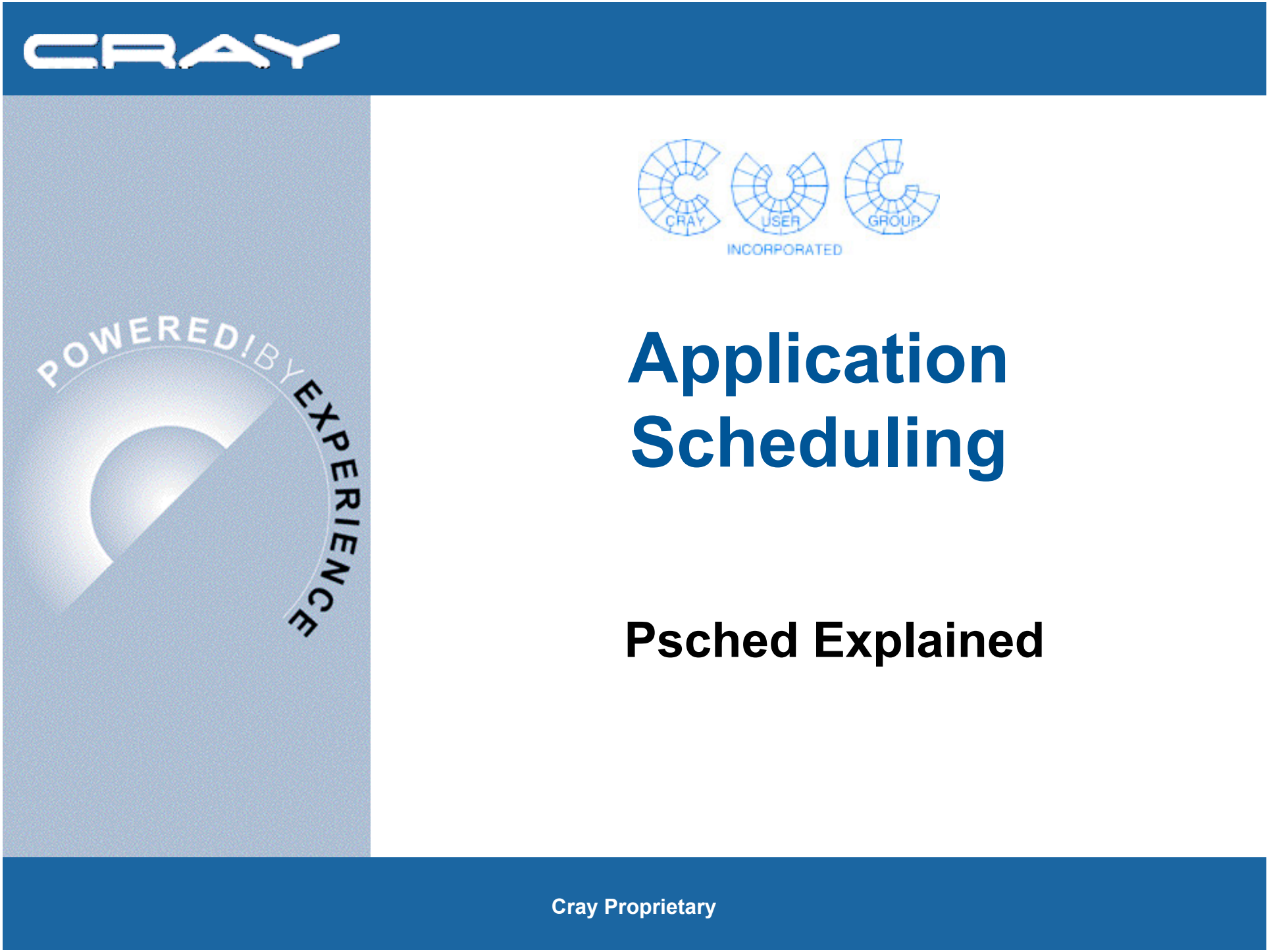

CRAY
POWERED BY EXPERIENCE
CRAY USER GROUP
INCORPORATED
Application Scheduling
Psched Explained
Cray Proprietary

# Introduction

- Definitions
- Application placement rules
- Migration and checkpoint/restart
- Prime applications
- Gang scheduling
- Load balancing
- Memory management
- Configuration flexibility
- Recovery features

CUG 2004

# Definitions – 1

- Node- 4 MSPs/16 SSPs and a single cache domain of memory
- Module-
  - X1: consists of one node named cache domain 0
  - X1E: consists of two nodes named cache domain 0 and cache domain 1
- Node numbers are
  - cache domain 0: 2 * moduleNumber
  - cache domain 1: 2 * moduleNumber + 1

# Definitions – 2

**Placement *reserves resources* for applications on each module and node**

- **The kernel CPU scheduler selects the processors for each application. *Placement cannot choose which processors on a node an app will be given***
- **By selecting which apps are eligible to execute at the same time, placement guarantees processor competition does not occur**

# Application Placement

- GASID assignment
- Power-of-2 PEs per module
- Memory residency rule
- Module contiguity rule
- What does depth mean?
- Shape freezing

CUG 2004

# GASID Assignment

- Required by each ACX module-spanning app
- Each app uses the same GASID over all occupied modules
- Property of the MCM – four GASIDs / module
- May be donated to a prime app. Donor is suspended but may migrate and run.
- Is reallocated for migration and restart

# Power-of-2 PEs Per Module

- **The off-node addressing model requires that every module allocated to an app but the last has the same power-of-2 PE count.**
- **Placement will fill modules with PEs unless the aprun –N option alters that behavior or memory requirements or partial modules need smaller PE counts to observe the uniform power-of-2 rule.**

# Memory Residency Rule

- A module-spanning app is a real memory app
- A single module app is a VM app
- An off-module memory reference must not generate an exception (page fault) if the app is to survive so all app memory is pinned during execution
- The gang scheduler and kernel coordinate real memory residency for executing apps

# Module Contiguity Rule

- **An ACX app must be placed on contiguous modules and cannot wrap**
- **A FLX app need not be placed on contiguous modules and may wrap**

# What Does Depth Mean?

- Depth is the number of processors per PE specified with the aprun –d option
- All of the processors for a PE must be in the same cache domain (node – not module)
- Maximum depth for an MSP app is 4
- Maximum depth for an SSP app is 16
- Placement simply reserves the processors; the app must be able to use them

# Shape Freezing

- The shape an app (PEs per module) is originally given cannot be changed because the PE to module mapping is established by startup and patched into the app
- The aprun –N option exploits this placement restriction by making an initial launch appear as a restart to the app allocation function

# How Migration Works

- A target placement list is created
- The app is deselected – it stops executing
- RTT memory pages are unlocked
- The apteam placement list and GASID assignment are changed by psched
- The app context moves to the target modules
- Psched selects the app for execution
  - Memory is copied by each module in parallel from the source to the target and locked
  - When all of the memory is locked, app execution continues

# Checkpoint and Restart

- Psched is not involved with checkpoint
- Restart is a combination of application recovery and posting
  - Restart creates an apteam with the psched recovery data restored from the checkpoint
  - Restart makes a special posting request to psched
  - Psched recovers the app information from the apteam and reconstructs its internal representation
  - Psched finds a place for the app and signals restart to restore the app in the allocated places
  - Psched now treats the app normally

# Launching a Prime App

- **Prime can be specified by a privileged user (having psched `a` or `p` privilege) with the aprun –f option**
- **Prime can be set by the administrator with psmgr –f <*apid*>**
- **Try not to create competing prime apps**

CUG 2004

# Placing a Prime App

- If there is a place, launch it marked as prime
- Otherwise
  - Find a place for the app ignoring GASID availability
  - Identify all apps that must donate their GASID. Mark them donors, deselect them, remove their GASID assignment and change them to FLX
  - Assign the donated GASID to the prime app
  - Select the app for execution and mark it prime

# What Happens to the Donors?

- The donors are “under” the prime and cannot execute
  - A donor may be migrated to a place with an available GASID and resume execution
  - A donor may wait under the prime until the prime exits and its donated GASID is returned. (The GASID need not have the same number as the one donated)

# Gang Scheduling

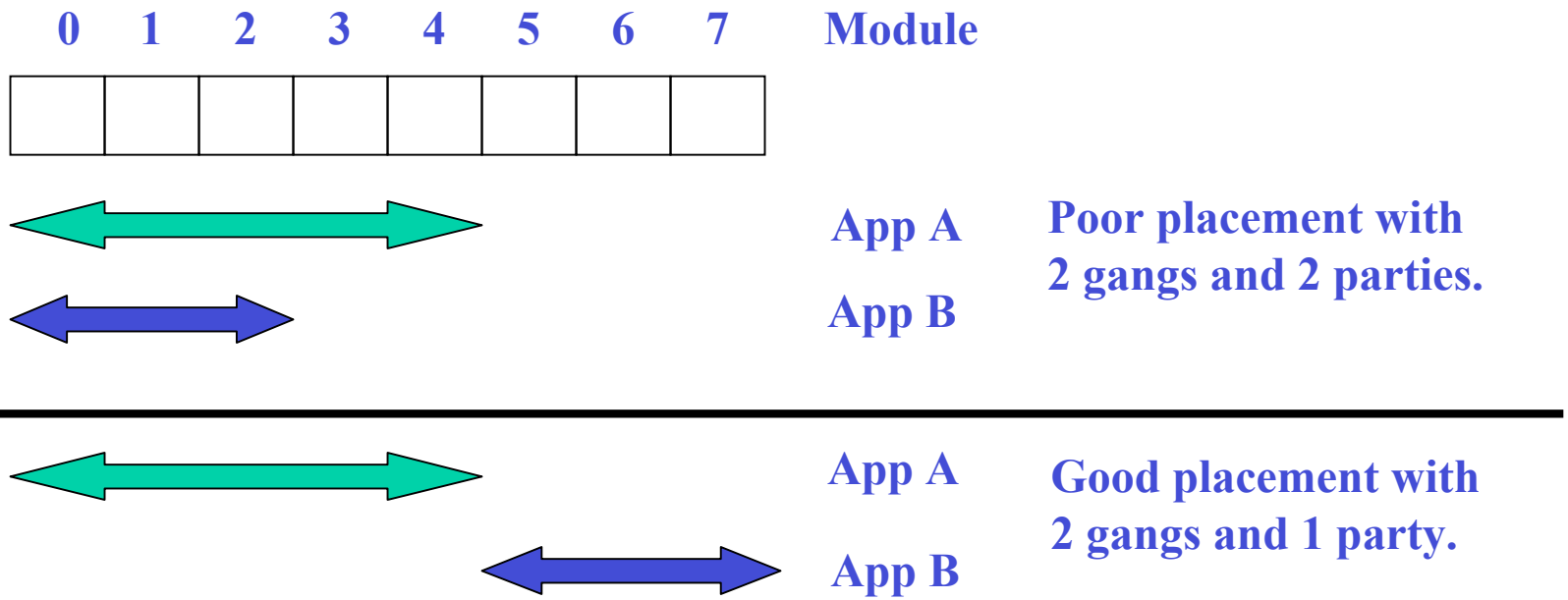
**Gang scheduling is active only when memory and/or processors are oversubscribed**

- **Gang –** ***the processors assigned to an app are a gang***
- **Party –** ***all of the gangs that can execute together without interference are a party***
- **Inter-gang interference issues**
- **Context switching strategy**

# Inter-gang Interference

- Processors cannot be oversubscribed on a node because application synchronization will not be stable if the local kernel is doing CPU scheduling for the app
- Executing applications cannot oversubscribe memory on a node because module-spanning apps must have shared (RTT) memory locked during execution

# Gang / Party Relationship




- The top placement needs two time slices, only one app runs per slice and 3 or 5 modules are idle during a slice.
- The lower placement needs one time slice to run both apps and all modules are used.

# Context Switching Strategy

- **Context switching proceeds in 2 steps:**
  - **One system call provided with a list of *apids* is used to disconnect. This synchronously stops execution and unlocks RTT memory pages**
  - **A dedicated thread for each connecting app makes a connect system call**
- **Psched will connect 16 apps in parallel. Each node manages its own memory pages so a great deal of parallel work speeds the connection process**

# Load Balancing – 1

- Launch applications
- Migrate apps according to the rules
- Implement manual migration requests
- Find a place for a prime app by identifying donors of GASIDs, memory and processors so the prime app can be placed

*Migration can take some time to complete so psched tries constantly to satisfy the rules assuming the resources will be needed for new work*

# Load Balancing – 2

- Rules can be changed during operation
- Rule changes take effect on the next launch or balance cycle
- An induced migration will have its reason for migration logged
- If “NoPreemptiveMigration” is true, migration will only occur
  - if there is a backlog of posted apps or
  - the population has changed since the last time.
- Only one launch or migration at once will be scheduled

# Placement Vector Ordering

***An evaluation cycle begins with a list of possible places for the application. The order of the list is important since the first "best" place will be chosen.***

**The two primary sorting rules defined in `SortRules` are:**

**LabelScore and Distance**

**With sub-ordering by:**

**wrap and gravity**

# LabelScore

**Move places matching the requested label to the top of the list**

- **This rule attracts the application to places with a matching soft label. Hard label mismatches will never appear in the list of possible places.**

***Always define this rule first***

# Distance

**This rule orders the list by increasing maximum hop count.**

- **Although this rule would typically improve the performance of tightly synchronized apps, it tends to badly fragment the application space.**
- **For this reason and because it appears to place apps in seemingly arbitrary places, use of this rule is not recommended. (Unless you want to constantly answer the question, “Why is my app placed where it is?”)**

# Wrap and Gravity

**These rules are always used in this order**

- **The wrap rule favors places that do not wrap from high to low modules.**
- **The gravity rule orders the placement list by increasing distance from the gravity edge (low or high) of the application space.**

***The placement list is now prepared for presentation to the six rules of load balancing…***

# Load Balancing – 6 Rules

- The six rules:
  - Prime
  - Swapping
  - Parties
  - Fragmentation
  - Utilization
  - Idle
- A rule returns “better”, “equal” or “worse”
- The order they appear in the `Rules` config variable is followed for evaluation. The first better/worse decision stops evaluation.

CUG 2004

# Load Balancing – Prime

**Minimize the number of stacked prime apps**

- **This rule chooses a scenario that has the fewest interfering prime applications.**

***This rule should always be present and appear first in the list of rules to give best service to prime applications.***

# Load Balancing – Swapping

**Minimize node memory allocation**

***The name was inherited from the Cray T3E and is technically incorrect for the X1***

- **This rule chooses a scenario with the least amount of node memory allocation – to minimize memory oversubscription**

# Load Balancing - Parties

**Maximize the number of concurrent applications in execution**

- **An application position is picked that minimizes the number of parties (time slices) required to execute all of the placed applications.**

# Load Balancing - Fragmentation

***This is a complex rule that often exhibits unexpected behavior because of its four very different comparisons***

- **In this order select the first of:**
  - **the minimum number of allocated modules**
  - **the minimum number of fragments**
  - **a larger unallocated span of modules**
  - **a choice that matches gravity direction**

# Load Balancing - Utilization

**Maximize the number of allocated modules**

- **This rule minimizes the amount of module processor oversubscription**

  ***A porting error makes this rule ineffective***

# Load Balancing - Idle

**Maximize the number of idle modules**

- **Useful in an environment where it is important to leave as many empty modules as possible for assignment to a newly posted app**
- **This rule may oversubscribe modules when it might not be necessary since the the future workload is unknown**

# Memory Management – 1

**Starting with UNICOS/mp 2.4, psched will use memory as an allocation attribute.**

- **A startup problem requires apps to be relinked to run with memory management enabled.**
- **For 2.4 ONLY psched can be configured to mimic 2.3 memory management behavior. This is done by setting `/Global/UseMemoryLimit` to 0. This variable will be disabled in release 2.5**

# Memory Management – 2

- An app without an aprun memory size (-m option) specification will by default be given _ node memory if MSP and 1/16 node memory if SSP
- Memory size defaults are configured with `/Global/DefaultMemoryMsp` and `/Global/DefaultMemorySsp`
- An explicit `UNLIMITED` memory specification will mean all of the memory on a node

# Memory Management – 3

**Memory oversubscription: `unlimited`**
**Processor oversubscription: `1`**

***Memory use is defined as what is declared NOT what is actually allocated. Limits prevent abuse.***

- **Multiple parties will be created if two or more applications allocated to a node together use more than the total memory. This causes time slicing.**
- **Setting memory oversubscription to 1 eliminates time slicing**

# Configuration Flexibility – 1

- **Region: a named set of one or more modules having identical gates and limits. The modules making up a region may not appear in another region**
- **Domain: a named instance of load balancing and gang scheduling with its own configuration, gates, limits and region list. A region may be named in only one domain.**

***Experience with the X1 suggests that a single domain made up one or more regions is the most useful and flexible configuration***

CUG 2004

# Configuration Flexibility – 2

- Configuration variable values can be changed dynamically and their new values will be used
- The content of a domain or region are frozen when psched is initialized
  - Gates or limits cannot be added or removed although their values may be changed
  - Regions cannot be added or removed from a domain
  - Modules cannot be added or removed from a region

CUG 2004

# Configuration Flexibility – 3

- If variables are deprecated, psched will tolerate old `psmgr.config` files
- Typically a callback is added so the value of a deprecated variable cannot be changed by `psmgr`
- Whenever possible new variables will be forced to a default value so old `psched.conf` files will work

*The administrator should watch for changes*

# Psched Recovery – 1

- All needed application state information is recorded in the apteam kernel structure.
- Psched recovers running app information when it initializes.
- It is usually safe to stop psched with `SIGINT` or `SIGTERM` and restart it for upgrade or other purposes.
- Removing modules from the scheduling domain will make recovery of apps allocated to removed modules impossible.

# Psched Recovery – 2

- Trace messages are recorded in the psched log file when the daemon terminates. Traces can be useful along with the remainder of the log file to understand scheduling events.
- The trace buffer is circular so a record of events is not retained forever. If a trace record is desired, send psched a `SIGUSR2` to cause a dump of the current trace buffer to the log without disrupting operation.
- We can extract traces from a core file with a debugger.

# End

- Questions
- Comments
- Discussion