OpenMP®

SC24 Booth Talk Series

NUMA:
Stay Close to Home with OpenMP

Ruud van der Pas
Senior Principal Software Engineer, Oracle Linux Engineering

What is NUMA?

Modern Times

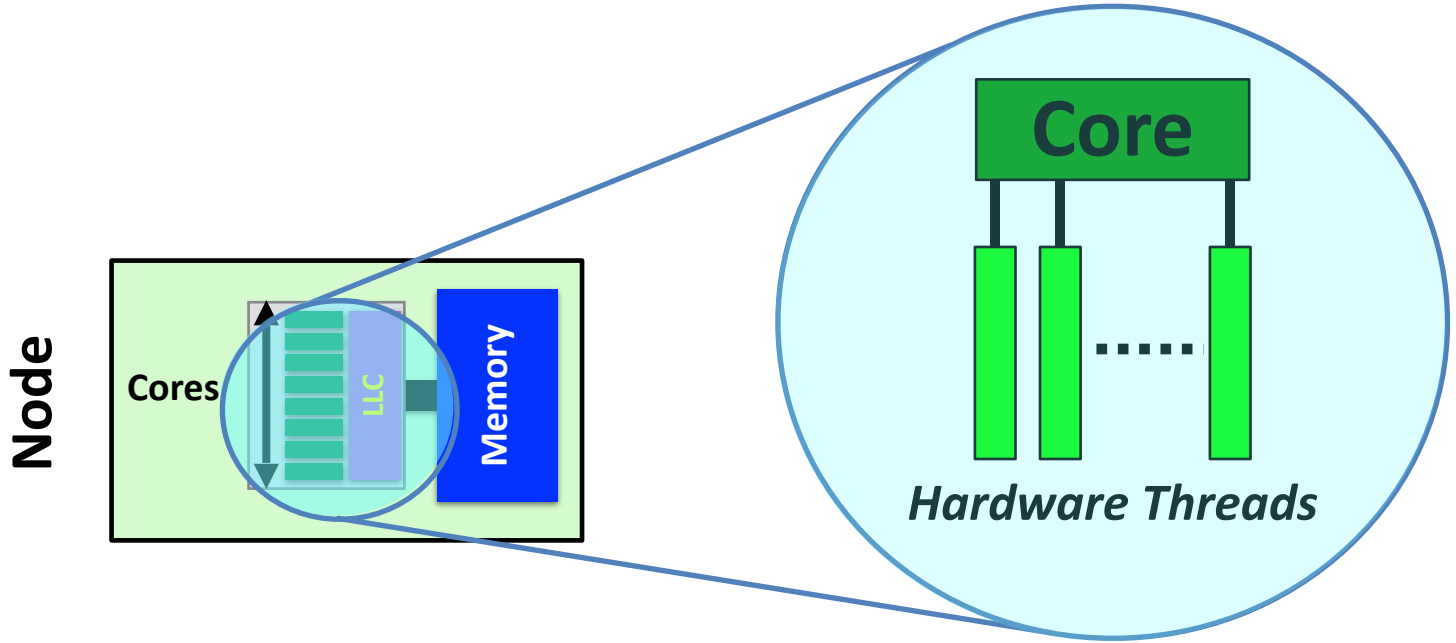
Non-Uniform Memory Access (NUMA) used to be the realm of large servers only

This is no longer true and therefore a concern to all

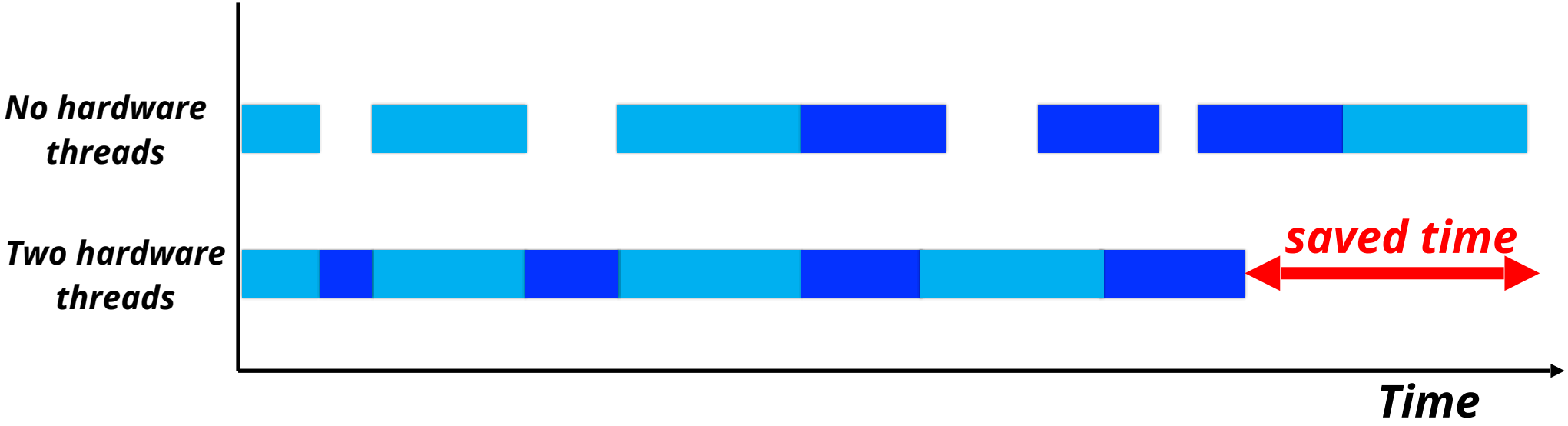
The tricky thing is that "things just work"

But do you know how efficiently your code performs?

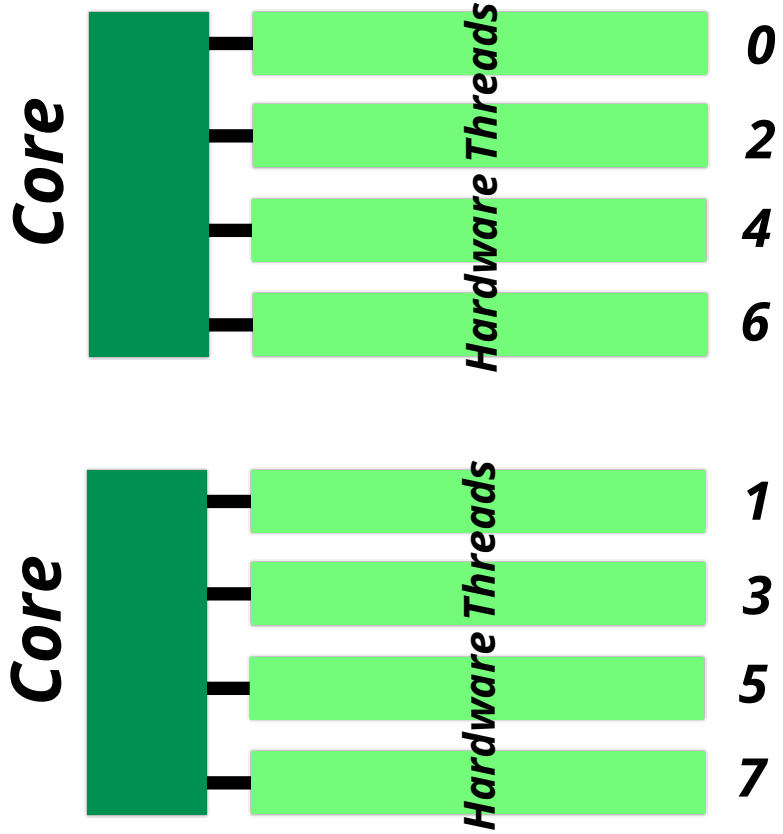
Intermezzo - Hardware Threads



Intermezzo - How Hardware Threads Work



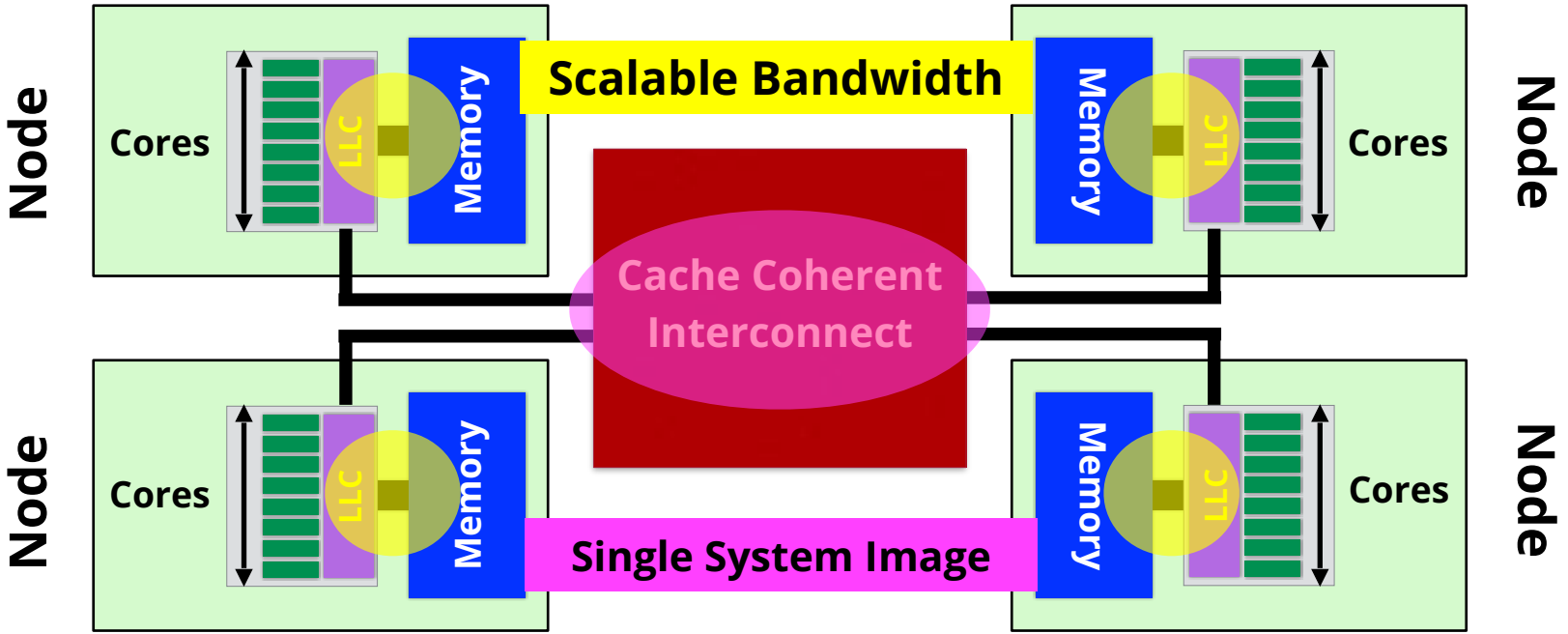
Intermezzo - Hardware Thread IDs



NUMA - The System Most of Us Use Today

OpenMP

A Generic, but very Common and Contemporary NUMA System



The NUMA View

OpenMP

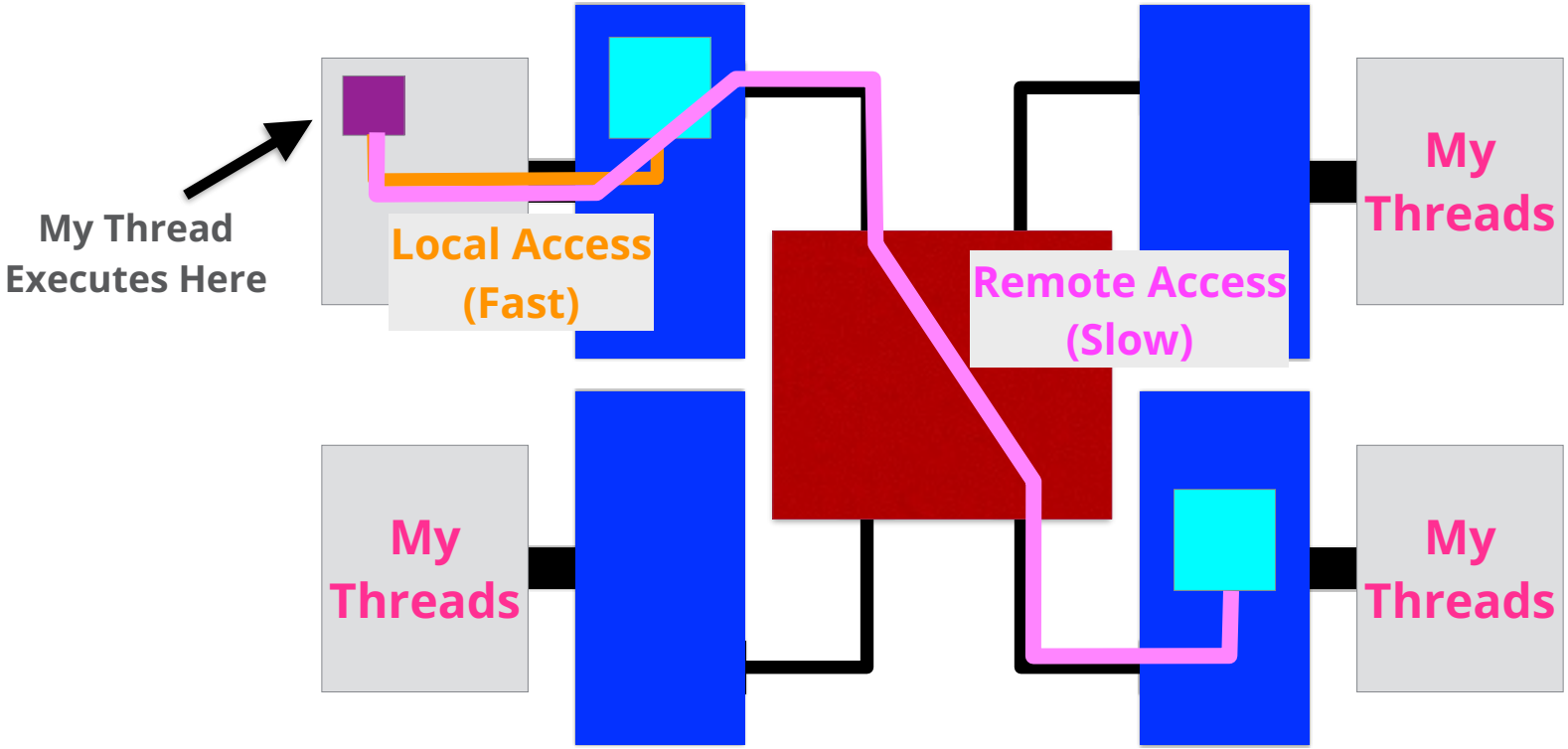
Memory is physically distributed, but logically shared

Shared data is accessible to all threads

You don't know where the data is and it doesn't matter

Unless you care about performance …

Local Versus Remote Access Times



Understanding Your System

The NUMA Information for a System

OpenMP

$ lscpu

8 cores/node

8 NUMA Nodes

```
      ......
NUMA node0 CPU(s):   0-7   ,  64-71
NUMA node1 CPU(s):   8-15  ,  72-79
NUMA node2 CPU(s):   16-23,   80-87
NUMA node3 CPU(s):   24-31,   88-95
NUMA node4 CPU(s):   32-39,   96-103
NUMA node5 CPU(s):   40-47,  104-111
NUMA node6 CPU(s):   48-55,  112-119
NUMA node7 CPU(s):   56-63,  120-127
      ......
```

2 columns => 2 hardware threads/core

$ numactl -H

```
node distances:
node   0   1   2   3   4   5   6   7
  0:  10  16  16  16  32  32  32  32
  1:  16  10  16  16  32  32  32  32
  2:  16  16  10  16  32  32  32  32
  3:  16  16  16  10  32  32  32  32
  4:  32  32  32  32  10  16  16  16
  5:  32  32  32  32  16  10  16  16
  6:  32  32  32  32  16  16  10  16
  7:  32  32  32  32  16  16  16  10
```

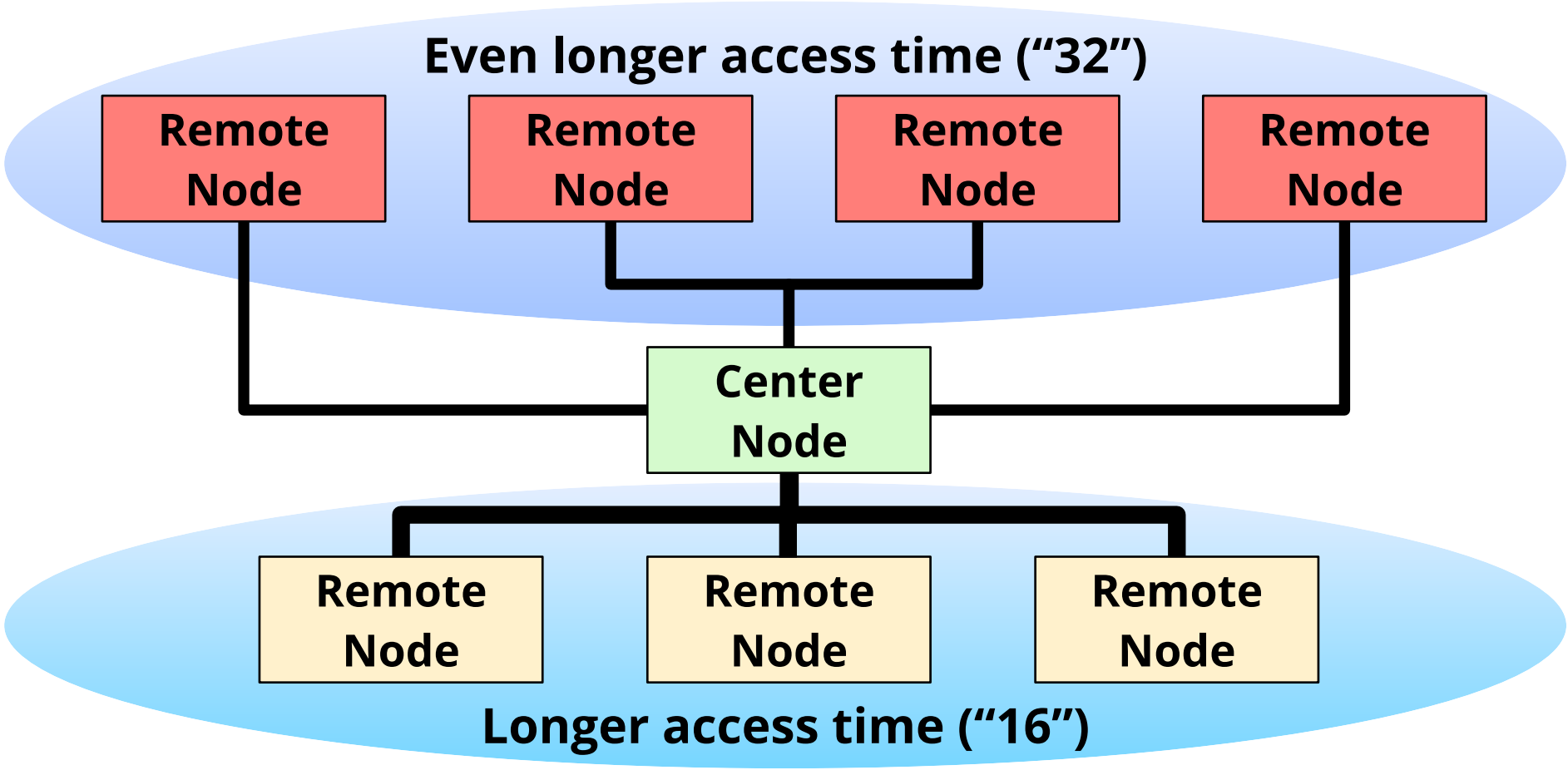
The NUMA Structure of the System

OpenMP

lscpu	There are 8 NUMA nodes
lscpu	There are 8 cores per node
lscpu	Each core has 2 hardware threads
numactl -H	Two levels of NUMA ("16" and "32")

This NUMA system has 64 cores and 128 hardware threads

The Abstract System Topology (numactl -H)



Tuning for a NUMA System

OpenMP

The Goal: keep threads and their data close

The data location is fixed, a thread may be moved closer to it

Not the other way round, because that is more expensive

The affinity constructs in OpenMP control where threads run

This is a powerful feature, but it is up to you to get it right
(in this context,"right" is not about correctness, but about the performance)

NUMA and Data Placement

The First Touch Data Placement Policy

OpenMP

Question: where does data get allocated then?

The First Touch Placement policy allocates the data page in the memory closest to the thread accessing this page for the first time

This defines the fixed Home Node for the particular page

Staying Close to Home

OpenMP

The goal of NUMA tuning

Keep threads close to their Home Node

The First Touch Data Placement Policy

OpenMP

This policy is the default on Linux and other OSes

It is the right thing to do for a sequential application

But this may not work so well in a parallel application

First Touch and Parallel Computing

OpenMP

What if a single thread initializes most, or all of the data?

Then, all the data ends up in the memory of a single node

This increases memory access times for certain threads (and may also cause congestion on the network)

Luckily, the solution is (often) surprisingly simple

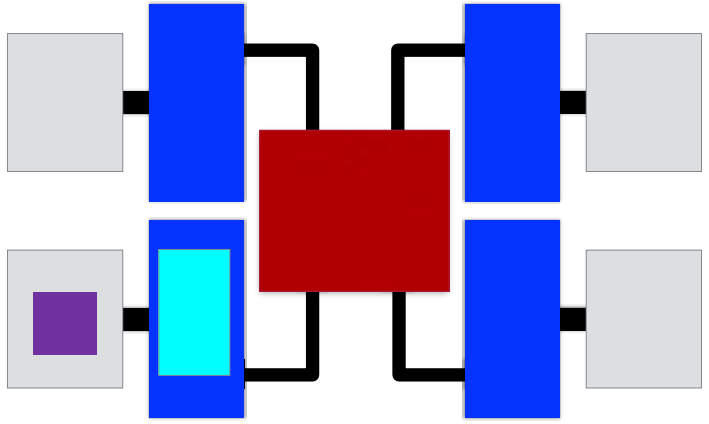
A Sequential Initialization

OpenMP

```
for (int64_t i=0; i<n; i++)
    a[i] = 0;
```

One thread executes this loop

All of "a" is in a single node



Note: The allocation is on a virtual memory page basis

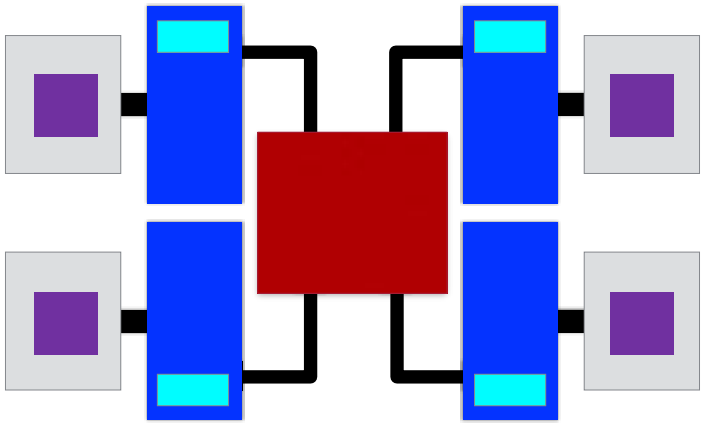
Leverage the First Touch Placement Policy

OpenMP

```
#pragma omp parallel for schedule(static)
for (int64_t i=0; i<n; i++)
    a[i] = 0;
```

Four threads execute this loop

The data is spread out



= Thread

= Data

Note: The allocation is on a virtual memory page basis

OpenMP Support for NUMA Systems

OpenMP Places and the Place List

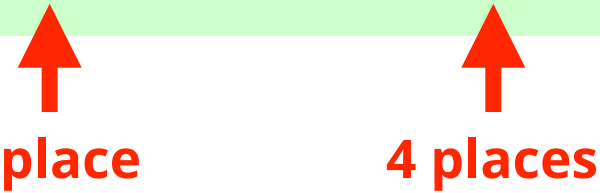
OpenMP

A place defines a set where a thread is allowed to run

The place list consists of a set of places

The place list consists of either symbolic names, or a set of numbers

- *An example of a symbolic name: cores*
- *An example of a set of numbers: {1, 3, 5, 7} or {1}, {3}, {5}, {7}*



The NUMA OpenMP Environment Variables

OpenMP

OMP_PLACES

Defines the places where threads may run

OMP_PROC_BIND

Defines how threads map onto the OpenMP places (relevant if there are more places than threads)

Placement Targets Supported by OMP_PLACES

Keyword	*Place definition*
threads	*A hardware thread*
cores	*A core*
ll_caches	*A set of cores that share the last level cache*
numa_domains	*A set of cores that share a memory with the same distance to that memory*
sockets	*A single socket*

Note: The number of places may be restricted - For example: cores(4)

Hardware Thread ID Support to Define Places

OpenMP

The OMP_PLACES variable also supports hardware thread IDs

Places can be defined using any sequence of valid numbers

A compact set notation is supported as well

Notation: {start:total:increment}

For example: {0:4:2} expands to {0,2,4,6}

Examples How to Use OMP_PLACES

OpenMP

Threads are scheduled on the NUMA domains in the system:

```
$ export OMP_PLACES=numa_domains
```

Use 4 places using hardware thread IDs 0, 8, 16, and 24:

```
$ export OMP_PLACES="{0},{8},{16},{24}"
$ export OMP_PLACES={0}:4:8
```

Use 1 place using hardware thread IDs 0, 8, 16, and 24:

```
$ export OMP_PLACES="{0,8,16,24}"
$ export OMP_PLACES={0:4:8}
```

Map Threads onto Places

OpenMP

Use variable OMP_PROC_BIND to map threads onto places

The settings define the mapping of threads onto places

The following settings are supported:
true, false, primary, close, or spread

The definitions of close and spread are in terms of the place list

An Example Using Places and Binding

OpenMP

Threads are scheduled on the cores in the system:

```
$ export OMP_PLACES=cores
```

And they should be placed on cores as far away from each other as possible:

```
$ export OMP_PROC_BIND=spread
```

Remember this Example?

OpenMP

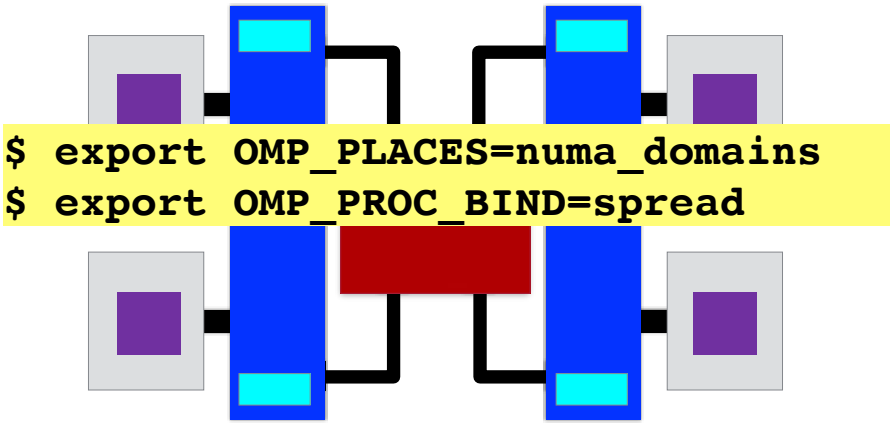
```
#pragma omp parallel for schedule(static)
for (int64_t i=0; i<n; i++)
    a[i] = 0;
```

Four threads execute this loop

Wishful Thinking

Data placement depends on where threads execute

Use the NUMA Controls



= Thread

= Data

Important: Use the NUMA Diagnostics!

OpenMP

It is very easy to make a mistake with the NUMA setup
Two very simple, but yet powerful features to assist you
Variable OMP_DISPLAY_ENV echoes the initial settings
Variable OMP_DISPLAY_AFFINITY prints information at run time
Highly recommended to use these diagnostic features!

Thank You And … Stay Tuned!

Bad OpenMP
Does Not Scale

Ruud van der Pas

SC24 OpenMP Booth Talk

OpenMP®

SC24 Booth Talk Series

openmp.org — OpenMP API specs, forum, reference guides, and more

link.openmp.org/sc24talks — OpenMP SC24 booth talk videos and presentations