

Performance Analysis of One-sided Communication Mechanisms

Bernd Mohr

Forschungszentrum Jülich (FZJ)
John von Neumann Institute of Computing (NIC)
Central Institute for Applied Mathematics (ZAM)
52425 Jülich, Germany



b.mohr@fz-juelich.de

Contents

- KOJAK introduction
- Event-based modeling of one-sided communication
 - "Generic" one-sided communication
 - Description
 - Basic event model
 - MPI-2 one-sided communication
 - Description
 - Extended event model
 - Implementation issues
- Performance properties of one-sided communication
 - SHMEM
 - MPI-2
- Conclusion

The KOJAK Project

- **K**it for **O**bjective **J**udgement and **A**utomatic **K**nowledge-based detection of bottlenecks
- Forschungszentrum Jülich
- Innovative Computing Laboratory, TN
- **Long-term goals**
 - Design and Implementation of a Portable, Generic, and Automatic Performance Analysis Environment
- **Approach**
 - Instrument C, C++, and Fortran parallel applications
 - Based on MPI, OpenMP, SHMEM, or hybrid
 - Collect event traces
 - Search trace for event patterns representing inefficiencies
- <http://www.fz-juelich.de/zam/kojak/>



KOJAK Tool Components

■ Instrument user application with EPILOG tracing library calls

□ User functions and regions:

- Automatically by TAU source instrumentor 
- Automatically by Compiler (PGI, Hitachi, NEC, Sun f90, IBM, gnu)
- Manually using POMP directives

□ MPI calls: Automatically by PMPI Wrappers (MPI 1.2 + MPI 2 RMA)

□ OpenMP: Automatically by OPARI source instrumentor

□ Record HW counter with PAPI

■ Analyze measured event trace

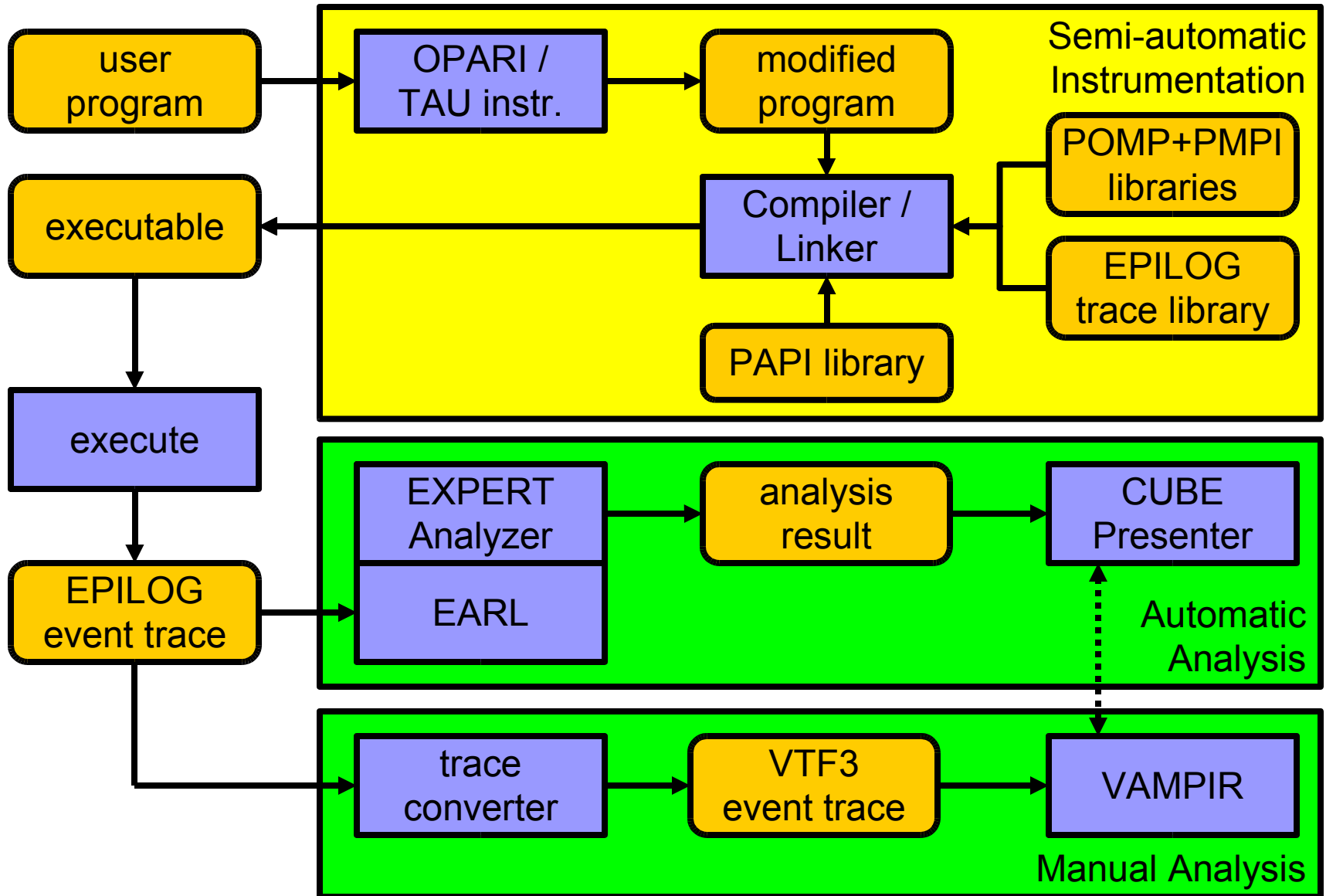
□ Automatically with EXPERT trace analyzer (based on EARL trace analysis language) and CUBE result visualizer



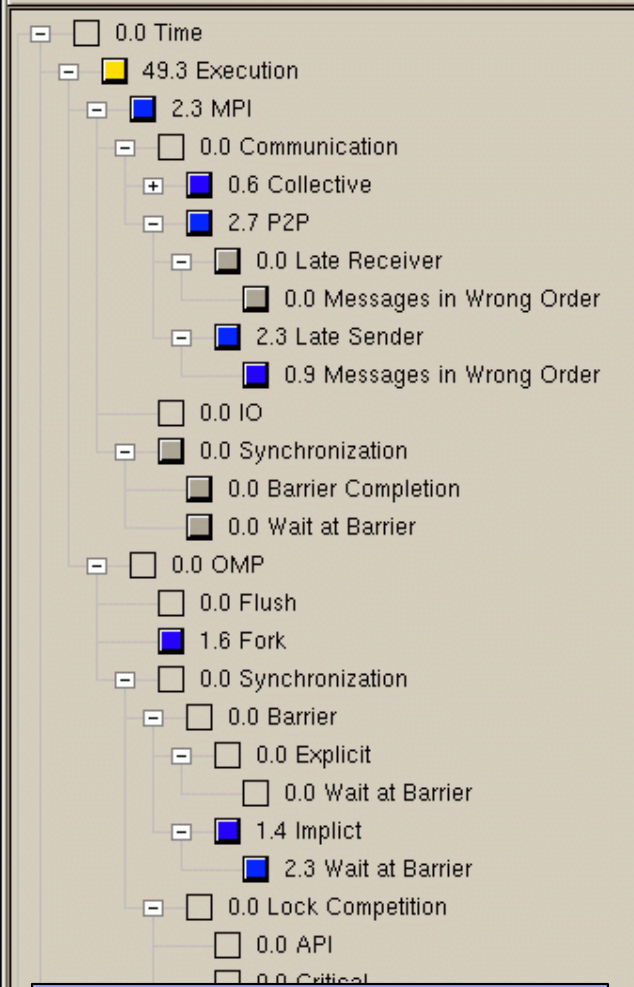
□ Manually with VAMPIR (using EPILOG-VTF3 converter)



KOJAK Architecture

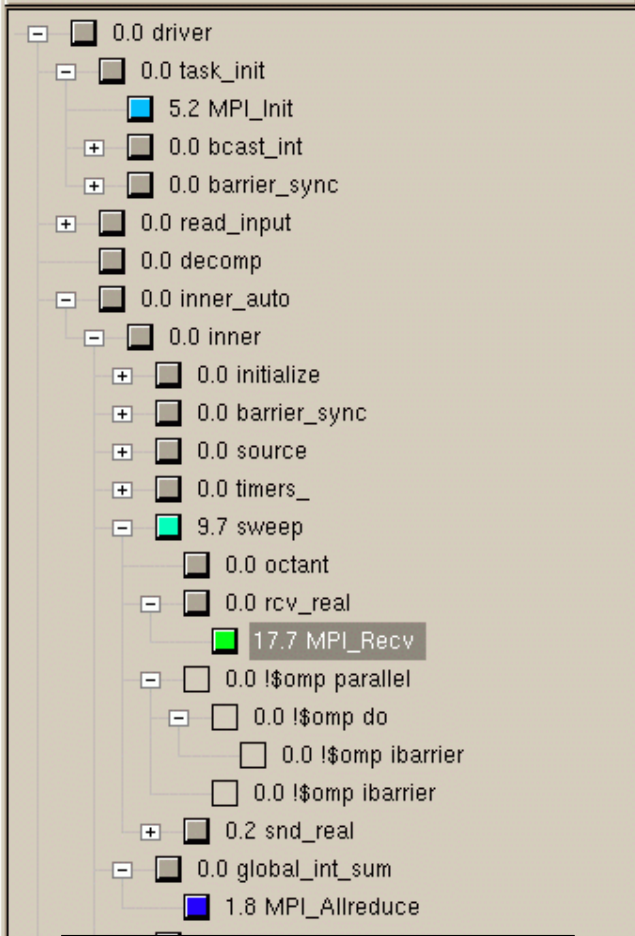


Performance Metrics



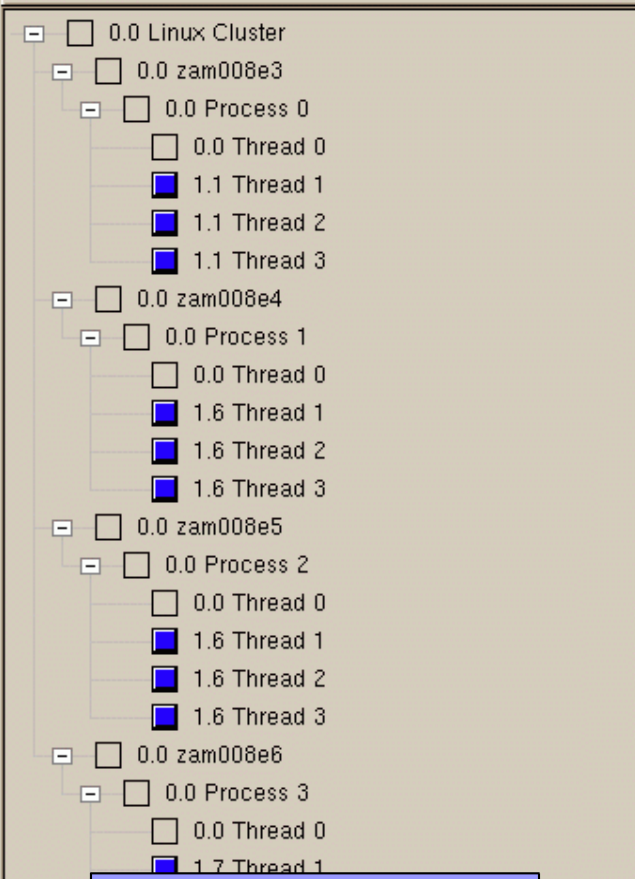
Performance Property
What problem?

Call Tree



Region Tree
Where in source code?
In what context?

System Tree



Location
How is the problem distributed
across the machine?



Color Coding How severe is the problem?

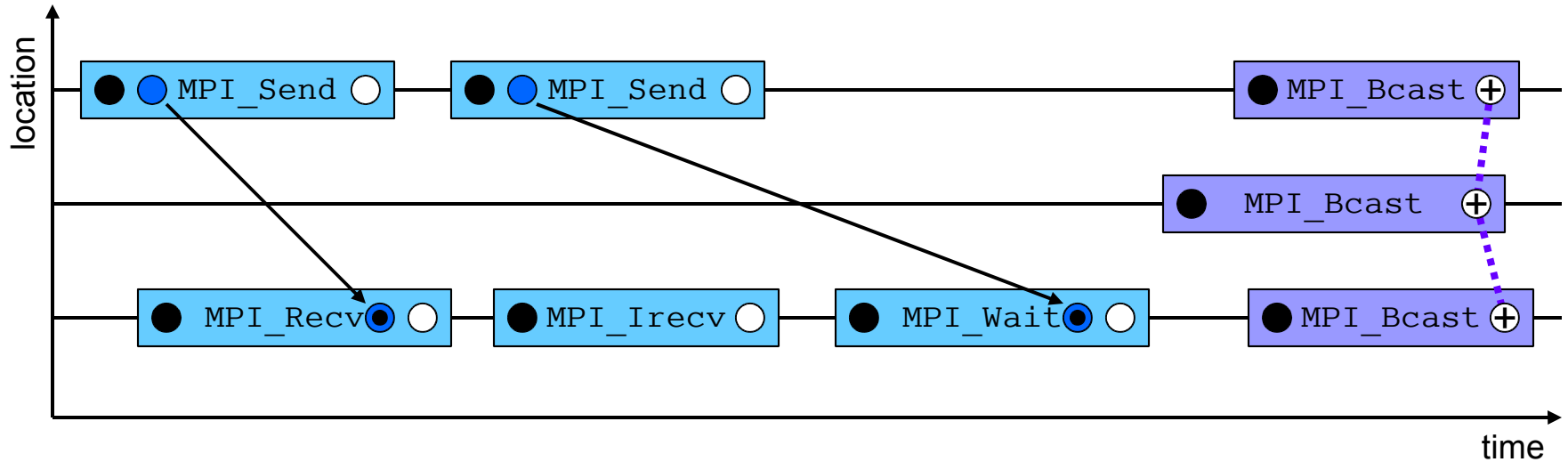
KOJAK: Supported Platforms

- **Instrumentation and measurement only**
(analysis on front-end or workstation)
 - Cray T3E, Cray XD1, Cray X1, and Cray XT3
 - IBM BlueGene/L
 - Hitachi SR-8000
 - NEC SX
- **Full support**
(instrumentation, measurement, and automatic analysis)
 - Linux IA32, IA64, and EMT64/x86_64 based clusters
 - IBM AIX Power3 and Power4 based clusters (SP2, Regatta)
 - SGI Irix MIPS based clusters (Origin 2K, Origin 3K)
 - SGI Linux IA64 based clusters (Altix)
 - SUN Solaris Sparc and x86 based clusters (SunFire, ...)
 - DEC/HP Tru64 Alpha based clusters (Alphaserver, ...)

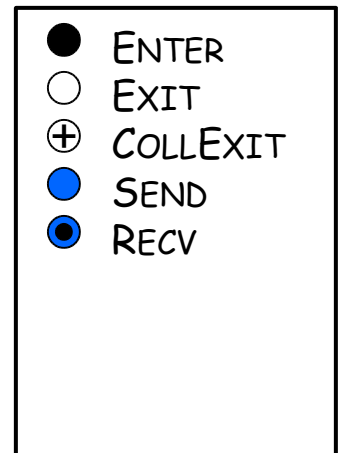
Event-based Models

- **Dynamic behavior** of application program is modeled as **stream of events**
- **Event**
 - Represents **important point during execution of program**
 - Abstraction level is determined by purpose of the analysis
 - Modeled as **tuple of event attributes**
 - Event type
 - Location (Machine, Node, Process, Thread)
 - Timestamp
 - Event type specific attributes
 - ENTER, EXIT ⇒ region
 - SEND, RECV ⇒ source/destination, length, ...
 - ...

Accepted Model: Message Passing



- ENTER/EXIT delimits regions
- ENTER/COLLEXIT delimits collective regions
- SEND/RECV delimits messages
- Model describes user-visible behavior
(\Rightarrow RECV in `MPI_Wait/_Test` for immediate communication)

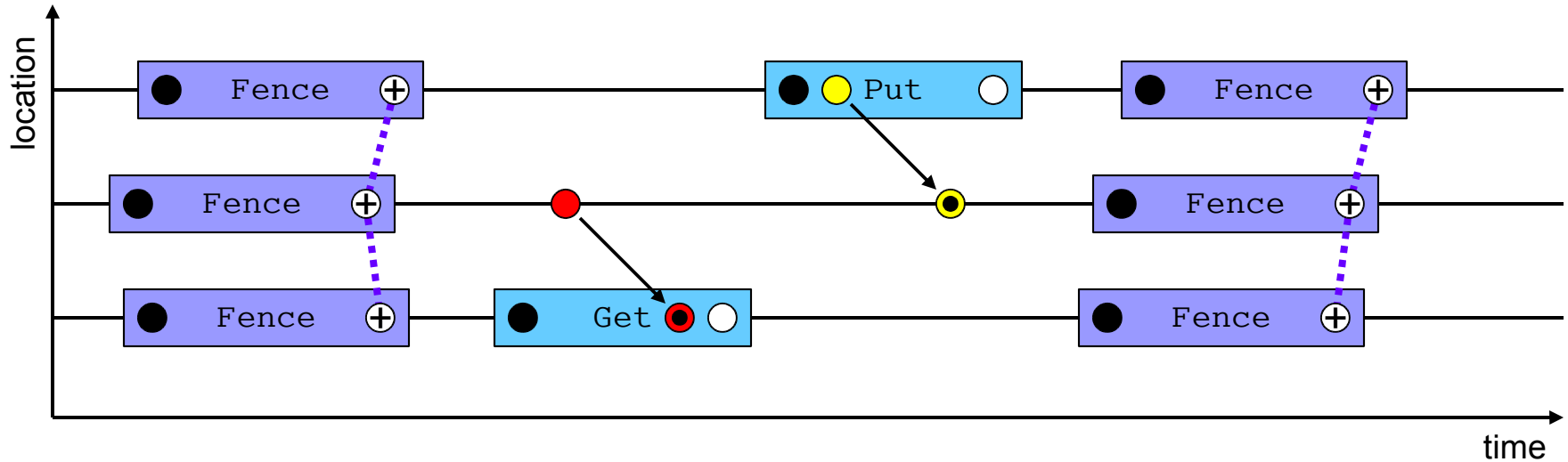


One-sided Communication

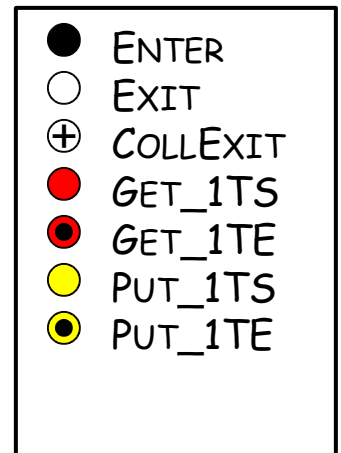
- Conventional message passing is **two-sided**:
 - Send to destination process / Receive from source process
- **One-sided** communication:
 - Parameters of data transfer are determined by one process only
 - Typically expressed through **Get** and **Put** operations
- Also called **remote memory access (RMA)**

- Two basic approaches
 - **Explicit, Library-based**
 - Vendor-specific: SHMEM (Cray/SGI), LAPI (IBM), ...
 - Portable: MPI-2
 - **Implicit, Language-based**
 - Co-Array Fortran (CAF)
 - Unified Parallel C (UPC)

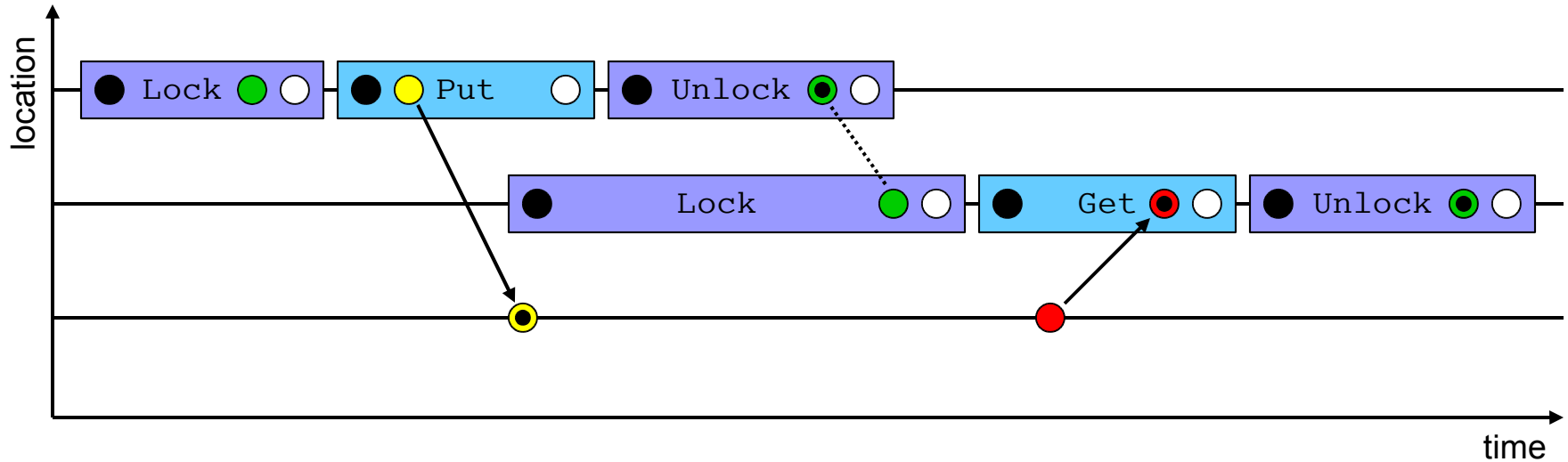
Basic Model: Fences/Barriers



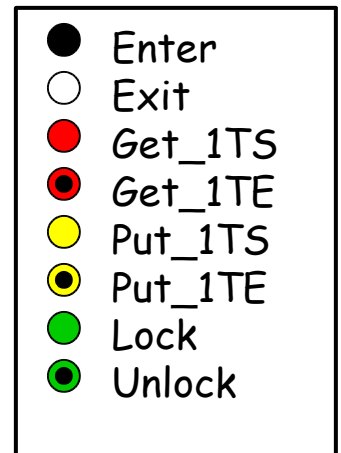
- ENTER/EXIT delimits regions
- ENTER/COLLEXIT delimits collective regions
- *GET_1TS/GET_1TE delimits remote reads*
- *PUT_1TS/PUT_1TE delimits remote writes and remote updates*



Basic Model: Locks



- ENTER/EXIT delimits regions
- GET_1TS/GET_1TE delimits remote reads
- PUT_1TS/PUT_1TE delimits remote writes and remote updates
- **LOCK and UNLOCK mark lock operations**



Basic Model: Discussion

■ Advantages

- Events and their attributes can be recorded at time/place modeled
- Straight-forward implementation
- KOJAK: used for SHMEM and CAF measurements

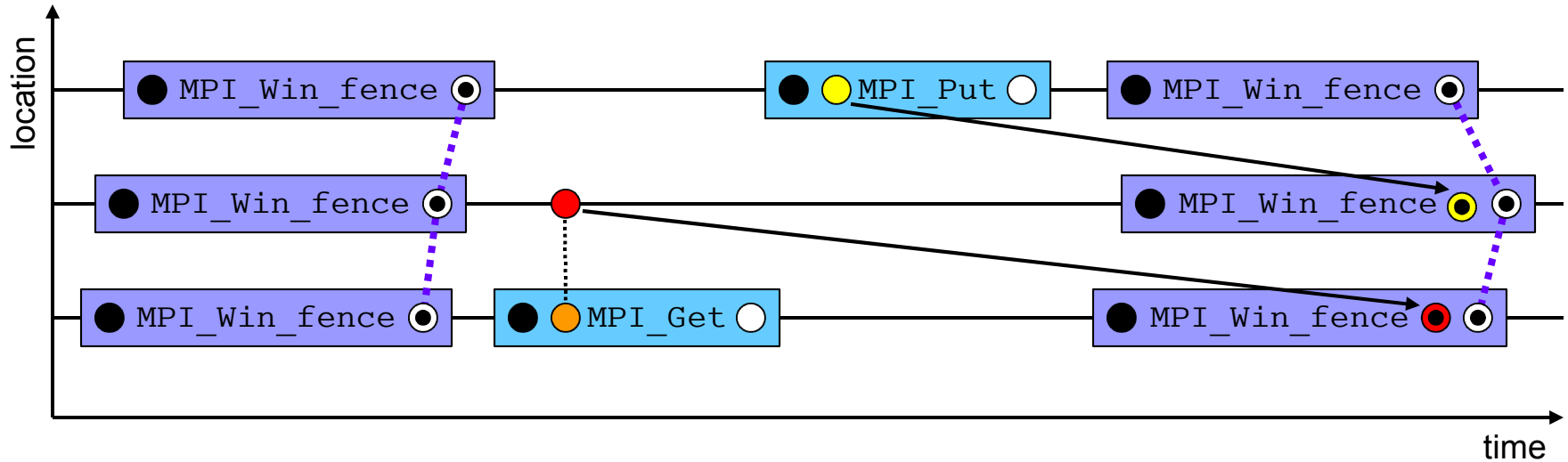
■ However for MPI-2

- Model is too simplistic
 - Model does not always reflect real behavior
 - Does not reflect user-visible behavior
- ⇒ Need better model

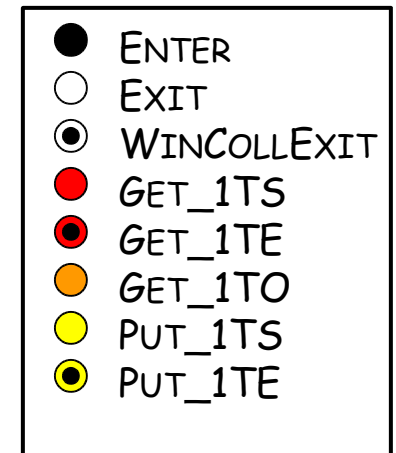
MPI-2 Remote Memory Access (RMA)

- Access only to designated parts of a process's memory: **window**
- **MPI_Get** ⇒ "Remote read" ⇒ Transfer from target to origin process
- **MPI_Put** ⇒ "Remote write" ⇒ Transfer from origin to target
- **MPI_Accumulate** ⇒ "Remote update"
- All operations **can be non-blocking**
 - ⇒ calls **can** return before actual data transfer is completed!
- Communication only completed after synchronization call on associated window
- **Active target** synchronization ⇒ two-sided!
 - (1) Collective synchronization with **fences**
 - (2) General active target synchronization (**GATS**)
- **Passive target** synchronization ⇒ really one-sided
 - (3) Synchronization through **locks**

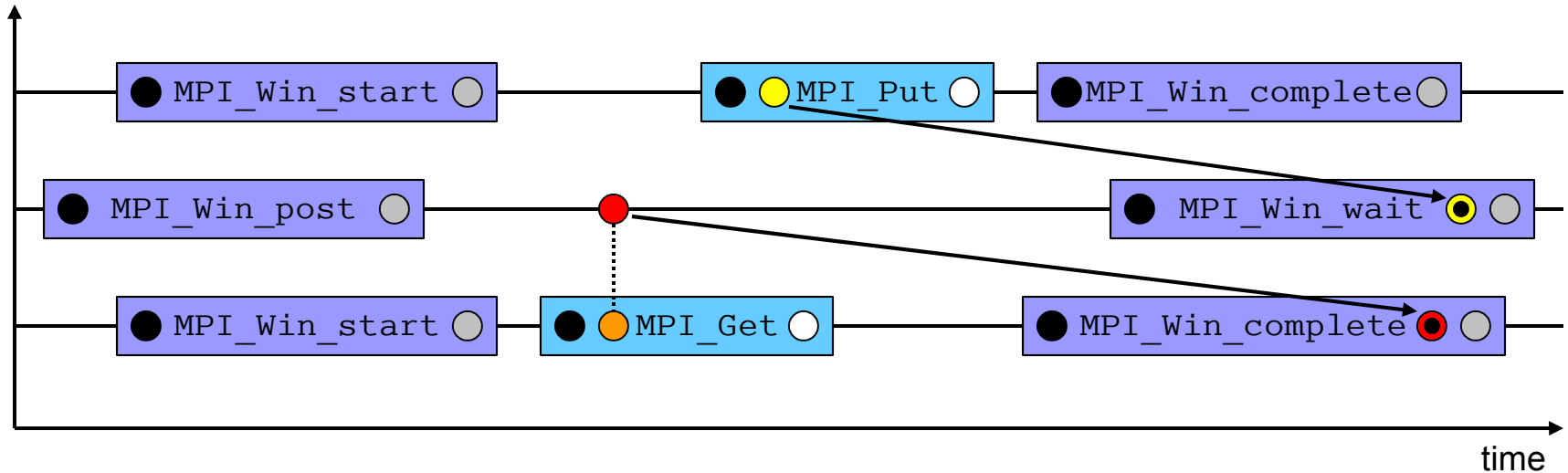
Extended Model: Fences



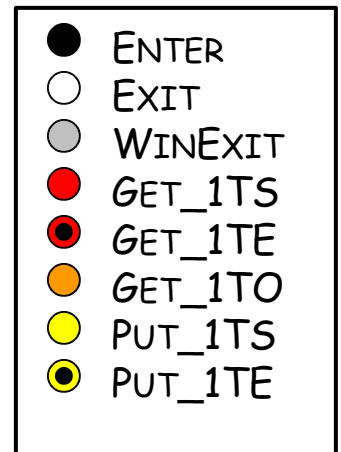
- ENTER/**WINCOLLEXIT** delimits **window** collective regions
- **Transfer end events move to corresponding synchronization operation**
- **GET_1TO** mark origin of remote read



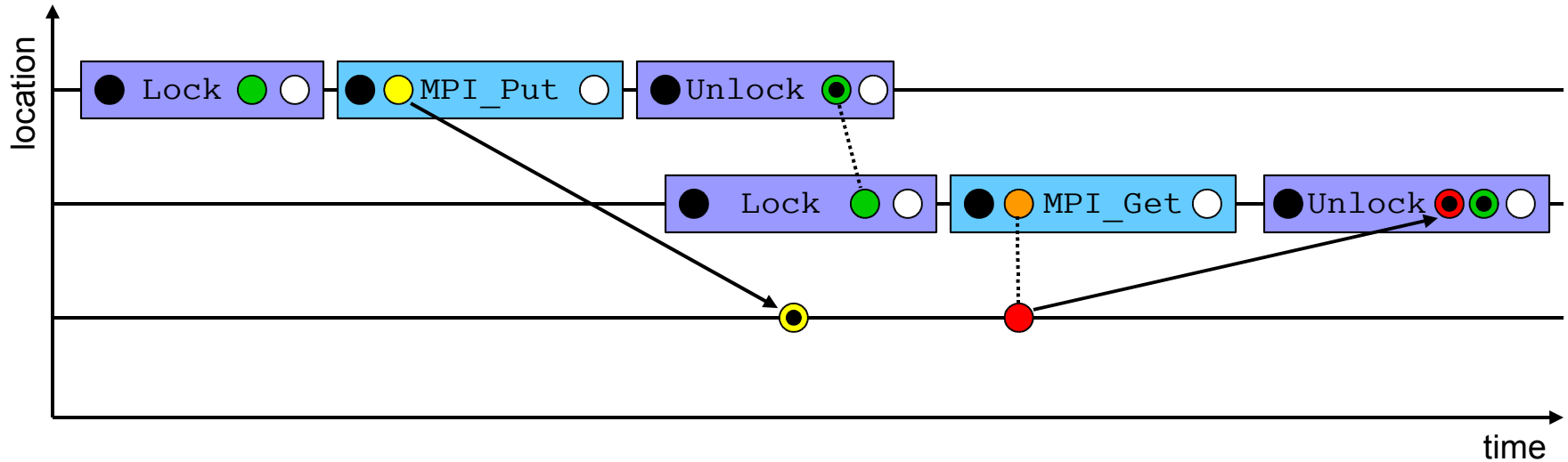
Extended Model: GATS



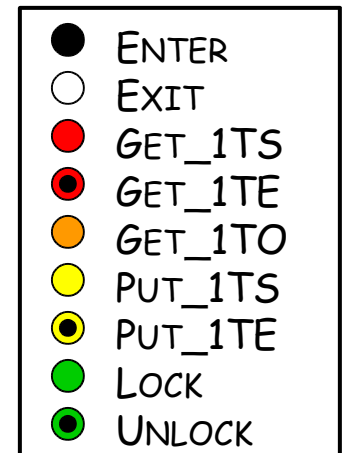
- ENTER/**WINEXIT** delimits **window group** collective regions
- Transfer end events move to corresponding synchronization operation
- GET_1TO mark origin of remote read



Extended Model: Locks



- Transfer end events move to corresponding synchronization operation
- GET_1TO mark origin of remote read



Extended Model: Discussion

■ Advantages

- Takes MPI-2 semantics into account
- Correctly **models user-visible behavior**
- For most MPI implementations (non-blocking) **closer to reality**

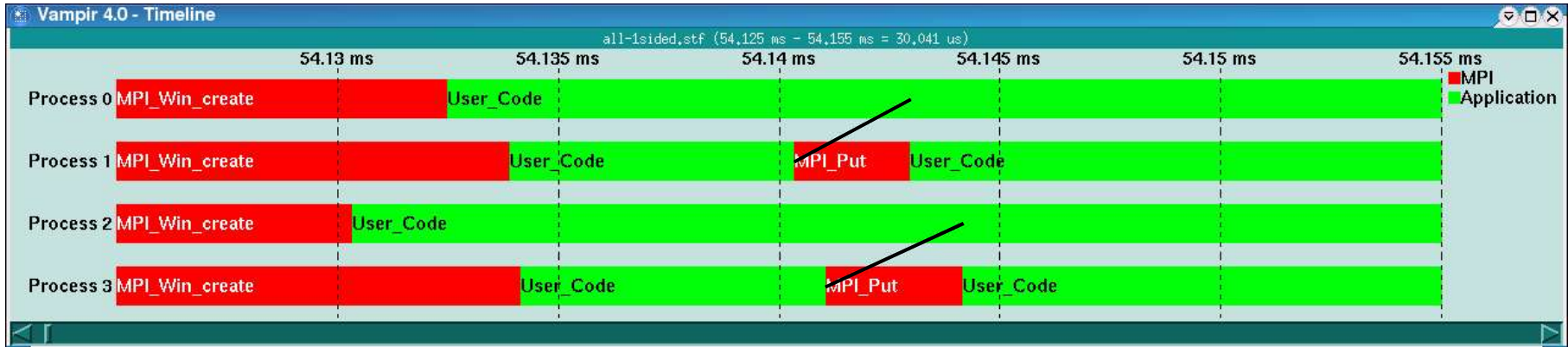
■ Disadvantages

- Events cannot be recorded at location where prescribed by model
 - ⇒ **Complex post-processing needed**
- GET: data transfer no longer "connected" with get operation
 - ⇒ Introduction of `GET_1TO` event needed
 - ⇒ Visualization?!

Implementation Issues

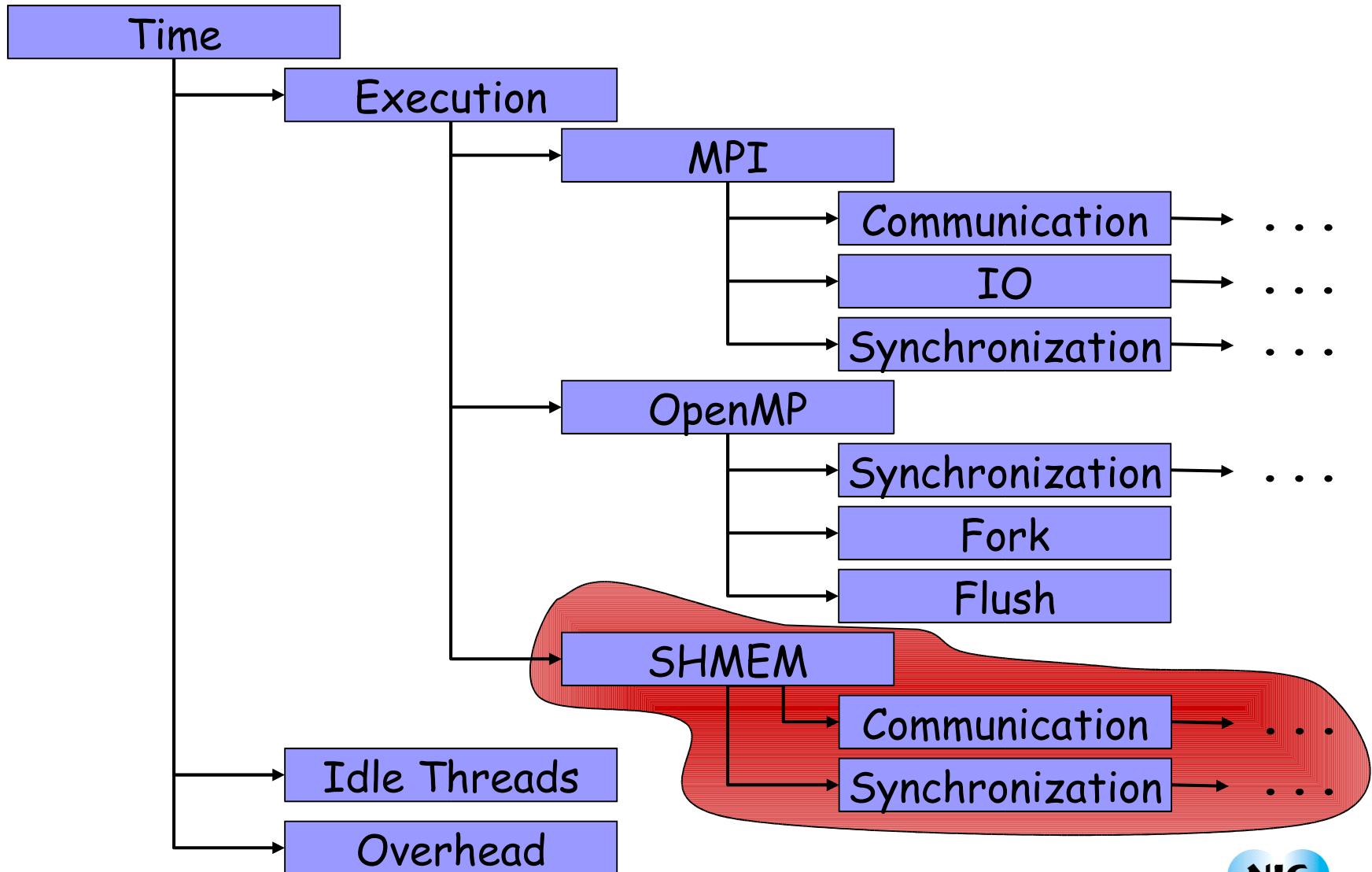
- PMPI wrapper ⇒ **process-local measurement**
 - Generate temporary **REM_GET_1TS** and **REM_PUT_1TE events**
 - Replaced by “real” events during global trace merge
 - Merge also generates **GET_1TO** events
- For MPI-2, **merge also adjusts timestamps of transfer end events**
 - Queues transfer end events per window and location
 - Tracks windows and access and exposure epochs to locate positions where transfers are completed
 - De-queues corresponding transfer end events with corrected timestamps
- Done in merge phase ⇒ very low overhead during measurement!

GATS recorded by Vampirtrace (4.0)

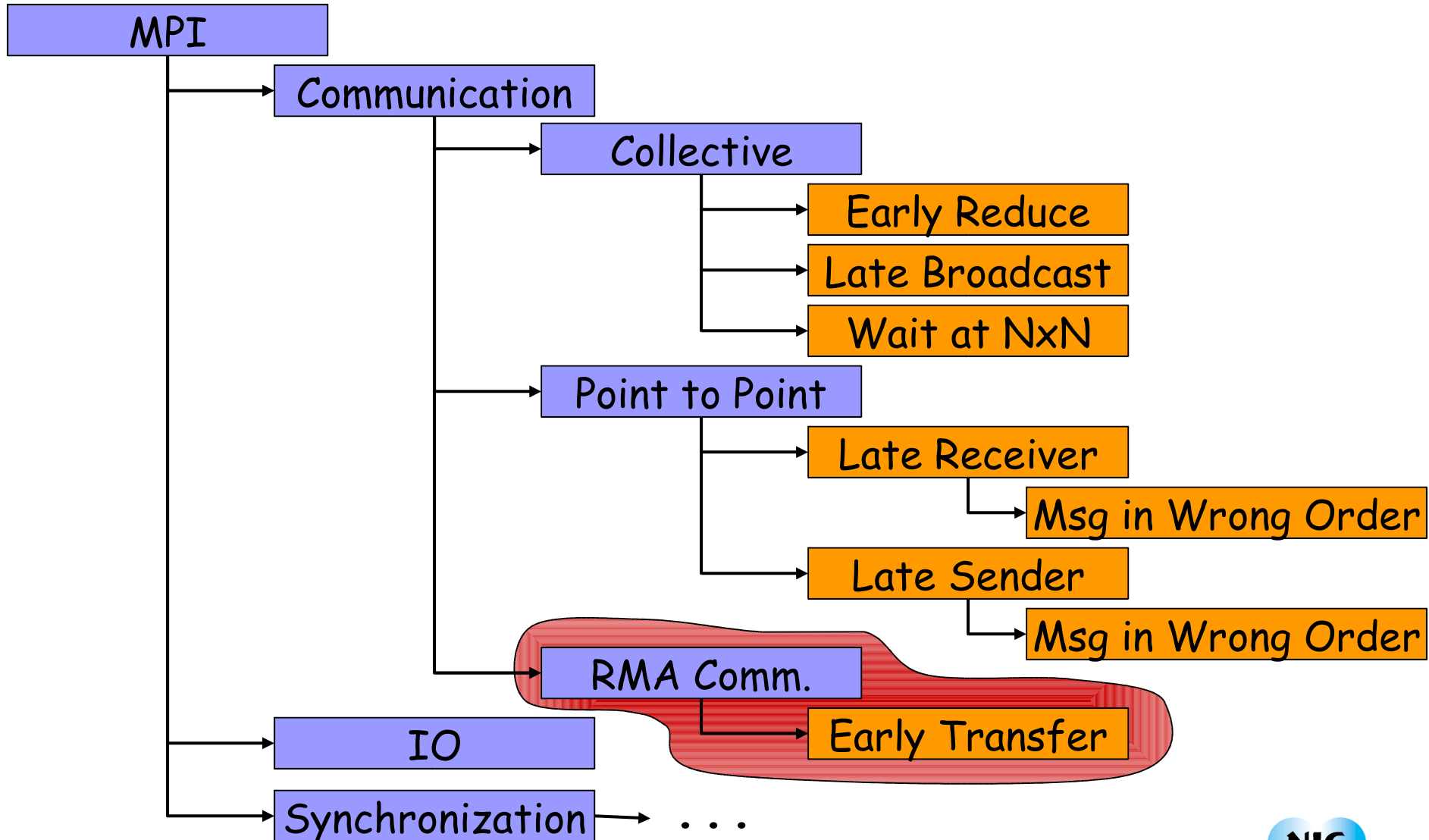


- Data transfer does not reflect user-visible behavior
- No recording of MPI GATS synchronization functions
- Window creation and release not marked as collective operation

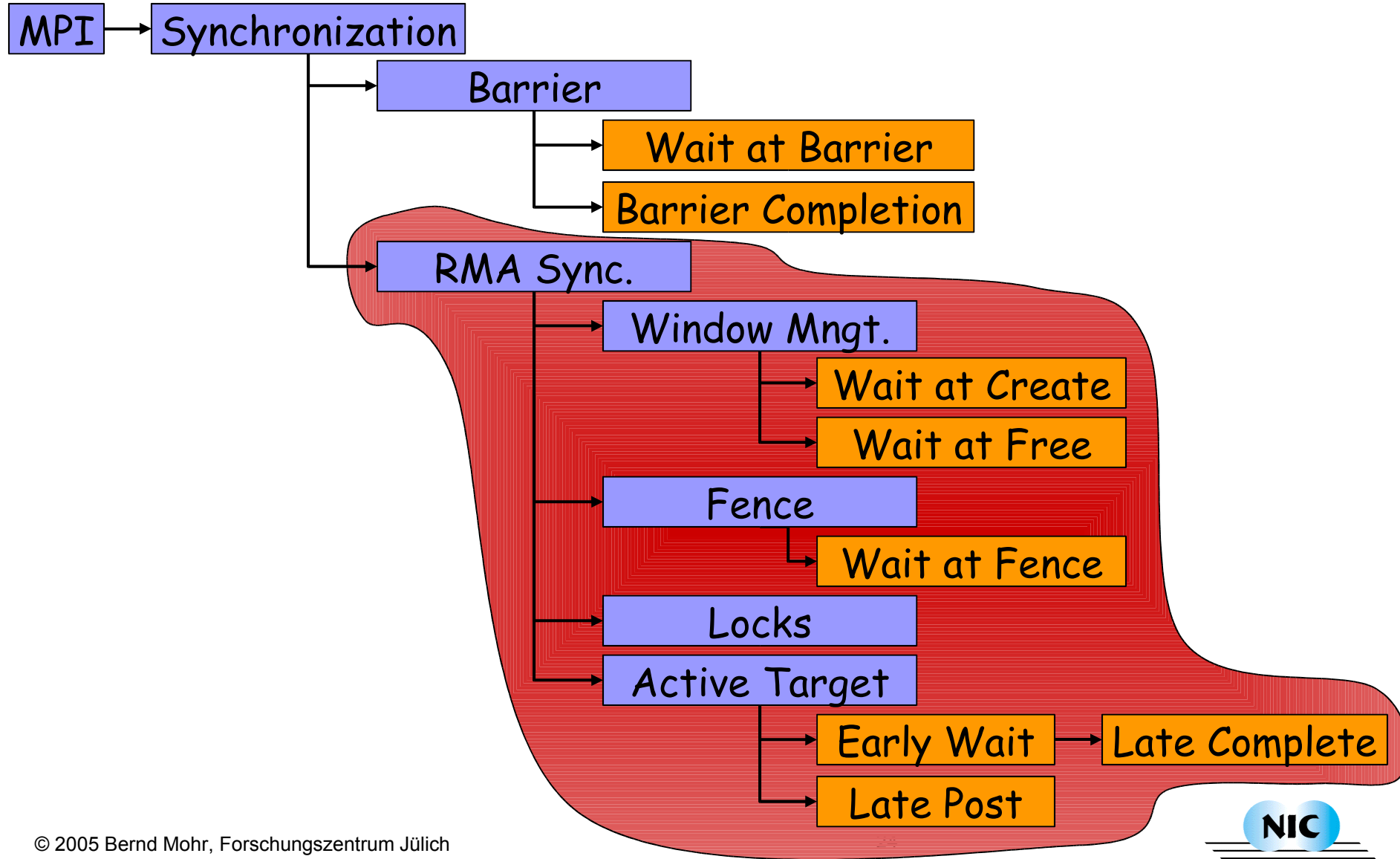
KOJAK: Basic Pattern Hierarchy



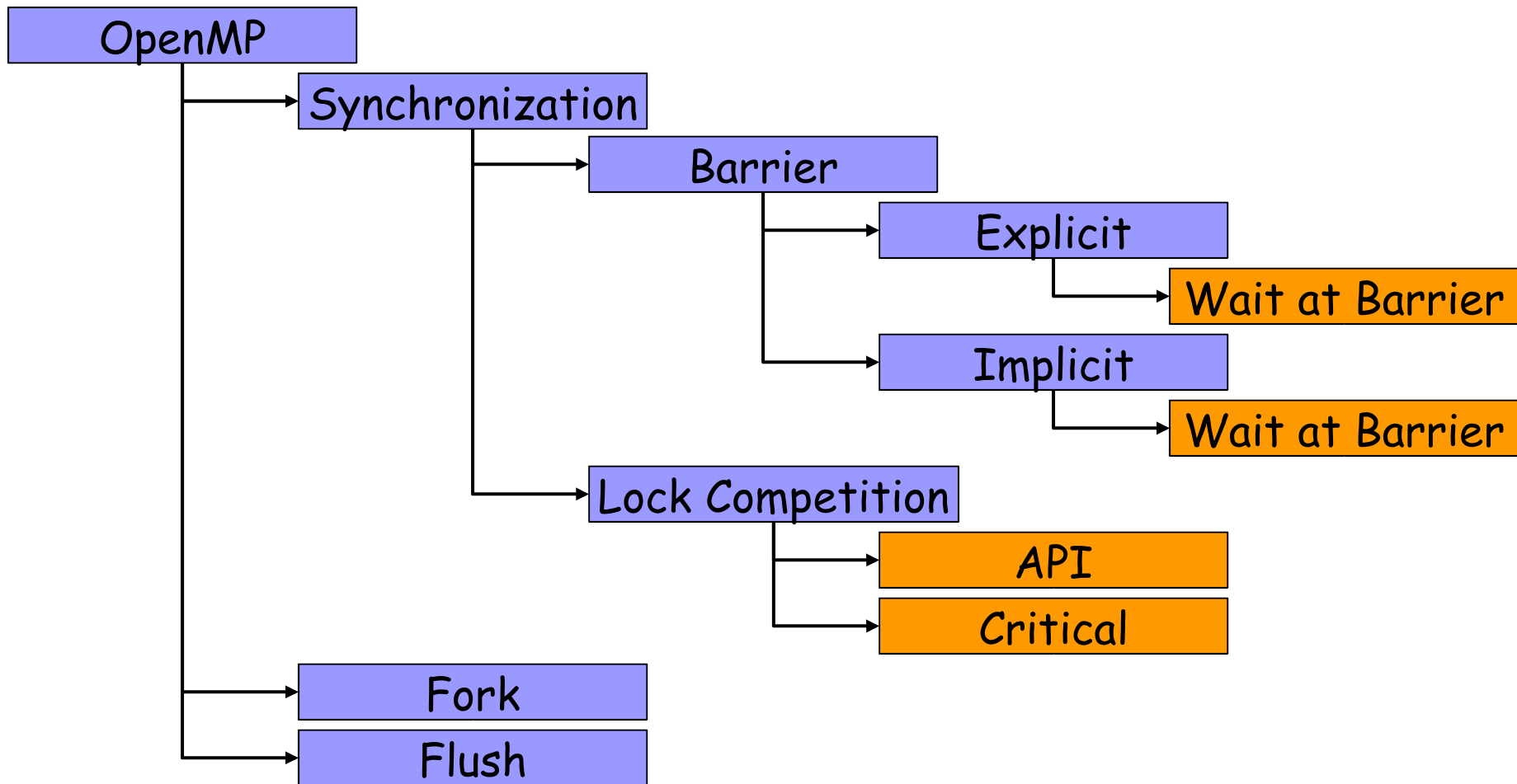
KOJAK: MPI Pattern Hierarchy I



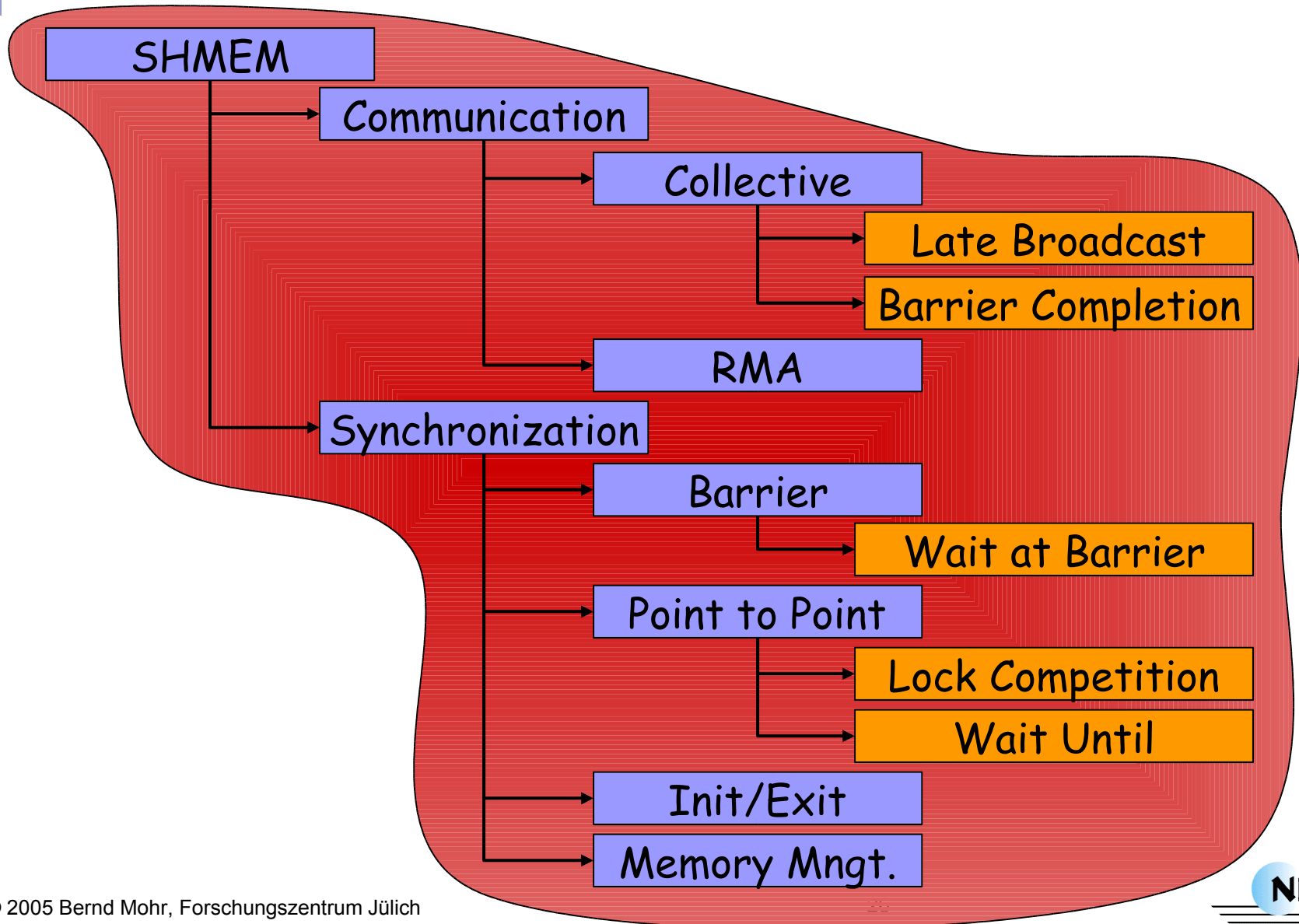
KOJAK: MPI Pattern Hierarchy II



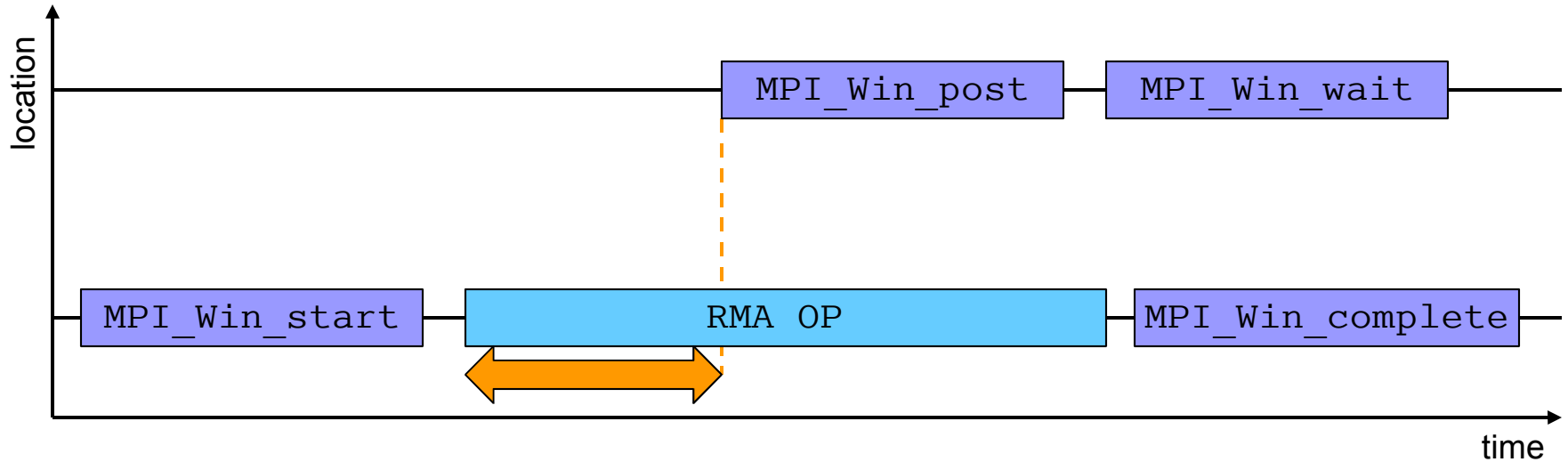
KOJAK: OpenMP Pattern Hierarchy



KOJAK: SHMEM Pattern Hierarchy

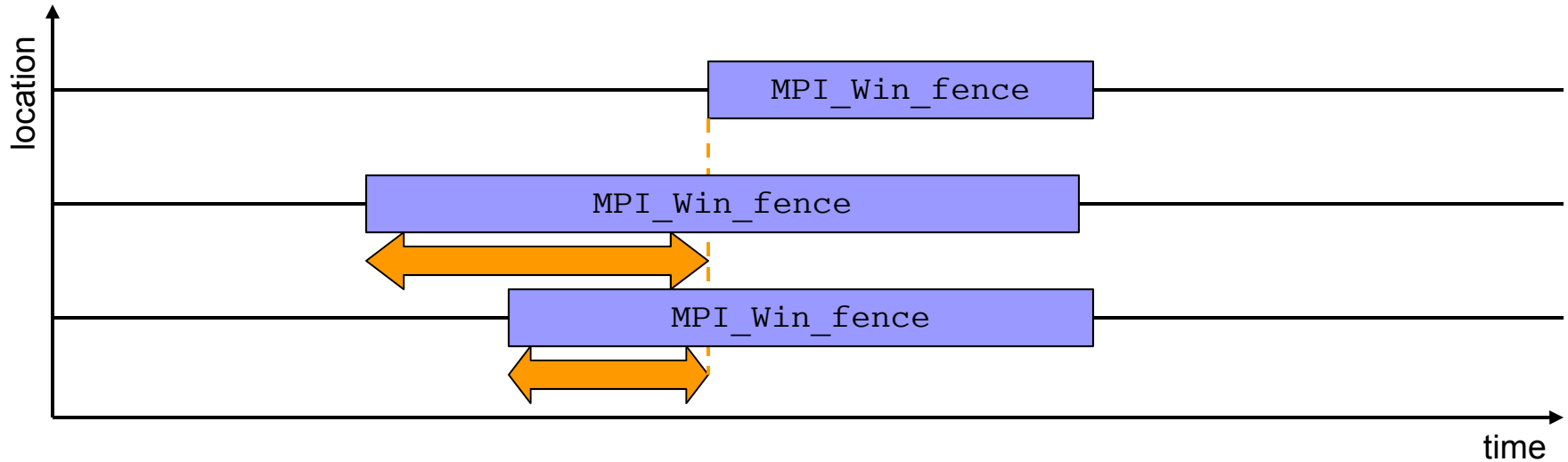


MPI-2 Pattern: Early Transfer



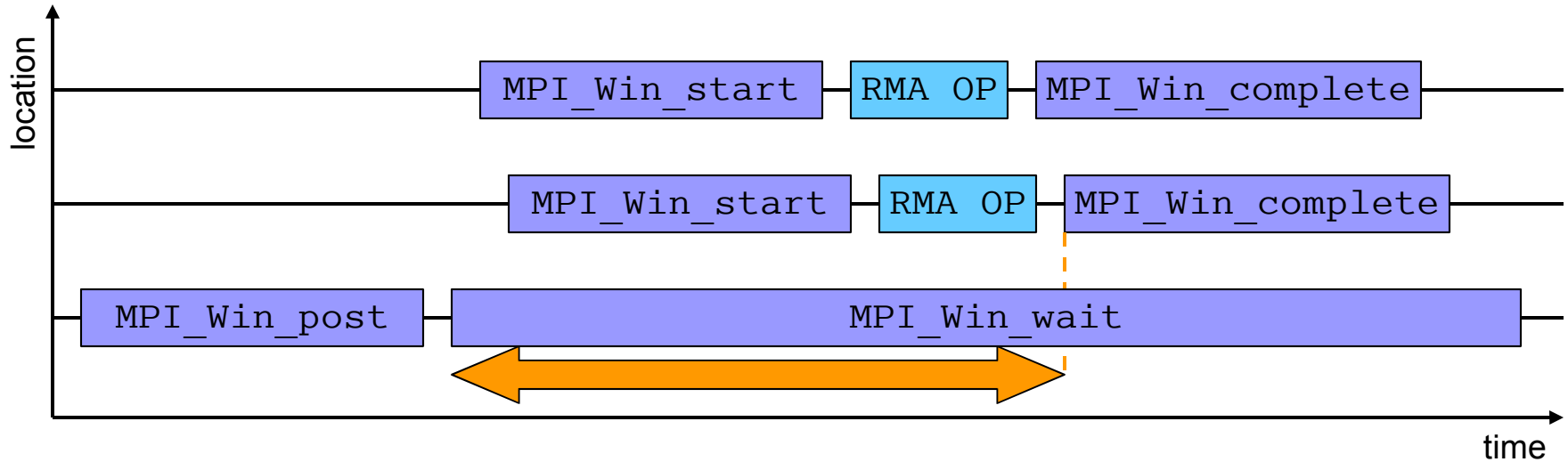
- RMA operation is blocked until exposure epoch is opened with `MPI_Win_post`

MPI-2 Pattern: Wait at Fence, ...



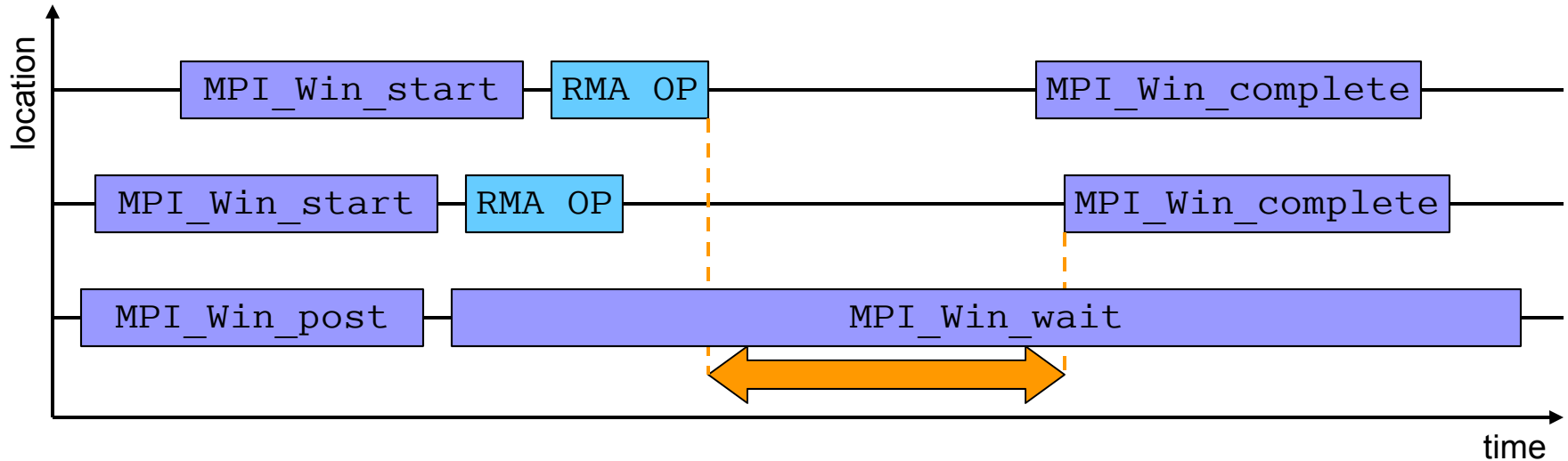
- Time spent waiting in front of a collective MPI_Win_fence call until the last process reaches the fence
- Similar patterns also for
 - Wait at Create (MPI)
 - Wait at Free (MPI)
 - Wait at Barrier (SHMEM)
 - Wait at NxN (SHMEM)

MPI-2 Pattern: Early Wait



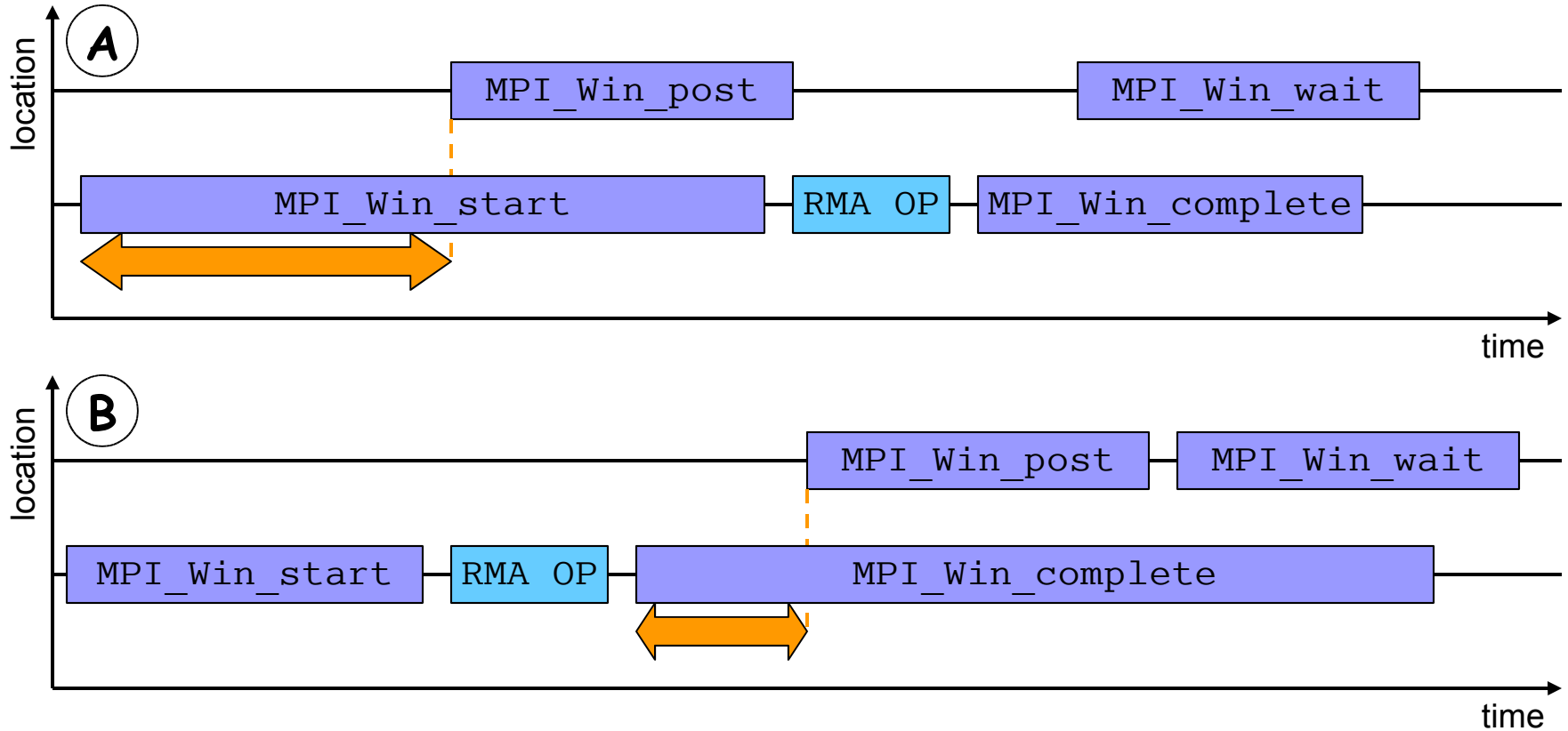
- MPI_Win_wait blocks until access epoch is closed by last MPI_Win_complete

MPI-2 Pattern: Late Complete



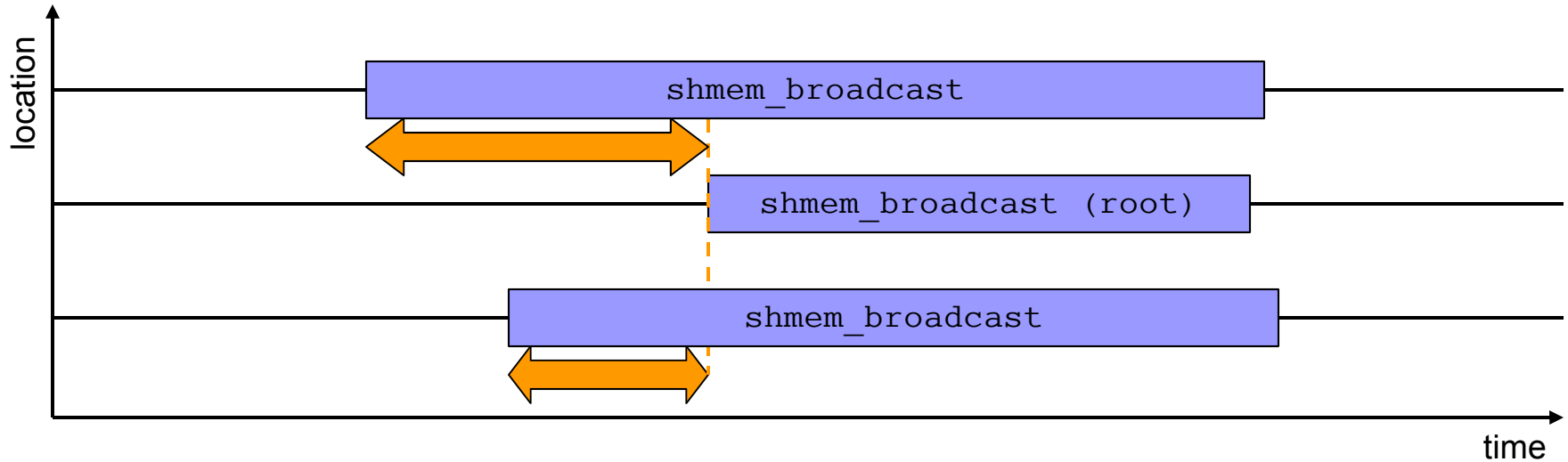
- MPI_Win_wait blocks until access epoch is closed by last MPI_Win_complete
 - Sub pattern of Early Wait
 - But additional waiting time because of unnecessary pause between last RMA operation and last MPI_Win_complete

MPI-2 Pattern: Late Post



- MPI_Win_start (variant A) or MPI_Win_complete (variant B) block until exposure epoch is opened with MPI_Win_post

SHMEM Pattern: Late Broadcast



- Root process of broadcast operation comes too late

Conclusion

- **Development of event model** which can correctly represent MPI-2 and non-MPI one-sided communication and synchronization
- **Definition of performance properties** for one-sided communication
- **Prototype implementation** of the necessary KOJAK components for
 - Instrumentation
 - Measurement
 - Conversion to VTF3
 - Automatic analysis (EXPERT)for MPI-2 and SHMEM one-sided communication and synchronization
- **Incomplete prototype implementation** of Co-Array Fortran support



Thank you!



MPI-2 Extra Slides

Bernd Mohr

Forschungszentrum Jülich (FZJ)
John von Neumann Institute of Computing (NIC)
Central Institute for Applied Mathematics (ZAM)
52425 Jülich, Germany

b.mohr@fz-juelich.de

MPI-2 Remote Memory Access (RMA)

- **Design goal:** allow efficient implementation even on platforms which do not provide direct hardware support for remote memory access
 - Correct ordering of accesses has to be specified by user with explicit synchronization calls
 - Implementation can delay actual data transfer until synchronization calls for efficiency
 - Access only to designated parts of a process's memory: **window**
 - Explicitly initialized (`MPI_Win_create`) and released (`MPI_Win_free`)
 - Collective between all participating partners
- **Target processes** provide window
- **Origin processes** read data from or writes data to window

MPI-2 RMA: Communication Calls

- `MPI_Get` ⇒ "Remote read"
 - Data transfer from target (source) to origin process (destination)
- `MPI_Put` ⇒ "Remote write"
 - Data transfer from origin (source) to target process (destination)
- `MPI_Accumulate` ⇒ "Remote update"
 - Data at target is replaced by result of binary reduction operation on local and remote data
- All operations **can be non-blocking**
 - ⇒ calls **can** return before actual data transfer is completed!
- Communication only completed after synchronization call on associated window

MPI-2 RMA: Synchronization Methods

- **Active target** synchronization

- Origin and target processes participate in the synchronization
⇒ two-sided!

- (1) Collective synchronization with **fences**

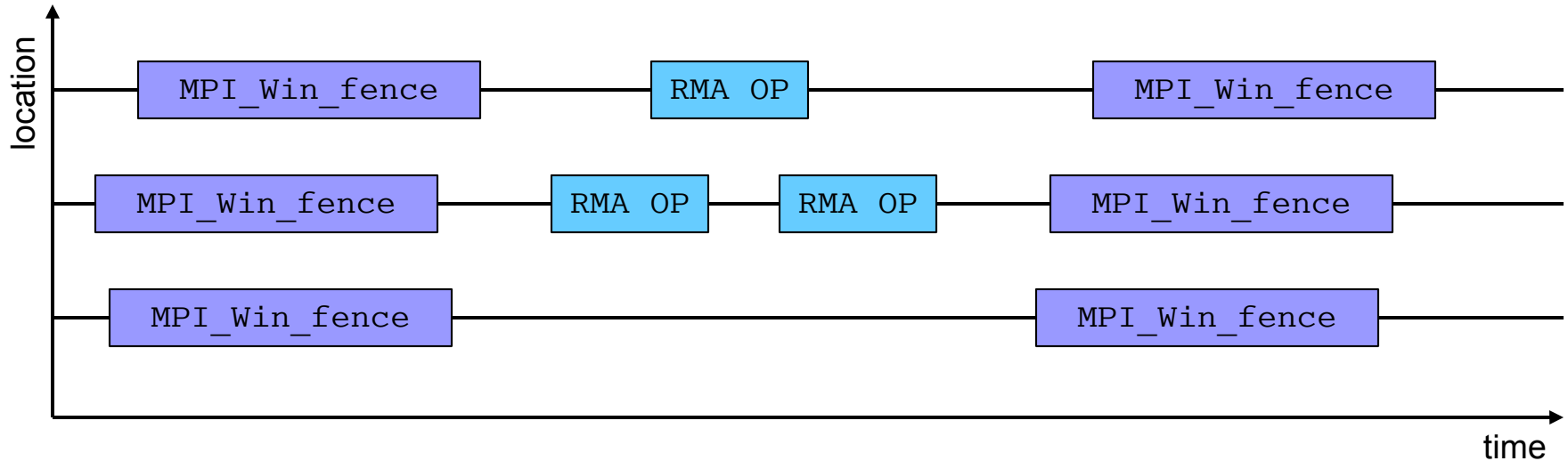
- (2) General active target synchronization (**GATS**)

- **Passive target** synchronization

- Only origin processes participate in the synchronization
⇒ Really one-sided

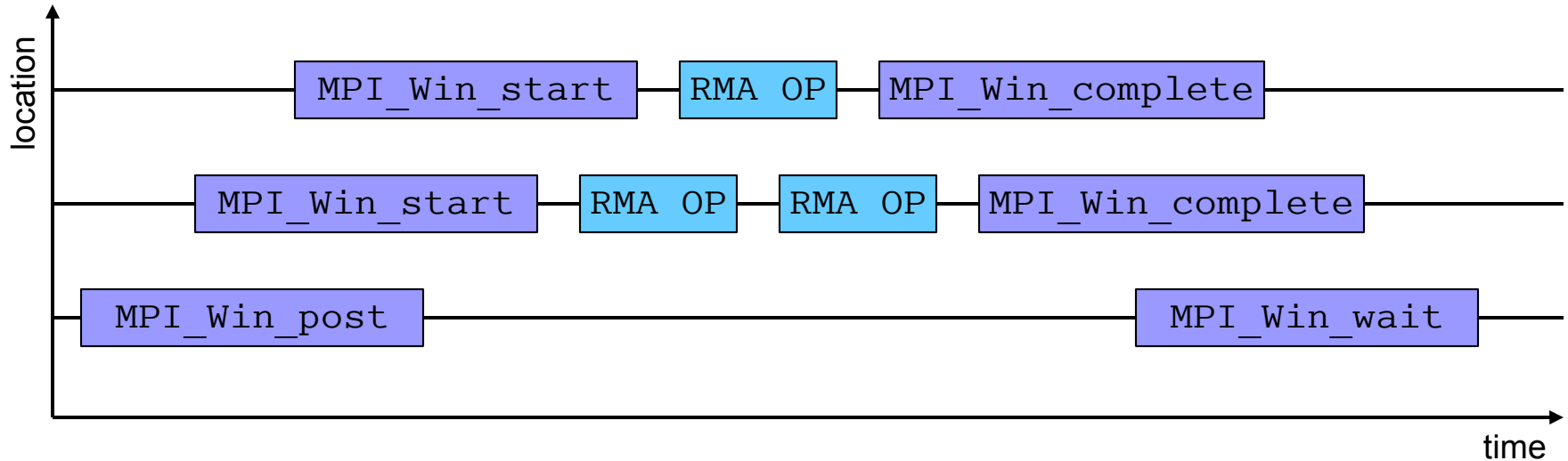
- (3) Synchronization through **locks**

Collective Synchronization with Fences



- Series of RMA operations delimited by MPI_Win_fence calls
- **Collective call** with implicit **barrier**
- Executed by all processes which defined window
- Data transferred only accessible to user code after fence

General Active Target Synchronization

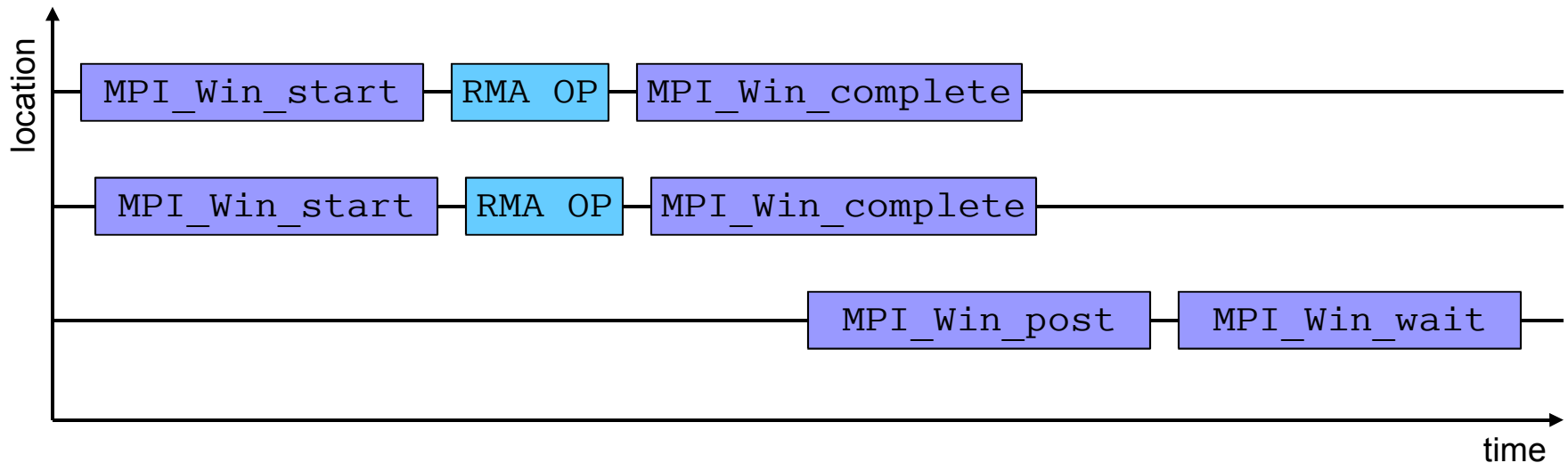


- Synchronization between **subgroups** of window communicator
- `MPI_Win_post` and `MPI_Win_wait` on target process define **exposure epoch**
- `MPI_Win_start` and `MPI_Win_complete` on origin processes delimit **access epoch**
- Data transferred only accessible to user code after end of corresponding epoch

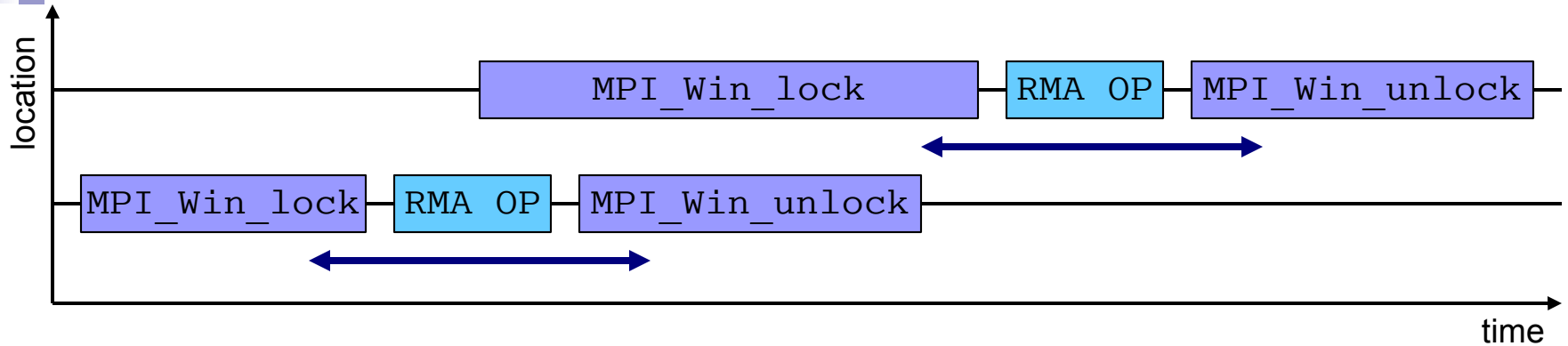
General Active Target Synchronization

■ However

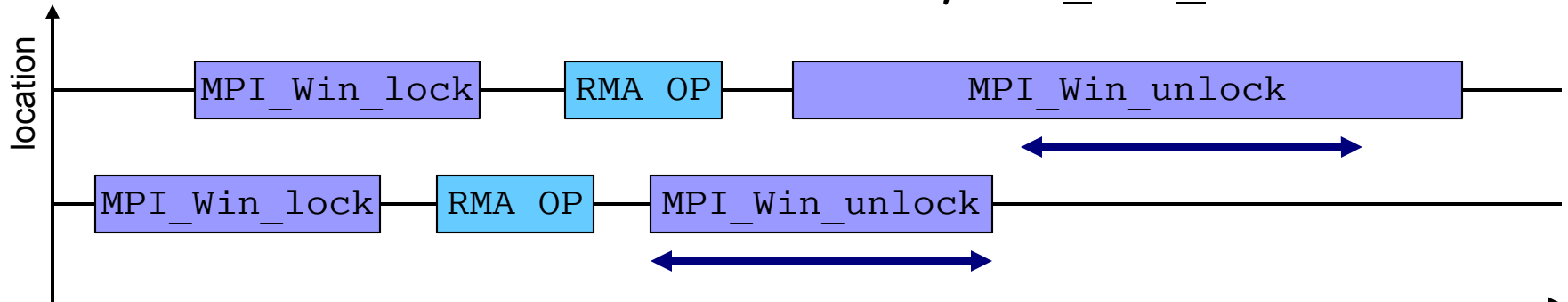
- MPI_Win_start need not wait for MPI_Win_post
- Up to one access epoch might be buffered by the system
- Exposure epoch might come after corresponding access epoch



Passive Target Synchronization



- Shared and exclusive locks through MPI_Win_lock / _unlock
- Data transferred only accessible to user code after unlock
- However, lock call **can be non-blocking!!**
 - ⇒ Lock, transfer, unlock all handled by MPI_Win_unlock





CAF Extra Slides

Bernd Mohr

Forschungszentrum Jülich (FZJ)
John von Neumann Institute of Computing (NIC)
Central Institute for Applied Mathematics (ZAM)
52425 Jülich, Germany

b.mohr@fz-juelich.de

Our Approach to Performance Analysis of CAF

- **Extension of KOJAK framework**
 - Minimize work / maximize re-use
 - Allows for analysis of hybrid MPI/OpenMP/SHMEM/CAF programs
- Extension of KOJAK's **event model** to support CAF
- Definition of **PCAF**: an open, portable CAF monitoring interface
- Extension of **OPARI source code instrumentor** to handle CAF constructs
 - ⇒ Allows for re-use of infrastructure for other projects
- Performance **Visualization with VAMPIR**
 - Based on extended EPILOG to VTF3 converter

CAF Instrumentation Example

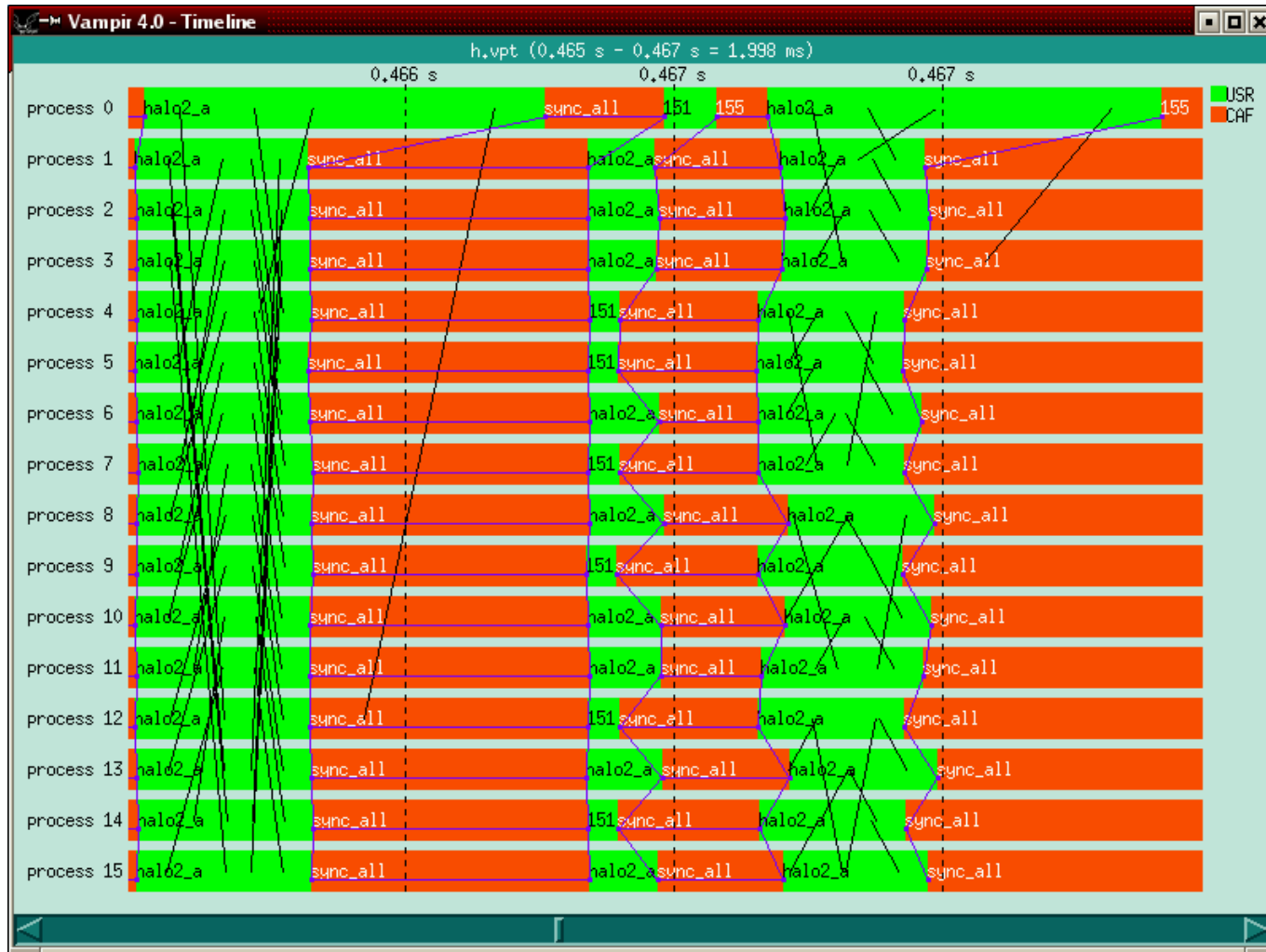
```
integer :: me, num, left
integer :: A(1024, 1024) [*]

me      = this_image()
num     = num_images()
left    = me - 1
if ( left < 1 ) left = num

call PCAF_rma_write_begin(-1+left, &
    1 * max((ubound(A,2)-lbound(A,2)+2/2,0))
A(me,::2)[left] = me
Call PCAF_rma_write_end(-1+left)

Call PCAF_sync_all()
```

Visualization: 1 Iteration, sync_all()



Visualization: 1 Iteration, sync_team(wait)

