How To Befriend NUMA

Ruud van der Pas
Senior Principal Software Engineer
Oracle Linux and Virtualization Engineering

OpenMP Booth Talk, Tuesday November 13, 2018

SC18, Dallas, TX, USA

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

- What is Oracle Linux?
- A Generic Contemporary NUMA System
- About NUMA and Data Placement
- OpenMP Support for NUMA Systems
- A Performance Tuning Example
- Conclusions

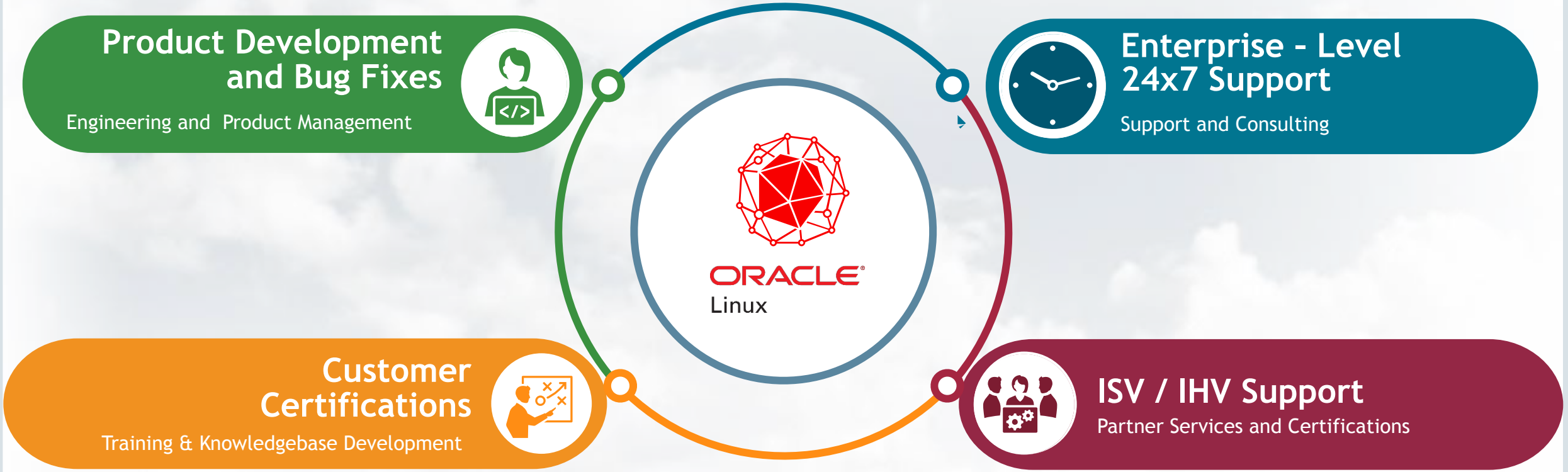
What is Oracle Linux?

Oracle Linux

Oracle Linux

- Shipping for more than 11 years
- Maintains Application Compatibility with RHEL
- 100% Binary Compatible Kernel; Oracle supplies patches and updates
- Powers Oracle Cloud & Engineered Systems
- Tens of thousands of enterprises supported
- Over 1 million Docker hub downloads
- Linux Foundation Platinum board member
- Cloud Native Computing Foundation Platinum member

Oracle is a Complete Full-Service Linux Vendor



Contribute Back to the Community

Oracle is an active contributor to multiple open source projects, including kernel.org

- One of the largest Linux engineering teams in the industry
- Numerous lead Linux maintainers --Linux Security, iSCSI, XFS, NFS Client, Open vSwitch …
- Linux Foundation Platinum board member
- Cloud Native Computing Foundation (CNCF) Platinum member

Focus on contributing enterprise features

- Xen / KVM
- Btrfs, XFS, etc.
- Linux Data Integrity Project (T10 DIF)
- Linux Test Project (LTP)

Just part of Oracle's extensive open source offerings

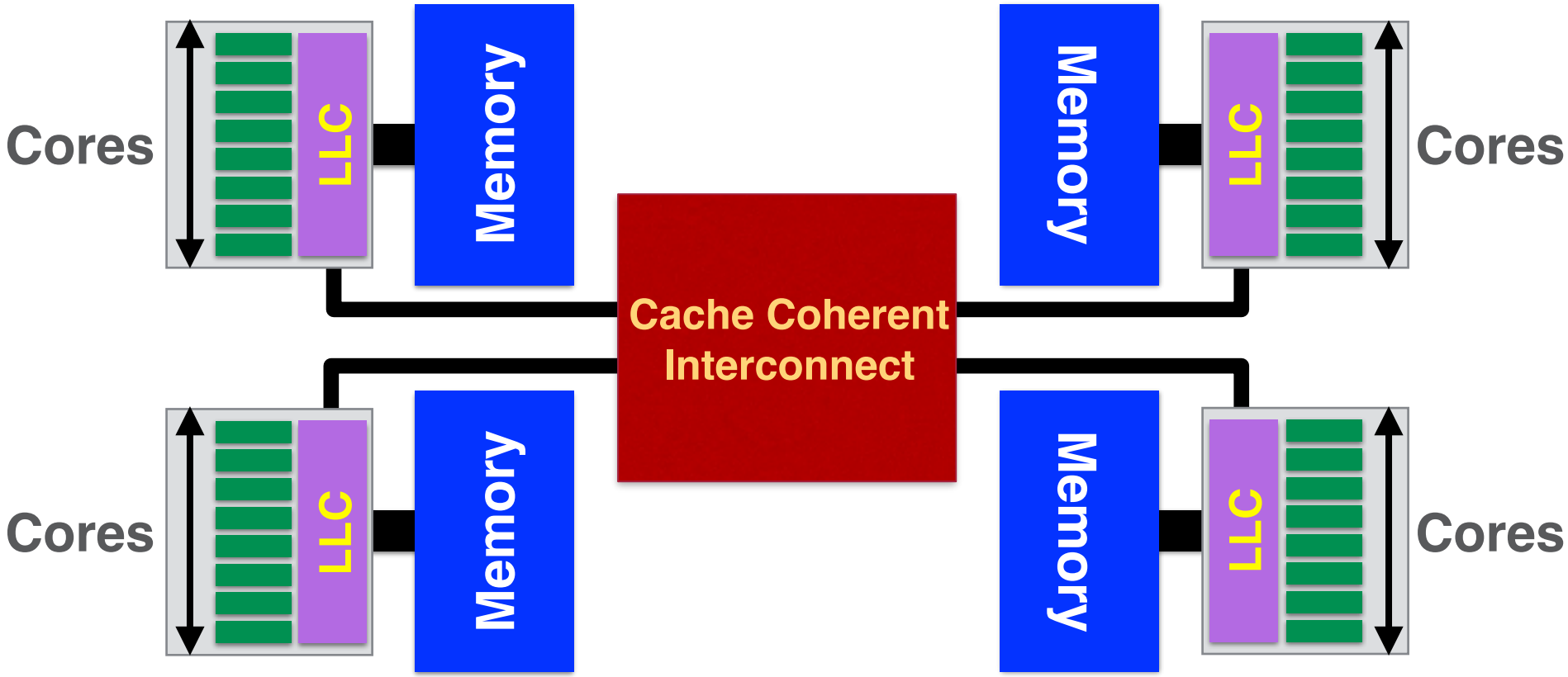
Download Oracle Linux!

https://www.oracle.com/technetwork/server-storage/linux/downloads/index.html

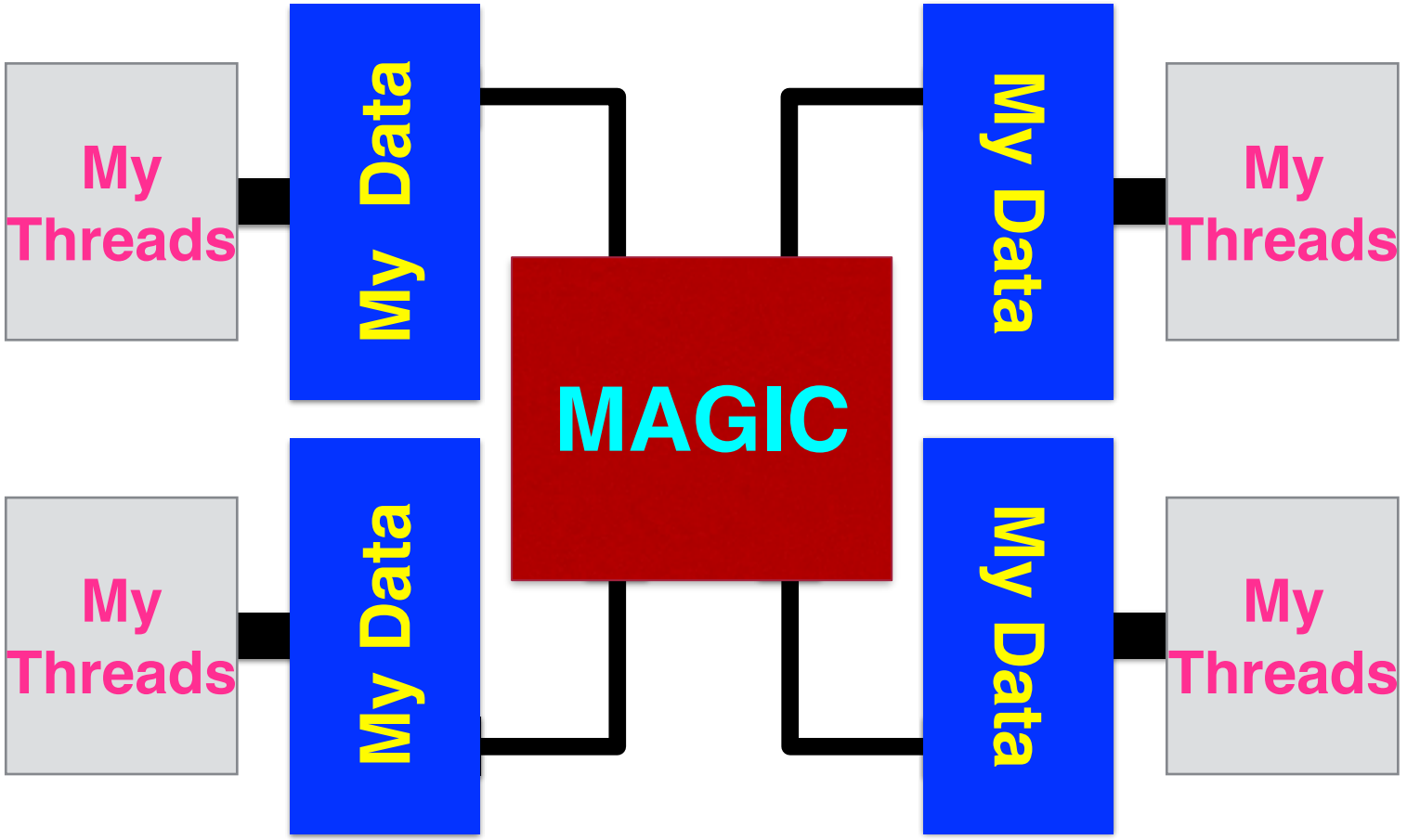
https://github.com/oracle/linux-uek

A Generic Contemporary NUMA System

A Generic Contemporary NUMA System



The Developer's View



The NUMA View

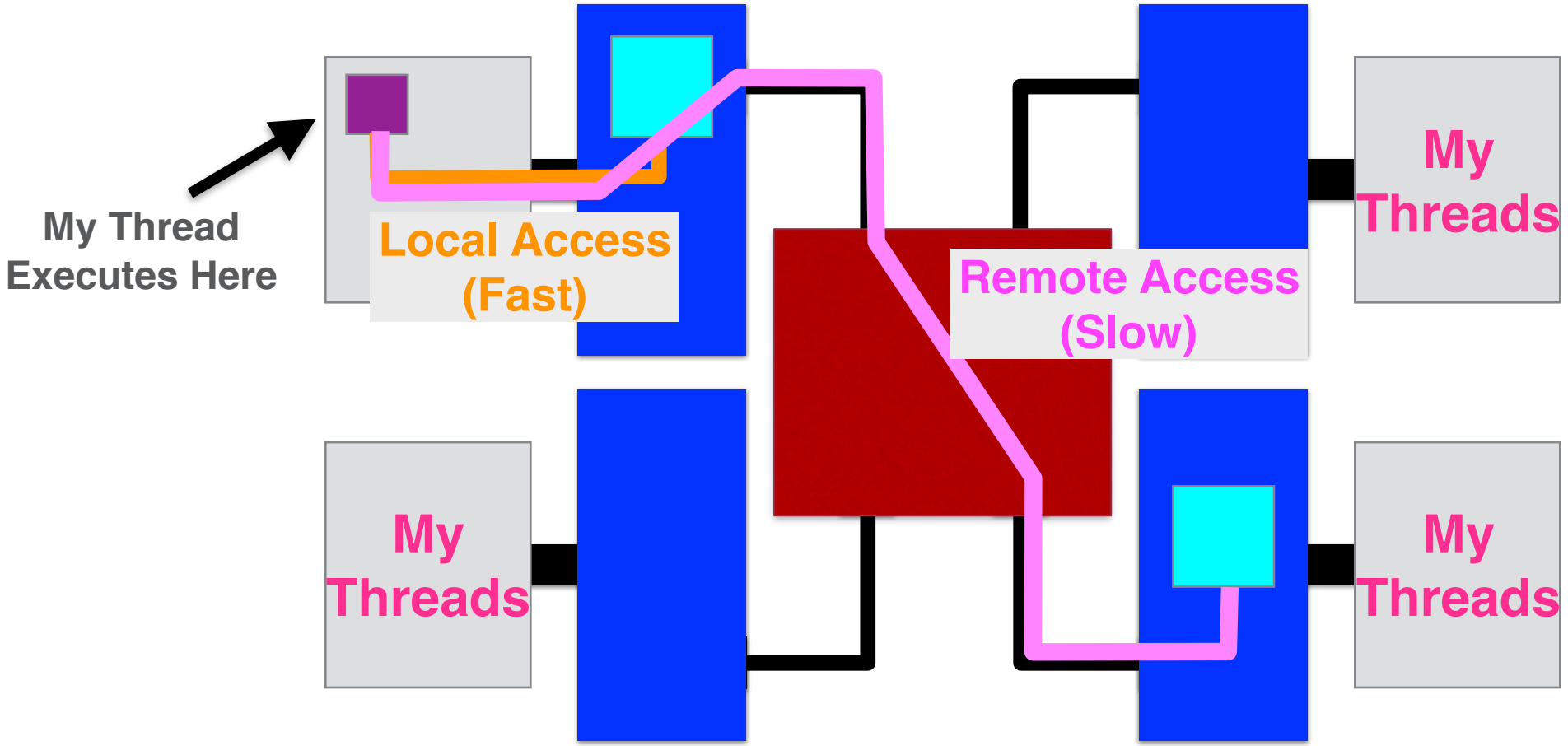
Memory is physically distributed, but logically shared

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



Terminology - Hardware Thread IDs and Strands

A Hardware Thread is also called a "strand"

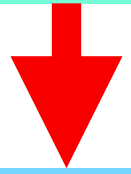
To avoid confusion with (OpenMP) threads, we use "strand" for a Hardware Thread

Each strand has a unique ID
(plus a certain amount of hardware "state")

Use the "lscpu" tool in Linux to see them

Example “lscpu” output*

```
$ lscpu
…… <lines removed> ……
NUMA node0 CPU(s): 0-25,52-77
NUMA node1 CPU(s): 26-51,78-103
```



There are two NUMA nodes: 0 and 1

Each NUMA node has 26 cores

Each core has 2 strands (e.g. {0,52} or {51,103})

**) Other information shown has been omitted here*

More NUMA Details With “numactl -H”

```
available: 2 nodes (0-1)
node 0 cpus: 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 25 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77
node 0 size: 385386 MB
node 0 free: 384584 MB
node 1 cpus: 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
51 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103
node 1 size: 387061 MB
node 1 free: 386796 MB

node distances:
node   0   1
  0:  10  21
  1:  21  10
```

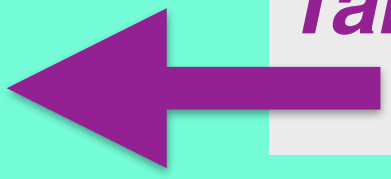
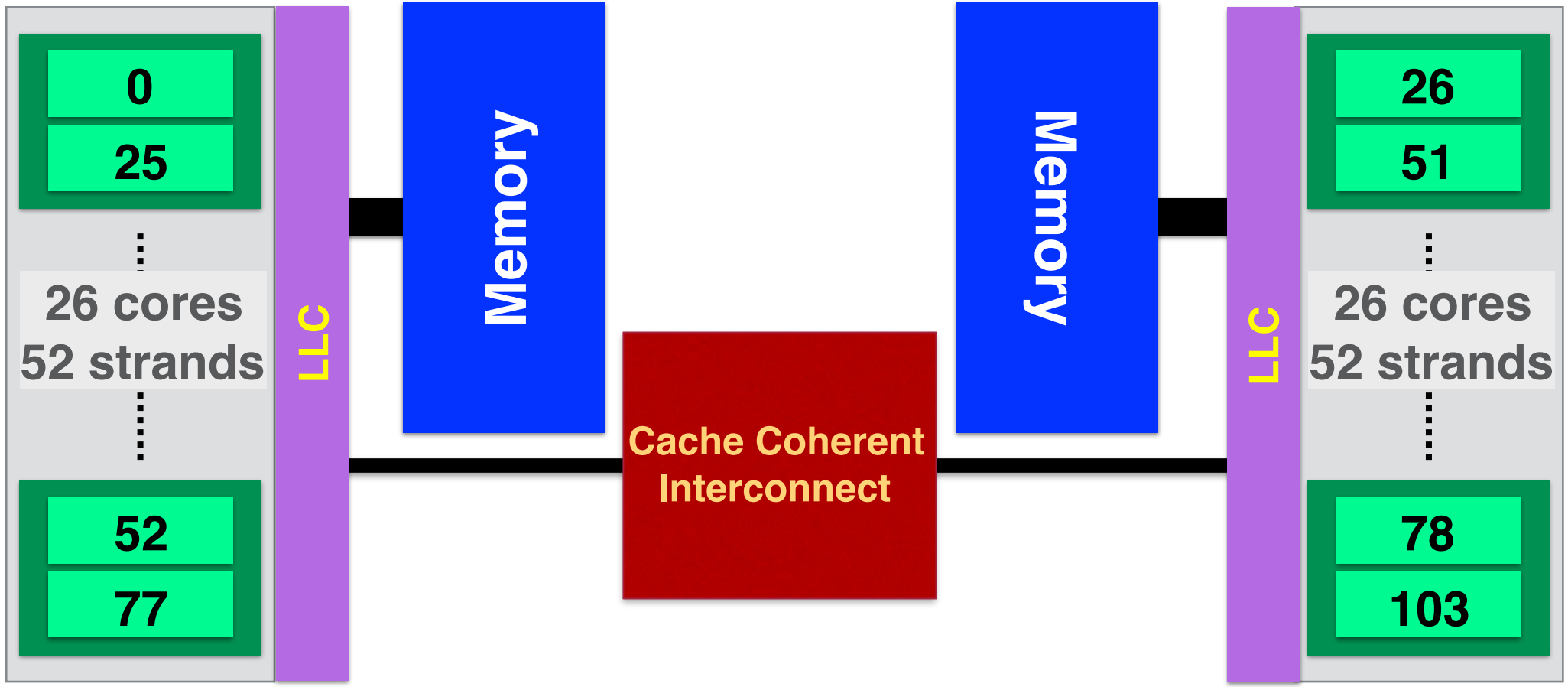


Table with relative latencies (normalized to 10)

This Is The Underlying System - 52 cores, 104 strands



About NUMA and Data Placement

The First Touch Data Placement Policy (“First Touch”)

So 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 policy is the default on Linux and other OSes

And makes sense because it is the right thing to do for a sequential application

First Touch And Parallel Computing

First Touch works fine, but 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 access times for certain threads and may cause congestion at the memory controller

Luckily, the solution is (often) surprisingly simple

How To Leverage First Touch

Parallelize the data initialization part!

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

Now, each thread has a slice of "a" in its local memory

OpenMP Support for NUMA Systems

OpenMP Support For Thread Affinity

Philosophy:

- ***The data is where it is***
- ***Move a thread to the data it needs most***

There are two environment variables to control this

The Affinity Related OpenMP Environment Variables

OMP_PLACES

Defines where threads may run

OMP_PROC_BIND

Defines how threads map onto the OpenMP places

Note: Highly recommended to also set OMP_DISPLAY_ENV=verbose

An Example

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
```

The OMP_PLACES Environment Variable

Value	Definition
sockets [<n>]	***Threads are scheduled on sockets***
cores [<n>]	***Threads are scheduled on cores***
threads [<n>]	***Threads are scheduled on strands***
“user defined set”	***Use strand IDs to schedule threads***

Examples OMP_PLACES

Threads are scheduled on the sockets in the system:

```
$ export OMP_PLACES=sockets
```

Use Strand IDs 0, 8, 16, and 24:

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

The OMP_PROC_BIND Environment Variable

Value	Definition
master	*Schedule threads in the same place where the master thread is executing*
close	*Keep threads "close" in terms of the places*
spread	*Spread threads as far as possible in terms of the places*

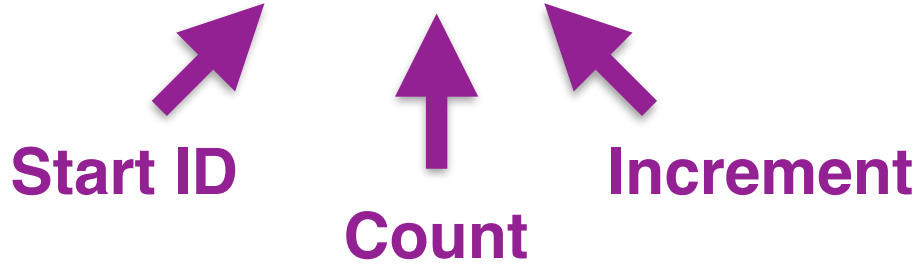
Examples How To Define Places On This System

The first strand on each core in the first socket:

```
$ export OMP_PLACES="{0},{1},{2},….,{25}"
```

Using a more compact notation:

```
$ export OMP_PLACES="{0}:26:1"
```




```
NUMA node0 CPU(s): 0-25,52-77
NUMA node1 CPU(s): 26-51,78-103
```

Another Example

The first strand in the first four cores in socket 0, and the second strand in the first four cores in socket 1

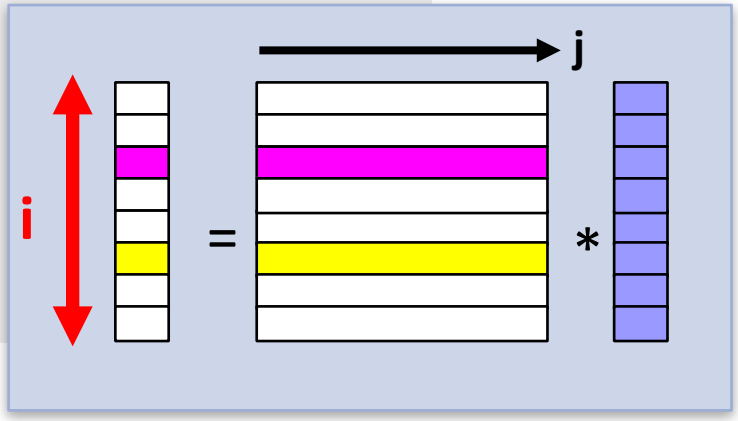
```
$ export OMP_PLACES="{0}:4:1,{78}:4:1"
```

```
NUMA node0 CPU(s): 0-25,52-77
NUMA node1 CPU(s): 26-51,78-103
```

A Performance Tuning Example

The Matrix Times Vector Test Code

```
#pragma omp parallel for default(none) \
        shared(m,n,a,b,c)
for (int i=0; i<m; i++)
{
  double sum = 0.0;
  for (int j=0; j<n; j++)
     sum += b[i][j]*c[j];
  a[i] = sum;
}
```



The System Used

AMD EPYC server with 2 sockets

Consists of 2*4 = 8 NUMA nodes according to “lscpu”

Each NUMA node has 8 cores with 2 strands each

In total 64 cores and 128 strands

The NUMA Nodes Of The System

```
$ lscpu
        ......
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
        ......
$
```

```
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 OpenMP Affinity Setup*

Threads are evenly distributed across the cores and nodes

For example:

Use the first two strands in each NUMA node of the system

The next slide shows how to do this in OpenMP

**) For this kind of CPU intensive algorithm, the second strand is not meaningful to use*

Example Using OpenMP Affinity

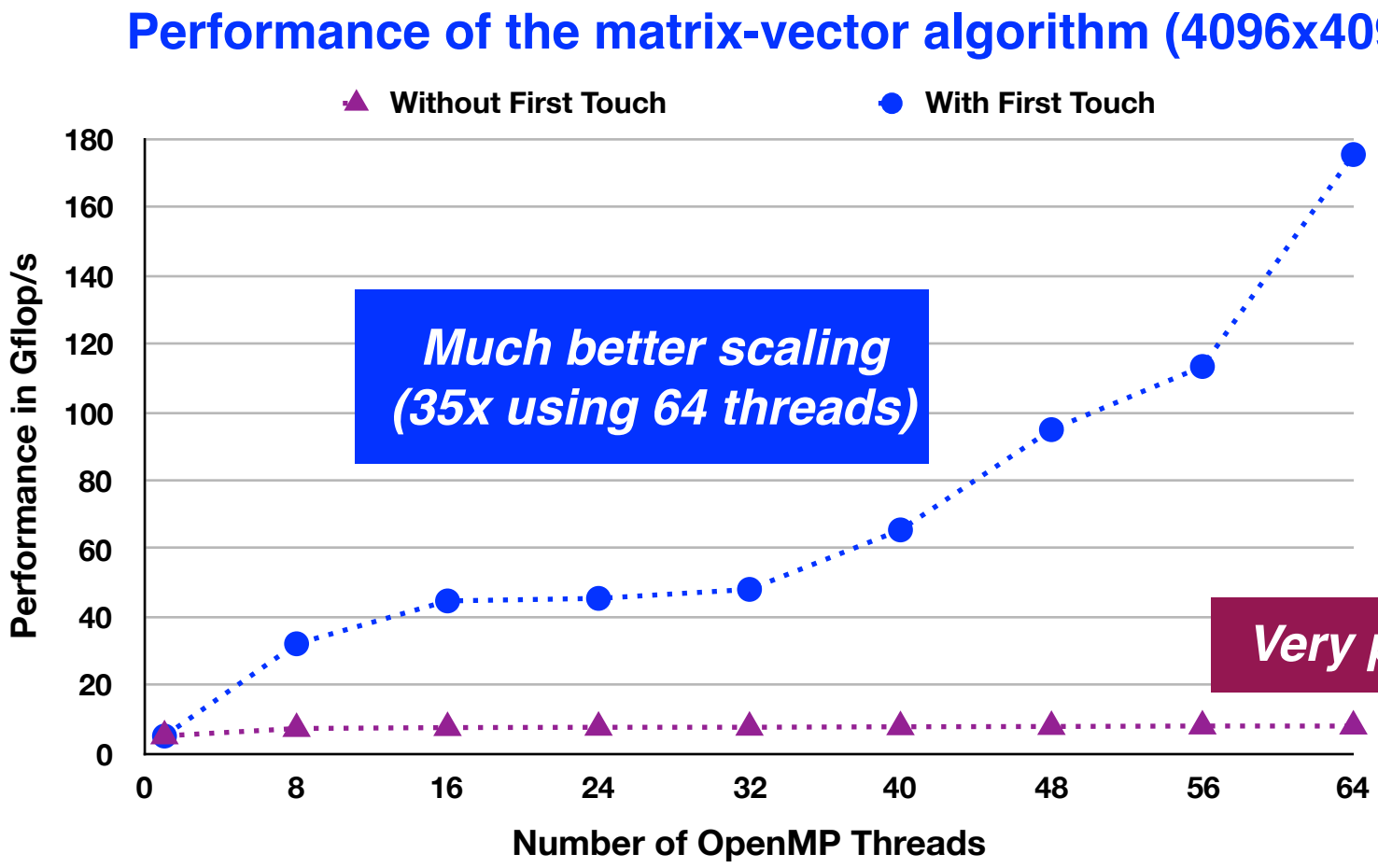
```
$ OMP_PLACES={0}:2:1,{8}:2:1,{16}:2:1,{24}:2:1
$ OMP_PLACES+=,{32}:2:1,{40}:2:1,{48}:2:1,{56}:2:1
$ export $OMP_PLACES

$ export OMP_PROC_BIND=close

$ export OMP_NUM_THREADS=16

$ ./a.out
```

The Performance Using Two Sockets (64 cores)



First Touch improves the performance by a factor of 22x

2 * AMD EPYC 7551 32 Core Processor
Oracle Linux 4.14.35-1821.el7uek.x86_64

Conclusions

Data and thread placement matters (a lot)

Important to leverage First Touch Data Placement

OpenMP has elegant, yet powerful, support for NUMA

And more support has been added in OpenMP 5.0!

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Thank You And … Stay Tuned!

ruud.vanderpas@oracle.com