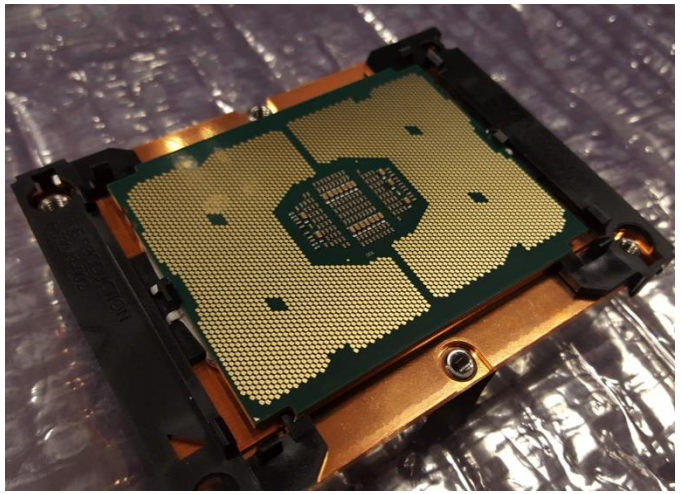
TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

The University of Texas at Austin

# Intel Knights Landing (KNL)

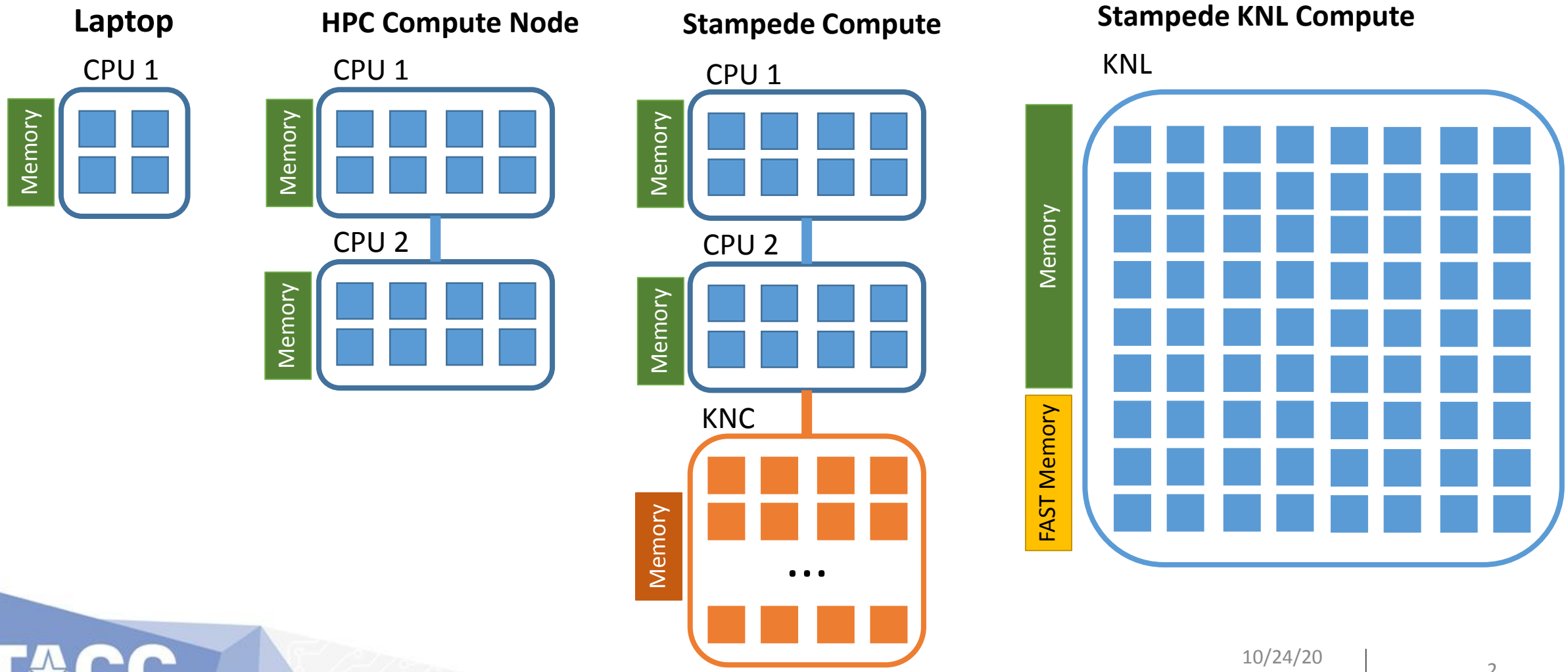


PRESENTED BY:

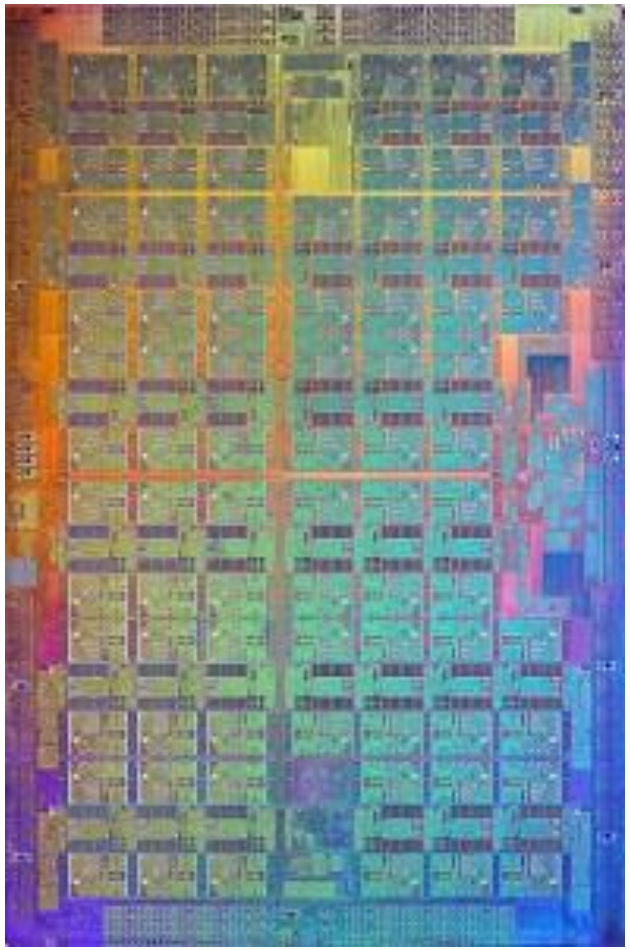
HPC Group

TACC

# From Laptop to Next-Gen Supercomputer



# 2nd Generation Intel Xeon Phi Knights Landing



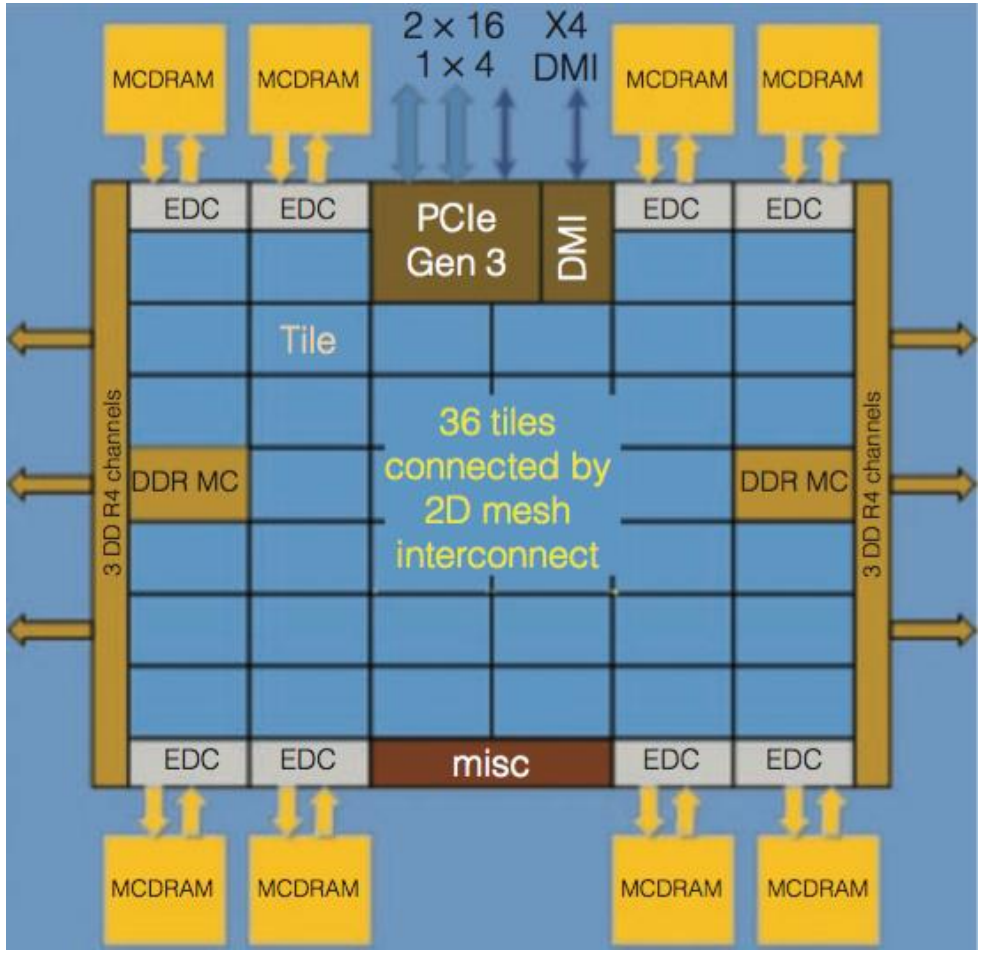
- Many Integrated Cores (MIC) architecture
- Up to 72 cores (based on Silvermont)
  - 4 H/W threads per core
  - Possible 288 threads of execution
- 16 GB MCDRAM* (high bandwidth) on-package
- 1 socket – self hosted (no more PCI bottleneck!)
- 3+ TF DP peak performance
- 6+ TF SP peak performance
- 400+ GB/s STREAM performance
- Supports Intel Omni-Path Fabric

* Multi-Channel DRAM

# Knights Corner → Knights Landing

| KNC | KNL |
|---|---|
| **Co-processor** | **Self hosted** |
| Stripped down Linux | CentOS 7 |
| Binary incompatible with other architectures | **Binary compatible** with prior Xeon (non Phi) architectures |
| 1.1 GHz processor | 1.4 GHz processor |
| Up to 16 GB RAM | Up to 400 GB RAM (including 16 GB MCDRAM) |
| 22 nm process | 14 nm process |
| 1 512-bit VPU | **2 512-bit VPUs** |
| No support for:<br>• Out of order<br>• Branch prediction<br>• Fast unaligned memory access | Support for:<br>• Out of order<br>• Branch prediction<br>• Fast unaligned memory access |

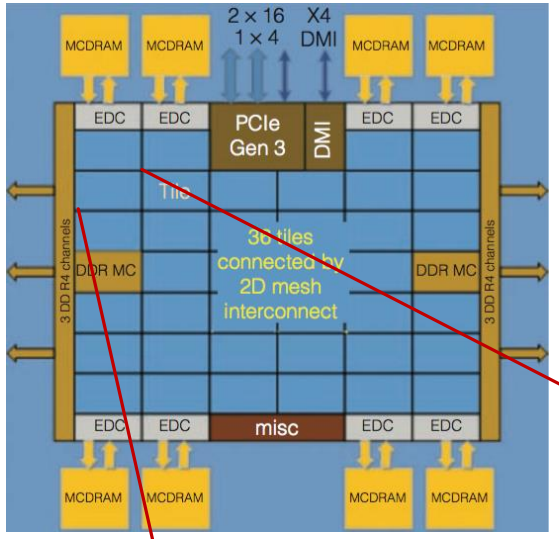
# KNL Diagram



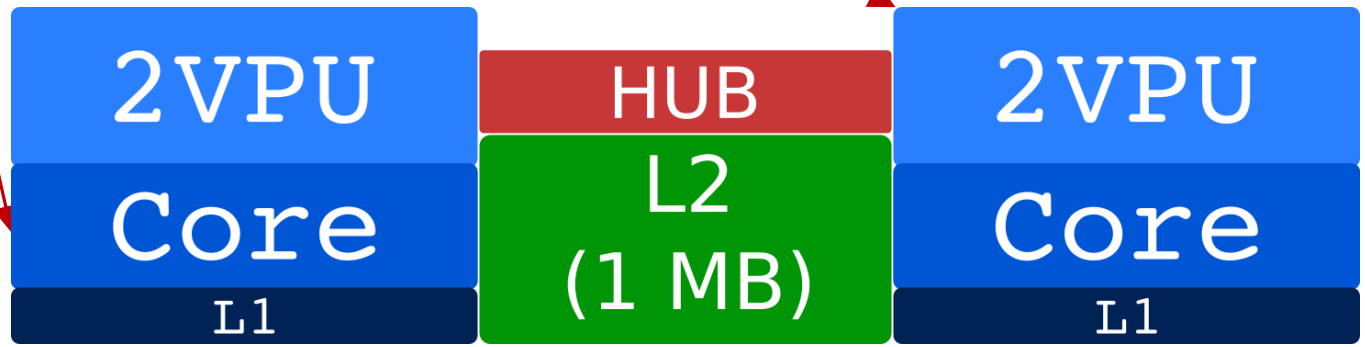

- Cores are grouped in pairs (tiles)
- 34 tiles (68 cores)
- 2D mesh interconnect
- 2 DDR memory controllers
- 6 channels DDR4
  - Up to 90 GB/s
- 16 GB MCDRAM
  - 8 embedded DRAM controllers
  - Up to 475 GB/s

(*KNIGHTS LANDING: SECOND- GENERATION INTEL XEON PHI PRODUCT*, A. Sodani, et.al.,IEEE Micro March/April 2016)

# KNL Tile




- Each core (based on Intel Silvermont):
  - Local L1 cache
  - 2 512-bit VPUs (almost symmetric)
- 2 cores/tile
  - 1 MB shared L2 cache (up to 36 MB L2 per KNL)
- Shared mesh connection (mesh at the tile level, not core level)



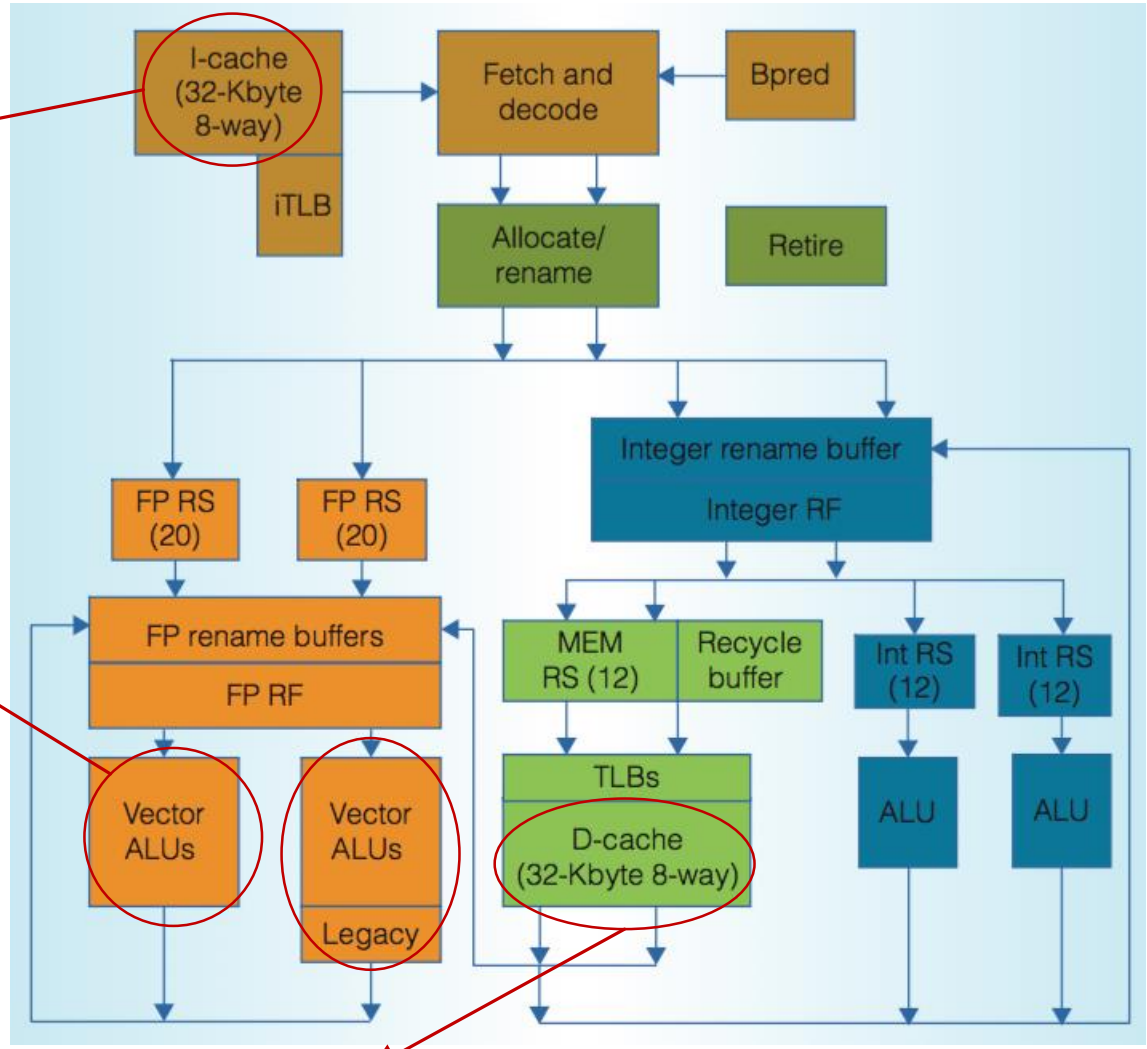

(*KNIGHTS LANDING: SECOND- GENERATION INTEL XEON PHI PRODUCT*, A. Sodani, et.al.,IEEE Micro March/April 2016)

# KNL Core

8-way 32KB instruction cache

2 VPUs, only one has support for legacy floating point ops

- Compile with **-xMIC-AVX512** to use both VPUs *(flag supported by Intel compiler)*

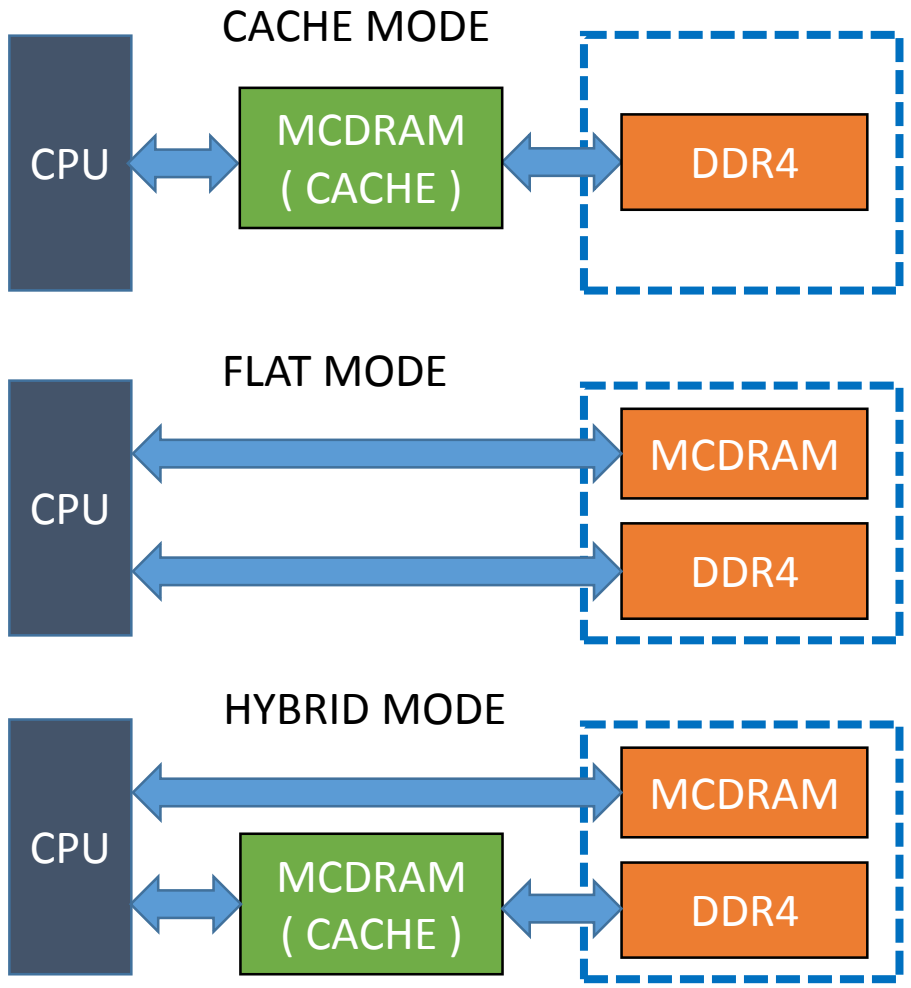



8-way 32KB data cache

(*KNIGHTS LANDING: SECOND- GENERATION INTEL XEON PHI PRODUCT*, A. Sodani, et.al.,IEEE Micro March/April 2016)

# Memory Architecture

- Two main memory types
  - DDR4
  - MCDRAM
- Three memory modes
  - Cache
  - Flat
  - Hybrid
- Hybrid mode
  - Three choices
  - 25% / 50% / 75%



# Cluster Modes

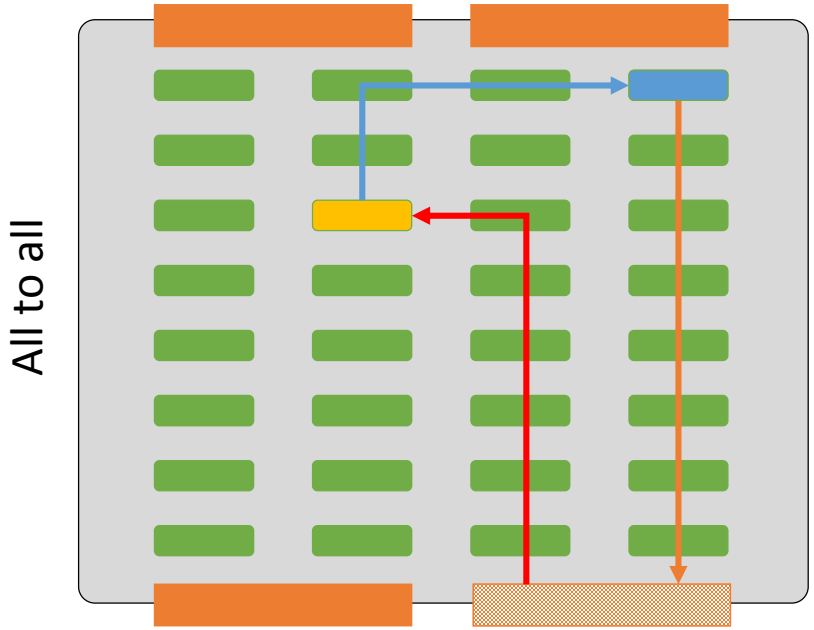
## Cluster Modes determine L2 coherency traffic flow

- Three supported modes
  - All-to-all (A2A)
  - Quadrant (Quad)
  - Sub-NUMA Cluster (SNC)

- Cluster modes modify the distance that coherency traffic flows go through in the mesh

# Clustering Mode: All to all

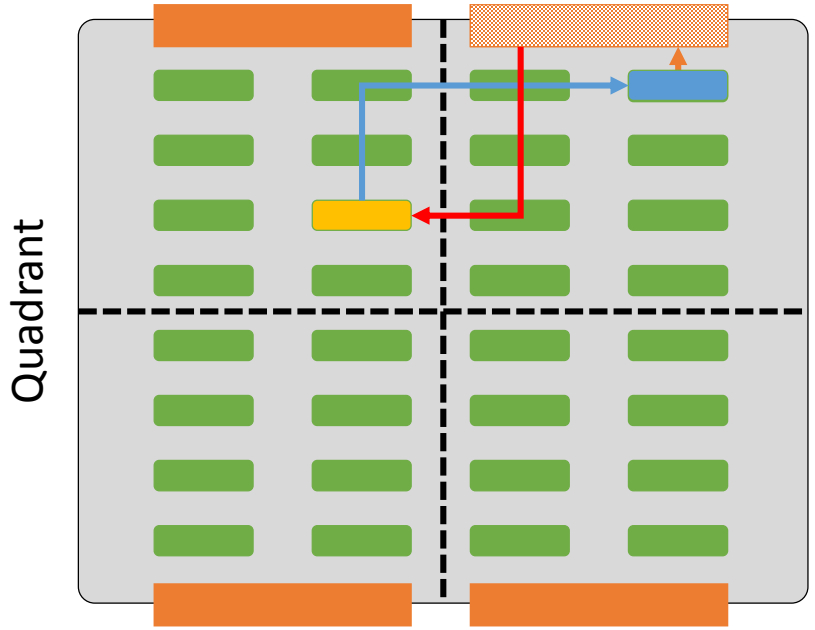
1. L2 Miss: data not in local L2
2. Directory access: look for tag in directory
3. Memory access: look for data in memory
4. Data: send data to original tile




- No affinity between tile, directory, and memory
- An L2 miss may have to traverse the mesh in order to read the required 64-byte line from memory
- This should provide the lowest throughput of all clustering modes

# Clustering Mode: Quadrant

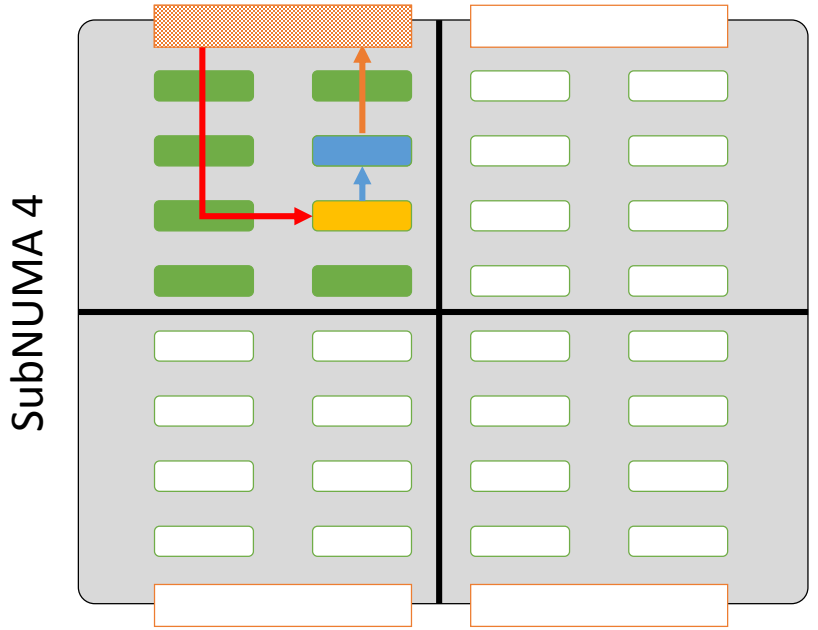
1. L2 Miss: data not in local L2
2. Directory access: look for tag in directory
3. Memory access: look for data in memory
4. Data: send data to original tile




- KNL Chip divided in 4 quadrants
- No affinity between tile and directory
  - When there is an L2 miss in a tile the directory tag may be anywhere on the chip
- Affinity between directory and memory
  - Data associated to a directory tag will be in the same quadrant that the directory tag is located
- Early testing shows this mode provides the best memory bandwidth

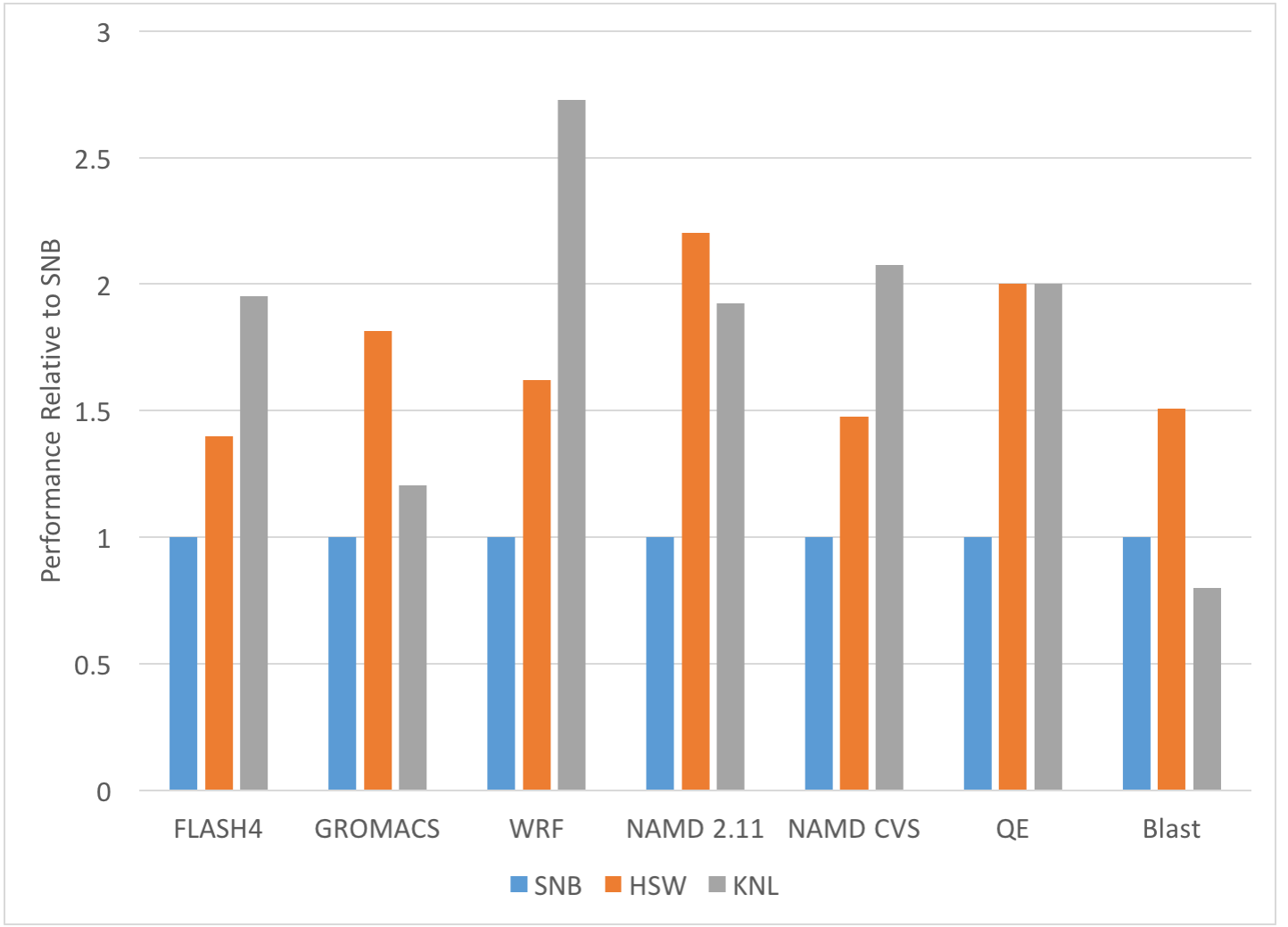
# Clustering Mode: Sub-NUMA 4

1. L2 Miss: data not in local L2
2. Directory access: look for tag in directory
3. Memory access: look for data in memory
4. Data: send data to original tile



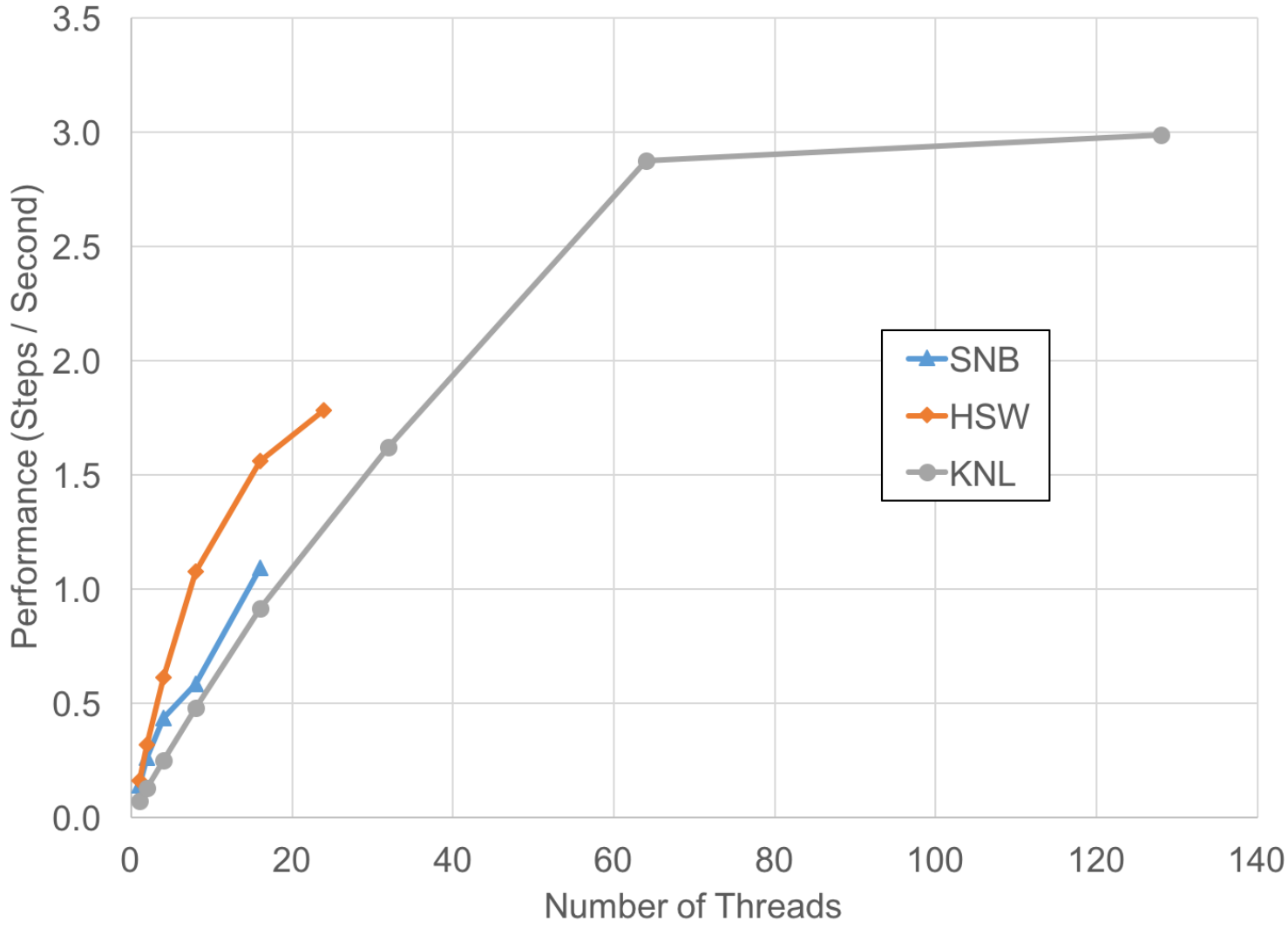

- Affinity between tile and directory
  - When there is an L2 miss in a tile the directory tag will be in the same quadrant
- Affinity between directory and memory
  - Data associated to a directory tag will be in the same quadrant that the directory tag is located
- Two options: **SNC-2 / SNC-4**
- Potentially highest memory throughput for properly configured runs

# Performance vs Stampede & Lonestar5



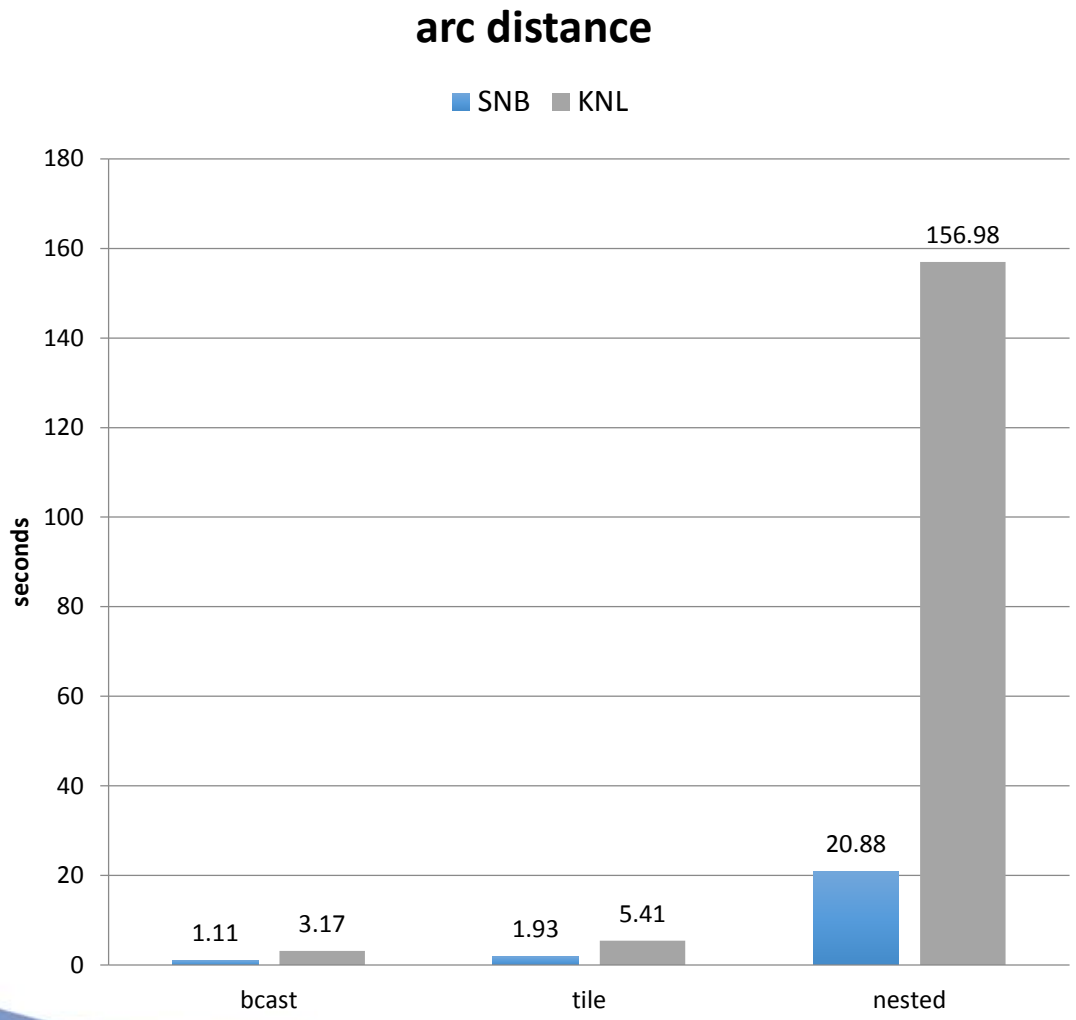

(higher is better)

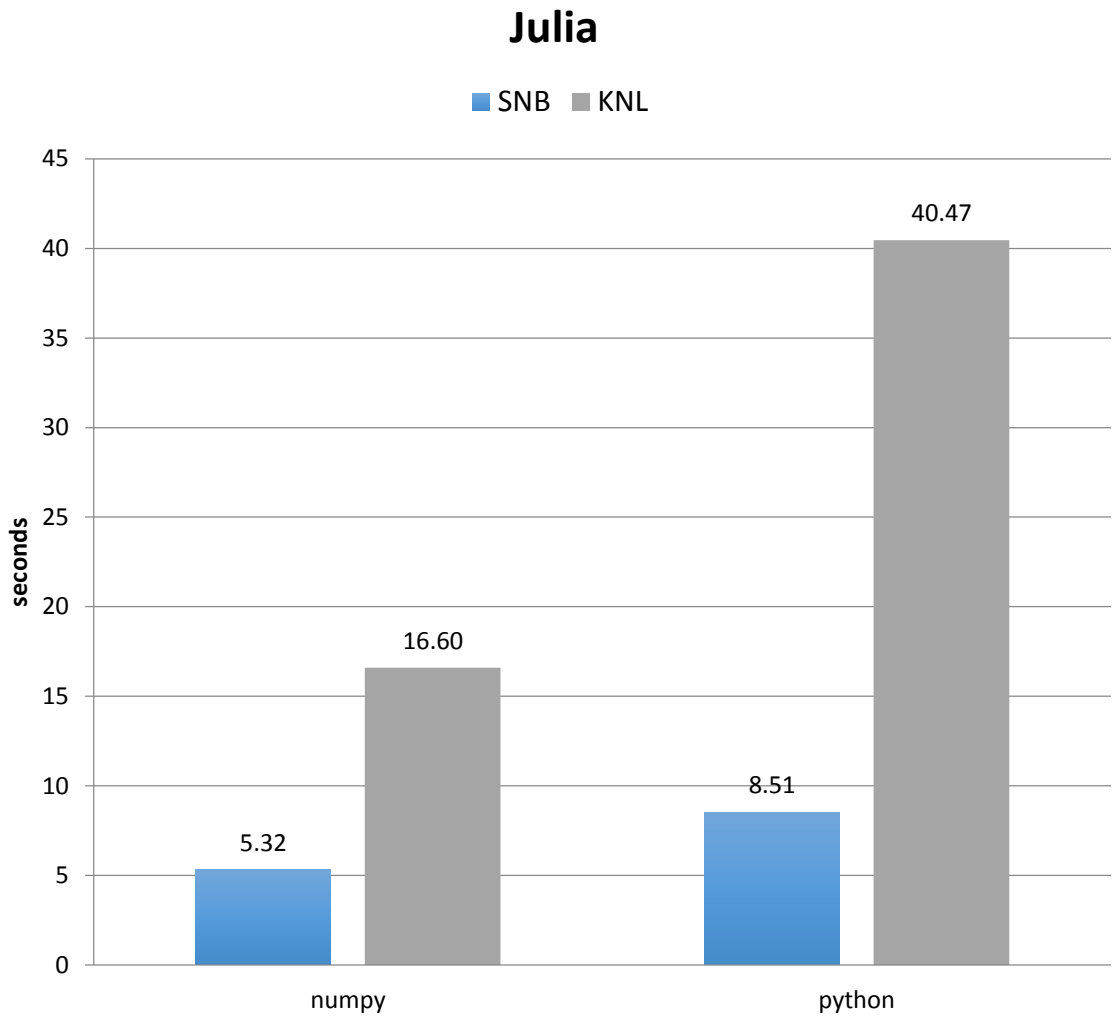
# WRF Performance




(higher is better)

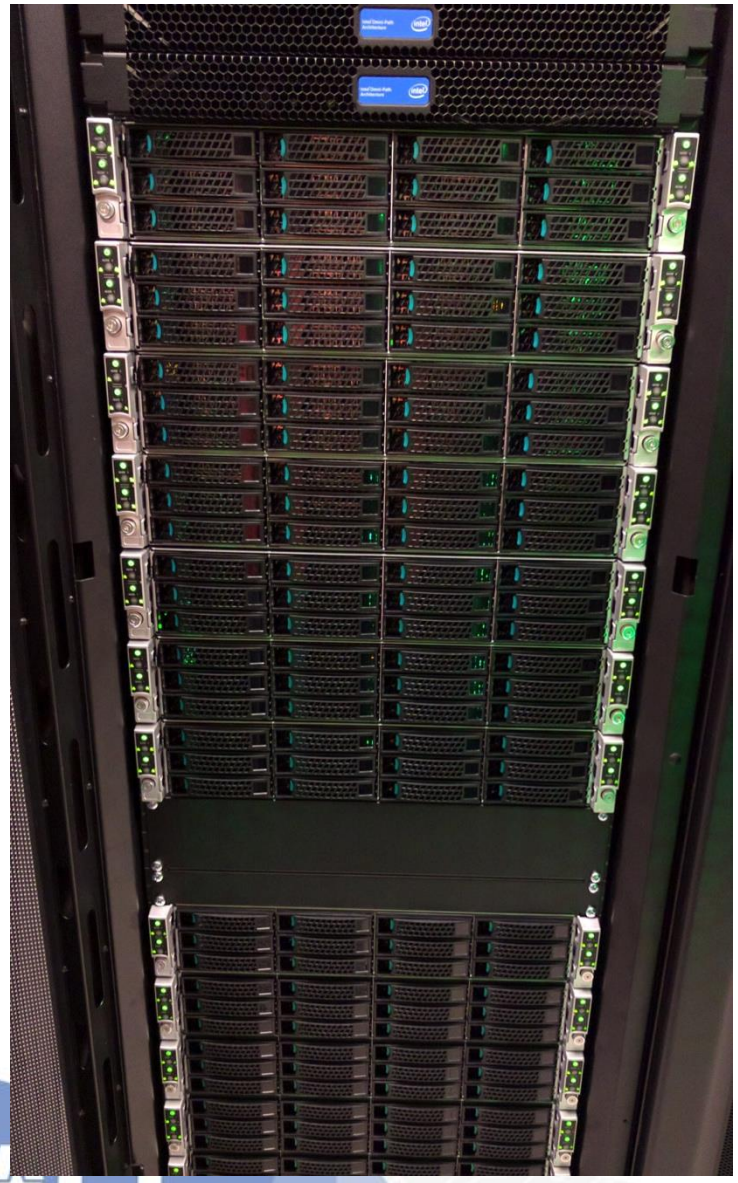
# Python Performance (serial codes)




(lower is better)

# Stampede KNL Upgrade



- Upgrade to TACC's Stampede cluster
- ~1.5 PF additional performance
  - 117 in Top 500
  - First KNL system on the list
- 504 68-core KNL nodes
- Intel's Omni-Path Fabric Network
- Separate cluster that shares filesystems with Stampede

# Using Stampede KNL Upgrade

- login-knl**1**.stampede.tacc.utexas.edu
- Login node is Haswell
  - Compilation recommended on login node
  - **`-xCORE-AVX2 -axMIC-AVX512 (only Intel)`**
- Normal queue is Cache Quadrant
  - Queues for different Flat modes
- 96 GB DDR4
- Two factor authentication

# Hybrid Computing on KNL

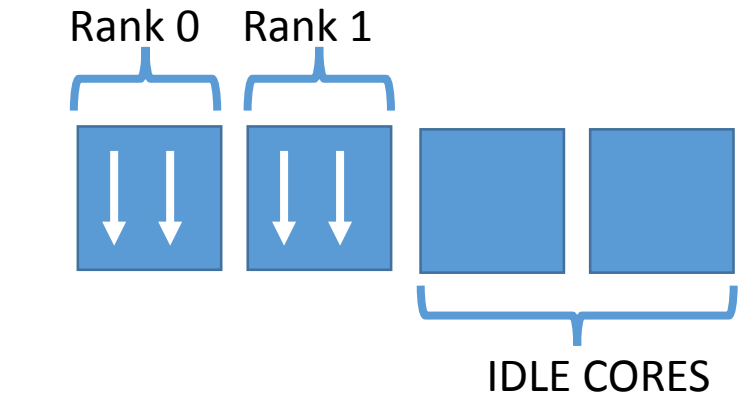
# Hybrid Computing

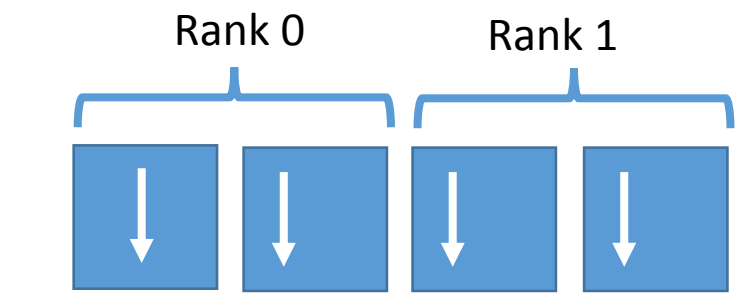
- While it is possible to run 68+ MPI processes on a KNL, it might not be a great idea
- MPI processes act as containers for threads
- Each MPI process is assigned a range of processors
- Threads spawned by an MPI process can only run within the assigned range.
- Two main issues:
  - Process binding (Where does my process run?)
  - Process affinity  (Where does my process access memory?)

# Why do I Care About Process Affinity?

- Process binding and affinity are critical in hybrid codes because of hardware complexity
- Often hybrid codes use less MPI tasks than cores
- If MPI binding is "compact" some hardware is not utilized!
- Multiple threads could be running on the same logical processor, overloading the hardware and serializing execution

Example: 2 MPI tasks, OMP_NUM_THREADS=2
Assume: 4 core CPU, 2 logical processors / core



# Binding

- Without any binding:
  - Threads can switch from a processor to another…
  - Thrashing the cache
  - And reducing performance due to the first touch policy
    - First touch: the first time you "touch" an array (allocation) determines the location in memory it resides
    - That location will be LOCAL if your thread does not float around

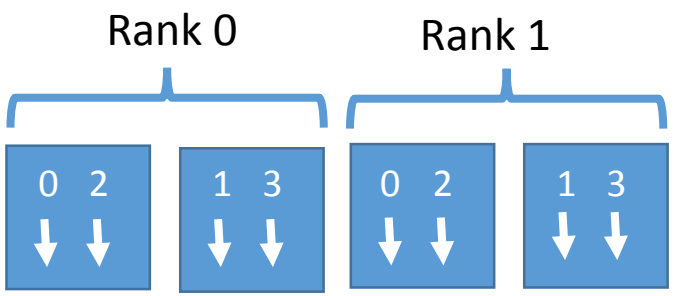
# Defaults with Intel MPI

- Sensible, but can be improved
- MPI tasks will be given a range of processors equal to the `OMP_NUM_THREADS` value
- Threads will then be assigned in a "scatter" sequence within the allowed processor range for each MPI task

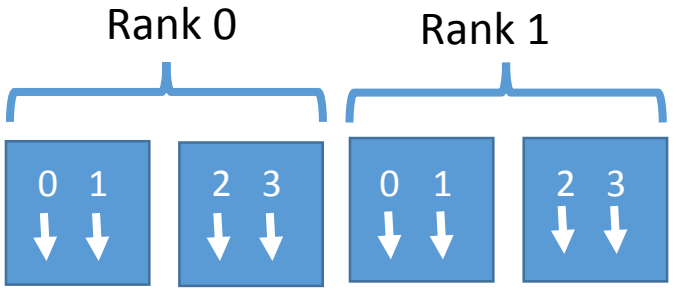
# Small example…

Assume 2 MPI tasks and OMP_NUM_THREADS=4

**Default**



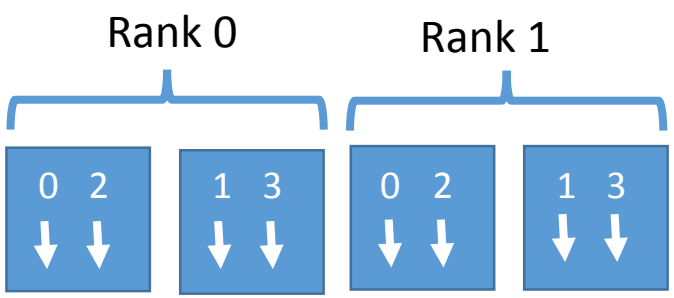

**Likely Optimal**



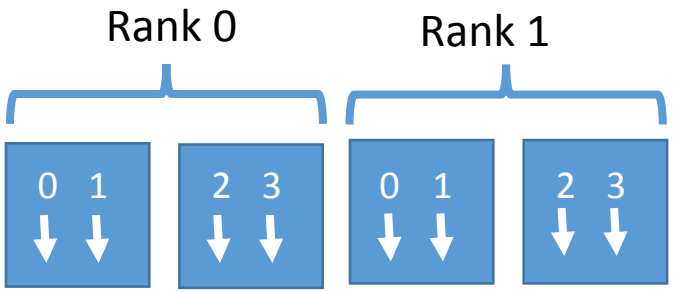
# Small example…

Assume 2 MPI tasks and OMP_NUM_THREADS=4

**Intel Default**




`OMP_PROC_BIND=spread`



# Checking MPI Process Binding

- Use the following to get Intel MPI to report process bindings:

```
export I_MPI_DEBUG=4
```

- As part of your standard output you will see something like:

```
[0] MPI startup(): Rank    Pid      Node name       Pin cpu
[0] MPI startup(): 0       105228   test1          {0,1,2,3,4,5, … 31, … }
[0] MPI startup(): 1       105229   test1          {32,33,34,35, … 63, … }
```

TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

The University of Texas at Austin

# Thanks!

agomez@tacc.utexas.edu