StarPU: Hybrid CPU/GPU Task Programming

C Extensions and MPI Support

Ludovic Courtès and Nathalie Furmento

ComPAS, Grenoble, January 2013

Welcome!

Runtime research team
joint team Inria - LaBRI
(CNRS, Université de Bordeaux)

`http://runtime.bordeaux.inria.fr/`

1. Rationale
2. Enter StarPU
   - Execution model
   - Task scheduling policy
   - Writing your own scheduler
   - Task model
3. The case for compiler support
4. MPI
5. Conclusion

1

Rationale

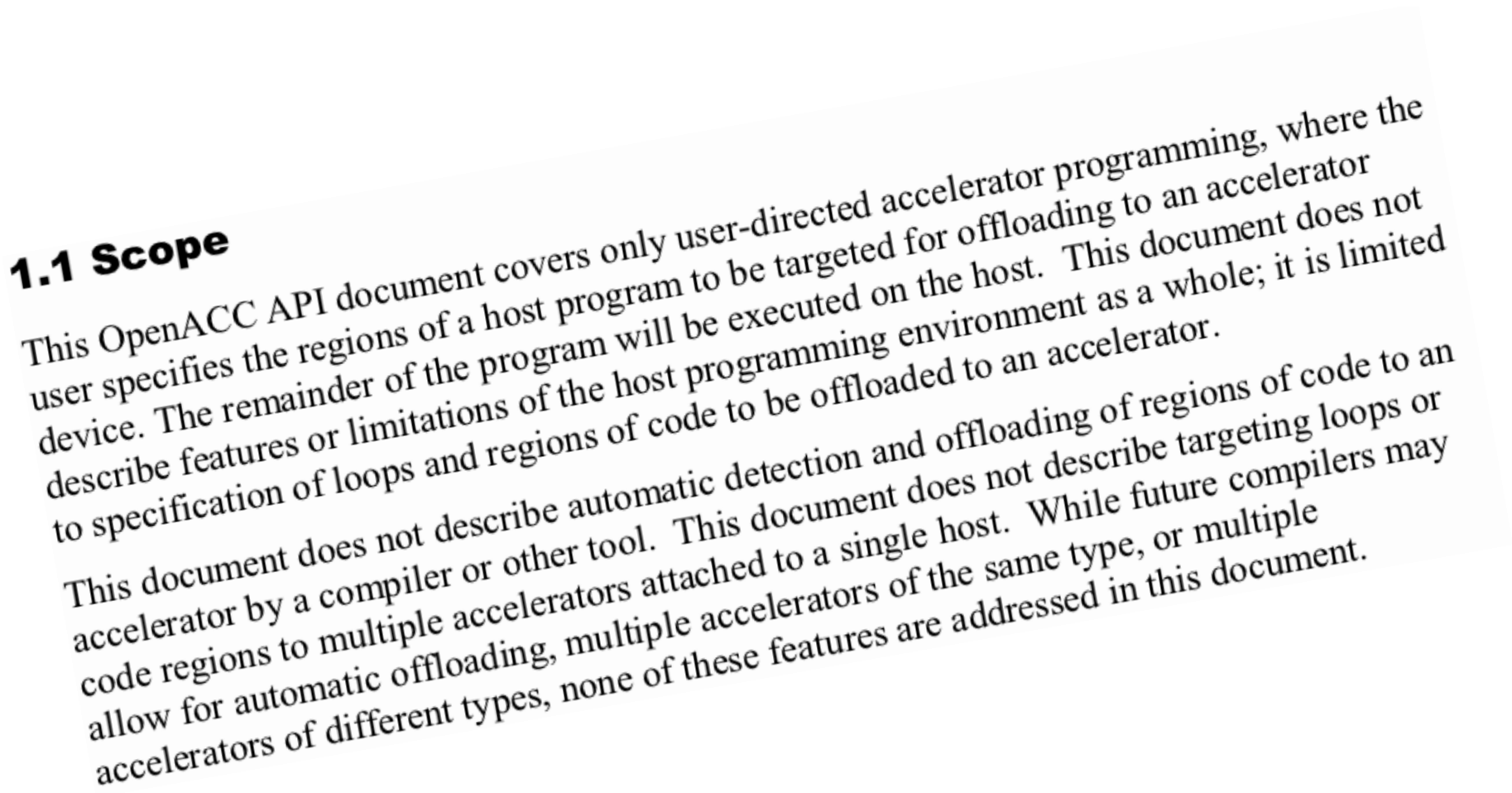
CPUs, GPUs, and all that

CPUs, GPUs, and all that

CPUs, GPUs, and all that

CPUs, GPUs, and all that

Short-sightedness in the multicore + GPU era

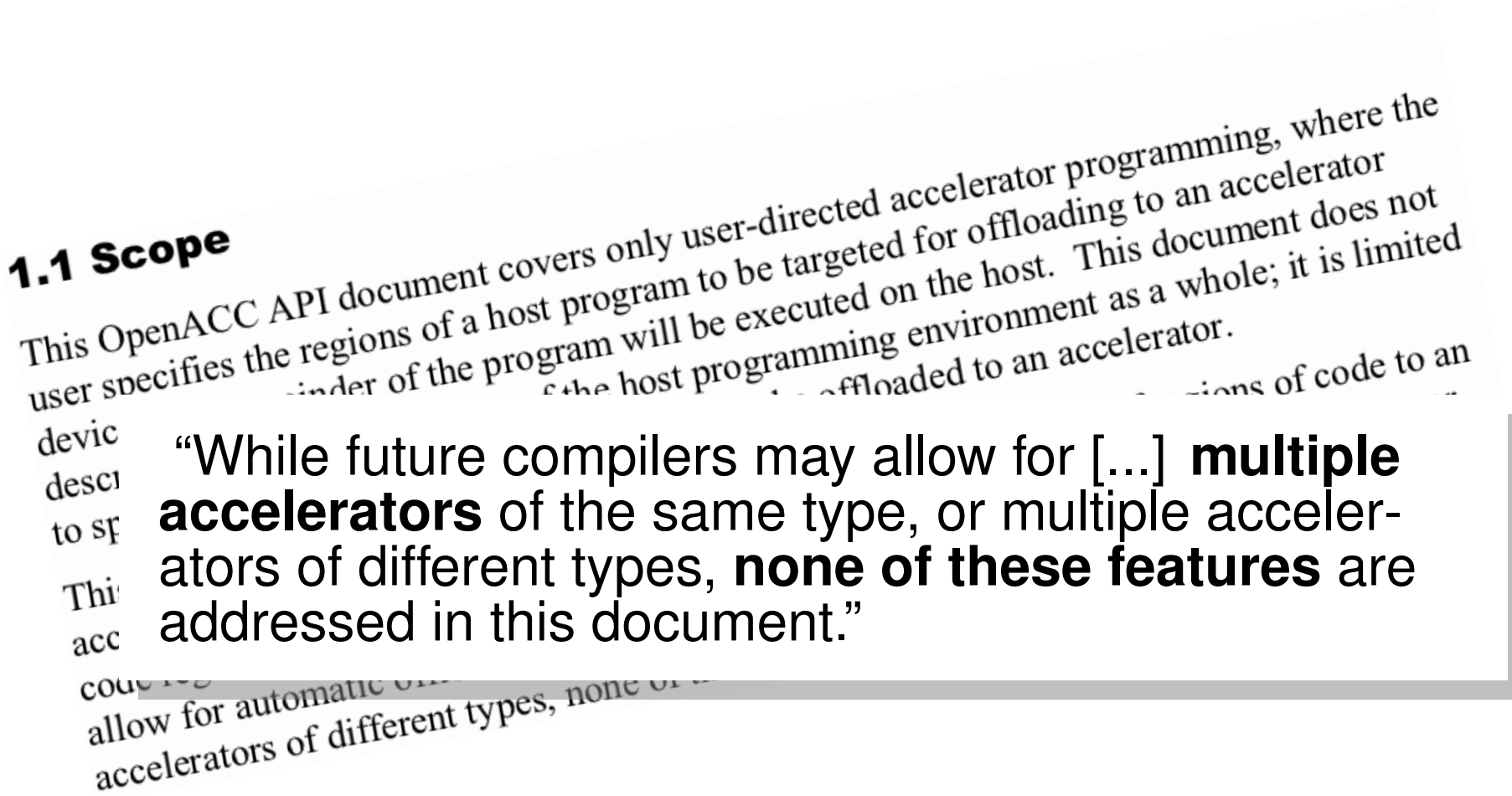


1.1 Scope

This OpenACC API document covers only user-directed accelerator programming, where the user specifies the regions of a host program to be targeted for offloading to an accelerator device. The remainder of the program will be executed on the host. This document does not describe features or limitations of the host programming environment as a whole; it is limited to specification of loops and regions of code to be offloaded to an accelerator.

This document does not describe automatic detection and offloading of regions of code to an accelerator by a compiler or other tool. This document does not describe targeting loops or code regions to multiple accelerators attached to a single host. While future compilers may allow for automatic offloading, multiple accelerators of the same type, or multiple accelerators of different types, none of these features are addressed in this document.

Short-sightedness in the multicore + GPU era

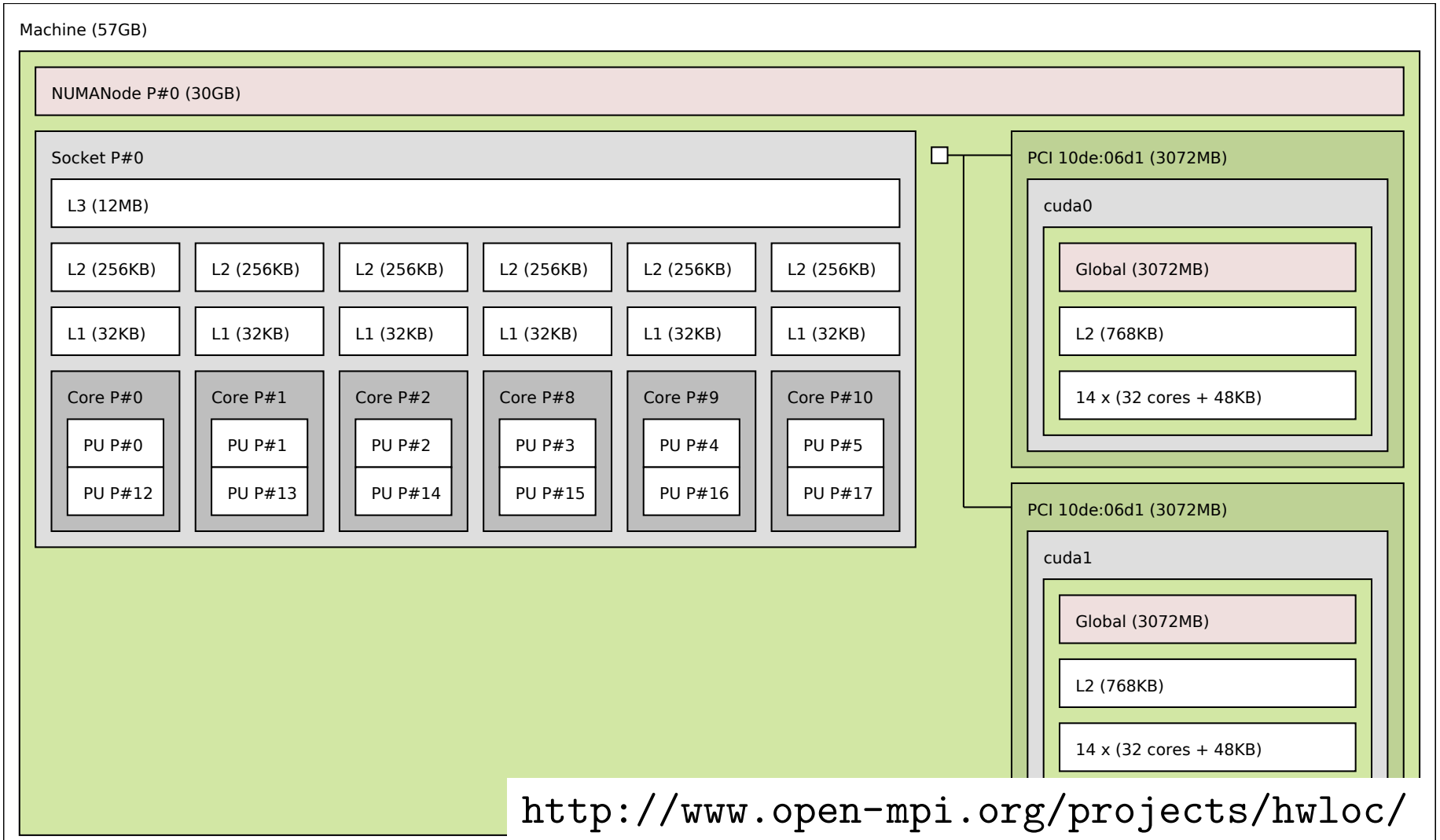


1.1 Scope

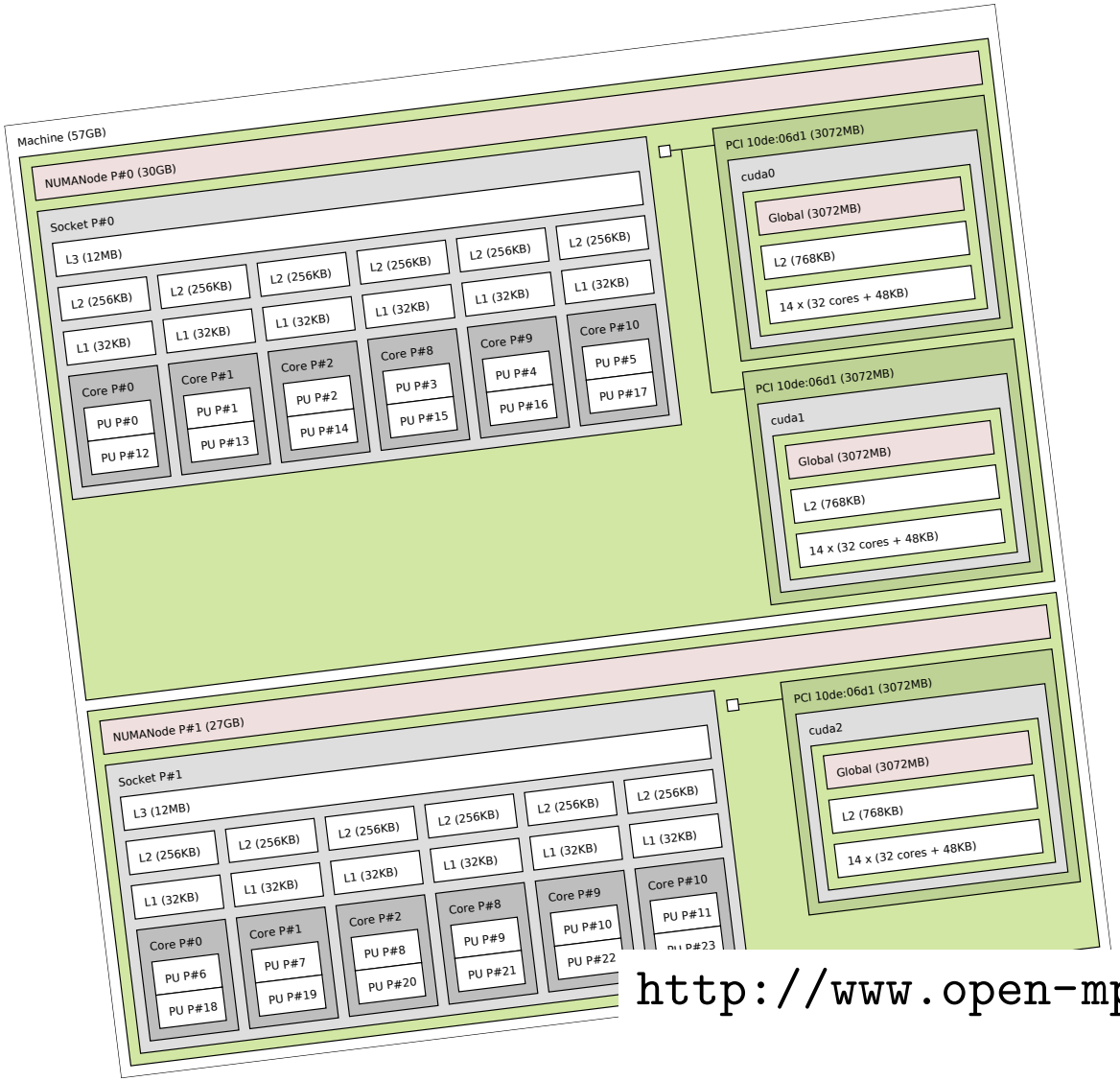
This OpenACC API document covers only user-directed accelerator programming, where the user specifies the regions of a host program to be targeted for offloading to an accelerator

"While future compilers may allow for [...] **multiple accelerators** of the same type, or multiple accelerators of different types, **none of these features** are addressed in this document."

What today's machines really look like

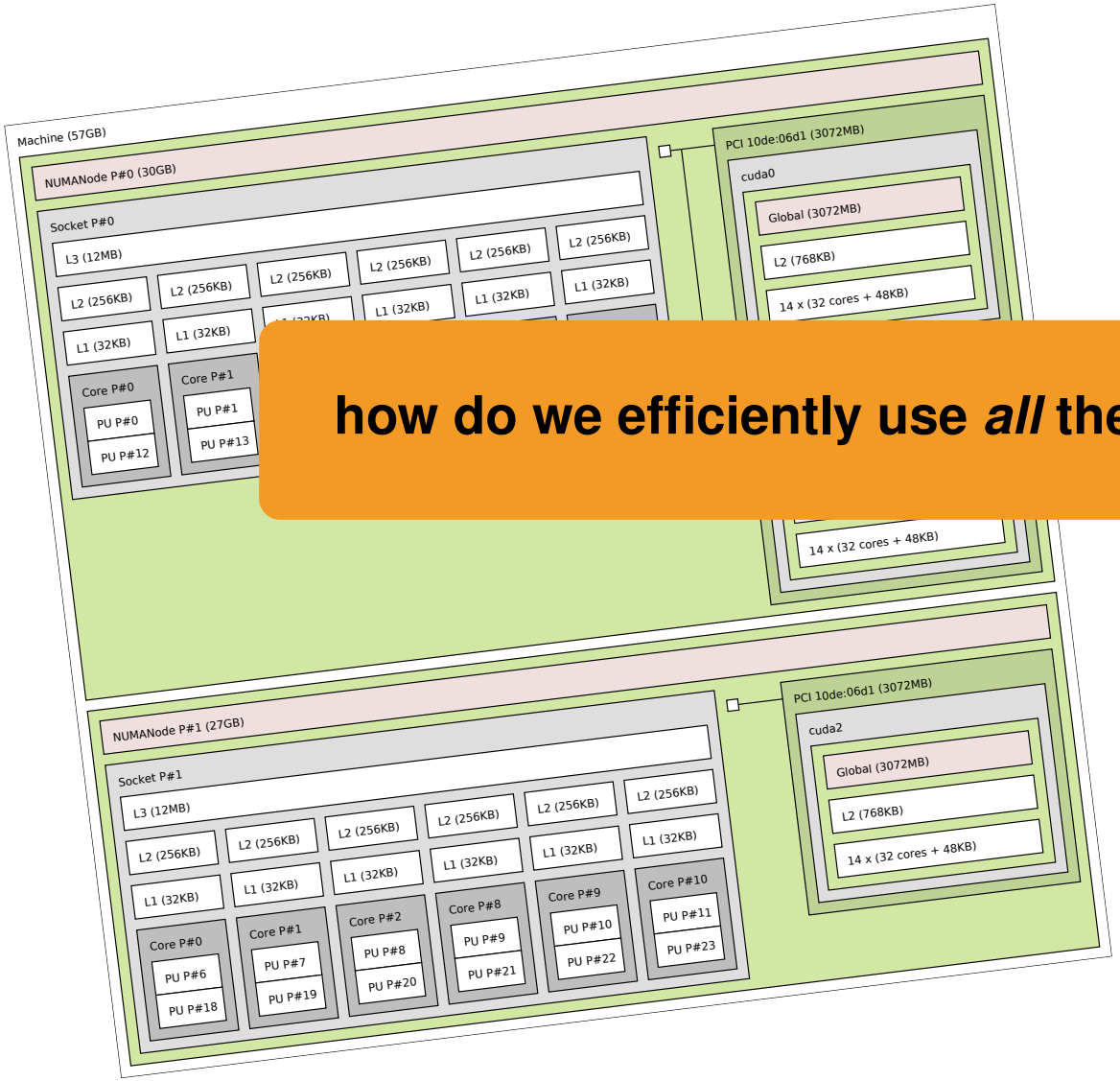


What today's machines really look like



http://www.open-mpi.org/projects/hwloc/

What today's machines really look like



how do we efficiently use *all* these PUs?

Short-sightedness in the multicore + GPU era

```
clGetDeviceIDs (NULL, CL_DEVICE_TYPE_DEFAULT, 1,
                &device_id, NULL);
queue = clCreateCommandQueue (context,  device_id , 0, NULL);


buf = clCreateBuffer (context, CL_MEM_READ_ONLY, ...);
 clEnqueueReadBuffer  (xfer_queues[device_id], buf,
                       CL_FALSE, ...);


...

global_work_size[0] = num_entries;
local_work_size[0] = 64;
clEnqueueNDRangeKernel (queue, kernel, 1, NULL,
                        global_work_size, local_work_size,
                        0, NULL, NULL);
```

Short-sightedness in the multicore + GPU era

```
clGetDeviceIDs (NULL, CL_DEVICE_TYPE_DEFAULT, 1,
                &device_id, NULL);
queue = clCreateCommandQueue (context, device_id , 0, NULL);
```

explicit choice of device

```
buf = clCreateBuffer (context, CL_MEM_READ_ONLY, ...);
 clEnqueueReadBuffer  (xfer_queues[device_id], buf,
                       CL_FALSE, ...);
```

explicit data transfer

```
...

global_work_size[0] = num_entries;
local_work_size[0] = 64;
clEnqueueNDRangeKernel (queue, kernel, 1, NULL,
                        global_work_size, local_work_size,
                        0, NULL, NULL);
```

Short-sightedness in the multicore + GPU era

```
clGetDeviceIDs (NULL, CL_DEVICE_TYPE_DEFAULT, 1,
                &device_id, NULL);
queue = clCreateCommandQueue (context, device_id , 0, NULL);
```

explicit choice of device

```
buf = clCreateBuffer (context, CL_MEM_READ_ONLY, ...);
clEnqueueReadBuffer  (xfer_queues[device_id], buf,
                      CL_FALSE, ...);
```

explicit data transfer

```
...

global_work_size[0] = num_entries;
local_wor
clEnqueue                                          _work_size,
                          0, NULL, NULL);
```

what about performance portability?

2

Enter StarPU

1. Rationale

2. Enter StarPU
   - Execution model
   - Task scheduling policy
   - Writing your own scheduler
   - Task model

3. The case for compiler support

4. MPI

5. Conclusion

runtime support to

schedule tasks over all the

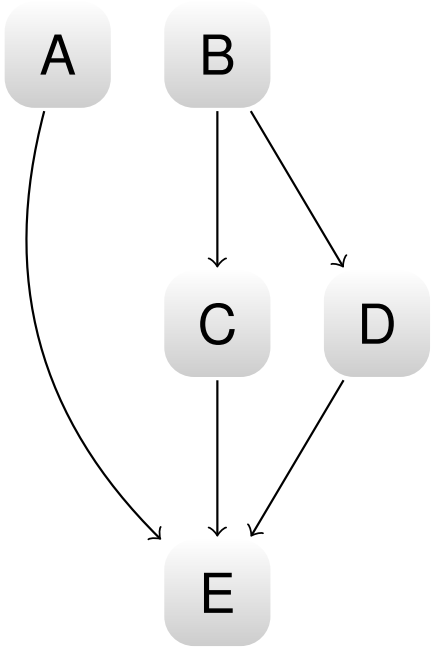
available **processing units**

runtime support to
schedule tasks over all the
available **processing units**

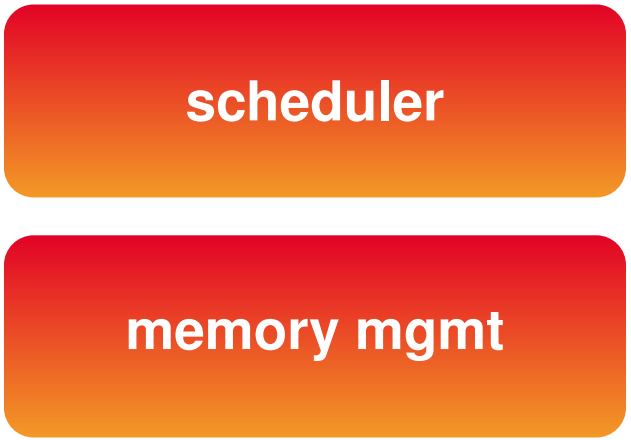
- C library, LGPLv2.1+
- started in 2009

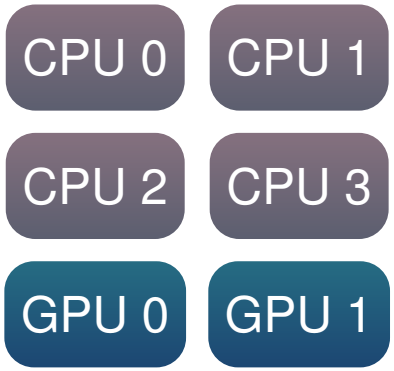
In a nutshell

DAG of tasks

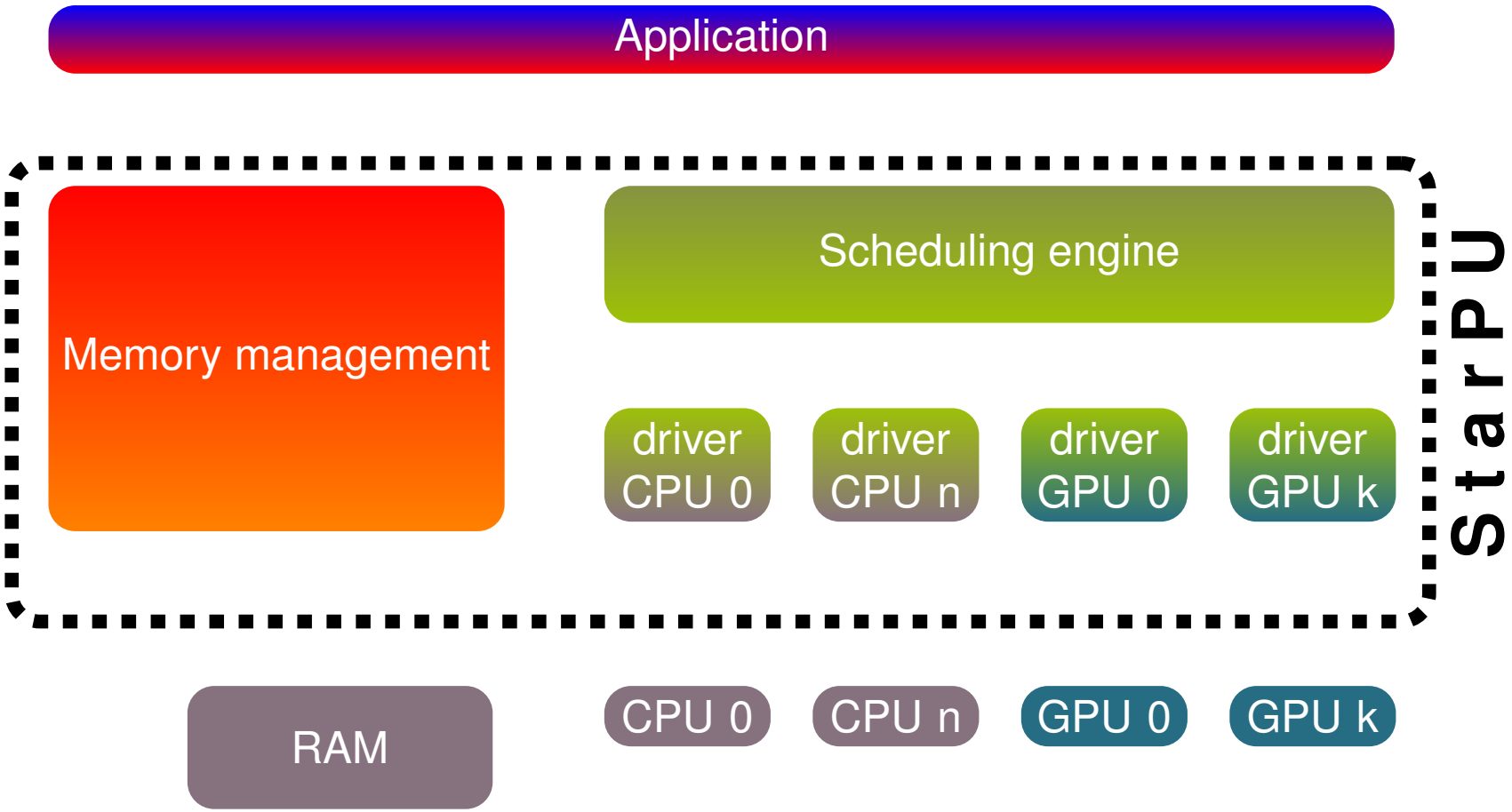


StarPU's runtime

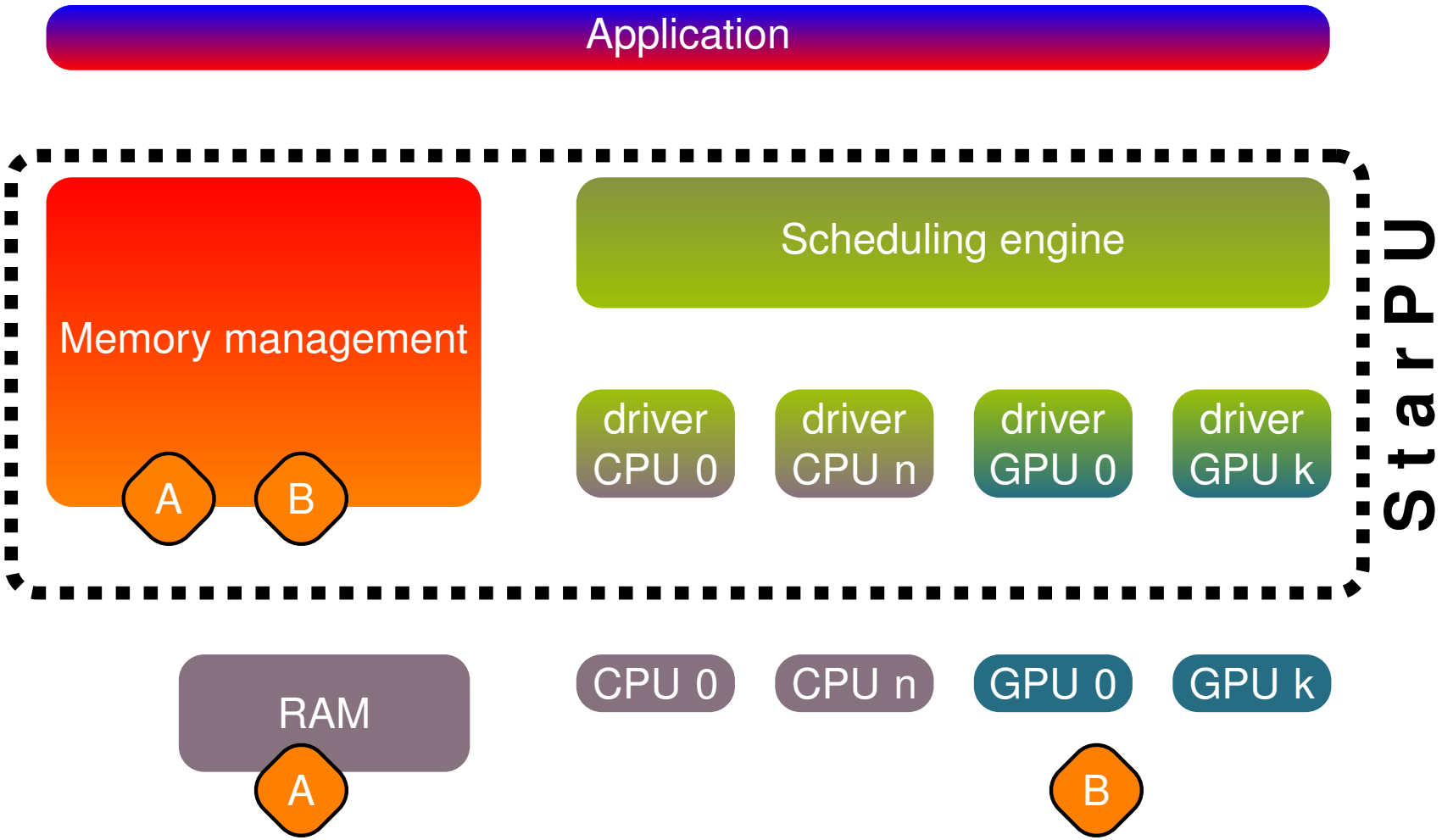




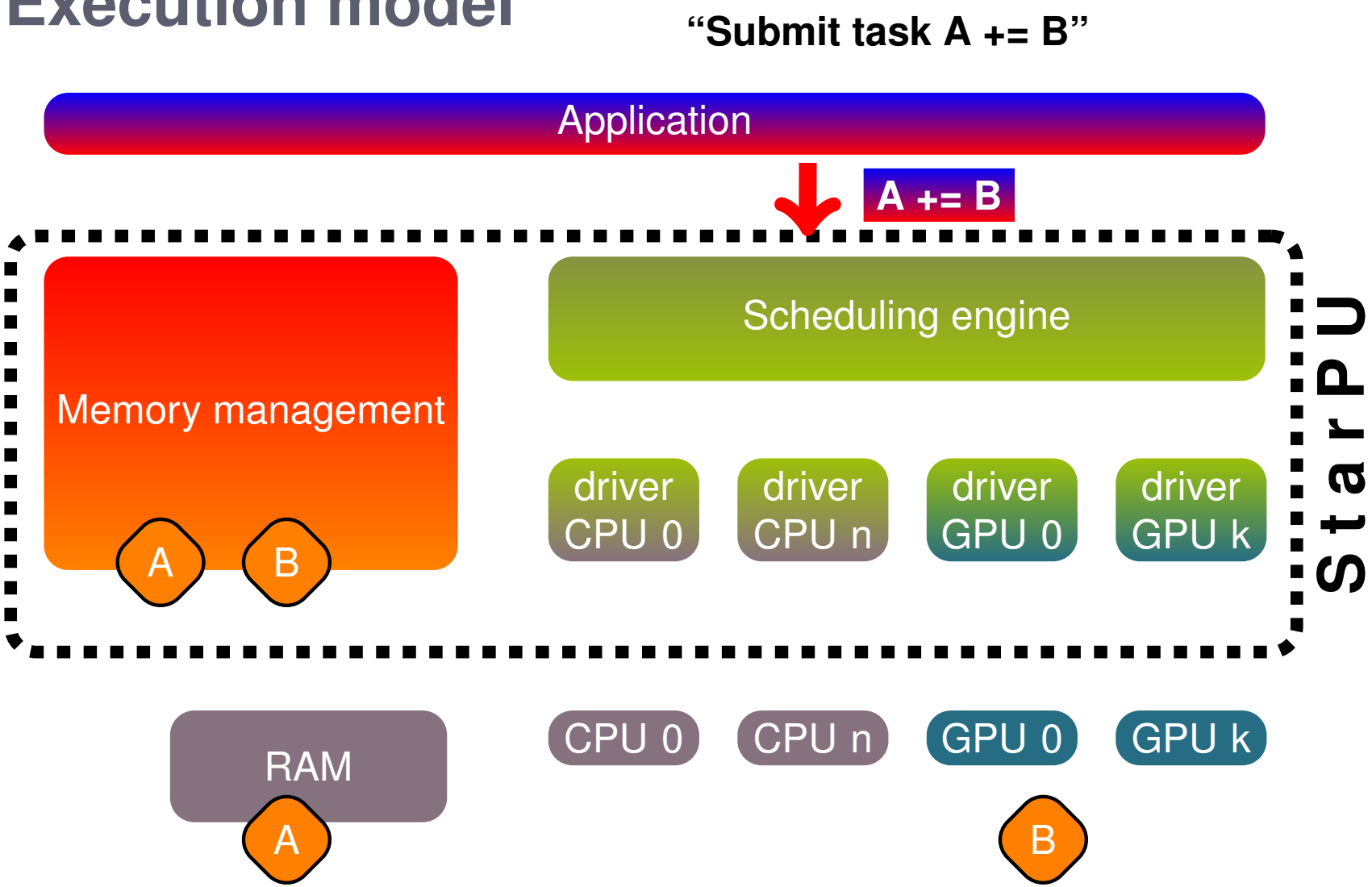
Execution model



Execution model

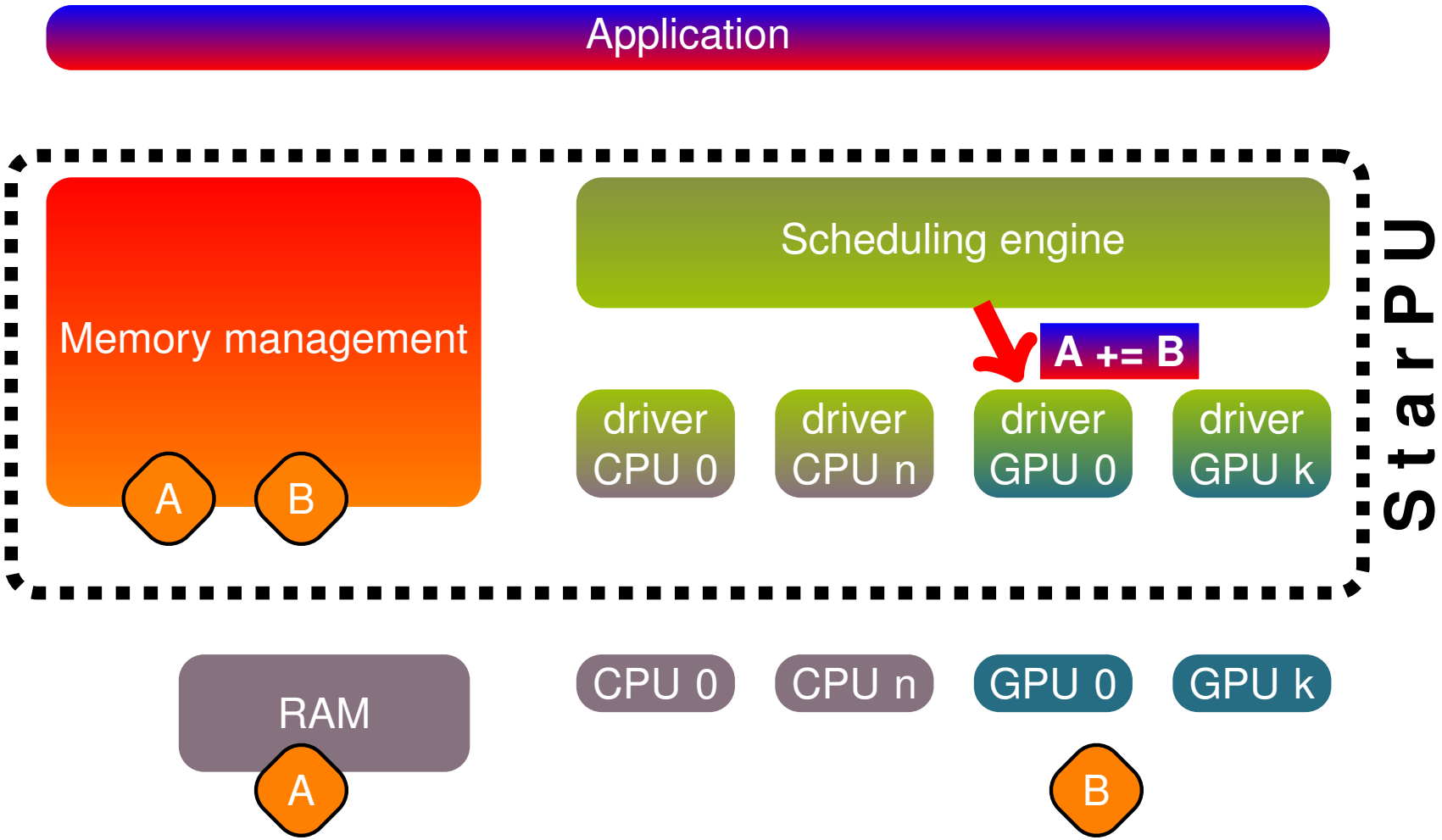


Execution model

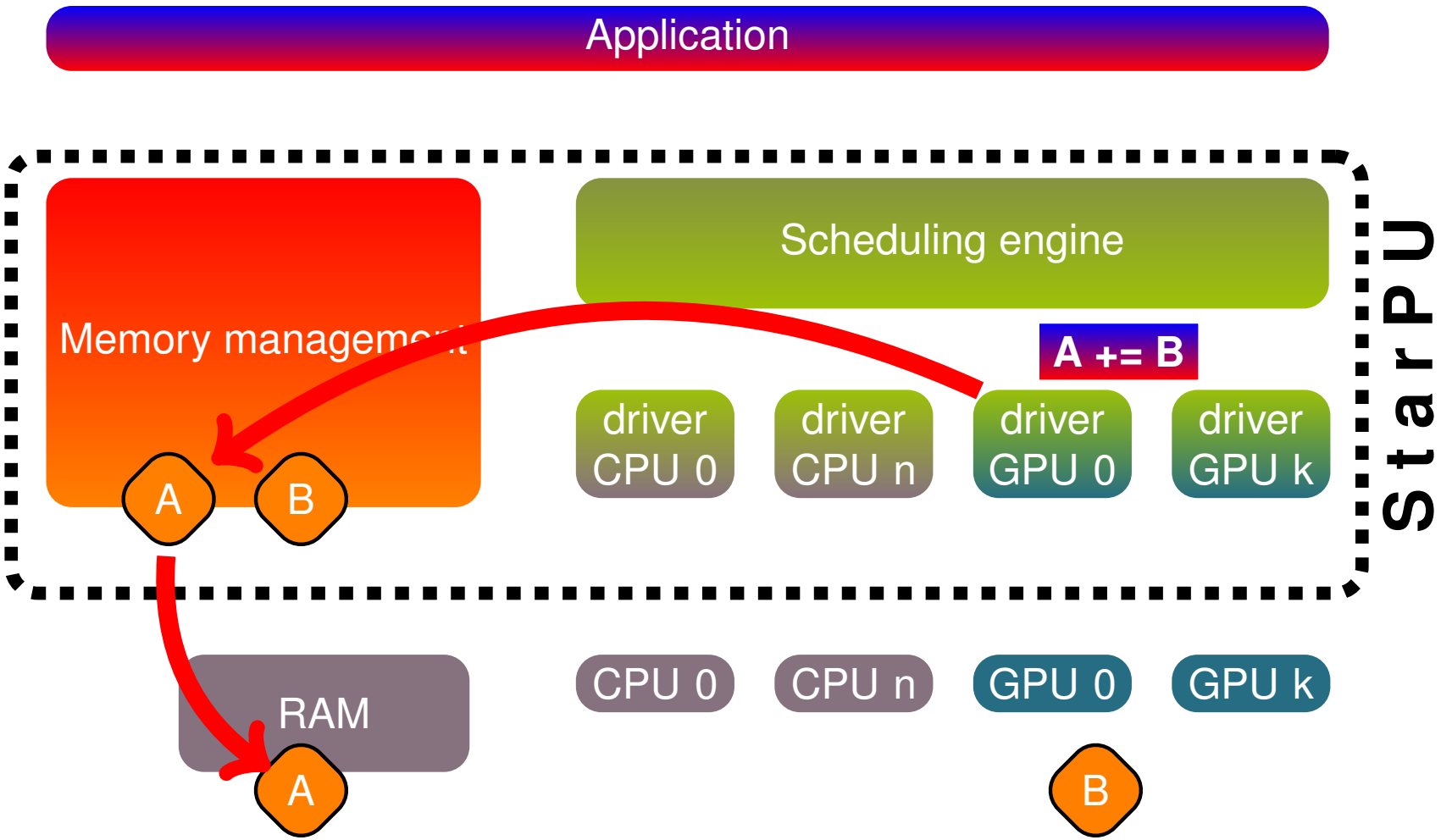


Execution model

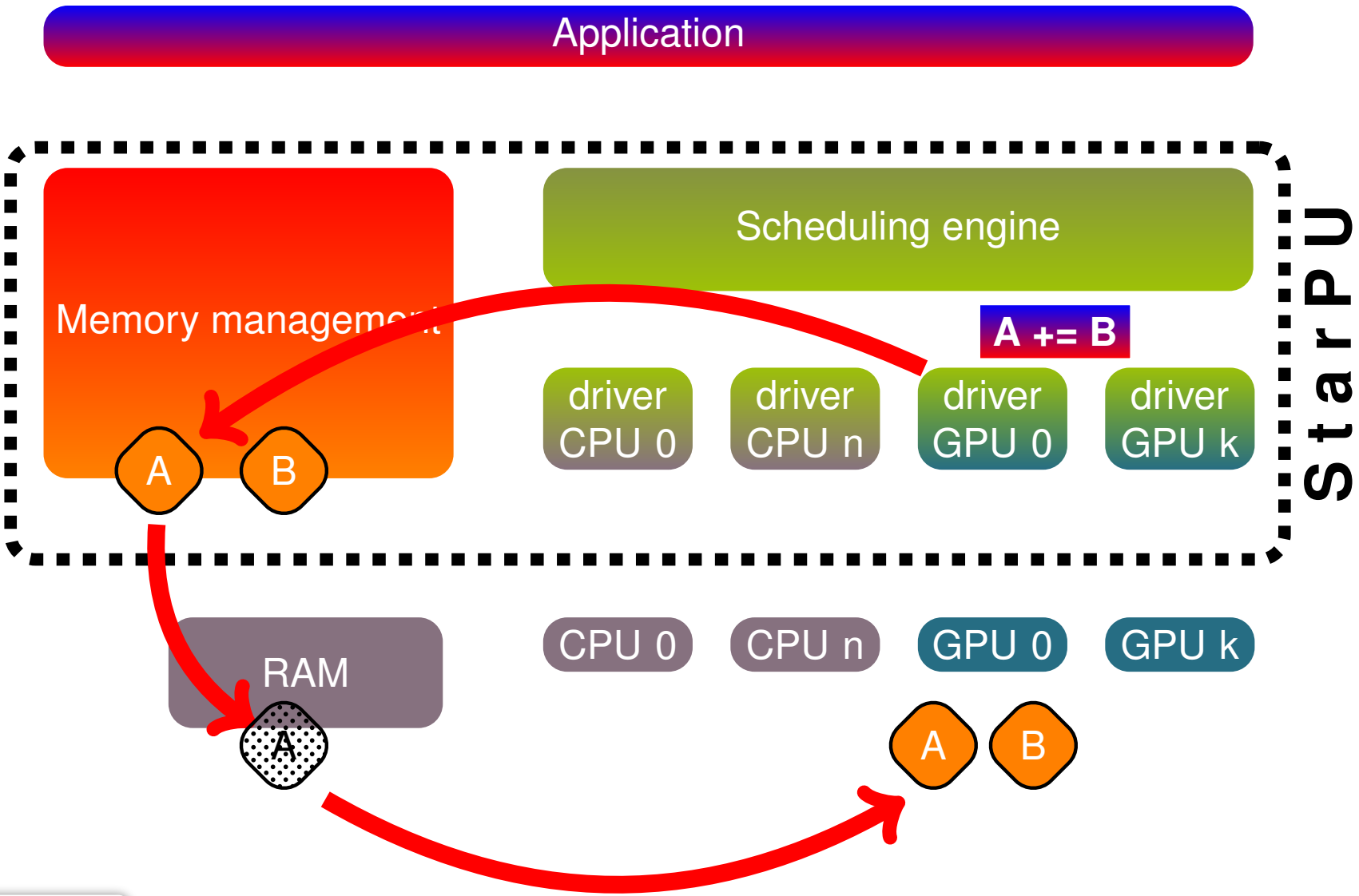
"Schedule task A += B"



Execution model

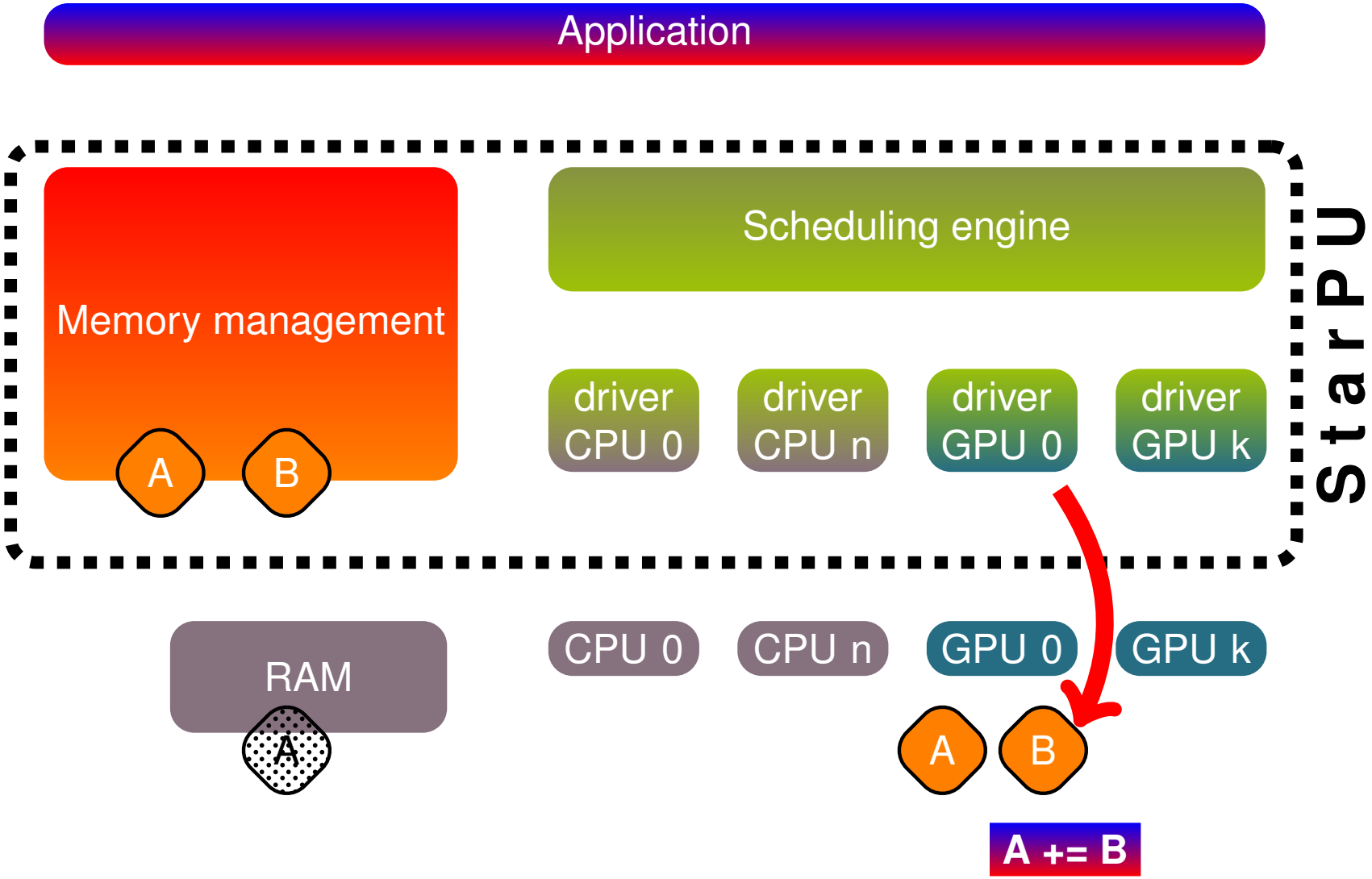


Execution model

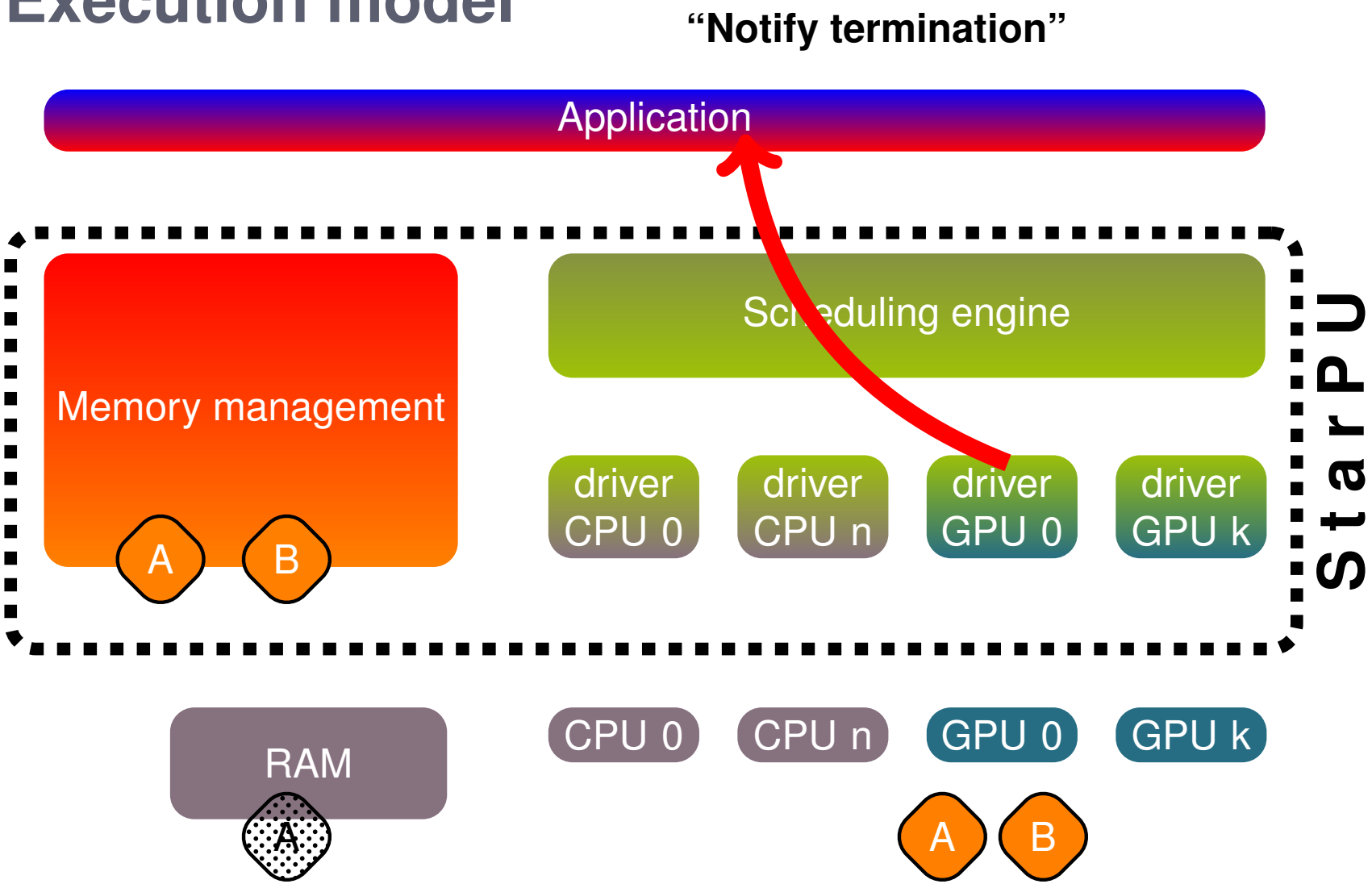


Execution model

“Offload computation”



Execution model



Task scheduling policy

- Default scheduler: eager simple greedy scheduler.
  - provide correct load balancing with or without performance models;
  - single central queue for all workers
  - no data prefetching (scheduling decision taken too late).

Task scheduling policy

- Default scheduler: eager simple greedy scheduler.
  - provide correct load balancing with [illegible]erformance models;
  - single central queue [illegible]
  - no da[illegible]o late).

Let's define performance models ...

Task scheduling policy

Let's define performance models ...

- StarPU will then really take scheduling decision in advance according to performance models, and issue data prefetch requests, to overlap data transfers and computations.

Task scheduling policy

Let's compare eager and dmda scheduling policies...

```
$ STARPU_BUS_STATS=1 STARPU_WORKER_STATS=1 sgemm -x 2048 -y 2048
-z 2048
```

Task scheduling policy

1. Eager (default) scheduling policy

```
$ STA                                                    t 2048 -y 2048
-z 20
```

Time: 695.62 ms
GFlop/s: 246.97

Total transfers: 206.00 MB

Worker statistics:
CUDA 0 27 task(s)
CUDA 1 22 task(s)
CUDA 2 25 task(s)
CPU 0 11 task(s)
CPU 1 11 task(s)
CPU 2 10 task(s)
... CPU 8 10 task(s)

Task scheduling policy

Let's compare eager and dmda scheduling policies...

```
$ STARPU_BUS_STATS=1 STARPU_WORKER_STATS=1 STARPU_SCHED=dmda sgemm
-x 2048 -y 2048 -z 2048
```

Task scheduling policy

2. dmda scheduling policy

```
$ STA                                                    CHED=dmda sgemm
-x 20
```

Time: 146.51 ms
GFlop/s: 1172.60

Total data transfers: 102.00 MB

Worker statistics:
CUDA 0 63 task(s)
CUDA 1 51 task(s)
CUDA 2 52 task(s)
CPU 0 0 task(s)
...
CPU 8 0 task(s)

Writing your own scheduler

- Define a **struct starpu_sched_policy** object with all the functions needed by a new scheduler
  - → Name of the policy
    const char *policy_name
  - → Description of the policy
    const char *policy_description

Writing your own scheduler

- Define a **struct starpu_sched_policy** object with all the functions needed by a new scheduler
  - → Initialize the scheduling policy
    void (*init_sched)(...)
  - → Cleanup the scheduling policy
    void (*deinit_sched)(...)
  - → Insert a task into the scheduler
    int (*push_task)(struct starpu_task *)

Writing your own scheduler

- Define a **struct starpu_sched_policy** object with all the functions needed by a new scheduler
  - → Get a task from the scheduler
    struct starpu_task *(*pop_task)(void)
  - → Remove all available tasks from the scheduler
    struct starpu_task *(*pop_every_task)(void)
  - → Notify a task is starting
    void (*pre_exec_hook)(struct starpu_task *)
  - → Notify a task has been executed
    void (*post_exec_hook)(struct starpu_task *)

Writing your own scheduler

- Define a **struct starpu_sched_policy** object with all the functions needed by a new scheduler
  - → Get a task from the scheduler
    struct sta
  - → **is starting**
    void (*pre_exec_hook)(struct starpu_task *)
  - → Notify a task has been executed
    void (*post_exec_hook)(struct starpu_task *)

See examples/scheduler/dummy_sched.c

Task Model


```
void scale_vector_cpu (void *buffers[], void *args);
void scale_vector_opencl (void *buffers[], void *args);

static struct starpu_codelet scale_vector_codelet =
{
    .cpu_funcs = { scale_vector_cpu, NULL },
    .opencl_funcs = { scale_vector_opencl, NULL },
    .nbuffers = 1,
    .modes =  STARPU_RW
};
```

Task Model

2. defining the task's CPU implementation

```
void scale_vector_cpu (void *buffers[], void *arg)
{
    /* Unpack the arguments... */
    float *factor = arg;
    starpu_vector_interface_t *vector = buffers[0];
    unsigned n = STARPU_VECTOR_GET_NX (vector);
    float *val = (float *) STARPU_VECTOR_GET_PTR (vector);

    /* scale the vector */
    for (unsigned i = 0; i < n; i++)
      val[i] *= *factor;
}
```

Task Model


```
float v[NX];
starpu_data_handle v_handle;
starpu_vector_data_register (&v_handle, 0, v,
                             NX, sizeof(v[0]));

float factor = 3.14;

starpu_insert_task (&scale_vector_codelet,
                    STARPU_VALUE, &factor, sizeof(factor),
                    STARPU_RW, v_handle,
                    0);

...
starpu_task_wait_for_all ();
starpu_data_unregister (v_handle);
```

Task Model


```
float v[NX];
starpu_data_handle v_handle;
starpu_vector_data_register (&v_handle, 0, v,
                             NX, sizeof(v[0]));

float factor = 3.14;

starpu_insert_task (&scale_vector_codelet,
                    STARPU_VALUE, &factor, sizeof(factor),
                    STARPU_RW, v_handle,
                    0);

...
starpu_task_wait_for_all ();
starpu_data_unregister (v_handle);
```

3

The case for compiler support

promoting **library** interfaces

as **language constructs**

promoting **library** interfaces as **language constructs**

- GCC plug-in, GPLv3+
- started in 2011
- for GCC 4.5, 4.6, and 4.7

Tasks are functions

```
void scale_vector (int size, float vector[size],
                   float factor)
  __attribute__ ((task));

void scale_vector_cpu (int size, float vector[size],
                       float factor)
  __attribute__ ((task_implementation
                  ("cpu", scale_vector)));

void
scale_vector_cpu (int size, float vector[size], float factor)
{
  for (int i = 0; i < size; i++)
    vector[i] *= factor;
}
```

Tasks are functions

```
void scale_vector (int size, float vector[size],
                   float factor)
  __attribute__ ((task));

/* The implicit CPU implementation.  */
void
scale_vector (int size, float vector[size], float factor)
{
  for (int i = 0; i < size; i++)
    vector[i] *= factor;
}
```

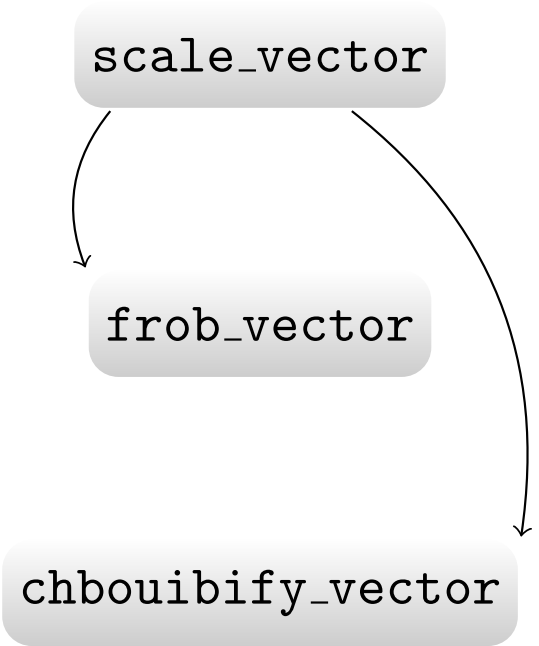
Tasks submissions are async. function calls

```
    static float input[NX];
    ...

#pragma starpu register input
    ...
    scale_vector (NX, input, 42);



#pragma starpu wait
```

scale_vector

Tasks submissions are async. function calls

```
    static float input[NX], out1[NX], out2[NX];
    ...

#pragma starpu register input
    ...
    scale_vector (NX, input, 42);
    ...
    frob_vector (NX, input, out1);
    chbouibify_vector (NX, input, out2);
#pragma starpu wait
```



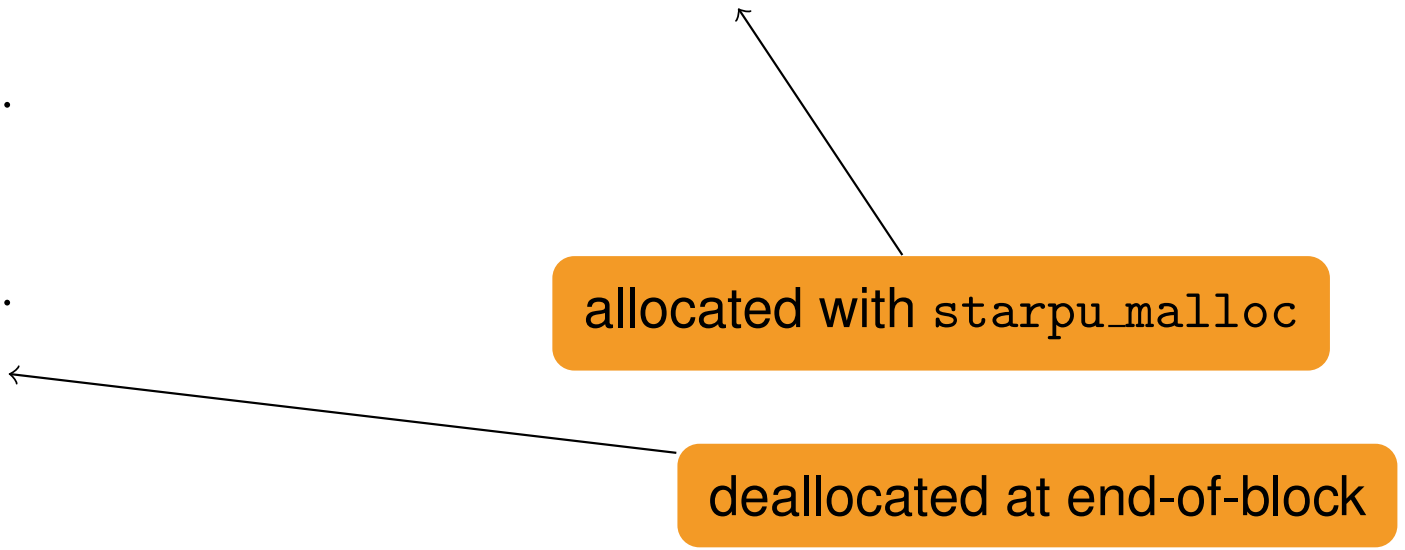
Tasks submissions are async. function calls

```
    static float input[NX], out1[NX], out2[NX];
    ...

#pragma starpu register input
    ...
    scale_vector (NX, input, 42);
    ...
    frob_vector (NX, input, out1);
    chbouibify_vector (NX, input, out2);
#pragma starpu wait

#pragma starpu  unregister
  display_vector (vector);
```

make sure data is in main memory

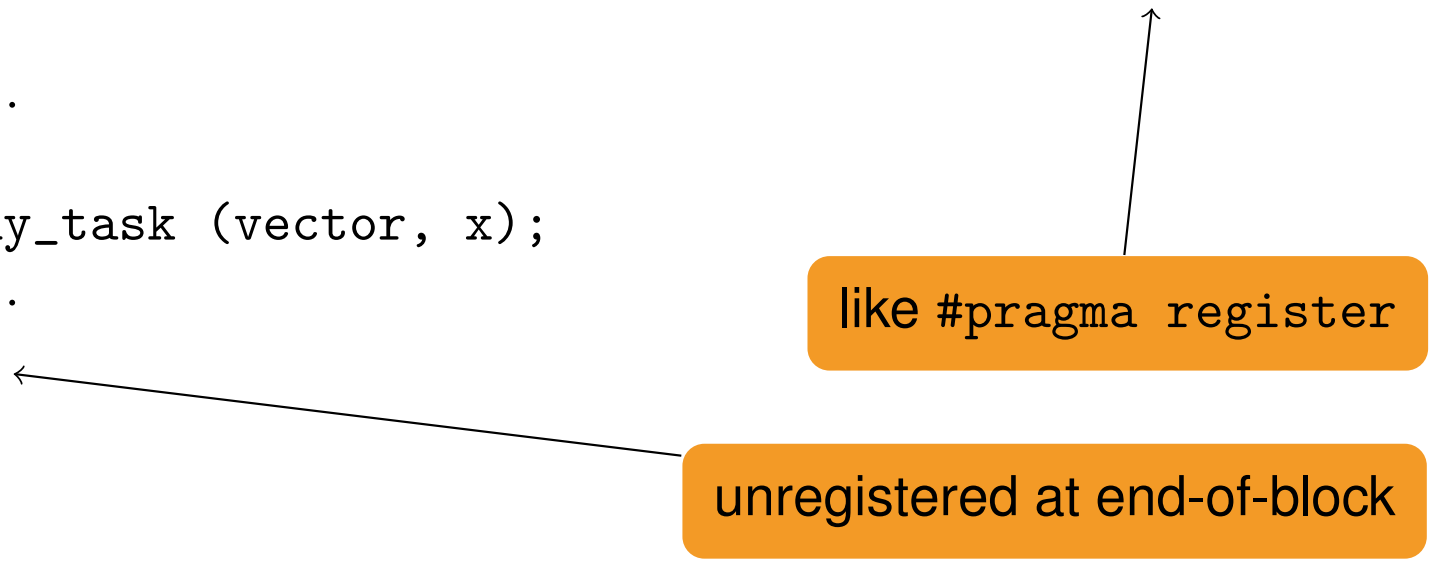
Memory management helpers

```
int
foo (int x)
{
  float vector[x]
    __attribute__ (( heap_allocated ));

  ...


  ...

}
```



Memory management helpers

```
int
foo (int x)
{
  float vector[x]
    __attribute__ (( heap_allocated ,  registered ));

  ...

  my_task (vector, x);
  ...
}
```



OpenCL task implementations

```
void vector_scal_opencl (int size, float vec[size],
                         float factor)
  __attribute__ ((task_implementation
                  ("opencl", vector_scal)));

void vector_scal_opencl (...)
{
  ...
  err = starpu_opencl_load_kernel (&kernel, &queue, &cl_programs,
                                   "vector_scal_opencl", devid);
  err = clSetKernelArg (kernel, 0, sizeof (val), &val);
  err |= clSetKernelArg (kernel, 1, sizeof (size), &size);
  ...
  err = clEnqueueNDRangeKernel (queue, kernel, 1, NULL, &global,
                                &local, 0, NULL, &event);
  ...
  clFinish (queue);
```

OpenCL task implementations

```
void vector_scal_opencl (int size, float vec[size],
                         float factor)
  __attribute__ ((task_implementation
                  ("opencl", vector_scal)));

void vector_scal_opencl (...)
{
  ...
  err = starpu                                    rnel, &queue, &cl_programs,
                                                  "vector_scal_opencl", devid);
  err = clSetKernelArg (kernel, 0, sizeof (val), &val);
  err |= clSetKernelArg (kernel, 1, sizeof (size), &size);
  ...
  err = clEnqueueNDRangeKernel (queue, kernel, 1, NULL, &global,
                                &local, 0, NULL, &event);
  ...
  clFinish (queue);
```

say no to copy/paste!

OpenCL task implementations

```
void vector_scal_opencl (int size, float vec[size],
                         float factor)
  __attribute__ ((task_implementation
                  ("opencl", vector_scal)));

#pragma starpu opencl vector_scal_opencl \
    "vector-scale.cl" "vector_scal_kern" \
    group_size ngroups
```

Future work

- automatic registration of static arrays
- error out for buffers provably not registered
- OpenMP integration—e.g., generating tasks with `parallel for`
- OpenCL kernel code generation? (GRAPHITE-OpenCL?)
- ...

4

MPI

Task Model

```
float v[NX];
starpu_data_handle v_handle;
starpu_vector_data_register (&v_handle, 0, v,
                             NX, sizeof(v[0]));

float factor = 3.14;

starpu_insert_task (&scale_vector_codelet,
                    STARPU_VALUE, &factor, sizeof(factor),
                    STARPU_RW, v_handle,
                    0);

...
starpu_task_wait_for_all ();
starpu_data_unregister (v_handle);
```

Towards MPI

```
for(i=0 ; i<N ; i++)
    starpu_vector_data_register(&v_handle[i], 0,
                v[i*(N+1)], NX, sizeof(v[0]));

float factor = 3.14;

for(i=0 ; i<N ; i++)
    starpu_insert_task(&scale_vector_codelet,
                STARPU_VALUE, &factor, sizeof(factor),
                STARPU_RW, v_handle[i],
                0);
```

Let's go parallel

```
for(i=0 ; i<N ; i++)

    starpu_vector_data_register(&v_handle[i], 0, v[i*(N+1)], NX,
                                sizeof(v[0]));
```

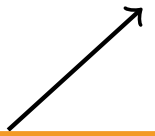
Let's go parallel

function specifying data distribution

```
for(i=0 ; i<N ; i++)

    int mpi_rank =  my_distrib (i, size);

    starpu_vector_data_register(&v_handle[i], 0, v[i*(N+1)], NX,
                    sizeof(v[0]));
```

Let's go parallel

```
for(i=0 ; i<N ; i++)

    int mpi_rank =  my_distrib (i, size);

    starpu_vector_data_register(&v_handle[i], 0, v[i*(N+1)], NX,
                    sizeof(v[0]));

    starpu_data_set_rank(v_handle[i],  mpi_rank );
```



specify the node owning the data

Let's go parallel

```
for(i=0 ; i<N ; i++)

    int mpi_rank =  my_distrib (i, size);

    starpu_vector_data_register(&v_handle[i], 0, v[i*(N+1)], NX,
                    sizeof(v[0]));

    starpu_data_set_rank(v_handle[i],  mpi_rank );

    starpu_data_set_tag(v_handle[i],  i );
```

specify the tag which will be used when exchanging the data

Let's go parallel

identify node which will execute the codelet

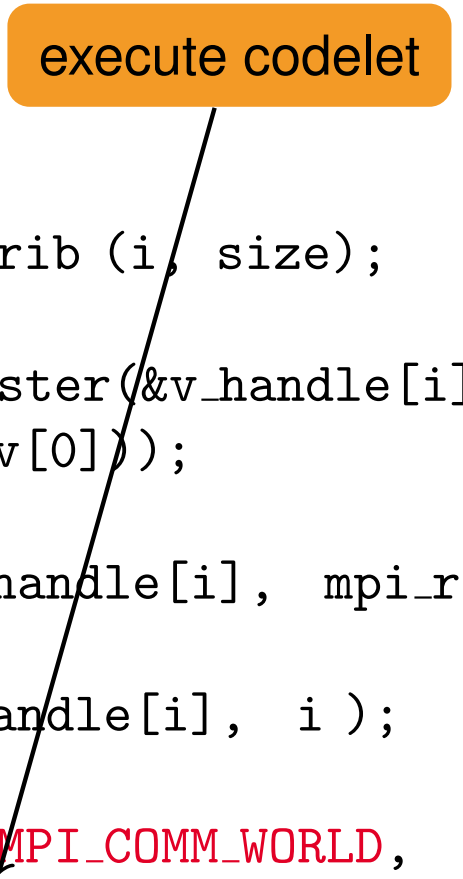
```
for(i=0 ; i<N ; i++)

    int mpi_rank =  my_distrib (i, size);

    starpu_vector_data_register(&v_handle[i], 0, v[i*(N+1)], NX,
                    sizeof(v[0]));

    starpu_data_set_rank(v_handle[i],  mpi_rank );

    starpu_data_set_tag(v_handle[i],  i );

    starpu_mpi_insert_task(MPI_COMM_WORLD,
                &scale_vector_codelet ,
                STARPU_VALUE, &factor, sizeof(factor),
                STARPU_RW , v_handle[i],
                STARPU_R , v_handle[i-1],
                0);
```

Let's go parallel

send data to node which will execute the codelet

```
for(i=0 ; i<N ; i++)

    int mpi_rank =  my_distrib (i, size);

    starpu_vector_data_register(&v_handle[i], 0, v[i*(N+1)], NX,
                    sizeof(v[0]));

    starpu_data_set_rank(v_handle[i],  mpi_rank );

    starpu_data_set_tag(v_handle[i],  i );

    starpu_mpi_insert_task(MPI_COMM_WORLD,
                &scale_vector_codelet ,
                STARPU_VALUE, &factor, sizeof(factor),
                STARPU_RW , v_handle[i],
                STARPU_R , v_handle[i-1],
                0);
```

Let's go parallel




```
for(i=0 ; i<N ; i++)

    int mpi_rank =  my_distrib (i, size);

    starpu_vector_data_register(&v_handle[i], 0, v[i*(N+1)], NX,
                    sizeof(v[0]));

    starpu_data_set_rank(v_handle[i],  mpi_rank );

    starpu_data_set_tag(v_handle[i],  i );

    starpu_mpi_insert_task(MPI_COMM_WORLD,
                &scale_vector_codelet ,
                STARPU_VALUE, &factor, sizeof(factor),
                STARPU_RW , v_handle[i],
                STARPU_R , v_handle[i-1],
                0);
```

Let's go parallel

send back data which have been modified

```
for(i=0 ; i<N ; i++)

    int mpi_rank =  my_distrib (i, size);

    starpu_vector_data_register(&v_handle[i], 0, v[i*(N+1)], NX,
                    sizeof(v[0]));

    starpu_data_set_rank(v_handle[i],  mpi_rank );

    starpu_data_set_tag(v_handle[i],  i );

    starpu_mpi_insert_task(MPI_COMM_WORLD,
                &scale_vector_codelet ,
                STARPU_VALUE, &factor, sizeof(factor),
                STARPU_RW , v_handle[i],
                STARPU_R , v_handle[i-1],
                0);
```

Communication

- Automatically replace data dependencies across sub-graphs with MPI transfers
- Transparently overlap MPI communication with computation
  Implement an optional cache mechanism to eliminate redundant data transfers

Communication

- Automatically replace data dependencies across sub-graphs with MPI transfers
- Transparently overlap MPI communication with computation
- Implement an optional cache mechanism to eliminate redundant data transfers

Memory optimization

All nodes do not need to have all the data

```
for(i=0 ; i<N ; i++)
    int mpi_rank = my_distrib(i, size);
    if (mpi_rank == my_rank)
        // Owning data
        starpu_vector_data_register(&v_handle[i], 0,
                    v[i*(N+1)], NX, sizeof(v[0]));
```

Memory optimization

All nodes do not need to have all the data

```
for(i=0 ; i<N ; i++)
    int mpi_rank = my_distrib(i, size);
    if (mpi_rank == my_rank)
        // Owning data
        starpu_vector_data_register(&v_handle[i], 0,
                    v[i*(N+1)], NX, sizeof(v[0]));

    else if (my_rank == my_distrib(i+1, size) ||
             my_rank == my_distrib(i-1,size))
        // Not owning data, but needing it for my computations
        starpu_vector_data_register(&v_handle[i], -1, NULL,
                    NX, sizeof(v[0])); //lazy registration
```

Memory optimization

All nodes do not need to have all the data

```
for(i=0 ; i<N ; i++)
    int mpi_rank = my_distrib(i, size);
    if (mpi_rank == my_rank)
        // Owning data
        starpu_vector_data_register(&v_handle[i], 0,
                    v[i*(N+1)], NX, sizeof(v[0]));

    else if (my_rank == my_distrib(i+1, size) ||
             my_rank == my_distrib(i-1,size))
        // Not owning data, but needing it for my computations
        starpu_vector_data_register(&v_handle[i], -1, NULL,
                    NX, sizeof(v[0])); //lazy registration

    else
        // Not needing data
        v_handle[i] = NULL;
```

5

Conclusion

Summary

- StarPU
  - Express application as DAG
  - Dynamically scheduled over CPUs, GPUs, ...
  - Automatic optimized data transfers

Summary

- StarPU
  - Express application as DAG
  - Dynamically scheduled over CPUs, GPUs, ...
  - Automatic optimized data transfers
- C Extensions
  - a step forward in **portable heterogeneous programming**
  - the case for **language & compiler support**
  - GCC plug-ins allow for **richer programming interfaces**

Summary

- StarPU
  - Express application as DAG
  - Dynamically scheduled over CPUs, GPUs, ...
  - Automatic optimized data transfers
- C Extensions
  - a step forward in **portable heterogeneous programming**
  - the case for **language & compiler support**
  - GCC plug-ins allow for **richer programming interfaces**
- MPI
  - Extend the StarPU model to **clusters**
  - Avoid global **cluster-wide scheduler**
  - **Minimize** changes to single node clusters
  - **Preserve** the StarPU model within processes

ludovic.courtes@inria.fr

nathalie.furmento@labri.fr

`http://runtime.bordeaux.inria.fr/StarPU/`