Memory-Aware Scheduling of Tasks Sharing Data on Multiple GPUs

From an IPDPS 2022 paper

Maxime GONTHIER

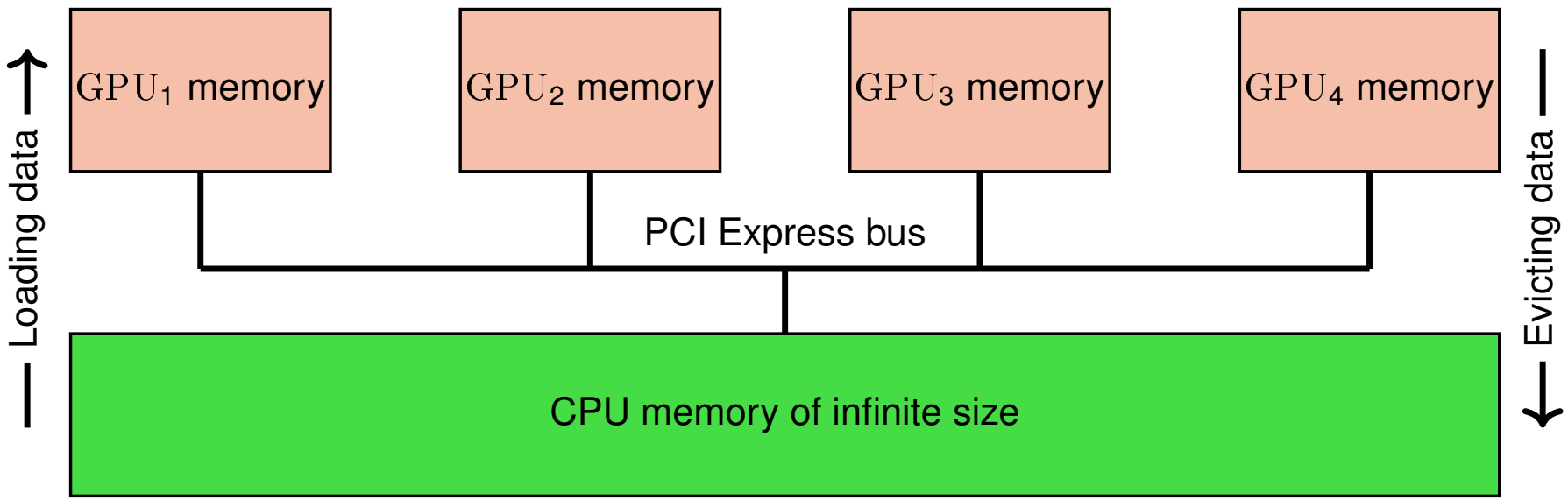
Supervised by Loris MARCHAL and Samuel THIBAULT

maxime.gonthier@ens-lyon.fr

LIP - ROMA - LaBRI - STORM - Inria
ENS de Lyon - Université de Bordeaux

February 9, 2022

Why focus on memory-aware scheduling ?



GPUs are quick but have a limited memory

They share the same PCI bus with limited bandwidth

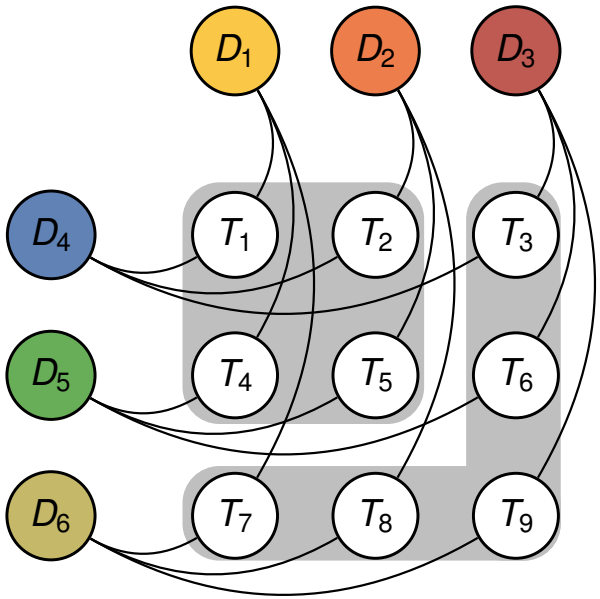
ENS DE LYON université de BORDEAUX

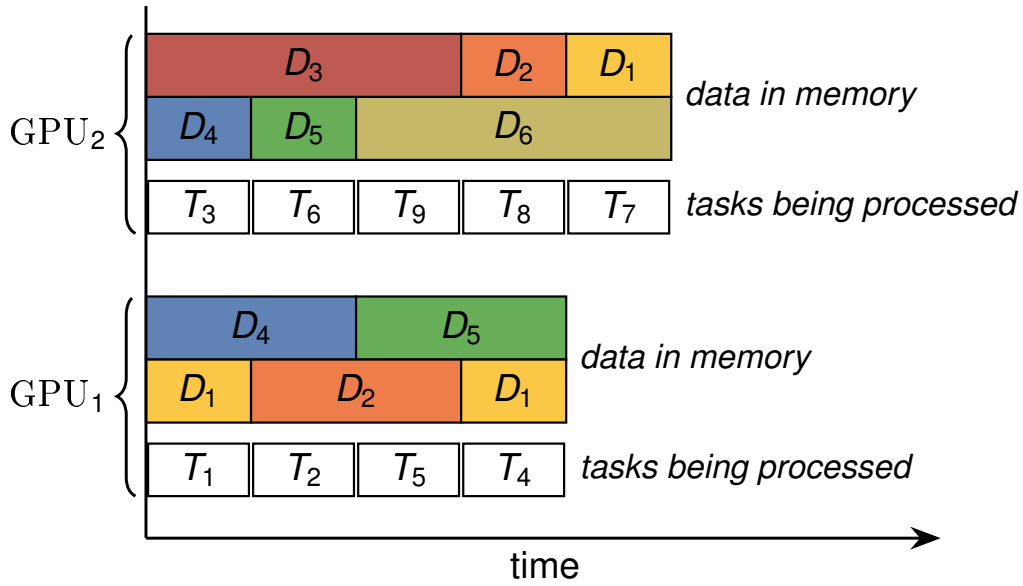
The goal is to minimize makespan with:

- Independent tasks
- Tasks share data
- Limited GPU memory

- Balancing tasks among GPUs $\rightarrow$ Reduce total execution time
- Ordering tasks inside each GPU $\rightarrow$ Reduce data transfers

Example with 2D grid dependencies





- In this example $Memory = 2$
- D_1 is the only data loaded twice

ENS DE LYON université de BORDEAUX

Problem modeling

Steps before processing each task

1. GPU_k wants to process task T
2. Some data are evicted from the memory of GPU_k
3. Data in T that are not yet in memory are loaded in the memory of GPU_k
4. Task T is processed on GPU_k

Hypothesis

- Task are independant and homogeneous (can be adapted to heterogeneous task)
- Data all have the same size (again, can be adapted to data of different sizes)

ENS DE LYON université de BORDEAUX

A bi-objective optimization problem

Objective 1: Load Balancing

$$\textit{minimize} \max_k \textit{ number of tasks allocated to } \mathrm{GPU}_k$$

Objective 2: Data Movement

Minimize the number of **load** from the main memory:

$$\#\textit{Loads}_k = \sum \textit{data load for each } T \textit{ computed on } \mathrm{GPU}_k$$

$$\textit{minimize} \sum_k \#\textit{Loads}_k$$

Algorithms tested

2 schedulers from StarPU

- Eager (our baseline)
- Deque Model Data Aware Ready (DMDAR)

1 algorithm from the state of the art

- hMETIS

1 new algorithm

- Data-Aware Reactive Task Scheduling (DARTS)

Dynamic scheduler of StarPU: DMDAR

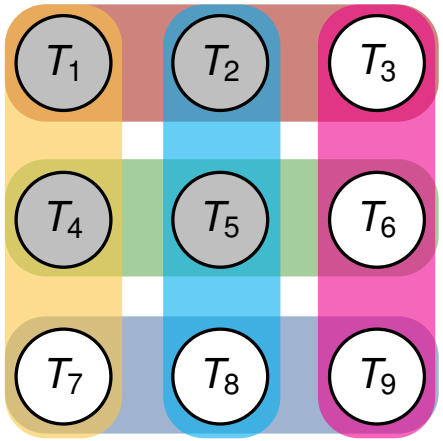
Strategy

Schedule tasks so their completion time is minimal based on computation + communication times

+ **Ready** reordering heuristic on GPU_k

input : List L of tasks allocated on GPU_k
while $L \neq \emptyset$ **do**
    Search first $T \in L$ requiring the fewest data transfers
    Wait for all data in $\mathcal{D}(T)$ to be in GPU_k memory
    Start processing T
end

Using (hyper-)graph partitioning: hMETIS



Hypergraph $\rightarrow$ Represent data D being used by different tasks
One hyperedge for each data $D \rightarrow$ Includes all the vertices corresponding to tasks using $D \rightarrow$ Better represent data reuse

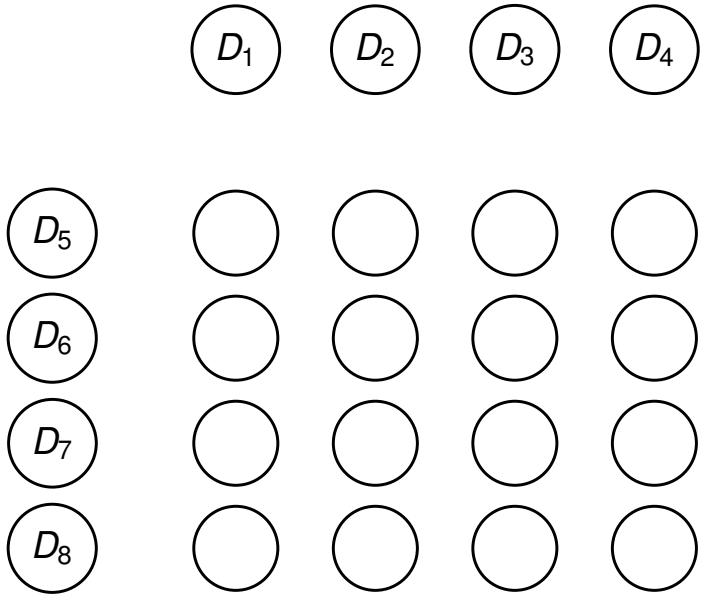
Strategy

1. Decompose the set of tasks into several subsets of similar size while minimizing the number of common data among subsets
2. Each subset is allocated to a GPU
3. Use the **Ready** strategy
4. Implement dynamic load balancing using task stealing

ENS DE LYON université de BORDEAUX

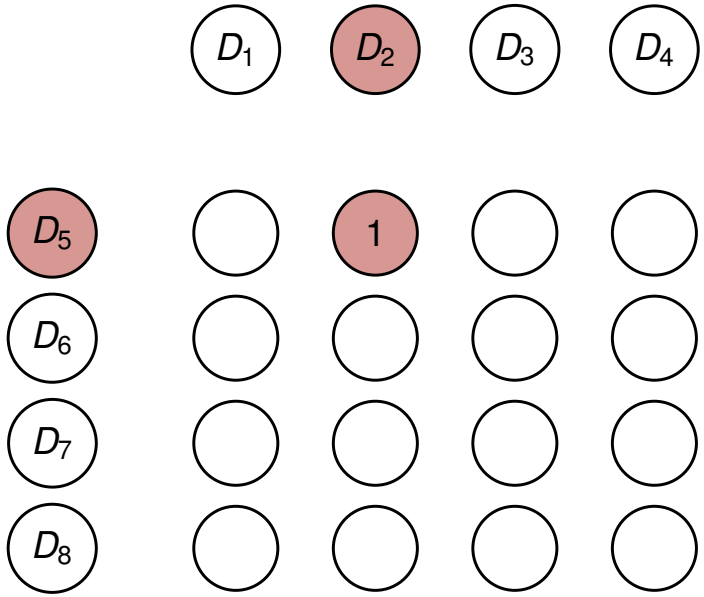
Data-Aware Reactive Task Scheduling: DARTS

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



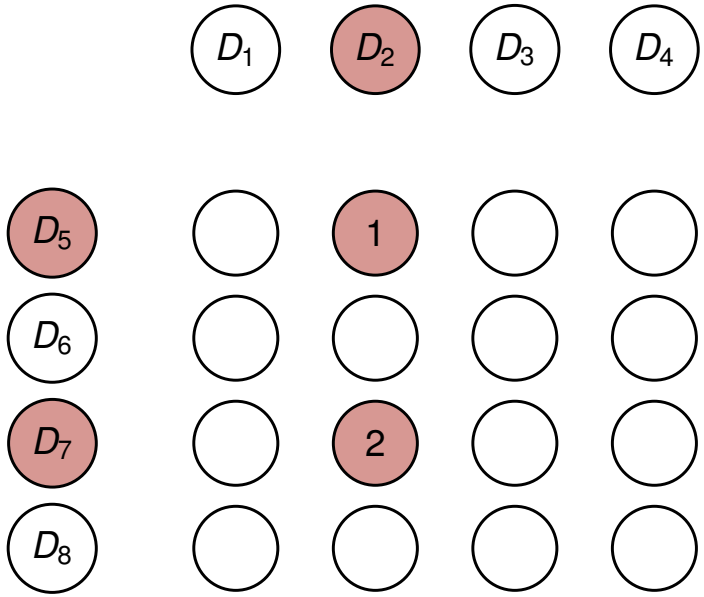
Data-Aware Reactive Task Scheduling: DARTS

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



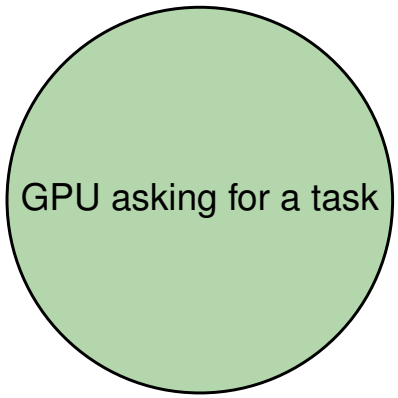
Data-Aware Reactive Task Scheduling: DARTS

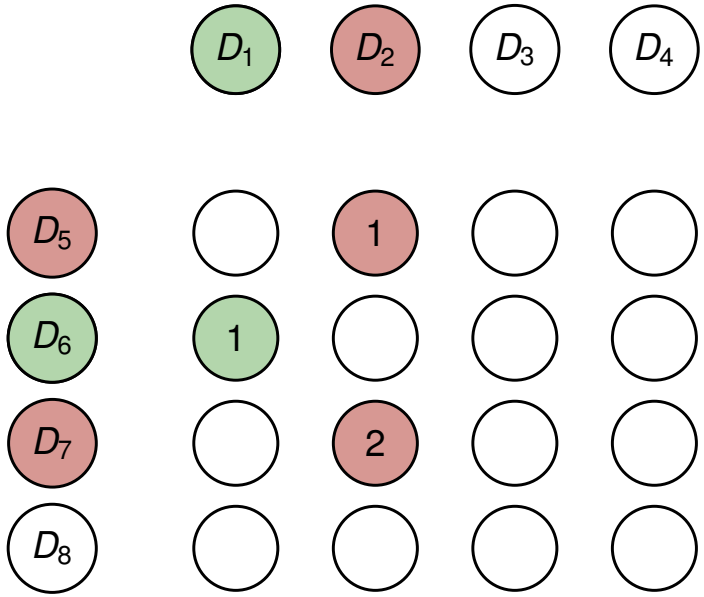
- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



Data-Aware Reactive Task Scheduling: DARTS

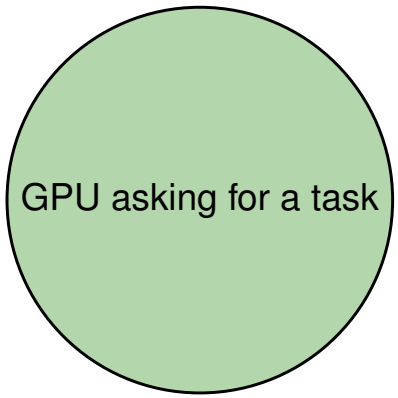
- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**

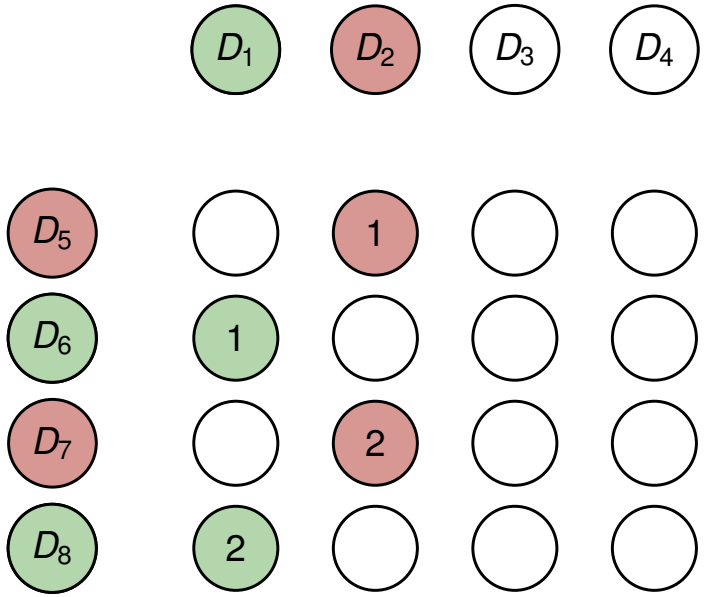




Data-Aware Reactive Task Scheduling: DARTS

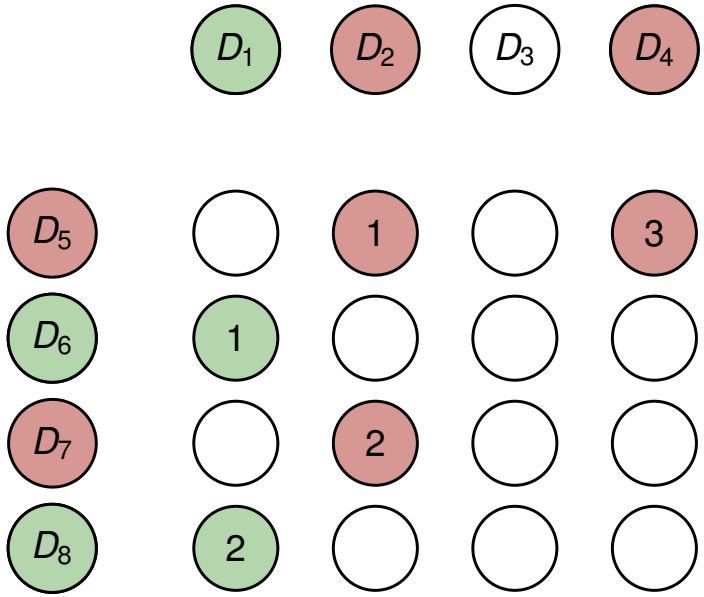
- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**





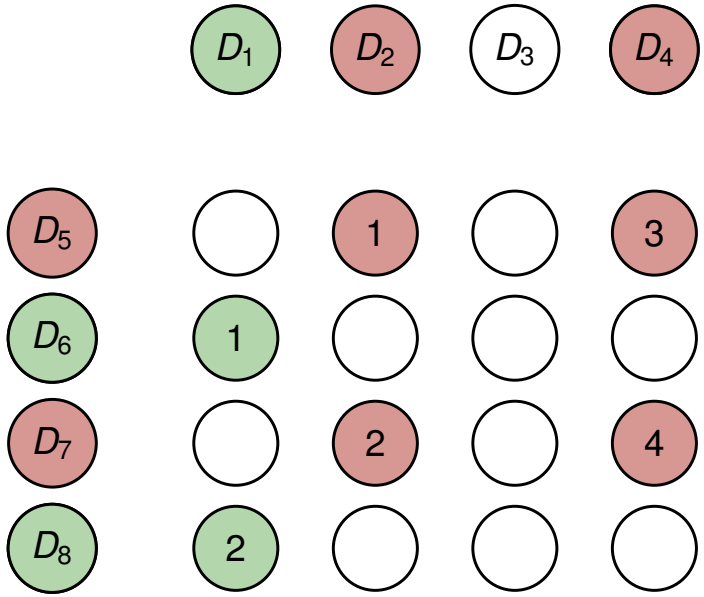
Data-Aware Reactive Task Scheduling: DARTS

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



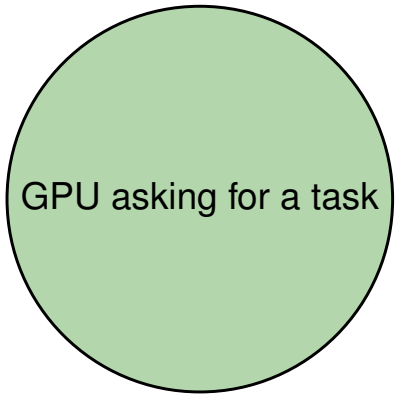
Data-Aware Reactive Task Scheduling: DARTS

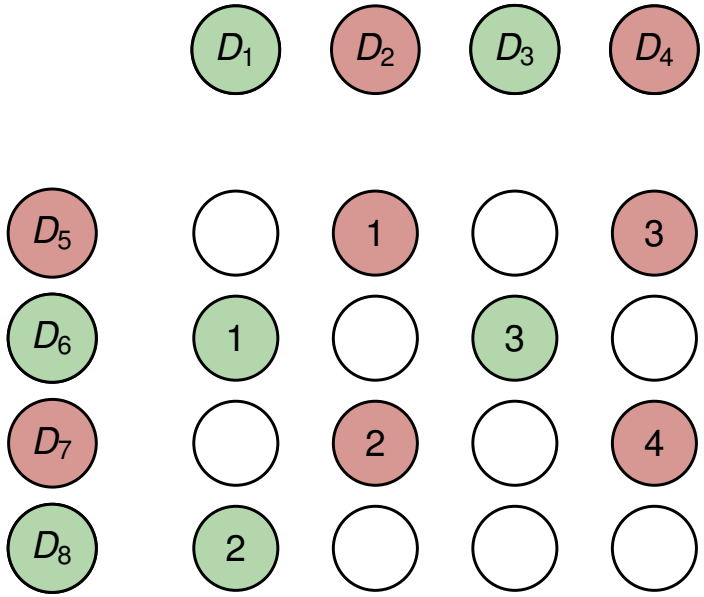
- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**



Data-Aware Reactive Task Scheduling: DARTS

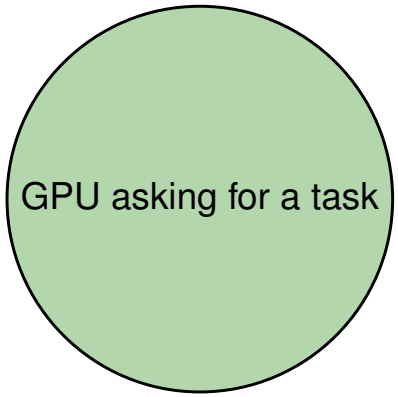
- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**

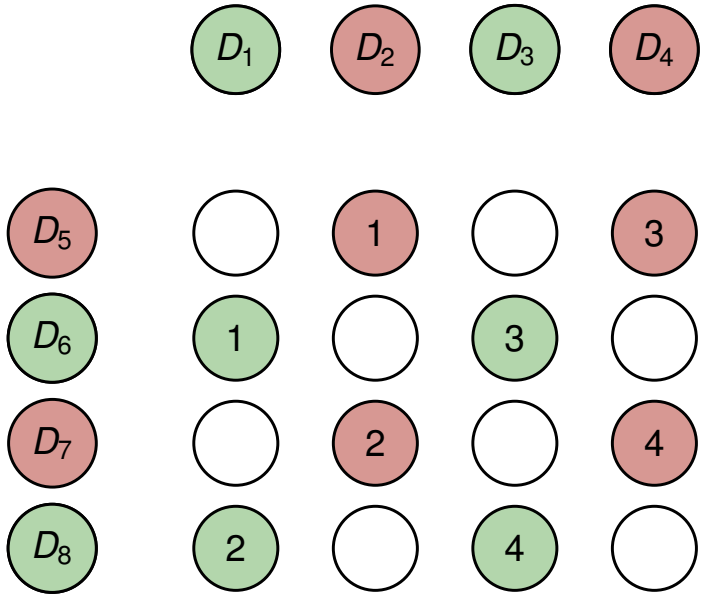




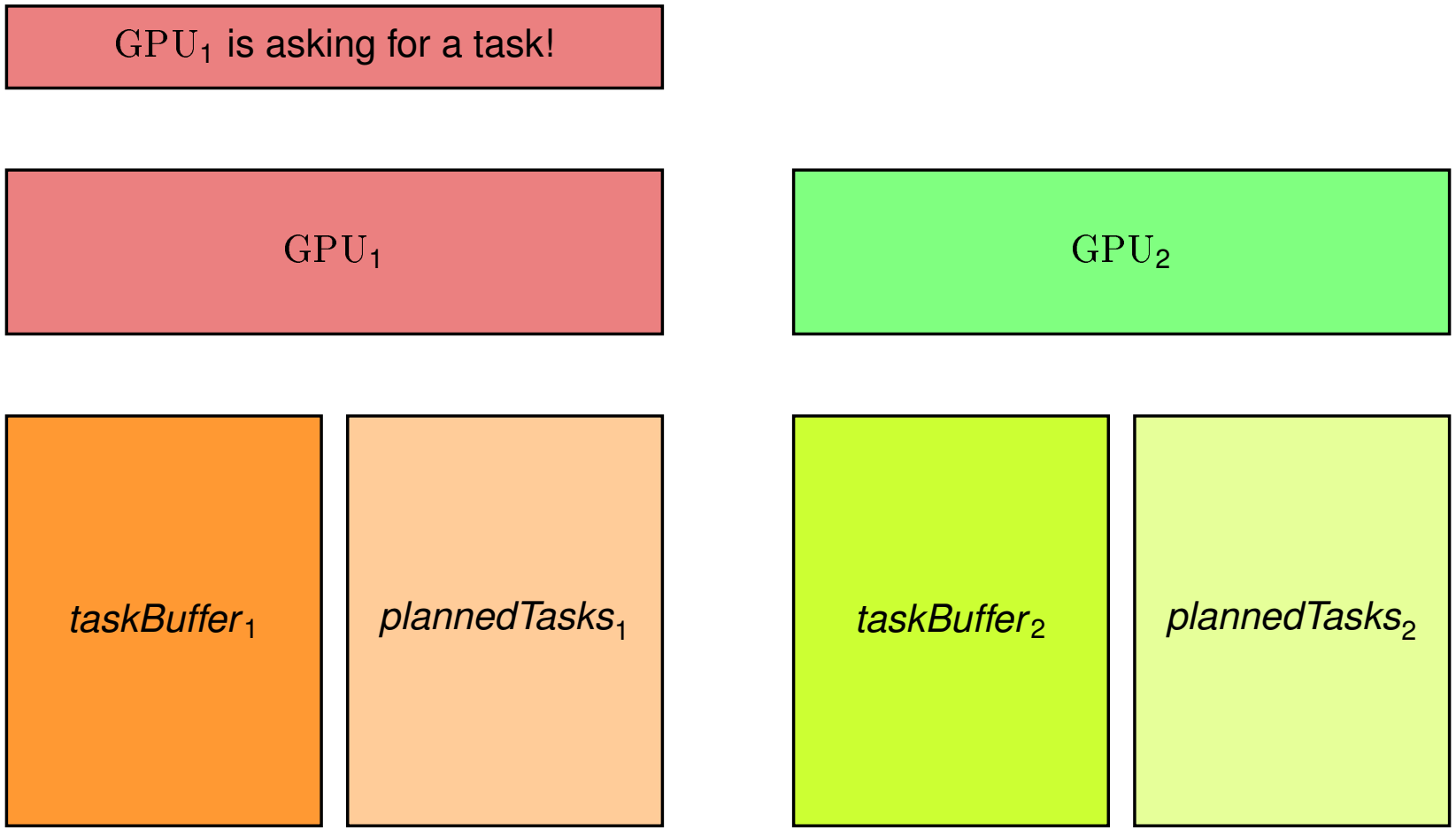
Data-Aware Reactive Task Scheduling: DARTS

- **Consider data movement before task allocation**
- **Perform as many tasks as possible with the data at hand**

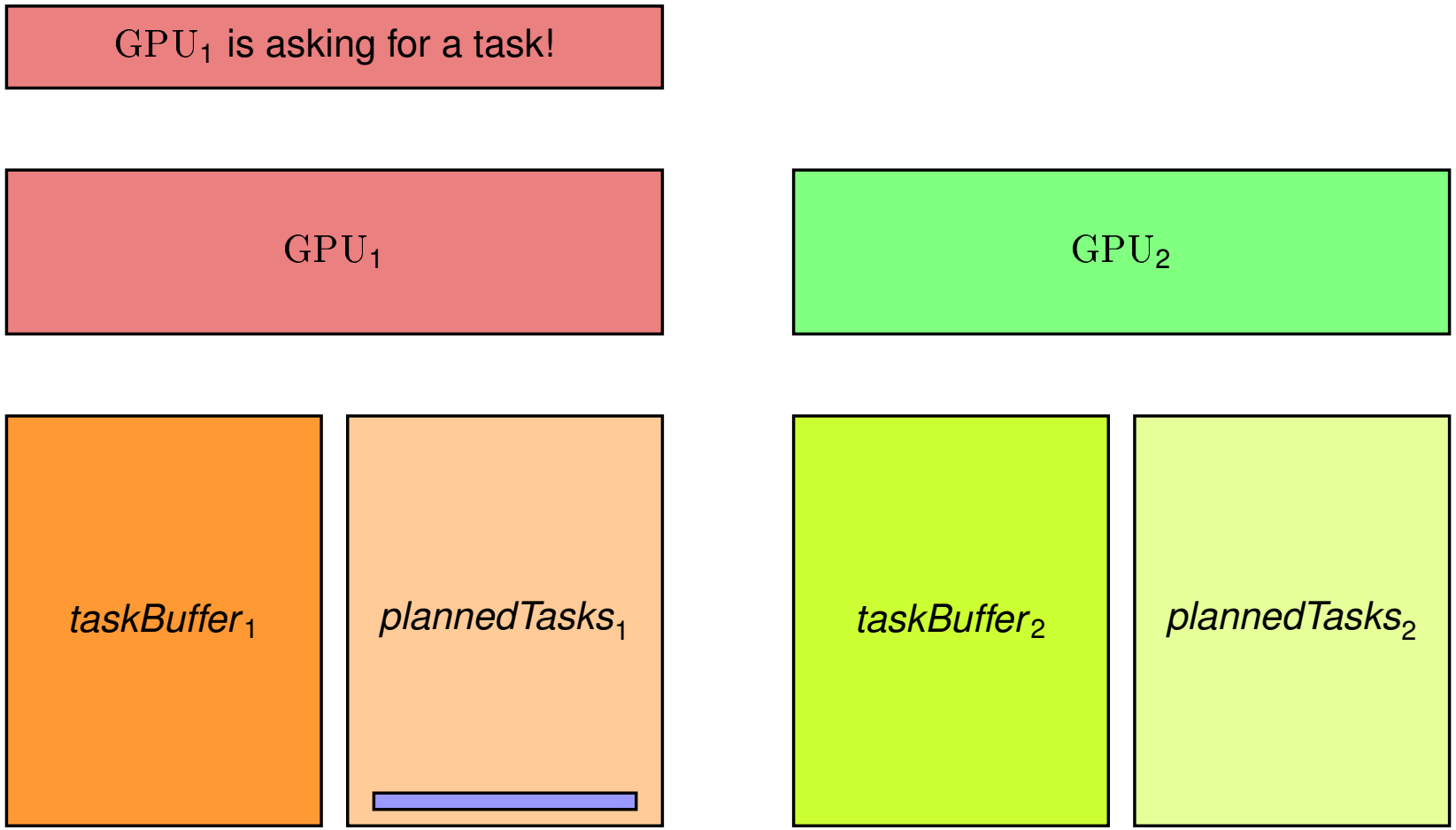




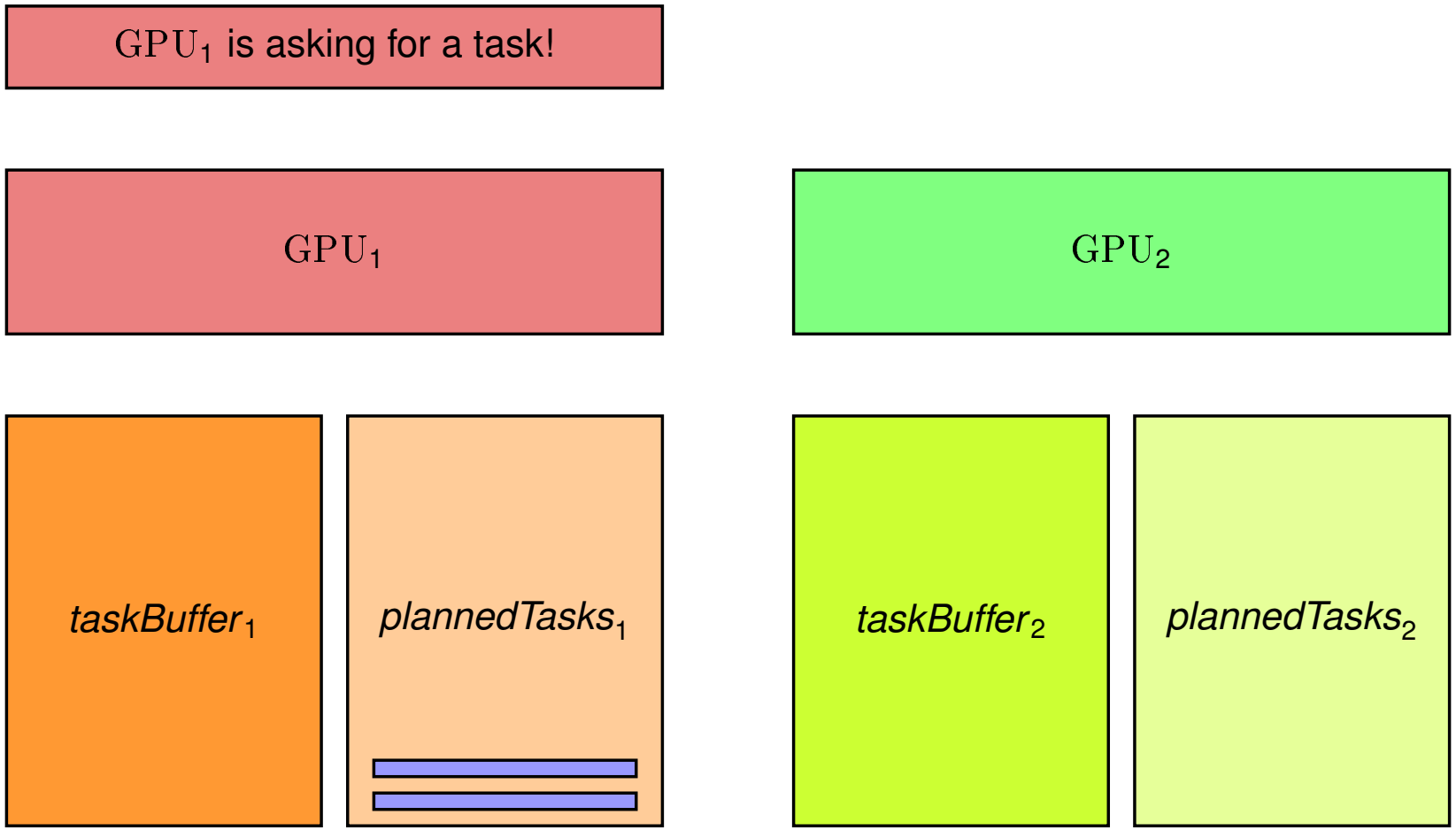
Flow of task distribution



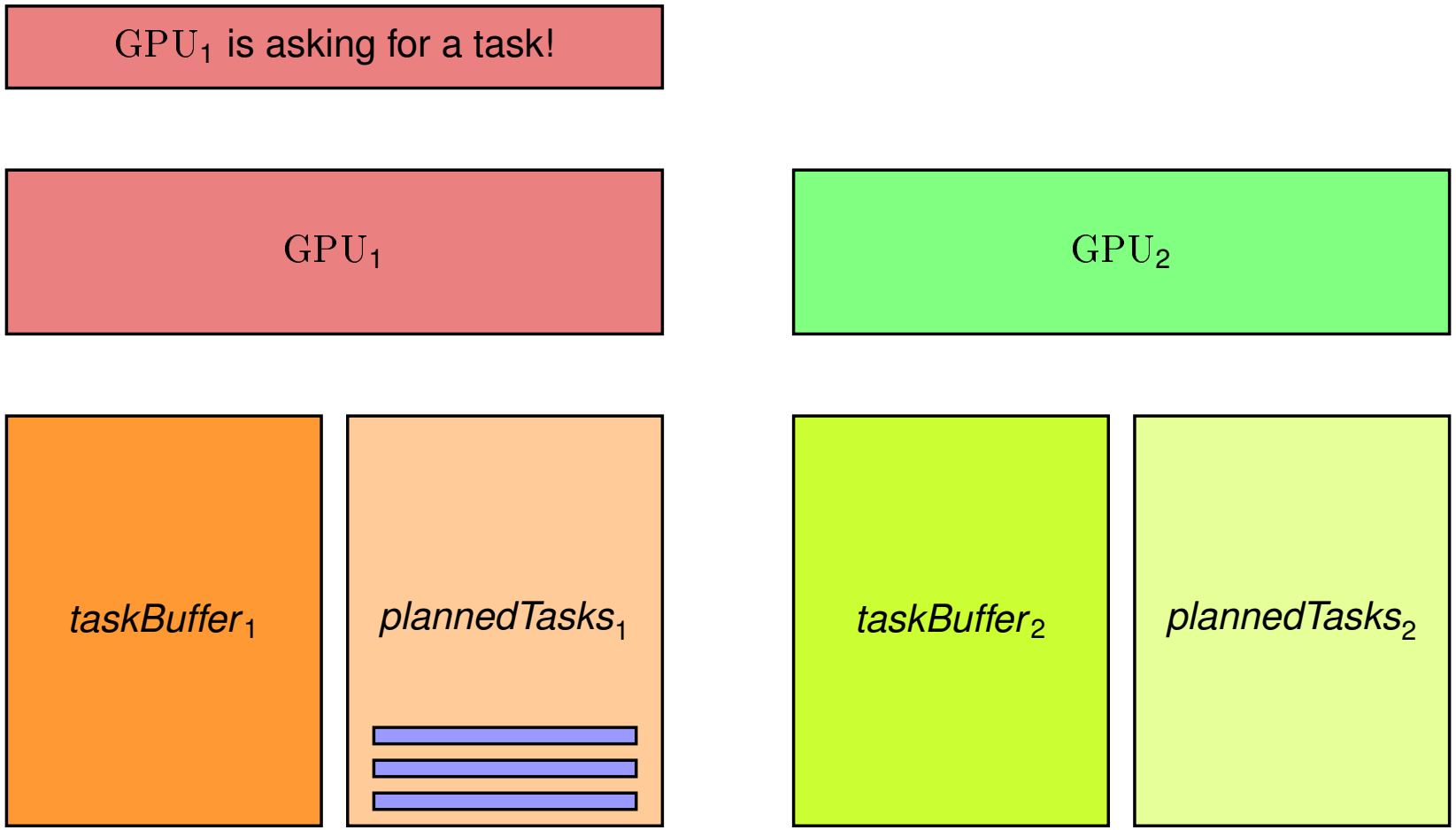
Flow of task distribution



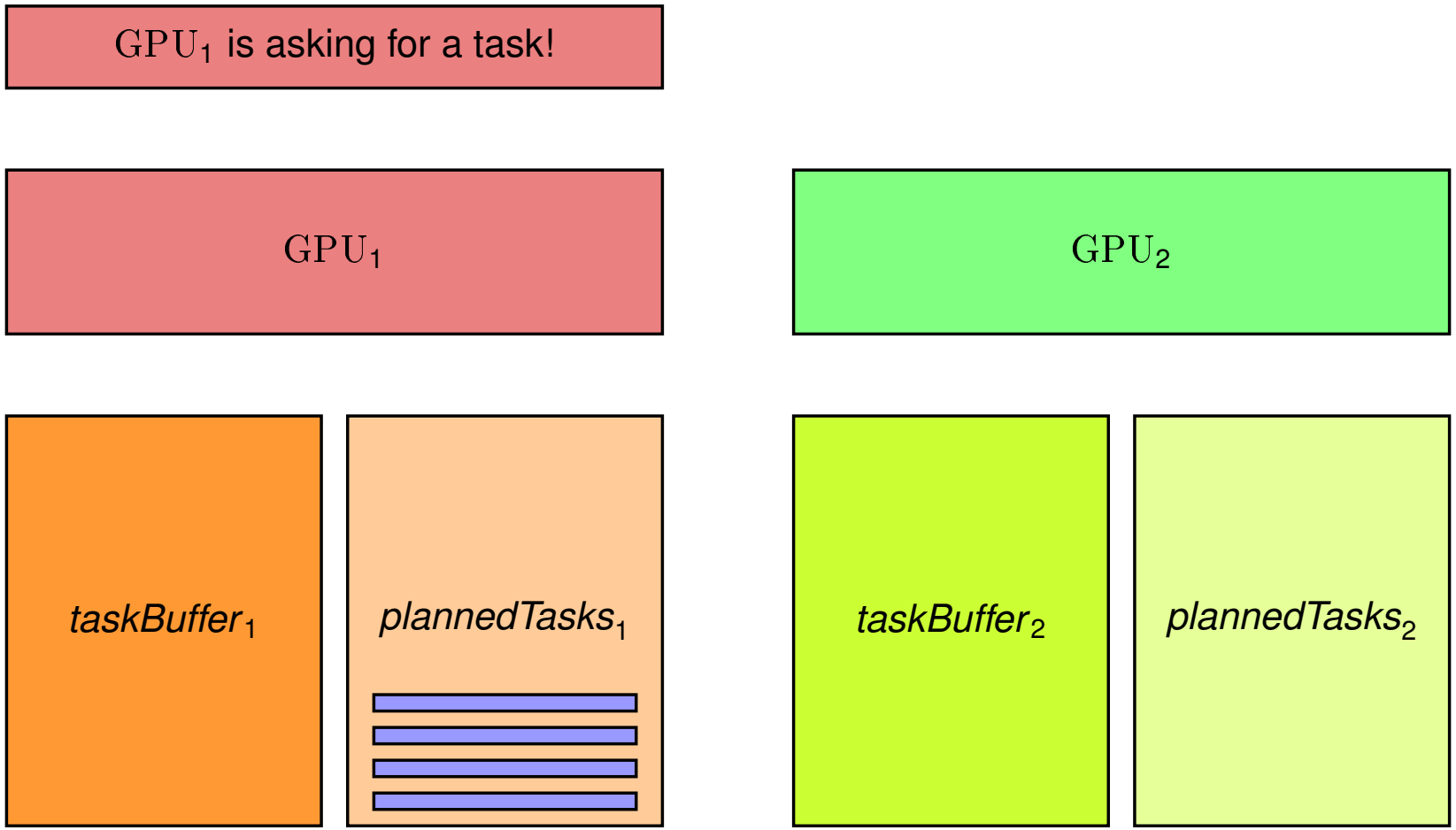
Flow of task distribution



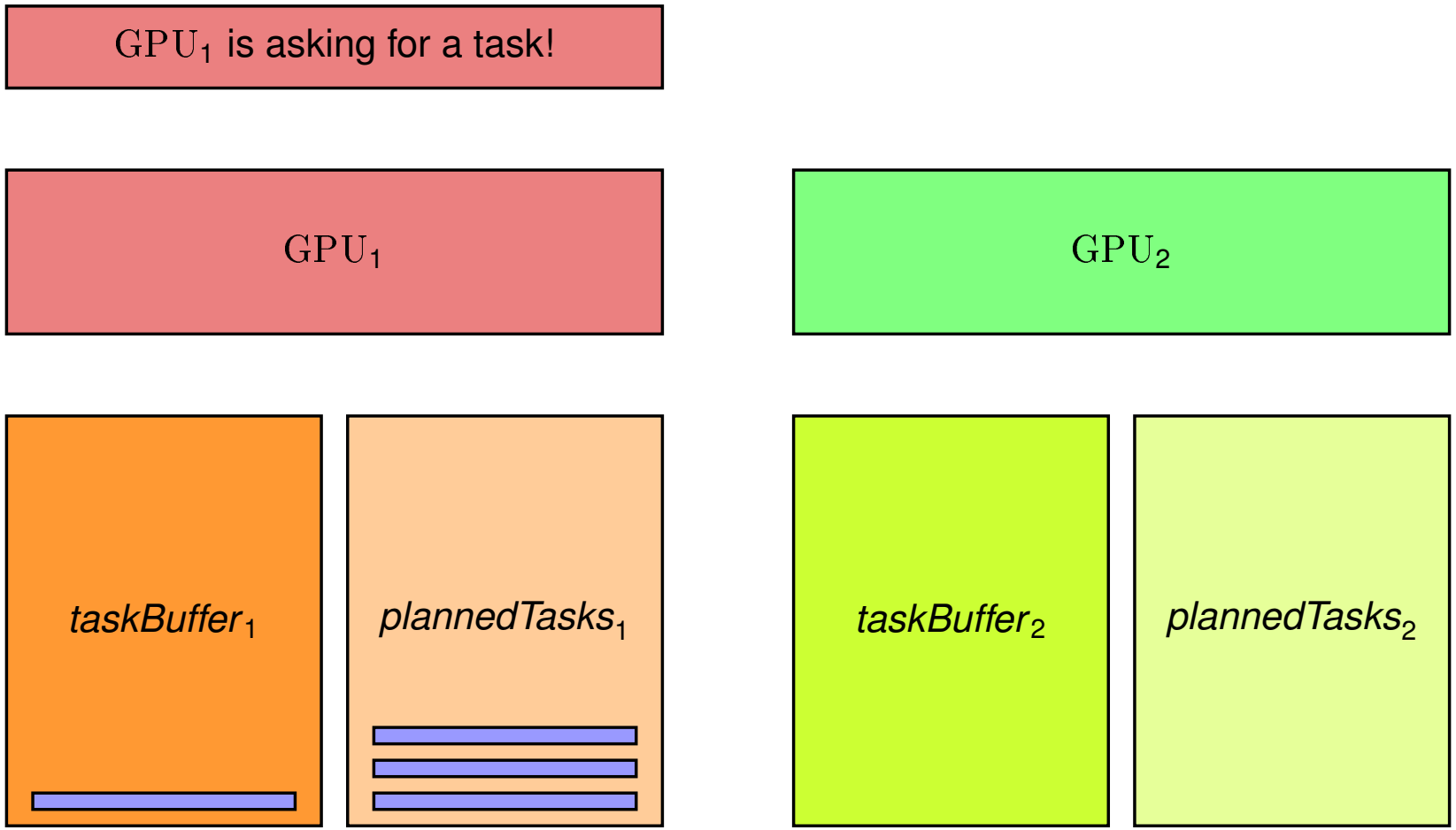
Flow of task distribution



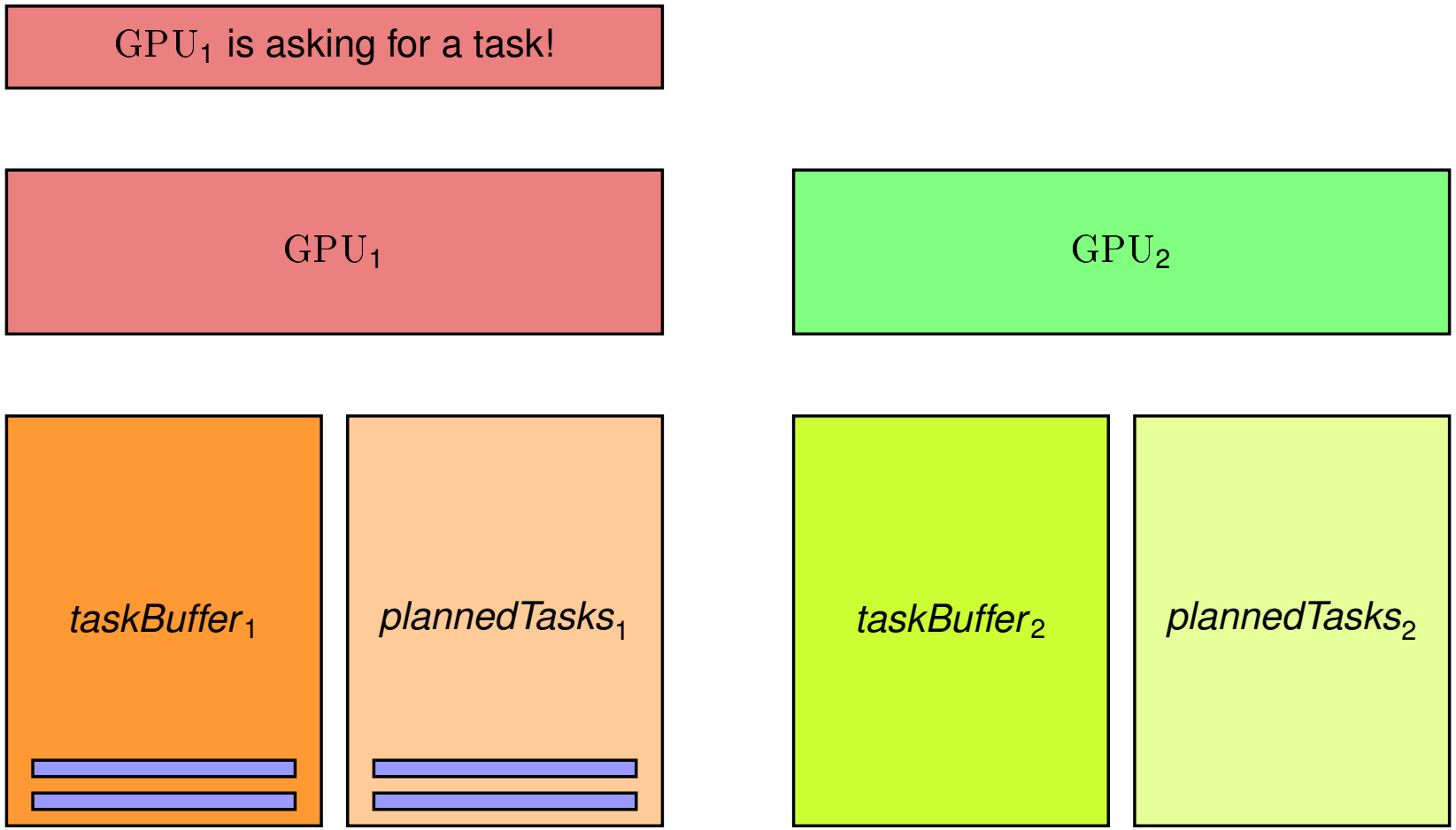
Flow of task distribution



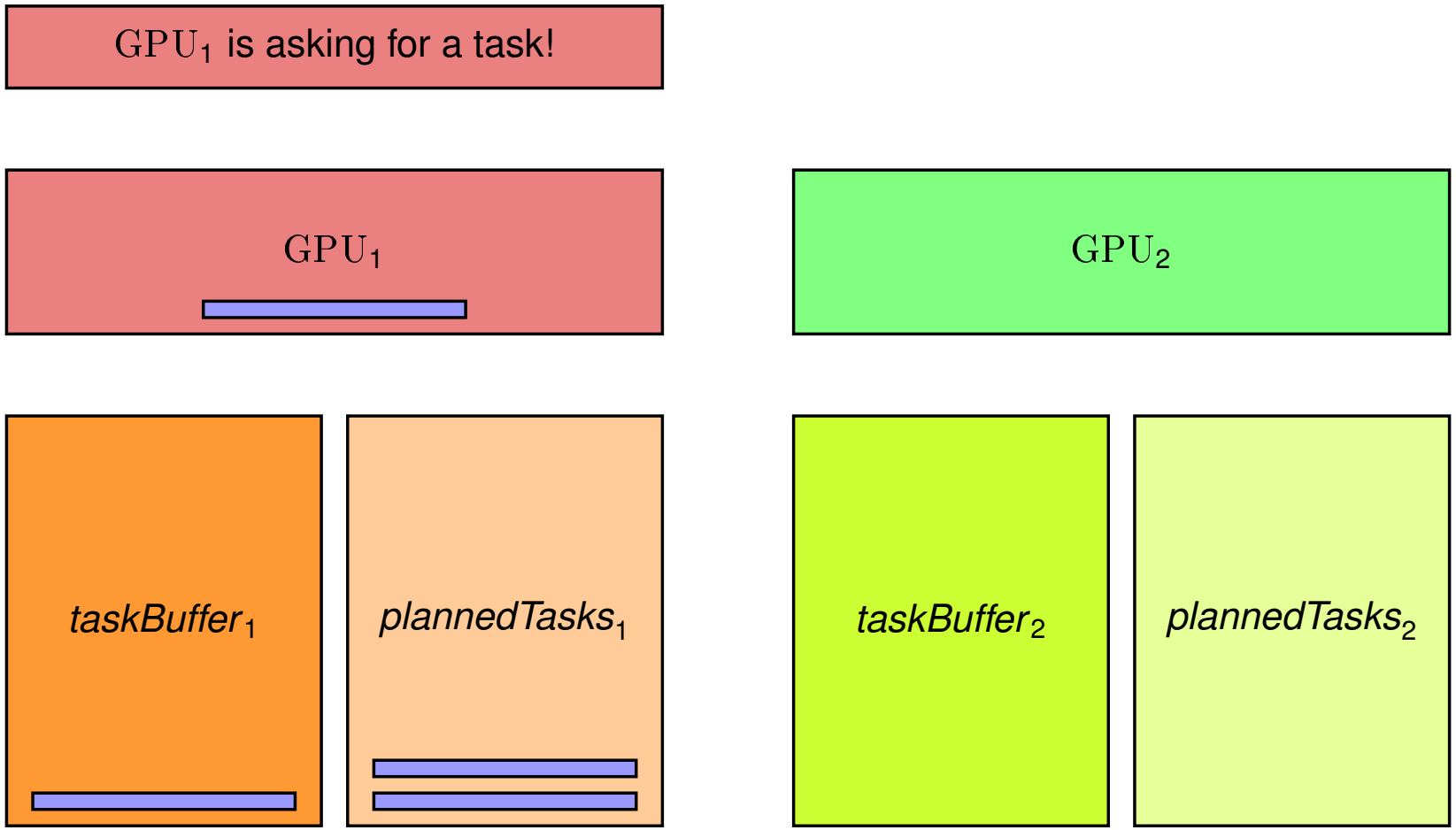
Flow of task distribution



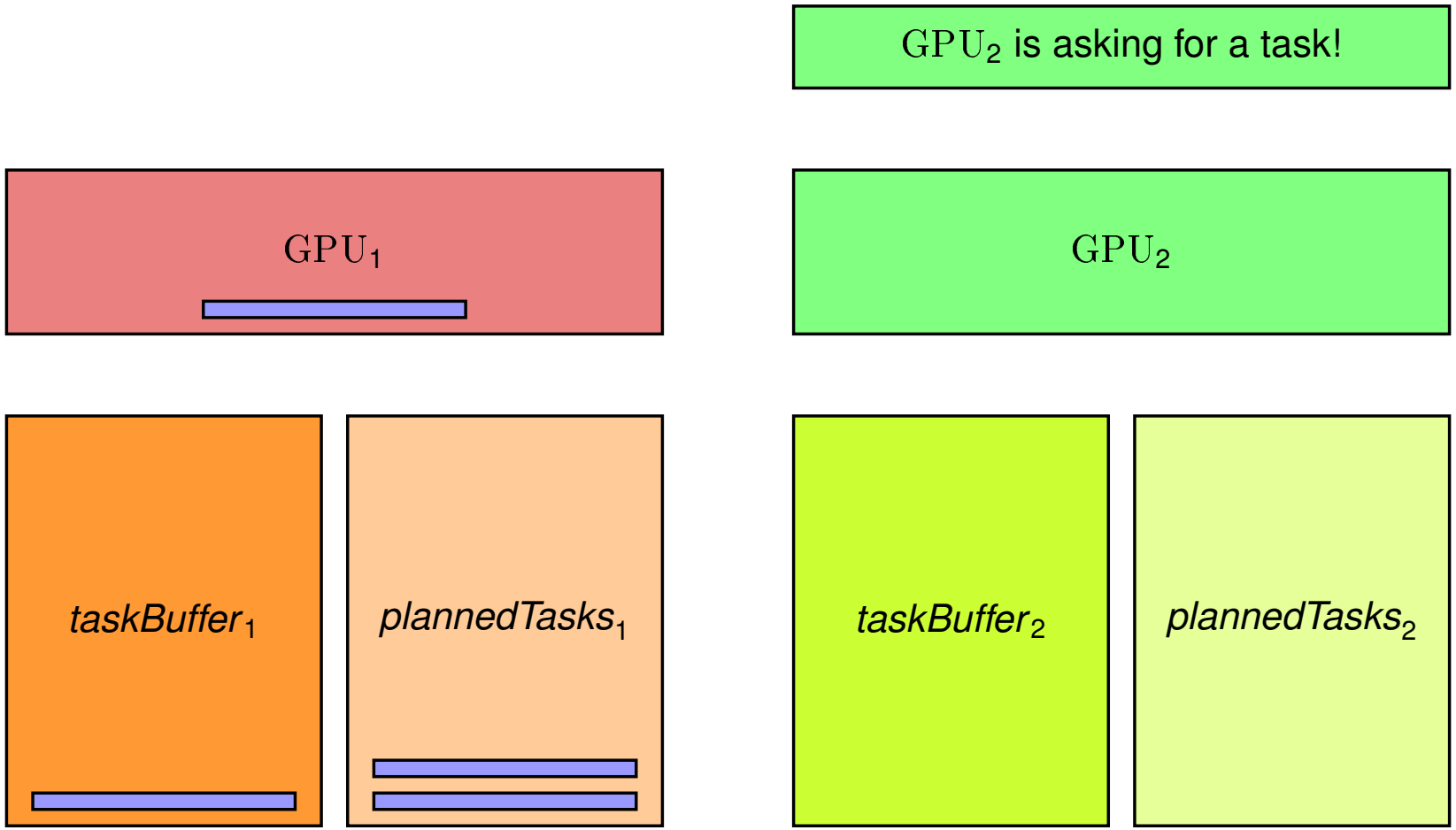
Flow of task distribution



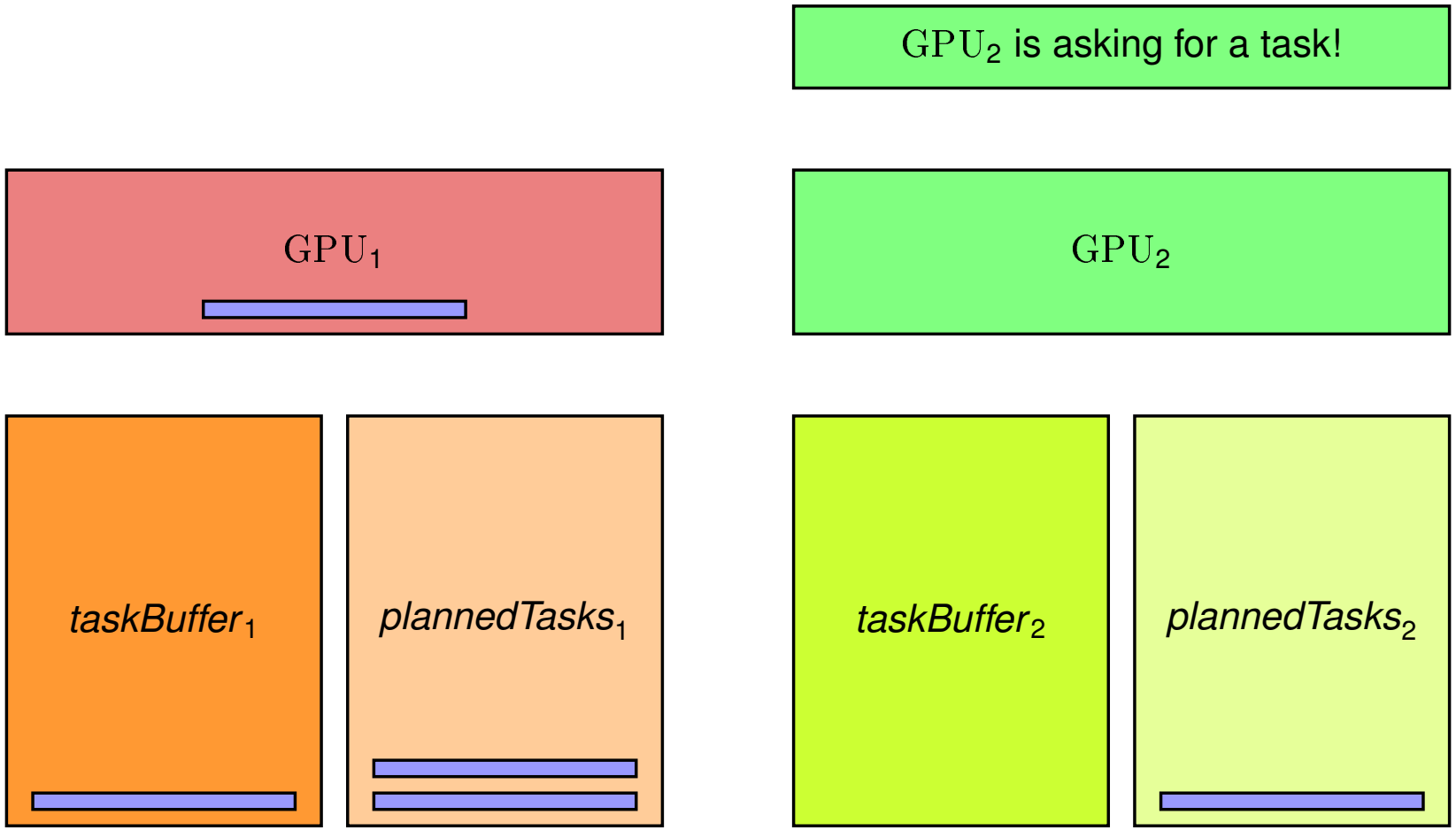
Flow of task distribution



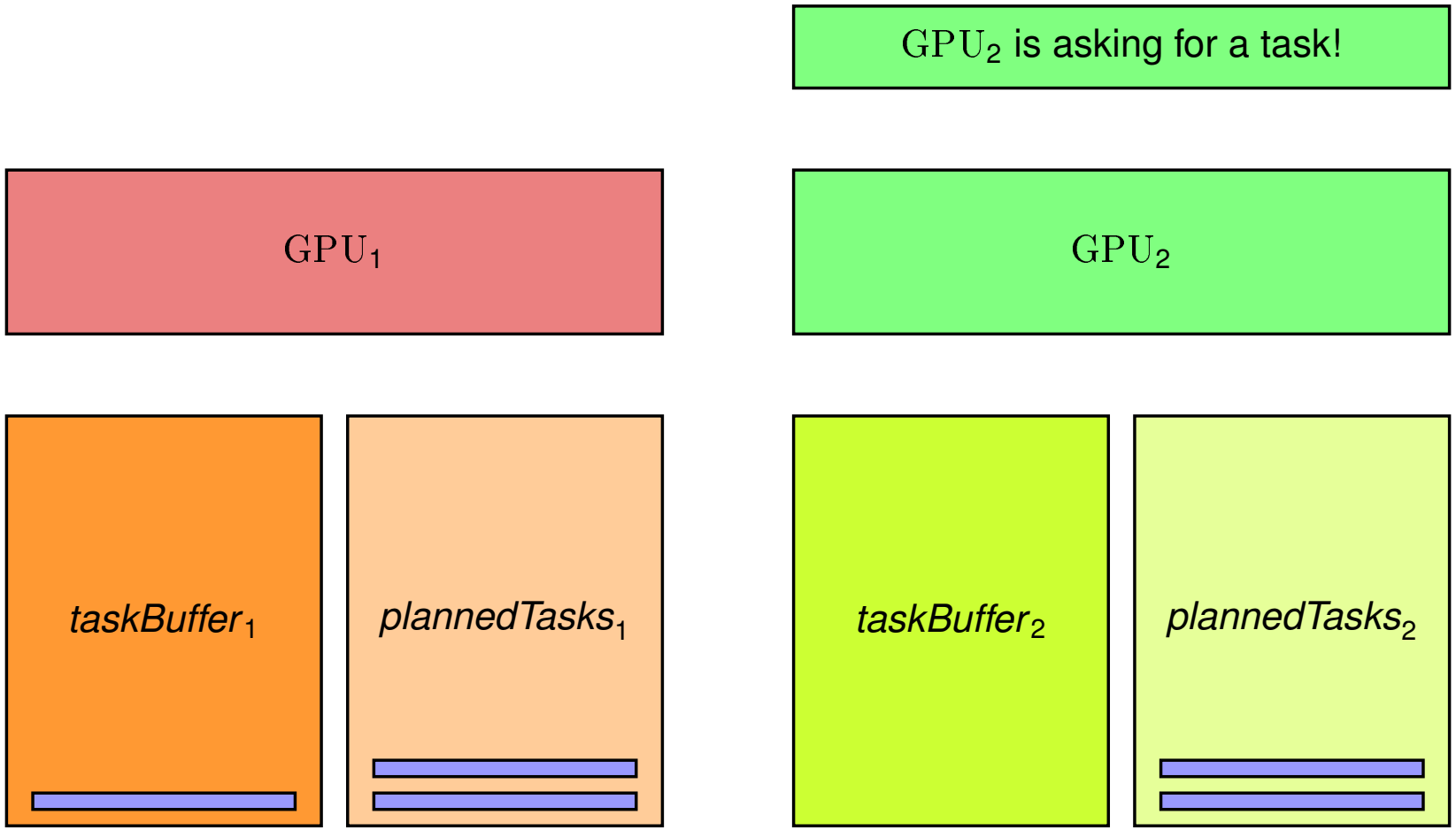
Flow of task distribution



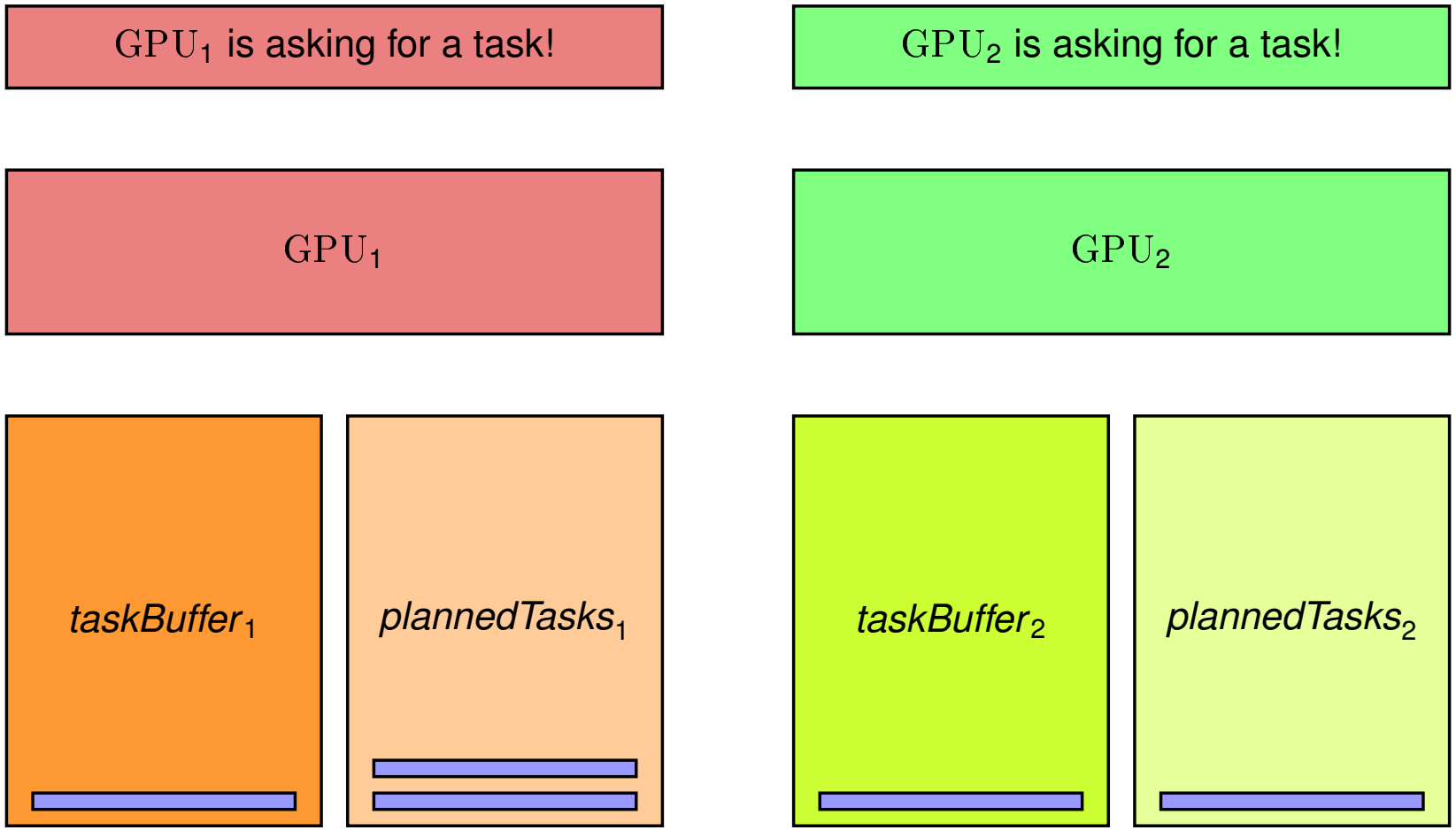
Flow of task distribution



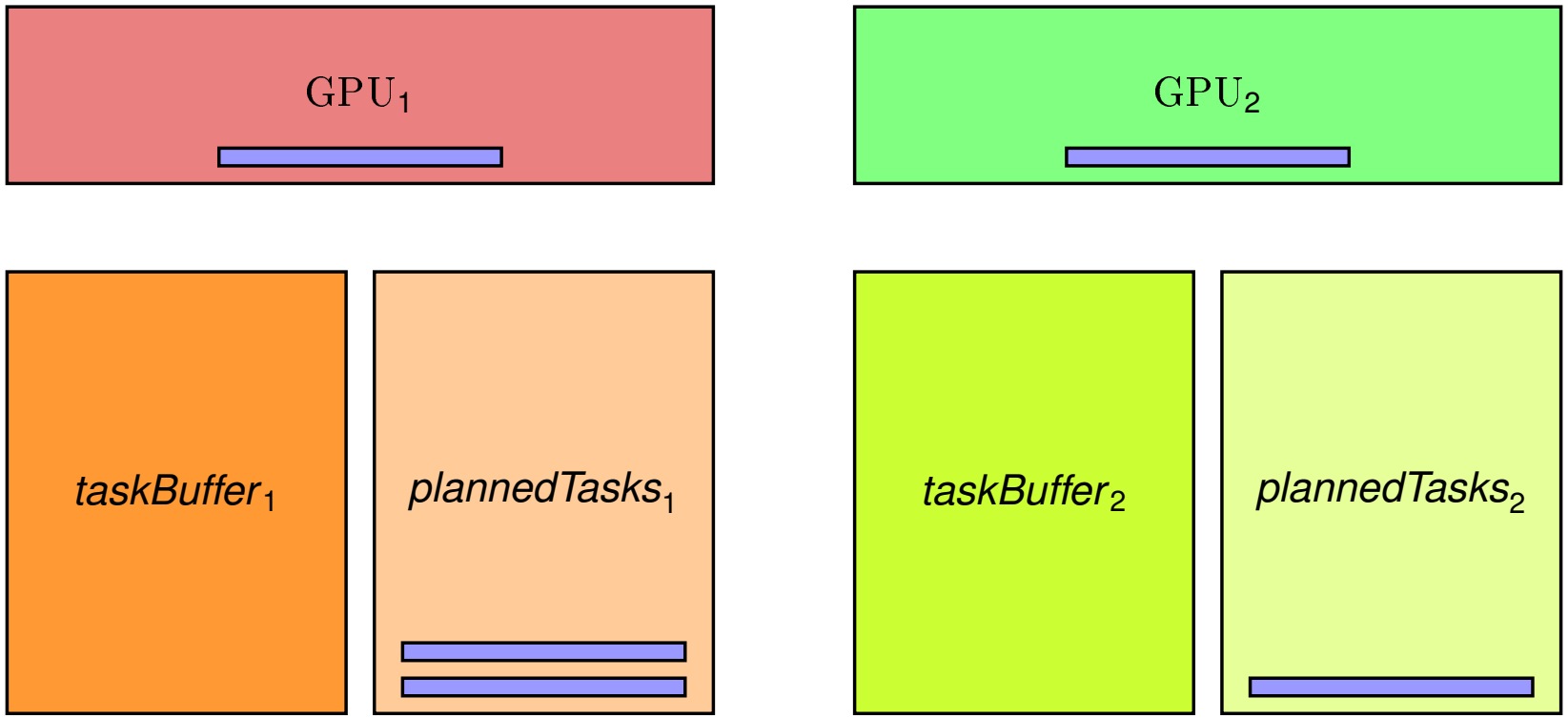
Flow of task distribution



Flow of task distribution



Flow of task distribution



Settings and buffers of task

Initialization

- Each GPU has a randomized data set: $dataNotInMem_k$ (which initially contains all data)
- Task set is randomized

Each GPU has 2 buffers of task:

1. $plannedTasks_k$: Task that have been reserved for a later allocation
2. $taskBuffer_k$: Task popped from $plannedTasks_k$ for execution on GPU_k. Cannot be changed any more

When $taskBuffer_k$ is empty GPU_k will pop a task from $plannedTasks_k$.
When $plannedTasks_k$ is empty, GPU_k call our algorithm to fill $plannedTasks_k$.

Strategy

Filling $plannedTasks_k$

Everything is distributed. The scheduling is done by GPU_k when it needs new tasks.

- Find $D_{opt} \in dataNotInMem_k$ such that the number of tasks depending on D_{opt} and on other data already in memory is maximum (if there is a tie: choose data with most total unprocessed task)
- $plannedTasks_k \leftarrow$ set of unprocessed tasks depending only on D_{opt} and on other data already in memory
- Remove D_{opt} from $dataNotInMem_k$
- If D_{opt} does not allow to process at least one "free" task -> Select a random unprocessed task T

Eviction and optimizations

Our eviction policy: LUF (Least Used in the Future)

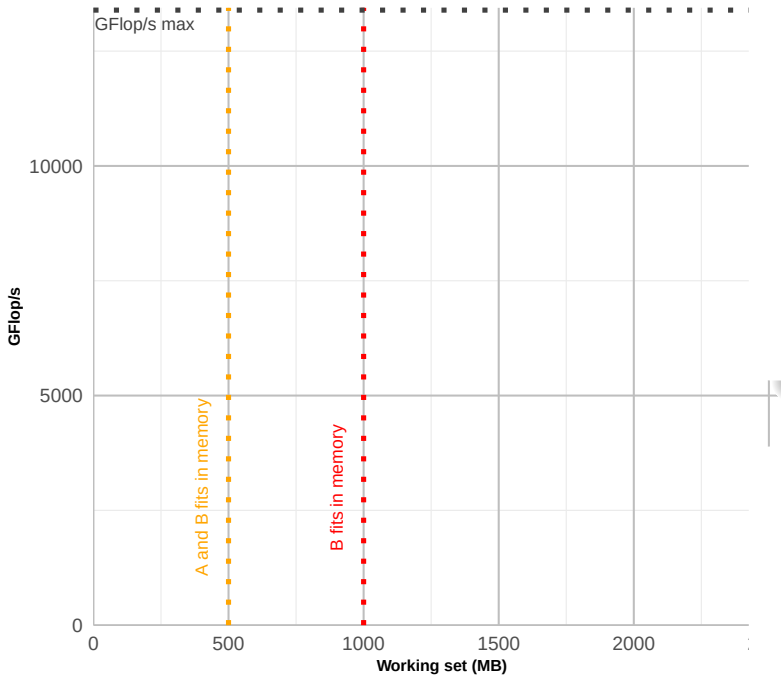
1. If possile, try to evict data not useful for any task in $taskBuffer_k$ and used by a minimal number of tasks in $plannedTasks_k$
2. Else, apply Belady's rule on tasks already allocated
3. Then, update $plannedTasks_k$

Improvements

- *3inputs*: Extension of DARTS for tasks with an heterogenous number of data per task
- *OPTI*: We stop the search a soon as we find a data allowing to compute at least one task

ENS DE LYON université de BORDEAUX

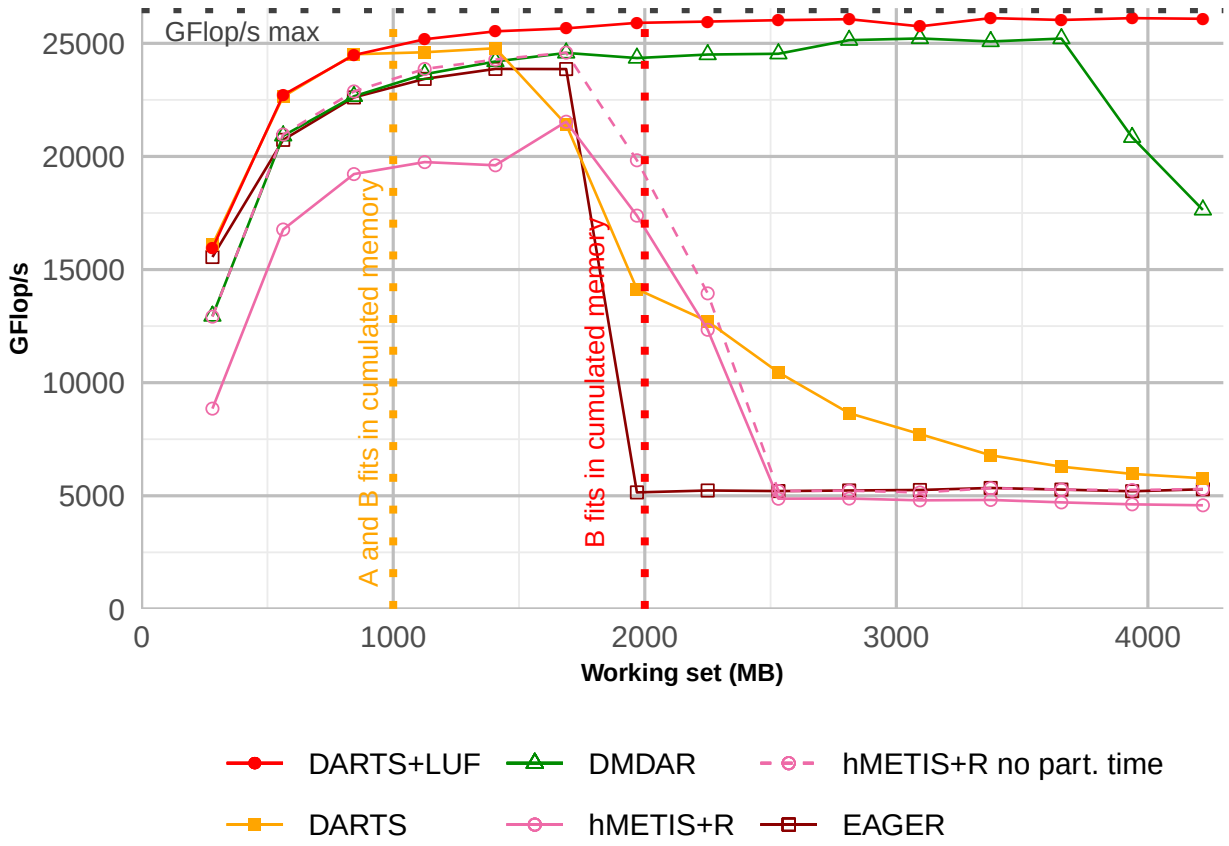
Experimental settings



- Tesla V100 GPUs
- Simulations on Simgrid
- PCI bandwidth not limited (12000 MB/s)
- GPU memory limited to 500 $MB \rightarrow$ To better distinguish performance on small datasets

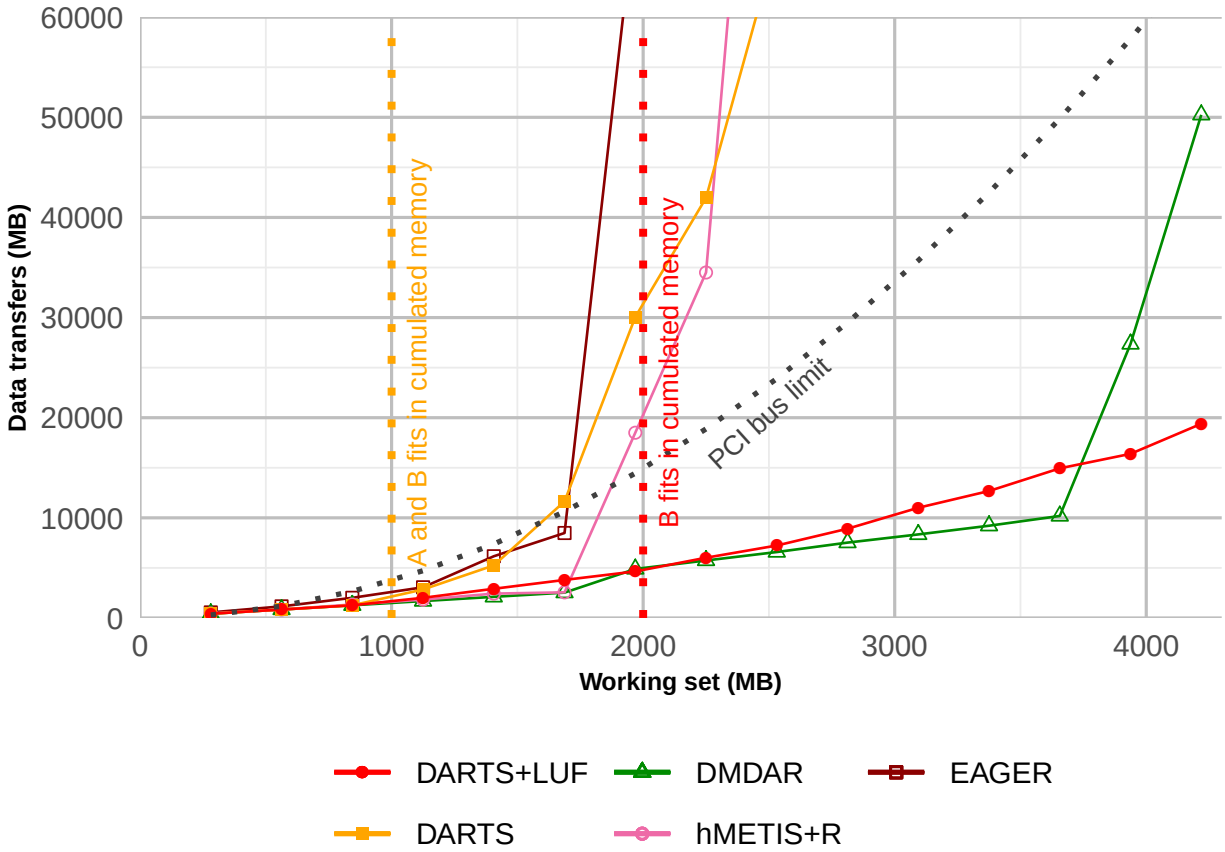
- 2D, 3D, sparse matrix multiplication
- Task set from the Cholesky decomposition (without dependencies)

2D matrix multiplication with 2 Tesla V100 GPUs



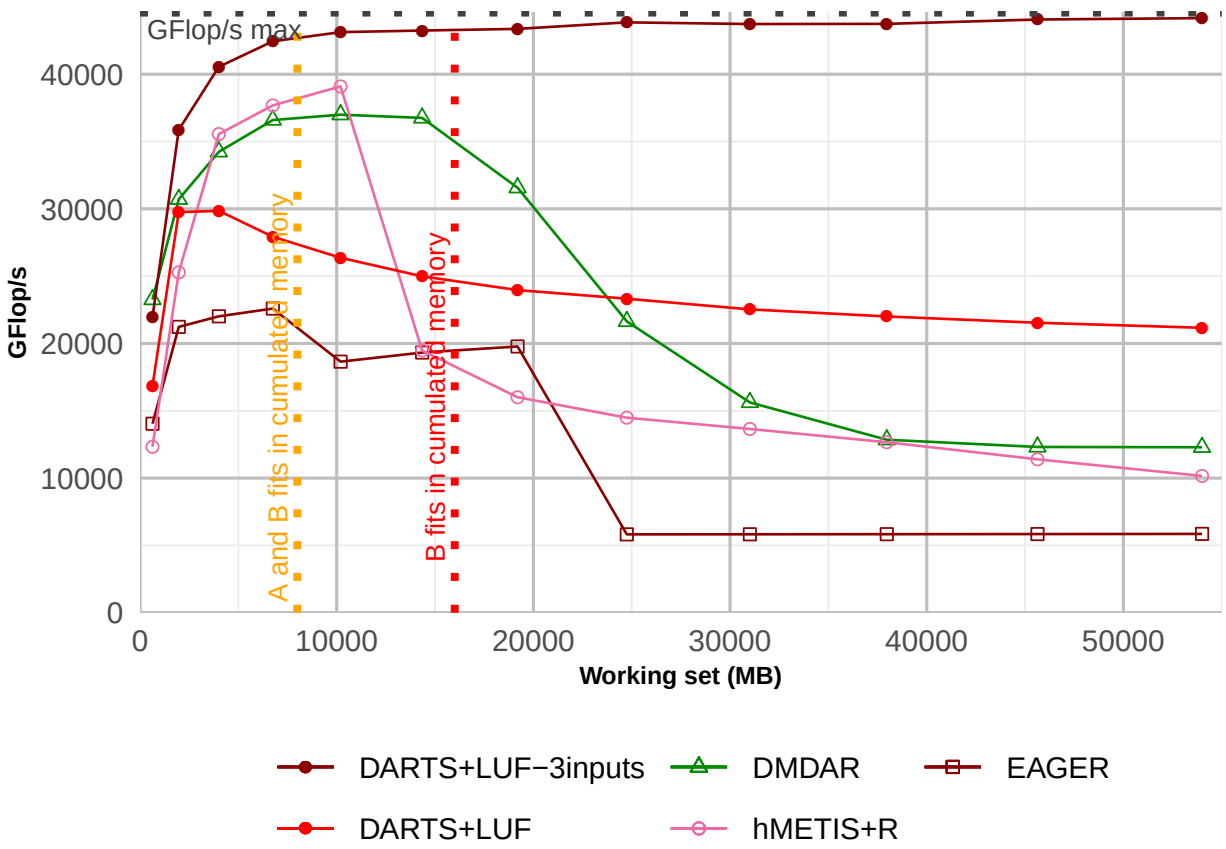
- EAGER, hMETIS & DARTS: Pathological matrix size after the red line
- DMDAR: Conflict between prefetch and eviction

Amount of data transfers on the 2D matrix multiplication with 2 Tesla V100 GPUs



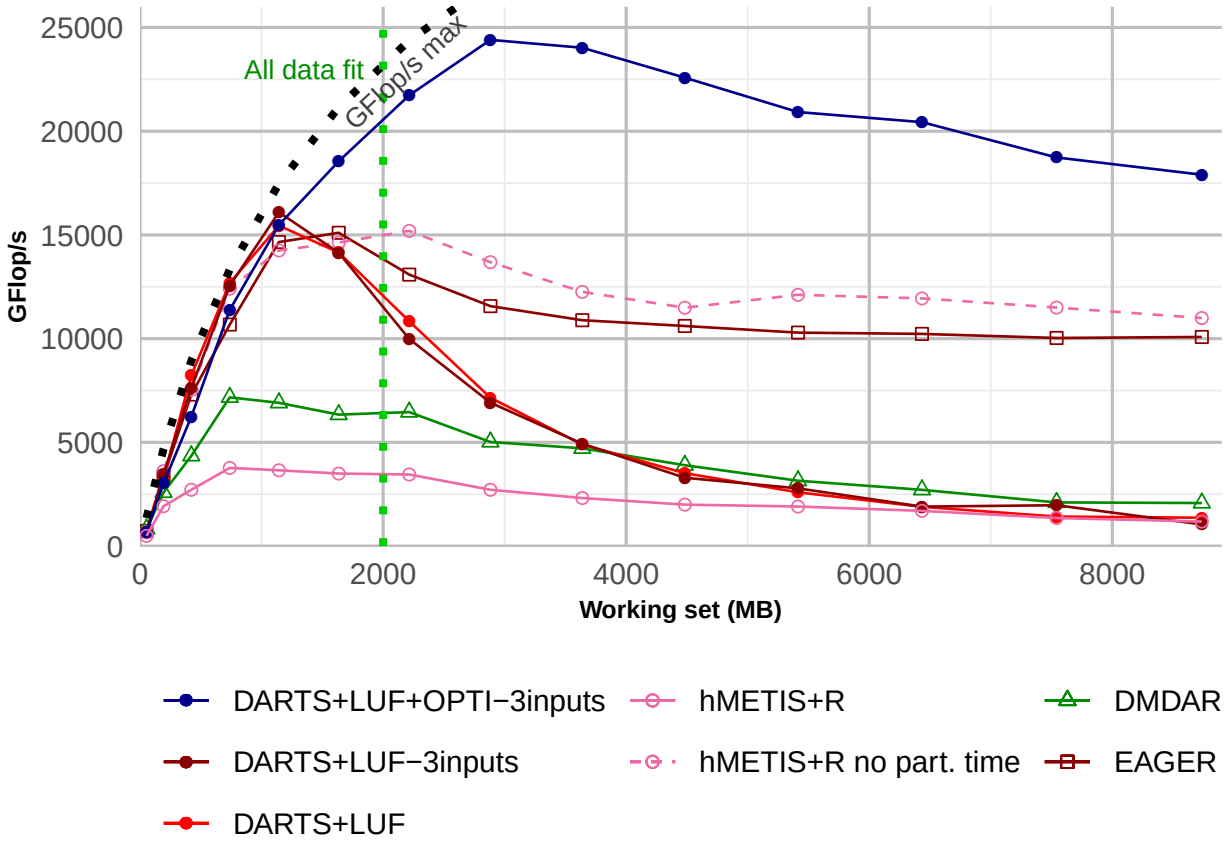
- DARTS outperform DMDAR even with more data transfers → **DARTS is better at overlapping communications and computations**

3D matrix multiplication in simulation with the performance models of 4 Tesla V100 GPUs



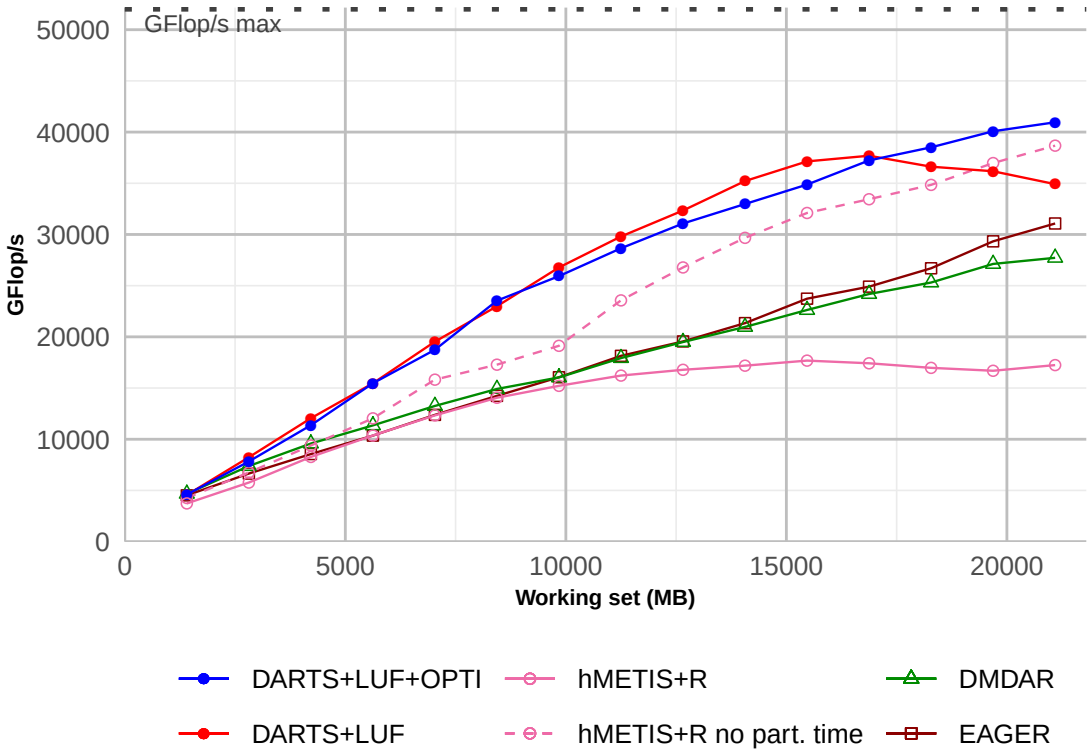
- The *3inputs* improvement is crucial

Tasks from the Cholesky decomposition with 4 GPUs



- *OPTI* reduce scheduling time which improve performance
- DMDAR suffer from a large scheduling time

Sparse 2D matrix multiplication without memory limitation (32GB by GPU) with 4 Tesla V100 GPUs



- No memory constraint
- DARTS produces a **processing order that best distributes transfers over time**
- hMETIS suffers from an important partitioning cost

Conclusion

Limiting data movements is crucial

→ **New strategy: DARTS**
Focused on data movement

Conclusion on the experiments

Tested on 2D, 3D, sparse matrix multiplication and Cholesky decomposition → DARTS achieves very good performance when:

- Memory is not limited
- Memory is a scarce resource

Future works

DARTS's limitations

- Only manage a single MPI node
- No dependencies

Areas for improvement

- Improve computational complexity
- Take inter-GPU communications into account
- Consider tasks with dependencies

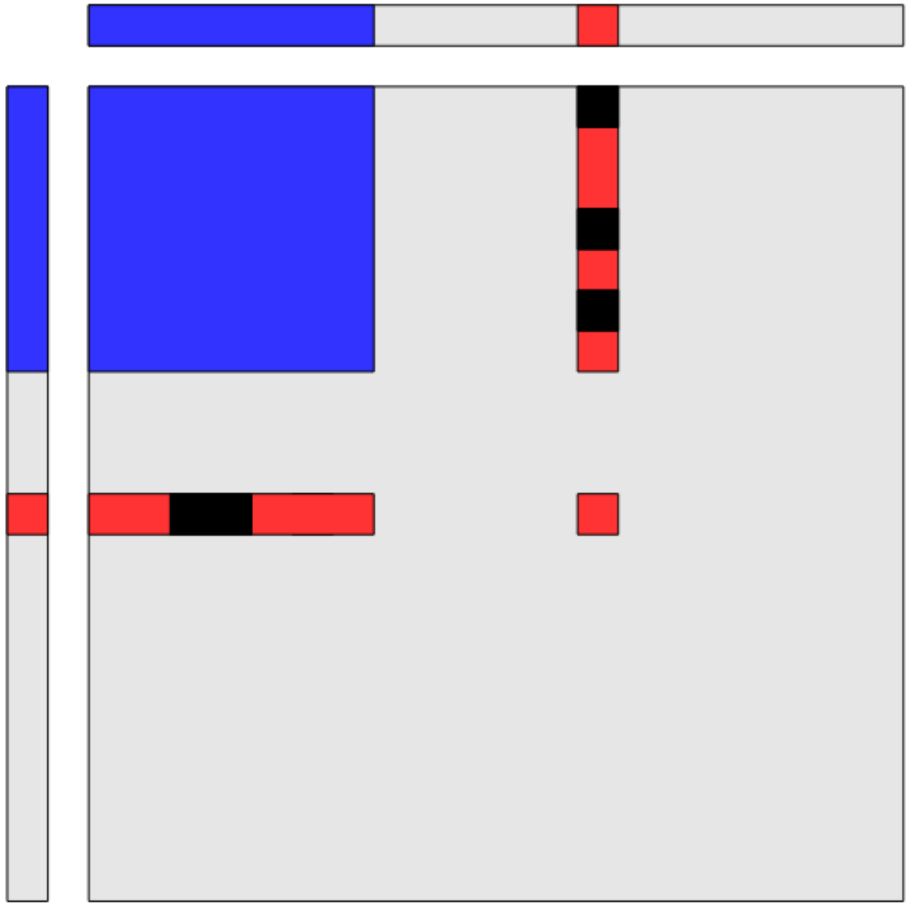
Thank you for your attention!

Contact:

maxime.gonthier@ens-lyon.fr

ENS DE LYON université de BORDEAUX

Joker slide



Source: Analysis of Dynamic Scheduling Strategies for Matrix Multiplication on Heterogeneous Platforms - Marchal - Beaumont

ENS DE LYON université de BORDEAUX