Java for AI

Paul Sandoz,
Java Platform Group,
Oracle

For the times, they are a-changin’

- The environment of computing is rapidly changing
  - Some form of Agentic AI will become an essential tool in the software development toolbox
- We need to adapt (technically & culturally)
  - “Adapt or perish, now as ever, is Nature’s inexorable imperative.” (H.G. Wells, 1945)
  - “ADAPT OR DIE.” (Andrew Grove, 1983)
- It’s an exciting time, but it’s also an anxious time
  - Hard to filter out all the nonsense
  - There is still much we don’t know
- With adaptability and resilience it’s also a hopeful time…

“Change is the only constant”

Java in 2018: Change is the Only Constant Keynote by Mark Reinhold

Java has changed and adapted over 30 years

Development of Java (Oak) began in 1991 and was first released in 1995

- Object oriented programming
- Internet
- World Wide Web
- Open Source
- Mobile and Cloud Computing

Java has changed and adapted over 30 years

Development of Java (Oak) began in 1991 and was first released in 1995

- Object oriented programming
- Internet
- World Wide
- Open Source
- Mobile and

In the face of constantly-evolving programming paradigms, application areas, deployment styles, and hardware.

Java has changed and adapted over 30 years

Development of Java (Oak) began in 1991 and was first released in 1995

- Object oriented programming
- Internet
- World Wide
- Open Source
- Mobile and
- …
- Artificial Intelligence and Agentic AI

In the face of constantly-evolving programming paradigms, application areas, deployment styles, and hardware.

Java has had over 30 years of “crystallized cognition”

“a financial asset”, “crystallized human labor”, “sheer insight density”

Java has had over 30 years of “crystallized cognition”

“a financial asset”, “crystallized human labor”, “sheer insight density”

- The source code, the specifications, the ecosystem
  - Java continues to behave the same way with respect to its specifications
  - *Intellectual* compatibility
- Building your own Java platform from scratch is a daunting research project
  - The JDK successfully solves many genuinely hard problems with elegance
  - Same applies to many Java frameworks/libraries, git, Kubernetes, databases, etc.

Java has had over 30 years of “crystallized cognition”

“a financial asset”, “crystallized human labor”, “sheer insight density”

- It would be crazy to even try and re-synthesize the Java platform
  - There are better ways to spend money on tokens
  - The OpenJDK source is a git clone away at a minuscule fraction of the cost
    - Similarly so for JDKs, even with support
- LLMs prefer to work with tools that compress insights
  - Java is full of compressed insights, and therefore your Java code is too
  - There is only so much context an LLM can handle until it becomes polluted
  - Like humans (no surprise there)

Java has had over 30 years of “crystallized cognition”

“a financial asset”, “crystallized human labor”, “sheer insight density”

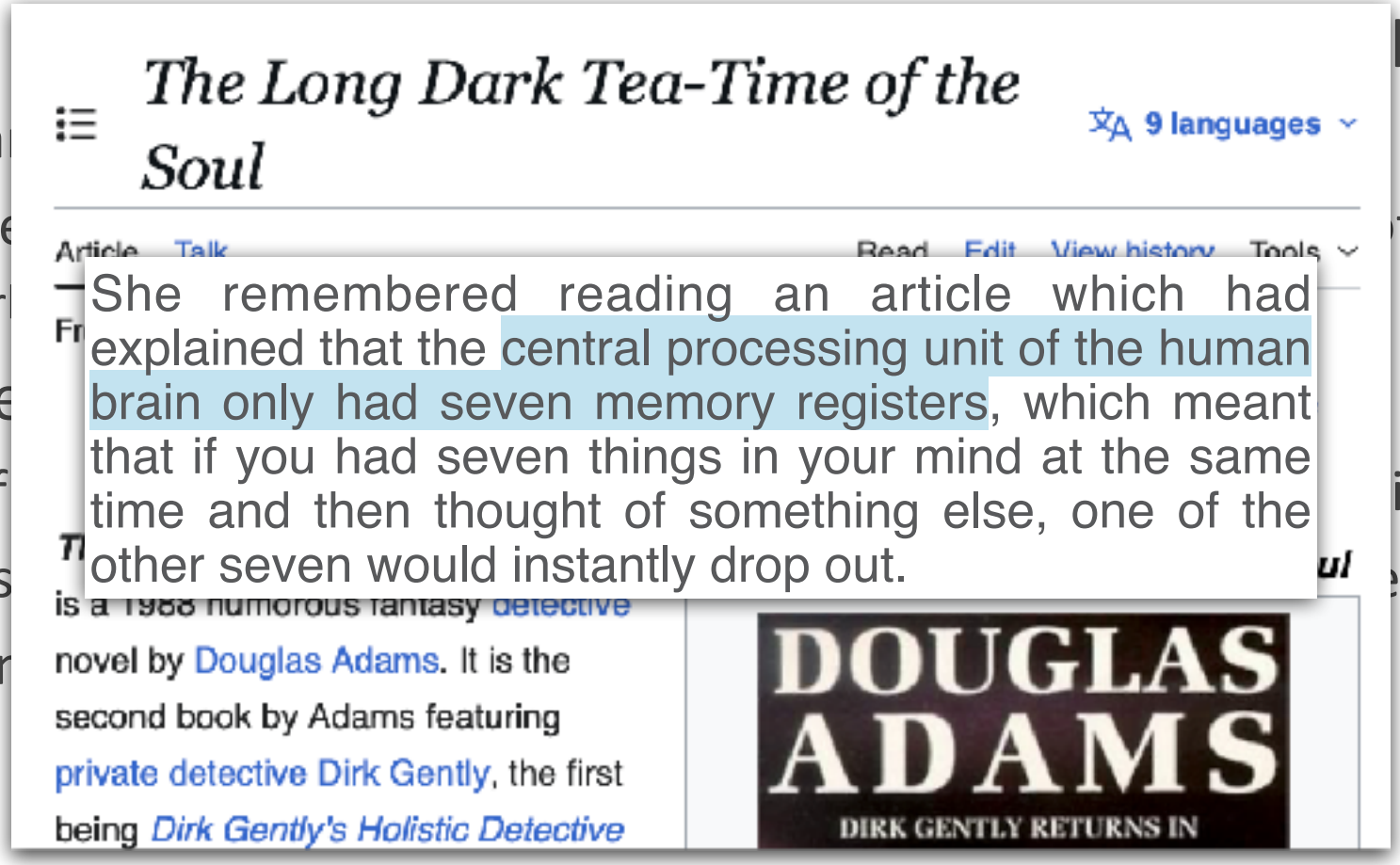
- It would
  - There a
  - The Ope
    - Similar
- LLMs pre
  - Java is f
  - There is
  - Like hur

latform

f the cost

is too

s polluted



Java has had over 30 years of “crystallized cognition”
“a financial asset”, “crystallized human labor”, “sheer insight density”

- It would be crazy to even try and re-synthesize the Java platform
  - You have better ways to spend you time and money… on tokens
  - The OpenJDK source is a git clone away at a minuscule fraction of the cost
- LLMs prefer to work with tools that compress insights
  - Java is full of compressed insights, and therefore your Java code is too
  - There is only so much context an LLM can handle until it becomes polluted
  - Like humans (no surprise there)
- Agents orchestrate the interaction between LLMs and *tools*
  - So they work best using good tools, especially those with highly compressed insight
  - Like the Java platform, and Java code

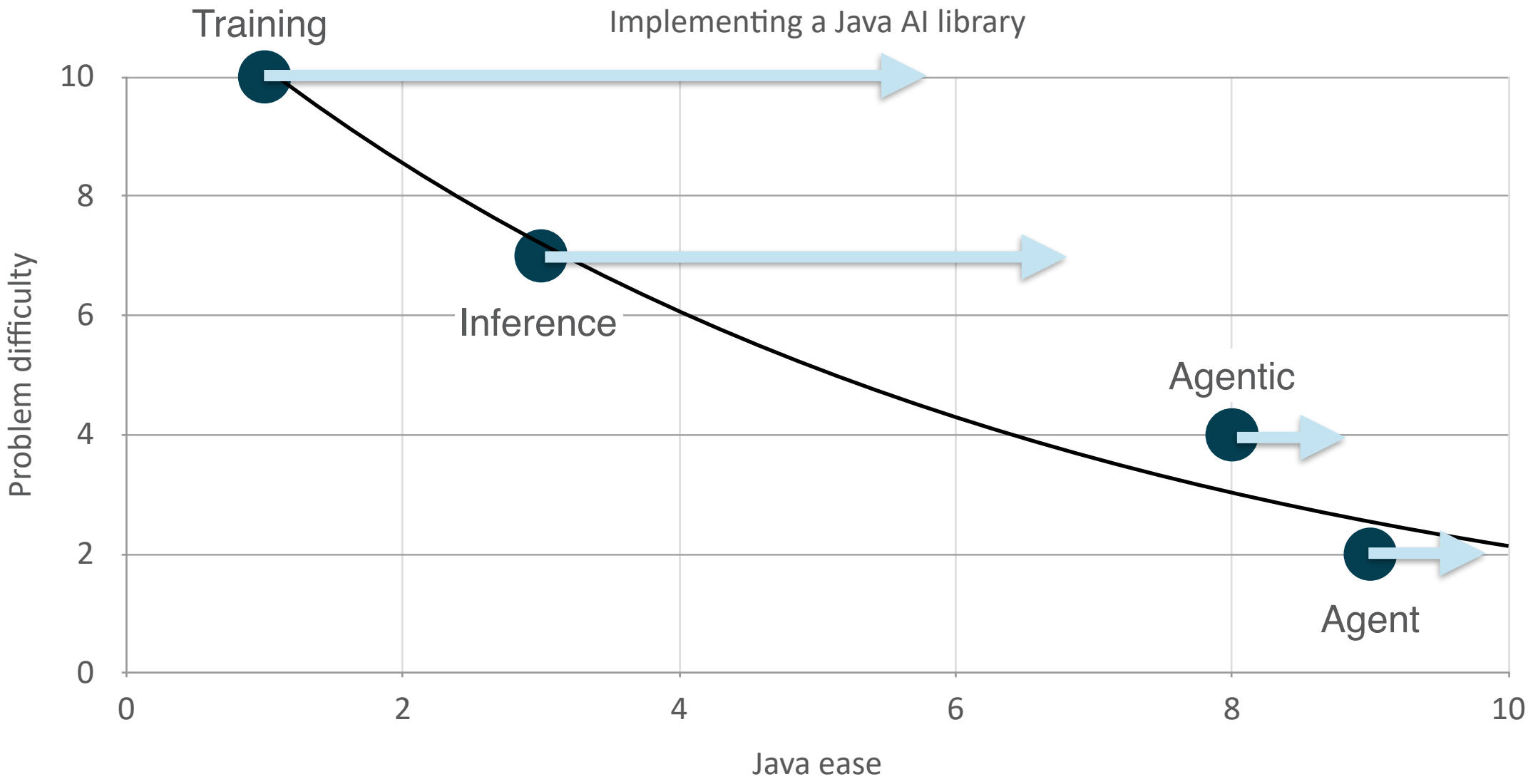
In compressed summary

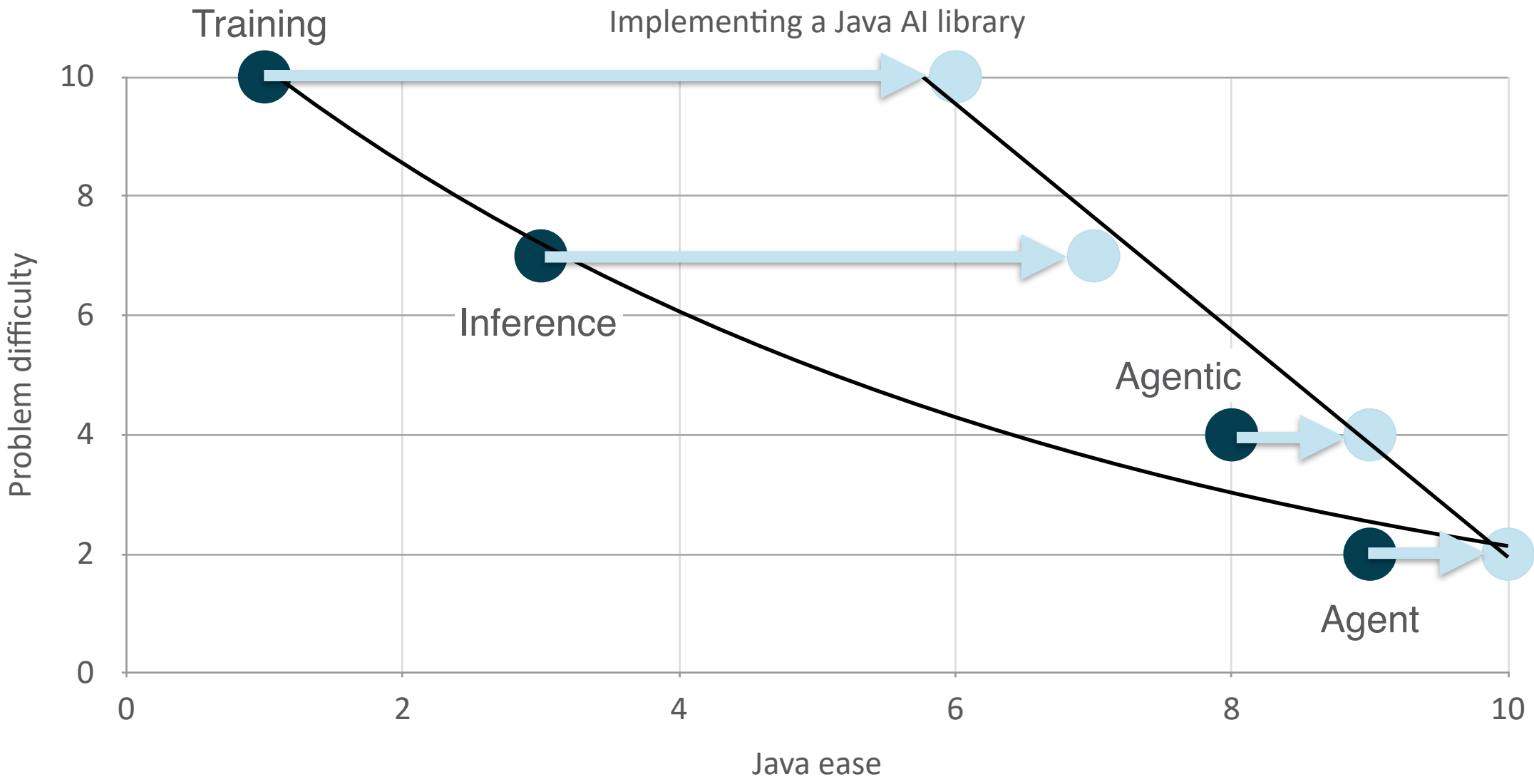
Use three of your seven memory registers (or further compress)

- Change is happening and we need to adapt
- We, the stewards of the Java platform and the Java community, are used to change and have successfully adapted over the past 30 years
- The Java platform and its ecosystem has over 30 years of compressed insight that is rocket fuel for LLMs

AI Layers

- Training
  - Consuming enormous quantities of data and compute
- Inference
  - The product of training
  - In aggregate consuming more compute than training
- Agent
  - The composition of inference
  - Orchestrating LLM, tools, and information sources
- Agentic
  - The composition of agents, with dynamic context
  - Orchestrating agents (workflow) to autonomously achieve a desired outcome





Agentic (& Agent)

Java for building Agentic AI libraries and applications

- Java is already excellent for this
  - The problems are very similar to other domains where Java excels
  - Reusing existing building blocks and patterns (compressed insight)
  - There is nothing magical about this - it requires good experience and skill to engineer a well-architected solution
- Each Java release makes this better
  - Adding and compressing further insights
  - And we can learn how to improve Java
    - e.g., better string manipulation? better process execution and management?

Agentic (& Agent)

Agentic AI for building Java libraries and applications

- Qualitative evidence
  - 30 years of compressed insights and compatibility
  - Java should be great a fit (that's my intuition and experience so far)
- Quantifiable evidence
  - In short supply — it's all evolving so rapidly
  - We don't currently know what makes an LLM prefer one language platform over another
    - Reduces to economics? i.e., cost-per-token
  - Agents will likely prefer efficient compilers and runtimes
    - Fast compile & test cycles

Agentic Java challenge

- Build an Agentic AI assistant in Java
  - Using an existing coding assistant
- Call it *JavaPaw*
  - A friendly, safer, more deployable, scalable, and maintainable solution
  - Deployed instances would be more cattle-like than pet-like
- Use JavaPaw to help build Java libraries and applications
  - And use it to further develop JavaPaw
  - How do we know if the AI is drifting away from our instructions?

Training and Inference
The pain becomes noticeable

- Add the right (general purpose) building blocks to the JDK
  - Rather than direct support
  - Make it less painful to build frameworks and libraries
- OpenJDK projects are factories that produce the building blocks
  - Babylon, Detroit, Panama, Valhalla

Inference

- Panama — Foreign Function & Memory (FFM) API
  - Deploy and execute ONNX models using native ONNX runtime
- Panama — Vector API
  - Implement model runtimes on the CPU
- Babylon - HAT
  - Implement model runtimes on the GPU
- Valhalla — Numerics
  - Express more numeric types arrays of which are flat in memory
- Detroit
  - Deploy and execute PyTorch models

Training… motivation for Project Babylon

It's hard for Java developers to…

- … automatically differentiate mathematical code
- … develop and run on GPUs
- … develop and run machine learning models

In general… motivation for Project Babylon

It's hard for Java developers to…

- … automatically differentiate mathematical code
- … develop and run on GPUs
- … develop and run machine learning models

- … express type-safe SQL statements and run on databases
- … develop eBPF programs and run on Linux kernels
- … easily transform Java code

Inferior choices

- There is a mismatch between what we want the Java developer to code and what they *have* to code. The Java platform currently provides inferior choices:
  - Embed non-Java code in the text blocks of Java code
  - Write tedious Java code that builds up data structures to model Java code or non-Java code
  - Use unsafe, unsupported, or unsuitable mechanisms to inspect Java code
- The prospect of writing ordinary *portable* Java code that is type safe, testable, able to call methods, and able to *represent* GPU code is extremely attractive
  - Economies of scale, as Java developers can implicitly become CUDA developers
  - Protected investment in Java, as competition in multi-vendor hardware heats up

What do those painful use cases have in common?

- The need to *translate* Java code to code of a *foreign* programming model
  - Programming models other than *the* Java programming model, as defined by the Java specifications
- e.g., translation of Java code in the domain of a GPU programming
  - Not all Java code is representable as GPU code, translation is partial (↛)
  - The translation knows about the GPU programming model, not the Java programming model
- Java code ↛ Java code that runs on the JVM
  - Java code ↛ Differentiated Java code, and run on the JVM
- Java code ↛ *foreign* code that runs on a *foreign* runtime
  - Java code ↛ CUDA, and run on GPUs
  - Java code ↛ ONNX models, and run on the ONNX runtime

Project Babylon
The analgesic

Enables users to write code
for foreign programming models
in Java code; and

enables tools to automatically
validate and transform it
to execute on foreign runtimes

Project Babylon
The analgesic

1. Enhance the Java platform with the *right* feature so
2. Java developers can *easily* build GPU libraries that
3. enable other Java developers to
4. develop *portable* Java code and run it on GPUs

Project Babylon: part 1, the right platform feature

1. Enhance Java's dynamic capabilities with *code reflection*
   - A standard way to *access* the Java code of methods and lambda expressions at run time (and eventually at compile time)
   - A standard way to symbolically *represent* Java code, a Java *code model*, suitable for translation e.g., to *foreign* code of *foreign* programming models

Project Babylon: part 1, the right platform feature

1. Enhance Java's dynamic capabilities with *code reflection*
   - A standard way to *access* the Java code of methods and lambda expressions at run time (and eventually at compile time)
   - A standard way to symbolically *represent* Java code, a Java *code model*, suitable for translation e.g., to *foreign* code of *foreign* programming models

Project Panama & Project Babylon

Greater than the sum of the parts

+

Project Panama is to *foreign libraries*

as

Project Babylon is to *foreign programming models*

Project Panama + Project Babylon
Greater than the sum of the parts

- The two combined enable Java developers to build powerful libraries
  - Use Panama's FFM API and jextract to call foreign compilers & runtimes
  - Use Babylon's code reflection to translate Java code to foreign code, input that code to a foreign compiler, the result of which is then executed by a foreign runtime
- Using features of both projects we have built Java libraries for
  - Machine learning programming with ONNX
  - GPU programming

Project Babylon: part 2, a Java GPU library

1. Enhance Java's dynamic capabilities with *code reflection*
   - A standard way to *access* the Java code of methods and lambda expressions at run time and compile time
   - A standard way to symbolically *represent* Java code, a Java *code model*, suitable for translation to *foreign* code of *foreign* programming models
2. Heterogeneous Accelerator Toolkit (HAT)
   - Develop portable Java code, debug on the CPU, run on the GPU
   - Uses code reflection to translate Java code to foreign GPU code
   - Uses Panama's Foreign Function & Memory to call foreign GPU compilers and GPU runtimes

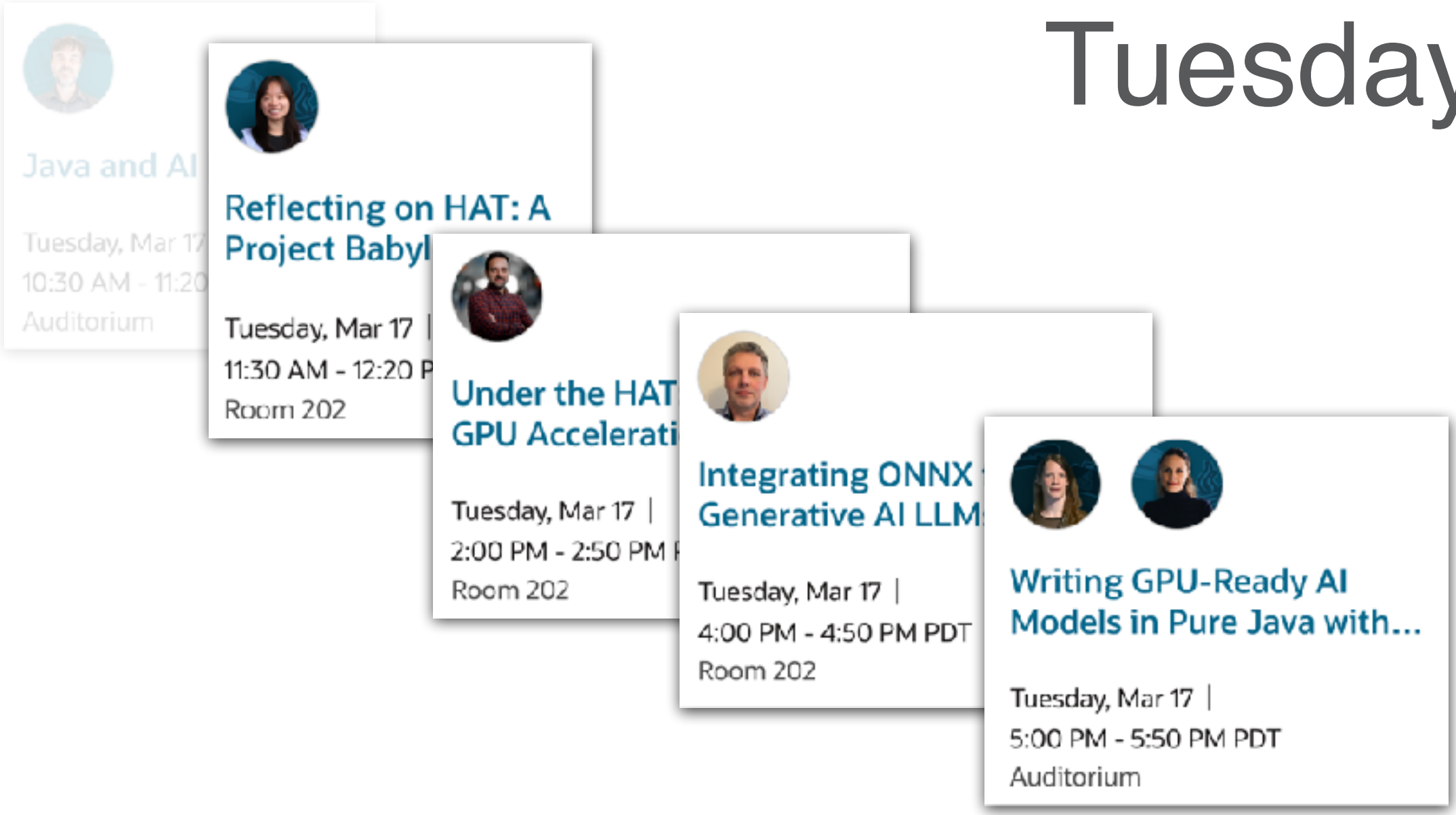
JEP draft: Code reflection (Incubator)

Owner	Paul Sandoz
Type	Feature
Scope	JDK
Status	Submitted
Component	core-libs
Discussion	babylon dash dev at openjdk dot org
Effort	L
Duration	L
Reviewed by	Adam Sotona, Gary Frost, Juan Fumero, Maurizio Cimadamore
Created	2025/06/30 19:54
Updated	2026/03/03 15:33
Issue	8361105

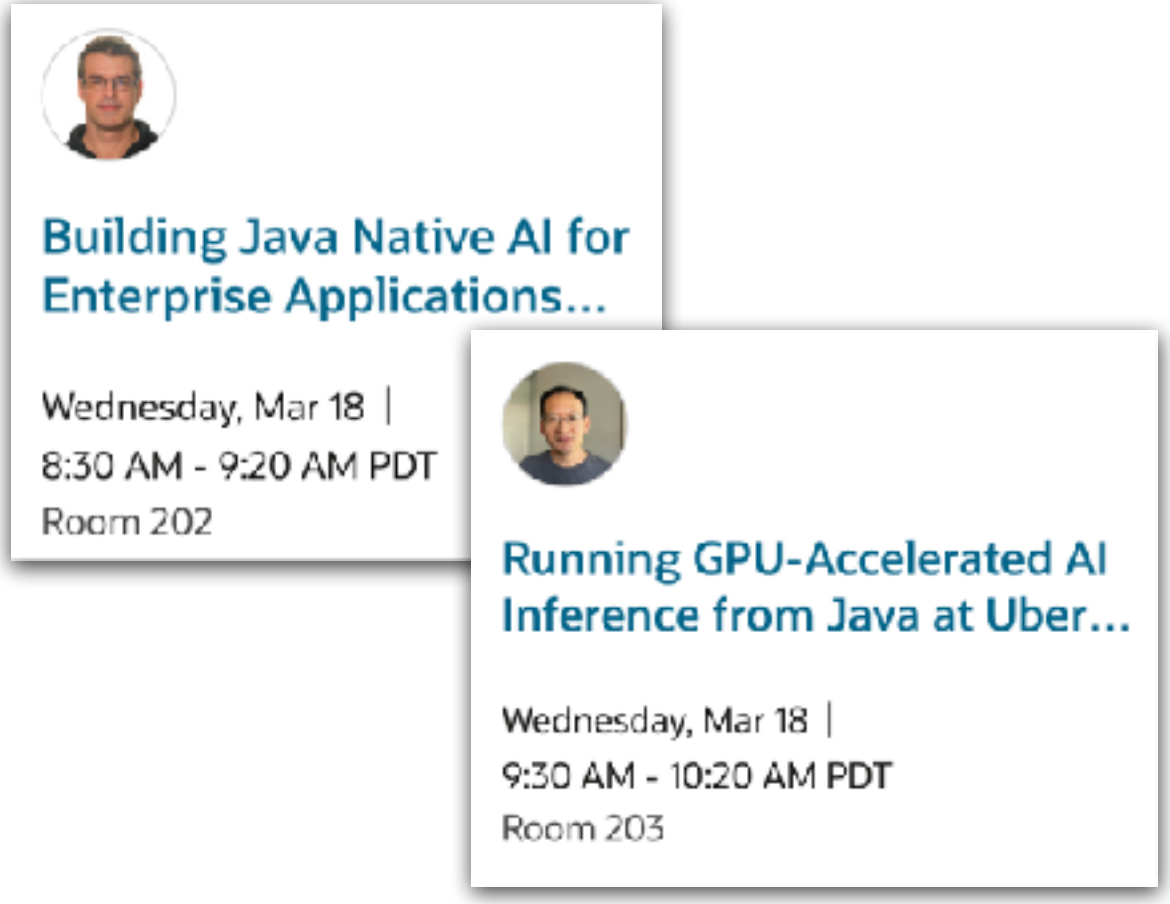
Summary

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.

Tuesday



Wednesday



Demos

- Java shader toy
- Using code reflection to transform Java code