StarPU, a Task-Based Runtime System

for Heterogeneous Platform Programming (1/2)

Olivier Aumage, Team STORM
Inria – LaBRI
olivier.aumage@inria.fr

STORM
Static Optimizations – Runtime Methods

Contents

1

Programming with StarPU

Terminology

- Codelet
- Task
- Data handle

Definition: A Codelet

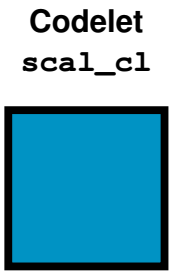
A **Codelet**...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s

Definition: A Codelet

A **Codelet**...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s



Definition: A Codelet

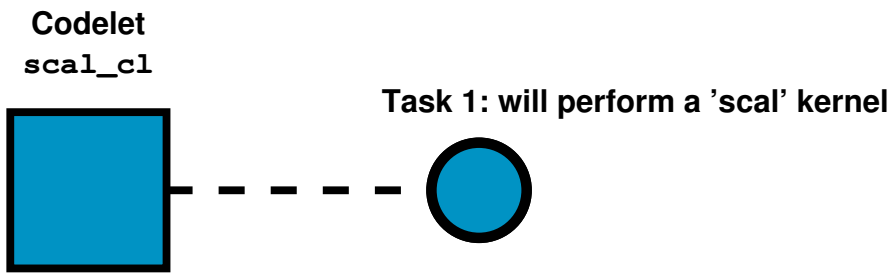
A **Codelet**...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s

Definition: A Codelet

A **Codelet**...

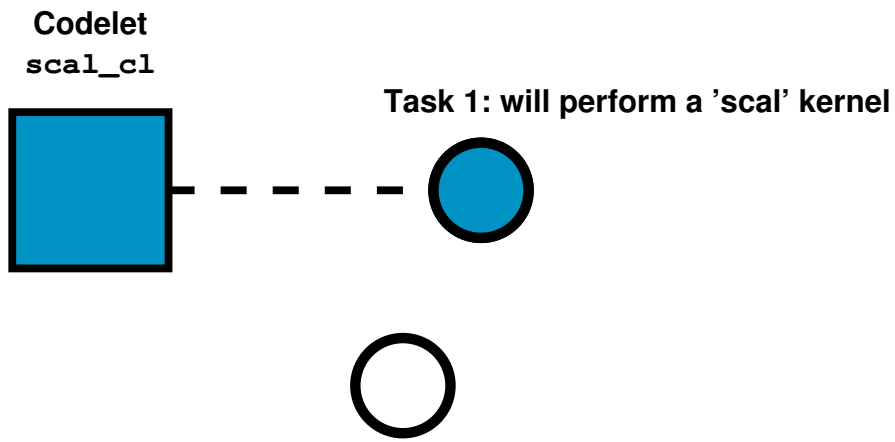
- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s



Definition: A Codelet

A **Codelet**...

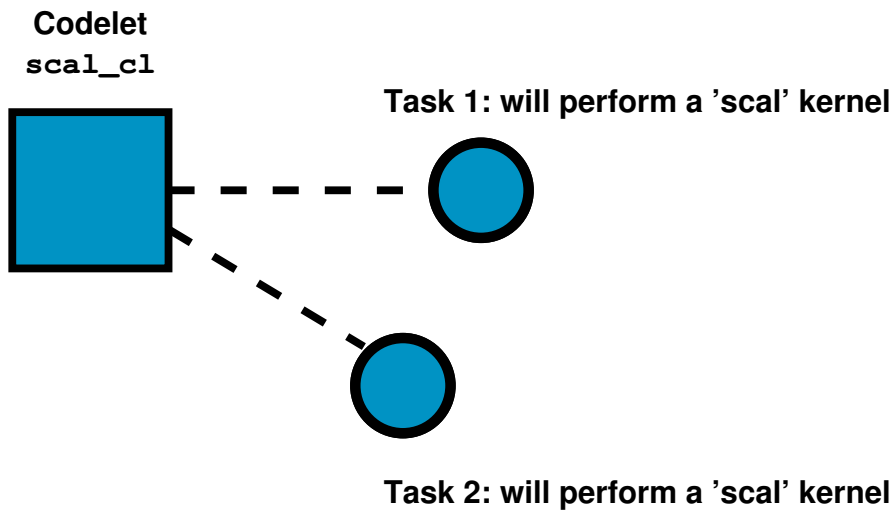
- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s



Definition: A Codelet

A **Codelet**...

- ... relates an abstract computation kernel to its implementation(s)
- ... can be instantiated into one or more **tasks**
- ... defines characteristics common to a set of **task**s



Definition: A Task

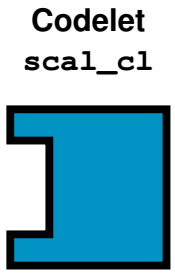
A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

Definition: A Task

A **Task**...

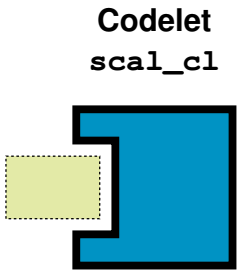
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A **Task**...

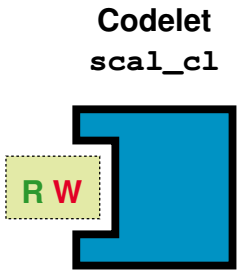
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A **Task**...

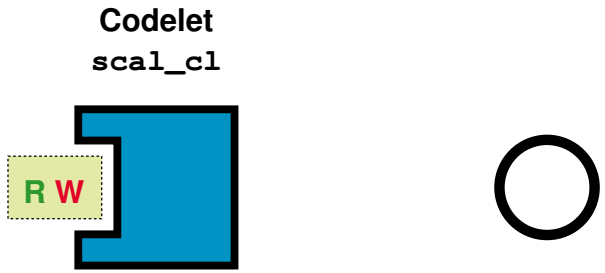
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A **Task**...

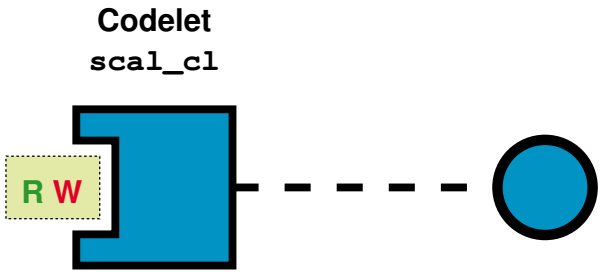
- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

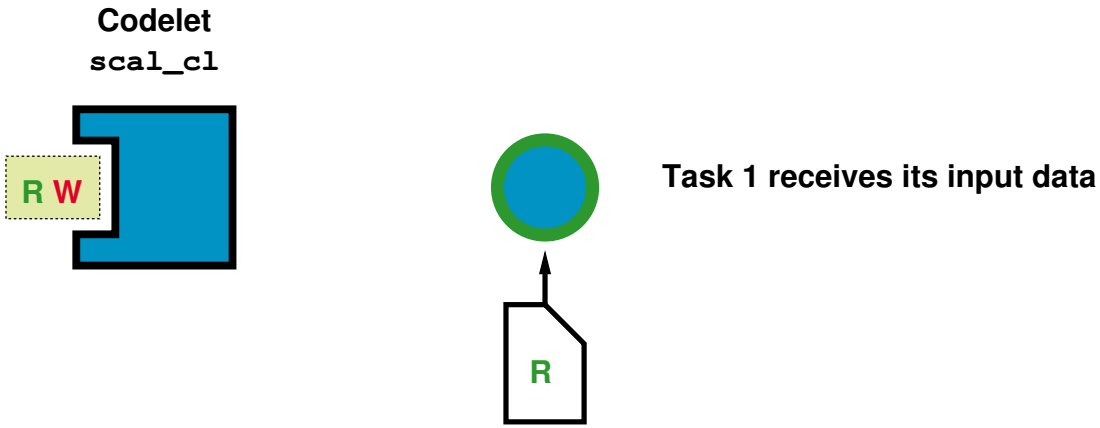
A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

Definition: A Task

A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Task

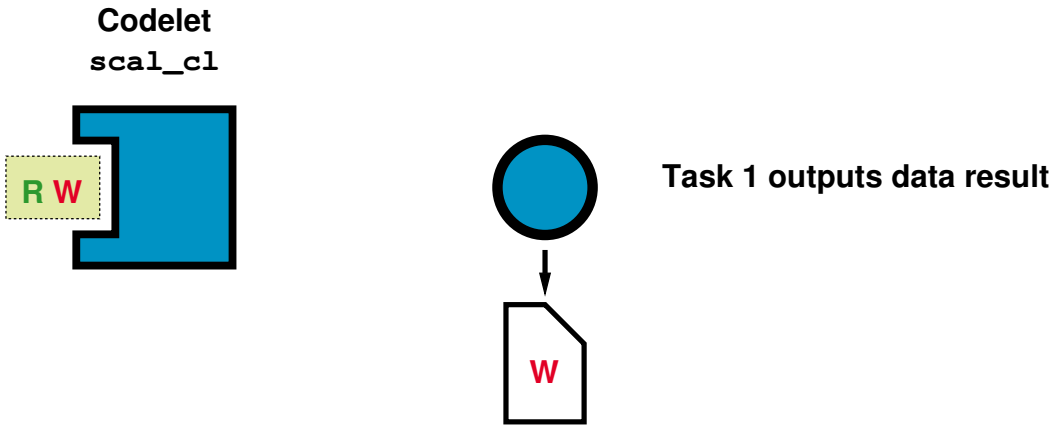
A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output

Definition: A Task

A **Task**...

- ... is an instantiation of a **Codelet**
- ... atomically executes a kernel from its beginning to its end
- ... receives some input
- ... produces some output



Definition: A Data Handle

A **Data Handle**...

- ... designates a piece of data managed by StarPU
- ... is typed (vector, matrix, etc.)
- ... can be passed as input/output for a **Task**

Elementary API

- Declaring a codelet
- Declaring and Managing Data
- Writing a Kernel Function
- Submitting a task
- Waiting for submitted tasks

Declaring a Codelet

Define a **struct** starpu_codelet

```
1 struct starpu_codelet scal_cl = {
2       ...
3 };
```

Declaring a Codelet

Define a **struct** **starpu_codelet**

- Plug the kernel function
  - Here: scal_cpu_func

```
1 struct starpu_codelet scal_cl = {
2       .cpu_func  = { scal_cpu_func, NULL },
3       ...
4 };
```

Declaring a Codelet

Define a **struct** **starpu_codelet**

- Plug the kernel function
  - Here: scal_cpu_func
- Declare the number of data pieces used by the kernel
  - Here: A single vector

```
1 struct starpu_codelet scal_cl = {
2       .cpu_func  = { scal_cpu_func, NULL },
3       .nbuffers = 1,
4       ...
5 };
```

Declaring a Codelet

Define a **struct** **starpu_codelet**

- Plug the kernel function
  - Here: scal_cpu_func
- Declare the number of data pieces used by the kernel
  - Here: A single vector
- Declare how the kernel accesses the piece of data
  - Here: The vector is scaled in-place, thus R/W

```
1 struct starpu_codelet scal_cl = {
2       .cpu_func  = { scal_cpu_func, NULL },
3       .nbuffers = 1,
4       .modes = { STARPU_RW },
5 };
```

Declaring and Managing Data

Put data under StarPU control

Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data

```
1 float vector[NX];
2 /* ... fill data ... */
```

Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control

```
1 float vector[NX];
2 /* ... fill data ... */
3
4 starpu_data_handle_t vector_handle;
5 starpu_vector_data_register(&vector_handle, 0,
6                             (uintptr_t)vector, NX, sizeof(vector[0]));
```

Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control
- Use data through the handle

```
1 float vector[NX];
2 /* ... fill data ... */
3
4 starpu_data_handle_t vector_handle;
5 starpu_vector_data_register(&vector_handle, 0,
6                     (uintptr_t)vector, NX, sizeof(vector[0]));
7
8 /* ... use the vector through the handle ... */
```

Declaring and Managing Data

Put data under StarPU control

- Initialize a piece of data
- Register the piece of data and get a handle
  - The vector is now under StarPU control
- Use data through the handle
- Unregister the piece of data
  - The handle is destroyed
  - The vector is now back under user control

```
1  float vector[NX];
2  /* ... fill data ... */
3
4  starpu_data_handle_t vector_handle;
5  starpu_vector_data_register(&vector_handle, 0,
6                              (uintptr_t)vector, NX, sizeof(vector[0]));
7
8  /* ... use the vector through the handle ... */
9
10 starpu_data_unregister(vector_handle);
```

Writing a Kernel Function

- Every kernel function has the same C prototype

```
1 void scal_cpu_func(void *buffers[], void *cl_arg) {
2     ...
3 }
```

Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector's handle

```
1 void scal_cpu_func(void *buffers[], void *cl_arg) {
2     struct starpu_vector_interface *vector_handle = buffers
          [0];
3
4     ...
5 }
```

Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector's handle
- Get vector's number of elements and base pointer

```
1 void scal_cpu_func(void *buffers[], void *cl_arg) {
2     struct starpu_vector_interface *vector_handle = buffers
          [0];
3
4     unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
5     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
6
7     ...
8 }
```

Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector's handle
- Get vector's number of elements and base pointer
- Get the scaling factor as inline argument

```
1  void scal_cpu_func(void *buffers[], void *cl_arg) {
2      struct starpu_vector_interface *vector_handle = buffers
           [0];
3
4      unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
5      float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
6
7      float factor;
8      starpu_codelet_unpack_args(cl_arg, &factor)
9
10     ...
11 }
```

Writing a Kernel Function

- Every kernel function has the same C prototype
- Retrieve the vector's handle
- Get vector's number of elements and base pointer
- Get the scaling factor as inline argument
- Compute the vector scaling

```
1  void scal_cpu_func(void *buffers[], void *cl_arg) {
2      struct starpu_vector_interface *vector_handle = buffers
           [0];
3
4      unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
5      float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
6
7      float factor;
8      starpu_codelet_unpack_args(cl_arg, &factor)
9
10     unsigned i;
11     for (i = 0; i < n; i++)
12         vector[i] *= factor;
13 }
```

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure

```
1 starpu_task_insert(&scal_cl
2                    ...);
```

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data

```
1 starpu_task_insert(&scal_cl,
2                     STARPU_RW, vector_handle,
3                     ...);
```

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data

```
1 starpu_task_insert(&scal_cl,
2                     STARPU_RW, vector_handle,
3                     STARPU_VALUE, &factor, sizeof(factor),
4                     ...);
```

Submitting a task

The starpu_task_insert call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

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

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks' data...

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks' data...
- ... following the natural order of the program

Submitting a task

The **starpu_task_insert** call

- **Inserts** a task in the StarPU DAG

Arguments

- The codelet structure
- The StarPU-managed data
- The small-size inline data
- 0 to mark the end of arguments

Notes

- The task is submitted non-blockingly
- Dependencies are enforced with previously submitted tasks' data...
- ... following the natural order of the program
- This is the **Sequential Task Flow Paradigm**

Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly

Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly

```
1 /* non-blocking task submits */
2 starpu_task_insert(...);
3 ...
```

Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly
- Wait for all submitted tasks to complete their work

```
1 /* non-blocking task submits */
2 starpu_task_insert(...);
3 ...
```

Waiting for Submitted Task Completion

- Tasks are submitted non-blockingly
- Wait for all submitted tasks to complete their work

```
1 /* non-blocking task submits */
2 starpu_task_insert(...);
3 ...
4
5 /* wait for all task submitted so far */
6 starpu_task_wait_for_all();
```

Basic Example: Scaling a Vector (main prog.)

```
1 float factor = 3.14;
2 float vector[NX];
```

Basic Example: Scaling a Vector (main prog.)

```
1 float factor = 3.14;
2 float vector[NX];
3 starpu_data_handle_t vector_handle;
```

Basic Example: Scaling a Vector (main prog.)

```
1  float factor = 3.14;
2  float vector[NX];
3  starpu_data_handle_t vector_handle;
4
5  /* ... fill vector ... */
6
7  starpu_vector_data_register(&vector_handle, 0,
8                              (uintptr_t)vector, NX, sizeof(vector[0]))
                                   ;
```

Basic Example: Scaling a Vector (main prog.)

```
1  float factor = 3.14;
2  float vector[NX];
3  starpu_data_handle_t vector_handle;
4
5  /* ... fill vector ... */
6
7  starpu_vector_data_register(&vector_handle, 0,
8                              (uintptr_t)vector, NX, sizeof(vector[0]))
                                   ;
9
10 starpu_task_insert(
11                    &scal_cl,
12                    STARPU_RW, vector_handle,
13                    STARPU_VALUE, &factor, sizeof(factor),
14                    0);
```

Basic Example: Scaling a Vector (main prog.)

```
1  float factor = 3.14;
2  float vector[NX];
3  starpu_data_handle_t vector_handle;
4
5  /* ... fill vector ... */
6
7  starpu_vector_data_register(&vector_handle, 0,
8                              (uintptr_t)vector, NX, sizeof(vector[0]))
                                   ;
9
10 starpu_task_insert(
11                   &scal_cl,
12                   STARPU_RW, vector_handle,
13                   STARPU_VALUE, &factor, sizeof(factor),
14                   0);
15
16 starpu_task_wait_for_all();
```

Basic Example: Scaling a Vector (main prog.)

```
1  float factor = 3.14;
2  float vector[NX];
3  starpu_data_handle_t vector_handle;
4
5  /* ... fill vector ... */
6
7  starpu_vector_data_register(&vector_handle, 0,
8                              (uintptr_t)vector, NX, sizeof(vector[0]))
                                   ;
9
10 starpu_task_insert(
11                    &scal_cl,
12                    STARPU_RW, vector_handle,
13                    STARPU_VALUE, &factor, sizeof(factor),
14                    0);
15
16 starpu_task_wait_for_all();
17 starpu_data_unregister(vector_handle);
18
19 /* ... display vector ... */
```

Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms

Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms

- Multiple kernel implementations for a CPU
  - SSE, AVX, ... optimized kernels

```
1 struct starpu_codelet scal_cl = {
2       .cpu_func  = { scal_cpu_func,
3                      scal_sse_cpu_func, scal_avx_cpu_func, NULL },
4       .nbuffers = 1,
5       .modes = { STARPU_RW },
6 };
```

Heterogeneity: Device Kernels

Extending a codelet to handle heterogeneous platforms

- Multiple kernel implementations for a CPU
  - SSE, AVX, ... optimized kernels
- Kernels implementations for accelerator devices
  - OpenCL, NVidia Cuda kernels

```
1 struct starpu_codelet scal_cl = {
2       .cpu_func  = { scal_cpu_func,
3                      scal_sse_cpu_func, scal_avx_cpu_func, NULL },
4       .opencl_func  = { scal_cpu_opencl, NULL },
5       .cuda_func  = { scal_cpu_cuda, NULL },
6       .nbuffers = 1,
7       .modes = { STARPU_RW },
8 };
```

Writing a Kernel Function for CUDA

Writing a Kernel Function for CUDA

```
1
2
3
4
5
6
7
8  extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
       {
9      struct starpu_vector_interface *vector_handle = buffers
           [0];
10     unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
11     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
12     float factor;
13     starpu_codelet_unpack_args(cl_arg, &factor)
14
15     ...
16
17
18
19
20 }
```

Writing a Kernel Function for CUDA

```
1
2
3
4
5
6
7
8  extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
       {
9      struct starpu_vector_interface *vector_handle = buffers
           [0];
10     unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
11     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
12     float factor;
13     starpu_codelet_unpack_args(cl_arg, &factor)
14
15     unsigned threads_per_block = 64;
16     unsigned nblocks = (n+threads_per_block-1)/
           threads_per_block;
17
18     ...
19
20 }
```

Writing a Kernel Function for CUDA

```
1
2
3
4
5
6
7
8  extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
       {
9      struct starpu_vector_interface *vector_handle = buffers
           [0];
10     unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
11     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
12     float factor;
13     starpu_codelet_unpack_args(cl_arg, &factor)
14
15     unsigned threads_per_block = 64;
16     unsigned nblocks = (n+threads_per_block-1)/
           threads_per_block;
17
18     vector_mult_cuda<<<nblocks,threads_per_block,0,
19         starpu_cuda_get_local_stream()>>>(n,vector,factor);
20 }
```

Writing a Kernel Function for CUDA

```
1  static __global__ void vector_mult_cuda(unsigned n,
2                                          float *vector, float factor
                                               ){
3      unsigned i = blockIdx.x*blockDim.x + threadIdx.x;
4
5      ...
6  }
7
8  extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
       {
9      struct starpu_vector_interface *vector_handle = buffers
           [0];
10     unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
11     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
12     float factor;
13     starpu_codelet_unpack_args(cl_arg, &factor)
14
15     unsigned threads_per_block = 64;
16     unsigned nblocks = (n+threads_per_block-1)/
           threads_per_block;
17
18     vector_mult_cuda<<<nblocks,threads_per_block,0,
19         starpu_cuda_get_local_stream()>>>(n,vector,factor);
20 }
```

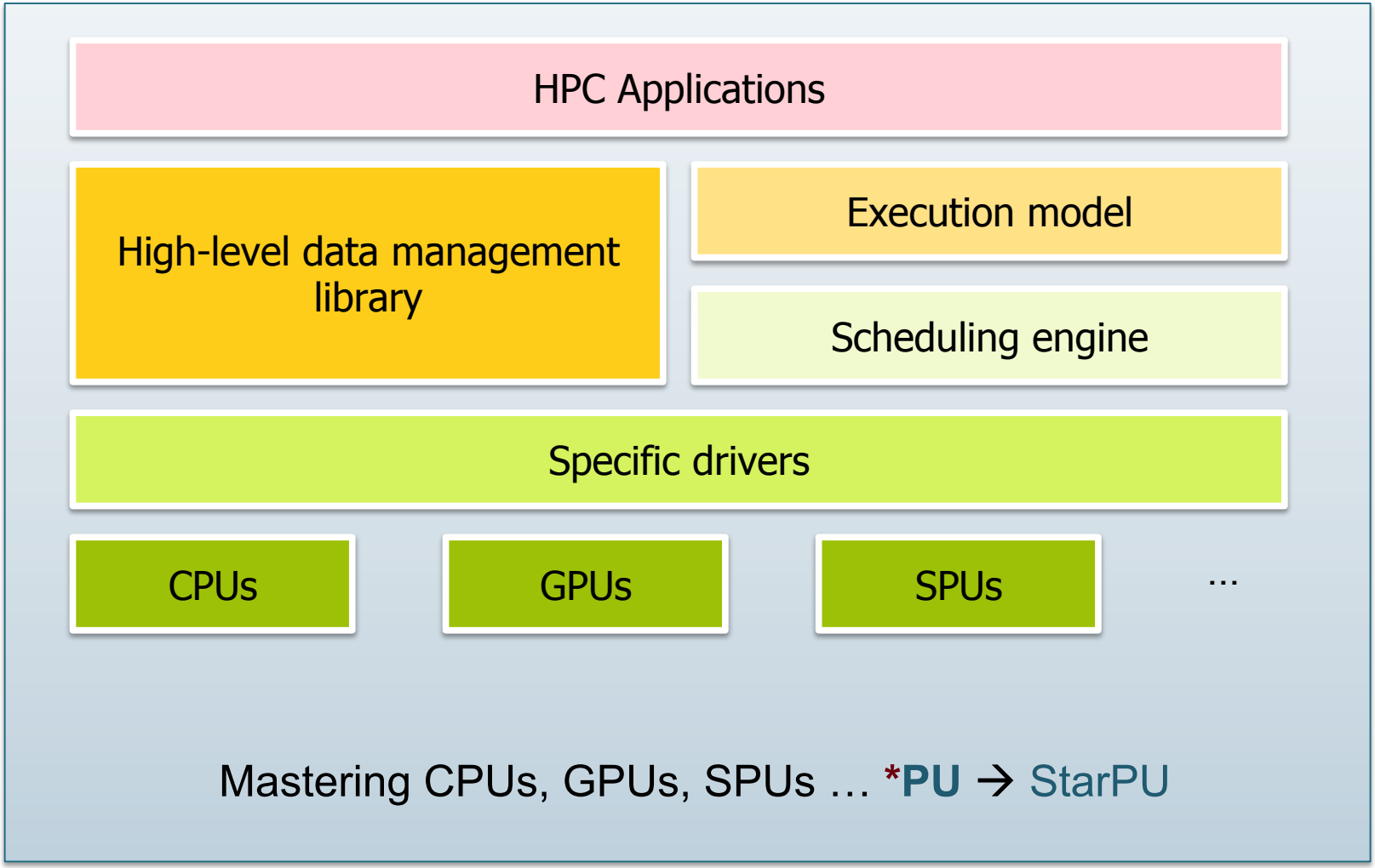
Writing a Kernel Function for CUDA

```
1  static __global__ void vector_mult_cuda(unsigned n,
2                                          float *vector, float factor
                                               ){
3      unsigned i = blockIdx.x*blockDim.x + threadIdx.x;
4      if (i < n)
5          vector[i] *= factor;
6  }
7
8  extern "C" void scal_cuda_func(void *buffers[], void *cl_arg)
       {
9      struct starpu_vector_interface *vector_handle = buffers
           [0];
10     unsigned n    = STARPU_VECTOR_GET_NX(vector_handle);
11     float *vector = STARPU_VECTOR_GET_PTR(vector_handle);
12     float factor;
13     starpu_codelet_unpack_args(cl_arg, &factor)
14
15     unsigned threads_per_block = 64;
16     unsigned nblocks = (n+threads_per_block-1)/
           threads_per_block;
17
18     vector_mult_cuda<<<nblocks,threads_per_block,0,
19         starpu_cuda_get_local_stream()>>>(n,vector,factor);
20 }
```

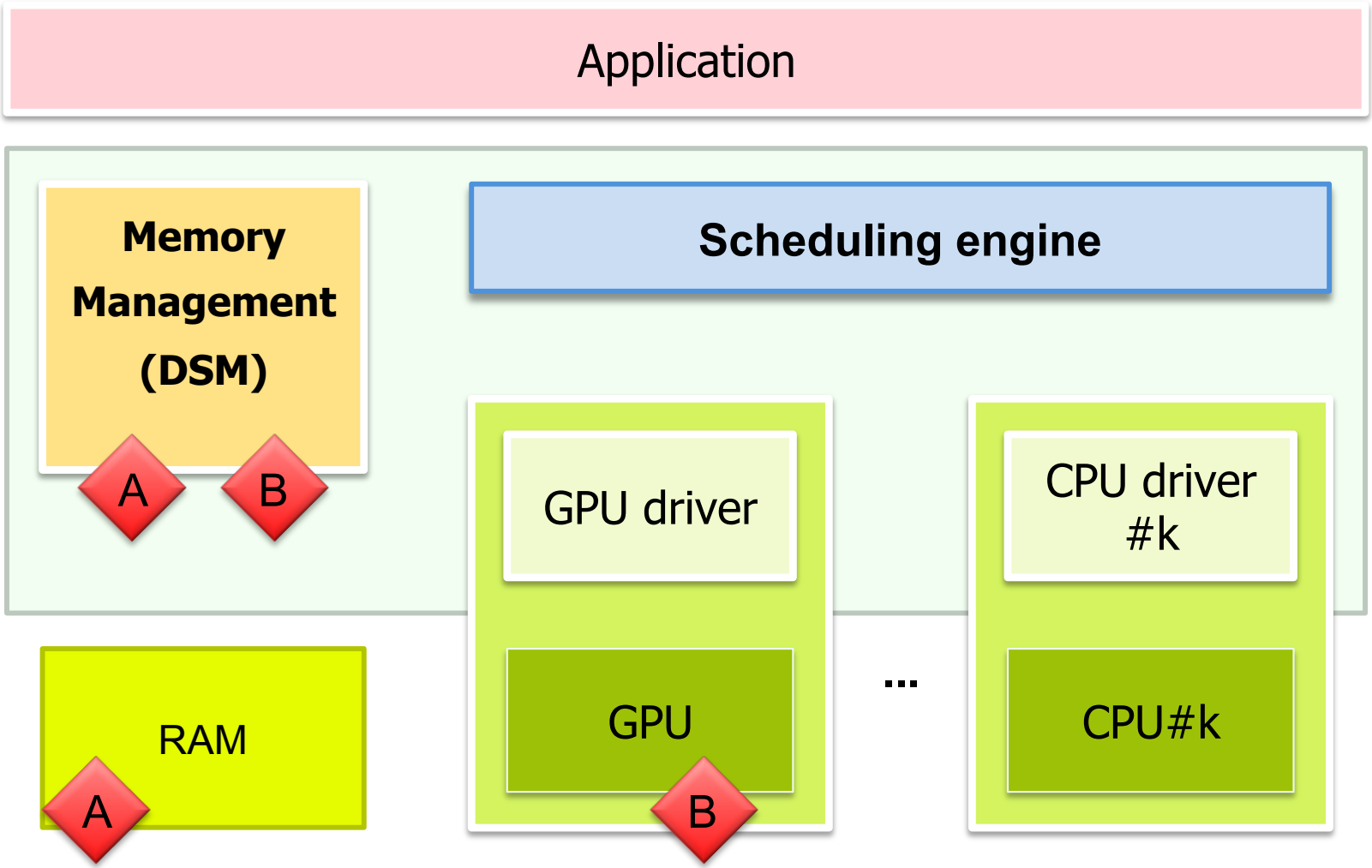
2

StarPU Internals

StarPU Internal Structure

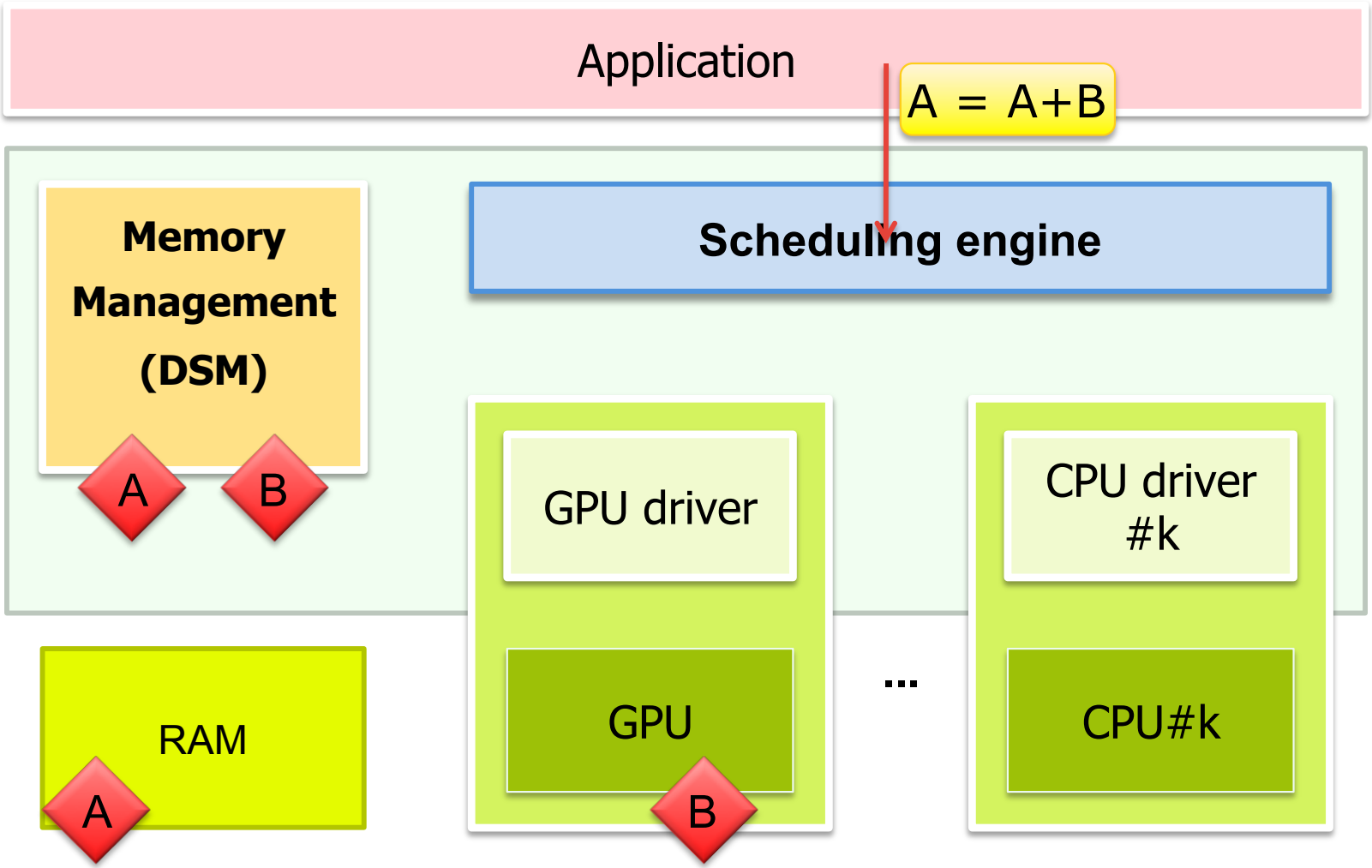


StarPU Internal Functioning



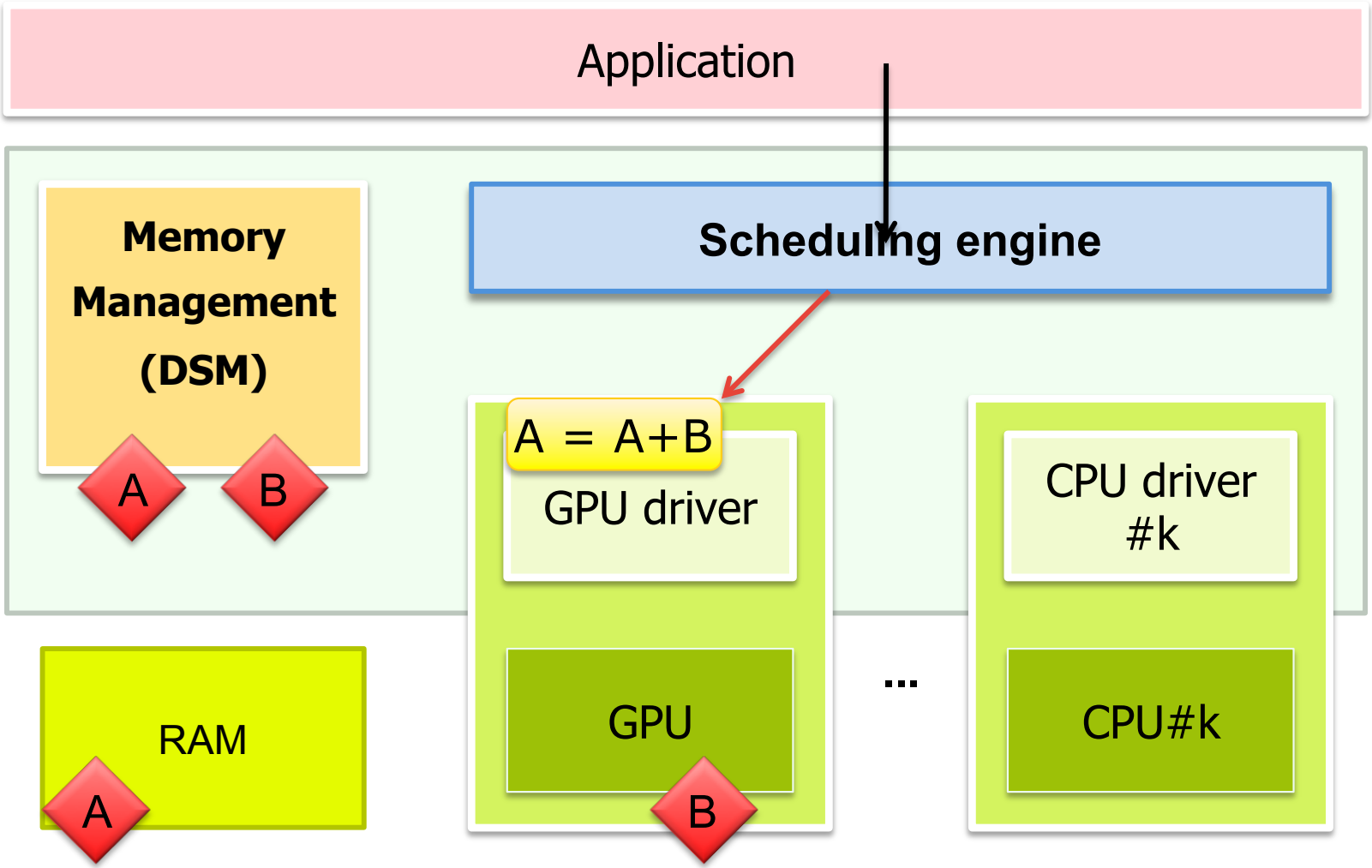
Submit task « A+=B »

StarPU Internal Functioning



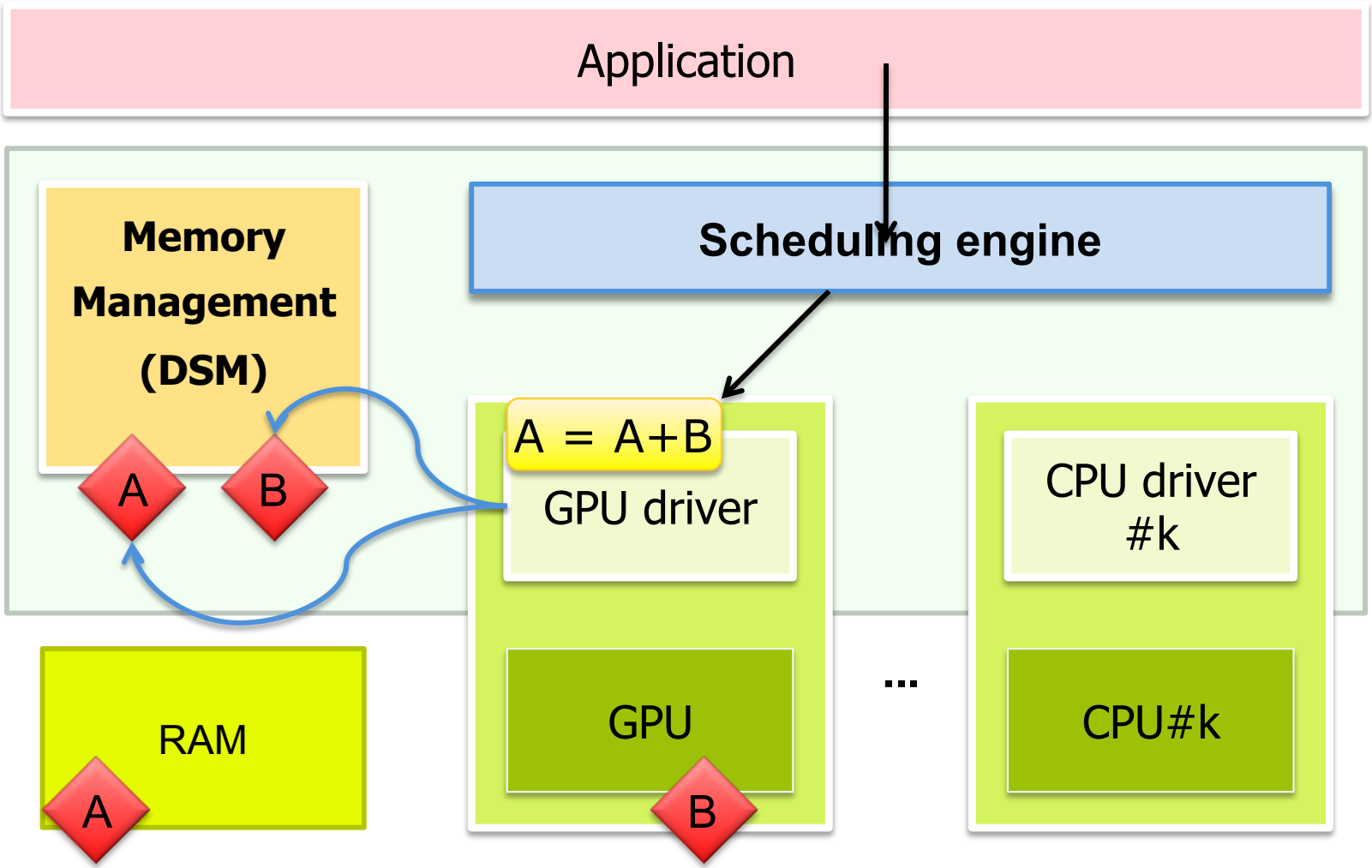
Submit task « A+=B »

StarPU Internal Functioning

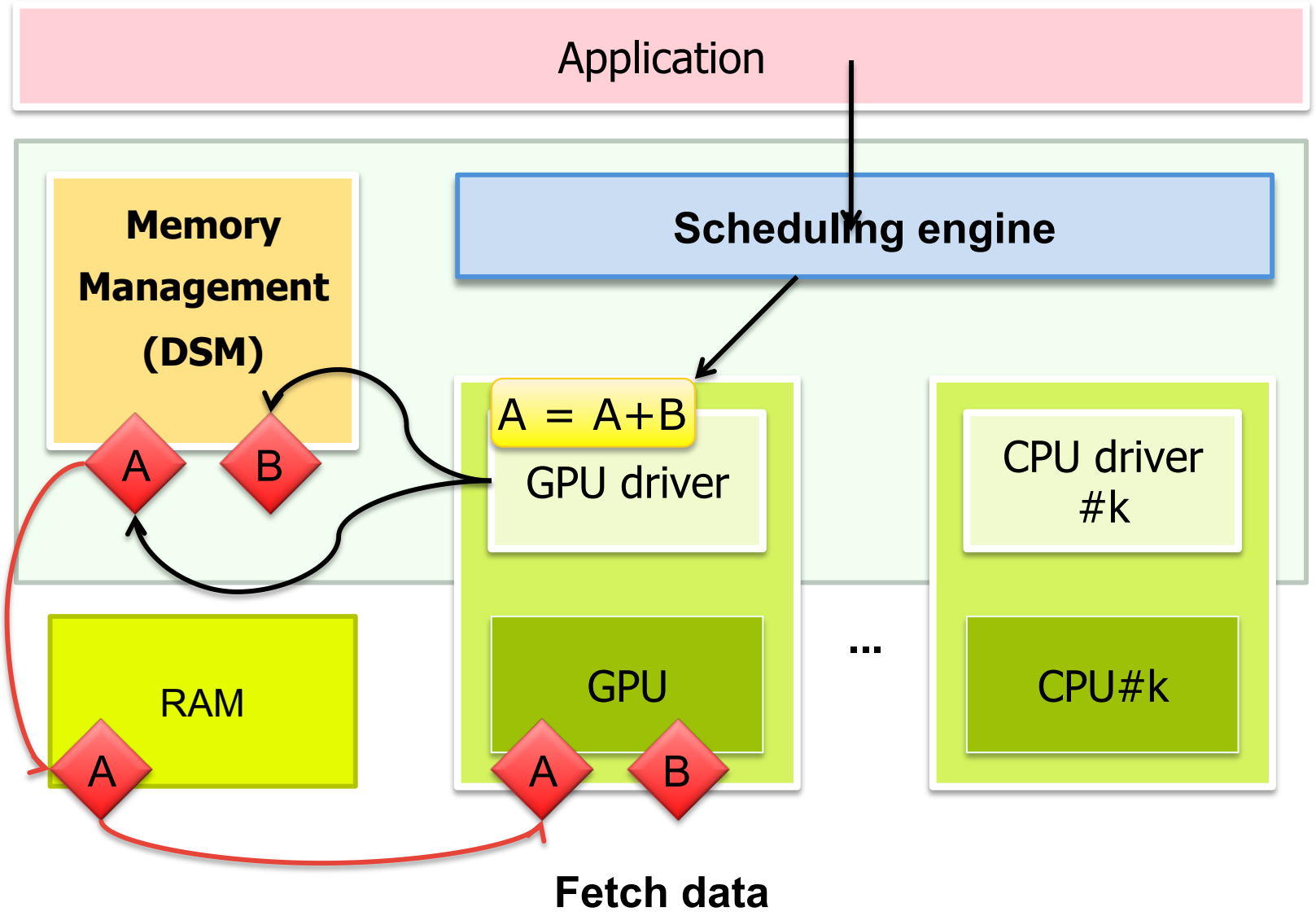


Schedule task

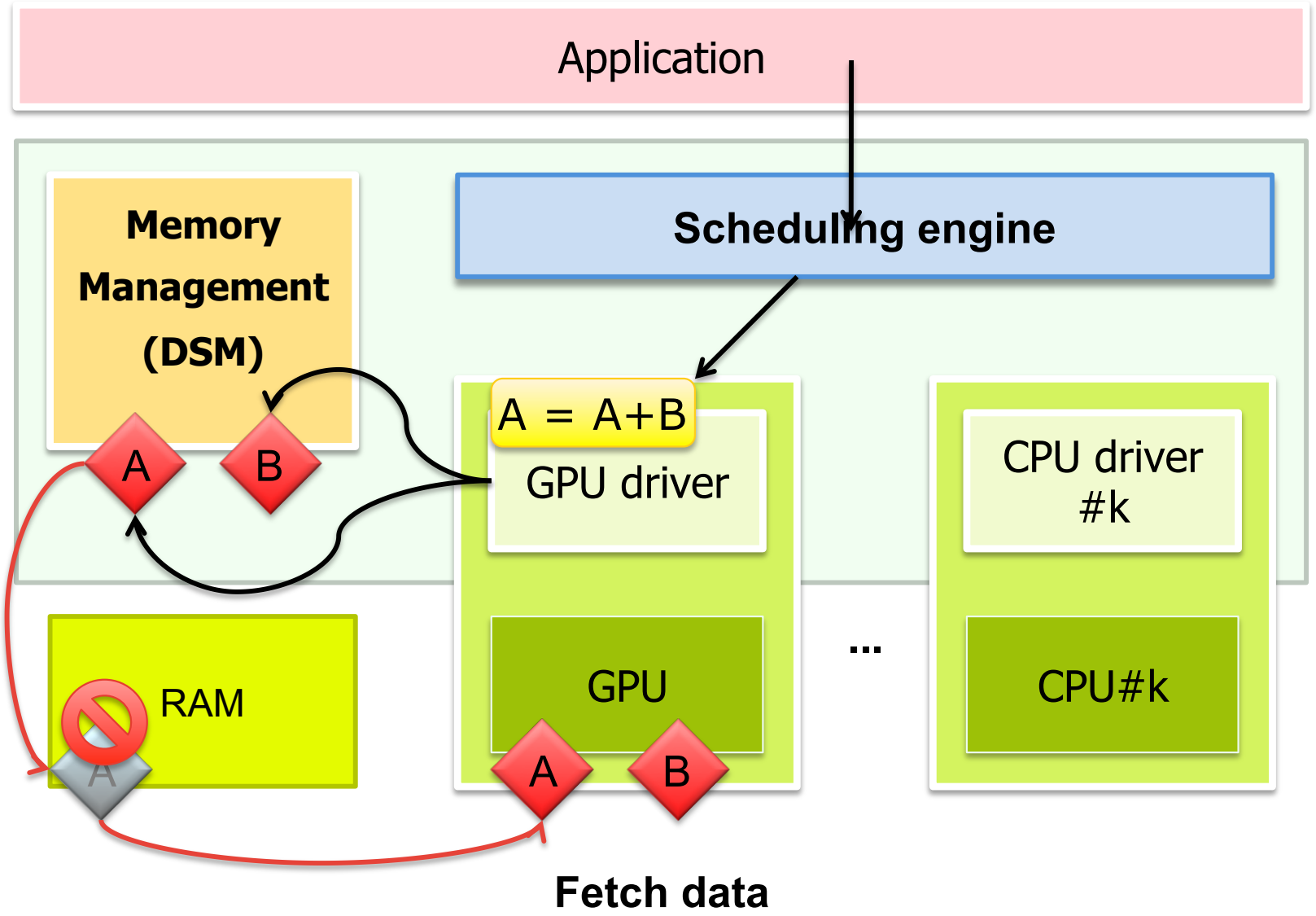
StarPU Internal Functioning



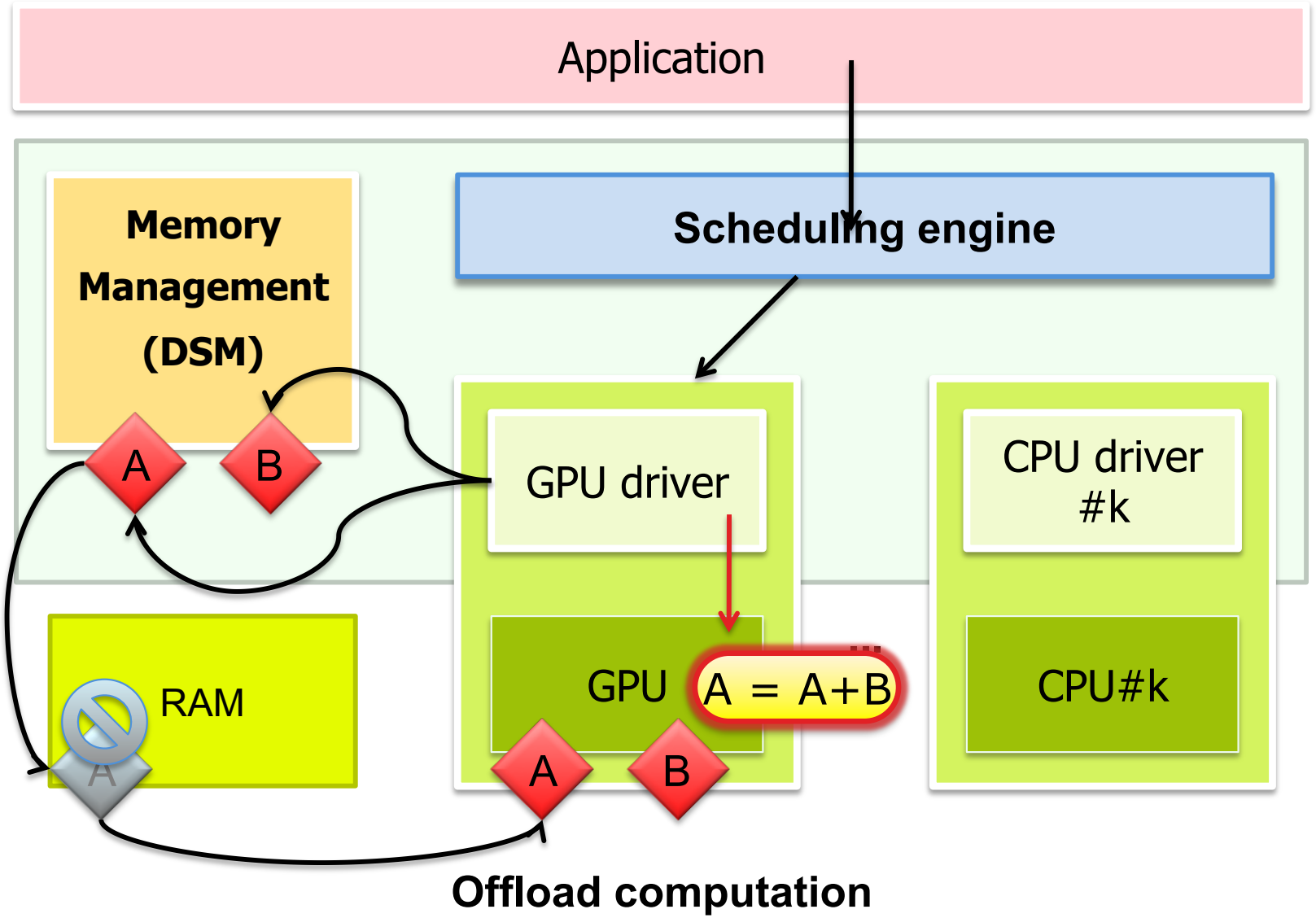
StarPU Internal Functioning



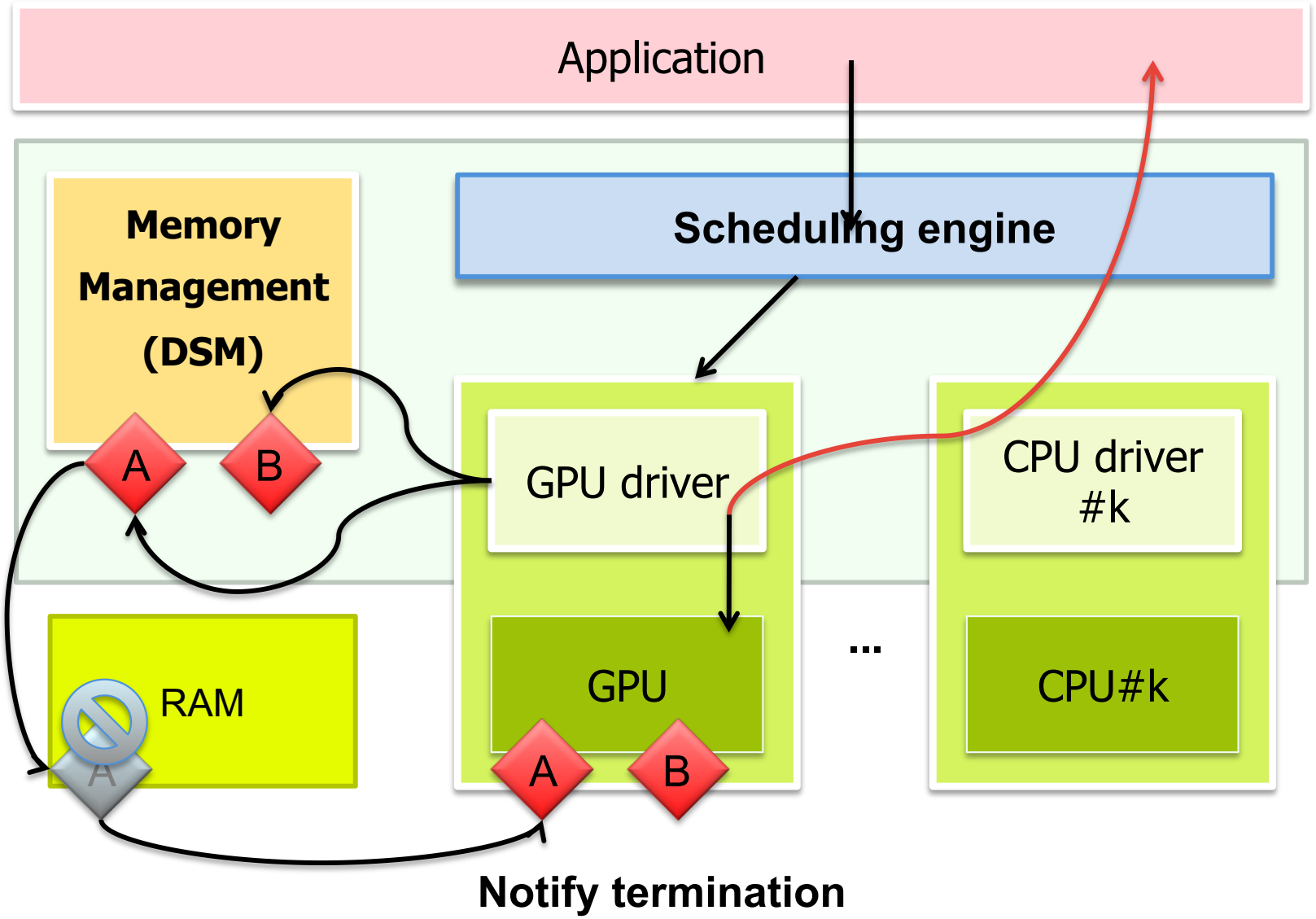
StarPU Internal Functioning



StarPU Internal Functioning



StarPU Internal Functioning



3

Scheduling Policies

StarPU Scheduling Policies

- No *one size fits all* policy
- Selectable scheduling policy
  - Predefined set of popular policies: eager, work-stealing, etc.

StarPU Scheduling Policies

- No *one size fits all* policy
- Selectable scheduling policy
  - Predefined set of popular policies: eager, work-stealing, etc.

Going beyond?

StarPU Scheduling Policies

- No *one size fits all* policy
- Selectable scheduling policy
  - Predefined set of popular policies: eager, work-stealing, etc.

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

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

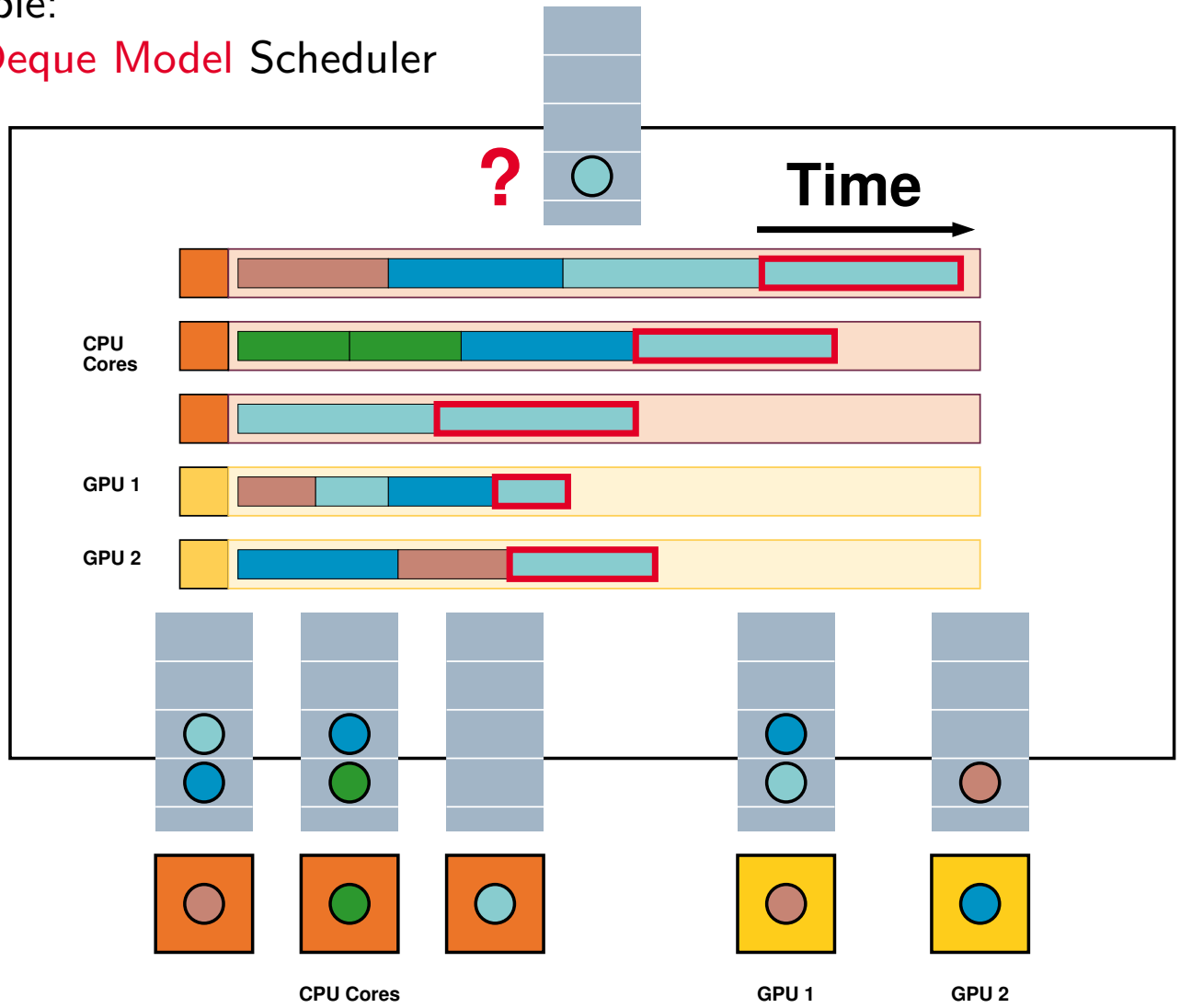
```
$ export STARPU_SCHED=prio
$ my_program
...
```

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

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

Task Mapping using a Performance Model

- Example:
  The Deque Model Scheduler



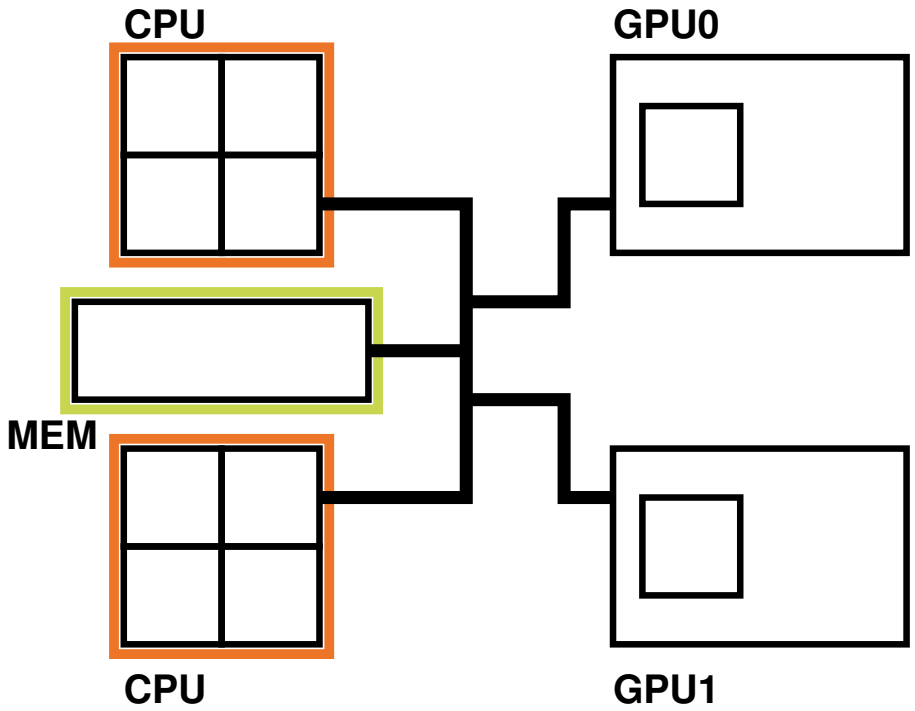
Task Mapping using a Performance Model

- 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(s)
  - **Data size**
  - **User-defined** parameters

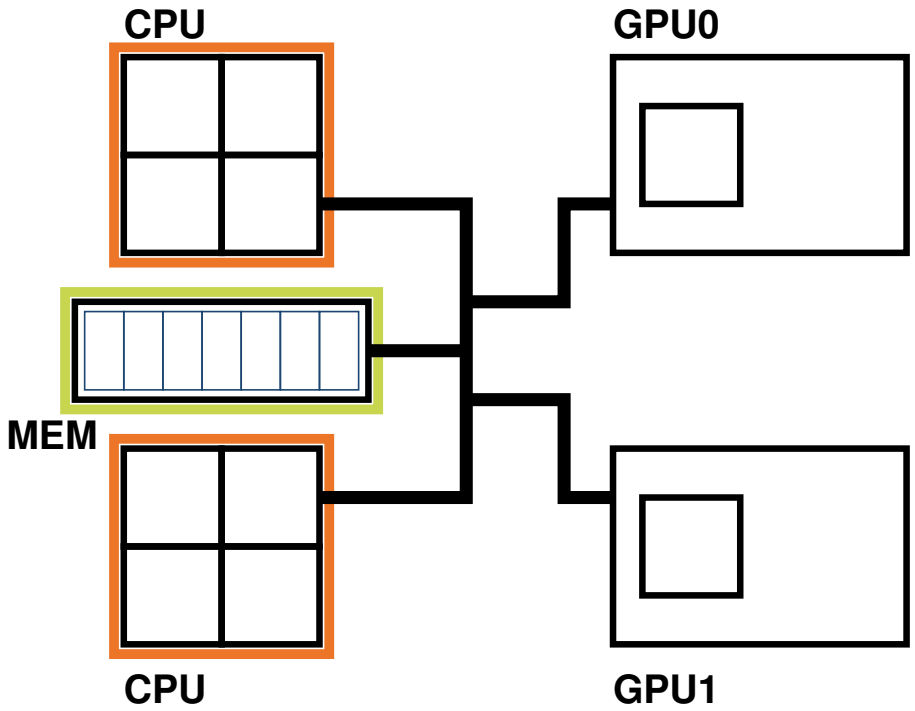
4

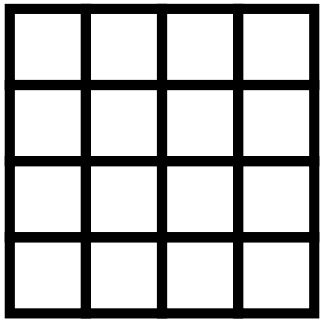
Data Management

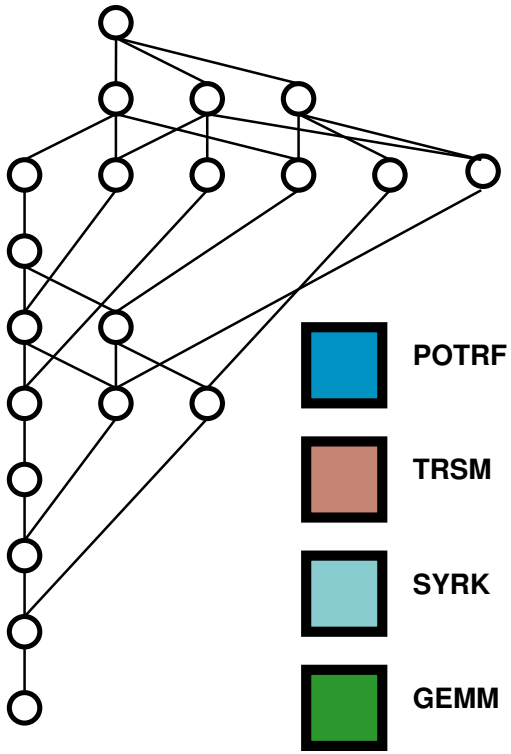
StarPU Heterogeneous Execution Model / Data Management



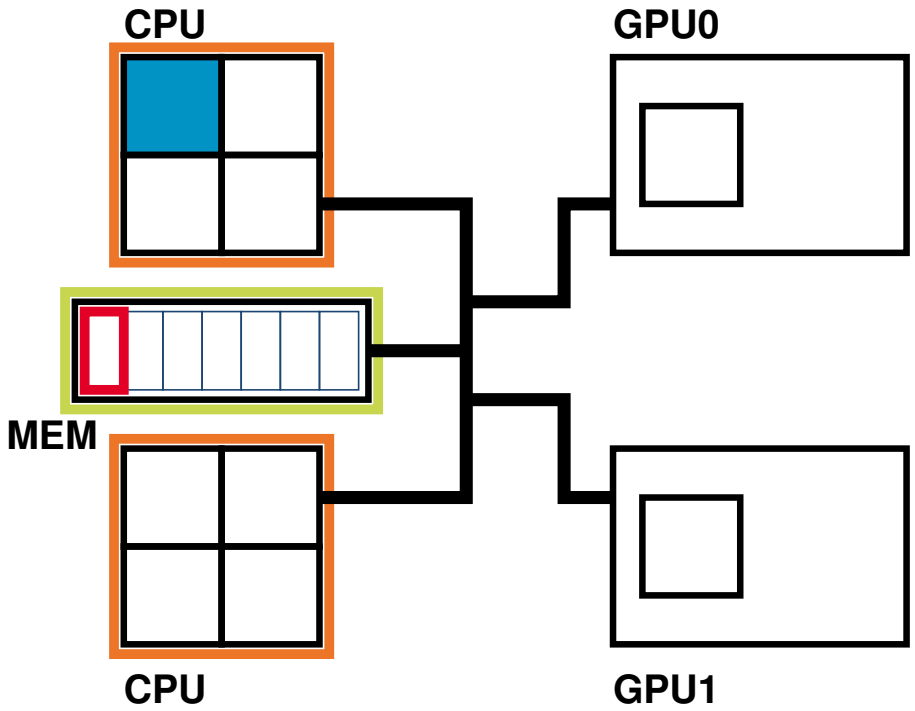
StarPU Heterogeneous Execution Model / Data Management

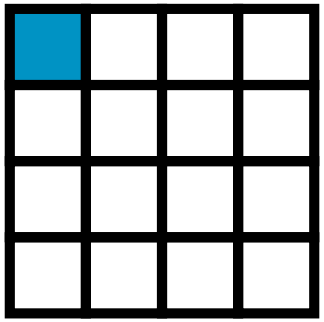


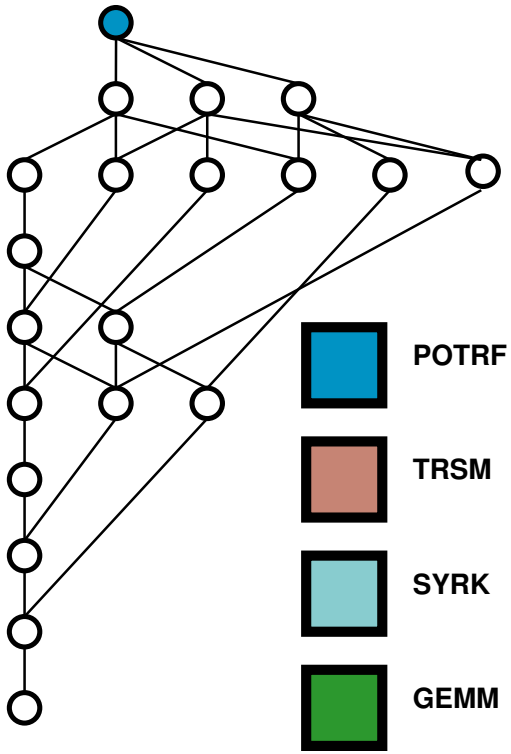




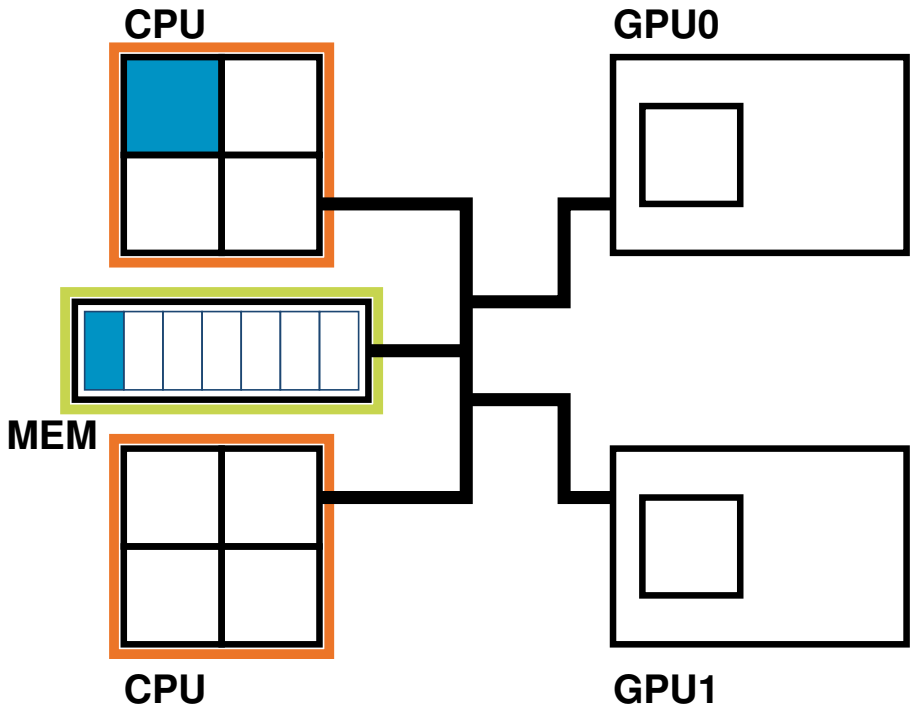
StarPU Heterogeneous Execution Model / Data Management

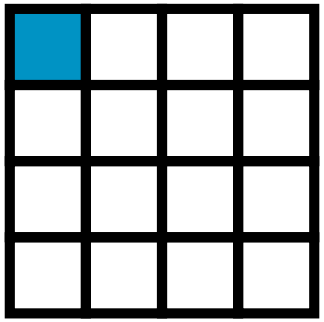


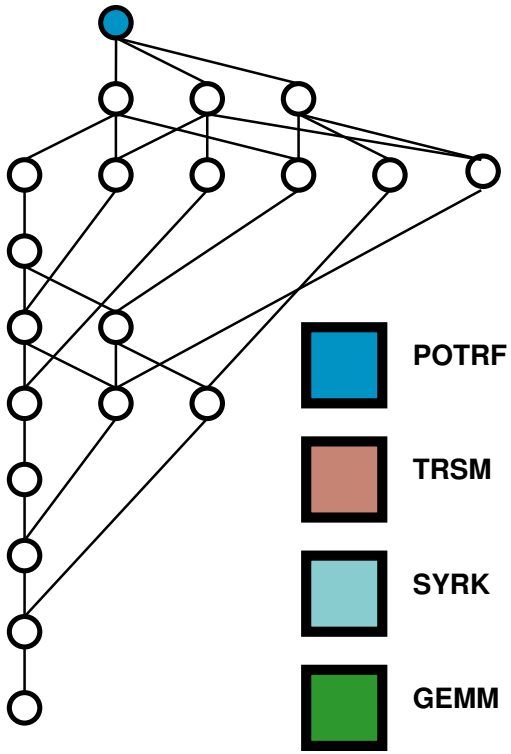




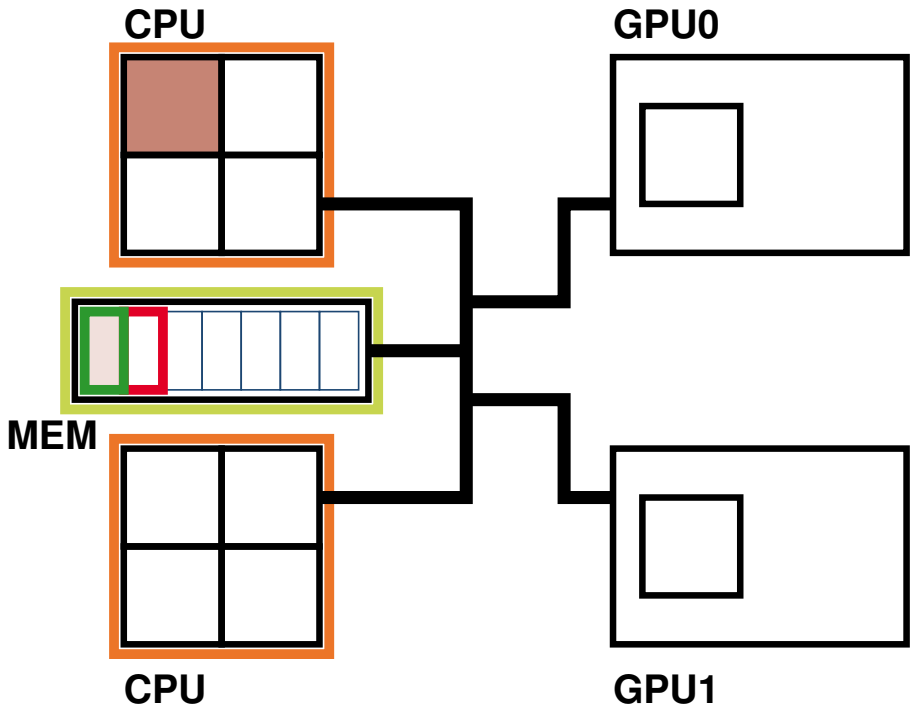
StarPU Heterogeneous Execution Model / Data Management

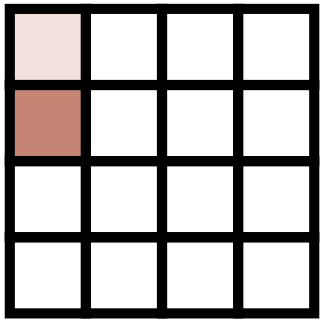


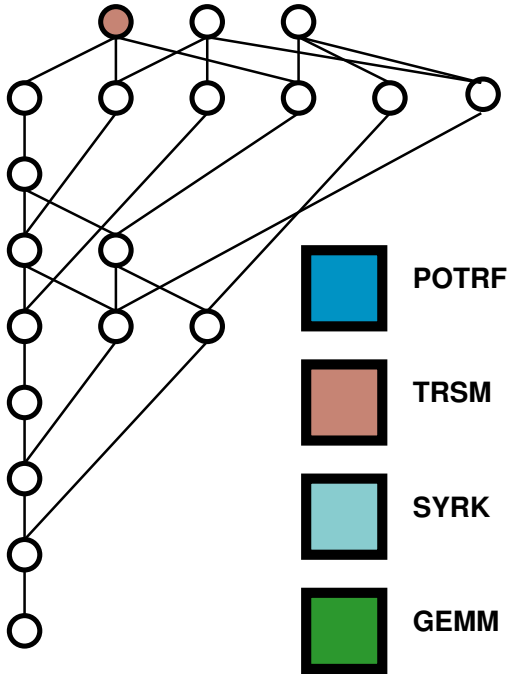




StarPU Heterogeneous Execution Model / Data Management

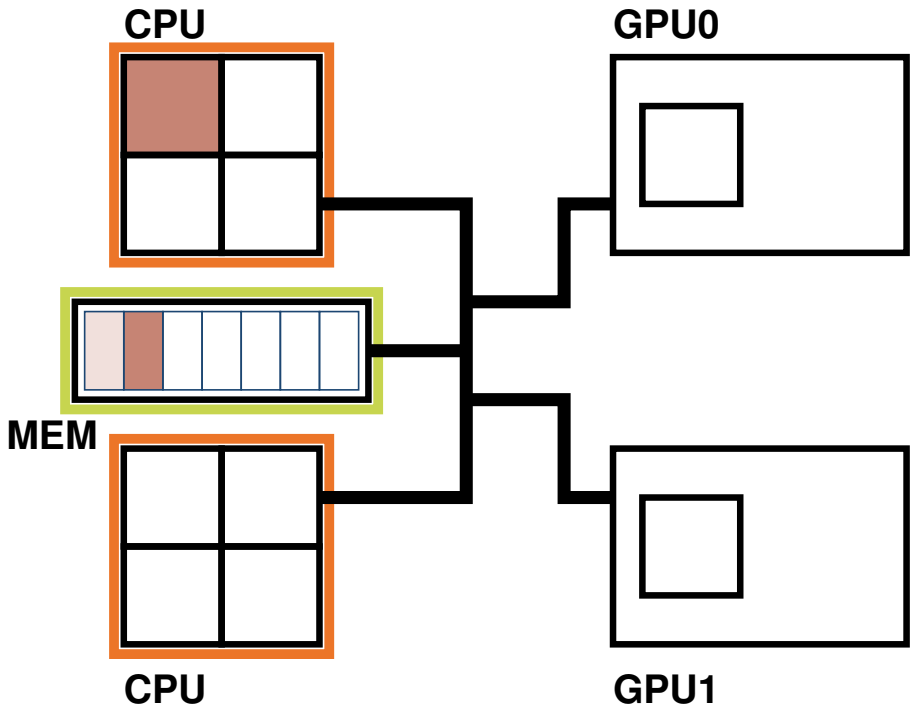


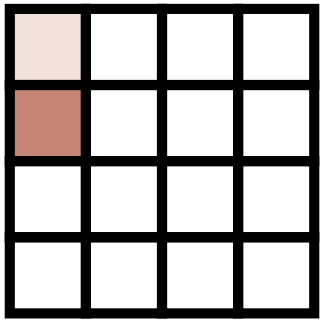


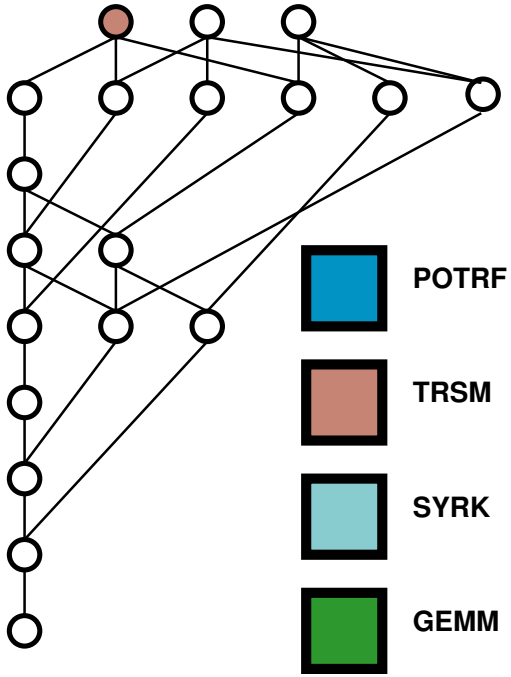


- Handles dependencies

StarPU Heterogeneous Execution Model / Data Management

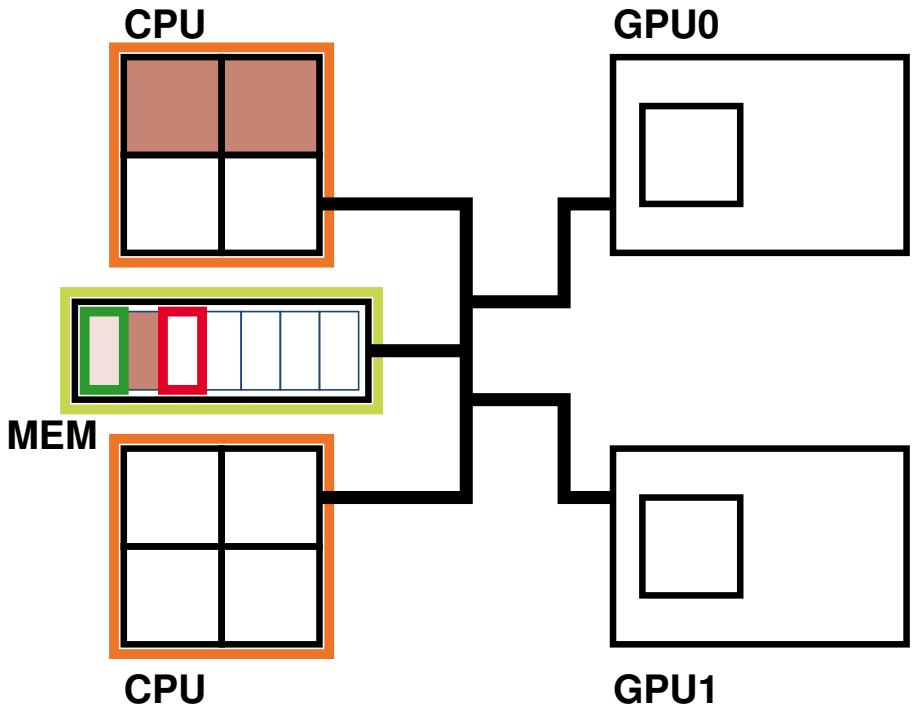


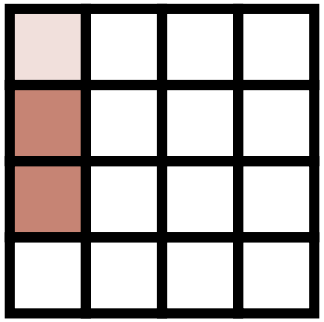


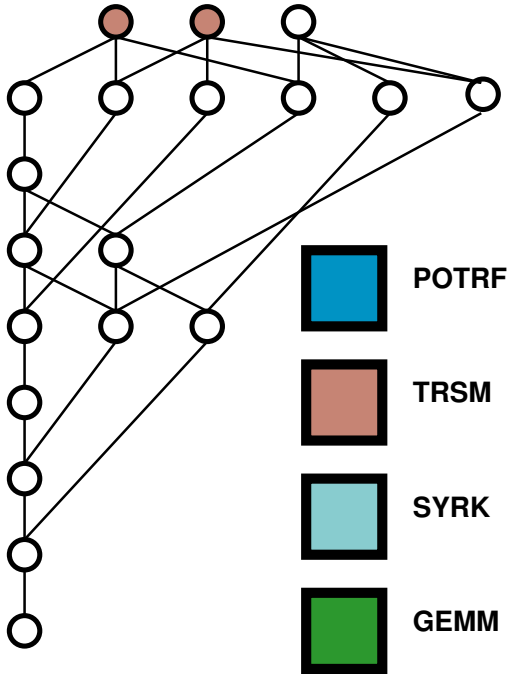


- Handles dependencies

StarPU Heterogeneous Execution Model / Data Management

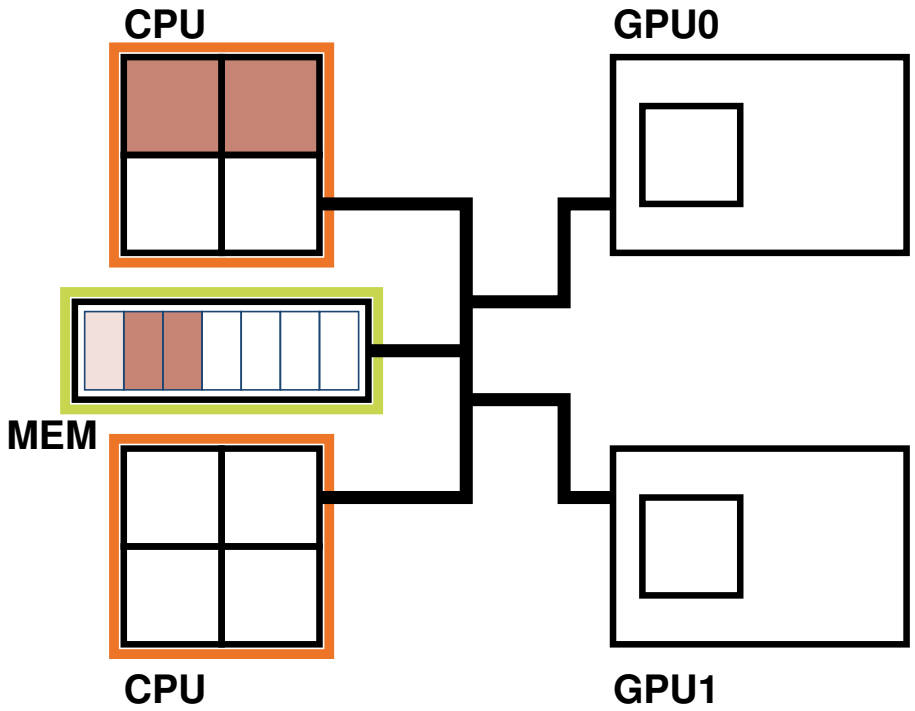


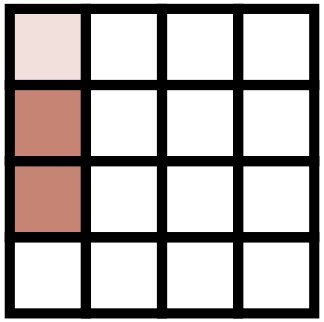


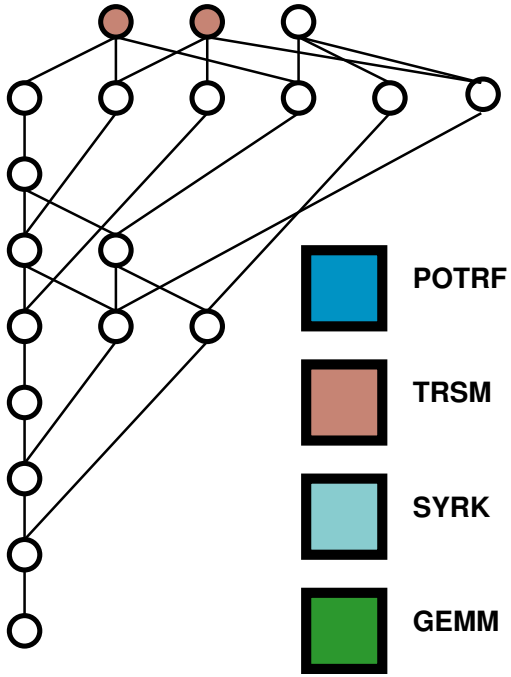


- Handles dependencies

StarPU Heterogeneous Execution Model / Data Management

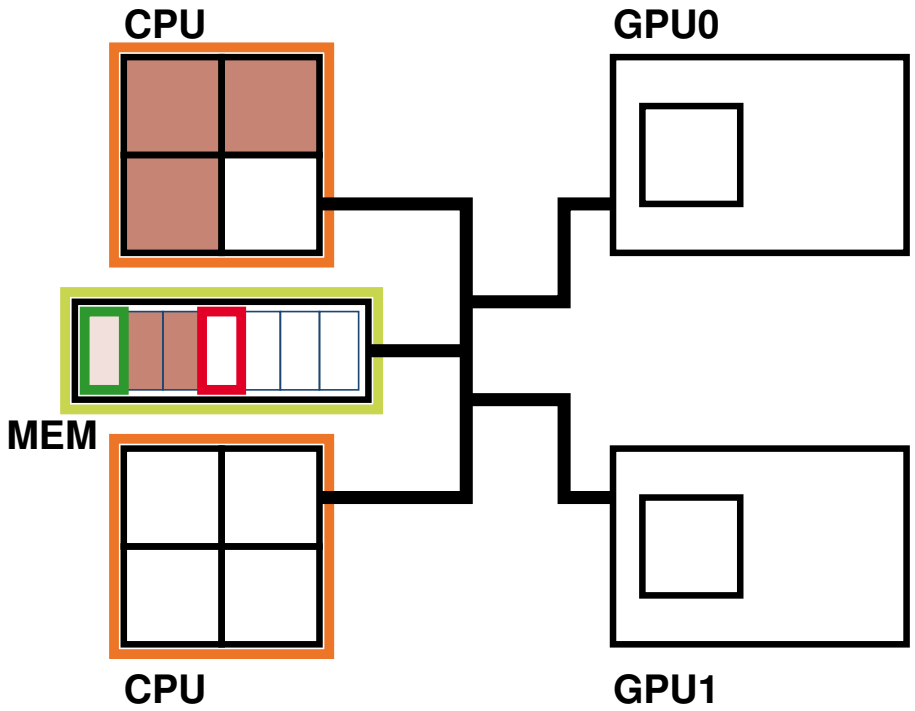


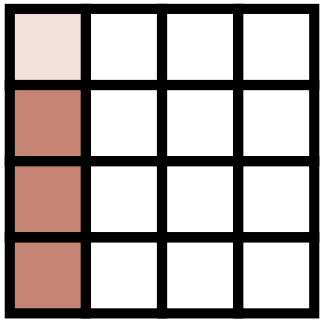


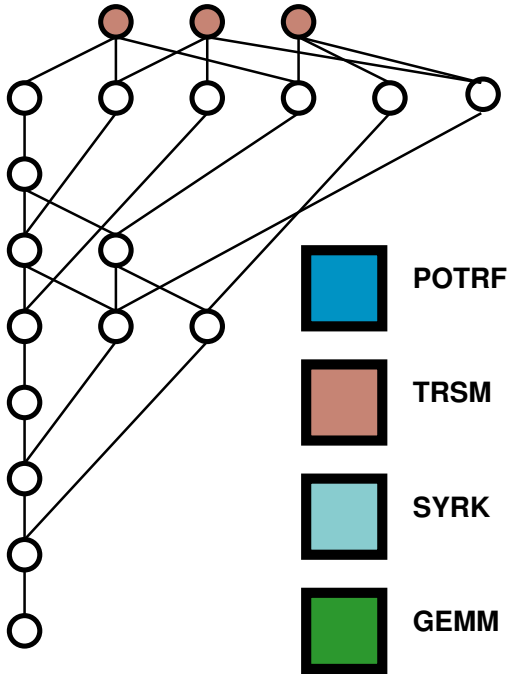


- Handles dependencies

StarPU Heterogeneous Execution Model / Data Management

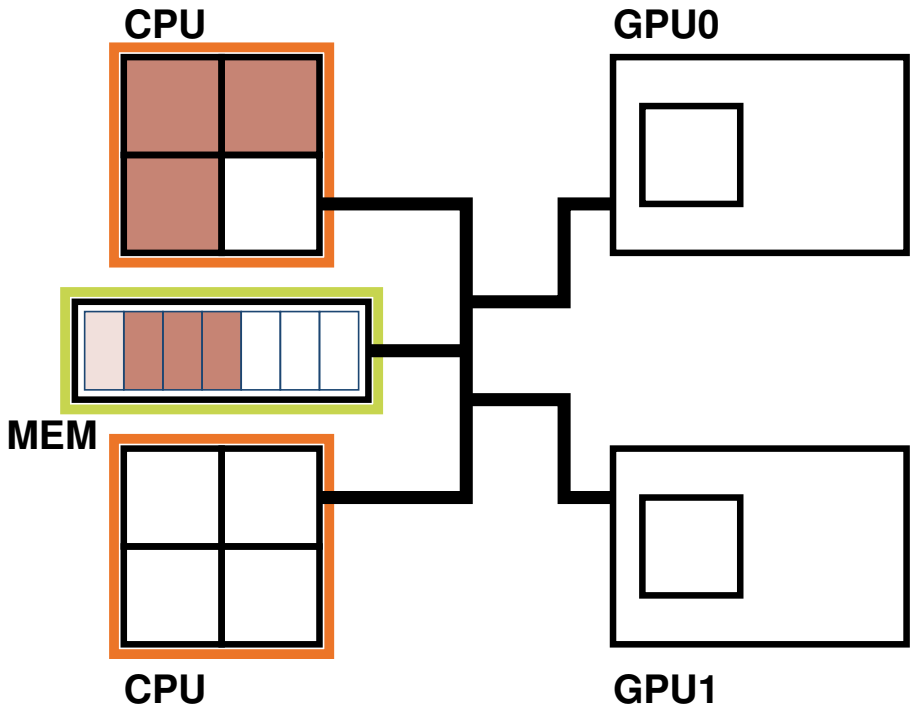


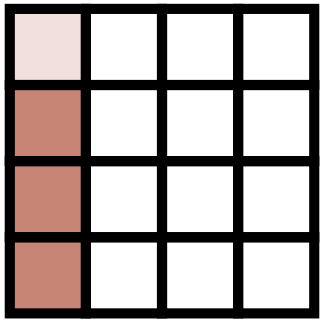


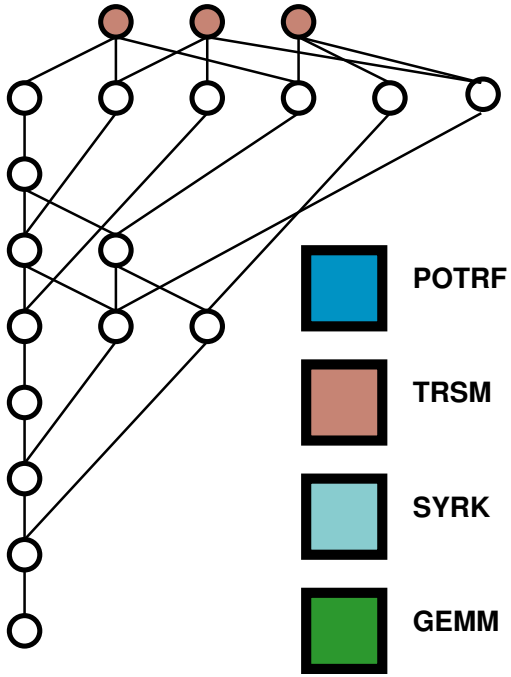


- Handles dependencies

StarPU Heterogeneous Execution Model / Data Management

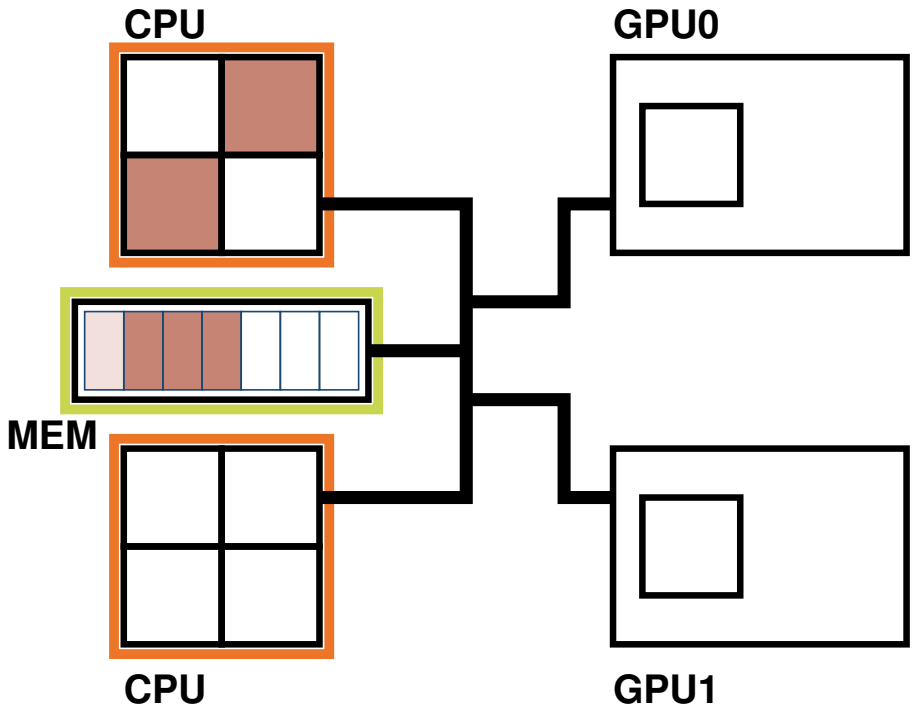


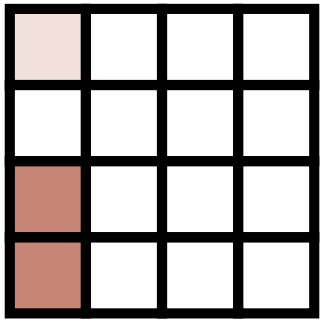


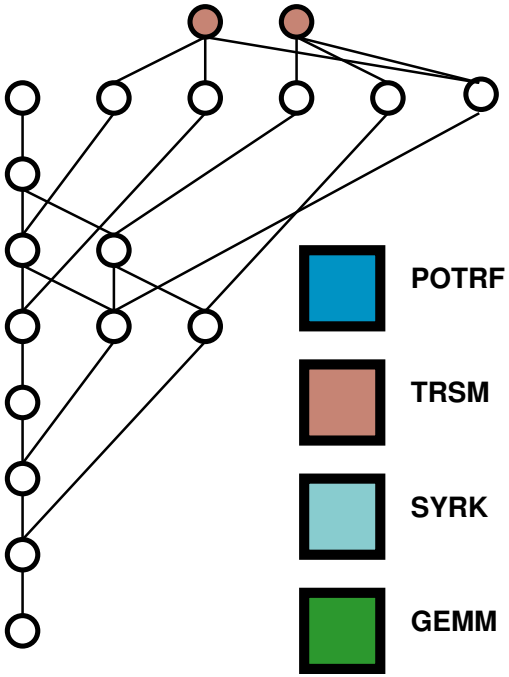


- Handles dependencies

StarPU Heterogeneous Execution Model / Data Management

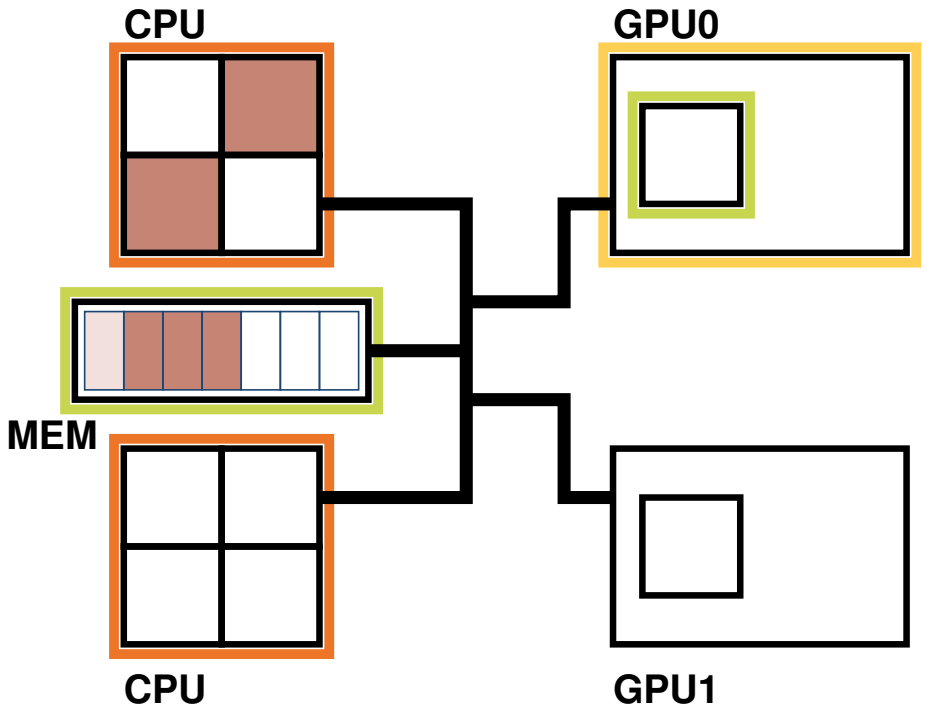


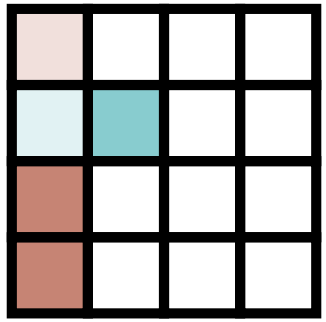


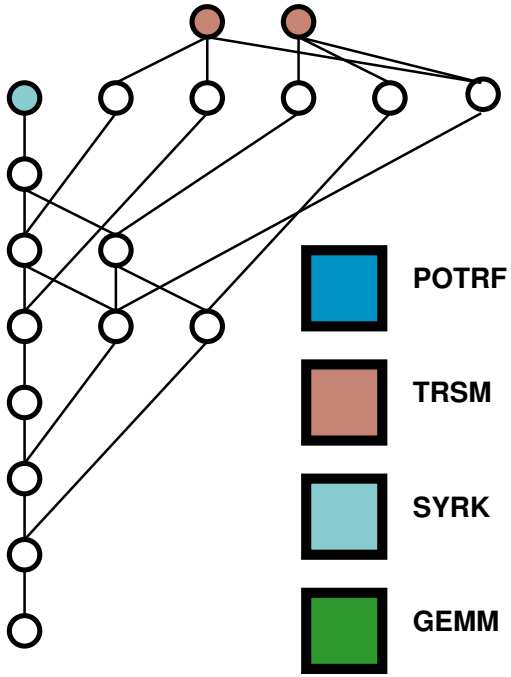


- Handles dependencies

StarPU Heterogeneous Execution Model / Data Management

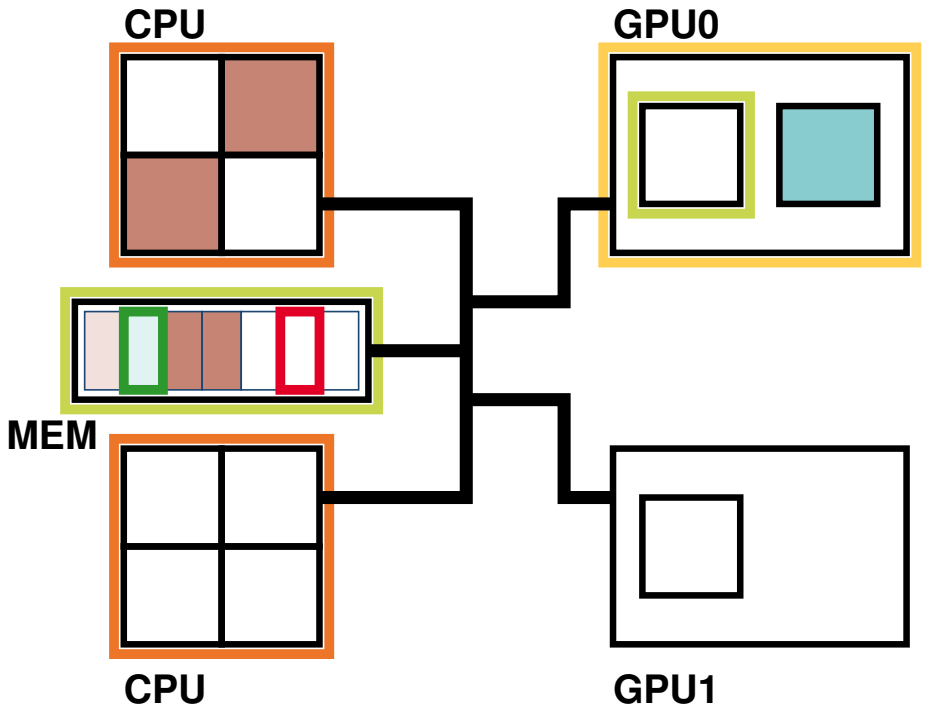


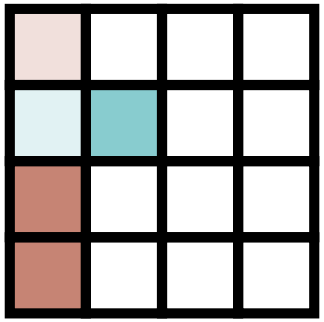


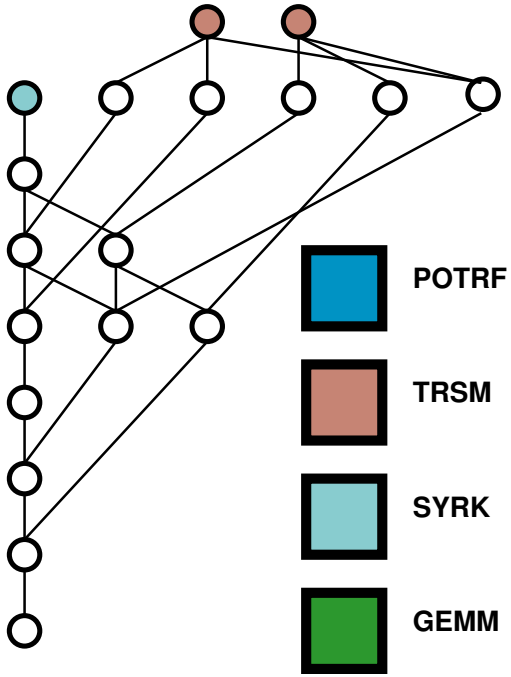


- Handles dependencies
- Handles scheduling (policy)

StarPU Heterogeneous Execution Model / Data Management

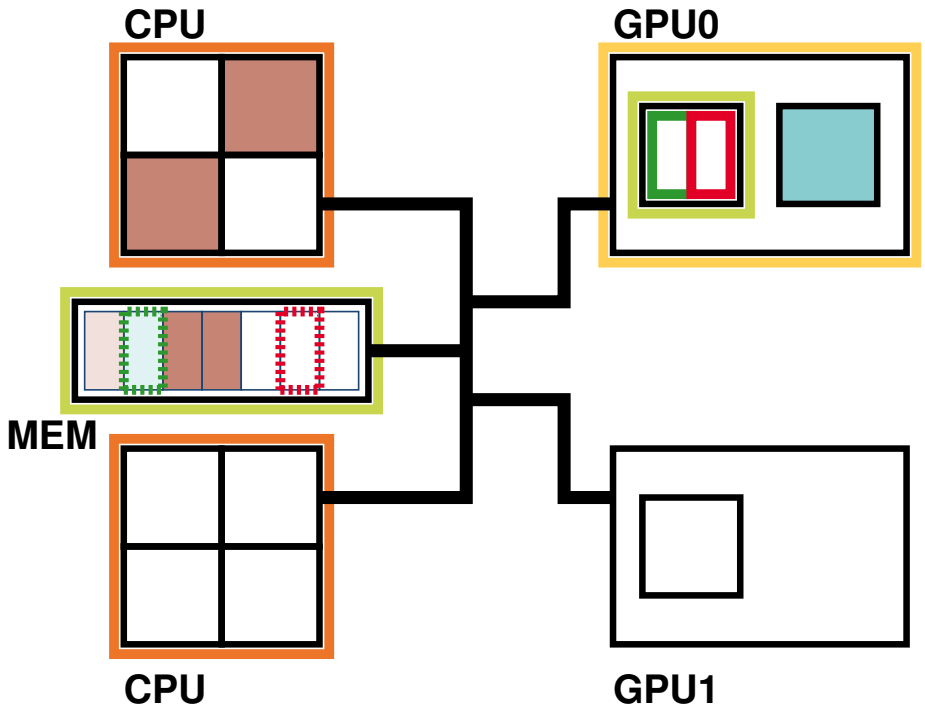


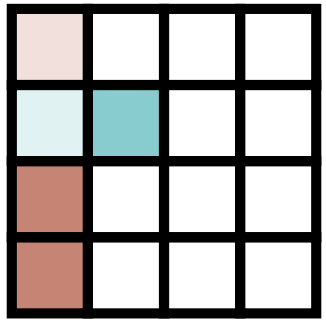


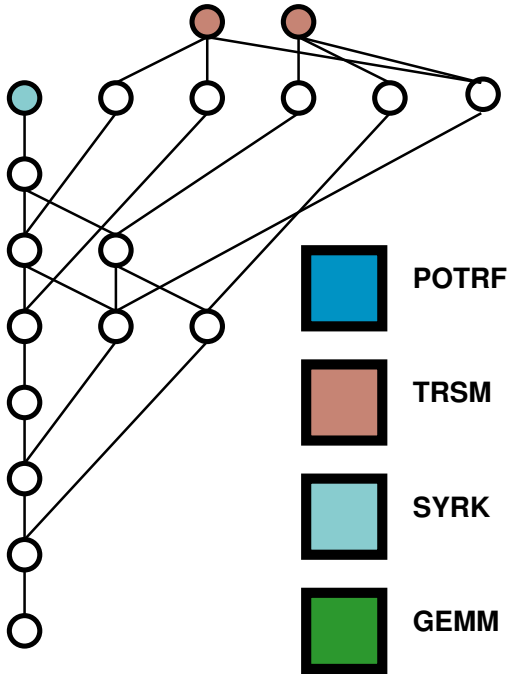


- Handles dependencies
- Handles scheduling (policy)

StarPU Heterogeneous Execution Model / Data Management

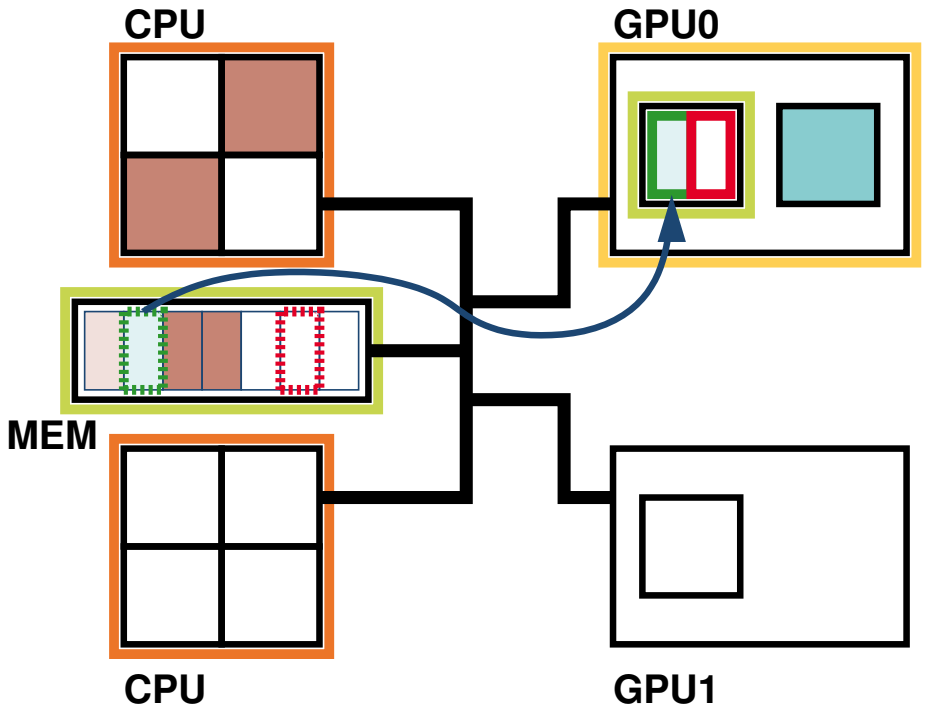


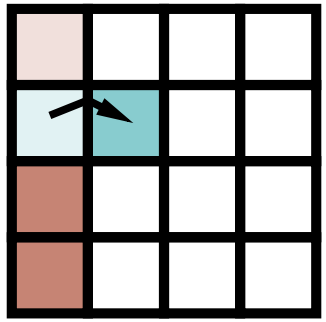


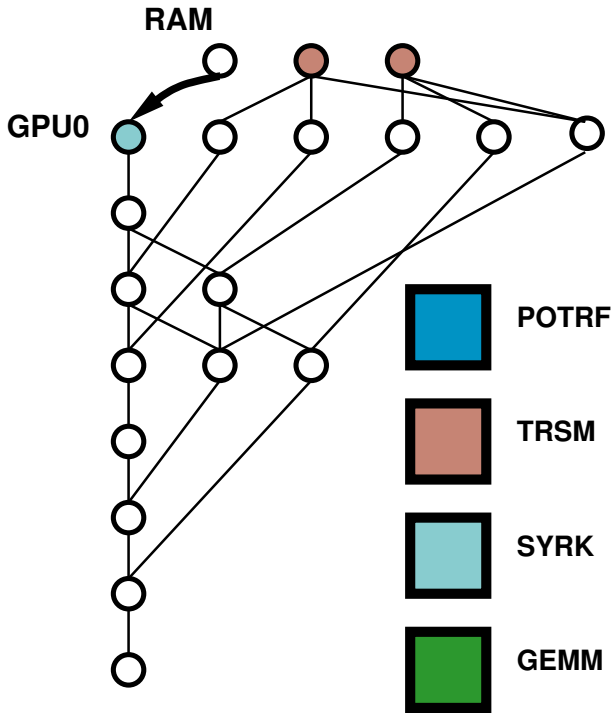


- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)

StarPU Heterogeneous Execution Model / Data Management

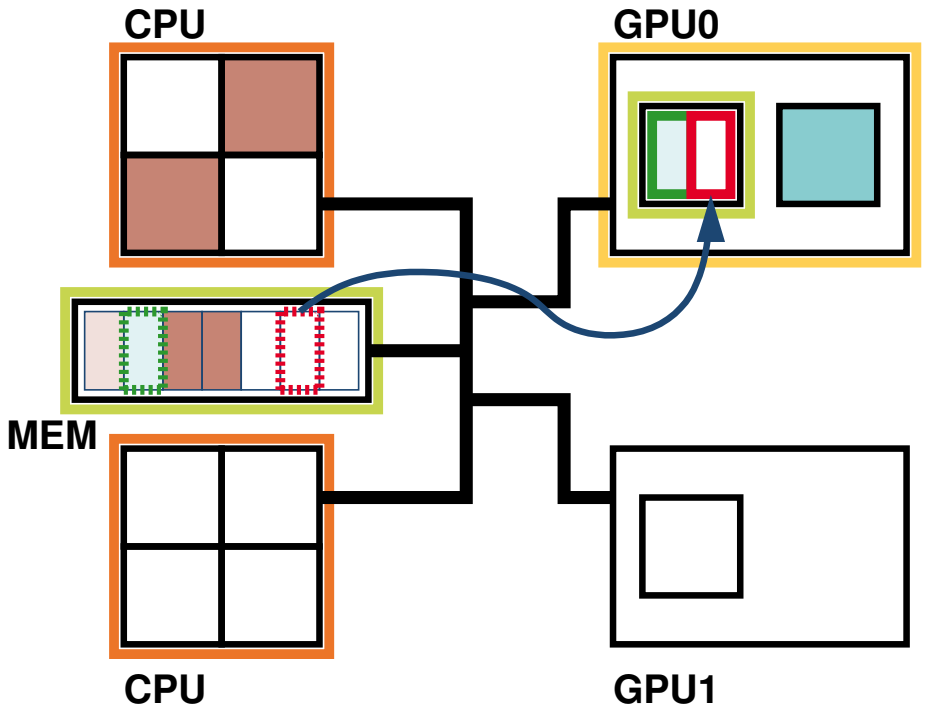


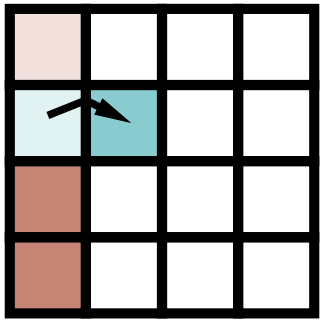


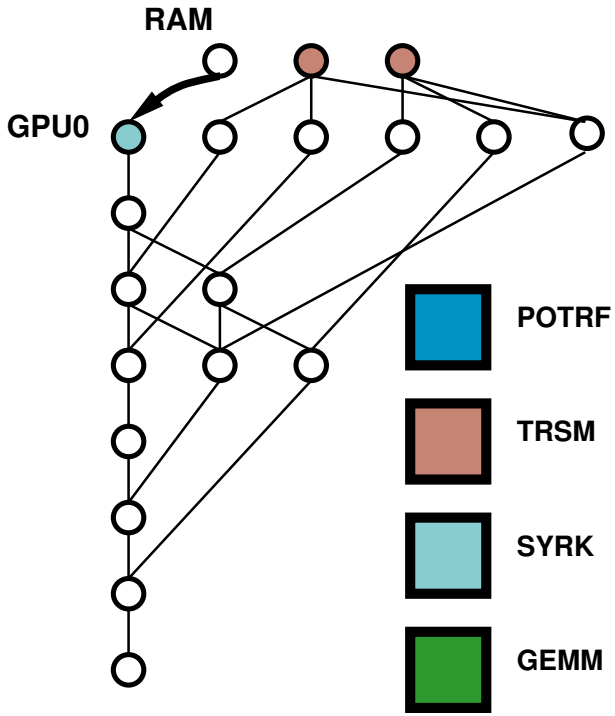


- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)

StarPU Heterogeneous Execution Model / Data Management

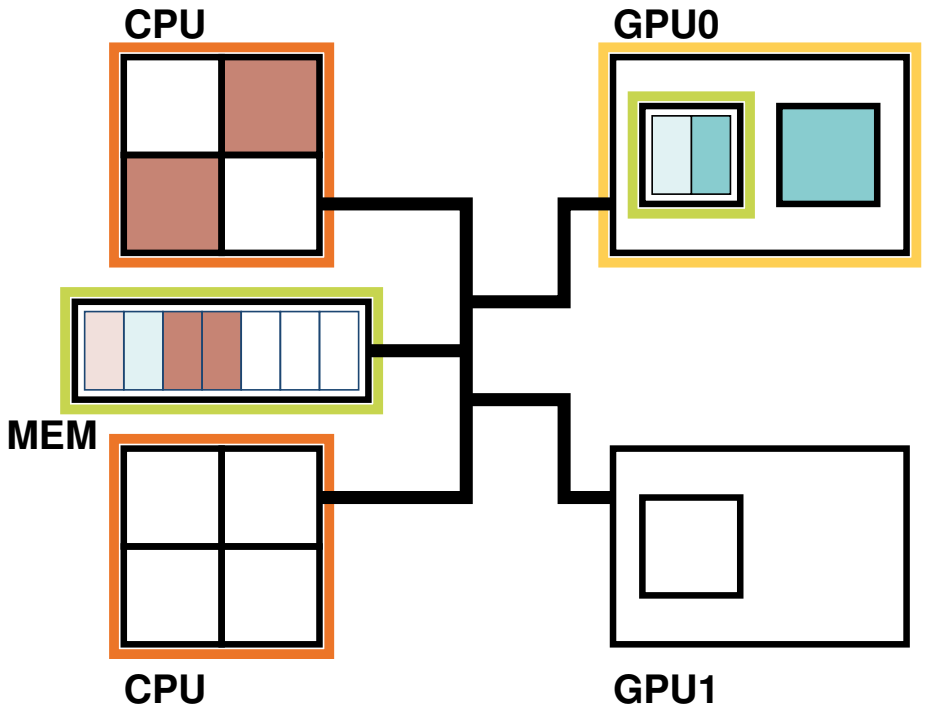


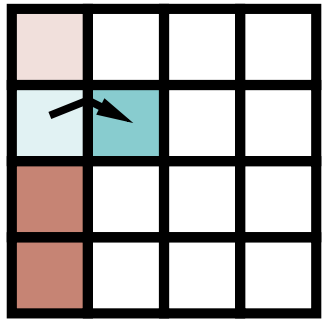


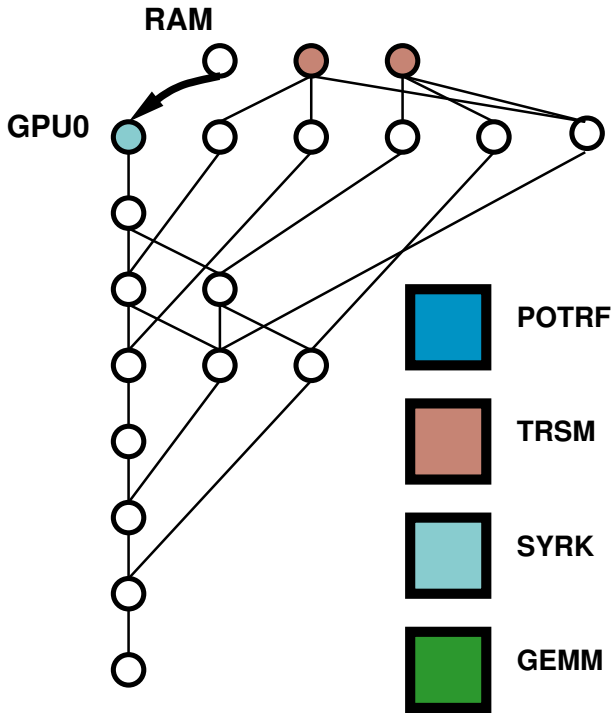


- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)

StarPU Heterogeneous Execution Model / Data Management

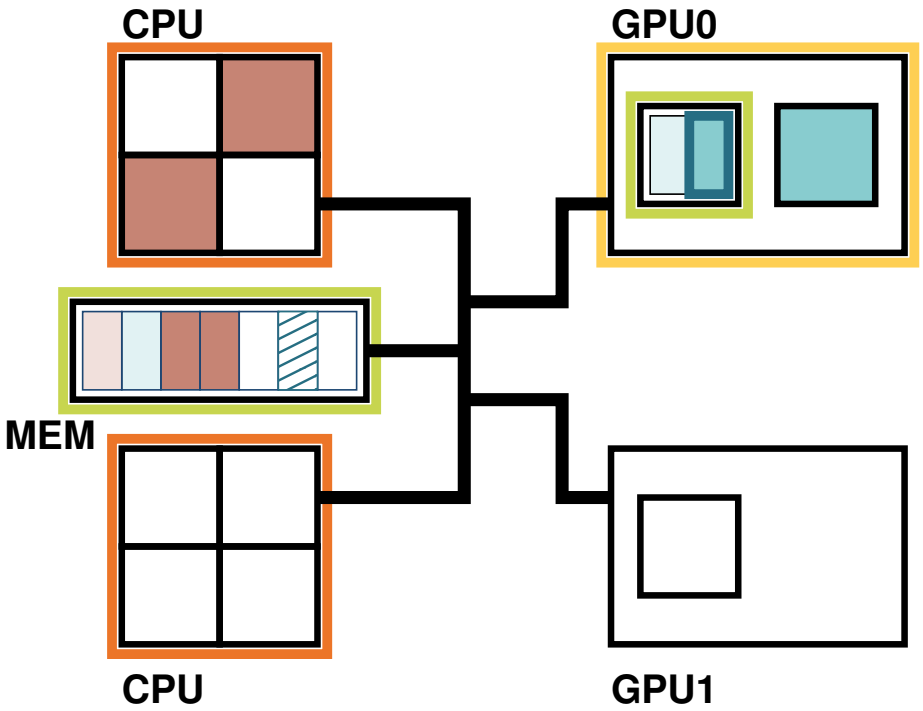


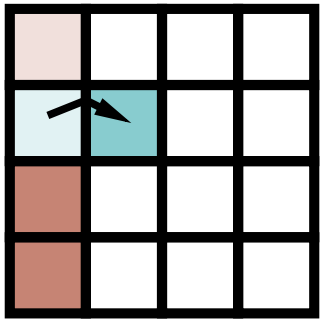


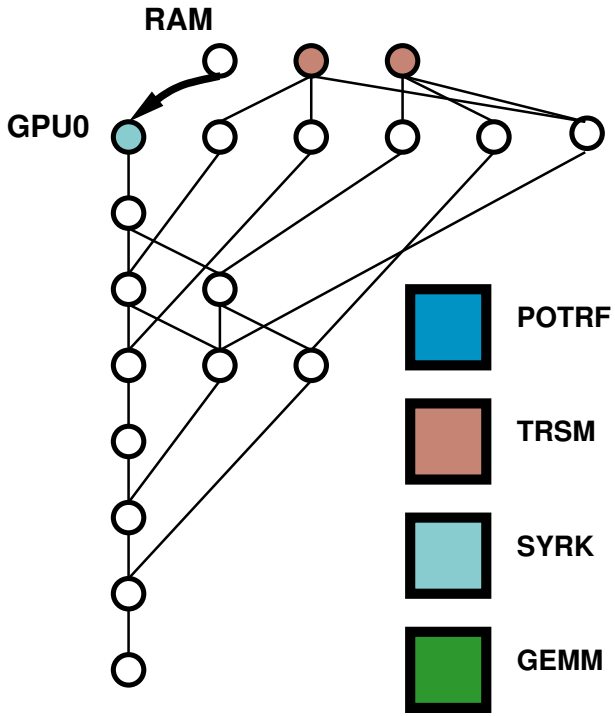


- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)

StarPU Heterogeneous Execution Model / Data Management





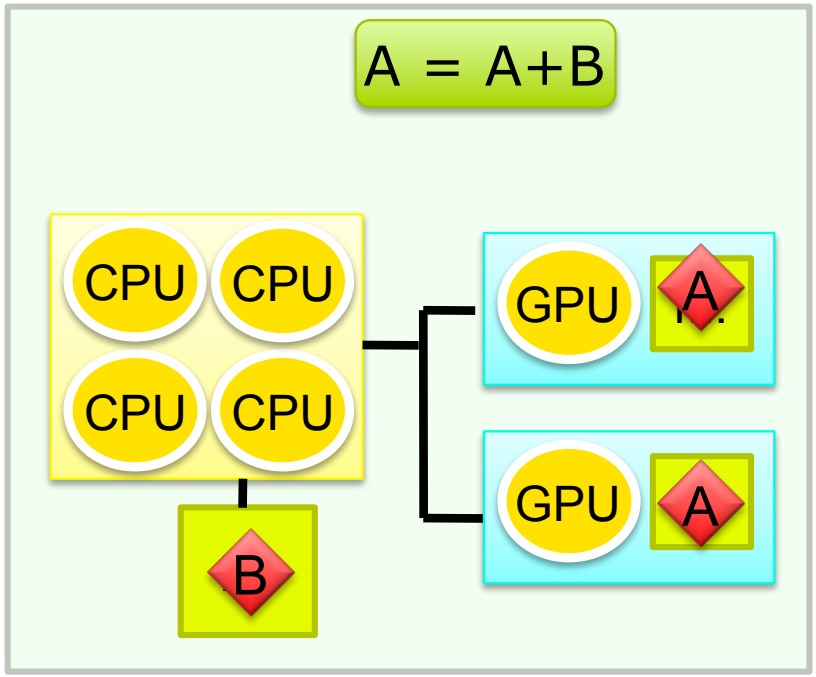


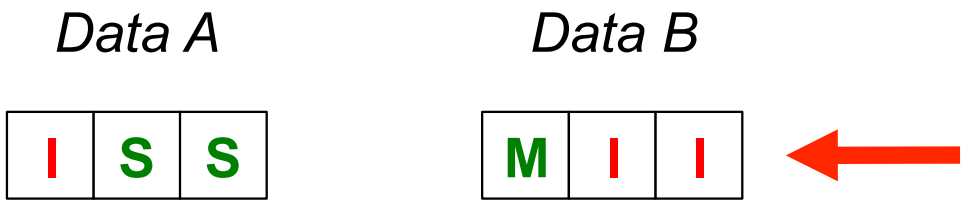
- Handles dependencies
- Handles scheduling (policy)
- Handles data consistency (MSI protocol)

Distributed Shared Memory Consistency

MSI Protocol

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

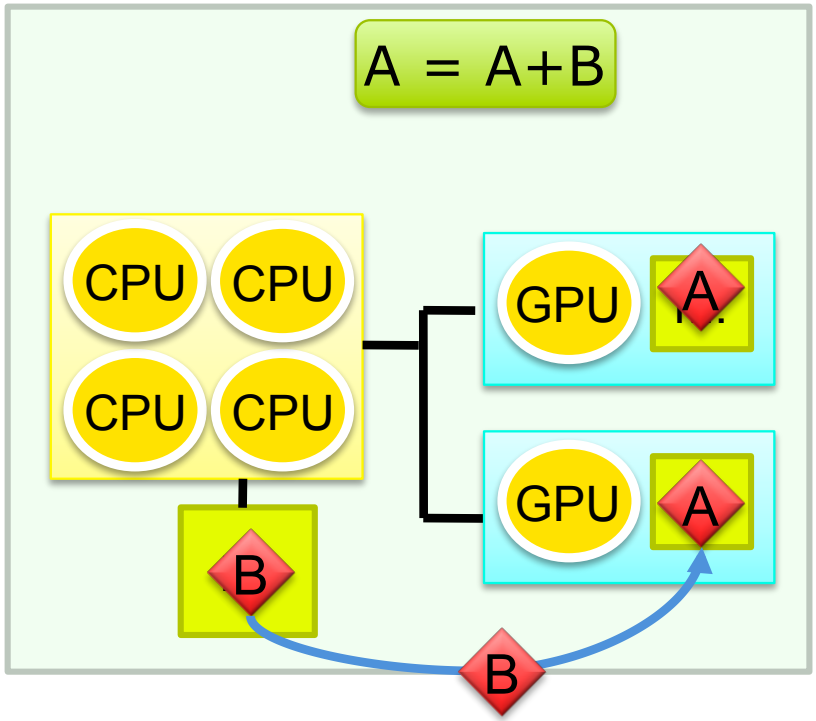




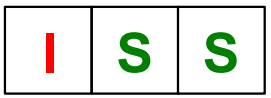
Distributed Shared Memory Consistency

MSI Protocol

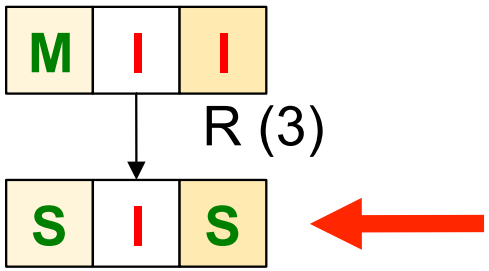
- M: Modified
- S: Shared
- I: Invalid



Data A



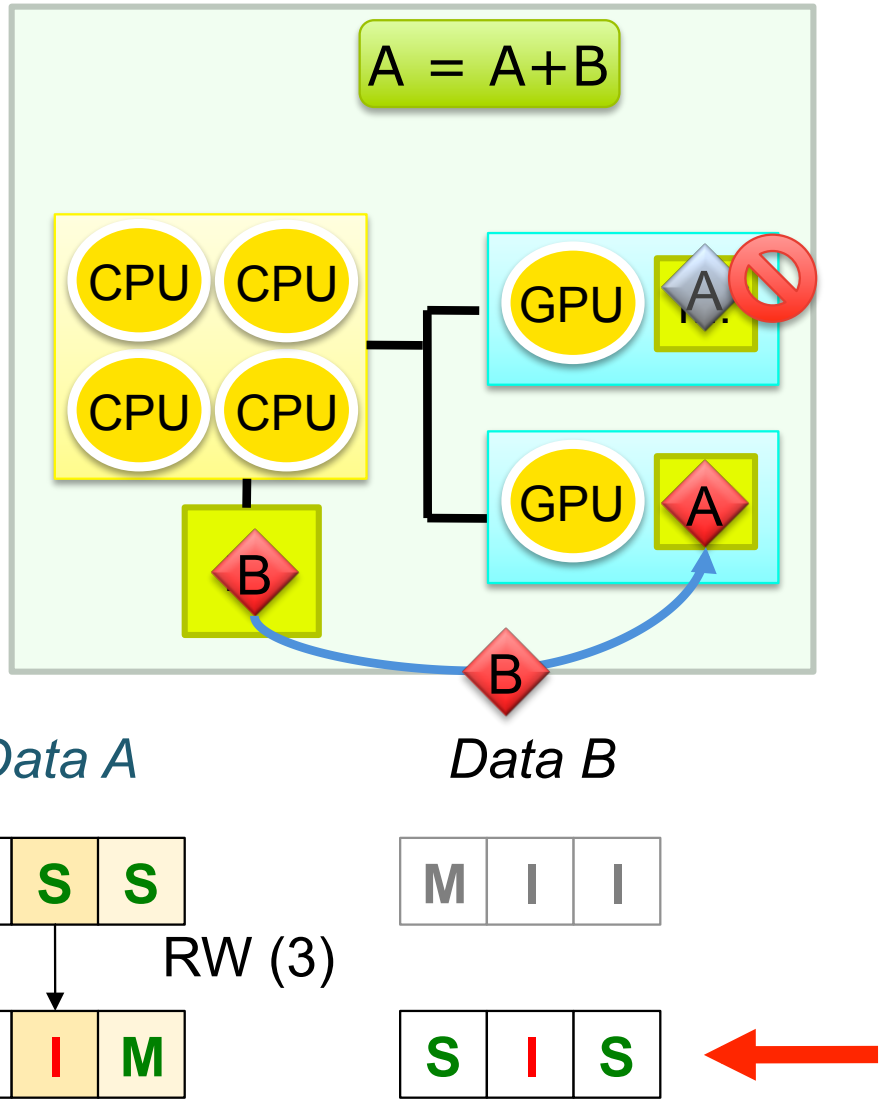
Data B



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

```
1 int vector[NX];
2 starpu_data_handle_t handle;
3
4 starpu_vector_data_register(&handle, 0, (uintptr_t)vector,
5                             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

```
1 ...
2 starpu_data_handle_t handle;
3
4 starpu_bcsr_data_register(&handle, 0, NNZ, NROW,
5             (uintptr_t)bcsr_matrix_data,
6             bcsr_matrix_indices, bcsr_matrix_rowptr,
                  first_entry,
7             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, ...)

Defining a New Data Interface

Supporting custom data types

- Generic `starpu_data_register()` func
- Accepts a `struct starpu_data_interface_ops`
- Set of allocation/management/serialization routines

```
1  void starpu_complex_data_register(starpu_data_handle_t *handle,
2      unsigned home_node, double *real, double *imag, int nx) {
3      struct starpu_complex_interface cmplx_if = {
4          .real = real,
5          .imaginary = imag,
6          .nx = nx
7      };
8
9      if (cmplx_ops.interfaceid == STARPU_UNKNOWN_INTERFACE_ID) {
10         cmplx_ops.interfaceid = starpu_data_interface_get_next_id();
11     }
12
13   starpu_data_register(handle, home_node, &cmplx_if, &cmplx_ops);
14 }
```

Defining a New Data Interface

Supporting custom data types

- Generic `starpu_data_register()` func
- Accepts a `struct starpu_data_interface_ops`
- Set of allocation/management/serialization routines

Structure `starpu_data_interface_ops`:

```
1  struct starpu_data_interface_ops interface_complex_ops =
2  {
3          .register_data_handle = complex_register_data_handle,
4          .allocate_data_on_node = complex_allocate_data_on_node,
5          .copy_methods = &complex_copy_methods,
6          .get_size = complex_get_size,
7          .footprint = complex_footprint,
8          .pack_data = complex_pack_data,
9          .unpack_data = complex_unpack_data,
10         .interfaceid = STARPU_UNKNOWN_INTERFACE_ID,
11         .interface_size = sizeof(struct starpu_complex_interface),
12 };
```

Defining a New Data Interface

Supporting custom data types

- Generic `starpu_data_register()` func
- Accepts a `struct starpu_data_interface_ops`
- Set of allocation/management/serialization routines

Example for `starpu_data_interface_ops::register_data_handle()`:

```
1  static void complex_register_data_handle(starpu_data_handle_t handle,
2      unsigned home_node, void *data_if) {
3      struct starpu_complex_interface *cmplx_if = data_if;
4      for (unsigned node = 0; node < STARPU_MAXNODES; node++) {
5          struct starpu_complex_interface *local_if =
6              starpu_data_get_interface_on_node(handle, node);
7
8          local_if->nx = cmplx_if->nx;
9          if (node == home_node) {
10             local_if->real = cmplx_if->real;
11             local_if->imag = cmplx_if->imag;
12         } else {
13             local_if->real = NULL;
14             local_if->imag = NULL;
15         }
16     }
17 }
```

Defining a New Data Interface

Supporting custom data types

- Generic `starpu_data_register()` func
- Accepts a `struct starpu_data_interface_ops`
- Set of allocation/management/serialization routines

Example for `starpu_data_interface_ops::pack_data()`:

```
1  static int complex_pack_data(starpu_data_handle_t handle, unsigned node,
2                               void **ptr, ssize_t *count) {
3      STARPU_ASSERT(starpu_data_test_if_allocated_on_node(handle, node));
4      struct starpu_complex_interface *cmplx_if =
5          starpu_data_get_interface_on_node(handle, node);
6
7      *count = complex_get_size(handle);
8      *ptr = starpu_malloc_on_node_flags(node, *count, 0);
9
10     memcpy(*ptr, cmplx_if->real, cmplx_if->nx*sizeof(double));
11     memcpy(*ptr+cmplx_if->nx*sizeof(double), cmplx_if->imag,
12             cmplx_if->nx*sizeof(double));
13
14     return 0;
15 }
```

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

```
1  int vector[NX];
2  starpu_data_handle_t handle;
3  starpu_vector_data_register(&handle, 0, (uintptr_t)vector,
4                              NX, sizeof(vector[0]));
5
6  /* Partition the vector in NB_PARTS sub-vectors */
7  struct starpu_data_filter filter = {
8      .filter_func = starpu_vector_filter_block,
9      .nchildren = NB_PARTS
10 };
11 starpu_data_partition(handle, &filter);
12
13 /* 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

```
1 /* Wait for submitted tasks to complete */
2 starpu_task_wait_for_all();
3
4 /* Unpartition data */
5 starpu_data_unpartition(handle, 0);
6
7 /* 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

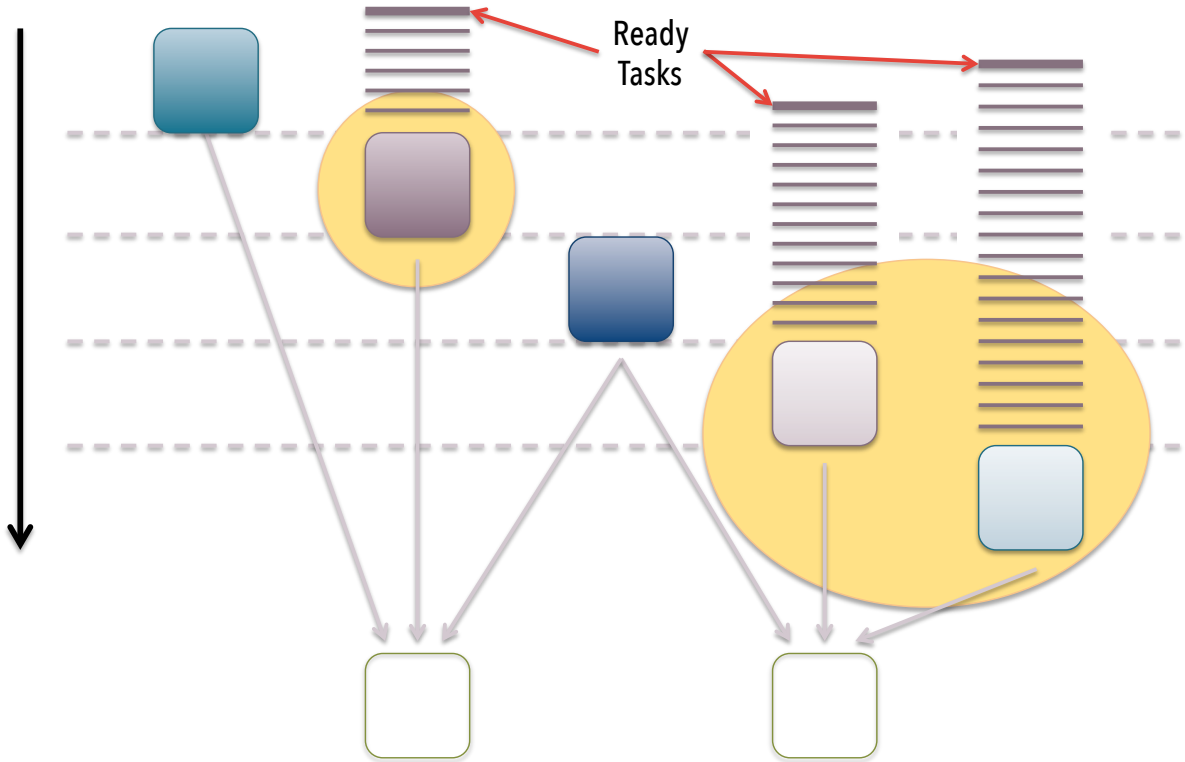
```
starpu_variable_data_register(&accum_handle, -1,
                              NULL, sizeof(type));
starpu_data_set_reduction_methods(accum_handle,
                                  &accumulate_cl, &bzero_cl);

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, FMM

Commutative Write Accesses

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



Commutative Write Accesses

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

Commutative Write Accesses

- Write accesses enforce sequential consistency by default
  - To strong for some kind of workloads
  - N-body, unstructured meshes, FMM
- **Commute**: allows a set of tasks to modify a buffer in any order
- Use of centralized arbiters may be needed on pathological cases

```
1  starpu_arbiter_t arbiter1, arbiter2;
2  arbiter1 = starpu_arbiter_create();
3  arbiter2 = starpu_arbiter_create();
4    ...
5  starpu_data_assign_arbiter(handles[idxHandle],
6                             (idxHandle%2)?arbiter1:arbiter2);
7    ...
8  ret = starpu_task_insert(&codelet,
9        (STARPU_RW|STARPU_COMMUTE), handles[idxHandle1],
10       (STARPU_RW|STARPU_COMMUTE), handles[idxHandle2],
11       0);
12   ...
```

5

Analysis and Monitoring

Feedback mechanisms

Online Tools

- Statistics
- Visual debugging

Offline Tools

- Trace-based analysis

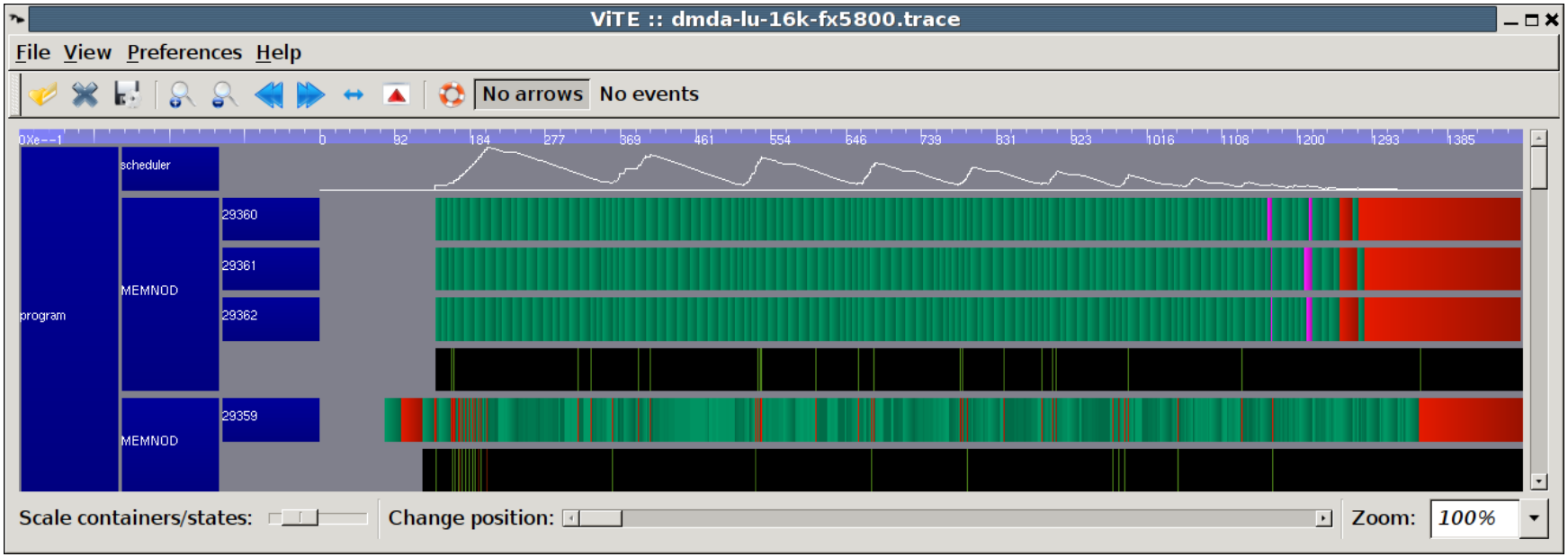
Offline Trace-Based Feedback

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

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

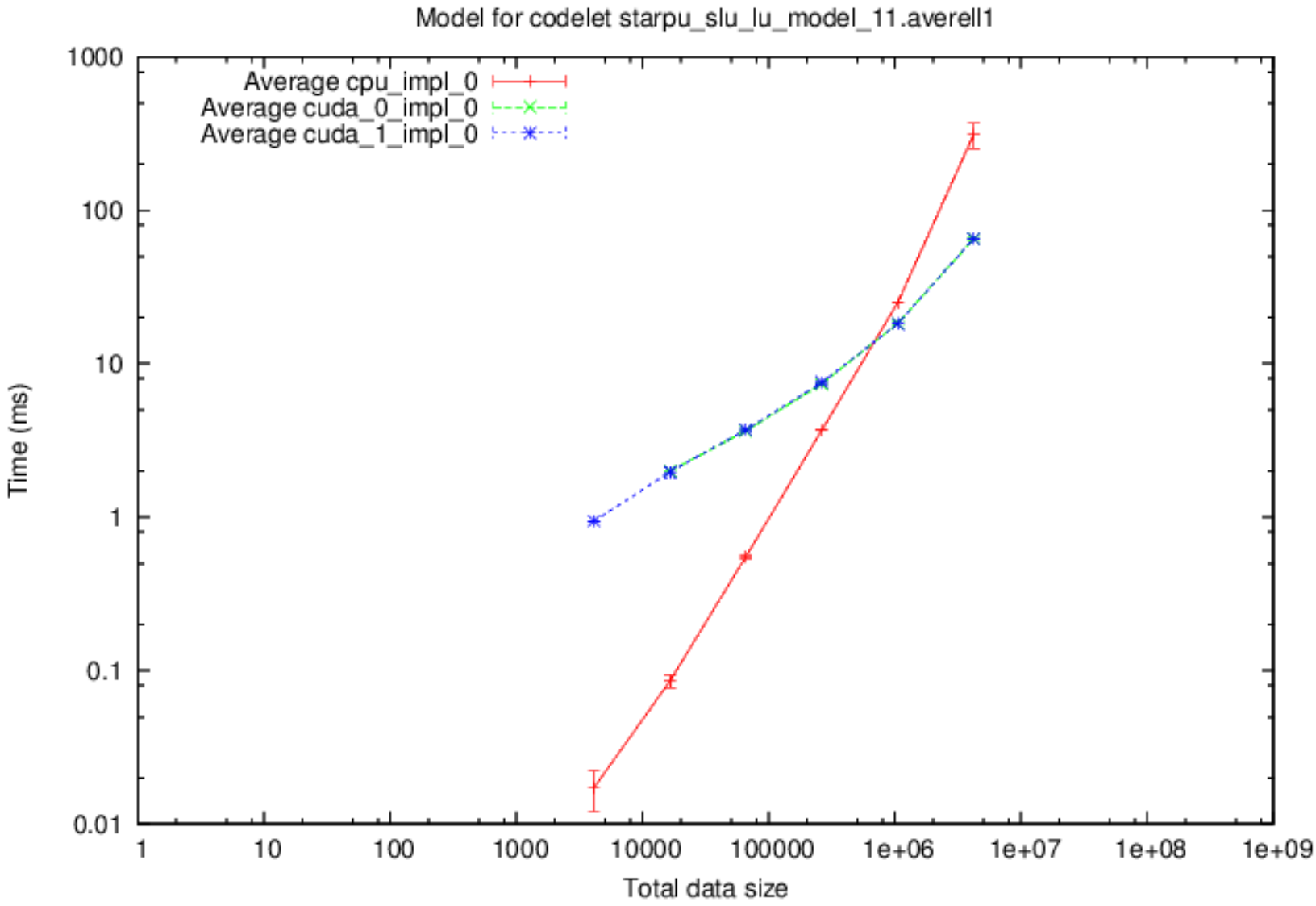
Offline Feedback – Kernel Model

Display the codelet performance models recorded by StarPU

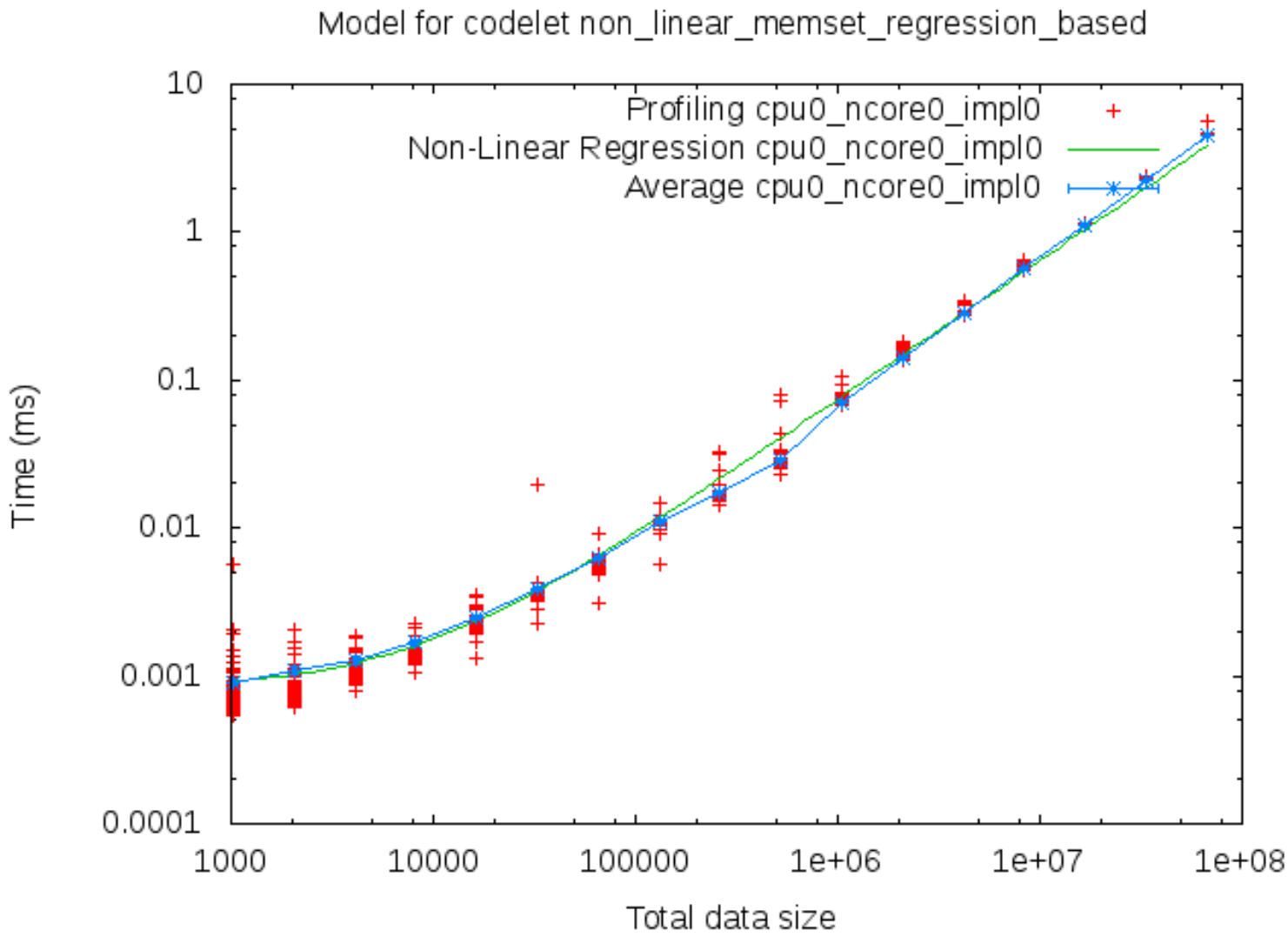
- Command-line tool starpu_perfmodel_display
- History-based models
- Regression-based models

```
$ 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
```

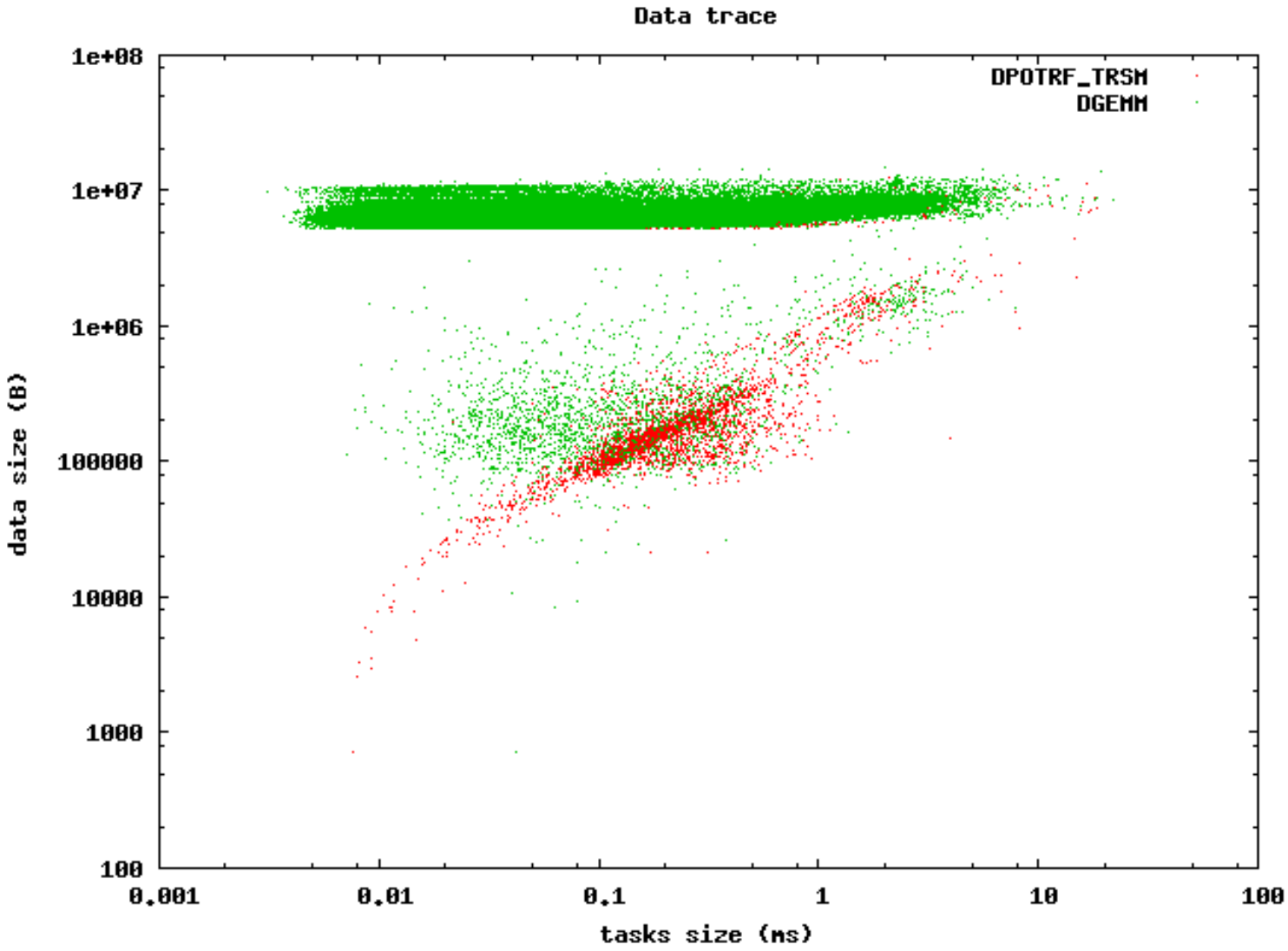
Offline Feedback – Kernel Model Characteristics



Offline Feedback – Kernel Model Regression Fitness



Offline Feedback – Synthetic Kernels' Behaviour



6

Distributed Computing

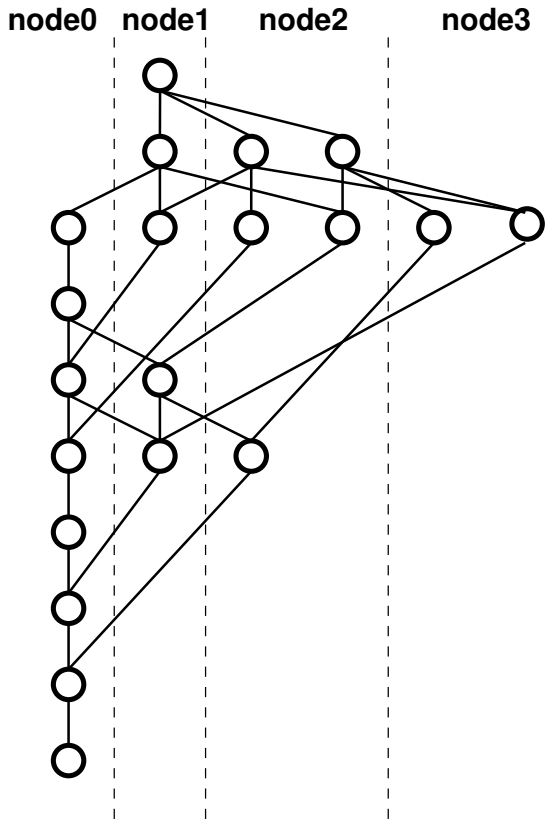
Distributed Support

Sequential Task Flow Paradigm on Clusters

Each node unrolls the sequential task flow

Data↔Node Mapping

- Provided by the application
- Can be altered dynamically



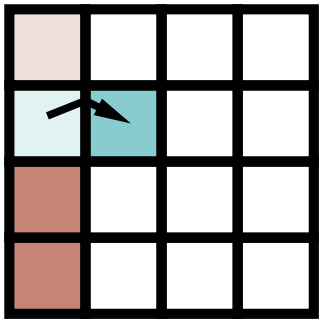
Distributed Support

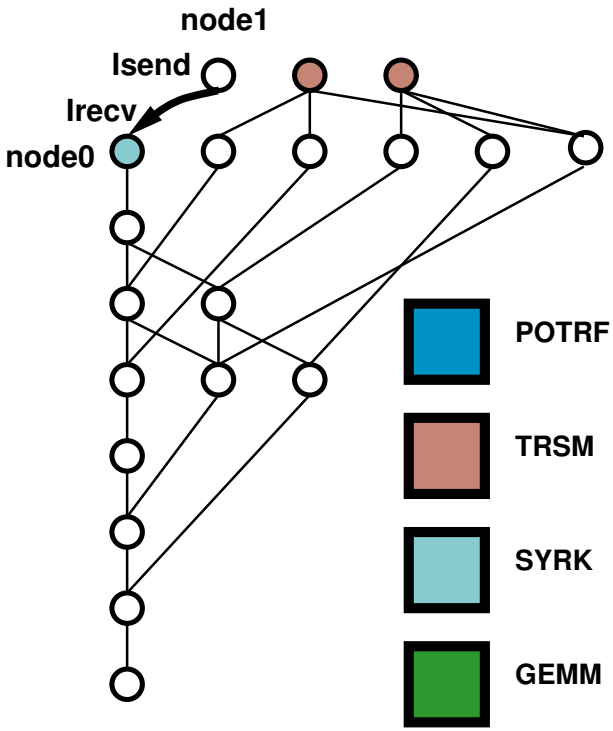
Sequential Task Flow Paradigm on Clusters

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

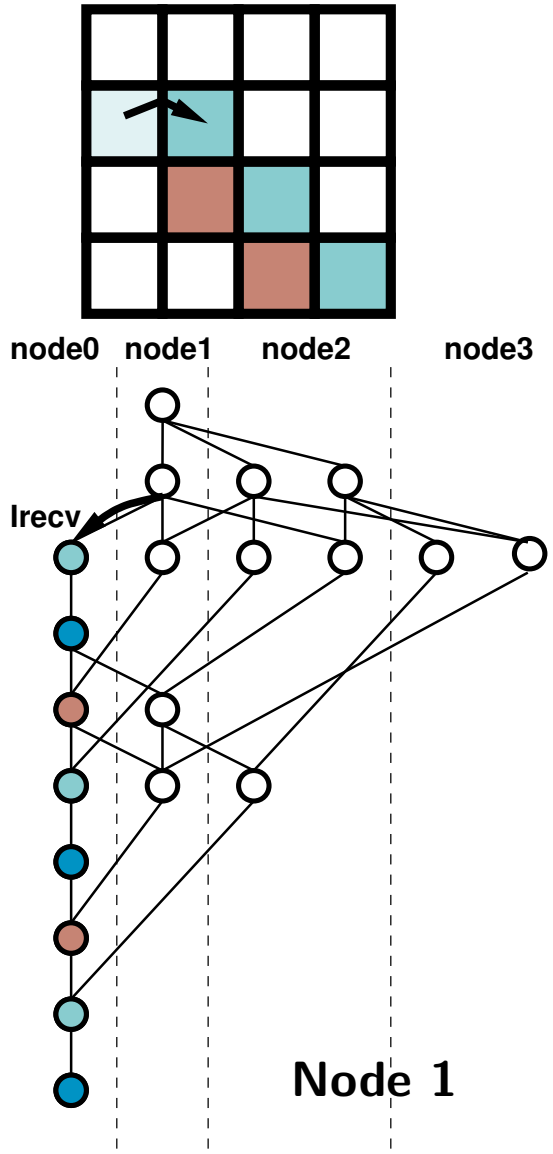
Each node unrolls the sequential task flow

Task↔Node Mapping

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

Optimization

- Local DAG pruning



Distributed STF Programming

Specific init/shutdown:

```
1       starpu_mpi_init_conf(&argc, &argv, 1, MPI_COMM_WORLD, NULL);
2       starpu_mpi_comm_rank(MPI_COMM_WORLD, &rank);
3       starpu_mpi_comm_size(MPI_COMM_WORLD, &size);
4
5       starpu_mpi_shutdown();
```

Distributed STF Programming

(Optional) distribution function:

```
1 int my_distrib(int x, int y, int nb_nodes) {
2     /* example: block distrib */
3     return ((int)(x / sqrt(nb_nodes)
4         + (y / sqrt(nb_nodes)) * sqrt(nb_nodes))) % nb_nodes;
5 }
```

Distributed STF Programming

Data registration:

```
1  unsigned matrix[X][Y];
2  starpu_data_handle_t data_handles[X][Y];
3  for(x = 0; x < X; x++) {
4      for (y = 0; y < Y; y++) {
5          int mpi_rank = my_distrib(x, y, size);
6
7          /* StarPU registration step */
8          if (mpi_rank == my_rank)
9              /* Owning data */
10             starpu_variable_data_register(&data_handles[x][y],
11                 STARPU_MAIN_RAM, (uintptr_t)&(matrix[x][y]), sizeof(unsigned));
12         else if (my_rank == my_distrib(x+1, y, size)
13                 my_rank == my_distrib(x-1, y, size)
14                 my_rank == my_distrib(x, y+1, size)
15                 my_rank == my_distrib(x, y-1, size))
16             /* Don't own this index, but will need it for computations */
17             starpu_variable_data_register(&data_handles[x][y],
18                                           -1, (uintptr_t)NULL, sizeof(unsigned));
19         else
20             /* Do not allocate anything for this */
21             data_handles[x][y] = NULL;
22
23         /* StarPU-MPI registration step */
24         if (data_handles[x][y]) {
25             starpu_mpi_data_register(data_handles[x][y], x*X+y, mpi_rank);
26         }
27     }
28 }
```

Distributed STF Programming

Task submission:

```
1  for(loop=0; loop<niter; loop++)
2      for (x = 1; x < X-1; x++)
3          for (y = 1; y < Y-1; y++)
4              starpu_mpi_task_insert(MPI_COMM_WORLD, &stencil5_cl,
5                                     STARPU_RW, data_handles[x][y],
6                                     STARPU_R, data_handles[x-1][y],
7                                     STARPU_R, data_handles[x+1][y],
8                                     STARPU_R, data_handles[x][y-1],
9                                     STARPU_R, data_handles[x][y+1],
10                                    0);
11 starpu_task_wait_for_all();
```

Distributed STF Programming

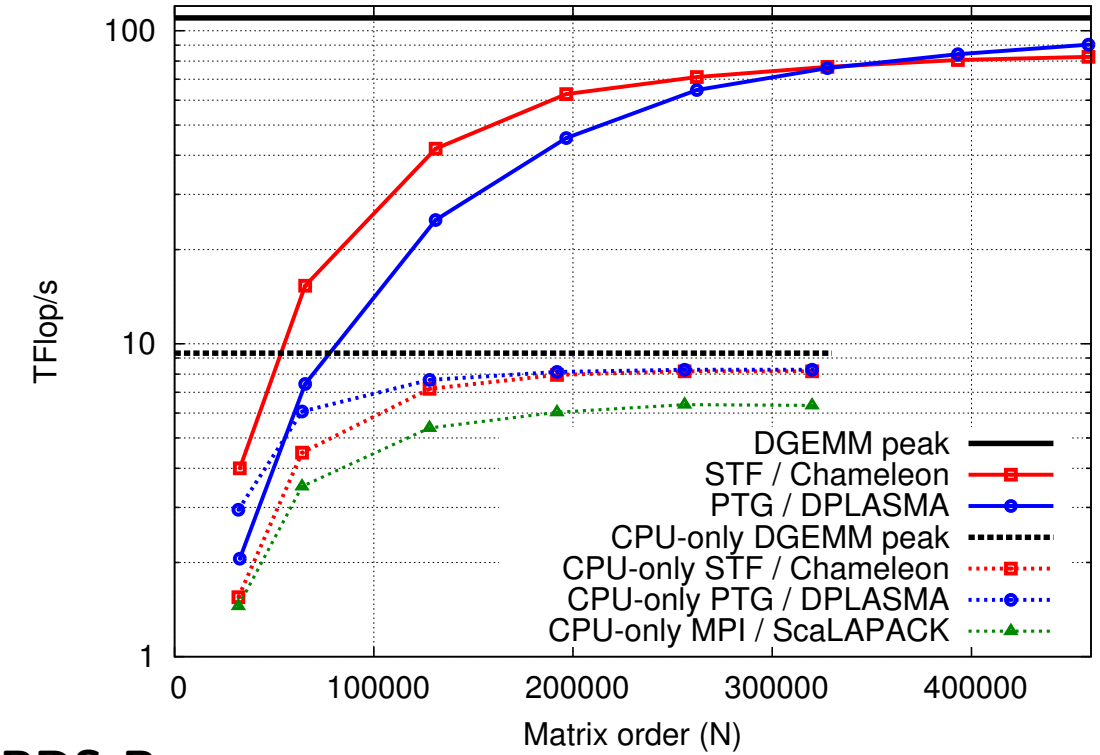
Task submission with pruning:

```
1  for(loop=0; loop<niter; loop++)
2      for (x = 1; x < X-1; x++)
3          for (y = 1; y < Y-1; y++)
4              if (my_distrib(x,y,size) == my_rank
5                  my_distrib(x-1,y,size) == my_rank
6                  my_distrib(x+1,y,size) == my_rank
7                  my_distrib(x,y-1,size) == my_rank
8                  my_distrib(x,y+1,size) == my_rank)
9                      starpu_mpi_task_insert(MPI_COMM_WORLD, &stencil5_cl,
10                                            STARPU_RW, data_handles[x][y],
11                                            STARPU_R, data_handles[x-1][y],
12                                            STARPU_R, data_handles[x+1][y],
13                                            STARPU_R, data_handles[x][y-1],
14                                            STARPU_R, data_handles[x][y+1],
15                                            0);
16 starpu_task_wait_for_all();
```

Distributed Scalability Study Results

Chameleon linear algebra library (Inria Team HiePACS)

- Heterogeneous cluster: 1152 CPU cores+288 GPUs



IEEE TPDS Paper:
DOI: 10.1109/TPDS.2017.2766064 — https://hal.inria.fr/hal-01618526

End of Part I...

StarPU runtime system

Web Site: https://starpu.gitlabpages.inria.fr/

LGPL License

Open to external contributors

StarPU, a Task-Based Runtime System

for Heterogeneous Platform Programming (2/2)

Olivier Aumage, Team STORM
Inria – LaBRI
olivier.aumage@inria.fr

STORM
Static Optimizations – Runtime Methods

Contents

1

Interoperability and Composition

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

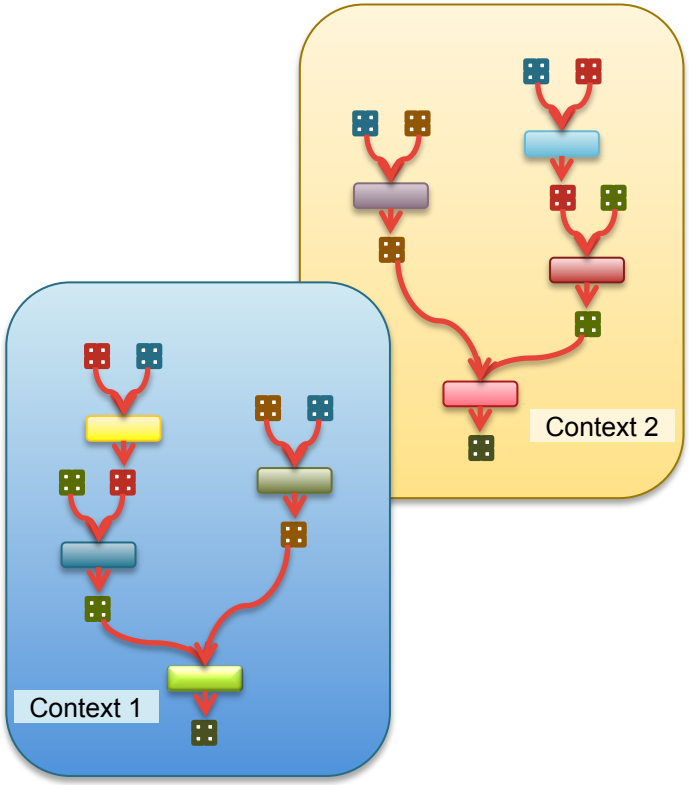
- Sharing computing resources...
- ... among multiple DAGs
- ... simultaneously

Composing Multiple Codes

Rationale

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

Scheduling Contexts



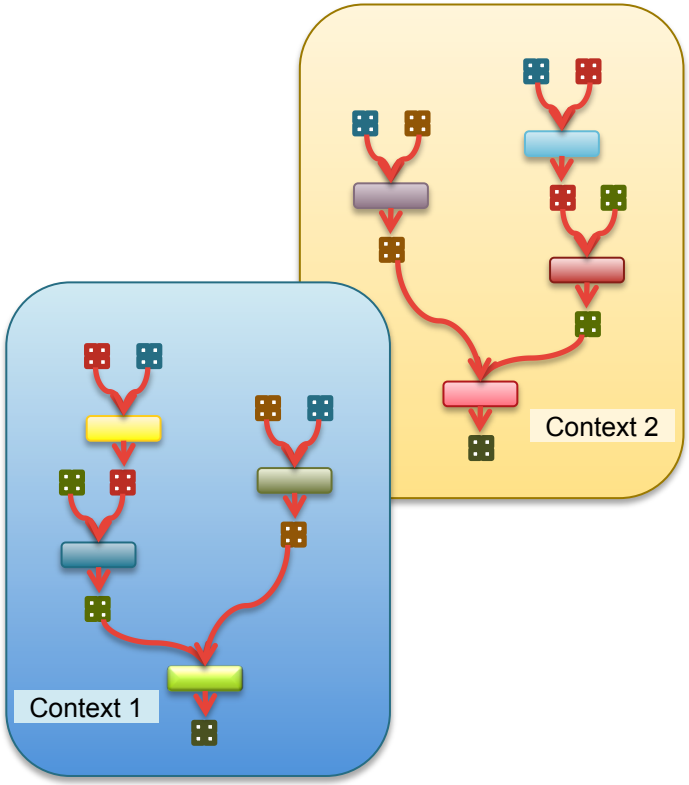
Composing Multiple Codes

Rationale

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

Scheduling Contexts

- Map DAGs on subsets of computing units



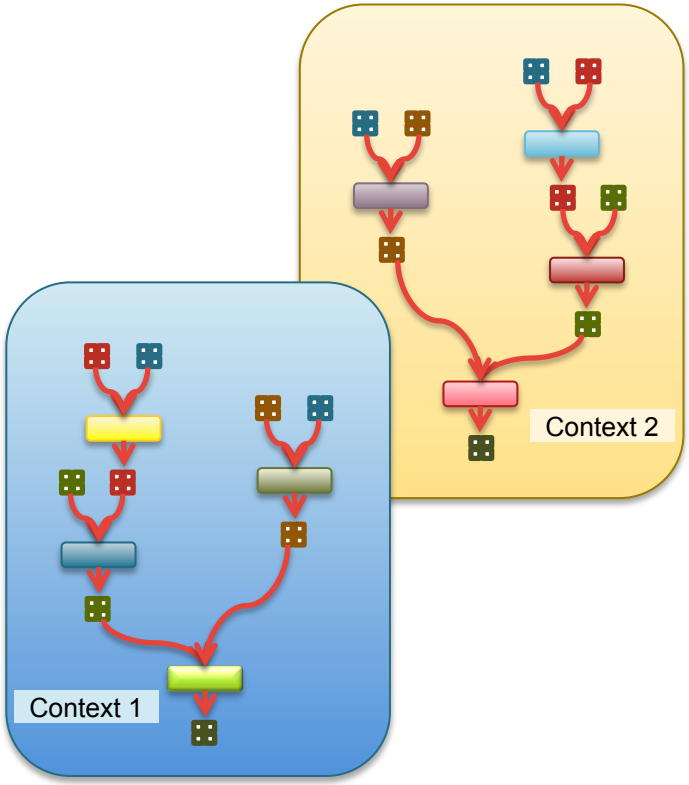
Composing Multiple Codes

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.



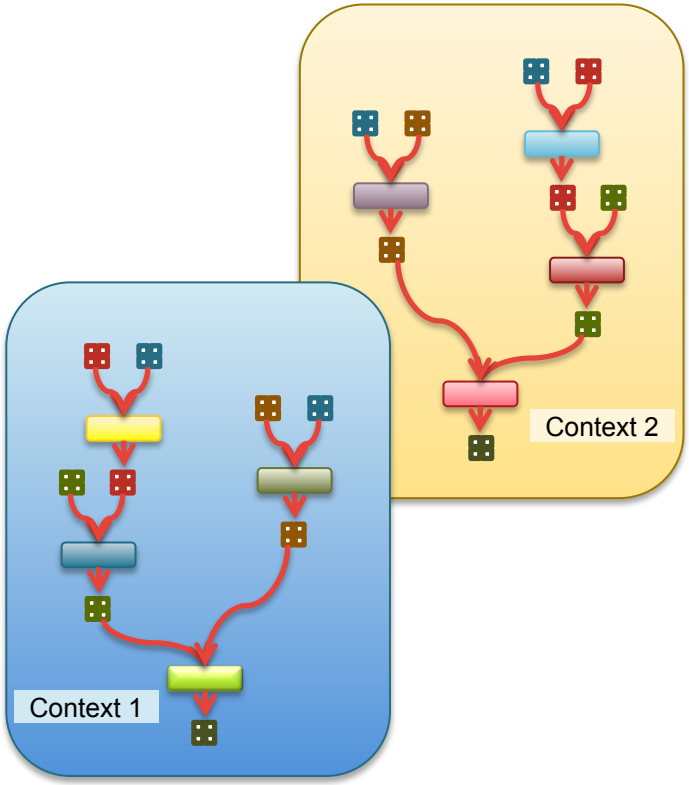
Composing Multiple Codes

Rationale

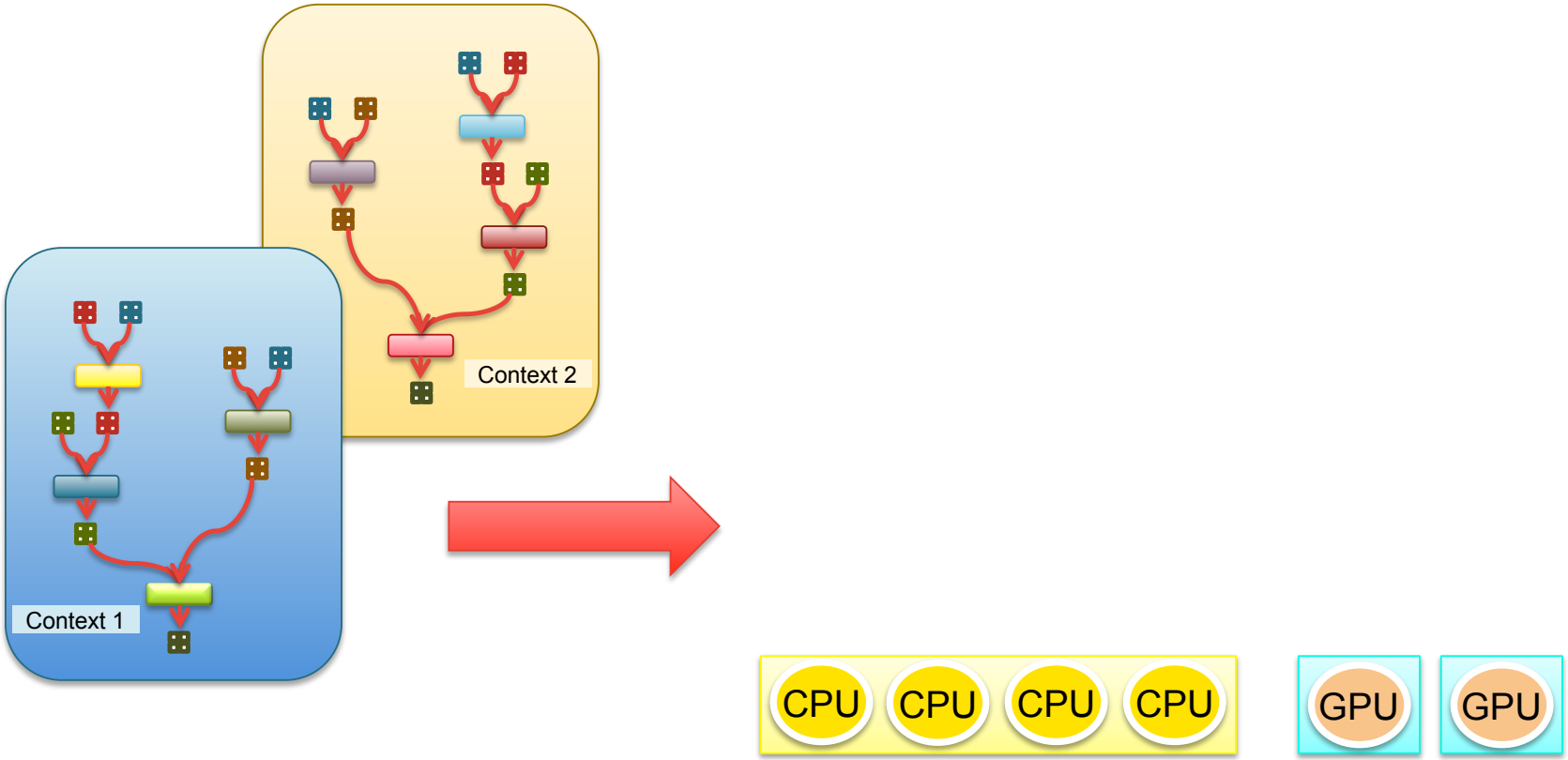
- Sharing computing resources. . .
- . . . among multiple DAGs
- . . . simultaneously

Scheduling Contexts

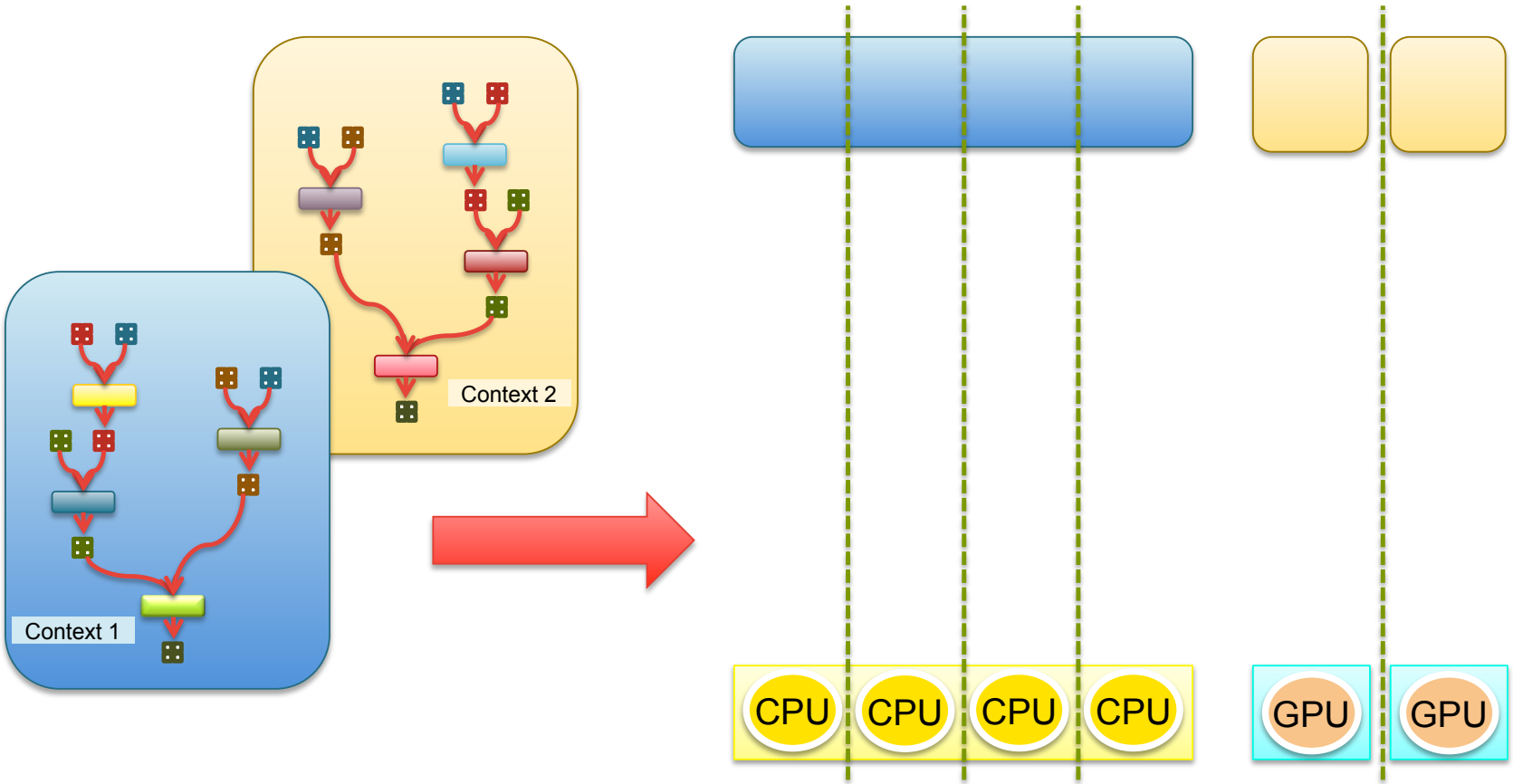
- Map DAGs on subsets of computing units
- Isolate competing kernels or library calls
  - OpenMP kernel, Intel MKL, etc.
- Select scheduling policy per context



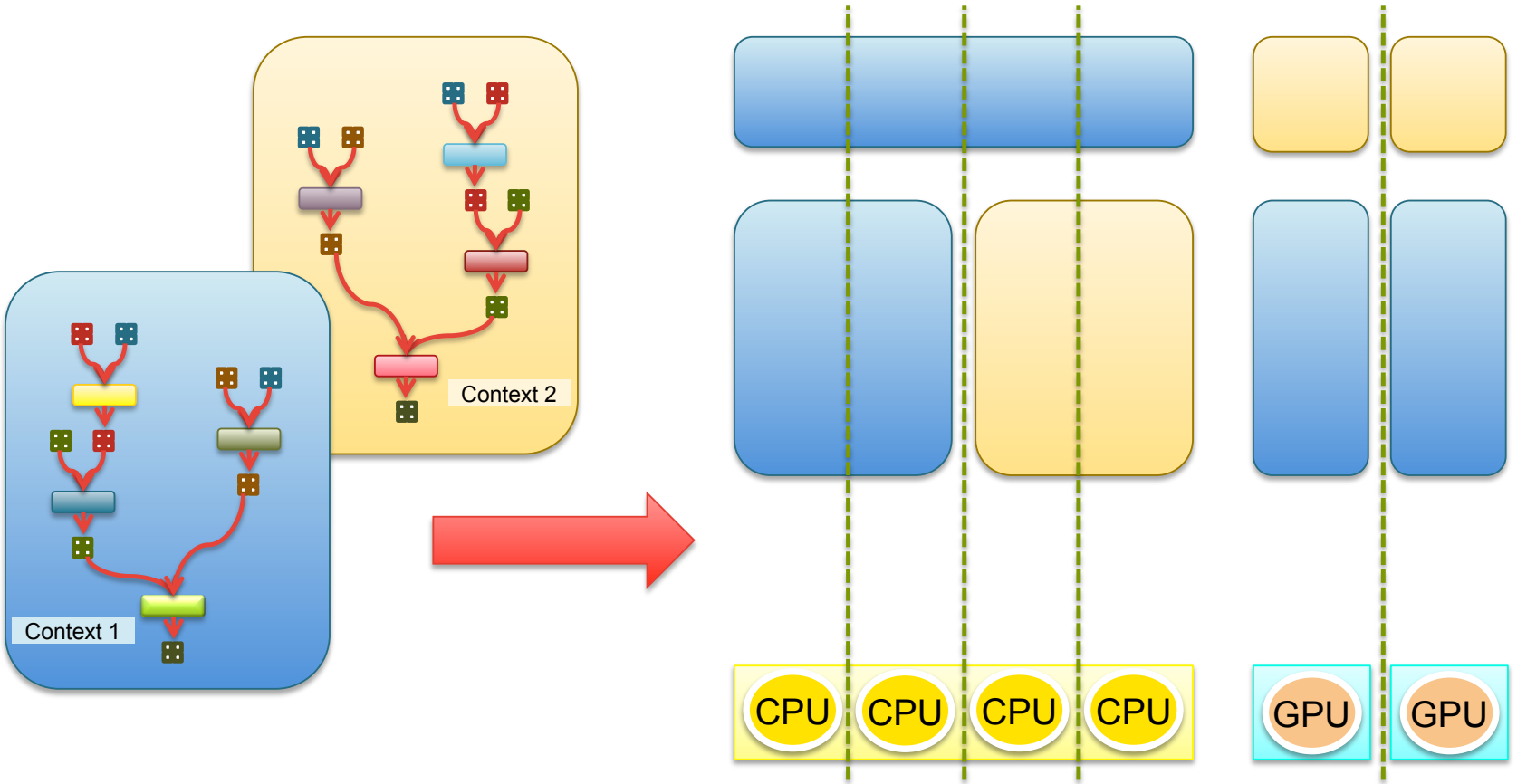
Contexts: Dynamic Resource Management



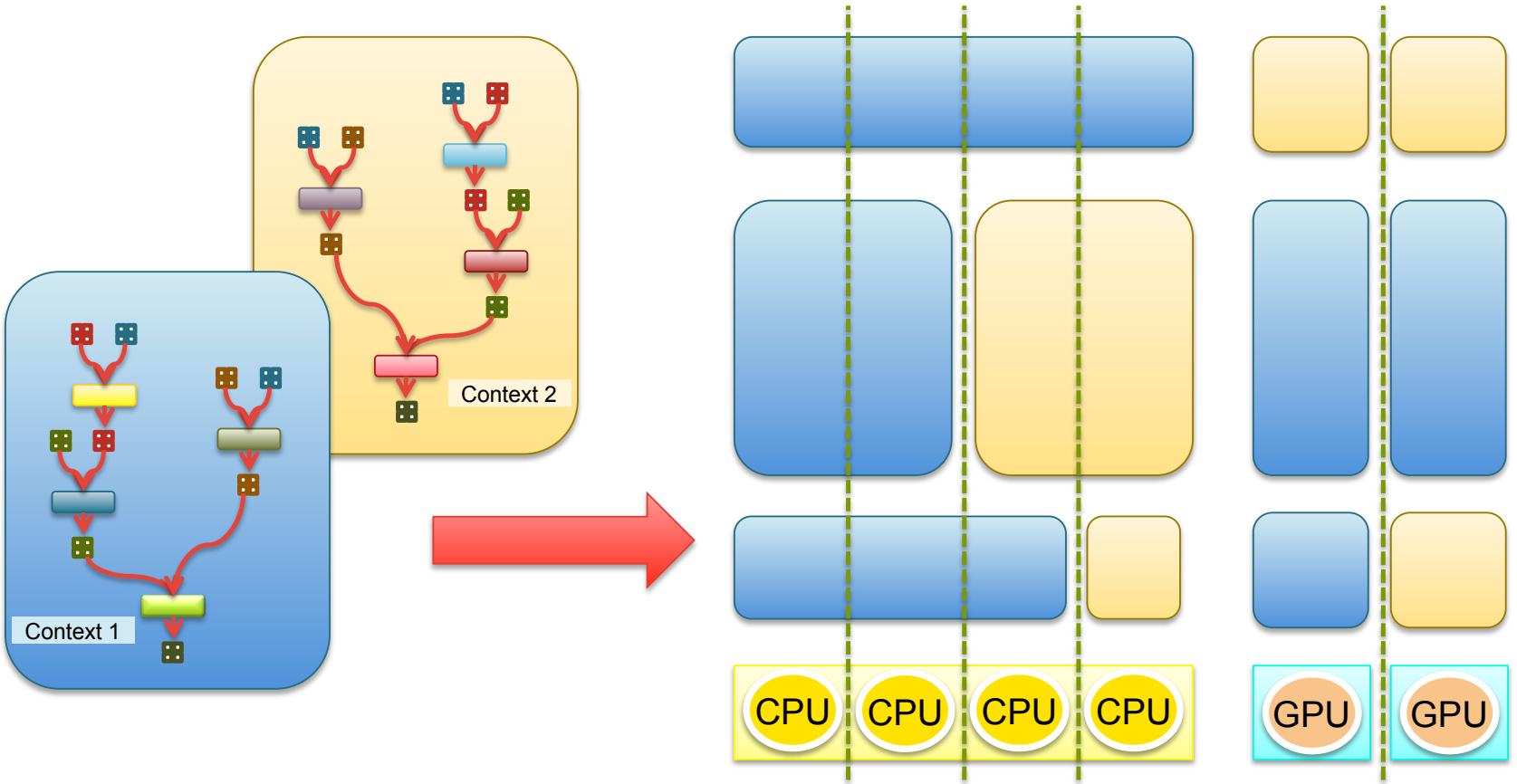
Contexts: Dynamic Resource Management



Contexts: Dynamic Resource Management



Contexts: Dynamic Resource Management



Interoperability

How to Make Runtimes, Libs Cooperate?

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

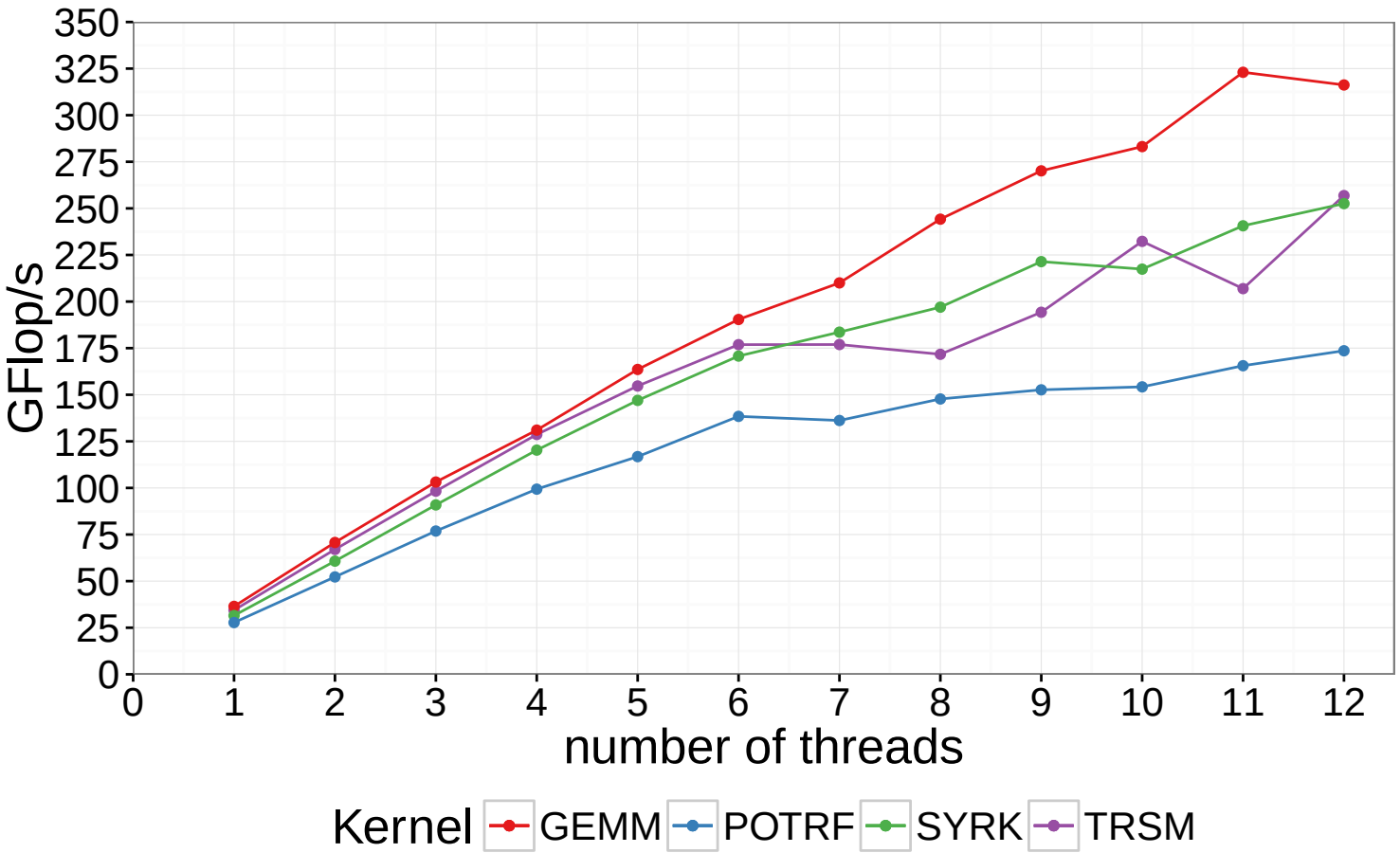
2

Advanced Scheduling Topics

Multicore CPUs: Parallel Tasks

Multicore CPUs: Parallel Tasks (T. Cojean)

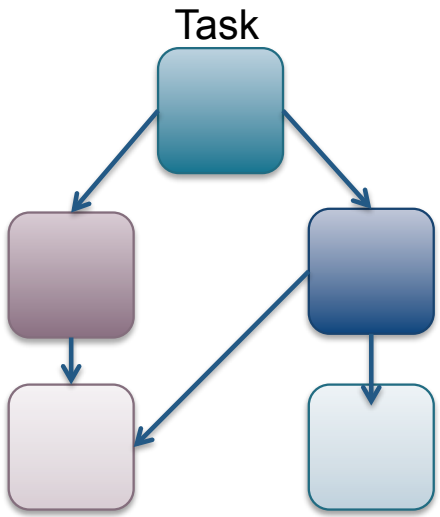
Kernel sweet spots: example with Cholesky factorization kernels
(1x Xeon E5-2680v3 2.5GHz 12 cores)



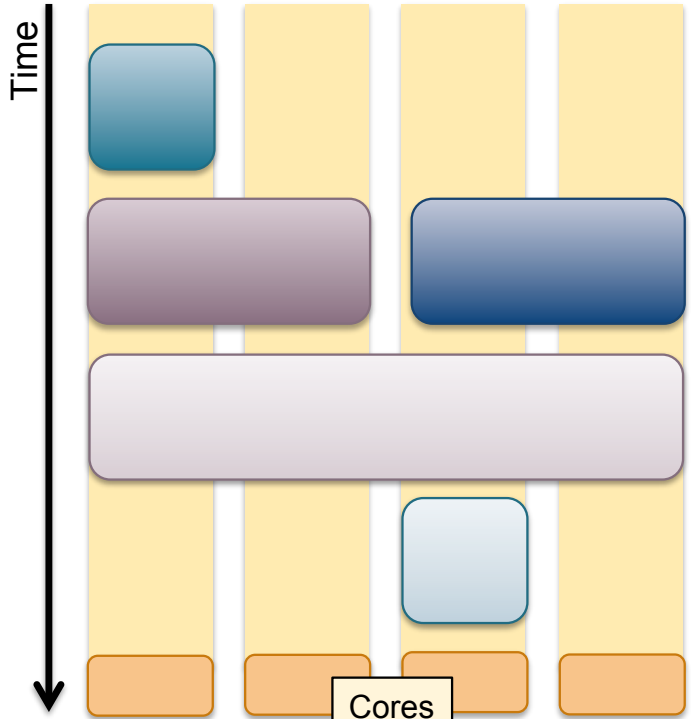
Multicore CPUs: Parallel Tasks

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

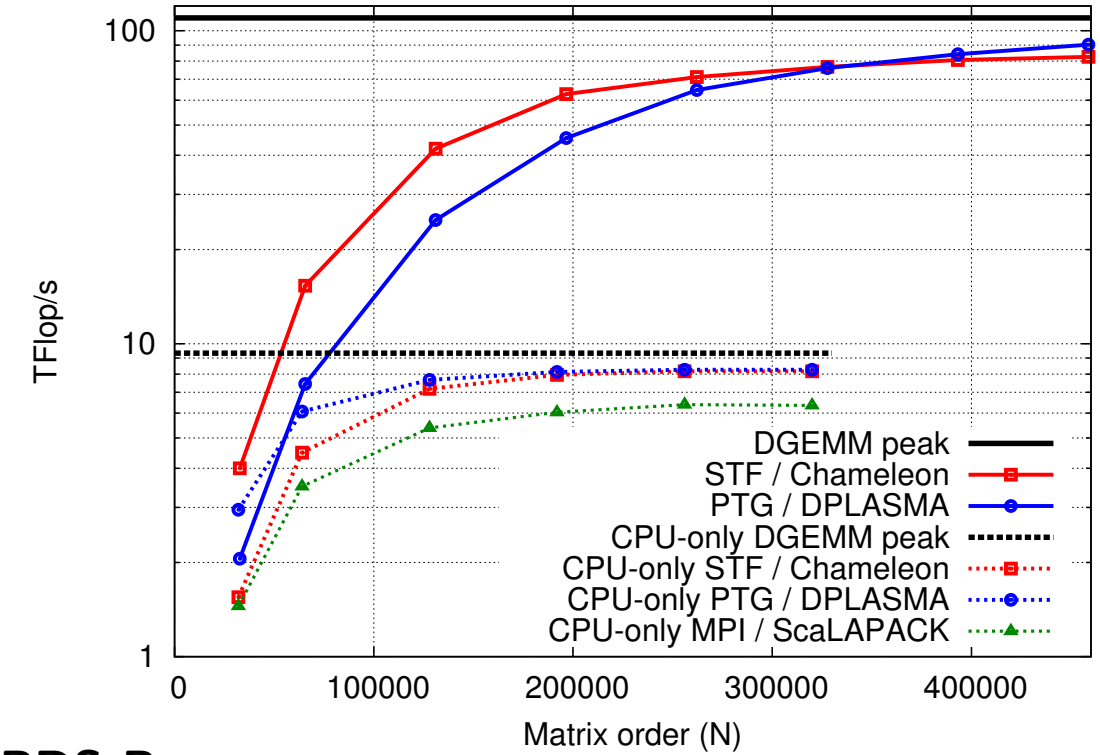
Submission-side Task Flow Optimizations

- Global task-graph pruning in distributed computing sessions
- Memory subscription control

Distributed Scalability Study Results

Chameleon linear algebra library (Inria Team HiePACS)

- Heterogeneous cluster: 1152 CPU cores+288 GPUs



IEEE TPDS Paper:
DOI: 10.1109/TPDS.2017.2766064 — https://hal.inria.fr/hal-01618526

Distributed Support

Sequential Task Flow Paradigm on Clusters

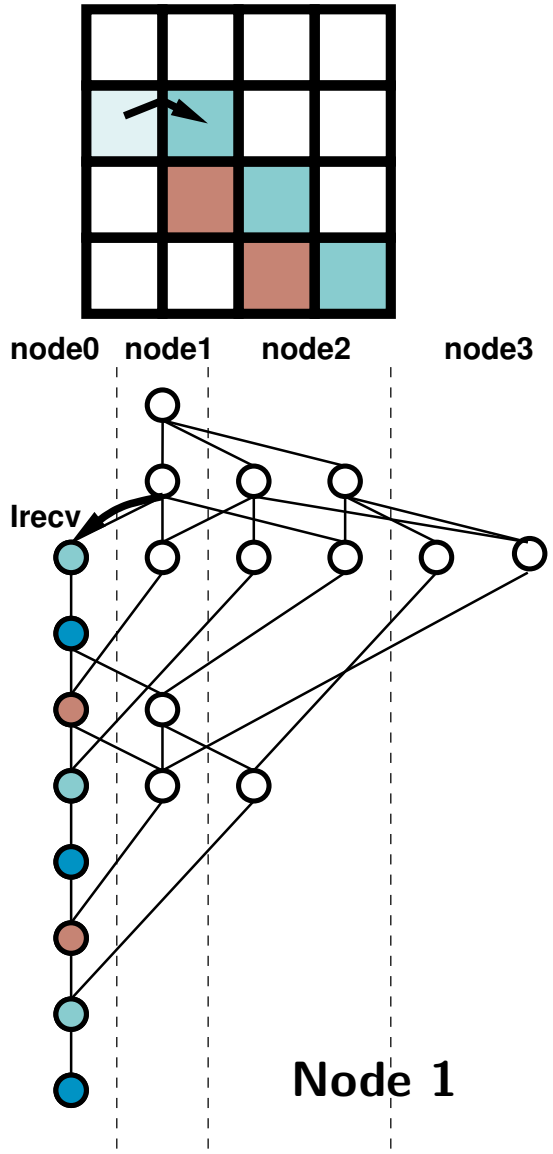
Each node unrolls the sequential task flow

Task↔Node Mapping

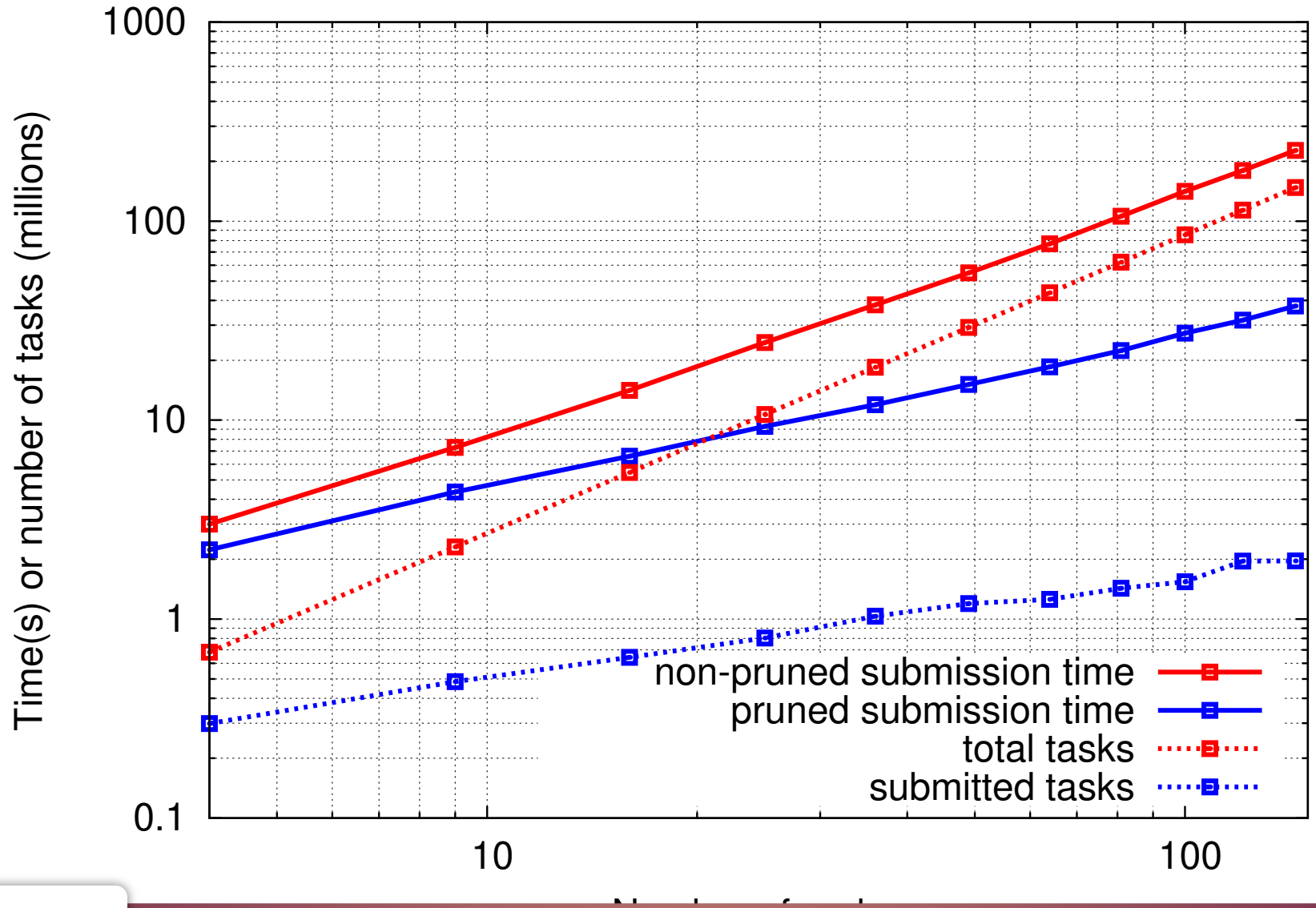
- Inferred from data location:
  - *Tasks move to data they modify*
- No global scheduling
- No synchronizations

Optimization

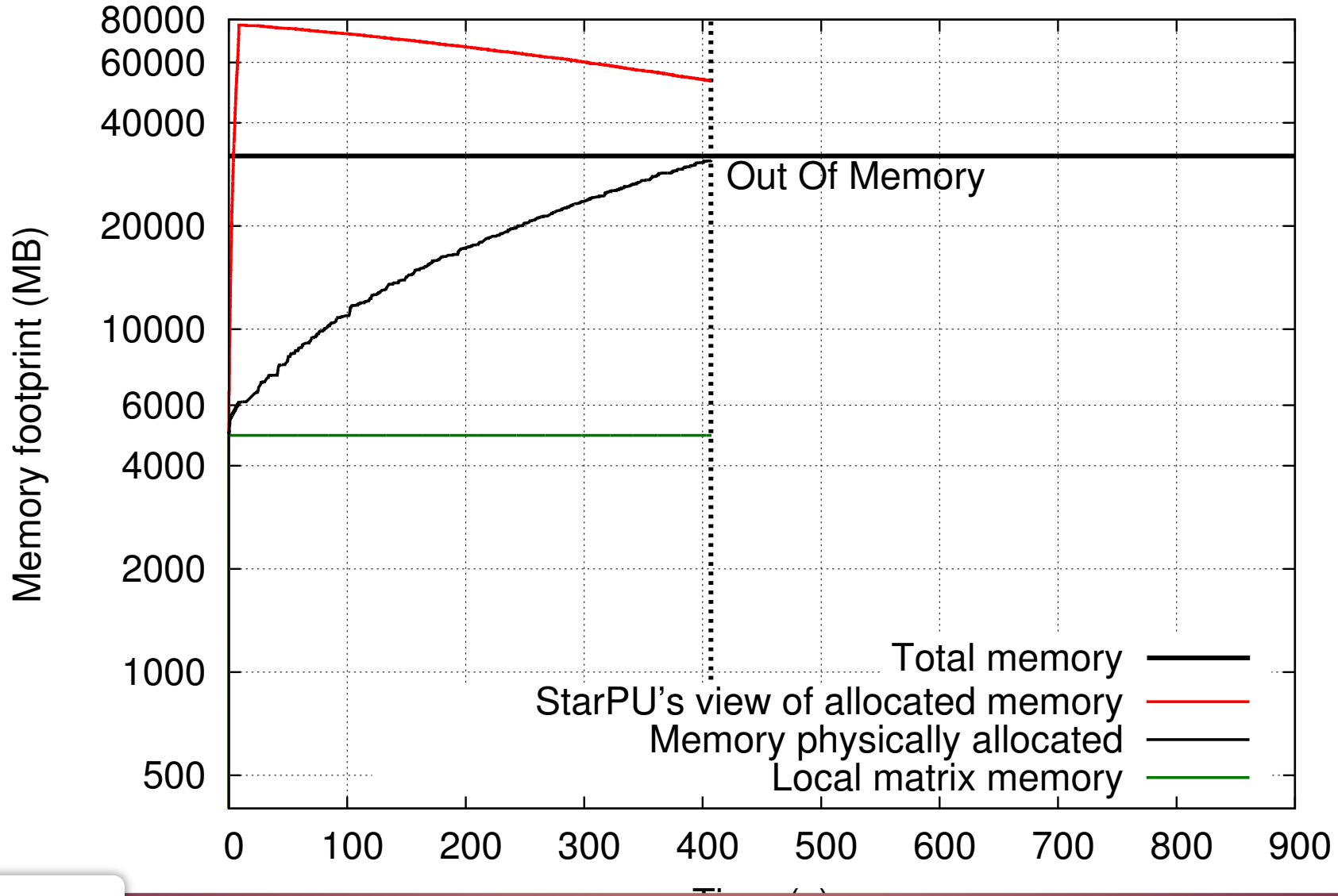
- **Local DAG pruning**



Global Task-Graph Pruning Issue



Unbounded Task Submission Issue

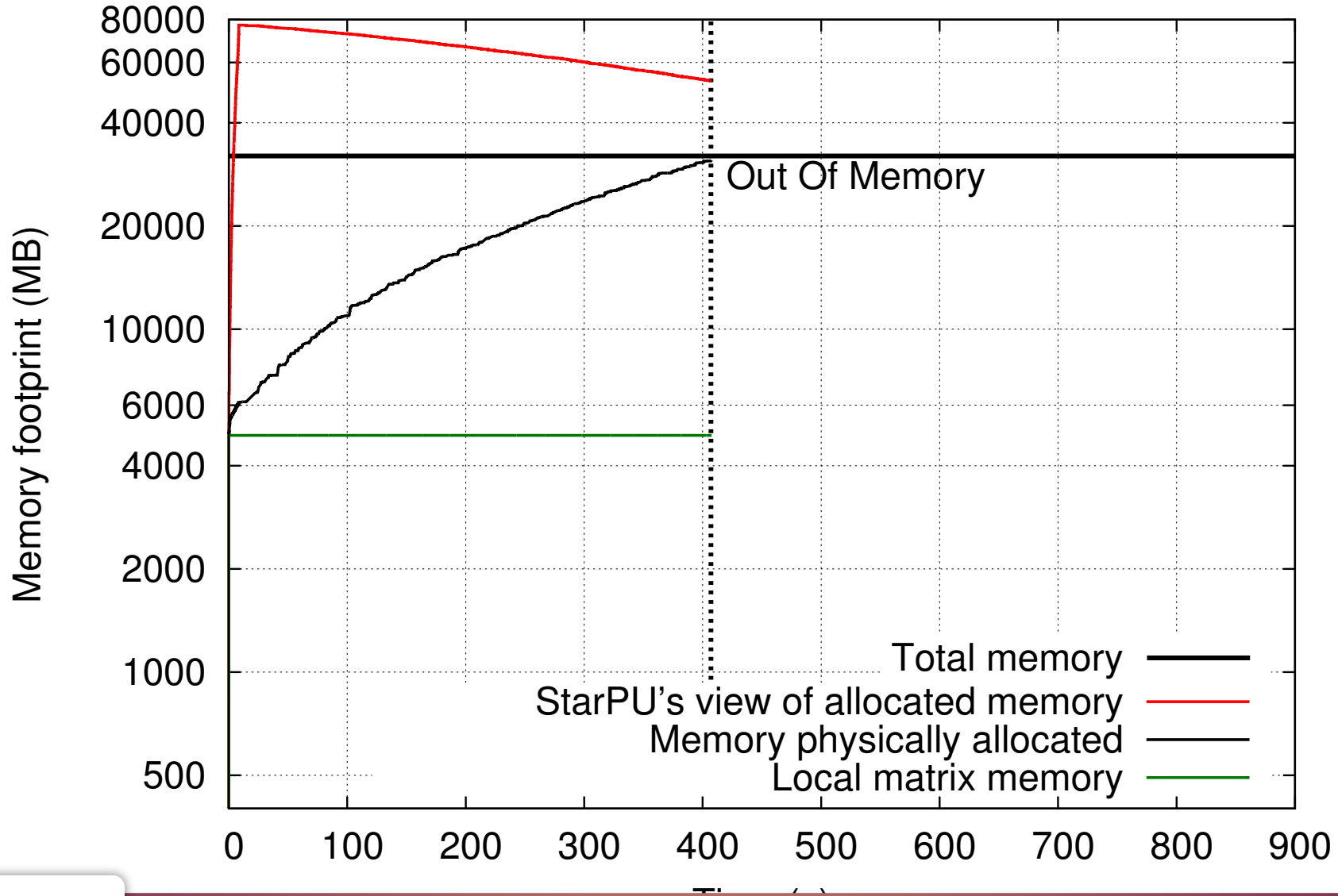


Implementing Some Scheduling Lookahead Window

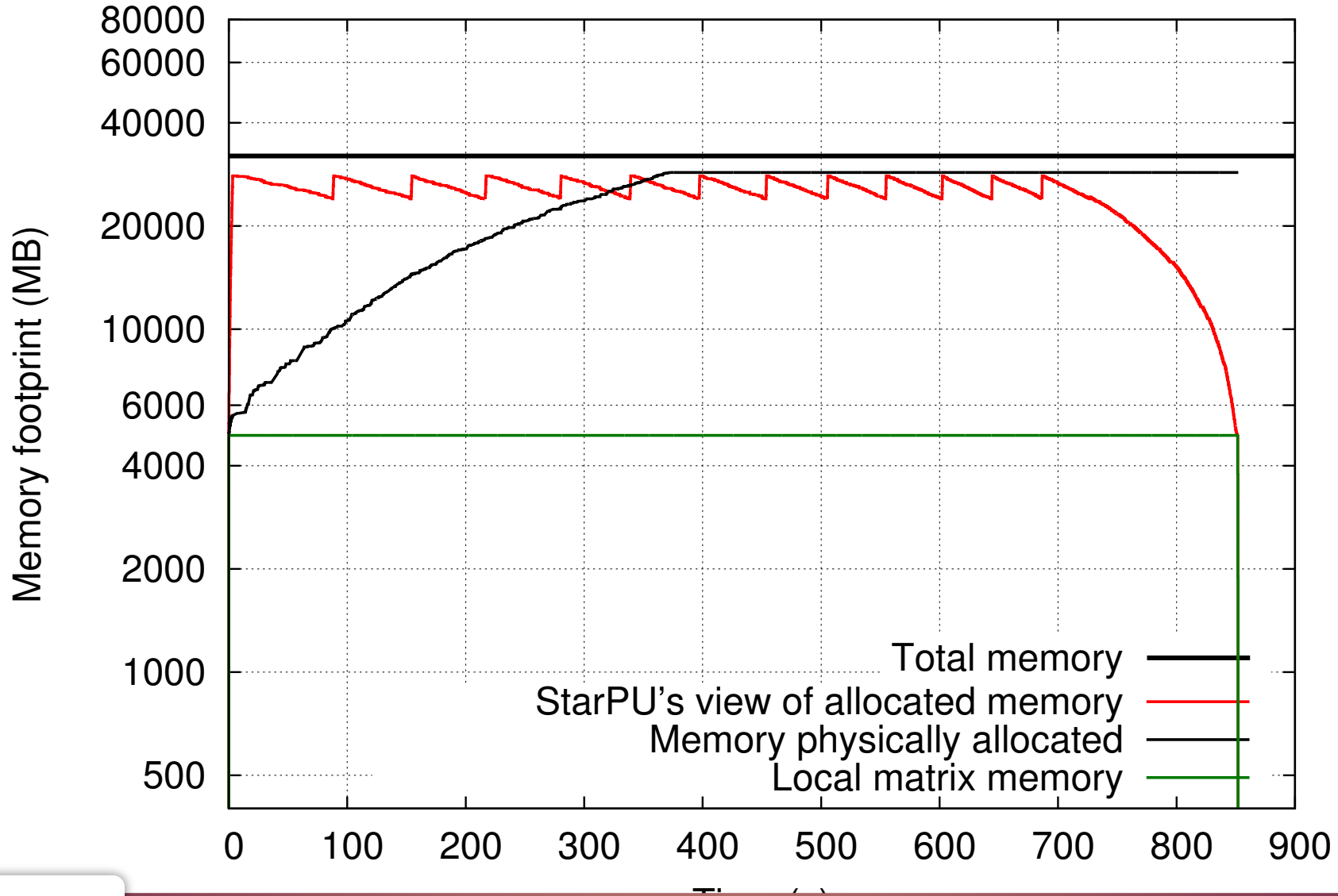
Control of the task submission flow

- **Memory tracking**
  - Account the memory subscription
- **Task submission throttling**
  - Blocking mechanism of the task submission flow
  - Allows the task submission to be controlled by an external criteria
- A **control policy** which uses the **memory tracking** to **throttle the task submission flow**

Memory Behaviour Without Memory Control



Memory Behaviour With Memory Control



3

Advanced Data Management Topics

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

```
1  /* CPU Computation Kernel */
2
3  void
4  multiformat_scal_cpu_func(void *buffers[], void *cl_arg) {
5      struct point *aos;
6      unsigned int n;
7
8      aos = STARPU_MULTIFORMAT_GET_CPU_PTR(buffers[0]);
9      n = STARPU_MULTIFORMAT_GET_NX(buffers[0]);
10     ...
11 }
```

Multiformat

Example

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

```
1  /* GPU Computation Kernel */
2
3  extern "C" void
4  multiformat_scal_cuda_func(void *buffers[],void *cl_arg) {
5      unsigned int n;
6      struct struct_of_arrays *soa;
7
8      soa = (struct struct_of_arrays *)
9                  STARPU_MULTIFORMAT_GET_CUDA_PTR(buffers[0]);
10     n = STARPU_MULTIFORMAT_GET_NX(buffers[0]);
11
12     ...
13 }
```

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

Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance

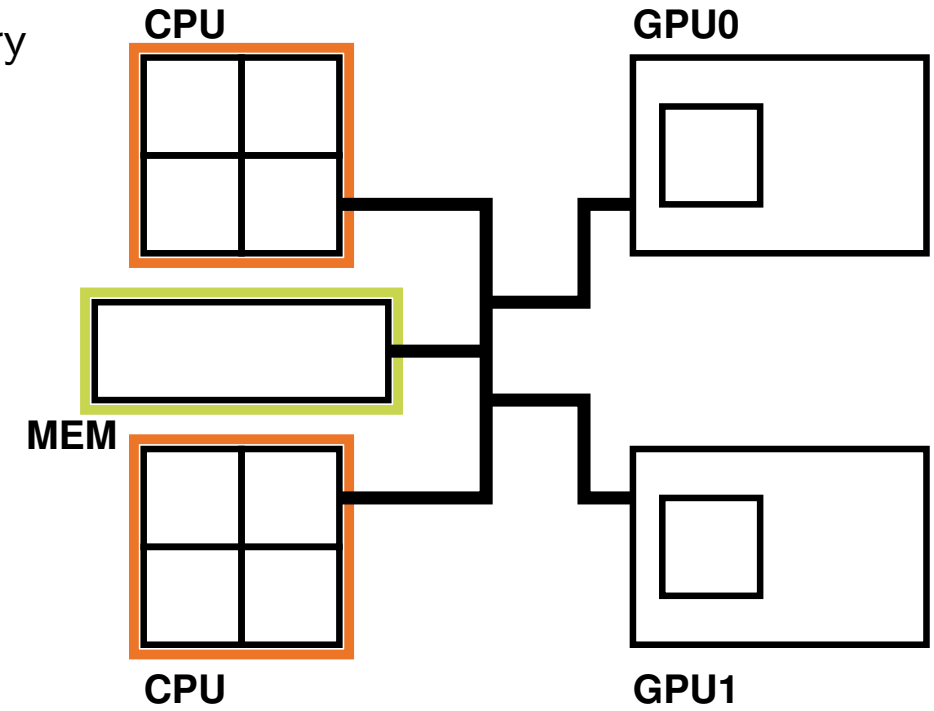
Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance



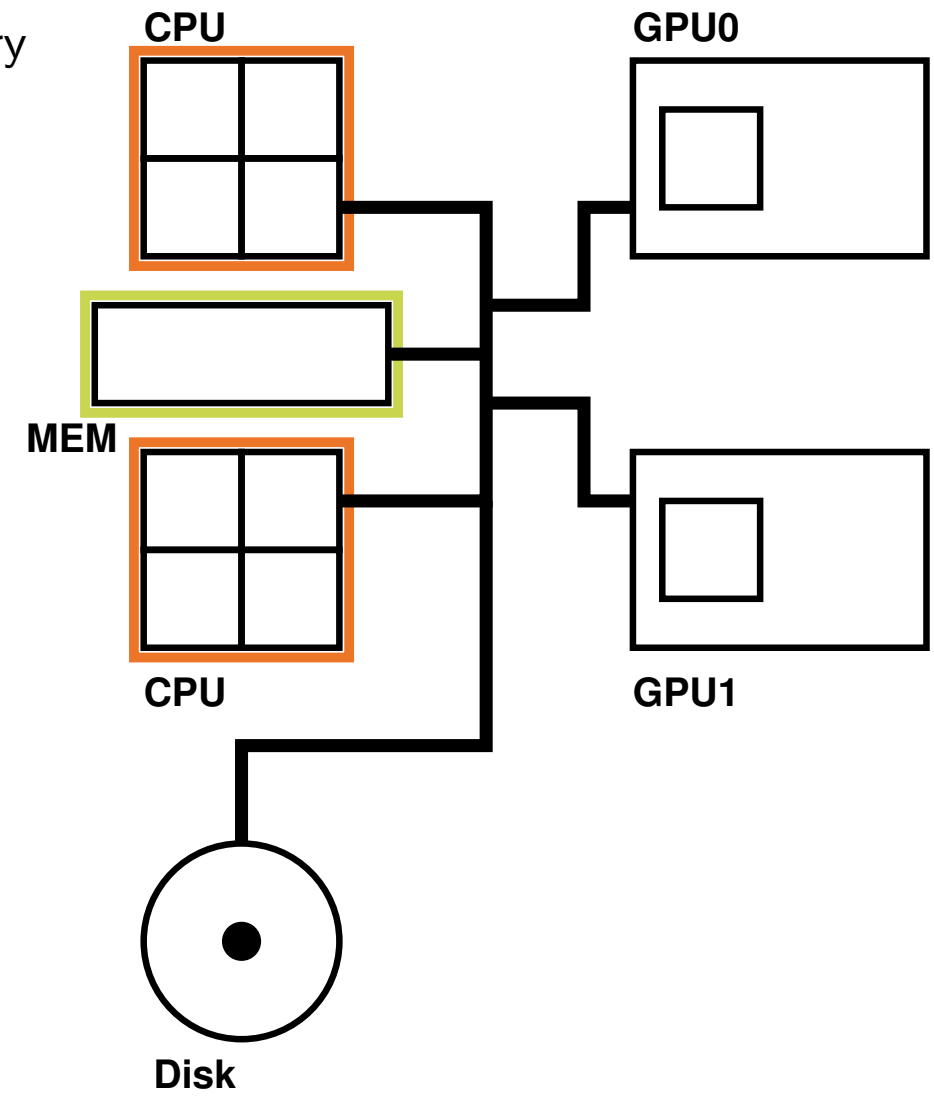
Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance



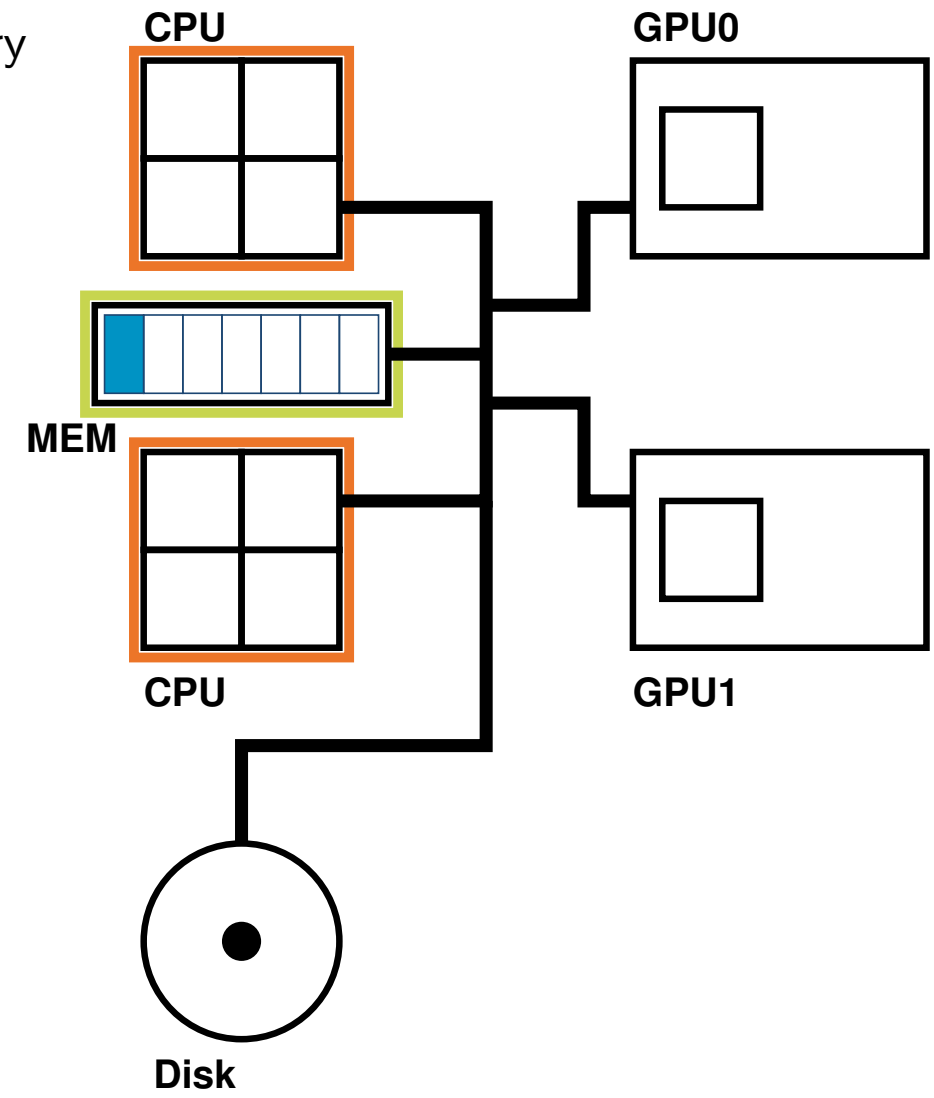
Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance



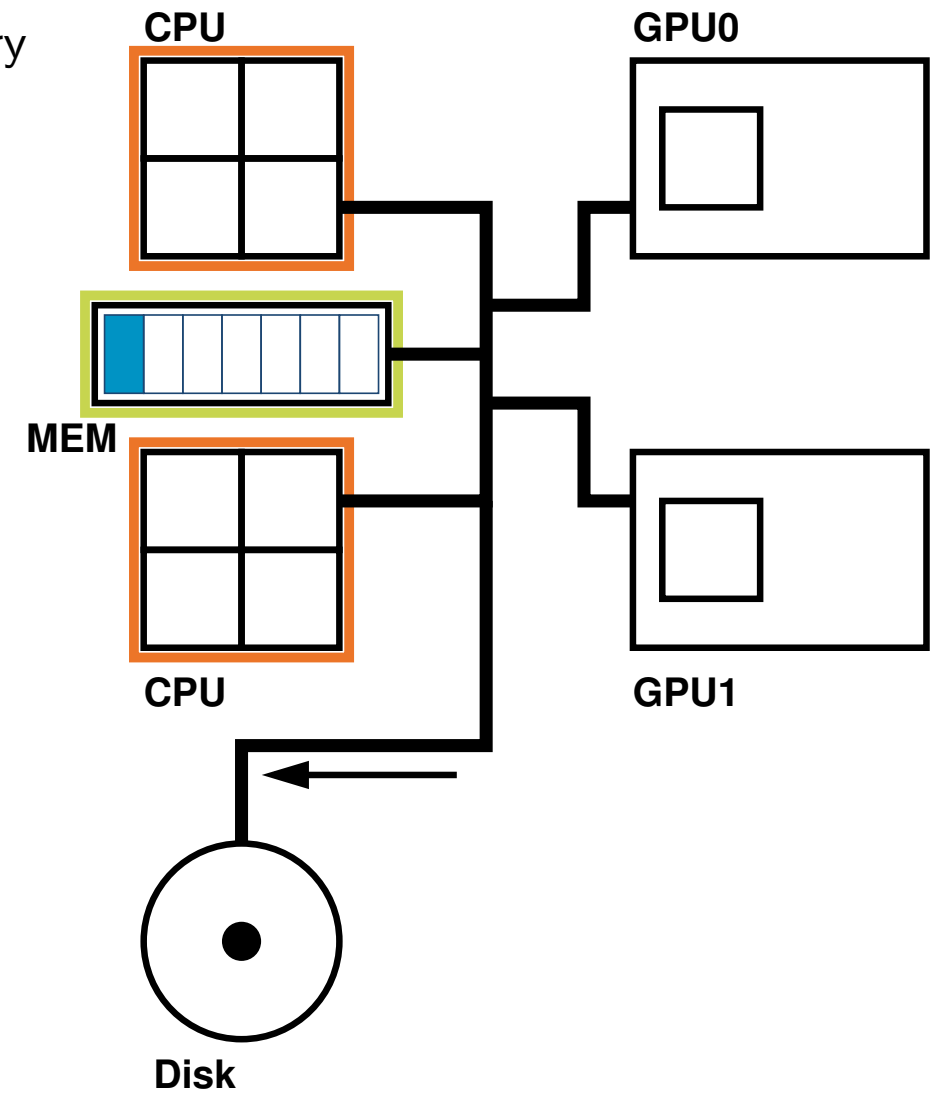
Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance



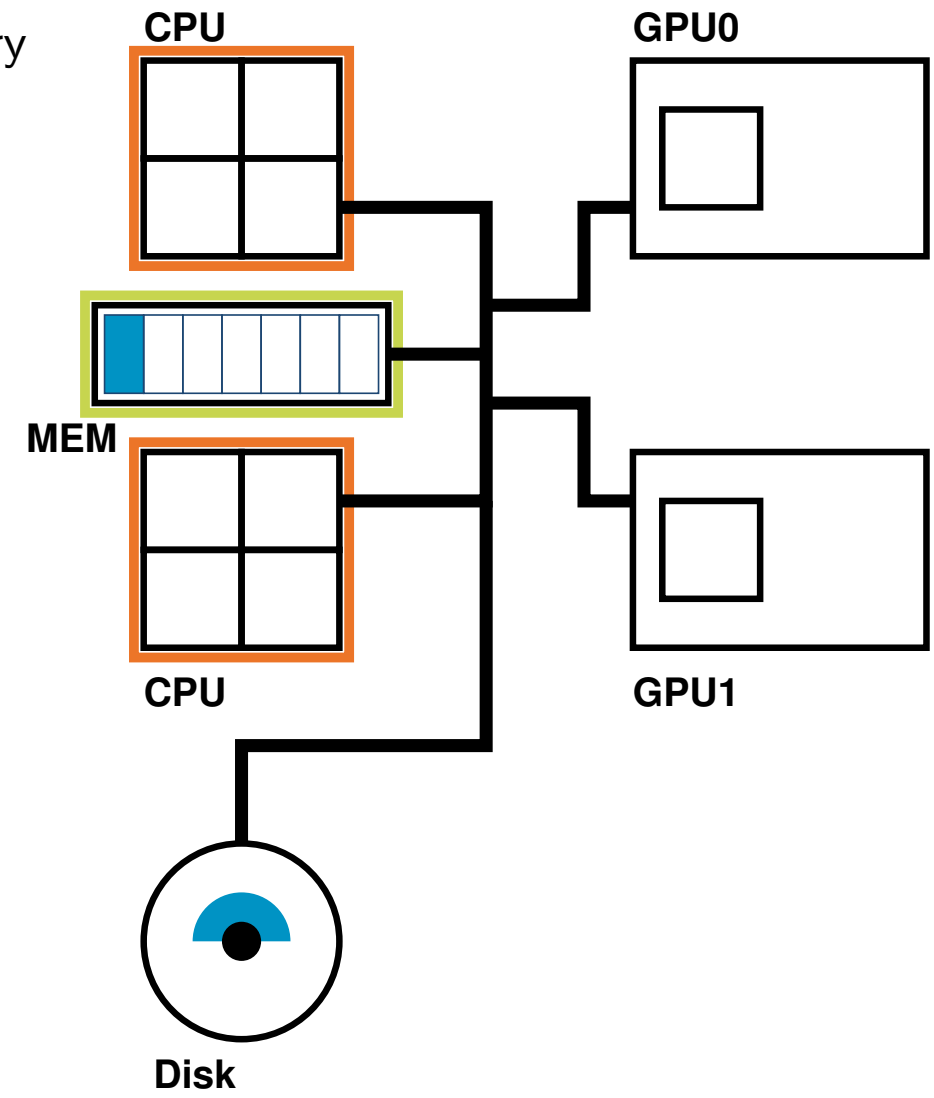
Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance



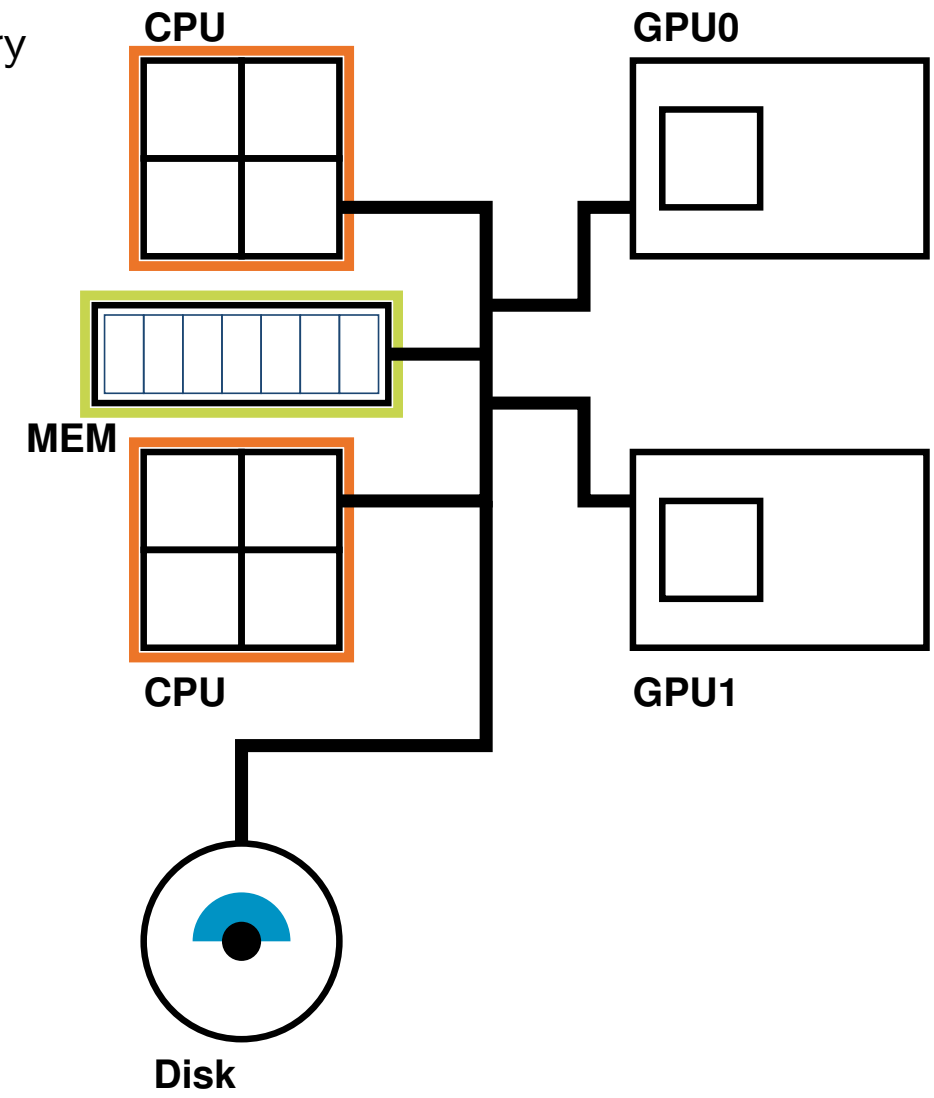
Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance



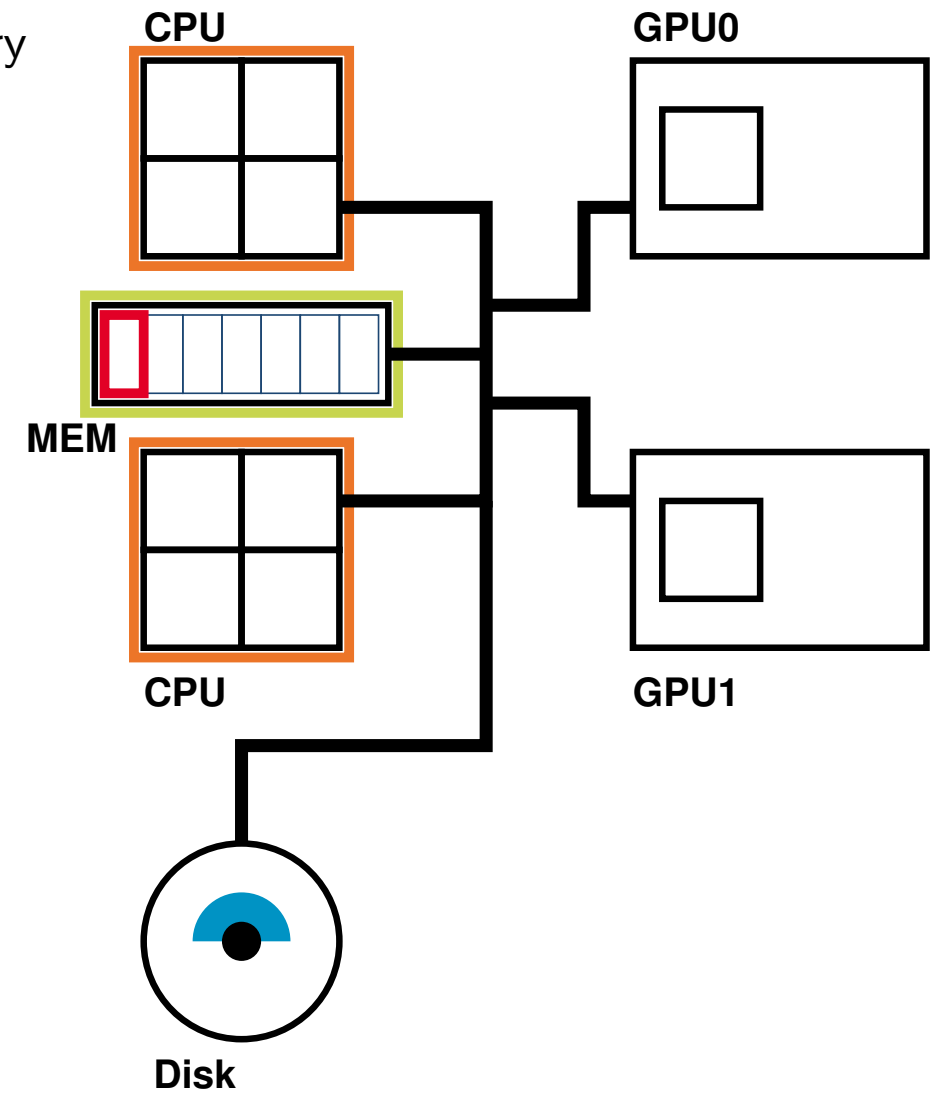
Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance



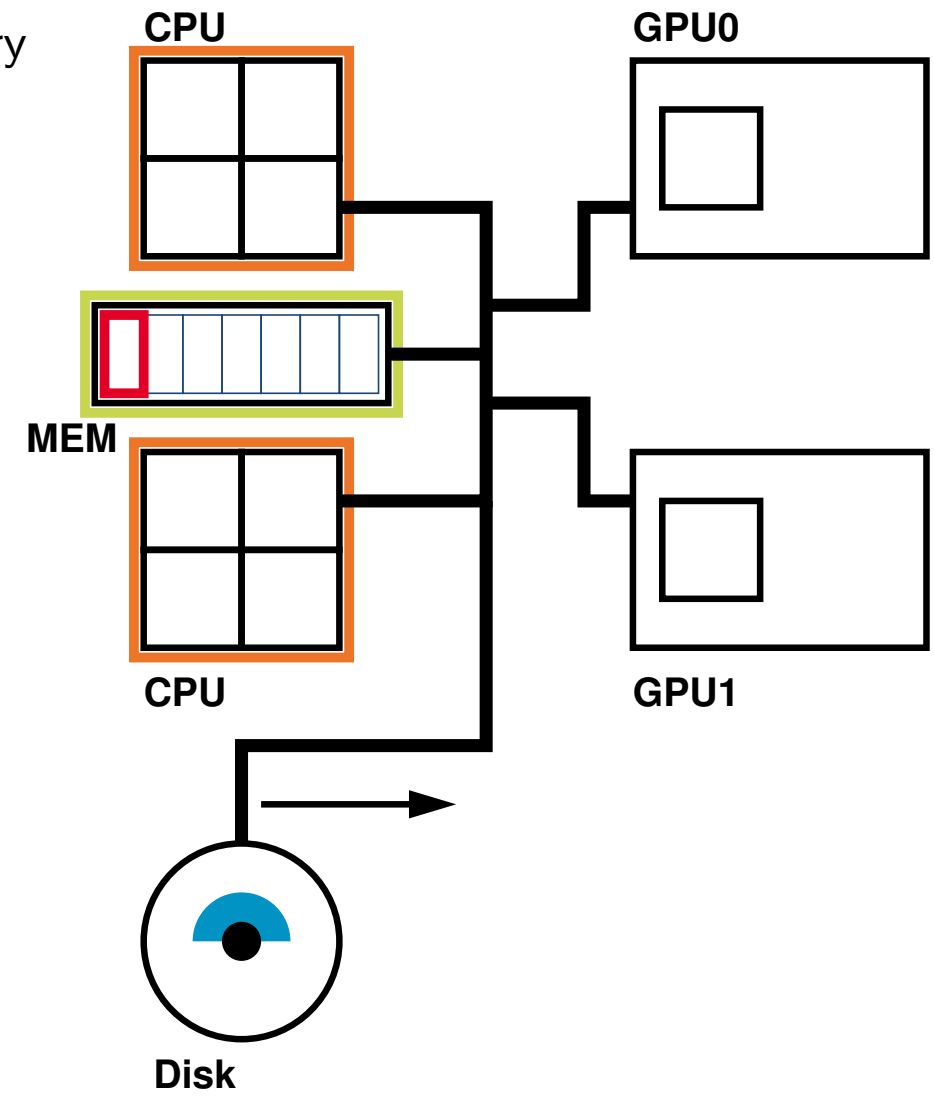
Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance



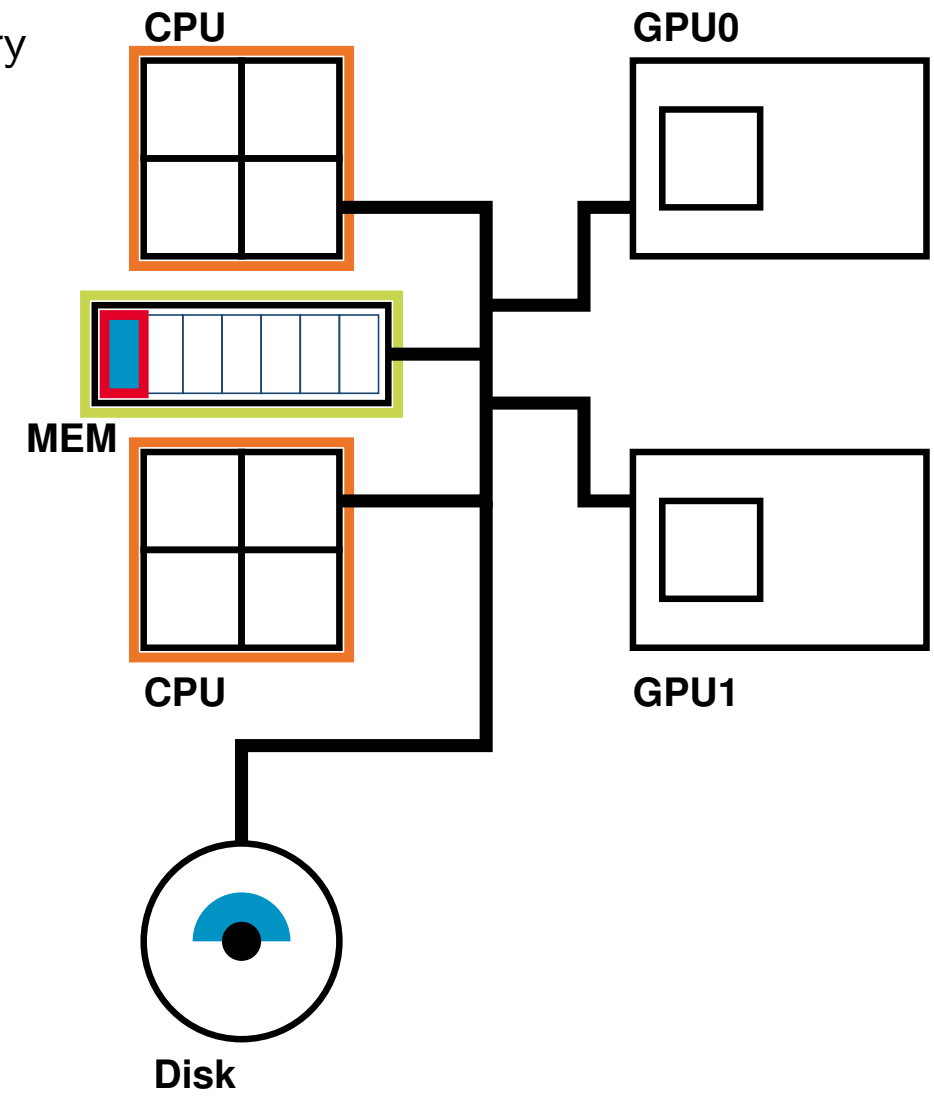
Input/Output Support

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
- Multiple I/O drivers supported

Many possible use scenarios

- **Out-of-core** / swap
- Mitigated startup load / solution output
- Building block for fault tolerance



4

Advanced Analysis and Monitoring Topics

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

```
1  int ret = starpu_init(NULL);
2  ...
3
4  starpu_task_insert(...);
5  starpu_task_insert(...);
6  ...
7  starpu_task_wait_for_all();
8
9
10 ...
```

Computing the Theoretical Lower Bound...

... on Execution Time

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

```
1  int ret = starpu_init(NULL);
2  ...
3  starpu_bound_start();
4  starpu_task_insert(...);
5  starpu_task_insert(...);
6  ...
7  starpu_task_wait_for_all();
8  starpu_bound_stop();
9
10 ...
```

Computing the Theoretical Lower Bound...

... on Execution Time

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

```
1  int ret = starpu_init(NULL);
2  ...
3  starpu_bound_start();
4  starpu_task_insert(...);
5  starpu_task_insert(...);
6  ...
7  starpu_task_wait_for_all();
8  starpu_bound_stop();
9  starpu_bound_print_lp();
10 ...
```

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.)

```
1  int ret = starpu_init(NULL);
2  ...
3  starpu_bound_start();
4  starpu_task_insert(...);
5  starpu_task_insert(...);
6  ...
7  starpu_task_wait_for_all();
8  starpu_bound_stop();
9  starpu_bound_print_lp();
10 ...
```

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

```
1  int ret = starpu_init(NULL);
2  ...
3
4  starpu_task_insert(...);
5  starpu_task_insert(...);
6  ...
7  starpu_task_wait_for_all();
8
9
10 ...
```

Computing the Theoretical Lower Bound...

... on Execution Time

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

```
1  int ret = starpu_init(NULL);
2  ...
3  starpu_bound_start();
4  starpu_task_insert(...);
5  starpu_task_insert(...);
6  ...
7  starpu_task_wait_for_all();
8  starpu_bound_stop();
9
10 ...
```

Computing the Theoretical Lower Bound...

... on Execution Time

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

```
1  int ret = starpu_init(NULL);
2  ...
3  starpu_bound_start();
4  starpu_task_insert(...);
5  starpu_task_insert(...);
6  ...
7  starpu_task_wait_for_all();
8  starpu_bound_stop();
9  starpu_bound_print_lp();
10 ...
```

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.)

```
1  int ret = starpu_init(NULL);
2  ...
3  starpu_bound_start();
4  starpu_task_insert(...);
5  starpu_task_insert(...);
6  ...
7  starpu_task_wait_for_all();
8  starpu_bound_stop();
9  starpu_bound_print_lp();
10 ...
```

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

```
1 $ $STARPU_DIR/configure --enable-simgrid [... other opts ...]
2 ...
```

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

```
1 $ $STARPU_DIR/configure --enable-simgrid [... other opts ...]
2 ...
```

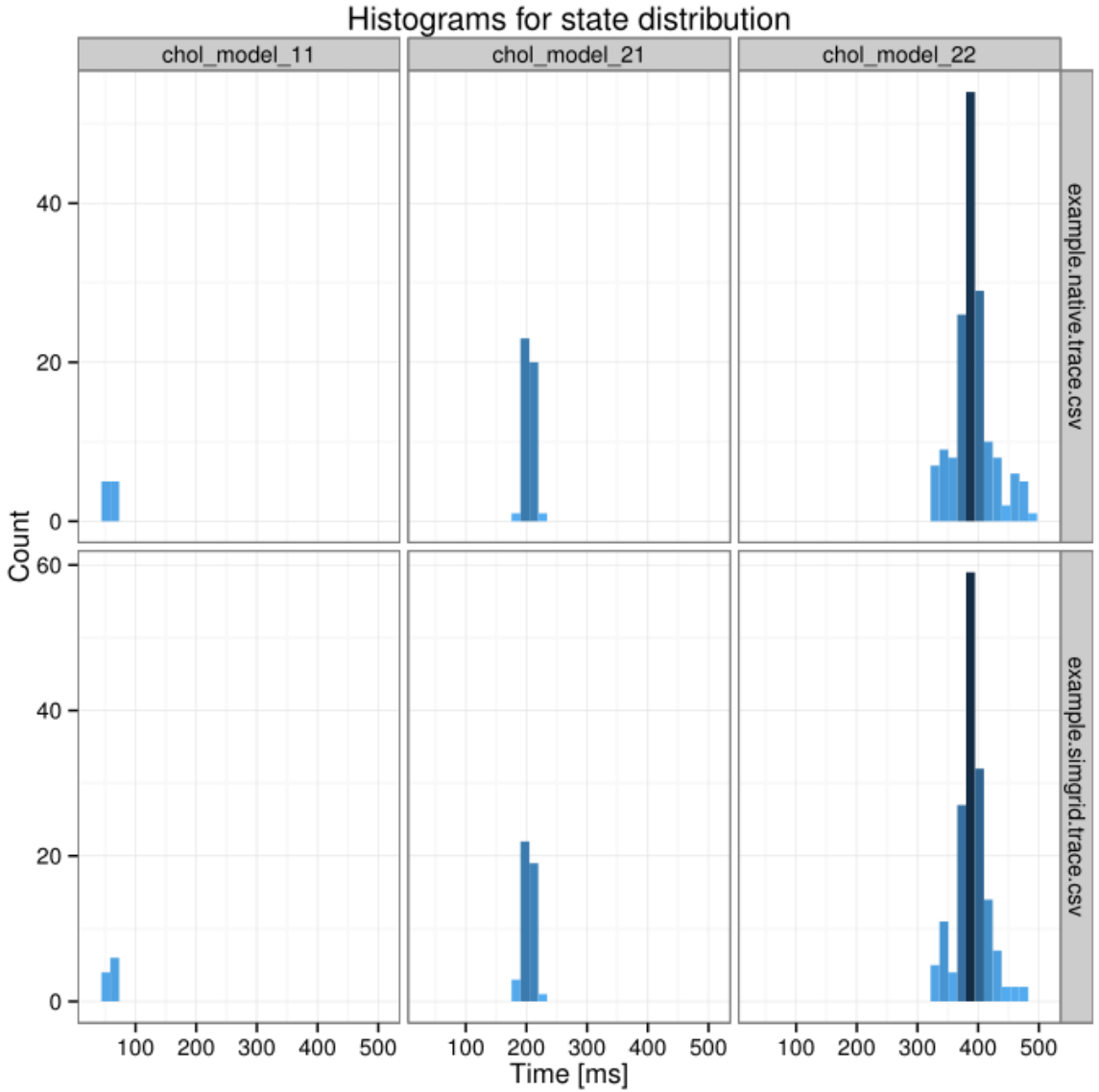
Simulation with SimGrid

Scheduling without executing kernels

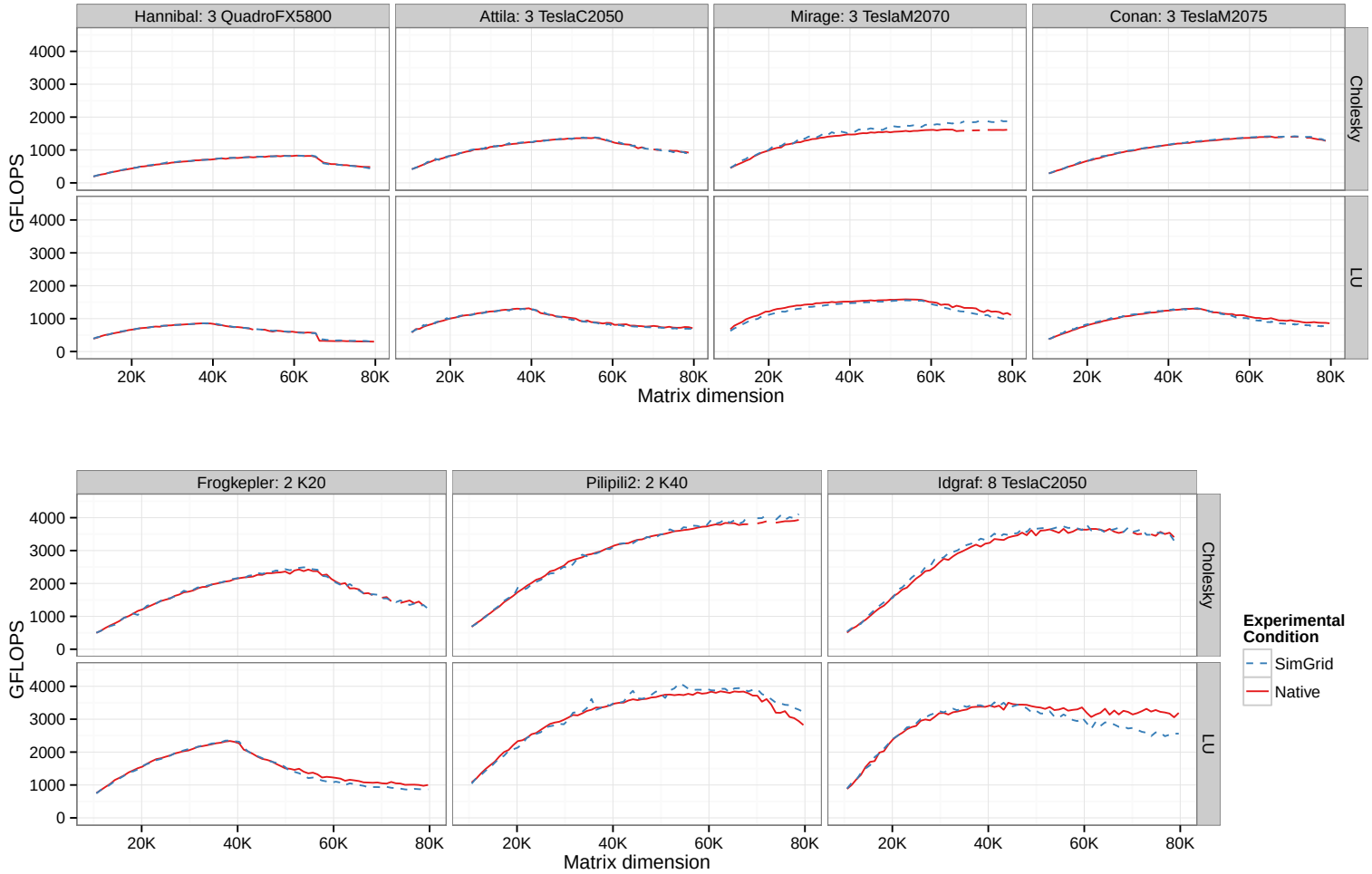
- Requires the SimGrid simulation environment
- Enables simulating large-scale scenarios
  - Large data sets
  - Large simulated hardware plaform
- Relies on **real** performance models...
- ...collected by StarPU on a real machine
- Enables fast experiments when designing application algorithms
- Enables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

```
1 $ $STARPU_DIR/configure --enable-simgrid [... other opts ...]
2 ...
```

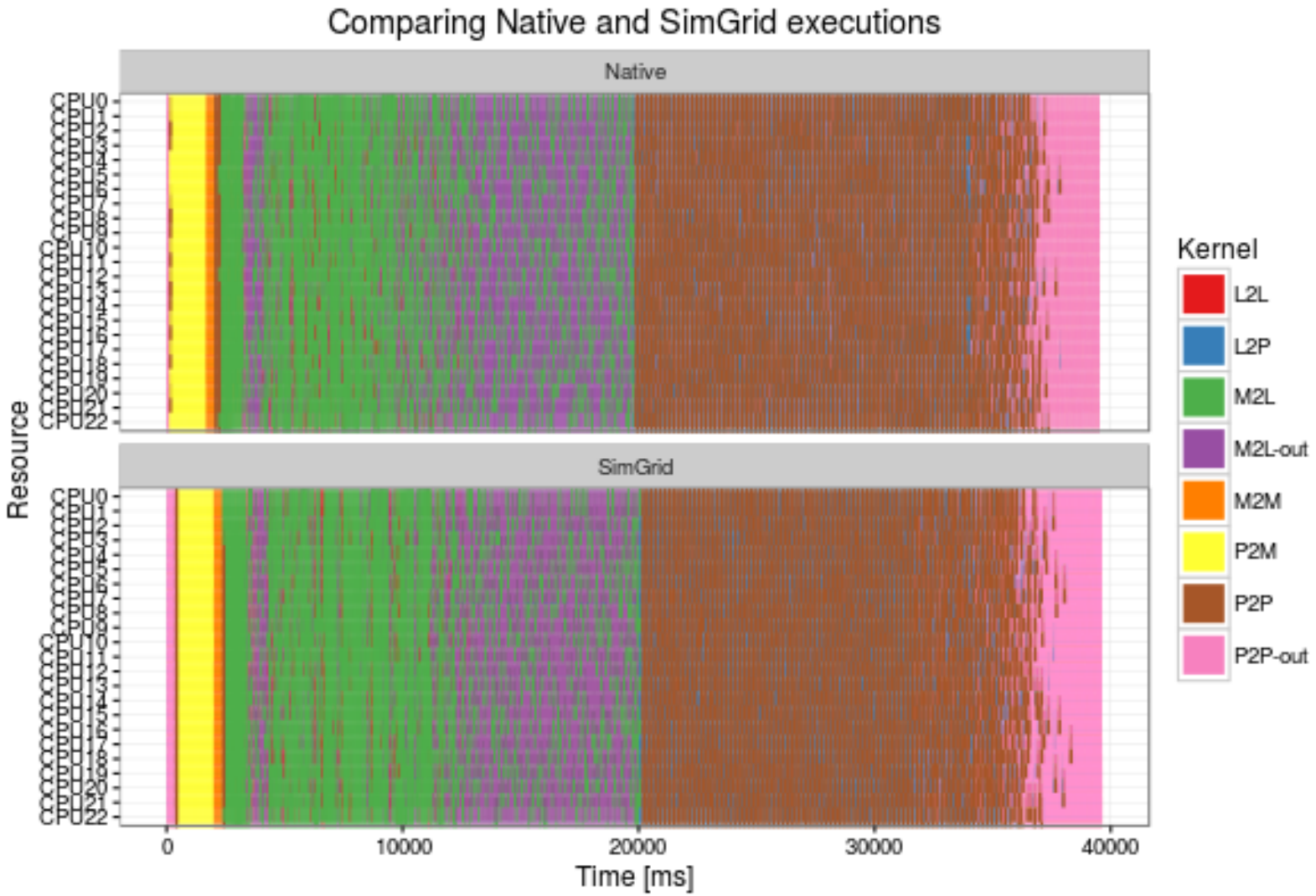
Simulation accuracy with SimGrid



Simulation with StarPU/SimGrid (L. Stanisic)



Simulation with StarPU/SimGrid (L. Stanisic)



5

Conclusion

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

Thanks for your attention.

StarPU runtime system

Web Site: https://starpu.gitlabpages.inria.fr/

LGPL License

Open to external contributors