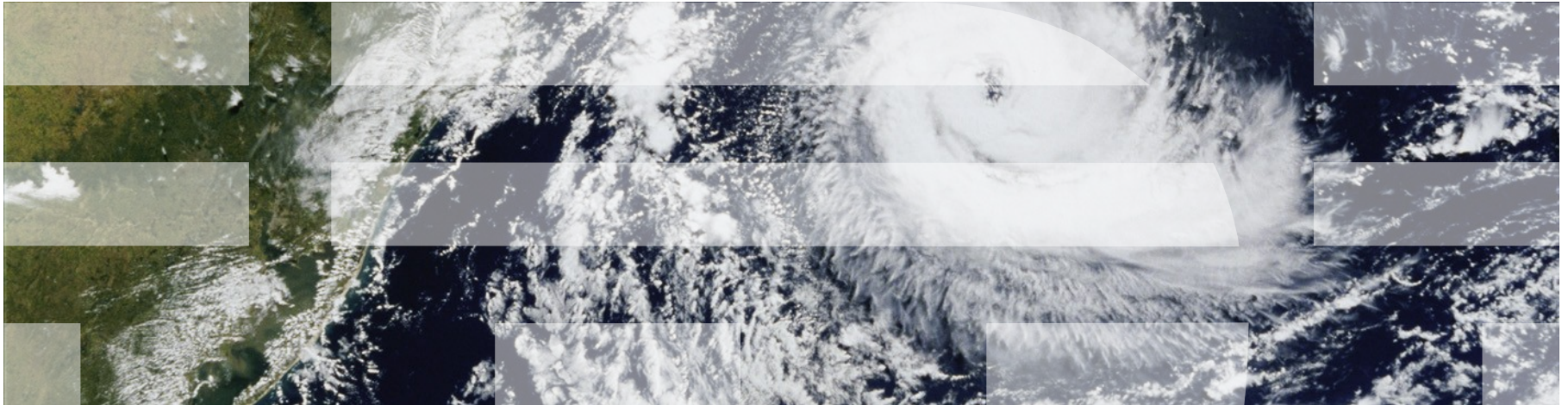


# Distributed Checkpoint / Restart on very large clusters



## Legal Statement

This work represents the view of the author and does not necessarily represent the view of IBM.

IBM, IBM (logo), e-business (logo), pSeries, e (logo) server, and xSeries are trademarks or registered trademarks of International

Business Machines Corporation in the United States and/or other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product, and service names may be trademarks or service marks of others.

## Introduction

- Today's High Performance Computing clusters often exceed the 10K nodes figure in an attempt to reduce computation times
- The probability to have a hardware failure is no longer negligible
- How to protect the long running jobs ?
- Save the data at the application level ?
  - Need a specific development for each distributed application ...
- Provide a framework to transparently save the application in a distributed way
  - Checkpoint / Restart works for a single application
  - A collaborative Checkpoint / Restart can be distributed

## Plan

- Part 1 – The containers
- Part 2 – The checkpoint / restart
- Part 3 – The distributed checkpoint / restart

## Containers - Definition



## Containers - Definition

– A container is an operating system object which has two properties:

- Isolation

The resources are allocated for a group of processes and are not accessible for the other processes

- Aggregation

The processes are grouped together under a single identifier and system resources are assigned to it

## Containers - Definition

Two kinds of container:

- Application containers : running an application in a container
  - bash, sshd, etc ...
  
- System containers : running a system in a container
  - /sbin/init booting the whole system

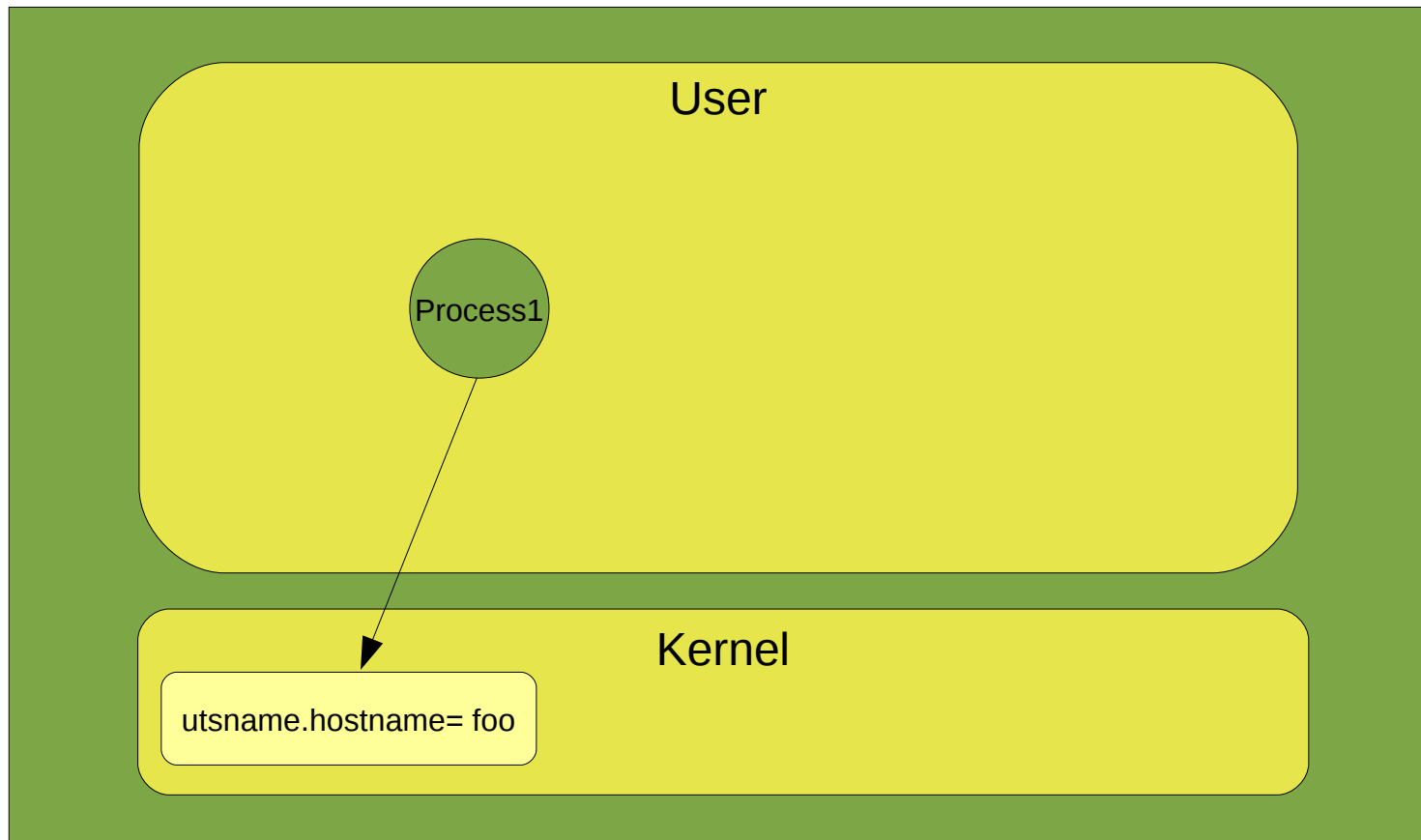
## Container – use cases

- Resource management
- Security
- Virtual OS
- Mobility
- Multihosting
- Cross distros compilation
- Server consolidation
- Process tracking

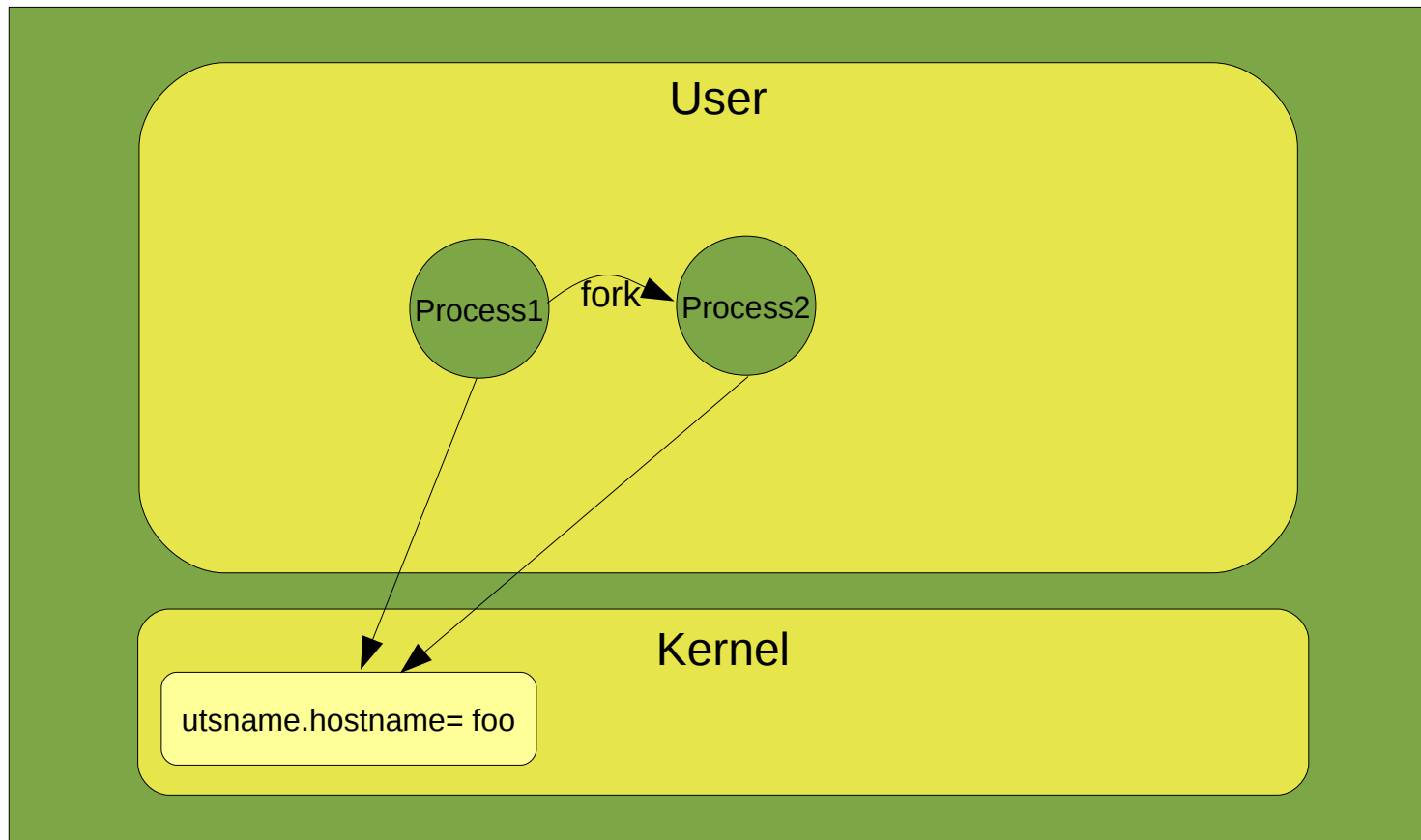
## Container - isolation

- The isolation is provided by a base brick called a “namespace”
- System resources are accessed via the namespaces
- A namespace per kernel subsystem (mount, network, ipc, ...)
- A namespace contains new system resources instances
- A namespace is created with one syscall (unshare / clone)
- Kernel algorithms are not impacted, just the way the resource is accessed

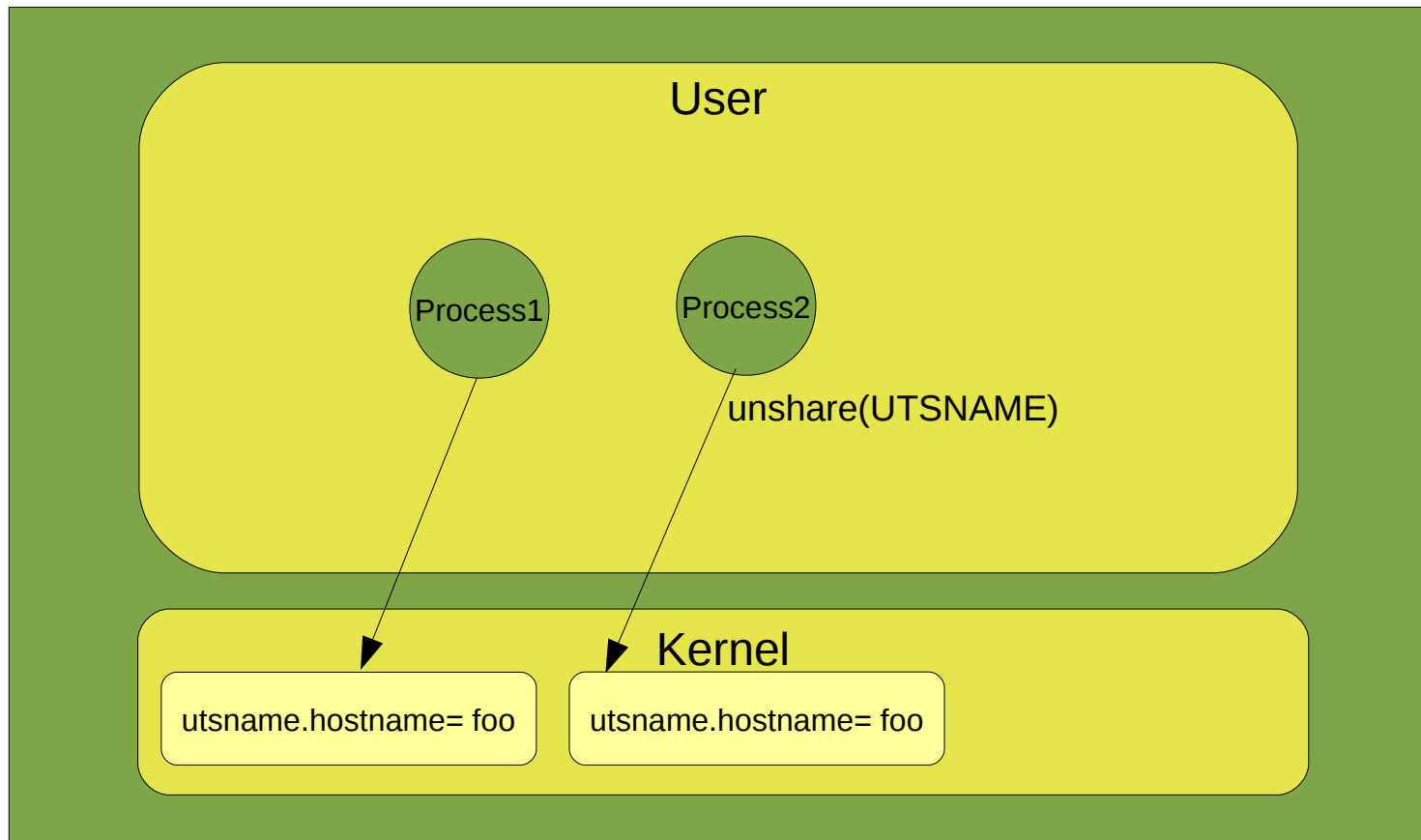
## Container – isolation example with the utsname



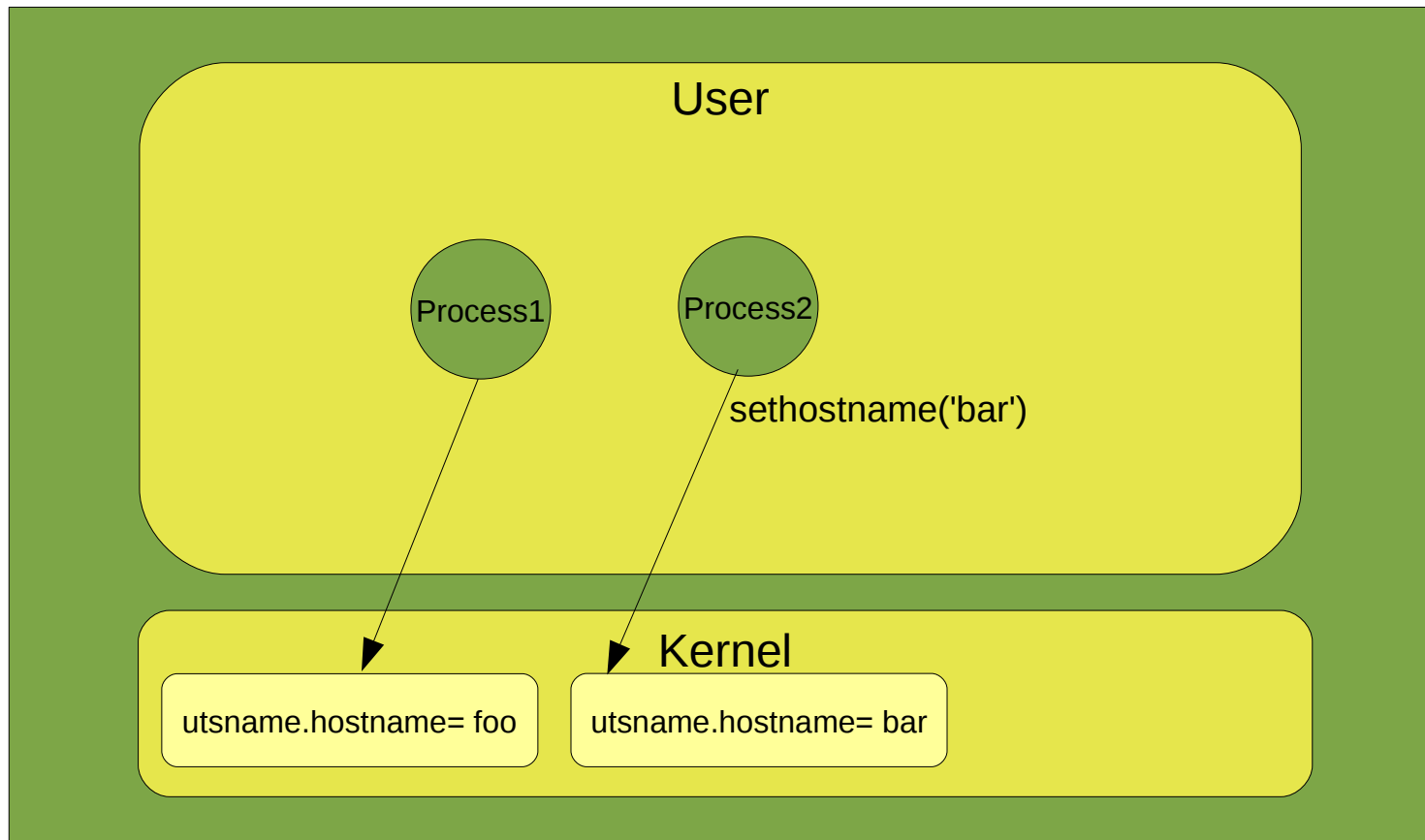
## Container – isolation example with the utsname



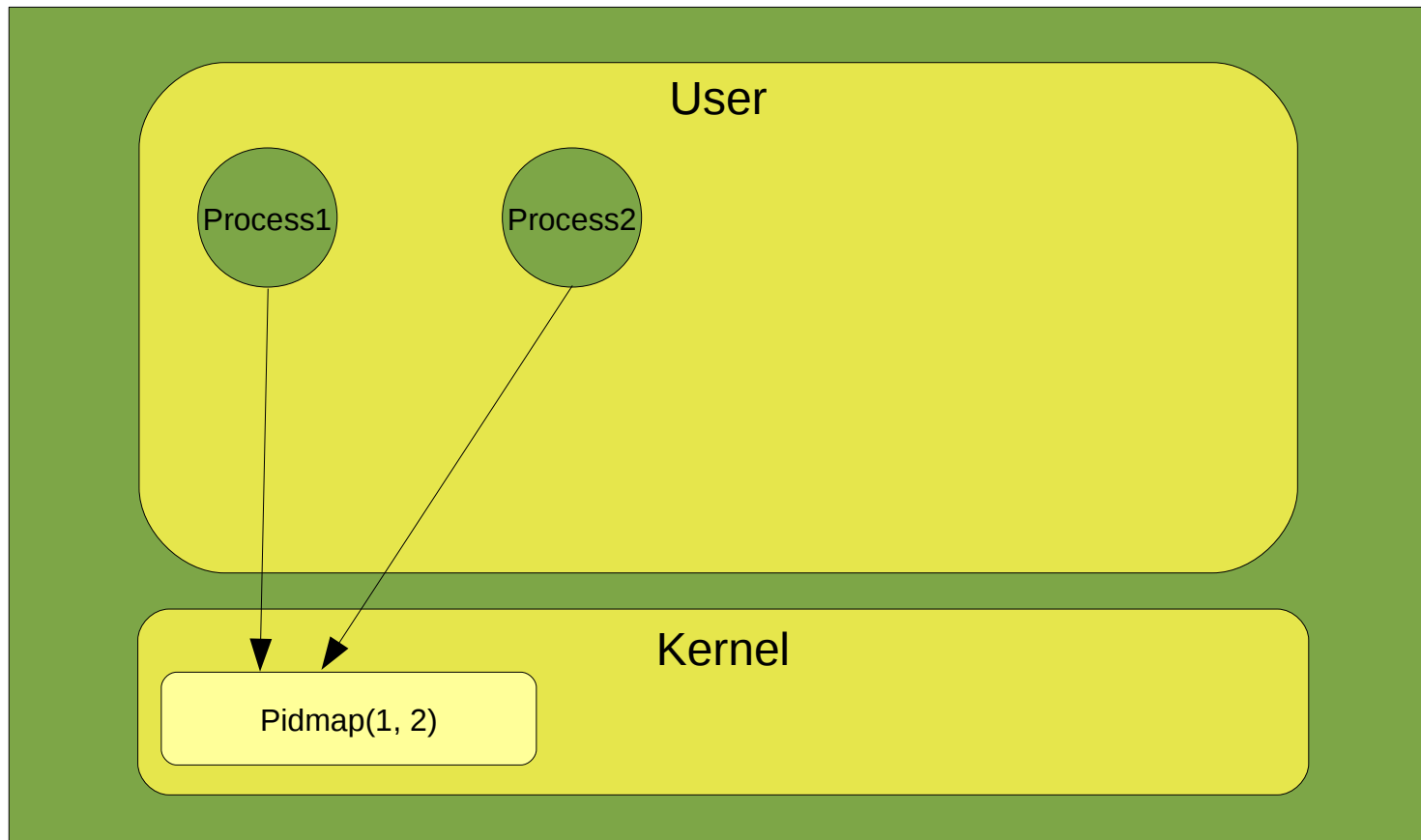
## Container – isolation example with the utsname



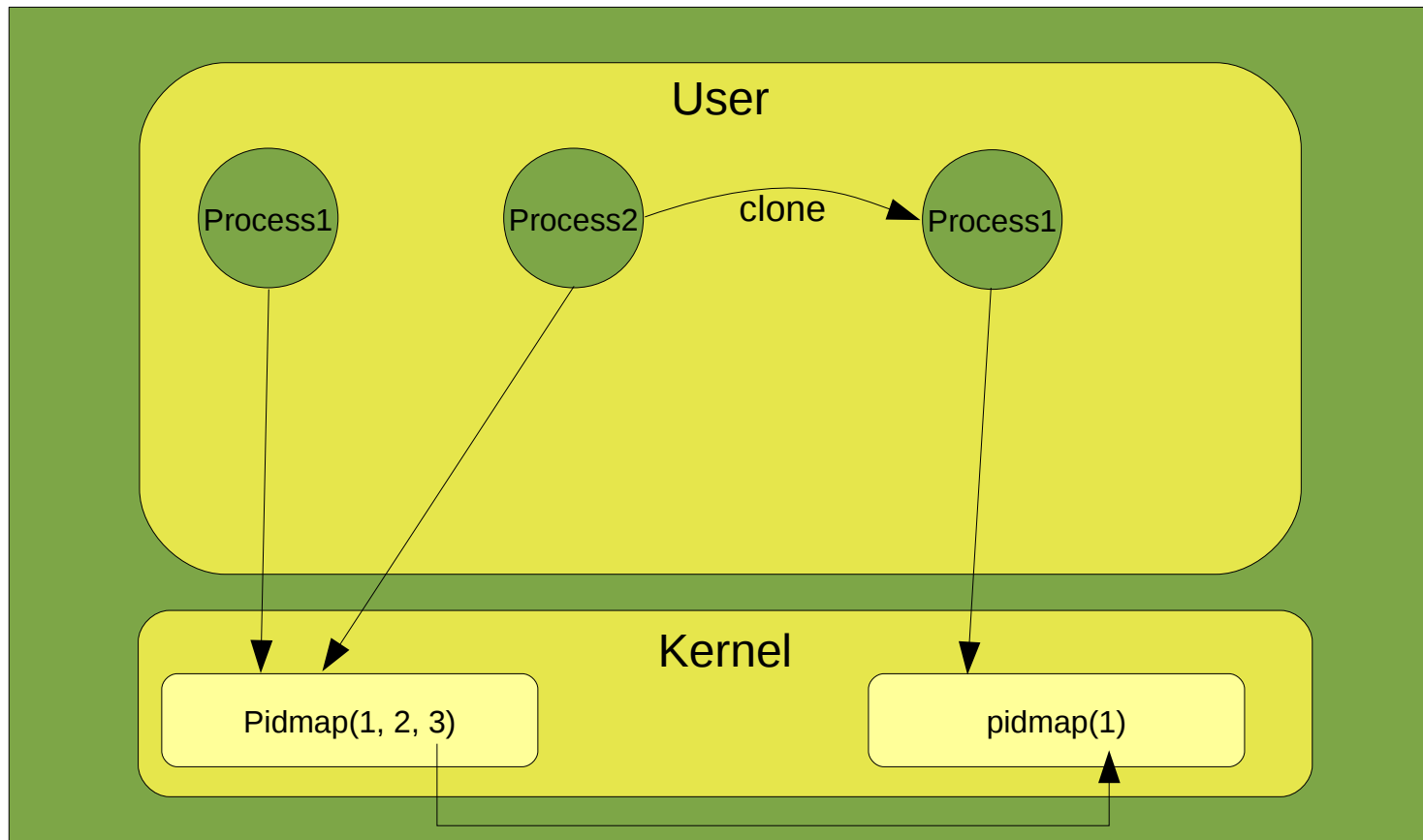
## Container – isolation example with the utsname



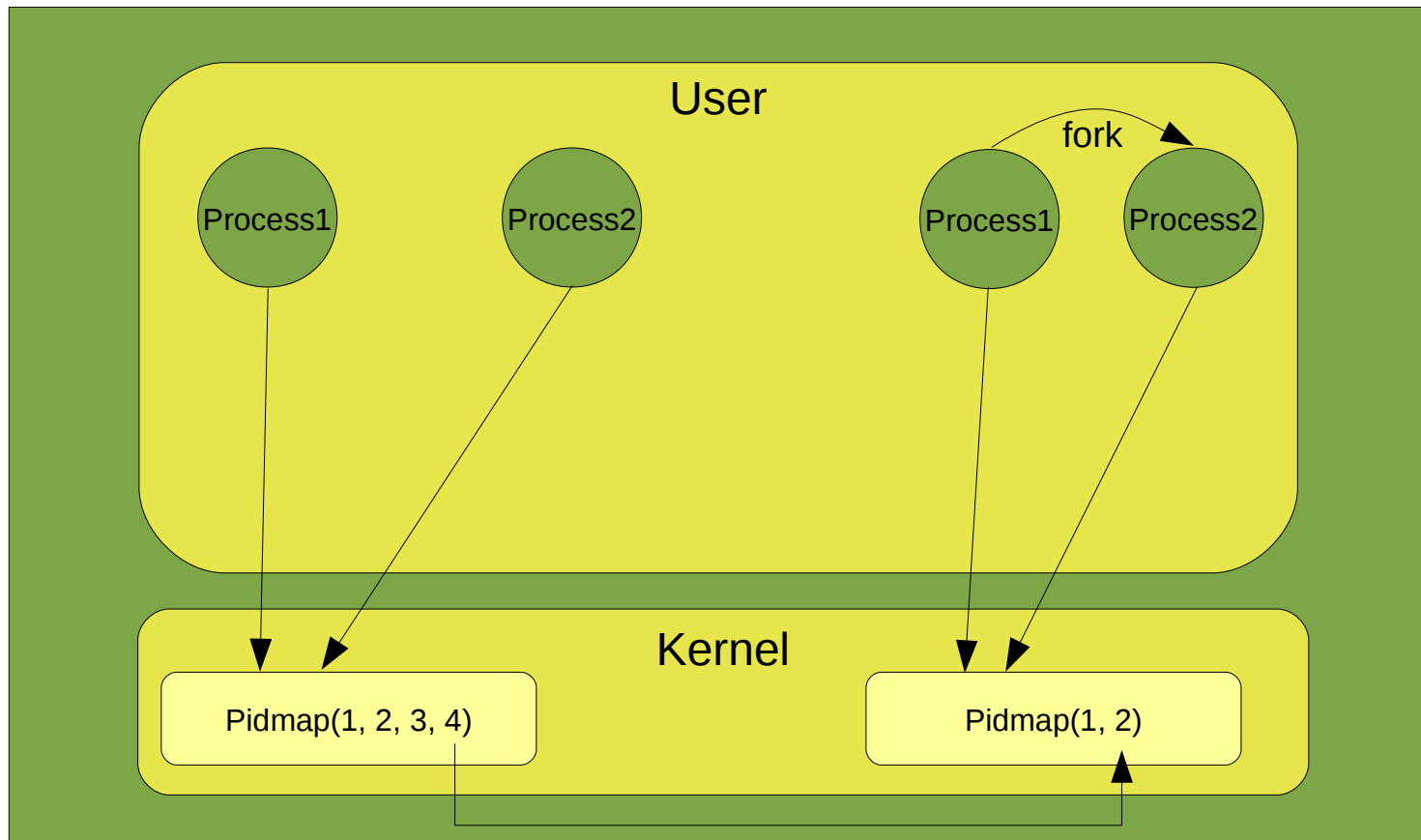
## Container – another isolation example with the pids



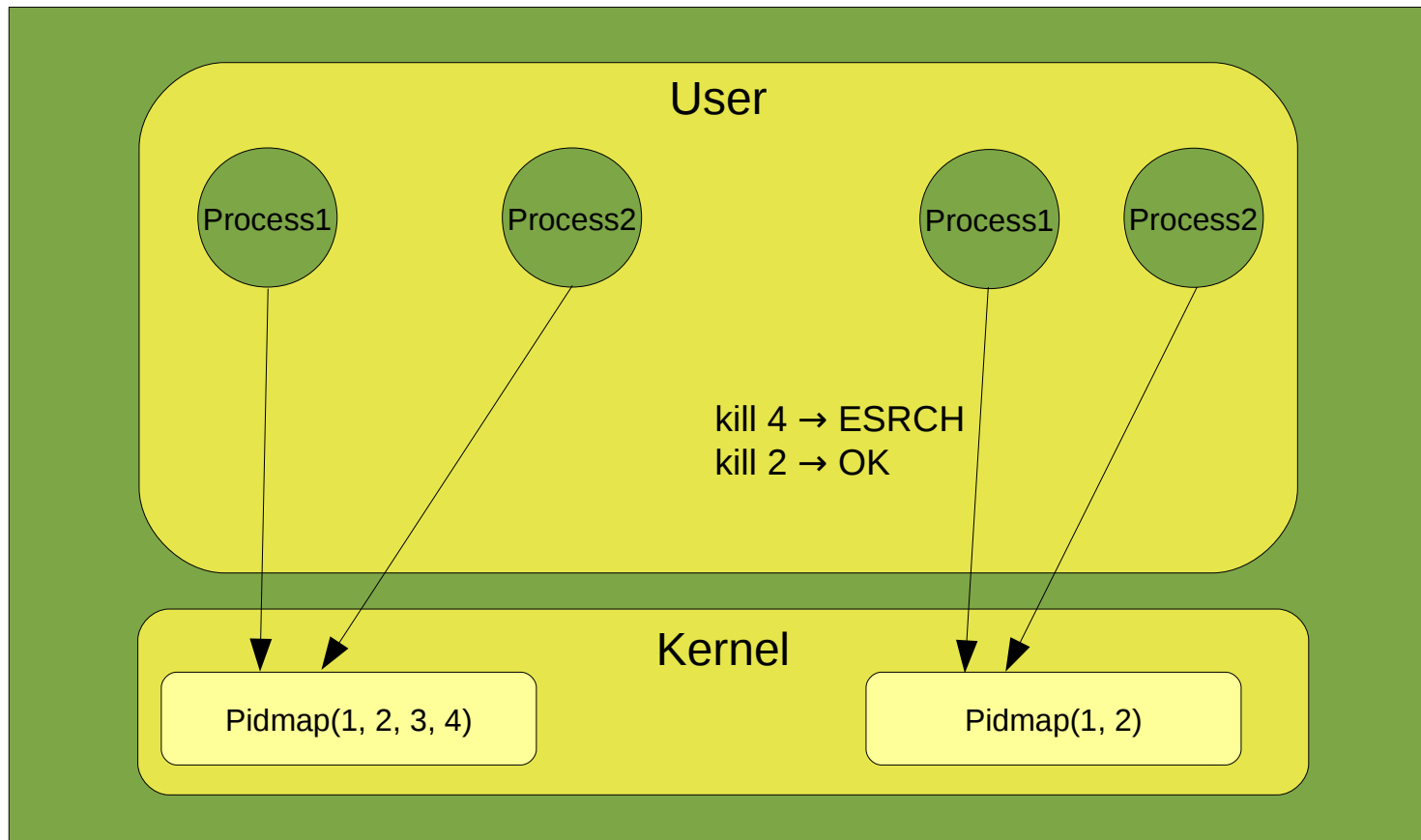
## Container – another isolation example with the pids



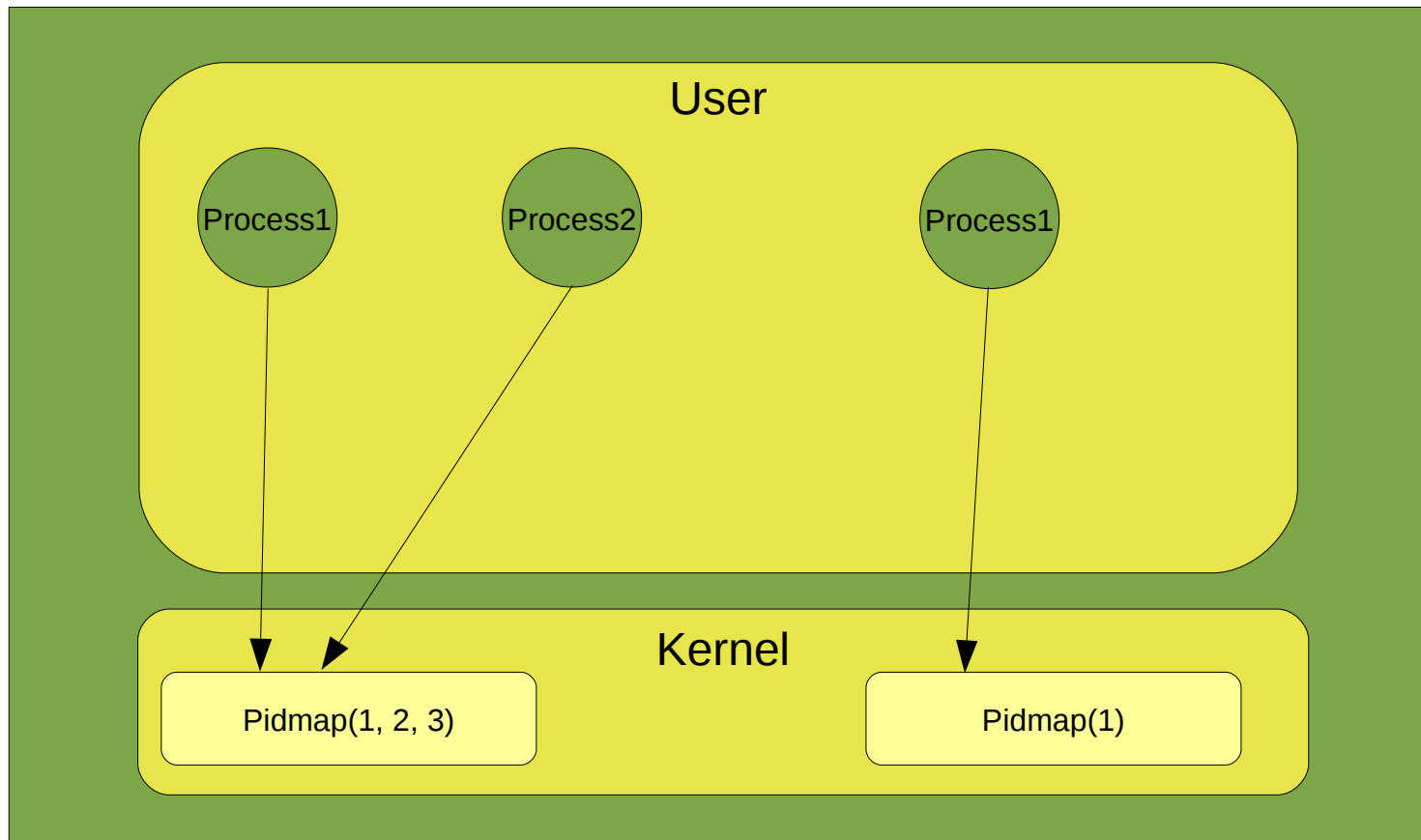
## Container – another isolation example with the pids



## Container – another isolation example with the pids



## Container – another isolation example with the pids



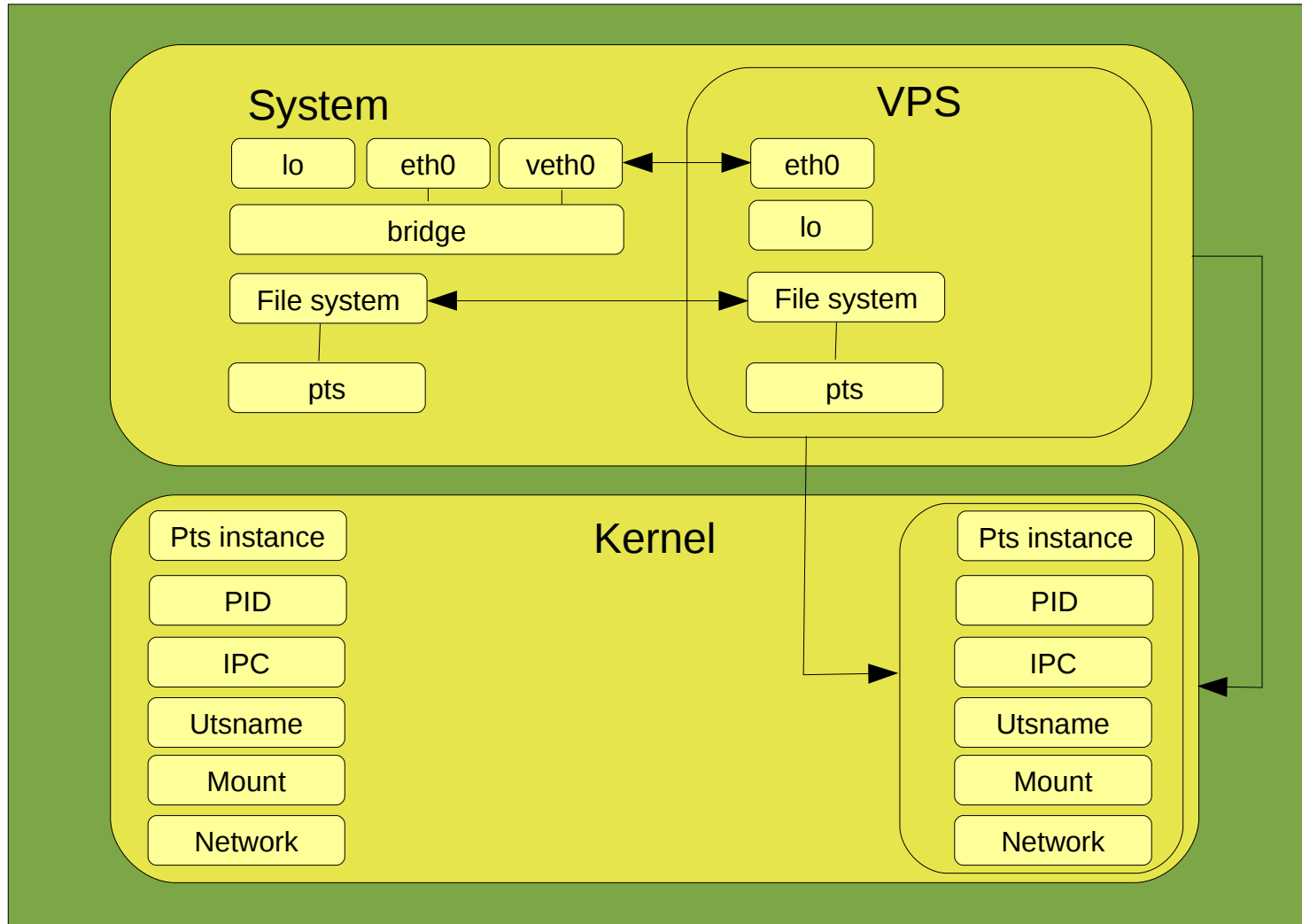
## Container – some observations

- Same identifiers for different processes
  - Resources identifiers get virtualized
- System resources are directly accessed in the process context
  - Performances are not impacted
- Trivial syscall
  - Easy implementation
  - Fine grained isolation

## Container – namespace status

- All the needed namespaces are in the mainstream kernel since 2.6.29
  - Mount points
  - IPC (sysV and posix)
  - Tasks (processes, threads, ...)
  - Network
  - Utsname
  - New Pts instance
- 
- Future namespaces
    - User (partially in mainstream)
    - Time (?)

## Container – A full VPS



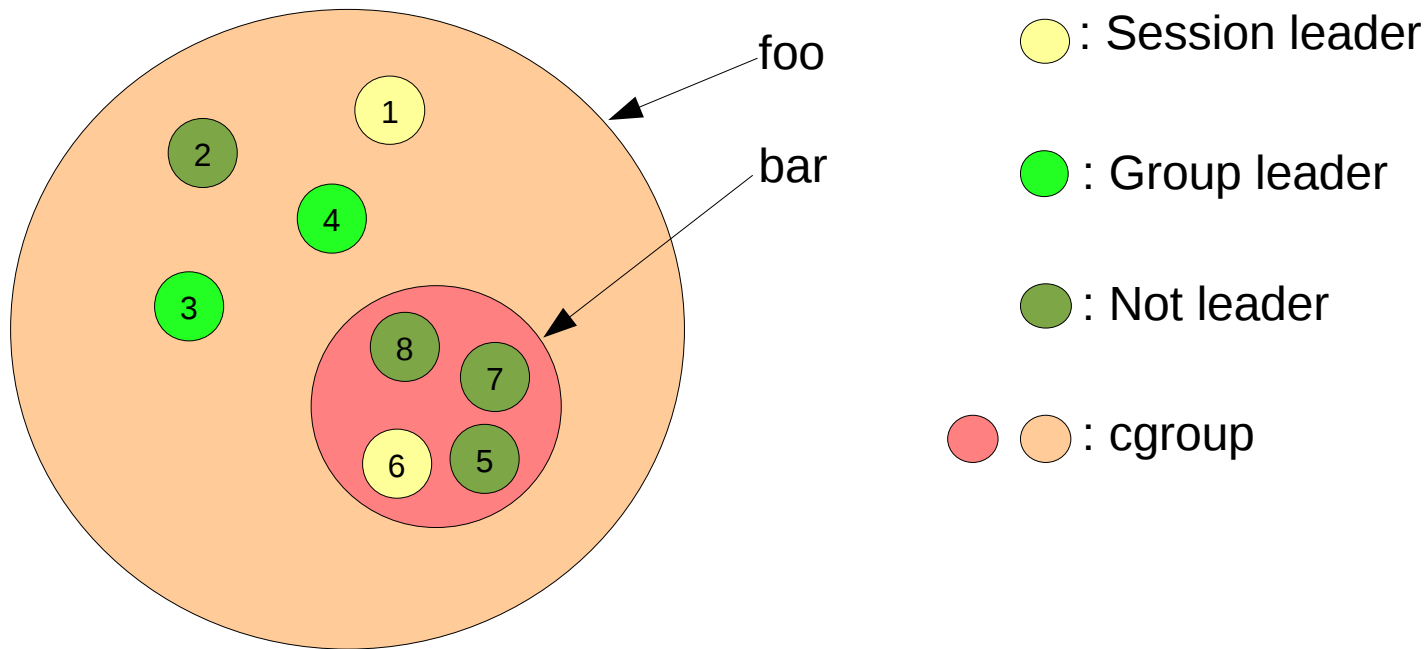
## Container - aggregation

- A set of tasks are grouped under a single identifier
- This identifier is like a “super-session leader”
- The kernel provides a framework to group the processes:
  - the control group (aka process container)
  - manageable through a filesystem
- The cgroup allows to track the processes

## Container – cgroup : a little example

- Mount the cgroup somewhere
  - `mount -t cgroup cgroup /cgroup`
  
- Create one cgroup
  - `mkdir /cgroup/foo`
  - `/cgroup/foo` is automatically populated
  
- Assign a process number 1 to the cgroup
  - `echo 1 > /cgroup/foo/tasks`
  - At this point all the child processes will be in 'foo'
  
- Create another one, nested with the first one
  - `mkdir /cgroup/foo/bar`
  
- Assign a process to this cgroup
  - `Echo '5' > /cgroup/foo/bar/tasks`

## Container – cgroup and process tracking



Tasks belonging to 'foo' : 1, 2, 3, 4

Tasks belonging to 'bar' : 5, 6, 7, 8

## Container – cgroup and subsystems

- The system resources are controlled with a subsystem relying on the cgroup
- The cgroup is designed to include new subsystems
- Some subsystems
  - Freezer
  - Memory controller
  - Cpu accounting
  - Devices white list
  - Cpuset
  - Network bandwidth
  - And more are coming ! IO quota, etc ...

## Container – userspace tools

– Just one userspace tool working with a non-patched kernel

- <http://lxc.sourceforge.net>

– Available in the development repositories:

- Fedora
- Debian
- OpenSuse
- Ubuntu
- Foresight Linux
- AltLinux

– A little demo ...

## Checkpoint / Restart - Definitions

“Take a snapshot of an application, this snapshot is converted into data which can be reused later for restarting the application in the same state”

- [http://en.wikipedia.org/wiki/Checkpoint\\_restart](http://en.wikipedia.org/wiki/Checkpoint_restart) :

“[ ... ] The purpose of checkpointing is to minimize the amount of time and effort wasted when a long software process is interrupted by a hardware failure, a software failure, or resource unavailability. With checkpointing, the process can be restarted from the latest checkpoint rather than from the beginning [ ... ]”

## Checkpoint / Restart – use cases

- Golden image, fast application start-up
- Predictive fail over tied to system health monitoring framework
- Manage quality of service by moving application across the servers
- Record and Replay, application execution time travel (tuning and debugging)
- Long running job protection

## Checkpoint / Restart

- Existing solutions:
  - Berkeley Checkpoint / Restart aka bcr
  - OpenVZ, system container
  - Metacluster, application container
  - Community initiative developing the Checkpoint / Restart for a mainline inclusion attempt
  - And much more exotic checkpoint / restart ...

## Checkpoint / Restart – Technical problems

- Identify the resources belonging to an application to be checkpointed / restarted
- Avoid resource identifier conflicts at restart
  - One concept solving all the problems : the container
- The resources are isolated : identify
- The resource identifiers are virtualized : avoid conflicts

## Checkpoint / Restart – Metacluster

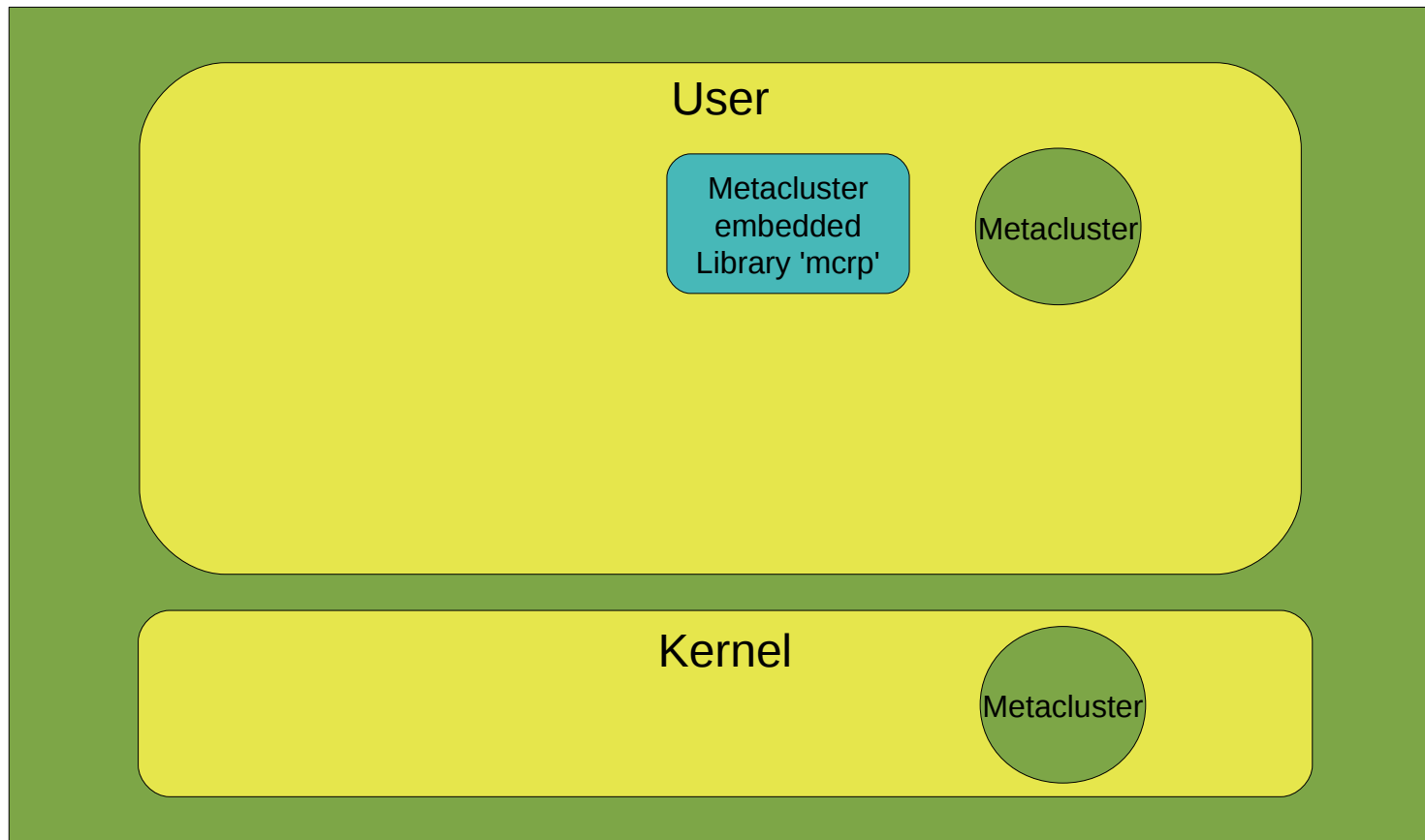
- An IBM home made Checkpoint / Restart
- Hybrid implementation : User and Kernel space
- Supports x86, x86\_64, s390 and ppc architectures
- Used in the HPC4U project : <http://www.hpc4u.eu/>
- Used in the PERCS project :
  - More than 100.000 cpus !
  - <http://www.almaden.ibm.com/StorageSystems/projects/percs/>
  - <http://www.hpcwire.com/features/17883339.html>
  - [http://domino.research.ibm.com/comm/pr.nsf/pages/news.20030710\\_darpa.html](http://domino.research.ibm.com/comm/pr.nsf/pages/news.20030710_darpa.html)

## Checkpoint / Restart - Implementation

- Checkpoint / Restart occurs in 3 steps
  - Reach a quiescent point for the application (freeze)
  - Dump or restore the resources internal state and process hierarchy
  - Release the application (thaw)

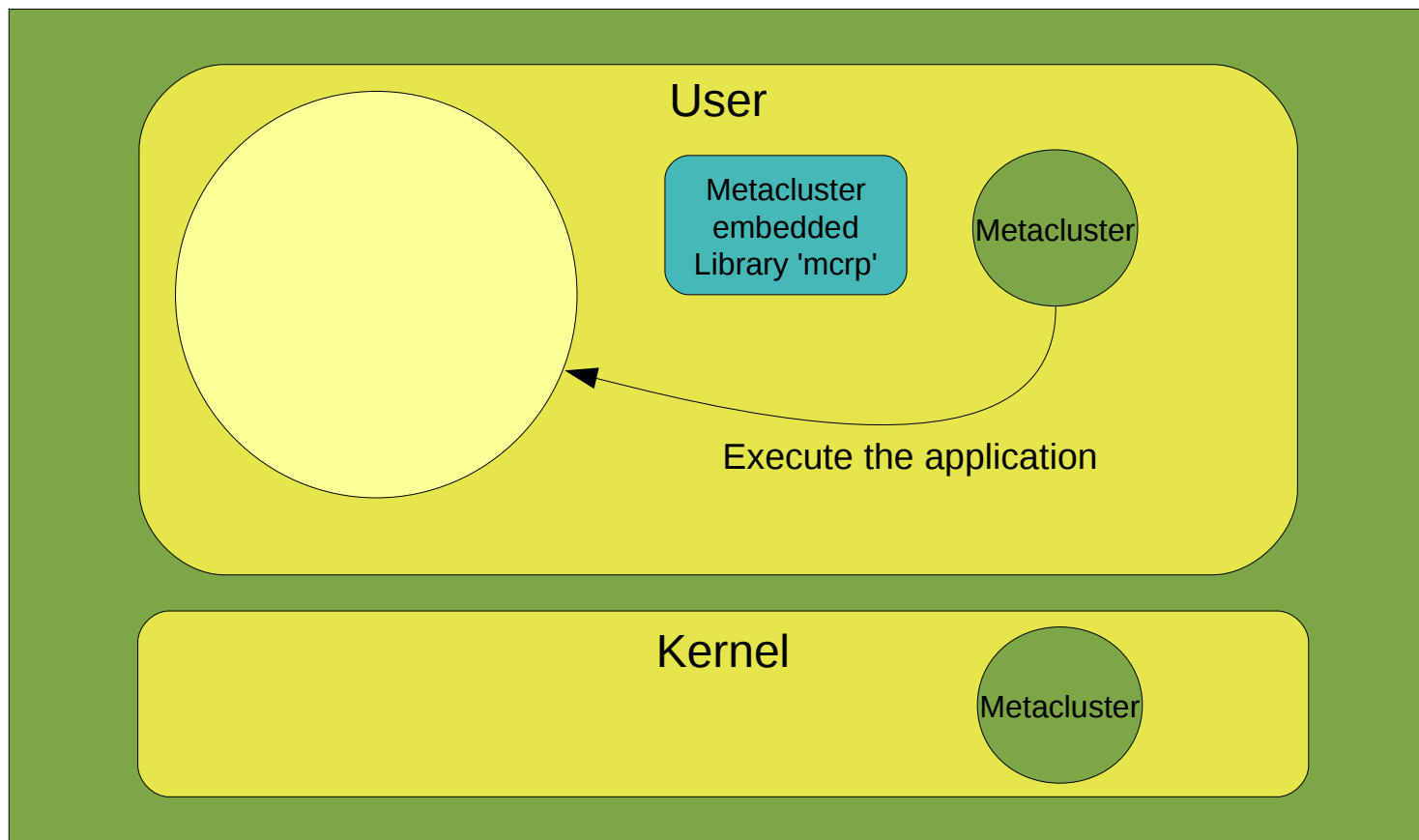
## Checkpoint / Restart

Initial state: a command line, a library and a kernel component



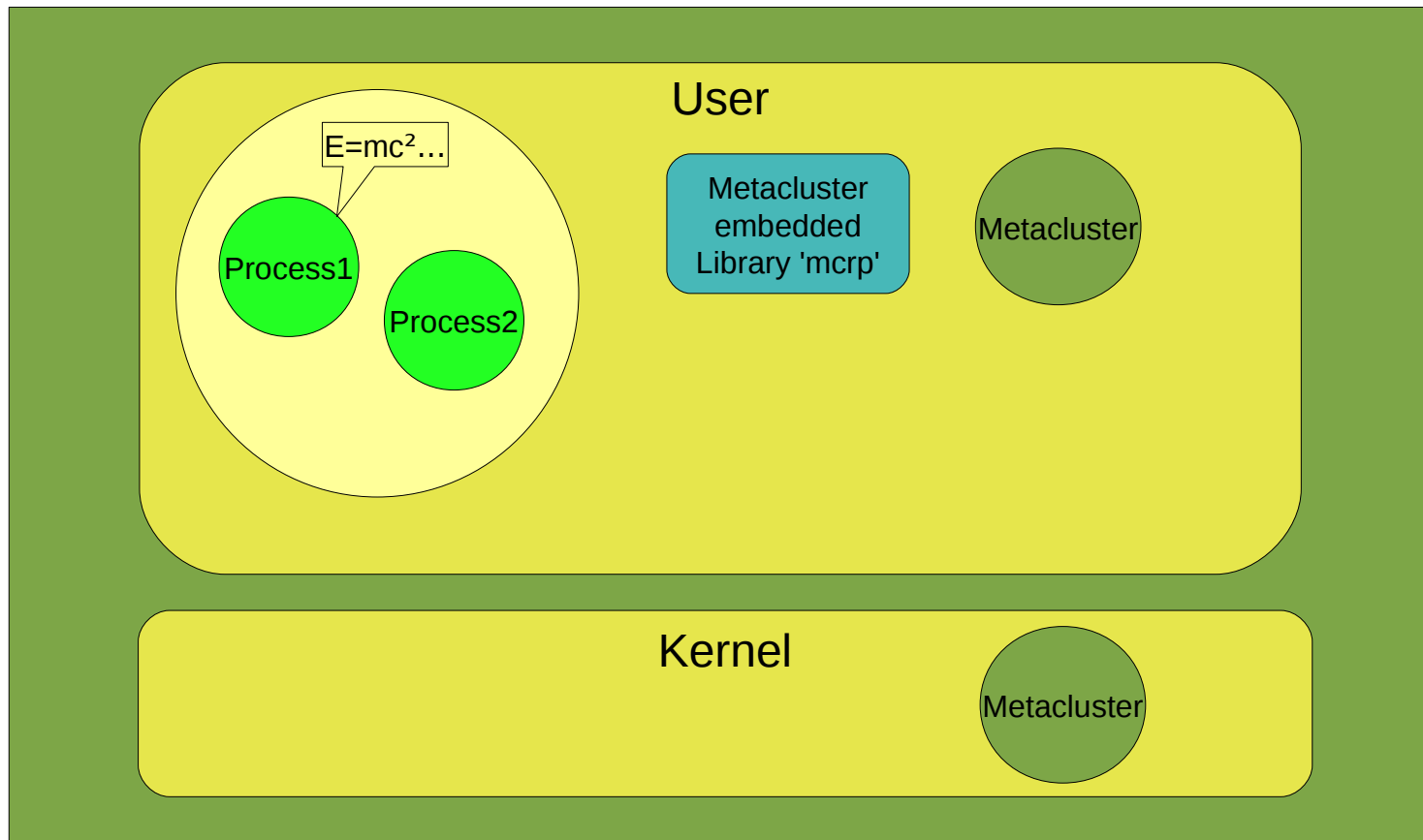
## Checkpoint / Restart

Metacluster creates the container and executes the application inside



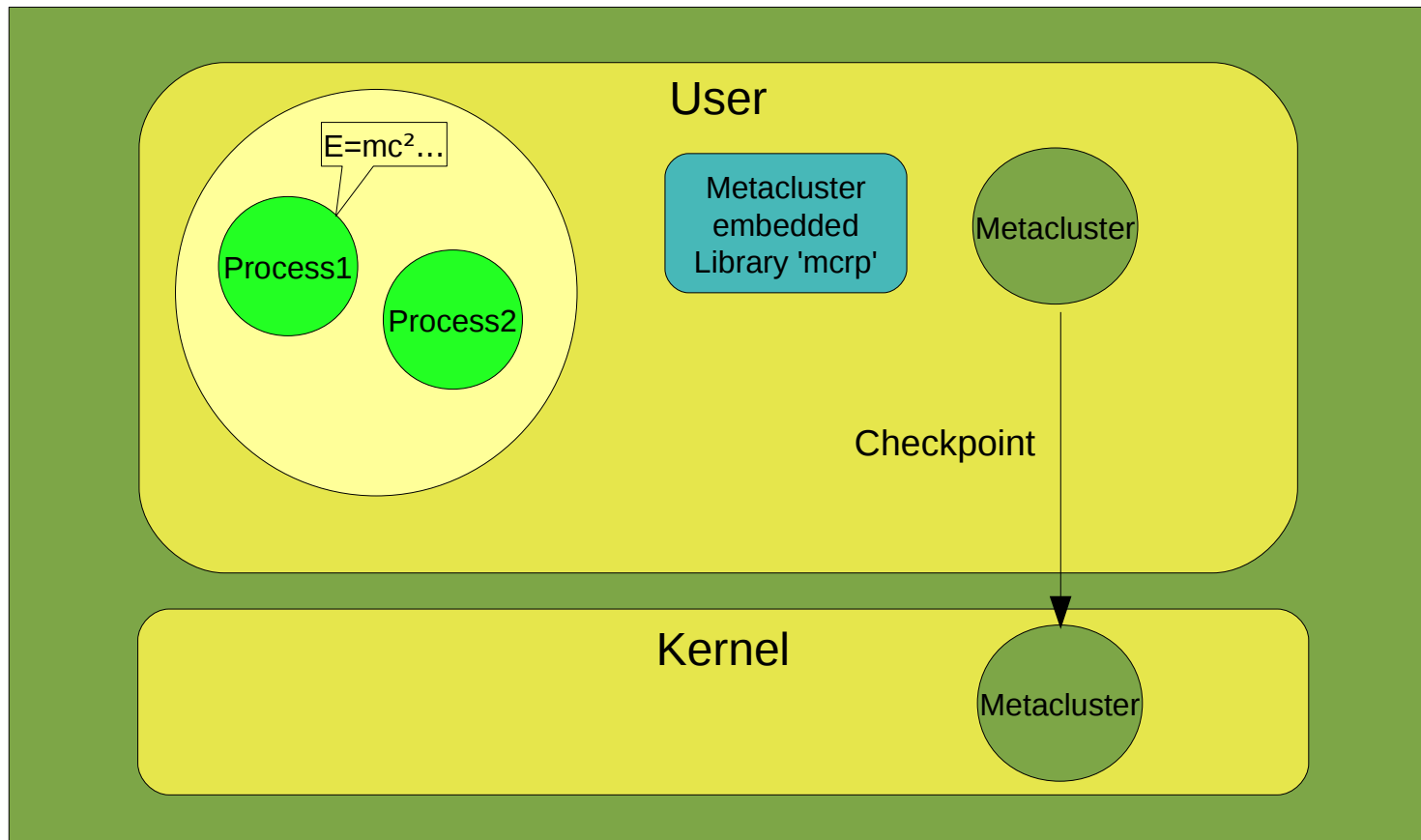
# Checkpoint / Restart

The application runs normally in the container



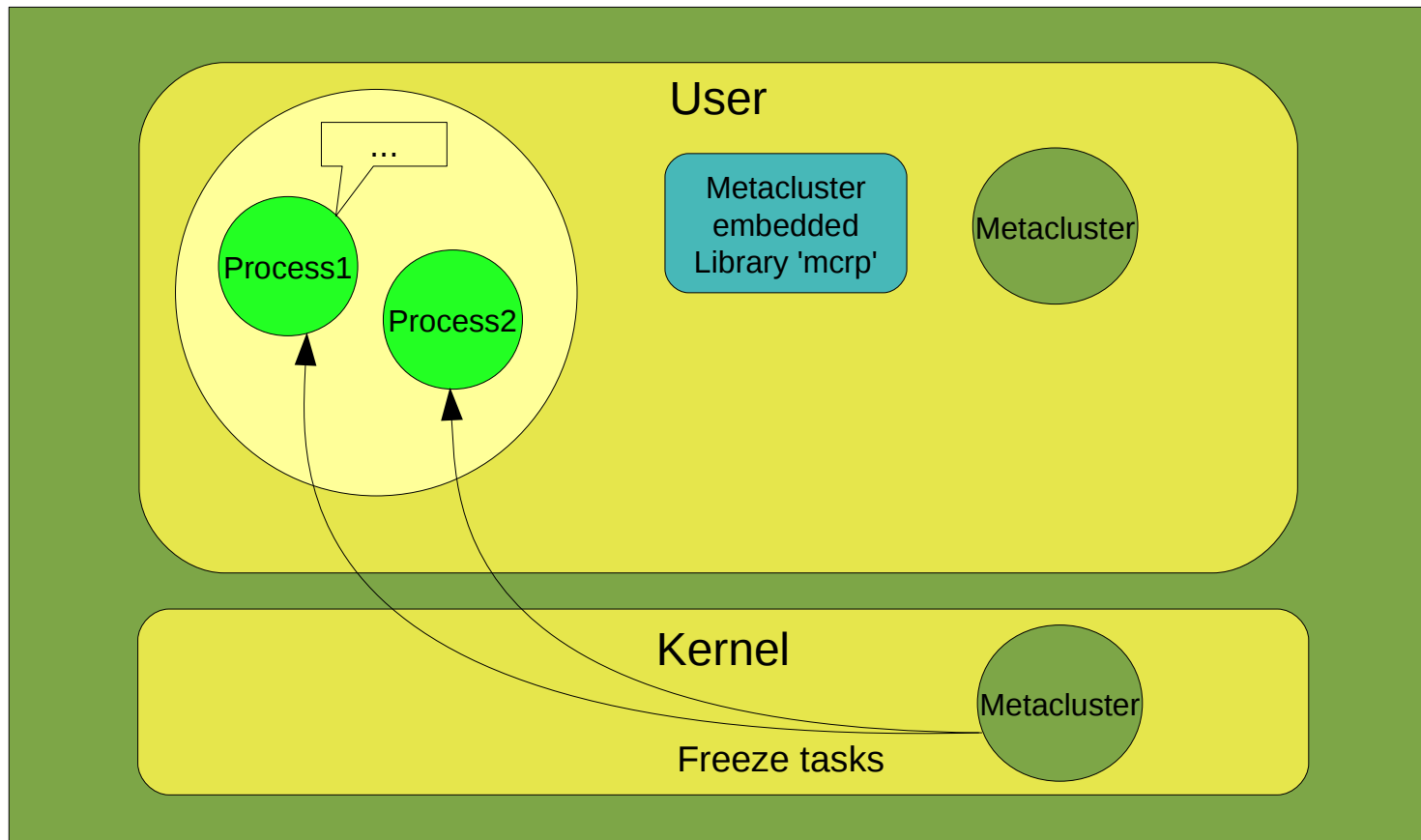
# Checkpoint / Restart

We initiate the checkpoint of this container



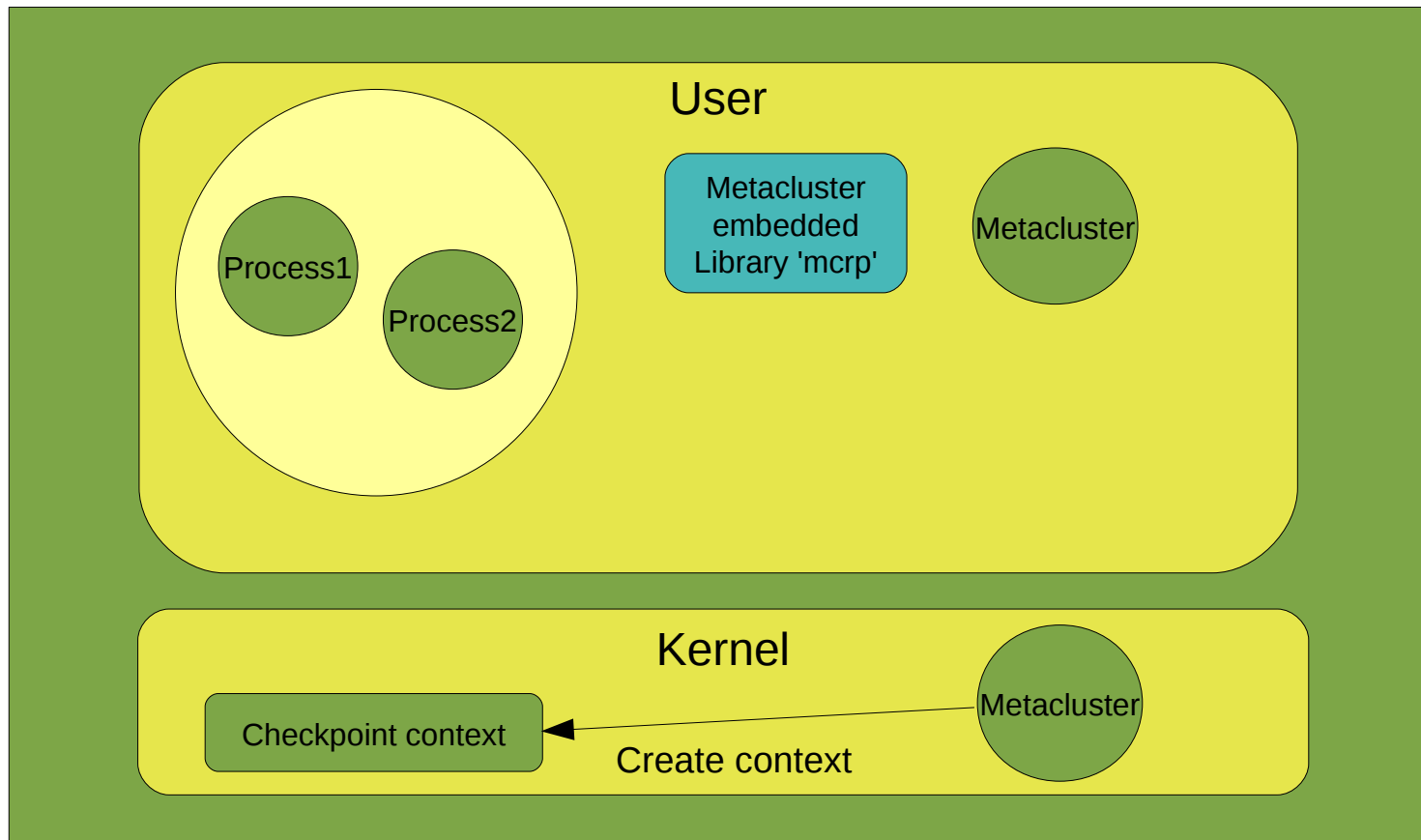
# Checkpoint / Restart

All the tasks of the container are frozen



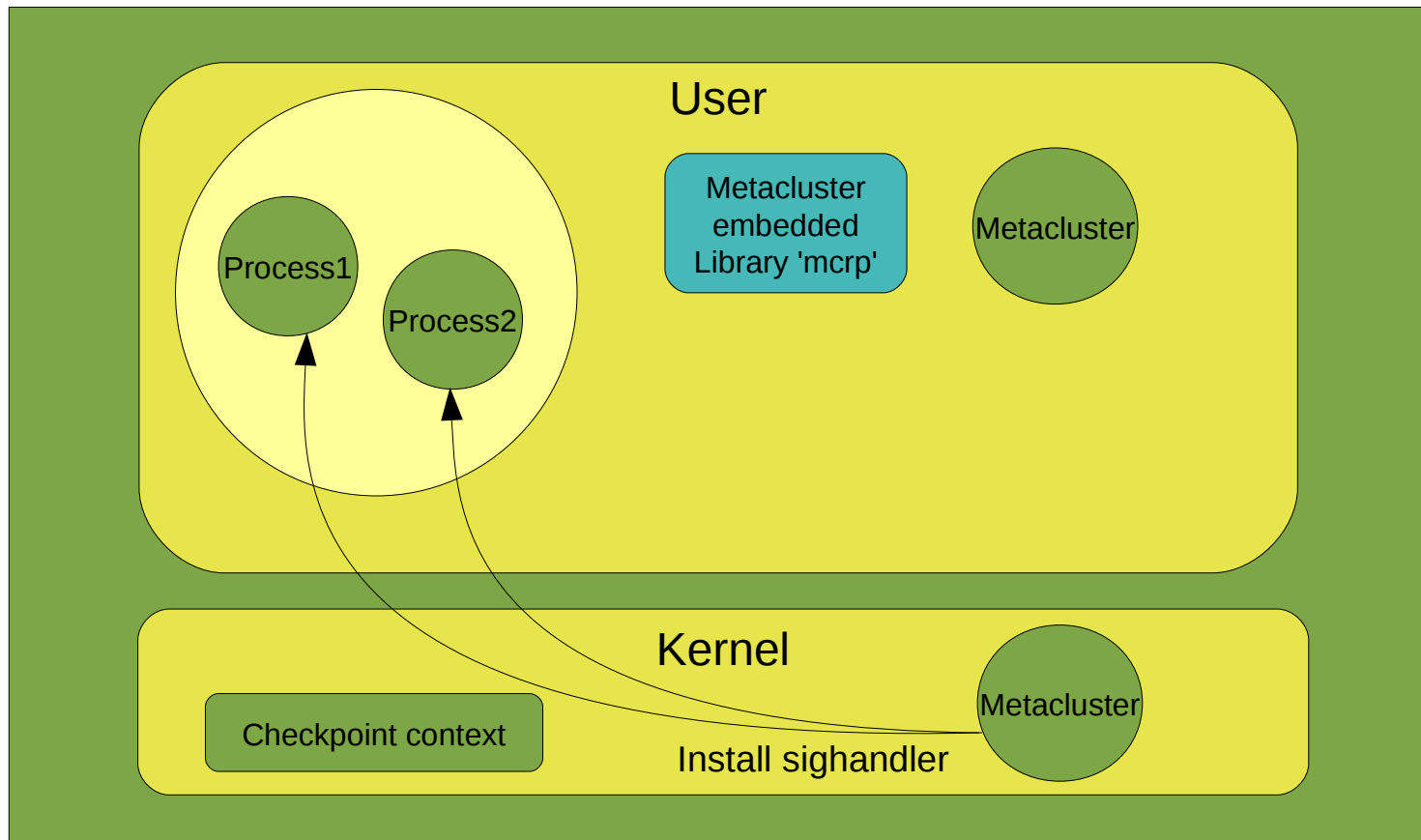
# Checkpoint / Restart

A checkpoint context is created for this container



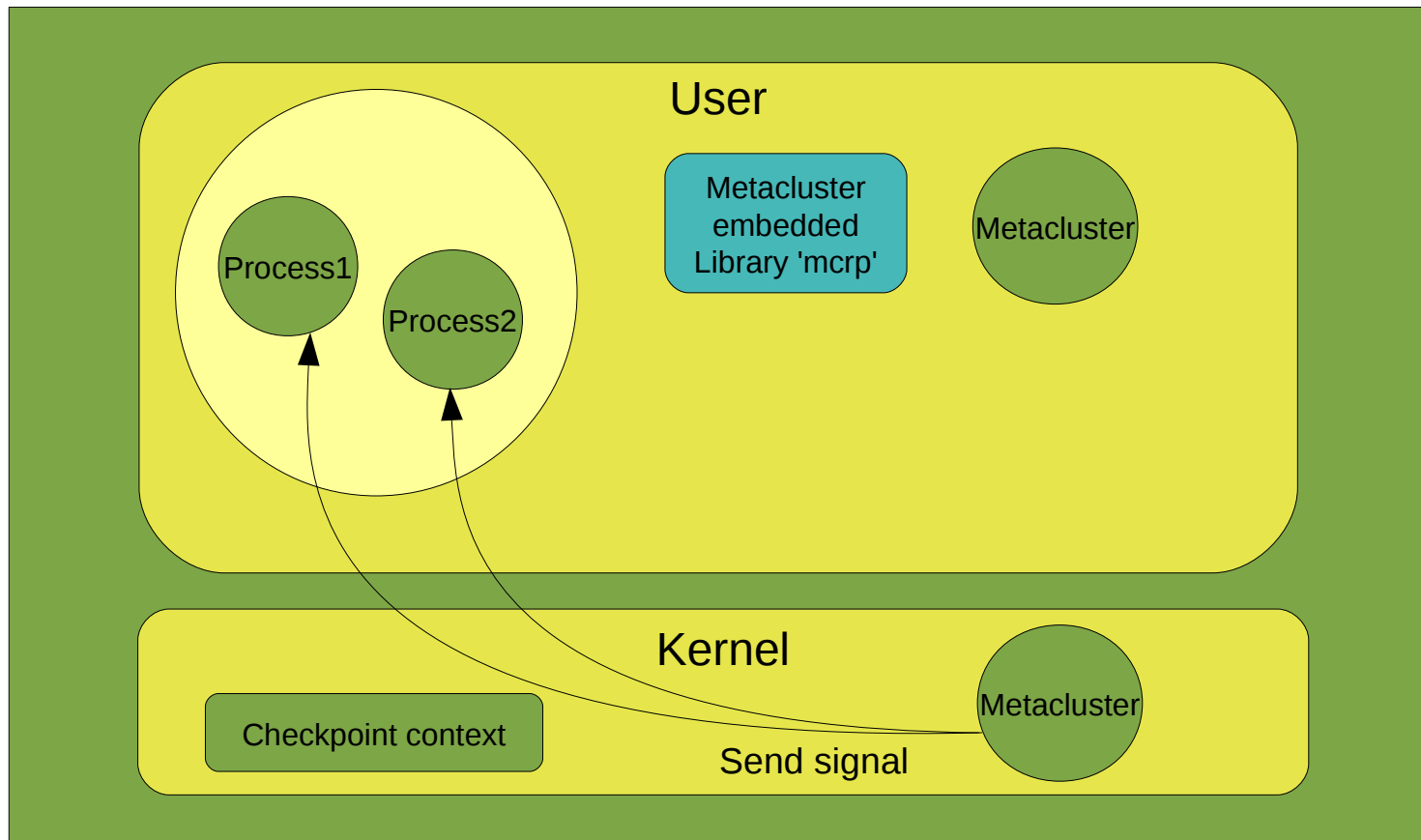
# Checkpoint / Restart

A sighandler corresponding to the library entry point is set on all the tasks



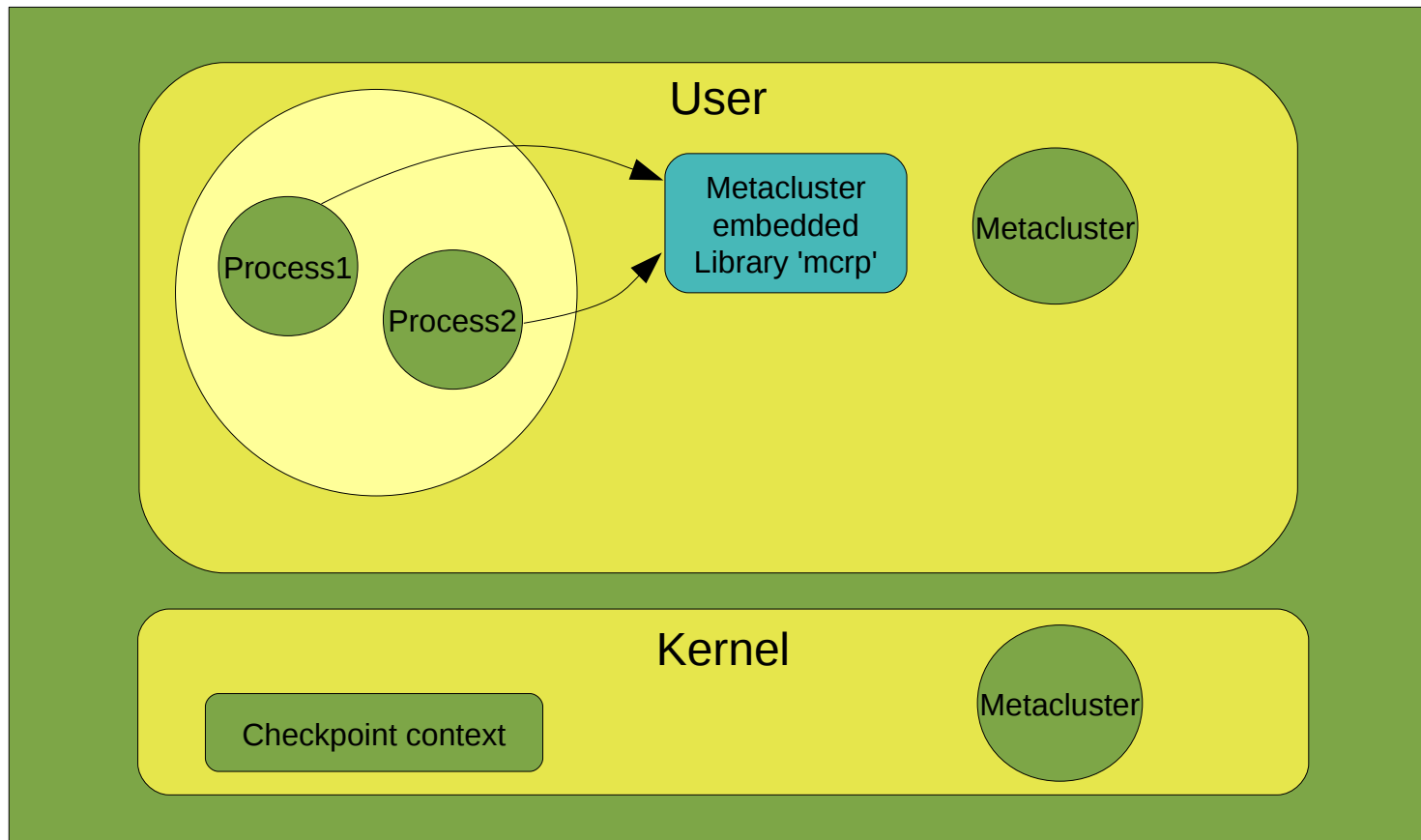
# Checkpoint / Restart

A specific signal is sent to all the tasks



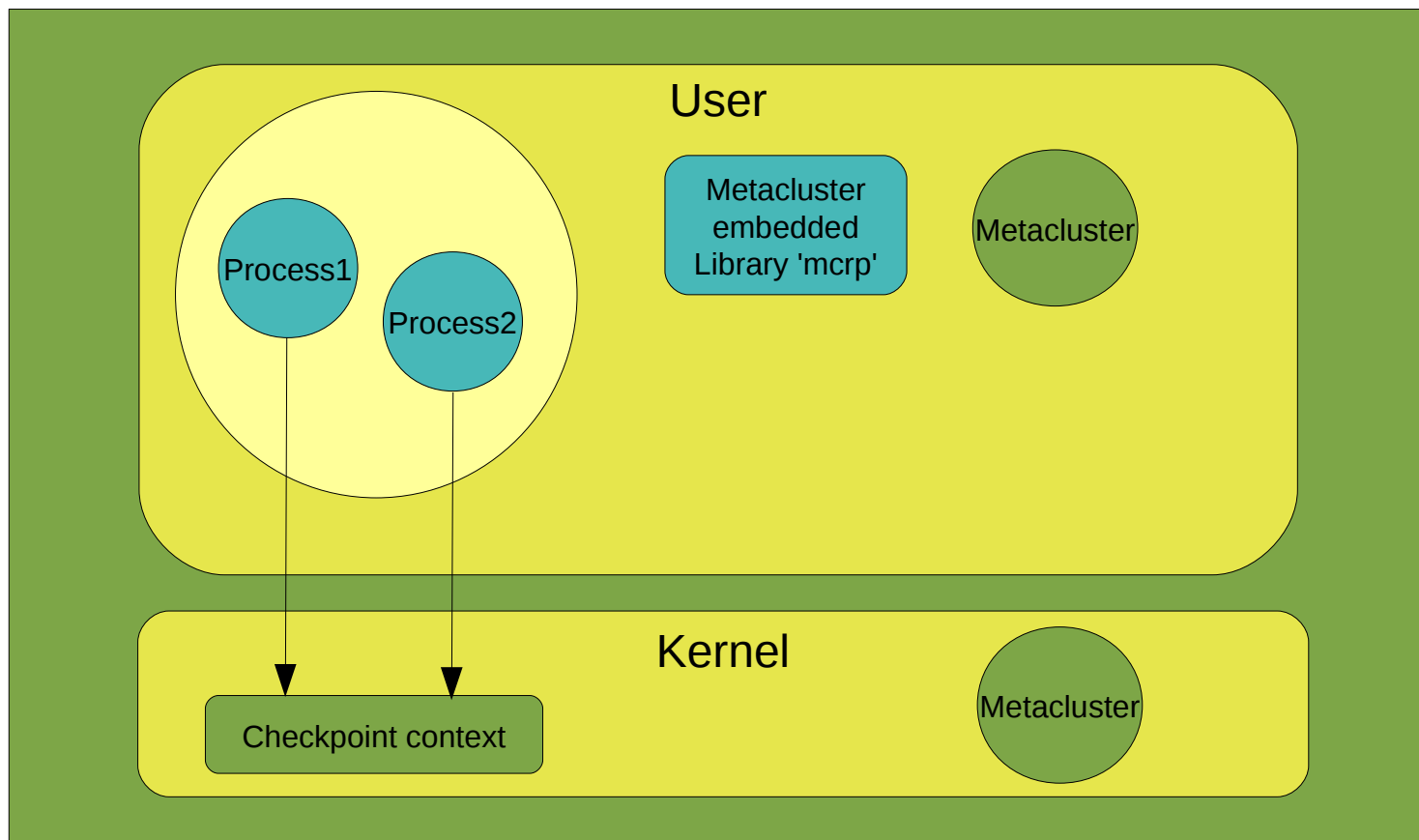
# Checkpoint / Restart

All the tasks execute the checkpoint library



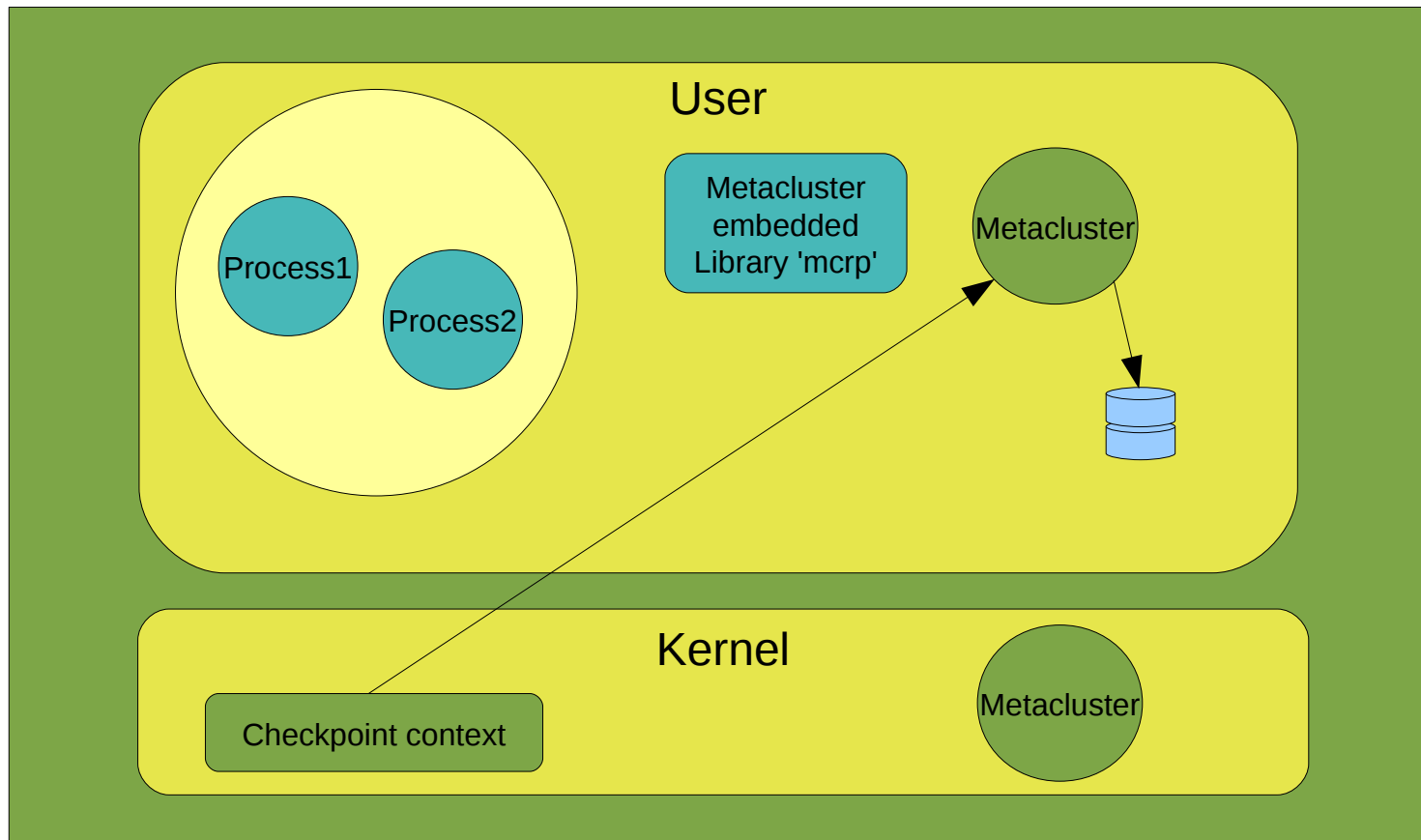
## Checkpoint / Restart

The library code stores process information into the checkpoint context



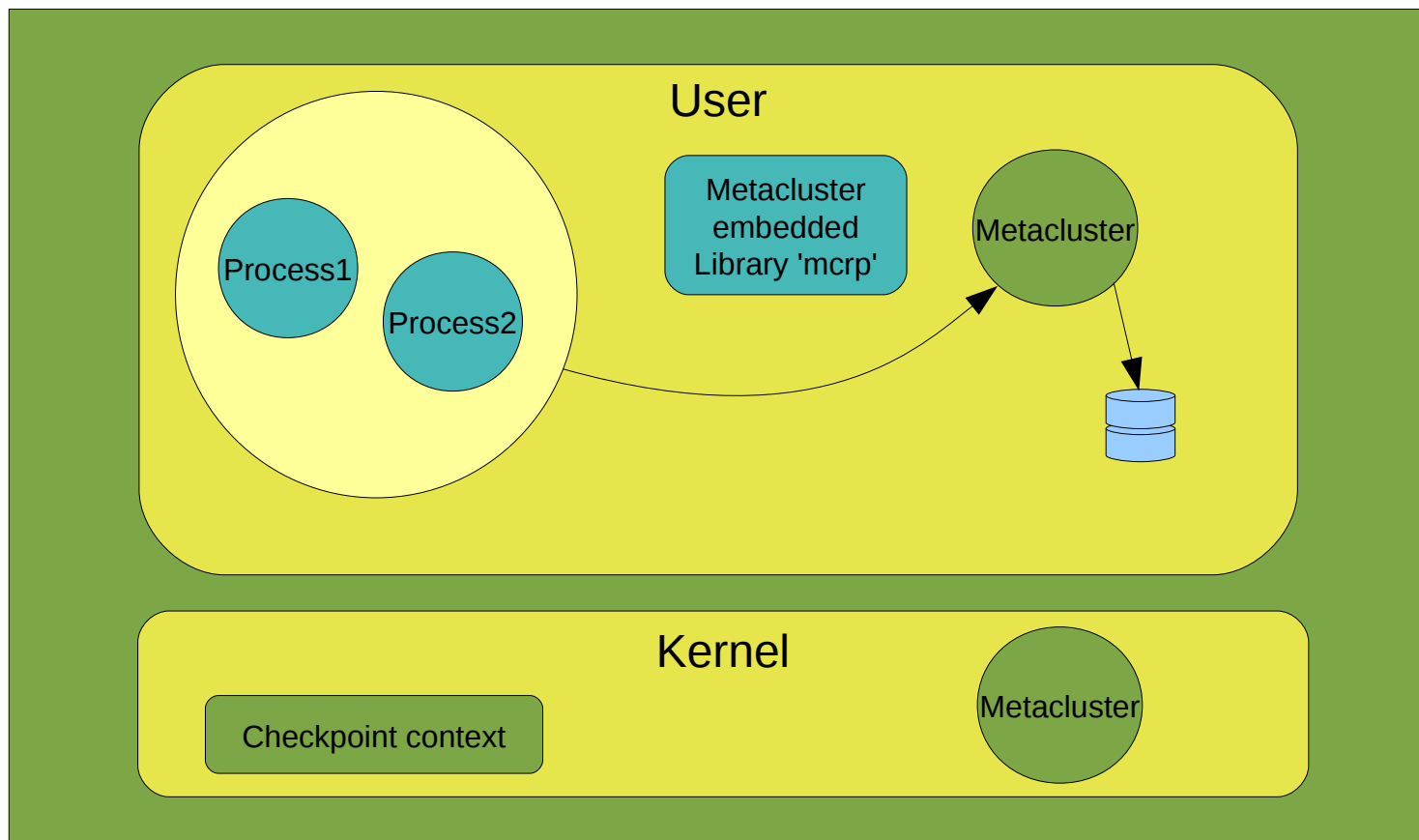
# Checkpoint / Restart

Metacluster collects this context and store it in the statefile



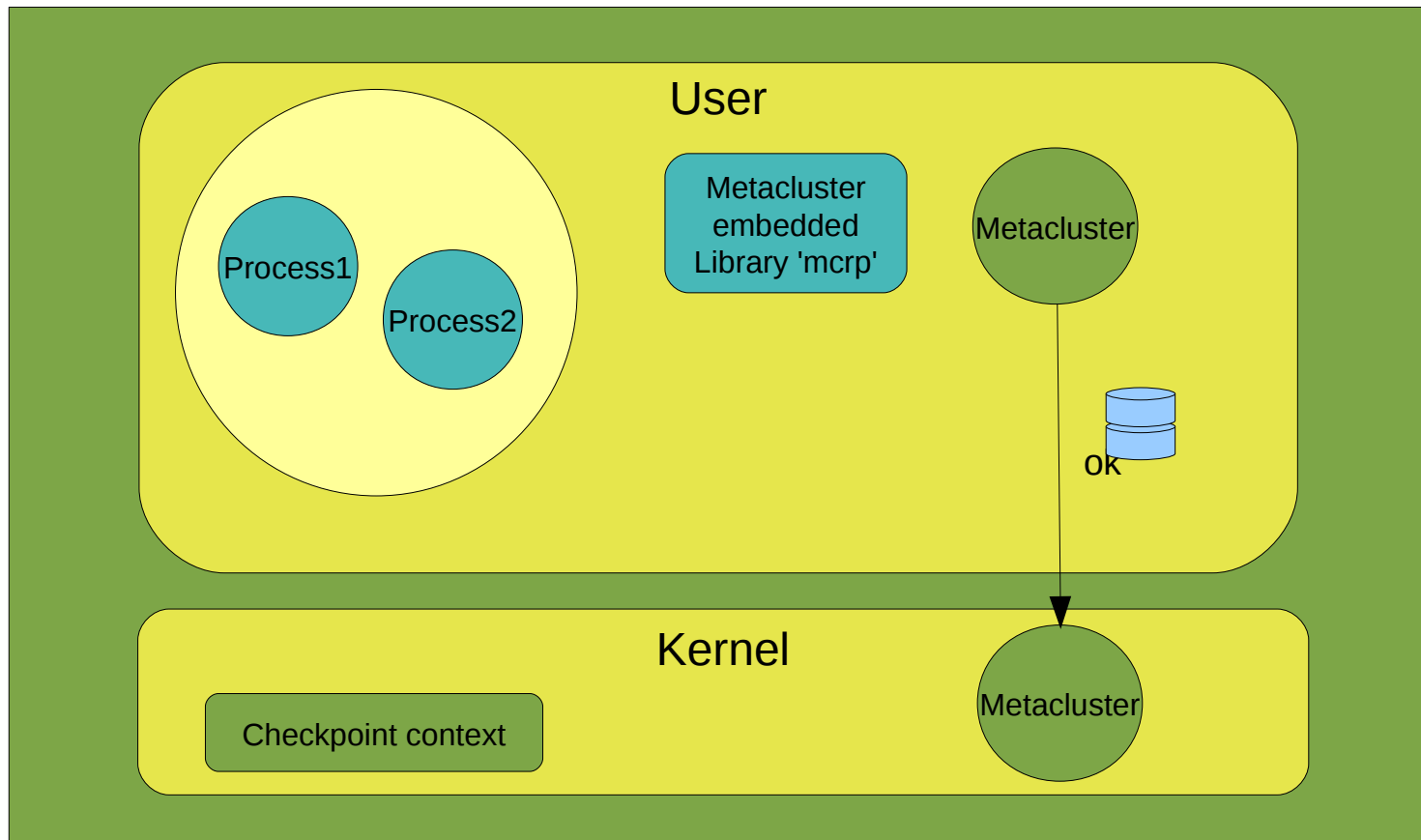
## Checkpoint / Restart

Metacluster collects container's system wide resources and store them in the statefile



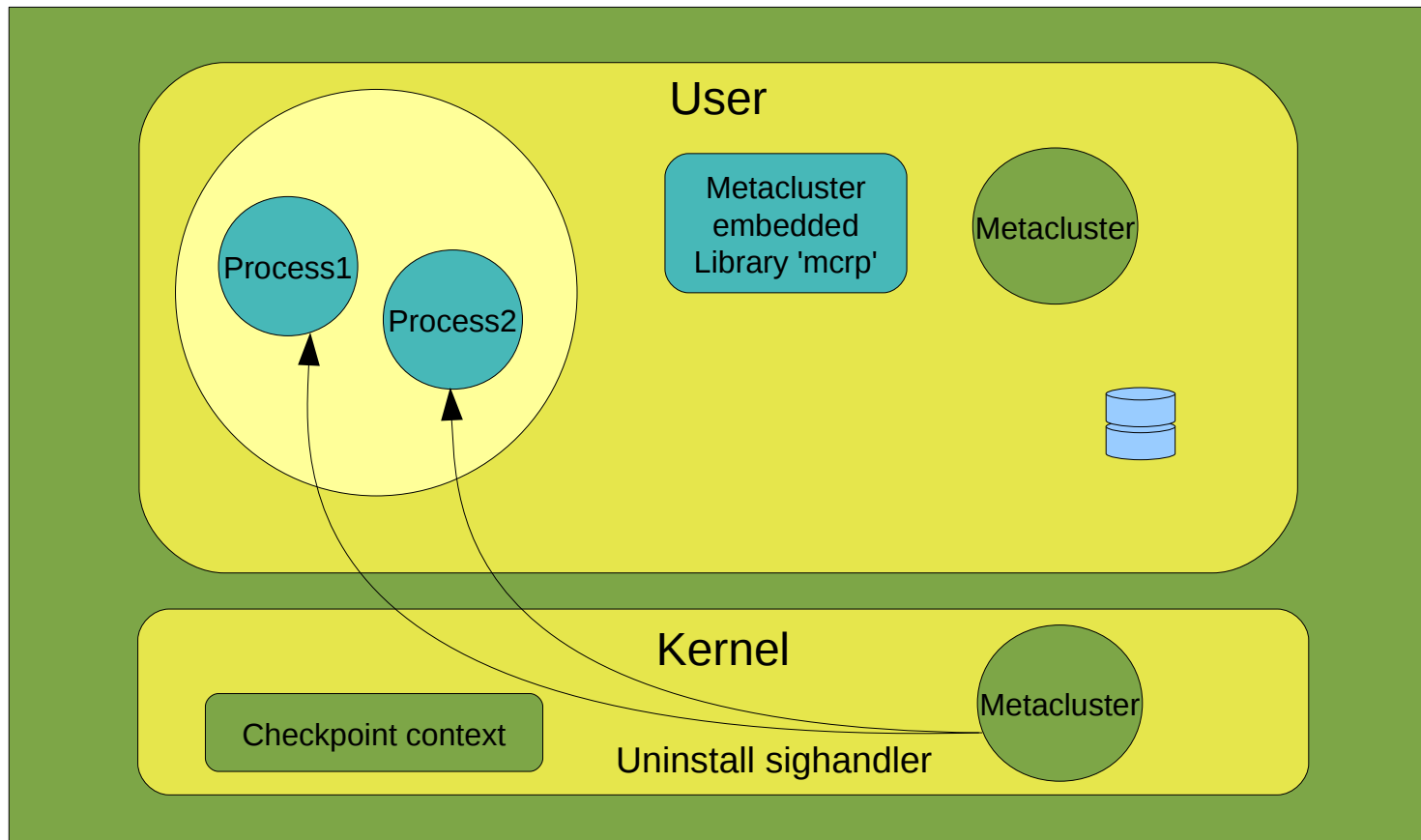
# Checkpoint / Restart

Metacluster notify the checkpoint is finished



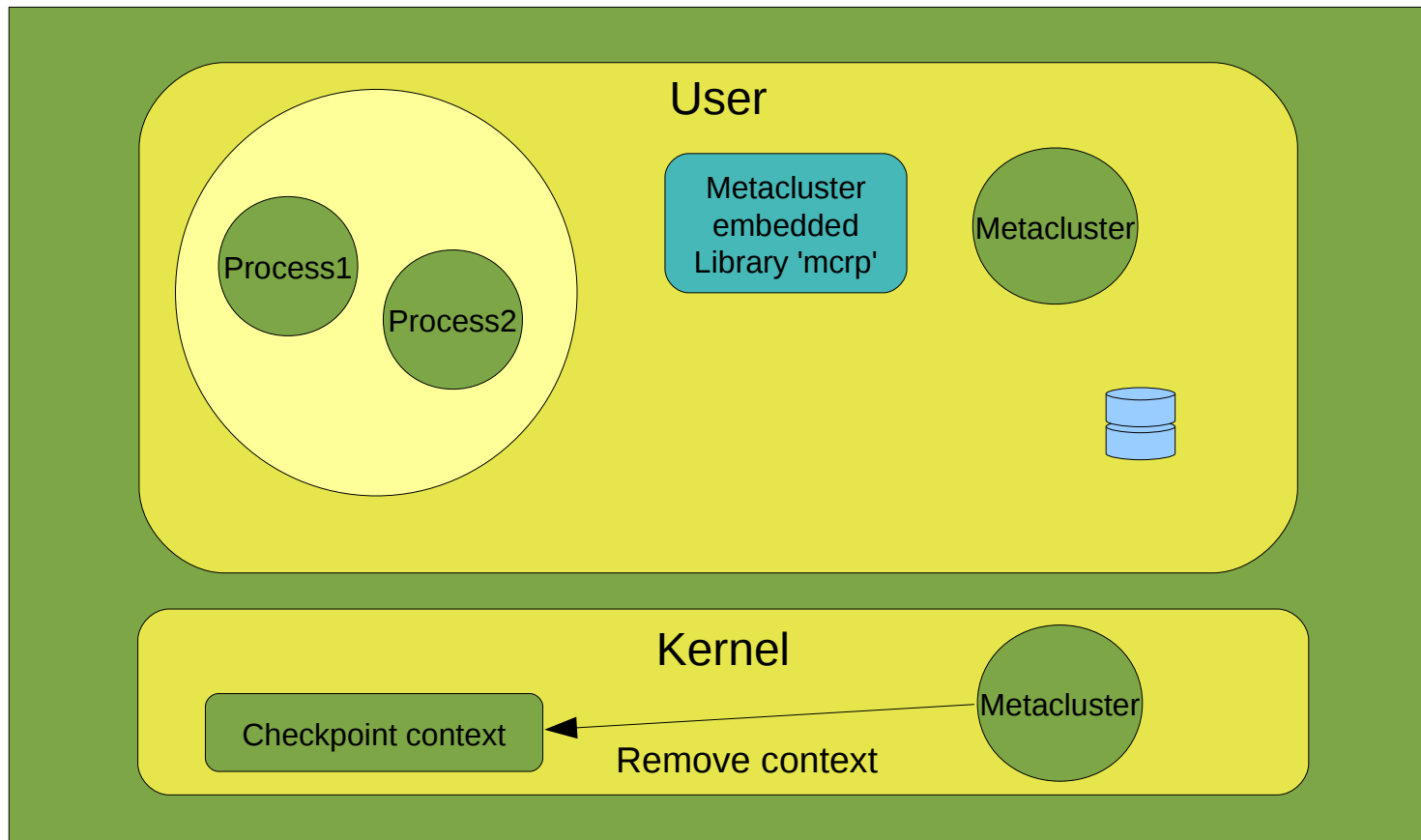
# Checkpoint / Restart

The sighandlers are uninstalled



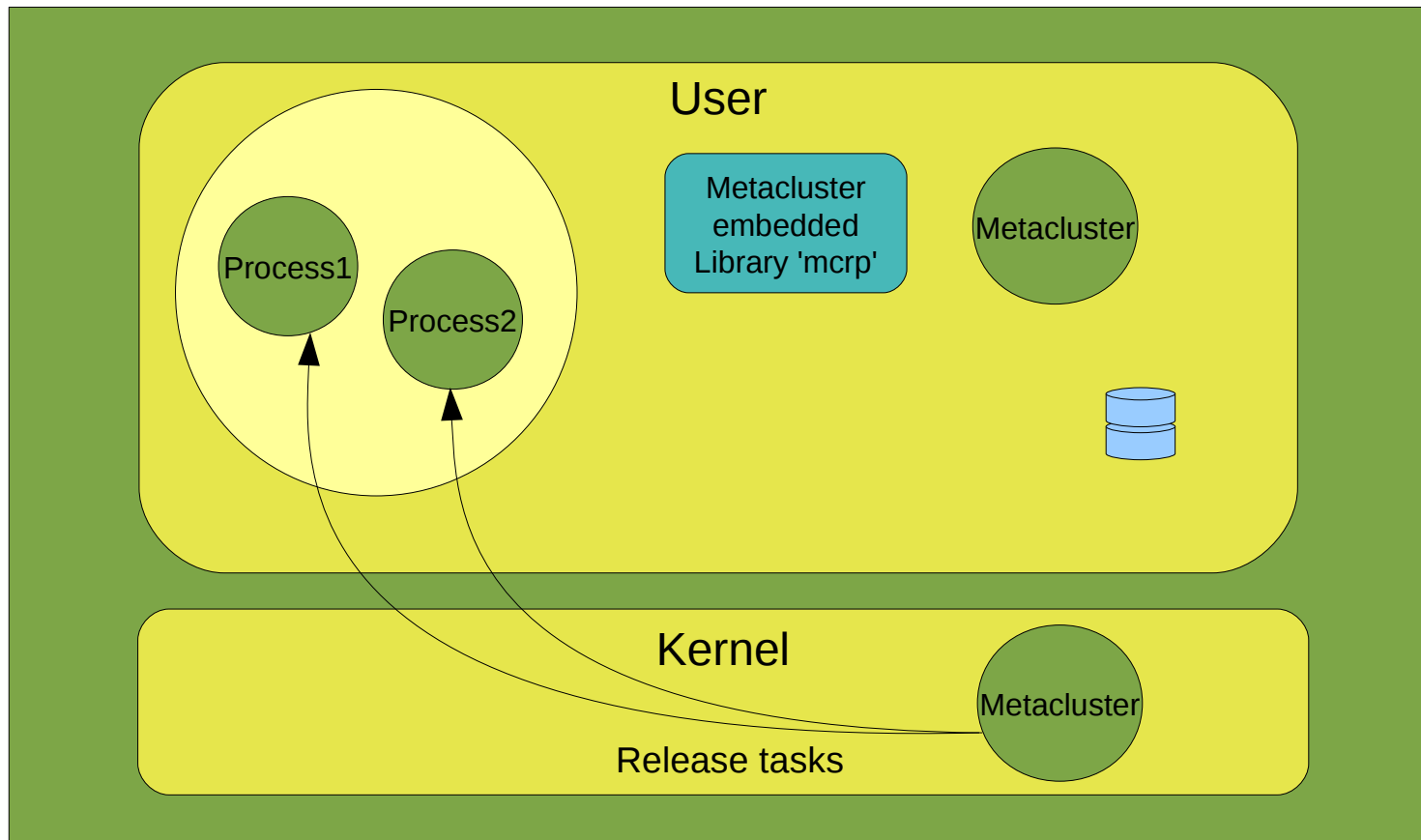
# Checkpoint / Restart

The context is freed



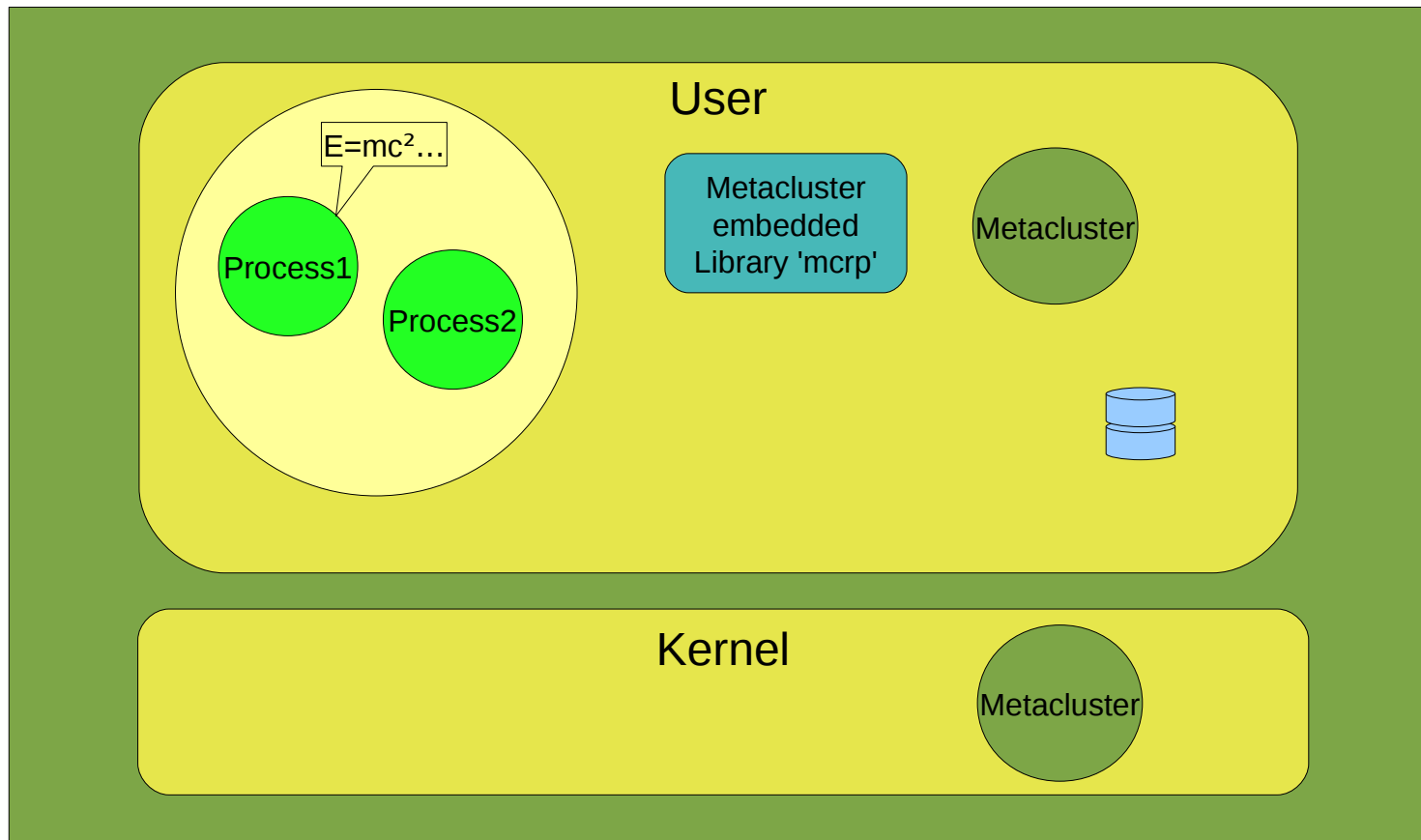
# Checkpoint / Restart

Tasks are released



# Checkpoint / Restart

... and the application continues its execution

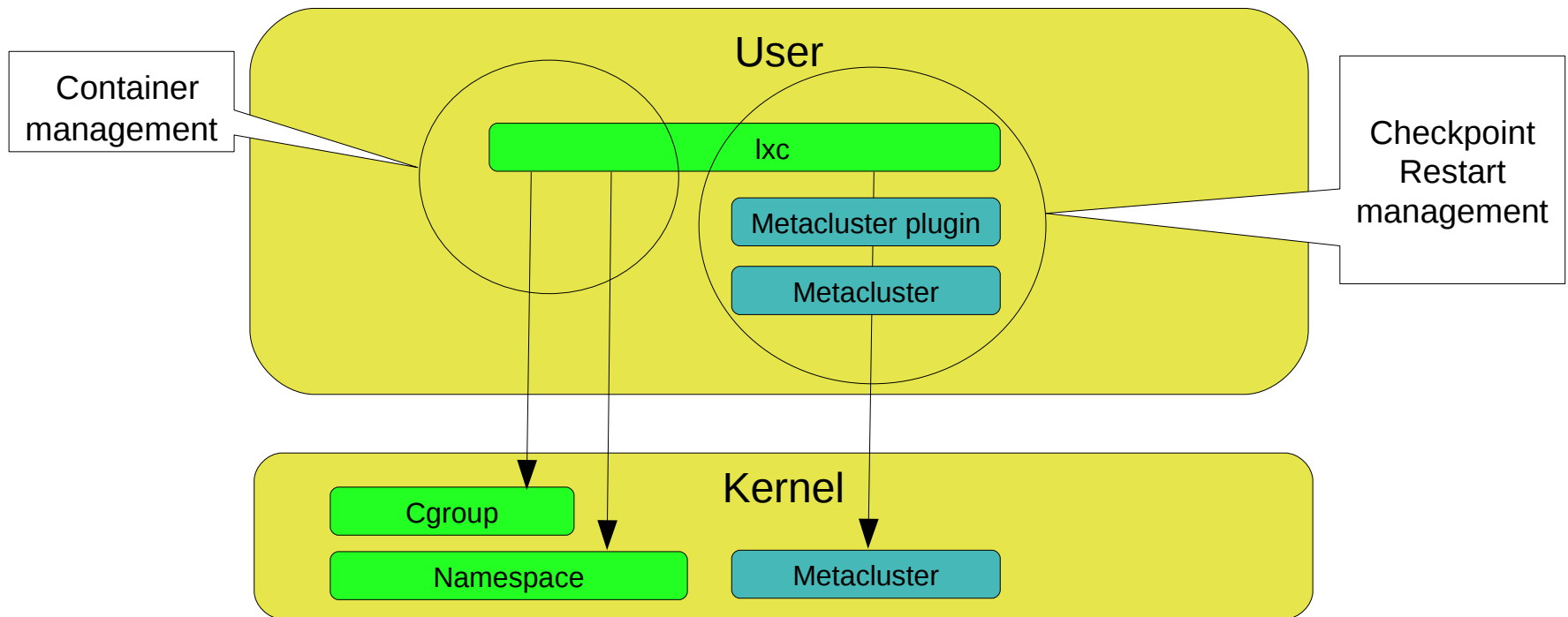


## Checkpoint / Restart

- The restart is symmetric to the checkpoint
- The checkpoint is sequential
- The statefile can be written directly to a pipe or a socket

## Metacluster and lxc

- Lxc accepts checkpoint / restart plugins
- Metacluster is integrated with lxc as plugin



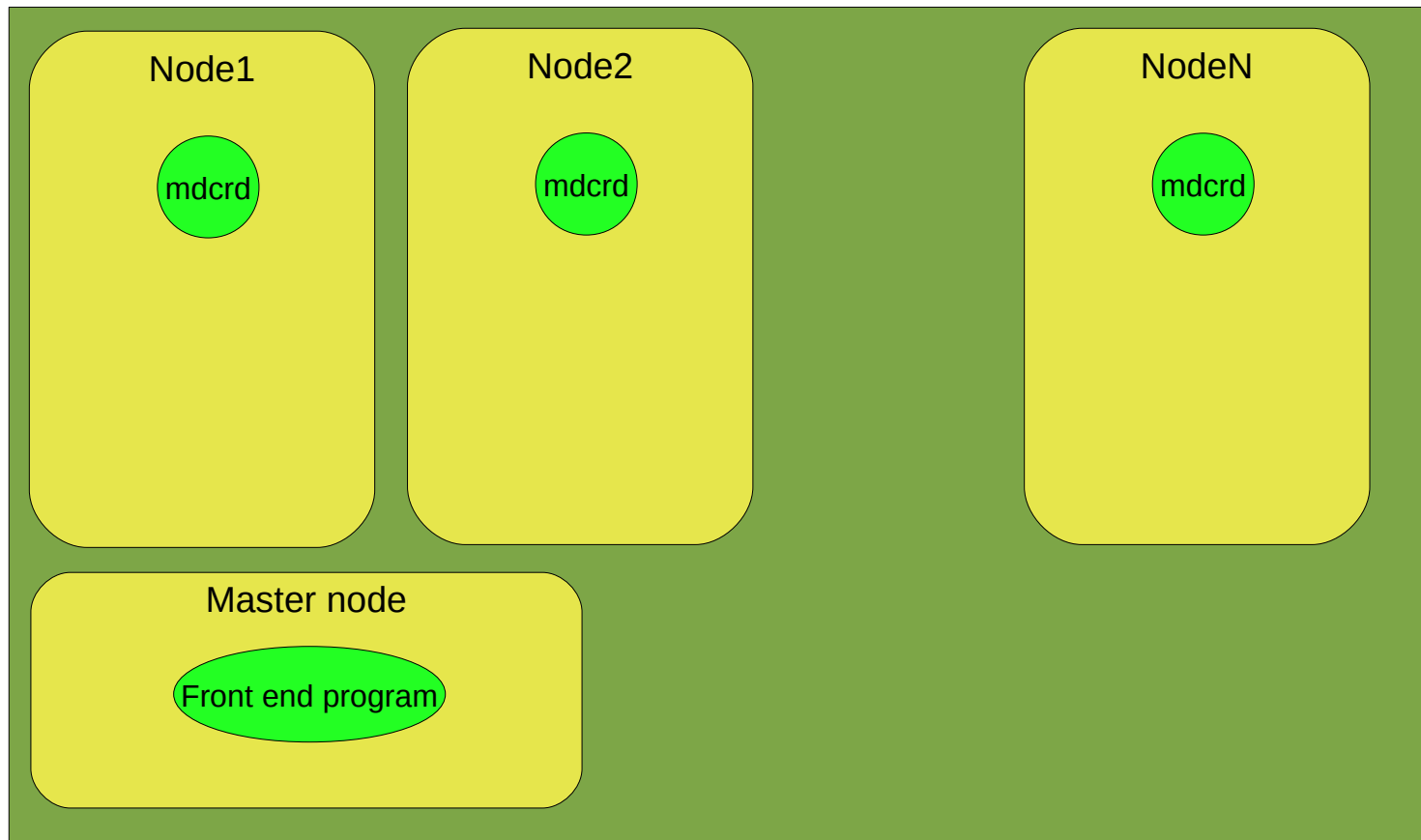
## Distributed Checkpoint / Restart

With the distributed checkpoint / restart, we solve 3 problems:

- Failure : the cluster is periodically checkpointed
- Priority : an application is moved to another node
- Preemption : the cluster or one node of the cluster is paused

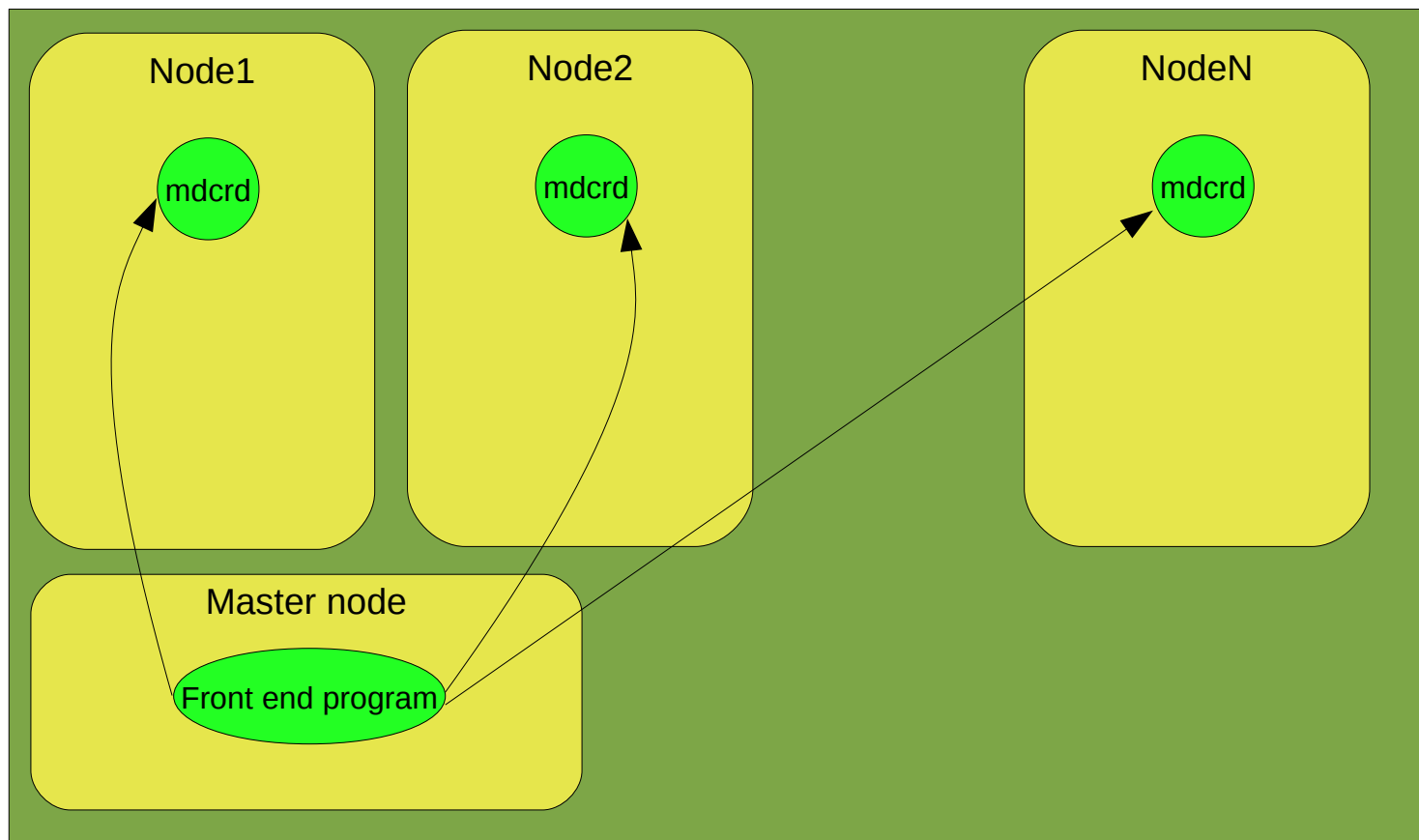
## Distributed Checkpoint / Restart

On each node, there is a specific daemon



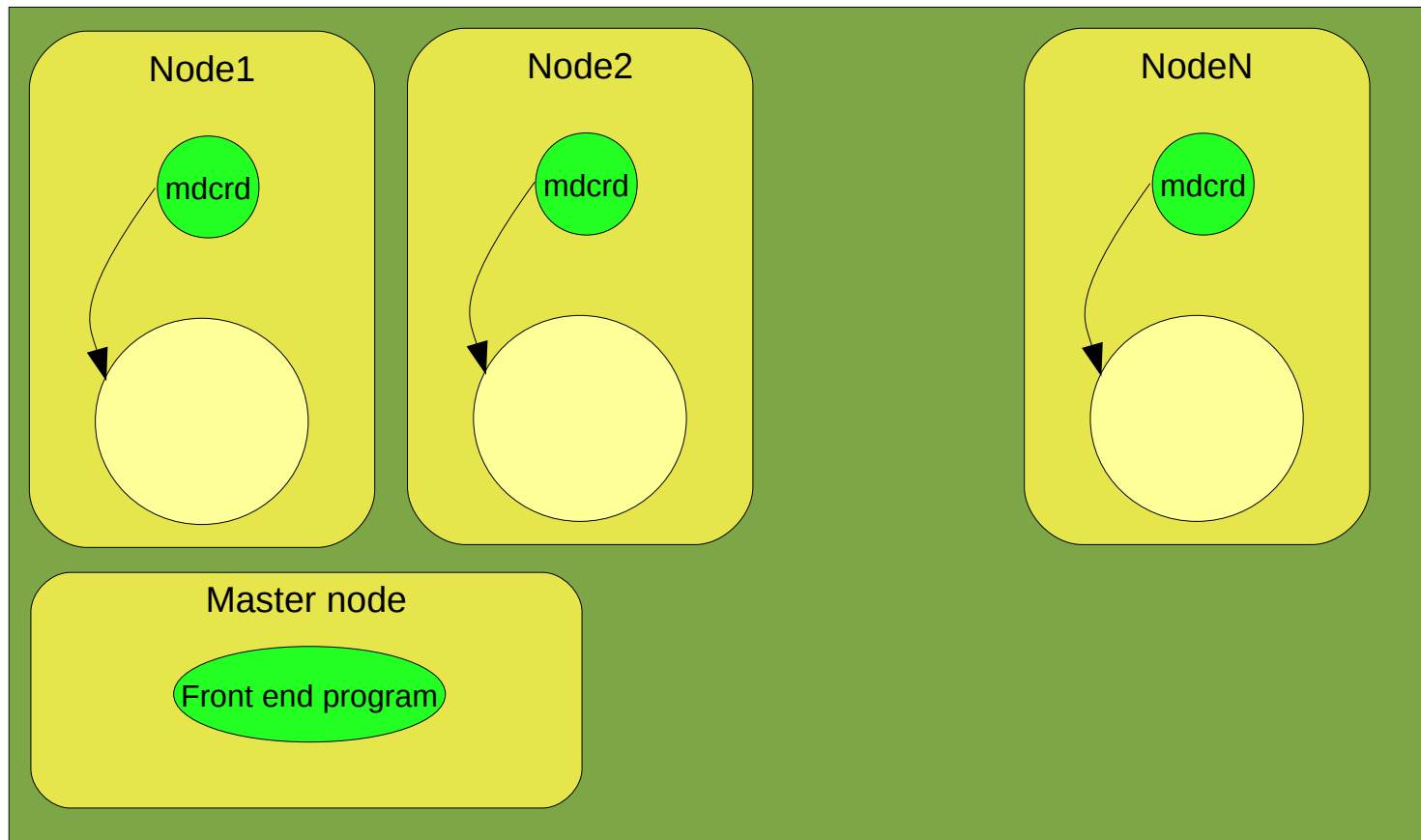
## Distributed Checkpoint / Restart - Executing

The front end application tells to each nodes to execute a specific job



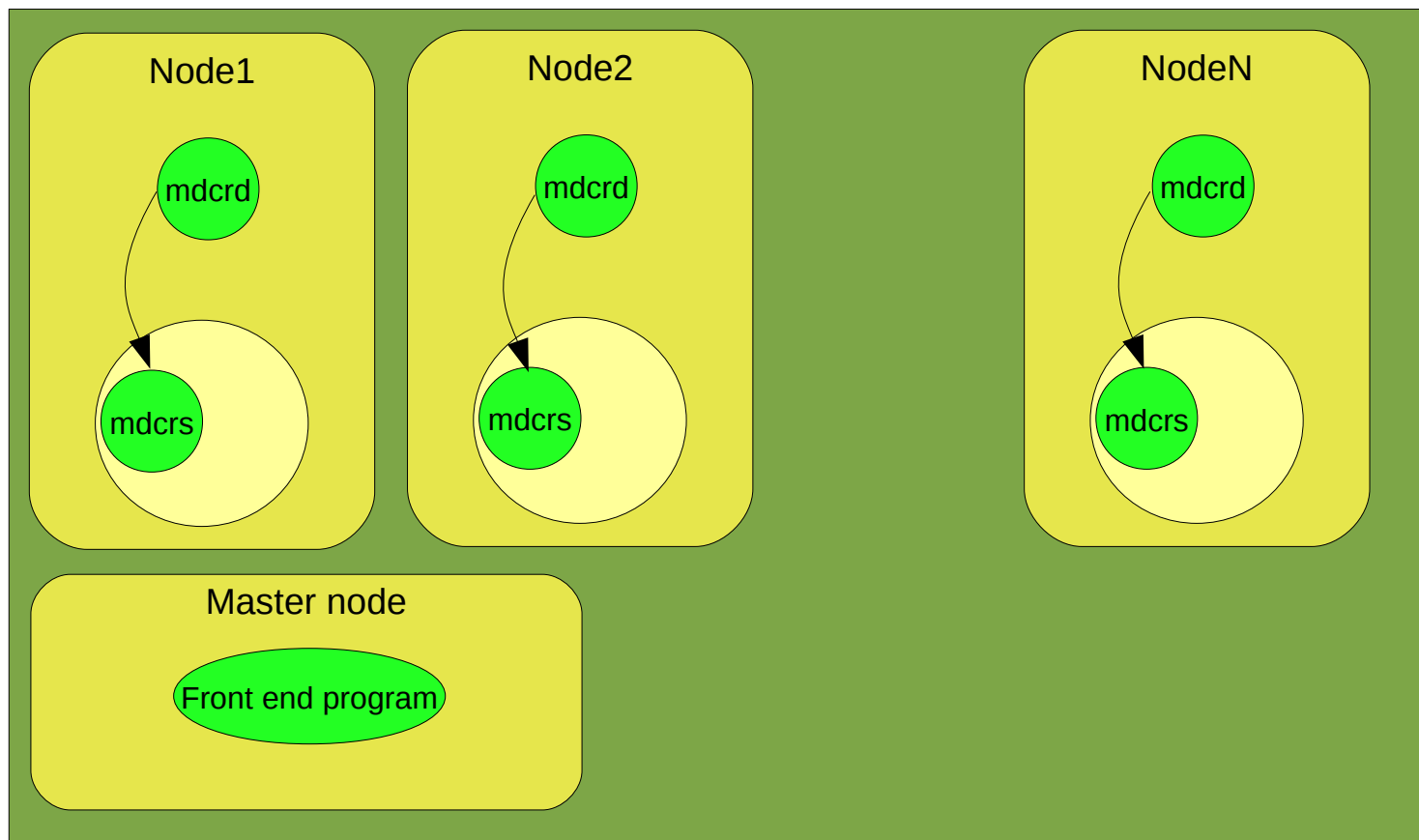
## Distributed Checkpoint / Restart - Executing

The daemons creates a container



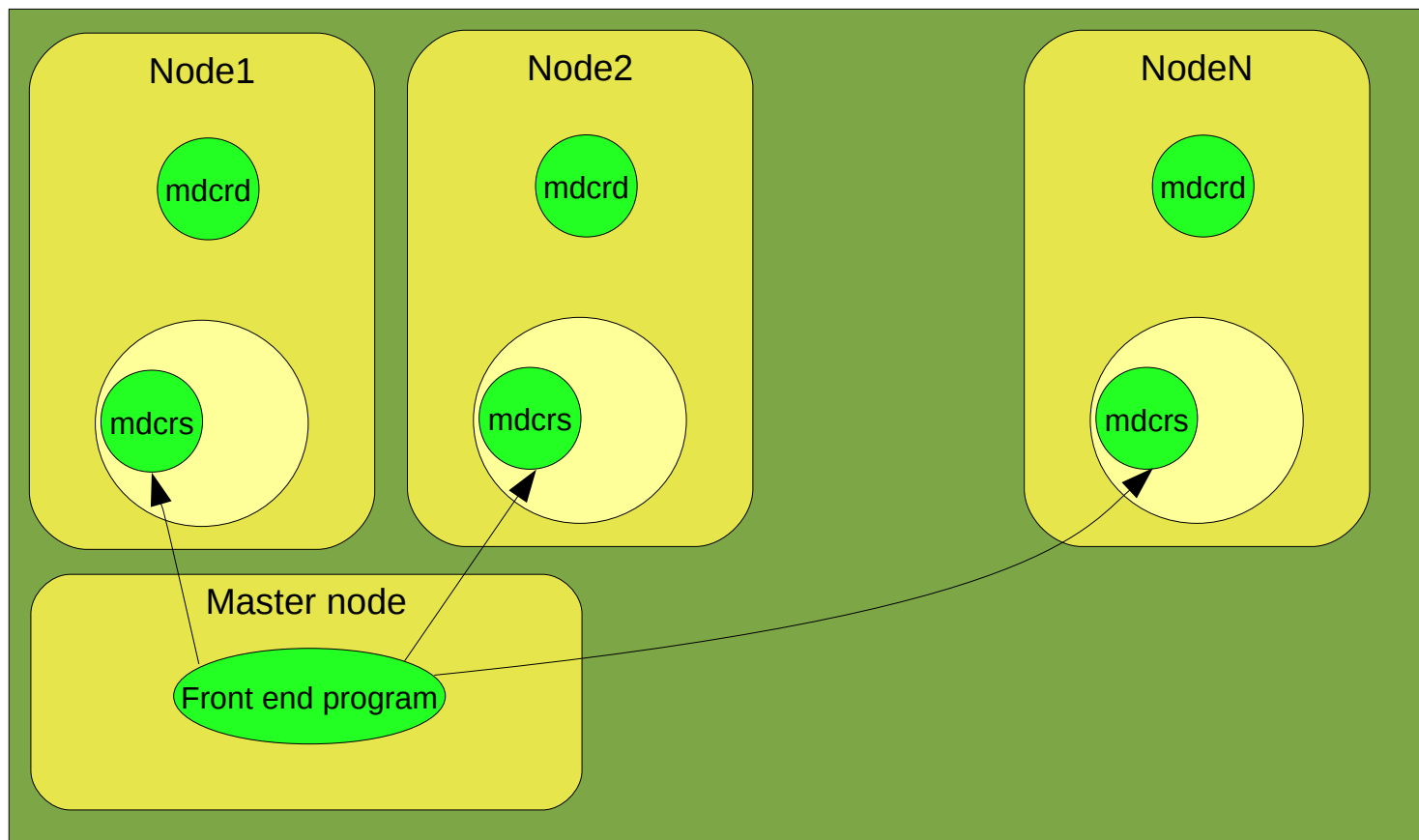
## Distributed Checkpoint / Restart - Executing

And execute a specific supervisor for this job in the container



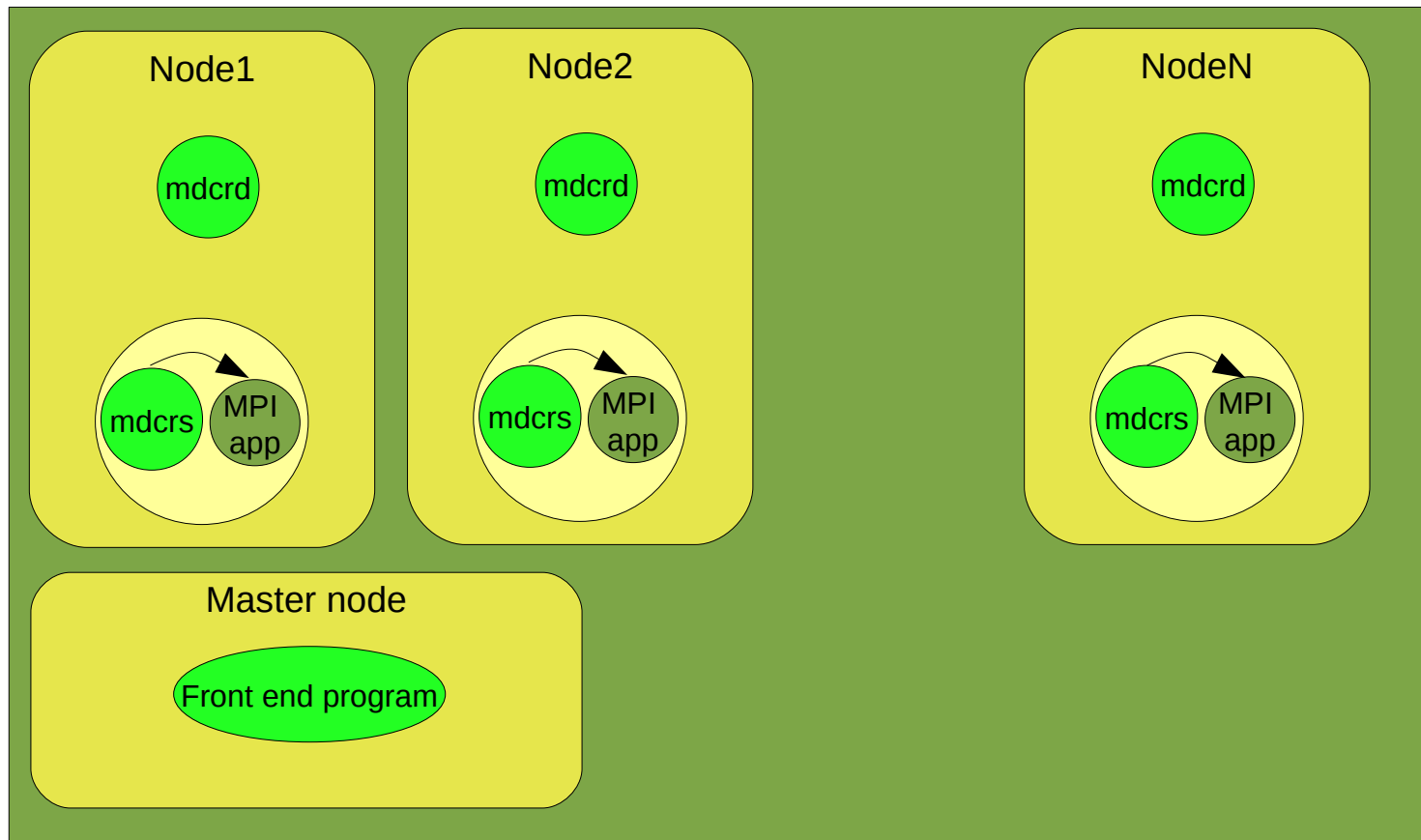
## Distributed Checkpoint / Restart - Executing

The front end application tells the supervisors to launch the job



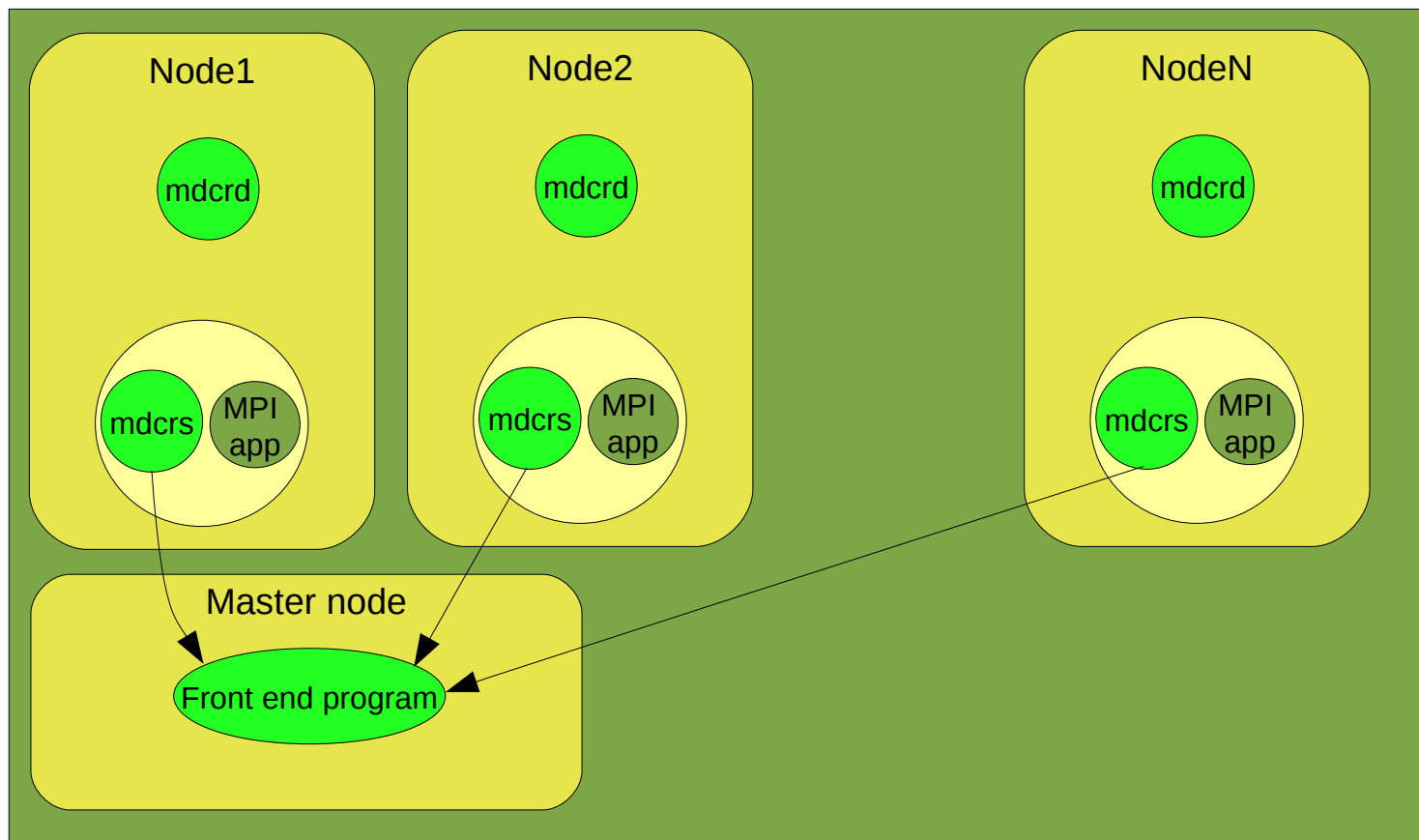
# Distributed Checkpoint / Restart - Executing

The HPC job is launched



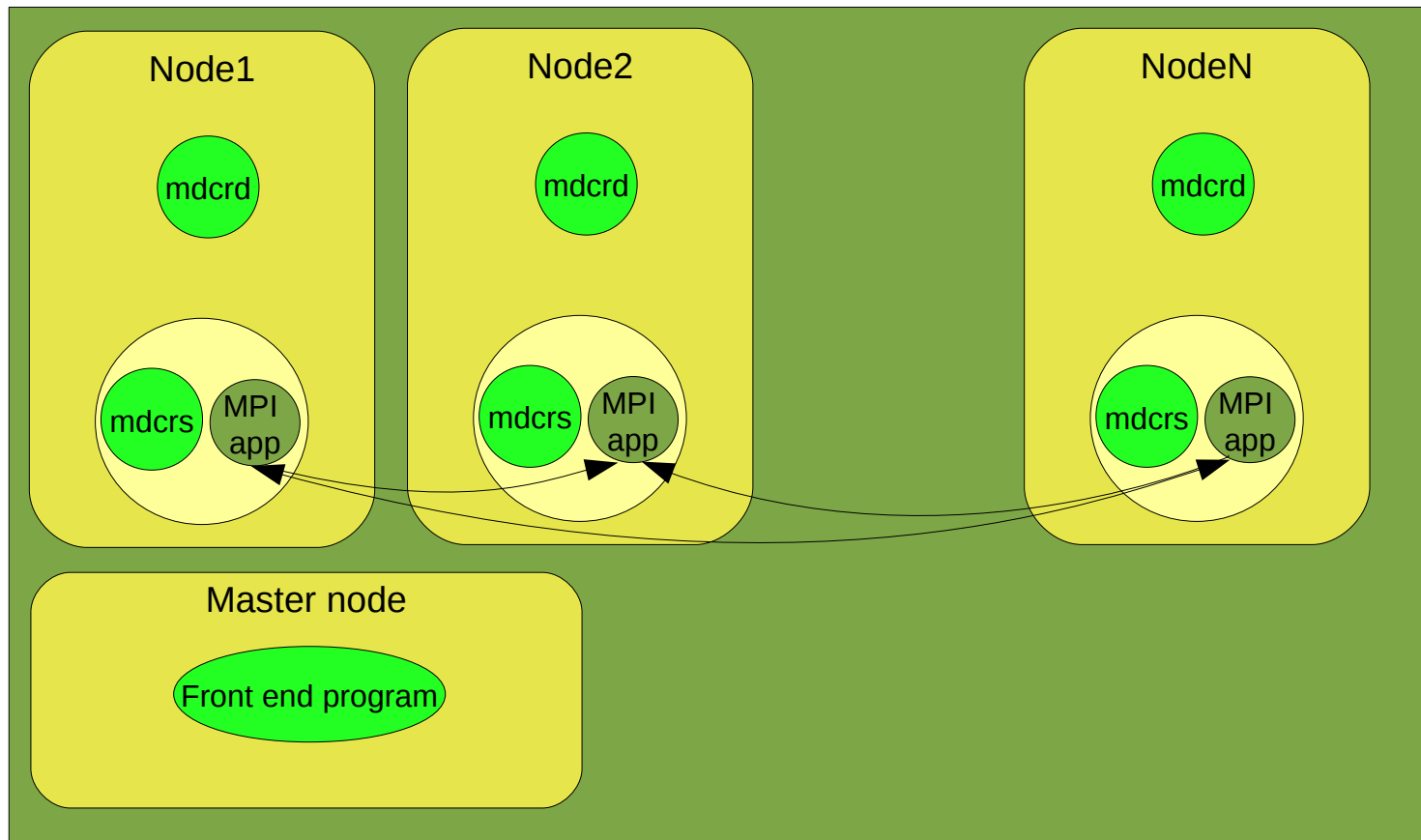
## Distributed Checkpoint / Restart - Executing

And the mdcrs tells the operation was successful



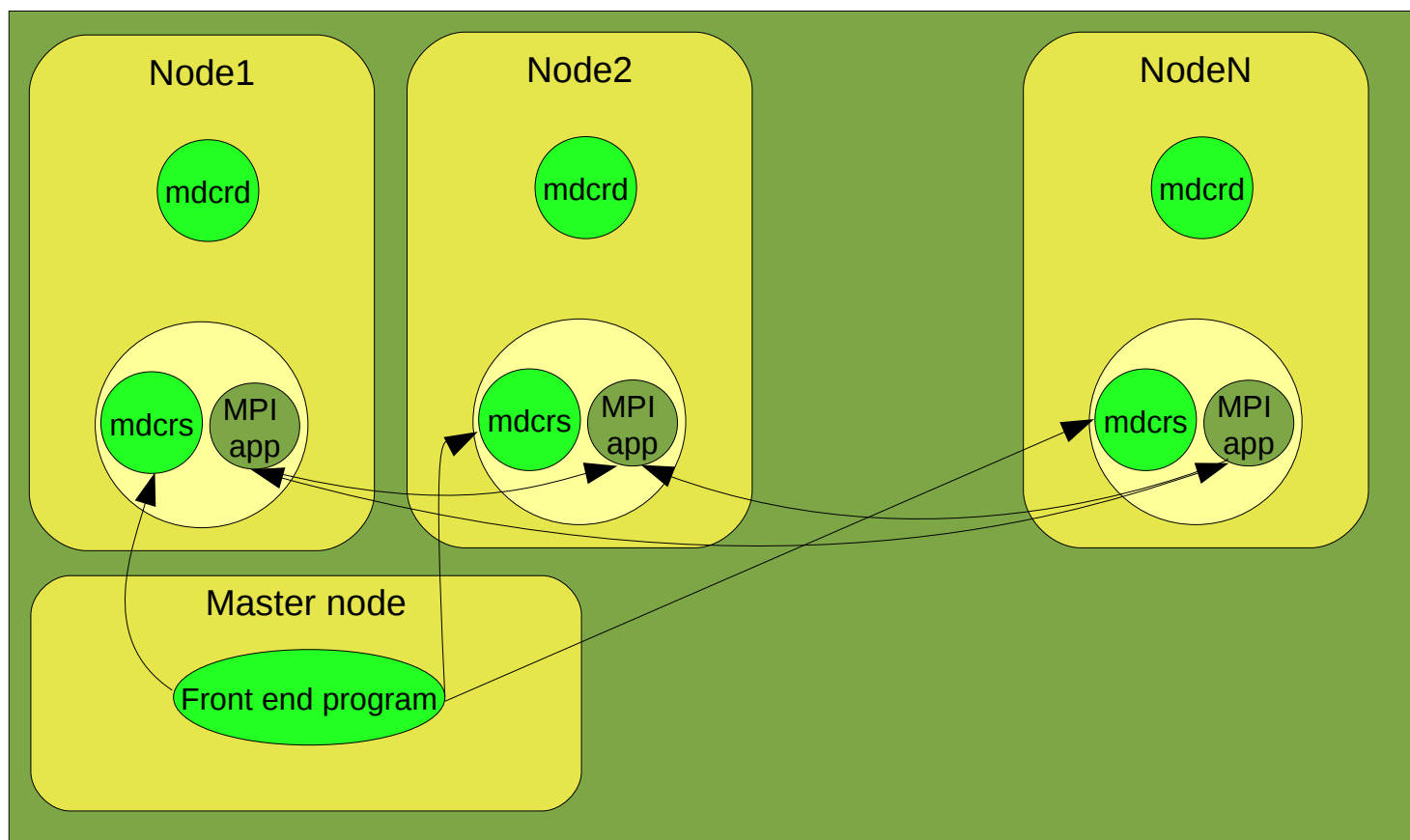
# Distributed Checkpoint / Restart - Executing

The job computation begins



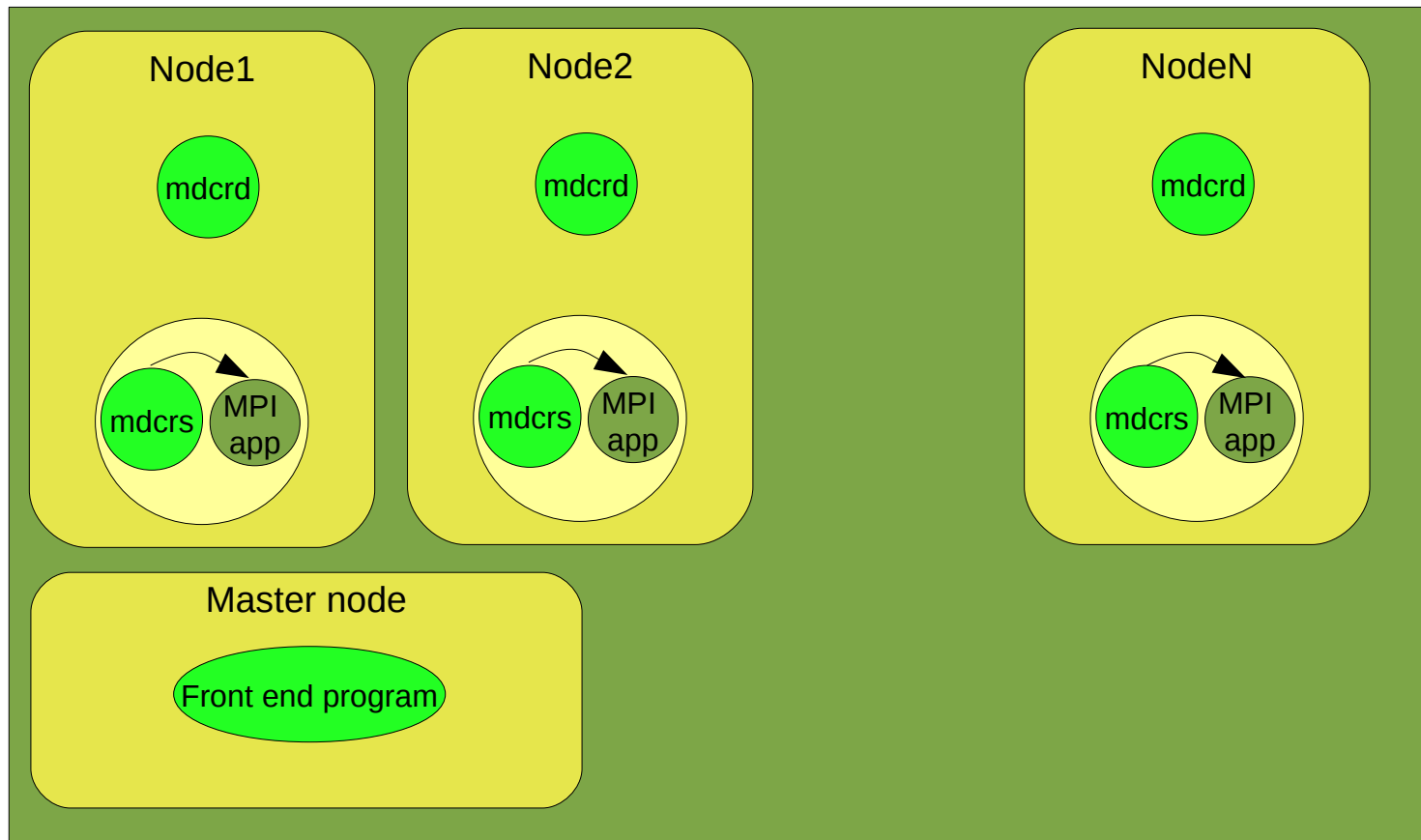
## Distributed Checkpoint / Restart - Checkpointing

We initiate the checkpoint



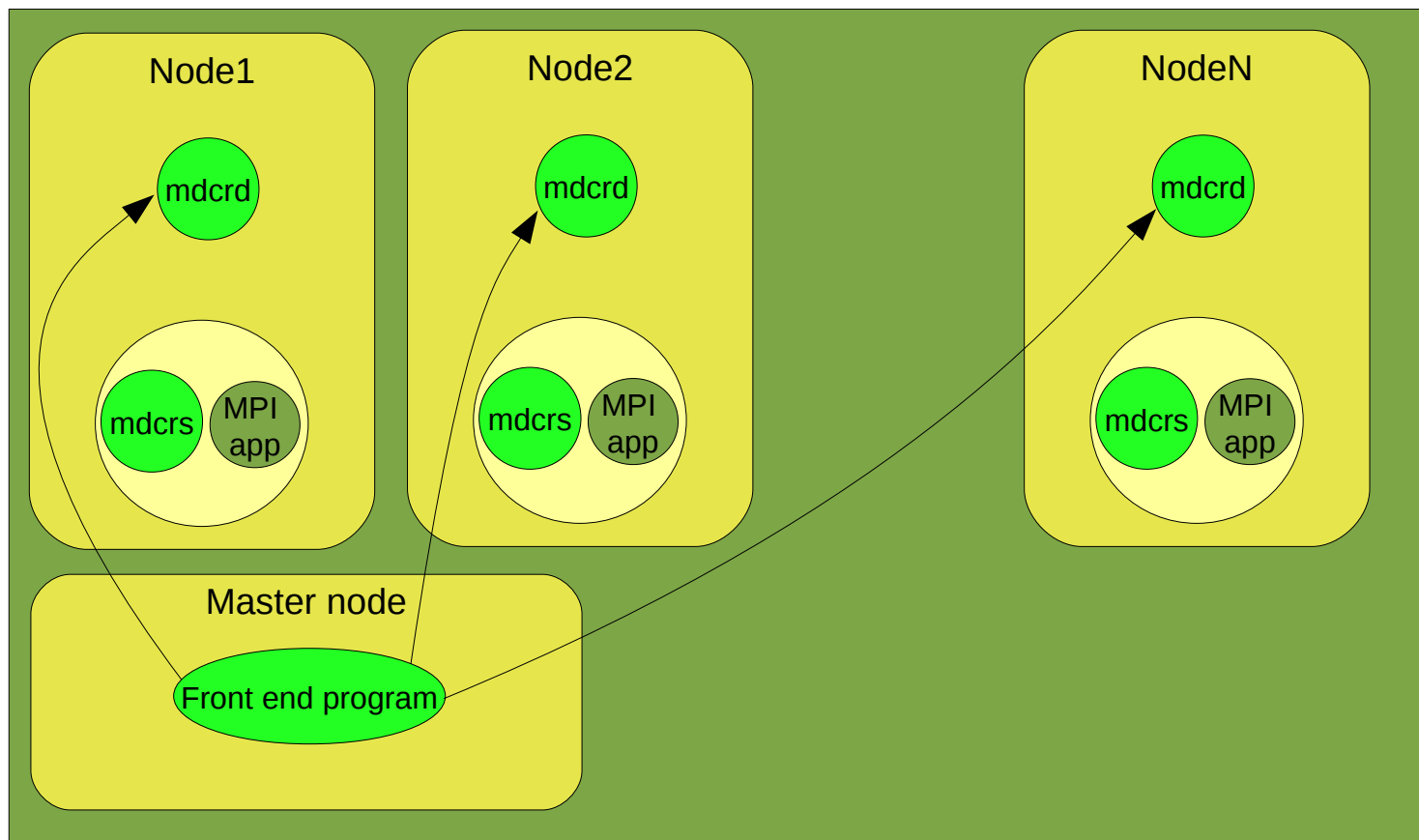
## Distributed Checkpoint / Restart - Checkpointing

The mpi communication is frozen



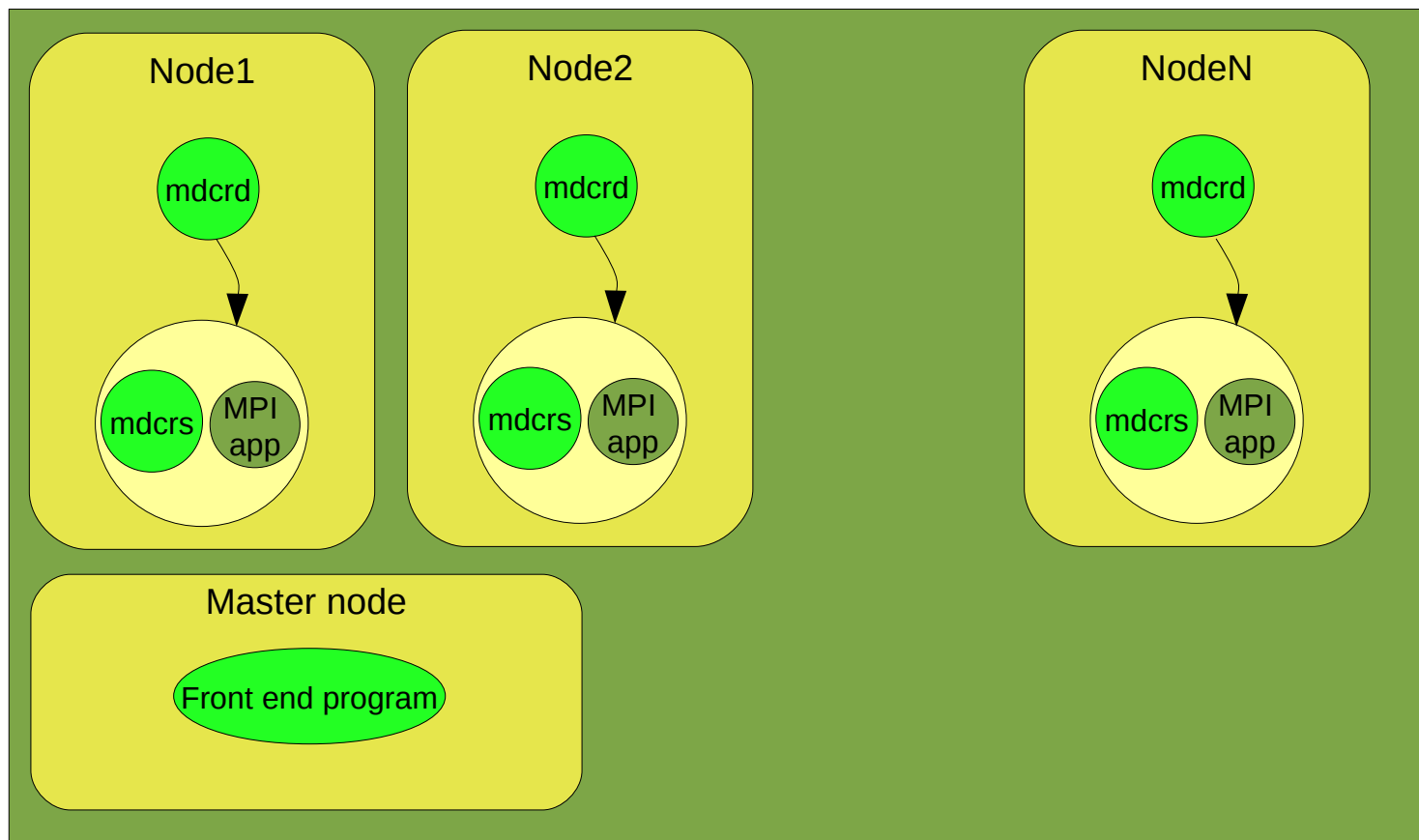
## Distributed Checkpoint / Restart - Checkpointing

Checkpoint is requested on all the daemons of the all nodes



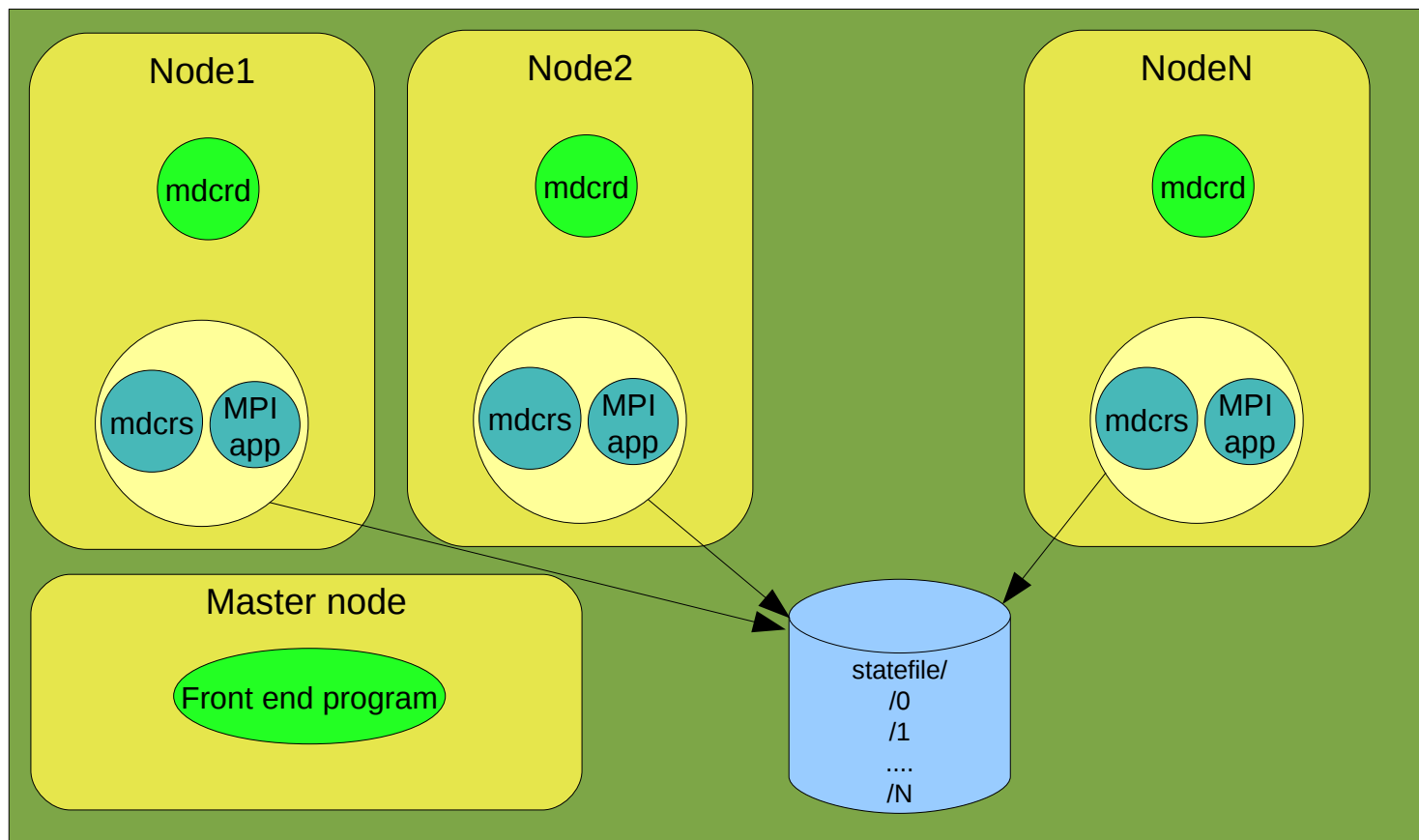
## Distributed Checkpoint / Restart - Checkpointing

The containers on all the nodes are checkpointed



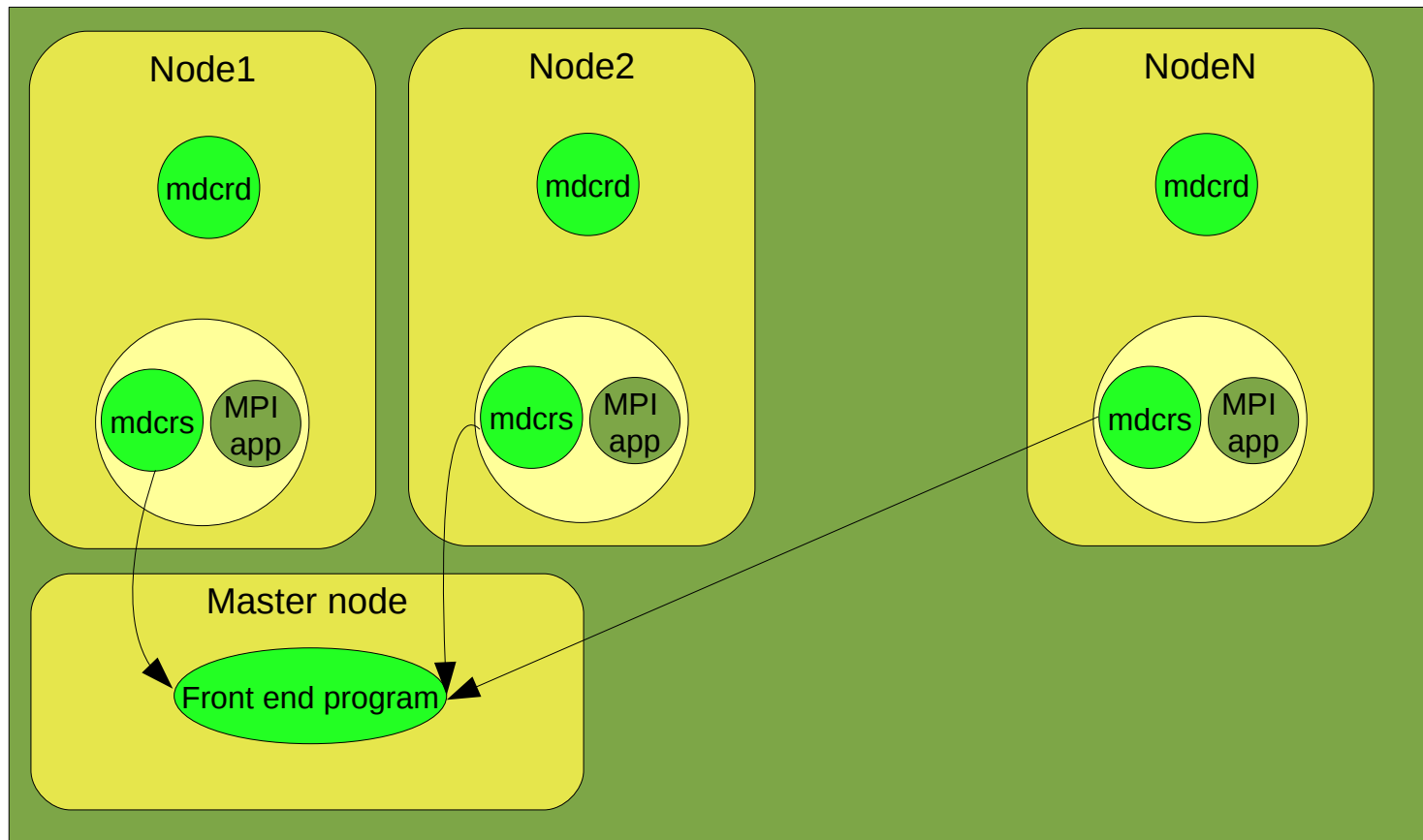
## Distributed Checkpoint / Restart - Checkpointing

The different statefiles are stored in a shared filesystem



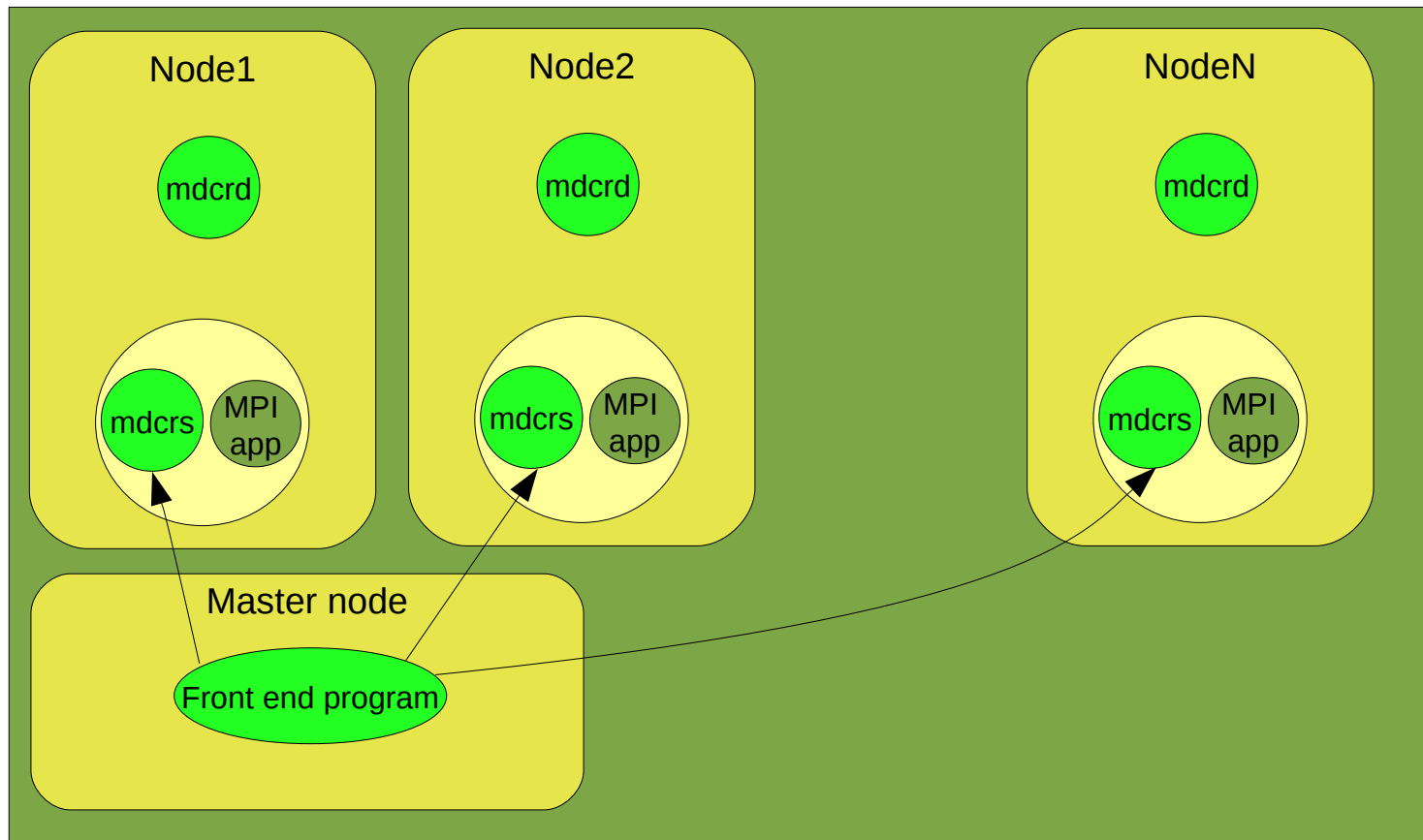
## Distributed Checkpoint / Restart - Checkpointing

The supervisors tell the checkpoint is complete and wait for an acknowledgment



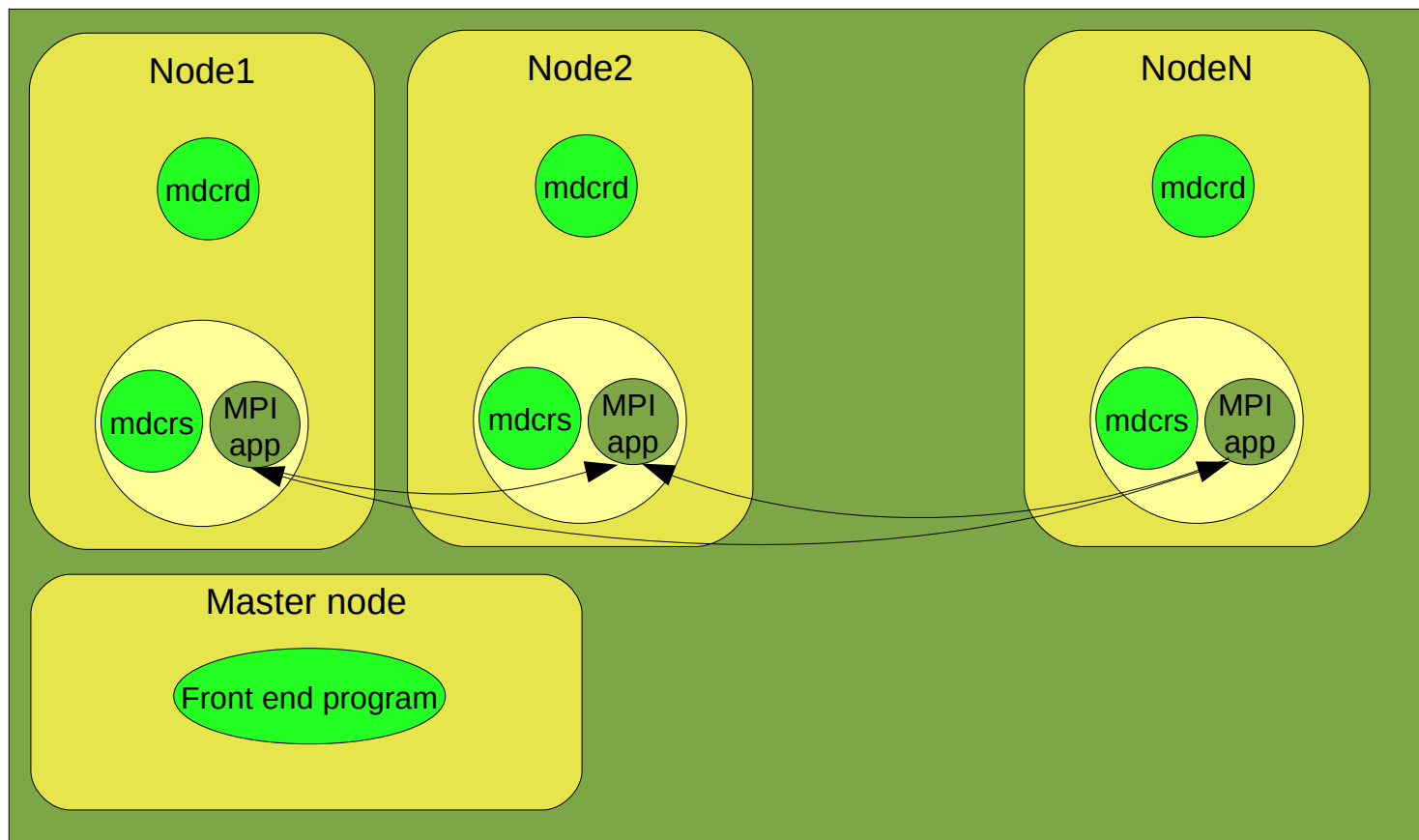
## Distributed Checkpoint / Restart - Checkpointing

The front end application tells the supervisor to enable mpi communication



## Distributed Checkpoint / Restart - Checkpointing

The job computation continues its execution



## Conclusion

- The Containers exists for the mainline kernel
- There is good probability the Checkpoint / Restart will be in mainline someday
- The Distributed Checkpoint / Restart is necessary for wide cluster

## References

- “Fault Tolerant Checkpointing Solution for Cluster and Grid Systems”, HPC4U project
- “HPC4U – System Architecture definition”, HPC4U project
- “Evaluation of Linux native isolation mechanisms for XtremOS flavours”, XtremOS project
- “Making application mobile under Linux”, IBM
- “Virtual Servers and Checkpoint Restart in mainstream Linux”, IBM
- “LXC: Linux container tools”, IBM
- “Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors”, Princeton University