TornadoVM: Transparent Hardware Acceleration for Java…and Beyond!

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

MANCHESTER
1824
The University of Manchester

Devoxx Ukraine 2021
20th Nov 2021

TORNADOVM

Outline

1. Background
   1. Why TornadoVM, why now?
2. Overview TornadoVM
   1. TornadoVM as multi-backend
   2. Discussion of each backend
3. SPIR-V Backend
4. Performance Evaluation
5. Interoperability with Python, R, …
6. Use cases and how TornadoVM is being piloted in Industry
7. Conclusions

Motivation

Current Computer Systems

GPUs

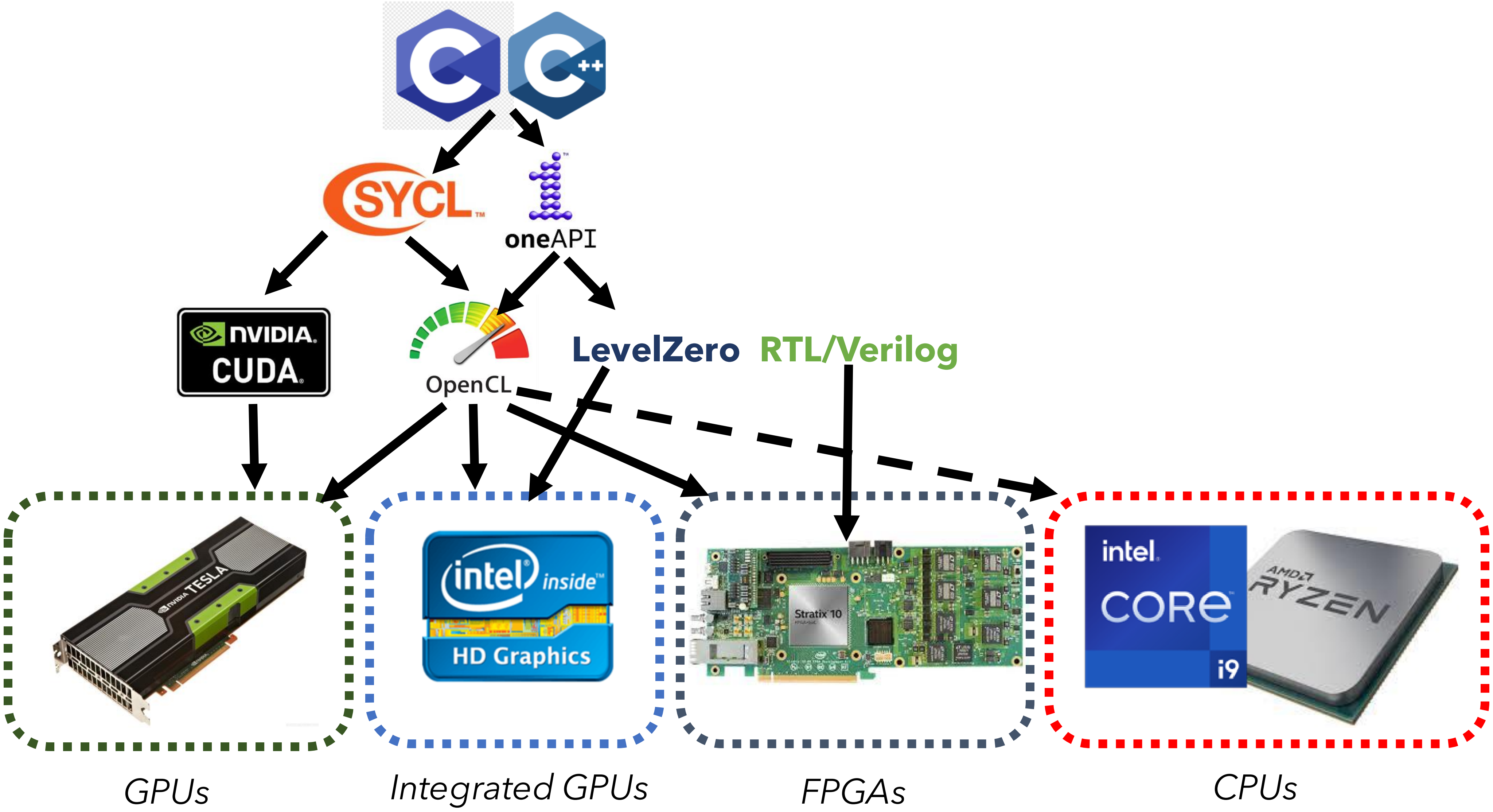
Integrated GPUs

FPGAs

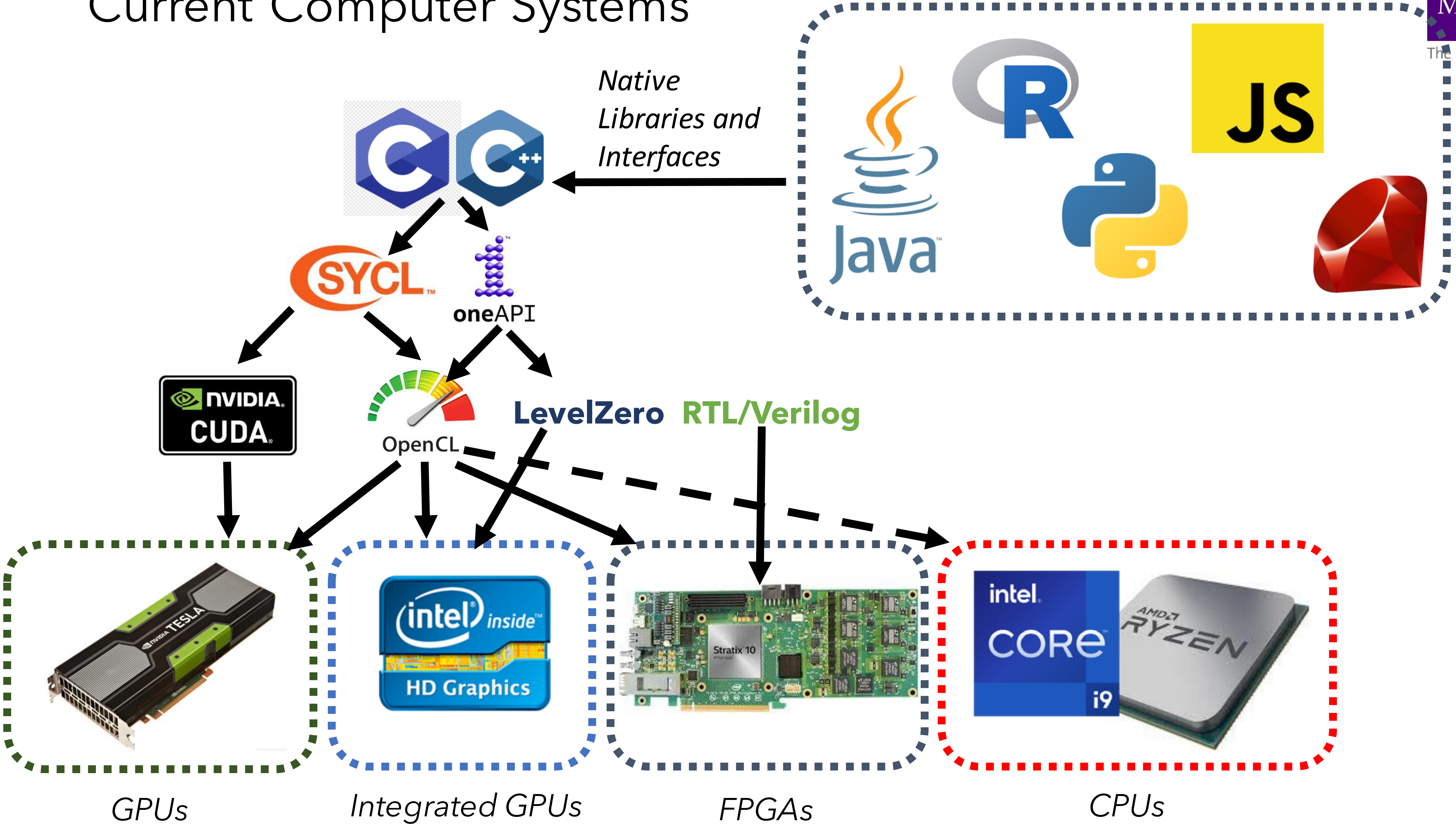
CPUs

Current Computer Systems

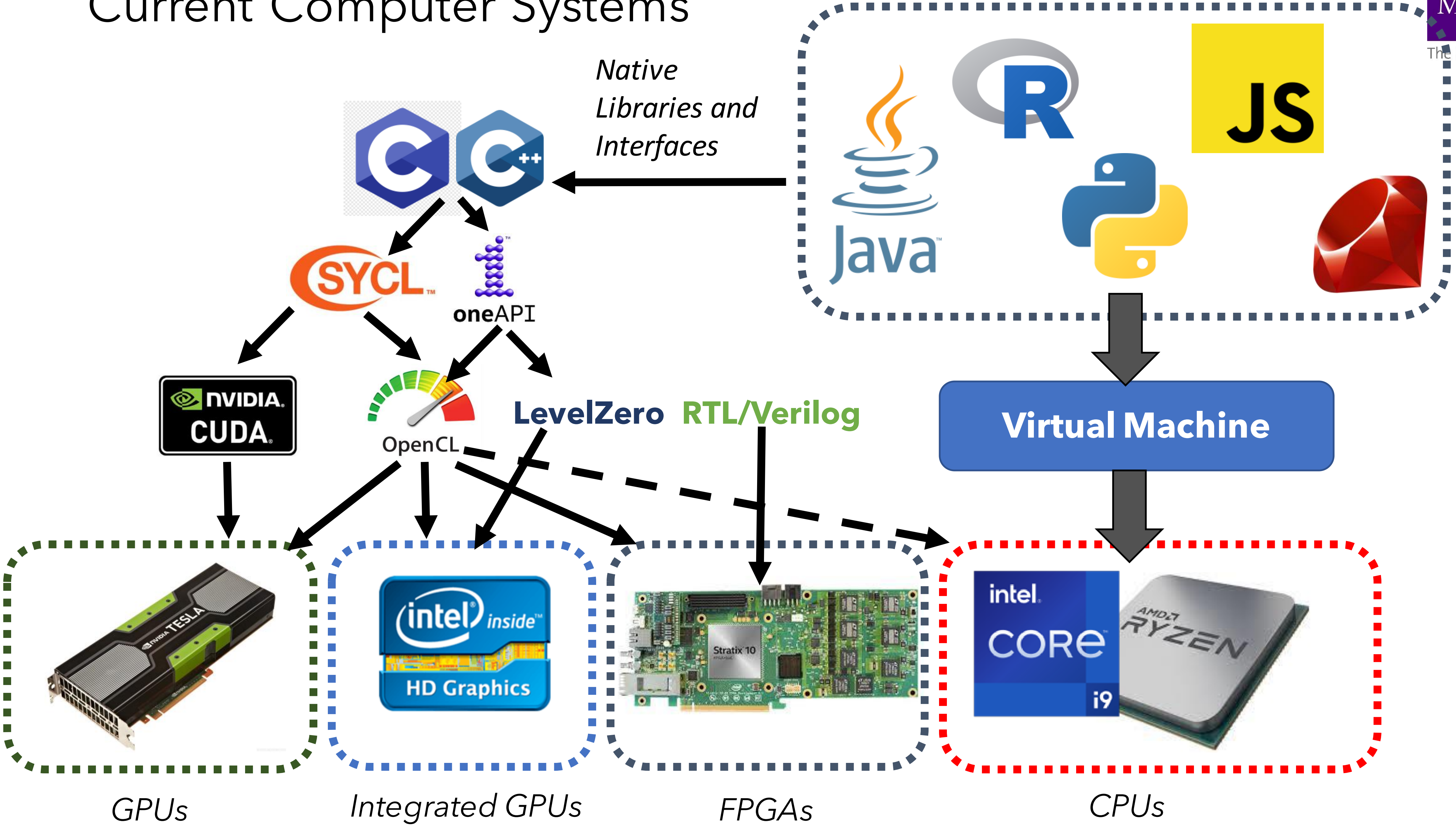
MANCHESTER 1824
The University of Manchester



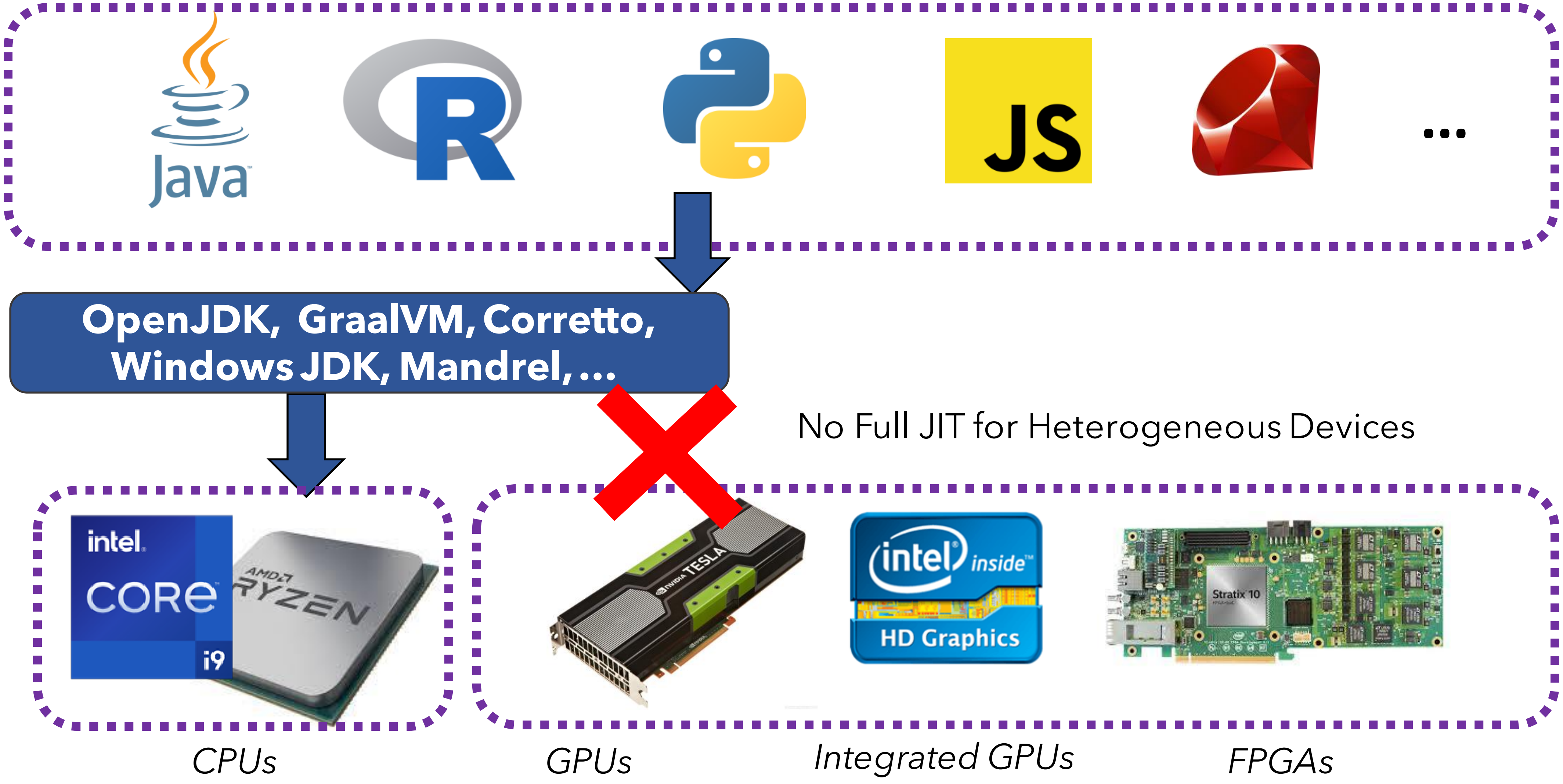
Current Computer Systems



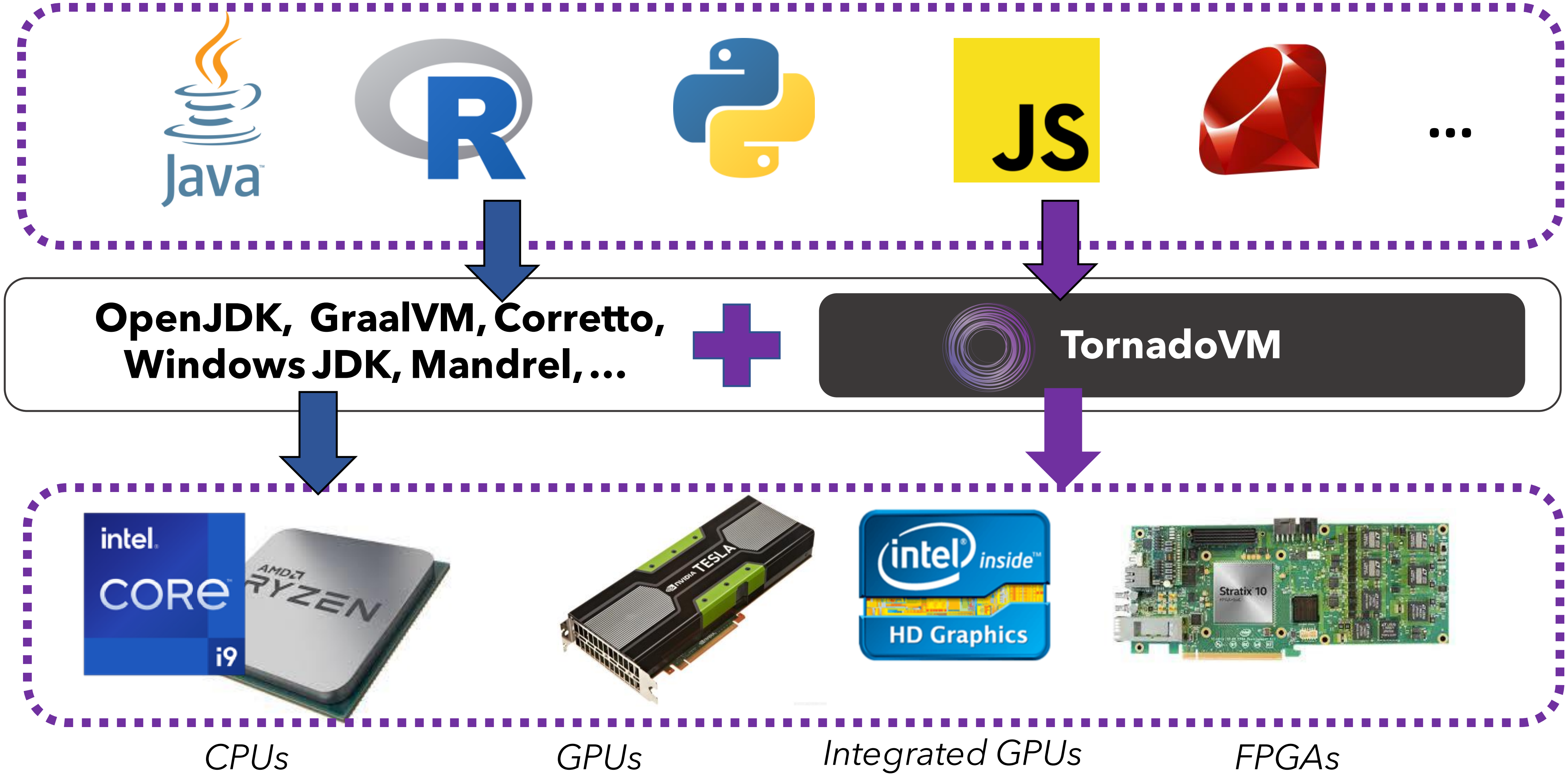
Current Computer Systems



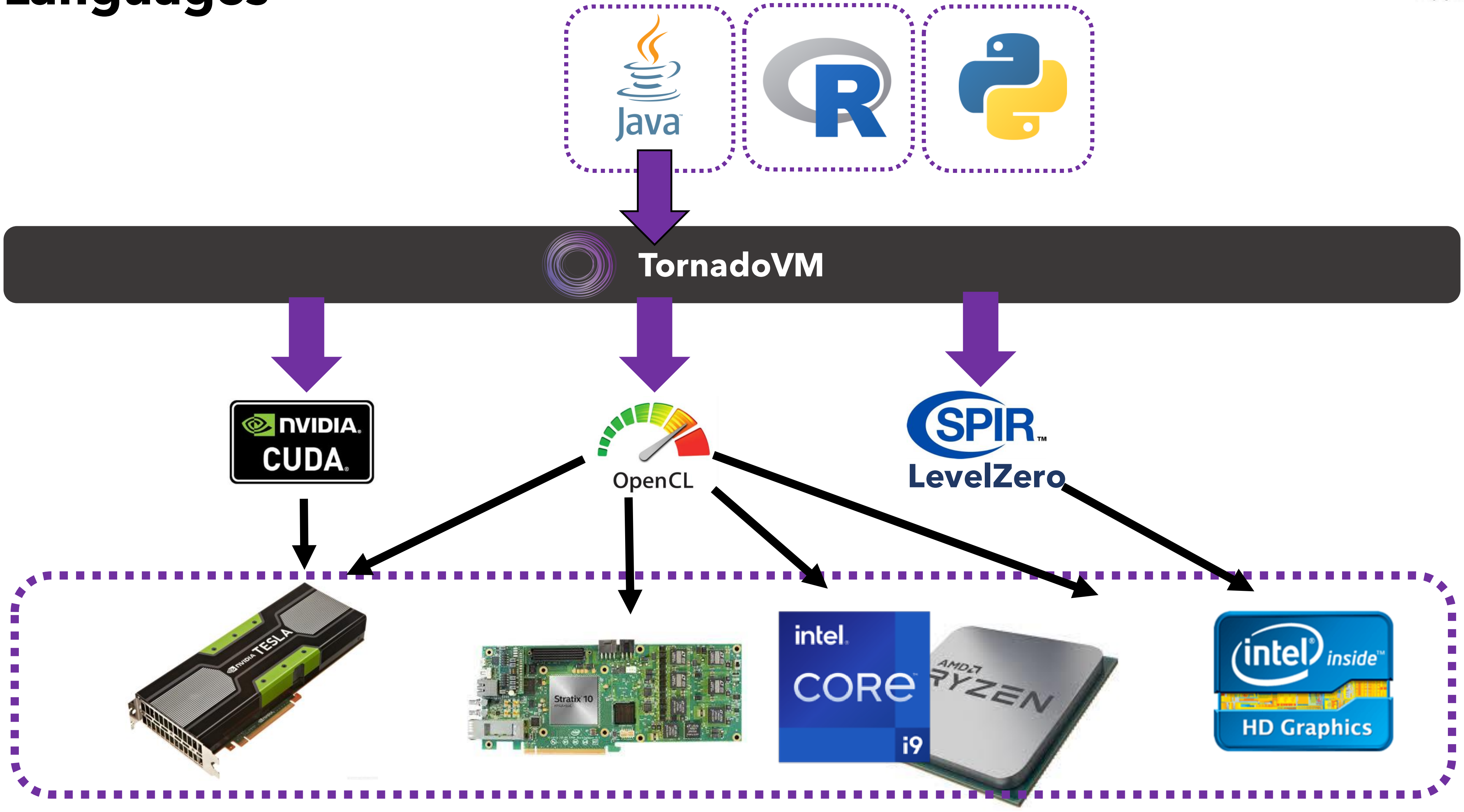
Fast Path to GPUs and FPGAs



Fast Path to GPUs and FPGAs

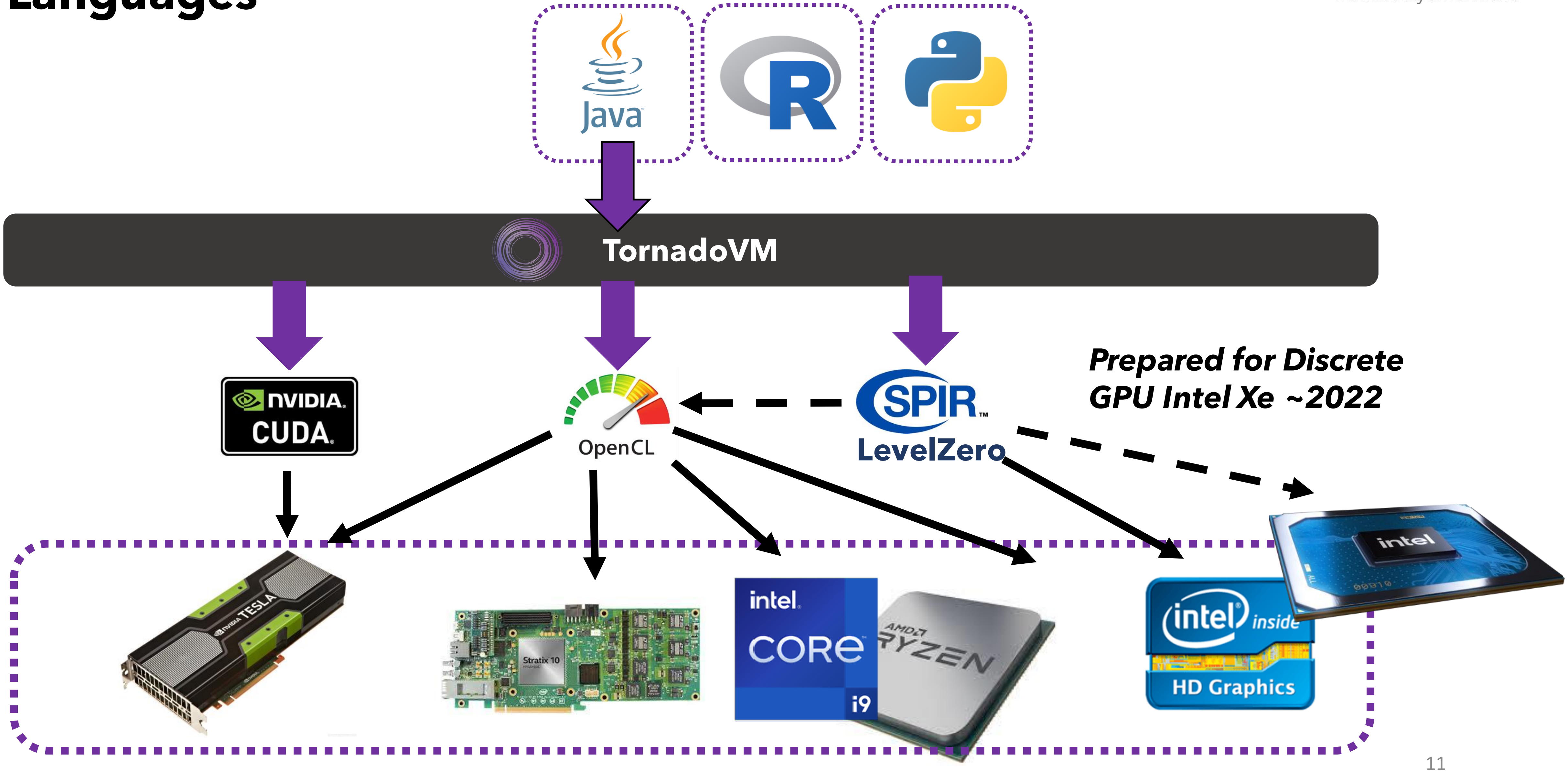


Enabling Acceleration for Managed Runtime Languages



Enabling Acceleration for Managed Runtime Languages

The University of Manchester



TORNADOVM

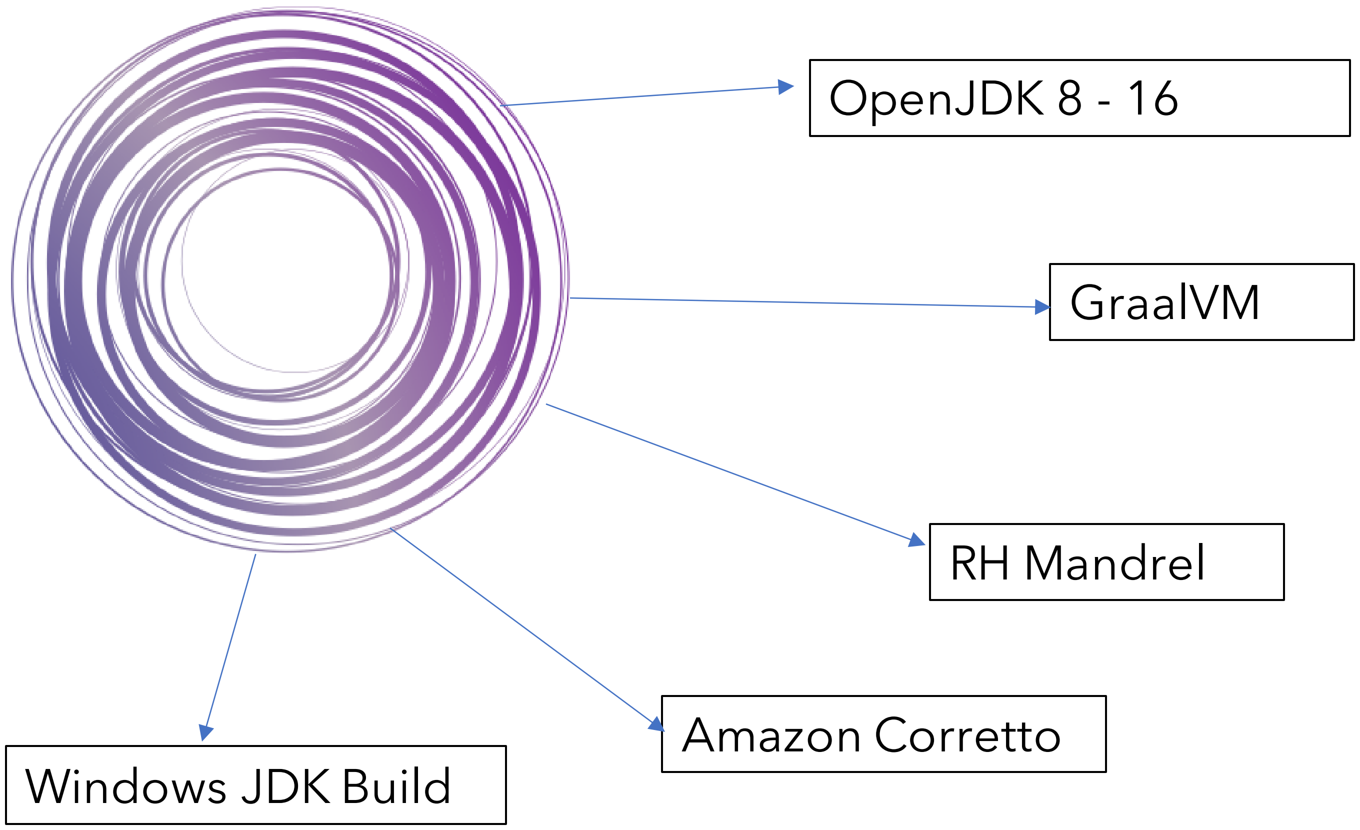
www.tornadovm.org

TornadoVM Overview

MANCHESTER 1824
The University of Manchester

www.tornadovm.org

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

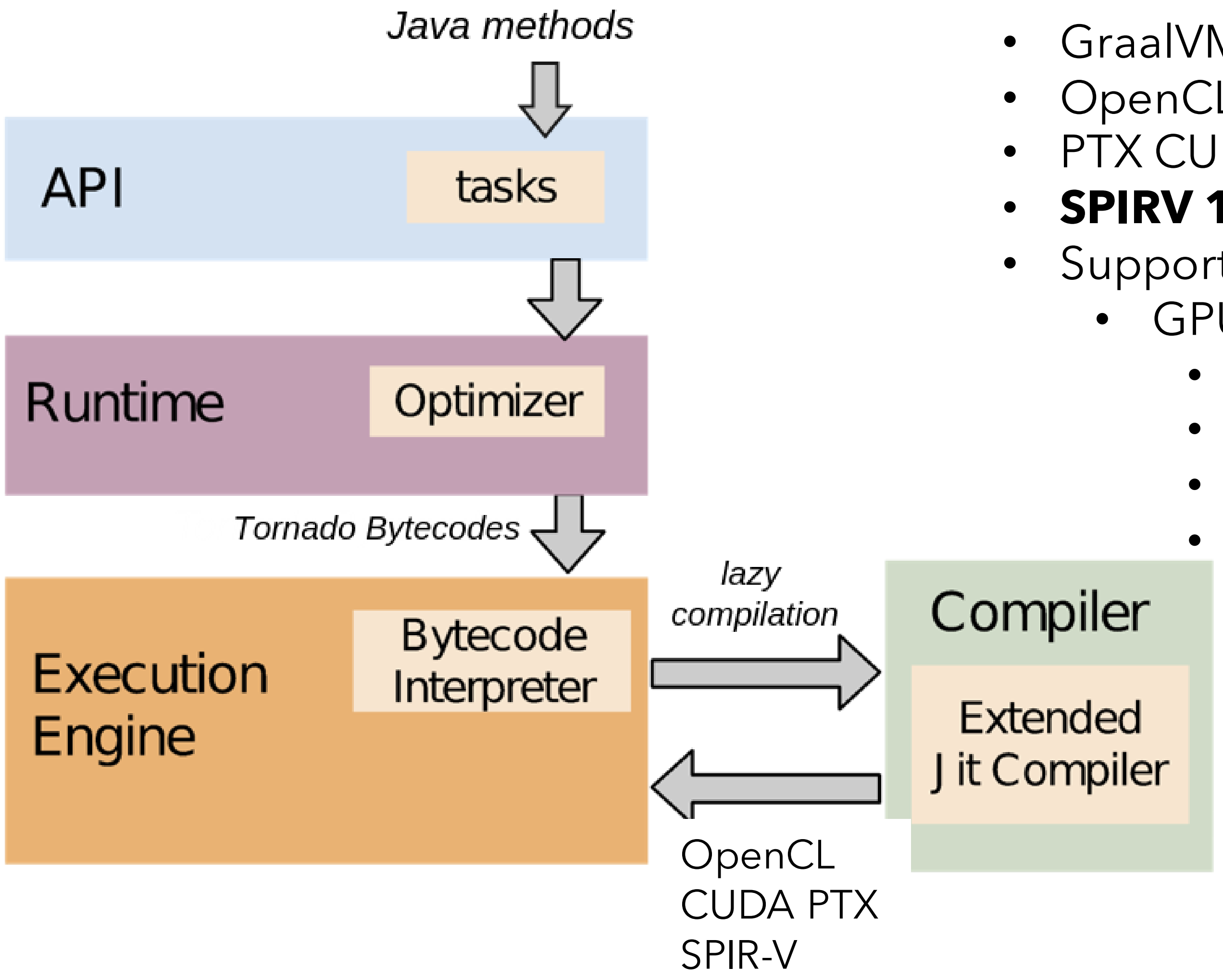


> Open-source Plug-in to multiple JVMs that allows developers to run JVM based programs on heterogeneous hardware

- Perform Automatic Task Migration
- Optimising JIT Compiler for GPUs/FPGAs
- Vendor agnostic, GPUs, CPUs, FPGAs within the same source

License: GPLv2 + CE

TornadoVM Overview



- GraalVM 21.2.0
- OpenCL >= 1.2
- PTX CUDA >= 10.0
- **SPIRV 1.2 (Prototype)**
- Support for:
  - GPUs:
    - NVIDIA
    - AMD
    - Intel
    - ARM Mali
  - FPGAs:
    - Xilinx
    - Intel
  - CPUs:
    - Intel/AMD

Different Backends

Open Computing **L**anguage

Open Standard – Khronos Group

Writing programs portable* across platforms
(source code portability)

Run on CPUs, GPUs, DSPs, FPGAs

Different Backends

Open Computing **L**anguage

Open Standard – Khronos Group

Writing programs portable* across platforms (source code portability)

Run on CPUs, GPUs, DSPs, FPGAs

PTX: **Parallel Thread eXecution**

ISA used in NVIDIA CUDA's programming model

Developed by NVIDIA

Only for NVIDIA GPUs

Different Backends

OpenCL

Open Computing **L**anguage

Open Standard – Khronos Group (non-profit tech consourtium)

Writing programs portable* across platforms (source code portability)

Run on CPUs, GPUs, DSPs, FPGAs

NVIDIA CUDA

PTX: **P**arallel **T**hread e**X**ecution

ISA used in NVIDIA CUDA's programming model

Developed by NVIDIA

Only for NVIDIA GPUs

SPIR™

Standard **P**ortable **I**ntermediate **R**epresentation

Standard IR binary originally created for OpenCL (>= 2.1)

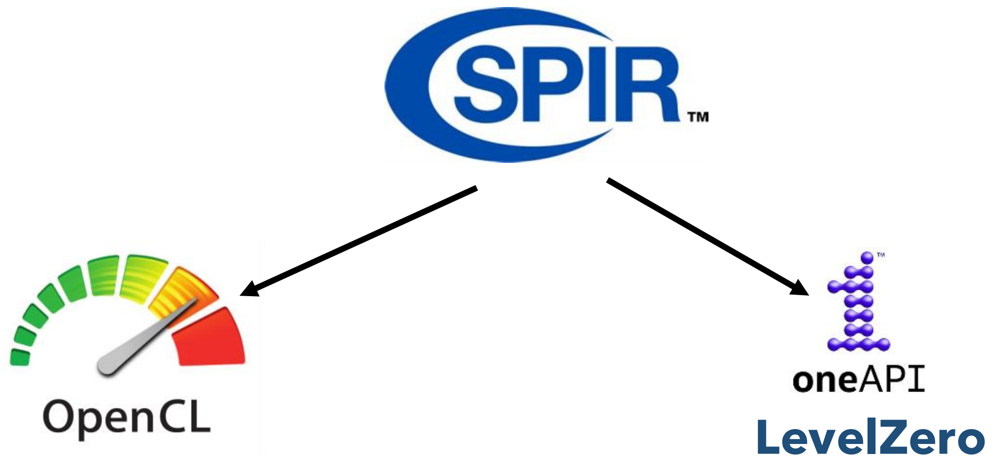
Enables distribution of compute binaries for OpenCL

Any OpenCL >= 2.1 device

Shared IR with Vulkan for Graphics

And Intel Level Zero?

- It a brand new baremetal API for low-level programming of heterogeneous architectures.
- It is part of the Intel oneAPI ecosystem and can be used as a standalone library.
- Level Zero consumes SPIRV binaries for compute



But … why Level Zero?

MANCHESTER 1824
The University of Manchester

oneAPI
LevelZero

- Clearly influenced by OpenCL
- It can evolve independently
- It supports:
  - Low latency command queues
  - **Virtual functions**
  - **Memory visibility control, caching control**
  - Unified memory
  - Device partitioning
  - Instrumentation and debugging
  - **Control of power management**
  - **Control of frequency**
  - **Hardware diagnostics**
  - …
- **This level of control is very appealing for system programming, runtime systems and compilers**

It is part of the oneAPI stack and can be accessed as a standalone library: https://github.com/oneapi-src/level-zero

More Info:

- Level Zero Spec: https://spec.oneapi.io/level-zero/latest/index.html
- https://jjfumero.github.io/posts/2021/09/introduction-to-level-zero/

Comparisons

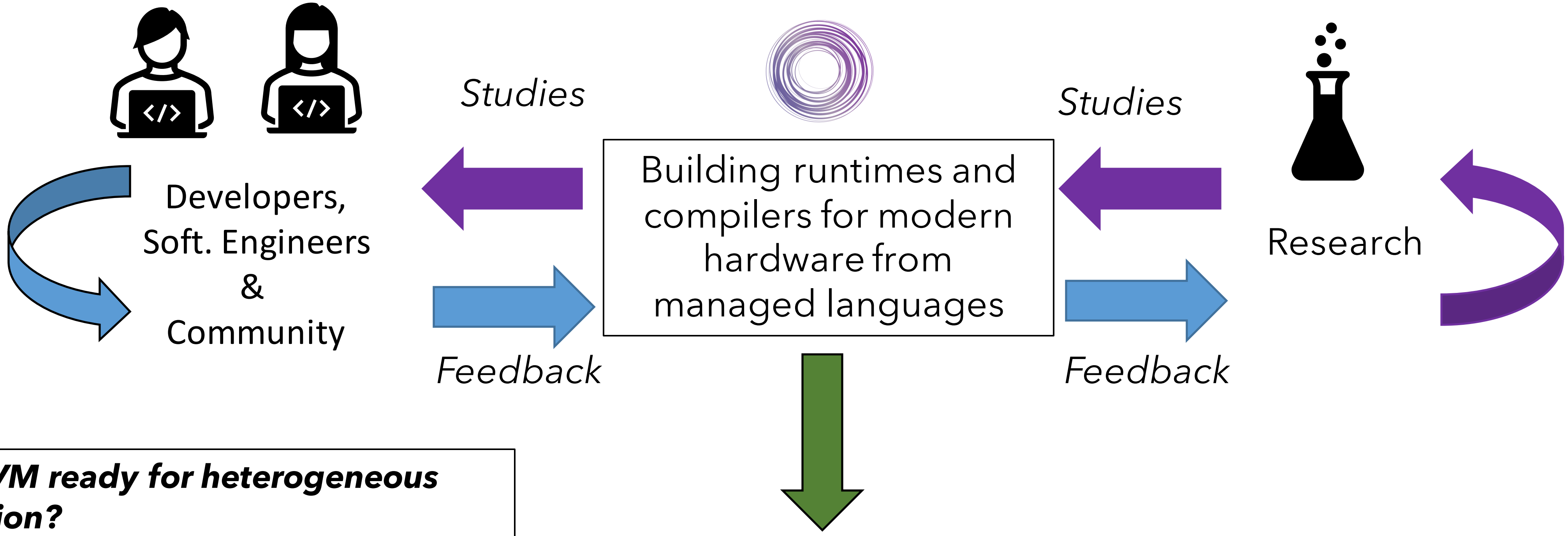
	Advantages	Disadvantages
OpenCL	- Easier to write than other alternatives - Source code portable - Wide variety of devices	- Performance is not portable (hard to know what the compiler driver will do)
NVIDIA CUDA	- Highly Tuned for NVIDIA GPUs - High Performance - Low-level features	- Only works for NVIDIA GPUs. - No control over the final compilation (PTX -> bin)
oneAPI LevelZero	- Very low-level control of the hardware resources - It dispatches SPIR-V kernels - Higher control of execution - Prepared for a wide set of devices	- Exposed to users but designed for coupling with runtimes/compilers (by design) - New technology

… and now experimenting with

SPIR™

SPIR-V Kernels can be consumed by OpenCL runtime and Intel Level Zero API

So why all of these backends?



Q) Is JVM ready for heterogeneous execution?
Q) Which technology is the best bet for managed runtime languages?
Q) What can they offer to managed runtime languages?

Next Gen of Runtime Systems for More Efficient Computing

But, how TornadoVM compiles parallel code from Java?

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

TornadoVM JIT Compiler

Multi-backend JIT Compiler Workflow

The University of Manchester

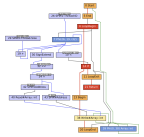
Programmer's view

```
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];
    }
}
```

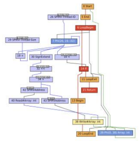
javac

Java Bytecodes

TornadoVM JIT Compiler



Graal IR

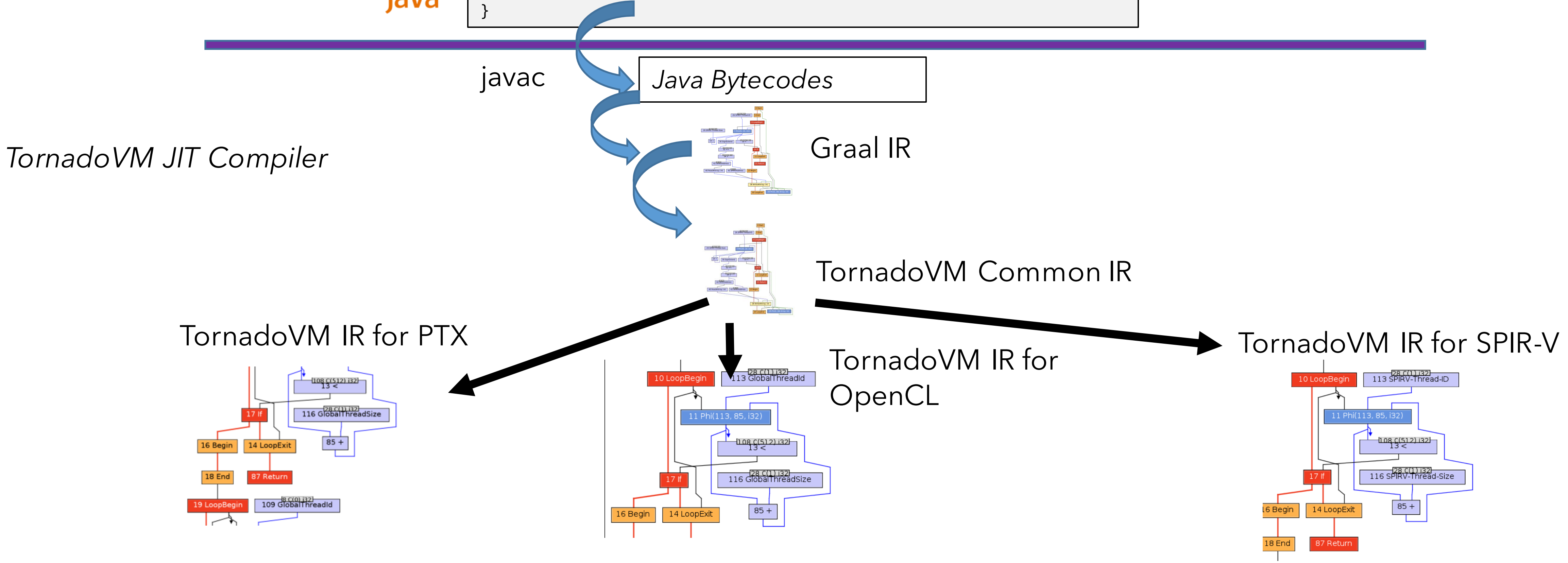


TornadoVM Common IR

Multi-backend JIT Compiler Workflow

Programmer's view

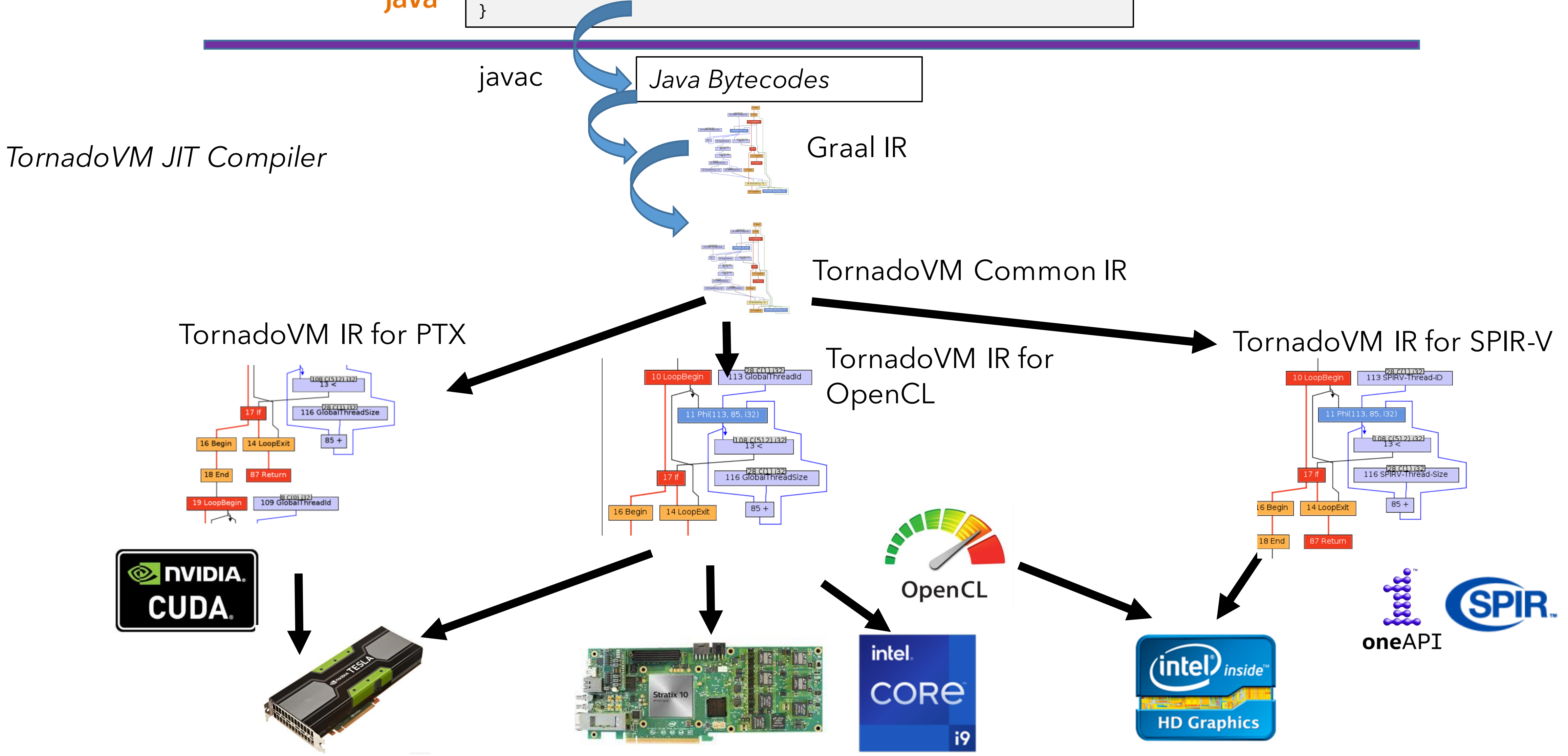
```
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```



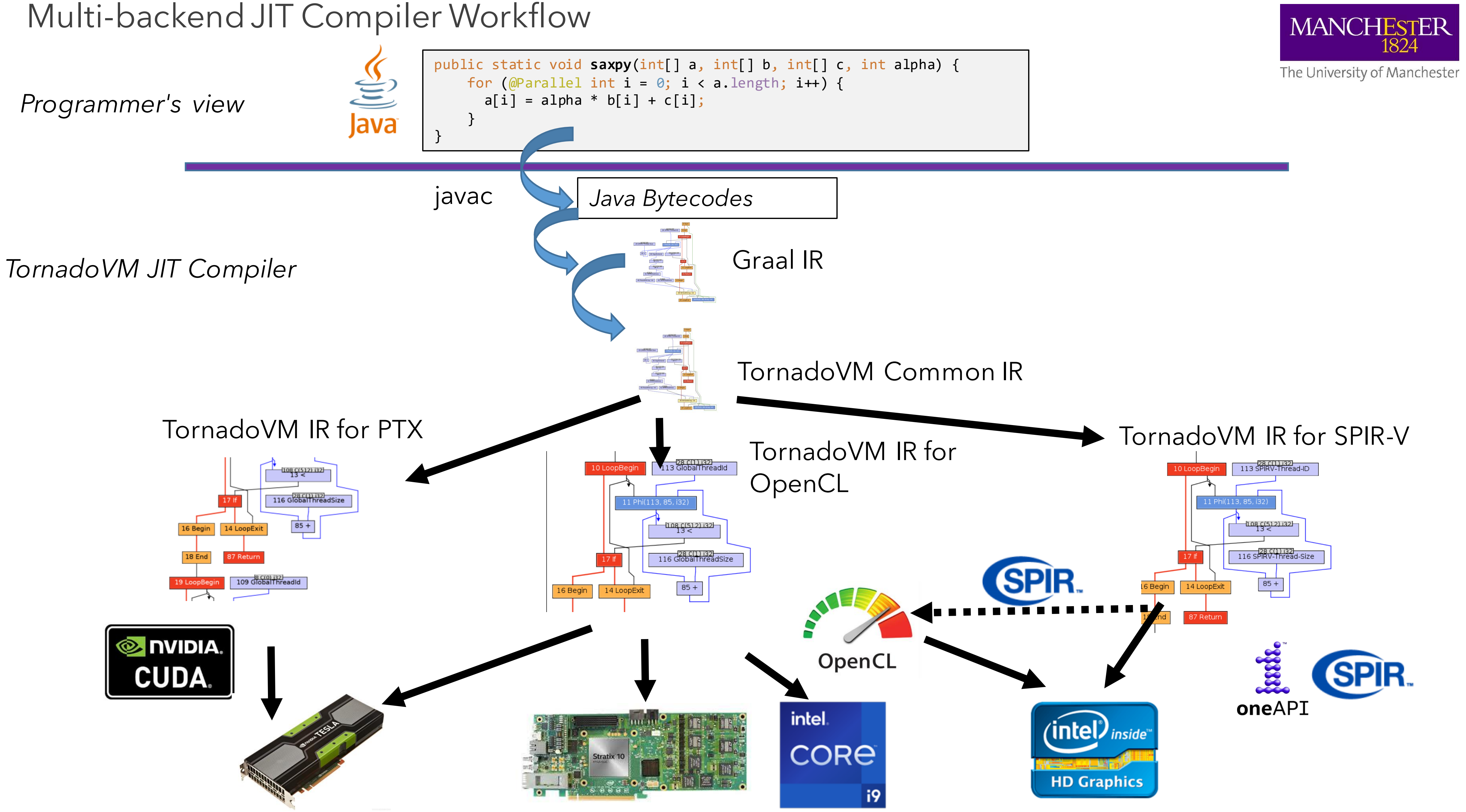
Multi-backend JIT Compiler Workflow

Programmer's view

```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

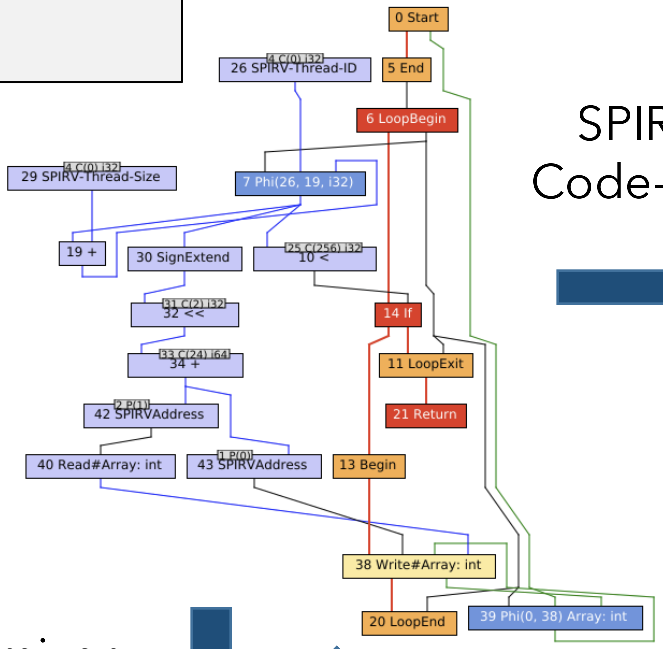


SPIR-V JIT compiler (and runtime) for TornadoVM

MANCHESTER
1824
The University of Manchester

```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];
    }
}
```

Build
Graal/Tornado IR



JIT Optimizer
For SPIR-V

SPIR-V
Code-Gen

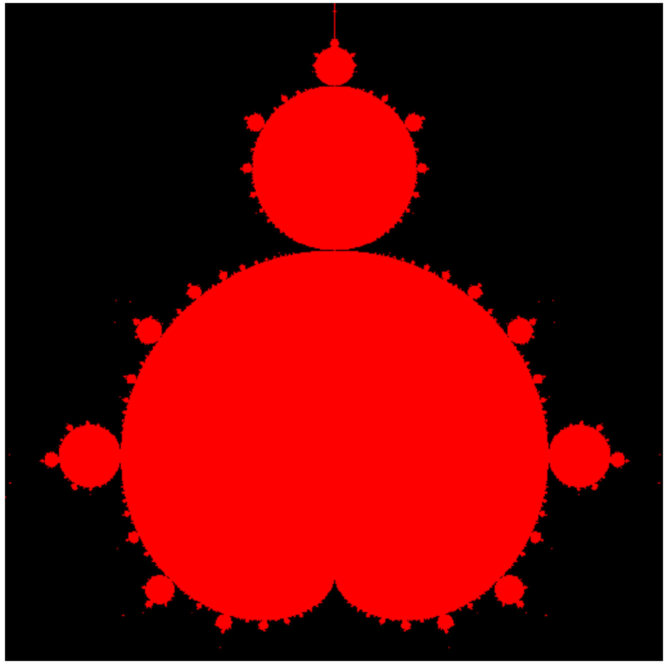
```
. . .
%B2 = OpLabel
%77 = OpLoad %uint %spirv_i_4 Aligned 4
%78 = OpSConvert %ulong %77
OpStore %spirv_l_6 %78 Aligned 8
%79 = OpLoad %ulong %spirv_l_6 Aligned 8
%80 = OpShiftLeftLogical %ulong %79 %uint_2
OpStore %spirv_l_7 %80 Aligned 8
%81 = OpLoad %ulong %spirv_l_7 Aligned 8
%82 = OpIAdd %ulong %81 %ulong_24
OpStore %spirv_l_8 %82 Aligned 8
%83 = OpLoad %ulong %spirv_l_1 Aligned 8
%84 = OpLoad %ulong %spirv_l_8 Aligned 8
%85 = OpIAdd %ulong %83 %84
OpStore %spirv_l_9 %85 Aligned 8
%86 = OpLoad %ulong %spirv_l_9 Aligned 8
%87 = OpConvertUToPtr %_ptr_CrossWorkgroup_uint %86
%88 = OpLoad %uint %87 Aligned 4
OpStore %spirv_i_10 %88 Aligned 4
%89 = OpLoad %ulong %spirv_l_2 Aligned 8
%90 = OpLoad %ulong %spirv_l_8 Aligned 8
. . .
```

oneAPI

LevelZero-JNI dispatch

Example – Mandelbrot Computation

```
public class Mandelbrot {

 static void mandelbrotFractal(final int size, short[] output) {
   for (@Parallel int i = 0; i < size; i++) {
      for (@Parallel int j = 0; j < size; j++) {
         // Mandelbrot computation
         // Compute the value of each pixel (x, y)
         // Check example on Github for the specifics
      }
   }
 }

 void createTaskAndRun(int size) {
      mandelbrotImage = new short[size * size];

      TaskSchedule ts = new TaskSchedule("s0")
         .task("t0", Mandelbrot::mandelbrotFractal, size, mandelbrotImage)
         .streamOut(mandelbrotImage);

      ts.execute();
 }

}
```

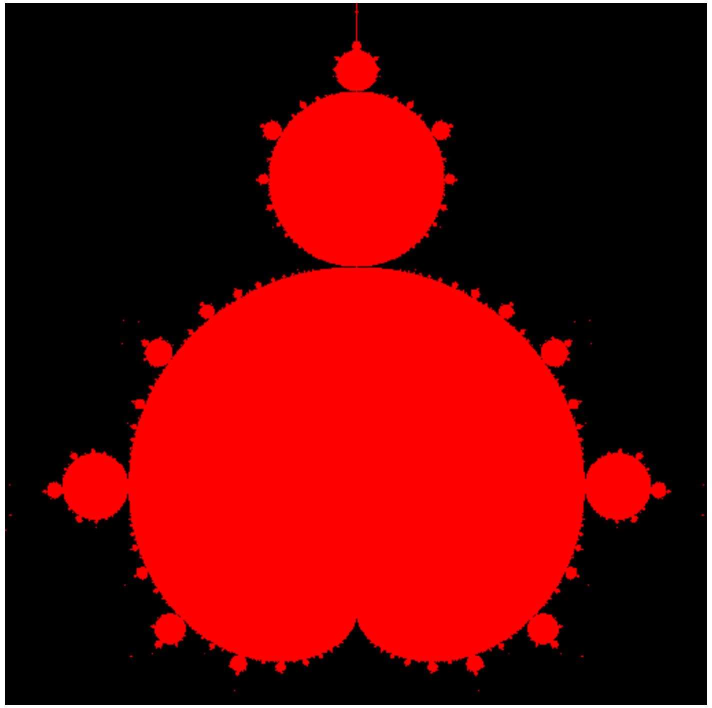


https://github.com/jjfumero/tornadovm-examples

Live Demo with the SPIR-V Backend

MANCHESTER 1824
The University of Manchester

Mandelbrot computation

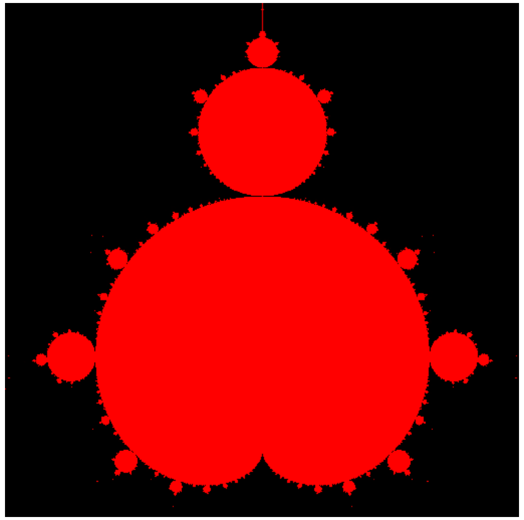


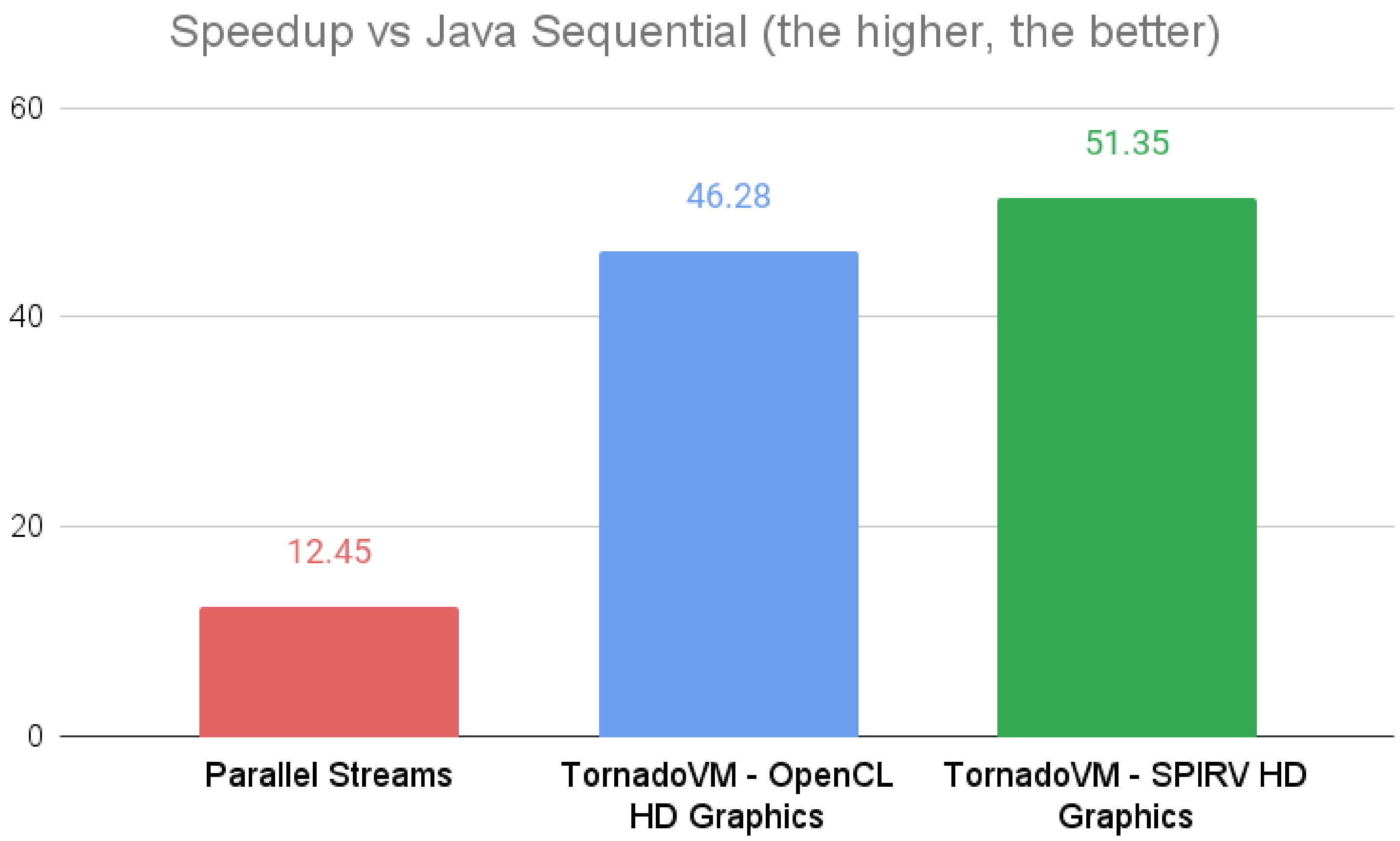
https://github.com/jjfumero/tornadovm-examples

Performance

MANCHESTER 1824
The University of Manchester

https://github.com/jjfumero/tornadovm-examples

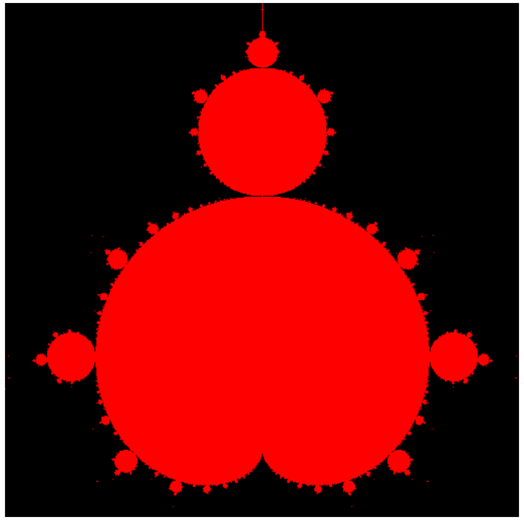


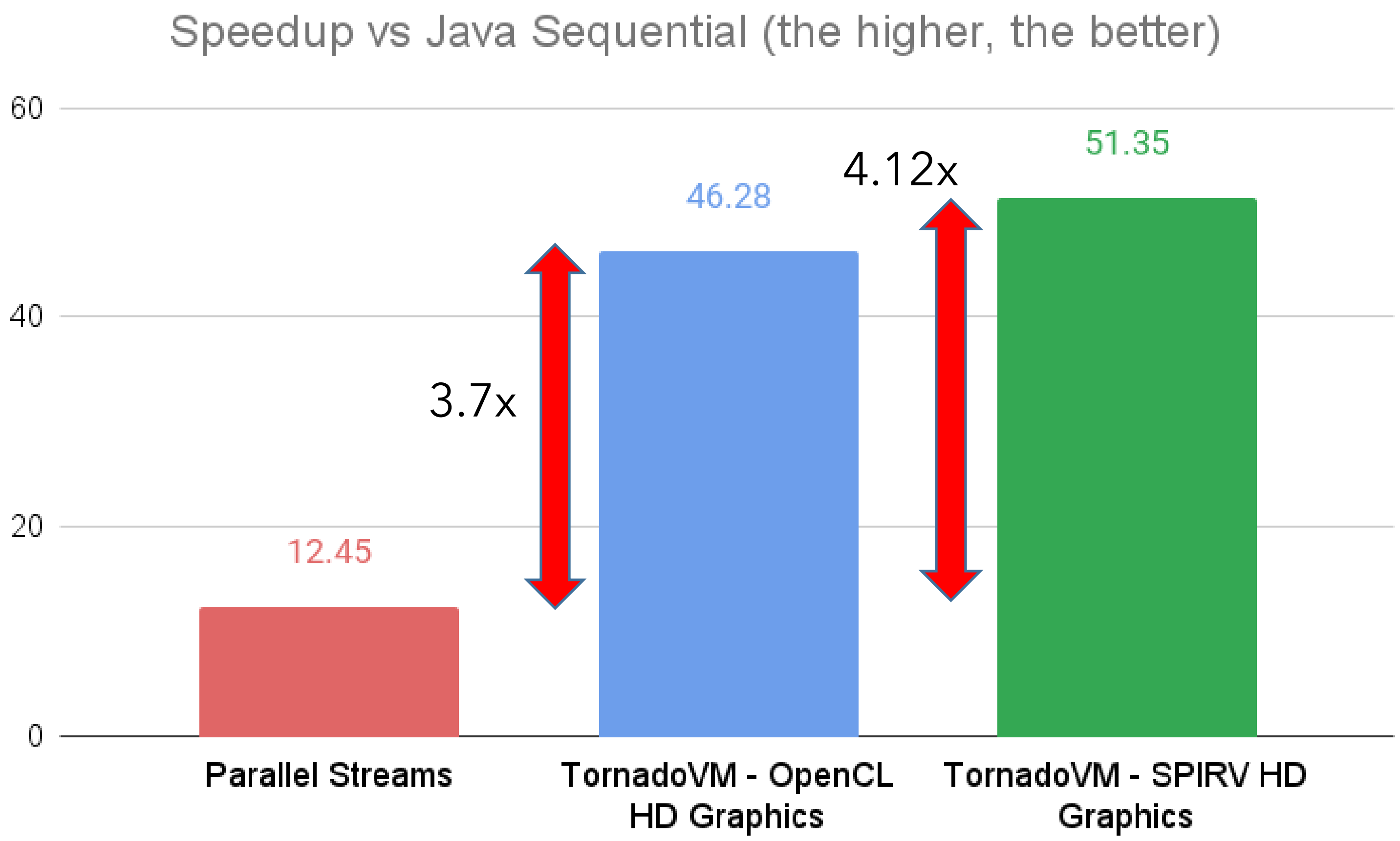


* CPU: Intel(R) Core(TM) i9-10885H
* GPU: Intel HD Graphics
* Java: 1.8.0_302
* LevelZero: 21.38.21026
* TornadoVM: 0.12

Performance

https://github.com/jjfumero/tornadovm-examples





** CPU: Intel(R) Core(TM) i9-10885H*
** GPU: Intel HD Graphics*
** Java: 1.8.0_302*
** LevelZero: 21.38.21026*
** TornadoVM: 0.12*

Profiling

Understanding Performance with the Profiler

```
$ tornado --enableProfiler console Program
```

Understanding Performance with the Profiler

$ tornado **--enableProfiler console** Program

All times are in nanoseconds

```
{
    "s0": {
        "TOTAL_KERNEL_TIME": "58591028",
        "COPY_OUT_TIME": "55693",
        "TOTAL_GRAAL_COMPILE_TIME": "179950755",
        "TOTAL_DISPATCH_DATA_TRANSFERS_TIME": "0",
        "TOTAL_TASK_SCHEDULE_TIME": "388705840",
        "COPY_IN_TIME": "50547",
        "TOTAL_BYTE_CODE_GENERATION": "6230794",
        "TOTAL_DRIVER_COMPILE_TIME": "58653972",
        "TOTAL_COPY_IN_SIZE_BYTES": "1048624",
        "TOTAL_COPY_OUT_SIZE_BYTES": "524312",
        "s0.t0": {
            "METHOD": "Mandelbrot.mandelbrotFractal",
            "DEVICE_ID": "0:0",
            "DEVICE": "Intel(R) UHD Graphics [0x9bc4]",
            "TASK_KERNEL_TIME": "58591028",
            "TASK_COMPILE_GRAAL_TIME": "179950755",
            "TASK_COMPILE_DRIVER_TIME": "58653972"
        }
    }
}
```

Task Scheduler's Name

Task-Name

Java Method Compiled

```java
TaskSchedule ts = new TaskSchedule("s0")
    .task("t0", Mandelbrot::mandelbrotFractal, size, mandelbrotImage)
    .streamOut(mandelbrotImage);
```

Understanding Performance

$ tornado **--enableProfiler console** Program


```
{
    "s0": {
        "TOTAL_KERNEL_TIME": "58591028",
        "COPY_OUT_TIME": "55693",
        "TOTAL_GRAAL_COMPILE_TIME": "179950755",
        "TOTAL_DISPATCH_DATA_TRANSFERS_TIME": "0",
        "TOTAL_TASK_SCHEDULE_TIME": "388705840",
        "COPY_IN_TIME": "50547",
        "TOTAL_BYTE_CODE_GENERATION": "6230794",
        "TOTAL_DRIVER_COMPILE_TIME": "58653972",
        "TOTAL_COPY_IN_SIZE_BYTES": "1048624",
        "TOTAL_COPY_OUT_SIZE_BYTES": "524312",
        "s0.t0": {
            "METHOD": "Mandelbrot.mandelbrotFractal",
            "DEVICE_ID": "0:0",
            "DEVICE": "Intel(R) UHD Graphics [0x9bc4]",
            "TASK_KERNEL_TIME": "58591028",
            "TASK_COMPILE_GRAAL_TIME": "179950755",
            "TASK_COMPILE_DRIVER_TIME": "58653972"
        }
    }
}
```


Total Time including data transfers, execution and TornadoVM runtime to dispatch the kernels.

Understanding Performance

MANCHESTER 1824
The University of Manchester

```
$ tornado --enableProfiler console Program
```

```
{
    "s0": {
        "TOTAL_KERNEL_TIME": "58591028",
        "COPY_OUT_TIME": "55693",
        "TOTAL_GRAAL_COMPILE_TIME": "179950755",
        "TOTAL_DISPATCH_DATA_TRANSFERS_TIME": "0",
        "TOTAL_TASK_SCHEDULE_TIME": "388705840",
        "COPY_IN_TIME": "50547",
        "TOTAL_BYTE_CODE_GENERATION": "6230794",
        "TOTAL_DRIVER_COMPILE_TIME": "58653972",
        "TOTAL_COPY_IN_SIZE_BYTES": "1048624",
        "TOTAL_COPY_OUT_SIZE_BYTES": "524312",
        "s0.t0": {
            "METHOD": "Mandelbrot.mandelbrotFractal",
            "DEVICE_ID": "0:0",
            "DEVICE": "Intel(R) UHD Graphics [0x9bc4]",
            "TASK_KERNEL_TIME": "58591028",
            "TASK_COMPILE_GRAAL_TIME": "179950755",
            "TASK_COMPILE_DRIVER_TIME": "58653972"
        }
    }
}
```

Compilation with Graal + code generation

(Java byte code -> Graal IR -> Tornado IR -> optimizations + code generation)

Internal Byte-Code Generation

Driver JIT compiler (e.g., SPIR-V -> final GPU binary)

Understanding Performance

$ tornado **--enableProfiler console** Program


```
{
    "s0": {
        "TOTAL_KERNEL_TIME": "58591028",
        "COPY_OUT_TIME": "55693",
        "TOTAL_GRAAL_COMPILE_TIME": "179950755",
        "TOTAL_DISPATCH_DATA_TRANSFERS_TIME": "0",
        "TOTAL_TASK_SCHEDULE_TIME": "388705840",
        "COPY_IN_TIME": "50547",
        "TOTAL_BYTE_CODE_GENERATION": "6230794",
        "TOTAL_DRIVER_COMPILE_TIME": "58653972",
        "TOTAL_COPY_IN_SIZE_BYTES": "1048624",
        "TOTAL_COPY_OUT_SIZE_BYTES": "524312",
        "s0.t0": {
            "METHOD": "Mandelbrot.mandelbrotFractal",
            "DEVICE_ID": "0:0",
            "DEVICE": "Intel(R) UHD Graphics [0x9bc4]",
            "TASK_KERNEL_TIME": "58591028",
            "TASK_COMPILE_GRAAL_TIME": "179950755",
            "TASK_COMPILE_DRIVER_TIME": "58653972"
        }
    }
}
```


Total Kernel Time

Total Copy Out (Device -> Java Heap)

Total Copy In (Java Heap -> Device)

Kernel Time For each task

Understanding Performance

MANCHESTER 1824
The University of Manchester

```
$ tornado --enableProfiler console Program
```


```
{
    "s0": {
        "TOTAL_KERNEL_TIME": "58591028",
        "COPY_OUT_TIME": "55693",
        "TOTAL_GRAAL_COMPILE_TIME": "179950755",
        "TOTAL_DISPATCH_DATA_TRANSFERS_TIME": "0",
        "TOTAL_TASK_SCHEDULE_TIME": "388705840",
        "COPY_IN_TIME": "50547",
        "TOTAL_BYTE_CODE_GENERATION": "6230794",
        "TOTAL_DRIVER_COMPILE_TIME": "58653972",
        "TOTAL_COPY_IN_SIZE_BYTES": "1048624",
        "TOTAL_COPY_OUT_SIZE_BYTES": "524312",
        "s0.t0": {
            "METHOD": "Mandelbrot.mandelbrotFractal",
            "DEVICE_ID": "0:0",
            "DEVICE": "Intel(R) UHD Graphics [0x9bc4]",
            "TASK_KERNEL_TIME": "58591028",
            "TASK_COMPILE_GRAAL_TIME": "179950755",
            "TASK_COMPILE_DRIVER_TIME": "58653972"
        }
    }
}
```

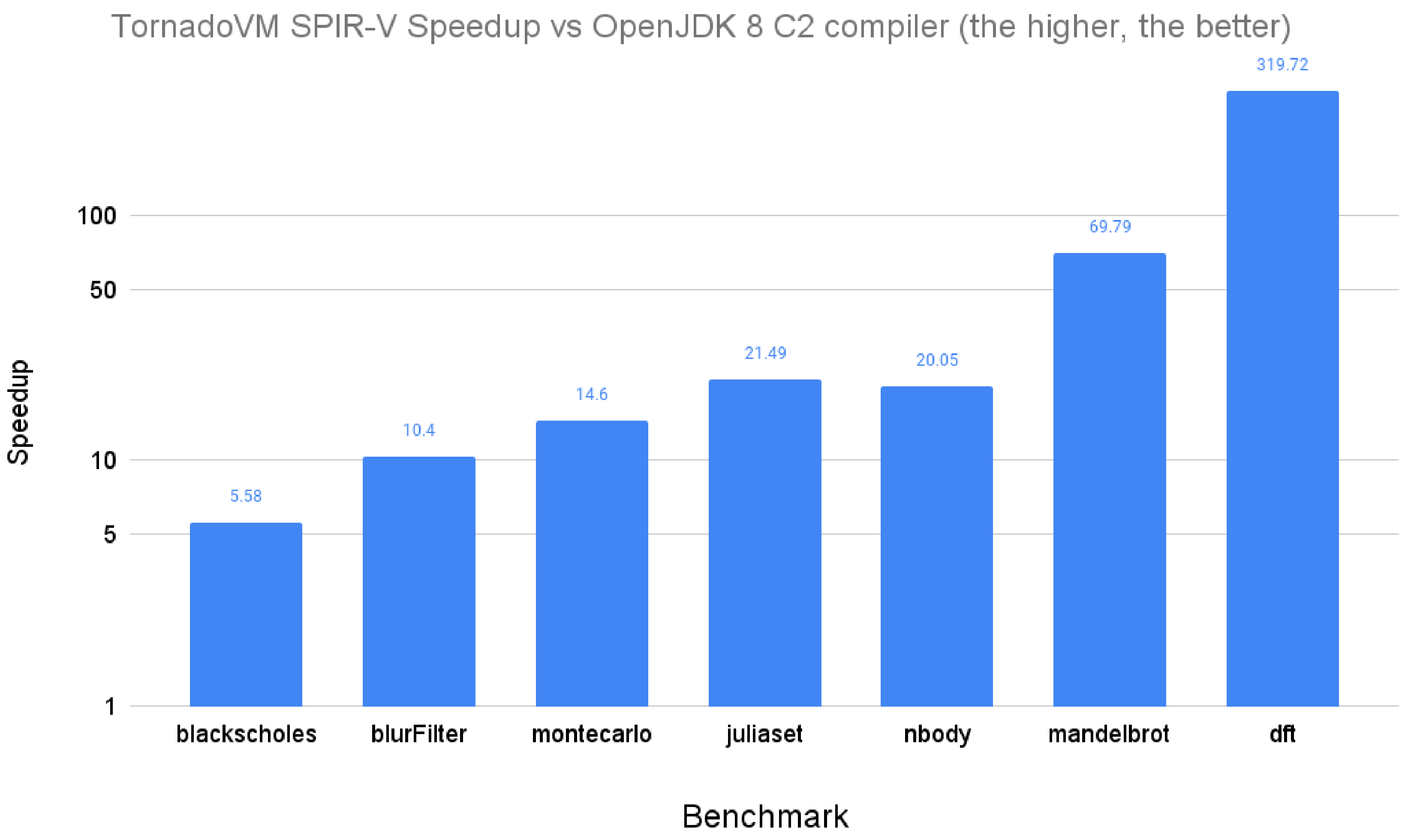

Ideally, most of the time should be spent in Kernel Execution
* Take advantage of the device's computing power
* Keep transfers to minimum

If the application has a lot of data transfers, it is worth trying with shared memory devices (e.g., Integrated GPU) --> In TornadoVM this is not currently handled (WIP)

https://github.com/jjfumero/tornadovm-examples

Performance

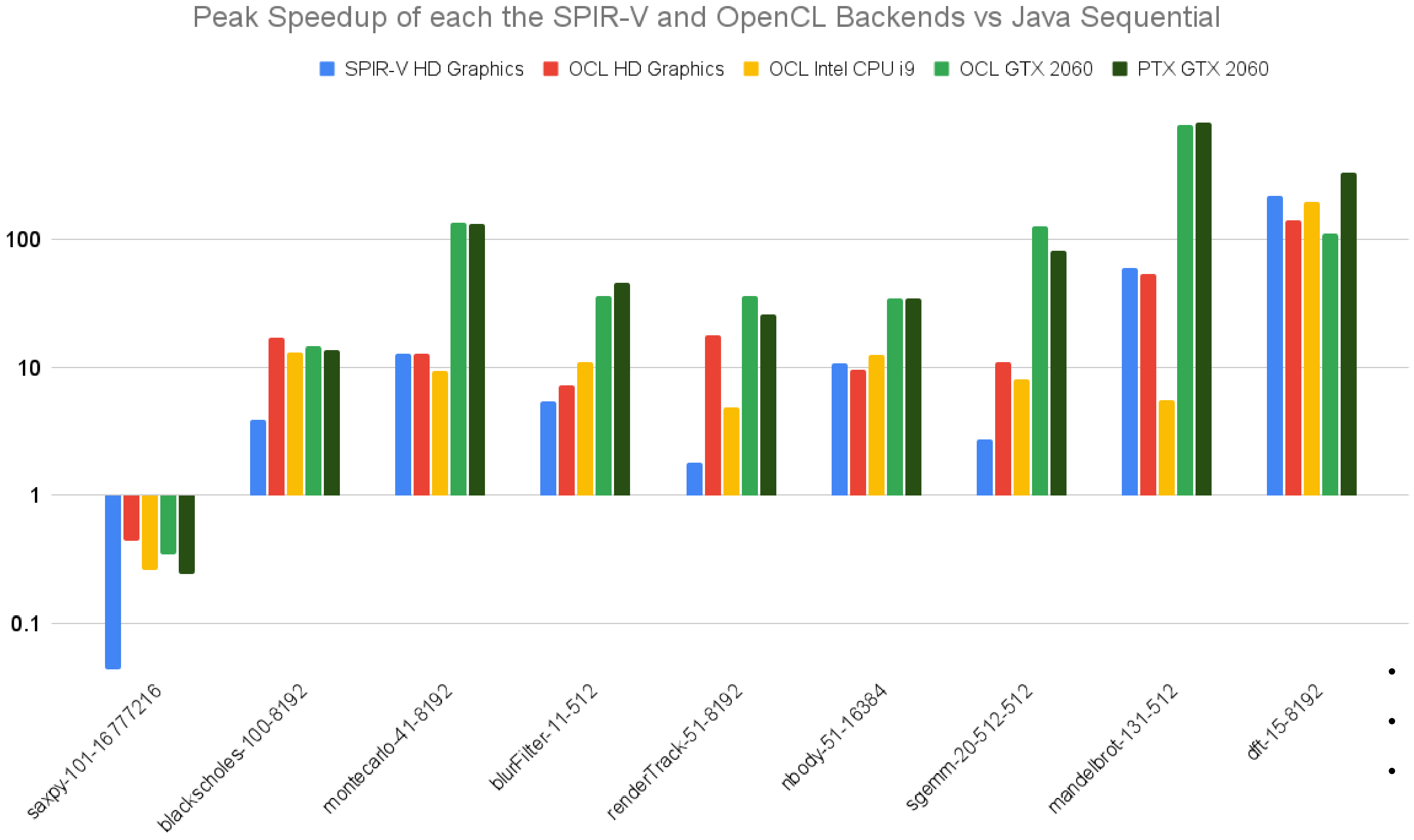
Performance – JMH Benchmarking



- Intel HD Graphics 630 (Intel i7-7700HQ)
- Running for ~4h – Report from JMH
- Up to 320x performance
- Level-Zero: 21.38.21026
- SPIRV-1.2
- TornadoVM v0.12

Performance vs OpenCL Backend

MANCHESTER 1824
The University of Manchester



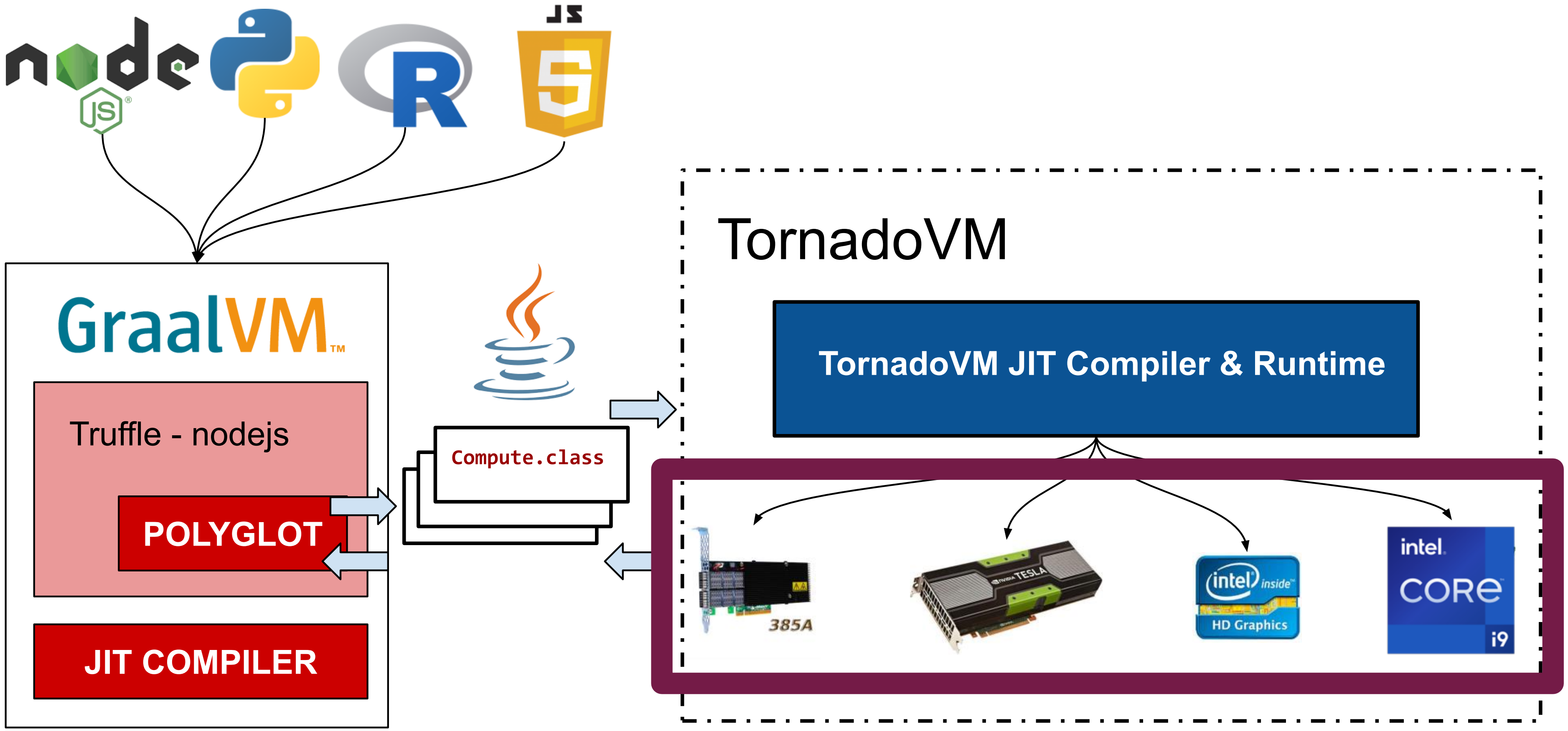
SPIR-V Backend and Level Zero is competitive with the PTX and OpenCL backends

> 200x vs Java Seq.

- Intel HD Graphics 630 (Intel i*9-10885H*)
- GTX 2060
- Level-Zero: 21.38.21026

Running other Programming Languages?

Support for other dynamic languages



Support for other dynamic languages

MANCHESTER
1824
The University of Manchester

```
$ ./graalvm-ce-java8-21.2.0/bin/graalpython [params] mxmWithTornadoVM.py
Running with tornadoVM
Task info: s0.t0
Backend            : SPIRV
Device             : SPIRV LevelZero - Intel(R) UHD Graphics [0x9bc4] GPU
Dims               : 2
Global work offset: [0, 0]
Global work size  : [256, 256]
Local  work size  : [256, 1, 1]
Number of workgroups   : [1, 256]
```

```python
#!/usr/bin/python
print("Running with tornadoVM")
import java
myclass = java.type('MyCompute')
output = myclass.compute()
```

https://www.tornadovm.org/resources

Final remarks

Areas of Interest

https://www.tornadovm.org/use-cases

MANCHESTER 1824
The University of Manchester

Levenshtein Distance	9.8x
K-Means	9.7x
Hierarchical Clustering	28x

Natural Language Processing

449x
Overall Performance Gains

1264 endpoints
Hardware IoT Deployment

IoT and smart buildings

DFT	4500x

Digital Signal Processing

NBody	> 2000x

Physics Simulation

DeepNets	6x and 88x (kernel)
Logistic Regression	14x

Machine Learning and Deep Learning

SLAM-Bench	150x
BlurFilter	> 300x
ViolaJones	22x
RenderTrack	80x

Computer Vision

BlackScholes	> 100x
MonteCarlo	> 10x

FinTech

https://e2data.eu/blog

https://e2data.eu/ (Deliverable 6.3)

MPLR 2020: Transparent acceleration of Java-based deep learning engines

VEE 2019: Dynamic application reconfiguration on heterogeneous hardware

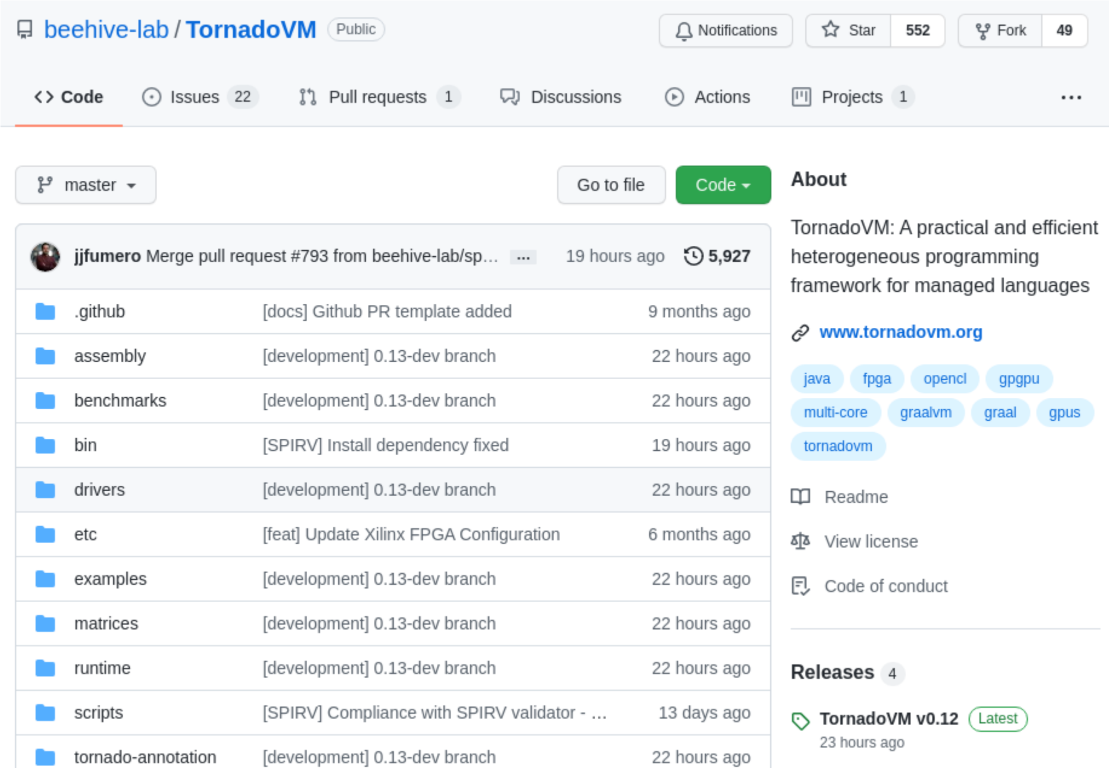
TornadoVM is Open Source and available on **GitHub**

MANCHESTER 1824
The University of Manchester

GPLv2 + CE

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

https://github.com/beehive-lab/tornadovm-installer




```
$ docker pull beehivelab/tornado-gpu

#And run!
$ ./run_nvidia.sh javac.py YourApp
$ ./run_nvidia.sh tornado YourApp
```

https://github.com/beehive-lab/docker-tornado

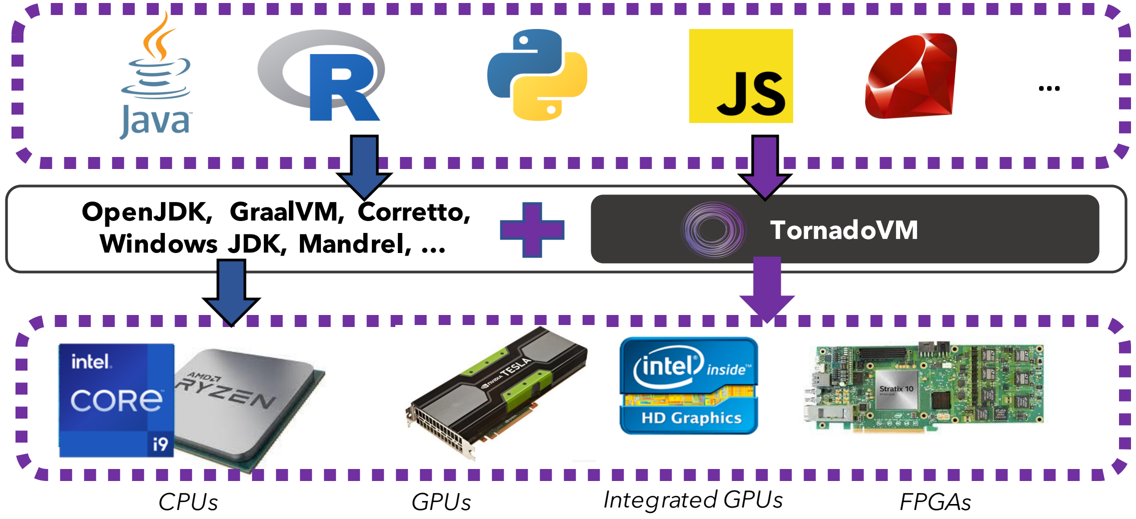
Team

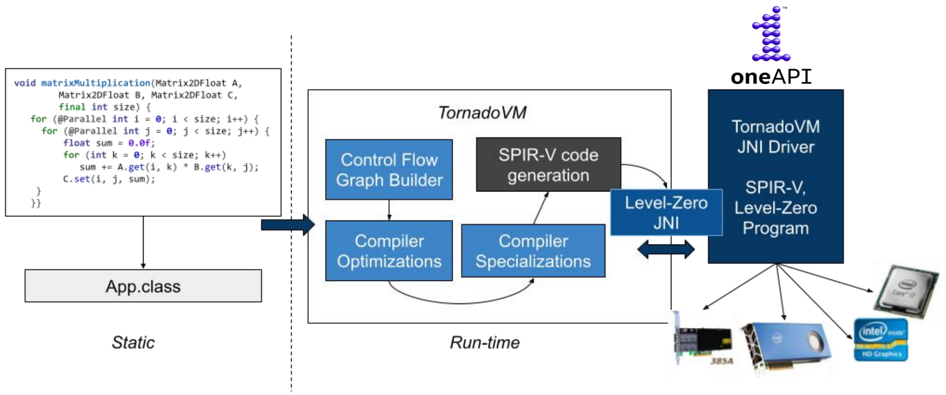
- Academic staff:
  Christos Kotselidis
- Research staff:
  Juan Fumero
  Thanos Stratikopoulos
- PhD Students:
  Maria Xekalaki
- Undergraduate Students:
  Vinh Pham Van
- Master Students:
  Florin Blanaru

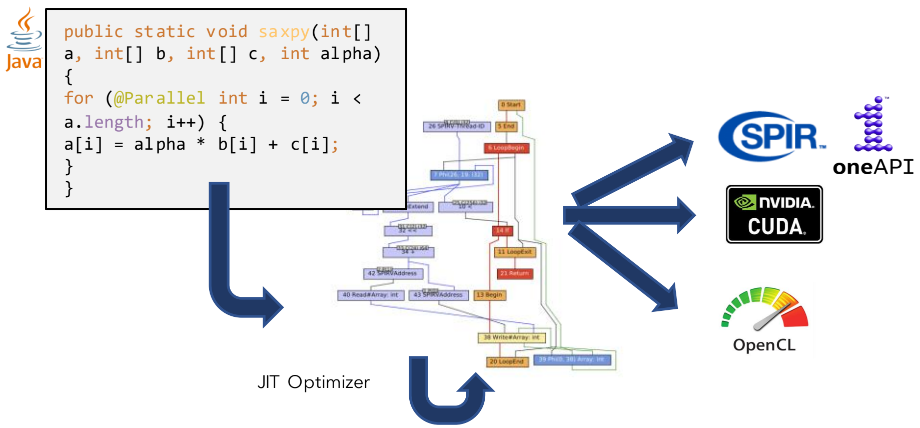
- Alumni:
  Michail Papadimitriou
  James Clarkson
  Benjamin Bell
  Amad Aslam
  Foivos Zakkak
  Gyorgy Rethy
  Mihai-Christian Olteanu
  Ian Vaughan
  Ales Kubicek

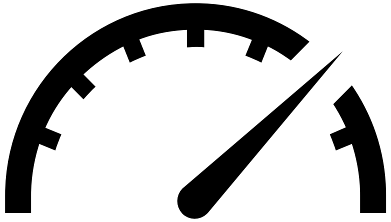
We are looking for collaborations (industrial & academics) -> Let's talk!

Takeaways









> 100x vs standard JVMs tornadovm.org

Thank you so much for your attention

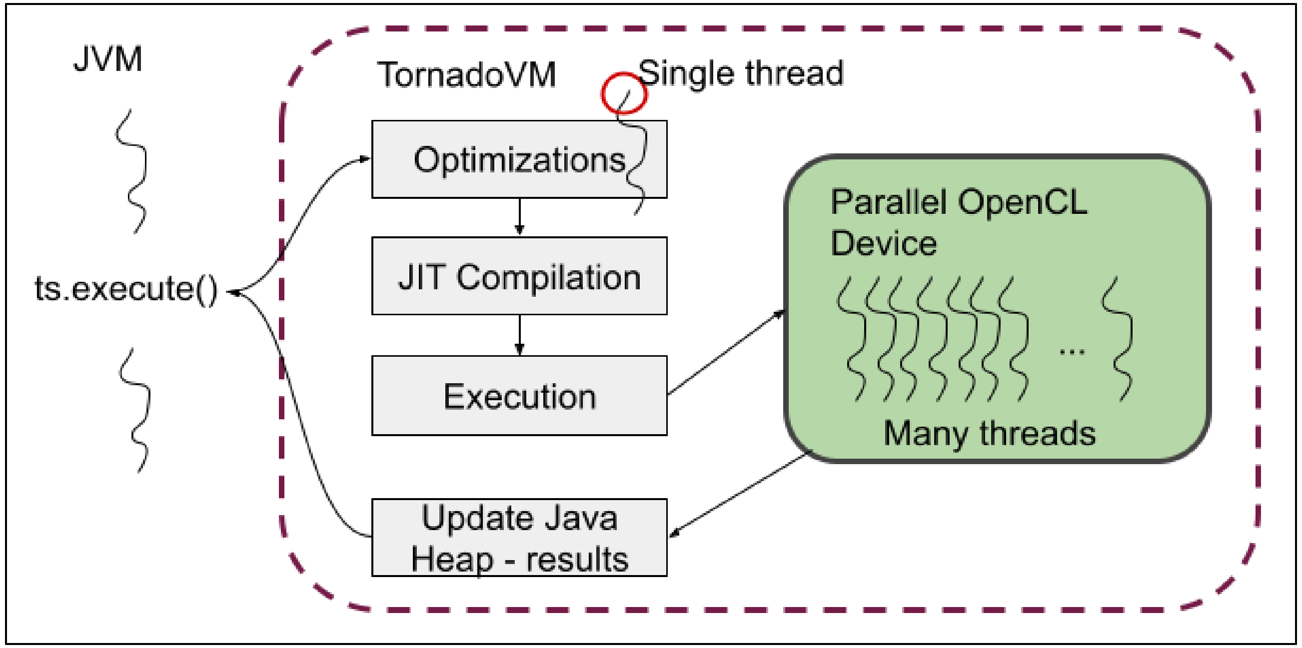
- Partially supported by the EU Horizon 2020:
    - E2Data 780245
    - ELEGANT 957286
- Partially supported by Intel Grant

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

@snatverk

Back up slides

How TornadoVM launches Java kernels on Parallel Hardware?




```java
void blurFilter(. . . ) {

for (@Parallel int r = 0; r < numRows; r++) {
  for (@Parallel int c = 0; c < numCols; c++) {

    computeFilter(. . . );

  }
}
```

Range of NxM threads
2D (numRow, numColumns)
Each thread computes the body of the parallel loop

SPIR-V Beehive Toolkit for code-gen within TornadoVM

- 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 Beehive Toolkit for code-gen within TornadoVM

MANCHESTER 1824
The University of Manchester

- 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
              29, // ID-Generator (new one)
              0,  // Bounds
              0)); // Schema
```

SPIR-V Beehive Toolkit for code-gen within TornadoVM

MANCHESTER 1824
The University of Manchester

- 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
                29, // ID-Generator (new one)
                0,  // Bounds
                0)); // Schema
```

```
; SPIR-V
; Version: 1.2
; Generator: Khronos; 29
; Bound: 77
; Schema: 0
```

SPIR-V Beehive Toolkit for code-gen within TornadoVM

ADD: a + b

```
SPIRVId add = module.getNextId();
blockScope.add(new SPIRVOpIAdd(
            uint,    // type ID
            add,     // result
            id74,    // a
            id75));  // b
```

→

```
%add = OpIAdd %uint %74 %75
```

SPIR-V Beehive Toolkit for code-gen within TornadoVM

ADD: a + b

```
SPIRVId add = module.getNextId();
blockScope.add(new SPIRVOpIAdd(
            uint,    // type ID
            add,     // result
            id74,    // a
            id75));  // b
```

→

```
%add = OpIAdd %uint %74 %75
```

Load a[i]

```
SPIRVId idLoad = module.getNextId();
blockScope.add(new SPIRVOpLoad(
      ptrCrossGroupUint,
      idLoad,
      a_addr, // Load A[i]
      new SPIRVOptionalOperand<>(
        SPIRVMemoryAccess.Aligned(
             new SPIRVLiteralInteger(8)))
));
```

→

```
%idLoad = OpLoad %_ptr_CrossWorkgroup_uint  %addr Aligned 8
```

SPIR-V JIT compiler (and runtime) for TornadoVM

- TornadoVM makes use of the LevelZero JNI and SPIR-V lib libraries.
- Three APIs for TornadoVM:

Pre-built Kernels

```
device = runtime.getDriver(SPIRV.class).getDevice(0);
String filePath = "/tmp/testCopy.spv";
TaskSchedule ts = new TaskSchedule("s0")
  .streamIn(a)
  .prebuiltTask("t0",
                "copyTest", // method to be launched
                filePath,   // path to SPIR-V binary
                new Object[] { a },  // data
                new Access[] { Access.WRITE }, // accessors
                device,  // level-zero device
                new int[] { numElements, 1, 1 })
  .streamOut(a);
ts.execute();
```

Loop Parallelism - JIT

```
public class TestSPIRV {
  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];
    }
  }
}
```

```
new TaskSchedule("s0")
      .task("t0", TestSPIRV::saxpy, a, b, c, 2)
      .streamOut(a)
      .execute();
```

SPIR-V JIT compiler (and runtime) for TornadoVM

- TornadoVM makes use of the LevelZero JNI and SPIR-V lib libraries.
- Three APIs for TornadoVM:

Loop Parallelism - JIT

```
public class TestSPIRV {
  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];
    }
  }
}
```

```
new TaskSchedule("s0")
      .task("t0", TestSPIRV::saxpy, a, b, c, 2)
      .streamOut(a)
      .execute();
```

Parallel Kernel API - JIT

```
public class TestSPIRV {
  public static void saxpy(int[] a, int[] b,
                           int[] c, int alpha,
                           KernelContext context)
{
   int i = context.globalIdx;
   a[i] = alpha * b[i] + c[i];
}
```

```
Grid grid = new Grid(new Worker1D(numThreads));
new TaskSchedule("s0")
      .task("t0", TestSPIRV::saxpy, a, b, c, 2, context)
      .streamOut(a)
      .execute(grid);
```

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);
```

The Intel Level Zero Spec: https://spec.oneapi.io/level-zero/latest/index.html

LevelZero JNI Library for TornadoVM

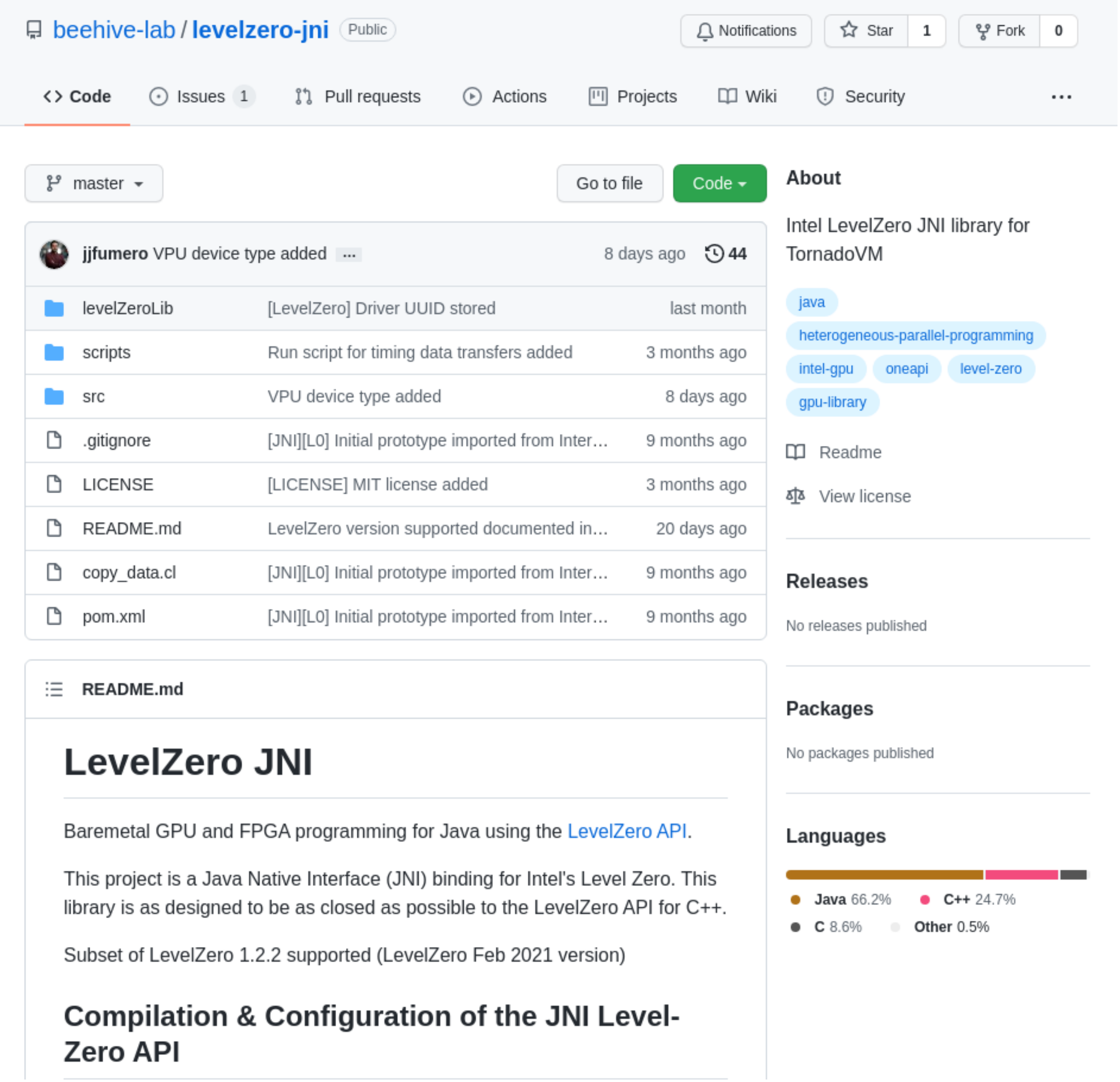
- Level Zero Bridge for TornadoVM
  - Since LevelZero is not stable, we tried to do a 1-1 mapping between the Java API and C-LevelZero.
  - Easy for us to adapt to new changes
  - In near future, we will leverage this API

```
// 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);
```

```
// Create buffer
LevelZeroBufferInteger bufferA = new LevelZeroBufferInteger();
// Declare buffer as a shared memory
result = context.zeMemAllocShared(context.getContextHandle(),      // Level Zero Context
                                  deviceMemAllocDesc,              // Device descriptor
                                  hostMemAllocDesc,                // Host Descriptor
                                  bufferSize,                      // Buffer size in Bytes
                                  1,                               // Alignment
                                  device.getDeviceHandlerPtr(),    // Device pointer
                                  bufferA);                        // Buffer to use
LevelZeroUtils.errorLog("zeMemAllocShared", result);
```

LevelZero JNI Libray for TornadoVM

- This library dispatches SPIR-V kernels
- It does not support full LevelZero, just what we need for TornadoVM, although it could be easy extensible
- It is open source under:
  - **MIT License**



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