Open Resilient Cluster Manager: A Distributed Approach to a Resilient Router Manager

Ralph H. Castain, Ph.D.
Greenplum/EMC

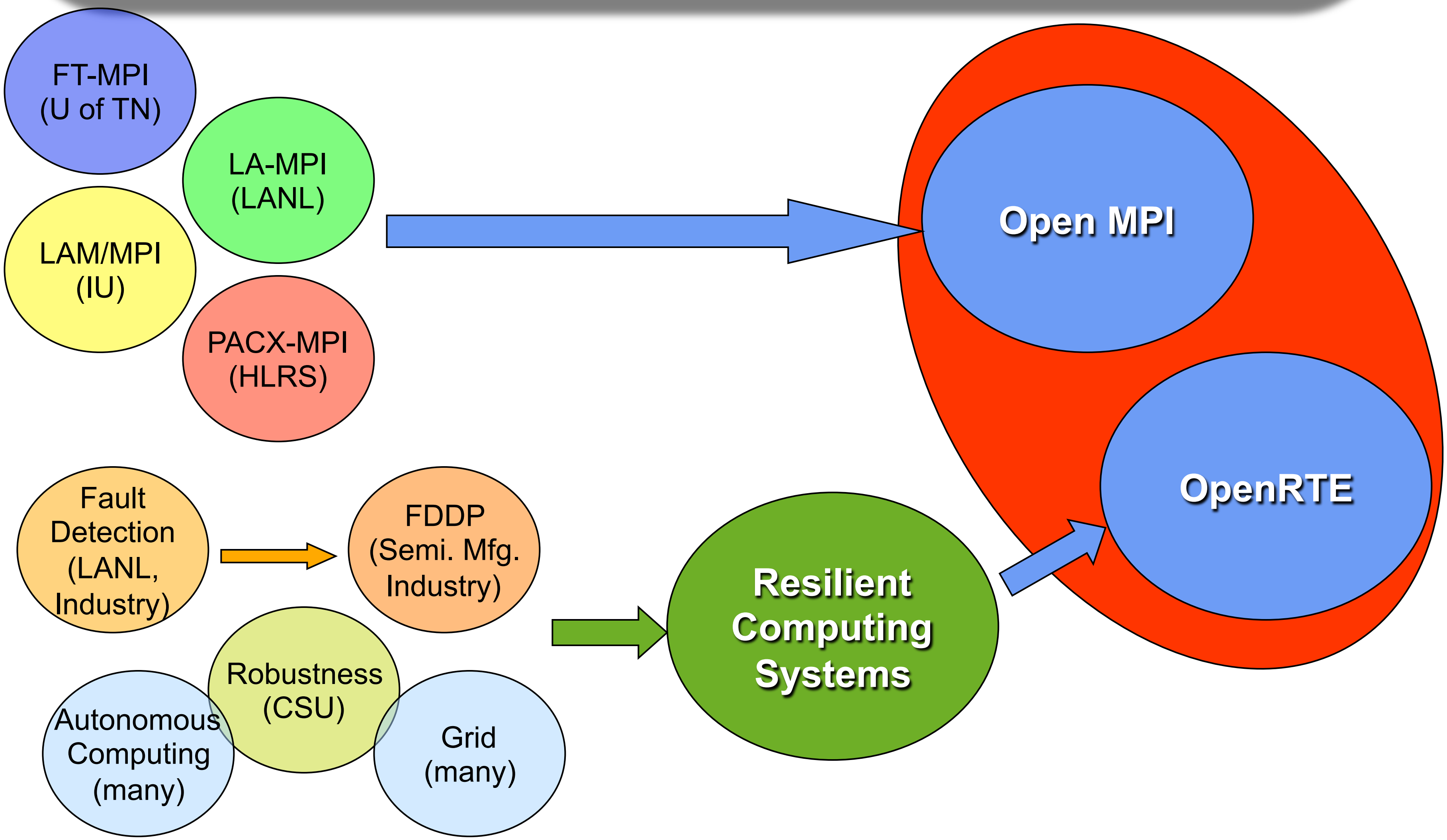
Objectives

- 100% uptime
  - Fully resilient to faults
  - On-the-fly maintenance
  - On-the-fly upgrades
- Test upgrades in the field without impact
  - Operate trial software systems in parallel, completely isolated
  - Transparent switch to production
- Maintain or enhance performance
  - Balance loads across all available processors

The Approach

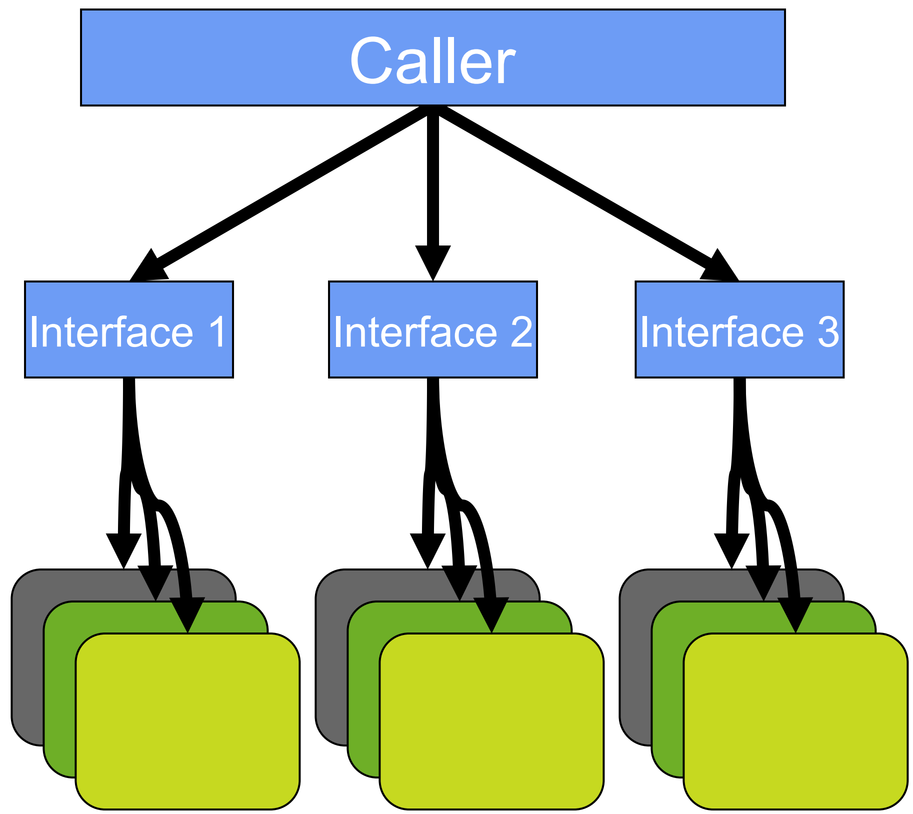
- Distributed redundancy
  - NO master
  - Multiple copies of everything
  - Running in tracking mode
    - Parallel, seeing identical input
    - Multiple ways of selecting "leader"
- Utilize component architecture
  - Multiple ways to do something => plug-in!
  - Encourage experimentation
  - Build on OpenRTE

ORTE: Convergence of Ideas



Reliance on Plug-ins

- Formalized interfaces
  - Specifies “black box” implementation
  - Different implementations available at run-time
  - Can compose different systems on the fly



OpenRTE and Plug-ins

- Plug-ins are shared libraries
  - Central set of plug-ins in installation tree
  - Users can also have plug-ins under $HOME
- Can add / remove plug-ins after install
  - No need to recompile / re-link apps
  - Download / install new plug-ins
  - Develop new plug-ins safely
- Update “on-the-fly”
  - Add, update plug-ins while running
  - Frameworks “pause” during update

Plug-in Benefits

- Stable, production quality environment for R&D
  - Can experiment inside the system without rebuilding everything else
  - Small learning curve (learn a few plug-ins, not the entire implementation)
  - Allow wide use, experience before exposing work
- Quickly roll out support for new platforms, features
  - Write only the plug-ins you want/need to change
  - Protect intellectual property (binary distro)

ORTE: Resiliency

- Fault
  - ***Events that hinder the correct operation of a process.***
    - May not actually be a “failure” of a component, but can cause system-level failure or performance degradation below specified level
  - Effect may be immediate or some time in the future.
  - Usually are rare. May not have many data examples.
- Fault prediction
  - Estimate probability of incipient fault within some time period in the future
- Fault Tolerance ……………………………………………*reactive, static*
  - Ability to recover from a fault
- Robustness……………………………………………….*metric*
  - How much can the system absorb without catastrophic consequences
- Resilience………………………………………………..*proactive, dynamic*
  - Dynamically configure system to minimize impact of potential faults

Key Frameworks

Error Manager (Errmgr)

- Receives all process state updates
  - Sensor, waitpid
  - Includes predictions
- Determines response strategy
  - Restart locally, globally, abort
- Executes recovery
  - Accounts for fault groups to avoid repeated failover

Sensor

- Monitors software and hardware state-of-health
  - Sentinel file size, mod & access times
  - Memory footprint
  - Temperature
  - Heartbeat
  - ECC errors
- Predicts incipient faults
  - Trend, fingerprint
  - AI-based algos coming

Universal PNP

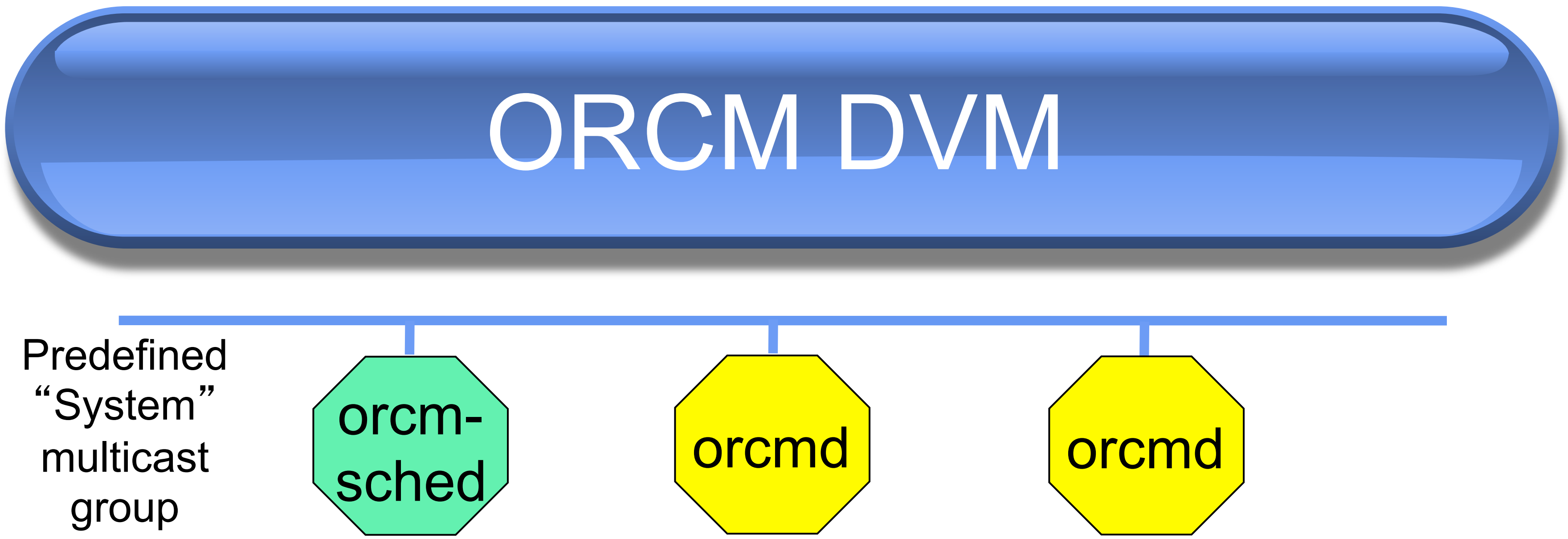
- Widely adopted standard
- ORCM uses only a part
  - PNP discovery via announcement on common multicast group
    - Includes application id, contact info
    - All applications respond
      - Wireup “storm” limits scalability
      - Various algorithms for storm reduction
  - APIs provided for sending/recving data
    - Broadcast-like output via multicast to anyone registered to recv output from that application
    - Unicast for direct IPC

Multicast: UDP

- Recvr-based methodology
  - All output sent to output group
  - Apps that want output from some other app join that app's output multicast group
- Each process joins several multicast groups
  - Given input, output for that app
  - System (announcements)
  - Error (recv process failure announcements)
- NACK-oriented protocol
  - Continuous stream of messages
  - Header contains sequence number
  - Missed message recovered via direct unicast from sender

Multicast: TCP

- Sender-based methodology
  - Output unicast directly to registered subscribers only
  - Each process maintains its own list of subscribers to each channel
  - Given contact info for daemon at startup
- Daemons
  - Simulate broadcast of all standard system channel messages (announcements, failures) by routing them across peers to apps
- Pros/cons
  - Resolves UDP messaging limits, lost message issues
  - Takes work to deal with things like race conditions on failure of subscribers



- One daemon (orcmd) per node
  - Started at node boot or launched by tool
  - Locally spawns and monitors processes, local system health sensors
  - Small footprint (≤1Mb)
- Each daemon tracks existence of others, has complete picture of system
  - PNP wireup
  - Know where all processes are located

orcm-sched

- Receives configuration input
  - Config mgr limitation => only one scheduler
  - Two mapping options
    - Maps process placement across available nodes, sends map to daemons
    - Sends configuration to daemons, daemons self-determine local procs
  - Daemons start local processes as required
- Provides point-of-contact for operational data requests
  - Returns snapshot of overall system state
- Responds to process failure (mapping mode 1)
  - Remaps process to new location

orcm-sched Failure

- Automatically restarted
  - Down time: ~2msec
  - No operational data available during down time
- Recover existing state from daemons upon startup
  - Each daemon provides current state
  - Scheduler assembles global picture
  - Total time to recover: ~100msec
- Receives configuration
  - Compares to existing state
  - Executes required corrections to match configuration

Multiple Parallel DVMs

- Allows
  - Concurrent development, testing in production environment
  - Sharing of development resources
- Unique identifier (ORTE jobid)
  - Maintains separation between orcmd's
  - Each application belongs to their respective DVM
  - No cross-DVM communication allowed

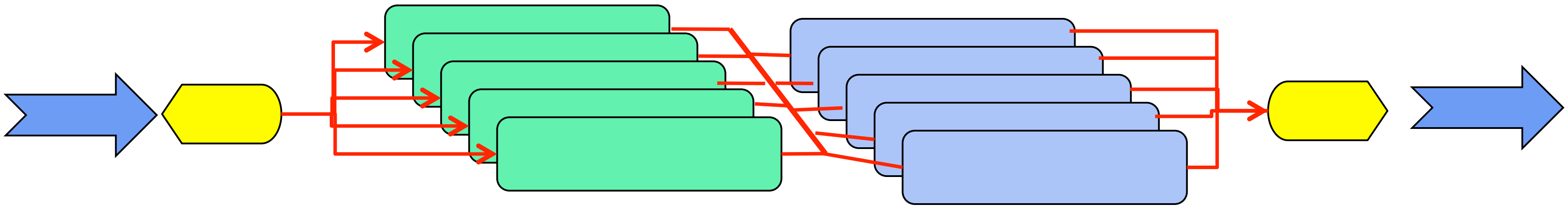
Apps: Multiple Replicas

- Multiple copies of each executable
  - Run on separate fault groups
  - Async, independent
- Shared pnp channels
  - Input: recvd by all
  - Output: broadcast to all, recvd by those who registered for input
    - Orcm-pnp delivers data to app layer only from leader => filtered so app only receives a single copy of each data stream

Leader Selection

- Two forms of leader selection
  - Internal to ORCM DVM
  - External facing
- Internal – framework, multiple plug-ins
  - App-specific plug-in
  - Configuration specified leader
  - Lowest rank alive
  - First contact
  - Leader-less (app receives raw data streams)

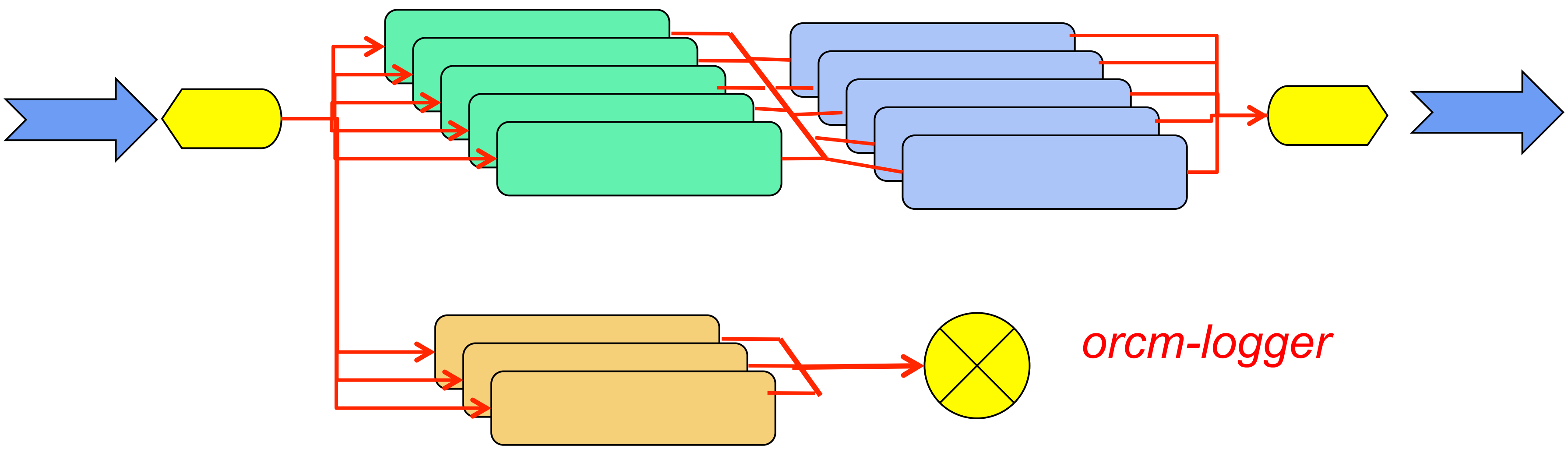
External Connections

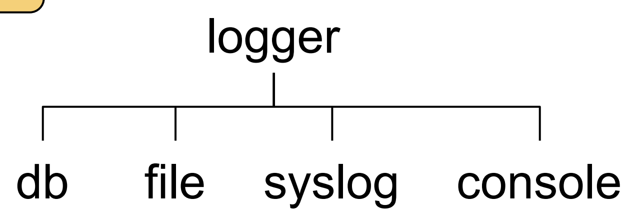


orcm-connector

- Input
  - Broadcast on respective PNP channel
- Output
  - Determines “leader” to supply output to rest of world
  - Utilize any leader method in framework

Testing in Production





Log messages, key framework behaviors (e.g., leadership selections)

Software Maintenance

- On-the-fly plug-in activation
  - Configuration manager can select new plug-ins to load, reload, activate
  - Change priorities of active plug-ins
- Full replacement
  - When more than a plug-in needs updating
  - Start replacement version, test until validated
  - Configuration manager switches “leader”
  - Stop old version

Detecting Failures

- Application failures - detected by local daemon
  - Monitors for self-induced problems
    - Memory and cpu usage
    - Orders termination if limits exceeded or are trending to exceed
  - Detects unexpected failures via waitpid
- Hardware failures
  - Local hardware sensors continuously report status
    - Read by local daemon
    - Projects potential failure modes to pre-order relocation of processes, shutdown node
  - Detected by DVM when daemon misses heartbeats

Resilient Mapper

- Fault groups
  - Nodes with common failure mode
  - Node can belong to multiple fault groups
  - Defined in system file
- Map instances across fault groups
  - Minimize probability of cascading failures
  - One instance/fault group
  - Pick lightest loaded node in group
  - Randomly map extras
- Next-generation algorithms
  - Failure mode probability => fault group selection

Application Fault Response

- Local daemon
  - Detects (or predicts) failure
  - Notifies all running processes of failure
    - New leaders selected as required
- orcm-sched
  - Utilizes resilient mapper to determine re-location
  - Sends launch message to all daemons
- Replacement app
  - Announces itself on application public address channel
  - Receives responses – wires up to recv input as specified by app

Application State Recovery

- Apps register callback at startup
  - Indicate ability to provide state recovery data for peers
- On startup
  - Apps provided with "incarnation" number
  - If restart detected, apps call orcm API to request state recovery data
  - Request sent to all peers
  - Active leader's registered state recovery function provides "blob"
  - Data returned to recovering app
- Recovery time
  - Few msec to restart process
  - State recovery limited by time to transfer blob between procs

Example: BGP

Simulator layout

- Multiple BGP siblings
  - All siblings store state locally in a routing table
  - Track sequence number of last update processed into table
  - Not really executing BGP algo, just processing faux updates into table
- Multiple shims
  - Each receives messages from faux external BGPs
    - Shim leader controls return channel
  - Relay messages to siblings
  - Retain local ring buffer of last N messages

Example: BGP

- Sibling restarts
  - Checks incarnation>0
  - Requests state recovery data
- Active sibling leader
  - Receives callback with request
  - Binary copies state table into return message, along with sequence number of last update
  - Orcm returns resulting blob to caller
- Restarted sibling
  - Places state table blob into its table
  - Notifies shims “ready” as of given sequence number
  - Shims send catch-up updates, continue from that point with normal updates

Node Replacement/Addition

- Failure or shutdown
  - Failure announced to all processes to permit leadership switch, if required
  - Orcm-sched relocates procs up to some max #times (can be unlimited)
- Auto-boot of local daemon on power up
  - Daemon announces to DVM
  - Orcm-sched adds node to available resources
- Processes will map to new resource as start/ restart demands
  - Future: rebalance existing load upon node availability

Current Status

- Overall system is functional, tested on router and cluster
  - Expected scaling issues (UDP stack limits on total messaging rate) observed
    - Several potential paths to improve scalability have been identified
    - Sender-based TCP messaging successfully avoids scaling problems
- No further development underway

Concluding Remarks

http://www.open-mpi.org
http://www.open-mpi.org/projects/orcm