Northwestern University

ILLINOIS INSTITUTE OF TECHNOLOGY

The Case for an Interwoven Parallel Hardware/Software Stack

Kyle Hale, Simone Campanoni, Nikos Hardavellas, Peter Dinda

HExSA

Northwestern Compiler Group

PARAGON
PARALLEL ARCHITECTURE GROUP @NORTHWESTERN

Plab

The Interweaving Project
http://interweaving.org

The parallel software stack has ossified

…but

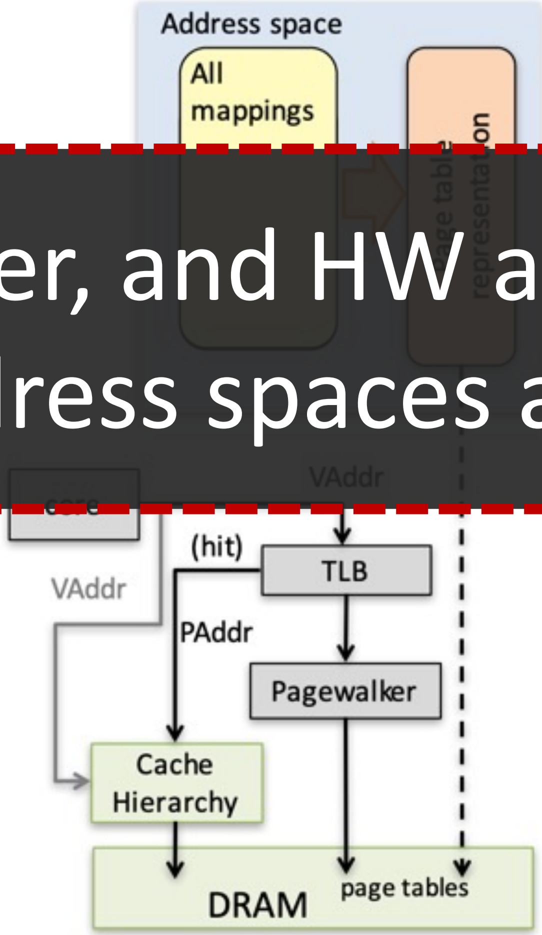
Hardware Diversity & Better Design Tools

Better Compiler Infrastructure

Virtualization & Flexibility for Experimental OS work

Good position to rethink the parallel HW/SW stack

Why Change?
Limitations of the Parallel HW/SW Stack

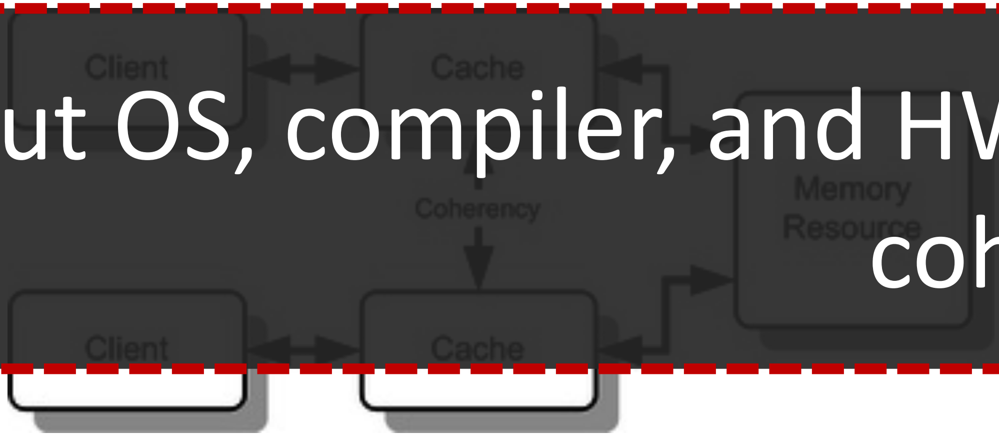


+ Simplicity for programmer
+ Efficient HW

But OS, compiler, and HW all make assumptions about address spaces and translation!

- Design baked into HW
- Workloads changing
- SASOS increasingly common
- Power consumption

Why Change?
Limitations of the Parallel HW/SW Stack



HW cache coherence

+ Enables efficient shared memory
+ Abstraction (hardware "takes care of it")

- Not all workloads *need* coherence
- False sharing
- No SW control (hidden from OS, language)

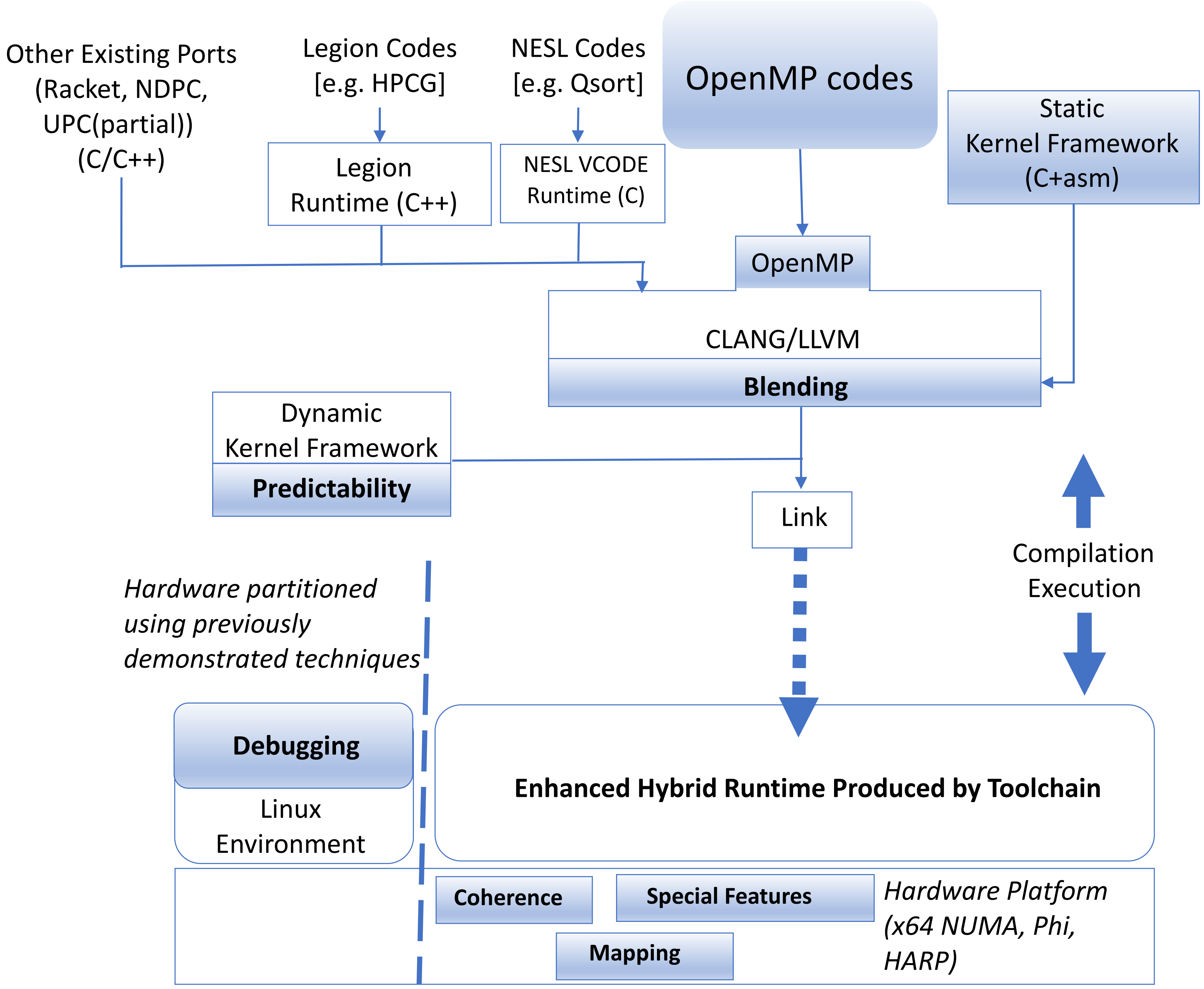
But OS, compiler, and HW all make assumptions about coherence!

Many assumptions span the stack…

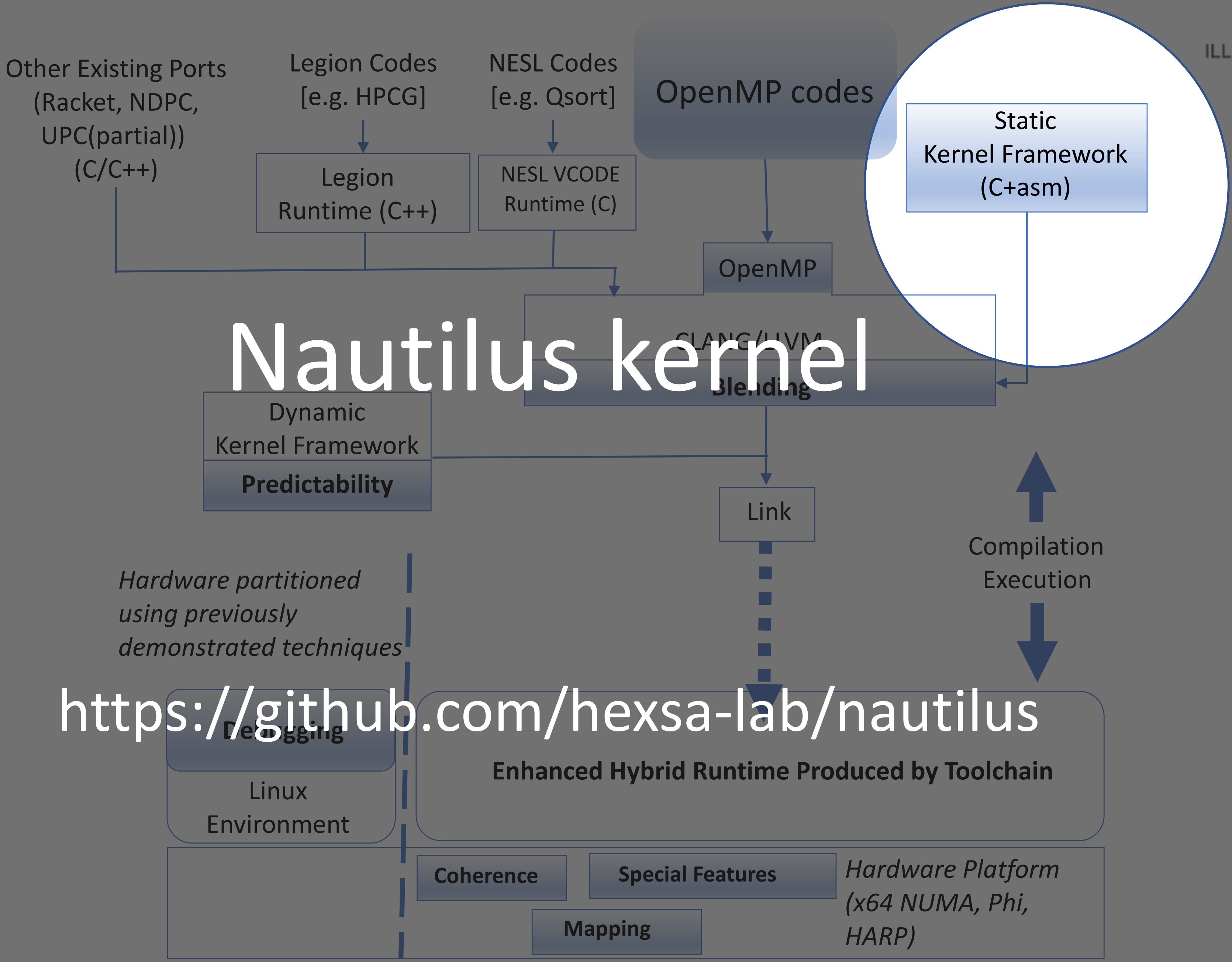
- Timing
- Synchronization
- Coherence
- Address Translation
- Execution contexts
- Scheduling (polling, lack of real-time, etc.)
- …

Interweaving: reconsider these at all layers

System Vision



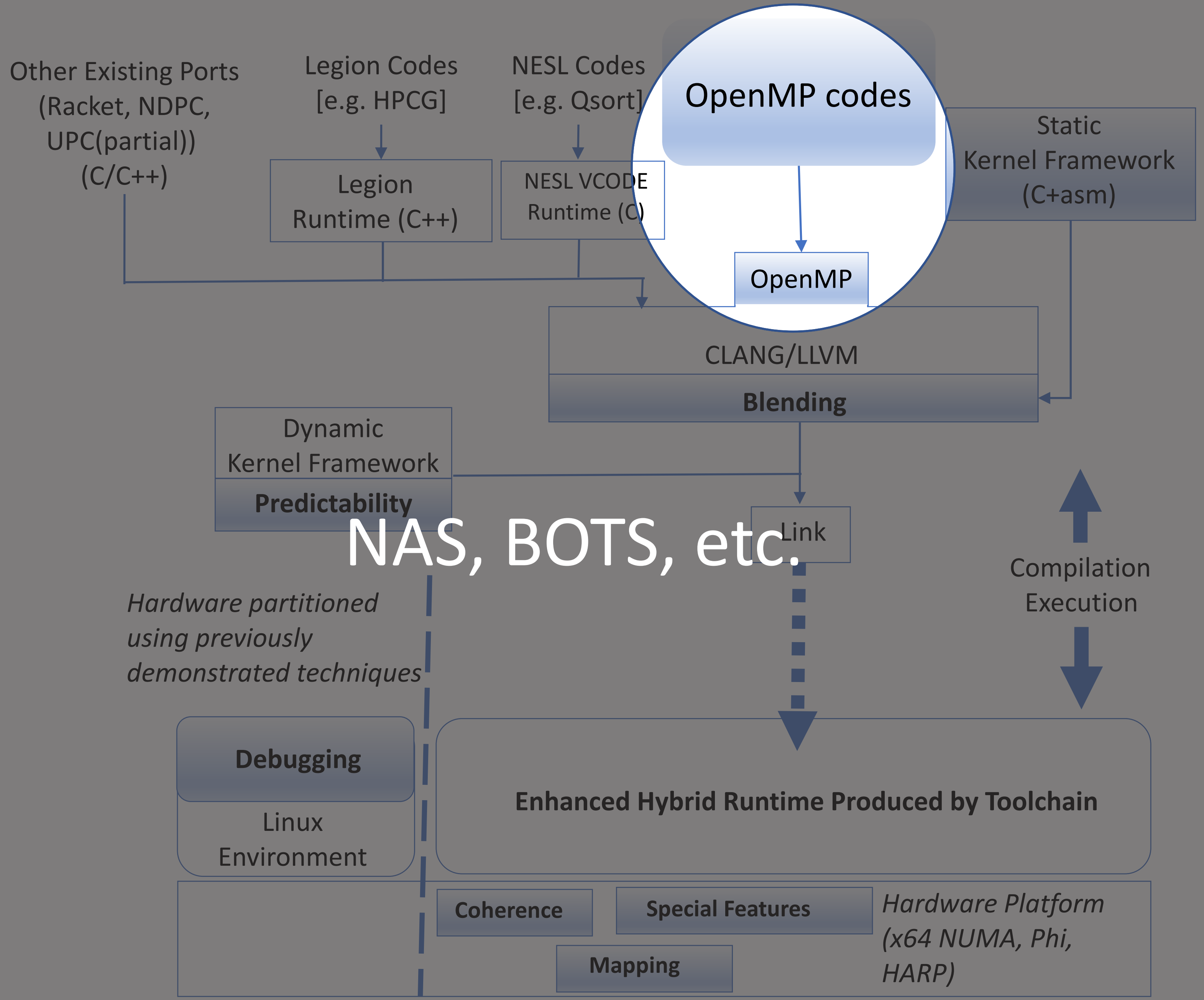
System Vision



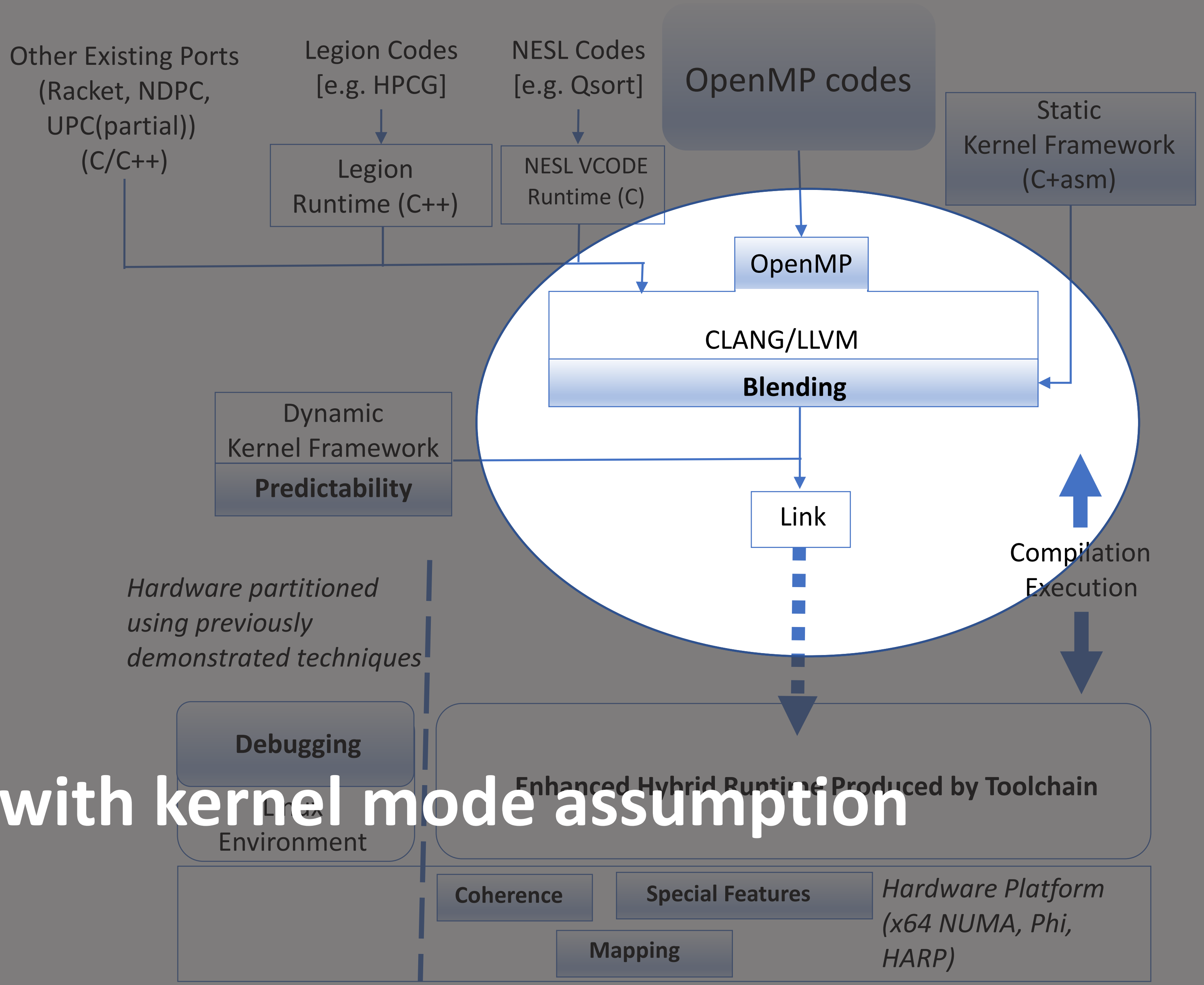
Nautilus kernel

https://github.com/hexsa-lab/nautilus

System Vision

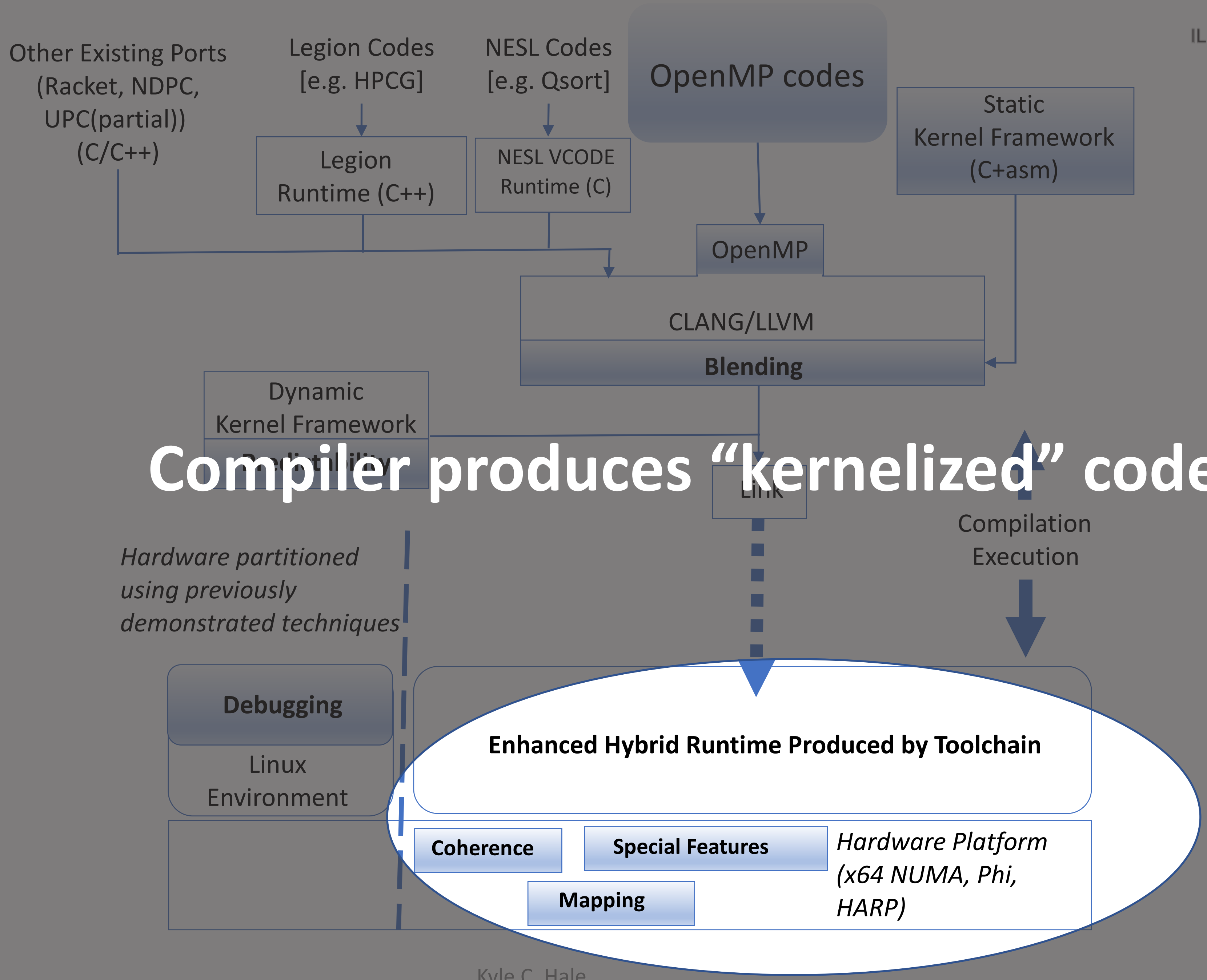


System Vision

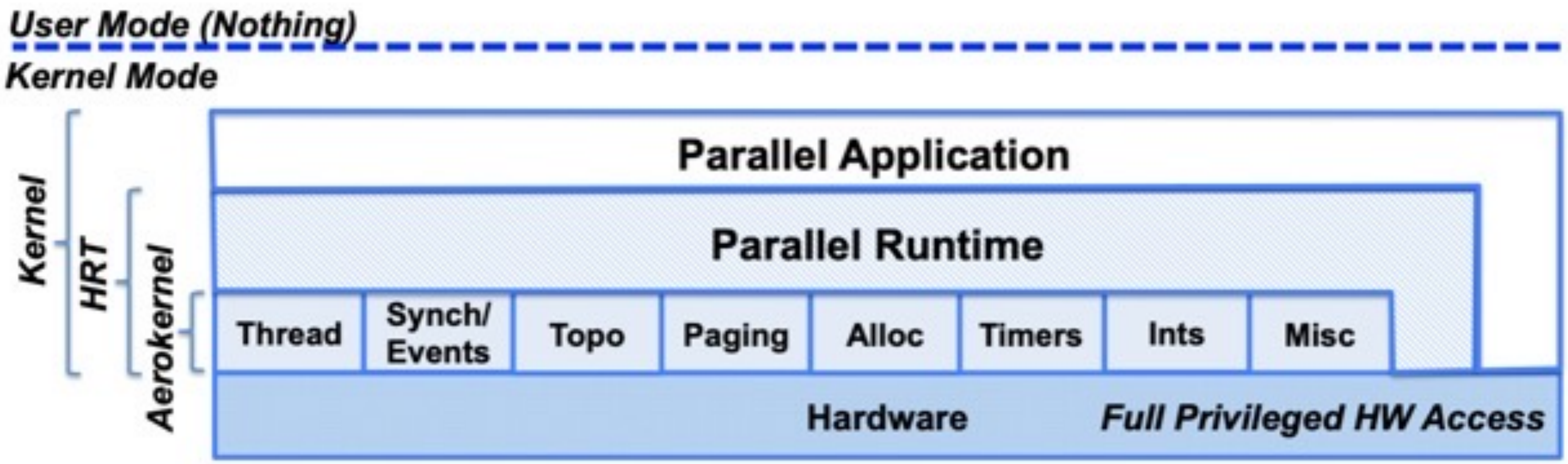


Compile with kernel mode assumption

System Vision



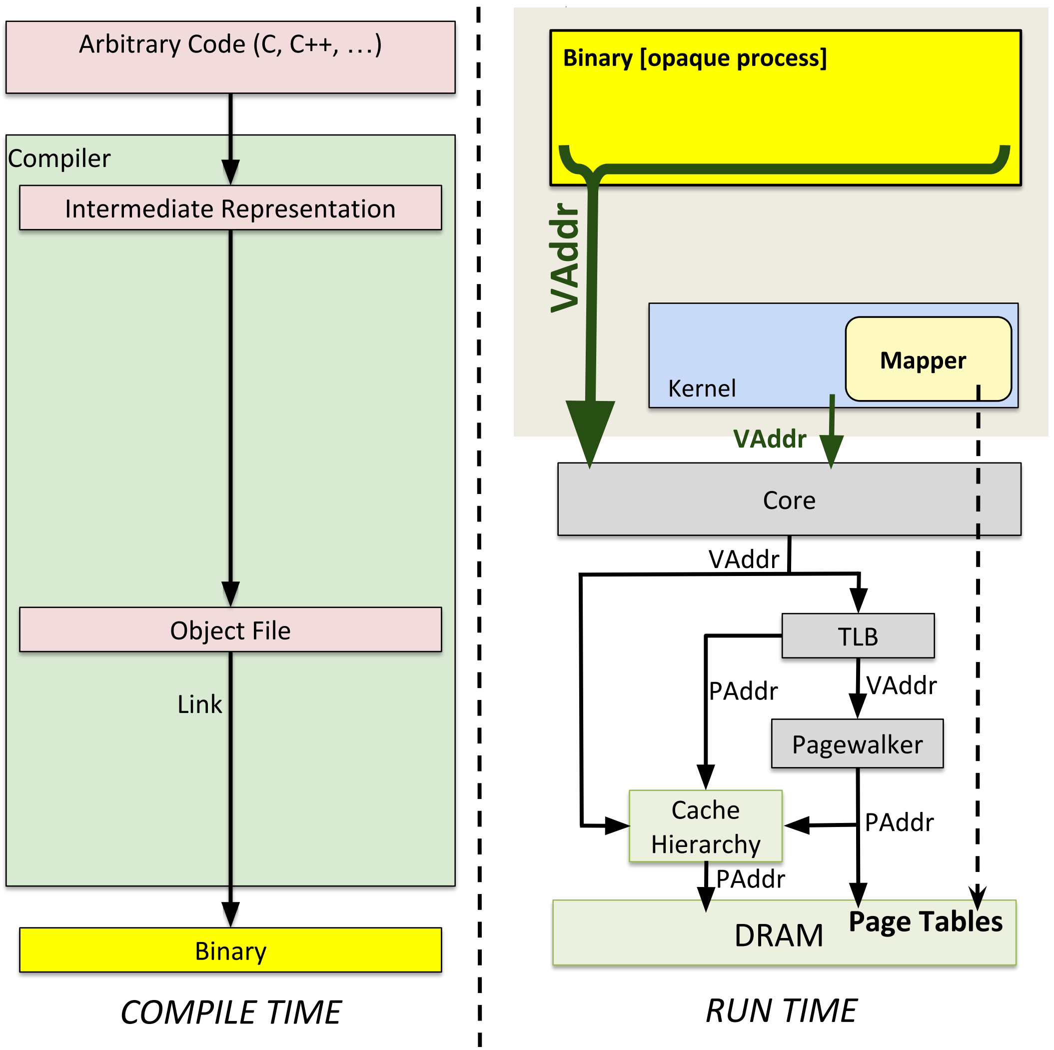
Nautilus Kernel



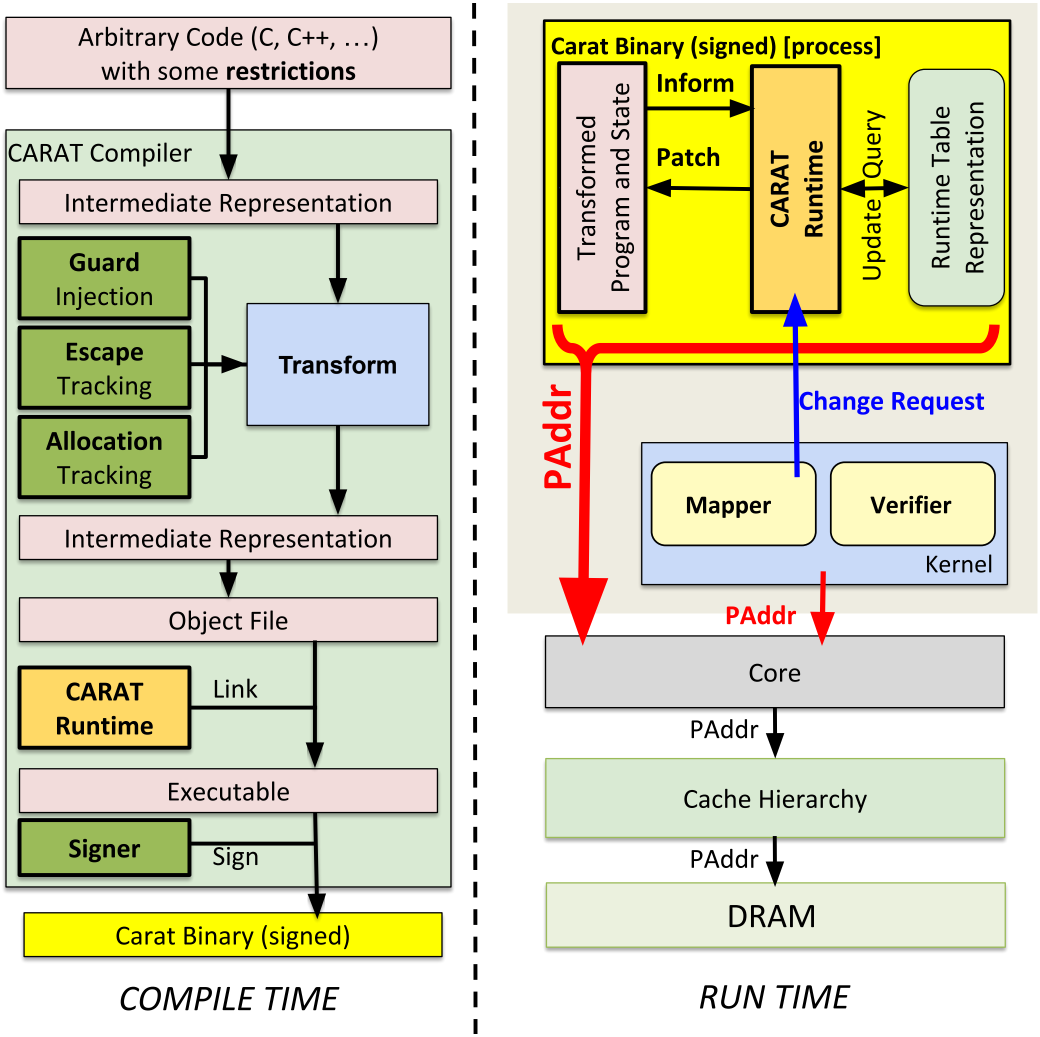
Interweaving Example:
CARAT: Compiler and Runtime-based Address Translation

- Address translation (via paging) is universal, yet showing its age
  - TLB misses are a significant performance inhibitor, including in HPC
  - TLBs/paging limits cache design
  - Fundamentally a hardware/kernel co-design
- Can we do better than paging?
- CARAT uses physical addresses instead (no paging, no TLB)
  - Protection and memory migration achieved via compiler/kernel co-design
- Proof-of-concept shows feasibility

Traditional Address Translation



CARAT: Compiler and Runtime-based Address Translation [PLDI '20]



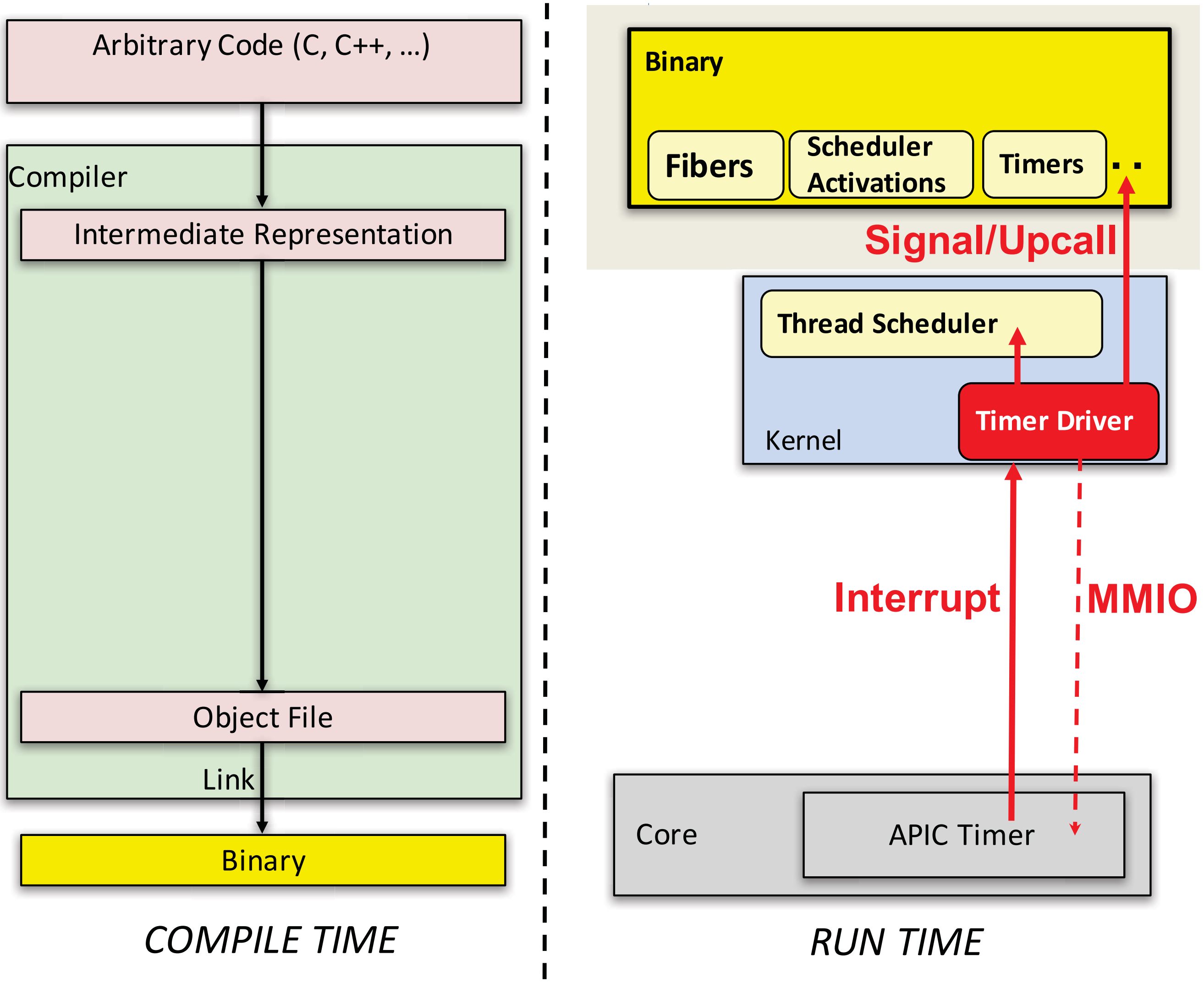
CARAT: Compiler and Runtime-based Address Translation [In Submission Work]

- CARAT integrated into Nautilus Kernel
- Linux-compatible process abstraction…
- …but executable runs as a component of the kernel
- … yet with protection and memory migration available
- All while using physical addressing

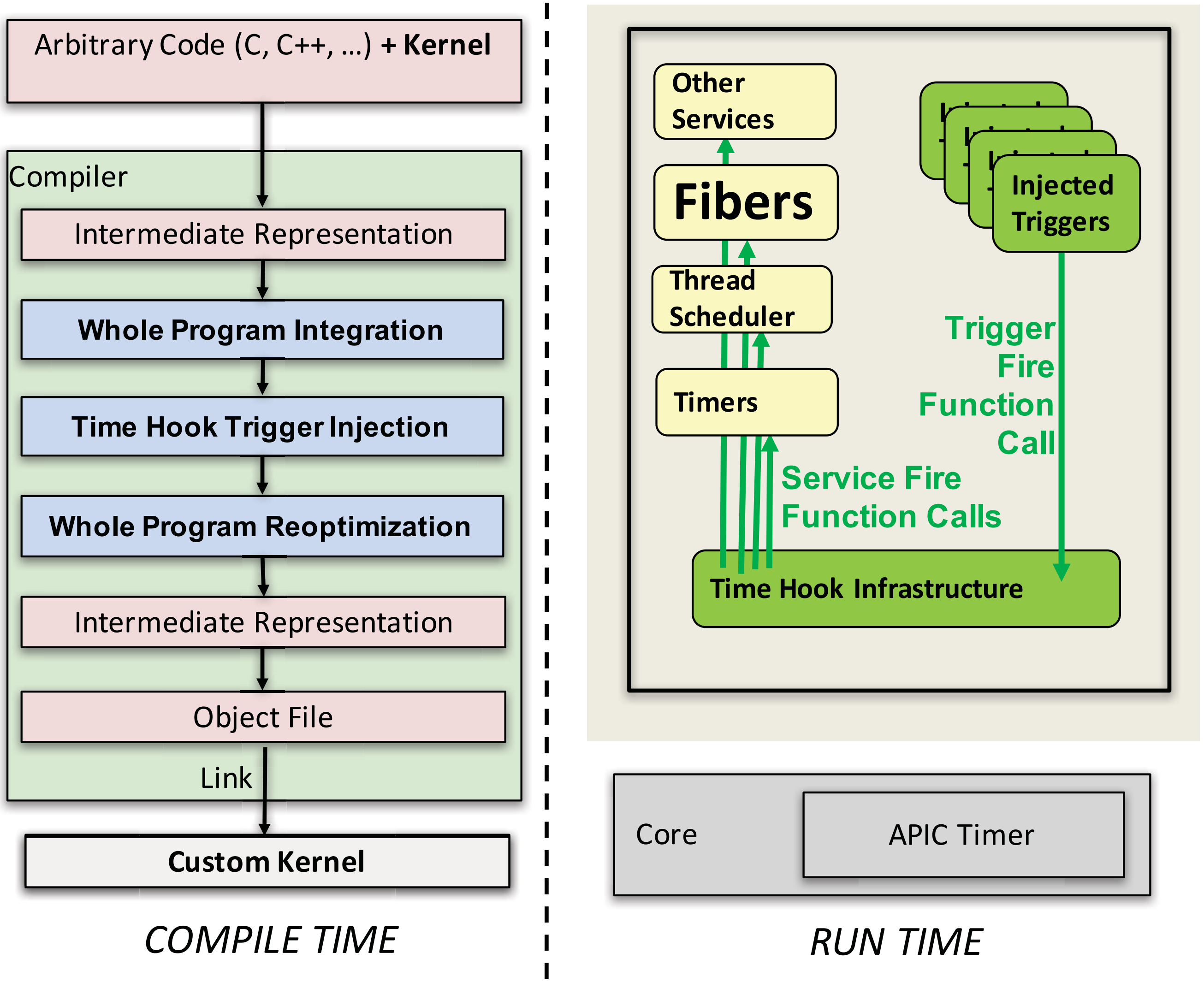
Interweaving Example: Compiler-based Timing (CT)

- Timing is traditionally driven by hardware timer interrupts
  - **Interrupt latency is high and not getting lower**
- Limits granularity of many parallel constructs
  - E.g. preemptive threads
- Can we replace hardware timers with callbacks introduced by the compiler *throughout the kernel and application codebase*?
  - Yes. And achieve similar precision. With 6x lower overhead.
  - Enabling preemptive threads with 4x smaller granularity

Traditional Timing



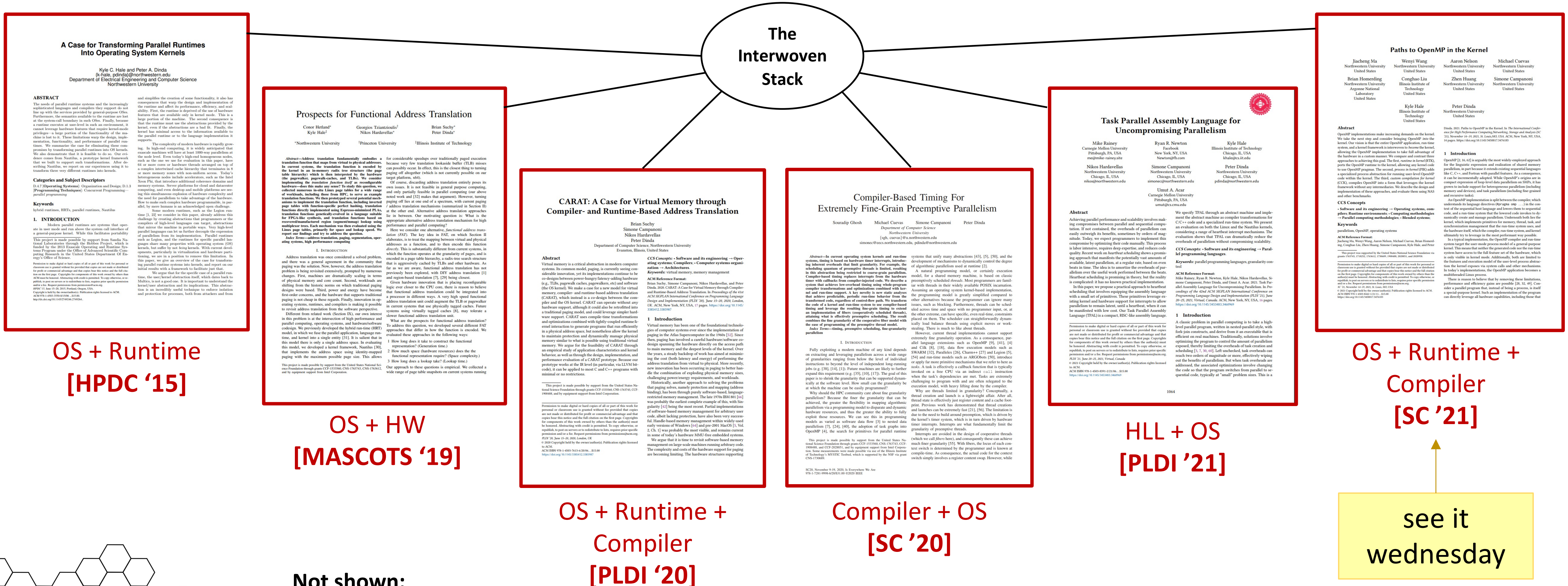
Compiler-based Timing [SC '20]



Northwestern University

ILLINOIS INSTITUTE OF TECHNOLOGY

Find out more…



Not shown:

Hard Real-time Scheduling for Parallelism (OS+Runtime) [HPDC '18]

Fast HW Barriers (HW+OS+Runtime) [HPDC '19]

Fast Queuing (Runtime+Compiler) [MASCOTS '21]

Fast Events (HW+OS) [MASCOTS '18]

…and others

http://interweaving.org

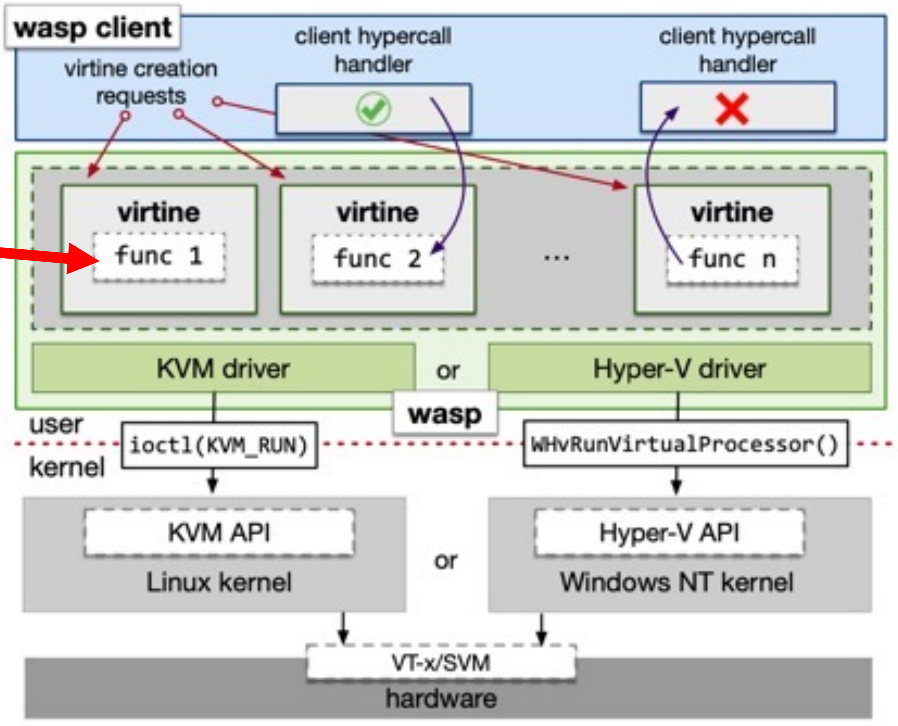
Northwestern University

ILLINOIS INSTITUTE OF TECHNOLOGY

Interweaving Teaser: Function-granularity Virtualization

- More need for single-function execution contexts…even in HPC
- But virtualization platforms not really designed for this

```
virtine int foo() {
    // isolated compute
    return 0;
}
```



Very low overheads for HLL function invocation (FULLY ISOLATED)

Northwestern University

ILLINOIS INSTITUTE OF TECHNOLOGY

Interweaving Teaser: OpenMP

Kern

m (see

c.)

Runtime

(Program

legw

Compilation

Kernel

piler does

gwork)

Paths to OpenMP in the Kernel

Jiacheng Ma, Northwestern University, United States

Wenyi Wang, Northwestern University, United States

Aaron Nelson, Northwestern University, United States

Michael Cuevas, Northwestern University, United States

Brian Homerding, Northwestern University, Argonne National Laboratory, United States

Conghao Liu, Illinois Institute of Technology, United States

Zhen Huang, Northwestern University, United States

Simone Campanoni, Northwestern University, United States

Kyle Hale, Illinois Institute of Technology, United States

Peter Dinda, Northwestern University, United States

Abstract

OpenMP implementations make increasing demands on the ker... We take the next step and consider bringing OpenMP into th... kernel. Our vision is that the entire OpenMP application, run-time system, and a kernel framework is interwoven to *become* the kernel, allowing the OpenMP implementation ... the hardware in a custom manner. We co... approaches to achieving this goal. The first, *runtime in kernel* (RTK), ports the OpenMP runtime to the kernel, allowing any kernel code to use OpenMP pragmas. The second, *process in kernel* (PIK) ... a specialized process abstraction for running user-level ... code within the kernel. The third, *custom compilation for kernel* (CCK), compiles OpenMP into a form that leverages the kernel framework without any intermediaries. We describe the design and implementation of these approaches, and evaluate them using NAS and other benchmarks.

CCS Concepts

• Software and its engineering → Operating systems; Compilers; Runtime environments; • Computing methodologies → Parallel computing methodologies; ...

Keywords

parallelism, OpenMP, operating systems

ACM Reference Format:

Jiacheng Ma, Wenyi Wang, Aaron Nelson, Michael Cuevas, Brian Homerding, Conghao Liu, Zhen Huang, Simone Campanoni, Kyle Hale, and Peter Dinda. 2021. Paths to OpenMP in the Kernel. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21), November 14–19, 2021, St. Louis, MO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3458817.3476183

1 Introduction

OpenMP [2, 40, 62] is arguably the most widely-employed approach for the linguistic expression and realization of shared memory parallelism ... because it extends existing sequential languages like ... Fortran with parallel features. As a consequence, it can be incrementally adopted. While OpenMP's origins are in compact expression of loop-level data parallelism on SMPs, it has grown to include support for heterogeneous parallelism (including memory and devices), and task parallelism (including fine-grained and recursive tasks).

An OpenMP implementation is split between the compiler, which ... (#pragma omp ...) in the context of the sequential host language and lowers them to sequential code, and a run-time system that the lowered code invokes to dy... parallelism. Underneath both lies the kernel, which implements primitives for memory, thread, task, and synchronization management that the run-time system uses, and the hardware itself, which the compiler, run-time system, and kernel ... most performant way possible.

In a typical implementation, the OpenMP compiler and run-time system target the user-mode process model of a general-purpose kernel. This means that neither the generated code nor the run-time

SC '21

Wednesday, 4PM

(227-228)

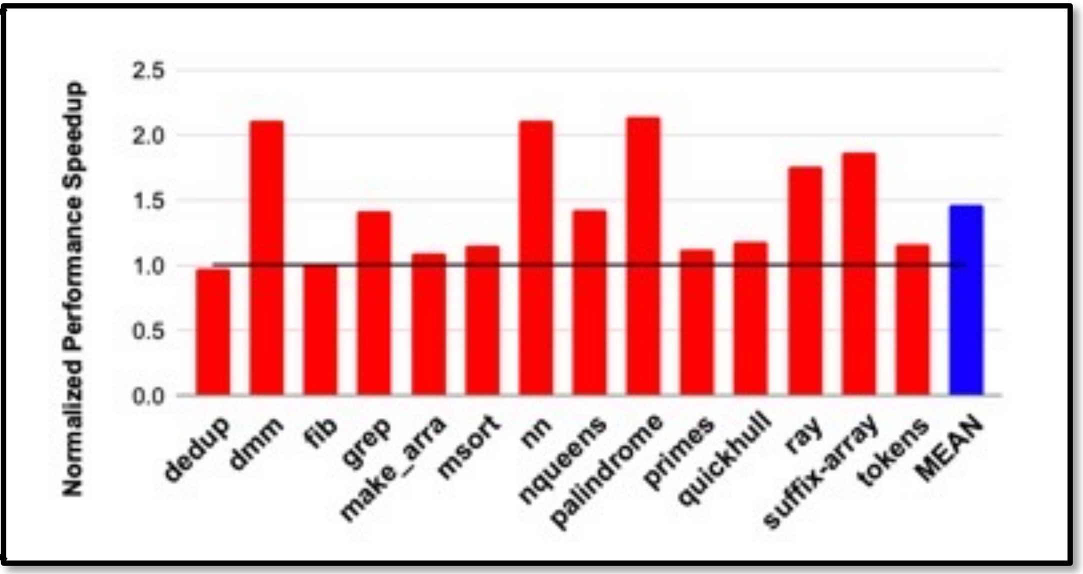
Speedups not the point, but, spoiler alert, there are speedups

What’s Next

- Selective Coherence
- Blended Device Drivers
- Pipeline Interrupts
- Bespoke Execution Contexts
- More emphasis on HLLs

What’s Next

- Selective coherence
- Blended Device Drivers
- Pipeline Interrupts
- Bespoke Execution Contexts
- More emphasis on HLLs

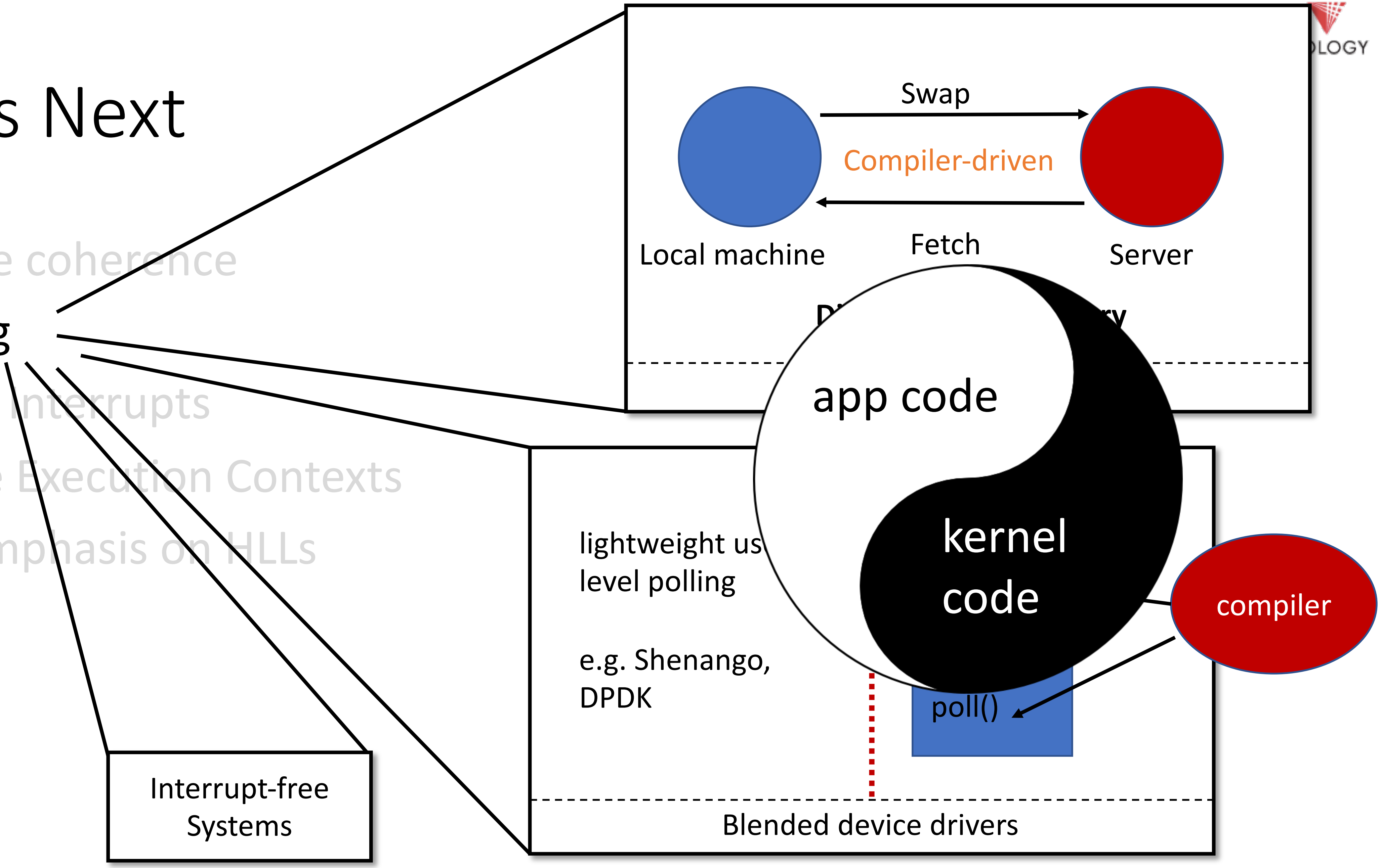


Using high-level program information to inform coherence protocol:

- ~46% speedup
- ~53% interconnect energy reduction

What’s Next

- Selective coherence
- **Blending**
- Pipeline Interrupts
- Bespoke Execution Contexts
- More emphasis on HLLs



Students and Collaborators

- Many students have or are contributing to these efforts
  - Complete list: http://interweaving.org
  - You should hire them!

- Work presented here:
  - Brian Suchy, Mike Wilkins, Souradip Ghosh, Brian Homerding, Jiacheng Ma, Wenyi Ma, Michael Cuevas, Zhen Huang, Conghao Liu, Brian Tauro, Nick Wanninger, Josh Bowden, Enrico Deiana, Vijay Kandihah, Drew Kersnar, Alex Bernat, Gaurav Chaudhary, Siyuan Chai, Kevin McAfee, Kevin Mendoza Tudares

- Heartbeat is in collaboration with Umut Acar, Mike Rainey, and Ryan Newton at CMU; Selective coherence is in collaboration with Umut Acar and Sam Westrick

Check out our OpenMP paper @ SC!

Wednesday, 4PM (227-228)

Thanks to our sponsors:

Thanks!

The Interweaving Project

http://interweaving.org

Thanks to our sponsors:

Northwestern Compiler Group