sidion

SIDION

Zuhören. Analysieren. Beraten.

ÜBER UNS

Matthias Koch, sidion GmbH

matthias.koch@sidion.de

Expert Software Developer

Java

Clean Code

Funktionale Programmierung

Künstliche Intelligenz

ÜBER UNS

Robert Palma, sidion GmbH

robert.palma@sidion.de

Software Developer

Java

Software Architektur

Verteilte Systeme

Künstliche Intelligenz

SOLLEN WIR LIEBER NACH BABYLON ODER NACH PANAMA FAHREN?

Matthias Koch und Robert Palma I Java Forum Stuttgart I 09.07.2026

sidion

PROJEKT PANAMA

JEP 454: Foreign Memory Access API (FFM)
Finalisiert mit JDK 22.

~~JEP 529: Vector API~~
~~Ist immer noch im Incubator Status~~

JEP454

- **Summary**

Introduce an API by which Java programs can interoperate with code and data outside of the Java runtime. By efficiently invoking foreign functions (i.e., code outside the JVM), and by safely accessing foreign memory (i.e., memory not managed by the JVM), the API enables Java programs to call native libraries and process native data without the brittleness and danger of JNI.

FOREIGN MEMORY ACCESS API (FFM)

Ziel: Vereinfache die Interaktion zischen Java und Foreign (non-Java) Api.
z.B. native Code in (C, C++,Rust, ...)

Bisher: Foreign Functions wurden über Java Native Interface (JNI) aufgerufen.

JNI

Aber: JNI (seit Java 1.1!) bedeudet

- Komplexer Glue Code
- Unsicherer Speicherzugriff
- JVM Crashes bei Fehlern
- …

ON-HEAP UND OFF-HEAP SPEICHER

On-Heap Speicher:

- Speicher im Java Heap, der durch den Garbage-Collector verwaltet wird

Off-Heap Speicher:

- Speicher der außerhalb des Java Heaps liegt -> nicht unter Kontrolle des Garbage-Collectors
- Argumente, die an native Code übergeben werden, müssen hier liegen
- Zugriff auf MemorySegment und volle Kontrolle über die Deallocation des Off-Heap Speichers

MEMORYSEGMENT UND ARENA

- FFM API befindet sich im package java.lang.foreign

MemorySegment:

- Zusammenhängender OffHeap-Speicher
- Sicherer, strukturierter Zugriff auf FFM

Arena:

- Managed den Lebenszyklus von MememorySegments

- Arena.ofConfined()
- Nur aktueller Thread hat Zugriff
- Arena muss explizit geschlossen werden

LINKER UND FUNCTIONDESCRIPTOR

- Linker linker = Linker.nativeLinker();
- Zugriff auf native Libraries

- SymbolLookup stdLib = linker.defaultLookup();
- Zugriff auf C Standard Library

- MemorySegment address = stdLib.findOrThrow(<LibraryFunction>);

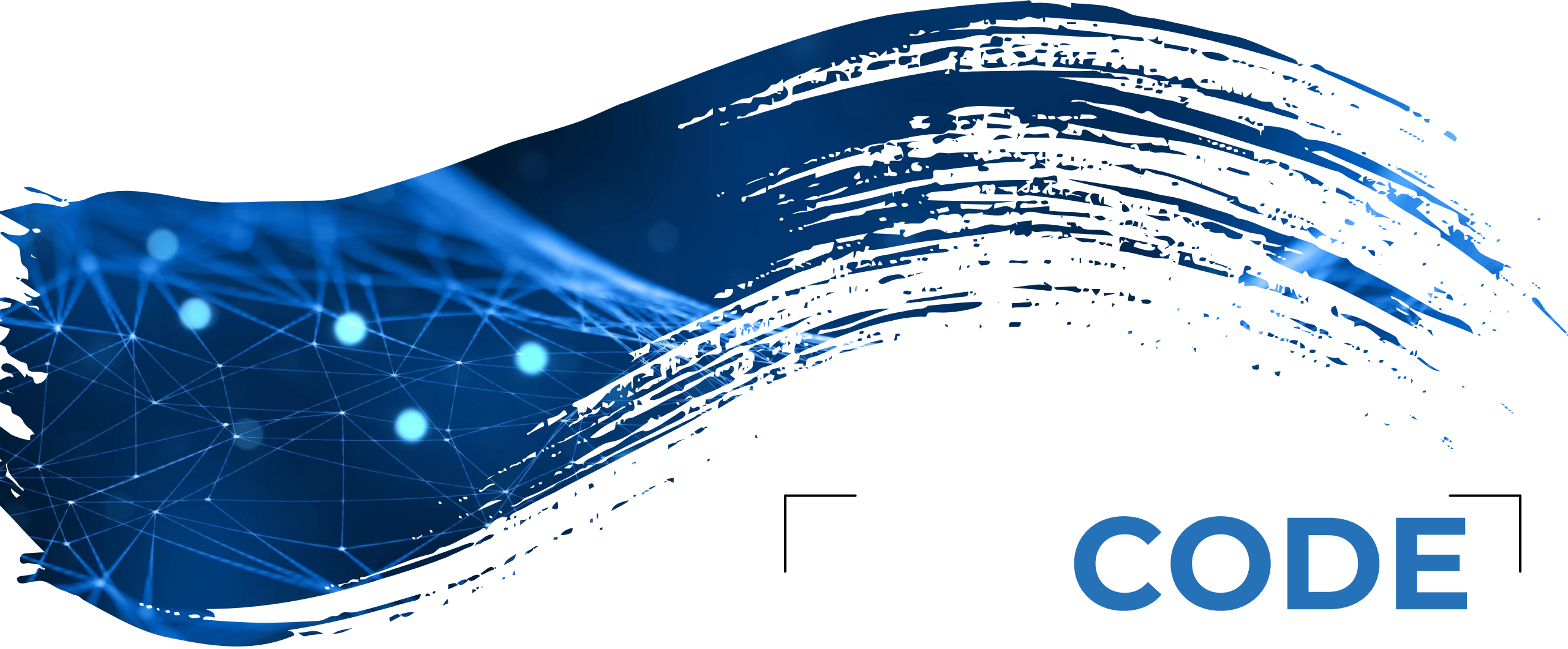
- FunctionDescriptor signature = FunctionDescriptor.of(<Result><Parameter…>);

- MethodHandle handle = linker.downcallHandle(MemorySegment address, FunctionDescriptor signature);

STRLEN

```
size_t strlen(const char *s);
```

size_t -> ValueLayout.JAVA_LONG
char * -> ValueLayout.JAVA_ADDRESS



CODE DEMO

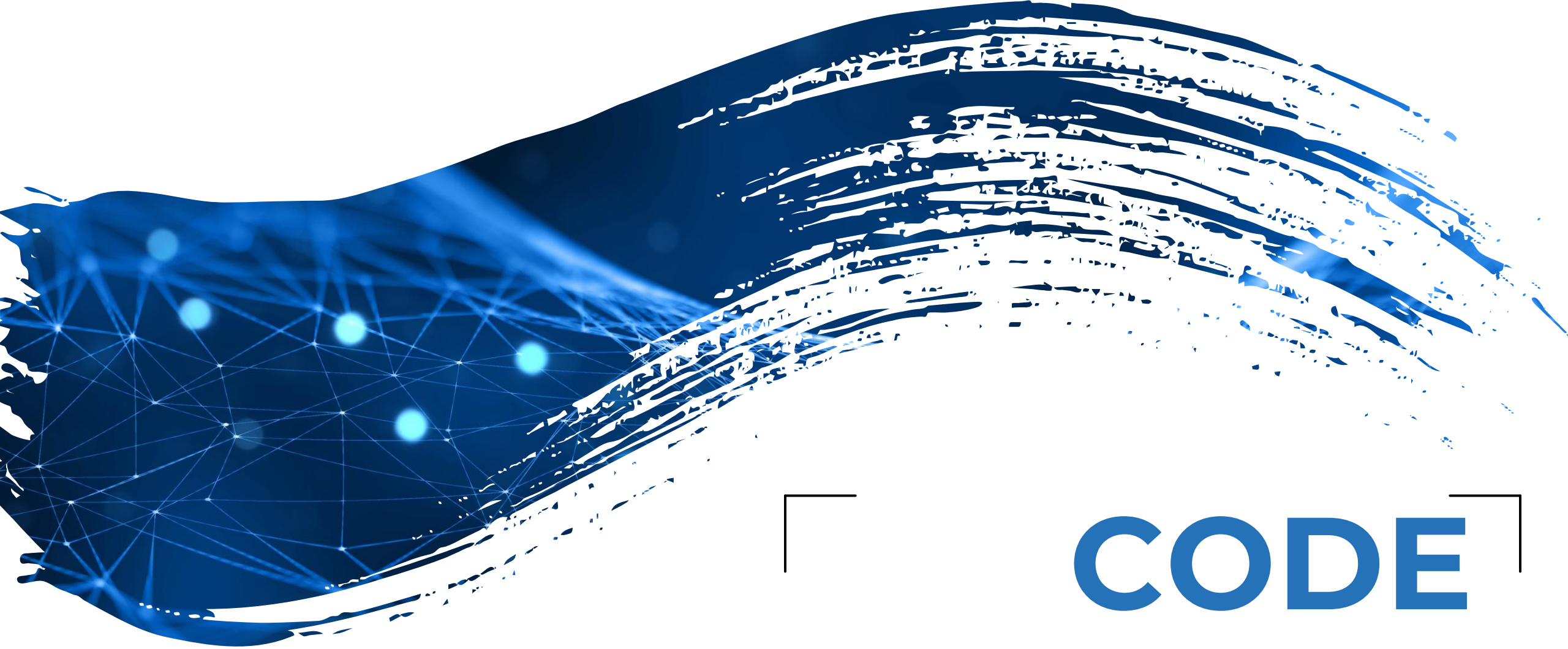
sidion

QSORT

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

Idee: Schreibe die compare-Funktion in Java

```
MemorySegment.upcallStub(MethodHandle target,
FunctionDescriptor function, Arena arena)
```

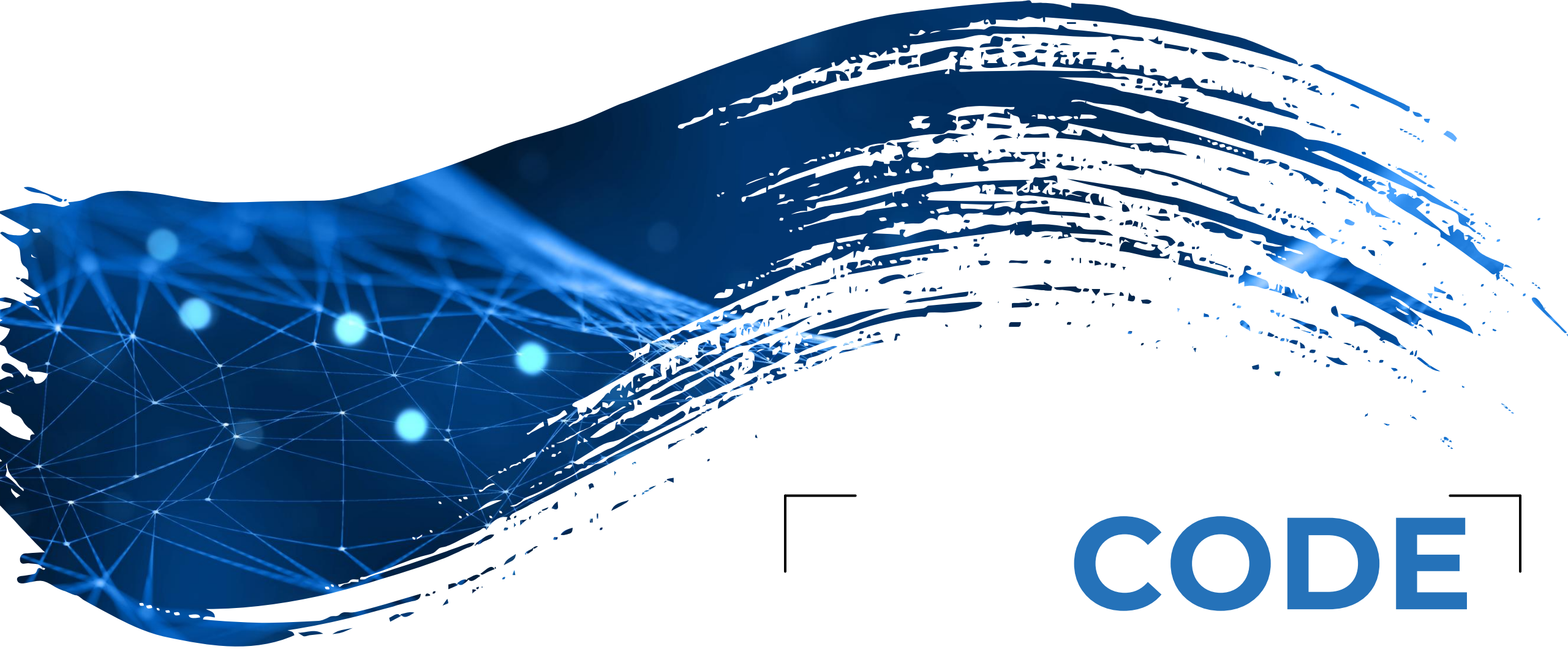


CODE DEMO

sidion

JEXTRACT

- Parst Header-Files von native Libraries
- Erstellt Java-Bindings



CODE DEMO

sidion

PROJEKT BABYLON

github.com/openjdk/babylon

sidion

JEP DRAFT 8361105

- **Summary**

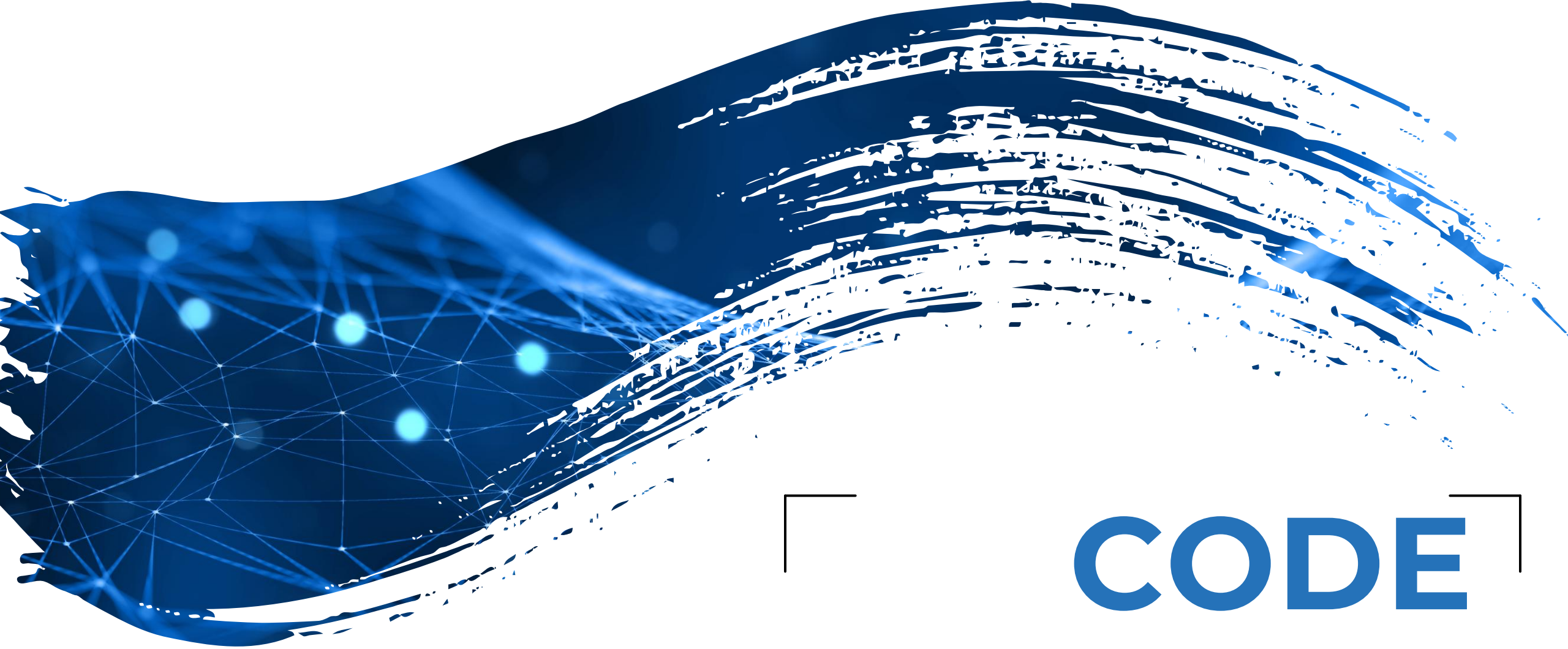
Enhance the core reflection API to model Java code, build and transform models of Java code, and access models of Java code in methods and lambda expressions. Libraries can use this enhancement to analyze Java code and extend its reach, such as executing it as code on GPUs. This is an incubating API.

STATUS QUO - JAVA REFLECTION

- Seit Java 1.1 (1997)

- Inspizieren, Analysieren und Manipulieren

Bekannt durch:
- Frameworks (Spring, Hibernate, JUnit, Dozer)
- Dependency Injection
- Serialization / Deserialization
- Dynamic method invocation
- Plugin Systems



CODE DEMO

sidion

GRENZEN

- Auf Java begrenzt
- Keine Typsicherheit
- Fehler erst zur Laufzeit
- Kein Zugriff auf Methodenkörper & Lambdas
- Erhöhter Ressourcenverbrauch (Zeit und Speicher)

BABYLON - CODE REFLECTION

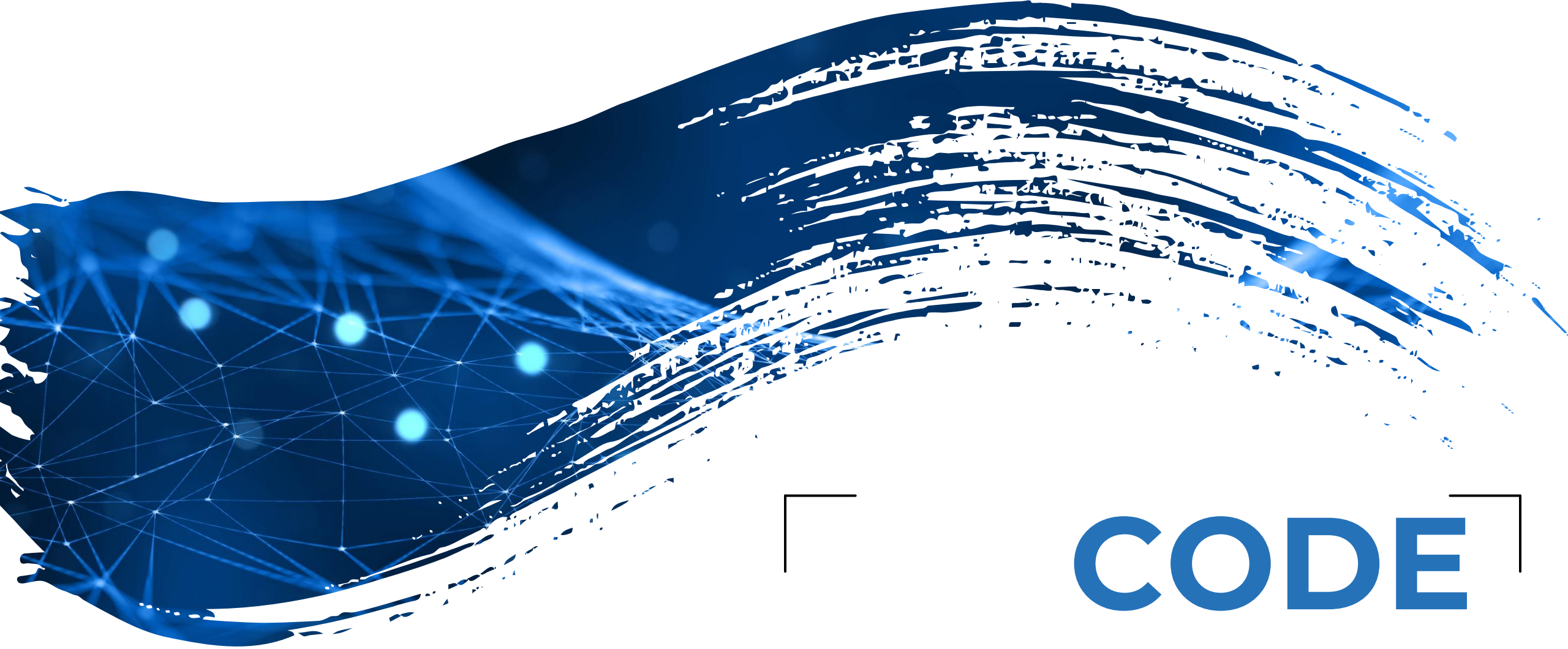
- Erweiterung von Java Reflection
- Modellierung von Java als Codemodelle
- Codemodell-APIs zum Erstellen, Analysieren und Transformieren
- Ziel: Java erweitern auf fremde Programmiermodelle
  - wie bspw. SQL
  - differenzierbare Programmierung
  - Modelle für maschinelles Lernen
  - GPUs

CODEMODELL

@Reflect
static int multiply (int a, int b) { return a * b; }

Codemodell:

```
func @loc="38:5:FirstTry.java" @"multiply" (%0 : java.type:"int", %1 : java.type:"int")java.type:"int" -> {
    %2 : Var<java.type:"int"> = var %0 @loc="38:5" @"a";
    %3 : Var<java.type:"int"> = var %1 @loc="38:5" @"b";
    %4 : java.type:"int" = var.load %2 @loc="40:16";
    %5 : java.type:"int" = var.load %3 @loc="40:20";
    %6 : java.type:"int" = mul %4 %5 @loc="40:16";
    return %6 @loc="40:9";
};
```



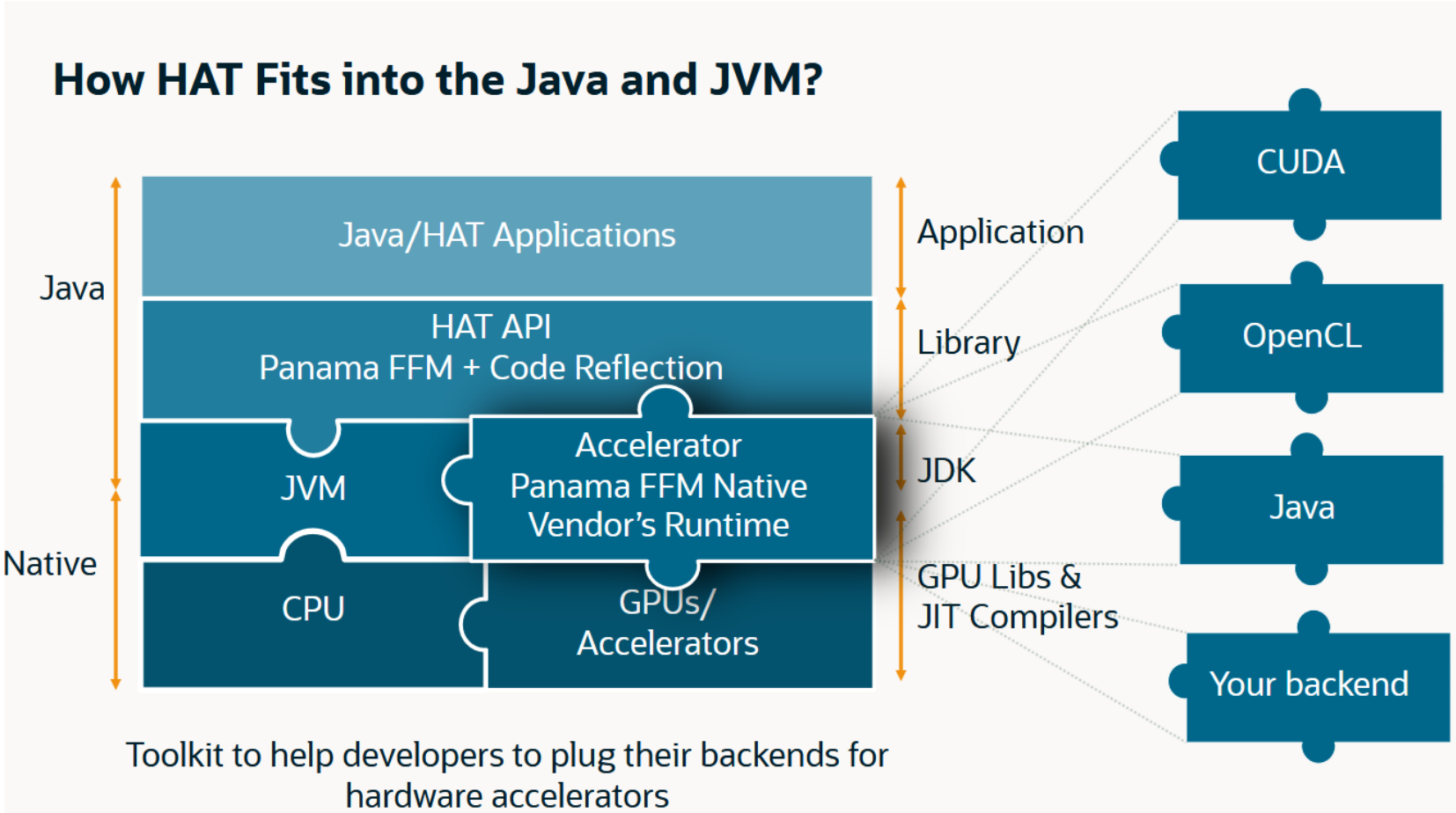
CODE DEMO

sidion

HETEROGENEOUS ACCELERATOR TOOLKIT

- Datenparallele (Java-)Anwendungen auf Hardwarebeschleunigern ausführen
- "Laufzeit- und Übersetzungsschicht von Babylon für Hardwarebeschleuniger"
- Nutzt Code-Modell, erzeugt daraus Accelerator-Kernel, verwaltet Speicher und startet die Ausführung auf dem Zielgerät.

HETEROGENEOUS ACCELERATOR TOOLKIT



https://cr.openjdk.org/~jfumero/presentations/JavaOne26-EmpoweringGPUsFromJava.pdf P.19

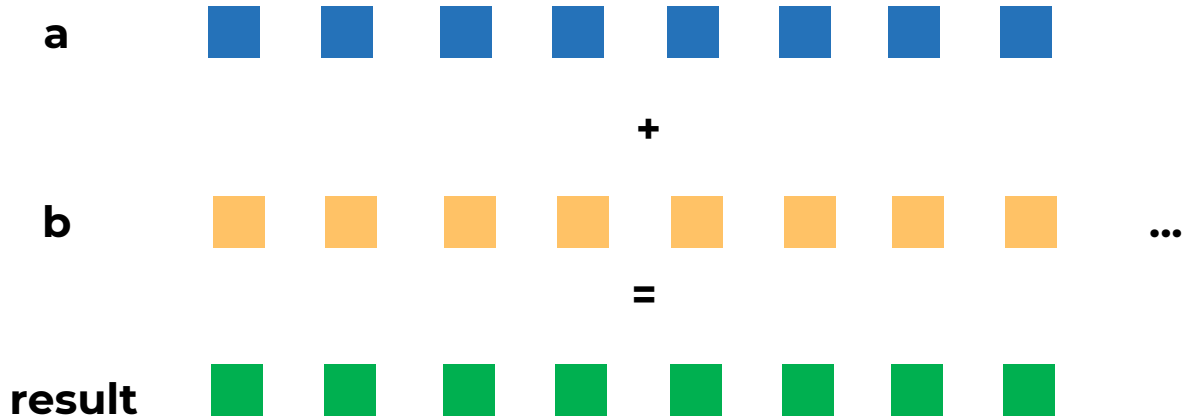
sidion

HETEROGENEOUS ACCELERATOR TOOLKIT

```
public static void vectorAdditionOnCpu(int[] a, int[] b, int[] result) {
    for (int i = 0; i < a.length; i++) {
        result[i] = a[i] + b[i];
    }
}
```

HETEROGENEOUS ACCELERATOR TOOLKIT

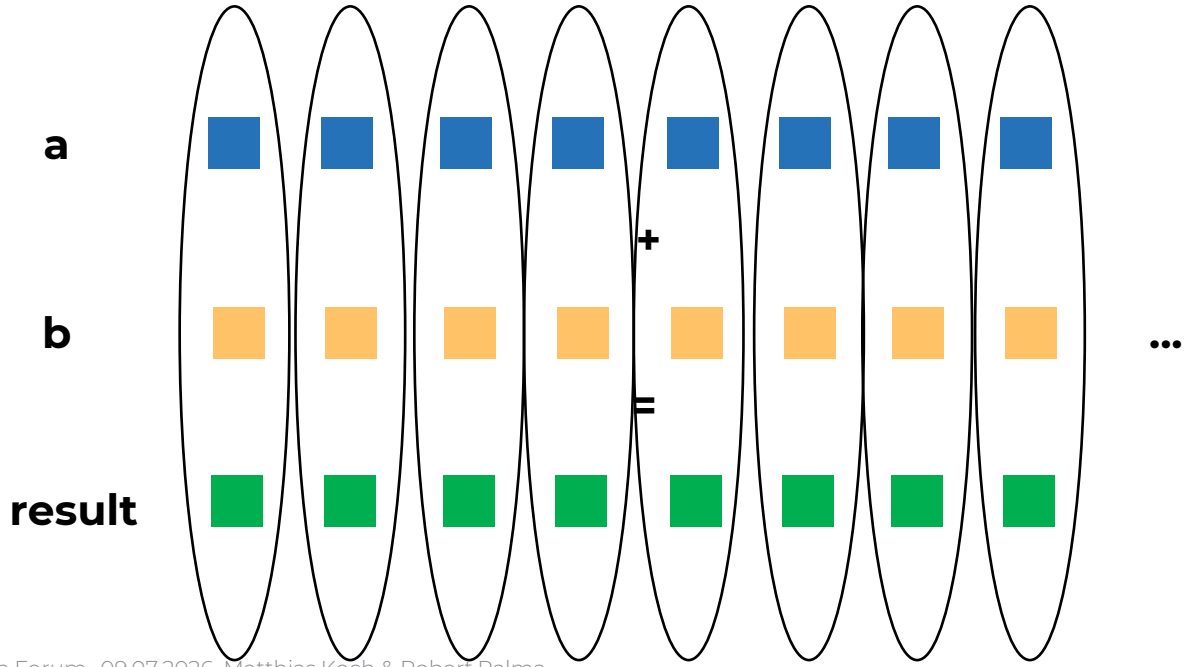
```
public static void vectorAdditionOnCpu(int[] a, int[] b, int[] result) {
    for (int i = 0; i < a.length; i++) {
        result[i] = a[i] + b[i];
    }
}
```



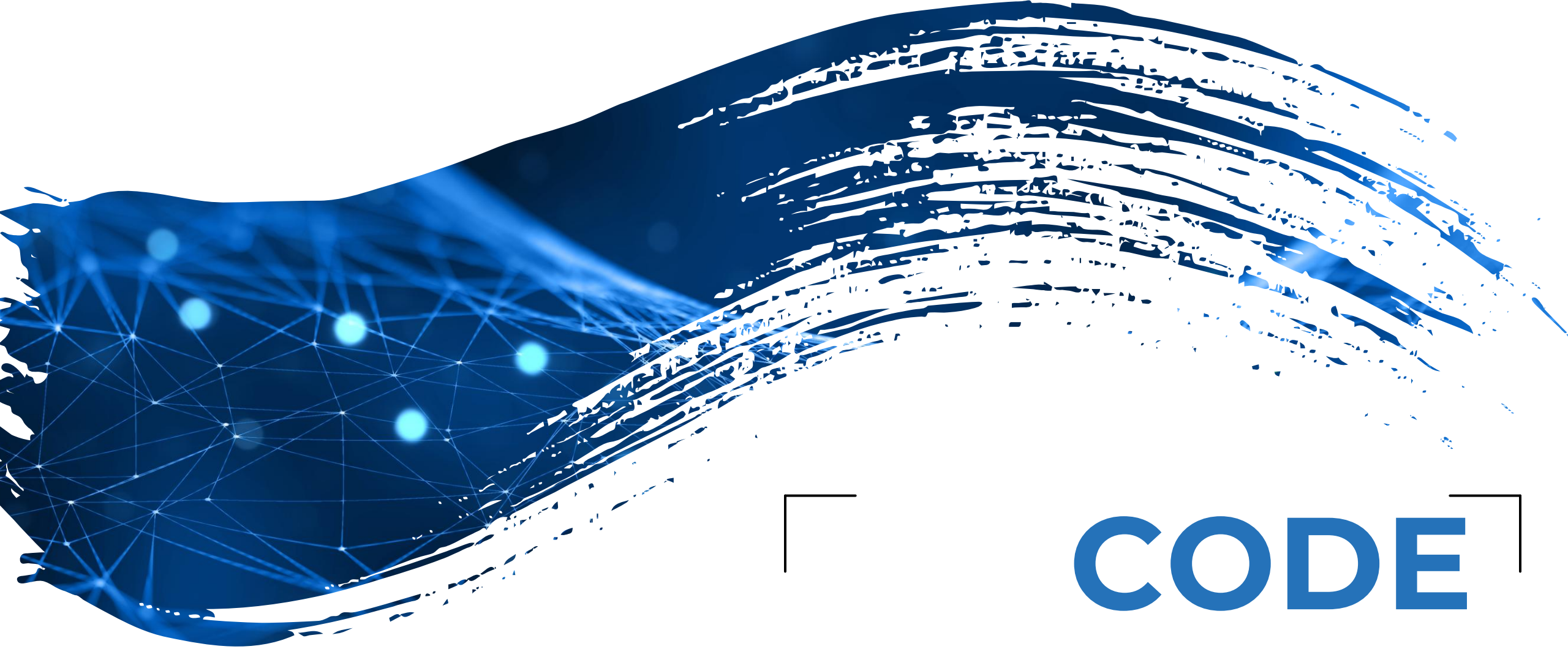
sidion

HETEROGENEOUS ACCELERATOR TOOLKIT

```
public static void vectorAdditionOnCpu(int[] a, int[] b, int[] result) {
    for (int i = 0; i < a.length; i++) {
        result[i] = a[i] + b[i];
    }
}
```



sidion



CODE DEMO

sidion

CODE DEMO ERGEBNIS

############ Prepare HAT Usage ############
Preparing Vectors with 10 elements
Preparation took: 126,87 ms
############ Run on GPU using HAT ############
GPU (HAT): 107,05 ms

############ Run on CPU ############
CPU (loop): 0,00 ms

############ Checkin Results ############
Speedup: 0,0x
Are all values equal? -> true

CODE DEMO ERGEBNIS

############ Prepare HAT Usage ############
Preparing Vectors with 200.000.000 elements
Preparation took: 24880,43 ms
############ Run on GPU using HAT ############
GPU (HAT): 69,10 ms

############ Run on CPU ############
CPU (loop): 44,77 ms

############ Checkin Results ############
Speedup: 0,6x
Are all values equal? -> true

CODE DEMO ERGEBNIS

############ Prepare HAT Usage ############

Preparing Vectors with 2.000.000.000 elements

Preparation took: 254516,62 ms

############ Run on GPU using HAT ############

GPU (HAT): 3866,96 ms

############ Run on CPU ############

CPU (loop): 7404,78 ms

############ Checkin Results ############

Speedup: 1,9x

Are all values equal? -> true

CODE DEMO ERGEBNIS

############ Prepare HAT Usage ############

Preparing Vectors with 2.100.000.000 elements

Preparation took: 237060,48 ms

############ Run on GPU using HAT ############

GPU (HAT): 4050,23 ms

############ Run on CPU ############

CPU (loop): 16949,05 ms

############ Checkin Results ############

Speedup: 4,2x

Are all values equal? -> true

ZUSAMMENFASSUNG

- Panama ausgereifter als Babylon
- Panama bietet deutliche Verbesserung für die Anbindung von native Libraries
- Aber: Trotzdem keine Echtzeit-Programmierung möglich

- Code-Reflection ist eine deutliche Verbesserung gegenüber Java-Reflection
- Babylon ermöglicht die Verwendung alternativer Laufzeitmodelle

- Babylon und Panama realisieren Erweiterung auf externe Hardware

FRAGEN

Mitdenken

Mitgestalten

Mitarbeiten

bei sidion

sidion

VIELEN DANK