CHERI-RISC-V VP++: Exploring CHERI using a SystemC-based Virtual Platform

Andreas Hinterdorfer, Daniel Große, Manfred Schlägl
Institute for Complex Systems (ICS)
Web: jku.at/ics
Email: andreas.hinterdorfer@jku.at, daniel.grosse@jku.at

23 April 2026

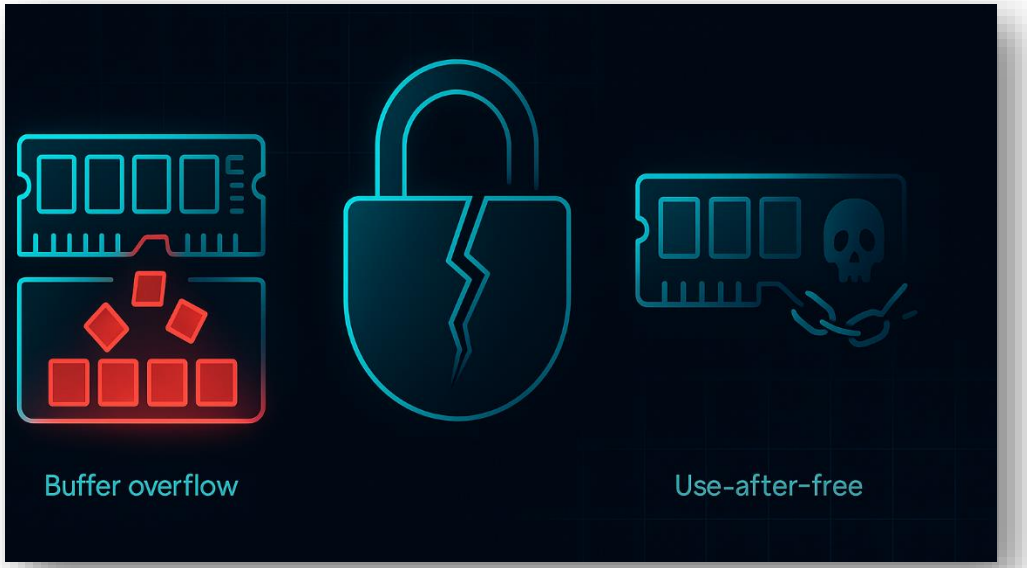
JOHANNES KEPLER UNIVERSITY LINZ

Table of Contents

- Motivation
- Simple Example
- CHERI
- RISC-V VP++
- Implementation of CHERI-RISC-V VP++
- Verification of CHERI-RISC-V VP++
- Complex Examples on CHERI-RISC-V VP++
- Conclusion

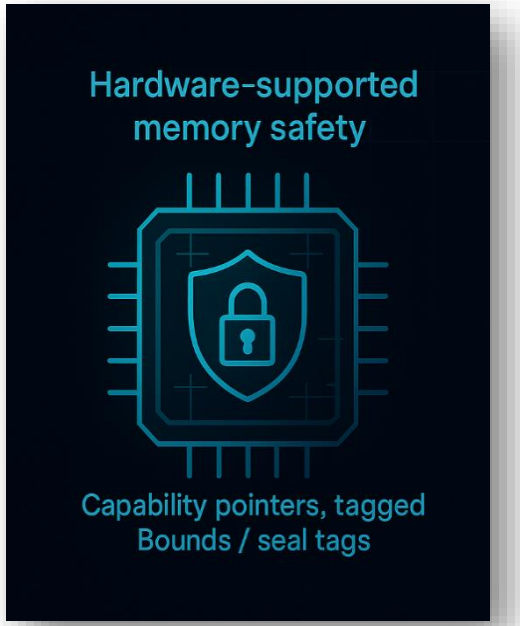
Motivation

- **Memory safety issues** cause majority of security vulnerabilities
  - Buffer overflows
  - Use-after-free



- **Software mitigations** are partial and costly
- **Need**: hardware-supported memory safety

- **Solution**: CHERI



CHERI Evaluation Platforms

RTL-Simulation

- Very slow
- Deterministic
- Cycle-accurate
- High Observability
- Running software basically impossible

Virtual Prototypes

- Faster than RTL
- Deterministic
- Cycle-Approximate
- High Observability
- Can run entire OS

Emulation (QEMU)

- Fast
- Nondeterministic
- Not cycle-accurate
- Hard to inspect hardware model
- Can run entire OS

Goals of This Work

- Observable hardware simulation
- CHERI-enabled VP
  - Extension of existing VP
- Running CHERI-enabled software on VP
- Early software development
- Design space exploration
- Security analysis

Simple Example

```
1  int main() {
2      int32_t array[5] = {0};
3      uint64_t length = sizeof(array) / sizeof(array[0]);
4      int32_t *p_array = array;
5      // Intended read over the bounds
6      for (uint32_t i = 0; i <= length + 5; i++) {
7          printf("Count: %d, Value: %d\n", i, *(p_array + i));
8      }
9      return 0;
10 }
```

Simple Example – CHERI?

No-CHERI

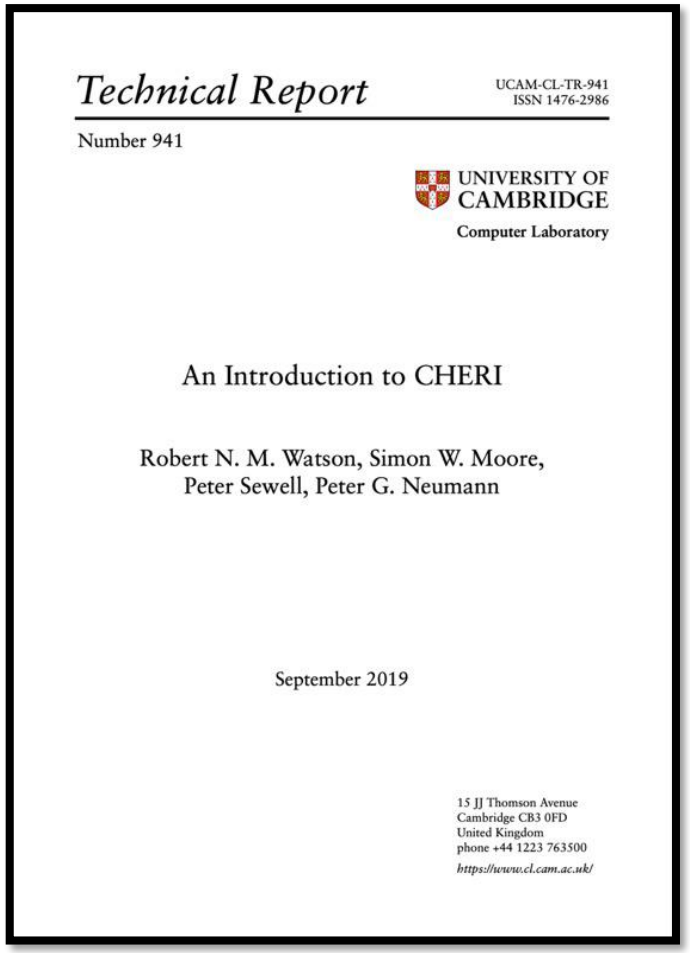
```
Count:  0, Value: 0
Count:  1, Value: 0
Count:  2, Value: 0
Count:  3, Value: 0
Count:  4, Value: 0
Count:  5, Value: 5
Count:  6, Value: 0
Count:  7, Value: 0
Count:  8, Value: 33554364
Count:  9, Value: 9
Count: 10, Value: 5
```

CHERI

```
Count: 0, Value: 0
Count: 1, Value: 0
Count: 2, Value: 0
Count: 3, Value: 0
Count: 4, Value: 0
CHERI Exception: LengthViolation
```

CHERI – Capability Hardware Enhanced RISC Instructions

- Hardware-based approach
- Developed by the University of Cambridge in 2010
- Increase Memory Safety of RISC processors
- Extension of existing ISAs
- ISA independent
  - RISC-V
  - MIPS
  - ARM (Morello)
  - x86
- Large ecosystem available



Technical Report

UCAM-CL-TR-941
ISSN 1476-2986

Number 941

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

An Introduction to CHERI

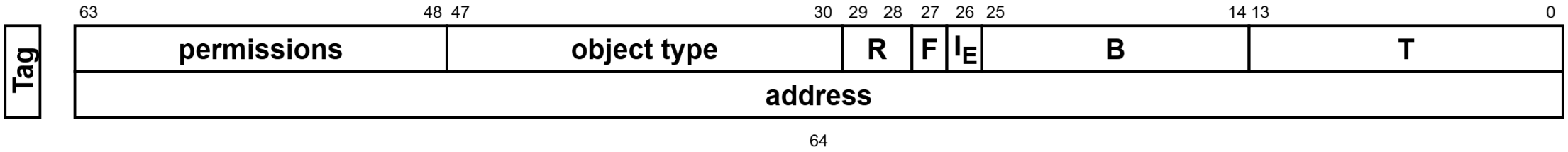
Robert N. M. Watson, Simon W. Moore,
Peter Sewell, Peter G. Neumann

September 2019

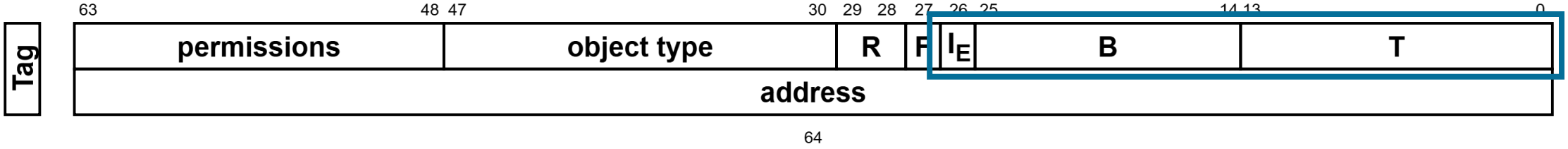
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500
https://www.cl.cam.ac.uk/

Capabilities

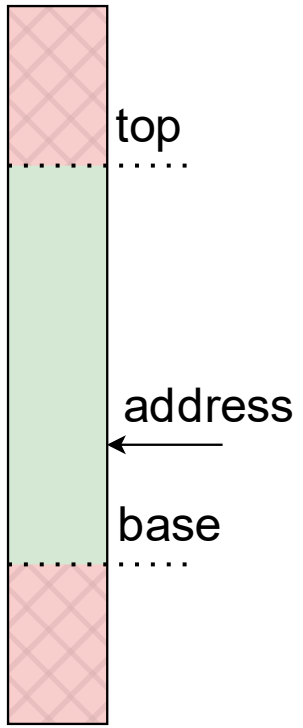
- Capabilities are an architectural primitive that compilers, systems software, and applications use to constrain their own future execution
- Capabilities extend **integer memory addresses** (now 128 bit)
- **Metadata** (bounds, permissions, …) control how it may be used
- **Tags** protect capability integrity/derivation in registers + memory



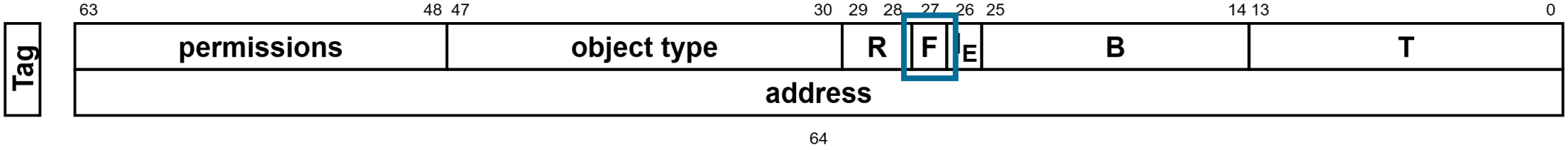
Capabilities



- Encoding of base and top
- Compression algorithm (relative to address)

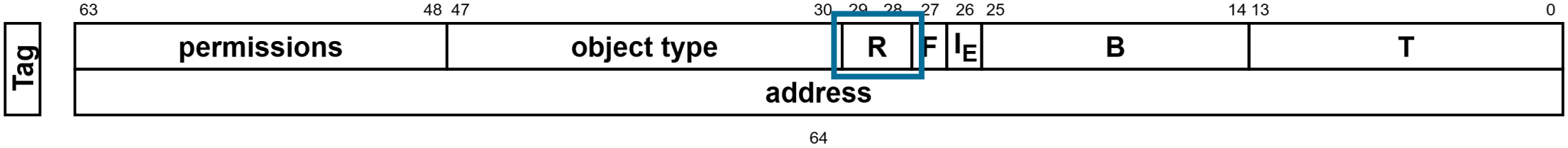


Capabilities



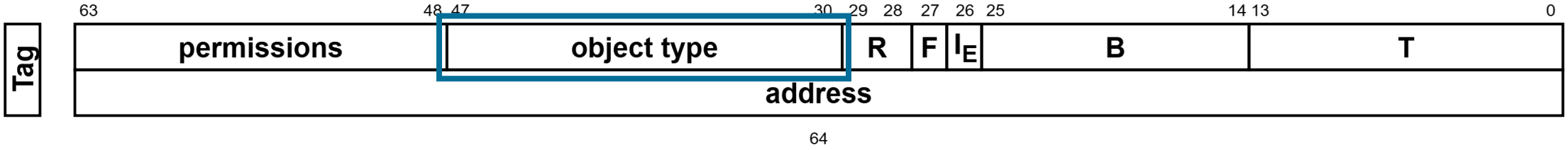
- Flag
- Required in Hybrid-Mode (CHERI aware and legacy code can run side by side)

Capabilities



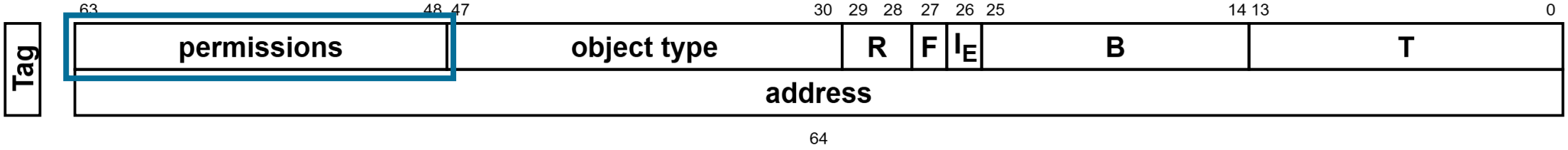
- Reserved

Capabilities



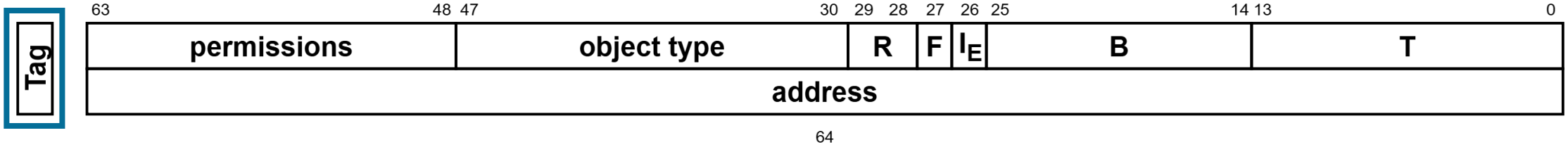
- **Unsealed capabilities**
  - Idea: An unsealed capability is an unlocked pointer that can be used, changed, or accessed freely within its allowed permissions
  - Data capabilities
- **Sealed capabilities**
  - Idea: A sealed capability is a locked pointer that cannot be used or changed until it is unlocked with the correct key, ensuring secure and controlled access
  - Immutable
  - E.g. for jump-targets

Capabilities



- **Memory access**: load/store data and capabilities
- **Control flow**: execute, invoke capabilities
- **Sealing**: seal and unseal capabilities
- **System**: set CIDs, access system registers
- **Scope**: global vs. local

Capabilities



- Out-of-band: stored **separately** from normal data
- Atomically bound to capability
- Validates capability

CHERI: Consequences for Hardware

- Registers become 129 bit (64 bit address + 64 bit metadata + 1 bit validity tag)
- Program counter becomes capability
- Tagged memory protects capability-sized and -aligned words in DRAM by adding validity tag
- ISA is extended with CHERI instructions
- ISA instructions enforce monotonicity and guarded manipulation
  - Capabilities are unforgeable

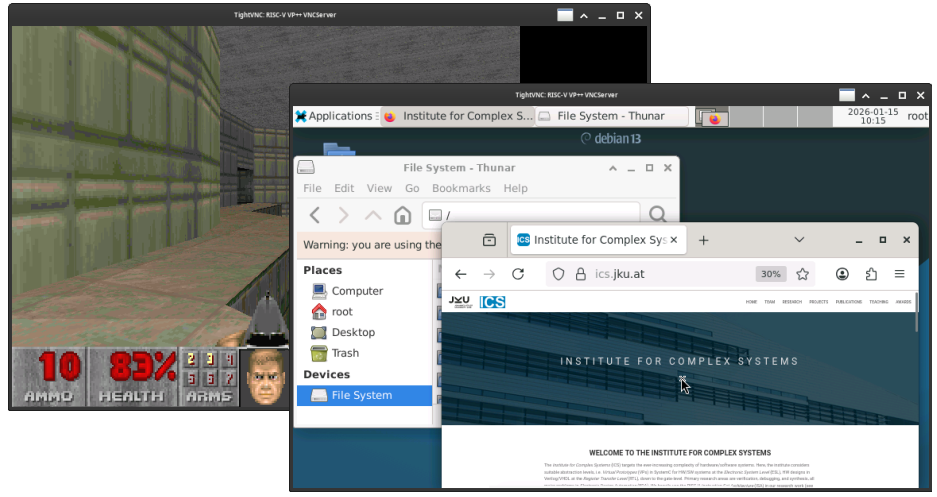
RISC-V VP++

Extensible and configurable
SystemC based, open-source RISC-V virtual prototype

: RISC-V VP++

- RISC-V 32 & 64 bit single/multi-core
  - Fast Interpreter-Based ISS → up to 440 MIPS
  - Vector Extension (RVV) version 1.0
- Small uC based systems (e.g. bare metal SW, RTOS)
- Complex application processor based systems with virtual memory, graphics, network … (e.g. Linux)

→ https://github.com/ics-jku/riscv-vp-plusplus.git



JOHANNES KEPLER UNIVERSITY LINZ

(*) This slide contained an outdated version in the video recording

Some RISC-V VP++ publications

VP Model

- Fast interpreter-based instruction set simulation for virtual prototypes (DATE 2025)
- A RISC-V “V” VP: Unlocking Vector Processing for Evaluation at the System Level (DATE 2024)
- RISC-V VP++: Next generation open-source virtual prototype (OSDA 2024)
- GUI-VP Kit: A RISC-V VP meets Linux graphics - enabling interactive graphical application development (GLSVLSI 2023)

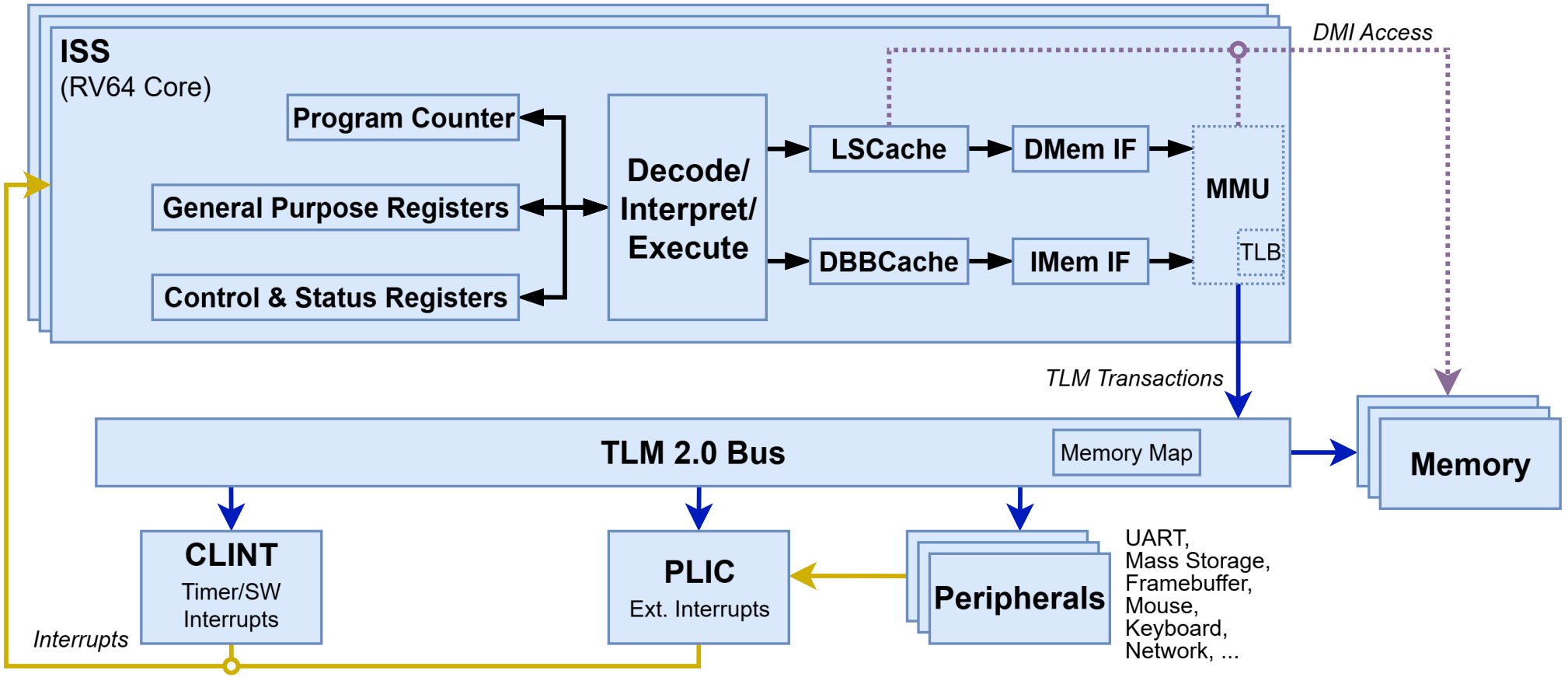
Verification

- LLM-assisted metamorphic testing of embedded graphics libraries (FDL 2025)
- ProtoLens: dynamic transaction visualization in virtual prototypes (FDL 2025)

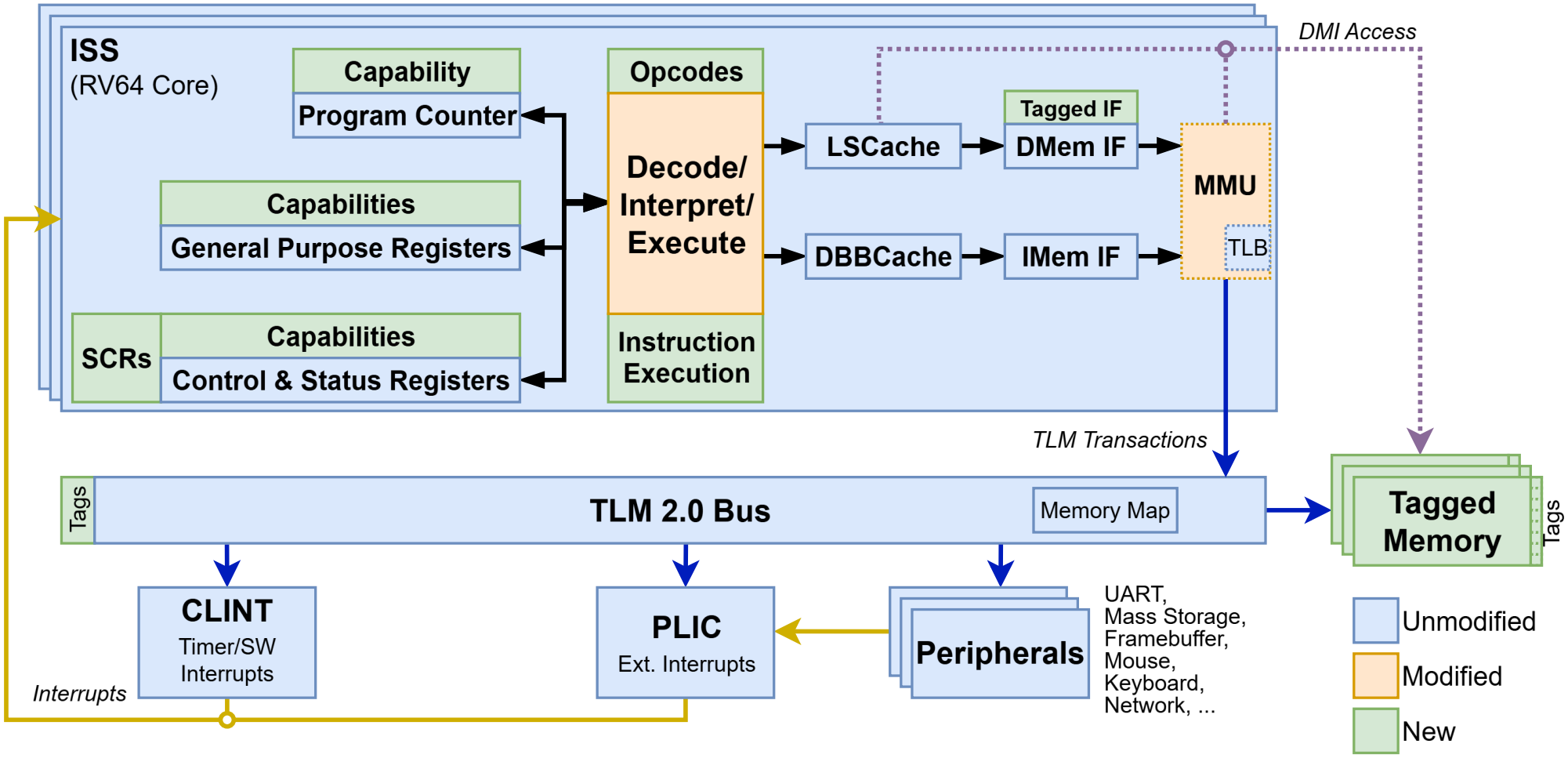
Single instruction isolation for RISC-V vector test failures (ICCAD 2024)

- Using virtual prototypes and metamorphic testing to verify the hardware/software-stack of embedded graphics libraries. (Integration 2025)
- Boosting SW development efficiency with function lifetime diagrams (DDECS 2025)
- Relation coverage: A new paradigm for hardware/software testing (ETS 2024)
- Verifying embedded graphics libraries leveraging virtual prototypes and metamorphic testing (ASP-DAC 2024)

RISC-V VP++


ISS
(RV64 Core)
Program Counter
General Purpose Registers
Control & Status Registers
Decode/
Interpret/
Execute
LSCache
DMem IF
DBBCache
IMem IF
MMU
TLB
DMI Access
TLM Transactions
TLM 2.0 Bus
Memory Map
Memory
CLINT
Timer/SW
Interrupts
PLIC
Ext. Interrupts
Peripherals
UART,
Mass Storage,
Framebuffer,
Mouse,
Keyboard,
Network, ...
Interrupts

CHERI-RISC-V VP++



Reading beyond bounds – Source code

```
1  int main() {
2      int32_t array[5] = {0};
3      uint64_t length = sizeof(array) / sizeof(array[0]);
4      int32_t *p_array = array;
5      // Intended read over the bounds
6      for (uint32_t i = 0; i <= length + 5; i++) {
7          printf("Count: %d, Value: %d\n", i, *(p_array + i));
8      }
9      return 0;
10 }
```

Reading beyond bounds - Output

No-CHERI

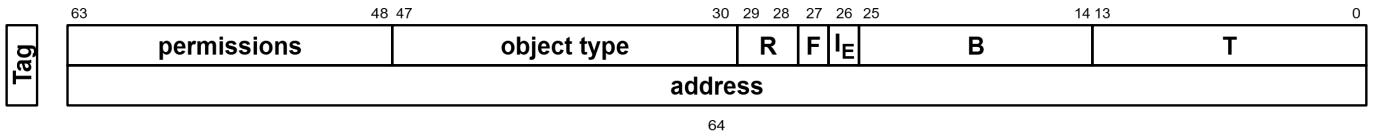
```
Count:  0, Value: 0
Count:  1, Value: 0
Count:  2, Value: 0
Count:  3, Value: 0
Count:  4, Value: 0
Count:  5, Value: 5
Count:  6, Value: 0
Count:  7, Value: 0
Count:  8, Value: 33554364
Count:  9, Value: 9
Count: 10, Value: 5
```

CHERI

```
Count: 0, Value: 0
Count: 1, Value: 0
Count: 2, Value: 0
Count: 3, Value: 0
Count: 4, Value: 0
CHERI Exception: LengthViolation
```

Reading beyond bounds – Source code

```
1  int main() {
2      int32_t array[5] = {0};
3      uint64_t length = sizeof(array) / sizeof(array[0]);
4      int32_t *p_array = array; // Capability pointing to array
5      // Intended read over the bounds
6      for (uint32_t i = 0; i <= length + 5; i++) {
7          printf("Count: %d, Value: %d\n", i, *(p_array + i));
8      }
9      return 0;
10 }
```



Reading beyond bounds – Details

```
1 int main() {
2   int32_t array[5] = {0};
3   int32_t *p_array = array;
4   // Intended read over the bounds
5   for (uint32_t i = 0; i <= 10; i++) {
6       int32_t b = *(p_array + i)
7   }
8   return 0;
9 }
```

```
clc ca0, -80(cs0)
clwu a1, -84(cs0) # i
slli a1, a1, 2 # *4
cincoffset ca0, ca0, a1
clw a0, 0(ca0)
csw a0, -88(cs0)
```

```
cincoffset ca0, cs0, -56
csetbounds ca0, ca0, 20
csc ca0, -112(cs0)
```

```
Capability{
  Addr: 0x9FFFFFB8
  Base: 0x9FFFFFB8
  Top:0x9FFFFFCC
}
… stored to address 0x9FFFFF80
```

```
clc ca0, -112(cs0)
csc ca0, -80(cs0)
```

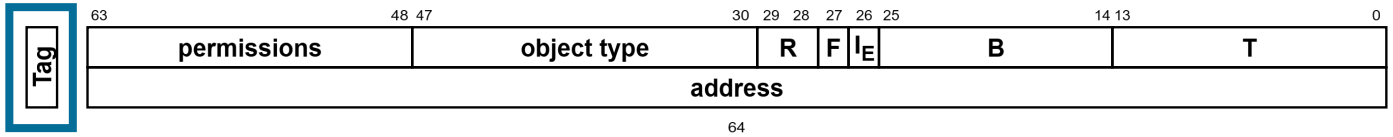
```
Loaded Capability from address 0x9FFFFF80
Stored to address 0x9FFFFFA0:
Capability{
  Addr: 0x9FFFFFB8
  Base: 0x9FFFFFB8
  Top:0x9FFFFFCC
}
```

```
Loaded Capability from
address 0x9FFFFFA0
```

```
CHERI Exception: LengthViolation
Capability{
  Addr: 0x9FFFFFCC
  Base: 0x9FFFFFB8
  Top: 0x9FFFFFCC
}
```

```
addr + size <= top
```

capability load word from given address:
array element is loaded into a0 if bound check passes



TLM TagExtension - Concept

- **Purpose**: Implements a custom extension for SystemC/TLM to support tagged data
- **Class Definition**: `TagExtension` inherits from `tlm::tlm_extension`
- **Key Features**:
  - Holds a single Boolean `tag` value to represent validity of a capability
- **Integration**:
  - Appended by our CHERI-enabled ISS to `tlm_generic_payload` to enable transport of tags alongside data
  - Ensures backward compatibility: Non-CHERI-aware modules ignore the extension and operate as usual
- **Code Simplicity**:
  - Minimal implementation, focusing on extending functionality w/o altering the TLM bus
- No modification to SystemC necessary!

TLM TagExtension - Code

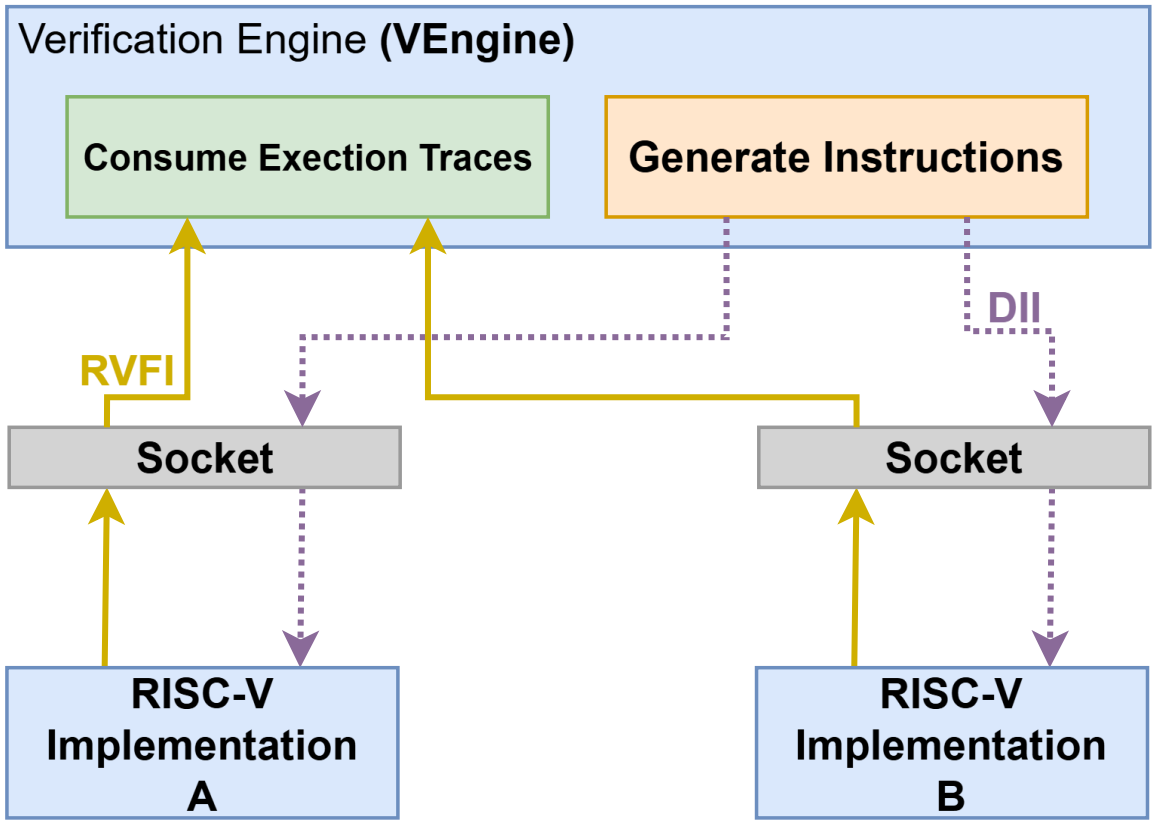
```
struct TagExtension : tlm::tlm_extension<TagExtension>
{
    bool tag;
    TagExtension(bool t)
    {
        tag = t;
    }

    tlm::tlm_extension_base* clone() const override
    {
        return new TagExtension(*this);
    }

    void copy_from(tlm::tlm_extension_base const &ext) override
    {
        tag = static_cast<TagExtension const &>(ext).tag;
    }
};
```

TLM TagExtension - Usage

```
tlm::tlm_generic_payload trans;
tlm_ext_initiator *trans_ext_initiator;
tlm_ext_tag *trans_ext_tag;
CombinedTaggedMemoryInterface(...){
    ...
    trans_ext_tag = new tlm_ext_tag(false);
    trans.set_extension(trans_ext_tag);
}
inline void _do_trans(tlm_command cmd, uint64_t addr, uint8_t* data, bool* tag, unsigned num)
{
    trans.set_command(cmd);
    trans.set_address(addr);
    trans.set_data_ptr(data);
    trans.set_data_length(num);
    trans.set_response_status(tlm::TLM_OK_RESPONSE);
    trans_ext_tag->tag = *tag; // set tag
    iss.commit_cycles();
    sc_core::sc_time local_delay = quantum_keeper.get_local_time();
    isock->b_transport(trans, local_delay);
    *tag = trans_ext_tag->tag; // get tag
    ...
}
```


(*) This slide contained an outdated version in the video recording

TestRIG


Verification Engine (VEngine)
Consume Exection Traces
Generate Instructions
DII
RVFI
Socket
Socket
RISC-V Implementation A
RISC-V Implementation B

TestRIG Results

- **Test Cases Executed**:
  2+ million test cases run across various categories, including CHERI-specific extensions
- **Instruction Coverage**:
  Executed 1.8+ billion instructions, averaging 860 instructions per test case
- **Code Coverage**:
  Relevant CHERI-related code coverage: High, with most critical lines tested

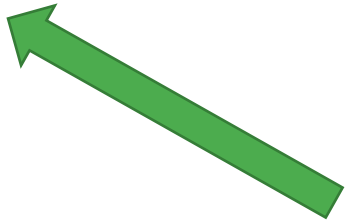
- **Planned**:
  - Adapt RVVTS (ICCAD 2024) for CHERI-RISC-V VP++

Booting CheriBSD on CHERI-RISC-V VP++

- CHERI-enabled FreeBSD
  - Full-scale general purpose OS
  - Unix-like
- Boots in 25 seconds
  - > 300 million instructions
  - 20 million data loads
  - 60 million data stores
  - 600 thousand capabilities stored
- File system, multitasking and networking supported
- CHERI memory protection enforced

CheriBSD Reading beyound bounds

```
# ./read_beyound_bounds
Count: 0, Value: 0
Count: 1, Value: 0
Count: 2, Value: 0
Count: 3, Value: 0
Count: 4, Value: 0
In-address space security exception (core dumped)
#
```



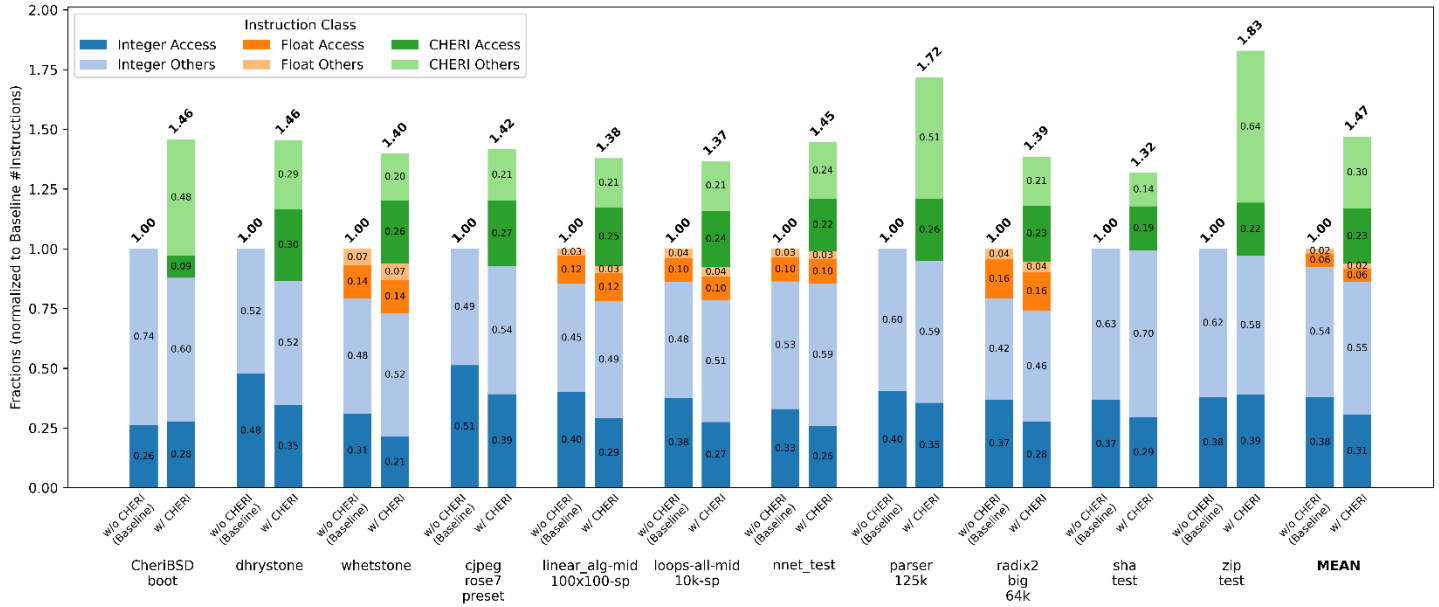
Operating system kills the program, but system still runs!

Conclusion (1)

- Extension of RISC-V VP++ with CHERI
- Easy integration of tagged memory architecture using SystemC TLM transaction extension mechanism
- Verification of CHERI-RISC-V VP++ w TestRIG
- Bare-metal software on CHERI-RISC-V VP++
- Booting CheriBSD on CHERI-RISC-V VP++
- Open-source on GitHub: RISC-V VP++
- Very promising results

Conclusion (2)

- [CHERI-RISC-V VP++: A Virtual Prototyping Platform Enabling Fine-Grained Memory Protection]
- [A RISC-V CHERI VP: Enabling System-Level Evaluation of the Capability-Based CHERI Architecture (ASP-DAC 2026)]
- Stay tuned!
  - RISC-V Summit 08-12.06.2026
  - CHERI Benchmarks
  - Security Evaluation



CHERI-RISC-V VP++: Exploring CHERI using a SystemC-based Virtual Platform

Andreas Hinterdorfer, Daniel Große, Manfred Schlägl
Institute for Complex Systems (ICS)
Web: jku.at/ics
Email: andreas.hinterdorfer@jku.at, daniel.grosse@jku.at

23 April 2026

JOHANNES KEPLER UNIVERSITY LINZ