Application Scheduling

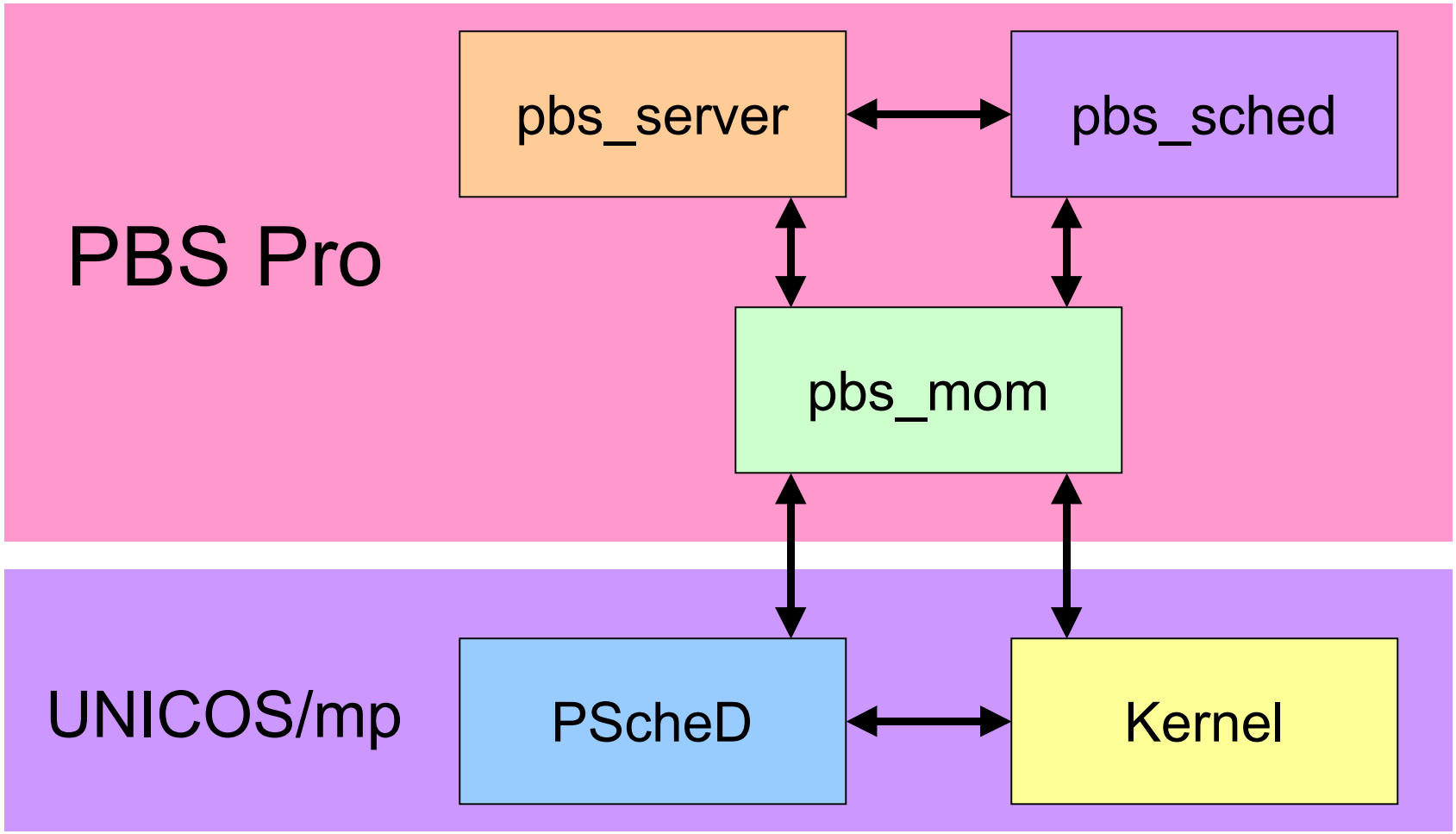
Richard Lagerstrom

15 MAY 2003

Scheduling Hierarchy and Scope

Name	Scope	Example
Grid	Global	Globus
Batch	Organizational, Departmental, or Cluster	PBS Pro
Placement	Single System, Multinode	PScheD
Process	Single Node, Multiprocessor	UNICOS/mp

Functional Organization



History

- Psched was ported from Cray T3E
- Enhanced to do initial placement
- Modified to support multi-CPU nodes
- User and admin. displays through `psview`
- More displays with `apstat`
- Cray X1 kernel cannot initiate applications without the assistance of psched

Introduction

- Placement strategies
- Placement requirements
- Starting an application
- Gang scheduling
- Migration
- The PBSpro interface

Placement Strategies

Many configurable options

- Equalize node workload
- Minimize node fragmentation
- Maximize processor utilization

Placement Requirements

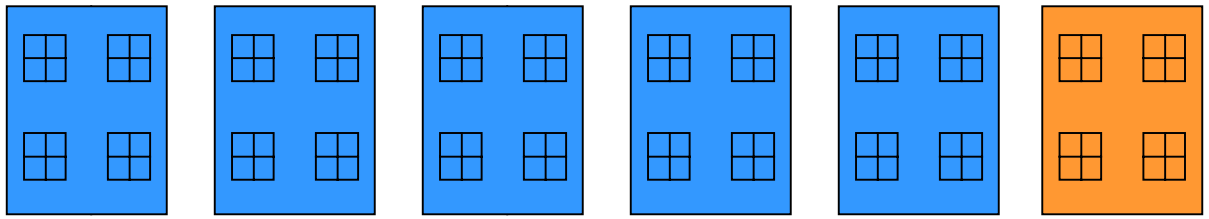
- Power-of-2 MSP/SSP per node
- Memory loaded when executing

Accelerated applications need:

- Global address space ID (GASID) for off-node accelerated memory references
- Node contiguity

Six-node Example

- Each node has 4 MSPs and 16GB memory
- Five with application flavor
- One with operating system and support flavors

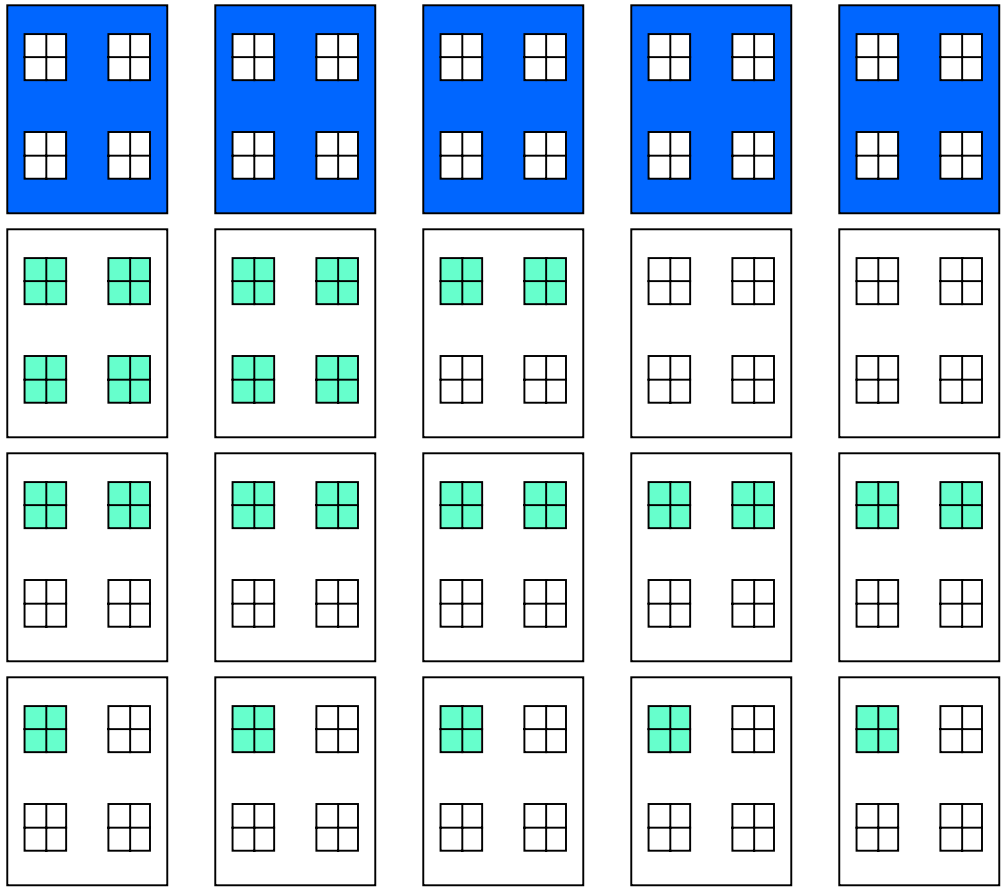


Application Mapping

How many PEs are allocated to a node?

- User option to choose PEs/node
- Psched will pick a mapping by default
- Memory usage per PE is the major reason for user specified mapping

Mapping Examples



-n 10 –N 4

-n 10 –N 2

-n 5 –N 1

Posting an Application

Support node phase

- Run `aprun -n x -N y a.out`
- aprun checks for option errors
- aprun sends an RPC request to psched to post the app for placement
- aprun waits for a signal to continue
- Psched gets PBSpro queue limits
- Psched creates an apteam and joins aprun

Placing an Application

- Psched generates placement information
- Psched sends placement information to the kernel
- On the next time slice psched sends a start signal to aprun to exec( ) PE 0 of the app

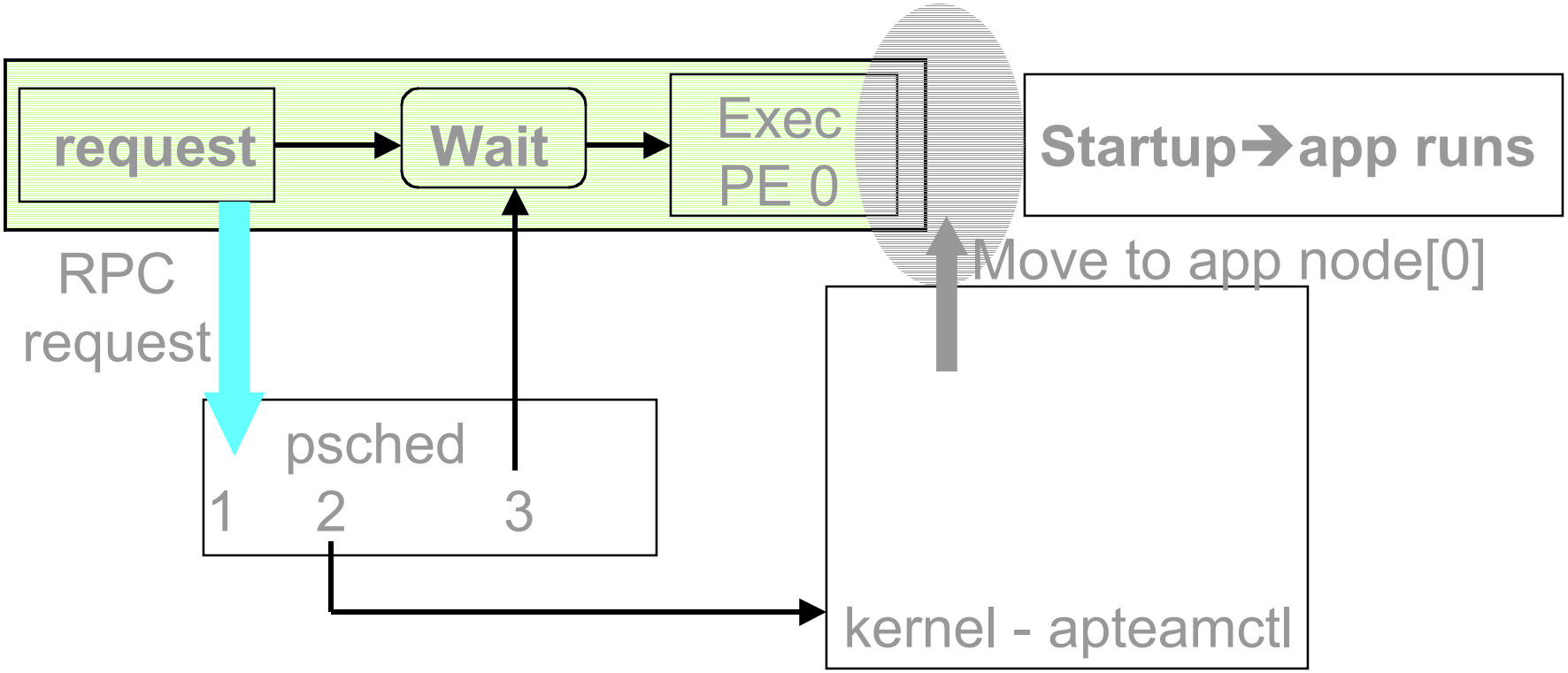
Application Startup

Application node(s) phase

- Execution begins in `startup()` which sets up the shared memory environment
- All PEs of the app are created with a *placed* `fork()`
- App execution begins in `main()`

Running an Application

User types `aprun -n20 a.out`

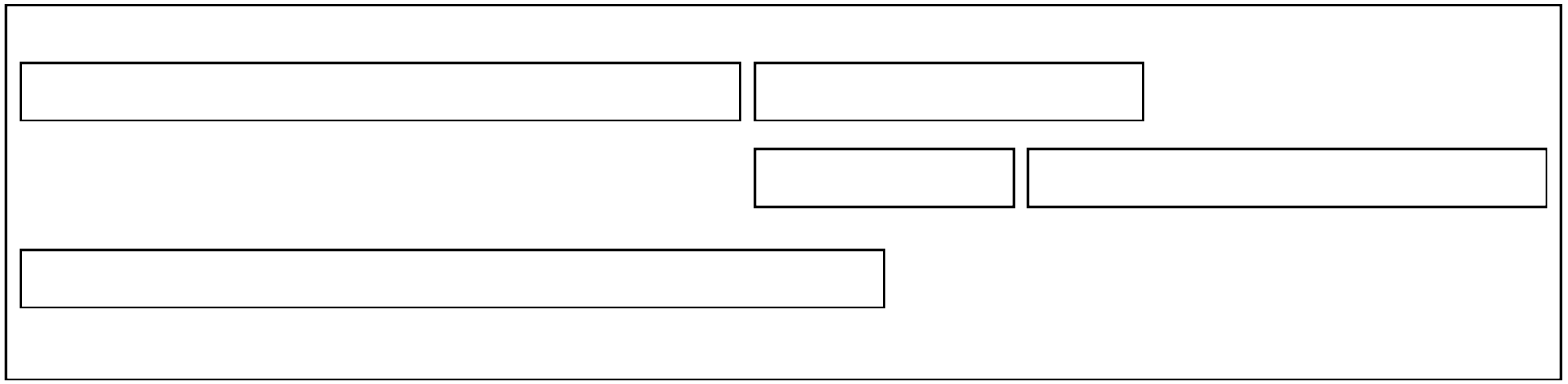


Application Execution

- Apps are time sliced by psched
- Memory of inactive apps may page out
- Memory of active apps is locked in

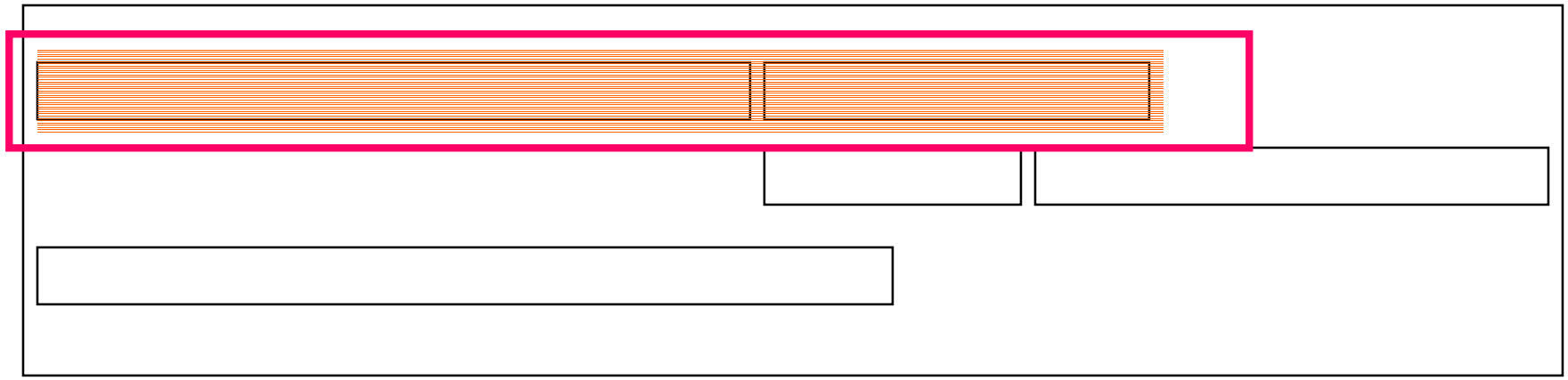
Gang Scheduling

Five gangs – three parties



Three time slice example

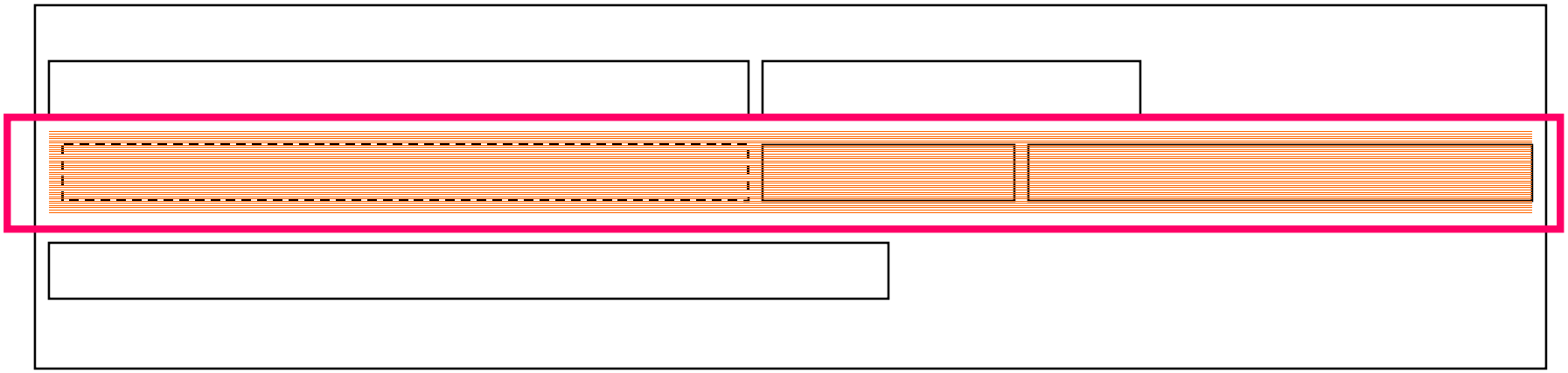
Gang Scheduling 1



Five gangs – three parties

First time slice

Gang Scheduling 2

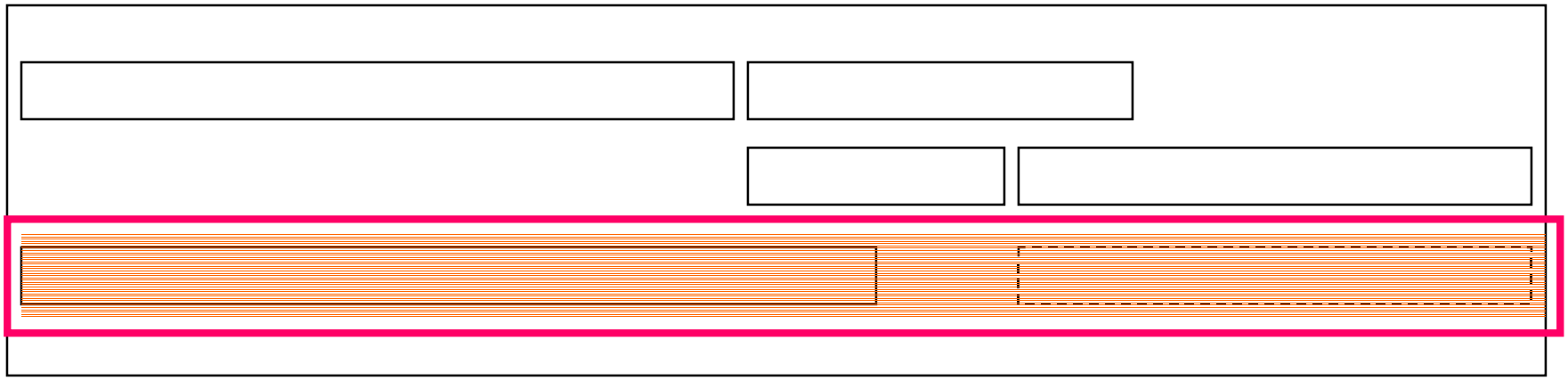


Five gangs – three parties

Second time slice

Gang Scheduling 3

Five gangs – three parties



Third time slice

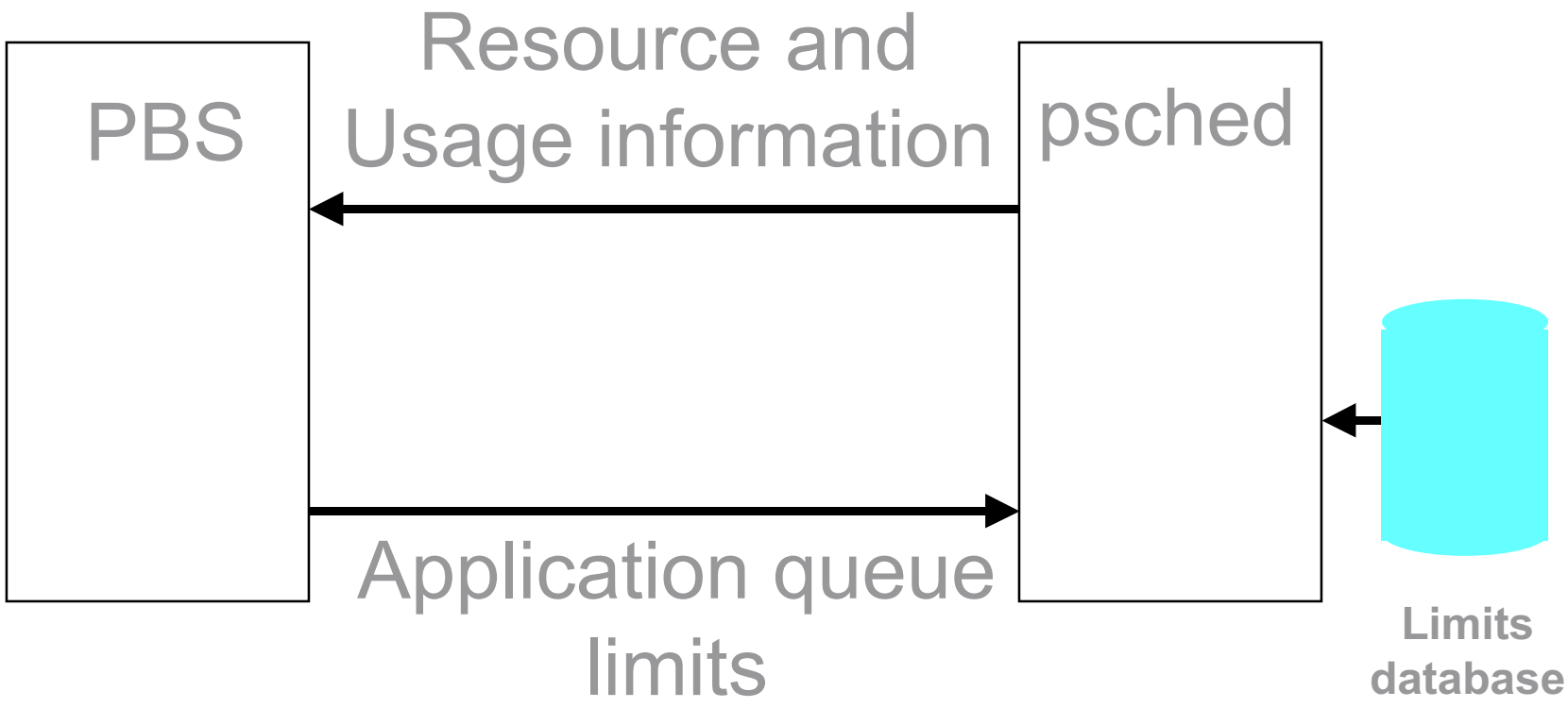
Application Migration

- A target place list is generated
- The app is disconnected to stop execution and unlock its memory pages
- The target place list is given to the kernel
- The app is connected
- Memory pages are moved from the origin nodes or disk to the target nodes
- Execution begins on the target nodes

Application Exit

- Each PE exits
- PE 0 waits for all other PEs to exit
- When PE 0 exits the kernel detects the PE count is zero
- The kernel sends psched the app exit signal
- Psched deletes the kernel's apteam entry and its internal information about the app

PBSpro/Psched Interface




End


The 45th CUG Conference
CRAY USER GROUP
OSC
FLIGHT TO INSIGHT

---end---