From CPU to GPU and FPGAs: Supercharging Java Applications with TornadoVM

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

MANCHESTER 1824
The University of Manchester

7th August 2023

TORNADOVM

Outline

MANCHESTER
1824
The University of Manchester

1. Enabling Acceleration from Managed Runtime Languages
2. Overview of TornadoVM
3. TornadoVM APIs
4. TornadoVM's Key Features
5. What is next? Our Roadmap
6. Call For Action
7. Conclusions

Who am I?

Dr. Juan Fumero

The University of Manchester

Research Fellow @ University of Manchester

Architect and Developer of TornadoVM

oneAPI **Intel Innovator**

- oneAPI Lang SIG
- oneAPI Hardware SIG

oneAPI

juan.fumero@manchester.ac.uk

@snatverk

Former Member of:

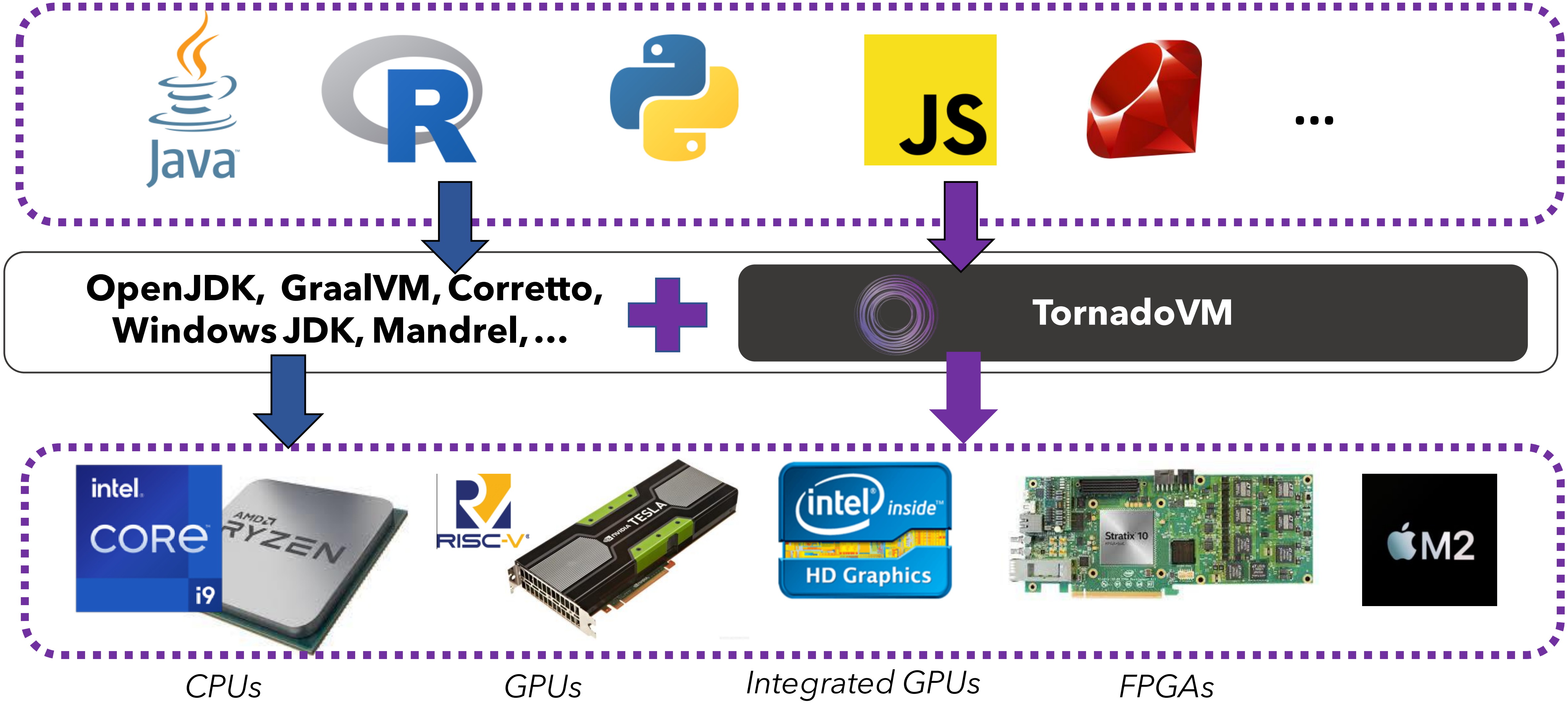
PhD: Java, JIT Compilers for GPUs

Truffle and FastR Team

Intel CilkPlus Vectorization Techniques for Root and GeantV

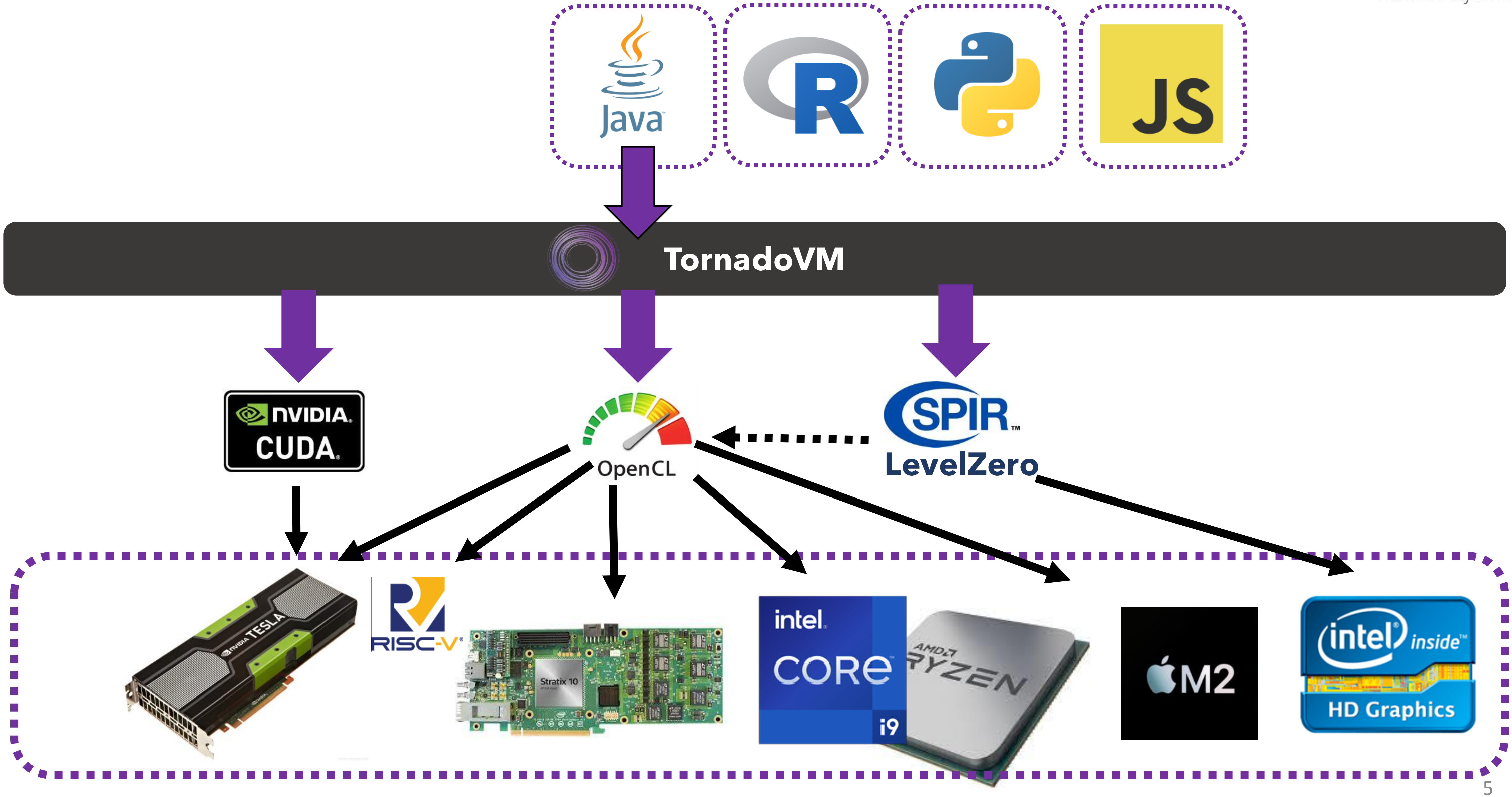
Enabling Acceleration for Managed Runtime Languages

MANCHESTER 1824
The University of Manchester



Enabling Acceleration for Managed Runtime Languages

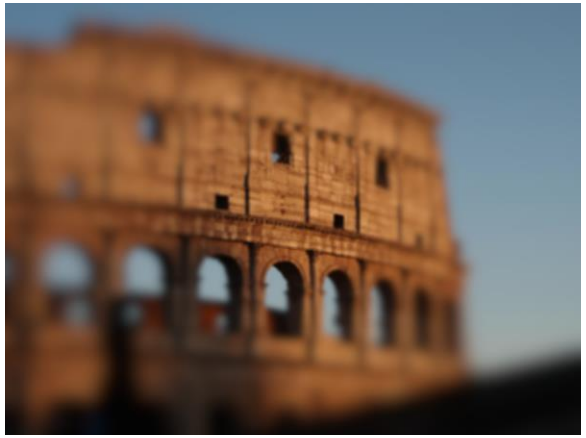
MANCHESTER
1824
The University of Manchester



TORNADOVM

www.tornadovm.org

Example – Blur Filter - Let's run it



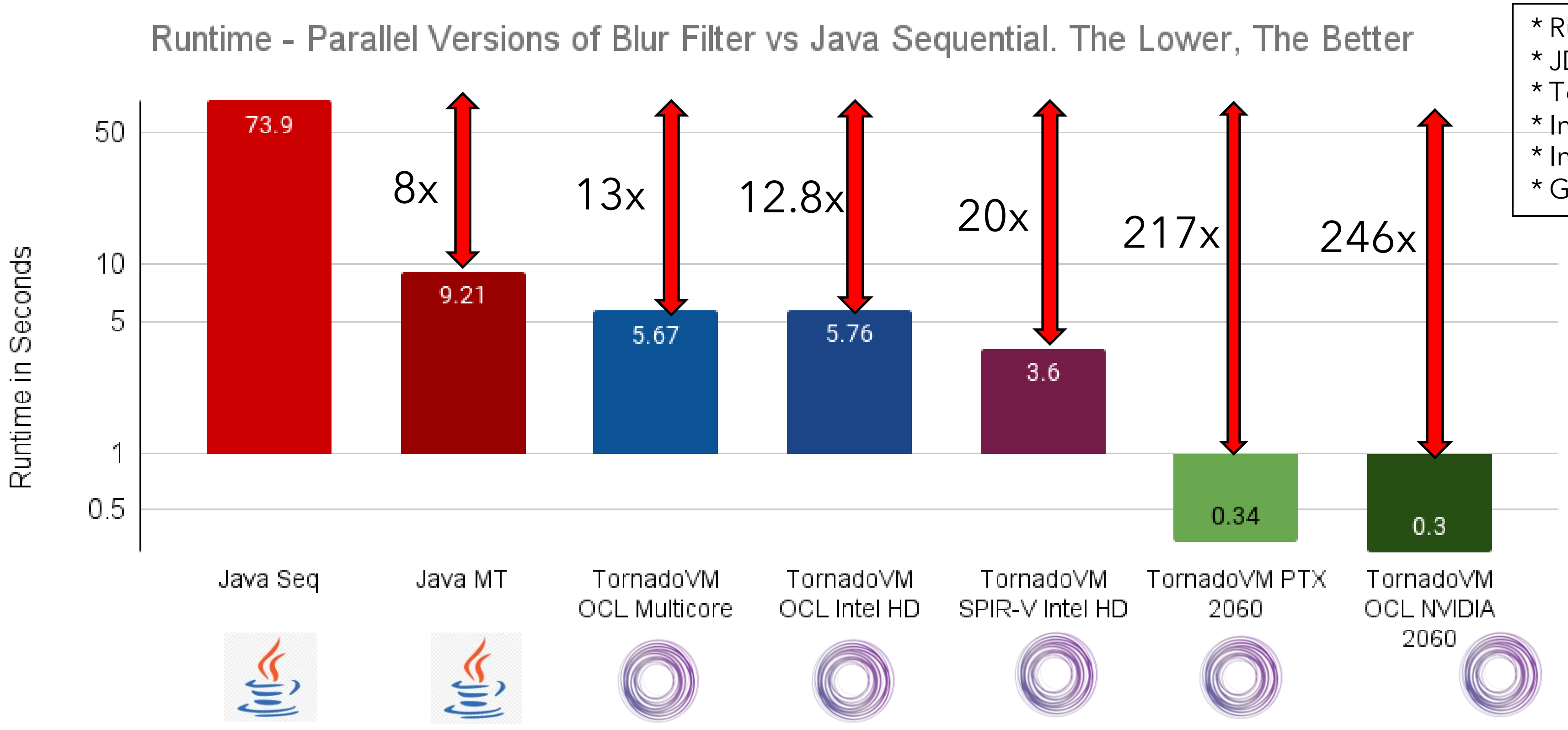
```
$ tornado \
 -cp target/tornadovm-examples-1.0-SNAPSHOT.jar \
 io.github.jjfumero.BlurFilter --tornado
```

The `**tornado**` command is an alias to `java` and all flags for TornadoVM.

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

Blur Filter Performance (on my laptop)

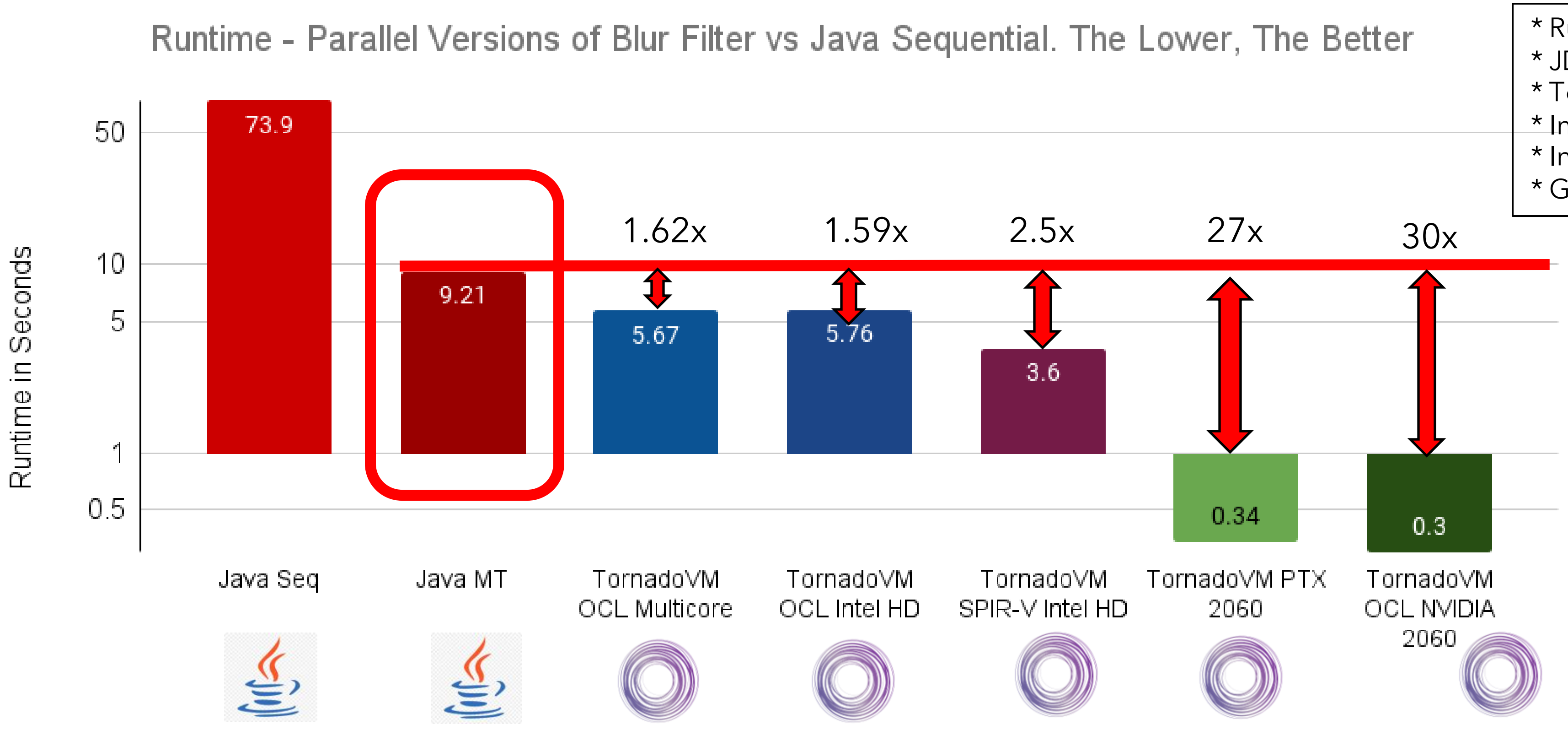
MANCHESTER 1824
The University of Manchester



* Running 5K x 4K image
* JDK 17
* TornadoVM v0.15-0.2-dev
* Intel Core i9-10885H@2.40GHz (16 cores)
* Intel HD 630
* GPU NVIDIA 2060 Mobile

Up to 246x when running with TornadoVM on a GPU

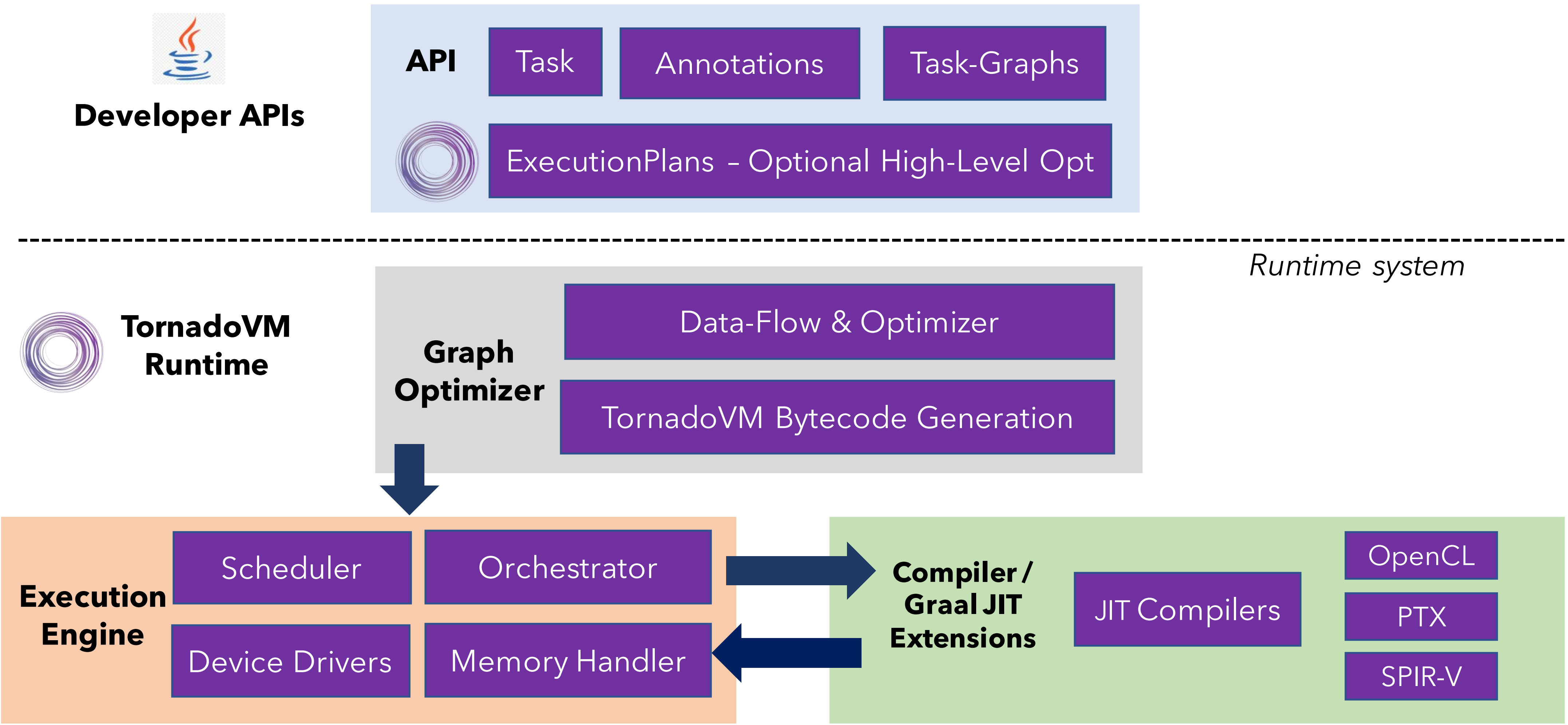
Blur Filter Performance (on my laptop)



* Running 5K x 4K image
* JDK 17
* TornadoVM v0.15-0.2-dev
* Intel Core i9-10885H@2.40GHz (16 cores)
* Intel HD 630
* GPU NVIDIA 2060 Mobile

Up to 30x compared to Java Multi-Thread Stream (16 cores) when running on a GPU

TornadoVM's Software Stack



Different components of the TornadoVM's User API

MANCHESTER 1824
The University of Manchester

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?

Build an **Execution Plan** to define different optimizations

Tornado API – example Java sequential code for MxM

The University of Manchester

```
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 the **Loop Parallel API**

MANCHESTER 1824
The University of Manchester

Device Code

```
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**

Tornado API – example using the **Kernel API**

Device Code

```
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(idx, k) * B.get(k, jdx);
     C.set(idx, jdx, 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

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", Compute::mxm, matrixA, matrixB, matrixC, size)

   .transferToHost(DataTransferMode.EVERY_EXECUTION, matrixC);
```

Host Code
(Runs on CPU)

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

MANCHESTER 1824
The University of Manchester

How to explore different optimizations? --> ExecutionPlan

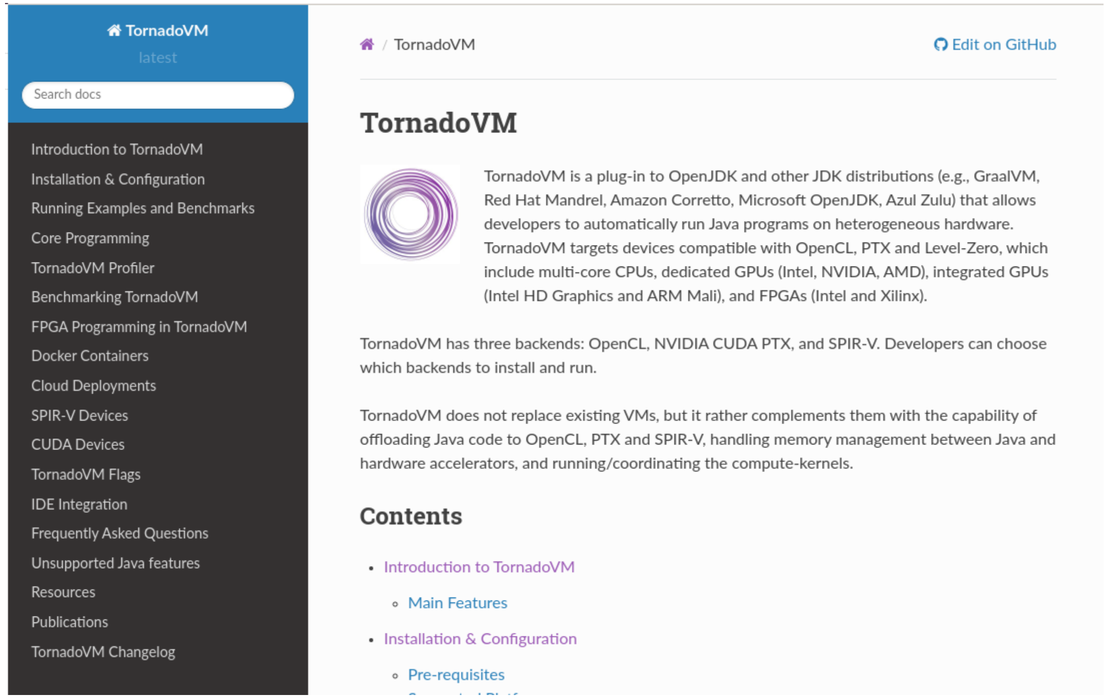
```
ImmutableTaskGraph itg = taskGraph.snapshot();

TornadoExecutionPlan executionPlan = new TornadoExecutionPlan(itg);

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

TornadoExecutionResult result = executionPlan.execute();

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

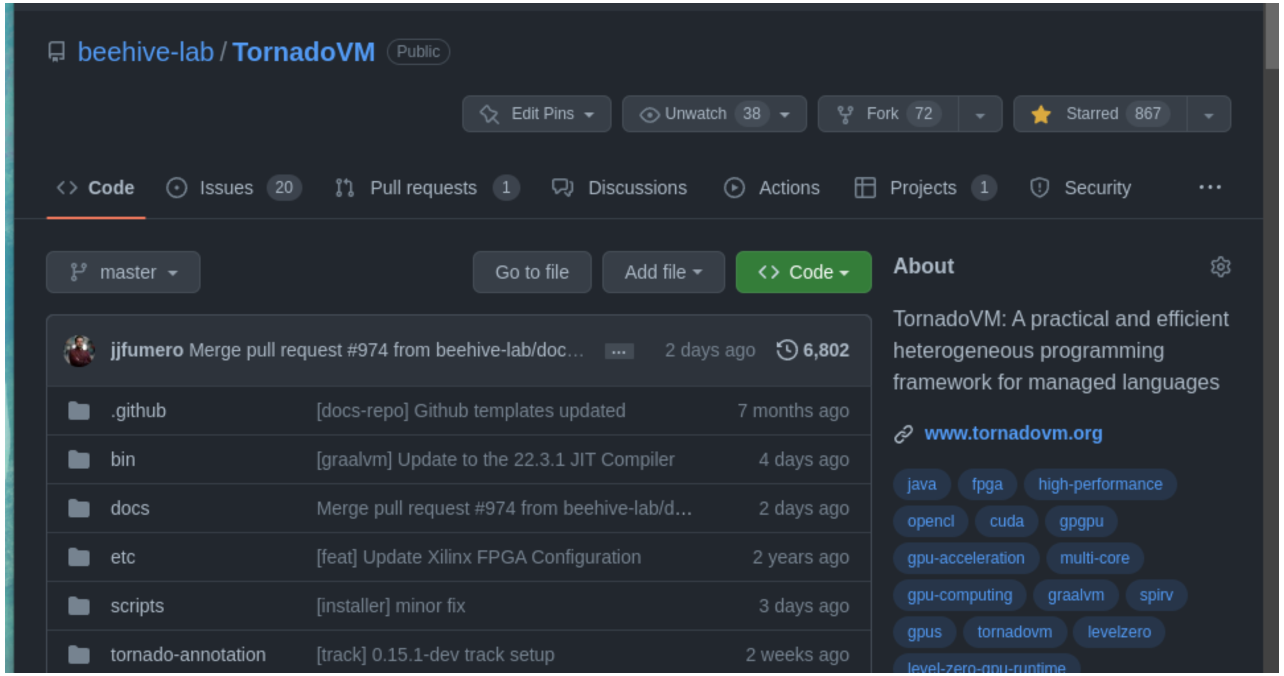
Host Code

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 learn more about the APIs





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

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

TornadoVM JIT Compiler

Multi-backend JIT Compiler Workflow

```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];
    }
}
```

```java
TaskGraph tg = new
TaskGraph("t")  .transferToDevice(FIRST
_EXECUTION, a, b)
.task("saxpy", Klass:saxpy, a, b, c)
.transferToHost(EVERY_EXECUTION, c);
```

```java
TornadoExecutorPlan tep = new
TorandoExecutorPlan(tg.snaphopt());

tep.execute();
```

Programmer's view *Annotations* + *Task-Graphs* + *Executor Plans*

Multi-backend JIT Compiler Workflow

```
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];
    }
}
```

```
TaskGraph tg = new
TaskGraph("t")  .transferToDevice(FIRST
_EXECUTION, a, b)
.task("saxpy", Klass:saxpy, a, b, c)
.transferToHost(EVERY_EXECUTION, c);
```

```
TornadoExecutorPlan tep = new
TorandoExecutorPlan(tg.snaphopt());

tep.execute();
```

Programmer's view

Annotations + *Task-Graphs* + *Executor Plans*

javac

Java Bytecodes

Static Compilation: No Modifications in Javac

Multi-backend JIT Compiler Workflow

```
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];
    }
}
```

```
TaskGraph tg = new
TaskGraph("t")  .transferToDevice(FIRST
_EXECUTION, a, b)
.task("saxpy", Klass:saxpy, a, b, c)
.transferToHost(EVERY_EXECUTION, c);
```

```
TornadoExecutorPlan tep = new
TorandoExecutorPlan(tg.snaphopt());

tep.execute();
```

Programmer's view

Annotations + *Task-Graphs* + *Executor Plans*

javac **_Java Bytecodes_** *Static Compilation: No Modifications in Javac*

TornadoVM JIT Compiler

Graal IR

TornadoVM Common IR

Multi-backend JIT Compiler Workflow

```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];
    }
}
```

```java
TaskGraph tg = new
TaskGraph("t")  .transferToDevice(FIRST
_EXECUTION, a, b)
.task("saxpy", Klass:saxpy, a, b, c)
.transferToHost(EVERY_EXECUTION, c);
```

```java
TornadoExecutorPlan tep = new
TorandoExecutorPlan(tg.snaphopt());

tep.execute();
```

Programmer's view — *Annotations* + *Task-Graphs* + *Executor Plans*

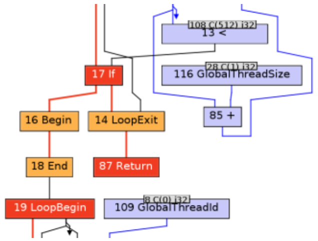
javac — ***Java Bytecodes*** — *Static Compilation: No Modifications in Javac*

TornadoVM JIT Compiler

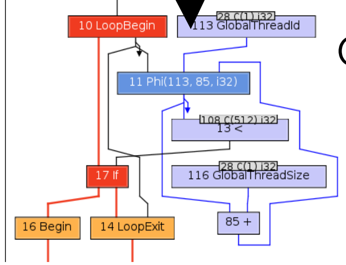
Graal IR

TornadoVM Common IR -> **Sketch**

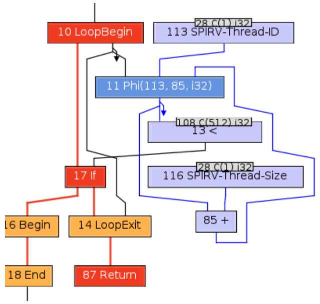
TornadoVM IR for PTX



TornadoVM IR for OpenCL



TornadoVM IR for SPIR-V

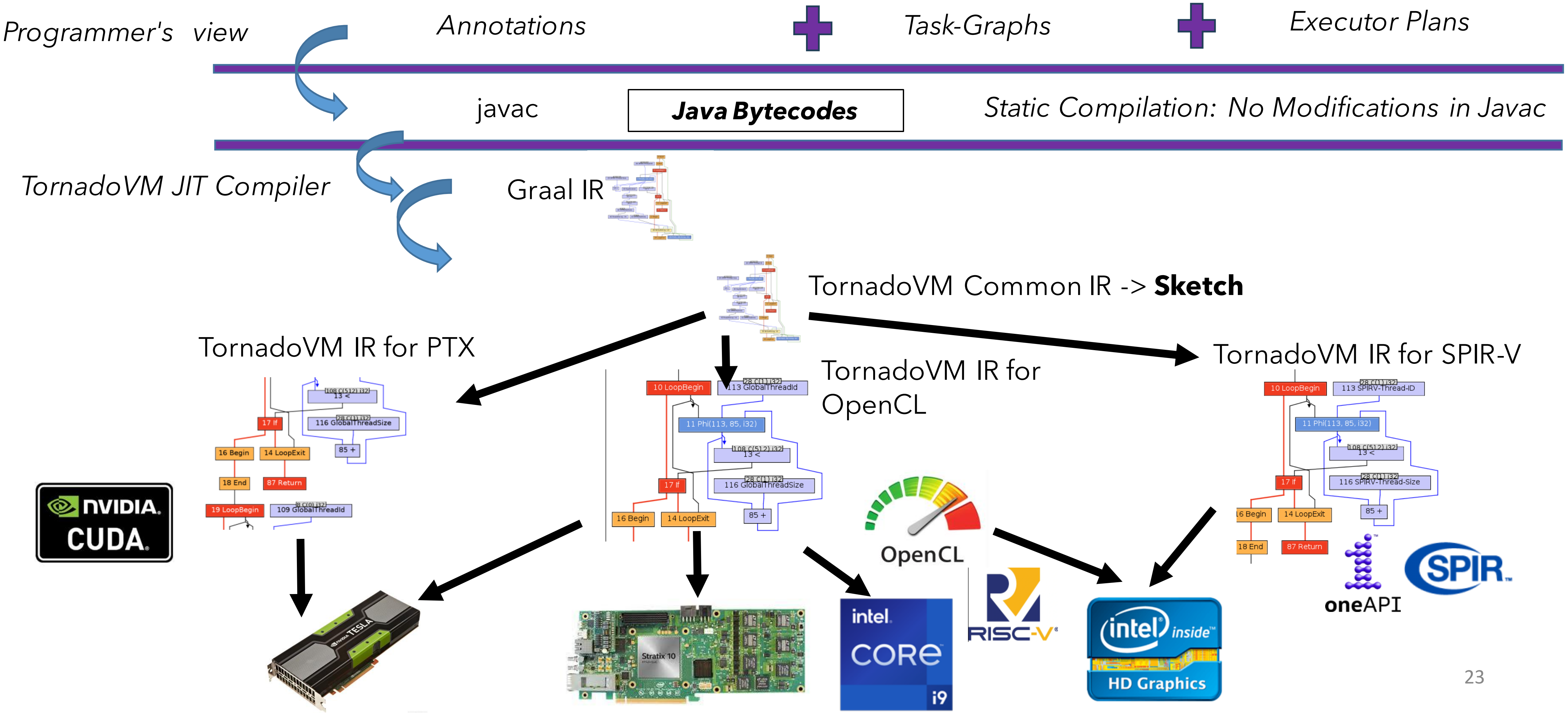


Multi-backend JIT Compiler Workflow

```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];
    }
}
```

```java
TaskGraph tg = new
TaskGraph("t")  .transferToDevice(FIRST
_EXECUTION, a, b)
.task("saxpy", Klass:saxpy, a, b, c)
.transferToHost(EVERY_EXECUTION, c);
```

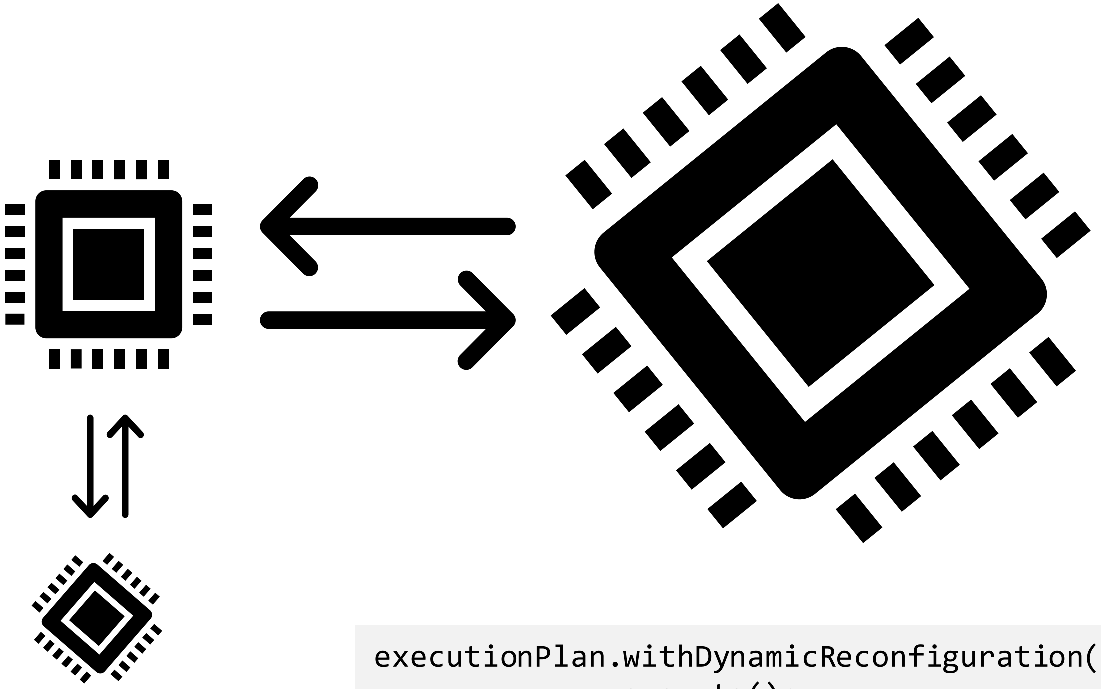
```java
TornadoExecutorPlan tep = new
TorandoExecutorPlan(tg.snaphopt());

tep.execute();
```



Key Features

1) Live Task Migration (aka Dynamic Reconfiguration)

The University of Manchester



Automatically migrates execution context from one device to another during runtime. Explores the best possible device for each task-graph in terms of performance.

In near future: dynamic reconfiguration for energy efficiency

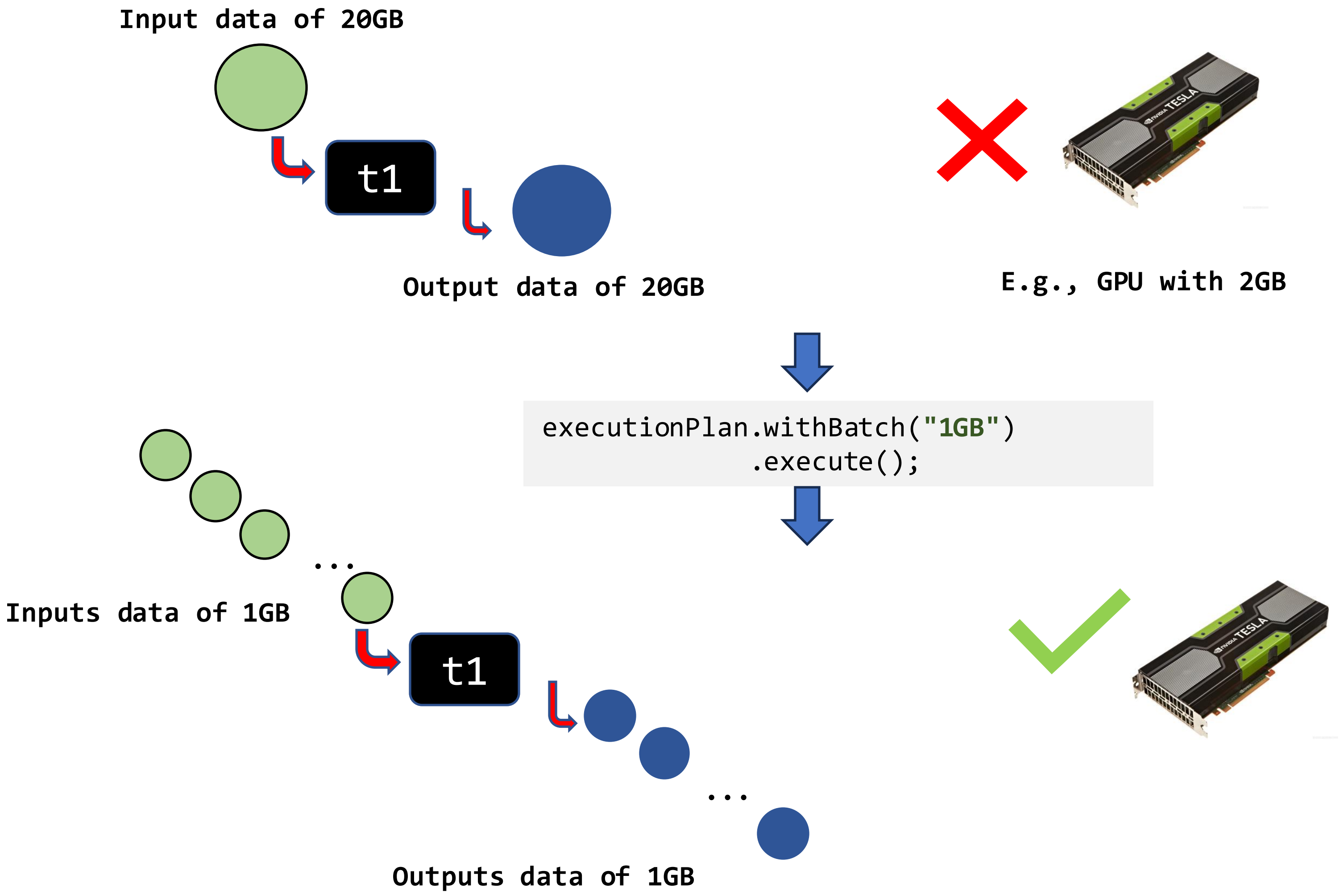
```
executionPlan.withDynamicReconfiguration(Policy.PERFORMANCE, DRMode.PARALLEL)
            .execute();
```

The goal is to only switch when only when it offers higher performance -> Up to 7.7x faster over static configurations (**VEE'19**).

VEE 2019: Dynamic Application Reconfiguration on Heterogeneous Hardware

2) Batch Processing for Big Data Workloads

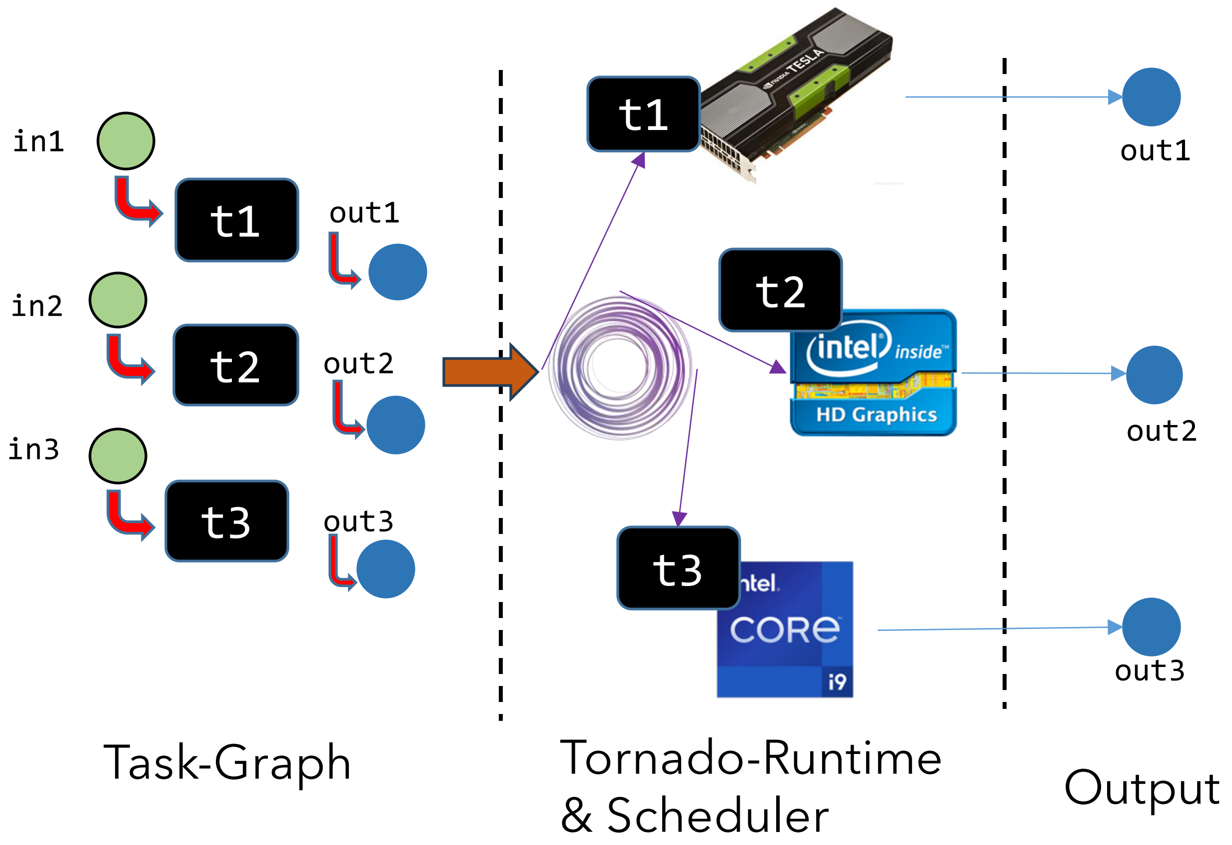
MANCHESTER 1824 The University of Manchester




```
executionPlan.withBatch("1GB")
             .execute();
```

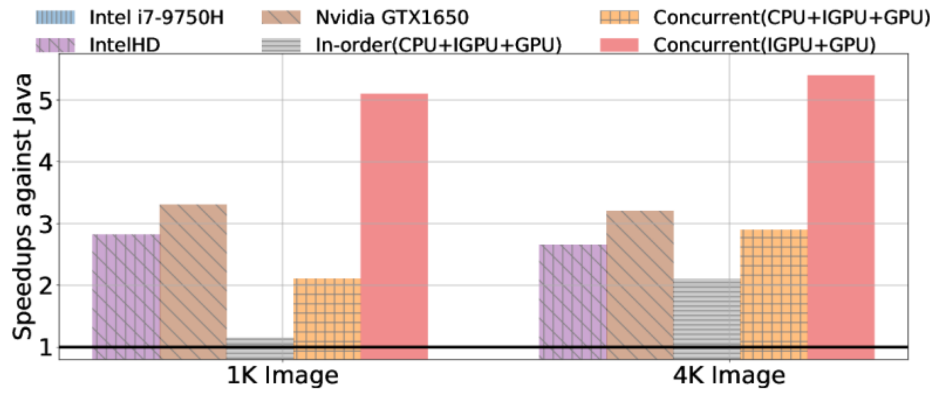
It can be also used to give "space" for other GPU applications

3) Multi-Device Execution for Single TaskGraphs



Multi-GPU Blur Filter is > **5x faster compared to a single GPU**

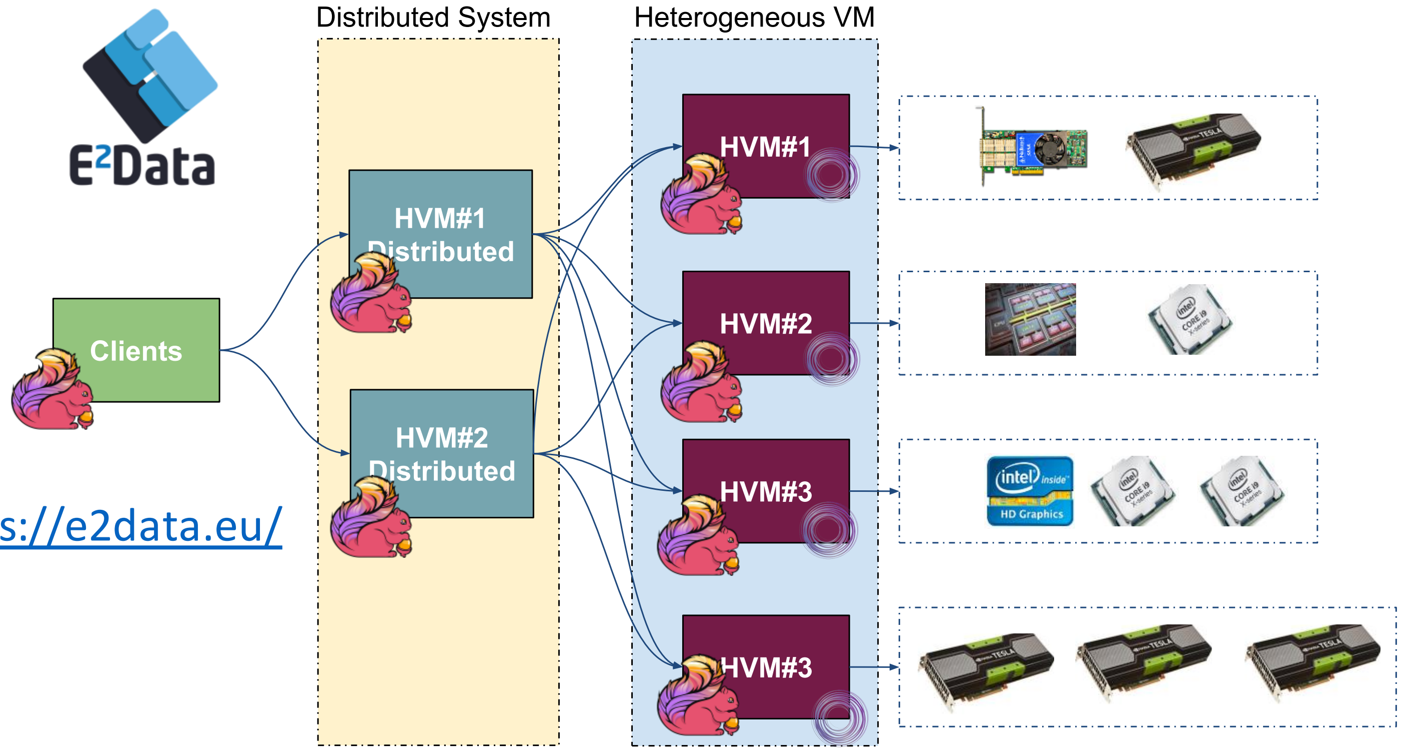
Transparent Multi-device Selection



Multiple-tasks on multiple-devices (MTMD): exploiting concurrency in heterogeneous managed runtimes. VEE'21

4) Integration with Big Data Platforms (Flink) [Experimental]

MANCHESTER 1824 The University of Manchester



https://e2data.eu/

Unmodified Flink code
accelerated on GPUs and FPGAs with TornadoVM

Accelerations of up to 65x on GPUs and 184x on FPGAs

Enabling Transparent Acceleration of Big Data Frameworks Using Heterogeneous Hardware. Maria Xekalaki, Juan Fumero, Athanasios Stratikopoulos, Katerina Doka, Christos, Katsakioris, Constantinos Bitsakos, Nectarios Koziris, Christos Kotselidis. ***VLDB23***

https://jjfumero.github.io/posts/2022/11/enabling-transparent-acceleration-bigdata-heterogeneous-hardware/

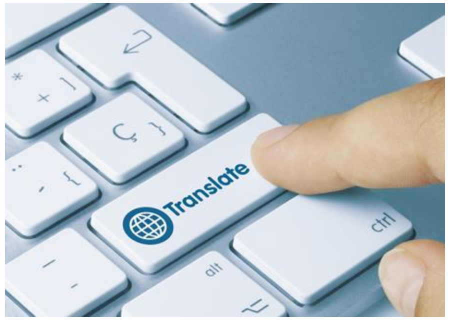
What is next? TornadoVM's Roadmap

1) [Roadmap] Hybrid API: Best of Native and JIT in one API

Options for Parallel Compute on GPUs from Managed Runtime Systems

Pre-built-Libraries (e.g, oneMKL, cuDNN, etc)

Use **vendor** optimized libraries
Not easily portable
Usually very fast and high performance



Full JIT Compiler (e.g., current TornadoVM)

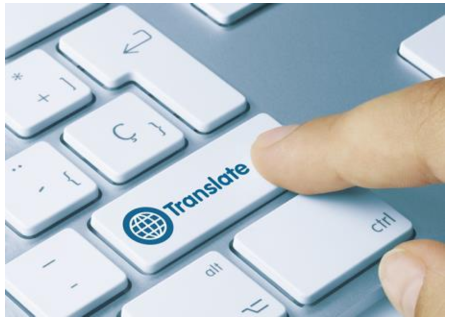
Flexible, Portable, Reusable
Lower Performance

1) [Roadmap] Hybrid API: Best of Native and JIT in one API

MANCHESTER 1824
The University of Manchester

Options for Parallel Compute on GPUs from Managed Runtime Systems

Pre-built-Libraries (e.g, oneMKL, cuDNN, etc)



Full JIT Compiler (e.g., current TornadoVM)

But, what if we combine both?

1) [Roadmap] Hybrid API: Best of Native and JIT in one API

Extension of the TornadoVM APIs for allowing JIT + Library Calls within the same Engine

1) [Roadmap] Hybrid API: Best of Native and JIT in one API

Extension of the TornadoVM APIs for allowing JIT + Library Calls within the same Engine

```
TaskGraph tg = new TaskGraph("compute")
   .transferToDevice(. . .)

   .task ("gemm", MyJavaCompute:sgemm, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

   .transferToHost(. . .);
```

Points to Existing Java Code annotated with
@Parallel
@Reduce

```
public static void sgemm(….) {
 for (@Parallel) {
      for (@Parallel) {
          …
   }
}
```

1) [Roadmap] Hybrid API: Best of Native and JIT in one API

MANCHESTER 1824
The University of Manchester

Example: Invoking SGEMM for Intel oneMKL (oneAPI toolkit)

```
TaskGraph tg = new TaskGraph("compute")
   .transferToDevice( . . . )

   .libraryTask ("gemm", OneMKL:sgemm, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)

   .transferToHost( . . .);
```

Points to a Proxy Class that gives you access to Intel oneMKL

1) [Roadmap] Hybrid API: So, what is the deal?

The University of Manchester

Going Hybrid: JIT + Library Tasks

Uses: Deep Learning, AI, Math Library, Data Bases, etc.

```
TaskGraph tg = new TaskGraph("compute")
 .taskGraph.transferToDevice(DataTransferMode.FIRST_EXECUTION, data)

 .task("prep", MyJavaPrepClass::dataInitOnGPU, data)   // FULL JIT (Java->Accelerator)

 .libraryTask("gemm",  CuDNN::cudnnActivationForward, alpha, data, beta, output) //call to native CuDNN

 .task("postProcessing", MyOtherJavaClass::post, output)   // FULL JIT (Java->Accelerator)

 .transferToHost(DataTransferMode.EVERY_EXECUTION, output);
```

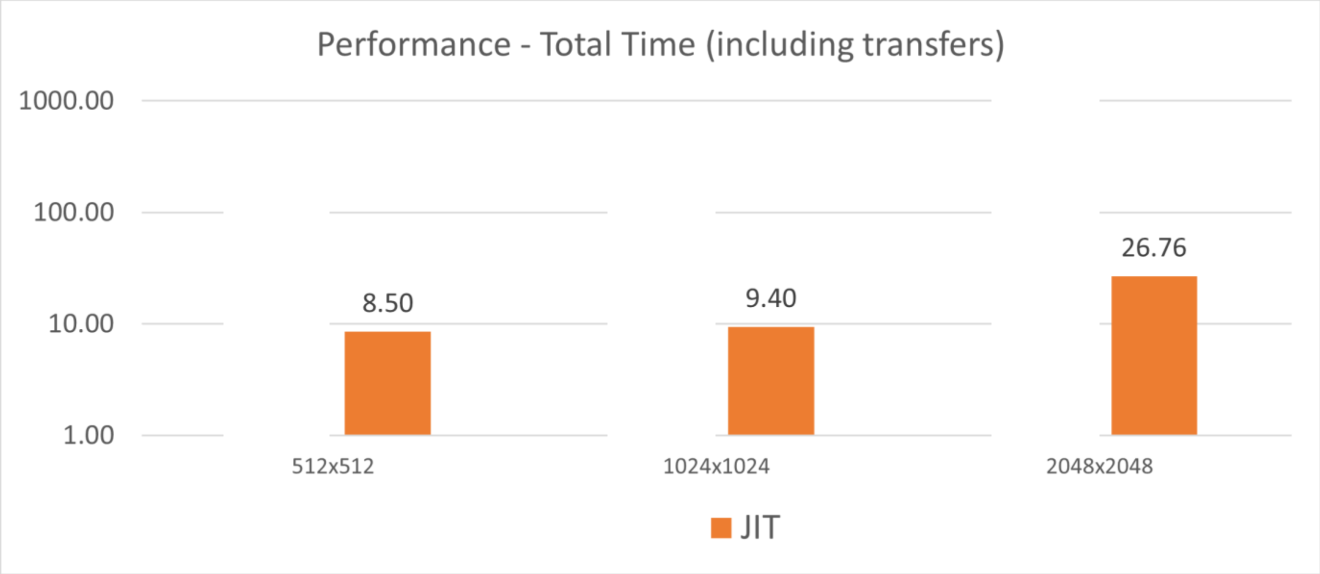
We have prototypes for oneMKL and cuDNN

What about performance? Running SGEMM from oneMKL

Running on Intel i9-10885H
Processors:
 - Intel UHD Graphics 630 ()

TornadoVM 0.15.2-dev

Intel Runtime: 21.38.21026



The Higher, the better. Performance vs single-threaded Java

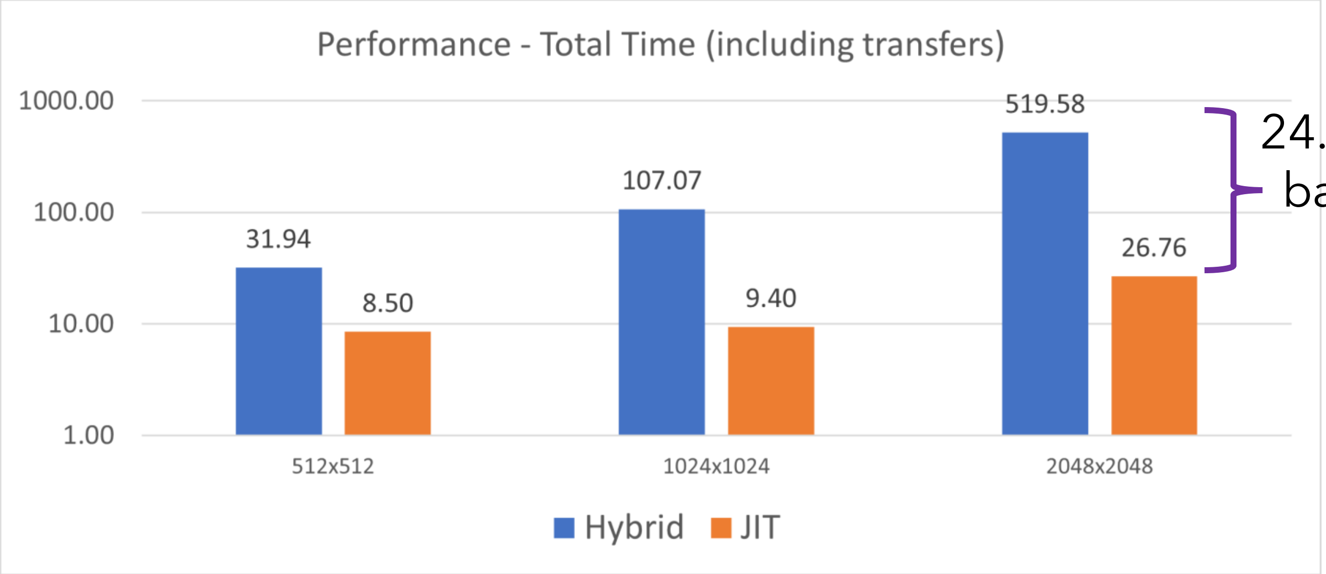
What about performance? Running SGEMM from oneMKL

MANCHESTER 1824
The University of Manchester

Running on Intel i9-10885H
Processors:
- Intel UHD Graphics 630 ()

TornadoVM 0.15.2-dev

Intel Runtime: 21.38.21026



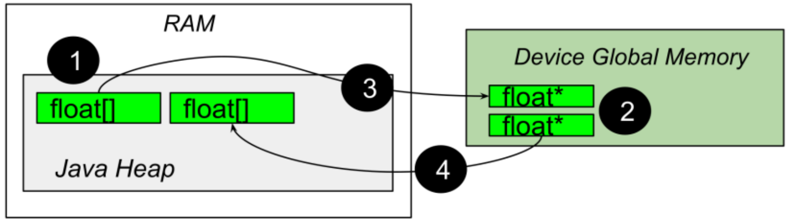
24.7x on top of the base TornadoVM Performance

The Higher, the better. Performance vs single-threaded Java

2) [Roadmap] Integration with Panama, but why?

MANCHESTER 1824
The University of Manchester

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



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 with the GC not to move pointers

LOCK GC and blocking operations

Off-heap Data Structures (e.g., Using Direct Memory or **Panama APIs**)

Besides, we have experimented with other ideas such as :
Unified Shared Memory Java Heap

Unified Shared Memory: Friend or Foe? Juan Fumero, Florin Blanaru, Athanasios Stratikopoulos, Steve Dohrmann, Sandhya Viswanathan, Christos Kotselidis. ***MPLR23***

2) [Roadmap] Integration with the Panama APIs

The University of Manchester

User code:

```
public static void add(FloatArray a, FloatArray b, FloatArray c) {
    for (@Parallel int i = 0; i < c.getSize(); i++) {
      c.set(i, a.get(i) + b.get(i));
    }
}
```

Off-heap Data Structures that we could share pointers with GPUs/FPGAs:

- Unified Memory
- Shared Memory

Example of Implementation

```
public FloatArray(int numberOfElements) {
    this.numberOfElements = numberOfElements;
    segment = MemorySegment.allocateNative(numBytes, ResourceScope.globalScope());
}
```

Next? Combine Panama off-heap types with Hybrid API -> Fast Data Flow across JIT and Library Tasks

```
TaskGraph tg = new TaskGraph("compute")
   .taskGraph.transferToDevice(DataTransferMode.FIRST_EXECUTION, data)
   .task("prep", MyJavaPrepClass::dataInitOnGPU, data)   // FULL JIT
   .libraryTask("gemm", CuDNN::cudnnActivationForward, alpha, data, beta, output) // call to CuDNN
   .transferToHost(DataTransferMode.EVERY_EXECUTION, output);
```

But is TornadoVM Enough?
Call For Action

Call For Action

A) Tight Collaboration with the Oracle Graal Core Team

- For example, via Special Interest Groups (SIGs)

- We already do something similar with **Intel oneAPI:** TornadoVM team participates at the Intel oneAPI/Level Zero SPECs

- We could get a sense of near future changes and plan ahead in our team

Call For Action

A) Tight Collaboration with the Oracle Graal Core Team

- For example, via Special Interest Groups (SIGs)

- We already do something similar with **Intel oneAPI:** TornadoVM team participates at the Intel oneAPI/Level Zero SPECs

- We could get a sense of near future changes and plan ahead in our team

B) Standardize Panama Types for AI and Heterogeneous Systems

- Each vendor can implement those types for different hardware accelerators

- Easier interaction with AI and Deep Learning Frameworks (PyTorch, DeepNetts, etc.)

- Enable types for AI
  - NDArrays
  - Float16 (half float)
  - Explicit Vector Types (vectorFloat4)

Open Discussion: Now, Let's Imagine the Future

How would the Java Platform evolve for the AI/Heterogeneous compute era?

- Goal: make GPU/FPGA acceleration compatible with JVM Spec

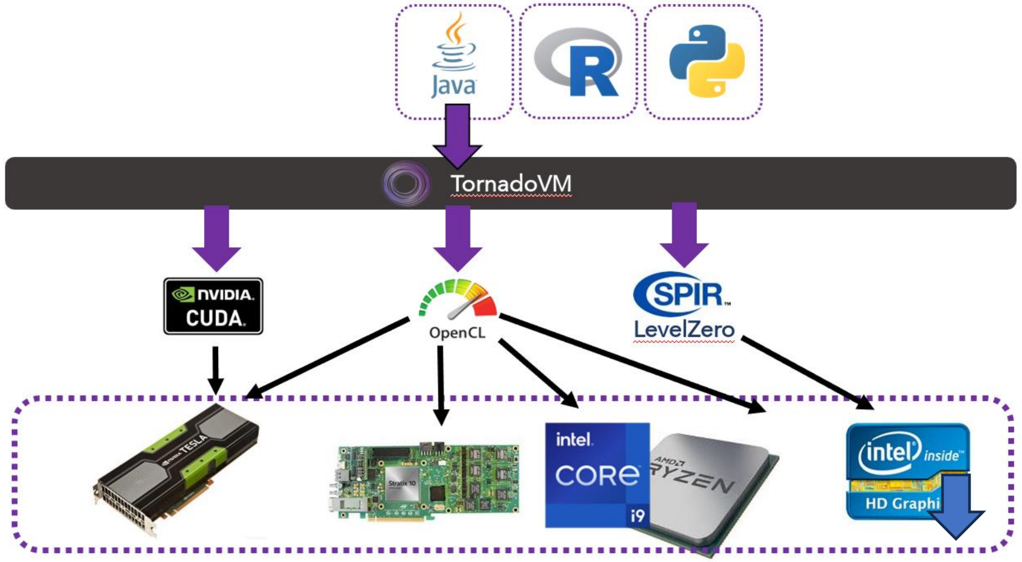
- Defining Math Precision for AI and HC operations: similar work done for the Vector API

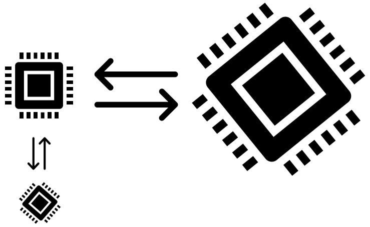
- Possible implementation: providing nested environments (or DSL within Java) Similar to LINQ for the .NET platform but for Heterogeneous compute executions

To conclude

Key Takeaways

MANCHESTER 1824
The University of Manchester





Live Task Migration

Interaction with BD frameworks

Batch Processing

Multi-Backend
Multi-Device

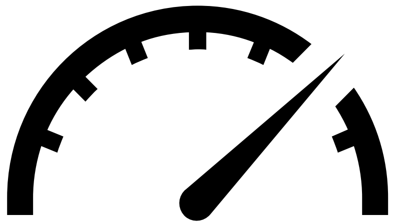
Roadmap

-**Hybrid API**
-Integration with **Panama**

Enable AI Workloads within Java

Call For Action:

A) Graal SIG?
B) Unified Panama Types
C) Discussions : AI math precisions, levels of isolation. ..



> 100x vs standard JVMs

tornadovm.org

Collaborations and Projects

MANCHESTER 1824
The University of Manchester

ELEGANT
Project Number: 957286

AERO
Project Number: 101092850

INCODE
Project Number: 101093069

encrypt
Project Number: 101070670

TANGO
Project Number: 101070052

INTEL
oneAPI

UK Research and Innovation

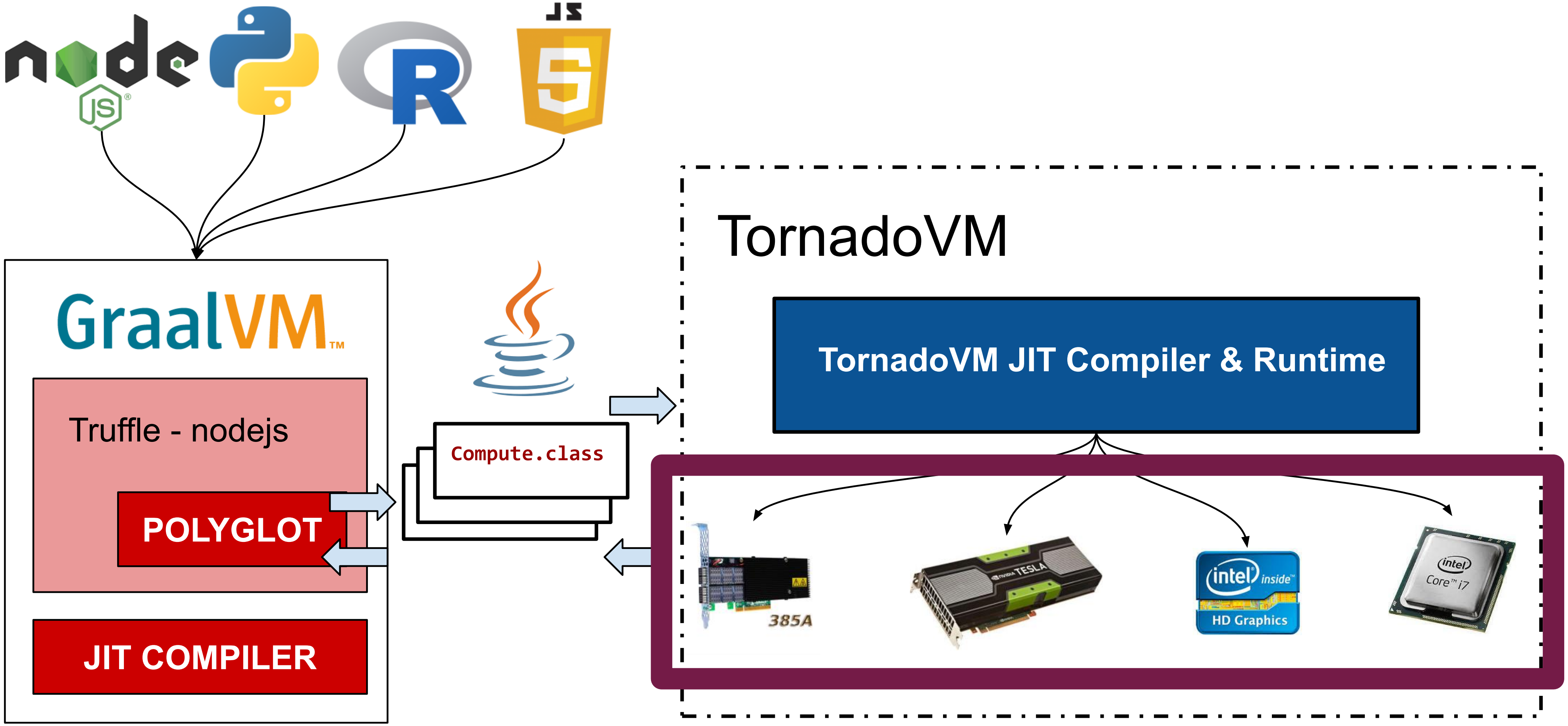
Thank you!

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

@snatverk

Back up slides

TornadoVM & Dynamic Languages



TornadoVM & Dynamic Languages

https://tornadovm.readthedocs.io/en/latest/truffle-languages.html

(*)

```
$ $JAVA_HOME/bin/gu install js
$ tornado --threadInfo --truffle js mxmWithTornadoVM.js

Hello TornadoVM from JavaScript!
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  : [512, 512]
    Local  work size  : [256, 1, 1]
    Number of workgroups  : [2, 512]

Total Time (s): 0.8159999847412109
Task info: s0.t0
```

JS

```
console.log("Hello TornadoVM from JavaScript!")

var comp = Java.type('MyCompute')
var start = new Date().getTime() / 1000;
comp.compute()
var end = new Date().getTime() / 1000;
console.log("Total Time (s): " + (end - start))
```