Inria

Dynamic Tasks Scheduling with Multiple Priorities on Heterogeneous Computing Systems

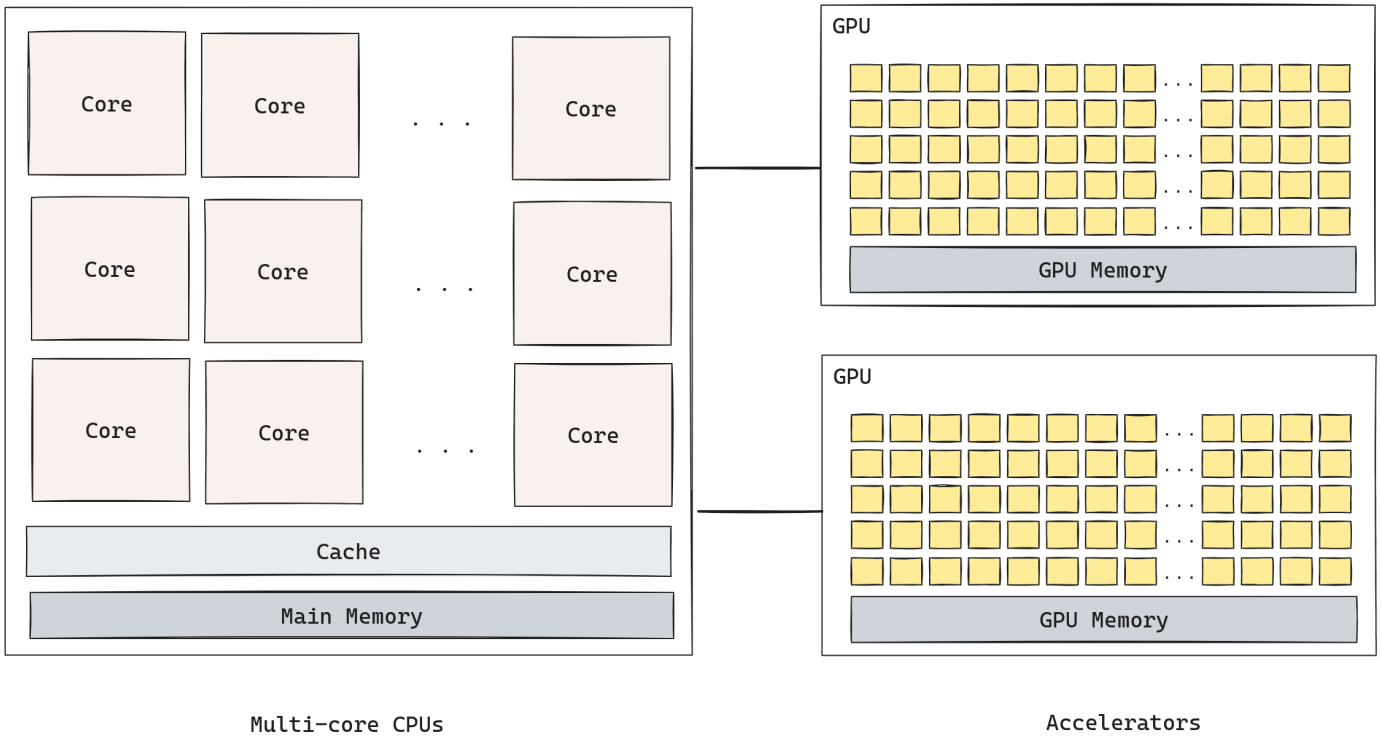
17^{th} Scheduling for Large Scale Systems Workshop, Aussois

Bérenger Bramas, Mathieu Faverge, Abdou Guermouche, Hayfa Tayeb

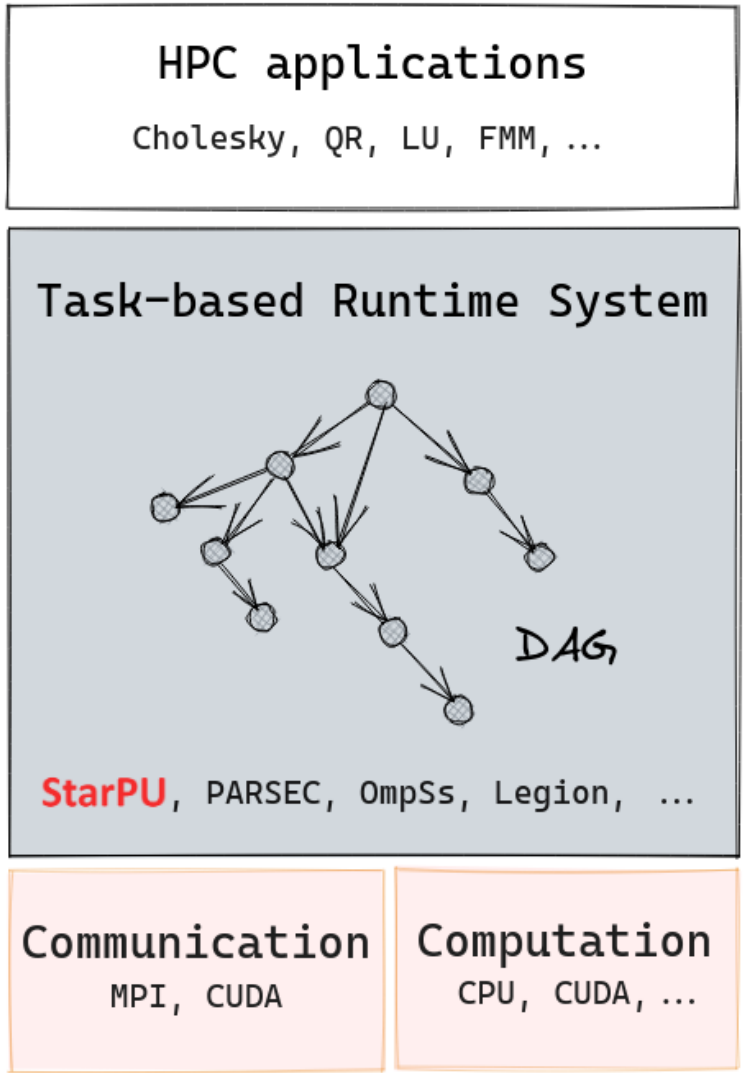
June 24-27, 2024

The challenges of heterogeneity:

- Increased complexity
- Additional programming effort
- When and how to make use of the different resources ?

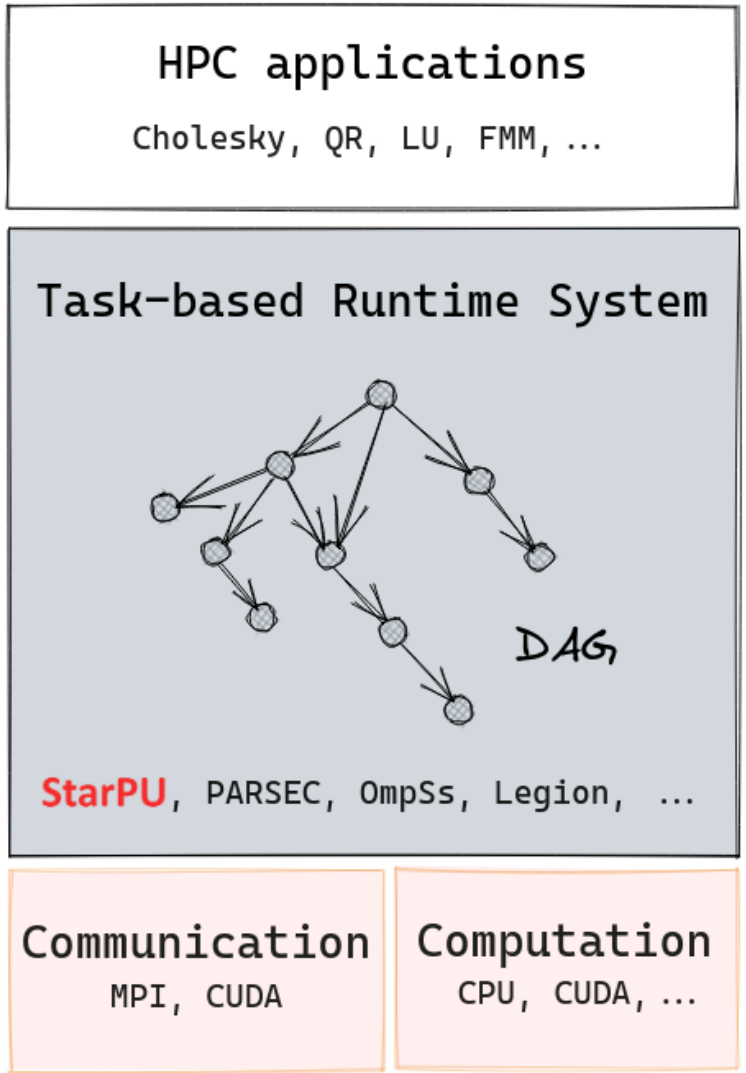


- HPC applications **taskified**



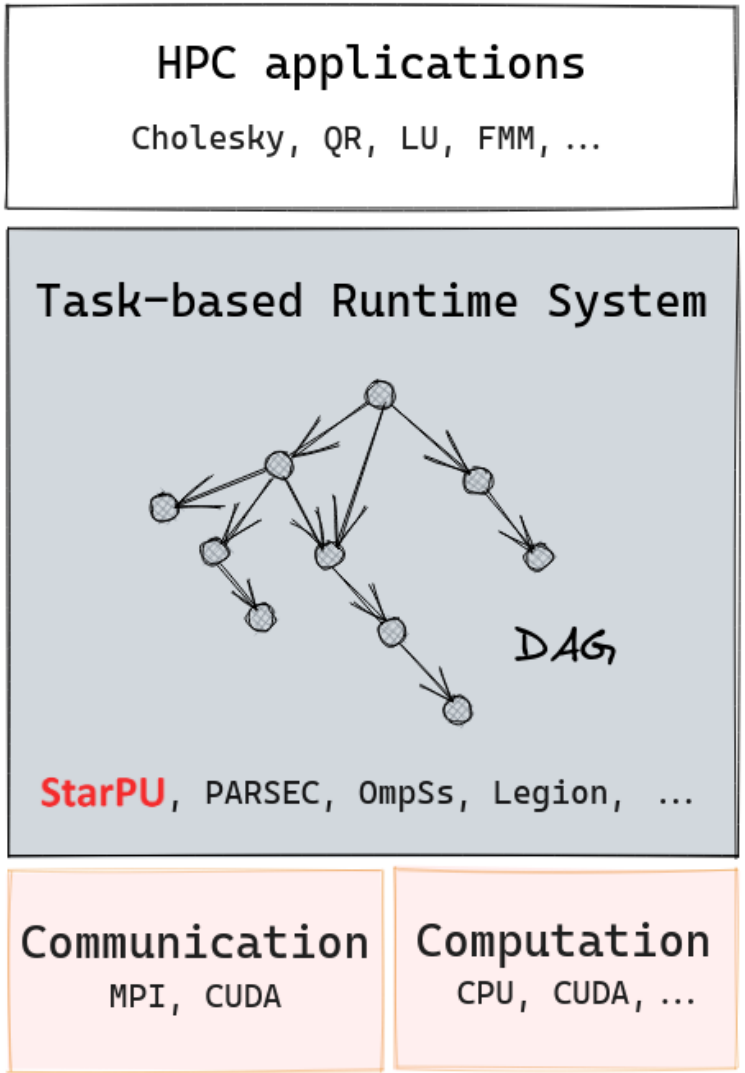
Task-based programming model

- HPC applications **taskified**
- Task-based Runtime System



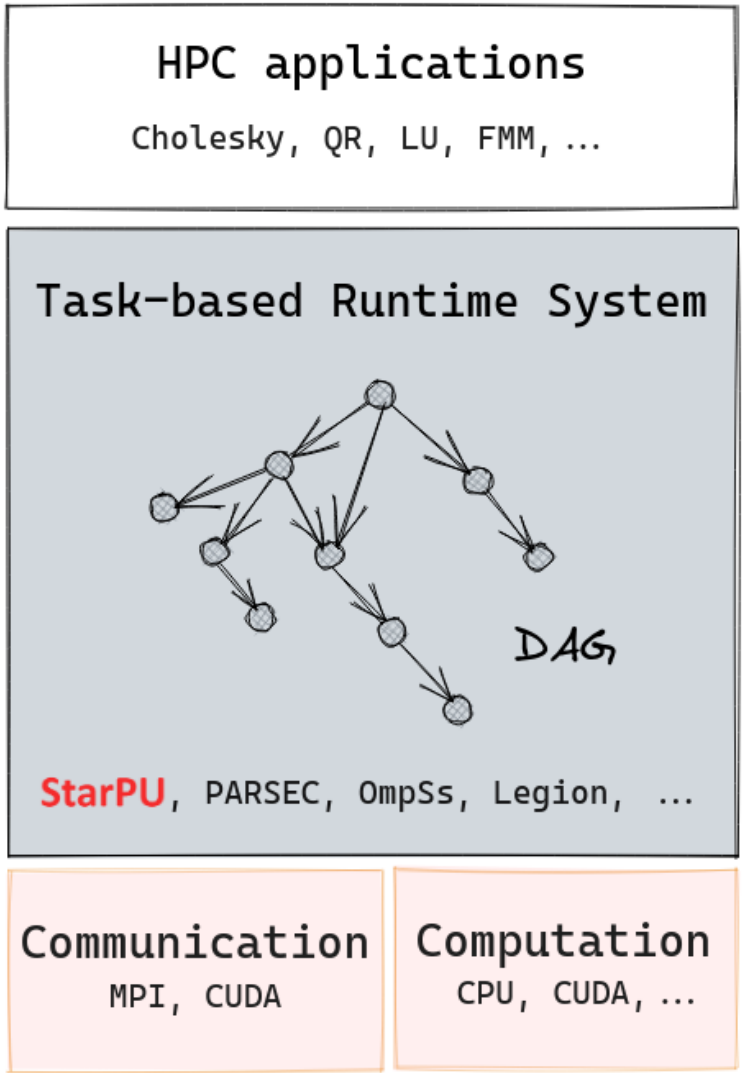
Task-based programming model

- HPC applications **taskified**
- Task-based Runtime System
  - > Resources management



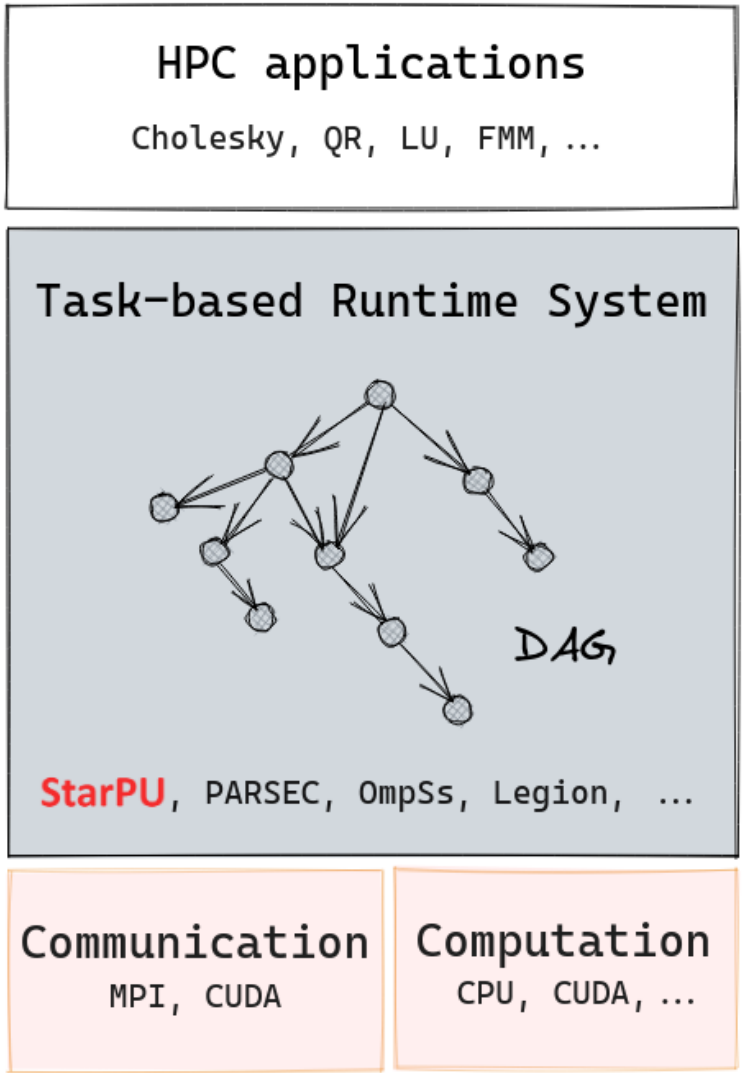
Task-based programming model

- HPC applications **taskified**
- Task-based Runtime System
  - > Resources management
  - > Data management



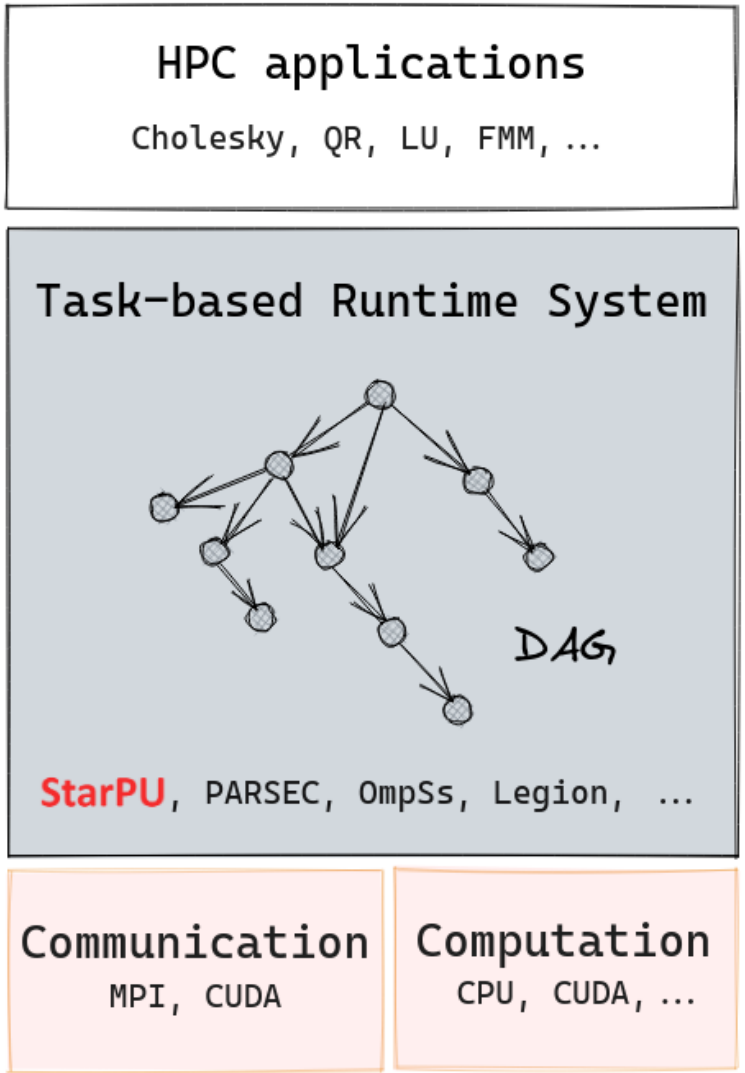
Task-based programming model

- HPC applications **taskified**
- Task-based Runtime System
  - > Resources management
  - > Data management
  - > Task scheduling



- HPC applications **taskified**
- Task-based Runtime System
  - > Resources management
  - > Data management
  - > Task scheduling

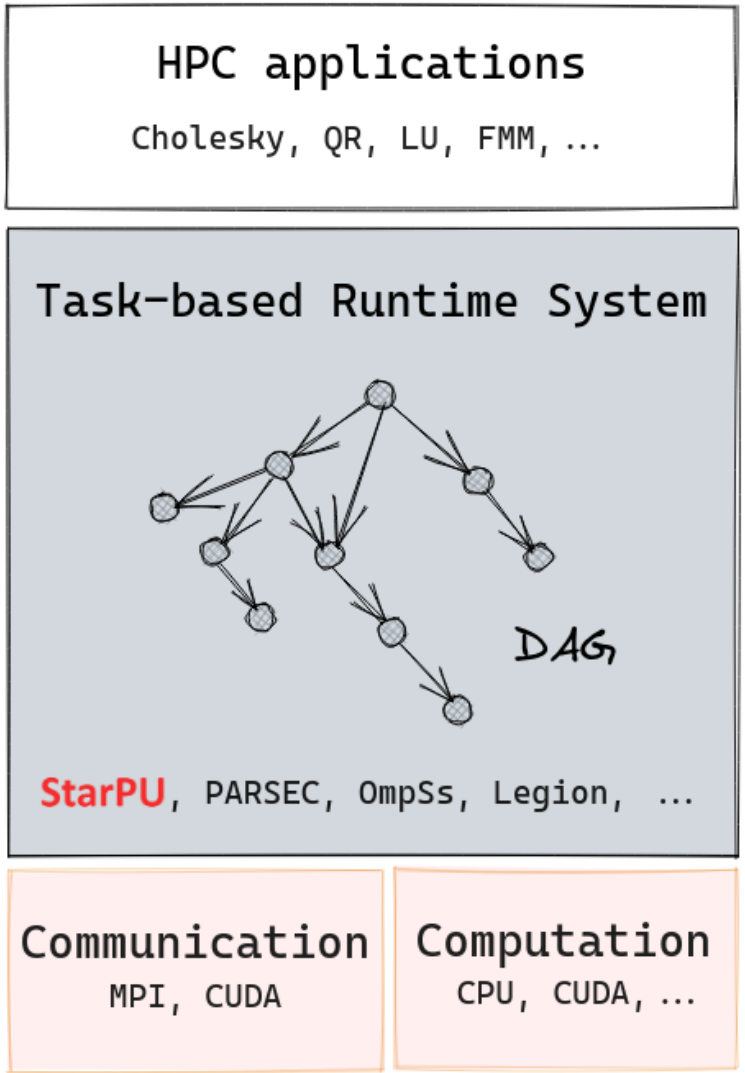
Scheduling goals



- HPC applications **taskified**
- Task-based Runtime System
  - > Resources management
  - > Data management
  - > Task scheduling

Scheduling goals

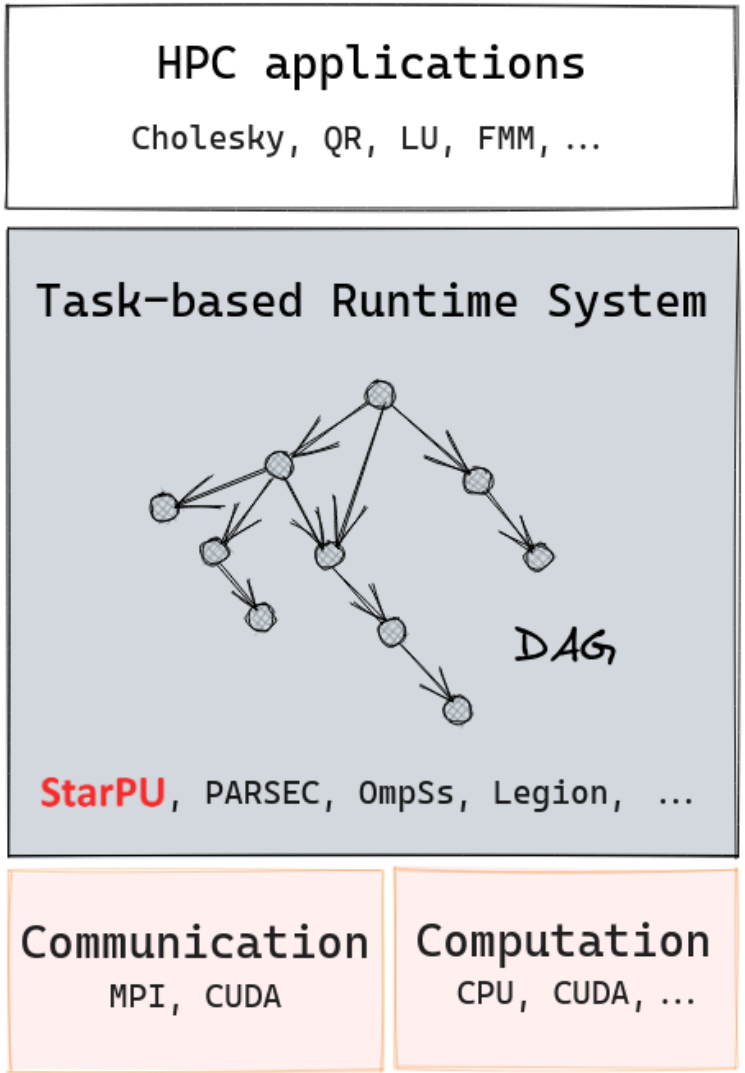
- Reduce the execution time



- HPC applications **taskified**
- Task-based Runtime System
  - > Resources management
  - > Data management
  - > Task scheduling

Scheduling goals

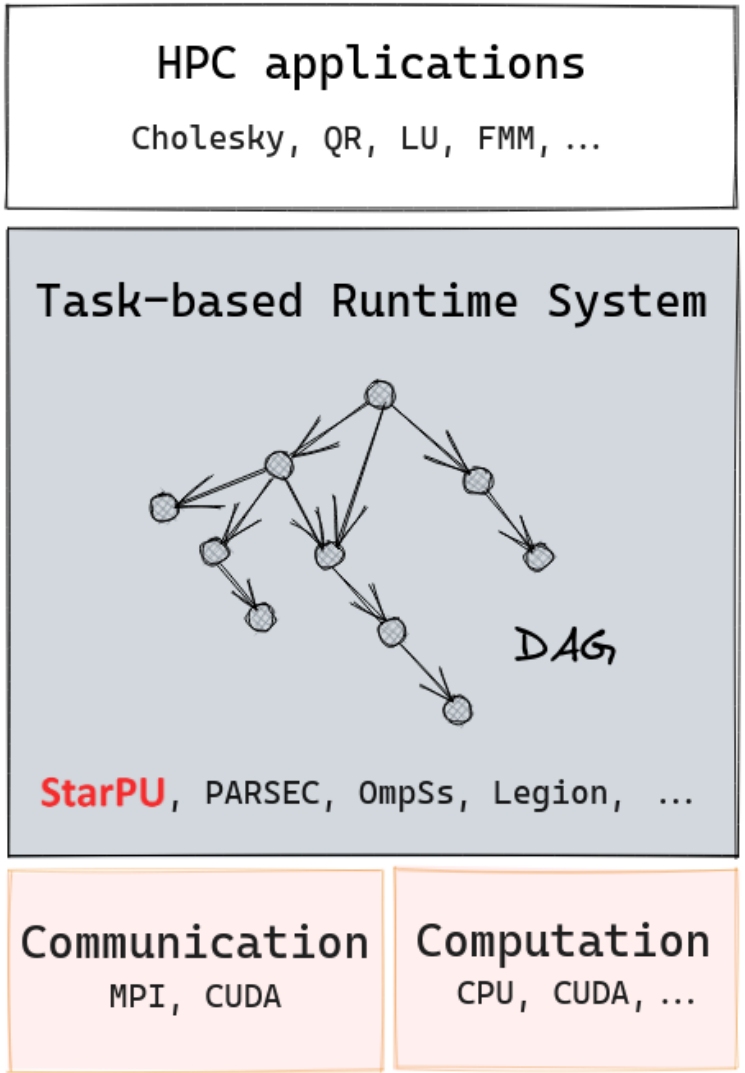
- Reduce the execution time
  - > Efficiently use the heterogeneous resources



- HPC applications **taskified**
- Task-based Runtime System
  - > Resources management
  - > Data management
  - > Task scheduling

Scheduling goals

- Reduce the execution time
  - > Efficiently use the heterogeneous resources
- Maximize the degree of parallelism



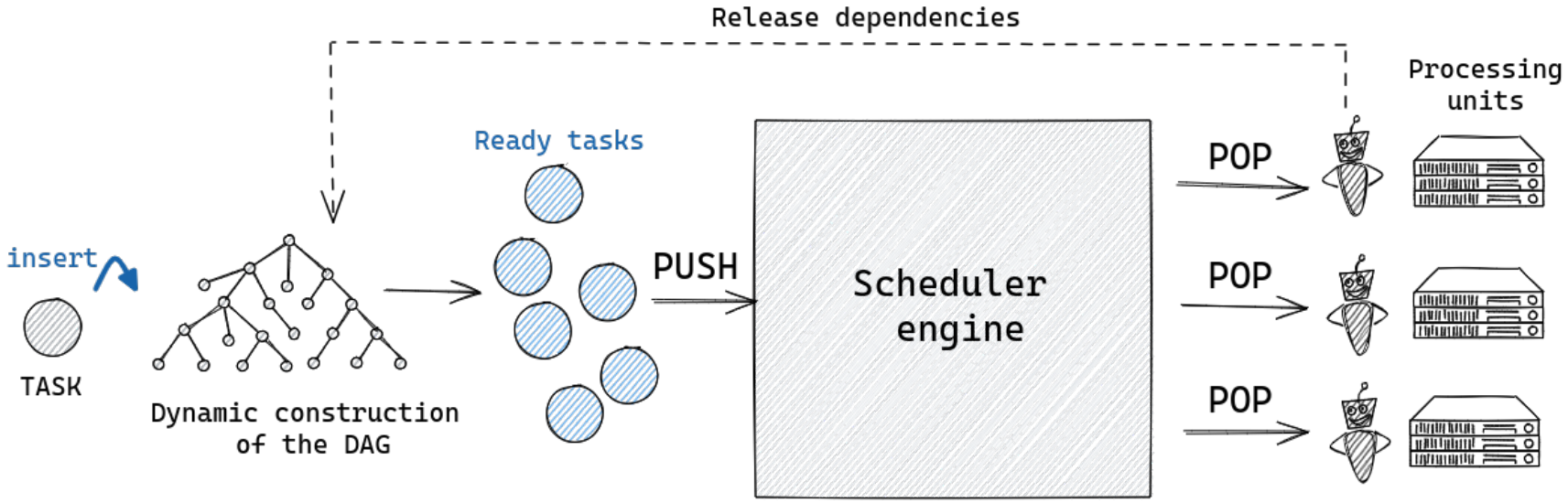
- **PaRSEC** : is an engine that provides multiple languages for describing the task graph.
- **Legion** : is a data-centric parallel programming system for writing portable HPC programs.
- **StarPU** : is a task programming library using Sequential Task Flow (STF) model.

- **PaRSEC** : is an engine that provides multiple languages for describing the task graph.
- **Legion** : is a data-centric parallel programming system for writing portable HPC programs.
- **StarPU** : is a task programming library using Sequential Task Flow (STF) model.

Some advantages of StarPU include:

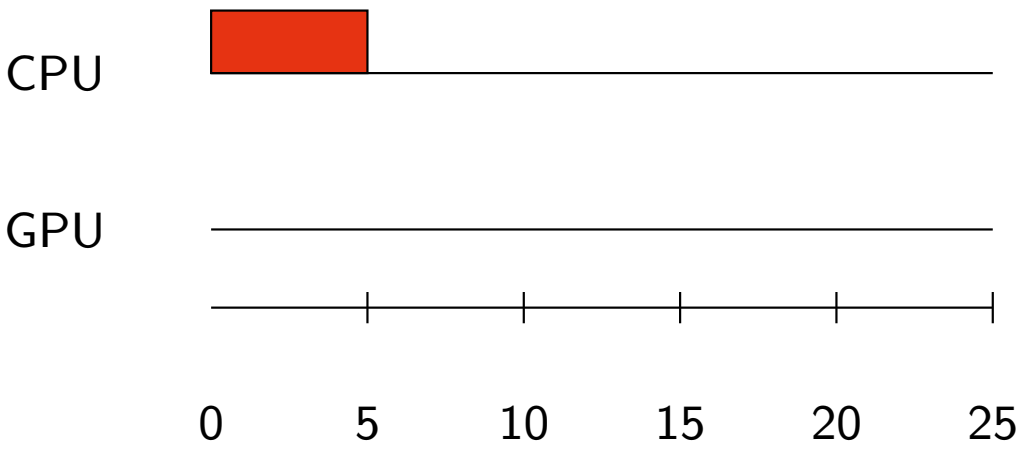
- A programmer can implement a custom scheduler.
- StarPU offers the execution time estimation of a task provided by a history-based performance model.

Dynamic scheduling in StarPU



- PUSH operation, when a **task** becomes ready to execute.
- POP operation, when a **resource** is idle and asks for a task to execute.

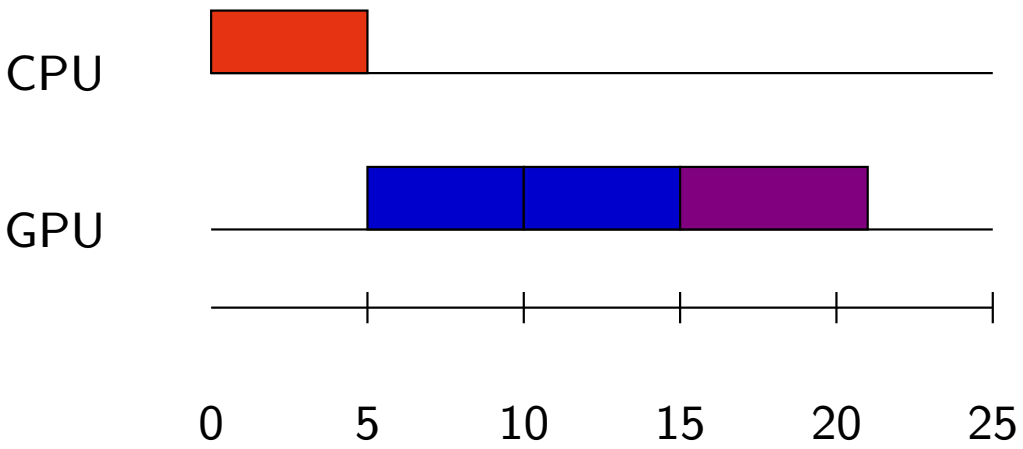
- First, an **affinity-based** strategy



Dependencies Task Graph

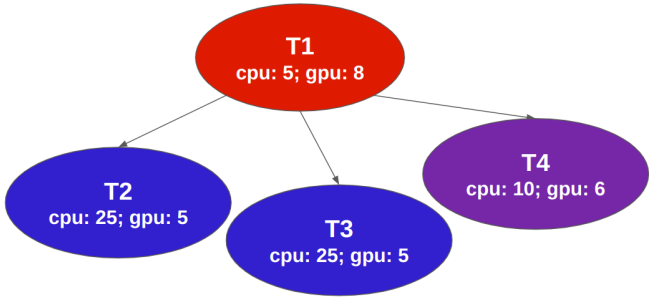
Dynamic construction

- First, an **affinity-based** strategy

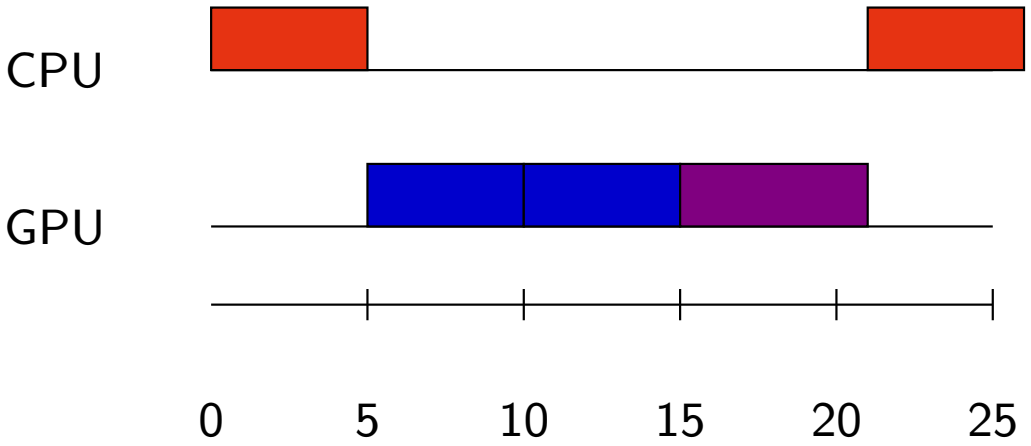


Dependencies Task Graph

Dynamic construction

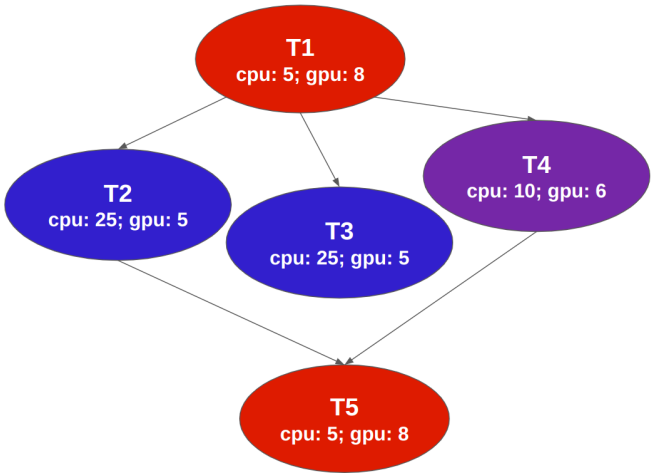


- First, an **affinity-based** strategy



Dependencies Task Graph

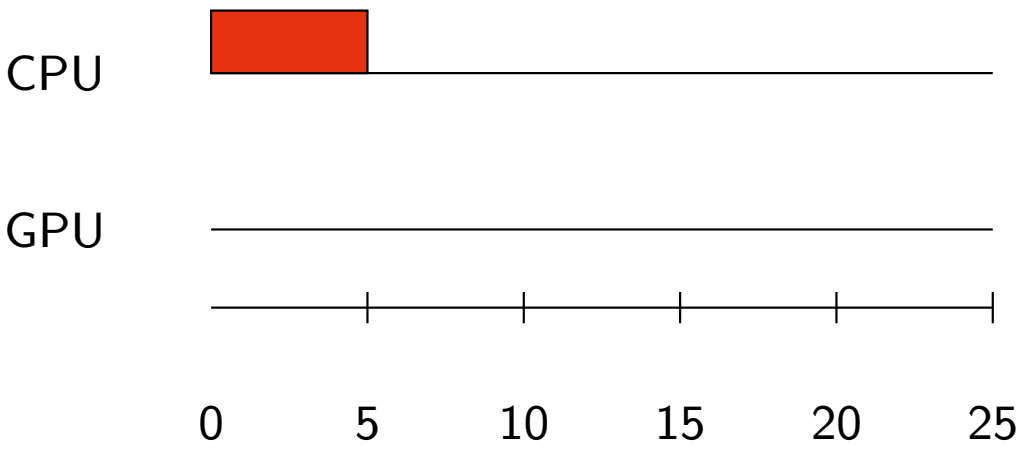
Dynamic construction



→ **makespan=26**

→ **idle time !**

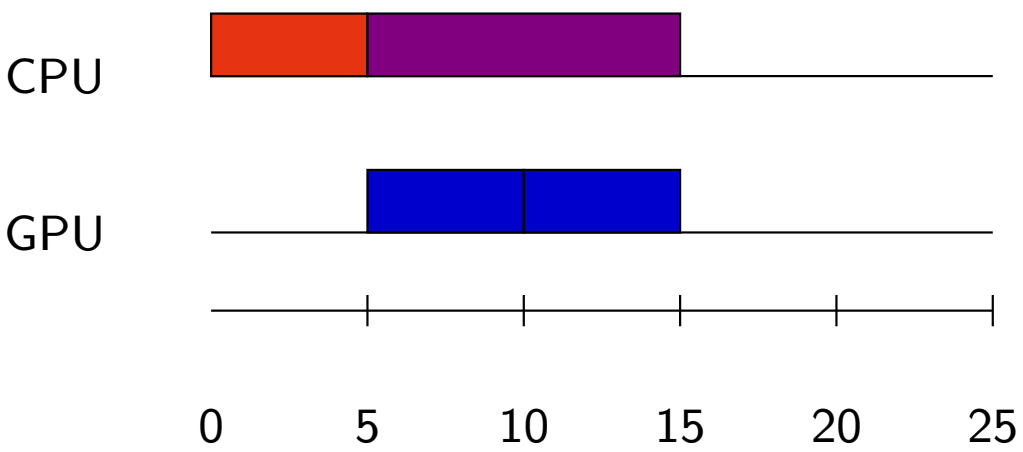
- A better strategy ? A **resource-based** strategy



Dependencies Task Graph

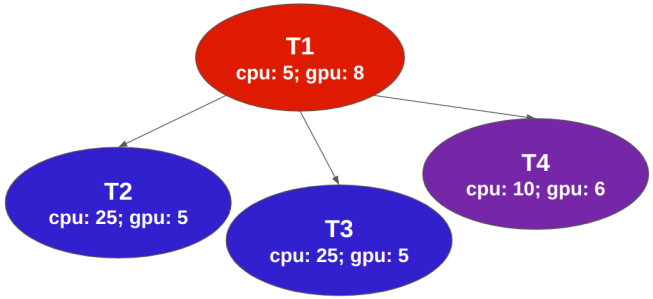
Dynamic construction

- A better strategy ? A **resource-based** strategy

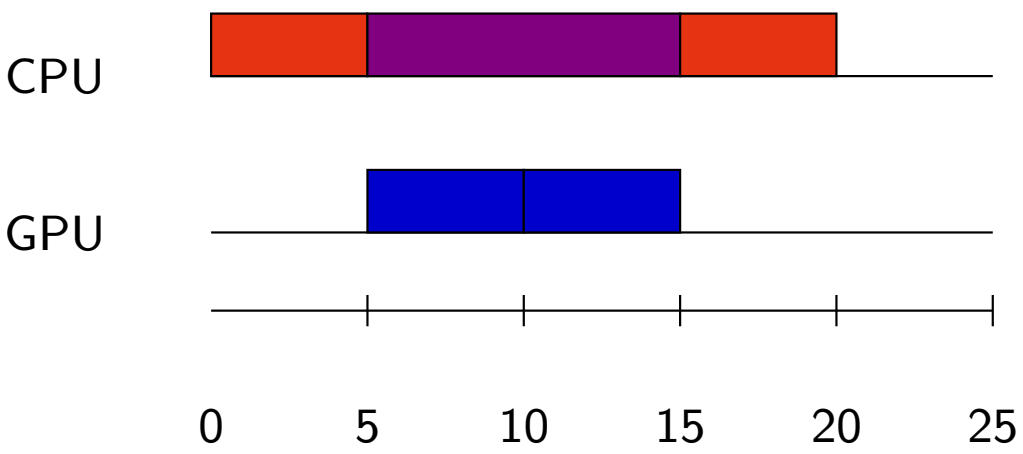


Dependencies Task Graph

Dynamic construction



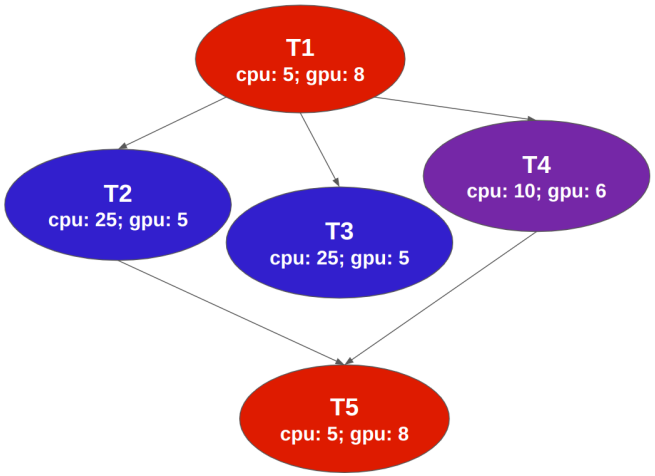
- A better strategy ? A **resource-based** strategy



→ **makespan=20**

Dependencies Task Graph

Dynamic construction



Existing scheduling strategies

- **Affinity-based:** assign tasks to resources based on task's preference for a specific resource: *HeteroPrio (Agullo et al.),..*
- **Resource-centric:** aim to maximize resource utilization by allocating tasks to processors based on resource availability and workload: *work stealing (Lima et al.),..*
- **Task-centric:** focus on improving task execution time and reducing task waiting time: *Dmdas (Augonnet et al.),..*

Existing scheduling strategies

- **Affinity-based:** assign tasks to resources based on task's preference for a specific resource: *HeteroPrio (Agullo et al.),..*
- **Resource-centric:** aim to maximize resource utilization by allocating tasks to processors based on resource availability and workload: *work stealing (Lima et al.),..*
- **Task-centric:** focus on improving task execution time and reducing task waiting time: *Dmdas (Augonnet et al.),..*

Existing scheduling strategies

- **Affinity-based:** assign tasks to resources based on task's preference for a specific resource: *HeteroPrio (Agullo et al.),..*
- **Resource-centric:** aim to maximize resource utilization by allocating tasks to processors based on resource availability and workload: *work stealing (Lima et al.),..*
- **Task-centric:** focus on improving task execution time and reducing task waiting time: *Dmdas (Augonnet et al.),..*

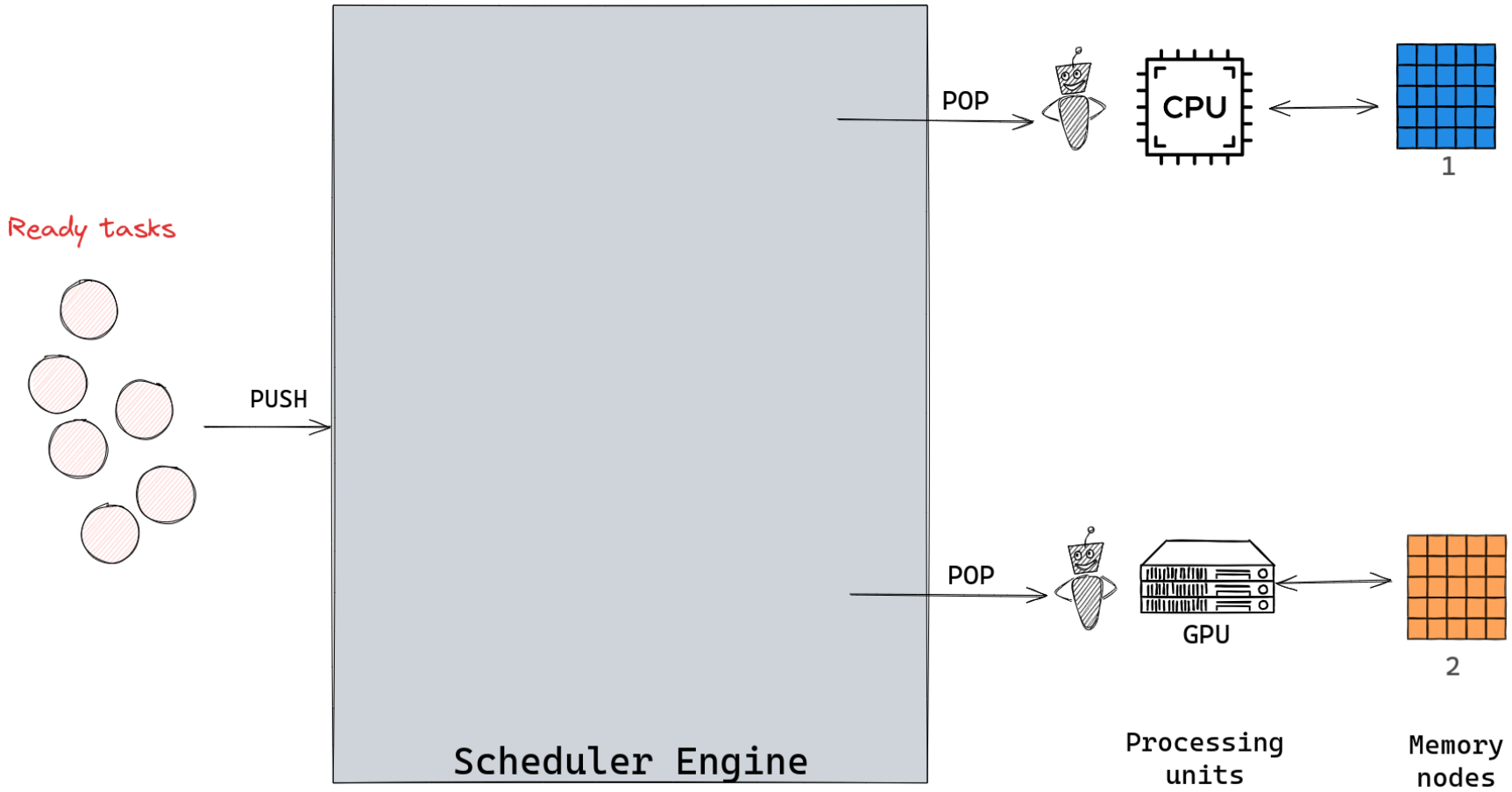
Existing scheduling strategies

- **Affinity-based:** assign tasks to resources based on task's preference for a specific resource: *HeteroPrio (Agullo et al.),..*
- **Resource-centric:** aim to maximize resource utilization by allocating tasks to processors based on resource availability and workload: *work stealing (Lima et al.),..*
- **Task-centric:** focus on improving task execution time and reducing task waiting time: *Dmdas (Augonnet et al.),..*

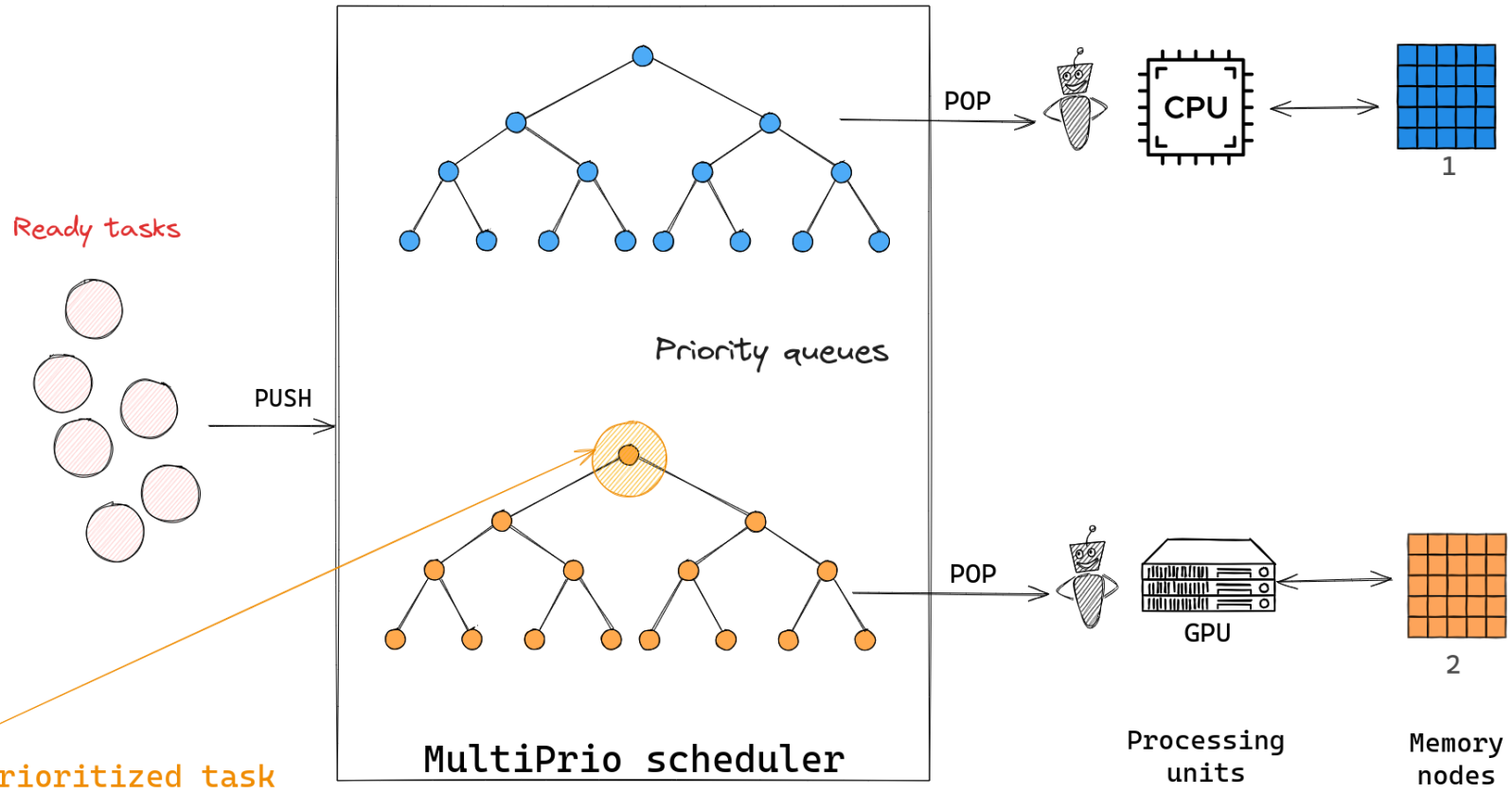
Our goal ?

Find a trade-off between resource affinity, task criticality, data locality, and workload balancing on the resources.

General idea of our scheduler MultiPrio implemented into StarPU

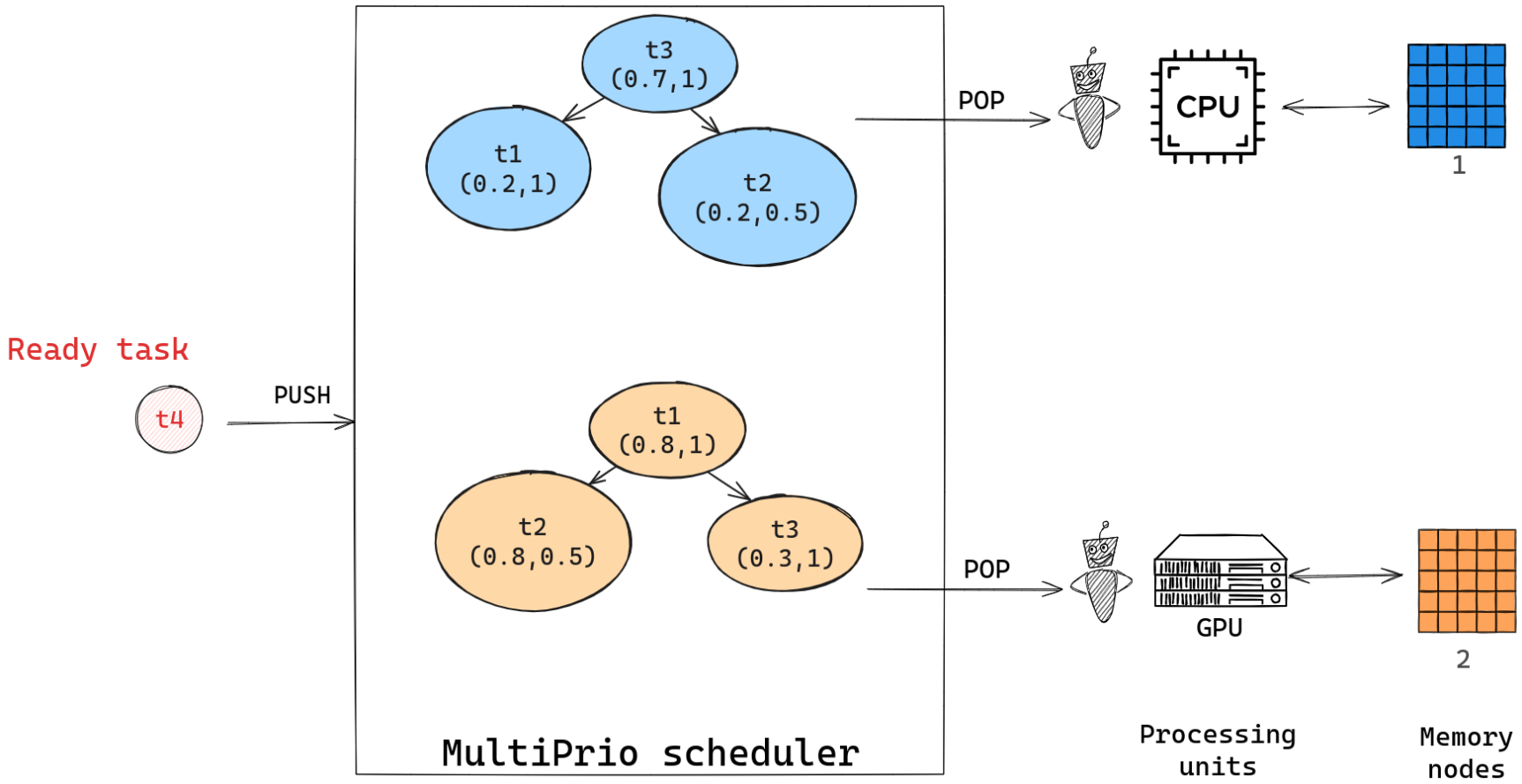


Priority queue per memory node (RAM, GPU memory, ...) that contains tasks with scores sorted in a lexicographical order.

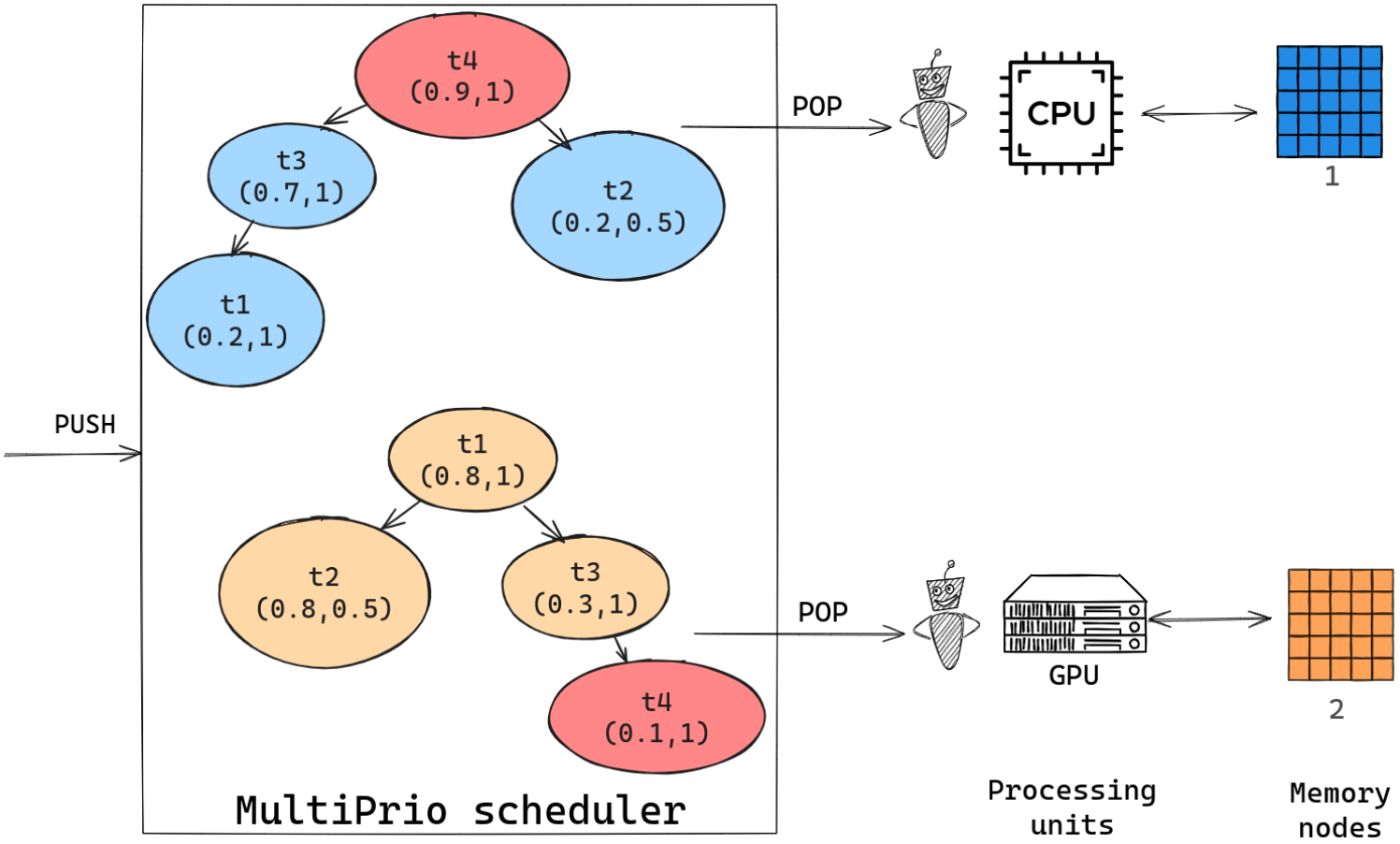


Most prioritized task
with a pair of scores (s1, s2)
can be executed on GPU tied to memory node 2

For a newly **ready task**, we compute the **scores** for the different memory nodes.



For a newly **ready task**, we compute the **scores** for the different memory nodes.



(1) Resource Affinity: Gain/Acceleration Heuristic

$$gain(t,a) = \begin{cases} 1 & |\mathcal{A}| = 1 \\ \frac{(\delta(t,a_{2nd}) - \delta(t,a)) + |hd(a)|}{2 * |hd(a)|} & a \text{ is the fastest} \\ \frac{(\delta(t,a_{1st}) - \delta(t,a)) + |hd(a)|}{2 * |hd(a)|} & \text{else} \end{cases}$$

$\delta(t,a)$: estimated execution time of t; $hd(a)$: the highest execution time difference

Example of the gain heuristic calculation with 3 tasks and 2 architecture types:

	t_A	t_B	t_C
$\delta(t,a_1)$	1ms	5ms	20ms
$\delta(t,a_2)$	20ms	10ms	10ms
$gain(t,a_1)$	1	0.631	0.236
$gain(t,a_2)$	0	0.368	0.763

(2) Task Criticality: NOD Heuristic

This heuristic calculates a criticality score for a task.

$$NOD(t) = \sum_{s_i \in \lambda^+(t,\mathcal{P}_m)} \frac{1}{|\lambda^-(s_i, \mathcal{P}_m)|}$$

$\lambda^-(t)$: predecessors of t; $\lambda^+(t)$: successors of t; $\mathcal{P}_m$: processing units

A task is prioritized if it has **more successors** that will be released, which will create more workload and **improve the parallelism**.

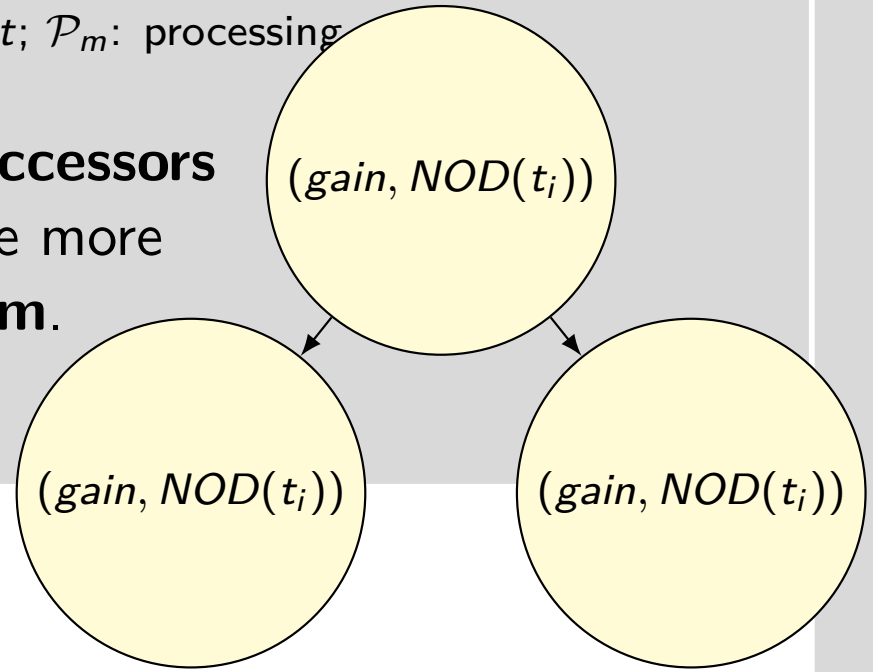
(2) Task Criticality: NOD Heuristic

This heuristic calculates a criticality score for a task.

$$NOD(t) = \sum_{s_i \in \lambda^+(t,\mathcal{P}_m)} \frac{1}{|\lambda^-(s_i, \mathcal{P}_m)|}$$

$\lambda^-(t)$: predecessors of t; $\lambda^+(t)$: successors of t; $\mathcal{P}_m$: processing

A task is prioritized if it has **more successors** that will be released, which will create more workload and **improve the parallelism**.



Gain scores are equal for the same type of task

Scheduling with Multiple Priorities (POP)

(3) Data Locality

Locality Strategy - Sum of Data Hosted heuristic (*Bramas, 2019*)

$$LS\_SDH^2(m,t) = \left(\sum_{d \in D^R_{t,m}} d.size \right) + \left(\sum_{d \in D^W_{t,m}} d.size^2 \right) \quad (1)$$

$D_{t,m}$: the set of data used by task t that is on memory node m.
$D^R_{t,m}$ and $D^W_{t,m}$: the sets of data used by t that is on m and is accessed in read and write mode, respectively.

→ Select the task with the most available data on the current node from the top priority queue.

Applications

- **CHAMELEON**: A dense linear algebra library. (*Regular*)
- **TBFMM**: Task-based Fast Multipole Method. (*Irregular*)
- **QR MUMPS**: Sparse direct linear solver. (*Highly irregular*)

Platforms

- **Intel-V100**:
  - 2 Intel Xeon Gold 6142 CPUs (16 cores each, 2.6GHz)
  - 384 GB of memory
  - 2 Nvidia V100 GPUs (16 GB each)
- **AMD-A100**:
  - 2 AMD Zen3 EPYC 7513 CPUs (32 cores each, 2.6GHz)
  - 512 GB of memory
  - 2 Nvidia A100 GPUs (40 GB each)

- **Dmdas Scheduler:**
  - > Highly tuned for dense linear algebraic routines (Cholesky).
  - > Over-prioritizes accelerators (GPUs), leading to under-utilization of CPUs.
    → **Priorities over Affinity**
- **HeteroPrio Scheduler:**
  - > Highly tuned for task-based FMM.
  - > A semi-automatic scheduler where users must provide acceleration ratio per task type.
    → **Affinity over Priorities**

- **Dmdas Scheduler:**
  - > Highly tuned for dense linear algebraic routines (Cholesky).
  - > Over-prioritizes accelerators (GPUs), leading to under-utilization of CPUs.
    → **Priorities over Affinity**
- **HeteroPrio Scheduler:**
  - > Highly tuned for task-based FMM.
  - > A semi-automatic scheduler where users must provide acceleration ratio per task type.
    → **Affinity over Priorities**
- **MultiPrio Scheduler:**
  - > Affinity and criticality scores.
  - > Effective utilization of both CPU and GPU resources for highly parallelized DAGs (sparse QR)
    → **Finding a compromise**

Chameleon dense linear algebra kernels

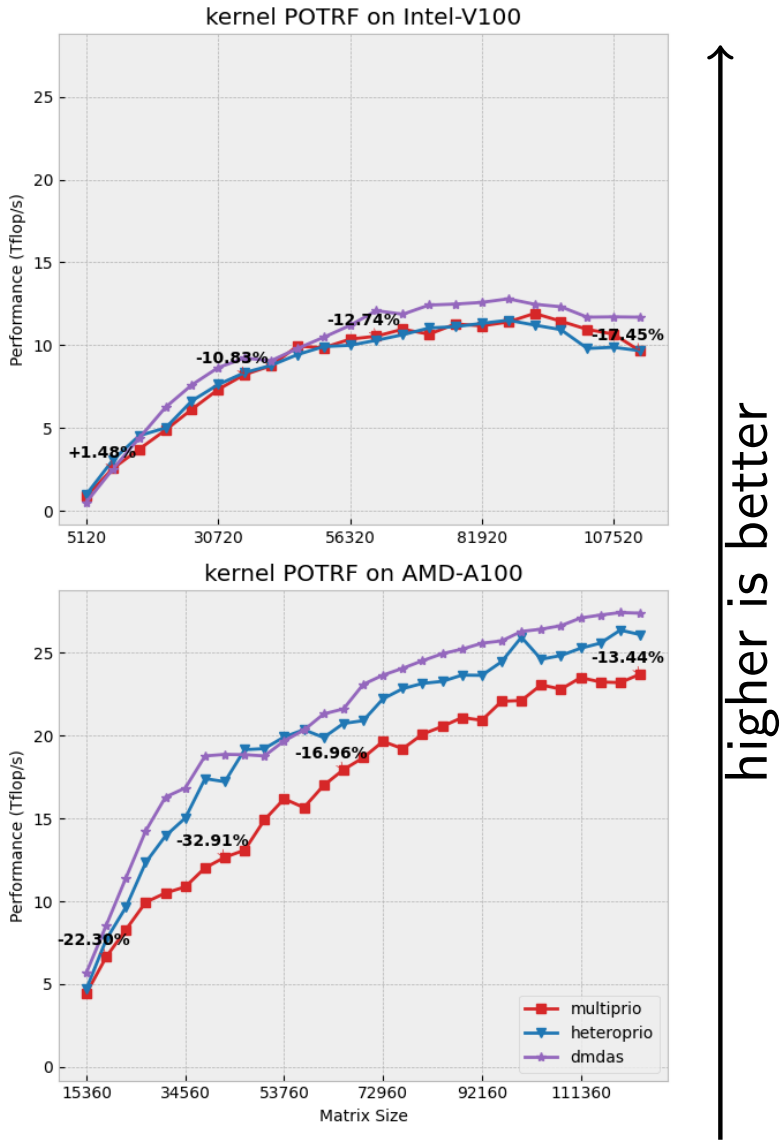
- Matrix sizes
  between 5k to 120k
- Runs with 1, 2, 4 and 8 gpu streams

- Regular application
- Long critical path
- **User-defined priorities**
  optimized by experts offline

Performance is affected by:

→ task-criticality (priorities)

→ data locality



TBFMM: Task-based FMM

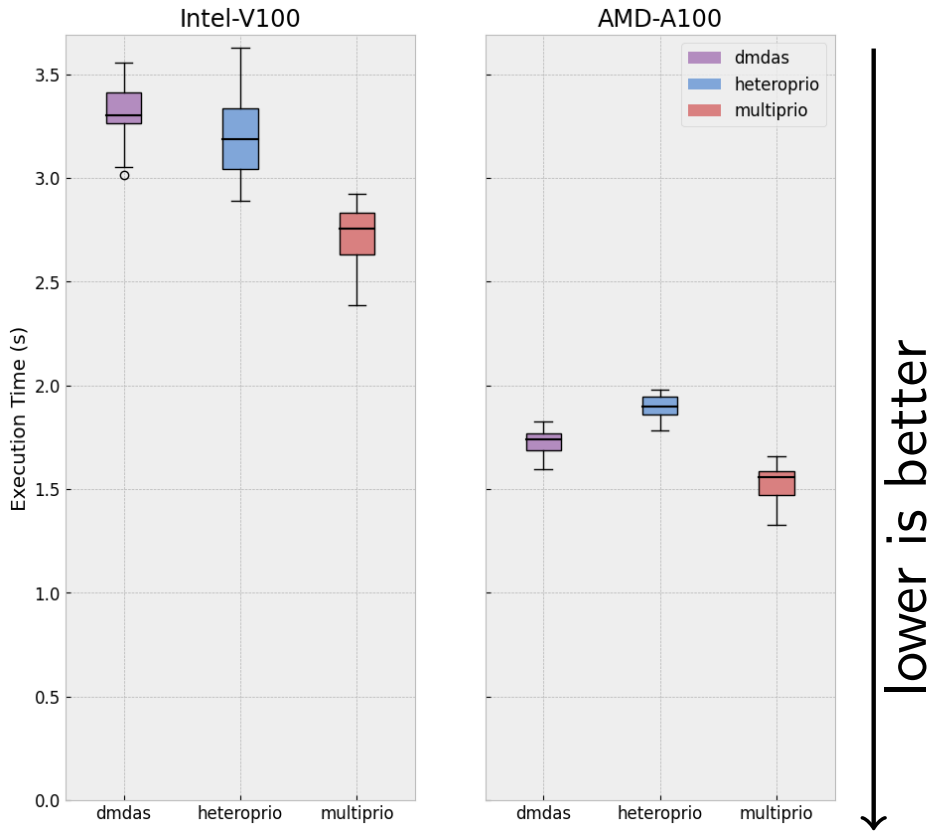
- Problem size:
  1M particles
- Tree height: 6
- Runs with 1, 2, 4 and 8 gpu streams

- Irregular application
- Disconnected DAG
- No user priorities

Results are affected by:

→ resource affinity

→ data locality



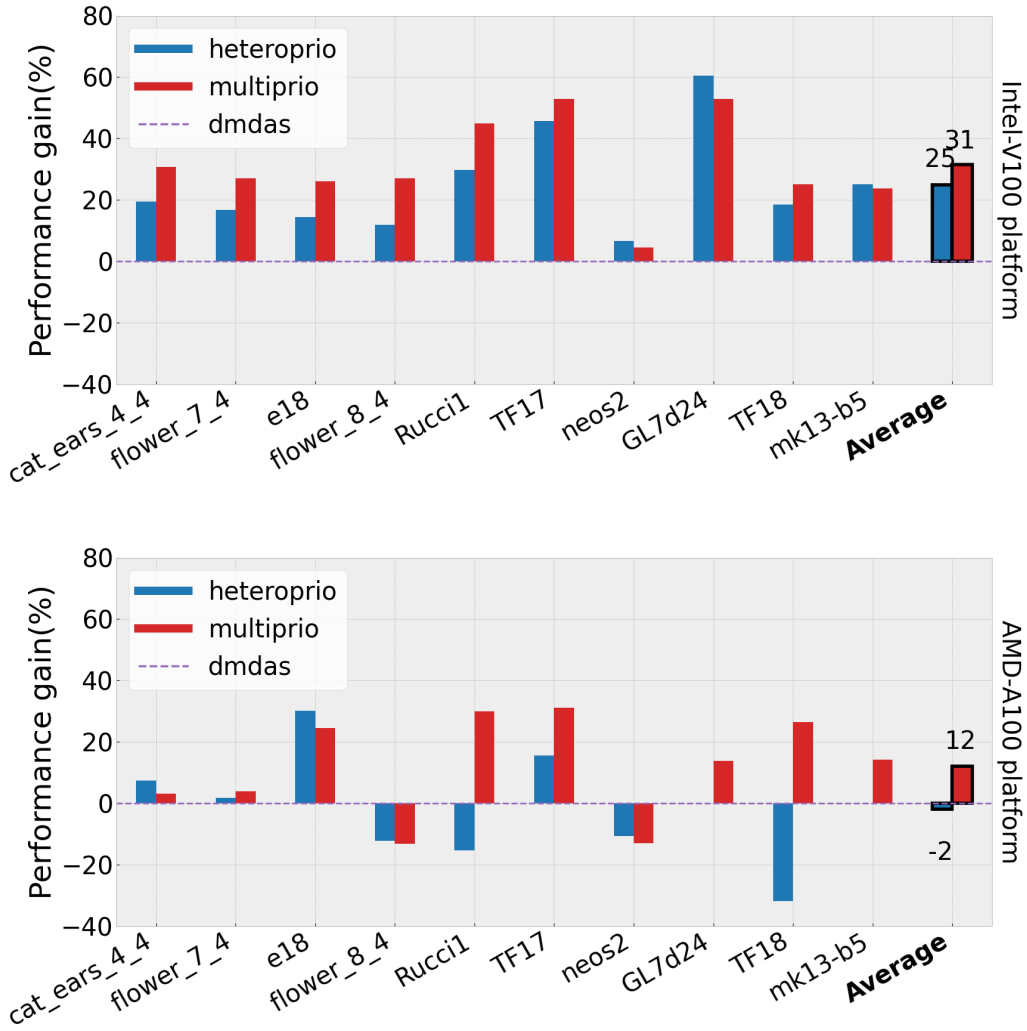
QR MUMPS: Sparse QR factorization

- Various sparse matrices
- Runs with 1, 2, 4 and 8 gpu streams

- Highly irregular application
- No user priorities

Performance is affected by:

→ resource affinity
→ task criticality
→ data locality



MultiPrio scheduler:

- Implemented in StarPU using priority queues data structures. https://gitlab.inria.fr/htayeb/starpu-multiprio-scheduler
- Assigns affinity and criticality scores. (compromise)
- Allows slower workers to assist busy faster workers (heterogeneity)
- Minimizes data transfers with locality heuristic.

Proven Efficiency:

- Up to 40% improvement over Dmdas in QR_MUMPS.
- Efficient for irregular workloads.
- Increased parallelism in the DAG leads to better efficiency.

Promising directions ..

- **Data locality:**
  - > Refine data locality heuristic to reduce the data transfers between different memory nodes, especially when the problem size is larger than the available storage.
- **Energy Efficiency:**
  - > Incorporate energy efficiency heuristics in the score per task.
  - > Re-balance workload between CPUs and accelerators without compromising performance.
    → Energy efficiency metric (Gflops/Watt)

Promising directions ..

- **Data locality:**
  - > Refine data locality heuristic to reduce the data transfers between different memory nodes, especially when the problem size is larger than the available storage.
- **Energy Efficiency:**
  - > Incorporate energy efficiency heuristics in the score per task.
  - > Re-balance workload between CPUs and accelerators without compromising performance.
    → Energy efficiency metric (Gflops/Watt)

Thank you !

Hayfa is looking for a postdoc position in France next spring.

This work is supported by the TEXTAROSSA project G.A. n.956831, as part of the EuroHPC initiative.