Workload Management with Cpusets

Sam Watters
SGI Engineering

Overview

- The Basics
- Configuration Options
- Features (Command & API)
- Examples
  - Example: Definitions
  - Example: Strictly Managed System
  - Example: Preemption
  - Example: Preemption Details - IRIX® 6.5.13-6.5.15
  - Example: Preemption Details - IRIX® 6.5.16+
- Review: The Newest Features
- Where We Might Be Going...

The Basics

What is a cpuset?

- A named set of CPUs: control process scheduling
- May be defined as *nonexclusive* (open) or *exclusive* (restricted)
  - *Nonexlusive*: threads attached to the cpuset can only run on CPUs assigned to the cpuset; threads not attached to the cpuset also can run on CPUs assigned to the cpuset
  - *Exclusive*: threads attached to the cpuset can only run on CPUs assigned to the cpuset; threads not attached to the cpuset cannot run on CPUs assigned to the cpuset
- Provides features that control memory management

The Basics

Why use a cpuset?

- Improve memory locality for applications
- Restrict consumption of CPU and memory resources to specified processes/threads
- Enhance your workload manager
  - Used by workload management (batch) systems (GRD, LSF, PBS)
  - Advanced scheduling extensions for batch systems (FNMOC)
- Limit run-time variability
- Reduce interference between jobs

The Basics

How are cpusets used?

- Static cpusets
  - Cpusets are defined by administrator after system startup
  - Users attach processes to the existing cpusets
  - Cpusets continue to exist after jobs finish executing
- Dynamic Cpusets
  - Workload management system creates cpuset when it is required by a job
  - Workload management system attaches job to the newly created cpuset
  - Workload manager destroys cpuset at end of job

The Basics

How are cpusets used? (cont)

- Boot cpuset
  - Only one boot cpuset can exist on system
  - Created during startup of *init* process
  - *Init* process attaches itself to this cpuset
  - Cpuset will always be named *boot*
  - Requires special configuration files and DSO library file
  - *Init* process and all descendents will be attached (contained) within the boot cpuset
  - Need to attach processes to static or dynamic cpusets to get out of the boot cpuset

The Basics

Requirements for cpuset

- Permissions file
  - A file name must be provided at cpuset creation
  - File is used to determine access permission for cpuset via the normal UNIX® file permissions
    - Read permission: user, group, or world can read information from cpuset
    - Write permission: user, group, or world can attach processes to the cpuset
  - Cpuset created using command line tool: file is also used to provide configuration information
  - Cpuset created using API: file is only used for permissions

The Basics.

Requirements for cpusets (cont.)

- Permissions file (cont.)
  - Permissions for file can be changed during existence of cpuset to alter access permissions for the cpuset
  - Permissions file can be deleted after creation, but then permissions cannot be changed
  - Boot cpuset permission/configuration file
    - /etc/config/boot_cpuset.config
- Name
  - Every cpuset must be given a unique name
    - Name must consist of 2-8 alphanumeric characters

The Basics

Restrictions on CPUs within cpusets

- CPU can only belong to one cpuset
- CPU 0 cannot belong to an exclusive cpuset
- CPU cannot be both restricted or isolated and a member of a cpuset
  - See *mpadmin(1)* and *sysmp(1)* concerning restricted or isolated CPUs
- Only superuser can create or destroy cpusets
- *Runon(1)* can only use CPU in cpuset if user also has write or group write access permission

The Basics

Things you need to know

- Memory Locality Domain (MLD)
  - IRIX® attempts to allocate pages on node where MLD is placed
- Global cpuset (global_cpuset)
  - The CPUs not assigned to a cpuset or otherwise restricted
  - All systems have a global_cpuset
- Nodes and cpusets
  - A node is within a cpuset if a CPU resident to that node belongs to the cpuset

Configuration Options

Cpuset Configuration File Options

(API options in parentheses)

- **EXCLUSIVE** (CPUSET_CPU_EXCLUSIVE)
  - Defines the CPUs in the cpuset to be restricted
  - If not defined, cpuset is nonexclusive, or open
- **CPU** (program provides array of CPU ID values)
  - Defines that a CPU of set of CPUs will be part of the cpuset
  - Format: CPU 4 or CPU 4,9-16,24-31,47
  - CPU numbering always begins with CPU 0
  - CPU 0 cannot be in an EXCLUSIVE cpuset

Configuration Options

- MEMORY_LOCAL (CPUSET_MEMORY_LOCAL)
  - Threads attached to cpuset:
    - MLDs can only placed on nodes with CPUs in cpuset
    - IRIX® attempts to allocate pages on nodes where MLDs are placed
  - Threads not attached to cpuset
    - Imposes no added restriction on MLD placement
- MEMORY_EXCLUSIVE (CPUSET_MEMORY_EXCLUSIVE)
  - Threads attached to cpuset:
    - Imposes no added restriction on MLD placement
  - Threads not attached to cpuset
    - MLDs cannot be placed on nodes with CPUs in cpuset

Configuration Options

- MEMORY_MANDATORY (CPUSET_MEMORY_MANDATORY)
  - Implies that MEMORY_LOCAL and MEMORY_EXCLUSIVE are set
  - Threads attached to cpuset
    - MLDs can only placed on nodes with CPUs in cpuset
  - Threads not attached to cpuset
    - MLDs cannot be placed on nodes with CPUs in cpuset
  - If memory requests cannot be satisfied, allocating process will sleep until memory becomes available
  - Process will be killed if no more memory can be allocated
  - POLICY_* options will further affect behavior

Configuration Options

- POLICY_PAGE (CPUSET_POLICY_PAGE)
  - Default policy if no policy is specified
  - IRIX® will page user pages to swap file to free physical memory on nodes
  - If swap is exhausted, process will be killed
- POLICY_KILL (CPUSET_POLICY_KILL)
  - IRIX will free as much space as possible from kernel heaps on nodes
  - No attempt made to page user pages to swap file
  - Process will be killed if no more memory can be allocated

Configuration Options

- MEMORY_KERNEL_AVOID
  (CPUSET_KERNEL_MEMORY_AVOID)
  - Only prevents system buffer cache from being placed on nodes with CPUs in cpuset
  - WARNING: only effective for certain workload patterns and will result in severe performance penalties

Features

Cpuset Creation

- Command Line
  - cpuset -q *qname* -c -f *filename*
- API
  - cpusetCreate(char **qname*, cpuset_QueueDef_t **qdef*)
- Descriptions
  - *qname* is the name of the cpuset (queue)
  - *filename* is the name of the permissions/config file
  - *qdef* provides the configuration information and name of permissions file to the API

Features

Command Line Example

- cpuset -q myqueue -c -f /tmp/myqueue.cpuset
- myqueue.cpuset

```
EXCLUSIVE
MEMORY_LOCAL
MEMORY_EXCLUSIVE

CPU 8-11
```

Features

API Example - data structures

```
/* cpuset queue definition structure */
typedef struct {
        int                 flags;    /* CPU & memory options */
        char                *permfile;/* permission file name */
        cpuset_CPUList_t *cpu;     /* ref to list of CPUs */
} cpuset_QueueDef_t;

/* cpuset CPU list structure */
typedef struct {
        int      count;    /* number of CPUs in list */
        int      *list; /* list of CPUs */
} cpuset_CPUList_t
```

Features

API Example - programming

```
char                        *permfile = “/tmp/myqueue.cpuset”
int                         cpuids[4] = {8,9,10,11};
cpuset_CPUList_t     cpu = {4, cpuids}
cpuset_QueueDef_t  qdef = {0, permfile, &cpu}
char                        *qname = “myqueue”;

qdef->flags = CPUSET_CPU_EXCLUSIVE | CPUSET_MEMORY_LOCAL
                     | CPUSET_MEMORY_EXCLUSIVE;
if (!cpusetCreate(qname, &qdef)) {
          perror(“cpusetCreate”);
          exit(1);
}
```

Features

Attaching process to cpusets

- Command Line
  - cpuset -q *qname* -A *command*
    - Run *command* in cpuset named *qname*
- API
  - cpusetAttach(char **qname*)
    - Attaches current process to cpuset named *qname*
  - cpusetAttachPID(char **qname*, pid_t *pid*)
    - Attaches process identified by *pid* to cpuset name *qname*

Features

Command

- **cpuset [-q cpuset_name [-A command] | [-c -f filename] | [-d] | [-l] | [-m] | [-Q] | [-p]] | -C | -Q | -h**
- **-q cpuset_name -A command**
  - Runs the command on the cpuset identified by cpuset_name
- **-q cpuset_name -c -f filename**
  - Creates the cpuset cpuset_name using filename as the configuration/permissions file
- **-q cpuset_name -l**
  - List all processes attached to the cpuset
- **-q cpuset_name -m**
  - Move all attached processes out of cpuset
- **-q cpuset_name -d**
  - Destroy the cpuset (cannot have any processes attached)

Features

Command

- **-q cpuset-name -Q**
  - List all CPUs in the cpuset
- **-q cpuset_name -p**
  - List all permissions: ACLs, MAC labels, flags, CPUs and number of processes for the cpuset
- **-C**
  - List the name of the cpuset to which the current process is attached
- **-Q**
  - List the names of all existing cpusets
- **-h**
  - Print command usage

Features

Command

New in 6.5.17

- **-q from_cpuset,to_cpuset -M idtype -i id**
  - Move processes identified by *id,* and migrate the memory they own, from current cpuset *(from_cpuset)* to destination cpuset *(to_cpuset).* The *idtype* specified can be either ASH, JID, PGID, PID, or SID.
- **-q from_cpuset,to_cpuset -T idtype -i id**
  - Move processes identified by *id*, from current cpuset *(from_cpuset)* to destination cpuset *(to_cpuset).* No memory is migrated The *idtype* specified can be either ASH, JID, PGID, PID or SID.

Features

API (Management Functions)

- int cpusetCreate(char *qname, cpuset_QueueDef_t *qdef)
  - Create a cpuset
- int cpusetAttach(char *qname)
  - Attach current process to cpuset
- int cpusetAttachPID(char *qname, pid_t pid)
  - Attach specified process to cpuset
- int cpusetDetachPID(char *qname, pid_t pid)
  - Detach specified process from cpuset
- int cpusetDetachAll(char *qname)
  - Detach all processes from cpuset
- int cpusetDestroy(char *qname)
  - Destroy (remove) the cpuset

Features

API (Management Functions)

- ***New in 6.5.16***
  - cpusetMove
    - Moves the process or group of processes between cpusets or into the global_cpuset
    - Memory (MLDs and pages) does not migrate
  - cpusetMoveMigrate
    - Moves the process or group of processes between cpusets or into the global_cpuset
    - Memory (MLDs and pages) is migrated
  - Two-step move allows reduction of memory migrations when you need to change system state before final placement

Features

API (Management Functions)

- ***New in 6.5.16***
  - **int cpusetMove(char *from_qname, char *to_qname, int type, uint64_t id)**
    - Move processes from one cpuset to another without moving memory
    - The *type* indicates the *id* value is an ASH, jid, pid, pgid, sid
  - **int cpusetMoveMigrate(char *from_qname, char *to_qname, int type, unit64_t id)**
    - Move processes from one cpuset to another and also move MLDs and pages (memory)
    - The *type* indicates the *id* value is an ASH, jid, pid, pgid, sid

Features

API (Info Functions)

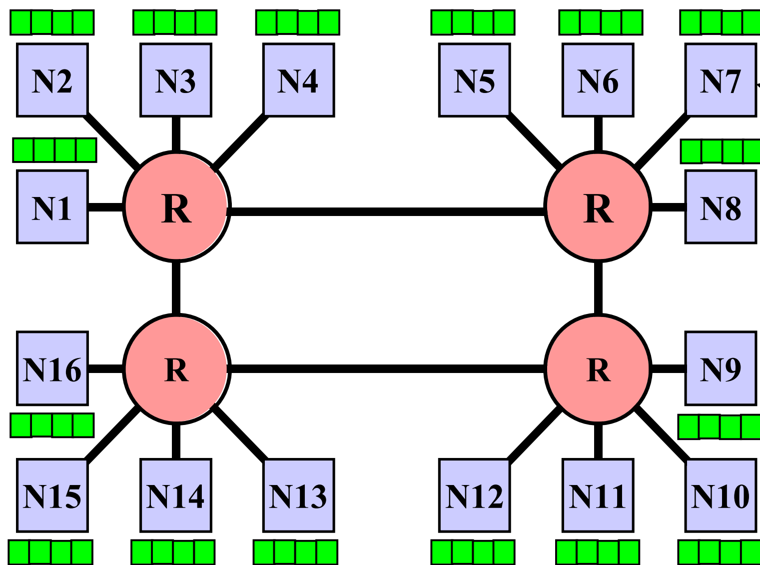
- int cpusetGetCPUCount(void)
  - Get number of CPUs configured on the system
- cpuset_CPU_list _t *cpusetGetCPUList(char *qname)
  - Get list of CPUs contained in a cpuset
- cpuset_NameList_t *cpusetGetName(pid_t pid)
  - Get name of cpuset the process, *pid*, is attached to
- cpuset_NameList_t *cpusetGetNameList(void)
  - Get names of all existing cpusets
- cpuset_PIDList_t *cpusetGetPIDList(char *qname)
  - Get list of processes PIDs attached to the cpuset
- cpuset_Properties_t *cpusetProperties(char *qname)
  - Get list of properties for cpuset

Features

API (Memory Mgmt Functions)

- **cpuset_QueueDef_t *cpusetAllocQueueDef(int count)**
  - Allocate a queue definition with room form *count* CPUs
- **void cpusetFreeQueueDef(cpuset_QueueDef_t *qdef)**
  - Free memory allocated for the referenced queue definition
- **void cpusetFreeCPUList(cpuset_CPUList_t *cpu)**
  - Free memory allocated for the referenced CPU list
- **void cpusetFreeNameList(cpuset_NameList_t *name)**
  - Free memory allocated for the referenced name list
- **void cpusetFreePIDList(cpuset_PIDList_t *pid)**
  - Free memory allocated for the referenced PID list
- **void cpusetFreeProperties(cpuset_Properties_t *csp)**
  - Free memory allocated for the referenced cpuset properties list

Example: Definitions

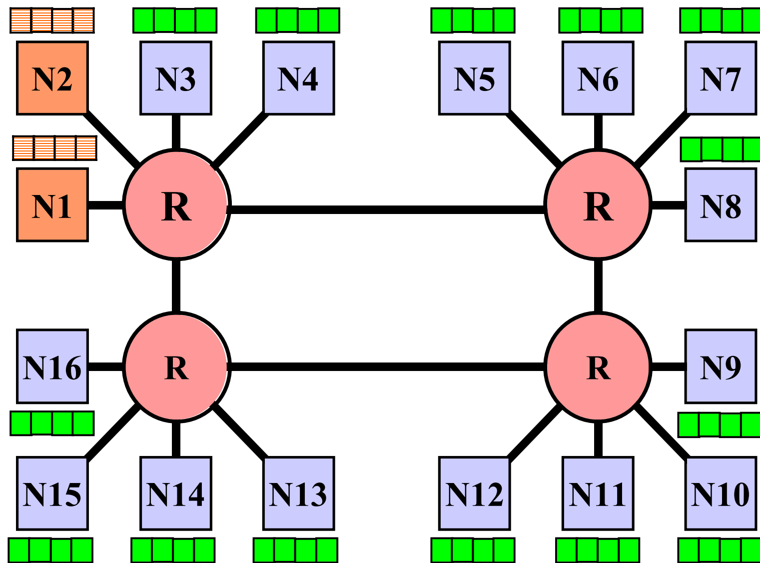


CPU numbering for each node is:

- CPU a = (N x 4) - 4
- CPU b = (N x 4) - 3
- CPU c = (N x 4) - 2
- CPU d = (N x 4) - 1

- CPUs (4 on each node)
- SGI® 3000 family node (4 CPUs and local memory)
- 16 node system with 64 CPUs
- CPUs are numbered from 0 to 63 (this is how CPUs are normally identified)
- On each node, the CPUs are identified as CPU a, b, c and d (you might see CPUs identified this way in the /hw filesystem)

Example: Strictly Managed System



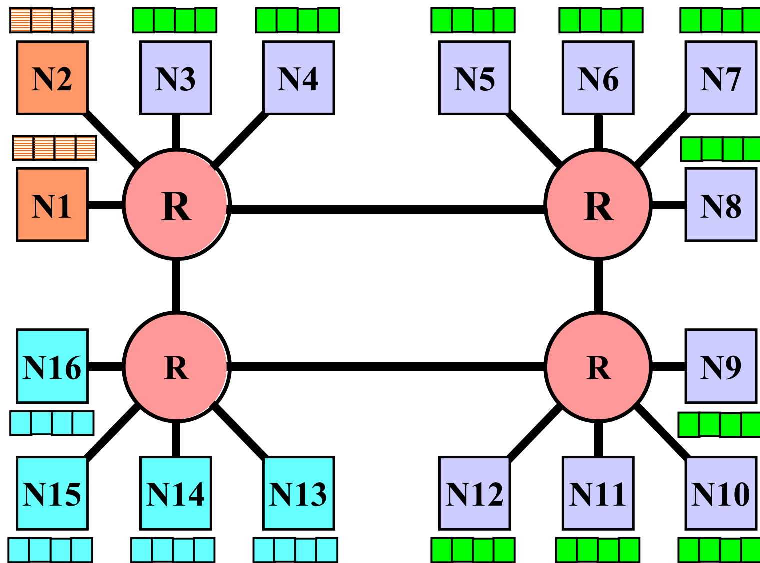
- /etc/config/boot_cpuset.config

```
MEMORY_MANDATORY
POLICY_PAGE

CPU 0-7
```

- The *boot* cpuset created during system initialization
- Created by *init* process
- The init process attaches itself to *boot* cpuset
- All threads will run in *boot* cpuset unless attached to a different cpuset
- All CPUs outside *boot* cpuset reserved for dedicated processing

Example: Strictly Managed System



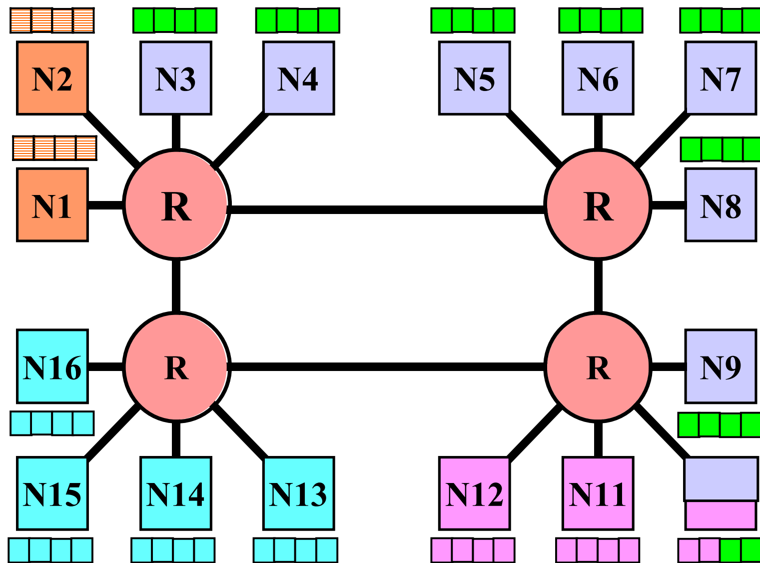
- /tmp/cyan.cpuset

```
EXCLUSIVE
MEMORY_LOCAL
MEMORY_EXCLUSIVE

CPU 48-63
```

- Cpuset boot defined
  - MEMORY_EXCLUSIVE | POLICY_PAGE

Example: Strictly Managed System



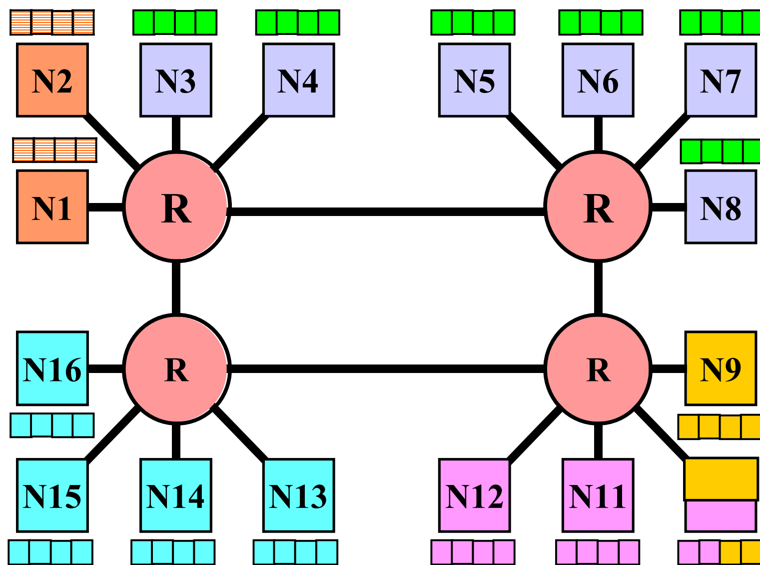
- /tmp/purple.cpuset

```
EXCLUSIVE
MEMORY_MANDATORY
POLICY_PAGE

CPU 38-47
```

- Cpuset boot defined
  - MEMORY_EXCLUSIVE | POLICY_PAGE
- Cpuset cyan defined
  - EXCLUSIVE | MEMORY_LOCAL | MEMORY_EXCLUSIVE

Example: Strictly Managed System



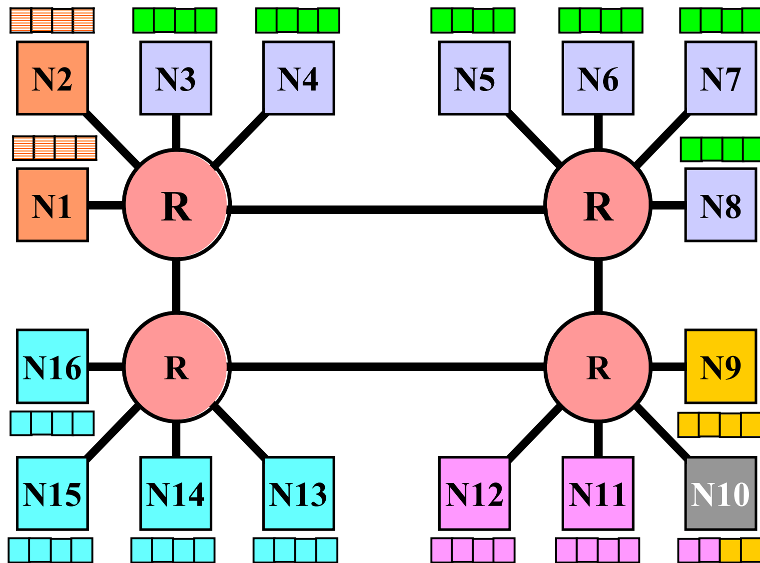
- /tmp/gold.cpuset

```
EXCLUSIVE
MEMORY_LOCAL


CPU 32-37
```

- Cpuset boot defined
  - MEMORY_EXCLUSIVE | POLICY_PAGE
- Cpuset cyan defined
  - EXCLUSIVE | MEMORY_LOCAL | MEMORY_EXCLUSIVE
- Cpuset purple defined
  - EXCLUSIVE | MEMORY_MANDATORY | POLICY_KILL

Example: Strictly Managed System

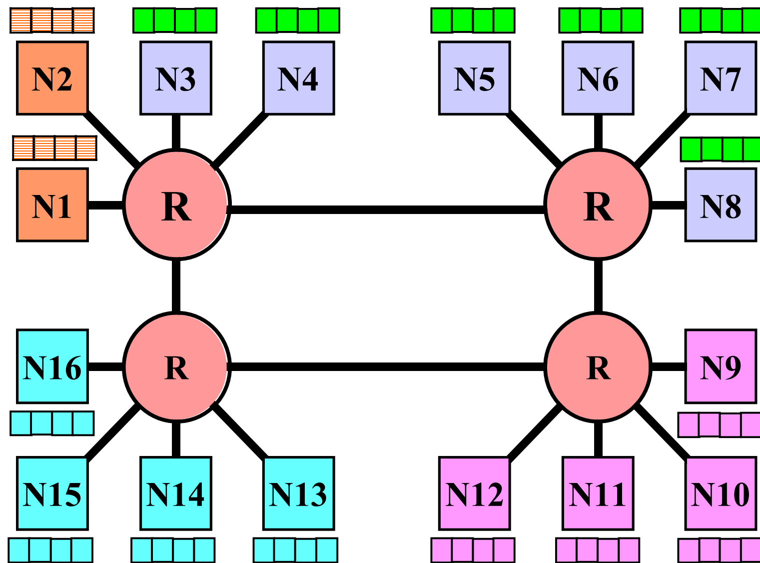


- Cannot “split” memory on a node.
- If two (or more) cpusets contain CPUs on the same node, they all share the memory on that node
- Threads running in those cpusets can cause memory conflicts

- Cpuset boot defined
  - MEMORY_EXCLUSIVE | POLICY_PAGE
- Cpuset cyan defined
  - EXCLUSIVE | MEMORY_LOCAL | MEMORY_EXCLUSIVE
- Cpuset purple defined
  - EXCLUSIVE | MEMORY_MANDATORY | POLICY_KILL
- Cpuset gold defined
  - EXCLUSIVE | MEMORY_LOCAL

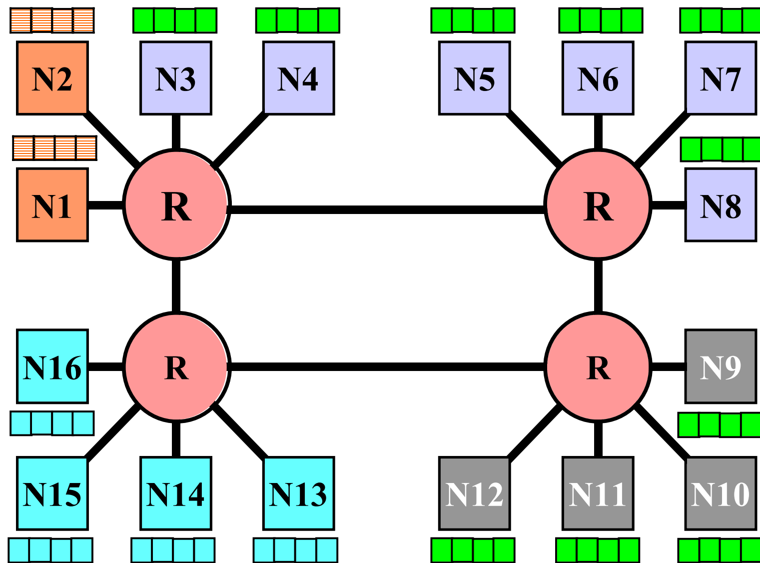
- IRIX® <=6.5.18: Possible to get some strange interaction
- IRIX® > 6.5.18: Memory management will follow the stricter limitation

Example: Preemption



- Cpuset boot defined
  - MEMORY_EXCLUSIVE | POLICY_PAGE
- Cpuset cyan defined
  - EXCLUSIVE | MEMORY_LOCAL | MEMORY_EXCLUSIVE
- Cpuset purple defined
  - EXCLUSIVE | MEMORY_LOCAL | MEMORY_EXCLUSIVE
- We need to run a prime job that requires 32 CPUs
- To define cpuset for prime job, need to borrow space from existing cpusets

Example: Preemption



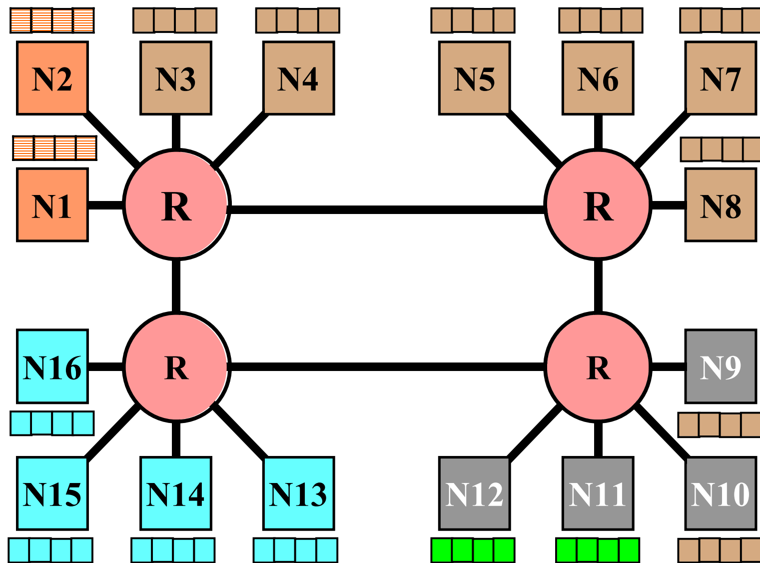
- /tmp/prime.cpuset

EXCLUSIVE
MEMORY_MANDATORY
POLICY_PAGE

CPU 8-39

- Processes in purple cpuset are suspended and then detached
  - using *cpusetDetachPID*()
- Memory used by processes in purple cpuset still exist on nodes 9-12
- The purple cpuset is destroyed
  - using *cpusetDestroy*()

Example: Preemption



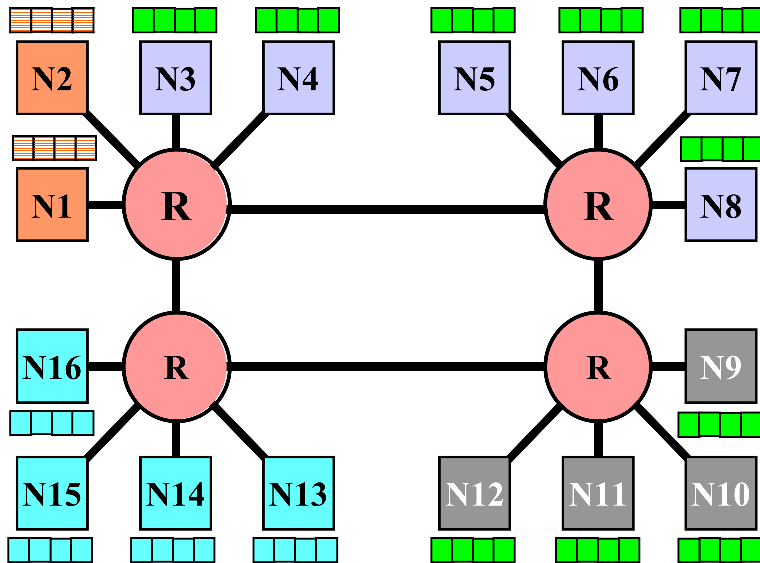
- /tmp/prime.cpuset

```
EXCLUSIVE
MEMORY_MANDATORY
POLICY_PAGE

CPU 8-39
```

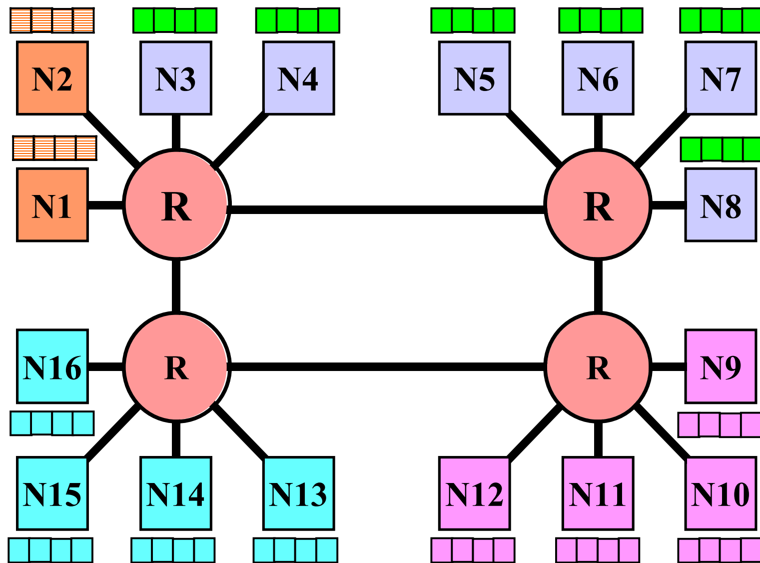
- The prime cpuset is created
  - using *cpusetCreate*()
- Prime job must be able to run in amount of memory that is left on nodes in prime cpuset

Example: Preemption



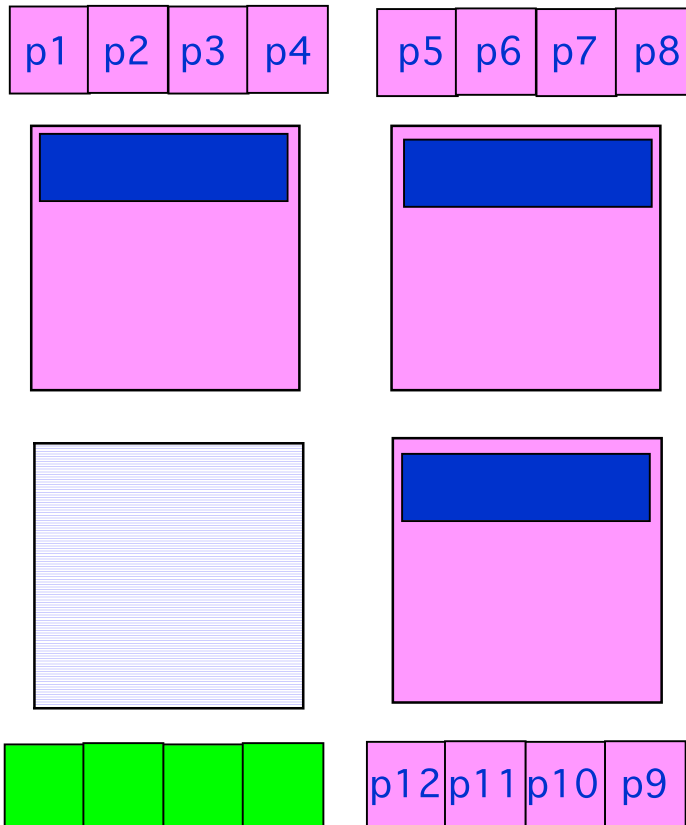
- The job running in the prime cpuset is complete, so cpuset is destroyed
  - using cpusetDestroy()
- Memory used by processes in purple cpuset still exists on nodes 9-12
- The purple cpuset is recreated
  - using cpusetCreate()

Example: Preemption



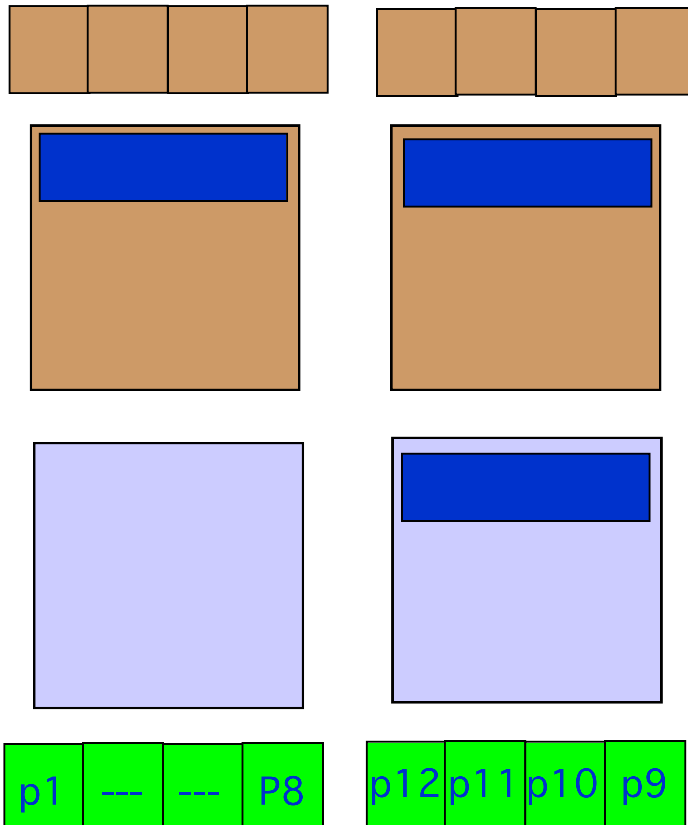
- The processes originally in the purple cpuset are reattached
  - using cpusetAttachPID()
- The processes in the purple cpuset are continued

Example: Preemption Details 6.5.13-6.5.15



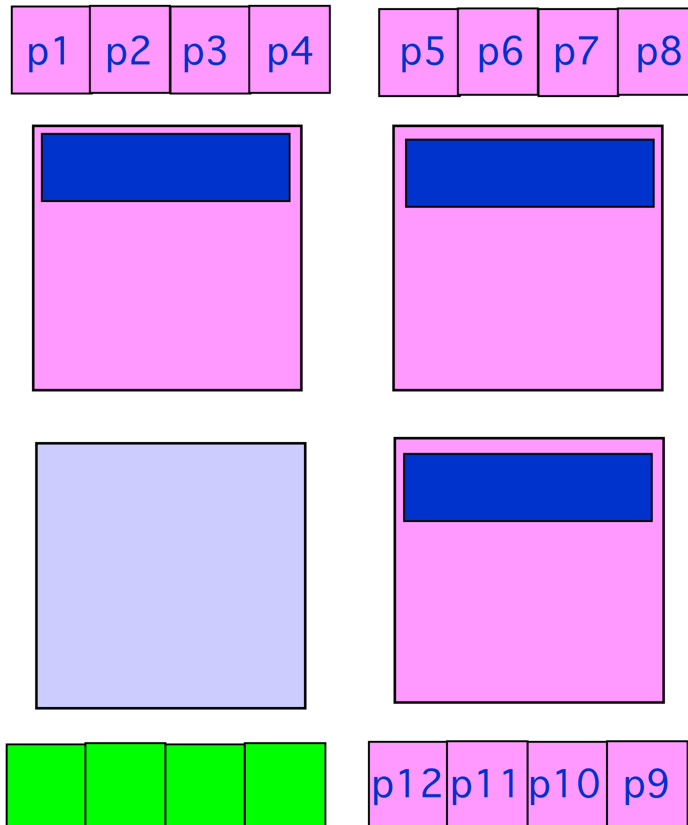
- Need to remove purple cpuset so you can reuse those nodes in a new cpuset for a prime job
- Use *cpusetDetachAll*() to move processes out of purple cpuset
- Purple cpuset is destroyed with *cpusetDestroy*()
- New cpuset, prime, is created with *CpusetCreate*() & job attached with *cpusetAttach*()
- What happens to the memory used by the processes detached from the purple cpuset (p1-p12)?

Example: Preemption Details 6.5.13-6.5.15



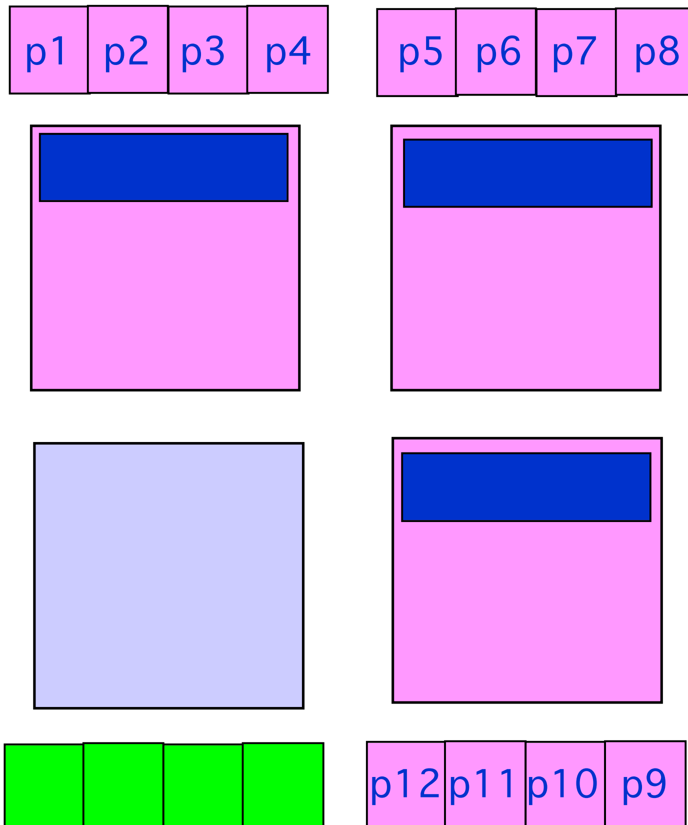
- The MLDs and pages for the p1-p12 process stay on the nodes where they were placed and allocated
- Memory will most likely be remote to the processes
- Use of this memory could interfere with execution of prime job in prime cpuset
- Will introduce variability for p1-p12 and prime job
- Best to suspend p1-p12

Example: Preemption Details 6.5.13-6.5.15



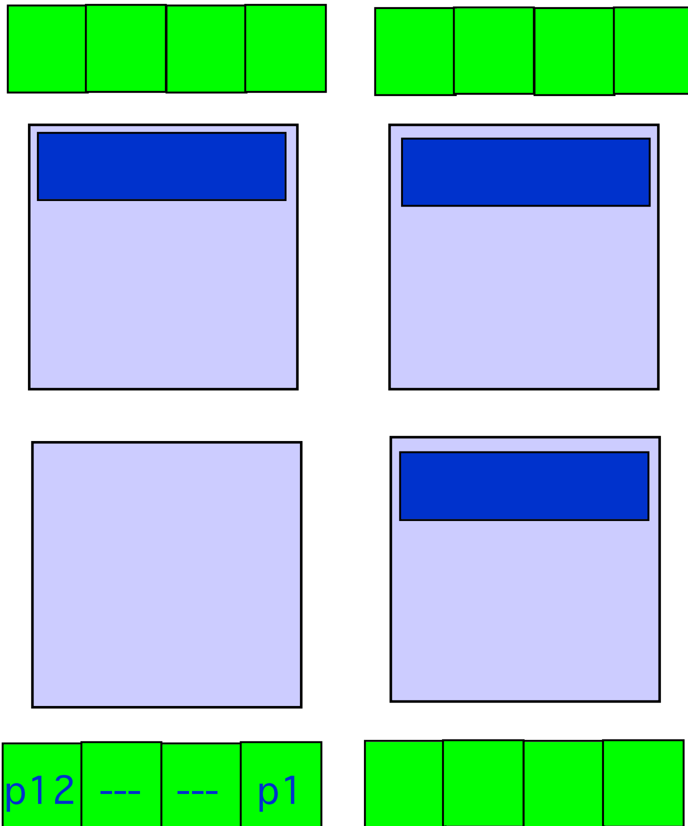
- When prime job is complete, you can destroy prime cpuset
- To continue p1-p12, need to re-create purple cpuset
- Need to reuse CPUs and nodes previously used by purple,or else memory placement will be undesireable
- Use *cpusetAttachPid()* to move process back to purple cpuset
- Need to keep track of processes that have to be moved between cpusets.
- Not ideal, but this scheme

Example: Preemption Details 6.5.16+



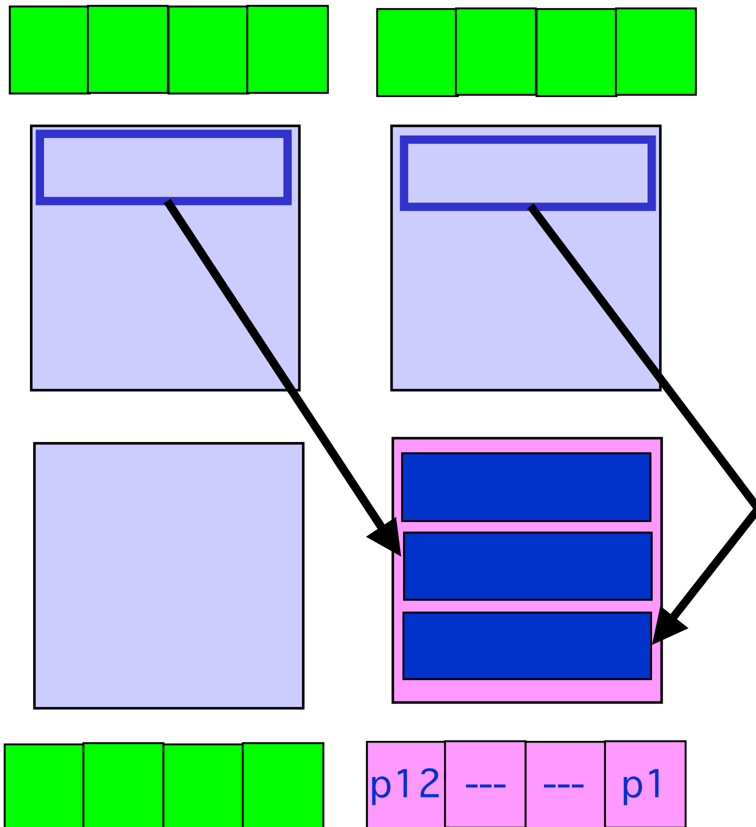
- Same situation as previous example
- Use *cpusetMove*() to move p1-p12 processes out of purple cpuset
- Use *cpusetDestroy*() to destroy purple cpuset
- What will happen to the memory used by p1-p12?

Example: Preemption Details 6.5.16+



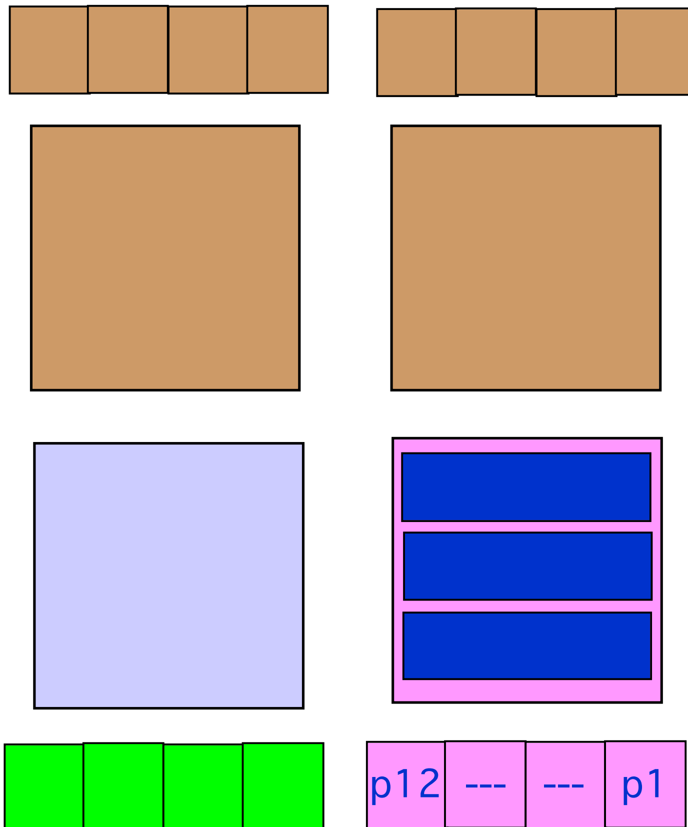
- *cpusetMove*() only moves the processes
- The MLDs and pages do not move with the processes.
- Purple cpuset was destroyed after processes moved out of cpuset.

Example: Preemption Details 6.5.16+



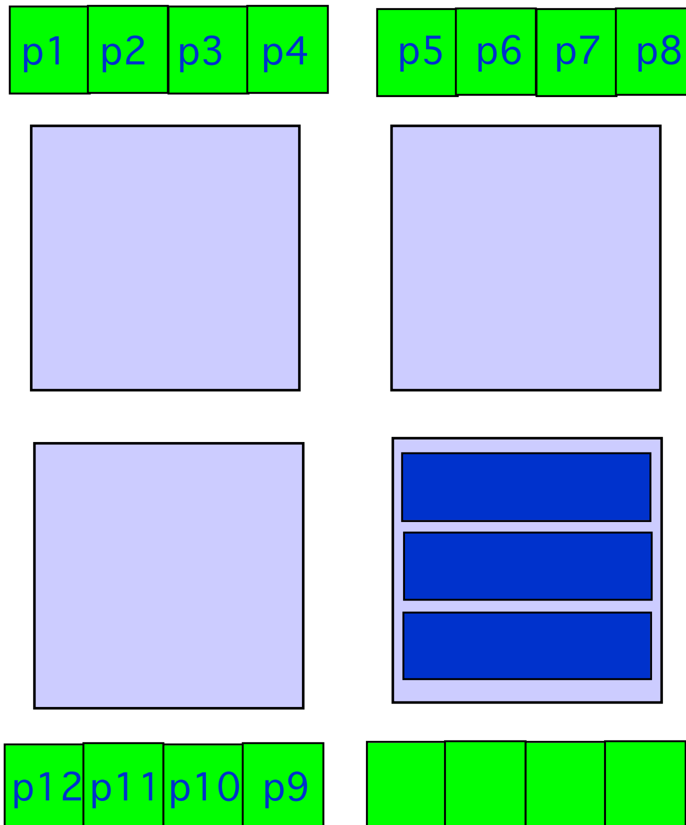
- *cpusetCreate*() creates new purple cpuset
- *cpusetMoveMigrate*() moves processes from the global_cpuset to the purple cpuset and migrates the memory owned by the processes.

Example: Preemption Details 6.5.16+



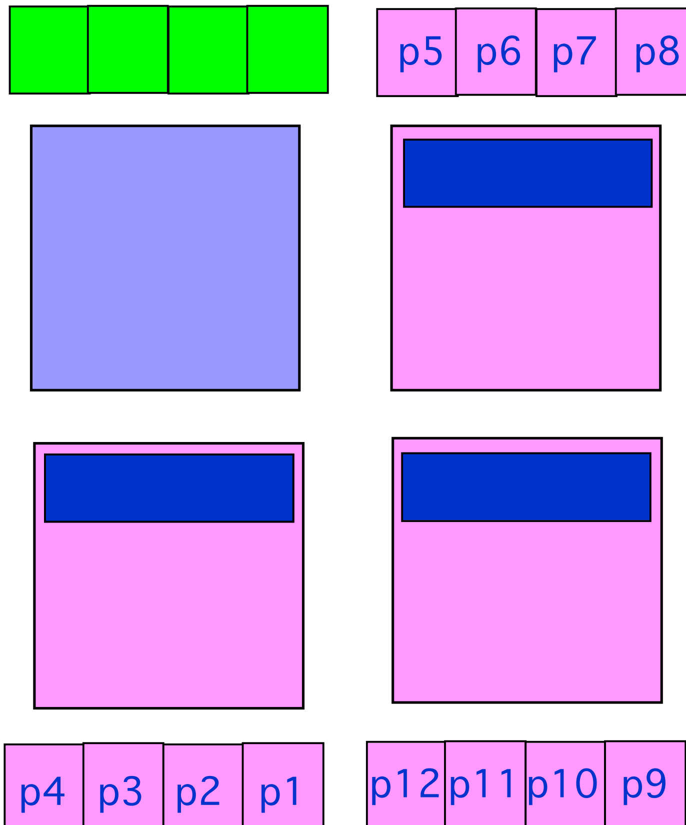
- *cpusetCreate*() used to create the prime cpuset.
- *cpusetAttach*() used to attach new job to prime cpuset
- Processes in purple cpuset continue to execute with degraded performance.
- The p1-p12 processes and the prime job can each continue to run without experiencing external interference.

Example: Preemption Details 6.5.16+



- When prime job completes, *cpusetDestroy*() removes prime cpuset.
- *cpusetMove*() is used to move processes out of purple cpuset.
- Following move, use *cpusetDestroy*() to remove the purple cpuset.

Example: Preemption Details 6.5.16+



- *cpusetCreate()* used to create new purple cpuset.
- *cpusetMoveMigrate()* used to move processes and migrate memory into purple cpuset.
- Ideal scheme, allows preemption of CPUs and memory

Review: The Newest Features

- 6.5.16
  - Work done in VM system to allow migration of MLDs & pages
    - Useful when the migration can be “triggered” based upon known system state
  - Checkpoint/Restart (cpr) modified to allow migration
    - On restart, user may supply option to allow migration
    - When you restart a job in a cpuset, it does not have to contain the same CPUs/nodes
  - Interfaces added to cpusets to allow migration of processes+memory between cpusets
    - Move processes+memory directly between cpuset A to cpuset B
    - Move processes out of cpuset A, destroy cpuset A, create cpuset B, move processes+memory to cpuset B

Where We Might Be Going

- Beyond 6.5.16
  - Enhance cpuset permissions
    - Remove requirement for permissions file
      - » will still be able to use a permissions file
    - Provide permissions scheme similar to access control lists
      - » designate at user or group level
      - » designate level of access: read or run
    - Resolves issue concerning persistent vnode reference
      - » means the filesystem where permissions file was located at cpuset creation cannot be unmounted until the cpuset is destroyed - even if the file is deleted
  - Ability to alter cpuset resources on-the-fly