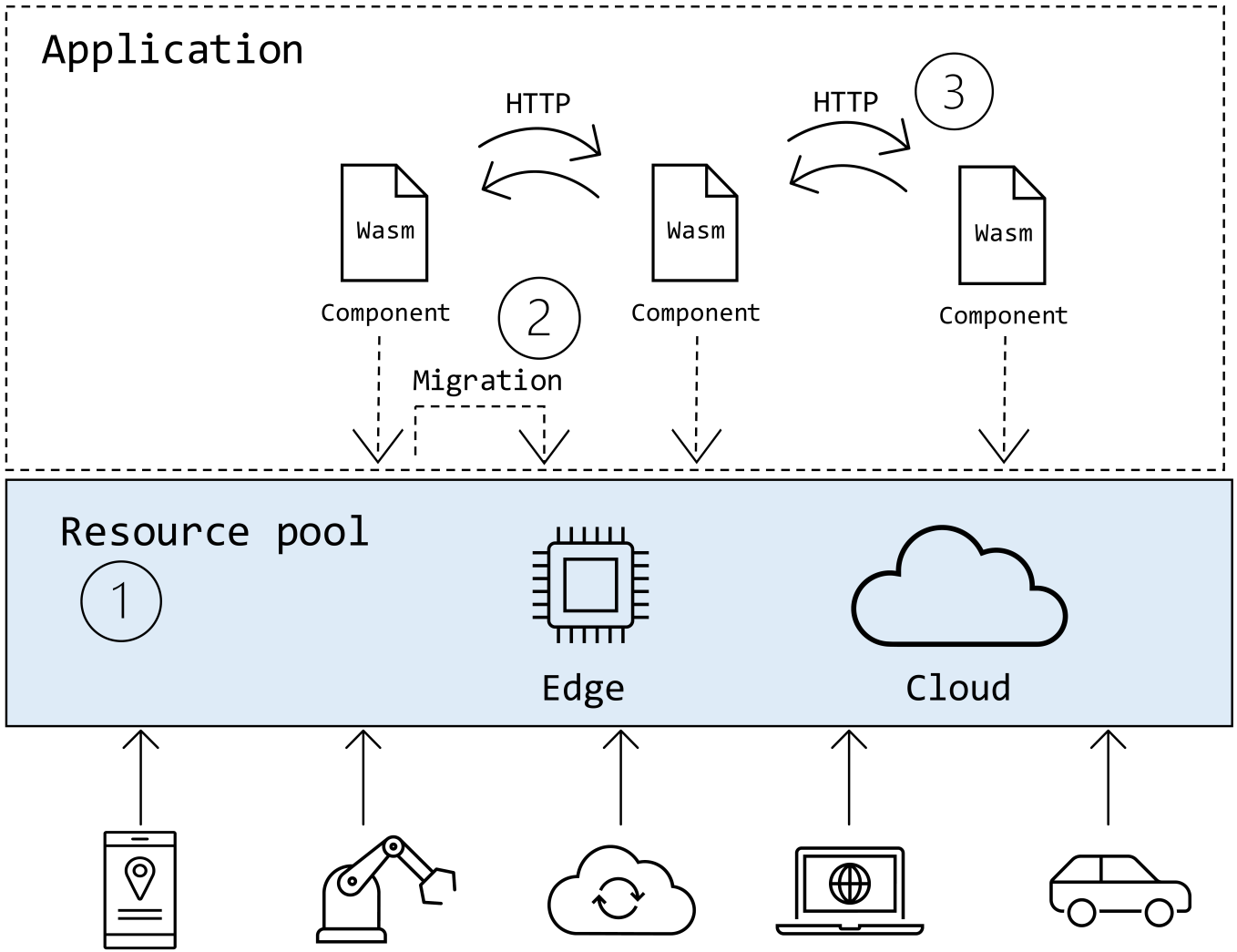
A runtime infrastructure for the Continuum of Computing

Edoardo Tinto, Tullio Vardanega

Department of Mathematics, University of Padova, Italy
edoardo.tinto@phd.unipd.it, tullio.vardanega@unipd.it

HPDC '24 PhD Symposium
June 6, 2024

A Model for the Continuum of Computing



On the use of WebAssembly and Rust

WebAssembly (Wasm)

- A common *sandboxed* execution environment
- A compilation target for several programming languages

Memory-safe concurrent programs with **Rust**

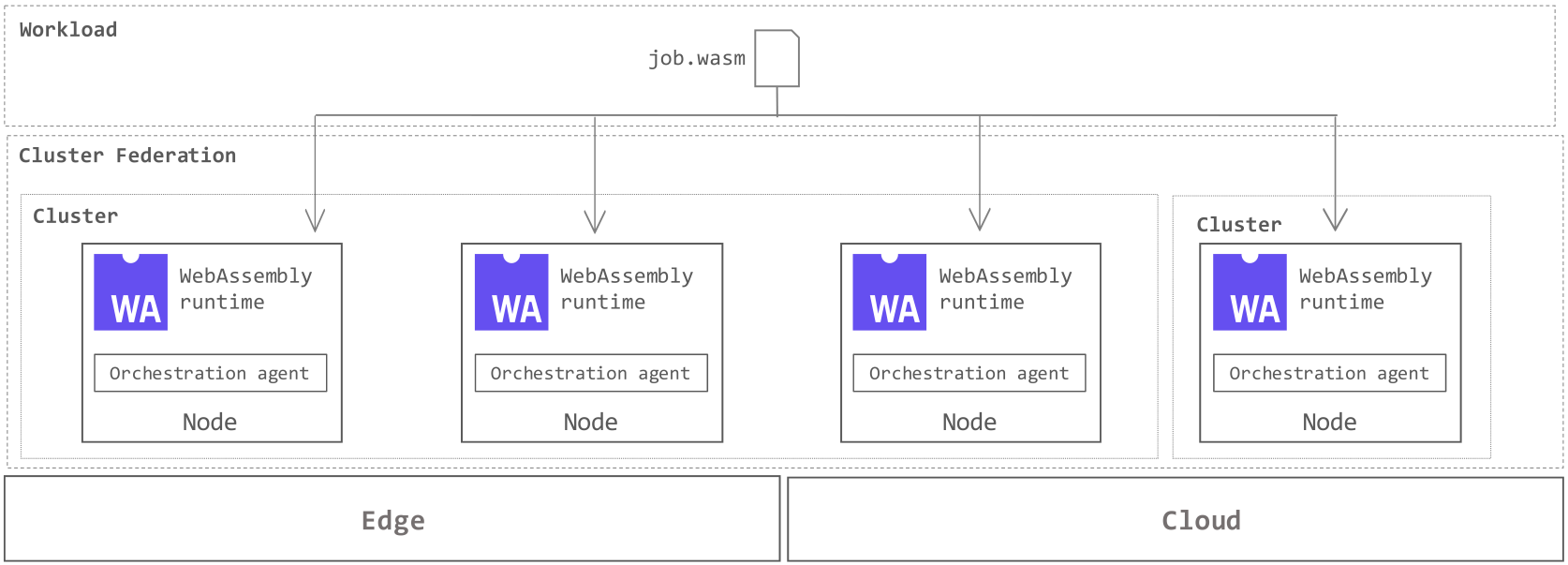
- Static, compile-time assessment of memory safety

Jointly they allow building a *safe-by-design* continuum infrastructure

Resource Pooling for High Performance Computing

The Continuum may suit several use cases

- *Throughput computing* systems, with loosely coupled jobs
- *Message-passing* systems, where communication-based collaboration matters



Project Goals

- Develop a migration-capable runtime infrastructure
  - Starting from existing Wasm runtimes (such as WAMR and Wasmtime)
- Orchestrate migrating computations
  - Do state-of-the-art orchestrators (e.g., K8s) suffice?
- Assess flexibility, performance, and overhead of the overall solution
  - Through benchmarking and real-world use cases

A Migration-Capable Runtime Infrastructure

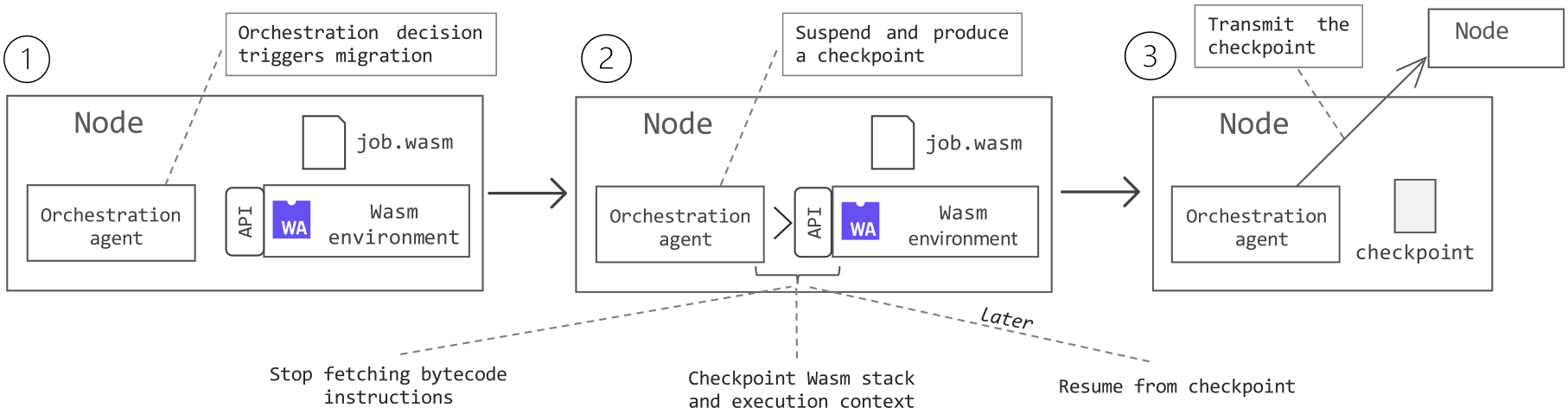
To be of interest to real-world applications, migration should be *live*

- Preserve the **state**, hence advancement, of the computation (*stateful*)
- Preserve any existing **dependencies**, directed toward
  - Other modules
  - Linear memories
  - Host-specific functionalities or specific hardware
- **Connections** (web-based in our model) should be preserved

The case for WAMR (WebAssembly Micro Runtime)

We have introduced two new APIs (application programming interfaces)

- `wasm_runtime_request_checkpoint`, triggering the suspend and checkpoint procedures
- `wasm_application_execute_func`, enriched to restore an input checkpoint



The case for WAMR (WebAssembly Micro Runtime)

To achieve *live* migration, after resuming

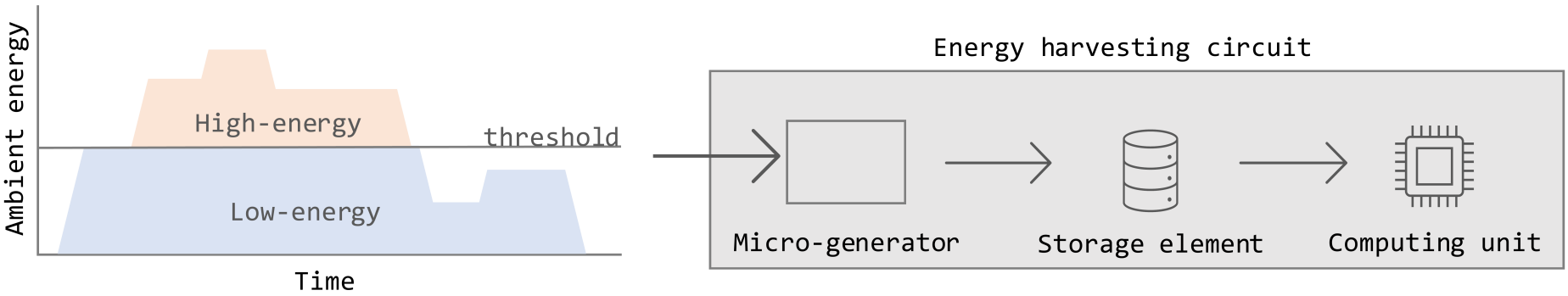
- The advancement of the computation should be preserved (github.com/TintoEdoardo/wasm-micro-runtime-interp-migration)
- Dependencies to other modules should be preserved (github.com/TintoEdoardo/wasm-micro-runtime-interp-migration)
- Access to additional memories should be preserved
- Active connections should be preserved (to be done)

Next Steps

1. Achieve live migration of *compiled* components
   - How to interrupt the computation in a *consistent* state?
   - How to resume from a different instruction set architecture (ISA)?
   - Focus is on components written in Rust
2. Orchestrate migrating components
   - Migration should be triggered by orchestration-level decisions
   - Orchestration should be *multi-level*, at node and cluster levels
   - Are there any candidate orchestrators for the *Continuum*?

Other Use Cases

- Software predictability in the Continuum
  - Edge devices may be used in highly critical scenarios
  - *Predictability* and *timeliness* are paramount for real-time applications
- Other scenarios might benefit from migrating computations
  - For example, Energy Harvesting Systems
    - Migrating instead of suspending during low-energy phases
    - Aiming at improving performance



Conclusions

- The envisioned model features
  1. A **single pool of resources**, across Edge *and* Cloud
  2. **Migrating computations** across that continuum, in accord with user needs and requirements
  3. Interactions over **web-level protocols**
- We are finalizing a migration-capable interpreter for Wasm components
- The following steps include
  1. Migration of compiled Wasm components
  2. Orchestration of dynamic aggregation and migration of computations

Acknowledgements and funding source

The work carried out in this project was funded under the National Recovery and Resilience Plan (NRRP), Italy, Mission 4 Component 2 Investment 1.4 - Call for tender No. 3138, 16 December 2021, by the Italian Ministry of University and Research funded by the European Union – NextGenerationEU.