Hewlett Packard Enterprise

# The Secret Sauce of Vendor-Neutral GPU Programming (in Chapel)

Jade Abraham

May 7th, 2025

## What is the Secret Sauce?

- What does it take to implement a programming language for performant and portable GPU code?
  - Modern programming language
    - Not another C/C++ library
    - First-class parallel programming features
  - A compiler that can target multiple GPU vendors
  - A portable runtime

- Does something exist today that fills this gap?
  - Yes!

## What is Chapel?

**Chapel:** A modern parallel programming language
- portable & scalable
- open-source & collaborative

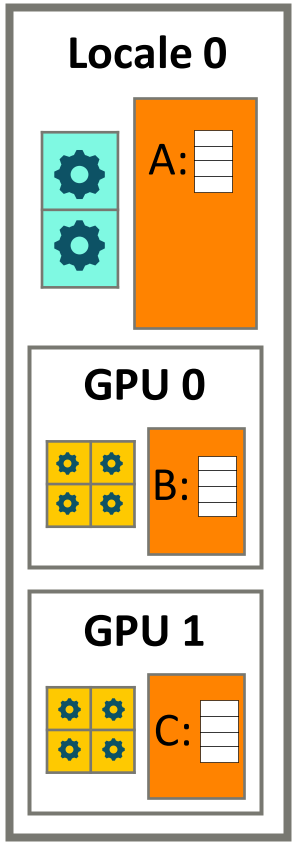
**Goals:**
- Support general parallel programming
- Make parallel programming at scale far more productive

chapel-lang.org

## First-Class Parallel Programming – By Example

🟦 CPU Core 🟨 GPU Core

🟧 Memory



```
var A: [1..10] int;

on Locales[0].gpus[0]
  var B: [1..10] int;
on Locales[0].gpus[1]
  var C: [1..10] int;

forall elem in A do
  elem += 1;

on Locales[0].gpus[0] do
  forall elem in B do
    elem += 1;

on Locales[0].gpus[1] do
  C += 1;
```

Local CPU array allocation

Local GPU array allocation

Compute on all CPUs in parallel

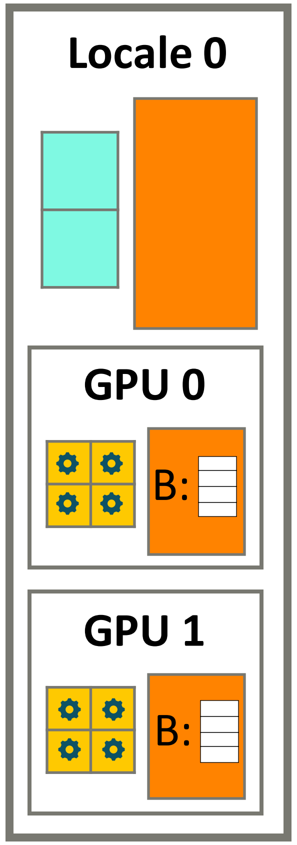
Launch a kernel on a single GPU

Launch a kernel on a single GPU (implicitly parallel)

## Hello, GPUs!

🟦 CPU Core 🟨 GPU Core

🟧 Memory



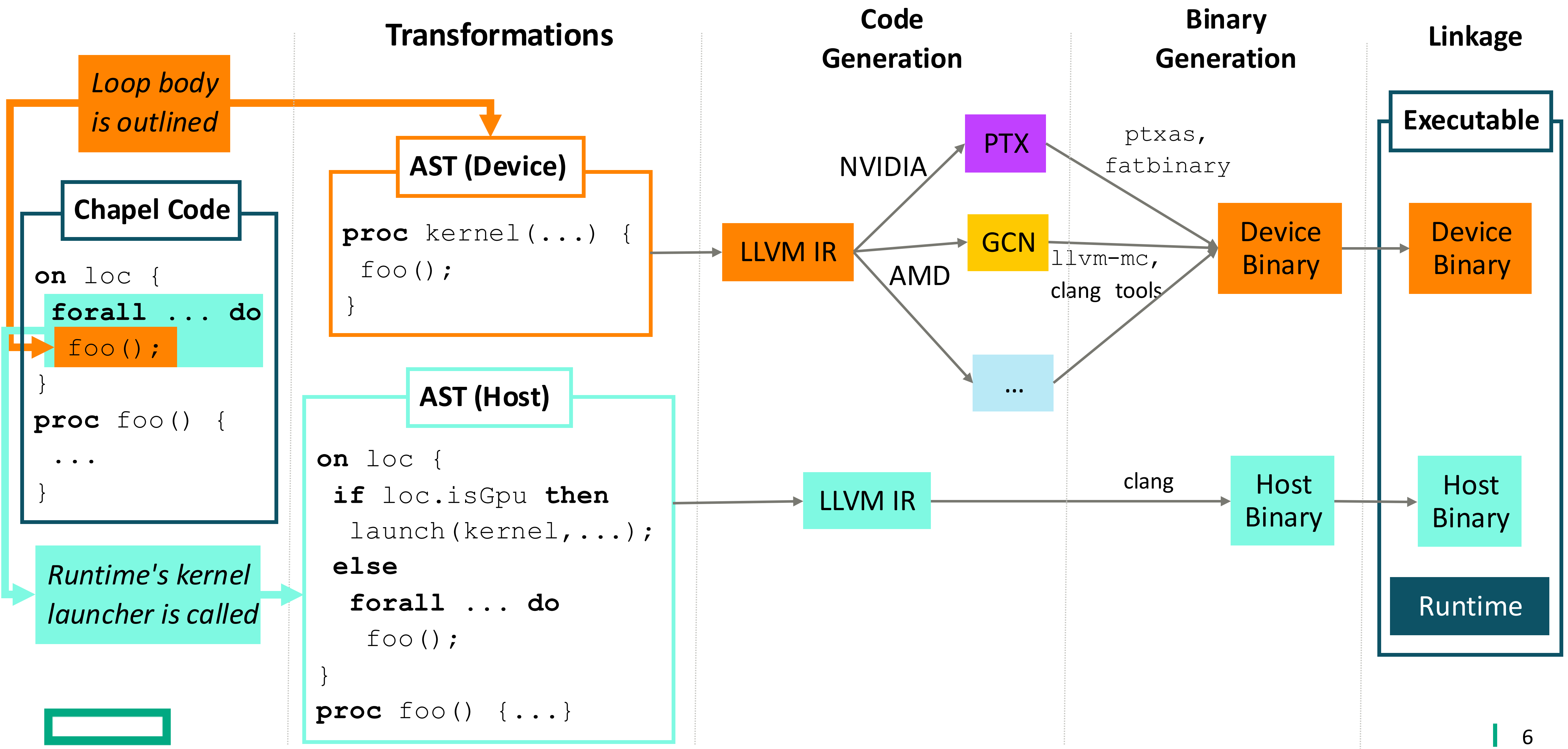
Execute on all GPUs in Serial

```
for gpu in Locales[0].gpus do on gpu {
  var B: [1..10] int;
  B += 1;
}
```

Execute on all GPUs in Parallel

```
coforall gpu in Locales[0].gpus do on gpu {
  var B: [1..10] int;
  B += 1;
}
```

# Portable LLVM-based Compiler



```
on loc {
 forall ... do
  foo();
}
proc foo() {
 ...
}
```

```
proc kernel(...) {
 foo();
}
```

```
on loc {
 if loc.isGpu then
  launch(kernel,...);
 else
  forall ... do
   foo();
}
proc foo() {...}
```

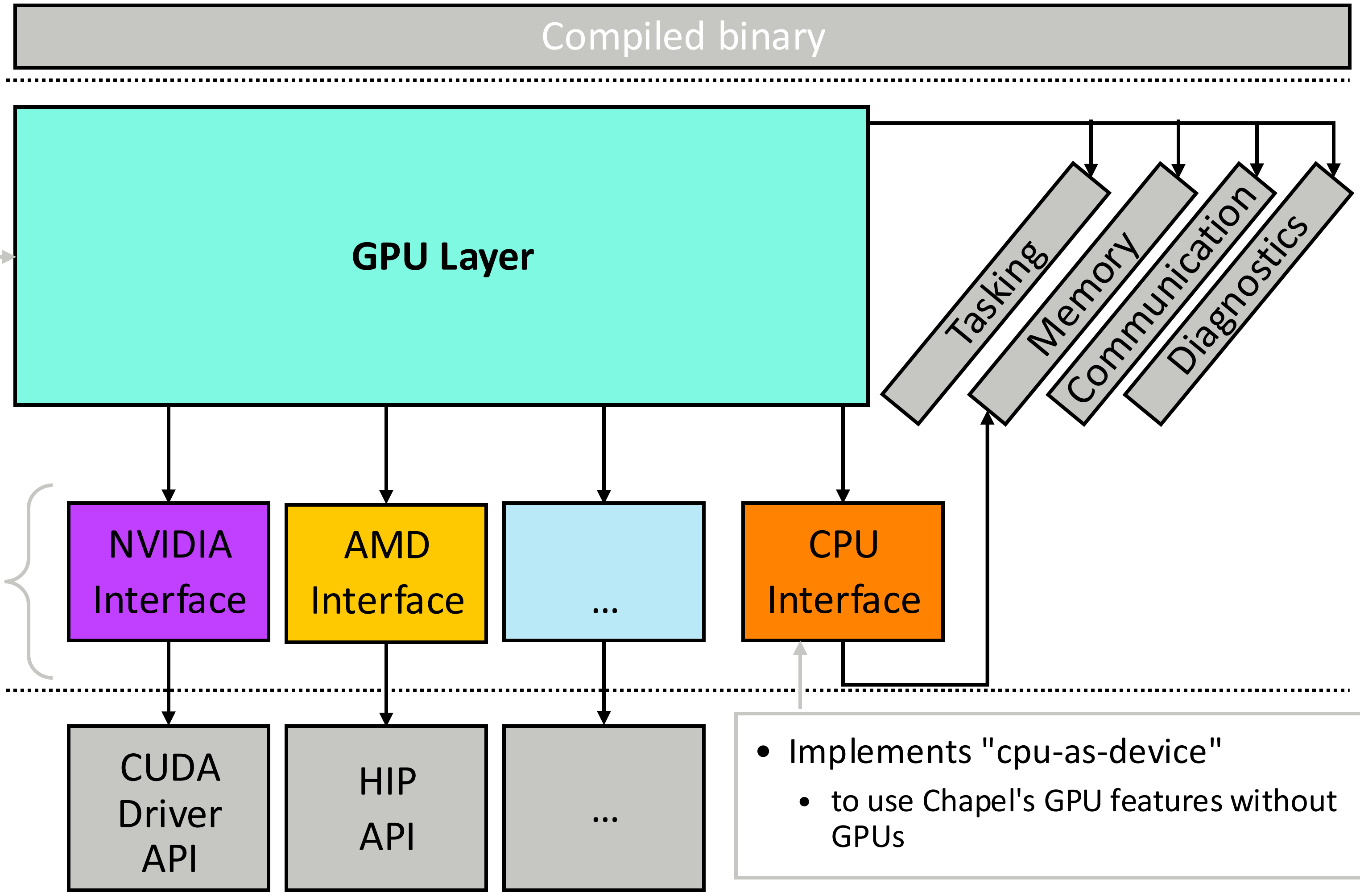
# Extensible Runtime Architecture

**Interface for:**

- Compiler-injected calls
  - e.g. kernel prep and launch
- Extern calls from modules
  - e.g. memory management, data movement

**Interacts with the rest of the runtime to:**

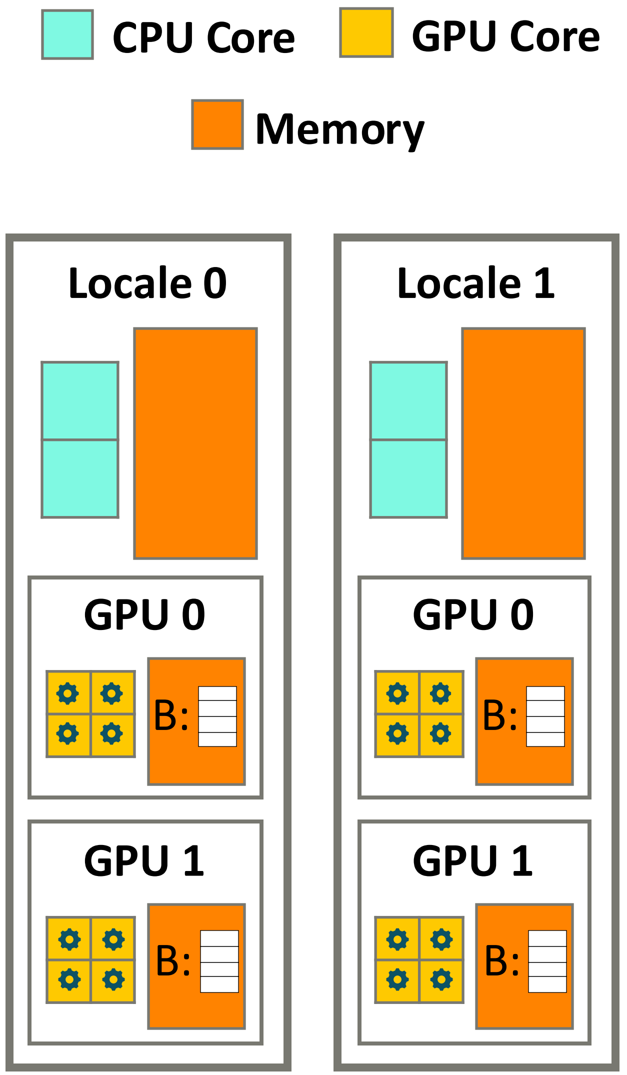
- Maintain task-private data
  - e.g. GPU streams
- Make host-based allocations
- Move data across locales
- Trigger diagnostics



- Thin layer for primitive GPU operations
  - e.g. call a kernel, initialize driver, query info
- Wraps around drivers

- Implements "cpu-as-device"
  - to use Chapel's GPU features without GPUs

# Bonus! Hello, Distributed GPUs!



```
coforall loc in Locales do on loc {
  coforall gpu in loc.gpus do on gpu {
    var B: [1..10] int;
    B += 1;
  }
}
```

## More about Chapel + GPUs

- How Does Chapel’s GPU Support Work?
  - A more in-depth look at Chapel’s GPU internals
  - https://www.youtube.com/watch?v=J0av4VJbS4o

- Chapel Runtime Overview
  - How the rest of Chapel’s runtime handles threading, remote communication, memory management, and more
  - https://www.youtube.com/watch?v=rC4Oz654bsU

- The Game of Life: A multi-GPU implementation in Chapel
  - A larger example of programming GPUs in Chapel
  - This video is part of a GPU series with other coding examples
  - https://www.youtube.com/watch?v=U96mA84KIqo

# Ways to Engage with the Chapel Community

## Live/Virtual Events

- ChapelCon (formerly CHIUW), annually
- Chapel project meeting, weekly

## Community / User Forums

- Discord
- Discourse
- Email Contact Alias `chapel+qs@discoursemail.com`
- GitHub Issues
- Gitter
- Reddit
- Stack Overflow

## Electronic Broadcasts

- Chapel Blog, ~biweekly
- Community Newsletter, quarterly
- Announcement Emails, around big events

## Social Media

- Bluesky
- Facebook
- LinkedIn
- Mastodon
- X / Twitter
- YouTube

## Thank you

https://chapel-lang.org
@ChapelLanguage