WHY ENTERPRISE MCP RUNS ON JAVA

MCP Dev Summit Korea

Kevin Dubois
Sr Principal Dev Advocate

Daniel Oh
Sr Principal Dev Advocate

IBM

Agentic AI Foundation

JAVA? BORING & OLD!

And yet...

What is the best programming language for performant, robust and modern enterprise development with a great ecosystem? Pick one

If I have to pick exactly one, it's **Java**.

Why Java over every other option?

- **Performance:** Modern JVM performance is excellent. For most enterprise workloads, Java is fast enough that database, network, and architecture decisions matter far more than language overhead.
- **Robustness:** Strong static typing, mature tooling, extensive testing frameworks, and decades of battle-tested production use.
- **Enterprise maturity:** Large organizations, banks, insurers, telecoms, governments, and major tech companies run enormous Java systems.
- **Modern language features:** Records, pattern matching, virtual threads, improved concurrency, and continuous language evolution have made modern Java much more pleasant than its reputation suggests.
- **Ecosystem:** The combination of the JVM, Maven/Gradle, Spring, Jakarta EE, observability tools, cloud-native frameworks, and a huge talent pool is difficult to match.
- **Longevity:** A Java codebase written today is very likely to remain maintainable, supported, and recruitable 10–20 years from now.

FAST *AND* ENERGY EFFICIENT

Energy Efficiency across Programming Languages

How Does Energy, Time, and Memory Relate?

Rui Pereira
HASLab/INESC TEC
Universidade do Minho, Portugal
ruipereira@di.uminho.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.l.couto@inesctec.pt

Francisco Ribeiro, Rui Rua
HASLab/INESC TEC
Universidade do Minho, Portugal
fribeiro@di.uminho.pt
rrua@di.uminho.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacome@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISUC
Universidade de Coimbra, Portugal
jpf@dei.uc.pt

João Saraiva
HASLab/INESC TEC
Universidade do Minho, Portugal
saraiva@di.uminho.pt

greenlab.di.uminho.pt/wp-content/uploads/2017/09/paperSLE.pdf

Table 4. Normalized global results for Energy, Time, and Memory

Total

	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

FAST *AND* ENERGY EFFICIENT

Energy Efficiency across Programming Languages

How Does Energy, Time, and Memory Relate?

Rui Pereira
HASLab/INESC TEC
Universidade do Minho, Portugal
ruipereira@di.uminho.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.l.couto@inesctec.pt

Francisco Ribeiro, Rui Rua
HASLab/INESC TEC
Universidade do Minho, Portugal
fribeiro@di.uminho.pt
rrua@di.uminho.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacome@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISUC
Universidade de Coimbra, Portugal
jpf@dei.uc.pt

João Saraiva
HASLab/INESC TEC
Universidade do Minho, Portugal
saraiva@di.uminho.pt

greenlab.di.uminho.pt/wp-content/uploads/2017/09/paperSLE.pdf

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) Ada	1.70			(c) C	1.17
(v) Java	1.98			(c) Fortran	1.24
(c) Pascal	2.14			(c) C++	1.34
(c) Chapel	2.18	(c) Go	2.83	(c) Ada	1.47
(v) Lisp	2.27	(c) Pascal	3.02	(c) Rust	1.54
(c) Ocaml	2.40	(c) Ocaml	3.09	(v) Lisp	1.92
(c) Fortran	2.52	(v) C#	3.14	(c) Haskell	2.45
(c) Swift	2.79	(v) Lisp	3.40	(i) PHP	2.57
(c) Haskell	3.10	(c) Haskell	3.55	(c) Swift	2.71
(v) C#	3.14	(c) Swift	4.20	(i) Python	2.80
(c) Go	3.23	(c) Fortran	4.20	(c) Ocaml	2.82
(i) Dart	3.83	(v) F#	6.30	(v) C#	2.85
(v) F#	4.13	(i) JavaScript	6.52	(i) Hack	3.34
(i) JavaScript	4.45	(i) Dart	6.67	(v) Racket	3.52
(v) Racket	7.91	(v) Racket	11.27	(i) Ruby	3.97
(i) TypeScript	21.50	(i) Hack	26.99	(c) Chapel	4.00
(i) Hack	24.02	(i) PHP	27.64	(v) F#	4.25
(i) PHP	29.30	(v) Erlang	36.71	(i) JavaScript	4.59
(i) Ruby	69.91			(i) TypeScript	4.69
(i) Python	75.88			(v) Java	6.01
(i) Perl	79.58	(i) Lua	82.91	(i) Perl	6.62
				(i) Lua	6.72
				(v) Erlang	7.20
				(i) Dart	8.64
				(i) Jruby	19.84

BUT JAVA IS SO

VERBOSE!

IS IT THOUGH?

```
void main() { println("Hello, World!"); }
```

Java 23+ — no class, no public static void, no System.out

THE AI LANDSCAPE

Most MCP examples are written in Python or TypeScript

MOST MCP EXAMPLES TODAY

Great for prototypes...

BUT ENTERPRISES DON'T SHIP PROTOTYPES

They ship:

- **Type-safe** services
- **Observable** systems
- **Resilient** applications
- **Secure** endpoints

Exactly what Java excels at.

QUARKUS + LANGCHAIN4J

Enterprise-grade AI in Java

HOW IT FITS TOGETHER

Quarkus LangChain4j

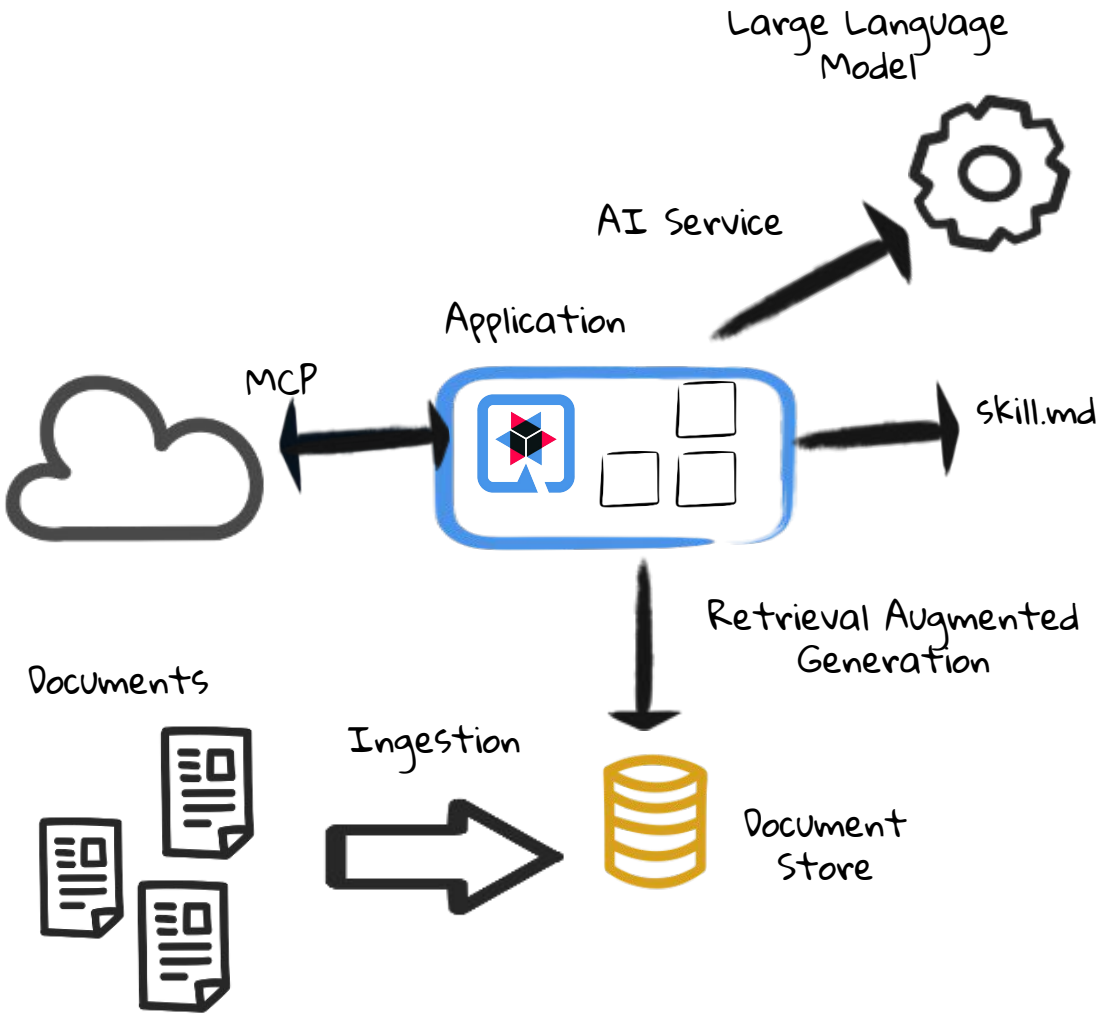
uses and extends the LangChain4j library with enterprise features.

Quarkus Application LangChain4j

AI Service LLM

QUARKUS LANGCHAIN4J - AI FOR PROD

- Declarative AI clients
- RAG building blocks
- Observability (e.g. OpenTelemetry, Prometheus)
- Auditing & logging
- Resilience (retry, fallback, circuit breaker)
- Mockable for testing
- MCP Agent, Dev Services, DevUI
- CDI Integration



BOOTSTRAPPING

Add one dependency. Pick your model provider:

Anthropic **Ollama** **Gemini**

WatsonX **Jlama** **Mistral** **... etc.**

```
<dependency>
    <groupId>
        io.quarkiverse.langchain4j
    </groupId>
    <artifactId>
        quarkus-langchain4j-openai
    </artifactId>
</dependency>
```

DEFINE AN AI SERVICE

> An interface + a few annotations = a fully functional AI service with dependency injection

MyAIService.java

```java
@RegisterAiService
interface Assistant {
    String chat(String message);
}
```

MyResource.java

```java
@Inject
private final Assistant assistant;
```

application.properties

```
quarkus.langchain4j.openai.api-key=sk-...
```

That's it. No boilerplate, no builders, no manual HTTP client setup.

MCP IN JAVA

As natural as creating REST servers and clients

Spec 2025-11-25 | Stateless 2026-07-28 | quarkus-mcp-server 2.0

3 KEY REASONS FOR MCP ON JAVA

- Data Proximity Moat vs. Python and TypeScript
- Official SDK Advantage vs. C# and Go
- True Parallel Execution vs. Python's GIL

MCP Java SDK

- Java Dependencies and BOM
- Java MCP Client
- Java MCP Server

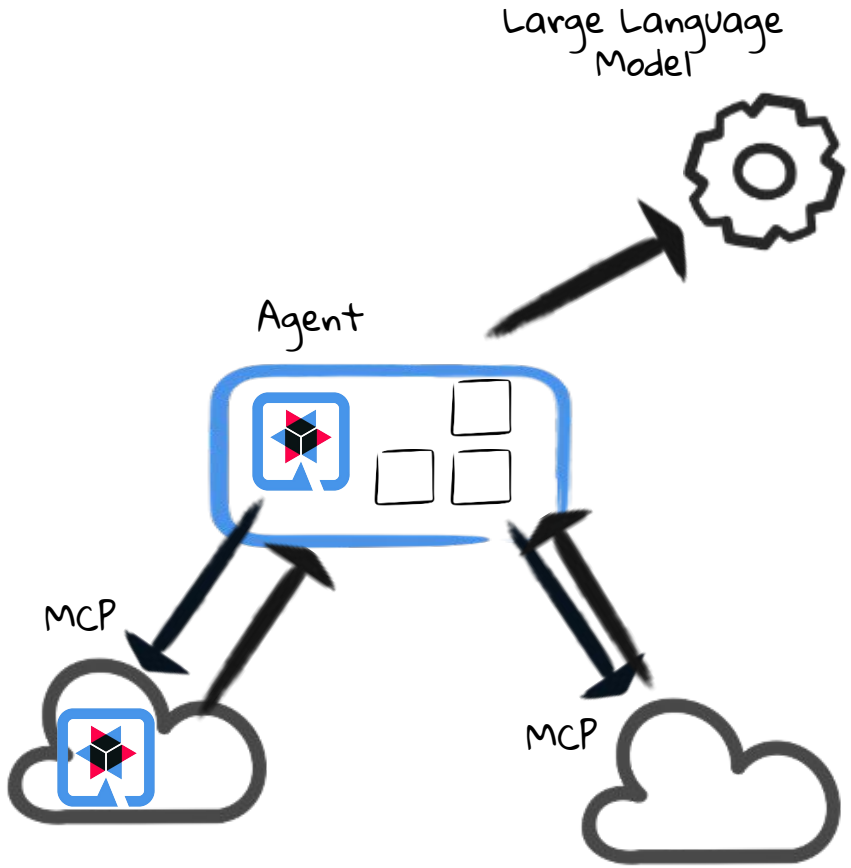
https://github.com/modelcontextprotocol/java-sdk

Extends MCP Java SDK

- Quarkus LangChain4j Integration
- Spring AI MCP

TOOL CALLING & MCP

- Mix **business code** with the model's probabilistic creativity
- Delegate to local or **external services**
- **Both** MCP Clients & Servers *can* be written in Java/Quarkus



MCP CLIENT

Connect to any MCP server with two lines of config

```
quarkus.langchain4j.mcp.weatherstation.transport-type=streamable-http
quarkus.langchain4j.mcp.weatherstation.url=http://localhost:9090/mcp
```

```
@RegisterAiService
public interface HomeBot {
    @SystemMessage("You have access to a weather station device")
    @McpToolBox("weatherstation")
    String houseFireStatus(String subject);
}
```

MCP SERVER — TOOLS

Annotate methods, describe tools — just like REST endpoints

```
@Tool(description = "Get weather data.",
  annotations = @Tool.Annotations(
      title = "Read weather station",
      readOnlyHint = true,
      destructiveHint = false))
String getCurrentWeather(
    @ToolArg(description = "Device ID")
    String deviceId,
    @ToolArg(description = "Sensor types",
            defaultValue = "all")
    String sensorTypes
) {
    return station.read(deviceId).toJson();
}
```

Add one dependency:

```
<dependency>
        <groupId>io.quarkiverse.mcp</groupId>
        <artifactId>
            quarkus-mcp-server-http
        </artifactId>
</dependency>
```

- No registration, no config file
- Input JSON schema derived from the signature
- defaultValue makes an argument optional
- Behavioral hints (readOnly, destructive, idempotent, openWorld) tell clients when to ask the human first

WHERE THE SPEED COMES FROM

BUILD TIME, NOT STARTUP TIME

- Tools, prompts and resources are **discovered and validated at build** — a broken reference is a compile error, not a 3am pager
- Capabilities and routing tables are pre-computed
- **Zero reflection** at runtime → ~30 MB, millisecond startup, native-image friendly

PICK YOUR THREADING MODEL

Inferred from the signature, overridable per method:

`Uni / CompletionStage`

`@RunOnVirtualThread`

`@Blocking`

> Vert.x and Mutiny underneath. Write blocking JDBC in a tool method and still serve thousands of concurrent sessions.

THE INNER LOOP

- **Live reload.** Edit a tool, save, call it again. No restart
- **Dev UI** at /q/dev-ui — every tool with its schema, plus a generated form to call it. Which also tells you whether your schema is sane
- **Traffic logging** — every JSON-RPC message, pretty-printed. Dev only: it's slow, and it logs whatever your users typed
- **MCP Inspector** for what the Dev UI can't reach, STDIO included
- **McpTrafficListener** — a CDI bean called on every message either way, whatever the logging config says. Your audit hook

```
quarkus.mcp.server.traffic-logging.enabled
=true
quarkus.mcp.server.traffic-logging.text-limit
=1000
```

> Debugging an MCP server without seeing the wire traffic is guesswork. This is the single most useful property in the extension.

THE INNER LOOP

- **Live reload.** Edit a tool, save, call it again. No restart
- **Dev UI** at /q/dev-ui — every tool with its schema, plus a generated form to call it. Which also tells you whether your schema is sane
- **Traffic logging** — every JSON-RPC message, pretty-printed. Dev only: it's slow, and it logs whatever your users typed
- **MCP Inspector** for what the Dev UI can't reach, STDIO included
- **McpTrafficListener** — a CDI bean called on every message either way, whatever the logging config says. Your audit hook

```
quarkus.mcp.server.traffic-logging.enabled
=true
quarkus.mcp.server.traffic-logging.text-limit
=1000
```

> Debugging an MCP server without seeing the wire traffic is guesswork. This is the single most useful property in the extension.

MCP FOR DEVELOPMENT

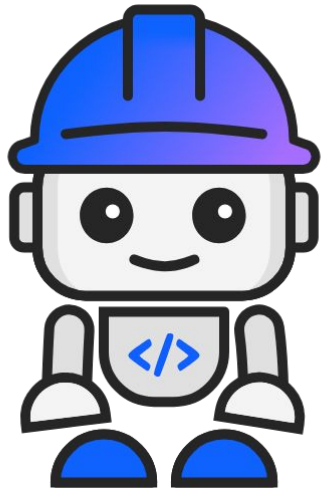
Your code assistant, supercharged

DEV MCP

Quarkus Dev Mode exposes live runtime intelligence to your code assistant via MCP

- Structured **exceptions** from the running app
- Dynamically discovered **tools**
- Extension-specific **coding patterns**
- Version-aware **documentation search**

> Your agent doesn't just write code — it gets **real-time feedback** from the running application.



Works with Cursor, Claude, Goose, IBM Bob, etc.

QUARKUS AGENT

An MCP-based agent built into Quarkus that exposes:

- Live runtime intelligence to your IDE
- RAG over Quarkus documentation
- Crash recovery and error context
- Extension-specific patterns

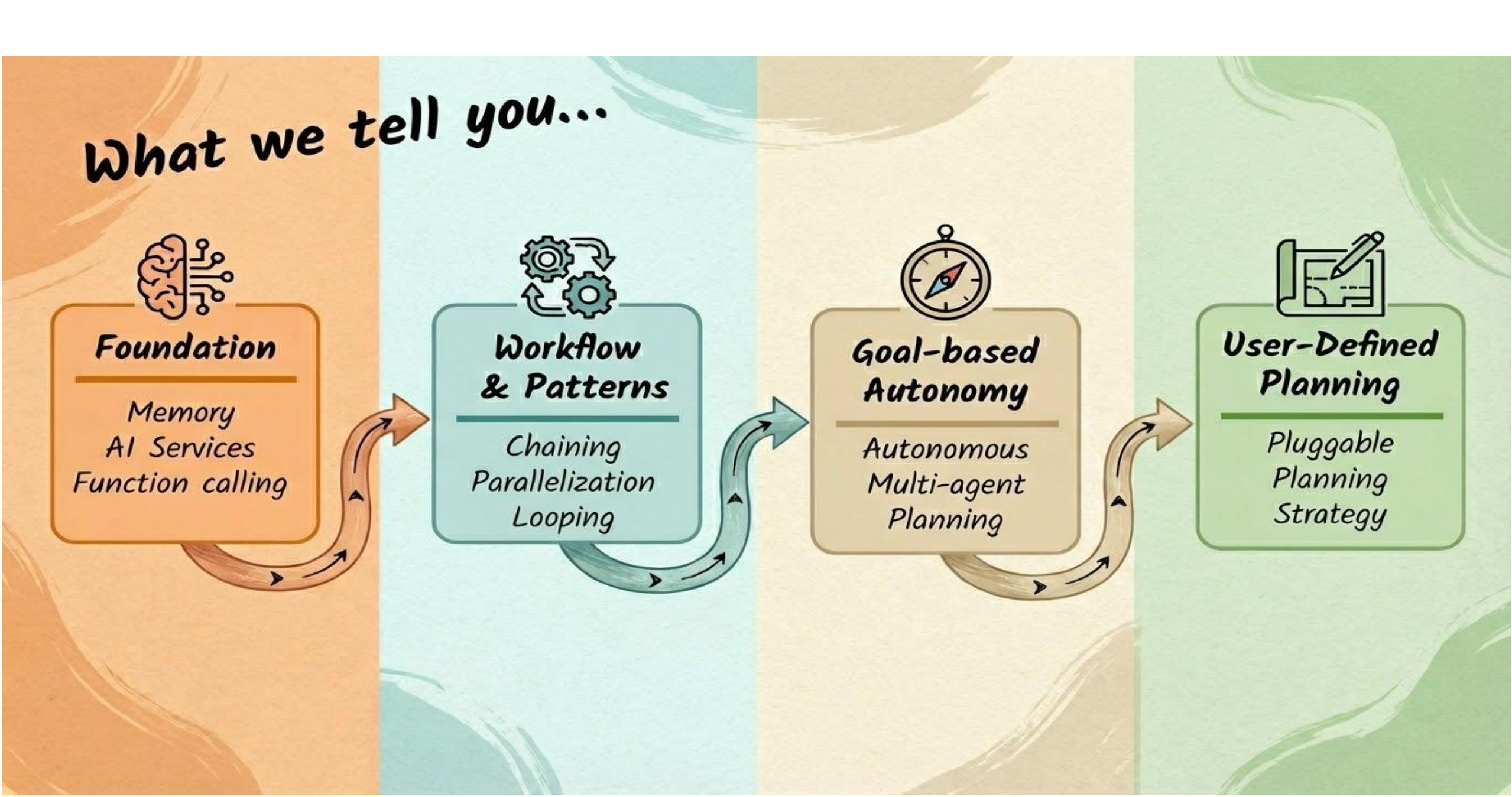
quarkus-agent-mcp

github.com/quarkusio/quarkus-agent-mcp

AND BEYOND: AGENTIC AI

Java can orchestrate multi-agent systems too

What we tell you...



MCP DEV SUMMIT SEOUL

LIVE DEMO

Weather Station with Multi Agents

Java | QUARKUS | LangChain4j | MCP

Collaborative AI Agents • Real-time Data • Intelligent Insights

KEY TAKEAWAYS

- Writing an MCP server is like **writing a REST endpoint**
- **Checked at build time.** Zero reflection, ~30 MB, milliseconds to start
- The Java ecosystem **already had the answers**: Bean Validation on arguments, OIDC on endpoints, Micrometer and OpenTelemetry on calls, CDI for guardrails
- **McpAssured** tests the protocol the way RestAssured tests HTTP
- Same code over STDIO or Streamable HTTP, **transport is a packaging choice**, not a rewrite

Try it yourself: **quarkus.io/quarkus-workshop-langchain4j/**

IBM Bob Free Trial

Thank you!

docs.quarkiverse.io/quarkus-mcp-server

docs.quarkiverse.io/quarkus-langchain4j

quarkus.io/quarkus-workshop-langchain4j/

github.com/kdubois/netatmo-java-mcp

github.com/kdubois/homebot