Bridging Efficiency and Flexibility: Dynamic Resource Management in HPC

Authors

Iker Martín-Álvarez

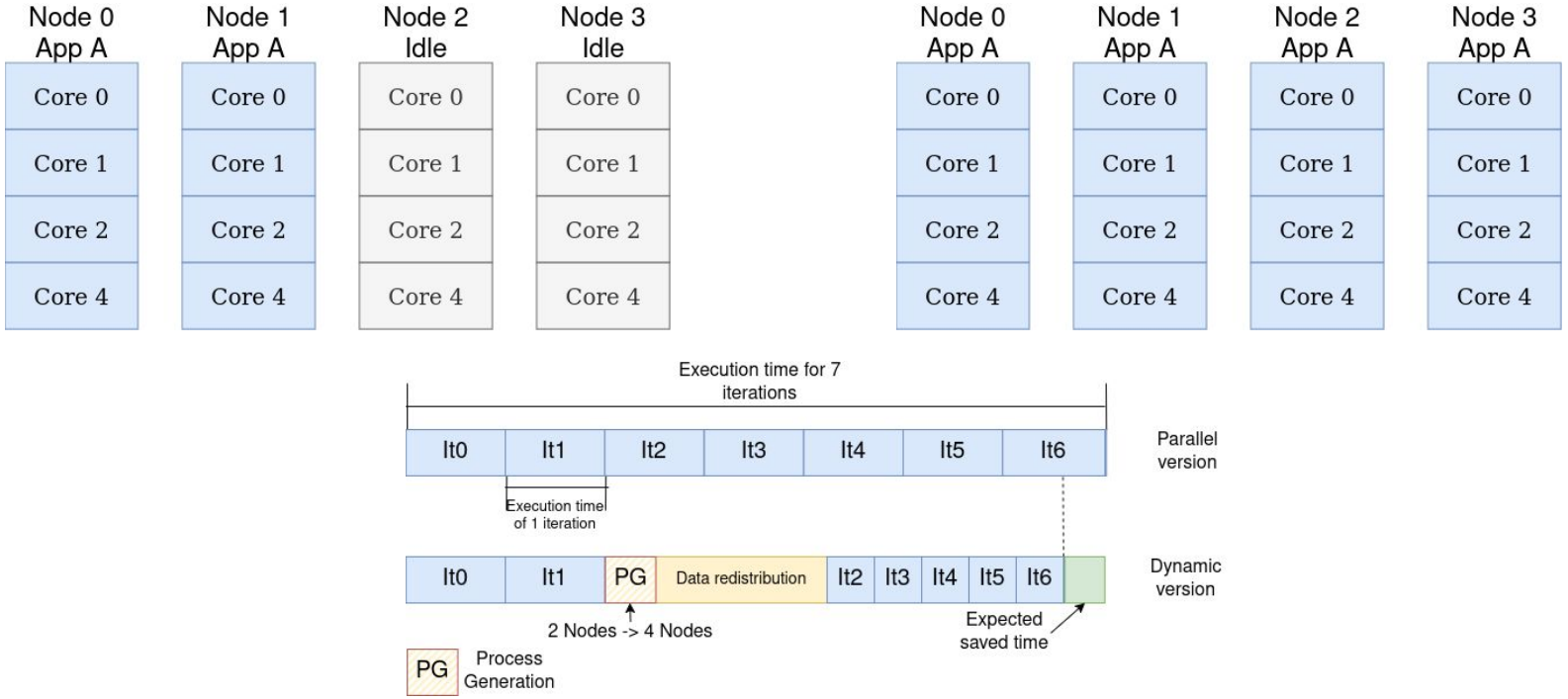
Outline:

1. Dynamic Resources
2. Proteo
3. Interfaces
4. Closing remarks

Outline:

1. Dynamic Resources
   - 1.1 What is it?
   - 1.2 Basics
   - 1.3 Benefits - Why using it?
   - 1.4 Challenges
2. Proteo
3. Interfaces
4. Closing remarks

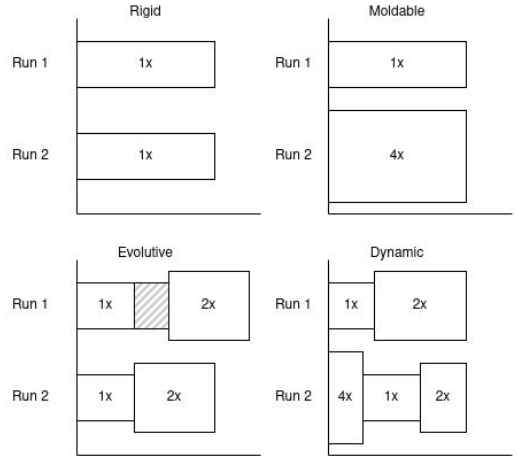
What is Dynamic Resource Management (DRM)?

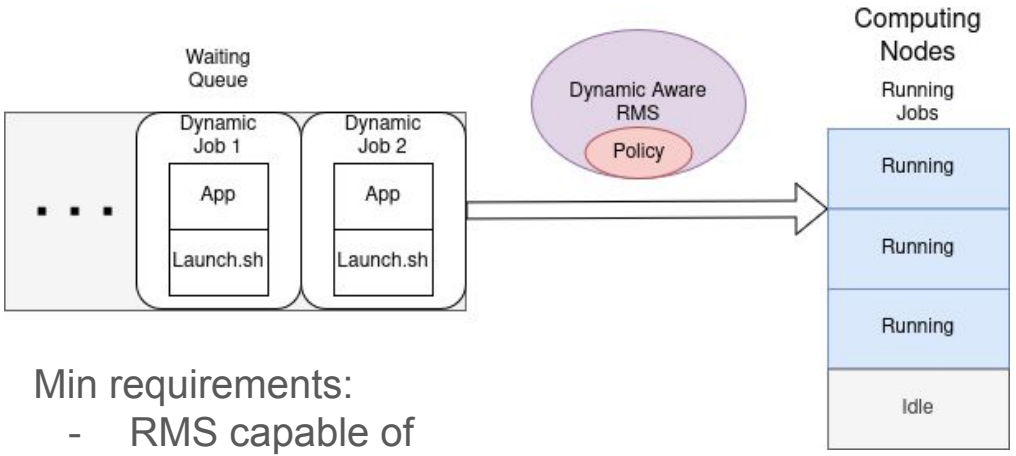


Basics

Types of jobs

Capable to resize	Who decides resource allocation	
	User 👤	RMS ⚙️
No	Rigid (Static)	Moldable (Static)
Yes	Evolutive	Dynamic





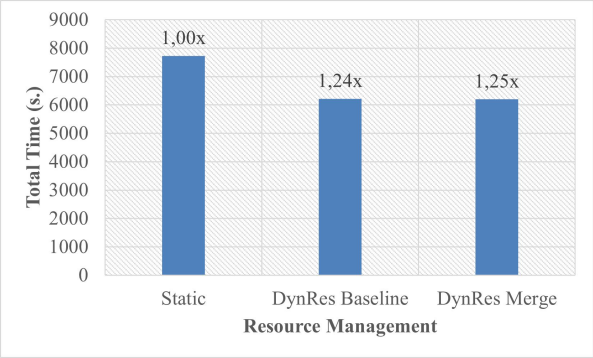
Min requirements:

- RMS capable of adding/removing resources from running jobs.
- Jobs capable of adapting to new amount of resources during execution.

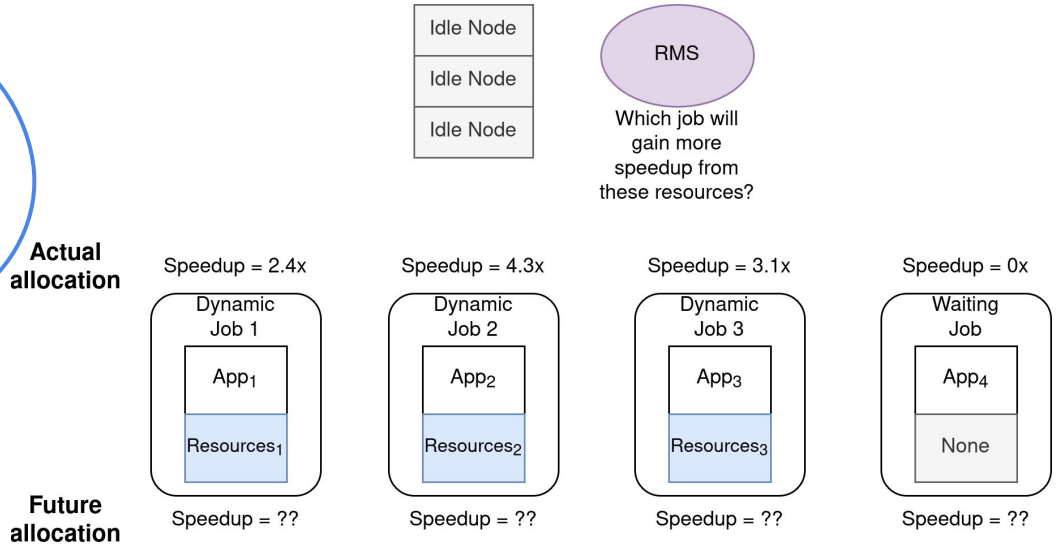
Benefits - Why using it?

Optimization objective:

- Increase system throughput
- Increase resource utilization
- Reduce workflow completion time
- Reduce job running time
- Reduce job turnaround time (wait + running time)

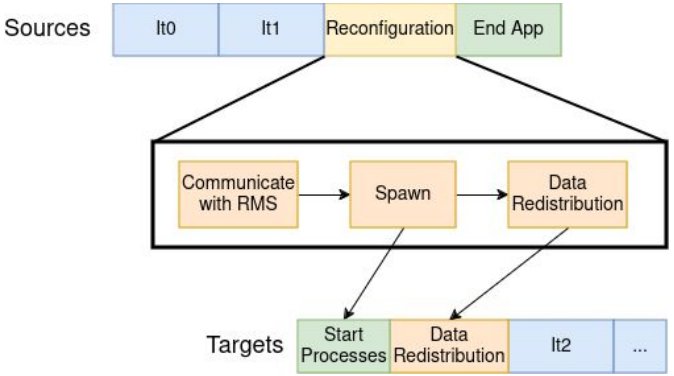


Example for reducing job running time

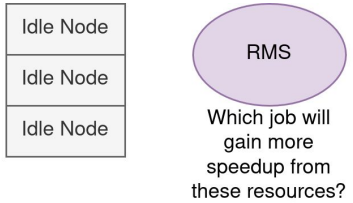


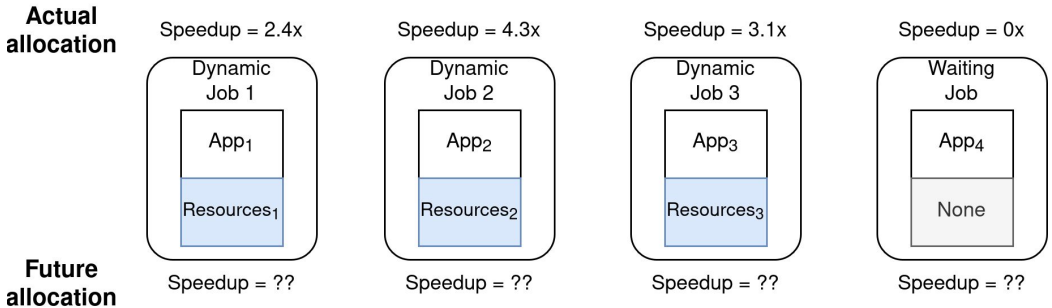
Challenges

- Scheduler requires collaboration from jobs to make informed decisions
- Applications require modifications to introduce reconfigurations



Example for reducing job running time



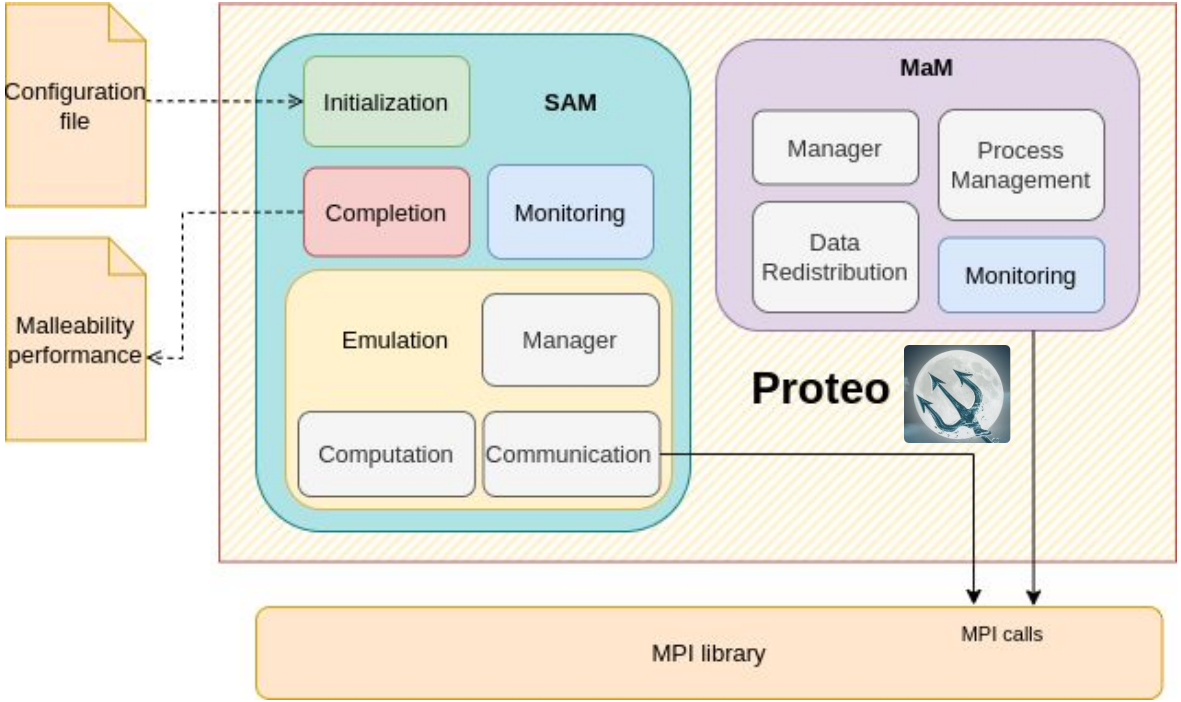


Outline:

1. Dynamic Resources
2. Proteo
   - 2.1 Overview
   - 2.2 Goals
   - 2.3 Modules
   - 2.4 Emulation (SAM)
   - 2.5 Reconfiguration (MaM)
3. Interfaces
4. Closing remarks

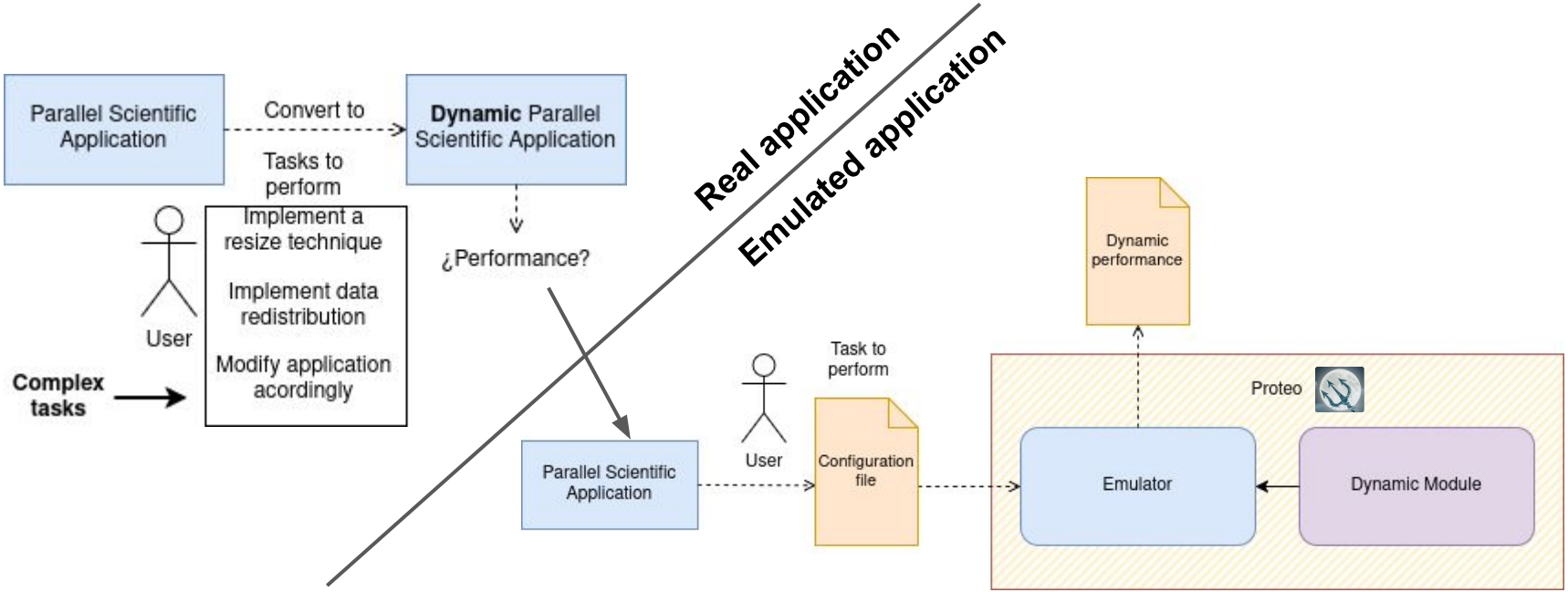
Proteo Overview

- Framework to emulate real applications as dynamic.
- Composed of two modules:
  - SAM: Takes care of emulation.
  - MaM: Takes care of reconfiguring the application.
- Requires a configuration file which describes how to emulate the real application.

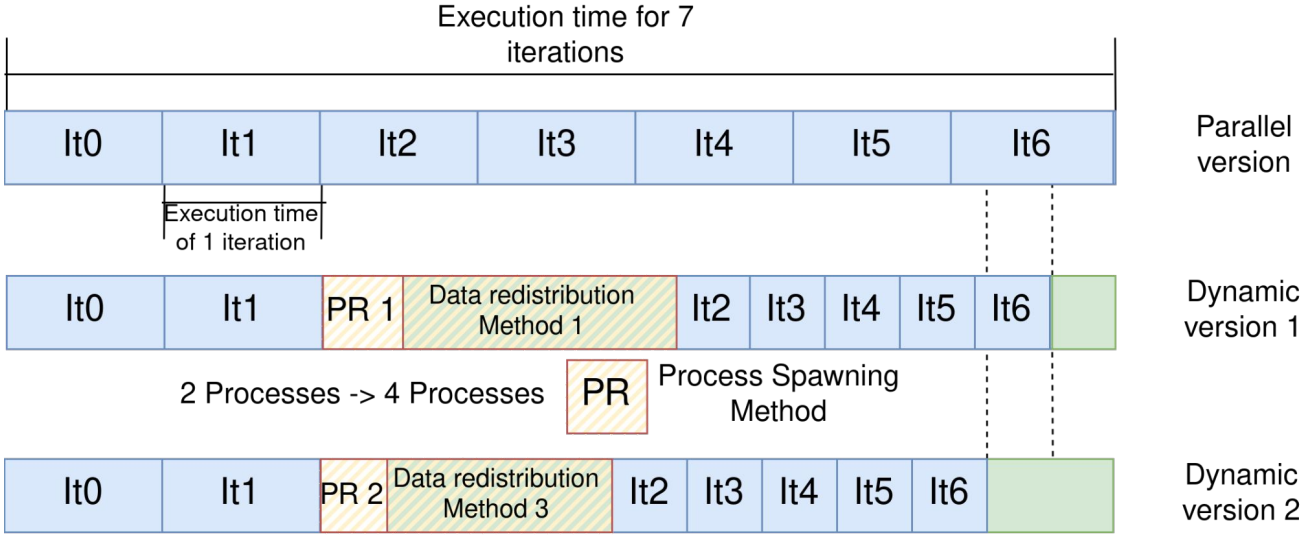


Related article

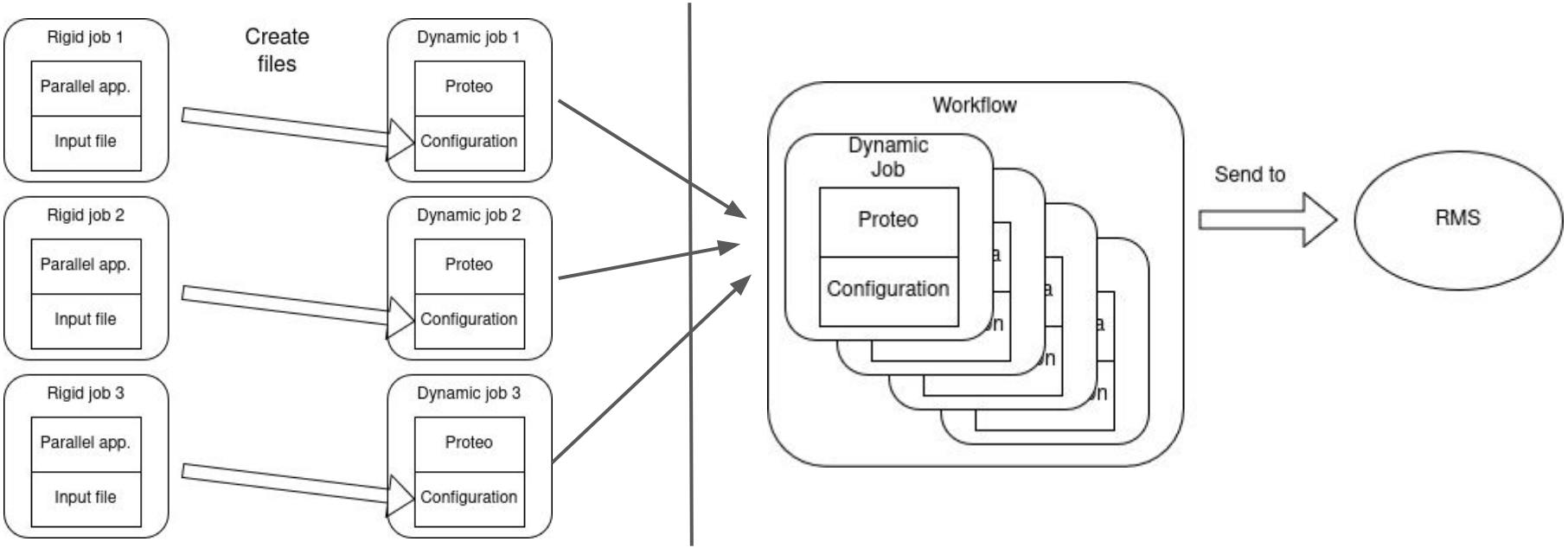
Goal 1: Check dynamic resources benefits



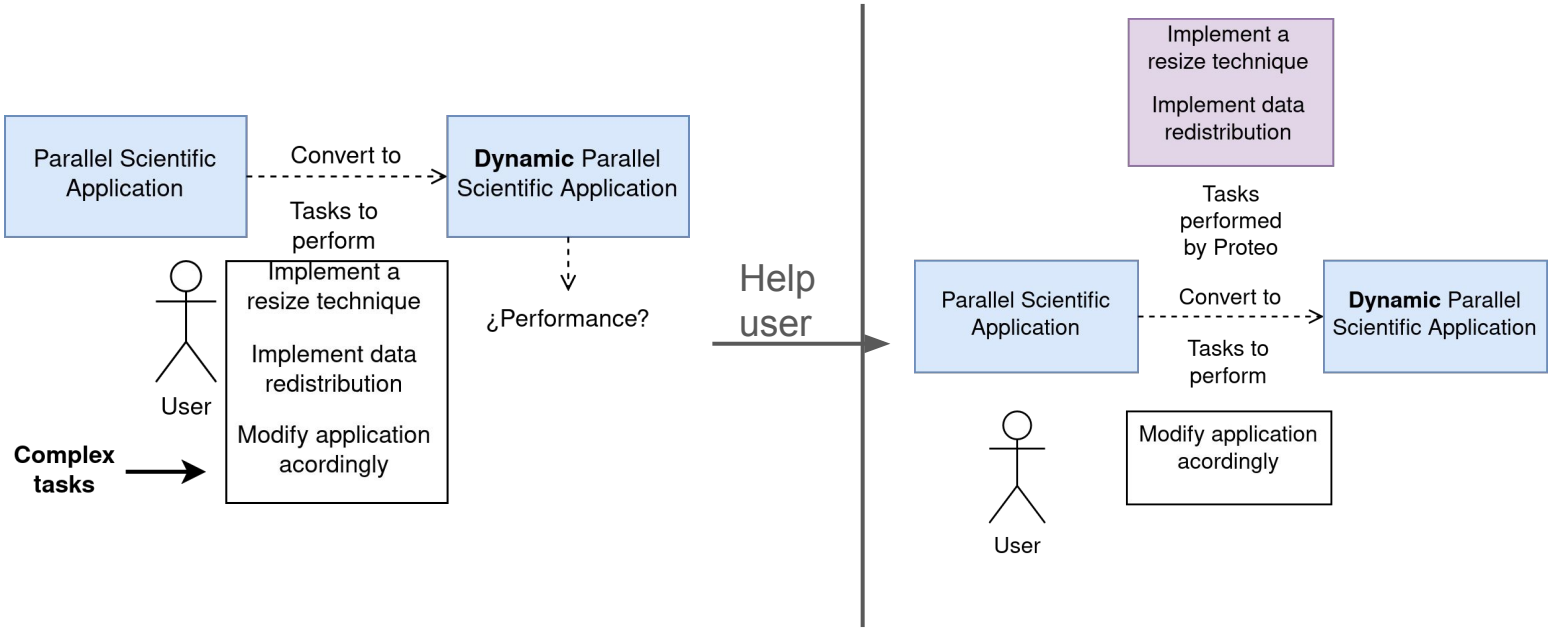
Goal 2: Reduce reconfiguration costs



Goal 3: Create dynamic workflows

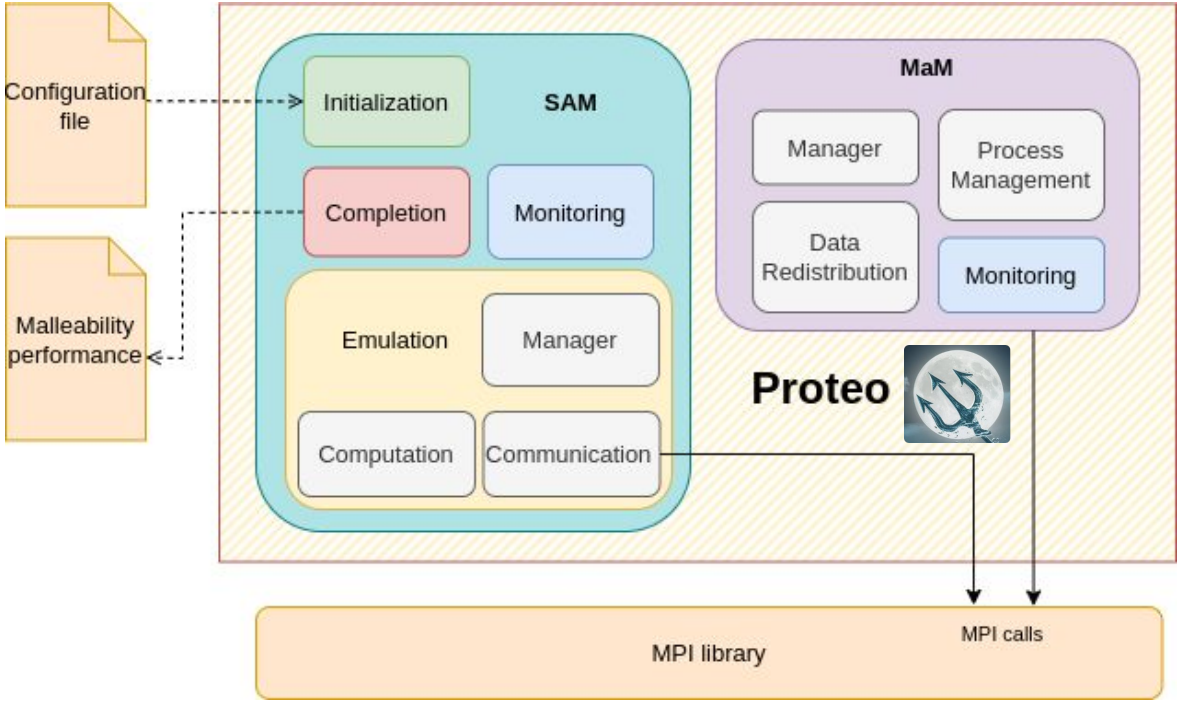


Goal 4: Help convert real applications into dynamic



Modules

- Framework to emulate real applications as dynamic.
- Composed of two modules:
  - SAM: Takes care of emulation.
  - MaM: Takes care of reconfiguring the application.
- Requires a configuration file which describes how to emulate the real application.

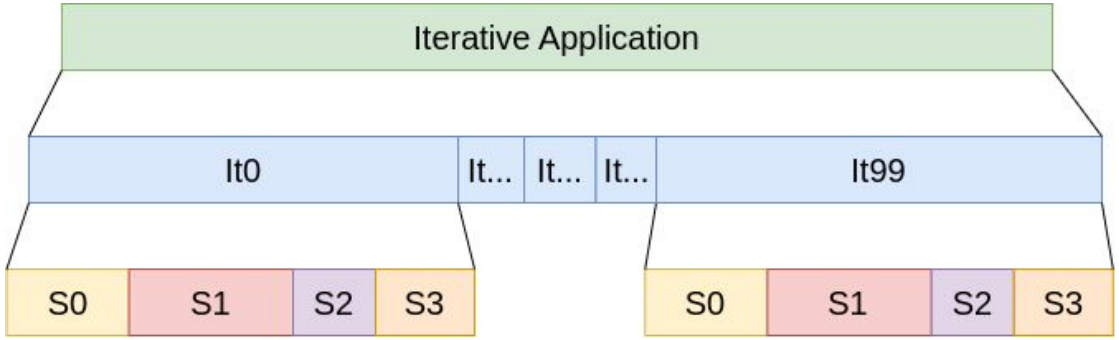


Repository Proteo

Proteo - Synthetic Application Module (SAM)

Application as a set of iterations

Emulate each iteration with the same sequence of stages



Stage types:

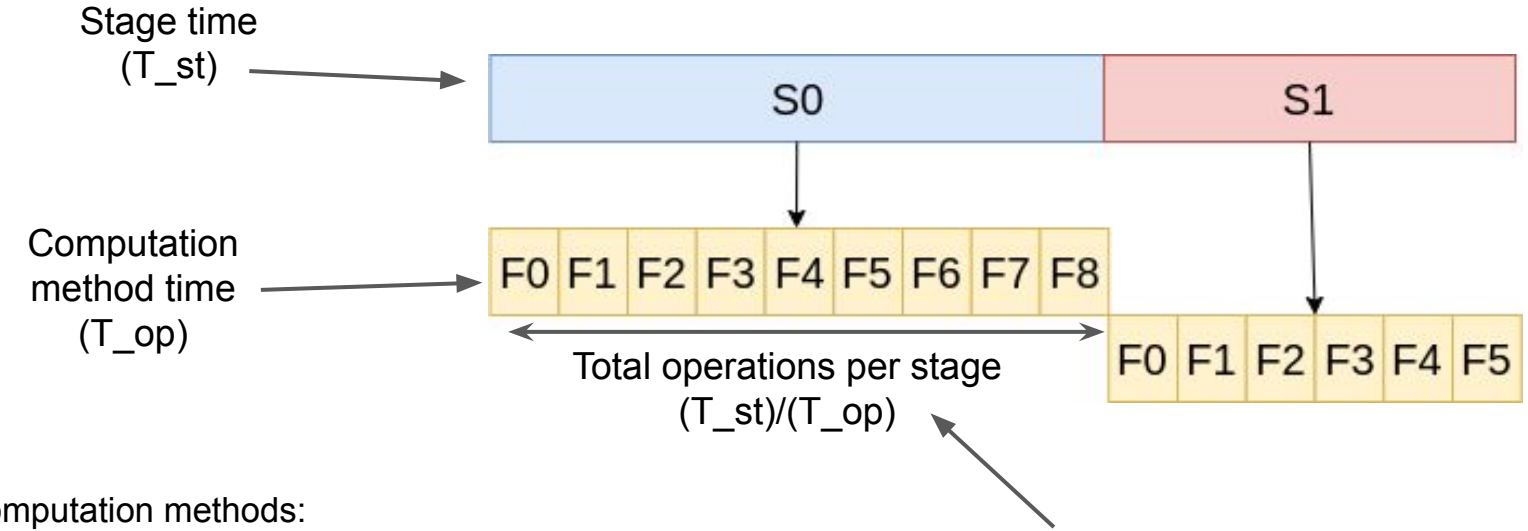
- Computation
- Communication

The cost of the stage Sx is constant between iterations

Each stage can have a different cost

Related article

SAM - Computation methods



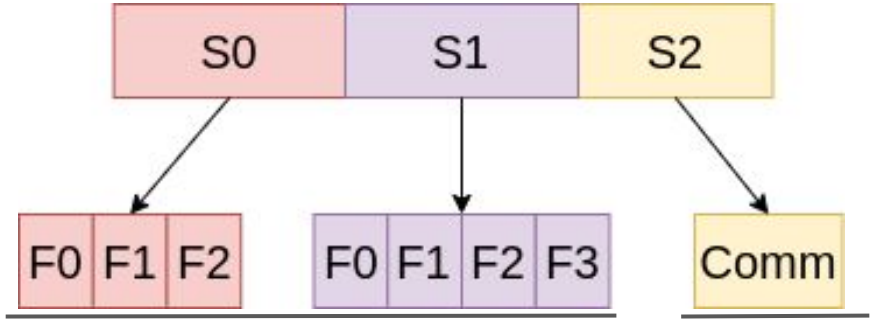
Computation methods:

- Compute-bound: Montecarlo method to calculate PI
- Memory-bound: Matrix-vector multiplication

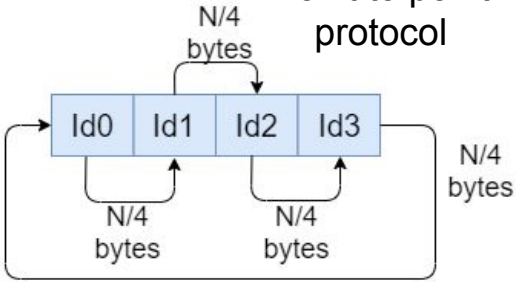
SAM - Communication methods

Communication methods:

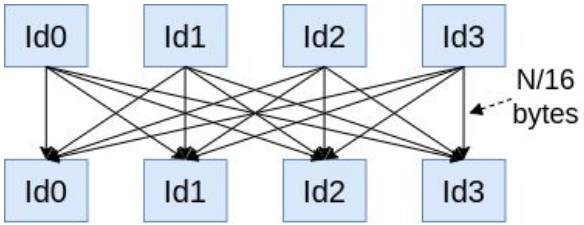
- Point to point (MPI_Send/Recv)
- Collective one to all (MPI_Bcast)
- Collective all to all (MPI_Allgatherv)
- Reduction (MPI_Reduce)
- Reduction all to all (MPI_Allreduce)

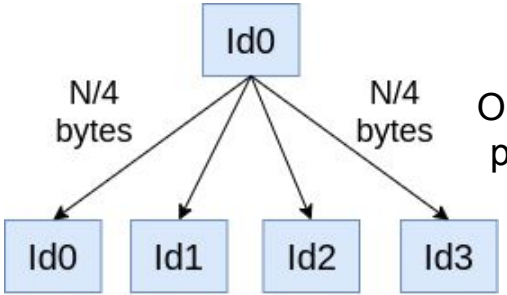


Point to point protocol



All to all protocol



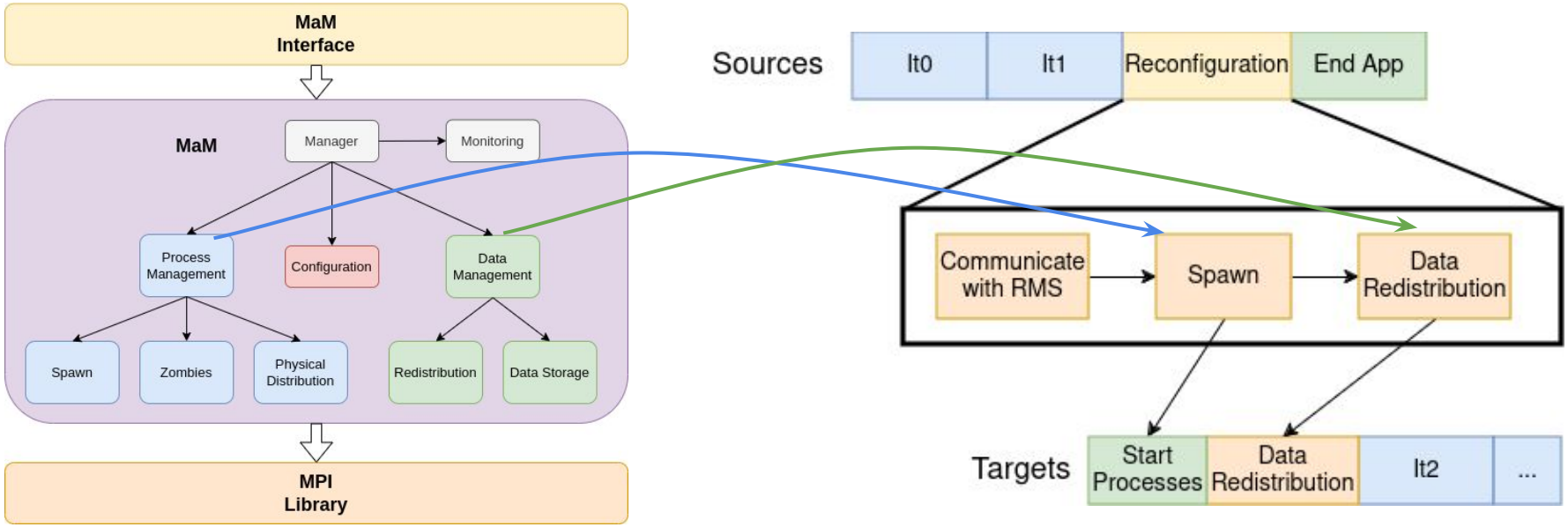


One to all protocol

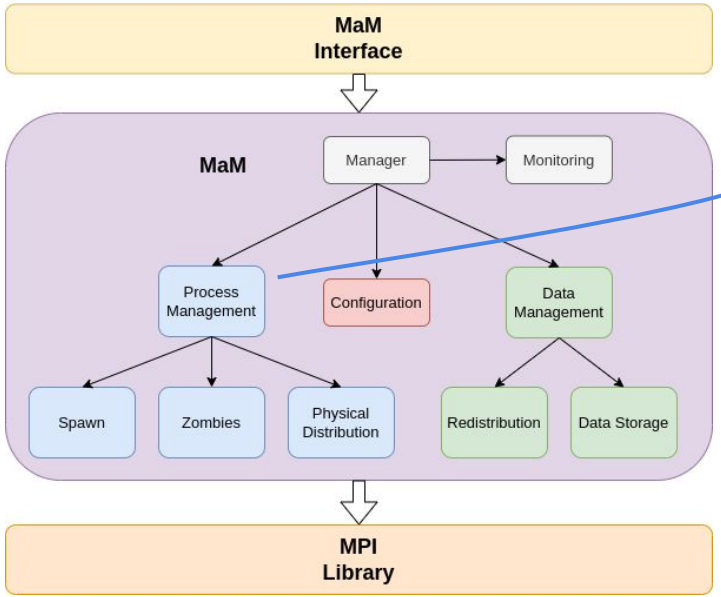
Outline:

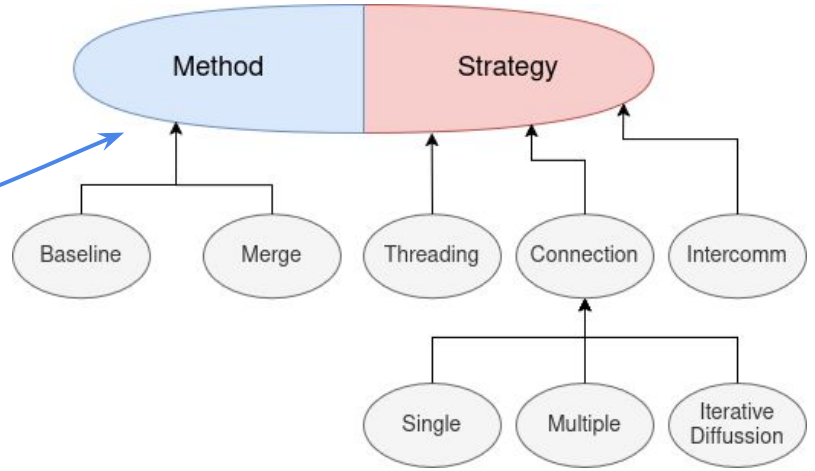
1. Dynamic Resources
2. Proteo
   - 2.1-4 …
   - 2.5 Reconfiguration (MaM)
     - 2.5.1 Overview
     - 2.5.2 Spawn Phase
     - 2.5.3 Redistribution Phase
3. Interfaces
4. Closing remarks

MaM Overview



Spawn Phase



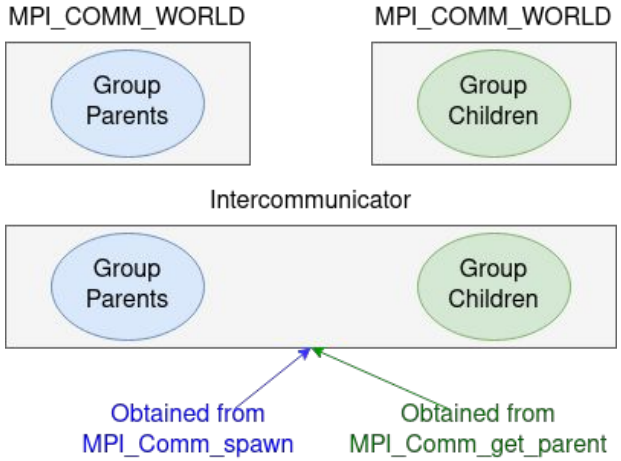


Related article

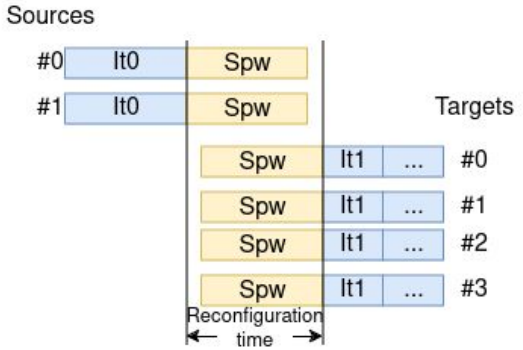
Baseline Method

2 Methods to spawn processes:

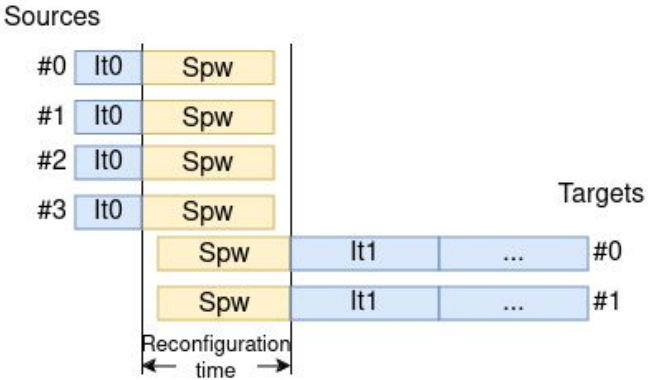
- **Baseline**
- Merge



Expand

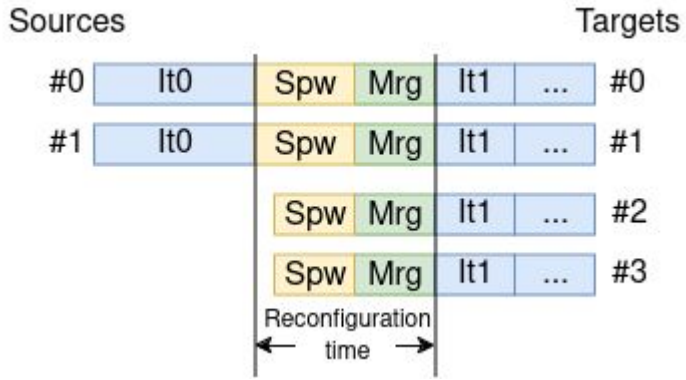


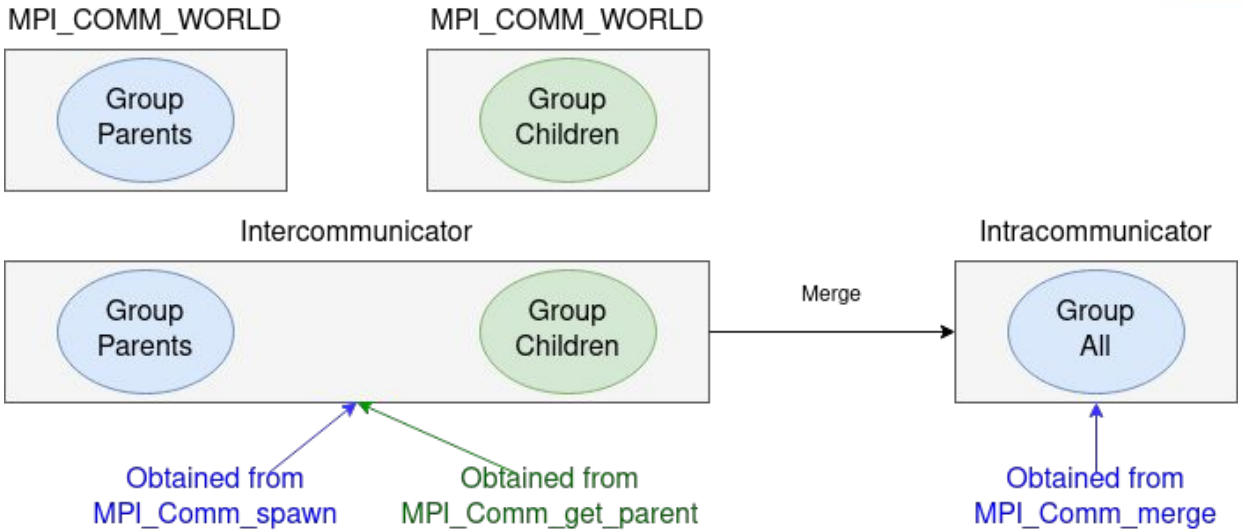
Shrink



Merge Method - Expand

2 Methods to spawn processes:
- Baseline
- **Merge**

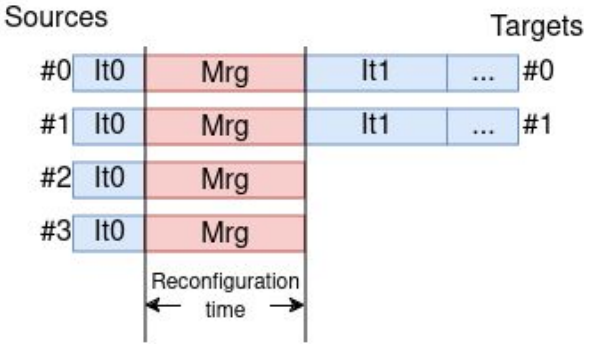


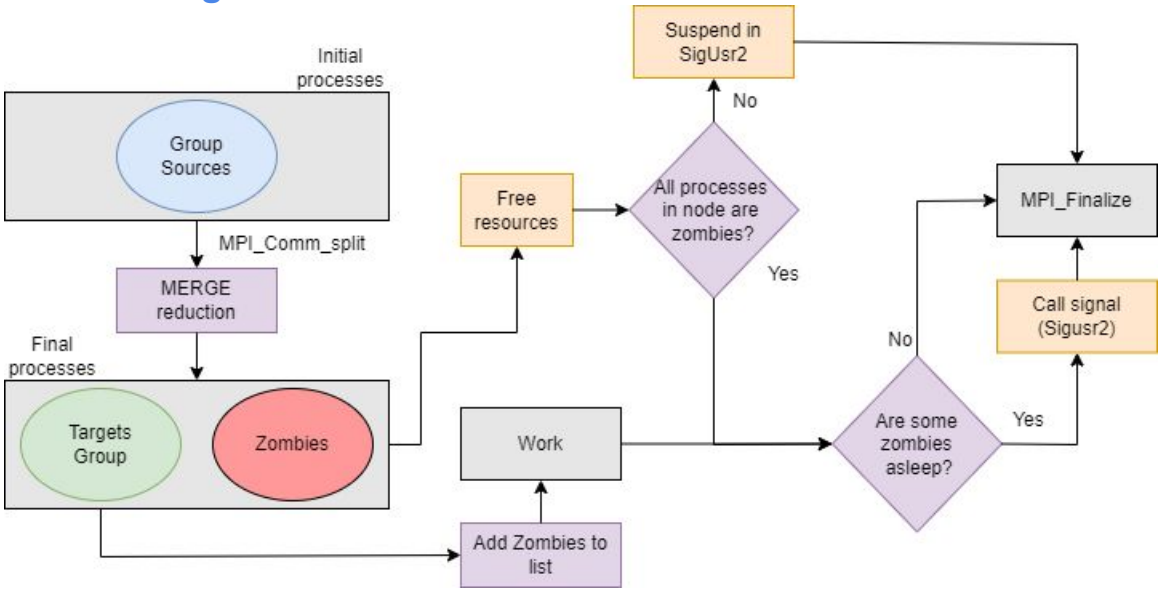


Merge Method - Shrink

2 Methods to spawn processes:

- Baseline
- **Merge**





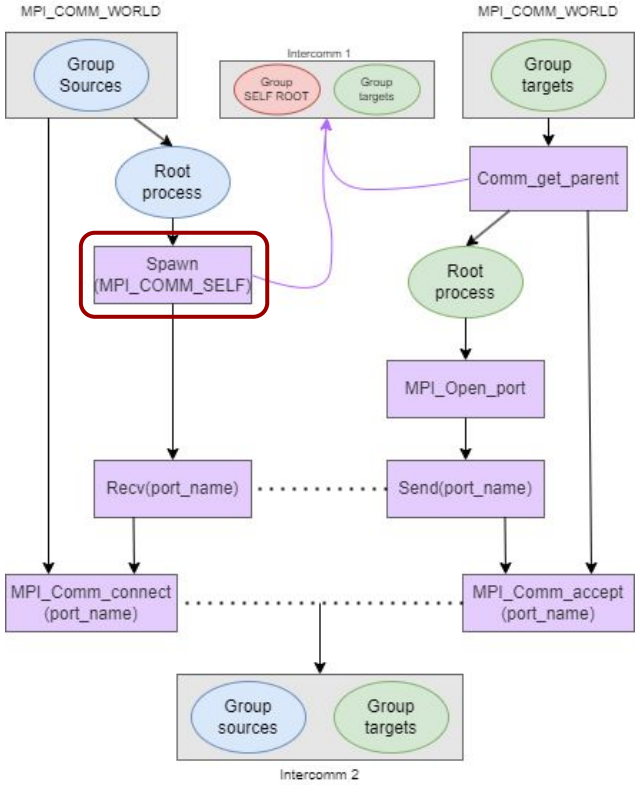
Strategies

5 Strategies to spawn processes:

- **Single**
- Threading
- Multiple
- Iterative Diffusion

The user can:

- Use neither.
- Use one of them.
- Use many.

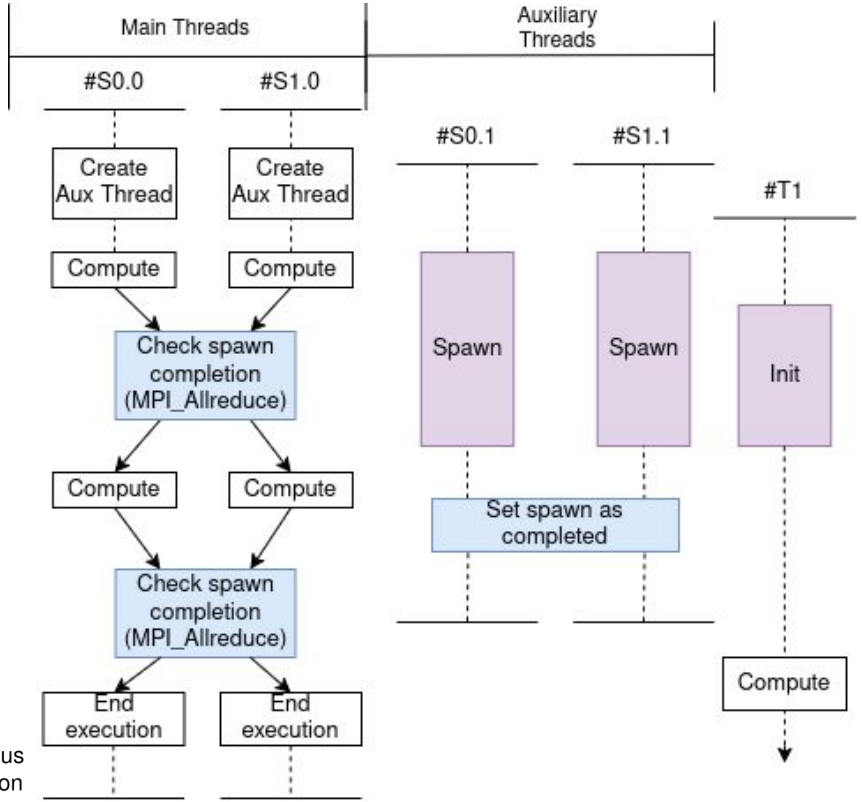


Goal: Reduce involved processes during spawn operations

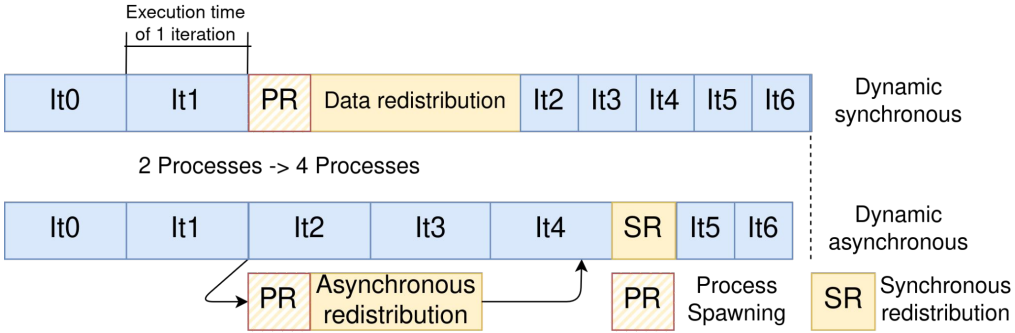
Strategies

5 Strategies to spawn processes:

- Single
- **Threading**
- Multiple
- Iterative Diffusion



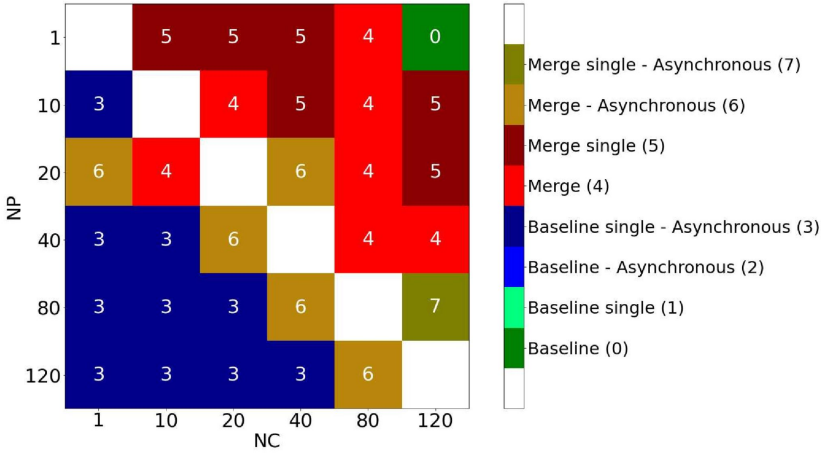
Goal: Allow computation and reconfiguration to overlap



Process management results

- Which combination of methods/strategies perform better?
- Executed application is Conjugate Gradient (CG)

Preferred methods



Expansion times

NP	NC	RT_S(s) Baseline	Baseline single	Merge	Merge single
1	10	0,316	0,313	**0,284**	0,289
	20	0,861	1,035	**0,716**	0,721
	40	0,861	0,995	**0,799**	0,809
	80	0,989	1,075	**0,932**	1,211
	120	**0,912**	1,029	0,992	1,023
10	20	1,286	1,654	**0,477**	0,486
	40	1,213	1,635	0,766	**0,743**
	80	1,293	1,693	**0,861**	0,888
	120	1,315	1,636	**0,891**	0,915
20	40	1,304	1,991	**0,790**	0,821
	80	1,407	1,932	**0,864**	0,958
	120	1,413	1,870	1,089	**1,071**
40	80	1,429	2,022	0,894	**0,877**
	120	1,526	2,113	**0,923**	0,942
80	120	1,522	2,316	**0,906**	0,996

Shrinking times

NP	NC	RT_S(s) Baseline	Baseline single	Merge
10	1	0,200	0,205	**0,001**
20	1	0,400	0,427	**0,001**
	10	0,933	1,220	**0,001**
40	1	0,400	0,423	**0,030**
	10	0,883	1,165	**0,025**
	20	1,271	1,740	**0,116**
80	1	0,388	0,418	**0,217**
	10	0,881	1,189	**0,181**
	20	1,262	1,826	**0,149**
	40	1,415	2,039	**0,148**
120	1	0,375	0,424	**0,231**
	10	0,953	1,251	**0,148**
	20	1,229	1,776	**0,178**
	40	1,300	2,102	**0,351**
	80	1,529	2,235	**0,156**

Related article

Strategies

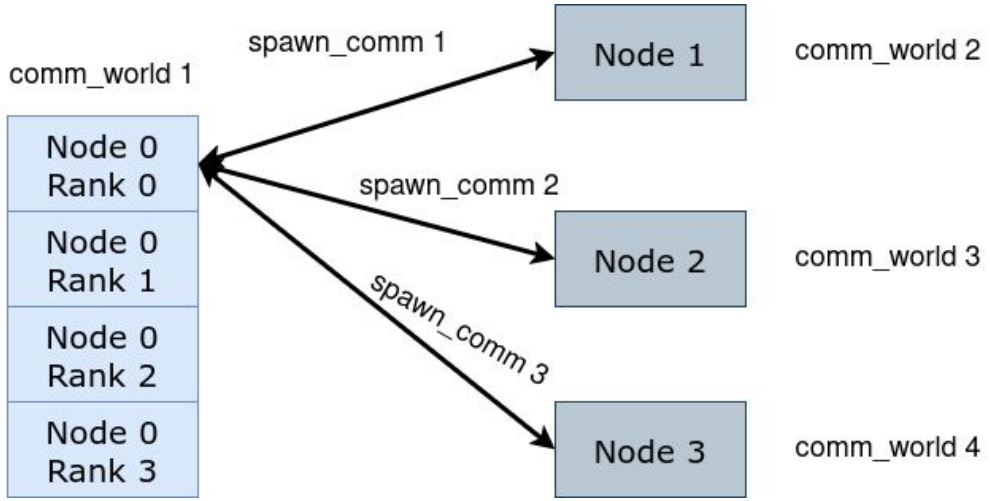
5 Strategies to spawn processes:

- Single
- Threading
- **Multiple**
- Iterative Diffusion

The user can:

- Use neither.
- Use one of them.
- Use many.

Goal: Remove MPI Comm World limitation for removing ranks



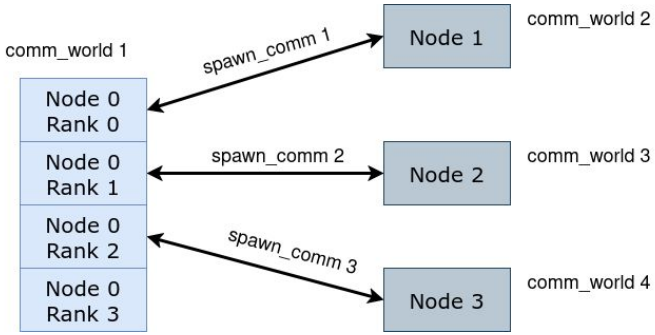
Strategies

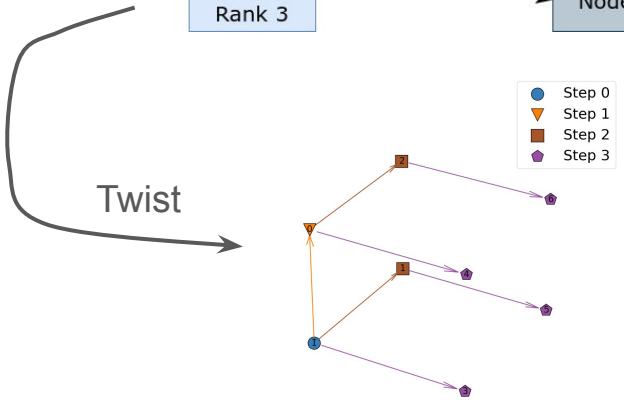
5 Strategies to spawn processes:

- Single
- Threading
- Multiple
- **Iterative Diffusion (WIP)**

The user can:

- Use neither.
- Use one of them.
- Use many.

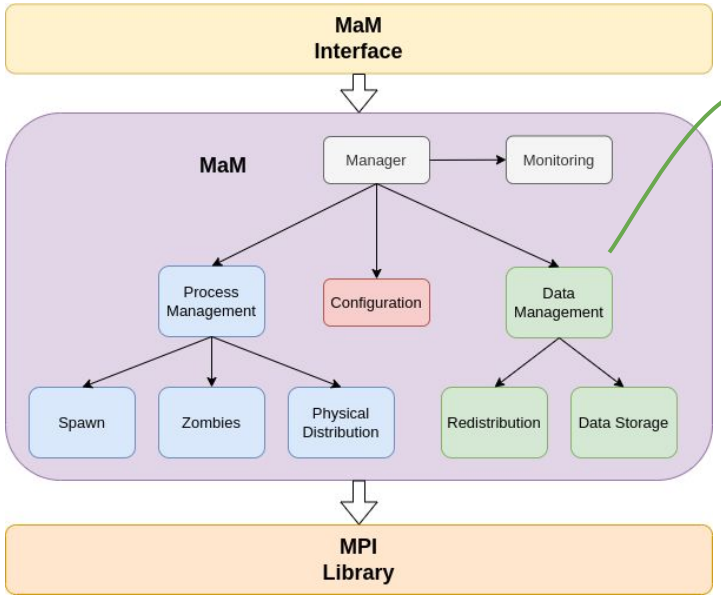


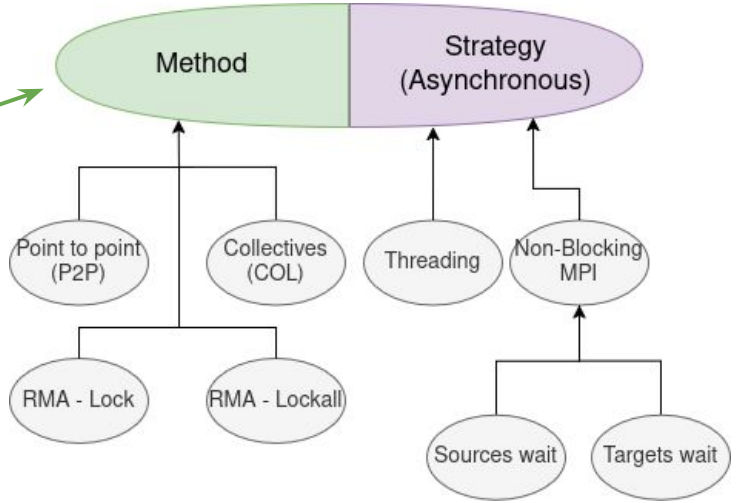


Example: 1 cpu per node
Obj: Get to 64 nodes

Example: 2 cpu per node
Obj: Get to 64 nodes

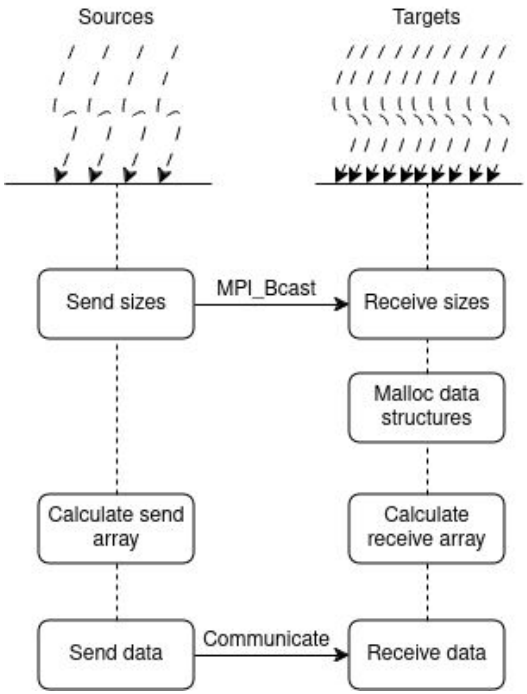
Redistribution Phase

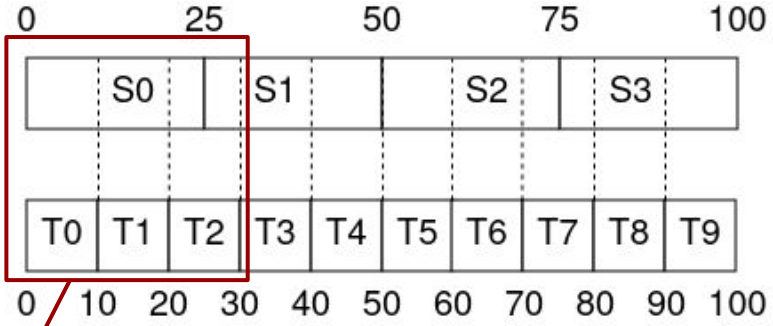




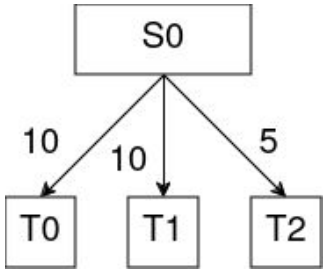
[Related article]

Baseline - Steps



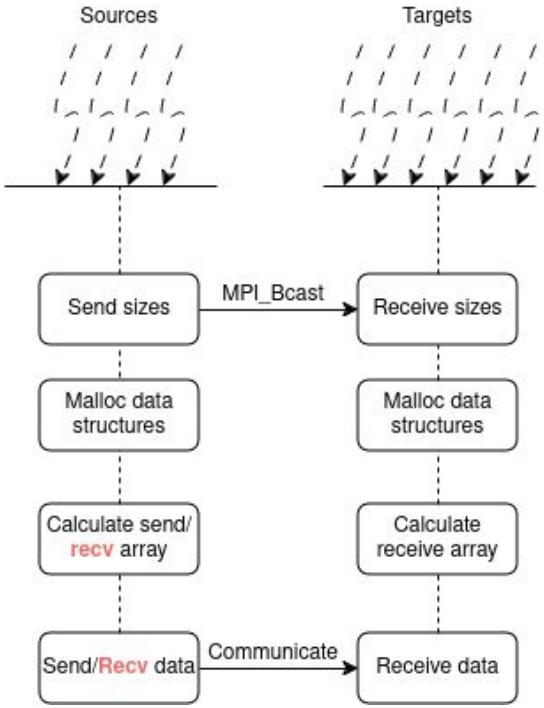


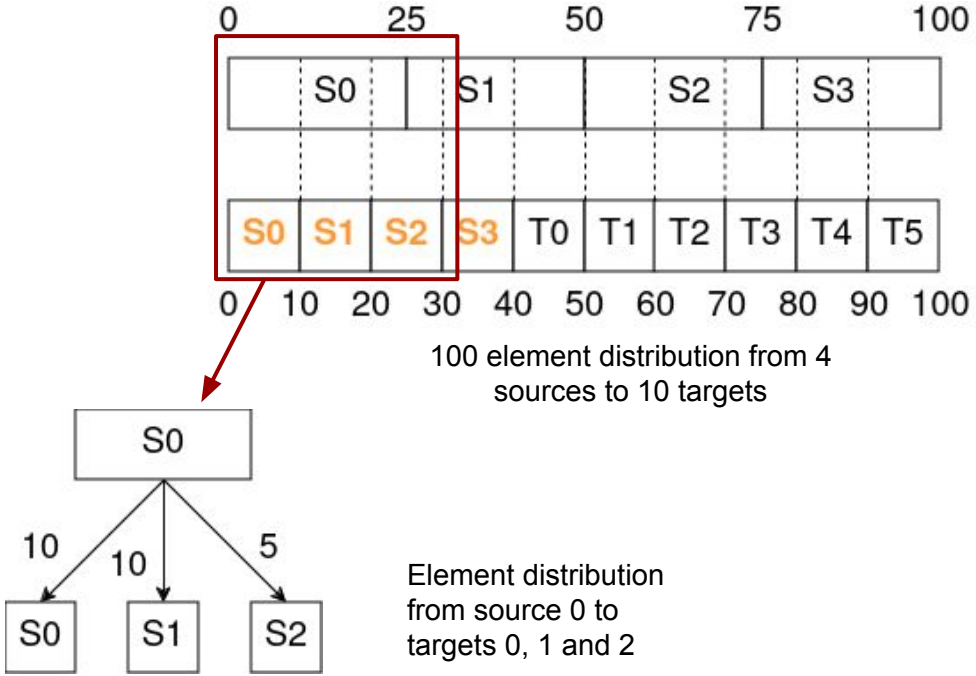
100 element distribution from 4 sources to 10 targets.



Element distribution from source 0 to targets 0, 1 and 2

Merge - Steps





100 element distribution from 4 sources to 10 targets

Element distribution from source 0 to targets 0, 1 and 2

MaM Data Storage

Data are divided into:

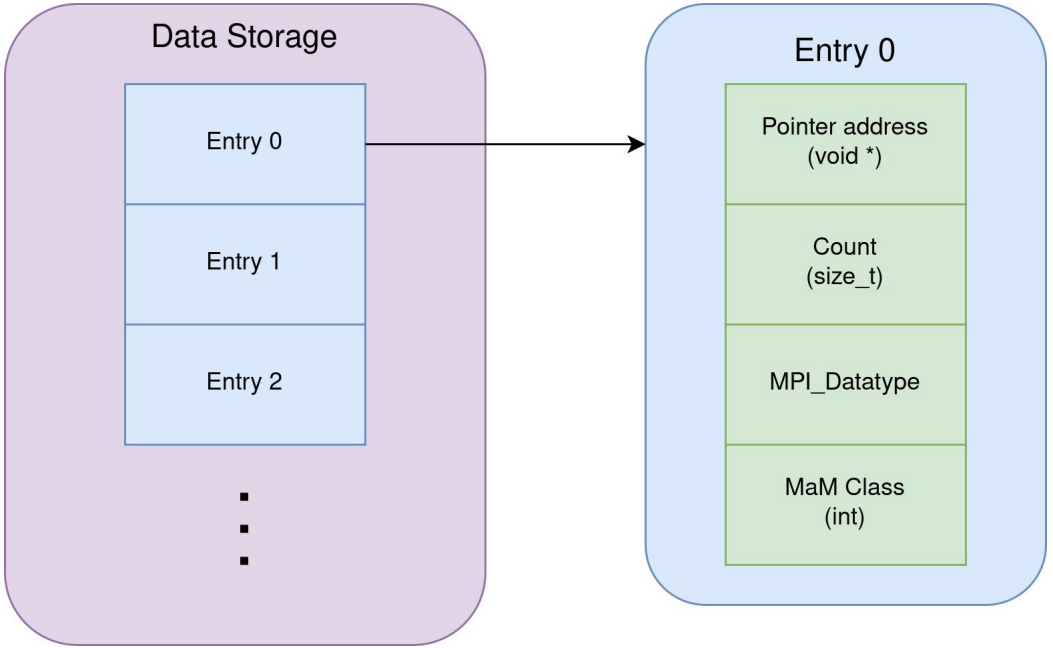
- Replicated
- Distributed

Data are also classified into:

- Variable
- Constant

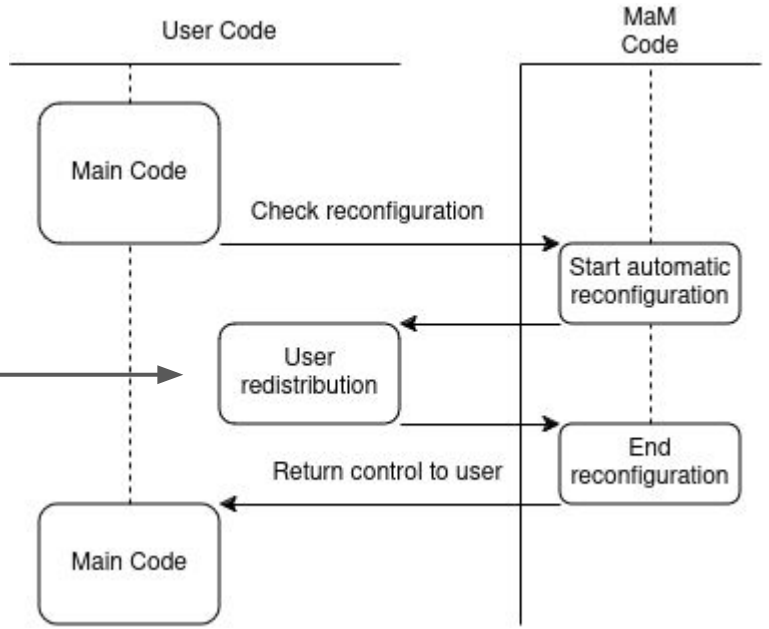
Sent in blocking calls

Sent either with blocking or non-blocking calls



User function - Semiautomatic Data Redistribution

Function created by the user



- Spawn ranks
- Asynchronous comm

- Synchronous comm
- Remove ranks

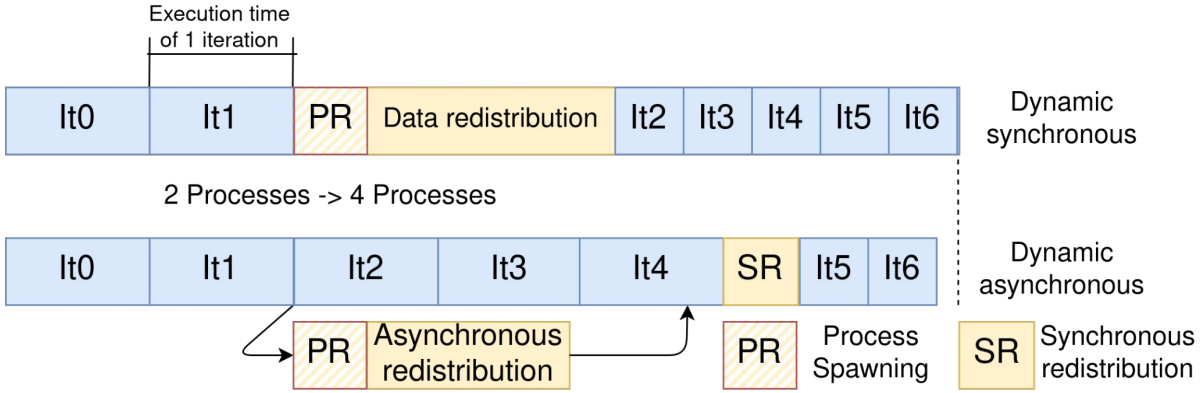
Strategies

3 Strategies to redistribute data:

- Non-Blocking
  - Sources Wait
  - Targets Wait
- Threading

The user can:

- Use neither
- Use one of them
- Use both 



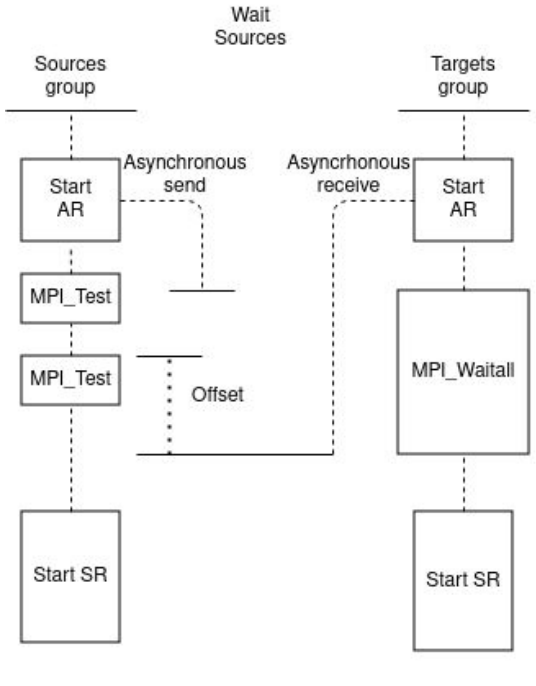
Strategies

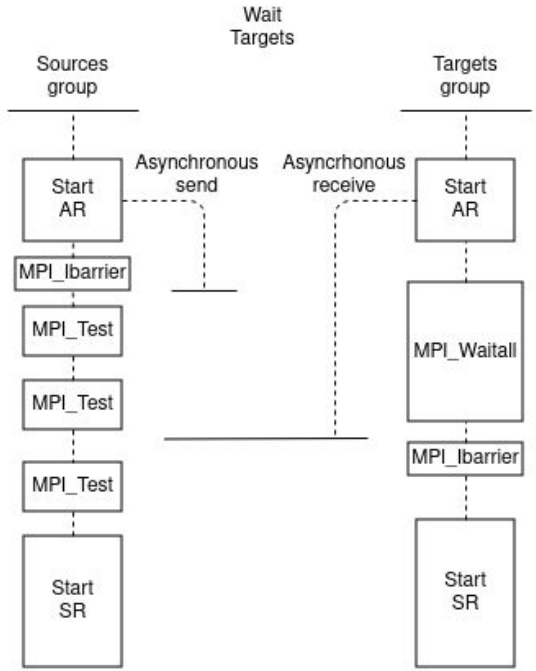
3 Strategies to redistribute data:

- **Non-Blocking - Sources Wait**
- **Non-Blocking - Targets Wait**
- Threading

The user can:

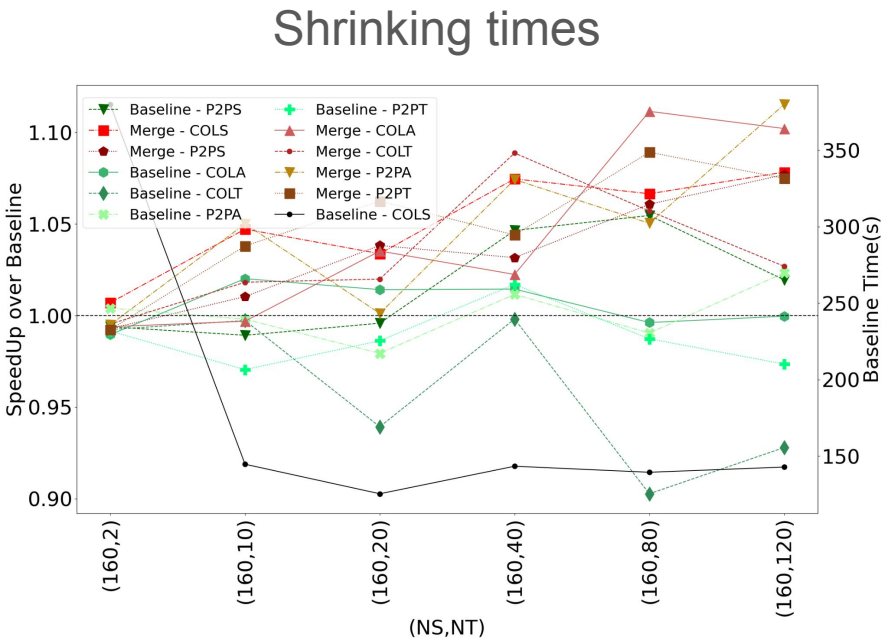
- Use neither
- Use one of them
- Use both





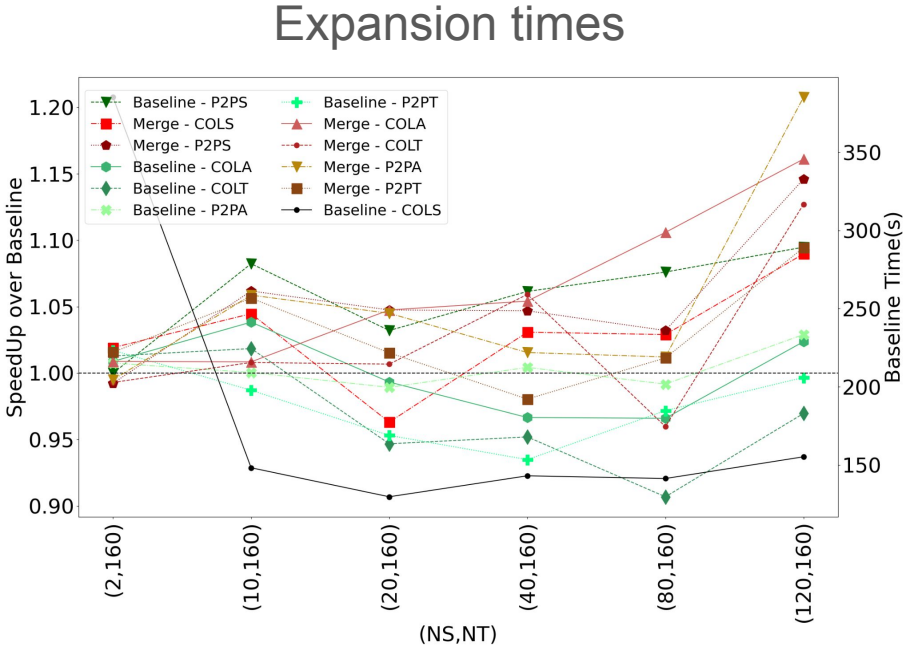
Data redistribution results

- Which combination of methods/strategies perform better?
- Executed application is Conjugate Gradient (CG)



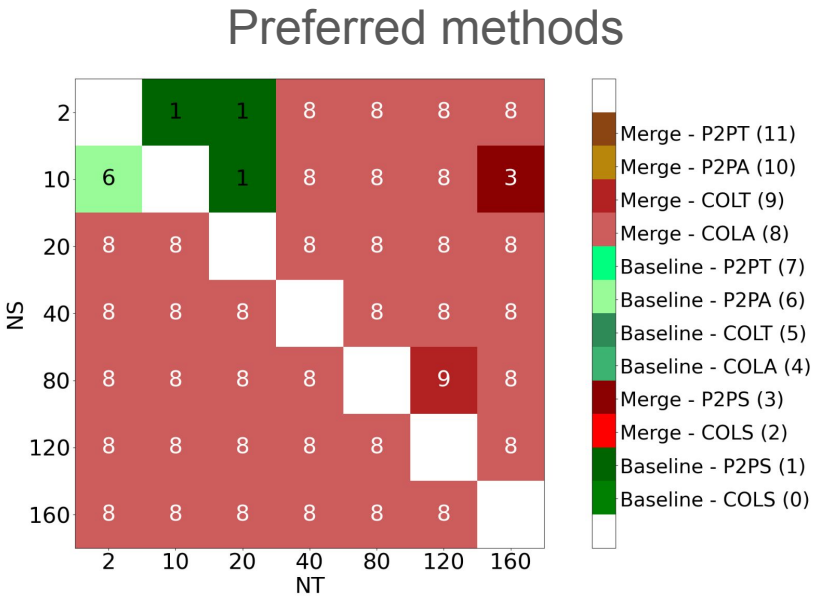
Data redistribution results

- Which combination of methods/strategies perform better?
- Executed application is Conjugate Gradient (CG)



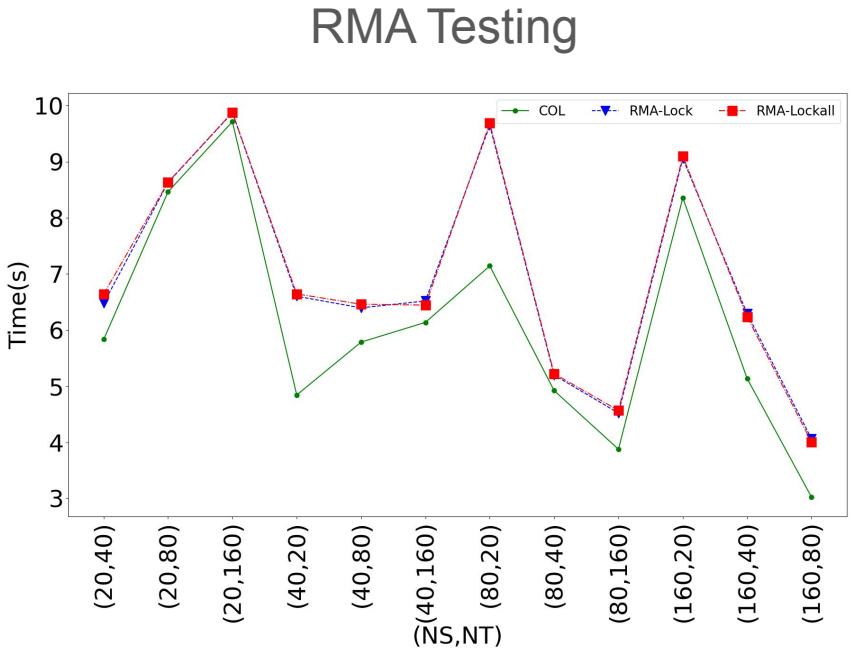
Data redistribution results

- Which combination of methods/strategies perform better?
- Executed application is Conjugate Gradient (CG)



Data redistribution results

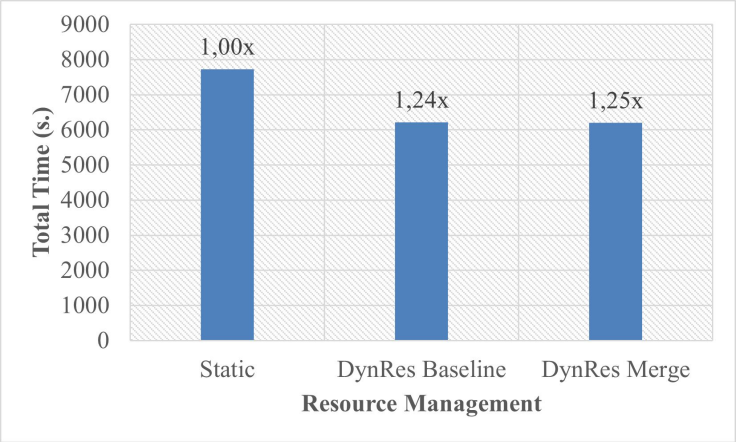
- Which combination of methods/strategies perform better?
- Executed application is Conjugate Gradient (CG)



Workflow results

- What happens when a fully dynamic workflow is executed?
- Executed application is MPDATA3D

Workflow completion times



Resource utilization

	n01	n02	n03	n04	n05	n06	n07	n08
Static	89.19%	89.19%	89.19%	89.19%	87.64%	87.64%	87.64%	87.64%
DynRes Baseline	93.18%	93.10%	93.02%	93.94%	91.88%	91.83%	91.38%	91.56%
DynRes Merge	93.01%	93.33%	94.01%	93.56%	91.38%	91.31%	91.80%	91.81%

Job metrics

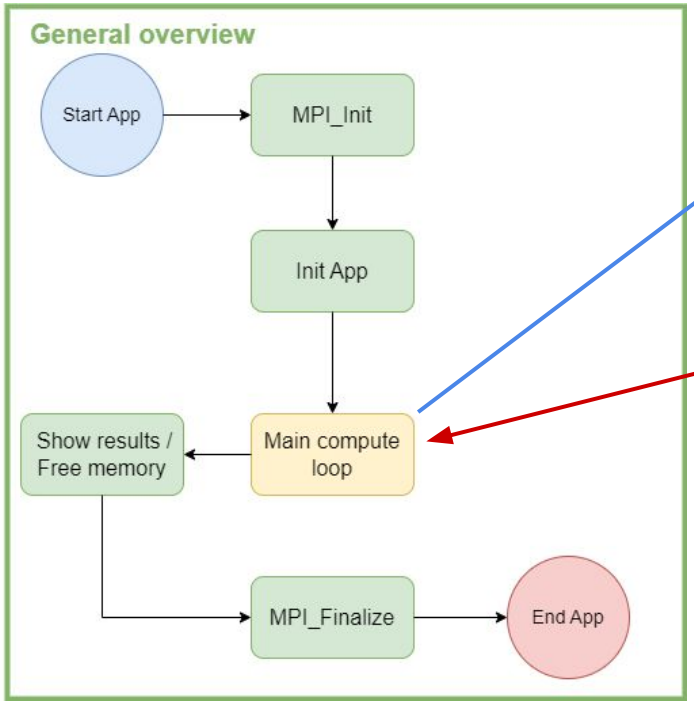
	Execution Mean Time	Waiting Mean Time	Completion Mean Time
Static	14.67 s.	3,356.32 s.	3,370.99 s.
DynRes Baseline	23.52 s.	2,516.51 s.	2,540.03 s.
DynRes Merge	23.51 s.	2,541.13 s.	2,564.64 s.

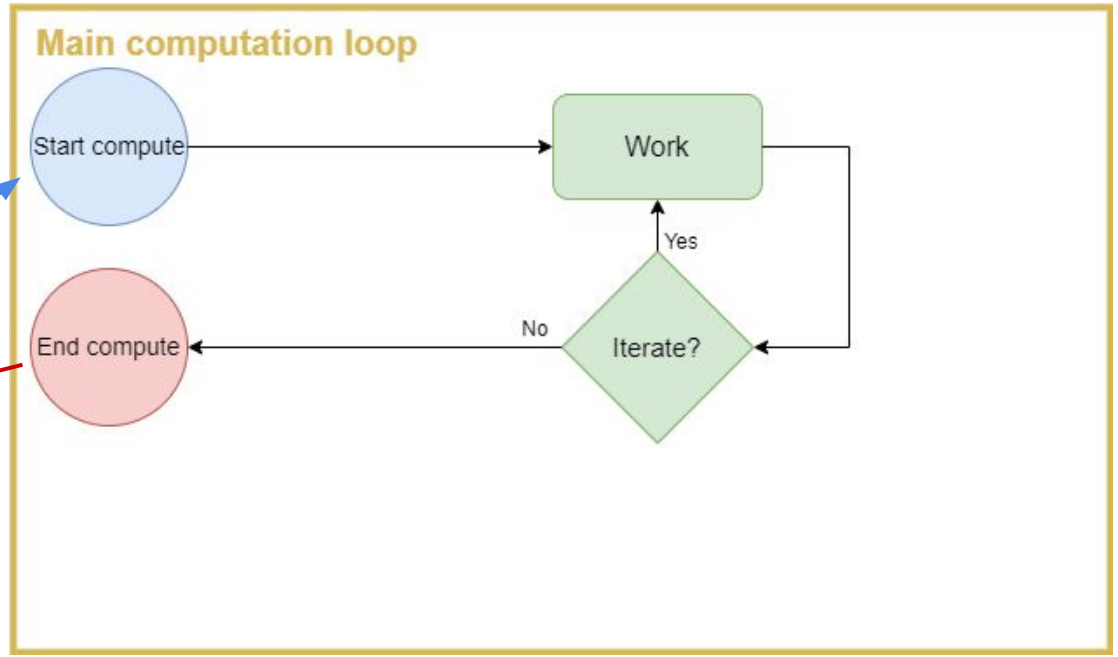
Related article

Outline:

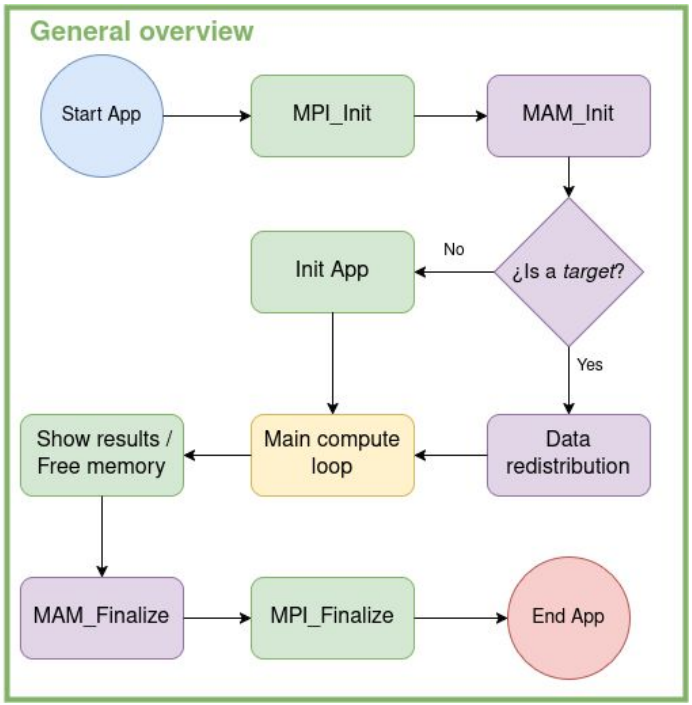
1. Dynamic Resources
2. Proteo
3. Interfaces
   - 3.1 Dynamic implementation flowchart
   - 3.2 Similar implementations
   - 3.3 LibADR
4. Closing remarks

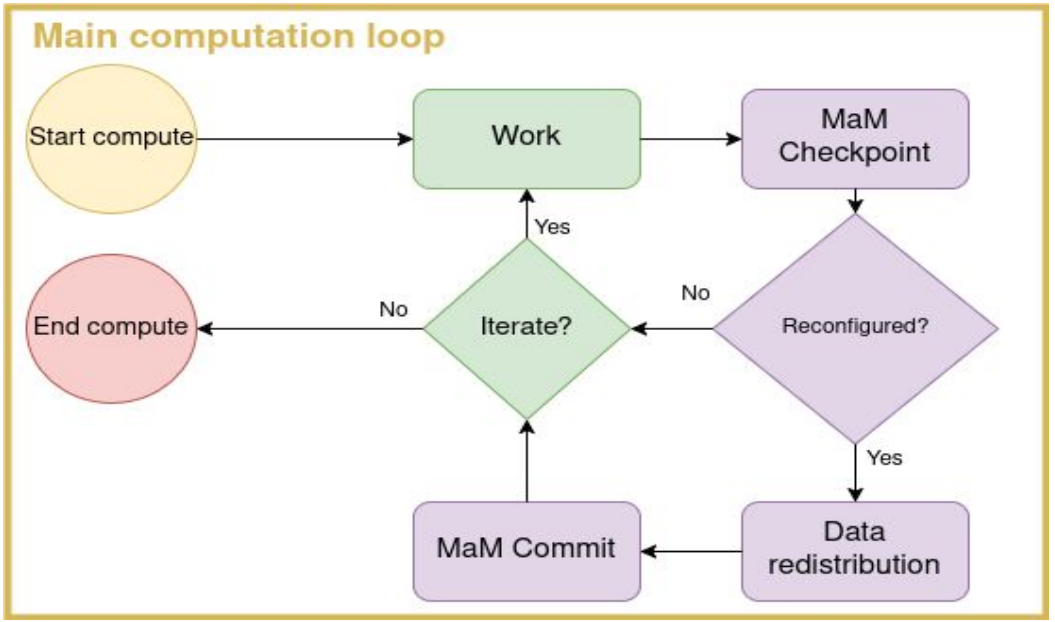
Dynamic implementation flowchart



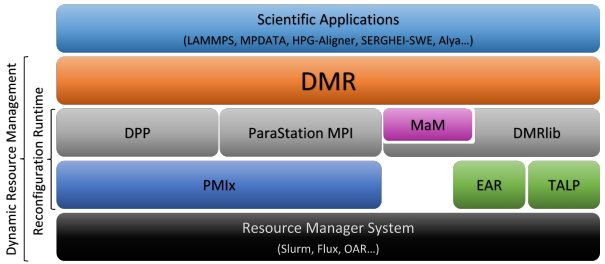


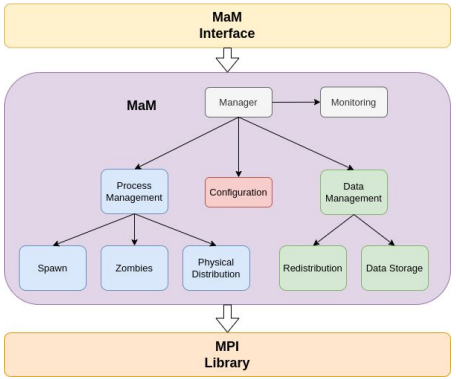
Dynamic implementation flowchart

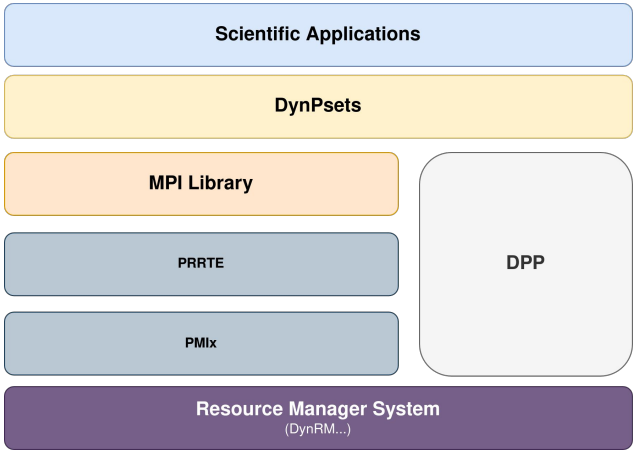




Similar interfaces







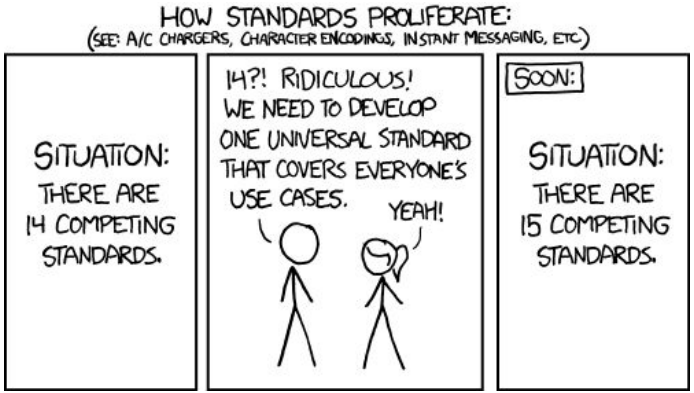

```
DMR_INITIALIZE( initialize(...), redistribution(...) );
DMR_RECONFIGURE( redistribution(...) );
DMR_FINALIZE( finalize(...) );
```

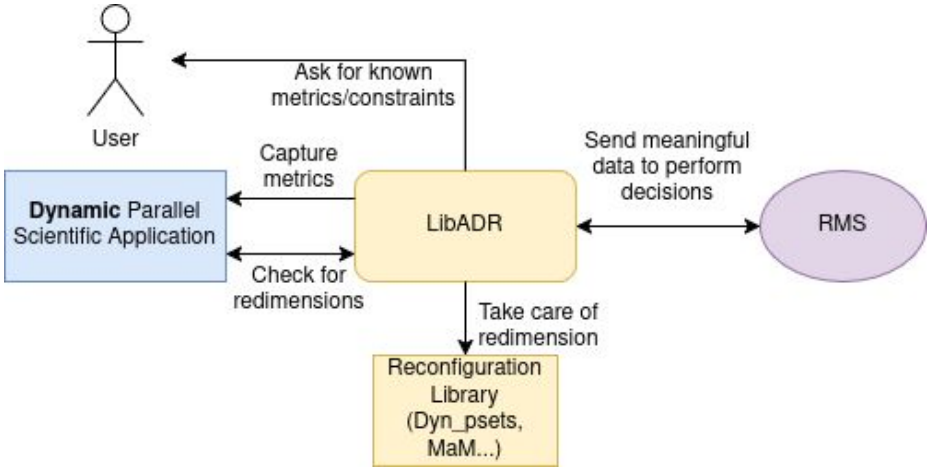
```
dyn_pset_init(user_data, redistribution(...));
dyn_pset_adapt(state);
dyn_pset_finalize();
```

```
MAM_Init(user_function(...), user_args);
MAM_Checkpoint(state, user_function(...), user_args);
MAM_Commit(state);
MAM_Finalize();
```

LibADR

- Awesome Dynamic Resources (ADR)
- Motivation: Reduce complexity for converting applications into dynamic
- Key: Able of capturing performance metrics and convert them into meaningful information for the RMS






```
ADR_init_pset();
ADR_adapt_check(state);
ADR_commit(state);
ADR_finalize_pset();
```

ADR repository

Outline:

1. Dynamic Resources
2. Proteo
3. Interfaces
4. Closing remarks

Dynamic resources can improve HPC systems metrics, but has many challenges to overcome

Proteo can be used to study the impact of dynamic techniques in applications and create dynamic workflows

LibADR as a meaningful interface to reduce complexity and provide information to RMS

Bridging Efficiency and Flexibility: Dynamic Resource Management in HPC

Authors

Iker Martín-Álvarez (martini@uji.es)