Error Types

- Program detected
  - Logic errors, e.g., some variables never negative
  - Treat "abort" procedure like an I/O operation, prevent error from propagating and flag error to see if it erased by rollback.
- Infinite loops
  - Interrupt mechanism to receive incoming messages
  - Poll for messages in loop
- Benign errors
  - Errors that impact only checkpointed state (e.g., divide by zero)
  - Trap mechanism to catch runtime execution errors
- Destructive errors
  - Difficult to detect these...
  - Example: overwrite state of Time Warp executive
  - Runtime checks (e.g., array bounds)
  - Strong type checking, avoid pointer arithmetic, etc.

Dynamic Memory Allocation

`malloc()` and `free()` How should they be handled?

**Issues:**

- Roll back of memory allocation (e.g., `malloc()`)  
  - **Problem:** Memory leak (when check pointing a pointer to a previously allocated memory location). Run out of memory...
  - **Solution:** release memory if malloc rolled back
- Roll back of memory release (e.g., `free()`)  
  - **Problem:** Reuse memory that has already been released. The LP did not really mean to free the memory ...
  - **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
Event Retraction

- **Goal:**
  - Need a primitive to un-schedule a previously scheduled event (application level primitive)

- **Example:**
  - ORD Schedules an arrival at JFK
  - Need to re-route aircraft to Boston (due to congestion at JFK)

- **Observation:**
  - Cancellation retracts events at the ‘simulation kernel level’

- **Problem:**
  - Need a mechanism to *undo* event retraction (cancellation) if the event computation that invoked the retraction is rolled back.

---

Event Retraction Approaches

- **Application Level**
- **Kernel Level**

---

Event Retraction: Approach 1

**Application Level Approach**

1. Schedule a retraction event with time stamp earlier than (<) the event being retracted
2. Process retraction event: Set flag in LP state to indicate the event has been retracted.
3. Process event: Check if it has been retracted before processing any event

---

Event Retraction: Approach 2

**Time Warp Executive Level Approach**

- **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: Application Approach

![Diagram showing the process of retraction handled within the application]

Example: Kernel Approach

![Diagram showing the process of 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 that were scheduled by rolled back event.

- **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

Lazy Cancellation: Evaluation

- **Benefit:**
  - avoid unnecessary message cancellations which in turn eliminate secondary rollbacks.

- **Liabilities:**
  - extra overhead (message comparisons)
  - delay in canceling wrong computations
    - may allow erroneous computations to spread further. -> more computations may need to rollback when anti-message is finally sent
  - more memory required (delayed memory reclamation)

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

Lazy Re-evaluation

- **Motivation:**
  - re-execution of event after rollback may 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

- **Benefit:**
  - 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
    - “Overoptimism” lead to very long and frequent rollbacks, may waste computation time.
Memory Management in Time Warp

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)

Memory Consumption Remedies

- **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**

Memory Consumption

- **Sequential Simulations:**
  - aborts
- **Parallel Simulations:**
  - abort?
  - more memory?
  - blocking?
  - Memory:
    1. positive and
    2. anti-messages and
    3. state vectors

Message Sendback

Basic Idea:

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

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
- 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.

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.

Other Memory Mechanisms

- Prune-back
- Adaptive mechanisms: predicts memory the program needs on-line
- Trading performance and Memory
  - Performance may DECREASE if memory is increased further – poorly balanced workloads
  - Limiting memory may provide a flow control mechanism that avoids overoptimistic 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?

Cancelback

- Shared memory machine mechanism
- Storage optimal
- Global pool to hold free buffers
- Uses Message Sendback mechanism (message TS > GVT)
- Requires:
  - GVT Computation
  - Fossil collection
  - Find and eligible event
  - Send back mechanism
- Batching – into a salvage parameter

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
- 25% to 75% extra buffer and beyond minimum did not improve performance
Performance Hazards

- **Chasing Down Incorrect Computations**
  - Incorrect computation spreads while correcting/canceling erroneous computations
  - Dog chasing its tail
- **Rollback Echoes**
  - Expensive rollbacks may cause length of rollback to expand at an exponential rate. Cost of rollback:
    1. Antimessage to all cancelled events
    2. Restore State
    3. Pointer updates of input queue
  - 1&2 suggest cost is proportional to #events being rolled back
  - What if rolling back T units of simulated time takes twice as long as going forward by the same amount?
    - Net rate of GVT progress decreases as the simulation proceeds!

Other Optimistic Algorithms

**Principal goal:** avoid excessive optimistic execution

A variety of protocols have been proposed:

- **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

Other Optimistic Algorithms

**Principal goal:** avoid excessive optimistic execution

A variety of protocols have been proposed:

- **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

Conservative Algorithms

**Advantages:**

- Good performance reported for many applications containing good lookahead (queuing networks, communication networks, war gaming)
- Relatively easy to implement
- Well suited for “federating” autonomous simulations, provided there is good lookahead

**Disadvantages:**

- 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

**Advantages:**

- Good performance reported for a variety of applications
  - Queuing networks, communication networks, logic circuits, combat models
- 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

**Disadvantages:**

- State saving overhead may severely degrade performance
- Rollback thrashing may occur (though a variety of solutions exist)

Implementation:

- 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)

Summary

- **Other Mechanisms**
  - Simple operations in conservative 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