

# Lightweight Fault-Tolerance

Lorenzo Alvisi

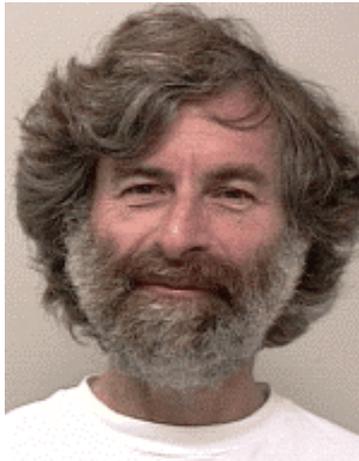
The University of Texas at Austin

# Why Fault-Tolerance?

---

“A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable.”

Leslie Lamport, May 1987



# Fault-Tolerance: The Good-Old Days

---

## Target:

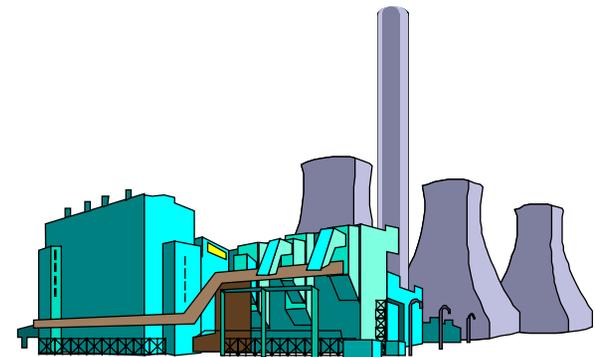
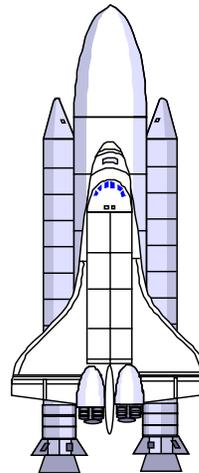
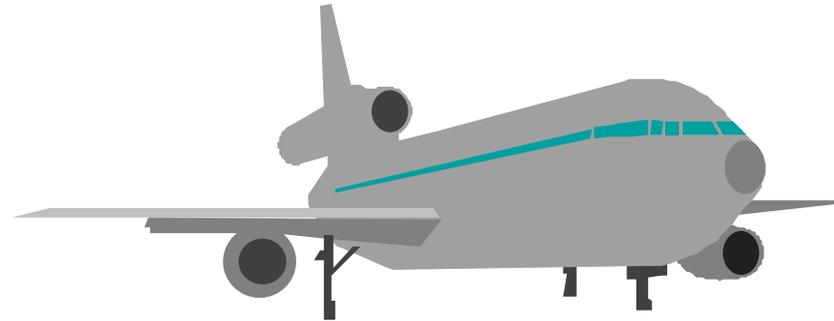
- life-critical applications

## Primary concern:

- tolerate arbitrary failures

## Secondary concerns:

- performance
- resources
- transparency



# Towards Lightweight Fault Tolerance

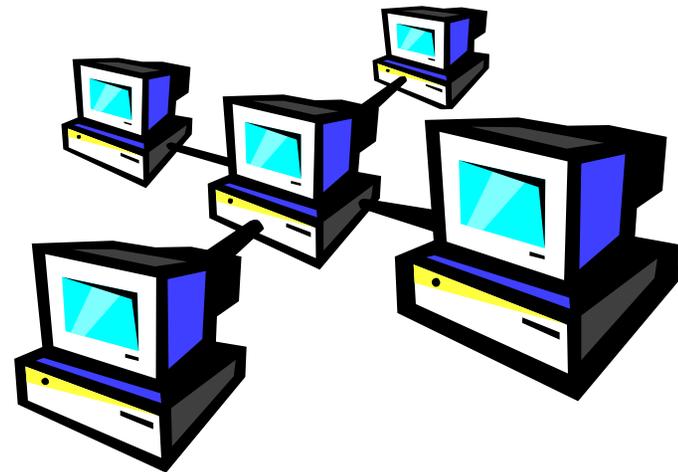
---

## Target:

- less critical applications

## Primary concerns:

- few dedicated resources
- negligible impact on performance
- fast recovery
- tolerance to common failures
- application-transparency



## Secondary concerns:

- Byzantine fault-tolerance
- continuous availability

# Rollback Recovery

---

- Log information on stable storage during failure free executions
- Use that information to recover after a failure



*Orphan*: process that depends on an unrecoverable state of a failed process

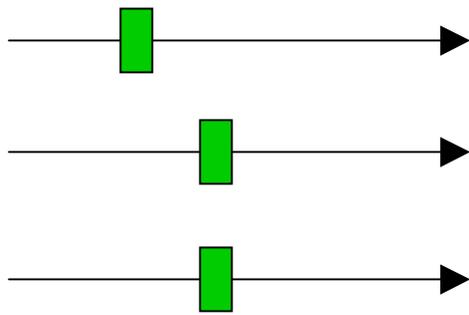
*Piecewise determinism*: all nondeterministic events can be identified and logged in the event's *determinant*

# Outline of the Talk

---

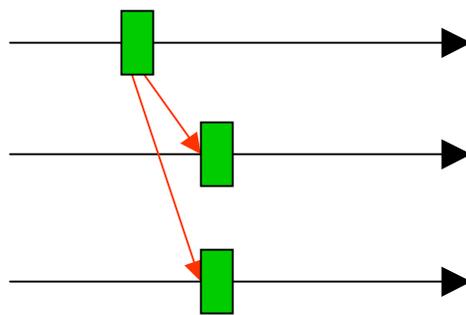
- Flavors of Rollback Recovery
- Egida: A Toolkit for Lightweight Fault-Tolerance
- Challenges

# Distributed Checkpointing at a Glance



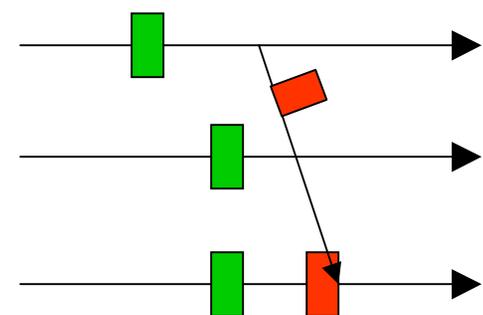
Independent

- 😊 Simplicity
- 😊 Autonomy
- 😊 Scalability
- ☹️ **Domino effect**



Coordinated

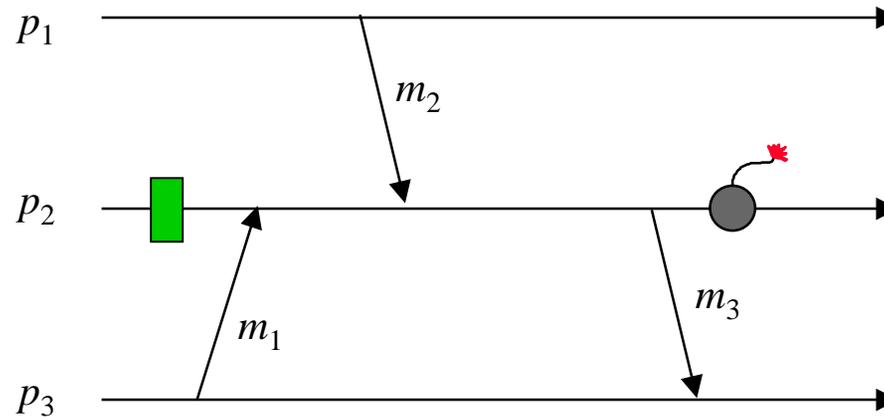
- 😊 Consistent states
- 😊 Good performance
- 😊 Garbage Collection
- ☹️ **Scalability**



Communication-induced

- 😊 Consistent states
- 😊 Autonomy
- 😊 Scalability

# Message Logging at a Glance



Pessimistic

- ☺ No orphans
- ☺ Easy recovery
- ☹ **Blocks**

Optimistic

- ☺ Non-blocking
- ☹ **Orphans**
- ☹ **Complex recovery**

Causal

- ☺ Non-blocking
- ☺ No orphans
- ☹ **Complex recovery**

# Rollback Recovery Protocols: A Success Story?

---

- Over 300 papers in the area
- Relatively few implementations
- Why?
  - Performance issues not understood
  - Integrating recovery protocols with applications non trivial
  - One size doesn't fit all

# Egida

---

A toolkit for supporting rollback recovery



- **Transparent**
  - seamless integration with applications
- **Extensible**
  - can easily handle new sources of non-determinism
  - can easily include new protocols
- **Flexible**
  - allows to select best protocol for application
- **Smart**
  - don't want to implement 300 protocols...
- **Powerful**
  - a “microscope” to understand rollback recovery

# The Unifying Theme

---

- All rollback recovery protocols enforce the *no-orphans* consistency condition
- The challenge is handling non-determinism
  - A process may execute non-deterministic events
  - A process may interact with other processes or with the environment and generate dependencies on these events
- Characterize a protocol according to how it handles non-determinism
  - Identify relevant events
  - Specify which actions to take when event occurs

# Relevant events

---

- Non-deterministic events
  - Ex: message delivery, file read, clock read, lock acquire
- Failure-detection events
  - time-out, message delivery
- Internal dependency-generating events
  - Ex: message send, file write, lock release
- External dependency generating events
  - output to printer or screen, file write
- Checkpointing events
  - Ex: timeout, explicit instruction, message delivery

# The Architecture

---

- Event handlers invoked on relevant events
- Library of modules
  - implement core functionalities
    - (checkpointing, creating determinants, logging, piggybacking, detecting orphans, restarting a faulty process, etc.)
  - provide basic services
    - (stable storage, failure detection, etc)
  - single interface; multiple implementations
- Use a specification language to select desired modules and corresponding implementations
- Synthesize protocol automatically from specification

# An Example of Protocol Specification

---

## Causal Logging

**/\* non-deterministic events statement \*/**

**receive:**

*determinant* : {source, ssn, dest,  
desn}

*Log* : determinant on volatile  
memory of processes

**/\* internal dependency-generating events  
statement \*/**

**send:**

*Piggyback* : determinants

*Log* : message on volatile memory  
of self

**/\* external dependency-generating events  
statement\*/**

**send:**

*Output Commit* : determinants

*Implementation* : independent

**/\* checkpoint statement \*/**

*Checkpoint* : independent, asynchronous  
on NFS disk

*Implementation* : incremental

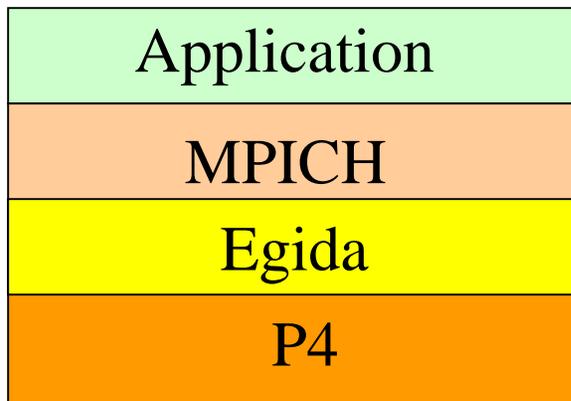
*Scheduling policy* : periodic

# Integration with MPI

---

## MPICH :

- 2-layered architecture
- upper layer exports MPI functions to application
- lower layer performs data transfer using platform specific libraries (e.g. P4)



## Modifications to MPICH:

- In upper layer, replace calls to P4 with corresponding calls to Egida API

## Modification to P4:

- Handle socket-level errors
- Allow recovering process to set up connections with correct processes

## Modifications to Applications:

**NONE**

# Status

---

- Works for Solaris\*
- *Almost* works for Linux
- Used at UCSD, Cornell

# What did we do with it?

---

## Protocols for fast recovery

- First comprehensive performance of recovery for RR protocols
- New protocols that provide:
  - Fast failure-free execution
  - Fast recovery
  - Fault containment

## Analysis of CIC protocols

- Consistent states
- Autonomy
- Scalability
- No useless checkpoints

Really?

# More Goodies

---

## Fault-Tolerant JVM

- capture non-determinism at the virtual machine interface

## Fault-Tolerant TCP

- Mask failure of server
- Don't change TCP
- Don't change client

## Secure Recovery Protocols

- what if the information used during recovery is tampered with?

# Challenges

---

- Transparency
- Scalability