The StarPU Runtime System

Part. 2 – Mastering StarPU

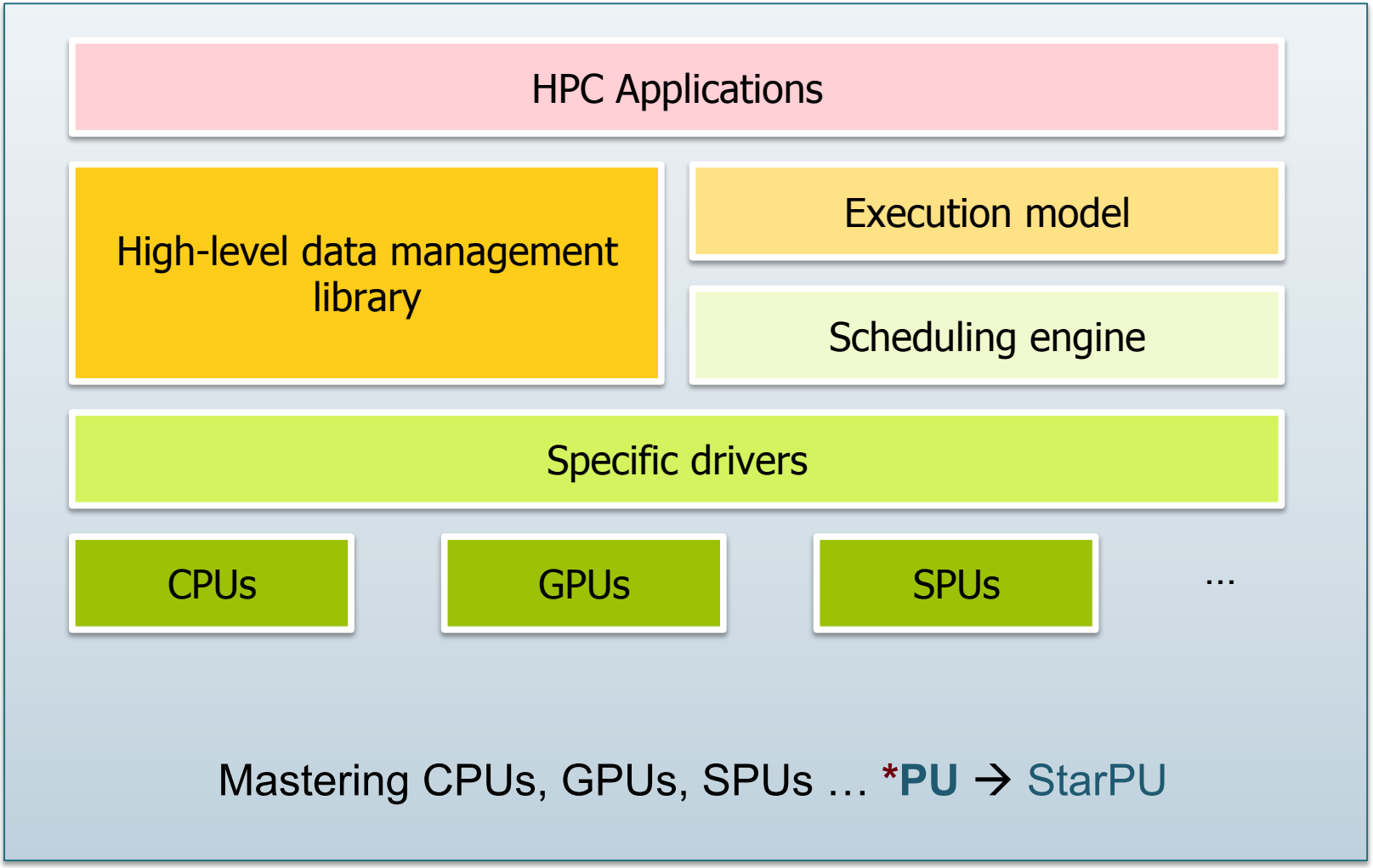
Team Storm – *Olivier Aumage*
Inria – LaBRI, in cooperation
with La Maison de la Simulation

Contents

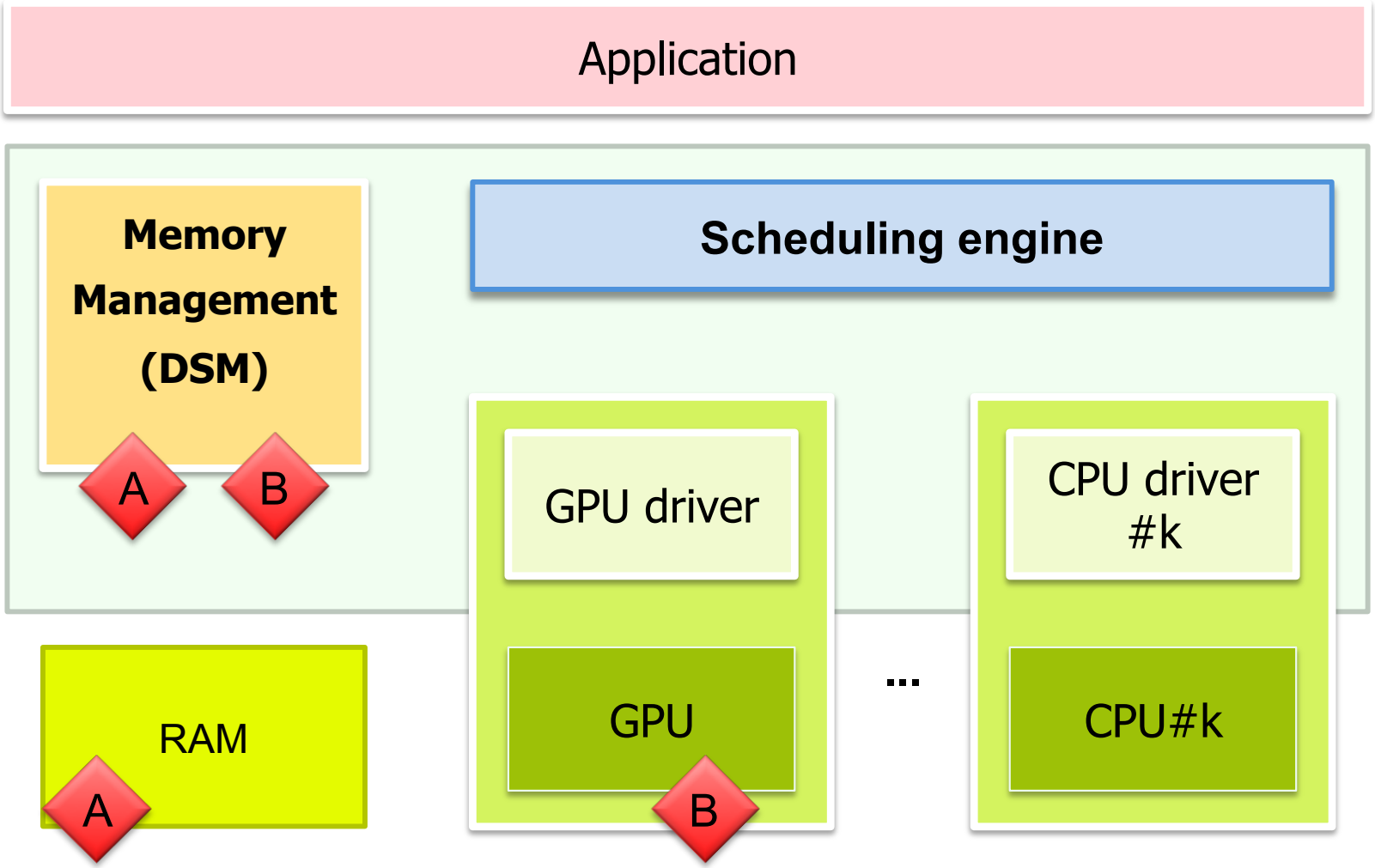
1

StarPU Internals

StarPU Internal Structure

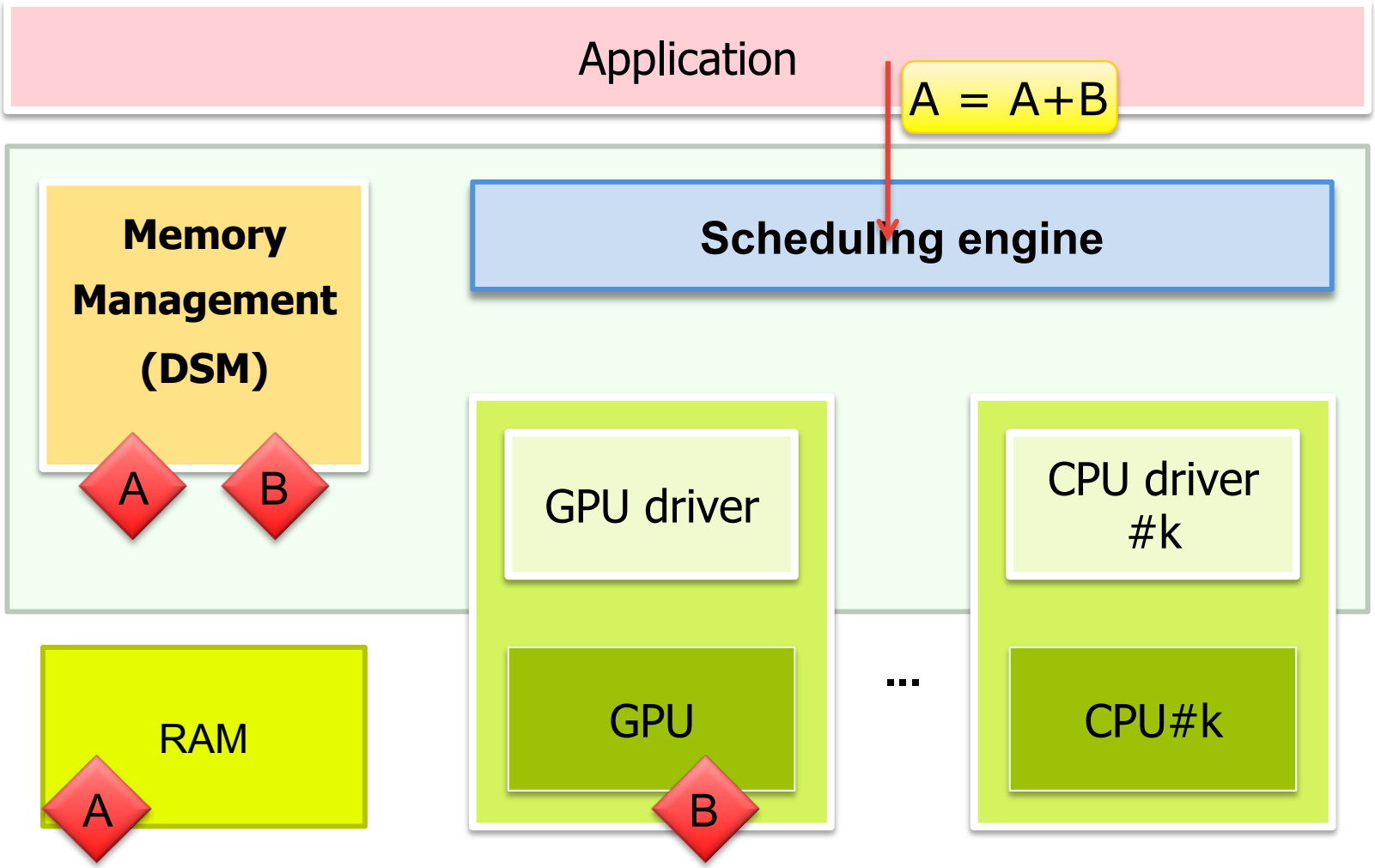


StarPU Internal Functioning



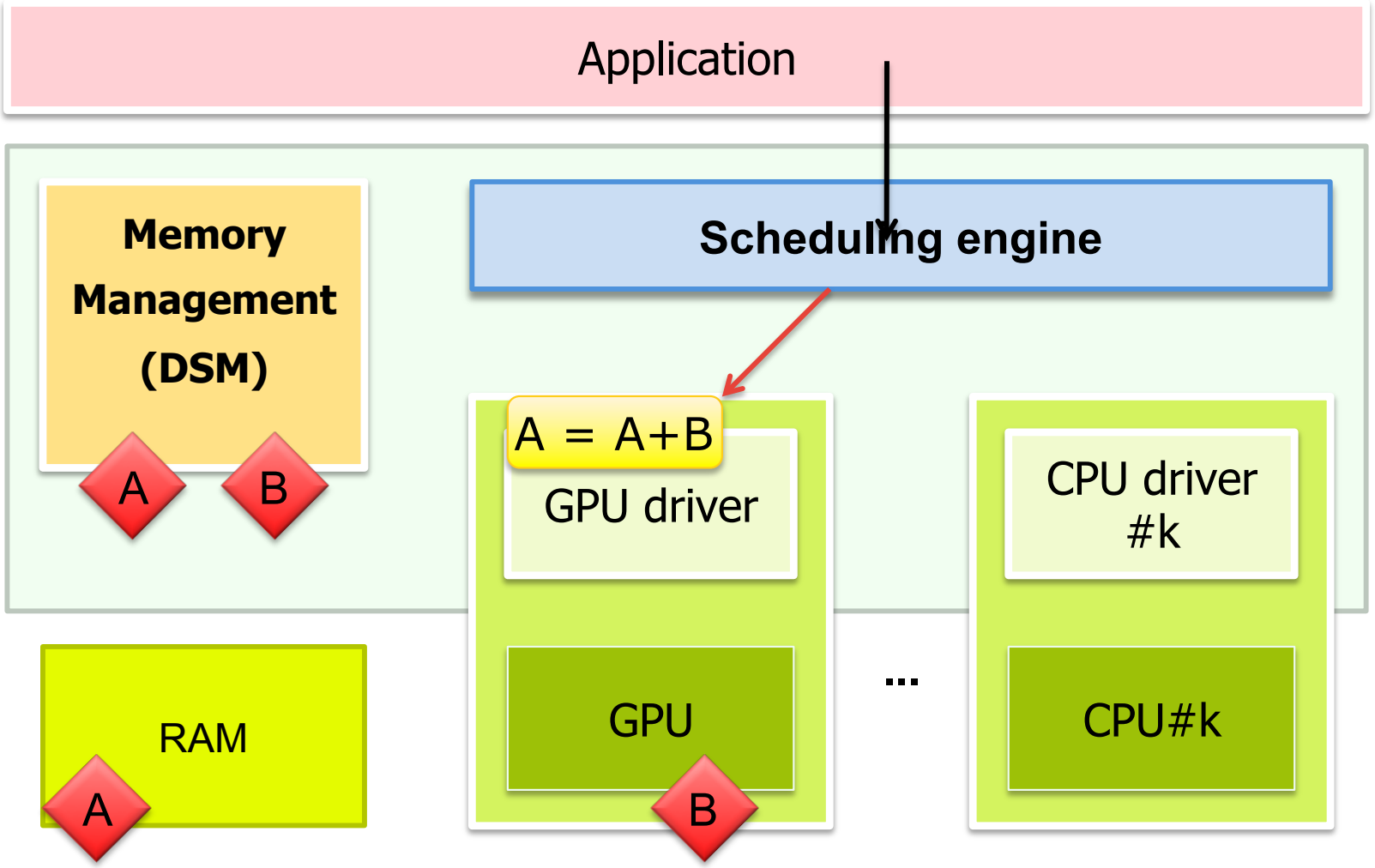
Submit task « A+=B »

StarPU Internal Functioning



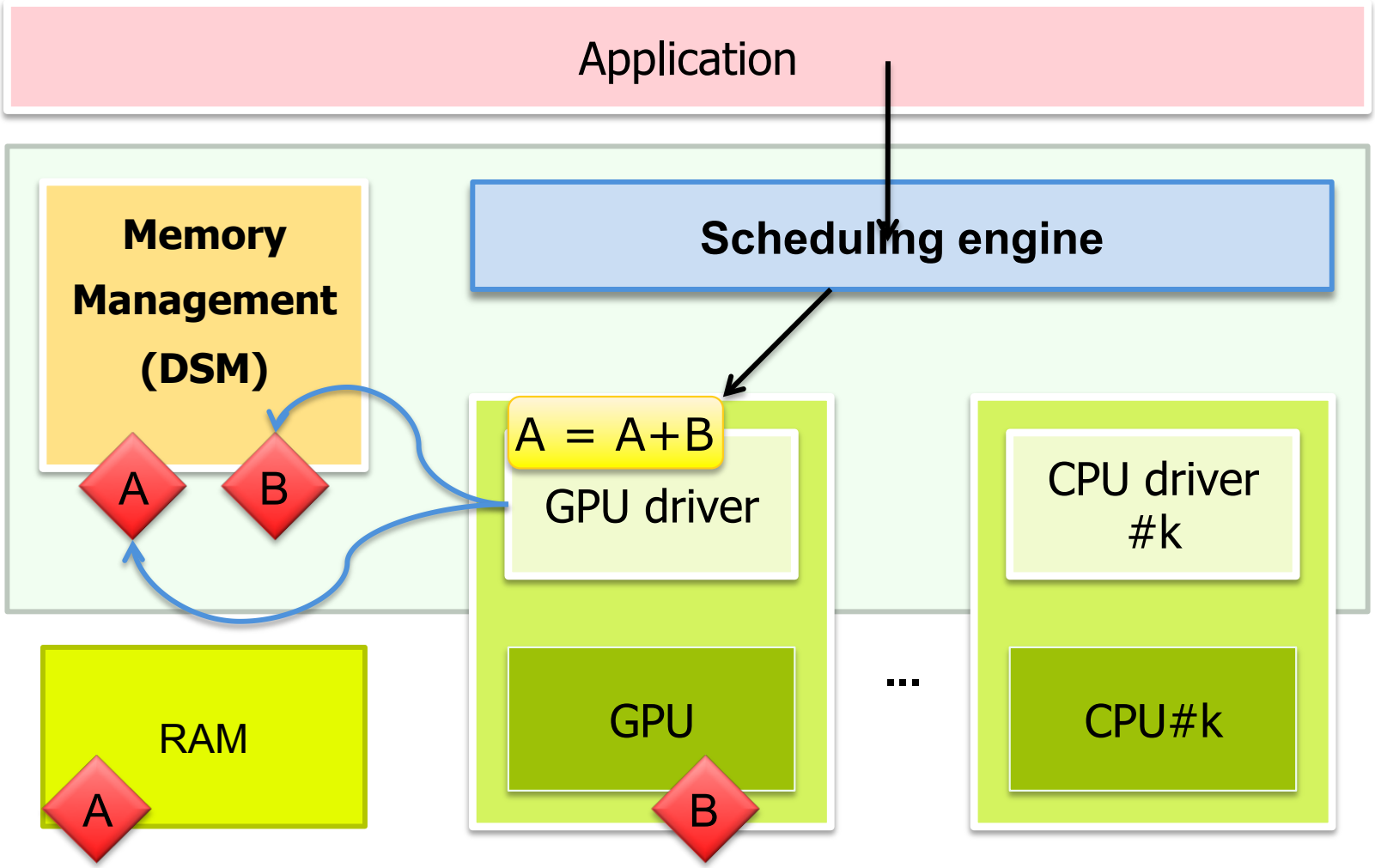
Submit task « A+=B »

StarPU Internal Functioning



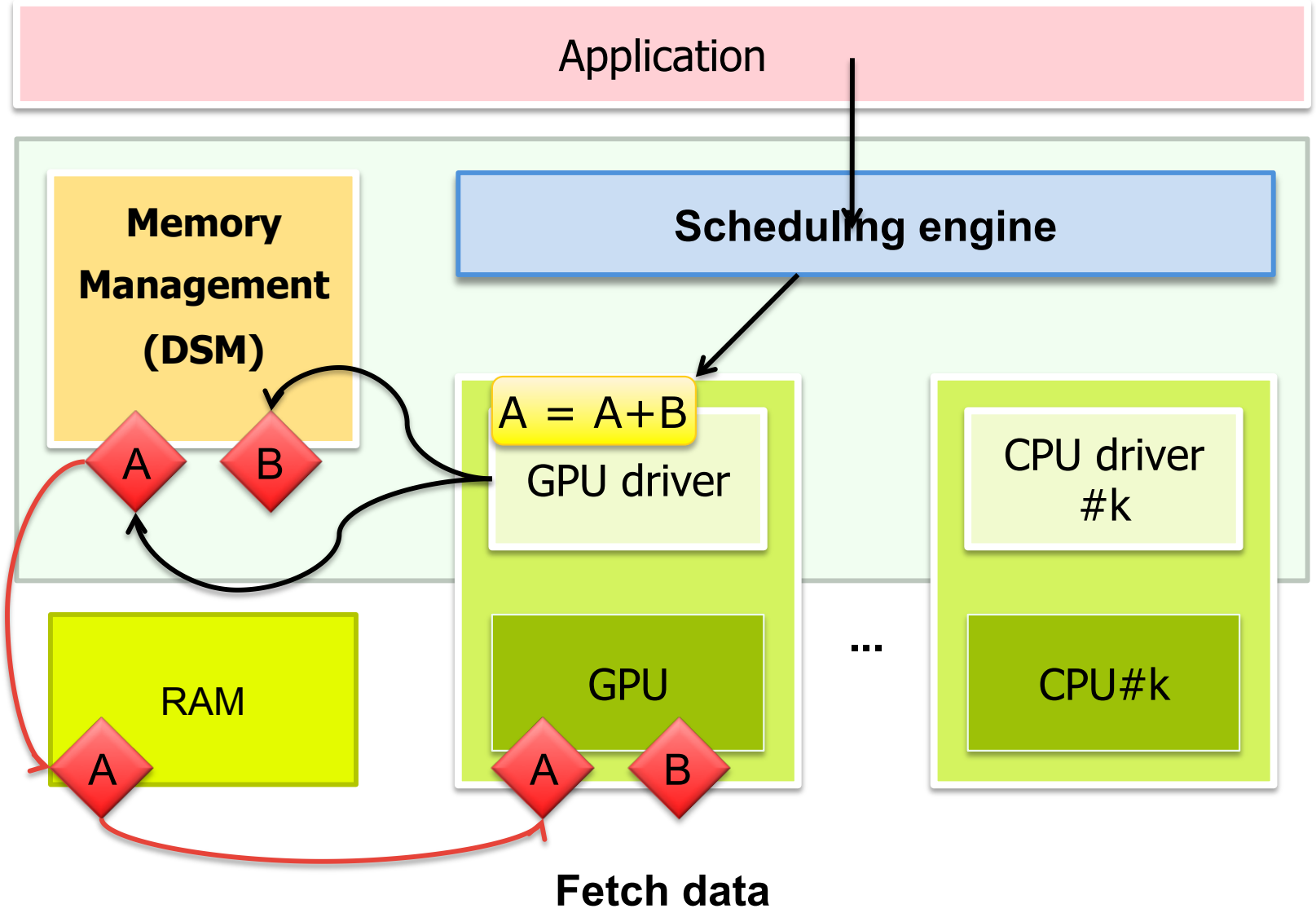
Schedule task

StarPU Internal Functioning

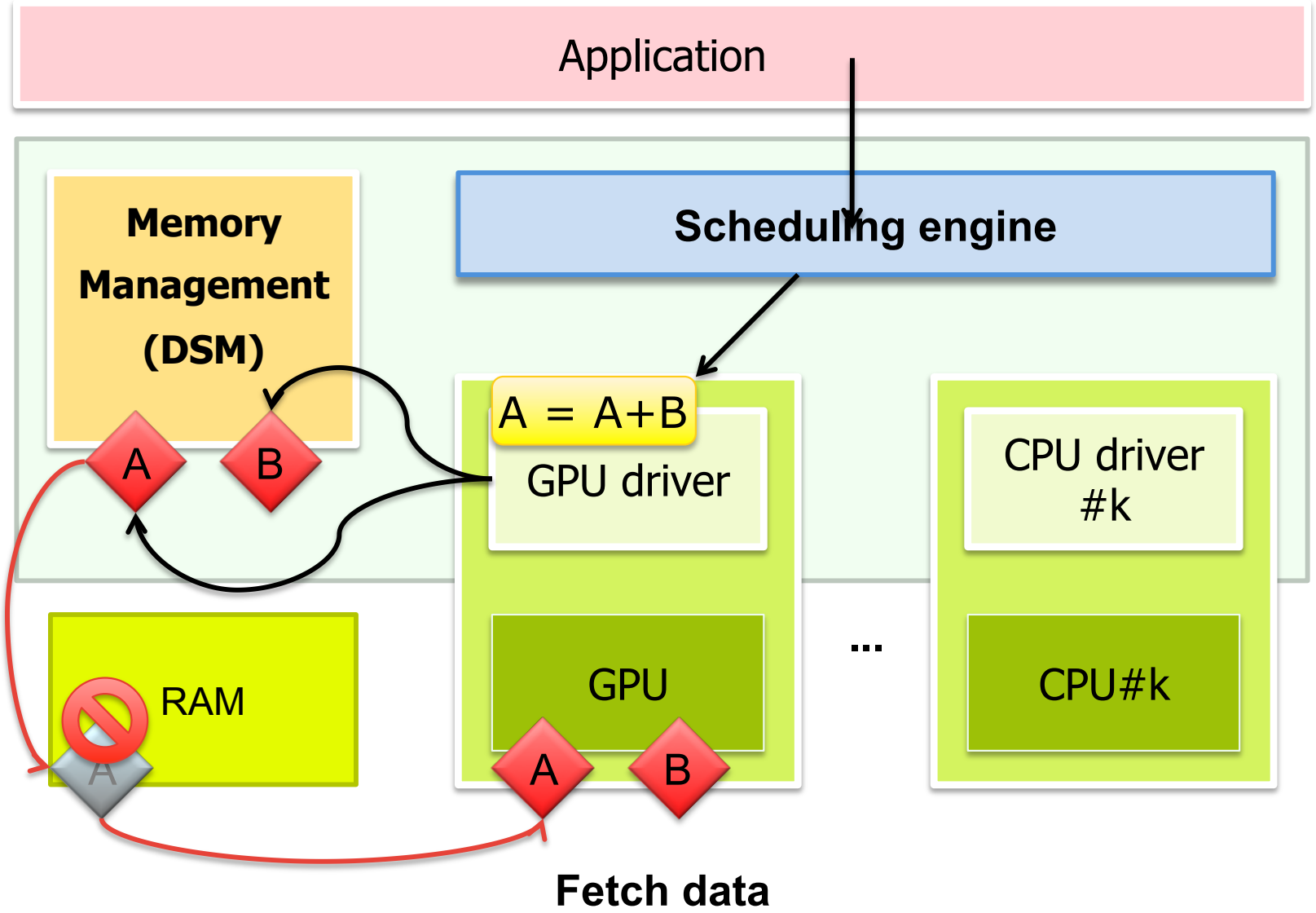


Fetch data

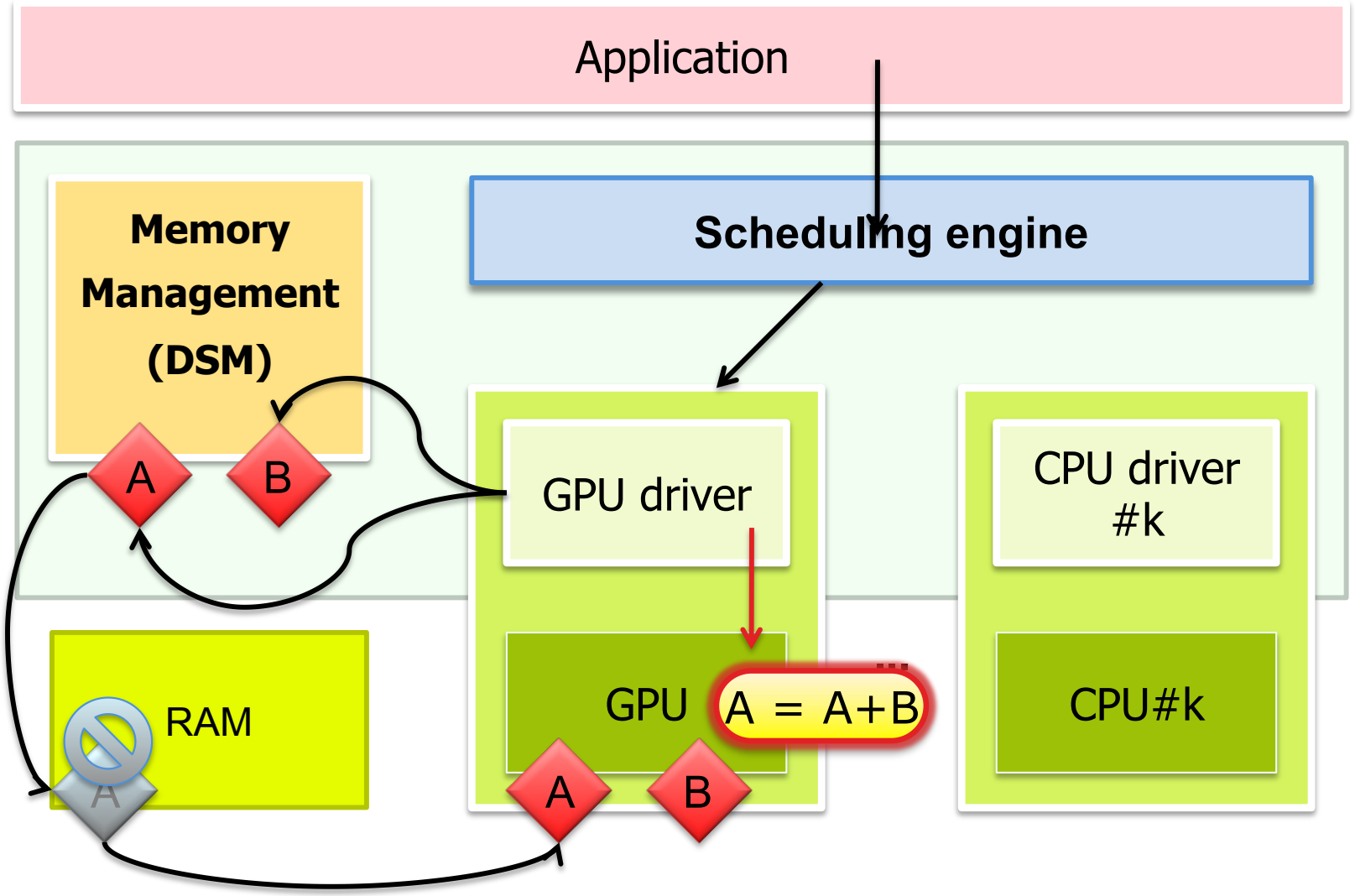
StarPU Internal Functioning



StarPU Internal Functioning

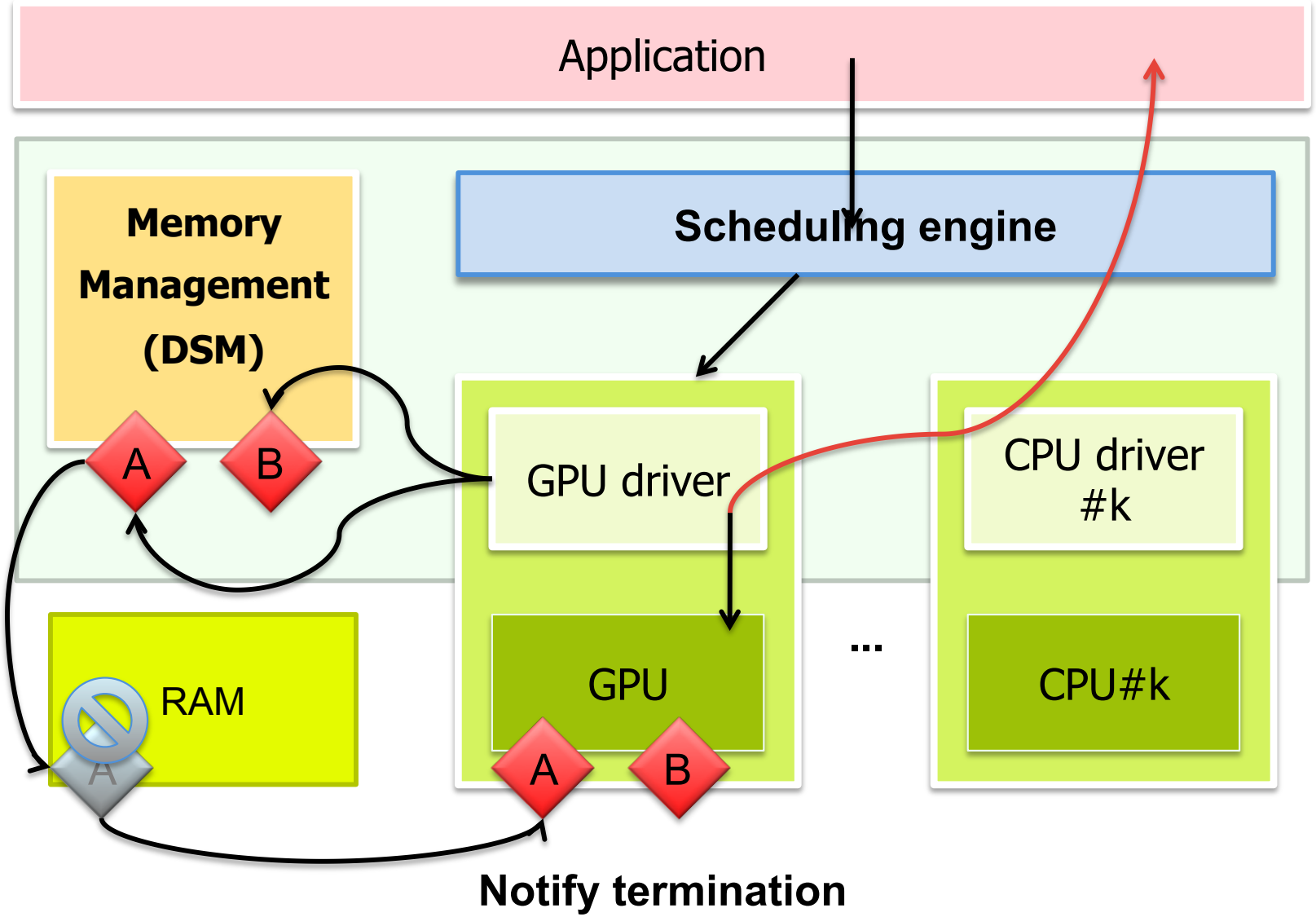


StarPU Internal Functioning



Offload computation

StarPU Internal Functioning



2

Scheduling

StarPU Scheduling Policies

- No *one size fits all* policy

StarPU Scheduling Policies

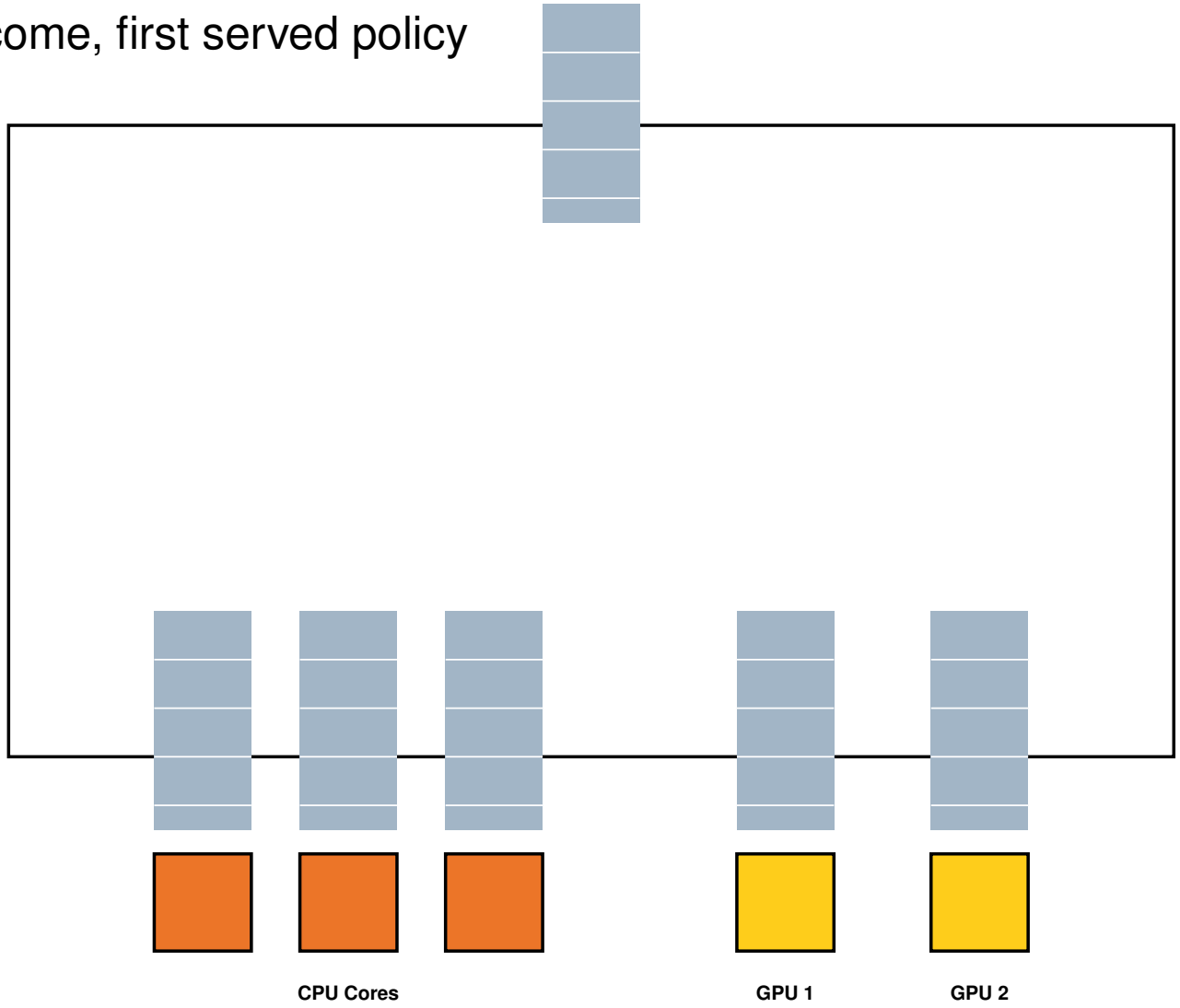
- No *one size fits all* policy
- Selectable scheduling policy
  - Predefined set of popular policies

The Eager Scheduler

- First come, first served policy

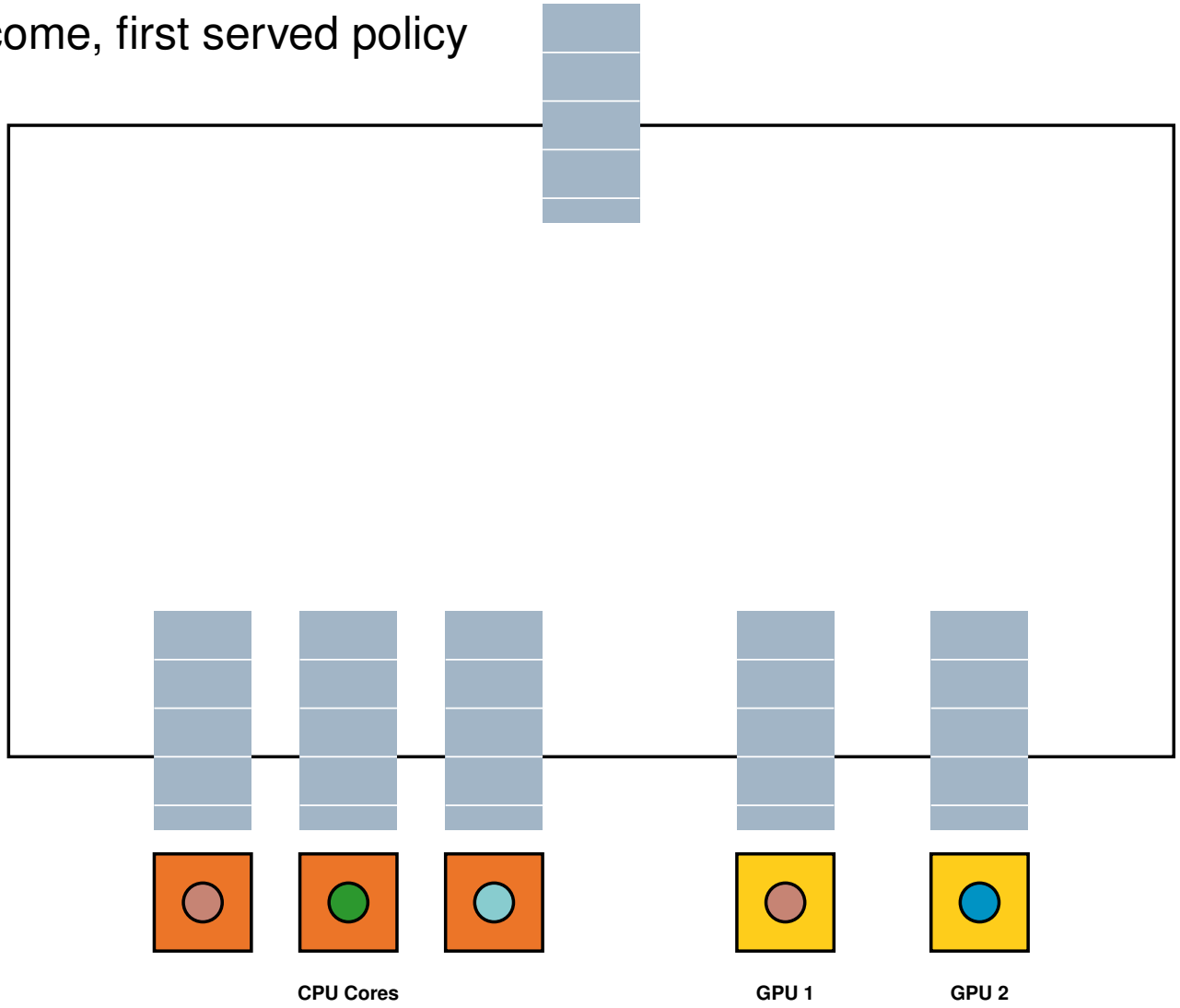
The Eager Scheduler

- First come, first served policy



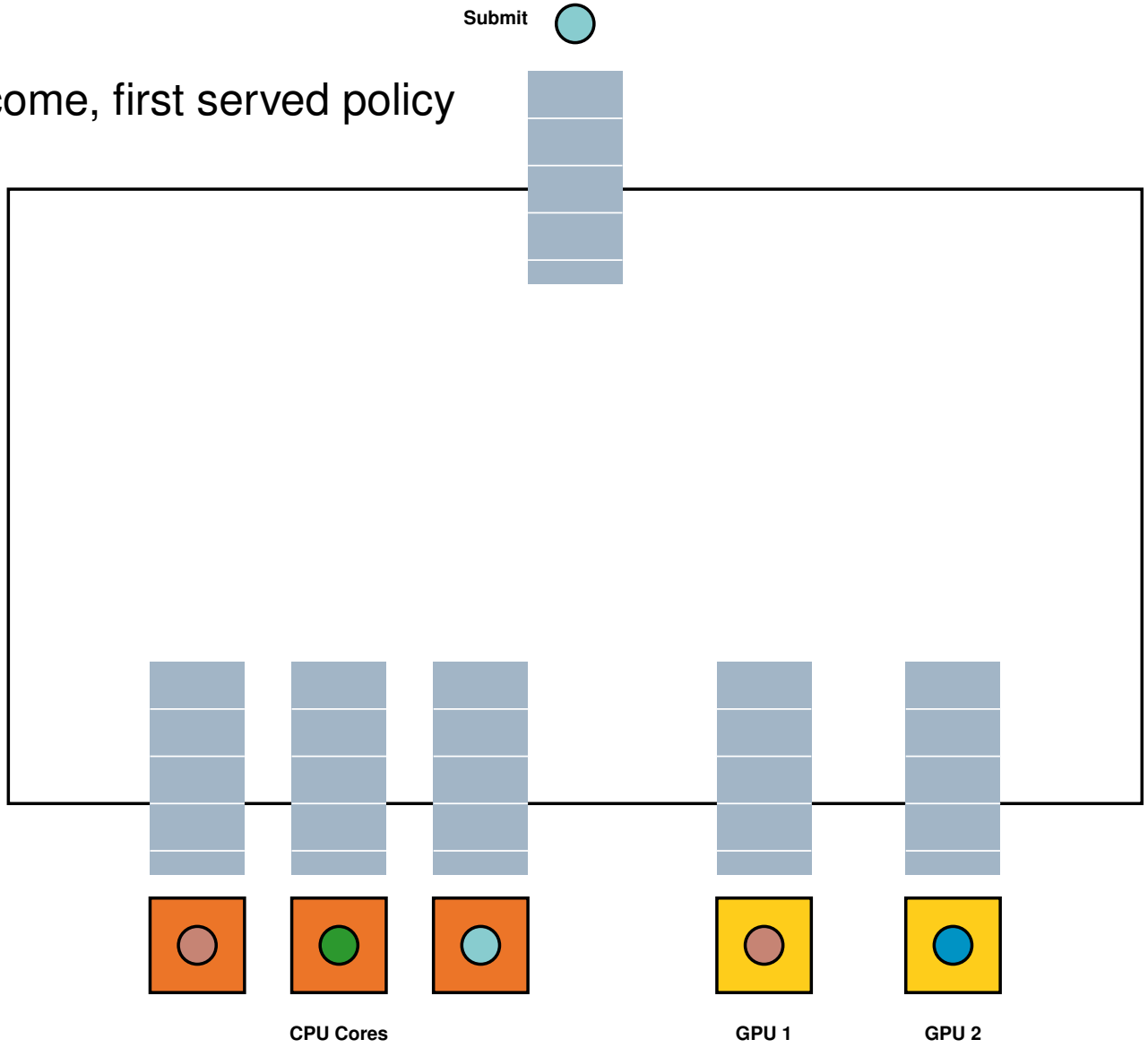
The Eager Scheduler

- First come, first served policy



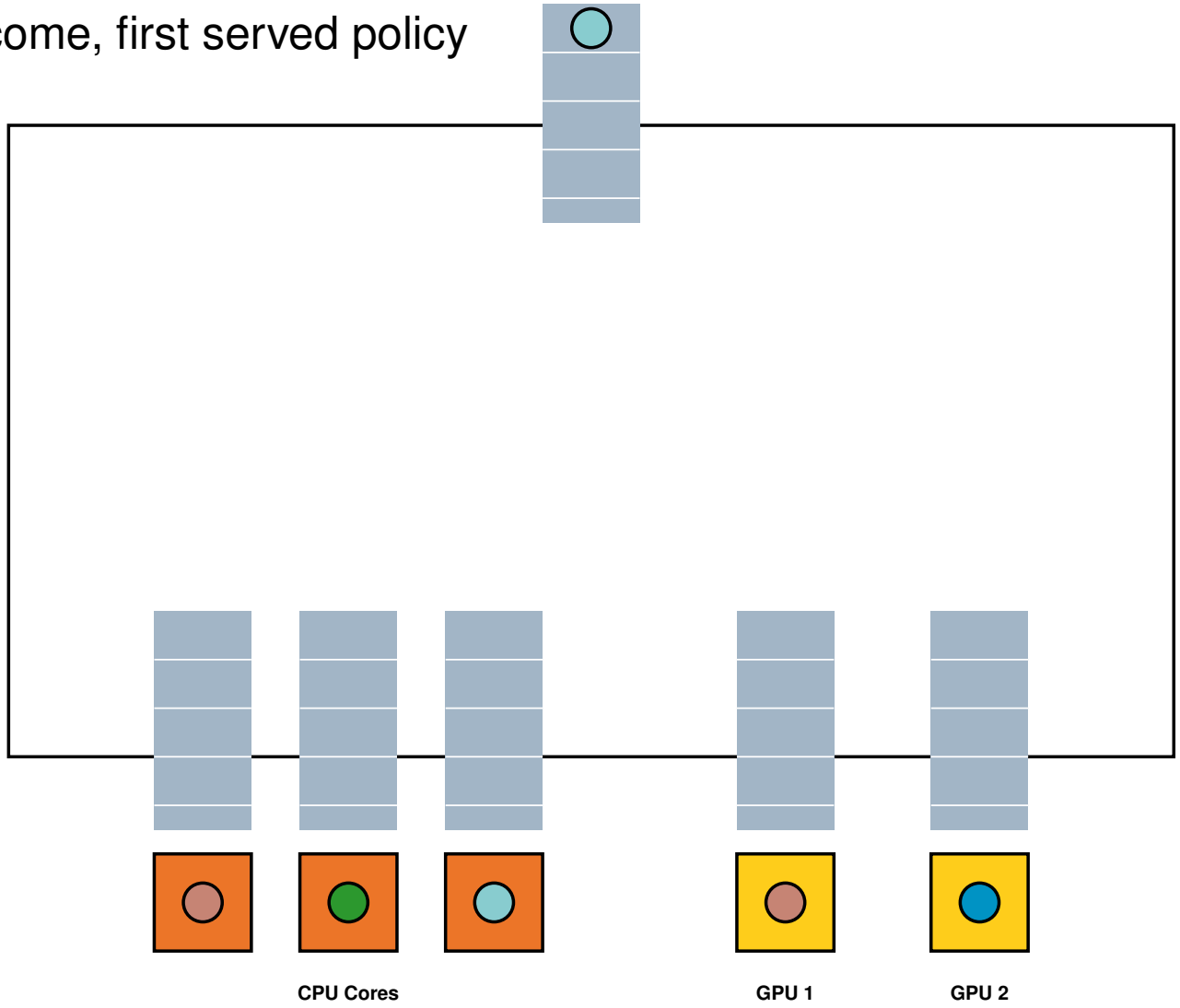
The Eager Scheduler

- First come, first served policy



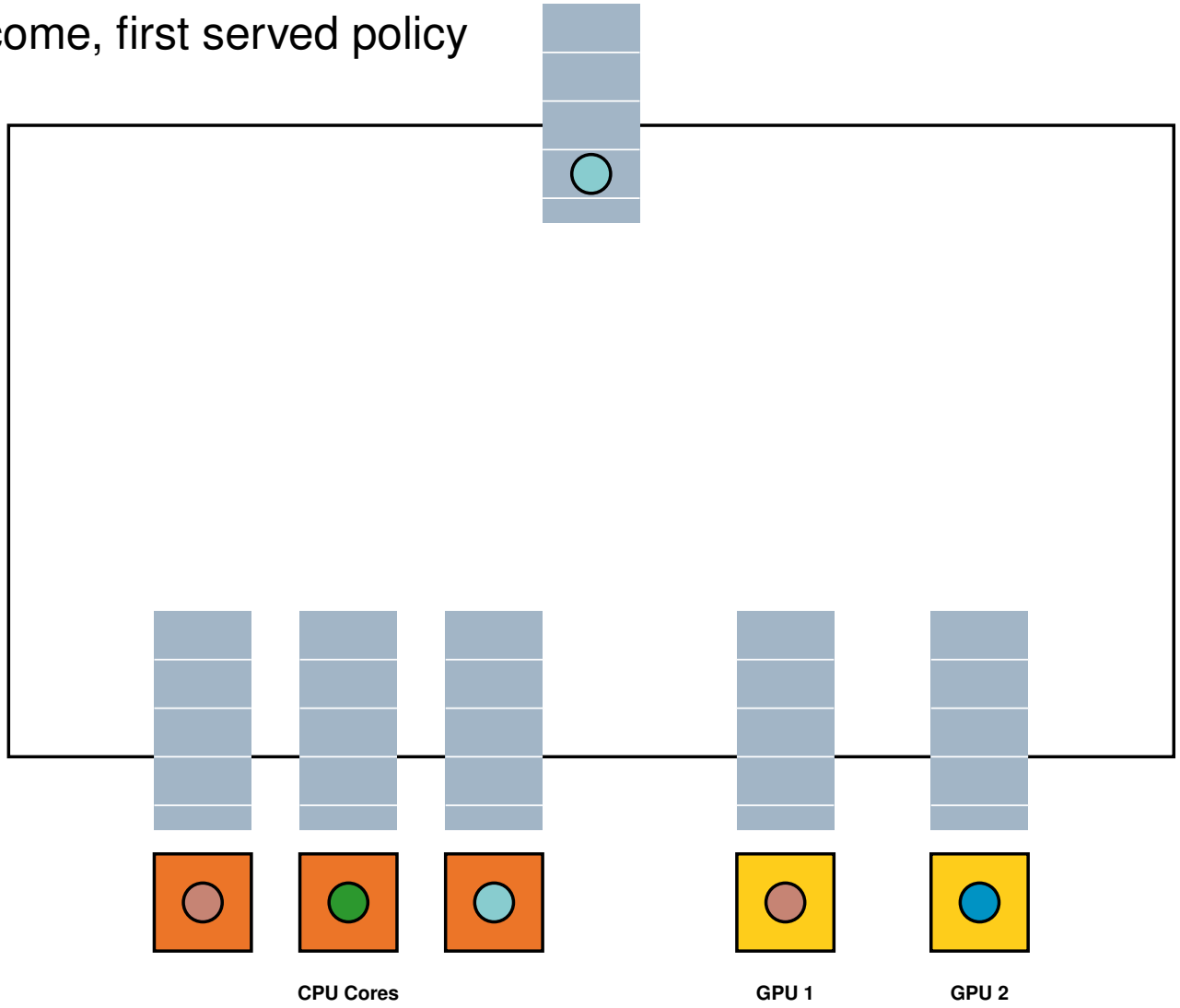
The Eager Scheduler

- First come, first served policy



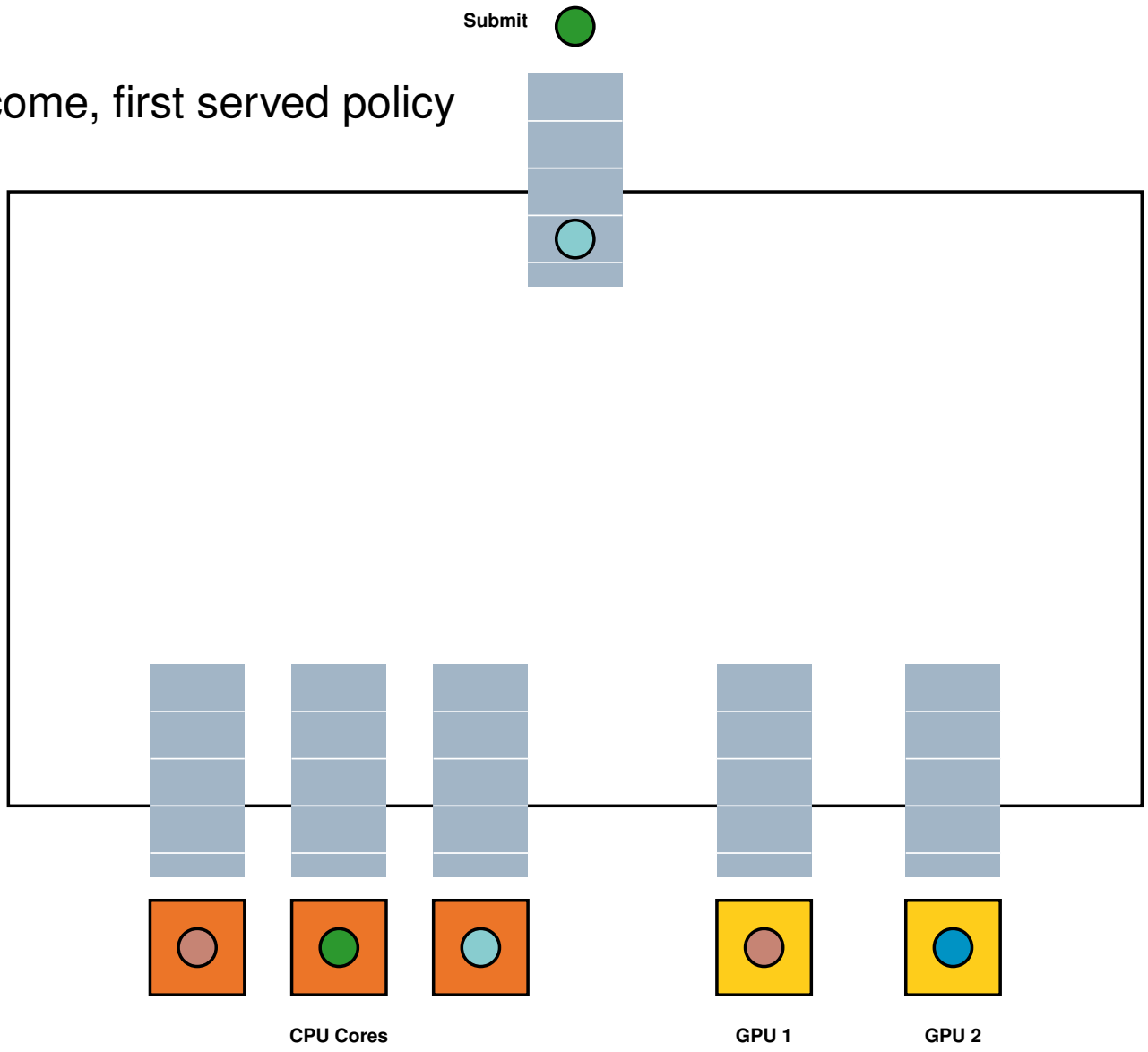
The Eager Scheduler

- First come, first served policy



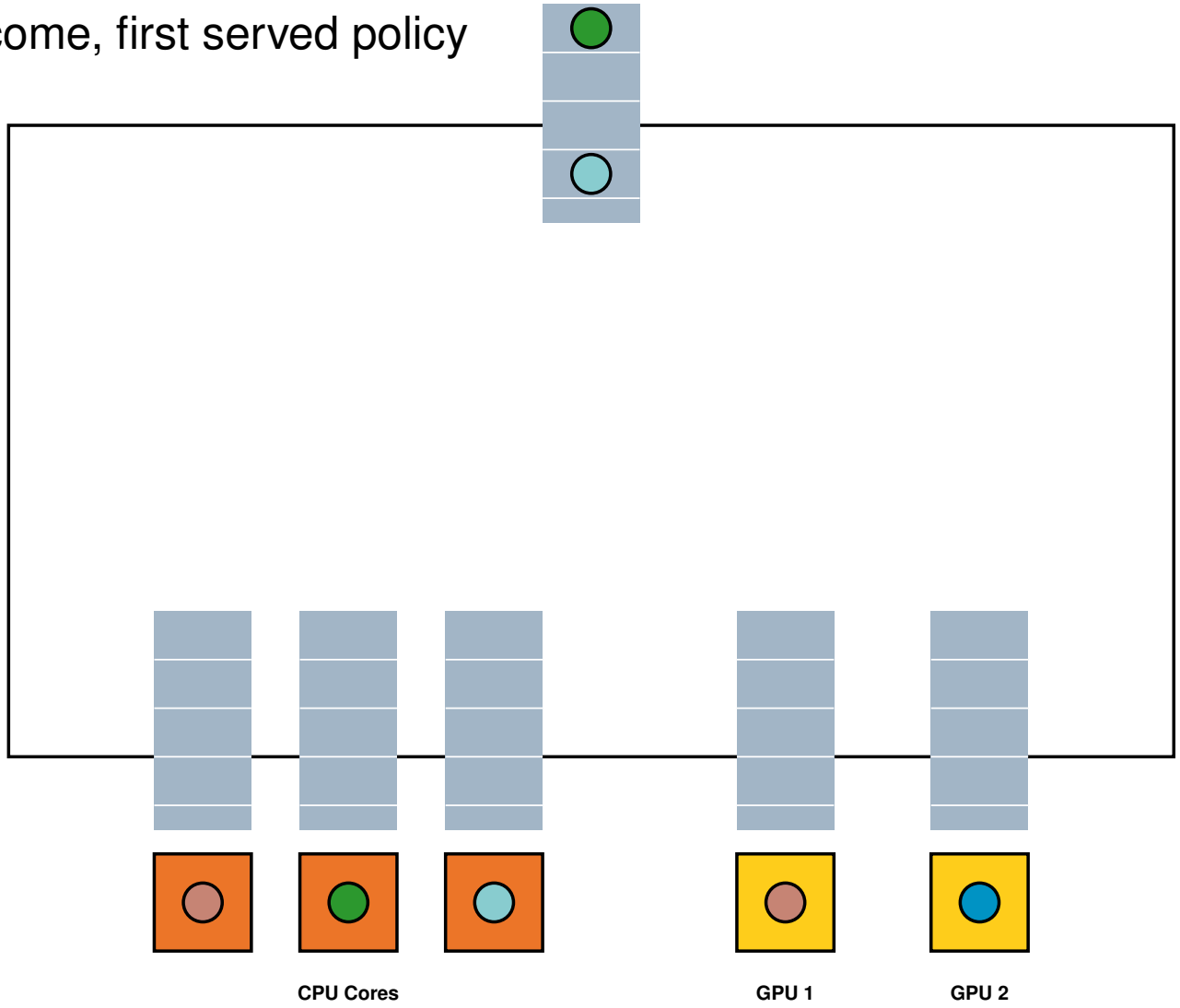
The Eager Scheduler

- First come, first served policy



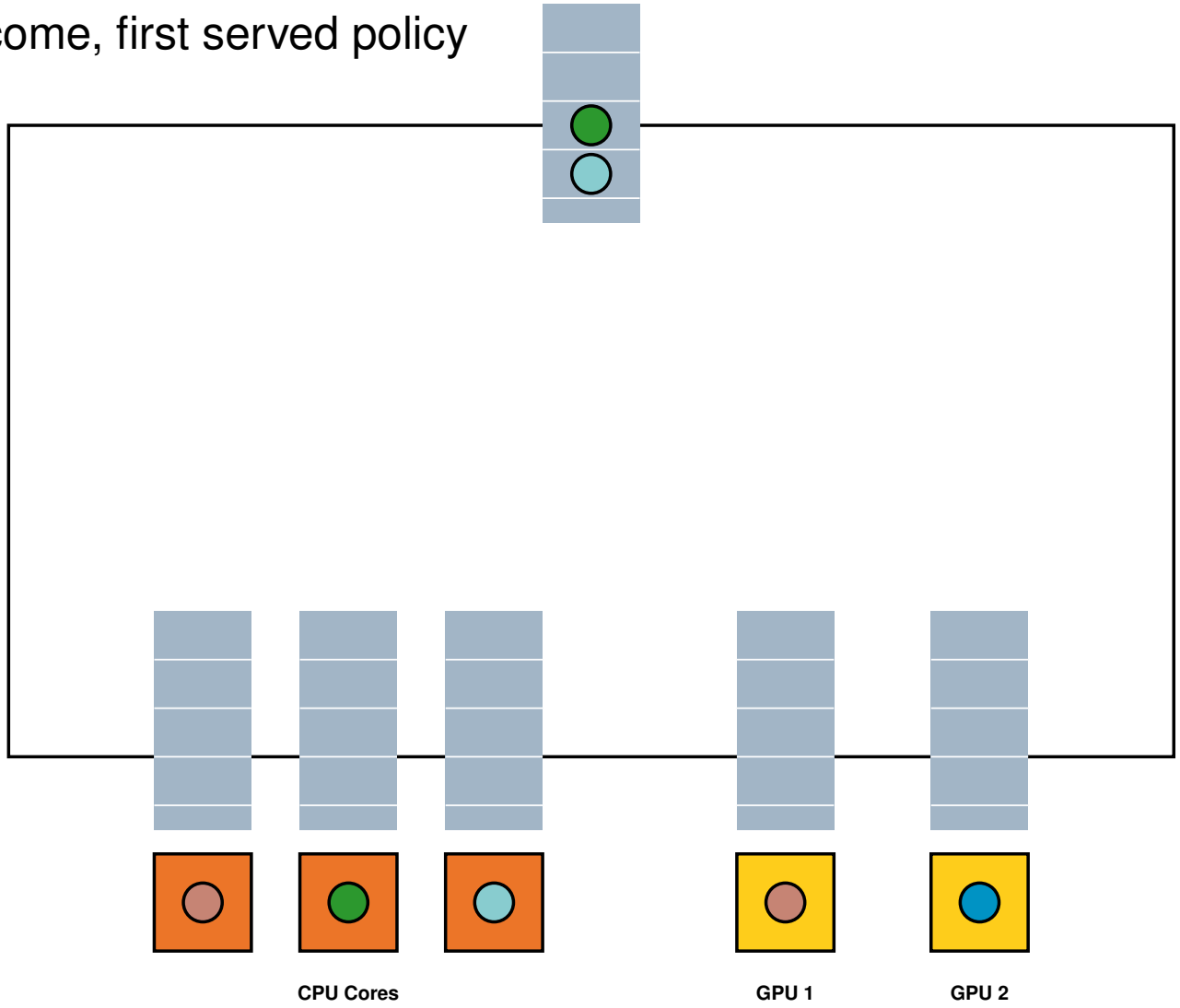
The Eager Scheduler

- First come, first served policy



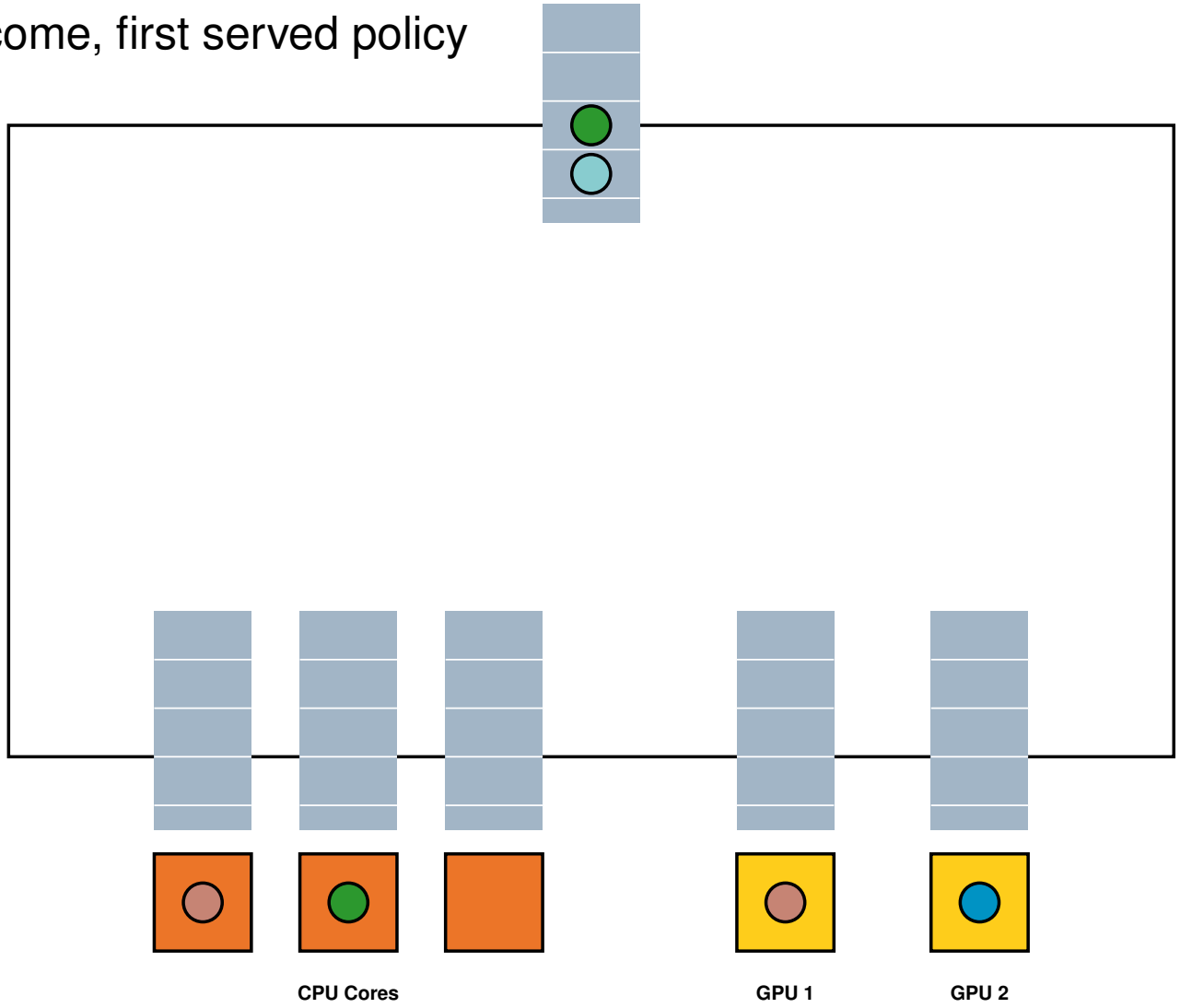
The Eager Scheduler

- First come, first served policy



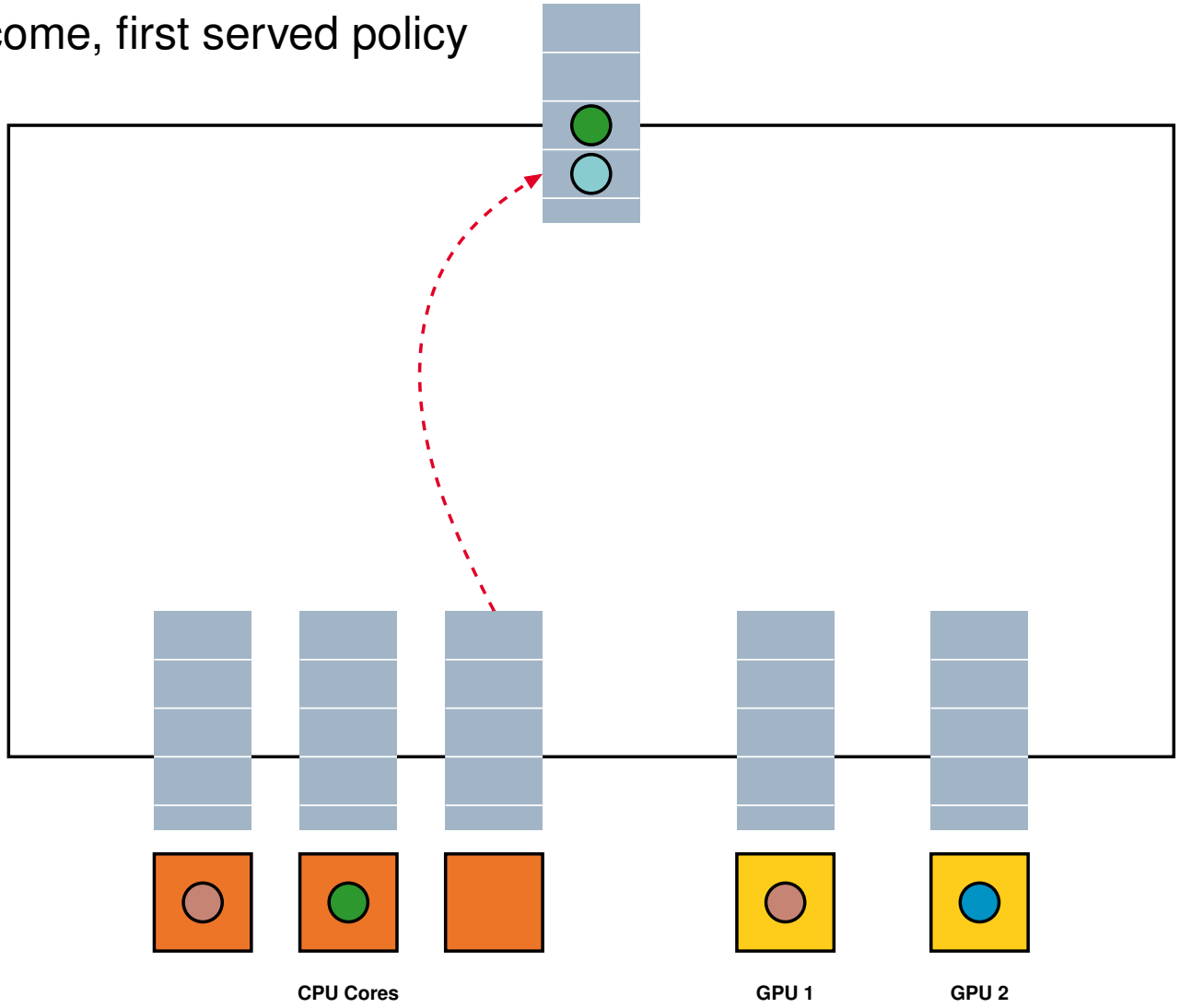
The Eager Scheduler

- First come, first served policy



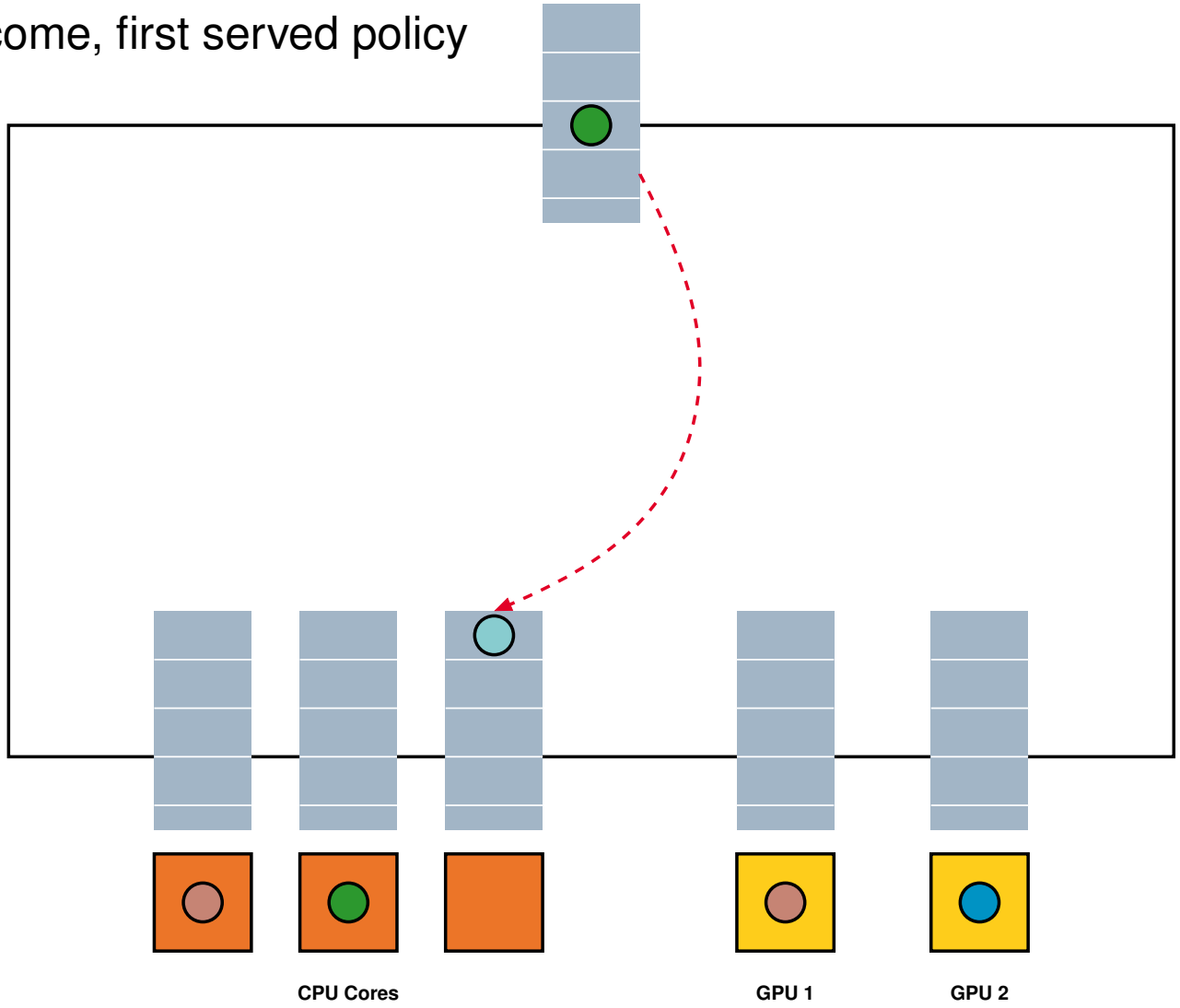
The Eager Scheduler

- First come, first served policy



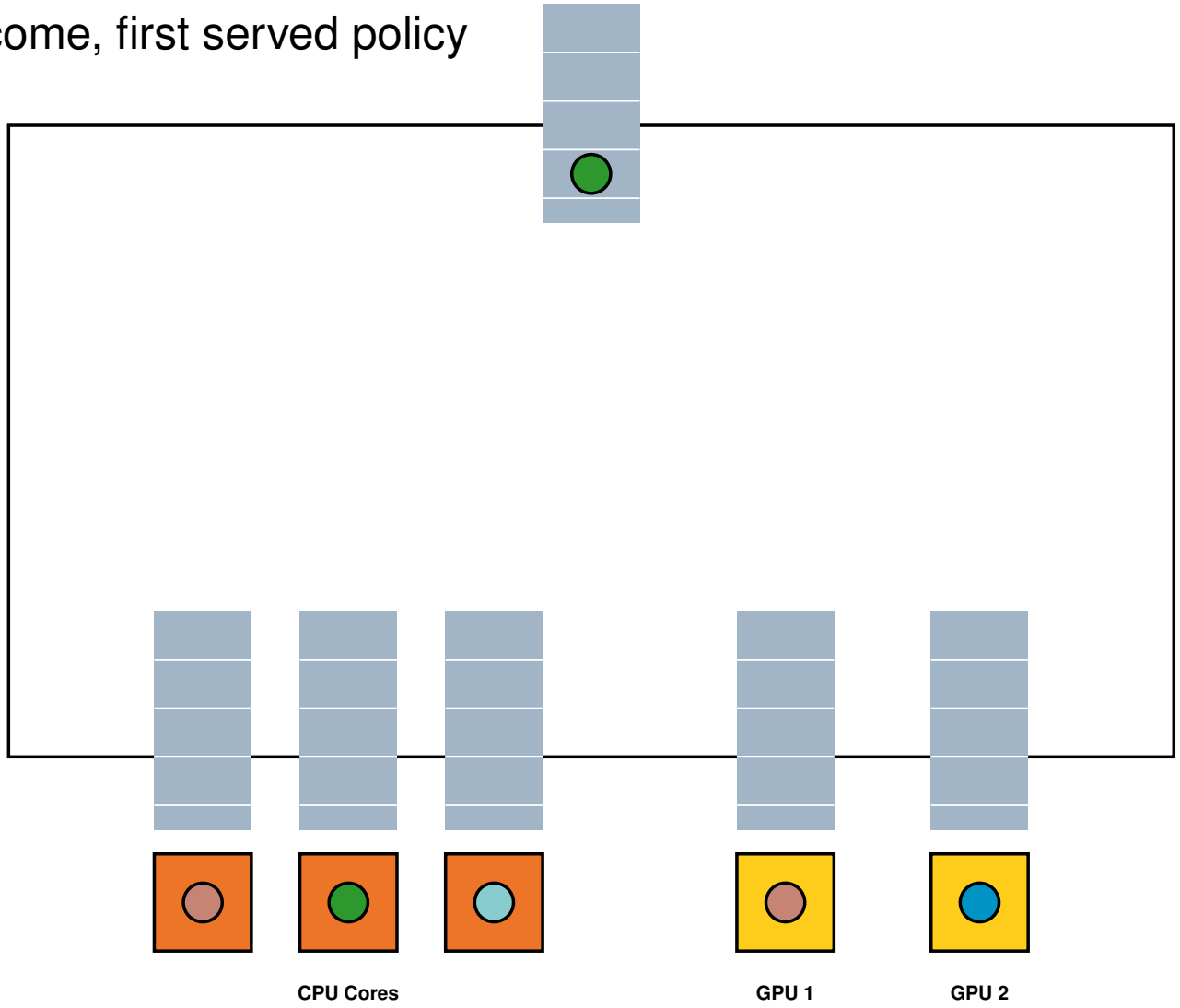
The Eager Scheduler

- First come, first served policy



The Eager Scheduler

- First come, first served policy

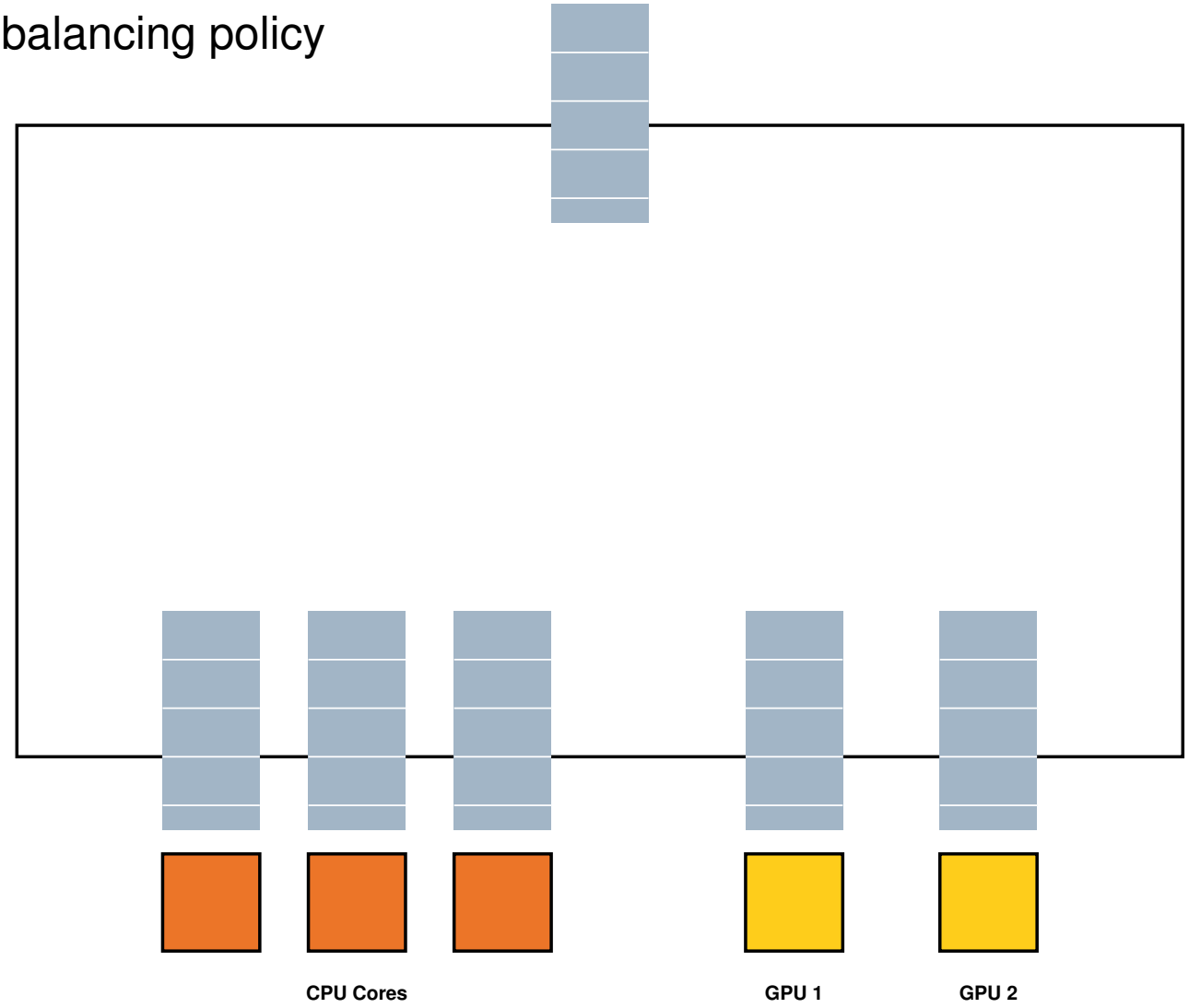


The Work Stealing Scheduler

- Load balancing policy

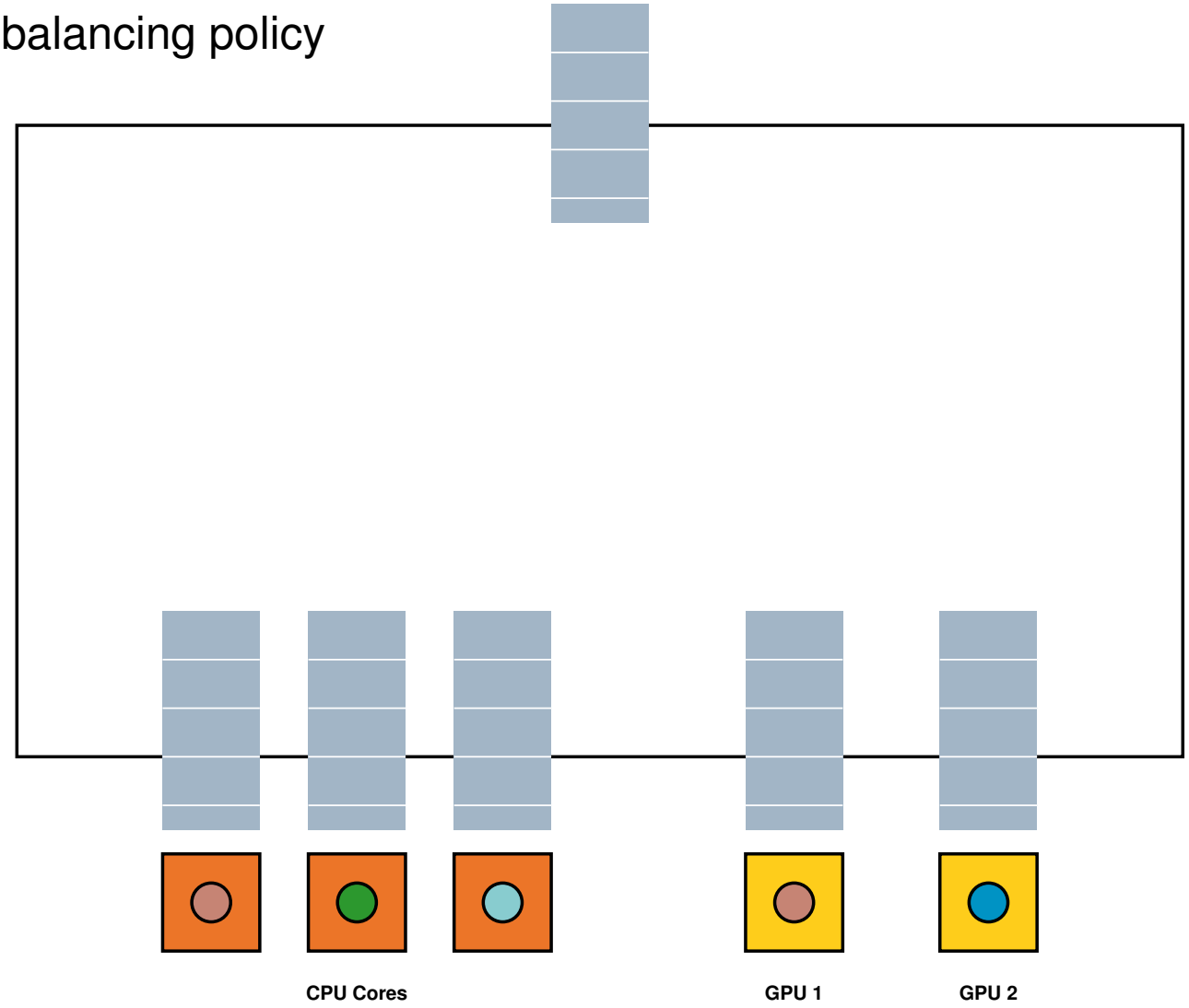
The Work Stealing Scheduler

- Load balancing policy



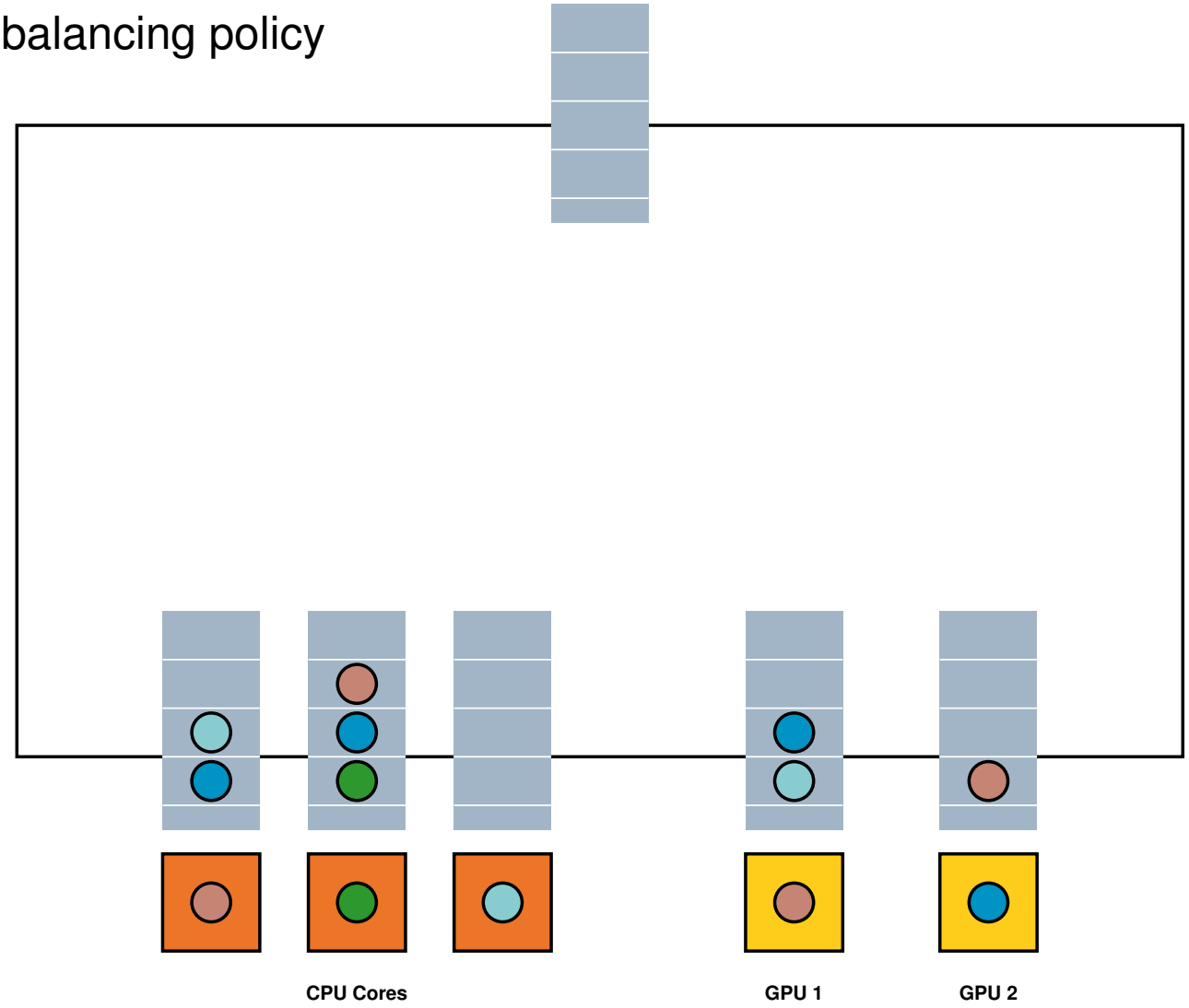
The Work Stealing Scheduler

- Load balancing policy



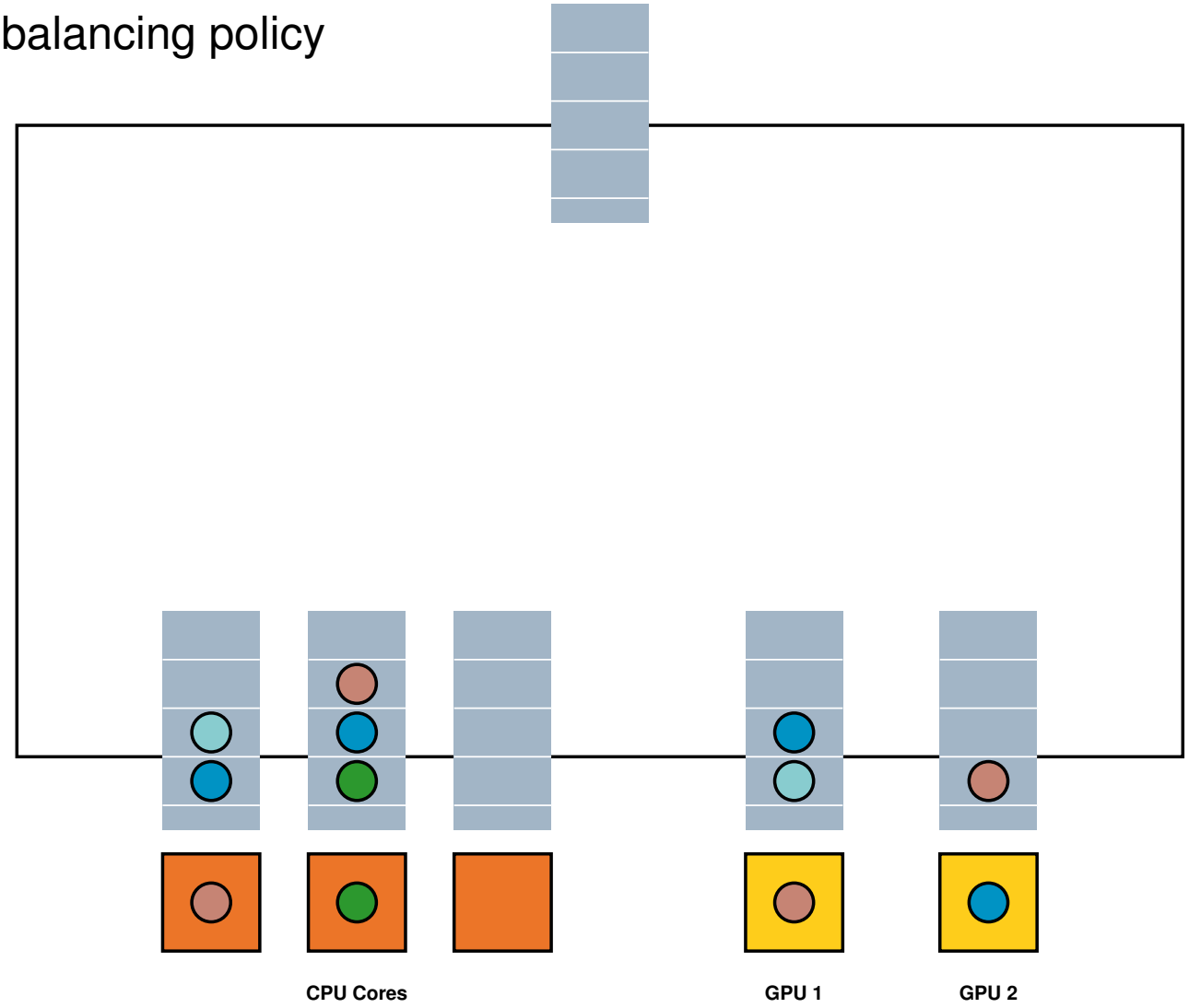
The Work Stealing Scheduler

- Load balancing policy



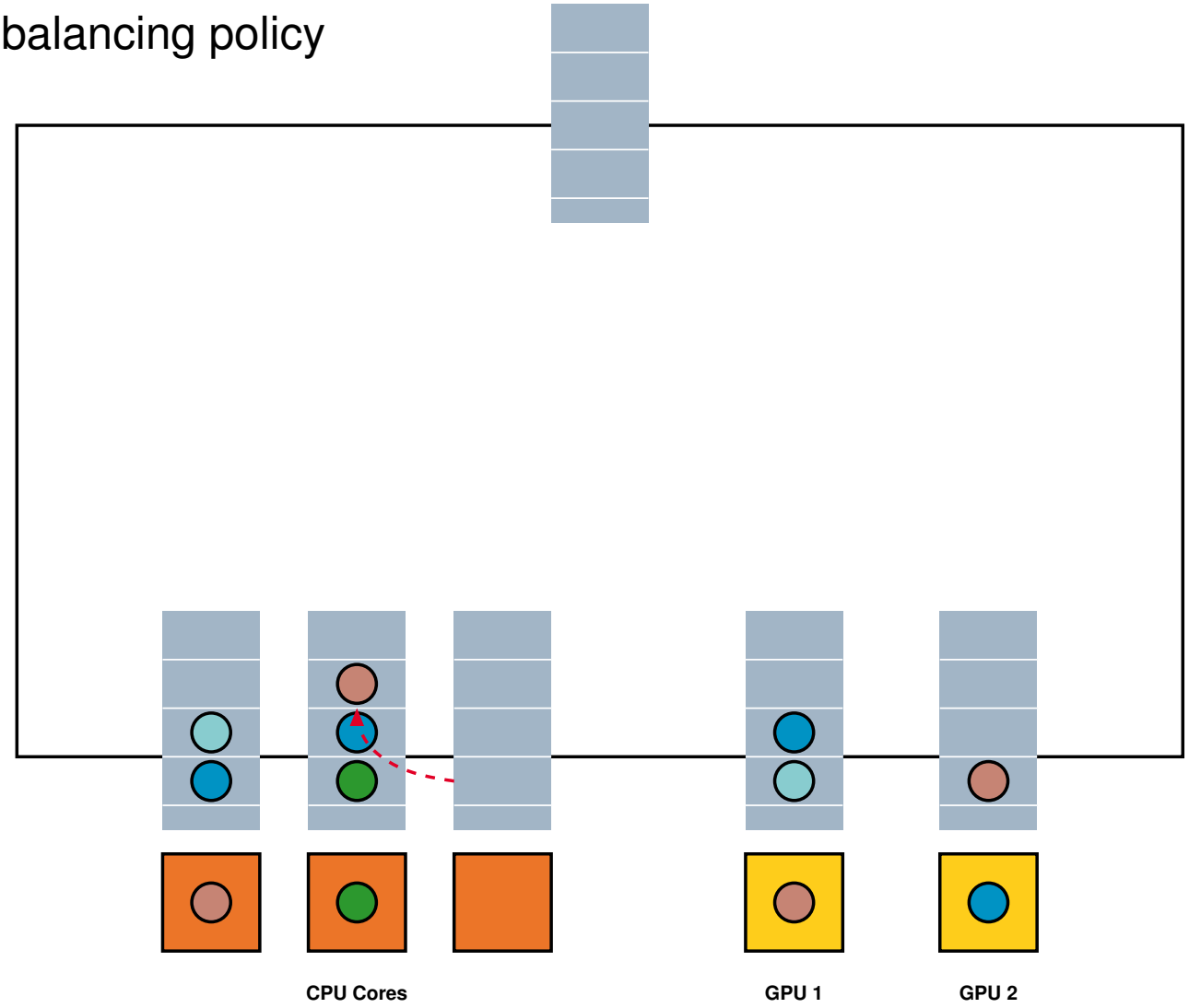
The Work Stealing Scheduler

- Load balancing policy



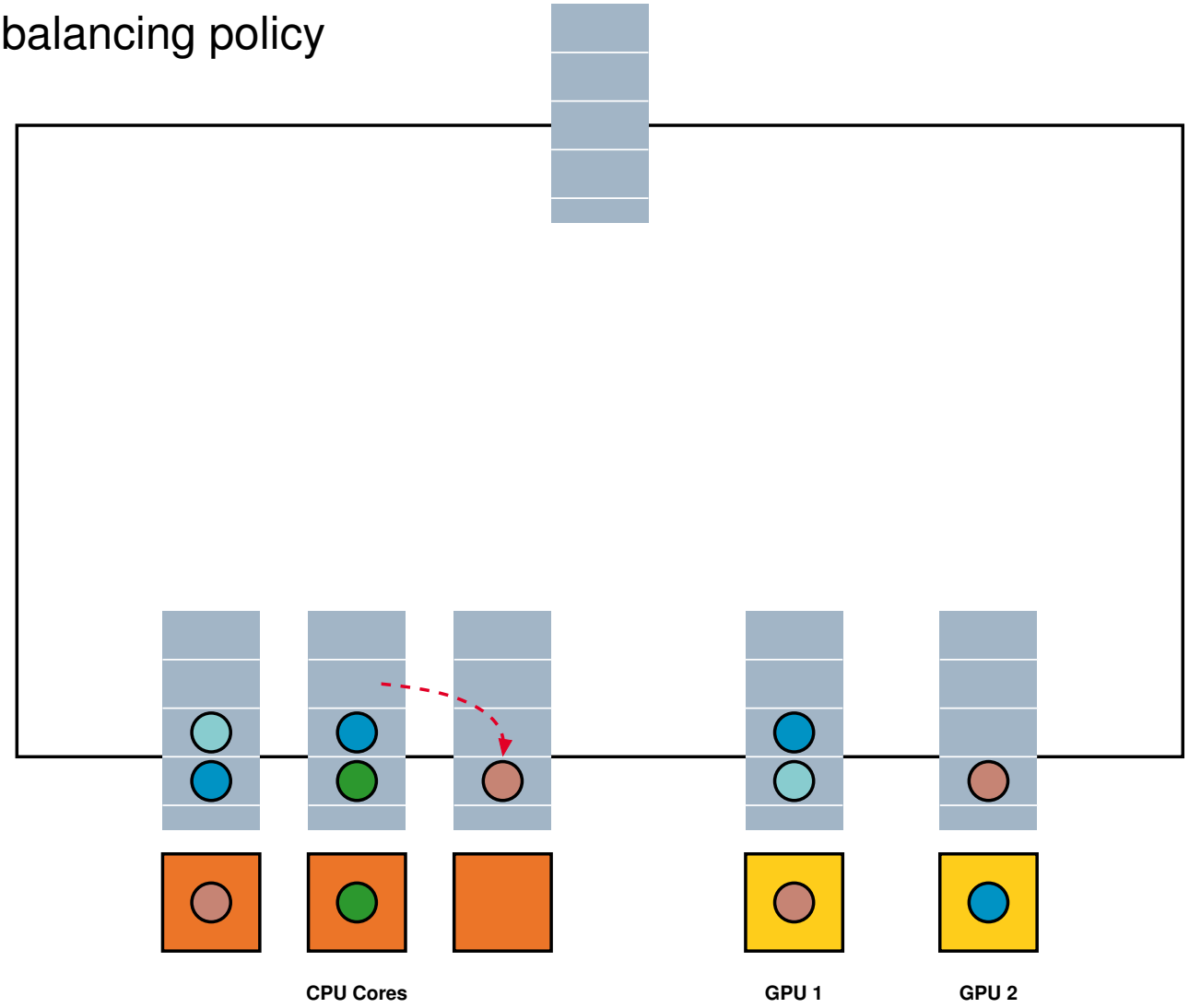
The Work Stealing Scheduler

- Load balancing policy



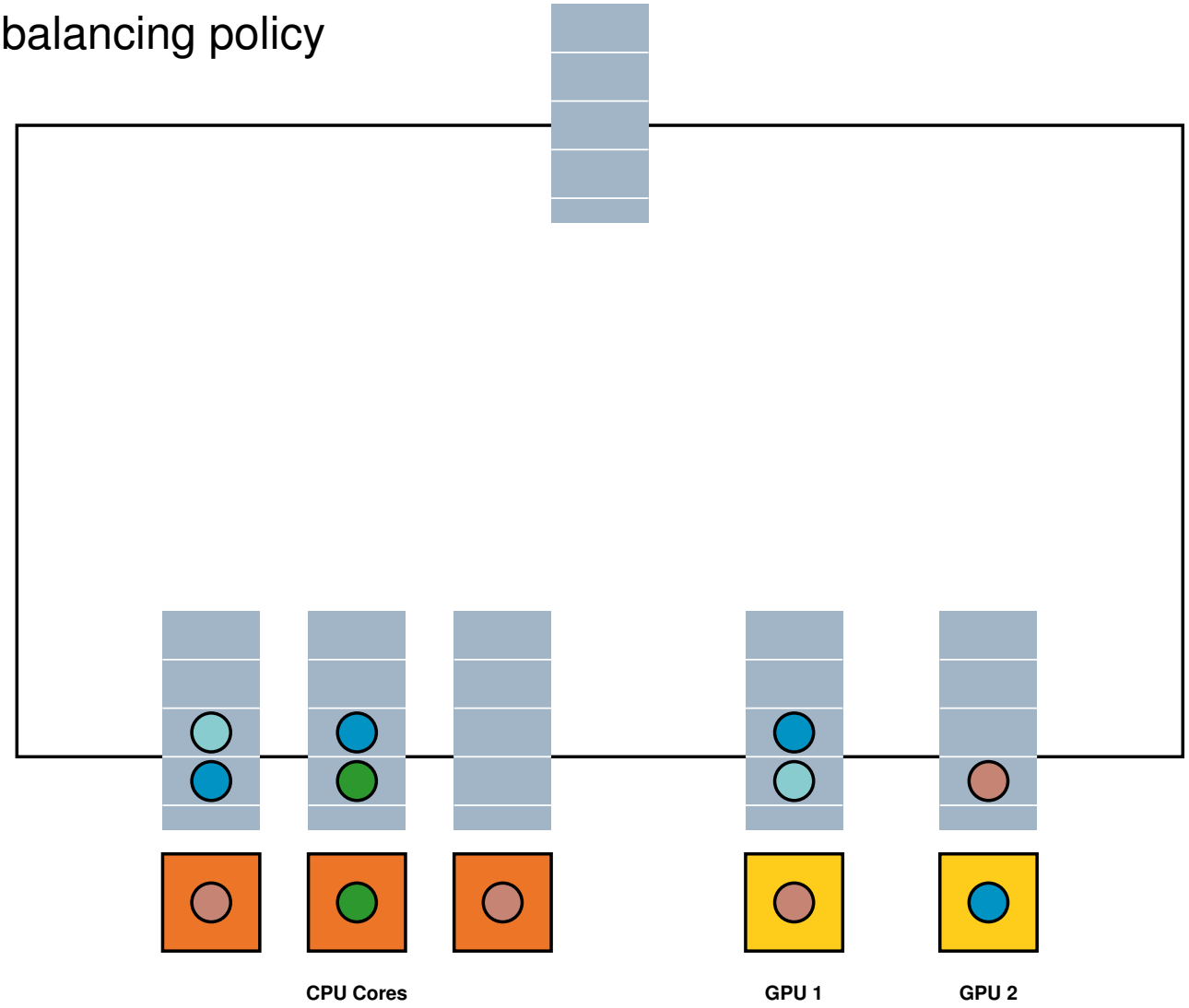
The Work Stealing Scheduler

- Load balancing policy



The Work Stealing Scheduler

- Load balancing policy



Going Beyond

Scheduling is a decision process

Going Beyond

Scheduling is a decision process

- Providing more input to the scheduler. . .

Going Beyond

Scheduling is a decision process

- Providing more input to the scheduler. . .
- . . . can lead to better scheduling decisions

Going Beyond

Scheduling is a decision process

- Providing more input to the scheduler. . .
- . . . can lead to better scheduling decisions

What kind of information?

Going Beyond

Scheduling is a decision process

- Providing more input to the scheduler. . .
- . . . can lead to better scheduling decisions

What kind of information?

- Relative importance of tasks
  - Priorities

Going Beyond

Scheduling is a decision process

- Providing more input to the scheduler...
- ... can lead to better scheduling decisions

What kind of information?

- Relative importance of tasks
  - Priorities
- Cost of tasks
  - Codelet models

Going Beyond

Scheduling is a decision process

- Providing more input to the scheduler. . .
- . . . can lead to better scheduling decisions

What kind of information?

- Relative importance of tasks
  - Priorities
- Cost of tasks
  - Codelet models
- Cost of transferring data
  - Bus calibration

The Prio Scheduler

- Describe the relative importance of tasks

The Prio Scheduler

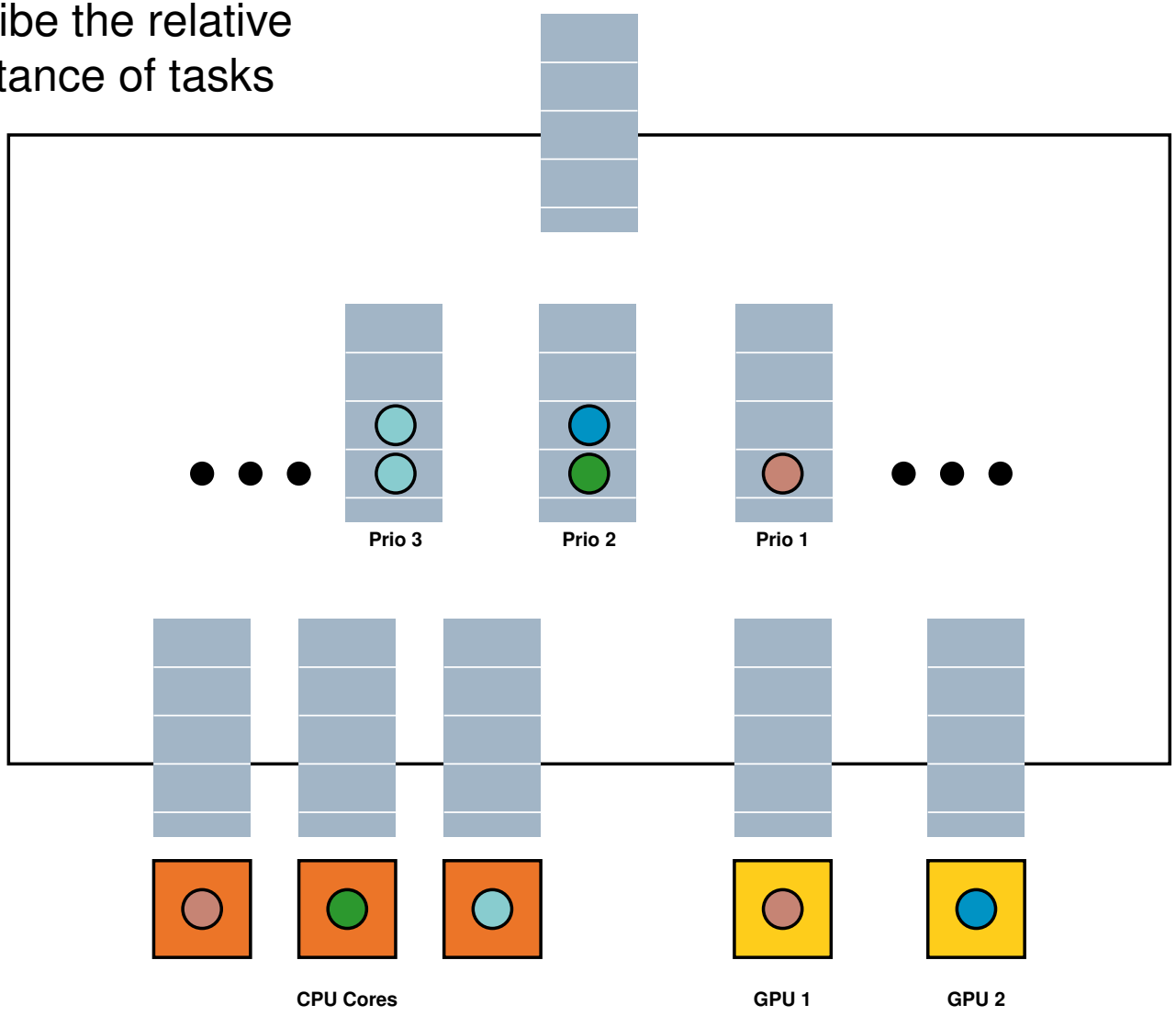
- Describe the relative importance of tasks
- Assign priorities to tasks
  - Values: -5 .. 0 .. $+5$

The Prio Scheduler

- Describe the relative importance of tasks
- Assign priorities to tasks
  - Values: -5 .. 0 .. $+5$
- Tell which task matter
  - Tasks that unlock key data pieces
  - Tasks that generate a lot of parallelism

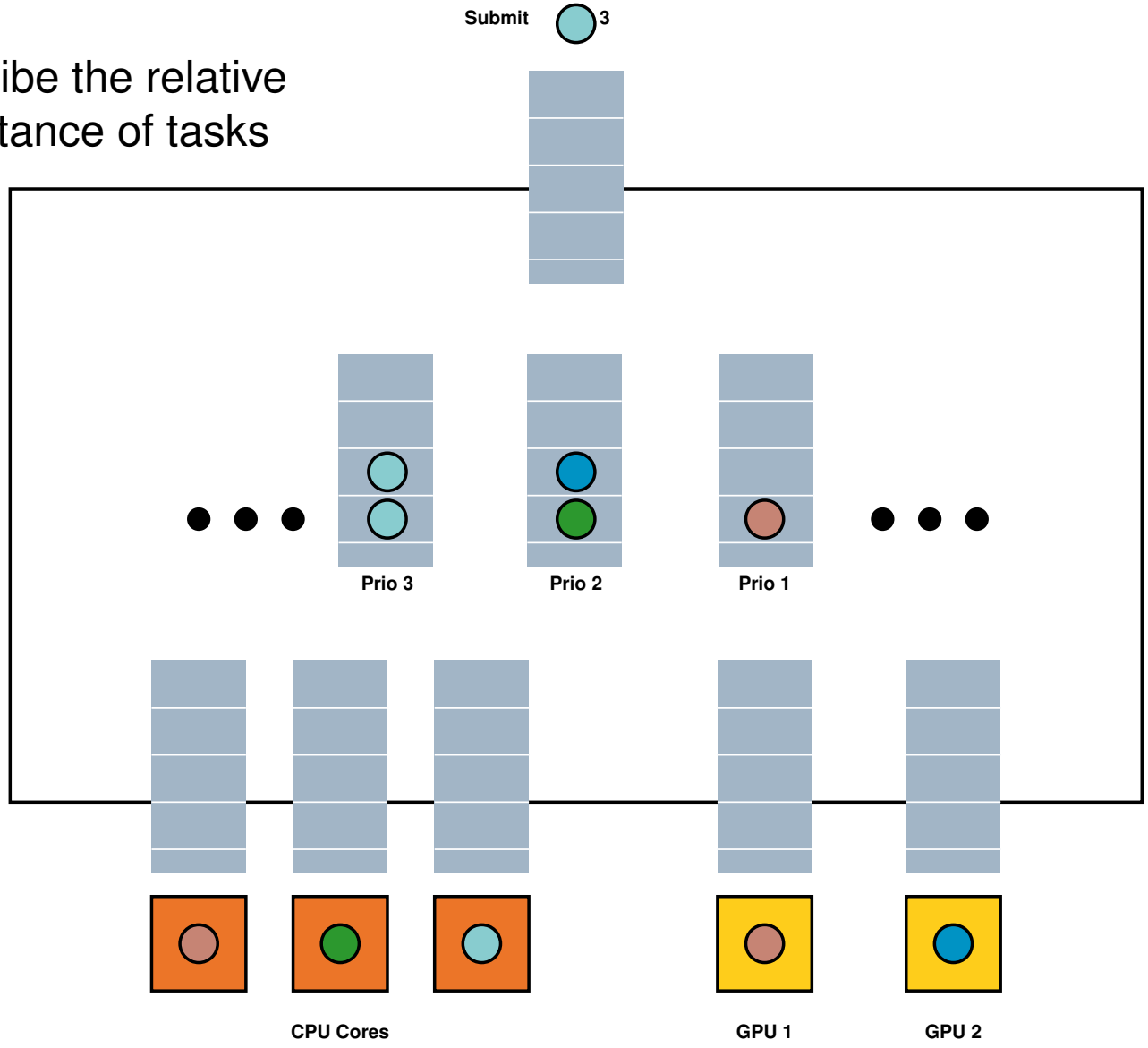
The Prio Scheduler

- Describe the relative importance of tasks



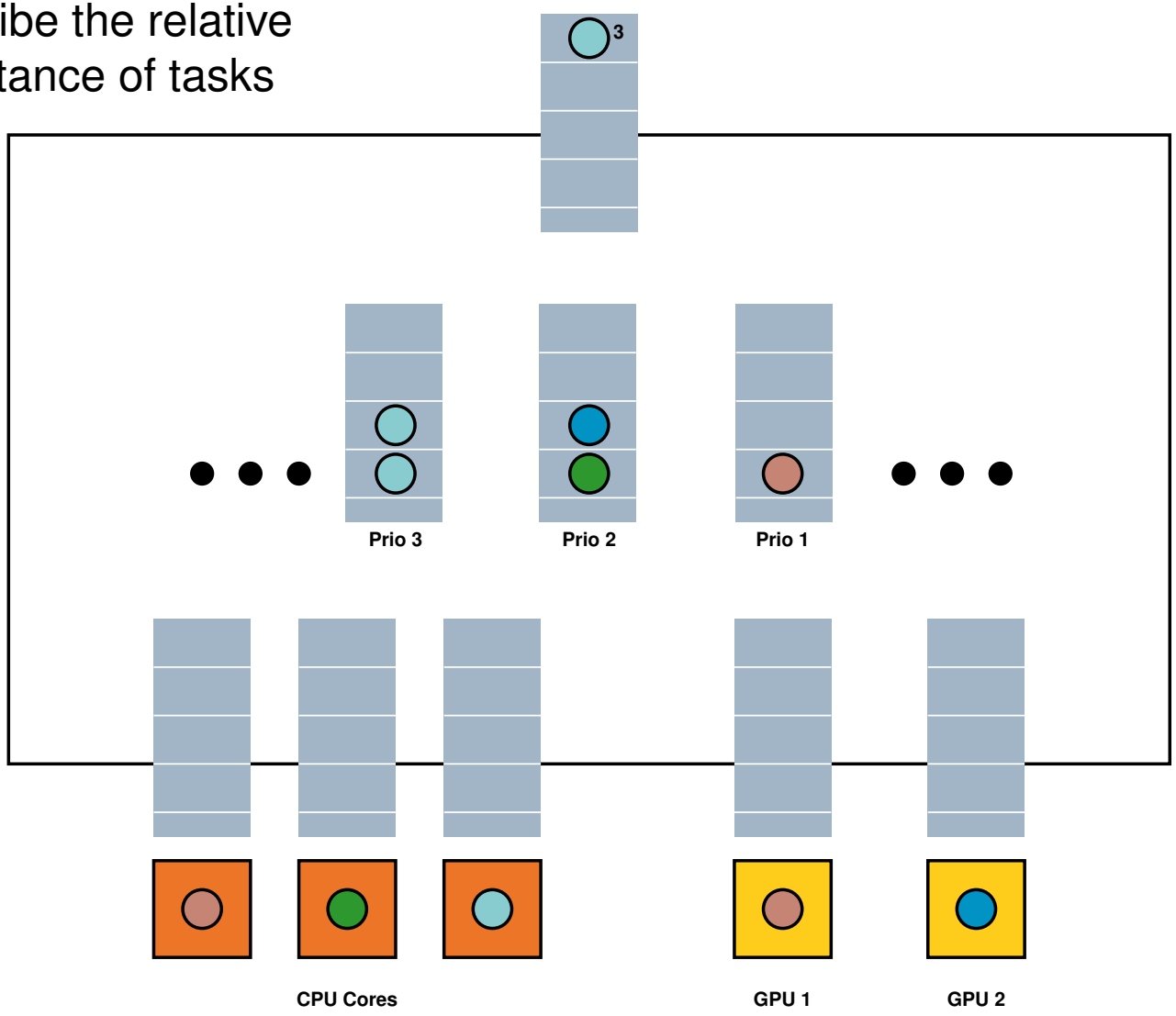
The Prio Scheduler

- Describe the relative importance of tasks



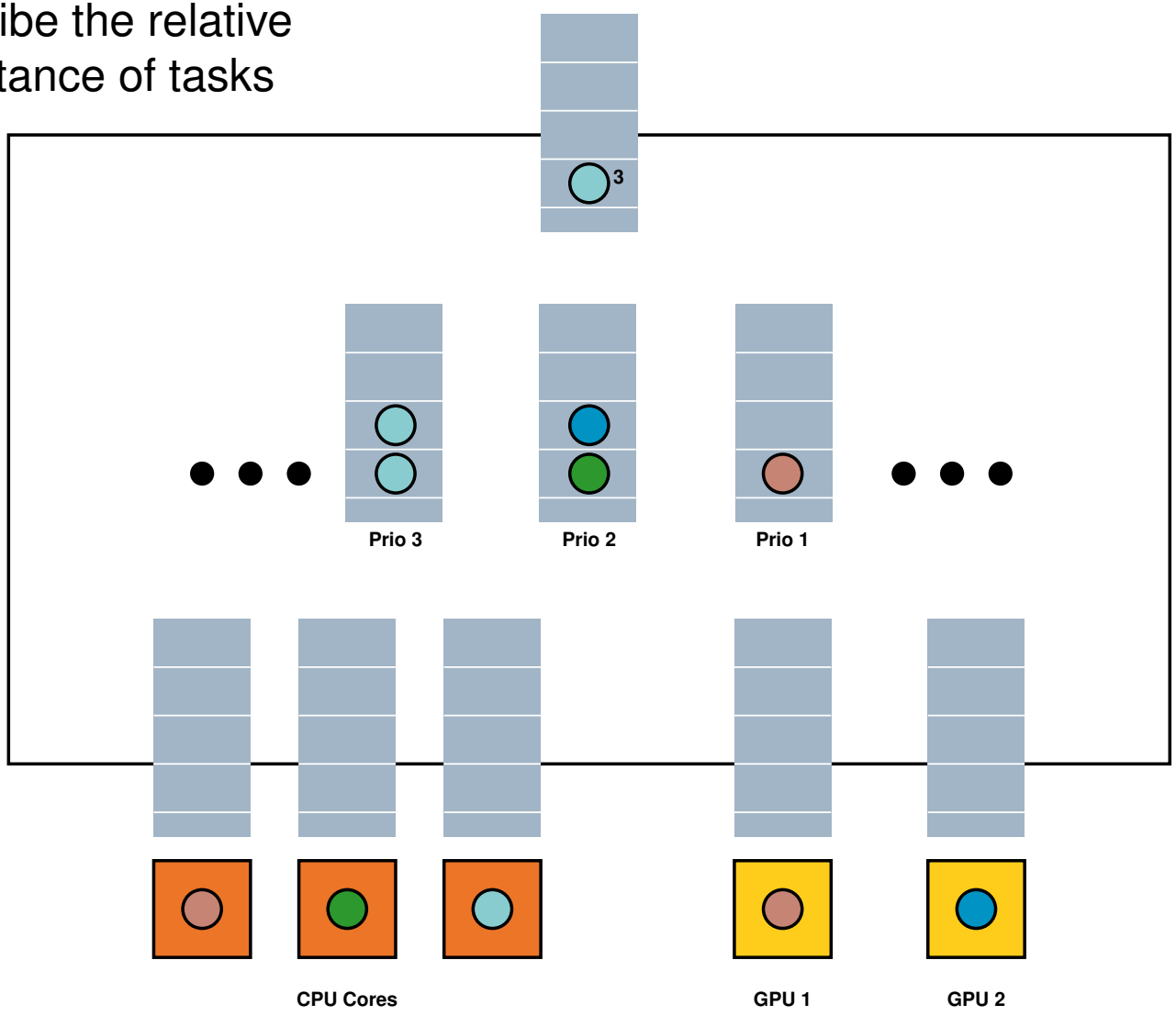
The Prio Scheduler

- Describe the relative importance of tasks



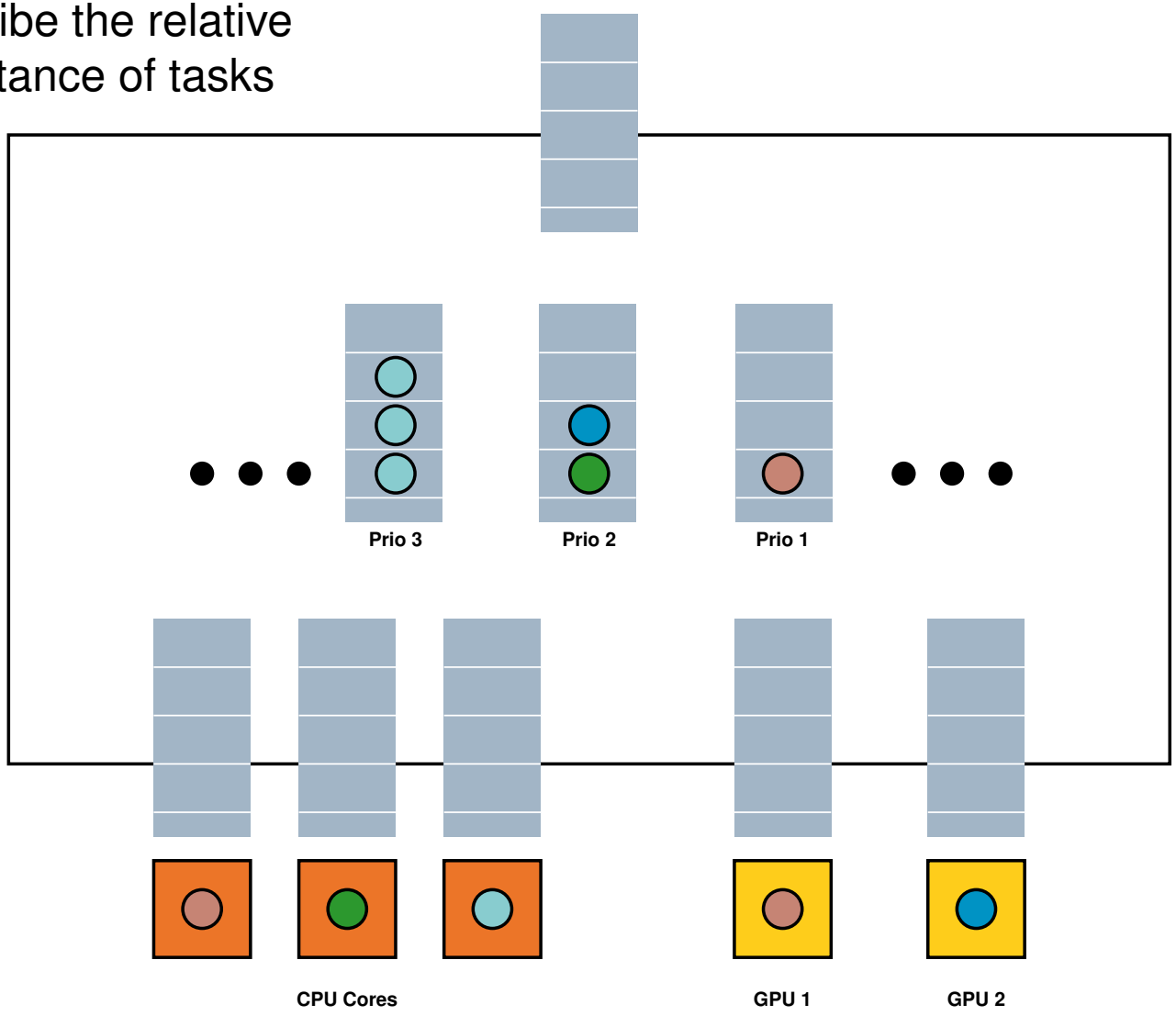
The Prio Scheduler

- Describe the relative importance of tasks



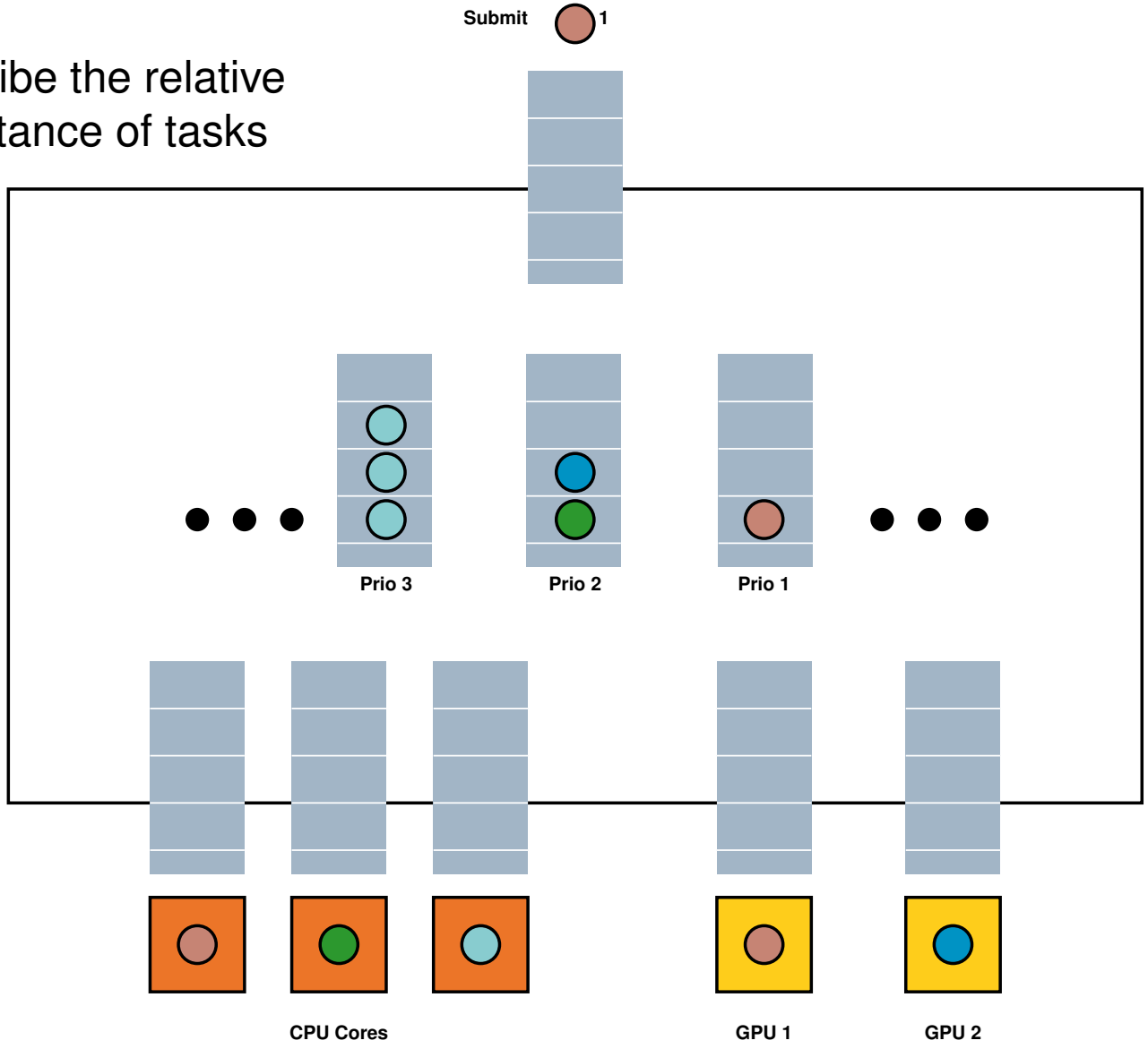
The Prio Scheduler

- Describe the relative importance of tasks



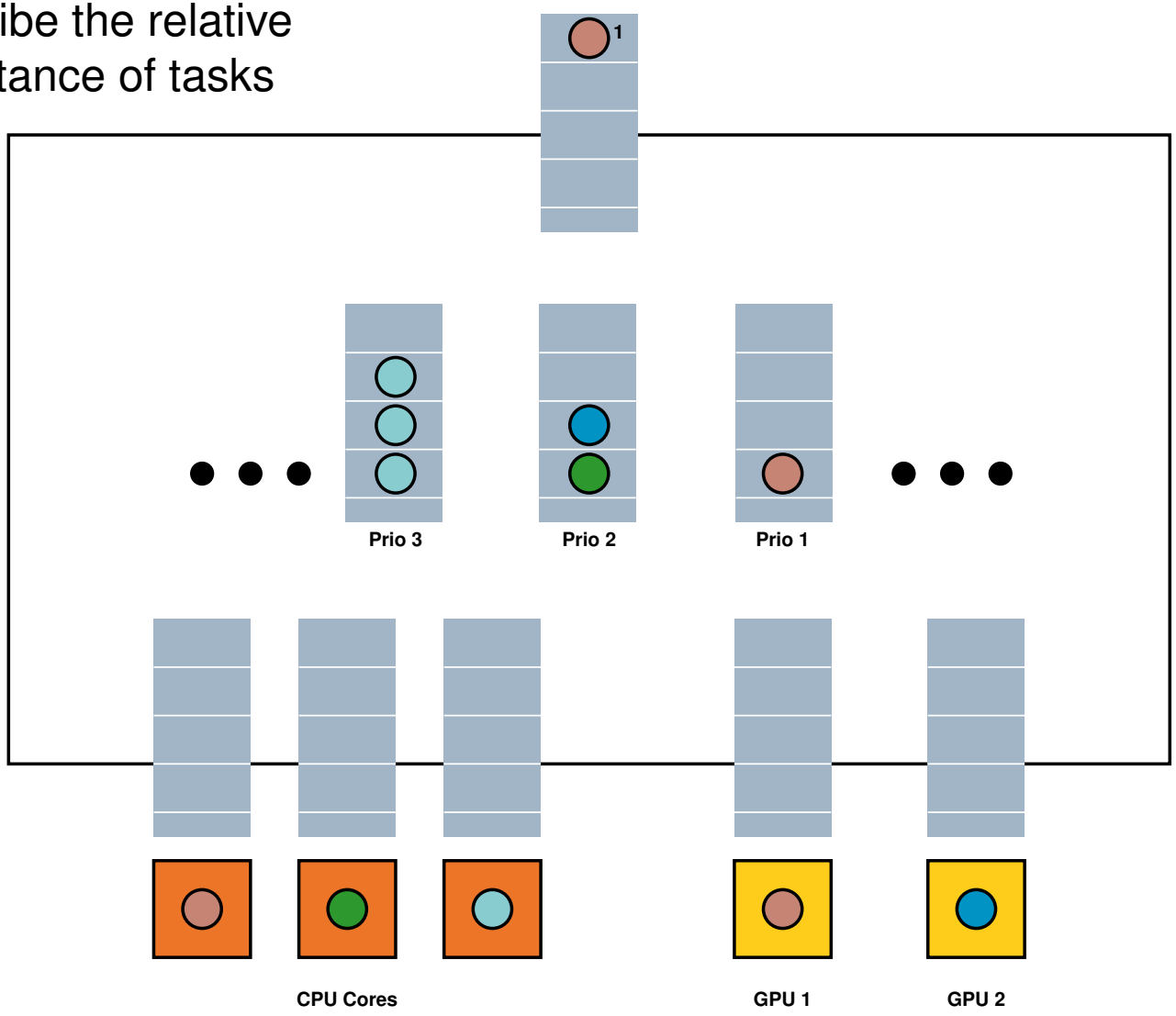
The Prio Scheduler

- Describe the relative importance of tasks



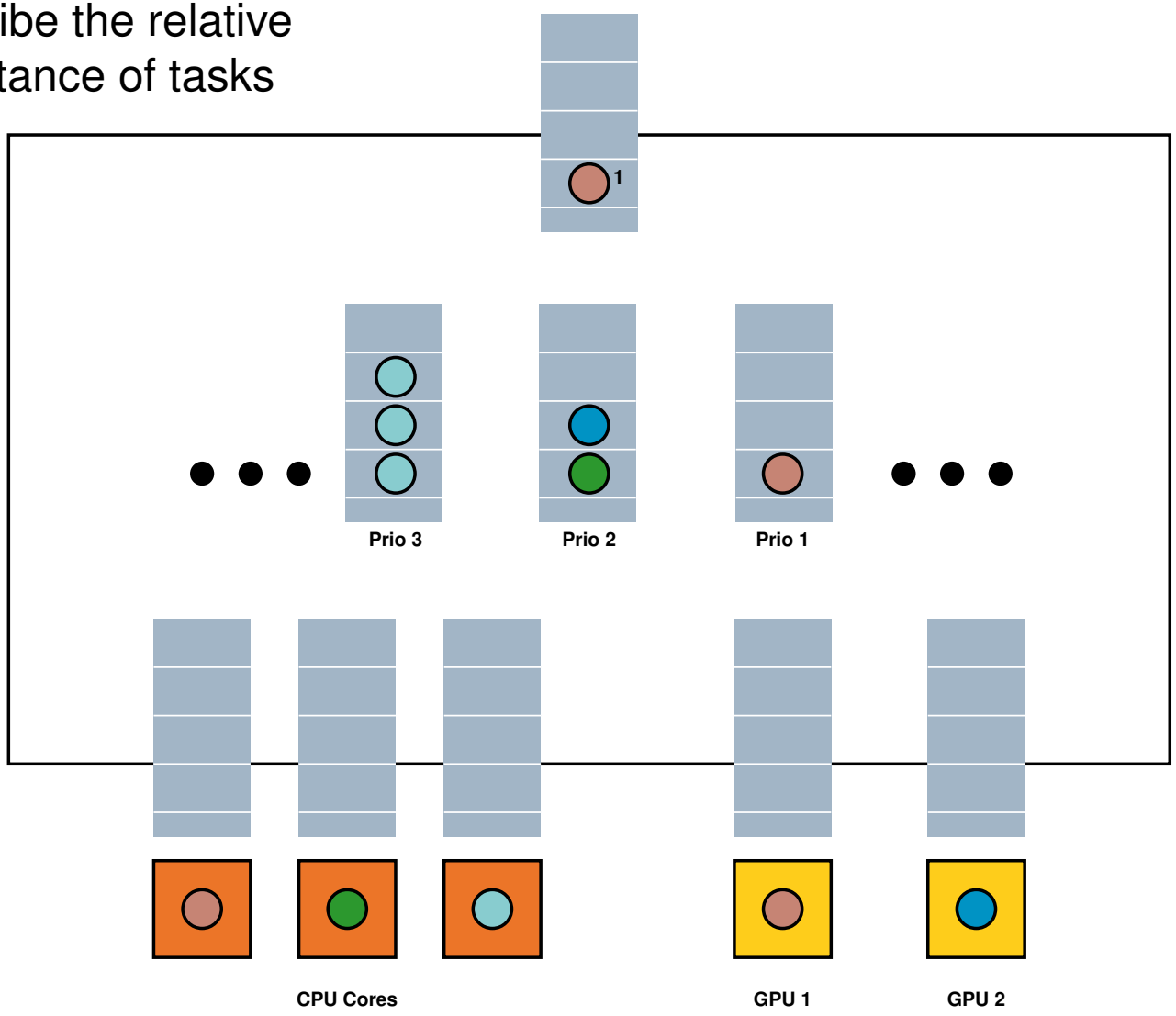
The Prio Scheduler

- Describe the relative importance of tasks



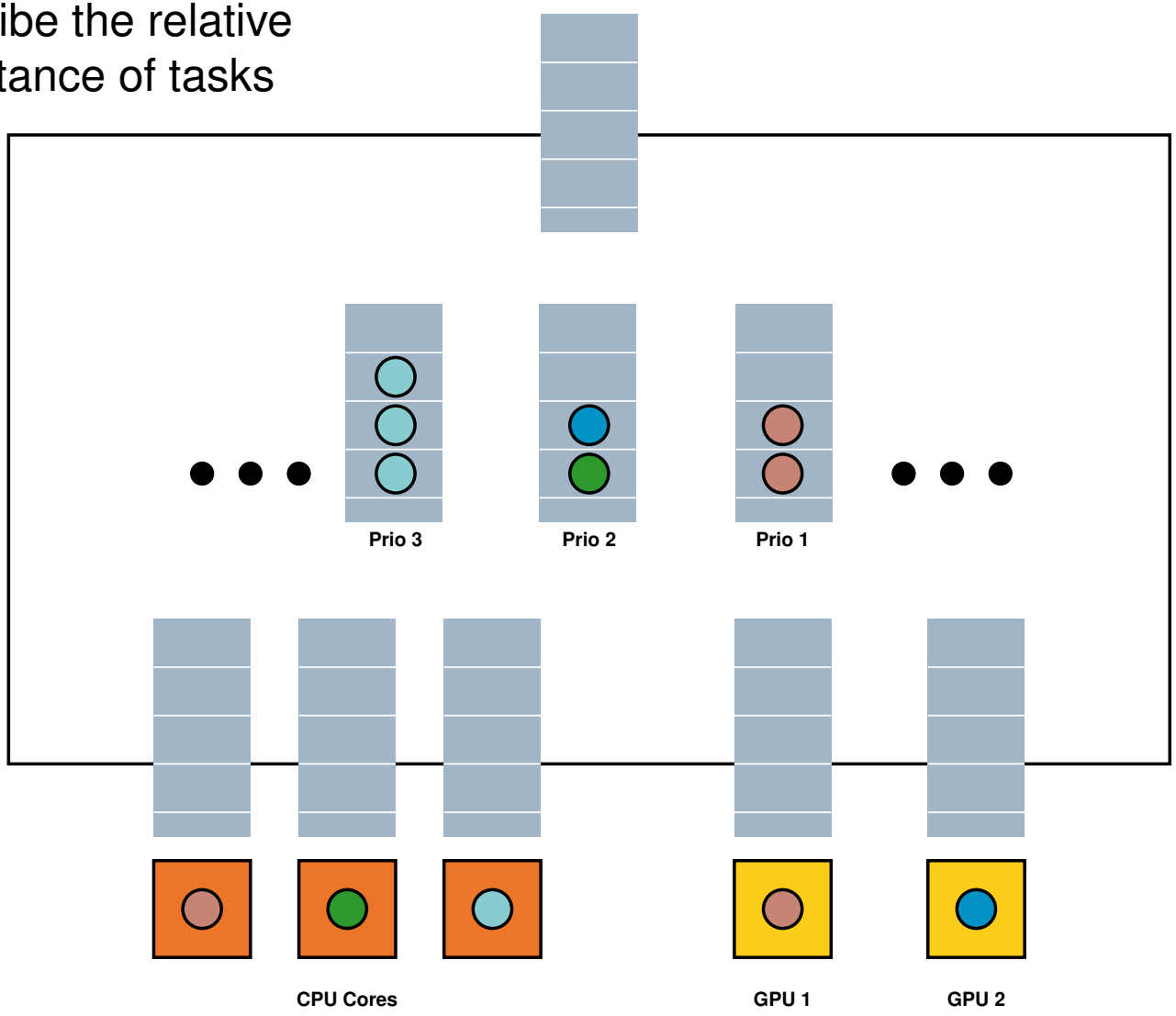
The Prio Scheduler

- Describe the relative importance of tasks



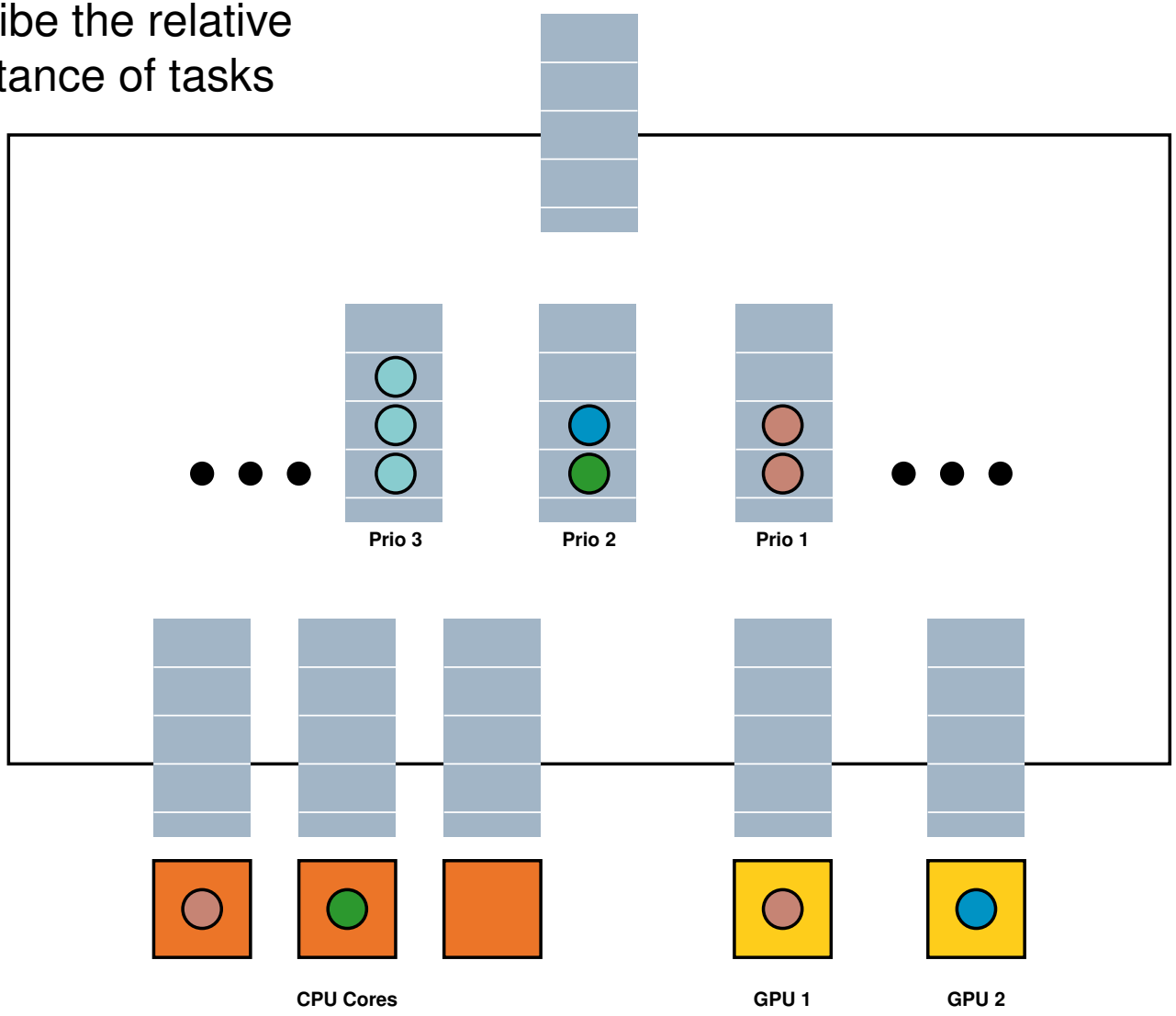
The Prio Scheduler

- Describe the relative importance of tasks



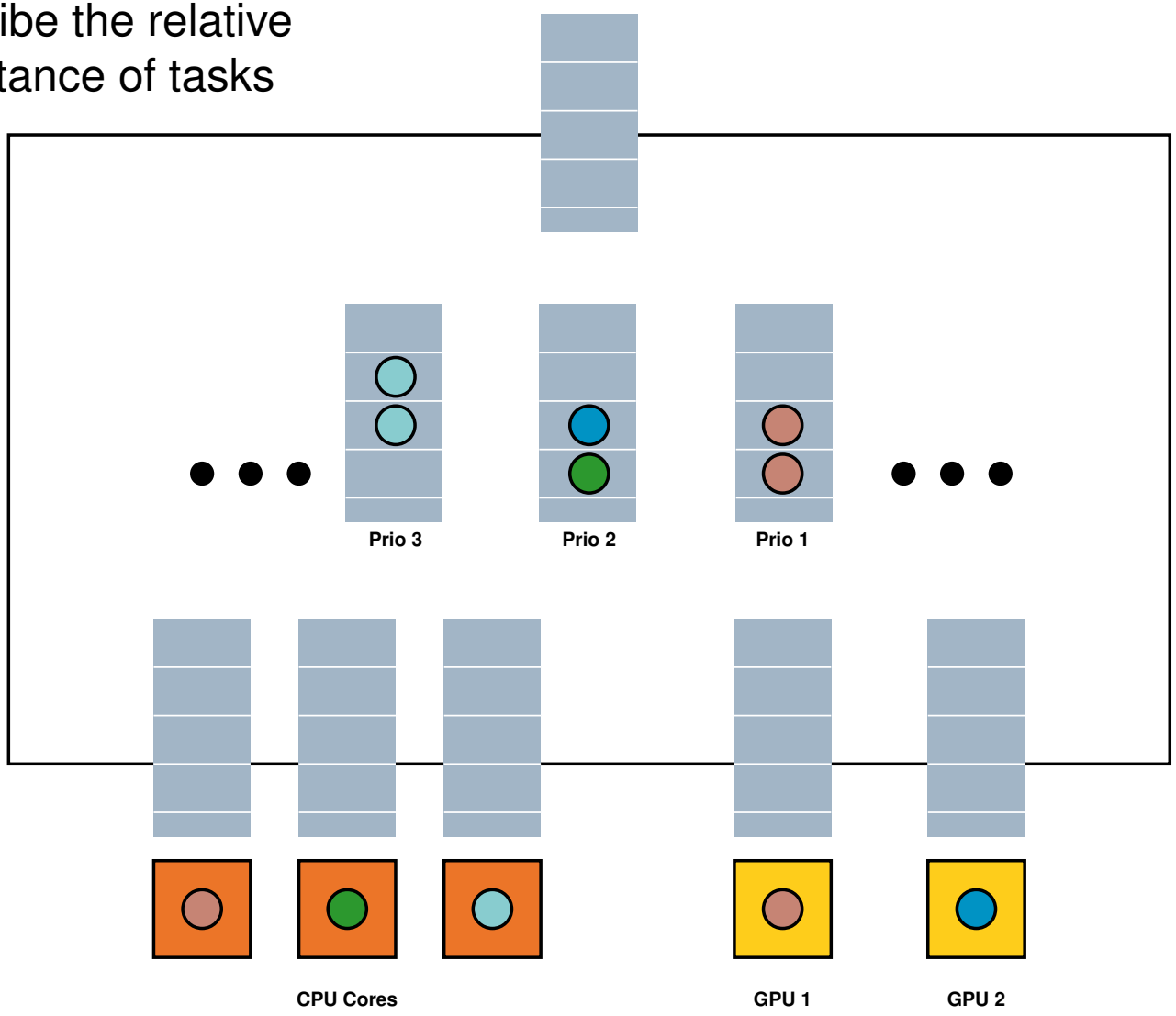
The Prio Scheduler

- Describe the relative importance of tasks



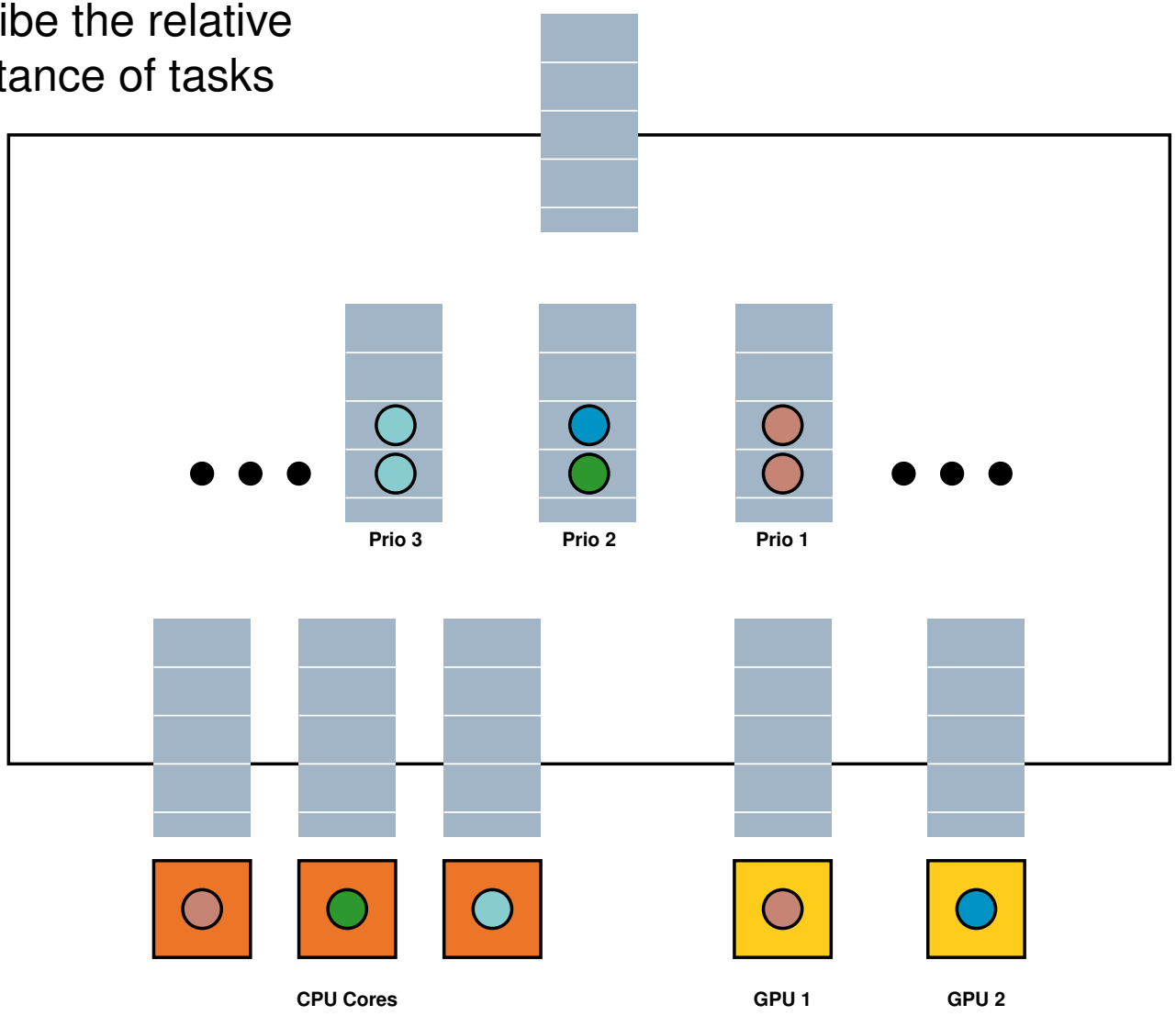
The Prio Scheduler

- Describe the relative importance of tasks



The Prio Scheduler

- Describe the relative importance of tasks



The Deque Model (dm) Scheduler

- Inspired by HEFT popular scheduling algorithm
  - Heterogeneous Earliest Finish Time
- Try to get the best from accelerators **and** CPUs

The Deque Model (dm) Scheduler

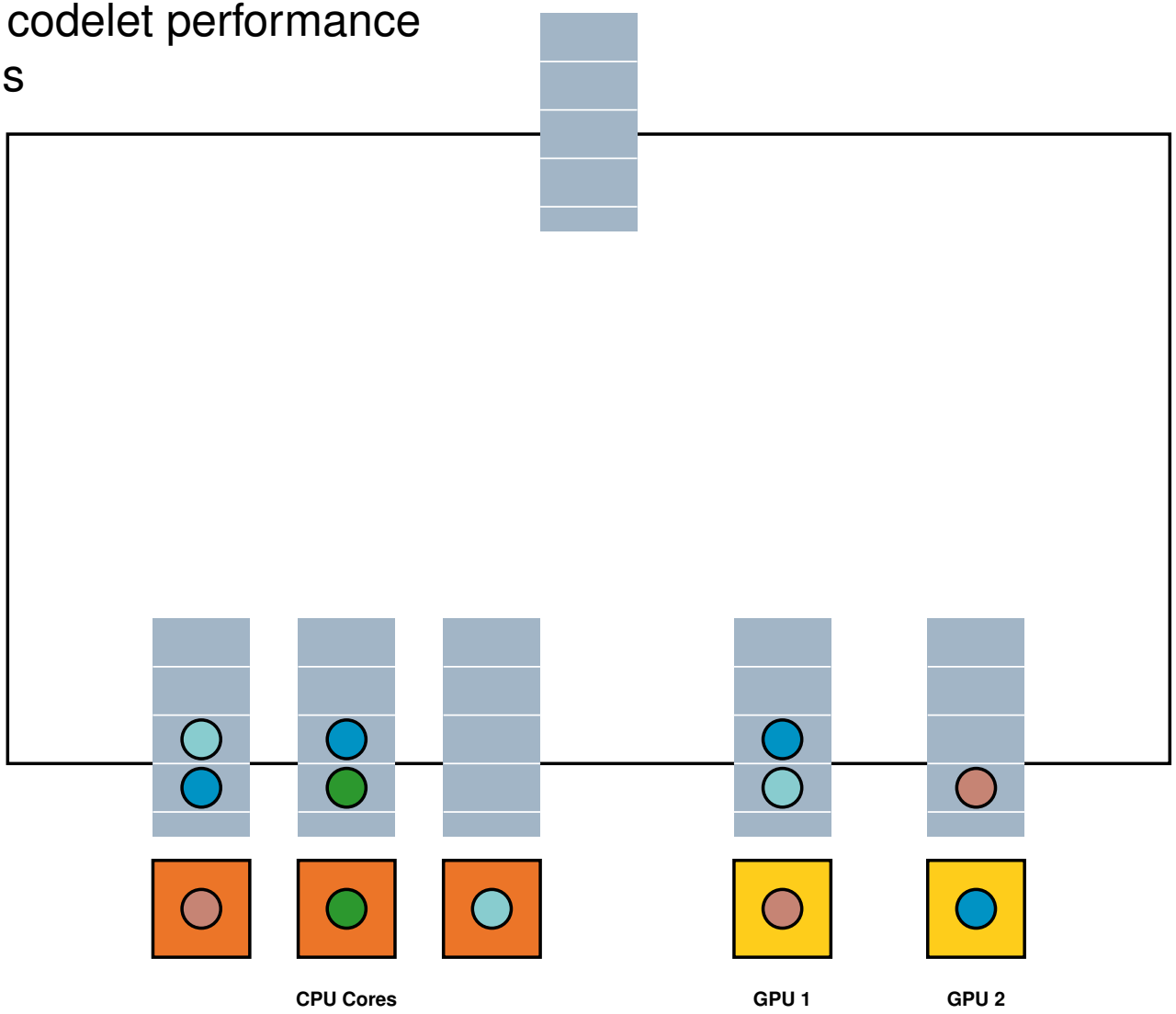
- Inspired by HEFT popular scheduling algorithm
  - Heterogeneous Earliest Finish Time
- Try to get the best from accelerators **and** CPUs
- Using codelet performance models
  - Kernel calibration on each available computing device
  - **Raw** history model of kernels' past execution times
  - **Refined** models using regression on kernels' execution times history

The Deque Model (dm) Scheduler

- Inspired by HEFT popular scheduling algorithm
  - Heterogeneous Earliest Finish Time
- Try to get the best from accelerators **and** CPUs
- Using codelet performance models
  - Kernel calibration on each available computing device
  - **Raw** history model of kernels' past execution times
  - **Refined** models using regression on kernels' execution times history
- Model parameter
  - **Data size** by default
  - **User-defined** for more complex cases
    - Sparse data structures
    - Iteratives kernels

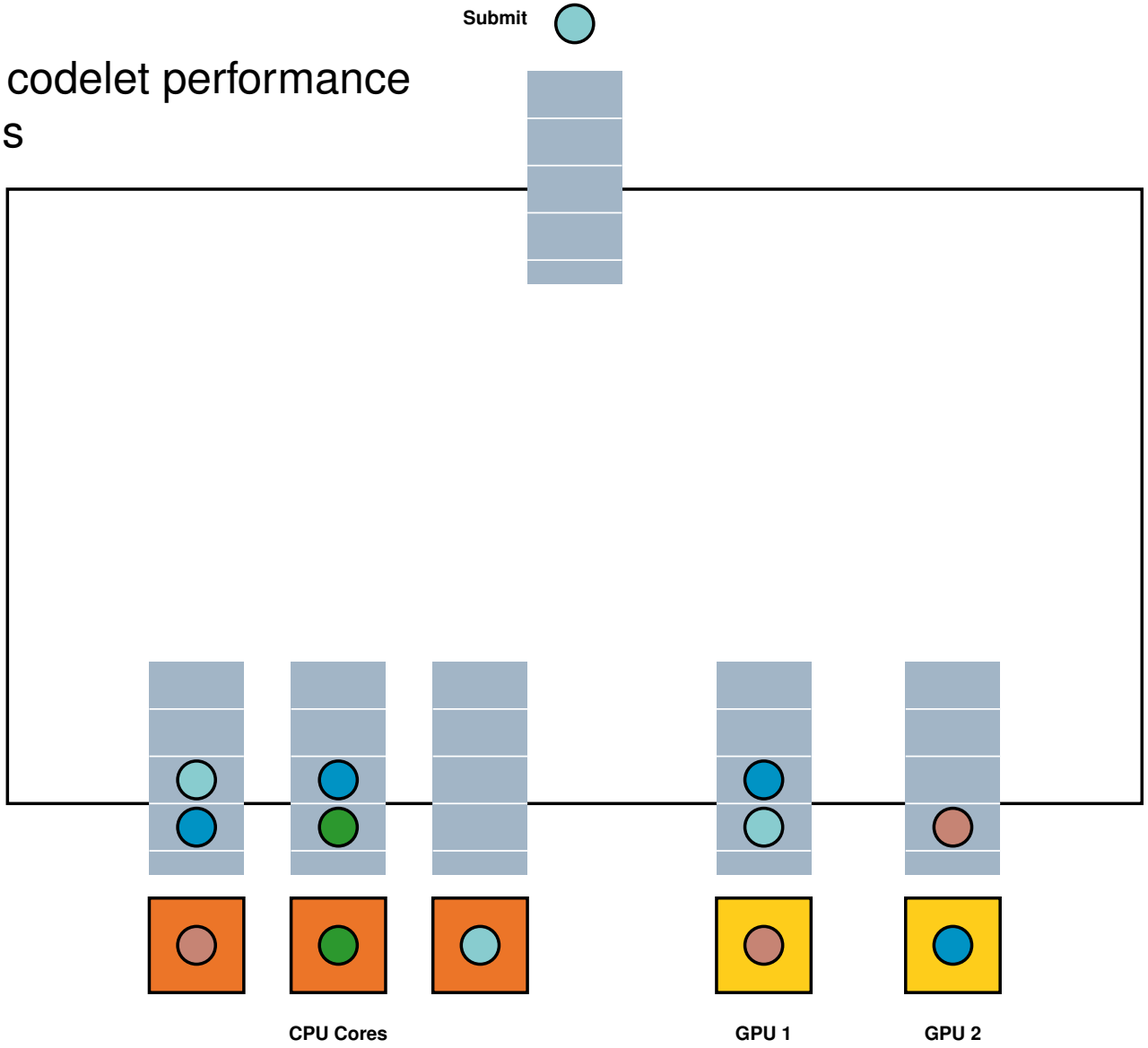
The Deque Model (dm) Scheduler

- Using codelet performance models



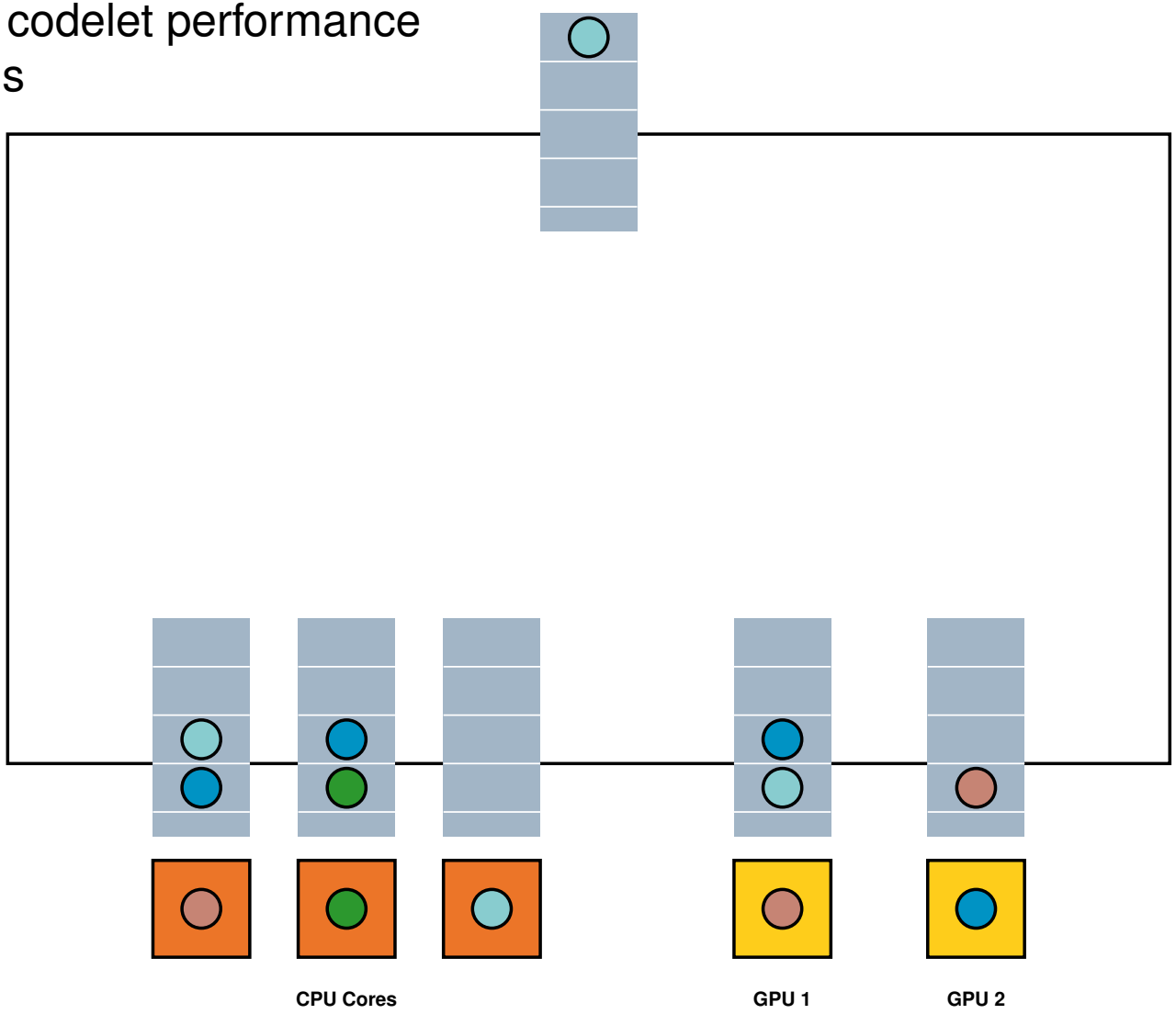
The Deque Model (dm) Scheduler

- Using codelet performance models



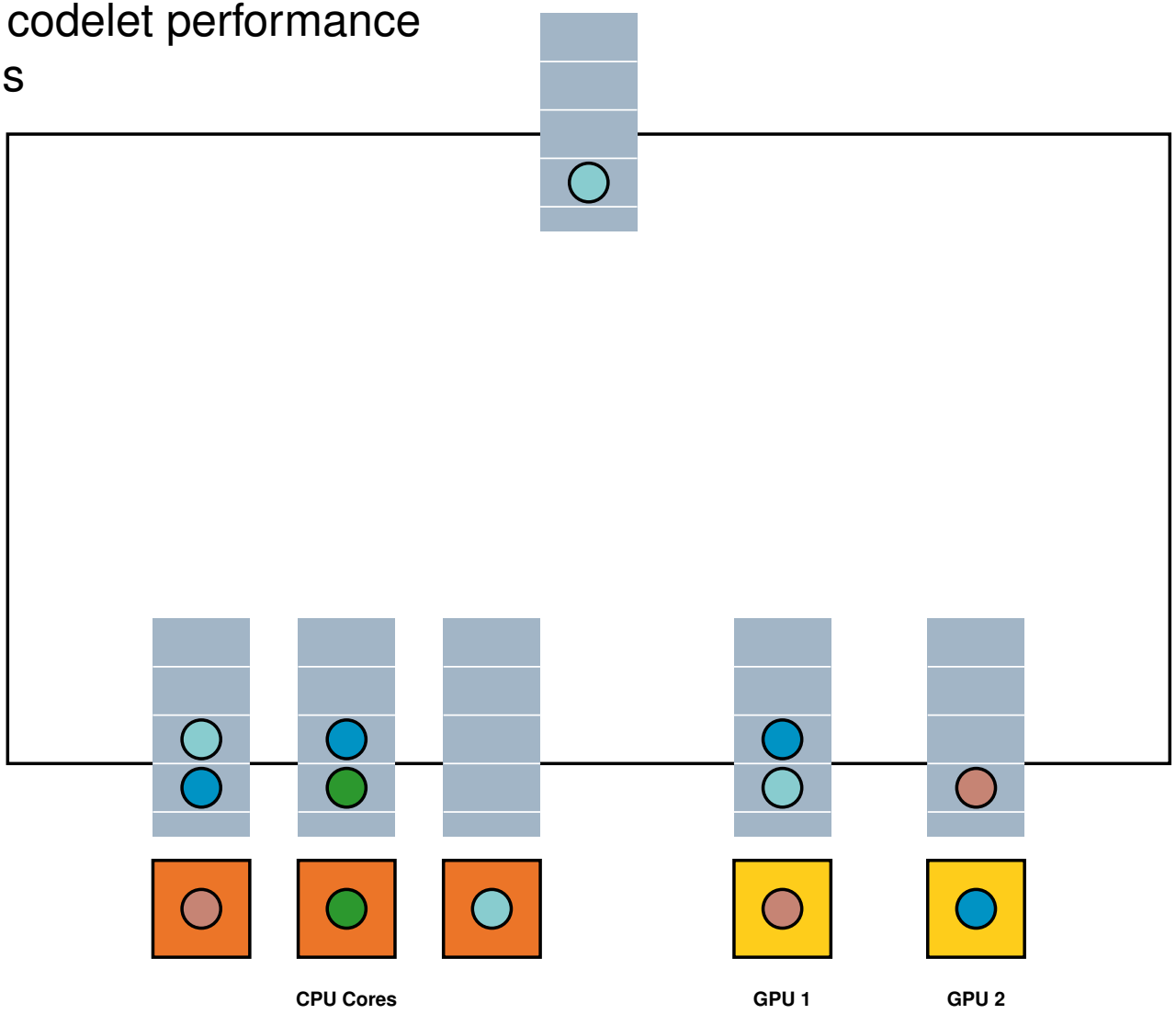
The Deque Model (dm) Scheduler

- Using codelet performance models



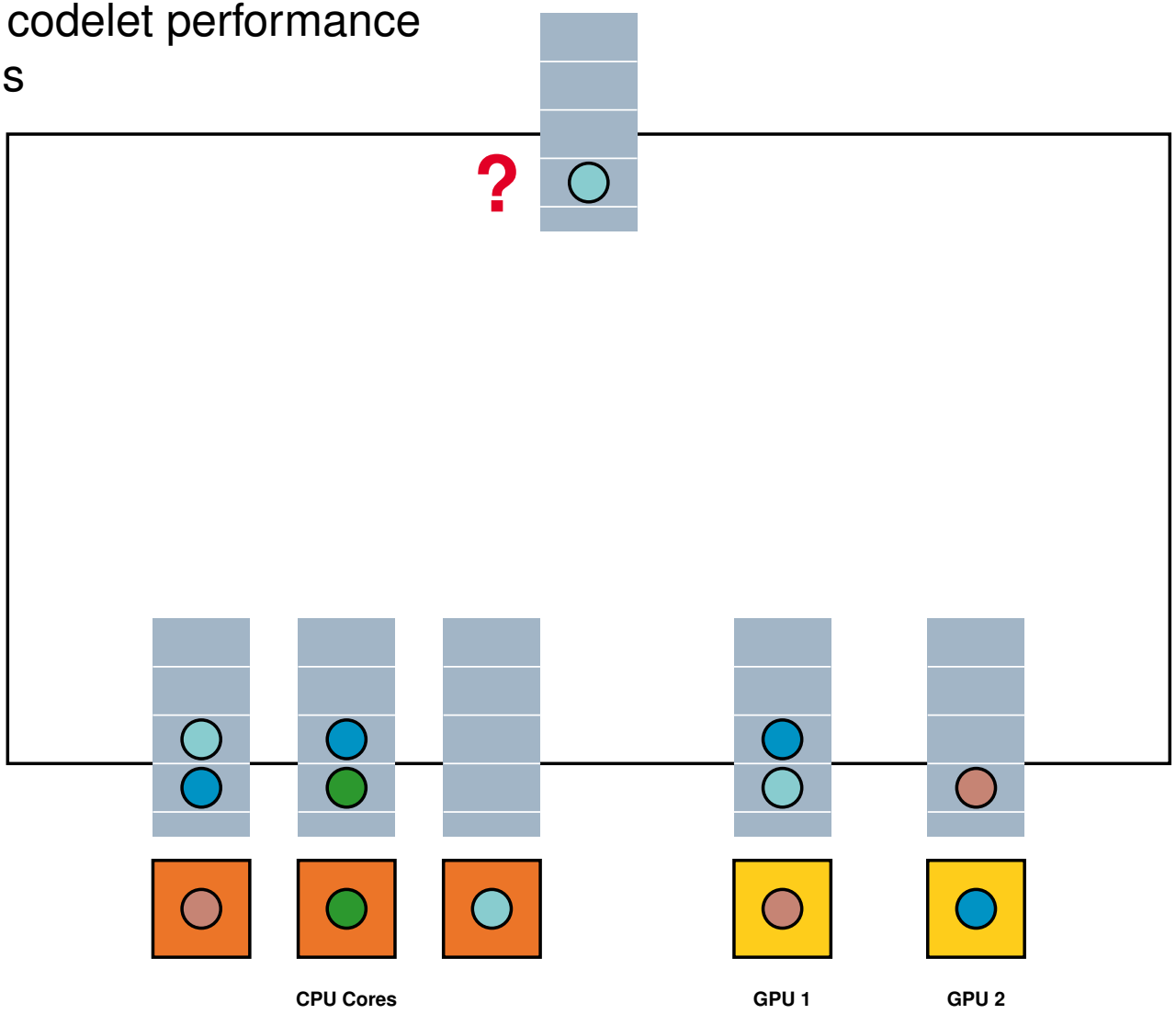
The Deque Model (dm) Scheduler

- Using codelet performance models



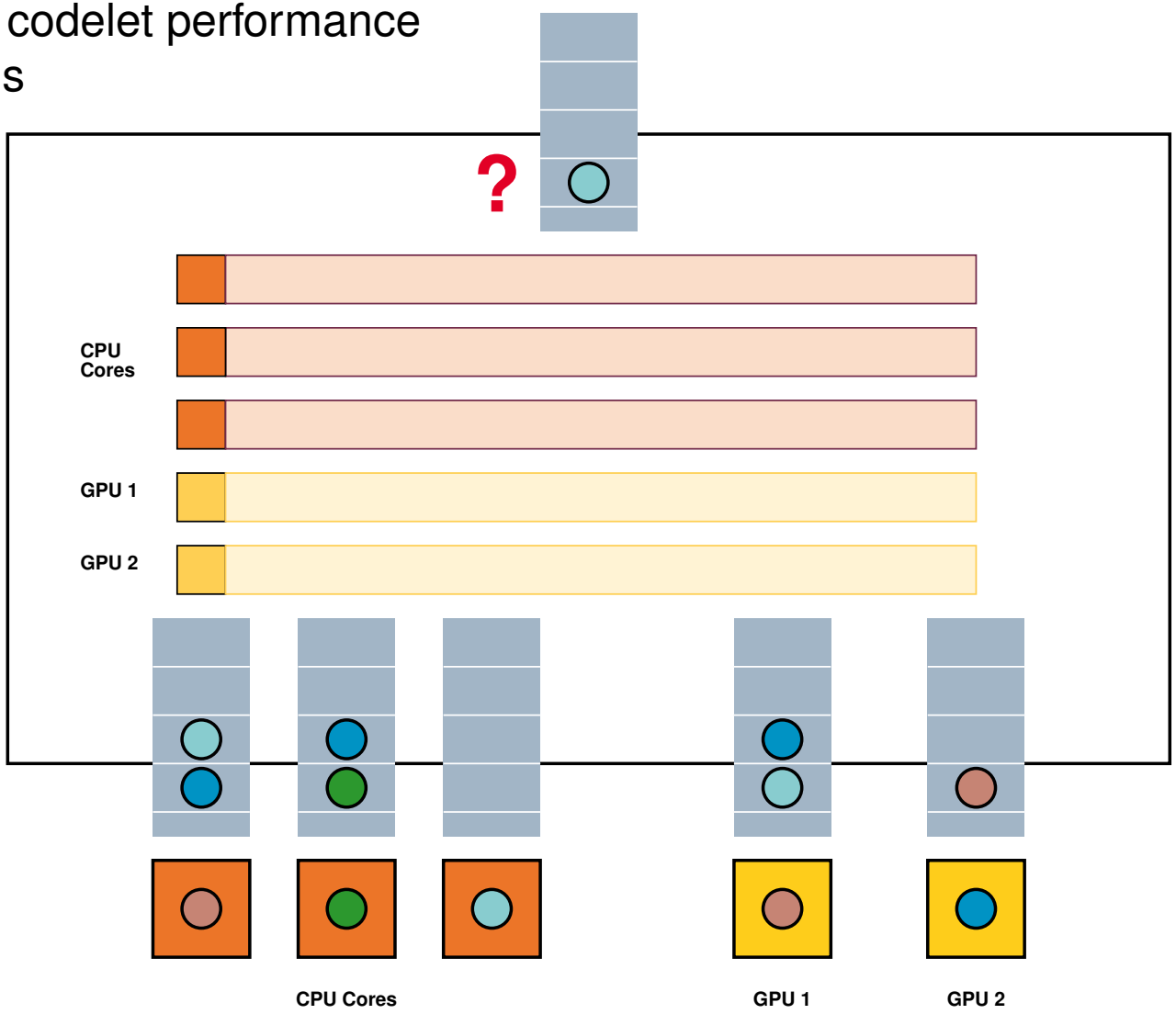
The Deque Model (dm) Scheduler

- Using codelet performance models



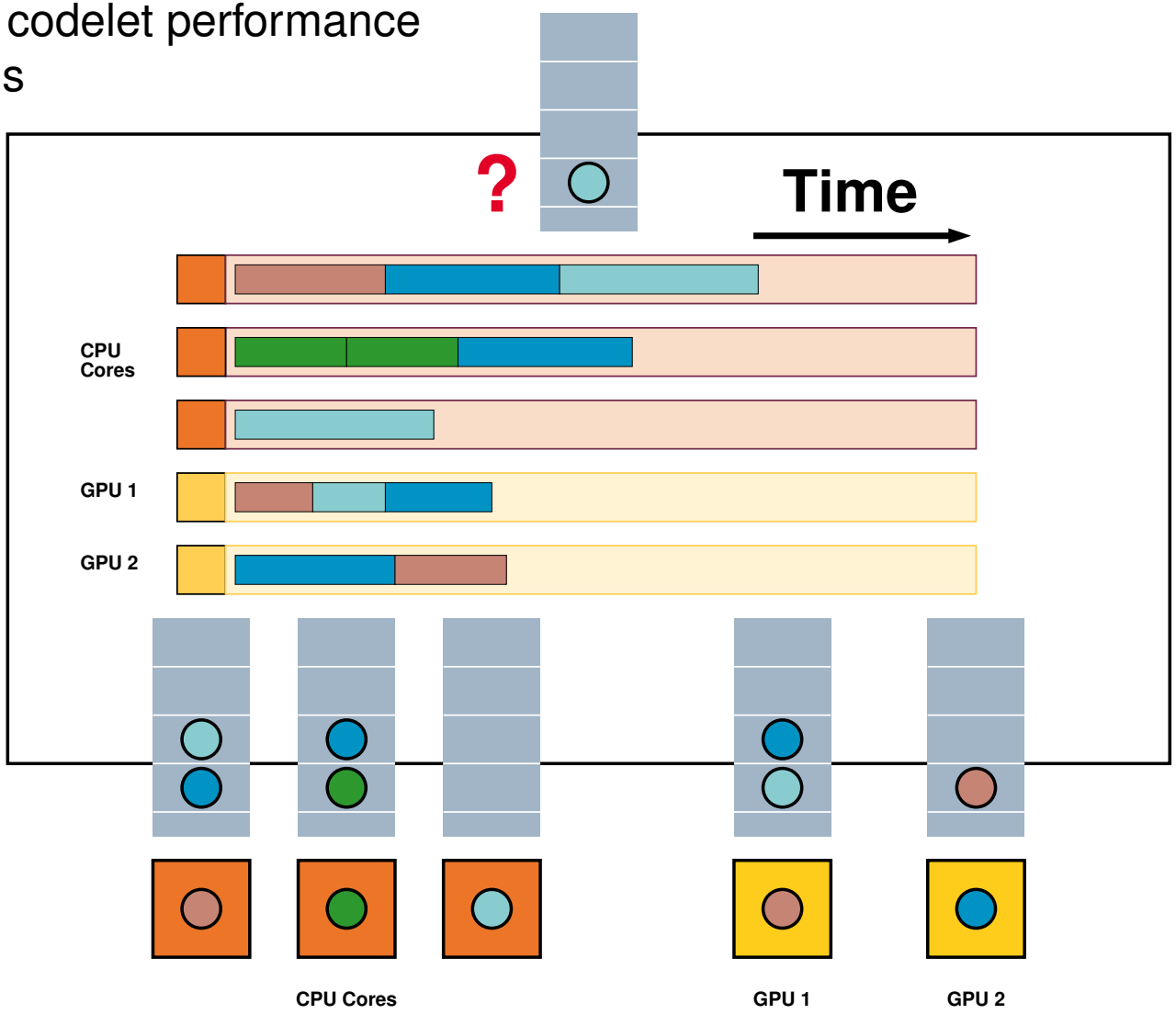
The Deque Model (dm) Scheduler

- Using codelet performance models



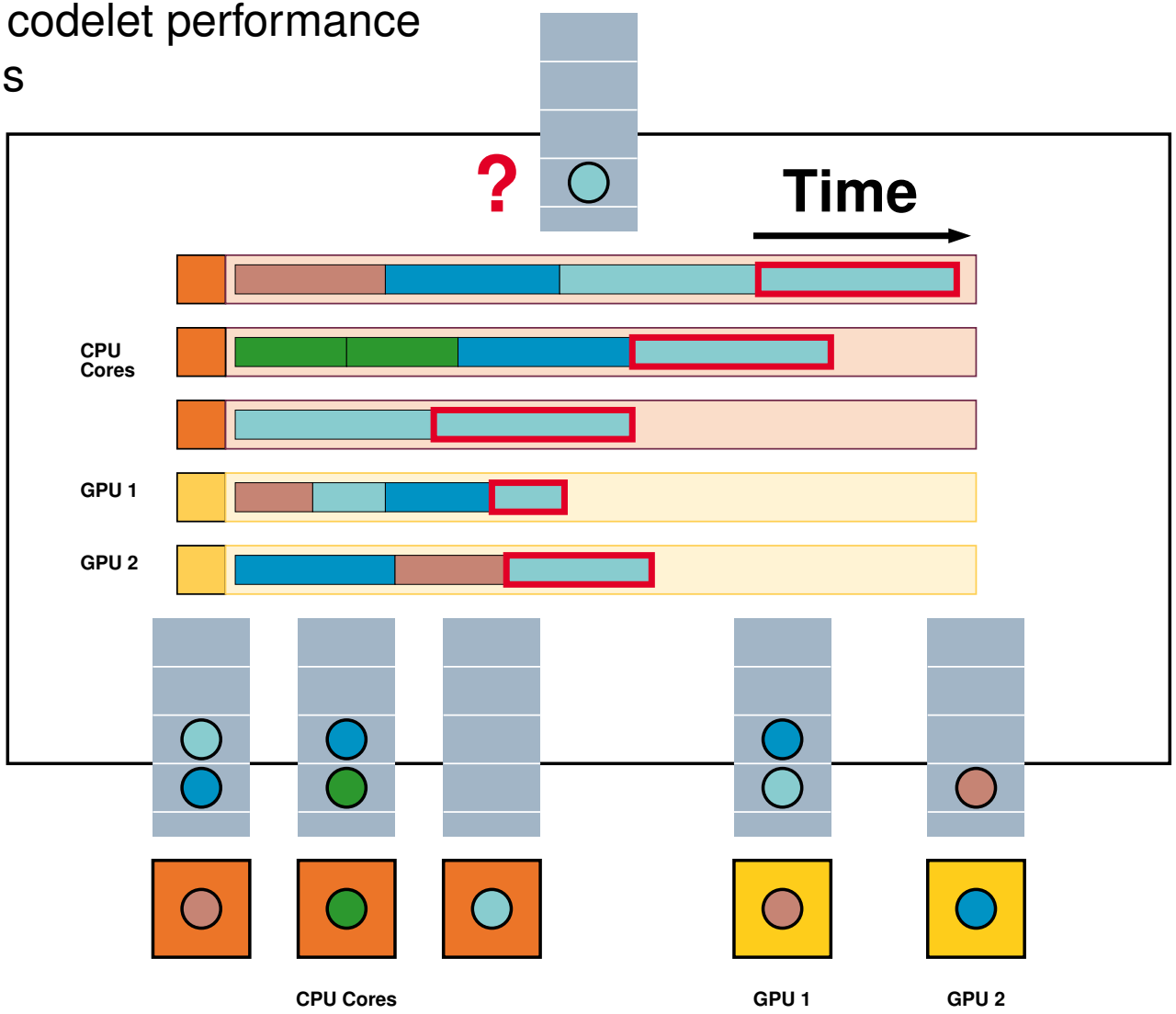
The Deque Model (dm) Scheduler

- Using codelet performance models



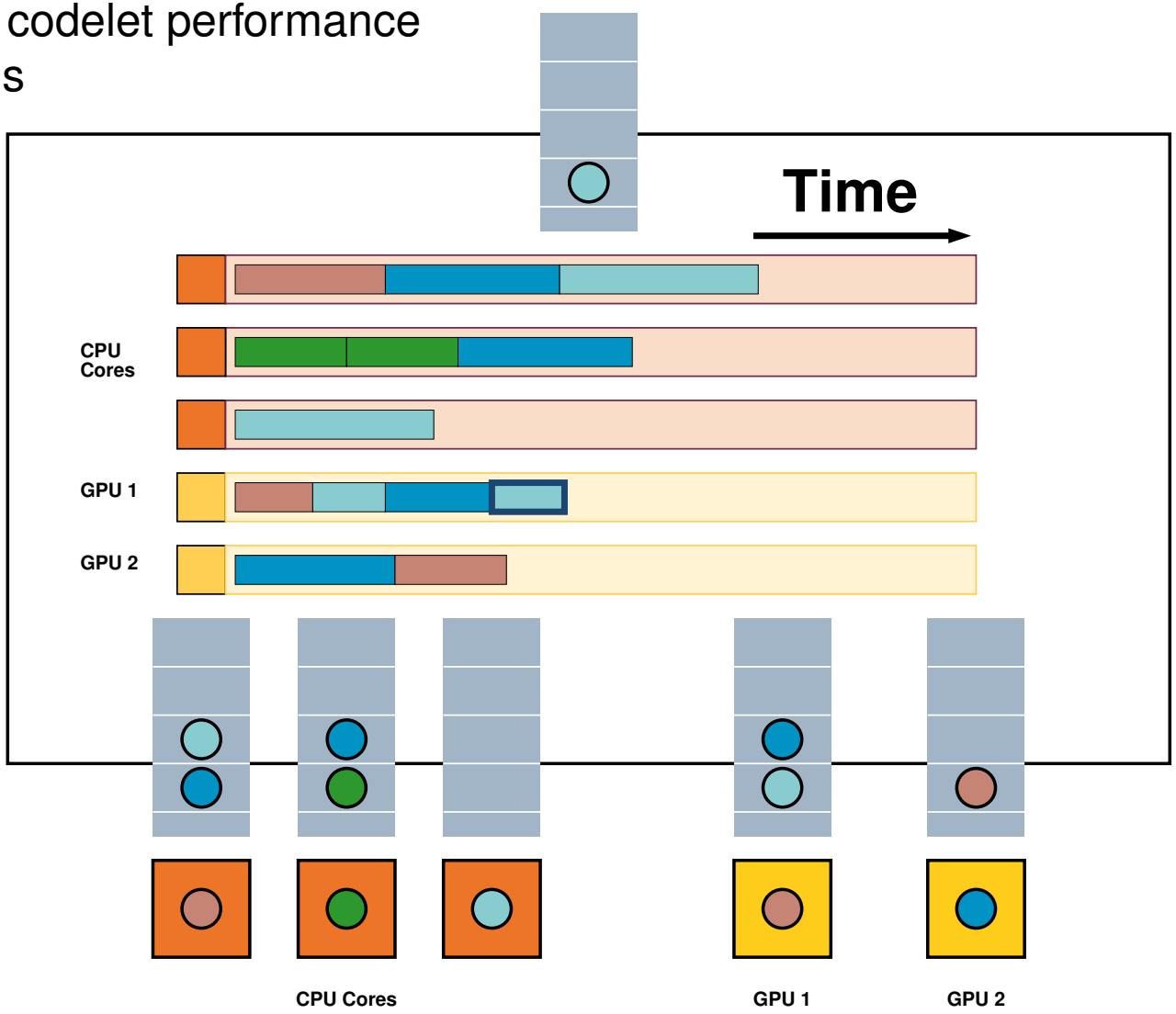
The Deque Model (dm) Scheduler

- Using codelet performance models



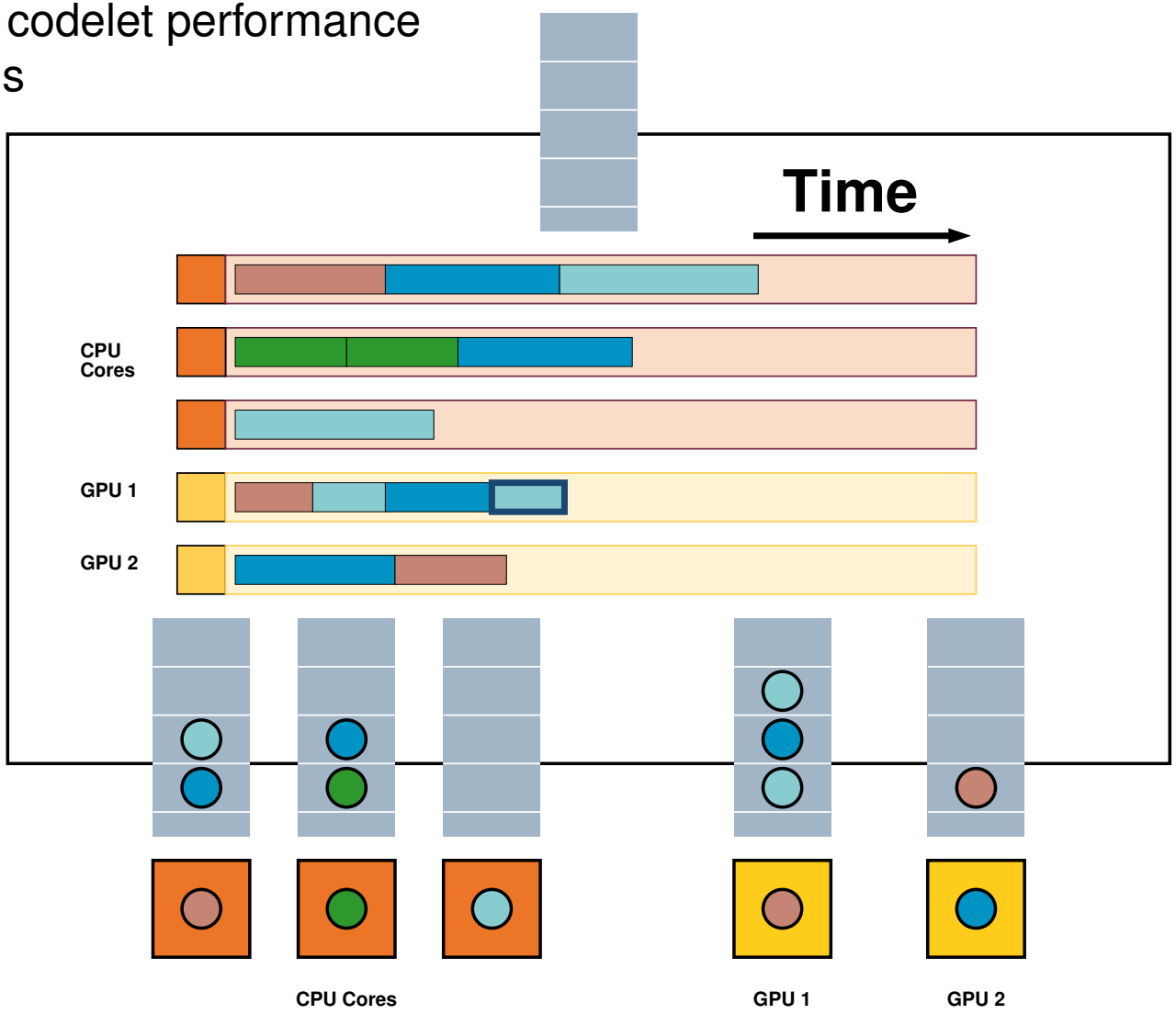
The Deque Model (dm) Scheduler

- Using codelet performance models



The Deque Model (dm) Scheduler

- Using codelet performance models



Selecting a Scheduling Policy

- Use the `STARPU_SCHED` environment variable

Selecting a Scheduling Policy

- Use the STARPU_SCHED environment variable
- Example 1: selecting the prio scheduler

```
$ export STARPU_SCHED=prio
$ my_program
...
```

Selecting a Scheduling Policy

- Use the STARPU_SCHED environment variable
- Example 1: selecting the prio scheduler
- Example 2: selecting the dm scheduler

```
$ export STARPU_SCHED=prio
$ my_program
...
```

```
$ export STARPU_SCHED=dm
$ my_program
...
```

Selecting a Scheduling Policy

- Use the STARPU_SCHED environment variable
- Example 1: selecting the prio scheduler
- Example 2: selecting the dm scheduler
- Example 3: resetting to default scheduler eager

```
$ export STARPU_SCHED=prio
$ my_program
...
```

```
$ export STARPU_SCHED=dm
$ my_program
...
```

```
$ unset STARPU_SCHED
$ my_program
...
```

Selecting a Scheduling Policy

- Use the STARPU_SCHED environment variable
- Example 1: selecting the prio scheduler
- Example 2: selecting the dm scheduler
- Example 3: resetting to default scheduler eager
- No need to recompile the application

```
$ export STARPU_SCHED=prio
$ my_program
...
```

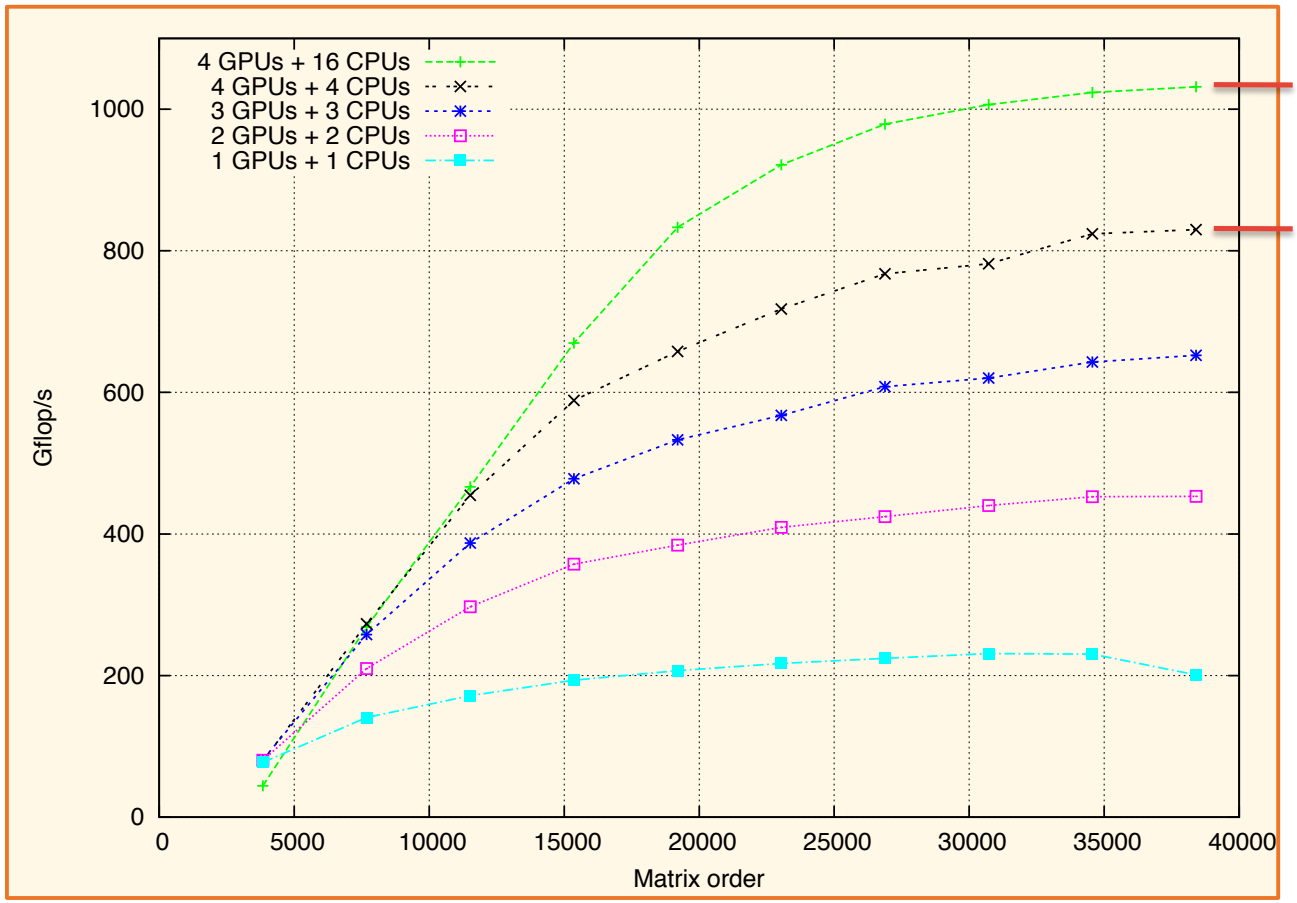
```
$ export STARPU_SCHED=dm
$ my_program
...
```

```
$ unset STARPU_SCHED
$ my_program
...
```

Showcase with the MAGMA Linear Algebra Library

University of Tennessee, INRIA HIEPACS, INRIA RUNTIME

- QR decomposition on 16 CPUs (AMD) + 4 GPUs (C1060)



Measured increase:
+12 CPUs
~200 GFlops

Expected increase:
+12 CPUs
~150 Gflops

Showcase with the MAGMA Linear Algebra Library

QR kernel properties

Kernel SGEQRT					
CPU:	9 GFlop/s	GPU:	30 GFlop/s	Speed-up:	3
Kernel STSQRT					
CPU:	12 GFlop/s	GPU:	37 GFlop/s	Speed-up:	3
Kernel SOMQRT					
CPU:	8.5 GFlop/s	GPU:	227 GFlop/s	Speed-up:	27
Kernel SSSMQ					
CPU:	10 GFlop/s	GPU:	285 GFlop/s	Speed-up:	28

Consequences

- Task distribution
  - SGEQRT: 20% Tasks on GPU
  - SSSMQ: 92% tasks on GPU
- Taking advantage of heterogeneity!
  - Only do what you are good for
  - Don't do what you are not good for

Beyond StarPU's Predefined Scheduling Policies

Predefined set of popular policies

- No *one size fits all* policy
- Selectable scheduling policy

Beyond StarPU's Predefined Scheduling Policies

Predefined set of popular policies

- No *one size fits all* policy
- Selectable scheduling policy

Extensible policy set

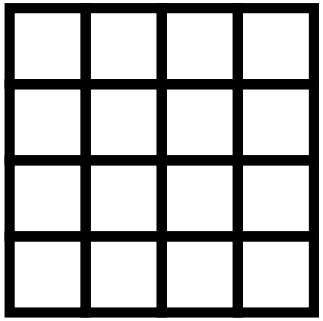
- You can write your own, specifically tailored policy
- Modular scheduler writing toolbox

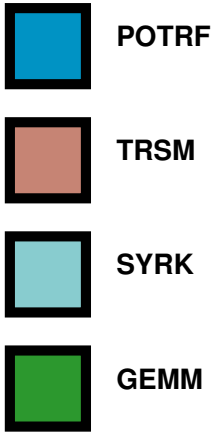
3

Data management

The Task-Based Cholesky Decomposition

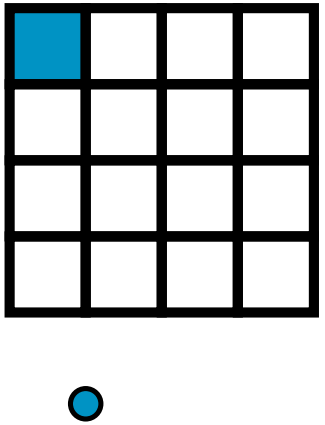
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

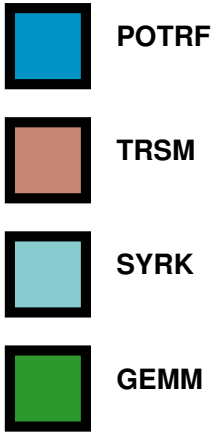




The Task-Based Cholesky Decomposition

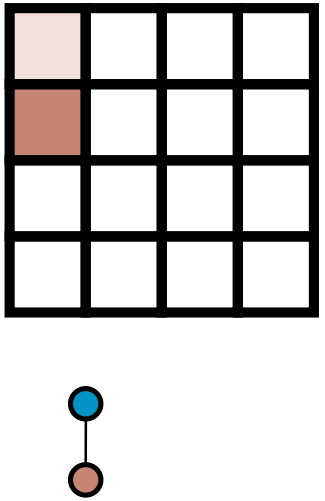
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

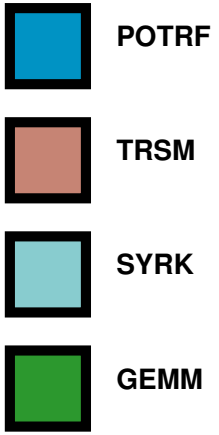




The Task-Based Cholesky Decomposition

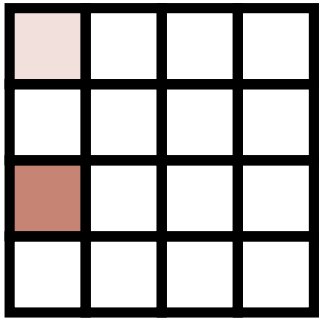
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

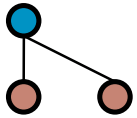


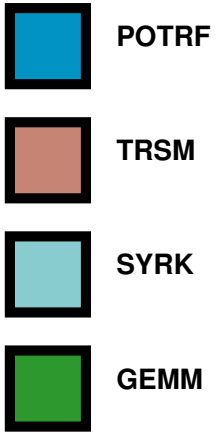


The Task-Based Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

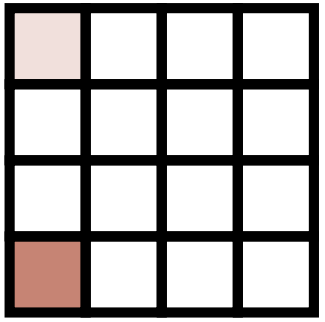


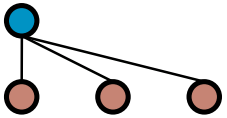


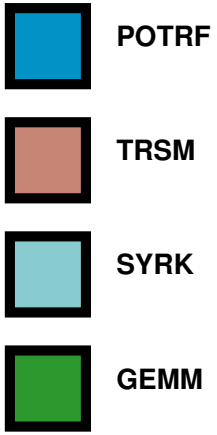


The Task-Based Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

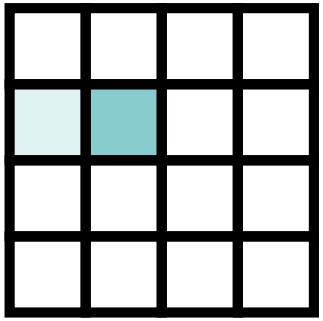


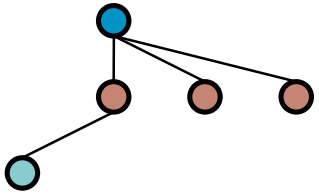


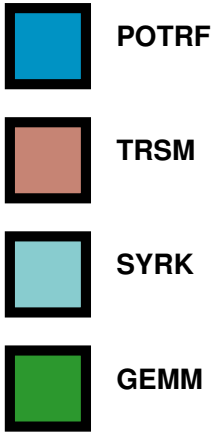


The Task-Based Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

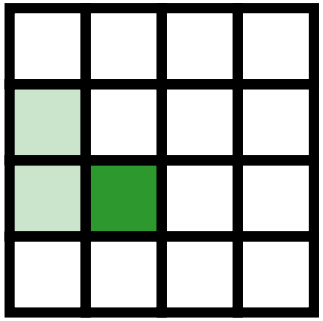


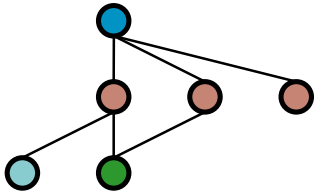


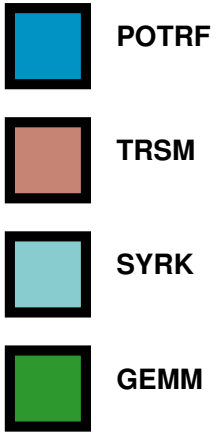


The Task-Based Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

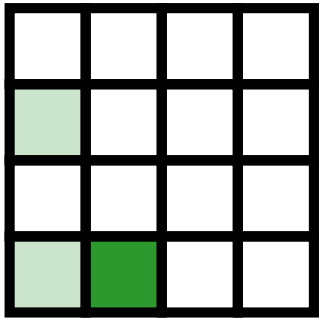


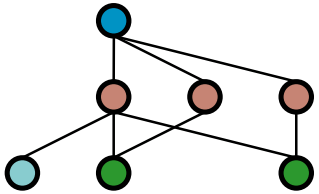


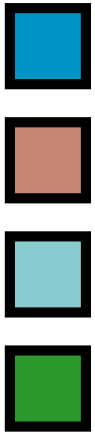


The Task-Based Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

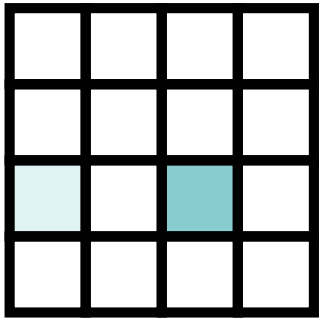


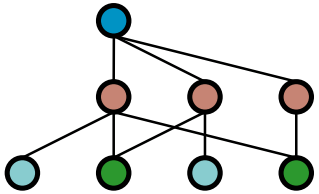


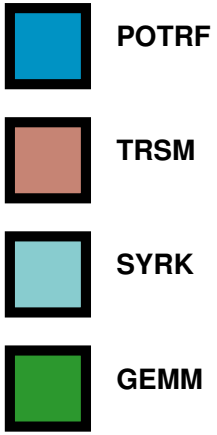


The Task-Based Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

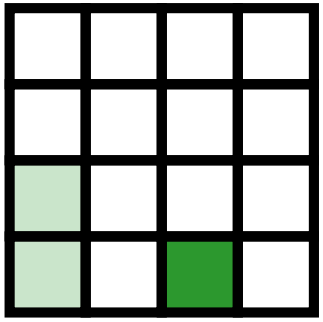


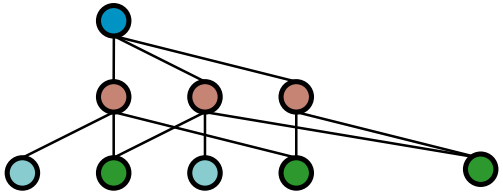


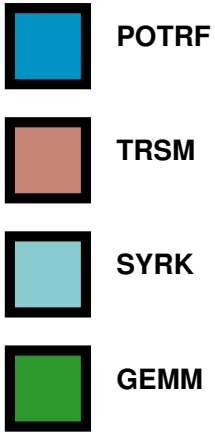


The Task-Based Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

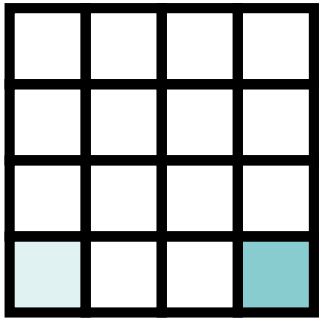


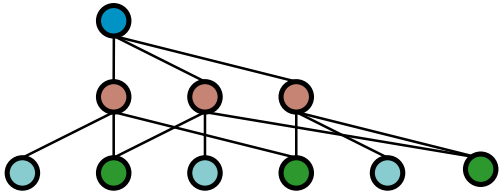


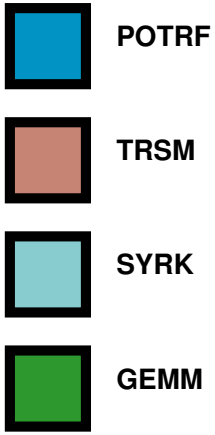


The Task-Based Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

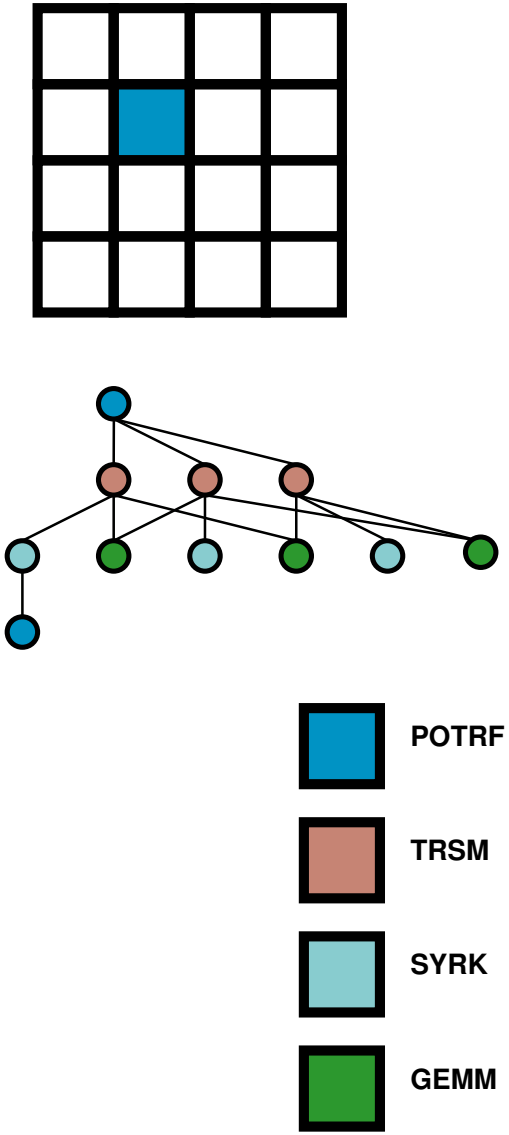






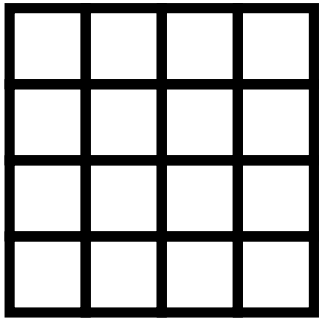
The Task-Based Cholesky Decomposition

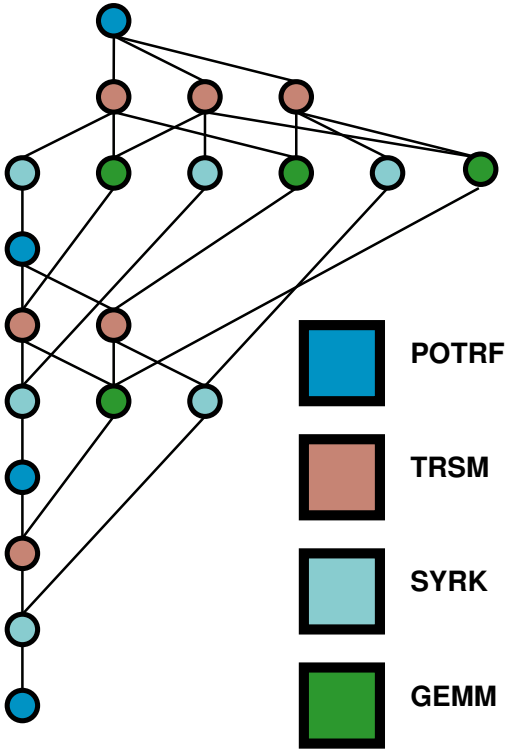
```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```



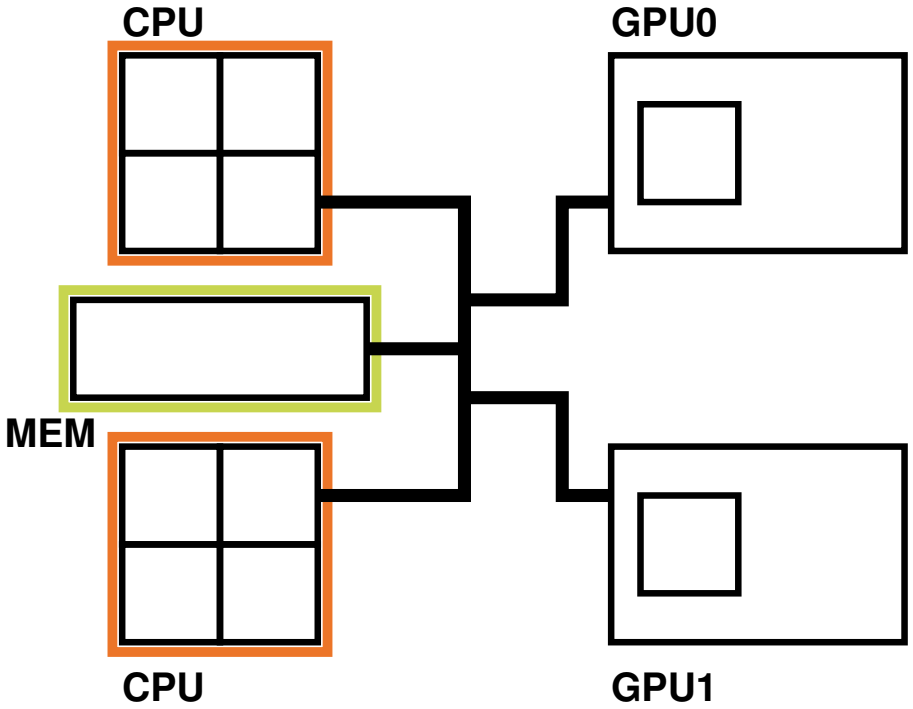
The Task-Based Cholesky Decomposition

```
for (j = 0; j < N; j++) {
  POTRF (RW,A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW,A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW,A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW,A[i][k],
            R,A[i][j], R,A[k][j]);
  }
}
starpu_wait_for_all();
```

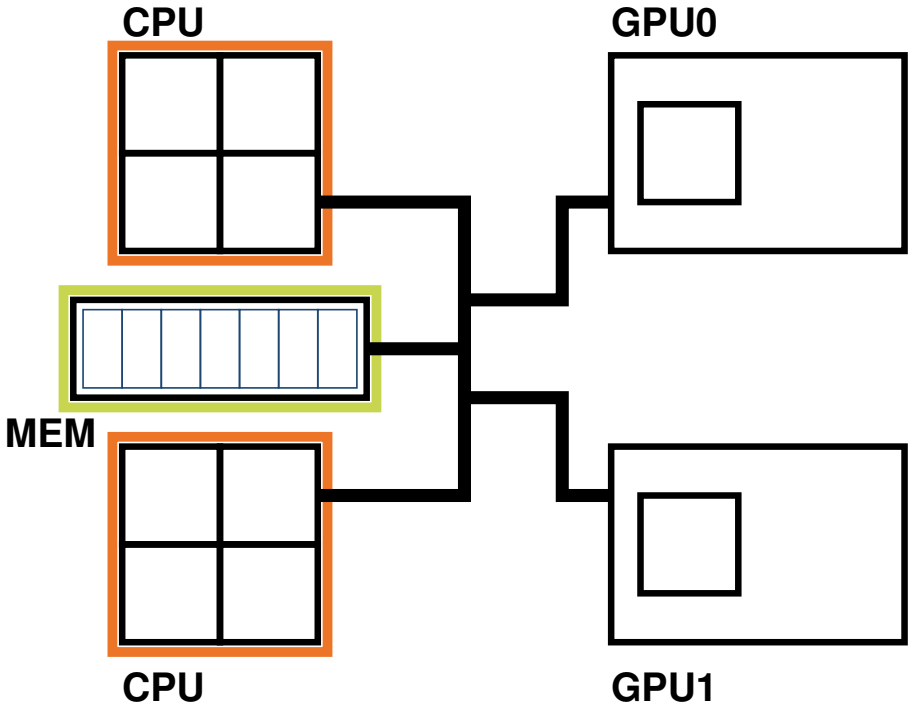


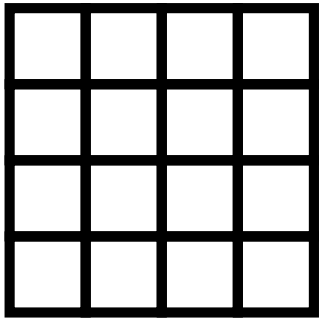


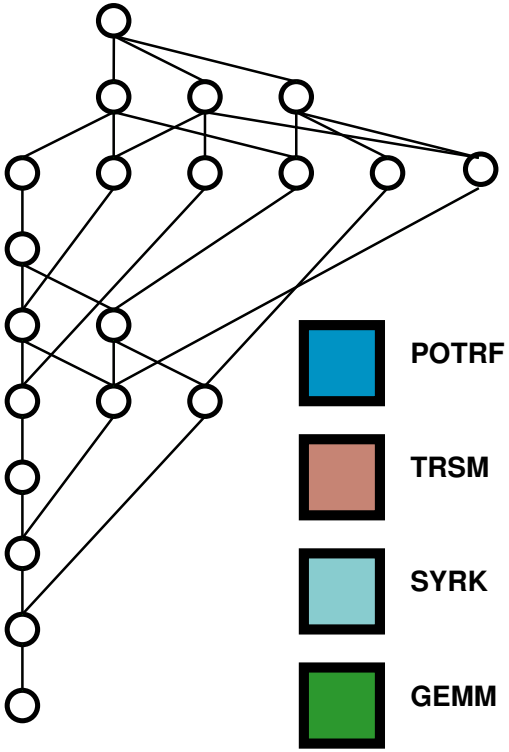
Runtime Parallel Execution on a Heterogeneous Node



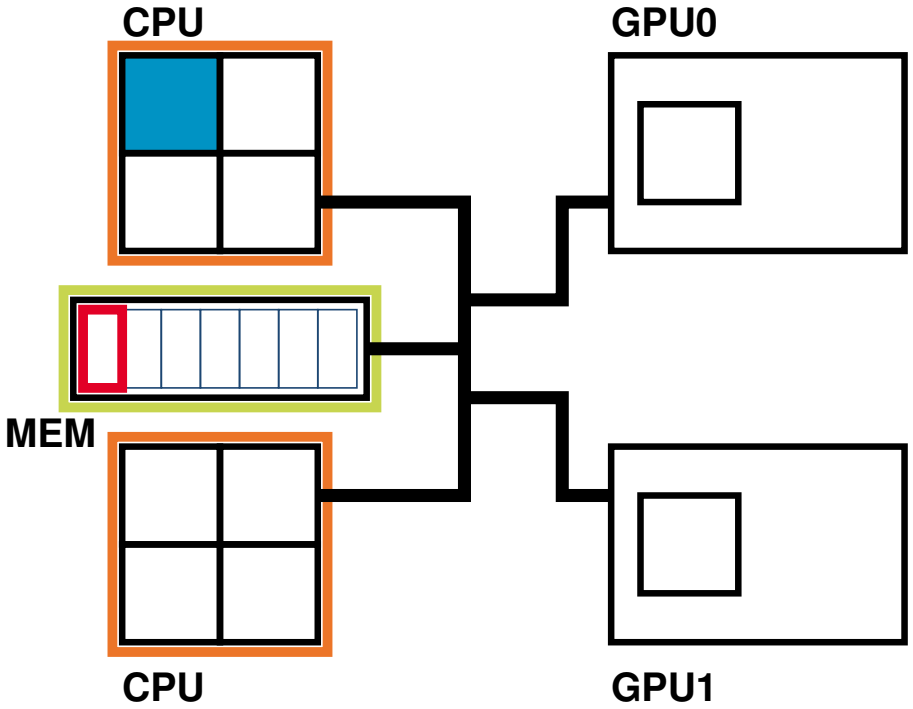
Runtime Parallel Execution on a Heterogeneous Node

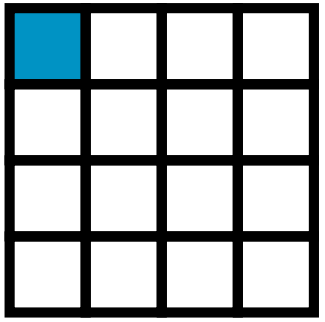


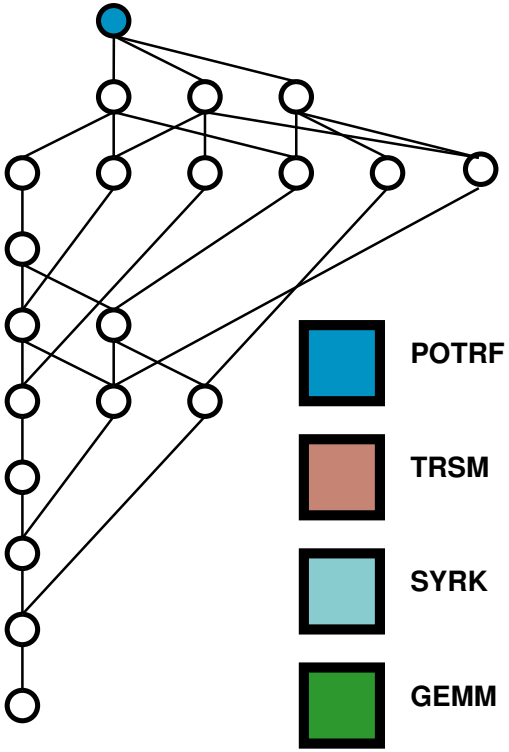




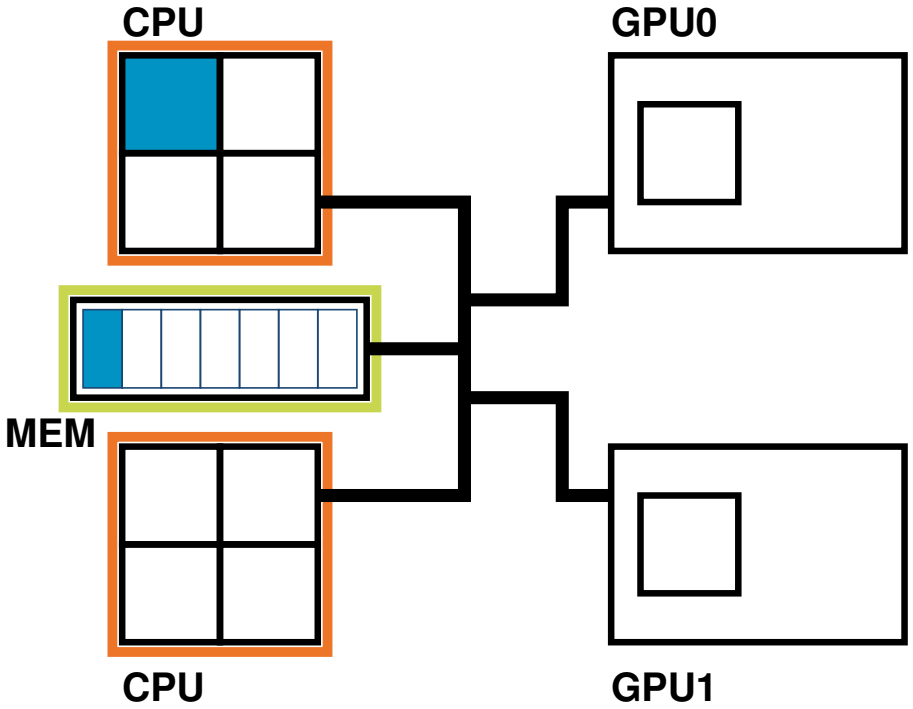
Runtime Parallel Execution on a Heterogeneous Node

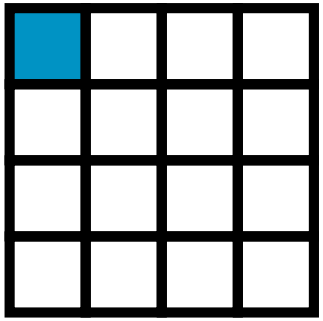


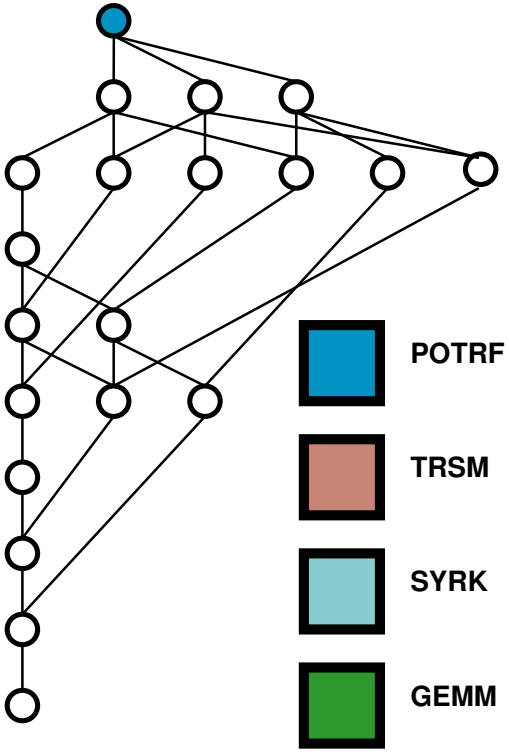




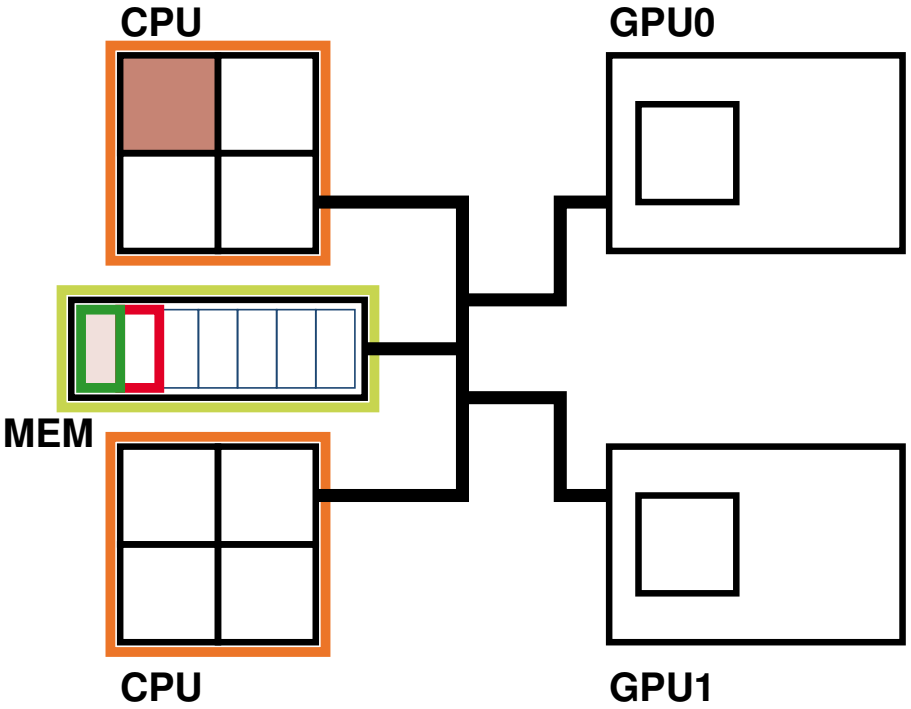
Runtime Parallel Execution on a Heterogeneous Node

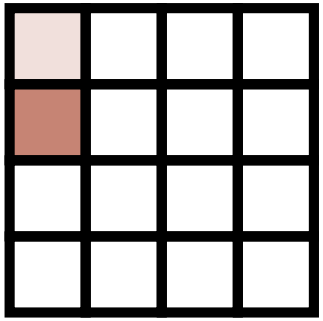


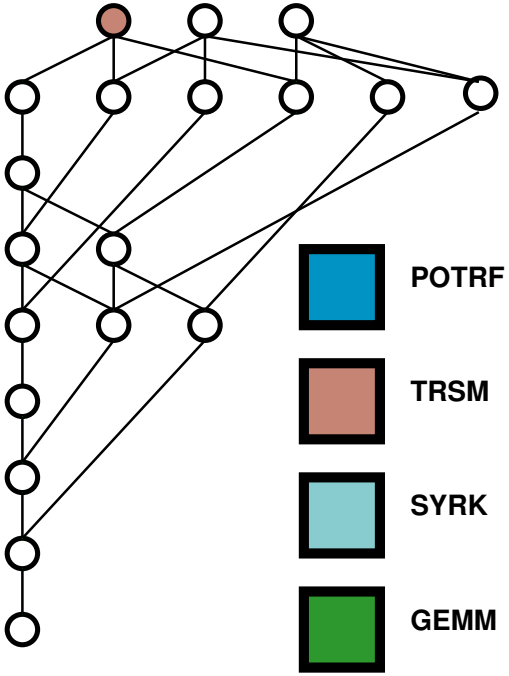




Runtime Parallel Execution on a Heterogeneous Node

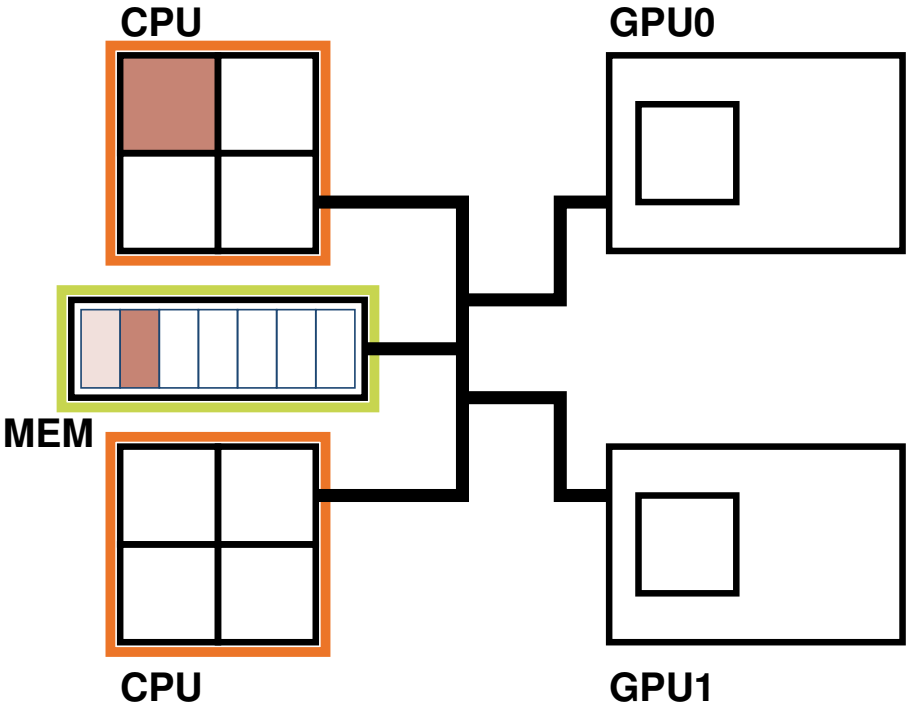


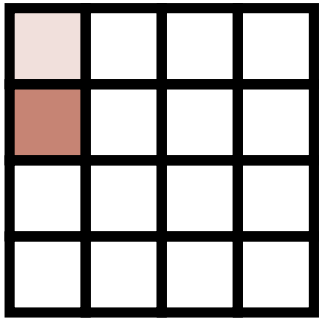


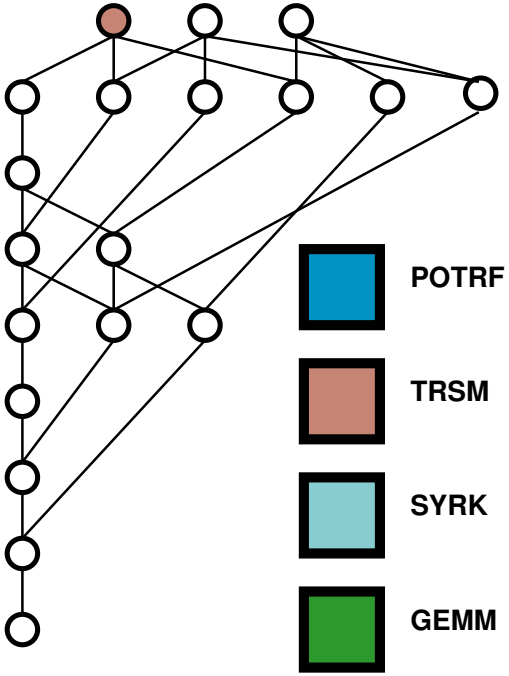


- Handles dependencies

Runtime Parallel Execution on a Heterogeneous Node

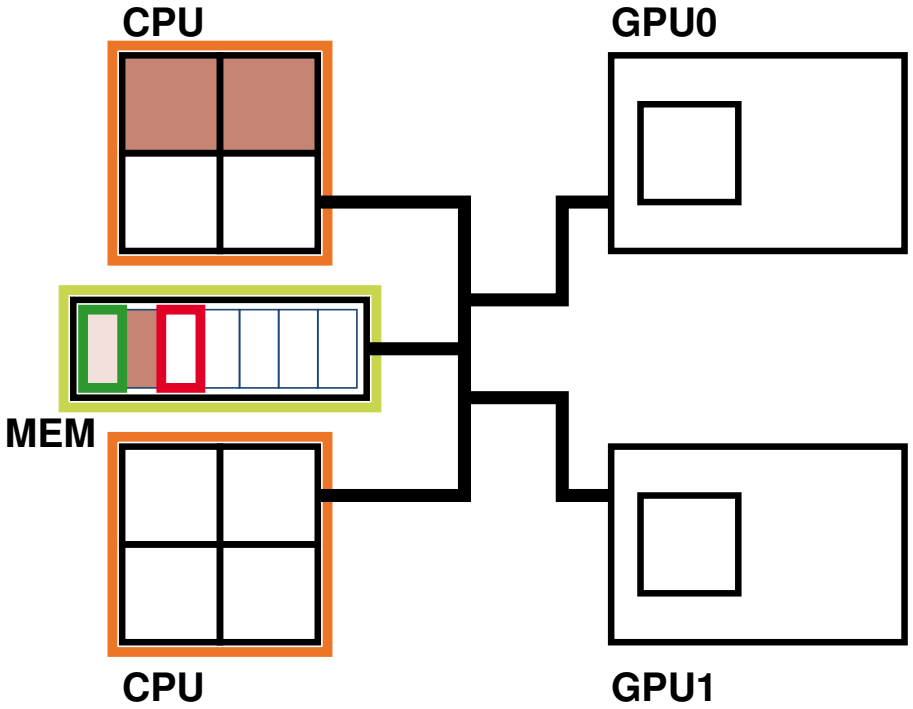


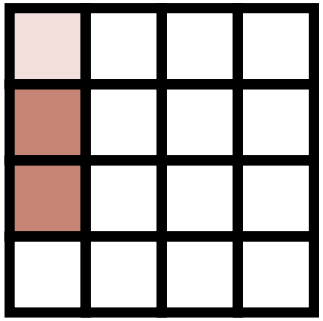


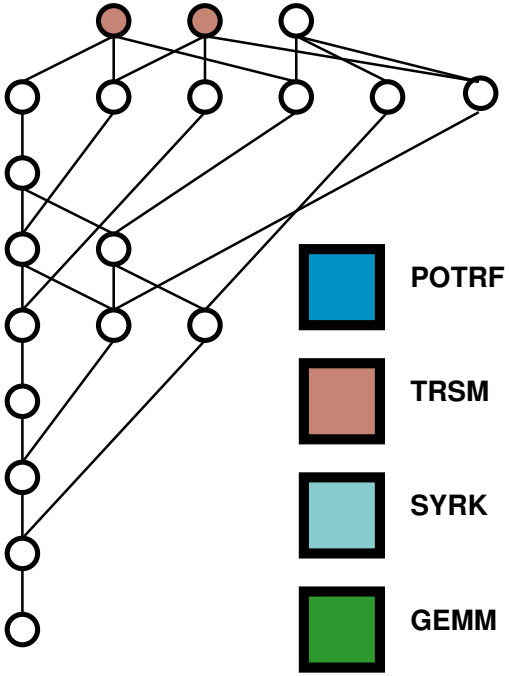


- Handles dependencies

Runtime Parallel Execution on a Heterogeneous Node

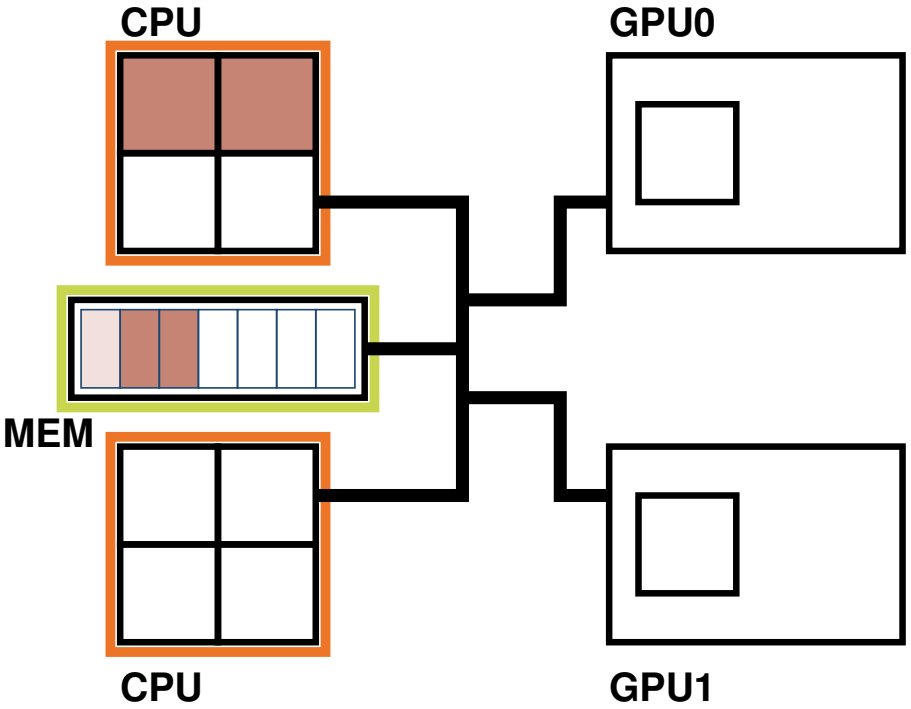


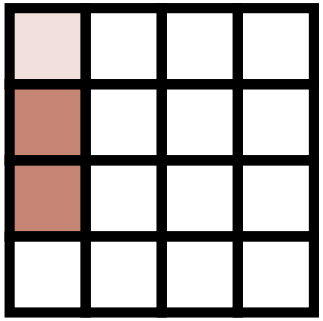


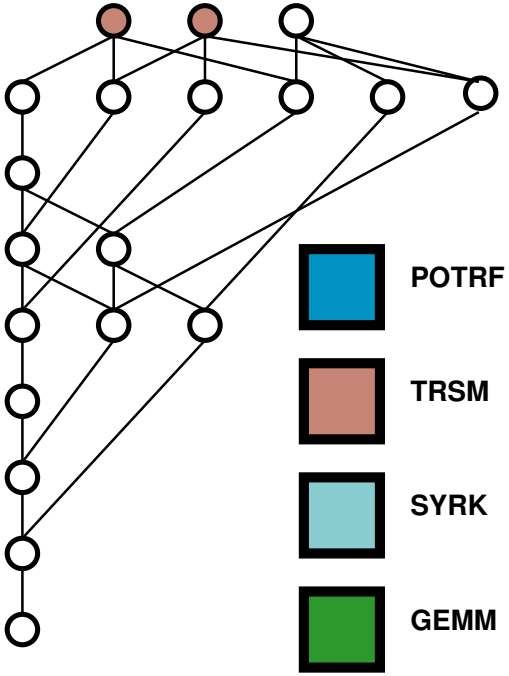


- Handles dependencies

Runtime Parallel Execution on a Heterogeneous Node

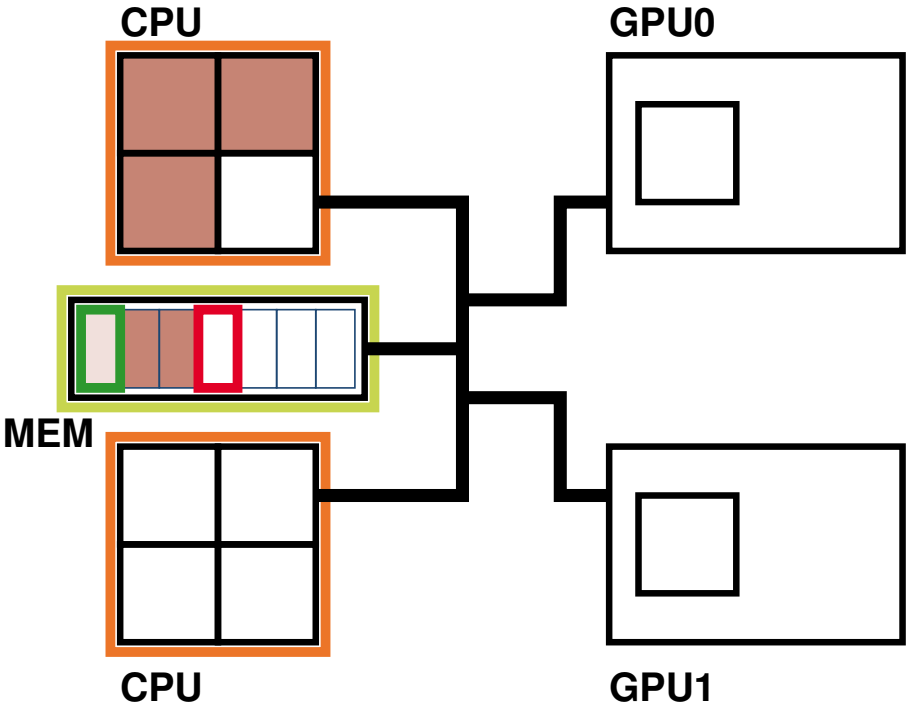


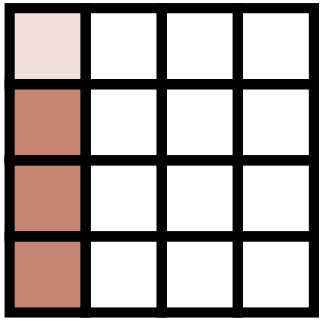


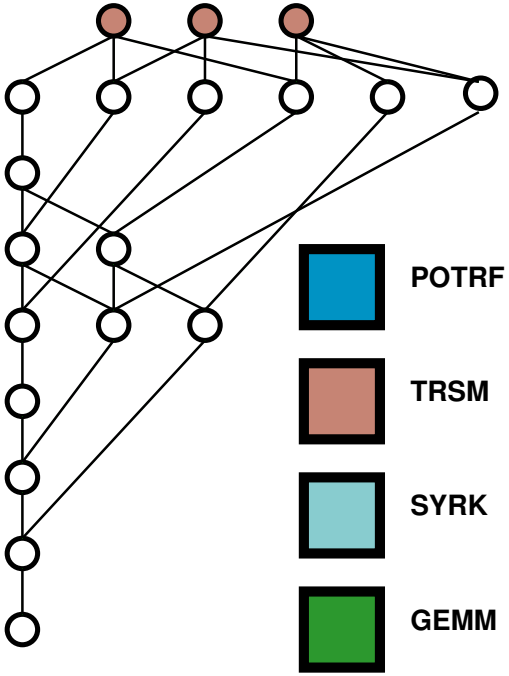


- Handles dependencies

Runtime Parallel Execution on a Heterogeneous Node

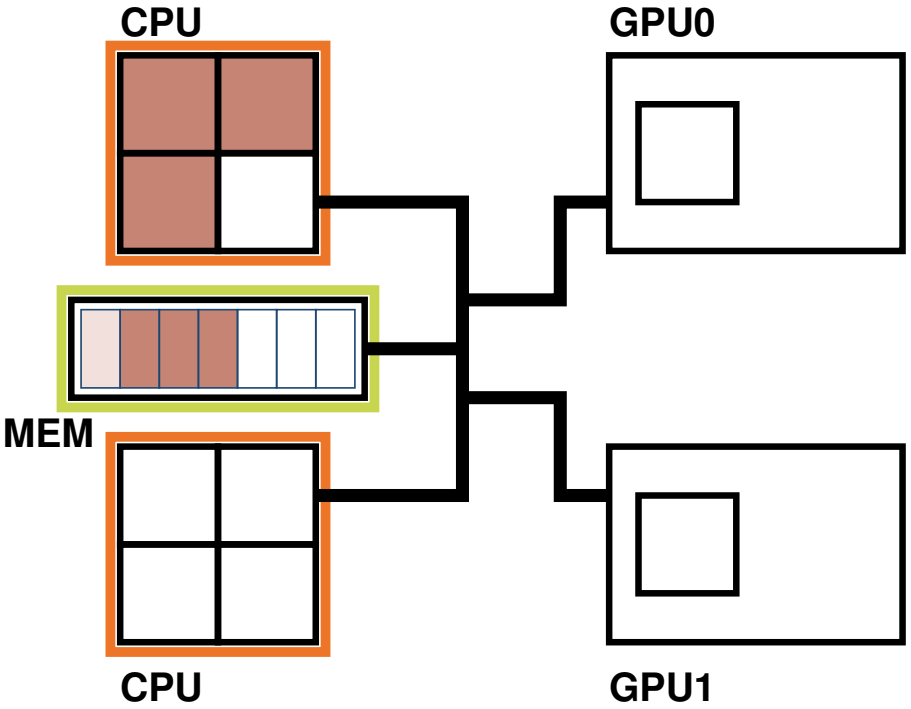


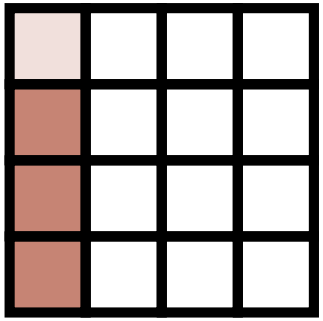


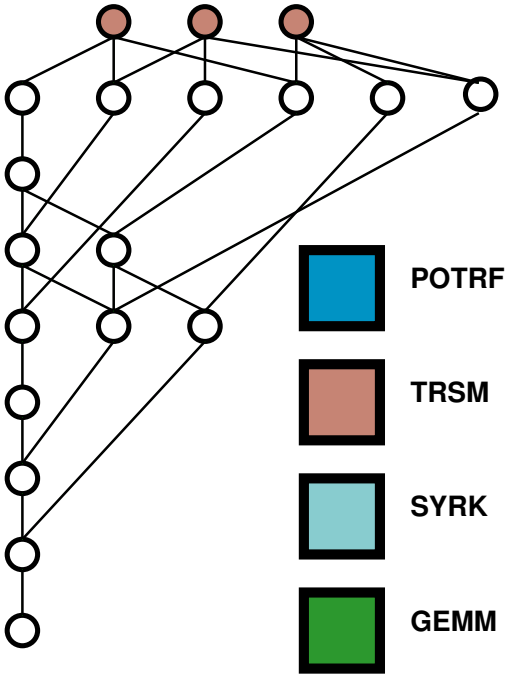


- Handles dependencies

Runtime Parallel Execution on a Heterogeneous Node

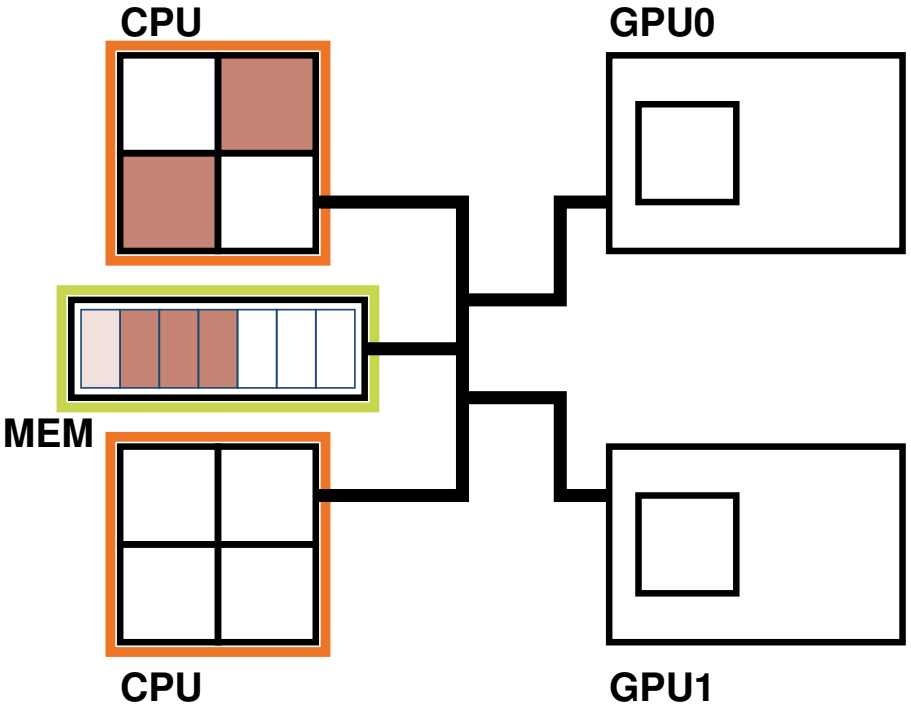


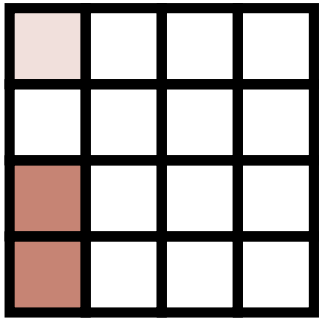


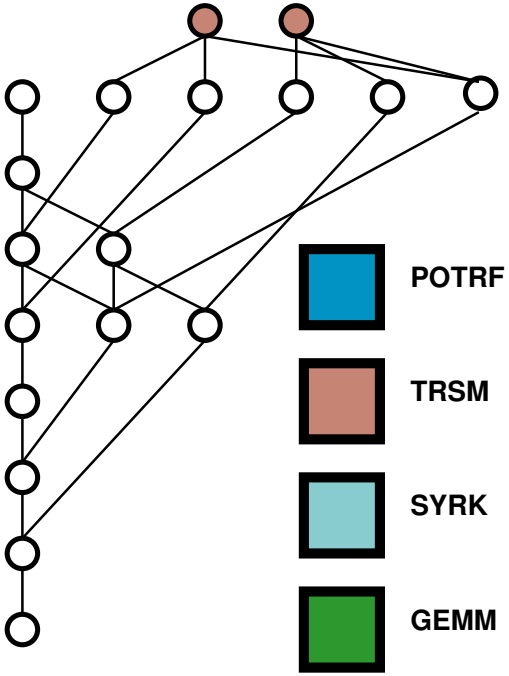


- Handles dependencies

Runtime Parallel Execution on a Heterogeneous Node

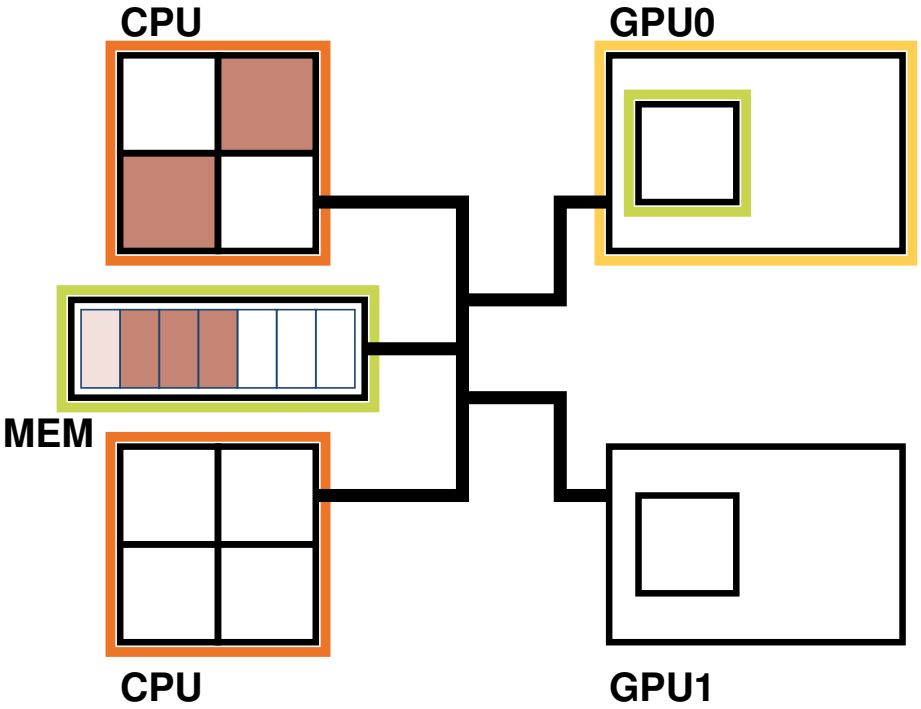


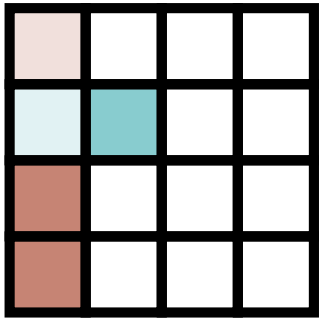


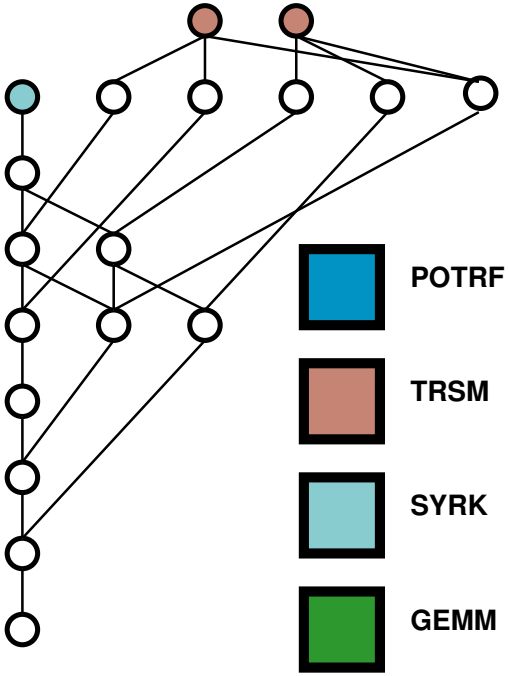


- Handles dependencies

Runtime Parallel Execution on a Heterogeneous Node

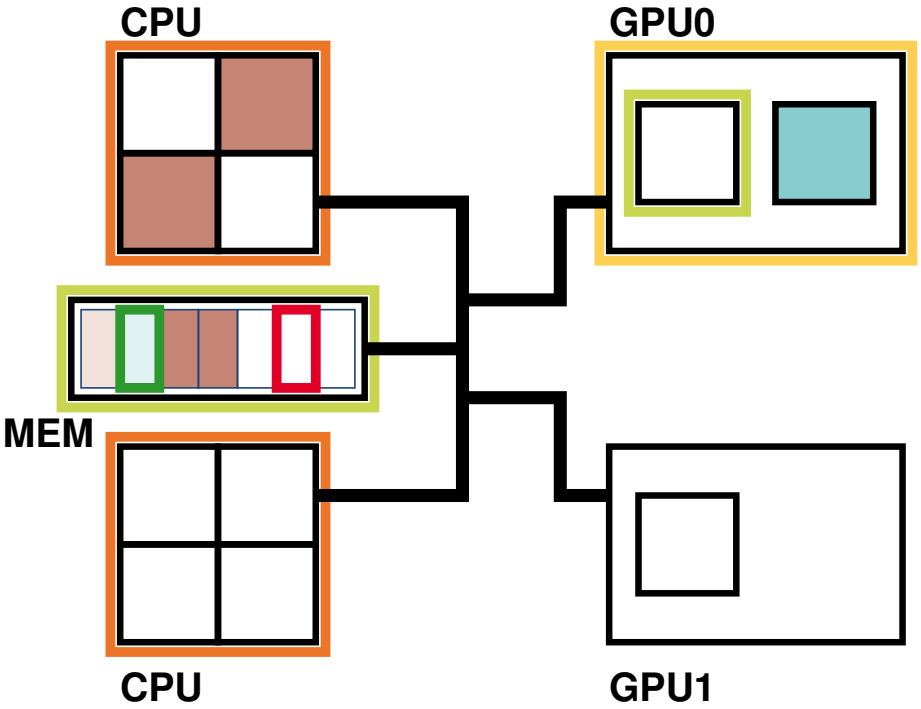


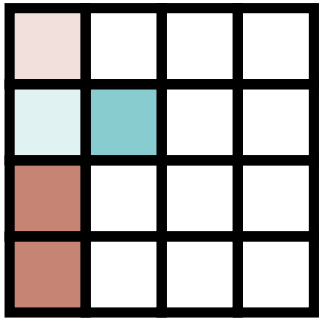


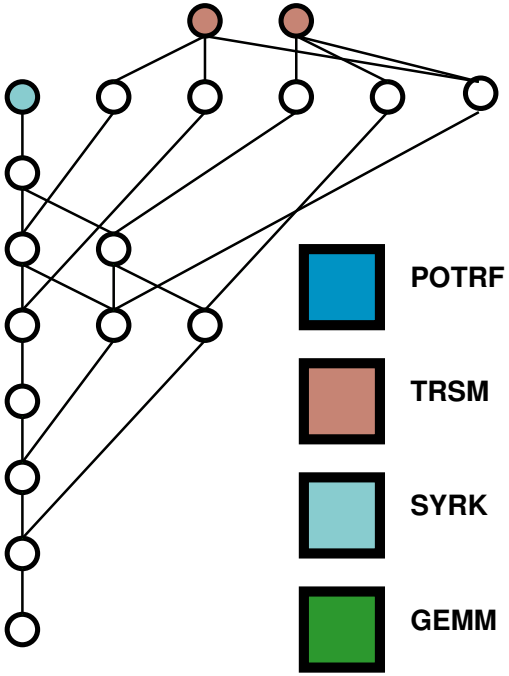


- Handles dependencies
- Handles scheduling (policy)

Runtime Parallel Execution on a Heterogeneous Node

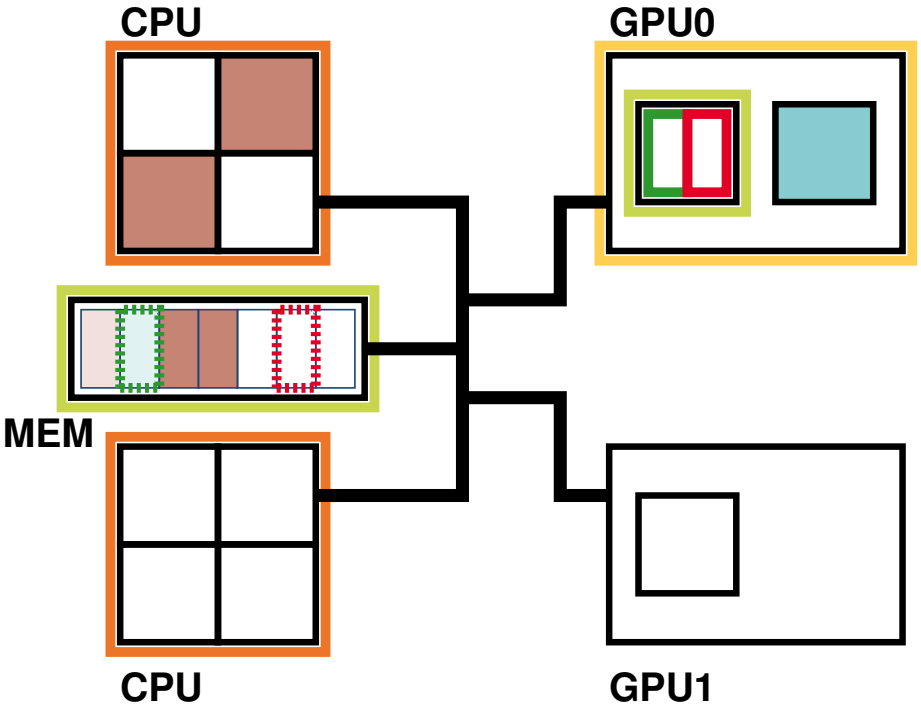


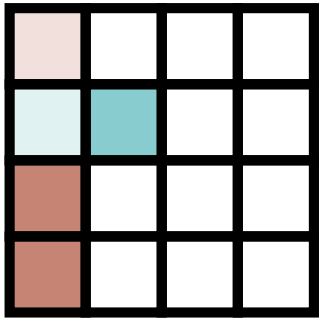


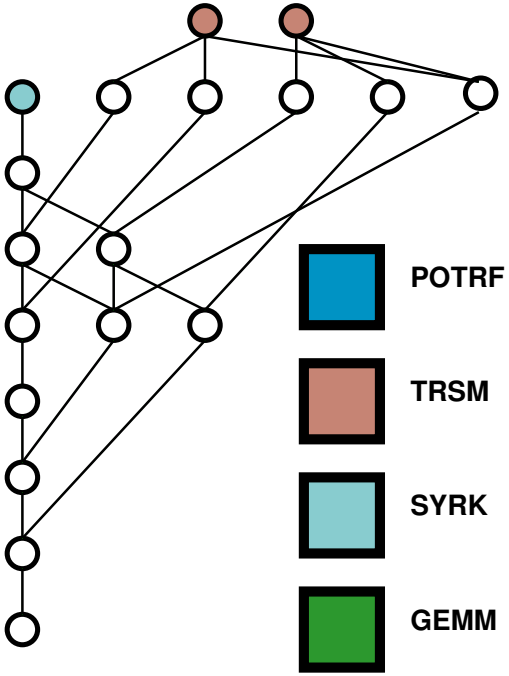


- Handles dependencies
- Handles scheduling (policy)

Runtime Parallel Execution on a Heterogeneous Node

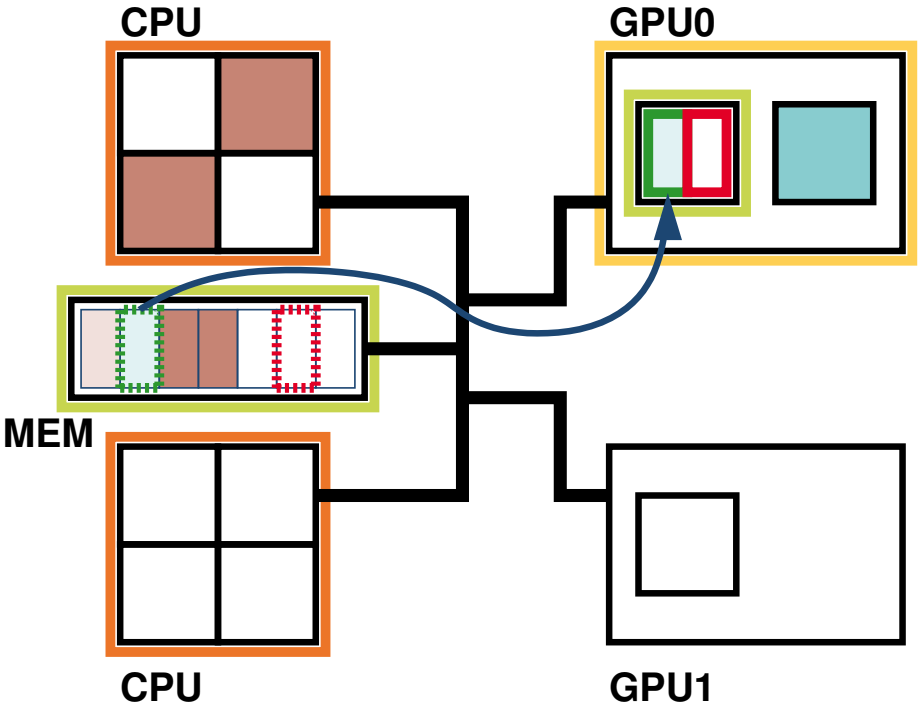


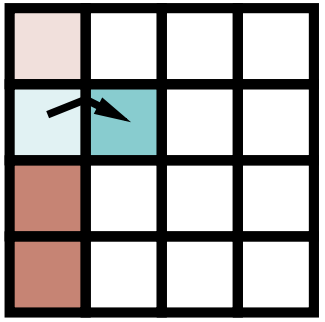


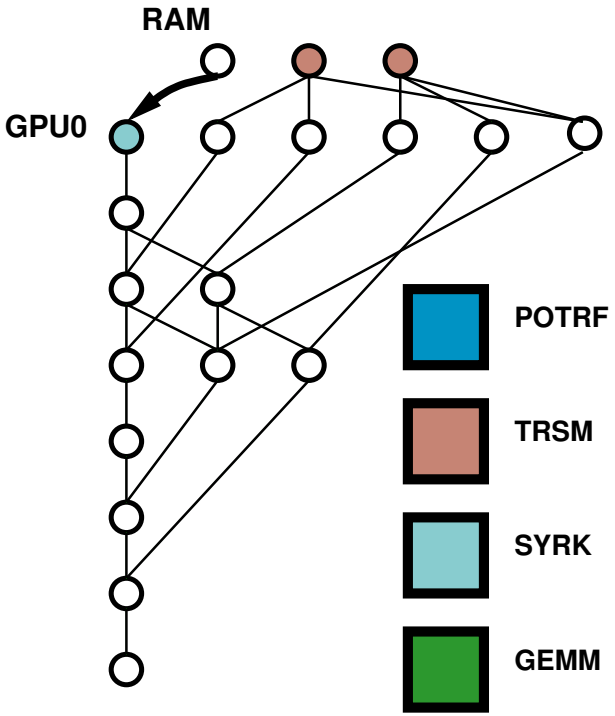


- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)

Runtime Parallel Execution on a Heterogeneous Node

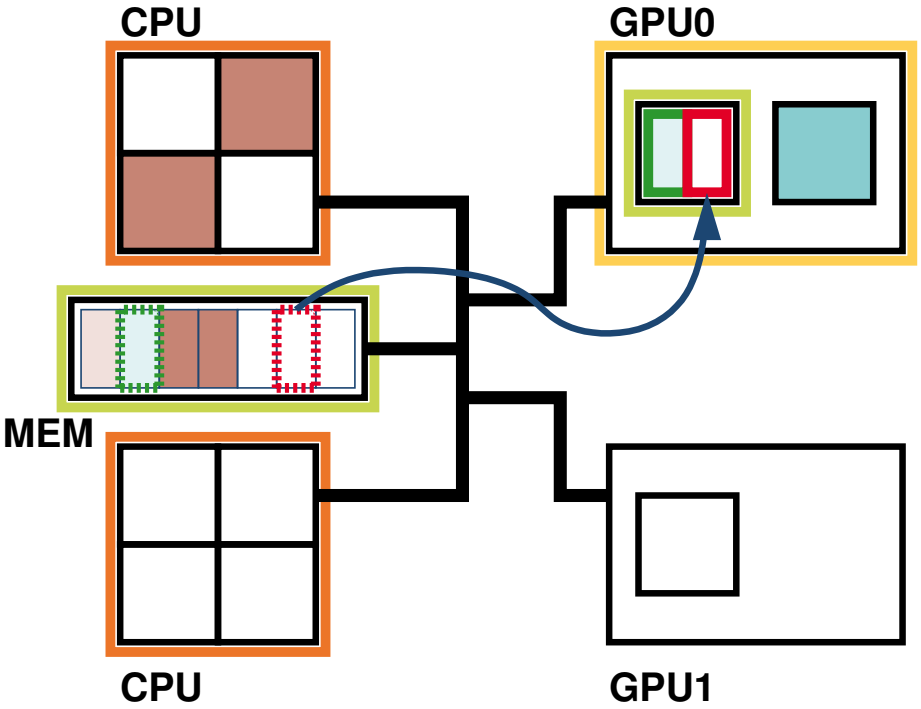


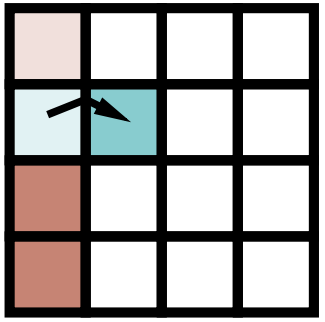


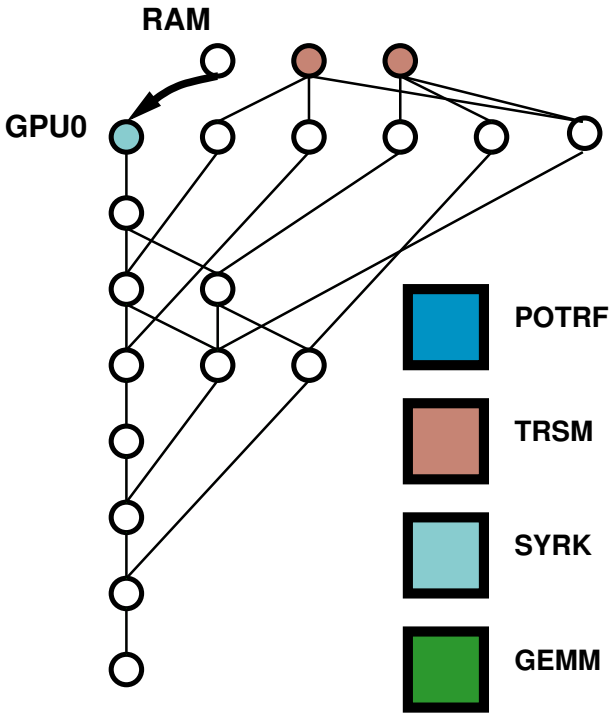


- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)

Runtime Parallel Execution on a Heterogeneous Node

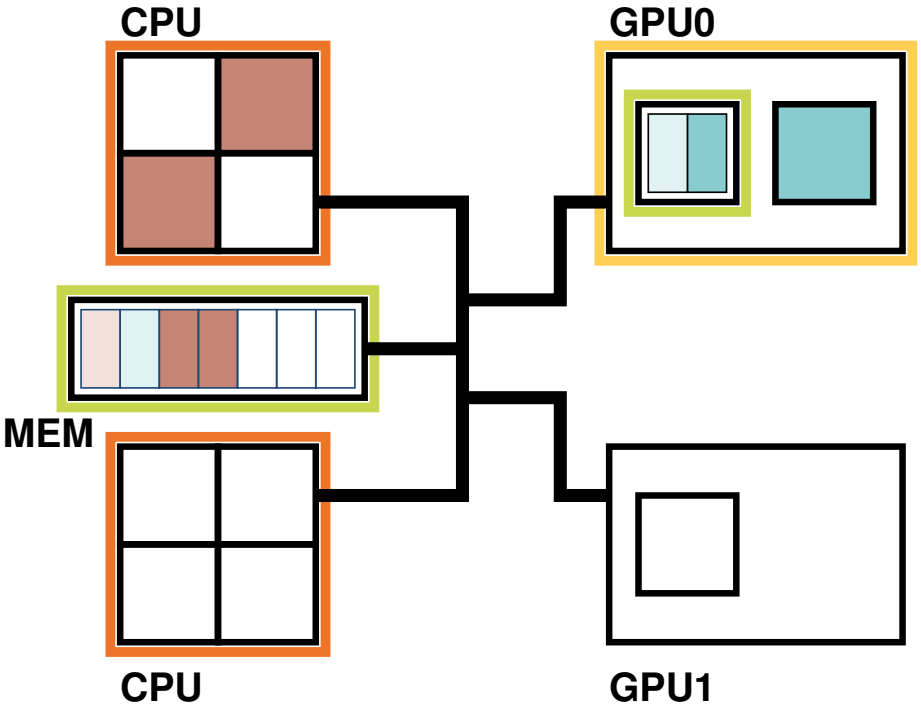


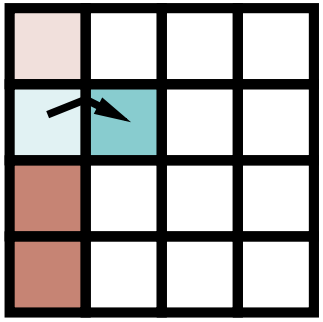


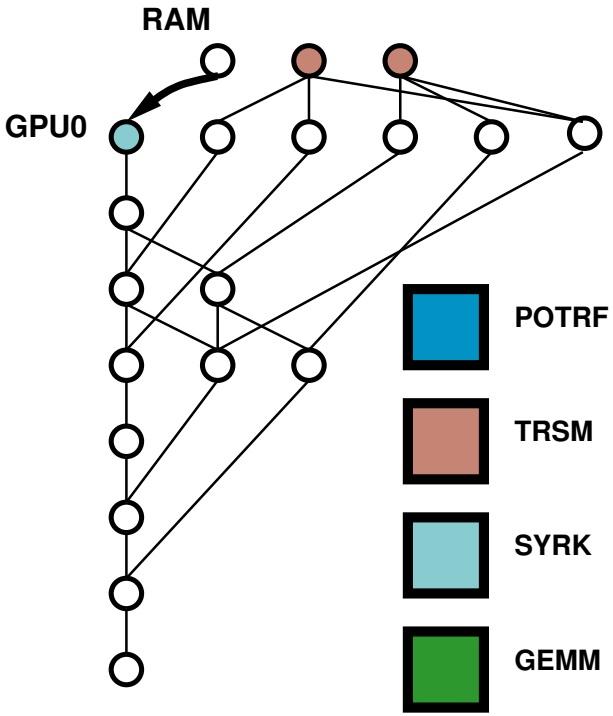


- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)

Runtime Parallel Execution on a Heterogeneous Node

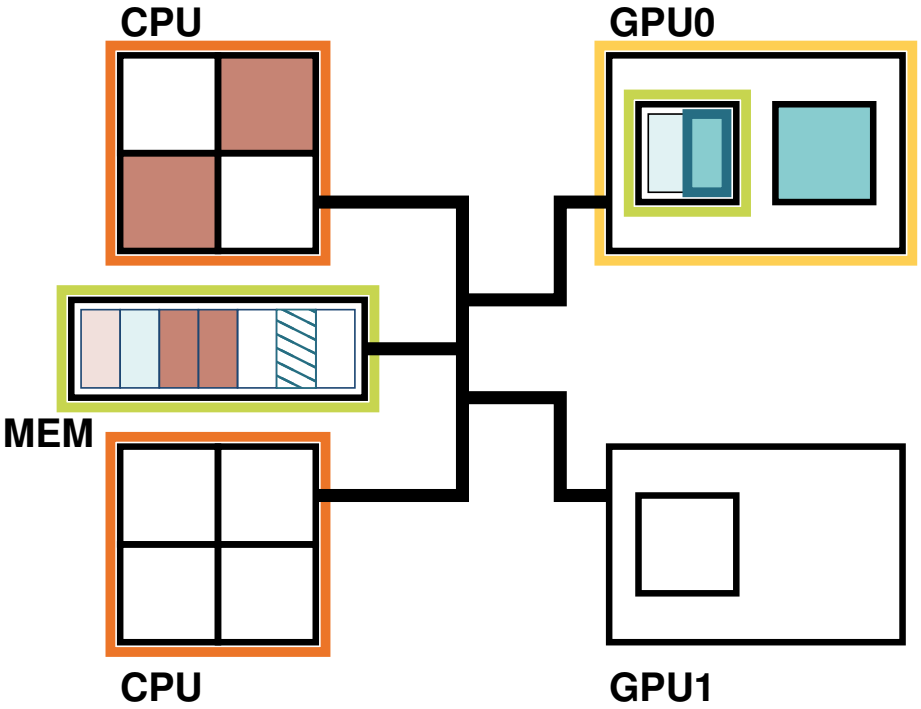


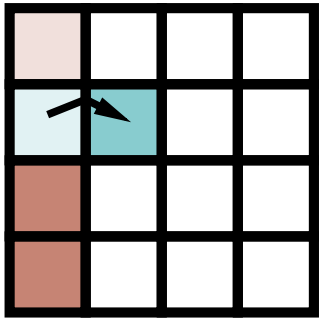


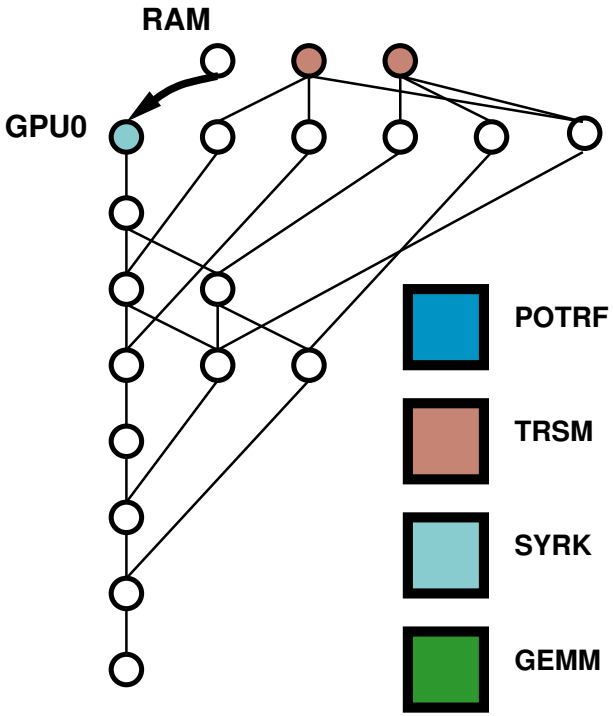


- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)

Runtime Parallel Execution on a Heterogeneous Node





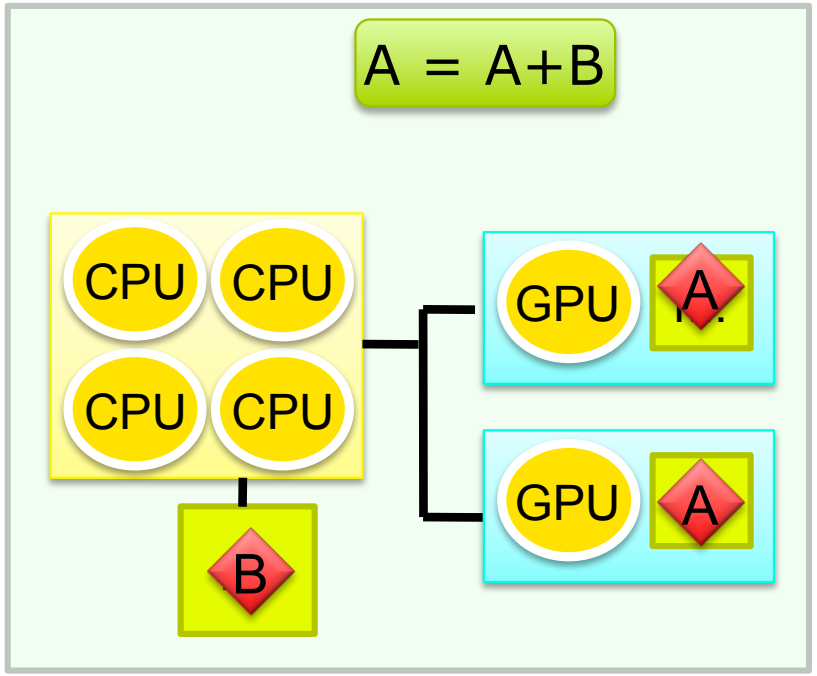


- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)

Distributed Shared Memory Consistency

MSI Protocol

- M: Modified
- S: Shared
- I: Invalid



Data A

I	S	S

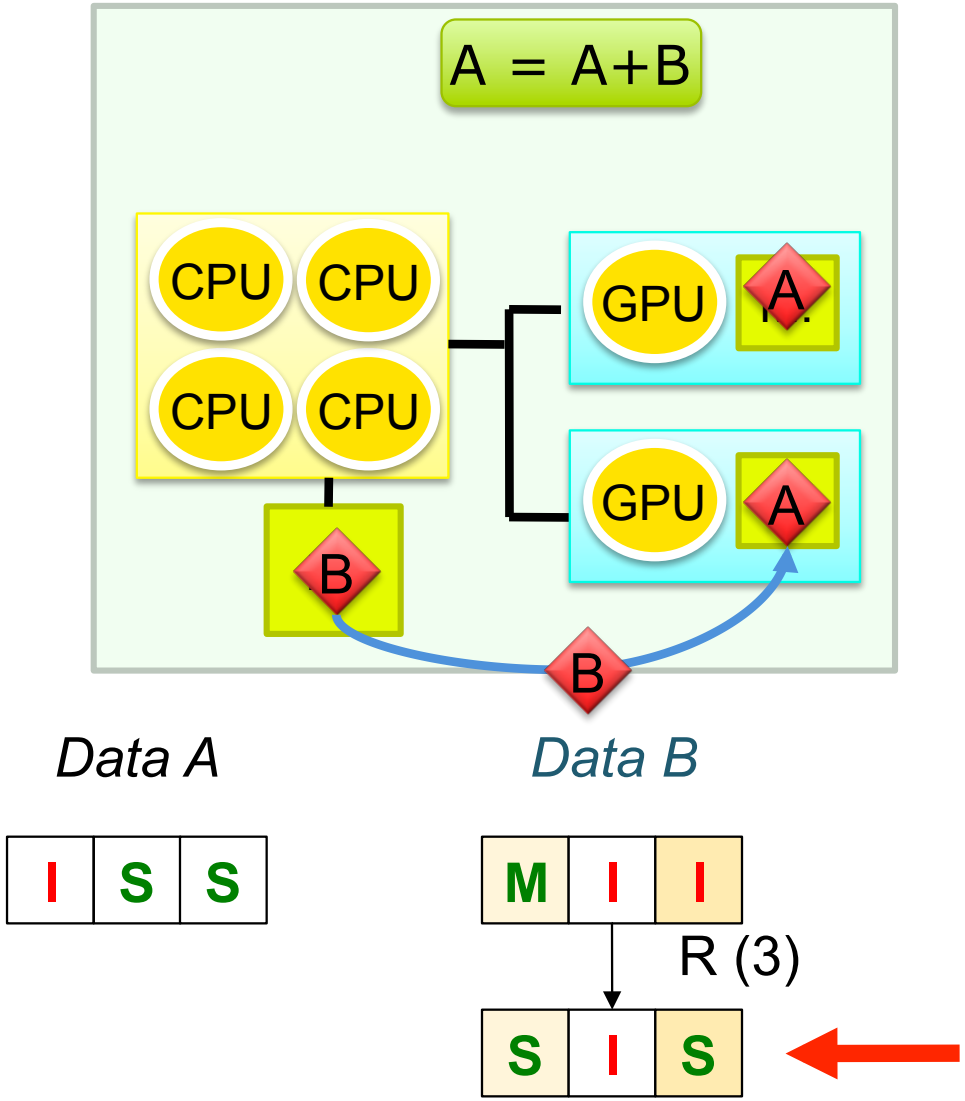
Data B

M	I	I

Distributed Shared Memory Consistency

MSI Protocol

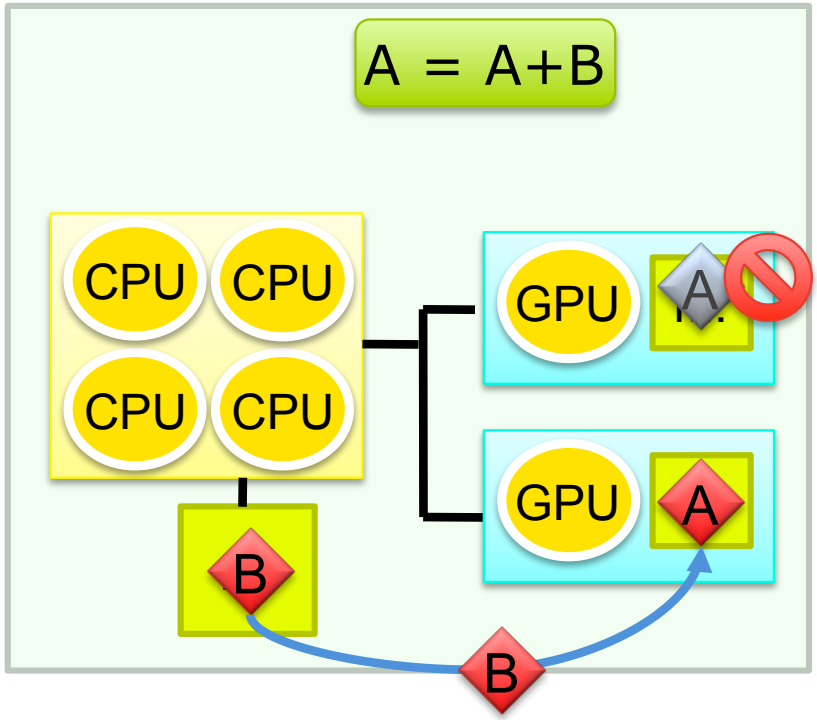
- M: Modified
- S: Shared
- I: Invalid

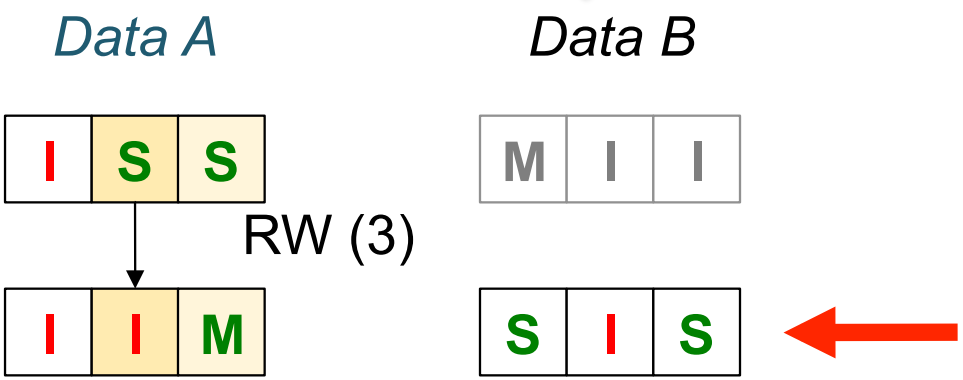


Distributed Shared Memory Consistency

MSI Protocol

- M: Modified
- S: Shared
- I: Invalid





Data Transfer Cost Modelling for Improved Scheduling

Discrete accelerators

- CPU $\leftrightarrow$ GPU transfers
- Data transfer cost vs kernel offload benefit

Data Transfer Cost Modelling for Improved Scheduling

Discrete accelerators

- CPU $\leftrightarrow$ GPU transfers
- Data transfer cost vs kernel offload benefit

Transfer cost modelling

- Bus calibration
  - Can differ even for identical devices
  - Platform's topology

Data Transfer Cost Modelling for Improved Scheduling

Discrete accelerators

- CPU $\leftrightarrow$ GPU transfers
- Data transfer cost vs kernel offload benefit

Transfer cost modelling

- Bus calibration
  - Can differ even for identical devices
  - Platform's topology

Data-transfer aware scheduling

- Deque Model Data Aware (dmda) scheduling policy variants
- Tunable data transfer cost bias
  - locality
  - vs load balancing

Data Prefetching

Task states

- Submitted
  - Task inserted by the application
- Ready
  - Task's dependencies resolved
- Scheduled
  - Task queued on a computing unit
- Executing
  - Task running on a computing unit

Data Prefetching

Task states

- Submitted
  - Task inserted by the application
- Ready
  - Task's dependencies resolved
- Scheduled
  - Task queued on a computing unit
- Executing
  - Task running on a computing unit

Anticipate on the **Scheduled** $\rightarrow$ **Executing** transition

- Prefetch triggered ASAP after **Scheduled** state

Data Prefetching

Task states

- Submitted
  - Task inserted by the application
- Ready
  - Task's dependencies resolved
- Scheduled
  - Task queued on a computing unit
- Executing
  - Task running on a computing unit

Anticipate on the **Scheduled** $\rightarrow$ **Executing** transition

- Prefetch triggered ASAP after **Scheduled** state
- Prefetch may also be triggered by the application

Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```
int vector[NX];
starpu_data_handle_t handle;

starpu_vector_data_register(&handle, 0, (uintptr_t)vector,
                            NX, sizeof(vector[0]));
```

Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```
1 float matrix[NX*NY];
2 starpu_data_handle_t handle;
3
4 starpu_matrix_data_register(&handle, 0, (uintptr_t)matrix,
5                             NX, NX, NY, sizeof(matrix[0]));
```

Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix

```
...
starpu_data_handle_t handle;

starpu_bcsr_data_register(&handle, 0, NNZ, NROW,
            (uintptr_t)bcsr_matrix_data,
            bcsr_matrix_indices, bcsr_matrix_rowptr, first_entry,
            BLOCK_NROW, BLOCK_NCOL, sizeof(double));
```

Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix
- Extensible data type set
  - You can write your own, specifically tailored data type

Data Interfaces

Multiple data types supported

- Vector
- Matrix
- BCSR sparse matrix
- Extensible data type set
  - You can write your own, specifically tailored data type
- Only the byte size and the shape of data matter, not the actual element type (integer, float, double precision float, . . .)

Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of **filter**

Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of **filter**

Partition

```
int vector[NX];
starpu_data_handle_t handle;
starpu_vector_data_register(&handle, 0, (uintptr_t)vector,
                            NX, sizeof(vector[0]));

/* Partition the vector in NB_PARTS sub-vectors */
struct starpu_data_filter filter = {
    .filter_func = starpu_vector_filter_block,
    .nchildren = NB_PARTS
};
starpu_data_partition(handle, &filter);

/* Data can only be accessed through sub-handles now */
```

Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of **filter**

Partition → Use

```
for (i=0; i<starpu_data_get_nb_children(handle); i++) {
    /* Get subdata number i */
    starpu_data_handle_t sub_handle =
        starpu_data_get_sub_data(handle, 1, i);

    starpu_task_insert(
        &scal_cl,
        STARPU_RW, sub_handle,
        STARPU_VALUE, &factor, sizeof(factor),
        0);
}
```

Partitioning

Splitting a piece of managed data into several handles

- Granularity adjustment
- Notion of **filter**

Partition → Use → Unpartition

```
/* Wait for submitted tasks to complete */
starpu_task_wait_for_all();

/* Unpartition data */
starpu_data_unpartition(handle, &filter);

/* Data can now be accessed through 'handle' only */
```

Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps

Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps

- Partition planning

```
int vector[NX];
starpu_data_handle_t handle;
starpu_vector_data_register(&handle, 0, (uintptr_t)vector,
                            NX, sizeof(vector[0]));

/* Partition the vector in NB_PARTS sub-vectors */
struct starpu_data_filter filter = {
    .filter_func = starpu_vector_filter_block,
    .nchildren = NB_PARTS
};
starpu_data_handle_t children[NB_PARTS];
starpu_data_partition_plan(handle, &filter, children);

/* Data can only be accessed through sub-handles now */
```

Asynchronous Partitioning

Inserting a partitioning request in the submission flow

Two steps

- Partition planning
- Asynchronous partition inforcement

```
starpu_task_insert(&scal_cl,
      STARPU_RW, handle,
      STARPU_VALUE, &factor1, sizeof(factor1), 0);
starpu_data_partition_submit(handle, NB_PARTS, children);
for (i=0; i<NB_PARTS; i++) {
    starpu_task_insert(&scal_cl,
          STARPU_RW, children[i],
          STARPU_VALUE, &factor2, sizeof(factor2),
          0);
}
starpu_data_unpartition_submit(handle, NB_PARTS, children, node);
starpu_task_insert(&scal_cl,
      STARPU_RW, handle,
      STARPU_VALUE, &factor3, sizeof(factor3), 0);
```

Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Define zero

```
1 void bzero_cpu(void *descr[], void *cl_arg) {
2     double *v_zero = (double *)STARPU_VARIABLE_GET_PTR(descr[0]);
3     *v_zero = 0.0;
4 }
5
6 struct starpu_codelet bzero_cl = {
7     .cpu_funcs = { bzero_cpu, NULL },
8     .nbuffers = 1
9 };
```

Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Define zero → Define op

```
1  void accumulate_cpu(void *descr[], void *cl_arg) {
2      double *v_dst = (double *)STARPU_VARIABLE_GET_PTR(descr[0]);
3      double *v_src = (double *)STARPU_VARIABLE_GET_PTR(descr[1]);
4      *v_dst = *v_dst + *v_src;
5  }
6
7  struct starpu_codelet accumulate_cl = {
8      .cpu_funcs = { accumulate_cpu, NULL },
9      .nbuffers = 1
10 };
```

Reduction

Merge contributions from a set of tasks into a single buffer

- Define neutral element initializer
- Define reduction operator

Define zero → Define op → Reduce task contributions

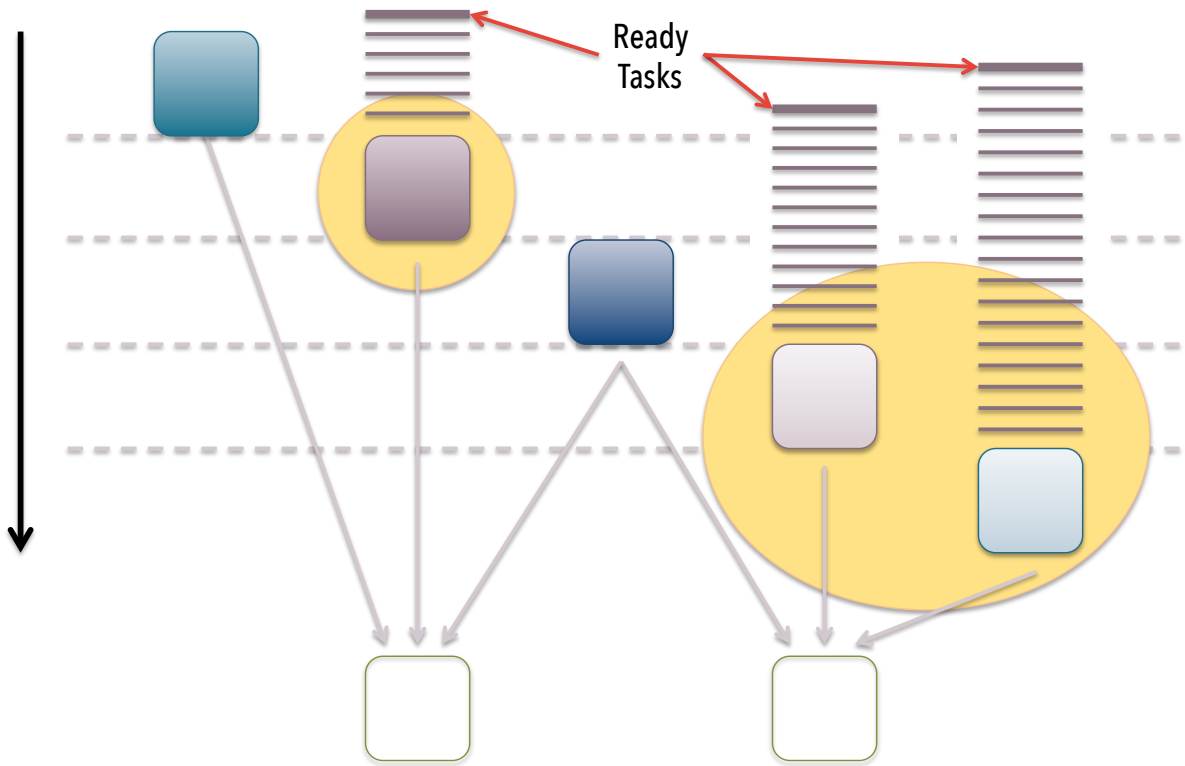
```
starpu_variable_data_register(&accum_handle, -1,
                              NULL, sizeof(type));
starpu_data_set_reduction_methods(accum_handle,
                                  &accumulate_cl, &bzero_cl);

for (b = 0; b < nblocks; b++)
    starpu_task_insert(&dot_kernel_cl,
        STARPU_REDUX, accum_handle,
        STARPU_R, starpu_data_get_sub_data(v1, 1, b),
        STARPU_R, starpu_data_get_sub_data(v2, 1, b),
        0);
```

Commutative Write Accesses

- Write accesses enforce sequential consistency by default
  - To strong for some kind of workloads
  - N-body, unstructured meshes

Commutative Write Accesses

- Write accesses enforce sequential consistency by default
  - To strong for some kind of workloads
  - N-body, unstructured meshes



Commutative Write Accesses

- Write accesses enforce sequential consistency by default
  - To strong for some kind of workloads
  - N-body, unstructured meshes
- **Commute**: allows a set of tasks to modify a buffer in any order

```
starpu_task_insert(&cl1,
    STARPU_R, handle0,
    STARPU_RW, handle,
    0);
starpu_task_insert(&cl2,
    STARPU_R, handle1,
    STARPU_RW|STARPU_COMMUTE, handle,
    0);
starpu_task_insert(&cl2,
    STARPU_R, handle2,
    STARPU_RW|STARPU_COMMUTE, handle,
    0);
starpu_task_insert(&cl3,
    STARPU_R, handle3,
    STARPU_RW, handle,
    0);
```

Hands-on Session

Session Part 2: Optimizations

- Training Platform 'poincare'
- Registered attendee should have received login/passwd credentials and connection info
- Reserved computing nodes:

```
- gpu nodes:     LoadLeveler class 'clgpu'
- batch script submission:     llsubmit <script.sh>
```

- Web site:

  http://starpu.gforge.inria.fr/tutorials/2016-06-PATC/

4

Debugging / Monitoring

Online vs offline

Online Tools

- Statistics
- Visual debugging

Offline Tools

- Trace-based analysis

Online Profiling Feeback

- Task/worker mapping stats
- Bus stats
- Additional stats (memory, MSI cache)
  - Configure-time options

```
$ export STARPU_PROFILE=1 STARPU_WORKER_STATS=1
$ my_program
...
```

Online Profiling Feeback

- Task/worker mapping stats
- Bus stats
- Additional stats (memory, MSI cache)
  - Configure-time options

```
$ export STARPU_PROFILE=1 STARPU_WORKER_STATS=1
$ my_program
...
```

```
$ export STARPU_PROFILE=1 STARPU_BUS_STATS=1
$ my_program
...
```

Online Profiling Feeback

- Task/worker mapping stats
- Bus stats
- Additional stats (memory, MSI cache)
  - Configure-time options

```
$ export STARPU_PROFILE=1 STARPU_WORKER_STATS=1
$ my_program
...
```

```
$ export STARPU_PROFILE=1 STARPU_BUS_STATS=1
$ my_program
...
```

```
$ $STARPU_DIR/configure --enable-stats --enable-memory-stats \
                        [... other opts ...]
...
```

Offline Trace-Based Feedback

- FxT trace collection
- Trace analysis and display
  - ViTE Gantt
  - Graphviz DAG
  - R plots

Offline Feedback – Trace Collection

- Requires FxT trace toolkit
- Compile-time option to enable trace collection
- Environment variable to enable trace post-processing

```
$ $STARPU_DIR/configure --with-fxt [... other opts ...]
...
```

Offline Feedback – Trace Collection

- Requires FxT trace toolkit
- Compile-time option to enable trace collection
- Environment variable to enable trace post-processing

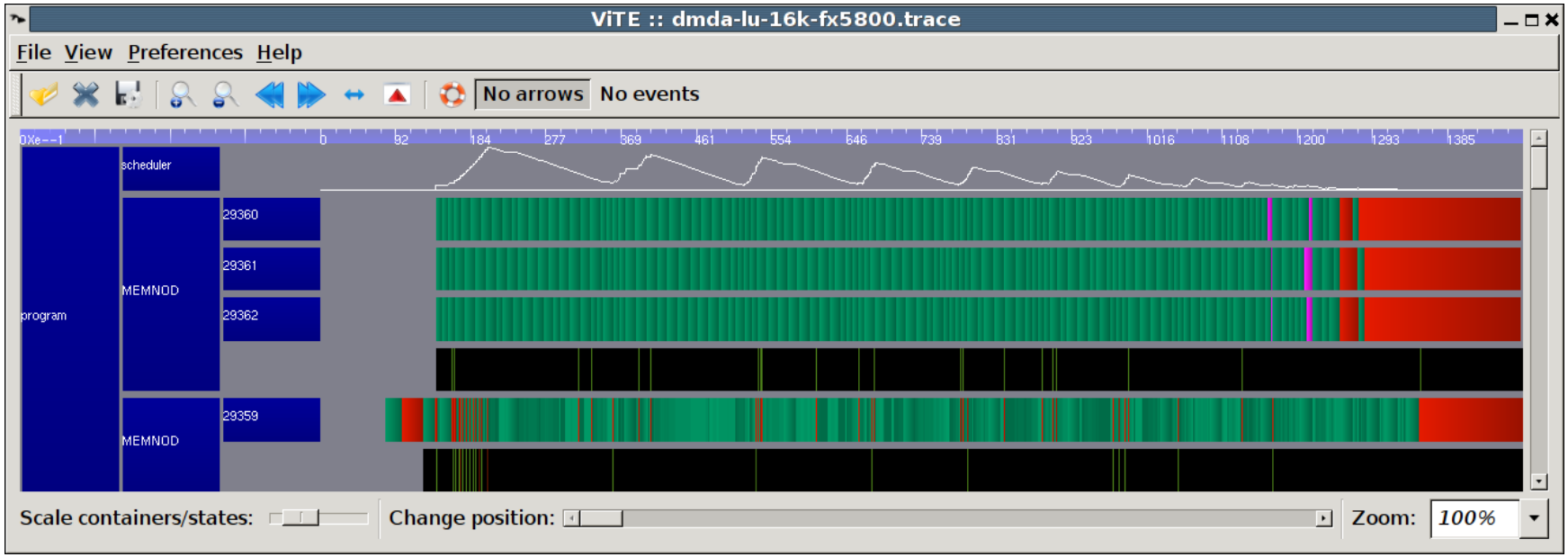
```
$ $STARPU_DIR/configure --with-fxt [... other opts ...]
...
```

```
$ export STARPU_GENERATE_TRACE=1
$ my_program
...
```

Offline Feedback – Trace Analysis

Automatically generated

- Dependency graph (DAG)
- Activity diagramm (GANTT)
  - Visualize with ViTE



Offline Feedback – Kernel Model

Display the codelet performance models recorded by StarPU

- Command-line tool starpu_perfmodel_display
- History-based models
- Regression-based models

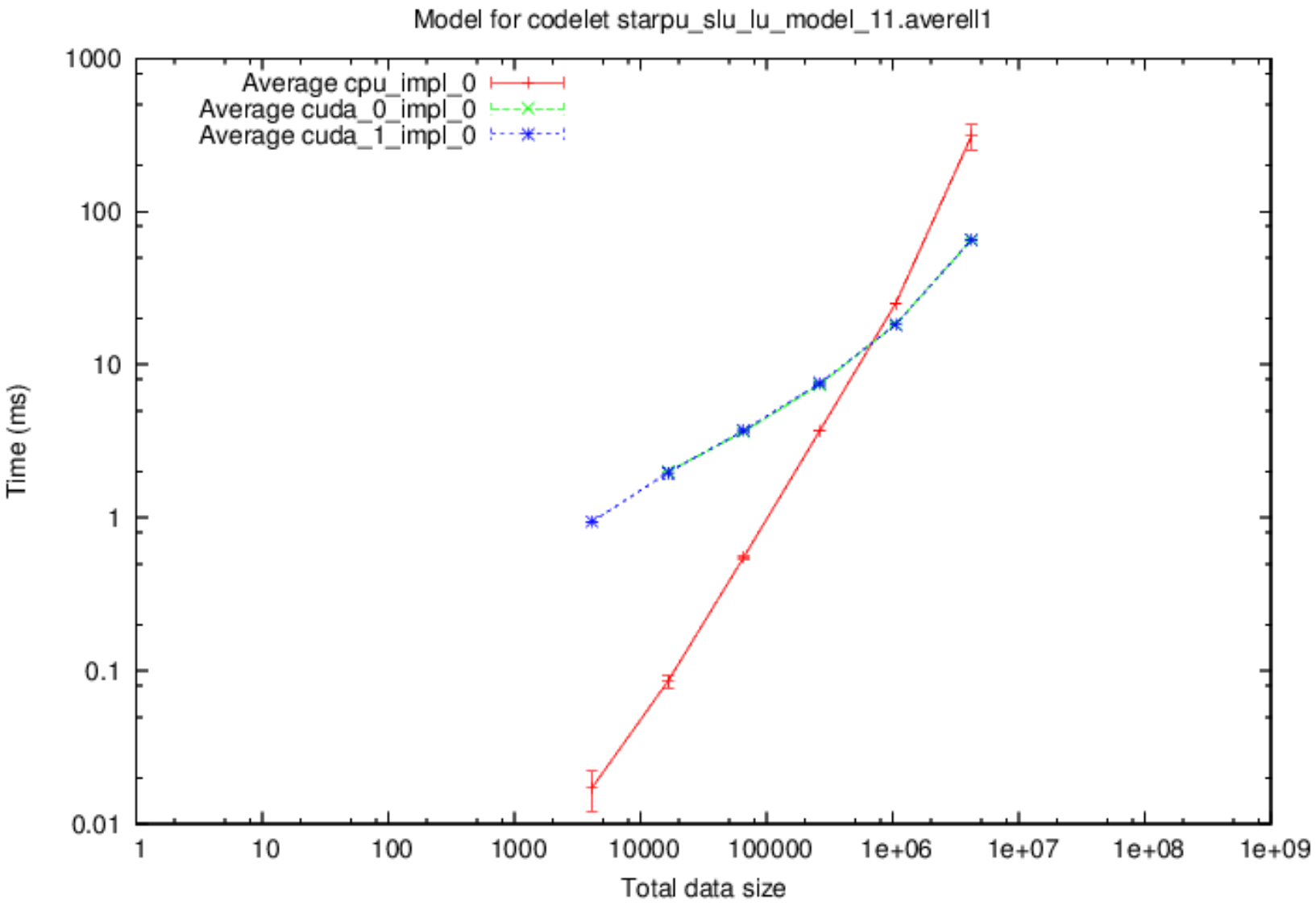
Offline Feedback – Kernel Model

Display the codelet performance models recorded by StarPU

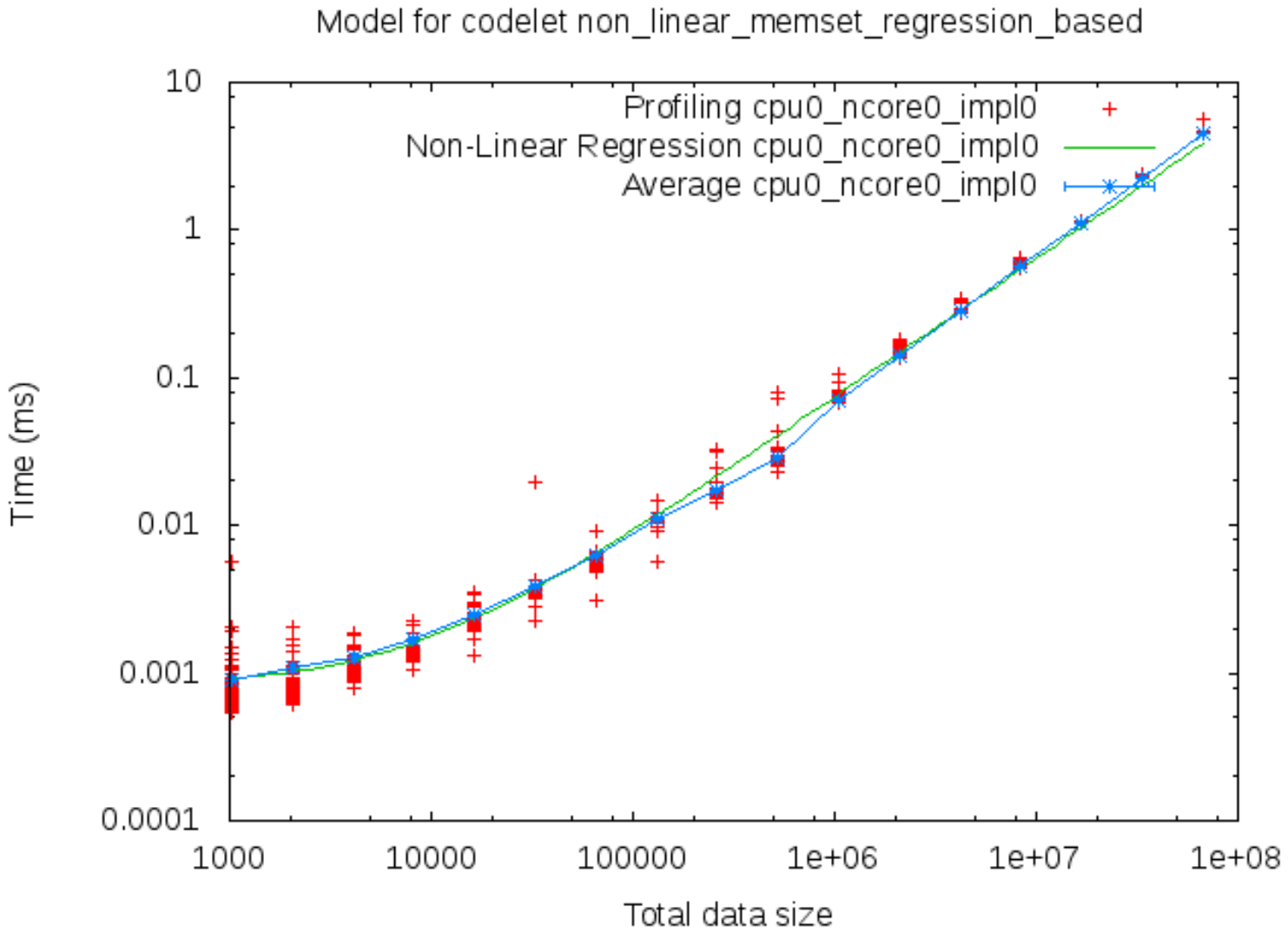
- Command-line tool starpu_perfmodel_display
- History-based models
- Regression-based models

```
$ starpu_perfmodel_display -s starpu_slu_lu_model_11

performance model for cpu0_parallel1_impl0
# hash        size        mean (us)        stddev (us)      n
aa6d4ef7      4194304     3.055501e+05     5.804822e+04     48
```

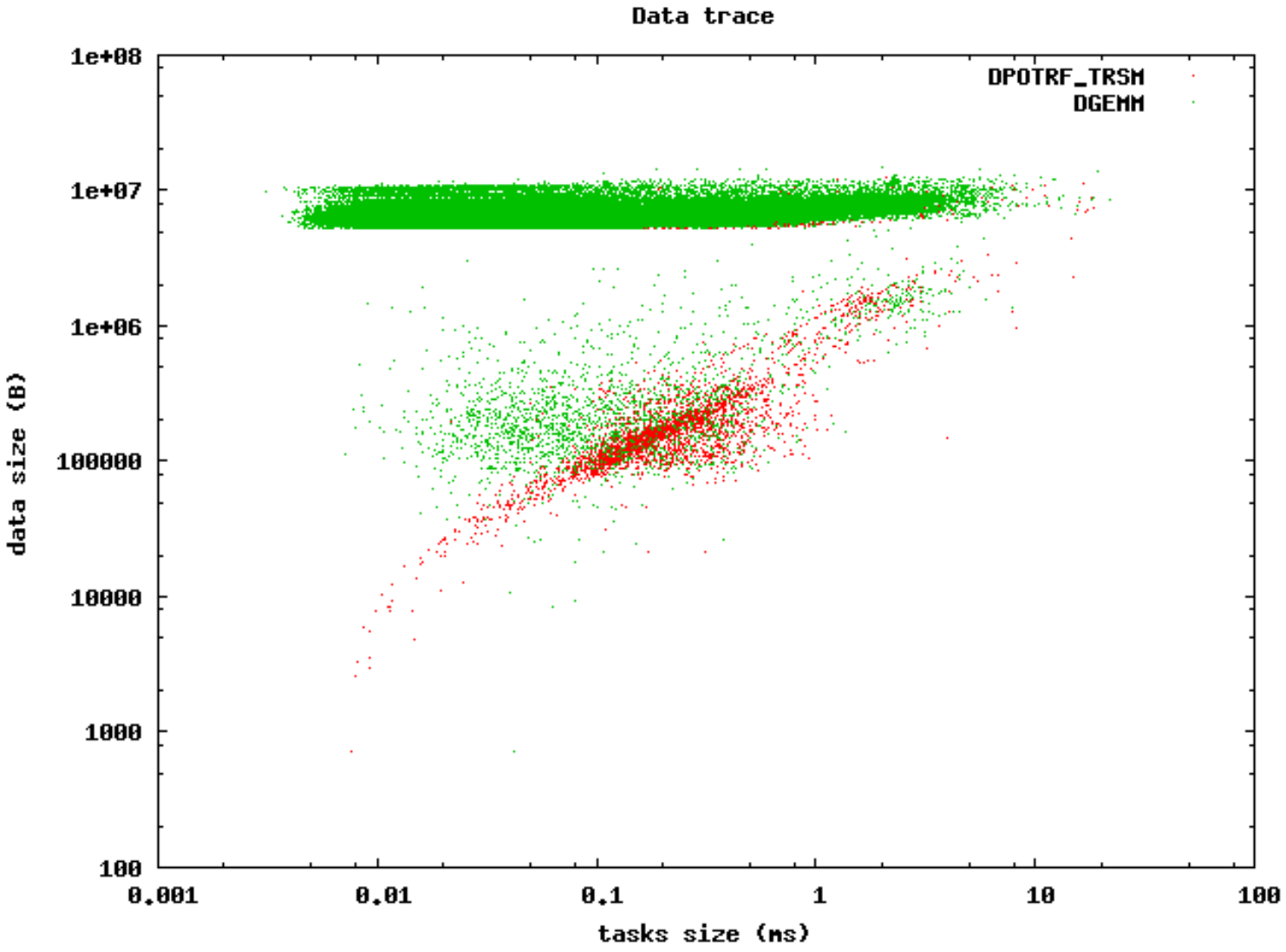
Offline Feedback – Kernel Model Characteristics



Offline Feedback – Kernel Model Regression Fitness



Offline Feedback – Synthetic Kernels' Behaviour



5

Distributed computing

Basic StarPU/MPI Example

Straightforward port of existing MPI code

```
for (loop = 0; loop < NLOOPS; loop++) {
    if (!(loop == 0 && rank == 0))
        MPI_Recv(&data, prev_rank, ...);

    inc(&data);

    if (!(loop == NLOOPS-1 && rank == size-1))
        MPI_Send(&data, next_rank, ...);
}
```

Basic StarPU/MPI Example

Straightforward port of existing MPI code

- Asynchronous requests

```
for (loop = 0; loop < NLOOPS; loop++) {
    if (!(loop == 0 && rank == 0))
        starpu_mpi_irecv_submit(data_handle, prev_rank, ...);

    starpu_mpi_task_insert(&inc_cl, STARPU_RW, data_handle, 0);

    if (!(loop == NLOOPS-1 && rank == size-1))
        starpu_mpi_isend_submit(data_handle, next_rank, ...);
}
starpu_task_wait_for_all();
```

Basic StarPU/MPI Example

Straightforward port of existing MPI code

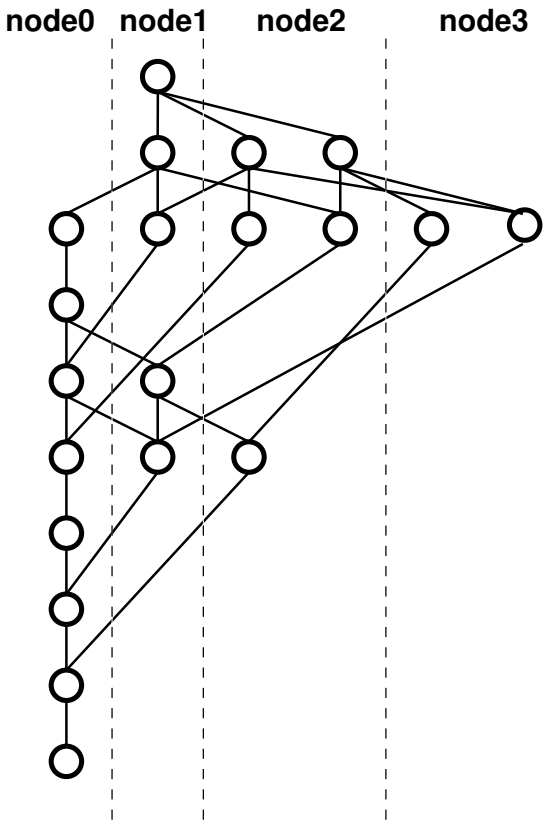
- Asynchronous requests
- Requests are DAG elements
  - Task may depend on **Irecv** completion
  - **Isend** may depend on task completion

```
for (loop = 0; loop < NLOOPS; loop++) {
    if (!(loop == 0 && rank == 0))
        starpu_mpi_irecv_submit(data_handle, prev_rank, ...);

    starpu_mpi_task_insert(&inc_cl, STARPU_RW, data_handle, 0);

    if (!(loop == NLOOPS-1 && rank == size-1))
        starpu_mpi_isend_submit(data_handle, next_rank, ...);
}
starpu_task_wait_for_all();
```

Distributed Support

Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow



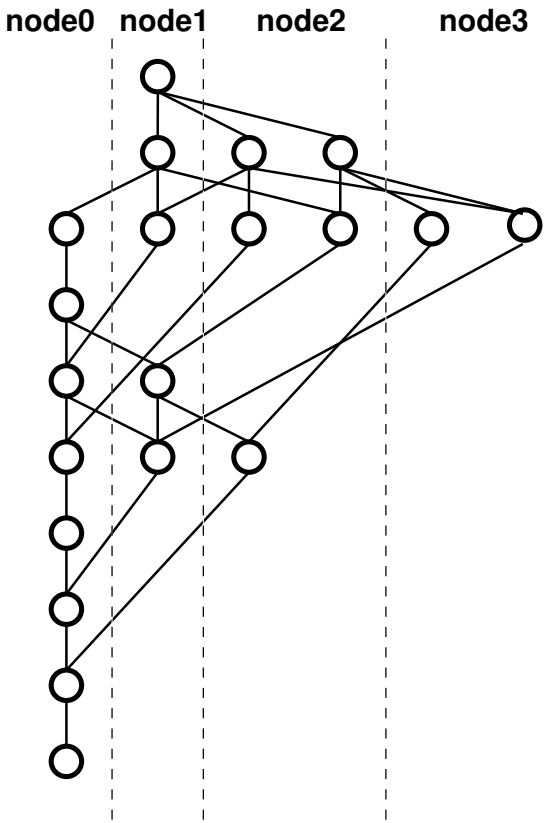
Distributed Support

Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow

Data↔Node Mapping

- Provided by the application
- Can be altered dynamically



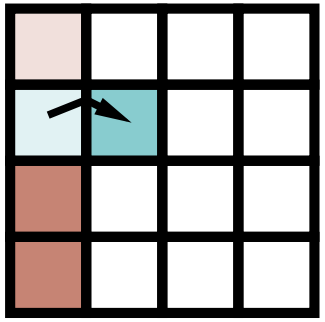
Distributed Support

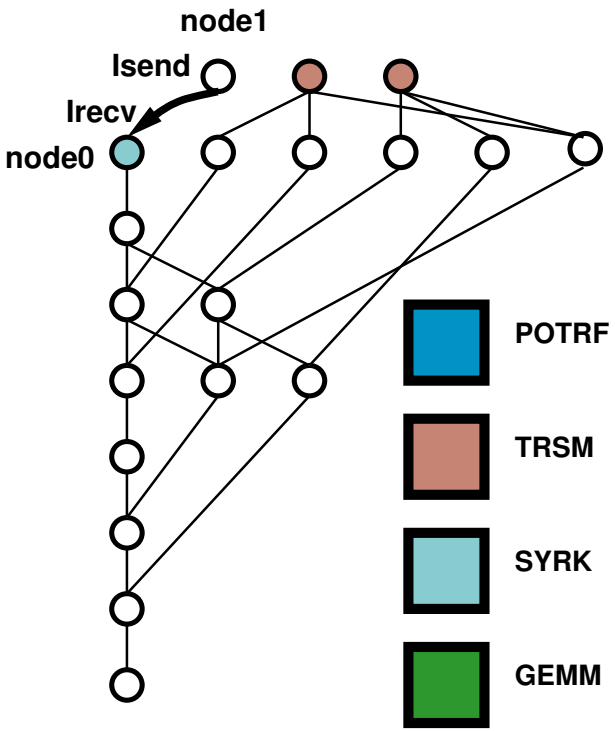
Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow

Inter-node dependence management

- Inferred from the task graph edges
- Automatic **Isend** and **Irecv** calls





Distributed Support

Sequential Task Flow Paradigm on Clusters

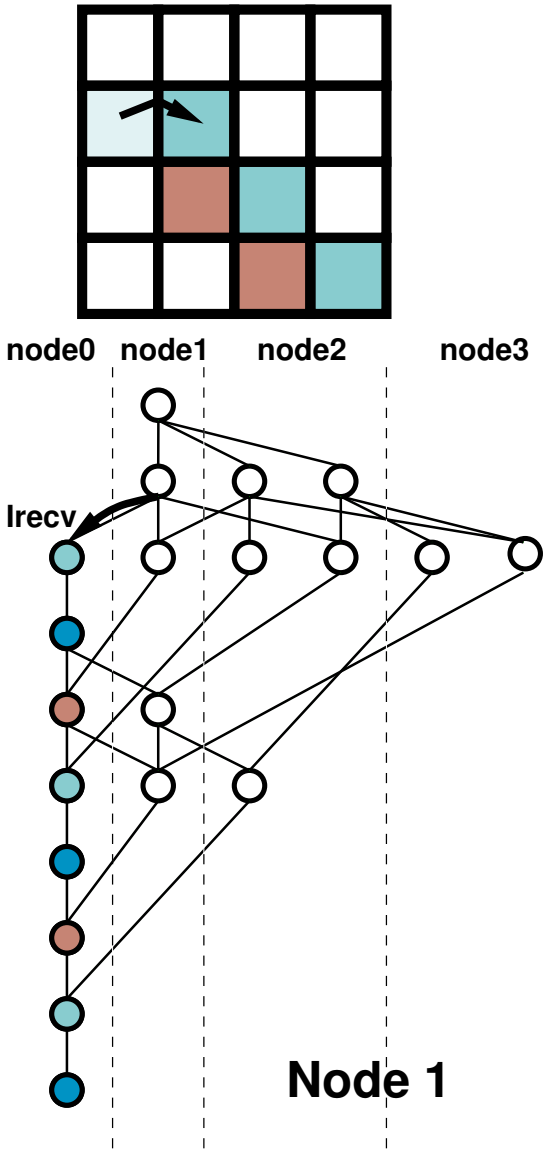
Each node unrolls the sequential task flow

Task↔Node Mapping

- Inferred from data location:
  - *Tasks move to data they modify*
- No global scheduling
- No synchronizations

Optimization

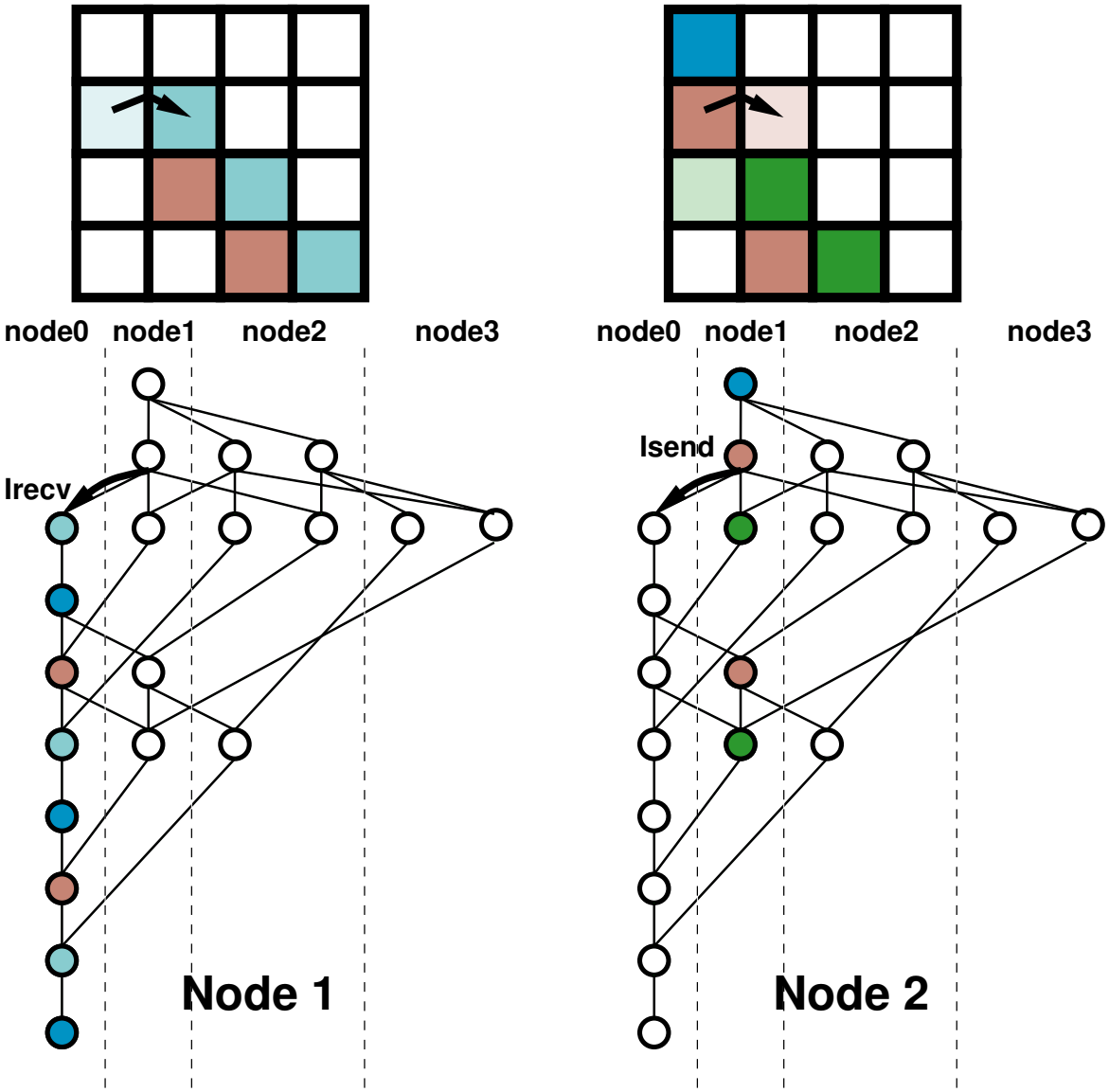
- Local DAG pruning



Communication

Nodes infer required transfers

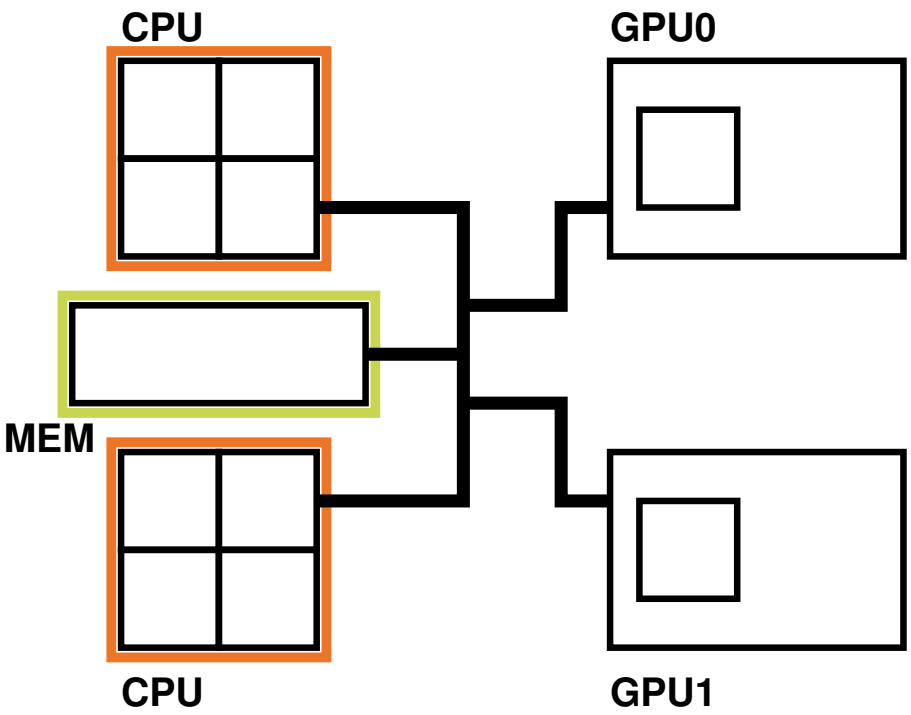
- Task dependencies



Communication

Nodes infer required transfers

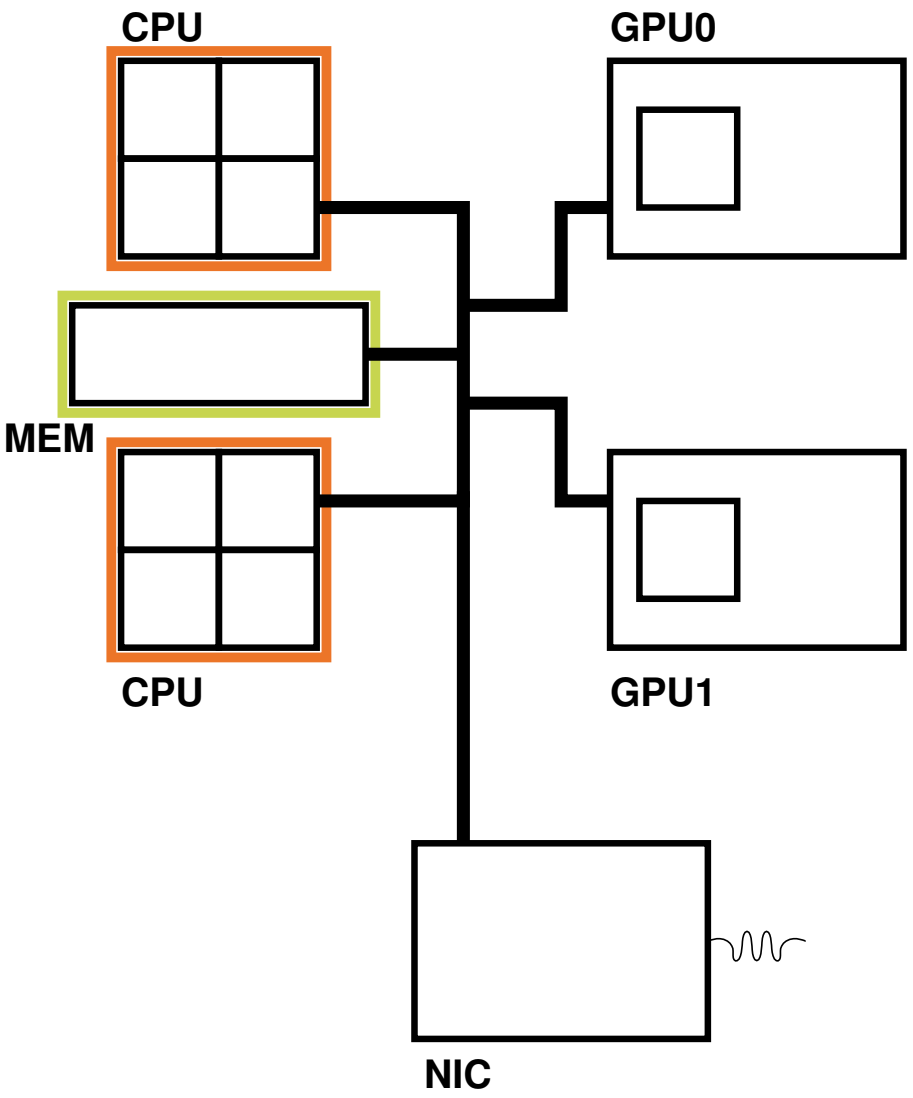
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

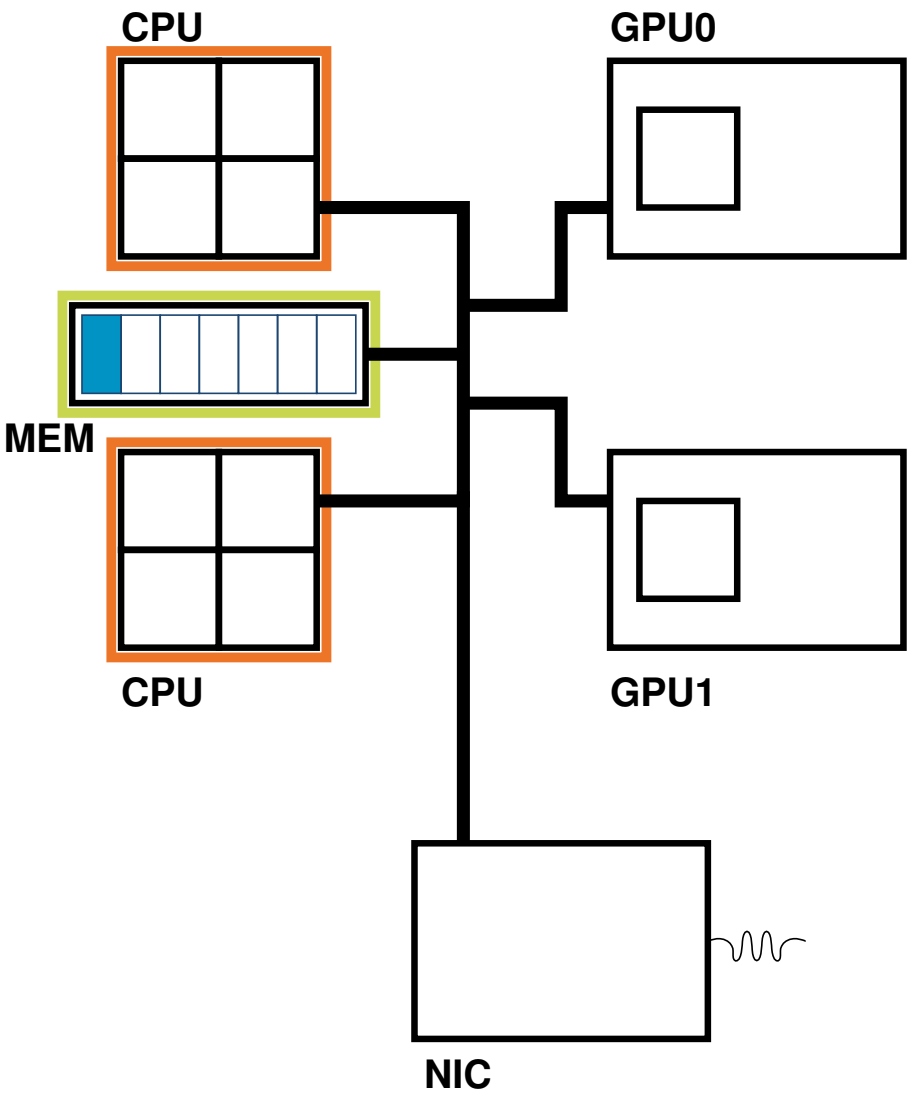
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

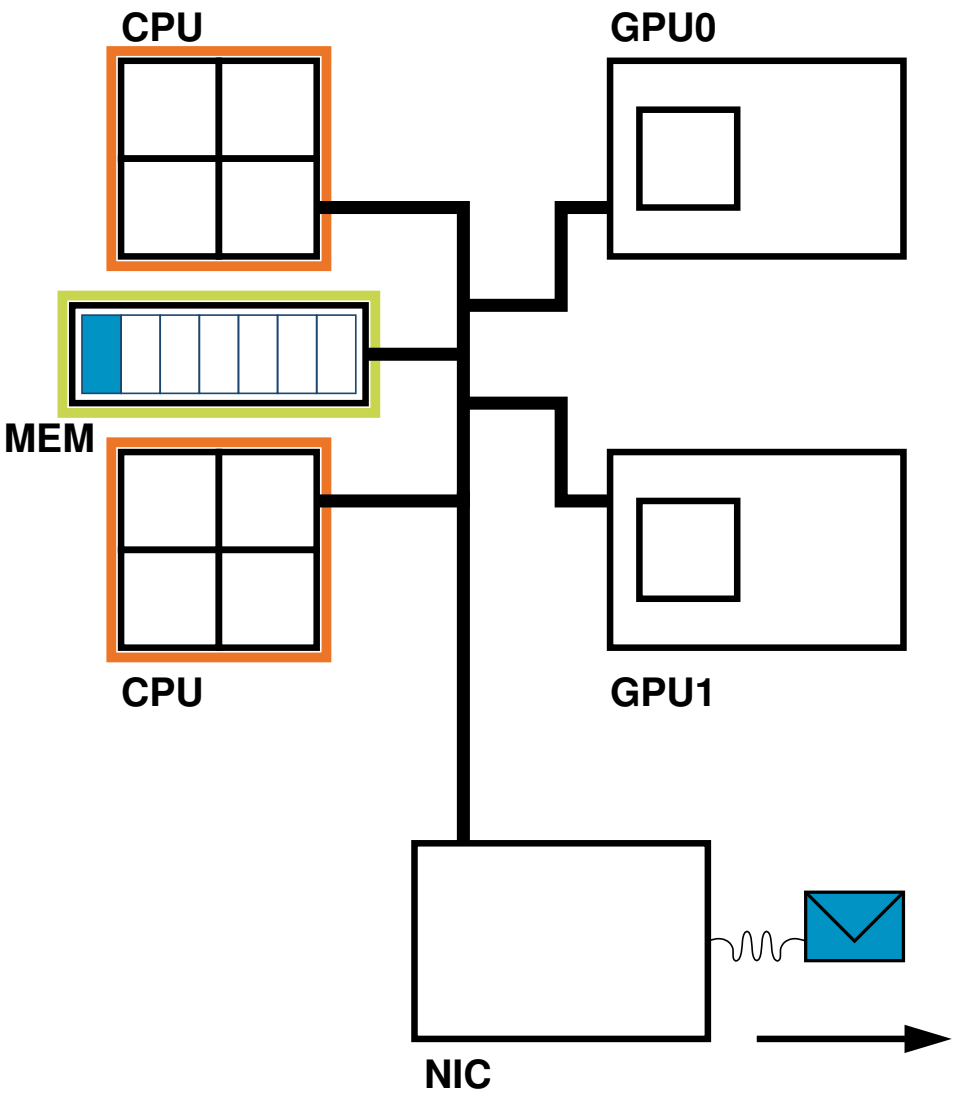
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

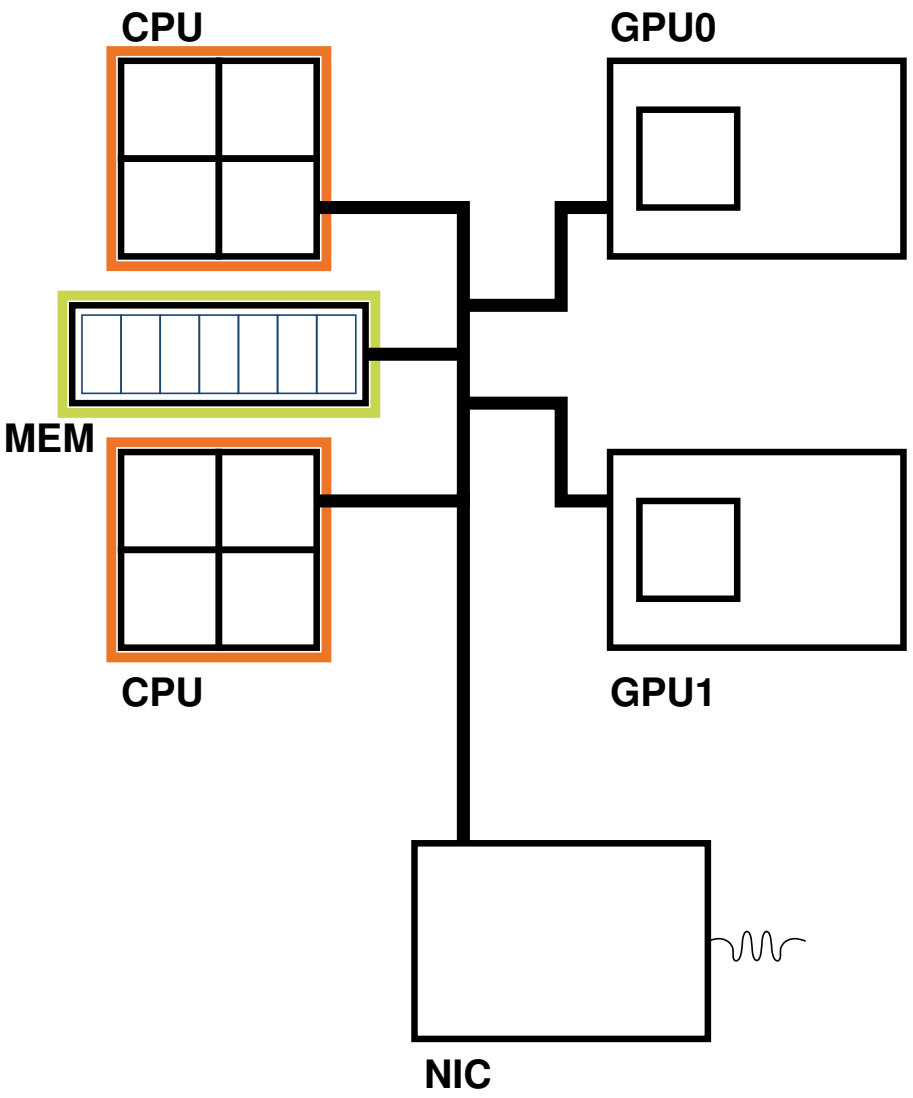
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

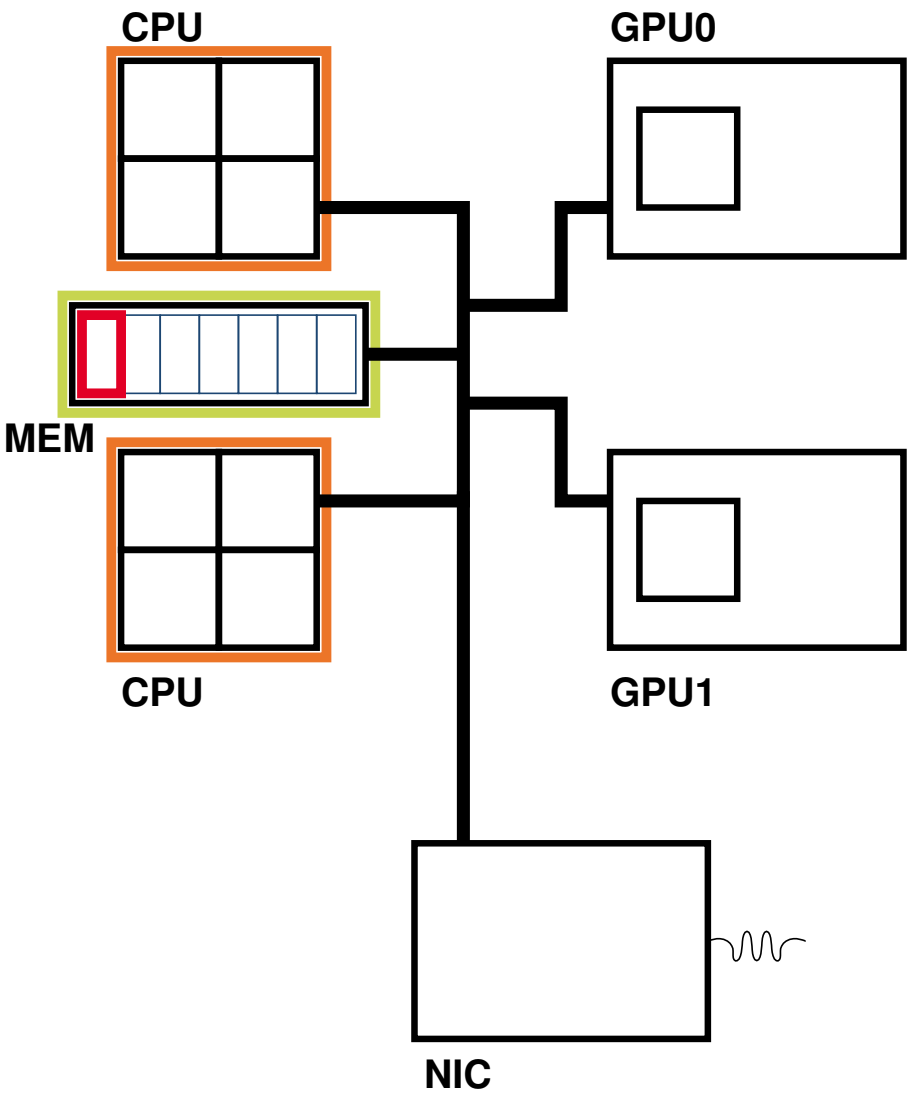
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

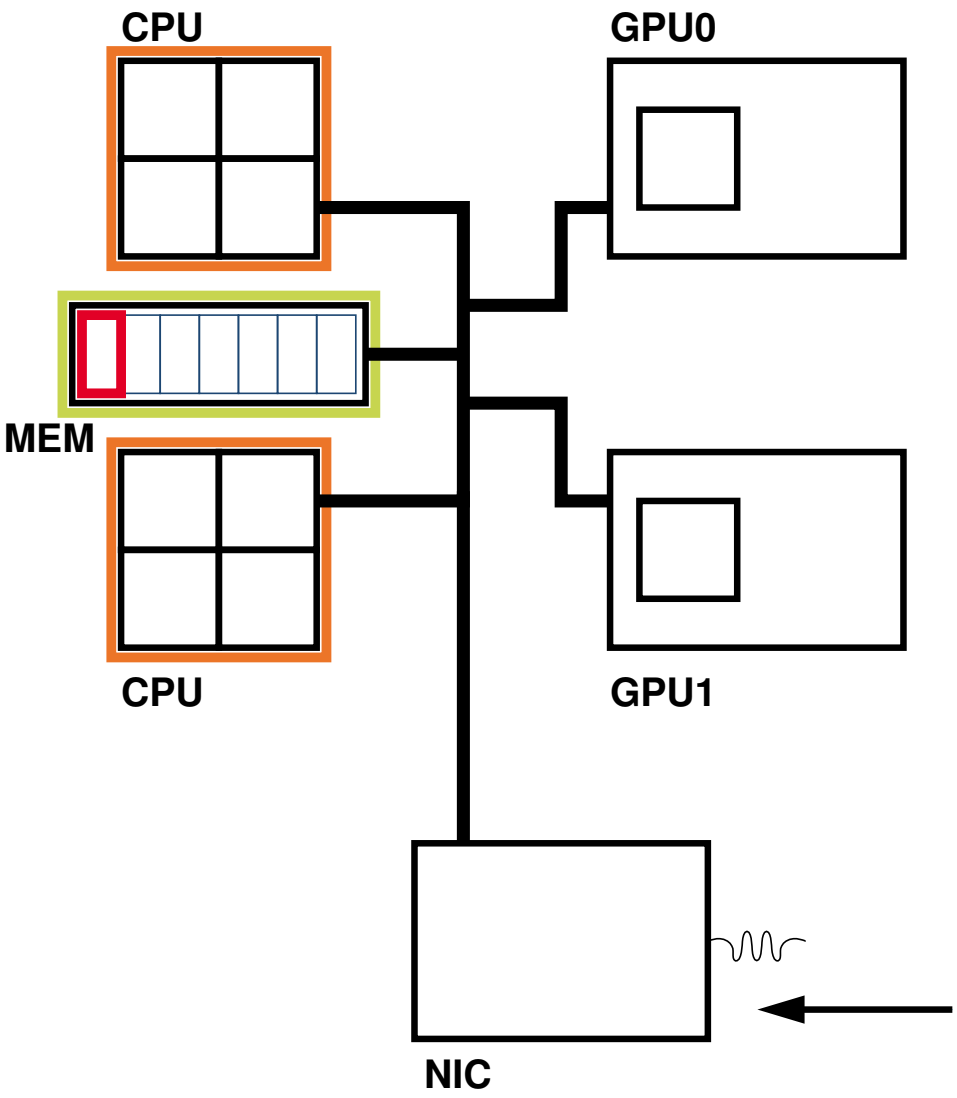
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

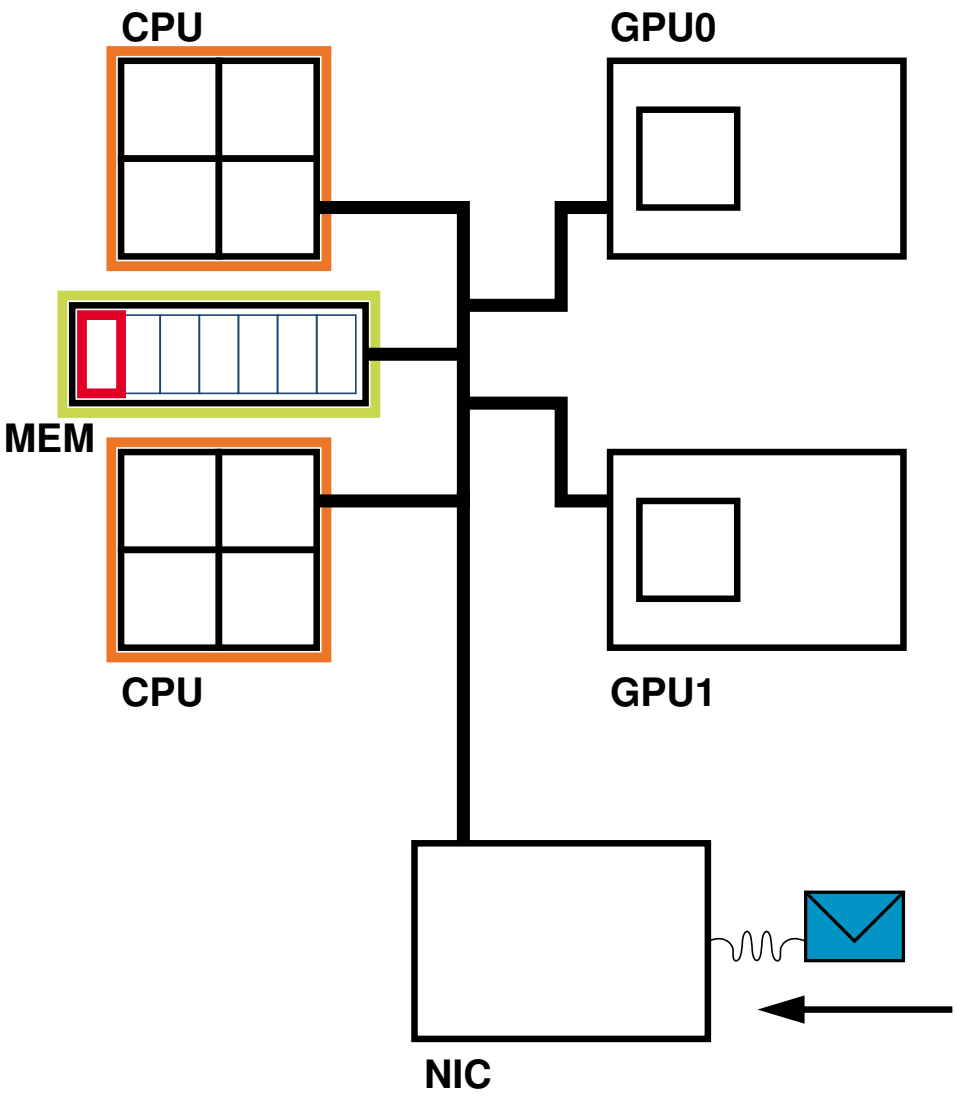
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

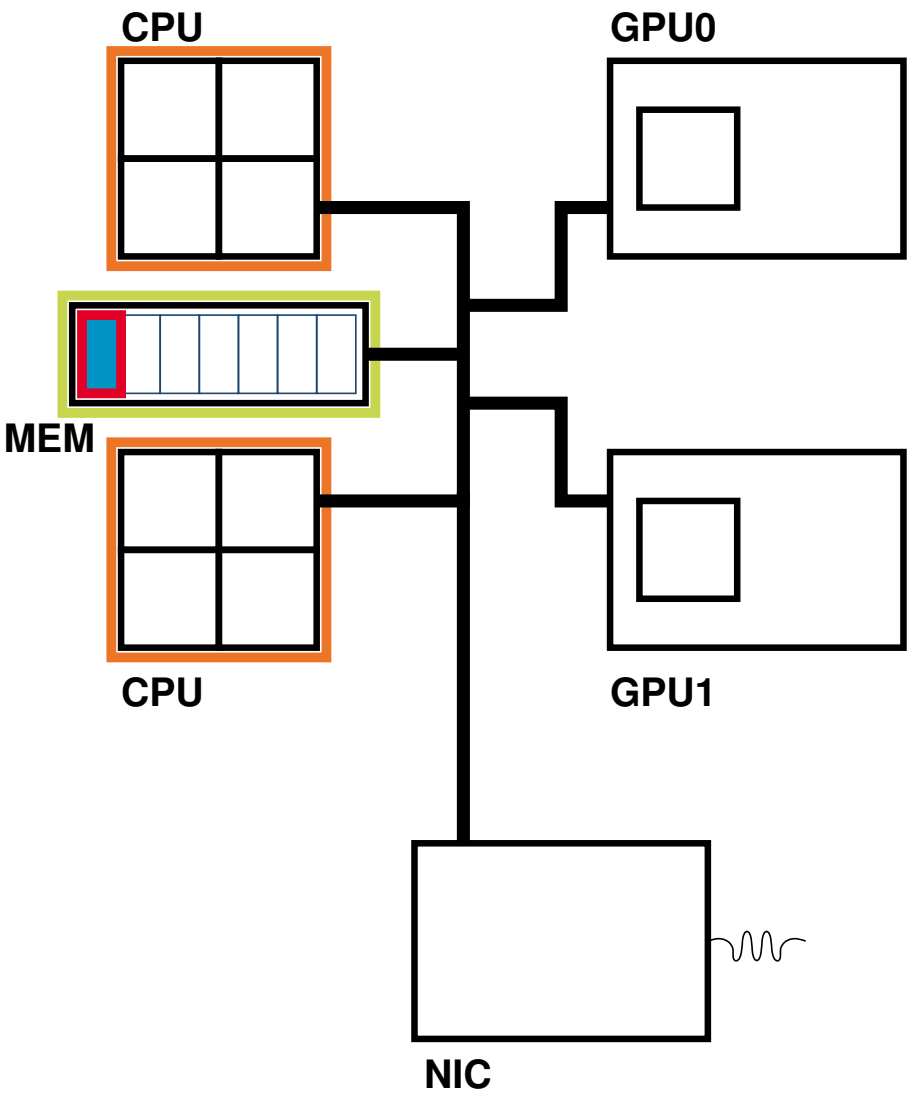
- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Communication

Nodes infer required transfers

- Task dependencies
- Automatic MPI calls
  - **Isend**
  - **Irecv**
- Tasks wait for MPI requests



Same Paradigm for Clusters (vs Single Node)

same code

```
for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j]);
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j]);
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j]);
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j]);
  }
}
task_wait_for_all();
```

Same Paradigm for Clusters (vs Single Node)

Almost same code

- MPI communicator

```
for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j], MPI_COMM_WORLD);
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j], MPI_COMM_WORLD);
  }
}
task_wait_for_all();
```

Same Paradigm for Clusters (vs Single Node)

Almost same code

- MPI communicator
- Mapping function

```
int getnode(int i, int j) { return((i%p)*q + j%q); }
```

```
for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j], MPI_COMM_WORLD, getnode(j,j));
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j], MPI_COMM_WORLD, getnode(i,j));
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j], MPI_COMM_WORLD, getnode(i,i));
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j], MPI_COMM_WORLD, getnode(i,k));
  }
}
task_wait_for_all();
```

Same Paradigm for Clusters (vs Single Node)

Almost same code

- MPI communicator
- Mapping function

```
int getnode(int i, int j) { return((i%p)*q + j%q); }
set_rank(A, getnode);

for (j = 0; j < N; j++) {
  POTRF (RW, A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++)
    TRSM (RW, A[i][j], R,A[j][j], MPI_COMM_WORLD);
  for (i = j+1; i < N; i++) {
    SYRK (RW, A[i][i], R,A[i][j], MPI_COMM_WORLD);
    for (k = j+1; k < i; k++)
      GEMM (RW, A[i][k],
        R,A[i][j], R,A[k][j], MPI_COMM_WORLD);
  }
}
task_wait_for_all();
```

Hands-on Session

Session Part 3: MPI Support

- Training Platform 'poincare'
- Registered attendee should have received login/passwd credentials and connection info
- Reserved computing nodes:

```
- gpu nodes:     LoadLeveler class 'clgpu'
- batch script submission:     llsubmit <script.sh>
```

- Web site:

  http://starpu.gforge.inria.fr/tutorials/2016-06-PATC/

6

High-level programming

High-Level StarPU Programming

Pragma-Oriented Languages

- OpenMP 4.x Compiler “Klang-OMP”: source-to-source translation of C/C++ into StarPU API calls

Klang-omp OpenMP C/C++ Compiler

High level programming

Klang-omp OpenMP C/C++ Compiler

High level programming

- Translate directives into runtime system API calls
  - **StarPU** Runtime System
  - XKaapi Runtime System (INRIA Team MOAIS)

Klang-omp OpenMP C/C++ Compiler

High level programming

- Translate directives into runtime system API calls
  - **StarPU** Runtime System
  - XKaapi Runtime System (INRIA Team MOAIS)
- OpenMP 3.1
  - Virtually full support
- OpenMP 4.0
  - Dependent tasks
  - Heterogeneous targets (on-going work)

Klang-omp OpenMP C/C++ Compiler

High level programming

- Translate directives into runtime system API calls
  - **StarPU** Runtime System
  - XKaapi Runtime System (INRIA Team MOAIS)
- OpenMP 3.1
  - Virtually full support
- OpenMP 4.0
  - Dependent tasks
  - Heterogeneous targets (on-going work)
- LLVM-based source-to-source compiler
- Builds on open source Intel compiler `clang-omp`

Available on:

- K'Star project website – **http://kstar.gforge.inria.fr/**

Klang-omp Example: Tasks

```
int item[N];

void g(int);

void f()
{
#pragma omp parallel
    {
#pragma omp single
        {
            int i;
            for (i=0; i<N; i++)
#pragma omp task untied
                g(item[i]);
        }
    }
}
```

Klang-omp Example: Dependencies

```
void f()
{
    int a;

#pragma omp parallel
#pragma omp single
    {
#pragma omp task shared(a) depend(out: a)
        foo(&a);

#pragma omp task shared(a) depend(in: a)
        bar(&a);
    }
}
```

Klang-omp Example: Targets

```
#pragma omp declare target
extern void g(int *A, int *B, int *C);
#pragma omp end declare target

int A[N], B[N], C[N];

void f()
{
    int a;
#pragma omp parallel
#pragma omp master
#pragma omp target map(to: A, B) map(from: C)
    g(A, B, C);
}
```

Hands-on Session

Short Session Part 4: OpenMP

- Training Platform 'poincare'
- Registered attendee should have received login/passwd credentials and connection info
- Try out the following sequence

```
cd
source /gpfslocal/pub/training/runtime_june2016/openmp/environment
cp -r /gpfslocal/pub/training/runtime_june2016/openmp/Cholesky .
cd Cholesky
make
./cholesky_omp4.starpu
```

- Web site:

  http://starpu.gforge.inria.fr/tutorials/2016-06-PATC/

Summary

1. StarPU Internals
2. Scheduling
3. Data management
4. Debugging / Monitoring
5. Distributed computing
6. High-level programming
7. Advanced scheduling
8. Advanced Data Management
9. Advanced Debugging / Monitoring

7

Advanced scheduling

Advanced Scheduling

Interoperability and composition

- Project H2020 INTERTWinE
- Parallel tasks
- Scheduling contexts

Advanced Scheduling

Interoperability and composition

- Project H2020 INTERTWinE
- Parallel tasks
- Scheduling contexts

Submission-side control

- Look-ahead depth...
- ... vs resource subscription

Interoperability

How to Make Runtimes, Libs Cooperate?

Interoperability

How to Make Runtimes, Libs Cooperate?

- Project INTERTWinE (EU H2020, 3-years, 2015-2018)
  - Task-based runtimes: StarPU, OmpSs, PaRSEC, OpenMP
  - Networking APIs: MPI, GASPI
  - Libraries: Plasma, DPlasma
  - Applications

How to Make Runtimes, Libs Cooperate?

- Project INTERTWinE (EU H2020, 3-years, 2015-2018)
  - Task-based runtimes: StarPU, OmpSs, PaRSEC, OpenMP
  - Networking APIs: MPI, GASPI
  - Libraries: Plasma, DPlasma
  - Applications
- Cooperative resource allocation and management
  - Cores
  - Accelerators
  - Memory
  - Pinned memory segments
  - ...

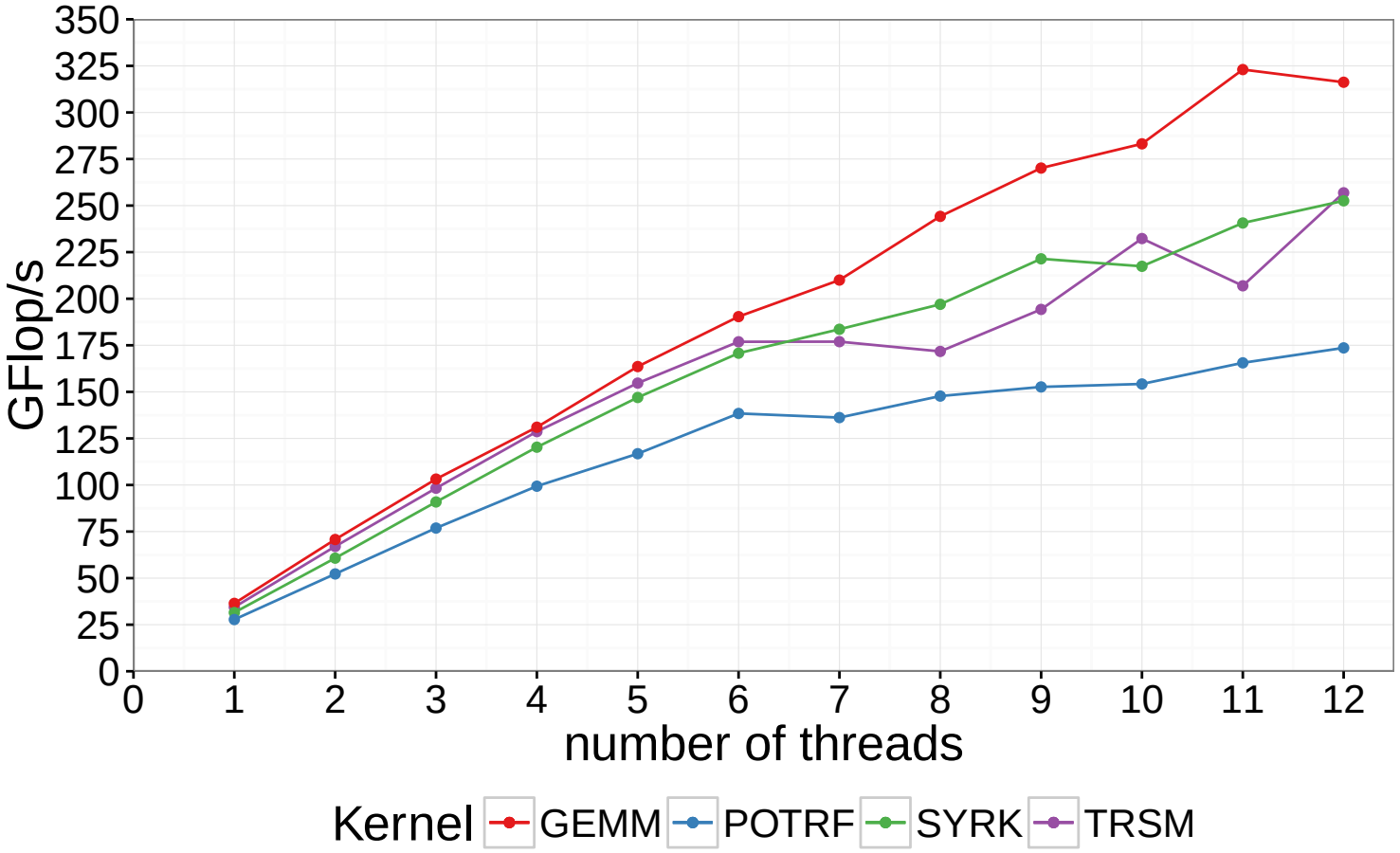
www.intertwine-project.eu

Multicore CPUs: Parallel Tasks

Multicore CPUs: Parallel Tasks (T. Cojean)

Kernel sweet spots: example with Cholesky factorization kernels

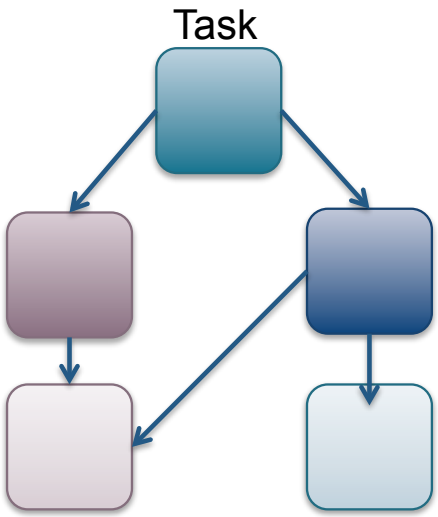
(1x Xeon E5-2680v3 2.5GHz 12 cores)



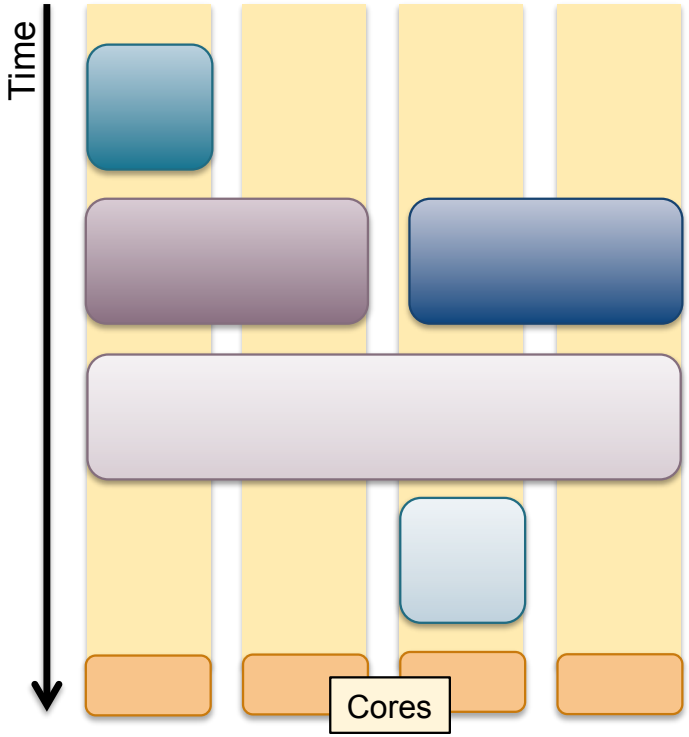
Multicore CPUs: Parallel Tasks

Rationale

- Run parallel kernels on multiple CPU cores
- Address CPU/GPU computing power imbalance
- Address nested-runtime interoperability



Task DAG



Execution

Multicore CPUs: Parallel Tasks

Rationale

- Run parallel kernels on multiple CPU cores
- Address CPU/GPU computing power imbalance
- Address nested-runtime interoperability

Reduce computing power imbalance between CPU and GPU

- Big kernel for GPU
- Small kernel for a single CPU core
- Run “bigger” kernel on several CPU cores

Multicore CPUs: Parallel Tasks

Rationale

- Run parallel kernels on multiple CPU cores
- Address CPU/GPU computing power imbalance
- Address nested-runtime interoperability

Reduce computing power imbalance between CPU and GPU

- Big kernel for GPU
- Small kernel for a single CPU core
- Run “bigger” kernel on several CPU cores

Make use of existing parallel kernels/codes

- Interoperability
- Libraries: BLAS, FFT, . . .
- OpenMP code

Multicore CPUs – Technical details

Two flavors of **parallel tasks**

Multicore CPUs – Technical details

Two flavors of **parallel tasks**

Fork-mode

- StarPU provides threads on the participating cores

Multicore CPUs – Technical details

Two flavors of **parallel tasks**

Fork-mode

- StarPU provides threads on the participating cores

SPMD-mode

- StarPU launches the task on a single core
- ...and let the task create its own threads
  - Black-box mode

Multicore CPUs – Technical details

Two flavors of **parallel tasks**

Fork-mode

- StarPU provides threads on the participating cores

SPMD-mode

- StarPU launches the task on a single core
- . . . and let the task create its own threads
  - Black-box mode

Locality enforcement in NUMA context

- Combined worker threads

Composing Multiple Codes

Rationale

Composing Multiple Codes

Rationale

- Sharing computing resources. . .

Composing Multiple Codes

Rationale

- Sharing computing resources. . .
- . . . among multiple DAGs

Composing Multiple Codes

Rationale

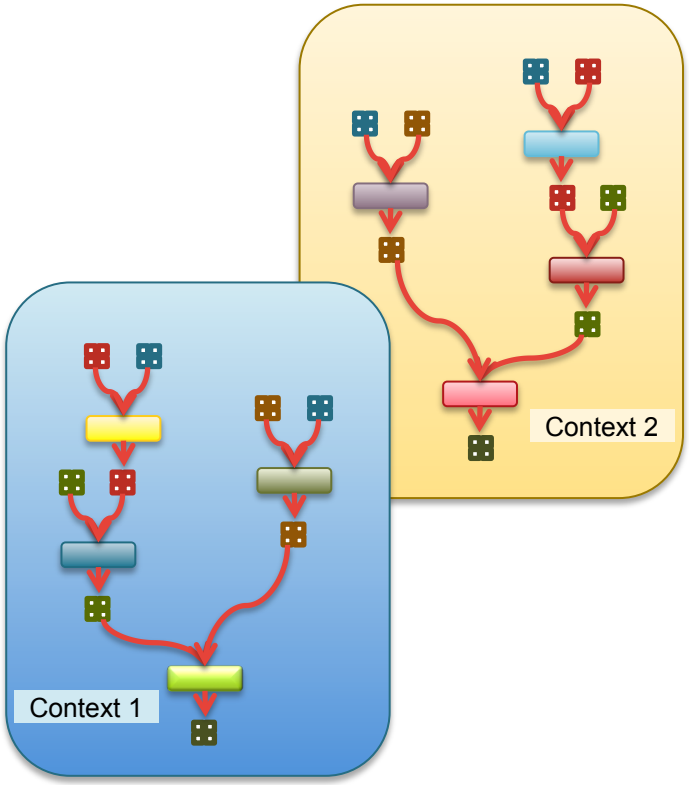
- Sharing computing resources. . .
- . . . among multiple DAGs
- . . . simultaneously

Composing Multiple Codes

Rationale

- Sharing computing resources. . .
- . . . among multiple DAGs
- . . . simultaneously

Scheduling Contexts



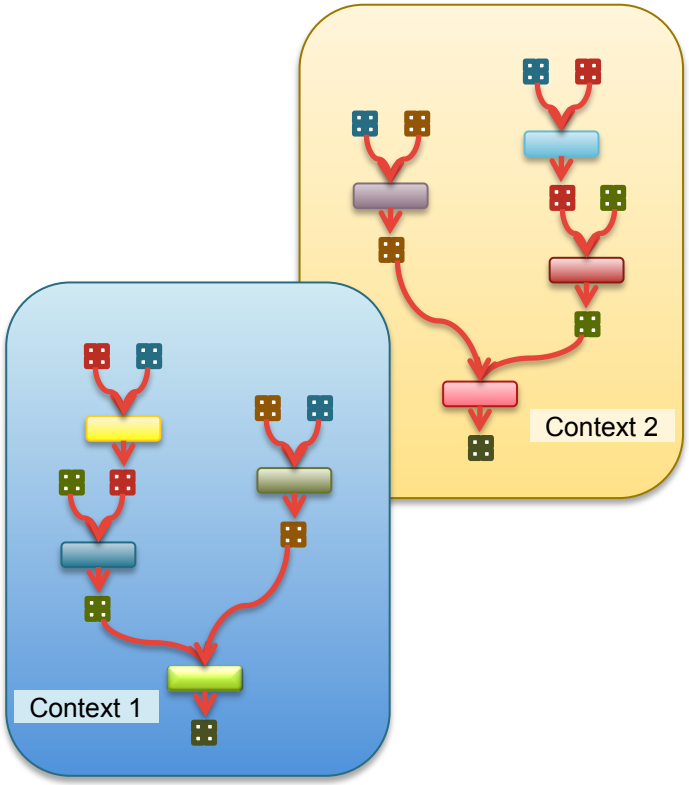
Composing Multiple Codes

Rationale

- Sharing computing resources. . .
- . . . among multiple DAGs
- . . . simultaneously

Scheduling Contexts

- Map DAGs on subsets of computing units



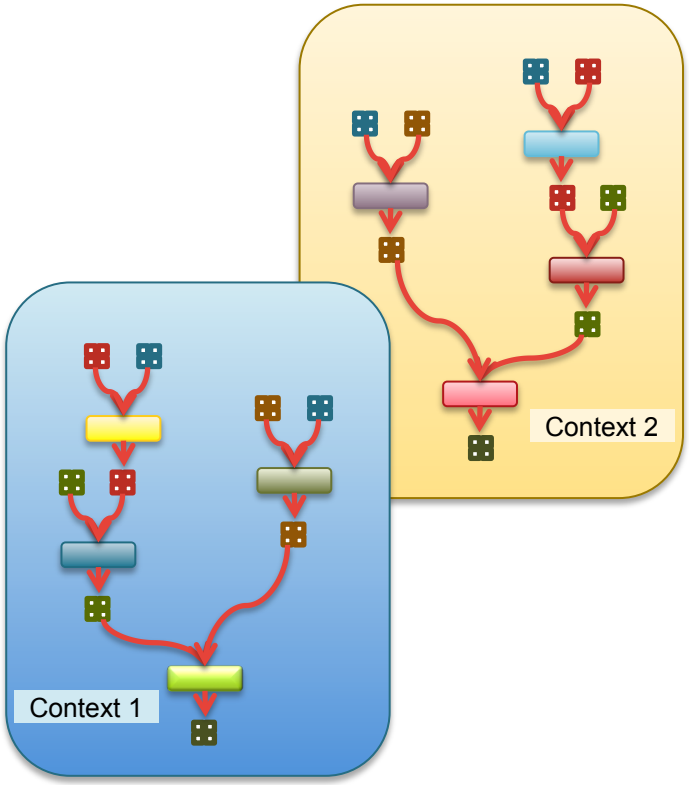
Composing Multiple Codes

Rationale

- Sharing computing resources. . .
- . . . among multiple DAGs
- . . . simultaneously

Scheduling Contexts

- Map DAGs on subsets of computing units
- Isolate competing kernels or library calls
  - OpenMP kernel, Intel MKL, etc.



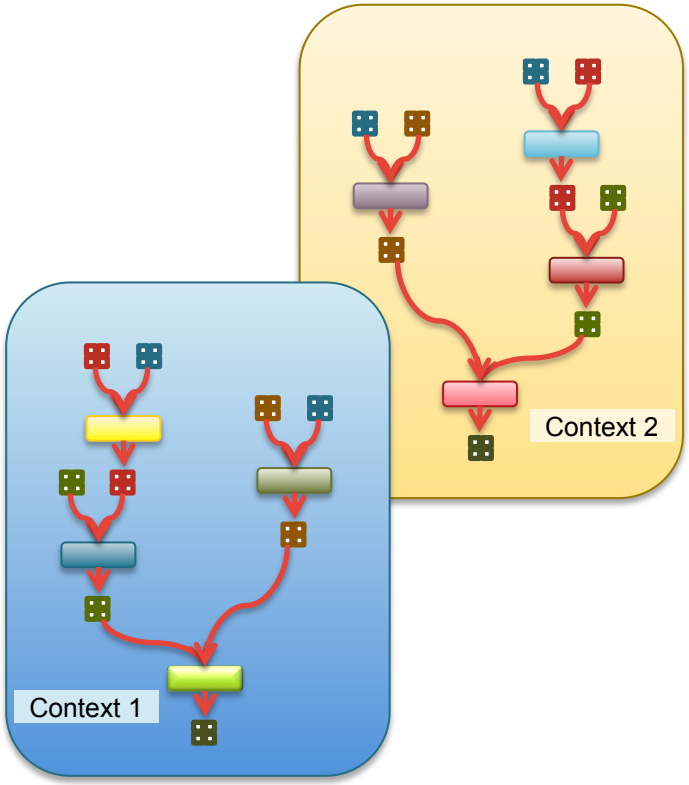
Composing Multiple Codes

Rationale

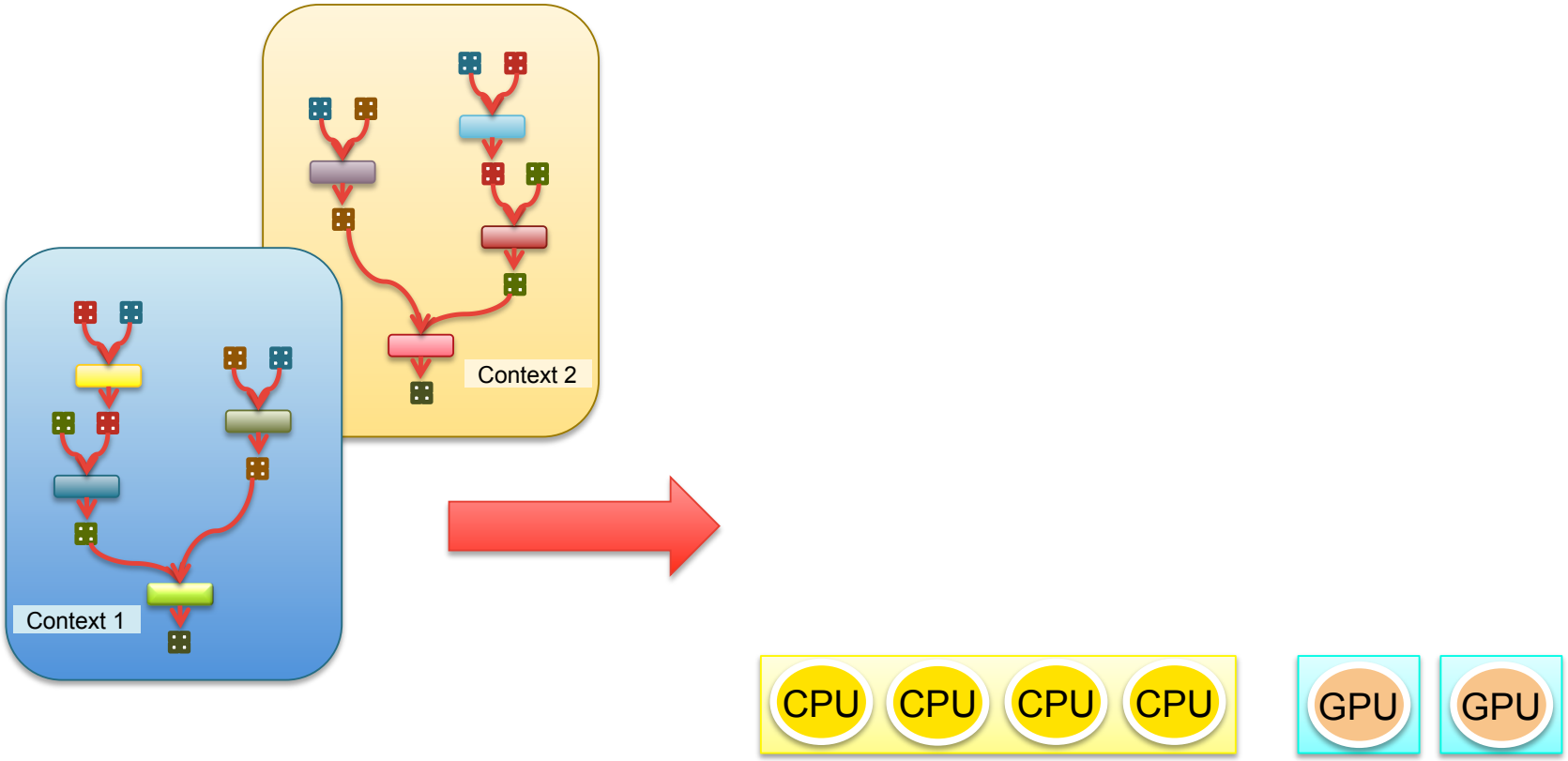
- Sharing computing resources. . .
- . . . among multiple DAGs
- . . . simultaneously

Scheduling Contexts

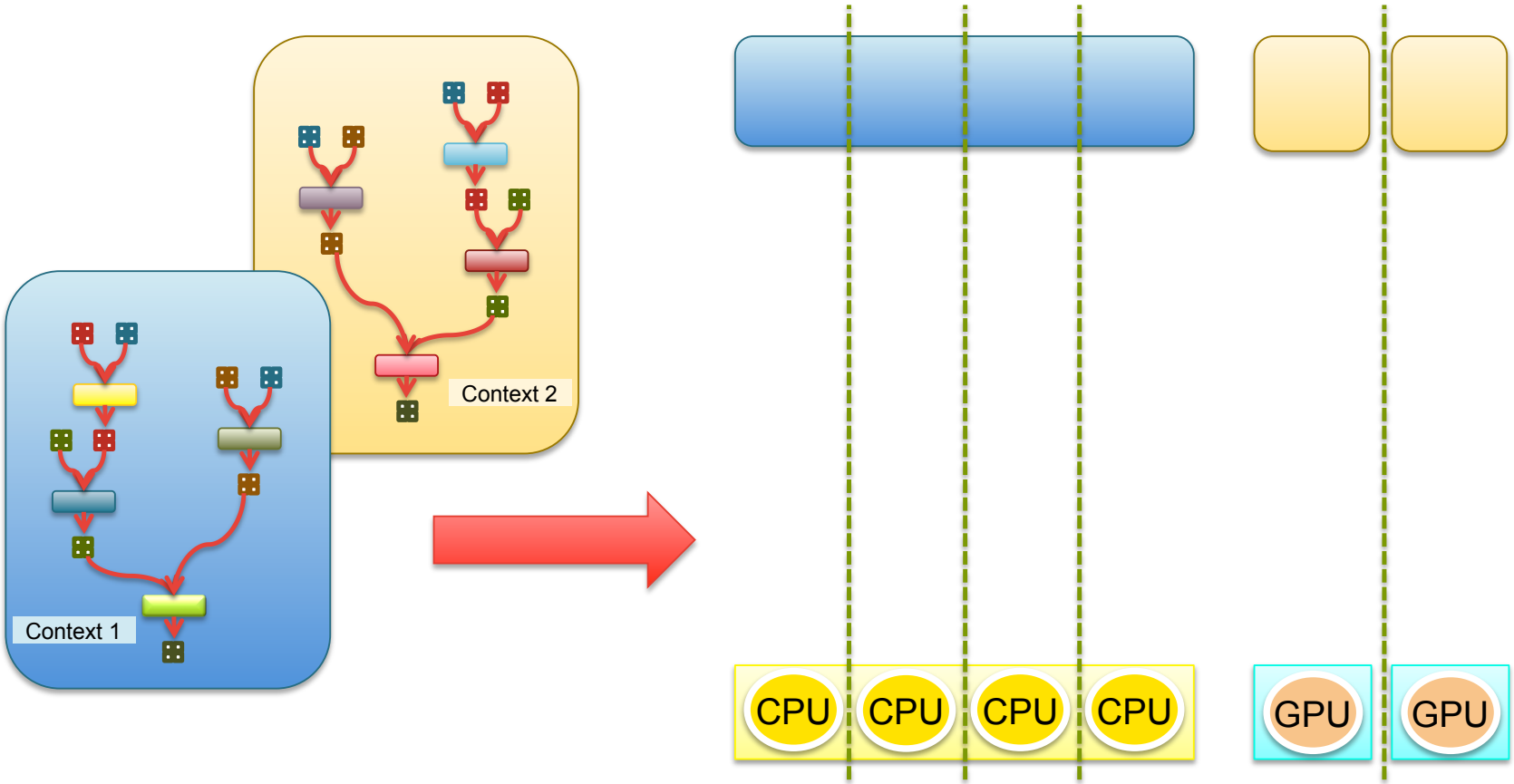
- Map DAGs on subsets of computing units
- Isolate competing kernels or library calls
  - OpenMP kernel, Intel MKL, etc.
- Select scheduling policy per context



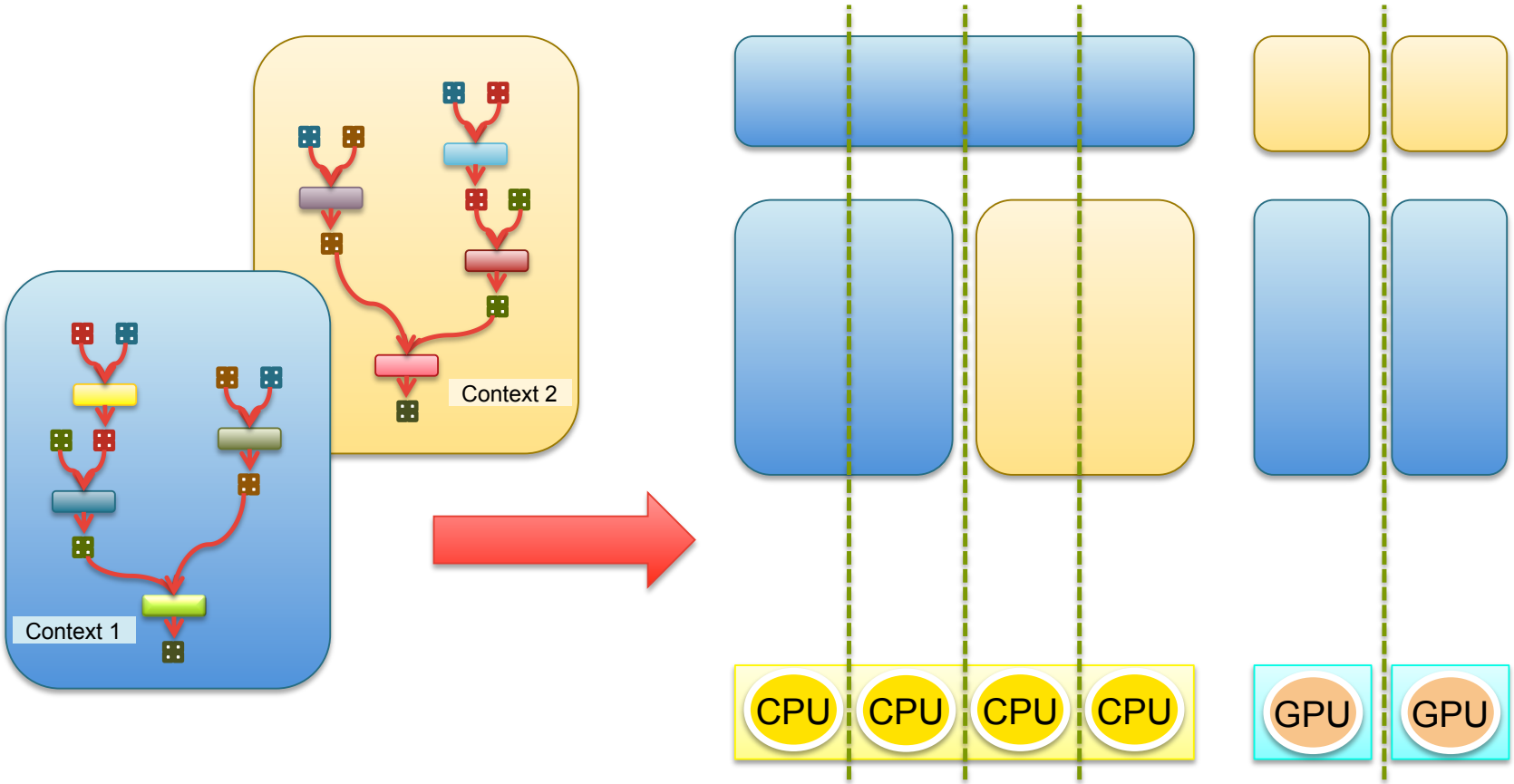
Contexts: Dynamic Resource Management



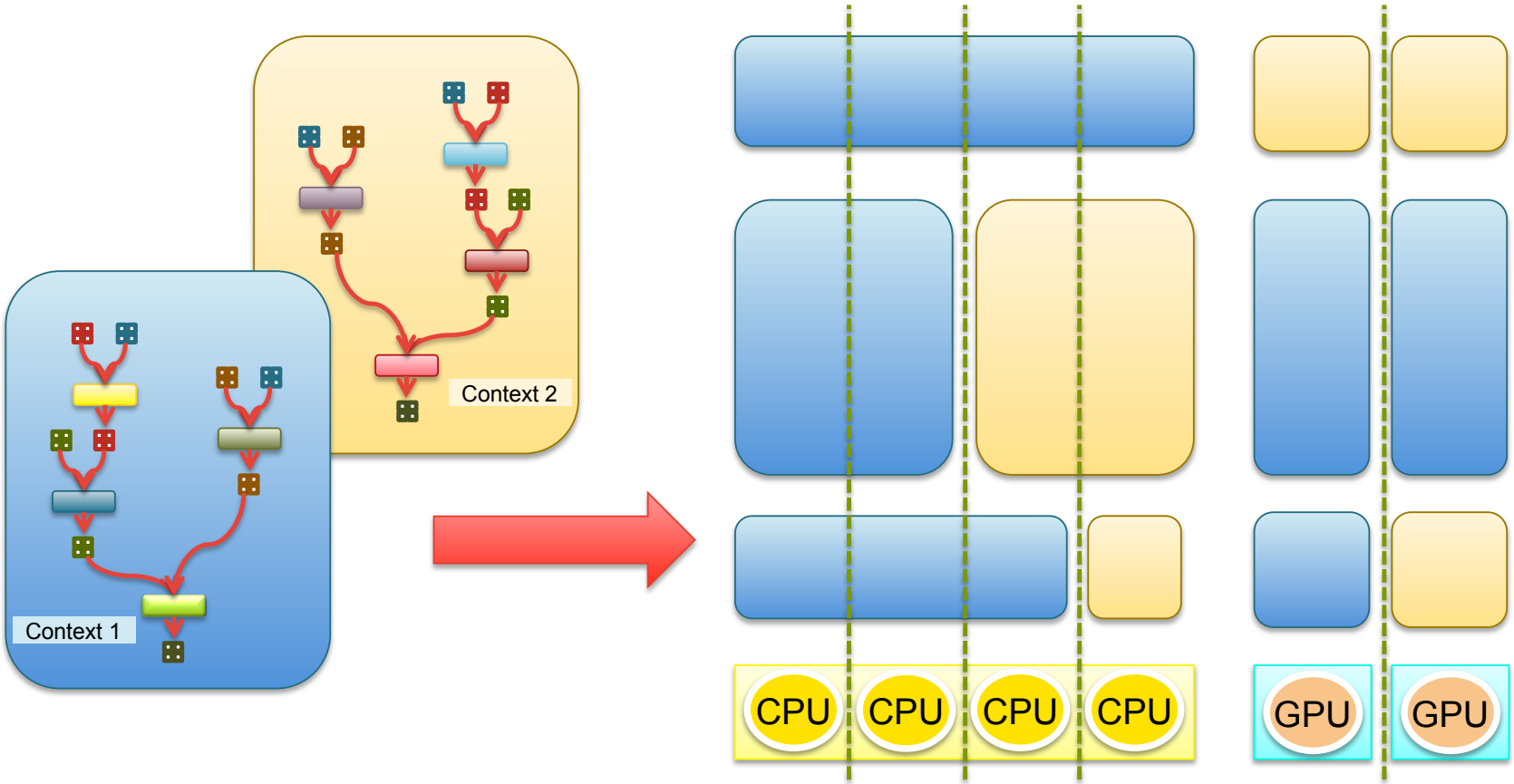
Contexts: Dynamic Resource Management



Contexts: Dynamic Resource Management



Contexts: Dynamic Resource Management



Submission-Side Matters

Decoupled submission/execution stages

Submission-Side Matters

Decoupled submission/execution stages

Benefits

- Lookahead build-up
  - Anticipation
- Advanced scheduling
  - Long-term planning
- Prefetch
  - Computation/transfer overlap

Submission-Side Matters

Decoupled submission/execution stages

Benefits

- Lookahead build-up
  - Anticipation
- Advanced scheduling
  - Long-term planning
- Prefetch
  - Computation/transfer overlap

Risks

- Resource subscription
  - StarPU internal data structures (tasks)
  - Networking buffers
- Processing overhead
  - Algorithm complexity
- (Distributed) Scalability
  - Strict global sequential task flow

Submission-Side Control

Control flow on task submission

Submission-Side Control

Control flow on task submission

Pause/resume task submission on condition

- Max number of *in-flight* tasks
- Others (memory. application phase, etc.)

Submission-Side Control

Control flow on task submission

Pause/resume task submission on condition

- Max number of *in-flight* tasks
- Others (memory. application phase, etc.)

Dead-lock free by design

- Sequential task flow property

Submission-Side Control

Control flow on task submission

Pause/resume task submission on condition

- Max number of *in-flight* tasks
- Others (memory. application phase, etc.)

Dead-lock free by design

- Sequential task flow property

```
/* API function: */
starpu_task_wait_for_n_submitted( n ) ;
```

Submission-Side Control

Control flow on task submission

Pause/resume task submission on condition

- Max number of *in-flight* tasks
- Others (memory. application phase, etc.)

Dead-lock free by design

- Sequential task flow property

```
/* API function: */
starpu_task_wait_for_n_submitted( n ) ;
```

```
# Environment variables
$ export STARPU_LIMIT_MAX_SUBMITTED_TASKS=10000
$ export STARPU_LIMIT_MIN_SUBMITTED_TASKS=9000
$ my_program
...
```

Submission-Side Control

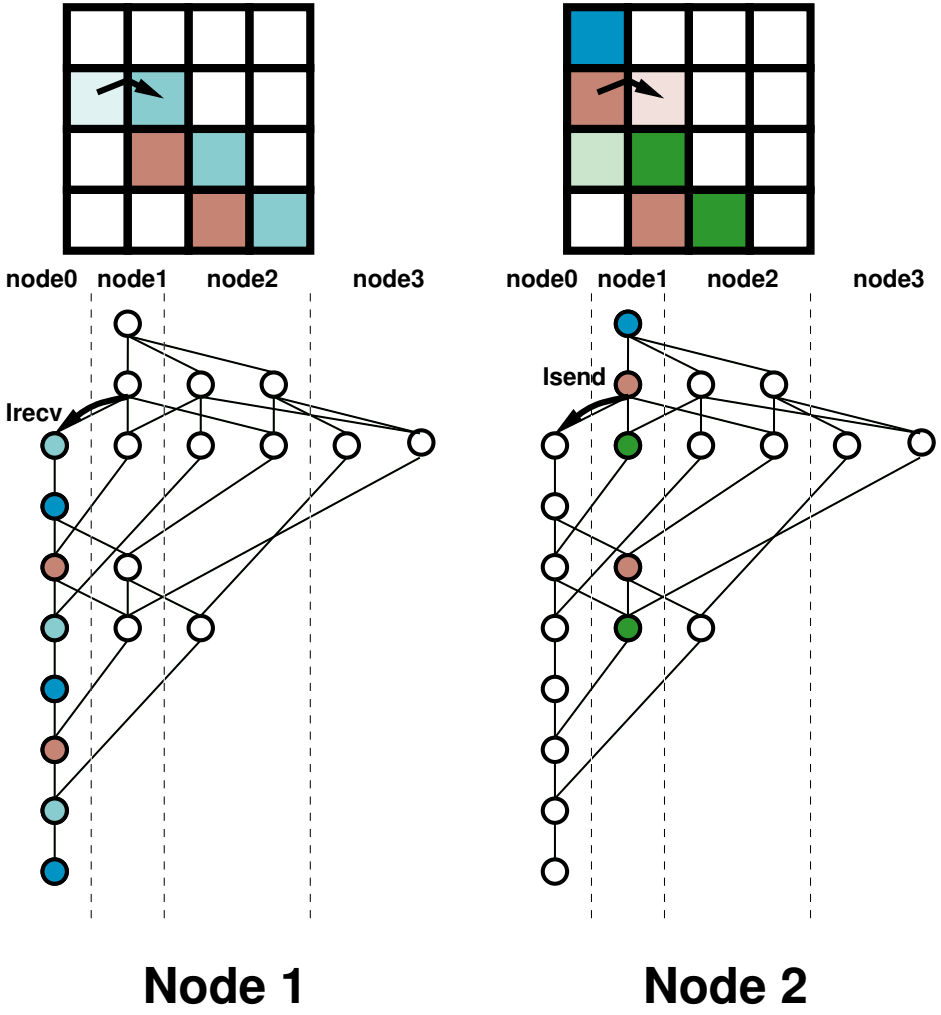
Optimisation on task submission

Distributed sessions

- Relax global sequential task flow
- Prune irrelevant tasks
  - Non-local tasks with no dependence edge to/from node

Dead-lock free by design

- Sequential task flow property



8

Advanced Data Management

Advanced Data Management

Advanced Data Management

Heterogeneous data layout

- Multiformat support

Advanced Data Management

Heterogeneous data layout

- Multiformat support

Large workloads

- Out-of-core support

Data Layout

Heterogeneous platforms

- Heterogeneous data layout requirements
- Example:
  - Arrays of Structures (AoS), for CPU cache locality
  - vs Structures of Arrays (SoA), for GPU coalesced memory accesses
  - vs Arrays of Structures of Arrays (AoSoA), for MIC/Xeon Phi
  - . . . any other data layout

Data Layout

Heterogeneous platforms

- Heterogeneous data layout requirements
- Example:
  - Arrays of Structures (AoS), for CPU cache locality
  - vs Structures of Arrays (SoA), for GPU coalesced memory accesses
  - vs Arrays of Structures of Arrays (AoSoA), for MIC/Xeon Phi
  - . . . any other data layout

StarPU enables Multiformat kernel implementations

- User-provided data layout conversion codelets. . .
- . . . automatically called upon transfers between devices

Multiformat

Example

- Declare conversion codelets

```
/* Conversion codelets */
struct starpu_multiformat_data_interface_ops format_ops = {
    .cuda_elemsize = 2 * sizeof(float),
    .cpu_to_cuda_cl = &cpu_to_cuda_cl,

    .cuda_to_cpu_cl = &cuda_to_cpu_cl,
    .cpu_elemsize = 2 * sizeof(float),
    ...
};

/* Multiformat handle registration */
starpu_multiformat_data_register(handle, 0,
                                 &array_of_structs, NX, &format_ops);
```

Multiformat

Example

- Declare conversion codelets
- Array of structures for CPU

```
/* CPU Computation Kernel */

void
multiformat_scal_cpu_func(void *buffers[],void *cl_arg) {
    struct point *aos;
    unsigned int n;

    aos = STARPU_MULTIFORMAT_GET_CPU_PTR(buffers[0]);
    n = STARPU_MULTIFORMAT_GET_NX(buffers[0]);
    ...
}
```

Multiformat

Example

- Declare conversion codelets
- Array of structures for CPU
- Structure of arrays for NVidia CUDA GPU

```
/* GPU Computation Kernel */

extern "C" void
multiformat_scal_cuda_func(void *buffers[],void *cl_arg) {
    unsigned int n;
    struct struct_of_arrays *soa;

    soa = (struct struct_of_arrays *)
                STARPU_MULTIFORMAT_GET_CUDA_PTR(buffers[0]);
    n = STARPU_MULTIFORMAT_GET_NX(buffers[0]);

    ...
}
```

Large workloads

Using disks as StarPU memory nodes

- Out-of-Core

Large workloads

Using disks as StarPU memory nodes

- Out-of-Core
- Enable StarPU to evict temporarily unused data to disk

Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)

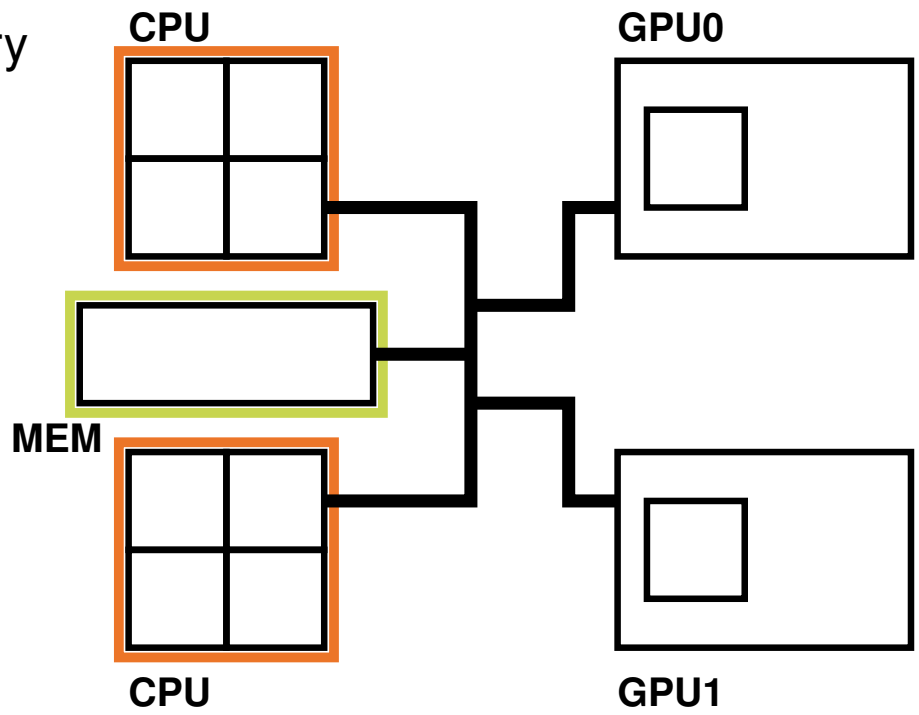
Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)



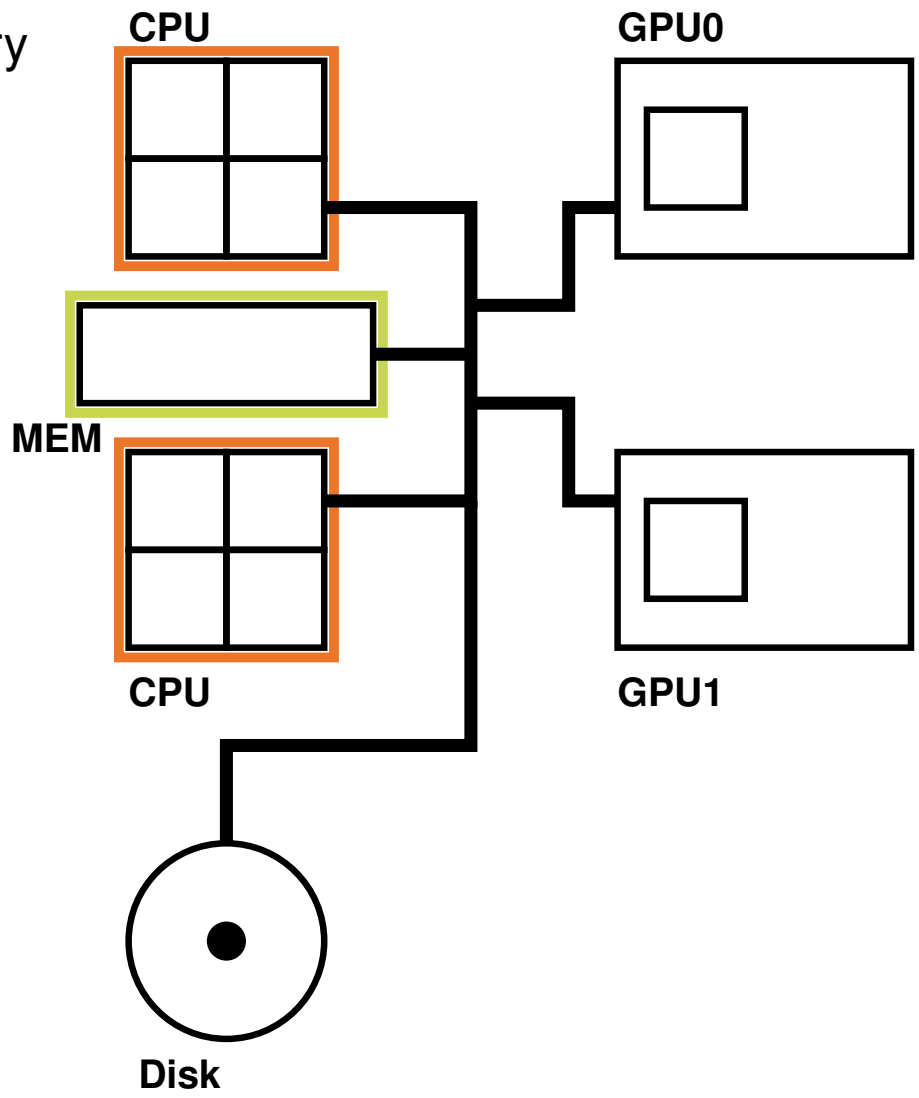
Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)



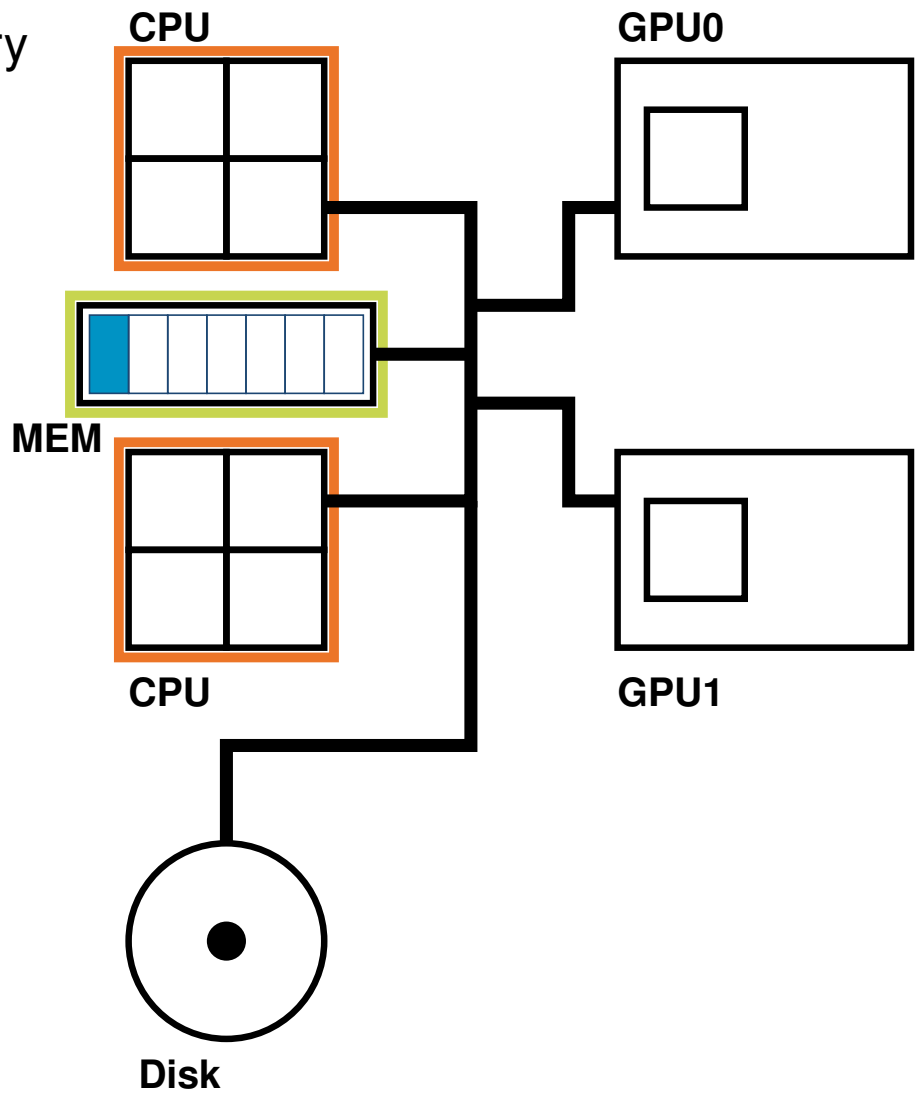
Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)



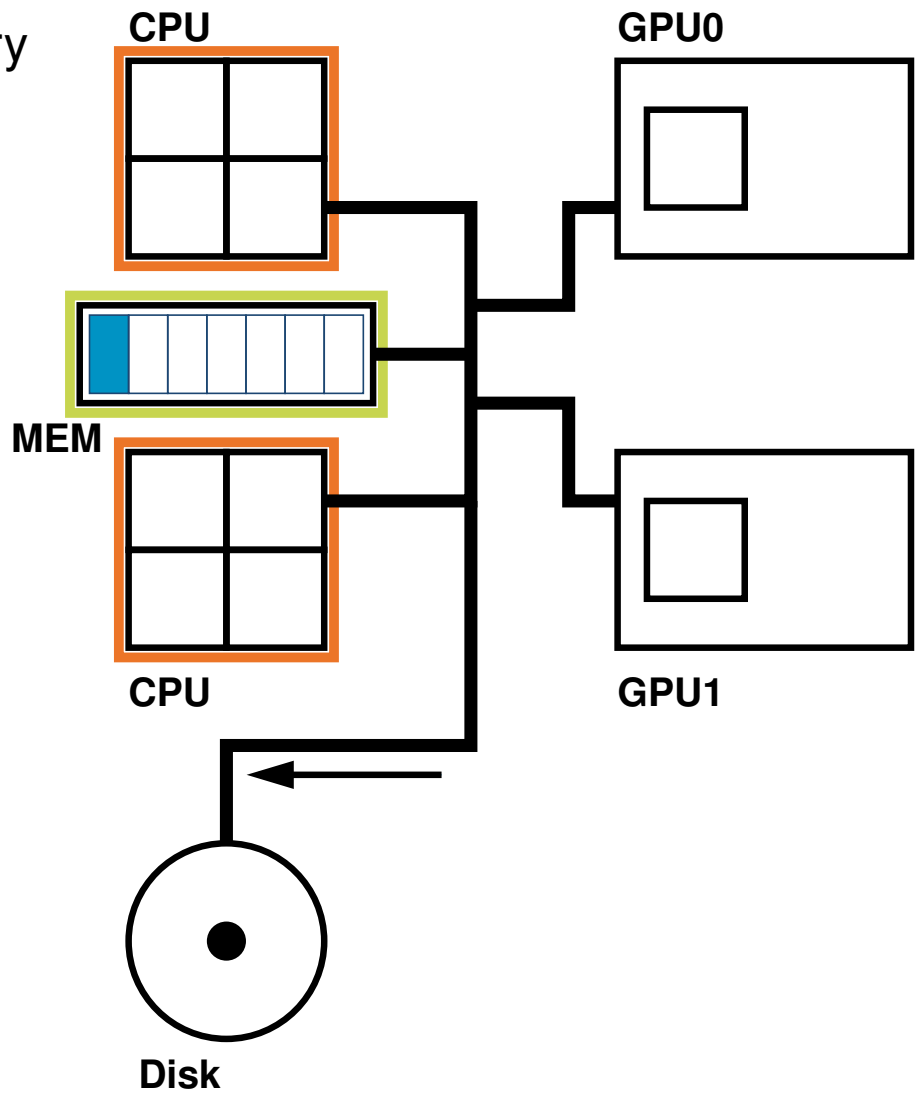
Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)



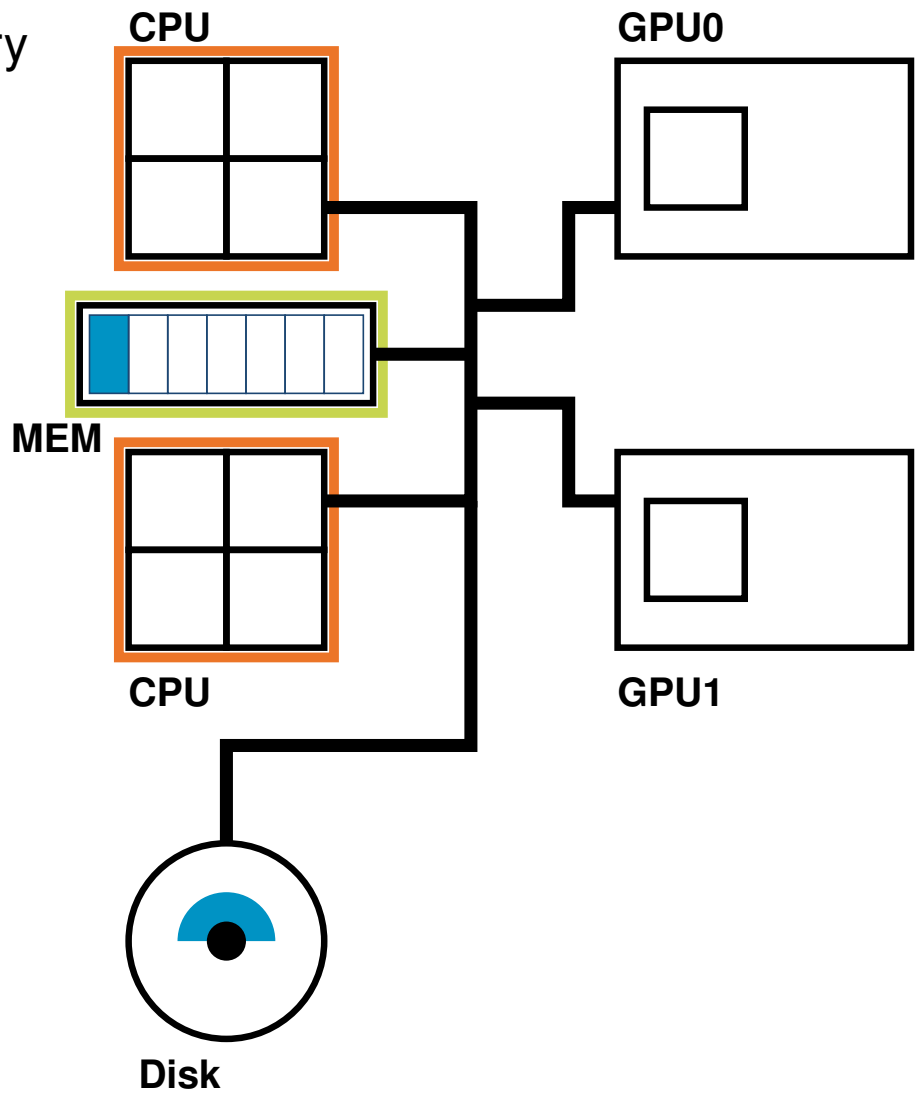
Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)



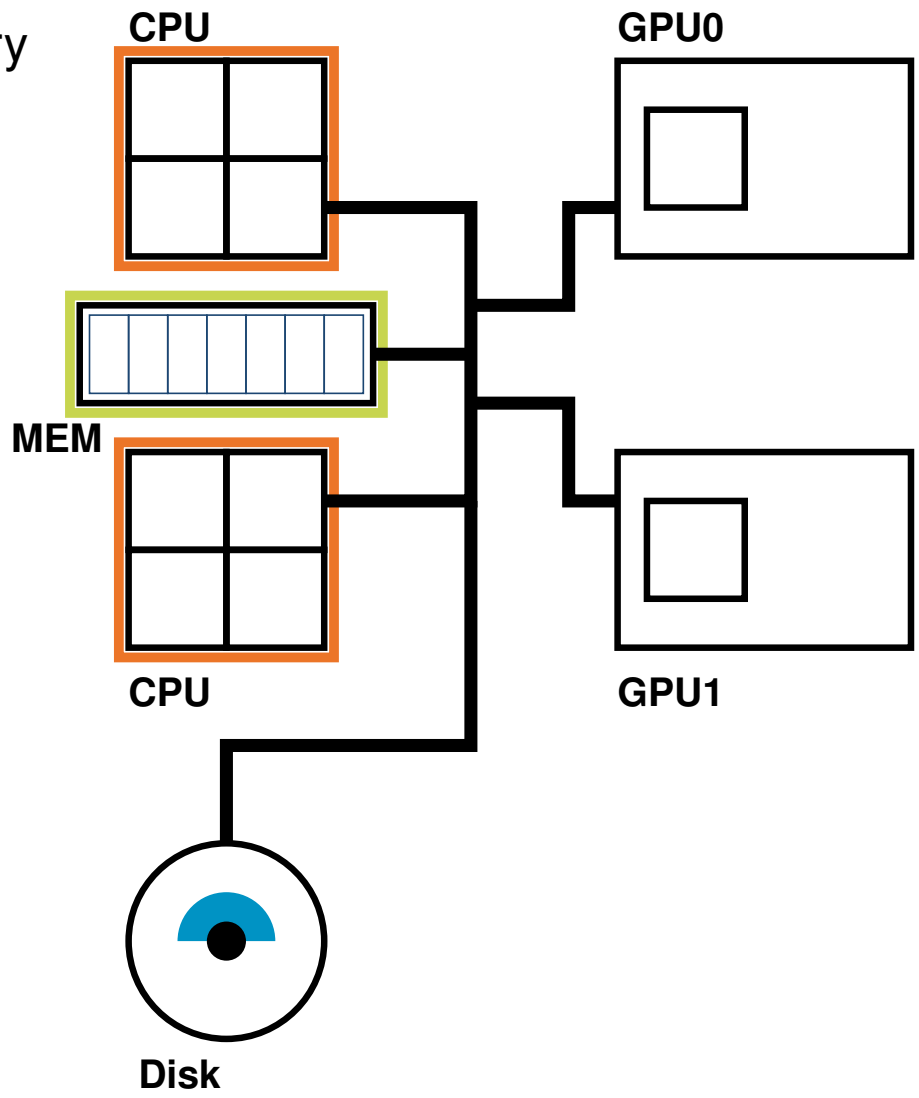
Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)



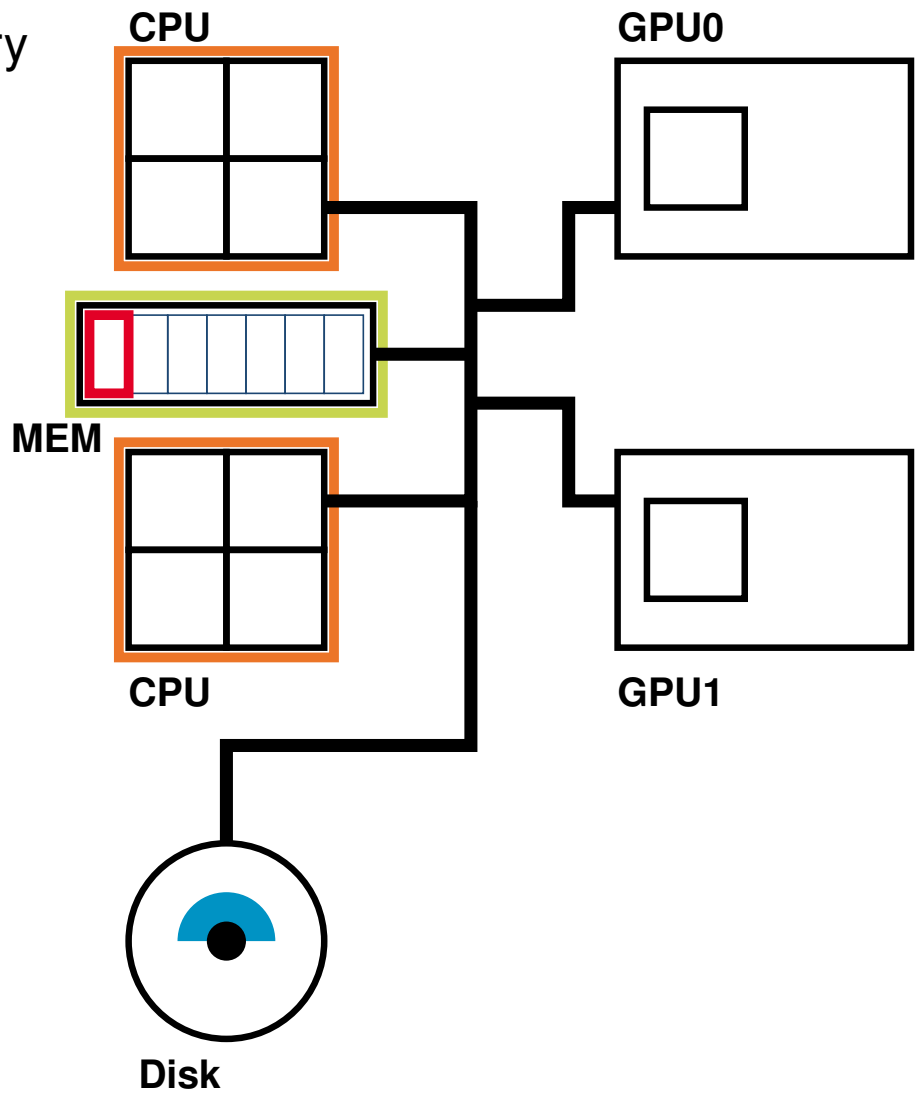
Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)



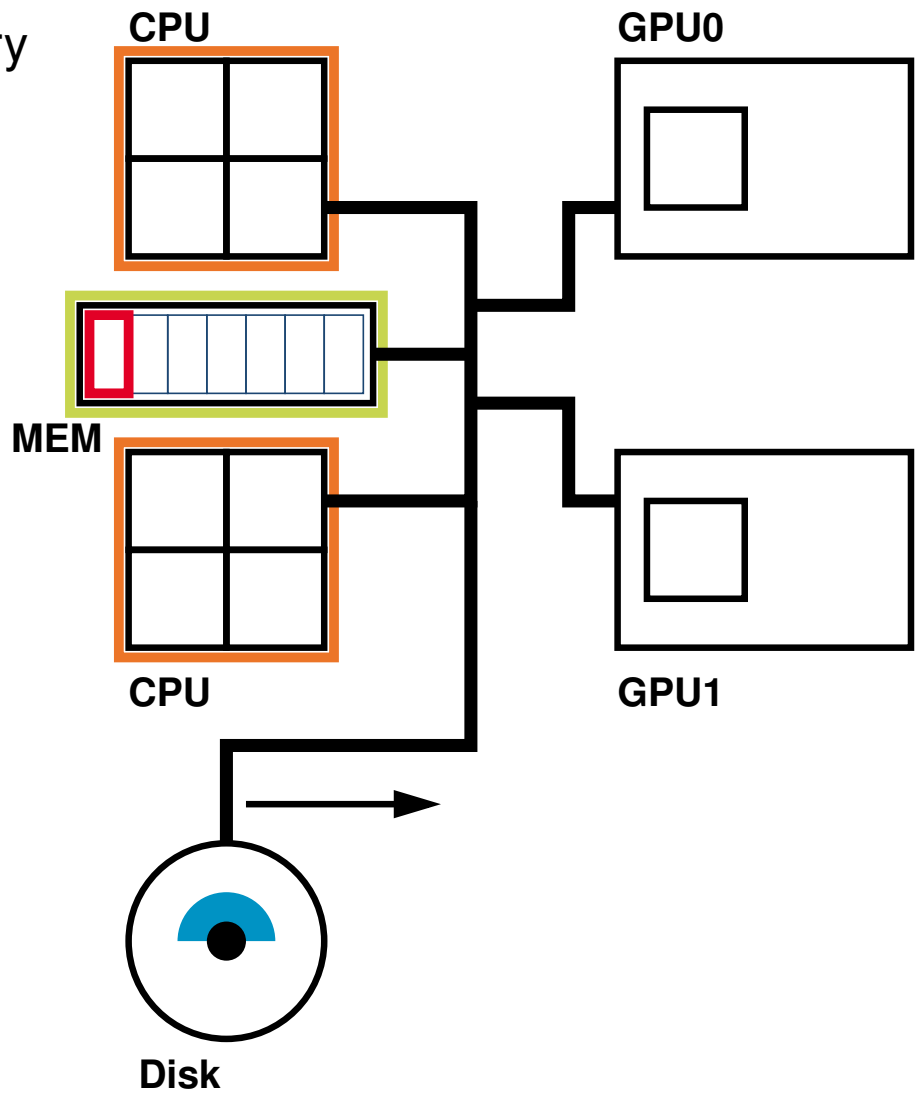
Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)



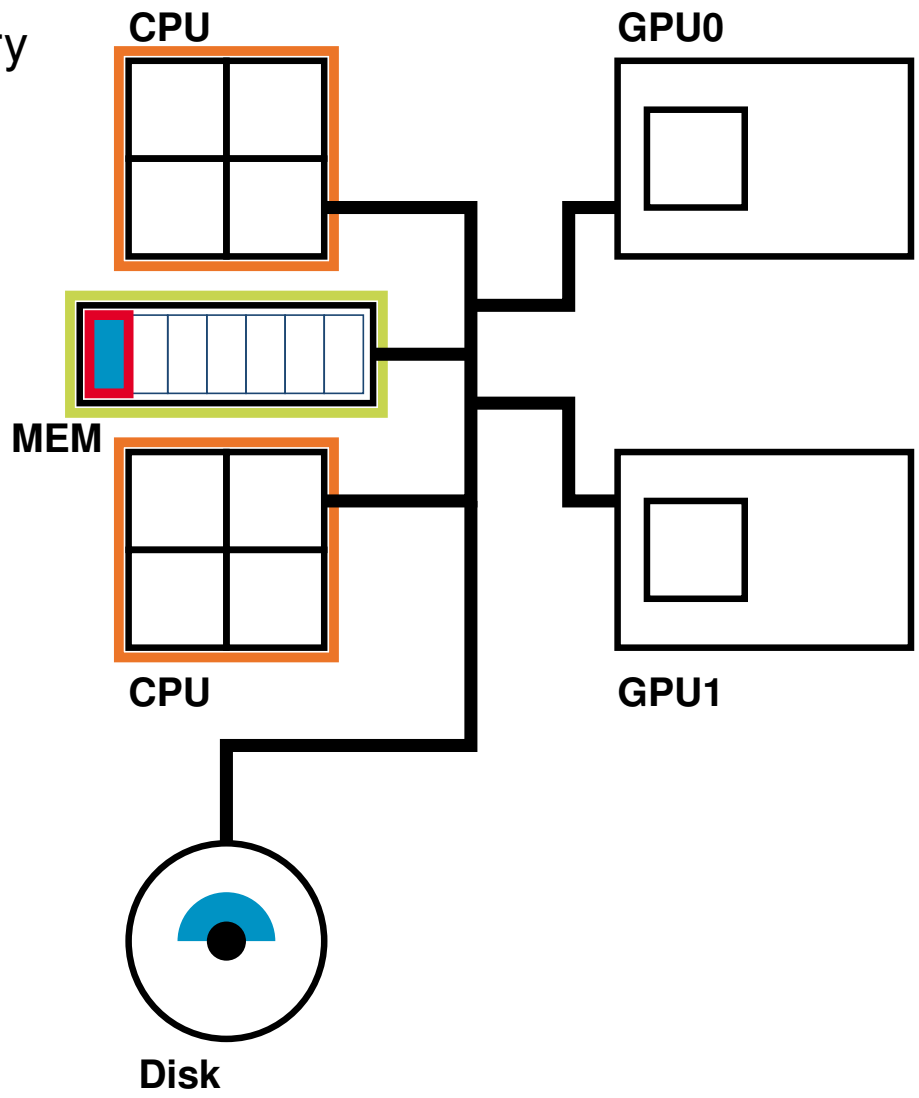
Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)



9

Advanced Debugging / Monitoring

Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs

Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs

```
int ret = starpu_init(NULL);
...

starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();


...
```

Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs

```
int ret = starpu_init(NULL);
...
starpu_bound_start();
starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();
starpu_bound_stop();

...
```

Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs

```
int ret = starpu_init(NULL);
...
starpu_bound_start();
starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();
starpu_bound_stop();
starpu_bound_print_lp();
...
```

Computing the Theoretical Lower Bound...

... on Execution Time

- Have realistic expectations from the scheduler
- Identify issues
  - Abnormal overhead
  - Bugs
- Generate a Linear Programming problem...
  - ... to be solved externally (**lp_solve**, etc.)

```
int ret = starpu_init(NULL);
...
starpu_bound_start();
starpu_task_insert(...);
starpu_task_insert(...);
...
starpu_task_wait_for_all();
starpu_bound_stop();
starpu_bound_print_lp();
...
```

Simulation with SimGrid

Scheduling without executing kernels

- Requires the SimGrid simulation environment
- Enables simulating large-scale scenarios
  - Large data sets
  - Large simulated hardware plaform
- Relies on **real** performance models...
- ...collected by StarPU on a real machine
- Enables fast experiments when designing application algorithms
- Enables fast experiments when designing scheduling algorithms

```
$ $STARPU_DIR/configure --enable-simgrid [... other opts ...]
...
```

Simulation with SimGrid

Scheduling without executing kernels

- Requires the SimGrid simulation environment
- Enables simulating large-scale scenarios
  - Large data sets
  - Large simulated hardware plaform
- Relies on **real** performance models...
- ...collected by StarPU on a real machine
- Enables fast experiments when designing application algorithms
- Enables fast experiments when designing scheduling algorithms

```
$ $STARPU_DIR/configure --enable-simgrid [... other opts ...]
...
```

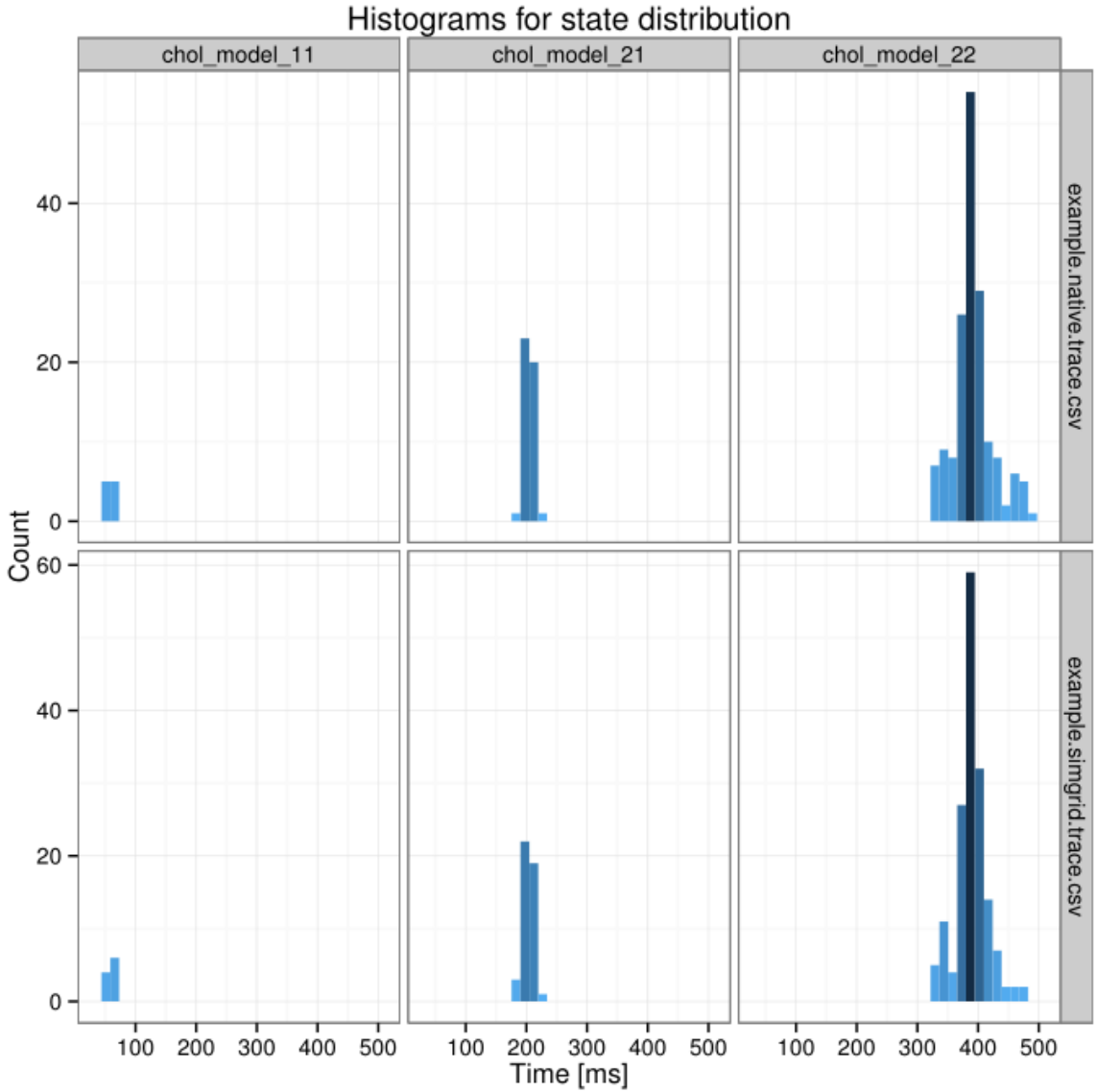
Simulation with SimGrid

Scheduling without executing kernels

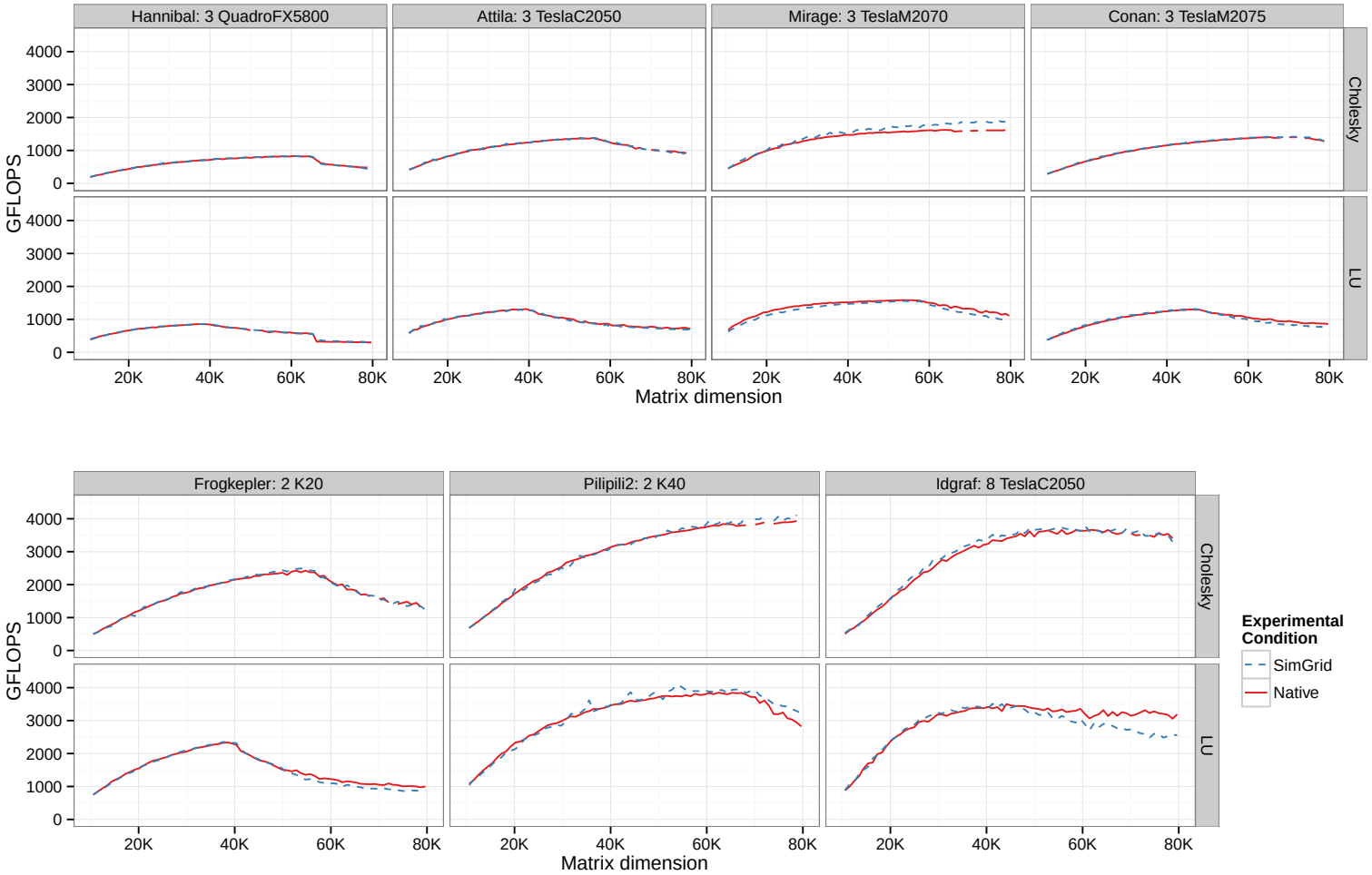
- Requires the SimGrid simulation environment
- Enables simulating large-scale scenarios
  - Large data sets
  - Large simulated hardware plaform
- Relies on **real** performance models...
- ...collected by StarPU on a real machine
- Enables fast experiments when designing application algorithms
- Enables fast experiments when designing scheduling algorithms

```
$ $STARPU_DIR/configure --enable-simgrid [... other opts ...]
...
```

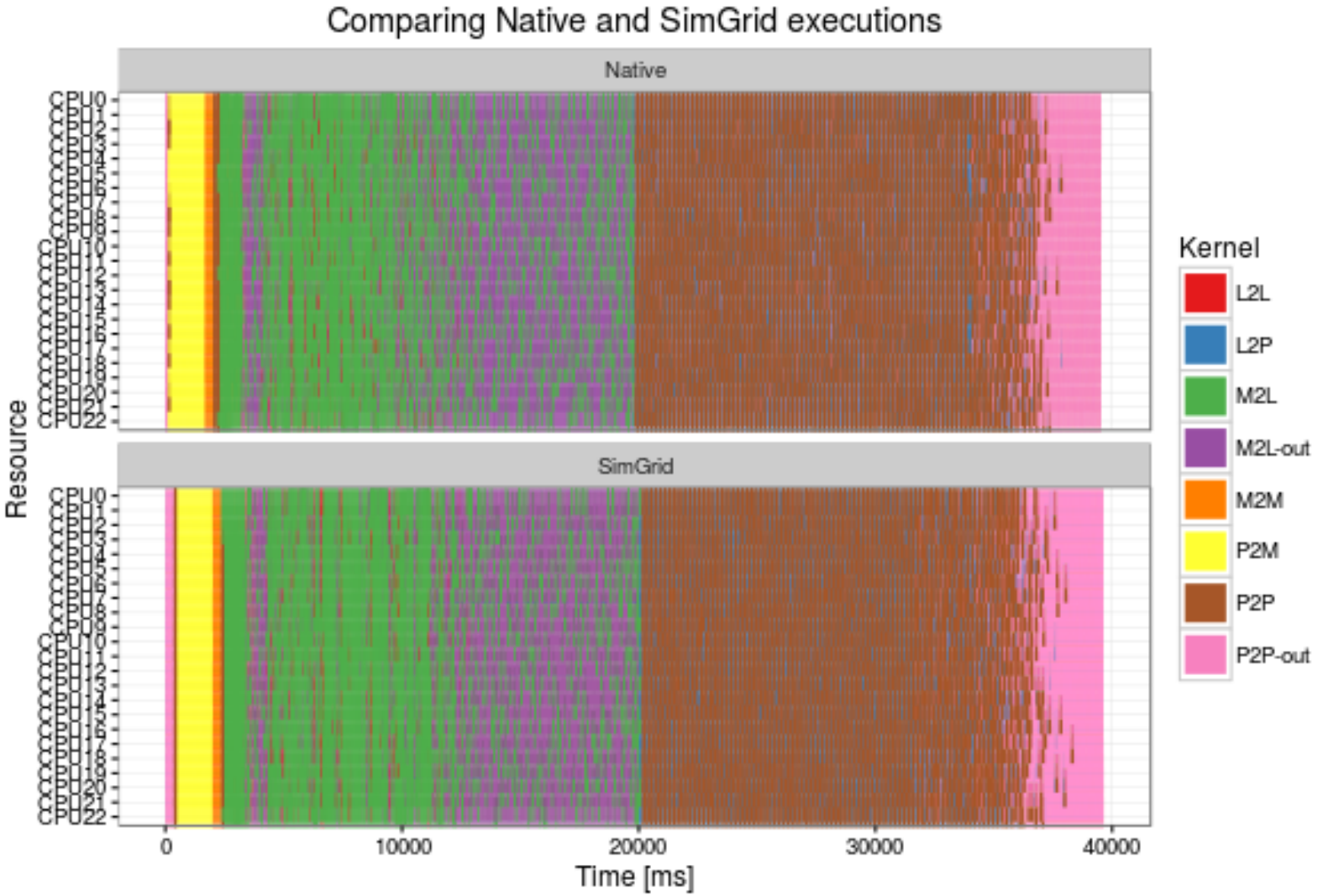
Simulation accuracy with SimGrid



Simulation with StarPU/SimGrid (L. Stanisic)



Simulation with StarPU/SimGrid (L. Stanisic)



Conclusion

StarPU

A Unified Runtime System for Heterogeneous Multicore Architectures

Conclusion

StarPU

A Unified Runtime System for Heterogeneous Multicore Architectures

Programming Model: **Async. Task Submission + Inferred Dependencies**

Conclusion

StarPU

A Unified Runtime System for Heterogeneous Multicore Architectures

Programming Model: **Async. Task Submission + Inferred Dependencies**
Execution Model: **Scheduler + Distributed Shared Memory**

Conclusion

StarPU

A Unified Runtime System for Heterogeneous Multicore Architectures

Programming Model: **Async. Task Submission + Inferred Dependencies**
Execution Model: **Scheduler + Distributed Shared Memory**

The key combination for:

- Portability
- Control
- Adaptiveness
- Optimization

Portability of Performance

Summary

1. StarPU Internals
2. Scheduling
3. Data management
4. Debugging / Monitoring
5. Distributed computing
6. High-level programming
7. Advanced scheduling
8. Advanced Data Management
9. Advanced Debugging / Monitoring

Partnerships

- Industrial Partnerships
  - Airbus Group, CEA, IMACS
- EU H2020 INTERTWinE (France, UK, Spain, Germany, Sweden)
  - Runtime system interoperability
- MORSE Associated Team: INRIA/UTK
  - Linear Algebra
- DGA RAPID Hi-BOX
  - FMM toolbox on top of StarPU
- ANR SOLHAR
  - Sparse Linear Algebra
- ANR SONGS
  - SimGrid simulation
- INRIA IPL C2S@Exa
  - Federation/integration of INRIA's HPC Software
- INRIA ADT K'Star
  - OpenMP source-to-source compiler

StarPU Session Slides and Feedback

PDF Slides at the end of the page:
http://starpu.gforge.inria.fr/tutorials/2016-06-PATC/

— — —

H2020 INTERTWinE Project: **www.intertwine.eu**

INTERTWinE Feedback Form on **StarPU** Session:
https://www.surveymonkey.co.uk/r/NJ3MGKW

Voluntarily, anonymous form. (approx. 2 min to complete)

End of Part. 2 — Mastering StarPU

StarPU

Web Site: http://starpu.gforge.inria.fr/

LGPL License

Open to external contributors