Introducing CHERI-RISC-V

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 and SRI International

RISC-V Week

Paris, 3-5 May 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.

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: What is CHERI?

- **CHERI=Capability Hardware Enhanced RISC Instructions**
- **CHERI is a new hardware technology that mitigates software security vulnerabilities**
  - Developed by the University of Cambridge and SRI International starting in 2010, supported by DARPA and others
  - Arm collaboration from 2014
- CHERI for RISC-V is now mature, but being refined
- **Today's talk:**
  - Why develop CHERI?
  - What is CHERI and how does it work?
  - What software will I be able to run on it?
  - What sort of evaluations have been run to date?

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

SRI International

UNIVERSITY OF CAMBRIDGE

Why develop CHERI?

"Buffer overflows have not objectively gone down in the last 40 years.

The impact of buffer overflows have if anything gone up."

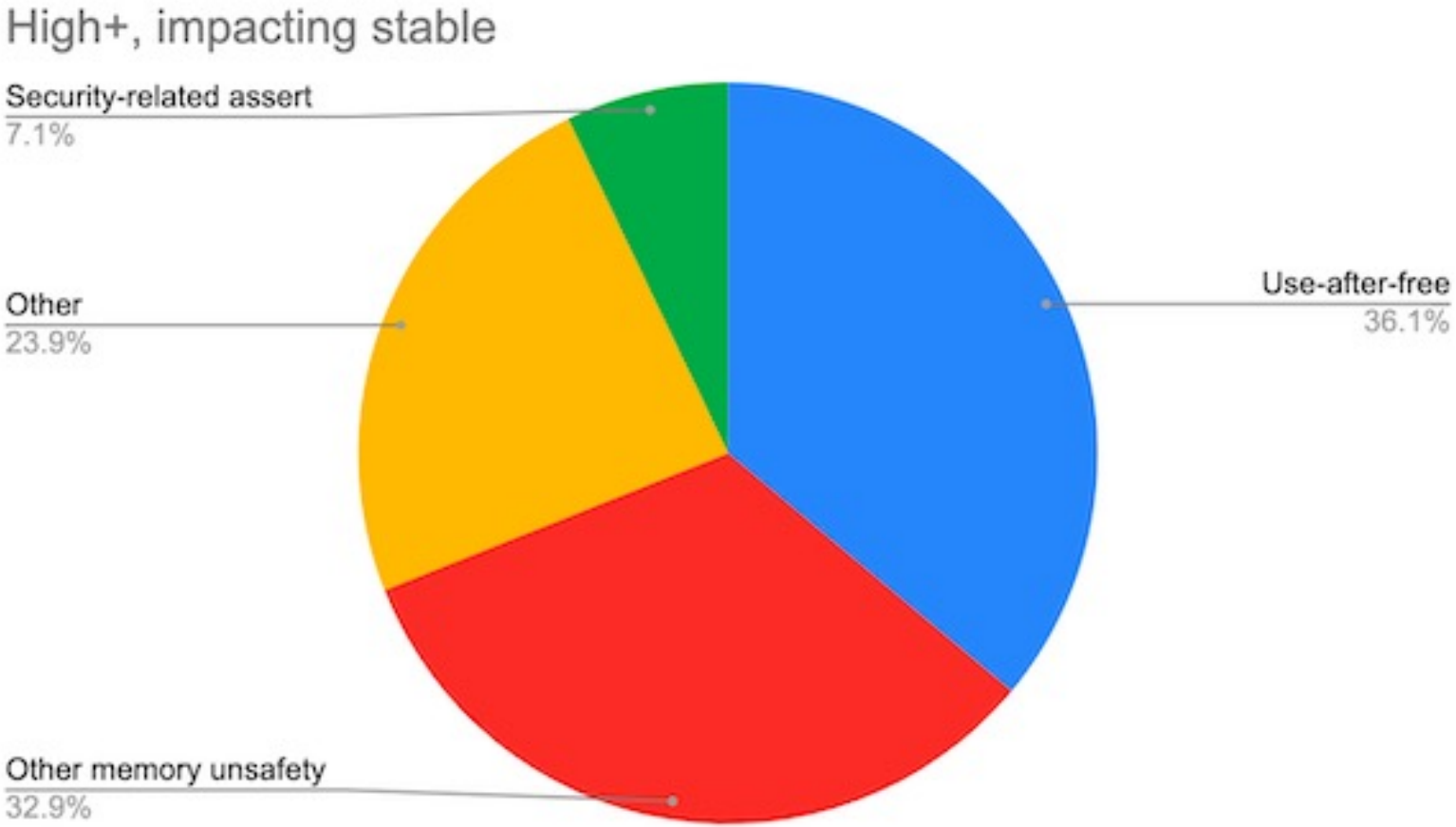
Ian Levy, NCSC

- Matt Miller (MS Response Center) @ BlueHat 2019:
  - From 2006 to 2018, year after year, 70% MSFT CVEs are memory safety bugs.
  - First place: spatial safety
    - Addressed directly by CHERI
  - Second place: use after free
    - Our recent work exploiting CHERI capability validity tags to precisely find pointers

More Motivation – Chromium Browser Safety

"70% of our serious security bugs are memory safety problems"

www.chromium.org/Home/chromium-security/memory-safety

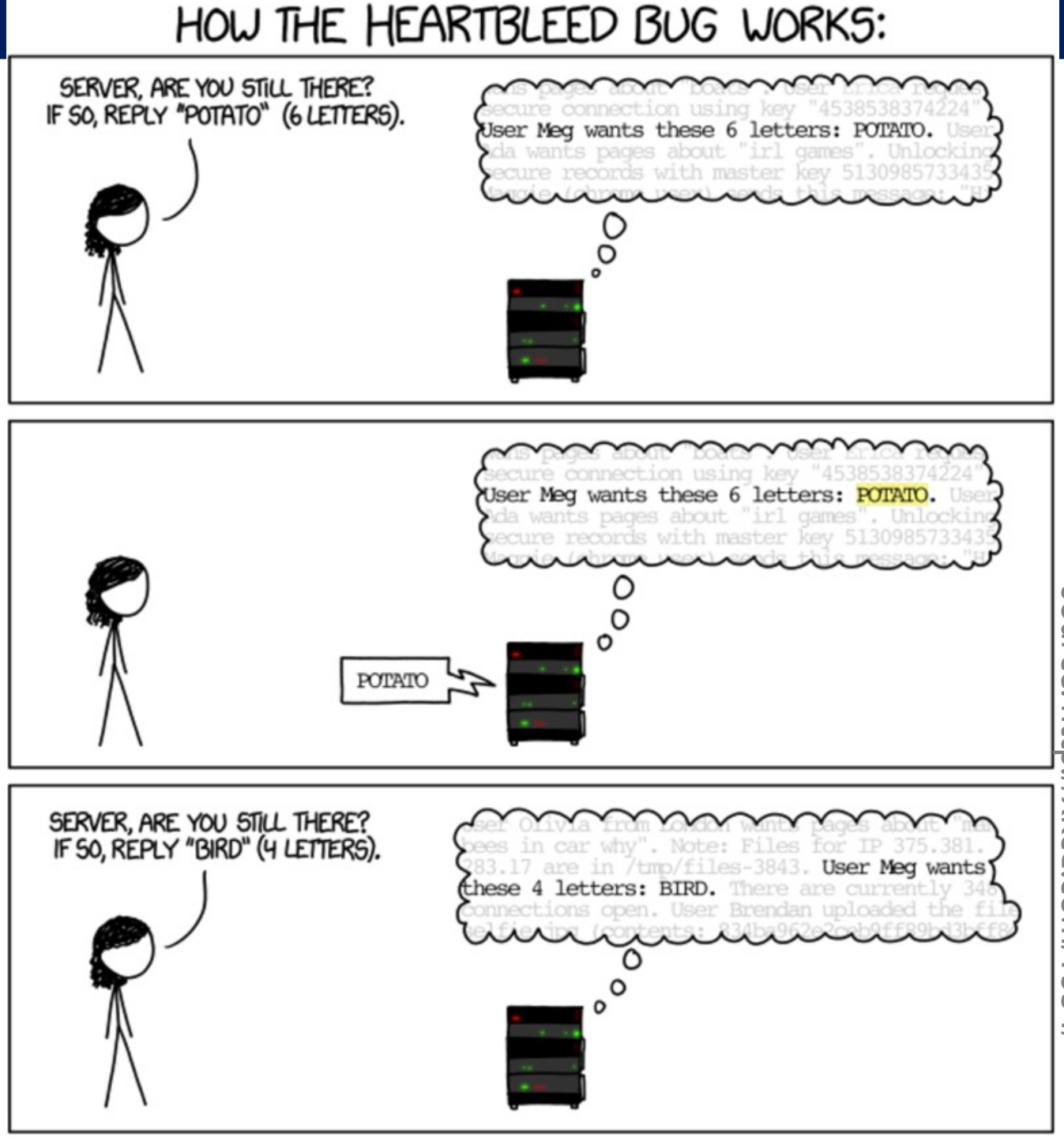


SRI International

UNIVERSITY OF CAMBRIDGE

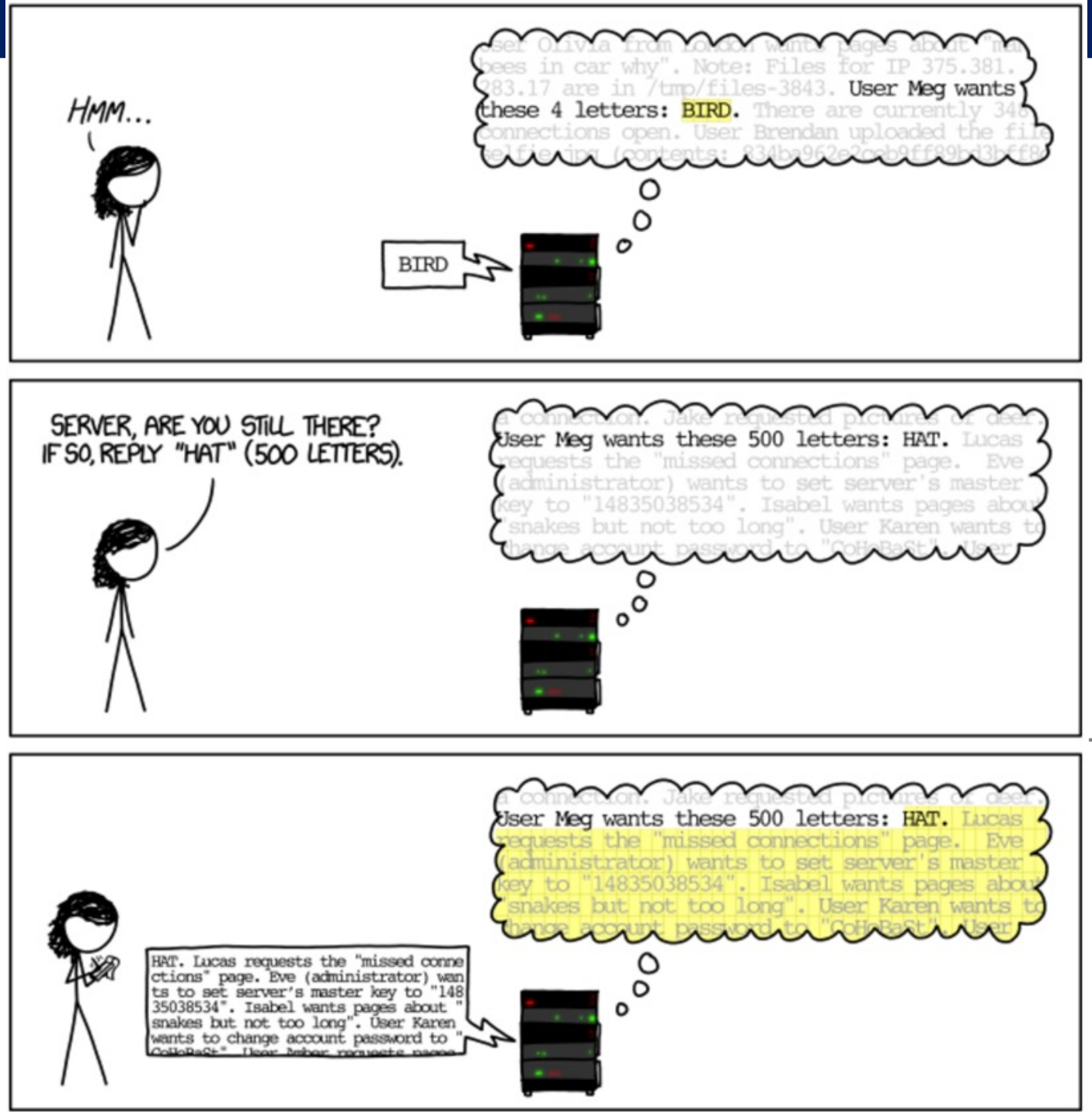
Example 1

HeartBleed



source: http://xkcd.com/1354/

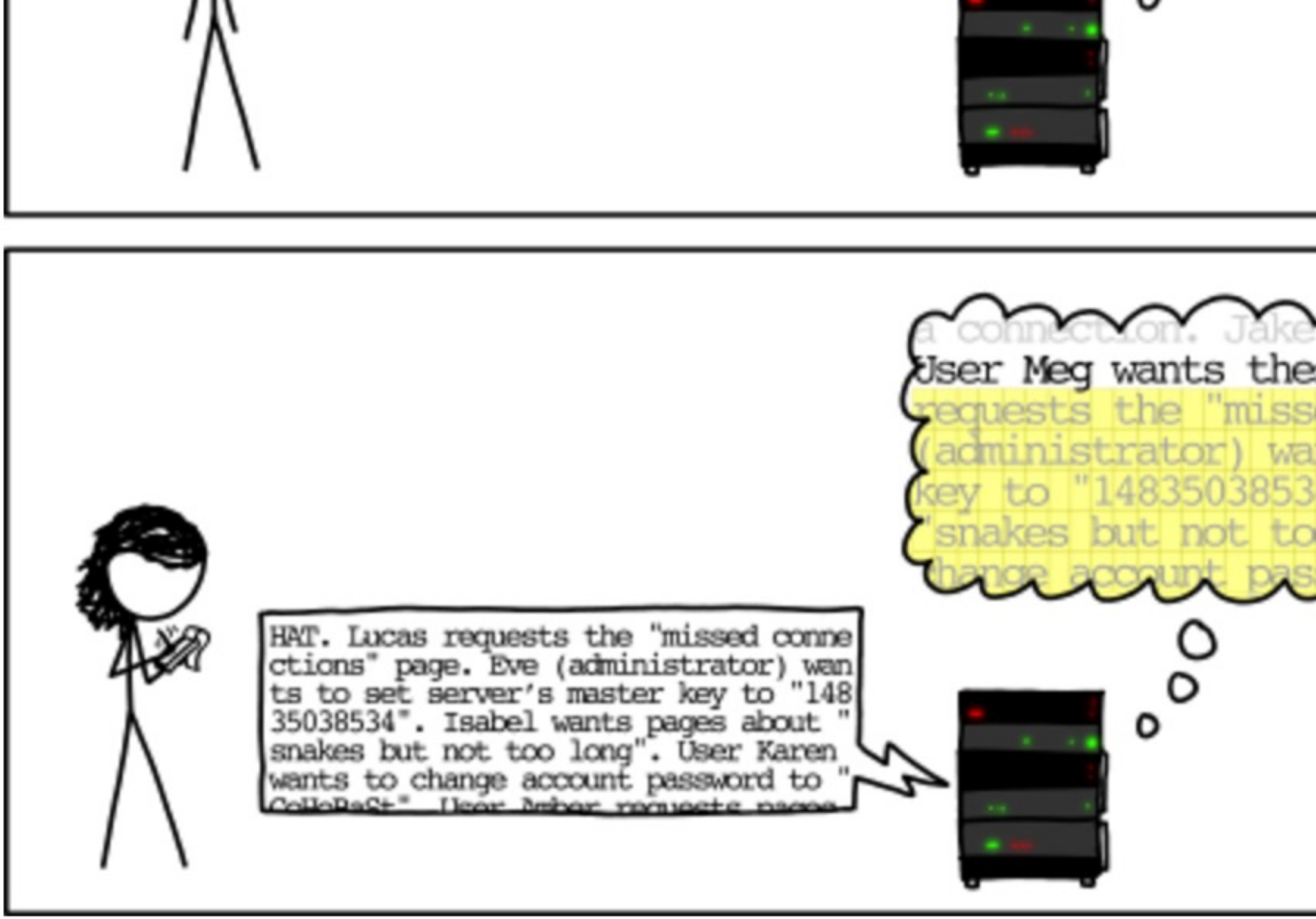
HeartBleed



source: http://xkcd.com/1354/

SRI International

UNIVERSITY OF CAMBRIDGE


HAT. Lucas requests the "missed conne
ctions" page. Eve (administrator) wan
ts to set server's master key to "148
35038534". Isabel wants pages about "
snakes but not too long". User Karen
wants to change account password to "

Went wrong? How do we do better?

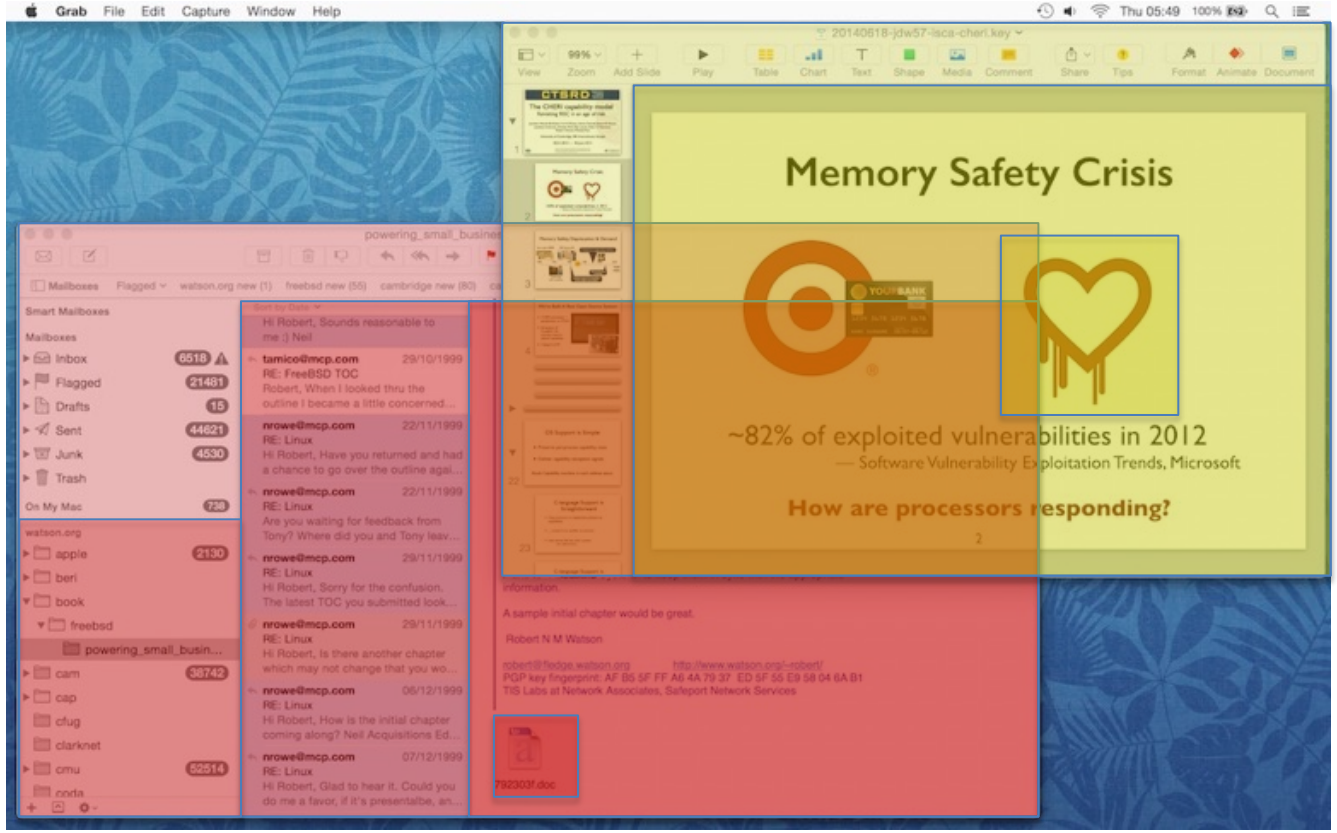
- Classical answer:
  - The programmer forgot to check the bounds of the data structure being read
  - Fix the vulnerability in hindsight – one-line fix:
    if (1+2+payload+16 **>** s->s3->rrec.length) **return** 0;
- Our answer:
  - Preserve bounds information during compilation
  - Use hardware (CHERI processor) to dynamically check bounds with little overhead and guarantee pointer integrity & provenance

Example 2: how to reduce the attack surface?

- The software attack surface keeps getting bigger
  - Applications just keep getting larger
  - Huge libraries of code aid rapid program development
  - Everything is network connected
- This aids the attacker: an expanding number of ways to break in

CHERI solution: application-level least privilege

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



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

SRI International

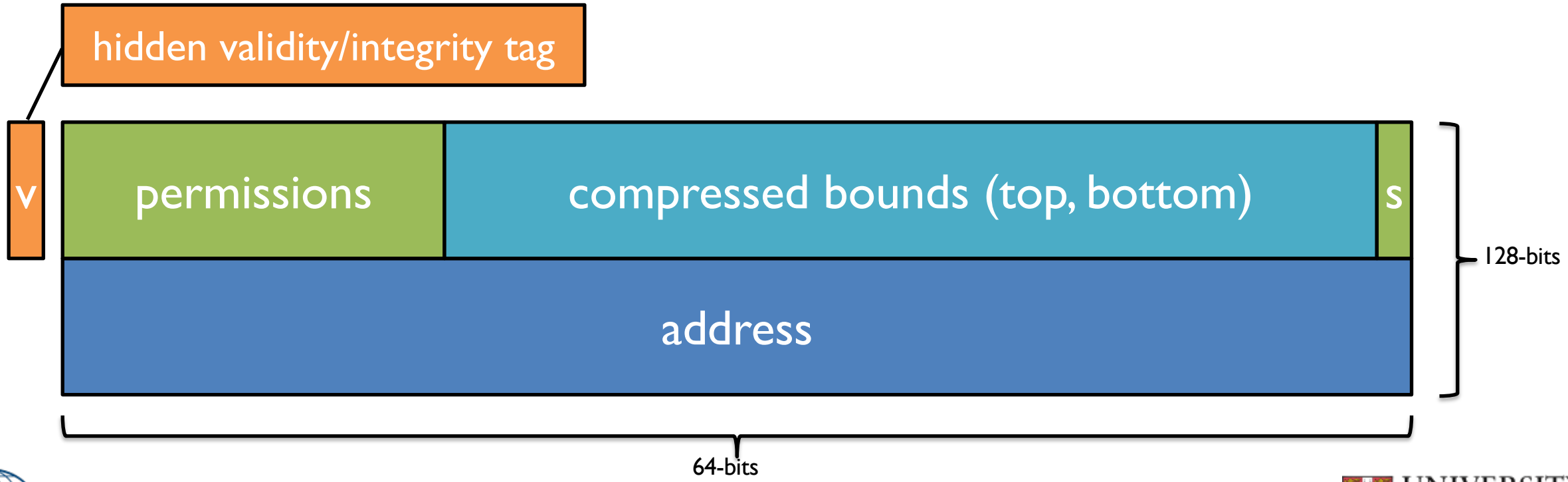
UNIVERSITY OF CAMBRIDGE

Principles CHERI helps to uphold

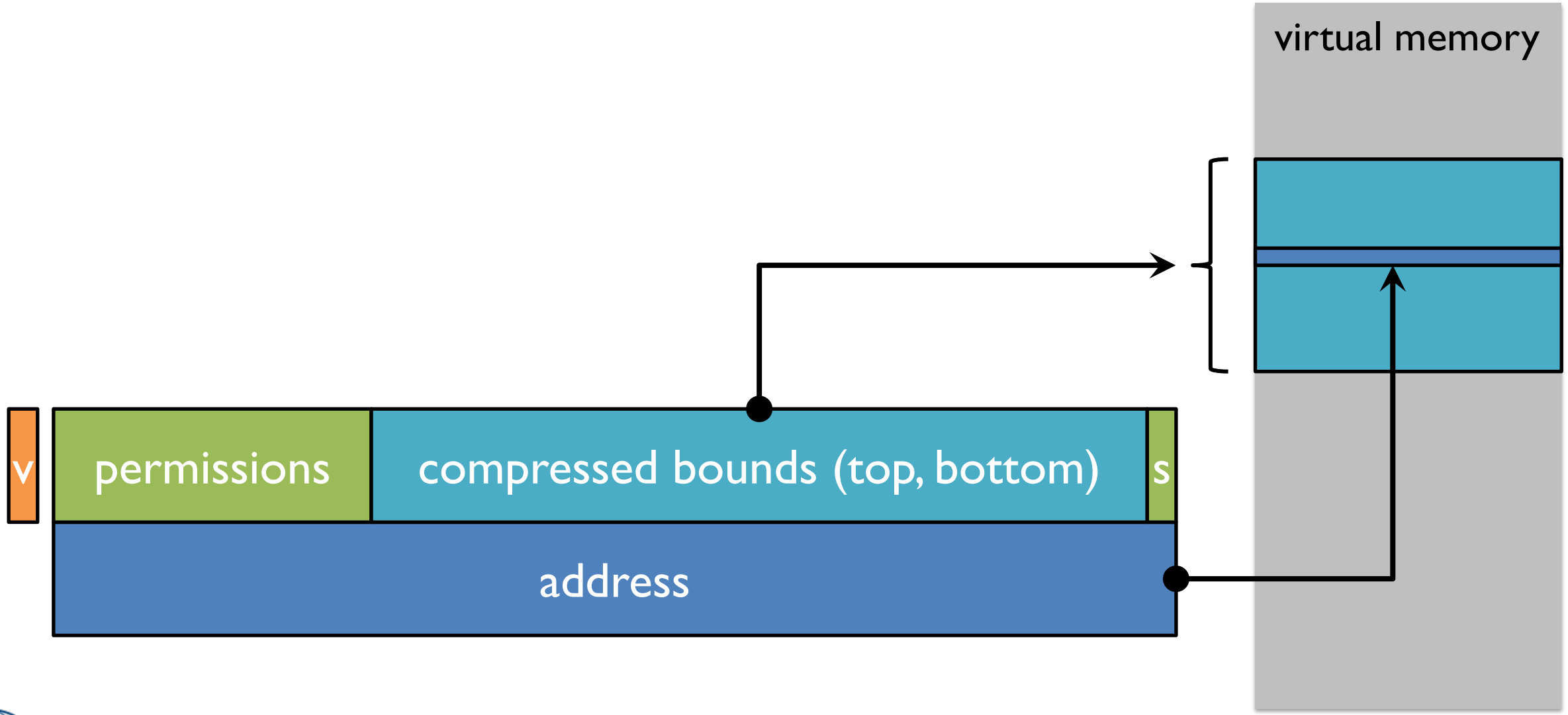
- The **principle of intentional use**
  - Ensure that software runs the way the programmer intended, not the way the attacker tricked it
  - Approach: guaranteed pointer integrity & provenance, with efficient dynamic bounds checking
- The **principle of least privilege**
  - Reduce the attack surface using software compartmentalization
  - Mitigates known and unknown exploits
  - Approach: highly scalable and efficient compartmentalization

CHERI hardware adds a new type – the **Capability**

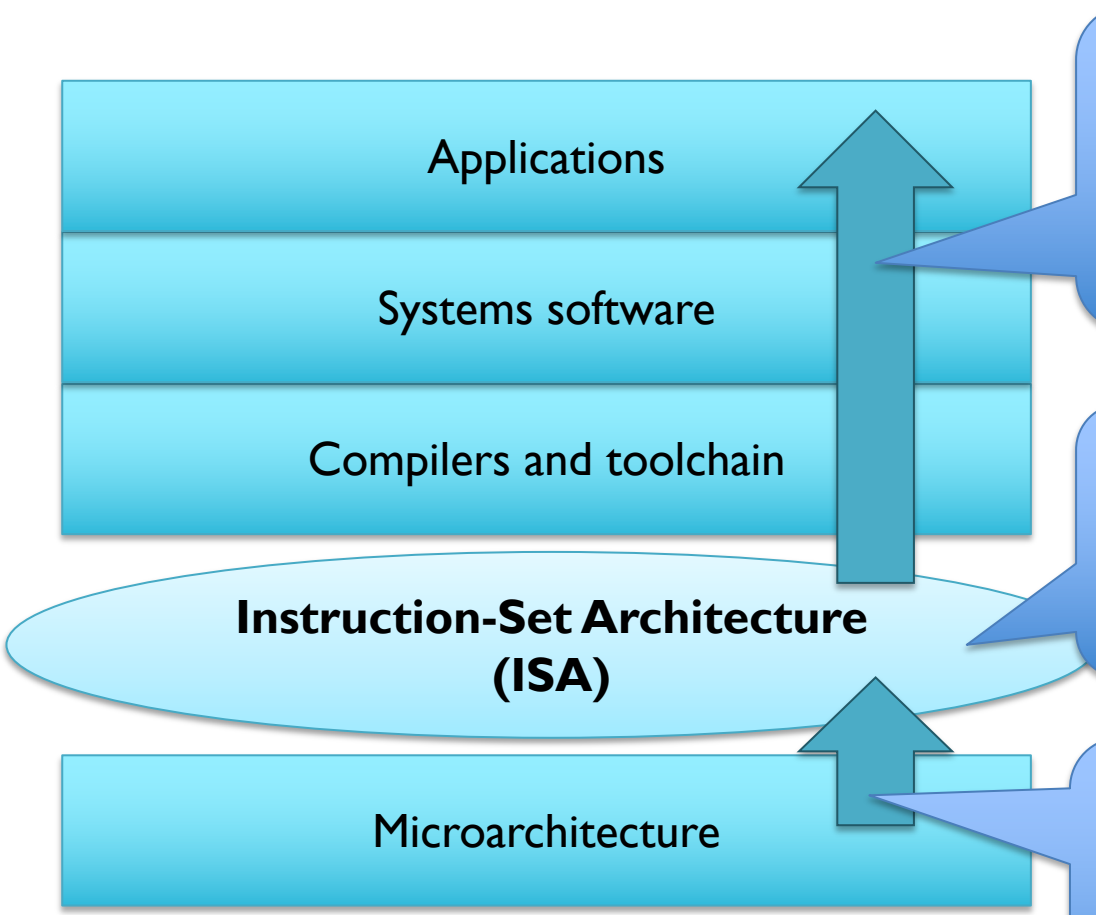
- CHERI Capability = bounds checked pointer with integrity
- Held in memory and in (new or extended) registers



A new type – the **Capability**



Processor 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

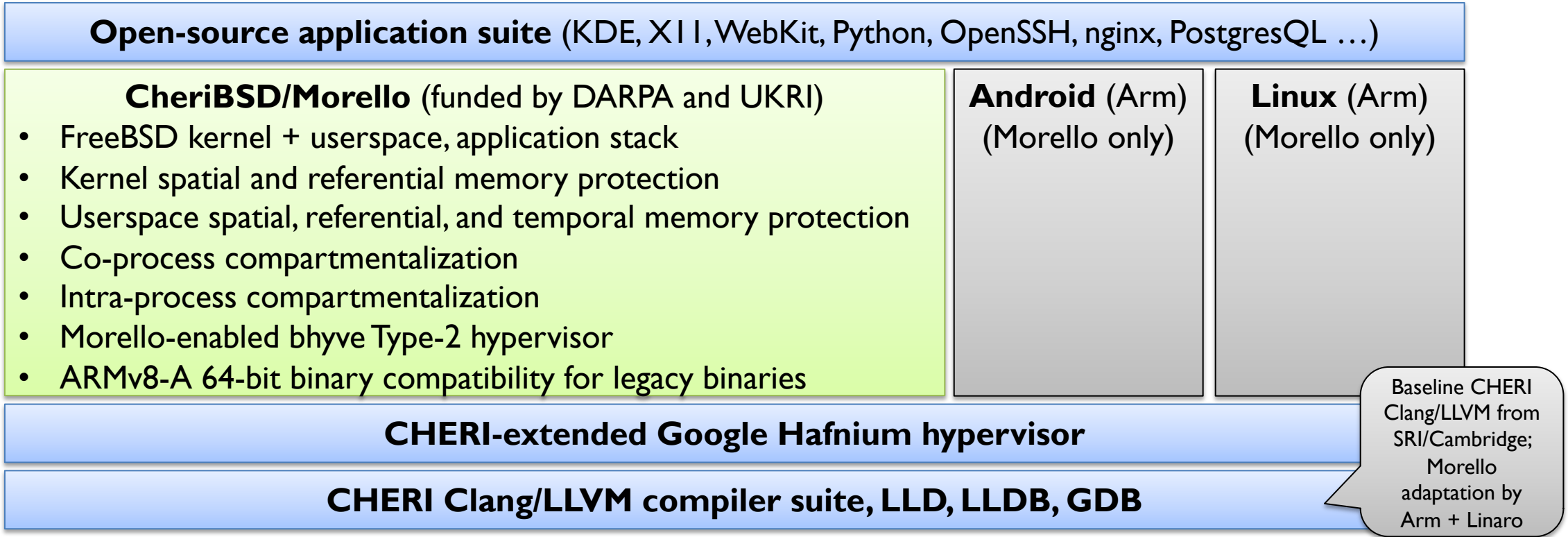
The microarchitecture implements the **capability data type** and **tagged memory**, enforcing invariants on their manipulation and use such as **capability bounds**, **monotonicity**, and **provenance validity**

Two key applications of the CHERI primitives

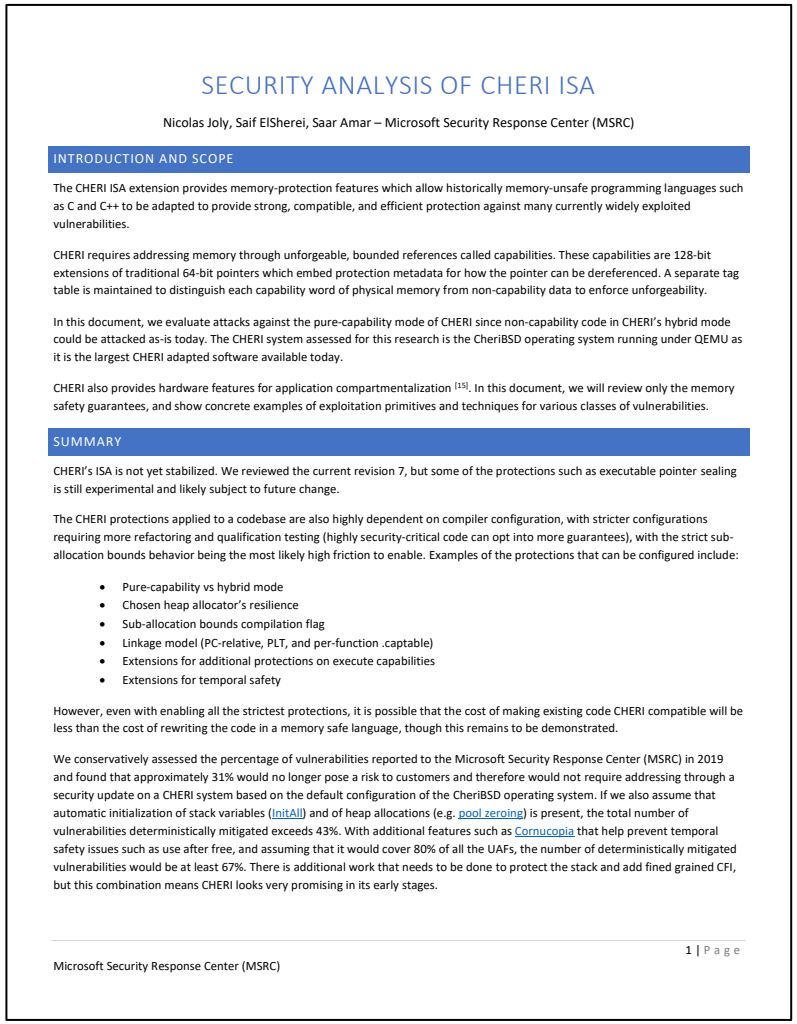
1. **Efficient, fine-grained memory protection for C/C++**
   - Strong source-level compatibility, but requires recompilation
   - Deterministic and secret-free referential, spatial, and temporal memory safety
   - Retrospective studies estimate ⅔ of memory-safety vulnerabilities mitigated
   - Generally 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)

CHERI 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
  - Aim: Mature and highly useful research and development platform for 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)

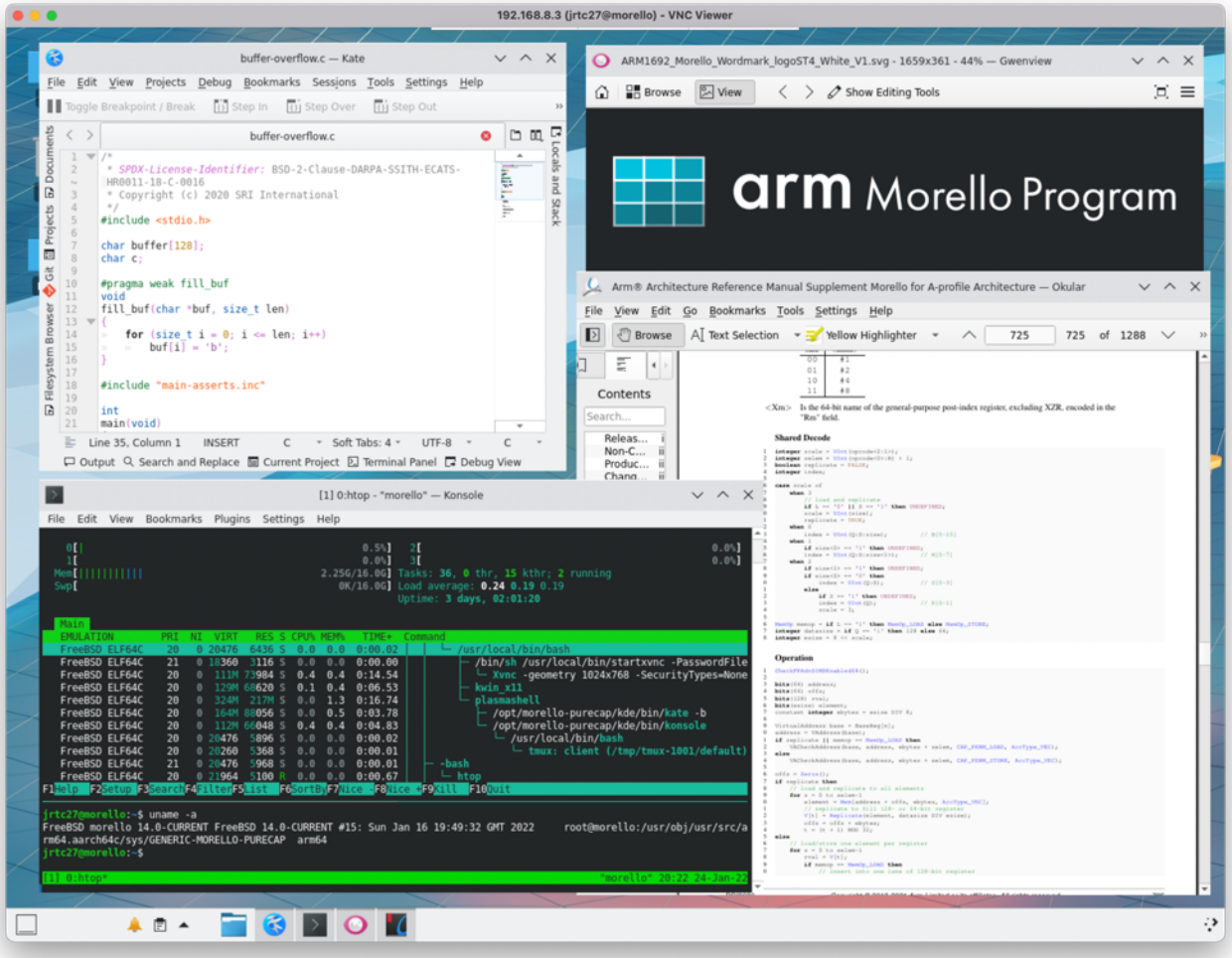
- Microsoft Security Research Center (MSRC) study analyzed all 2019 Microsoft critical memory-safety security vulnerabilities
  - Metric: "Poses a risk to customers → requires a software update"
  - Vulnerability mitigated if **no security update required**
- Blog post and 42-page report
  - Concrete 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/

SRI International

UNIVERSITY OF CAMBRIDGE

CHERI desktop ecosystem study: Key outcomes



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

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

SRI International

UNIVERSITY OF CAMBRIDGE

Where to learn more?

An Introduction to CHERI

- 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

- **Project web pages:**
  - http://www.cheri-cpu.org/
- **An Introduction to CHERI**, Technical Report UCAM-CL-TR-941, Computer Laboratory, September 2019
- **Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 8)**, UCAM-CL-TR-951, October 2020
- **CHERI C/C++ Programming Guide**, UCAM-CL-TR-947, June 2020

CHERI research and development timeline

Over 150 researcher years of effort by Cambridge & SRI
Many engineer years by Arm

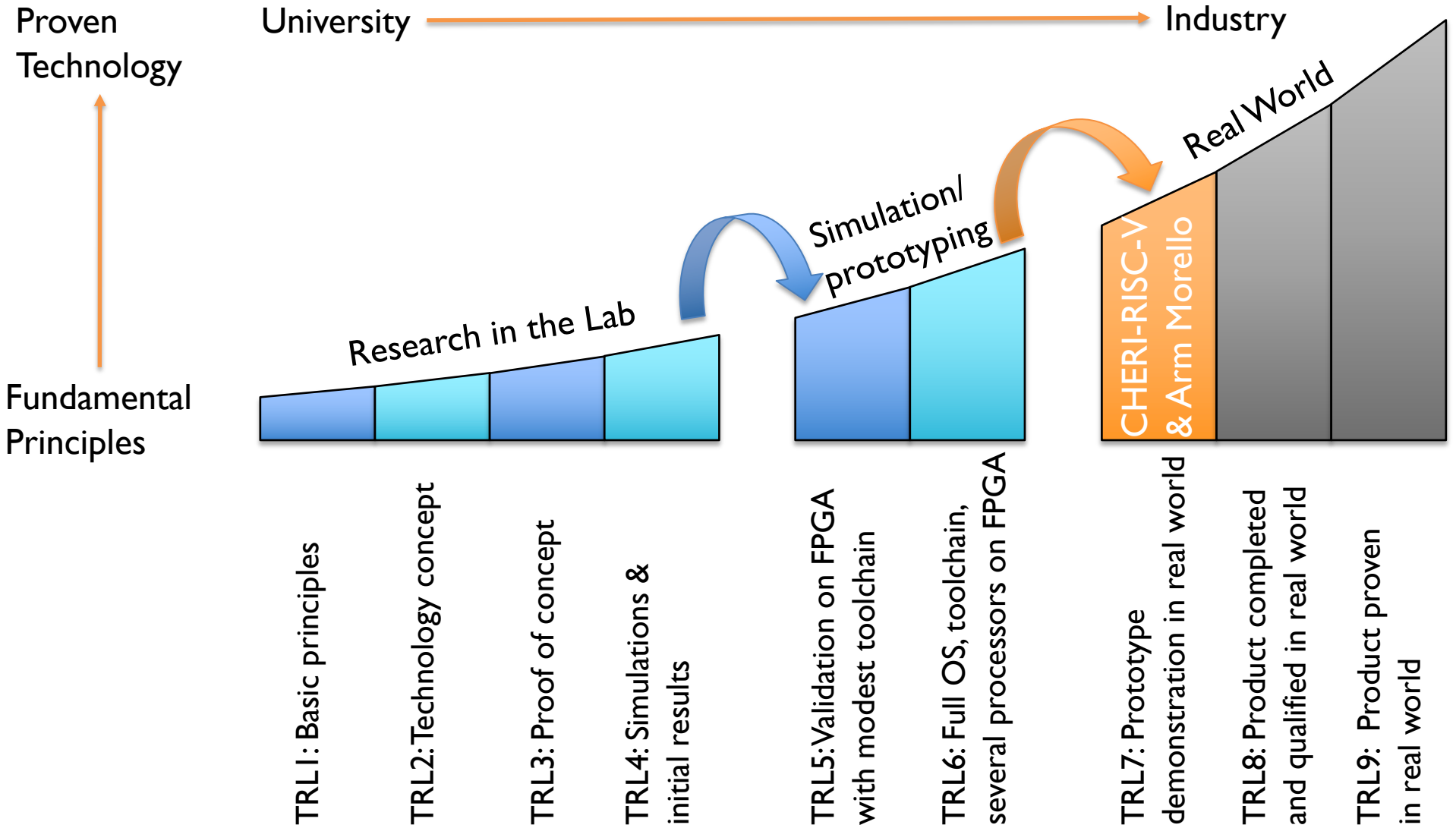
Years 1-2: Research platform, protot ec

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

Years 2-4: Hybrid C/OS model, comp ent model

Years 8-9: RISC-V, temporal safety, formal proof

Bridging the commercialisation chasm



First we made an FPGA-based hardware tablet

Open Source Stack: Research and Deployment

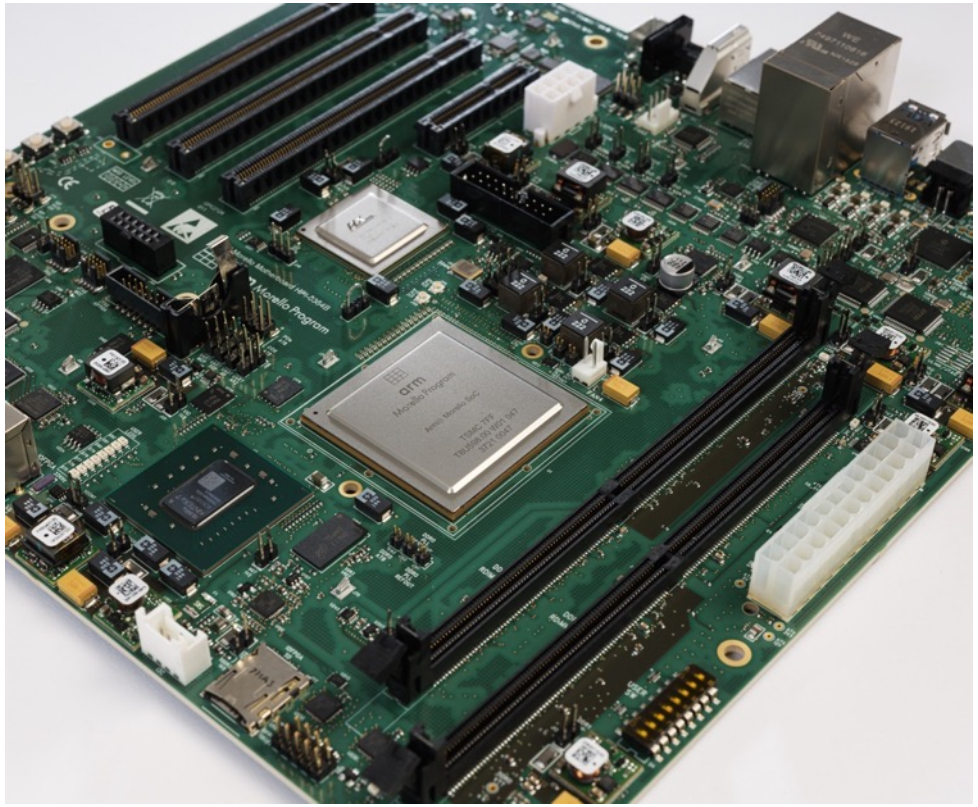
- CHERI-RISC-V developed open source:
  - Documentation (ISA ref, architecture overview, etc)
  - Specification in Sail
  - Simulators: Spike, Qemu
  - Clang/LLVM toolchain
  - OS support: CheriBSD, CheriFreeRTOS, CheriRTEMS
  - Hardware implementations
    - 3-stage, 5-stage and OoO cores on FPGA including AWS F1

Project URL:
http://cheri-cpu.org/
links to:
https://www.cl.cam.ac.uk/ research/security/ctsrd/

Open Source CHERI-RISC-V Cores

- Piccolo 32b microcontroller:
  https://github.com/CTSRD-CHERI/Piccolo
- Flute 64b/32b scalar core:
  https://github.com/CTSRD-CHERI/Flute
- Toooba 64b out-of-order core based on MIT Riscy-OOO core:
  https://github.com/CTSRD-CHERI/Toooba

Arm Morello Demonstrator Board



Conclusions

- CHERI protections are completely deterministic and solve fundamental security issues
- CHERI provides the hardware with more semantic knowledge of what the programmer intended
  - Toward the **principle of intentionality**
- Efficient **pointer integrity** and **bounds checking**
  - Eliminates buffer overflow/over-read attacks (finally!)
- Provide scalable, efficient compartmentalisation
  - Allows the **principle of least privilege** to be exploited to **mitigate known and unknown attacks**
- Transitioning the technology via **CHERI-RISC-V** and **Arm Morello**

The CHERI-RISC-V Extension

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

University of Cambridge and SRI International

RISC-V Week

Paris, 3-5 May 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.

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.

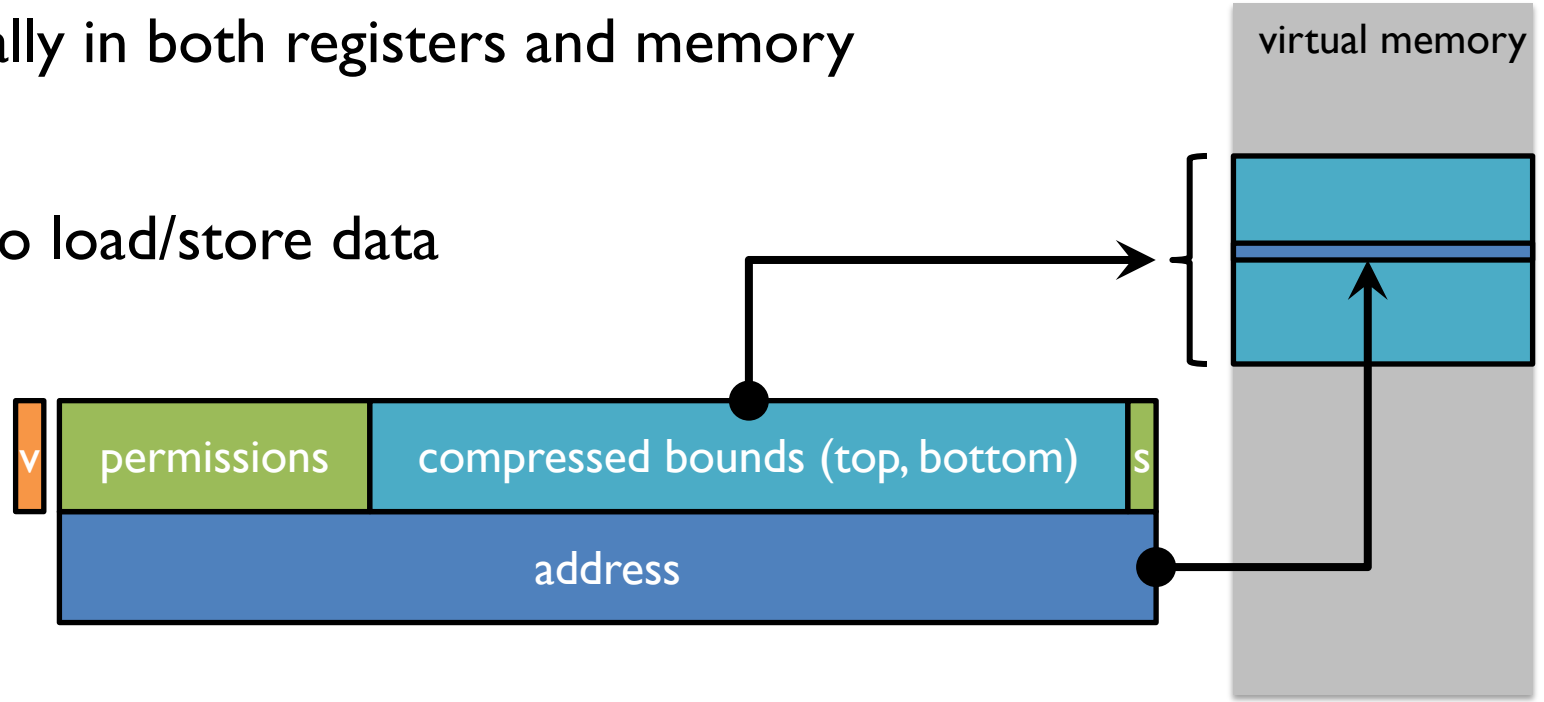
We also acknowledge the EPSRC REMS Programme Grant (EP/K008528/1), the ERC ELVER Advanced Grant (789108), the Isaac Newton Trust, the UK Higher Education Innovation Fund (HEIF), Thales E-Security, Microsoft Research Cambridge, Arm Limited, Google, Google DeepMind, HP Enterprise, and the Gates Cambridge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

CHERI Overview

- CHERI capability words are hardware-defined pointer structures that include bounds and permissions.

Tags distinguish capabilities and data in registers and memory

- Capabilities are preserved atomically in both registers and memory
- Capabilities can be dereferenced to load/store data (and other capabilities)
- All capabilities must be derived from more permissive capabilities



Basic Requirements

- Load and store capabilities (2 * XLEN + tag)

  Must move capabilities atomically

- Load and store through capability pointers

  What are my bounds and permissions?

- Jump to capabilities

  Change PC with its bounds and permissions (ie PCC)

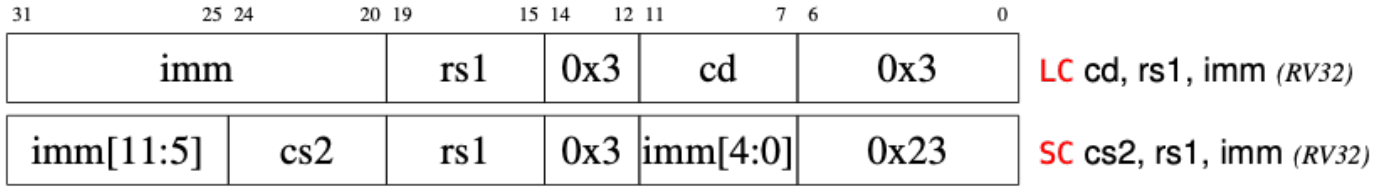
- Manipulate capability metadata in registers (bounds, permissions)

  - Derive narrower capabilities
  - Read capability fields
  - Pointer arithmetic

CHERI-RISC-V Load and Store Capabilities

- Used reserved LQ/SQ for 128-bit capabilities

When using 64-bit capabilities in RV32, the RV64 instructions `LD` and `SD` are reused to behave as `LC` and `SC` respectively.

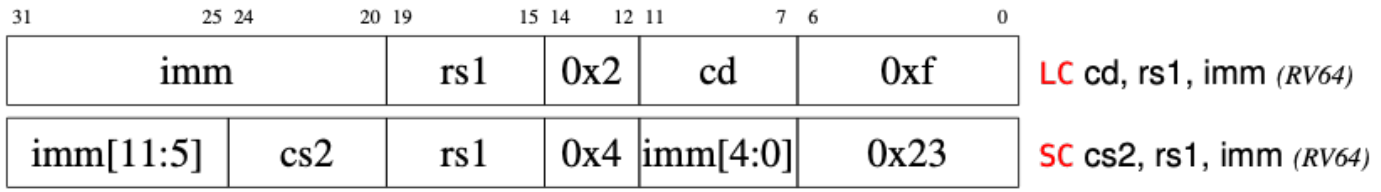


`LC` cd, rs1, imm *(RV32)*

`SC` cs2, rs1, imm *(RV32)*

Moves 2×XLEN, regardless of tag value; used for **memcpy**

When using 128-bit capabilities in RV64, the RV128 instructions `LQ` and `SQ` *(anticipated encoding)* are reused to behave as `LC` and `SC` respectively.



`LC` cd, rs1, imm *(RV64)*

`SC` cs2, rs1, imm *(RV64)*

Capability words must be aligned (unlike data) due to tag constraints

- Is this rv128? Not quite; will discuss later...

Load and Store through Capability Pointers

- New memory instruction encodings are expensive due to large immediates
- Use a *mode* bit for standard loads and stores to expect a capability address operand
  - Common cases: all integer pointers OR all capability pointers (depending on ABI)
  - Sacrifices intentionality in machine code
  - Mode bit is in PCC, so is naturally restored on function return
  - Integer pointers use Default Data Capability (DDC) bounds
    Capability pointers use their own bounds
- Also add loads and stores explicit to each kind of pointer (with no immediate)

Could we decide dynamically based on tag?
Intentionality says no!

Uncompressed instructions affected by capability mode

Integer load	LB	LH	LW	LD	LQ
Integer load (unsigned)	LBU	LHU	LWU	LDU	
Integer store	SB	SH	SW	SD	SQ
Floating-point load	FLW	FLD	FLQ		
Floating-point store	FSW	FSD	FSQ		
Atomic	LR	SC	AMOSWAP	AMOADD	AMOAND
Atomic (cont)	AMOOR	AMOXOR	AMOMAX	AMOMIN	
Address calculation	AUIPC[3]				

Compressed instructions affected by capability mode

Control flow	C.JALR	C.JR		
Compressed integer load	C.LW	C.LD	C.LWSP	C.LDSP
Compressed integer store	C.SW	C.SD	C.SWSP	C.SDSP
Compressed floating-point load	C.FLW	C.FLD	C.FLWSP	C.FLDSP
Compressed floating-point store	C.FSW	C.FSD	C.FSWSP	C.FSDSP

Split or Merged Register File

CHERI-MIPS used a split register file

Split register file

OR

Merged register file

- New register file vs. Unified (& extended) register file
- C0 holds the NULL capability
- CHERI-RISC-V supports split or merged
  - Instructions are identical
  - Only semantics are changed

- So-far only implemented merged
  - Better scientific comparability with the baseline

Losing the chance to add more registers at almost no encoding cost!

  - Less context to save and restore

Capabilities can point to code, not just data!

Jump to Capabilities

- Use explicit new jump to expect capability pointers
- Why not reuse the old jumps, repurposed with the mode bit? Because the mode bit is flipped by jumping to a capability!
- Helps intentionality, and encoding is cheap (just two operands)
- Compressed jumps do use the mode bit

Can't jump into capability mode with a compressed jump; only out.

0x7f	0xc	cs1	0x0	cd	0x5b	`JALR.CAP` cd, cs1
0x7f	0x14	cs1	0x0	cd	0x5b	`JALR.PCC` cd, cs1
0x7e	cs2	cs1	0x0	0x1	0x5b	`CInvoke` cs1, cs2

Capability Manipulation Instructions

- Legacy integer instructions produce upper bits of the NULL capability (tag cleared)
- Pointer arithmetic uses dedicated instructions (IncOffset) rather than reusing legacy ones (ADD)
  - Makes pointer arithmetic intentional
  - Could help micro-architectural optimisation (IncOffset must check representability)
- Throw exception vs. clear tag on illegal transformations
  - Throwing an exception gives precise debugging
  - Clearing the tag is more convenient for software and micro-architecture, and is also safer
  - We're moving from exceptions to tag clearing

Product of merged register file!

Morello is tag clearing

C.1.2 Capability-Modification Instructions

31–25	24–20	19–15	14–12	11–7	6–0	
0xb	cs2	cs1	0x0	cd	0x5b	CSeal cd, cs1, cs2
0xc	cs2	cs1	0x0	cd	0x5b	CUnseal cd, cs1, cs2
0xd	rs2	cs1	0x0	cd	0x5b	CAndPerm cd, cs1, rs2
0xe	rs2	cs1	0x0	cd	0x5b	CSetFlags cd, cs1, rs2
0xf	rs2	cs1	0x0	cd	0x5b	CSetOffset cd, cs1, rs2
0x10	rs2	cs1	0x0	cd	0x5b	CSetAddr cd, cs1, rs2
0x11	rs2	cs1	0x0	cd	0x5b	CIncOffset cd, cs1, rs2
imm[11:0]		cs1	0x1	cd	0x5b	CIncOffsetImm cd, cs1, imm
0x8	rs2	cs1	0x0	cd	0x5b	CSetBounds cd, cs1, rs2
0x9	rs2	cs1	0x0	cd	0x5b	CSetBoundsExact cd, cs1, rs2
uimm[11:0]		cs1	0x2	cd	0x5b	CSetBoundsImm cd, cs1, uimm
0x7f	0xb	cs1	0x0	cd	0x5b	CClearTag cd, cs1
0x1d	cs2	cs1	0x0	cd	0x5b	CBuildCap cd, cs1, cs2
0x1e	cs2	cs1	0x0	cd	0x5b	CCopyType cd, cs1, cs2
0x1f	cs2	cs1	0x0	cd	0x5b	CCSeal cd, cs1, cs2
0x7f	0x11	cs1	0x0	cd	0x5b	CSealEntry cd, cs1

C.1.3 Pointer-Arithmetic Instructions

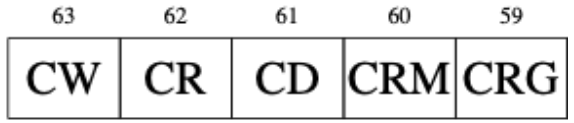
31–25	24–20	19–15	14–12	11–7	6–0	
0x12	cs2	cs1	0x0	rd	0x5b	CToPtr rd, cs1, cs2
0x13	rs2	cs1	0x0	cd	0x5b	CFromPtr cd, cs1, rs2
0x14	cs2	cs1	0x0	rd	0x5b	CSub rd, cs1, cs2
0x7f	0xa	cs1	0x0	cd	0x5b	CMove cd, cs1

Page Table Permissions

- Can track "capability free" pages to support sweeping for revocation
- Also supports experimental capability load detection

Possibly more complex than commercially necessary for experimentation...

Five new bits for sv39 Page Table Entries (PTEs):



PTE bits for capability stores (mirrors W & D flags)

CW	CD	Behavior
0	X	Trap on capability stores (exception code 0x1B)
1	0	Capability stores atomically raise CD or fault (as above)
1	1	Capability stores permitted

PTE bits for capability loads

CR	CRM	CRG	Behavior
0	0	0	Capability loads strip tags on loaded result
0	1	0	Capability loads fault (exception code 0x1A)
0	X	1	*Reserved for future use*
1	0	0	Capability loads are unaltered
1	0	1	*Reserved for future use*
1	1	X	*Reserved for generational load barriers*

CSRs and Special Capability Registers

- PCC and DDC are universally accessible CSRs
- DDC and PCC hold the "almighty capability" on reset
- Scratch registers per privilege level
- New *x*tval for capability violations
- CSR whitelist when Access System Registers (ASR) is not set in PCC

DDC in a GPR causes new dependencies for all non-capability memory operations

To gain permissions on ring change

Allows compartmentalisation in the kernel and a user-space supervisor

New CSRs

Encoding	Register
0x8C0	User capability control and status register (`uccsr`)
0x9C0	Supervisor capability control and status register (`sccsr`)
0xBC0	Machine capability control and status register (`mccsr`)

CSR whitelist

CSR	Read/Write
`cycle(h)`	Read-Only
`time(h)`	Read-Only
`instret(h)`	Read-Only
`hmpcounter(h)`	Read-Only
`fflags`	Read-Write
`frm`	Read-Write
`fcsr`	Read-Write

Special Capability Registers

	Register	Modes	Access	Reset	Extends
0	Program counter capability (**PCC**)	U, S, M	RO	∞	**PC**
1	Default data capability (**DDC**)	U, S, M	-	∞	-
4	User trap code capability (**UTCC**)	U, S, M	ASR	∞	`utvec`
5	User trap data capability (**UTDC**)	U, S, M	ASR	∅	-
6	User scratch capability (**UScratchC**)	U, S, M	ASR	∅	-
7	User exception PC capability (**UEPCC**)	U, S, M	ASR	∞	`uepc`
12	Supervisor trap code capability (**STCC**)	S, M	ASR	∞	`stvec`
13	Supervisor trap data capability (**STDC**)	S, M	ASR	∅	-
14	Supervisor scratch capability (**SScratchC**)	S, M	ASR	∅	-
15	Supervisor exception PC capability (**SEPCC**)	S, M	ASR	∞	`sepc`
28	Machine trap code capability (**MTCC**)	M	ASR	∞	`mtvec`
29	Machine trap data capability (**MTDC**)	M	ASR	∅	-
30	Machine scratch capability (**MScratchC**)	M	ASR	∅	-
31	Machine exception PC capability (**MEPCC**)	M	ASR	∞	`mepc`

Both RV32 and RV64 are supported

- RV32 (CHERI64) and RV64 (CHERI128) are specified

Research is ongoing for mixed-width implementations

- Would be interesting to experiment with RV64 with CHERI64 (sv32) to learn about implications of RV128 in CHERI128

CHERI128 Relationship with RV128

- Not quite RV128
  - Added Load and Store Quad instructions
  - But lacking 128-bit arithmetic instructions
- If an implementation had both, would we want new loads and stores to preserve intentionality? (New tag-clearing LQ/SQ)

sv64 (64-bit virtual addresses) with RV128 is not unreasonable; i.e. XLEN = 2×"VA-LEN"

Conclusions

- CHERI academic research has moved fully to the CHERI-RISC-V architecture
- Lots of implementations:

The Sail-CHERI-RISC-V model (CHERI-64/CHERI-128)

QEMU simulator (CHERI-64/CHERI-128)

CHERI Ibex (CHERI-64) (not up-to-date)

CHERI Piccolo (CHERI-64)

CHERI Flute (CHERI-64/CHERI-128)

CHERI RiscyOO CHERI-RISC-V (CHERI-128)

- Fully featured LLVM compiler support
- CheriBSD with full software stack building in pure-capability mode

Four CHERI-RISC-V Micro-Architectures

Peter Rugg, Jonathan Woodruff, Alexandre Joannou, Ivan Ribeiro,
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, 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, Alex Richardson, Michael Roe, Colin Rothwell, Hassen Saidi, Peter Sewell, Thomas Sewell, Stacey Son, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Konrad Witaszczyk, Hongyan Xia, and Bjoern A. Zeeb

University of Cambridge and SRI International

RISC-V Week

Paris, 3-5 May 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.

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.

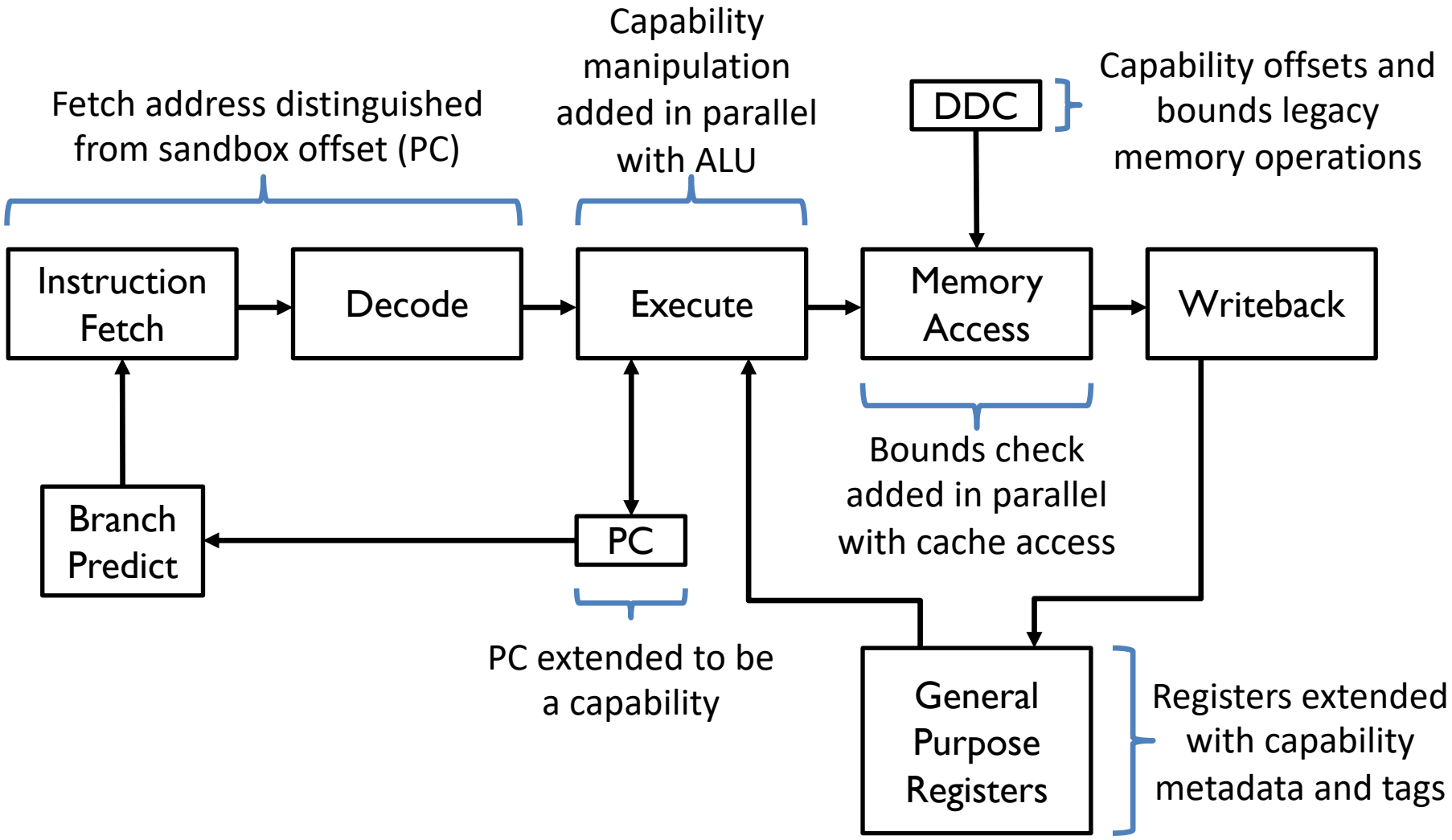
Contents

- How to add CHERI to a core
- Shared components
  - Tag controller
  - Compression library
- Four implementations
- Per-core optimizations

CHERI-RISC-V Microarchitectural Changes

- Extend decode, registers, arithmetic, and memory access for capabilities
- Capability CSRs and new capability exception behaviors
- Widen caches and add tag support
- Tagged AXI interconnect
- Performance counters
- Analyze performance, identify bottlenecks, and optimize microarchitecture to meet timing and performance goals

CHERI-RISC-V Pipeline Changes



CHERI-RISC-V Reused Components

- AXITagController

  Tag controller module manages combining tags with data
- Capability encoding library

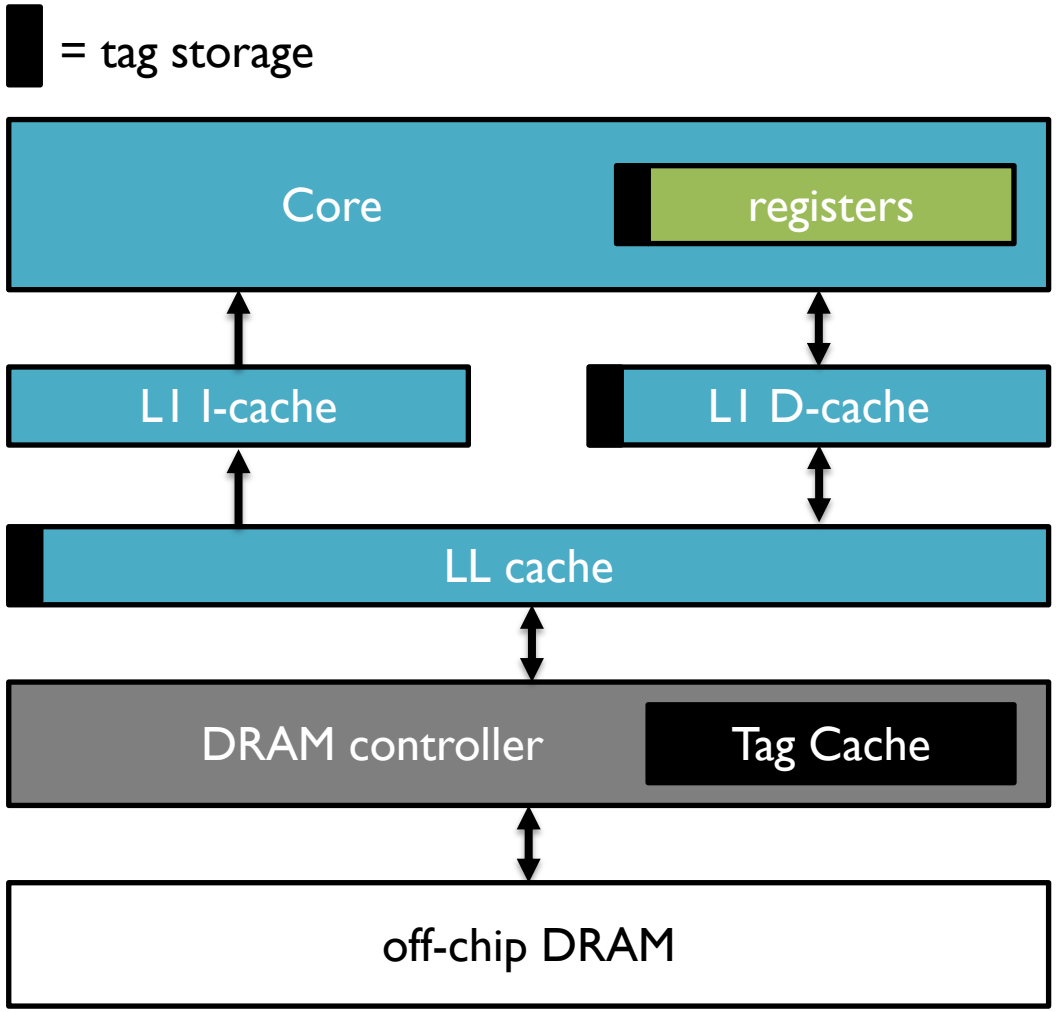
  Memory, register, and pipeline capability formats with functions for each
- TestRIG testing framework

  CHERI instruction definitions and templates for testing deep capability states

Tagging Capabilities

- Capabilities have a hidden validity tag
  - In registers and memory
- Tag bit is critical to security
  - Conventional operations (arith, memory) clear the tag
  - Only capability instructions preserve the tag and guarantee monotonic decrease in rights
- One hidden bit per 128-bits avoids using other integrity measures (no crypto needed…)

Propagating tags from registers to DRAM



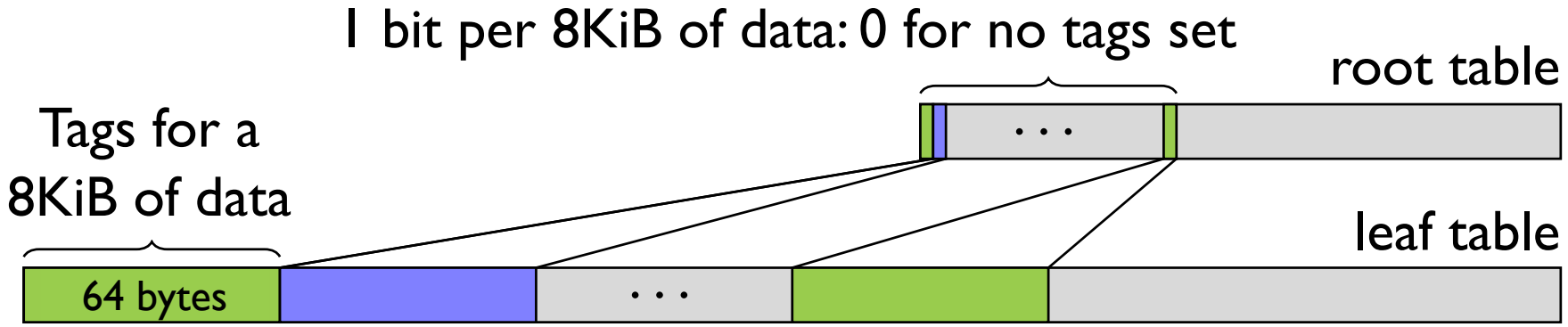
- Tags stored in registers and caches with data to ensure consistency
- Off-chip storage:
  - Tags stored in upper 1% of commodity DRAM
  - Tag cache per DRAM controller reduces DRAM traffic
  - No consistency issues

Hierarchical Tag Compression

- Size tag cache line length to 64-byte DDR4 burst transfer size
  ⇒ one line covers tags for 8KiB of memory (128-bit capabilities)
- Many lines don't contain tags (code, large blocks of data, disk cache, etc.)
  - So handling tag sparseness is important
  - **Only want to pay for tagging when needed**

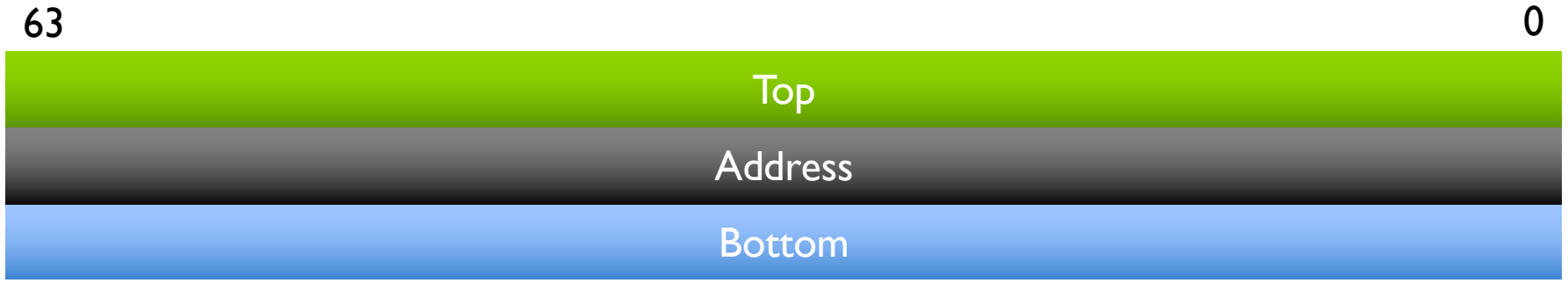
Tag Compression

- 2-level tag table
- Each bit in the **root** level indicates all zeros in a **leaf** group
- Reduces tag cache footprint
- Amplifies cache capacity

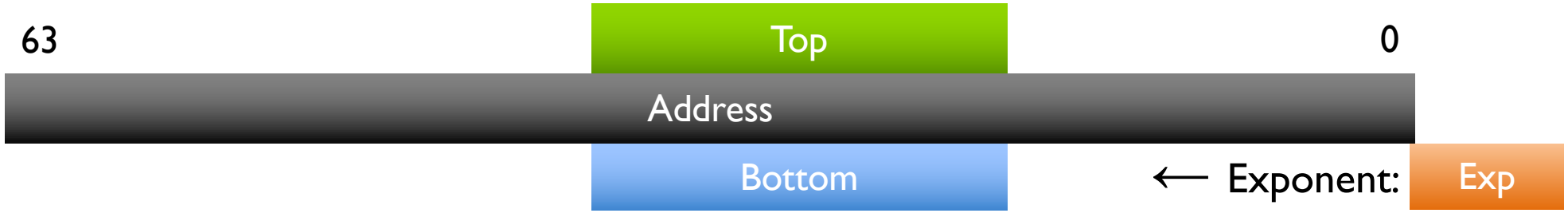


Capability Compression

Capabilities encode at least 3 64-bit fields:



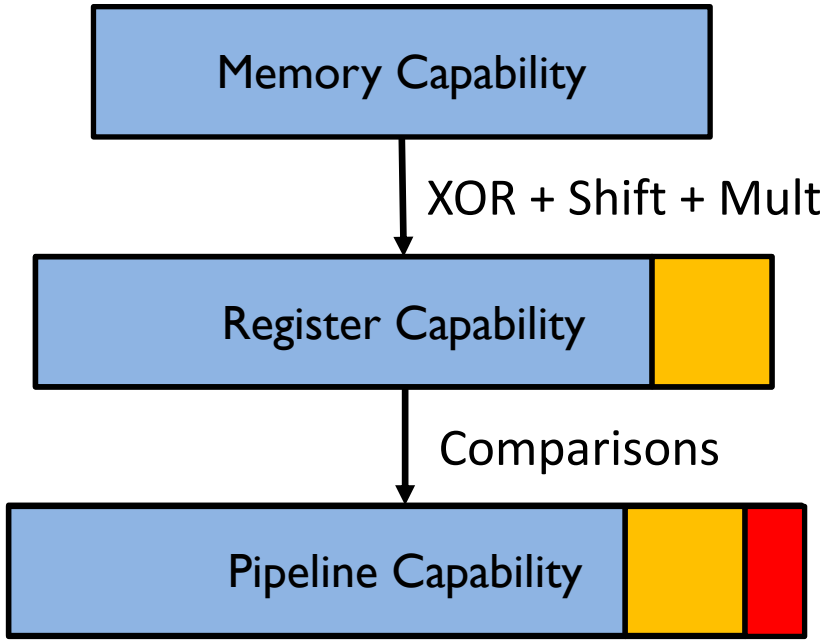
But we can encode the Top and Bottom relative to the Address:



- Larger objects require greater alignment
- Address must be "near" the Top and Bottom

Capability Compression Shared Library

- Capabilities partially decompressed in stages throughout pipeline
- Shared library between all four implementations



Architectural “CHERI-concentrate” format
128 bits

“Exponent” and “Mantissa” extracted
152 bits

Compression edge-cases identified
162 bits

Baseline Processors

- Piccolo - Bluespec

  32-bit 3-stage in-order microcontroller (Bluespec Inc.)

- Ibex - Verilog

  32-bit highly area-optimised microcontroller (lowRISC)

- Flute - Bluespec

  64-bit 5-stage in-order microcontroller (Bluespec Inc.)

- Toooba - Bluespec

  64-bit out-of-order superscalar (Bluespec Inc.) based on RiscyOO (MIT)

CHERI-RISC-V Scaling Across Cores

- CHERI Piccolo and CHERI Ibex – Power and Area

  Small core (no MMU or FPU), so logic overhead is most significant

  Where present, very small cache, so DRAM traffic (i.e. power) overhead is pronounced

- CHERI Flute – Frequency and Prediction

  Deeper pipeline, higher frequency (100Mhz), so sensitive to timing

  Performance depends on predicting branches and forwarding results

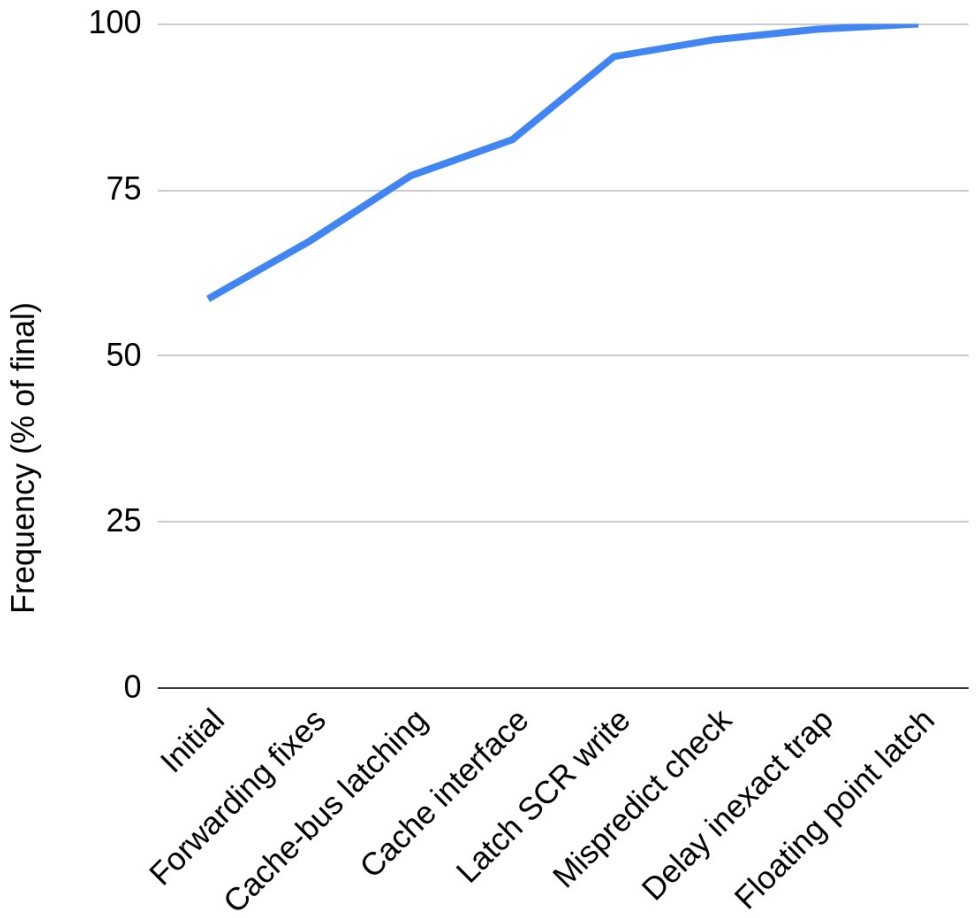
- CHERI Toooba – Concurrency, Prediction, and Multicore

  Superscalar execution and performance is sensitive to parallelism

  Higher cost of misprediction and performance is very sensitive to accuracy

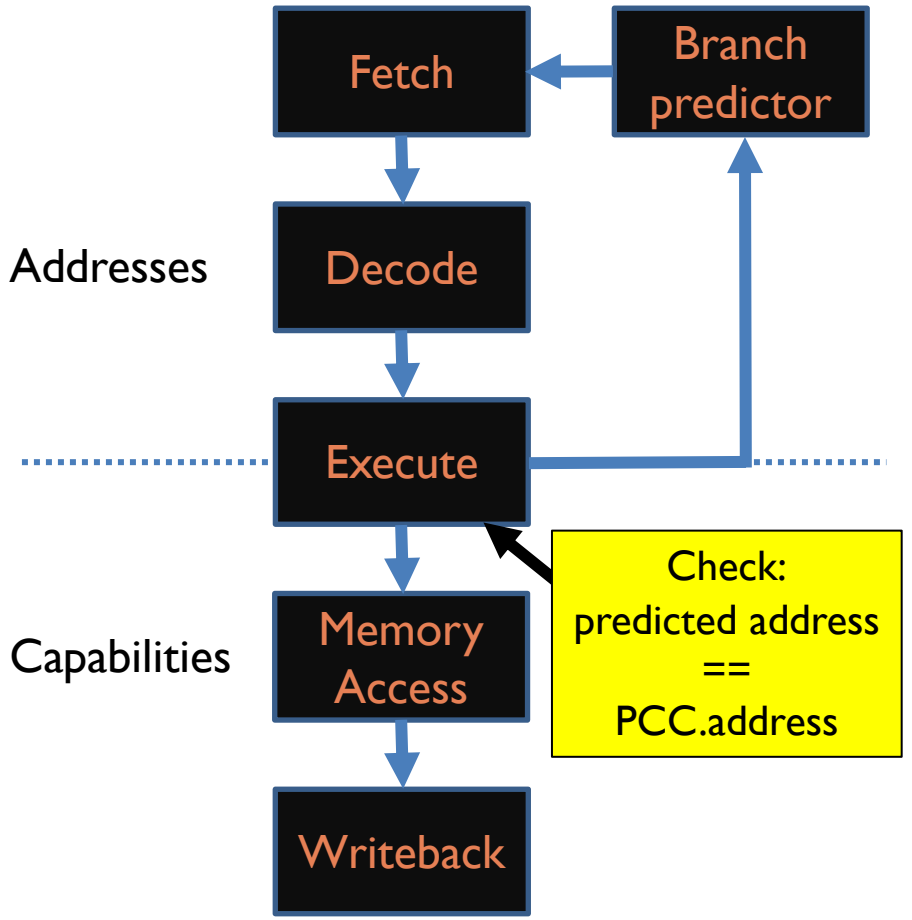
CHERI Flute – Challenge 1 – Frequency

- Highest target frequency (100MHz). Initial design failed timing.
- Solution: careful optimization
  - Make forwarding independent of bounds check
  - Add latch between instruction cache and AXI interconnect
  - Expose cache invariants to synthesis tool
  - Latch computed SCR value
  - Refactor check for misprediction
  - Delay capability trap infrequent path (performance penalty only on trap)
  - Rebalance floating point latching



CHERI Flute – Challenge 2 – Prediction

- First core with prediction: fetch requests are sent to the instruction memory before the capability used is known
- Various possibilities, e.g. predict entire capability, predict address or offset
- Solution: Divide core into front-end working with predicted raw addresses, and back-end working with full capabilities
- Also need to speculate "capability encoding mode": add to predicted state



CHERI Flute – Challenge 3 – DRAM traffic

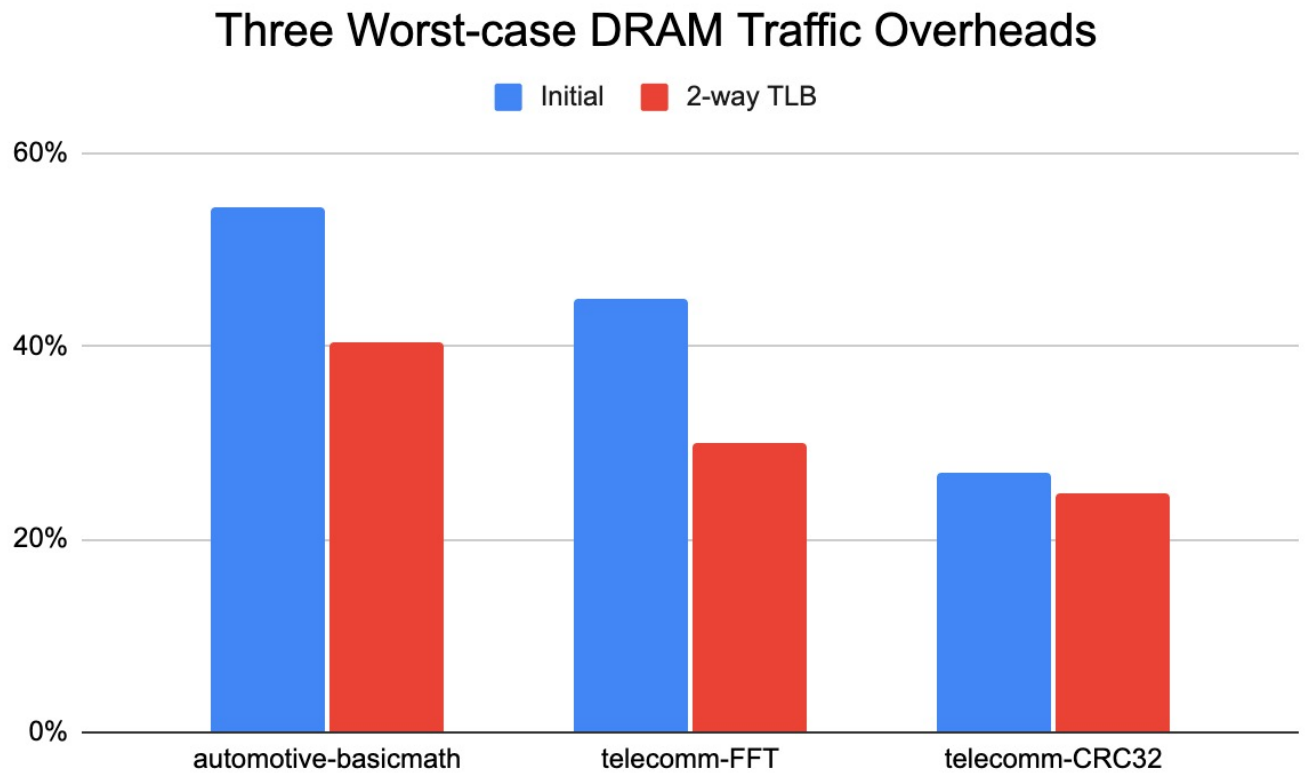
- DRAM overhead was high initial CHERI-Flute implementation (**16.9%** overhead)

  Write-through caches: capability writes double DRAM traffic of integer writes. Pushing to the stack twice as expensive with purecap!

- Page table miss traffic was a large contribution to DRAM traffic

  The page table walks (uncached) directly accessed DRAM

  2-way associative TLB meets the power/DRAM 5% target (**-0.6%** overhead)



CHERI Toooba – Challenge 1 – Reorder Buffer

Initial **Reorder Buffer** (state for 64 in-flight instructions)
184% overhead

- Solution 1: Change enum structure to avoid explosion

  Bluespec compiler was inefficient for sparse enum assignments; CHERI Toooba used these more than Toooba

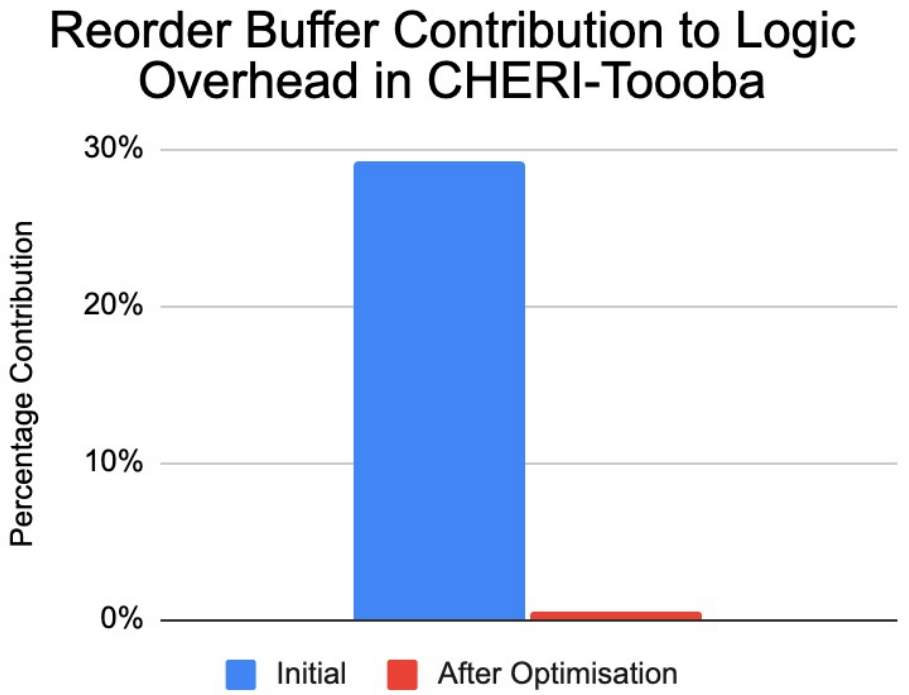
- Solution 2: Eliminate three 64-bit registers from records

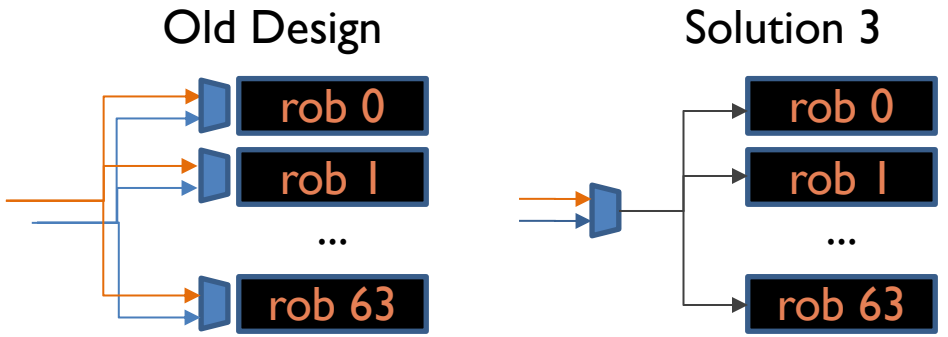
  Derivable from other state (TVAL) or unnecessary in our configuration (dest_data, store_data)

- Solution 3: MUX out of Reorder Buffer rows into ALU

  64 MUXs (1 per record) -> 2 MUXs (1 per ALU pipe)

Currently **2%** overhead for Reorder buffer





CHERI Toooba – Challenge 2 – PCC Metadata

- Prediction/fetch/decode granularity
  32 bits -> 16 bits for compressed instructions

  Each granule has PCC and predicted next PCC:
  256 bits of address metadata per 16-bit granule

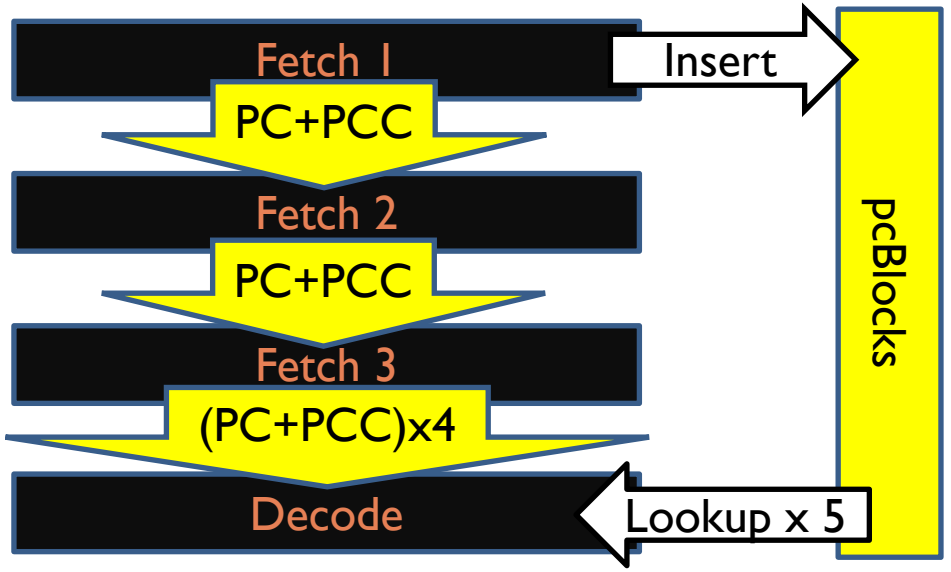
- The distinct PC segments guaranteed to be 1/8 the maximum granules

  Fetch blocks of up to 4 granules share a segment
  Any predicted PC will be shared with the next block

- Introduce compression table in 4 stages of Fetch

  PC now represented by 15 bits

  30 bits per 16-bit granule



Pipeline Token

```
typedef struct {
    PcLSB lsb;
    PcIdx idx;
} PcCompressed deriving(Bits,Eq,FShow);
```

12 bits

3 bits

8 entries

Table Instantiation

```
IndexedMultiset#(PcIdx, PcMSB, SupSizeX2) pcBlocks <- mkIndexedMultisetQueue;
function CapMem decompressPc(PcCompressed p) = {pcBlocks.lookup(p.idx),p.lsb};
```

CHERI Toooba – Challenge 3 – BTB Contention

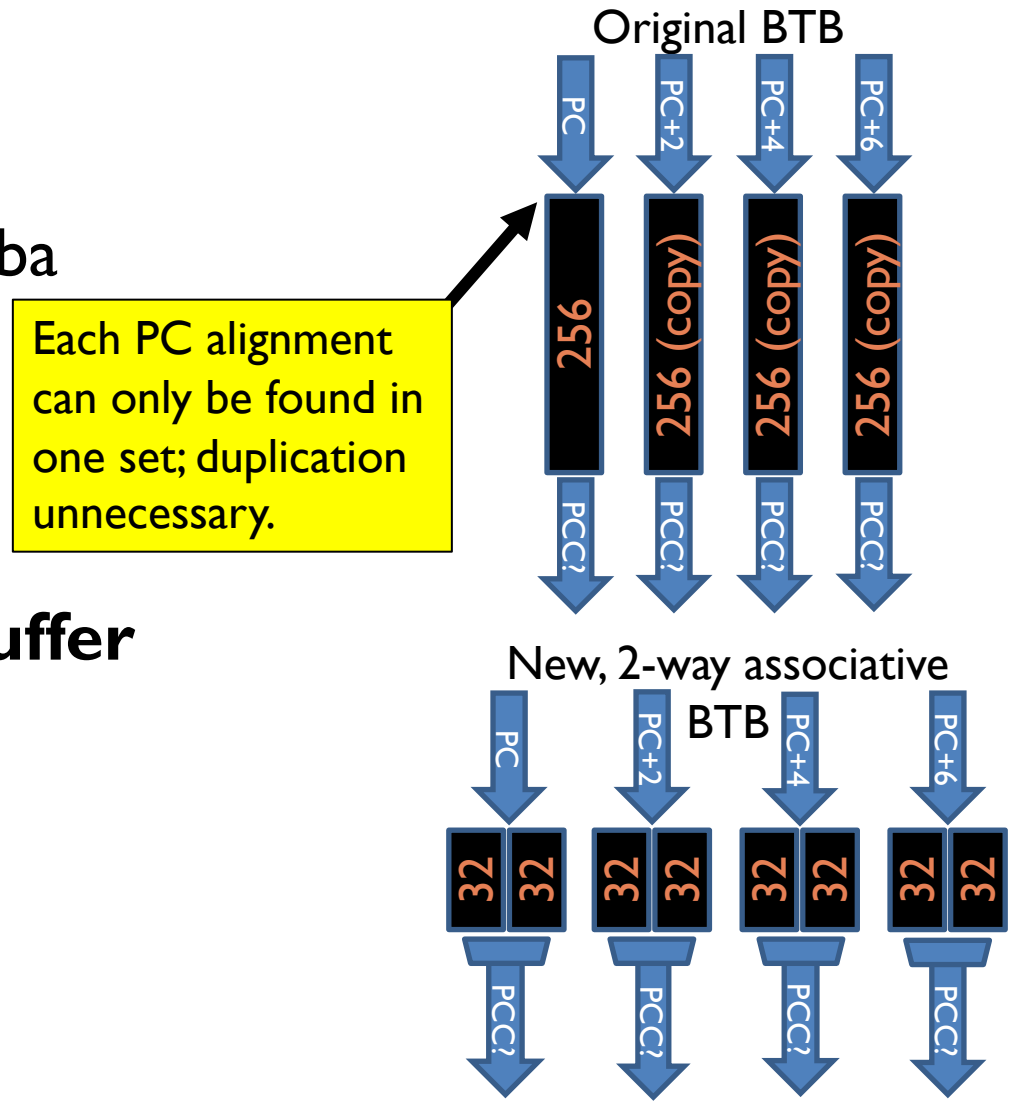
- CHERI pointers cause higher DRAM traffic
- Misprediction causes wasted traffic in Toooba

  Deep, out-of-order speculation means unnecessary memory traffic caused by misspeculated loads

- Use a 2-way associative **Branch Target Buffer** (BTB) to compensate for DRAM overhead

  13.07% -> 4.6% DRAM Overhead

- Segmented BTB to more-than-compensate for the area overhead



Conclusions

- CHERI security extensions have been applied to a range of cores
  - Microcontrollers to a superscalar core for CHERI-RISC-V
  - Arm Morello SoC also demonstrates that CHERI can be added to a commercial core (Neoverse N1)

- CHERI comes with some costs, but careful microarchitectural optimization reduces these substantially

The CHERI-RISC-V Software Ecosystem and Toolchain

Alex Richardson, Jessica Clarke, David Chisnall, Brooks Davis, John Baldwin
Robert N. M. Watson, Simon W. Moore, Peter G. Neumann, Hesham Almatary, Alasdair Armstrong, Peter Blandford-Baker, Hadrien Barrel, Thomas Bauereiss, Ruslan Bukin, 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, Prashanth Mundkur, Edward Napierala, George Neville-Neil, Robert Norton-Wright, Lucian Paul-Trifu, Allison Randal, Ivan Ribeiro, Michael Roe, Peter Rugg, Peter Sewell, Thomas Sewell, Stacey Son, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Konrad Witaszczyk, Jonathan Woodruff, and Hongyan Xia

University of Cambridge and SRI International

RISC-V Week

Paris, 3-5 May 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.

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.

Compiling for CHERI RISC-V

- How can we make use of CHERI capabilities for C/C++ code?
- Use CHERI LLVM: https://github.com/CTSRD-CHERI/llvm-project

C

```
#include <stdio.h>

int main(void) {
    printf("Hello world\n");
}
```

1. Adjust stack pointer
2. Save return address
3. Generate pointer to .Lstr
4. Call printf()
5. Set return value to zero
6. Restore return address
7. Adjust stack pointer
8. Return

RISC-V Assembly
-march=rv64gc -mabi=lp64d

```
main:
    addi sp, sp, -16
    sd ra, 8(sp)
.LBB0_1:
    auipc a0, %pcrel_hi(.Lstr)
    addi a0, a0, %pcrel_lo(.LBB0_1)
    call puts
    mv a0, zero
    ld ra, 8(sp)
    addi sp, sp, 16
    ret
.Lstr:
.asciz "Hello world"
```

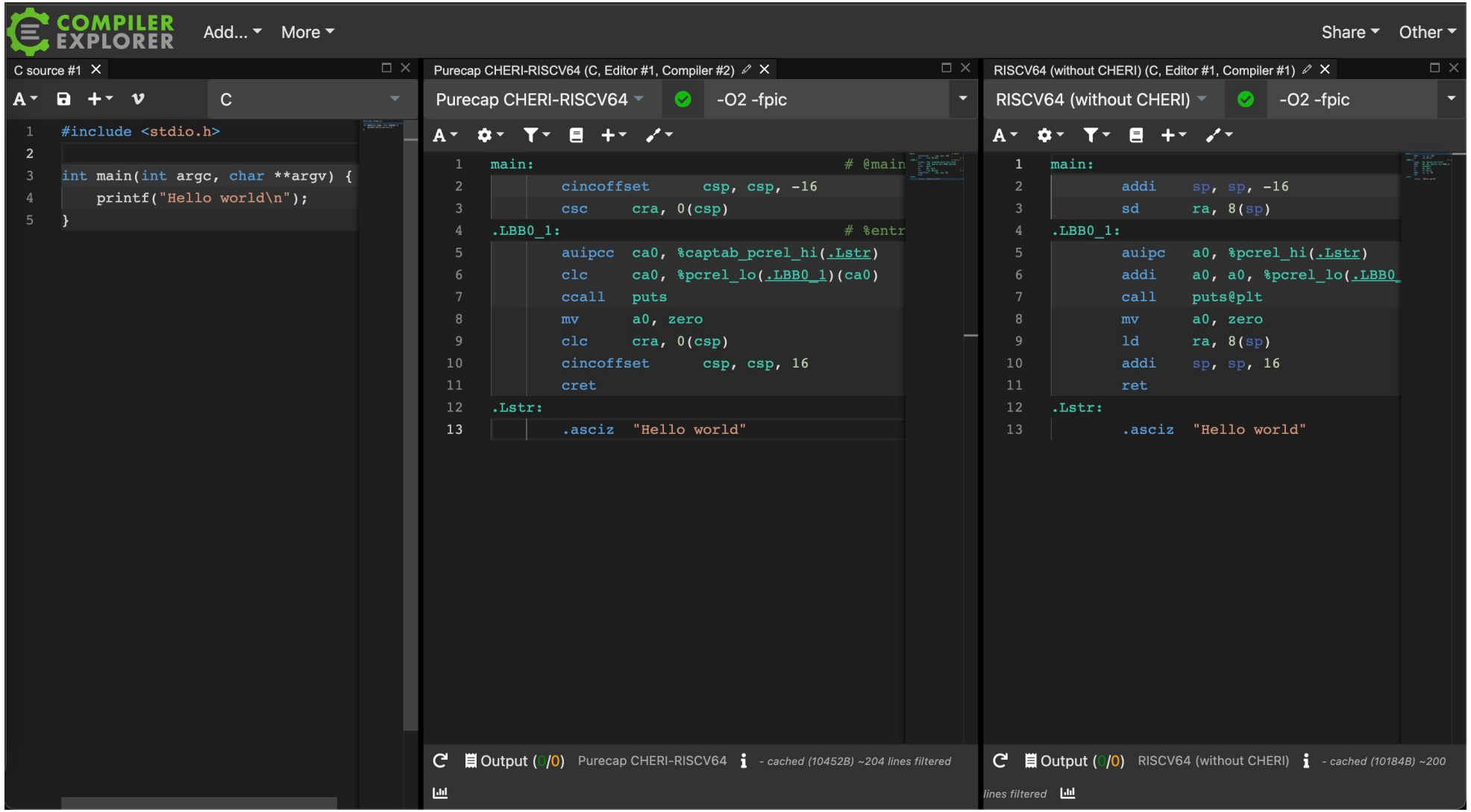
CHERI RISC-V Assembly
-march=rv64gcxcheri -mabi=l64pc128d

```
main:
    cincoffset csp, csp, -16
    csc cra, 0(csp)
.LBB0_1:
    auipcc ca0, %captab_pcrel_hi(.Lstr)
    clc ca0, %pcrel_lo(.LBB0_1)(ca0)
    ccall puts
    mv a0, zero
    clc cra, 0(csp)
    cincoffset csp, csp, 16
    cret
.Lstr:
.asciz "Hello world"
```

SRI International

UNIVERSITY OF CAMBRIDGE

https://cheri-compiler-explorer.cl.cam.ac.uk

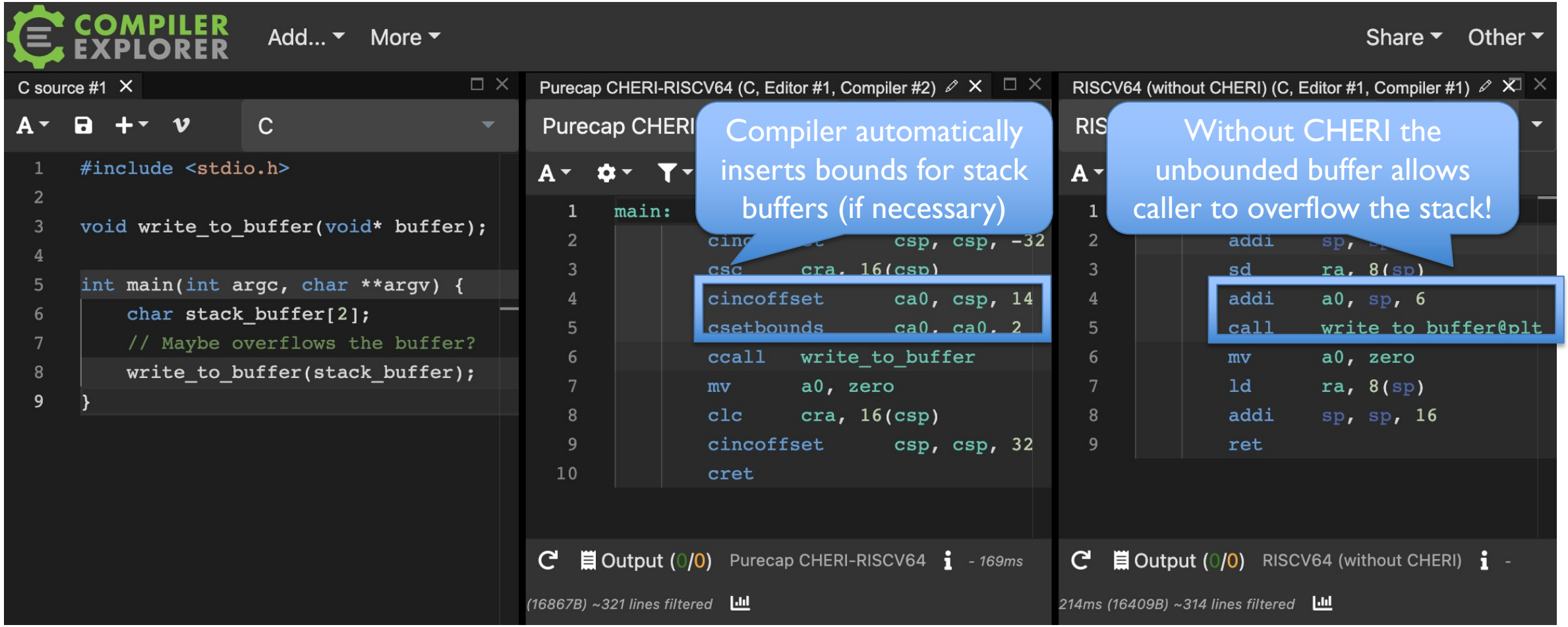


Automatic capability bounds (1)

- OS kernel, run-time linker, memory allocator(s) and compiler take care of **automatic capability bounds refinement**
- On `execve()` initial (bounded) capabilities set up by OS kernel
- Run-time linker loads data & code for libraries using `mmap()`
- Kernel returns a new bounded capability for `mmap()`
- Run-time linker then processes relocations and creates bounded capabilities for global variables and functions
- `malloc()` ensures that allocation is correctly bouded

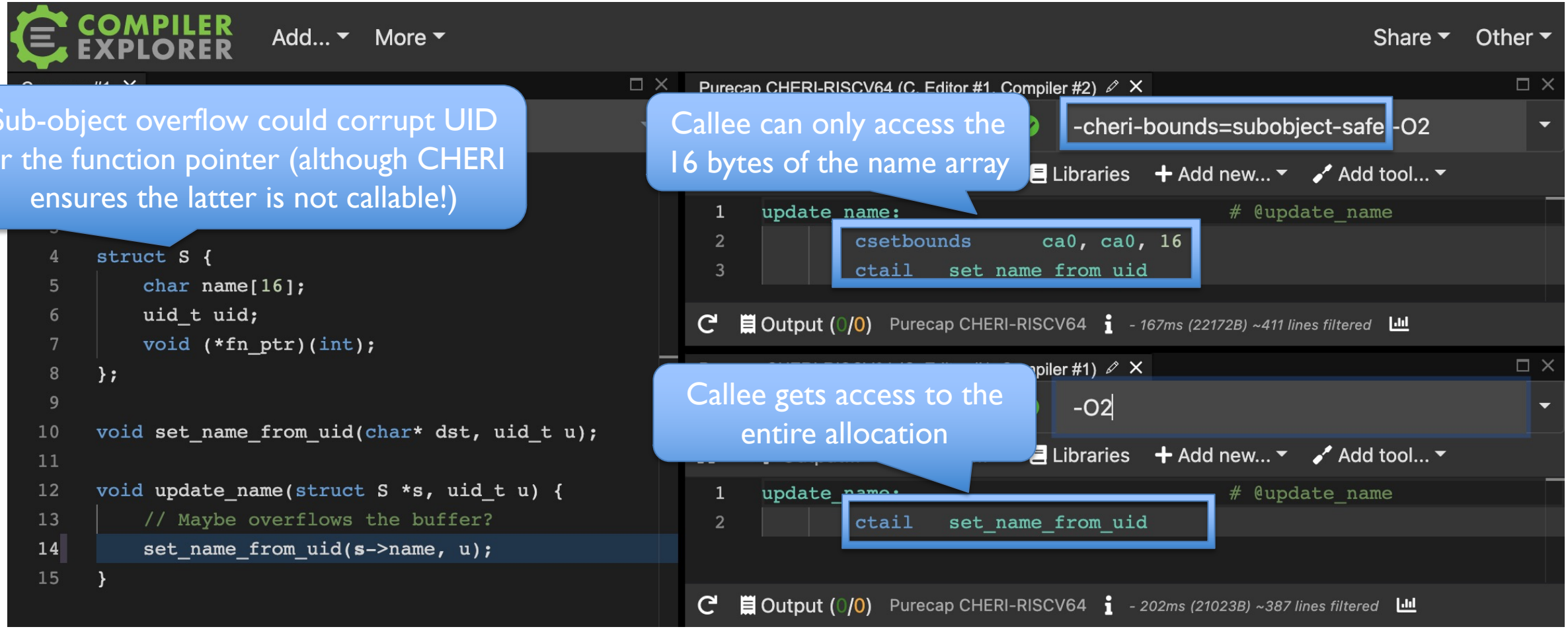
Automatic capability bounds (2)

The compiler automatically adds bounds e.g. for stack allocations:

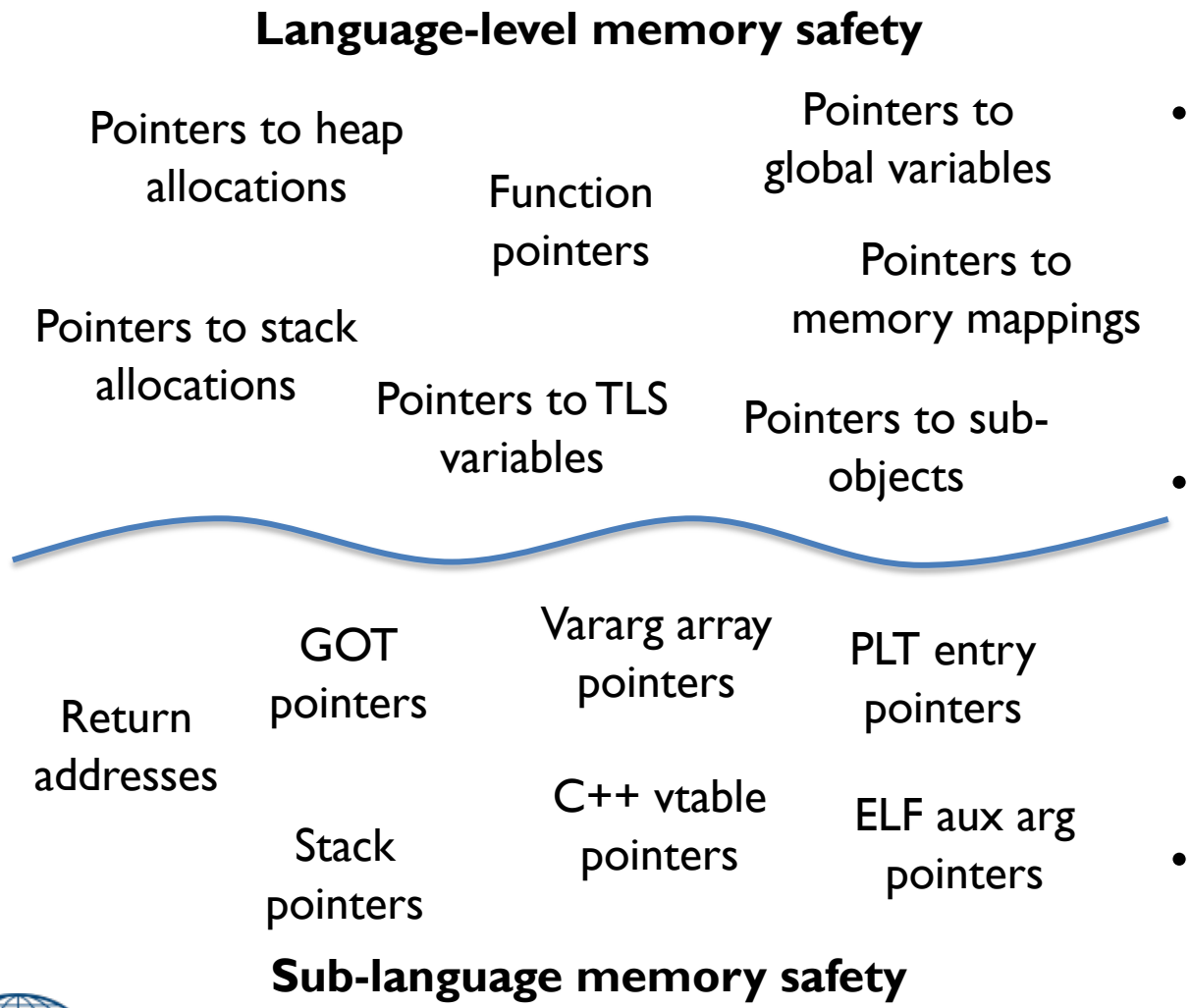


Automatic capability bounds (3)

Opt-in support for **preventing sub-object overflows** (at a moderate compatibility cost).



Memory protection for the language and the language runtime

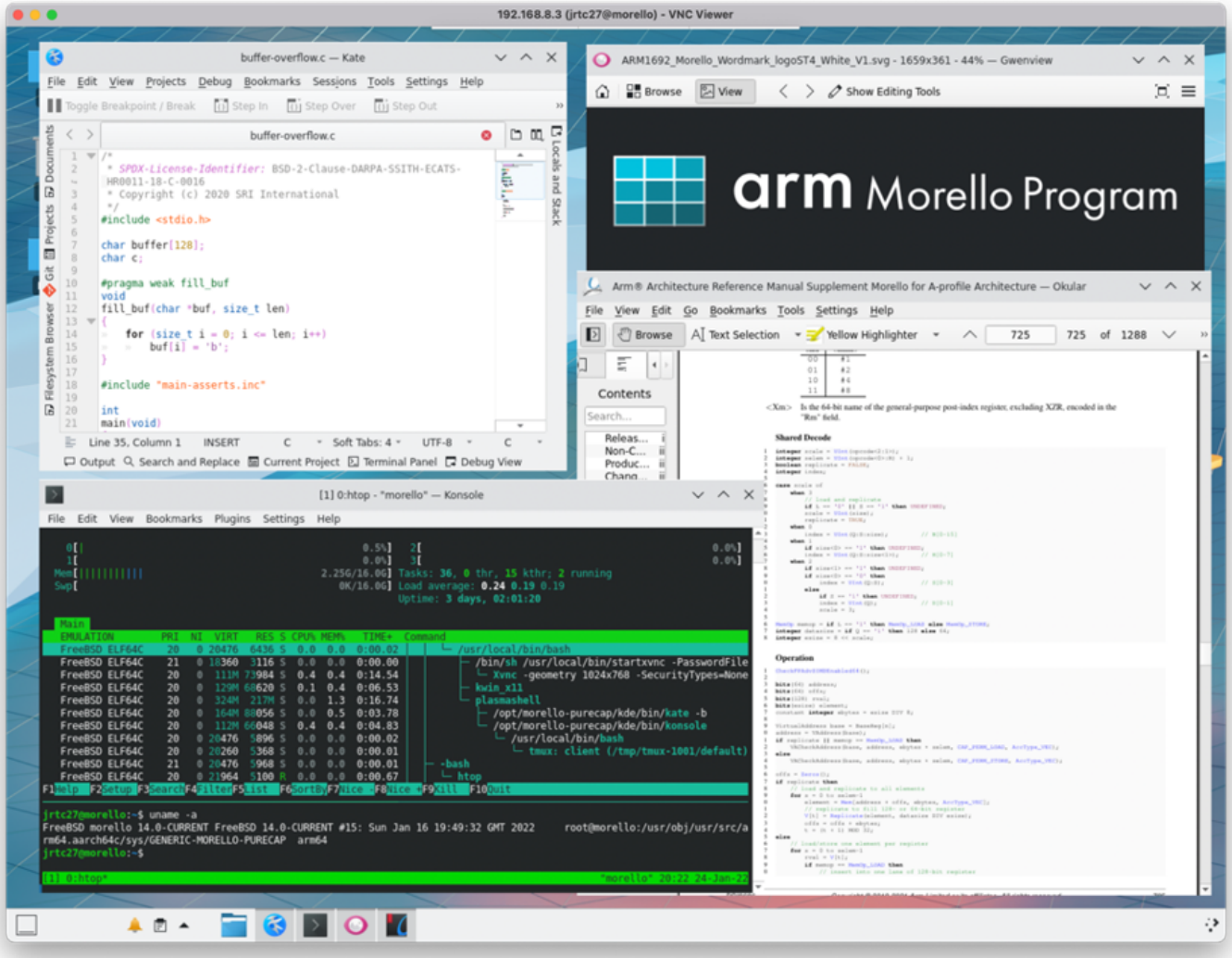


- Capabilities are refined by the kernel, run-time linker, compiler-generated code, heap allocator, …
- Protection mechanisms:
  - Referential memory safety
  - Spatial memory safety + privilege minimization
  - Temporal memory safety
- Applied **automatically** at two levels:
  - **Language-level pointers** 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

Portability when compiling for CHERI-RISC-V

- In general, most code will just work as-is or be flagged by compiler warnings
- However, a few common issues exist:
  - Insufficient alignment (e.g. in memory allocators)
    - Loading/storing pointers needs stricter alignment (16 bytes for CHERI-128)
    - Allocators should use `alignof(max_align_t)` instead of hardcoding 8
  - Incorrectly casting pointers to or from `long`
    - `long` can only hold the address part, bounds and validity are lost when casting
    - Please use C99 `uintptr_t` instead (using `long` is not allowed by C standard)
  - Updating pointers after `realloc()` may require auditing
- More details: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-947.pdf

Operating system support



- Most mature OS is CheriBSD, FreeBSD with full support for pure-capability CHERI code
  - https://github.com/CTSRD-CHERI/cheribsd
- Even runs a memory safe KDE graphical desktop!
  - **0.026% LoC modification rate** across full corpus for memory safety
  - **73.8% mitigation rate** across full corpus, using memory safety and compartmentalization

Debugging

- We have a version GDB with support for CHERI-RISC-V available at https://github.com/CTSRD-CHERI/gdb
- Generally works just as you are used to it

```
Starting program: /opt/cheri-exercises/buffer-overflow-stack-cheri

Program received signal SIGPROT, CHERI protection violation
Capability bounds fault caused by register ca1.
0x0000000000101dae in write_buf (buf=0x3fffdfff5c [rwRW,0x3fffdfff5c-0x3fffdfff6c] "", ix=16)
at src/exercises/buffer-overflow-stack/buffer-overflow-stack.c:13
13      src/exercises/buffer-overflow-stack/buffer-overflow-stack.c: No such file or
directory.
(gdb) bt
#0  0x0000000000101dae in write_buf (buf=0x3fffdfff5c [rwRW,0x3fffdfff5c-0x3fffdfff6c] "",
ix=16) at src/exercises/buffer-overflow-stack/buffer-overflow-stack.c:13
#1  0x0000000000101e98 in main () at src/exercises/buffer-overflow-stack/buffer-overflow-
stack.c:31
(gdb)
```

Debugging

- We have a version GDB with support for CHERI-RISC-V available at https://github.com/CTSRD-CHERI/gdb
- Generally works just as you are used to it

```
Starting program: /opt/cheri-exercise
Program received signal SIGPROT, CHERI protection violation
Capability bounds fault caused by register ca1.
0x0000000000101dae in write_buf (buf=0x3fffdfff5c [rwRW,0x3fffdfff5c-0x3fffdfff6c] "", ix=16)
at src/exercises/buffer-overflow-stack/buffer-overflow-stack.c:13
13      src/exercises/buffer-overflow-stack/buffer-overflow-stack.c: No such file or
directory.
(gdb) bt
#0  0x0000000000101dae in write_buf (buf=0x3fffdfff5c [rwRW,0x3fffdfff5c-0x3fffdfff6c] "",
ix=16) at src/exercises/buffer-overflow-stack/buffer-overflow-stack.c:13
#1  0x0000000000101e98 in main () at src/exercises/buffer-overflow-stack/buffer-overflow-
stack.c:31
(gdb)
```

Capability permissions

Upper and lower bound

Emulators

- QEMU: https://github.com/CTSRD-CHERI/qemu
  - Fast and mature emulator including instruction tracing and GDB support for bare-metal debugging
  - Easiest way to get started with CHERI – no need for an FPGA
- Sail model: https://github.com/CTSRD-CHERI/sail-cheri-riscv
  - Reference model for 32 and 64-bit CHERI-RISC-V

Tying it all together - cheribuild

- Yet another meta build system for CHERI software
- Builds all the projects needed to run CheriBSD (and much more)
- To get started: `cheribuild.py run-riscv64-purecap -d`
- Can also cross-compile hundreds of additional projects that aren't packaged yet for CheriBSD, e.g. `kde-x11-desktop`
- Automates various steps such as build and installation, booting CheriBSD and running (cross-compiled) test suites
- Available at https://github.com/CTSRD-CHERI/cheribuild

Conclusions

- CHERI RISC-V has a mature software ecosystem including OS support (CheriBSD), emulators (QEMU) and debuggers (GDB)
- All of these projects are open-source and available on GitHub: https://github.com/CTSRD-CHERI/
- Key takeaways from this talk:
  - Using CHERI-RISC-V should feel essentially the same as RISC-V
  - Even if you don't plan on using CHERI for your code: please use `(u)intptr_t` when casting pointers to integers

CHERI-RISC-V demo

Demonstrating Memory-safety Features under CheriBSD on a Multi-core, Superscalar Softcore

Franz Fuchs, 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, 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 and SRI International

RISC-V Week
Paris, 3-5 May 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.

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.

This presentation is based on material prepared by the CHERI project:

Adversarial CHERI Exercises and Missions:

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

CHERI Workshops:

https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/workshops/

Cross-compiling CHERI Binaries

```
clang -target riscv64-unknown-freebsd --sysroot=/path/to/sysroot-
riscv64-purecap/ -mno-relax -march=rv64gc -mabi=lp64d source.c -o
binary

clang -target riscv64-unknown-freebsd --sysroot=/path/to/sysroot-
riscv64-purecap/ -mno-relax -march=rv64gcxcheri -mabi=lp64pc128d
-Wcheri source.c -o binary
```

Compile for baseline

Compile for CHERI-RISC-V

Enable all CHERI warnings

Compile for 128-bit capability CHERI target

CHERI Pointers and Addresses

```
Interacting with CheriBSD. Use CTRL+] to exit
/dev/ttyUSB2  115200,8,N,1 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---

--- Settings: /dev/ttyUSB2  115200,8,N,1
--- RTS: active    DTR: active    BREAK: inactive
--- CTS: inactive  DSR: inactive  RI: inactive  CD: inactive
--- software flow control: inactive
--- hardware flow control: inactive
--- serial input encoding: UTF-8
--- serial output encoding: UTF-8
--- EOL: LF
--- filters: default

[PEXPECT\PROMPT]>sysctl hw.machine hw.ncpu
hw.machine: riscv
hw.ncpu: 2
[PEXPECT\PROMPT]>./print-pointer-baseline
size of pointer: 8
size of address: 8
[PEXPECT\PROMPT]>./print-pointer-cheri
size of pointer: 16
size of address: 8
[PEXPECT\PROMPT]>
```

```
int main(void){
  printf("size of pointer: %zu\n", sizeof(void *));
  printf("size of address: %zu\n", sizeof(ptraddr_t));
  return (0);
}
```

Dual-core processor running on FPGA

Legacy: pointer size = address size

CHERI: pointer size = address size + 8 bytes of CHERI metadata

CHERI tag protection (1/2)

```
char buf[0x1FF];
volatile union { char *ptr;
                 char bytes[sizeof(char*)];
}p;

for (size_t i = 0; i < sizeof(buf); i++) {
     buf[i] = i;
}

p.ptr = &buf[0x10F];
char *q = (char*)(((uintptr_t)p.ptr) & ~0xFF);
printf("q=%" PRINTF_PTR " (0x%zx into buf)\n", q, q - buf);
printf("*q=%02x\n", *q);

p.bytes[0] = 0;
char *r = p.ptr;
printf("r=%" PRINTF_PTR " (0x%zx)\n", r, r - buf);
printf("*r=%02x\n", *r);
```

Declare a character buffer and a union struct

Fill the buffer with sequence of values

Assign *p.ptr* to part of *buf*

Assign *q* to an aligned address in *buf*, and print status

Attempt to assign *r* to an aligned address in *buf* via the *p.bytes* field and print status

CHERI tag protection

```
[PEXPECT\PROMPT]>./corrupt-pointer-baseline
buf=0x80d352b9 &p=0x80d352b0
p.ptr=0x80d353c8 (0x10f into buf) *p.ptr=0f
q=0x80d35300 (0x47 into buf)
*q=47
r=0x80d35300 (0x47)
*r=47
[PEXPECT\PROMPT]>./corrupt-pointer-cheri
buf=0x3fffdffd71 [rwRW,0x3fffdffd71-0x3fffdfff70] &p=0x3fffdffd60 [rwRW,0x3fffdf
fd60-0x3fffdffd70]
p.ptr=0x3fffdffe80 [rwRW,0x3fffdffd71-0x3fffdfff70] (0x10f into buf) *p.ptr=0f
q=0x3fffdffe00 [rwRW,0x3fffdffd71-0x3fffdfff70] (0x8f into buf)
*q=8f
r=0x3fffdffe00 [rwRW,0x3fffdffd71-0x3fffdfff70] (invalid) (0x8f)
In-address space security exception
[PEXPECT\PROMPT]>
```

Writing to pointer and writing to a data array are both valid on the baseline

Write to capability with a capability operation works fine

Data write strips validity tag

Dereferencing an invalid capability leads to an exception

Buffer Overflow (1/3)

```
char upper[0x10];
char lower[0x10];

printf("upper = %p, lower = %p, diff = %zx\n",
  upper, lower, (size_t)(upper - lower));

upper[0] = 'a';
printf("upper[0] = %c\n", upper[0]);

lower[sizeof(lower)] = 'b';
printf("upper[0] = %c\n", upper[0]);
```

Declare two buffers at consecutive addresses

Print memory layout of buffers

Write to buffer *upper* and print status

Out-of-bounds write to *lower* and print status of *upper* to see whether buffer overflow worked

Buffer Overflow (2/3)

```
[PEXPECT\PROMPT]>./buffer-overflow-stack-baseline
upper = 0x80c7b6c0, lower = 0x80c7b6b0, diff = 10
upper[0] = a
upper[0] = b
[PEXPECT\PROMPT]>./buffer-overflow-stack-cheri
upper = 0x3fffdfff50, lower = 0x3fffdfff40, diff = 10
upper[0] = a
In-address space security exception
[PEXPECT\PROMPT]>
```

Buffer overflow successful due to missing bounds information on baseline

upper and *lower* are guarded by capabilities limiting each buffer to 16 byte bounds

Out-of-bounds write leads to CHERI exception

Buffer Overflow (3/3)

```
<setup_cap>:
cincoffset      ca0, csp, 48
csetbounds      cs0, ca0, 16

cincoffset      ca0, csp, 64
csetbounds      cs1, ca0, 16

…

<write_to_lower>:
addi            a1, zero, 16
cincoffset      cs0, cs0, a1

addi            a1, zero, 98

csb             a1, 0(cs0)
```

Create capability for buffer *lower* and set 16 byte bounds

Create capability for buffer *upper* at consecutive addresses and set 16 byte boundaries

Move capability address out of bounds because a1 = 16

Store character 'b' in a1

Store byte to memory; CHERI exception due to address out-of-bounds

CHERI C/C++: pointer provenance validity (1/2)

- An integer data type cast to a pointer data type results in a NULL-derived capability without a tag;
- However, there are data types that can hold pointer or integer values (e.g., uintptr_t).
- In the CHERI memory protection model, capabilities are derived from a single other capability;
- In CHERI C/C++, a capability can be a result of a complex expression with multiple data types and casts.

CHERI C/C++ Programming Guide, Section 4.2, 4.2.1, and 4.2.3
(https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-947.pdf)

CHERI C/C++: pointer provenance validity (2/2)

- Ideally, we would like to recompile source code for CheriABI and automatically gain security;
- Unfortunately, there is a lot of software that use incorrect data types to hold values that fit in them but have different semantics.

CHERI LLVM: CHERI compiler warnings and errors

CHERI LLVM can identify capability-related issues and print warnings:

- Loss of provenance (-Wcheri-capability-misuse);
- Ambiguous provenance (-Wcheri-provenance);
- Underaligned capabilities of packed structures (-Wcheri-capability-misuse);
- Underaligned load of capability type (-Wcheri-inefficient).

CHERI C/C++ Programming Guide, Chapter 6
(https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-947.pdf)

Example broken cat program

We modified the cat(1) program from CheriBSD/FreeBSD to introduce two bugs:

1. Loss of provenance.
2. Provenance-free integer type to pointer type cast.

CHERI LLVM: CHERI compiler warnings and errors

```
faf28@choisi:/local/scratch/faf28/demo/cheri-exercises/src/exercises/adapt-c$ cl
ang -g -O2 -target riscv64-unknown-freebsd --sysroot=/local/scratch/faf28/cheri/
output/sdk/sysroot-riscv64-purecap/ -fuse-ld=lld -mno-relax -march=rv64gcxcheri
-mabi=l64pc128d -Wall -Wcheri cat/methods.c cat/cat.c -o cat-cheri
cat/methods.c:70:43: warning: binary expression on capability types 'ptroff_t' (
aka 'unsigned __intcap') and 'uintptr_t' (aka 'unsigned __intcap'); it is not cl
ear which should be used as the source of provenance; currently provenance is in
herited from the left-hand side [-Wcheri-provenance]
        return (write(fildes, (const void *)(off + (uintptr_t)buf), nbyte));
                                             ~~~ ^ ~~~~~~~~~~~~~~

cat/methods.c:80:7: warning: cast from provenance-free integer type to pointer t
ype will give pointer that can not be dereferenced [-Wcheri-capability-misuse]
        fp = (FILE *)file;
             ^

2 warnings generated.
faf28@choisi:/local/scratch/faf28/demo/cheri-exercises/src/exercises/adapt-c$ ex
it
exit
```

Let's try to compile cat!

Potential loss of provenance: we have given the compiler multiple choices for a source of provenance and the compiler might have picked the wrong one

We are trying to case a non-pointer type to a pointer type; we need a valid capability as a source of provenance

Conclusions

- CHERI HW/SW stack is fully working
- CHERI enforces intentionality
- Provenance validity and bounds checking lead to strong spatial memory safety that eliminates security bugs
- CHERI LLVM helps a developer to adapt a C/C++ program to CHERI C/C++
- Source code changes are needed where source of provenance is not clear, where non-pointer to pointer casting is done, and where alignments enforced by the developer are incorrect

WRAP UP

Wrap Up

- CHERI is a stable platform:
  - Demonstrated on RISC-V and Arm (previously on MIPS)
  - Full software stack (compiler, linker, OSs, etc.)
  - Formal verification of key security properties
- Our aim:
  - CHERI on every platform with no IP restrictions
  - Looking to ratify CHERI as an official RISC-V extension

Project website: https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/