Real Time Fault Tolerant CORBA

Michael Melliar-Smith
Louise Moser

Eternal Systems, Inc.

What Is Fault Tolerance

- **Murphy's Law of Fault Tolerance:**
  - **The only thing that is certain is that the system is going to fail**
- The best that we can do is to reduce the probability of failure **(but not to zero)**

What Is Real Time

- Hard Real Time
  - Results delivered, and inputs read, at specific predetermined times
  - Events occur at specific times
  - Processing and communication are time bounded
  - Fault recovery is scheduled as part of normal processing
  - Variability is hidden beneath the schedule
- Soft Real Time
  - The probability of meeting real-time deadlines is high
  - Activities are event triggered
  - Processing and communication can take variable times
  - Priority, deadline or least-laxity scheduling can be used
  - Recovery is part of processing variability
  - Variability must be handled by the scheduler to meet deadlines

Even for Hard Real Time Fault Tolerance,
meeting the deadline is still probabilistic

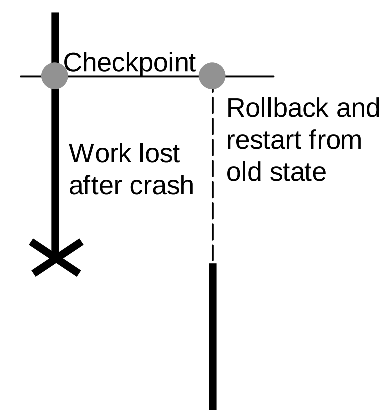
eternal™ SYSTEMS

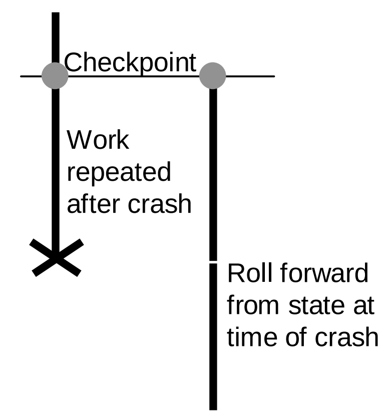
Types of Faults

- **Fail Stop or Crash**
  - Correct results up to fault – then nothing
- **Timing**
  - Some results are correct but delayed
- **Omission**
  - Some results are not generated while others are
- **Commission**
  - Some results are wrong values
- **Malicious**
  - Wrong results deliberately chosen to damage the system

Replication-Based Fault Tolerance

- Application objects or processes are replicated
  - Replicas are located on multiple distinct processors
- Replication protects both processing and data
  - Data are not lost
  - Processing is not lost
  - Messages are not lost
- Roll forward recovery rather than rollback/abort
  - Clients never see the fault or recovery
  - Messages are never retracted
  - Application continues as if no fault had occurred
  - But slow recovery from fault because processing must be repeated





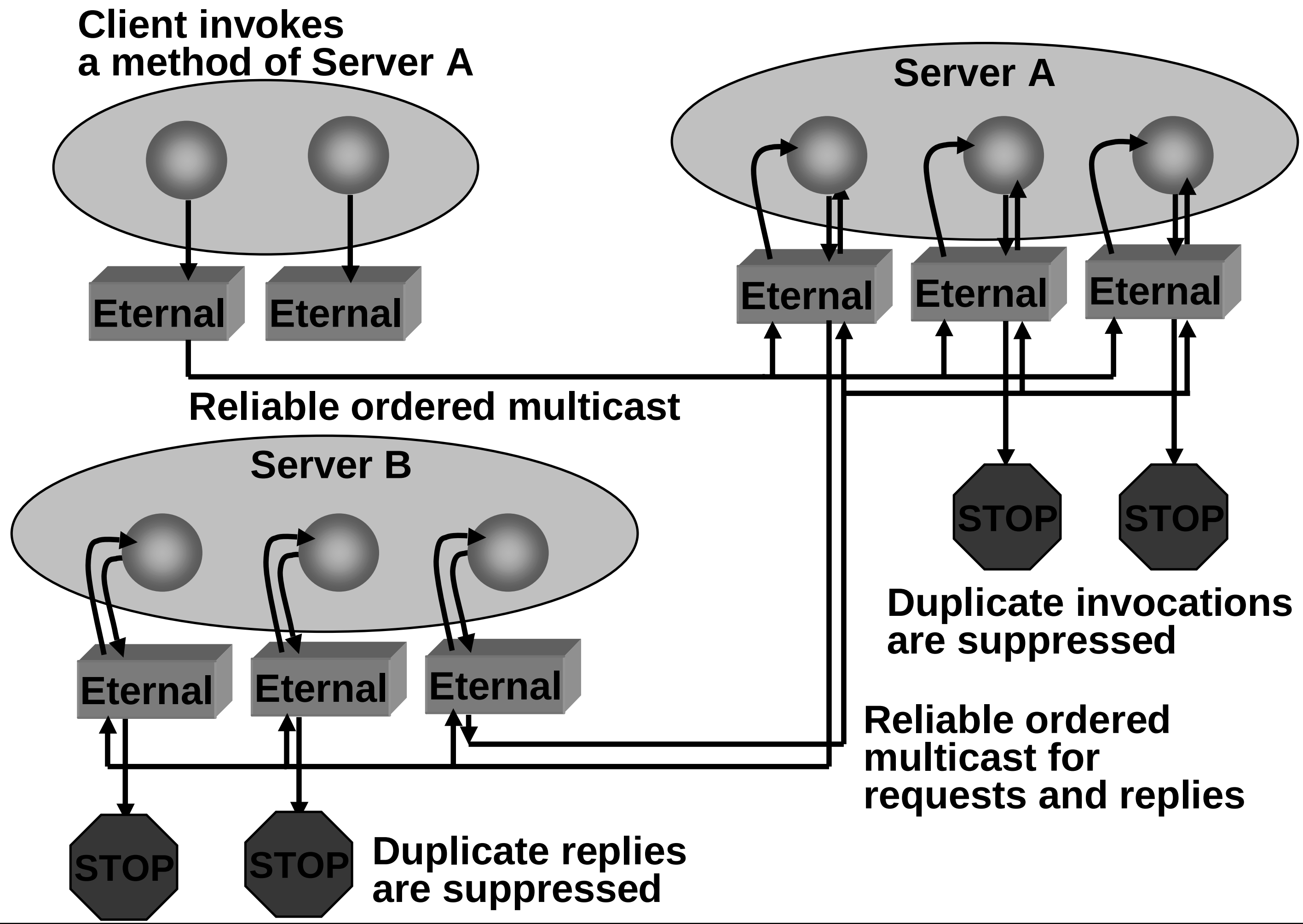
Application Transparency

- **Replication Transparency**
  - Application does not know that processes or objects are replicated
- **Fault Transparency**
  - Application does not know that faults have occurred
- Application programs can be rendered fault-tolerant with minimal modification
- Fault-tolerant application programming is simpler and quicker

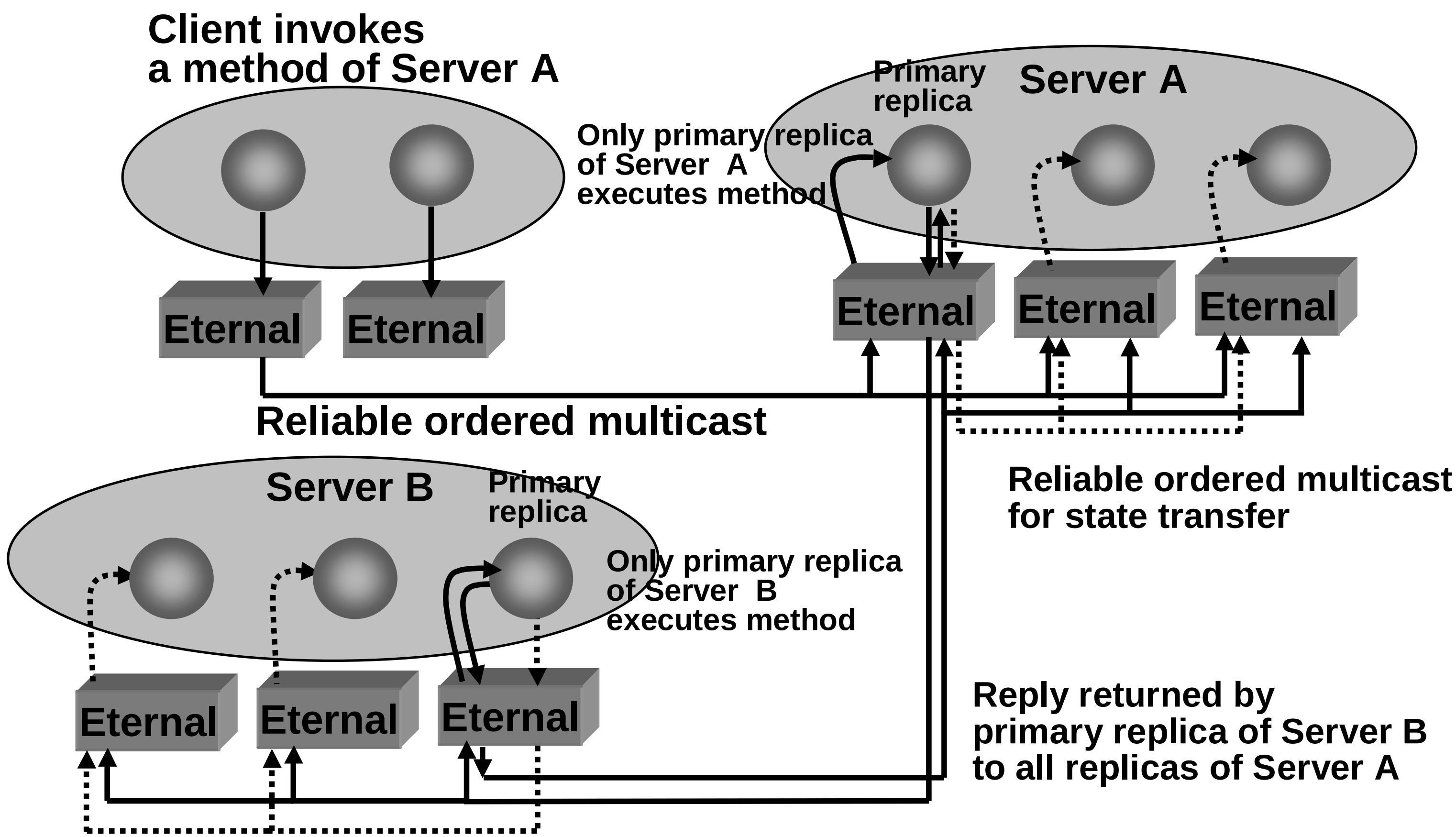
Replication Policies

- **Active Replication**
  - All replicas execute each request
  - Fastest recovery from faults
  - Requires synchronization between replicas
- **Passive Replication**
  - Only one replica (the primary) executes each request
  - Other replicas are available as backups
    - **Warm passive** – Program code is loaded into backups but backups do not execute requests, primary transfers its state periodically to backups
    - **Cold passive** – Program code is not even loaded into backups
  - Lower memory and processing costs
  - Slower recovery from faults
- **Proactive Replication**
  - Primary and all backup replicas execute each request
  - Primary replica determines order of operations
  - Transmits order to backup replicas

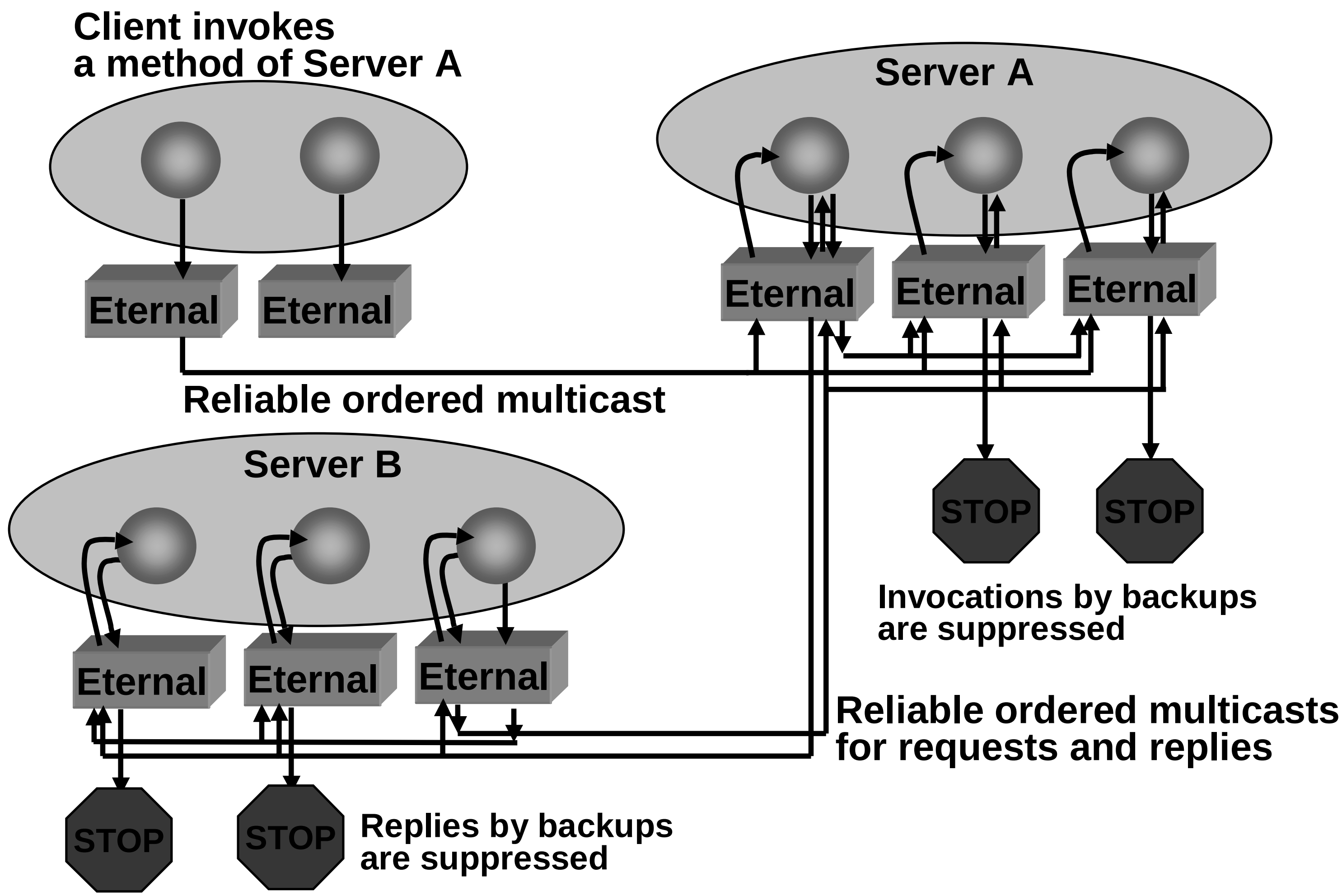
Active Replication



Passive Replication



Proactive Replication



Strong Replica Consistency

- Strong Replica Consistency
  - Means that all replicas have the same state at the same logical time
  - Requires that all replicas exhibit the same deterministic behavior
- Strong Replica Consistency greatly simplifies application design
  - Lower costs
  - Faster time-to-market

  But requires
  - Virtually deterministic behavior of the application processes or objects
  - Strong mechanisms in the fault tolerance infrastructure
- Sources of Replica Non-Determinism
  - Message order
  - Multithreading and mutex locking
  - Sockets and select
  - Time and timeouts

Strong Replica Consistency

- **Active Replication**
  - After each operation that they perform, all of the replicas of a process or object have the same state
- **Passive Replication**
  - After each checkpoint and replay of messages, all of the replicas of a process or object have the same state
- **Proactive Replication**
  - Ensure Strong Replica Consistency by masking or sanitizing replica non-determinism

eternal™ SYSTEMS

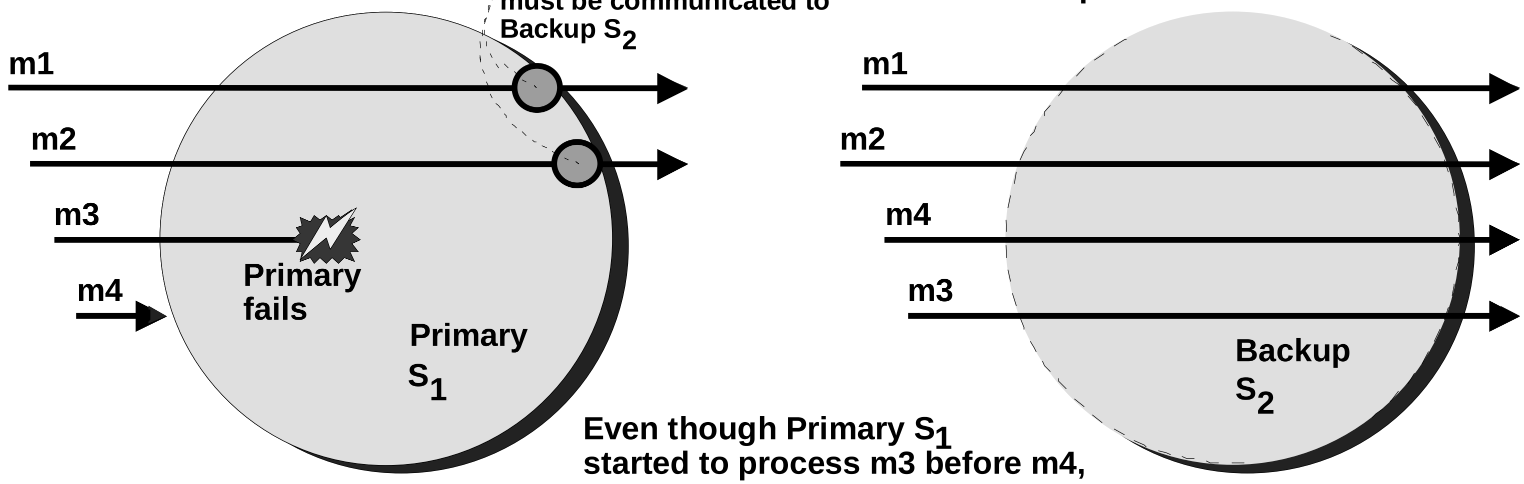
Proactive Replication

- Both Primary and Backup replicas process all requests
- Primary replica leads, processes requests first
  - Makes decisions on order in which messages are processed, mutexes are claimed, values of local functions such as gettimeofday(), rand(), etc
  - Communicates decisions to backup replicas
- Backups process requests as directed by decisions that they receive from Primary
  - Backups lag behind Primary by a few milliseconds
- Checkpoints needed only to bring up a new backup replica
- Performance tradeoffs
  - Fast response time for requests
  - Faster recovery from faults
  - Higher processing load
  - Lower communication load

Consistent Message Ordering

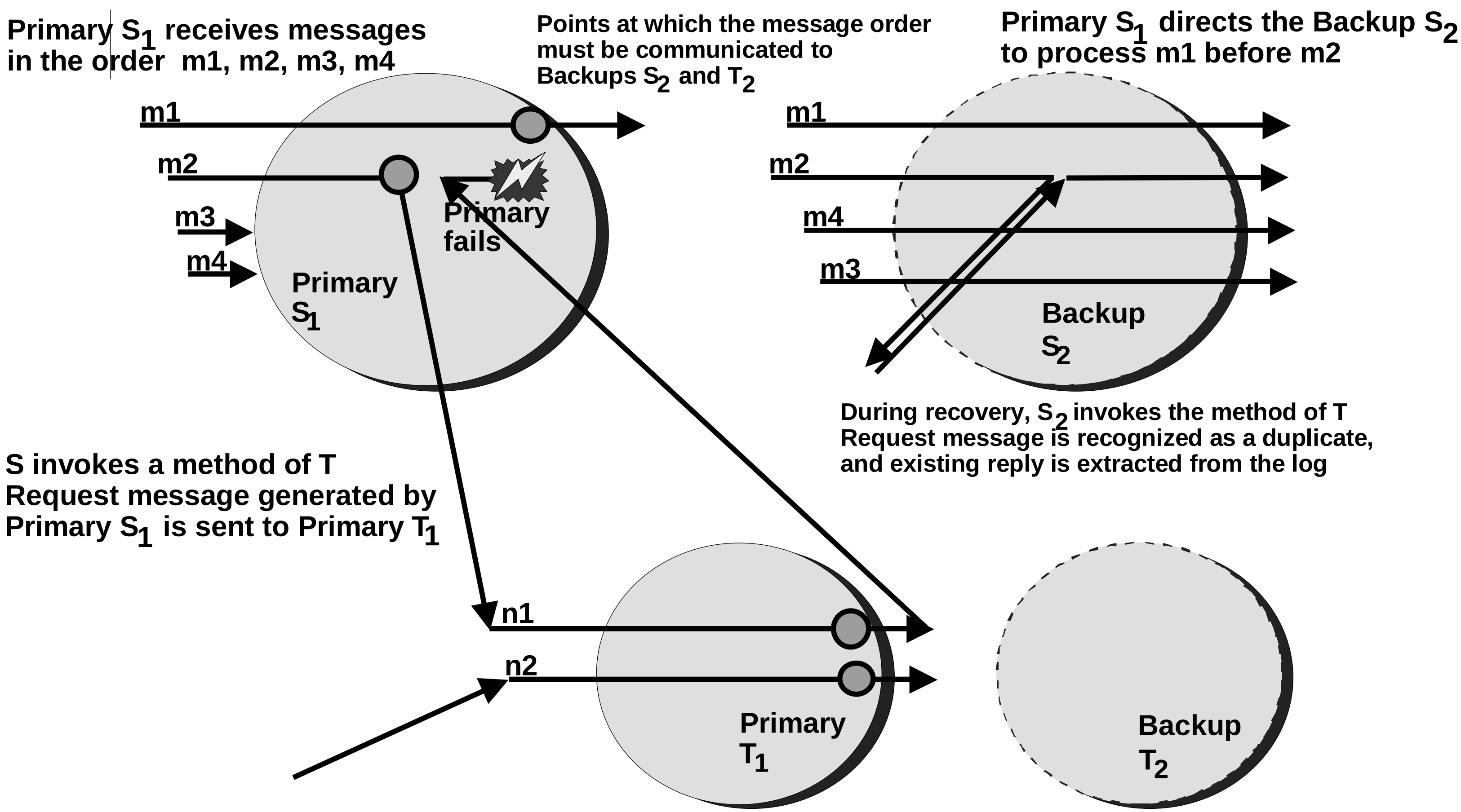
Primary S_1 receives messages in the order m1, m2, m3, m4

Backup S_2 receives messages in the order m2, m1, m4, m3
Primary S_1 directs the Backup S_2 to process m1 before m2



Even though Primary S_1 started to process m3 before m4, no results of processing m3 were communicated to any other object
Consequently, it is acceptable for the Backup S_2 to process m4 before m3

Consistent Message Ordering



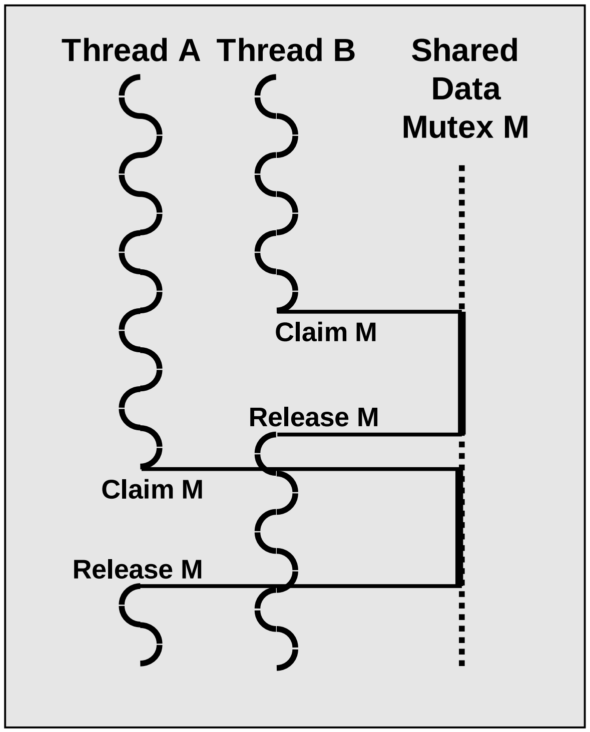
Multithreading

- Several threads within a process execute concurrently
- Threads can share data, provided that they are protected by mutexes
- Multithreading presents challenges for
  - Consistent operation of the replicas
  - Checkpointing the state of the replicas
- In Primary, mutexes are granted to threads in whatever order the mutex library grants them
- In Backup, mutexes must be granted to threads in exactly the same order as they are granted in Primary

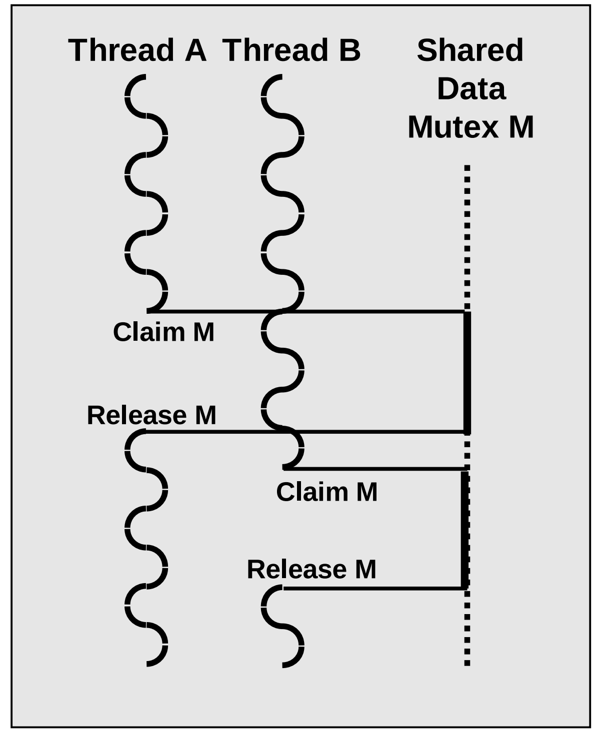
eternal SYSTEMS

Inconsistent Multithreading

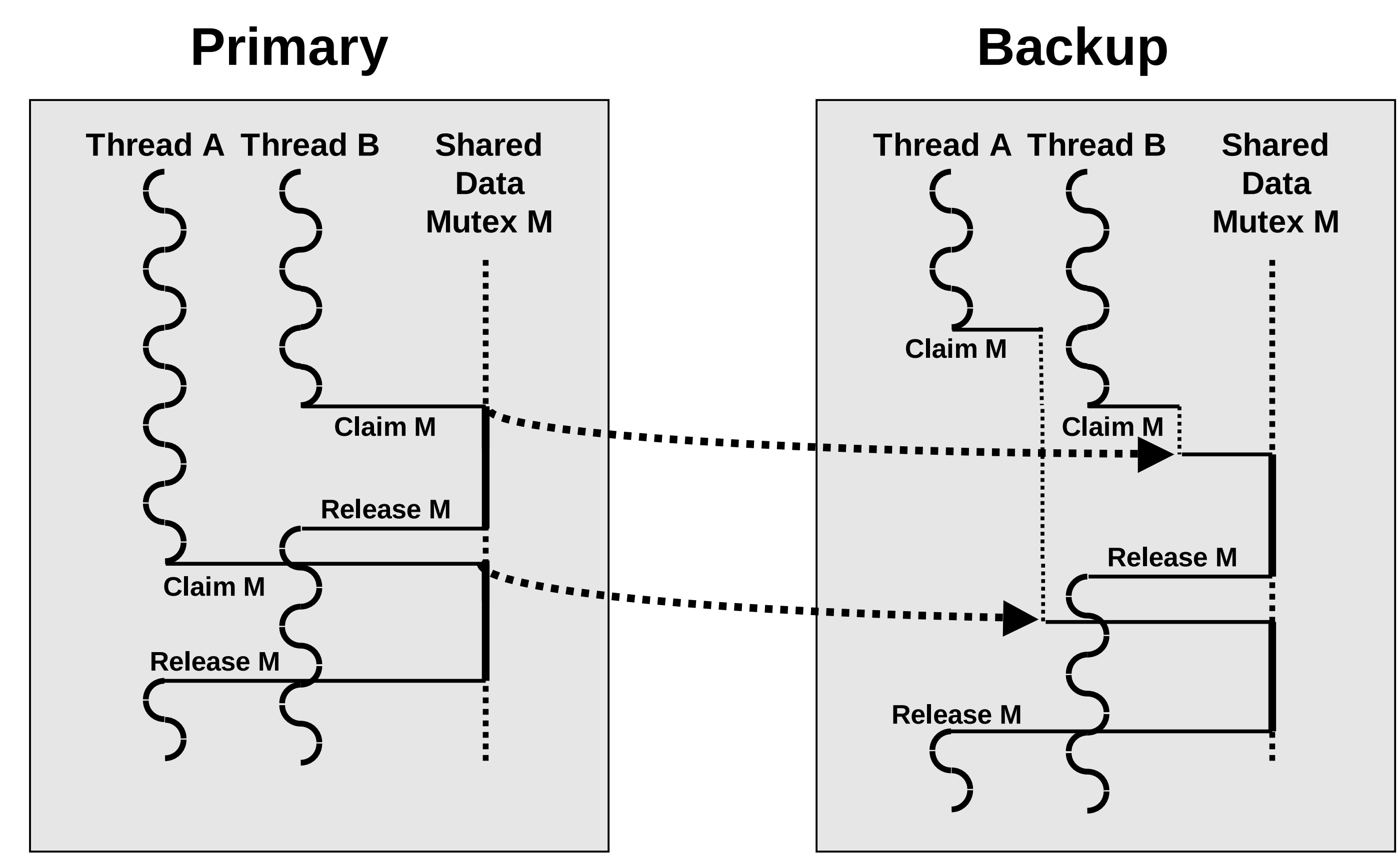
Primary



Backup



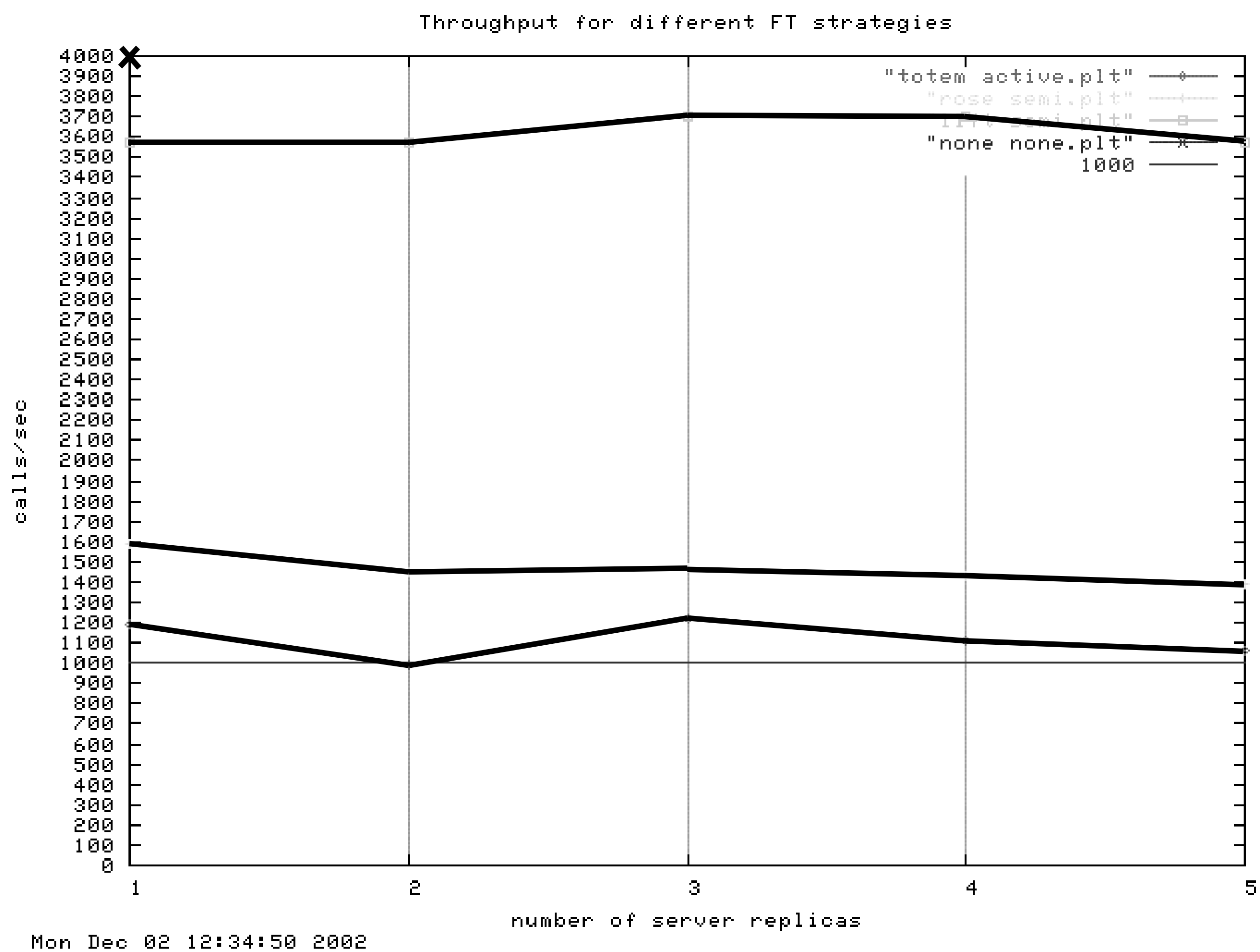
Consistent Multithreading



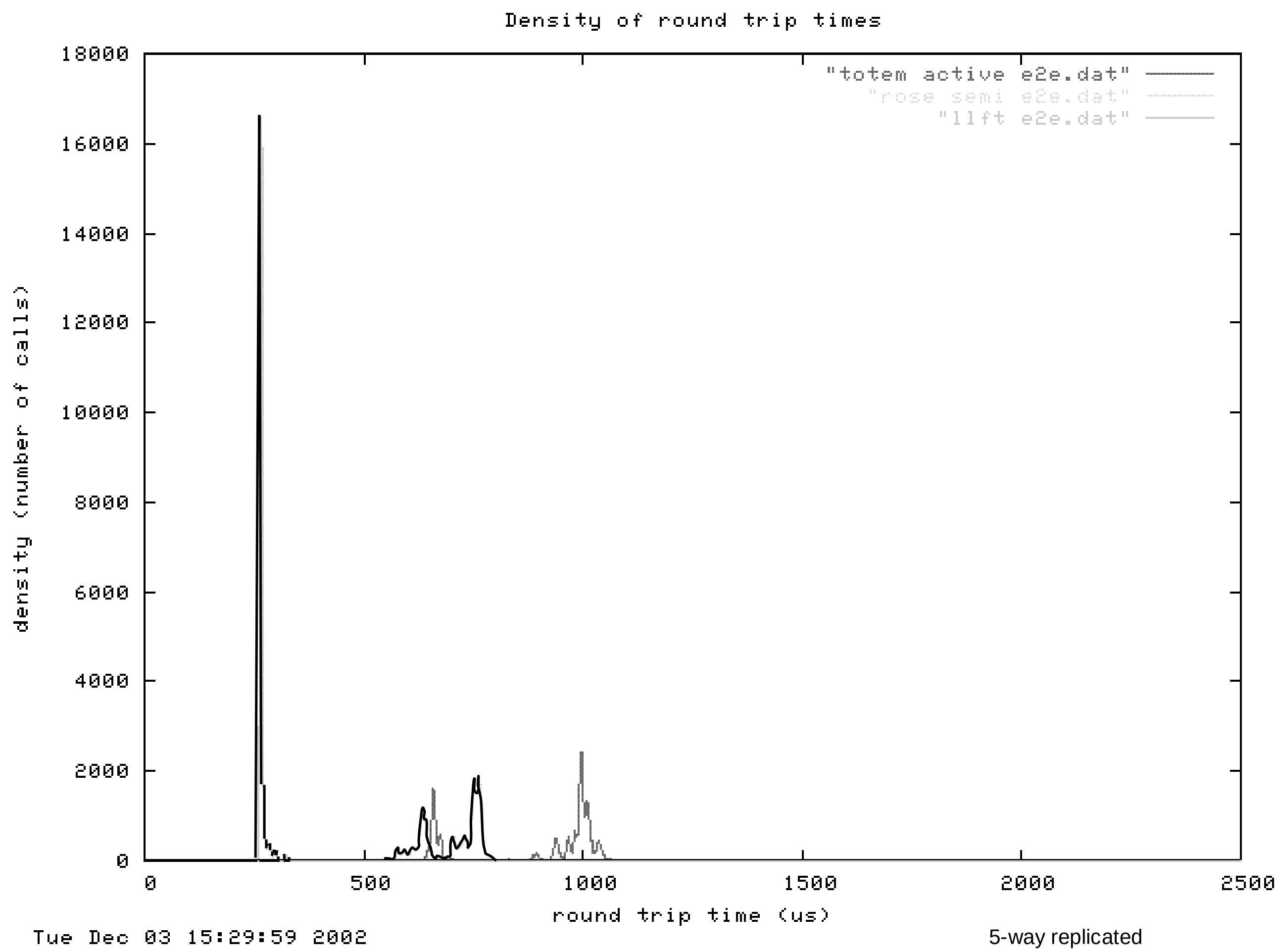
Performance Characteristics

- Eternal Systems' Real Time Fault Tolerant CORBA Infrastructure has excellent performance
- Under fault-free conditions
  - Exhibits low end-to-end latency
  - Exhibits predictable and deterministic behavior (i.e., little variation in the mean latency)
  - Respects the real-time scheduling of the operating system, e.g., priorities of threads and messages
- When a fault occurs
  - Continues to provide service to the clients without interruption
  - No loss of messages, data or processing
  - Exhibits short recovery times, e.g. to promote Backup to become new Primary
  - The fault is hidden from the clients

Throughput


Throughput for different FT strategies
calls/sec
4000
3900
3800
3700
3600
3500
3400
3300
3200
3100
3000
2900
2800
2700
2600
2500
2400
2300
2200
2100
2000
1900
1800
1700
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
1
2
3
4
5
number of server replicas
"totem active.plt"
"rose semi.plt"
"none none.plt"
1000
Mon Dec 02 12:34:50 2002

Probability Density for Latency



5-way replicated

Conclusion

- Real Time Fault Tolerance is difficult to provide
  - Replication is necessary to achieve short and predictable recovery times
  - Replication infrastructure must respect real-time scheduling priorities of the operating system
  - End-to-end latency (request/reply response time) seen by the clients must be short and predictable
  - Strong Replica Consistency is necessary to minimize the complexity of the application programs
  - Masking non-determinism is necessary for strong replica consistency
- Eternal Systems' Real Time Fault Tolerant CORBA infrastructure with Proactive replication satisfies these requirements

Contact Information

Any questions?

Thank you!

Michael Melliar-Smith
5290 Overpass Road, Building D
Santa Barbara, CA 93111
805 696-9051 X251
pmms@eternal-systems.com

eternal™ SYSTEMS