MYX: "M"ust correctness check for the "Y"ML-"X"MP multi SPMD programming model

Miwako Tsuji





France-Japan-Germany trilateral workshop :
Convergence of HPC and Data Science for Future Extreme Scale Intelligent Applications
6-8 Nov 2019 Tokyo (Japan)

XMP+YML and FP3C project

- FP3C: Framework and Programming for Post Petascale Computing
  - a collaborative project between Japan and France
  - September. 2010 – March. 2014
- Various research fields and their integration
  - Programming model and programming language design
  - Runtime libraries
  - Accelerator
  - Algorithm and mathematical libraries
  - etc…

2013? @ Akihabara

MYX/SPPEXA Project Consortium

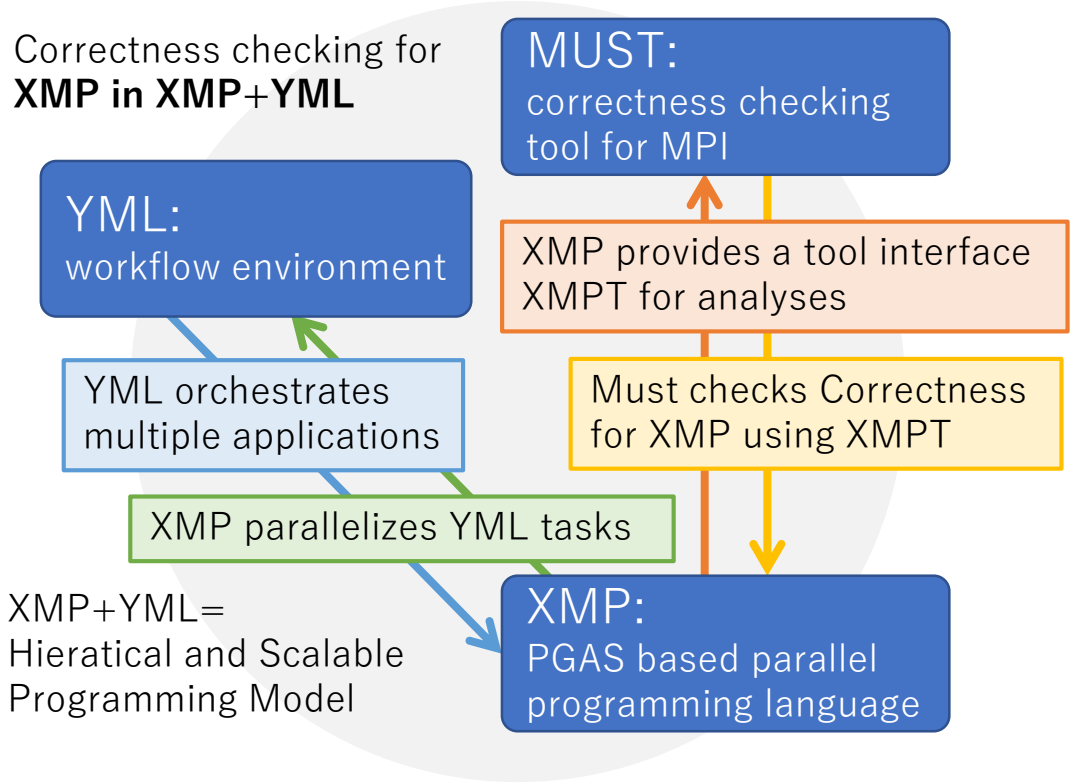
- MUST Correctness Checking for YML and XMP Programs.
- International collaboration among Germany (DFG), Japan (JST), and France (ANR).
- Part of the Priority Programme "Software for Exascale Computing" (SPPEXA) in German.

- Partner from Germany (project coordinator)
  - RWTH Aachen, IT Center and Institute for High Performance Computing
  - Prof. Matthias S. Mueller, Joachim Protze, Christian Terboven
- Partner from Japan
  - University of Tsukuba, Center for Computational Sciences, and Center for Computational Science, RIKEN
  - Prof. Taisuke Boku, Hitoshi Murai, Miwako Tsuji
- Partner from France
  - Maison de la Simulation
  - Prof. Serge Petiton. Prof. Nahid Emad, Thomas Dufaud

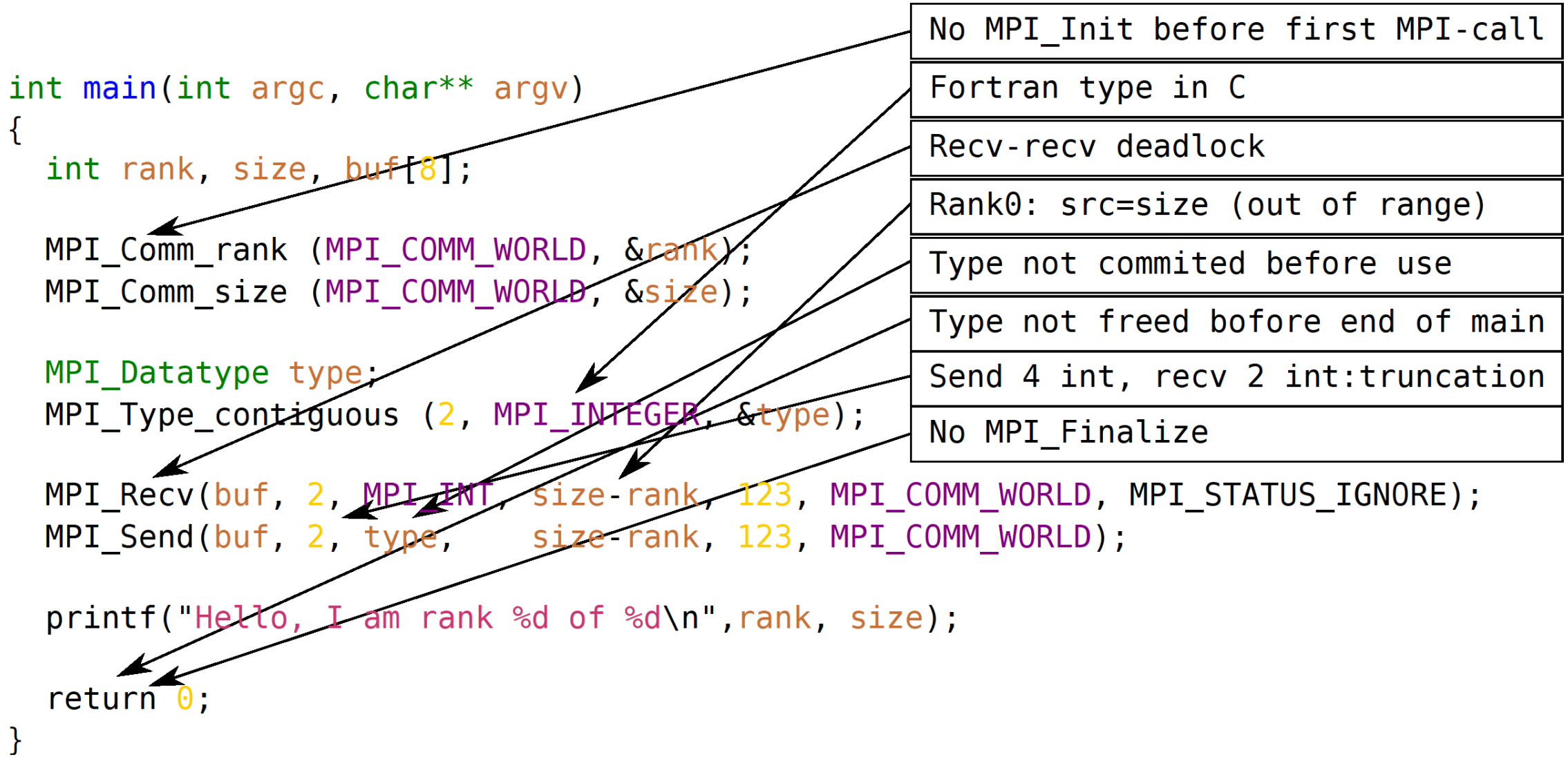
Overview of MYX

- Runtime Correctness check for multi SPMD (mSPMD) programming model
  - MUST (Germany)
  - YML (France)
  - XMP (Japan)

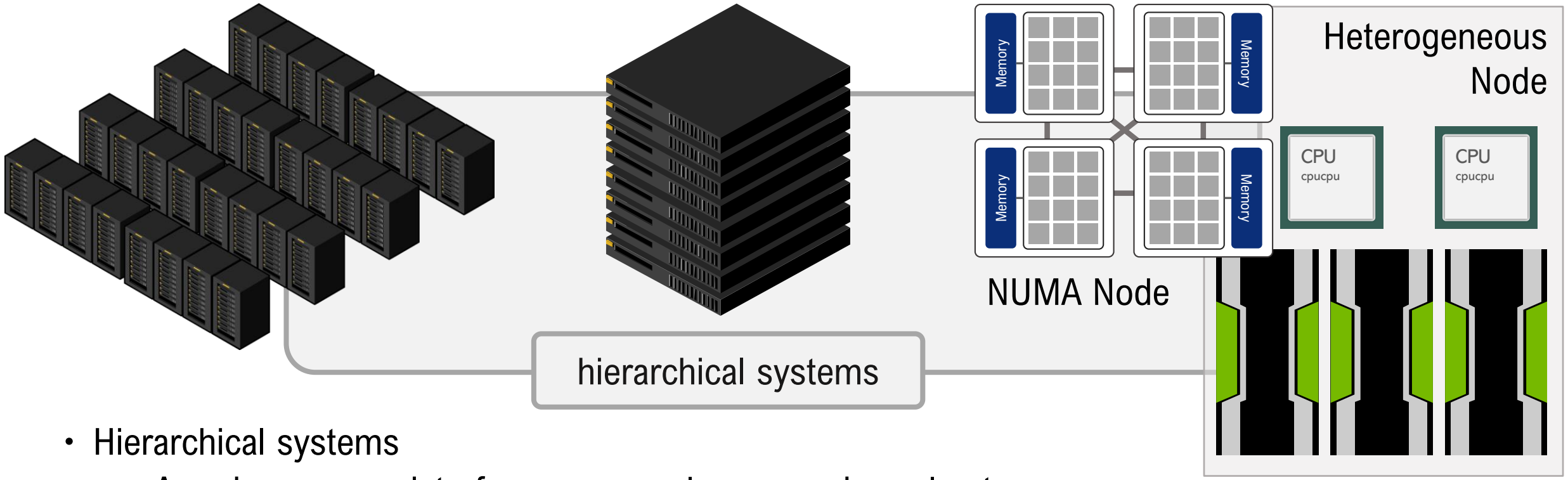
} Trilateral collaboration for scalable and productive computation



Overview of MUST



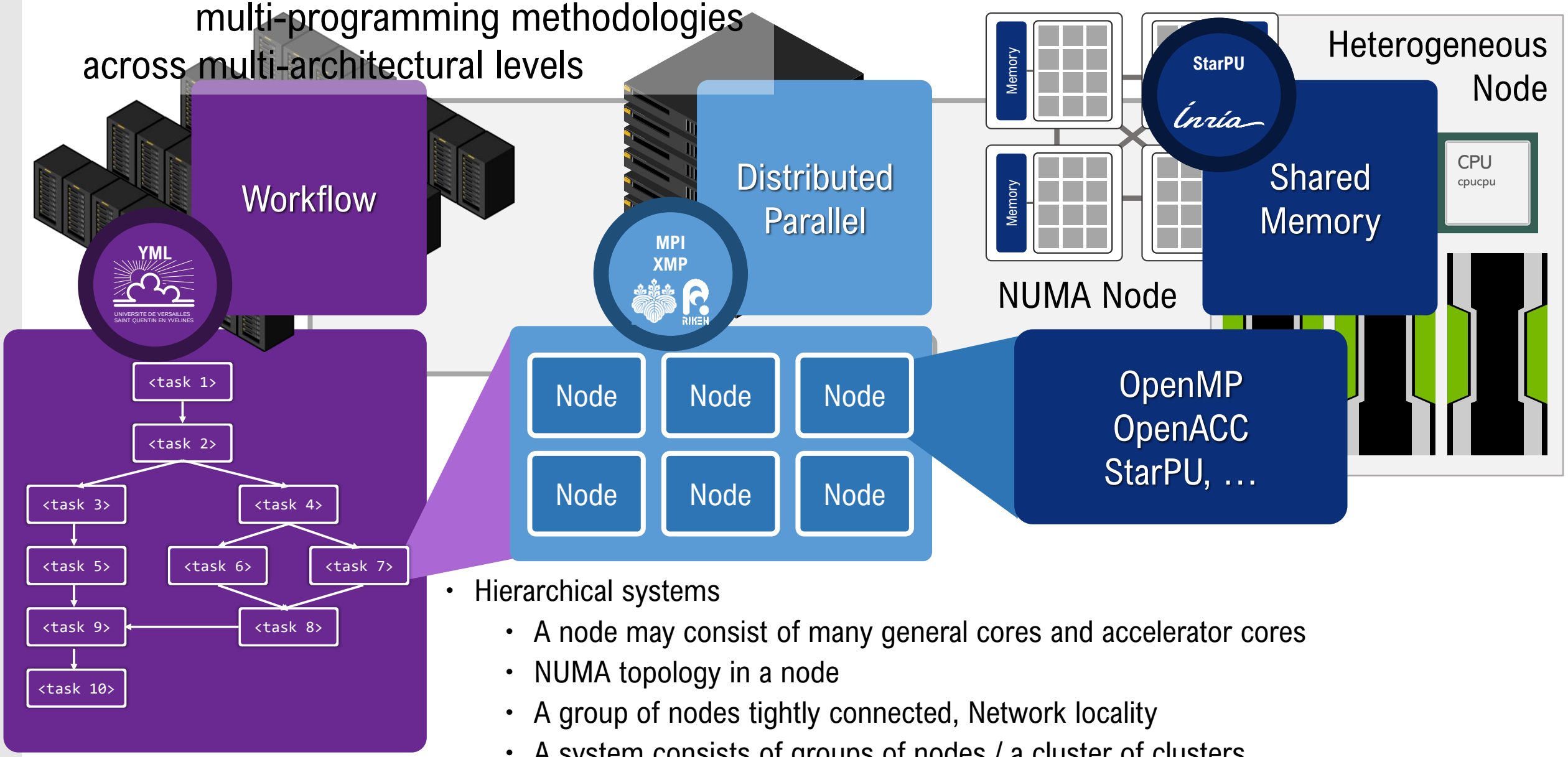
Multi SPMD (mSPMD) Programming Model / FP3C



- Hierarchical systems
  - A node may consist of many general cores and accelerator cores
  - NUMA topology in a node
  - A group of nodes tightly connected, Network locality
  - A system consists of groups of nodes / a cluster of clusters
- Multi-programming methodologies across multi-architectural levels
- Software had been developed to execute applications based on this programming model

Multi SPMD (mSPMD) Programming Model

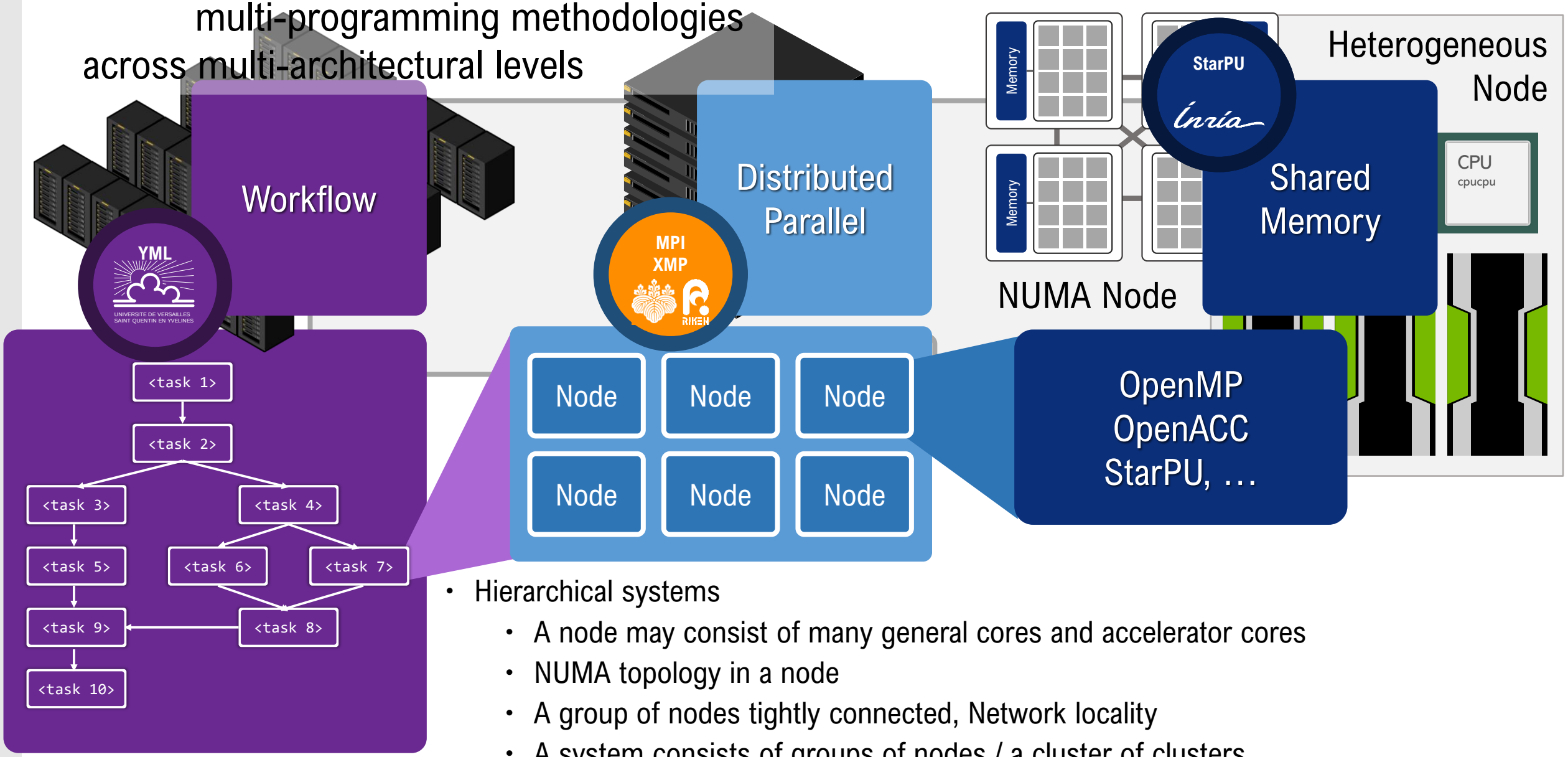
multi-programming methodologies
across multi-architectural levels



- Hierarchical systems
  - A node may consist of many general cores and accelerator cores
  - NUMA topology in a node
  - A group of nodes tightly connected, Network locality
  - A system consists of groups of nodes / a cluster of clusters

Multi SPMD (mSPMD) Programming Model

multi-programming methodologies
across multi-architectural levels



- Hierarchical systems
  - A node may consist of many general cores and accelerator cores
  - NUMA topology in a node
  - A group of nodes tightly connected, Network locality
  - A system consists of groups of nodes / a cluster of clusters

XcalableMP (XMP)

- Directive based parallel programming language
- Data distribution and work mapping can be declared by XMP directives
- XMP Compiler
  - Source-to-source compiler
  - C+XMP ⇒ C+XMP-runtime library call
  - The XMP runtime library uses MPI in its communication layer

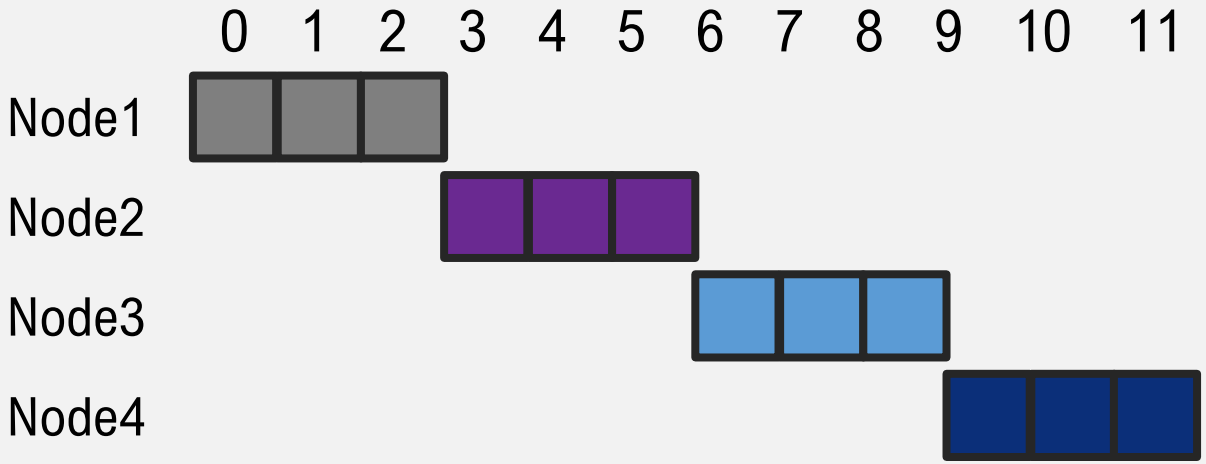
Data can be distributed over different processes of a task automatically

```
int B[12];

#pragma xmp nodes p(4)
#pragma xmp template t(0:11)
#pragma xmp distribute t(block) ont p
#pragma xmp align B[i] with t(i)
```

Data Mapping

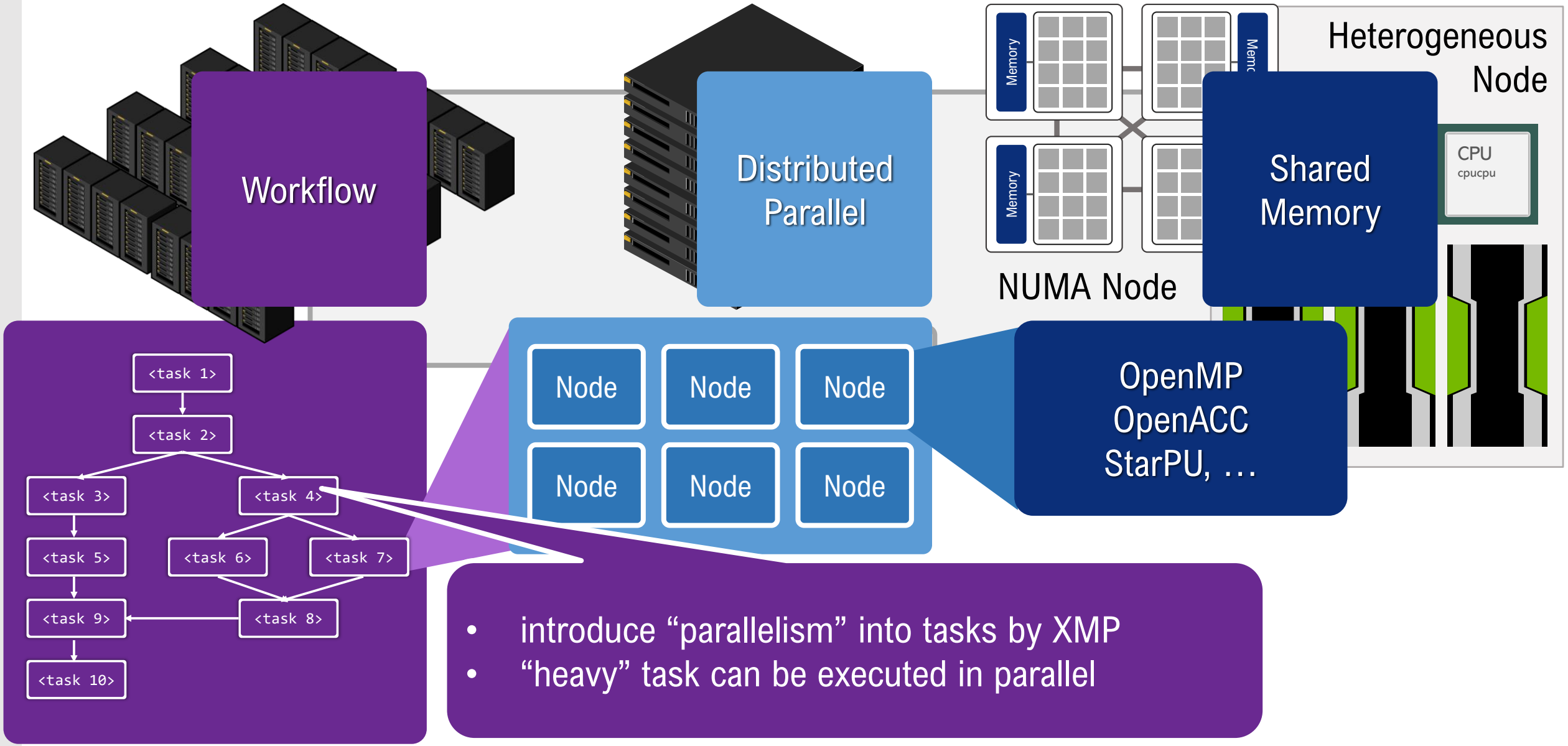
a one-dimensional block-distributed array B[] distributed over four nodes



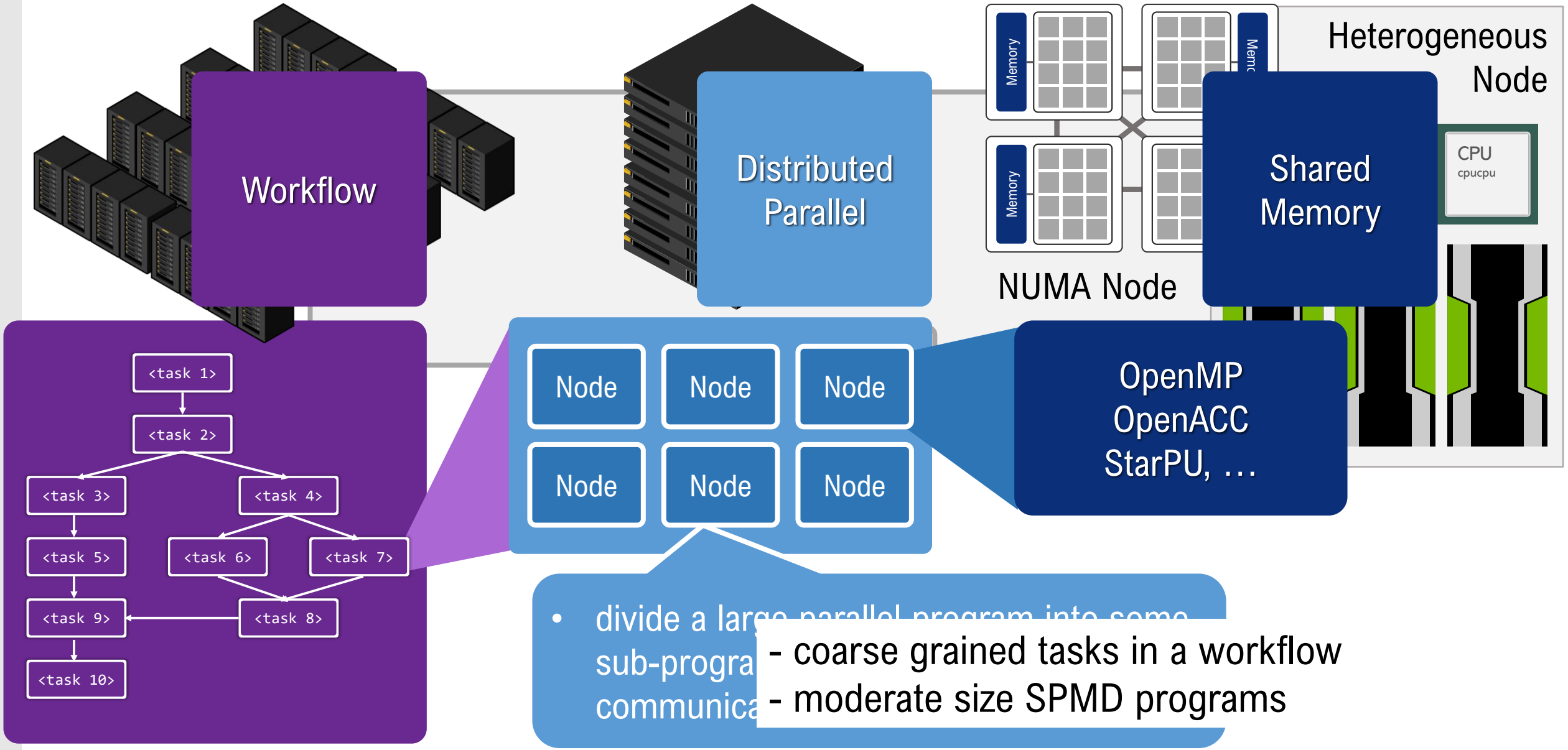

```
#pragma xmp loop (i) on t(i)
for(i=0; i<12; i++){
   B[i] = B[i]*2;
}
```

Work Mapping

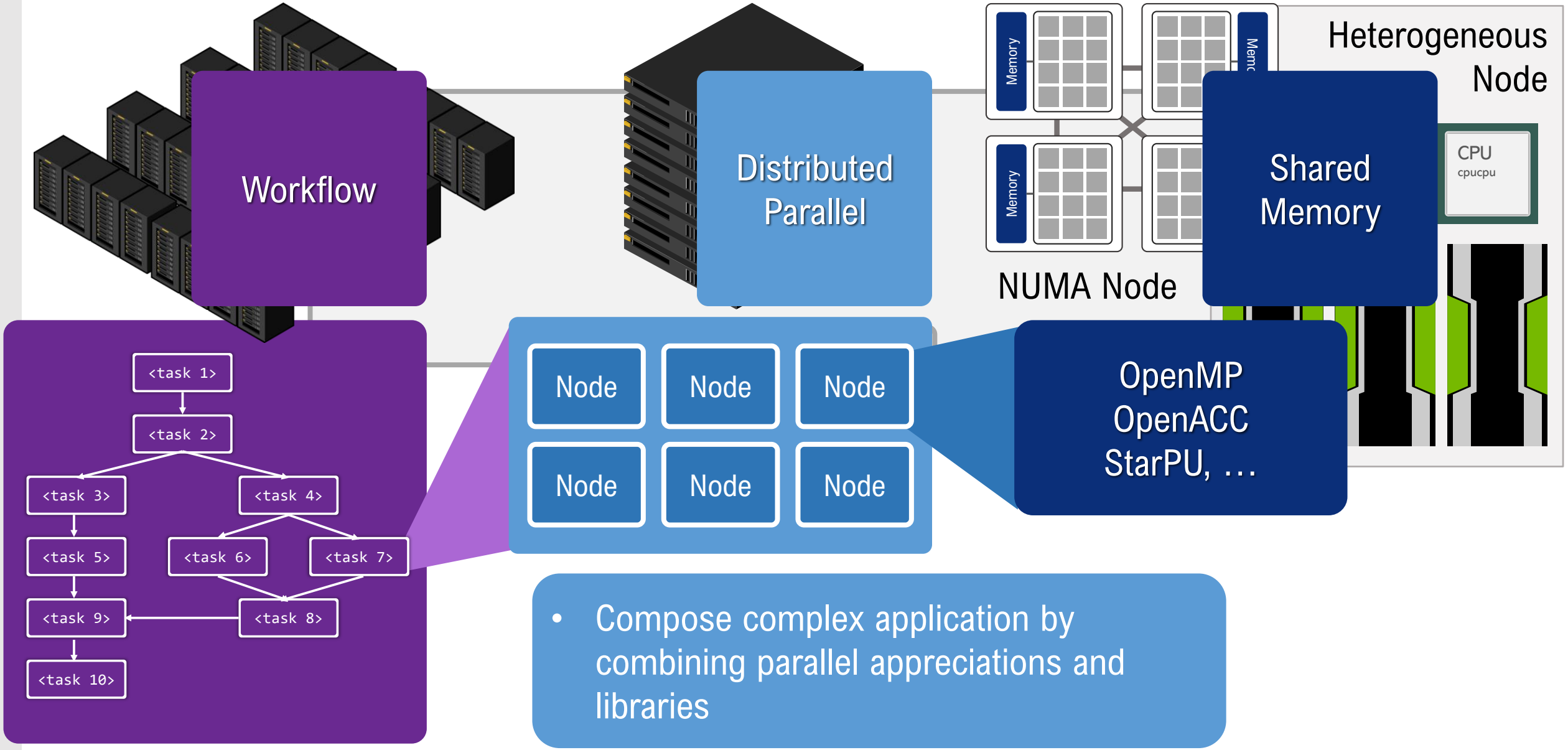
Multi SPMD (mSPMD) Programming Model



Multi SPMD (mSPMD) Programming Model

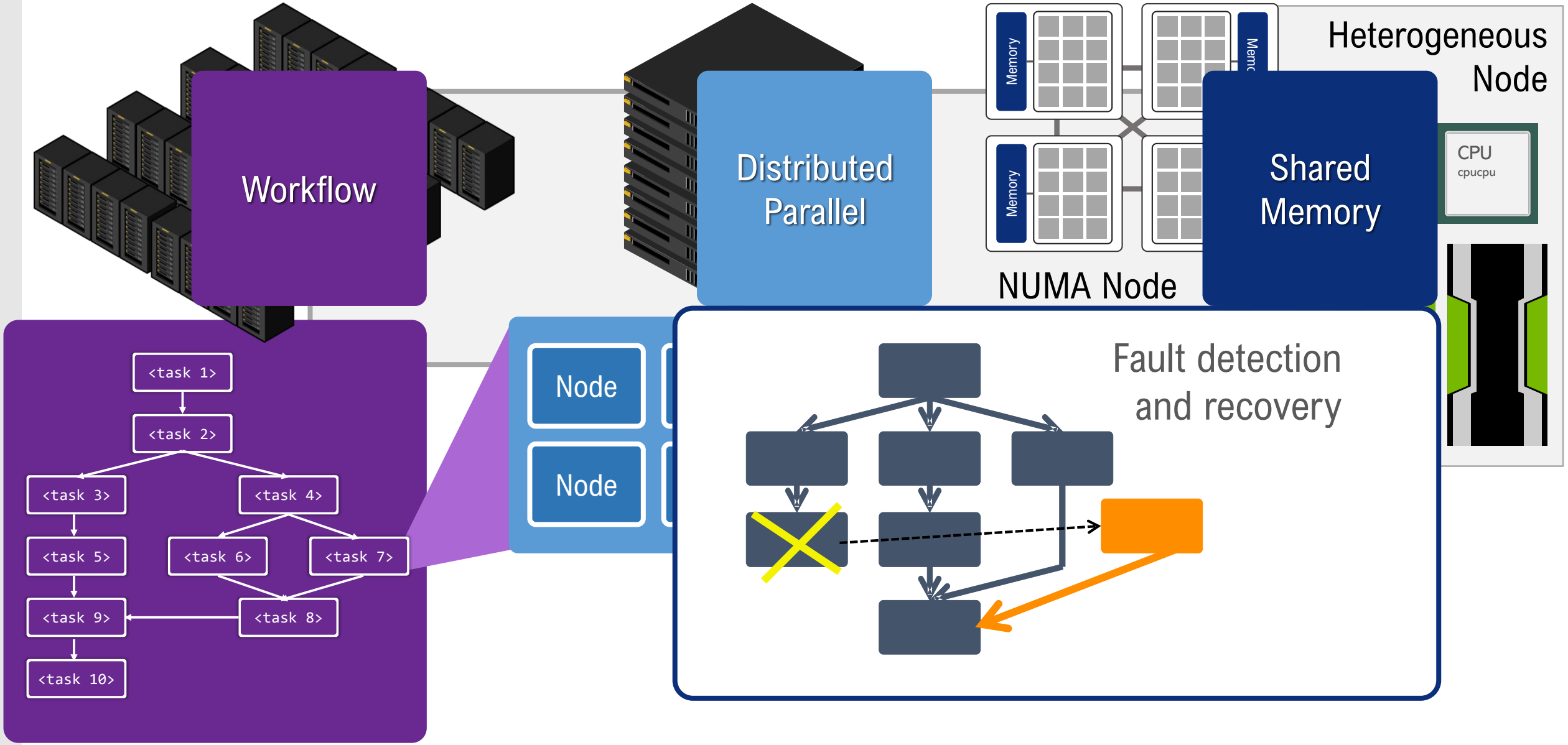


Multi SPMD (mSPMD) Programming Model



- Compose complex application by combining parallel appreciations and libraries

Multi SPMD (mSPMD) Programming Model

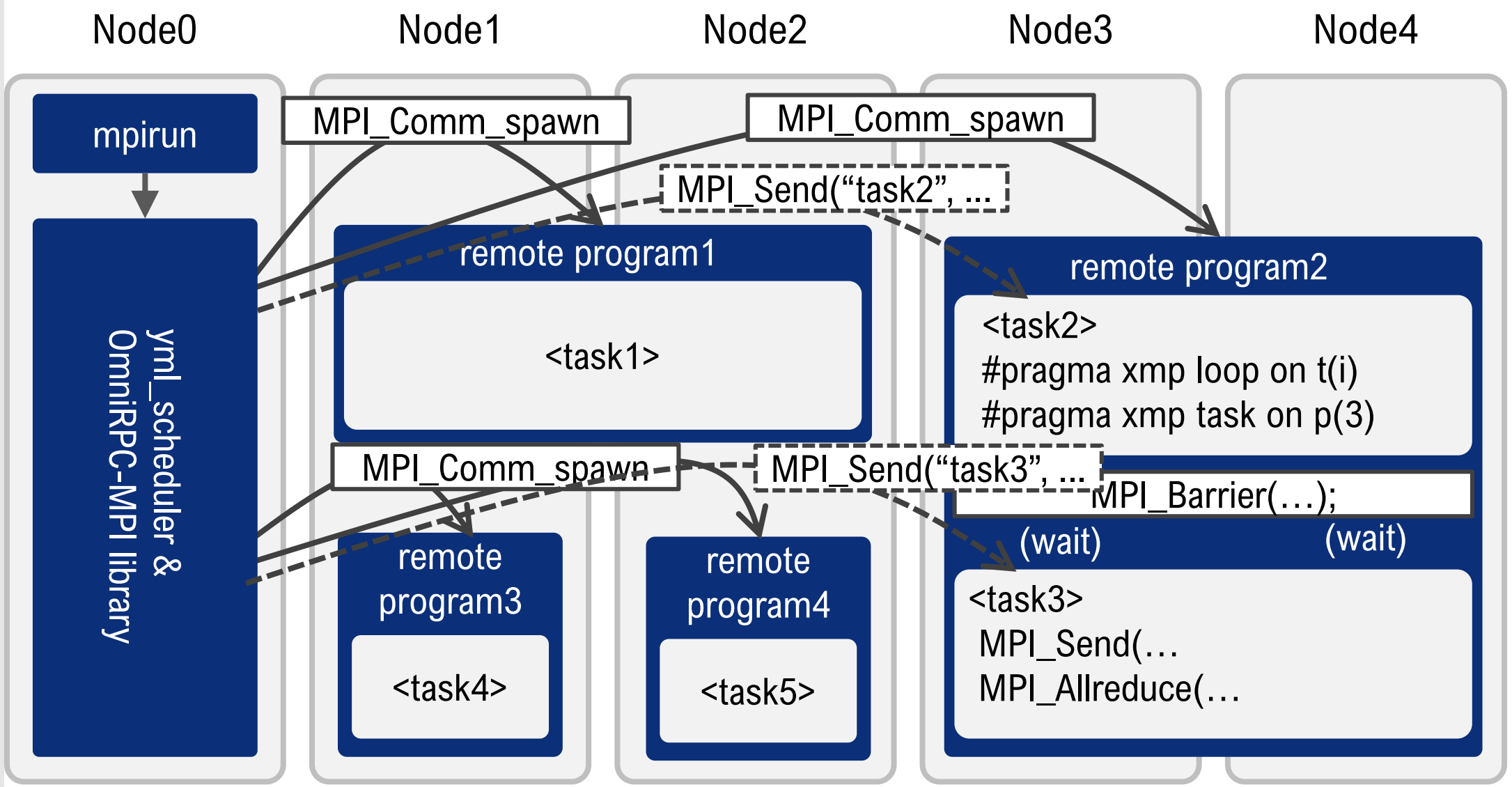


Multi SPMD (mSPMD) Programming Model

- Scalability
  - by combining different parallel programming paradigm across different architectural level
- Reliability
  - fault tolerant futures have been supported
- Productivity
  - **correctness checking by MUST library**

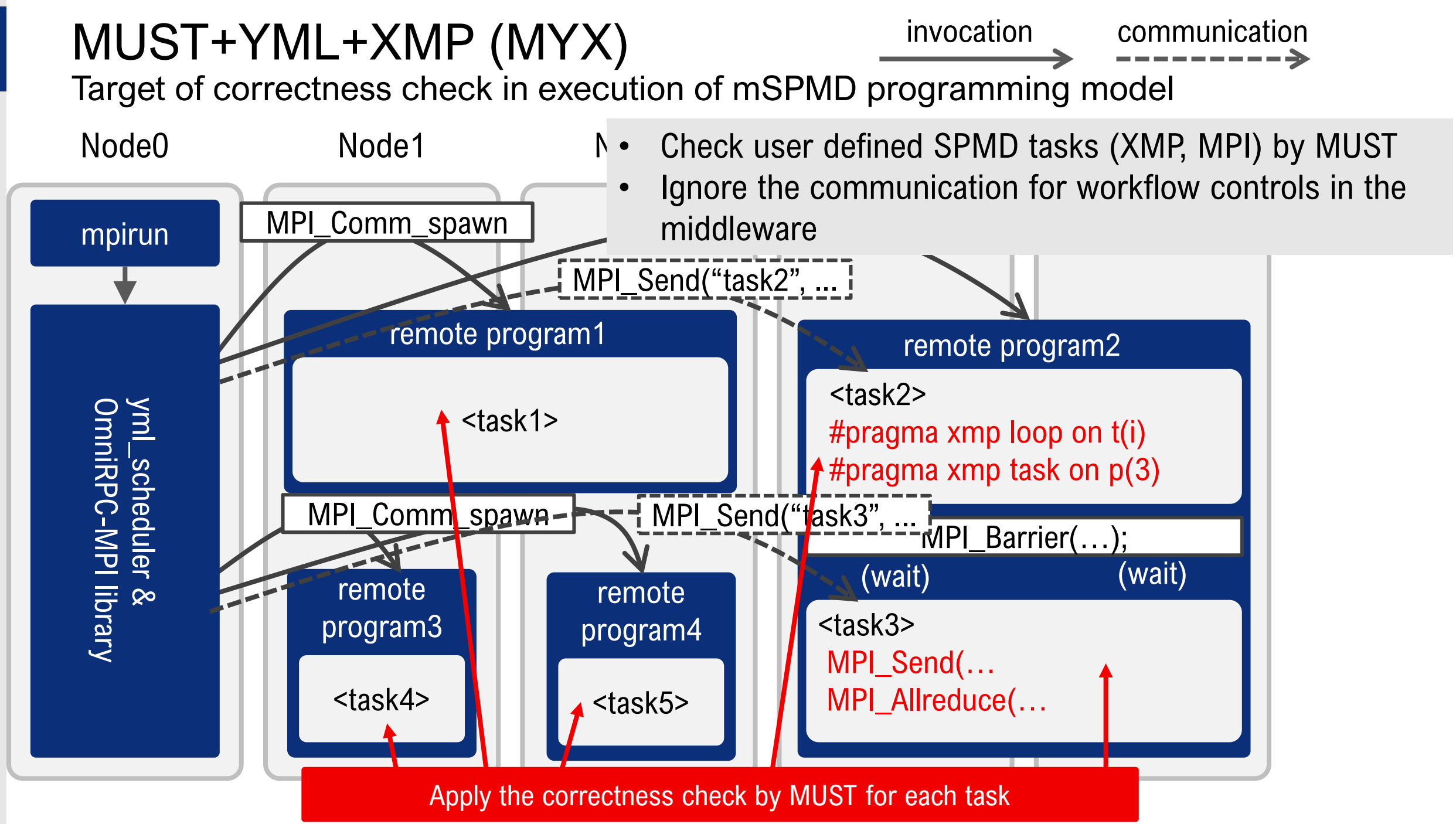
MUST+YML+XMP (MYX)

Overview of execution of mSPMD programming model



MUST+YML+XMP (MYX)

Target of correctness check in execution of mSPMD programming model



XMPT Tool Interface

- ... is a generic tool API of XMP.
- Its basic idea is inspired by *OMPT.*
  - event- and callback-based
- (Planned) targets:
  - *MYX* (SPPEXA project by RWTH Aachen, UVSQ, and R-CCS)
  - *Extrae* @ BSC
  - *Score-P* / *Scalasca* @ JSC
  - etc.

Basic Design of XMPT

- At initialization

Callbacks are registered through `xmpt_set_callback`.

```
void xmpt_set_callback(...);
```

Provided by an XMP compiler.

```
void xmp_init(){
 xmpt_initialize(...);
...
}
```

Provided by tools

```
void xmpt_initialize(...){
 xmpt_set_callback(XMPT_BCAST_BEGIN, myx_bcast_begin);
 xmpt_set_callback(XMPT_BCAST_END, myx_bcast_end);
 ...
}
```

`xmp_init` invokes `xmpt_initialize`.

```
void xmpt_initialize(...) __attribute__((weak));
```

- At each event

The registered callbacks are invoked.

```
void xmp_bcast(...){
 (*xmpt_bcast_begin)(...);
 xmp_bcast_body(...);
 (*xmpt_bcast_end)(...);
}
```

```
void
myx_bcast_begin(...);
```

```
void
myx_bcast_end(...);
```

MUST+YML+XMP (MYX): Implementation

- MUST+MPI / MUST+XMP : to check a single SPMD program
  - **mustrun** –np n application.exe
  - prepare a dedicated dynamic library for the application.exe, set the environmental variables
  - mpirun –np $(n+1)$ application.exe: 1 process should be kept for the MUST analysis
- MUST+YML+MPI/XMP: to check multiple SPMD program
  - Instead of mustrun (mpirun), MPI_Comm_spawn is used to invoke remote SPMD programs in mSPMD
  - extend the middleware of workflow scheduler and the remote program generator in mSPMD
    - MPI_* functions in the workflow control are replaced with PMPI_* functions
    - MPI_Comm_spwan(“prog”, **n**, …) → PMPI_Comm_spwan(“prog”, **n+1**, …)
  - preparation steps performed within the mustrun script before mpirun should be performed before starting a workflow
    - set the environmental variables required by MUST manually (Originally, they are set by the mustrun scprit)
    - prepare a dedicated dynamic library to analyze each remote program

Experiments on Oakforest-PACS

- Compare the behavior of workflow applications w/ and w/o error, w/ and w/ MUST
- Evaluate the overhead to apply MUST for tasks in a mSPMD application
- Oakforest-PACS (OFP): supercomputer installed in Kashiwa, operated by U. Tokyo and U. Tsukuba
  - 8208 KNL nodes, Connected via Intel Omni Path
  - Compiler intel/2018.1.163
  - MPI impi/2018.1.163
- 30 processes (flat-MPI) for each task, 1, 2, 4, 8, 16, 32 tasks in each application, all tasks are run simultaneously

Test codes

Allreduce

```
for(i=0; i<100; i++){
  MPI_Allreduce(buf, rbuf, 1, MPI_LONG, MPI_SUM, MPI_COMM_WORLD);
  usleep(100000);
}
```

Allreduce: Type conflict

```
for(i=0; i<100; i++){
  if(myrank==0)
    MPI_Allreduce(buf, rbuf, 1, MPI_INTEGER, MPI_SUM,
MPI_COMM_WORLD);
  else
    MPI_Allreduce(buf, rbuf, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
  usleep(100000);
}
```

Test codes (cont.

Allreduce: Operation conflict

```
for(i=0; i<100; i++){
  if(myrank==0)
    MPI_Allreduce(buf, rbuf, 1, MPI_LONG, MPI_MAX, MPI_COMM_WORLD);
  else
    MPI_Allreduce(buf, rbuf, 1, MPI_LONG, MPI_MIN, MPI_COMM_WORLD);
  usleep(100000);
}
```

Allreduce: Size conflict

```
for(i=0; i<100; i++){
  if(myrank==0)
    MPI_Allreduce(buf, rbuf, 1, MPI_LONG, MPI_SUM, MPI_COMM_WORLD);
  else
    MPI_Allreduce(buf, rbuf, 2, MPI_LONG, MPI_SUM, MPI_COMM_WORLD);
  usleep(100000);
}
```

Test codes (cont.

Pingpong

```
for(i=0; i<100; i++){
  if(myrank%2==0) MPI_Send(buf, 1, MPI_LONG, dest, tag, MPI_COMM_WORLD);
  else            MPI_Recv(buf, 1, MPI_LONG, src , tag, MPI_COMM_WORLD, ..);
  usleep(100000);
  if(myrank%2==0) MPI_Recv(buf, 1, MPI_LONG, src , tag, MPI_COMM_WORLD,..);
  else            MPI_Send(buf, 1, MPI_LONG, dest, tag, MPI_COMM_WORLD);
  usleep(100000);}
```

Pingpong, Type conflict

```
for(i=0; i<100; i++){
  if(myrank%2==0) MPI_Send(buf, 1, MPI_UNSINGED_LONG, dest, tag, MPI_COMM_WORLD);
  else            MPI_Recv(buf, 1, MPI_LONG, src , tag, MPI_COMM_WORLD, ..);
  usleep(100000);
  if(myrank%2==0) MPI_Recv(buf, 1, MPI_LONG, src , tag, MPI_COMM_WORLD,..);
  else            MPI_Send(buf, 1, MPI_LONG, dest, tag, MPI_COMM_WORLD);
  usleep(100000);}
```

Result (1) Status

	w/ MUST		w/o MUST	
	completed?	reported?	completed?	reported?
allreduce w/o error	completed	-	completed	-
allreduce type conflict	completed	error report	completed	**no**
allreduce operation conflict	completed	error report	completed	**no**
allreduce size conflict	failed	error report	failed	simple error report
pingpong w/o error	completed	-	completed	-
pingpong type conflict	completed	error report	completed	**no**

Result (2) Example of error report from MUST

MUST Output, starting date: Tue Jan 22 19:28:40 2019.

Rank(s)	Type	Message
1	**Error**	A send and a receive operation use datatypes that do not match! Mismatch occur···

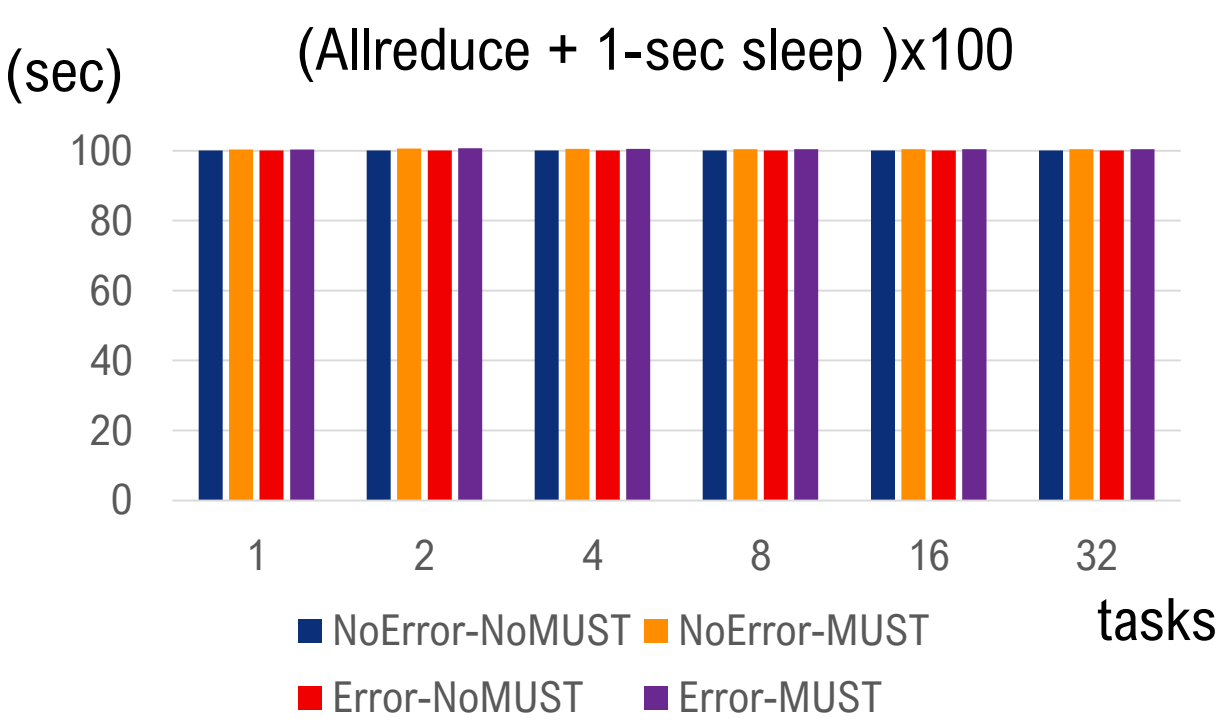
Details:

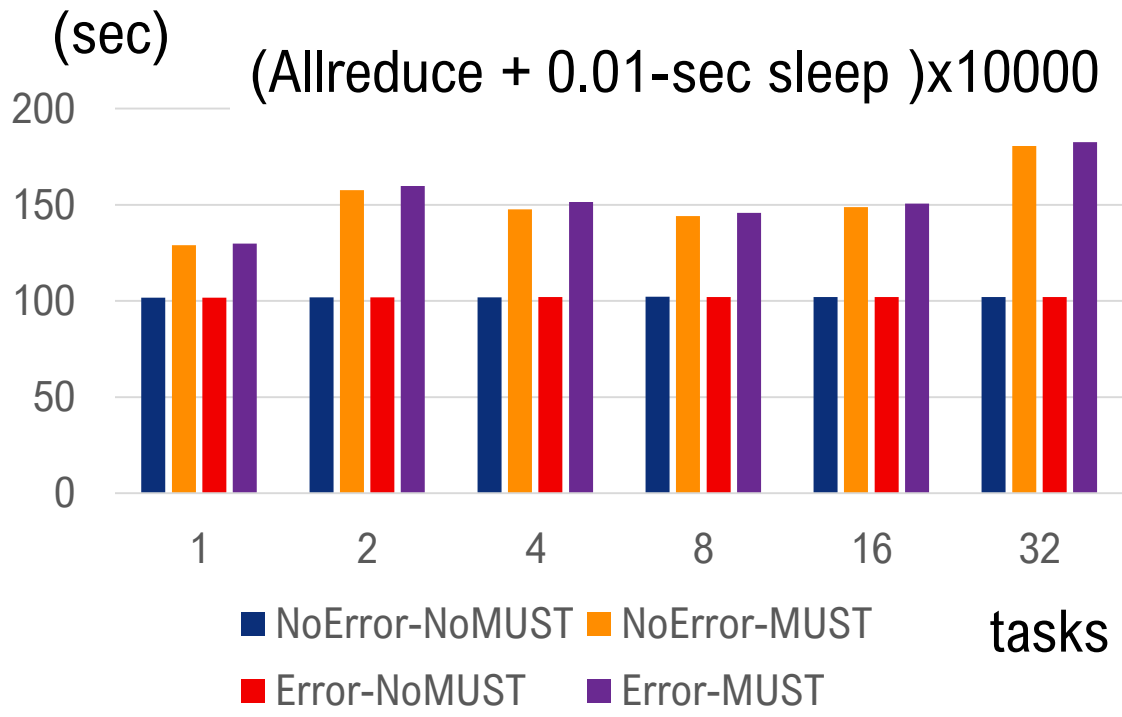
Message	From	References
A send and a receive operation use datatypes that do not match! Mismatch occurs at (MPI_INTEGER) in the send type and at (MPI_INT) in the receive type (consult the MUST manual for a detailed description of datatype positions). A graphical representation of this situation is available in the file named "MUST_Output-files/MUST_Typemismatch_1.dot". Use the dot tool of the graphviz package to visualize it, e.g. issue "dot -Tps MUST_Output-files/MUST_Typemismatch_1.dot -o mismatch.ps". The graph shows the nodes of the involved Datatypes that form the root cause of the type mismatch. The send operation was started at reference 1, the receive operation was started at reference 2. (Information on communicator: MPI_COMM_WORLD) (Information on send of count 1 with type:MPI_INTEGER) (Information on receive of count 1 with type:MPI_INT)	Representative location: call MPI_Recv (1st occurrence)	References of a representative process: reference 1 rank 0: call MPI_Send (1st occurrence) reference 2 rank 1: call MPI_Recv (1st occurrence)

3	**Error**	A send and a receive operation use datatypes that do not match! Mismatch occur···
0	**Error**	A send and a receive operation use datatypes that do not match! Mismatch occur···
0-3	**Warning**	You requested 12 threads by OMP_NUM_THREADS but used MPI_Init to start your ap···
0-3	**Error**	Argument 1 (comm) is an unknown communicator where a valid communicator was ex···
3	**Error**	There are 16 communicators that are not freed when MPI_Finalize was issued, a ···
1	**Error**	There are 16 communicators that are not freed when MPI_Finalize was issued, a ···
2	**Error**	There are 16 communicators that are not freed when MPI_Finalize was issued, a ···
0-3	**Error**	There are 2 operations that are not freed when MPI_Finalize was issued, a qual···
0	**Error**	There are 16 communicators that are not freed when MPI_Finalize was issued, a ···

MUST has completed successfully, end date: Tue Jan 22 19:28:41 2019.

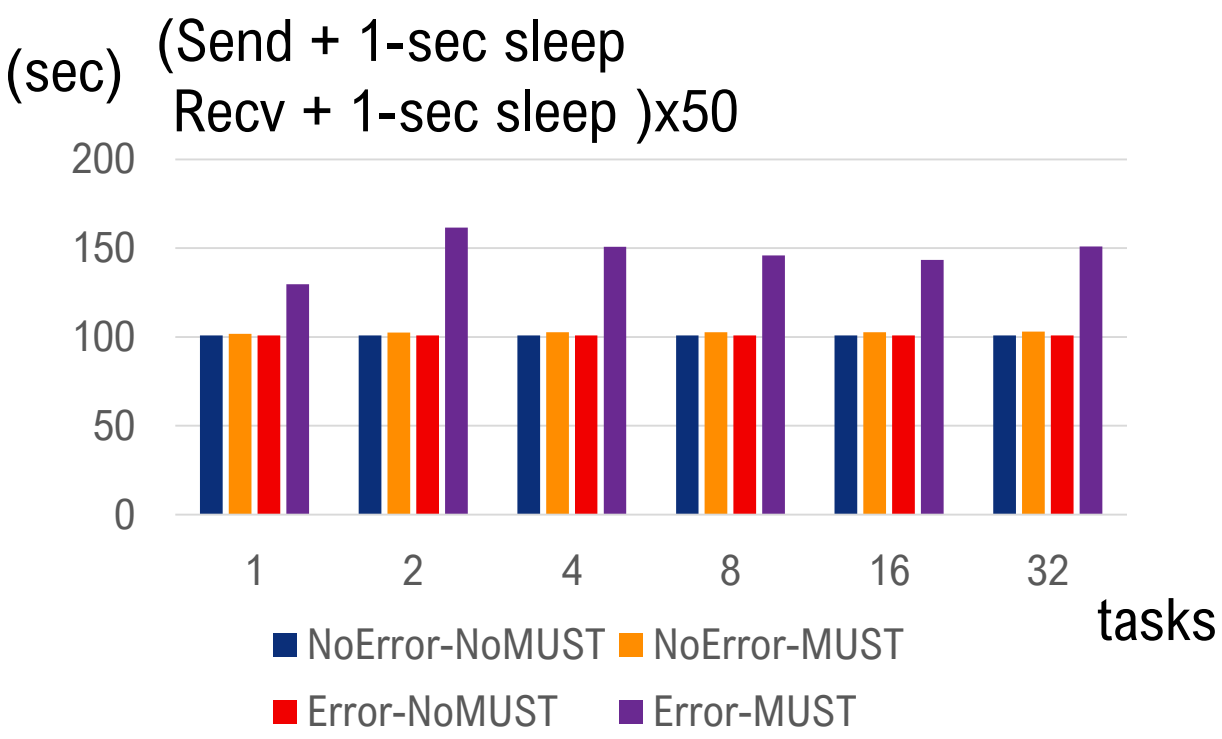
Result (3) Overhead: MPI_Allreduce

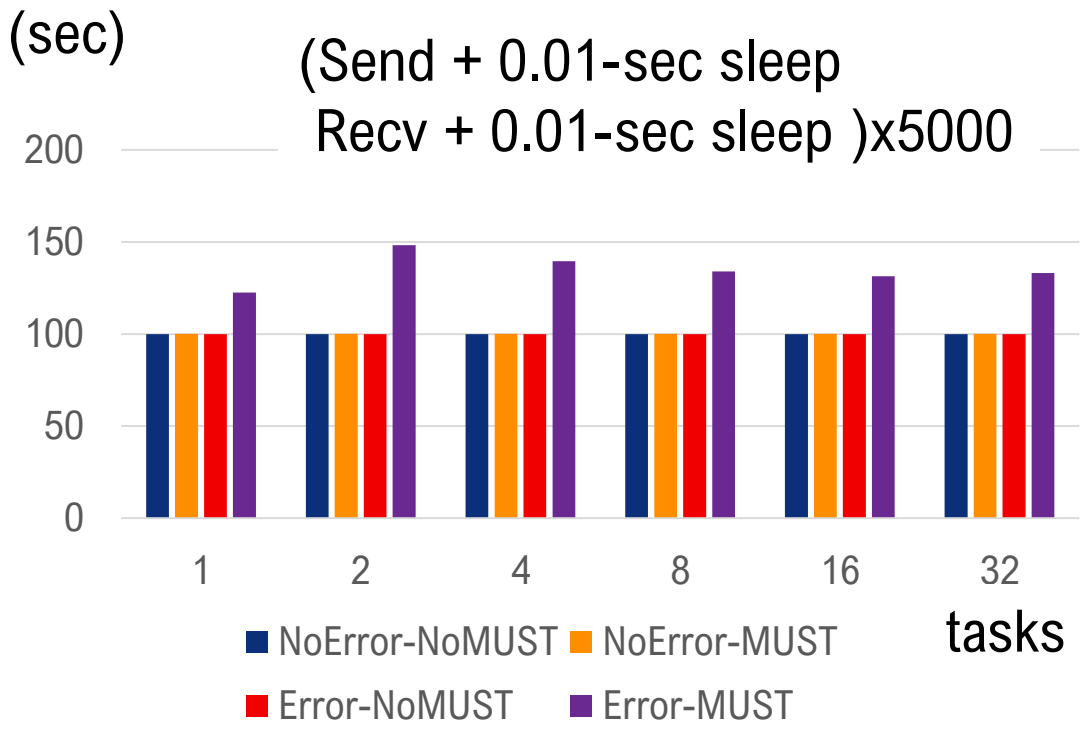




- The overheads depend on the frequency of the communication
  - The overhead is ignorable if we don't perform communication very intensively
- Some overheads even if there is no error if we call MPI_allreduce 100 times per second

Result (3) Overhead: pingpong



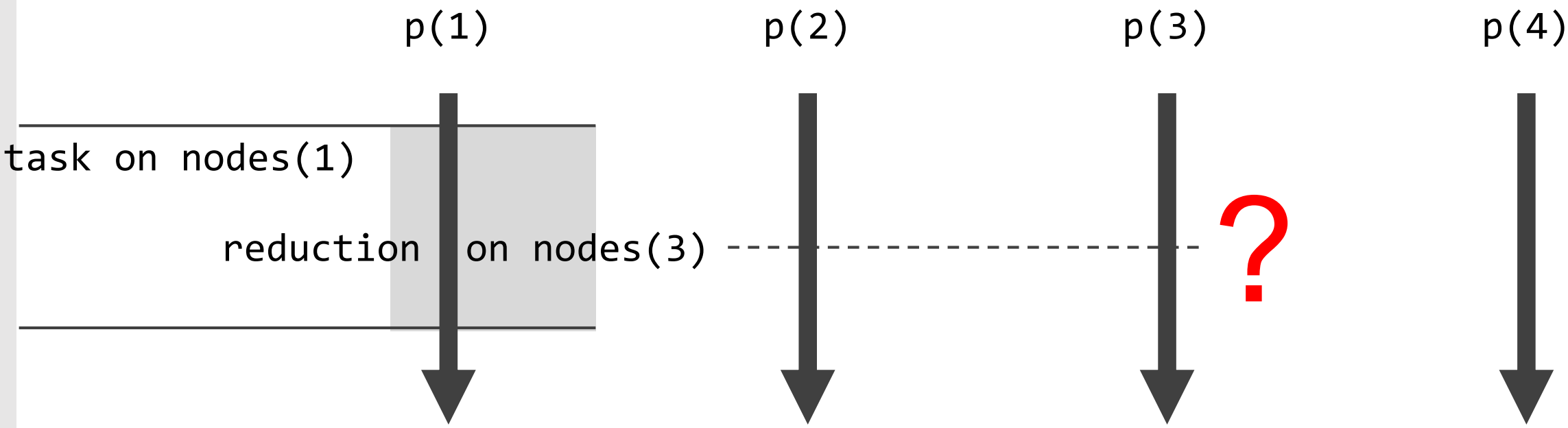


- No overhead if there is no error
- The overhead to **record** errors in the point-to-point communications is large (even when 1 point-to-point communication per second) due to the complexity of MPI function call dependencies

Experiments for XMP-tasks: Test codes

Uncorrect: reduction out of nodes

```
#pragma xmp task on nodes(1)
{
#pragma xmp reduction (+:sum) on nodes(3)
}
```



Experiments for XMP-tasks: Results

```
[c5093.ofp:**pp_4.rex:16c4b] Ninf_stub_SET_ARG(1)=0x1765140, 0x1765140
[c5093.ofp:**pp_4.rex:16c4b] Ninf_stub_SET_ARG(2)=0x17653c0, 0x17653c0
[c5093.ofp:**pp_4.rex:16c4b] Ninf_stub_BEGIN
[CurrentNodesTrack [CurrentNodesTrack 2 ]  initialNodes (2)
[CurrentNodesTrack 2 ] is subset: isInCurrentNodeSet ()
[CurrentNodesTrack 3 ]  initialNodes (2)
[CurrentNodesTrack 3 ] is subset: isInCurrentNodeSet ()
[CurrentNodesTrack 3 ] selectNodes (2, [1:1:1]): 0xe15300 / 2
1 ]  initialNodes (2)
[CurrentNodesTrack 1 ] is subset: isInCurrentNodeSet ()
[CurrentNodesTrack 1 ] selectNodes (2, [1:1:1]): [CurrentNodesTrack 2 ] s
[CurrentNodesTrack 2 ]  popNodes ()
[CurrentNodesTrack 3 ]  popNodes ()
0x2928300 / 2
[CurrentNodesTrack 1 ]  popNodes ()
[CurrentNodesTrack 0 ]  initialNodes (2)
[CurrentNodesTrack 0 ] is subset: isInCurrentNodeSet ()
[CurrentNodesTrack 0 ] selectNodes (2, [1:1:1]): 0x176d800 / 2
[CurrentNodesTrack 0 ] is no subset: isInCurrentNodeSet ()
[CurrentNodesTrack 0 ] selectNodes (2, [3:3:1]): 0x176d800 / 2
[CurrentNodesTrack 0 ]  popNodes ()
[CurrentNodesTrack 0 ]  popNodes ()
[c5093.ofp:**pp_4.rex:16c4b] Ninf_stub_END begin
[c5093.ofp:**pp_4.rex:16c4b] Ninf_stub_END send out args
```

Conclusion and future works

- MYX: an international collaborative project for higher productivity in exascale computing.
  Runtime correctness check by MUST for multi SPMD Programming Model by YML+XMP
  - MUST is a correctness checking tool.
  - YML is a workflow language (to be presented by Miwako)
  - XMP is a directive-based PGAS extension for Fortran & C supporting the global- and local-view programming.
- XMP+MUST
  - XMP provides an interfere, XMPT, for performance tools
  - MUST uses the XMPT and check the correctness of XMP
- XMP+YML
  - Tasks written in XMP of a workflow managed by YML
- MUST+YML+XMP
  - The task generator and middleware in mSPMD have been extended

  ⇒ Scalable, reliable programming model with high productively

  Scalable : Combination of multiple-SPMDs by YML and XMP

  Reliable : Fault-detection and recovery are supported

  High Productively : XMP, YML are easier than C+MPI

  MUST and XMPT provide a debug tool for SPMD
- future works for society 5.0, and IoT era