# Optimal Tracing and Replay for Debugging Message-Passing Parallel Programs

Robert H. B. Netzer Brown University

Barton P. Miller University of Wisconsin-Madison

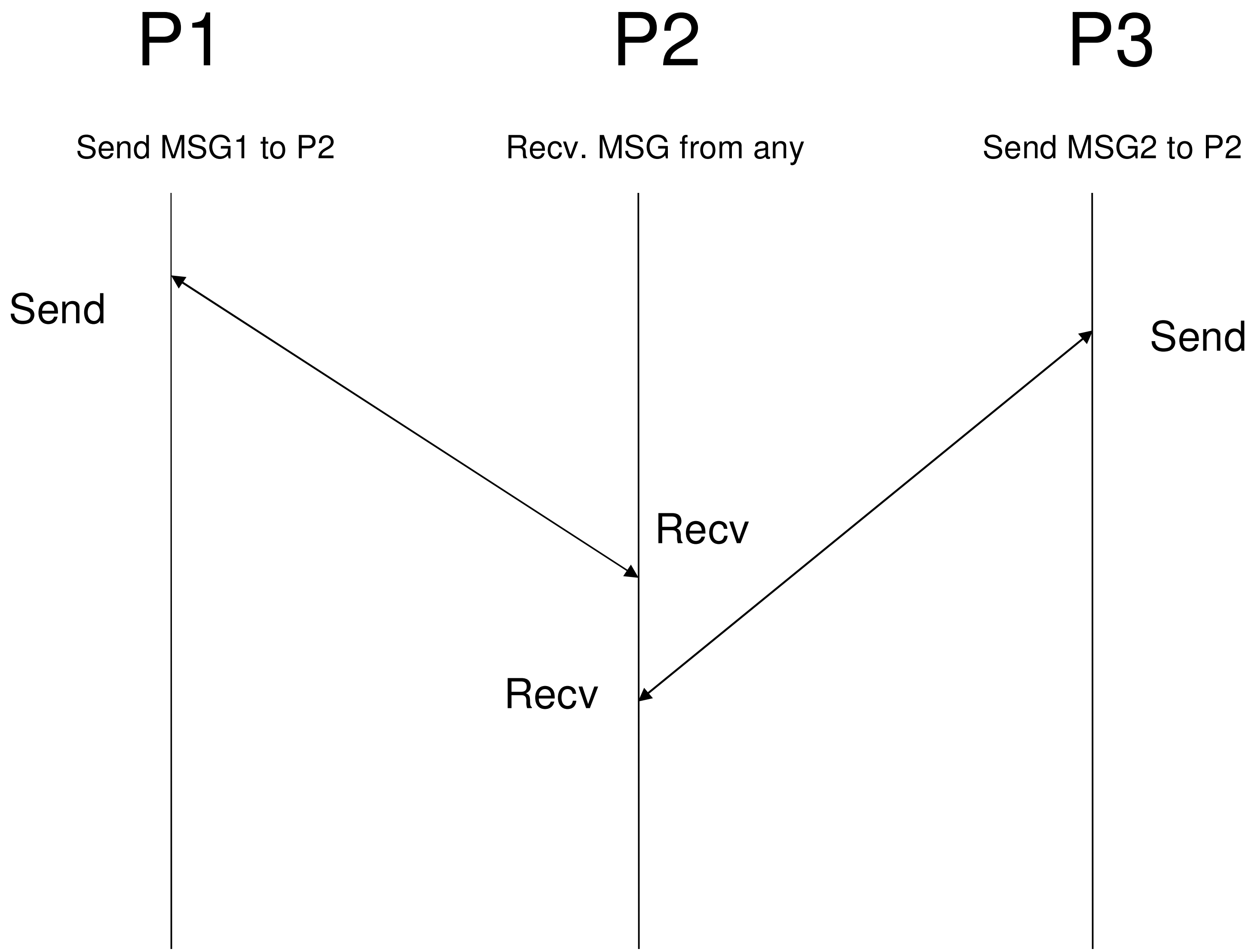
slides made by Qing Zhang

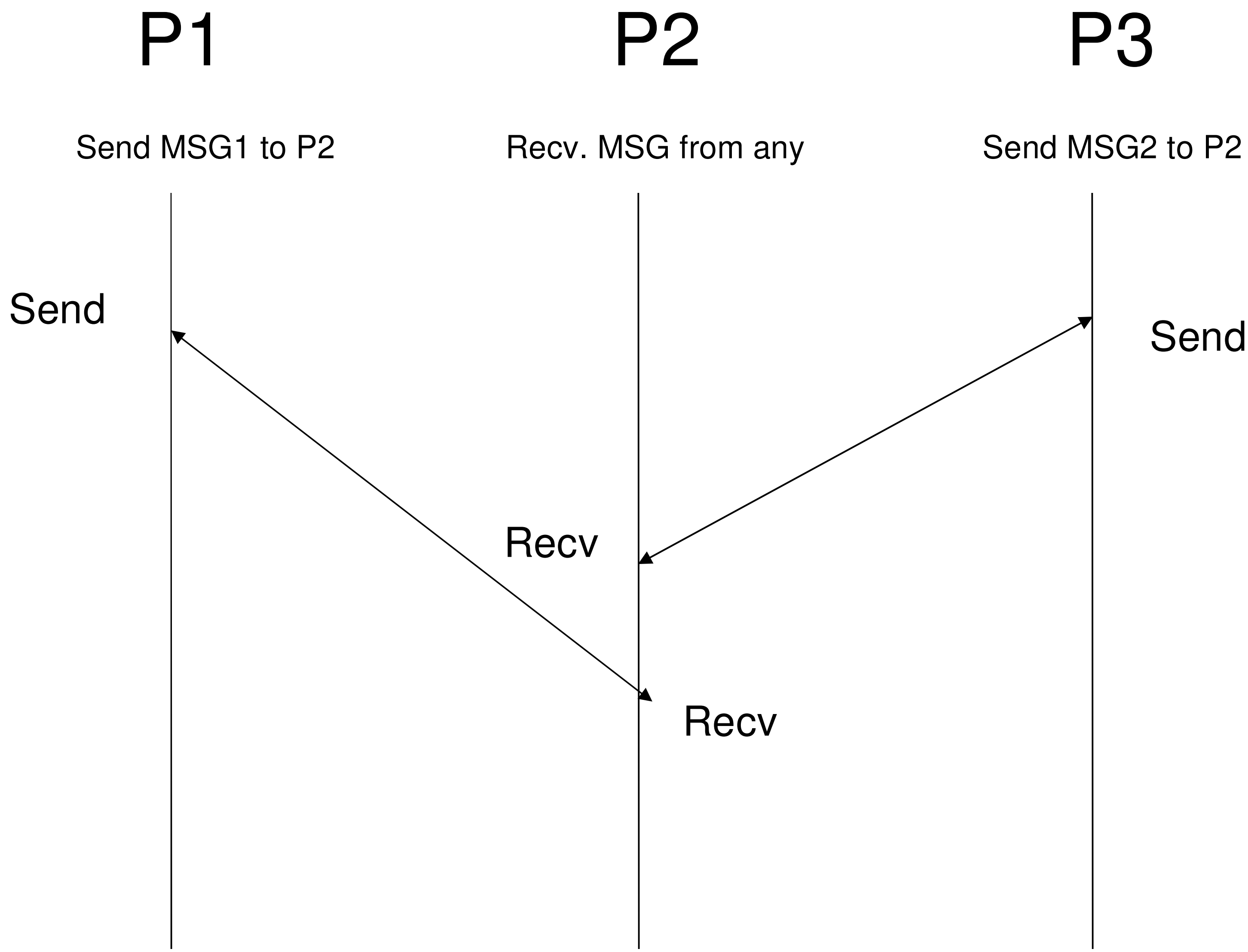
# Motivation

- Reverse execution is effective in debugging
- Message passing programs are hard to replay due to races
- Early work involves all the messages to be traced

# Introduction

- Tracing all messages is expensive
- Trace only racing messages
- Check for races for each messages
- Only trace one


P1
P2
P3
Send MSG1 to P2
Recv. MSG from any
Send MSG2 to P2
Send
Send
Recv
Recv


P1
P2
P3
Send MSG1 to P2
Recv. MSG from any
Send MSG2 to P2
Send
Send
Recv
Recv

# Race Condition

- Definition of racing frontiers
- Frontier - divides the events
- Two or more sends are just after frontier
- A receive can accept either after the frontier
- All receive before must have senders before

# Frontier Example

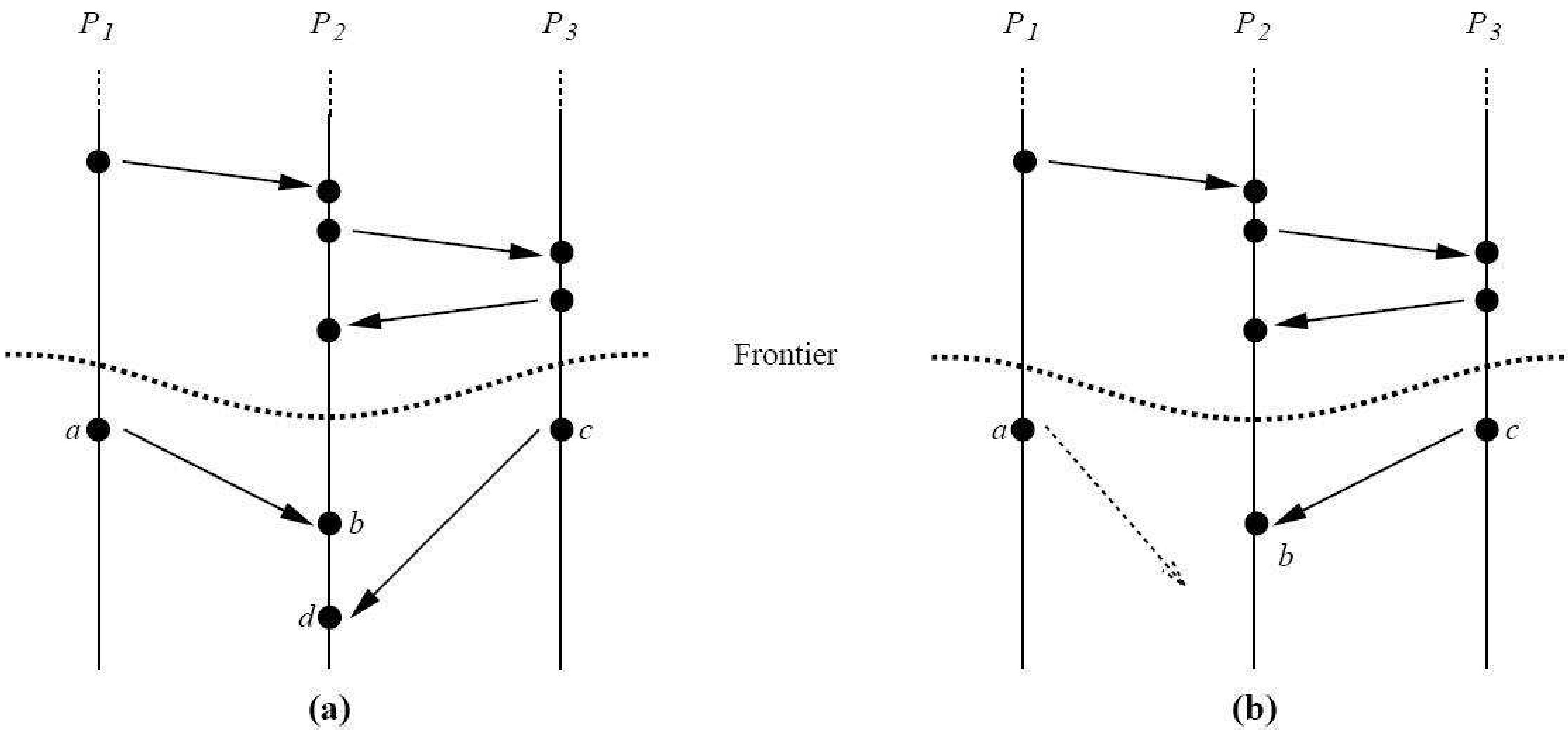
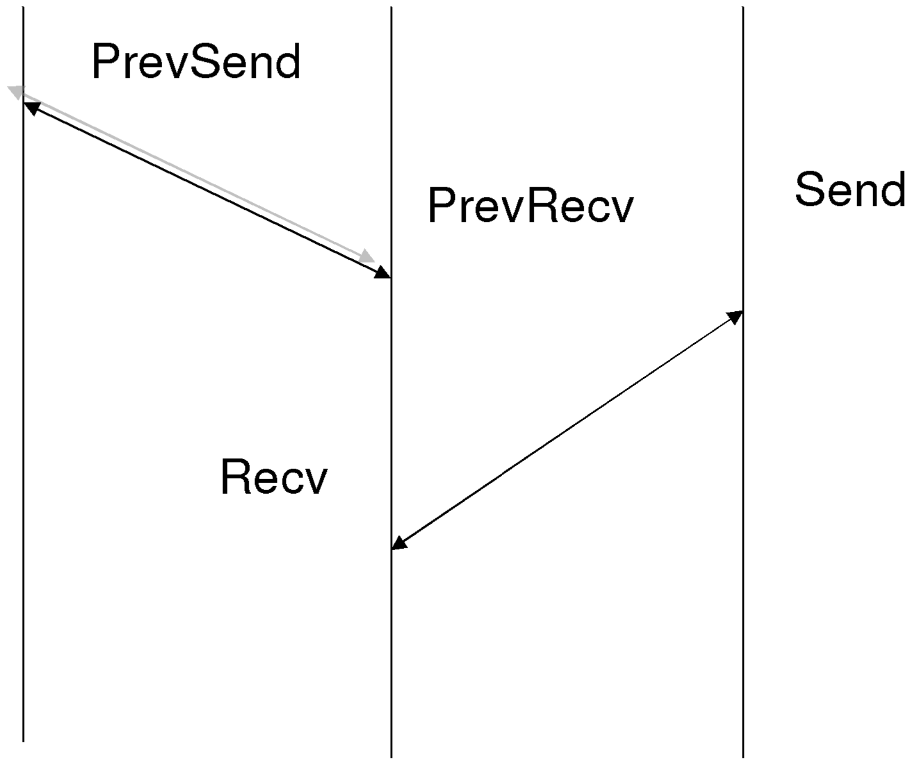



**Figure 2. (a) one possible frontier, (b) the message from *c* could have been received by *b***

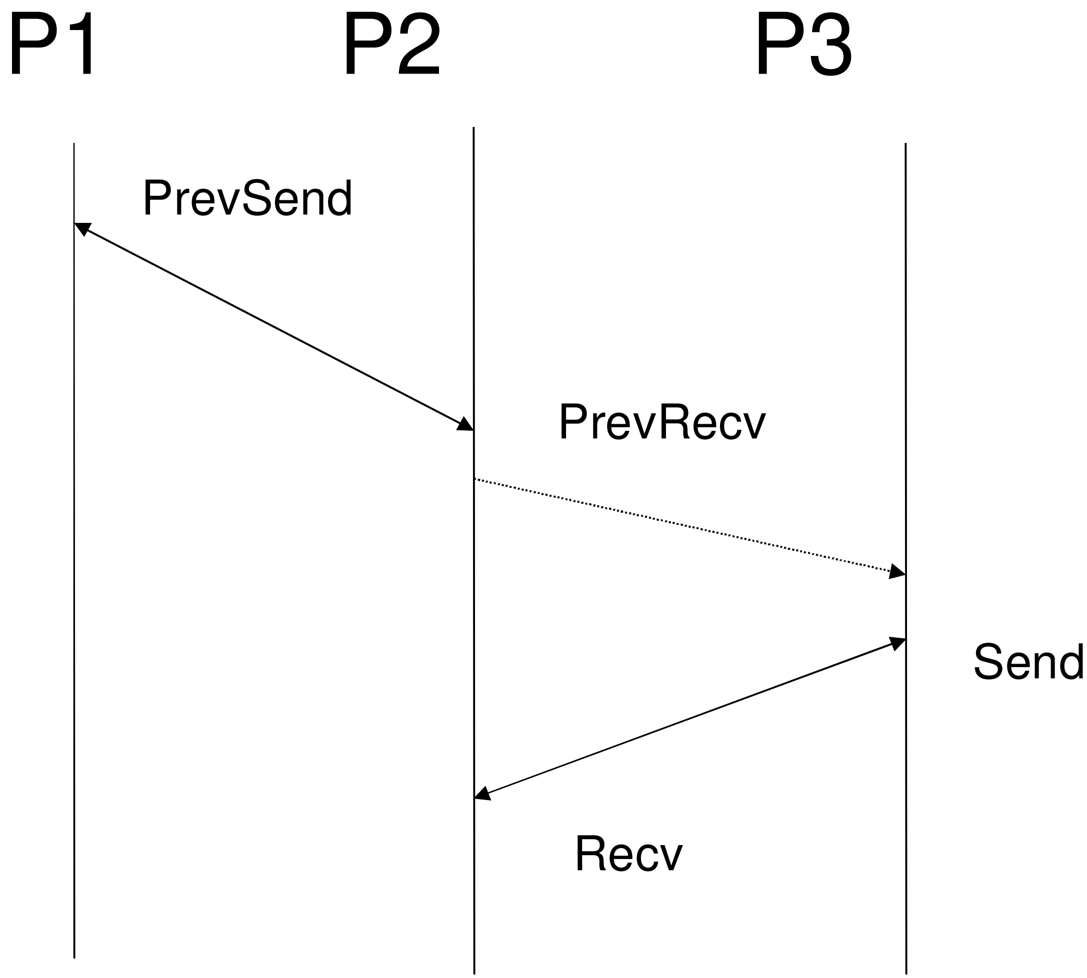
# Race Detection

- Assume that receive is assoc with single process
- Race check after each receive
- Check the order of the sender and a previous received msg.
- If PrevRecv did not happen before Send trace the PrevSend

- Example PrevSend  need to be traced
- P1            P2            P3



- No Trace Needed

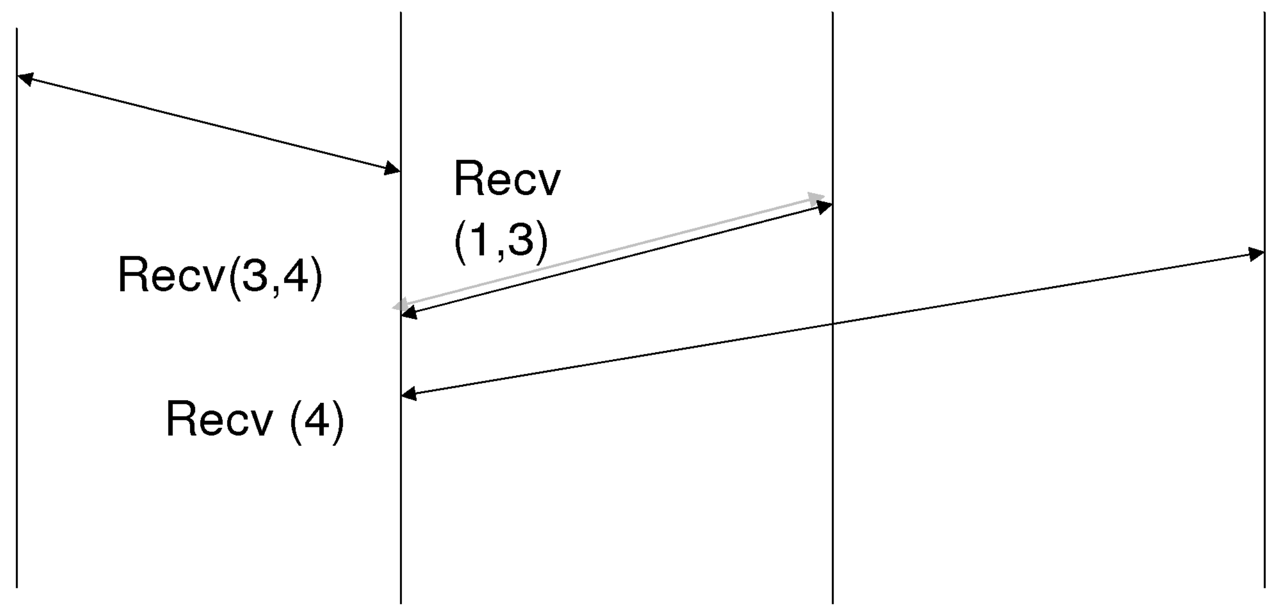


# Replay

- Ensure the delivery of traced messages
- Maintaining a global counter in each process
- increment global after each synchronization operation

# Optimal

- Only if each msg is only involved in one trace (non-transitive)



# Implementation

- Use vector timestamps
- Appended to user messages
- Updated after each Recv operation
- Each proc append the current val to the msg it sends
- Tested on 64-node Thinking Machine and Intel iPSC/2 hypercube

# Results

| Program | Messages Sent | Messages Traced | % of Opt. Trace | Overhead all msgs | Overhead racing msgs |
|---|---|---|---|---|---|
| det‡ | 4713 | 63 (1%) | optimal | 568% | 8% |
| line‡ | 31 | 0 (0%) | optimal | 0.3% | 8% |
| mesh‡ | 10210 | 1392 (14%) | optimal | 28% | 14% |
| mult‡ | 1120 | 0 (0%) | optimal | 15% | 0.1% |
| sys‡ | 9424 | 332 (2%) | optimal | 561% | 14% |
| tycho† | 1791 | 412 (19%) | within 46% | – | – |

† 64-node Thinking Machines CM-5
‡ 32-node Intel iPSC/2

# Conclusion

- Good algorithm when it comes to transitive races.
- Not optimal when it comes to non-transitive races.