Cluster Management Middleware (CMM) Functionality

- Resource management
  - Assignment of tasks to nodes
  - Scheduling
- Interactions with the outside world
  - Task submission
  - Status information
- Coordination of fault tolerance actions

Management Middleware Implementation

Typical characteristics:

- Agent process on each node
  - monitor local node status
  - perform local operations based on remote commands
  - provide local processes with an interface to the management middleware
- Centralized decision making - manager process
  - maintain global system status information
  - allocate resources / scheduling

Examples: Glunix, Condor, Hector, LSF, Sun Grid...

Fault-Tolerant Clusters

- Assumption: individual nodes (hw/sw) may fail
  communication errors
- Goal: fault tolerance for applications
- Simplest approach:
  restart application on working nodes
  - foundation for more advanced mechanisms
    e.g. checkpointing/rollback
- Requirement:
  fault-tolerant management middleware

Fault-Tolerant Management Middleware

Most existing systems:

- Periodic heartbeats: nodes → manager
- Fault tolerance for the central manager:
  - none → obvious single point of failure
  - primary + warm/cold backup →
    - fail-stop assumption
    - slow recovery

UCLA Fault-Tolerant Cluster Testbed (FTCT) Project

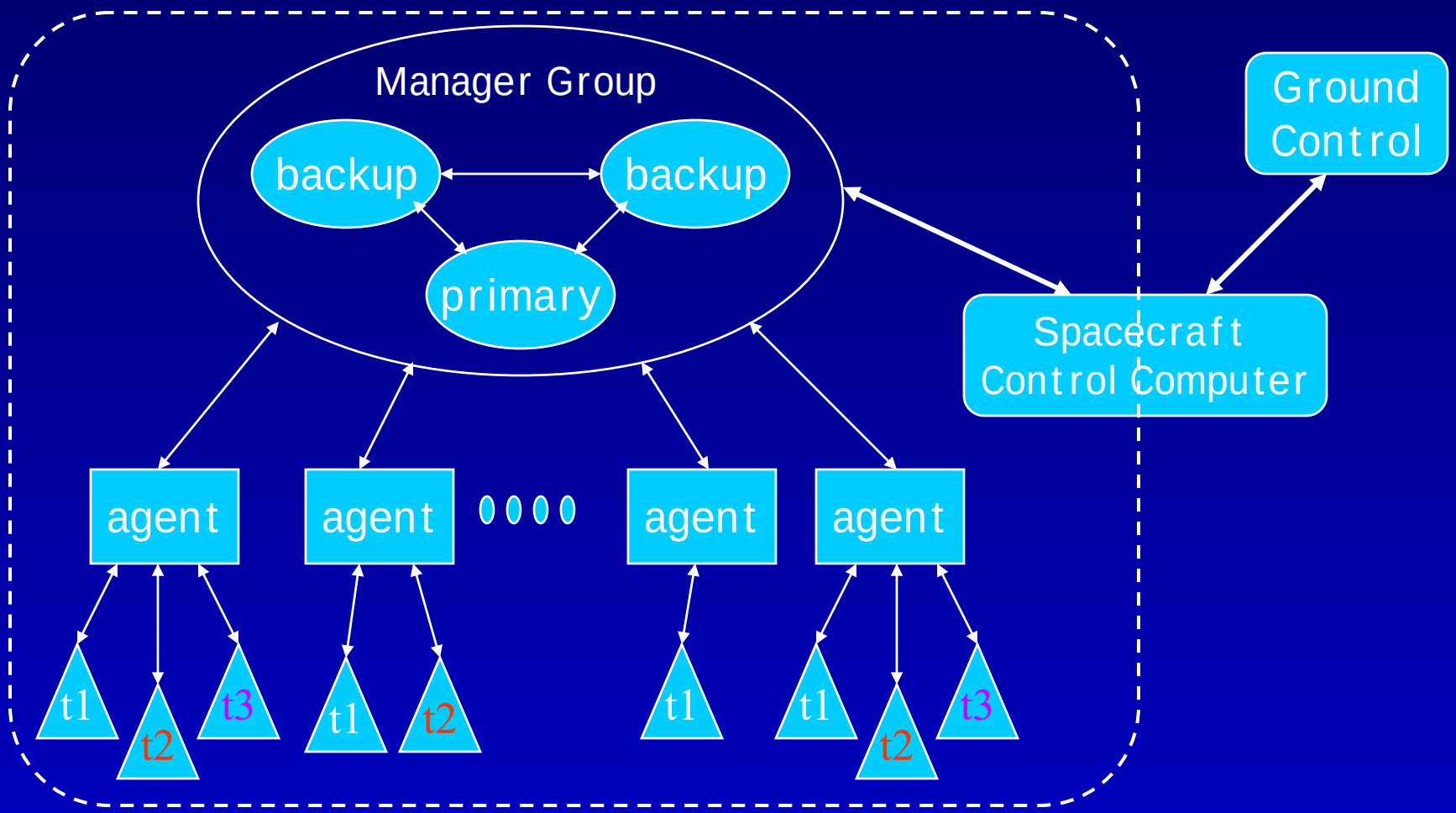
Design, implement and evaluate fault-tolerant cluster management middleware

Motivation (and support): JPL REE Project – high performance computing in space on COTS hw/sw

- support parallel tasks
- handle a wide class of faults
- provide infrastructure for application-level fault tolerance schemes

• • •

Logical System Structure



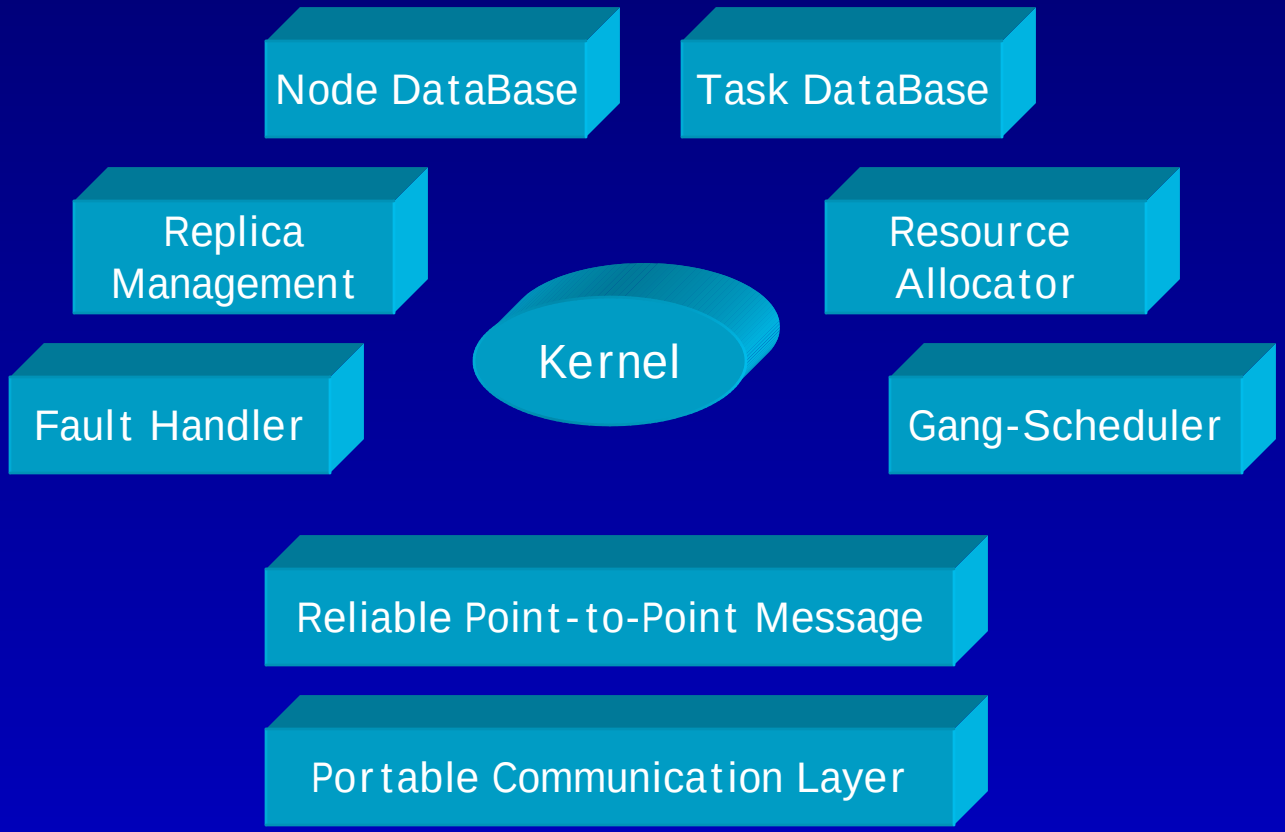
Middleware Building Blocks

- Managers
- Agents
- Reliable communication
- *Space craft Control Computer (SCC)*

Functionality of Managers

- Maintain states of resources and tasks
- Resource allocation
- Gang scheduling of tasks
- Interaction with spacecraft computer
- Handling of system-level fault events
- Maintaining the integrity of the manager group
- Provide infrastructure for application-level fault tolerance schemes

Manager Structure



Manager Implementation

Event-driven kernel

- Event types:
  - message delivery
  - time events
- Priority-based:
  time-consuming low priority handlers preempted by high priority events
  - implemented using lightweight user-level multithreading

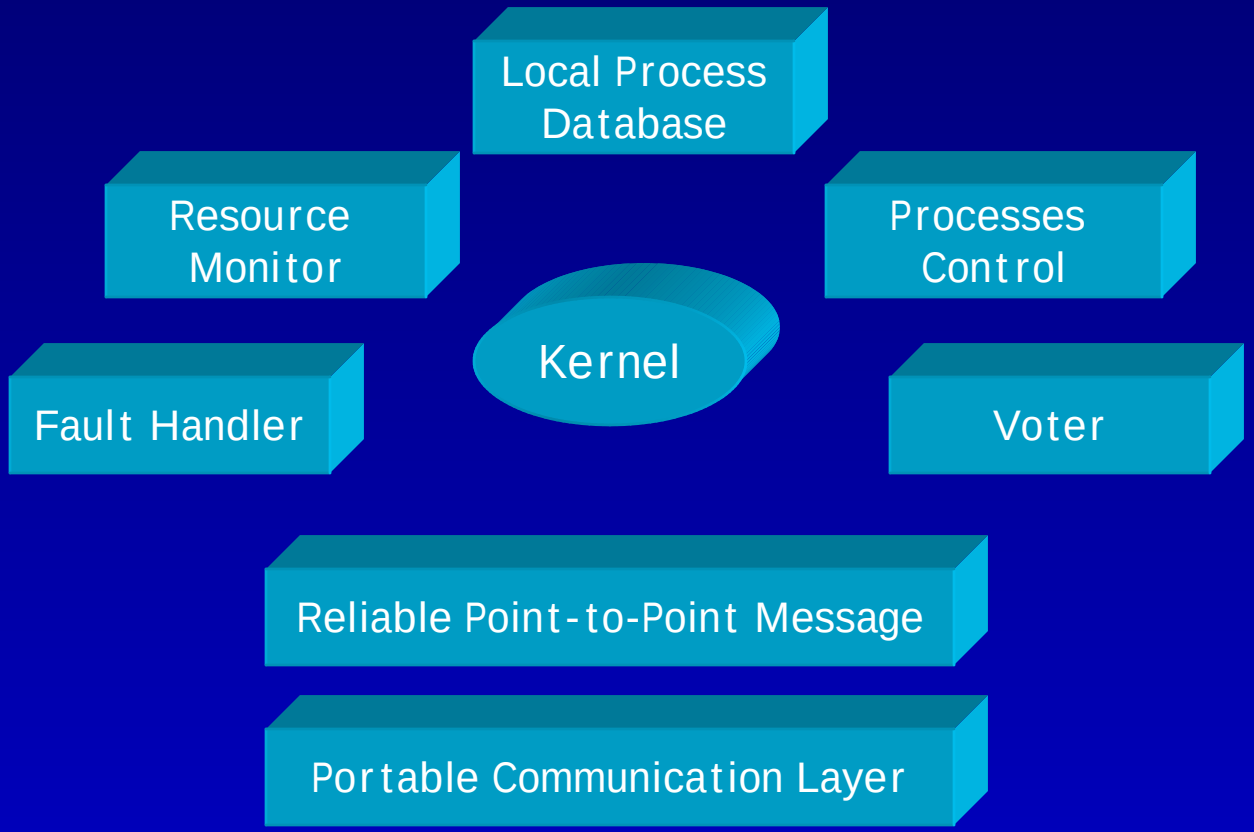
Event Loop

```
BEGIN LOOP
    IF there are any expired time events THEN
        push the event into event queue;
    IF there are any delivered messages THEN
        push the message into event queue;
    IF events queue is not empty THEN
        pop out an event with the highest priority;
        process the event;
    ELSE
        block until new event occurs;
END LOOP
```

Functionality of Agents

- Maintain state of node and processes running on node
- Report node info to manager group
- Perform local actions in response to manager commands
  - initiate,start, stop, kill processes
- Vote on manager commands and report errors
- Interface between application and management middleware

Agent Structure



Portable Communication Layer

- Message-based, connectionless, no delivery guarantees
- Supports NIC-host shared memory
  - exposes buffer management to upper layer
- Implementations
  - GM / Myrinet
  - TCP
  - UDP

Message Transmission Layer for Management Middleware

- Integrated with the management system (end-to-end)
- Message types:
  - unreliable
  - reliable, ordered with positive acknowledgement
  - reliable, ordered with negative acknowledgement
- Signed messages:
  - prevent spoofing ⇨
    greatly simplify many distributed tasks

Assumptions for First Prototype

- Network remains fully connected
  (messages may be lost intermittently)
- Single faults (until recovery action completes)
- Fault tolerance for applications ➔
  implemented by user-level libraries
  (may require key primitives from middleware)
- Local damage from “malicious” agents is ok
- *Space craft control computer (SCC)*

Error Detection

- Crash failures: periodic heartbeats
  - manager replicas exchange heartbeats
  - agents send heartbeats to manager group
- Corrupt manager replica: voting at agents
  - manager group is informed
  - manager group initiates self-diagnosis procedure

Space Craft Control Computer

- Controls entire space craft
- Controls communication between operators on Earth and the cluster (the manager group)
- Can power reset individual nodes or entire system

SCC failure → space craft failure

- "High reliability" – radiation-hard + aggressive FT
- Limited processing and communication capabilities
- Critical – software complexity must be minimized

SCC cannot handle the entire management task

Space Craft Control Computer

SCC failure → space craft failure

- "Fault-free" – radiation-hard + aggressive FT
- Limited processing and communication capabilities
- Critical – software complexity must be minimized

SCC cannot handle the entire management task

Conclusion: utilize, but sparingly

recovery mechanism of last resort

→ power reset based on manager group info

individual nodes or entire system

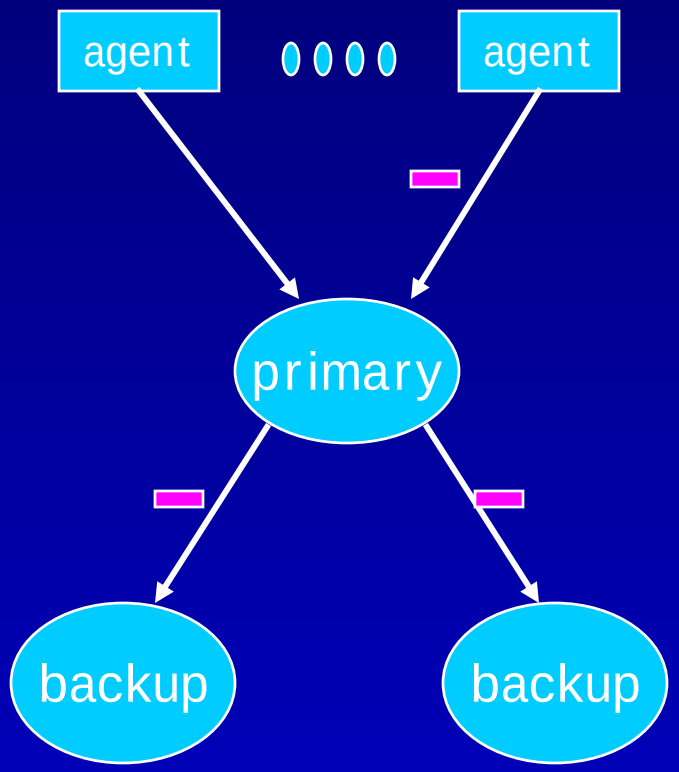
Atomic Multicast

Highly reliable managers → active replication →input consistency →reliable atomic multicast

- Amoeba's reliable multicast protocol
- Based on a fixed sequencer: the primary manager replica
- Simple and efficient
  - low overhead in the normal case
  - recovery from lost messages

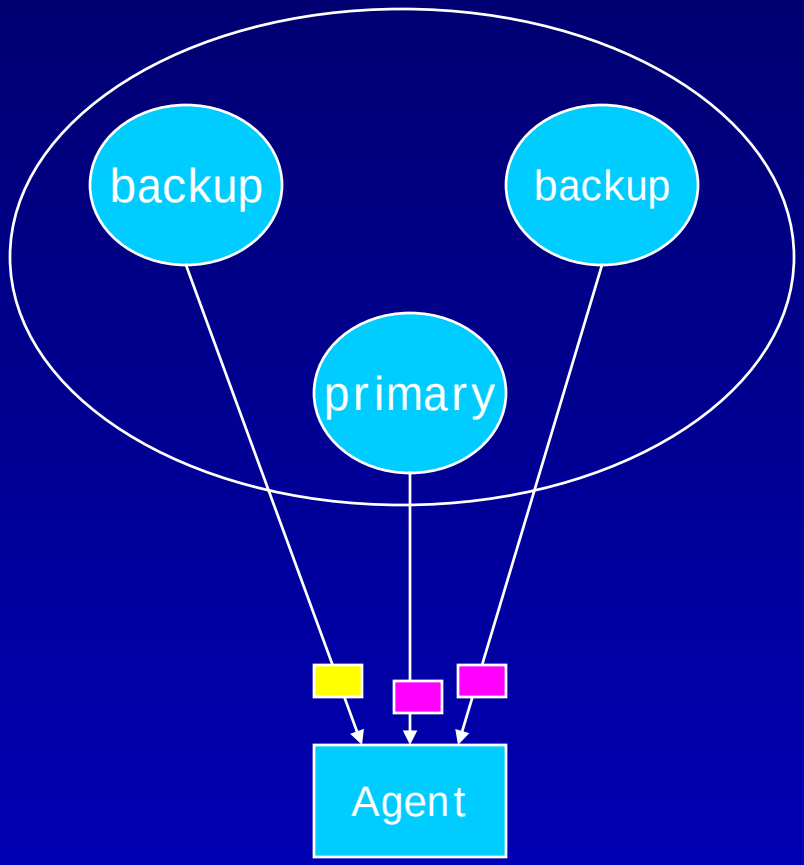
Reliable Multicast Protocol

- Agents send messages to the primary
- Primary(sequencer) orders messages and forwards them to backups



Voting at Agents

- Agents proceed based on two consistent copies of a message
- Agents detect missing (late) message replicas
- Agents report suspects to manager group → final diagnosis made by managers



Manager Diagnosis Procedure

- Invoked when agents report manager errors
- Manager replicas exchange/compare state checksums

- Faulty replica identified $\Rightarrow$
  replica killed, manager recovery initiated
- No faulty replica identified $\Rightarrow$
  report ignored, agent marked

SCC — Recovery Mechanism of Last Resort

- Manager replicas send heartbeats to SCC
- Manager replicas inform SCC when they enter self-diagnosis

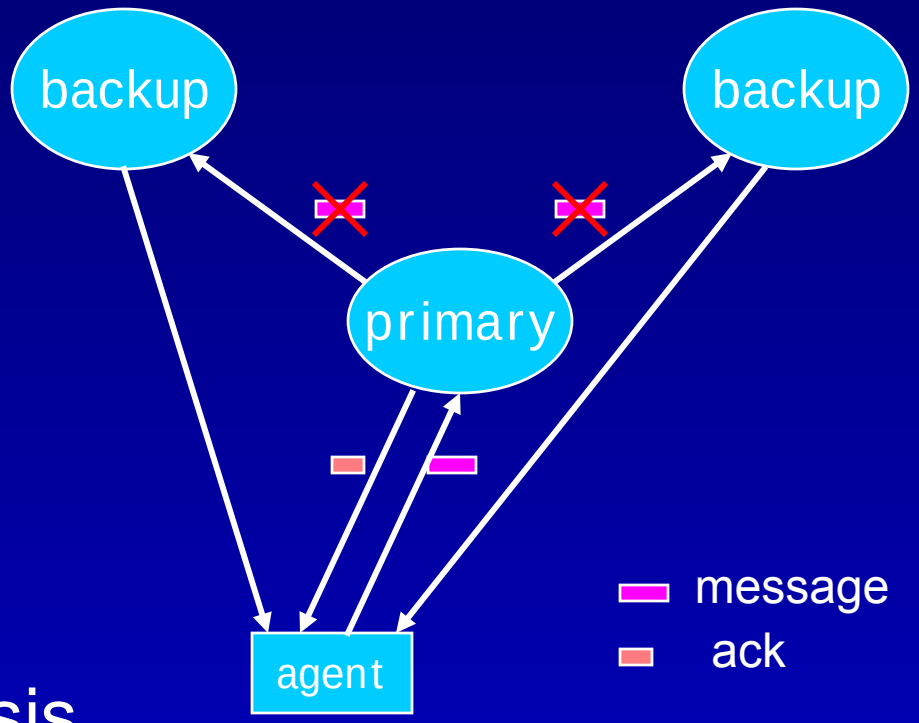
- SCC detects missing manager replica hearbeats
- SCC detects high frequency of multiple replicas entering self diagnosis

manager group is faulty ➔**system power reset**

Reliable Multicast: Error in Primary

Primary fails to forward messages

- Only one ack is received by agent
- Agent retransmits the message
- If the error persists, agent reports suspicion of primary failure to all the manager replicas

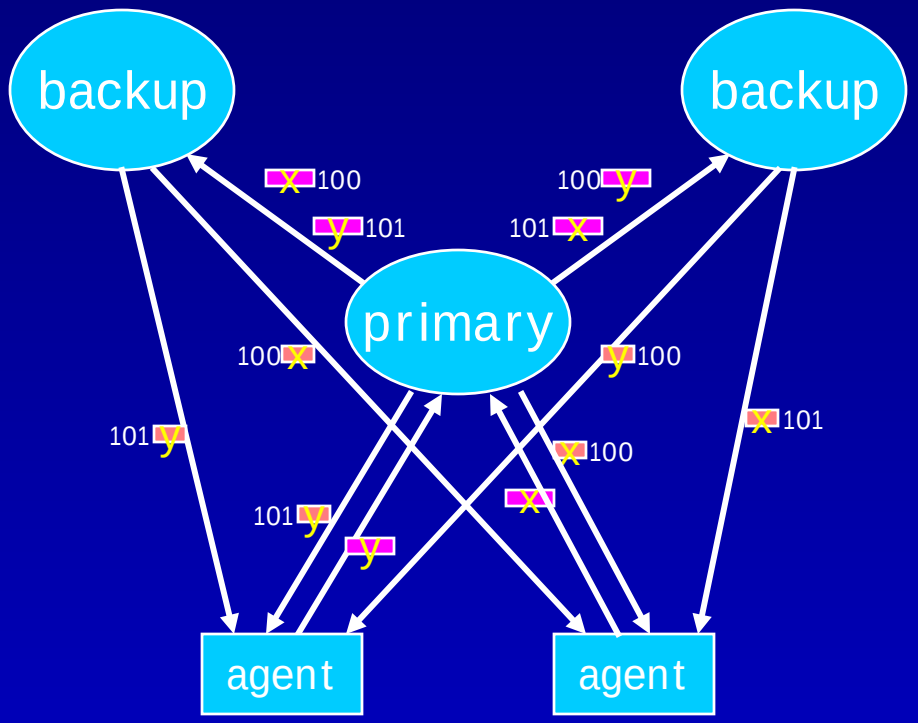


SCC involvement:
Repeated self diagnosis fails to reveal culprit
➔SCC resets system

Reliable Multicast: Primary Mischief

Primary assigns inconsistent RSNs

- Agents receive acks with different RSN
- Agent reports error
- Manager diagnosis initiated — the backup is terminated



Manager Group Time Events

- Consistently ordered
- Consistently ordered w.r.t. messages

- Events triggered on primary replica:
  primary forwards to backup replicas as messages
- Handling primary failure
  - Backup replicas schedule delayed “backup events’’
  - Firing of backup event →initiate manager diagnosis procedure

Application ⇔ Management Middleware Communication

Motivation: request service from middleware

- Built-in middleware service (e.g. terminate task)
- Distribute asynchronous message to all members of the task (e.g., initiate roll forward)

Implementation:

application process ⇔ agent communication

Application ⇔ Agent Implementation

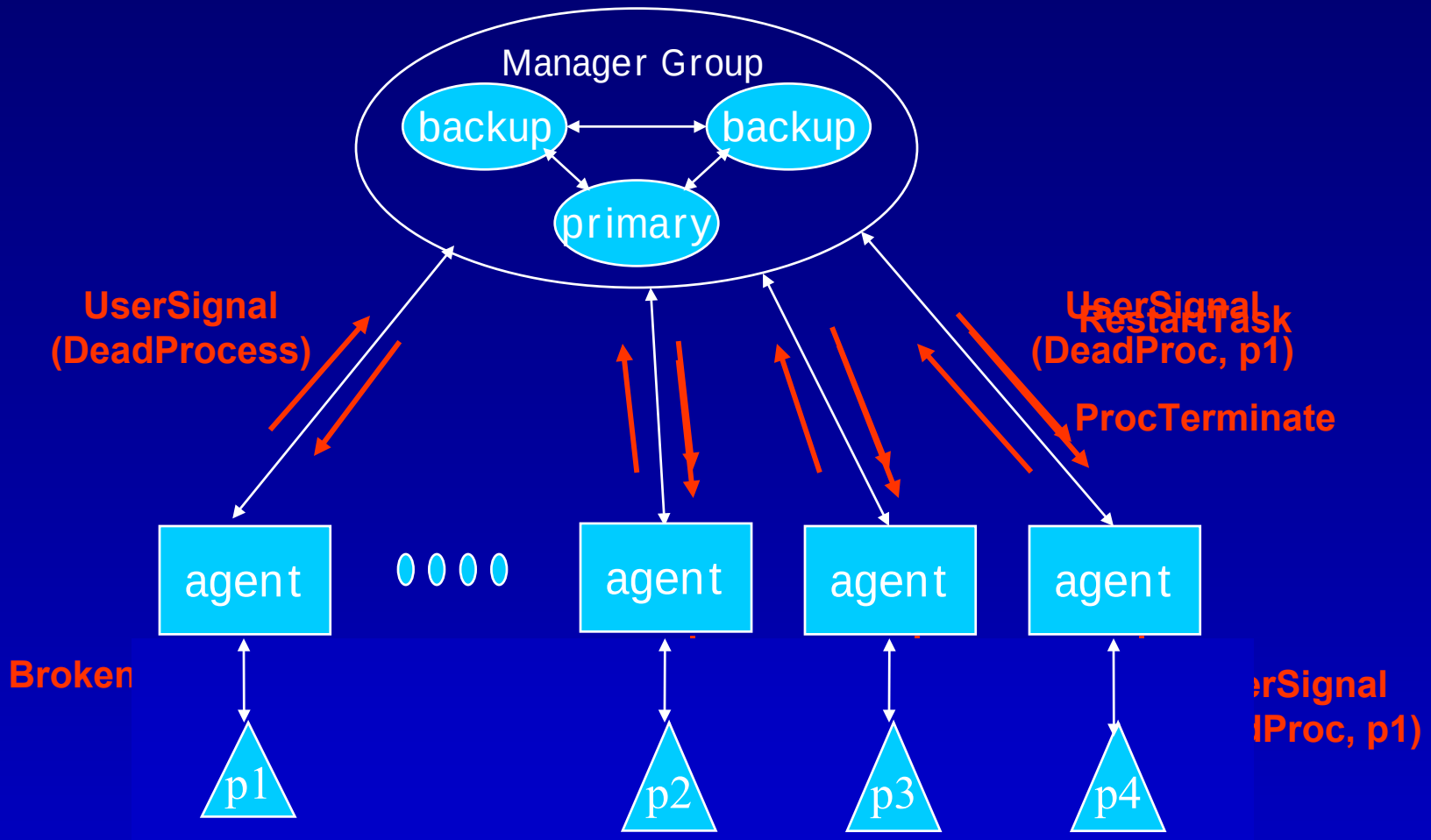
Difficulties:

- protection — agent from app, app1 from app2
  e.g., app cannot send to agent a UNIX signal
- communicating after agent recovery
  - Cannot use pipes

Solution:

- Special “app liaison” thread in agent
- Named UNIX FIFO between *app liaison* and every application process
  - liaison thread can wait on any FIFO “change”
- UNIX signals from agent to app processes

Restart Failed Application



FTCT: Current Status

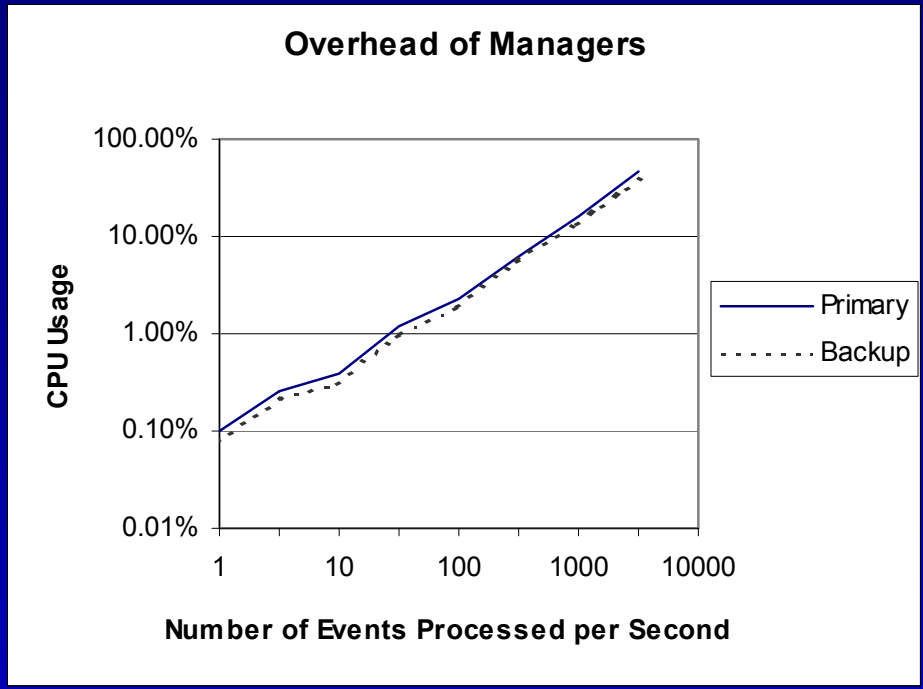
- Eight dual Pentium II-350 PCs
- Myrinet LAN + switched 100Mb Ethernet
- Solaris x86 and Redhat Linux
- Implemented in C++

- Operational since March 2001
  - runs multiple MPI tasks
  - a research tool — numerous enhancements changes, and fixes since then
- Approximately 30,800 lines

Preliminary Experimental Results: CPU Usage of Manager Replicas

- Overhead → handling heartbeats
- Measured as fraction of CPU not available for application

Conclusions:
5% overhead for 35 nodes, each reporting every 100 ms



Preliminary Experimental Results: Manager Recovery Time

Recovery time: detection → normal operation
excluding OS load time

- Primary Manager Failure: 63 msec
- Backup Manager Failure: 60 msec

- 40 msec initializing the communication system
- 3.7 msec to re-elect the primary manager

Future Work

- Evaluate alternative schemes
  (e.g. reduce reliance on SCC)
- Bootstrapping
- Scalability
- Fault injection experiments
- Integration of application-level error recovery mechanisms
- Reliable communication for the apps (mostly done)
- Characterization and optimization of the (soft) real-time performance of the system
- Evaluation of gang scheduling schemes

More Information

Daniel Goldberg, Ming Li, Wenchao Tao, and Yuval Tamir, ``The Design and Implementation of a Fault-Tolerant Cluster Manager,'' Computer Science Department Technical Report CSD-010040, University of California, Los Angeles, CA (October 2001).

http://www.cs.ucla.edu/~tamir/papers/ftct_tr.pdf