

Design of Data Management for Multi SPMD Workflow Programming Model

T. Dufaud*, **M. Tsuji****, M. Sato**

*University of Versailles, France

*RIKEN Center for Computational Science, Japan



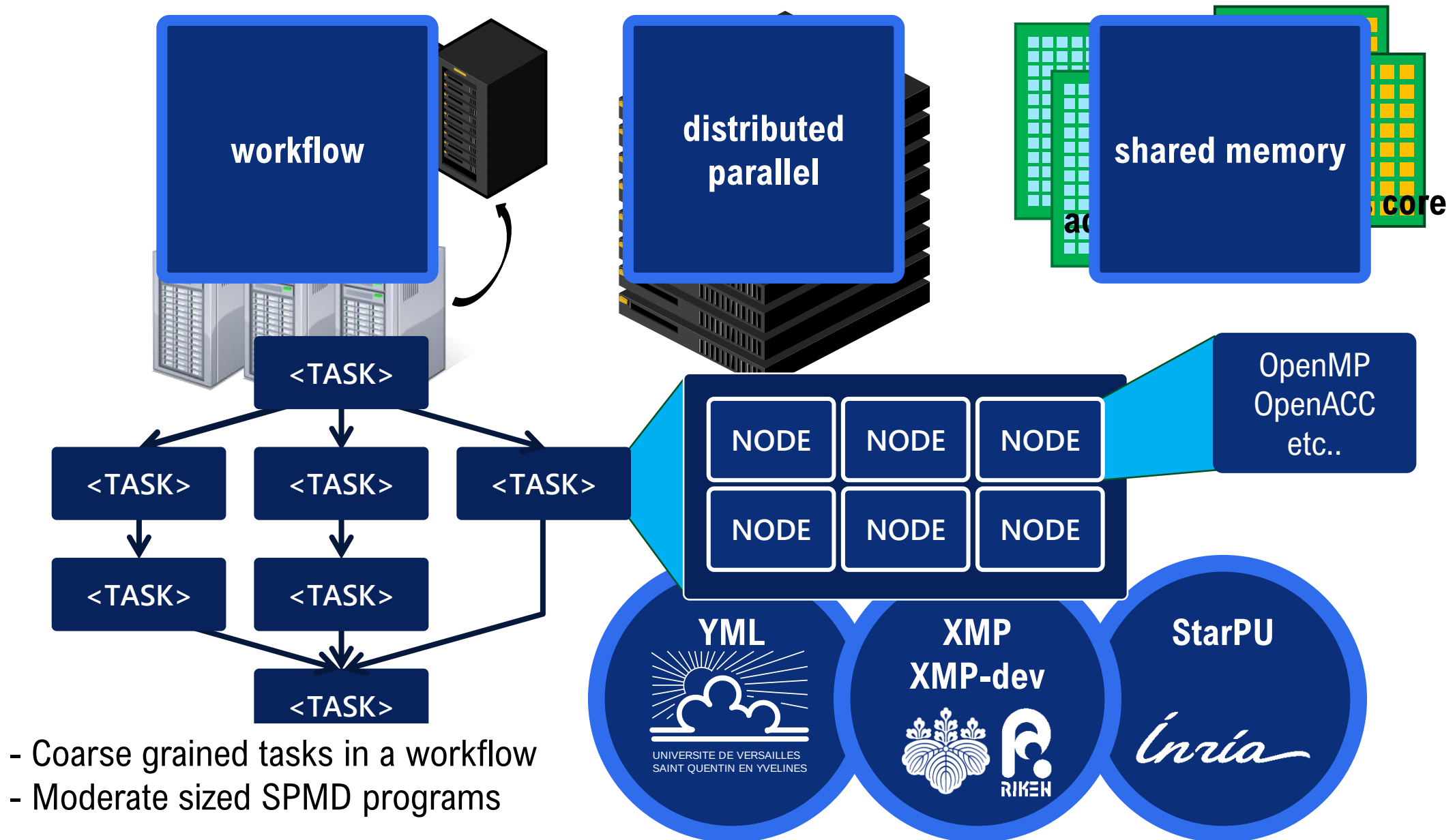
Agenda

- Motivation and background
- Data exchange issues and model
- Implementation through YAML user interface
- Development of Block Gauss-Jordan Algorithm

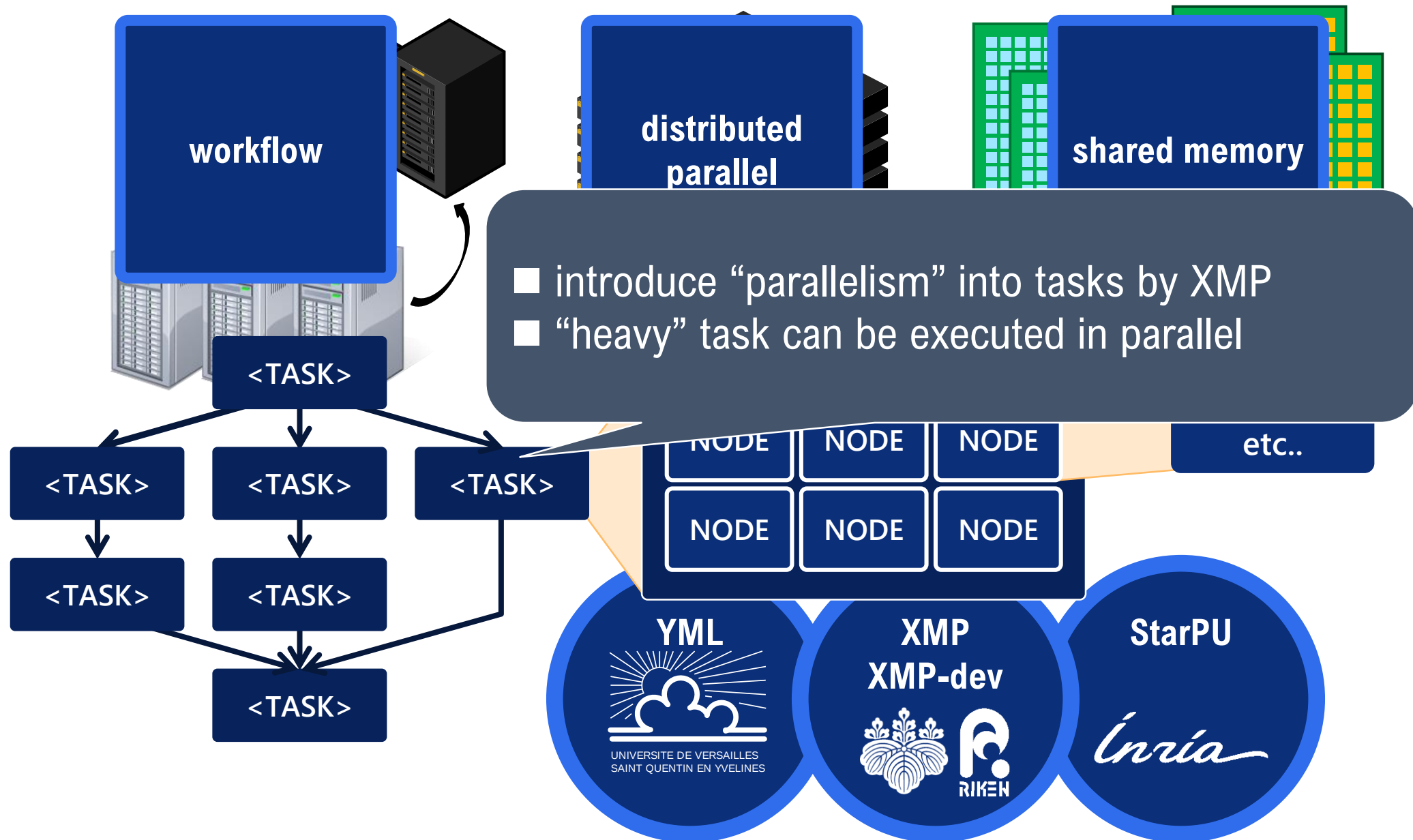
Multi SPMD (mSPMD) for large scale computing

- Why Multi SPMD
 - Flat MPI reaches a limit
 - Simulation code may consists in several parallel programs
 - Each SPMD is an independent program $\Rightarrow$ component approach, reusability
- Programming environment
 - Dominant execution model approach: **X+MPI**, MPI + X
 - Coupling between models and apps involves I/O
 - **Proposed approach for 2-level programming model**
 - Workflow programing with task dependencies
 - Single Program Multiple Data at the fine layer
 - Separation of data and computation (N.Emad, O. Delannoy, M. Dandouna AICCSA 2010)
 - **Programing environment mSPMD**
 - **Distributed parallel** XMP is a PGAS language <http://www.xcalablemp.org>
 - **Workflow** YML: graph of task, each task is described by a component, <http://www.yml.prism.uvsq.fr>

Multi SPMD (mSPMD) Programming Model

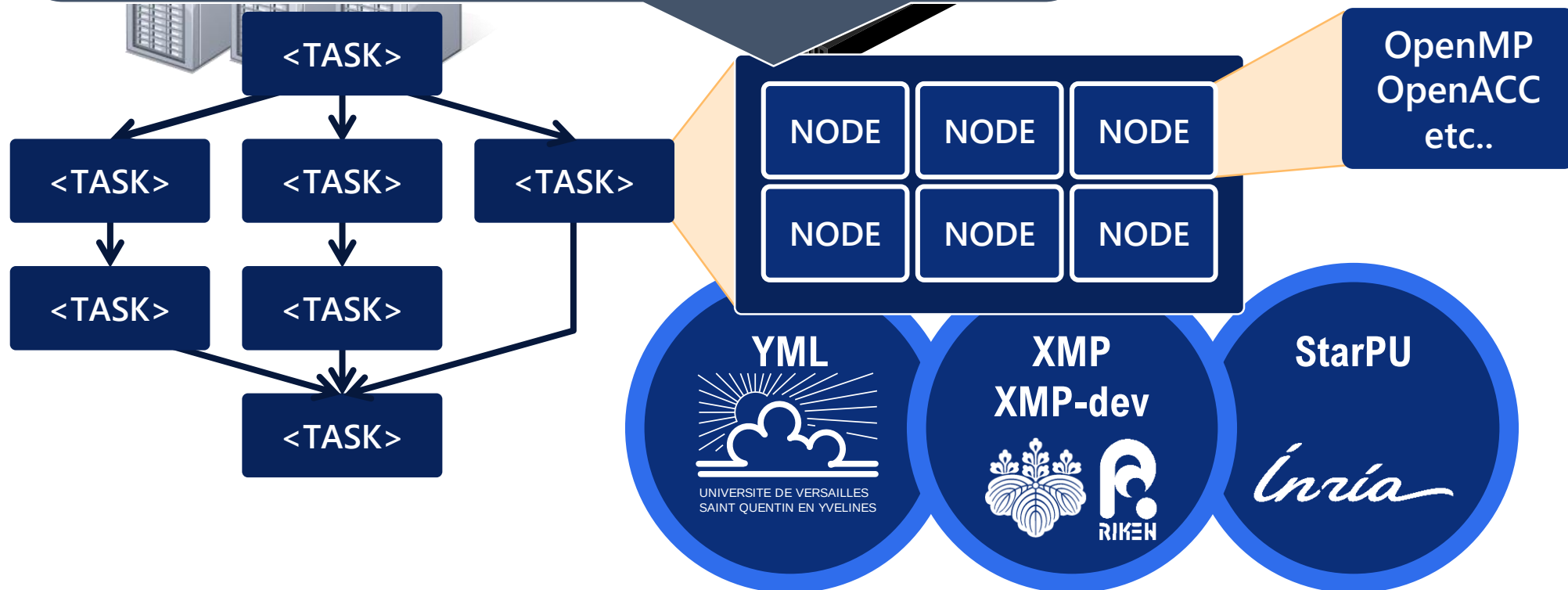


Multi SPMD (mSPMD) Programming Model



Multi SPMD (mSPMD) Programming Model

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



Multi SPMD (mSPMD) Programming Model

- Compose complex application by combining parallel applications and libraries

shared memory

core

<TASK>

<TASK>

<TASK>

<TASK>

<TASK>

<TASK>

<TASK>

NODE

NODE

NODE

NODE

NODE

NODE

OpenMP
OpenACC
etc..

YML



UNIVERSITE DE VERSAILLES
SAINT QUENTIN EN YVELINES

XMP

XMP-dev

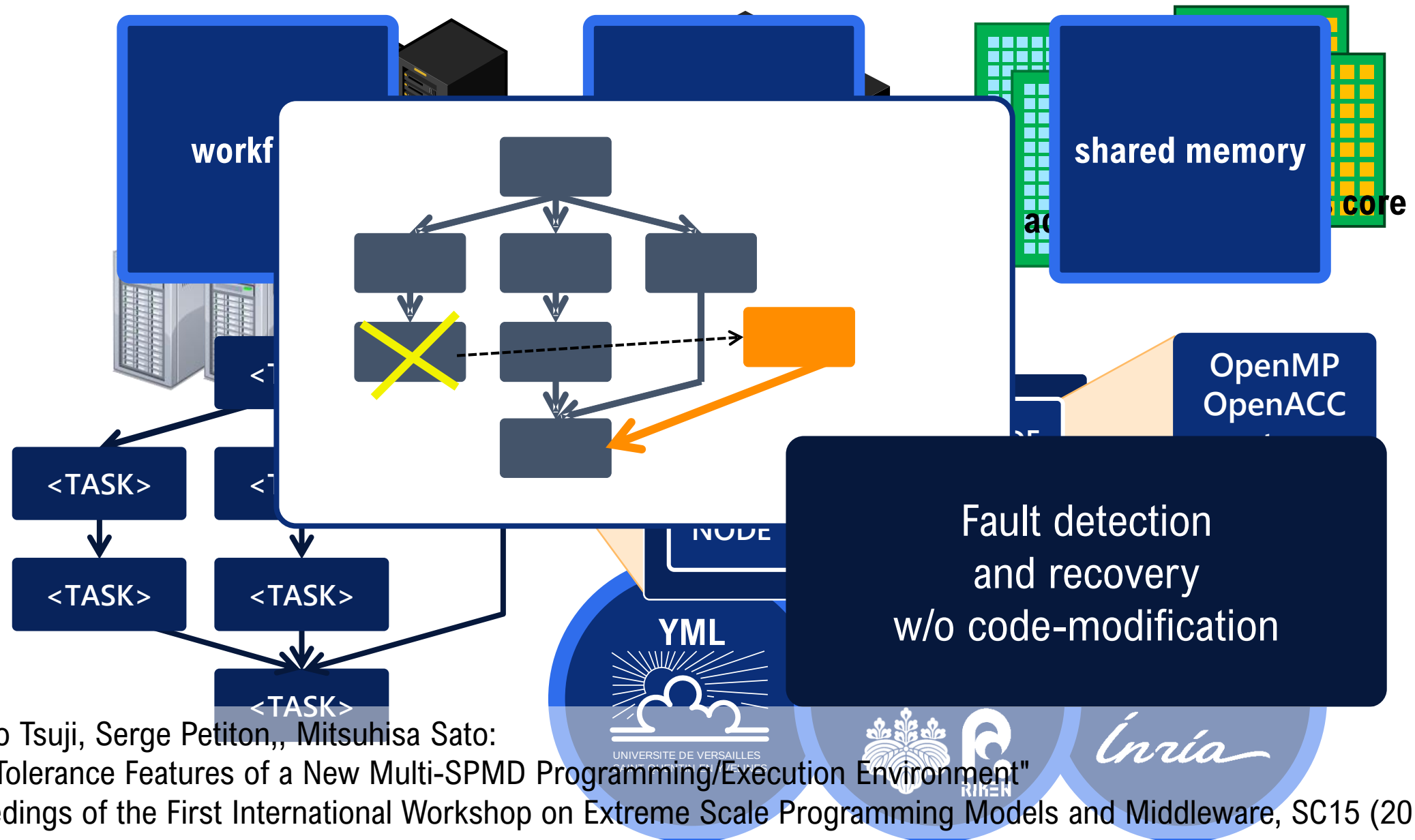


RIKEN

StarPU

Inria

Multi SPMD (mSPMD) Programming Model

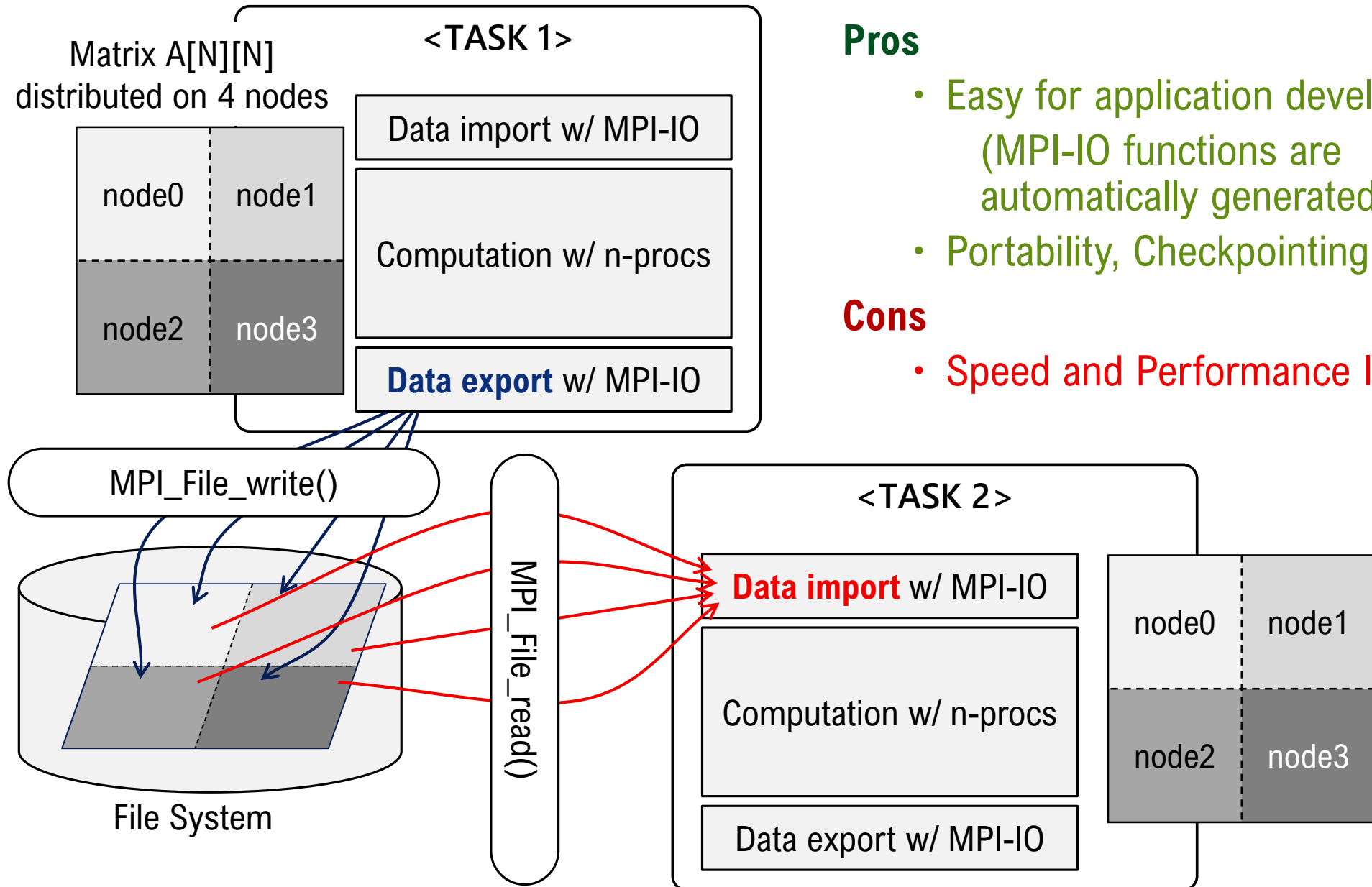


Miwako Tsuji, Serge Petiton,, Mitsuhsa Sato:

"Fault Tolerance Features of a New Multi-SPMD Programming/Execution Environment"

Proceedings of the First International Workshop on Extreme Scale Programming Models and Middleware, SC15 (2015)

I/O based Data Exchange in mSPMD



Pros

- Easy for application developers (MPI-IO functions are automatically generated)
- Portability, Checkpointing

Cons

- Speed and Performance Instability

Agenda

- Motivation and background
- **Data exchange issues and model**
- Implementation through YAML user interface
- Development of Block Gauss-Jordan Algorithm

Data exchange in mSPMD

- **Context and choices**

- Parallel programming model on each level
 - **Graph of tasks** on the higher level
 - **SPMD** at the fine level
- Data definition
 - Component approach at the higher level (large parallel tasks)
 - Definition of data separated from computation
 - **High level $\Rightarrow$ Global data (To be Imported/Exported)**
 - **Low level $\Rightarrow$ Group of local data**
 - Import/Export are synchronous
 - Import from / export to a data repository
- Data placement and dependencies
 - data flow deduced from workflow
- Data persistency
 - Data repository can enable persistent storage

- **Quality**

- High performance in case of intense computation
- Fault tolerance, persistency of important data

Key point of the system

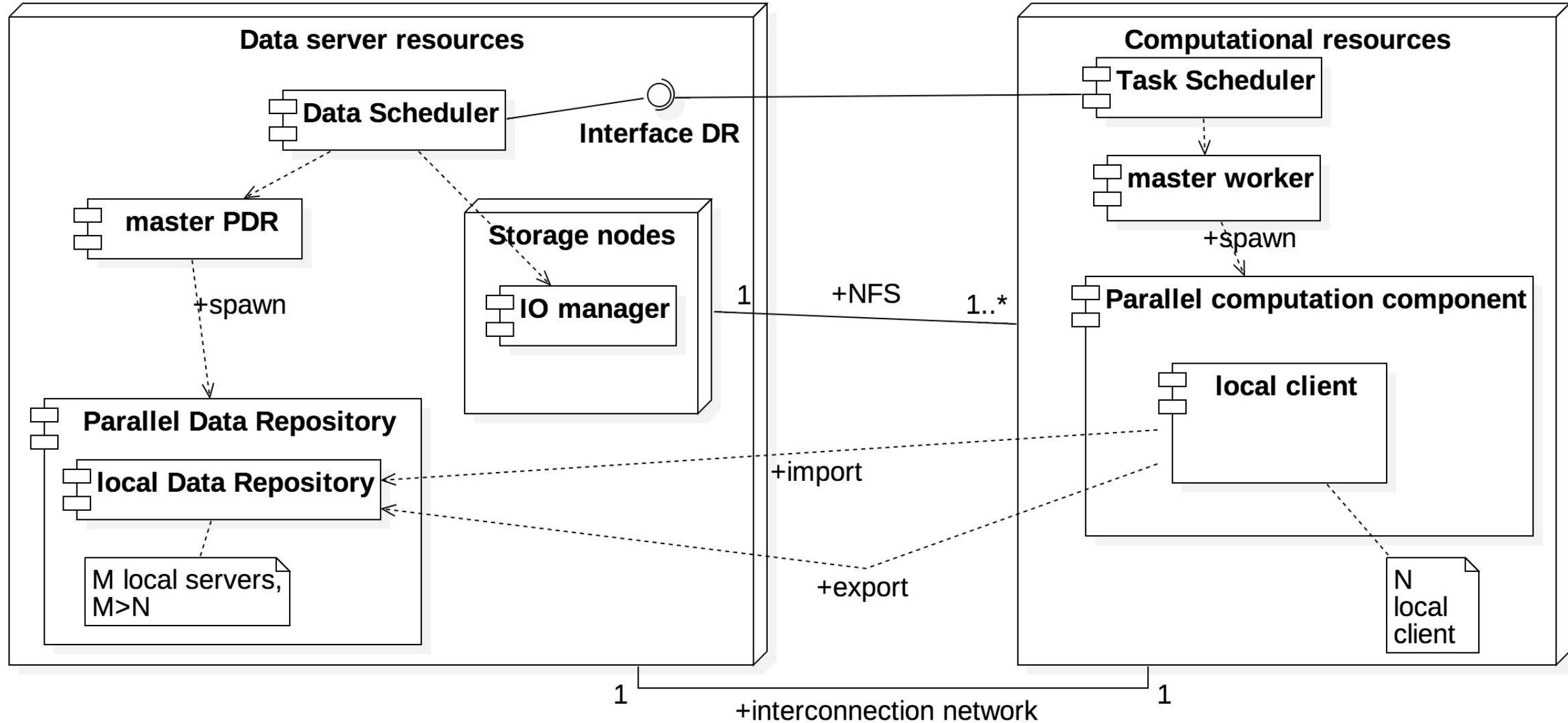
- **Main design**

- Detection of data distribution (Given by execution model)
- Performance of communication according to data distribution...
- ... and **architecture and middleware**
- adapt strategy at runtime(considering size of data and distribution)
- earn benefit of both dataflow and workflow knowledge (Drawing our inspiration from M. Hugues et. al. ASIODS 2011)
 - consider a task scheduler and a data scheduler
 - interaction between schedulers for optimization
 - ⇒ anticipate data migration, remapping of parallel data

- **Remarks**

- Component approach + ability to get information from interfaces
 - ⇒ enable integration of automatic decision tools

Target system



Agenda

- Motivation and background
- Data exchange issues and model
- **Implementation through YML user interface**
- Development of Block Gauss-Jordan Algorithm

First step implementation enable by YML front end

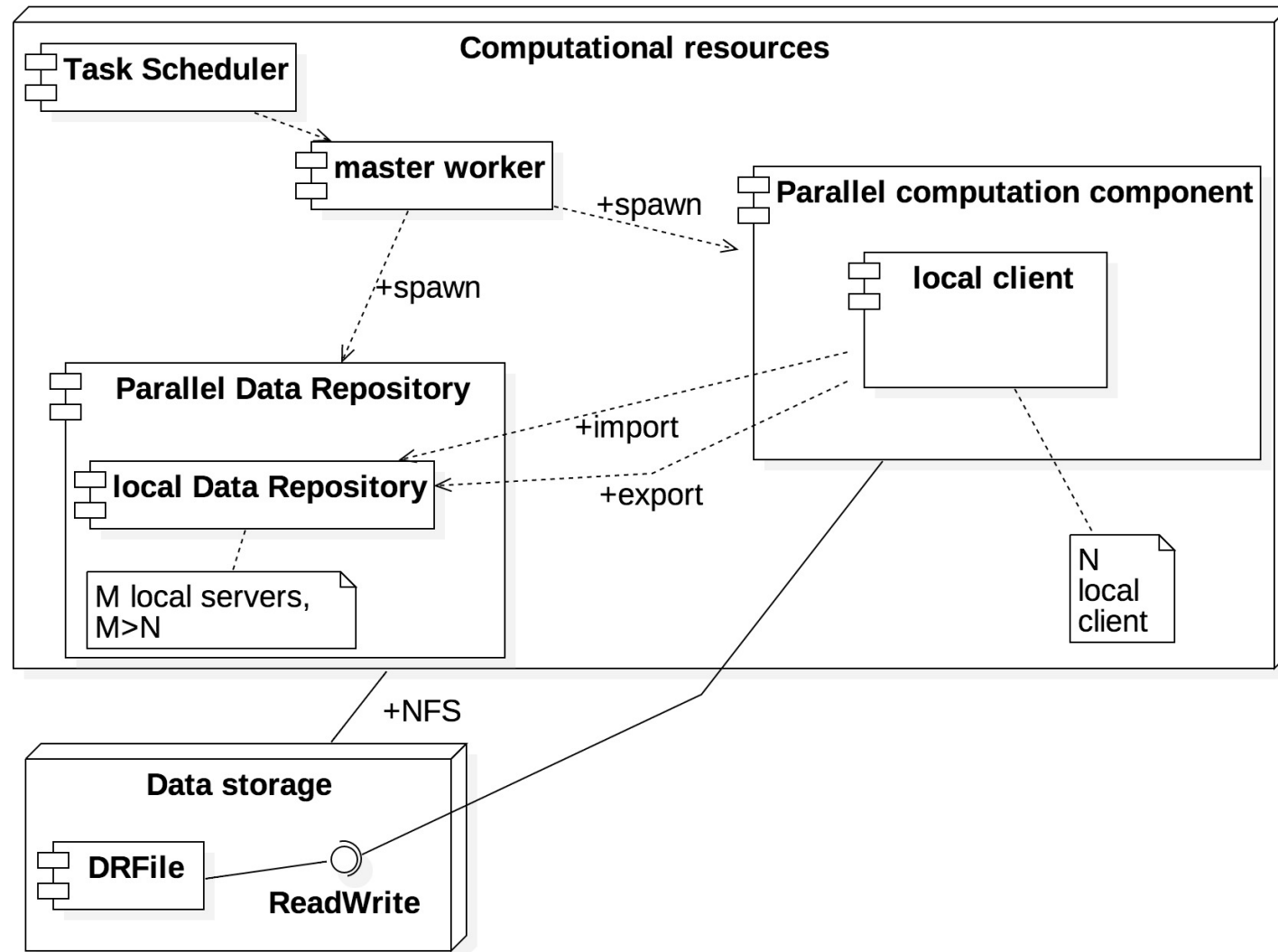
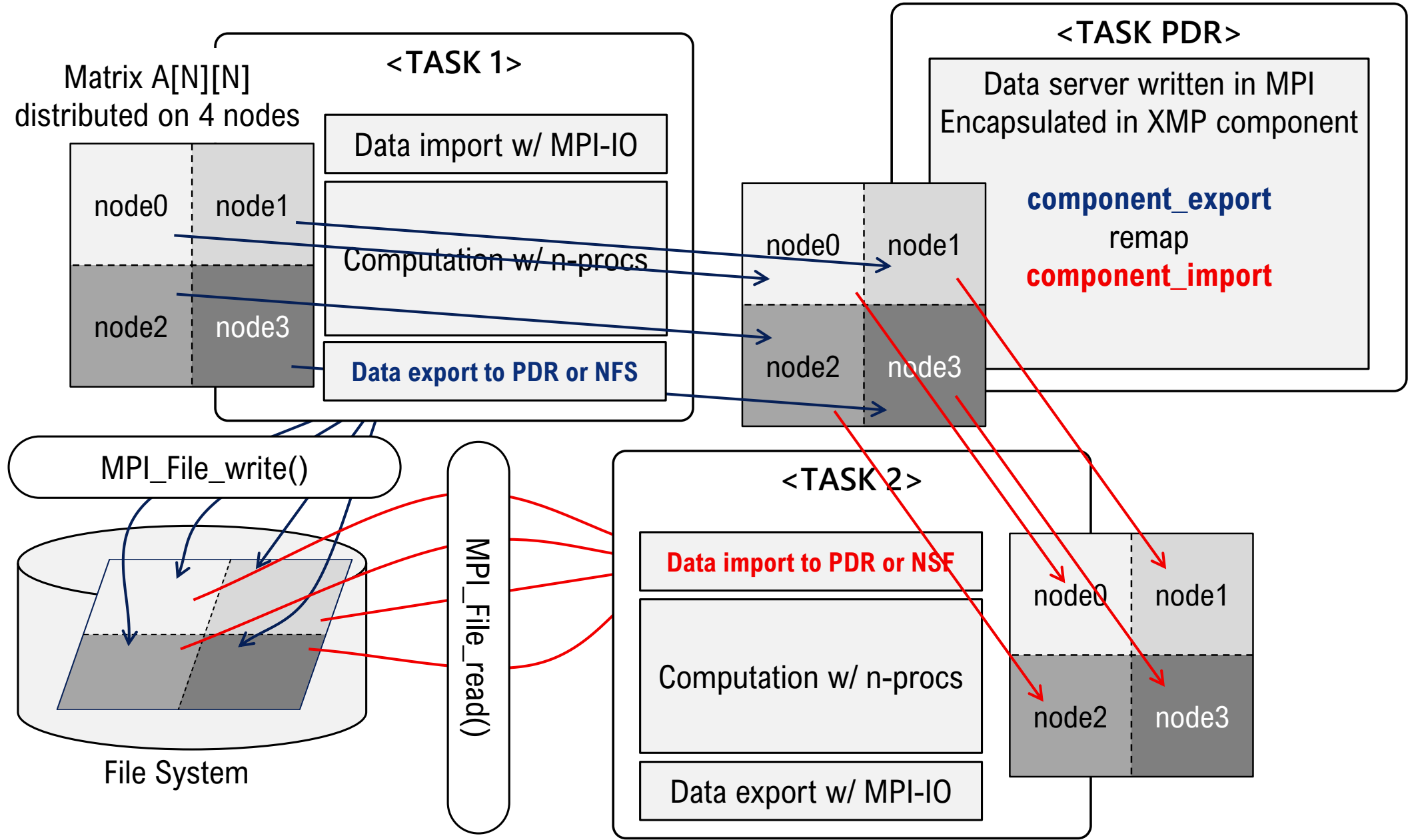


Figure: Integrate new data management through YML front end

Import/export with 2 strategies in YML-XMP



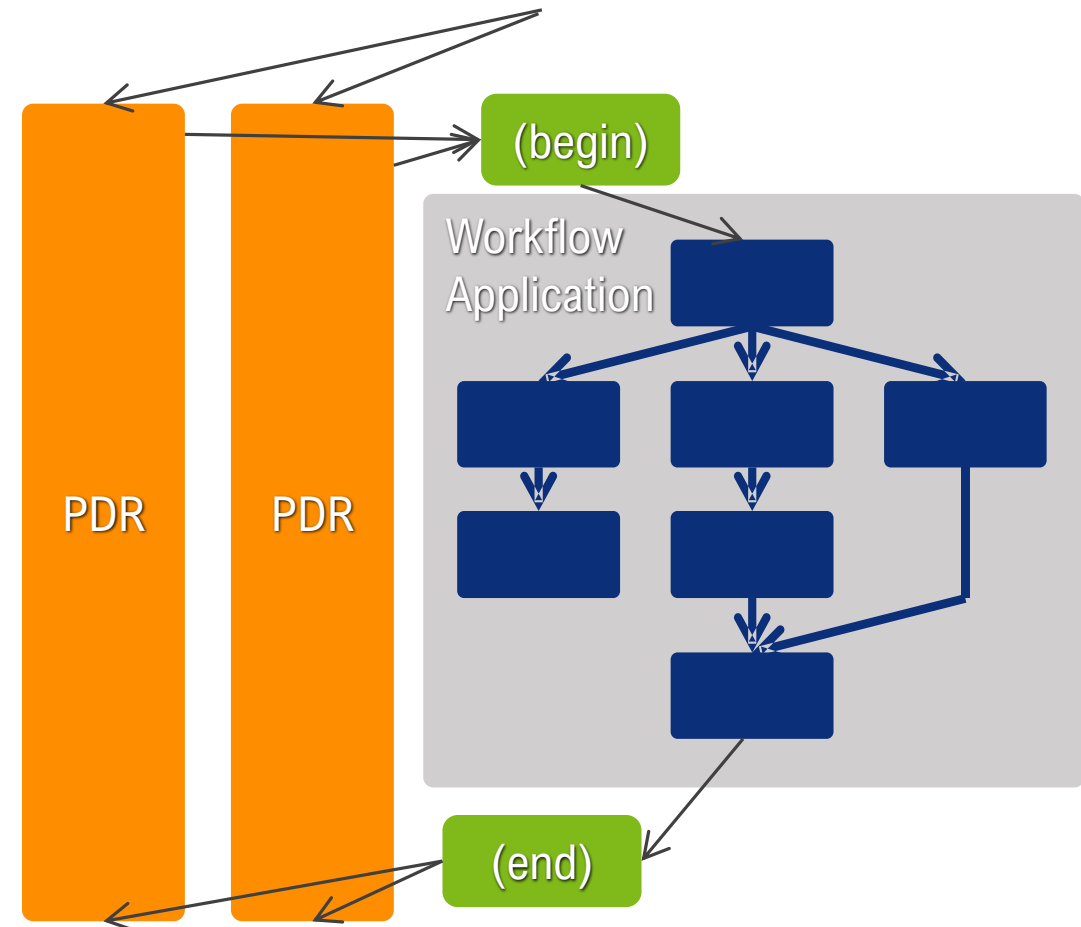
Current Implementation of Parallel Data Repositories

```
par
  # Start PDRs
  par(i:=0;nPDRs-1)
  do
    compute startPdr(i);
    notify(begin[i]);
  enddo
//

  # Workflow Application
  ...
  notify(end);
//

  # Stop PDRs
  wait(end);
  par(i:=0;nPDRs-1)
  do
    compute stopPDR(i);
  enddo
endpar
```

The PDRs are implemented as tasks in a workflow



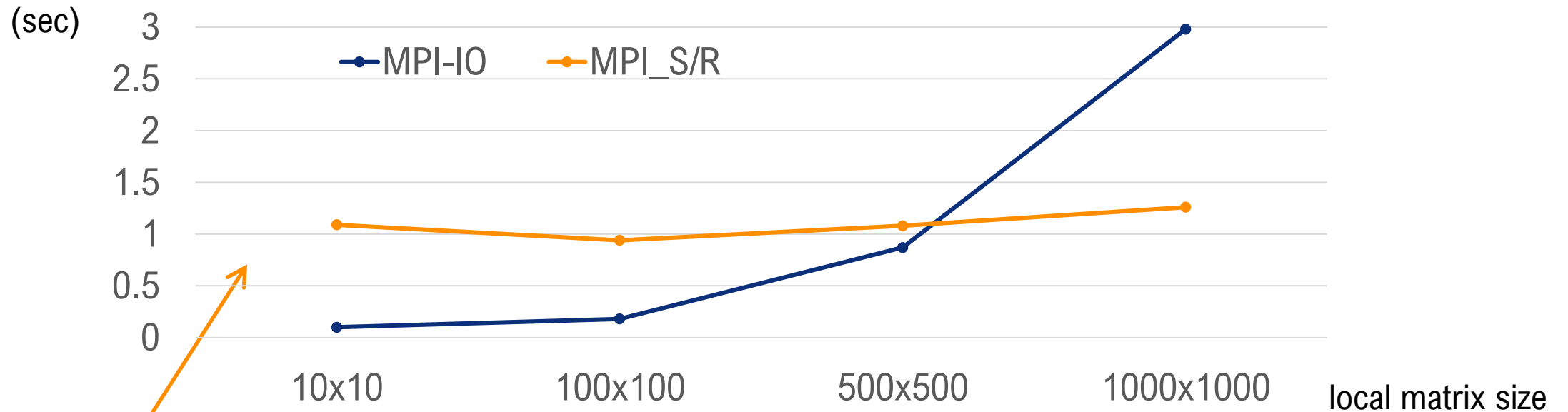
Preliminary Experiment:

Performance of our model implementation

- Key points of evaluation
 1. Import export by one task increasing load
 2. Weak scaling in case of one task
 3. Multiple accesses (N tasks, one PDR)
- Platform
 - 92 nodes IBM cluster (iDataPlex dx360 M4 servers)
 - 2 CPU Sandy Bridge E5-2670 (2.60GHz)
 - 8 cores per CPU / 16 cores per node
 - 32 GB RAM per node

Experiment (1) Import/export by one task, increasing load

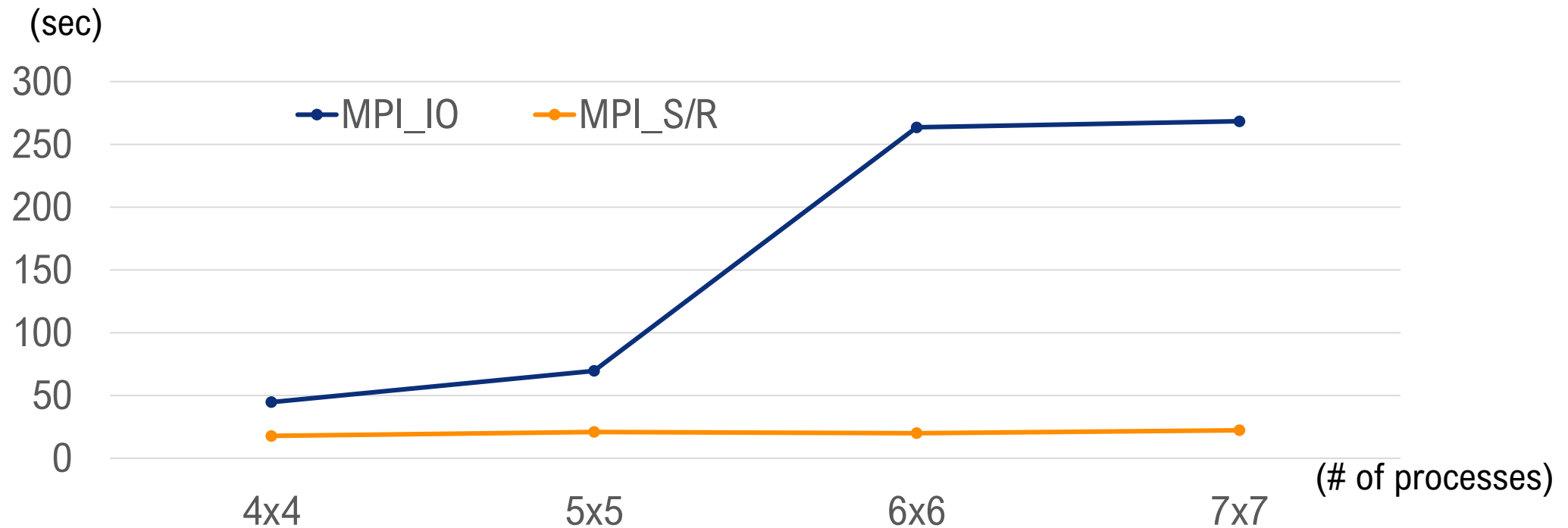
- Performance for Import/Export a real matrix
 - 1 client: $n \times n$ Matrix, 16 procs
 - 1 pdr: 16 procs
 - MPI-IO: Current mSPMD implementation using MPI-IO
 - MPI_Send/Recv: No I/O, use MPI Comm connect (etc.) and MPI Send / MPI Recv
 - <Time to perform 100 import/export> vs <matrix size in each procs>



MPI_S/R is bounded by connection time

Experiment (2) Weak scaling in case of one task

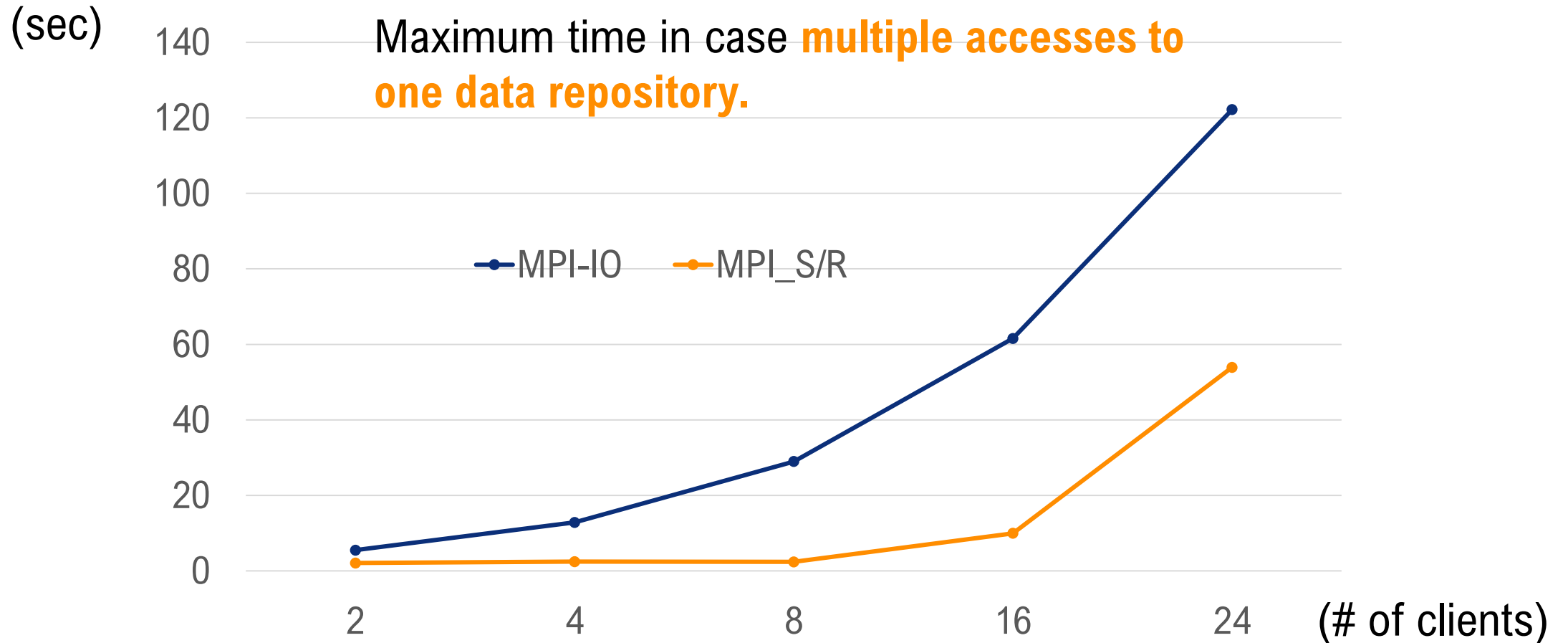
- Weak scaling
 - 1 client: 1000x1000 matrix for each process, a client is executed by 4x4 to 7x7 procs
 - 1 pdr: executed on 4x4 to 7x7 procs
 - MPI-IO: Current mSPMD implementation using MPI-IO
 - MPI_Send/Recv: No I/O, use MPI Comm connect (etc.) and MPI Send / MPI Recv
 - <Time to perform 100 import/export> vs <# of nodes>



Experiment(3) Multiple accesses (N tasks, one PDR)

- **n-clients vs 1-pdr**

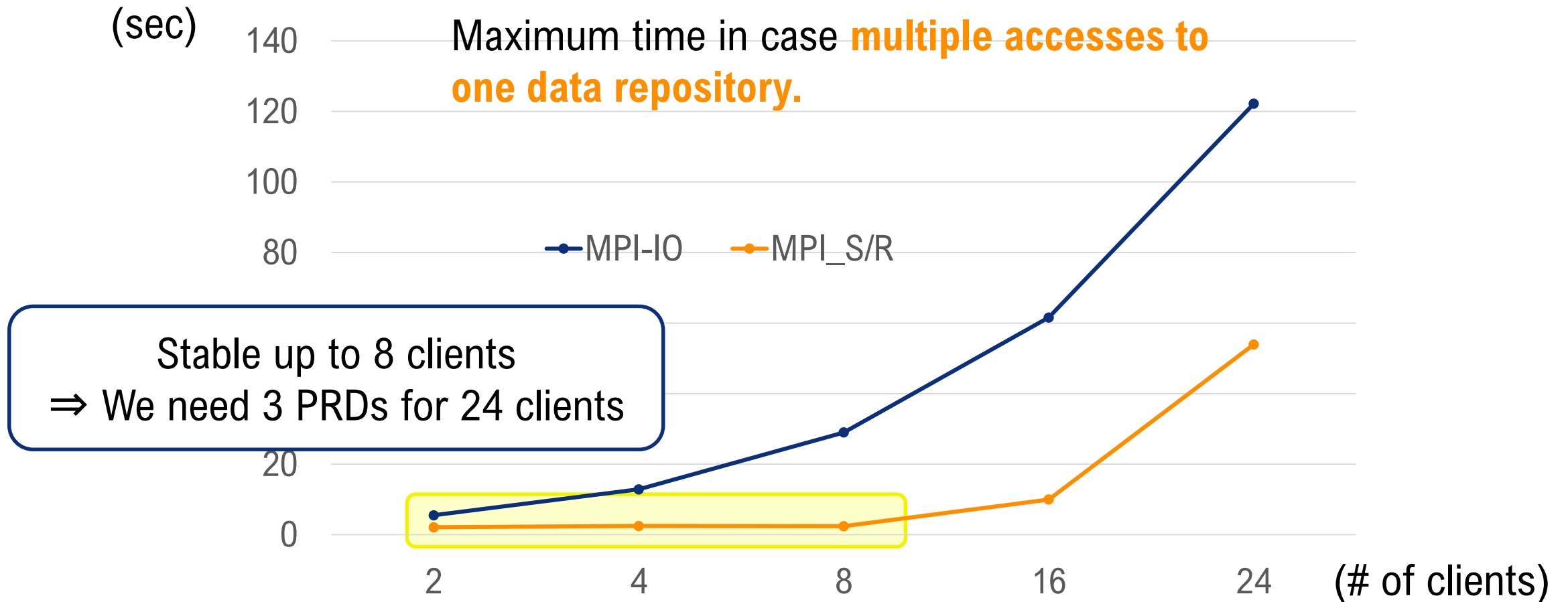
- 2 to 24 client: each client uses 16 procs, each client has 1000x1000 matrix
- 1 pdr: 16 procs



Experiment(3) Multiple accesses (N tasks, one PDR)

- **n-clients vs 1-pdr**

- 2 to 24 client: each client uses 16 procs, each client has 1000x1000 matrix
- 1 pdr: 16 procs



Agenda

- Motivation and background
- Data exchange issues and model
- Implementation through YAML user interface
- **Development of Block Gauss-Jordan Algorithm**

The Block Gauss Jordan (BGJ) Algorithm

Input: A (partitioned into $p \times p$ blocks)

Output: $B = A^{-1}$

For $k = 0$ to $p - 1$

1 $B_{kk} = A_{kk}^{-1}$

For $i = k + 1$ to $p - 1$

2 $A_{ki} = B_{kk} \times A_{ki}$

End For

For $i = 0$ to $p - 1$

If ($i \neq k$)

2 $B_{ik} = -A_{ik} \times B_{kk}$

End If

If ($i < k$)

2 $B_{ki} = B_{kk} \times B_{ki}$

End If

End For

For $i = 0$ to $p - 1$

If ($i \neq k$)

For $j = k + 1$ to $p - 1$

3 $A_{ij} = A_{ij} - A_{ik} \times A_{kj}$

End For

For $j = 0$ to $k - 1$

3 $B_{ij} = B_{ij} - A_{ik} \times B_{kj}$

End For

End If

End For

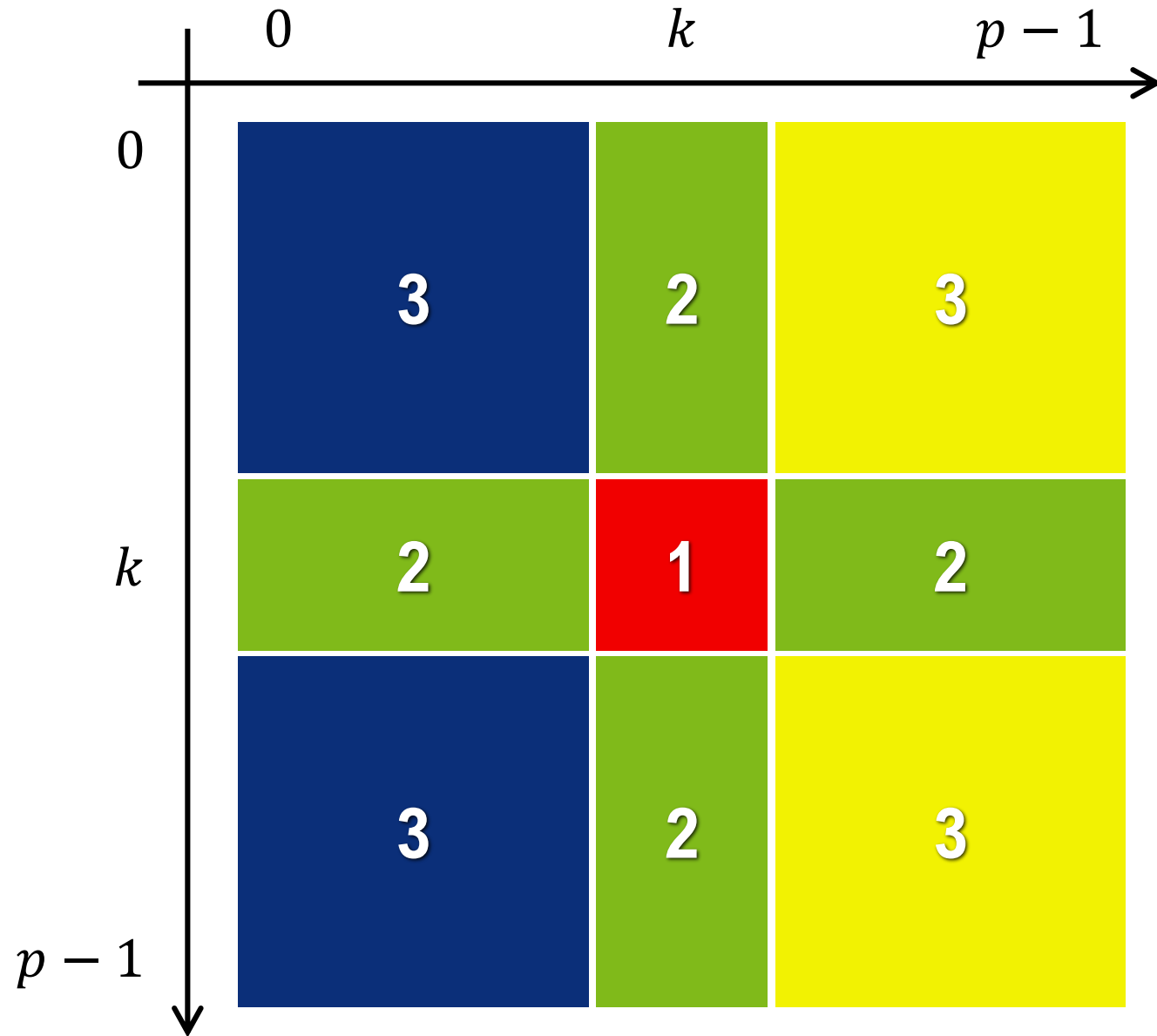
End For

1. Pivot

2. Update row and column

3. Update A and Update B

Illustration of BGJ steps



- 1 Pivot**
getrf_tr
- 2 Update row and column**
gemm for step 2
- 3 Update A**
gemm for step 3
- 3 Update B**
gemm for step 3

3 import/export strategies are compared

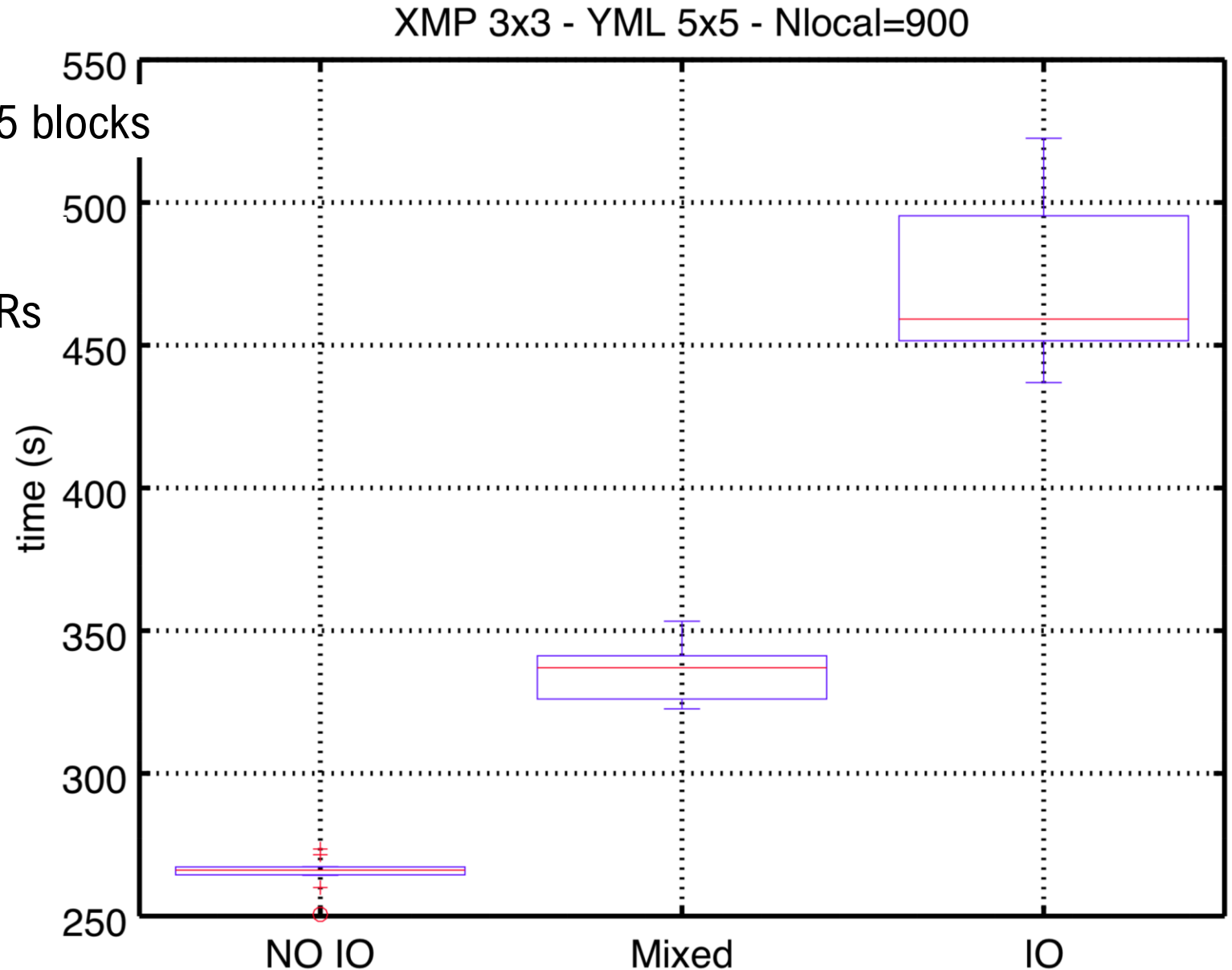
- MPI-IO
- Mix:MPI-IO and PDRs
 - After **Update row and column**, blocks are exported to a NFS
 - Otherwise, blocks are sent to a PDR
- PDR

Large block size

4500x4500 matrix is divided into 5x5 blocks

- 5x5 PDRs, each PDR 3x3 procs
⇒ $2 \times (5 \times 5) \times (3 \times 3) = 450$ cores for PDRs
- Each task uses 3x3 procs
- (maximum) 16 tasks in parallel
- 10 independent runs

NO I/O strategy = faster (1.73x speedup), more stable / Mixed = compromise **1.36x**, checkpoint, stable

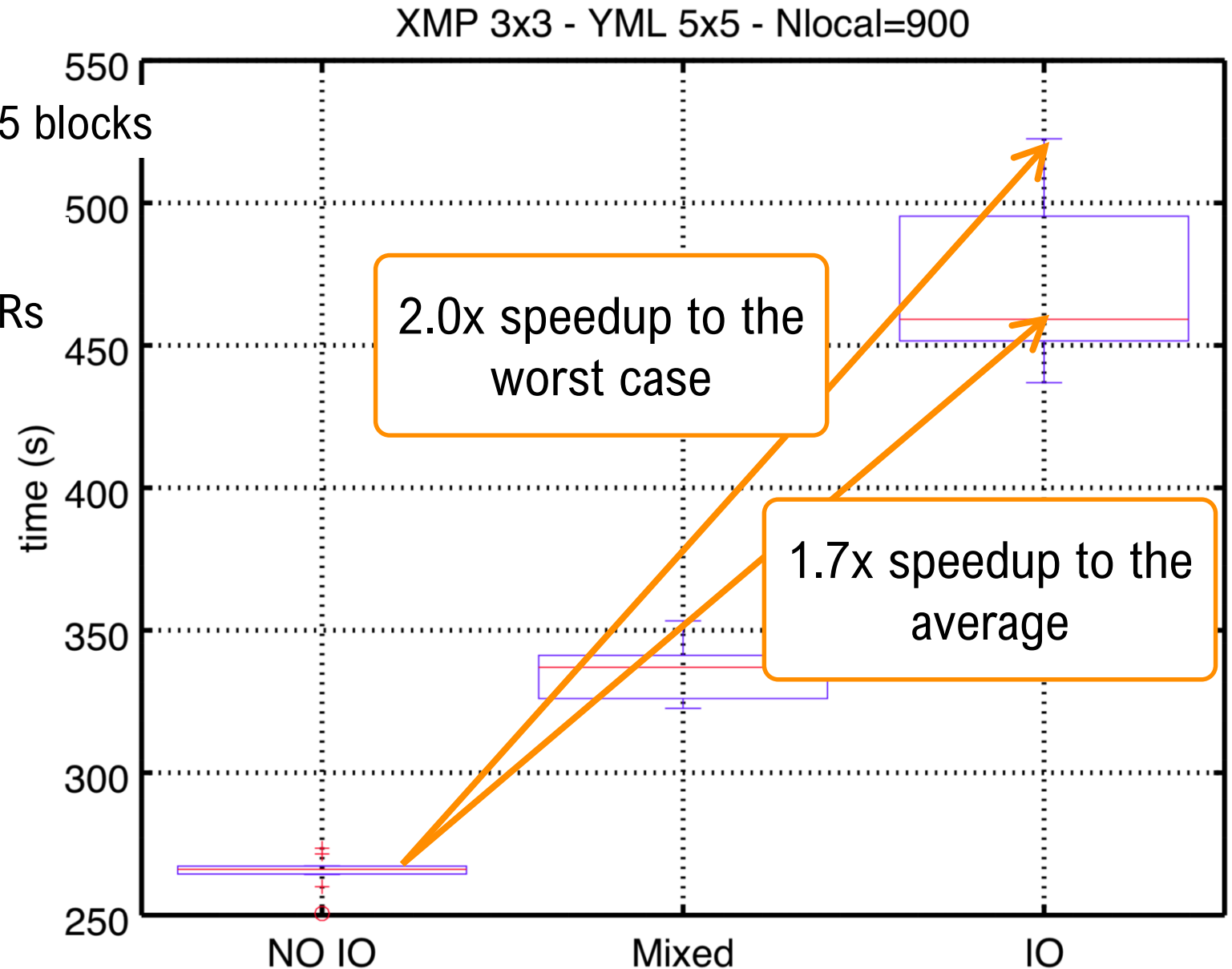


Large block size

4500x4500 matrix is divided into 5x5 blocks

- 5x5 PDRs, each PDR 3x3 procs
⇒ $2 \times (5 \times 5) \times (3 \times 3) = 450$ cores for PDRs
- Each task uses 3x3 procs
- (maximum) 16 tasks in parallel
- 10 independent runs

NO I/O strategy = faster (1.73x speedup), more stable / Mixed = compromise **1.36x**, checkpoint, stable

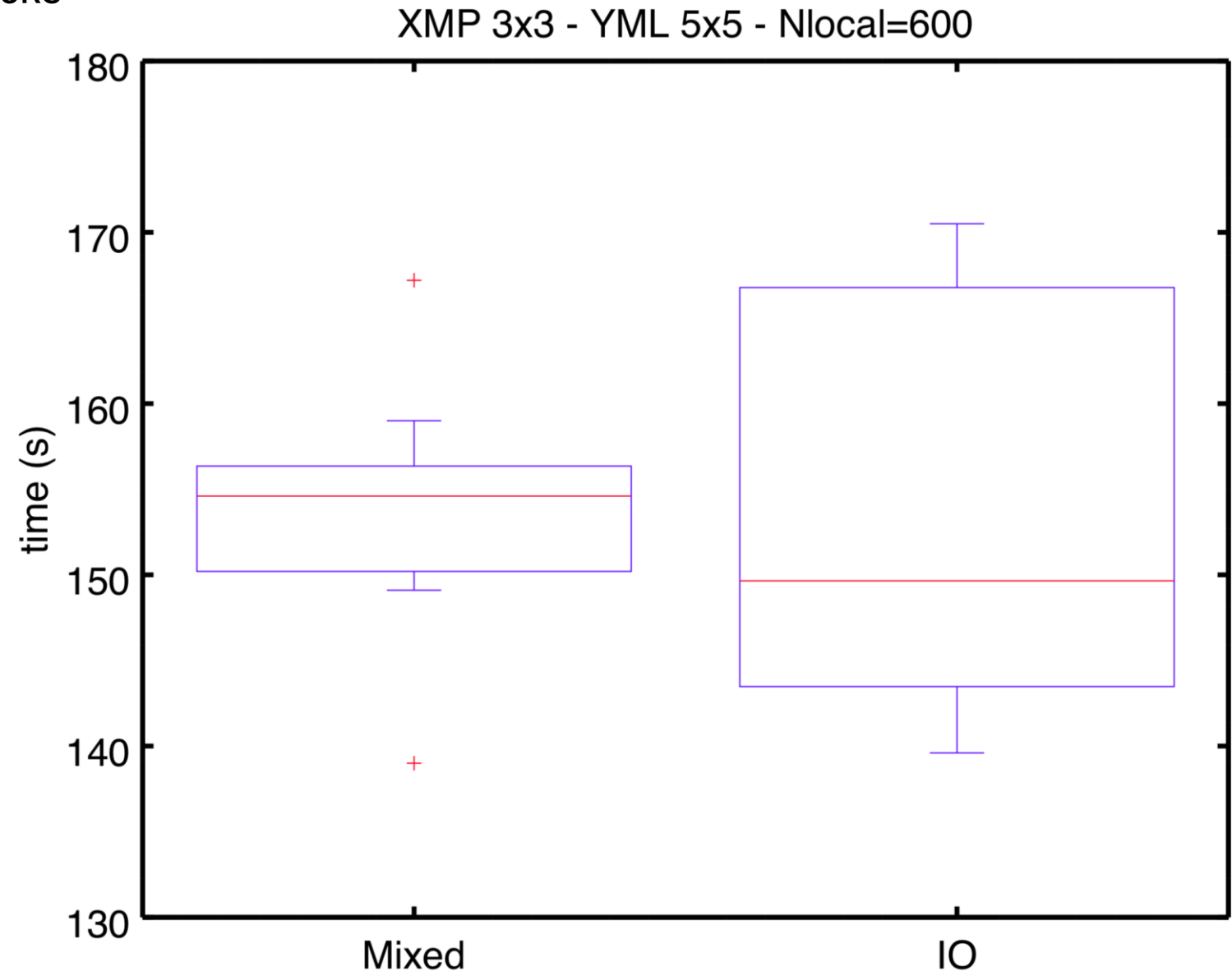


Small block size

3000x3000 matrix is divided into 5x5 blocks

- 5x5 PDRs, each PDR 3x3 procs
⇒ $2 \times (5 \times 5) \times (3 \times 3) = 450$ cores for PDRs
- Each task uses 3x3 procs
- (maximum) 16 tasks in parallel
- 10 independent runs

Mixed strategy **improve stability**
and limit loss of performance



Conclusion

Design

- Design of data exchange for Multi SPMD
 - **Graph of tasks + PGAS**
- **Design enables to extract and use information**
 - parallel programming model
 - execution model
 - architecture

Toward implementation

- mSPMD and data-exchange through a data repository
- Task scheduler
- YAML interface can be used for PDR
- **Choices: MPI-IO or MPI server (No I/O) or mix**
- **Speedup up to 2.0x for benchmark application (BGJ)**