Intel Xeon® , Intel Xeon Phi™ Coprocessor and KNL Architecture

Silvio Stanzani , Raphael Cóbe , Rogério Iope

UNESP - Núcleo de Computação Científica

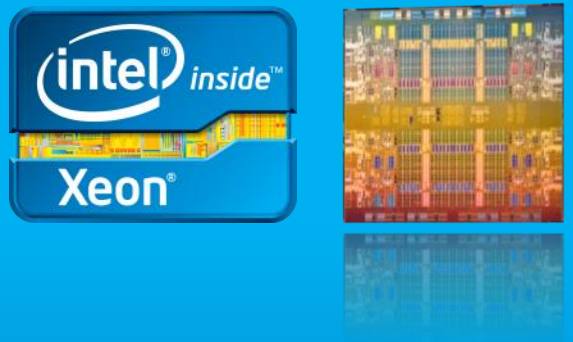
silvio@ncc.unesp.br , rmcobe@ncc.unesp.br , rogerio@ncc.unesp.br

Module Outline

- **Intel Xeon and Intel® Xeon Phi™ Overview**
- Programming Model
- Vector Processing Unit
- Setting Expectations
- System Software and OS
- Application Environment

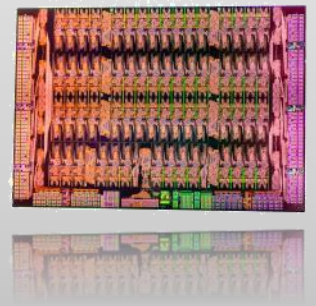
Intel Xeon and Intel® Xeon Phi™ Overview

Intel® Multicore Architecture



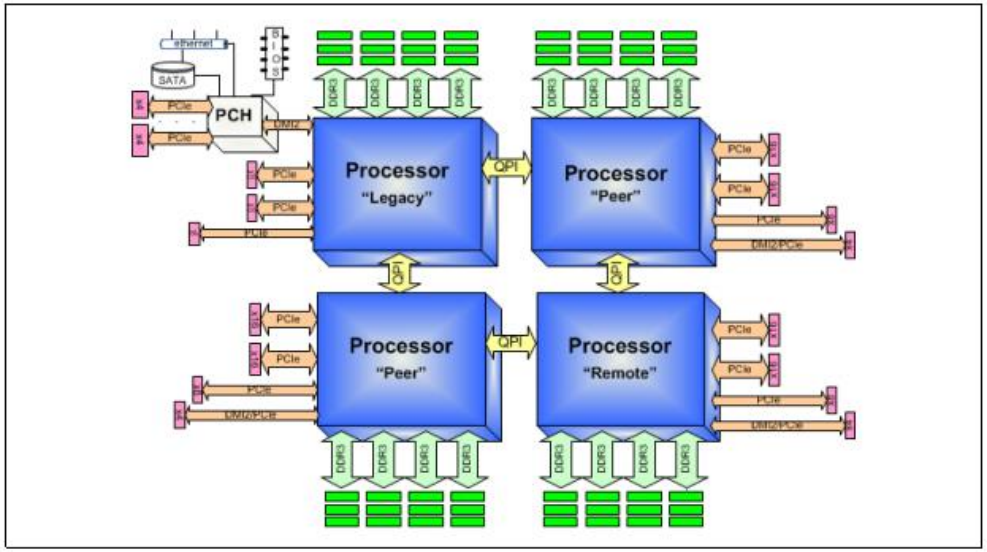
- Foundation of HPC Performance
- Suited for full scope of workloads
- Industry leading performance and performance/watt for serial & parallel workloads
- Focus on fast single core/thread performance with “moderate” number of cores

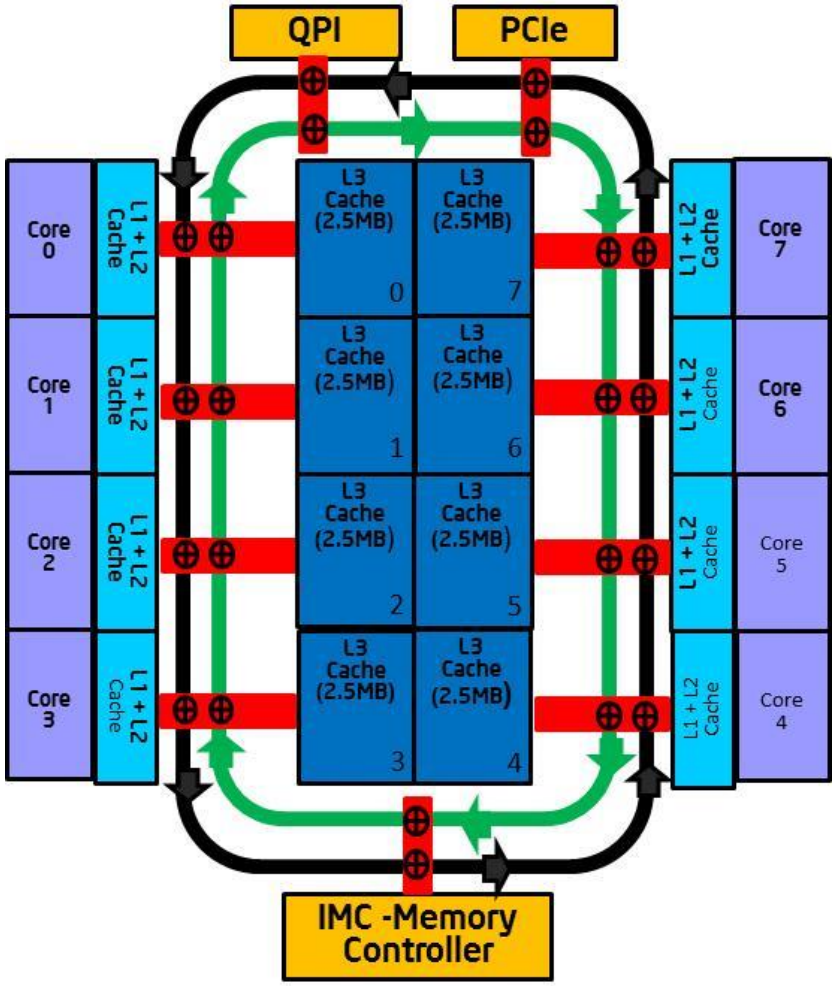
Intel® Many Integrated Core Architecture



- Performance and performance/watt optimized for highly parallelized compute workloads
- Common software tools with Xeon enabling efficient application readiness and performance tuning
- IA extension to Manycore
- Many cores/threads with wide SIMD

Intel Xeon Architecture Overview



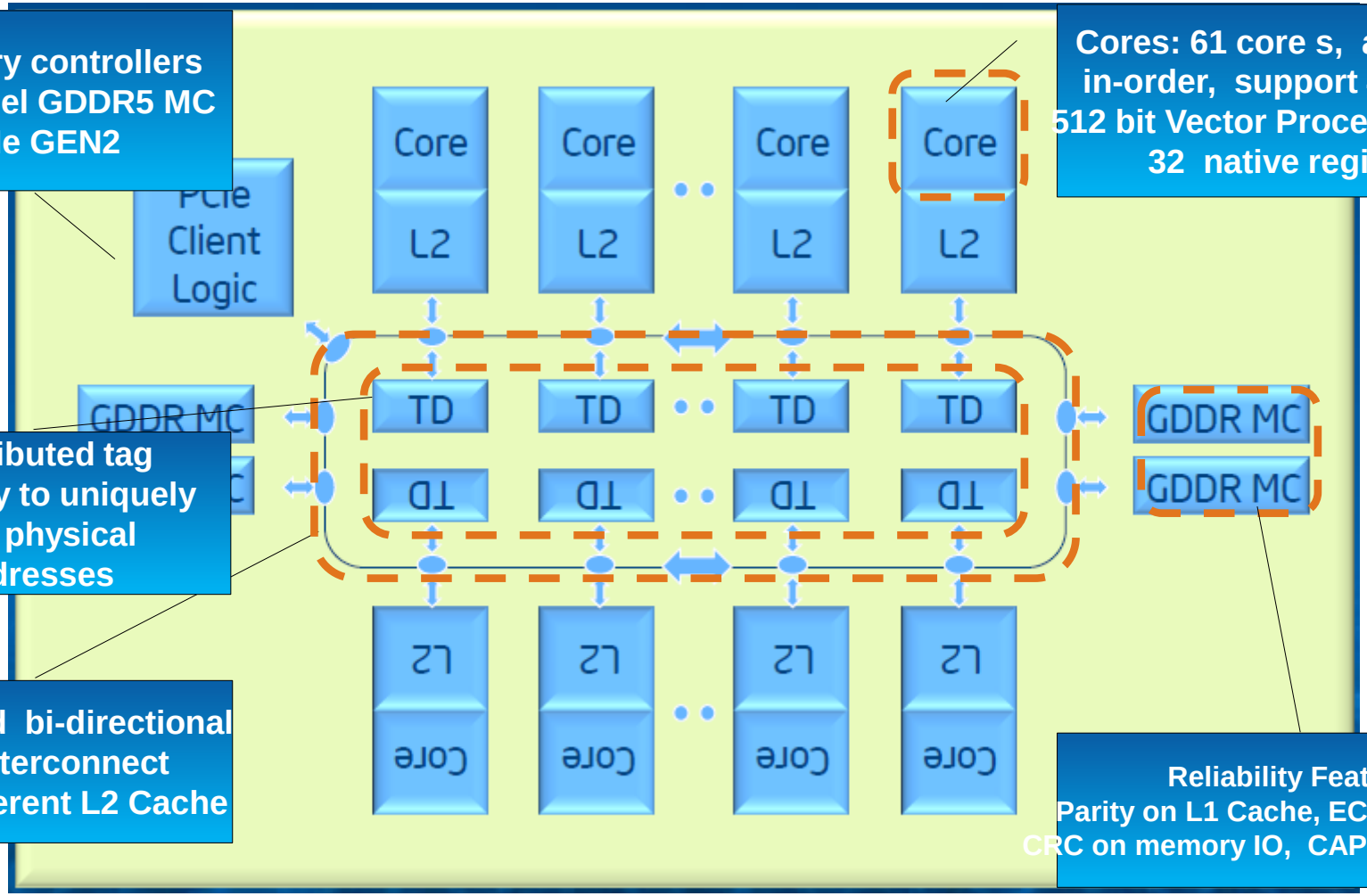


Intel Xeon Architecture Overview

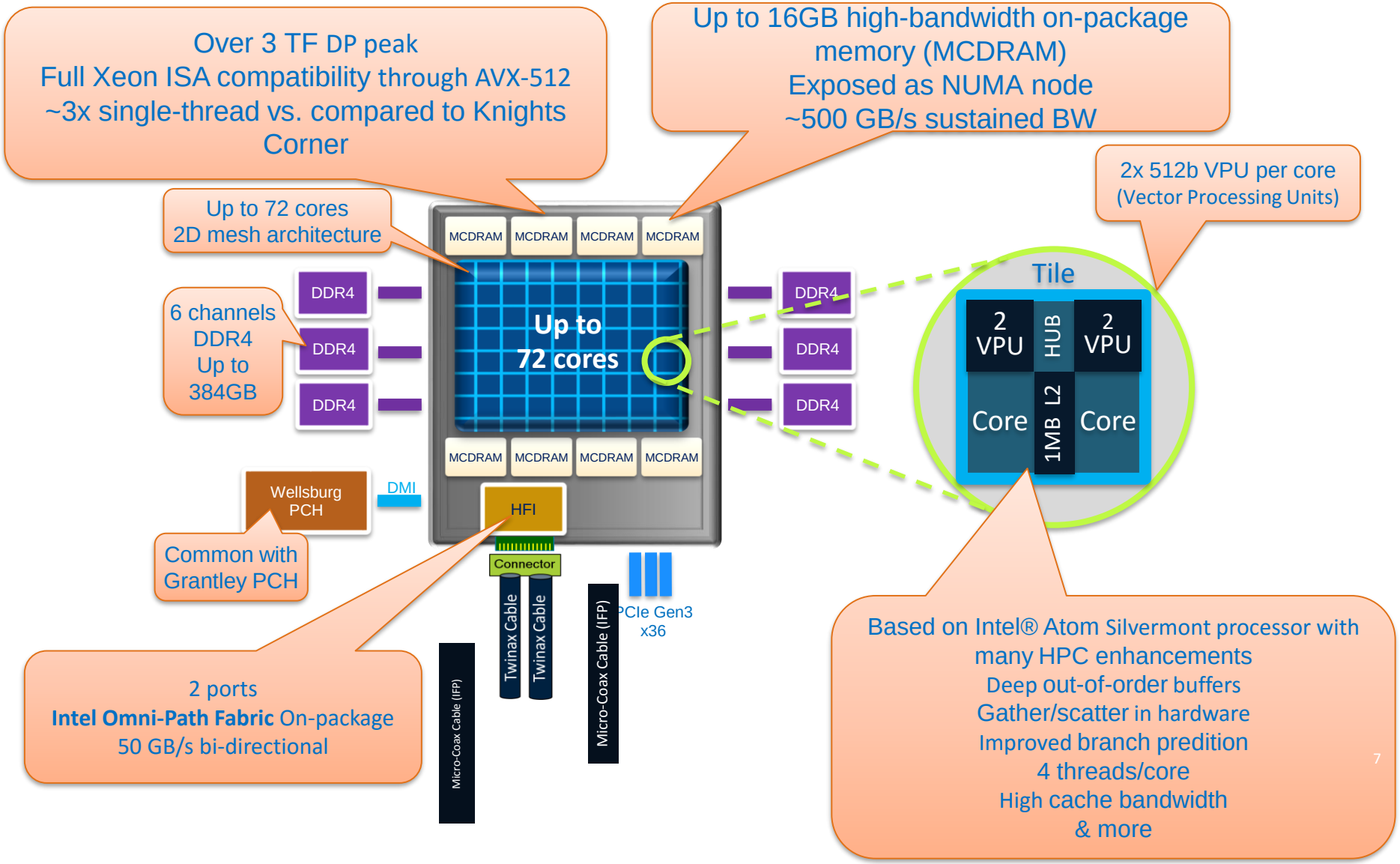
- Socket: mechanical component that provides mechanical and electrical connections between a microprocessor and a printed circuit board (PCB).

- QPI (Intel QuickPath Interconnect): high speed, packetized, point-to-point interconnection, that stitch together processors in distributed shared memory and integrated I/O platform architecture.

Intel® Xeon Phi™ Architecture Overview



Knights Landing (KNL)



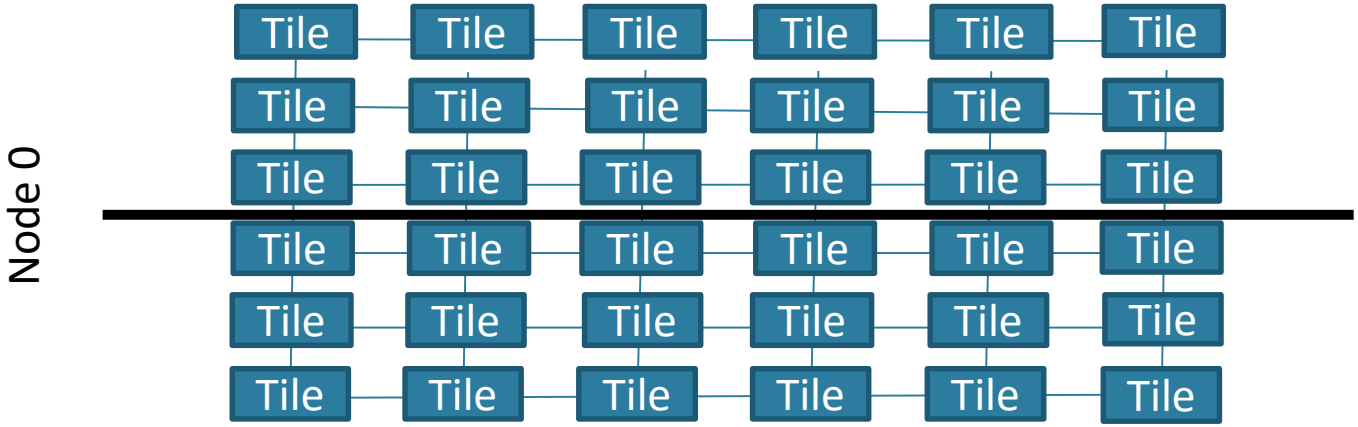
Knights Landing (KNL)

- KNL Knights Landing has 72 cores;
- Each one has an L1 cache;
- Pairs of cores are organized into tiles with a slice of the L2 cache symmetrically shared between the two cores;
- All caches are kept coherent;
- Two VPUs (Vector Processing Units) per core;

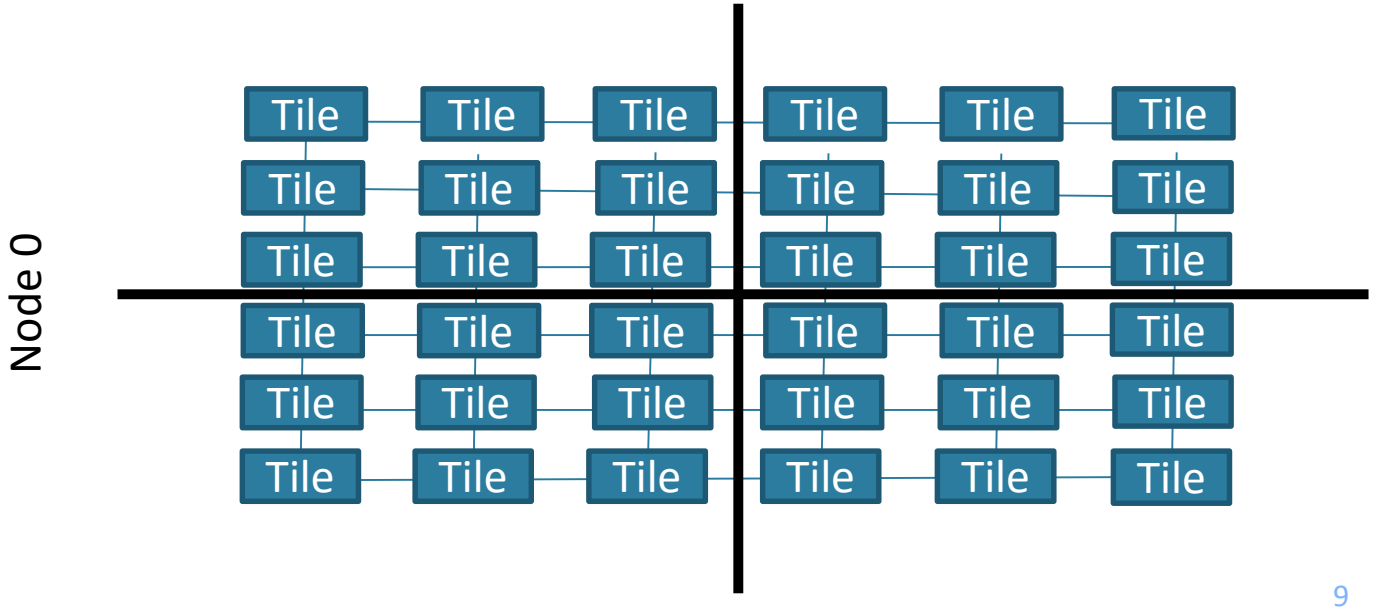
Cluster modes

One single space address

Hemisphere:
the tiles are divided into two parts called hemisphere



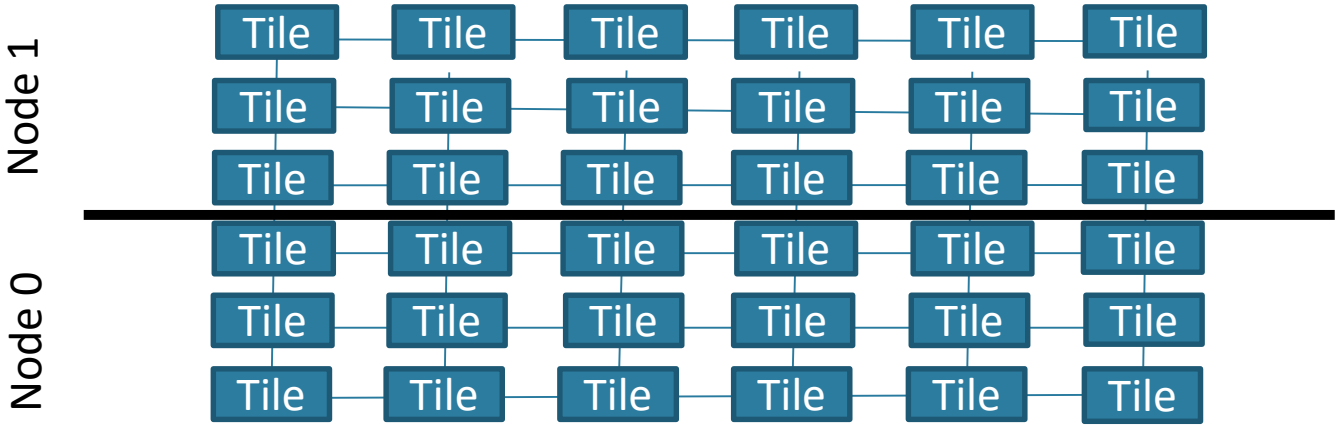
Quadrant:
tiles are divided into two parts called hemisphere or into four parts called qudrants



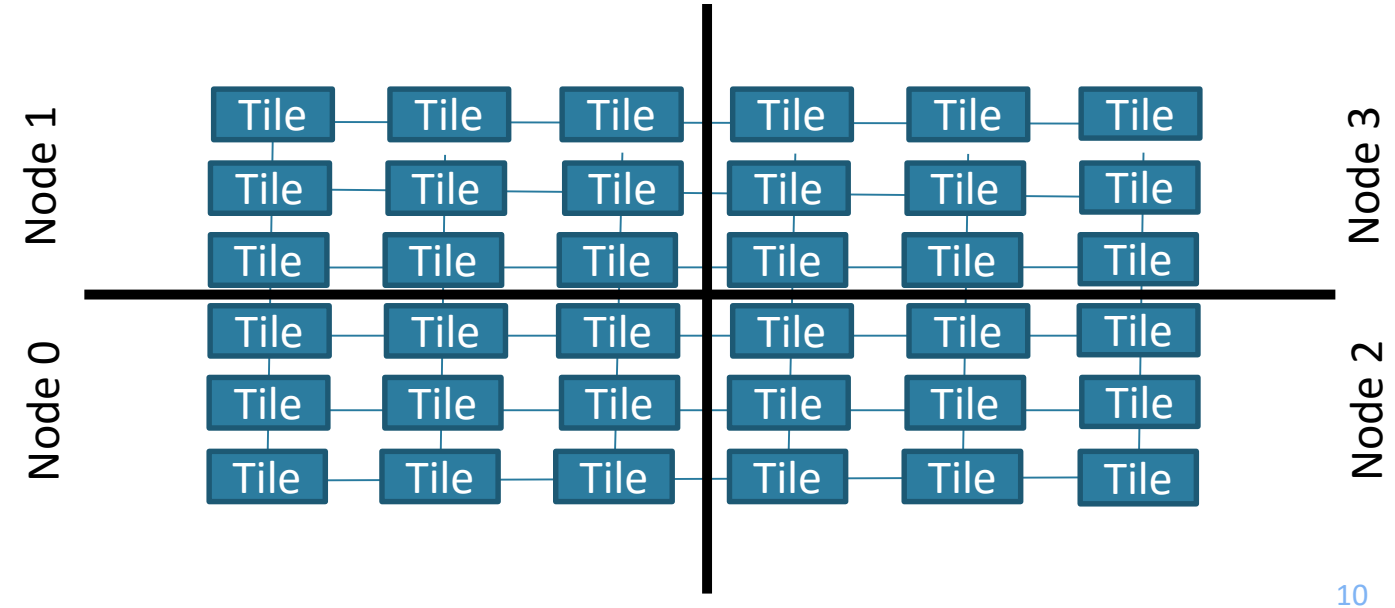
Cluster modes

Cache data are isolated in each sub numa domain

SNC-2:
the tiles are divided into two Numa Nodes

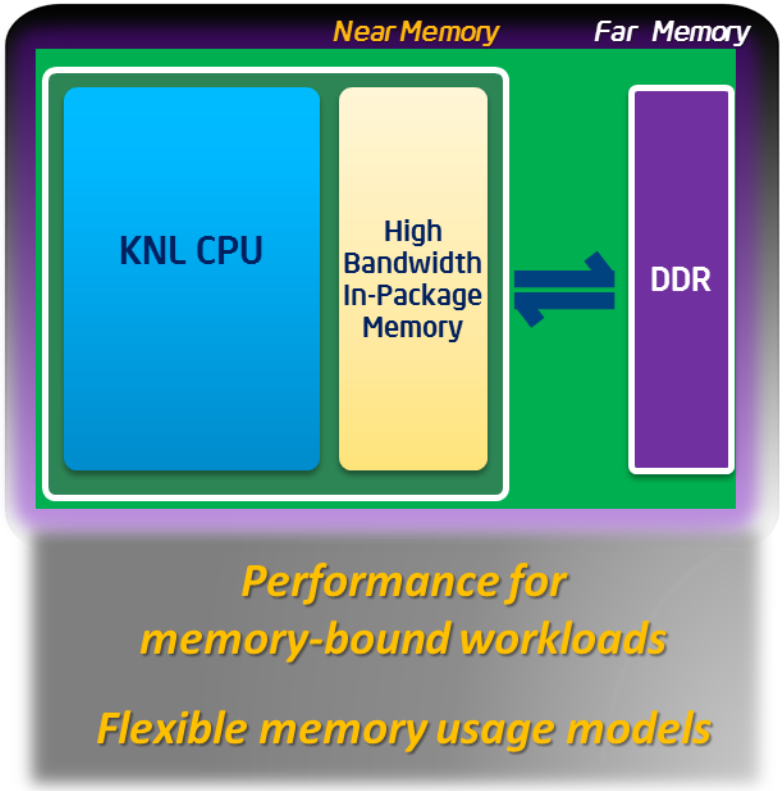


SNC-4:
the tiles are divided into two Numa Nodes



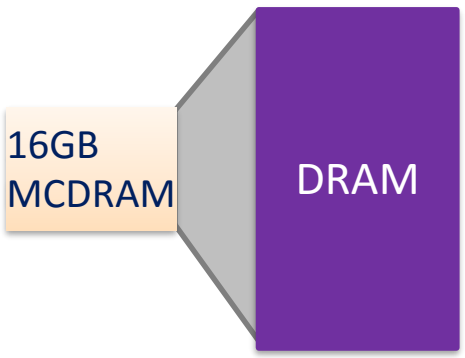
Knights Landing Integrated On-Package Memory

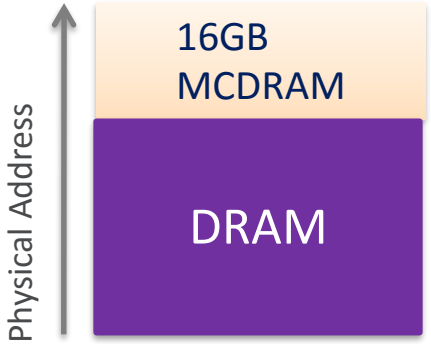
Multi-Channel DRAM (High-bandwidth memory)

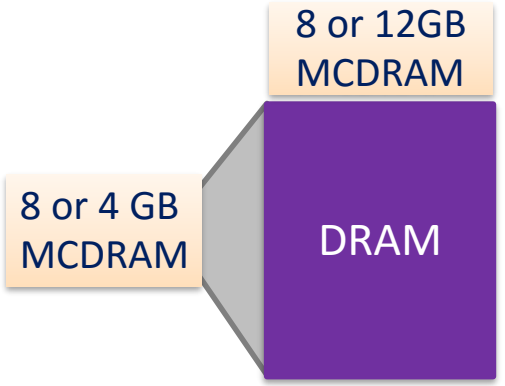


Integrated On-Package Memory Usage Models

Model configurable at boot tiem and software exposed through NUMA







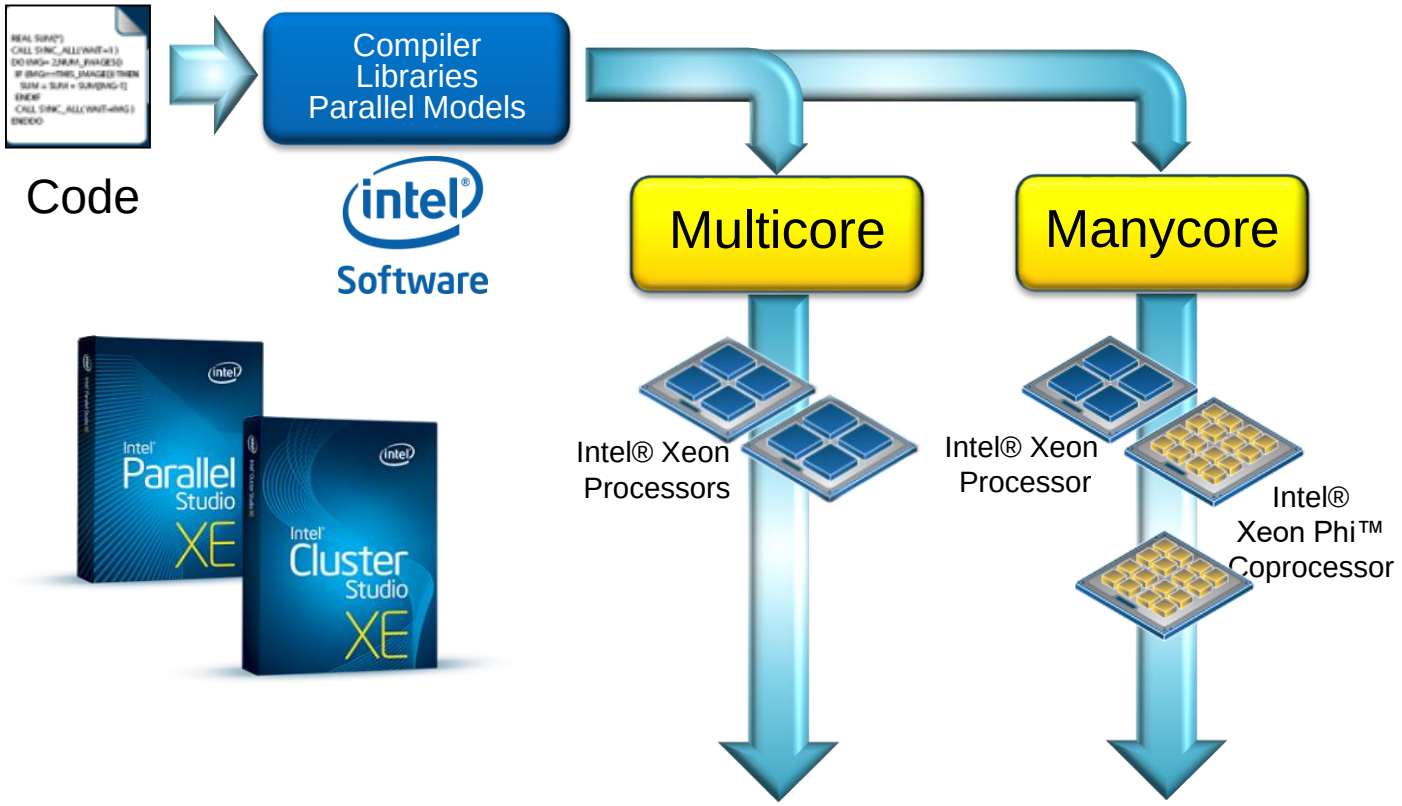
Split Options:
25/75% or 50/50%

Cache Model	Flat Model	Hybrid Model
Hardware automatically manages the MCDRAM as a "L3 cache" between CPU and ext DDR memory	Manually manage how the app uses the integrated on-package memory and external DDR for peak perf	Harness the benefits of both Cache and Flat models by segmenting the integrated on-package memory
▪ App and/or data set is very large and will not fit into MCDRAM ▪ Unknown or unstructured memory access behavior	▪ App or portion of an app or data set that can be, or is needed to be "locked" into MCDRAM so it doesn't get flushed out	▪ Need to "lock" in a relatively small portion of an app or data set via the Flat model ▪ Remaining MCDRAM can then be configured as Cache

Module Outline

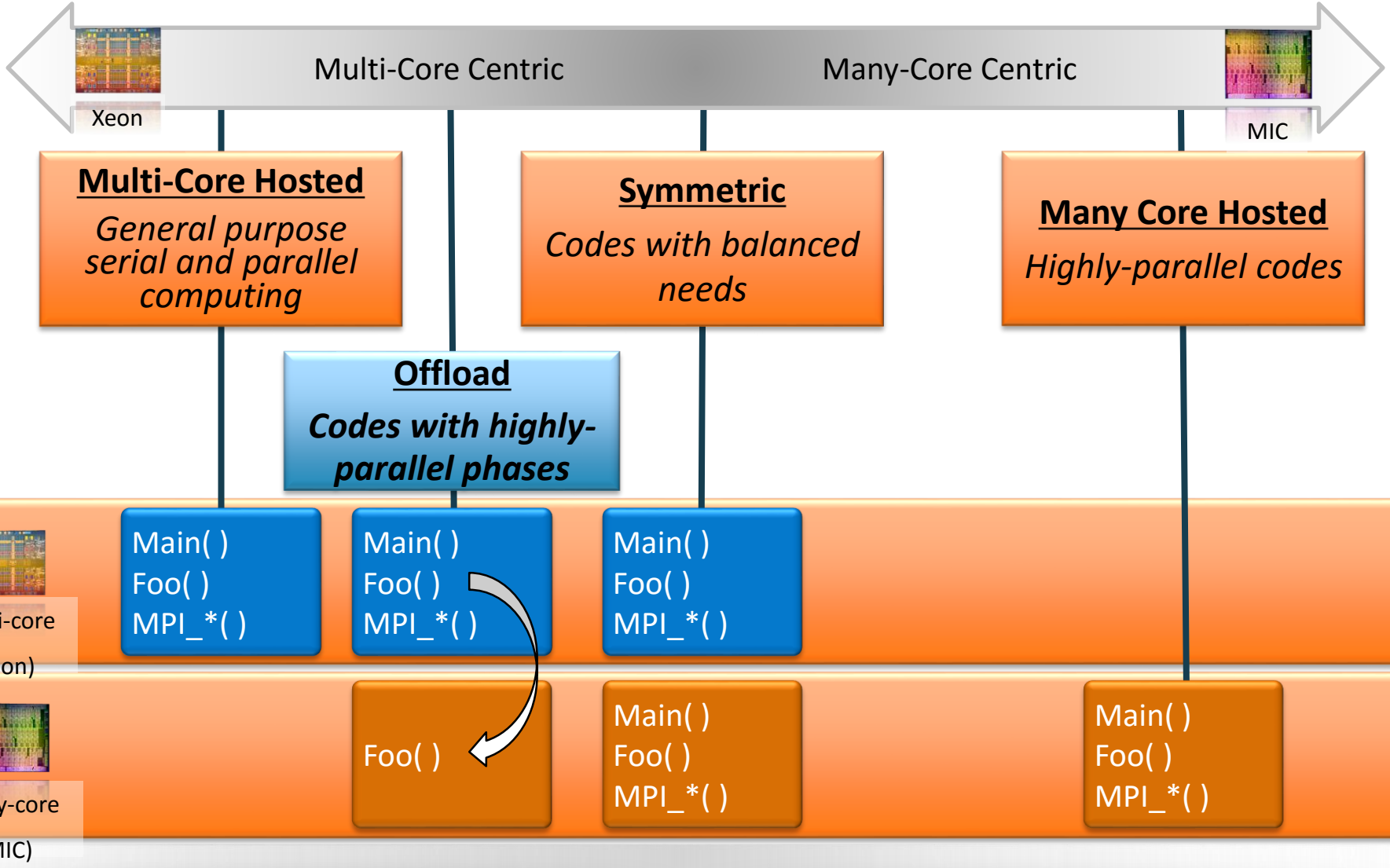
- Intel Xeon and Intel® Xeon Phi™ Overview
- **Programming Model**
- Vector Processing Unit
- Setting Expectations
- System Software and OS
- Application Environment

Programming Models



Standards Programming Models
Vectorize, Parallelize, & Optimize

Programming Models



Range of models to meet application needs

Module Outline

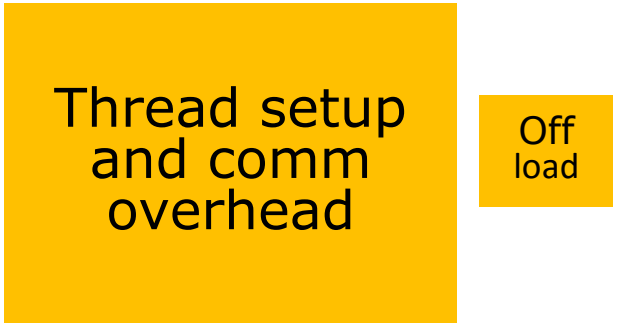
- Intel Xeon and Intel® Xeon Phi™ Overview
- Programming Model
- Vector Processing Unit
- **Setting Expectations**
- System Software and OS
- Application Environment

Intel® MIC Programming Considerations

- Getting full performance from the Intel® MIC architecture requires both a high degree of parallelism and vectorization
  - Not all code can be written this way
  - Not all programs make sense on this architecture

- Intel® MIC is not an Intel® Xeon® processor
  - It specializes in running highly parallel and vectorized code.
  - New vector instruction set and 512-bit wide registers
  - Not optimized for processing serial code

Intel® MIC Programming Considerations



- KNC comes with 8GB of memory
  - Only ~7GB is available to your program
    - The other ~1GB is used for data transfer and is accessible to your Intel® MIC Architecture code as buffers.

- Very short (low-latency) tasks not optimal for offload to the coprocessor
  - Costs that you need to amortize to make it worthwhile:
    - Cost of code and data transfer
    - Cost of process/thread creation
  - Fastest data transfers currently require careful data alignment
- This architecture is not optimized for serial performance

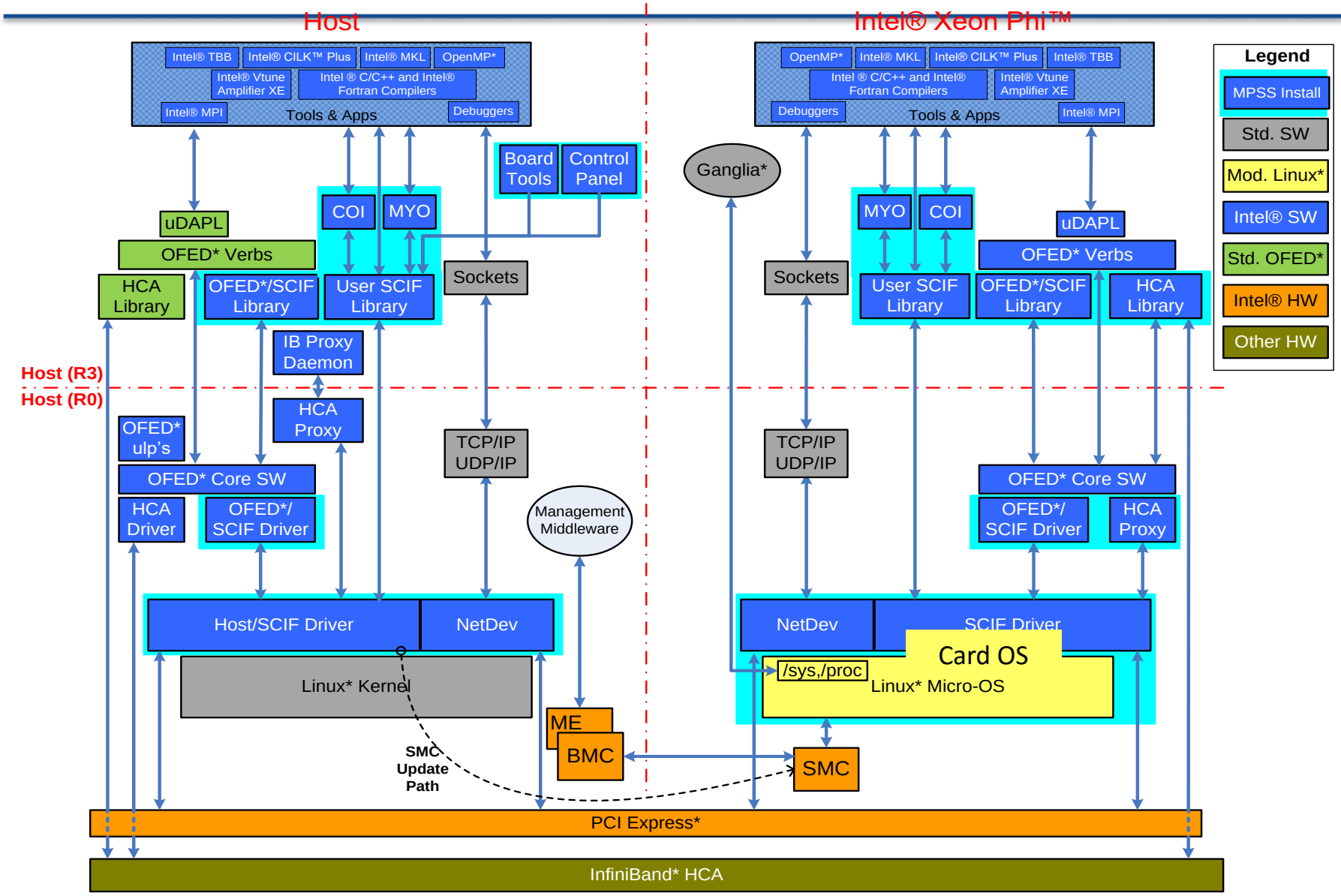
Module Outline

- Intel Xeon and Intel® Xeon Phi™ Overview
- Programming Model
- Vector Processing Unit
- Setting Expectations
- **System Software and OS**
- Application Environment

Intel® Xeon Phi™ Coprocessor Arch – System SW Perspective

- Large SMP UMA machine – a set of x86 cores to manage
  - 4 threads and 32KB L1I/D, 512KB L2 per core
  - Supports loadable kernel modules
  - Standard Linux kernel from kernel.org
    - 2.6.38 in the most recent release
  - Completely Fair Scheduler (CFS), VM subsystem, File I/O
- Virtual Ethernet driver– supports NFS mounts from Intel® Xeon Phi™ Coprocessor
- New vector register state per thread for Intel® IMCI
  - Supports “Device Not Available” for Lazy save/restore
- Different ABI – uses vector registers for passing floats
  - Still uses the x86_64 ABI for non-float parameter passing (rdi, rsi, rdx ..)

System SW Environment



Module Outline

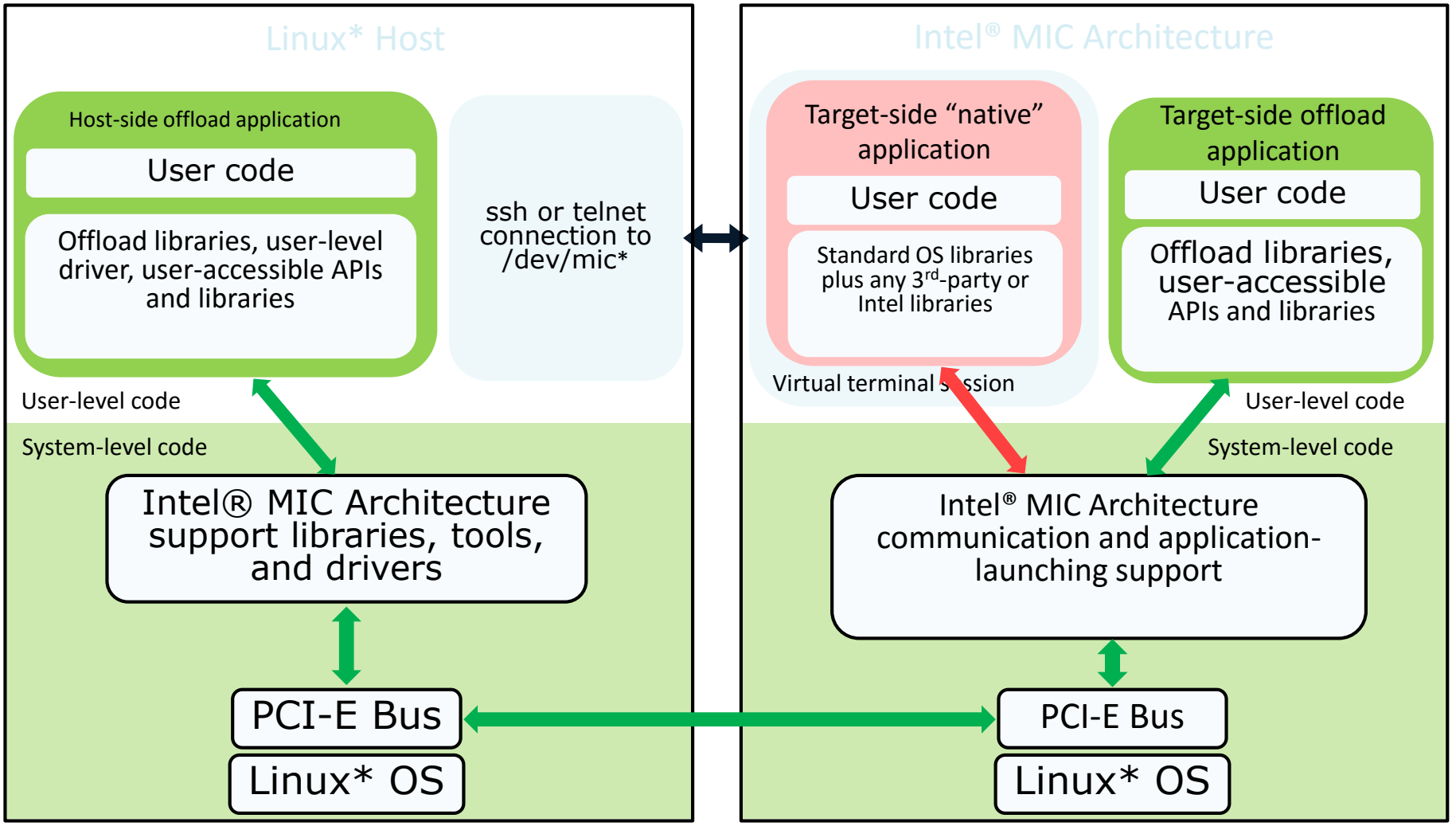
- Intel Xeon and Intel® Xeon Phi™ Coprocessor Overview
- Core and Vector Processing Unit
- Setting Expectations
- System Software and OS
- **Application Environment**

Offload Programming Model Data Transfer

- Explicit distributed memory programming via MPI (or sockets)
- Two offload data transfer models are available:
  - Explicit Copy
    - Programmer designates variables that need to be copied between host and card in the offload directive
    - Syntax: Pragma/directive-based
    - C/C++ Example: #pragma offload target(mic) in(data:length(size))
    - Fortran Example: !dir$ offload target(mic) in(a1:length(size))
  - Implicit Copy
    - Programmer marks variables that need to be shared between host and card
    - The same variable can then be used in both host and coprocessor code
    - Runtime automatically maintains coherence at the beginning and end of offload statements
    - Syntax: keyword extensions based
    - Example: _Cilk_shared double foo; _Offload func(y);

Two Application Execution Environments

+Hybrid



MICSMC


Intel® Xeon Phi™ Coprocessor Platform Control Panel
Cards
Advanced
Cards: 3
Memory: 16.82 GB
Cores: 171
Utilization View (All Devices)
System [%]
User [%]
100
80
60
40
20
0
13:41:54
13:42:14
13:42:34
Average Core Utilization
53
Average Core Temperature °C
3.11 GB
Total Memory Usage
279 Watts
Total Power Usage
mic0: Utilization View
System [%]
User [%]
13:41:54
13:42:14
13:42:34
Core Utilization
52
Processor Core Temperature °C
1986.2 MB
Memory Usage
95 Watts
Power Usage
mic1: Utilization View
System [%]
User [%]
13:41:54
13:42:14
13:42:34
Core Utilization
50
Processor Core Temperature °C
503.5 MB
Memory Usage
99 Watts
Power Usage

Hello World

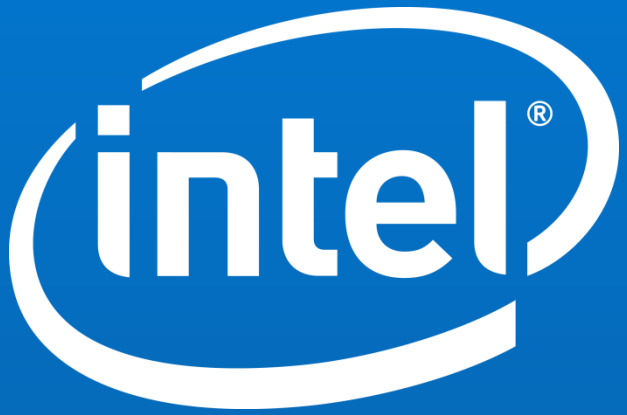
```
[silvio@phi01 ~]$ cat helloWorld.c
#include <stdio.h>

#include <unistd.h>

int main() {

printf("Hello world! I have %ld logical cores.\n", sysconf(_SC_NPROCESSORS_ONLN ));

}
[silvio@phi01 ~]$ icc helloWorld.c -o helloWorld
[silvio@phi01 ~]$ ./helloWorld
Hello world! I have 32 logical cores.
[silvio@phi01 ~]$ icc helloWorld.c -o helloWorld.mic -mmic
[silvio@phi01 ~]$ ssh mic0
silvio@phi01-mic0:~$ ./helloWorld.mic
Hello world! I have 228 logical cores.
silvio@phi01-mic0:~$
```


intel