OpenSolaris and NUMA Architectures

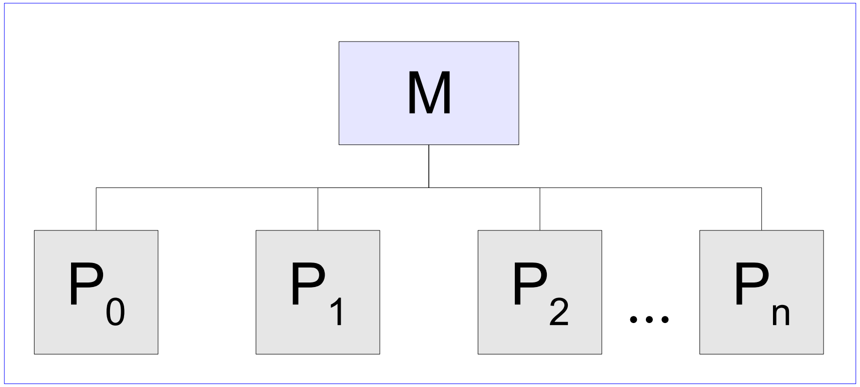
Rafael Vanoni Polanczyk
Solaris Kernel Performance Group
Sun Microsystems

Overview

- Where did NUMA come from ?
- Access latencies
- Topology representation
- Optimizing for NUMA
- Topology discovery
- A peek at the implementation
- Where NUMA is going

Where did NUMA come from ?

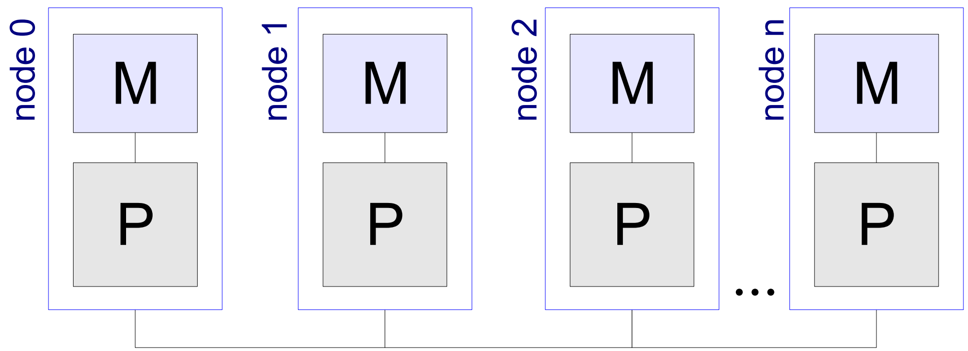
- Symmetric Multiprocessing (SMP)
  - processors connected to memory through a single bus
  - shared memory, single address space
  - Uniform Memory Access (UMA)



  - traditional programming paradigm
  - model doesn't scale as processors are added

Where did NUMA come from ?

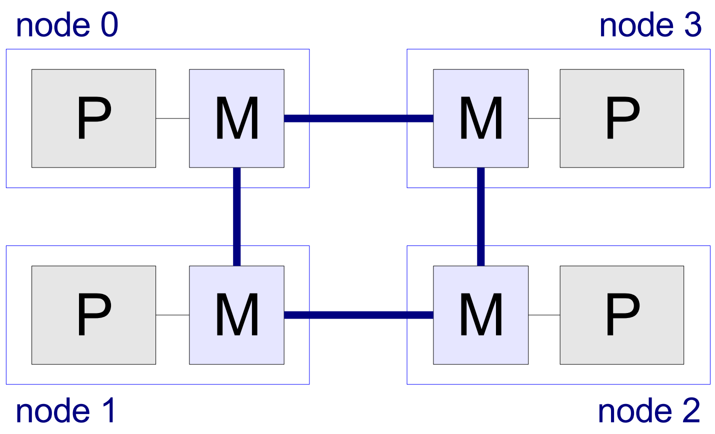
- Clusters
  - each node has it's own memory
  - network connected
  - multiple address spaces



  - scalable
  - network programming paradigm
  - replication (software, data, ..)

Where did NUMA come from ?

- Non-Uniform Memory Access
  - nodes composed of processors with *local* memory
  - can access *remote* memory through the *interconnect*
  - *local* access faster than *remote* (non-uniform)
  - shared memory, single address space



  - more scalable than SMPs
  - traditional programming paradigm

Where did NUMA come from ?

- Non-Uniform Memory Access
  - it's actually "*cache coherent*" NUMA, or *ccNUMA*
  - hardware cache coherence
  - Distributed Shared Memory (DSM)
    - cache coherency algorithm and implementation varies from model to model and between manufacturers
    - affects performance

NUMA and OpenSolaris

- Memory Placement Optimization (MPO)
  - framework to provide better performance through *locality awareness*
  - implemented through
    - Common framework support
      - usr/src/uts/common/sys/lgrp*.h
      - usr/src/uts/common/os/lgrp*.c
    - Platform specific support
      - usr/src/uts/{i86pc,sun4}/os/lgrpplat.c
  - interfaces for observability and control
    - APIs
    - Tools

Access Latencies

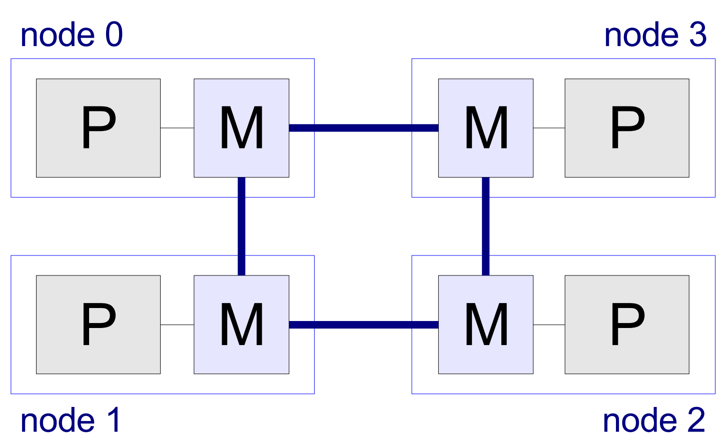
- Local x Remote
  - NUMA factor = Local Access / Remote Access
  - varies between machines/vendors/manufacturers
- Locality awareness
  - keep threads close to where their data/device is
  - avoid remote accesses
  - take advantage of cache warmth
- Conflicting objectives
  - load balance across nodes
  - minimize data migration

Topology Representation

- Locality groups (*lgroups)*
  - represent sets of resources close to each other
  - each lgroup has at least one processor and possibly some memory and/or devices
  - hierarchical: different levels of locality
    - leaf lgroups
    - intermediate lgroups
    - root lgroup
  - forms a *latency topology*

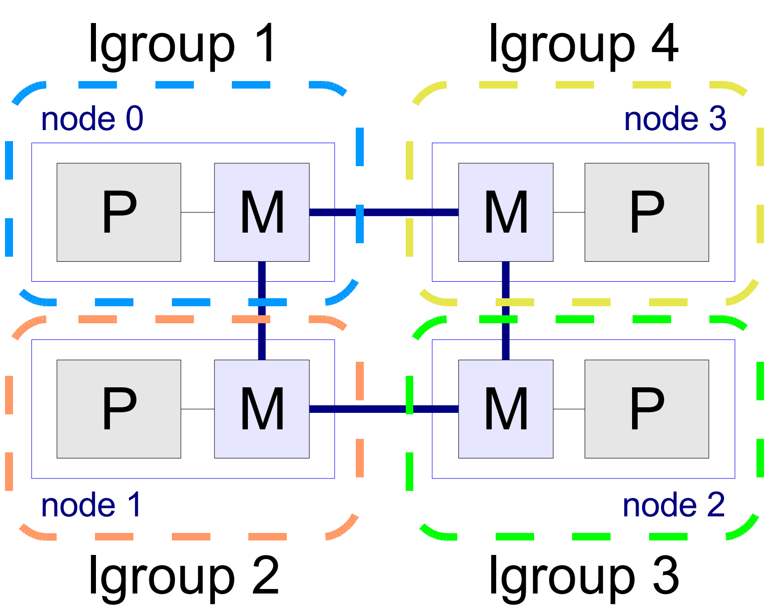
Topology Representation

- Leaf lgroups
  - contain resources at the lowest level of latency
  - only local accesses
  - Example: 4 node, ring topology



Topology Representation

- Leaf lgroups
  - contain resources at the lowest level of latency
  - only local accesses

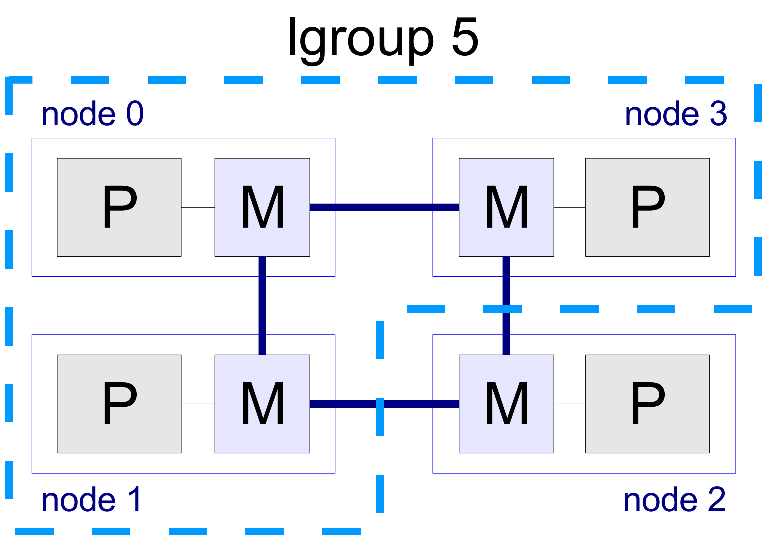


lgroup topology:

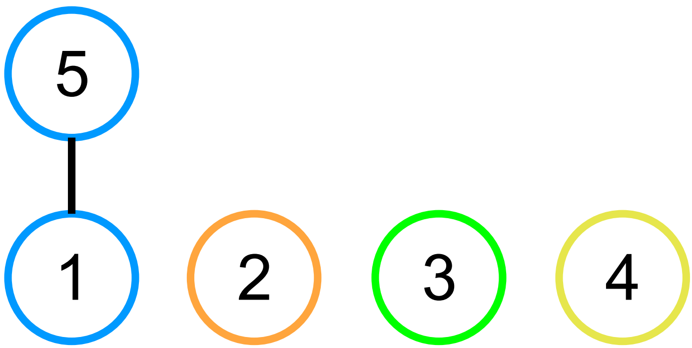
opensolaris

Topology Representation

- Intermediate lgroups
  - contain CPU and memory resources with one hop
  - lgroup 5 contains lgroups 1, 2 and 4

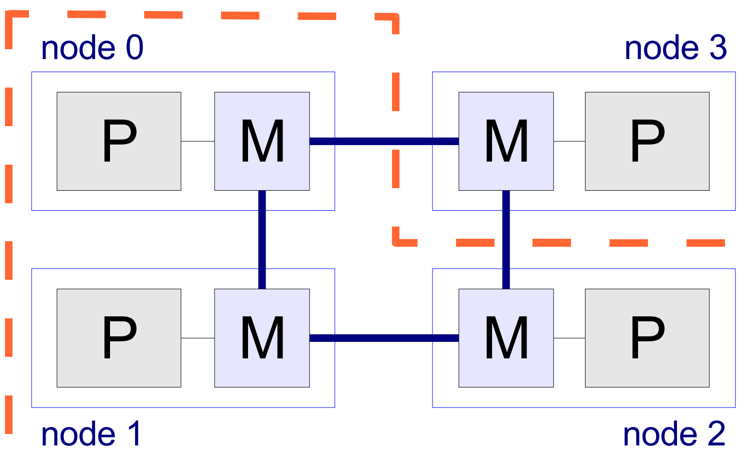


lgroup topology:



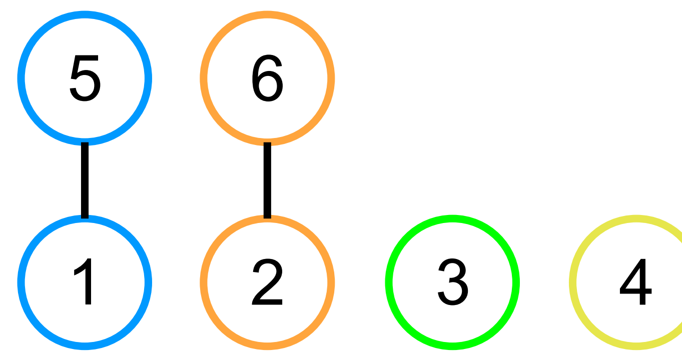
Topology Representation

- Intermediate lgroups
  - contain CPU and memory resources with one hop
  - lgroup 6 contains lgroups 1, 2 and 3



lgroup 6

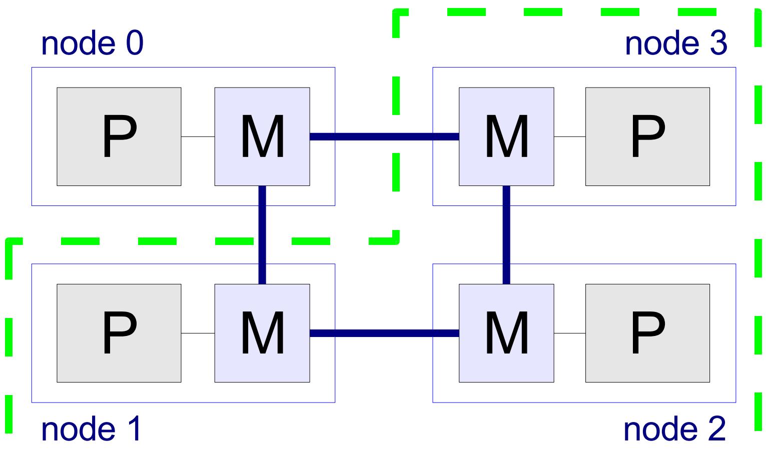
lgroup topology:



opensolaris

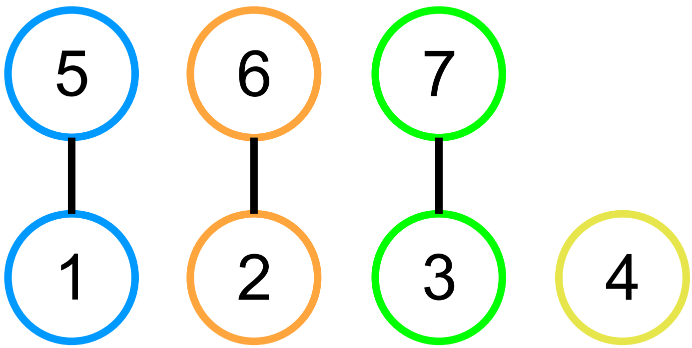
Topology Representation

- Intermediate lgroups
  - contain CPU and memory resources with one hop
  - lgroup 7 contains lgroups 2, 3 and 4



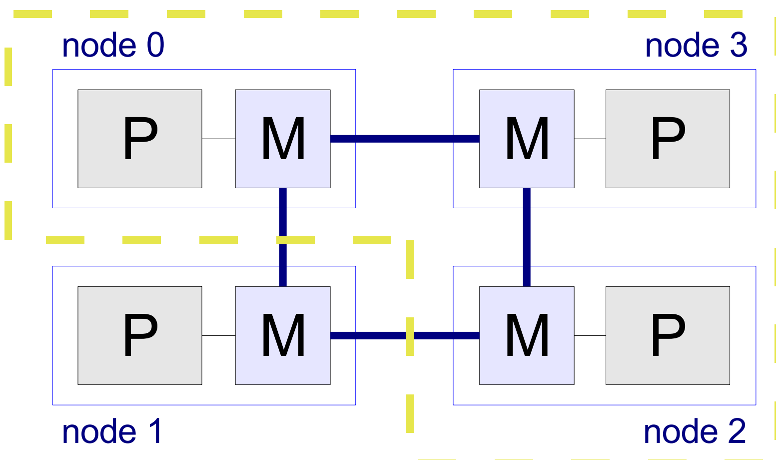
lgroup 7

lgroup topology:



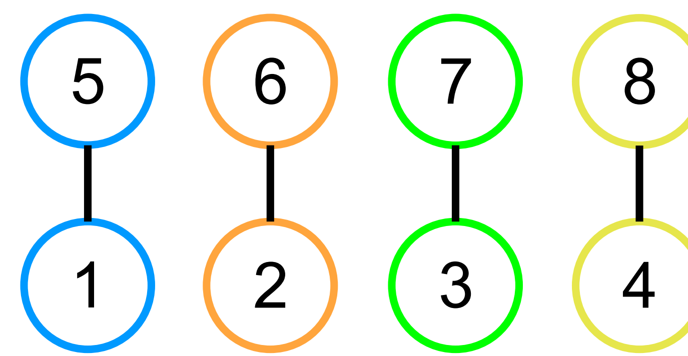
Topology Representation

- Intermediate lgroups
  - contain CPU and memory resources with one hop
  - lgroup 8 contains lgroups 1, 3 and 4



lgroup 8

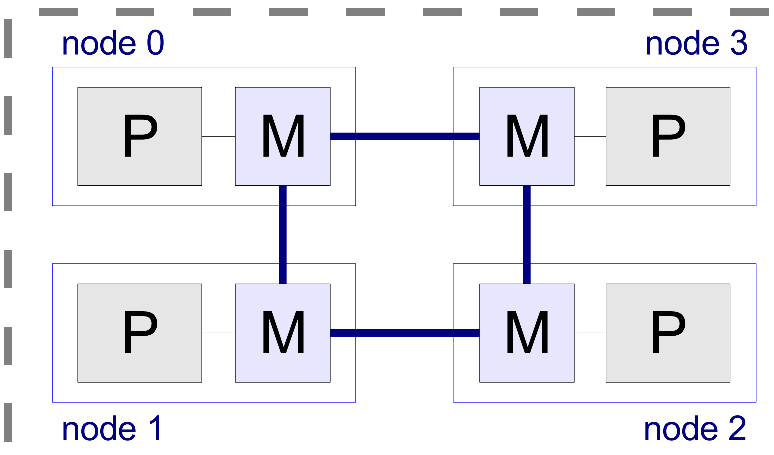
lgroup topology:



opensolaris

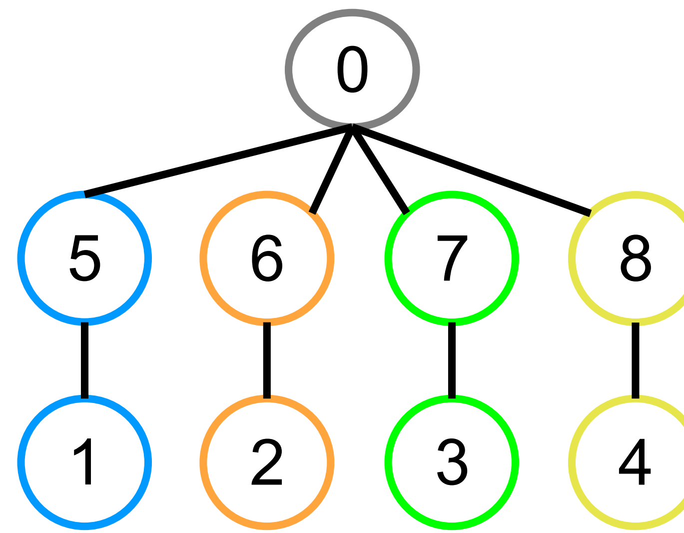
Topology Representation

- Root lgroup
  - contain all the lgroups in the system
  - highest level of latency
  - always lgroup 0



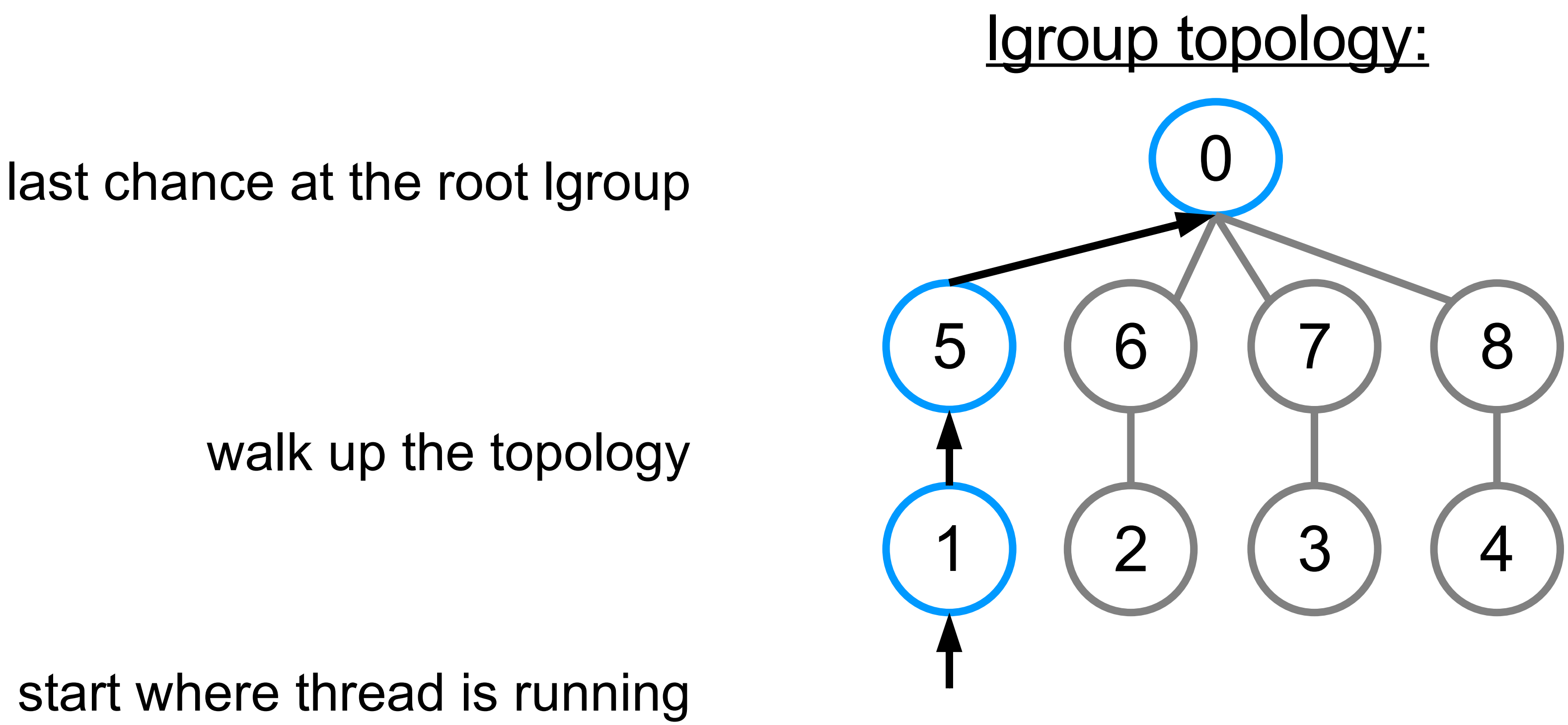
lgroup 0 (root)

lgroup topology:



Topology Representation

- Latency topology
  - used by the scheduler and VM subsystems
  - look for resources closer to the processor where the thread is running



Optimizing for NUMA

- Scheduler
  - threads are associated with a *home lgroup* upon creation
  - scheduler always tries to dispatch thread to a processor at the home lgroup
  - load balancing across CPUs nearby
  - use latency topology to dispatch to/steal from

Optimizing for NUMA

- VM
  - allocation policies:
    - first touch
      - allocate from the faulting thread's home lgroup
      - default for private memory
    - random
      - randomly across leaf lgroups
      - default for shared memory

Optimizing for NUMA

- liblgrp(3LGRP)
  - allows the developer to traverse the topology
  - discover the contents of an lgroup
  - change a thread's affinity for an lgroup
    - affinity != home lgroup
- meminfo(2)
- madvise(2)
- madv.so.1(1)
  - LD_PRELOAD = $LD_PRELOAD:madv.so.1
  - MADV= { normal, random, sequential, access_lwp, access_many, access_default }

Optimizing for NUMA

- lgrpinfo(1)
  - display information about the lgroup hierarchy
- plgrp(1)
  - observe and affect home lgroup and thread affinities
- pmadvise(1)
  - apply advices about memory to a process

Optimizing for NUMA

- lgrpinfo(1)
  - UMA machine

```
# lgrpinfo -Ta
0
|   CPUs: 0 1
|   Memory: installed 1015M, allocated 762M, free 253M
|   Lgroup resources: 0 (CPU); 0 (memory)
|   Load: 0.0493

Lgroup latencies:

-----
  | 0
-----
0 | 0
-----
```

Optimizing for NUMA

- lgrpinfo(1)
  - 2 node NUMA

```
# lgrpinfo -Ta
0
|-- 1
|    CPU: 0
|    Memory: installed 2.0G, allocated 168M, free 1.8G
|    Load: 0.951
|    Latency: 59
`-- 2
     CPU: 1
     Memory: installed 1.9G, allocated 265M, free 1.7G
     Load:    0
     Latency: 59

Lgroup latencies:

------------
  | 0  1  2
------------
0 | 96 96 96
1 | 96 59 96
2 | 96 96 59
------------
```

Optimizing for NUMA

- lgrpinfo(1)
  - 8 node NUMA (lgroup latencies matrix)

```
------------------------------------------------------------------------------------------------------
   |   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24
------------------------------------------------------------------------------------------------------
 0 | 185 185 147 147 147 147 147 147 185 185 185 185 185 185 185 185 147 185 147 185 185 185 185 185 185
 1 | 185  79 112 112 147 147 147 147 185 112 147 147 185 147 185 185 147 185 147 185 185 185 185 185 185
 2 | 147 112  79 147 112 147 147 112 147 147 147 112 147 147 147 147 147 147 147 147 147 147 147 147 147
 3 | 147 112 147  79 147 112 112 147 147 147 147 147 147 112 147 147 147 147 147 147 147 147 147 147 147
 4 | 147 147 112 147  79 112 112 147 147 147 147 147 147 147 147 147 112 147 147 147 147 147 147 147 147
 5 | 147 147 147 112 112  79 147 112 147 147 147 147 147 147 147 147 147 147 112 147 147 147 147 147 147
 6 | 147 147 147 112 112 147  79 147 112 147 147 147 147 147 147 147 147 147 147 147 112 147 147 147 147
 7 | 147 147 112 147 147 112 147  79 112 147 147 147 147 147 147 147 147 147 147 147 147 147 112 147 147
 8 | 185 185 147 147 147 147 112 112  79 185 185 185 185 185 185 185 147 185 147 185 147 185 147 147 112
 9 | 185 112 147 147 147 147 147 147 185 112 147 147 185 147 185 185 147 185 147 185 185 185 185 185 185
10 | 185 147 147 147 147 147 147 147 185 147 147 147 185 147 185 185 147 185 147 185 185 185 185 185 185
11 | 185 147 112 147 147 147 147 147 185 147 147 112 185 147 185 185 147 185 147 185 185 185 185 185 185
12 | 185 185 147 147 147 147 147 147 185 185 185 185 147 185 185 185 147 185 147 185 185 185 185 185 185
13 | 185 147 147 112 147 147 147 147 185 147 147 147 185 112 185 185 147 185 147 185 185 185 185 185 185
14 | 185 185 147 147 147 147 147 147 185 185 185 185 185 185 147 185 147 185 147 185 185 185 185 185 185
15 | 185 185 147 147 147 147 147 147 185 185 185 185 185 185 185 147 147 185 147 185 185 185 185 185 185
16 | 147 147 147 147 112 147 147 147 147 147 147 147 147 147 147 147 112 147 147 147 147 147 147 147 147
17 | 185 185 147 147 147 147 147 147 185 185 185 185 185 185 185 185 147 147 147 185 185 185 185 185 185
18 | 147 147 147 147 147 112 147 147 147 147 147 147 147 147 147 147 147 147 112 147 147 147 147 147 147
19 | 185 185 147 147 147 147 147 147 185 185 185 185 185 185 185 185 147 185 147 147 185 185 185 185 185
20 | 185 185 147 147 147 147 112 147 147 185 185 185 185 185 185 185 147 185 147 185 112 185 147 147 147
21 | 185 185 147 147 147 147 147 147 185 185 185 185 185 185 185 185 147 185 147 185 185 147 185 185 185
22 | 185 185 147 147 147 147 147 112 147 185 185 185 185 185 185 185 147 185 147 185 147 185 112 147 147
23 | 185 185 147 147 147 147 147 147 147 185 185 185 185 185 185 185 147 185 147 185 147 185 147 147 147
24 | 185 185 147 147 147 147 147 147 112 185 185 185 185 185 185 185 147 185 147 185 147 185 147 147 112
------------------------------------------------------------------------------------------------------
```

Topology discovery

- How does the system finds out about latencies and topology?
  - probe memory
  - probe PCI space registers (AMD Opteron)
  - ACPI 3.0
    - Static Resource Affinity Table (SRAT)
    - System Locality Information Table (SLIT)
    - default as of Solaris Nevada build 88
    - some gotchas ;)

A peek at the implementation

- UML Diagrams
  - MPO related routines
  - flow and brief description
  - based on snv65
  - work in progress
  - http://blogs.sun.com/rv/resource/browser/browser.html

A peek at the implementation

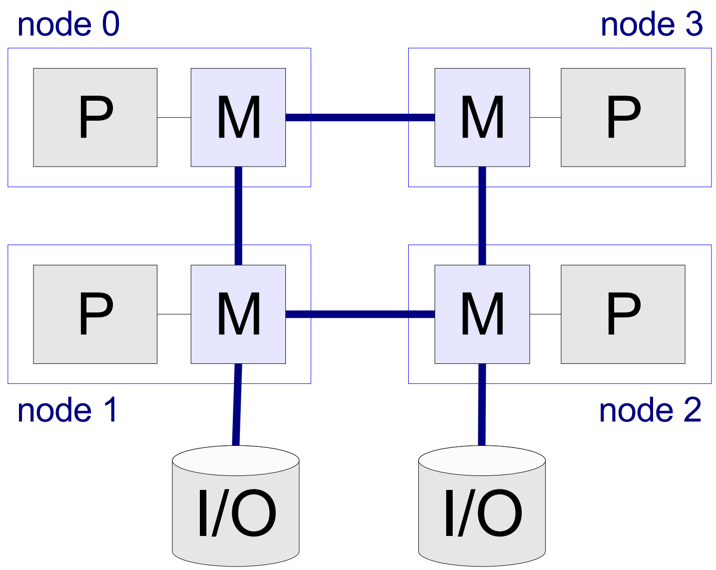
- Home lgroup
  - choosing a thread's home lgroup
    - lgrp_choose()
  - changing a thread's home lgroup
    - lgrp_move_thread()
- Scheduler
  - core routines under
    - usr/src/disp/disp.c

A peek at the implementation

- VM
  - how do we find an lgroup to allocate from ?
    - lgrp_mem_choose()
  - when allocating pages:
    - page_get_freelist()
    - page_get_cachelist()

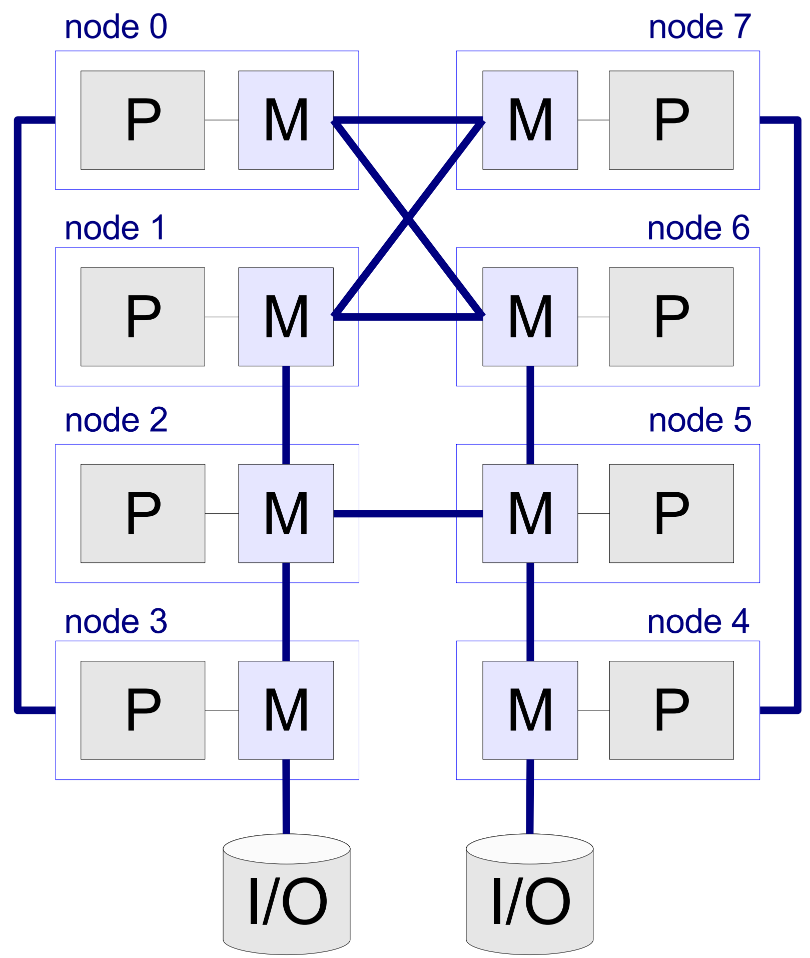
Where NUMA is going

- Virtualization
  - host is not NUMA aware
    - no optimizations
  - host is NUMA aware
    - exports NUMA characteristics to the guest OS
    - doesn't export, does everything by itself
- I/O locality
  - machines with NUMA I/O
  - where to place threads ?
    - app, kernel, driver
  - where to place memory ?
    - app, kernel, DMA buffers, device metadata



Where NUMA is going

- NUMA everywhere
  - Sun, AMD and Intel platforms
  - Sun, IBM, HP, SGI, Bull and others offer NUMA machines
- NUMA and CMT
  - Increased CMT density
- Different topologies
  - support and optimize
- Observability
  - hardware performance counters
- Power Management
  - performance x PM



Questions ?

Interested ?

- OpenSolaris Performance Community
  - http://opensolaris.org/os/community/performance/
  - perf-discuss-subscribe@opensolaris.org
- NUMA Project
  - http://opensolaris.org/os/community/performance/numa/
- Tesla Project
  - http://opensolaris.org/os/project/tesla/
  - tesla-dev-subscribe@opensolaris.org

開放的
열린
مفتوح
libre
मुक्त
ಮುಕ್ತ
livre
libero
ముక్త
开放的
açık
open
nyílt
פתוח
オープン
livre
ανοικτό
offen
otevřený
öppen
открытый
வெளிப்படை

open

Thank you!

Rafael Vanoni Polanczyk
Solaris Kernel Performance Group
rafael.vanoni@sun.com
blogs.sun.com/rv

“open” artwork and icons by chandan:
http://blogs.sun.com/chandan