CHERI
Capability Hardware Enhanced RISC Instructions

Robert N. M. Watson, Simon W. Moore, Peter Sewell, Peter G. Neumann
Hesham Almatary, Jonathan Anderson, John Baldwin, Hadrien Barrel, Ruslan Bukin, David Chisnall, James Clarke, Nirav Dave, Brooks Davis, Lawrence Esswood, Nathaniel W. Filardo, Khilan Gudka, 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, Alex Richardson, Michael Roe, Colin Rothwell, Peter Rugg, Hassen Saidi, Peter Sewell, Stacey Son, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Jonathan Woodruff, Hongyan Xia, and Bjoern A. Zeeb

University of Cambridge and SRI International
InnovateUK Digital Security by Design Challenge – Collaborators' Workshop – 26 September 2019

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.

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

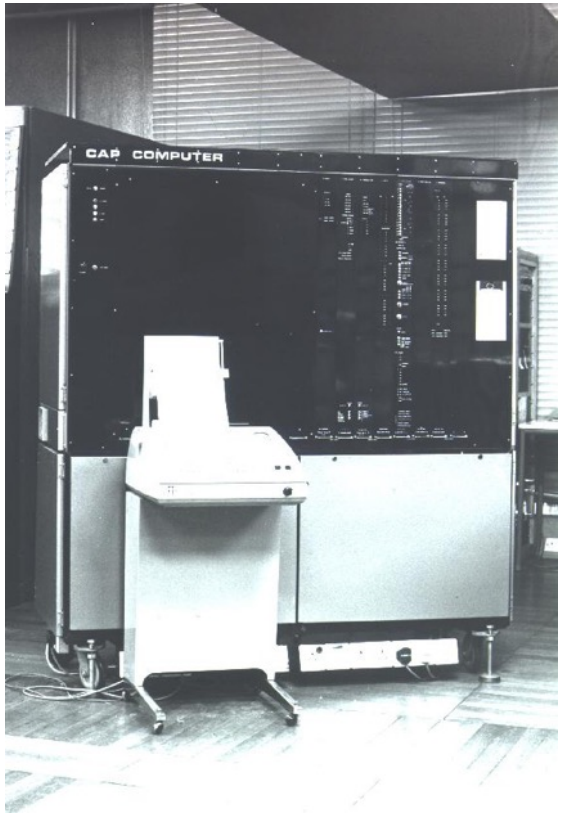
- A brief introduction to capabilities and the CHERI architecture
- What CHERI brings to the Digital Security by Design Challenge

- To learn more about the CHERI architecture and prototypes:

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

- Watson, et al. **An Introduction to CHERI**, Technical Report UCAM-CL-TR-941, Computer Laboratory, September 2019. (~40 pages)
- Watson, et al. **Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 7)**, Technical Report UCAM-CL-TR-927, Computer Laboratory, June 2019. (~500 pages)

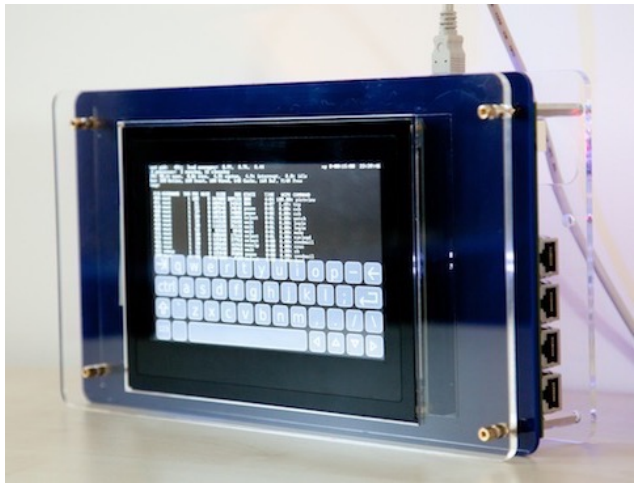
Capability systems



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
  - **Capability-based systems** are those in which 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

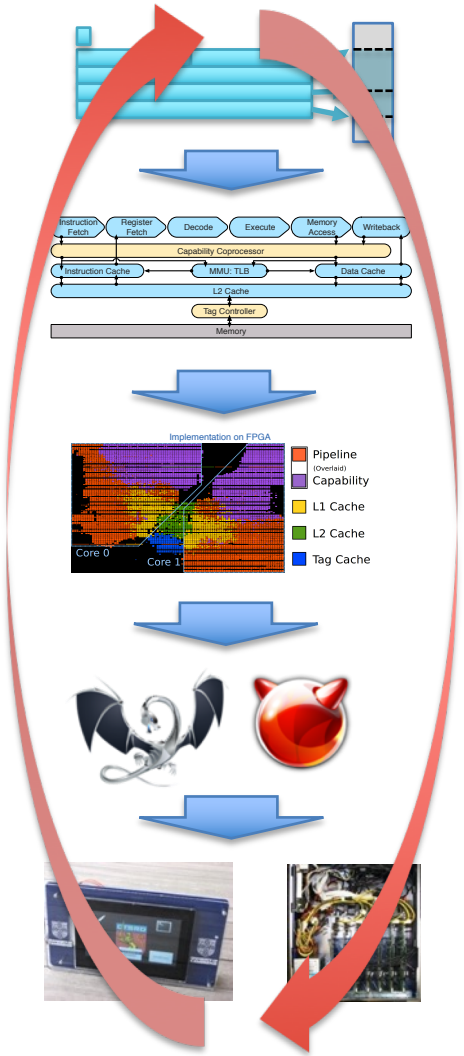
What is CHERI?



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

- CHERI is an **architectural protection model**
  - Composes the 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, scalable compartmentalization
  - Directly impedes common exploit-chain tools used by attackers
  - Mitigates many vulnerability classes .. even unknown future classes

Hardware-software-semantics co-design

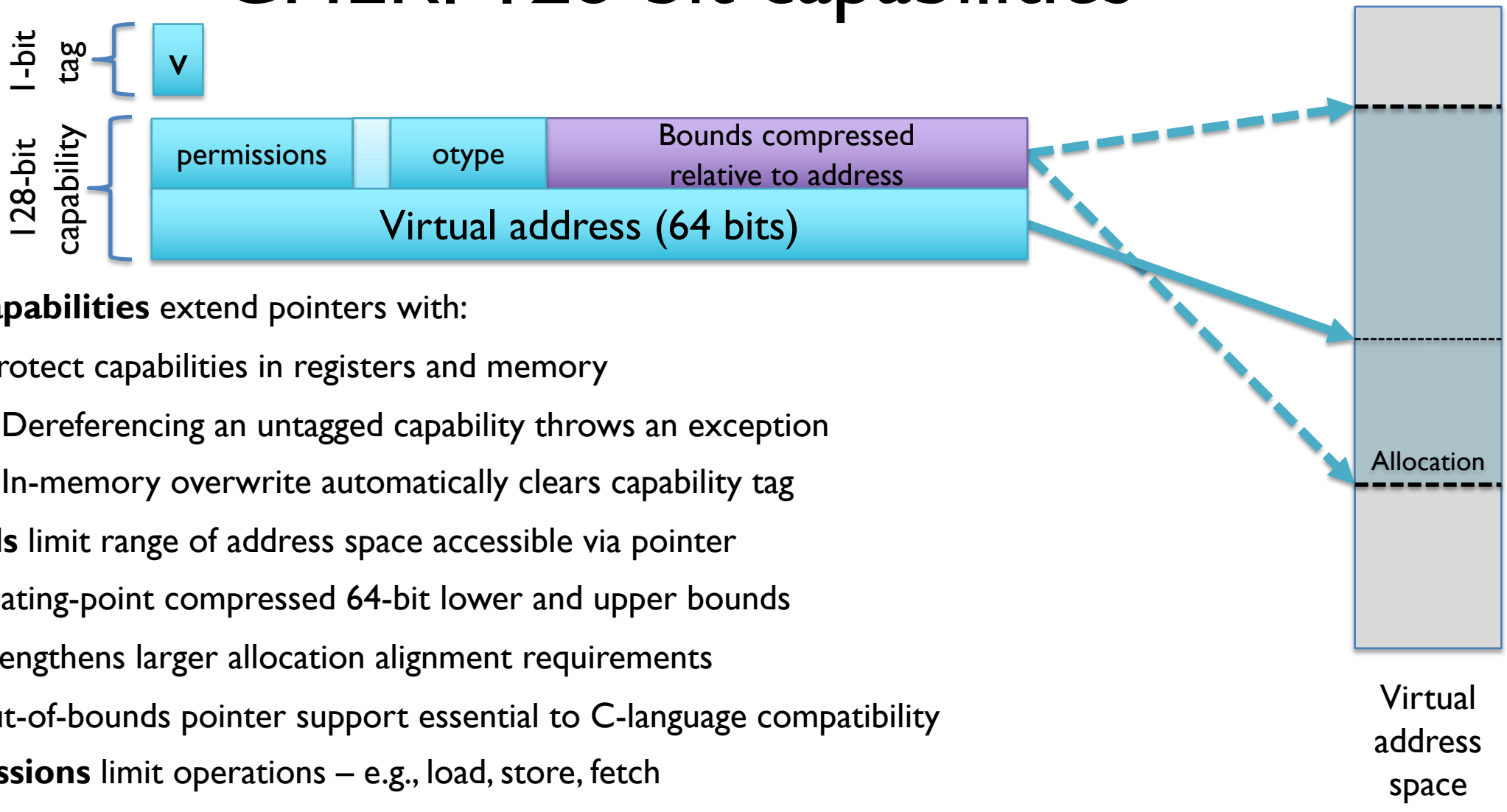


- Architectural mitigation for C/C++ TCB vulnerabilities
  - Tagged memory, new hardware capability data type
  - Model hybridizes cleanly with contemporary RISC ISAs, CPU designs, MMU-based OSes, and C/C++-language software
  - New hardware enables incremental software deployment
- Hardware-software-semantics co-design + concrete prototyping:
  - CHERI abstract protection model; concrete ISA instantiations in 64-bit MIPS, 32/64-bit RISC-V, 64-bit ARMv8-A
  - 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 design goals and approach

- **De-conflate memory virtualization and protection**
  - Memory Management Units (MMUs) protect by **location (address)**
  - CHERI protect 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, …)

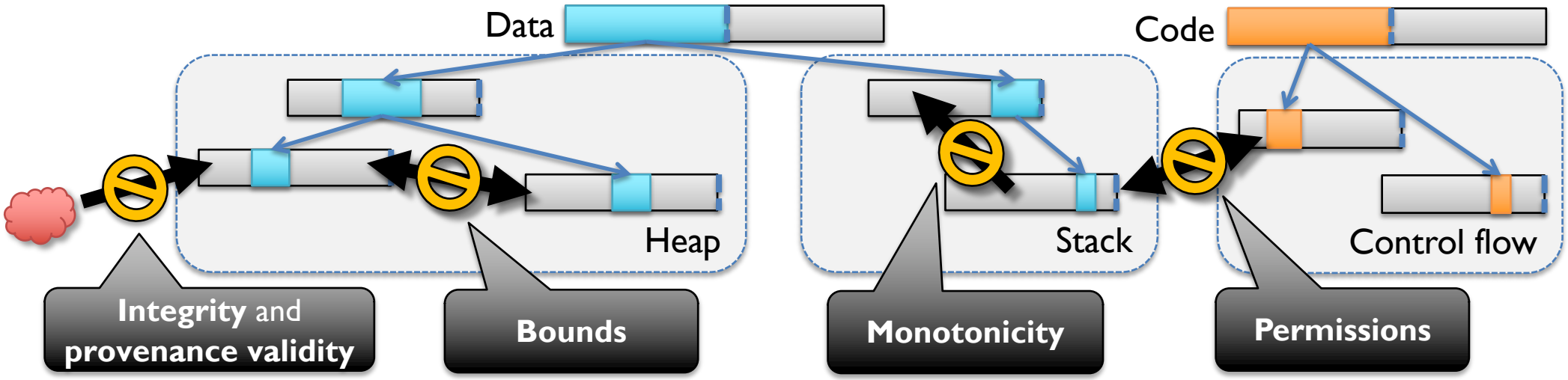
CHERI 128-bit capabilities



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** for **encapsulation**: **immutable**, **non-dereferenceable**

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**
  - E.g., Received network data cannot be interpreted as a code or data pointer
- **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 memory protection**, but also higher-level policies such as **scalable software compartmentalization**

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?

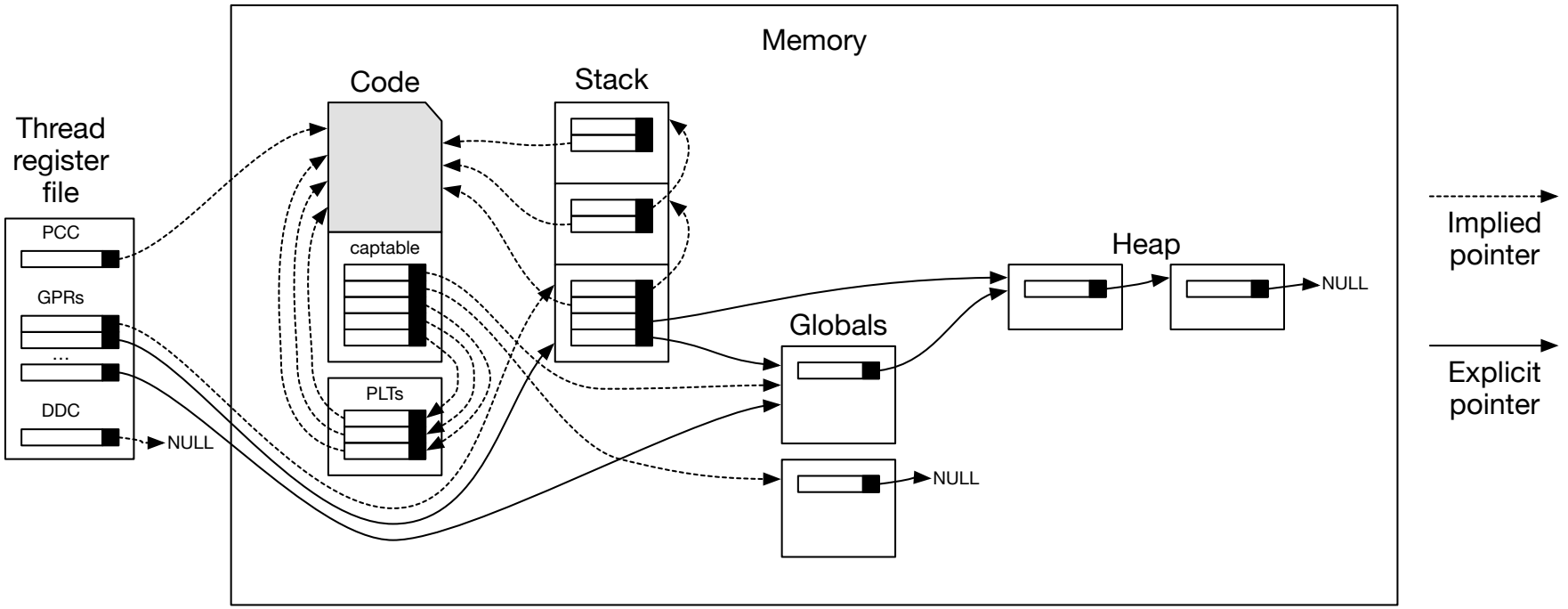
  Monotonicity: Q: Can real software play by these rules?

- Scalable fine-grained **software compartmentalization**

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

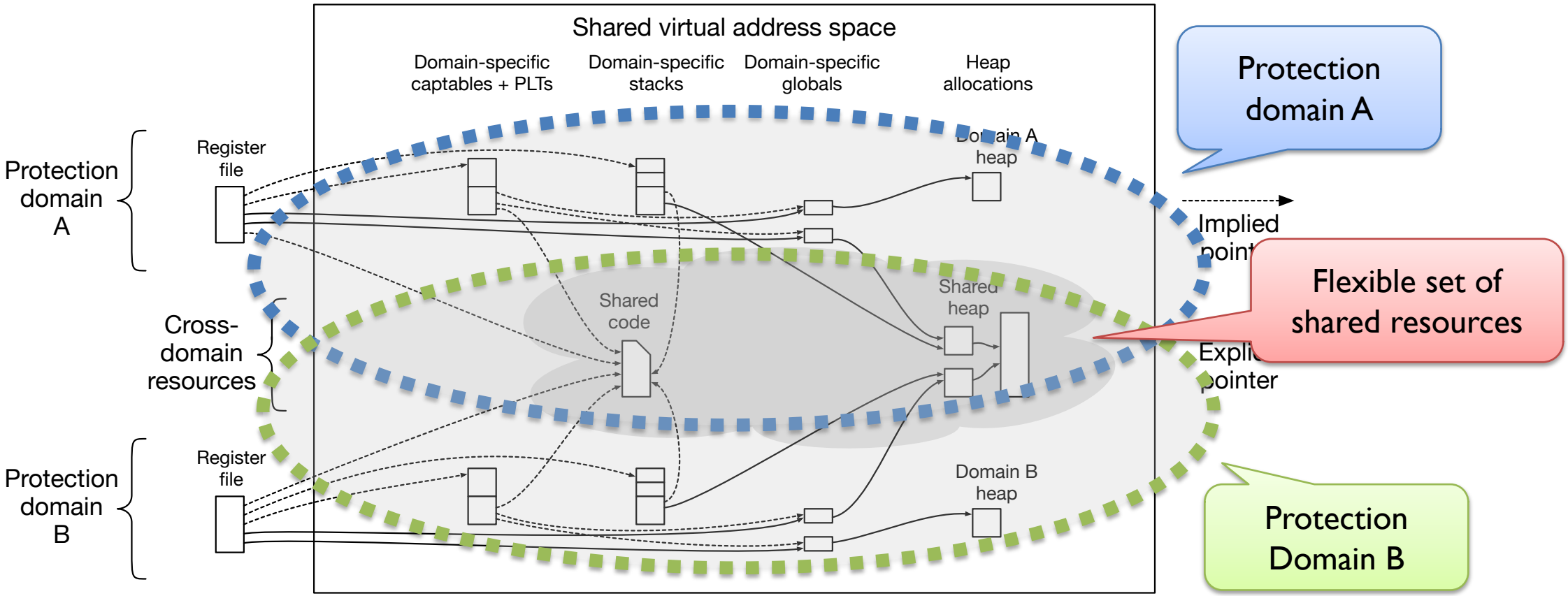
  Q: Can **sealed capabilities**, **controlled non-monotonicity**, and **capability-based sharing** enable safe, efficient compartmentalization?

CHERI-based 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-based compartmentalization

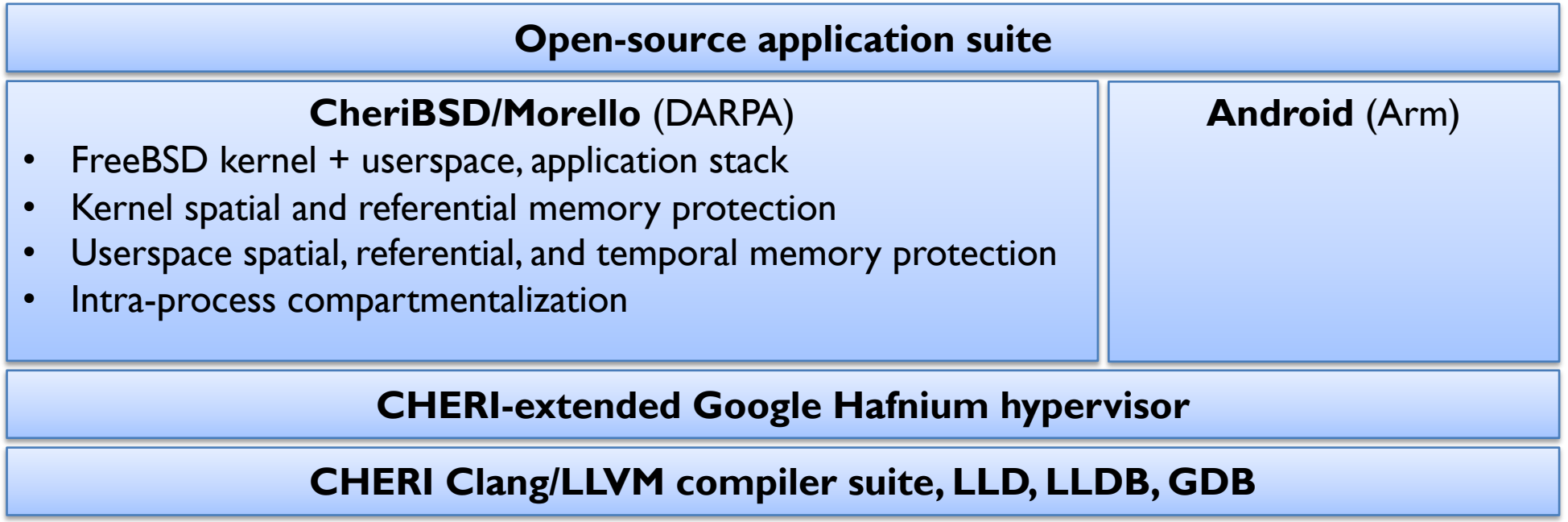


- Isolated compartments can be created using closed graphs of capabilities, combined with a constrained non-monotonic domain-transition mechanism

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 experimentation: large-scale application experiments, operating-system use, debuggers, …
  - Extend work in formal modeling and security 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

DARPA software prototype stack



- Complete hybrid software stack from bare metal up: compilers, toolchain, debuggers, operating systems, applications
- Focused on deploying CHERI incrementally, rather than clean-slate

Potential areas for CHERI research (.. and there are many others as well ..)

- Quantitative ISA optimization
- Compiler semantics and optimization
- Superscalar microarchitectures
- Tag tables vs. native DRAM tags
- Toolchain: linker, debugger, …
- C++ compilation to CHERI
- Growing the software corpus
- CHERI and ISO C/POSIX APIs
- Compartmentalization frameworks
- MMU-free CHERI microkernel
- Safe Foreign Function Interfaces (FFIs)
- Safe inter-language interoperability
- C-language temporal memory safety
- Integration with managed languages
- Formal proofs of ISA properties
- Formal proofs of software properties
- Verified hardware implementations
- Use with large or non-volatile memory
- Security analysis and red teaming
- Microarchitectural optimization opportunities from exposed software semantics
- MMU-free HW designs for IoT

Invitation to collaborate

- It is an exciting moment for CHERI
  - Finally able to talk about a significant industrial collaboration
  - Experimental adaptations to mainstream architecture(s)
  - Rich and maturing software baselines for experimentation
  - Strong formal foundations available to be built on
  - New research funding program create opportunity to broaden collaborations and bring new ideas and expertise to CHERI
  - EPSRC and ESRC calls around the CHERI technology and possible deployment
- Many new things to learn – we've done a lot with CHERI, but the more we do, the more we learn remains to be done!

www.cheri-cpu.org