21st Summer School of PARALLEL COMPUTING

July 2 - 13, 2012 (Italian)
September 3 - 14, 2012 (English)

Introduction to Scalasca

Gabriele Fatigati - g.fatigati@cineca.it
SuperComputing Group

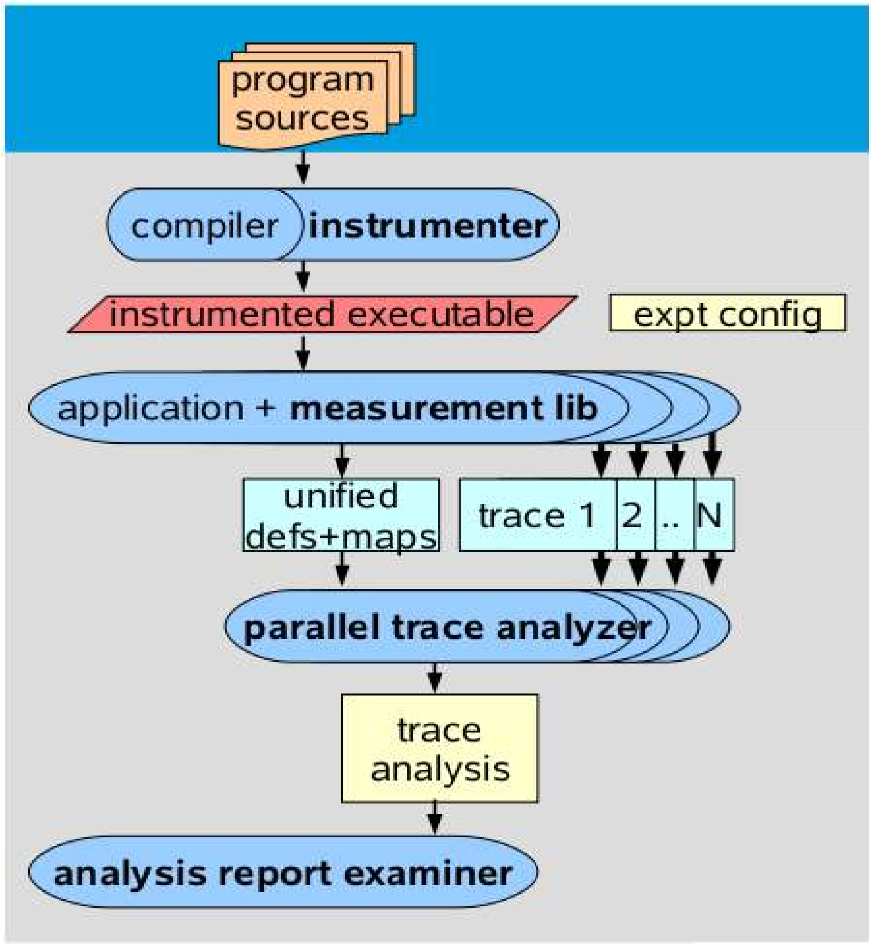
CINECA

- SCalable performance Analysis of LArge SCale Applications
- Developed by Julich Supercomputer Centre
- Toolset for performance analysis of parallel applications on a large scale
- Manage programs MPI, OpenMP, MPI+OpenMP
- Latest releast 1.3
- www.scalasca.org

Event tracing

During the measurement there is a buffer for each thread/process

Final collect of the results



Compilation

Original command:	SCALASCA instrumentation command:
mpcc -c foo.c	scalasca -instrument mpcc -c foo.c
mpxlf90 -o bar bar.f90	skin mpxlf90 -o bar bar.f90

Hybrid programming MPI+OpenMP

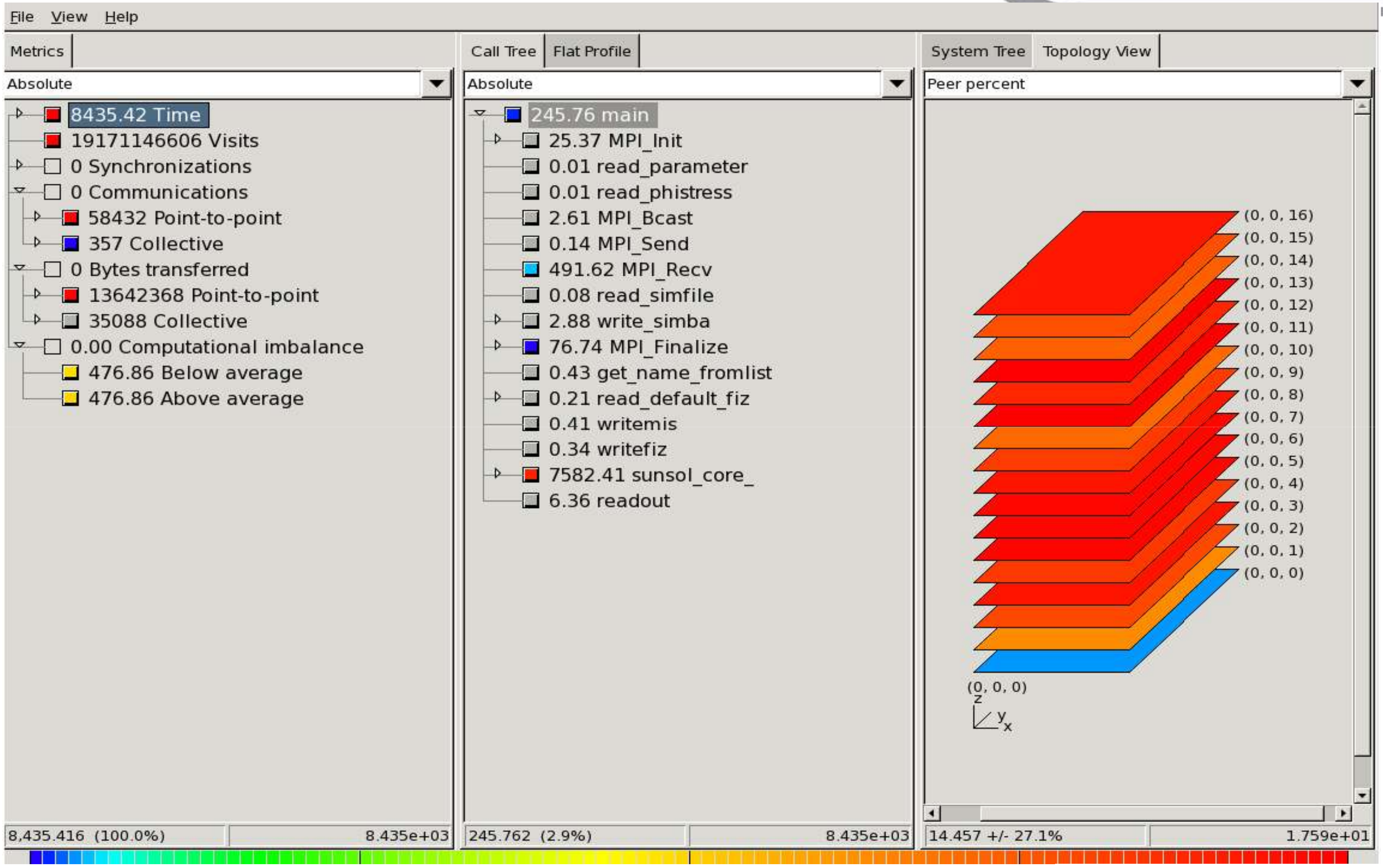
```
#!/bin/bash
#
# @ job_name     = myjob
# @ output       = myjob.$(jobid)
# @ error        = myjob.$(jobid)
# @ wall_clock_limit = 0:10:00
# @ total_tasks = 8
# @ task_affinity=core(1)
# @ parallel_threads=1
# @ job_type     = parallel
# @ resources    = ConsumableMemory(320Mb)
# @ queue

module load profile/advanced
module load qt/4.5.2--xl--10.1
module load scalasca/1.2

scalasca -analyze poe ./c_example
```

Results analysis:
scalasca -examine epik_...

Log

```
[00000]EPIK: Created new measurement archive ./epik_a
[00000]EPIK: Activated ./epik_a
  SWEEP3D – Pipelined Wavefront with Line-Recursion
  32 domains – 4 x 8 decomposition
  Iteration Monitor:
    its=1 err=1.000000 fixs=0
    ...
    its=12 err=5320.611978 fixs=19706584
  Balance quantities:
  ...
[00000]EPIK: Closing experiment ./epik_a
[00000]EPIK: 42 unique paths
[00000]EPIK: Unifying...done
[00000]EPIK: Collating...done
[00000]EPIK: Closed experiment ./epik_a
```

Hybrid programming MPI+OpenMP



File View Help

Metrics — Absolute

```
8435.42 Time
19171146606 Visits
0 Synchronizations
0 Communications
    58432 Point-to-point
    357 Collective
0 Bytes transferred
    13642368 Point-to-point
    35088 Collective
0.00 Computational imbalance
    476.86 Below average
    476.86 Above average
```

8,435.416 (100.0%) 8.435e+03

Call Tree | Flat Profile — Absolute

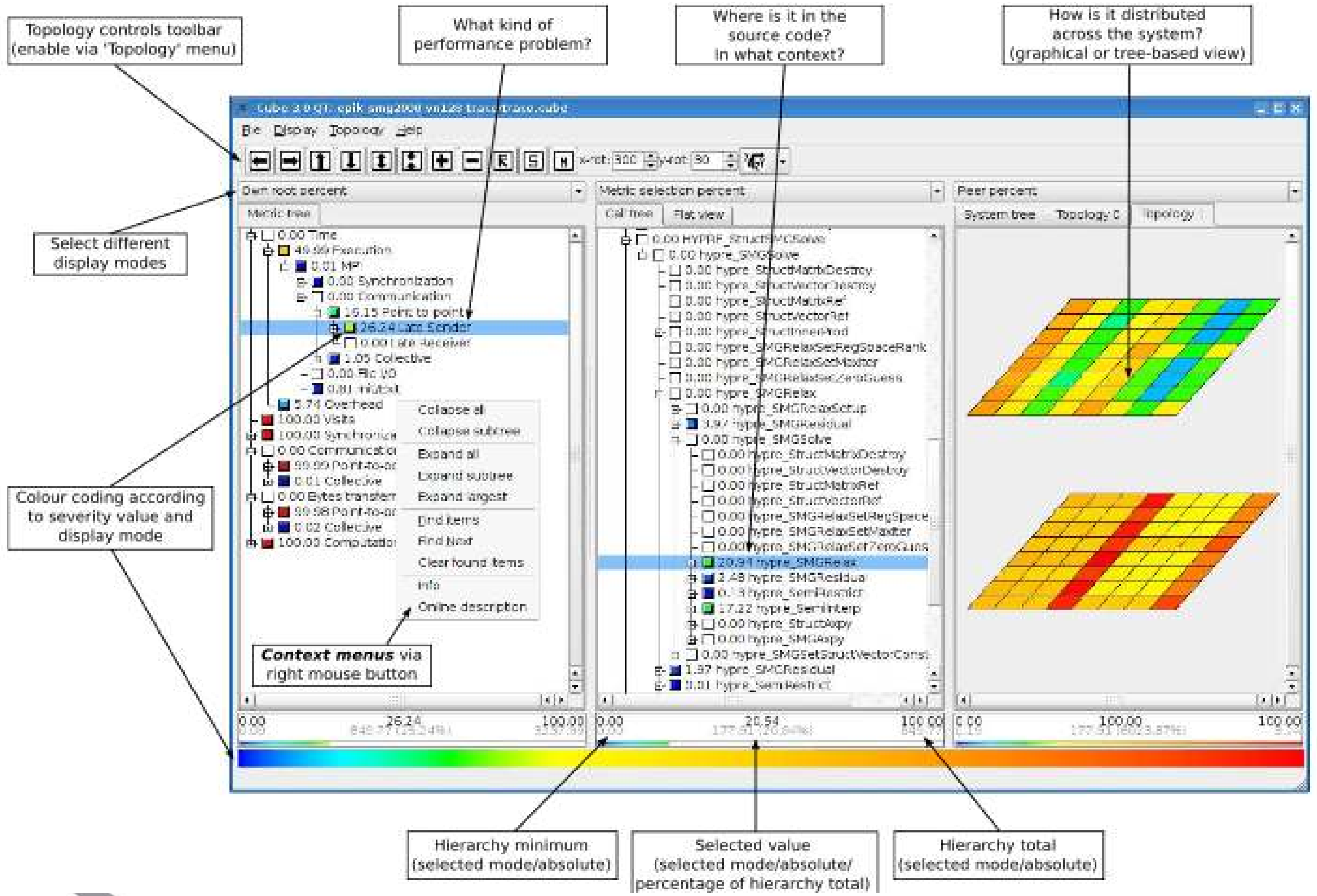
```
245.76 main
    25.37 MPI_Init
    0.01 read_parameter
    0.01 read_phistress
    2.61 MPI_Bcast
    0.14 MPI_Send
    491.62 MPI_Recv
    0.08 read_simfile
    2.88 write_simba
    76.74 MPI_Finalize
    0.43 get_name_fromlist
    0.21 read_default_fiz
    0.41 writemis
    0.34 writefiz
    7582.41 sunsol_core_
    6.36 readout
```

245.762 (2.9%) 8.435e+03

System Tree | Topology View — Peer percent

14.457 +/- 27.1% 1.759e+01

Hybrid programming MPI+OpenMP



Topology controls toolbar (enable via 'Topology' menu)
What kind of performance problem?
Where is it in the source code? In what context?
How is it distributed across the system? (graphical or tree-based view)
Select different display modes
Colour coding according to severity value and display mode
Context menus via right mouse button
Hierarchy minimum (selected mode/absolute)
Selected value (selected mode/absolute/ percentage of hierarchy total)
Hierarchy total (selected mode/absolute)
Collapse all
Collapse subtree
Expand all
Expand subtree
Expand largest
Find items
Find Next
Clear found items
Info
Online description
File
Display
Topology
Help
Own root percent
Metric selection percent
Peer percent
Metric tree
Call tree
Flat view
System tree
Topology 0
Topology 1

```
0.00 Time
  49.99 Execution
    0.01 MPI
      0.00 Synchronization
      0.00 Communication
        16.15 Point to point
          26.24 Late Sender
            0.00 Late Receiver
        1.05 Collective
      0.00 File I/O
      0.01 Init/Exit
  5.74 Overhead
100.00 Visits
100.00 Synchronizations
0.00 Communications
  99.99 Point-to-point
  0.01 Collective
0.00 Bytes transferred
  99.98 Point-to-point
  0.02 Collective
100.00 Computation
```

```
0.00 HYPRE_StructSMGSolve
  0.00 hypre_SMGSolve
    0.00 hypre_StructMatrixDestroy
    0.00 hypre_StructVectorDestroy
    0.00 hypre_StructMatrixRef
    0.00 hypre_StructVectorRef
    0.00 hypre_StructInnerProd
    0.00 hypre_SMGRelaxSetRegSpaceRank
    0.00 hypre_SMGRelaxSetMaxIter
    0.00 hypre_SMGRelaxSetZeroGuess
    0.00 hypre_SMGRelax
      0.00 hypre_SMGRelaxSetup
      3.97 hypre_SMGResidual
      0.00 hypre_SMGSolve
        0.00 hypre_StructMatrixDestroy
        0.00 hypre_StructVectorDestroy
        0.00 hypre_StructMatrixRef
        0.00 hypre_StructVectorRef
        0.00 hypre_SMGRelaxSetRegSpace
        0.00 hypre_SMGRelaxSetMaxIter
        0.00 hypre_SMGRelaxSetZeroGuess
        20.94 hypre_SMGRelax
        2.48 hypre_SMGResidual
        0.13 hypre_SemiRestrict
        17.22 hypre_SemiInterp
        0.00 hypre_StructAxpy
        0.00 hypre_SMGAxpy
      0.00 hypre_SMGSetStructVectorConst
    1.97 hypre_SMGResidual
    0.01 hypre_SemiRestrict
```

0.00	26.24	100.00
0.00	20.94	100.00
0.00	100.00	100.00

Topology view

- Hardware (only on some systems, like Blue Gene)
- MPI topology (eg: MPI_Cart_Create)
- Visual topology user-defined (next releases)

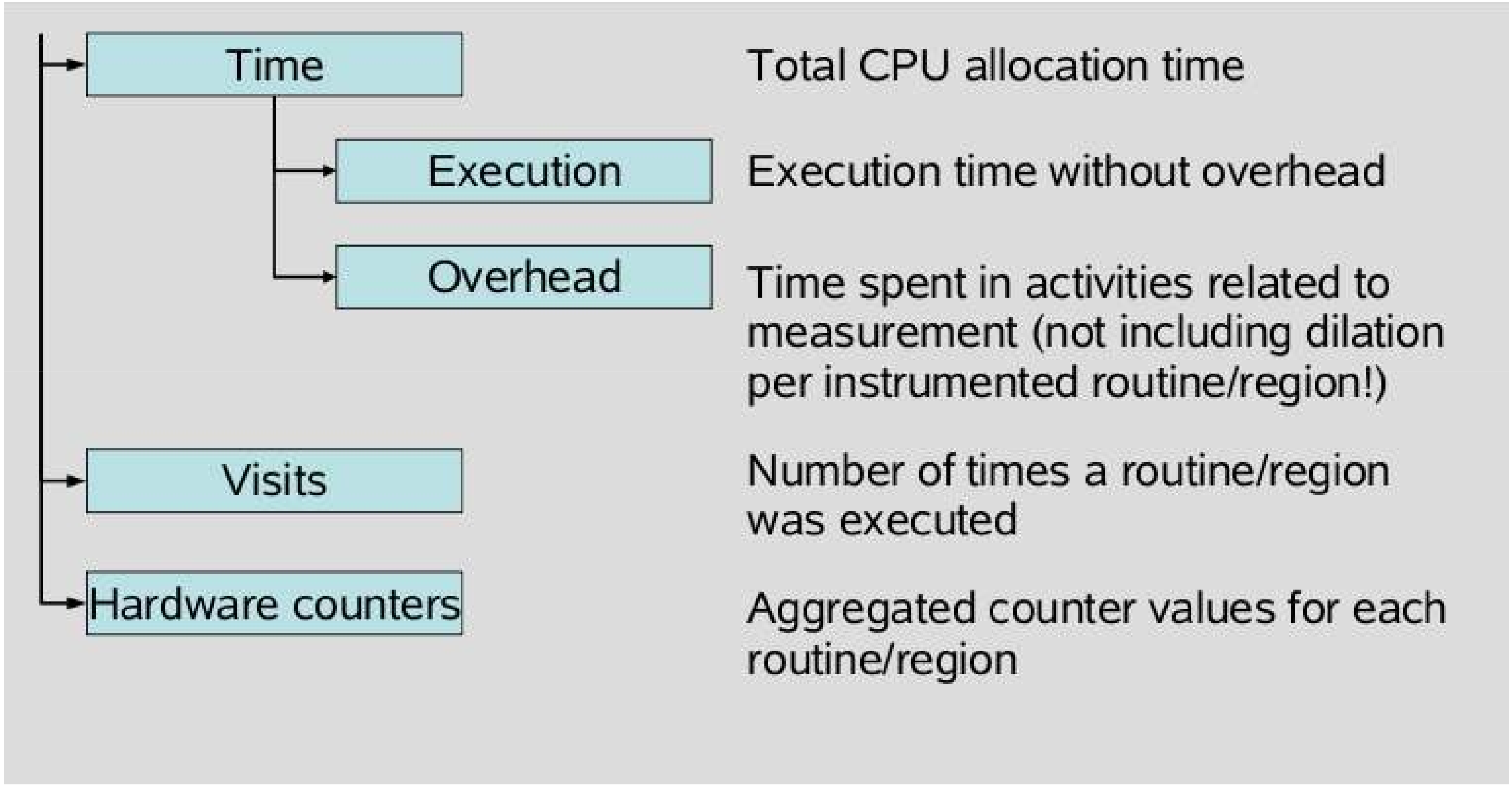
Currently supports cartesian topologies 1D,2D,3D

Display modes

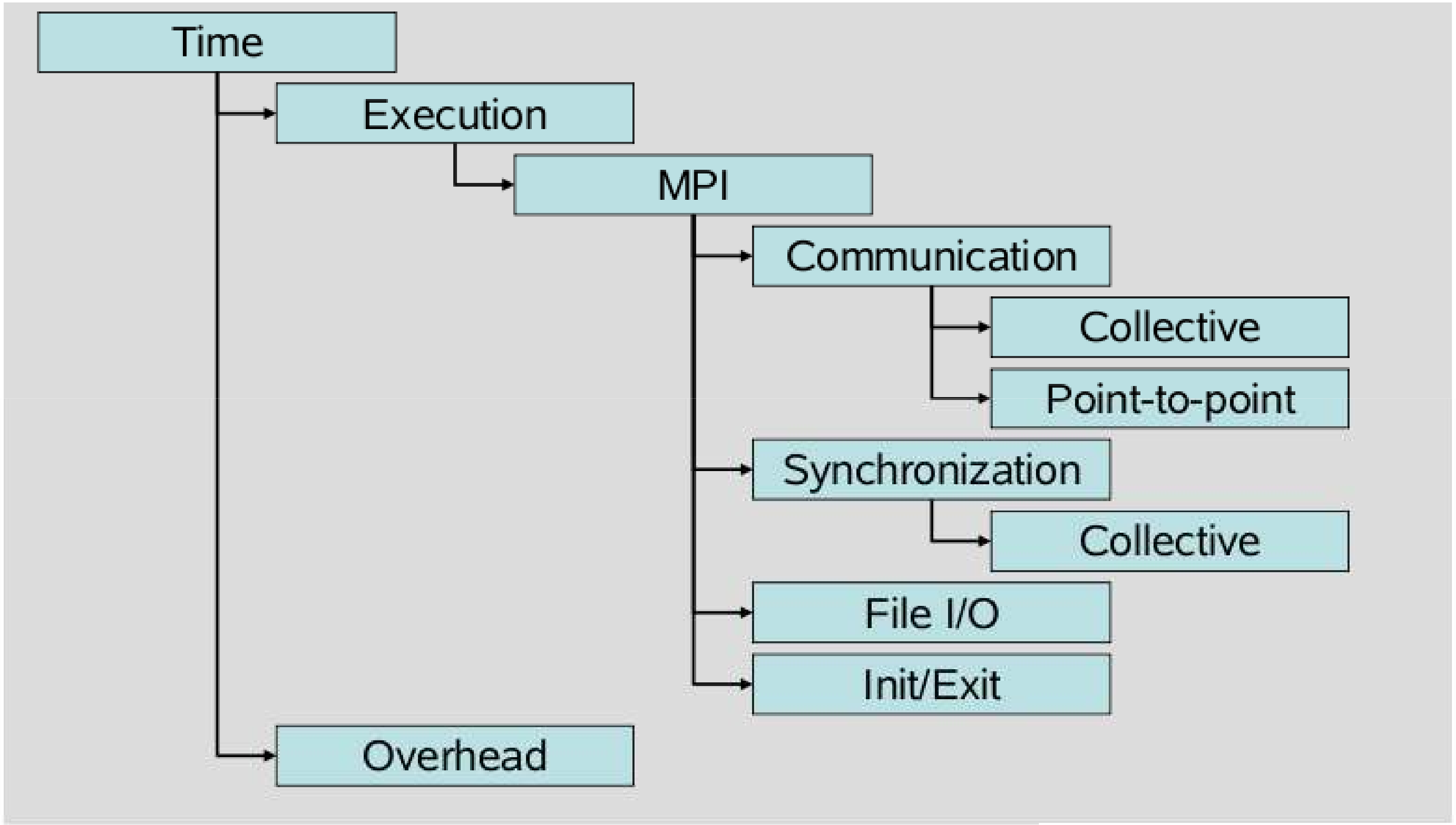
Absolute
Absolute value in seconds/number of occurrences

Root Percent
Percentage relative to the root ot the hierachy

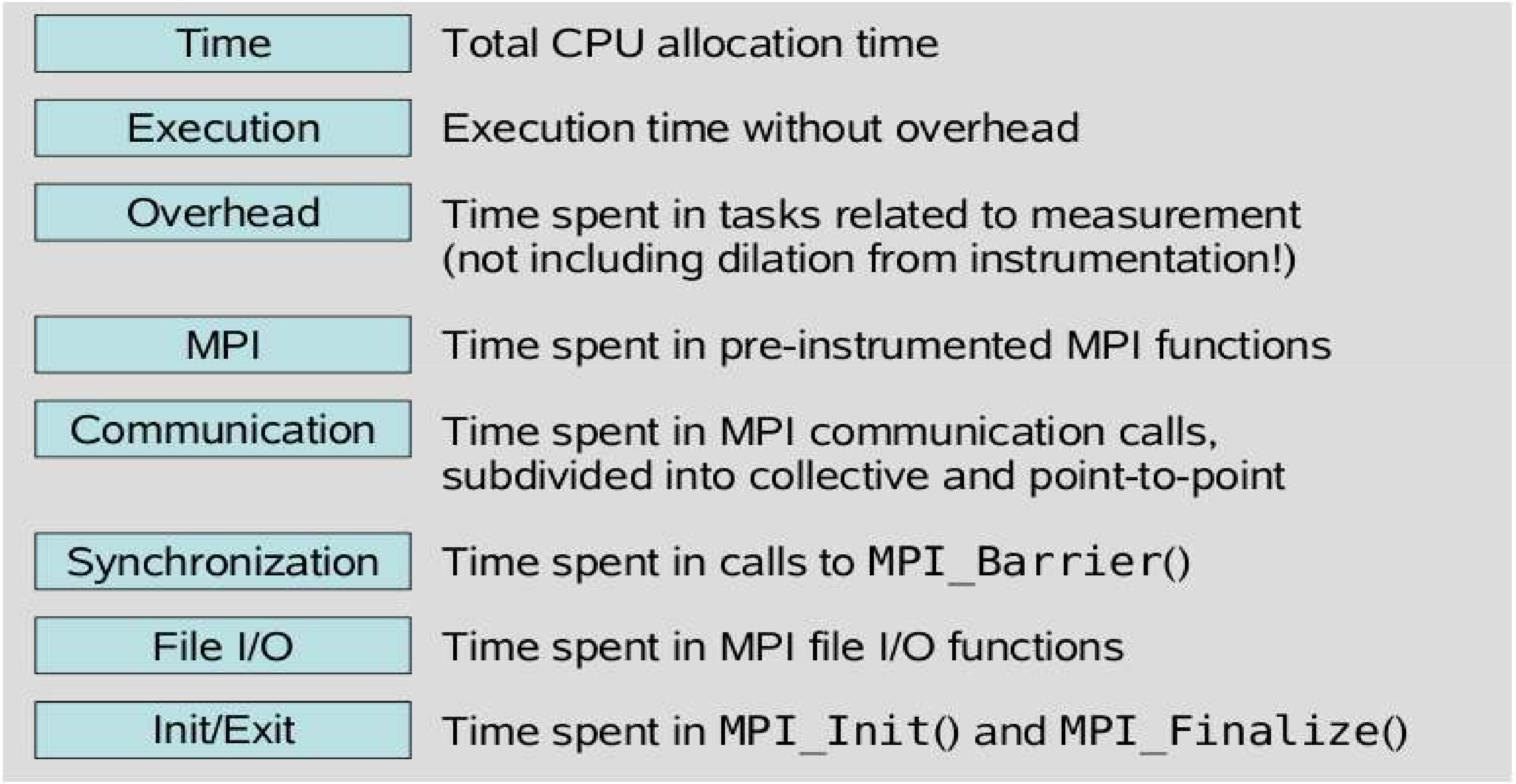
External percent
Similar to “root percent”, but for a different dataset



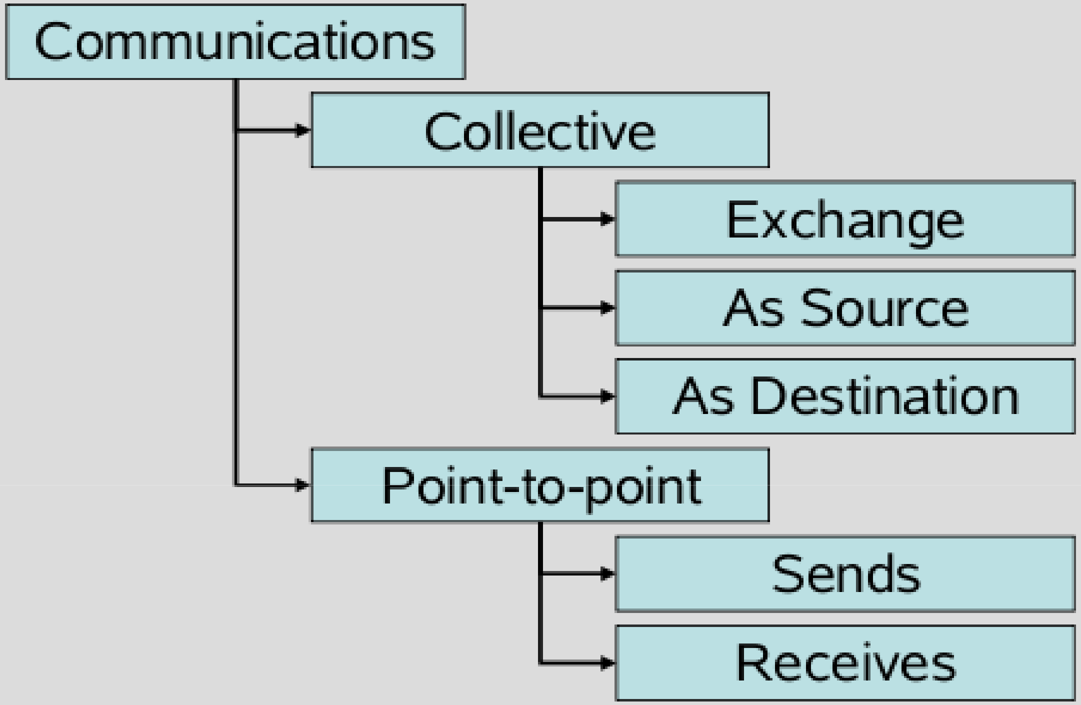
Time	Total CPU allocation time
Execution	Execution time without overhead
Overhead	Time spent in activities related to measurement (not including dilation per instrumented routine/region!)
Visits	Number of times a routine/region was executed
Hardware counters	Aggregated counter values for each routine/region


Time
Execution
MPI
Communication
Collective
Point-to-point
Synchronization
Collective
File I/O
Init/Exit
Overhead

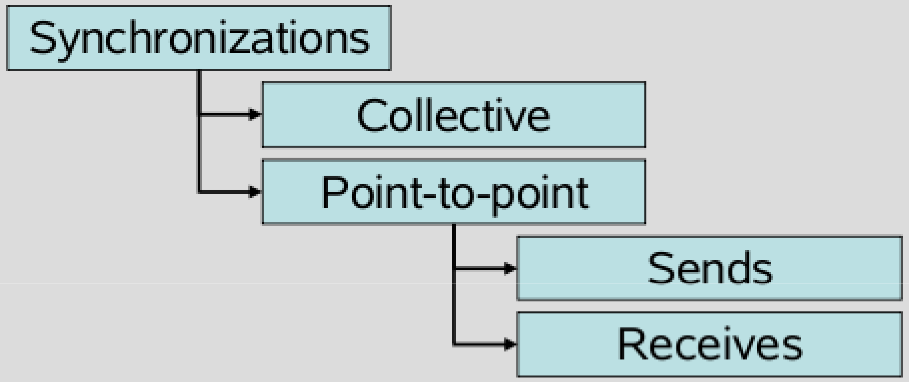
Hybrid programming MPI+OpenMP



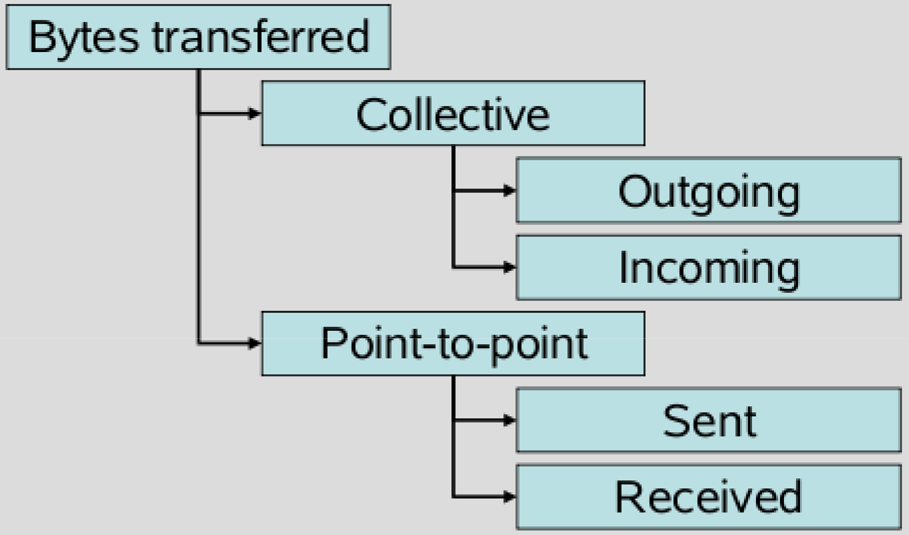
Time	Total CPU allocation time
Execution	Execution time without overhead
Overhead	Time spent in tasks related to measurement (not including dilation from instrumentation!)
MPI	Time spent in pre-instrumented MPI functions
Communication	Time spent in MPI communication calls, subdivided into collective and point-to-point
Synchronization	Time spent in calls to `MPI_Barrier()`
File I/O	Time spent in MPI file I/O functions
Init/Exit	Time spent in `MPI_Init()` and `MPI_Finalize()`



- Provides the number of calls to an MPI communication function of the corresponding class
- Zero-sized message transfers are considered *synchronization*!

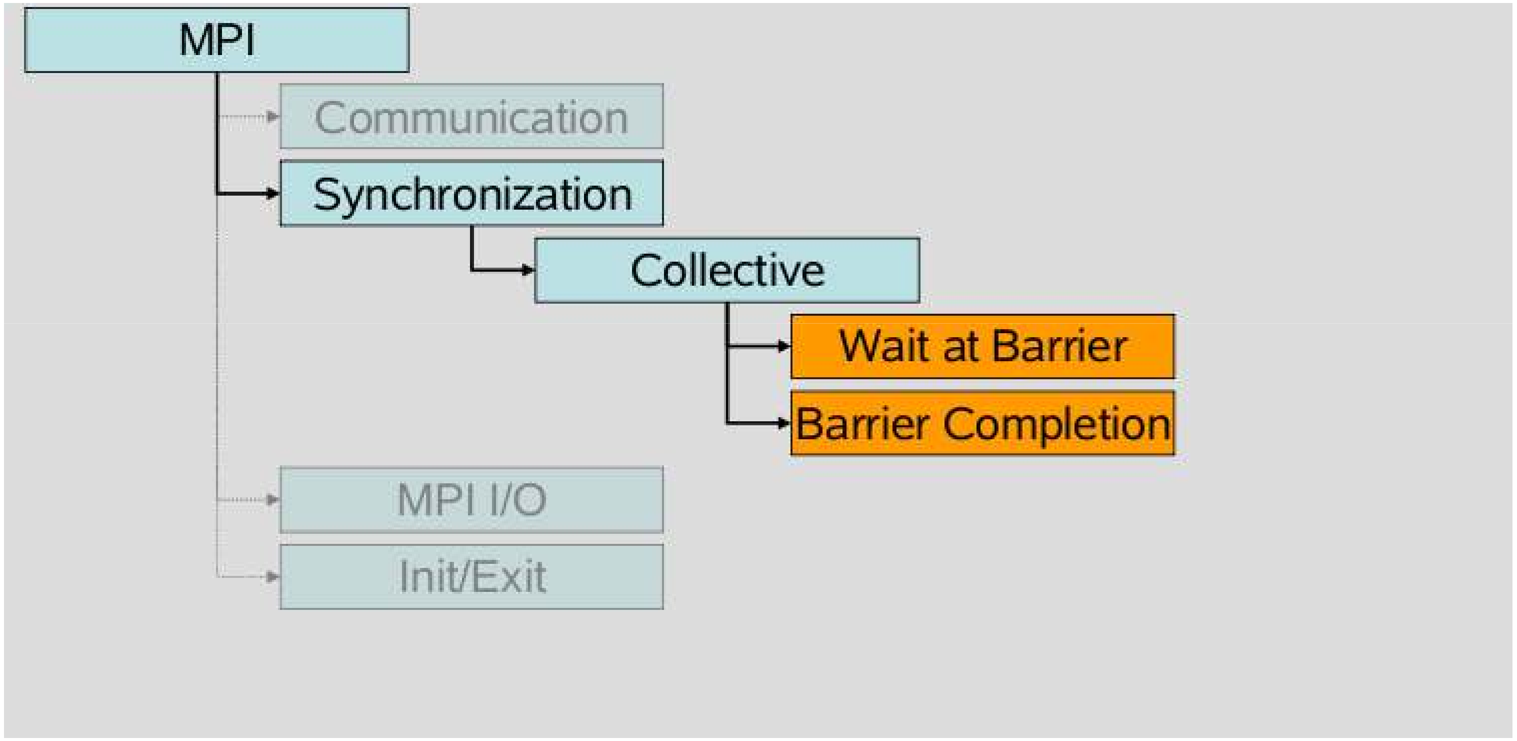


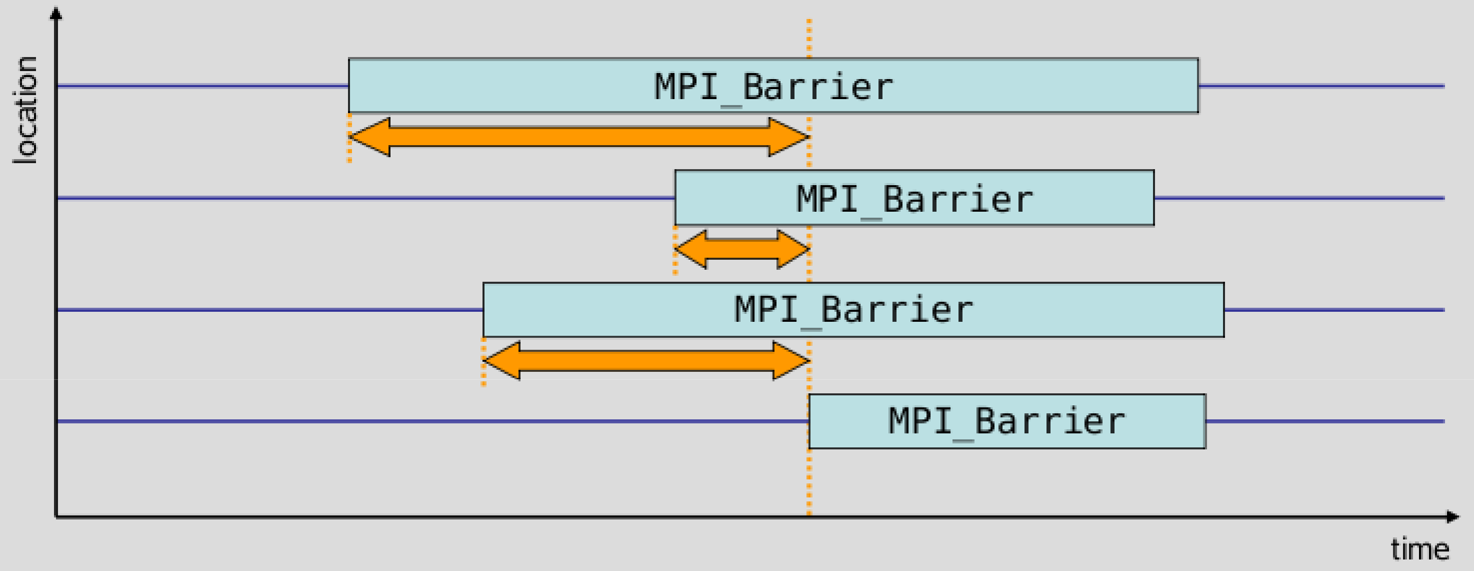
- Provides the number of calls to an MPI synchronization function of the corresponding class
- MPI synchronizations include zero-sized message transfers!



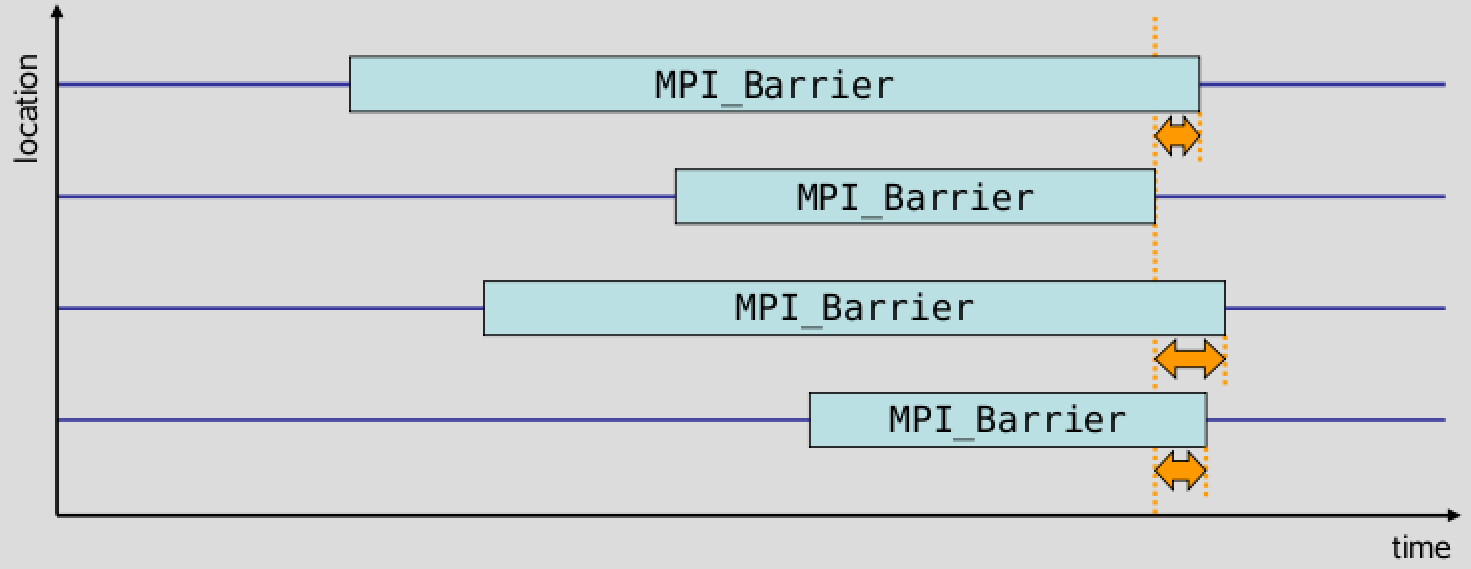
- Provides the number of bytes transferred by an MPI communication function of the corresponding class

MPI collective synchronization time

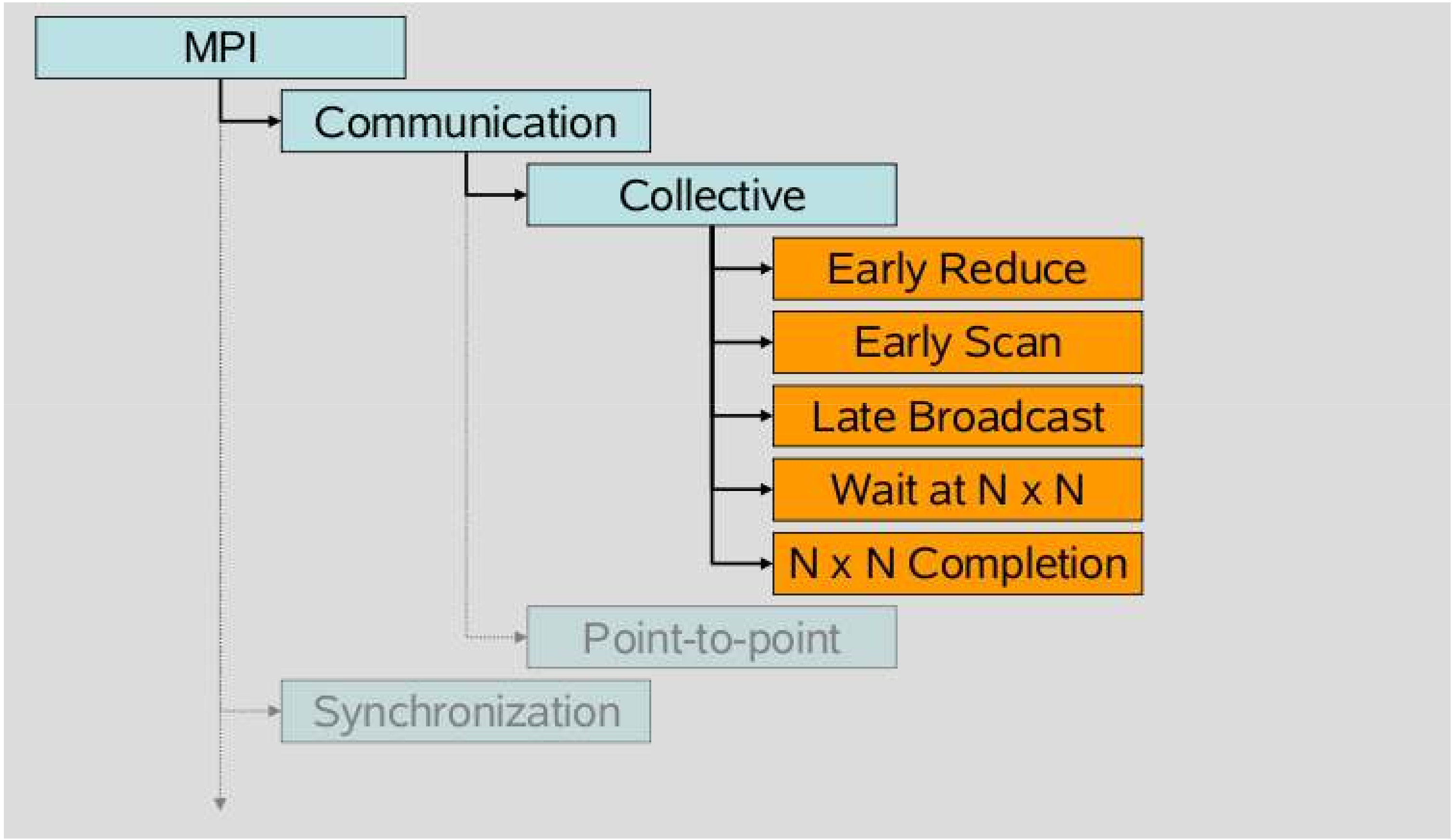


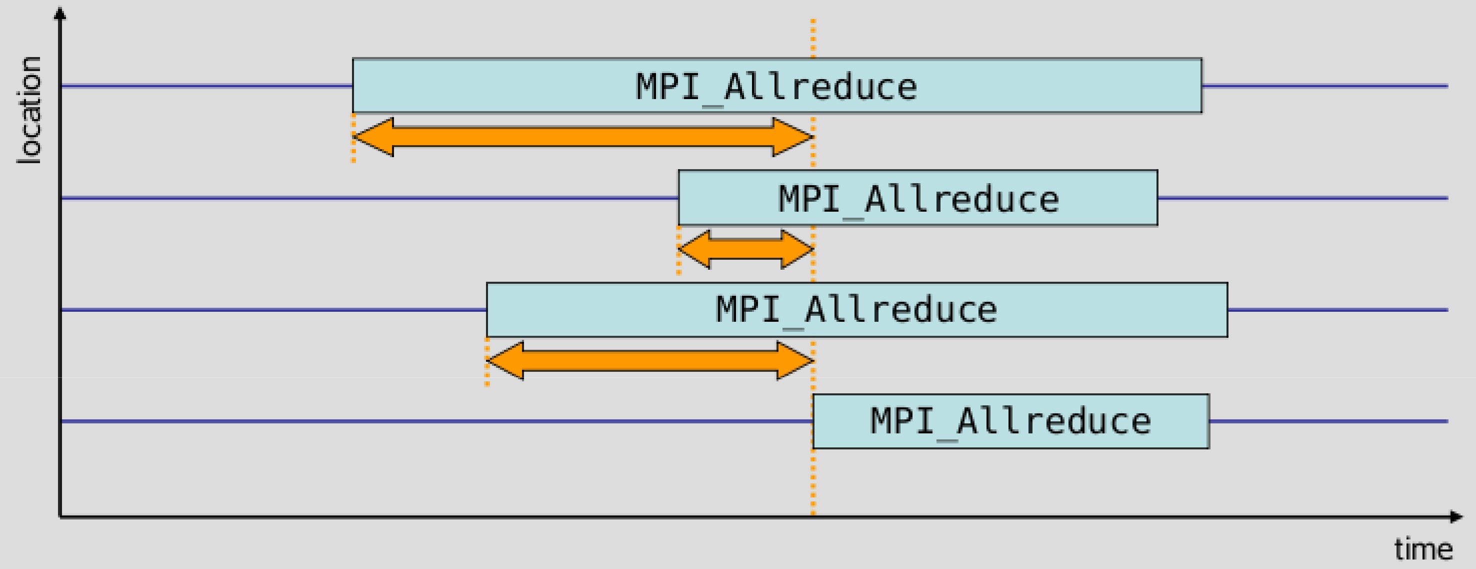


- Time spent waiting in front of a barrier call until the last process reaches the barrier operation
- Applies to: `MPI_Barrier()`

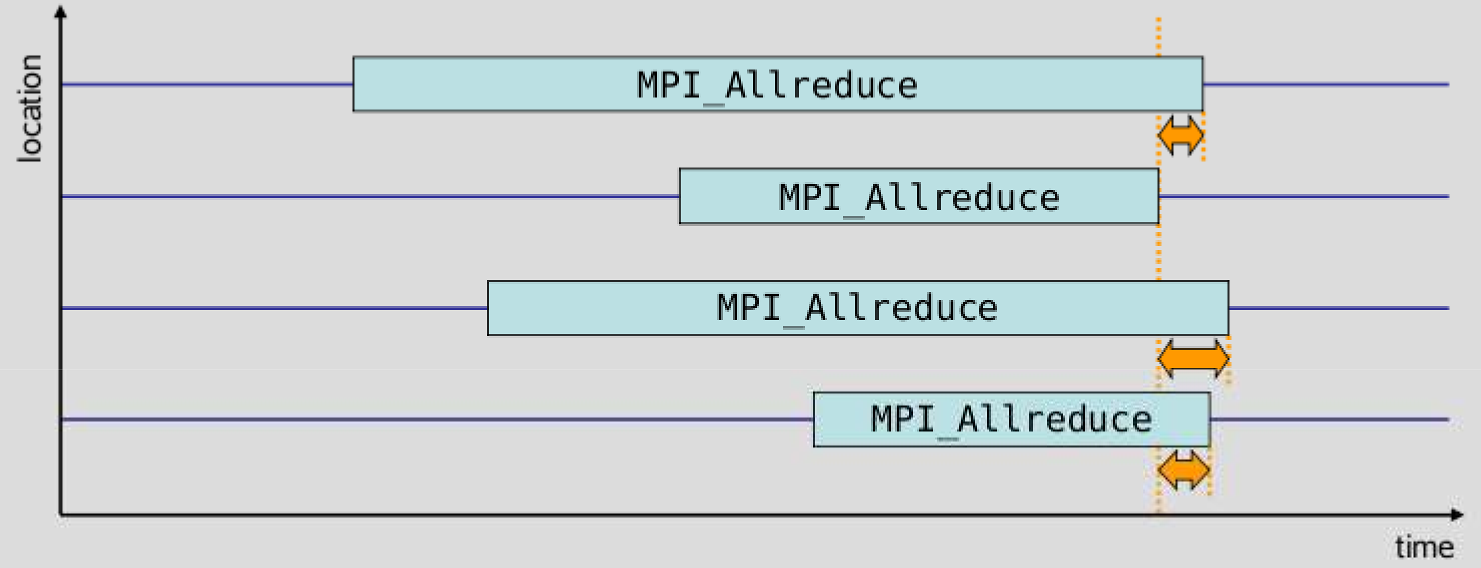


- Time spent in barrier after the first process has left the operation
- Applies to: `MPI_Barrier()`


MPI
Communication
Collective
Early Reduce
Early Scan
Late Broadcast
Wait at N x N
N x N Completion
Point-to-point
Synchronization

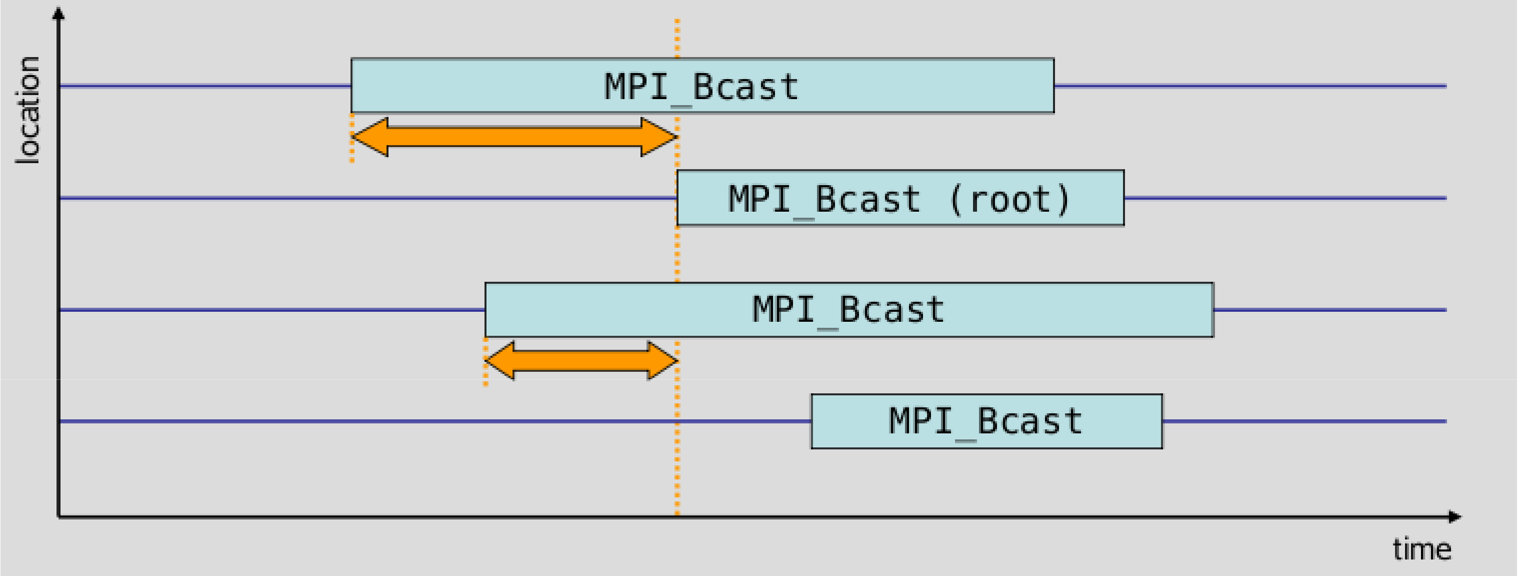


- Time spent waiting in front of a synchronizing collective operation call until the last process reaches the operation
- Applies to: MPI_Allreduce(), MPI_Alltoall(), MPI_Alltoallv(), MPI_Allgather(), MPI_Allgatherv(), MPI_Reduce_scatter()



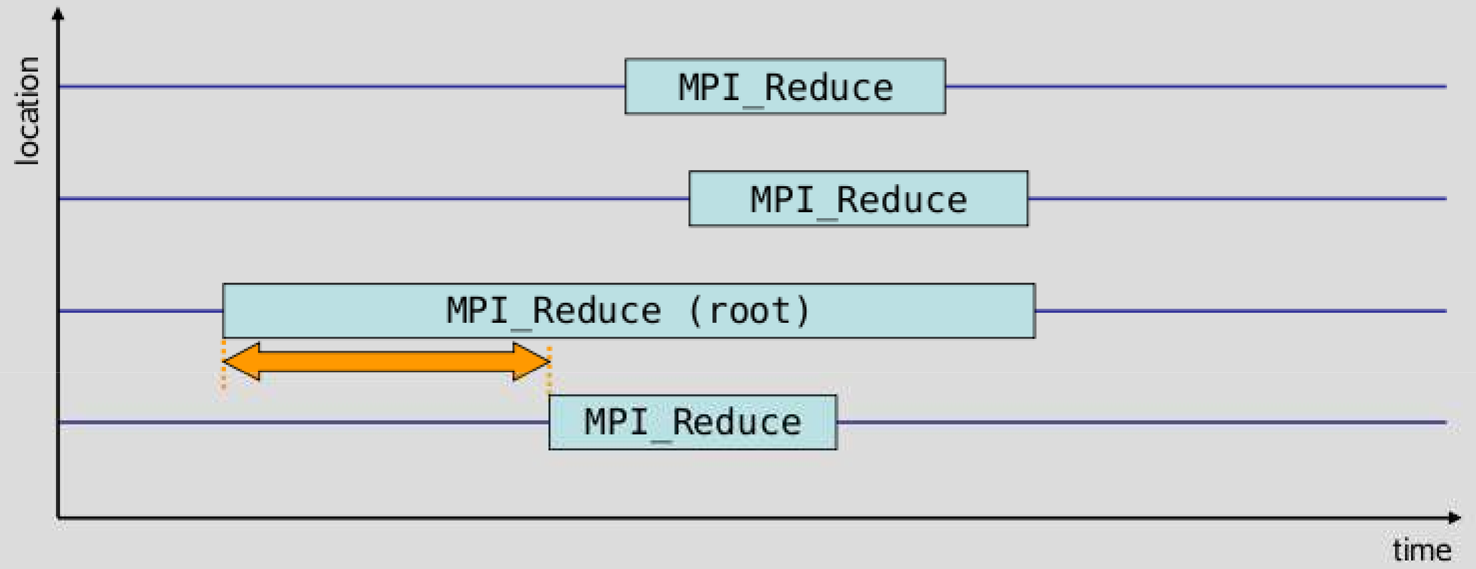
- Time spent in synchronizing collective operations after the first process has left the operation
- Applies to: MPI_Allreduce(), MPI_Alltoall(), MPI_Alltoallv(), MPI_Allgather(), MPI_Allgatherv(), MPI_Reduce_scatter()

Hybrid programming MPI+OpenMP

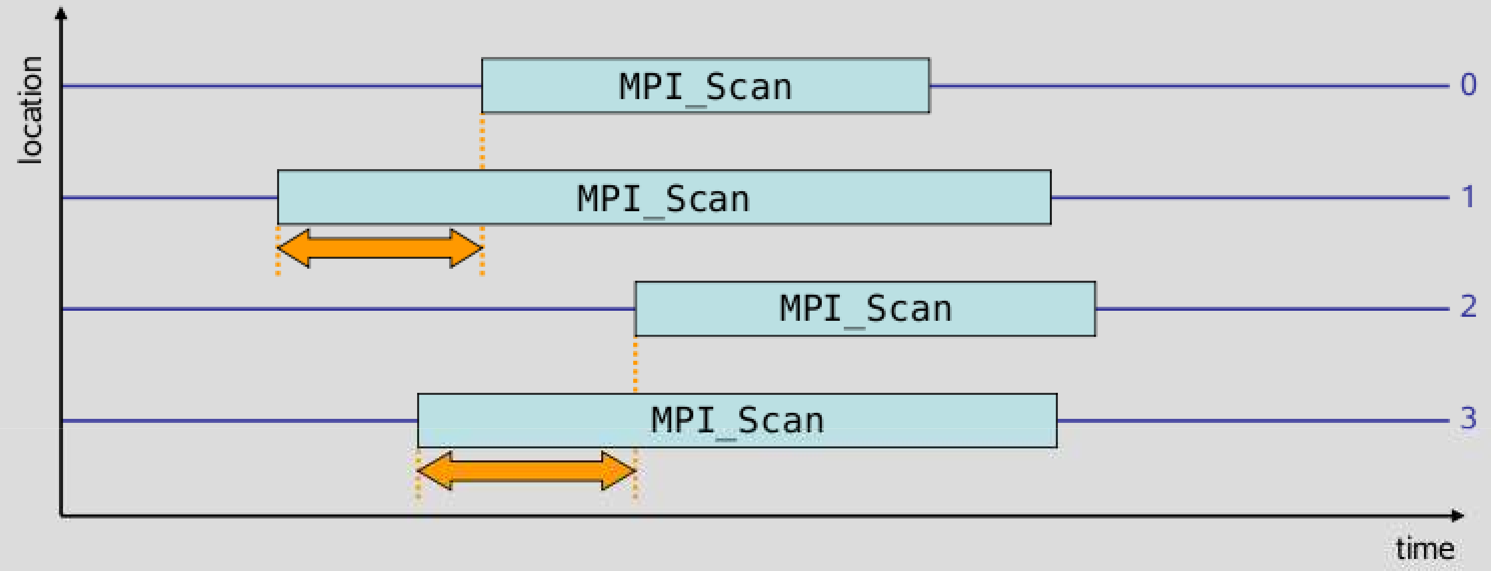


- Waiting times of the destination processes of a collective 1-to-N communication operation which enter the operation earlier than the source process (root)
  - Late Broadcast by source = Early Broadcast by destinations
- Applies to: `MPI_Bcast()`, `MPI_Scatter()`, `MPI_Scatterv()`

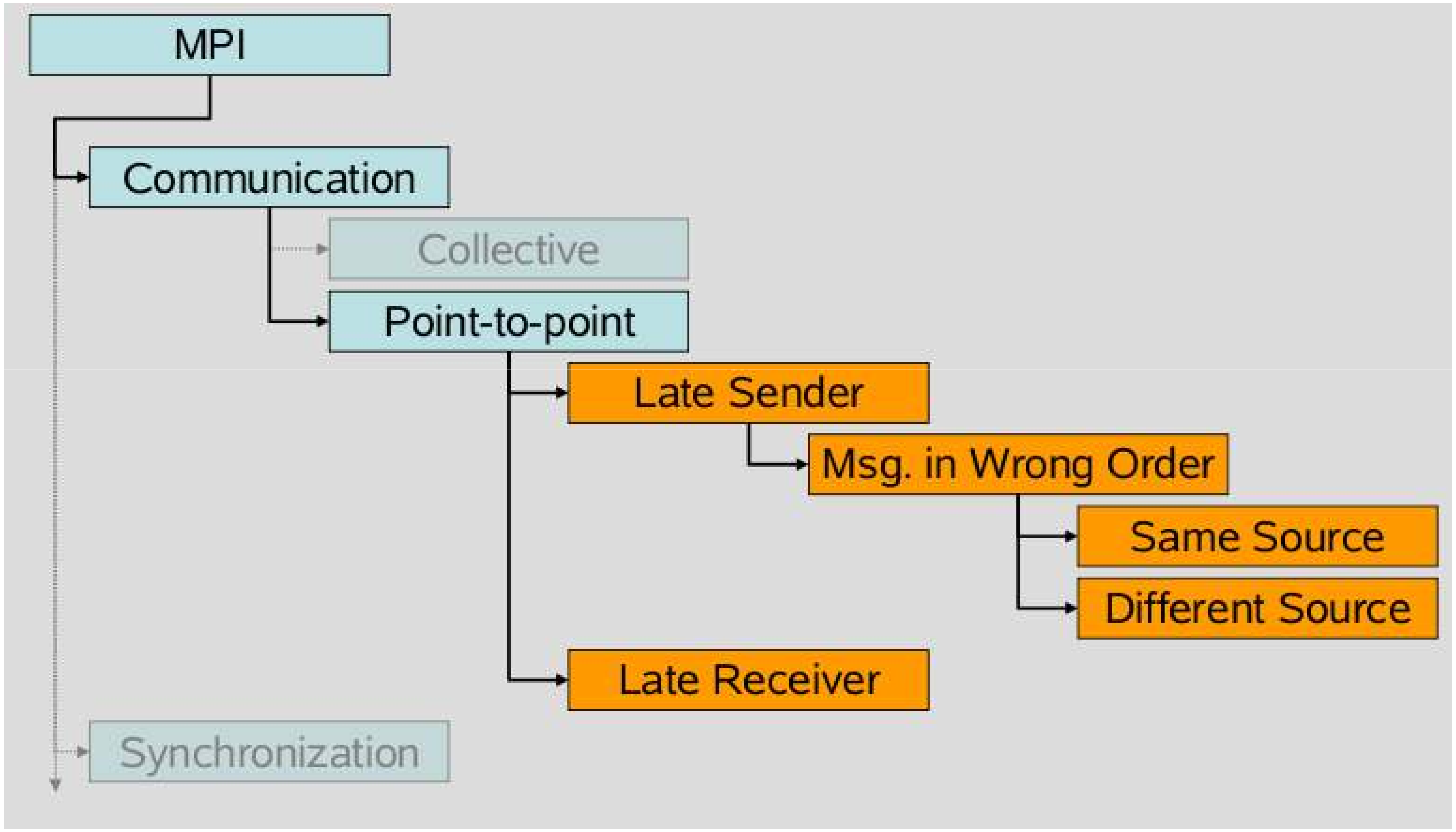
Hybrid programming MPI+OpenMP

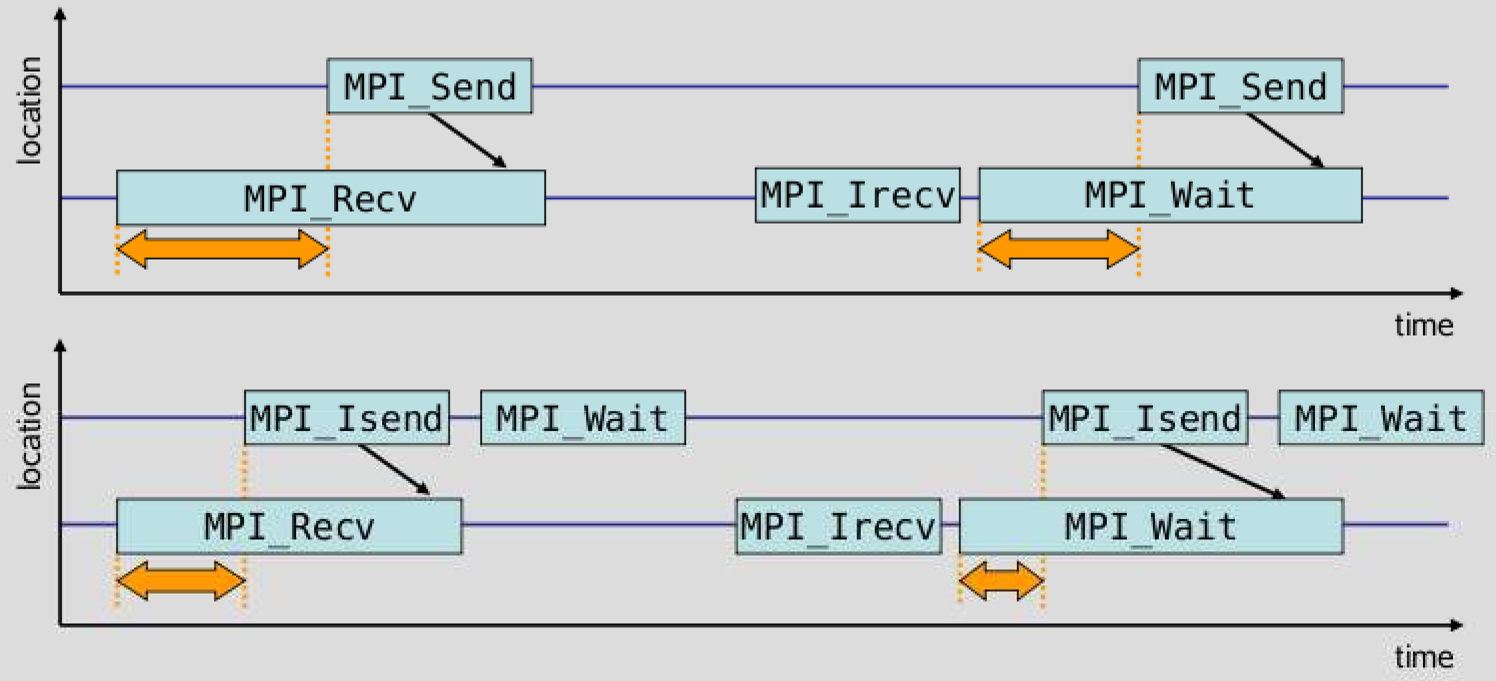


- Waiting time if the destination process (root) of a collective N-to-1 communication operation enters the operation earlier than its sending counterparts
- Applies to: `MPI_Reduce()`, `MPI_Gather()`, `MPI_Gatherv()`

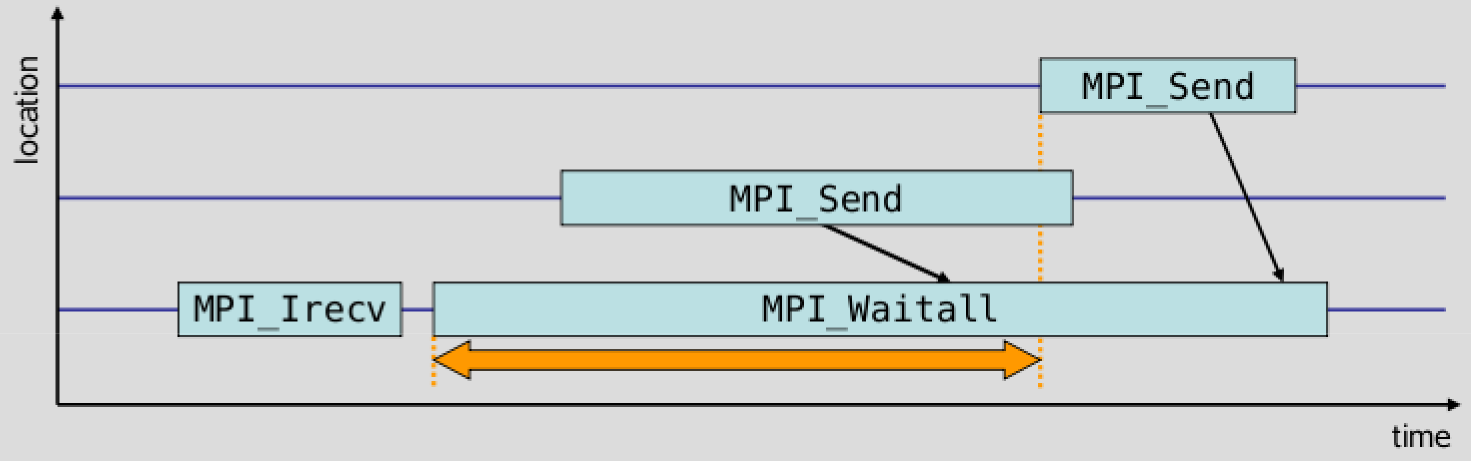


- Waiting time if process *n* enters a prefix reduction operation earlier than its sending counterparts (i.e., ranks 0..*n*-1)
- Applies to: `MPI_Scan()`

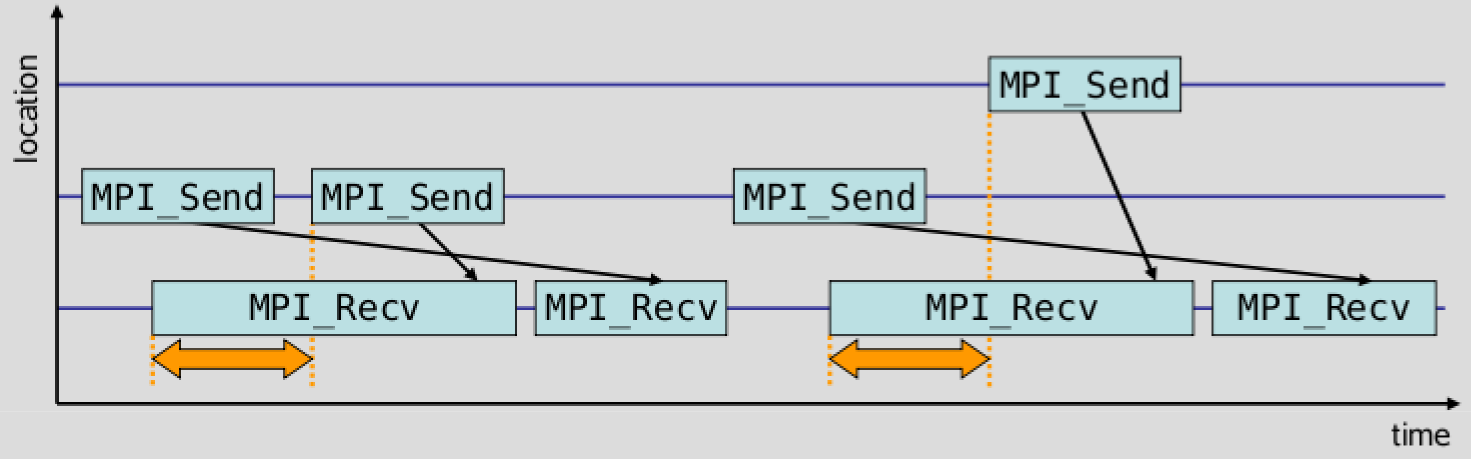

MPI
Communication
Collective
Point-to-point
Late Sender
Msg. in Wrong Order
Same Source
Different Source
Late Receiver
Synchronization



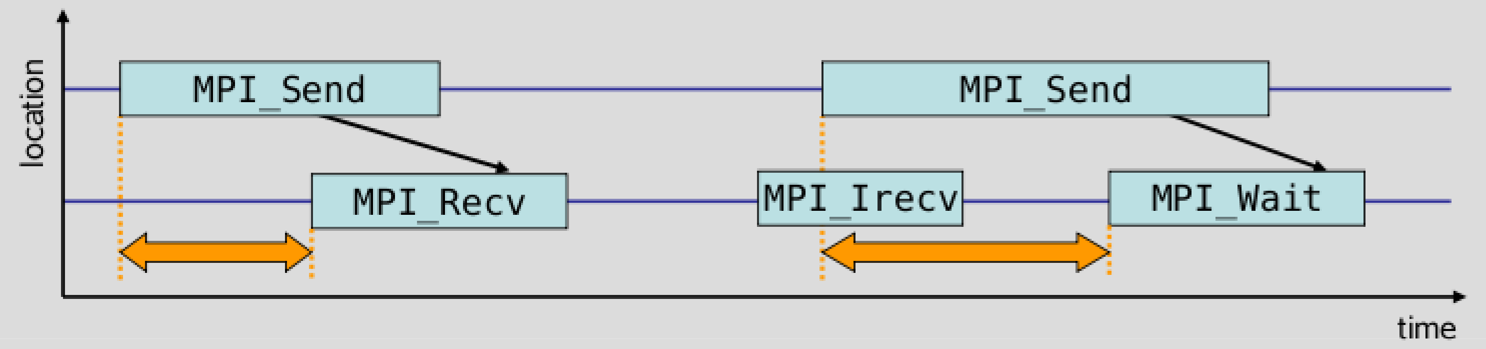
- Waiting time caused by a blocking receive operation posted earlier than the corresponding send operation
- Applies to blocking as well as non-blocking communication



- While waiting for several messages, the maximum waiting time is accounted
- Applies to: `MPI_Waitall()`, `MPI_Waitsome()`

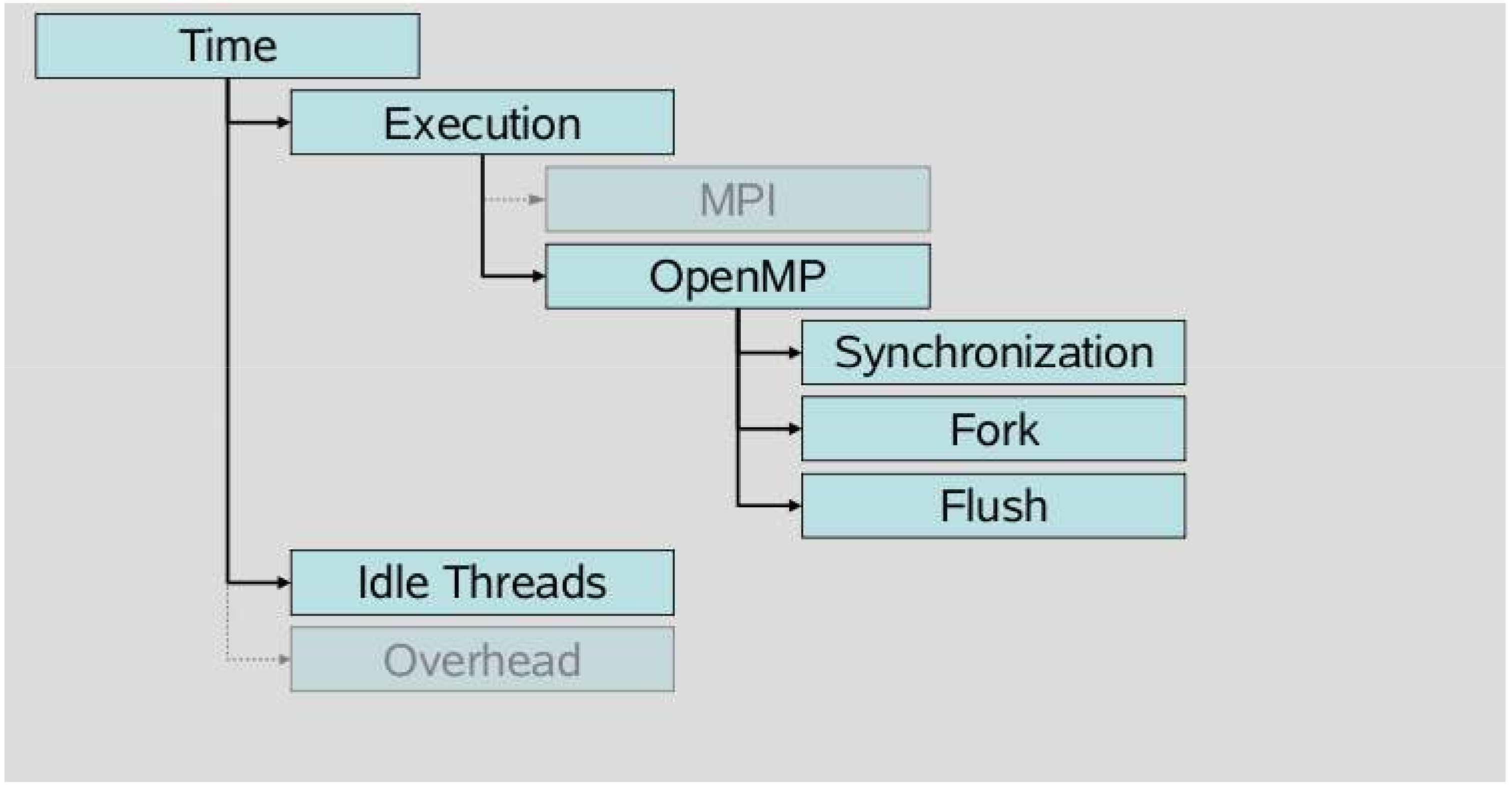


- Refers to Late Sender situations which are caused by messages received in wrong order
  - Early receive of message out of order
- Comes in two flavours:
  - Messages sent from same source location
  - Messages sent from different source locations



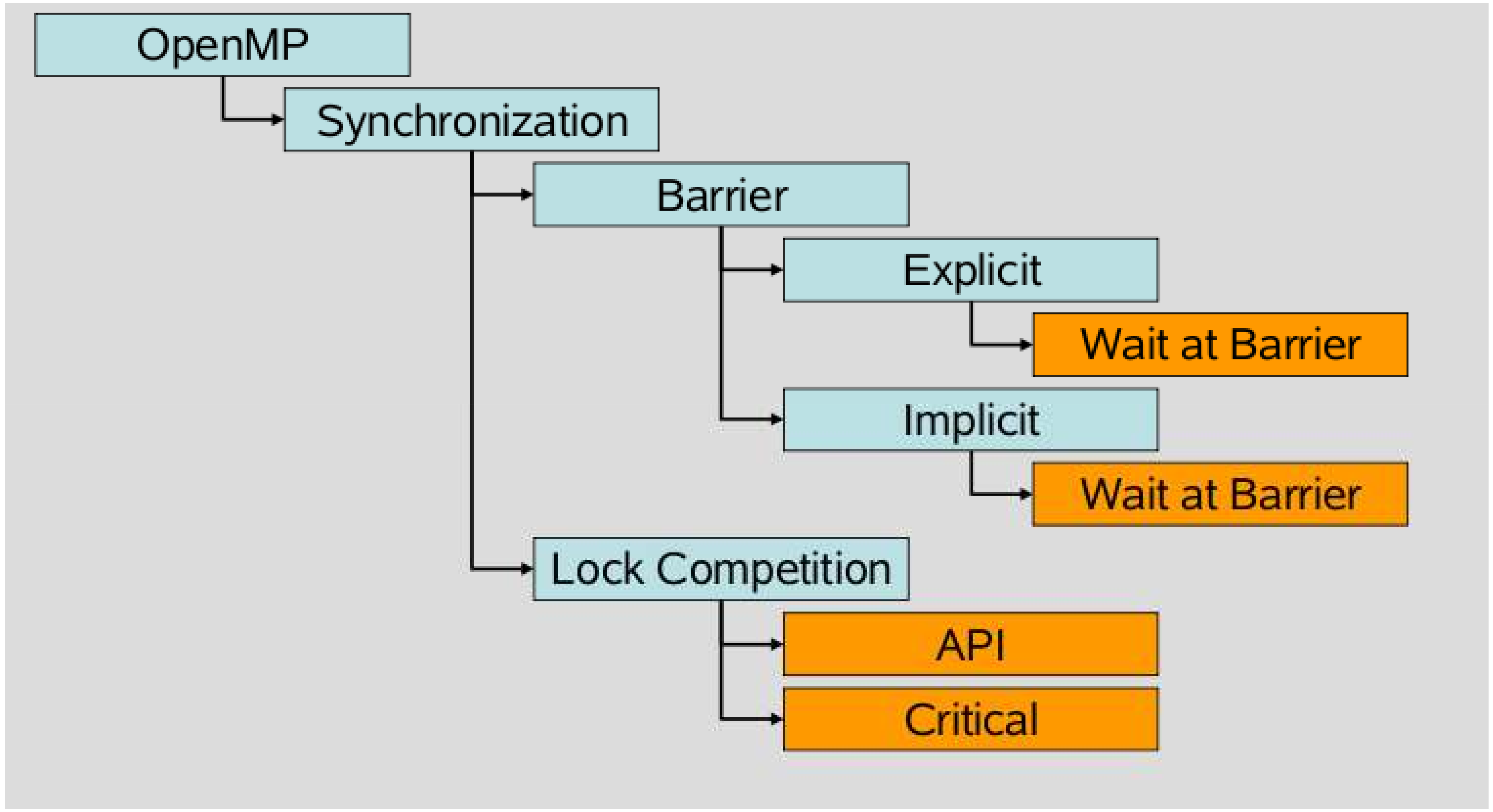
- Waiting time caused by a blocking send operation posted earlier than the corresponding receive operation
- Does not apply to non-blocking sends

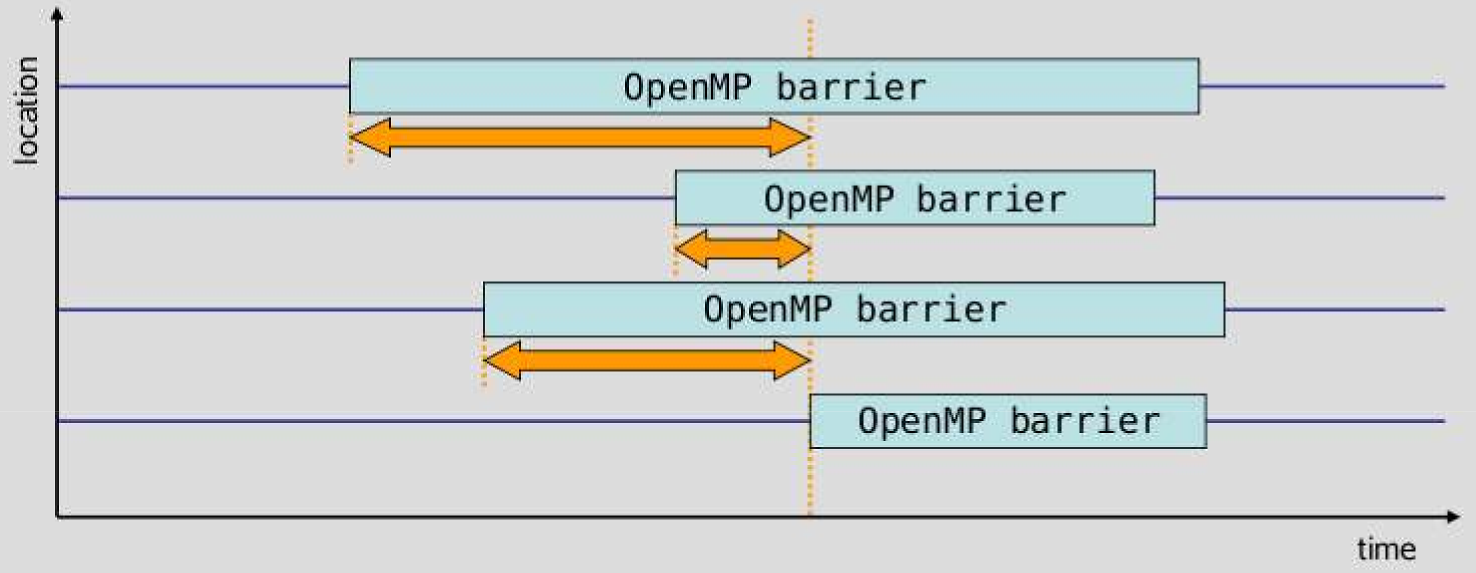
Hybrid programming MPI+OpenMP



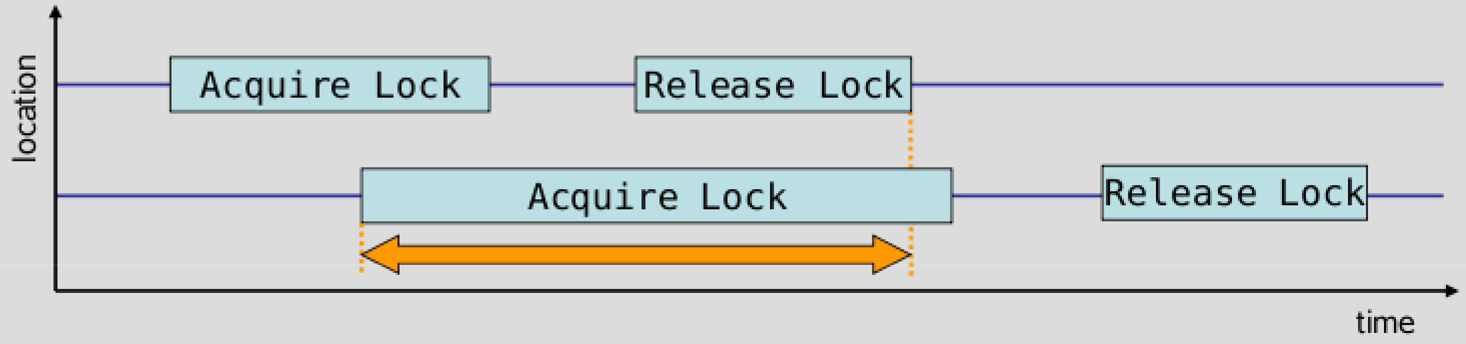
OpenMP	Time spent for all OpenMP-related tasks
Synchronization	Time spent synchronizing OpenMP threads
Fork	Time spent by master thread to create thread teams
Flush	Time spent in OpenMP flush directives
Idle Threads	Time spent idle on CPUs reserved for slave threads

Hybrid programming MPI+OpenMP





- Time threads spend waiting in front of a barrier call until the last thread reaches the barrier operation
- Applies to: Implicit/explicit barriers



- Time a thread spends waiting for a lock that is held by other threads until it is released and can be acquired by this thread
- Applies to: critical sections, OpenMP lock API

Code instrumentation

C/C++:
```
#include "epik_user.h"
...
void foo() {
   ...  // local declarations
   ...  // more declarations
   EPIK_FUNC_START();
   ...  // executable statements
   if (...)  {
     EPIK_FUNC_END();
     return;
   } else {
     EPIK_USER_REG(r_name,"region");
     EPIK_USER_START(r_name);
     ...
     EPIK_USER_END(r_name);
   }
   ...  // executable statements
   EPIK_FUNC_END();
   return;
}
```

Fortran:
```
#include "epik_user.inc"
...
subroutine bar()
   EPIK_FUNC_REG("bar")
   ...  ! local declarations
   EPIK_FUNC_START()
   ...  ! executable statements
   if (...)  then
     EPIK_FUNC_END()
     return
   else
     EPIK_USER_REG(r_name,"region")
     EPIK_USER_START(r_name)
     ...
     EPIK_USER_END(r_name)
   endif
   ...  ! executable statements
   EPIK_FUNC_END()
   return
end subroutine bar
```

C++:
```
#include "epik_user.h"
...
{
   EPIK_TRACER("name");
   ...
}
```

scalasca

- EPIK_FUNC_START, EPIK_FUNC_END mark the entry and exit from the piece of code
- The regions should be initialized with EPIK_USER_REG
- Each exit/break/continue/return must have EPIK_FUNC_END
- Need **-user** flag to decode instrumentations