Other Optimistic Mechanisms, Memory Management

Richard M. Fujimoto
Professor

Computational Science and Engineering Division
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0765, USA

http://www.cc.gatech.edu/~fujimoto/

Outline

- Dynamic Memory Allocation
- Error Handling
- Event Retraction
- Lazy Cancellation
- Lazy Re-Evaluation
- Memory Management
  - Mechanisms
  - Storage optimal protocols
  - Artificial Rollback
- Other optimistic protocols
- Summary: Conservative and Optimistic Execution

Dynamic Memory Allocation

Issues

- Roll back of memory allocation (e.g., malloc)
  - Memory leak
  - Solution: release memory if malloc rolled back
- Roll back of memory release (e.g., free)
  - Reuse memory that has already been released
  - Solution:
    - Treat memory release like an I/O operation
    - Only release memory when GVT has advanced past the simulation time when the memory was released

Error Handling

- What if an execution error is rolled back?
  - Solution: do not abort program until the error is committed (GVT advances past the simulation time when the error occurred)
  - Requires Time Warp executive to “catch” errors when they occur
- Types of error
  - Program detected
    - Treat “abort” procedure like an I/O operation
  - Infinite loops
    - Interrupt mechanism to receive incoming messages
    - Poll for messages in loop
  - Benign errors (e.g., divide by zero)
    - Trap mechanism to catch runtime execution errors
  - Destructive errors (e.g., overwrite state of Time Warp executive)
    - Runtime checks (e.g., array bounds)
    - Strong type checking, avoid pointer arithmetic, etc.

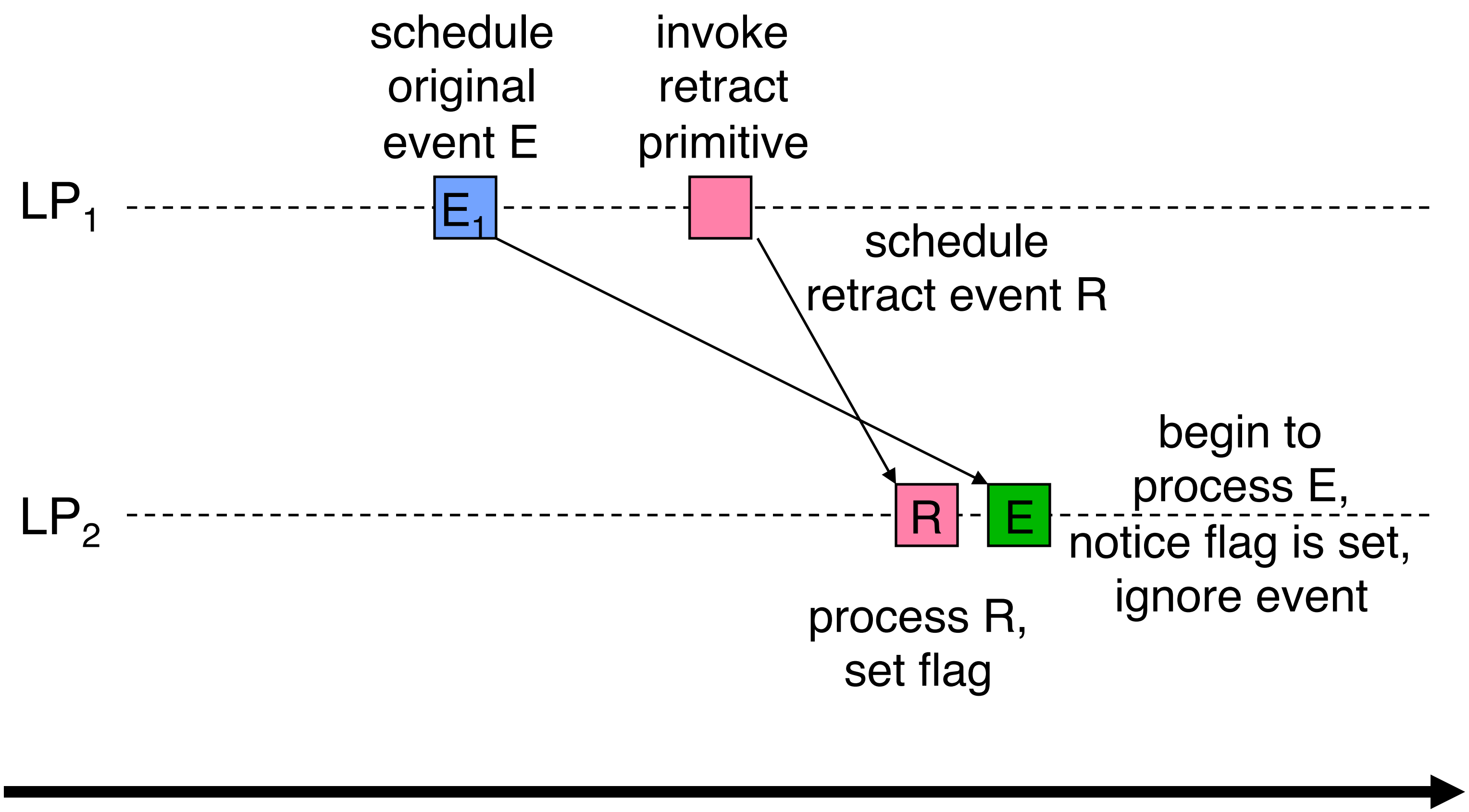
Event Retraction

Unschedule a previously scheduled event

Approach 1: Application Level Approach

- Schedule a retraction event with time stamp < the event being retracted
- Process retraction event: Set flag in LP state to indicate the event has been retracted
- Process event: Check if it has been retracted before processing any event

Example: Application Approach



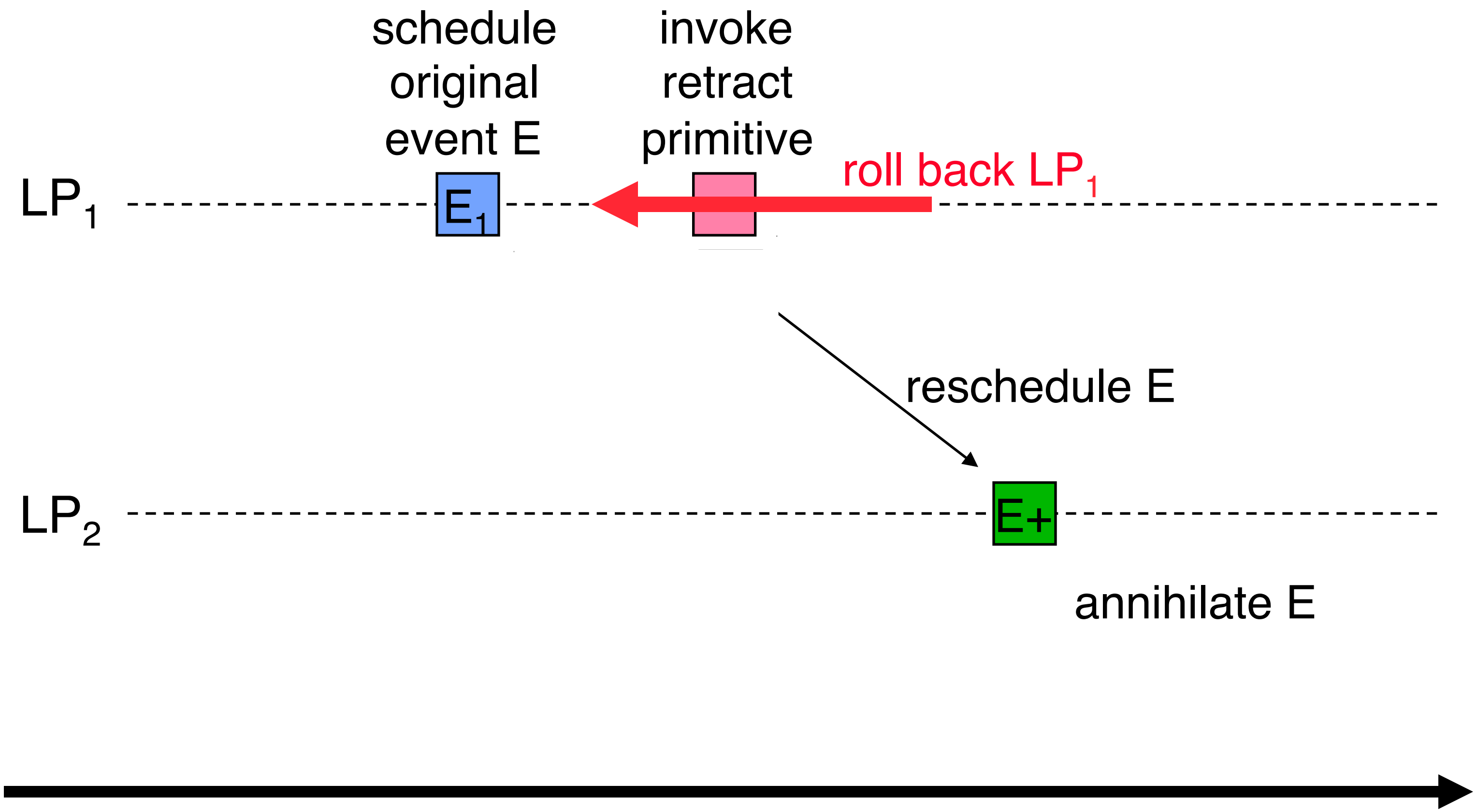
Retraction handled within the application

Event Retraction (cont.)

Approach 2: Implement in Time Warp executive

- Retraction: send anti-message to cancel the retracted event
  - Retraction: invoked by application program
  - Cancellation: invoked by Time Warp executive (transparent to the application)
- Rollback retraction request
  - Reschedule the original event
  - Retraction: place positive copy of message being retracted in output queue
  - Rollback: Send messages in output queue (same as before)

Example: Kernel Approach



Retraction handled within Time Warp executive

Lazy Cancellation

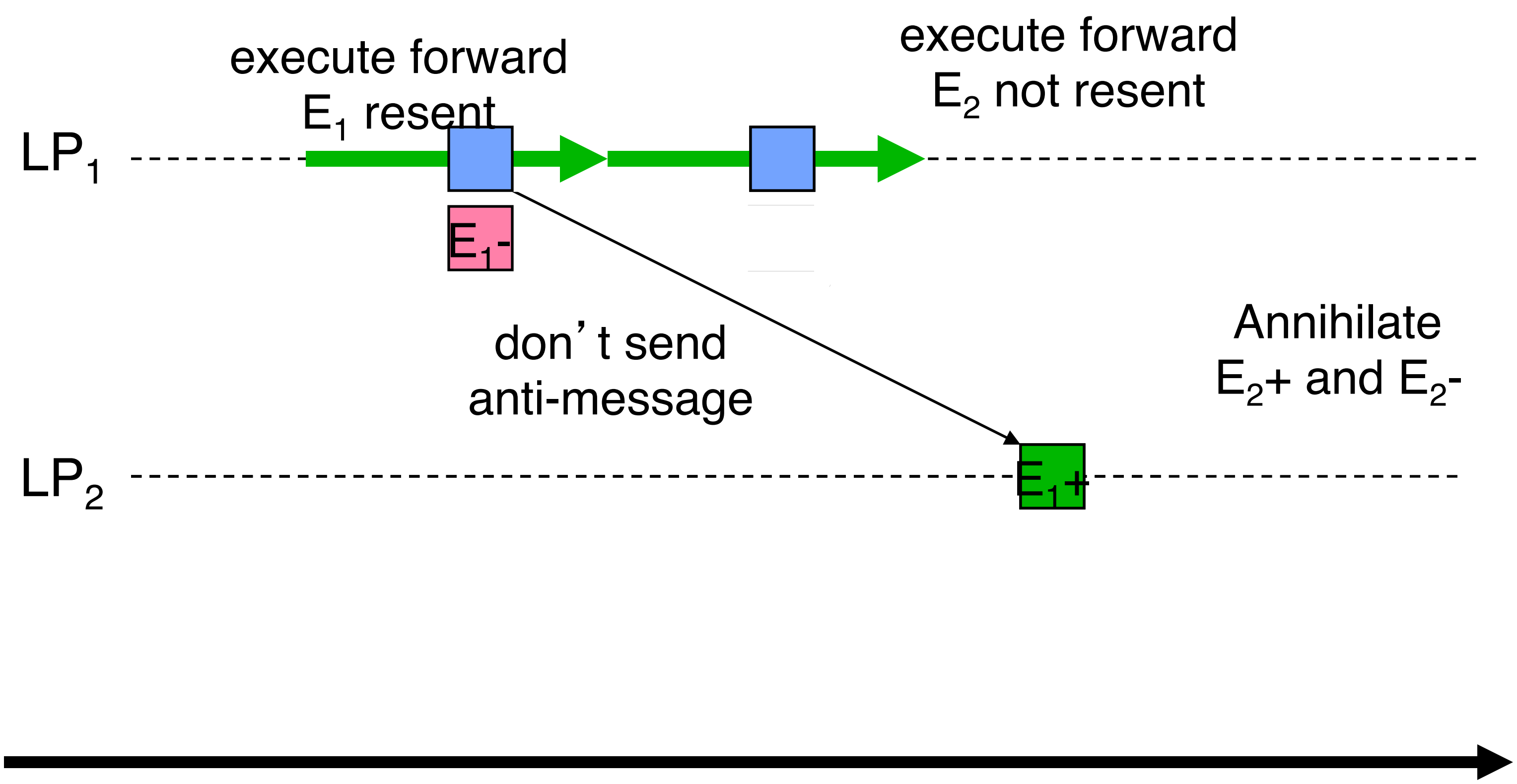
Motivation:

- re-execution after rollback may generate the same messages as the original execution
- in this case, need not cancel original message

Mechanism:

- rollback: do not immediately send anti-messages
- after rollback, recompute forward
- only send anti-message if recomputation does NOT produce the same message again

Example: Lazy Cancellation



Lazy cancellation avoids unnecessary rollback

Lazy Cancellation: Evaluation

Benefit:

- avoid unnecessary message cancellations

Liabilities:

- extra overhead (message comparisons)
- delay in canceling wrong computations
- more memory required

Conventional wisdom

- Lazy cancellation typically improves performance
- Empirical data indicate 10% improvement typical

Lazy Re-evaluation

Motivation:

- re-execution of event after rollback may be produce same result (LP state) as the original execution
- in this case, original rollback was unnecessary

Mechanism:

- rollback: do not discard state vectors of rolled back computations
- process straggler event, recompute forward
- during recomputation, if the state vector and input queue match that of the original execution, immediately “jump forward” to state prior to rollback.

Lazy Re-evaluation

Benefit:

- avoid unnecessary recomputation on rollback
- works well if straggler does not affect LP state (query events)

Liabilities:

- extra overhead (state comparisons)
- more memory required

Conventional wisdom

- Typically does not improve overall performance
- Useful in certain special cases (e.g., query events)

Memory Management in Time Warp

Parallel execution using Time Warp tends to use much more memory than a sequential execution (even with fossil collection)

- State vector and event history
- Memory consumption can be unbounded because an LP can execute arbitrarily far ahead of other LPs

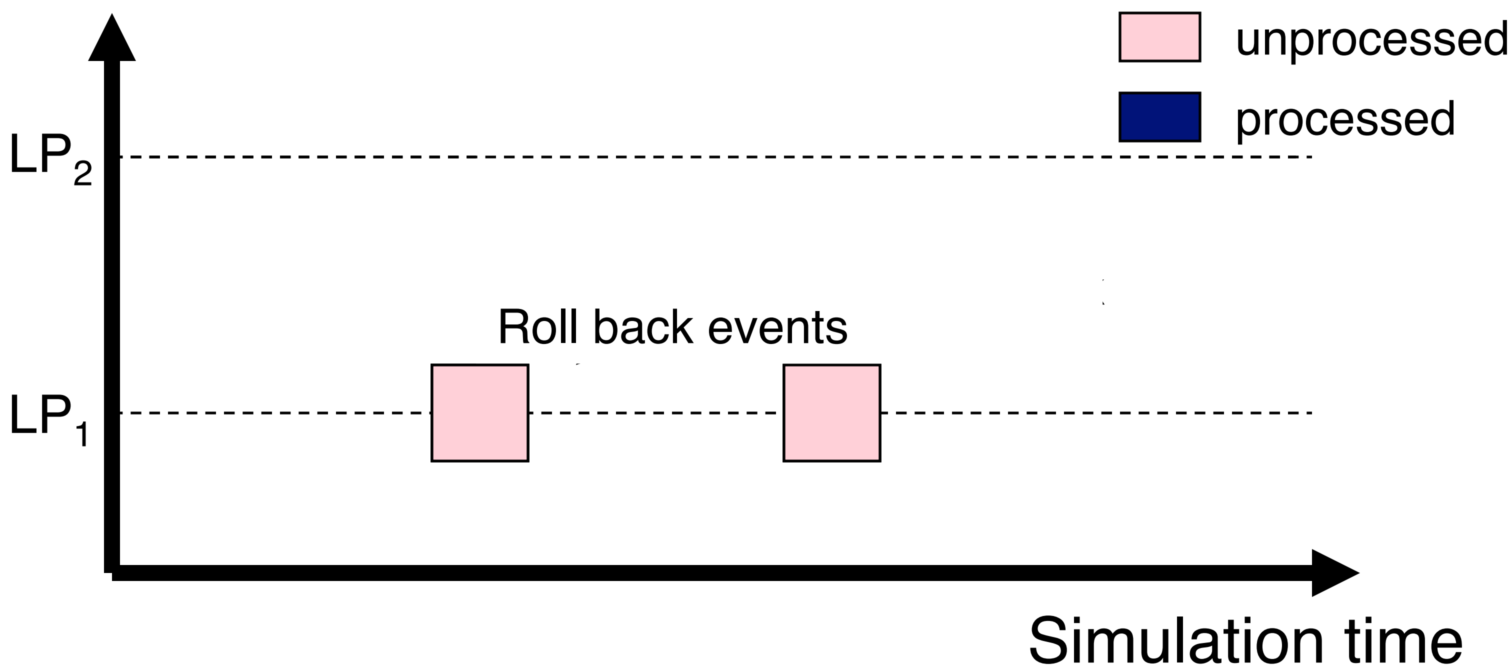
Mechanisms to reduce memory consumption:

- Infrequent / incremental state saving
- Pruning: dynamically release copy state saved memory
- Blocking: block certain LPs to prevent overly optimistic execution
- Roll back to reclaim memory
- Message sendback

Message Sendback

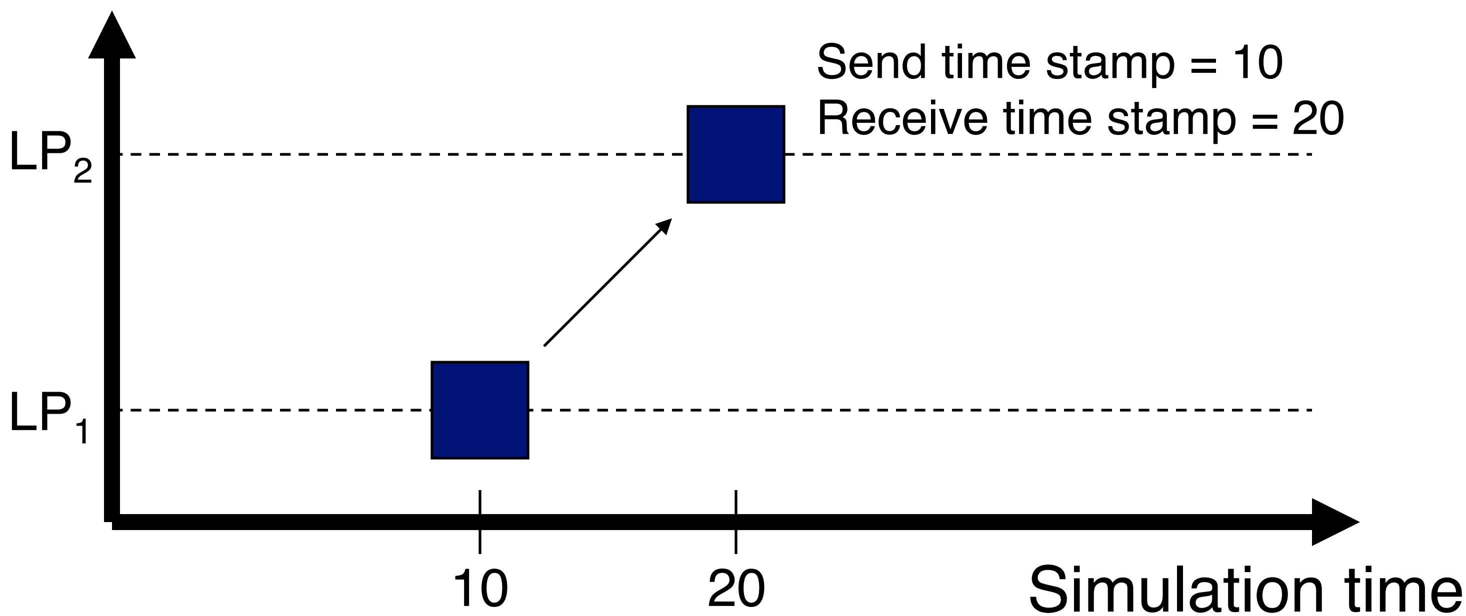
Basic Idea

- Reclaim memory used by a message by returning it to the original sender
- Usually causes the sender to roll back



Event Time Stamps

- Receive time stamp: time stamp indicating when the event occurs (conventional definition of time stamp)
- Send time stamp of event E: time stamp of the LP when it scheduled E (time stamp of event being processed when it scheduled E)



Message Sendback

- Causes sender to roll back to the send time of event being sent back
- Can any message be sent back?
  - No! Can only send back messages with send time greater than GVT
- Also, a new definition of GVT is needed

GVT(T) (GVT at wallclock time T) is the minimum among

- Receive time stamp of unprocessed and partially processed events
- Send time stamp of backward transient messages at wallclock time T

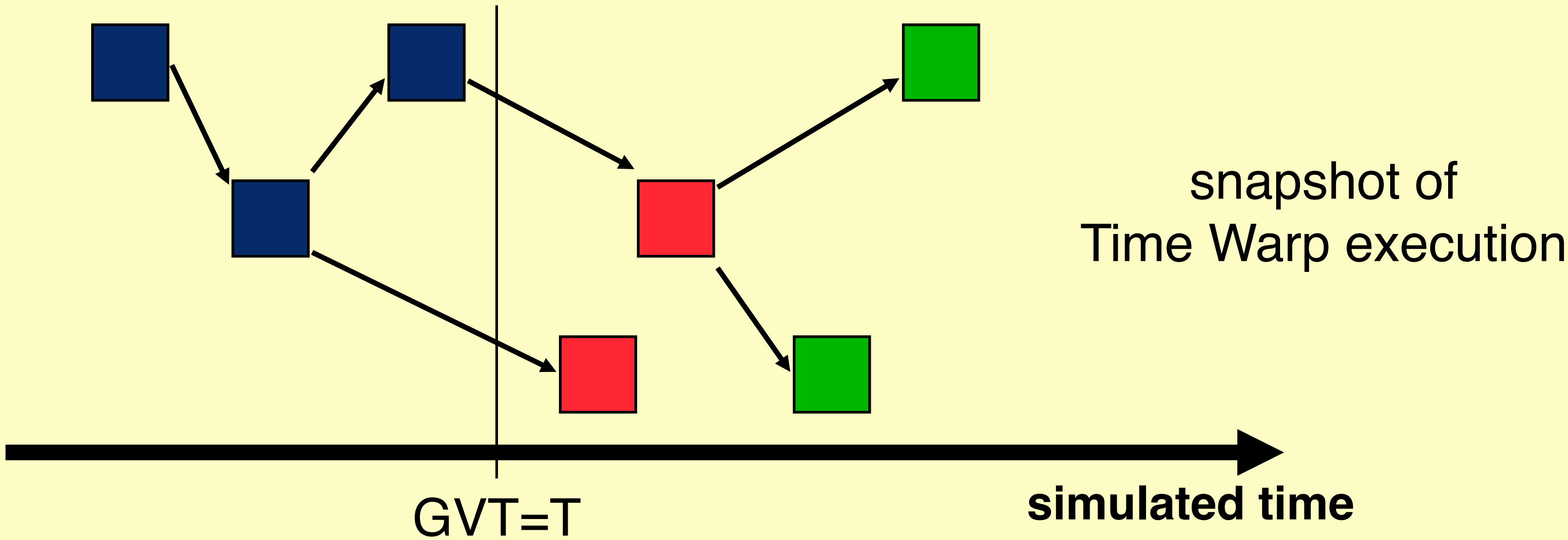
Storage Optimal Protocols

Storage Optimality: A memory management protocol is storage optimal iff it ensures that every parallel simulation uses memory O(M), where M is the number of units of memory utilized by the corresponding sequential simulation.

Basic idea: if the Time Warp program runs out of memory

- identify the events (message buffers) that would exist in a sequential execution at time T, where T is the current value of GVT
- roll back LPs, possibly eliminating (via annihilation) all events except those that exist in the corresponding sequential execution.

Classifying Events



Sequential:
- already processed
- pending, in event list
- not yet generated

Time Warp:
- can be fossil collected
- ineligible for deletion
- eligible for deletion (can be re-created)

Sequential execution: Which events occupy storage in a sequential execution at simulation time T?

Time Warp: For which events can storage be reclaimed?

Observations

- In a sequential execution at simulation time T, the event list contains the events with
  - Receive time stamp greater than T
  - Send time stamp less than T.
- Time Warp can restore the execution to a valid state if it retains events with
  - Send time less than GVT and receive time stamp greater than GVT.
  - All other events can be deleted (as well as their associated state vector, anti-messages, etc.)
- Storage optimal protocols: roll back LPs to reclaim all memory not required in corresponding sequential execution

Artificial Rollback

Salvage parameter: Amount of memory to be reclaimed when a processor runs out of memory

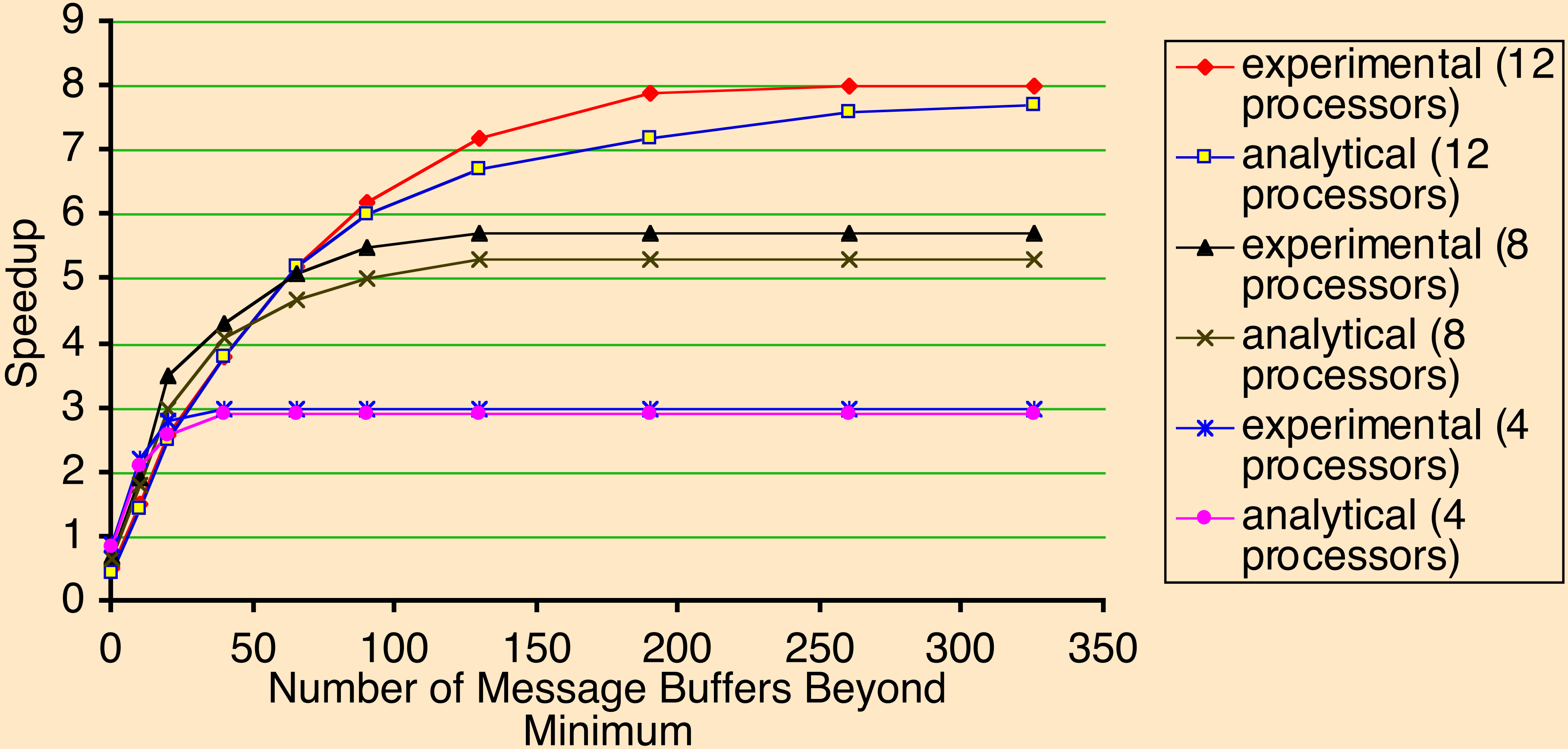
When system runs out of memory

- Sort LPs, in order of their current simulation time (largest to smallest): LP_1, LP_2, LP_3, …
- Roll back LP_1 to current simulation time of LP_2
- If additional memory must be reclaimed, roll back LP_1 and LP_2 to current simulation time of LP_3
- Repeat above process until sufficient memory has been reclaimed

Artificial rollback is storage optimal when executed on a shared memory multiprocessor with a shared buffer pool

Performance will be poor if too little memory is available

Effect of Limited Memory on Speedup



- symmetric synthetic workload (PHold)
- one logical processor per processor
- fixed message population
- KSR-1 multiprocessor
- sequential execution requires 128 (4 LPs), 256 (8 LPs), 384 (12 LPs) buffers

Other Optimistic Algorithms

Principal goal: avoid excessive optimistic execution

A variety of protocols have been proposed, among them:

- window-based approaches
  - only execute events in a moving window (simulated time, memory)
- risk-free execution
  - only send messages when they are guaranteed to be correct
- add optimism to conservative protocols
  - specify "optimistic" values for lookahead
- introduce additional rollbacks
  - triggered stochastically or by running out of memory
- hybrid approaches
  - mix conservative and optimistic LPs
- scheduling-based
  - discriminate against LPs rolling back too much
- adaptive protocols
  - dynamically adjust protocol during execution as workload changes

Summary

- Other Mechanisms
  - Simple operations in conventional systems (dynamic memory allocation, error handling) present non-trivial issues in Time Warp systems
  - Solutions exist for most, but at the cost of increased complexity in the Time Warp executive
- Event retraction
  - Not to be confused with cancellation
  - Application & kernel level solutions exist
- Optimizations
  - Lazy cancellation often provides some benefit
  - Conventional wisdom is lazy re-evaluation costs outweigh the benefits

Conservative Algorithms

Pro:

- Good performance reported for many applications containing good lookahead (queueing networks, communication networks, wargaming)
- Relatively easy to implement
- Well suited for "federating" autonomous simulations, provided there is good lookahead

Con:

- Cannot fully exploit available parallelism in the simulation because they must protect against a "worst case scenario"
- Lookahead is essential to achieve good performance
- Writing simulation programs to have good lookahead can be very difficult or impossible, and can lead to code that is difficult to maintain

Optimistic Algorithms

Pro:

- good performance reported for a variety of application (queuing networks, communication networks, logic circuits, combat models, transportation systems)
- offers the best hope for “general purpose” parallel simulation software (not as dependent on lookahead as conservative methods)
- “Federating” autonomous simulations
  - avoids specification of lookahead
  - caveat: requires providing rollback capability in the simulation

Con:

- state saving overhead may severely degrade performance
- rollback thrashing may occur (though a variety of solutions exist)
- implementation is generally more complex and difficult to debug than conservative mechanisms; careful implementation is required or poor performance may result
- must be able to recover from exceptions (may be subsequently rolled back)