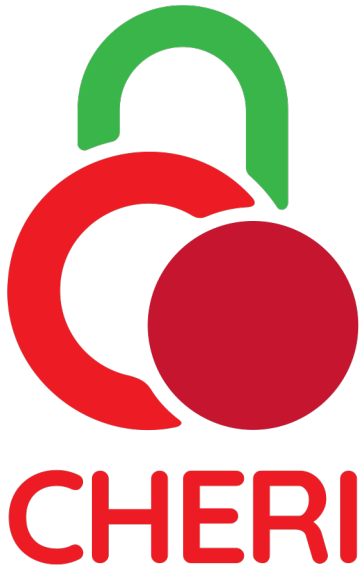
CHERI and Morello: Arming systems with hardware-enforced memory safety capabilities



Jessica Clarke
jessica.clarke@cl.cam.ac.uk
jrtc27@debian.org

MiniDebConfCambridge 2023

SRI UNIVERSITY OF CAMBRIDGE

Approved for public release; distribution is unlimited.

This work was supported in part by the Innovate UK project 105694 (“Digital Security by Design (DSbD) Technology Platform Prototype”, and Innovate UK project 10027440 (“Developing and Evaluating an Open-Source Desktop for Arm Morello”).

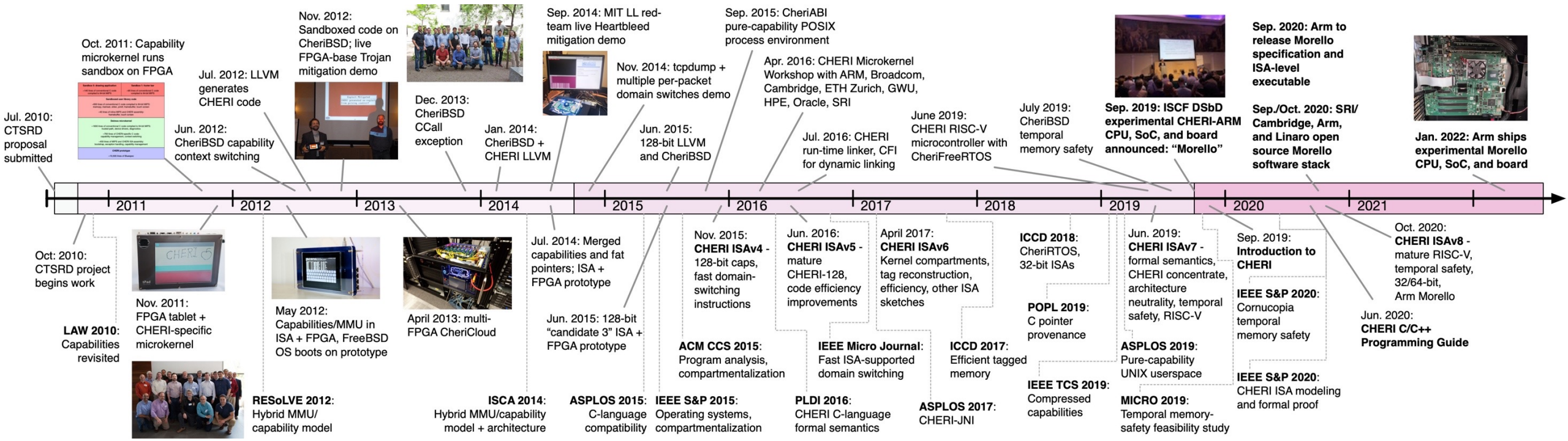
This work was also supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237 (“CTSRD”), with additional support from FA8750-11-C-0249 (“MRC2”), HR0011-18-C-0016 (“ECATS”), FA8650-18-C-7809 (“CIFV”), HR001122C0110 (“ETC”), and HR001123C0031 (“MTSS”) as part of the DARPA I2O CRASH, I2O MRC, and MTO 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 EPSRC REMS (EP/K008528/1), EPSRC CHaOS (EP/V000292/1), ERC ELVER (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

SRI UNIVERSITY OF CAMBRIDGE

CHERI Research and Development Timeline



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 CHERIoT

CHERI

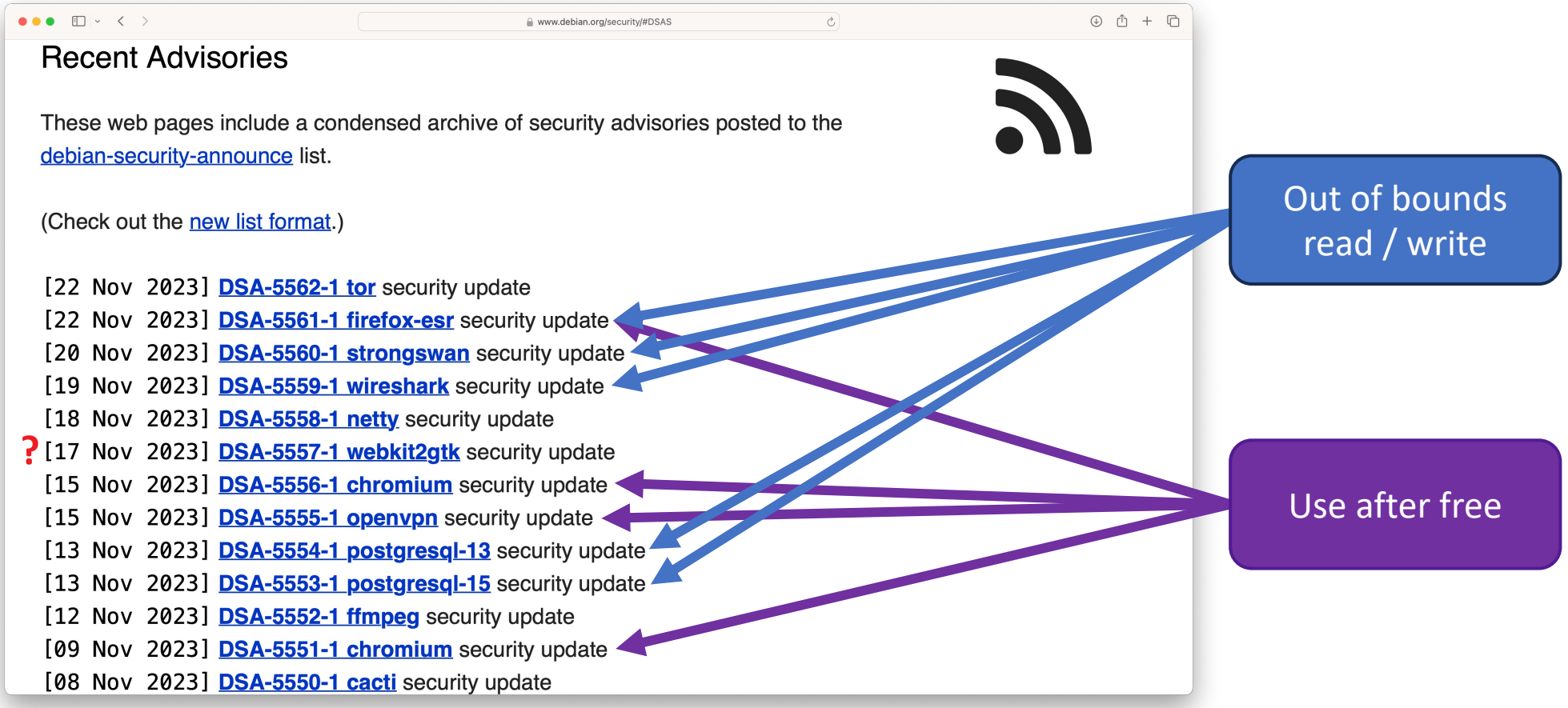
SRI

UNIVERSITY OF CAMBRIDGE

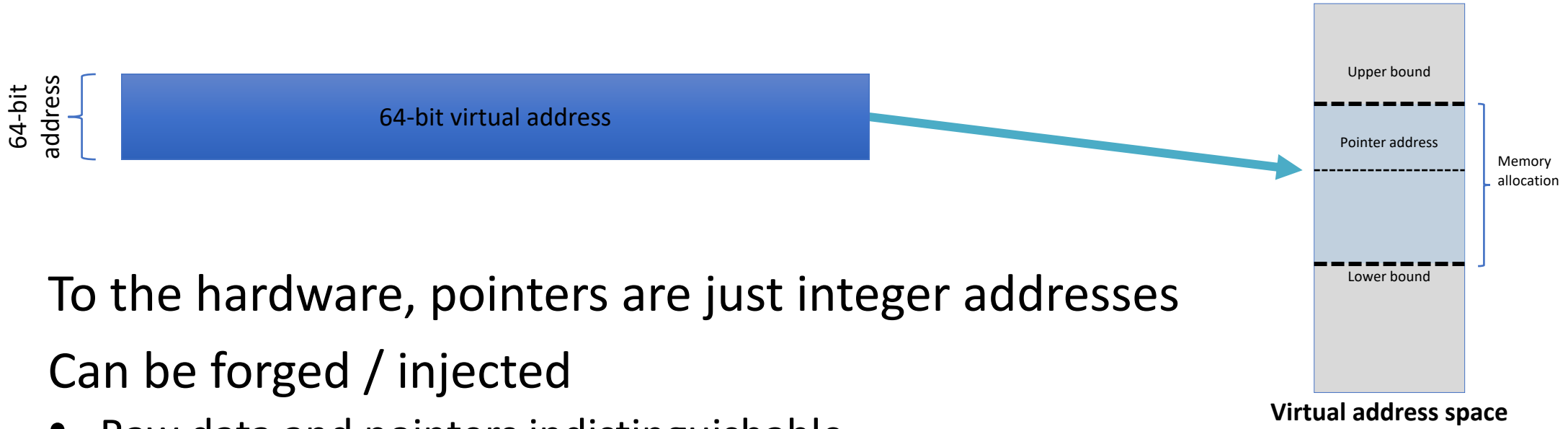
35 Years Ago

- 2nd November 1988: Morris Worm appeared
- Four methods of propagation
  1. rsh to host that trusts this one
  2. rexec with same username and password as local system
  3. sendmail compiled with debug mode (command injection) enabled
  4. Buffer overflow of on-stack string buffer in fingerd

Two Weeks of DSAs



Pointers Today

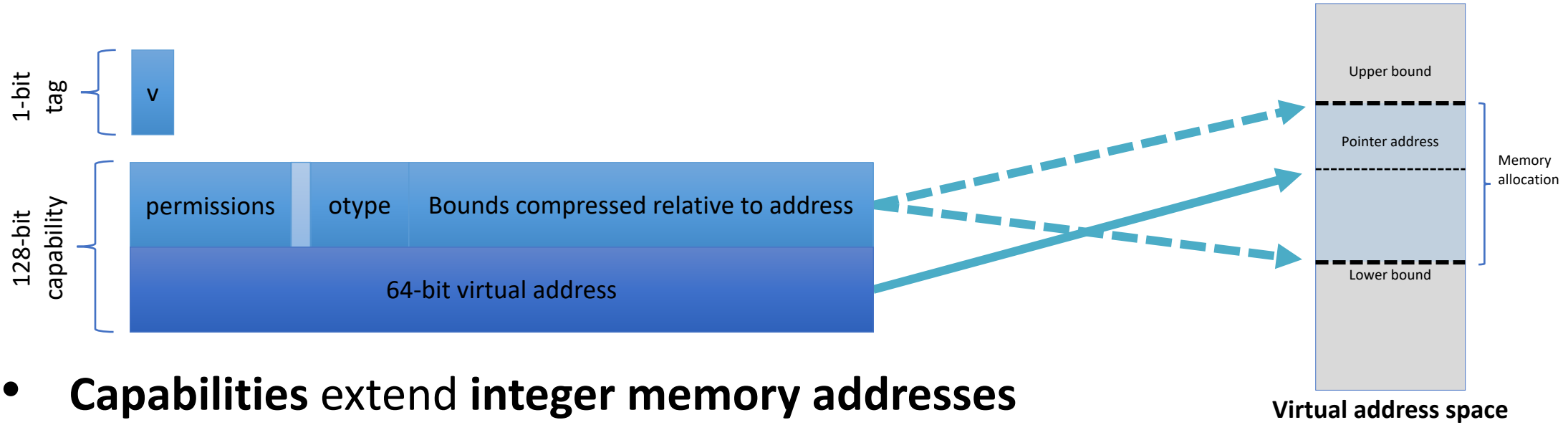


- To the hardware, pointers are just integer addresses
- Can be forged / injected
  - Raw data and pointers indistinguishable
- No programmer intent conveyed
  - Just check that the address is mapped

Some Limited Solutions

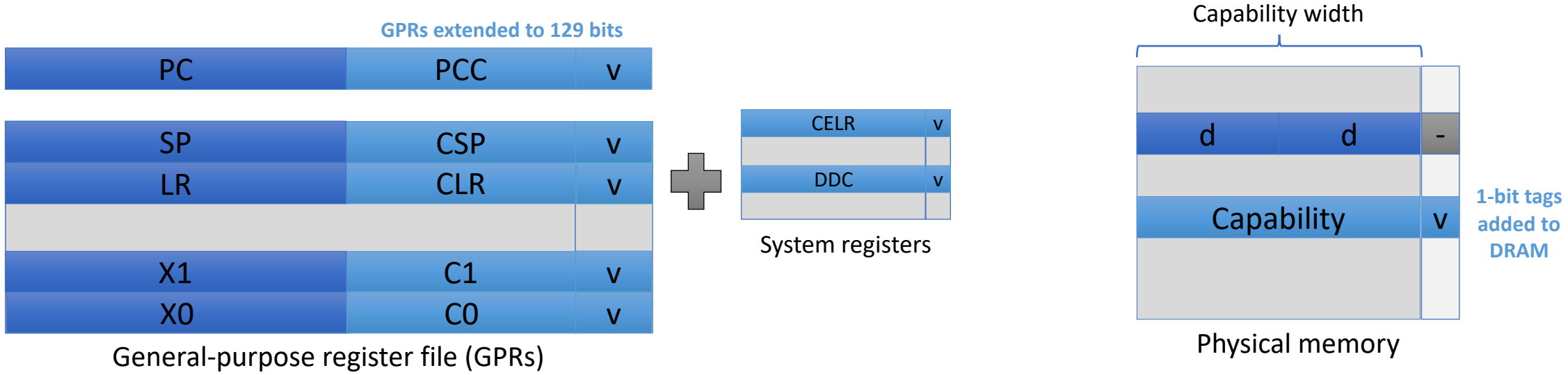
- Various extensions exist to make attacks harder: BTI, PAC, shadow stack, MTE, …
- Generally suffer from at least one of:
  - Probabilistic
  - Rely on secrets
  - Target symptom not cause
  - Target secondary cause

CHERI Capabilities



- **Capabilities** extend **integer memory addresses**
- **Bounds** restrict the **range** of memory addresses they can access
- **Permissions** restrict **how** the capability can be used (e.g. read-only)
- **Tags** protect capability integrity/derivation in registers + memory
- **Guarded manipulation** controls how capabilities may be manipulated; e.g., **provenance validity** and **monotonicity**

Capabilities in Registers and Memory



- **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., new TLB/PTE permission bits, exception codes, exception/interrupt vectors etc.)

Hardware Prototypes

- Original research used home-grown pipelined “BERI” MIPS core (CHERI-MIPS)
- Transitioned 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 out-of-order, 64-bit, MMU
- Novel microarchitectural contributions include **capability compression model, tagged memory implementation techniques**
- All our CPU designs are open source
- QEMU full-system and user-level simulators for CHERI-RISC-V and Morello
- **Arm Morello and Microsoft CHERIoT (later slides)**

Microsoft CHERIoT (2023)

CHERIoT: Complete Memory Safety for Embedded Devices

Saar Amar*
saaramar5@gmail.com
Microsoft
Tel Aviv, Israel

David Chisnall*
David.Chisnall@cl.cam.ac.uk
Microsoft
Cambridge, UK

Tony Chen
tonychen@microsoft.com
Microsoft
Redmond, Washington, USA

Nathaniel Wesley Filardo*
nwf20@cam.ac.uk
Microsoft
Cambridge, UK

Ben Laurie
benl@google.com
Google
London, UK

Kunyan Liu*
kunyanliu@microsoft.com
Microsoft
San Diego, California, USA

Robert Norton*
robert.norton@microsoft.com
Microsoft
Cambridge, UK

Simon W. Moore
Simon.Moore@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Yucong Tao
Yucong.Tao@microsoft.com
Microsoft
Mountain View, California, USA

Robert N. M. Watson
robert.watson@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Hongyan Xia†*
Jerryxia32@gmail.com
Arm Ltd.
Cambridge, UK

ABSTRACT

The ubiquity of embedded devices is apparent. The desire for increased functionality and connectivity drives ever larger software stacks, with components from multiple vendors and entities. These stacks *should* be replete with isolation and memory safety technologies, but existing solutions impinge upon development, unit cost, power, scalability, and/or real-time constraints, limiting their adoption and production-grade deployments. As memory safety vulnerabilities mount, the situation is clearly not tenable and a new approach is needed.

To slake this need, we present a novel adaptation of the CHERI capability architecture, co-designed with a green-field, security-centric RTOS. It is scaled for embedded systems, is capable of fine-grained software compartmentalization, and provides affordances for full inter-compartment memory safety. We highlight central design decisions and offloads and summarize how our prototype RTOS uses these to enable memory-safe, compartmentalized applications. Unlike many state-of-the-art schemes, our solution deterministically (not probabilistically) eliminates memory safety vulnerabilities while maintaining source-level compatibility. We characterize the power, performance, and area microarchitectural impacts, run microbenchmarks of key facilities, and exhibit the practicality of an end-to-end IoT application. The implementation shows that full memory safety for compartmentalized embedded systems is achievable without violating resource constraints or real-time guarantees, and that hardware assists need not be expensive, intrusive, or power-hungry.

*These authors made significant contributions to the design and implementation without which the project would not have been possible.
†Work conducted while at Microsoft.

MICRO '23, October 28–November 01, 2023, Toronto, ON, Canada

ACM ISBN 979-8-4007-0329-4/23/10.
https://doi.org/10.1145/3613424.3614266

ACM Reference Format:
Saar Amar, David Chisnall, Tony Chen, Nathaniel Wesley Filardo, Ben Laurie, Kunyan Liu, Robert Norton, Simon W. Moore, Yucong Tao, Robert N. M. Watson, and Hongyan Xia. 2023. CHERIoT: Complete Memory Safety for Embedded Devices. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28–November 01, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3613424.3614266

1 INTRODUCTION

The attack surface of embedded devices is no longer limited to physical attacks, in an increasingly connected world. From consumer electronics (smart watches, WiFi chips) to security-critical devices (self-driving vehicles, aviation and smart grids) and more recently IoT applications, physical isolation is rarely the boundary in modern day embedded devices. With the increase of connectivity comes combinatorial growth of the attack surface. Sadly, the resource constraints and the low-level programming environment mean solving even the most basic problem of memory safety still poses as a monumental challenge. Worse, the gap between the attack surface area and the level of defense widens further when such embedded devices are deployed into complicated multi-tasking scenarios with a Real-Time Operating System (RTOS) and multiple software stacks from different vendors.

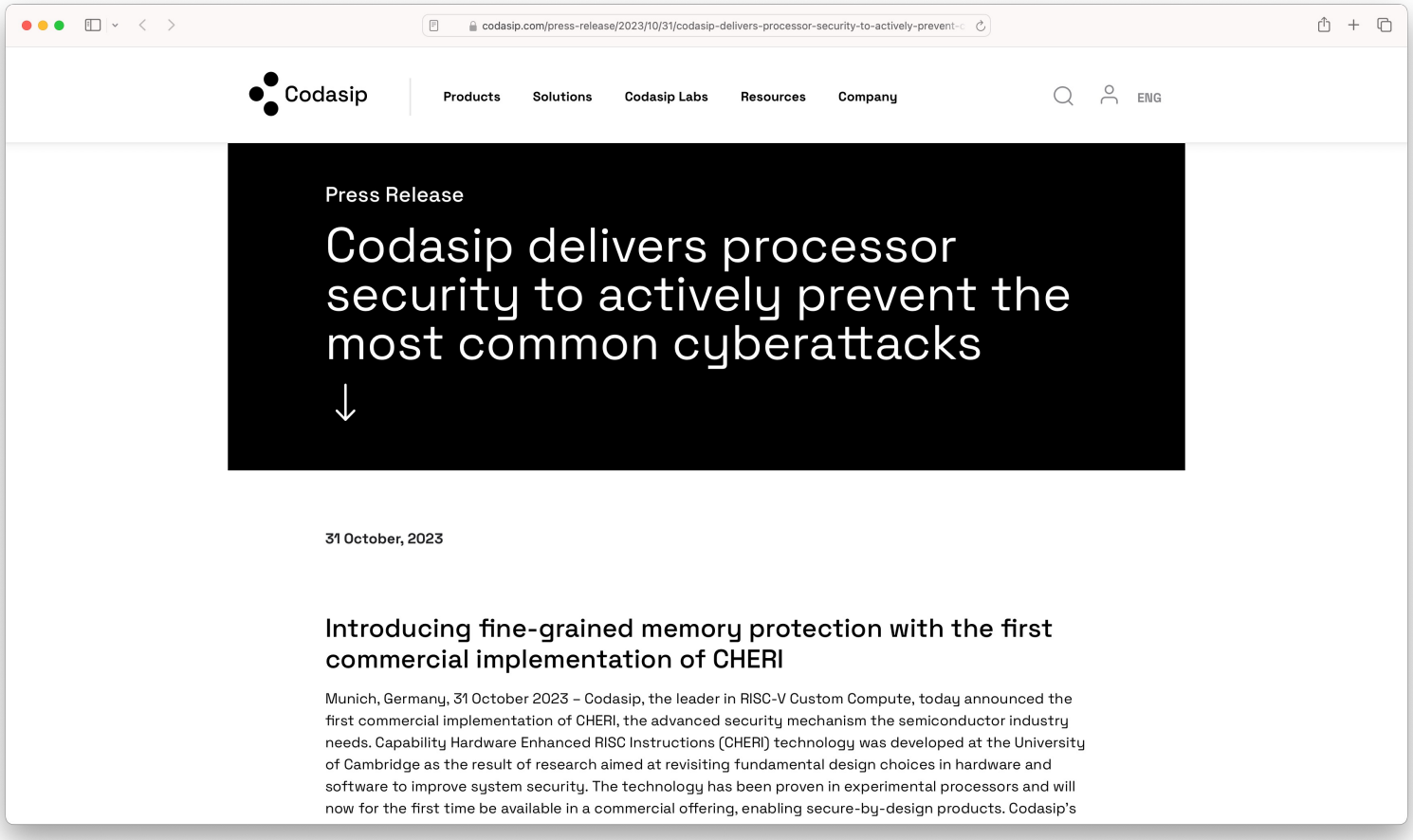
Even though researchers have disclosed an alarming number of memory vulnerabilities in recent years [6, 11, 15], the lessons learned from desktop and server systems do not directly translate to embedded systems. Page table techniques, sanitizers, dynamic

- Production CHERI-extended Ibex microcontroller
  - Small-scale microcontroller used in OpenTitan, etc.
  - CHERI-RISC-V tuned for small microcontrollers
  - Clean-slate memory-safe, compartmentalized embedded OS for high-risk applications
  - Open sourced in February 2023
  - RISC-V embedded standardization candidate
- Collaboration across Microsoft Research, MSRC, Azure Silicon, and Azure Edge + Platform
- lowRISC Sunburst FPGA board reference platform
- Published in IEEE MICRO 2023

CHERI

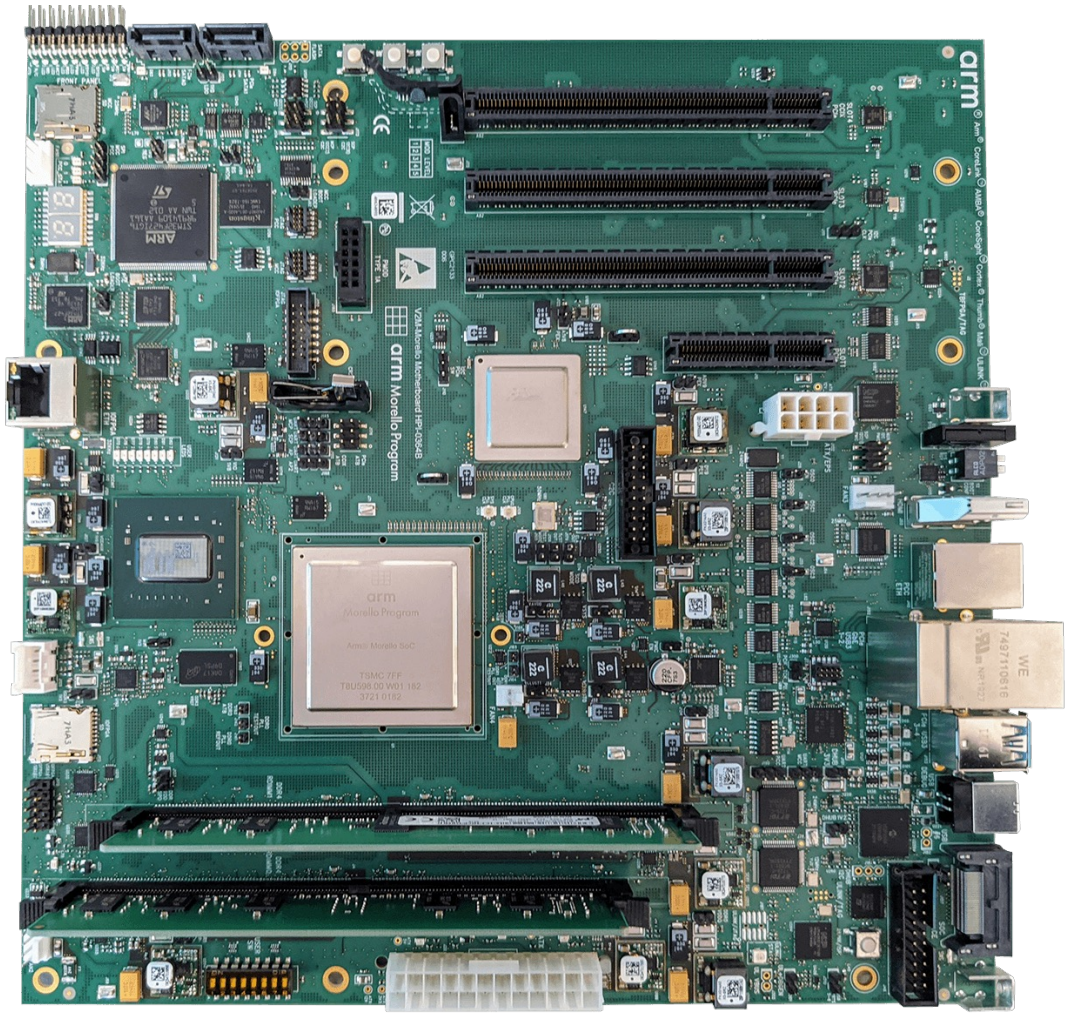
SRI UNIVERSITY OF CAMBRIDGE

Codasip (2023)



DSbD and Arm Morello

- $225M government, academia, and industrial research program led by UK Research and Innovation (UKRI)
  - Announced partners: Arm, Google, Microsoft
  - 15+ UK universities with research grants
  - 70+ funded business incubation projects
- Baseline for design: Neoverse N1 core
  - 2.5GHz quad-core, superscalar
- Roughly a thousand chips manufactured for use by research + development labs



CHERI

SRI UNIVERSITY OF CAMBRIDGE

Pure-Capability ABI

- CHERI introduces new “pure-capability” ABI
- All C/C++ language pointers are CHERI capabilities
  - **NB: includes (u)intptr_t**
- All “sub-language” pointers also CHERI capabilities
  - Return addresses, C++ vtables, GOT and stack pointers, varargs, …
- Provides full always-on CHERI protection
- Often just called “CHERI C/C++”
- Most source code requires few, if any, changes

Hybrid ABI

- Compatible extension of existing non-CHERI ABI
- Capabilities are opt-in:
  - `void *` → `void * __capability`
  - `(u)intptr_t` → `(u)intcap_t`
- Allows interfacing between “legacy” and pure-capability code
- Very limited protection
- Awkward to use at scale
- Not for widespread use, but useful in very specific scenarios

Compatibility

- Familiar with running 32-bit application on 64-bit system: COMPAT in Linux, COMPAT_FREEBSD32 in FreeBSD
- Similarly, can run non-CHERI (or hybrid) 64-bit process on a CHERI system via compatibility interface
  - All your existing binaries continue to run
  - … but no security benefit
- In theory can also run 32-bit applications on a CHERI system, but no current hardware prototypes support 32-bit mode

Subobject Bounds

- Current security extensions generally protect only the **allocation**
- Some vulnerabilities involve overflowing between adjacent "sub-objects" within the **same** allocation
- Our bounds (and permissions) tied to the pointer, not the allocation; can derive subsets
- Compiler can (optionally*) do this automatically
  - `&p->x` → `cheri_bounds_set(&p->x, sizeof(p->x))`

* With varying levels of aggressiveness, at the cost of decreased C compatibility

CHERI

SRI UNIVERSITY OF CAMBRIDGE

Temporal Safety

- So far, illustrated **referential** and **spatial** memory safety
- Tag bit allows us to **find** all capabilities
- On free, “quarantine” allocation: references remain valid, memory not yet repurposed
- When quarantine grows too large, sweep through process’s memory and invalidate (“revoke”) all capabilities to freed memory
- Tricks with special page table bits to allow sweeping concurrently with process execution

Compartmentalisation Scalability

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

  Early benchmarks show 1-to-2 order of magnitude performance improvement for inter-compartment communication compared to conventional designs
- Compartment can only access memory it has capabilities for
- Many potential use cases – e.g., sandbox processing of each image in web browser, processing each message in mail application
- Unlike memory protection, software compartmentalisation requires **careful software refactoring** to support strong encapsulation, and affects software operational model

Compartmentalisation Models

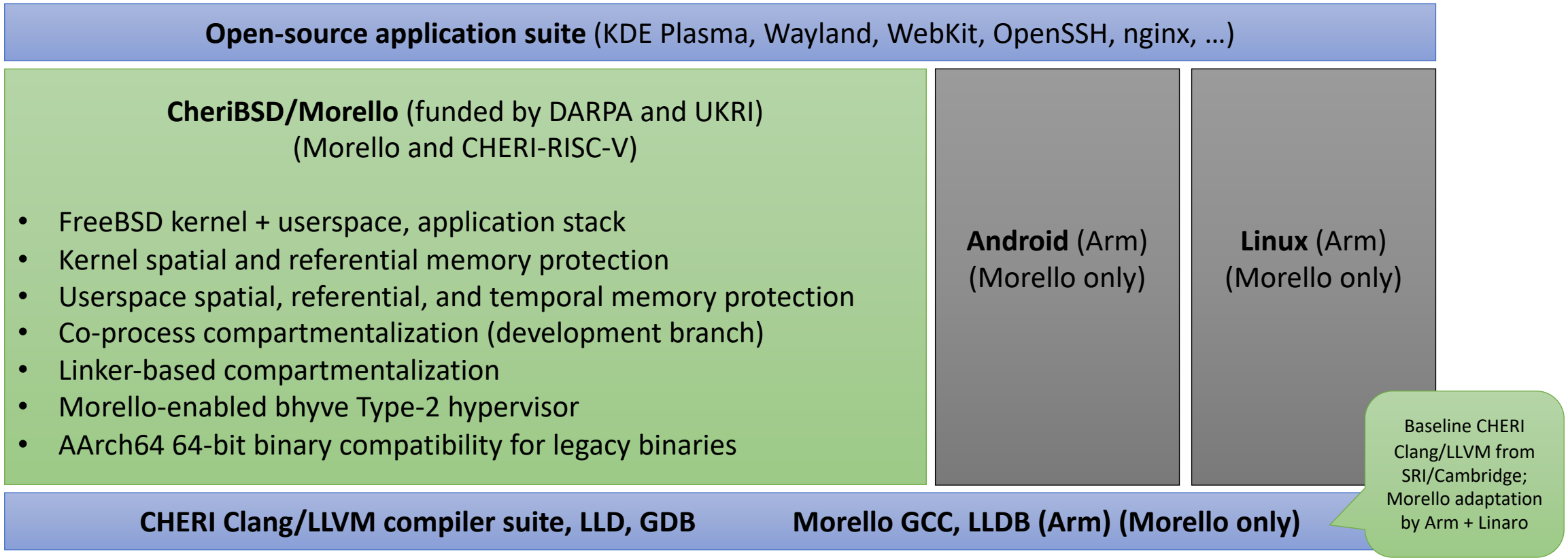
- Two models being explored:

1. Intra-process compartmentalisation
   - Every library is its own compartment
   - Simple programming model – compartment invocation is normal function call
   - Automatically provide additional robustness for unmodified source code

2. Co-process compartmentalisation
   - Multiple processes share address space
   - CHERI allows fast IPC and domain transitions
   - Fits into existing process-based compartmentalisation designs
   - Requires structuring code into multiple processes

Prototype Software Stack

- **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



CHERI C/C++ vs High-Level Languages

Language	Approximate open-source LoC*	Memory safe	Memory safe with CHERI
C	10,317,800,000	✗	✓
C++	2,937,550,000	✗	✓
Java	2,600,000,000	✓	✓
Rust	39,500,000	✓	✓

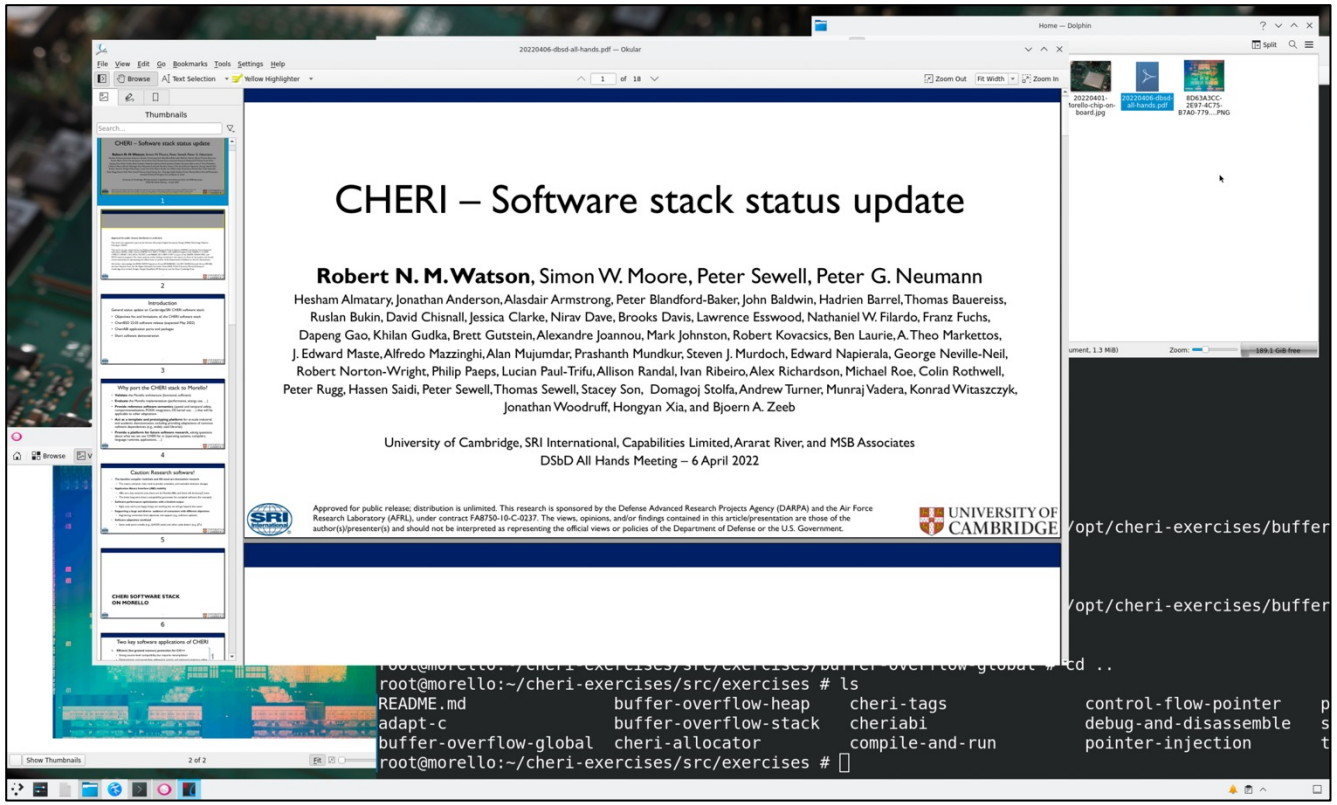
More lines of open-source code have been ported to CHERI C/C++ memory safety than the Rust ecosystem has created in its entire history

CHERI

* Synopsys Black Duck Open Hub: https://www.openhub.net/languages

SRI UNIVERSITY OF CAMBRIDGE

2021 Desktop Pilot Study



Developed:

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

Evaluation results:

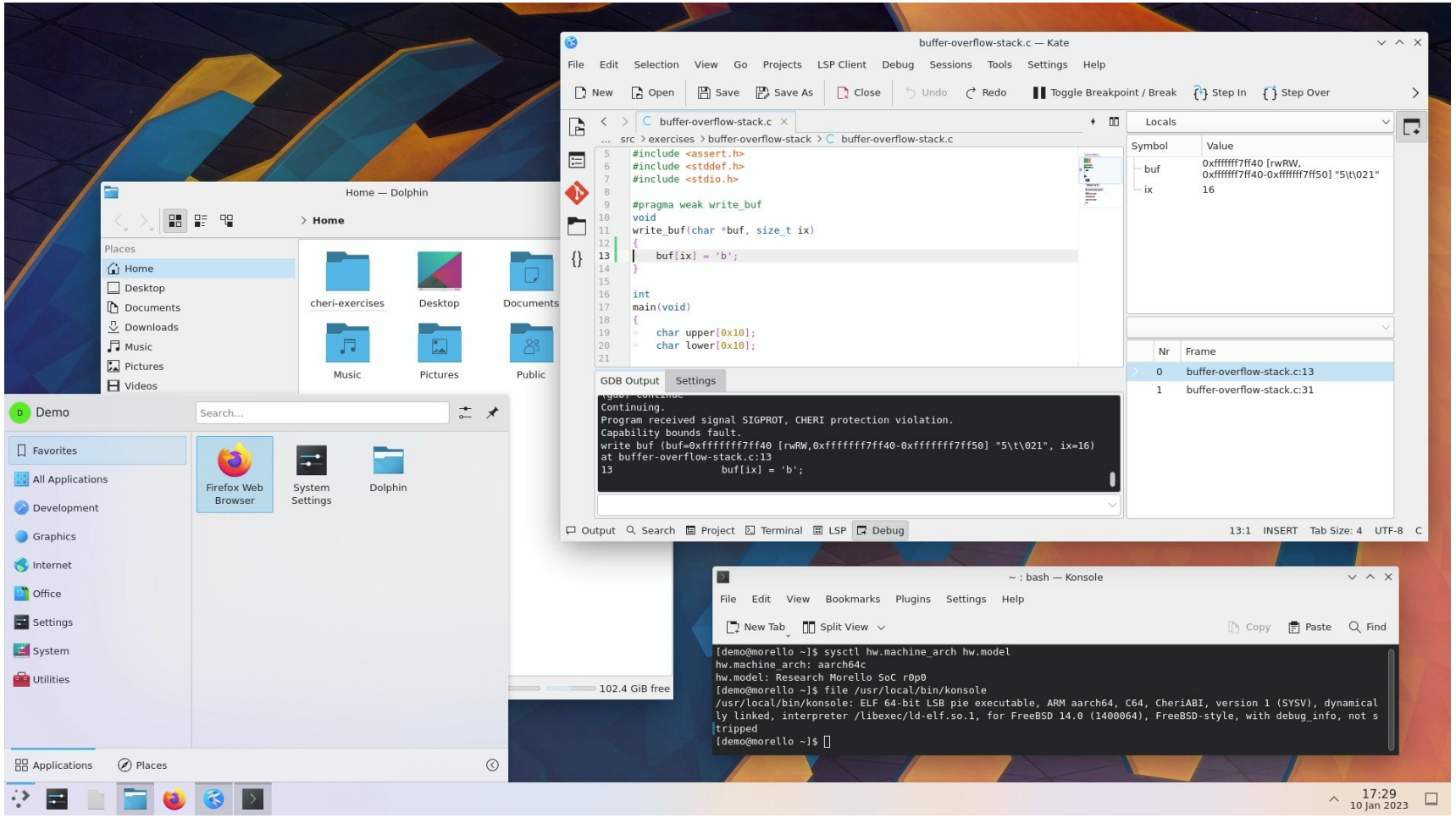
- **0.026% LoC modification rate** across full corpus for memory safety
- **73.8% mitigation rate** across full corpus, using memory safety and compartmentalization

Useful observation to be made about memory safety: also need compartmentalization to address the de facto threat model of quite a few libraries

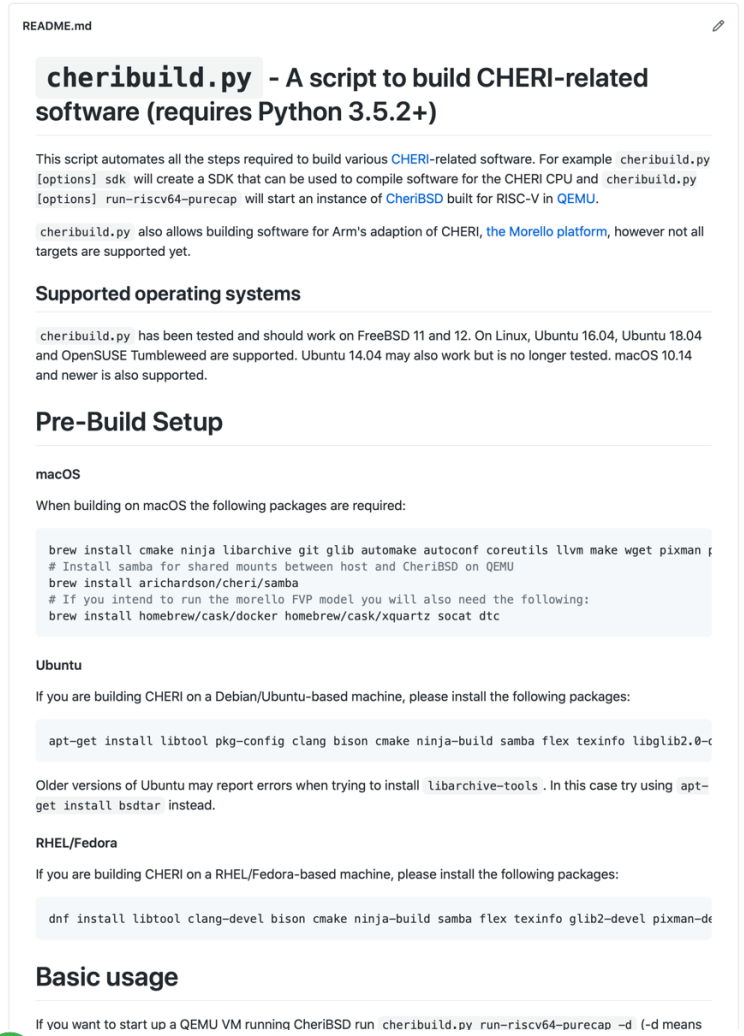
CHERI

SRI UNIVERSITY OF CAMBRIDGE

CHERI Desktop



Obtaining 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 p
# 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-d
```

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-de
```

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:
  - CHERI/Morello QEMU (or Morello FVP)
  - CheriBSD 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
- Pre-built CheriBSD installer for Morello available from https://www.cheribsd.org

CHERI

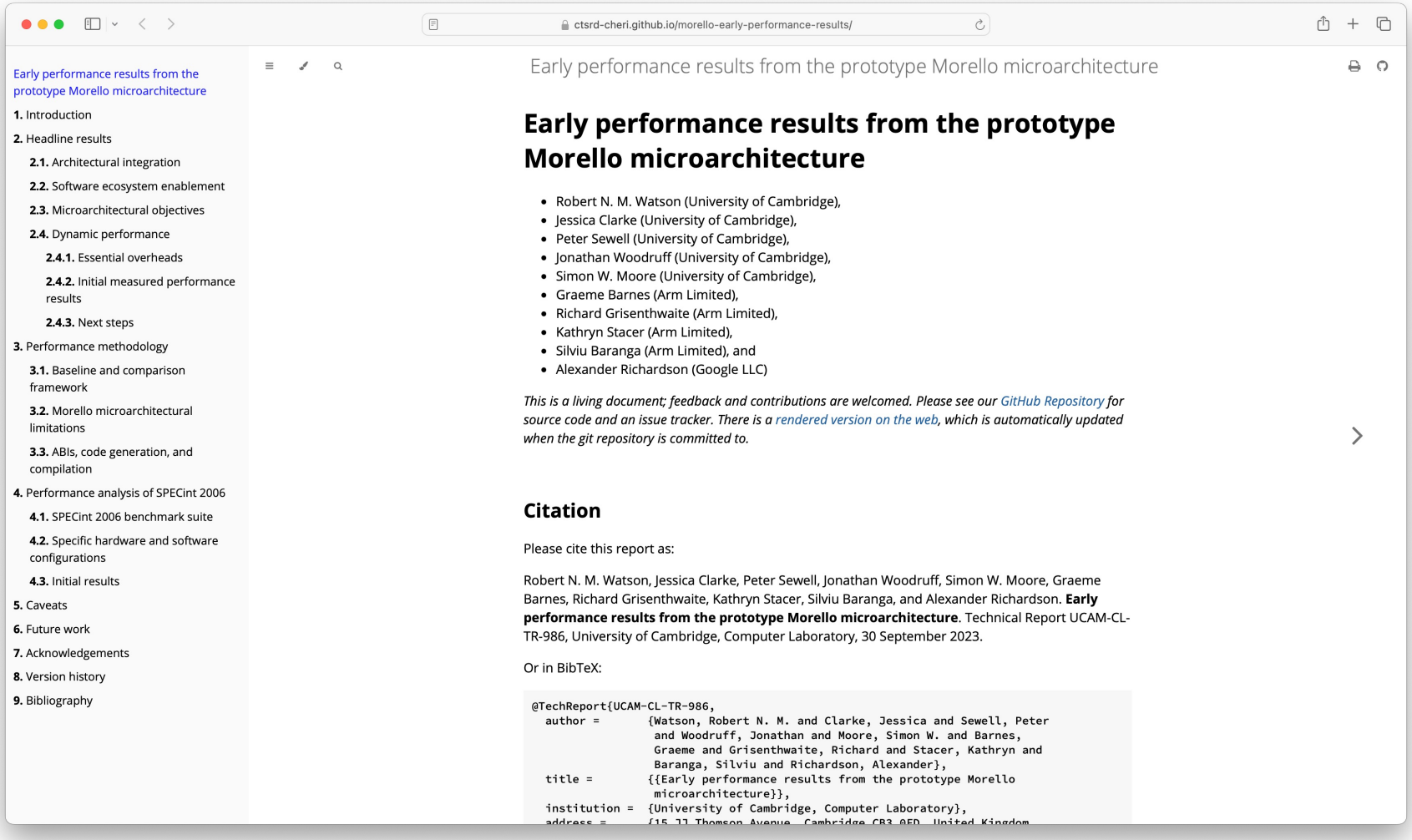
SRI UNIVERSITY OF CAMBRIDGE

Getting Involved

- Testing code on CHERI improves code quality
  - Find potential bugs
  - Find bad assumptions (e.g. pointers <= 8 bytes, uintptr_t == long)
- Hosting board for GCC Compile Farm project, usable for any open source development: cfarm240.cfarm.net
- UK and international organisations can request a Morello board: https://www.dsbd.tech/get-involved/morello-board-request/
- Technical Access Program (UK-only), support and funding for small companies: https://www.dsbd.tech/technology-access-programme/
- Talk to us if interested

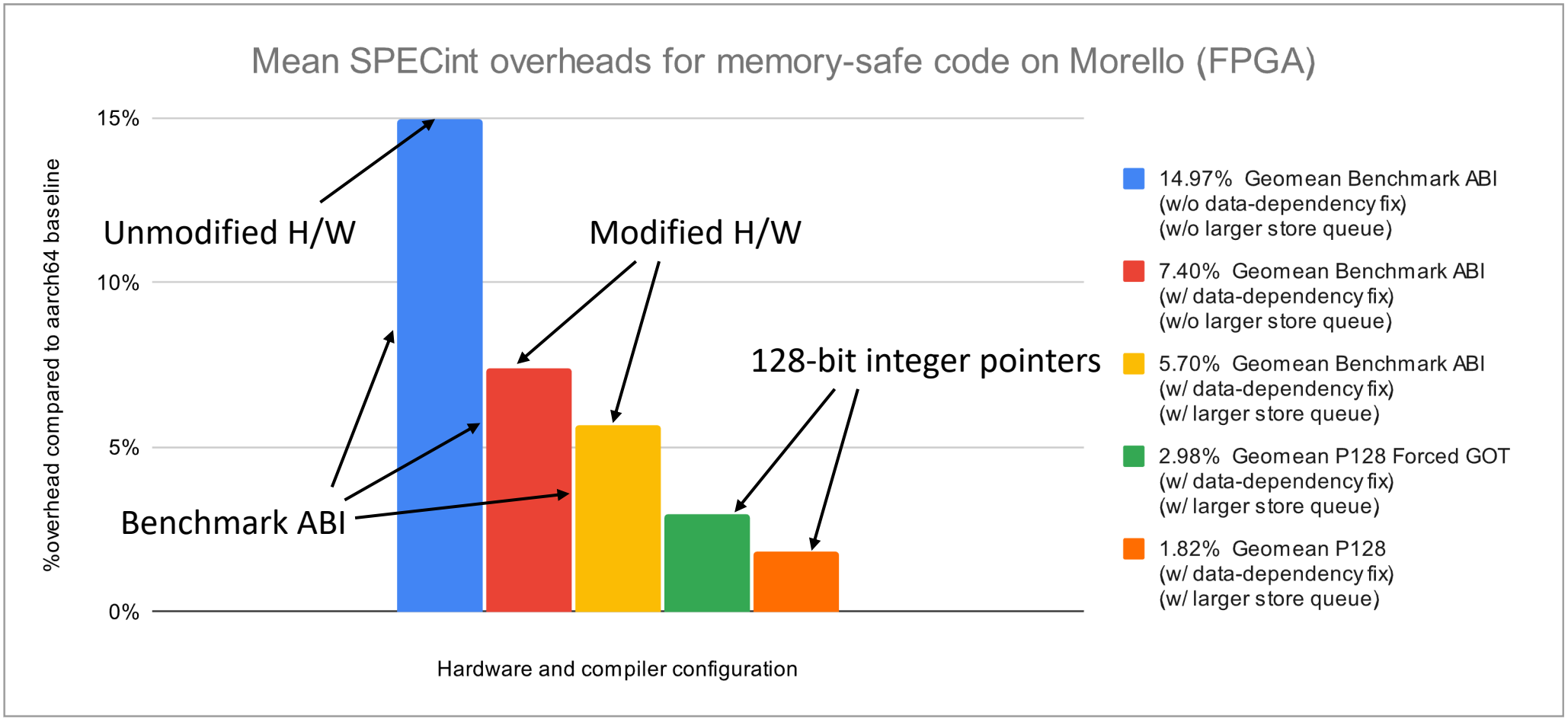
Demo / Q&A

Extra Slides



https://ctsrd-cheri.github.io/morello-early-performance-results/

Headline results



Capability branch prediction

- Microarchitecture only predicts PCC's address in the Morello prototype
  - This is due to the research engineering timeline, lack of optimization data, and desire to avoid floorplan changes
  - Arm has strong confidence that this could be addressed in a production microarchitecture
- Instructions that consume PCC's metadata (e.g. C64 BL/BLR and ADRP) need to wait for prior capability branches (NB: includes RET) to execute
  - Includes ADRP+LDR sequence to load from GOT for globals
- Capability branch-heavy code incurs additional stalls

Benchmark ABI: Overview

- Aims to work around lack of capability branch prediction
- Models expected performance of an improved second-generation microarchitecture
- PCC given bounds for the whole address space
- Indirect branches and returns use integer branches
  - Return addresses and function pointers remain as capabilities in memory; only branches themselves altered
- NB: Weakens control flow protection, **not intended for security evaluation**

Data-dependent exception delivery

- Used to track capabilities for heap temporal safety
  - Deliver a precise exception based on the value stored to memory, not just the address it is stored to
- Not a requirement in the baseline Neoverse N1 design, and as a result there isn't the necessary plumbing to make it microarchitecturally efficient
  - Stores of capabilities stall until both address and data are known
- A similar requirement affects recent Arm microarchitectures
- Modified Morello design on FPGA allows us to experiment with eliminating this overhead

Untuned store queues

- The baseline Neoverse N1 has store-buffer queues (which track in-flight memory stores) tuned to the memory traffic generated by the Armv8-A
  - With a 128-bit bus, “store pair” instructions for 64-bit integers could be issued as a single operation
- Morello has “store pair” instructions for 128-bit capabilities
  - These cannot be satisfied by a single 128-bit memory operation
  - Store pair capability is therefore “cracked” microarchitecturally into two 128-bit operations
  - The store-buffer queue can become full as a result of the potential to double the number of in-flight transactions, stalling memory accesses
  - Modified Morello design on FPGA allows us to experiment with increasing the store-buffer queue size

P128 code generation

- A key conclusion of the Morello project is somewhat expected: that the essential overhead to CHERI is pointer-size growth (64 → 128 bits)
  - Other costs, such as the implementation of tags, capability compression, instruction scheduling, etc., turned out not to be significant in this work
- To understand how a more optimized and mature microarchitecture might perform, we modified Morello LLVM to target the Armv8.2-A ISA while using 128-bit storage for language-level pointers to identify new upper bounds for overheads
  - Sub-language pointers (GOT entries, return addresses, etc.) currently remain as 64-bit integers
  - Treated as 64-bit values when in registers (NB: including spilling to stack)
- Two variants depending on whether (a) all loads and stores are forced through the GOT, or (b) PC-relative loads and stores are used
  - A mature CHERI-enabled compiler would use a combination of the two strategies based on security and performance considerations