Leveraging Heterogeneous Computing with TornadoVM and oneAPI

Juan Fumero, PhD
Research Fellow at The University of Manchester, UK
@snatverk

MANCHESTER 1824
The University of Manchester

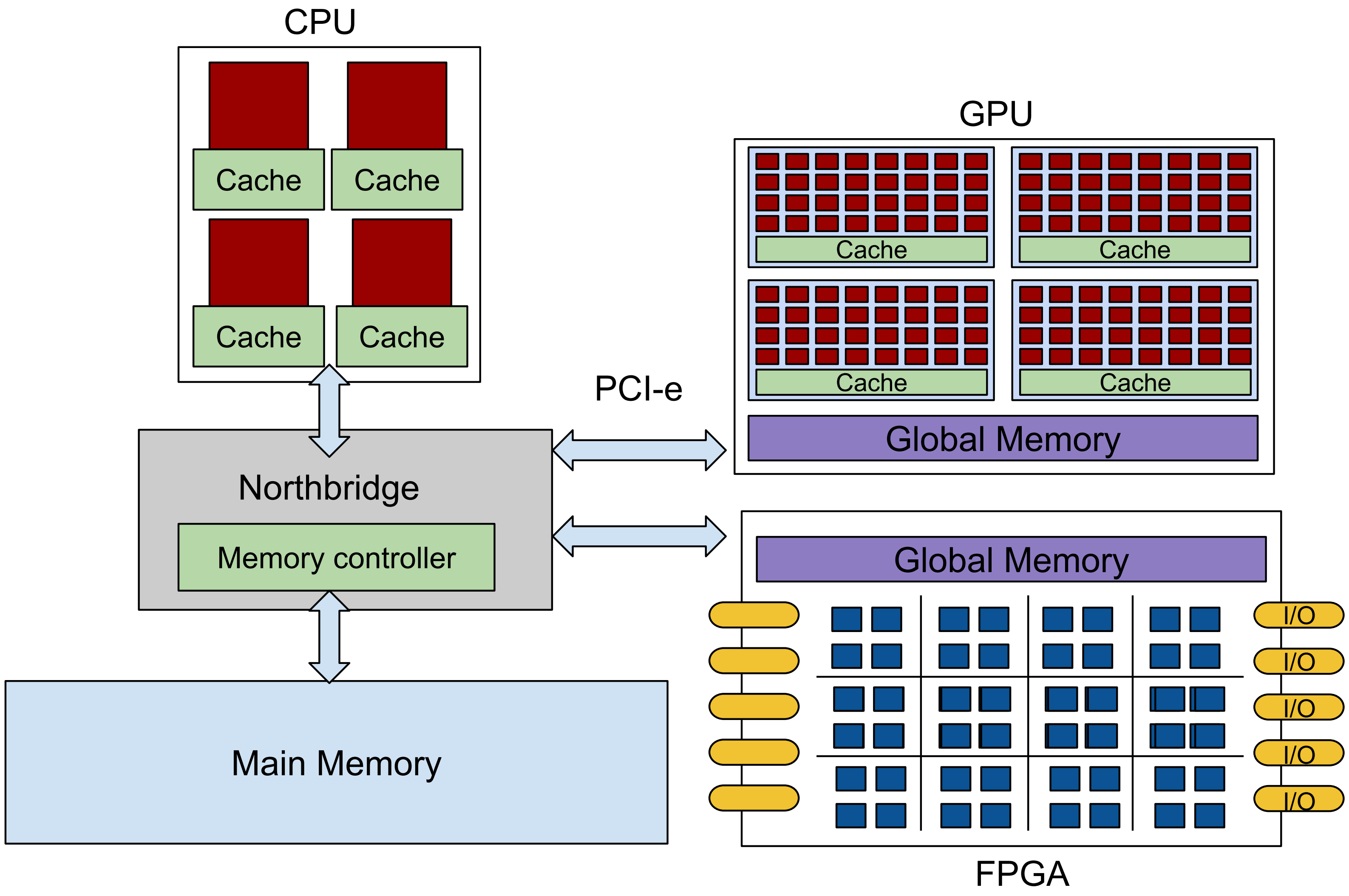
oneAPI Language SIG
22nd June 2023

TORNADOVM

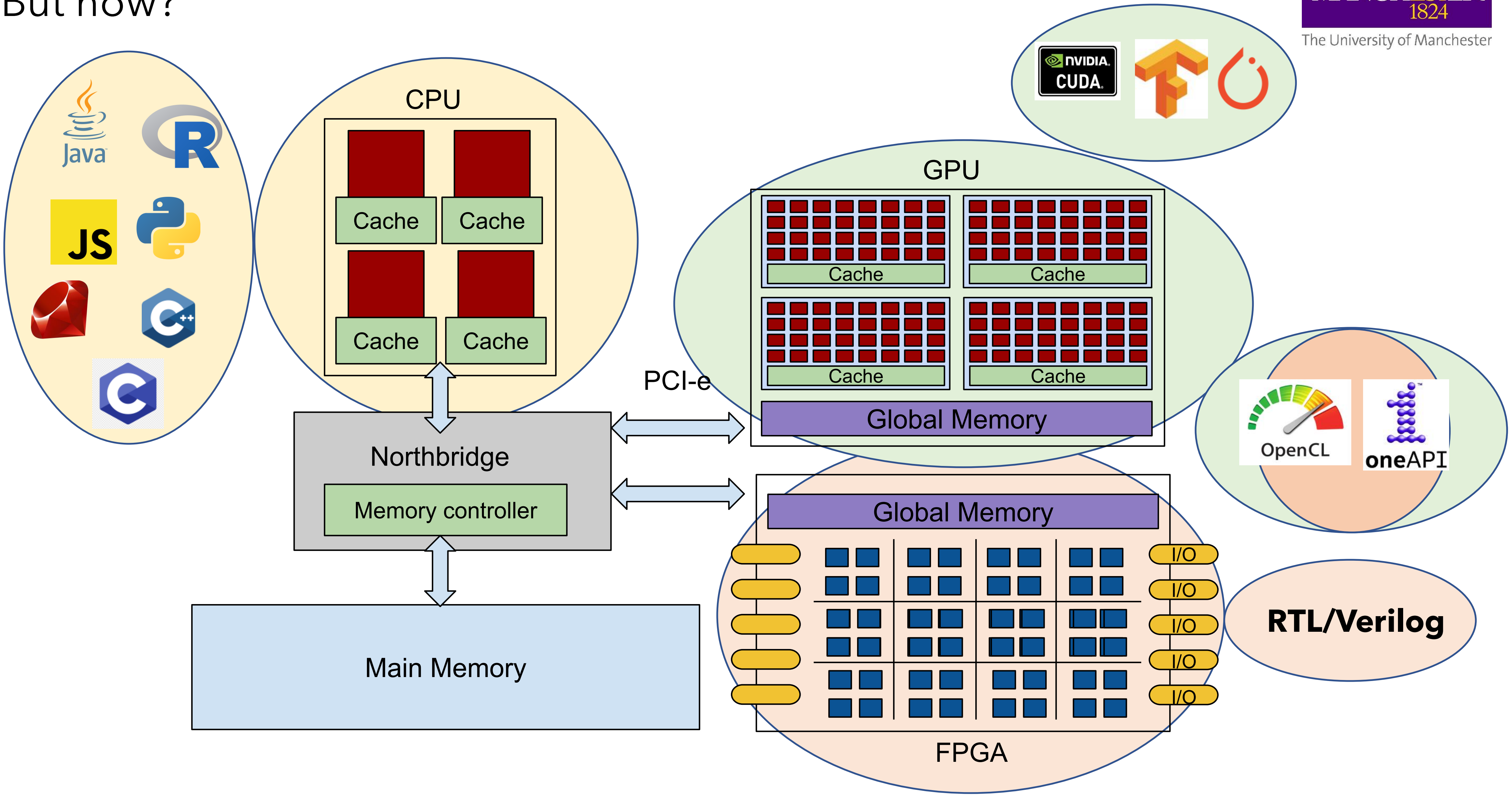
Outline

1. Motivation & Quick Overview of TornadoVM
2. TornadoVM's main abstractions for CPUs, GPUs & FPGAs
   1. User API abstractions
   2. Multi-backend Runtime System
   3. Multi-backend JIT Compilers
3. **Lessons learnt**: Approaches for Memory Manager for Het. Managed Runtimes
4. **Generating Value** -> Open Sourcing Internal Libraries
   1. SPIR-V Code Gen
   2. Level Zero Java JNI
5. **Ideas/Feedback** to the LevelZero Software Stack

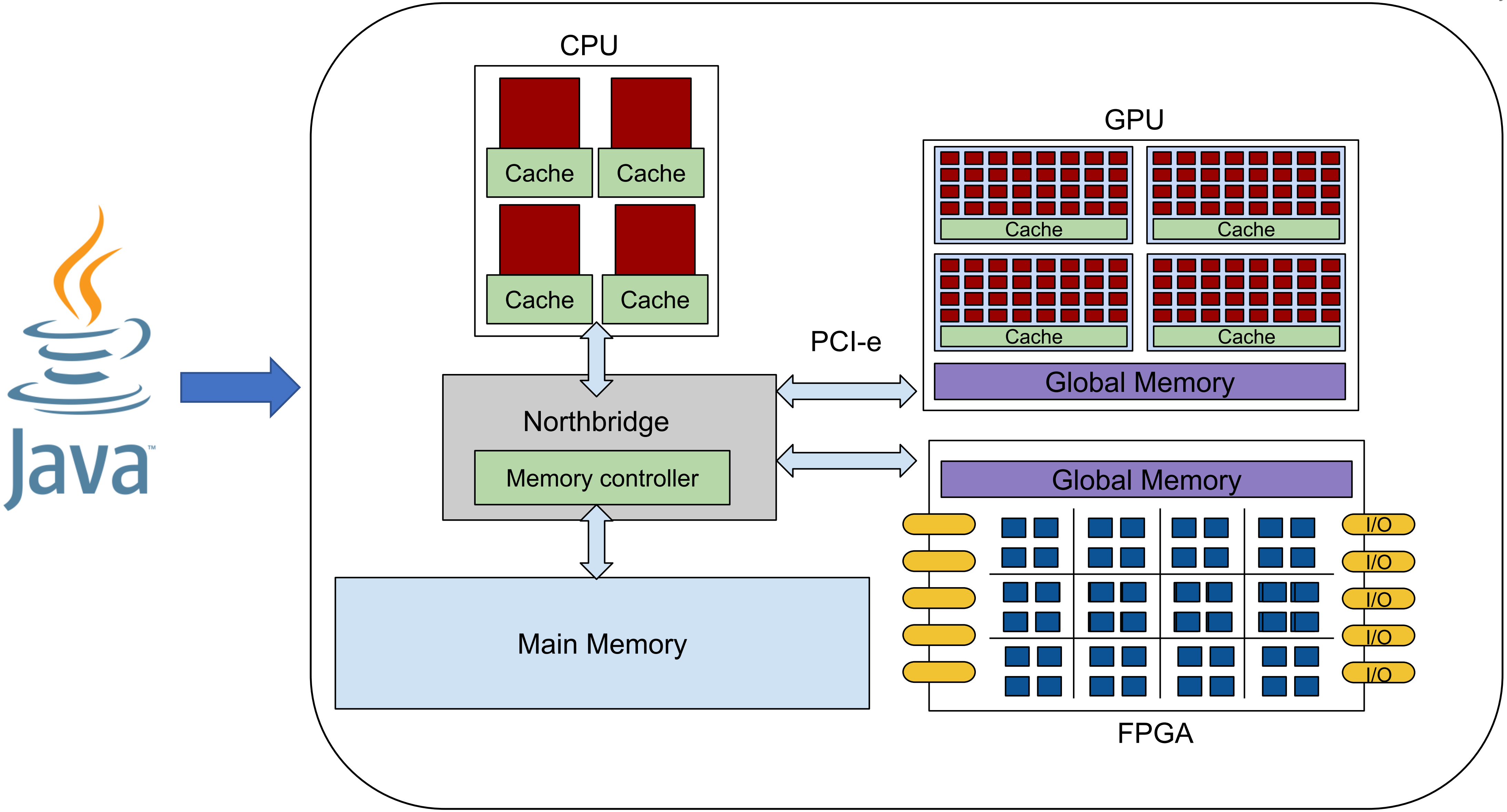
We could potentially use ALL devices!



But how?

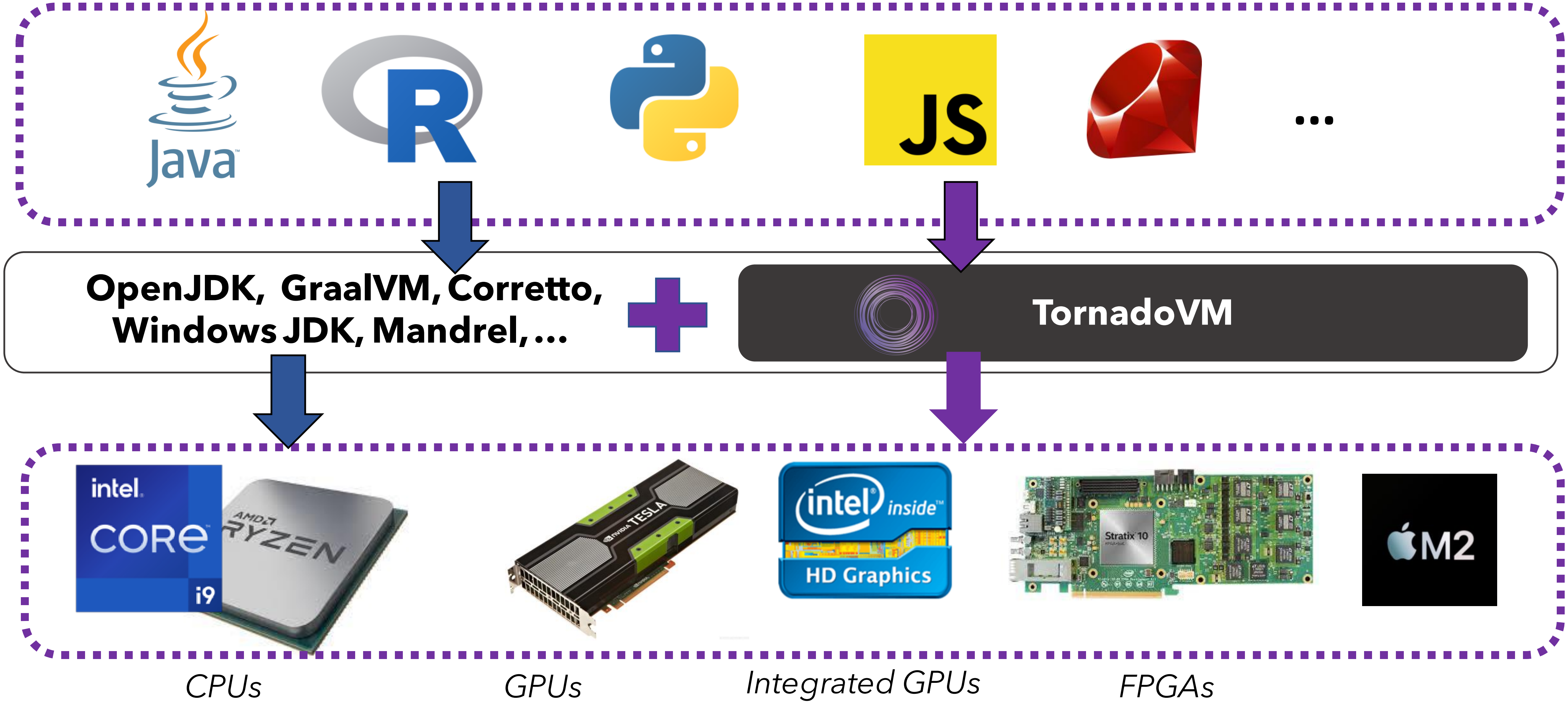

Java
JS
CPU
Cache
Cache
Cache
Cache
Northbridge
Memory controller
Main Memory
PCI-e
NVIDIA
CUDA
GPU
Cache
Cache
Cache
Cache
Global Memory
OpenCL
oneAPI
Global Memory
I/O
I/O
I/O
I/O
I/O
FPGA
RTL/Verilog

What if we could use a single High-Level Programming Language?



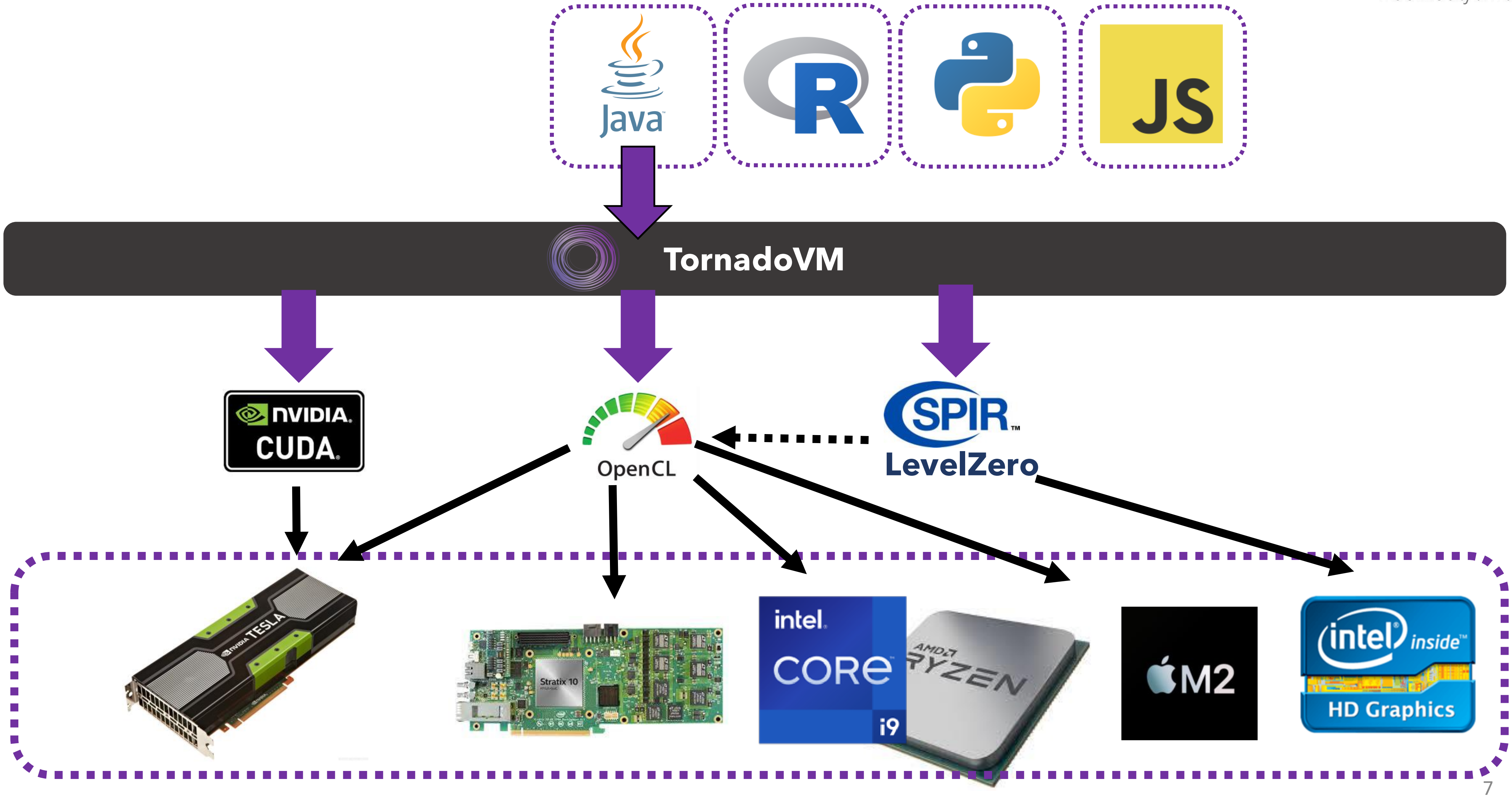
Fast Path to GPUs and FPGAs

MANCHESTER 1824
The University of Manchester



Enabling Acceleration for Managed Runtime Languages

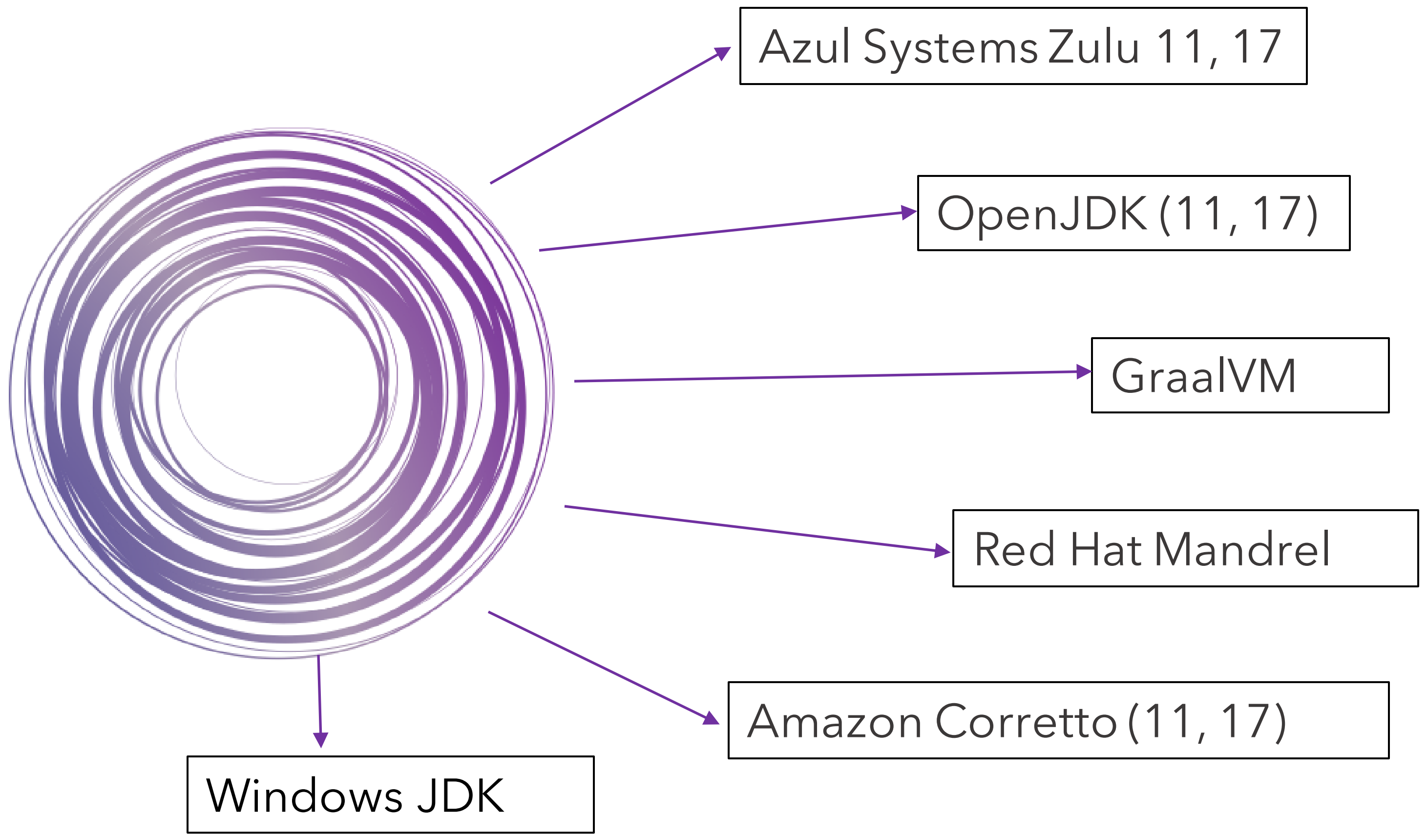
MANCHESTER 1824
The University of Manchester



TornadoVM Overview

www.tornadovm.org

https://github.com/beehive-lab/TornadoVM



Features such as:

- Dynamic Migration
- Cloud deployment | AWS
- Multiple devices | CPU, GPU, FPGA
- Multi-backend | OCL, CUDA, SPIR-V
- JIT compilation specialization
- Hardware Agnostic APIs
- And more …

TornadoVM is Open Source

How do we Program with TornadoVM? User APIs

Different components of the User API

a) How to represent parallelism within functions/methods?

- A.1: Java annotations for expressing parallelism (**@Parallel, @Reduce**) for Non-Experts
- A.2: Kernel API for GPU experts (use of **kernel context** object)

b) How to define which methods to accelerate?

Build a **Task-Graph API** to define data In/Out and the code to be accelerated

c) How to explore different optimizations?

Execution Plan

Tornado API – example using Annotations

MANCHESTER 1824
The University of Manchester

```java
class Compute {
  public static void mxm(Matrix2DFloat A, Matrix2DFloat B,
                         Matrix2DFloat C, final int size) {
     for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
           float sum = 0.0f;
           for (int k = 0; k < size; k++) {
              sum += A.get(i, k) * B.get(k, j);
           }
           C.set(i, j, sum);
        }
     }
  }
}
```

Tornado API – example using Annotations

MANCHESTER 1824
The University of Manchester

```java
class Compute {
  public static void mxm(Matrix2DFloat A, Matrix2DFloat B,
                         Matrix2DFloat C, final int size) {
     for (@Parallel int i = 0; i < size; i++) {
        for (@Parallel int j = 0; j < size; j++) {
           float sum = 0.0f;
           for (int k = 0; k < size; k++) {
              sum += A.get(i, k) * B.get(k, j);
           }
           C.set(i, j, sum);
        }
     }
  }
}
```

We add the parallel annotation as a hint for the compiler

We only have 2 annotations:

- **@Parallel**
- **@Reduce**

+ A light API to identify which methods to accelerate

Tornado API – example using Kernel Context

```
class Compute {
  public static void mxm(Matrix2DFloat A, Matrix2DFloat B,
                         Matrix2DFloat C, final int size,
                         KernelContext context) {
     int idx = context.globalIdx;
     int jdx = context.globalIdy;
     float sum = 0.0f;
     for (int k = 0; k < size; k++)
        sum += A.get(i, k) * B.get(k, j);
     C.set(i, j, sum);
  }
}
```

Kernel-Context accesses thread ids, local memory and barriers

It needs a **Grid of Threads** to be passed during the kernel launch

Tornado API – example

MANCHESTER 1824
The University of Manchester

How to identify which methods to accelerate? --> TaskGraph

```
TaskGraph taskGraph = new TaskGraph("s0")
   .transferToDevice(DataTransferMode.EVERY_EXECUTION , matrixA, matrixB)
   .task("t0", objectCompute::mxm, matrixA, matrixB, matrixC, size)
   .transferToHost(DataTransferMode.EVERY_EXECUTION, matrixC);
```

Host Code

Task-Graph is a new Tornado object exposed to developers to define :

a) **The code to be accelerated** (which Java methods?)

b) **The data (Input/Output)** and how data should be streamed

Adding Execution Plans

How to explore different optimizations? --> ExecutionPlan

```java
ImmutableTaskGraph itg = taskGraph.snapshot();

TornadoExecutionPlan executionPlan = new TornadoExecutionPlan(itg);

executionPlan.withWarmUp()
             .withProfiler(ProfilerMode.CONSOLE)
             .withDynamicReconfiguration(PERFORMANCE, PARALLEL);

TornadoExecutionResult result = executionPlan.execute();

long elapsedKernelTime = result.getProfiler().getDeviceKernelTime(); // in ns
```

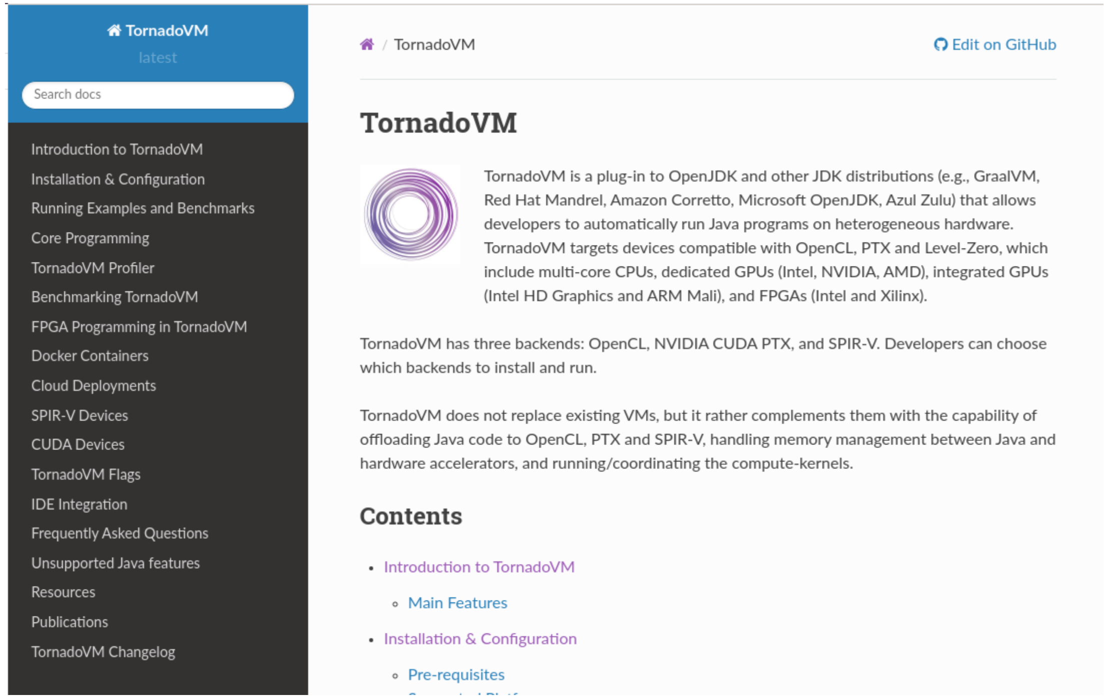
Host Code

Optional High-Level Optimization Pipelines:

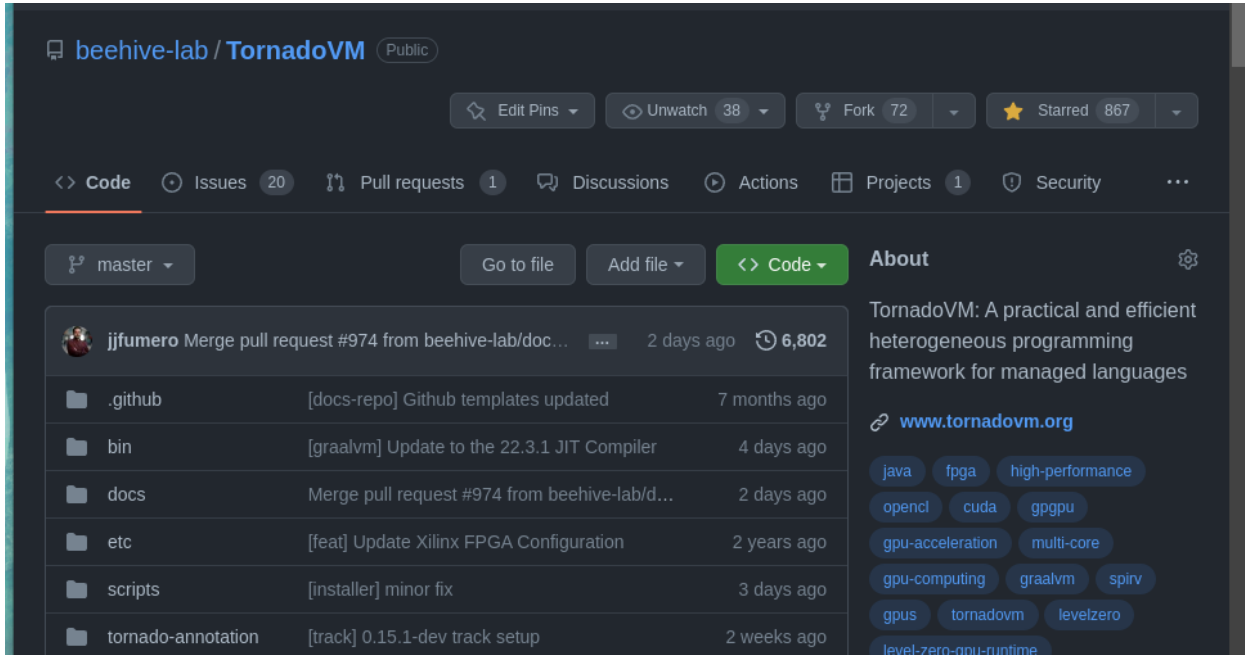
- Enable/Disable Profiler
- Enable Warmup
- Enable Dynamic Reconfiguration
- Enable Batch Processing
- Enable Thread Scheduler (no need for recompilation for different grids schedulers)

To know more about the APIs

MANCHESTER 1824
The University of Manchester



https://tornadovm.readthedocs.io/en/latest/



https://github.com/beehive-lab/TornadoVM

Runtime System and JIT Compilers

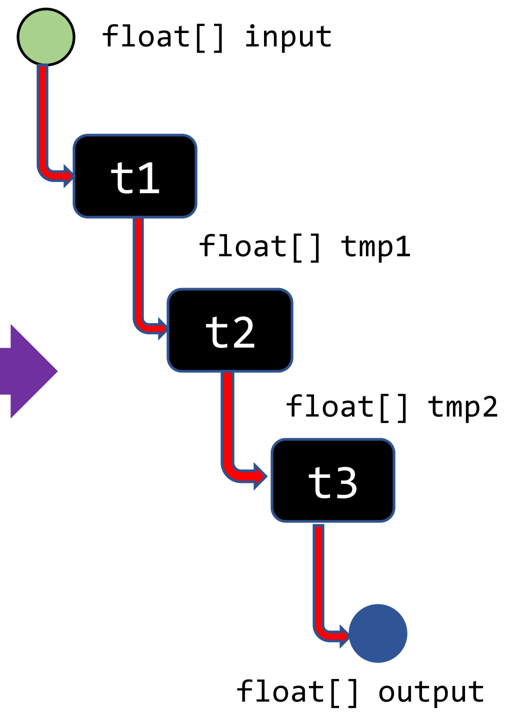
The Main Abstraction: TornadoVM Bytecode + Immediate Actions

MANCHESTER 1824
The University of Manchester

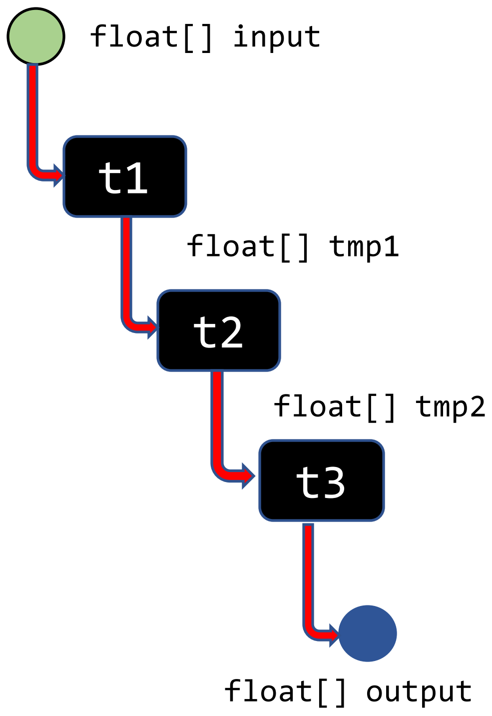
```
public class Sample {
      public void task1(float[] input, float[] tmp1) {…}
      public void task2(float[] tmp1,  float[] tmp2) {…}
      public void task3(float[] tmp2,  float[] output) {…}

      public void buildTaskGraphAndPlan(float[] input, float[] output) {
         TaskGraph tg = new TaskGraph("sample");
         tg.transferToDevice(DataTransferMode.EVERY_EXECUTION, input, out1, out2)
           .task("t1", this::task1, input, tmp1)
           .task("t2", this::task2, tmp1, tmp2)
           .task("t3", this::task3, tmp2, output)
           .transferToHost(DataTransferMode.EVERY_EXECUTION, output);

        ImmutableTaskGraph itg = tg.snapshot();
        TornadoExecutionPlan plan = new TornadoExecutionPlan(itg);
        plan.execute();
      }
}
```

[RUNTIME] Build a Data-Flow Graph

```
public class Sample {
      public void task1(float[] input, float[] tmp1) {…}
      public void task2(float[] tmp1,  float[] tmp2) {…}
      public void task3(float[] tmp2,  float[] output) {…}

      public void buildTaskGraphAndPlan(float[] input, float[] output) {
         TaskGraph tg = new TaskGraph("sample");
         tg.transferToDevice(DataTransferMode.EVERY_EXECUTION, input)
           .task("t1", this::task1, input, tmp1)
           .task("t1", this::task1, tmp1, tmp2)
           .task("t1", this::task1, tmp2, output)
           .transferToHost(DataTransferMode.EVERY_EXECUTION, output);

        ImmutableTaskGraph itg = tg.snapshot();
        TornadoExecutionPlan plan = new TornadoExecutionPlan(itg);
        plan.execute();
      }
}
```

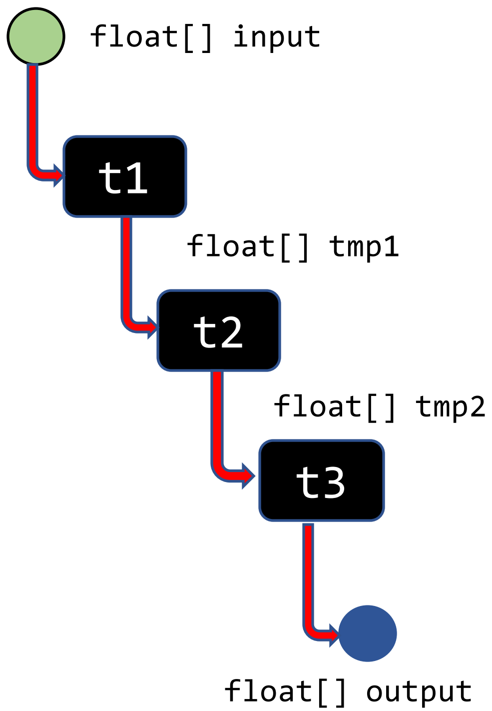


[RUNTIME] Generate BC From DFG



[RUNTIME] Generate BC From DFG

One Java Thread Per VM instance

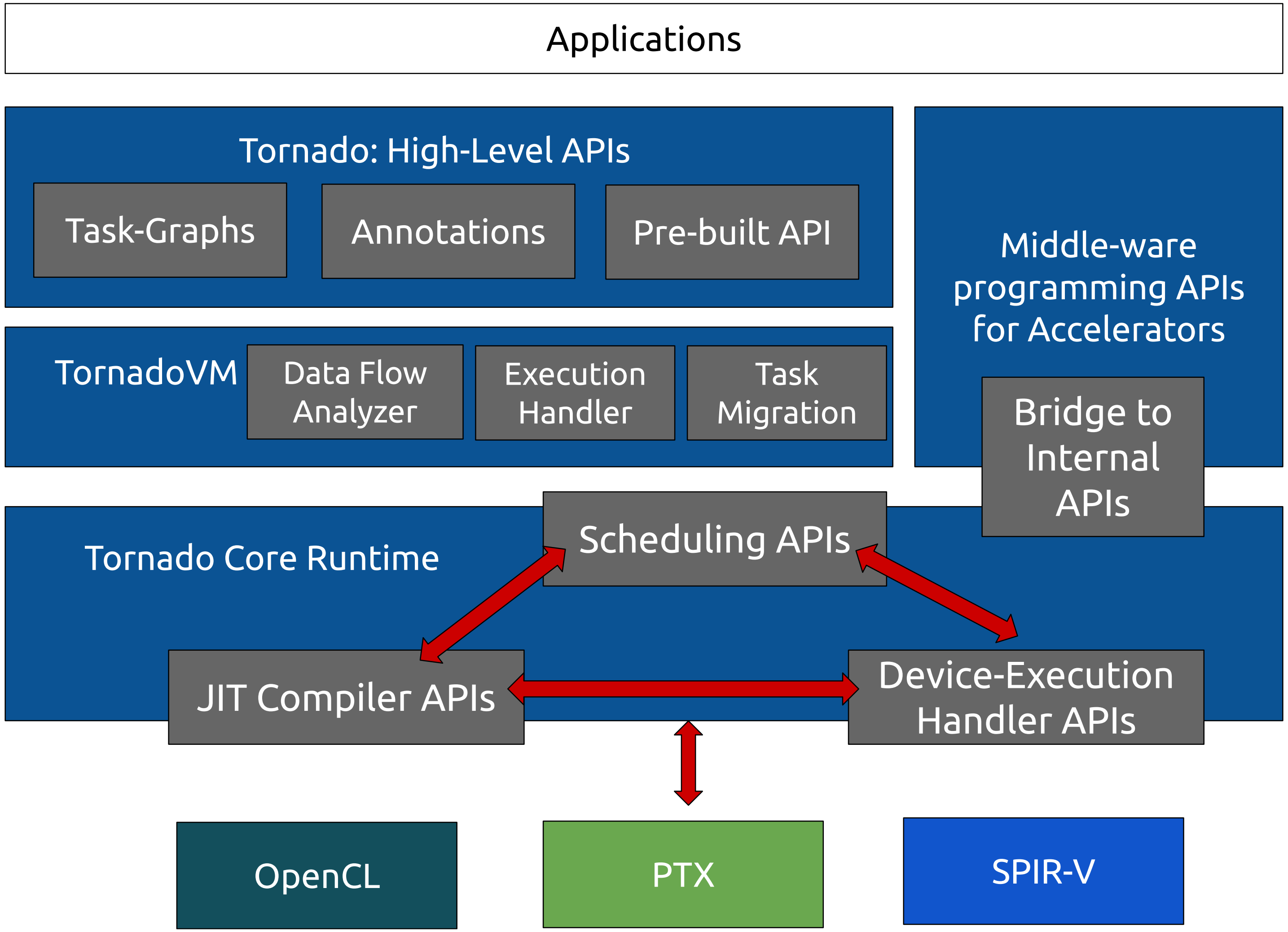



```
BEGIN CONTEXT <ID>
   ALLOC input
   ALLOC tmp1
   ALLOC tmp2
   ALLOC output
   TRANSFER_TO_DEVICE, INPUT, SIZE, OFFSET:0
     LAUNCH T1, INPUT, TMP1
   BARRIER
     LAUNCH T2, TMP1, TMP2
   BARRIER
     LAUNCH T3, TMP2, OUTPUT
   TRANSFER_TO_HOST, OUTPUT, SIZE, OFFSET:0
   BARRIER
   DEALLOC INPUT
   DEALLOC TMP1
   DEALLOC TMP2
   DEALLOC OUTPUT
END
```

TornadoVM can reorder bytecode and repeat patterns (e.g., batch processing), under demand

Dynamic Application Reconfiguration on Heterogeneous Hardware. VEE'19

[RUNTIME] Hardware Agnostic Middle-ware APIs



The TornadoVM BC Interpreter makes calls to the middle-ware API

Multi-backend JIT Compiler Workflow

Programmer's view

```
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```

Multi-backend JIT Compiler Workflow

Programmer's view

```
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```

javac

Java Bytecodes

Static Compilation: No Modifications in Javac

TornadoVM JIT Compiler

Multi-backend JIT Compiler Workflow

Programmer's view

```java
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```

javac

Java Bytecodes

TornadoVM JIT Compiler

Graal IR

TornadoVM Common IR

Multi-backend JIT Compiler Workflow

Programmer's view

```
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```

javac

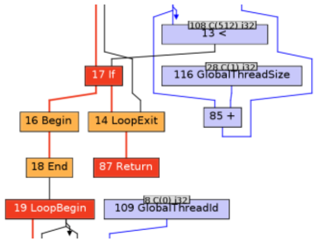
Java Bytecodes

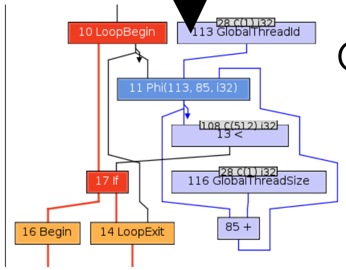
TornadoVM JIT Compiler

Graal IR

TornadoVM Common IR -> **Sketch**

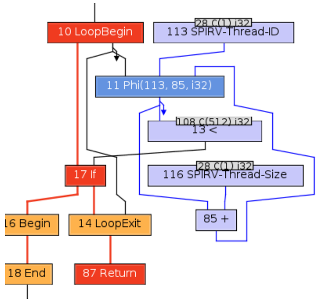
TornadoVM IR for PTX





TornadoVM IR for OpenCL

TornadoVM IR for SPIR-V

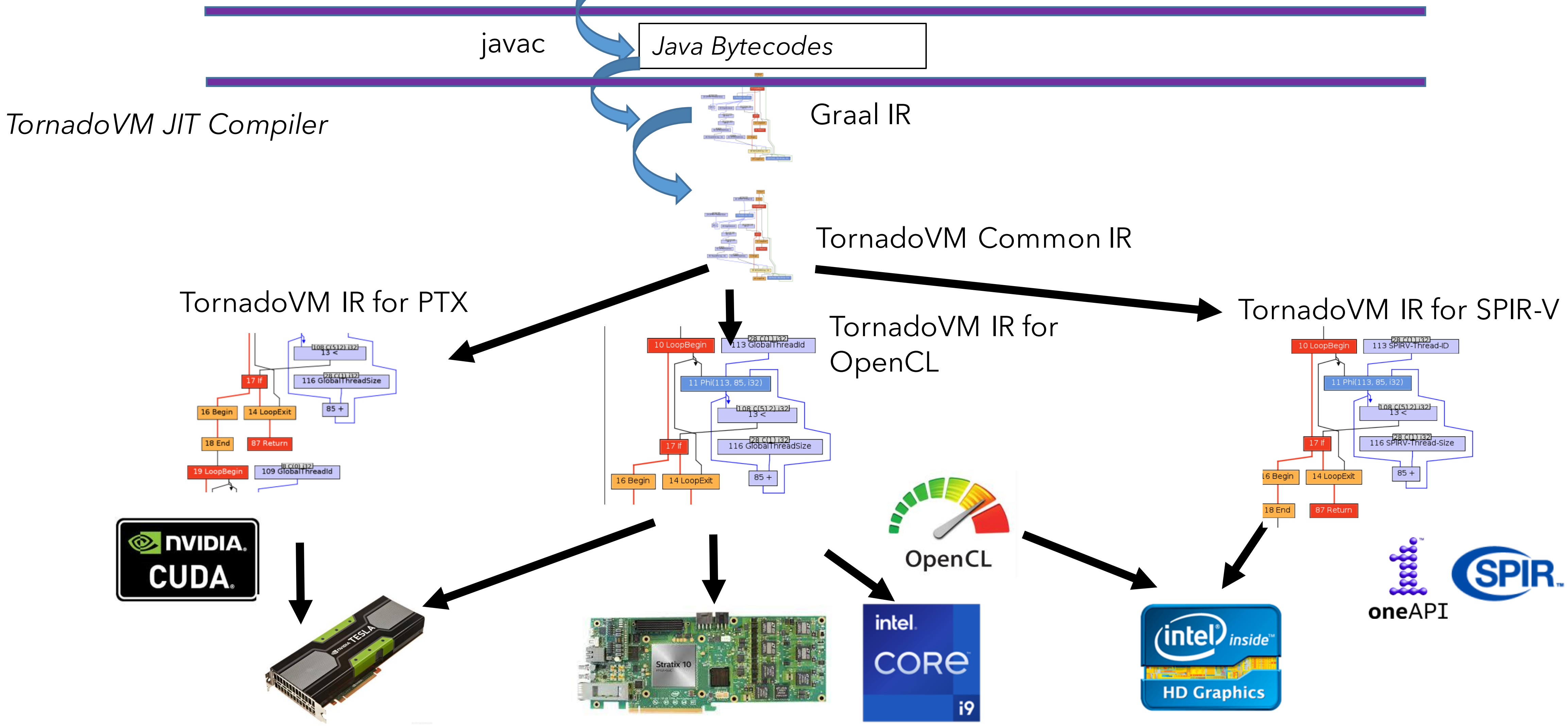


Multi-backend JIT Compiler Workflow

The University of Manchester

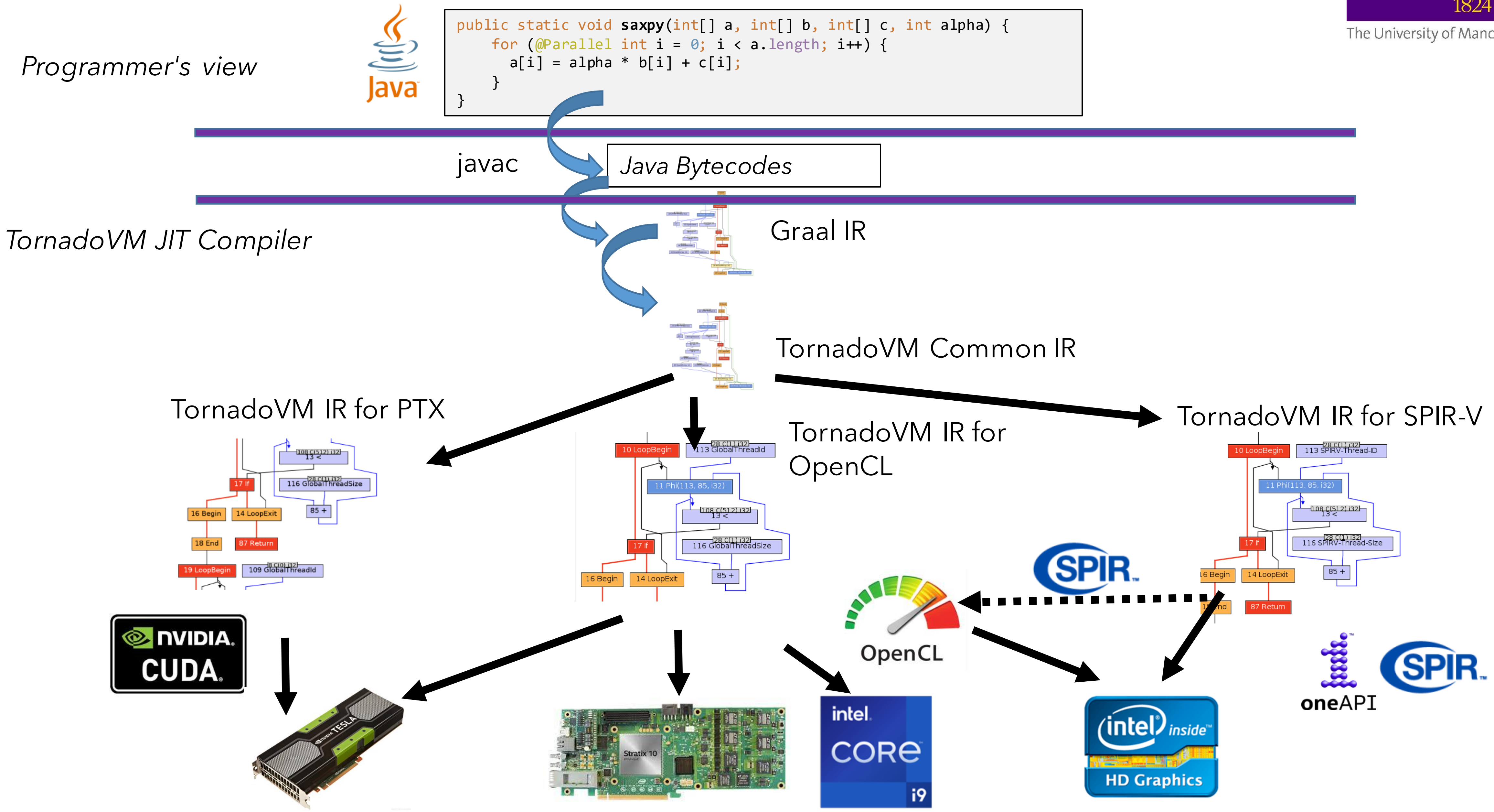
Programmer's view

```java
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```



Multi-backend JIT Compiler Workflow

MANCHESTER 1824
The University of Manchester



Memory Handling for Heterogenous & Managed Runtime Systems

Some research directions

Memory Handler for Het. Managed Runtime Systems

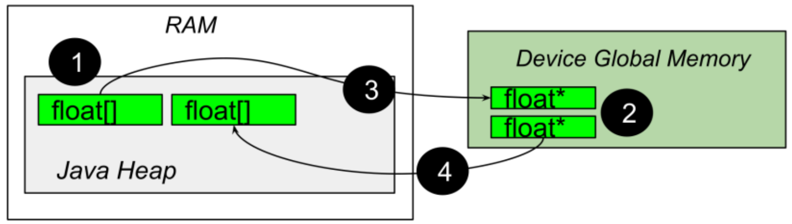
1. Efficient Memory Handler is as important as a JIT compiler in the context of Managed Runtime Systems to access GPUs, FPGAs, etc
   - **[ON-HEAP]** Data resides in a managed heap (Java Heap)
     - Data must be copied from the Java heap to a host buffer <and|or> device memory.
     - Thus, data migration could be more expensive than traditional oneAPI/OpenCL/CUDA programs
     - We must be very careful with when the GC operates (if it moves data while the GPU kernel is running, we might get SegFaults) --> **GC must be stopped.**

     - UNLESS: **Off-heap** memory is used
       - Objects are not managed by the JVM anymore.

2. We have worked on both implementations

On-Heap Data Structures (TornadoVM's current approach)

The University of Manchester



1. Memory reserved in the Java Heap
2. Device Buffer Malloc (e.g., GPU)
3. Data Transfer (host->device)
4. **Kernel Execution**
5. Data Transfer (device -> host)

Good Luck the GC not moving pointers

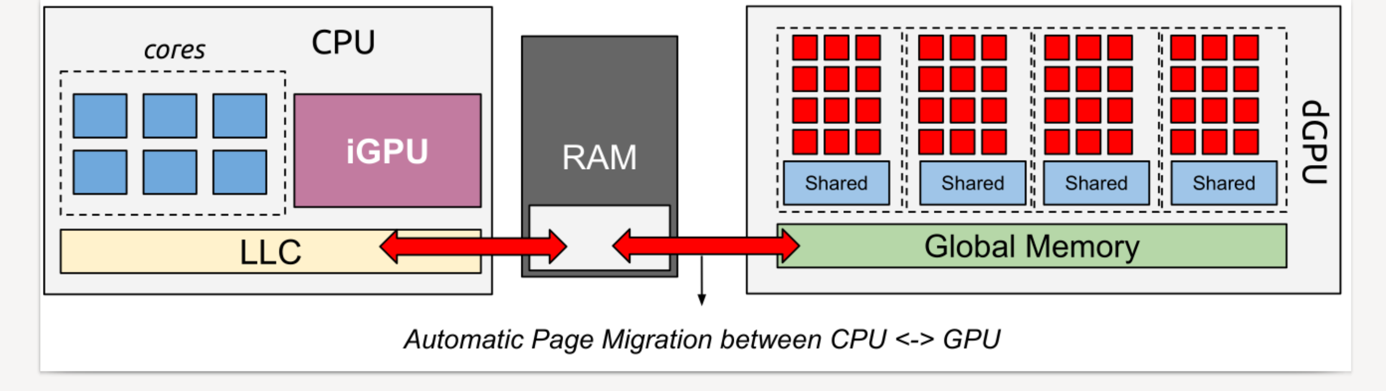
LOCK GC and blocking operations

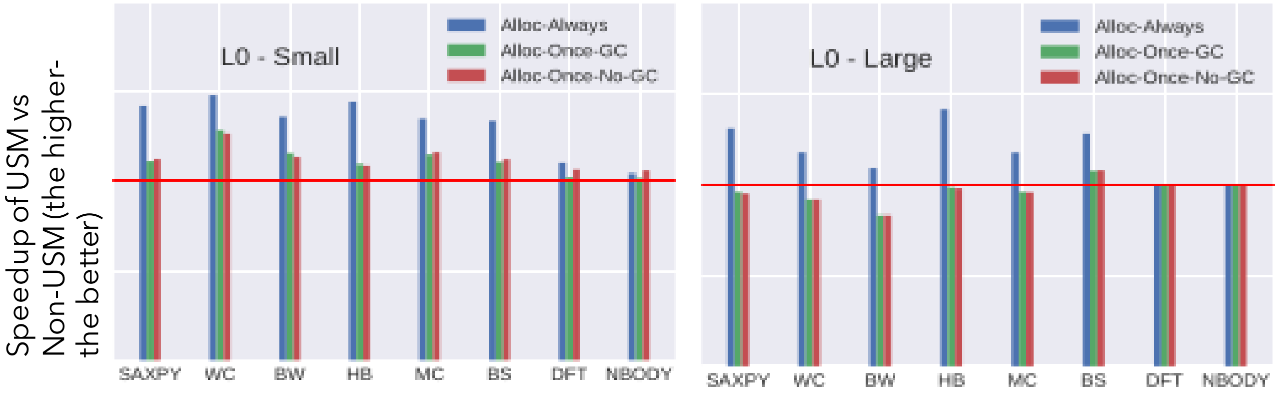
Possible Solutions:

1) **Off-heap Data Structures** (e.g., Using Direct Memory or Panama APIs)
2) **Unified Shared Memory Java Heap**
   - Sounds crazy, but does this work? And it is worth it?

On-Heap Data Structures with USM Java Heap

1. Shared Memory Pointers are shared -> no need for data marshalling and unmarshalling
   1. No Seg-Faults
   2. BUT: Need to add a sync point (e.g., flush Level Zero command queue) before the GC





For Integrated GPUs (and Leve Zero) it is beneficial and it does not degrade performance. However, it needs coordination with the Java GC

- *Memory provisioning in JDK seems to be filesystem based with* ***mmap****.*
- *Possible extensions with Level Zero to share files (**/dev/shm**) from the OS?*

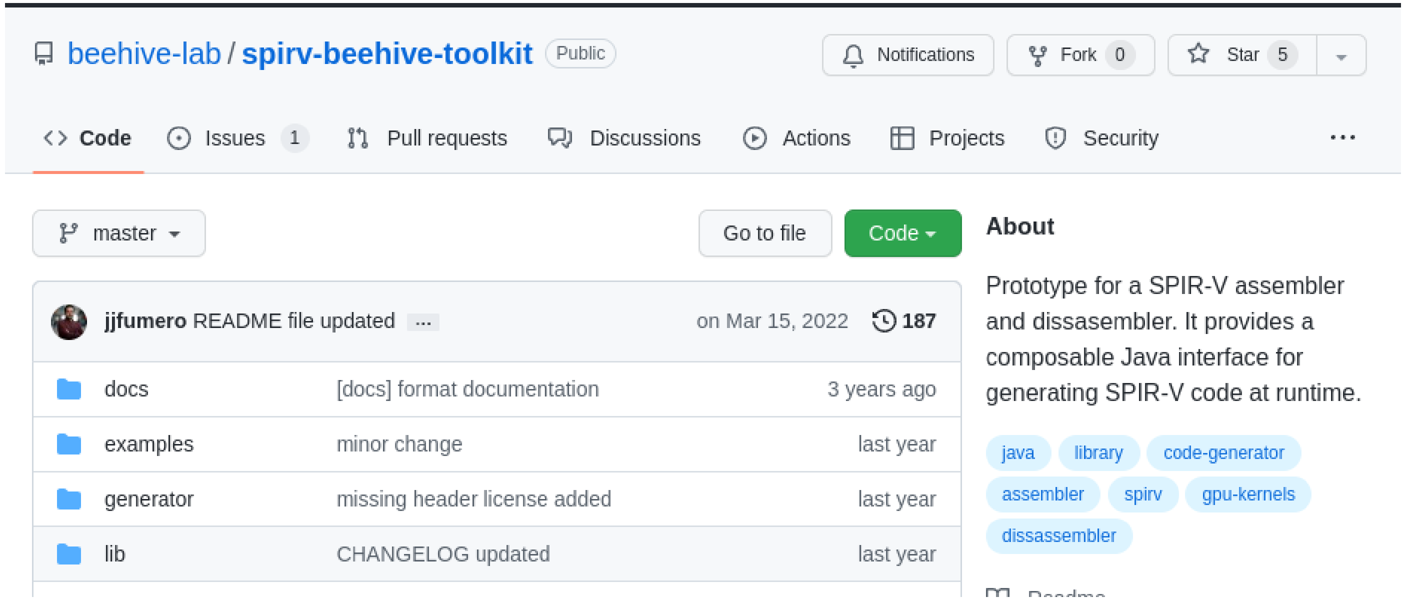
Standalone CodeGen Library for SPIR-V

Beehive SPIR-V Toolkit for code-gen within TornadoVM

MANCHESTER 1824
The University of Manchester

- In-House Java Library for SPIR-V code generation
- Works totally independent from TornadoVM
- It implements **full SPIR-V 1.2**
    - We can sync with SPIR-V 1.5 or any other version quickly
- Plans for open-source it as a stand-alone library

GraalVM → SPIR



https://github.com/beehive-lab/beehive-spirv-toolkit

Beehive SPIR-V Toolkit for code-gen within TornadoVM

- In-House Java Library for SPIR-V code generation
- Works totally independent from TornadoVM
- It implements **full SPIR-V 1.2**
  - We can sync with SPIR-V 1.5 or any other version quickly
- Plans for open-source it as a stand-alone library

```
// SPIR-V Header
asm.module = new SPIRVModule(
        new SPIRVHeader(
                1,  // Major Version
                2,  // Minor Version
                32, // ID-Generator (new one)
                0,  // Bounds
                0)); // Schema
```

```
; SPIR-V
; Version: 1.2
; Generator: Khronos; 32
; Bound: 77
; Schema: 0
```

Standalone library for low-level GPU programming

LevelZero JNI Library for TornadoVM

- Level Zero Bridge for TornadoVM
  - Since LevelZero is not stable yet, we tried to do a 1-1 mapping between the Java API and C-LevelZero.
  - Easy for us to adapt to new changes
  - In near future, we will leverage this API

```
// Create the Level Zero Driver
LevelZeroDriver driver = new LevelZeroDriver();
int result =
driver.zeInit(ZeInitFlag.ZE_INIT_FLAG_GPU_ONLY);
LevelZeroUtils.errorLog("zeInit", result);

// Get the number of drivers
int[] numDrivers = new int[1];
result = driver.zeDriverGet(numDrivers, null);
LevelZeroUtils.errorLog("zeDriverGet", result);
```

https://github.com/beehive-lab/levelzero-jni/

4. Ideas/Feedback & Discussions

Would TornadoVM benefit from the Unified Runtime?

MANCHESTER 1824
The University of Manchester

1. Not immediately: TornadoVM has already ports for CUDA PTX, OpenCL C, and SPIR-V
    - And through its runtime:
        - Automatic Live Task Migration (even with different backends)
        - Automatic Multi-device support
        - Automatic Batch processing

2. However (my opinion):

    1. If the Unified Runtime becomes a standard --> We believe this is appealing for Java/JVM architects

    2. Access to other Prog. Models (e.g., Metal, AMD HIP, etc), even OpenMP or Intel TBB

    3. If it enables dynamic task migration across backends (TornadoVM already does this)

    4. The same SPIR-V code can be used to dispatch in all compatible backend/implementations

Brainstorming the Future of oneAPI/LevelZero for Managed Runtime PL

1. Memory Page Faults/Memory Page migration counters
   - Similar to the NVIDIA NSys Profiler
   - Related issue: https://github.com/oneapi-src/level-zero/issues/100

2. Interaction with the Garbage Collectors (e.g., Java GC) (as we talked: e.g., shared memory FS and mmap
3. Async Device Exception Handling support
   - E.g., How to handle arithmetic exception in hardware?
4. Features: Device aggregation
   - E.g., Does it make sense to have 2 GPUs acting as 1 big GPU? -> Dynamic kernel dispatch across GPUs using the same system (e.g., Level Zero, SYCL oneAPI, etc)
   - Best device/s mapping (smart device selection mode)

5. Can the Relaxed Limited mode be the default mode?
   - https://github.com/oneapi-src/level-zero/issues/89

6. Improvements in the Kernel Suggest for Group sizes. We see differences in performance between the suggest threads on iGPU vs dGPUs and manual tuning.

7. Use Device Buffers Cached Version by default: https://github.com/intel/compute-runtime/issues/515

Java Garbage Collector Related Issues in the context of GPUs:

[ISSUE] https://github.com/gpu/JOCL/issues/7

- https://github.com/gpu/JOCL/commit/d01208c9687dae6015047d4cd55c16f65dbcc6da
- https://github.com/gpu/JOCL/commit/5c6e44f8dd6a84d539ec8cf2b489f707e12f3d07

Thank you!

- Partially supported by the EU Horizon 2020:
    - ELEGANT 957286
    - AERO 101092850
    - INCODE 101093069
    - TANGO 101070052
    - ENCRYPT 101070670
    - E2Data 780245
    - ACTICLOUD 732366
- Partially supported by Intel Grant

Juan Fumero: juan.fumero@manchester.ac.uk

@snatverk