# CHERI
# Capability Hardware Enhanced RISC Instructions

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

Hesham Almatary, Ricardo de Oliveira Almeida, Jonathan Anderson, Alasdair Armstrong, Rosie Baish, Peter Blandford-Baker, John Baldwin, Hadrien Barrel, Thomas Bauereiss, Ruslan Bukin, Brian Campbell, David Chisnall, Jessica Clarke, Nirav Dave, Brooks Davis, Lawrence Esswood, Nathaniel W. Filardo, Franz Fuchs, Dapeng Gao, Ivan Gomes-Ribeiro, Khilan Gudka, Brett Gutstein, Angus Hammond, Graeme Jenkinson, Alexandre Joannou, Mark Johnston, Robert Kovacsics, Ben Laurie, A. Theo Markettos, J. Edward Maste, Alfredo Mazzinghi, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch, Edward Napierala, George Neville-Neil, Kyndylan Nienhuis, Robert Norton-Wright, Philip Paeps, Lucian Paul-Trifu, Allison Randal, Ivan Ribeiro, Alex Richardson, Michael Roe, Colin Rothwell, Peter Rugg, Hassen Saidi, Thomas Sewell, Stacey Son, Ian Stark, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Konrad Witaszczyk, Jonathan Woodruff, Hongyan Xia, Vadim Zaliva, and Bjoern A. Zeeb

University of Cambridge and SRI International

Web Slide Deck – 12 October 2022

Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contracts FA8750-10-C-0237 ("CTSRD"), with additional support from FA8750-11-C-0249 ("MRC2"), HR0011-18-C-0016 ("ECATS"), FA8650-18-C-7809 ("CIFV"), and HR001122C0110 ("ETC") . The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

**Approved for public release; distribution is unlimited.**

This work was supported in part by the Innovate UK project Digital Security by Design (DSbD) Technology Platform Prototype, 105694.

This work was also supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237 (“CTSRD”), with additional support from FA8750-11-C-0249 (“MRC2”), HR0011-18-C-0016 (“ECATS”), FA8650-18-C-7809 (“CIFV”), and HR001122C0110 (“ETC”) as part of the DARPA CRASH, MRC, and SSITH research programs. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

We further acknowledge the EPSRC REMS Programme Grant (EP/K008528/1), the ERC ELVER Advanced Grant (789108), the Isaac Newton Trust, the UK Higher Education Innovation Fund (HEIF), Thales E-Security, Microsoft Research Cambridge, Arm Limited, Google, Google DeepMind, HP Enterprise, and the Gates Cambridge Trust.

# CHERI introduction

- **CHERI is a new processor technology that mitigates software security vulnerabilities**
  - Developed by the University of Cambridge and SRI International starting in 2010, supported by DARPA
  - Arm collaboration from 2014
  - Arm Morello CPU, SoC, and board announced 2019, with support from UKRI; shipping as of Jan 2022
- Today's talk:
  - What is CHERI, how does it work, and is it any good?
  - What is a Morello board, and what can I do with one?
- http://www.cheri-cpu.org/

An early experimental FPGA-based CHERI tablet prototype running the CheriBSD operating system and applications, Cambridge, 2013.

High-performance Arm Morello chip able to run a full CHERI software stack, Cambridge, 2022

## Introduction

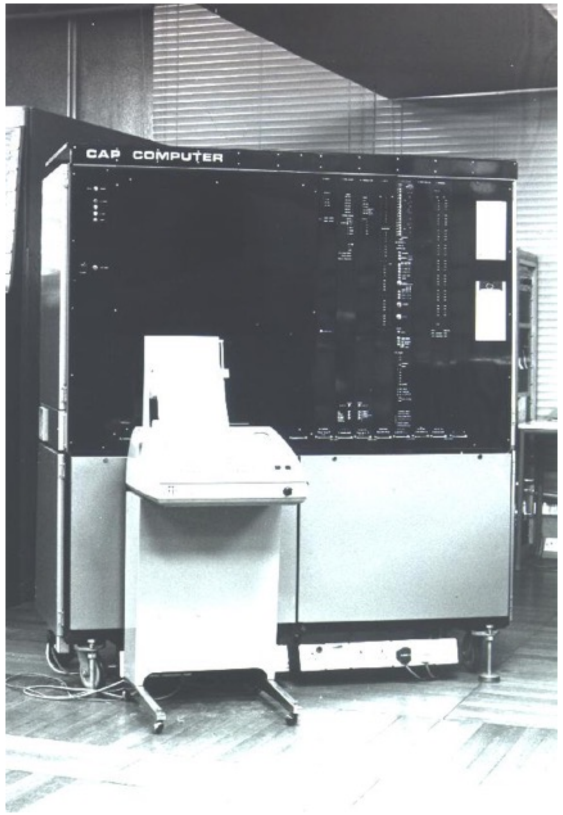
- An introduction to capabilities and the CHERI architecture
- Ongoing CHERI research and transition

- To learn more about the CHERI architecture and prototypes:

  http://www.cheri-cpu.org/

- Watson, et al. **An Introduction to CHERI**, UCAM-CL-TR-941, September 2019.
- Watson, et al. **Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 8)**, UCAM-CL-TR-951, October 2020.
- Watson, et al. **CHERI C/C++ Programming Guide**, UCAM-CL-TR-947, June 2020.
- Watson, et al. **DSbD CHERI and Morello Capability Essential IP (Version 1)** UCAM-CL-TR-953, December 2020.

# Capability systems

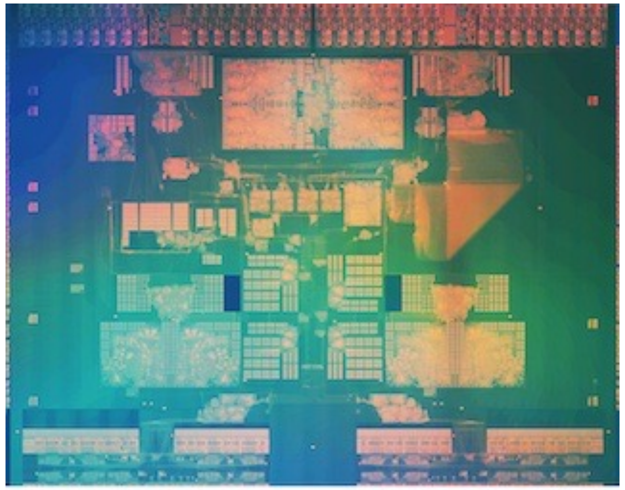


The CAP computer project ran from 1970-1977 at the University of Cambridge, led by R. Needham, M. Wilkes, and D. Wheeler.

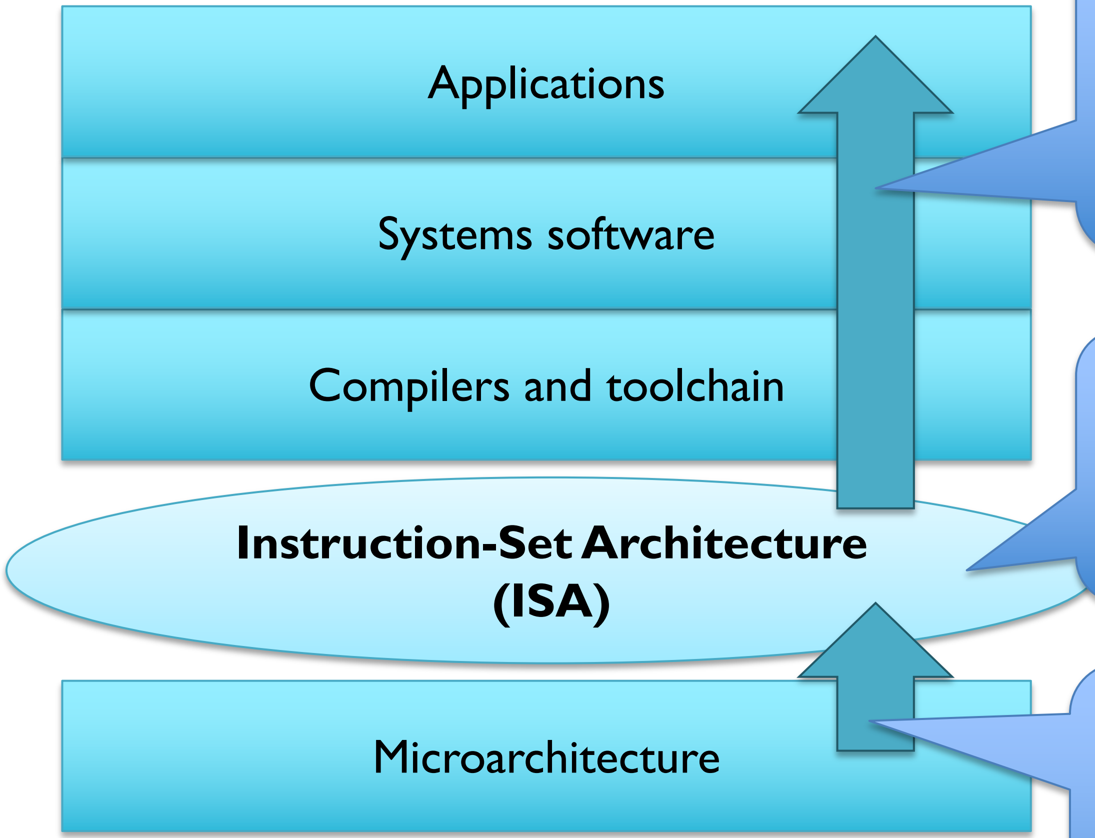
- The capability system is a **design pattern** for how CPUs, languages, OSes, … can control access to resources
  - **Capabilities** are communicable, unforgeable tokens of authority
  - In **capability-based systems,** resources are reachable **only** via capabilities
- Capability systems limit the **scope and spread of damage** from accidental or intentional software misbehavior
- They do this by making it **natural and efficient** to implement, in software, two security design principles:
  - The **principle of least privilege** dictates that software should run with the minimum privileges to perform its tasks
  - The **principle of intentional use** dictates that when software holds multiple privileges, it must explicitly select which to exercise

# What is CHERI?

- CHERI is a processor **architectural protection model**
  - Composes a **capability-system model** with hardware and software
  - Adds new security primitives to Instruction-Set Architectures (ISAs)
  - Implemented by microarchitectural extensions to the CPU and SoC
  - Enables new security behavior in software
- CHERI mitigates vulnerabilities in **C/C++ Trusted Computing Bases**
  - Hypervisors, operating systems, language runtimes, browsers, ….
  - **Fine-grained memory protection** deterministically closes many arbitrary code execution attacks, and directly impedes common exploit-chain tools
  - **Scalable compartmentalization** mitigates many vulnerability classes .. even unknown future classes .. by extending the idea of software sandboxing
- **CHERI-RISC-V** research architecture and prototype FPGA implementations
- **Arm Morello** industrial quality demonstrator CPU, SoC, board


Morello chip – quad-core multi-GHz Arm processor and SoC with CHERI extensions, Arm, 2022.

# Architectural primitives for software security

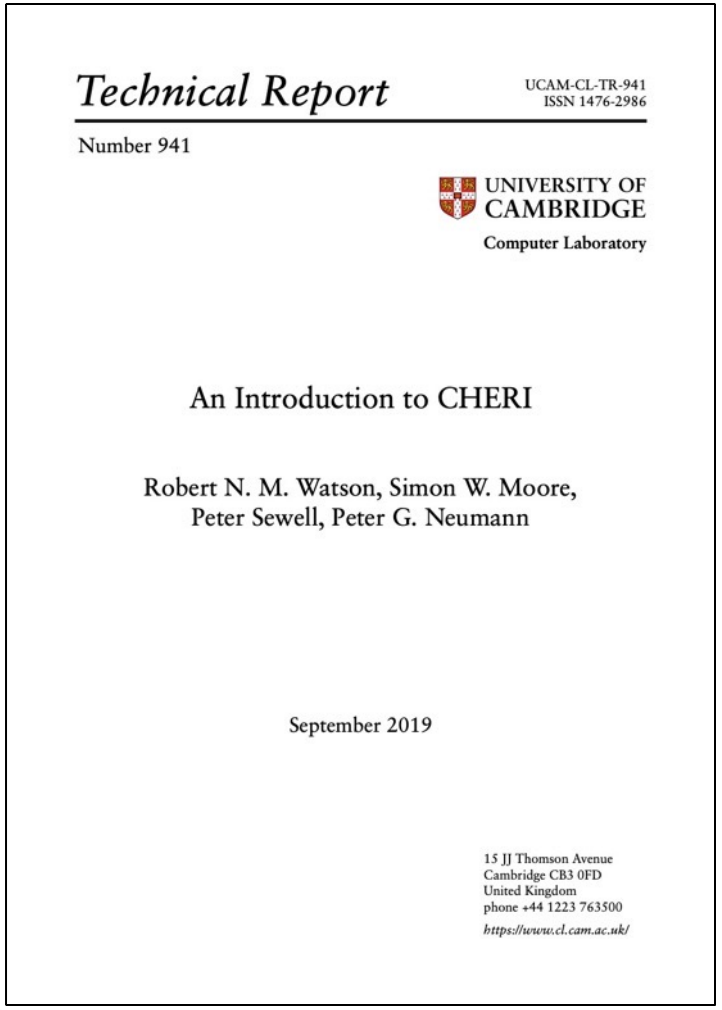


Software configures and uses capabilities to continuously enforce safety properties such as **referential, spatial, and temporal memory safety,** as well as higher-level security constructs such as **compartment isolation**

**CHERI capabilities** are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

The microarchitecture implements the **capability data type** and **tagged memory**, enforcing invariants on their manipulation and use such as **capability bounds**, **monotonicity**, and **provenance validity**

# An Introduction to CHERI



*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/*

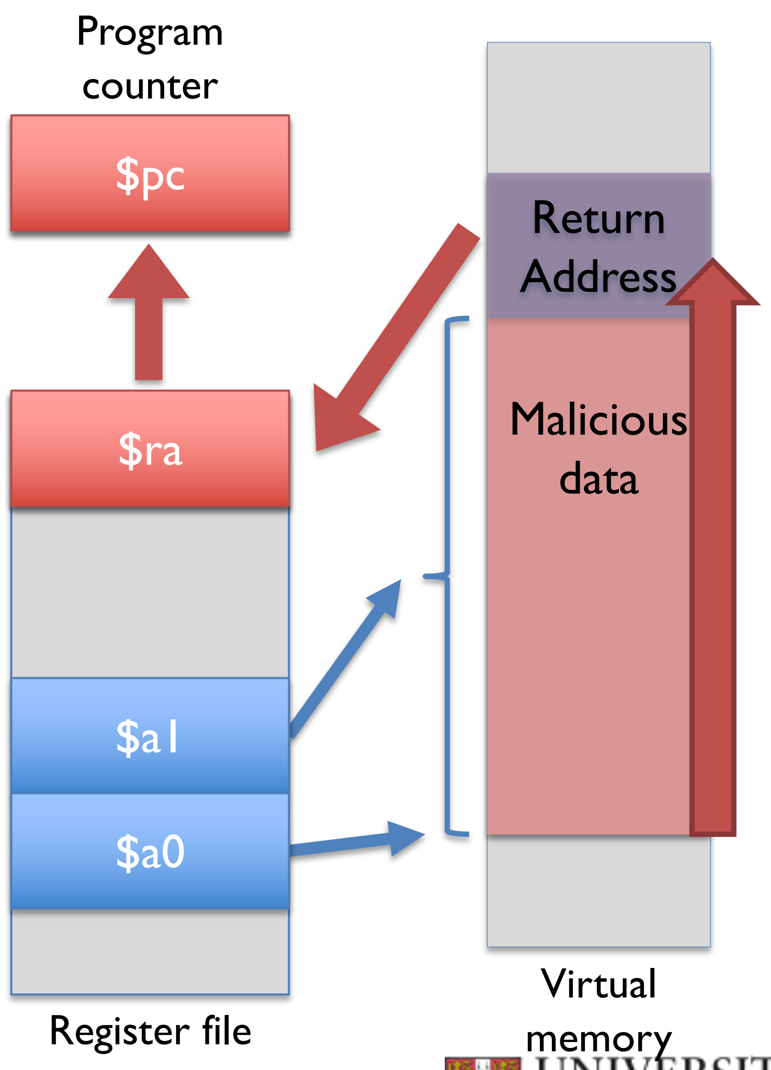
- Watson, et al. **An Introduction to CHERI**, UCAM-CL-TR-941, September 2019
  - Architectural capabilities and the CHERI ISA
  - CHERI microarchitecture
  - ISA formal modeling and proof
  - Software construction with CHERI
  - Language and compiler extensions
  - OS extensions
  - Application-level adaptations

NB: Predates public announcement of Morello

# (Lack of) architectural least privilege

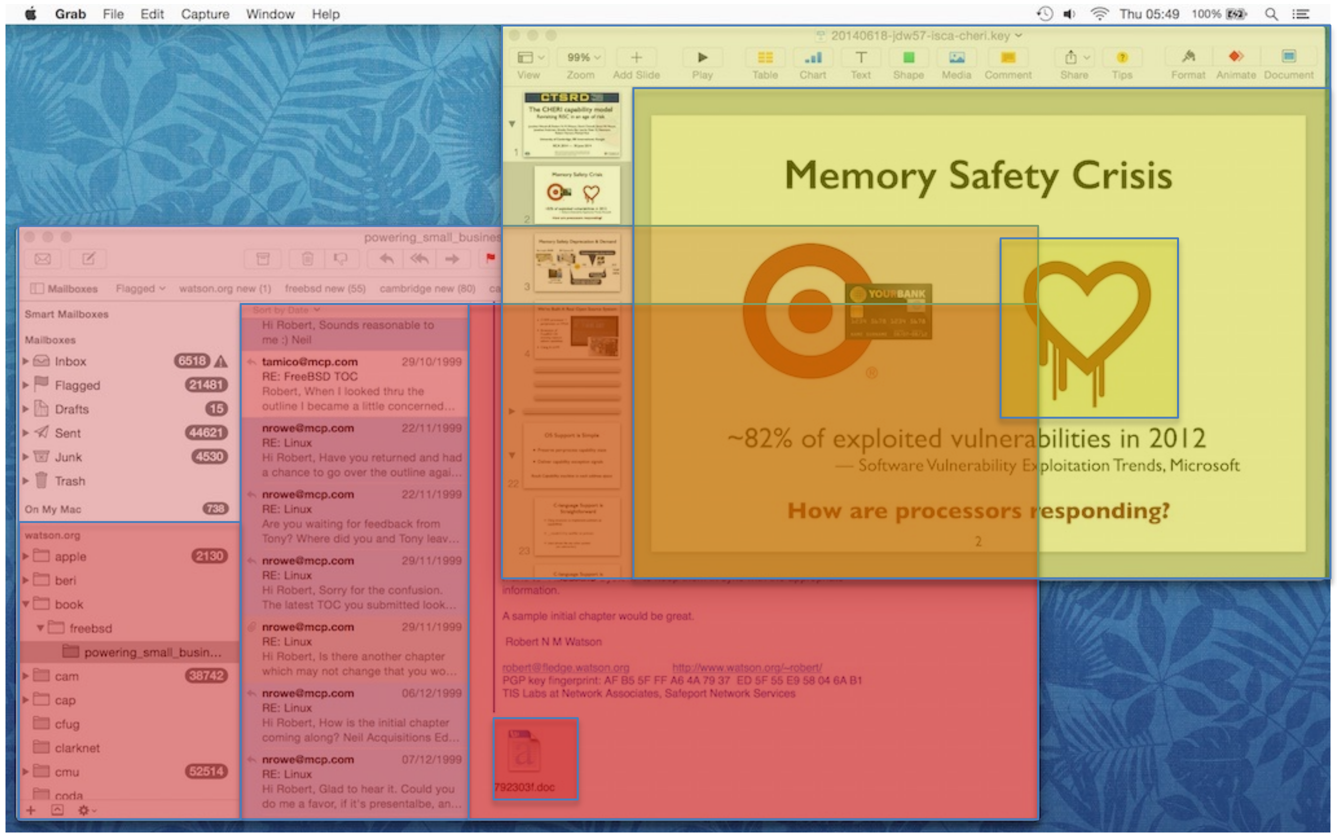
- Classical buffer-overflow attack
  1. Buggy code overruns a buffer, overwrites return address with attacker-provided value
  2. Overwritten return address is loaded and jumped to, allowing the attacker to manipulate control flow
- These privileges were not required by the C language; why allow code the ability to:
  - Write outside the target buffer?
  - Corrupt or inject a code pointer?
  - Execute data as code / re-use code?
- Limiting privilege doesn't fix bugs – but does provide **vulnerability mitigation**

➢ Memory Management Units (MMUs) do not enable **efficient, fine-grained privilege reduction**



# Application-level least privilege

**Software compartmentalization** decomposes software into **isolated compartments** that are delegated **limited rights**



Memory Safety Crisis

~82% of exploited vulnerabilities in 2012
— Software Vulnerability Exploitation Trends, Microsoft

How are processors responding?

2

Hi Robert, Sounds reasonable to me :) Neil

tamico@mcp.com 29/10/1999
RE: FreeBSD TOC
Robert, When I looked thru the outline I became a little concerned...

nrowe@mcp.com 22/11/1999
RE: Linux
Hi Robert, Have you returned and had a chance to go over the outline agai...

nrowe@mcp.com 22/11/1999
RE: Linux
Are you waiting for feedback from Tony? Where did you and Tony leav...

nrowe@mcp.com 29/11/1999
RE: Linux
Hi Robert, Sorry for the confusion. The latest TOC you submitted look...

nrowe@mcp.com 29/11/1999
RE: Linux
Hi Robert, Is there another chapter which may not change that you wo...

nrowe@mcp.com 06/12/1999
RE: Linux
Hi Robert, How is the initial chapter coming along? Neil Acquisitions Ed...

nrowe@mcp.com 07/12/1999
RE: Linux
Hi Robert, Glad to hear it. Could you do me a favor, if it's presentalbe, an...

information.

A sample initial chapter would be great.

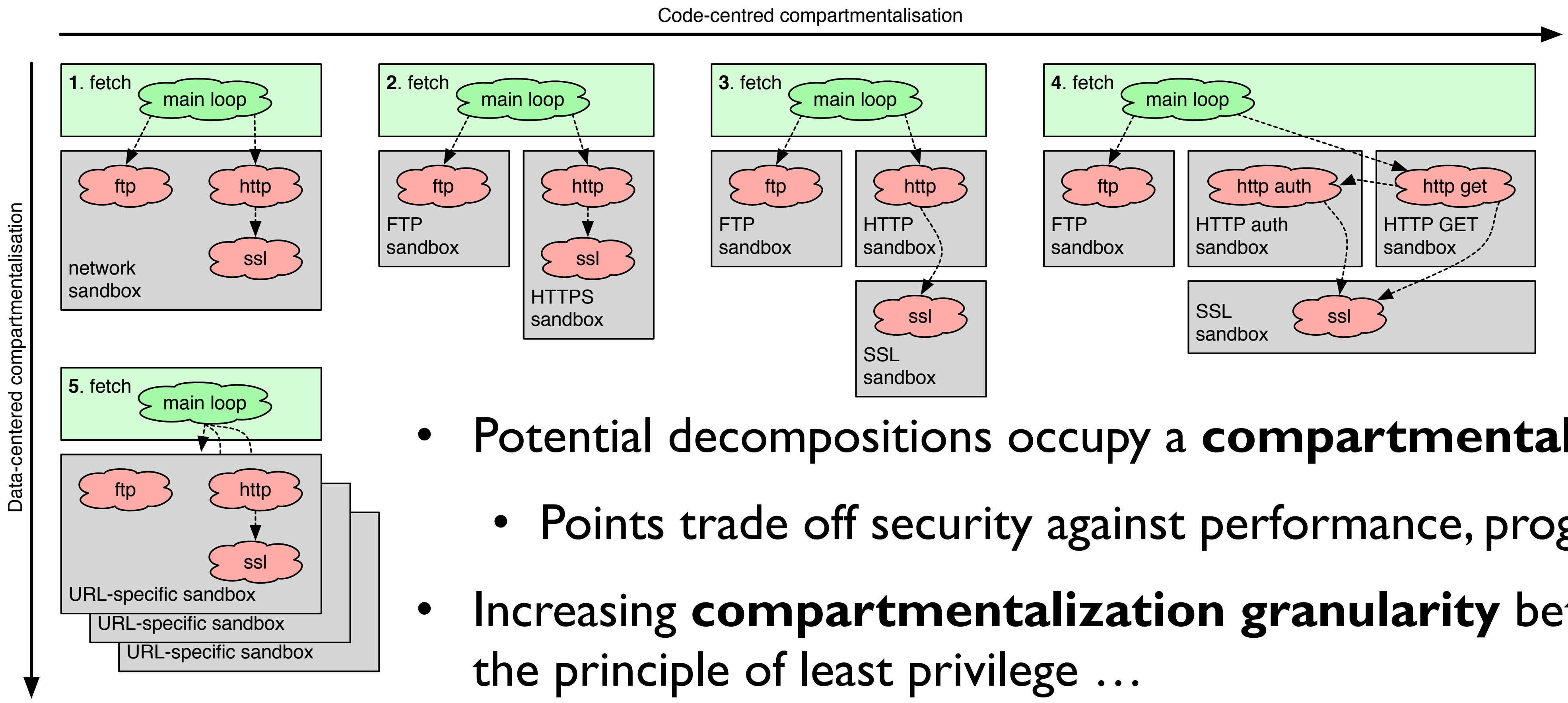
Robert N M Watson

robert@fledge.watson.org http://www.watson.org/~robert/
PGP key fingerprint: AF B5 5F FF A6 4A 79 37 ED 5F 55 E9 58 04 6A B1
TIS Labs at Network Associates, Safeport Network Services

**Potential compartmentalization boundaries** matching reasonable user expectations for **least privilege** can be found in many user-facing apps.

E.g., a malicious email attachment should not be able to gain access to other attachments, messages, folders, accounts, or the system as a whole.

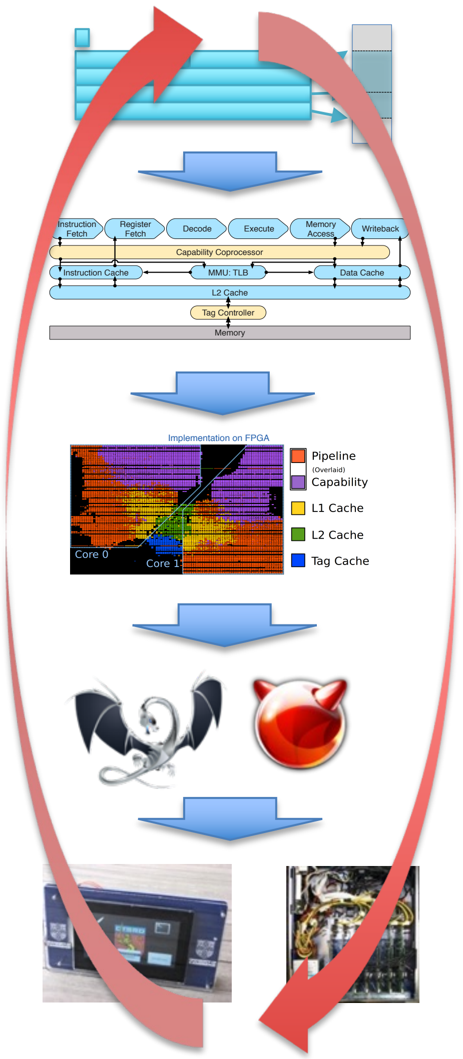
Able to mitigate not only **unknown vulnerabilities**, but also **as-yet undiscovered classes of vulnerabilities and exploits**



- Potential decompositions occupy a **compartmentalization space**:
  - Points trade off security against performance, program complexity
- Increasing **compartmentalization granularity** better approximates the principle of least privilege …
- … but **MMU-based architectures** do not scale to many processes:
  - Poor spatial protection granularity
  - Limited simultaneous-process scalability
  - Multi-address-space programming model

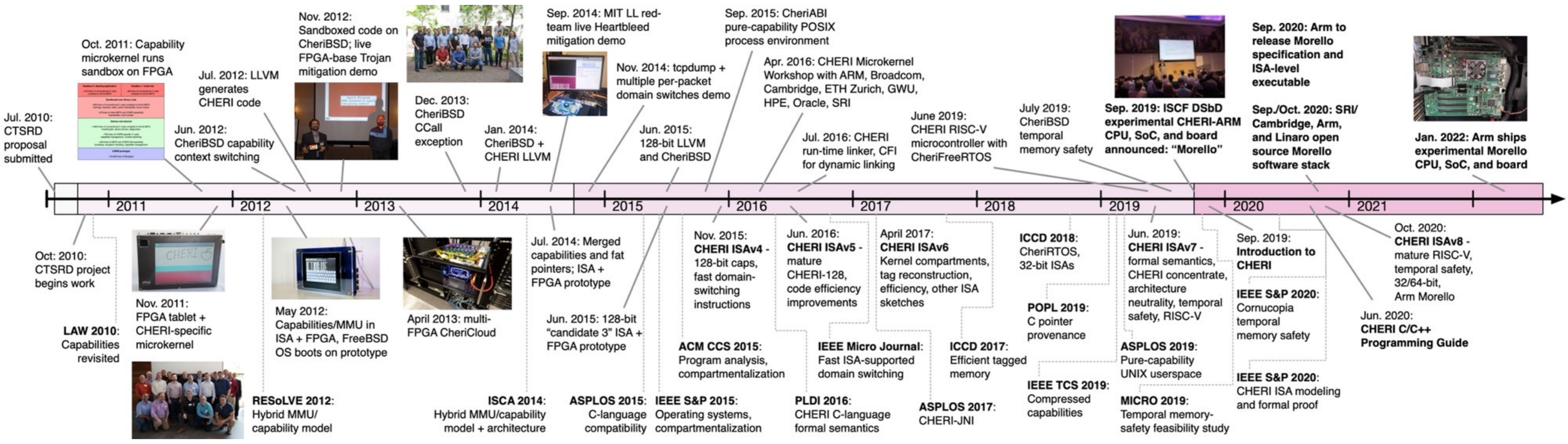
## HARDWARE-SOFTWARE CO-DESIGN FOR CHERI

# Hardware-software-semantics co-design



- University of Cambridge and SRI International from 2010 supported by DARPA
- Architectural mitigation for C/C++ TCB vulnerabilities
  - Tagged memory, new hardware capability data type
  - Model hybridizes cleanly with contemporary hardware and software designs
  - New hardware enables incremental software deployment
- Hardware-software-semantics co-design + concrete prototyping:
  - CHERI abstract protection model; concrete ISA instantiations in 64-bit MIPS, 32/64-bit RISC-V, 64-bit Armv8-a (Morello)
  - Formal ISA models, Qemu-CHERI, and multiple FPGA prototypes
  - Formal proofs that ISA security properties are met, automatic testing
  - CHERI Clang/LLVM/LLD, CheriBSD, C/C++-language applications
  - Repeated iteration to improve {performance, security, compatibility, ..}

# CHERI research and development timeline



Jul. 2010: CTSRD proposal submitted

Oct. 2011: Capability microkernel runs sandbox on FPGA

Jun. 2012: CheriBSD capability context switching

Jul. 2012: LLVM generates CHERI code

Nov. 2012: Sandboxed code on CheriBSD; live FPGA-base Trojan mitigation demo

Dec. 2013: CheriBSD CCall exception

Jan. 2014: CheriBSD + CHERI LLVM

Sep. 2014: MIT LL red-team live Heartbleed mitigation demo

Nov. 2014: tcpdump + multiple per-packet domain switches demo

Jun. 2015: 128-bit LLVM and CheriBSD

Sep. 2015: CheriABI pure-capability POSIX process environment

Apr. 2016: CHERI Microkernel Workshop with ARM, Broadcom, Cambridge, ETH Zurich, GWU, HPE, Oracle, SRI

Jul. 2016: CHERI run-time linker, CFI for dynamic linking

June 2019: CHERI RISC-V microcontroller with CheriFreeRTOS

July 2019: CheriBSD temporal memory safety

**Sep. 2019: ISCF DSbD experimental CHERI-ARM CPU, SoC, and board announced: "Morello"**

**Sep. 2020: Arm to release Morello specification and ISA-level executable**

**Sep./Oct. 2020: SRI/ Cambridge, Arm, and Linaro open source Morello software stack**

**Jan. 2022: Arm ships experimental Morello CPU, SoC, and board**

2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021

Oct: 2010: CTSRD project begins work

**LAW 2010**: Capabilities revisited

Nov. 2011: FPGA tablet + CHERI-specific microkernel

May 2012: Capabilities/MMU in ISA + FPGA, FreeBSD OS boots on prototype

**RESoLVE 2012**: Hybrid MMU/ capability model

April 2013: multi-FPGA CheriCloud

Jul. 2014: Merged capabilities and fat pointers; ISA + FPGA prototype

**ISCA 2014**: Hybrid MMU/capability model + architecture

Jun. 2015: 128-bit "candidate 3" ISA + FPGA prototype

**ASPLOS 2015**: C-language compatibility

**IEEE S&P 2015**: Operating systems, compartmentalization

Nov. 2015: **CHERI ISAv4** - 128-bit caps, fast domain-switching instructions

**ACM CCS 2015**: Program analysis, compartmentalization

Jun. 2016: **CHERI ISAv5** - mature CHERI-128, code efficiency improvements

**IEEE Micro Journal**: Fast ISA-supported domain switching

**PLDI 2016**: CHERI C-language formal semantics

April 2017: **CHERI ISAv6** Kernel compartments, tag reconstruction, efficiency, other ISA sketches

**ICCD 2017**: Efficient tagged memory

**ASPLOS 2017**: CHERI-JNI

**ICCD 2018**: CheriRTOS, 32-bit ISAs

**POPL 2019**: C pointer provenance

**IEEE TCS 2019**: Compressed capabilities

Jun. 2019: **CHERI ISAv7** - formal semantics, CHERI concentrate, architecture neutrality, temporal safety, RISC-V

**ASPLOS 2019**: Pure-capability UNIX userspace

**MICRO 2019**: Temporal memory-safety feasibility study

Sep. 2019: **Introduction to CHERI**

**IEEE S&P 2020**: Cornucopia temporal memory safety

**IEEE S&P 2020**: CHERI ISA modeling and formal proof

Jun. 2020: **CHERI C/C++ Programming Guide**

Oct. 2020: **CHERI ISAv8** - mature RISC-V, temporal safety, 32/64-bit, Arm Morello

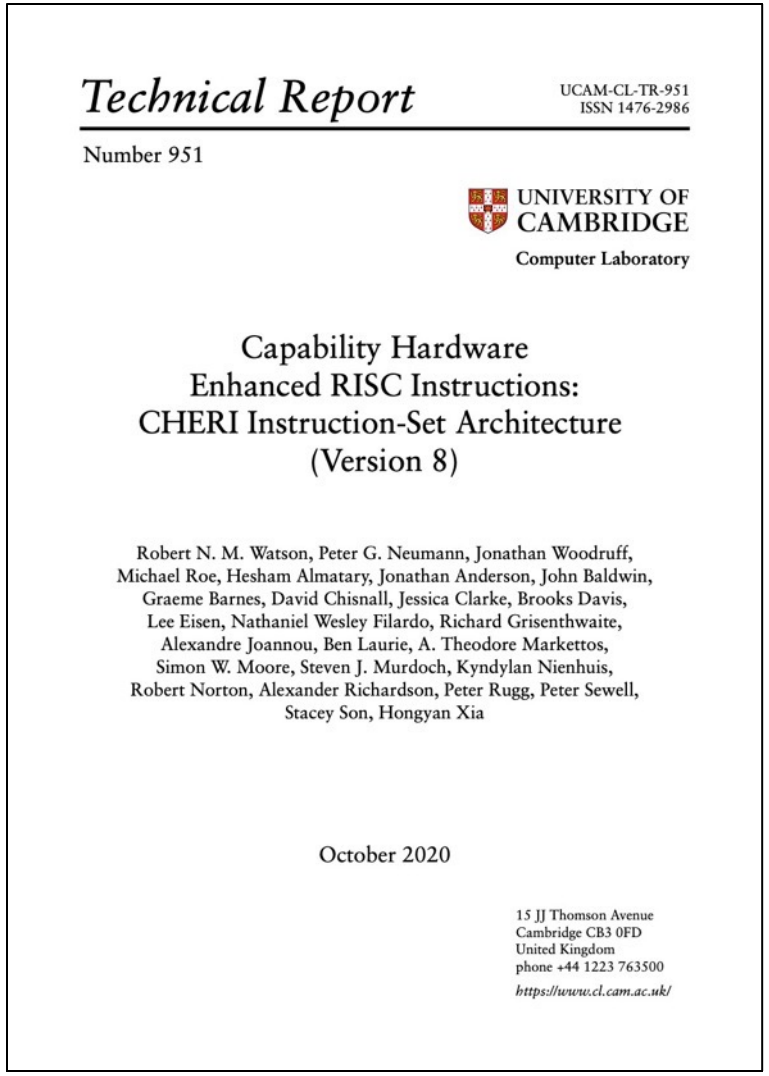
**Years 1-2**: Research platform, prototype architecture

**Years 2-4**: Hybrid C/OS model, compartment model

**Years 4-7**: Efficiency, CheriABI/C/C++/linker, ARMv8-A

**Years 8-12:** RISC-V, temporal safety, proof, Arm Morello, Microsoft CHERI Ibex

# CHERI ISA refinement over 10 years



*Technical Report*

UCAM-CL-TR-951
ISSN 1476-2986

Number 951

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

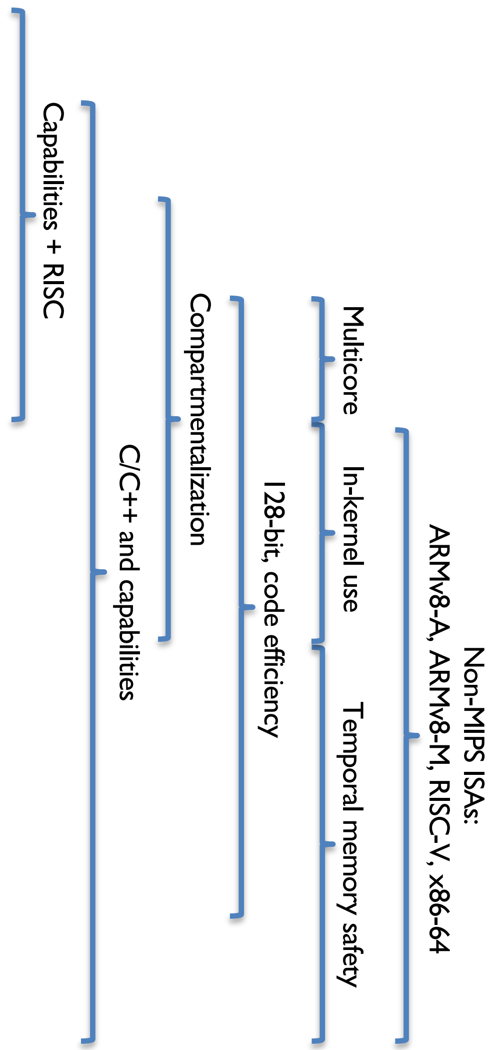
Capability Hardware
Enhanced RISC Instructions:
CHERI Instruction-Set Architecture
(Version 8)

Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff, Michael Roe, Hesham Almatary, Jonathan Anderson, John Baldwin, Graeme Barnes, David Chisnall, Jessica Clarke, Brooks Davis, Lee Eisen, Nathaniel Wesley Filardo, Richard Grisenthwaite, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, Simon W. Moore, Steven J. Murdoch, Kyndylan Nienhuis, Robert Norton, Alexander Richardson, Peter Rugg, Peter Sewell, Stacey Son, Hongyan Xia

October 2020

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

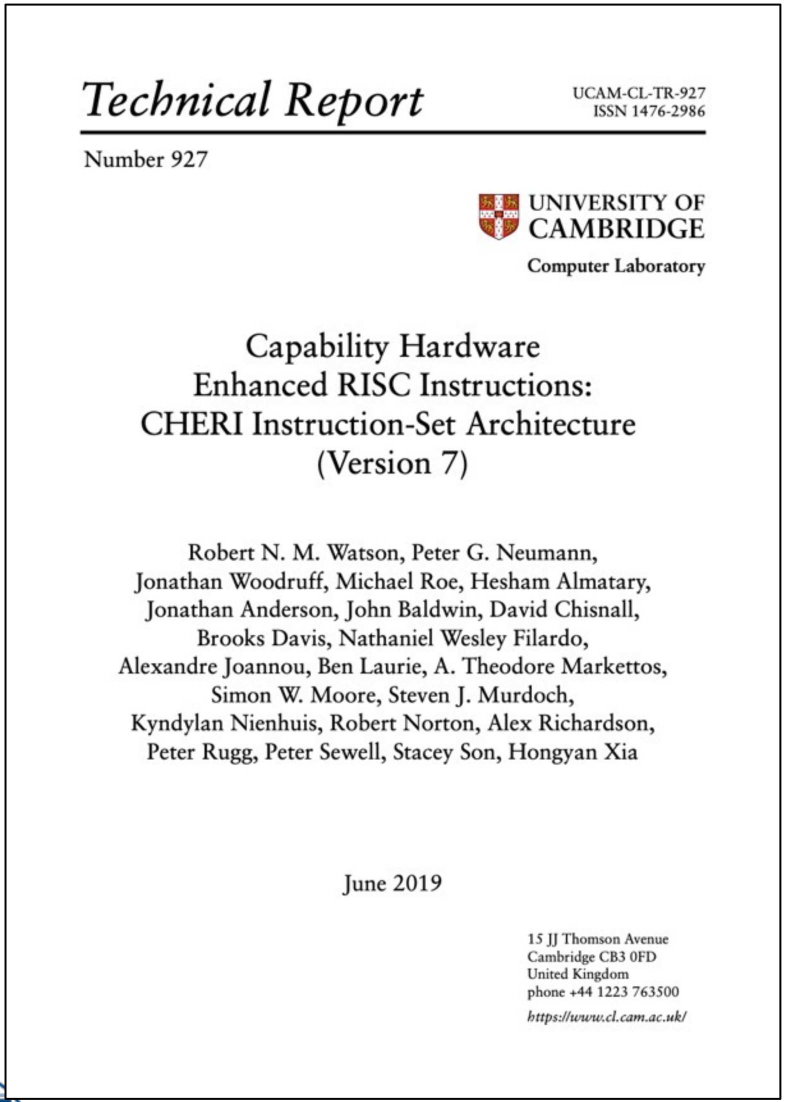
| Year | Version | Description |
|---|---|---|
| 2010-2012 | ISAv1 | RISC capability-system model w/64-bit MIPS<br>Capability registers, tagged memory<br>Guarded manipulation of registers |
| 2012 | ISAv2 | Extended tagging to capability registers<br>Capability-aware exception handling<br>Boots an MMU-based OS with CHERI support |
| 2014 | ISAv3 | Fat pointers + capabilities, compiler support<br>Instructions to optimize hybrid code<br>Sealed capabilities, CCall/CReturn |
| 2015 | ISAv4 | MMU-CHERI integration (TLB permissions)<br>ISA support for compressed 128-bit capabilities<br>HW-accelerated domain switching<br>Multicore instructions: full suite of LL/SC variants |
| 2016 | ISAv5 | CHERI-128 compressed capability model<br>Improved generated code efficiency<br>Initial in-kernel privilege limitations |
| 2017 | ISAv6 | Mature kernel privilege limitations<br>Further generated code efficiency<br>Architectural portability: CHERI-x86, CHERI-RISC-V sketches<br>Exception-free domain transition |
| 2019 | ISAv7 | Architectural performance optimization for C++ applications<br>Microarchitectural side-channel resistance features<br>Architecture-neutral CHERI protection model<br>All instruction pseudocode from a formal model<br>CHERI Concentrate capability compression<br>Improved C-language support, dynamic linking, sentry capabilities<br>Elaborated CHERI-RISC-V ISA<br>64-bit capabilities for 32-bit architectures<br>Accelerated tag operations for temporal memory safety |
| 2020 | ISAv8 | MMU temporal memory-safety assist; e.g., capability dirty bit<br>Optimizations for sentry capabilities<br>CHERI-RISC-V privileged support, general maturity<br>Further C-language semantics improvements |



Arm Morello architecture synchronization point

Watson, et al. **Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 8)**, UCAM-CL-TR-951, October 2020.

# CHERI ISAv7 – June 2019



Technical Report

UCAM-CL-TR-927
ISSN 1476-2986

Number 927

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Capability Hardware
Enhanced RISC Instructions:
CHERI Instruction-Set Architecture
(Version 7)

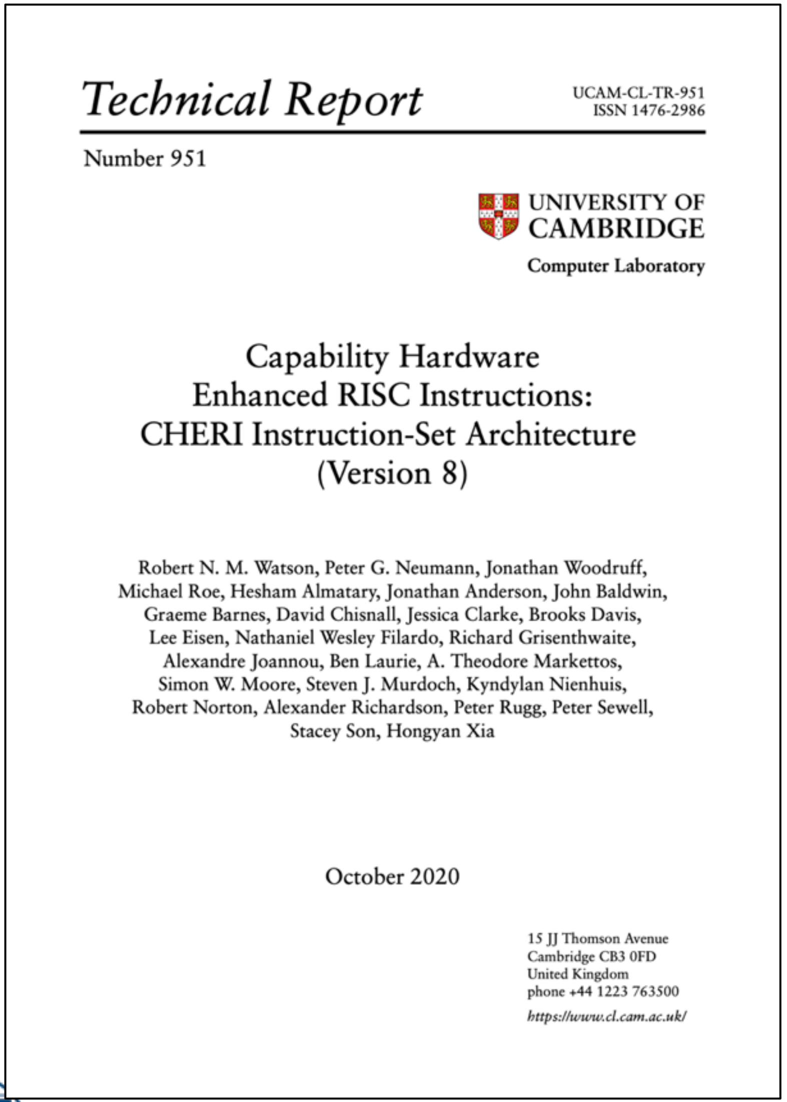
Robert N. M. Watson, Peter G. Neumann,
Jonathan Woodruff, Michael Roe, Hesham Almatary,
Jonathan Anderson, John Baldwin, David Chisnall,
Brooks Davis, Nathaniel Wesley Filardo,
Alexandre Joannou, Ben Laurie, A. Theodore Markettos,
Simon W. Moore, Steven J. Murdoch,
Kyndylan Nienhuis, Robert Norton, Alex Richardson,
Peter Rugg, Peter Sewell, Stacey Son, Hongyan Xia

June 2019

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

- First CHERI ISA spec release in two years
- Key features:
  - Architecture-neutral CHERI model
  - Elaborated CHERI-RISC-V ISA
  - CHERI Concentrate capability compression (**IEEE TC 2019**)
  - Side-channel resistance features
  - Improved C-language compatibility, dynamic linkage, performance optimizations (**ASPLOS 2019**)
  - Experimental features including 64-bit capabilities for 32-bit architectures (**ICCD 2018**), temporal safety (**IEEE Micro 2019, IEEE SSP 2020**)
  - All instruction pseudocode derived from Sail formal models, formally proven properties (**IEEE SSP 2020**)

# CHERI ISAv8 (October 2020)


*Technical Report*

UCAM-CL-TR-951
ISSN 1476-2986

Number 951

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Capability Hardware
Enhanced RISC Instructions:
CHERI Instruction-Set Architecture
(Version 8)

Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff, Michael Roe, Hesham Almatary, Jonathan Anderson, John Baldwin, Graeme Barnes, David Chisnall, Jessica Clarke, Brooks Davis, Lee Eisen, Nathaniel Wesley Filardo, Richard Grisenthwaite, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, Simon W. Moore, Steven J. Murdoch, Kyndylan Nienhuis, Robert Norton, Alexander Richardson, Peter Rugg, Peter Sewell, Stacey Son, Hongyan Xia

October 2020

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

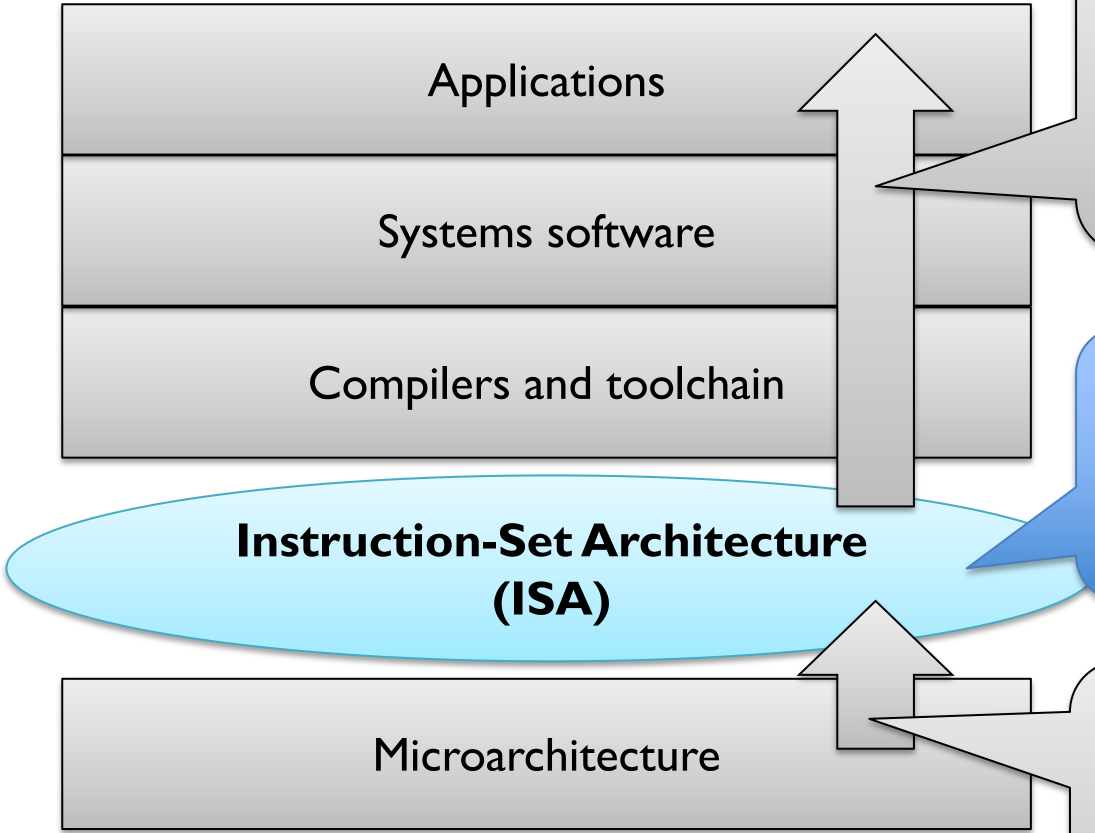
- 590 pages specifying CHERI-MIPS, CHERI-RISC-V
- Key changes
  - Capability compression is now part of the abstract protection model
  - Both 32-bit and 64-bit architectural address sizes are supported
  - Various experimental features are now mature: Sentry capabilities, CHERI-RISC-V
  - New MMU temporal memory-safety mechanisms based on load-side barrier model
  - CHERI microarchitecture chapter
- **Synchronized with Arm Morello**

## CHERI ISAv9 (2022 Q4?)

- Increasing CHERI-RISC-V maturity
  - Control-flow improvements to reduce function call bloat
  - Compressed instruction support
  - Shift away from exception generation to tag clearing
  - Sundry clarifications
- CHERI-MIPS removed
- Substantially more detailed CHERI-x86 sketch
- Further information on CHERI microarchitecture

## CHERI PROTECTION MODEL AND ARCHITECTURE

# Architectural primitives for software security



Software configures and uses capabilities to continuously enforce safety properties such as **referential, spatial, and temporal memory safety,** as well as higher-level security constructs such as **compartment isolation**

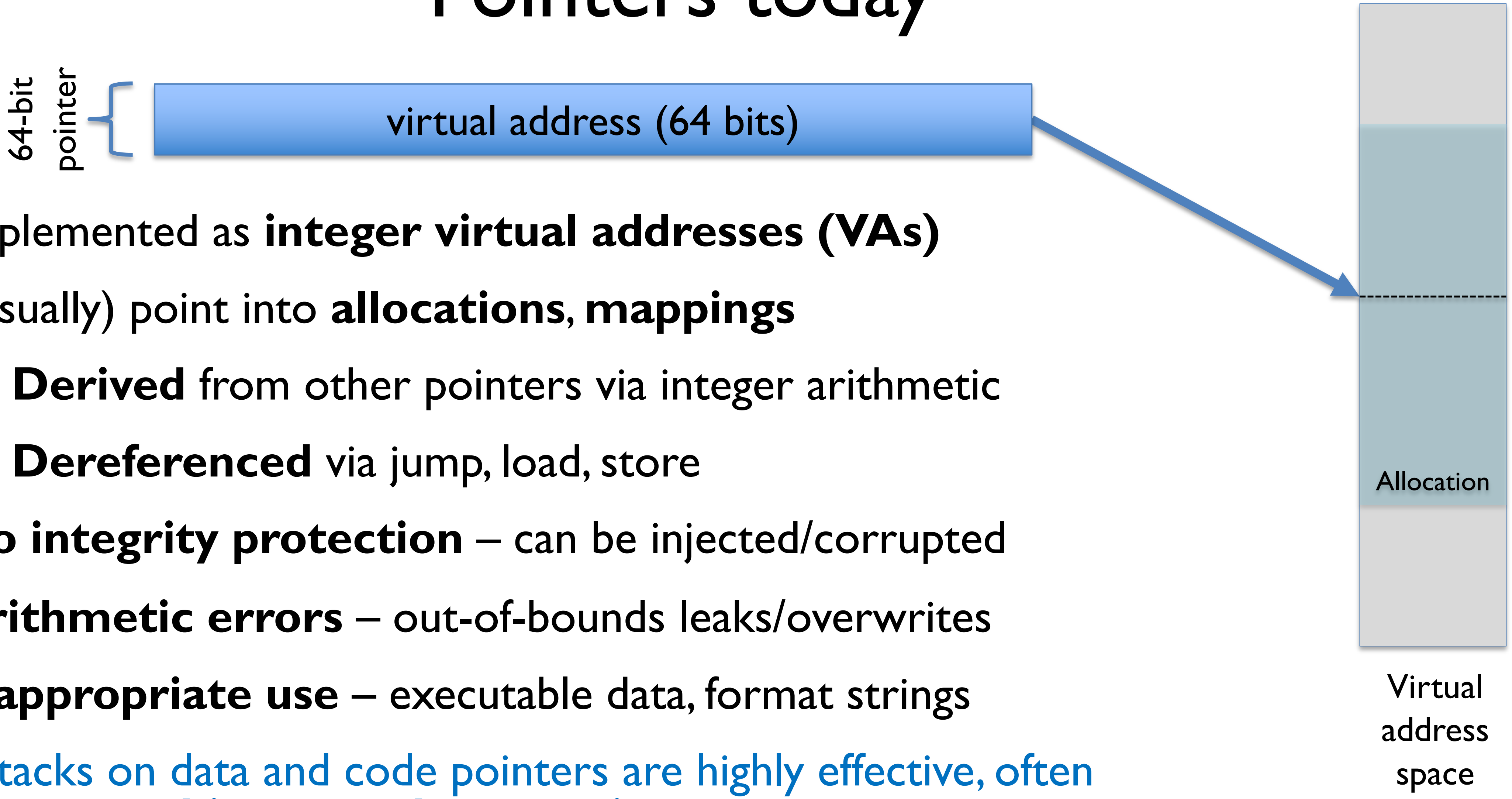
**CHERI capabilities** are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

The microarchitecture implements the **capability data type** and **tagged memory**, enforcing invariants on their manipulation and use such as **capability bounds**, **monotonicity**, and **provenance validity**

## CHERI design goals and approach

- **De-conflate memory virtualization and protection**
  - Memory Management Units (MMUs) protect by **location (address)**
  - CHERI protects existing **references (pointers)** to code, data, objects
  - Reusing **existing pointer indirection** avoids adding new architectural table lookups
- **Architectural mechanism** that enforces **software policies**
  - **Language-based properties** – e.g., referential, spatial, and temporal integrity (C/C++ compiler, linkers, OS model, runtime, …)
  - **New software abstractions** – e.g., software compartmentalization (confined objects for in-address-space isolation, …)

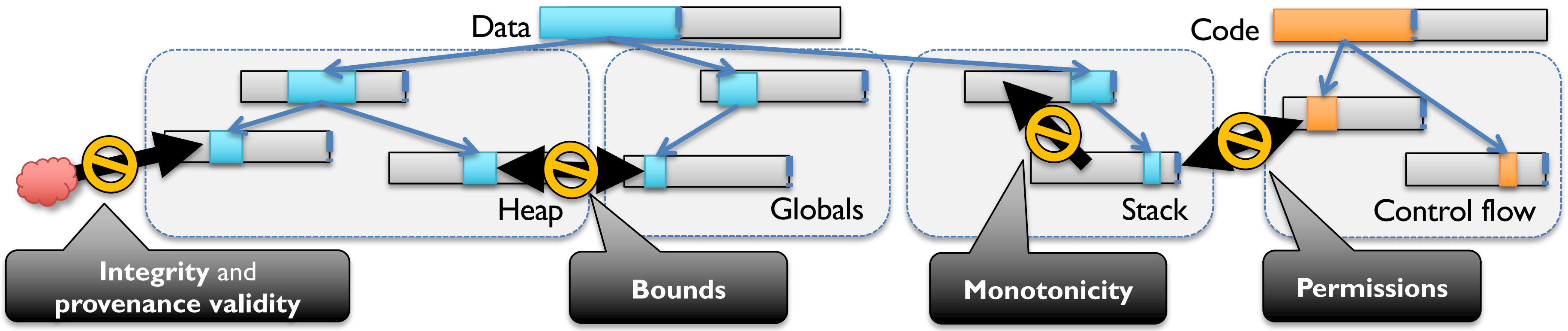
# Pointers today



- Implemented as **integer virtual addresses (VAs)**
- (Usually) point into **allocations**, **mappings**
  - **Derived** from other pointers via integer arithmetic
  - **Dereferenced** via jump, load, store
- **No integrity protection** – can be injected/corrupted
- **Arithmetic errors** – out-of-bounds leaks/overwrites
- **Inappropriate use** – executable data, format strings

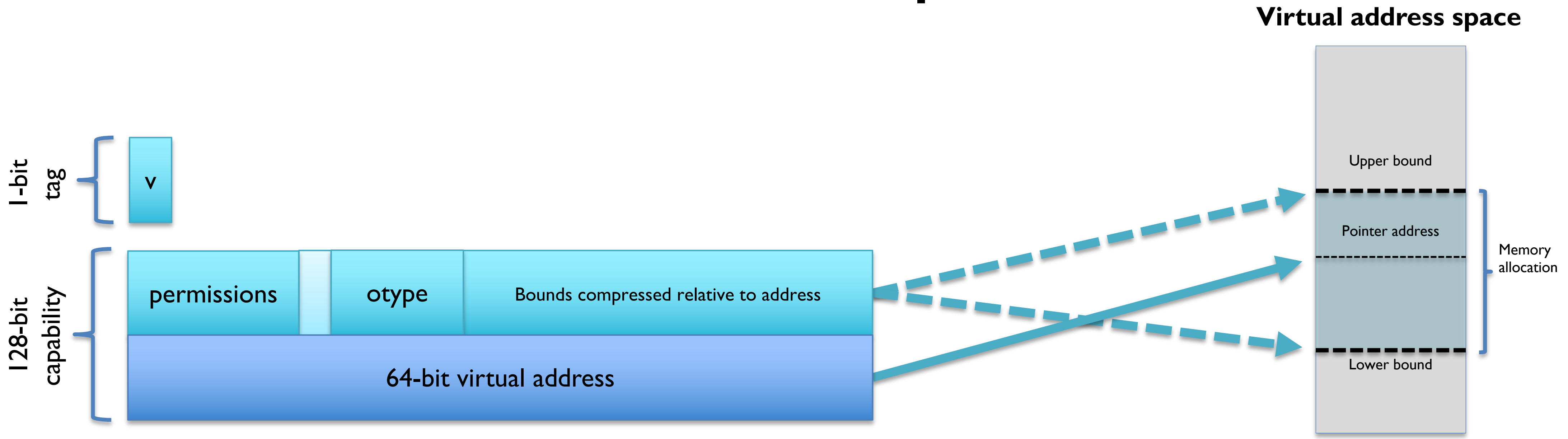
➢ Attacks on data and code pointers are highly effective, often achieving **arbitrary code execution**

# CHERI enforces protection semantics for pointers



- **Integrity** and **provenance validity** ensure that valid pointers are derived from other valid pointers via valid transformations; **invalid pointers cannot be used**
  - Valid pointers, once removed, cannot be reintroduced solely unless rederived from other valid pointers
  - E.g., Received network data cannot be interpreted as a code/data pointer – even previously leaked pointers
- **Bounds** prevent pointers from being manipulated to access the wrong object
  - Bounds can be minimized by software – e.g., stack allocator, heap allocator, linker
- **Monotonicity** prevents pointer privilege escalation – e.g., broadening bounds
- **Permissions** limit unintended use of pointers; e.g., W^X for pointers
- These primitives not only allow us to implement **strong spatial and temporal memory protection**, but also higher-level policies such as **scalable software compartmentalization**

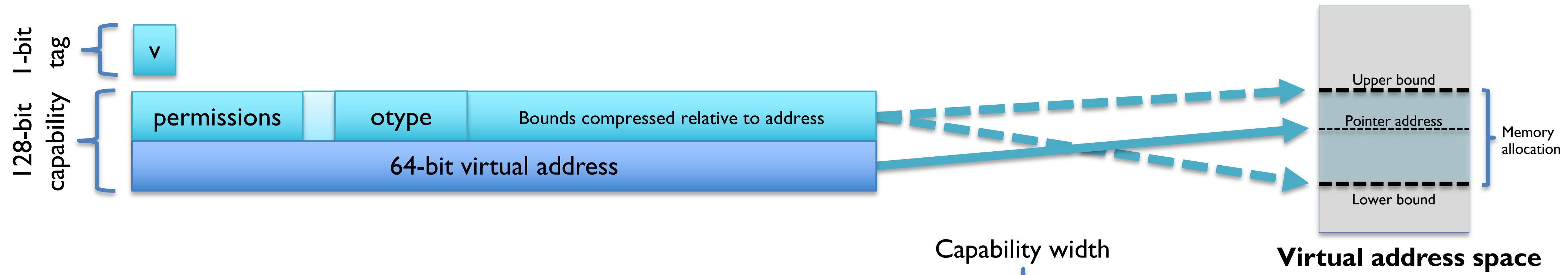
# CHERI 128-bit capabilities

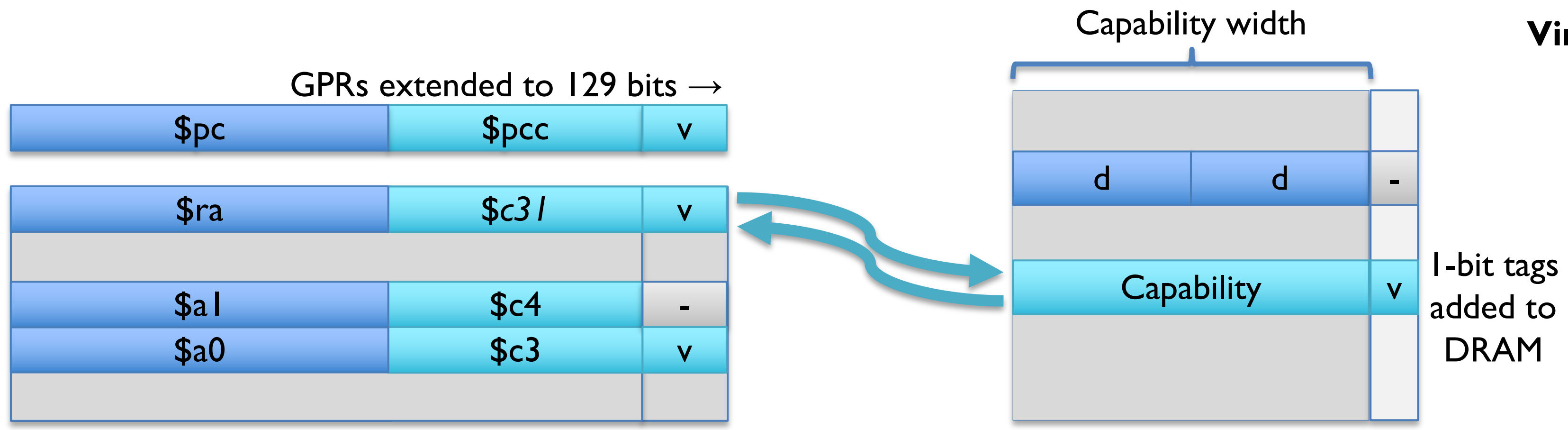


- **Capabilities** extend **integer memory addresses**
- **Metadata** (bounds, permissions, …) control how they may be used
- **Guarded manipulation** controls how capabilities may be manipulated; e.g., **provenance validity** and **monotonicity**
- **Tags** protect capability integrity/derivation in registers + memory

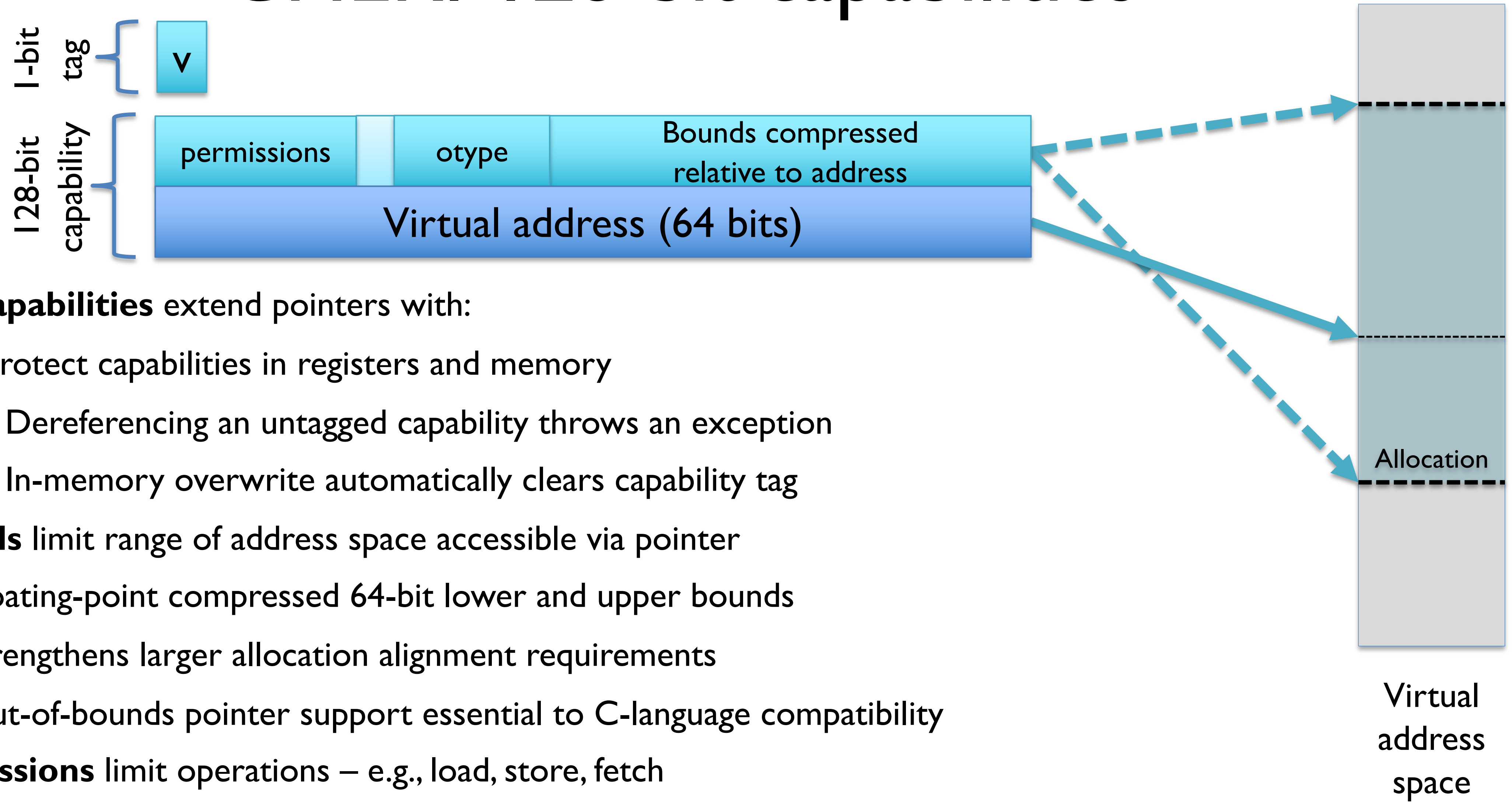
# CHERI 128-bit capabilities

- **CHERI capabilities** are a new architectural data type extending integer addresses
- **Capability metadata** (bounds, permissions, …) control how a capability may be used
- **Capability tags** protect the integrity + safe derivation of capabilities in registers and memory





# CHERI 128-bit capabilities

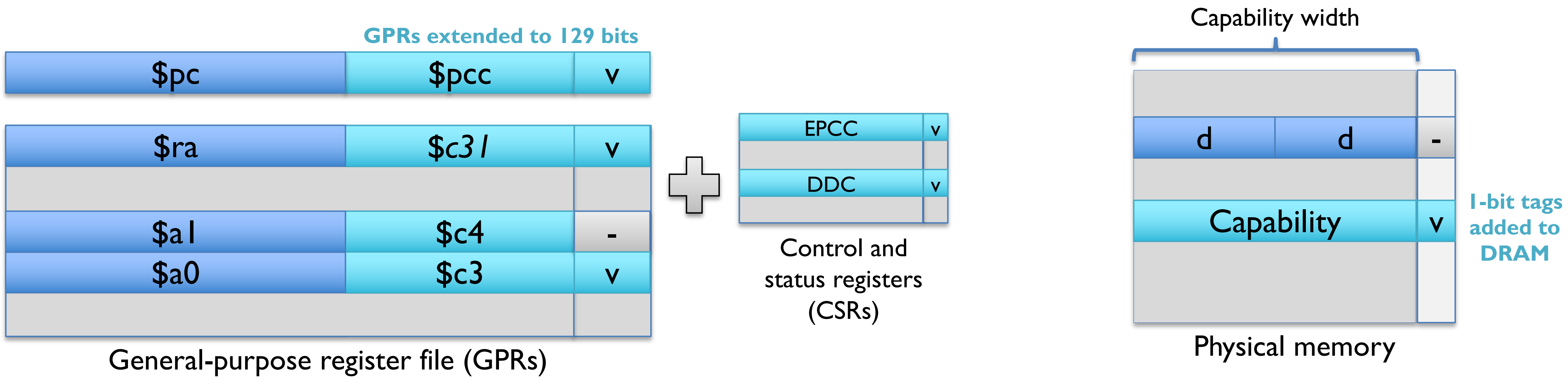


**CHERI capabilities** extend pointers with:

- **Tags** protect capabilities in registers and memory
  - Dereferencing an untagged capability throws an exception
  - In-memory overwrite automatically clears capability tag
- **Bounds** limit range of address space accessible via pointer
  - Floating-point compressed 64-bit lower and upper bounds
  - Strengthens larger allocation alignment requirements
  - Out-of-bounds pointer support essential to C-language compatibility
- **Permissions** limit operations – e.g., load, store, fetch
- **Sealing**: **immutable**, **non-dereferenceable capabilities** – used for non-monotonic transitions

# Merged capability register file + tagged memory

(as found in Morello and CHERI-RISC-V; MIPS used a split register file)

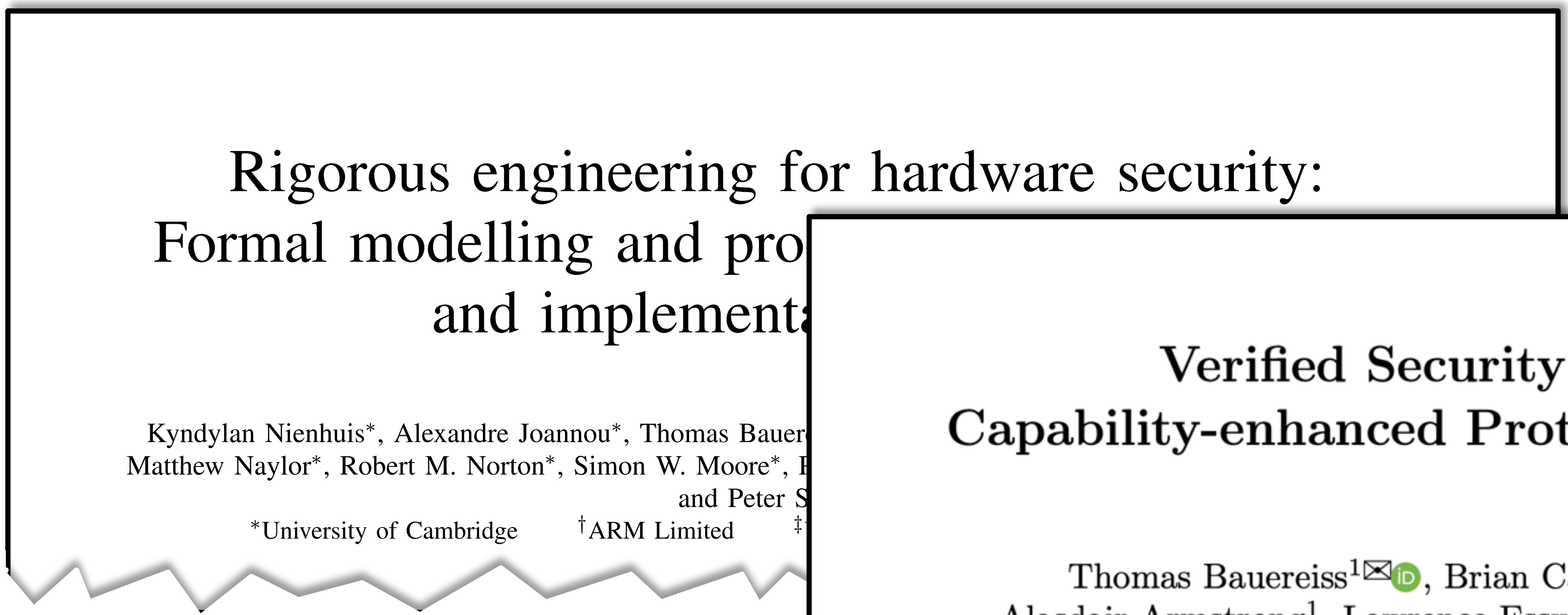


- **64-bit general-purpose registers (GPRs)** are extended with **64 bits of metadata** and a **1-bit validity tag**
- **Program counter (PC)** is extended to be the **program-counter capability ($PCC)**
- **Default data capability ($DDC)** constrains legacy integer-relative ISA load and store instructions
- **Tagged memory** protects capability-sized and -aligned words in DRAM by adding a **1-bit validity tag**
- **Various system mechanisms** are extended (e.g., capability-instruction enable control register, new TLB/PTE permission bits, exception code extensions, saved exception stack pointers and vectors become capabilities, etc.)

## CHERI-RISC-V formal ISA model

- CHERI RISC-V ISA model extends RISC-V formal ISA specification, in Sail
- Sail RISC-V ISA specification developed by UCam + SRI
  - Selected as official RISC-V spec by the Foundation
  - Sail is a custom first-order imperative language for expressing ISA specifications, usable by engineers but with static type checking of bitvector lengths etc.
  - The Sail spec is inlined in versions of the unprivileged and privileged RISC-V manuals
  - Sail auto-generates a C emulator, theorem-prover definitions, and SMT definitions
  - Machinery for configuring model WRT YAML from compliance group
  - Readable, precise definition of ISA behavior, usable as test oracle for testing hardware against and for software bring-up, and providing prover definitions if you want more rigorous reasoning
- Paper on earlier CHERI-MIPS L3 modelling and proof work at IEEE SSP 2020
- Most recently completed monotonicity proofs for the Arm Morello architecture

# ISA formal modelling and verification

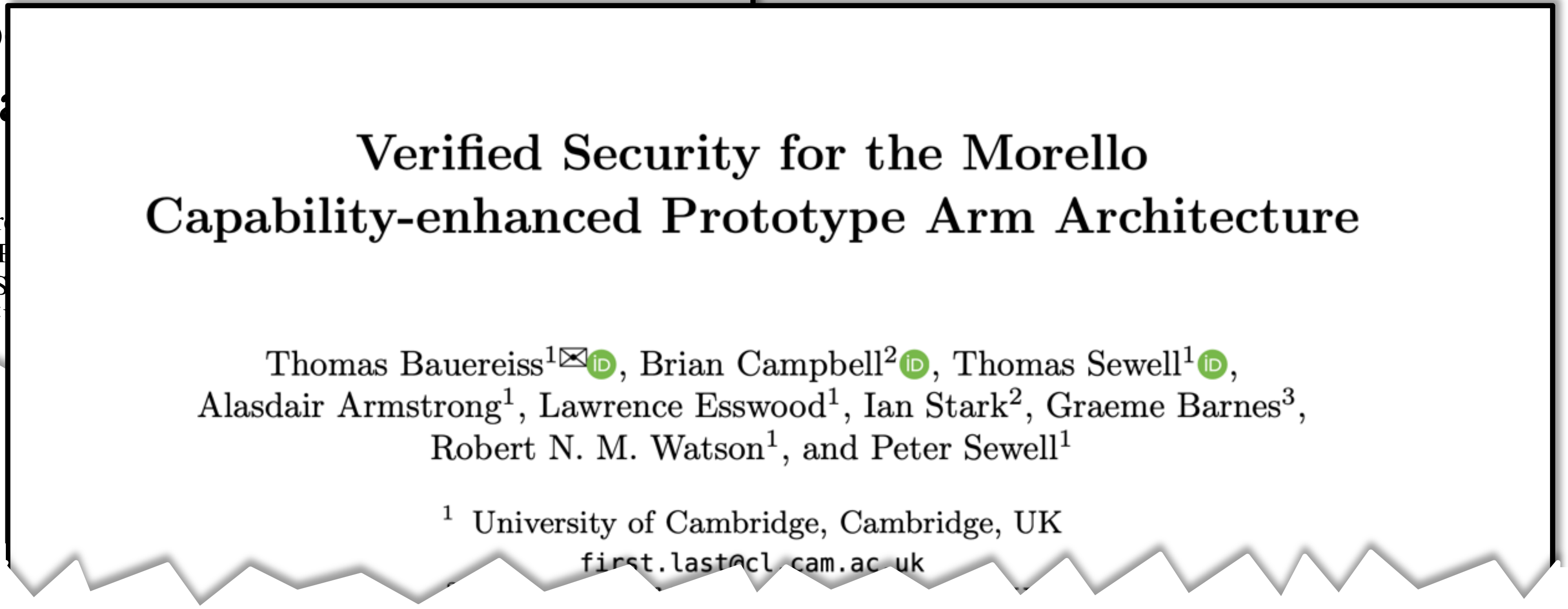


Rigorous engineering for hardware security:
Formal modelling and pro
and implement

Kyndylan Nienhuis[*], Alexandre Joannou[*], Thomas Bauer
Matthew Naylor[*], Robert M. Norton[*], Simon W. Moore[*], F
and Peter S
[*]University of Cambridge [†]ARM Limited [‡]

IEEE SSP 2020

ESOP 2022



Verified Security for the Morello
Capability-enhanced Prototype Arm Architecture

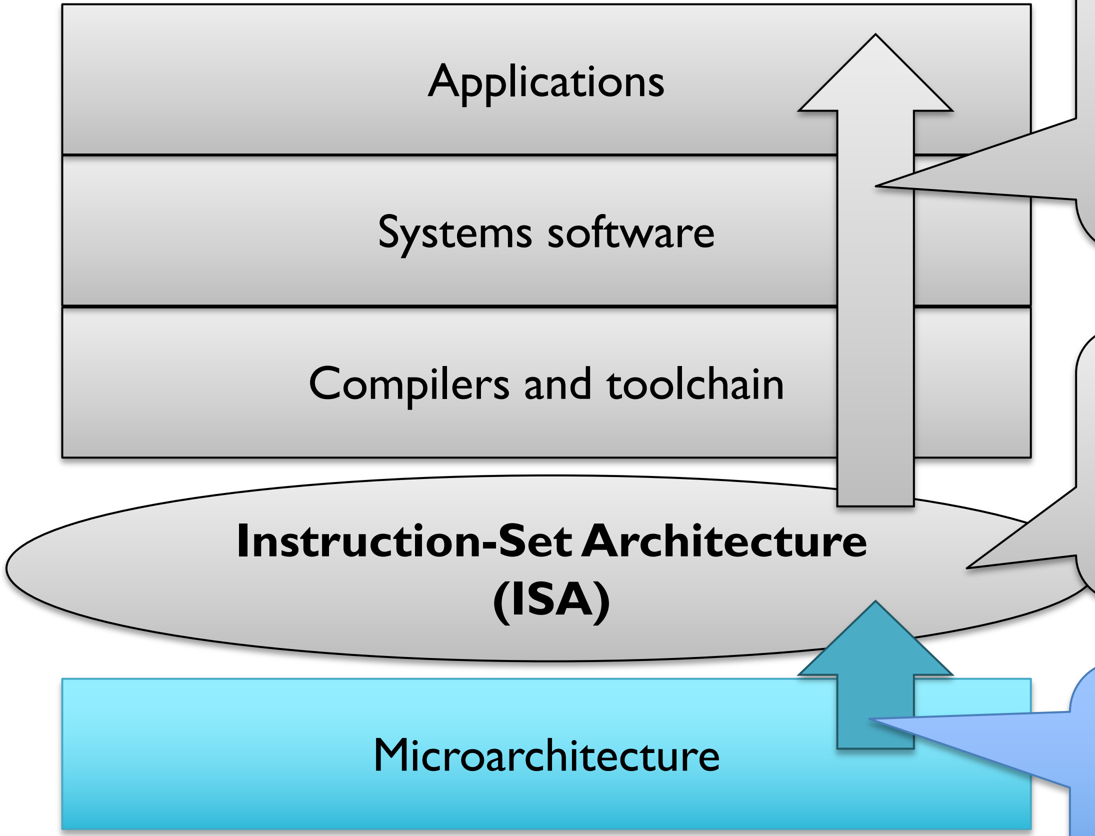
Thomas Bauereiss[1]✉, Brian Campbell[2], Thomas Sewell[1],
Alasdair Armstrong[1], Lawrence Esswood[1], Ian Stark[2], Graeme Barnes[3],
Robert N. M. Watson[1], and Peter Sewell[1]

[1] University of Cambridge, Cambridge, UK
first.last@cl.cam.ac.uk

- Formal ISA models CHERI-MIPS, CHERI-RISC-V, and Morello
- Formal proof of compartmentalization for CHERI-MIPS, Morello

## CHERI MICROARCHITECTURE AND PROTOTYPES

## Architectural primitives for software security



Software configures and uses capabilities to continuously enforce safety properties such as **referential, spatial, and temporal memory safety,** as well as higher-level security constructs such as **compartment isolation**

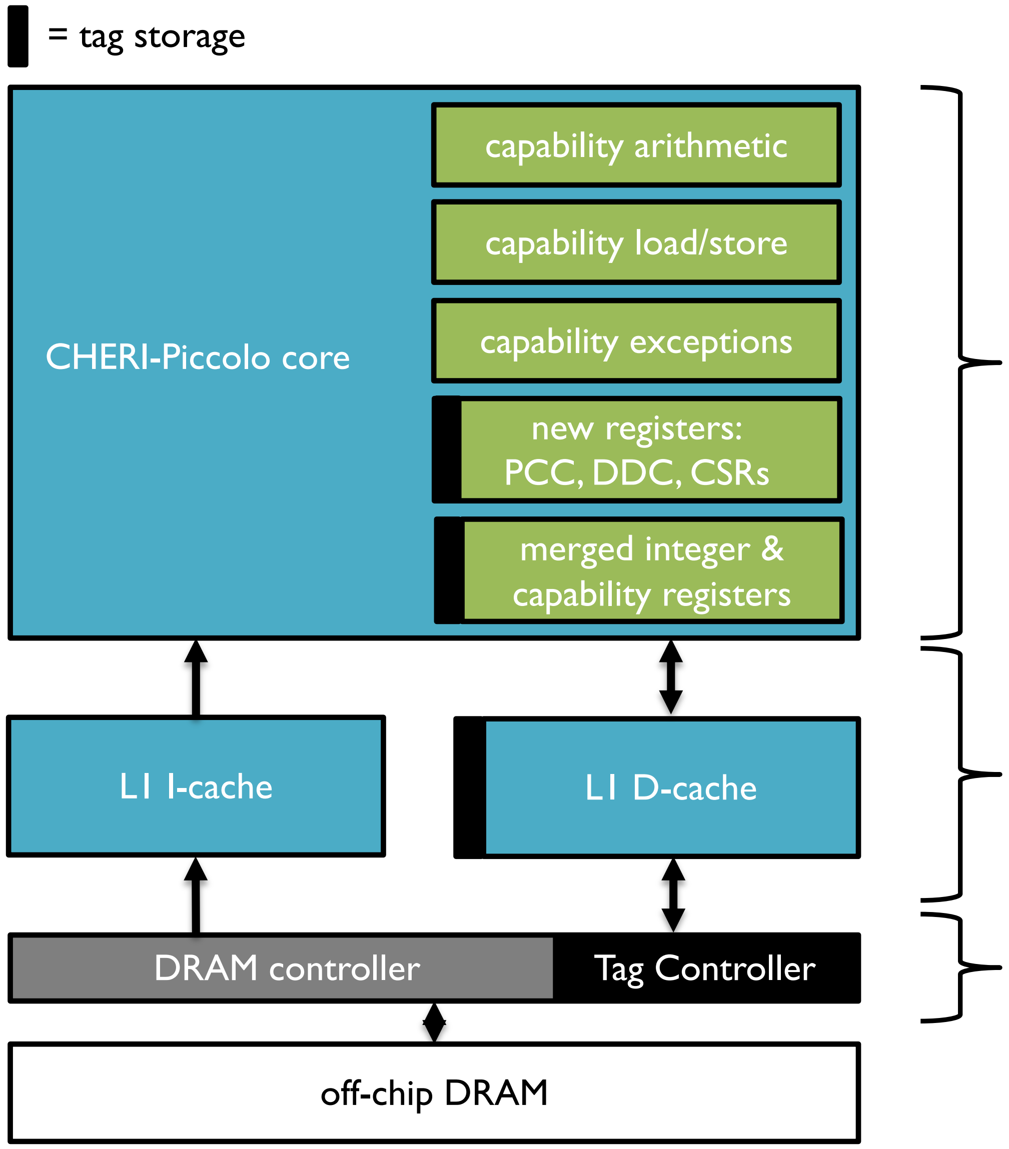
**CHERI capabilities** are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

The microarchitecture implements the **capability data type** and **tagged memory**, enforcing invariants on their manipulation and use such as **capability bounds**, **monotonicity**, and **provenance validity**

# CHERI hardware prototypes

- Original research based on our home-grown pipelined BERI MIPS core (CHERI-MIPS)
- We have transitioned our CHERI research to extended versions of open-source off-the-shelf BSV RISC-V cores (CHERI-RISC-V)
  - CHERI-Piccolo 3-stage pipeline, 32-bit, no MMU
  - CHERI-Flute 5-stage pipeline, 32- or 64-bit, MMU
  - CHERI-Toooba Superscalar, 64-bit, MMU
- Novel microarchitectural contributions include **capability compression model**, **tagged memory implementation techniques**
- All of our CPU designs are open source
- We also provide a Qemu full-system and userlevel simulators for CHERI-RISC-V
- **Arm Morello and Microsoft CHERI Ibex (later slides)**

# Example microarchitecture: CHERI-Piccolo microcontroller



Changes to the Piccolo core (RISC-V 3-stage pipeline):

- capability arithmetic
- capability load/store operations with bounds checking
- extended exception model
- PC becomes a capability (PCC)
- default data capability (DDC)
- new control/status registers
- merged integer & capability register file

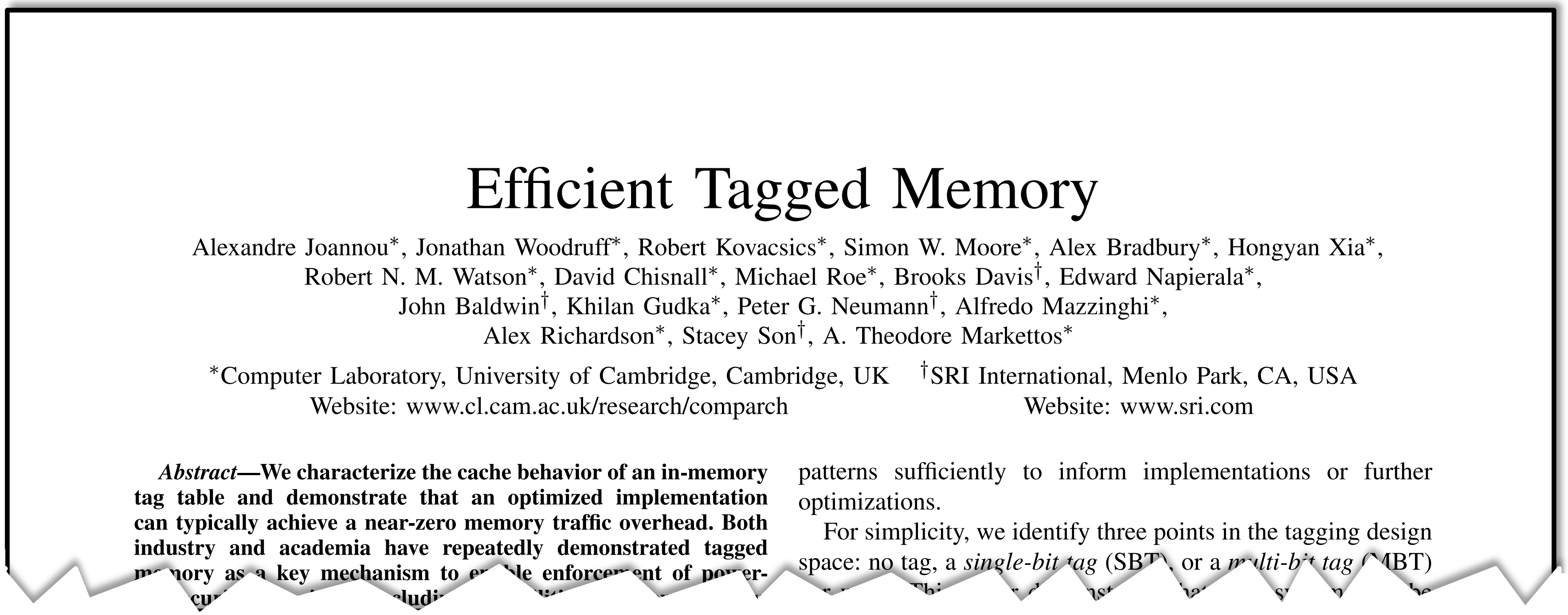
Memory subsystem:

- AXI user-field added to transport tag bits & data width doubled
- caches extended to include tags

DRAM changes:

- New tag controller uses a hierarchical tag table to efficiently store tag bits backed by top of DRAM

# Microarchitectural tag storage for off-the-shelf DRAM



## Efficient Tagged Memory

Alexandre Joannou*, Jonathan Woodruff*, Robert Kovacsics*, Simon W. Moore*, Alex Bradbury*, Hongyan Xia*, Robert N. M. Watson*, David Chisnall*, Michael Roe*, Brooks Davis†, Edward Napierala*, John Baldwin†, Khilan Gudka*, Peter G. Neumann†, Alfredo Mazzinghi*, Alex Richardson*, Stacey Son†, A. Theodore Markettos*

*Computer Laboratory, University of Cambridge, Cambridge, UK
Website: www.cl.cam.ac.uk/research/comparch

†SRI International, Menlo Park, CA, USA
Website: www.sri.com

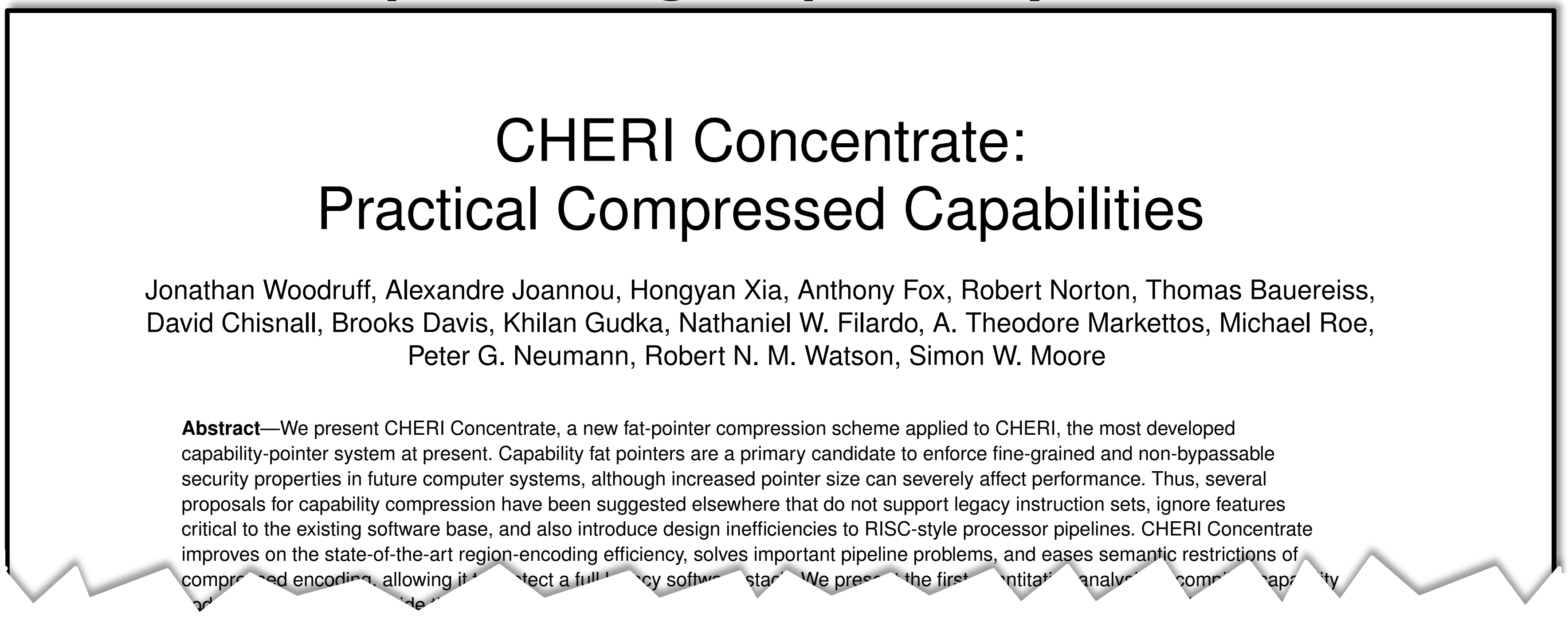
*Abstract*—**We characterize the cache behavior of an in-memory tag table and demonstrate that an optimized implementation can typically achieve a near-zero memory traffic overhead. Both industry and academia have repeatedly demonstrated tagged memory as a key mechanism to enable enforcement of power-**

patterns sufficiently to inform implementations or further optimizations.

For simplicity, we identify three points in the tagging design space: no tag, a *single-bit tag* (SBT), or a *multi-bit tag* (MBT)

- Published in the IEEE International Conference on Computer Design (ICCD) 2017
- Shift from flat to hierarchal tag table to hold tags in DRAM
  - Exploit inconsistent density of tags in physical memory
  - Reduces DRAM access overhead for a variety of workloads

# Compressing capability bounds



CHERI Concentrate:
Practical Compressed Capabilities

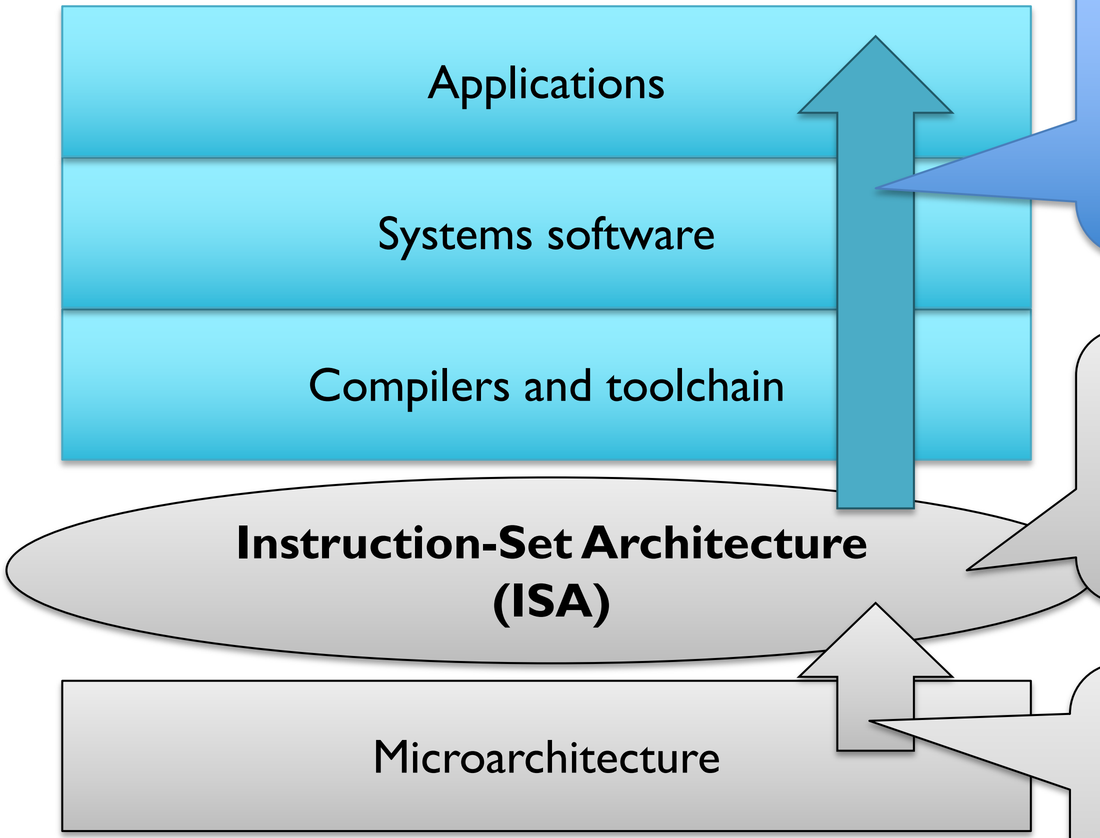
Jonathan Woodruff, Alexandre Joannou, Hongyan Xia, Anthony Fox, Robert Norton, Thomas Bauereiss, David Chisnall, Brooks Davis, Khilan Gudka, Nathaniel W. Filardo, A. Theodore Markettos, Michael Roe, Peter G. Neumann, Robert N. M. Watson, Simon W. Moore

**Abstract**—We present CHERI Concentrate, a new fat-pointer compression scheme applied to CHERI, the most developed capability-pointer system at present. Capability fat pointers are a primary candidate to enforce fine-grained and non-bypassable security properties in future computer systems, although increased pointer size can severely affect performance. Thus, several proposals for capability compression have been suggested elsewhere that do not support legacy instruction sets, ignore features critical to the existing software base, and also introduce design inefficiencies to RISC-style processor pipelines. CHERI Concentrate improves on the state-of-the-art region-encoding efficiency, solves important pipeline problems, and eases semantic restrictions of compr ed encoding, allowing it t tect a full l cy softw stac We pres the first ntitati analy comp capa ity

- Published in IEEE Transactions on Computers, April 2019
- Efficient compressed capabilities for 32-bit and 64-bit processors
  - Reduces size of capabilities from 4x machine word size to 2x
  - Large reduction in cache overheads
  - Efficiently fits into a RISC pipeline with negligible impact on clock frequency
  - Maintains all security and software compatibility properties

## HOW SOFTWARE WORKS ON CHERI

# Architectural primitives for software security



Software configures and uses capabilities to continuously enforce safety properties such as **referential, spatial, and temporal memory safety,** as well as higher-level security constructs such as **compartment isolation**

**CHERI capabilities** are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

The microarchitecture implements the **capability data type** and **tagged memory**, enforcing invariants on their manipulation and use such as **capability bounds**, **monotonicity**, and **provenance validity**

# Two key applications of the CHERI primitives

1. **Efficient, fine-grained memory protection for C/C++**
   - Strong source-level compatibility, but requires recompilation
   - Deterministic and secret-free referential, spatial, and temporal memory safety
   - Retrospective studies estimate ⅔ of memory-safety vulnerabilities mitigated
   - Generally modest overhead (0%-5%, some pointer-dense workloads higher)
2. **Scalable software compartmentalization**
   - Multiple software operational models from objects to processes
   - Increases exploit chain length: Attackers must find and exploit more vulnerabilities
   - Orders-of-magnitude performance improvement over MMU-based techniques (<90% reduction in IPC overhead in early FPGA-based benchmarks)

# What are CHERI's implications for software?

- Efficient fine-grained **architectural memory protection** enforces:

| | |
|---|---|
| **Provenance validity**: | Q: Where do pointers come from? |
| **Integrity**: | Q: How do pointers move in practice? |
| **Bounds, permissions**: | Q: What rights should pointers carry? |
| **Monotonicity**: | Q: Can real software play by these rules? |

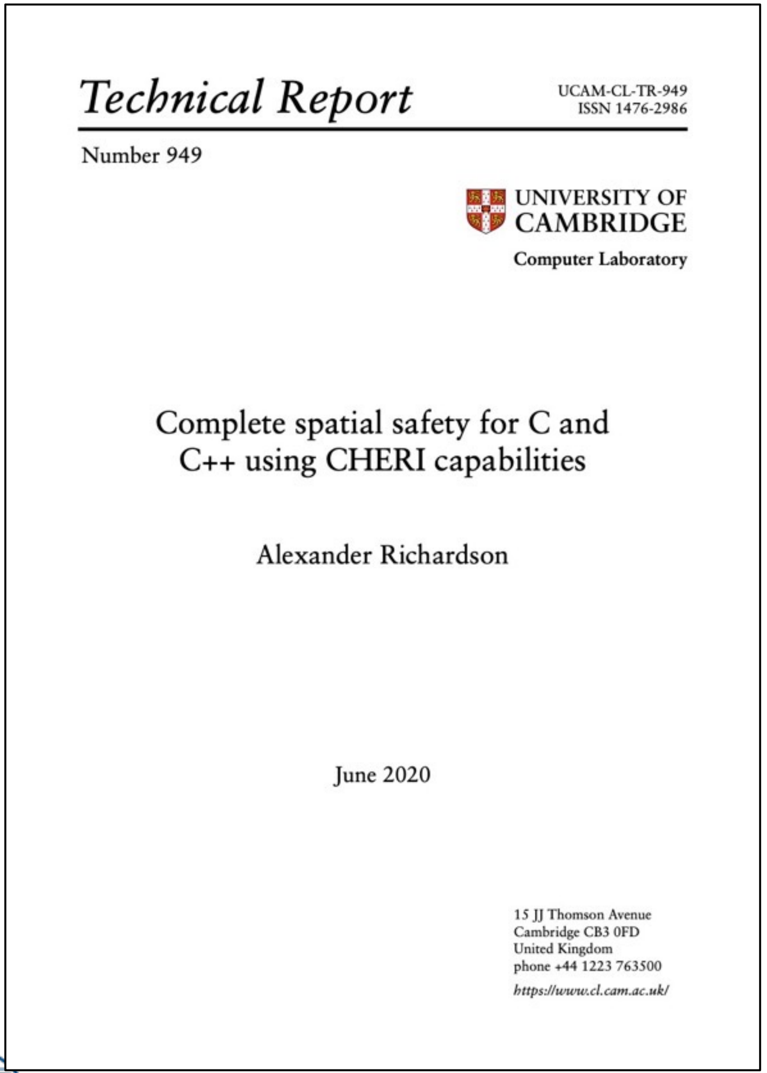
- Scalable fine-grained **software compartmentalization**

  **Q**: Can we construct **isolation** and **controlled communication** using integrity, provenance, bounds, permissions, and monotonicity?

  **Q**: Can **sealed capabilities**, **controlled non-monotonicity**, and **capability-based sharing** enable safe, efficient compartmentalization?

## CHERI C/C++ MEMORY PROTECTION

# Memory-safe CHERI C/C++



*Technical Report* UCAM-CL-TR-949
ISSN 1476-2986

Number 949

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Complete spatial safety for C and C++ using CHERI capabilities

Alexander Richardson

June 2020

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

- Capabilities used to implement all pointers
  - **Implied** – Control-flow pointers, stack pointers, GOTs, PLTs, …
  - **Explicit** – All C/C++-level pointers and references
- Strong referential, spatial, and heap temporal safety
- Minor changes to C/C++ semantics; e.g.,
  - All pointers must have well defined single provenance
  - Increased pointer size and alignment
  - Care required with integer-pointer casts and types
  - Memory-copy implementations may need to preserve tags
- Watson, et al. **CHERI C/C++ Programming Guide**, UCAM-CL-TR-947, June 2020

## Memory protection for the language and the language runtime

**Language-level memory safety**

Pointers to heap allocations

Function pointers

Pointers to global variables

Pointers to memory mappings

Pointers to stack allocations

Pointers to TLS variables

Pointers to sub-objects

Return addresses

GOT pointers

Vararg array pointers

PLT entry pointers

Stack pointers

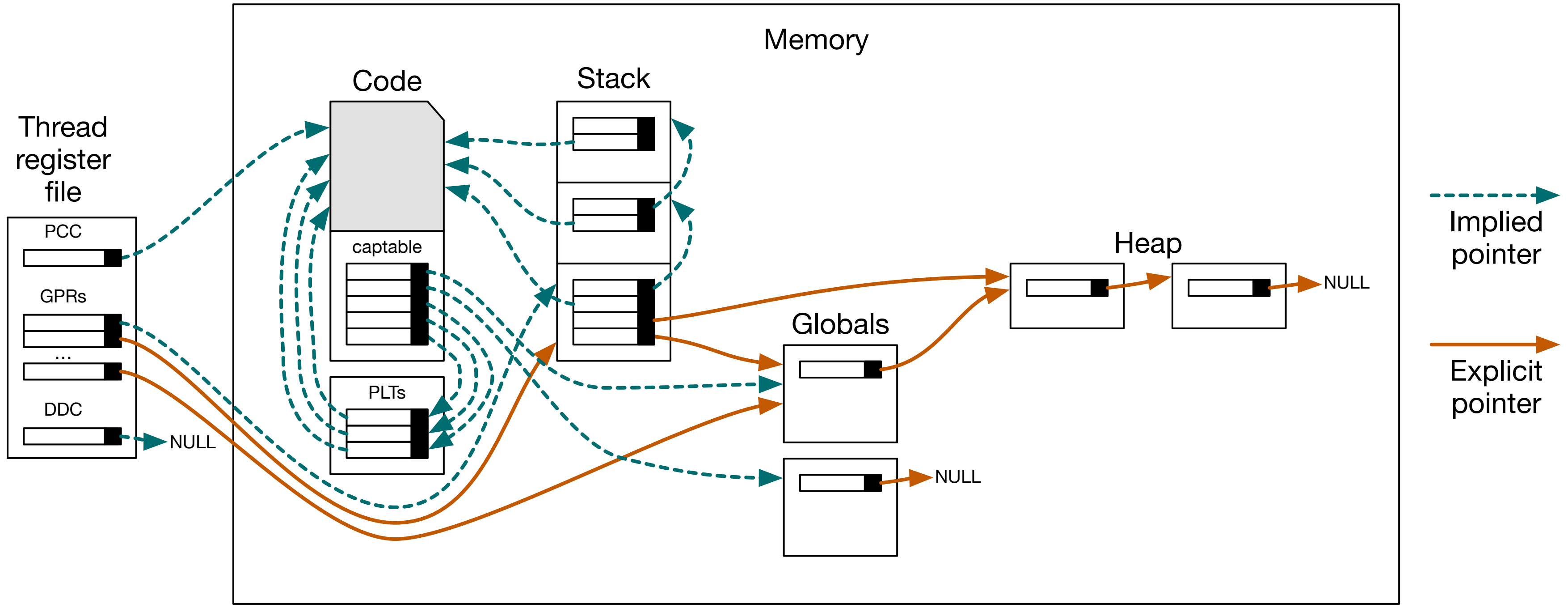
C++ v-table pointers

ELF aux arg pointers

**Sub-language memory safety**

- Capabilities are refined by the kernel, run-time linker, compiler-generated code, heap allocator, …
- Protection mechanisms:
  - Referential memory safety
  - Spatial memory safety + privilege minimization
  - Temporal memory safety
- Applied **automatically** at two levels:
  - **Language-level pointers** point explicitly at stack and heap allocations, global variables, …
  - **Sub-language pointers** used to implement control flow, linkage, etc.
- Sub-language protection mitigates bugs in the language runtime and generated code, as well as attacks that cannot be mitigated by higher-level memory safety
  - (e.g., union type confusion)

# CHERI-based pure-capability process memory



- Capabilities are substituted for integer addresses throughout the address space
- Bounds and permissions are minimized by software including the kernel, run-time linker, memory allocator, and compiler-generated code
- Hardware permits fetch, load, and store only through granted capabilities
- Tags ensure integrity and provenance validity of all pointers

# RISC-V vs. CHERI-RISC-V generated code

```
struct timezone tz;

time_t get_unix_time(void)
{
    struct timeval tv;

    gettimeofday(&tv, &tz);
    return tv.tv_sec;
}
```

```
get_unix_time_riscv:
 addi    sp, sp, -32
 sd ra, 24(sp)
 addi    a0, sp, 8
.LBB0_1:
 auipc   a1, %pcrel_hi(tz)
 addi    a1, a1, %pcrel_lo(.LBB0_1)
 call    gettimeofday
  (expands to auipc, possibly cld, cjalr)
 ld  a0, 8(sp)
 ld  ra, 24(sp)
 addi    sp, sp, 32
 ret
```

```
get_unix_time_cheririscv:
 cincoffset  csp, csp, -32
 csc     cra, 16(csp)
 cincoffset  ca0, csp, 0
 csetbounds  ca0, ca0, 16
.LBB0_1:
 auipcc  ca1, %captab_pcrel_hi(tz)
 clc ca1, %pcrel_lo(.LBB0_1)(ca1)
.LBB0_2:
 auipcc  ca2, %captab_pcrel_hi(gettimeofday)
 clc ca2, %pcrel_lo(.LBB0_2)(ca2)
 cjalr   cra, ca2
 cld     a0, 0(csp)
 clc     cra, 16(csp)
 cincoffset  csp, csp, 32
 cret
```

1. Adjust stack address/capability
2. Save return address/capability
3. Create address/capability to local ‘tv’

4. Generate address/capability to global ‘tz’

5. Call gettimeofday()

6. Load return value from ‘tv’
7. Load return address/capability
8. Restore stack address/capability
9. Return

- The general code structure is unchanged except that:
  - The integer stack pointer becomes a capability stack pointer
  - The pointer to a local stack allocation becomes capability
  - Compiler-specified bounds are set on the local variable pointer before use
  - The loaded jump target is a capability rather than an integer address

# CheriBSD: A pure-capability operating system

- Complete memory- and pointer-safe FreeBSD C/C++ kernel + userspace
  - **OS kernel**: Core OS kernel, filesystems, networking, device drivers, …
  - **System libraries**: crt/csu, ld-elf.so, libc, zlib, libxml, libssl, …
  - **System tools and daemons**: echo, sh, ls, openssl, ssh, sshd, …
  - **Applications**: PostgreSQL, nginx, WebKit (C++)
- **Valid provenance**, **minimized privilege** for **pointers, implied VAs**
  - Userspace capabilities originate in **kernel-provided roots**
  - Compiler, allocators, run-time linker, etc., **refine** bounds and perms
- Trading off **privilege minimization**, **monotonicity**, **API conformance**
  - Typically in memory management – realloc(), mmap() + mprotect()

# CHERI C compatibility: CheriBSD Code Changes

| Area | Files total | Files modified | % files | LoC total | LoC changed | % LoC |
|---|---|---|---|---|---|---|
| **Kernel** | **11,861** | **896** | **7.6** | **6,095k** | **6,961** | **0.18** |
| • Core | 7,867 | 705 | 9.0 | 3,195k | 5,787 | **0.18** |
| • Drivers | 3,994 | 191 | 4.8 | 2,900k | 1,174 | **0.04** |
| **Userspace** | **16,968** | **649** | **3.8** | **5,393k** | **2,149** | **0.04** |
| • Runtimes (excl. libc++) | 1,493 | 233 | 15.6 | 207k | 989 | **0.48** |
| • libc++ | 227 | 17 | 7.5 | 114k | 133 | **0.12** |
| • Programs and libraries | 15,475 | 416 | 2.7 | 5,186k | 1,160 | **0.02** |

**Notes:**

- Numbers from cloc counting modified files and lines for identifiable C, C++, and assembly files
- Kernel includes changes to be a hybrid program and most changes to be a pure-capability program
  - Also includes most of support for CHERI-MIPS, CHERI-RISC-V, Morello
  - Count includes partial support for 32 and 64-bit FreeBSD and Linux binaries.
  - 67 files and 25k LoC added to core in addition to modifications
  - Most generated code excluded, some existing code could likely be generated

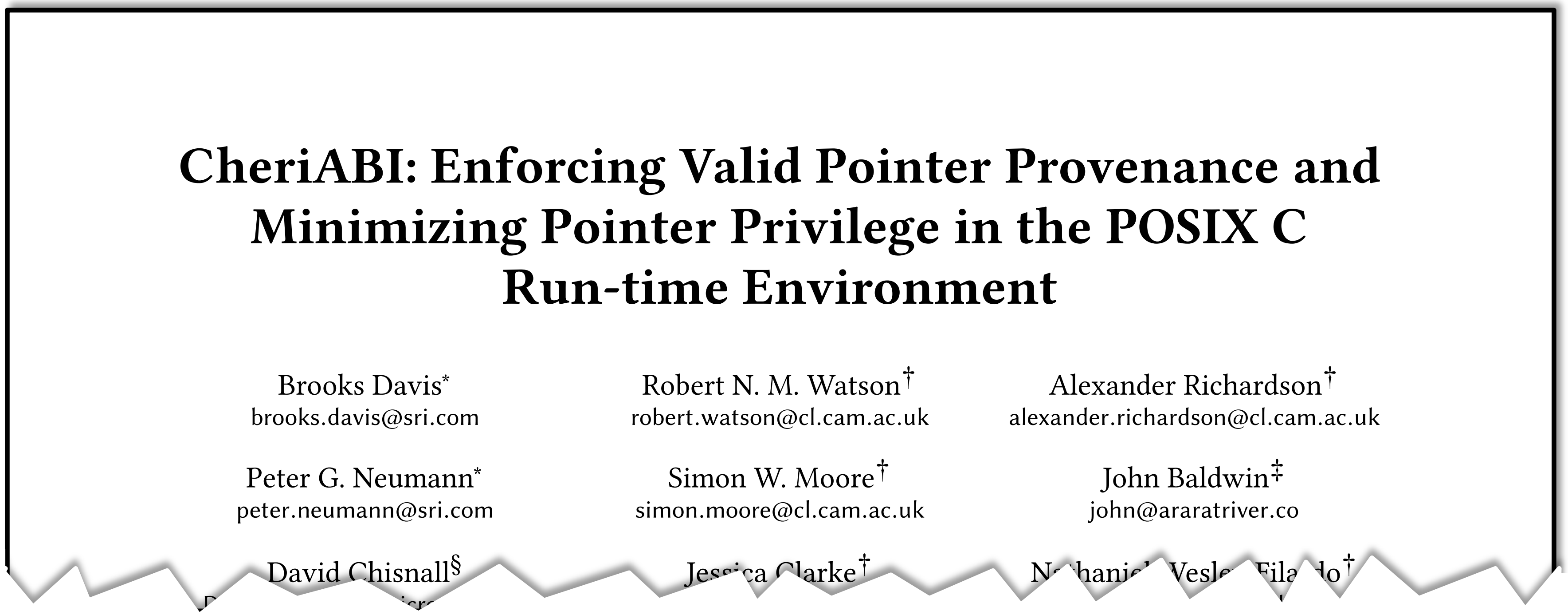
# C/C++ compatibility: WebKit - JSC Code Changes

| Area | Files total | Files modified | % Files | LoC total | LoC changed | % LoC |
|---|---|---|---|---|---|---|
| JSC-C | 3368 | 148 | 4.4 | 550k | 2217 | **0.40** |
| JSC-JIT | 3368 | 339 | 10.1 | 550k | 7581 | **1.38** |

**Notes:**

- JSC-C is a port of the C-language JavaScriptCore interpreter backend
- JSC-JIT includes support for a meta-assembly language interpreter and JIT compiler
- Runs SunSpider JavaScript benchmarks to completion
- Language runtimes represent worst-case in compatibility for CHERI
  - Porting assembly interpreter and JIT compiler requires targeting new encodings
- Changes reported here did not target diff minimization
  - Prioritized debugging and multiple configurations (including integer offsets into bounded JS heap) for performance and security evaluation
  - Some changes may not be required with modern CHERI compiler

# Pure-capability UNIX process environment



CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment

Brooks Davis[*]
brooks.davis@sri.com

Robert N. M. Watson[†]
robert.watson@cl.cam.ac.uk

Alexander Richardson[†]
alexander.richardson@cl.cam.ac.uk

Peter G. Neumann[*]
peter.neumann@sri.com

Simon W. Moore[†]
simon.moore@cl.cam.ac.uk

John Baldwin[‡]
john@araratriver.co

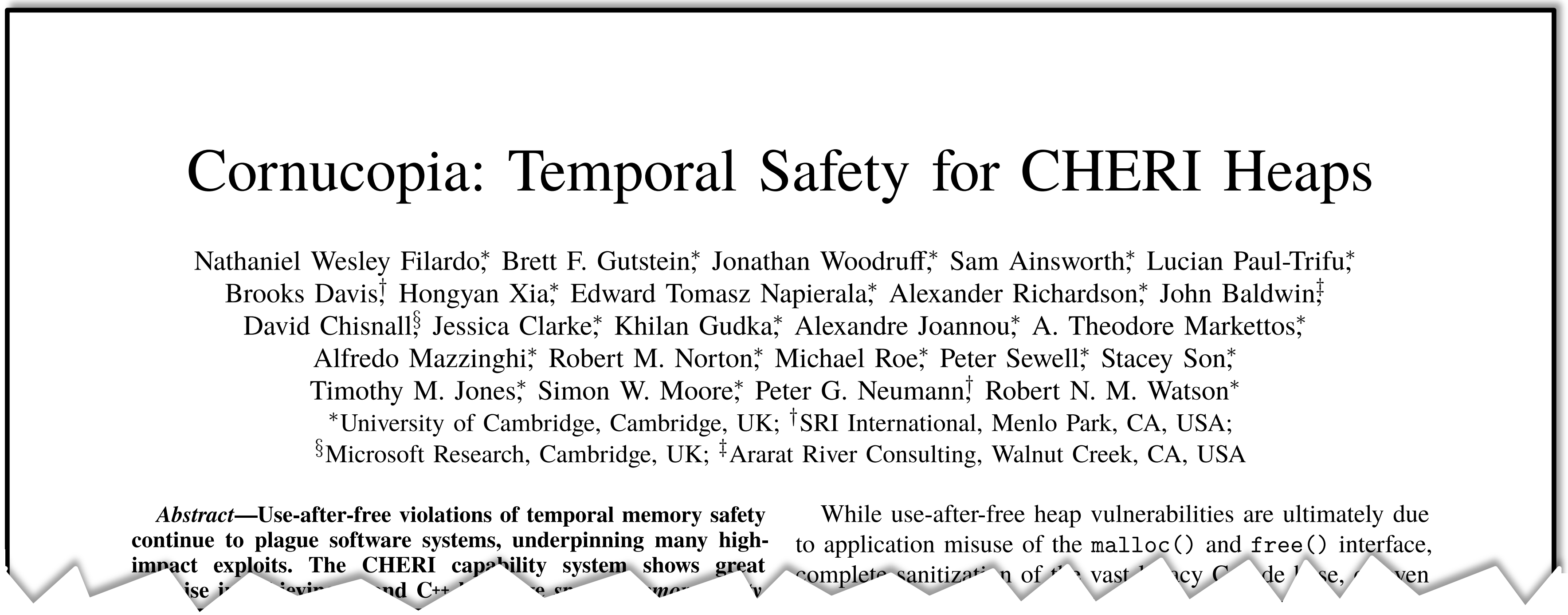
David Chisnall[§]

Jessica Clarke[†]

Nathaniel Wesley Filardo[†]

- Received best paper award at ASPLOS, April 2019
- Complete pure-capability UNIX OS userspace with spatial memory safety
  - Usable for daily development tasks
  - Almost vast majority of FreeBSD tests pass
  - Management interfaces (e.g. ioctl), debugging, etc., work
  - Large, real-world applications have been ported: PostgreSQL and WebKit

# Heap temporal memory safety



Cornucopia: Temporal Safety for CHERI Heaps

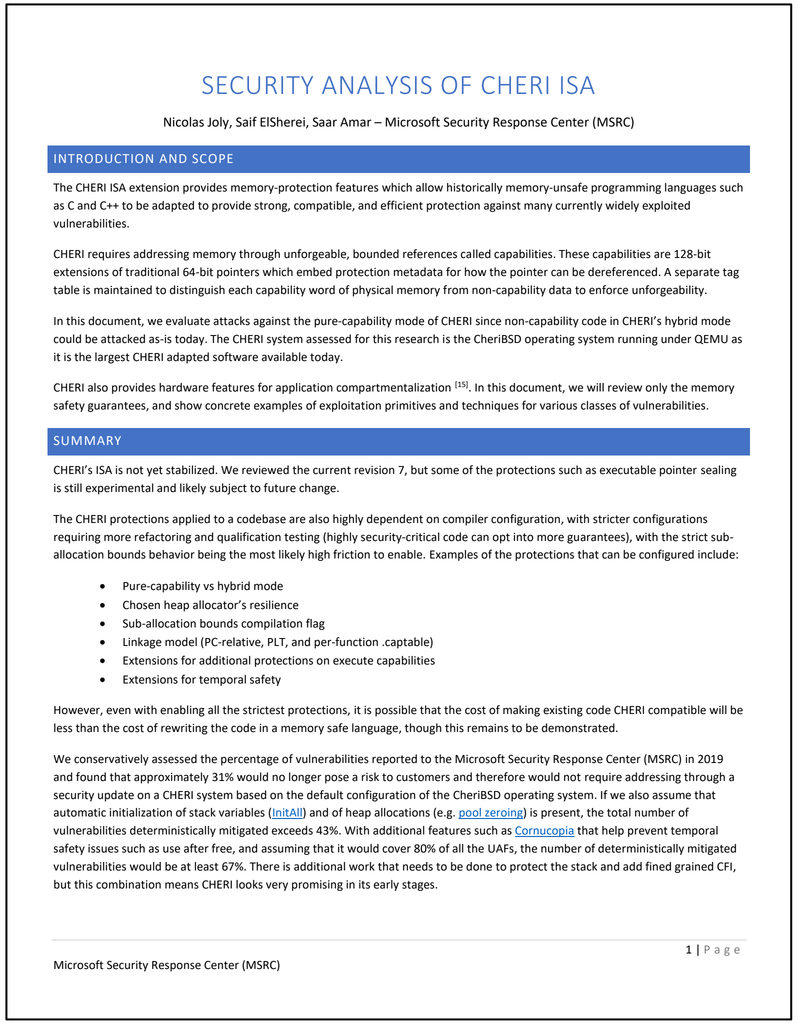
Nathaniel Wesley Filardo[*], Brett F. Gutstein[*], Jonathan Woodruff[*], Sam Ainsworth[*], Lucian Paul-Trifu[*], Brooks Davis[†], Hongyan Xia[*], Edward Tomasz Napierala[*], Alexander Richardson[*], John Baldwin[‡], David Chisnall[§], Jessica Clarke[*], Khilan Gudka[*], Alexandre Joannou[*], A. Theodore Markettos[*], Alfredo Mazzinghi[*], Robert M. Norton[*], Michael Roe[*], Peter Sewell[*], Stacey Son[*], Timothy M. Jones[*], Simon W. Moore[*], Peter G. Neumann[†], Robert N. M. Watson[*]
[*]University of Cambridge, Cambridge, UK; [†]SRI International, Menlo Park, CA, USA;
[§]Microsoft Research, Cambridge, UK; [‡]Ararat River Consulting, Walnut Creek, CA, USA

***Abstract*—Use-after-free violations of temporal memory safety continue to plague software systems, underpinning many high-impact exploits. The CHERI capability system shows great**

While use-after-free heap vulnerabilities are ultimately due to application misuse of the `malloc()` and `free()` interface, complete sanitization

- IEEE Symposium on Security and Privacy (“Oakland”), May 2020
- Hardware and software support for deterministic temporal memory safety for C/C++-language heaps using capability revocation
- Hardware enables fast tag searching using MMU-assisted tracking of tagged values, tag controller and cache

# MSRC: Security analysis of CHERI C/C++



## SECURITY ANALYSIS OF CHERI ISA

Nicolas Joly, Saif ElSherei, Saar Amar – Microsoft Security Response Center (MSRC)

### INTRODUCTION AND SCOPE

The CHERI ISA extension provides memory-protection features which allow historically memory-unsafe programming languages such as C and C++ to be adapted to provide strong, compatible, and efficient protection against many currently widely exploited vulnerabilities.

CHERI requires addressing memory through unforgeable, bounded references called capabilities. These capabilities are 128-bit extensions of traditional 64-bit pointers which embed protection metadata for how the pointer can be dereferenced. A separate tag table is maintained to distinguish each capability word of physical memory from non-capability data to enforce unforgeability.

In this document, we evaluate attacks against the pure-capability mode of CHERI since non-capability code in CHERI's hybrid mode could be attacked as-is today. The CHERI system assessed for this research is the CheriBSD operating system running under QEMU as it is the largest CHERI adapted software available today.

CHERI also provides hardware features for application compartmentalization [15]. In this document, we will review only the memory safety guarantees, and show concrete examples of exploitation primitives and techniques for various classes of vulnerabilities.

### SUMMARY

CHERI's ISA is not yet stabilized. We reviewed the current revision 7, but some of the protections such as executable pointer sealing is still experimental and likely subject to future change.

The CHERI protections applied to a codebase are also highly dependent on compiler configuration, with stricter configurations requiring more refactoring and qualification testing (highly security-critical code can opt into more guarantees), with the strict sub-allocation bounds behavior being the most likely high friction to enable. Examples of the protections that can be configured include:

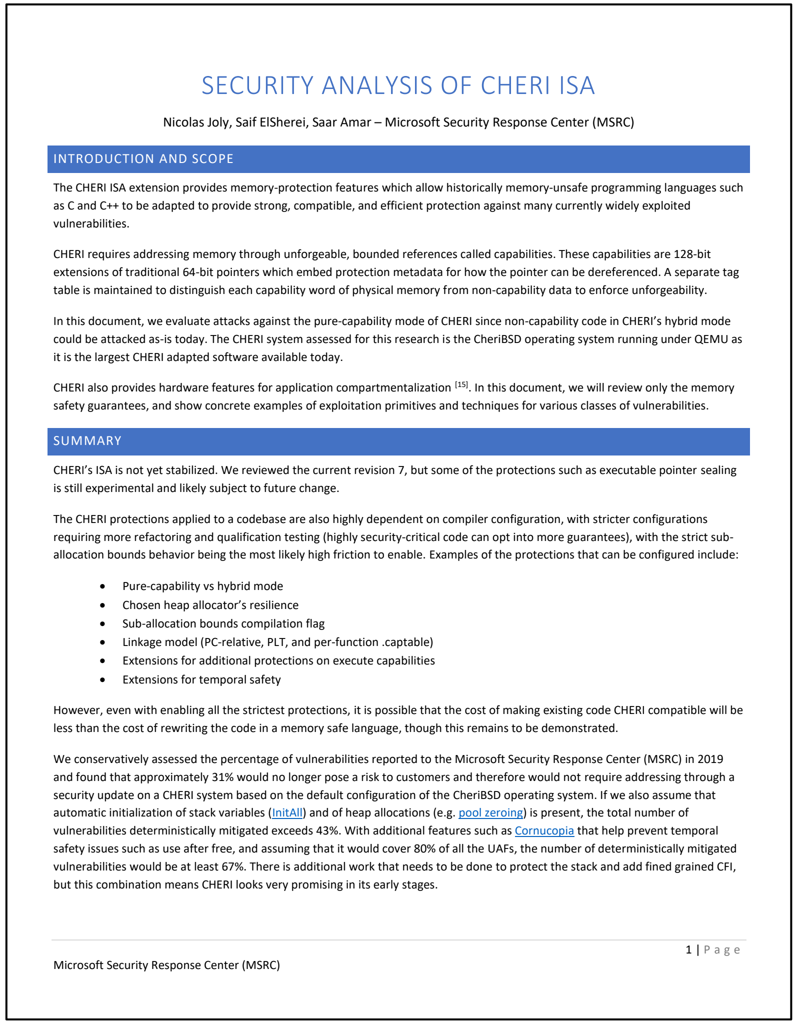
- Pure-capability vs hybrid mode
- Chosen heap allocator's resilience
- Sub-allocation bounds compilation flag
- Linkage model (PC-relative, PLT, and per-function .captable)
- Extensions for additional protections on execute capabilities
- Extensions for temporal safety

However, even with enabling all the strictest protections, it is possible that the cost of making existing code CHERI compatible will be less than the cost of rewriting the code in a memory safe language, though this remains to be demonstrated.

We conservatively assessed the percentage of vulnerabilities reported to the Microsoft Security Response Center (MSRC) in 2019 and found that approximately 31% would no longer pose a risk to customers and therefore would not require addressing through a security update on a CHERI system based on the default configuration of the CheriBSD operating system. If we also assume that automatic initialization of stack variables (InitAll) and of heap allocations (e.g. pool zeroing) is present, the total number of vulnerabilities deterministically mitigated exceeds 43%. With additional features such as Cornucopia that help prevent temporal safety issues such as use after free, and assuming that it would cover 80% of all the UAFs, the number of deterministically mitigated vulnerabilities would be at least 67%. There is additional work that needs to be done to protect the stack and add fined grained CFI, but this combination means CHERI looks very promising in its early stages.

- Study analyzed all 2019 critical security vulnerabilities
  - Metric: "Poses a risk to customers → requires a software update"
- Blog post and 42-page report
  - Concrete vulnerability analysis for spatial safety
  - Abstract analysis of the impact of temporal safety
  - Red teaming of specific artifacts to build CHERI experience
  - Potential adversarial techniques post-CHERI
  - Recently shifted from CHERI-MIPS to CHERI-RISC-V and Arm Morello

# Microsoft security analysis of CHERI C/C++



SECURITY ANALYSIS OF CHERI ISA

Nicolas Joly, Saif ElSherei, Saar Amar – Microsoft Security Response Center (MSRC)

INTRODUCTION AND SCOPE

The CHERI ISA extension provides memory-protection features which allow historically memory-unsafe programming languages such as C and C++ to be adapted to provide strong, compatible, and efficient protection against many currently widely exploited vulnerabilities.

CHERI requires addressing memory through unforgeable, bounded references called capabilities. These capabilities are 128-bit extensions of traditional 64-bit pointers which embed protection metadata for how the pointer can be dereferenced. A separate tag table is maintained to distinguish each capability word of physical memory from non-capability data to enforce unforgeability.

In this document, we evaluate attacks against the pure-capability mode of CHERI since non-capability code in CHERI's hybrid mode could be attacked as-is today. The CHERI system assessed for this research is the CheriBSD operating system running under QEMU as it is the largest CHERI adapted software available today.

CHERI also provides hardware features for application compartmentalization [15]. In this document, we will review only the memory safety guarantees, and show concrete examples of exploitation primitives and techniques for various classes of vulnerabilities.

SUMMARY

CHERI's ISA is not yet stabilized. We reviewed the current revision 7, but some of the protections such as executable pointer sealing is still experimental and likely subject to future change.

The CHERI protections applied to a codebase are also highly dependent on compiler configuration, with stricter configurations requiring more refactoring and qualification testing (highly security-critical code can opt into more guarantees), with the strict sub-allocation bounds behavior being the most likely high friction to enable. Examples of the protections that can be configured include:

- Pure-capability vs hybrid mode
- Chosen heap allocator's resilience
- Sub-allocation bounds compilation flag
- Linkage model (PC-relative, PLT, and per-function .captable)
- Extensions for additional protections on execute capabilities
- Extensions for temporal safety

However, even with enabling all the strictest protections, it is possible that the cost of making existing code CHERI compatible will be less than the cost of rewriting the code in a memory safe language, though this remains to be demonstrated.

We conservatively assessed the percentage of vulnerabilities reported to the Microsoft Security Response Center (MSRC) in 2019 and found that approximately 31% would no longer pose a risk to customers and therefore would not require addressing through a security update on a CHERI system based on the default configuration of the CheriBSD operating system. If we also assume that automatic initialization of stack variables (InitAll) and of heap allocations (e.g. pool zeroing) is present, the total number of vulnerabilities deterministically mitigated exceeds 43%. With additional features such as Cornucopia that help prevent temporal safety issues such as use after free, and assuming that it would cover 80% of all the UAFs, the number of deterministically mitigated vulnerabilities would be at least 67%. There is additional work that needs to be done to protect the stack and add fined grained CFI, but this combination means CHERI looks very promising in its early stages.

1 | Page

Microsoft Security Response Center (MSRC)

- Microsoft Security Research Center (MSRC) study analyzed all 2019 Microsoft critical memory-safety security vulnerabilities
  - Metric: "Poses a risk to customers → requires a software update"
  - Vulnerability mitigated if **no security update required**
- Blog post and 42-page report
  - Concrete vulnerability analysis for spatial safety
  - Abstract analysis of the impact of temporal safety
  - Red teaming of specific artifacts to gain experience
- CHERI, "in its current state, and combined with other mitigations, it would have **deterministically mitigated at least two thirds of all those issues**"

https://msrc-blog.microsoft.com/2020/10/14/security-analysis-of-cheri-isa/

## Security Analysis of CHERI ISA

Security Research & Defense / By MSRC Team / October 14, 2020 /
Memory Corruption, Memory Safety, Secure Development, Security Research

Is it possible to get to a state where memory safety issues would be deterministically mitigated? Our quest to mitigate memory corruption vulnerabilities led us to examine CHERI (Capability Hardware Enhanced RISC Instructions), which provides memory protection features against many exploited vulnerabilities, or in other words, an architectural solution that breaks exploits. We've looked at how CHERI would break class-specific categories of vulnerabilities and considered additional mitigations to put in place to get to a comprehensive solution. **We've assessed the theoretical impact of CHERI on all the memory safety vulnerabilities we received in 2019, and concluded that in its current state, and combined with other mitigations, it would have deterministically mitigated at least two thirds of all those issues**.

We've reviewed revision 7 and used CheriBSD running under QEMU as a test environment. In this research, we've also looked for weaknesses in the model and ended up developing exploits for various security issues using CheriBSD and qtwebkit. We've highlighted several areas that warrant improvements, such as vulnerability classes that CHERI doesn't mitigate at the architectural level, the importance of using reliable and CHERI compliant memory management mechanisms, and multiple exploitation primitives that would still allow memory corruption issues to be exploited. **While CHERI does a fantastic job at breaking spatial safety issues, more is needed to tackle temporal and type safety issues**.

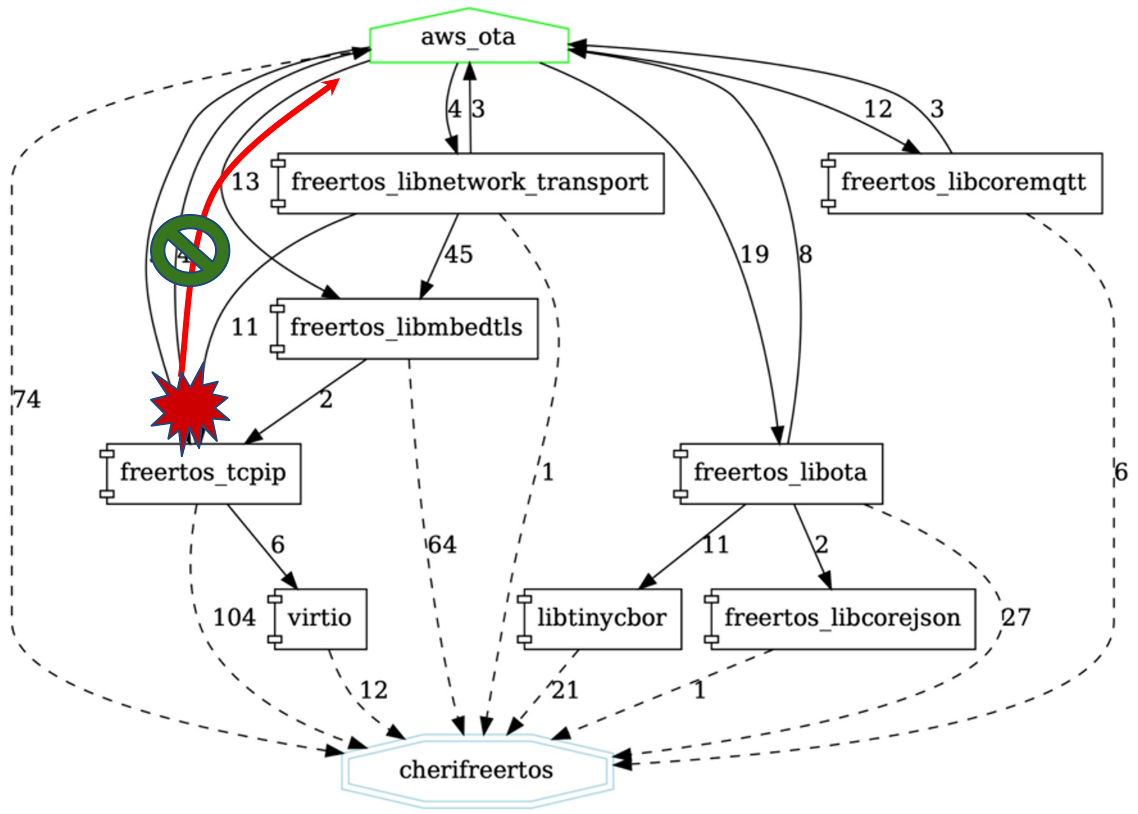
Your feedback is extremely important to us as there's certainly much more to discover and mitigate. We're looking forward to your comments on our paper.

Nicolas Joly, Saif ElSherei, Saar Amar – Microsoft Security Response Center (MSRC)

https://msrc-blog.microsoft.com/2020/10/14/security-analysis-of-cheri-isa/

## CHERI SOFTWARE COMPARTMENTALISATION

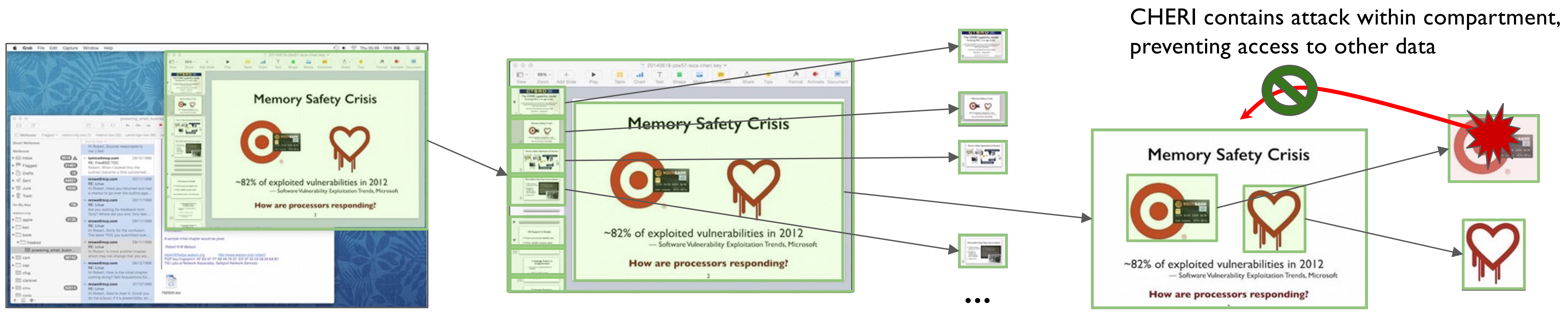
# What is software compartmentalization?



CheriFreeRTOS components and the application execute in compartments. CHERI contains an attack within TCP/IP compartment, which access neither flash nor the internals of the software update (OTA) compartment.

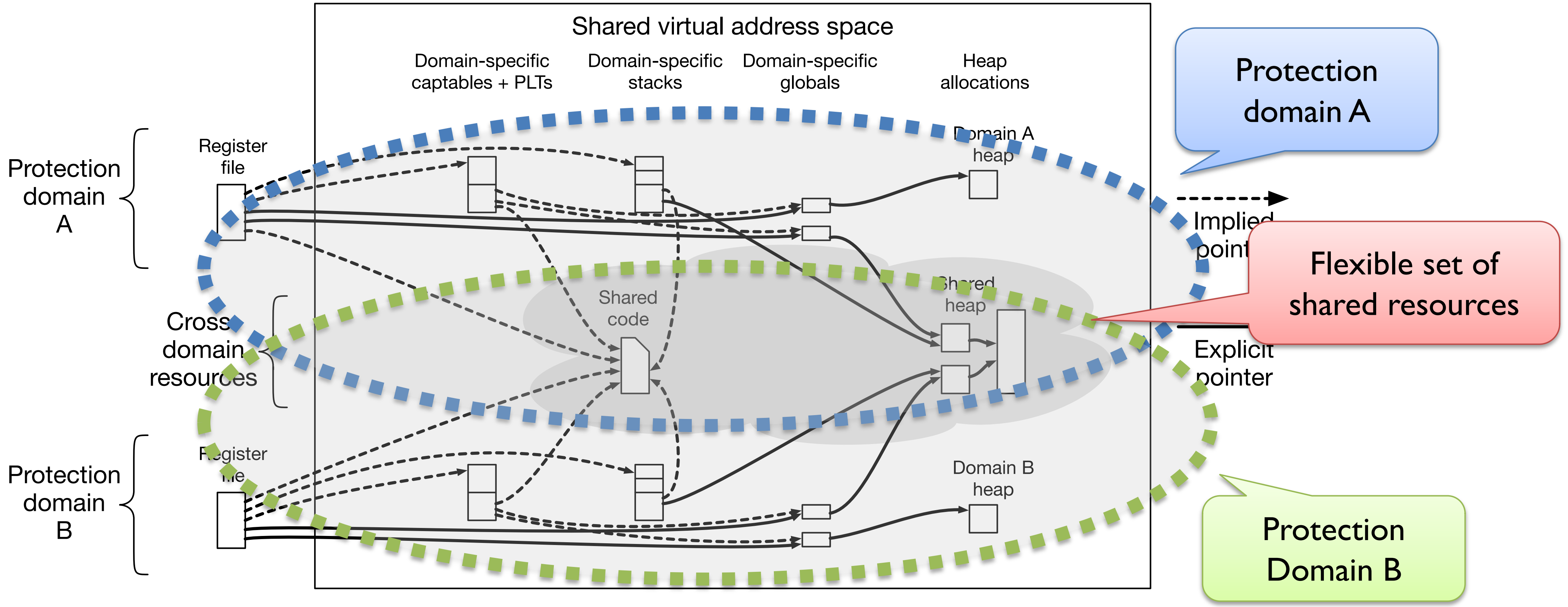
- Fine-grained decomposition of a larger software system into **isolated modules** to constrain the impact of faults or attacks
- Goals is to **minimize privileges yielded by a successful attack, and to limit further attack surfaces**
- Usefully thought about as a **graph of interconnected components**, where the attacker's goal is to compromise nodes of the graph providing a route from a point of entry to a specific target

# Software compartmentalization at scale


CHERI contains attack within compartment, preventing access to other data
Memory Safety Crisis
~82% of exploited vulnerabilities in 2012
— Software Vulnerability Exploitation Trends, Microsoft
How are processors responding?
…

- Current CPUs limit:
  - The number of compartments and rate of their creation/destruction
  - The frequency of switching between them, especially as compartment count grows
  - The nature and performance of memory sharing between compartments
- CHERI is intended to improve each of these – by at least an order of magnitude

# CHERI-based compartmentalization



- Isolated compartments can be created using closed graphs of capabilities, combined with a constrained non-monotonic domain-transition mechanism

# Compartmentalization scalability

- CHERI dramatically improves **compartmentalization scalability**
  - More compartments
  - More frequent and faster domain transitions
  - Faster shared memory between compartments

Early benchmarks show a 1-to-2 order of magnitude performance inter-compartment communication improvement compared to conventional designs

- Many potential use cases – e.g., sandbox processing of each image in a web browser, processing each message in a mail application
- Unlike memory protection, software compartmentalization requires **careful software refactoring** to support strong encapsulation, and affects the software operational model

# Operational models for CHERI compartmentalization

- An **architectural protection model** enabling new software behavior
- As with virtual memory, multiple **operational models** can be supported
  - E.g., with an MMU: Microkernels, processes, virtual machines, etc.
  - How are compartments created/destroyed? Function calls vs. message passing? Signaling, debugging, …?
- We have explored multiple viable CHERI-based models to date, including:

| | |
|---|---|
| **Isolated dynamic libraries** | Efficient but simple sandboxing in processes |
| **UNIX co-processes** | Multiple processes share an address space |

- Improved performance and new paradigms using CHERI primitives
- Both will be available in CheriBSD/Morello

# Proposed operational models: Isolated libraries and UNIX co-processes

**Isolated dynamically linked libraries**

- New API loads libraries into in-process sandboxes.
- Calling functions in isolated libraries performs a domain transition, with overheads comparable to function calls.
- Simple model eschews asynchrony, independent debugging, etc.

Prototype to appear in CheriBSD 22.10

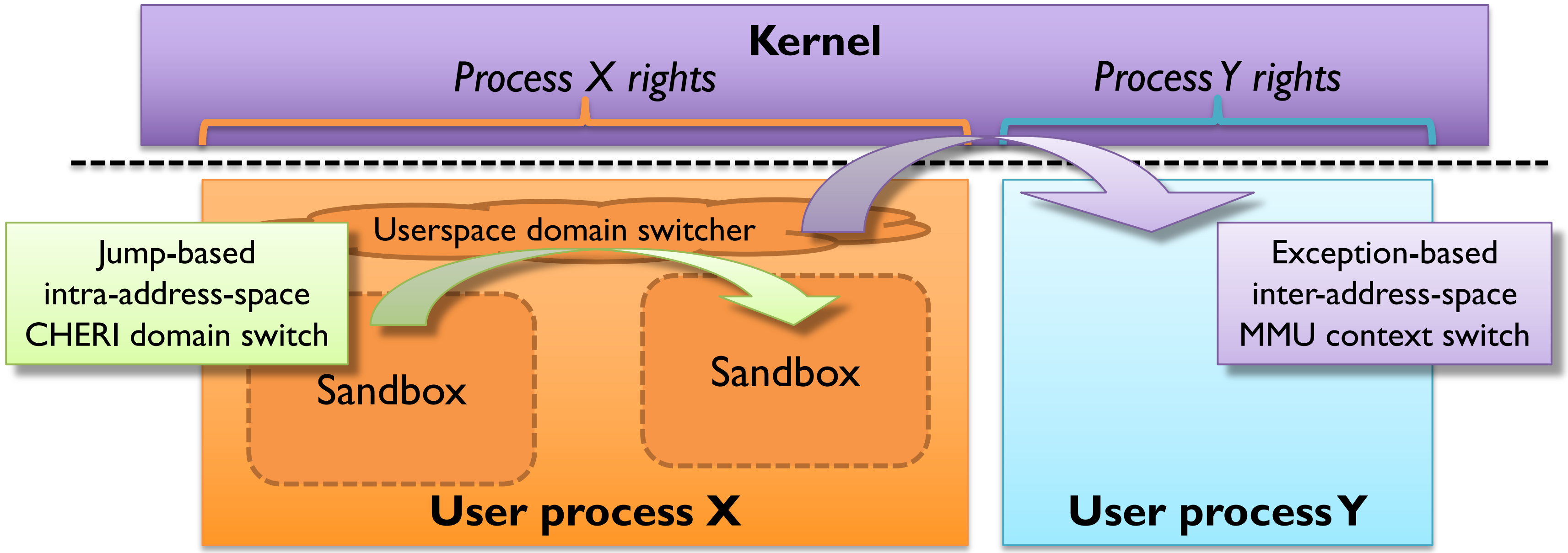
**UNIX co-processes**

- Multiple processes share a single virtual address space, separated using independent CHERI capability graphs.
- CHERI capabilities enable efficient sharing, domain transition.
- Rich model associates UNIX process with each compartment.

Prototype to appear in future CheriBSD release

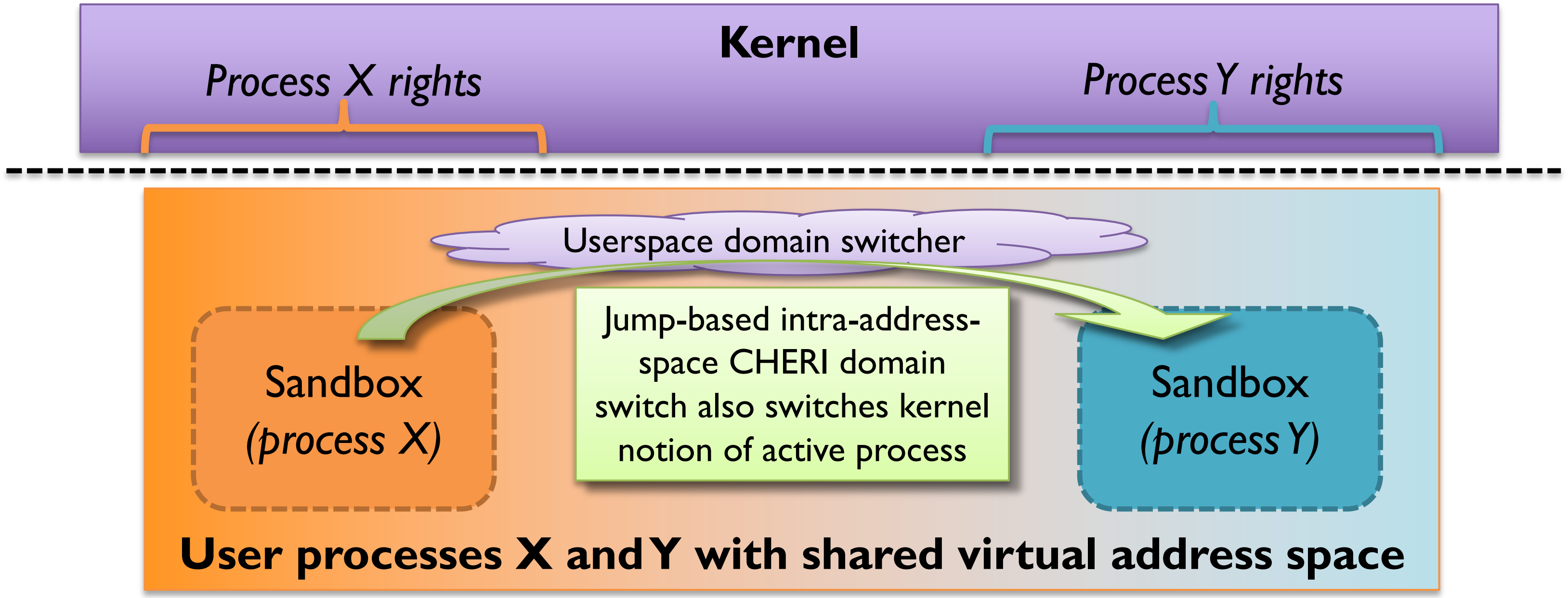
- **Active area of research; early prototype available for co-processes**

# Example: Robust shared libraries


Kernel
Process X rights
Process Y rights
Userspace domain switcher
Jump-based intra-address-space CHERI domain switch
Sandbox
Sandbox
User process X
Exception-based inter-address-space MMU context switch
User process Y

- User compartments exist **within individual UNIX processes** ("robust shared libraries"):
  - CHERI isolates compartments within each address spaces
  - Compartment switcher is itself a trusted userspace library
  - Compartments have strict subset of OS rights of the process
- Intra-process domain switches take **no architectural exceptions** and **do not enter the kernel**
- Multiple processes + IPC required if differing OS right sets needed

# Example: CHERI co-process model


Kernel
Process X rights
Process Y rights
Userspace domain switcher
Sandbox (process X)
Jump-based intra-address-space CHERI domain switch also switches kernel notion of active process
Sandbox (process Y)
User processes X and Y with shared virtual address space

- CHERI isolates **multiple processes** within a single virtual address space
  - Kernel-provided trusted compartment switcher runs in userspace (actually a microkernel)
  - CHERI-based inter-process memory sharing + domain switching
  - A compartment's OS rights correspond to the owning process
- Inter-process context switches take **no architectural exceptions** and **do not enter the kernel**
- CHERI can be pitched as **improving IPC performance** while **retaining a (largely) conventional process model**

## CHERI TRANSITION

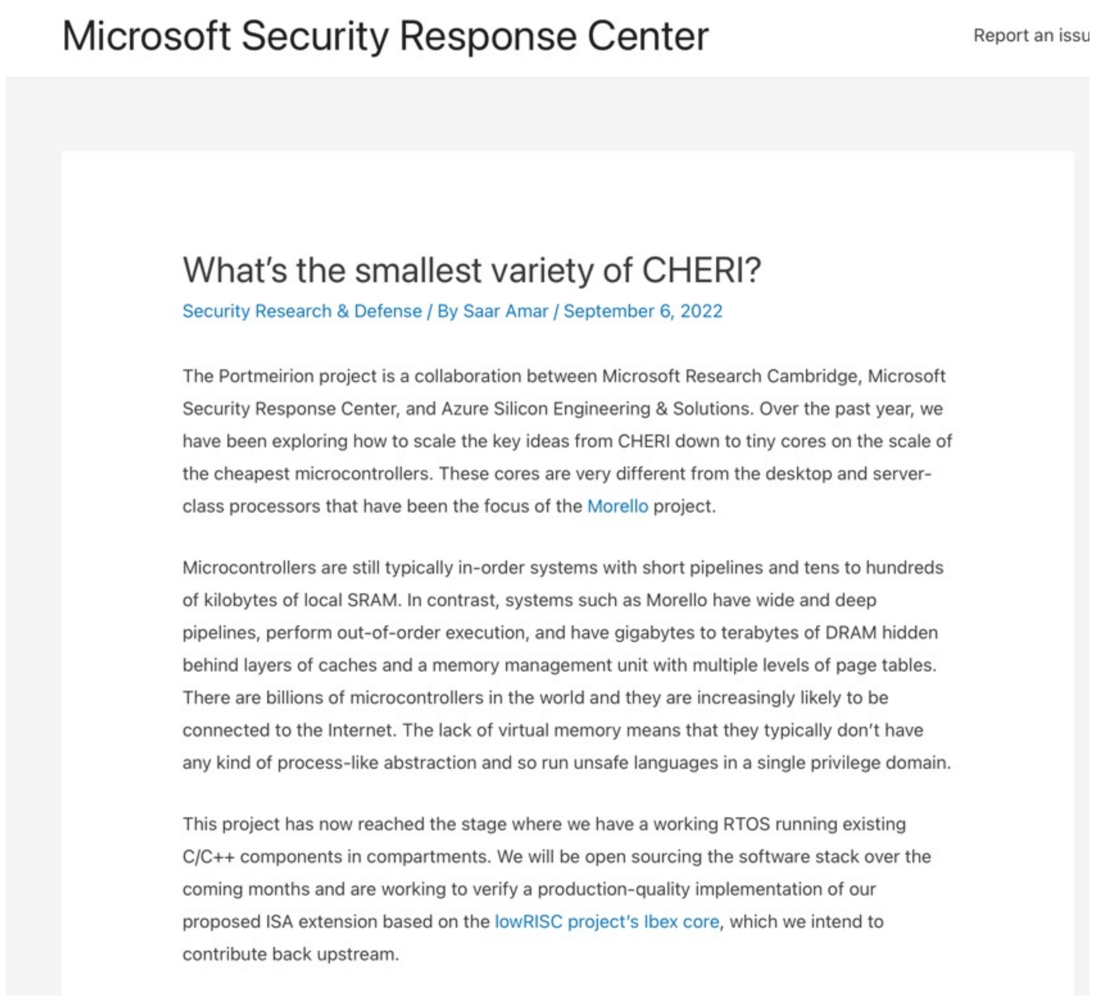
# Morello and CHERI-RISC-V

- We are pursing two CHERI adaptations to post-MIPS ISAs:
  - 2014 Joint with Arm, an experimental adaptation of 64-bit ARMv8-A **Arm Morello** multicore SoC, development board, etc. (**announced Oct. 2019; experimental SoC shipped 2022)**
  - 2017 An experimental adaptation of 32/64-bit RISC-V (**open-source research processors on FPGA**)
- Complete elaborations of the full hardware-software stack for each ISA:
  - All aspects of the architectures (e.g., ARMv8-A VM features, etc.)
  - Formal models + proofs, hardware implementations, compilers, OSes
- Potential for transition through both paths

# CHERI target architectures

| Architecture | Features | CHERI challenges |
|---|---|---|
| **64-bit MIPS** | 1990s RISC architecture (CHERI baseline) | Our legacy research architecture. Poor code density and addressing modes: harder to differentiate 'essential' CHERI costs; few transition opportunities with MIPS |
| **64-bit ARMv8-A** | Mature and widely deployed load-store architecture | Feature-rich; exception-adverse; rich address modes; constrained opcode space; hardware page tables; virtualization features; ecosystem |
| **32-bit and 64-bit RISC-V** | Open RISC ISA in active development (MIPS + 10 years?) | Limited addressing modes (expects micro-op fusion); hardware page tables; only partially standardized; features missing (e.g., hypervisor); immature software stack |

# What's the smallest variety of CHERI?



Microsoft Security Response Center

Report an issu

What's the smallest variety of CHERI?

Security Research & Defense / By Saar Amar / September 6, 2022

The Portmeirion project is a collaboration between Microsoft Research Cambridge, Microsoft Security Response Center, and Azure Silicon Engineering & Solutions. Over the past year, we have been exploring how to scale the key ideas from CHERI down to tiny cores on the scale of the cheapest microcontrollers. These cores are very different from the desktop and server-class processors that have been the focus of the Morello project.

Microcontrollers are still typically in-order systems with short pipelines and tens to hundreds of kilobytes of local SRAM. In contrast, systems such as Morello have wide and deep pipelines, perform out-of-order execution, and have gigabytes to terabytes of DRAM hidden behind layers of caches and a memory management unit with multiple levels of page tables. There are billions of microcontrollers in the world and they are increasingly likely to be connected to the Internet. The lack of virtual memory means that they typically don't have any kind of process-like abstraction and so run unsafe languages in a single privilege domain.

This project has now reached the stage where we have a working RTOS running existing C/C++ components in compartments. We will be open sourcing the software stack over the coming months and are working to verify a production-quality implementation of our proposed ISA extension based on the lowRISC project's Ibex core, which we intend to contribute back upstream.

- Production-quality CHERI-RISC-V-extended Ibex core
  - Small-scale microcontroller used in OpenTitan and other use cases
  - Clean-slate memory-safe, compartmentalized OS
  - Will be open-source hardware and software
  - CHERI-RISC-V tuned for small microcontrollers
  - RISC-V embedded standardization candidate
- Collaboration across Microsoft Research, MSRC, Azure Silicon, and Azure Edge + Platform

https://msrc-blog.microsoft.com/2022/09/06/whats-the-smallest-variety-of-cheri/

# RISC-V CHERI Special Interest Group (SIG)

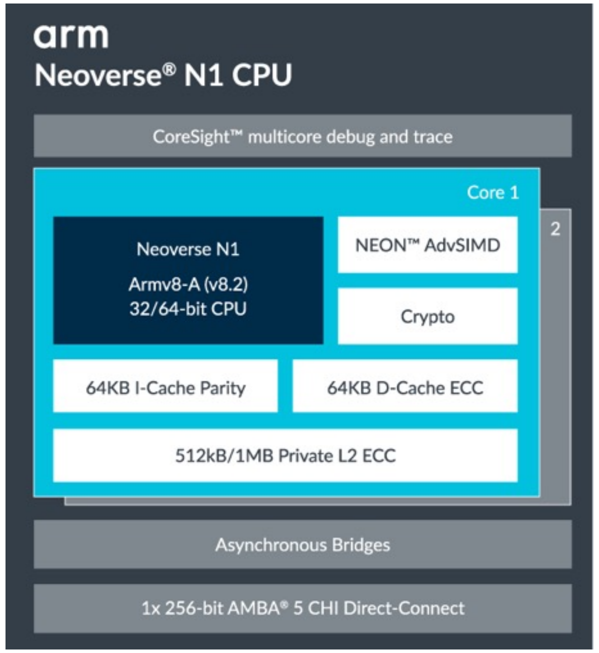
- Created in early October 2022
- SIG acting chair is Alex Richardson (Google)
- Intention to build interest and consensus around CHERI-RISC-V standardization
- Likely at least two closely coupled standardization efforts:
  - Microcontroller CHERI building on Microsoft's recent work
  - 64-bit CHERI-RISC-V building on SRI/Cambridge's ISA
- Lots of open questions -- e.g., do we need multiple working groups, how to we best capture the commonalities of the two ISA encodings, etc.
- SRI and Cambridge have recently joined RISC-V International to factilitate this

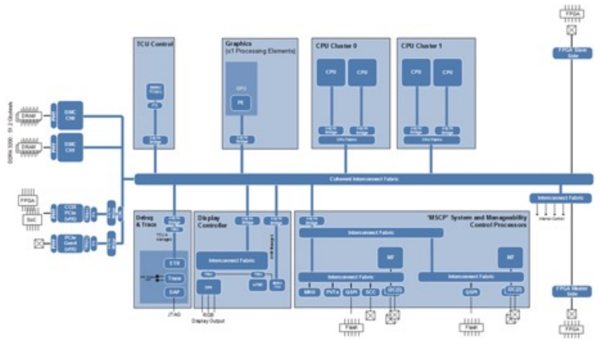
# CHERI-ARM research since 2014

- Since 2014, in collaboration with Arm, we have been pursuing joint research to experimentally incorporate CHERI into ARMv8-A:
  - Develop CHERI as an architecture-neutral and portable protection model implemented in multiple concrete architectures
  - Refine and extend the CHERI architecture – e.g., capability compression, tagging µarch, domain transition, and temporal safety
  - Apply concept of architecture neutrality to the CHERI-enabled software stack, including compiler, OS, and applications
  - Expand software: large-scale application experiments, OS use, debuggers, …
  - Extend work in formal modeling and proofs to an industrial-scale architecture
- Solve arising practical {hardware, software, …} problems as part of the research
- Build evidence, demonstrations, SW templates to support potential CHERI adoption

# ISCF: Digital Security by Design (UKRI)

- 5-year **Digital Security by Design** UKRI program: £70M UK gov. funding, £117M UK industrial match, to create CHERI-ARM demonstrator SoC + board with proven ISA
- Leap supply-chain gap that makes adopting new architecture difficult – in particular, validation of concepts in microarchitecture, architecture, and software "at scale"
- Support industrial and academic R&D (EPSRC, ESRC, InnovateUK)
- Baseline CPU is Neoverse N1; reuses existing SoC/board designs
- Collaborative review distillation of CHERI ISAv8; experimental additions relating to temporal safety, compartmentalization
- Science designed allowed: Multiple architectural + microarchitectural design choices for software-based evaluation
- 2020 emulation models; **2022 Morello board shipped!**





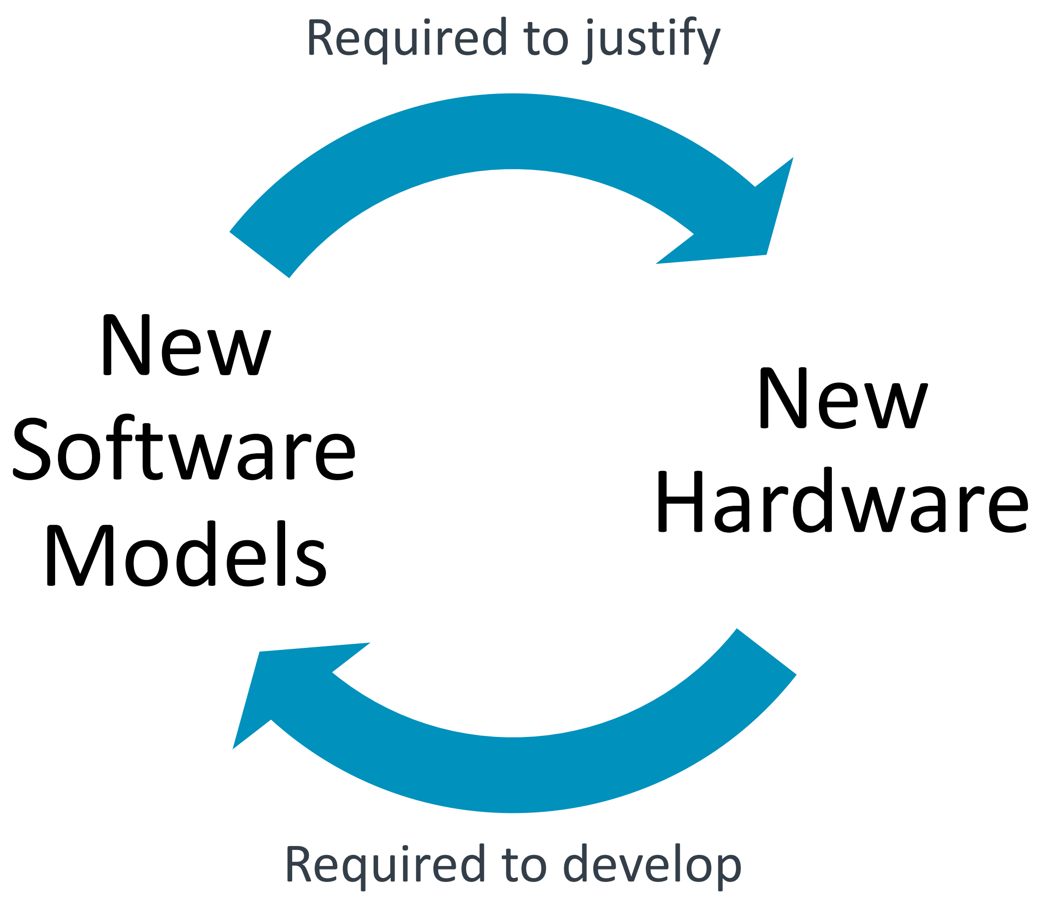
# Digital Security by Design

Richard Grisenthwaite

SVP Chief Architect and Fellow

Richard.Grisenthwaite@arm.com

## Challenges with creating substantially new architecture



## Why is Arm interested in the CHERI architecture

- Arm had been working with UoCambridge on CHERI for some 4-5 years
- Big step to addressing security based on strong fundamental principles
- Addresses spatial memory safety robustly and some ideas for temporal safety
  - Memory safety issues reported to be involved with ~70% of vulnerabilities (Matt Miller, BlueHat IL, 2019)
- Has scope to be the foundation of a new mechanism for compartmentalisation
  - Potentially far cheaper than using translation tables
- Interesting scope to address temporal safety issues as well as spatial ones….
- Many of the Arm software vendors are similarly interested in the possibilities of CHERI
  - Microsoft, Google and others have expressed strong interest in exploring the concept…
  - … but lots of questions about the real-world performance costs and usage models
  - …understanding the intended usage models is important to refine the architectural features
- But is a novel thing to do with additional costs to the system and software
  - Adding a 129th tag bit has a lot of impacts to the memory system
  - it is an ABI change, so non-trivial costs for compatibility for some uses

## IP Position

- Today's CPU architectures have largely the same basic functionality
  - "Similar but different" approaches to most aspects of system architecture
  - Small scale optimisations exist
- This position very beneficial for the porting of system software
  - Anything that fundamentally changes the system software architecture is likely to be ignored
- Arm believes that this reality needs to continue with capabilities
  - Implication is that we'd like the world's leading architectures to adopt capabilities
  - The Digital Security by Design program

## Arm Morello specification

Arm® Architecture Reference Manual Supplement Morello for A-profile Architecture

| | |
|---|---|
| Document number | DDI0606 |
| Document version | A.f |
| Document confidentiality | Non-confidential |

*Copyright © 2020 Arm Limited or its affiliates. All rights reserved.*

**Important message**
Morello is a prototype architecture, which has a particular meaning to Arm of which the recipient must be aware as follows:
Subject to change without consent of all parties, and it is not committed for product development.
Includes the majority of expected features.
Includes detail on the majority of expected features.
Includes some necessary information from documentation relating to earlier architectures, but some cross-referencing might be necessary.
See the architecture release notes for more detail.
No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

- Experimental application of CHERI ISAv8 to ARMv8-A
- Much richer base ISA .. Much longer spec - 2,155 pages excluding additional material!
- Describes ISA as implemented in Arm Morello FVP and processor/SoC
- Includes recent features such as sentry and load-side barrier support

# The Morello Board

- An Industrial Demonstrator of a Capability architecture
- Uses a prototype capability extension to the Arm Architecture
  - Prototype is a “superset” of what could be adopted into the Arm architecture
- Use of a superset of the architecture is very unusual
  - Also unrealistic as a commercial product – there will be some frequency effects
  - However, there are tight timescales so architecture is nearly complete now
- The superset of the architecture will allow a lot of software experimentation
  - Various different mechanisms for compartmentalisation
  - Collection of features for which the justification is unclear
  - Techniques for holding the capability tag bit
- Architecture will have formally proved security properties (with UoC and UoE)
- **Morello Board will be the ONLY physical implementation of this prototype architecture**
  - **Learnings from these experiments will be adopted into a mainstream extension to the Arm architecture**
  - **NO COMMITMENT TO FULL BINARY COMPATIBILITY TO THE PROTOTYPE ARCHITECTURE**
    - **But successful concepts are expected to be carried forward into the architecture and can be reused there**

INDUSTRIAL STRATEGY
UK Research and Innovation

arm

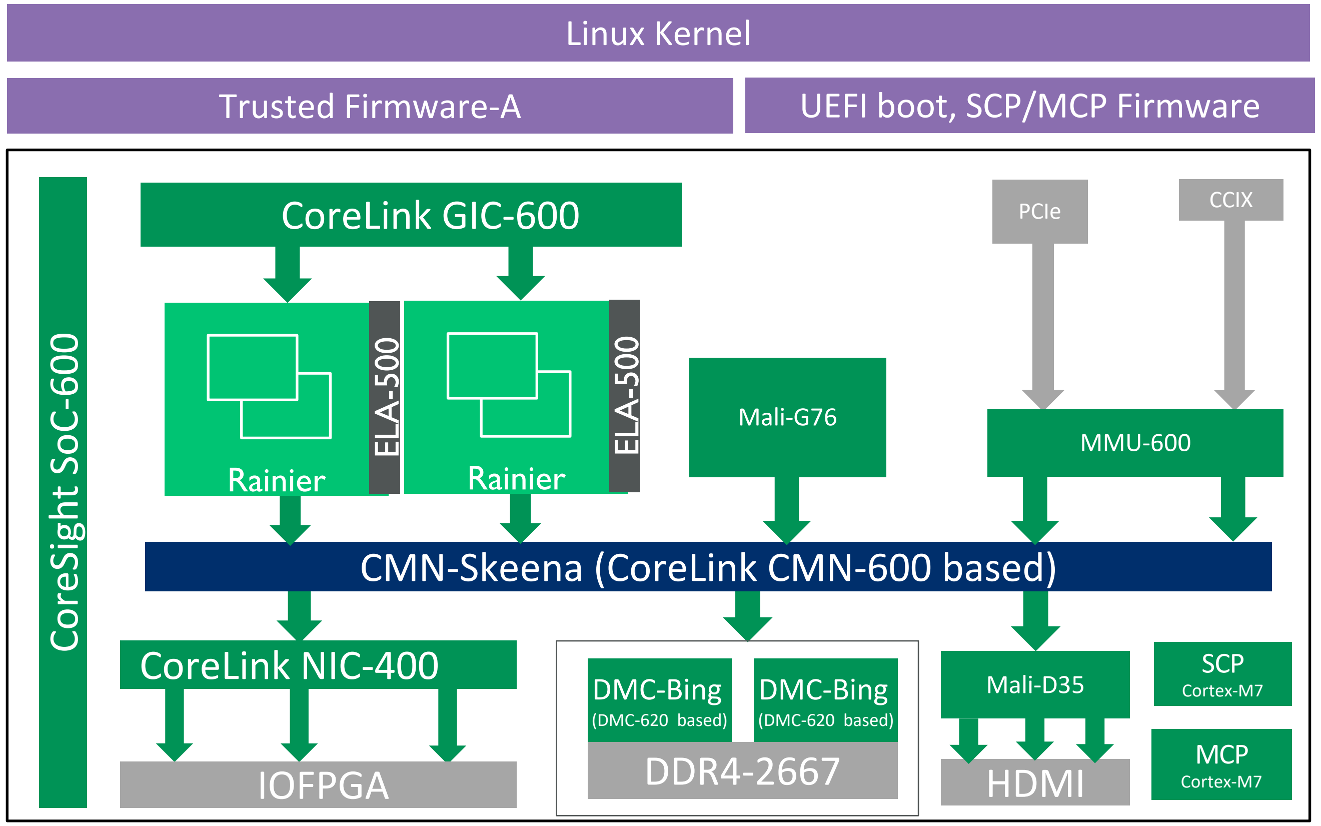
## Morello Board overview (subject to change)

- Quad core bespoke high-end CPU with prototype capability extensions
  - Backwards compatibility with v8.2 AArch64-only
  - Based on Neoverse N1 core
    - Multi-issue out-of-order superscalar core with 3 levels of cache
  - Build in 7nm process
  - Targeting clock frequency around 2GHz
- Reasonable performance GPU and Display controller
  - Standard Mali architecture core – not extended with capability
  - Supports Android
- PCIe and CCIx interfaces including to FPGA based accelerators
- FPGA for peripheral expansion
- SBSA compliant system
- 16GB of System Memory (expandable to 32GB – tbc)

## Morello Board: Capability Hardware Prototype Platform

INDUSTRIAL STRATEGY
UK Research and Innovation

- Silicon implementation of a Capability Hardware CPU Instruction Set Architecture
  - Implements Morello Profile for A-class Prototype Architecture
  - Two clusters each of two Rainier CPUs
  - Interconnect and Memory Controller support for tagged memory
  - Two channel DDR4 DRAM interface
  - PCIe Gen3 and Gen4 x16 interface
  - CCIX (Cache Coherent Interconnect for Accelerators) interface
  - Mid-range GPU, display processor and HDMI output
  - On standard uATX form factor board



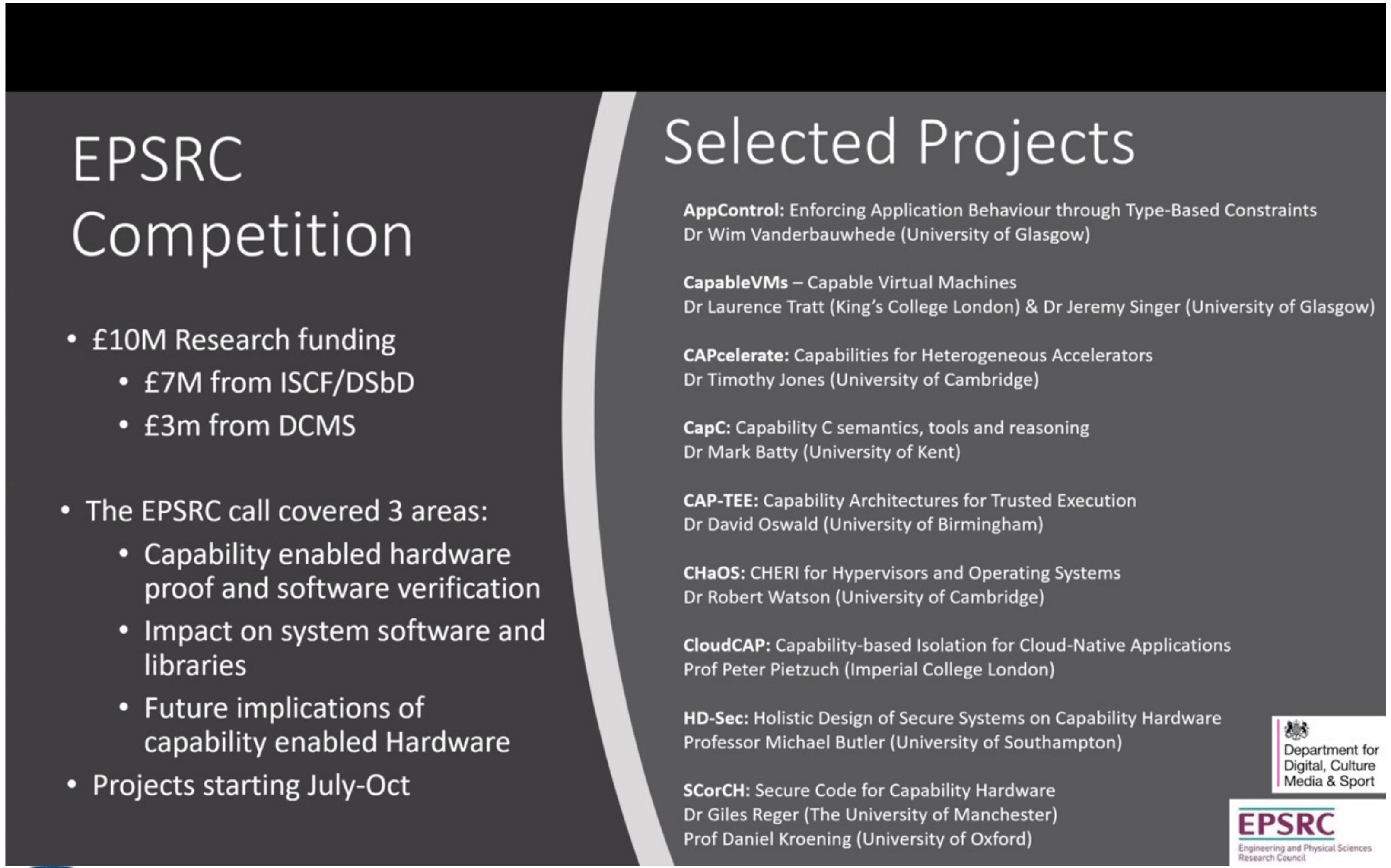
🟩 Supporting Arm system IP: GIC-600 (Generic Interrupt Controller), MMU-600 (IO MMU), Dynamic Memory Controller derived from DMC-620, SoC-600 (SoC Debug and Trace), Coherent Mesh Network derived from CMN-600, NIC-400 (Non-coherent interconnect)

⬜ Supporting 3rd party system IP/hardware: PCIe/CCIX Root Complex (PHY and controller), DDR4/3 PHY, DDR4 memory, IO FPGA

🟪 Open-source software stack

arm

## UK EPSRC DSbD research program 2020-2023



EPSRC Competition

- £10M Research funding
  - £7M from ISCF/DSbD
  - £3m from DCMS
- The EPSRC call covered 3 areas:
  - Capability enabled hardware proof and software verification
  - Impact on system software and libraries
  - Future implications of capability enabled Hardware
- Projects starting July-Oct

Selected Projects

**AppControl:** Enforcing Application Behaviour through Type-Based Constraints
Dr Wim Vanderbauwhede (University of Glasgow)

**CapableVMs** – Capable Virtual Machines
Dr Laurence Tratt (King's College London) & Dr Jeremy Singer (University of Glasgow)

**CAPcelerate:** Capabilities for Heterogeneous Accelerators
Dr Timothy Jones (University of Cambridge)

**CapC:** Capability C semantics, tools and reasoning
Dr Mark Batty (University of Kent)

**CAP-TEE:** Capability Architectures for Trusted Execution
Dr David Oswald (University of Birmingham)

**CHaOS:** CHERI for Hypervisors and Operating Systems
Dr Robert Watson (University of Cambridge)

**CloudCAP:** Capability-based Isolation for Cloud-Native Applications
Prof Peter Pietzuch (Imperial College London)

**HD-Sec:** Holistic Design of Secure Systems on Capability Hardware
Professor Michael Butler (University of Southampton)

**SCorCH:** Secure Code for Capability Hardware
Dr Giles Reger (The University of Manchester)
Prof Daniel Kroening (University of Oxford)

Department for Digital, Culture Media & Sport

EPSRC Engineering and Physical Sciences Research Council

- 9 EPSRC projects funded across 10 UK universities
- Several InnovateUK industrial projects supporting exploration, evaluation, demonstration

## CHERI REFERENCE SOFTWARE STACK

## Why port the CHERI stack to Morello?

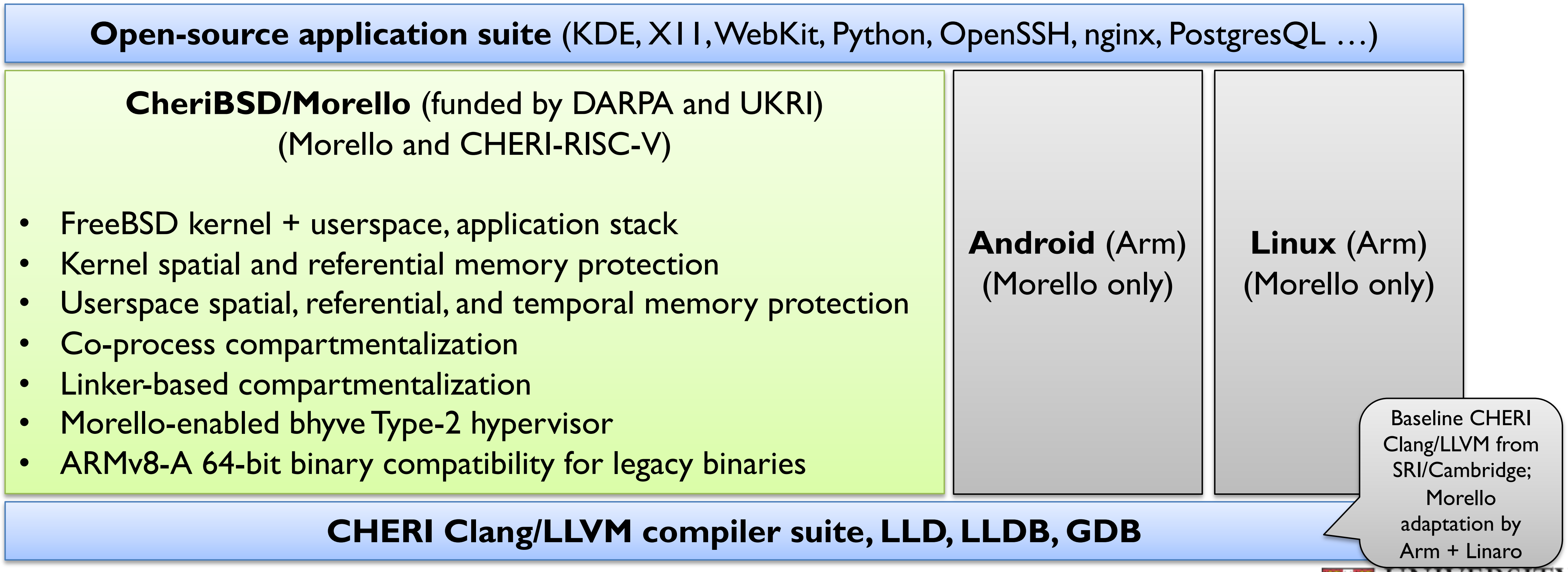
- **Validate** the Morello architecture (functional, sufficient)
- **Evaluate** the Morello implementation (performance, energy use, …)
- **Provide reference software semantics** (spatial and temporal safety, compartmentalization, POSIX integration, OS kernel use, …) that will be applicable to other adaptations
- **Act as a template and prototyping platform** for at-scale industrial and academic demonstration, including providing adaptations of common software dependencies (e.g., widely used libraries)
- **Provide a platform for future software research,** asking questions about what we can use CHERI for in {operating systems, compilers, language runtimes, applications, …}
- **Enable a growing academic and industrial community** around CHERI and Morello, including dozens of UK universities and companies associated with DSbD

## Caution: Research software!

- **The baseline compiler toolchain and OS stack are themselves research**
  - This means unknown risks, hard-to-predict schedules, and inevitable direction changes
- **Application Binary Interface (ABI) stability**
  - ABIs are a key research area; there are 2x Morello ABIs, and there will be [many?] more
  - This limits long-term binary compatibility guarantees for compiled software (for example)
- **Software performance optimization with a limited corpus**
  - Right now, we're just happy things are working, but we will get beyond that soon!
- **Supporting a large and diverse  audience of consumers with different objectives**
  - Engineering constraints limit objectives and support (e.g., software updates)
- **Software adaptation workload**
  - Some code ports trivially (e.g., Qt/KDE stack) and other code doesn't (e.g., JITs)

# CHERI prototype software stack on Morello

- **Complete open-source software stack** from bare metal up: compilers, toolchain, debuggers, hypervisor, OS, applications – all demonstrating CHERI
- Rich CHERI feature use, but fundamentally incremental/hybridized deployment



**Open-source application suite** (KDE, X11, WebKit, Python, OpenSSH, nginx, PostgresQL …)

**CheriBSD/Morello** (funded by DARPA and UKRI)
(Morello and CHERI-RISC-V)

- FreeBSD kernel + userspace, application stack
- Kernel spatial and referential memory protection
- Userspace spatial, referential, and temporal memory protection
- Co-process compartmentalization
- Linker-based compartmentalization
- Morello-enabled bhyve Type-2 hypervisor
- ARMv8-A 64-bit binary compatibility for legacy binaries

**Android** (Arm)
(Morello only)

**Linux** (Arm)
(Morello only)

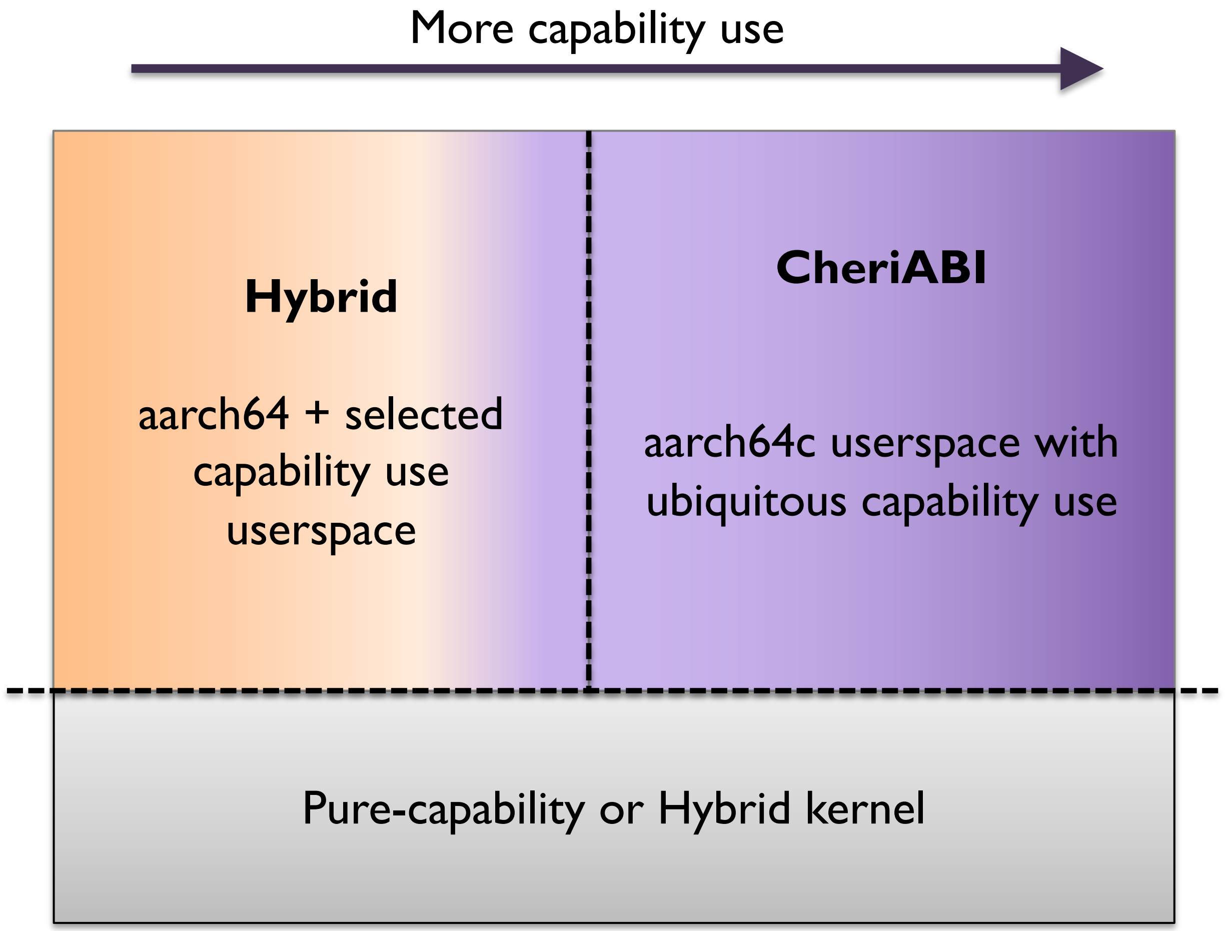
**CHERI Clang/LLVM compiler suite, LLD, LLDB, GDB**

Baseline CHERI Clang/LLVM from SRI/Cambridge; Morello adaptation by Arm + Linaro

# Some of our in-flight software R&D efforts

| Feature | Status | Availability |
|---|---|---|
| 3rd-party packages (Hybrid) | 23K software packages with strong functionality expectations | Since May 2022 (22.05 release) |
| 3rd-party packages (CheriABI) | 9K software packages with mixed functionality expectations | Since May 2022 (22.05 release) |
| **Morello GPU device drivers** | **Hybrid + pure-capability kernel driver**<br>**Hybrid + pure-capability user driver**<br>**Hybrid + pure-capability applications** | **Autumn 2022** |
| **Linker-based compartmentalization** | **Prototype runs some UNIX applications; limited debugger support** | **Autumn 2022 as (highly) experimental feature** |
| Userlevel heap temporal safety | Prototype runs SPEC benchmarks | Autumn 2022 (development branch), but "plug-in" to release |
| bhyve (Type-2) hypervisor | Prototype boots pure-capability guest OS, but much more testing + review required | Autumn (development branch) |
| Co-process compartmentalization | Prototype runs some compartmentalized software (e.g., OpenSSL); API co-design | Early 2023 (development branch) |

# (At least) two code generation / ABI targets



More capability use
Hybrid
aarch64 + selected capability use userspace
CheriABI
aarch64c userspace with ubiquitous capability use
Pure-capability or Hybrid kernel

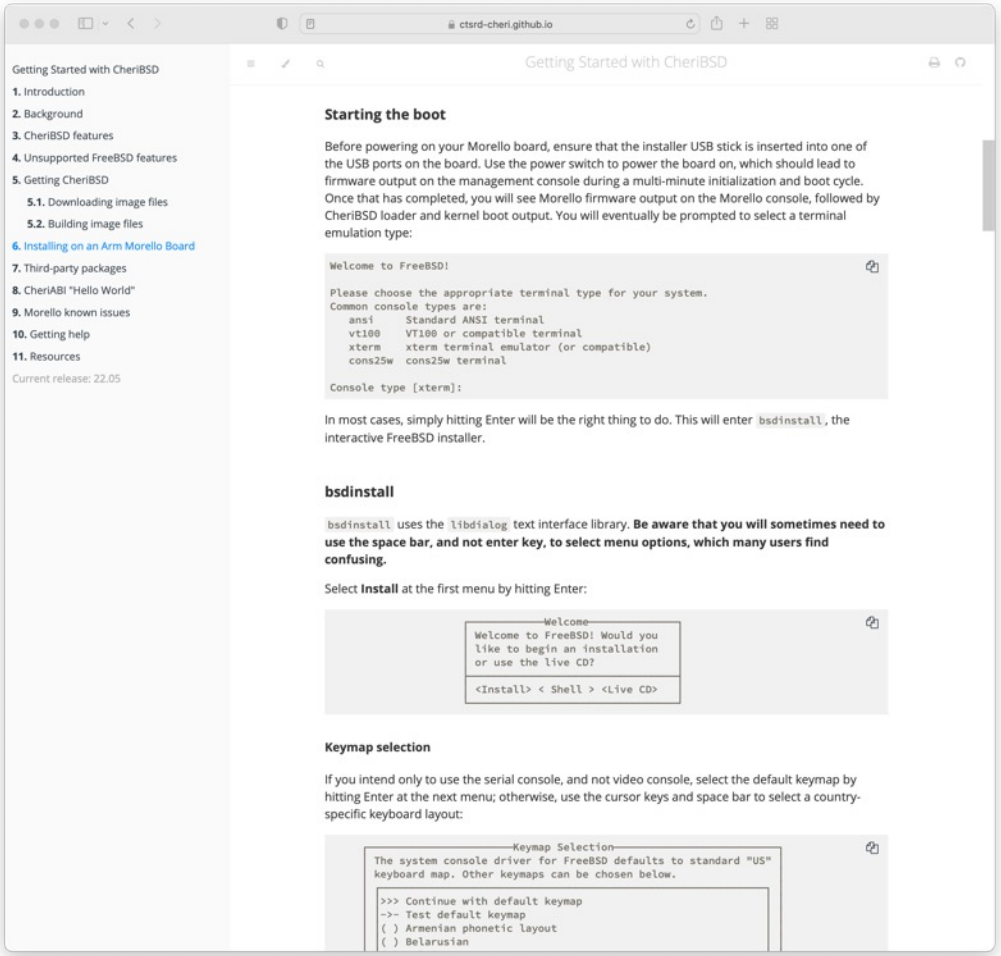
- **Hybrid code** is primarily aarch64 but with selected capability use:
  - Kernel: Mostly aarch64 with capability use for system-call arguments, context switching, virtual memory, signals
  - Userspace: Runs off-the-shelf arm64 programs without modification
- **Pure-capability code** implements all data and control-flow pointers with capabilities:
  - Kernel and userspace both spatially and referentially space
  - In the future userspace temporally safe

# FreeBSD base, ports/packages

| | | |
|---|---|---|
| **Base** | Base FreeBSD OS including kernel and key libraries, shells, daemons, and command-line tools | Well adapted to CHERI |
| **Ports** | Build infrastructure + FreeBSD adaptation patches – roughly 30,000 mainstream open-source libraries, runtimes, and application | Early prototype |
| **Packages** | Prebuilt binary packages built from ports, installed and managed using the pkg(8) package manager | Early prototype |

We provide a full set of ~20K-30K aarch64 (non-CHERI) packages to run on CheriBSD/Morello to use while the CheriABI collection matures.

# Getting Started with CheriBSD



ctsrd-cheri.github.io

Getting Started with CheriBSD

Current release: 22.05

**Starting the boot**

Before powering on your Morello board, ensure that the installer USB stick is inserted into one of the USB ports on the board. Use the power switch to power the board on, which should lead to firmware output on the management console during a multi-minute initialization and boot cycle. Once that has completed, you will see Morello firmware output on the Morello console, followed by CheriBSD loader and kernel boot output. You will eventually be prompted to select a terminal emulation type:

```
Welcome to FreeBSD!

Please choose the appropriate terminal type for your system.
Common console types are:
   ansi     Standard ANSI terminal
   vt100    VT100 or compatible terminal
   xterm    xterm terminal emulator (or compatible)
   cons25w  cons25w terminal

Console type [xterm]:
```

In most cases, simply hitting Enter will be the right thing to do. This will enter `bsdinstall`, the interactive FreeBSD installer.

**bsdinstall**

`bsdinstall` uses the `libdialog` text interface library. **Be aware that you will sometimes need to use the space bar, and not enter key, to select menu options, which many users find confusing.**

Select **Install** at the first menu by hitting Enter:

```
            ┌─────────Welcome─────────┐
            │Welcome to FreeBSD! Would you│
            │like to begin an installation│
            │or use the live CD?          │
            ├─────────────────────────────┤
            │<Install> < Shell > <Live CD>│
            └─────────────────────────────┘
```

**Keymap selection**

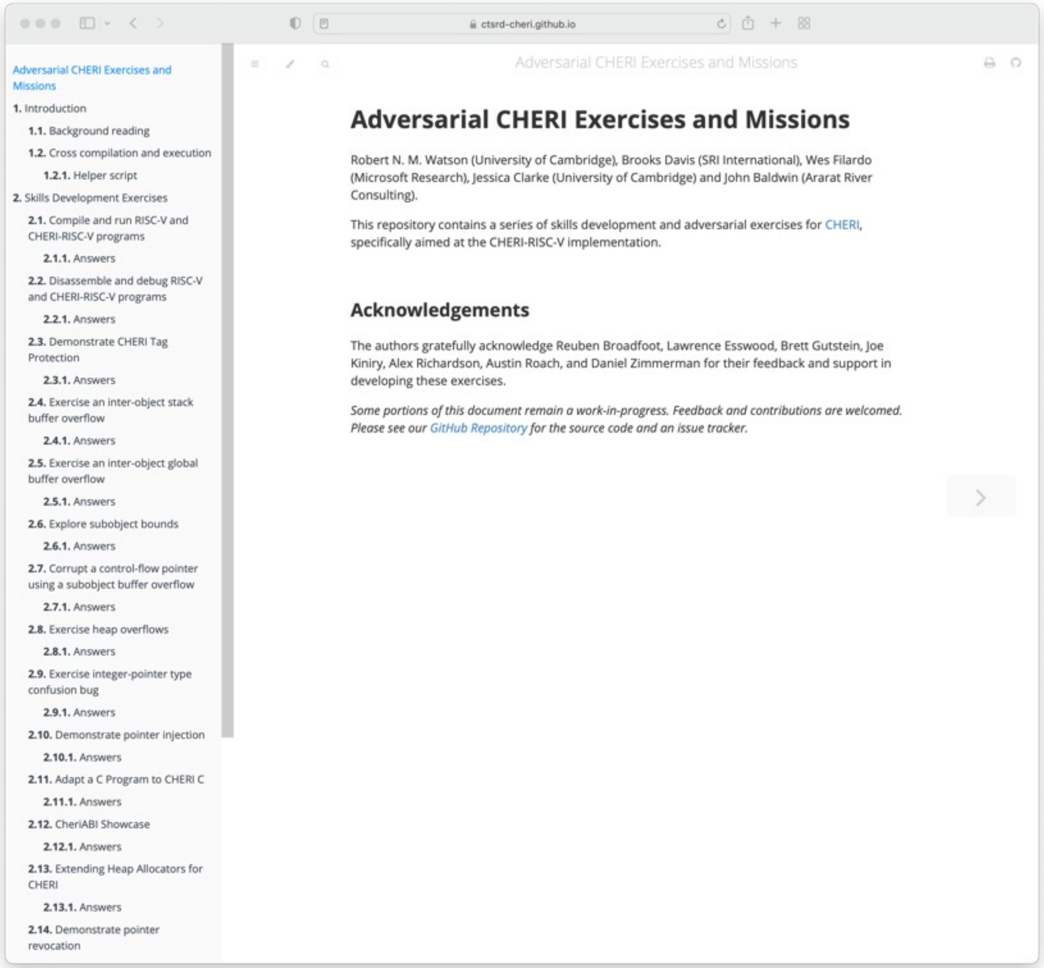
If you intend only to use the serial console, and not video console, select the default keymap by hitting Enter at the next menu; otherwise, use the cursor keys and space bar to select a country-specific keyboard layout:

```
┌──────────────────Keymap Selection──────────────────┐
│The system console driver for FreeBSD defaults to standard "US"│
│keyboard map. Other keymaps can be chosen below.    │
│┌──────────────────────────────────────────────┐    │
││>>> Continue with default keymap               │    │
││->- Test default keymap                        │    │
││( ) Armenian phonetic layout                   │    │
││( ) Belarusian                                 │    │
```

- Introduces CheriBSD
- Steps you through installation on a Morello board using a USB stick image that you can download
- Describes third-party package system and pkg64/pkg64c
- Illustrates "hello world" compilation and debugging
- Describes some known issues
- Explains how to get support

https://ctsrd-cheri.github.io/cheribsd-getting-started/

# Adversarial CHERI Exercises and Missions



ctsrd-cheri.github.io

Adversarial CHERI Exercises and Missions

**Adversarial CHERI Exercises and Missions**

Robert N. M. Watson (University of Cambridge), Brooks Davis (SRI International), Wes Filardo (Microsoft Research), Jessica Clarke (University of Cambridge) and John Baldwin (Ararat River Consulting).

This repository contains a series of skills development and adversarial exercises for CHERI, specifically aimed at the CHERI-RISC-V implementation.

**Acknowledgements**

The authors gratefully acknowledge Reuben Broadfoot, Lawrence Esswood, Brett Gutstein, Joe Kiniry, Alex Richardson, Austin Roach, and Daniel Zimmerman for their feedback and support in developing these exercises.

*Some portions of this document remain a work-in-progress. Feedback and contributions are welcomed. Please see our GitHub Repository for the source code and an issue tracker.*

- CHERI training exercises for developers, red teams, and bug bounties
- Adversarial missions where we want to understand exploitation better
- CHERI software adaptation
- Assume a strong level of knowledge about C, code generation, exploitation
  - (E.g., GOTs, PLTs, ROP, and JOP)
- Targets Morello and CHERI-RISC-V

https://ctsrd-cheri.github.io/cheri-exercises/

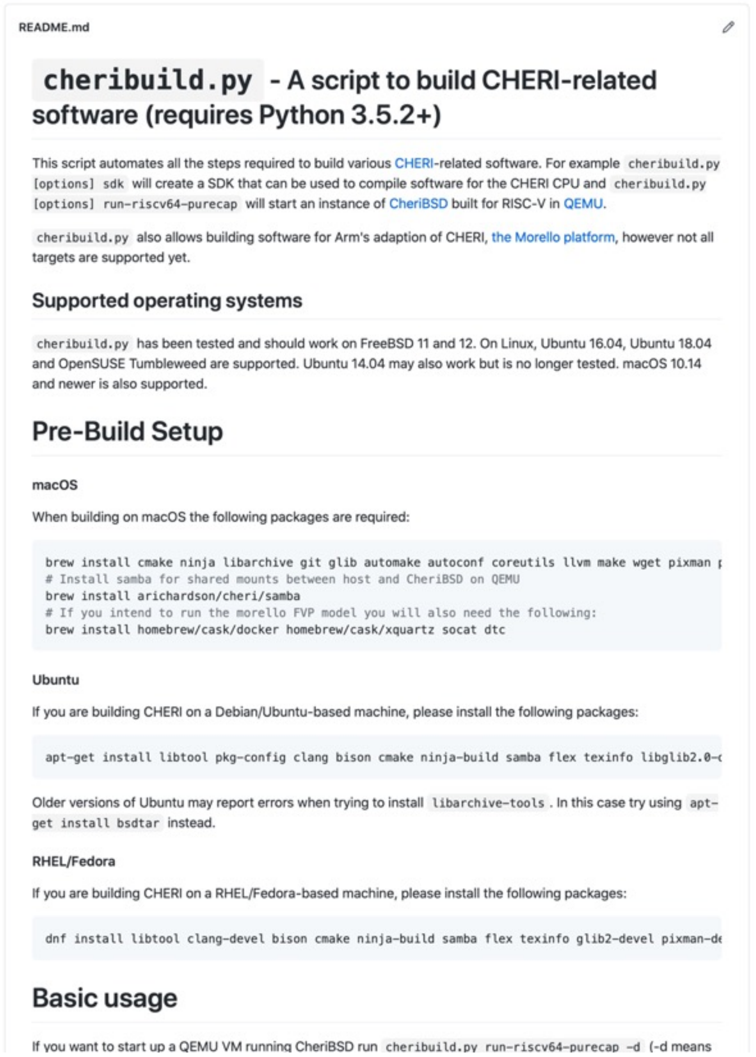
# CHERI software stack support channels

- cheri-cpu.slack.com Slack
  - Visit the CHERI website to request an invitation email/link
- Forthcoming mailing lists (not yet live)

| | |
|---|---|
| • cl-cheribsd-announce | Low-traffic announcement |
| • cl-cheribsd-discuss | General discussion and support |
| • cl-cheribsd-security | Report security issues |

- Sundry issue trackers in the github.com/CTSRD-CHERI organization
- Not just "How do I get the software to work", but also to assist with **experimental design**, **interpreting results**, and **seeking improvements**

# How to obtain and install the CHERI software stack



README.md

## `cheribuild.py` - A script to build CHERI-related software (requires Python 3.5.2+)

This script automates all the steps required to build various CHERI-related software. For example `cheribuild.py [options] sdk` will create a SDK that can be used to compile software for the CHERI CPU and `cheribuild.py [options] run-riscv64-purecap` will start an instance of CheriBSD built for RISC-V in QEMU.

`cheribuild.py` also allows building software for Arm's adaption of CHERI, the Morello platform, however not all targets are supported yet.

### Supported operating systems

`cheribuild.py` has been tested and should work on FreeBSD 11 and 12. On Linux, Ubuntu 16.04, Ubuntu 18.04 and OpenSUSE Tumbleweed are supported. Ubuntu 14.04 may also work but is no longer tested. macOS 10.14 and newer is also supported.

## Pre-Build Setup

**macOS**

When building on macOS the following packages are required:

```
brew install cmake ninja libarchive git glib automake autoconf coreutils llvm make wget pixman
# Install samba for shared mounts between host and CheriBSD on QEMU
brew install arichardson/cheri/samba
# If you intend to run the morello FVP model you will also need the following:
brew install homebrew/cask/docker homebrew/cask/xquartz socat dtc
```

**Ubuntu**

If you are building CHERI on a Debian/Ubuntu-based machine, please install the following packages:

```
apt-get install libtool pkg-config clang bison cmake ninja-build samba flex texinfo libglib2.0-
```

Older versions of Ubuntu may report errors when trying to install `libarchive-tools`. In this case try using `apt-get install bsdtar` instead.

**RHEL/Fedora**

If you are building CHERI on a RHEL/Fedora-based machine, please install the following packages:

```
dnf install libtool clang-devel bison cmake ninja-build samba flex texinfo glib2-devel pixman-d
```

## Basic usage

If you want to start up a QEMU VM running CheriBSD run `cheribuild.py run-riscv64-purecap -d` (-d means

- One build tool to rule them all: cheribuild

  https://github.com/CTSRD-CHERI/cheribuild

- Builds, installs, and/or runs:
  - QEMU CHERI-RISC-V and Morello, Morello FVP
  - CheriBSD/CHERI-RISC-V and Morello disk images
  - Small suite of adapted third-party applications
- Up and running with one command (CHERI-RISC-V):

  ./cheribuild.py --include-dependencies run-riscv64-purecap

## CHERI/MORELLO DESKTOP STUDY

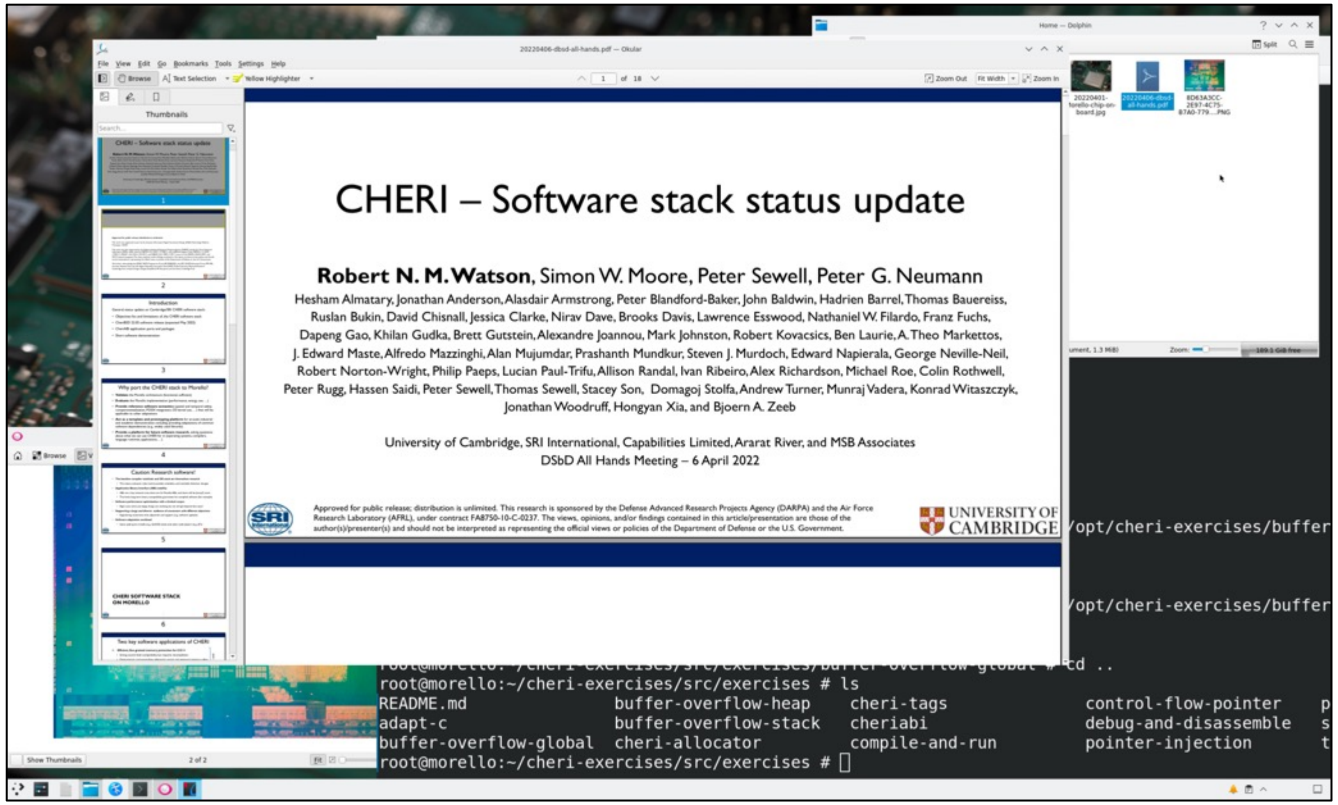
# 3-month CHERI Desktop UKRI pilot study

InnovateUK-funded project at Capabilities Limited to assess the viability of a CHERI/Morello open-source desktop software stack (on QEMU model):

- Selected slice of open-source desktop stack: X11, Qt, KDE, applications
- Implemented CHERI C/C++ referential and spatial memory protection
- Whiteboarded possible software compartmentalizations
- Evaluated software change as %LoC changed
- Evaluated security via 5-year retrospective vulnerability analysis

http://www.capabilitieslimited.co.uk/pdfs/20210917-capltd-cheri-desktop-report-version1-FINAL.pdf

# CHERI desktop ecosystem study: Key outcomes



CHERI – Software stack status update

**Robert N. M. Watson**, Simon W. Moore, Peter Sewell, Peter G. Neumann
Hesham Almatary, Jonathan Anderson, Alasdair Armstrong, Peter Blandford-Baker, John Baldwin, Hadrien Barrel, Thomas Bauereiss, Ruslan Bukin, David Chisnall, Jessica Clarke, Nirav Dave, Brooks Davis, Lawrence Esswood, Nathaniel W. Filardo, Franz Fuchs, Dapeng Gao, Khilan Gudka, Brett Gutstein, Alexandre Joannou, Mark Johnston, Robert Kovacsics, Ben Laurie, A. Theo Markettos, J. Edward Maste, Alfredo Mazzinghi, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch, Edward Napierala, George Neville-Neil, Robert Norton-Wright, Philip Paeps, Lucian Paul-Trifu, Allison Randal, Ivan Ribeiro, Alex Richardson, Michael Roe, Colin Rothwell, Peter Rugg, Hassen Saidi, Peter Sewell, Thomas Sewell, Stacey Son, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Konrad Witaszczyk, Jonathan Woodruff, Hongyan Xia, and Bjoern A. Zeeb

University of Cambridge, SRI International, Capabilities Limited, Ararat River, and MSB Associates
DSbD All Hands Meeting – 6 April 2022

Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237. The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

```
/opt/cheri-exercises/buffer

/opt/cheri-exercises/buffer

                                                                  cd ..
root@morello:~/cheri-exercises/src/exercises # ls
README.md               buffer-overflow-heap    cheri-tags              control-flow-pointer    p
adapt-c                 buffer-overflow-stack   cheriabi                debug-and-disassemble   s
buffer-overflow-global  cheri-allocator         compile-and-run         pointer-injection       t
root@morello:~/cheri-exercises/src/exercises # 
```

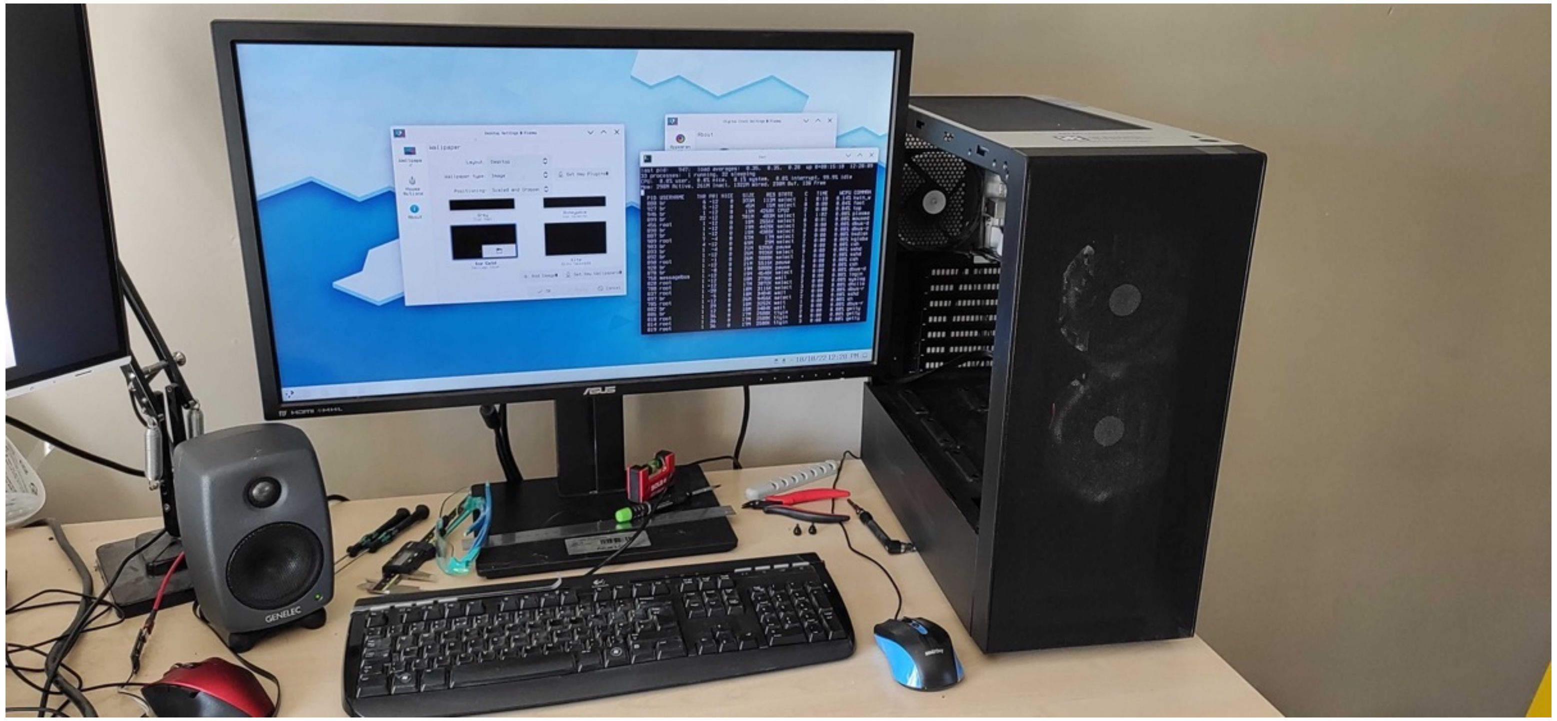
Developed:

- **6 million lines of C/C++ code** compiled for memory safety; modest dynamic testing
- **Three compartmentalization case studies** in Qt/KDE

Evaluation results:

- **0.026% LoC modification rate** across full corpus for memory safety
- **73.8% mitigation rate** across full corpus, using memory safety and compartmentalization

# Memory-safe Morello desktop environment



- Single FTE project over one calendar year developed:
  - Pure-capability CheriBSD kernel GPU device drivers
  - Pure-capability CheriBSD userspace including Mesa, Wayland, KDE

## Now on to the grand challenges

- We are now within reach of an exciting – and historically highly vulnerable – application corpus to which we can apply CHERI protections
- Memory-safe desktop applications at scale – especially those that contain one or more language runtimes:
  - Web browsers
  - Mail readers
  - Office suites
- Extending this to fine-grained compartmentalization as software prototypes mature – library compartmentalization, coprocesses, further models, …
- For example: UKRI- and Google-funded efforts around the Chromium web browser at CapLtd, Kings College London, Arm, and Cambridge

## CONCLUSION

# Some potential software research areas

- **Clean-slate OSes and languages**

  Current research has focused on incremental CHERI adoption within current software and languages. How would we design new OSes, languages, etc., assuming CHERI as an ISA baseline?

- **Compilers, language runtimes, and JITs**

  How can we mitigate the performance overheads of more pointer-dense executions, such as with language runtimes? Are vulnerabilities in code generated by compilers and JIT susceptible to mitigation using CHERI? How does CHERI break or potentially improve current compiler analyses and optimization?

- **Further C/C++ protections with CHERI**

  We have focused on spatial, referential, and temporal memory safety for C/C++. But the CHERI primitives could assist with data-oriented protections, garbage collection, type checking, etc. Could these improve security, and at what performance cost?

- **Safe and managed languages**

  Languages such as Java, Rust, C#, OCaml, etc., offer strong safety properties, but frequently depend on C/C++ runtimes and FFI-linked native code. Can CHERI provide stronger foundations for higher-level language stacks?

- **Virtualization**

  Can memory protection usefully harden hypervisors? Can we compartmentalize hypervisors? Can CHERI offer a better mechanism for virtualizing code than an MMU?

- **Debuggers and tracing**

  Debugging/tracing tools rely on high levels of privilege to operate. How can we reduce their privilege to mitigate vulnerabilities in these tools? With stronger architectural semantics, is new dynamic analysis possible?

- **Software compartmentalization tools**

  Granular software compartmentalization offers vulnerability mitigation through privilege reduction and strong encapsulation. How should current applications be refactored, and new applications be designed, to accomplish maintainable and more secure software?

- **Security evaluation and adversarial research**

  What is the impact of CHERI on known vulnerabilities and attack techniques? How does a CHERI-aware attacker change their behavior? Could formal models and proofs support stronger security arguments for CHERI?

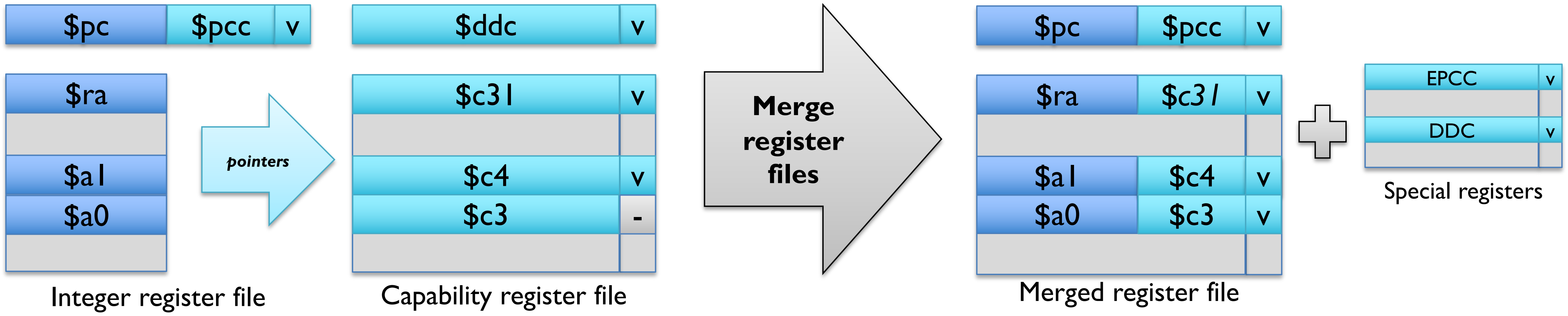
## Conclusion

- New architectural primitives require rich HW and SW evaluation:
  - Primitives support many potential usage patterns, use cases
  - Applicable uses depend on compatibility, performance, effectiveness
  - Best validation approach: full hardware-software prototype
  - Co-design methodology: hardware ↔ architecture ↔ software

http://www.cheri-cpu.org/

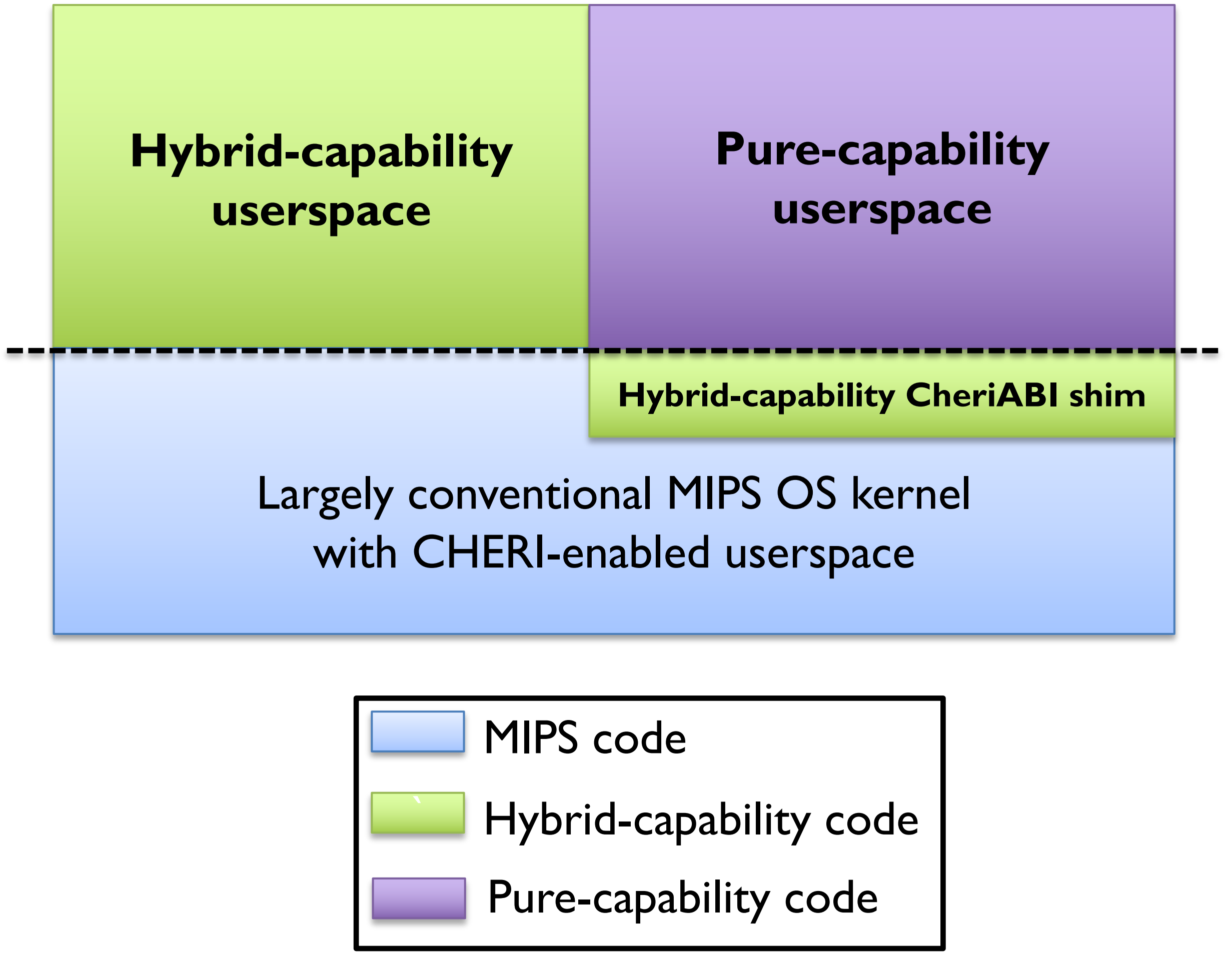
- Watson, et al. **An Introduction to CHERI**, Technical Report UCAM-CL-TR-941, Computer Laboratory, September 2019.
- Watson, et al. **Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 8)**, UCAM-CL-TR-951, October 2020.
- Watson, et al. **CHERI C/C++ Programming Guide**, UCAM-CL-TR-947, June 2020.

# Lessons learned: Split vs. merged register files



- CHERI-MIPS has **split register files** following coprocessor conventions
- … but new register files add control logic, increasing area overhead
- Instead merge register files along the lines of 32-bit → 64-bit extension
- Key design choice in CHERI-RISC-V: Implement both approaches, evaluate

# From hybrid-capability code to pure-capability code


Hybrid-capability userspace
Pure-capability userspace
Hybrid-capability CheriABI shim
Largely conventional MIPS OS kernel with CHERI-enabled userspace
MIPS code
Hybrid-capability code
Pure-capability code

- **n64 MIPS ABI:** hybrid-capability code
  - **Early investigation** – manual annotation and C semantics
  - Many pointers are integers (including syscall arguments, most implied VAs)
- **CheriABI:** pure-capability code
  - **More recently** – fully automatic use of capabilities wherever possible
  - All pointers, implied virtual addresses are capabilities (inc. syscall arguments)
- Now investigating pure-capability kernel

# OS changes required for CheriABI

(A grand tour of low-level OS behavior)

**Hybrid ABI** = MIPS ABI + …

- Kernel support for tagged memory, capability context switching, etc.
- Tag-preserving libc: memory copy, memory move, sort, …
- Bounds-aware malloc(), realloc(), free(), …
- setjmp(), longjmp(), sigcontext / signal delivery, pthreads updates for capabilities
- Run-time linkage for capability-based references to globals, code, vtables, etc. (bounds, permissions, …)
- Debugging APIs such as ptrace()

**CheriABI** = Hybrid ABI + …

- Kernel support for pure-capability userspace
- C start-up/runtime (CSU/CRT) changes
- Initial process state: reduced initial capability registers, ELF aux args, sigcode, etc.
- Pointer arguments/return values for syscalls are now capabilities, …
- Review and fix tag preservation, integer/pointer provenance and casts
- Run-time linkage for globals, code, vtables, etc. (bounds, permissions, …)

# Evaluating memory-protection compatibility

**Approach**: Prototype (1) "pure-capability" **CHERI C/C++ compiler** (Clang/LLVM) and (2) **full OS** (FreeBSD) that use capabilities for all explicit or implied userspace pointers

**Goal**: **Little or no software modification** (BSD base system + utilities)

Small changes to source files for 34 of 824 programs, 28 of 130 libraries.
Overall: modified ~200 of ~20,000 user-space C files/header

| | Pointer + integer integrity, prov. | Pointer size & alignment | Monotonicity | Calling conventions | Unsupported features |
|---|---|---|---|---|---|
| **BSD headers** | 11 | 6 | 0 | 2 | 0 |
| **BSD libraries** | 83 | 36 | 4 | 41 | 22 |
| **BSD programs** | 24 | 9 | 1 | 11 | 2 |

| | Pass | Fail* | Skip | Total |
|---|---|---|---|---|
| **MIPS** | 3501 (91%) | 90 | 244 | 3835 |
| **Pure capability** | 3301 (90%) | 122 | 246 | 3669 |

* Test failure investigation remains a work-in progress; we believe these can be resolved

# Evaluating memory-protection impact

- Adversarial / historical vulnerability analysis
  - ✓ Pointer integrity, provenance validity prevent ROP, JOP
  - ✓ Buffer overflows: Heartbleed (2014), Cloudbleed (2017)
  - ✓ Pointer provenance: Stack Clash (2017)
- Existing test suites – e.g., BOdiagsuite (buffer overflows)

| | OK | min | med | large |
|---|---|---|---|---|
| mips64 | 0 | 4 | 8 | 175 |
| CheriABI | 0 | 279 | 289 | 291 |
| LLVM Address Sanitizer (asan) on x86 | 0 | 276 | 286 | 286 |

- Davis, et al. **CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment**, ASPLOS 2019.
- Key evaluation concern: reasoning about a **CHERI-aware adversary**