Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

# DLB: Dynamic Load Balancing Library

Marta Garcia-Gasulla
Victor Lopez

November 2024

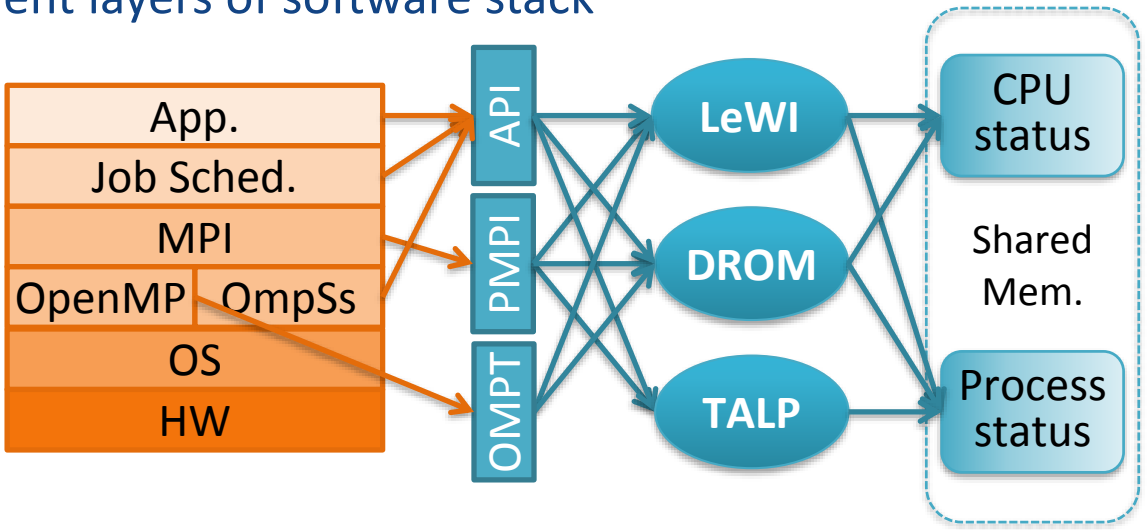
DLB library Tutorial

# DLB structure

- Three modules:
  - LeWI: For fine grain load balancing
  - DROM: For coarse grain resource management
  - TALP: For performance measurement

Three modules, integrated but independent

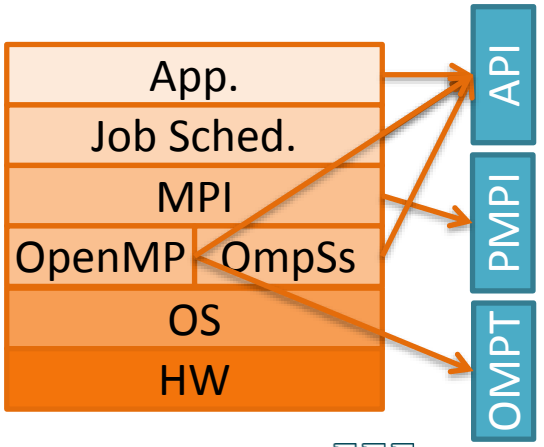
- Common infrastructure
  - Integration with different layers of software stack
  - API
  - Shared memory



# DLB integration

- Integration with different layers:
  - Application: DLB offers a user-level API
  - Job Scheduler: DLB offers an API for resource managers
  - MPI: PMPI interception
  - OpenMP:
    - OMPT interception (few OpenMP runtimes support it)
    - API added by user
    - OpenMP free agent threads **NeW!**
  - Other programming models: DLB offers an API for programming models

DLB offers mechanisms to be transparent to the application or user. DLB API is useful for advanced users that have a good knowledge of their application.



# Why Load Balance?

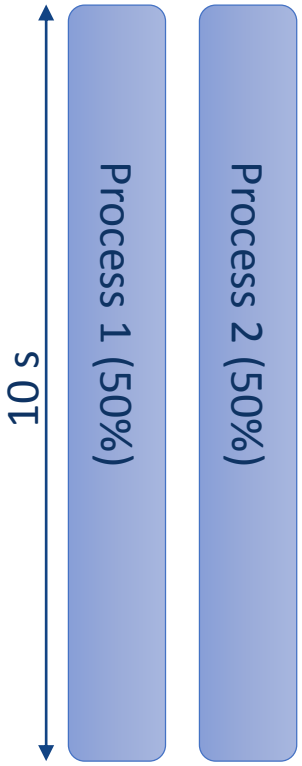

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# What is Load Imbalance

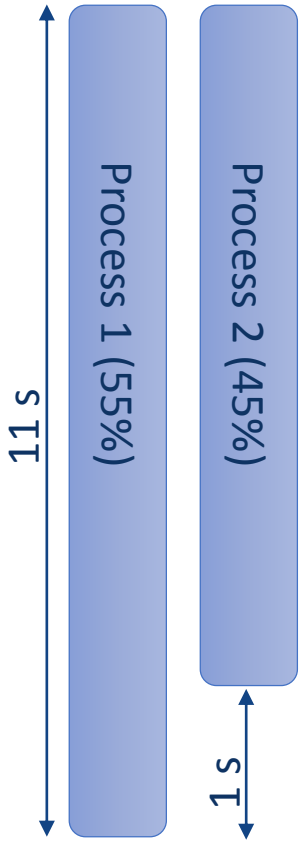
- Irregular distribution of load among resources.
  - Resources can be: computational, network, processing units…
- Our target: MPI load Imbalance
  - MPI is the standard de facto in HPC applications
  - MPI processes do not share data
    - Moving data around is expensive

# Load Imbalance: Magnitude of the tragedy

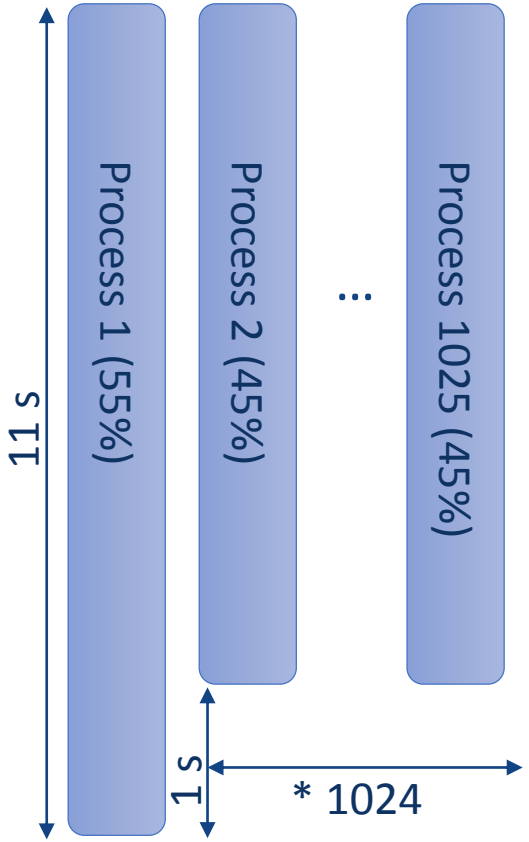
- Ideal




- 5% difference




- 5% difference in 1024 processes

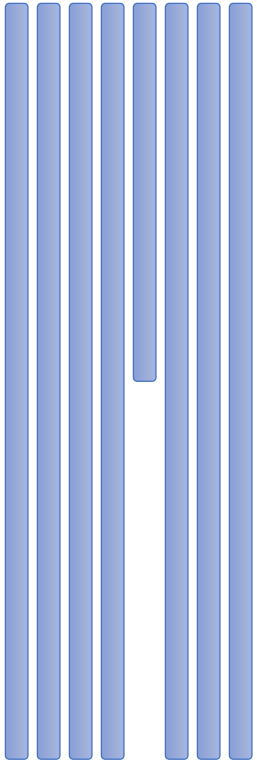



1s * 1024 CPUS = 1024 s = 17 minutes of CPU
17m * 10.000 time steps = **2.844 CPU hours**

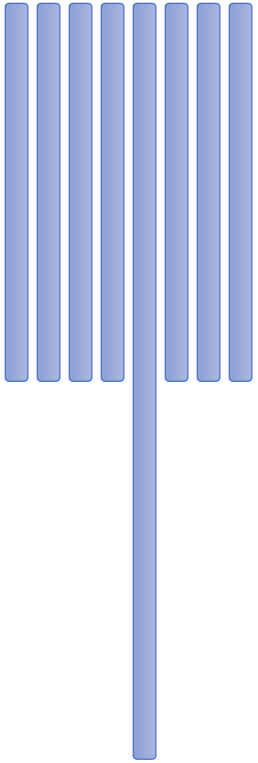
# Load Imbalance: Measuring it

- Which application is more unbalanced? 🤔

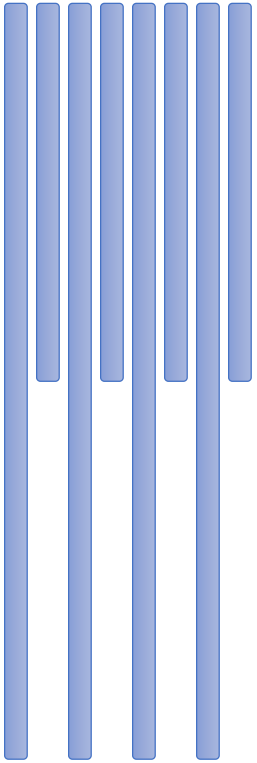
- A)



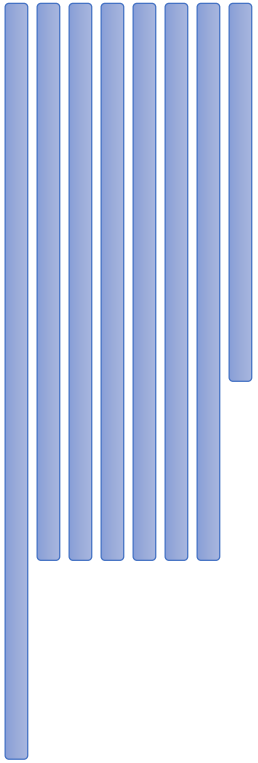
- B)



- C)



- D)



# Load Imbalance: Measuring it

➢ Our focus is to make the most efficient use of computational resources

$$Load\ Balance = \frac{Useful\ CPU\ time}{Total\ used\ CPU\ time} =$$

$$= \frac{\sum_{n=1}^{numProcs}(t_n)}{Max_{n=1}^{numProcs}(t_n) * numProcs} = \frac{Average_{n=1}^{numProcs}(t_n)}{Max_{n=1}^{numProcs}(t_n)}$$

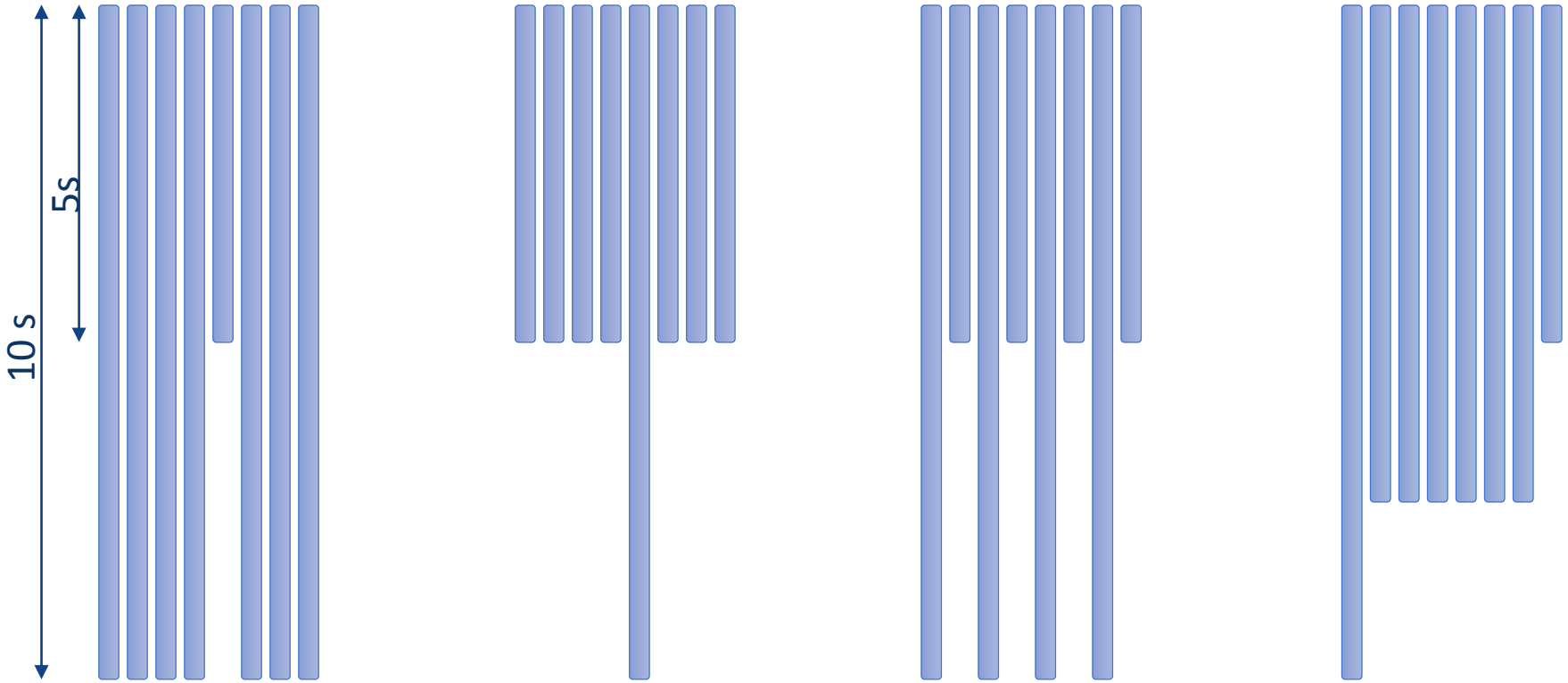
- $numProcs$ = number of MPI processes
- $t_n$ = execution time of process n
- $0 < LB < 1$
- $LB = 1$ → (Perfect Load Balance)


$$LB = \frac{Useful = 3 + 1{,}5 = \mathbf{4,5}}{UsedCPU = 3 * 2 = \mathbf{6}} = 0{,}75$$

# Load Imbalance: Measuring it

$$LB = \frac{useful\ CPU}{used\ CPU}$$

- Which application is more unbalanced?
  - A)
  - B)
  - C)
  - D)



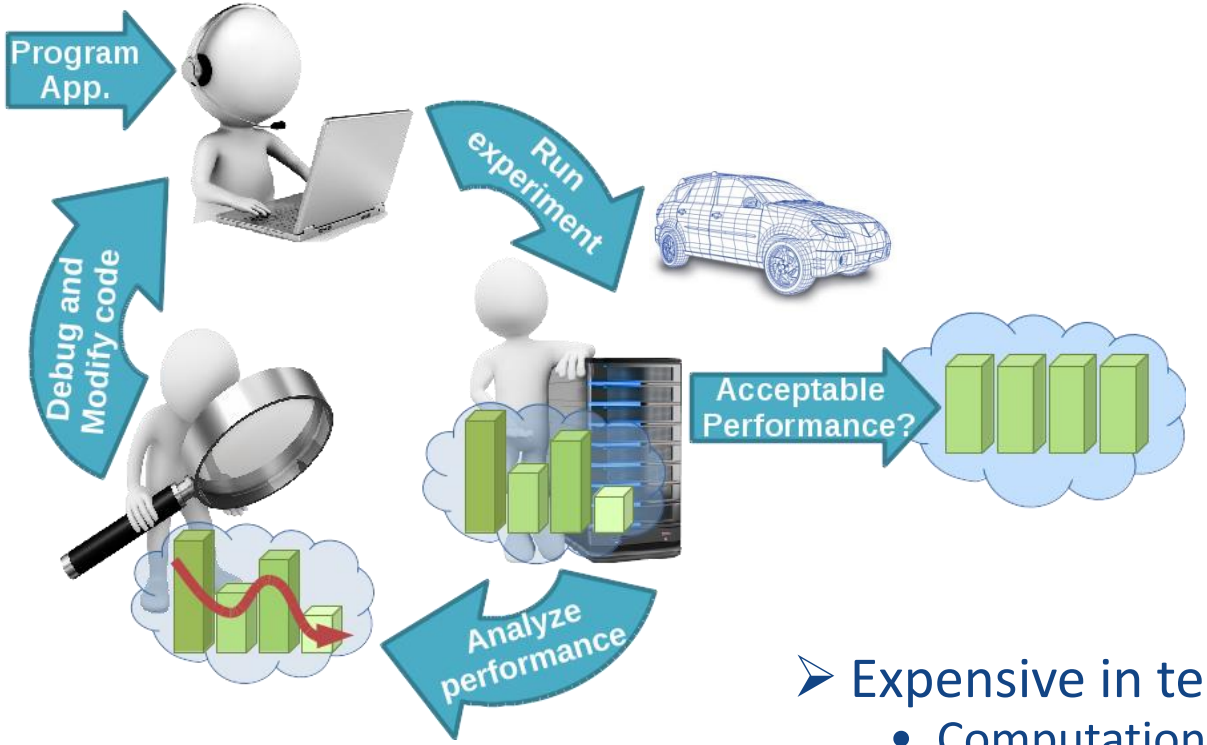

$$\frac{(7 * 10) + 5}{10 * 8} = 0{,}9375$$

$$\frac{(7 * 5) + 10}{10 * 8} = 0{,}5625$$

$$\frac{(4 * 10) + (4 * 5)}{10 * 8} = 0{,}75$$

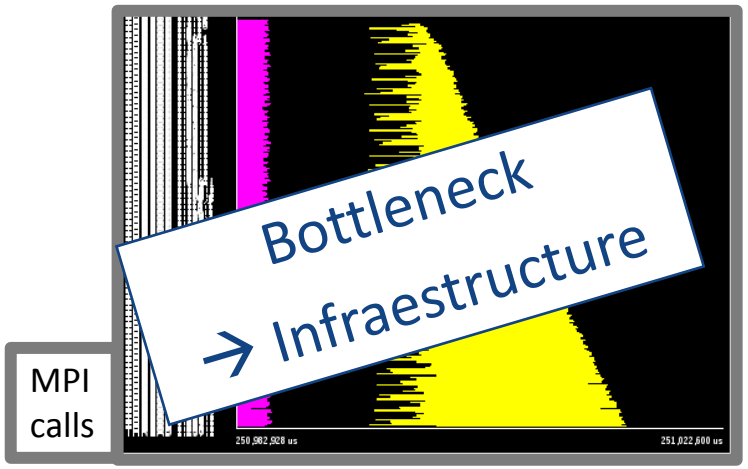
$$\frac{10 + 5 + (6 * 7{,}5)}{10 * 8} = 0{,}75$$

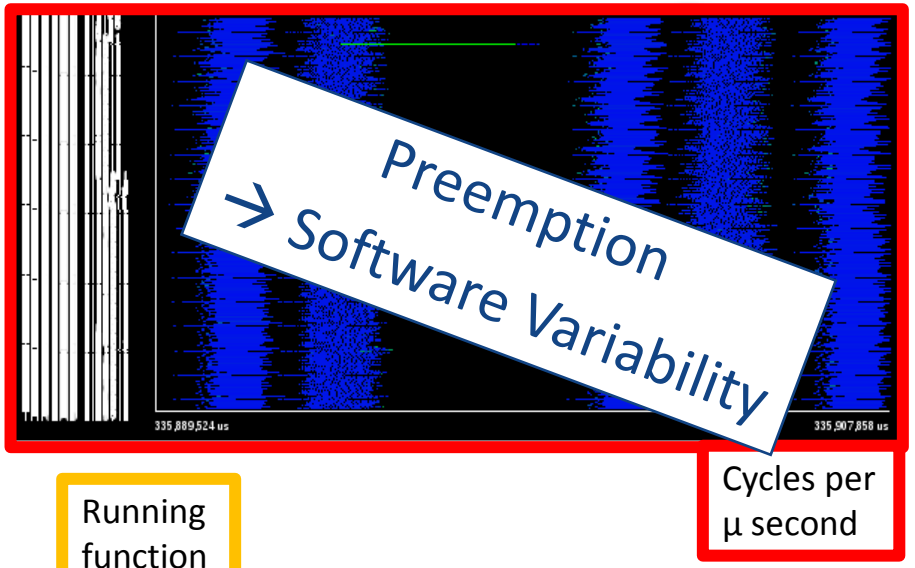
# Load Imbalance: Solution from developers?

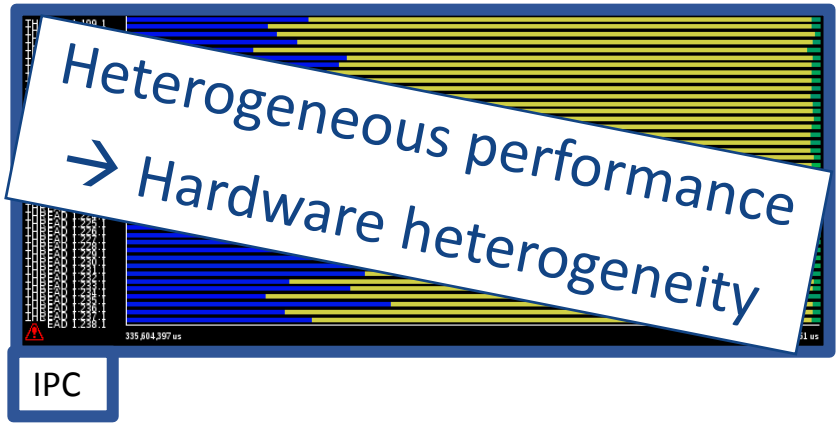


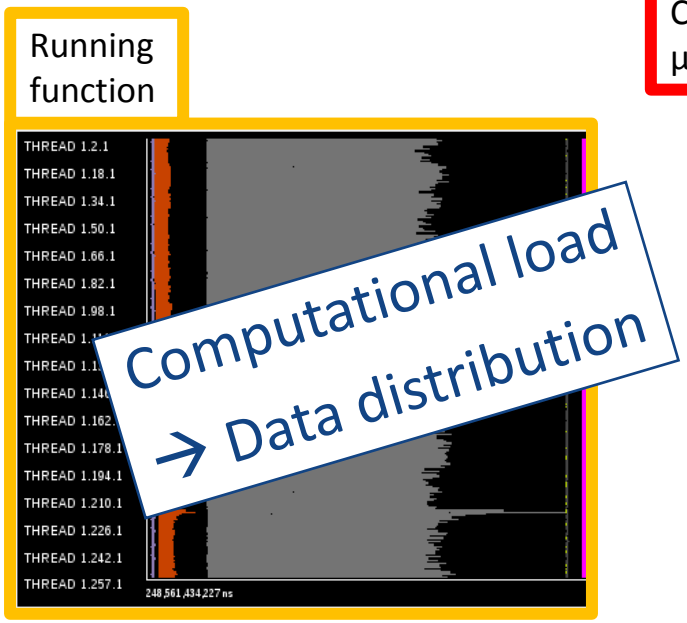

- Expensive in terms of:
  - Computational resources
  - Personal resources
- What happens if we change the input?
- And the hardware?
- Is it a real solution?

# Load Imbalance: Where?



# Load Imbalance: Still searching for a solution…

- Different sources… different solutions
  - Data distribution
    - Redistribute → New Input, redistribute again?
  - Hardware heterogeneity
    - Tune specifically for architecture → New machine, tune again?
  - Infrastructure
    - Adapt code to infrastructure → New software or hardware, adapt again?
  - Software/Hardware variability
    - ???

- Our Solution: React when imbalance is happening
  - We can not fight it, lets adapt!
  - One solution to ~~rule~~ solve them all

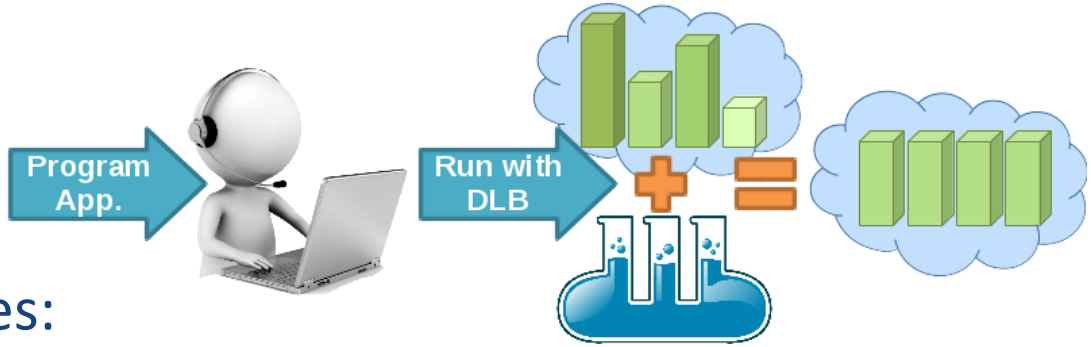
Be water, my friend !!!

Bruce Lee

# LeWI
# Lend When Idle


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

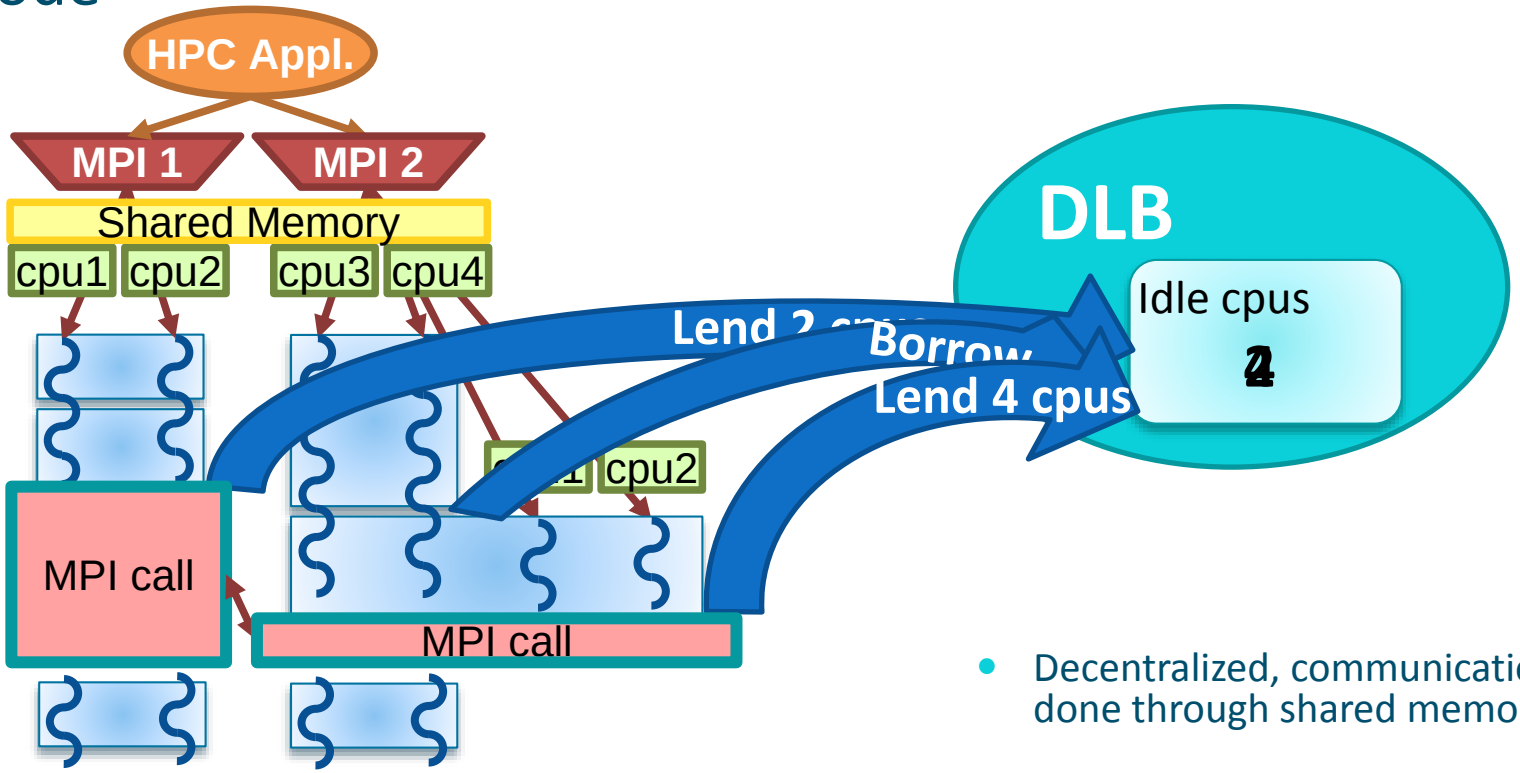
# Lend When Idle (LeWI)




- Our objectives:
  - Address all sources of imbalance
    - Fine Grain, dynamic…
    - How?
      - Detect imbalance at runtime
      - React immediately
  - Real product for HPC
    - Use common programming model/environment
      - MPI + OpenMP
  - Transparent to the application
    - Runtime library

# Lend When Idle (LeWI)

- **The idea:** Use computational resources of a process when not using them to speed up another process in the same node




- Decentralized, communication is done through shared memory

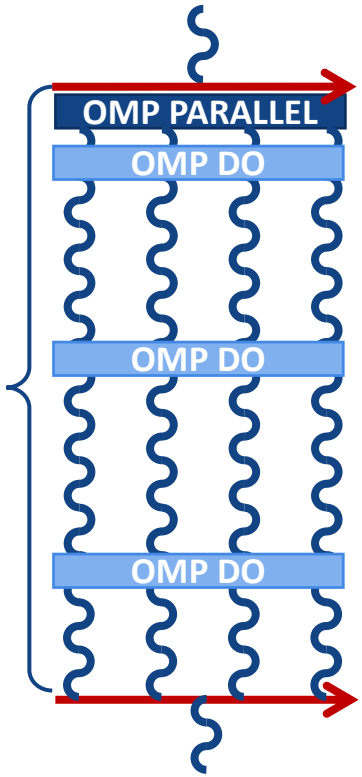
**LeWI a runtime balancing algorithm for nested parallelism.**
*In Proceedings of International Conference of Parallel Processing (ICPP 2009).*

# LeWI: Implementation

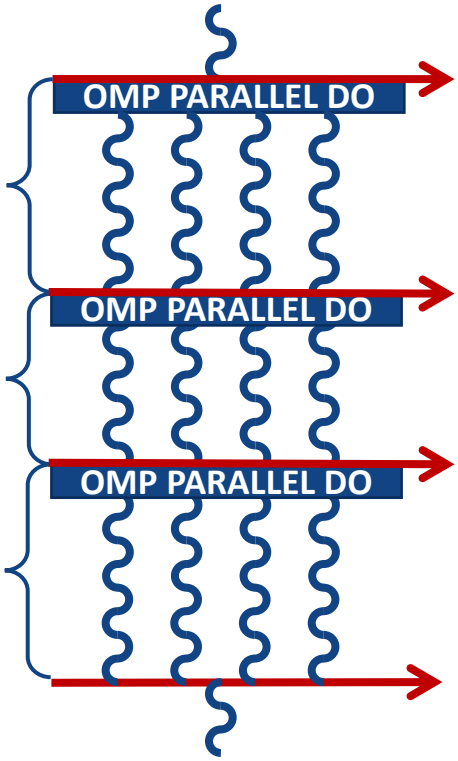
- **MPI interception:** Use standard PMPI interception avoids recompiling, DLB can be used with LD_PRELOAD.
- **Second level of parallelism**: We need shared memory parallelism. Current version supports OmpSs and OpenMP.
  - Must be malleable. Lack of malleability limits performance
    - Malleability can be limited by the programming model or the application.
  - **OpenMP**: Three levels/options of integration:
    - **API**: Works with any OpenMP implementation, must modify the code and link with DLB.
    - **OMPT**: Only works with OpenMP implementations implementing OMPT.
    - **Free agents**: Works preloading our own version of OpenMP implementation (LLVM based)
  - **OmpSs:** Fully integrated with DLB, high malleability.

# LeWI: Malleability vs. Programming Model

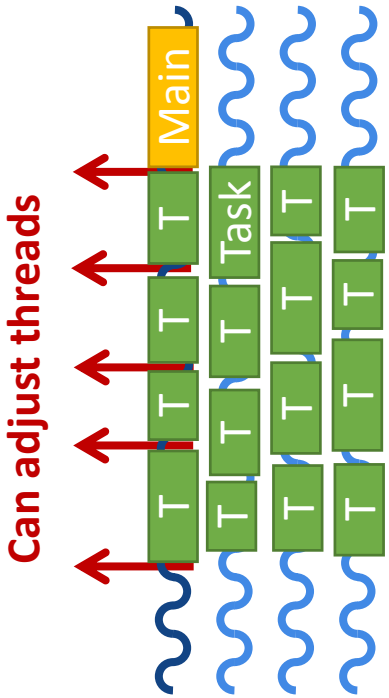
- OpenMP
  - Single parallel region



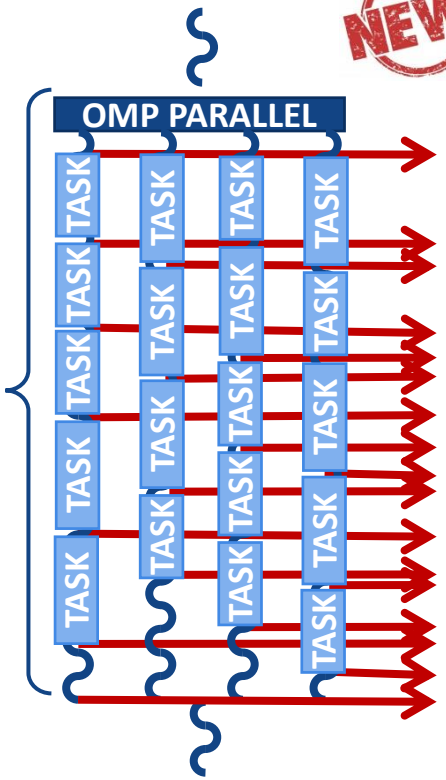

- OpenMP
  - Multiple parallel regions




- OmpSs




- OpenMP
  - Free agent threads

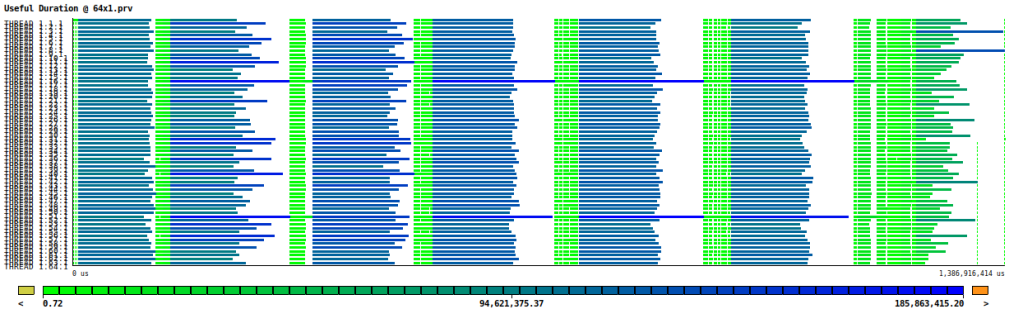



→ Number of threads can be changed

# OpenMP/OmpSs Summary

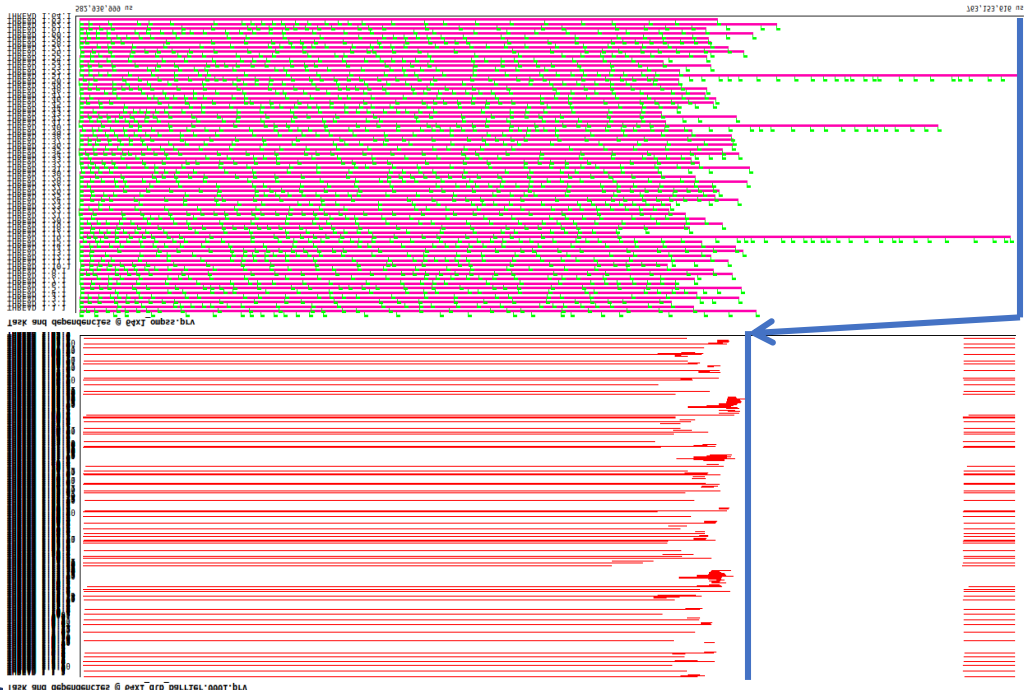
| | OpenMP | OpenMP + OMPT | OpenMP + free agent | OmpSs |
|---|---|---|---|---|
| **Requirements** | None | Intel OpenMP 2018 / LLVM 8.0 or greater | Custom LLVM OpenMP runtime | |
| **CPU binding** | No supported | Rebind through OMPT | Yes | Yes |
| **Malleability** | Only outside parallel regions | Only outside parallel regions | Malleability through tasking | Yes |
| **Integration** | Add DLB_borrow before each parallel region + LD_PRELOAD | LD_PRELOAD | LD_PRELOAD | Transparent, integrated through OmpSs |

# Success Story 1: ParMMG



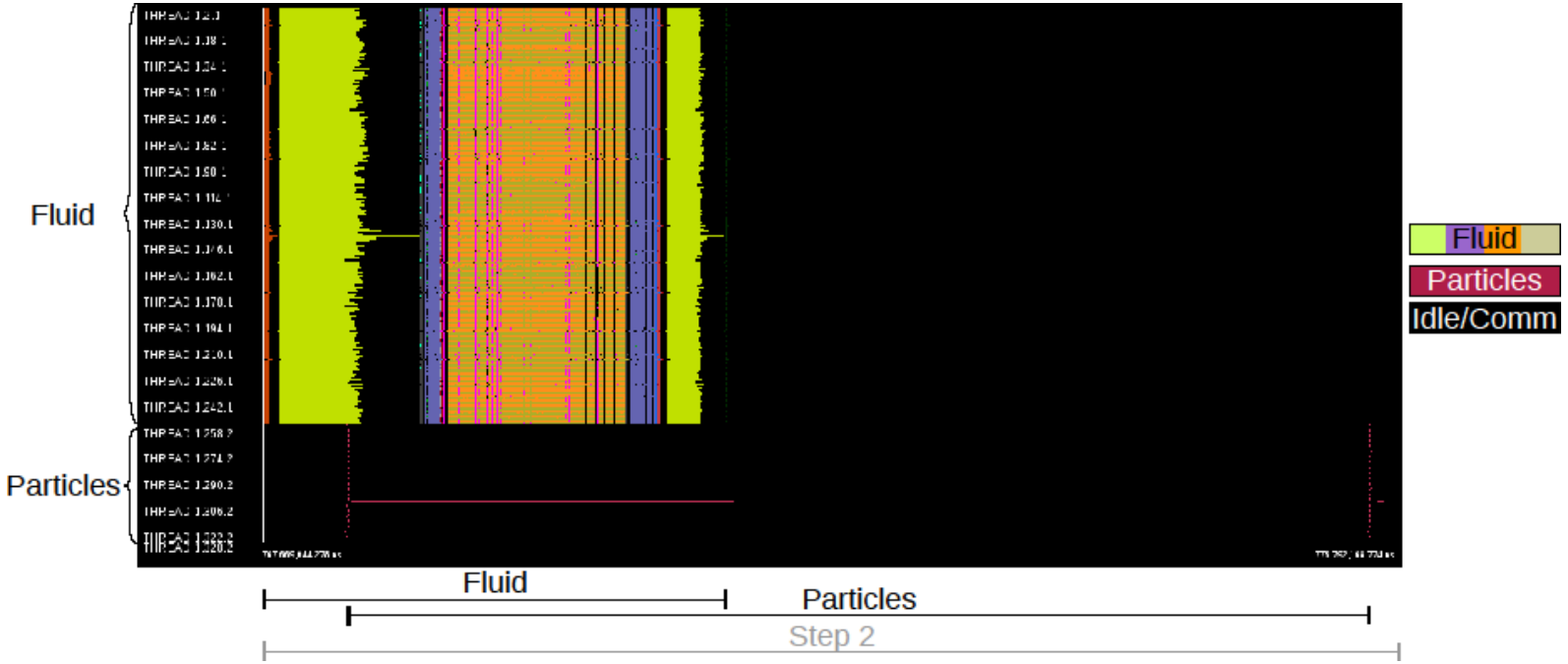

- Parallel mesh adaptation of 3D volume meshes. High imbalance and changing between iterations. Pure MPI code

- Added OmpSs parallelization to one loop + 1 call to DLB API.



- 1.2x Speedup overall execution.

# Success Story 2: Alya coupled codes

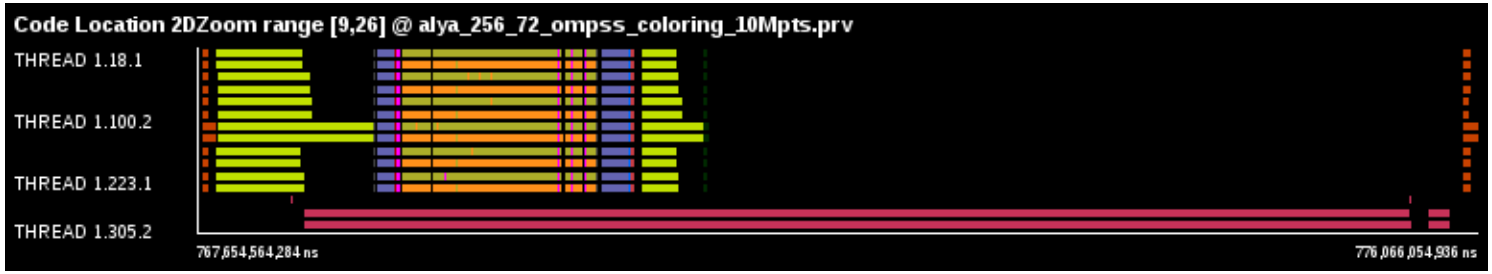



- Respiratory simulation coupling 2 codes:
  - Fluid + particle tracking
  - 256 MPI ranks → Fluid
  - 72 MPI ranks → Particles

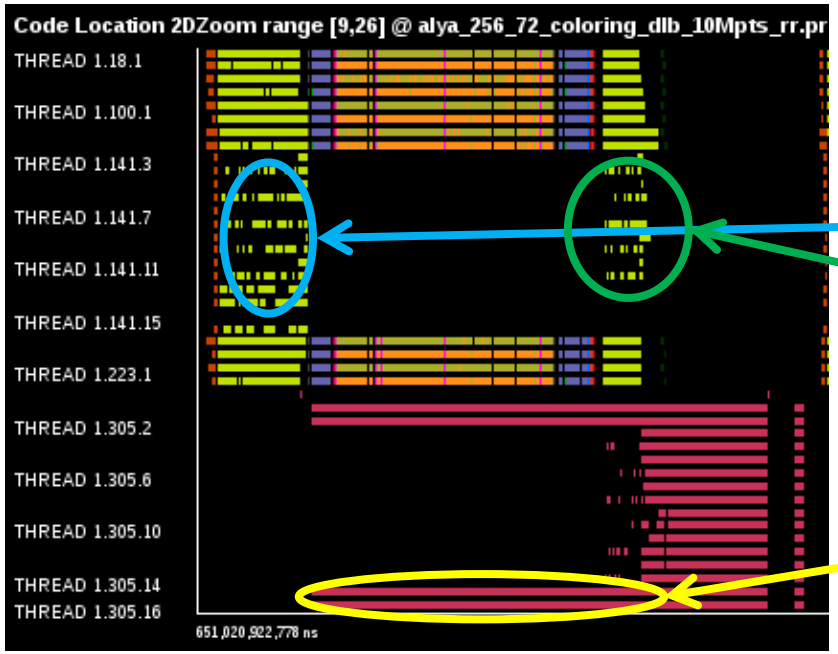
- Code modifications:
  - Parallelized with OpenMP matrix assembly of fluid
  - Parallelized with OpenMP particle transport

# Success Story 2: Alya coupled codes

Original




With DLB




- Zoom in 1 node
- 3 Actions:
  - Load Balance Fluid
  - Load Balance Particles
  - Load Balance 2 codes

# LeWI API

Targeted for code developers

- int DLB_Enable(void);
  - Enable DLB and all its features in case it was previously disabled otherwise it has no effect
- int DLB_Disable(void);
  - Disable DLB actions for the calling process.
- int DLB_SetMaxParallelism(int max);
  - Set the maximum number of resources to be used by the calling process.
- int DLB_UnsetMaxParallelism(void);
  - Unset the maximum number of resources to be used by the calling process.
- int DLB_Barrier(void);
  - Barrier between processes in the node

Targeted for runtime or programming model

- int DLB_Lend(…); ★
  - Lend current CPUs
- Int DLB_Reclaim(…); ★
  - Reclaim CPUs owned by the process
- int DLB_AcquireCpu(…); ★
  - Acquire a specific CPU, equivalent to Reclaim if the process it the owner and to Borrow if not.
- int DLB_Borrow(…); ★
  - Borrow possible CPUs registered on DLB
- int DLB_Return(…); ★
  - Return claimed CPUs of other processes

# DROM
## Dynamic Resource Ownership Management


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

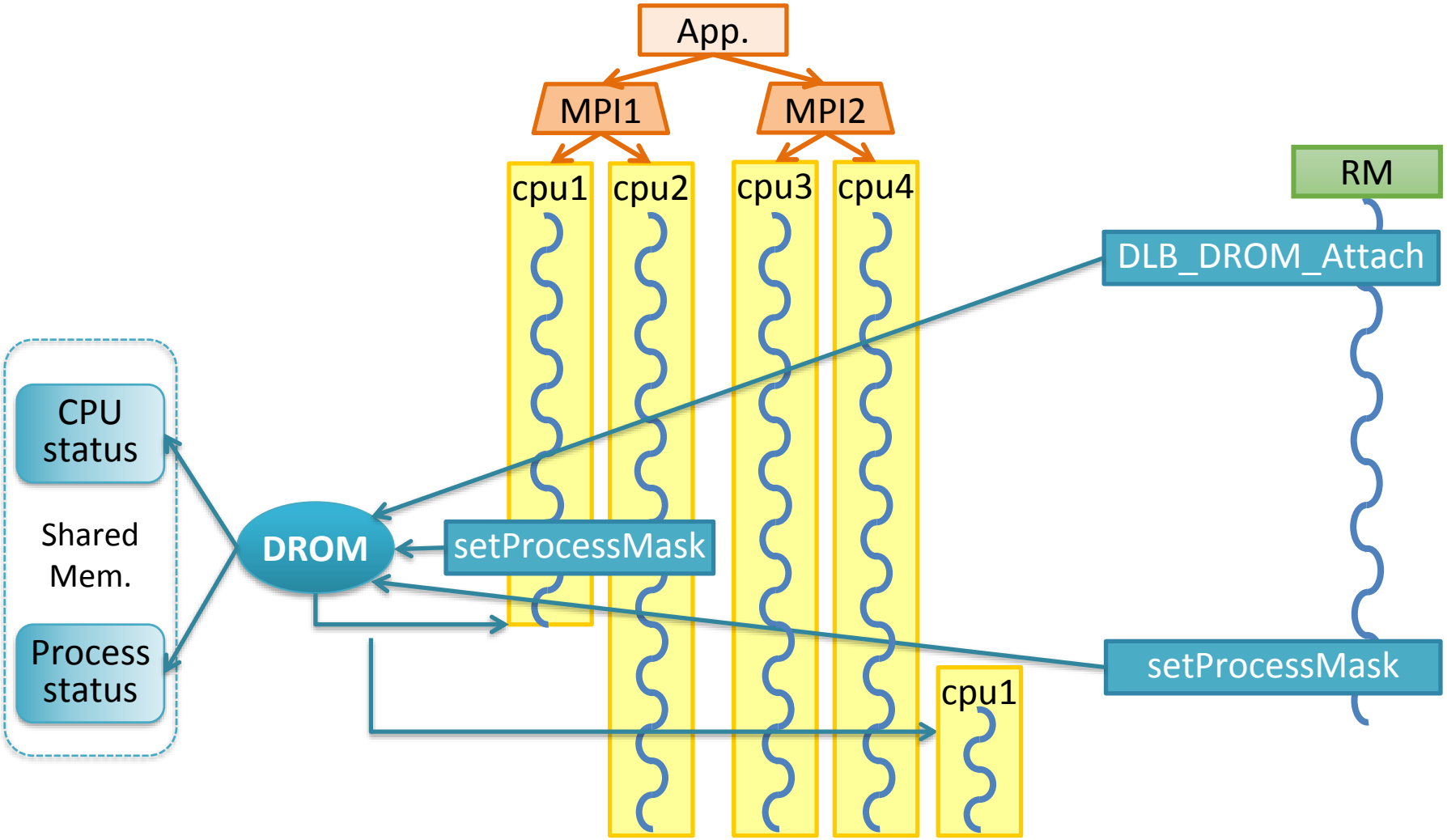
# DROM: Dynamic Resource Ownership Management

- API for superior entity
  - Job Scheduler
  - Resource manager
  - User

- Allow to change the **owner** of resources (CPUs)
  - By the process
  - By an external entity (Resource manager)

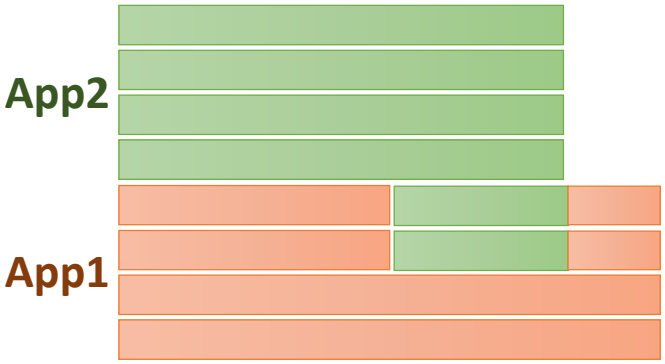
**DROM: Enabling Efficient and Effortless Malleability for Resource Managers.**
*In Proceedings of International Conference on Parallel Processing (ICPP 2018)*

# DROM: Dynamic Resource Ownership Management

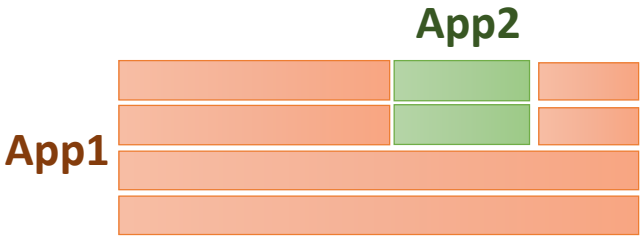


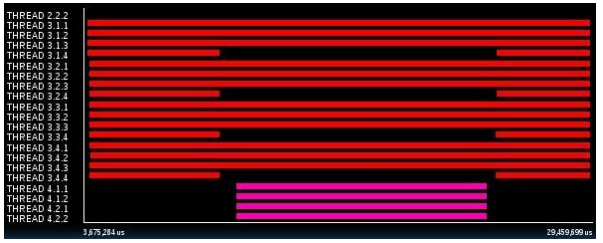
# DROM: Use cases

- A) User:
  Increase priority to App2

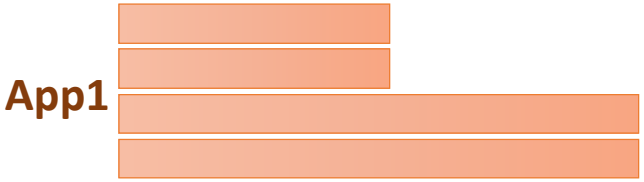



- B) Job Scheduler:
  Run High priority App2 in resources assigned to App1

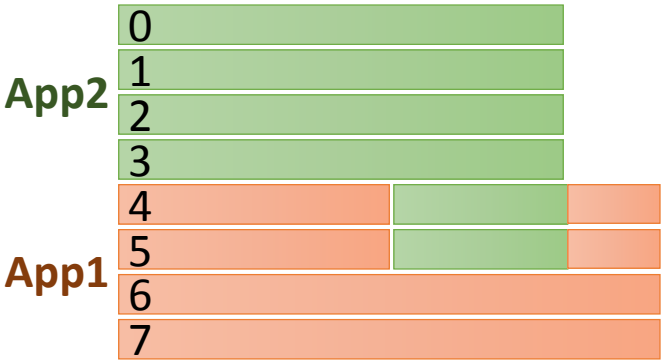


- C) App1: Release 2 CPUs because not using efficiently

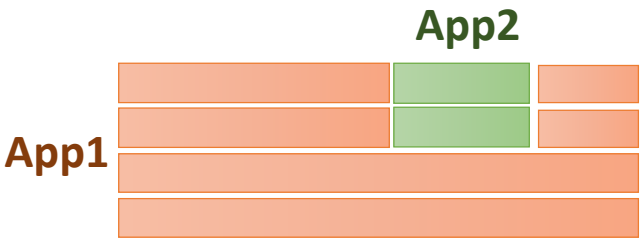


# DROM: How to




➢ A)

```
$> dlb_taskset -p pid_app2 -c 0-5
```




➢ B)

```
$> dlb_taskset -c 0,1 ./App2
```


➢ C)

```
DLB_DROM_SetProcessMask(my_pid, [0,0,1,1]);
```

# DROM API

- int DLB_DROM_Attach(void);
  - Attach current process to DLB system as DROM administrator
- int DLB_DROM_Detach(void);
  - Detach current process from DLB system
- int DLB_DROM_GetNumCpus(int *ncpus);
  - Get the number of CPUs in the node
- int DLB_DROM_GetPidList(int *pidlist, int *nelems, int max_len);
  - Get the list of running processes registered in the DLB system
- int DLB_DROM_GetProcessMask(int pid, dlb_cpu_set_t mask, dlb_drom_flags_t flags);
  - Get the process mask of the given PID
- int DLB_DROM_SetProcessMask(int pid, const_dlb_cpu_set_t mask, dlb_drom_flags_t flags);
  - Set the process mask of the given PID
- int DLB_DROM_PostFinalize(int pid, dlb_drom_flags_t flags);
  - Unregister a process from the DLB system

# TALP
# Tracking Application Live Performance


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# TALP: Tracking Application Live Performance

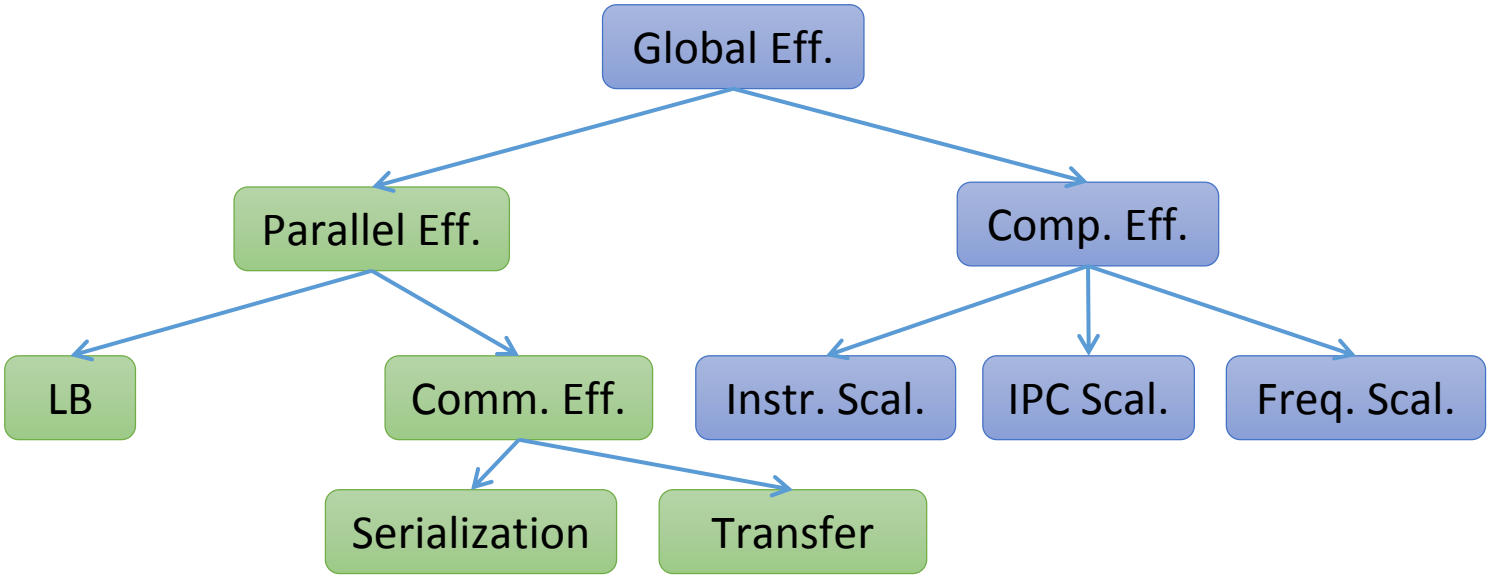
- Profiling tool with:
  - Low overhead
  - Report POP metrics
  - API to obtain metrics at runtime
  - API to instrument code and profile regions of code

- Current version profiles MPI performance

**TALP a lightweight tool to Unveil Parallel Efficiency of Large Scale Executions**. *In Proceedings of Performance Engineering, Modelling, Analysis, and Visualization Strategy (Permavost 2021).*

# TALP metrics vs. POP metrics

- POP metrics

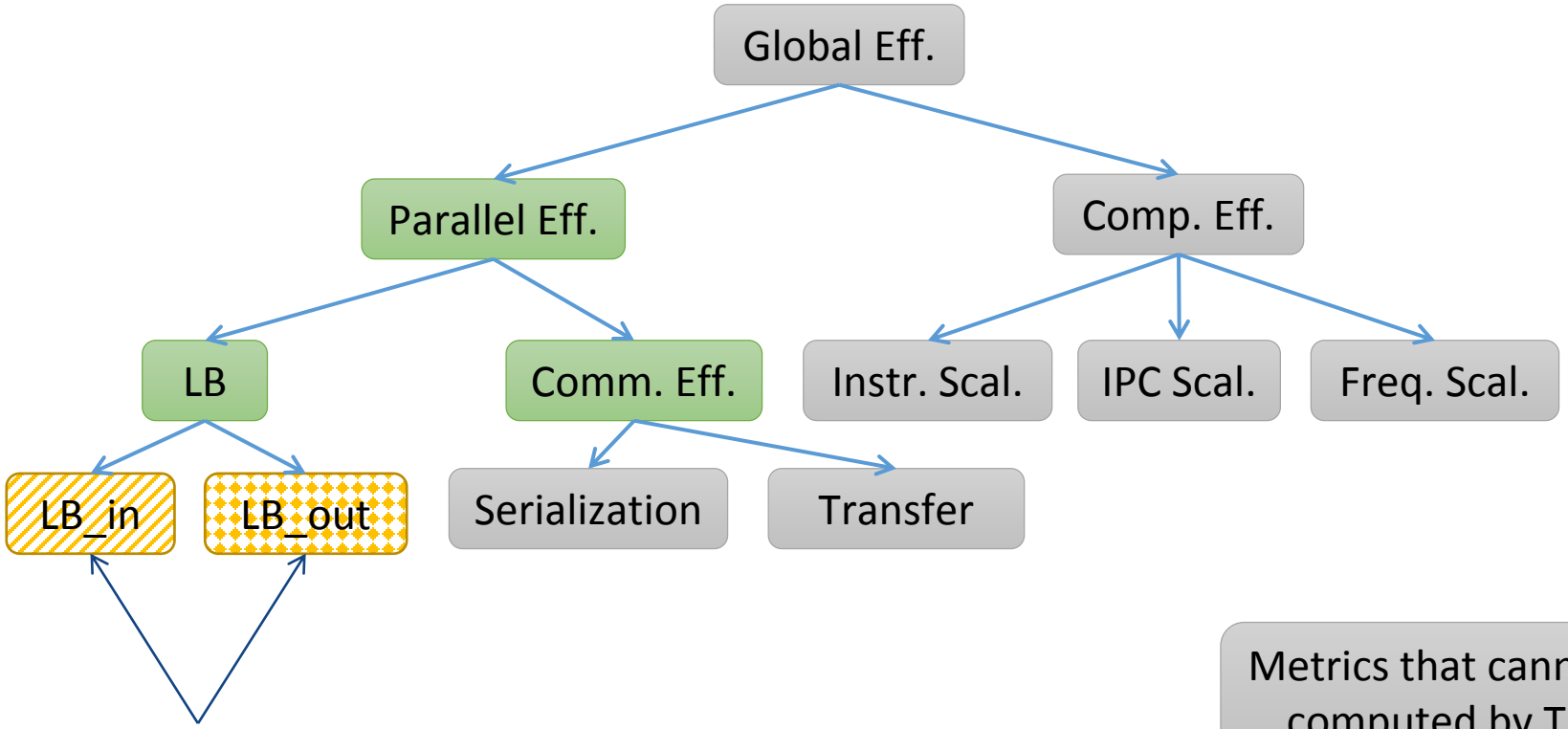



Scalability metrics computed based on a reference run

Efficiency metrics

# TALP metrics vs. POP metrics

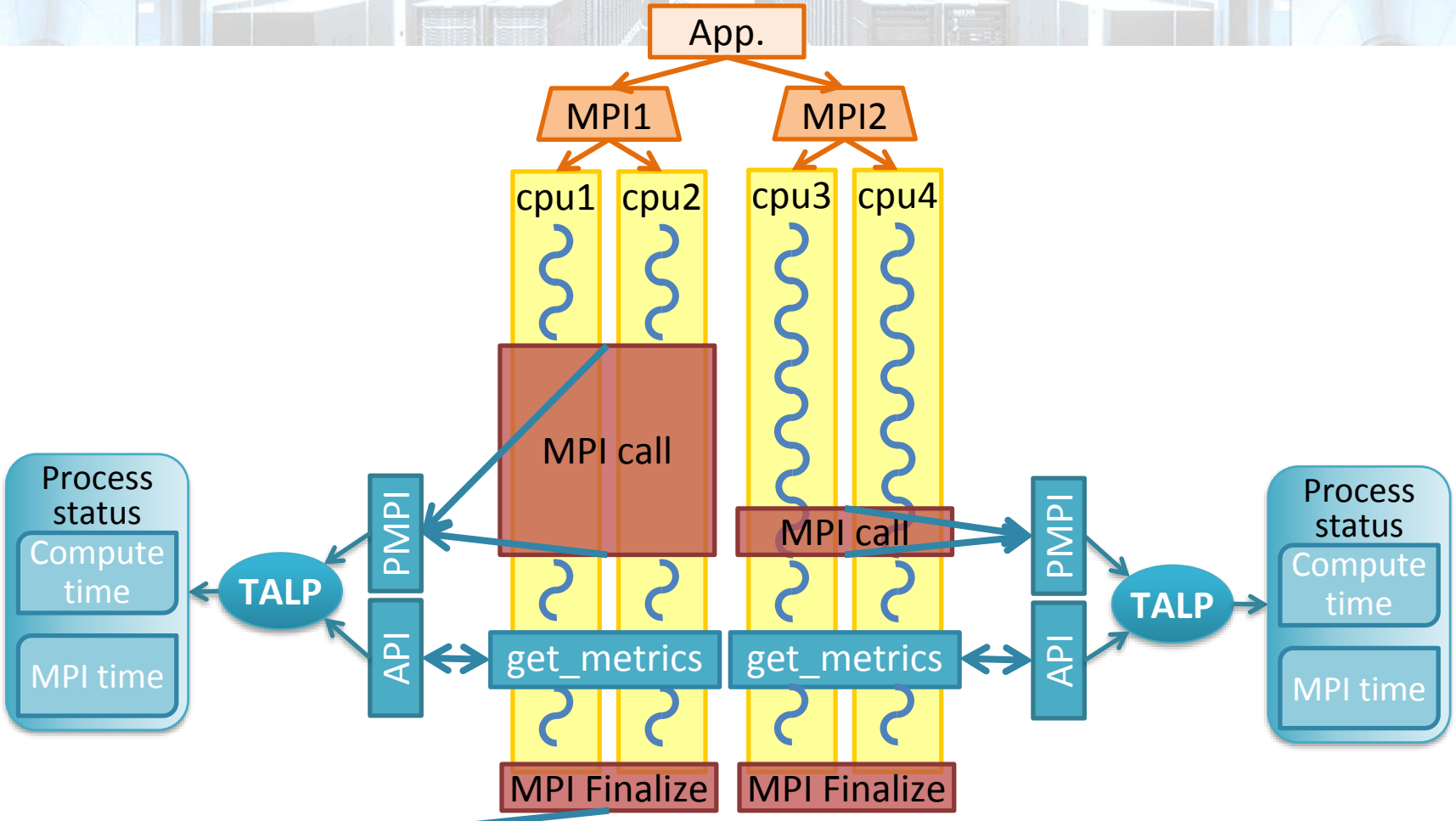
- TALP metrics




Metrics that cannot be computed by TALP

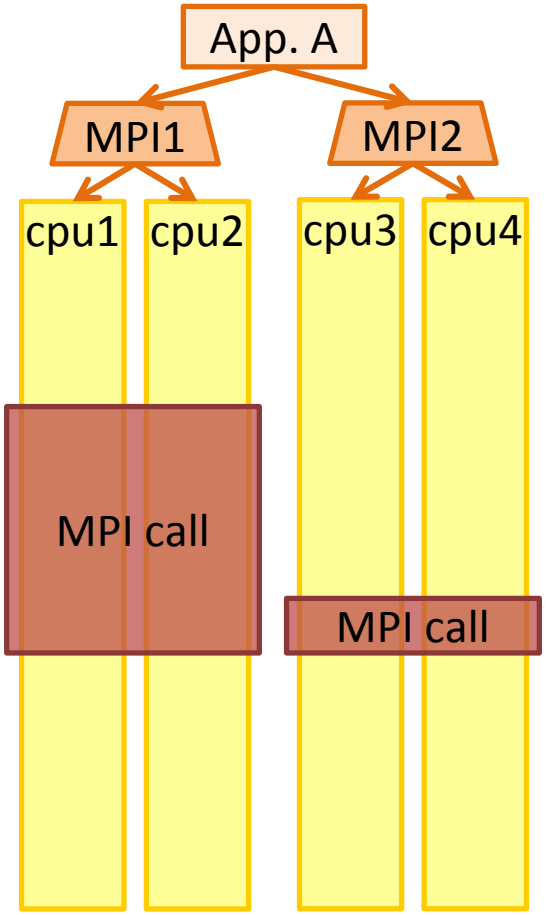
Efficiency metrics computed by TALP

# TALP

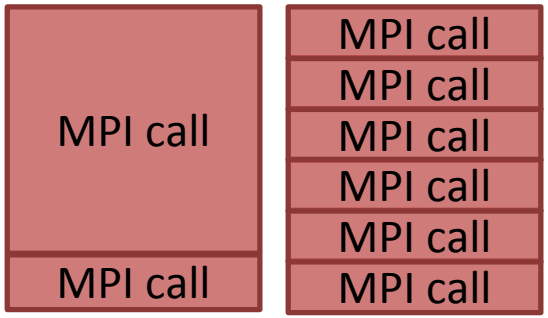



```
######### Monitoring Region App Summary #########
### Name:                       MPI Execution
### Elapsed Time :              2.66 s
### Parallel efficiency :       0.70
###    - Communication eff. :   0.98
###    - Load Balance :         0.72
###        - LB_in :            0.72
###        - LB_out:            1.00
```

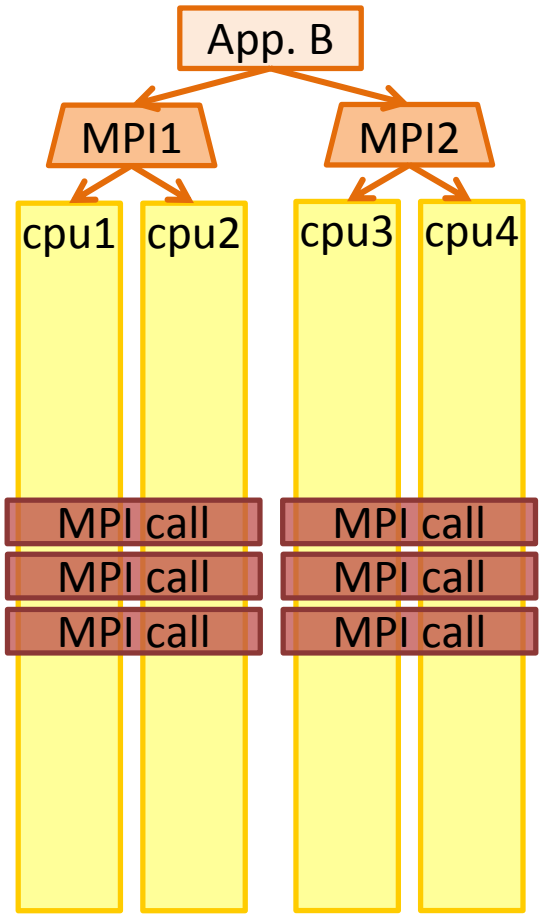
# Why is more than “yet another profiling tool”?




- A profiler will report same “issue” while both cases have very different problems.




- TALP will report a low Load Balance for App A and a low Communication efficiency for App B



# Using TALP

## At finalization

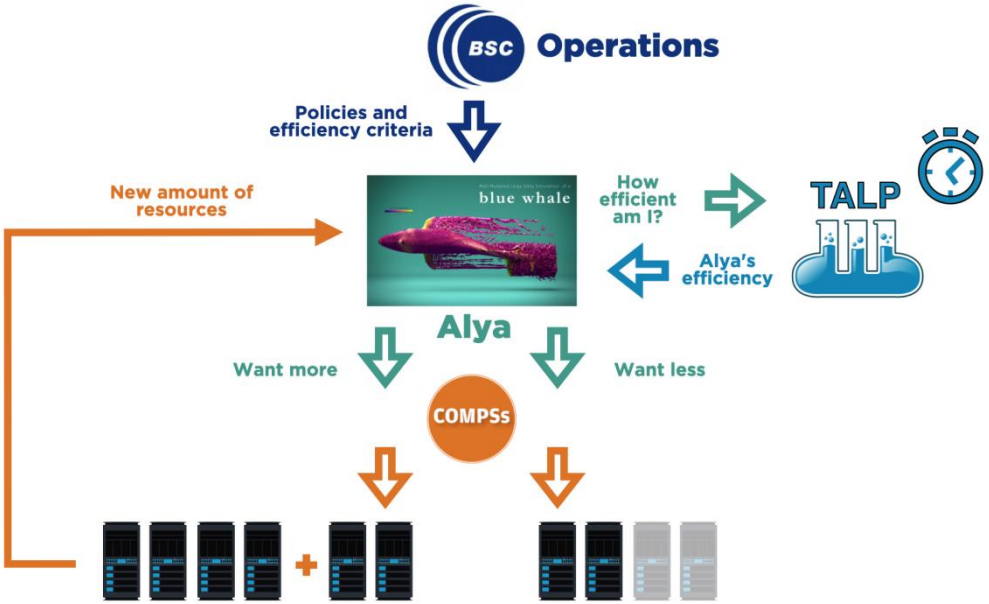
```
DLB_ARGS=" --talp [--talp-summary=pop-metrics]"
env LD_PRELOAD="$DLB_LIBS/libdlb_mpi.so" ./app
```

```
######### Monitoring Region App Summary #########
### Name:                        MPI Execution
### Elapsed Time :               2.66 s
### Parallel efficiency :        0.70
###    - Communication eff. :    0.98
###    - Load Balance :          0.72
###       - LB_in :              0.72
###       - LB_out:              1.00
```

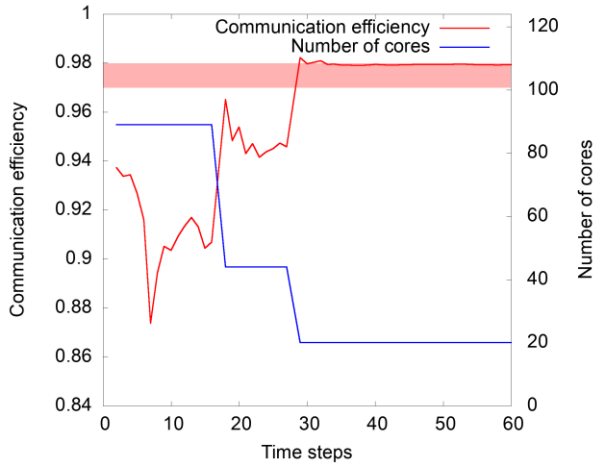
## At runtime

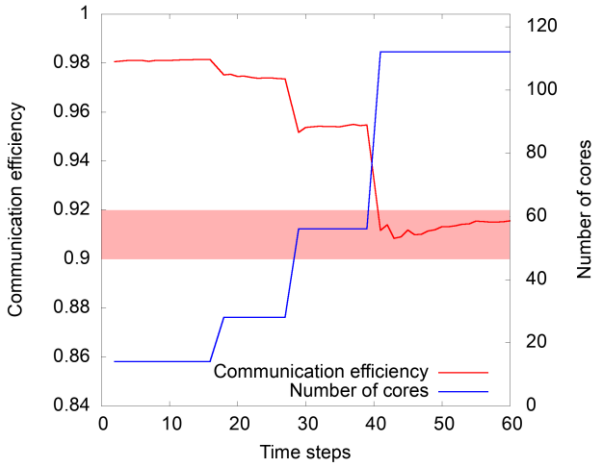
```
# include < dlb_talp .h >
...
// Register a new region or obtain an existing handler
dlb_monitor_t * monitor = DLB_MonitoringRegionRegister
("Name");
// Start TALP monitoring region
DLB_MonitoringRegionStart( monitor );
...
// Stop TALP monitoring region
DLB_MonitoringRegionStop( monitor );
...
// Print a report by standard output
DLB_MonitoringRegionReport( monitor );
...
// Manually obtain some metrics from the monitor
int64_t elapsed = monitor->elapsed_time;
int64_t elapsed_use =monitor->elapsed_computation_time;
float comm_eff = ( float ) elapsed_use / elapsed ;
```

# Success story 3: Malleable simulation




- Application that adjust number of resources used based on measures by TALP



# Work in progress and future work

- Work in progress:
  - Free-agent implementation in OpenMP LLVM runtime
  - Integration with OmpSs@Cluster
  - Integration with PyCOMPSs
- Future work:
  - Extend TALP
    - To report OpenMP metrics
    - To report HWC
    - To report accelerator metrics (GPUs)
  - Extend LLVM OpenMP runtime
    - Task scheduling (immediate successor, LIFO vs FIFO)
    - Work-sharing malleability
    - Thread roles
  - Extend DLB (LeWI)
    - Pro-active policy
    - Handling I/O regions
  - Integration with Job scheduler
    - Based on TALP measures use DROM or LeWI and
    - MPI malleability
  - MNG module
    - Based on TALP metrics
    - Reduce number of resources through DROM or MPI malleability

# About DLB

- Current stable: Version 3.2 (2022-03-16)
  - LeWI
  - DROM
  - TALP

- Free Download under LGPL-v3 license: https://pm.bsc.es/dlb-downloads
- User Guide: https://pm.bsc.es/ftp/dlb/doc/user-guide/

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

# Thank you

marta.garcia@bsc.es

victor.lopez@bsc.es

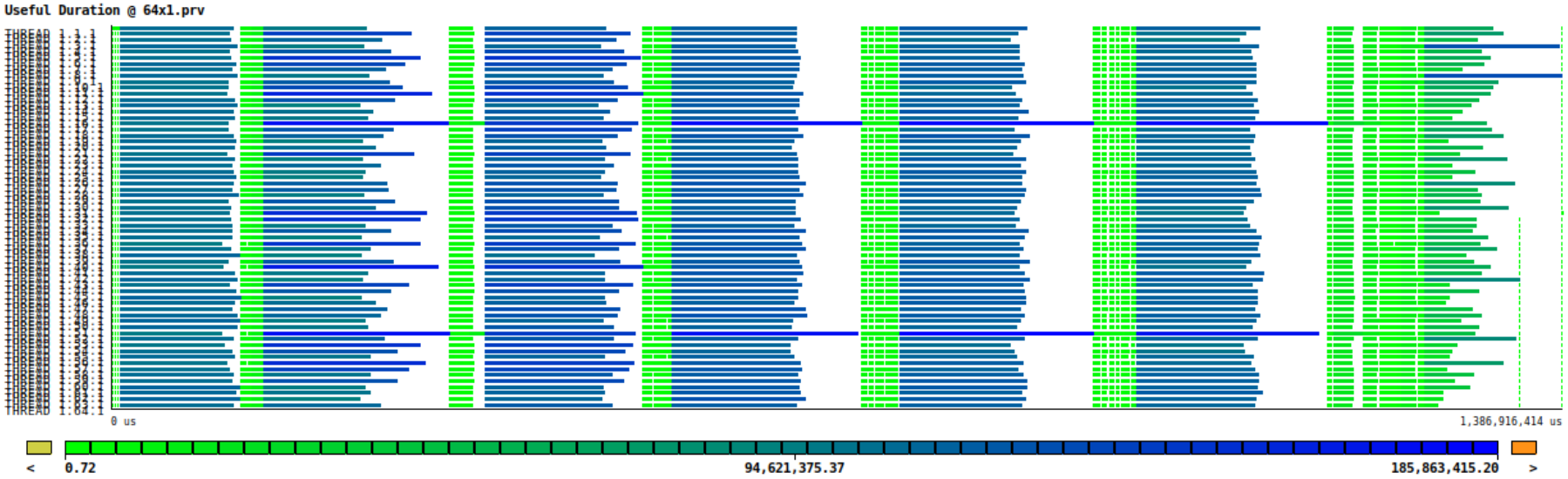
https://pm.bsc.es/dlb

# Backup


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

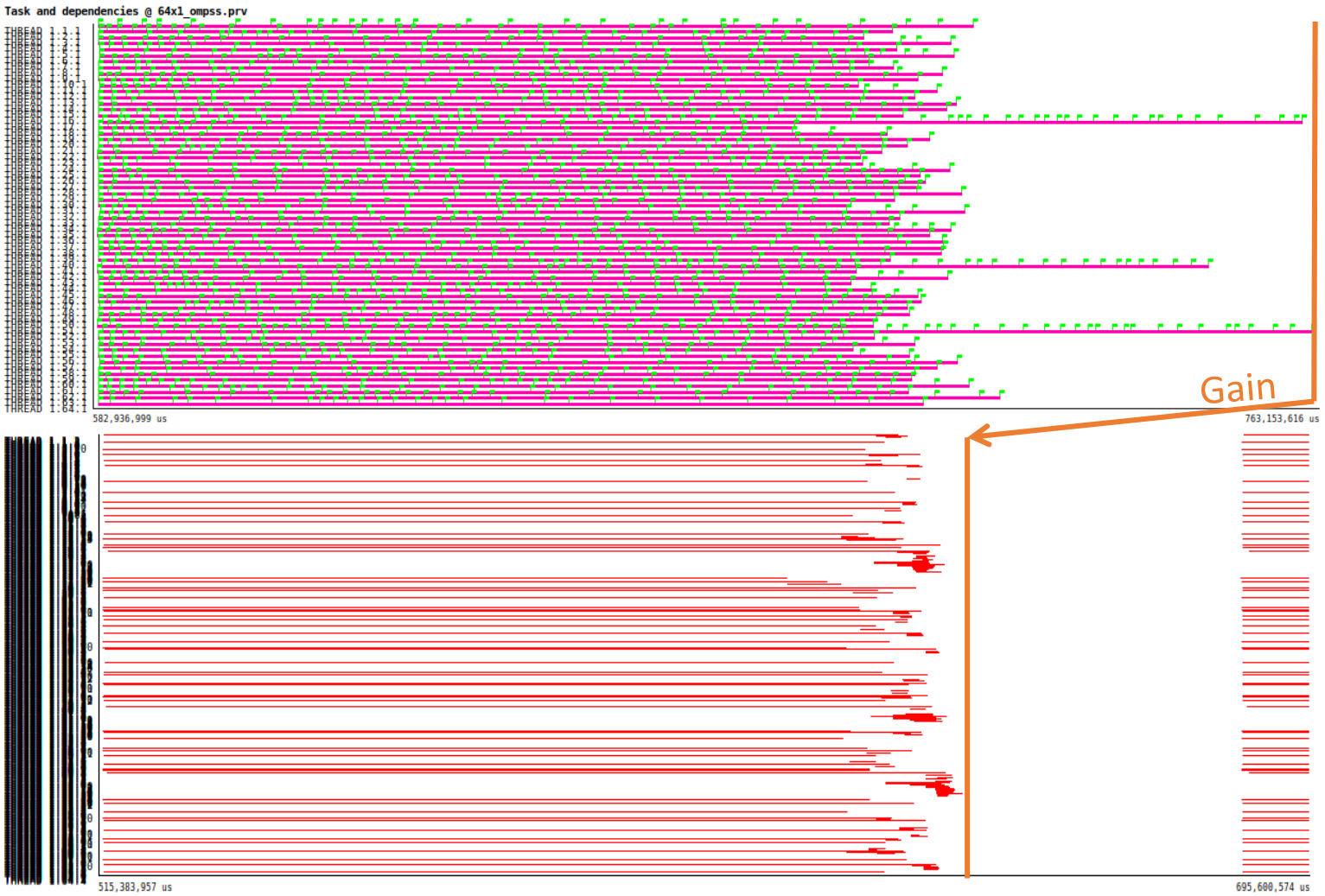
# ParMMG example

- ParMmg is an open-source software developed at INRIA for parallel mesh adaptation of 3D volume meshes. The parallel mesh adaptation algorithm is based on iterative remeshing and repartitioning of the distributed mesh. It uses the Mmg software to perform the sequential remeshing steps.
- Whole run: 6 steps with **high load imbalance** and **not regular**



# ParMMG with DLB

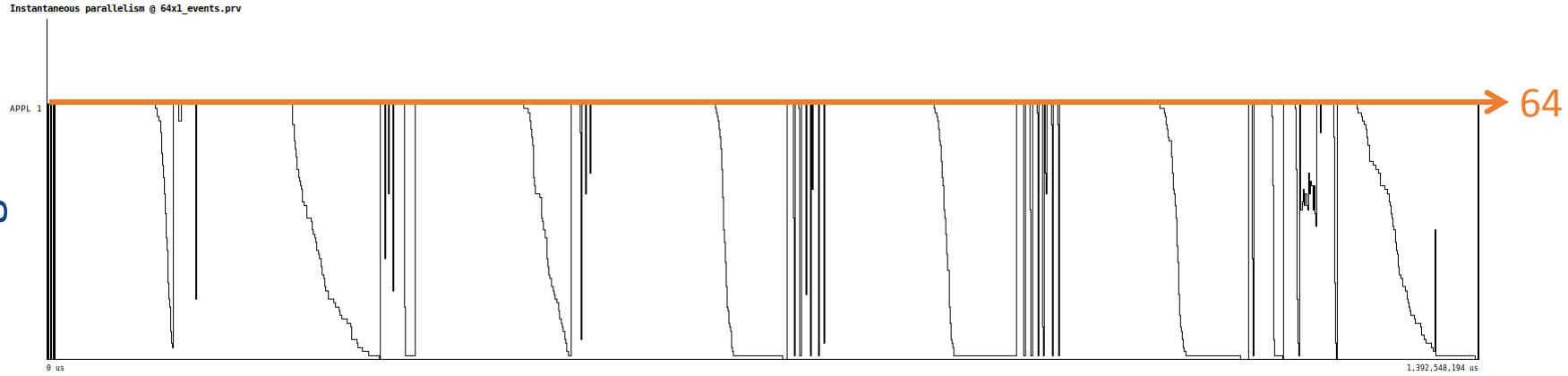
- Tasks
  - Original
  - With DLB



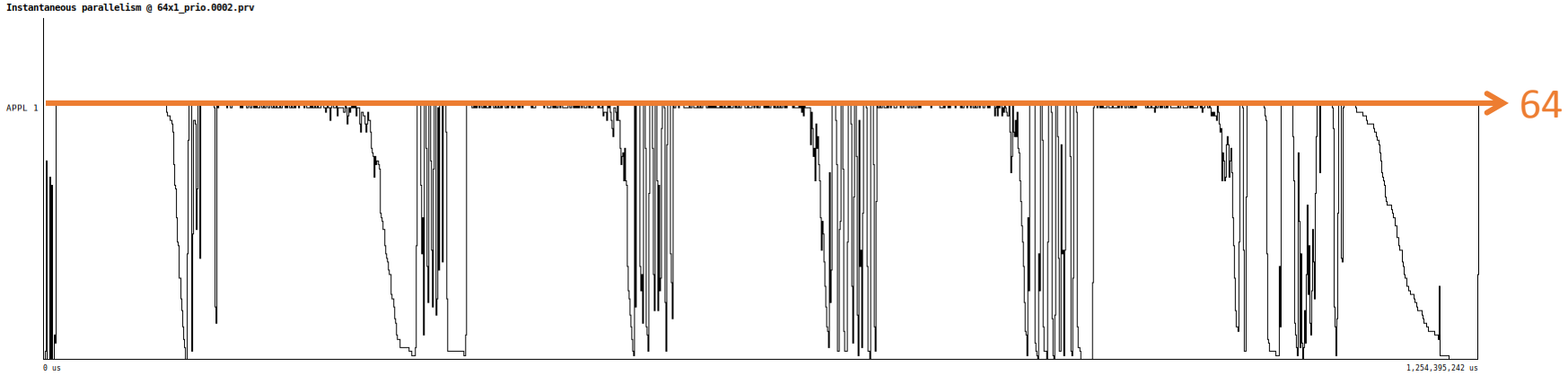
# ParMMG with DLB

- Instantaneous parallelism

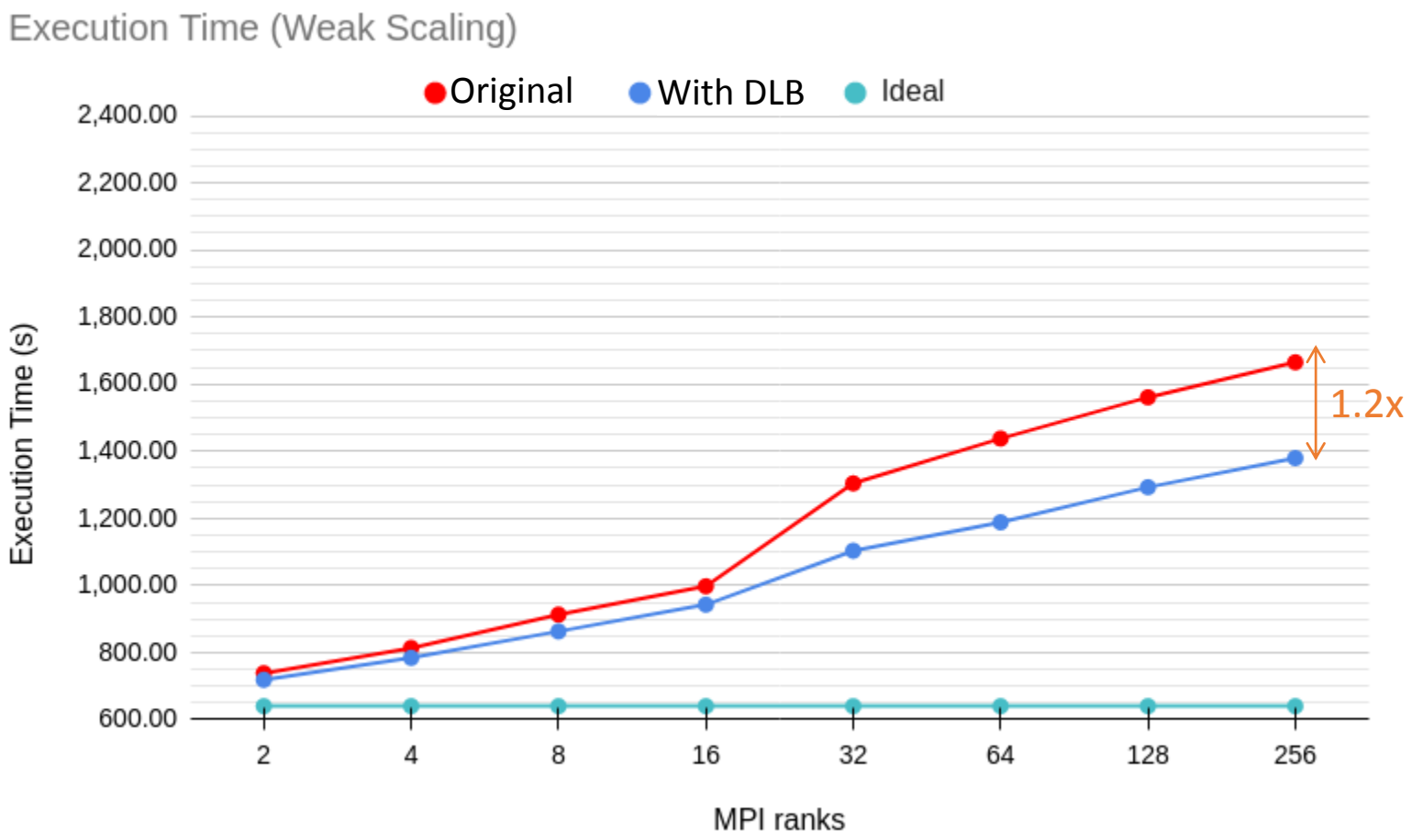
- Original




- With DLB



# ParMMG with DLB

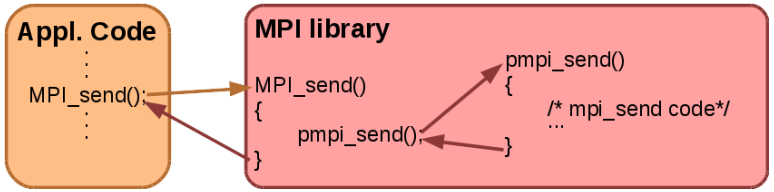


# Challenges

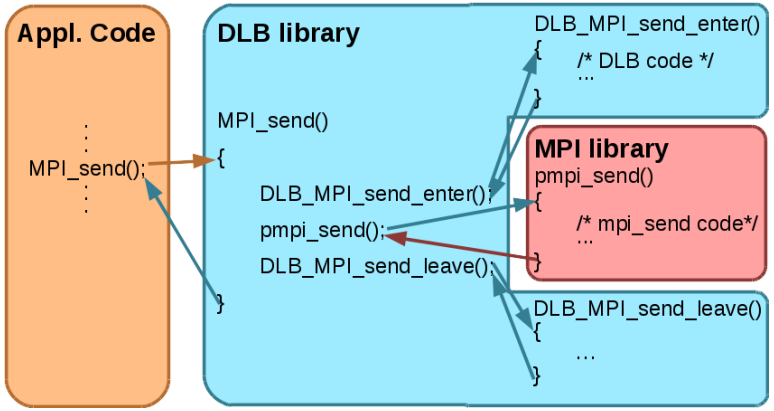
- Transversal to different layers, make the cooperate!!
  - MPI libraries are not willing to expose the non busy wait mode
    - They want all CPU cycles for them, but they are wasting them…
  - OS could help handling the cores? Giving priorities?
- Change mentality from “heroism programming” to trusting the runtime
  - Applications should stop doing things “by hand”
  - Let’s help them:
    - By addressing their needs and offering non intrusive solutions
    - By offering transversal solutions
- Malleability, malleability everywhere!!!
  - Application, Programming model, job scheduler…

# PMPI Interception

- PMPI: Profiling interface for MPI
  - MPI libraries implement an internal interface (PMPI) that implements the MPI call code

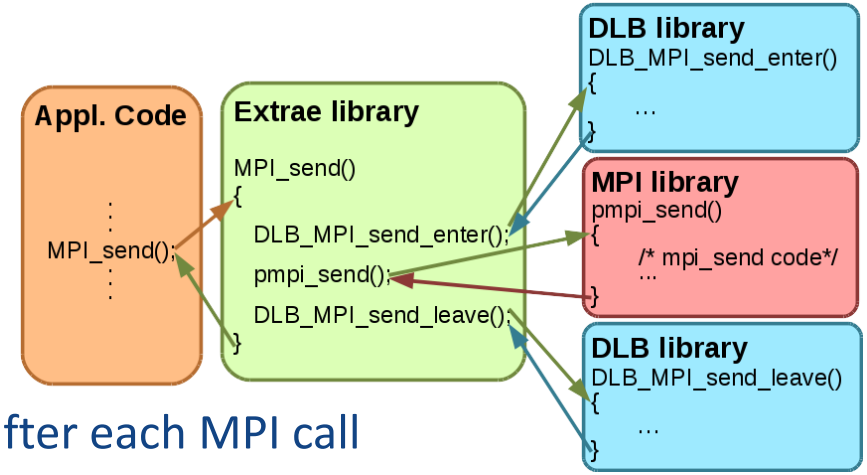



  - MPI calls can be redefined in a dynamic library
  - The intercepting library is loaded when starting the application
    - `export LD_PRELOAD = libdlb_mpi.so`
    - The dynamically loaded library has preference
  - Within the intercepted call the corresponding PMPI function must be called



# PMPI Interception

- Using DLB and Extrae
  - Both use PMPI interface




- Integration:
  - Extrae intercepts MPI calls with PMPI
  - DLB API called from Extrae before and after each MPI call
  - DLB does not intercept MPI calls
    - `export LD_PRELOAD = libdlb_mpi_instr.so`

- And other profiling tools using PMPI?
  - We are studding using PnMPI
    - Allows n tools intercepting MPI
    - An order between them must be selected
    - All the tools must support PnMPI
    - So far no conflicts have been found… Future Work

# MPI blocking mode

- MPI is greedy in the use of CPU
  - By default it will busy wait for messages/synchronizations to arrive
  - If the CPU is used by the MPI process waiting for the message we can not use it for doing useful computation by another thread.

- Different behavior for different MPI libraries 

- We have two options:
  - Leave all the CPUs assigned to a process but one
    - `export DLB_ARGS += "--lewi-mpi=no"`
  - Tell MPI not to busy wait
    - `export I_MPI_WAIT_MODE=1`
    - `export DLB_ARGS += "--lewi-mpi"`

# MPI blocking mode

- --lewi-mpi =no


- --lewi-mpi

# Integration with Nanos++

- There is no need to modify the application at all
  - The runtime will call the DLB API where necessary to ask for resources or return them
- Compile with Mercurium
- Run enabling DLB
  - Mandatory: `NX_ARGS+= "--enable-dlb --enable-block"`
  - Recommended: `NX_ARGS+= "--force-tie-master"`
  - In some cases: `NX_ARGS+= "--warmup-threads"`
- Win! 

# FAQ


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# FAQ

- Why not “learn” and use previous redistribution?
- What about data locality?
- My application does not perform well with OpenMP
- What about load balance between nodes?
- Why not overload CPUS, it’s the same you do!
- How do you decide to which process CPUS go?
- I already have a load balancing algorithm within my application
- How do I know the different options in DLB?

# Why not “learn” and use previous redistribution?

- There is a policy in DLB that does a “static” distribution of CPUs based in the load of each process
  - `--policy=WEIGHT`
    - Detects iterations, based in the MPI calls pattern
    - Computes an optimum distribution of CPUs
    - Applies it
  - Performance was much worse than LeWI → LeWI is more flexible
  - Code is deprecated

- Another policy that merge the functionality of WEIGHT and LeWI was implemented (Redistribute and Lend)
  - `--policy=RaL`
  - Performance was equal to the one obtained by LeWI

- We can recover these if we find the need

# How do you decide to which process CPUS go?

- We do not decide it, it is first come, first served
- So far, our experience is: If there is a free CPU and some one willing to use it, do it.
- But… we might implement some accounting in the future if more actors come in… different apps, different users, different programming models…
- We DO decide which CPU to take first…

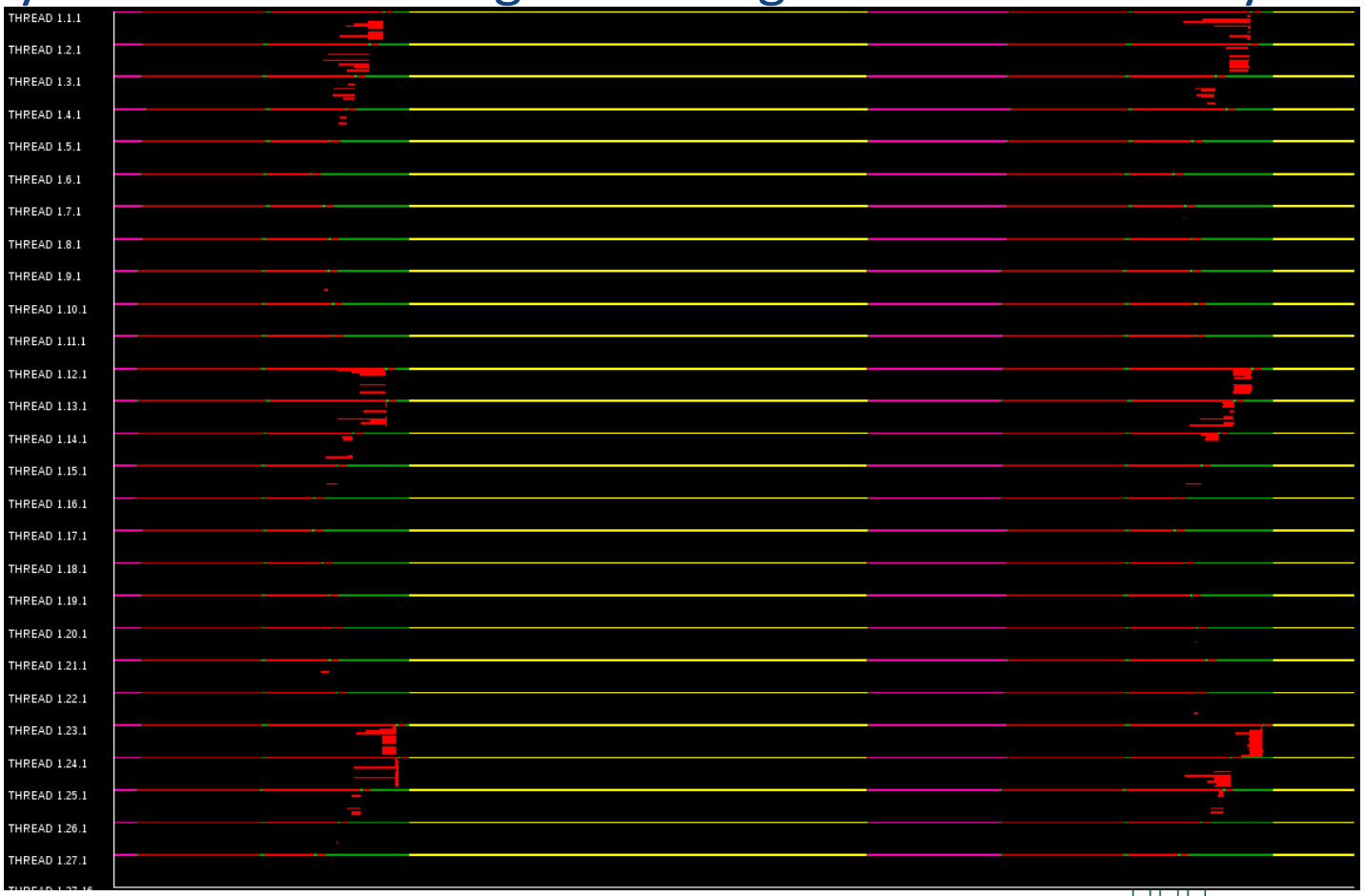
# What about data locality?

- In some kernels spawning threads to another socket can have a penalty

- We can choose with flag **`--lewi-affinity`** in DLB_ARGS environment variable which CPU a process will acquire **first** when asking for resources,,,
  - `any` : Take the first free CPU, does not take into account topology
  - `nearby-first` : Take first CPUs that are “affine” to me, and then the others
  - `spread-ifempty` : Take first CPUs that are affine to me, take CPUs from another socket only if all the CPUs in that socket are free (meaning no body is running there)
  - `nearby-only` : Take only CPUs that are affine to me

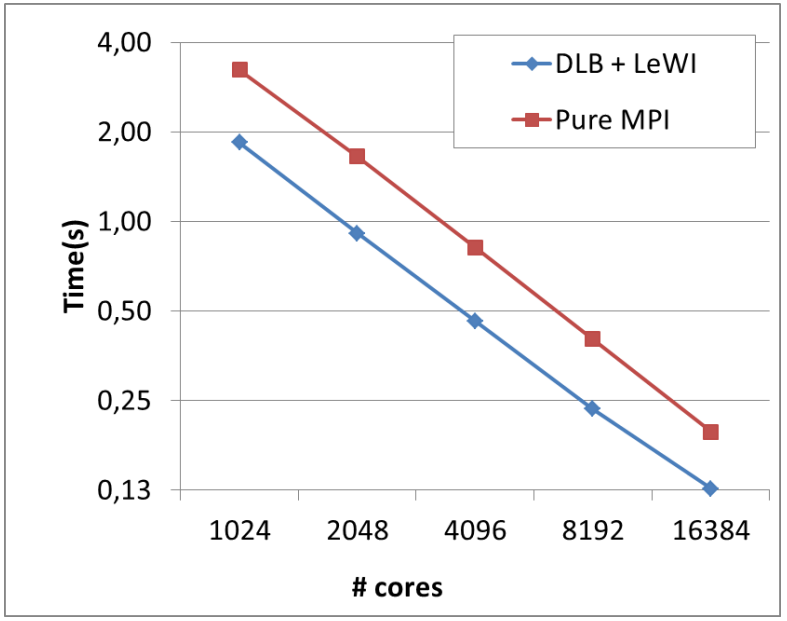
# My application does not perform well/it is not parallelized with OpenMP

- Don’t worry!
- In fact usually it is the best configuration… gives more flexibility to DLB



# What about load balance between nodes?

- We do not have any solution for this yet
- It is a quite different problem
  - Big difference in granularity, moving data between nodes is expensive
- But… good news is…
  - We are achieving very good results by balancing inside the node even when running up to 1024 nodes

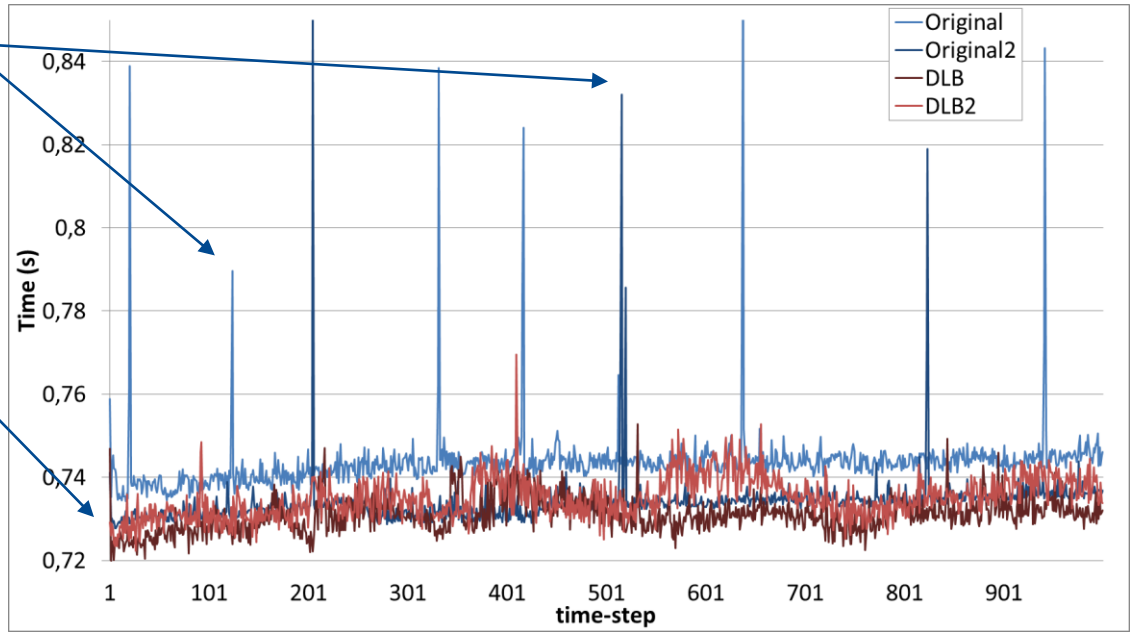


# I already have a load balancing algorithm within my application

- Does it solve this?

- Fine grain + system noise
- Blue lines original application (2 different runs)
- Red lines same run with DLB (2 different runs)

Clearly visible spikes without DLB are absorbed by DLB



# How do I know the different options in DLB?

- `[DLB_HOME]/bin/dlb –help`

```
The library configuration can be set using arguments
added to the DLB_ARGS environment variable.
Possible options are listed below:
--lewi:                 no              (bool)
--drom:                 no              (bool)
--mode:                 polling         [polling, async]
--verbose:          {api:microlb:shmem:mpi_api:mpi_intercept:stats:drom:async:ompt}
--verbose-format:       node:pid:thread {node:pid:mpinode:mpirank:thread}
--instrument:           yes             (bool)
--instrument-counters:  no              (bool)
--lewi-mpi:             no              (bool)
--lewi-mpi-calls:       all             [all, barrier, collectives]
--lewi-affinity:        nearby-first    [any, nearby-first, nearby-only, spread-
ifempty]
--lewi-greedy:          no              (bool)
--lewi-warmup:          no              (bool)
```