# CHERI
# Capability Hardware Enhanced RISC Instructions

**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, Khilan Gudka, Brett Gutstein, Alexandre Joannou, Robert Kovacsics, Ben Laurie, A. Theo Markettos, J. Edward Maste, Alfredo Mazzinghi, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch, Edward Napierala, Robert Norton-Wright, Philip Paeps, Lucian Paul-Trifu, Ivan Ribeiro, Alex Richardson, Michael Roe, Colin Rothwell, Peter Rugg, Hassen Saidi, Peter Sewell, Thomas Sewell, Stacey Son, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Jonathan Woodruff, Hongyan Xia, and Bjoern A. Zeeb

University of Cambridge and SRI International

DSbD Software Ecosystem Workshop – 5 October 2021

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.

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

This work was 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"), and FA8650-18-C-7809 ("CIFV") 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.

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

We also 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.

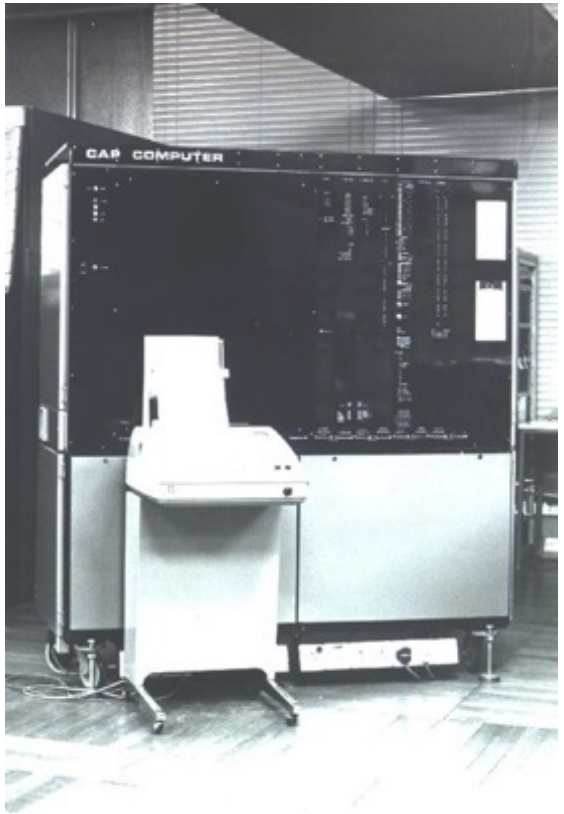
# Introduction

- An introduction to the CHERI architecture and software stack
- 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. **CHERI C/C++ Programming Guide**, UCAM-CL-TR-947, June 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

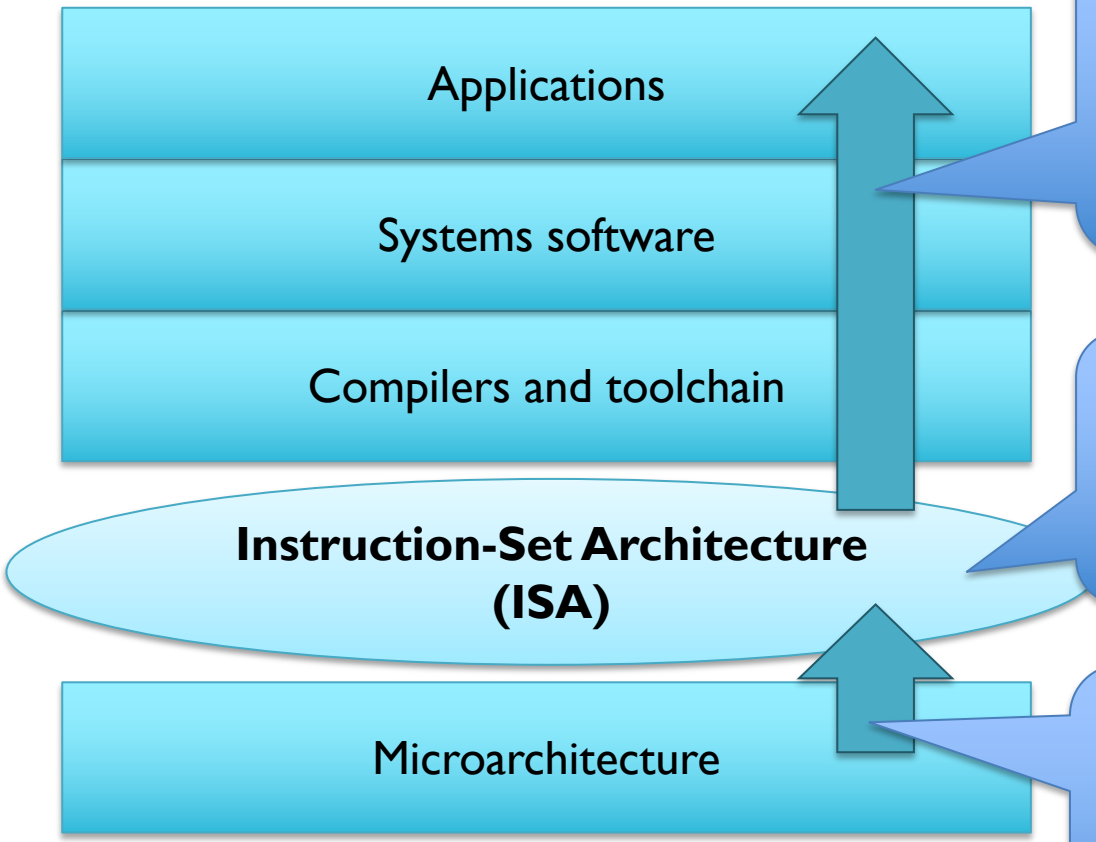
# What is CHERI? (2010-current)

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

- CHERI is an **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/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 scale + quality demonstrator CPU, SoC, board**

# 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**

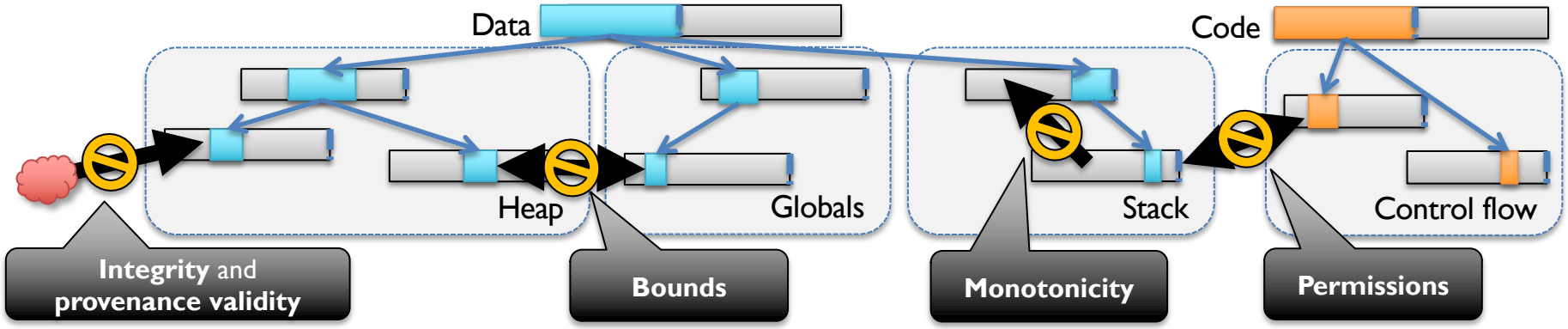
**CHERI capabilities** are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

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, …)

# 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**

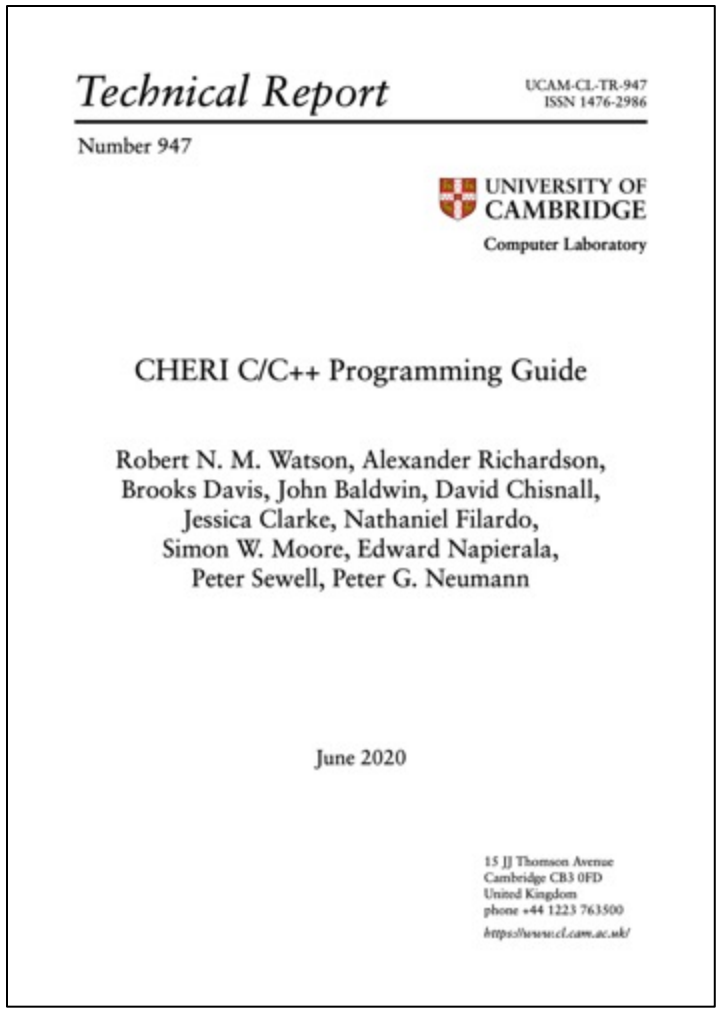
# Two key use cases for CHERI

1. **Efficient, fine-grained memory protection for C/C++**
   - Good source-level compatibility, but ABI disruptive to binaries
   - Supports referential, spatial, and temporal memory safety (with limitations)
   - Generally modest overhead (0%-5%, some workloads 10%)
2. **Scalable software compartmentalization**
   - Multiple software operational models from objects to processes
   - Orders-of-magnitude performance improvement over MMU-based techniques (<90% reduction in overhead in early benchmarks)

Other potential – but under-explored – use cases include within managed language runtimes, and as a substrate for safer inter-language interoperation

# CHERI C/C++ MEMORY PROTECTION

# Memory-safe CHERI C/C++



*Technical Report*

UCAM-CL-TR-947
ISSN 1476-2986

Number 947

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

CHERI C/C++ Programming Guide

Robert N. M. Watson, Alexander Richardson,
Brooks Davis, John Baldwin, David Chisnall,
Jessica Clarke, Nathaniel Filardo,
Simon W. Moore, Edward Napierala,
Peter Sewell, Peter G. Neumann

June 2020

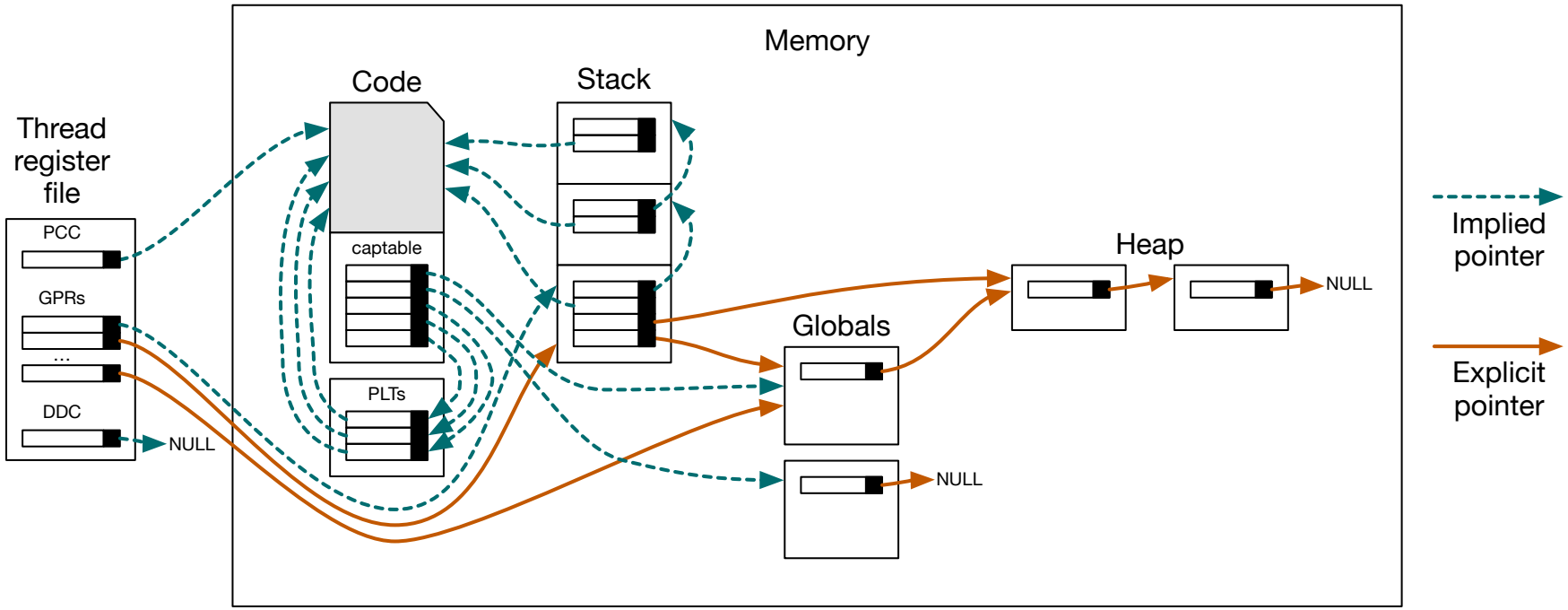
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500
https://www.cl.cam.ac.uk/

- 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

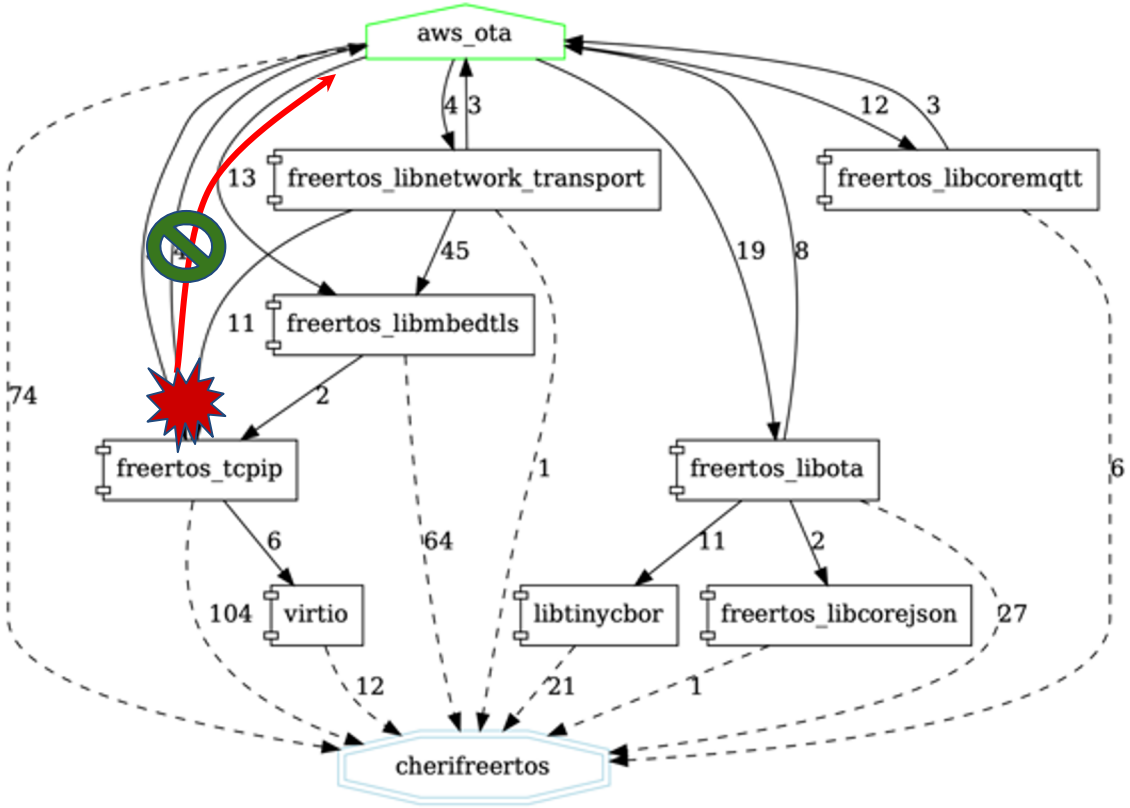
# CHERI-based pure-capability process memory




- 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

# CHERI SOFTWARE COMPARTMENTALISATION

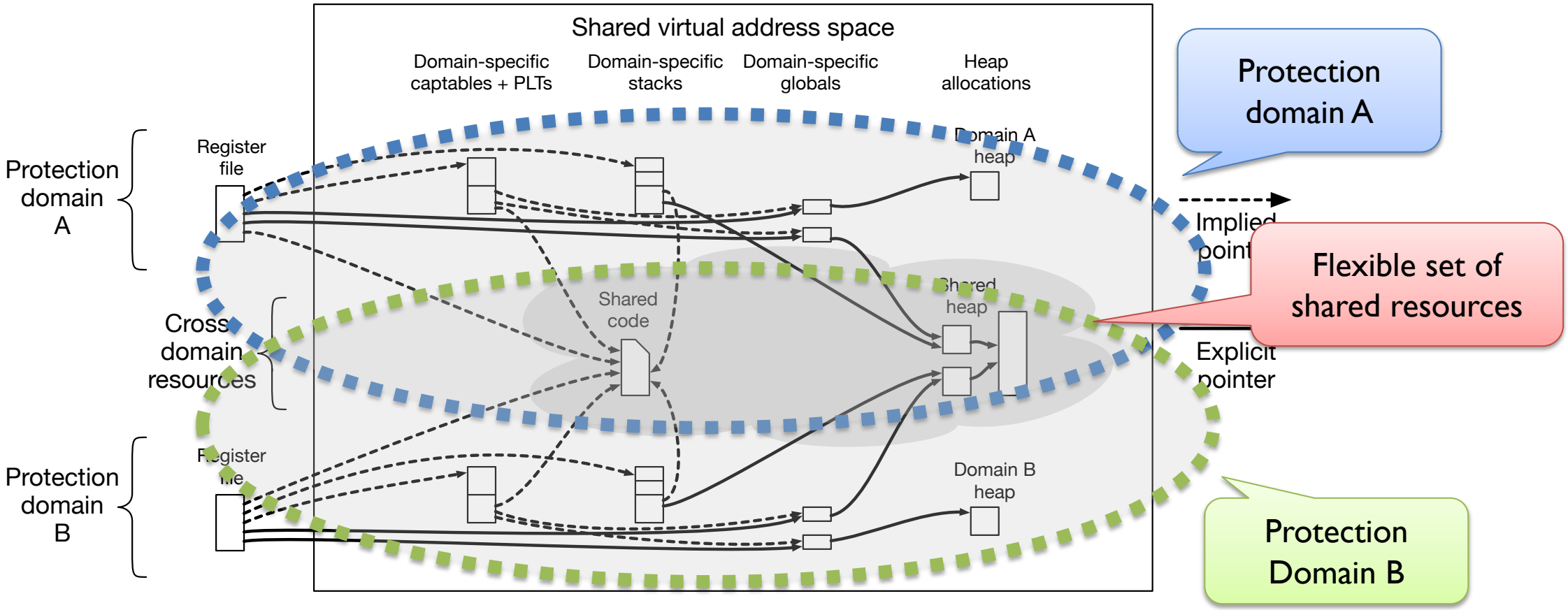
# What is software compartmentalization?




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

# CHERI-based compartmentalization




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

# Opportunities and challenges

- CHERI dramatically improves compartmentalization scalability
  - More compartments
  - More frequent domain transitions
  - Faster shared memory between compartments
- 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 also requires **careful software refactoring** to support strong encapsulation, and affects the software operational model

# 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.

**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.

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

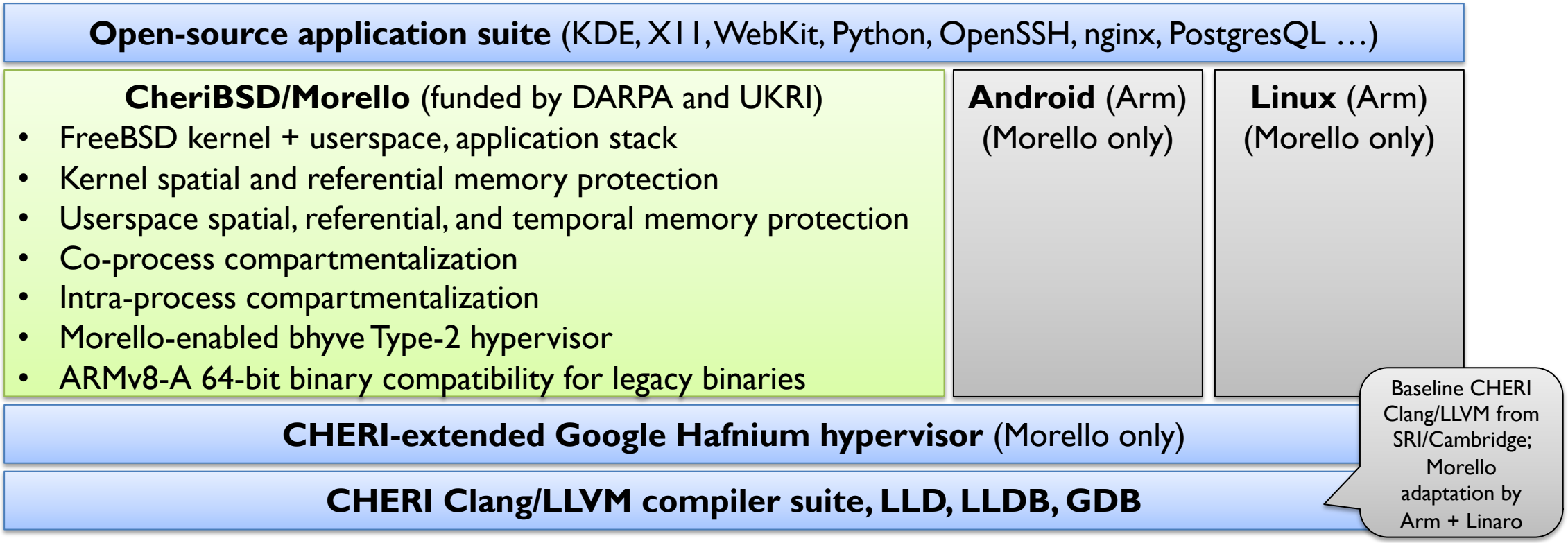
# CHERI REFERENCE SOFTWARE STACK

# Porting the CHERI software 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, …)
- Act as a template and prototyping platform for industrial demonstration (e.g., for Morello Consortium partners)
- Provide a platform for future research (e.g., 11 EPSRC projects at UK universities starting August-October 2020)

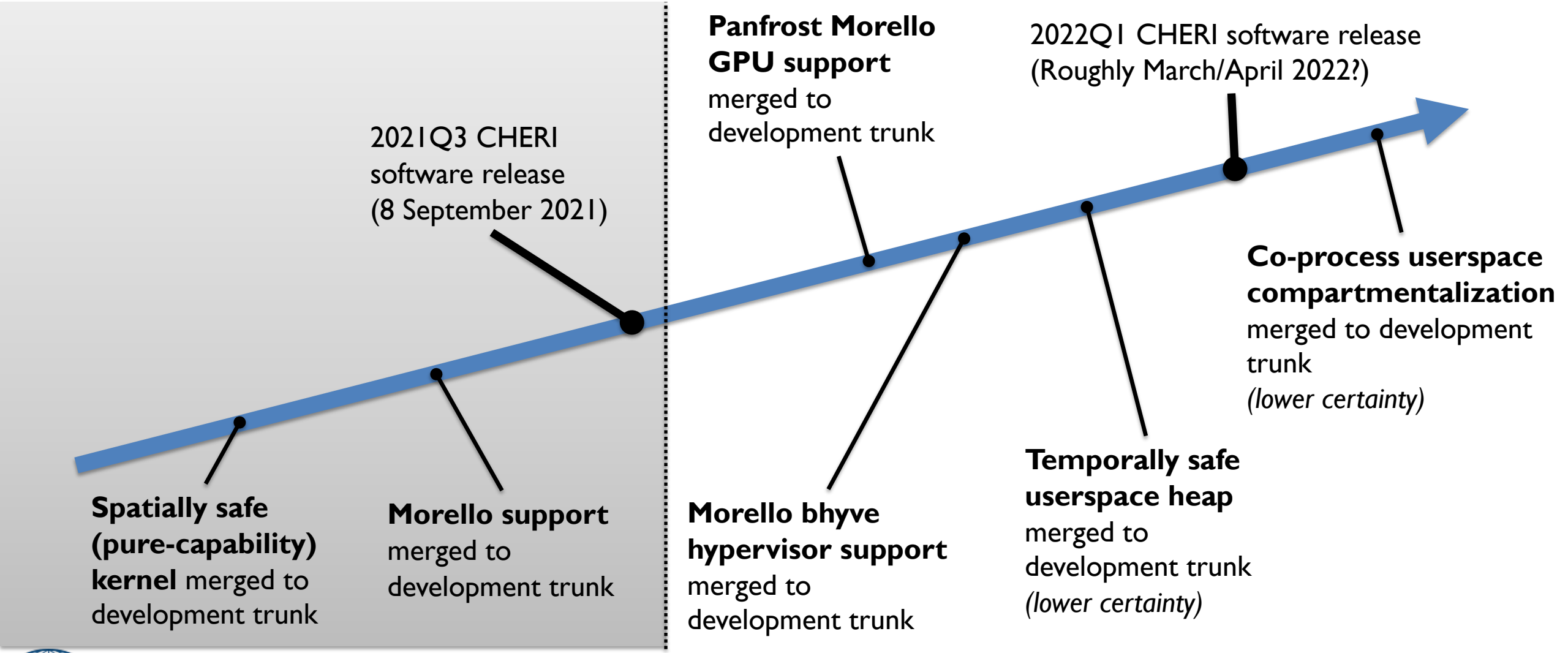
# CHERI prototype software stack on Morello

- **Complete open-source CHERI-enabled software stack** from bare metal up: compilers, toolchain, debuggers, operating systems, applications – all demonstrating CHERI ideas

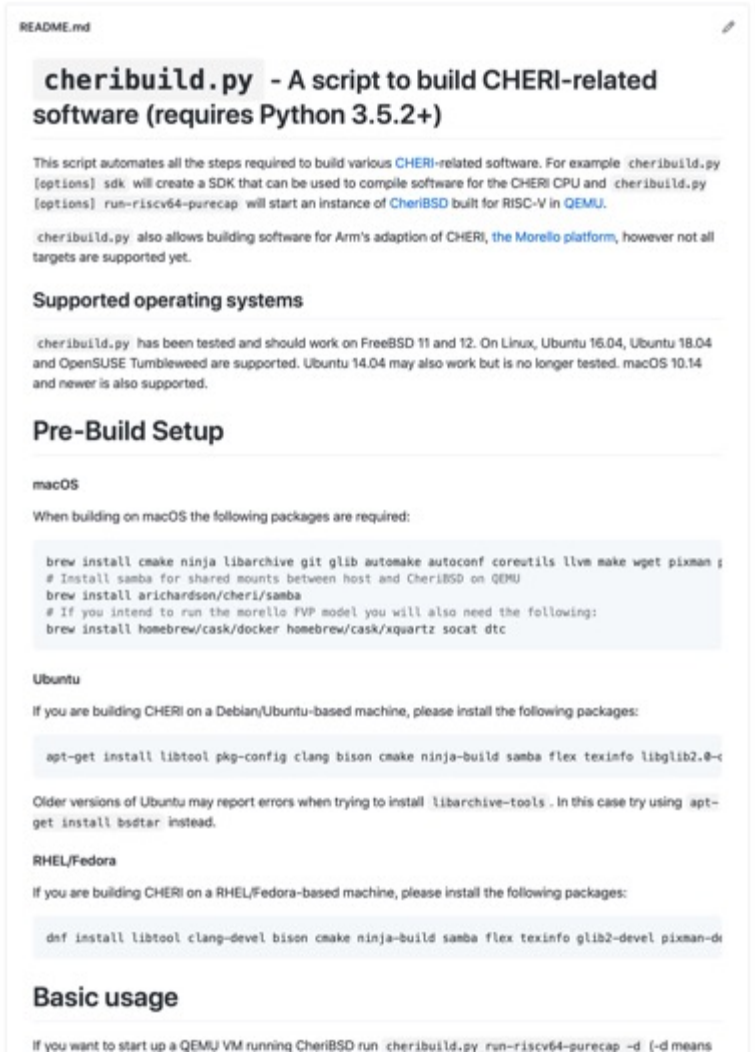



- Rich CHERI feature use, but fundamentally incremental/hybridized deployment
- Aim: Mature and highly useful research and development platform for Morello

# CHERI Reference Software Stack development plan (prospective)



# 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-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:
  - 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

# Getting support

- CHERI discussion mailing list (currently pretty quiet)
  - cl-cheri-discuss mailing list
  - cl-cheri-announce to be announced soon ☺

- **Slack: cheri-cpu.slack.com**

- Arm Morello support forum and mailing list on Morello-specific topics

# 3-month CHERI Desktop pilot study

Assess the viability of a CHERI/Morello-enabled open-source desktop software stack:

- **Select** sample open-source stack slice (window server, widget, window manager, application suite): X11, Qt, KDE, applications
- **Implement** CHERI C/C++ referential and spatial memory protection
- **Whiteboard** possible software compartmentalizations
- **Evaluate** software change as %LoC changed
- **Evaluate** security via retrospective vulnerability analysis (5 year sample)
- Improve CHERI compiler toolchain as needed

Detailed technical report published in mid-September 2021

# Results summary

- Adapted XVNC, X11 libraries, supporting libraries (e.g., libpng, …), Qt, KDE, selected KDE applications
  - **Roughly 6 million lines of C/C++ code** compiled for memory safety, with light dynamic testing
  - **Three compartmentalization case studies** in Qt/KDE
- Mitigation rates for selected software:
  - **91%** of X11 security advisories
  - **100%** of supporting library vulnerabilities (e.g., libpng, libxml2, …)
  - **82%** of Qt security advisories
  - **43%** of KDE security advisories
- Plenty of limitations discussed in detail in the report (e.g., language runtimes omitted)
- Lots of details in the technical report on CapLtd website - http://www.capabilitieslimited.co.uk/

# 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

    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. **CHERI C/C++ Programming Guide**, UCAM-CL-TR-947, June 2020.