Extracting Logical Structure and Identifying Stragglers in Parallel Execution Traces

Katherine E. Isaacs[1] Todd Gamblin[2] Abhinav Bhatele[2] Peer-Timo Bremer[2] Martin Schulz[2] Bernd Hamann[1]

[1]Institute for Data Analysis and Visualization, Department of Computer Science, University of California, Davis [2]Lawrence Livermore National Laboratory

Abstract

We introduce a new approach to automatically extract an idealized *logical structure* from a parallel execution trace. We use this structure to define intuitive metrics such as the *lateness* of a process involved in a parallel execution. By analyzing and illustrating traces in terms of logical steps, we leverage a developer's understanding of the happened-before relations in a parallel program. This technique can uncover dependency chains, elucidate communication patterns, and highlight sources and propagation of delays, all of which may be obscured in a traditional trace visualization.

Extracting Logical Structure

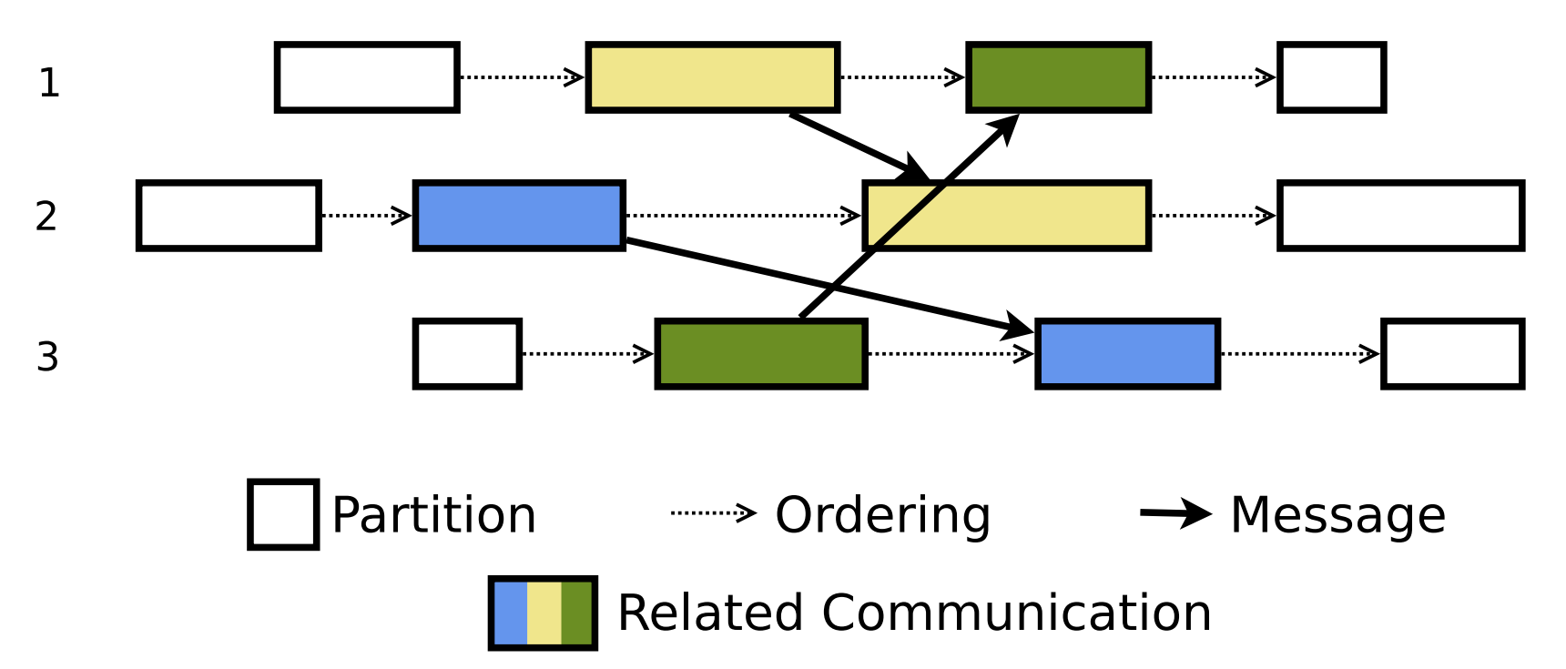
The logical structure of a program is the ordering of events implied by that program. We describe the logical structure by assigning a *logical step* to each event.

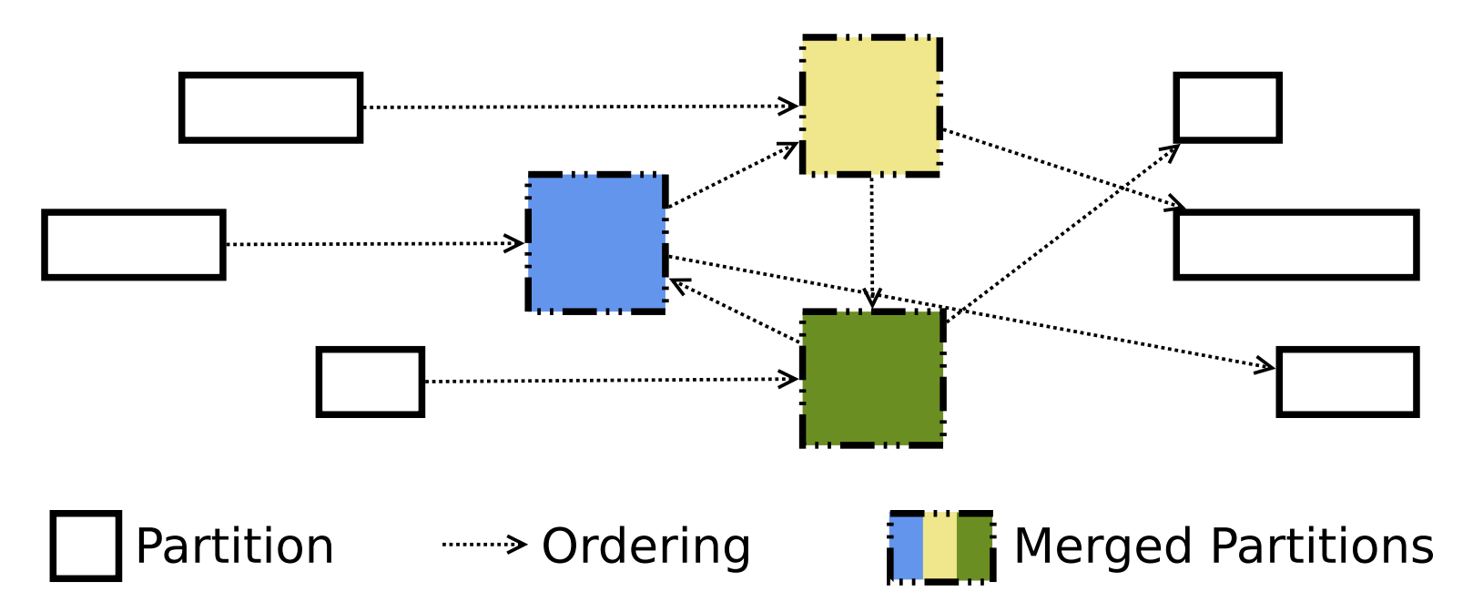
Structure extraction occurs in two phases:
1. Partitioning related communication
2. Step assignment

Partitioning

Partitions represent non-overlapping application phases. If not predefined, we derive them from the trace:

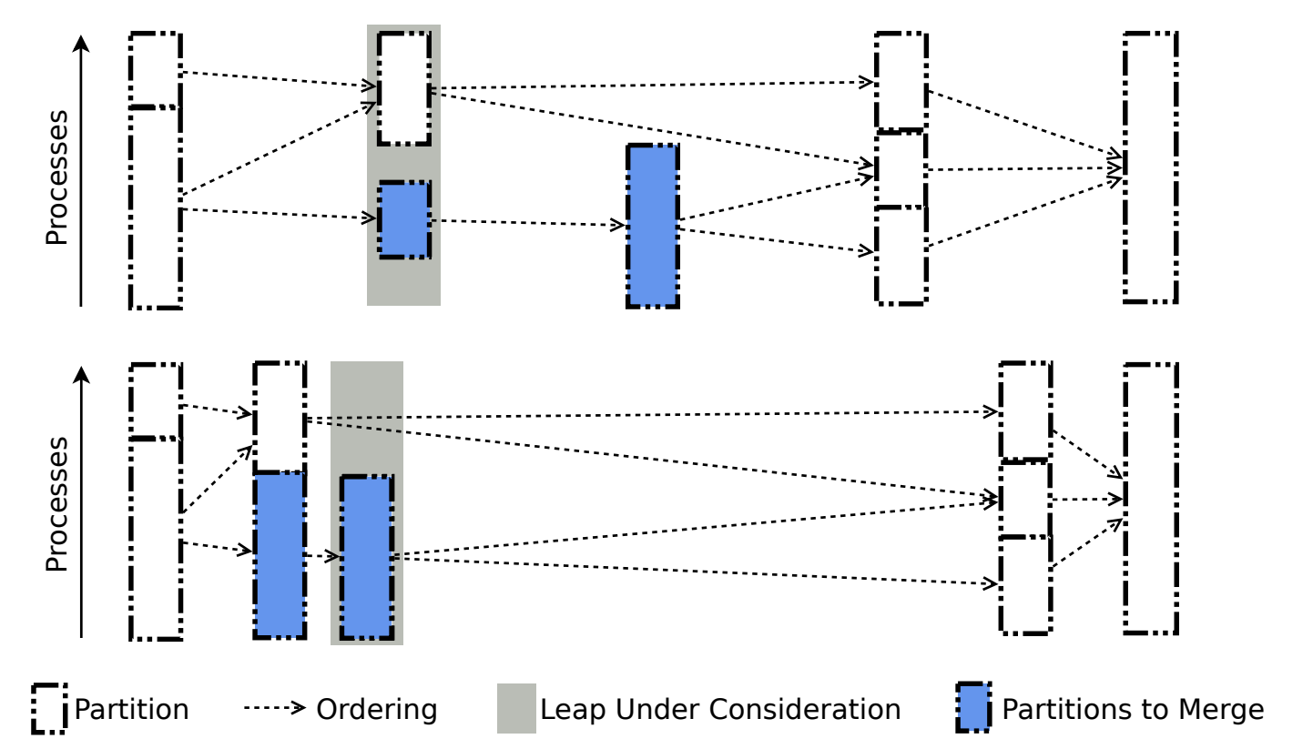
Matching sends and receives and communication handled by the same MPI call must be related and thus in the same partition. When merged, this can create cycles in ordering:





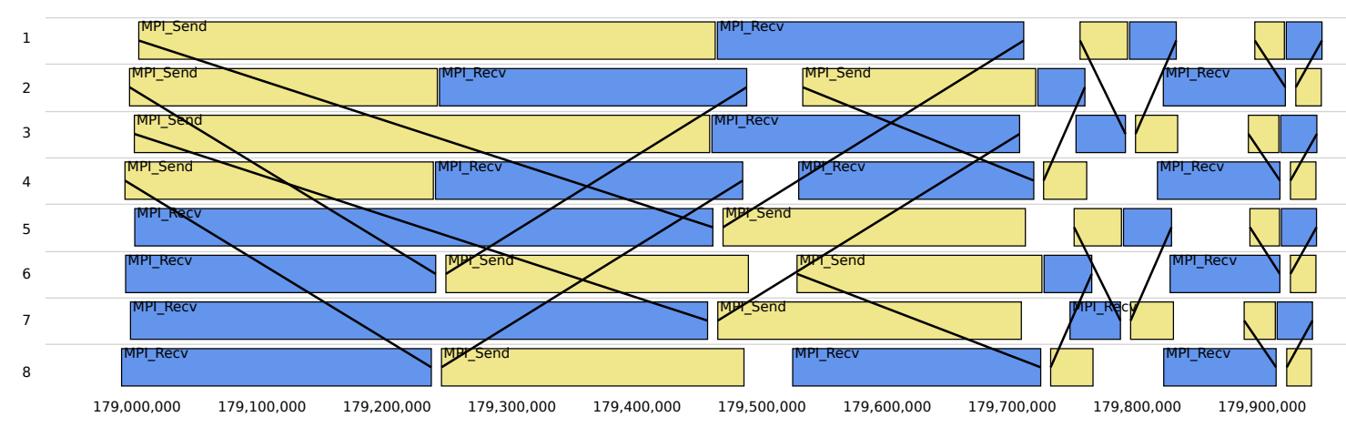
Communication partitions forming a cycle do not permit a partial order, so we infer these partitions are related and merge them.

In addition to merging due to ordering constraints, we can optionally merge due to behavioral assumptions. For example, in bulk synchronous codes we expect each process to be active at some distance in the partition graph.



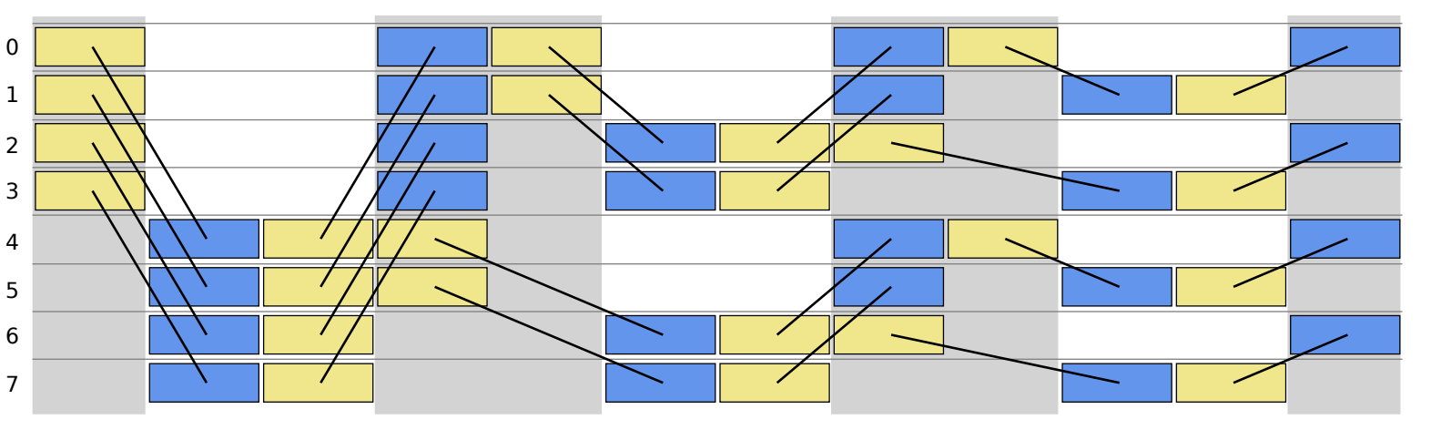
Step Assignment

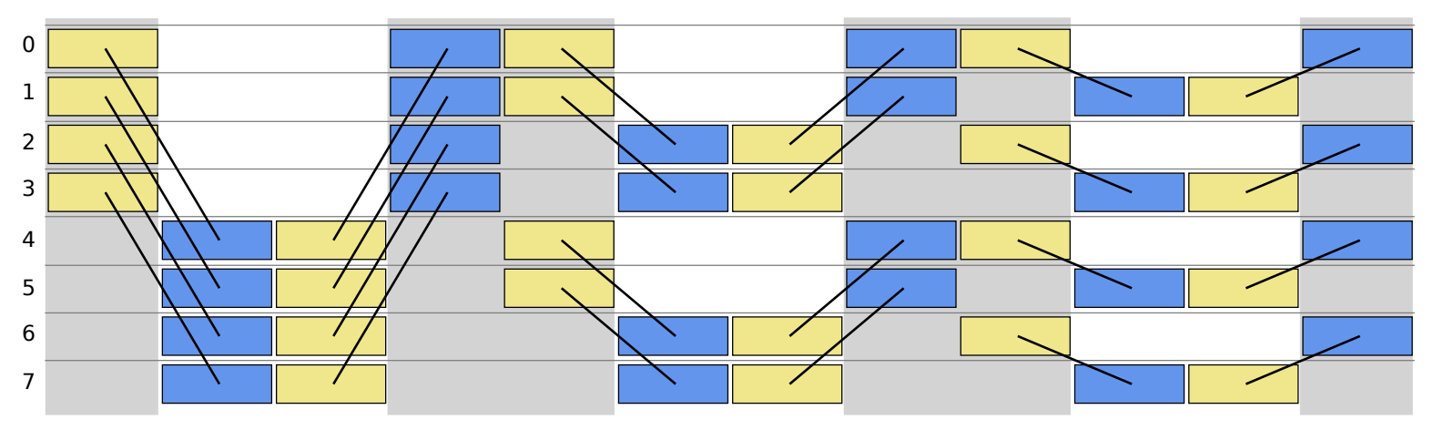
Each partition is independently assigned steps based on two principles:
1. Happened-before relationships must be maintained
2. Send events have greater impact on structure



Consider this trace segment from an 8-process run of the pF3D stencil communication benchmark [1].

First we determine groups of simultaneous sends (gray) using receives only for ordering.





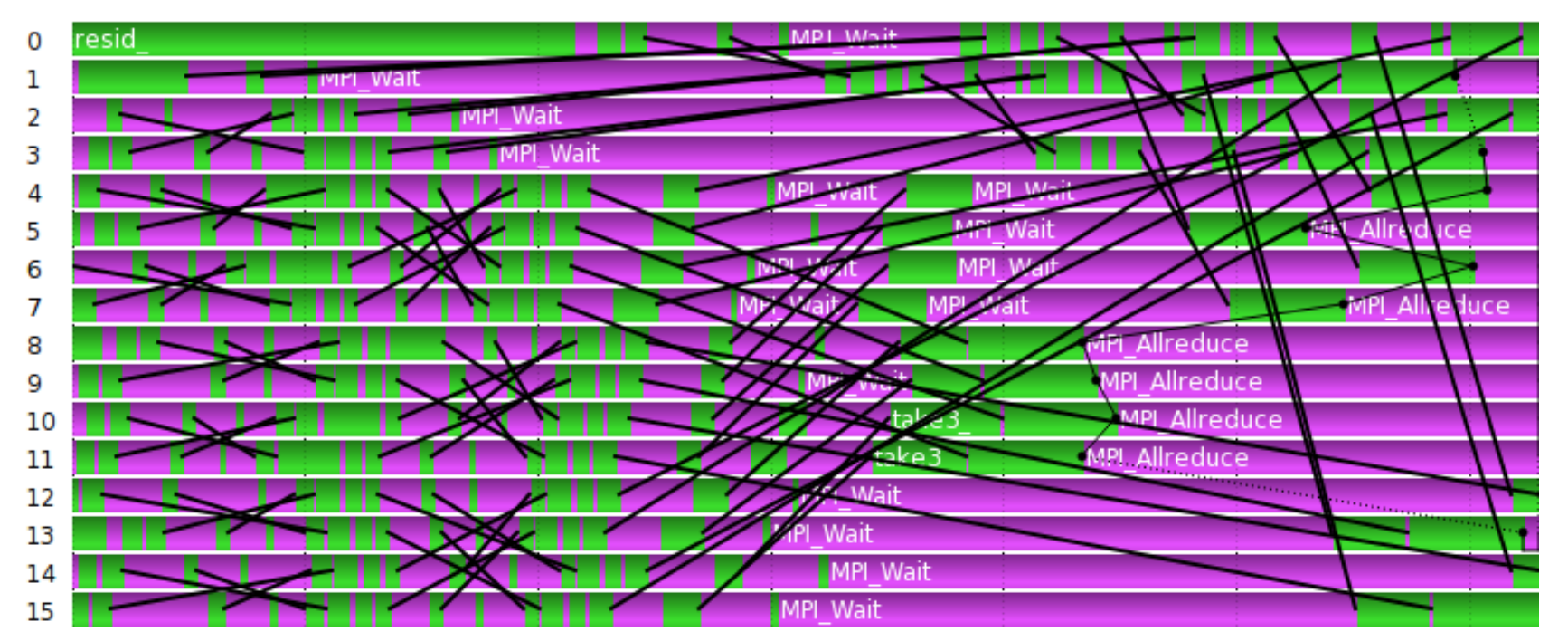
Then we assign the least step possible to each event.

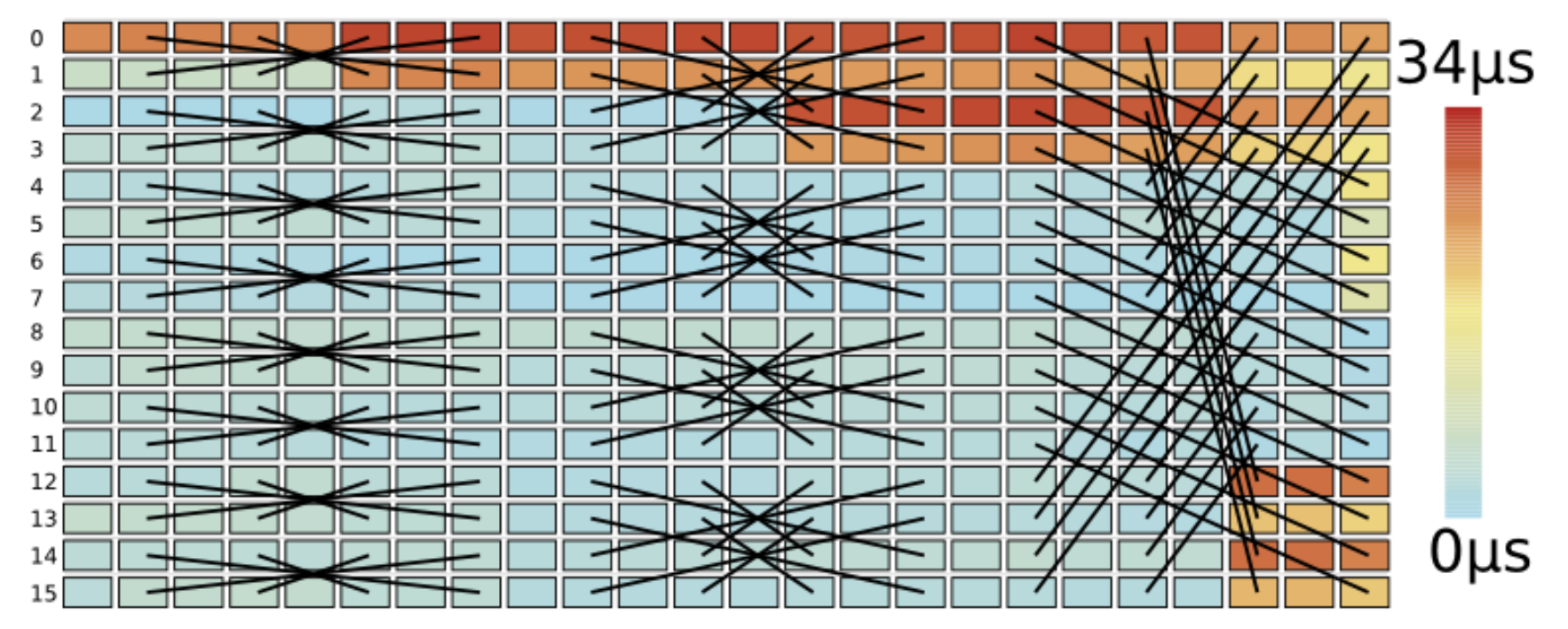
Finally we insert aggregated non-communication events between the sends and receives and determine global steps using partition ordering.

Temporal Metrics

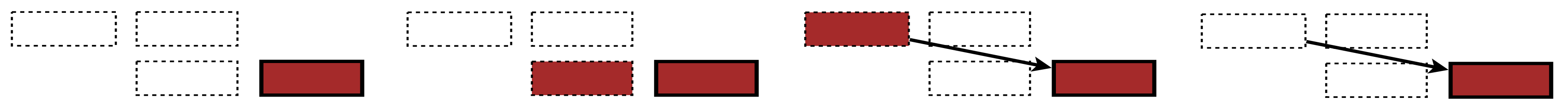
Having determined a logical structure, we can calculate how late an event was relative to its peers. We define *lateness* as excess completion time over the earliest related event at a step.

We visualize a portion of an MG [2] trace using traditional methods as represented by Vampir [3] (left) and logical structure and lateness (right). In the latter the communication pattern and delay propagation is clear.





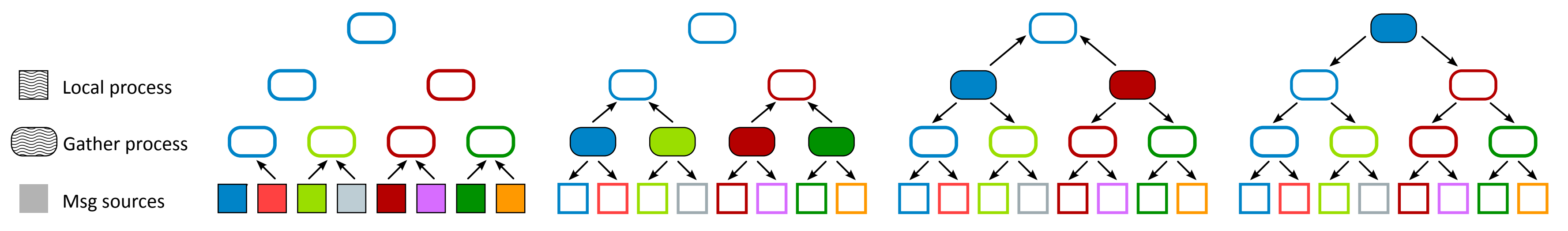
We classify four situations contributing to event lateness:



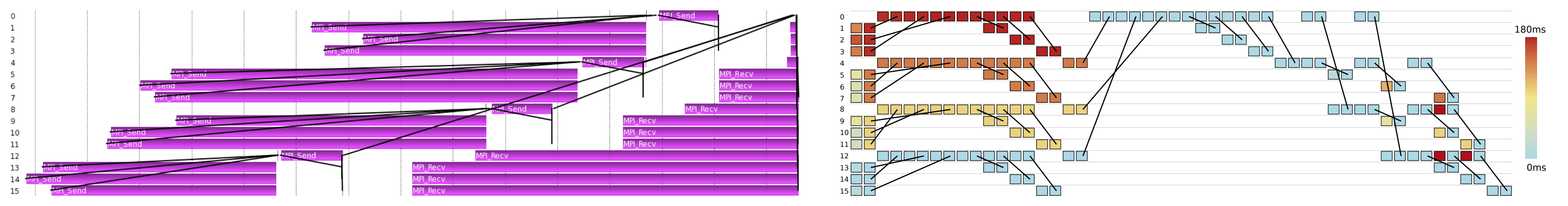
Using this classification, we can narrow our focus to events where lateness originates by subtracting out propagated lateness. This *differential lateness* allows us to pinpoint sources of delays automatically.

Case Study

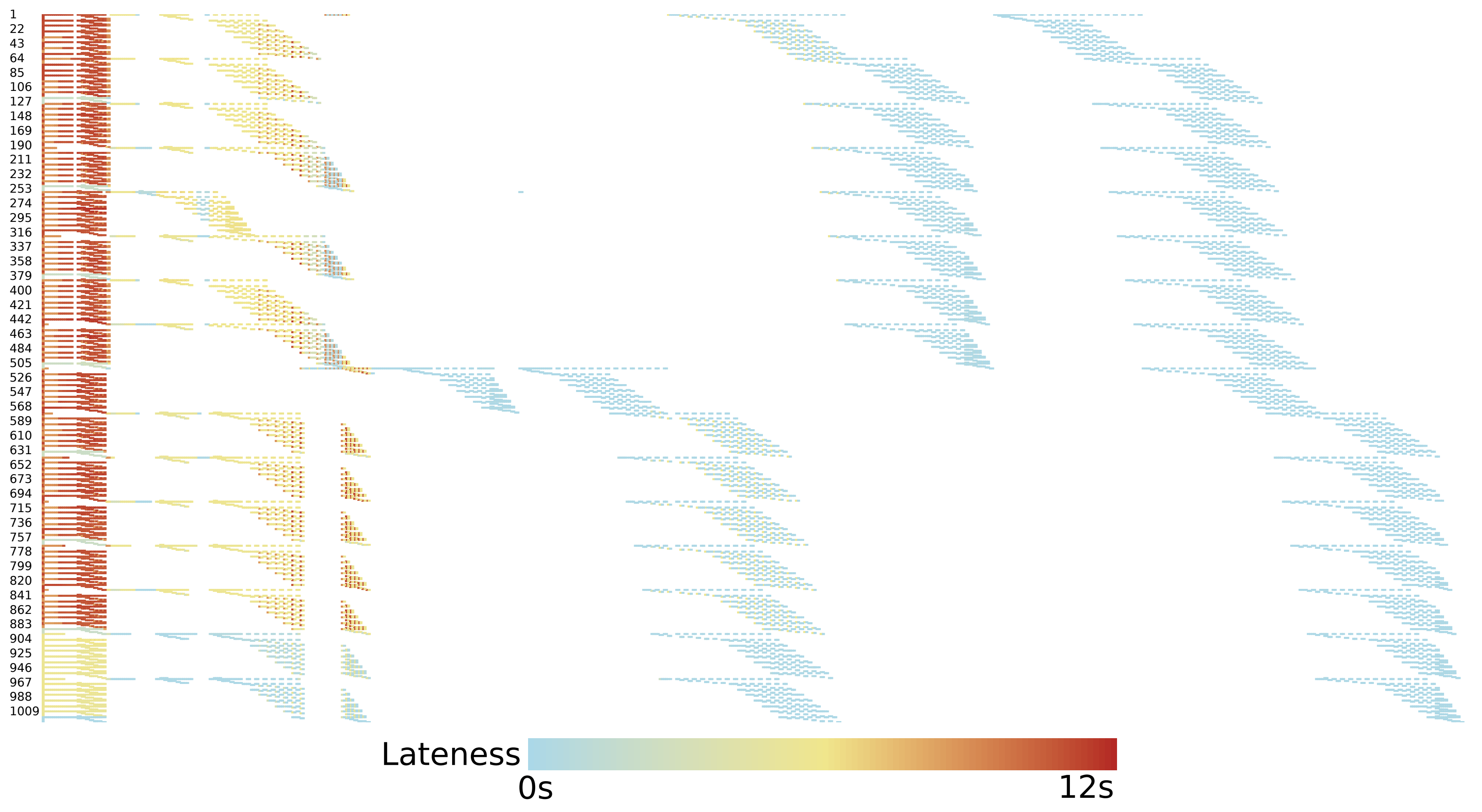
We analyze a massively parallel algorithm to compute *merge trees*. The algorithm relies on a global gather-scatter approach where each level requires messages sent both up and down a k-ary gather tree:



Below are the Vampir (left) and logical structure (right) visualizations of a 16 process, 4-ary merge tree calculation. In the logical structure view, lateness reflects data-dependent load imbalance. Logical steps highlight the gather tree structure, revealing that the gather processes send back to the leaves before sending up to the root, missing an opportunity for more aggressive pipelining.



The 1024-process, 8-ary tree below shows similar issues. The recurring "panhandle" shape highlights waiting due to sending down before up.



References

1. C. H. Still et al. Filamentation nd forward brillouin scatter of entire smoothed and aberrated laser beams. *Physics of Plasmas*, 7(5):2023, 2000.
2. D. H. Bailey et al. The nas parallel benchmarks. *Int. J. Supercomput. Appl.*, 5(3):63–73, 1991.
3. W. E. Nagel et al. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer,* 12(1):69-80, 1996.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. LLNL-POST-649674
This research is supported by the Department of Energy Office of Science Graduate Fellowship Program (DOE SCGF), made possible in part by the American Recovery and Reinvestment Act of 2009, administered by ORISE-ORAU under contract no. DE-AC05-06OR23100.