Comparative Evaluation of Lightweight Virtualization Technologies for Resource Constrained IoT Devices on RIOT OS

Master’s Thesis Defense

Outline

1. Introduction
2. Background on Virtualization
3. Qualitative Comparision
4. Empirical Evaluation
5. Conclusion & Outlook

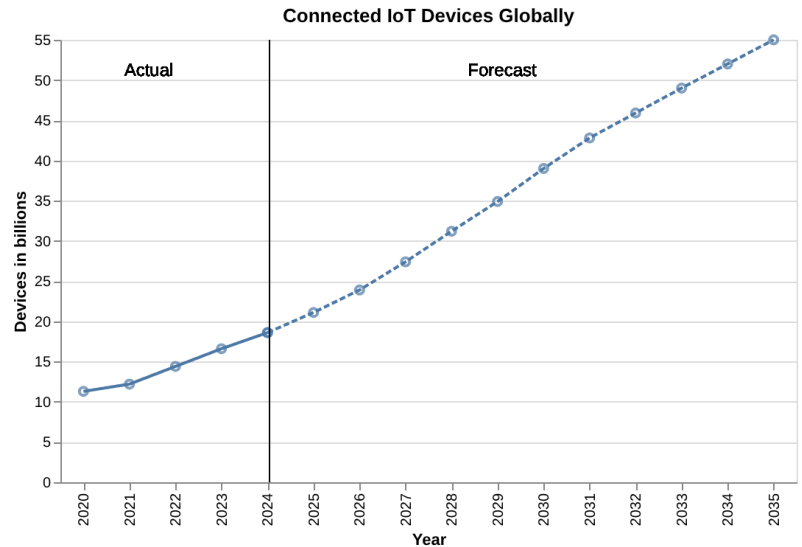
Motivation

- 55 billion devices by **2035**

Deployment Domains:

- Wearables
- Smart Home
- Health Care
- Manufacturing
- Agriculture
- ...

Consequence: Application complexity grows



Data from: IoT Analytics (Oct. 2025). State of IoT 2025. en-US. URL: https://iot-analytics.com/number-connected-iot-devices/ (visited on 01/05/2026).

Virtualization as an emerging paradigm for constrained devices

Potential Benefits:

- Memory Isolation
- Application Portability
- Usage of High-Level Programming Languages

Variety of different Solutions

Deployments:

FitBit LLC

Garmin LTD

Moddable Tech Inc

Tizen Project

Lack of comprehensive Evaluations

Research Goals

Classify virtualization approaches

Find relevant comparison metrics

Identify trade-offs

Research Goals

Classify virtualization approaches

Find relevant comparison metrics

Identify trade-offs

Method:

1. Qualitative Feature Comparison

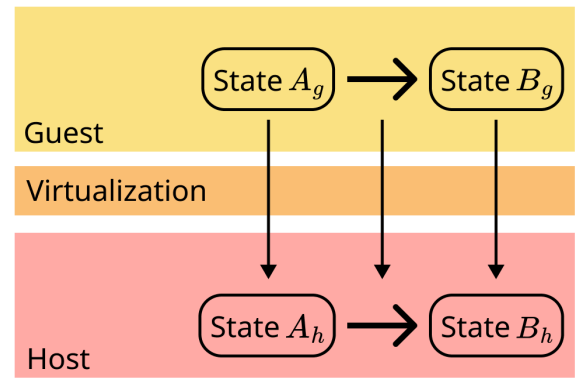
›

2. Empirical Evaluation on RIOT OS

What is Virtualization?

Definition by Popek and Goldberg (1974):

- virtual Guest and real Host
- Map Guest state and operation to Host

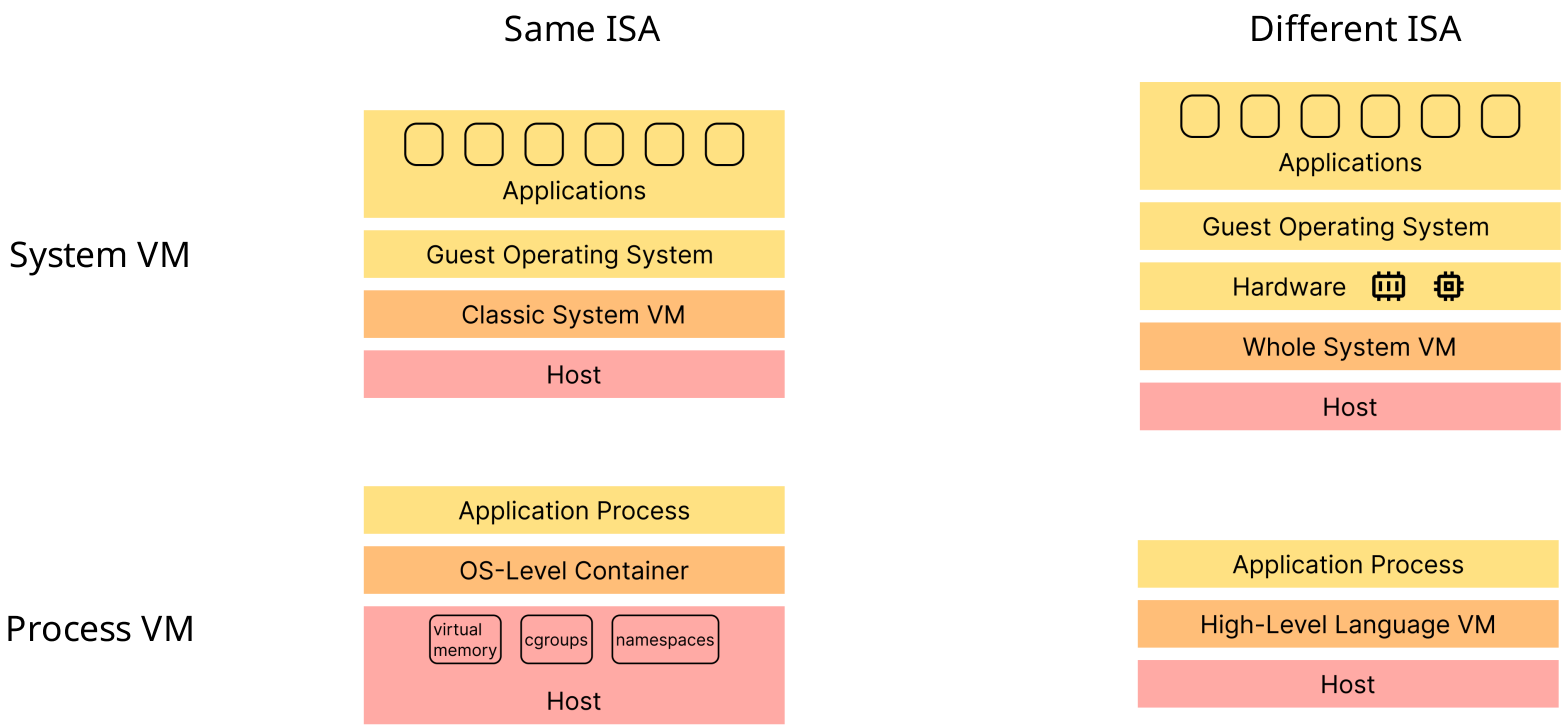


Categorization by J. E. Smith and Nair (2005)

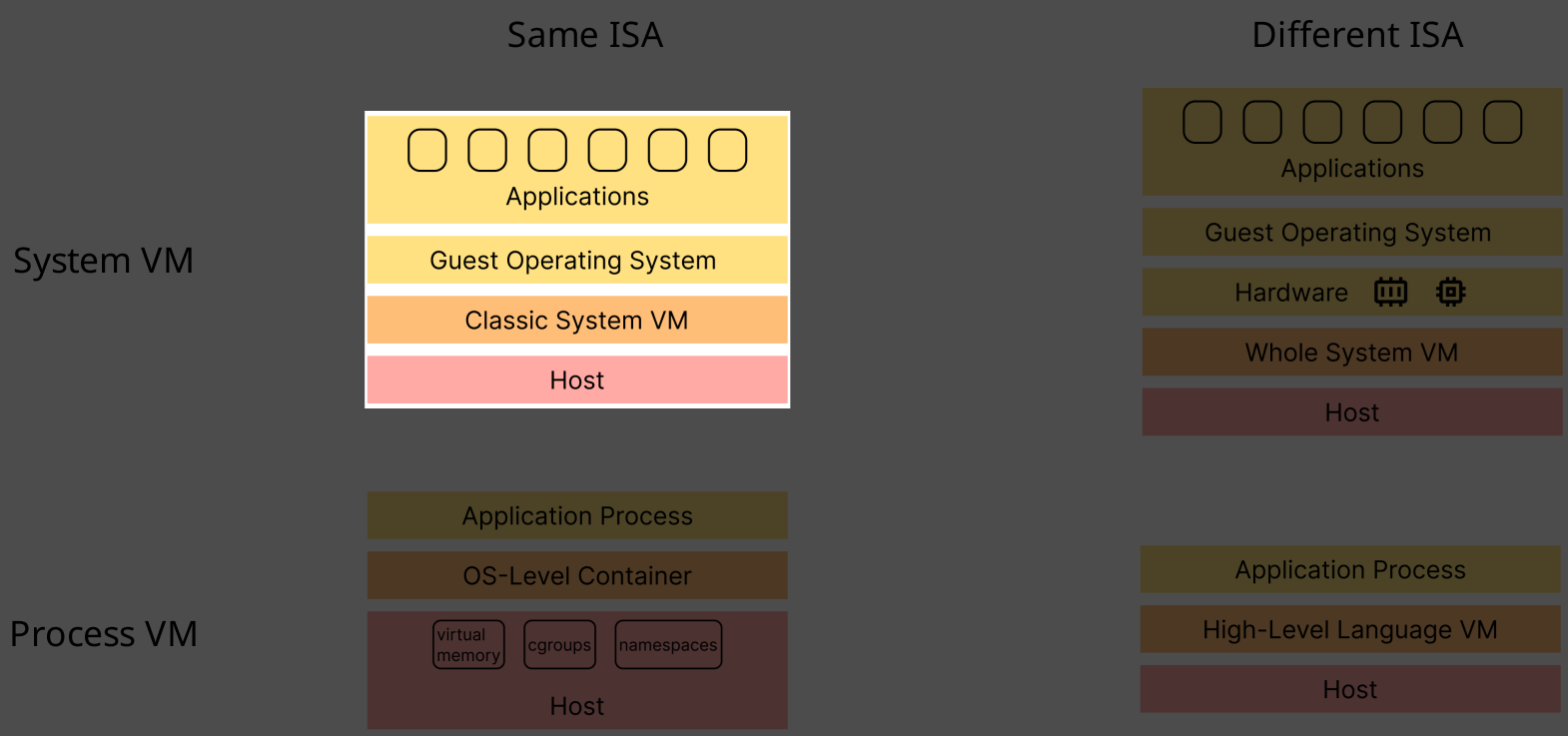
Same ISA vs. Different ISA

System VM vs. Process VM

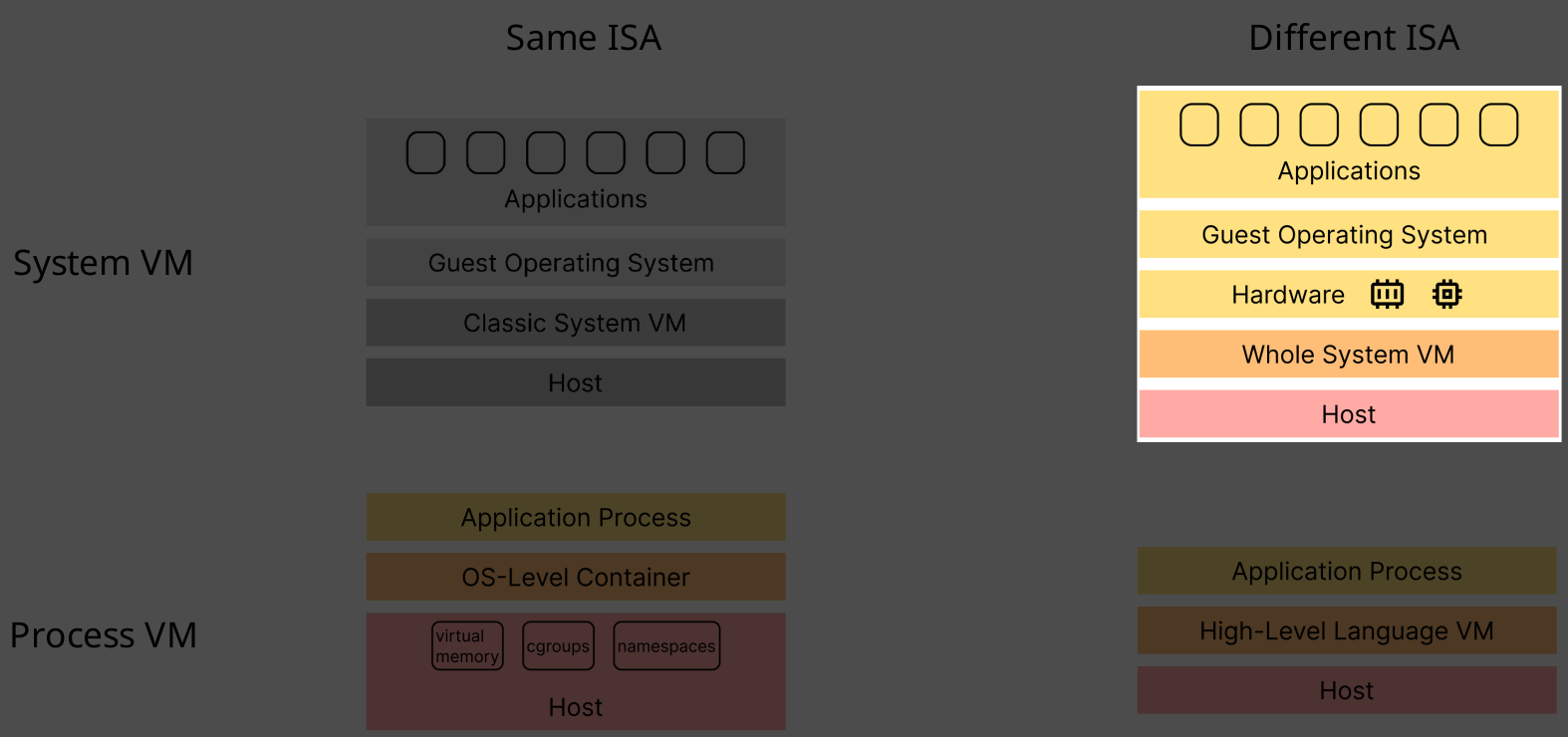
Virtualization Categories



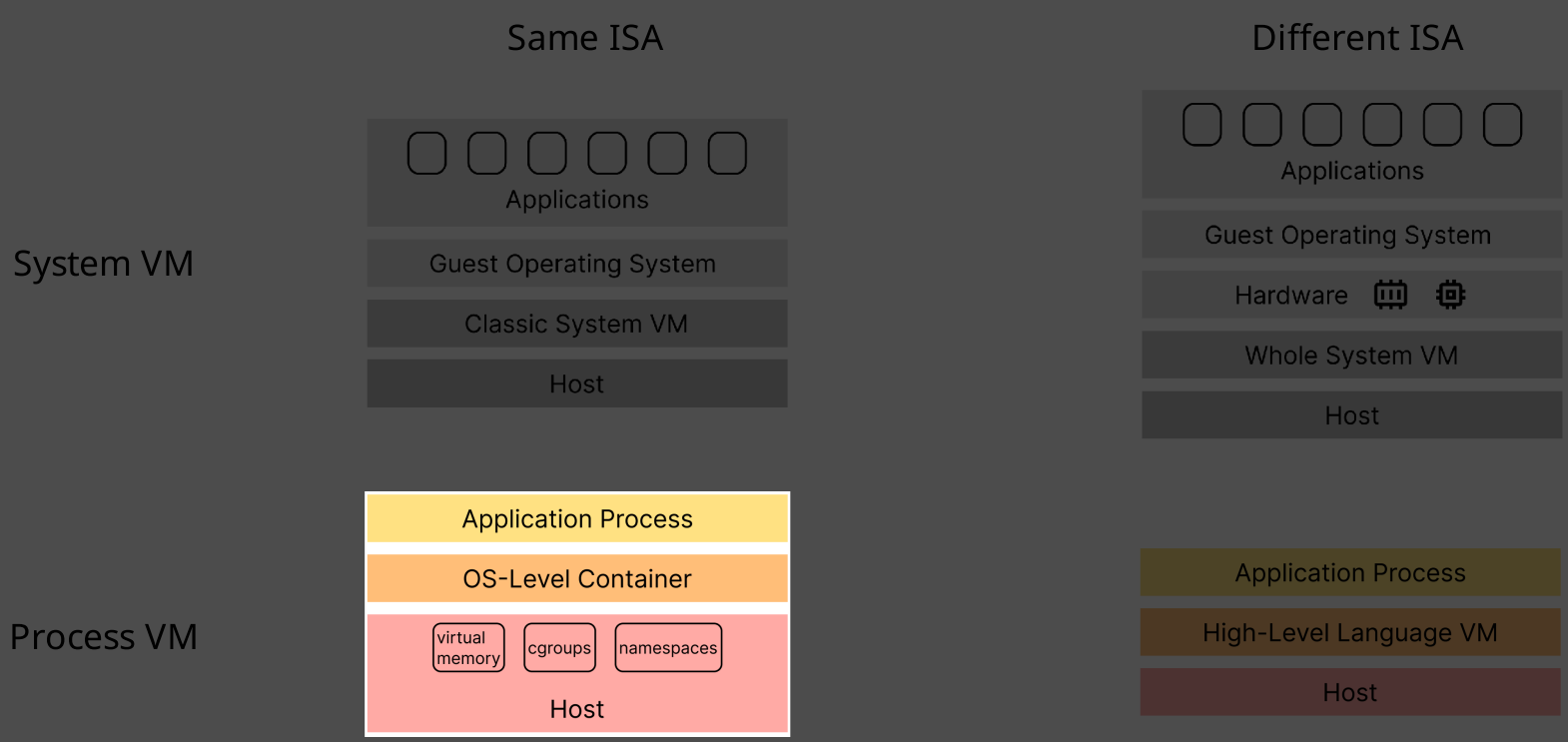
Virtualization Categories



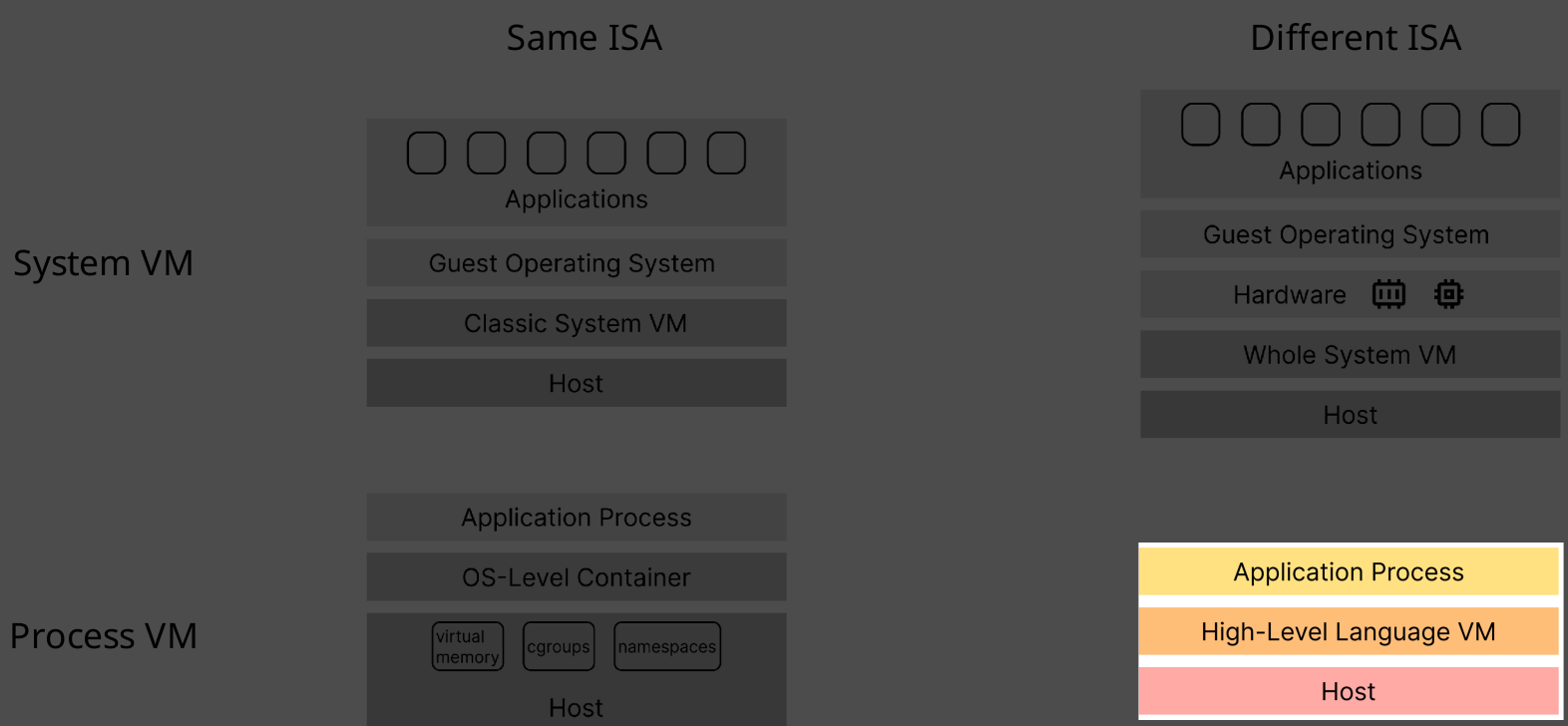
Virtualization Categories



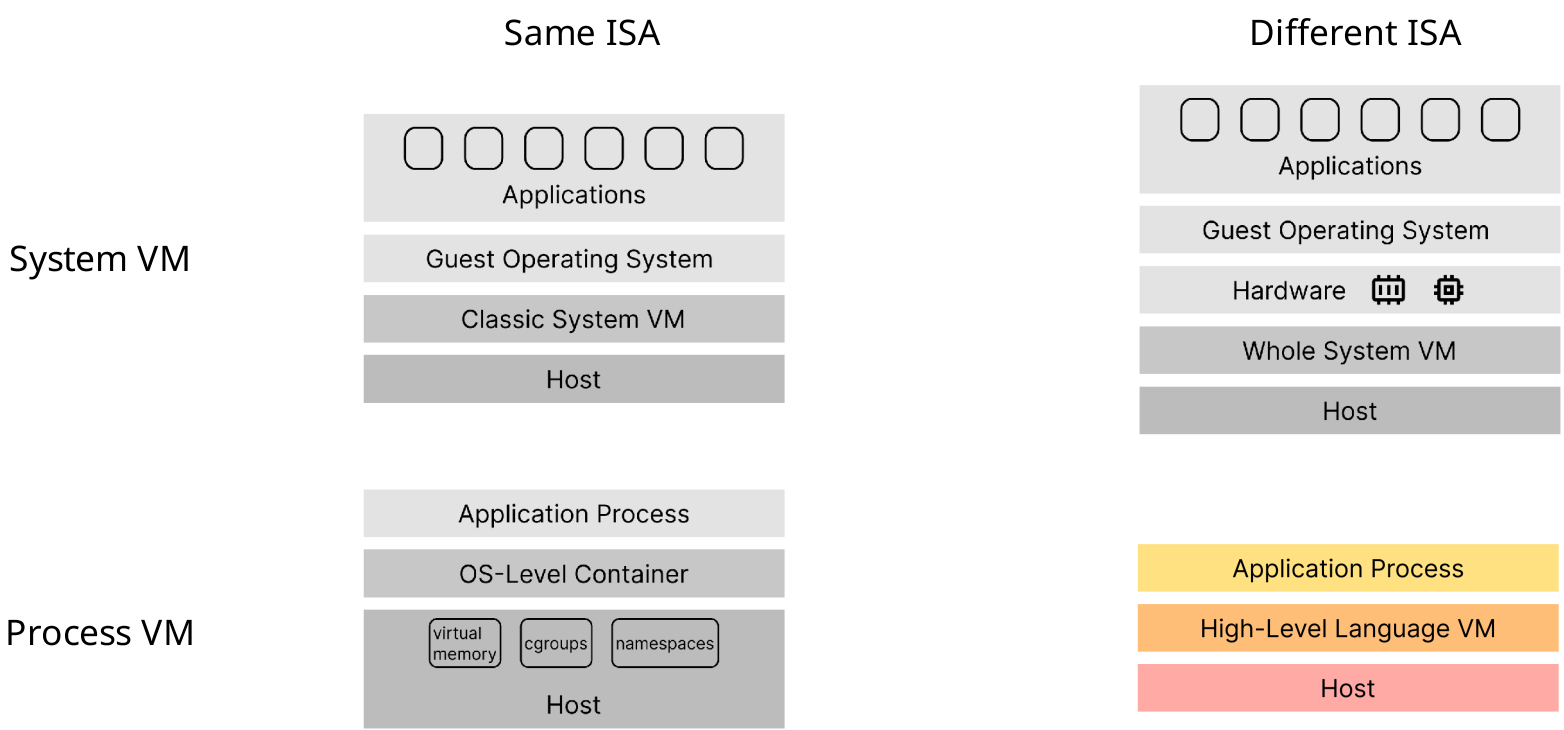
Virtualization Categories



Virtualization Categories



Virtualization Categories



Qualitative Comparison

1. Feature Categories
2. Summary Table

Feature Categories

Development

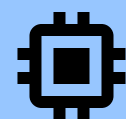
- Automatic Memory Management
- ...

Operational

- Updates At Runtime
- ...

Security

- Address Space Isolation
- ...



Compilation Modes

- JIT Compilation
- ...

Qualitative Comparison Summary

	Development					Operation			Security						Compilation	
Virtualization Technology	Automatic Memory Management	Portability of the Executable	Hardware Abstraction Layer	Runtime Portability	On-board Debugging	Updates at Runtime	Remote Application Management	Remote Monitoring	Address Space Isolation	Address Space Configuration	Memory Usage Restriction	Memory Usage Configuration	CPU Usage Restriction	CPU Usage Configuration	JIT Compilation	AOT Compilation
Cape VM	✓	✓				✓			✓		✓	✓	✓			✓
Femto-Container		✓				✓			✓	✓	✓		✓			
JerryScript	✓	✓		✓	✓	✓			✓							
Lua	✓	✓		✓	✓	✓			✓		✓	✓				
MicroPython	✓	✓	✓	✓		✓										
nanoframework	✓	✓	✓	✓	✓	✓			✓				✓			
Toit	✓	✓	✓			✓	✓	✓	✓							
Velox VM	✓	✓	✓	✓		✓			✓	✓	✓	✓	✓	✓		
WAMR		✓	✓	✓		✓	✓		✓	✓	✓	✓				✓
WARDuino		✓	✓		✓	✓			✓	✓	✓	✓				
Wasmico		✓				✓	✓	✓	✓	✓	✓	✓	✓			
µBPF		✓				✓	✓		✓	✓	✓		✓			✓

Qualitative Comparison Summary

Universal Features:

- Portability of the Executable
- Updates at Runtime
- Address Space Isolation, Except for Micropython

	Development					Operation			Security						Compilation	
Virtualization Technology	Automatic Memory Management	Portability of the Executable	Hardware Abstraction Layer	Runtime Portability	On-board Debugging	Updates at Runtime	Remote Application Management	Remote Monitoring	Address Space Isolation	Address Space Configuration	Memory Usage Restriction	Memory Usage Configuration	CPU Usage Restriction	CPU Usage Configuration	JIT Compilation	AOT Compilation
Cape VM	✓	✓				✓			✓		✓	✓	✓			✓
Femto-Container		✓				✓			✓	✓	✓		✓			
JerryScript	✓	✓		✓	✓	✓			✓							
Lua	✓	✓		✓	✓	✓			✓		✓	✓				
MicroPython	✓	✓	✓	✓		✓										
nanoframework	✓	✓	✓	✓	✓	✓			✓				✓			
Toit	✓	✓	✓			✓	✓	✓	✓							
Velox VM	✓	✓	✓	✓		✓			✓	✓	✓	✓	✓	✓		
WAMR		✓	✓	✓		✓	✓		✓	✓	✓	✓				✓
WARDuino		✓	✓		✓	✓			✓	✓	✓	✓				
Wasmico		✓				✓	✓	✓	✓	✓	✓	✓	✓			
µBPF		✓				✓	✓		✓	✓	✓		✓			✓

Qualitative Comparison Summary

Universal Features:

- Portability of the Executable
- Updates at Runtime
- Address Space Isolation, Except for Micropython

	Development					Operation			Security						Compilation	
Virtualization Technology	Automatic Memory Management	Portability of the Executable	Hardware Abstraction Layer	Runtime Portability	On-board Debugging	Updates at Runtime	Remote Application Management	Remote Monitoring	Address Space Isolation	Address Space Configuration	Memory Usage Restriction	Memory Usage Configuration	CPU Usage Restriction	CPU Usage Configuration	JIT Compilation	AOT Compilation
Cape VM	✓	✓				✓			✓		✓	✓	✓			✓
Femto-Container		✓				✓			✓	✓	✓		✓			
JerryScript	✓	✓		✓	✓	✓			✓							
Lua	✓	✓		✓	✓	✓			✓		✓	✓				
MicroPython	✓	✓	✓	✓		✓										
nanoframework	✓	✓	✓	✓	✓	✓			✓				✓			
Toit	✓	✓	✓			✓	✓	✓	✓							
Velox VM	✓	✓	✓	✓		✓			✓	✓	✓	✓	✓	✓		
WAMR		✓	✓	✓		✓	✓		✓	✓	✓	✓				✓
WARDuino		✓	✓		✓	✓			✓	✓	✓	✓				
Wasmico		✓				✓	✓	✓	✓	✓	✓	✓	✓			
µBPF		✓				✓	✓		✓	✓	✓		✓			✓

Qualitative Comparison Summary

Universal Features:

- Portability of the Executable
- Updates at Runtime
- Address Space Isolation, Except for Micropython

	Development					Operation			Security						Compilation	
Virtualization Technology	Automatic Memory Management	Portability of the Executable	Hardware Abstraction Layer	Runtime Portability	On-board Debugging	Updates at Runtime	Remote Application Management	Remote Monitoring	Address Space Isolation	Address Space Configuration	Memory Usage Restriction	Memory Usage Configuration	CPU Usage Restriction	CPU Usage Configuration	JIT Compilation	AOT Compilation
Cape VM	✓	✓				✓			✓		✓	✓	✓			✓
Femto-Container		✓				✓			✓	✓	✓		✓			
JerryScript	✓	✓		✓	✓	✓			✓							
Lua	✓	✓		✓	✓	✓			✓		✓	✓				
MicroPython	✓	✓	✓	✓		✓										
nanoframework	✓	✓	✓	✓	✓	✓			✓				✓			
Toit	✓	✓	✓			✓	✓	✓	✓							
Velox VM	✓	✓	✓	✓		✓			✓	✓	✓	✓	✓	✓		
WAMR		✓	✓	✓		✓	✓		✓	✓	✓	✓				✓
WARDuino		✓	✓		✓	✓			✓	✓	✓	✓				
Wasmico		✓				✓	✓	✓	✓	✓	✓	✓	✓			
µBPF		✓				✓	✓		✓	✓	✓		✓			✓

Qualitative Comparison Summary

Universal Features:

- Portability of the Executable
- Updates at Runtime
- Address Space Isolation, Except for Micropython

	Development					Operation			Security						Compilation	
Virtualization Technology	Automatic Memory Management	Portability of the Executable	Hardware Abstraction Layer	Runtime Portability	On-board Debugging	Updates at Runtime	Remote Application Management	Remote Monitoring	Address Space Isolation	Address Space Configuration	Memory Usage Restriction	Memory Usage Configuration	CPU Usage Restriction	CPU Usage Configuration	JIT Compilation	AOT Compilation
Cape VM	✓	✓				✓			✓		✓	✓	✓			✓
Femto-Container		✓				✓			✓	✓	✓		✓			
JerryScript	✓	✓		✓	✓	✓			✓							
Lua	✓	✓		✓	✓	✓			✓		✓	✓				
MicroPython	✓	✓	✓	✓		✓										
nanoframework	✓	✓	✓	✓	✓	✓			✓				✓			
Toit	✓	✓	✓			✓	✓	✓	✓							
Velox VM	✓	✓	✓	✓		✓			✓	✓	✓	✓	✓	✓		
WAMR		✓	✓	✓		✓	✓		✓	✓	✓	✓				✓
WARDuino		✓	✓		✓	✓			✓	✓	✓	✓				
Wasmico		✓				✓	✓	✓	✓	✓	✓	✓	✓			
µBPF		✓				✓	✓		✓	✓	✓		✓			✓

Qualitative Comparison Summary

Universal Features:

- Portability of the Executable
- Updates at Runtime
- Address Space Isolation, Except for Micropython

Three Solutions suppore **AOT** Compilation
However, none support **JIT** Compilation

	Development					Operation			Security						Compilation	
Virtualization Technology	Automatic Memory Management	Portability of the Executable	Hardware Abstraction Layer	Runtime Portability	On-board Debugging	Updates at Runtime	Remote Application Management	Remote Monitoring	Address Space Isolation	Address Space Configuration	Memory Usage Restriction	Memory Usage Configuration	CPU Usage Restriction	CPU Usage Configuration	JIT Compilation	AOT Compilation
Cape VM	✓	✓				✓			✓		✓	✓	✓			✓
Femto-Container		✓				✓			✓	✓	✓		✓			
JerryScript	✓	✓		✓	✓	✓			✓							
Lua	✓	✓		✓	✓	✓			✓		✓	✓				
MicroPython	✓	✓	✓	✓		✓										
nanoframework	✓	✓	✓	✓	✓	✓			✓				✓			
Toit	✓	✓	✓			✓	✓	✓	✓							
Velox VM	✓	✓	✓	✓		✓			✓	✓	✓	✓	✓	✓		
WAMR		✓	✓	✓		✓	✓		✓	✓	✓	✓				✓
WARDuino		✓	✓		✓	✓			✓	✓	✓	✓				
Wasmico		✓				✓	✓	✓	✓	✓	✓	✓	✓			
µBPF		✓				✓	✓		✓	✓	✓		✓			✓

Qualitative Comparison Summary

Universal Features:

- Portability of the Executable
- Updates at Runtime
- Address Space Isolation, Except for Micropython

Three Solutions suppore **AOT** Compilation
However, none support **JIT** Compilation

	Development					Operation			Security						Compilation	
Virtualization Technology	Automatic Memory Management	Portability of the Executable	Hardware Abstraction Layer	Runtime Portability	On-board Debugging	Updates at Runtime	Remote Application Management	Remote Monitoring	Address Space Isolation	Address Space Configuration	Memory Usage Restriction	Memory Usage Configuration	CPU Usage Restriction	CPU Usage Configuration	JIT Compilation	AOT Compilation
Cape VM	✓	✓				✓			✓		✓	✓	✓			✓
Femto-Container		✓				✓			✓	✓	✓		✓			
JerryScript	✓	✓		✓	✓	✓			✓							
Lua	✓	✓		✓	✓	✓			✓		✓	✓				
MicroPython	✓	✓	✓	✓		✓										
nanoframework	✓	✓	✓	✓	✓	✓			✓				✓			
Toit	✓	✓	✓			✓	✓	✓	✓							
Velox VM	✓	✓	✓	✓		✓			✓	✓	✓	✓	✓	✓		
WAMR		✓	✓	✓		✓	✓		✓	✓	✓	✓				✓
WARDuino		✓	✓		✓	✓			✓	✓	✓	✓				
Wasmico		✓				✓	✓	✓	✓	✓	✓	✓	✓			
µBPF		✓				✓	✓		✓	✓	✓		✓			✓

Qualitative Comparison Summary

Universal Features:

- Portability of the Executable
- Updates at Runtime
- Address Space Isolation, Except for Micropython

Three Solutions suppore **AOT** Compilation
However, none support **JIT** Compilation

	Development					Operation			Security						Compilation	
Virtualization Technology	Automatic Memory Management	Portability of the Executable	Hardware Abstraction Layer	Runtime Portability	On-board Debugging	Updates at Runtime	Remote Application Management	Remote Monitoring	Address Space Isolation	Address Space Configuration	Memory Usage Restriction	Memory Usage Configuration	CPU Usage Restriction	CPU Usage Configuration	JIT Compilation	AOT Compilation
Femto-Container		✓				✓			✓	✓	✓		✓			
JerryScript	✓	✓		✓	✓	✓			✓							
Lua	✓	✓		✓	✓	✓			✓		✓	✓				
MicroPython	✓	✓	✓	✓		✓										
WAMR		✓	✓	✓		✓	✓		✓	✓	✓	✓				✓
µBPF		✓				✓	✓		✓	✓	✓		✓			✓

Empirical Evaluation on RIOT OS

1. Candidate Introduction
2. Classification
3. HLL VM Architecture
4. Experiment
    1. Design
    2. Implementation
    3. Results

Candidates

JerryScript

Micropython

Lua

Femto-Container

µBPF

Web**A**ssembly **M**icro **R**untime

Candidates Classification

		Architectural Properties			Feature Counts			
	HLL VM	**Garbage Collection**	**Typing**	**Bytecode Compilation**	**Development**	**Operation**	**Security**	**Compilation**
Dynamic VMs	JerryScript	Yes	Dynamic	On-device	**4/5**	**1/3**	**1/6**	**0/2**
	MicroPython	Yes	Dynamic	On-Device	**4/5**	**1/3**	**0/6**	**0/2**
	Lua	Yes	Dynamic	On-Device	**4/5**	**1/3**	**3/6**	**0/2**
Static VMs	WAMR	No	Static	Off-Device	**3/5**	**2/3**	**4/6**	**1/2**
	WAMR (fast)	No	Static	Off-Device	**3/5**	**2/3**	**4/6**	**1/2**
	Femto-Container	No	Static	Off-Device	**1/5**	**1/3**	**4/6**	**0/2**
	µBPF	No	Static	Off-Device	**1/5**	**2/3**	**4/6**	**1/2**

Candidates Classification

		Architectural Properties			Feature Counts			
	HLL VM	Garbage Collection	Typing	Bytecode Compilation	Development	Operation	Security	Compilation
Dynamic VMs	JerryScript	Yes	Dynamic	On-device	4/5	1/3	1/6	0/2
	MicroPython	Yes	Dynamic	On-Device	4/5	1/3	0/6	0/2
	Lua	Yes	Dynamic	On-Device	4/5	1/3	3/6	0/2
Static VMs	WAMR	No	Static	Off-Device	3/5	2/3	4/6	1/2
	WAMR (fast)	No	Static	Off-Device	3/5	2/3	4/6	1/2
	Femto-Container	No	Static	Off-Device	1/5	1/3	4/6	0/2
	µBPF	No	Static	Off-Device	1/5	2/3	4/6	1/2

Candidates Classification

		Architectural Properties			Feature Counts			
	HLL VM	Garbage Collection	Typing	Bytecode Compilation	Development	Operation	Security	Compilation
Dynamic VMs	JerryScript	Yes	Dynamic	On-device	4/5	1/3	1/6	0/2
	MicroPython	Yes	Dynamic	On-Device	4/5	1/3	0/6	0/2
	Lua	Yes	Dynamic	On-Device	4/5	1/3	3/6	0/2
Static VMs	WAMR	No	Static	Off-Device	3/5	2/3	4/6	1/2
	WAMR (fast)	No	Static	Off-Device	3/5	2/3	4/6	1/2
	Femto-Container	No	Static	Off-Device	1/5	1/3	4/6	0/2
	µBPF	No	Static	Off-Device	1/5	2/3	4/6	1/2

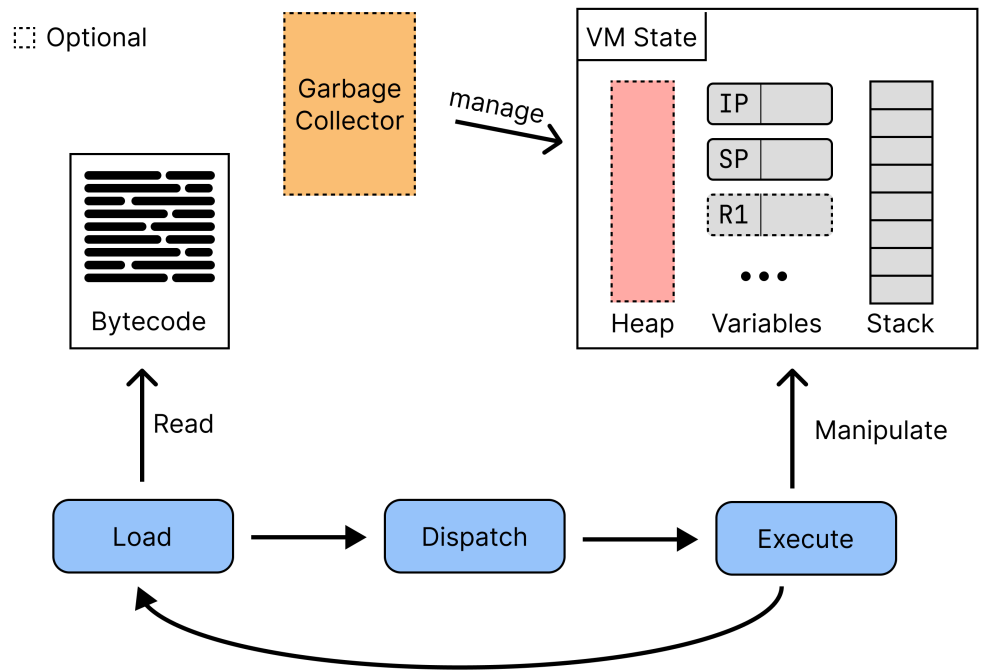
Candidates Classification

		Architectural Properties			Feature Counts			
	HLL VM	Garbage Collection	Typing	Bytecode Compilation	Development	Operation	Security	Compilation
Dynamic VMs	JerryScript	Yes	Dynamic	On-device	4/5	1/3	1/6	0/2
	MicroPython	Yes	Dynamic	On-Device	4/5	1/3	0/6	0/2
	Lua	Yes	Dynamic	On-Device	4/5	1/3	3/6	0/2
Static VMs	WAMR	No	Static	Off-Device	3/5	2/3	4/6	1/2
	WAMR (fast)	No	Static	Off-Device	3/5	2/3	4/6	1/2
	Femto-Container	No	Static	Off-Device	1/5	1/3	4/6	0/2
	µBPF	No	Static	Off-Device	1/5	2/3	4/6	1/2

Candidates Classification

		Architectural Properties			Feature Counts			
	HLL VM	**Garbage Collection**	**Typing**	**Bytecode Compilation**	**Development**	**Operation**	**Security**	**Compilation**
Dynamic VMs	JerryScript	Yes	Dynamic	On-device	**4/5**	**1/3**	**1/6**	**0/2**
	MicroPython	Yes	Dynamic	On-Device	**4/5**	**1/3**	**0/6**	**0/2**
	Lua	Yes	Dynamic	On-Device	**4/5**	**1/3**	**3/6**	**0/2**
Static VMs	WAMR	No	Static	Off-Device	**3/5**	**2/3**	**4/6**	**1/2**
	WAMR (fast)	No	Static	Off-Device	**3/5**	**2/3**	**4/6**	**1/2**
	Femto-Container	No	Static	Off-Device	**1/5**	**1/3**	**4/6**	**0/2**
	µBPF	No	Static	Off-Device	**1/5**	**2/3**	**4/6**	**1/2**

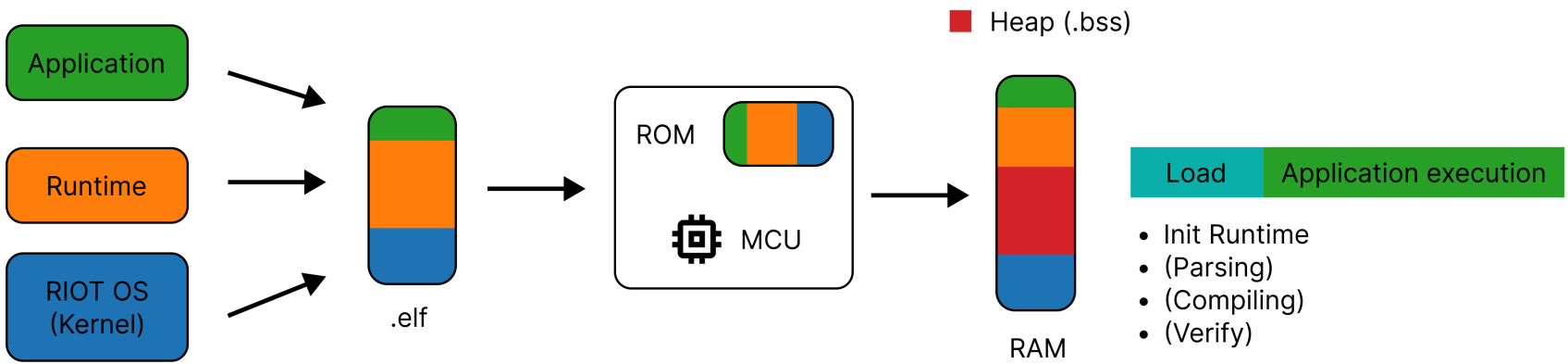
Fundamental HLL VM Architecture



Experiment

1. **Design**
    1. Metrics
    2. Benchmark Programms
2. **Measurement Implementation**
3. **Results**
    1. Load Time
    2. Execution Time
    3. Code Size
    4. ROM footprint
    5. RAM usage

Design: Metrics



Compilation
<>Code Size

Flash on Device
ROM footprint

Execution
RAM usage
Load Time
Execution Time

Design: Benchmark Programms

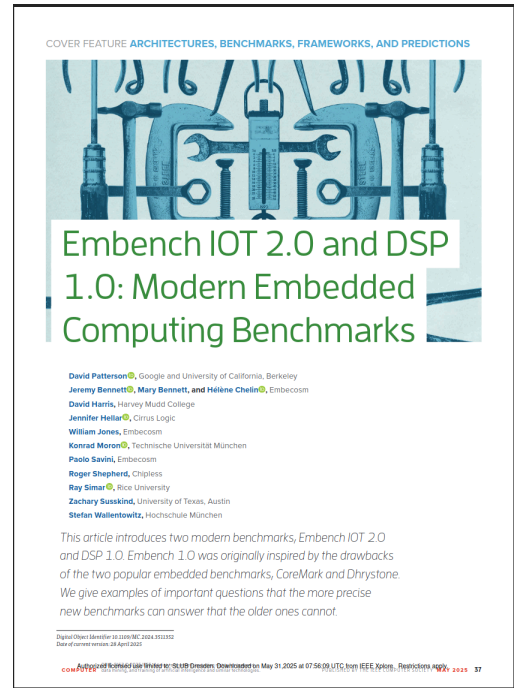
Subset of Embench IoT 2.0

Static Memory

- xgboost
- crc_32
- ud

Dynamic Memory

- md5sum
- tarfind



COVER FEATURE ARCHITECTURES, BENCHMARKS, FRAMEWORKS, AND PREDICTIONS

Embench IOT 2.0 and DSP 1.0: Modern Embedded Computing Benchmarks

David Patterson, Google and University of California, Berkeley
Jeremy Bennett, Mary Bennett, and Hélène Chelin, Embecosm
David Harris, Harvey Mudd College
Jennifer Hellar, Cirrus Logic
William Jones, Embecosm
Konrad Moron, Technische Universität München
Paolo Savini, Embecosm
Roger Shepherd, Chipless
Ray Simar, Rice University
Zachary Susskind, University of Texas, Austin
Stefan Wallentowitz, Hochschule München

This article introduces two modern benchmarks, Embench IOT 2.0 and DSP 1.0. Embench 1.0 was originally inspired by the drawbacks of the two popular embedded benchmarks, CoreMark and Dhrystone. We give examples of important questions that the more precise new benchmarks can answer that the older ones cannot.

Digital Object Identifier 10.1109/MC.2024.3513352
Date of current version: 28 April 2025

Implementation

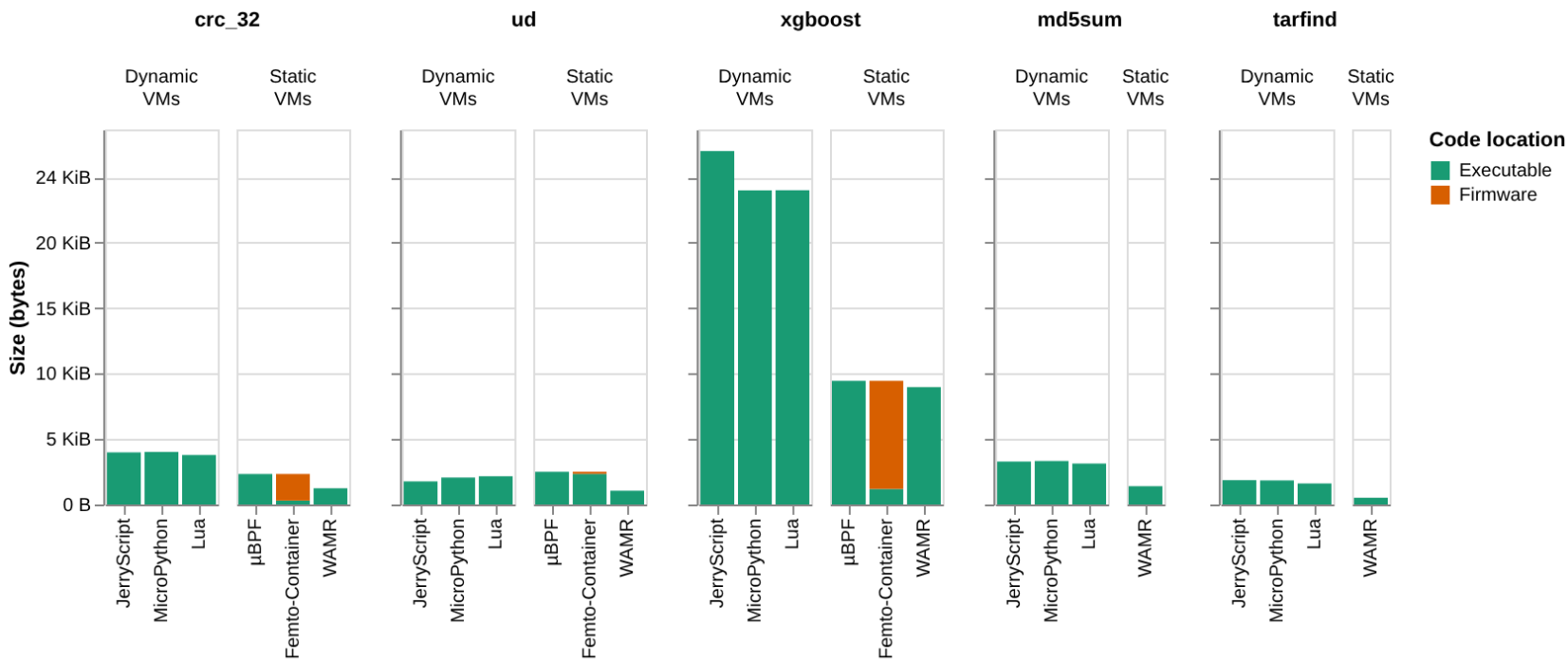
- **Code Size**: file size of the executable
- **Flash footpring**: cosy
- **RAM Usage**: cosy + `malloc_monitor`
- **Load Time + Execution Time**: ztimer

Average of **5** Executions.

Hardware: Adafruit Feather nRF52840 Sense

- CPU: ARM Cortex M4F (64 MHz)
- 1 MB flash
- 256 KB SRAM

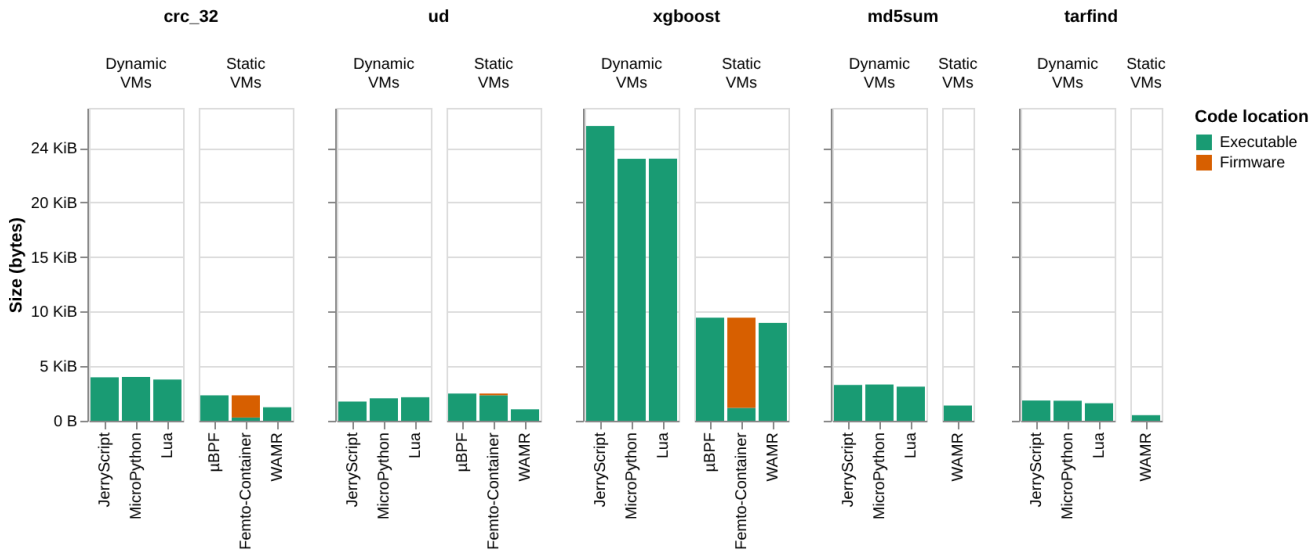
Results: Code Size



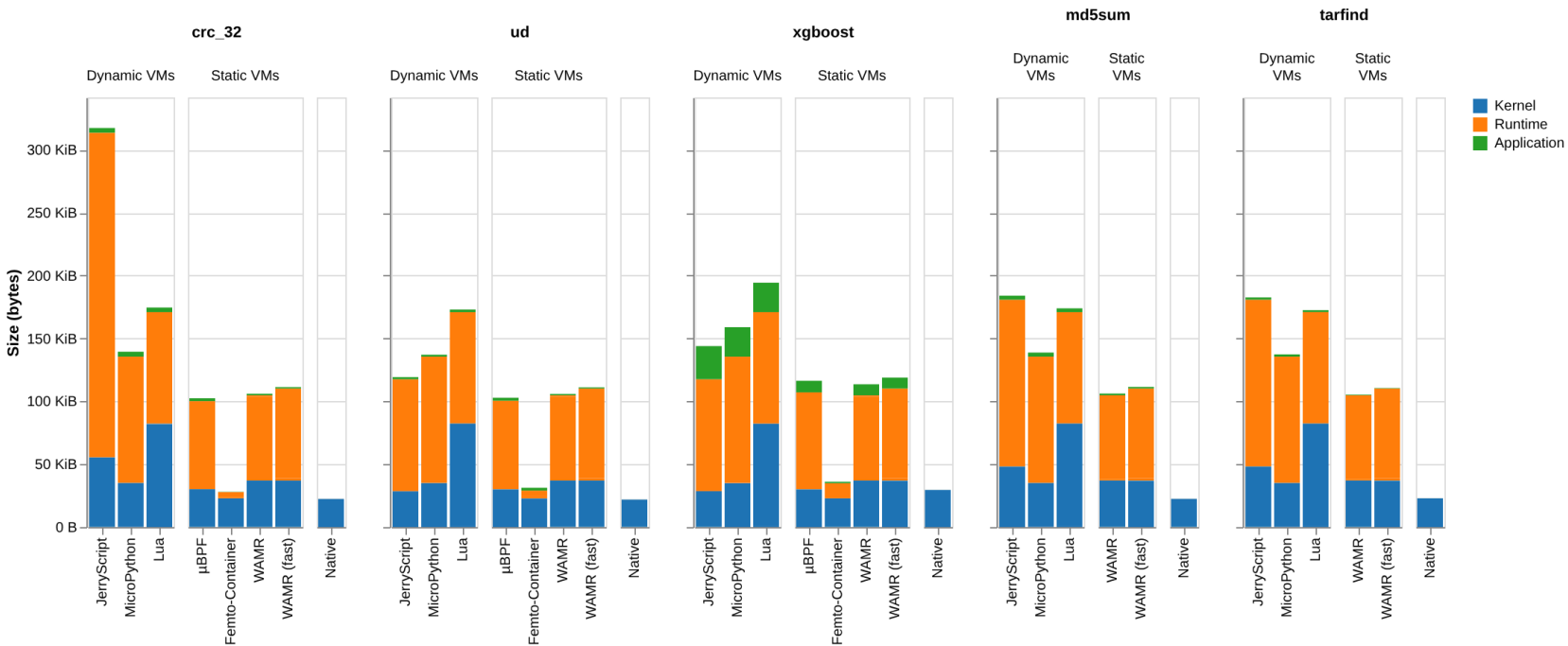
Results: Code Size

Key Insights

- Bytecode encoding more efficient than text based
- `0x77073096` 4 bytes as 32-bit int vs. 10 bytes in ASCII



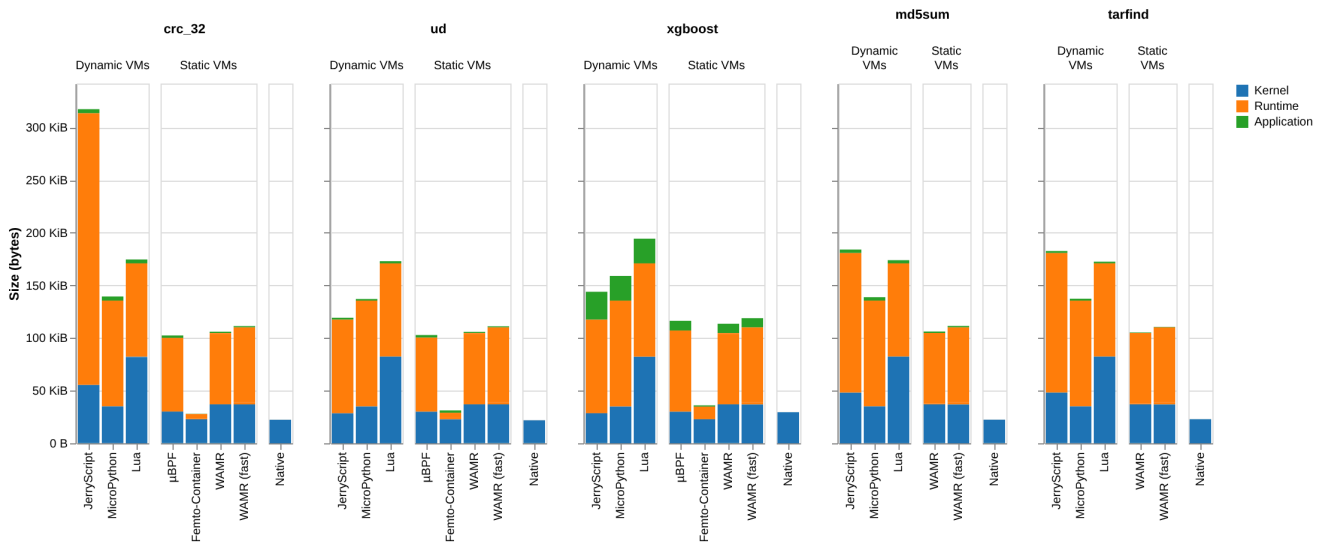
Results: Flash footprint



Results: Flash footprint

Key Insights

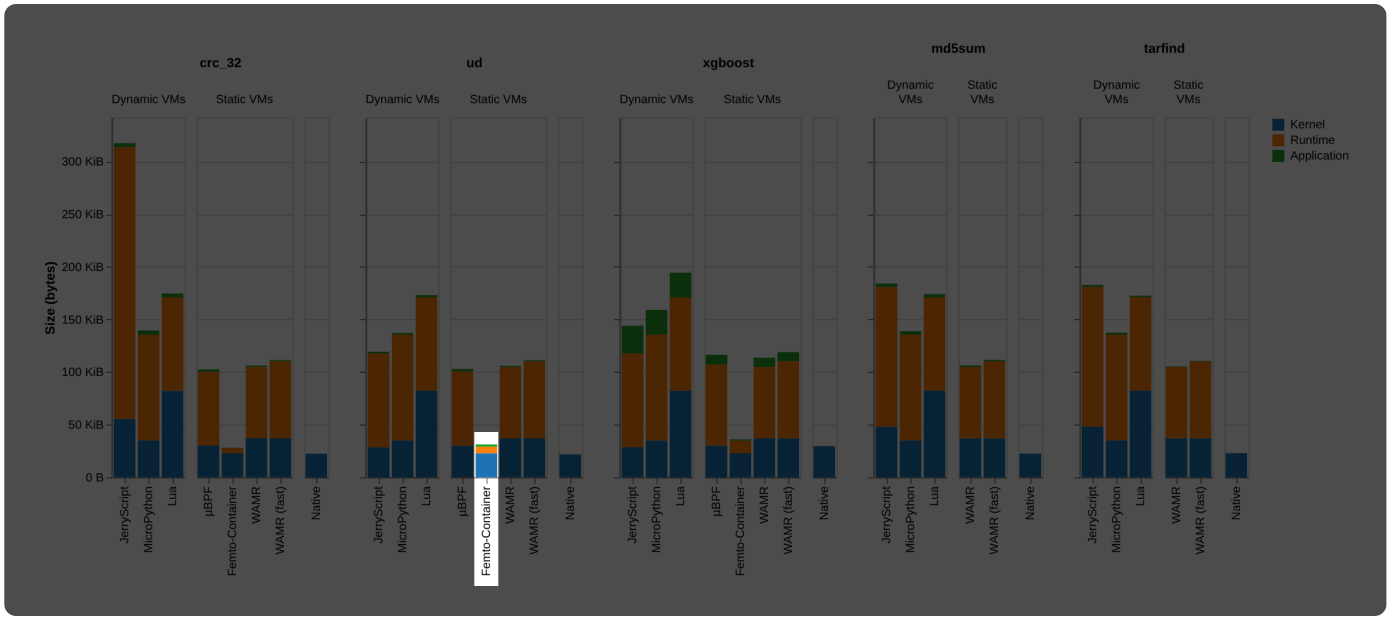
- Dynamic VMs occupy more space
- femto container ~40 Kib lowest footprint



Results: Flash footprint

Key Insights

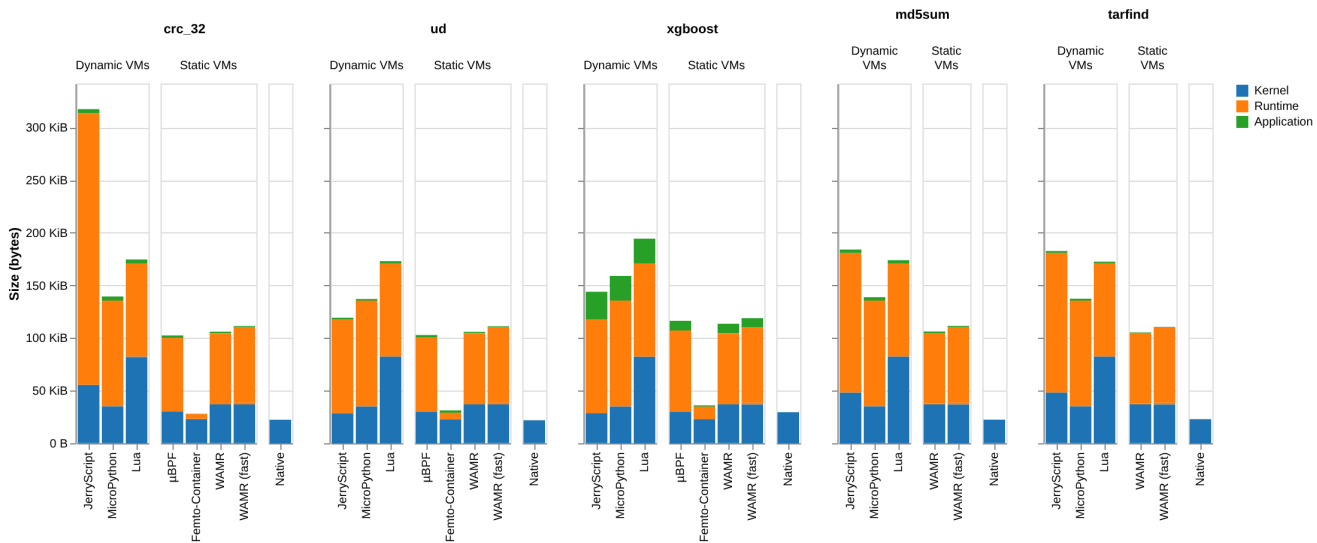
- Dynamic VMs occupy more space
- femto container ~40 Kib lowest footprint



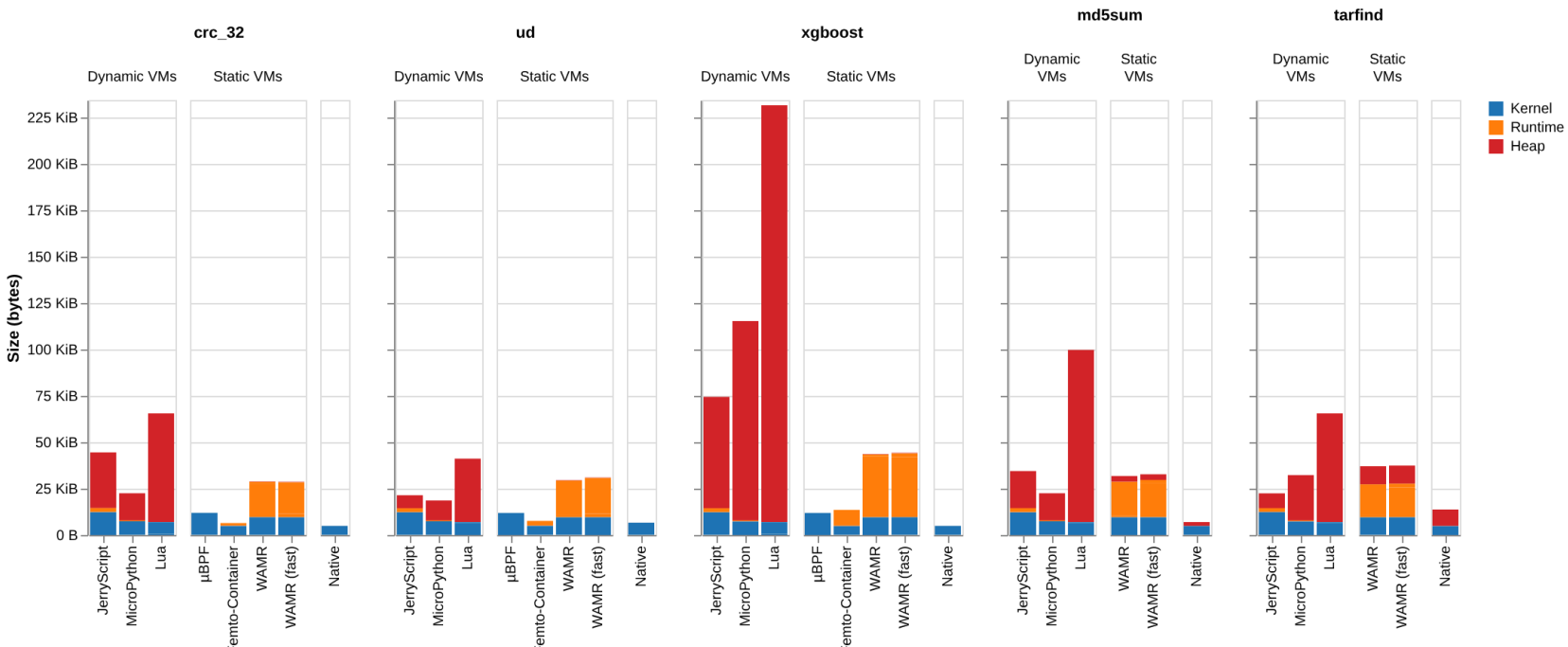
Results: Flash footprint

Key Insights

- Dynamic VMs occupy more space
- femto container ~40 Kib lowest footprint



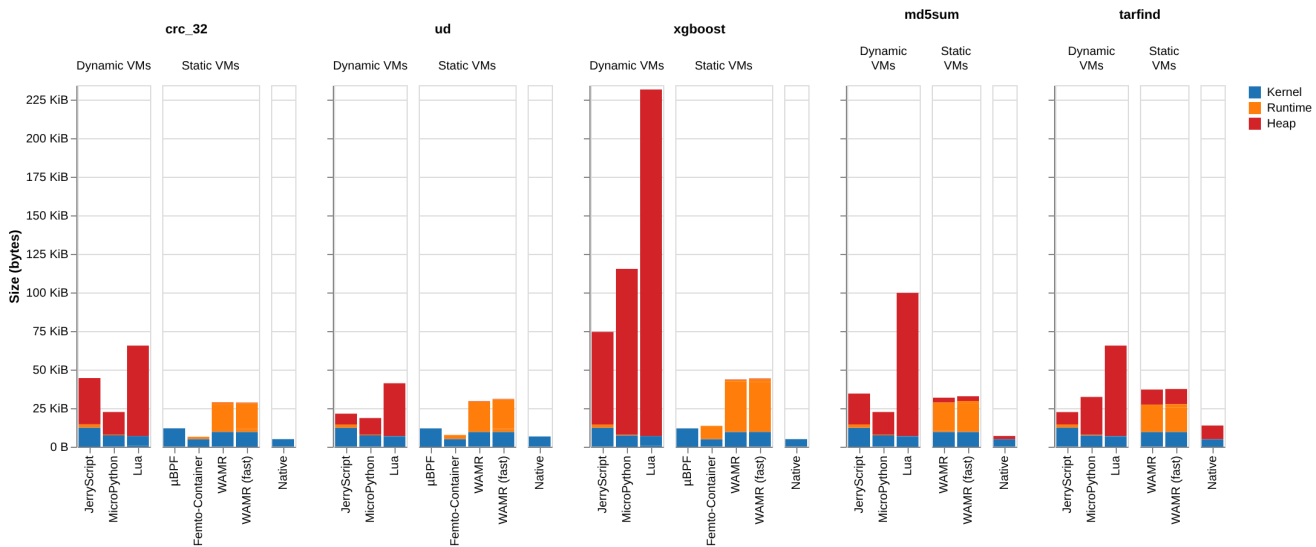
Results: RAM usage



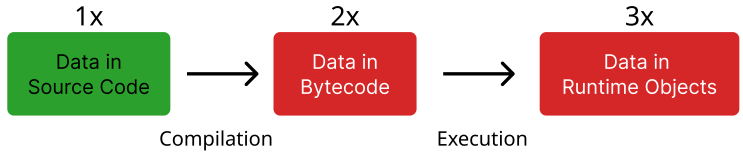
Results: RAM usage

Key Insights

- Dynamic VMs incur highest RAM (data duplication)
- Lua has overall highest footprint

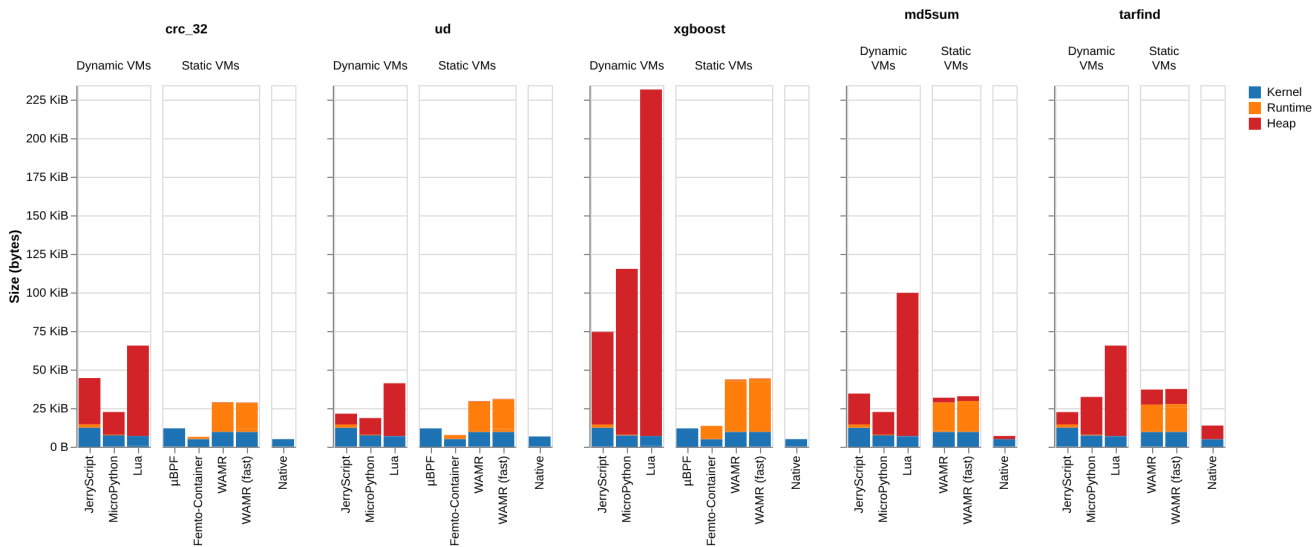


Results: RAM usage



Key Insights

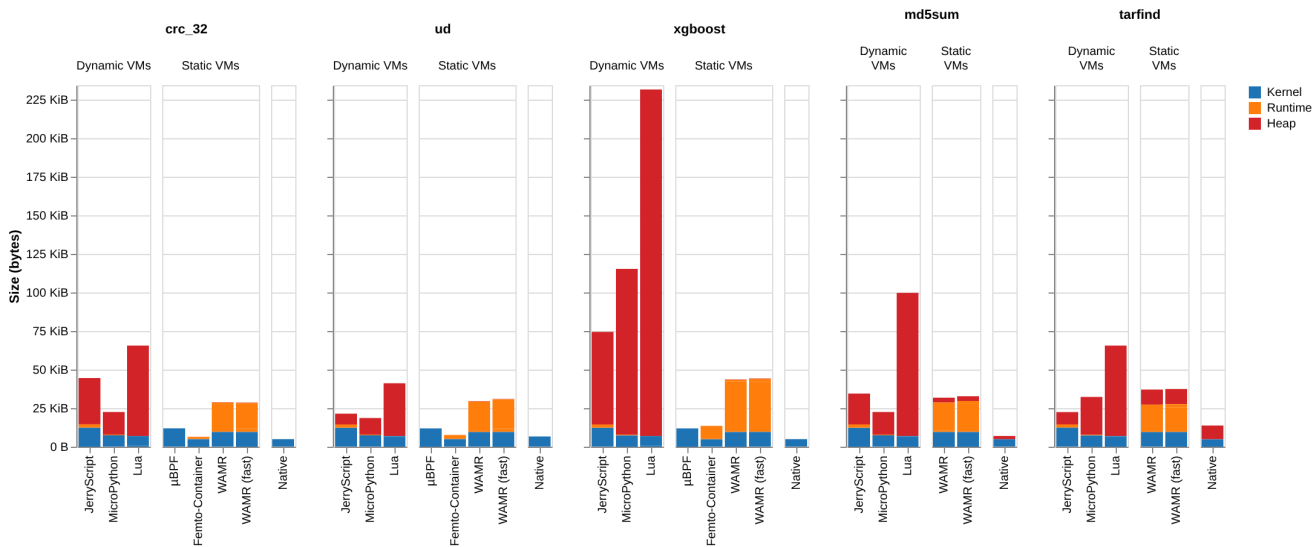
- Dynamic VMs incur highest RAM (data duplication)
- Lua has overall highest footprint



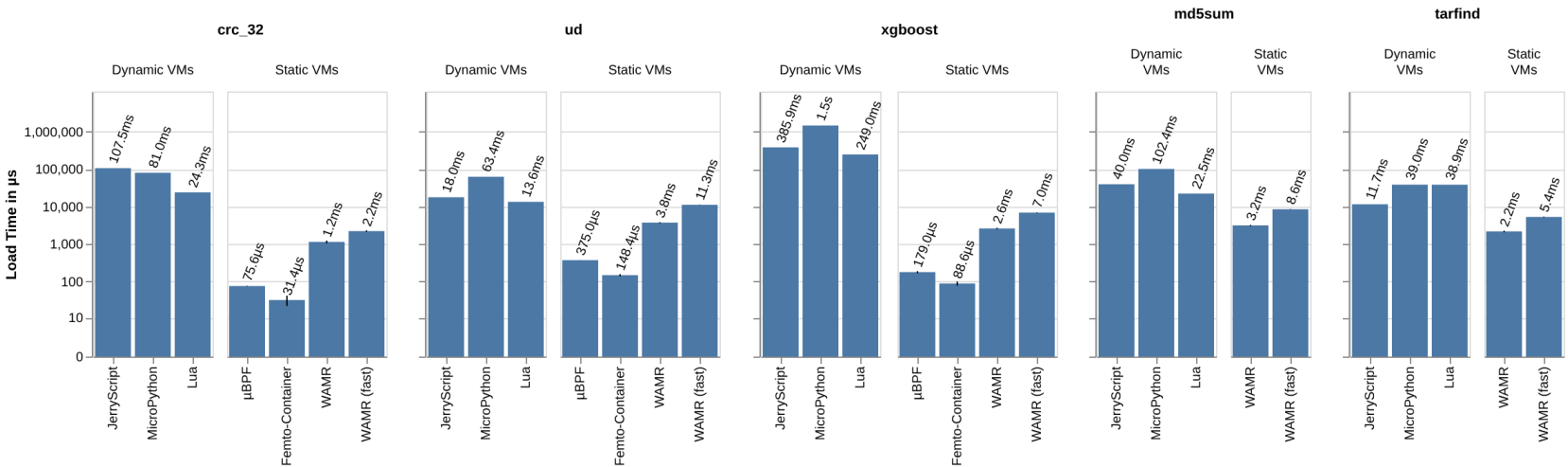
Results: RAM usage

Key Insights

- Dynamic VMs incur highest RAM (data duplication)
- Lua has overall highest footprint



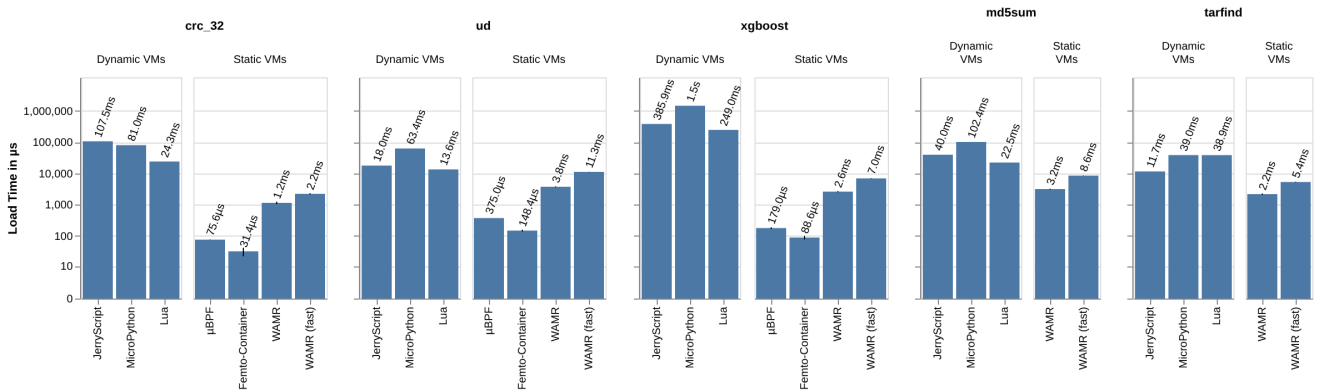
Results: Load Time



Results: Load Time

Key Insights

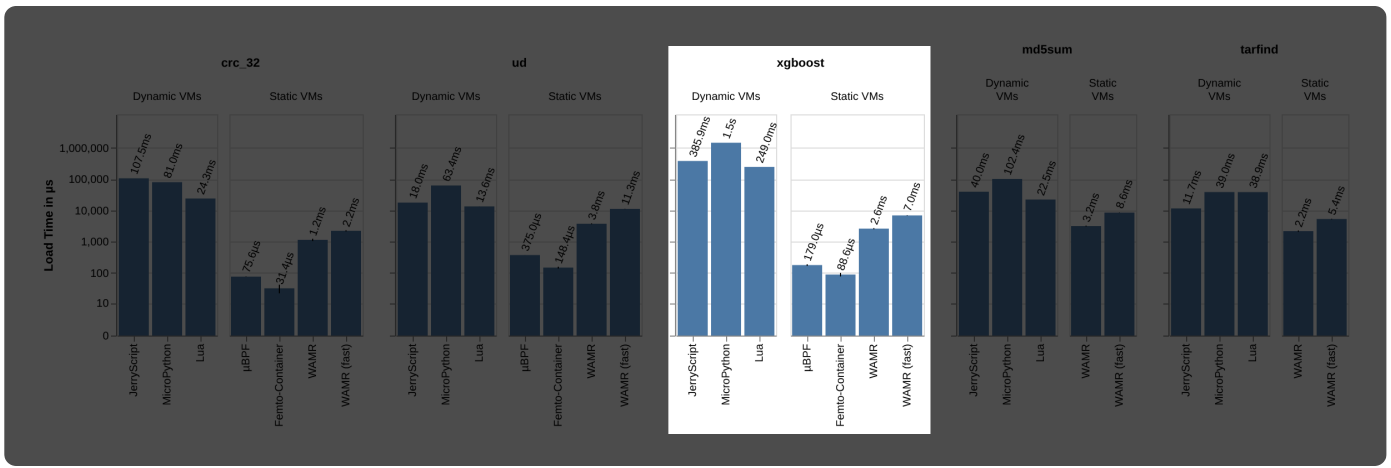
- Load Time of dynamic VMs **orders of magnitude** higher
- **Reason:** Bytecode compilation overhead
- WAMR (fast) longest load time of **static VMs**



Results: Load Time

Key Insights

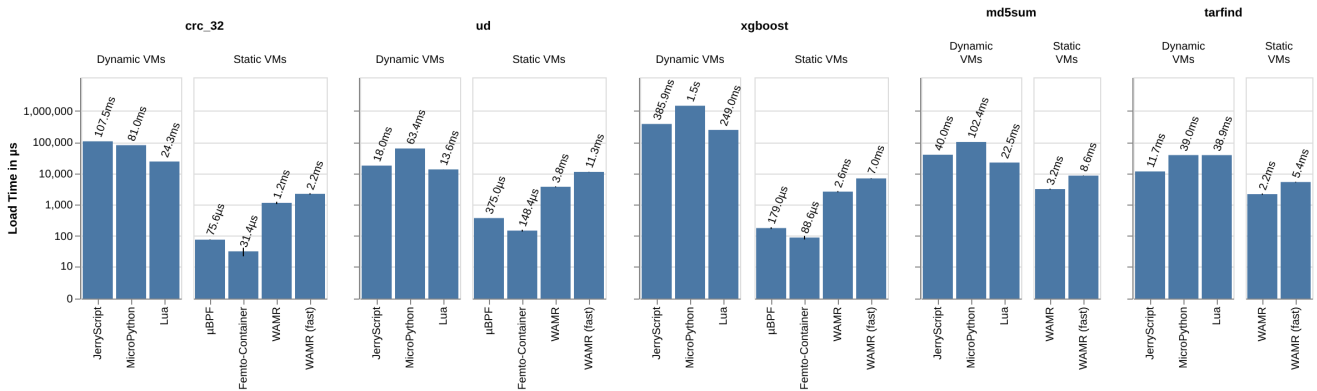
- Load Time of dynamic VMs **orders of magnitude** higher
- **Reason:** Bytecode compilation overhead
- WAMR (fast) longest load time of **static VMs**



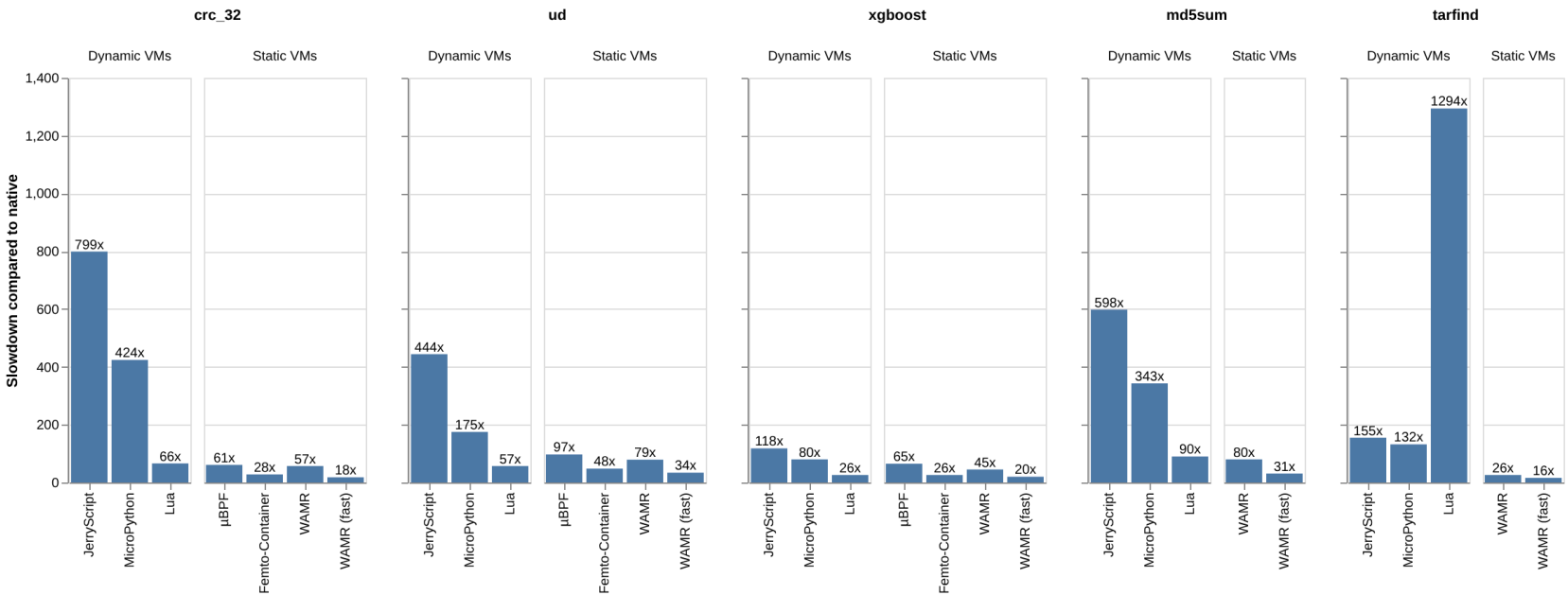
Results: Load Time

Key Insights

- Load Time of dynamic VMs **orders of magnitude** higher
- **Reason:** Bytecode compilation overhead
- WAMR (fast) longest load time of **static VMs**



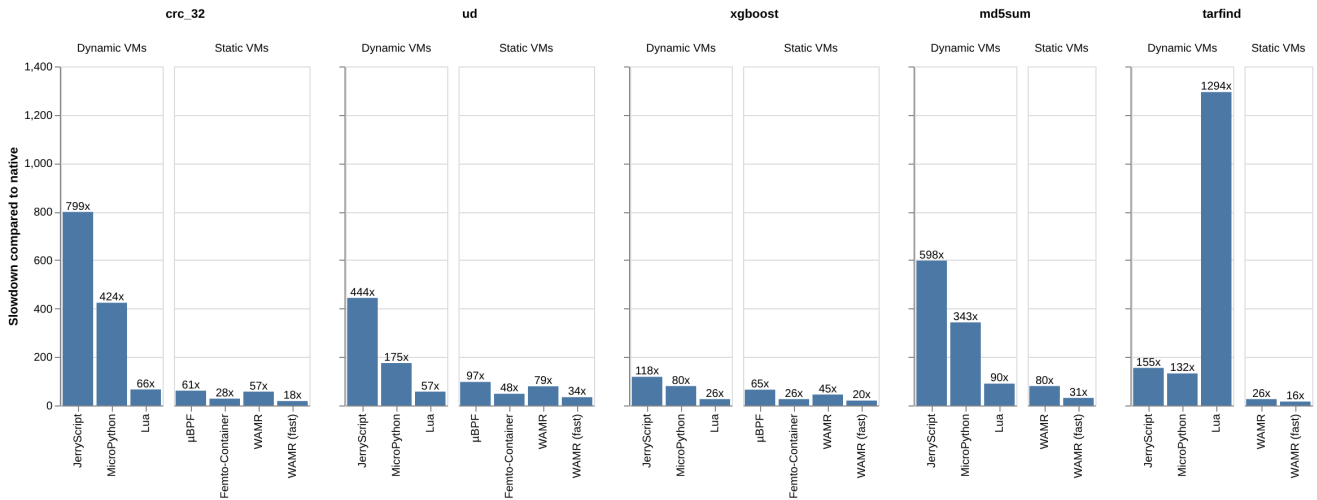
Results: Execution Time



Results: Execution Time

Key Insights

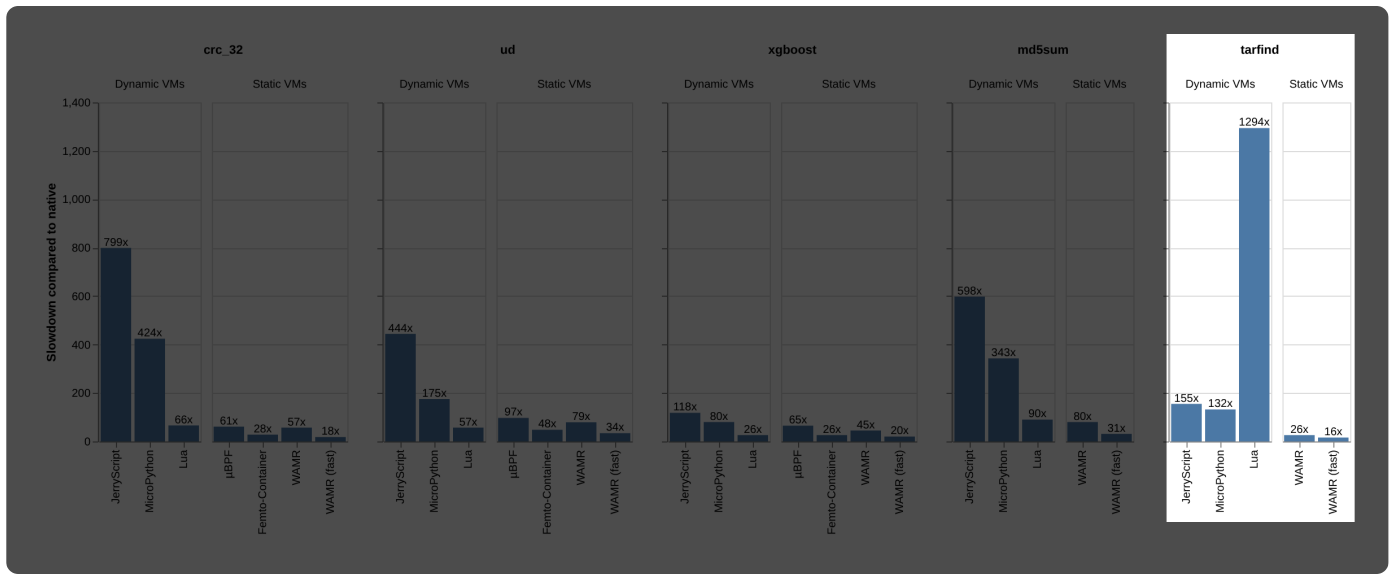
- HLL VMs impose significant slowdown (16x at best)
- WAMR (fast) most efficient (24x avg)
- Dynamic VMs are slower than static VMs
- Lua is the fastest of the dynamic VMs (~60x), except for string operations



Results: Execution Time

Key Insights

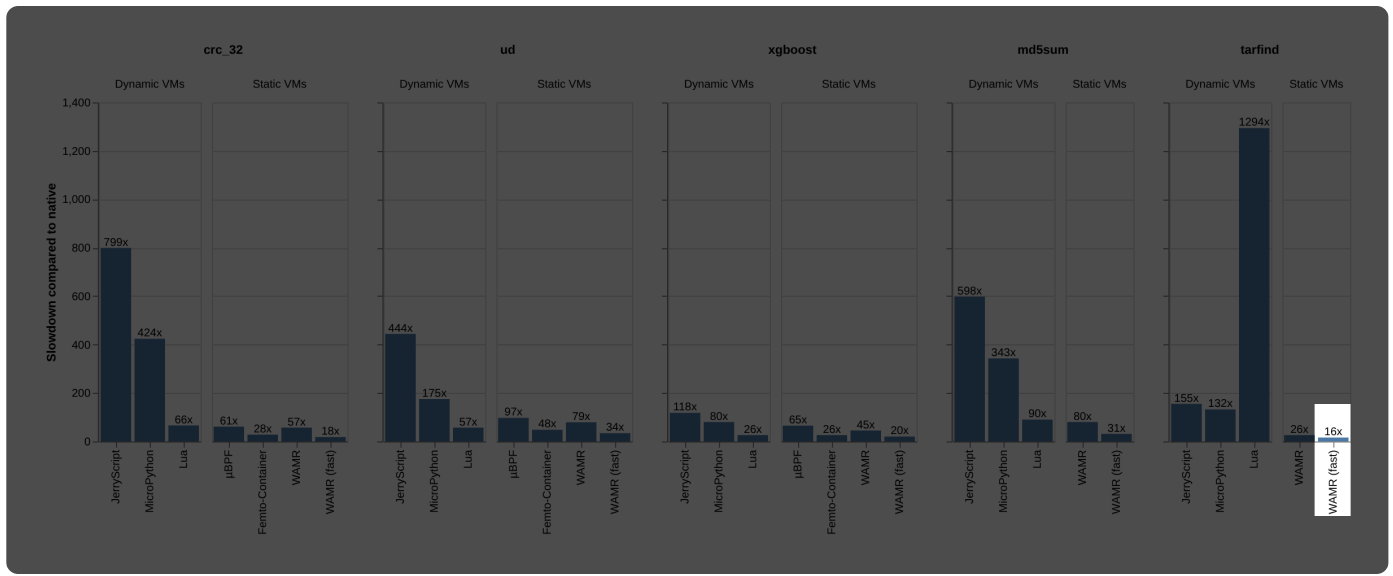
- HLL VMs impose significant slowdown (16x at best)
- WAMR (fast) most efficient (24x avg)
- Dynamic VMs are slower than static VMs
- Lua is the fastest of the dynamic VMs (~60x), except for string operations



Results: Execution Time

Key Insights

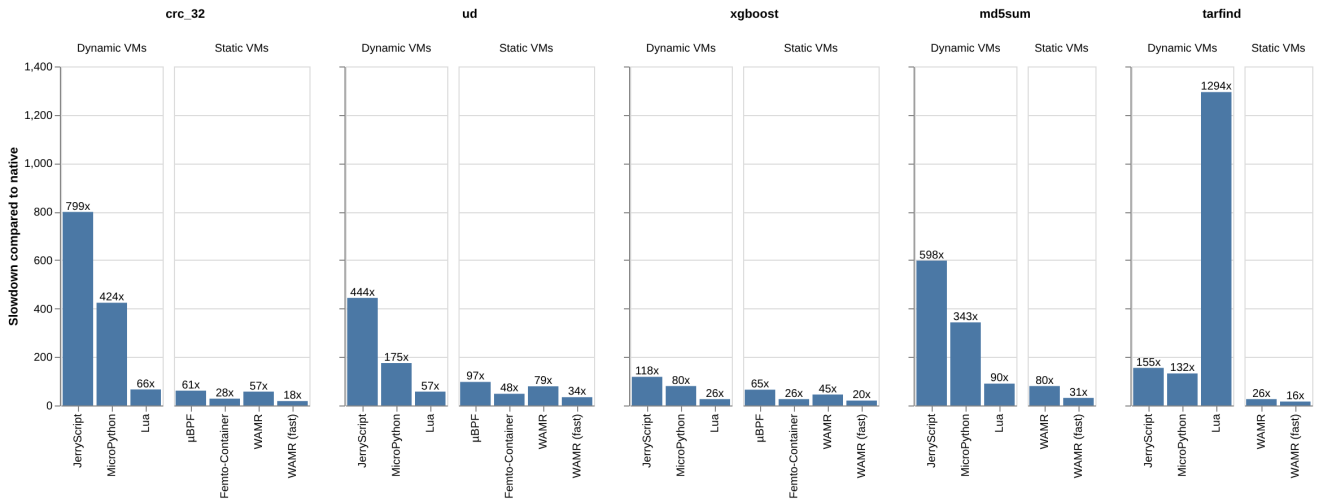
- HLL VMs impose significant slowdown (16x at best)
- WAMR (fast) most efficient (24x avg)
- Dynamic VMs are slower than static VMs
- Lua is the fastest of the dynamic VMs (~60x), except for string operations



Results: Execution Time

Key Insights

- HLL VMs impose significant slowdown (16x at best)
- WAMR (fast) most efficient (24x avg)
- Dynamic VMs are slower than static VMs
- Lua is the fastest of the dynamic VMs (~60x), except for string operations



Conclusion

Qualitative Comparison

Universal Features:

- Portability
- Updates at Runtime
- Memory Isolation

Conclusion

Qualitative Comparison

Universal Features:

- Portability
- Updates at Runtime
- Memory Isolation

Empirical Evaluation

Substantial Performance Cost

Developer-friendly **Dynamic** VMs vs. Performance and Security focused **Static** VMs

Recommendations for Use Cases:

- **WAMR** as a default choice
- **Lua** for developer experience
- **Femto-Container** for memory constrained use-cases

Conclusion

Qualitative Comparison

Universal Features:

- Portability
- Updates at Runtime
- Memory Isolation

Outlook

- Additional hardware boards
- Additional virtualization technologies
- Explore on-board AOT compilation capabilities

Empirical Evaluation

Substantial Performance Cost

Developer-friendly **Dynamic** VMs vs. Performance and Security focused **Static** VMs

Recommendations for Use Cases:

- **WAMR** as a default choice
- **Lua** for developer experience
- **Femto-Container** for memory constrained use-cases

Sources

IoT Analytics (Oct. 2025). State of IoT 2025. en-US. URL: https://iot-analytics.com/number-connected-iot-devices/ (visited on 01/05/2026)

Popek, Gerald J and Robert P Goldberg (Jan. 1974). “Formal requirements for virtualizable third generation architectures”. en. In: Communications of the ACM 17.7

Smith, J.E. and Ravi Nair (May 2005). “The architecture of virtual machines”. en. In: Computer 38.5, pp. 32–38. ISSN: 0018-9162. DOI: 10.1109/MC.2005.173. URL: http://ieeexplore.ieee.org/document/1430629/ (visited on 07/03/2025).

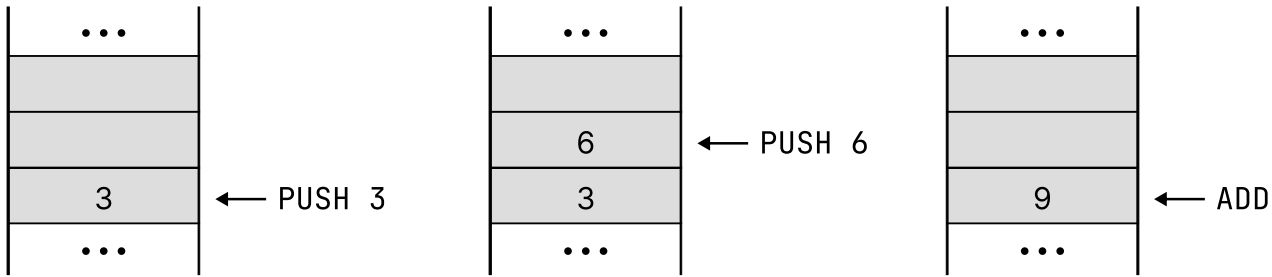
Baccelli, Emmanuel et al. (Dec. 2018). “RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT”. en. In: IEEE Internet of Things Journal 5.6, pp. 4428–4440. ISSN: 2327-4662, 2372-2541. DOI: 10 . 1109 / JIOT . 2018 . 2815038. URL: https://ieeexplore.ieee.org/document/8315125/ (visited on 10/09/2025).

Zandberg, Koen et al. (Nov. 2022). “Femto-containers: lightweight virtualization and fault isolation for small software functions on low-power IoT microcontrollers”. en. In: Proceedings of the 23rd ACM/IFIP International Middleware Conference. Quebec QC Canada: ACM, pp. 161–173. ISBN: 978-1-4503-9340-9. DOI: 10.1145/3528535. 3565242. URL: https://dl.acm.org/doi/10.1145/3528535.3565242 (visited on 05/19/2025)

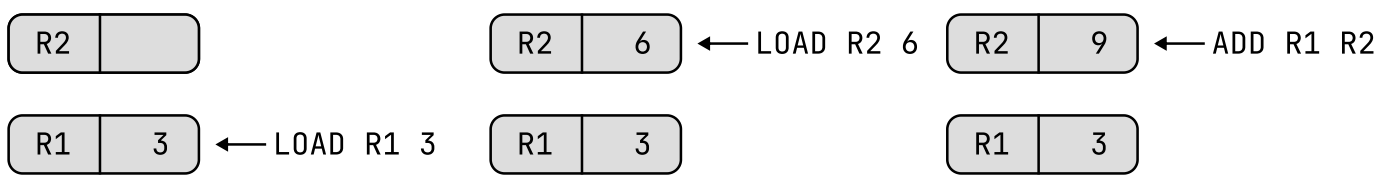
Kubica, Szymon and Marios Kogias (Aug. 2024). “µBPF: Using eBPF for Microcontroller Compartmentalization”. en. In: Proceedings of the SIGCOMM Workshop on eBPF and Kernel Extensions. Sydney NSW Australia: ACM, pp. 23–29. ISBN: 979-8-4007-0712-4. DOI: 10.1145/3672197.3673433. URL: https://dl.acm.org/doi/10.1145/ 3672197.3673433 (visited on 05/21/2025).

Execution Model

Stack based



Register based

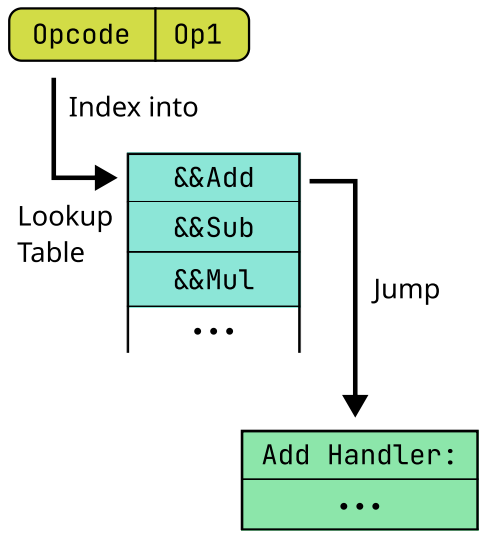


Dispatch Mechanism

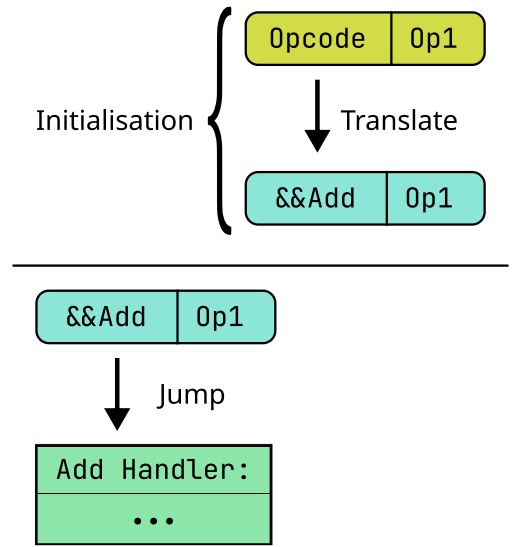
Switch Case Dispatch

```
switch(opcode) {
    case ADD:
        push(op1 + op2);
    ...
}
```

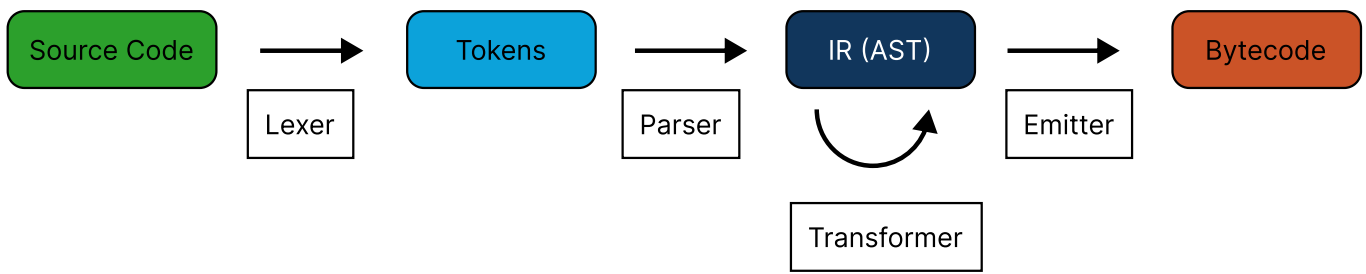
Indirect Threaded Dispatch



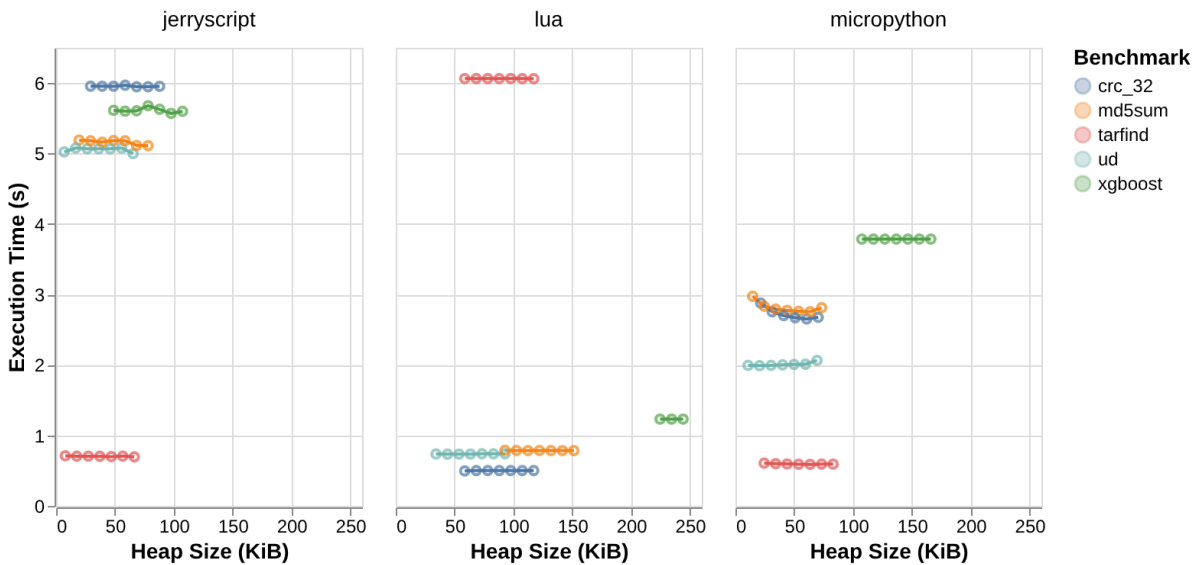
Direct Threaded Dispatch



Bytecode Compilation Pipeline



Execution time vs. Heap size



Minimum Heap size vs Peak Heap Usage