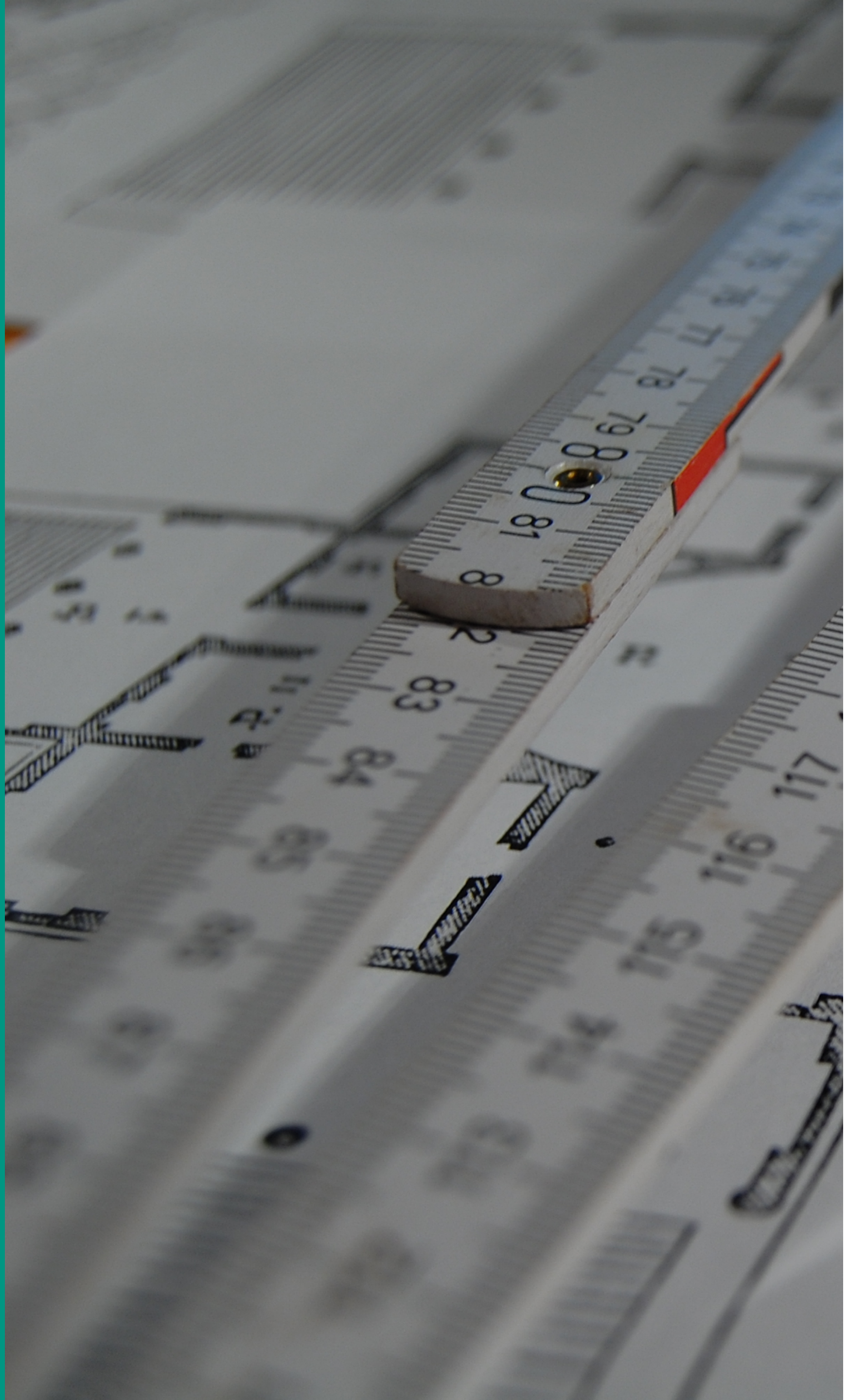
Generation of Checkpoints for Hardware Architecture Simulators

Sebastian Weber[*], Lars Weber[†], Thomas Weber[†], Jörg Henß[*], Robert Heinrich[‡]

Thomas Weber[†] | 4th November 2025

[*]FZI Research Center for Information Technology
[†]Karlsruhe Institute of Technology
[‡]Ulm University



FZI KIT

SDQ Software Design and Quality

Motivation

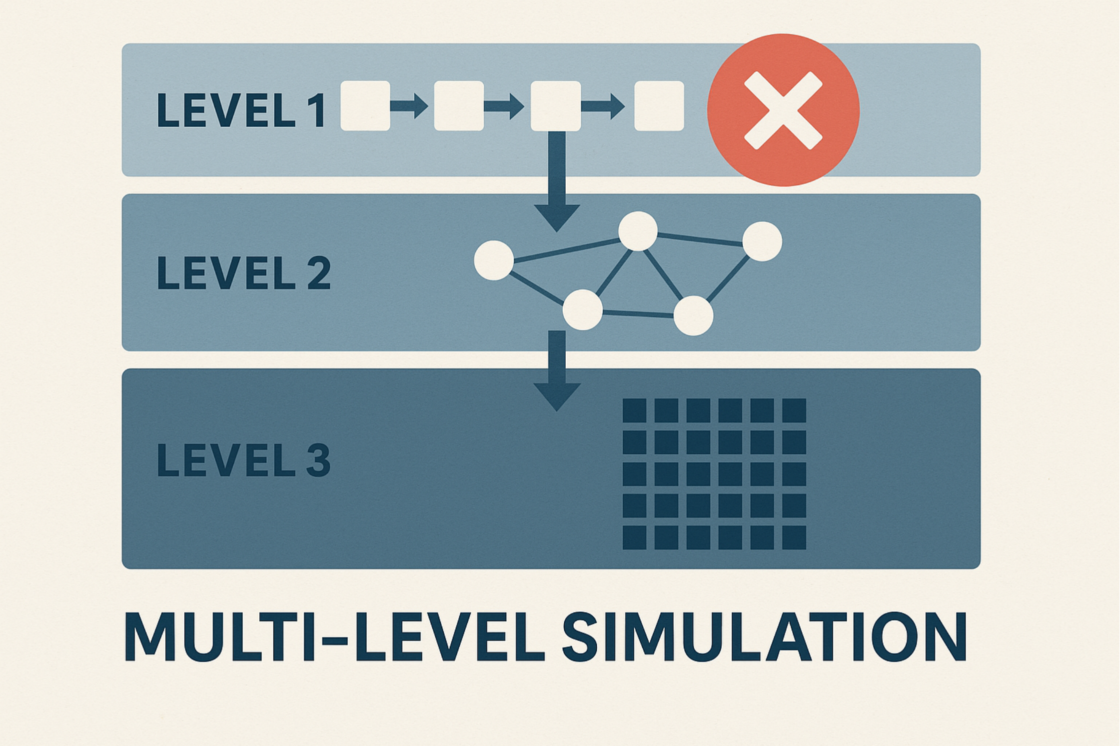


Figure: Abstract view on multiple simuluations on different levels of granularity.

Switching Levels

- Car simulation on autumn street

Motivation

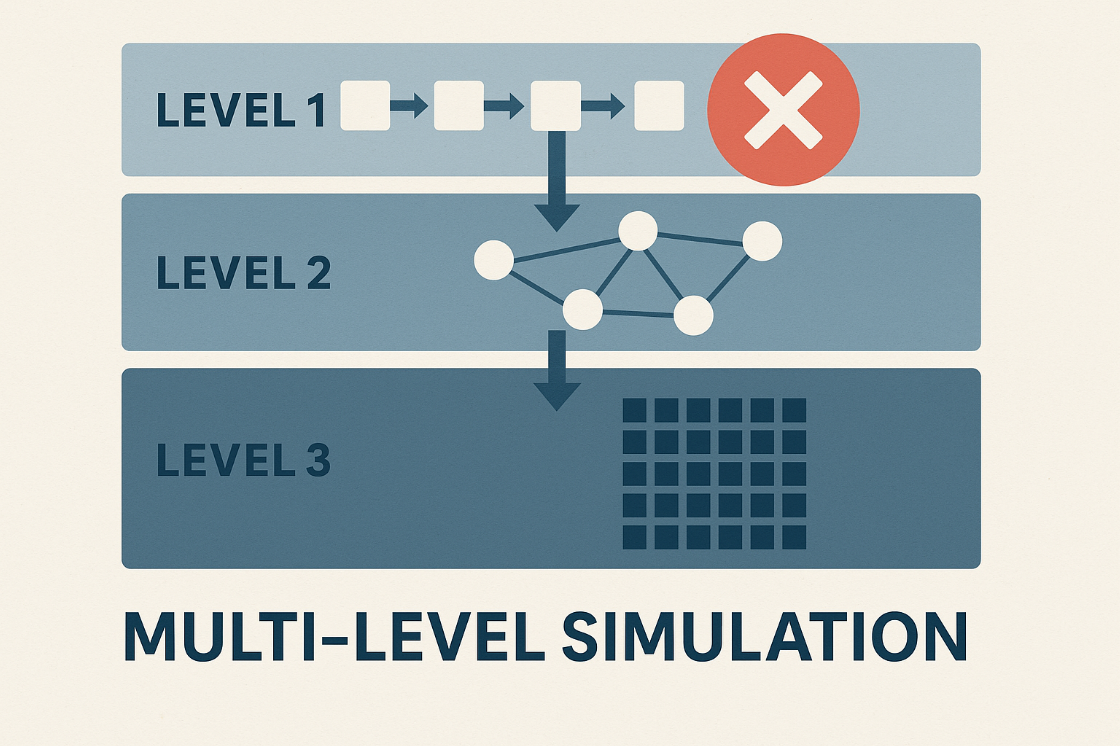


Figure: Abstract view on multiple simuluations on different levels of granularity.

Switching Levels

- Car simulation on autumn street
- Simulating on E/E-Architecture level
- Change in surface, e.g., wet leaves, requires different simulation

Motivation

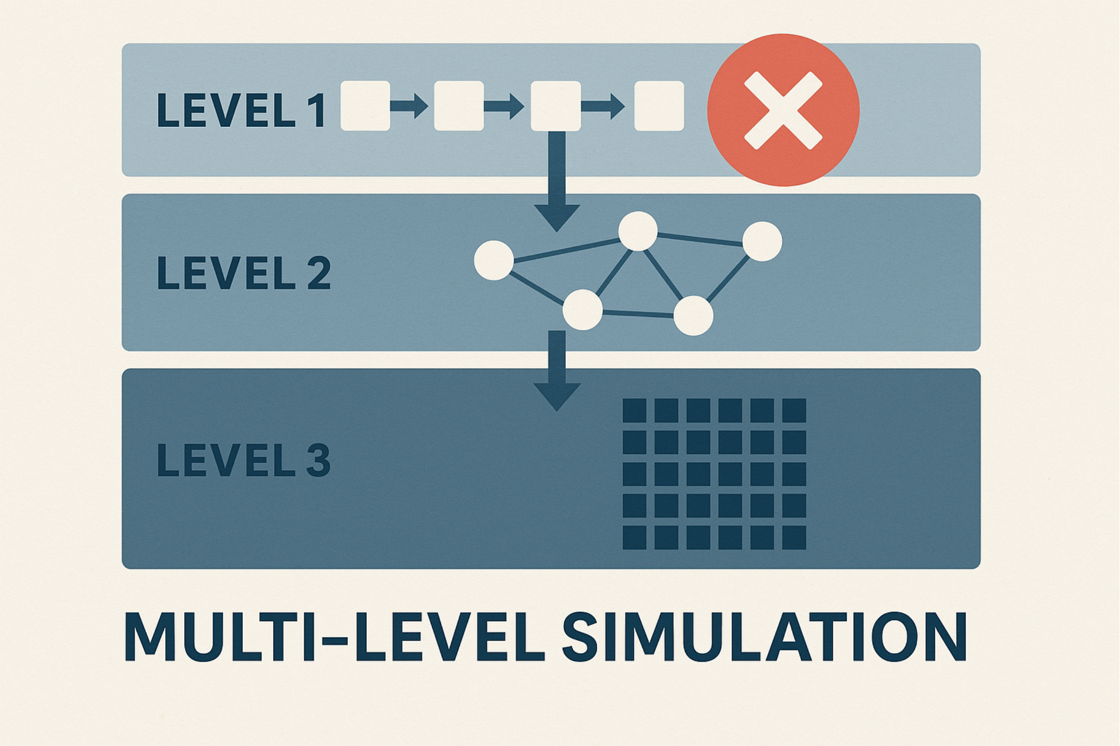
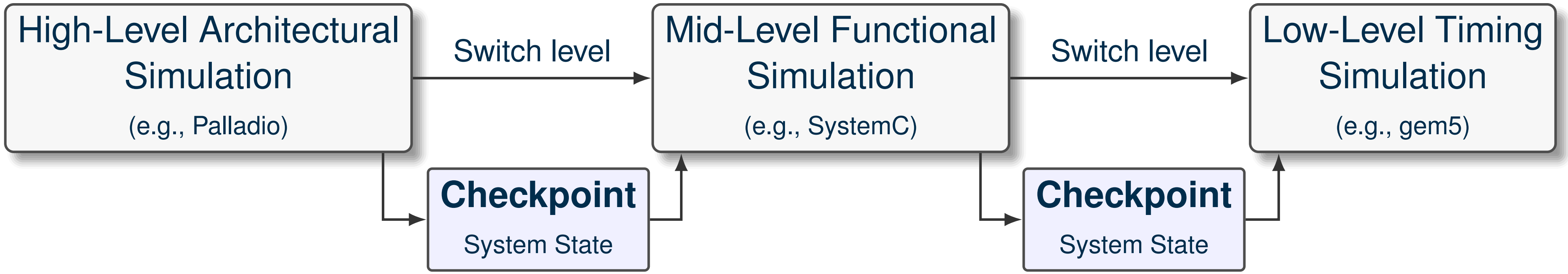


Figure: Abstract view on multiple simuluations on different levels of granularity.

Switching Levels

- Car simulation on autumn street
- Simulating on E/E-Architecture level
- Change in surface, e.g., wet leaves, requires different simulation
- Not only on physical level
- But also increased load on ECUs due to compensation calculations

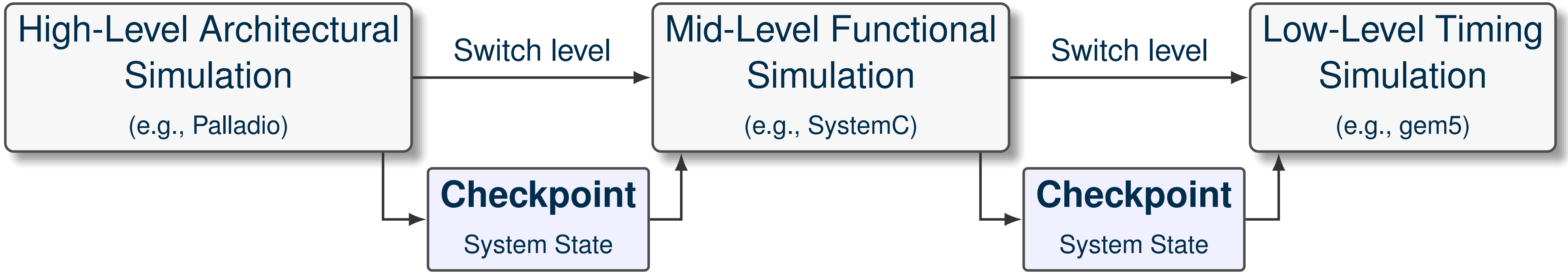
Motivation



Checkpoint

- Internal state of a (modeled) system at a given point in time
- Allows to start simulation at this point in time

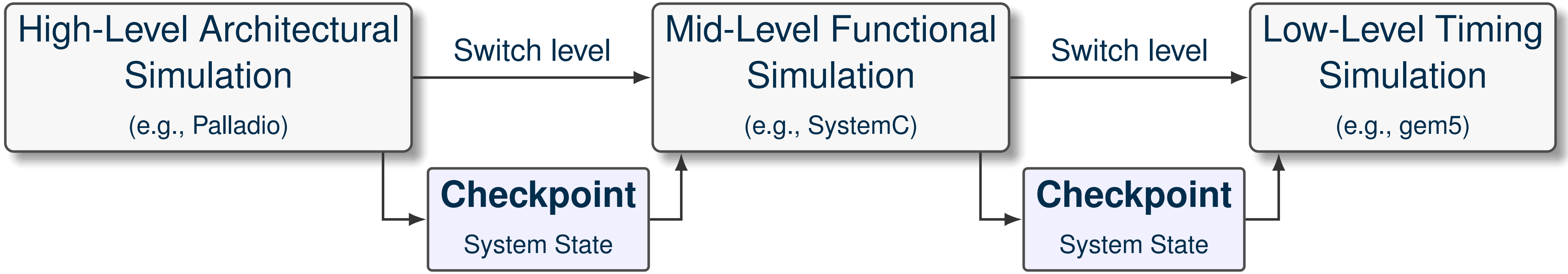
Motivation



Checkpoint

- Internal state of a (modeled) system at a given point in time
- Allows to start simulation at this point in time
- Transformation to initial states of other simulations
- Goal: use the most suitable level of abstraction at any point during the simulation

Motivation



Checkpoint

- Internal state of a (modeled) system at a given point in time
- Allows to start simulation at this point in time
- Transformation to initial states of other simulations
- Goal: use the most suitable level of abstraction at any point during the simulation
- Initialization bias: initial states can have huge impact on simulation results

Background

- “Quick Emulator”
- Open-source virtualization software

Background

- “Quick Emulator”
- Open-source virtualization software
- Supports broad range of processor architectures (e.g., x86, ARM, RISC-V)
- Emulation of system and processor architecture allows to extract system state

Background

- “Quick Emulator”
- Open-source virtualization software
- Supports broad range of processor architectures (e.g., x86, ARM, RISC-V)
- Emulation of system and processor architecture allows to extract system state

QEMU Interfaces

- **QEMU Machine Protocol (QMP)**
  - JSON-based protocol to control and query QEMU instances
  - Provides structured commands for automation and integration
  - Allows pausing, resuming, and inspecting virtual machines

Background

- "Quick Emulator"
- Open-source virtualization software
- Supports broad range of processor architectures (e.g., x86, ARM, RISC-V)
- Emulation of system and processor architecture allows to extract system state

QEMU Interfaces

- **QEMU Machine Protocol (QMP)**
  - JSON-based protocol to control and query QEMU instances
  - Provides structured commands for automation and integration
  - Allows pausing, resuming, and inspecting virtual machines
- **QEMU Human Monitor (QHM)**
  - Text-based command interface aimed at human readability
  - Can query detailed runtime information about devices and CPUs
  - No defined format for results, requires command-specific parsing
- Extraction of data should be based on these interfaces and not code modifications to remain valid across different QEMU versions

Related Work

QEMU-based Approaches

- **Checkpoint Extraction for parallelized distributed simulation by Baudis 2013**
  - Extract checkpoints from virtualized systems in QEMU
  - Modified QEMU source code and QHM commands for data extraction
  - Checkpoint data deduplication with hashing algorithm

Related Work

QEMU-based Approaches

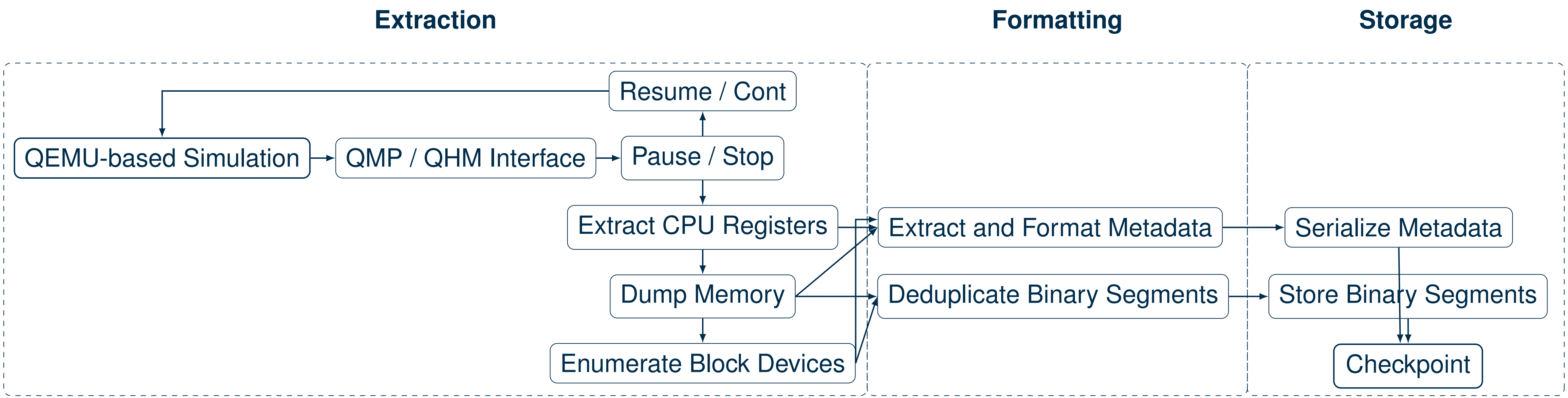
- **Checkpoint Extraction for parallelized distributed simulation by Baudis 2013**
  - Extract checkpoints from virtualized systems in QEMU
  - Modified QEMU source code and QHM commands for data extraction
  - Checkpoint data deduplication with hashing algorithm
- **Deterministic replay in QEMU for dynamic analysis by Dovgalyuk 2012**
  - Logging all non-deterministic events in QEMU, deterministic ones are simulated
  - Targeted at debugging and behavioral analysis rather than simulator coupling

Related Work

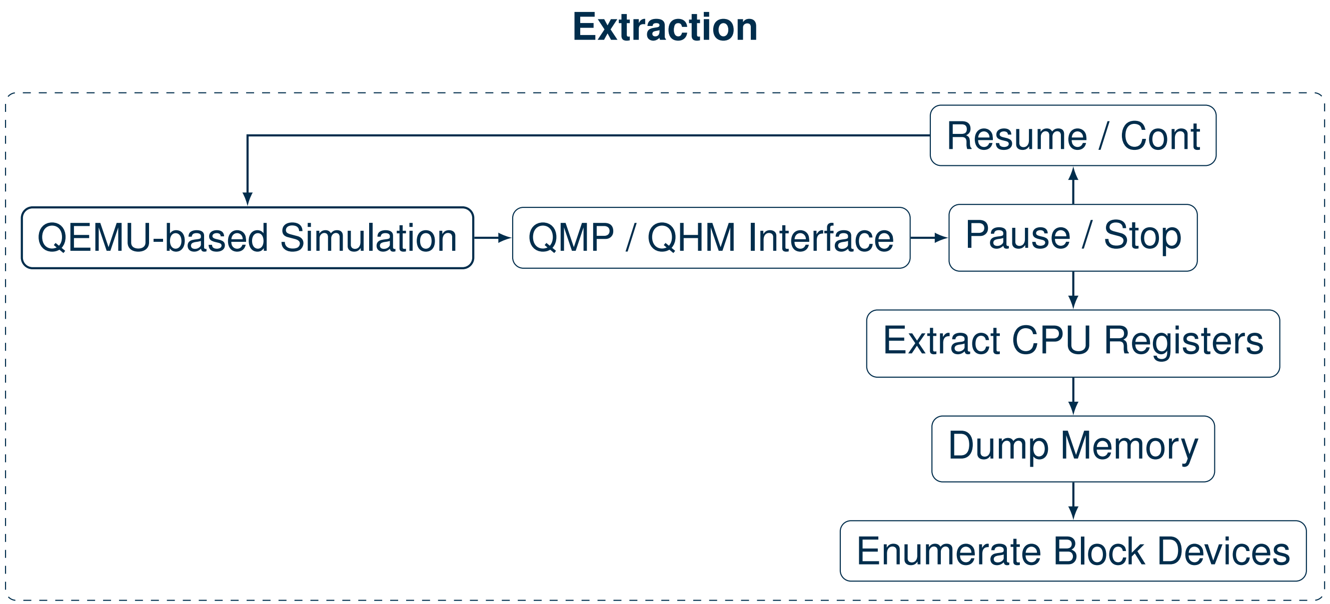
gem5-based Approaches

- **QPoints: Cross-Platform checkpointing from QEMU to gem5 by Godala et al. 2023**
  - Full-system checkpoints from QEMU to gem5, combining fast emulation in QEMU with detailed simulation in gem5 for ARM-based systems without modifying QEMU
  - Supports hardware acceleration and multi-core checkpoints, but is limited to 64-bit ARM platforms
- **Lapidary: Accelerating Checkpoint Creation for gem5 Simulations by Weisse et al. 2019**
  - Creates gem5-compatible checkpoints by attaching to running programs via GDB, capturing register and memory state directly from bare-metal execution
  - Greatly reduces initialization time and enables parallel simulations

Approach — Overview



Approach — Extraction



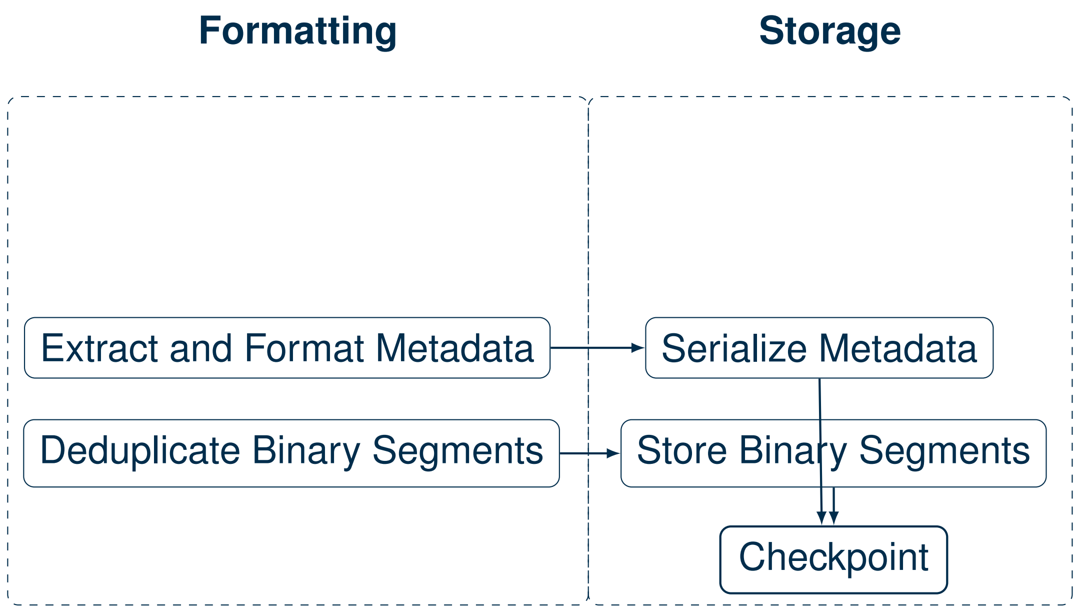
System State Extraction from QEMU

- Connect to QEMU via QMP and QHM
- Pause the VM to ensure consistent snapshot
- Capture CPU registers, memory dumps, and block device data
- Map each device to its corresponding image file for restoration

Approach — Formatting and Storage

Formatting and Storage of extracted system state

- Split extracted data into metadata and binary segments
- Store metadata (e.g., registers, configs) as JSON for portability
- Save memory and disk contents as binary files identified by SHA-256 hashes
- Use deduplication to avoid storing identical data across checkpoints



Evaluation

Setup

- Tested on an AMD Ryzen 9 7900X system with 48 GB RAM, NVMe SSD storage and Windows 11 Pro
- Evaluated using x86 and ARM virtual machines
- Workload is the Windows 11 setup (6.6 GiB image, 2 GiB RAM, 4 CPU cores)
- Measurements covered pause, extraction, and resume phases in QEMU

Evaluation

Setup

- Tested on an AMD Ryzen 9 7900X system with 48 GB RAM, NVMe SSD storage and Windows 11 Pro
- Evaluated using x86 and ARM virtual machines
- Workload is the Windows 11 setup (6.6 GiB image, 2 GiB RAM, 4 CPU cores)
- Measurements covered pause, extraction, and resume phases in QEMU

Goals and Results

- Correctness of Extraction, Formatting and Storage
  - All extracted data is stored correctly in the checkpoint
  - Manual comparison between checkpoint and QEMU command output
  - The developed tool correctly extracts CPU registers, memory dumps, and block device states
- Performance
  - Dominated by checkpointing the RAM
  - On average Windows checkpoints take 15 seconds

Evaluation

Threats to validity

- Results only manually validated against QEMU's internal snapshots
- Evaluation focused on a few representative systems (e.g., Windows 11, small Linux VMs) rather than a broad benchmark suite
- Performance results depend on the NVMe SSD used; slower storage could increase checkpoint times
- Regarding the motivation, only the extraction was tested, no reinjection or transformation to other simulators

Evaluation

Threats to validity

- Results only manually validated against QEMU's internal snapshots
- Evaluation focused on a few representative systems (e.g., Windows 11, small Linux VMs) rather than a broad benchmark suite
- Performance results depend on the NVMe SSD used; slower storage could increase checkpoint times
- Regarding the motivation, only the extraction was tested, no reinjection or transformation to other simulators

Limitations of the implementation

- Tool written in Java which restricts memory space for checkpoints to 2GB
- Currently supports only standard CPU, RAM, and block devices — no GPUs, TPMs, or external PCI devices
- Checkpoints can be extracted but not yet reloaded into a running QEMU instance
- While deduplication reduces redundancy, large binary segments can still consume significant disk space
- Checkpoint creation speed is limited by storage throughput, especially for large images

Conclusion

Results

- Checkpoint extraction tool for QEMU using only external interfaces (QMP, QHM)
  - Extract CPU registers, memory, and block devices
  - Format CPU registers in architecture-agnostic way
  - Deduplicate binary segments based on hashes to save disk space
- Evaluated correctness and performance on small set of examples

Conclusion

Results

- Checkpoint extraction tool for QEMU using only external interfaces (QMP, QHM)
  - Extract CPU registers, memory, and block devices
  - Format CPU registers in architecture-agnostic way
  - Deduplicate binary segments based on hashes to save disk space
- Evaluated correctness and performance on small set of examples

Future Work

- Reimplement the tool in a more performant programming language
- Add support for complex system architectures (e.g., GPU, FPGA)
- Evaluate the tool with a benchmark
- Test the reinjection of checkpoints into QEMU and other simulators

Conclusion

Results

- Checkpoint extraction tool for QEMU using only external interfaces (QMP, QHM)
  - Extract CPU registers, memory, and block devices
  - Format CPU registers in architecture-agnostic way
  - Deduplicate binary segments based on hashes to save disk space
- Evaluated correctness and performance on small set of examples

Future Work

- Reimplement the tool in a more performant programming language
- Add support for complex system architectures (e.g., GPU, FPGA)
- Evaluate the tool with a benchmark
- Test the reinjection of checkpoints into QEMU and other simulators

Contact

- Sebastian Weber
- sebastian.weber@fzi.de

References

[1] Nikolai Baudis. *Deduplicating Virtual Machine Checkpoints for Distributed System Simulation*. Bachelor's Thesis. Karlsruhe Institute of Technology (KIT). 2013. URL: `https://os.itec.kit.edu/downloads/ba_2013_baudis-nikolai_vm-checkpoints.pdf`.

[2] Pavel Dovgalyuk. "Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging.". In: *CSMR*. 2012, pp. 553–556.

[3] Bhargav Reddy Godala et al. "QPoints: QEMU to gem5 ARM Full System Checkpointing". In: *gem5 Workshop at ISCA 2023*. 2023. URL: `https://www.gem5.org/assets/files/workshop-isca-2023/posters/qpoints.pdf`.

[4] Ofir Weisse et al. "NDA: Preventing Speculative Execution Attacks at Their Source". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-52. Columbus, OH, USA: ACM, 2019, pp. 572–586. ISBN: 9781450369381. DOI: `10.1145/3352460.3358306`. URL: `https://doi.org/10.1145/3352460.3358306`.

Acknowledgements

The HAL4SDV project is co-funded by the Chips Joint Undertaking (Chips JU) and National Authorities under grant agreement n° 101139789.

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or National Authorities. Neither the European Union nor the granting authorities can be held responsible for them.

Co-funded by the European Union

Systems Safety Security Software

References