The 14th International Workshop on Load Testing and Benchmarking of Software Systems (LTB 2026)
co-located with
The 17th ACM/SPEC International Conference on Performance Engineering (ICPE 2026)

Performance Evolution of Jakarta EE Application Servers

Daniele Di Pompeo, Andrea Reale, Michele Tucci

SPENCER Lab (University of L'Aquila)
& Sapienza Universitá di Roma

spencer
Software Performance Engineering Laboratory

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

SAPIENZA
UNIVERSITÁ DI ROMA

Florence May, 4 2026

Motivation: The RxNetty vs. Tomcat Paradigm Shift (April 2015)

RxNetty (Reactive)

- Asynchronous, Non-blocking I/O
- Event-driven Architecture
- High Concurrency, Low Latency

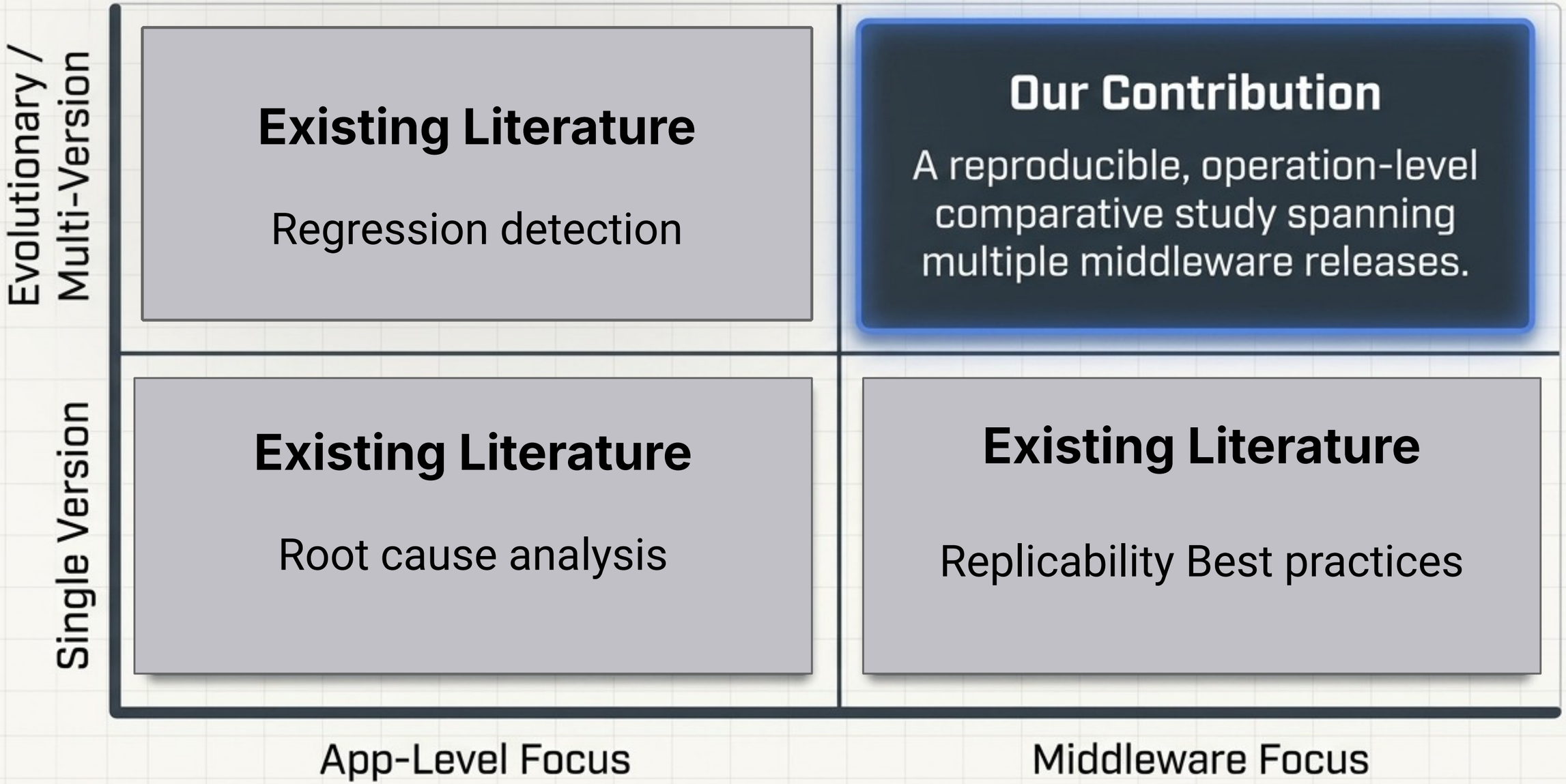
Tomcat (Traditional)

- Thread-per-Request Model
- Blocking I/O
- Mature, but scalability challenges at peak load

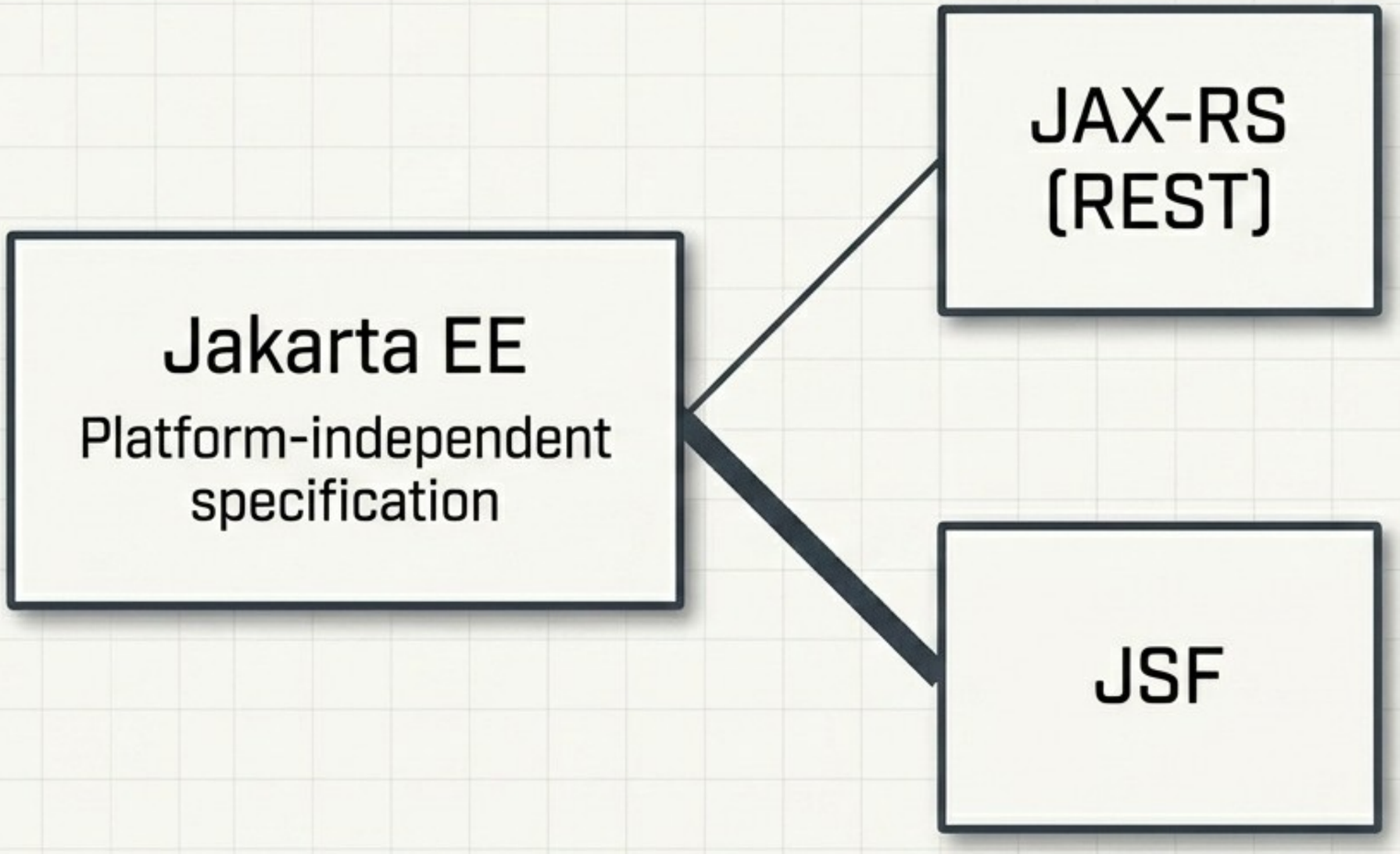
This industry shift highlights the critical need to understand the *performance evolution* of traditional application servers under modern workloads.

Brendan Gregg blog post:
https://www.brendangregg.com/Slides/RxNetty_vs_Tomcat_April2015

Beyond Application-Level Benchmarking



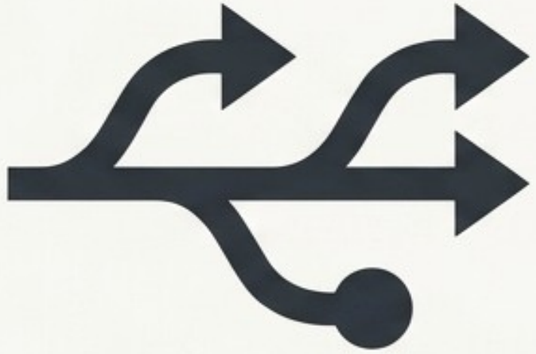
The Jakarta EE Web Profile



- Client-side rendering
- Lightweight request routing
- Content negotiation

- Server-side rendering
- Component-based framework
- Heavy view rendering and state management

Uncovering the Trajectory of Server Evolution

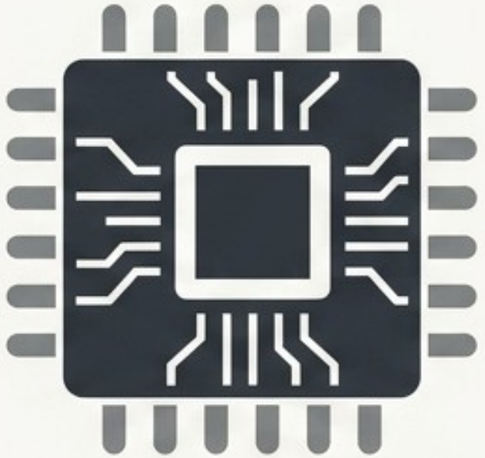


RQ1

How do performance characteristics evolve across versions?

RQ2

Do servers favor specific Jakarta EE technologies (REST vs. JSF vs. WebSockets)?

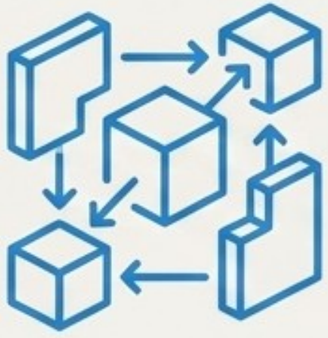


RQ3

How do memory footprints and potential leaks behave under sustained load?

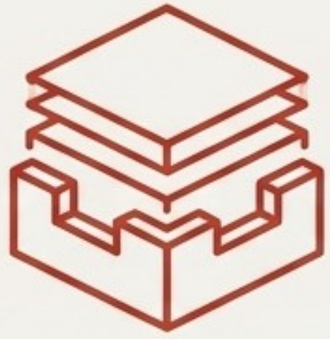
A Tale of Two Architectures

IBM OpenLiberty



- **Versions Evaluated:** v18–21
- **Core Philosophy:** Dynamic feature provisioning
- **Key Trait:** Highly modular core
- **Ecosystem:** Eclipse MicroProfile integration

Red Hat WildFly



- **Versions Evaluated:** v16–22
- **Core Philosophy:** Strict classloading isolation
- **Key Trait:** JBoss modules
- **Ecosystem:** Undertow embedded web server

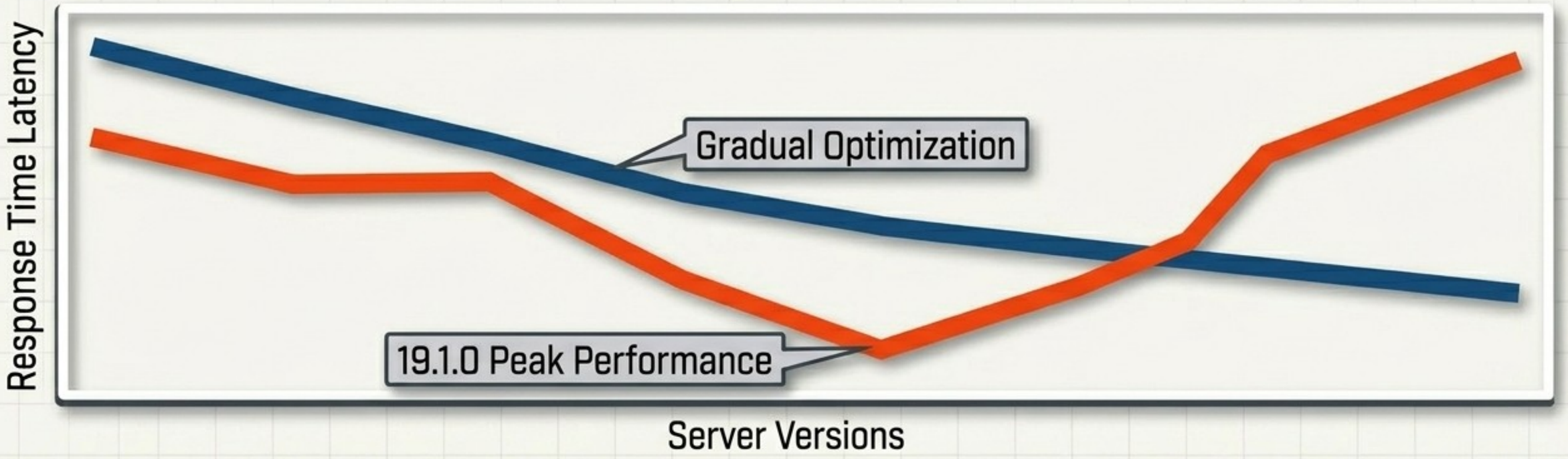
The Question: Does a dynamic, modular core outperform a strictly isolated architecture under enterprise load?

Experimental Design and Infrastructure

Execution: Dual-run strategy (Baseline for latency + JFR for profiling).

Hardware & Config	
CPU:	AMD Ryzen 9 5950X
RAM:	32GB
Runtime:	OpenJDK 8
Workload:	SPECJEnterprise 2018 (28 operations)
Harness:	Modified Faban framework

RQ1: Regression vs. Gradual Optimization

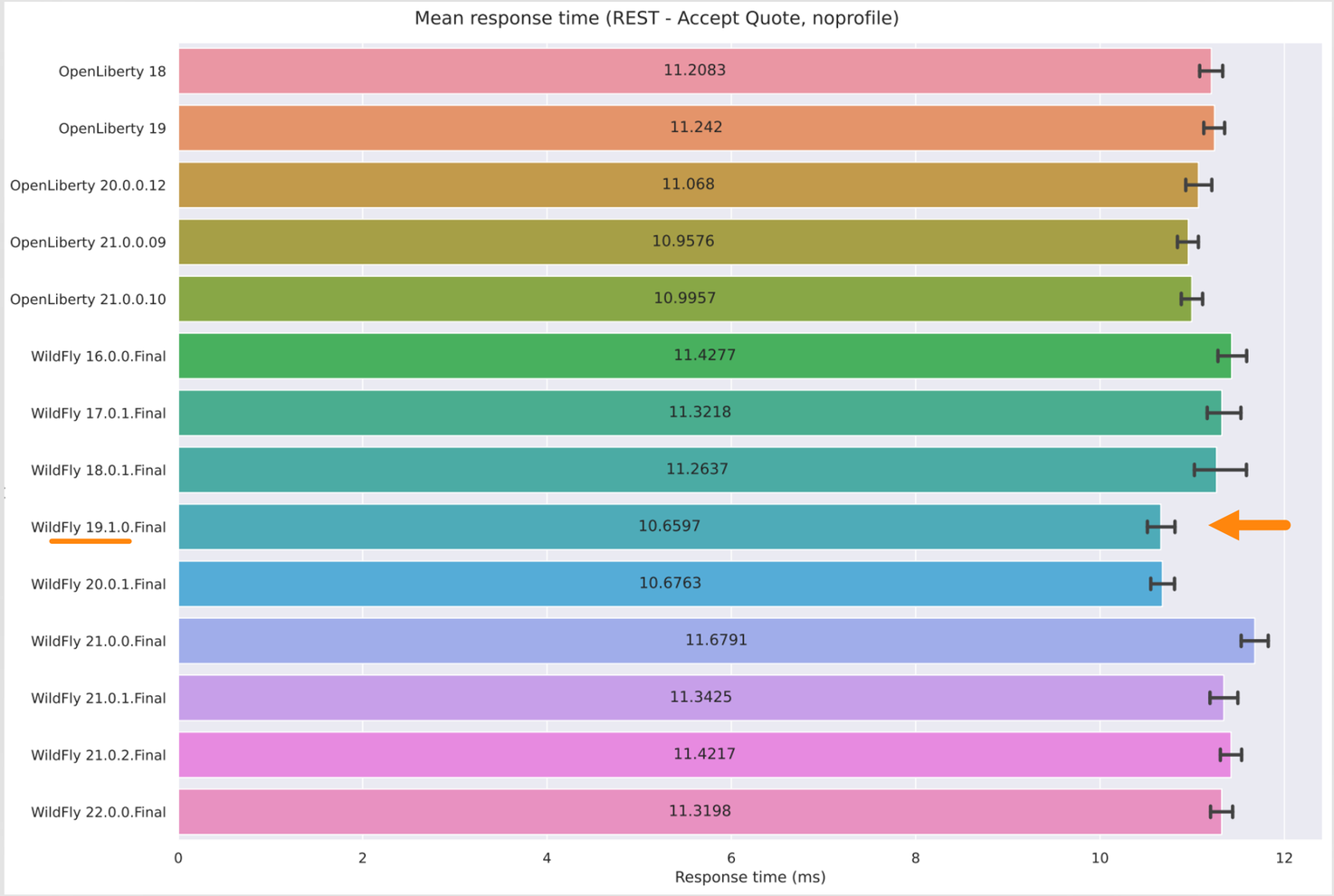


OpenLiberty: Demonstrated continuous, gradual latency improvements across releases.

WildFly: Peaked at version 19.1.0; subsequent versions regressed due to heavy core dependency updates.

Takeaway: Continuous regression testing is vital; dependency churn quietly erodes throughput.

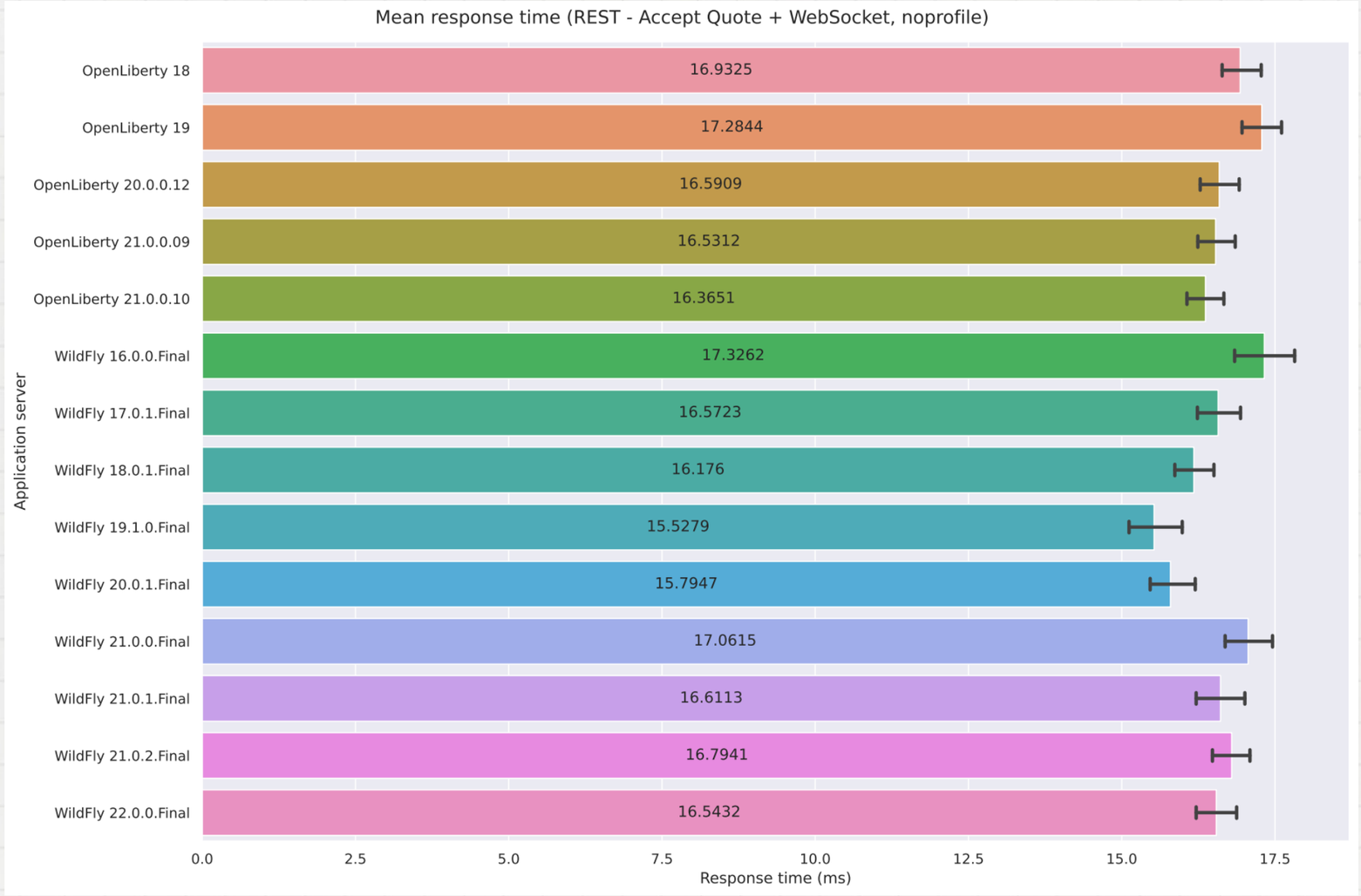
RQ2: Comparable Performance in REST Operations



REST architectures show minimal latency across both servers (~10.6 to 11.6 ms).

Performance is highly competitive between implementations. WildFly 19.1.0 edges out slightly as the peak performer (10.6 ms).

RQ2: The Overhead of WebSockets

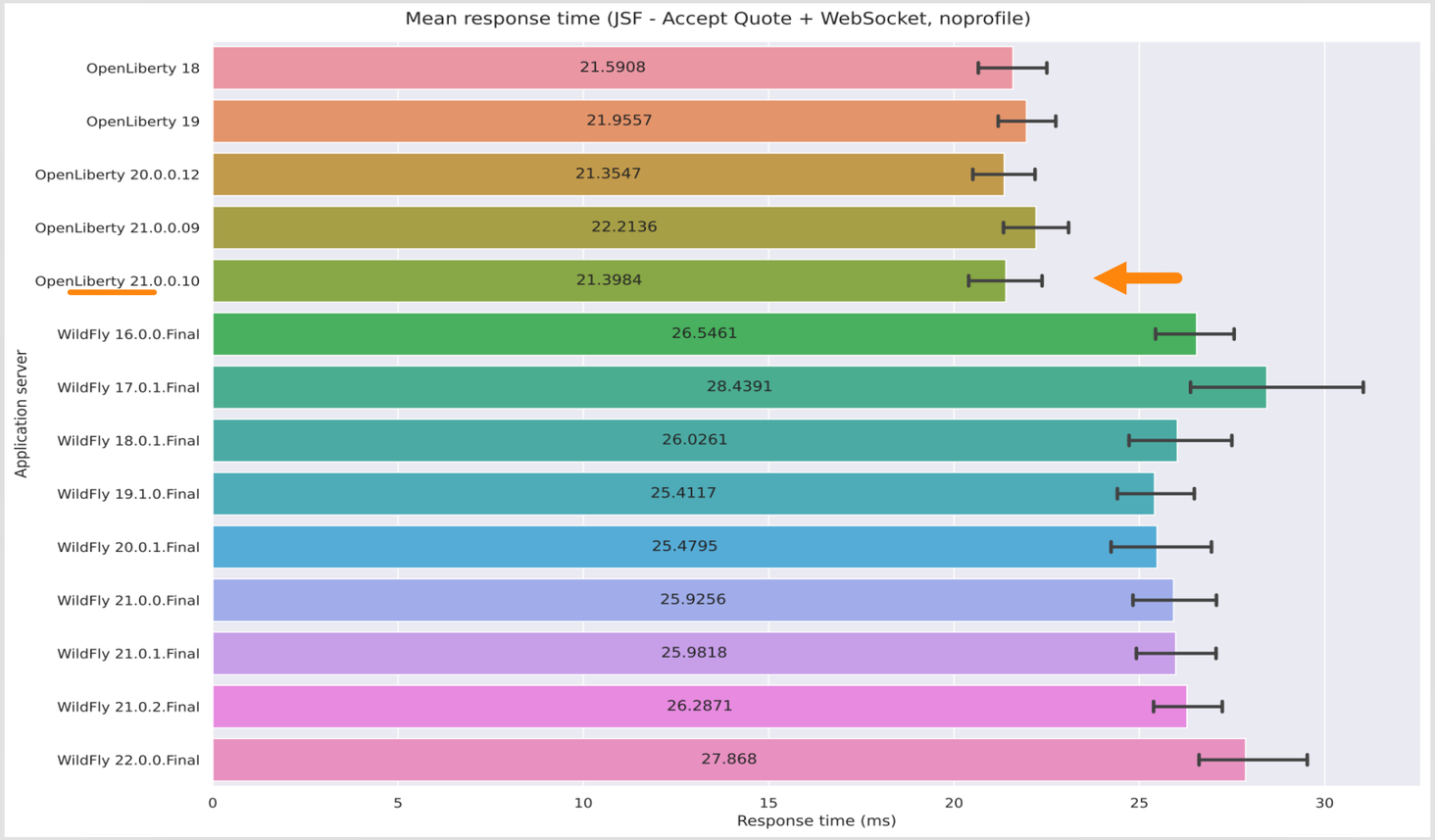


Adding **WebSocket** support increases REST latency by **~50%** (~16.4–17.3 ms).

WildFly 19.0.0 maintains top performance (~15.5 ms).

OpenLiberty 21 closely follows (~16.3 ms) and remains highly competitive.

RQ2: The JSF Rendering Penalty

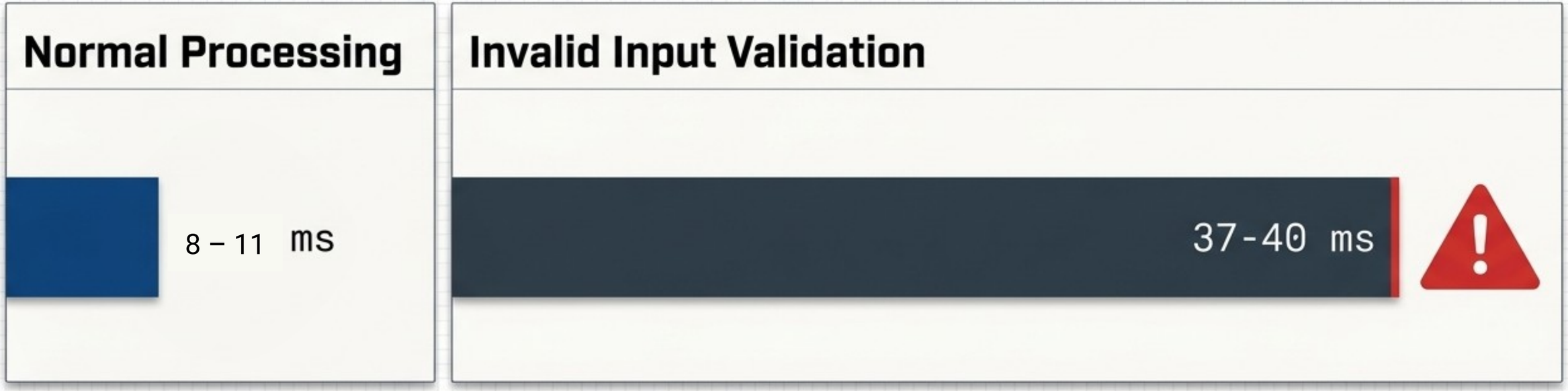


JSF Burden: Requests are 6-7x slower than REST due to heavy server-side component rendering.

OpenLiberty Lead: Consistently outperforms WildFly by 4-6 ms for JSF + WebSockets.

WildFly Variance: Exhibits much longer latency tails (up to 48 ms variance in version 17).

RQ2: Handling Edge Cases and Invalid Inputs



OpenLiberty degrades severely upon encountering invalid input data. Validation routines cause an 8x slowdown.

Highlights an asymmetric vulnerability to validation-based abuse or sudden load spikes.

RQ3: Memory Footprint and Resource Allocation

160 MB
Stabilized

OpenLiberty

Stable Execution: Zero memory leaks detected under sustained 60-minute load.

100 MB
Stabilized

WildFly

CPU Hotspots: TLS / Cryptographic sampling & Reflection

CPU Hotspots: JSON parsing & Tokenization

Shared Bottleneck: HashMap lookups dominate CPU time across both servers due to JSF component metadata.

Architectural Synthesis & Key Takeaways

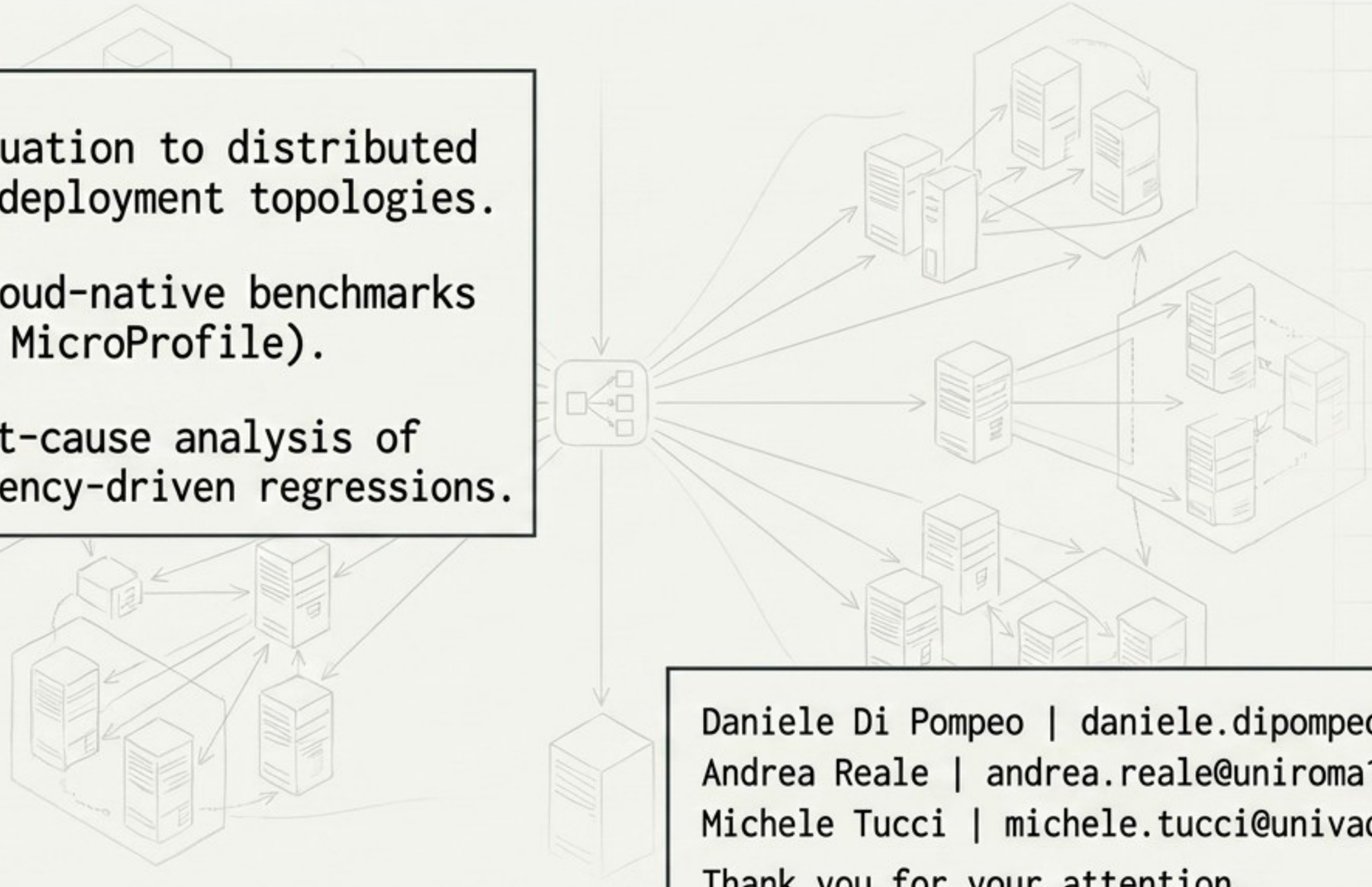
Workload Profile	Recommended Server	Expected Overhead
Pure REST	Interchangeable (**WildFly** 19 Peak)	Minimal (~11 ms)
REST + WebSockets	**WildFly** 19 / **OpenLiberty** 21	~50% Latency Tax
JSF Workloads	**OpenLiberty**	6-7x slower than REST

Crucial Rule: Do not blindly update.

Dependency updates shipped with new versions directly impact bottom-line performance.

Future Horizons

- Expanding evaluation to distributed and clustered deployment topologies.
- Integrating cloud-native benchmarks (e.g., Eclipse MicroProfile).
- Conducting root-cause analysis of specific dependency-driven regressions.



Daniele Di Pompeo | daniele.dipompeo@univaq.it
Andrea Reale | andrea.reale@uniroma1.it
Michele Tucci | michele.tucci@univaq.it

Thank you for your attention.