FAULT TOLERANCE FEATURES OF A NEW MULTI-SPMD PROGRAMMING/EXECUTION ENVIRONMENT

Miwako Tsuji (RIKEN AICS)
Serge Petiton (CNRS/LIFL)
Mitsuhisa Sato (RIKEN AICS/University of Tsukuba)

AICS

Challenges in post-petascale supercomputing

- Scalability
- Programmability
- Fault tolerance
- Energy efficiency
- Big Data
- etc….

AICS

Agenda

Introduction

Background : Multi-SPMD Programming/Execution Environment

FT in multi-SPMD

Experiments

Related work

Conclusion

AICS

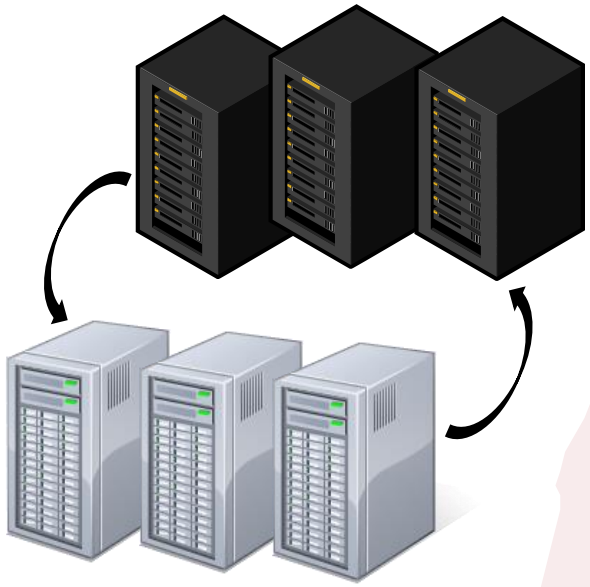
Introduction

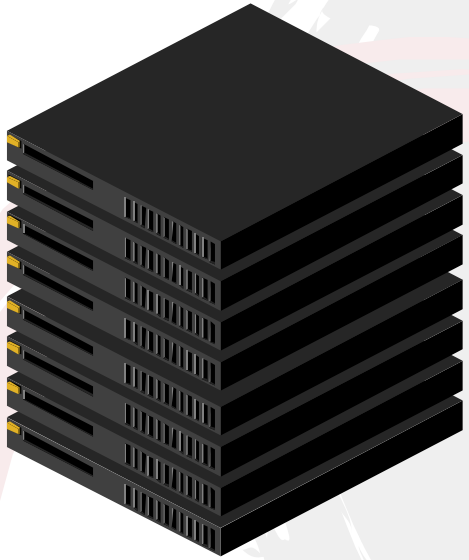
- Supercomputers in the exa-scale era would be
  - huge number of nodes arranged in a multi-level hierarchy
  - complex

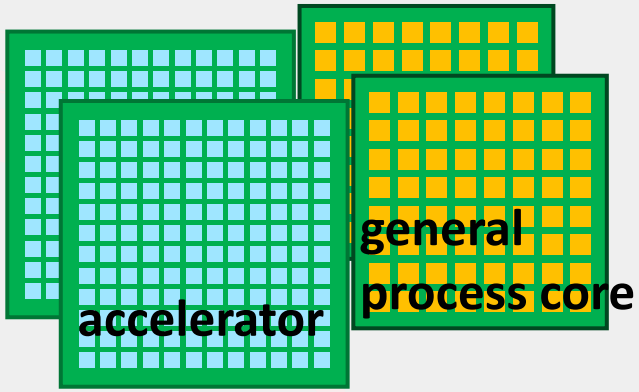
    → reduce MTBF (Mean time between failure)
- Fault tolerance is important
- But, it's a hassle
  - rewrite source codes for FT!?
- We develop a fault tolerance mechanism in a multi-SPMD programming / execution environment (called FP2C)
  - Without any modification in application source codes

AICS

Background FP2C



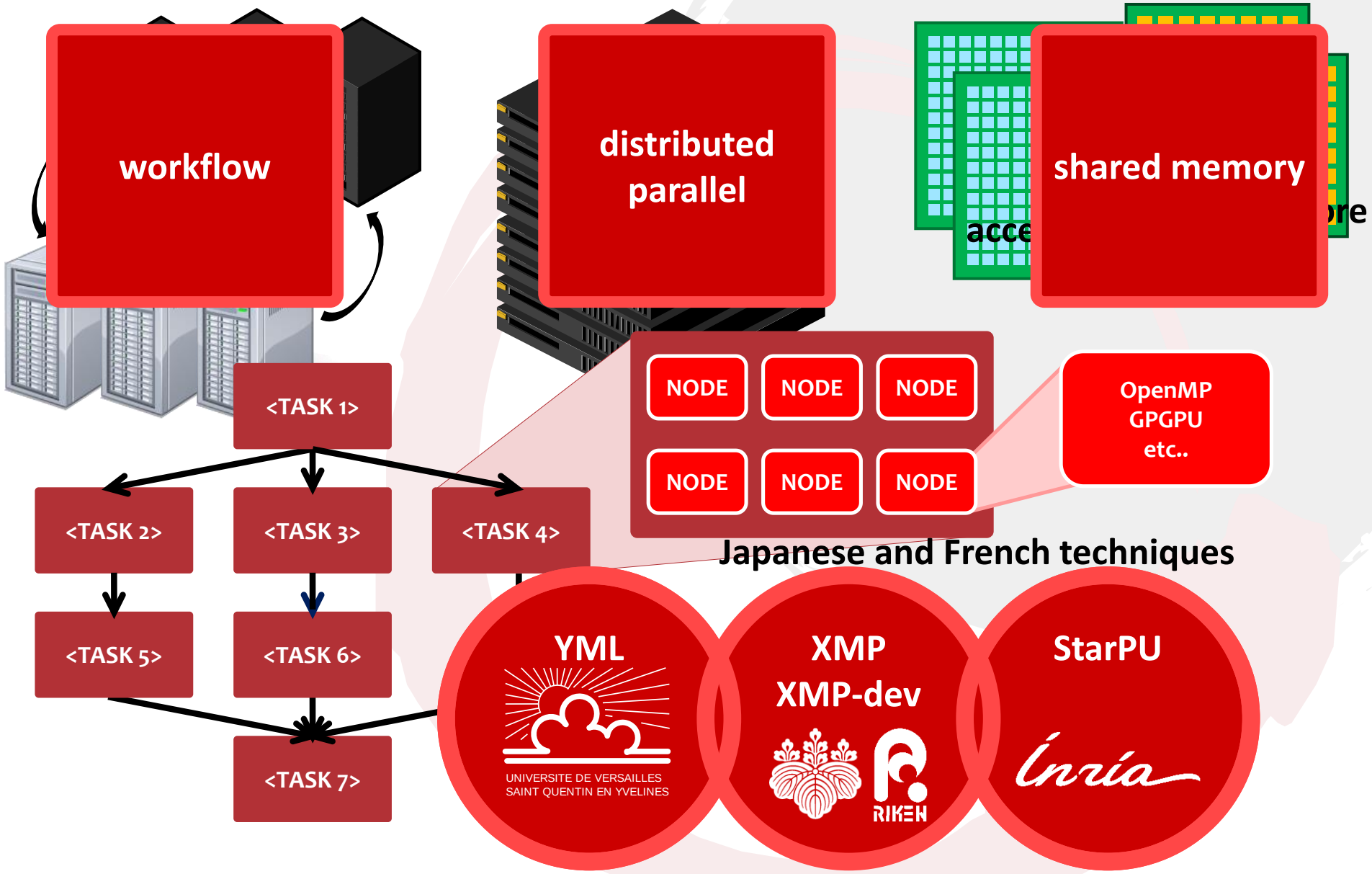




- Hierarchical systems
  - A node may consist of many general cores and accelerator cores
  - A group of nodes tightly connected
  - A system consists of groups of node / a cluster of clusters
- Multi-programming methodologies across multi-architectural levels
- FP2C had been proposed to execute applications based on this programming model

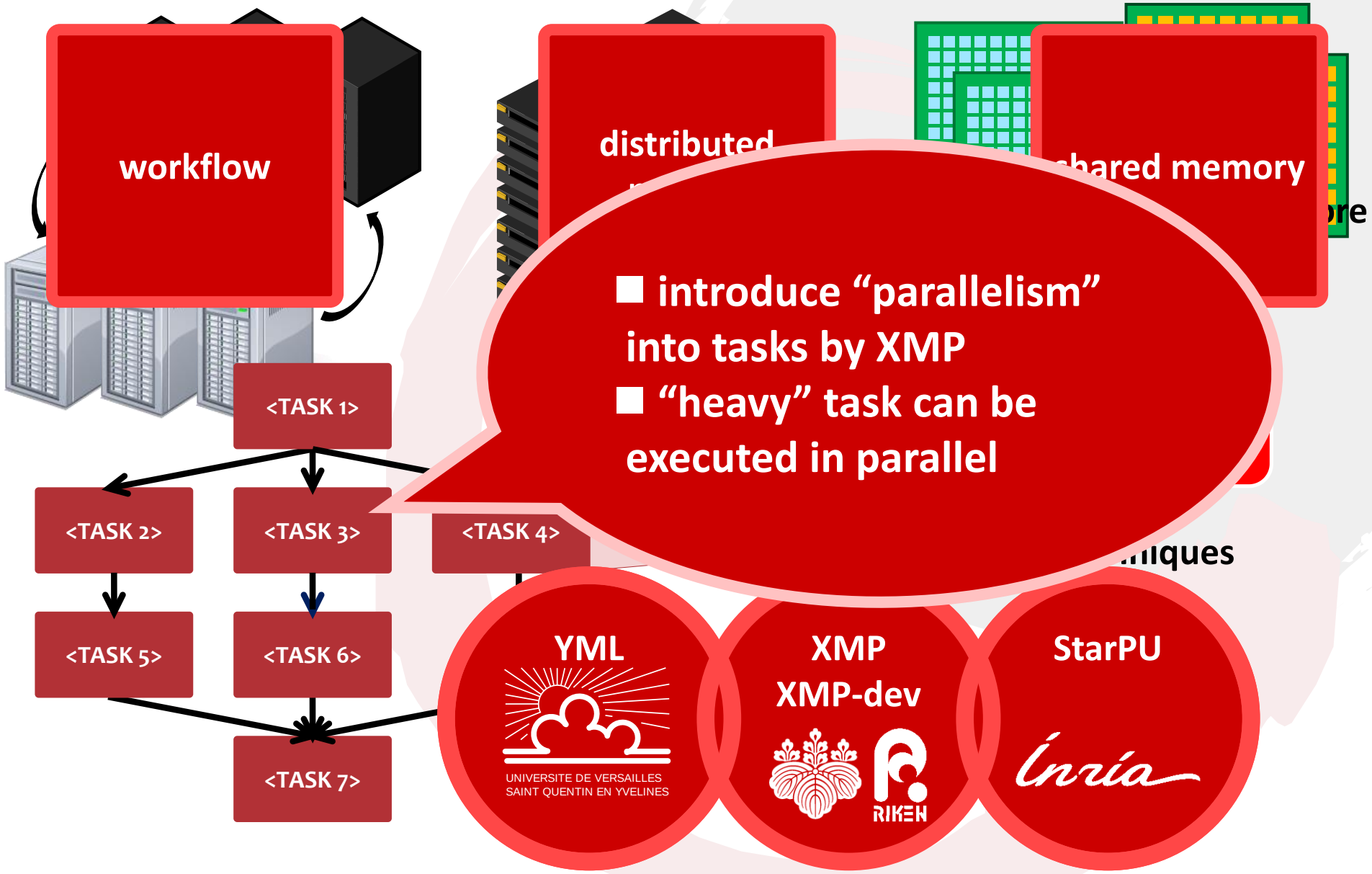
AICS

Background FP2C



AICS

Background FP2C

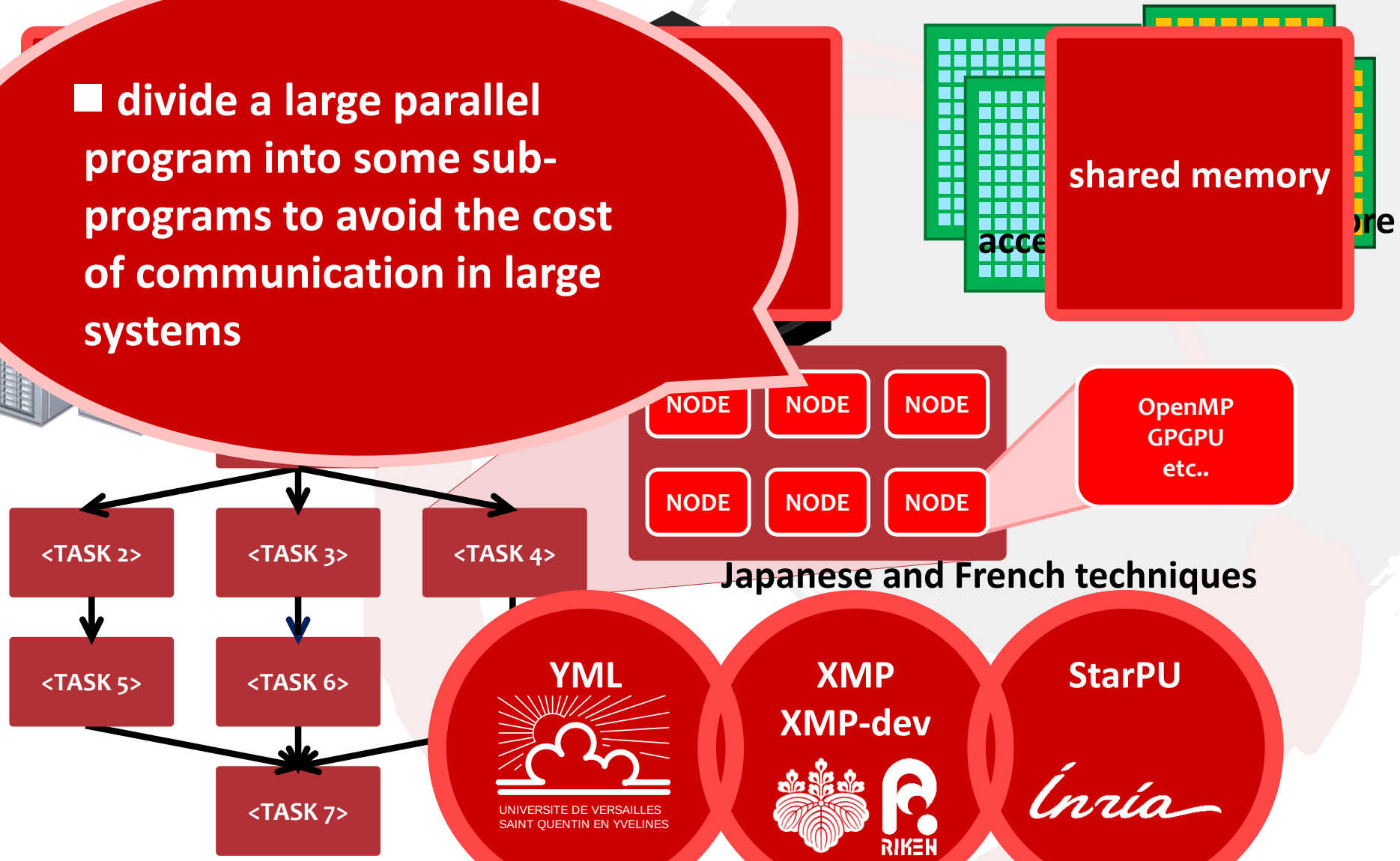


- introduce “parallelism” into tasks by XMP
- “heavy” task can be executed in parallel

AICS

Background FP2C

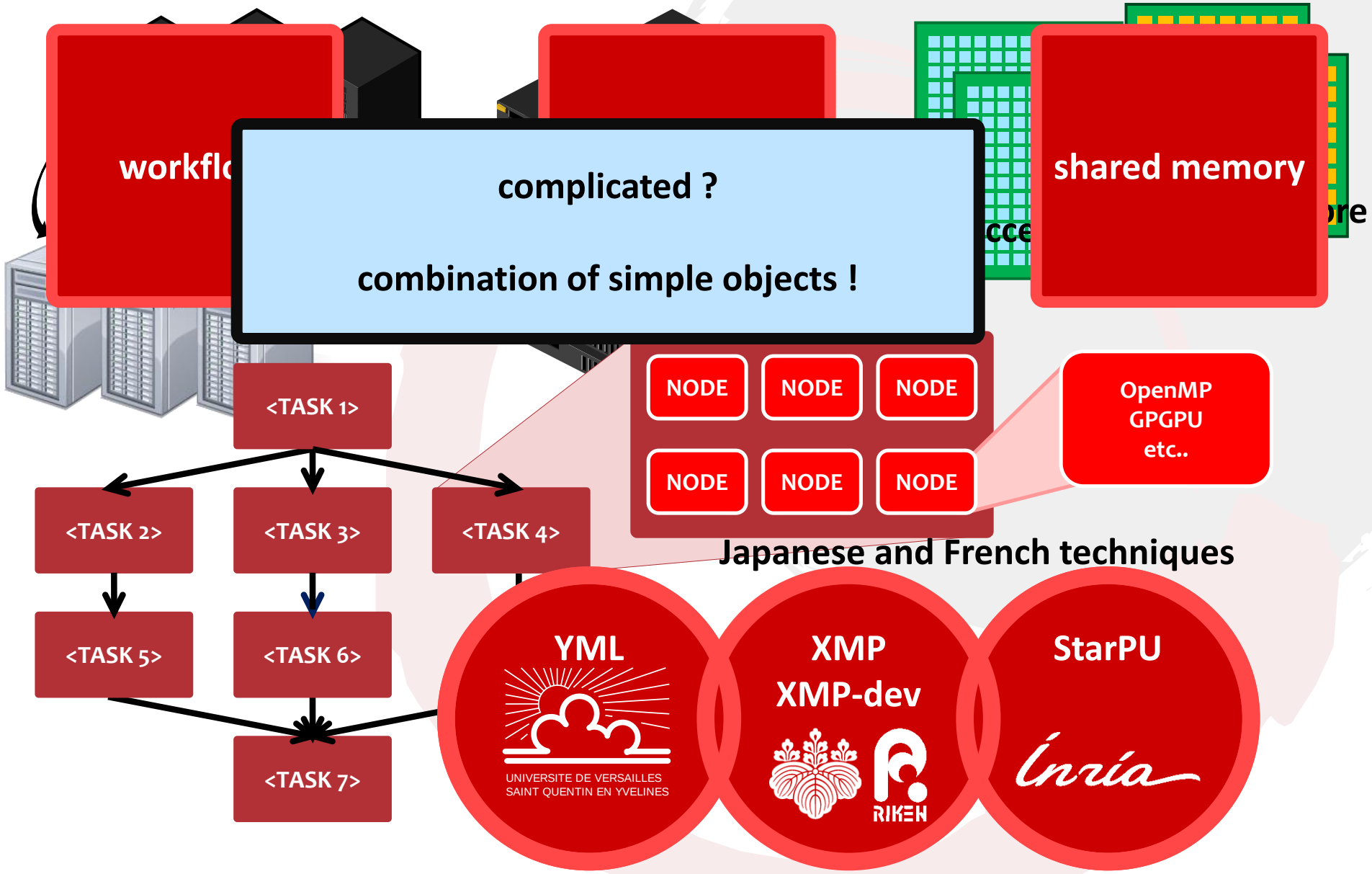
- divide a large parallel program into some sub-programs to avoid the cost of communication in large systems



Japanese and French techniques

AICS

Background FP2C

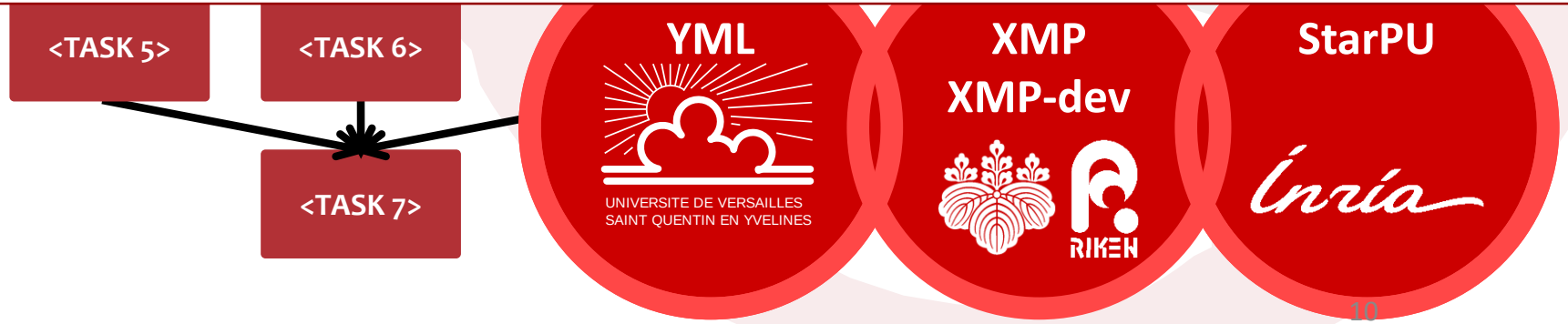


AICS

Background FP2C & XcalableMP (XMP)

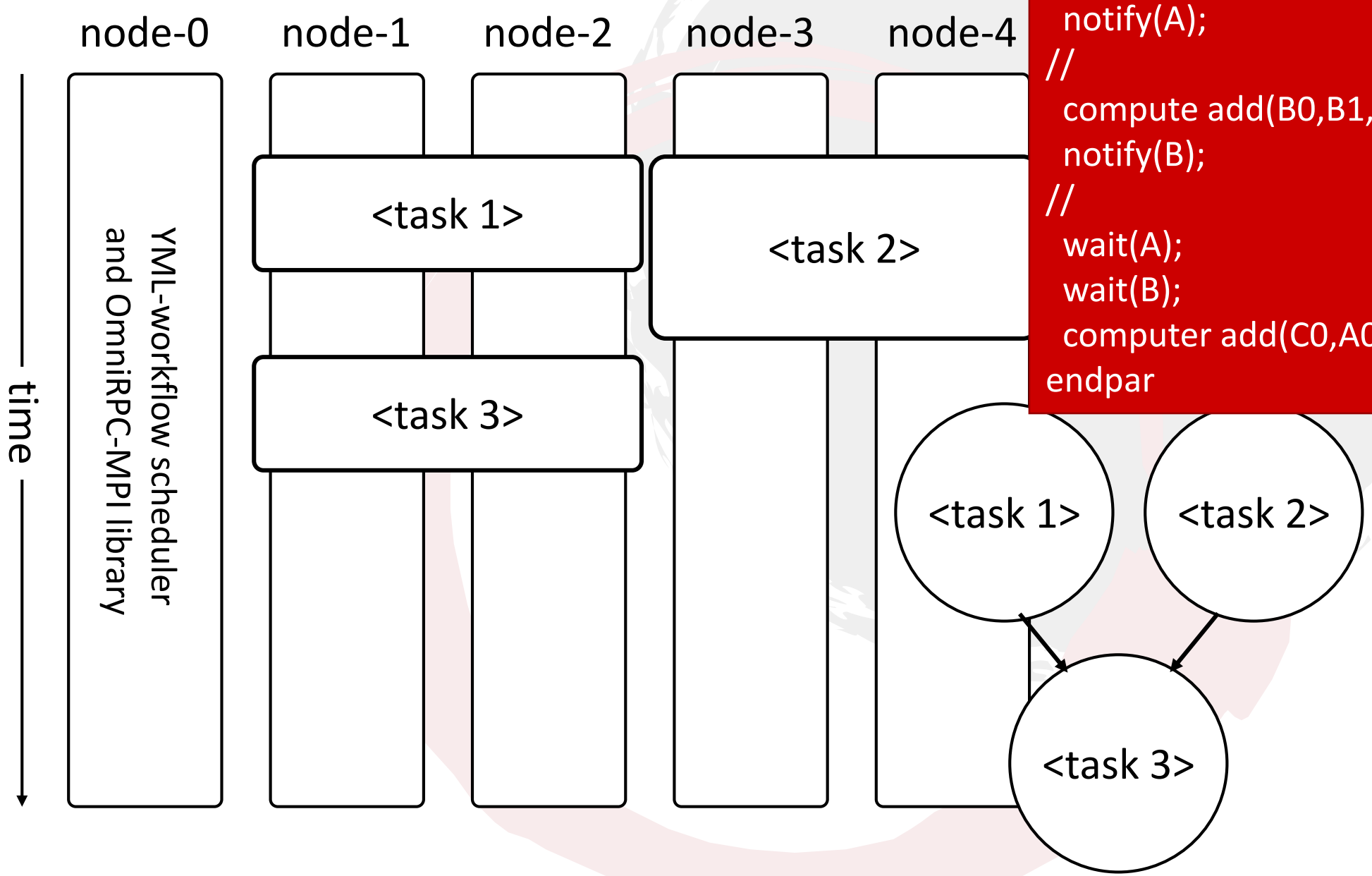
```
<?xml version="1.0"?>
<component type="impl" name="add" abstract="add">
<impl lang="XMP" nodes="CPU:(2,2)">
<templates>
<template name="t" format="block,block" size="8,8"/>
</templates>
<distribute>
<param template="t" name="A(8,8)" align="[i][j]:(j,i)"/>
<param template="t" name="B(8,8)" align="[i][j]:(j,i)"/>
<param template="t" name="C(8,8)" align="[i][j]:(j,i)"/>
</distribute>
<header>
#include<xmp.h>
</header>
```

```
<source>
int myrank,nprocs;
  int i,j,n;
  n=8;
#pragma xmp loop (j,i) on t(j,i)
  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      C[i][j]=A[i][j]+B[i][j];
    }}
</source>
</impl>
</component>
```



AICS

Overview of our proposal




```
par
  computer add(A0, A1, A2);
  notify(A);
//
  compute add(B0,B1,B2);
  notify(B);
//
  wait(A);
  wait(B);
  computer add(C0,A0,B0);
endpar
```

Agenda

Introduction

Background : Multi-SPMD Programming/Execution Environment

FT in multi-SPMD

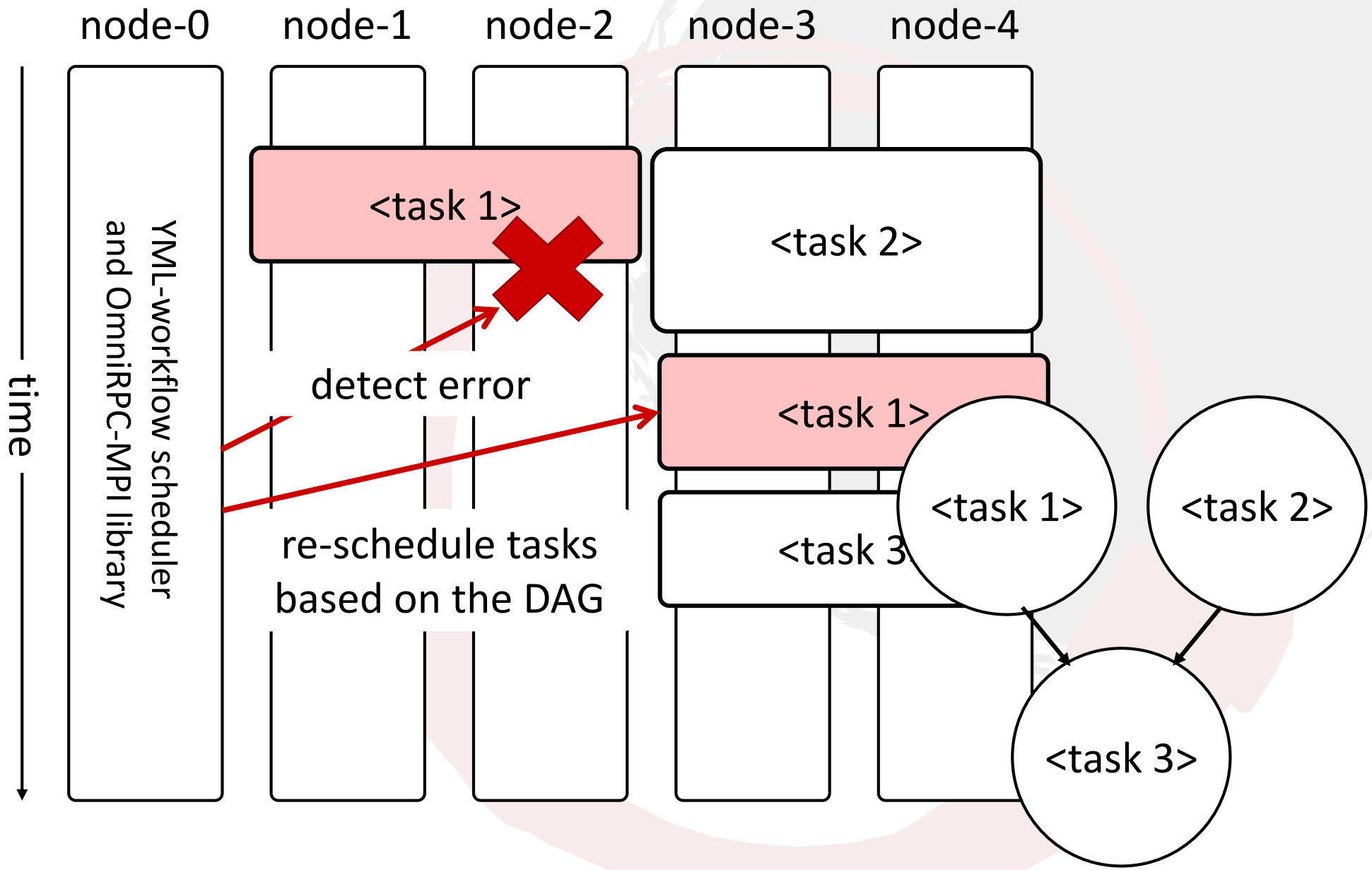
Experiments

Related work

Conclusion

AICS

Overview of our proposal



Implementation

- We have extend
  - Middleware
    - to detect errors
  - Workflow Scheduler
    - to recover errors

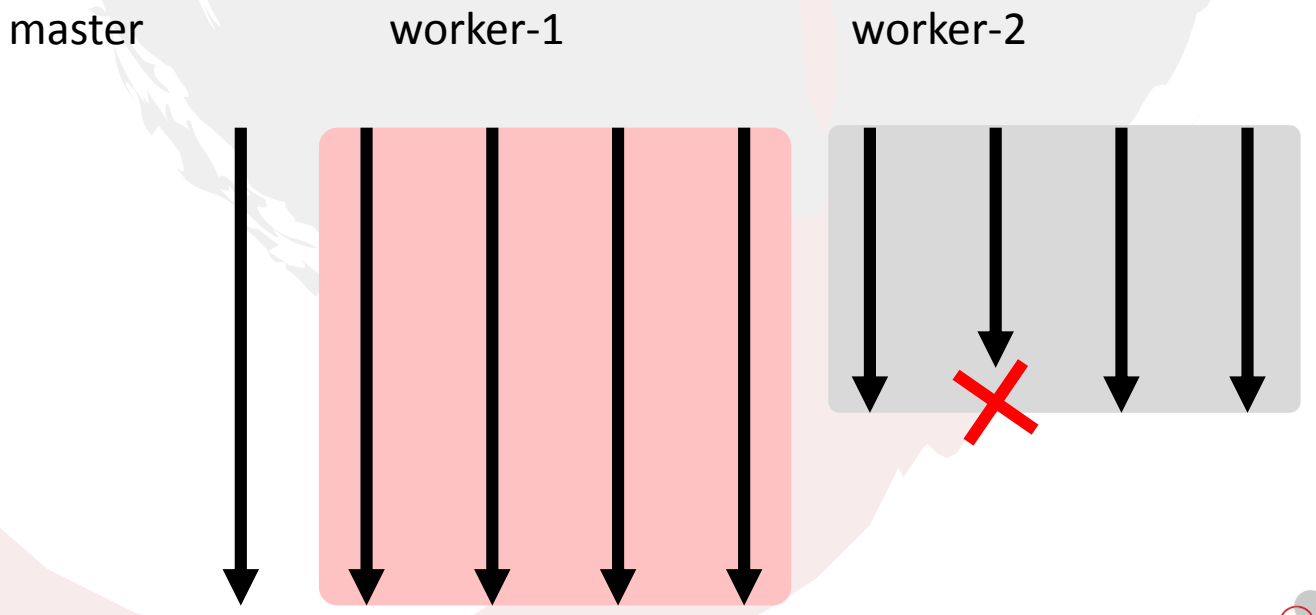
AICS

OmniRPC-MPI to OmniRPC-MPI-FT

- OmniRPC-MPI
  - OmniRPC: Remote Procedure Call library for Grid environment
    - supports a master-worker programming model
    - invoke a worker on a single remote node
- OmniRPC-MPI: Extension for cluster systems (which support MPI)
  - invoke a worker on multiple remote nodes via MPI_Comm_spawn
  - developed as a middleware of FP2C

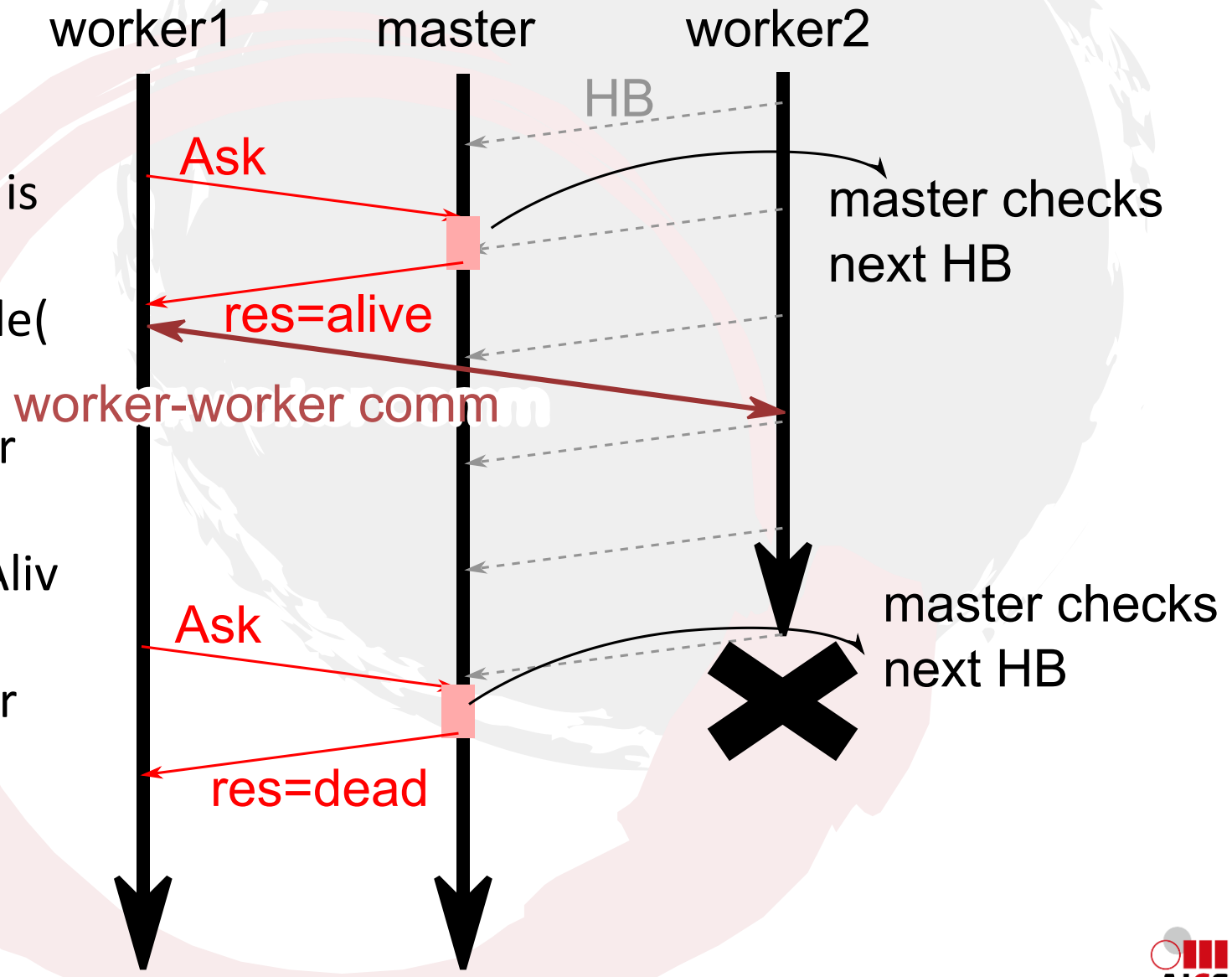
OmniRPC-MPI to OmniRPC-MPI-FT

- OmniRPC-MPI-FT
  - extension of OmniRPC-MPI to realize fault tolerance
- Assumption (a new job scheduler proposed [Mutai et al.2013])
  - there is an error in a node used by a worker program, all the other processes in the worker program are stopped. These processes are not available until the job is finished. On the other hand, the processes in other worker programs and master program can continue.
  - An error in a master is critical



OmniRPC-MPI to OmniRPC-MPI-FT

- Implementation
  - Error detection using Heart Beat (HB) messages
  - API to ask whether a worker is dead or not
    - OmniRpcMpiCheckHandle(void *hd);
      - master checks worker availability
    - OmniRpcMpiAskHandleAlive(int id);
      - worker checks worker availability



Workflow Scheduler

- YML workflow scheduler
  - sends requests to execute tasks to the middleware (OmniRPC-MPI library) based on the DAG of a workflow application
- YML workflow scheduler for FT
  - if there is an error in req[i] (reported by OmniRPC-MPI-FT), then remove it from the request-list and return main loop
  - The main loop executes the req again.

```
Yml::Core::SchedulerTask
*MpiBackend::retrieveImpl(void){
 for(i=0;i<NUMBER_REQUESTS;i++){
  if(OmniRpcProbe(req[i])==success){
   remove the req[i] from the request list
   return task[i];
  }else if(OmniRpcProbe(req[i])==fail){
   remove the req[i] from the request list
   set the status of task[i] error
   return task[i];
  }else{
   // req[i] is in execution
   // retrieveImpl do nothing
  }
 }
 return 0;
}
```

AICS

Agenda

Introduction

Background : Multi-SPMD Programming/Execution Environment

FT in multi-SPMD

Experiments

Related work

Conclusion

AICS

Experiments --Environment

- The overhead of the fault detection
- The ability to find a failure an to recover from the failure
- The elapsed time when error(s) occur.

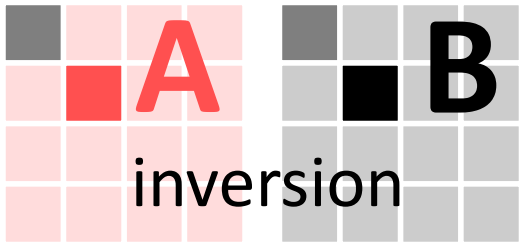
- 65 nodes
  - 1 node for YML workflow scheduler
  - 64 nodes (1024 processes) for worker-programs (tasks)

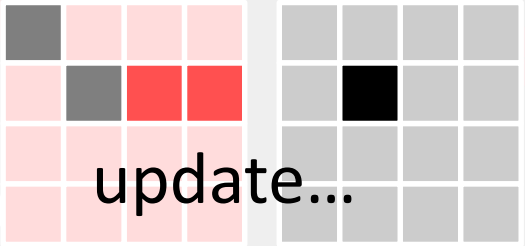
FX10 @ AICS	
CPU	FUJITSU SPARC64IXfx 16core 1.65 GHz
Memory	32GB/s, 85GB/s
Compiler	Fujitsu Compiler 1.2.1

AICS

Experiments -- Test Problem (Block-Gauss-Jordan)

- Compute an inversion of a matrix by inversions of a block of the matrix and the updates of other blocks based on the inversions.





- We can control the hierarchical parallelism levels easily by FP2C
  - Fix the matrix size (20480) total number of processes(1024)
  - Change the size of blocks and the number of processes for each task (block)

	64	256	512	1024
20480^2 (1^2 blocks)	-	-	-	
10240^2 (2^2 blocks)	-			
5120^2 (4^2 blocks)				
2560^2 (8^2 blocks)				

≒ Simple SPMD

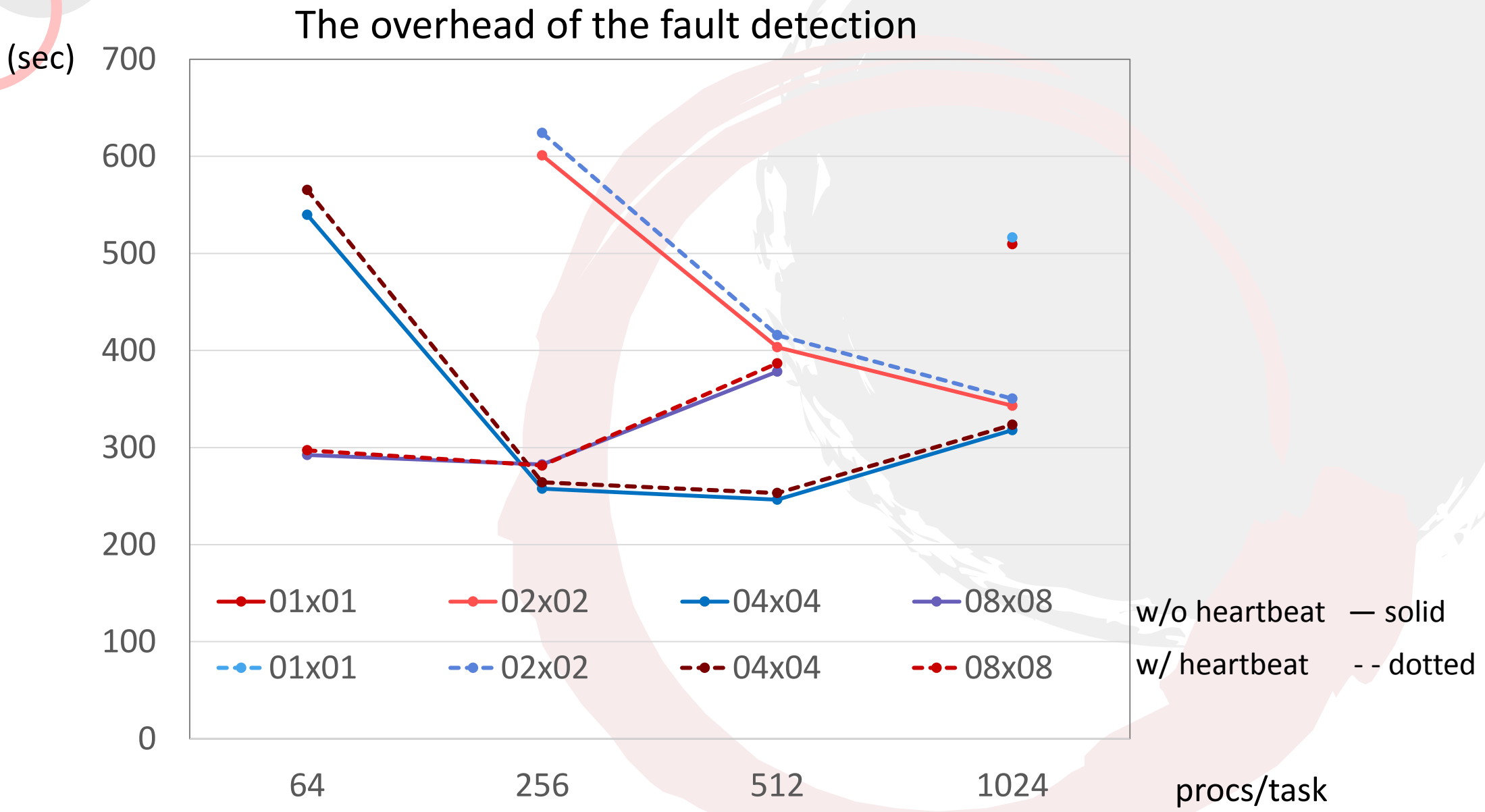
2 tasks can be executed simultaneously

≒ Simple workflow

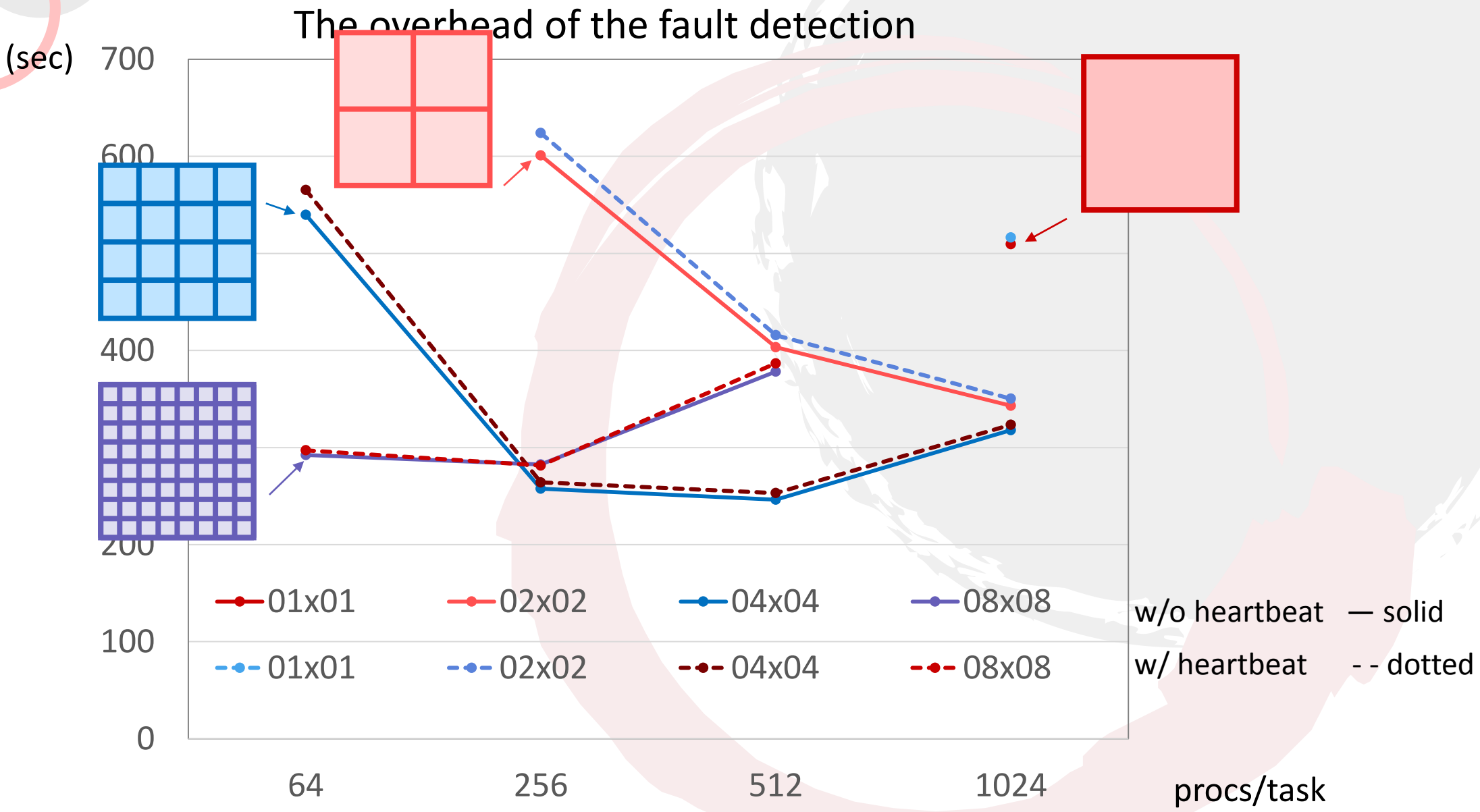
4 tasks can be executed simultaneously

AICS

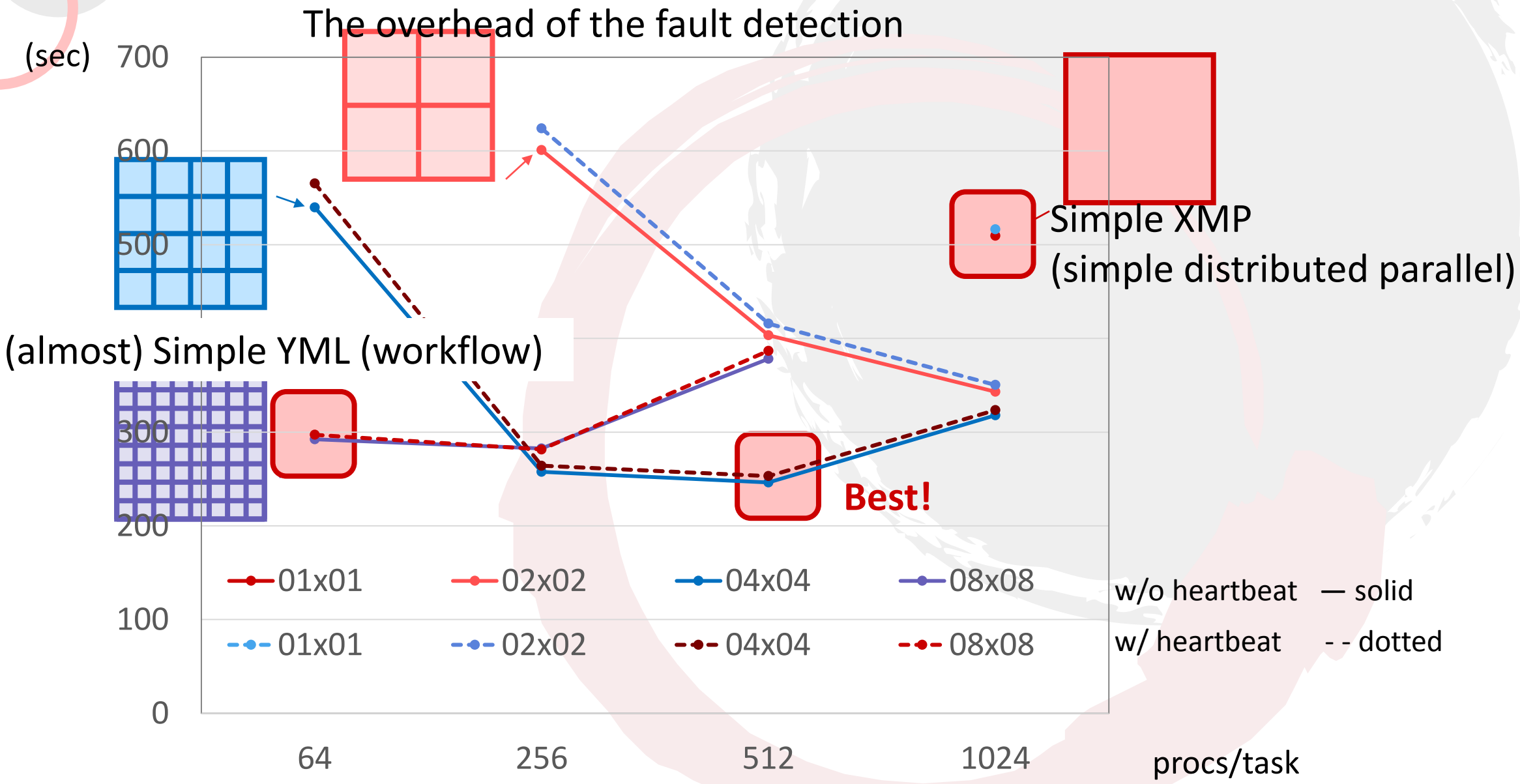
Experiments -- Results (w/o Error)



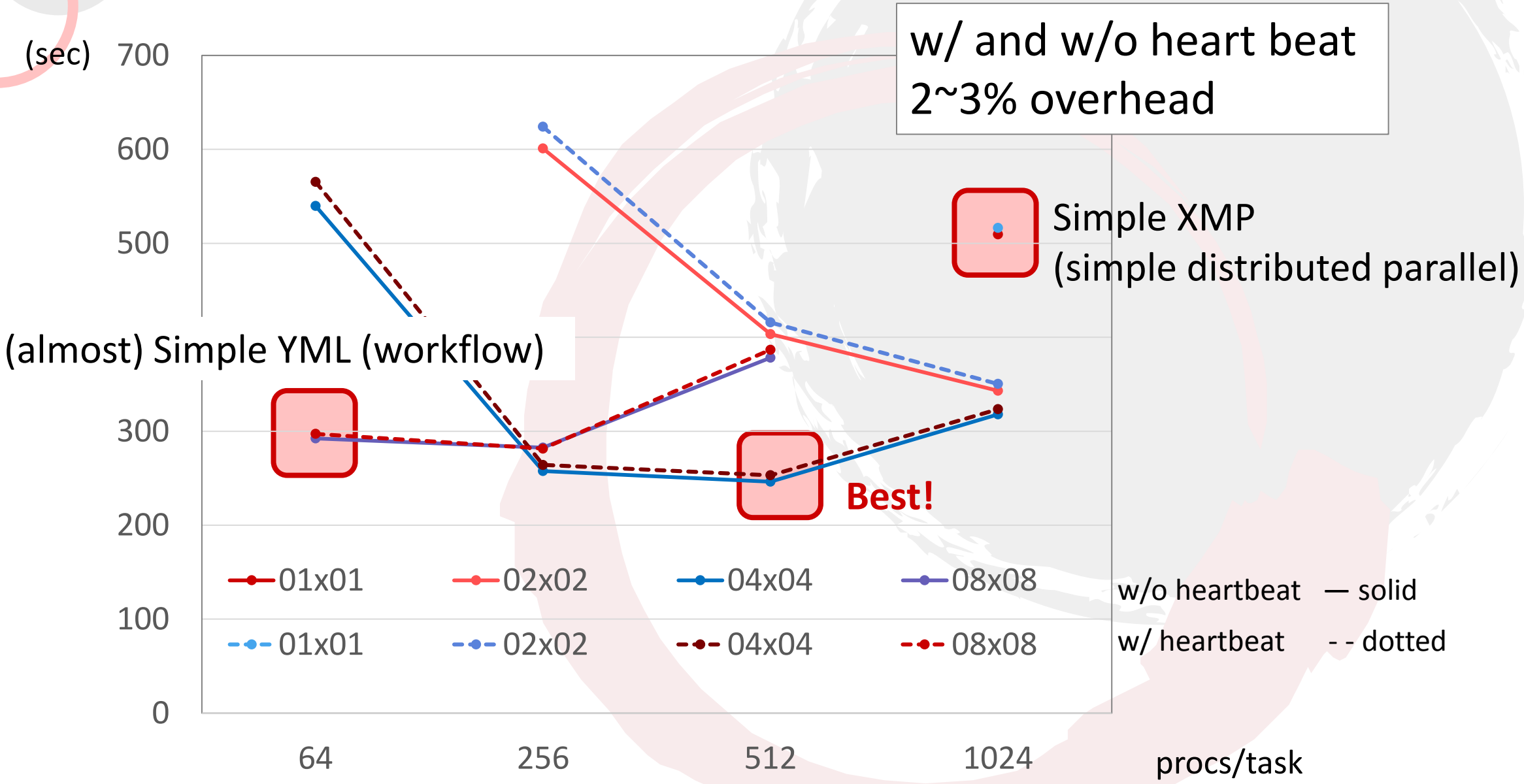
Experiments -- Results (w/o Error)



Experiments -- Results (w/o Error)



Experiments -- Results (w/o Error)



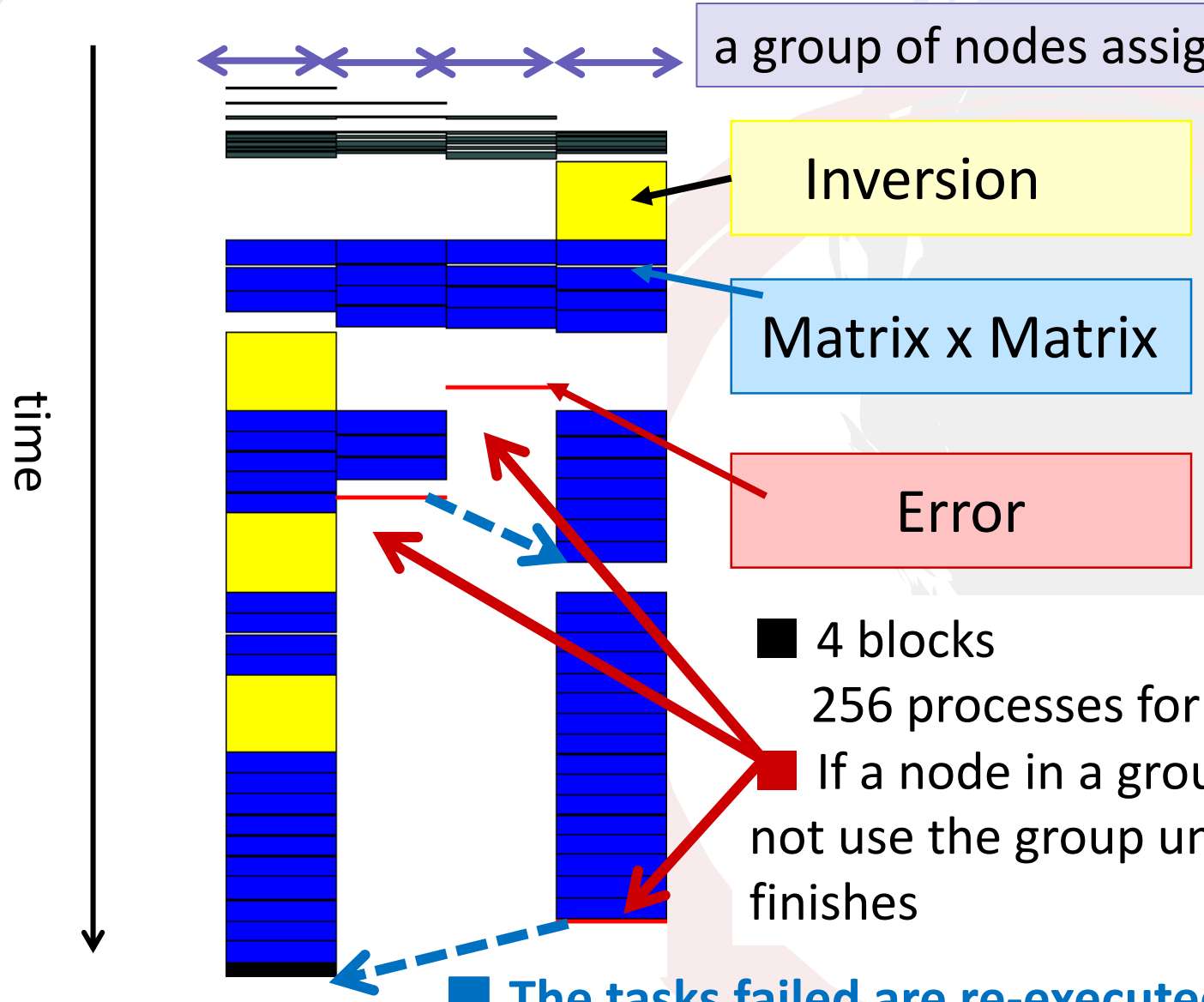
Experiments -- Error Scenarios

- The ability to find a failure an to recover from the failure

- Fault : process can not continue for some reason
  - Sleep a process in worker programs randomly based on expected MTBFs
    - 12.5, 25, 50 hours
- 10 times for each of (MTBF, procs/task, # of blocks) combinations

AICS

Experience - Timeline (observed in an experiment)



a group of nodes assigned for each task

Inversion

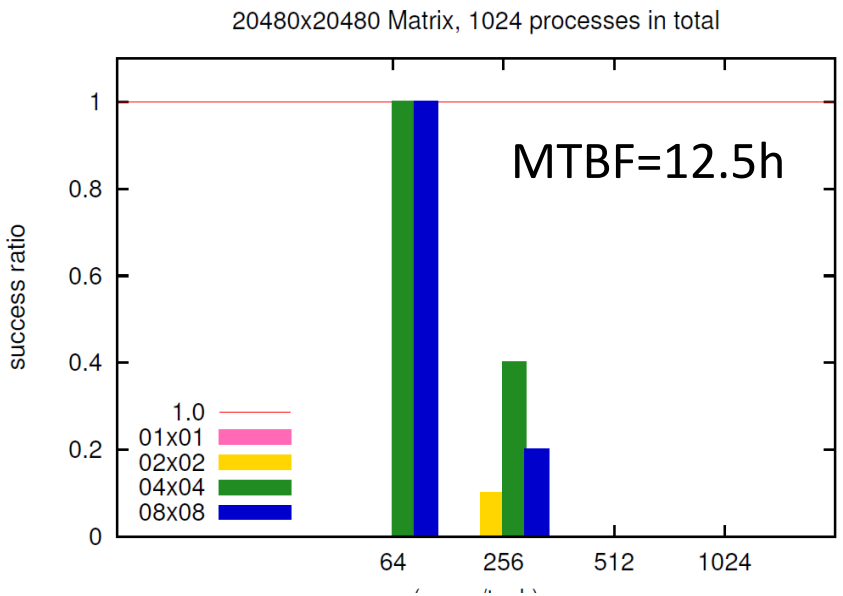
Matrix x Matrix

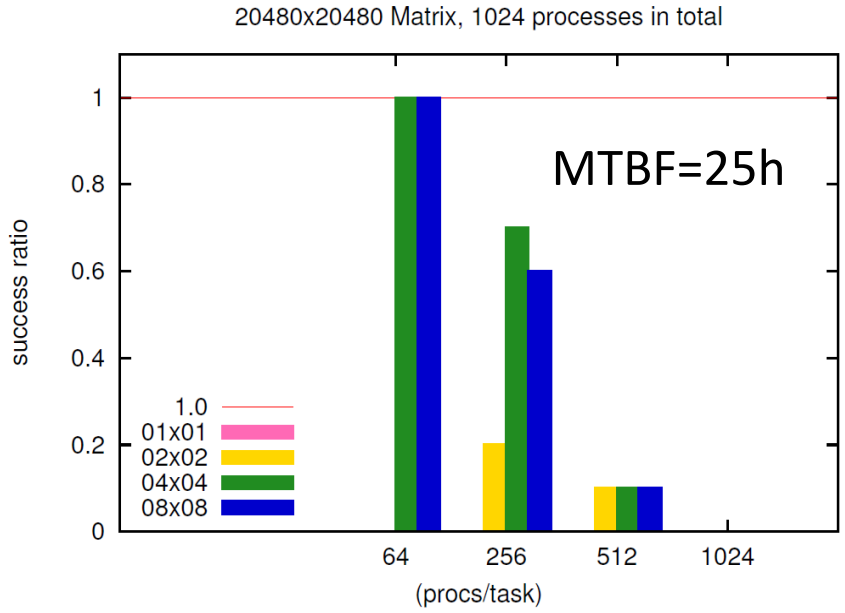
Error

- 4 blocks
  256 processes for each task
- If a node in a group fails, we can not use the group until the job finishes

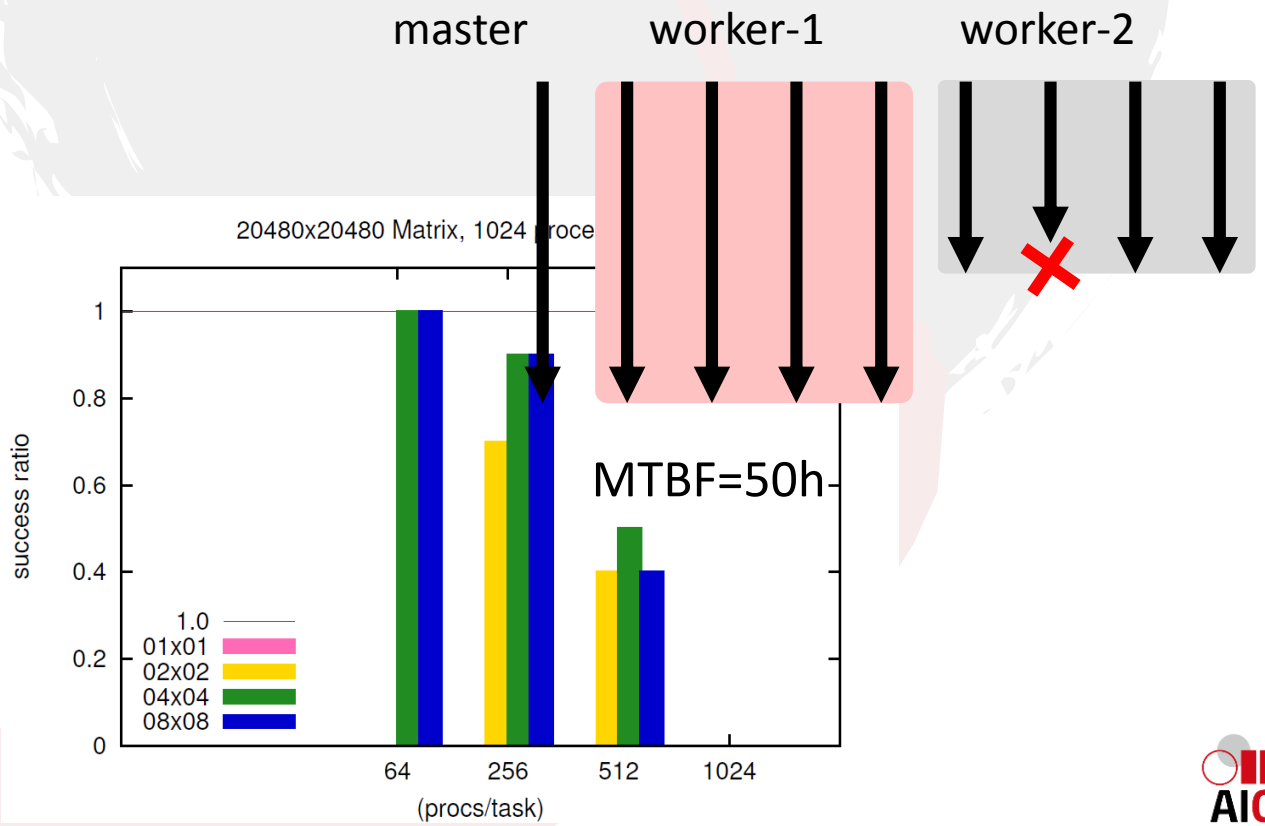
- **The tasks failed are re-executed on another group**

Experience -- Completion ratio for each MTBFs



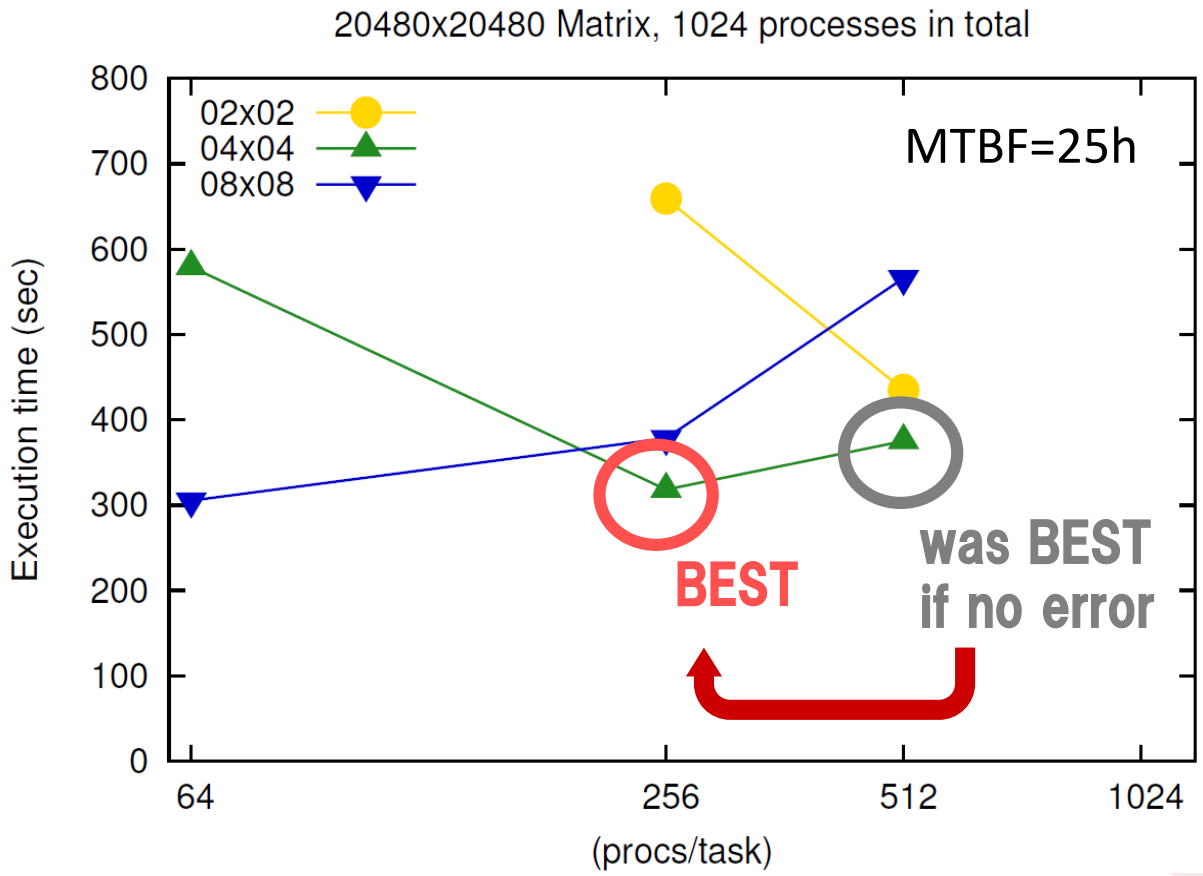


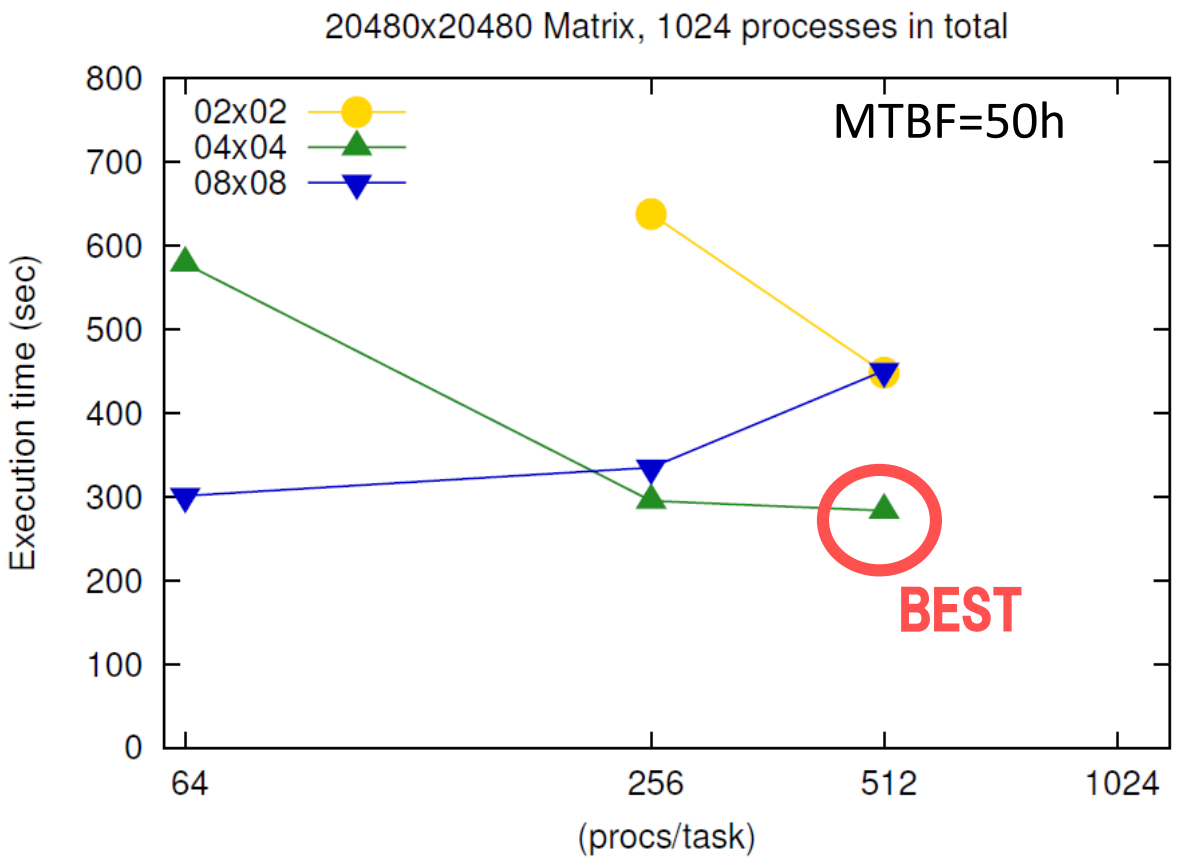
- 1x1 block, 1024 procs/task (simple XMP programming model) always fails when there is an error
- many small blocks and small # of procs/task are good



Experience -- Execution time ratio w/ error

- Execution time when there is **at least one error**
  - ignore the "lucky" case that an application is completed without any error
- Execution time increases
  - 12% average, **3% min**, 19% max (without any modification in application





Experience -- Summary

- The overhead to detect error (HB messages) is only 2~3%
- The overhead to detect an error(s) and complete application (even where there is an error(s) varies from 3-19%.
  - We can reduce it by controlling appropriate decomposition of computational resources for the multi SPMD programming model
  - The control is easy(!), if you use our programming tool
- We've find that the best combination of SPMD and workflow depends on MTBF
  - Again, we can control it easily by using our "multi-SPMD" programming model

AICS

Related Works

- Xcrypt [Hiraishi, *http://super.para.media.kyoto-u.ac.jp/xcrypt/*]: is a master-worker programming tool based on the management of batch-jobs. Failed jobs can be executed by submitting it the batch job scheduler again
- Falanx [*https://sites.google.com/site/spfalanx*], which is a programming middleware, uses User Level Fault Mitigation and provides APIs and a runtime library to support the development and execution of hierarchical programs.
- GVR (Global View Resilience): is a programming approach to exploits a global view data model and provides APIs to manage global view structures and handle errors
- CDs (Constraint Domains): takes a hierarchical and domain knowledge based approaches and provides APIs

- FP2C users does not need to add new APIs (APIs in OmniRPC-MPI-FT are used in YML-workflow scheduler not, in your application source codes)
- FP2C does not essentially require job-management programs
- FP2C supports PGAS to program tasks

AICS

Conclusion

- We have developed a fault tolerance mechanism in a multi-SPMD programming / execution environment called FP2C
- A fault tolerance and resilience mechanism without any modification of the application's source code
- We have developed
  - middleware OmniRPC-MPI-FT to detect errors in remote programs
  - workflow scheduler to realize fault resilience
- The overhead for fault detection is 2~3 % on average
- The execution time of each application increases up to 19 % under errors
  - But, we can reduce it to a few % by controlling appropriate decomposition of computational resources for the multi SPMD programming model

AICS