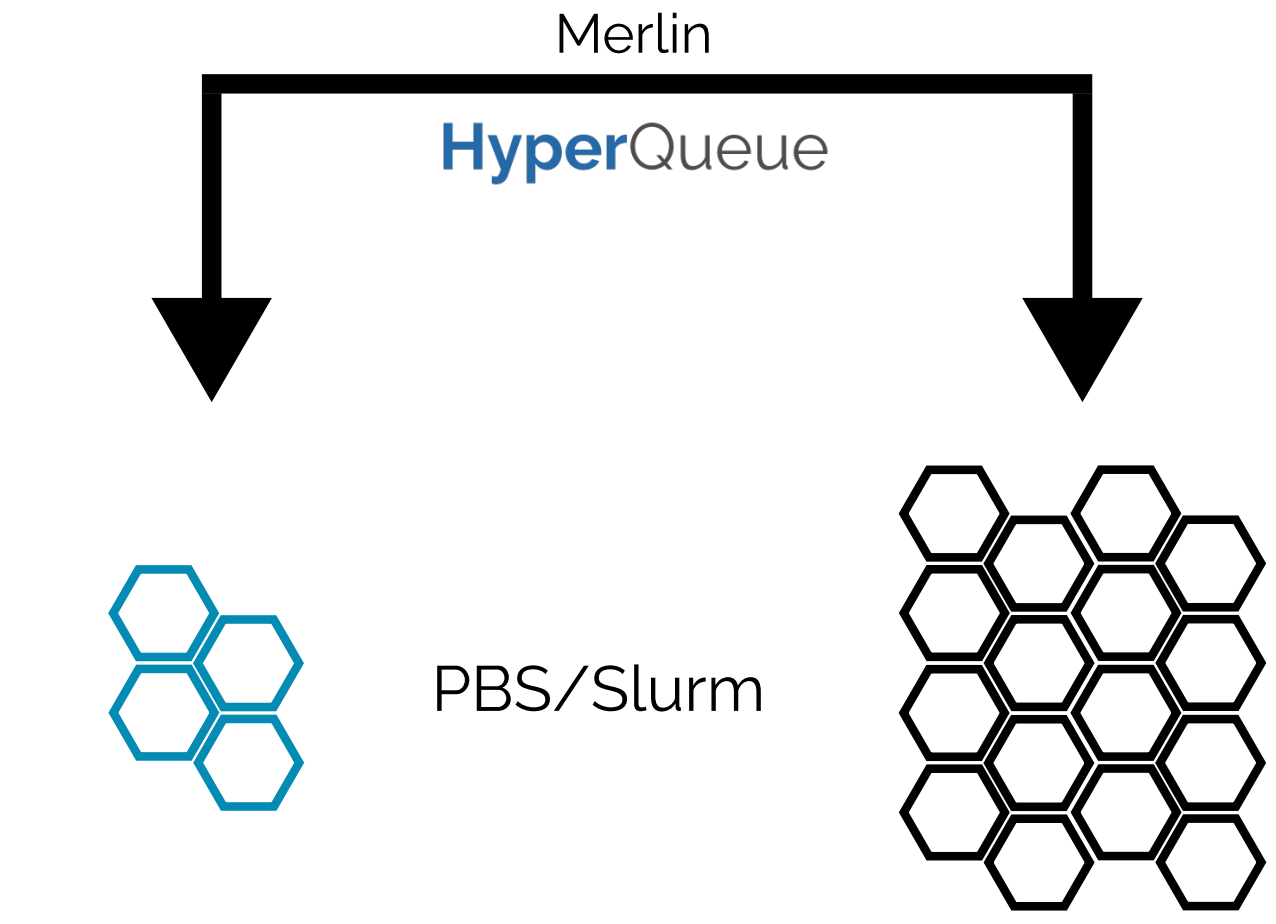
Login nodes

Compute nodes

State of the art: job manager



State of the art: job manager



State of the art: resources & multi-node

- Resource requirements: ✓ (simple)

State of the art: resources & multi-node

- Resource requirements: ✓ (simple)
- Multi-node tasks: x

Ergonomics and **efficiency**
of workflows on HPC clusters

The Journal of Supercomputing
https://doi.org/10.1007/s11227-022-04438-y

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek[1] · Stanislav Böhm[1] · Vojtěch Cima[1]

Accepted: 10 March 2022
© The Author(s) 2022

Abstract
Task graphs provide a simple way to describe scientific workflows (sets of tasks with dependencies) that can be executed on both HPC clusters and in the cloud. An important aspect of executing such graphs is the used scheduling algorithm. Many scheduling heuristics have been proposed in existing works; nevertheless, they are often tested in oversimplified environments. We provide an extensible simulation environment designed for prototyping and benchmarking task schedulers, which contains implementations of various scheduling algorithms and is open-sourced, in order to be fully reproducible. We use this environment to perform a comprehensive analysis of workflow scheduling algorithms with a focus on quantifying the effect of scheduling challenges that have so far been mostly neglected, such as delays between scheduler invocations or partially unknown task durations. Our results indicate that network models used by many previous works might produce results that are off by an order of magnitude in comparison to a more realistic model. Additionally, we show that certain implementation details of scheduling algorithms which are often neglected can have a large effect on the scheduler's performance, and they should thus be described in great detail to enable proper evaluation.

Keywords Distributed computing · DAG scheduling · Task Scheduling · Network models

Jakub Beránek, Stanislav Böhm and Vojtech Cima these authors contributed equally to this work.

✉ Jakub Beránek
jakub.beranek@vsb.cz

Stanislav Böhm
stanislav.bohm@vsb.cz

Vojtěch Cima
vojtech.cima@vsb.cz

[1] IT4Innovations, VSB – Technical University of Ostrava, Ostrava, Czech Republic

Published online: 14 April 2022

Springer

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek, Stanislav Böhm, Vojtěch Cima
(The Journal of Supercomputing 2022)

The Journal of Supercomputing
https://doi.org/10.1007/s11227-022-04438-y

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek[1] · Stanislav Böhm[1] · Vojtěch Cima[1]

Accepted: 10 March 2022

Abstract
Task graphs provide a simple way to describe scientific workflows (sets of tasks with dependencies) that can be executed on both HPC clusters and in the cloud. An important aspect of executing such graphs is the used scheduling algorithm. Many scheduling heuristics have been proposed in existing works; nevertheless, they are often tested in oversimplified environments. We provide an extensible simulation environment designed for prototyping and benchmarking task schedulers, which contains implementations of various scheduling algorithms and is open-sourced, in order to be fully reproducible. We use this environment to perform a comprehensive analysis of workflow scheduling algorithms with a focus on quantifying the effect of scheduling challenges that have so far been mostly neglected, such as delays between scheduler invocations or partially unknown task durations. Our results indicate that network models used by many previous works might produce results that are off by an order of magnitude in comparison to a more realistic model. Additionally, we show that certain implementation details of scheduling algorithms which are often neglected can have a large effect on the scheduler's performance, and they should thus be described in great detail to enable proper evaluation.

Keywords Distributed computing · DAG scheduling · Task Scheduling · Network models

Jakub Beránek, Stanislav Böhm and Vojtech Cima these authors contributed equally to this work.

✉ Jakub Beránek
jakub.beranek@vsb.cz

Stanislav Böhm
stanislav.bohm@vsb.cz

Vojtěch Cima
vojtech.cima@vsb.cz

[1] IT4Innovations, VSB – Technical University of Ostrava, Ostrava, Czech Republic

Published online: 14 April 2022

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek, Stanislav Böhm, Vojtěch Cima
(The Journal of Supercomputing 2022)

- Compare scheduler performance

The Journal of Supercomputing
https://doi.org/10.1007/s11227-022-04438-y

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek[1] · Stanislav Böhm[1] · Vojtěch Cima[1]

Accepted: 10 March 2022

Abstract
Task graphs provide a simple way to describe scientific workflows (sets of tasks with dependencies) that can be executed on both HPC clusters and in the cloud. An important aspect of executing such graphs is the used scheduling algorithm. Many scheduling heuristics have been proposed in existing works; nevertheless, they are often tested in oversimplified environments. We provide an extensible simulation environment designed for prototyping and benchmarking task schedulers, which contains implementations of various scheduling algorithms and is open-sourced, in order to be fully reproducible. We use this environment to perform a comprehensive analysis of workflow scheduling algorithms with a focus on quantifying the effect of scheduling challenges that have so far been mostly neglected, such as delays between scheduler invocations or partially unknown task durations. Our results indicate that network models used by many previous works might produce results that are off by an order of magnitude in comparison to a more realistic model. Additionally, we show that certain implementation details of scheduling algorithms which are often neglected can have a large effect on the scheduler's performance, and they should thus be described in great detail to enable proper evaluation.

Keywords Distributed computing · DAG scheduling · Task Scheduling · Network models

Jakub Beránek, Stanislav Böhm and Vojtech Cima these authors contributed equally to this work.

✉ Jakub Beránek
jakub.beranek@vsb.cz

Stanislav Böhm
stanislav.bohm@vsb.cz

Vojtech Cima
vojtech.cima@vsb.cz

[1] IT4Innovations, VSB – Technical University of Ostrava, Ostrava, Czech Republic

Published online: 14 April 2022

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek, Stanislav Böhm, Vojtěch Cima
(The Journal of Supercomputing 2022)

- Compare scheduler performance
- Analyze neglected factors

The Journal of Supercomputing
https://doi.org/10.1007/s11227-022-04438-y

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek[1] · Stanislav Böhm[1] · Vojtěch Cima[1]

Accepted: 10 March 2022

Abstract
Task graphs provide a simple way to describe scientific workflows (sets of tasks with dependencies) that can be executed on both HPC clusters and in the cloud. An important aspect of executing such graphs is the used scheduling algorithm. Many scheduling heuristics have been proposed in existing works; nevertheless, they are often tested in oversimplified environments. We provide an extensible simulation environment designed for prototyping and benchmarking task schedulers, which contains implementations of various scheduling algorithms and is open-sourced, in order to be fully reproducible. We use this environment to perform a comprehensive analysis of workflow scheduling algorithms with a focus on quantifying the effect of scheduling challenges that have so far been mostly neglected, such as delays between scheduler invocations or partially unknown task durations. Our results indicate that network models used by many previous works might produce results that are off by an order of magnitude in comparison to a more realistic model. Additionally, we show that certain implementation details of scheduling algorithms which are often neglected can have a large effect on the scheduler's performance, and they should thus be described in great detail to enable proper evaluation.

Keywords Distributed computing · DAG scheduling · Task Scheduling · Network models

Jakub Beránek, Stanislav Böhm and Vojtech Cima these authors contributed equally to this work.

✉ Jakub Beránek
jakub.beranek@vsb.cz
Stanislav Böhm
stanislav.bohm@vsb.cz
Vojtěch Cima
vojtech.cima@vsb.cz

[1] IT4Innovations, VSB – Technical University of Ostrava, Ostrava, Czech Republic

Published online: 14 April 2022

Springer

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek, Stanislav Böhm, Vojtěch Cima
(The Journal of Supercomputing 2022)

- Compare scheduler performance
- Analyze neglected factors
  - Delay between scheduling

The Journal of Supercomputing
https://doi.org/10.1007/s11227-022-04438-y

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek[1] · Stanislav Böhm[1] · Vojtěch Cima[1]

Accepted: 10 March 2022

Abstract
Task graphs provide a simple way to describe scientific workflows (sets of tasks with dependencies) that can be executed on both HPC clusters and in the cloud. An important aspect of executing such graphs is the used scheduling algorithm. Many scheduling heuristics have been proposed in existing works; nevertheless, they are often tested in oversimplified environments. We provide an extensible simulation environment designed for prototyping and benchmarking task schedulers, which contains implementations of various scheduling algorithms and is open-sourced, in order to be fully reproducible. We use this environment to perform a comprehensive analysis of workflow scheduling algorithms with a focus on quantifying the effect of scheduling challenges that have so far been mostly neglected, such as delays between scheduler invocations or partially unknown task durations. Our results indicate that network models used by many previous works might produce results that are off by an order of magnitude in comparison to a more realistic model. Additionally, we show that certain implementation details of scheduling algorithms which are often neglected can have a large effect on the scheduler's performance, and they should thus be described in great detail to enable proper evaluation.

Keywords Distributed computing · DAG scheduling · Task Scheduling · Network models

Jakub Beránek, Stanislav Böhm and Vojtech Cima these authors contributed equally to this work.

✉ Jakub Beránek
jakub.beranek@vsb.cz

Stanislav Böhm
stanislav.bohm@vsb.cz

Vojtěch Cima
vojtech.cima@vsb.cz

[1] IT4Innovations, VSB – Technical University of Ostrava, Ostrava, Czech Republic

Published online: 14 April 2022

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek, Stanislav Böhm, Vojtěch Cima
(The Journal of Supercomputing 2022)

- Compare scheduler performance
- Analyze neglected factors
  - Delay between scheduling
  - Knowledge about task durations

The Journal of Supercomputing
https://doi.org/10.1007/s11227-022-04438-y

Analysis of workflow schedulers in simulated distributed environments

Jakub Beránek[1] · Stanislav Böhm[1] · Vojtěch Cima[1]

Accepted: 10 March 2022

Abstract
Task graphs provide a simple way to describe scientific workflows (sets of tasks with dependencies) that can be executed on both HPC clusters and in the cloud. An important aspect of executing such graphs is the used scheduling algorithm. Many scheduling heuristics have been proposed in existing works; nevertheless, they are often tested in oversimplified environments. We provide an extensible simulation environment designed for prototyping and benchmarking task schedulers, which contains implementations of various scheduling algorithms and is open-sourced, in order to be fully reproducible. We use this environment to perform a comprehensive analysis of workflow scheduling algorithms with a focus on quantifying the effect of scheduling challenges that have so far been mostly neglected, such as delays between scheduler invocations or partially unknown task durations. Our results indicate that network models used by many previous works might produce results that are off by an order of magnitude in comparison to a more realistic model. Additionally, we show that certain implementation details of scheduling algorithms which are often neglected can have a large effect on the scheduler's performance, and they should thus be described in great detail to enable proper evaluation.

Keywords Distributed computing · DAG scheduling · Task Scheduling · Network models

Jakub Beránek, Stanislav Böhm and Vojtech Cima these authors contributed equally to this work.

✉ Jakub Beránek
jakub.beranek@vsb.cz
Stanislav Böhm
stanislav.bohm@vsb.cz
Vojtech Cima
vojtech.cima@vsb.cz

[1] IT4Innovations, VSB – Technical University of Ostrava, Ostrava, Czech Republic

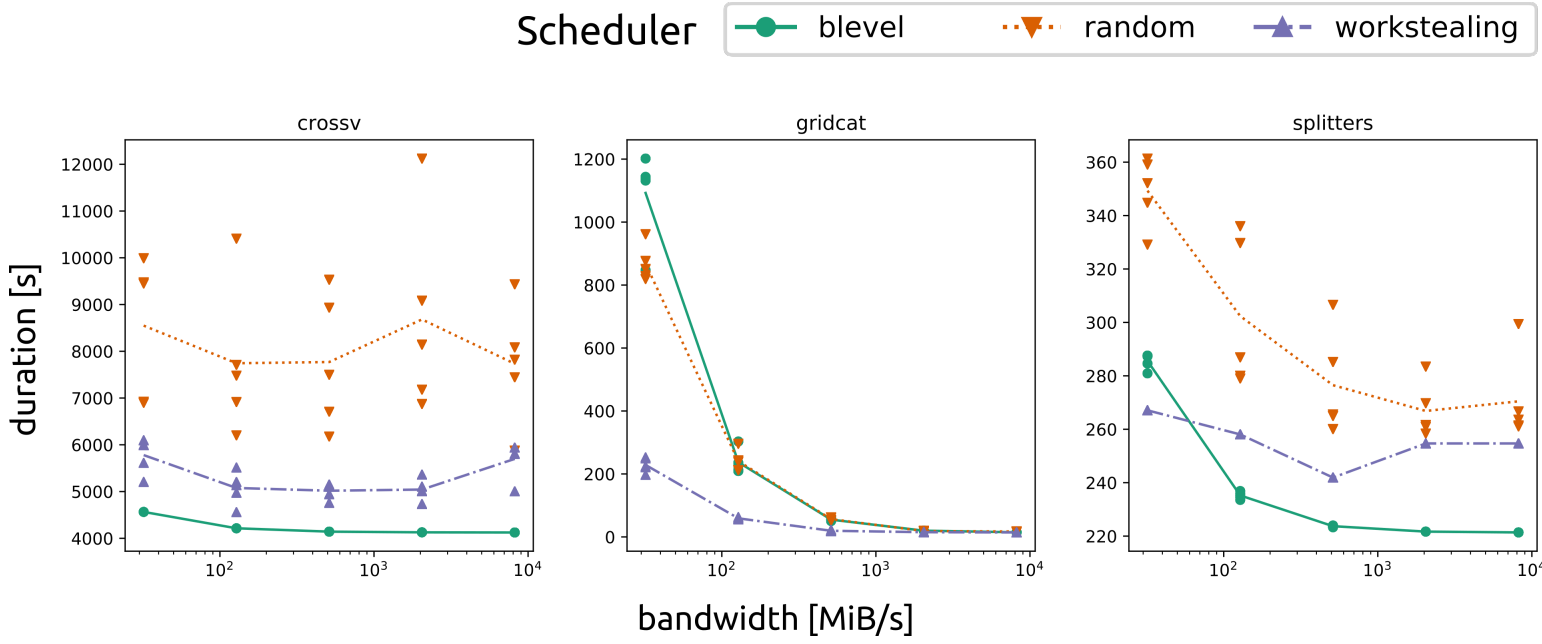
Published online: 14 April 2022

Analysis of workflow schedulers in simulated distributed environments

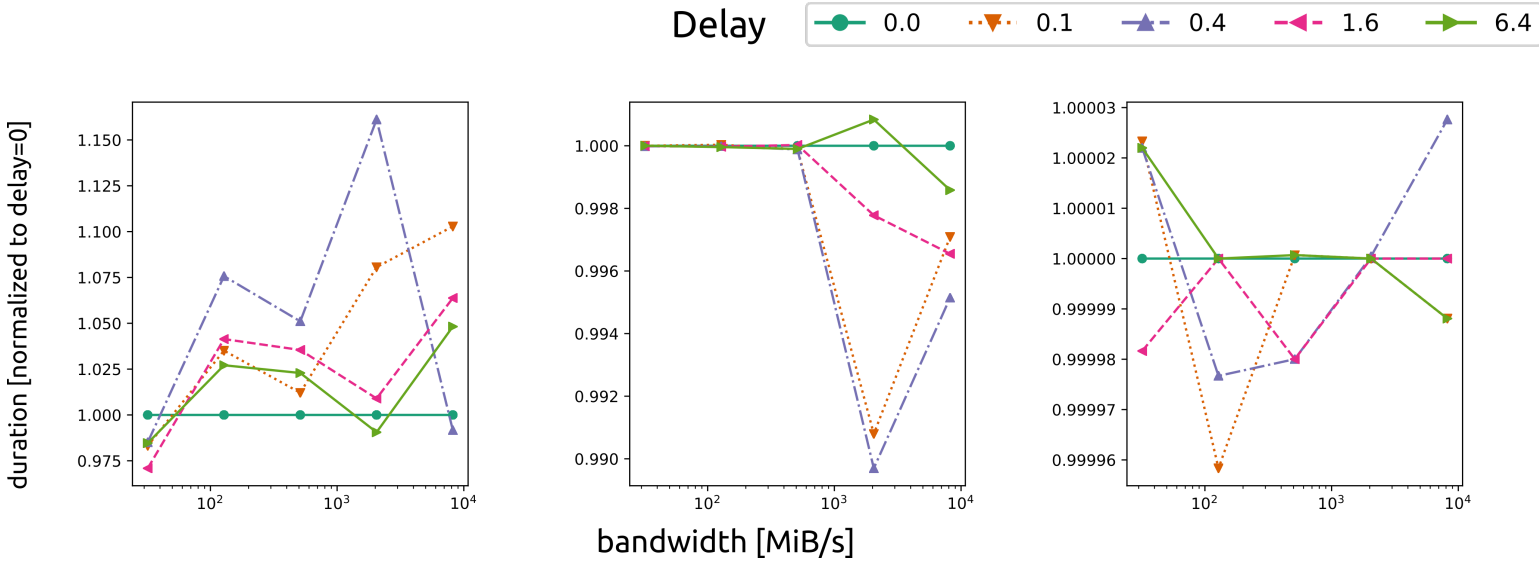
Jakub Beránek, Stanislav Böhm, Vojtěch Cima
(The Journal of Supercomputing 2022)

- Compare scheduler performance
- Analyze neglected factors
  - Delay between scheduling
  - Knowledge about task durations
  - Network model

Runtime vs. network speed



Scheduling delay effect



Outcome

- Most competitive schedulers

Outcome

- Most competitive schedulers
  - Work-stealing (used in Dask)

Outcome

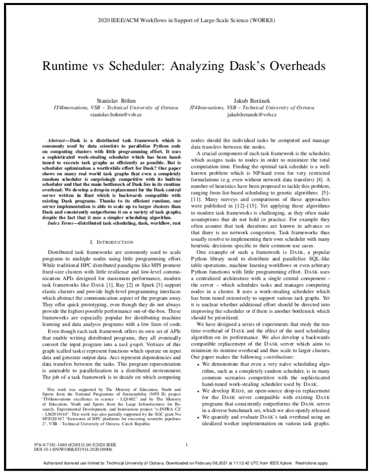
- Most competitive schedulers
  - Work-stealing (used in Dask)
  - B-level (basic heuristic)

Outcome

- Most competitive schedulers
  - Work-stealing (used in Dask)
  - B-level (basic heuristic)
- Implementation details matter a lot!

Outcome

- Most competitive schedulers
  - Work-stealing (used in Dask)
  - B-level (basic heuristic)
- Implementation details matter a lot!
- Open source scheduler simulator ESTEE
  - `github.com/it4innovations/estee`


2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)

Runtime vs Scheduler: Analyzing Dask's Overheads

Stanislav Böhm
IT4Innovations, VSB – Technical University of Ostrava
stanislav.bohm@vsb.cz

Jakub Beránek
IT4Innovations, VSB – Technical University of Ostrava
jakub.beranek@vsb.cz

Abstract—**Dask is a distributed task framework which is commonly used by data scientists to parallelize Python code on computing clusters with little programming effort. It uses a sophisticated work-stealing scheduler which has been hand-tuned to execute task graphs as efficiently as possible. But is scheduler optimization a worthwhile effort for Dask? Our paper shows on many real world task graphs that even a completely random scheduler is surprisingly competitive with its built-in scheduler and that the main bottleneck of Dask lies in its runtime overhead. We develop a drop-in replacement for the Dask central server written in Rust which is backwards compatible with existing Dask programs. Thanks to its efficient runtime, our server implementation is able to scale up to larger clusters than Dask and consistently outperforms it on a variety of task graphs, despite the fact that it uses a simpler scheduling algorithm.**

Index Terms—**distributed task scheduling, dask, workflow, rust**

I. Introduction

Distributed task frameworks are commonly used to scale programs to multiple nodes using little programming effort. While traditional HPC distributed paradigms like MPI promote fixed-size clusters with little resilience and low-level communication APIs designed for maximum performance, modern task frameworks like Dask [1], Ray [2] or Spark [3] support elastic clusters and provide high-level programming interfaces which abstract the communication aspect of the program away. They offer quick prototyping, even though they do not always provide the highest possible performance out-of-the-box. These frameworks are especially popular for distributing machine learning and data analysis programs with a few lines of code.

Even though each task framework offers its own set of APIs that enable writing distributed programs, they all eventually convert the input program into a *task graph*. Vertices of this graph (called tasks) represent functions which operate on input data and generate output data. Arcs represent dependencies and data transfers between the tasks. This program representation is amenable to parallelization in a distributed environment. The job of a task framework is to decide on which computing nodes should the individual tasks be computed and manage data transfers between the nodes.

A crucial component of each task framework is the scheduler, which assigns tasks to nodes in order to minimize the total computation time. Finding the optimal task schedule is a well-known problem which is NP-hard even for very restricted formulations (e.g. even without network data transfers) [4]. A number of heuristics have been proposed to tackle this problem, ranging from list-based scheduling to genetic algorithms [5]–[11]. Many surveys and comparisons of these approaches were published in [12]–[15]. Yet applying these algorithms to modern task frameworks is challenging, as they often make assumptions that do not hold in practice. For example they often assume that task durations are known in advance or that there is no network congestion. Task frameworks thus usually resolve to implementing their own scheduler with many heuristic decisions specific to their common use cases.

One example of such a framework is Dask, a popular Python library used to distribute and parallelize SQL-like table operations, machine learning workflows or even arbitrary Python functions with little programming effort. Dask uses a centralized architecture with a single central component – the server – which schedules tasks and manages computing nodes in a cluster. It uses a work-stealing scheduler which has been tuned extensively to support various task graphs. Yet it is unclear whether additional effort should be directed into improving the scheduler or if there is another bottleneck which should be prioritized.

We have designed a series of experiments that study the runtime overhead of Dask and the effect of the used scheduling algorithm on its performance. We also develop a backwards compatible replacement of the Dask server which aims to minimize its runtime overhead and thus scale to larger clusters. Our paper makes the following *contributions*:

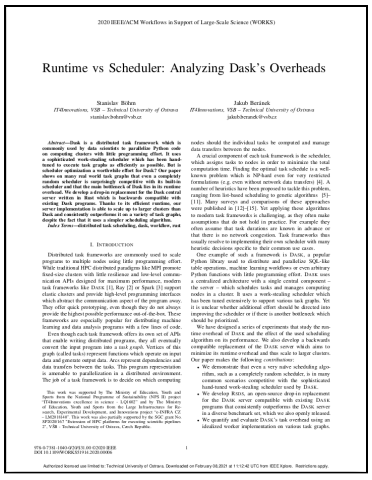
- We demonstrate that even a very naïve scheduling algorithm, such as a completely random scheduler, is in many common scenarios competitive with the sophisticated hand-tuned work-stealing scheduler used by Dask.
- We develop Rsds, an open-source drop-in replacement for the Dask server compatible with existing Dask programs that consistently outperforms the Dask server in a diverse benchmark set, which we also openly released.
- We quantify and evaluate Dask's task overhead using an idealized worker implementation on various task graphs.

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPS II) project "IT4Innovations excellence in science - LQ1602" and by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development, and Innovations project "e-INFRA CZ – LM2018140". This work was also partially supported by the SGC grant No. SP2020/167 "Extension of HPC platforms for executing scientific pipelines 2", VŠB - Technical University of Ostrava, Czech Republic.

978-0-7381-1040-0/20/$31.00 ©2020 IEEE
DOI 10.1109/WORKS51914.2020.00006

Runtime vs Scheduler: Analyzing Dask's Overheads

Stanislav Böhm, Jakub Beránek
(IEEE/ACM Workflows in Support of Large-Scale Science 2020)



2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)

Runtime vs Scheduler: Analyzing Dask's Overheads

Stanislav Böhm
IT4Innovations, VSB – Technical University of Ostrava
stanislav.bohm@vsb.cz

Jakub Beránek
IT4Innovations, VSB – Technical University of Ostrava
jakub.beranek@vsb.cz

Abstract—Dask is a distributed task framework which is commonly used to scale programs to multiple nodes and run on computing clusters with little programming effort. It uses a sophisticated work-stealing scheduler which has been hand-tuned to execute task graphs as efficiently as possible. But is scheduler optimization a worthwhile effort for Dask? Our paper shows that a very naive scheduling algorithm, such as a completely random scheduler is surprisingly competitive with its built-in scheduler and that the main bottleneck of Dask lies in its runtime overhead. We develop a drop-in replacement for the Dask central server written in Rust which is backwards compatible with Dask's Python ecosystem and show that our implementation of server implementation is able to scale up to larger clusters than Dask and consistently outperforms it on a variety of task graphs, despite the fact that it uses a simpler scheduling algorithm.

***Index Terms*—distributed task scheduling, dask, workflow, rust**

I. Introduction

Distributed task frameworks are commonly used to scale programs to multiple nodes using little programming effort. While traditional HPC distributed paradigms like MPI promote fixed-size clusters with little resilience and low-level communication APIs designed for maximum performance, modern frameworks like Dask [1], Ray [2], or Spark [3] target more elastic clusters and provide high-level programming interfaces which abstract the communication aspect of the program away. They offer quick prototyping, even though they do not always provide the highest possible performance out-of-the-box. These frameworks are especially popular for distributing machine learning and data analysis programs with a few lines of code.

Even though each task framework offers its own set of APIs that enable writing distributed programs, they all eventually convert the input program into a *task graph*. Vertices of this graph (called tasks) represent functions which operate on input data and generate output data. Arcs represent dependencies and data transfers between the tasks. This program representation is amenable to parallelization in a distributed environment. The job of a task framework is to decide on which computing nodes should the individual tasks be computed and manage data transfers between the nodes.

A crucial component of each task framework is the scheduler, which assigns tasks to nodes in order to minimize the total computation time. Finding the optimal task schedule is a well-known problem which is NP-hard even in very simplified formulations (e.g. even without network data transfers) [4]. A number of heuristics have been proposed to tackle this problem, ranging from list-based scheduling to genetic algorithms [5]–[11]. Many surveys and comparisons of these approaches were published in [12]–[15]. Yet applying these algorithms to modern task frameworks is challenging, as they often make assumptions that do not hold in practice. For example they often assume that task durations are known in advance or that there is no network congestion. Task frameworks thus usually resolve to implementing their own scheduler with many heuristic decisions specific to their common use cases.

One example of such a framework is Dask, a popular Python library used to distribute and parallelize SQL-like table operations, machine learning workflows or even arbitrary Python functions with little programming effort. Dask uses a centralized architecture with a single central component – a server – which manages the execution of tasks submitted by users in a cluster. It uses a work-stealing scheduler which has been tuned extensively to support various task graphs. Yet it is unclear whether additional effort should be directed into improving the scheduler or if there is another bottleneck which should be prioritized.

We have designed a series of experiments that study the runtime overhead of Dask and the effect of the used scheduling algorithm on its performance. We also develop a backwards compatible replacement of the Dask server which aims to minimize its runtime overhead and thus scale to larger clusters. Our paper makes the following contributions:

- We demonstrate that even a very naive scheduling algorithm, such as a completely random scheduler, is in many common scenarios competitive with the sophisticated hand-tuned work-stealing scheduler used by Dask.
- We develop Rsds, an open-source drop-in replacement for the Dask server compatible with existing Dask programs that consistently outperforms the Dask server by a factor of 3 despite using a simpler scheduler.
- We quantify and evaluate Dask's task overhead using an idealized worker implementation on various task graphs.

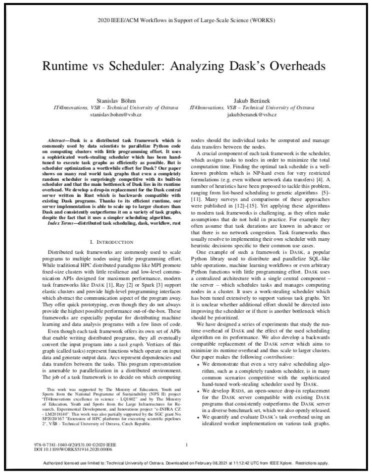
This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPS II) project "IT4Innovations excellence in science – LQ1602" and by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development and Innovations project "IT4Innovations National Supercomputing Center – LM2015070". This work was also partially supported by the SGC grant No. SP2020/167 "Automation of HPC platforms for executing scientific pipelines", VSB – Technical University of Ostrava, Czech Republic.

978-1-7281-1340-3/20/$31.00 ©2020 IEEE
DOI 10.1109/WORKS51914.2020.00006

Runtime vs Scheduler: Analyzing Dask's Overheads

Stanislav Böhm, Jakub Beránek
(IEEE/ACM Workflows in Support of Large-Scale Science 2020)

- Test schedulers "in the wild"


2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)

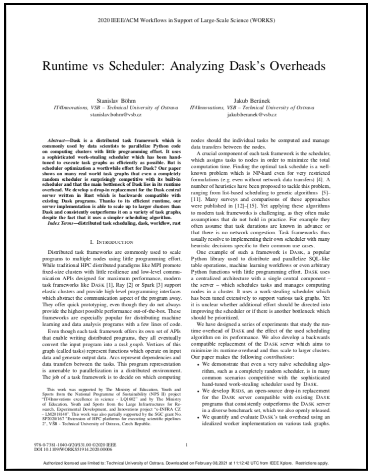
Runtime vs Scheduler: Analyzing Dask's Overheads

Runtime vs Scheduler: Analyzing Dask's Overheads

Stanislav Böhm, Jakub Beránek

(IEEE/ACM Workflows in Support of Large-Scale Science 2020)

- Test schedulers "in the wild"
- Analyze Dask's performance on HPC


2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)

Runtime vs Scheduler: Analyzing Dask's Overheads

Stanislav Böhm
IT4Innovations, VSB – Technical University of Ostrava
stanislav.bohm@vsb.cz

Jakub Beránek
IT4Innovations, VSB – Technical University of Ostrava
jakub.beranek@vsb.cz

***Abstract*—Dask is a distributed task framework which is commonly used by data scientists to parallelize Python code on computing clusters with little programming effort. It uses a sophisticated work-stealing scheduler which has been hand-tuned to execute task graphs as efficiently as possible. But is scheduler optimization a worthwhile effort for Dask? Our paper shows on many real world task graphs that even a completely random scheduler is surprisingly competitive with its built-in scheduler and that the main bottleneck of Dask lies in its runtime overhead. We develop a drop-in replacement for the Dask central server written in Rust which is backwards compatible with existing Dask programs. Thanks to its efficient runtime, our server implementation is able to scale up to larger clusters than Dask and consistently outperforms it on a variety of task graphs, despite the fact that it uses a simpler scheduling algorithm.**

***Index Terms*—distributed task scheduling, dask, workflow, rust**

I. INTRODUCTION

Distributed task frameworks are commonly used to scale programs to multiple nodes using little programming effort. While traditional HPC distributed paradigms like MPI promote fixed-size clusters with little resilience and low-level communication APIs designed for maximum performance, modern task frameworks like DASK [1], Ray [2] or Spark [3] support elastic clusters and provide high-level programming interfaces which abstract the communication aspect of the program away. They offer quick prototyping, even though they do not always provide the highest possible performance out-of-the-box. These frameworks are especially popular for distributing machine learning and data analysis programs with a few lines of code.

Even though each task framework offers its own set of APIs that enable writing distributed programs, they all eventually convert the input program into a *task graph*. Vertices of this graph (called tasks) represent functions which operate on input data and generate output data. Arcs represent dependencies and data transfers between the tasks. This program representation is amenable to parallelization in a distributed environment. The job of a task framework is to decide on which computing nodes should the individual tasks be computed and manage data transfers between the nodes.

A crucial component of each task framework is the scheduler, which assigns tasks to nodes in order to minimize the total computation time. Finding the optimal task schedule is a well-known problem which is NP-hard even for very restricted formulations (e.g. even without network data transfers) [4]. A number of heuristics have been proposed to tackle this problem, ranging from list-based scheduling to genetic algorithms [5]–[11]. Many surveys and comparisons of these approaches were published in [12]–[15]. Yet applying these algorithms to modern task frameworks is challenging, as they often make assumptions that do not hold in practice. For example they often assume that task durations are known in advance or that there is no network congestion. Task frameworks thus usually resolve to implementing their own scheduler with many heuristic decisions specific to their common use cases.

One example of such a framework is DASK, a popular Python library used to distribute and parallelize SQL-like table operations, machine learning workflows or even arbitrary Python functions with little programming effort. DASK uses a centralized architecture with a single central component – the server – which schedules tasks and manages computing nodes in a cluster. It uses a work-stealing scheduler which has been tuned extensively to support various task graphs. Yet it is unclear whether additional effort should be directed into improving the scheduler or if there is another bottleneck which should be prioritized.

We have designed a series of experiments that study the runtime overhead of DASK and the effect of the used scheduling algorithm on its performance. We also develop a backwards compatible replacement of the DASK server which aims to minimize its runtime overhead and thus scale to larger clusters. Our paper makes the following *contributions*:

- We demonstrate that even a very naïve scheduling algorithm, such as a completely random scheduler, is in many common scenarios competitive with the sophisticated hand-tuned work-stealing scheduler used by DASK.
- We develop RSDS, an open-source drop-in replacement for the DASK server compatible with existing DASK programs that consistently outperforms the DASK server in a diverse benchmark set, which we also openly released.
- We quantify and evaluate DASK's task overhead using an idealized worker implementation on various task graphs.

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPS II) project "IT4Innovations excellence in science - LQ1602" and by The Ministry of Education, Youth and Sports from the Large Infrastructures for Research, Experimental Development, and Innovations project "e-INFRA CZ – LM2018140". This work was also partially supported by the SGC grant No. SP2020/167 "Extension of HPC platforms for executing scientific pipelines 2", VŠB - Technical University of Ostrava, Czech Republic.

978-0-7381-1040-0/20/$31.00 ©2020 IEEE
DOI 10.1109/WORKS51914.2020.00006

1

Runtime vs Scheduler: Analyzing Dask's Overheads

Stanislav Böhm, Jakub Beránek
(IEEE/ACM Workflows in Support of Large-Scale Science 2020)

- Test schedulers "in the wild"
- Analyze Dask's performance on HPC
- Implement more efficient Dask server (RSDS)
  - Pluggable scheduler

Dask vs. RSDS: work-stealing scheduler

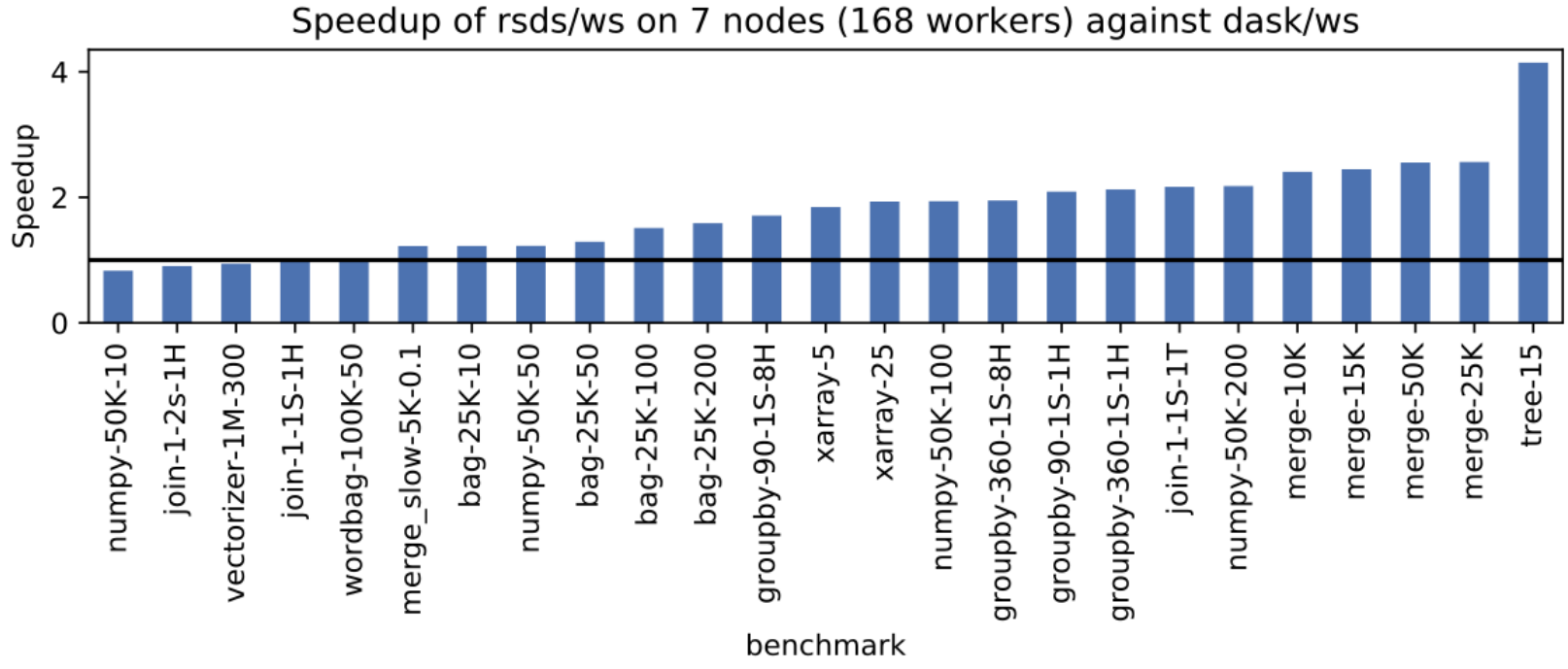


Fig. 3. Speedup of RSDS/ws scheduler; DASK/ws is baseline.

Dask vs. RSDS: scaling on Salomon

Outcome

- Runtime efficiency as important as scheduling

Outcome

- Runtime efficiency as important as scheduling
  - Even a random scheduler can be competitive!

Outcome

- Runtime efficiency as important as scheduling
  - Even a random scheduler can be competitive!
- Dask scaled poorly on HPC

Outcome

- Runtime efficiency as important as scheduling
  - Even a random scheduler can be competitive!
- Dask scaled poorly on HPC
  - Caused by inefficient runtime

Outcome

- Runtime efficiency as important as scheduling
  - Even a random scheduler can be competitive!
- Dask scaled poorly on HPC
  - Caused by inefficient runtime
  - <100 ms tasks problematic

Outcome

- Runtime efficiency as important as scheduling
  - Even a random scheduler can be competitive!
- Dask scaled poorly on HPC
  - Caused by inefficient runtime
  - <100 ms tasks problematic
- RSDS: open source alternative to Dask's server

Outcome

- Runtime efficiency as important as scheduling
  - Even a random scheduler can be competitive!
- Dask scaled poorly on HPC
  - Caused by inefficient runtime
  - <100 ms tasks problematic
- RSDS: open source alternative to Dask's server
  - Backward-compatible with existing Dask programs
  - `github.com/it4innovations/rsds`

Ergonomics and efficiency
of workflows on HPC clusters

HyperQueue

`github.com/it4innovations/hyperqueue`

Task runtime designed for HPC

HyperQueue

github.com/it4innovations/hyperqueue

Task runtime designed for HPC

HyperQueue

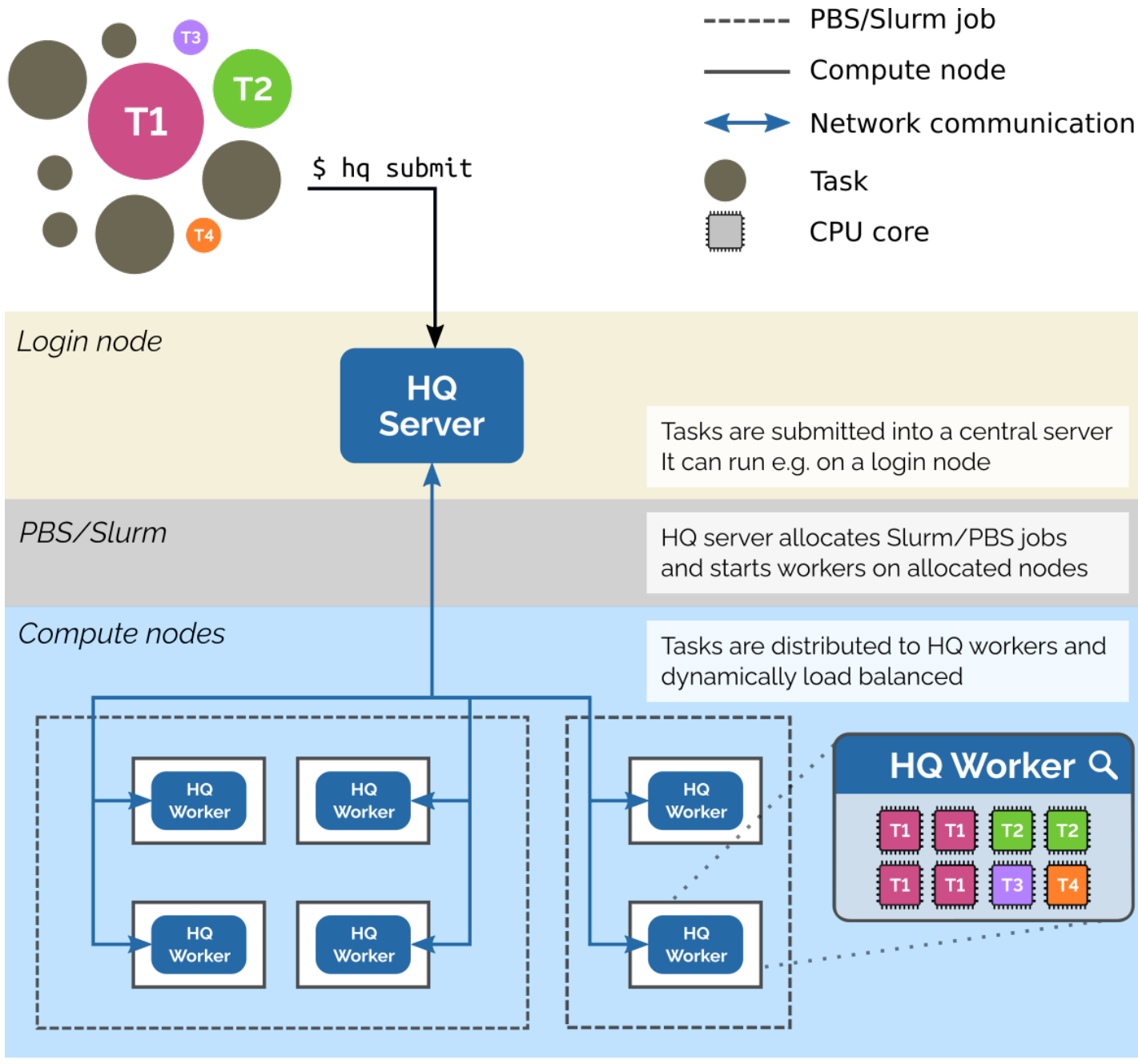
github.com/it4innovations/hyperqueue

Task runtime designed for HPC

Team effort @ IT4I

(primary contributors: Ada Böhm & me)

Architecture



Command-line interface

Start server

```
(login1) $ hq server start
```

Command-line interface

Start server

```
(login1) $ hq server start
```

Submit tasks

```
(login1) $ hq submit -- ./my-program --foo=bar
```

Command-line interface

Start server

```
(login1) $ hq server start
```

Submit tasks

```
(login1) $ hq submit -- ./my-program --foo=bar
```

Provide computational resources

```
(login1) $ hq alloc add pbs --time-limit 1h -- -qqexp
```

Command-line interface

Start server

```
(login1) $ hq server start
```

Submit tasks

```
(login1) $ hq submit -- ./my-program --foo=bar
```

Provide computational resources

```
(login1) $ hq alloc add pbs --time-limit 1h -- -qqexp
```

or

```
(node1)  $ hq worker start
```

Python API

```
def preprocess(path):
    # preprocess data

path = "/tmp/foo"

job = Job()
t1 = job.function(preprocess, args=(path, ))
job.program(["simulate", "--path", path], deps=[t1])
client.submit(job)
```

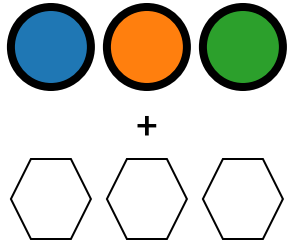
Challenge: PBS/Slurm

Disentangle computation description from resources

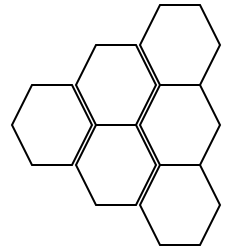
Challenge: PBS/Slurm

Disentangle computation description from resources

PBS only



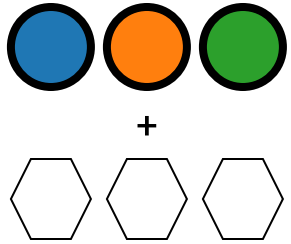
→ PBS →



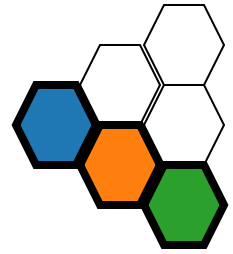
Challenge: PBS/Slurm

Disentangle computation description from resources

PBS only



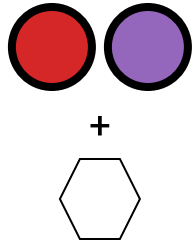
→ PBS →



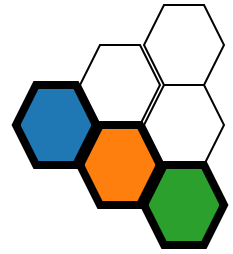
Challenge: PBS/Slurm

Disentangle computation description from resources

PBS only



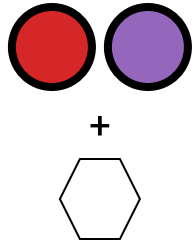
→ PBS →



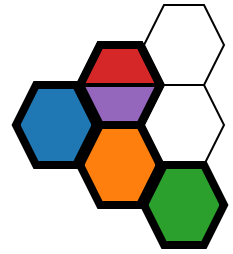
Challenge: PBS/Slurm

Disentangle computation description from resources

PBS only



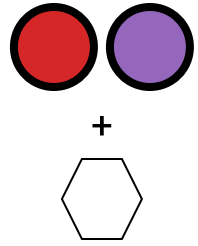
→ PBS →



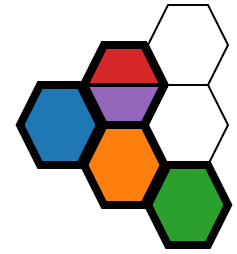
Challenge: PBS/Slurm

Disentangle computation description from resources

PBS only

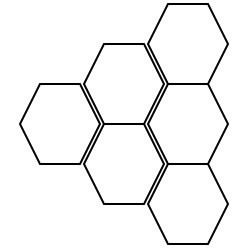


→ PBS →



PBS + HQ

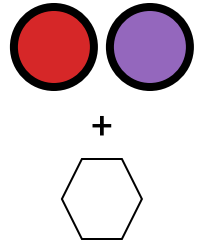
→ HyperQueue → PBS →



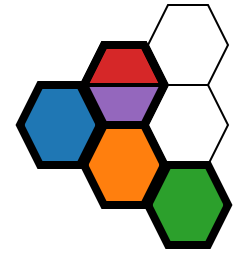
Challenge: PBS/Slurm

Disentangle computation description from resources

PBS only

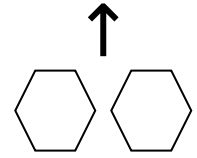


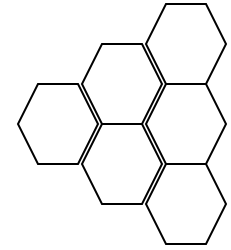
→ PBS →



PBS + HQ

→ HyperQueue → PBS →

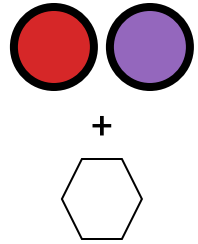




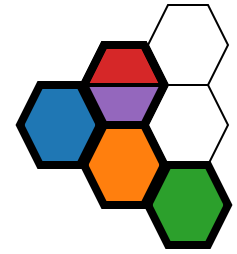
Challenge: PBS/Slurm

Disentangle computation description from resources

PBS only

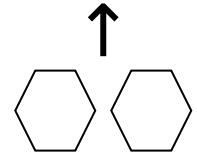


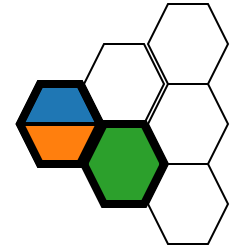
→ PBS →



PBS + HQ

→ HyperQueue → PBS →

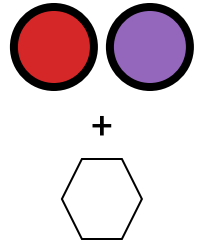




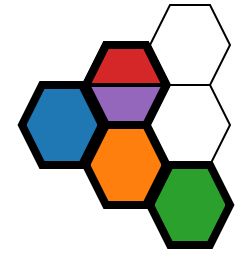
Challenge: PBS/Slurm

Disentangle computation description from resources

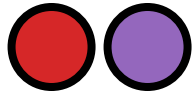
PBS only



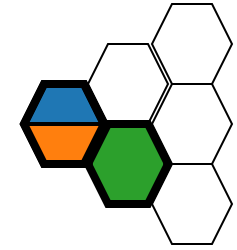
→ PBS →



PBS + HQ



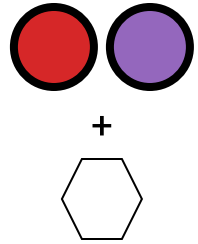
→ HyperQueue → PBS →



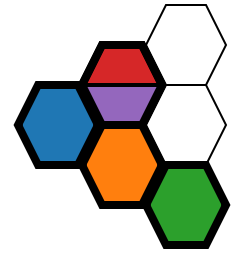
Challenge: PBS/Slurm

Disentangle computation description from resources

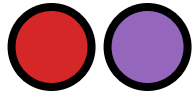
PBS only



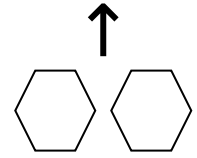
→ PBS →

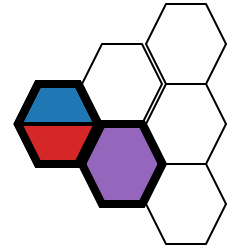


PBS + HQ



→ HyperQueue → PBS →

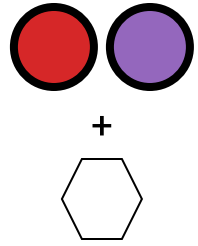




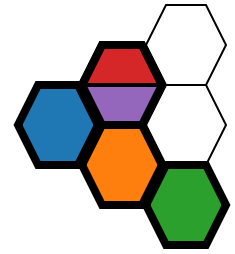
Challenge: PBS/Slurm

Disentangle computation description from resources

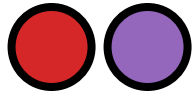
PBS only



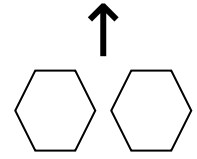
→ PBS →

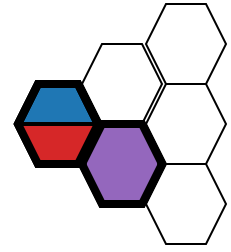


PBS + HQ



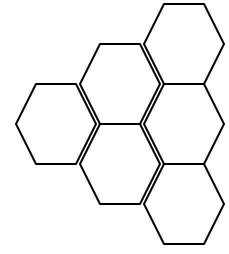
→ HyperQueue → PBS →





PBS + HQ
+autoalloc

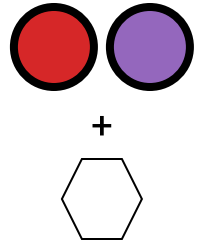
→ HyperQueue → PBS →



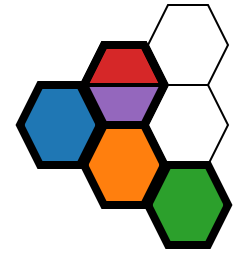
Challenge: PBS/Slurm

Disentangle computation description from resources

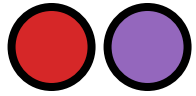
PBS only



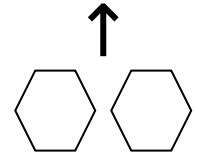
→ PBS →

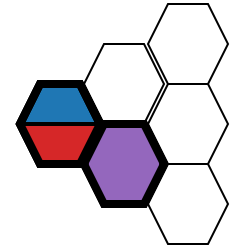


PBS + HQ



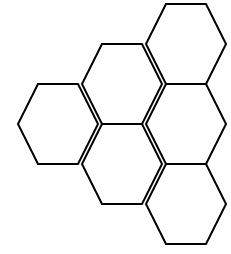
→ HyperQueue → PBS →





PBS + HQ
+autoalloc

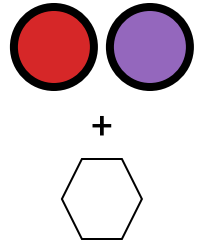
→ HyperQueue → PBS →



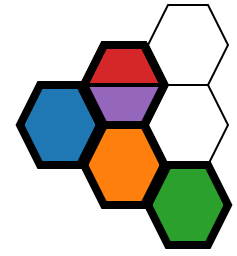
Challenge: PBS/Slurm

Disentangle computation description from resources

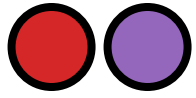
PBS only



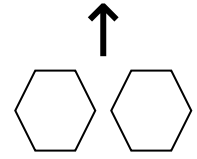
→ PBS →

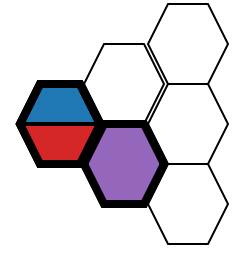


PBS + HQ



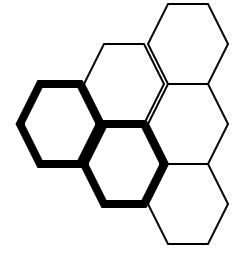
→ HyperQueue → PBS →





PBS + HQ
+autoalloc

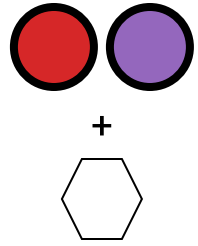
→ HyperQueue → PBS →



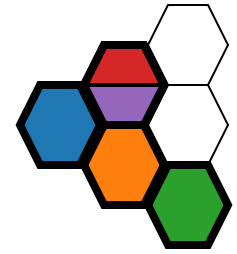
Challenge: PBS/Slurm

Disentangle computation description from resources

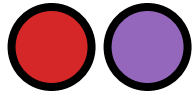
PBS only



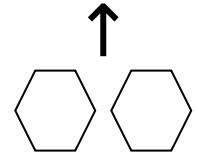
→ PBS →

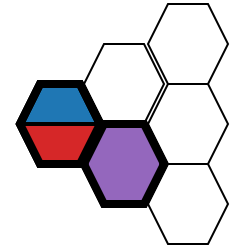


PBS + HQ



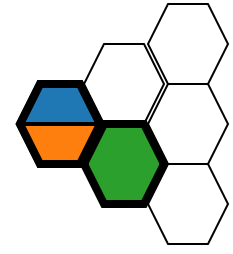
→ HyperQueue → PBS →





PBS + HQ
+autoalloc

→ HyperQueue → PBS →



Automatic allocation

- Automatic submit of PBS/Slurm jobs

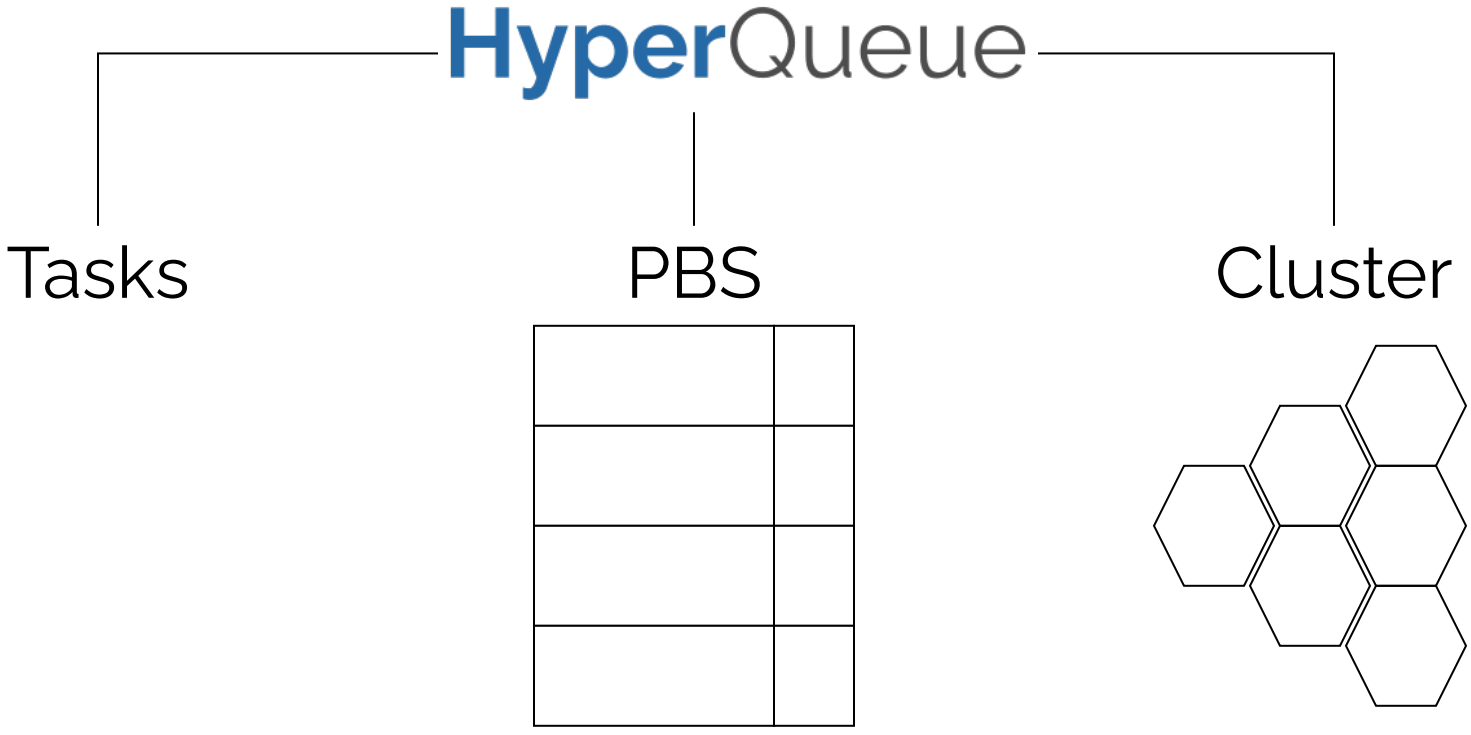
Automatic allocation

- Automatic submit of PBS/Slurm jobs
- Based on computational demand (submitted tasks)

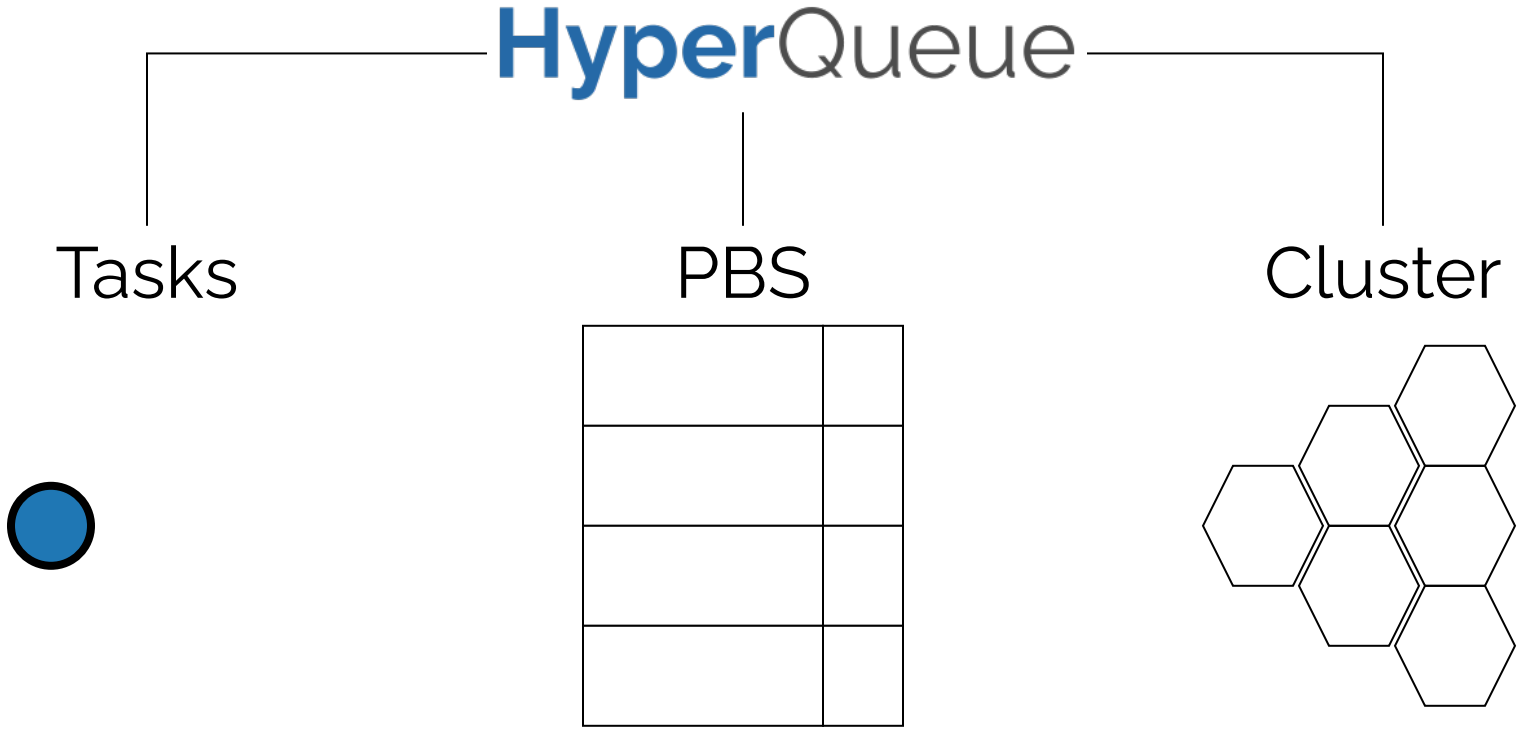
Automatic allocation

- Automatic submit of PBS/Slurm jobs
- Based on computational demand (submitted tasks)
- To be further improved and analysed

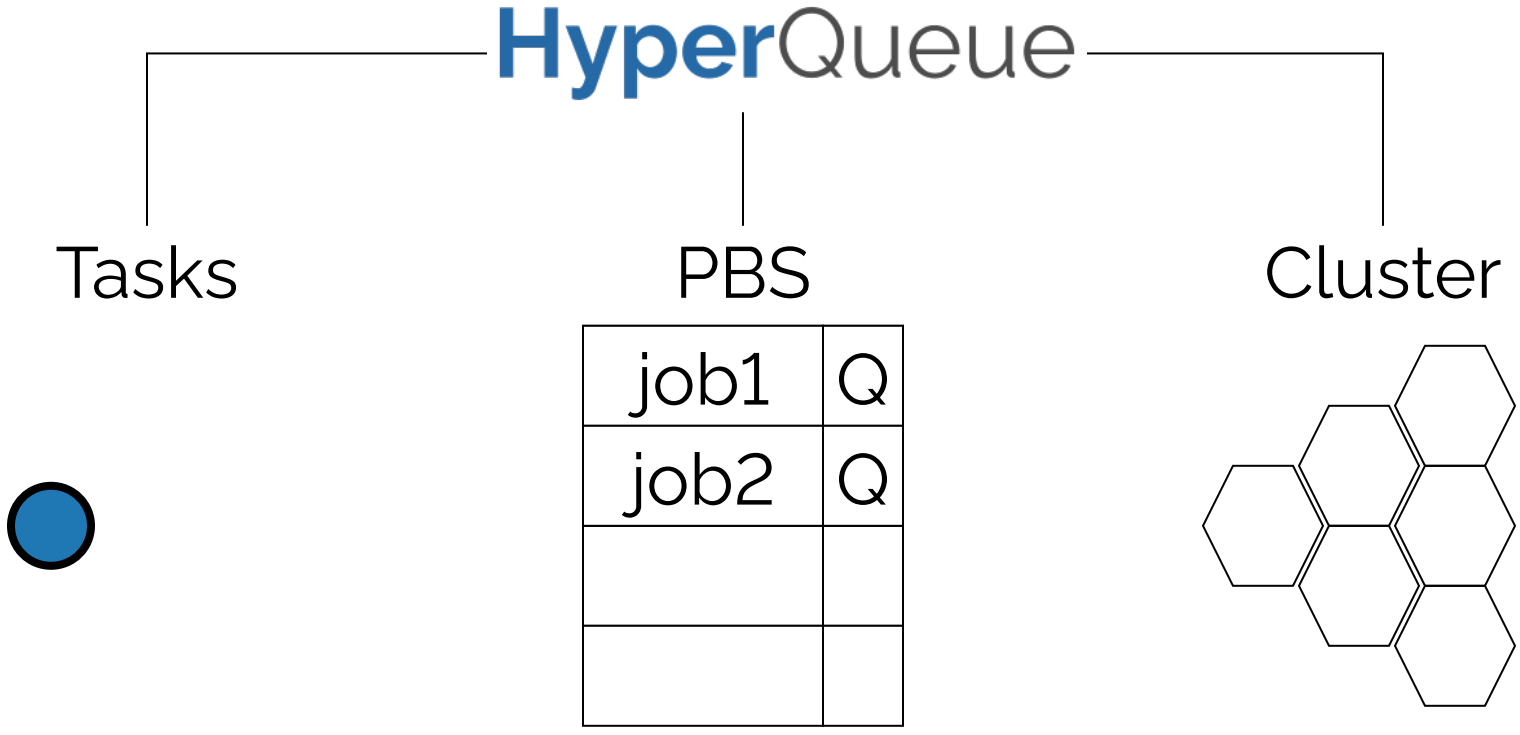
Automatic allocation



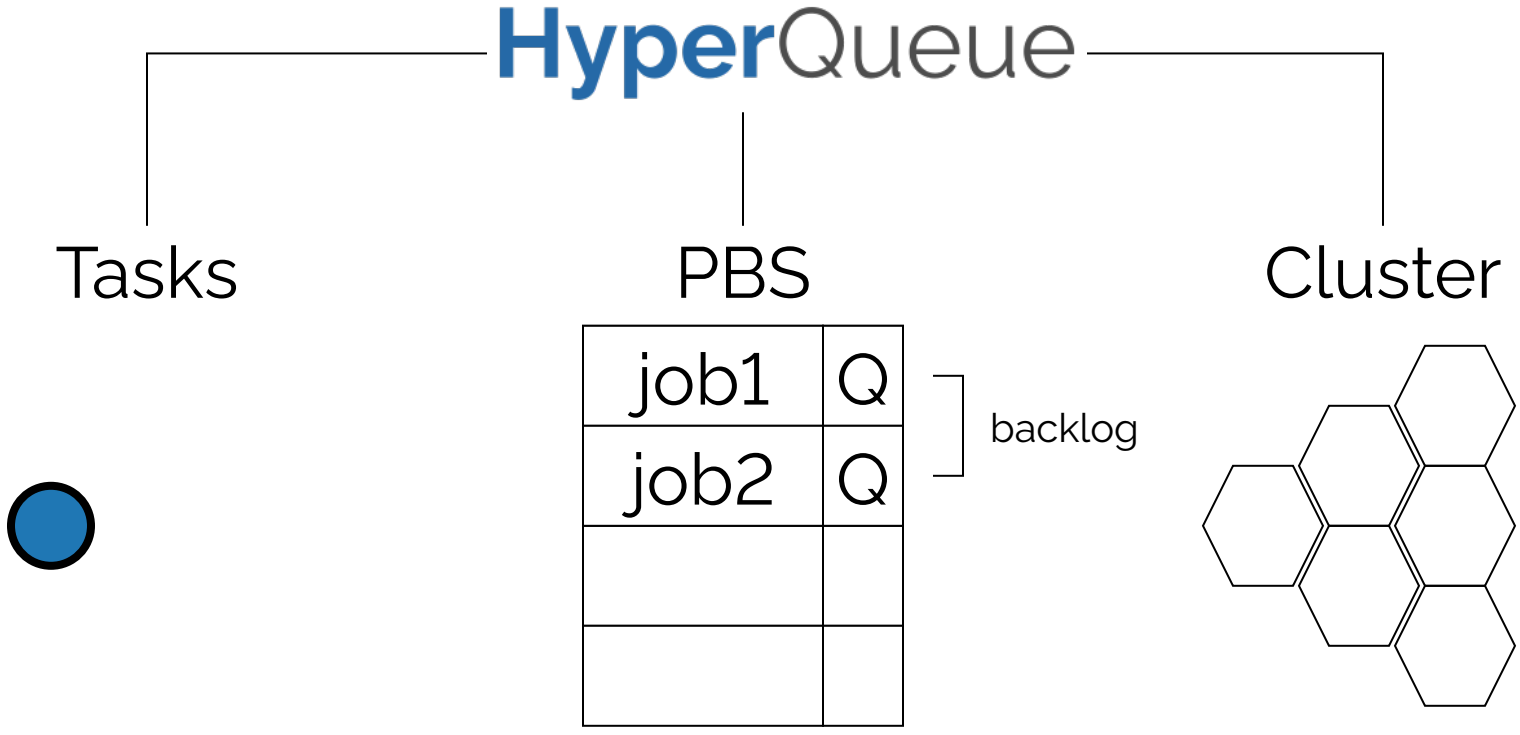
Automatic allocation



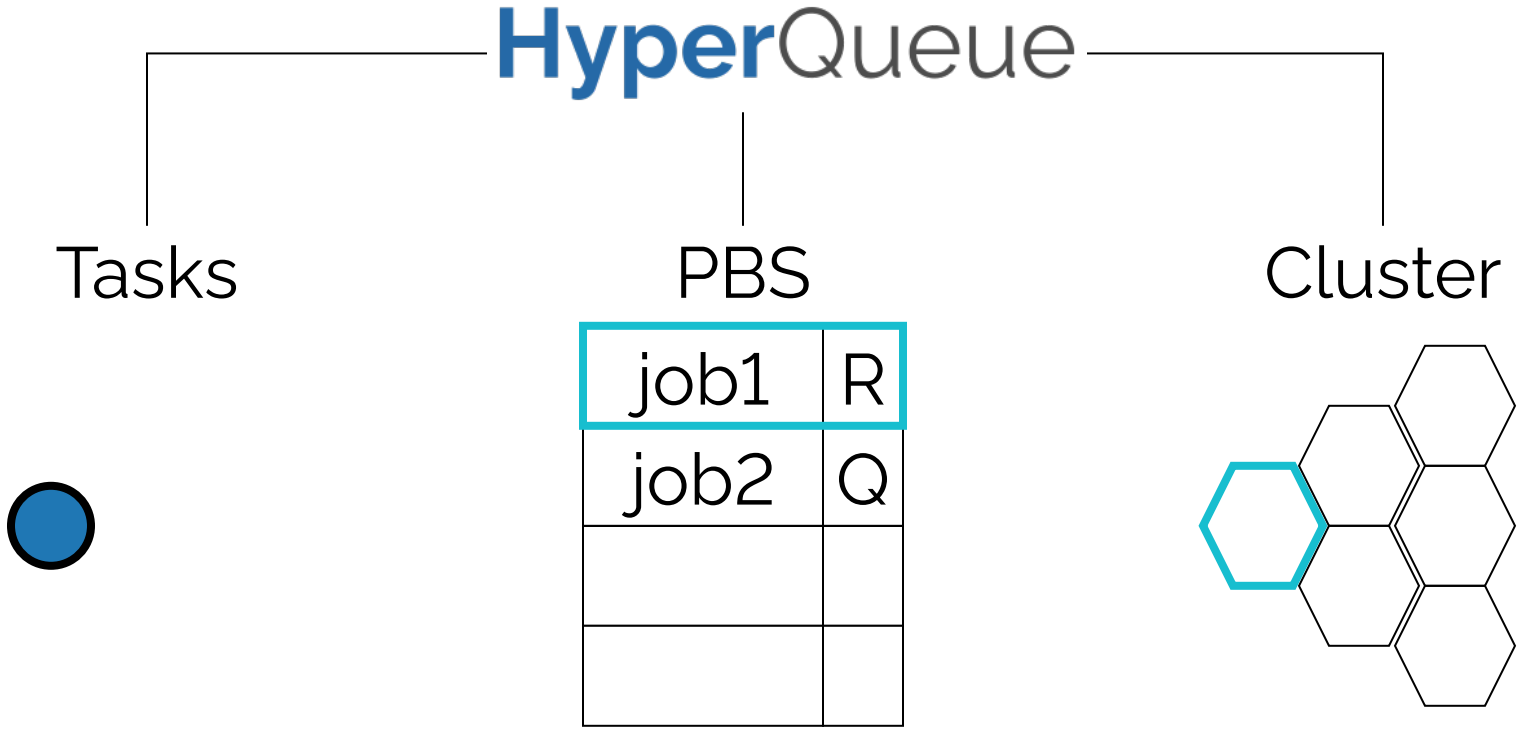
Automatic allocation



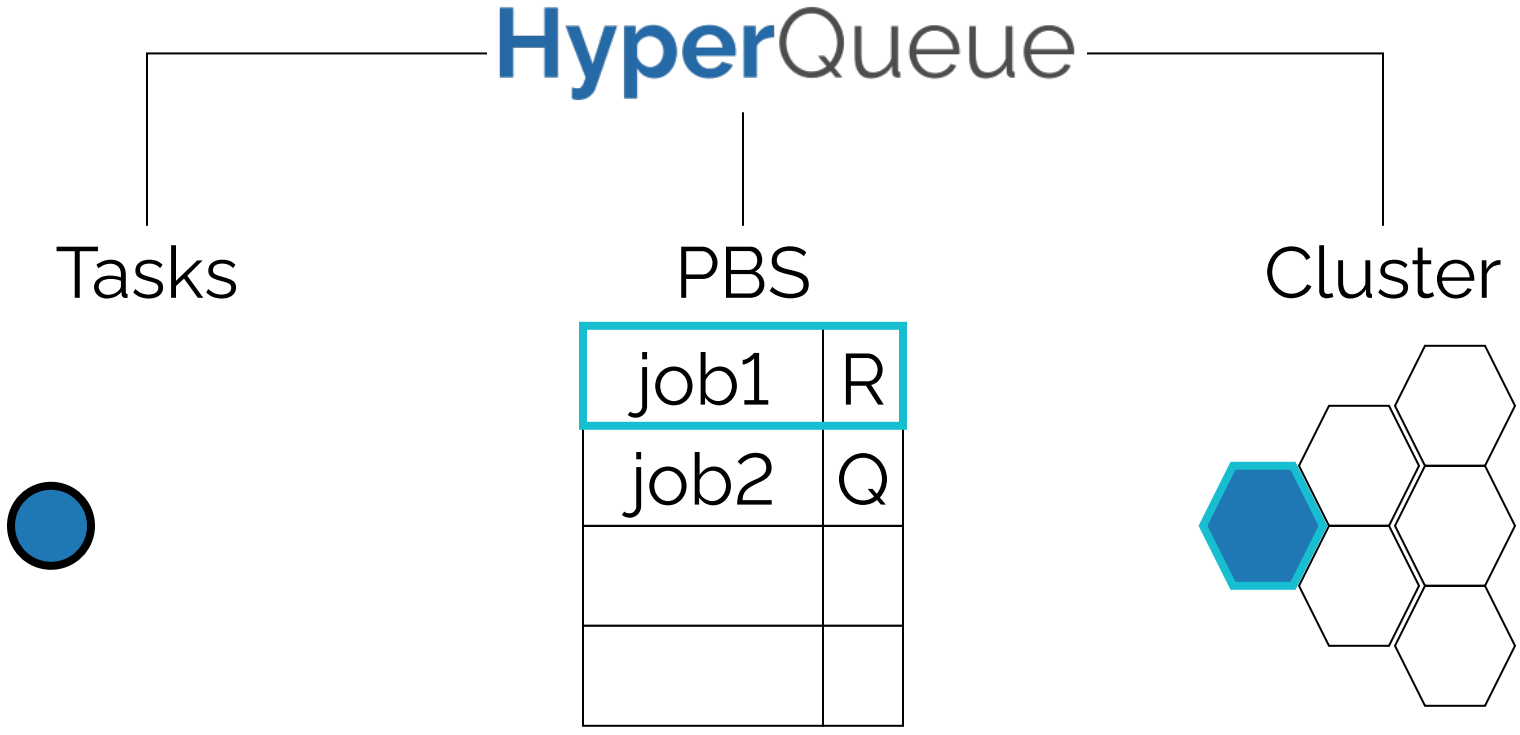
Automatic allocation



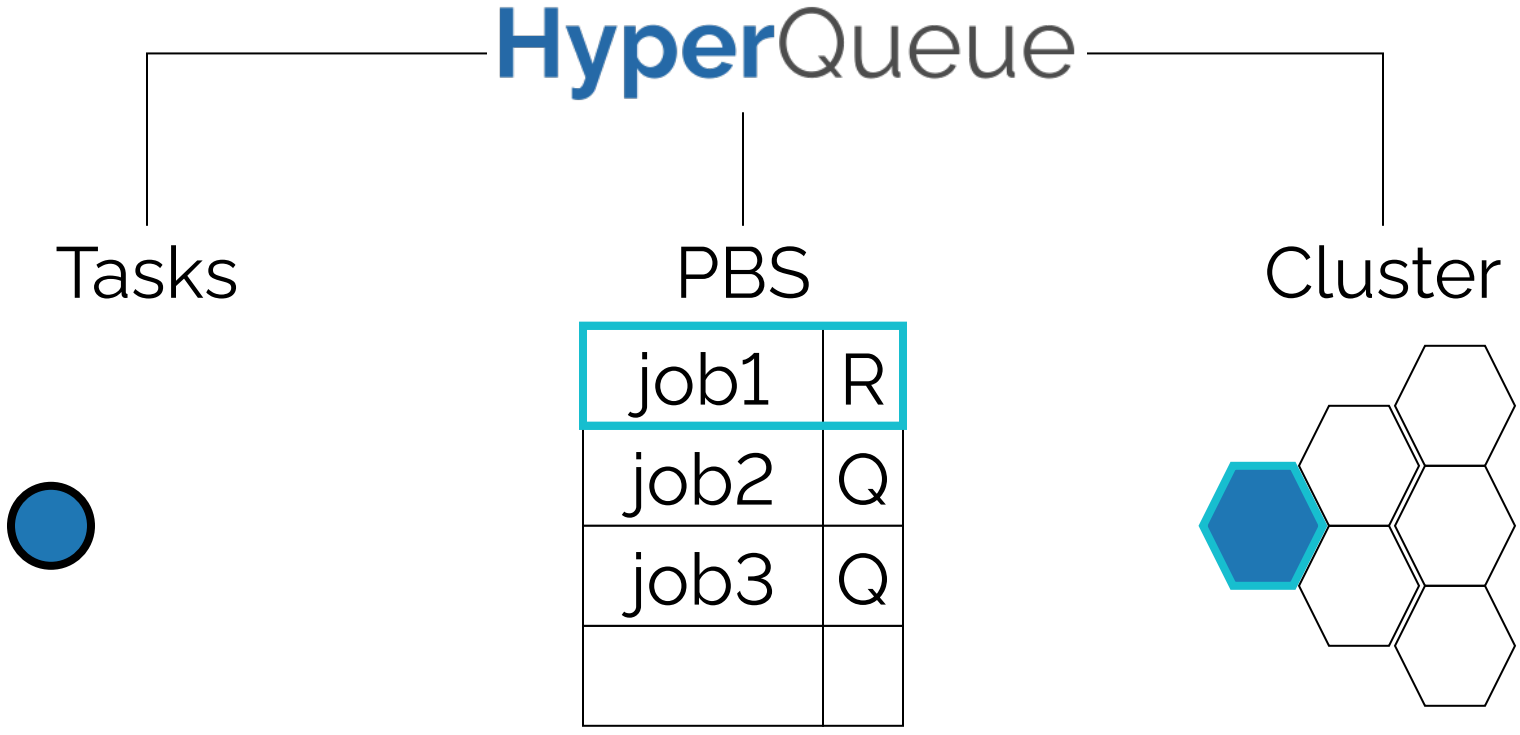
Automatic allocation



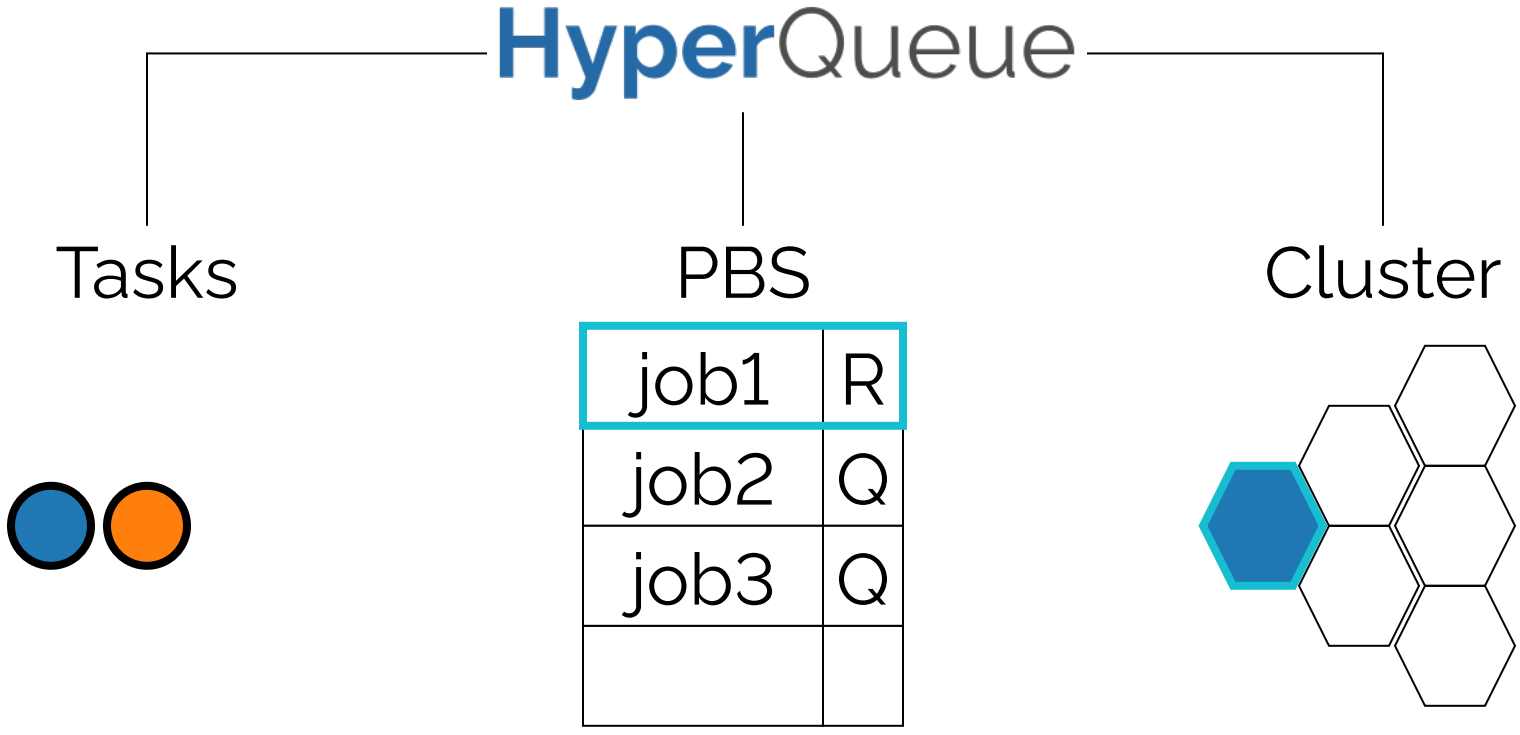
Automatic allocation



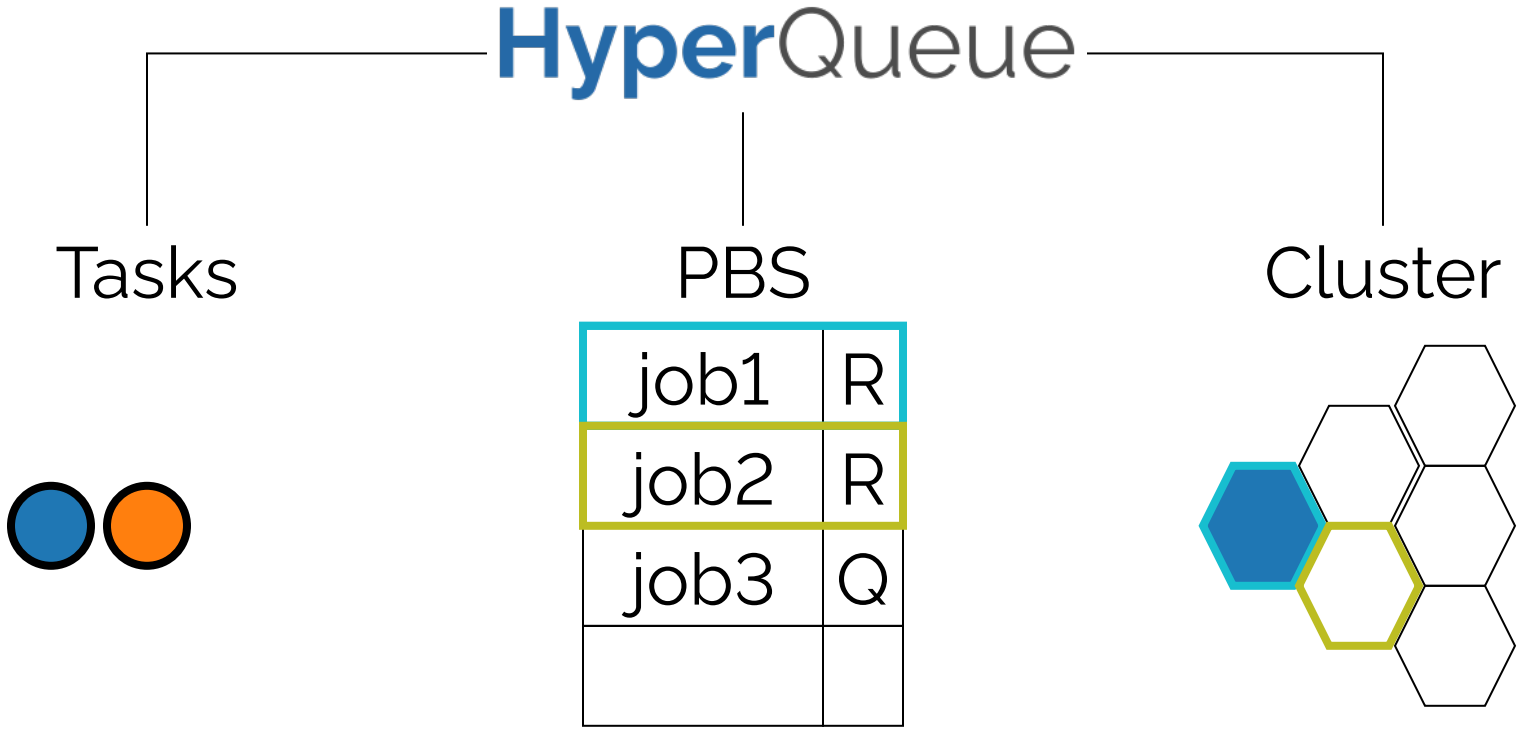
Automatic allocation



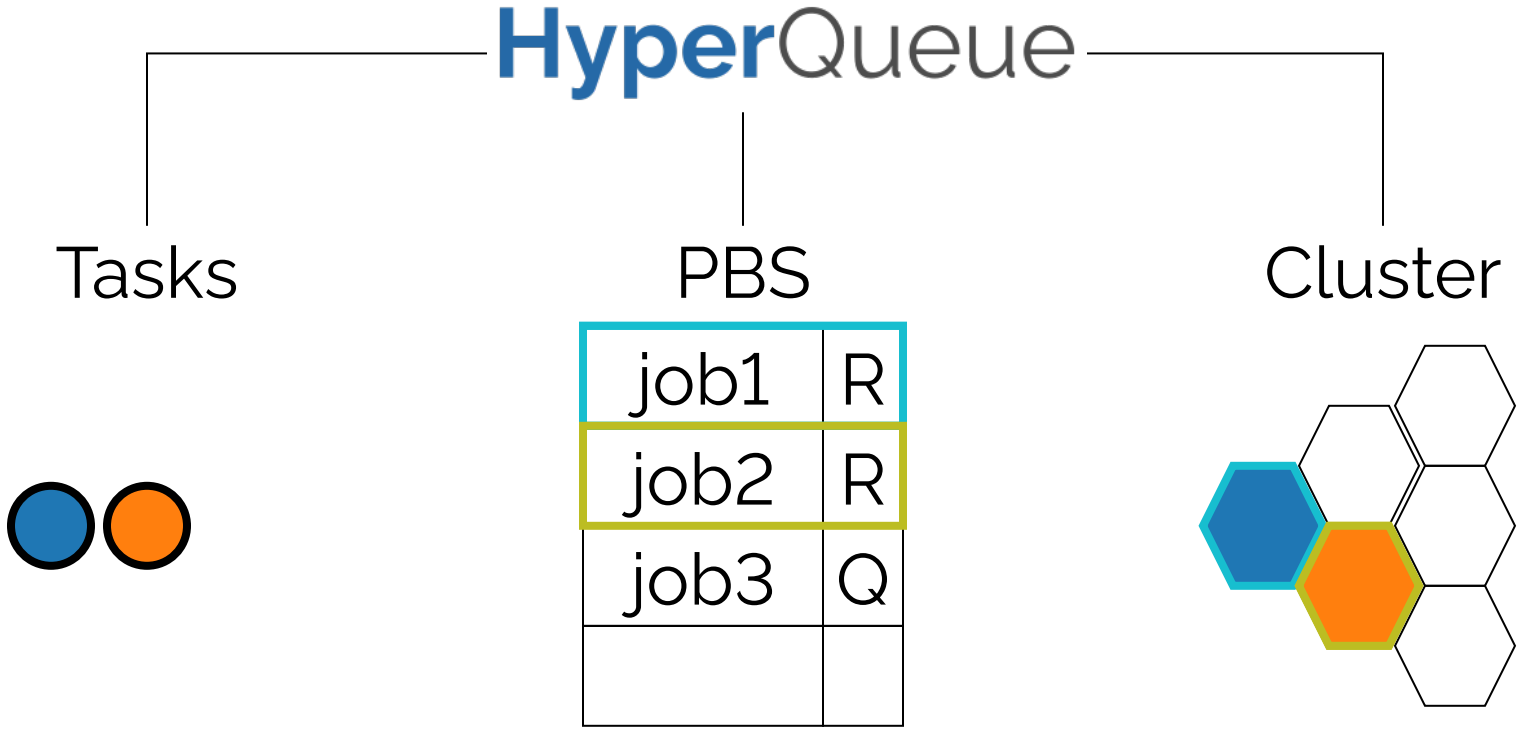
Automatic allocation



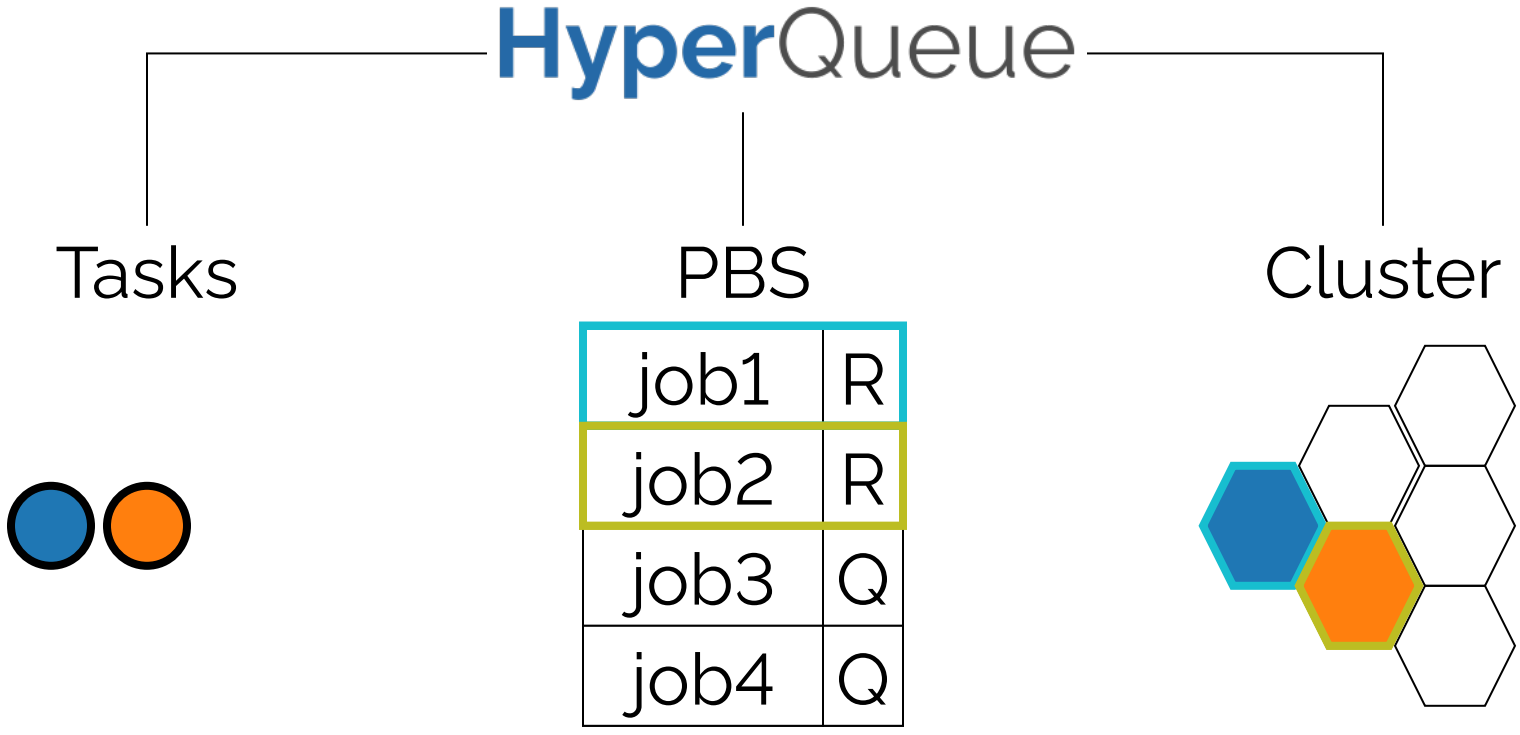
Automatic allocation



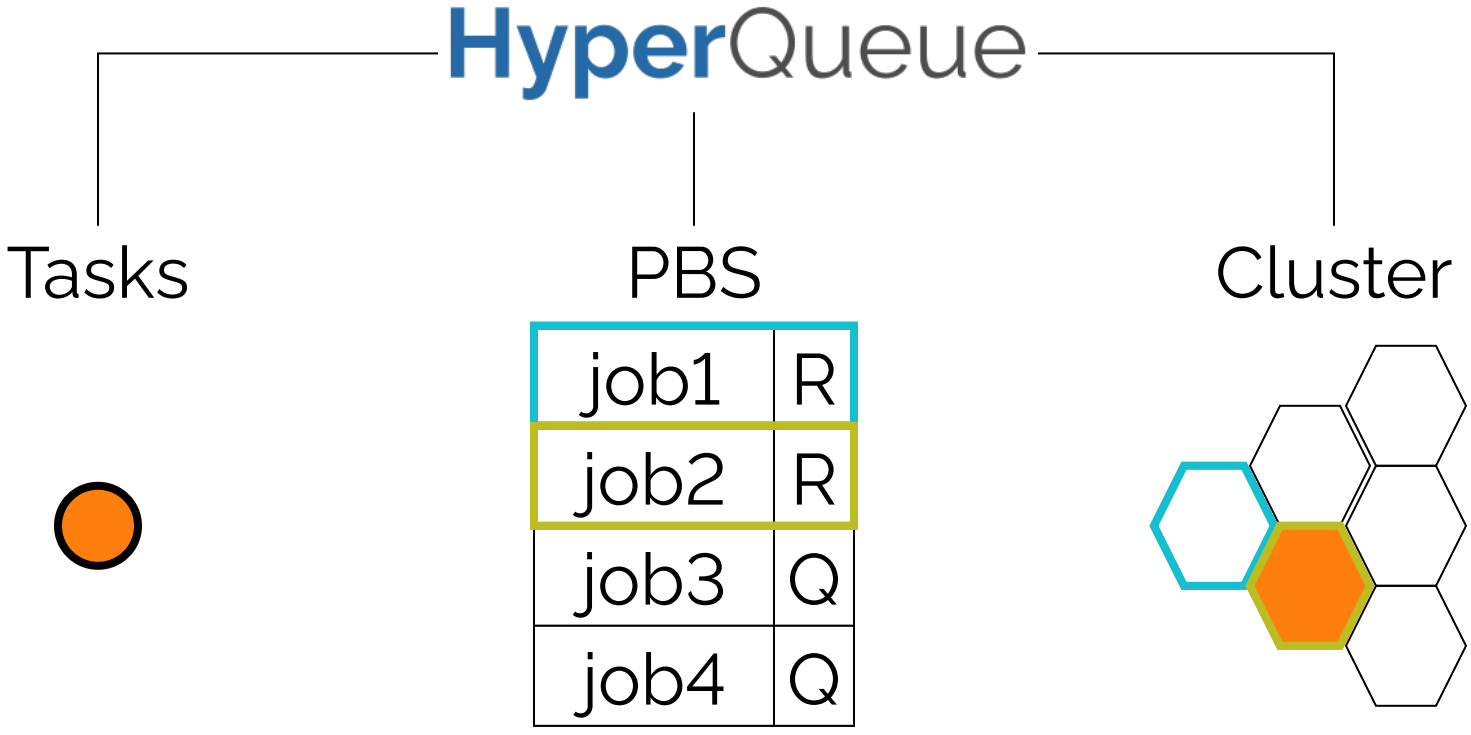
Automatic allocation



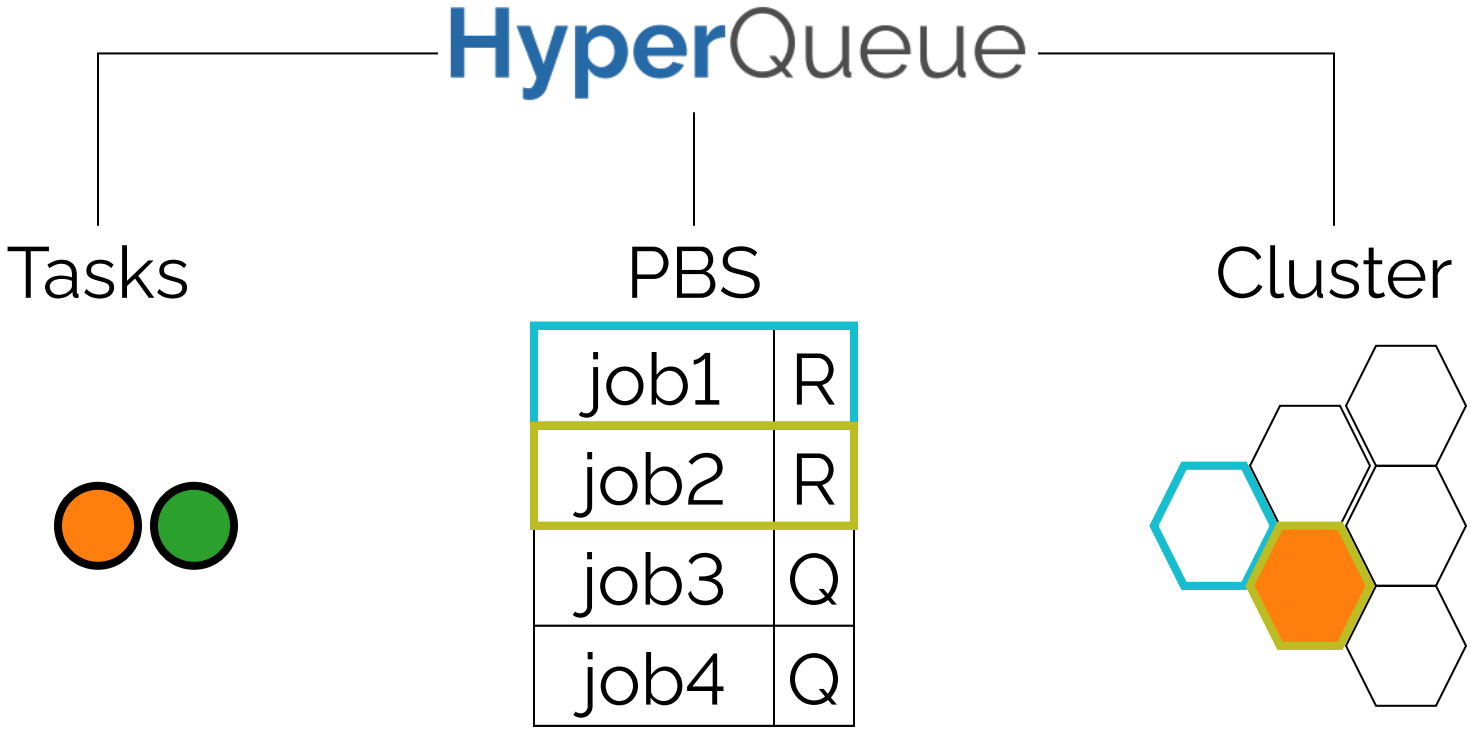
Automatic allocation



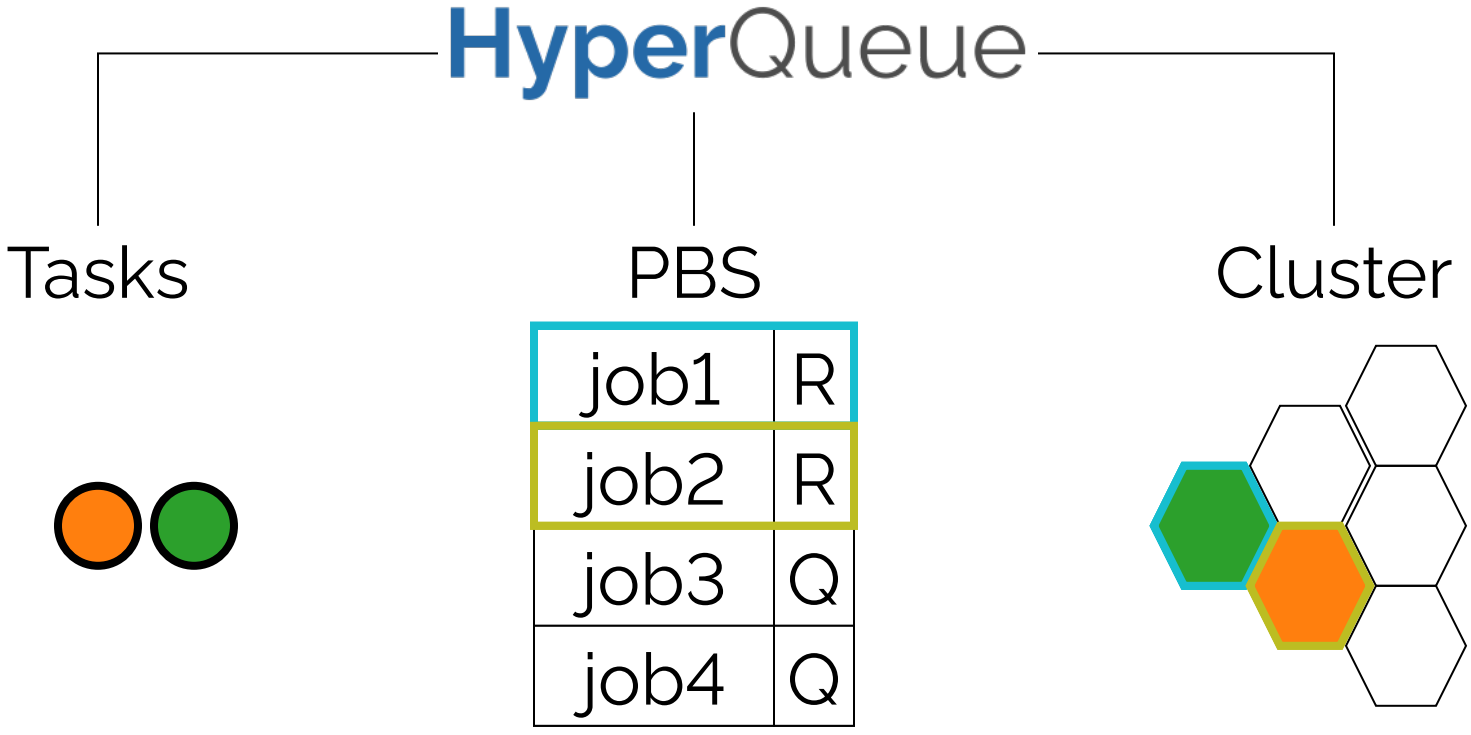
Automatic allocation



Automatic allocation



Automatic allocation



Challenge: fine-grained resources

- Arbitrary resources per task and worker

Challenge: fine-grained resources

- Arbitrary resources per task and worker
- Resource-aware scheduler

Challenge: fine-grained resources

- Arbitrary resources per task and worker
- Resource-aware scheduler
- NUMA support

Challenge: fine-grained resources

- Arbitrary resources per task and worker
- Resource-aware scheduler
- NUMA support

```
$ hq worker start --resource gpus=range(1-3)
```

Challenge: fine-grained resources

- Arbitrary resources per task and worker
- Resource-aware scheduler
- NUMA support

```
$ hq worker start --resource gpus=range(1-3)
```

```
$ hq submit --cpus=4 --resources gpus=2 ...
```

Challenge: multi-node tasks

- Initial support for multi-node tasks

Challenge: multi-node tasks

- Initial support for multi-node tasks

```
$ hq submit --nodes=4 ...
```

Challenge: efficiency

- Low overhead per task (~0.1ms)

Challenge: efficiency

- Low overhead per task (~0.1ms)
- I/O streaming: `stdout/stderr` over network

HyperQueue in the wild

- H2020 EU projects
  - 

Complex workflows (GROMACS + LiGen)
  - 

Heterogeneous nodes (traffic simulator)
  - 

Multi-node tasks

HyperQueue in the wild

- H2020 EU projects
  - LIGATE — Complex workflows (GROMACS + LiGen)
  - EVEREST — Heterogeneous nodes (traffic simulator)
  - ACROSS — Multi-node tasks
- HPC centers
  - VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER
  - CINECA
  - LUMI

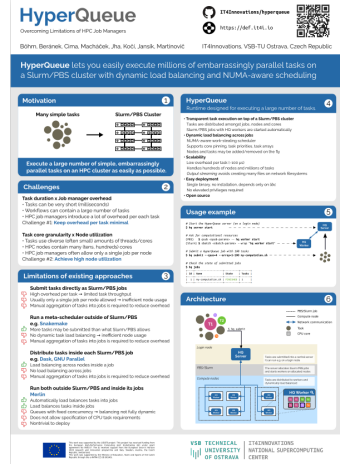
HyperQueue in the wild

- H2020 EU projects
  - LIGATE — Complex workflows (GROMACS + LiGen)
  - EVEREST — Heterogeneous nodes (traffic simulator)
  - ACROSS — Multi-node tasks
- Tool integrations
  - nextflow
  - AiiDA
  - ERT
- HPC centers
  - VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER
  - CINECA
  - LUMI

HyperQueue in the wild

- H2020 EU projects
  - LIGATE — Complex workflows (GROMACS + LiGen)
  - EVEREST — Heterogeneous nodes (traffic simulator)
  - ACROSS — Multi-node tasks
- Tool integrations
  - nextflow
  - AiiDA
  - ERT
- HPC centers
  - VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER
  - CINECA
  - LUMI
- Research
  - ATLAS experiment (CERN)

    ARC-CE+HyperQueue based submission system of ATLAS jobs for the Karolina HPC
  - Supercomputing'21 poster



Recap: HyperQueue

- **Ergonomics**
  - Job manager bypass
  - Fine-grained resource requirements
  - Multi-node tasks
  - …

Recap: HyperQueue

- **Ergonomics**
  - Job manager bypass
  - Fine-grained resource requirements
  - Multi-node tasks
  - …
- **Efficiency**
  - Low overhead per task
  - Efficient scheduler
  - …

Objectives + plan

- Identify HPC workflow challenges ✓

Objectives + plan

- Identify HPC workflow challenges ✓
- Analyze efficiency

Objectives + plan

- Identify HPC workflow challenges ✓
- Analyze efficiency
  - Schedulers ✓

Objectives + plan

- Identify HPC workflow challenges ✓
- Analyze efficiency
  - Schedulers ✓
  - Dask ✓

Objectives + plan

- Identify HPC workflow challenges ✓
- Analyze efficiency
  - Schedulers ✓
  - Dask ✓
  - **HyperQueue performance study** •

Objectives + plan

- Identify HPC workflow challenges ✓
- Analyze efficiency
  - Schedulers ✓
  - Dask ✓
  - **HyperQueue performance study** •
- Analyze ergonomics

Objectives + plan

- Identify HPC workflow challenges ✓
- Analyze efficiency
  - Schedulers ✓
  - Dask ✓
  - **HyperQueue performance study** •
- Analyze ergonomics
  - HyperQueue ✓

Objectives + plan

- Identify HPC workflow challenges ✓
- Analyze efficiency
  - Schedulers ✓
  - Dask ✓
  - **HyperQueue performance study** •
- Analyze ergonomics
  - HyperQueue ✓
  - **Automatic allocation analysis** •

Objectives + plan

- Identify HPC workflow challenges ✓
- Analyze efficiency
  - Schedulers ✓
  - Dask ✓
  - **HyperQueue performance study** •
- Analyze ergonomics
  - HyperQueue ✓
  - **Automatic allocation analysis** •
- Prepare HyperQueue publication

Publications related to thesis

- **Analysis of workflow schedulers in simulated distributed environments**
  Jakub Beránek, Stanislav Böhm, Vojtěch Cima
  The Journal of Supercomputing 2022
- **Runtime vs Scheduler: Analyzing Dask's Overheads**
  Stanislav Böhm, *Jakub Beránek*
  IEEE/ACM Workflows in Support of Large-Scale Science (WORKS) 2020

Publications related to thesis

- **Analysis of workflow schedulers in simulated distributed environments**
  Jakub Beránek, Stanislav Böhm, Vojtěch Cima
  The Journal of Supercomputing 2022
- **Runtime vs Scheduler: Analyzing Dask's Overheads**
  Stanislav Böhm, *Jakub Beránek*
  IEEE/ACM Workflows in Support of Large-Scale Science (WORKS) 2020

(collaboration with ETH Zurich)

- **Network-Accelerated Non-Contiguous Memory Transfer**
  S. Di Girolamo, K. Taranov, A. Kurth, M. Schaffner, T. Schneider, *J. Beránek*, M. Besta, L. Benini, D. Roweth, T. Hoefler
  SC (International Conference for High Performance Computing, Networking, Storage and Analysis) 2019
- **A RISC-V in-Network Accelerator for Flexible High-Performance Low-Power Packet Processing**
  S. Di Girolamo, A. Kurth, A. Calotoiu, T. Benz, T. Schneider, *J. Beránek*, L. Benini, T. Hoefler
  ISCA (International Symposium on Computer architecture) 2021

Publications unrelated to thesis

(collaboration with ETH Zurich)

- **SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory System**
  M. Besta, R. Kanakagiri, G. Kwasniewski, R. Ausavarungnirun, *J. Beránek*, K. Kanellopoulos, K. Janda, Z. Vonarburg-Shmaria, L. Gianinazzi, I. Stefan, J. G. Luna, J. Golinowski, M. Copik, L. Kapp-Schwoerer, S. Di Girolamo, N. Blach, M. Konieczny, O. Mutlu, T. Hoefler
  IEEE/ACM MICRO (International Symposium on Microarchitecture) 2021
- **GraphMineSuite: Enabling High-Performance and Programmable Graph Mining Algorithms with Set Algebra**
  M. Besta, Z. Vonarburg-Shmaria, Y. Schaffner, L. Schwarz, G. Kwasniewski, L. Gianinazzi, *J. Beránek*, K. Janda, T. Holenstein, S. Leisinger, P. Tatkowski, E. Ozdemir, A. Balla, M. Copik, P. Lindenberger, M. Konieczny, O. Mutlu, T. Hoefler
  PVLDB 2021
- **Streaming Message Interface: High-Performance Distributed Memory Programming on Reconfigurable Hardware**
  T. De Matteis, J. de Fine Licht, *J. Beránek*, T. Hoefler
  SC (International Conference for High Performance Computing, Networking, Storage and Analysis) 2019

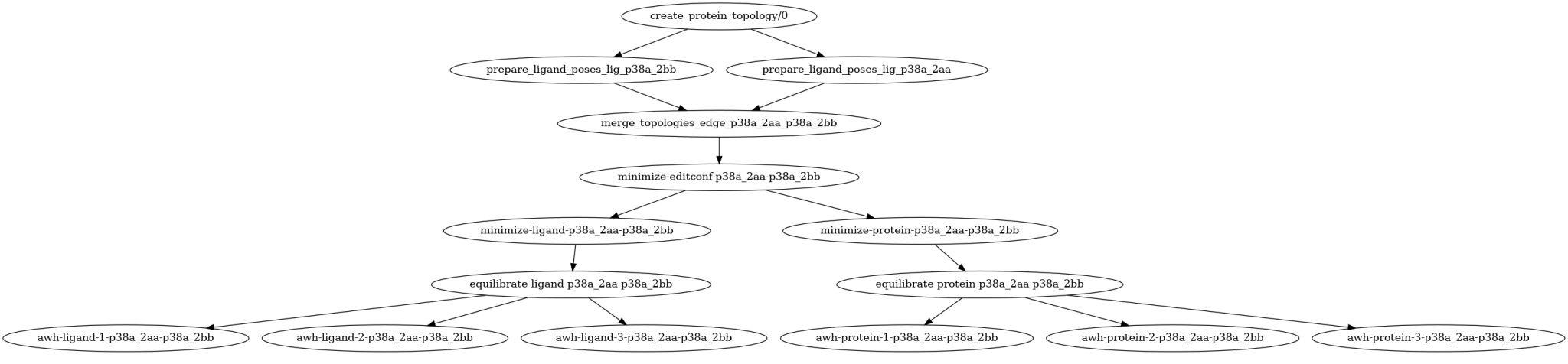
- **Haydi: Rapid Prototyping and Combinatorial Objects**
  Stanislav Böhm, *Jakub Beránek*, Martin Šurkovský
  Foundations of Information and Knowledge Systems 2018

- **Alternative Paths Reordering Using Probabilistic Time-Dependent Routing**
  M. Golasowski, *J. Beránek*, M. Šurkovský, L. Rapant, D. Szturcová, J. Martinovič, Kateřina Slaninová
  Advances in Networked-based Information Systems 2020
- **A Distributed Environment for Traffic Navigation Systems**
  J. Martinovič, M. Golasowski, K. Slaninová, *J. Beránek*, M. Šurkovský, L. Rapant, D. Szturcová, R. Cmar
  Complex, Intelligent, and Software Intensive Systems 2020

Thank you for your attention

Slides made with https://github.com/spirali/elsie

GROMACS + LiGEN LIGATE workflow



Projects and publications

- ANTAREX: distributed system for traffic simulator
- EVEREST: traffic simulator + HQ
- LIGATE: porting GROMACS + LiGEN workflow to HQ
- ETH internship: 5 publications (2 at SC'19)