scalasca

Trace-based detection of lock contention in MPI one-sided communication

Marc-André Hermanns Bernd Mohr Felix Wolf

October 4, 2016 — Parallel Tools Workshop, Stuttgart

JÜLICH FORSCHUNGSZENTRUM

German Research School for Simulation Sciences

Motivation

- The Message Passing Interface (MPI) standard
  - De-facto distributed-memory programming standard in HPC
  - Defines multiple communication paradigms
- MPI one-sided communication not often used (yet)
  - Initial interface not well adopted by users
  - Gaining traction since MPI-3
- Tool support for one-sided communication is narrow
  - Crucial for understanding of complex synchronization behavior
  - Required for supporting multi-paradigm applications

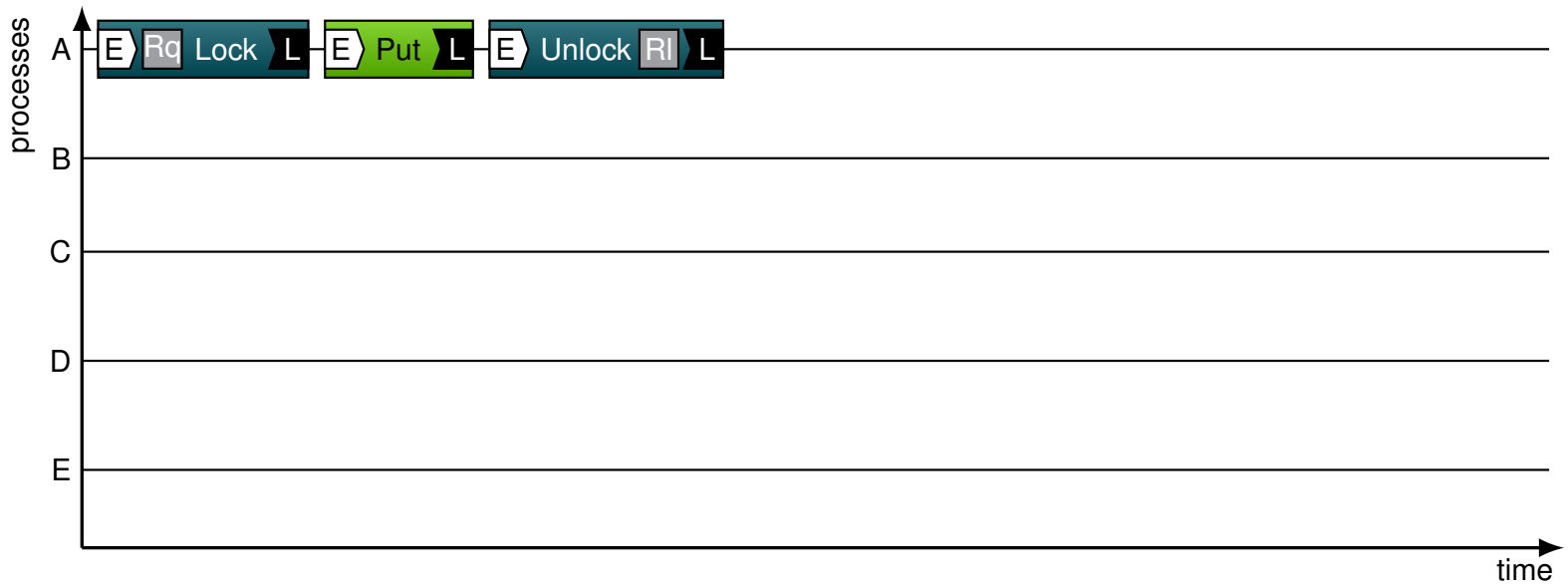
The Scalasca Toolkit

- Toolkit for trace-based performance analysis
  - Processes OTF2 traces created by Score-P
  - Also processes legacy traces in EPILOG format
- Parallel wait state detection
  - Inter-process synchronization
  - Inter-thread synchronization
- Uses message replay to interchange local data just in time
  - Synchronizing processes exchange data
  - Uses recorded communication information
  - Uses similar communication pattern

MPI one-sided communication

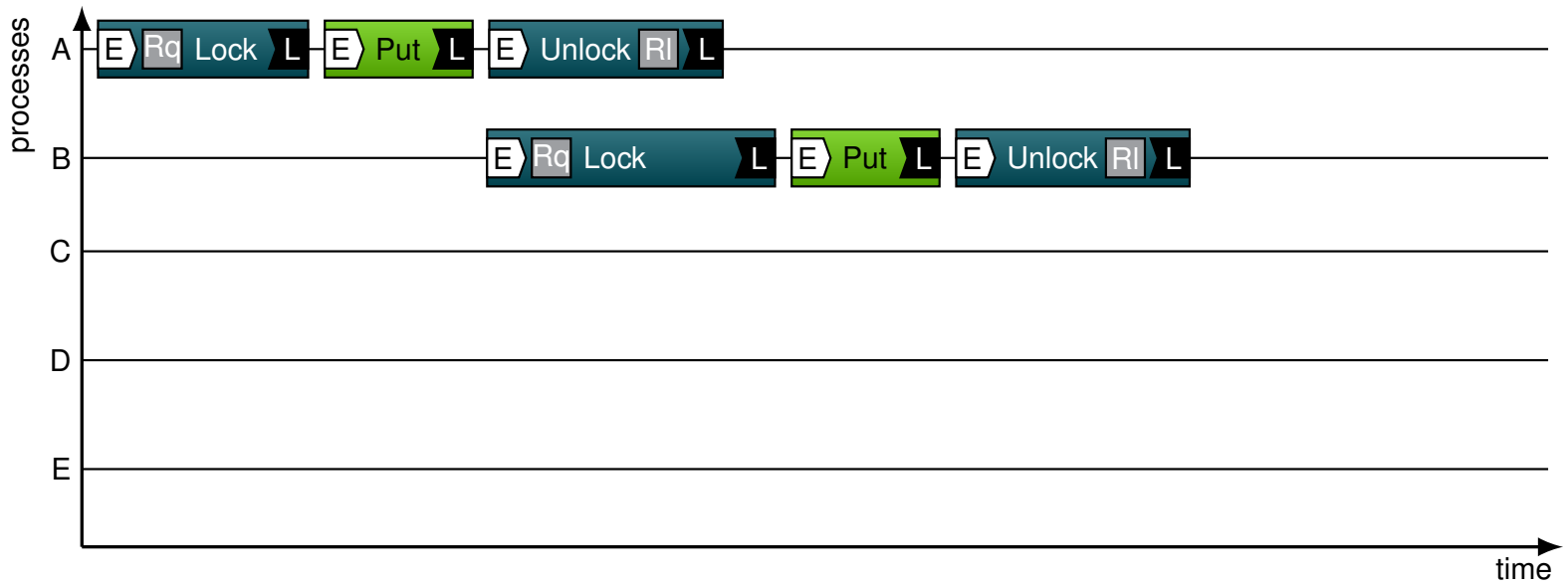
- Separate communication from synchronization
  - Multiple (logically concurrent) RMA operations
  - Single synchronization to ensure completion of pending operations
  - Two different synchronization modes
- Active-target synchronization
  - Both origin and target call synchronization functions
  - Target needs to know when to synchronize window
- Passive-target synchronization
  - Only origin calls synchronization functions
  - Target is not explicitly involved in synchronization
  - Mutual exclusion to window using locks (shared & exclusive)

Lock Contention



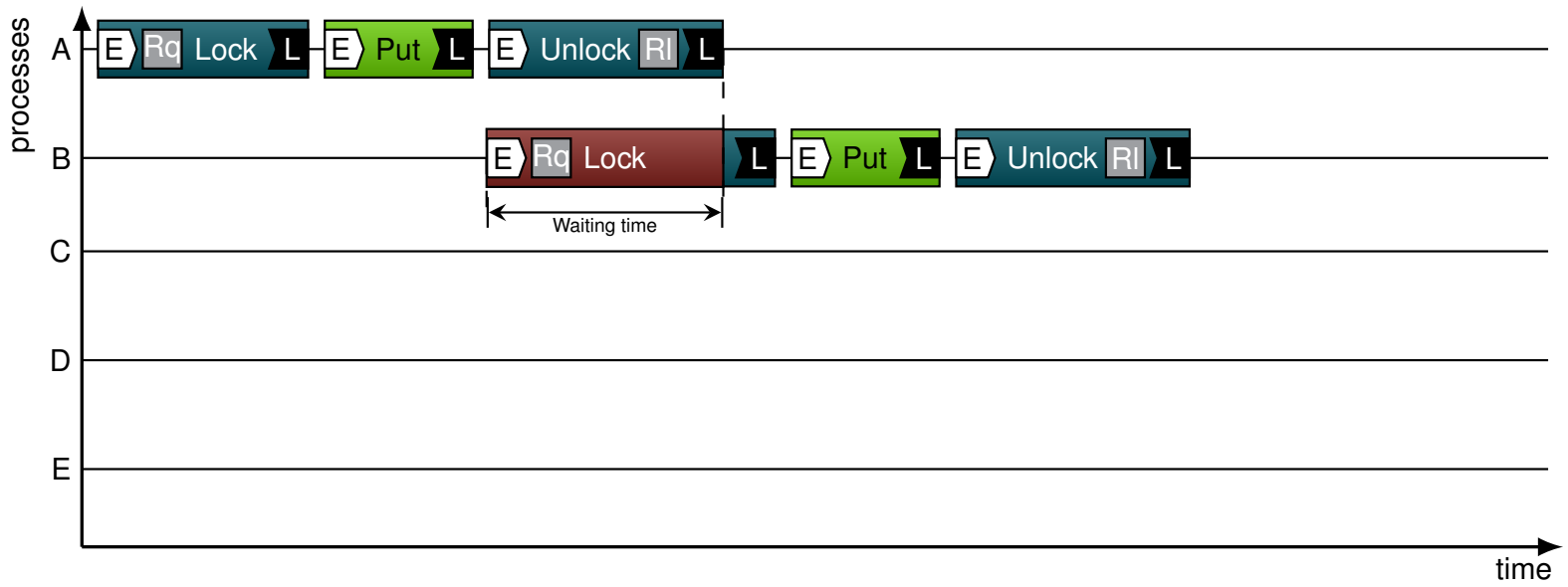
- RMA operations in passive target synchronization need to be placed in a lock epoch

Lock Contention



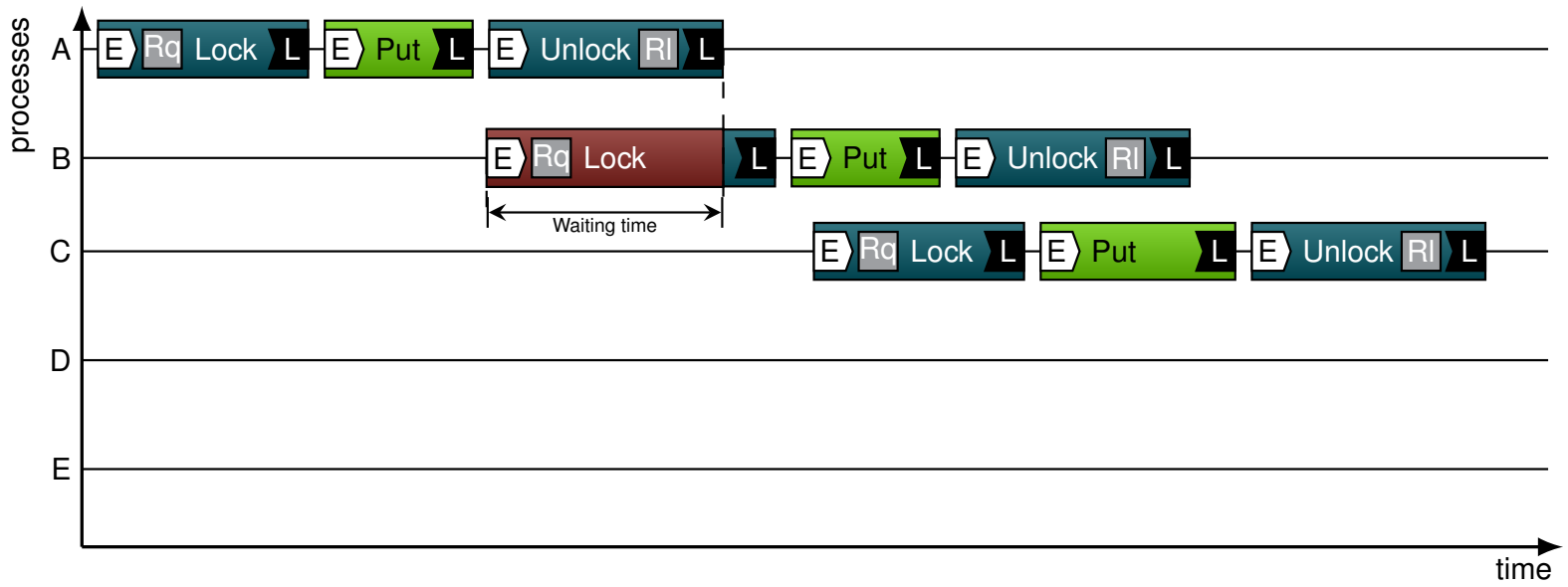
- The intuitive behavior would have competing lock epochs to be mutually exclusive

Lock Contention



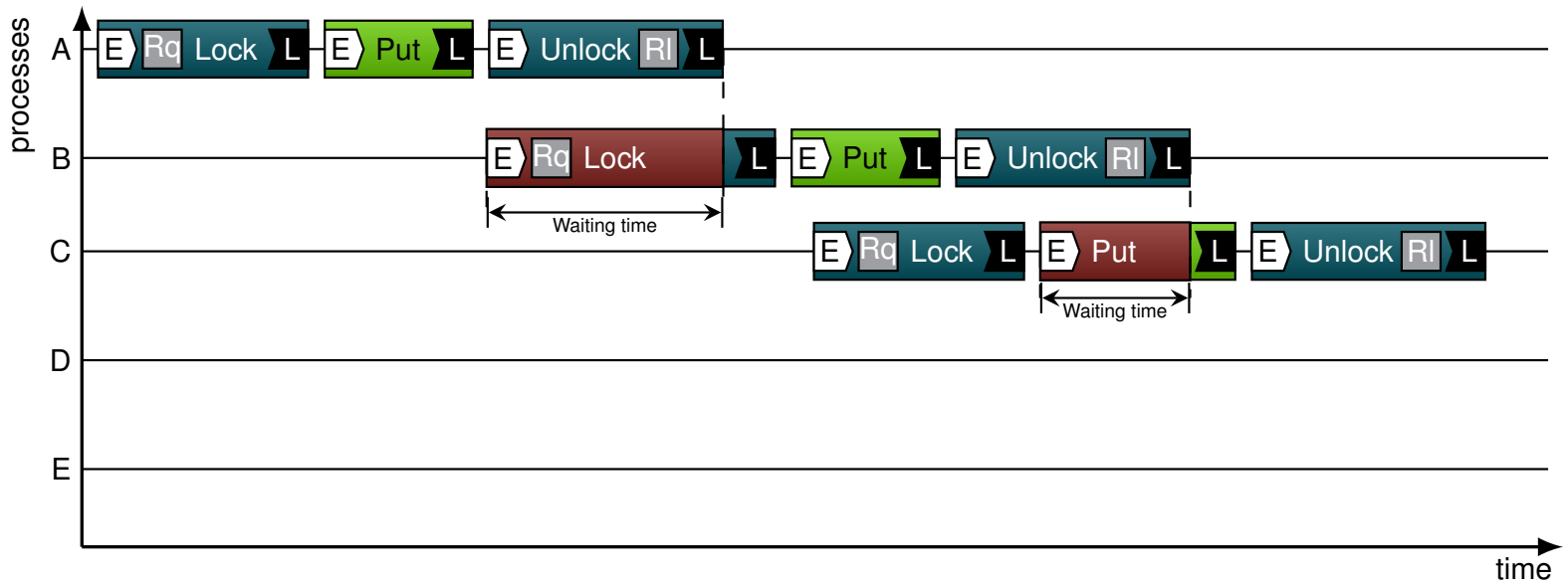
- Waiting time occurs in lock on process B, waiting for process A to release the lock

Lock Contention



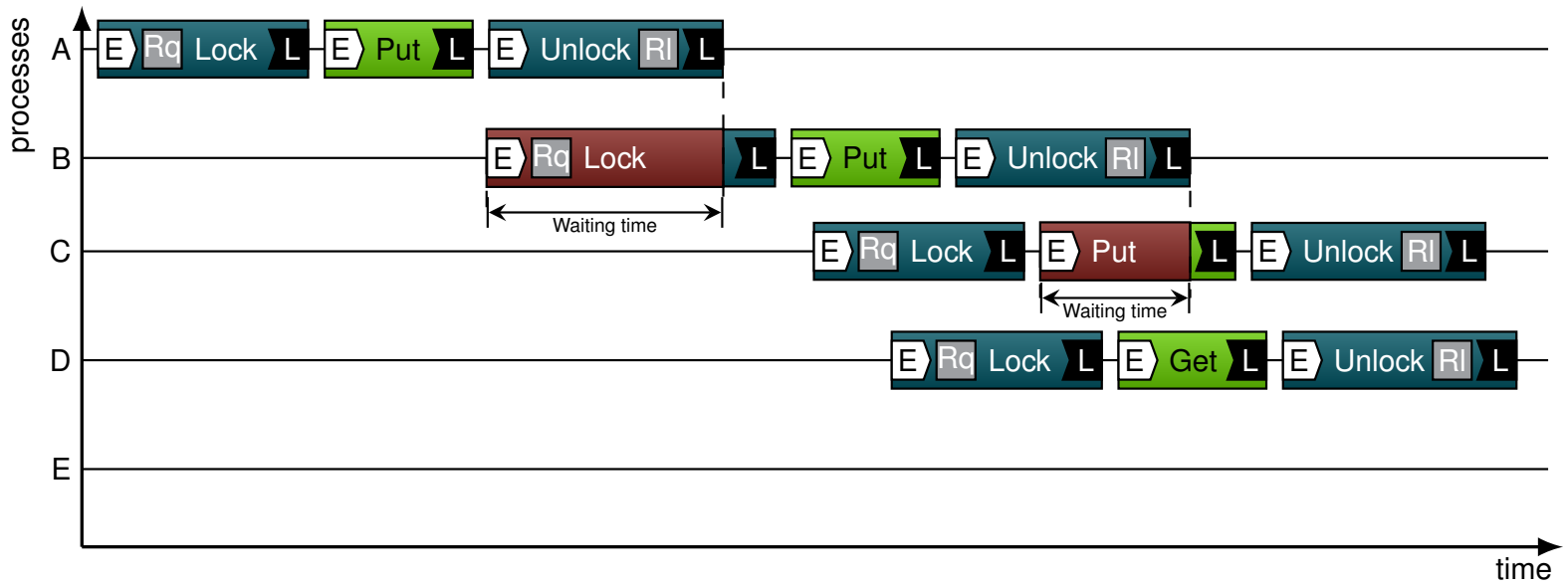
- MPI semantics allow the lock call to postpone the actual acquisition

Lock Contention



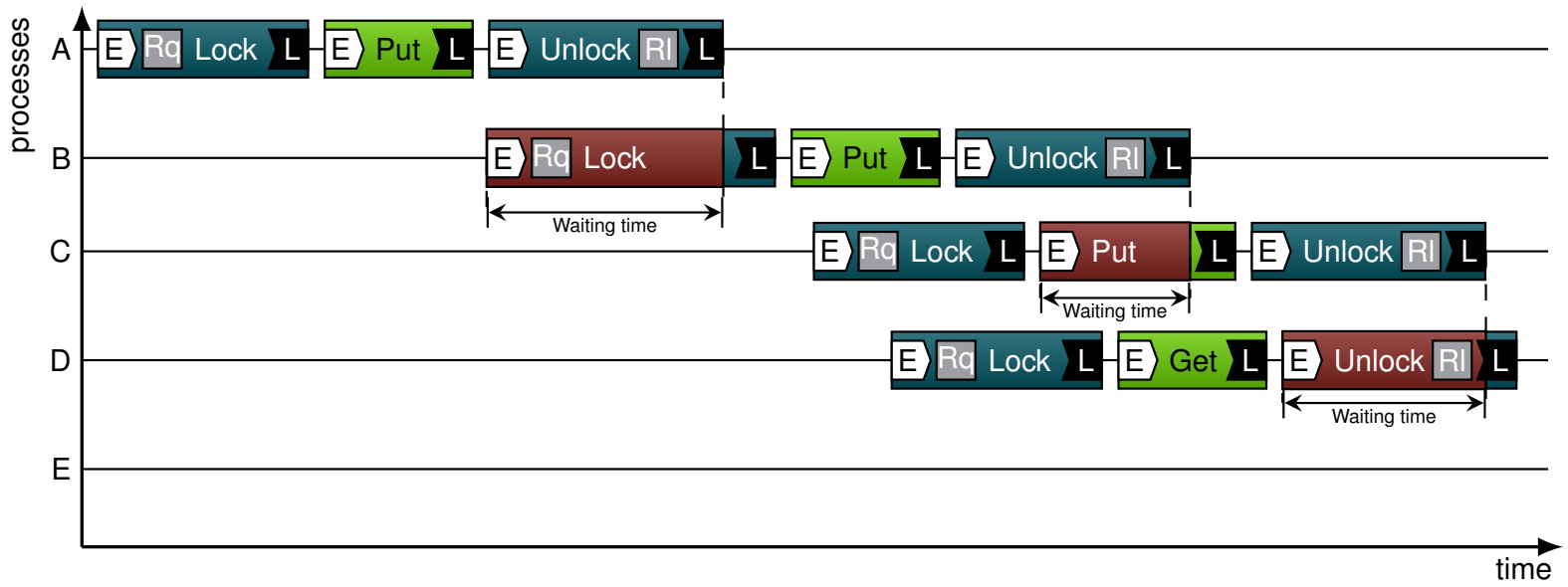
- Waiting time now occurs in RMA operation waiting for the lock

Lock Contention



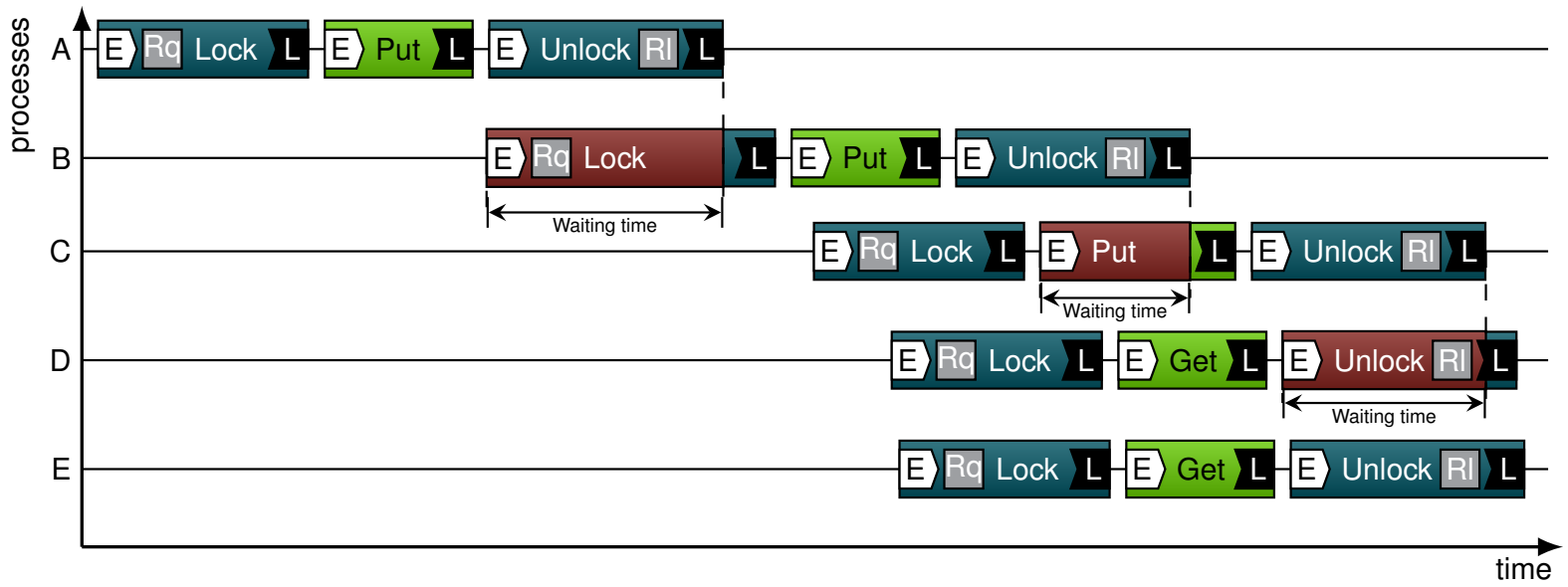
- MPI semantics even allow lock acquisition and RMA operations to be postponed until the unlock call

Lock Contention



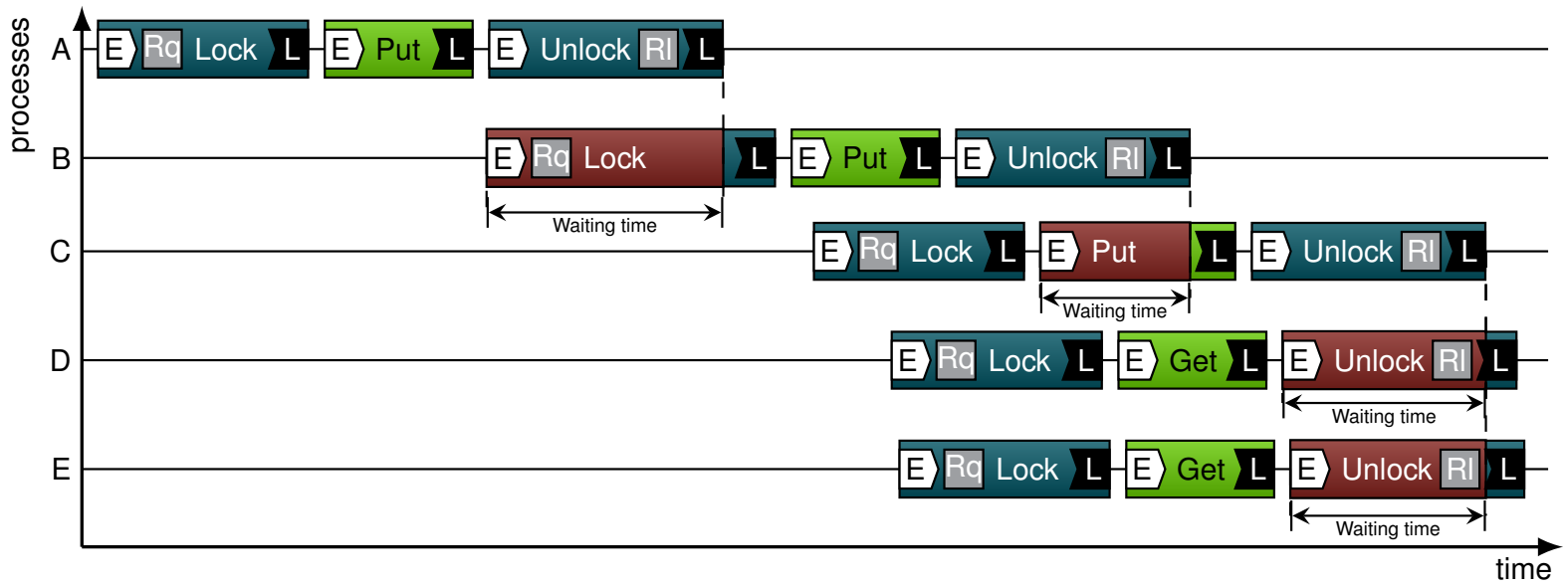
- Waiting time occurs in unlock operation

Lock Contention



- Lock epochs with shared locks may overlap

Lock Contention



- Waiting time can occur in context of previous conflicting locks

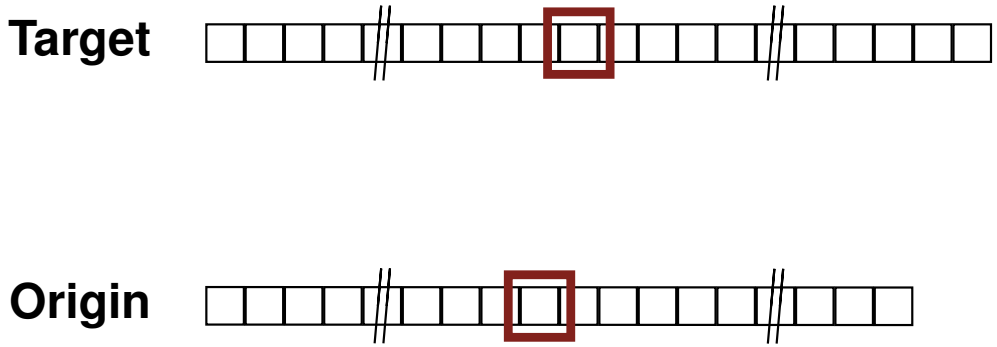
scalasca

Trace-based message-replay

Difficulties with passive-target synchronization

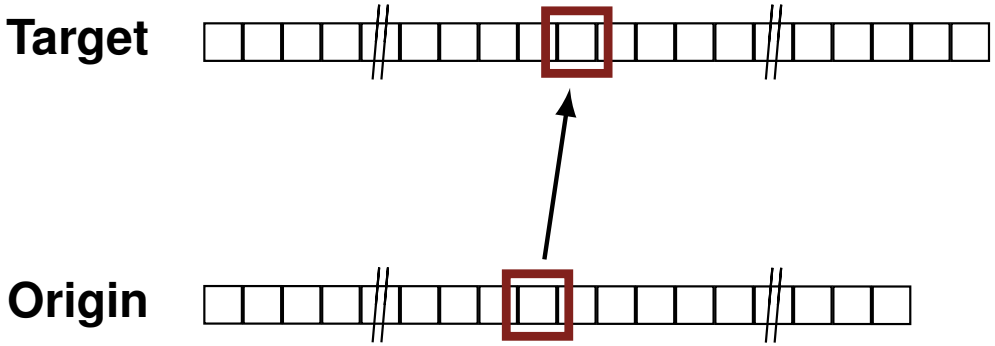
- Time of actual lock acquisition unknown
  - Use heuristic to compute lock acquisition
  - Full epoch information needed
- Target cannot record synchronization data via wrappers
  - No target-side events to trigger data exchange
- Incomplete synchronization information at the origin
  - Events contain target information
  - Access conflict is among two or more origin processes
- Locks may suffer contention and insufficient progress

Active-message replay



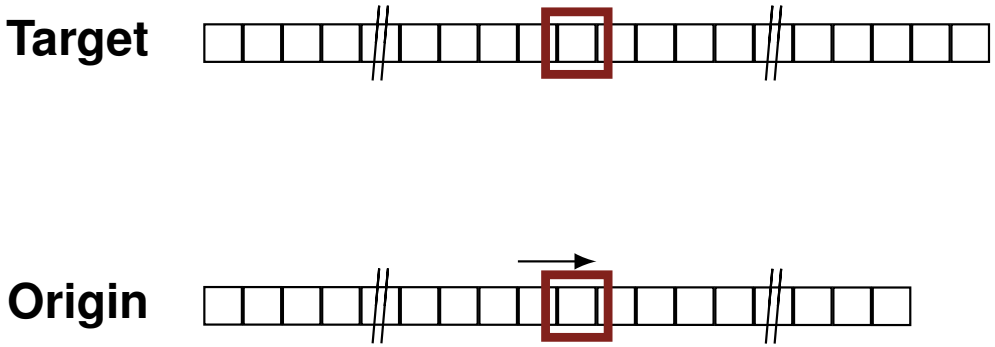
- Individual trace processing can be at arbitrary points

Active-message replay



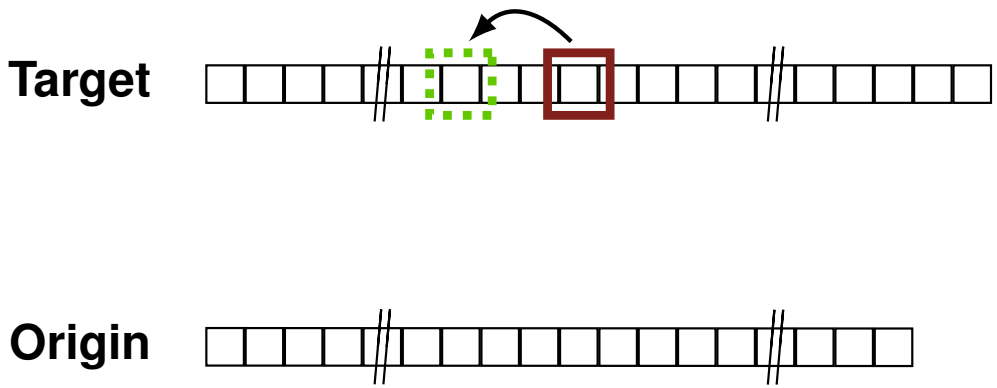
- Origin packs and sends active message to target

Active-message replay



- Origin continues processing

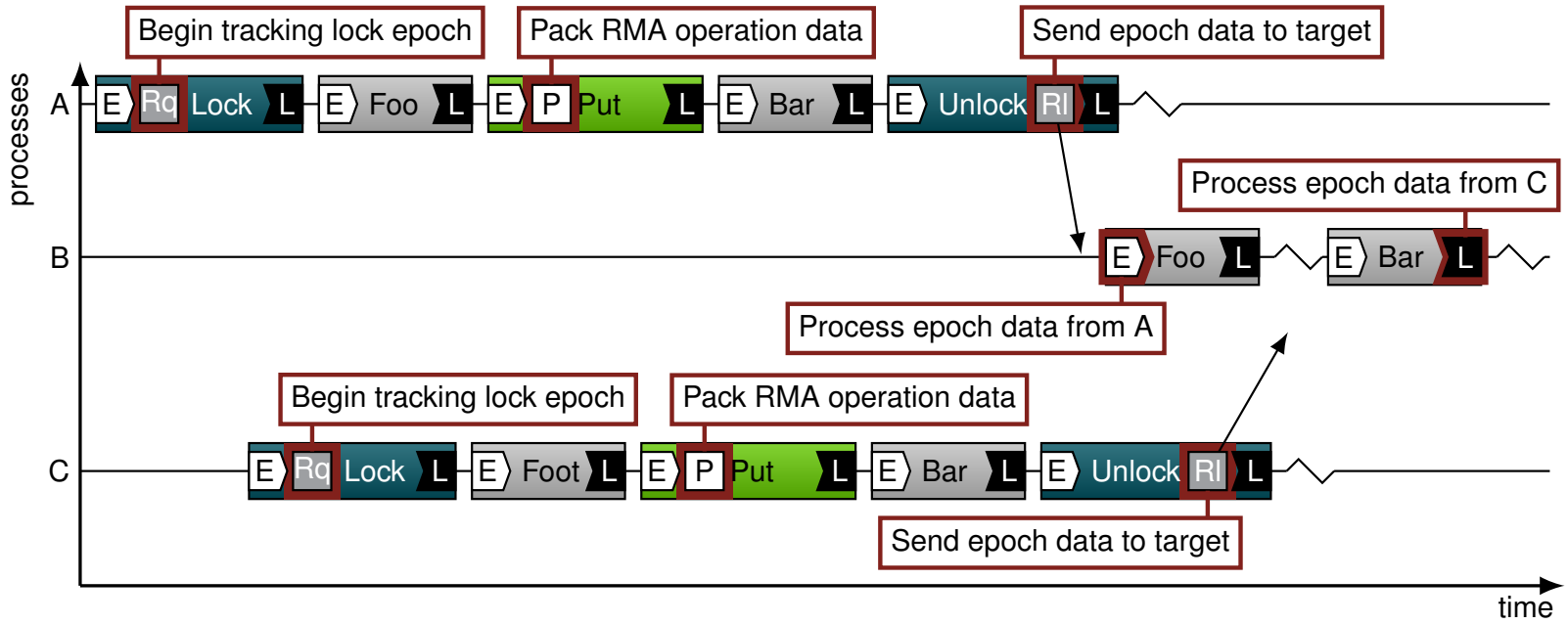
Active-message replay



- Target processes active message upon arrival
- Identifies corresponding event in $\mathcal{O}(\log n)$

Analysis phase I

Collation of epoch data



Analysis phase II

Detecting contention

- Epochs are sorted by latest unlock event
- Analysis starts with last epoch and continues back in time

For each lock epoch in queue:

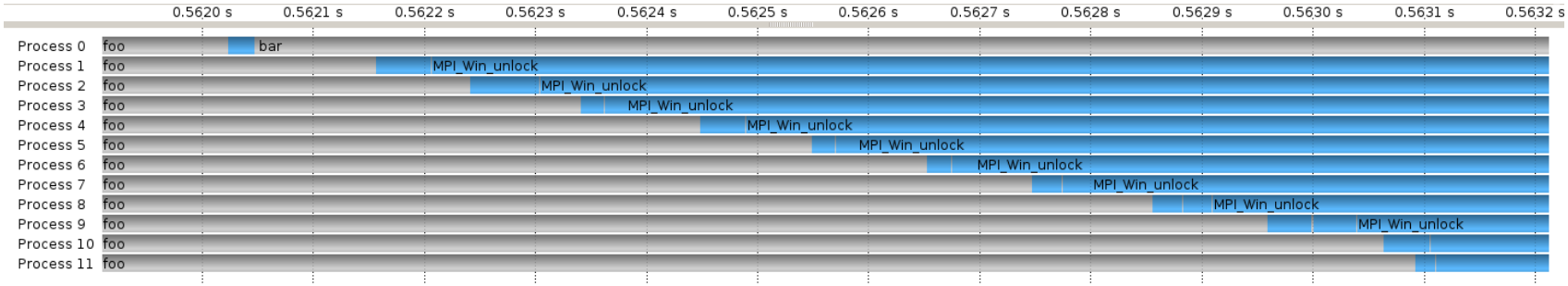
1. Find conflicting preceding epoch
2. Get unlock time from preceding epoch
3. Get local (target-side) progress region
4. Find location of wait state within current epoch
5. Send active message with wait state information to affected processes

Simple benchmark

- Single iteration
- Skewed begin of lock ensures lock request order on processes
- Target blocks window until all processes requested lock
- Target releases lock to let origins complete RMA operation

Simple benchmark

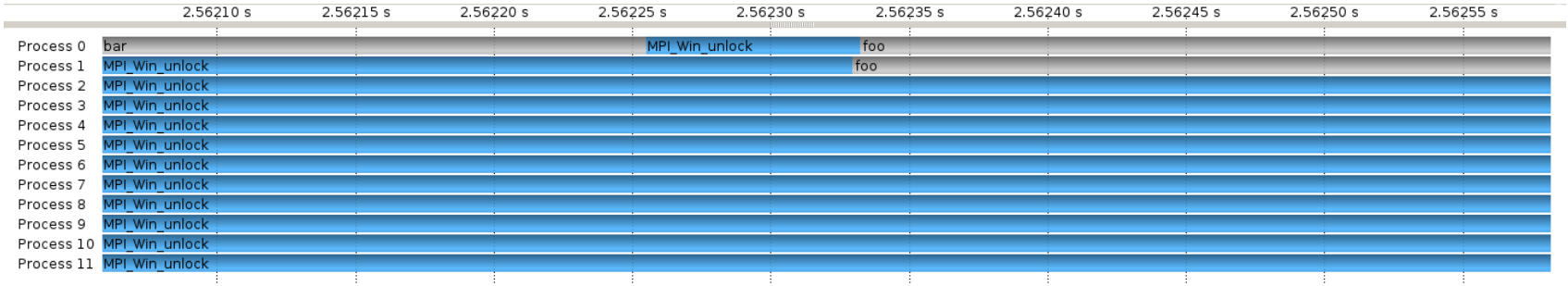
Vampir view: Lock phase



- All processes execute `foo` for time depending on their rank
- Process 0 is target for RMA operations
- Target locks window exclusively
- Target then executes `bar` for 2*s*
- Rest of processes wait for access in unlock operation

Simple benchmark

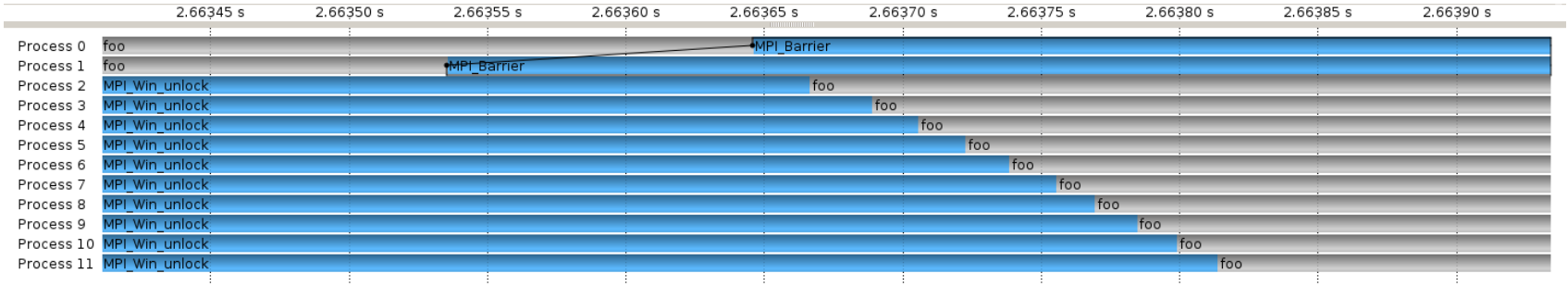
Vampir view: Target unlocks



- Target unlocks window after leaving `bar`
- First origin (process 1) gains access to window
- Target continues to execute `foo` again for 100*ms*
- Second origin (process 2) is waiting for target progress

Simple benchmark

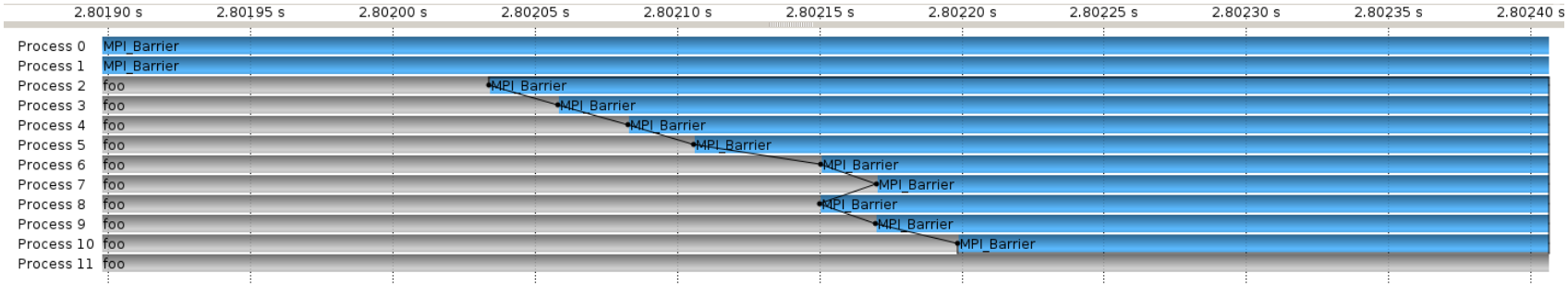
Vampir view: Unlock completion



- After `foo` completes, target enters `barrier`
- Barrier provides progress for remaining origin processes
- Remaining origins complete access

Simple benchmark

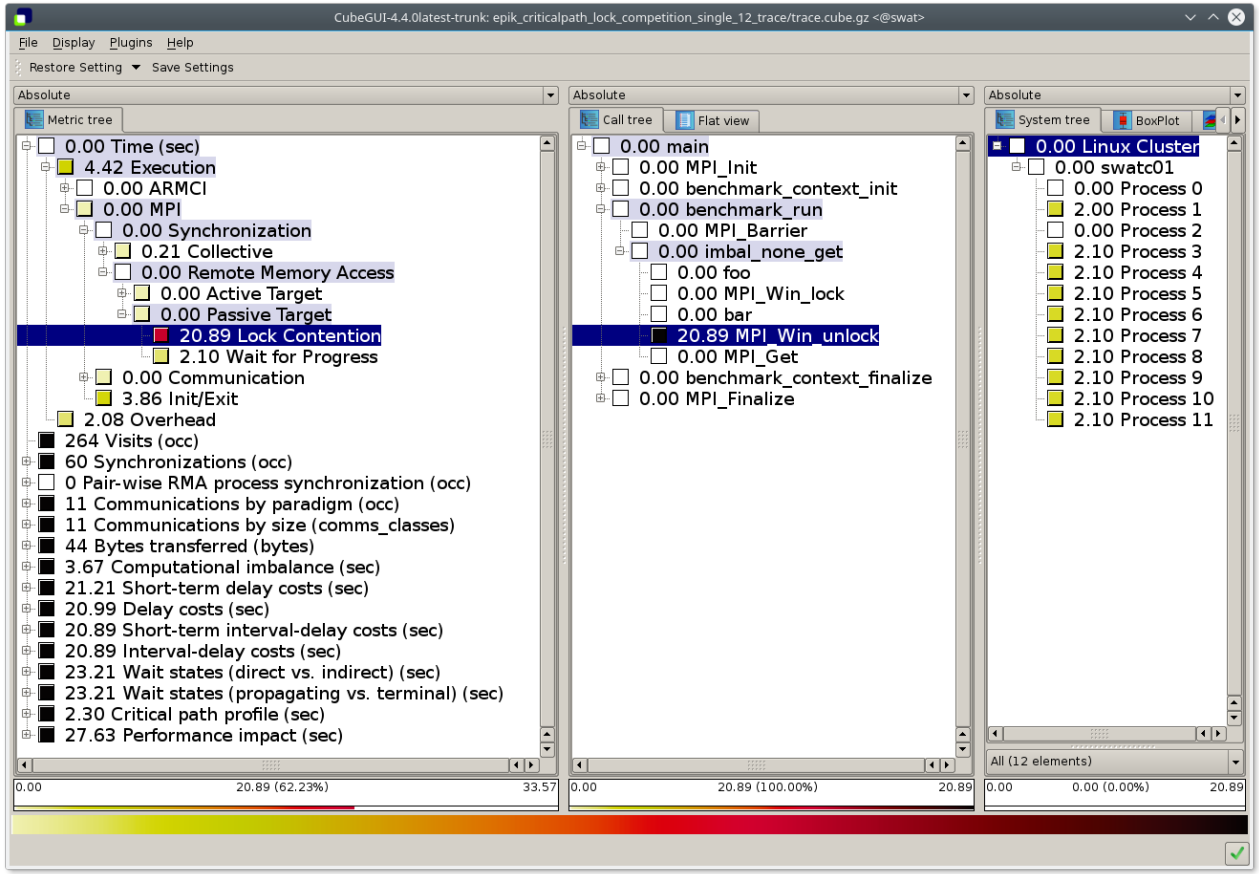
Vampir view: Benchmark completion



- `foo` completes on remaining processes
- Sequence completes with all processes entering `barrier`

Simple benchmark

Cube view: Lock contention & wait for progress

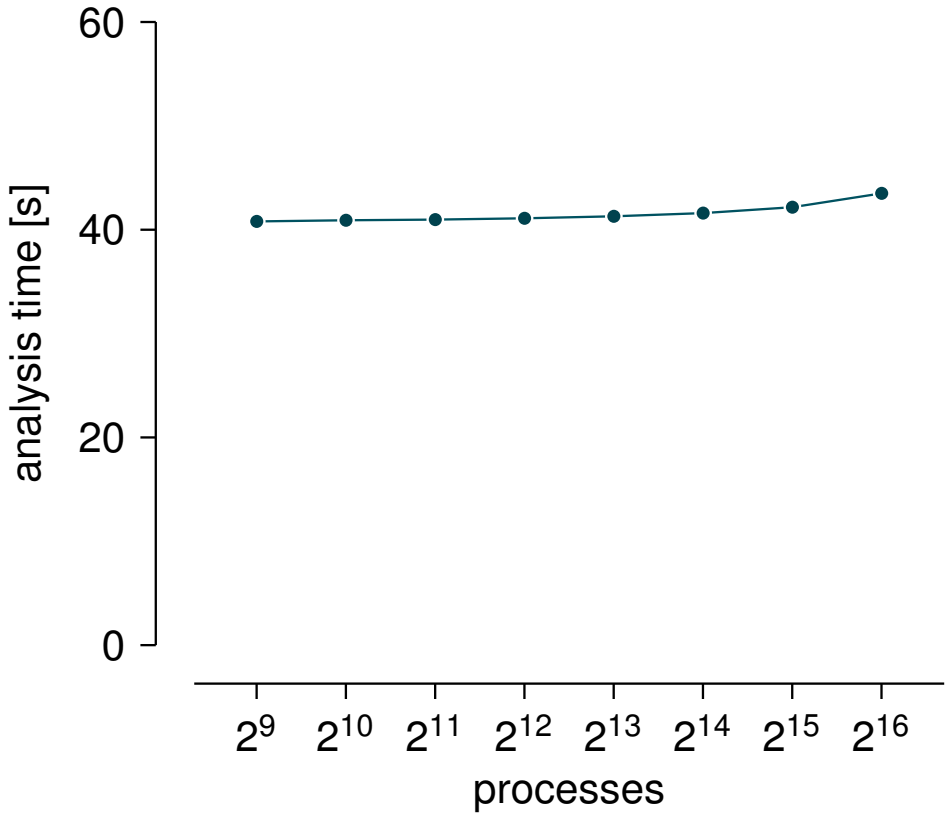


SOR benchmark

- Solves Poisson equation using successive over relaxation
- 2D domain decomposition
- Ghost-cell exchange configured for
  - MPI one-sided communication
  - Passive-target synchronization
- Configured for weak scaling (keeps local trace data constant)

SOR benchmark

Weak scaling



Conclusion

Trace-based detection of lock contention in MPI

- Identifies lock acquisition time and order
- Differentiates between lock contention and insufficient progress
- Enables understanding of complex synchronization schemes
- Narrows support gap in wait-state detection for one-sided communication

Outlook

- Port analysis prototype to Scalasca 2.x
  - Extend support to other one-sided libraries (OpenSHMEM, ARMCI, etc.)
- Further improve analysis performance
  - Target-side message handling
  - Work distribution
- Integrate contention wait states into higher-level analysis
  - Root-cause detection
  - Critical path detection
- Enhance MPI interfaces
  - Enable target-side event generation
  - Provide more precise locking information on origin

scalasca

Thank you.