Distributed Approach to a Resilient Router Manager

Ralph H. Castain, Ph.D.
Cisco Systems, Inc.

System Software Requirements

1) Turn on once with remote access thereafter
2) Non-Stop == max 20 events/day lasting < 200ms each
3) Hitless SW Upgrades and Downgrades
4) Upgrade/downgrade SW components across delta versions
5) Field Patchable
6) Beta Test New Features *in situ*
7) Extensive Trace Facilities: on Routes, Tunnels, Subscribers,…
8) Configuration
9) Clear APIs; minimize application awareness
10) Extensive remote capabilities for fault management, software maintenance and software installations

Our Approach

- Distributed redundancy
  - Multiple copies of everything
  - Running in tracking mode
    - Parallel, seeing identical input
    - Auto-select of “leader” by receiver
    - Leader may not be unique!
- Utilize component architecture
  - Multiple ways to do something => framework!
  - Create an initial working base
  - Encourage experimentation

Key Elements

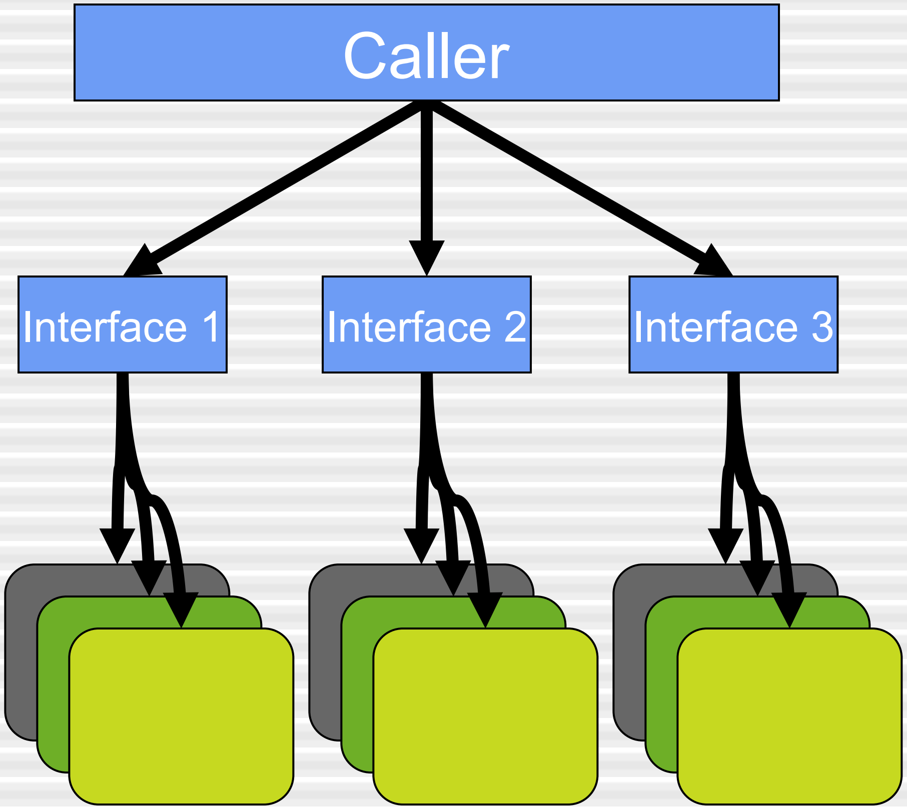
- Cluster Manager (CM)
  - Responsible for managing launch and placement of processes
- CM daemon (cmd)
  - Locally spawns and monitors processes, system health sensors
  - Small footprint
- Plug-and-play communications
  - Built on multicast
  - Auto-discovery, wireup, leader-selection

Use OpenRTE

- Exploit existing open source software
  - Reduce development time
  - Encourage outside participation
  - Cross-fertilize with HPC community
- Contribute back to OpenMPI
  - Proprietary modules as binary plug-ins
- Write new cluster manager (OpenCM)
  - Exploit new capabilities
  - Potential dual-use for HPC clusters

Reliance on Components

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



OpenRTE and Components

- Components are shared libraries
  - Central set of components in installation tree
  - Users can also have components under $HOME
- Can add / remove components after install
  - No need to recompile / re-link apps
  - Download / install new components
  - Develop new components safely

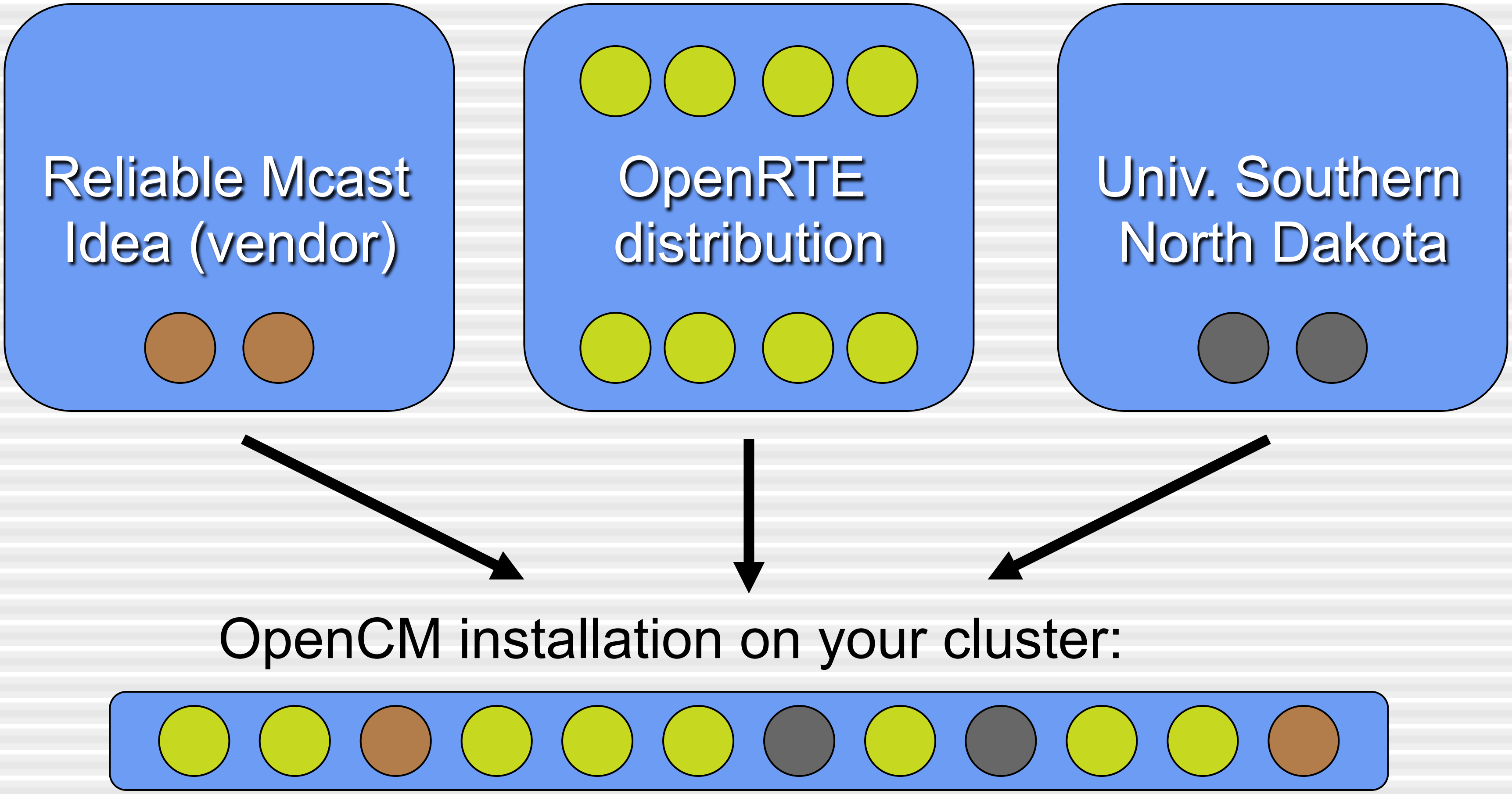
Component Benefits

- Stable, production quality environment for 3rd party researchers
  - Can experiment inside the system without rebuilding everything else
  - Small learning curve (learn a few components, not the entire implementation)
  - Allow wide use, experience before exposing work
- Vendors can quickly roll out support for new platforms
  - Write only the components you want/need to change
  - Protect intellectual property

3rd Party Components

- Independent development and distribution
  - No need to be part of main software distribution
  - No need to “fork” code base
- Compose to create unique combinations
  - Example: the pnp interface can utilize a new reliable multicast component without changing application
  - Select active combination by parameter at execution
- Can distribute as open or closed source binary plug-in module
  - Keep proprietary only what is proprietary

3rd Party Components

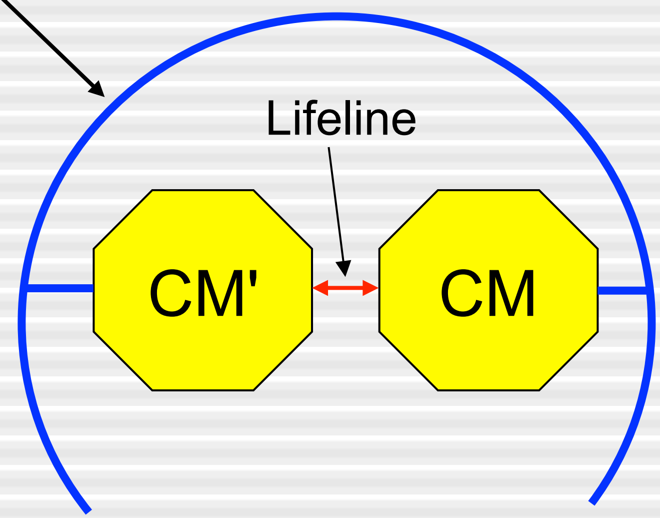


Reliable Mcast Idea (vendor)
OpenRTE distribution
Univ. Southern North Dakota
OpenCM installation on your cluster:

OpenCM Architecture

Predefined
“System” multicast
channel

Started at system boot



Exception: CM’s collectively agree on
leader, backup role during initial system boot

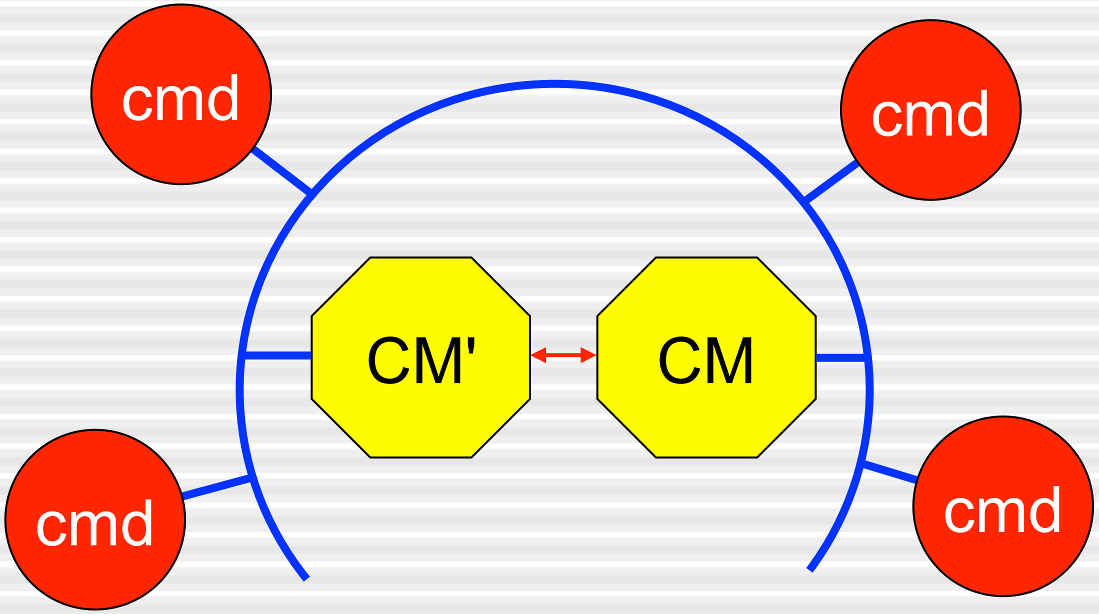
Why the "Exception"

- Leader selection
    - Normally left to individual apps, no centralized management
- Cluster Manager is different
    - Avoid conflicting launch commands and mappings, potential race conditions
    - Current algorithms are deterministic
        - Should arrive at identical conclusions
    - Future algorithms may be non-deterministic
        - Could cause conflicts
- Exception provides room for future extensions
    - Can remove later if not valuable

OpenCM Architecture

Announce existence during startup

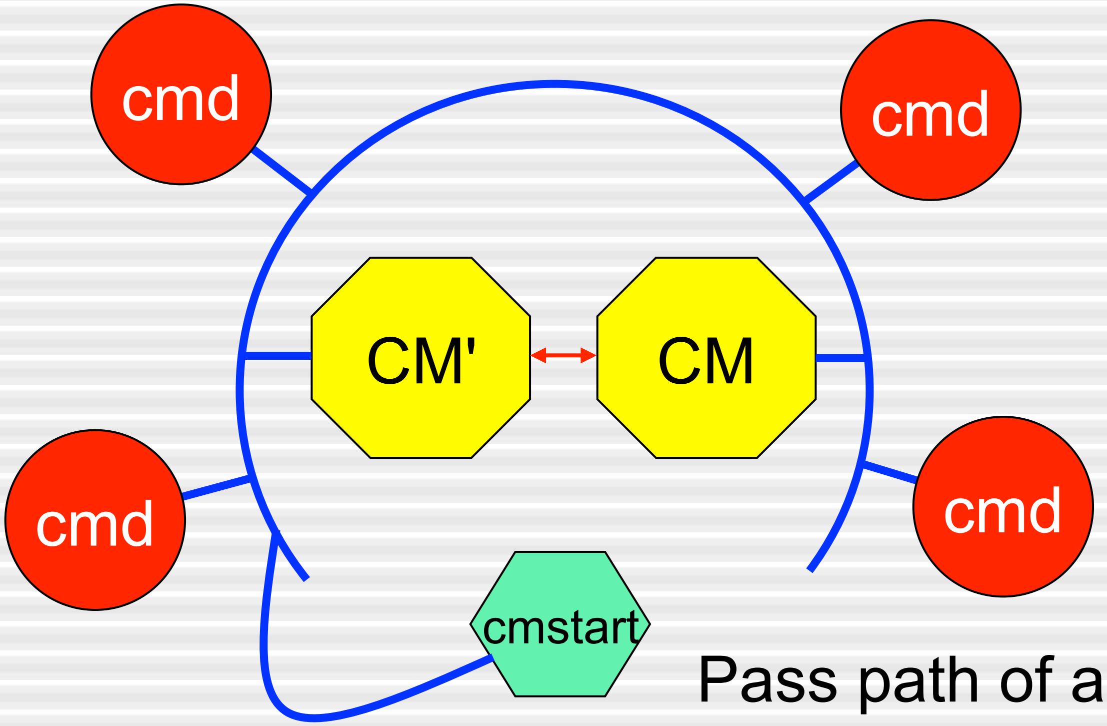
One per node
Started at node boot



CM adds node to known resources

OpenCM Architecture

Use file or tool to start apps

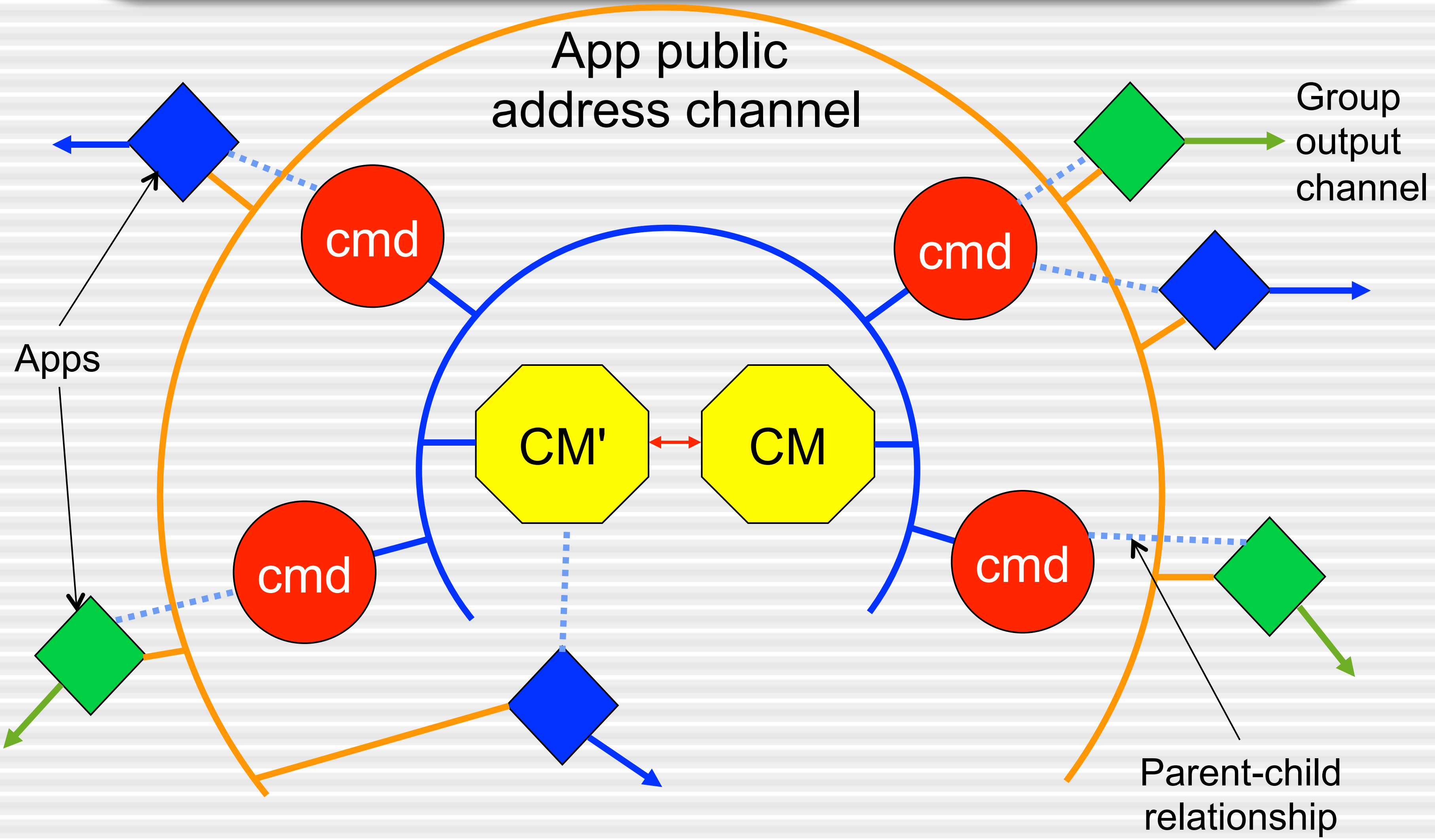


Pass path of app, number of instances to run

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

OpenCM Architecture



Application Startup

- Initialize library
  - ocm_init(OCM_APP);
- Announce our existence
  - ocm_pnp.announce(“app”, “version”, “release”);

Application “triplet”

- Register to receive inputs
  - ocm_pnp.register_input(“foo”, “1.3”, “alpha”, cbfunc);
  - ocm_pnp.register_input(“mother”, “1.0”, NULL, nag);
  - ocm_pnp.register_input(“wife”, NULL, NULL, yesdear);
- Wait for input

Application Operation

- Send output
  - ocm_pnp.output(buffer);
  - Multicast on application group output channel
  - Received by all “registered” apps
- Disconnect from input
  - ocm_pnp.deregister_input(“app”, “version”, “release”);
  - No notification sent to other process
- Normal termination
  - ocm_finalize();
  - Automatically closes all multicast channels
  - No notification is sent to other processes!

Under The Covers

- ORTE Multicast Framework
  - Allows multiple methods for reliable multicast
  - Current implementation is very basic
    - Message fragmentation not supported
    - Lacks tests for reliability
- Each application triplet is a multicast group
  - Own "channel" for output
  - Implications:
    - System limit on number of multicast memberships on a socket (~20) limit number of inputs
    - Limit on number of simultaneously running application triplets

Under The Covers

- PNP “Announce”
  - Outputs process triplet + triplet output channel on application public address channel
  - All processes maintain a log of announcements
  - Check received announcement against log of requested inputs
- Register input
  - Traverse log of announcements to find match
    - Found => open socket, join specified multicast group
    - Not found => log a request for input so that connection is made when announcement received

Startup Ordering Is Irrelevant

Receiving Messages

- Message handling by OMPI event library
  - Precedence: signal, file descriptor, timer
  - Multicast messages on socket => looks like fd
  - Fd’s handled in numerical order, cyclic, “fair”
  - Timer events: time, FIFO precedence
- All messages immediately “saved”
  - Read socket to get data
  - Generate 0-time timer event (include data)
  - Re-activate socket
- Messages delivered
  - In order received

Leadership Selection

- Framework => multiple methods supported
- First-In, First-Leader (FIFL)
  - Announcements tracked in order of receipt
  - First announcement from registered triplet = leader
  - Leader fails => take next in list
  - Triplet process restarts => goes on end of list
- Detecting leader failure
  - Heartbeat
  - Difference in message index received from other triplet instances
    - Settable trigger level
  - Messages saved for replay

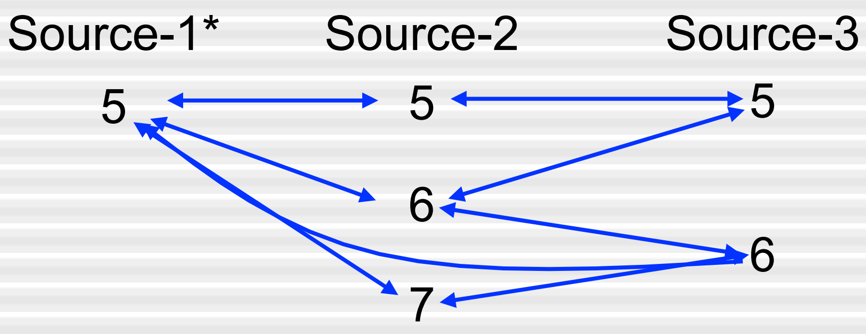
Leadership Failover

Trigger = 2

Messages logged to ring buffer for each source as they arrive

Locally assigned index for tracking purposes only

Delta message index computed at each message arrival



Source-1* Source-2 Source-3

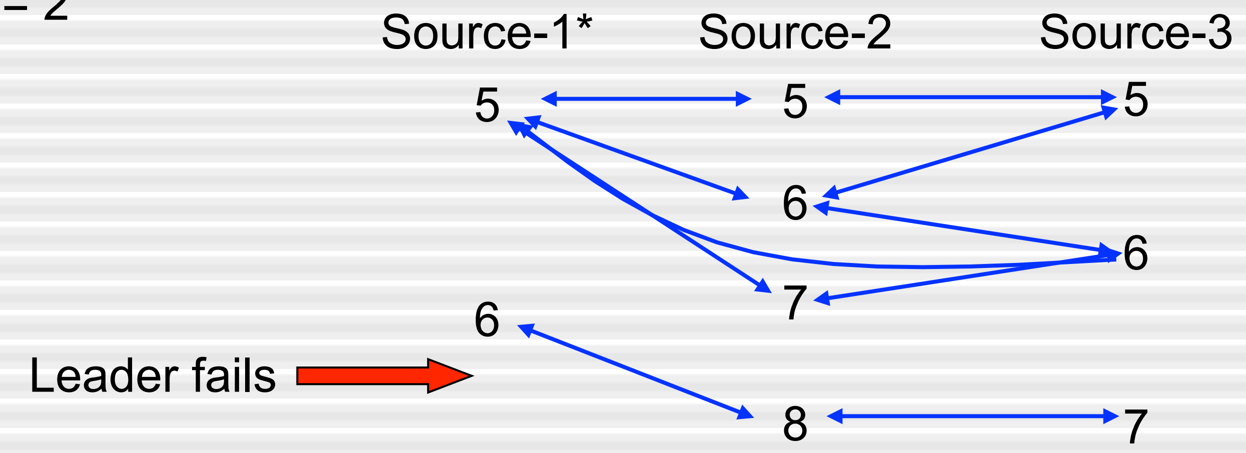
5 5 5

6 6

7

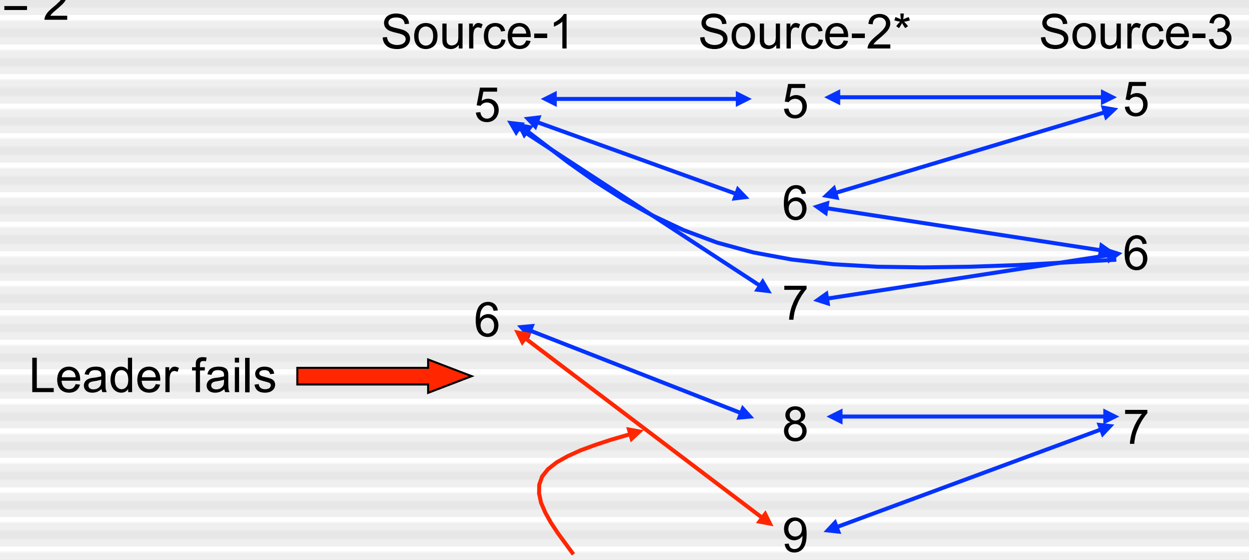
Leadership Failover

Trigger = 2



Leadership Failover

Trigger = 2



Source-1 Source-2* Source-3

5 5 5

6

6

7

6

Leader fails

8 7

9

Index delta > trigger

Change leader to Source-2
Replay messages 7-9 to callback function

Note: recvr gets all messages, in order, but time of receipt is delayed!

Making a Component

- Make a new subdirectory in the framework’s directory
  - leader/my_leader
- Copy some glue logic files
  - Allows build system to find and traverse files
  - Generate correct symbols
- Write your own versions of three API functions
  - valid_data
  - leader_failed
  - select_leader

Writing My_Leader

- bool valid_data(*source)
  - Ring buffer containing last N messages
  - Identity of the sender
- You can do anything you want to validate msg
  - Checksum data
  - Compare diffs between successive messages
  - Send email to your mother
- Must return…
  - True – data is okay
  - False – something is not okay

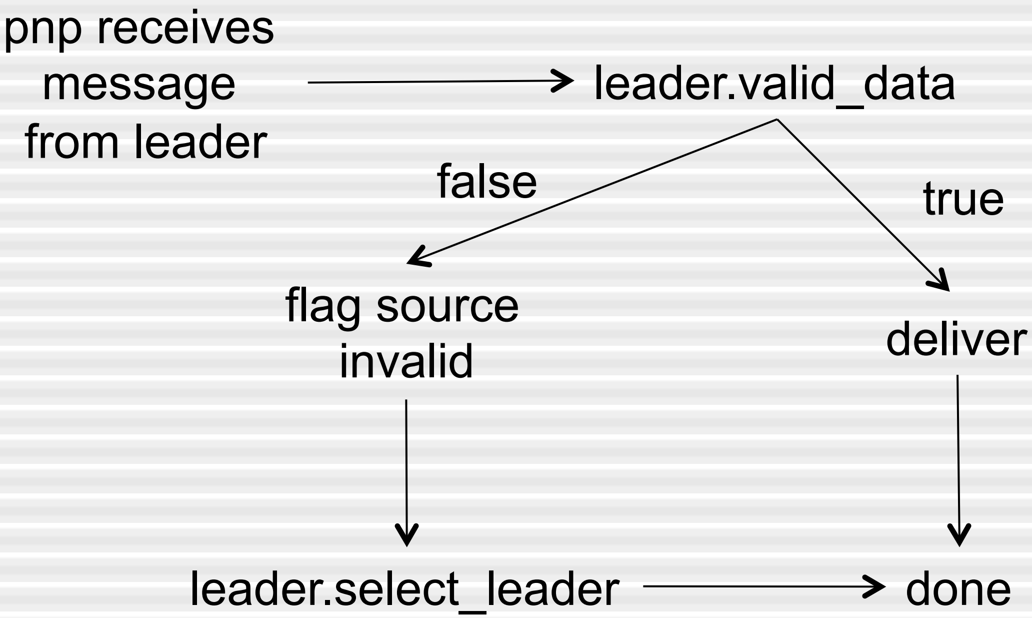
Writing My_Leader

- bool leader_failed(list of *sources)
  - List of pointers to source objects
  - Flag indicating the current leader
- You can do whatever you want
  - Time between messages
  - Relative message index
  - First recvd message
- Must return
  - True – leader has failed
    - Change leader, abort, whatever, next level up decides
  - False – leader still okay

Writing My_Leader

- int select_leader(list of *sources)
  - List of pointers to source objects
  - Flag indicating the current (failed) leader
- You can do anything you want
  - Randomly select someone on list
  - Collectively agree on selection
  - Use any of large number of provided utilities to facilitate decision
- Must do…
  - Set flag indicating the new leader
  - Return success or appropriate error code

Connecting The Dots



pnp receives message from leader

leader.valid_data

false

true

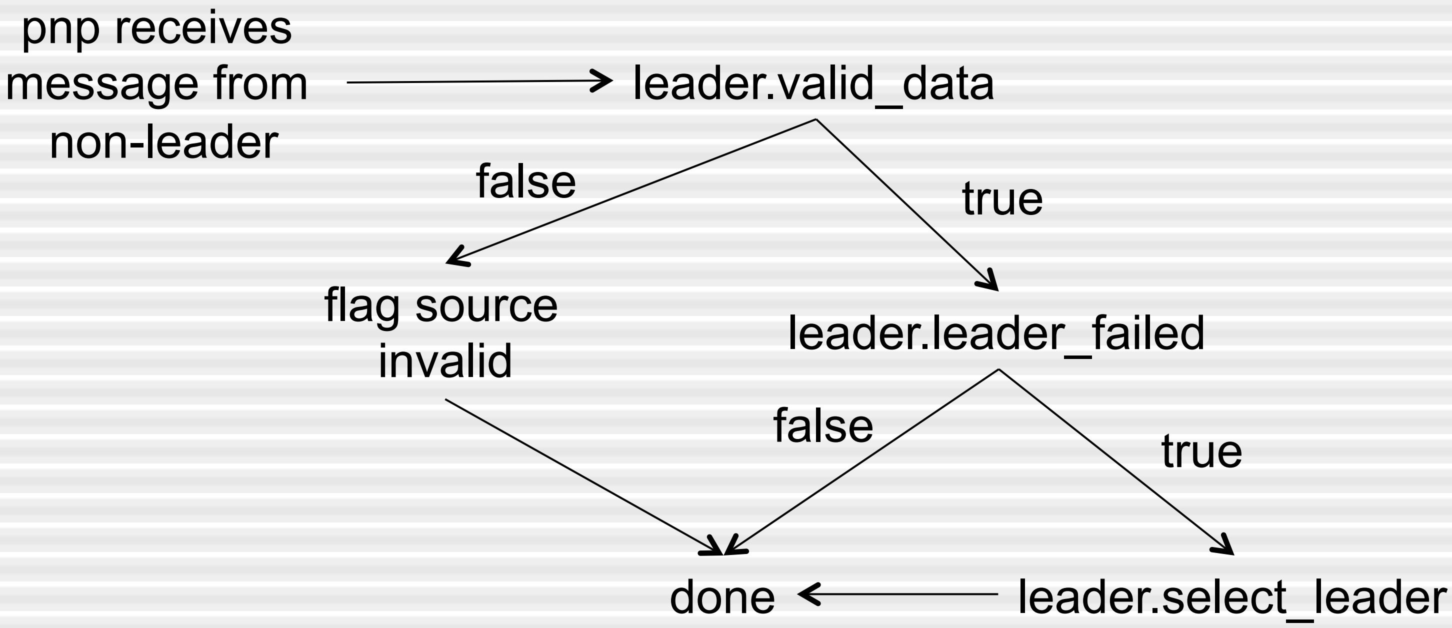
flag source invalid

deliver

leader.select_leader

done

Connecting The Dots



pnp receives message from non-leader

leader.valid_data

false

true

flag source invalid

leader.leader_failed

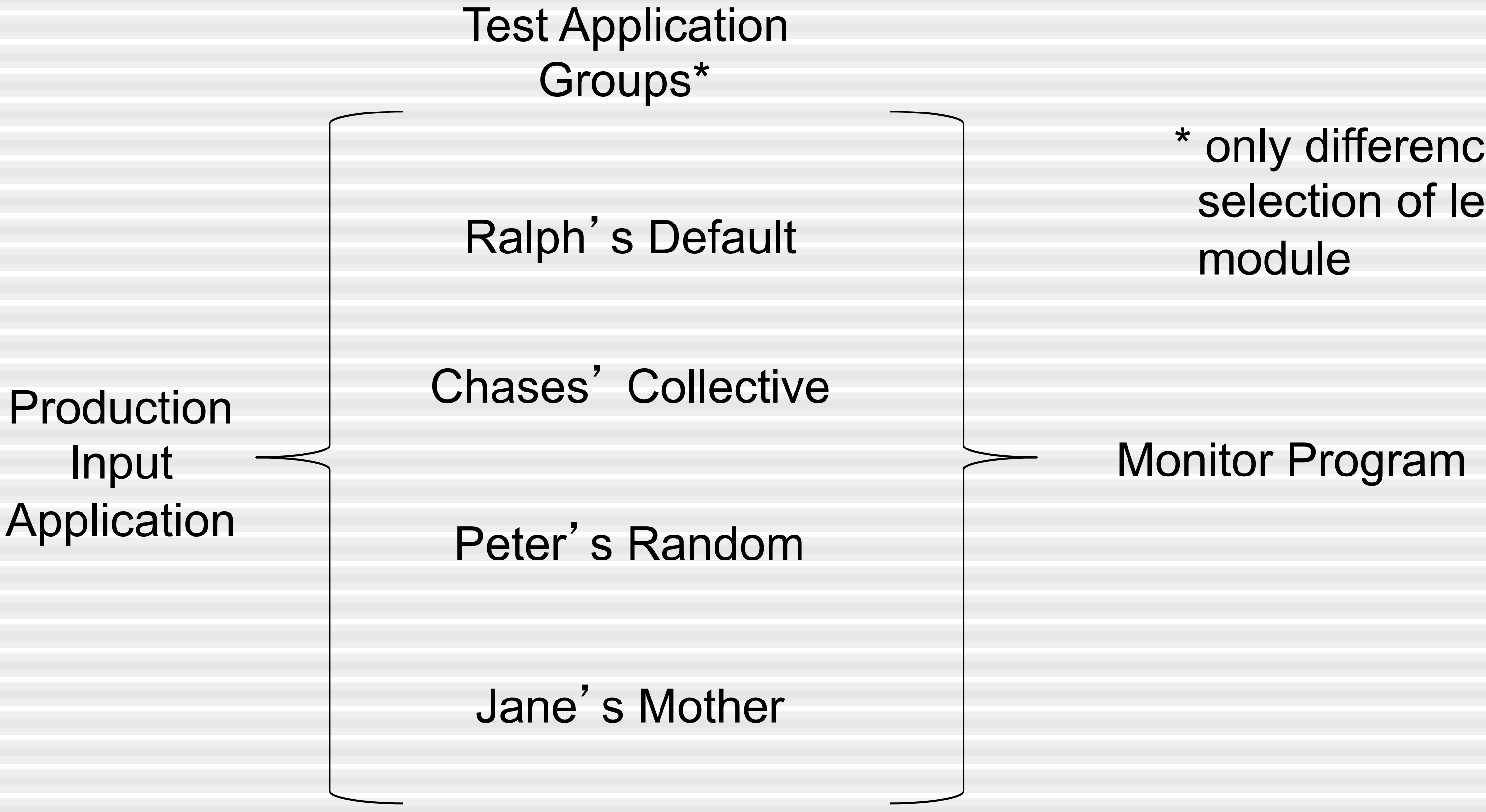
false

true

done

leader.select_leader

Using My_Leader



Test Application Groups*

Production Input Application

Ralph’s Default

Chases’ Collective

Peter’s Random

Jane’s Mother

Monitor Program

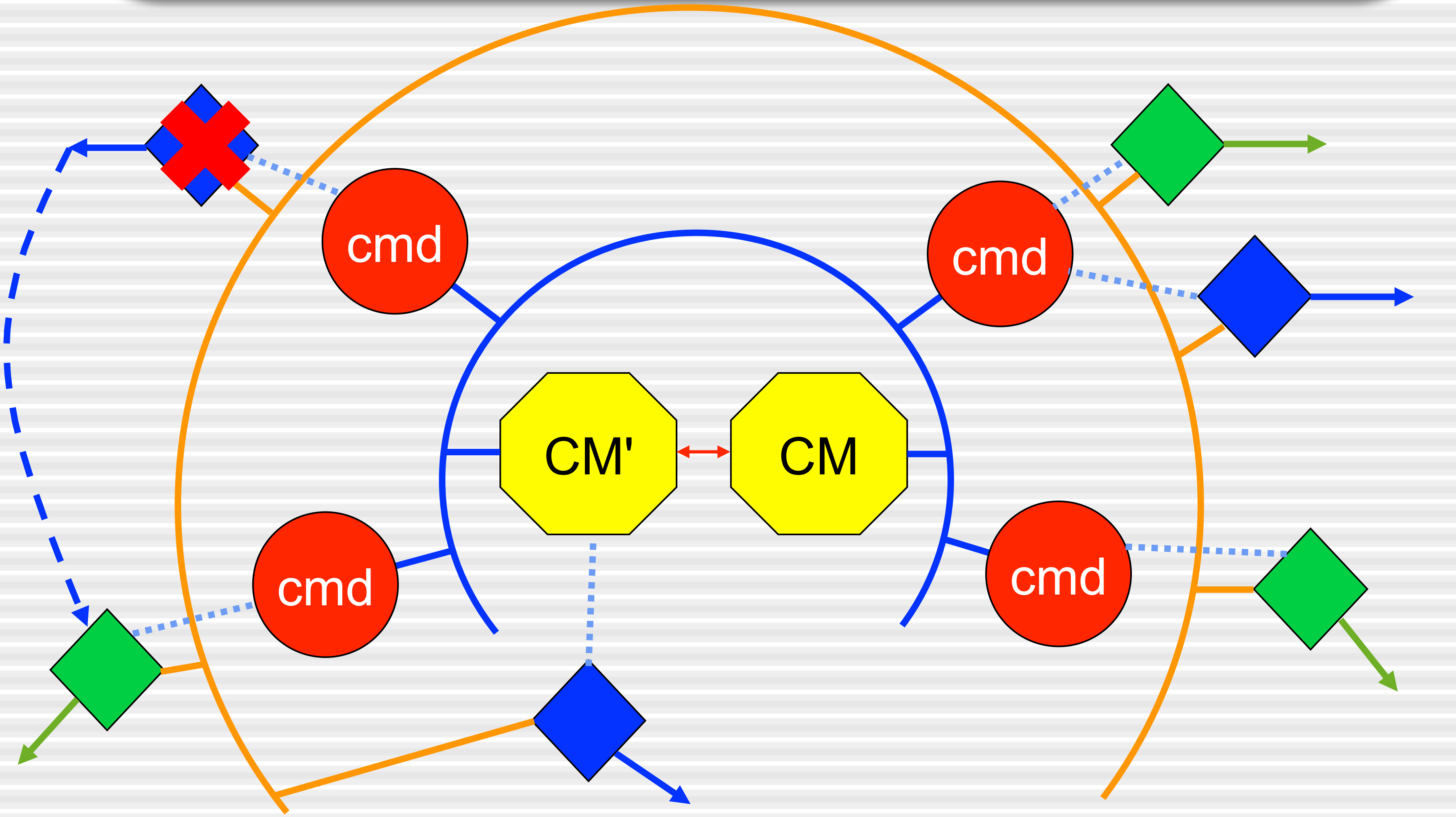
* only difference is selection of leader module

Which one works best, under what conditions?

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
- 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 CM when daemon misses heartbeats

Application Failure



Application Failure

- Local daemon
  - Detects (or predicts) failure, notifies CM
- CM utilizes resilient mapper to determine location of replacement, launches it
- Replacement app
  - Announces itself on application public address channel
  - Receives responses - registers own inputs
  - Begins operation
- Connected applications
  - Independently select new “leader”

Node Failure/Replacement

- Daemon fails
  - CM relocates and restarts all processes on that node
  - Connected apps automatically failover any affected leaders
- Node appears
  - Auto-boot of local daemon on power up
  - Daemon reports in to CM
  - CM adds node to available resources
  - Processes will map as start/restart demands
    - Future: rebalance existing load upon node availability

System Software Requirements

✔ 1) **Turn on once with remote access thereafter** Boot-level startup

✔ 2) **Non-Stop == max 20 events/day lasting < 200ms each** ~5ms recovery

✔ 3) **Hitless SW Upgrades and Downgrades** Start new app triplet, kill old one

✔ 4) **Upgrade/downgrade SW components across delta versions**

Start/stop triplets, leader selection

5) **Field Patchable**

✔ 6) **Beta Test New Features *in situ*** New app triplet, register for production input

7) **Extensive Trace Facilities: on Routes, Tunnels, Subscribers,…**

8) **Configuration**

✔ 9) **Clear APIs; minimize application awareness**

10) **Extensive remote capabilities for fault management, software maintenance and software installations**

Still A Ways To Go

- Security
  - Who can order CM to launch/stop apps?
  - Who can “register” for input from which apps?
  - Network extent of communications?
- Communications
  - Message size, fragmentation support
  - Speed of underlying transport
- Enhanced algorithms
  - Mapping
  - Fault prediction
  - Leader selection

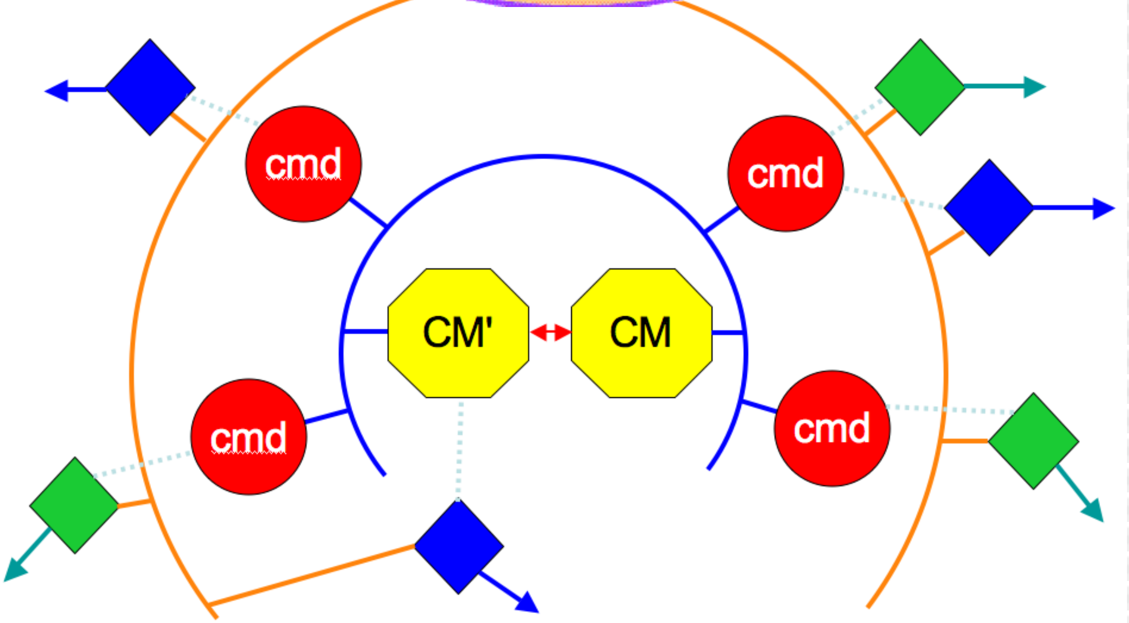
Connecting to the Outside World

- Orcm-connector
  - Input – fans out to respective pnp channel
  - Output – determines “leader” to supply output to rest of world
- On-the-fly module activation
  - Configuration manager can select new modules to load, activate
  - Change priorities of active modules
  - Restart/reload active module

Still A Ways To Go

- Transfer of state
  - How does a restarted application instance regain the state of its prior existence?
  - How do we re-sync state across instances so outputs track?
- Deterministic outputs
  - Same output from instances tracking same inputs
    - Assumes deterministic algorithms
  - Can we support non-deterministic algorithms?
    - Random channel selection to balance loads
    - Decisions based on instantaneous traffic sampling

Concluding Remarks


cmd
cmd
CM'
CM
cmd
cmd