What is CHERI?

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

Hesham Almatary, Jonathan Anderson, Alasdair Armstrong, Peter Blandford-Baker, John Baldwin, Hadrien Barrel, Thomas Bauereiss, Ruslan Bukin, David Chisnall, Jessica Clarke, Nirav Dave, 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

FreeBSD Developers Summit
June 16, 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.

Introducing: 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
  - Arm Morello announced in 2019, supported by UKRI
- **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?
  - CHERI and FreeBSD – the way forward?

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.

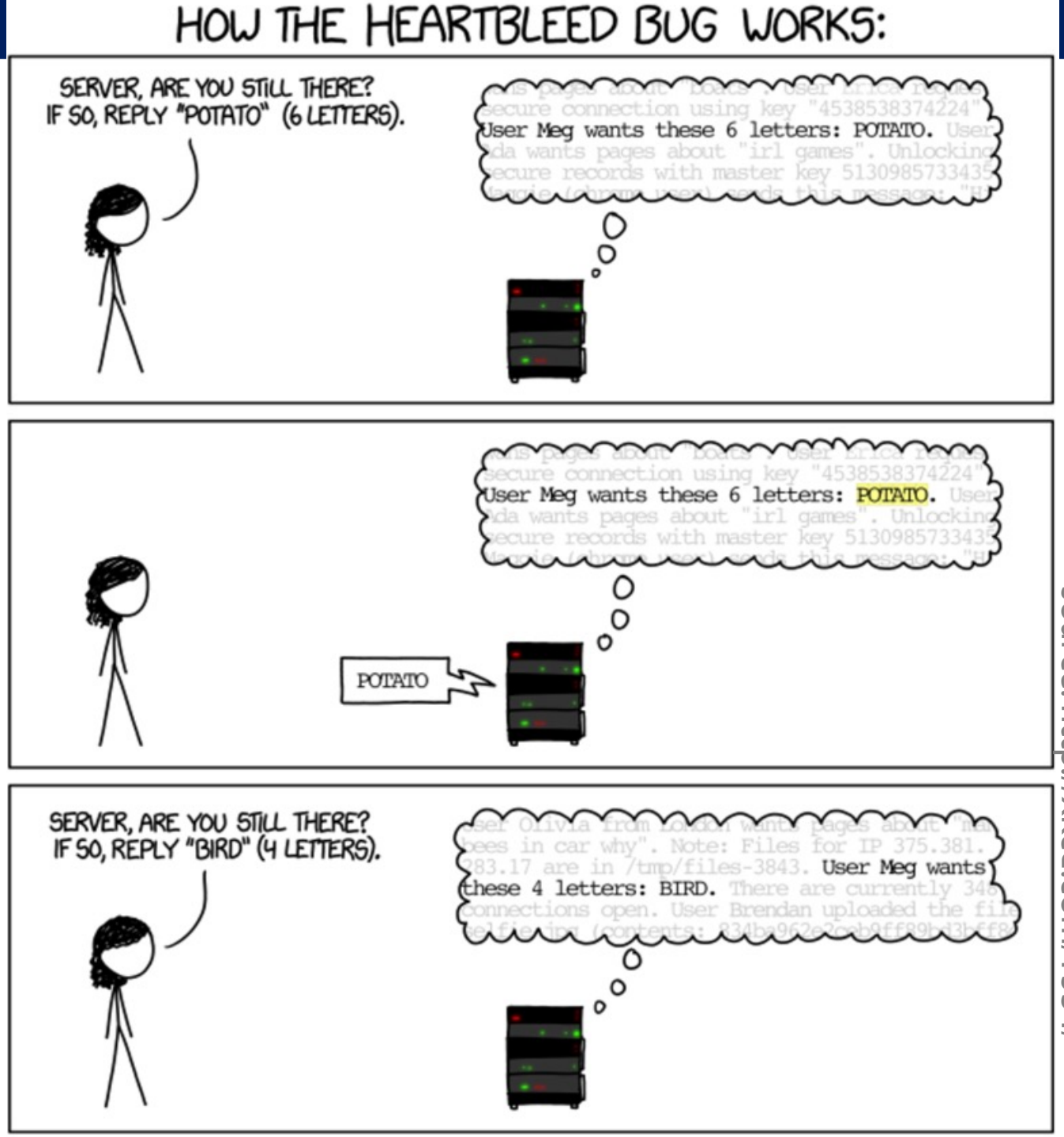
The impact of buffer overflows have if anything gone up."

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

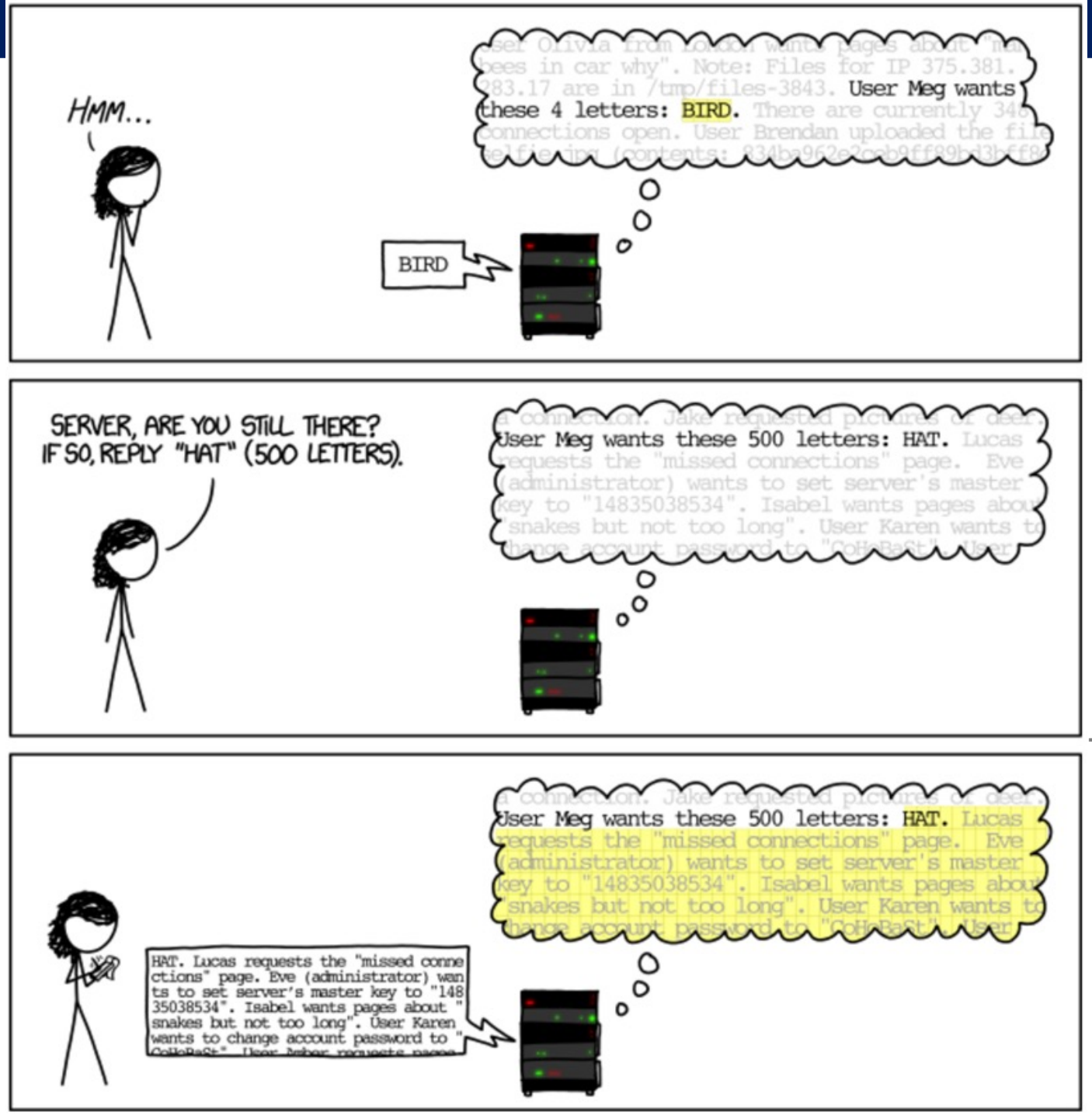
Example 1

HeartBleed



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

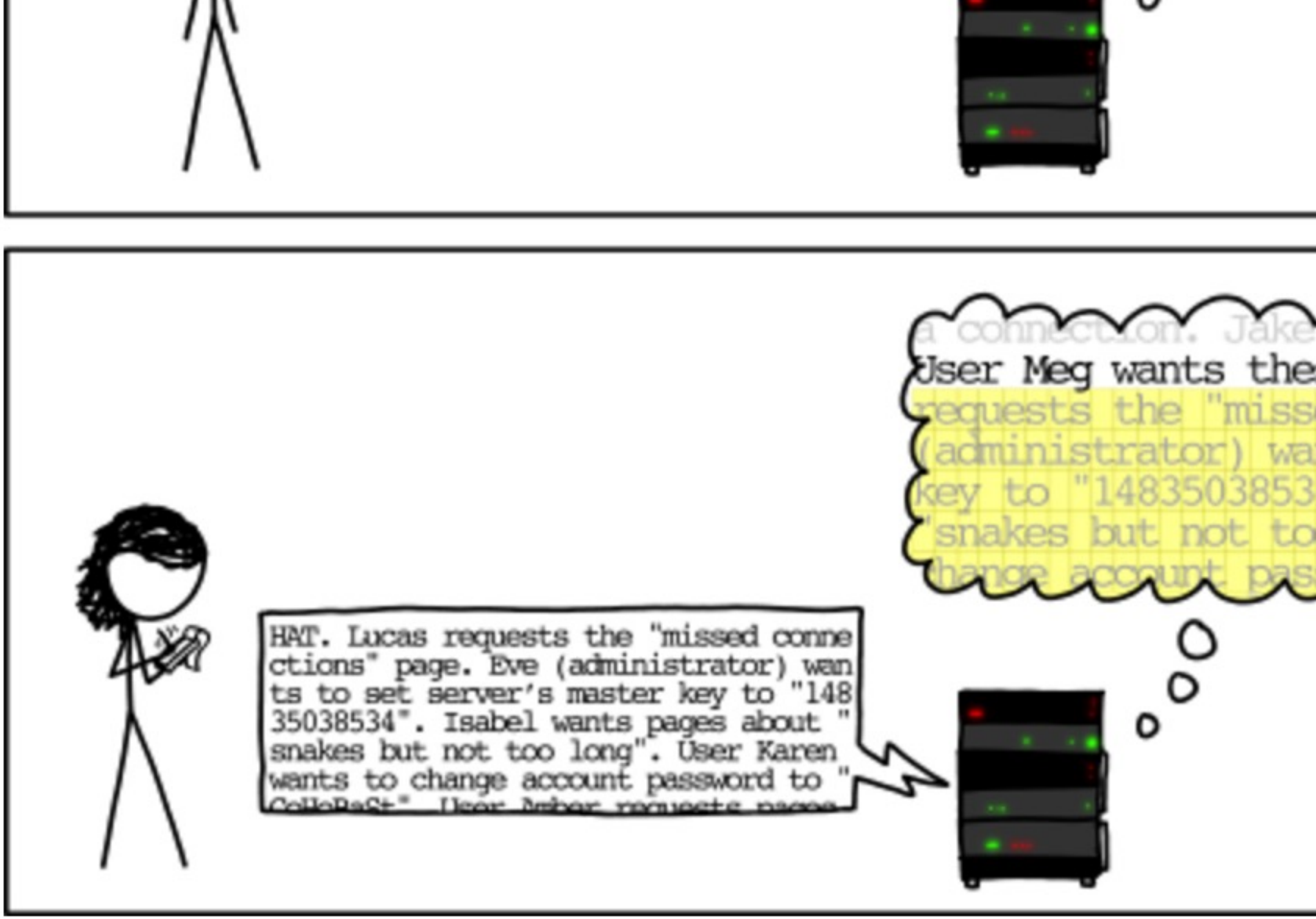
HeartBleed



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?

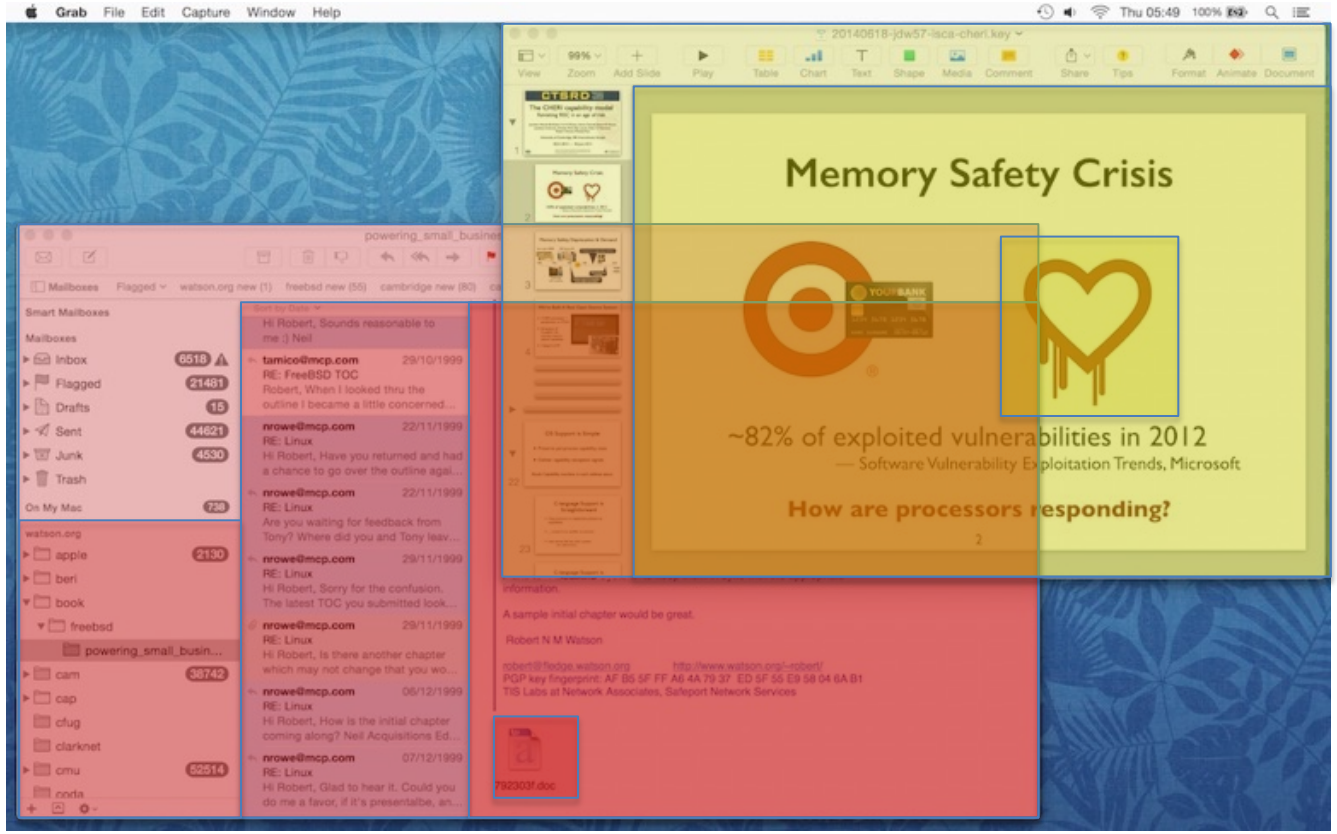
- Classical answer:
  - The programmer forgot to check the bounds of the data structure being read
  - Fix the vulnerability in hindsight – one-line fix:
    if (1+2+payload+16 **>** s->s3->rrec.length) **return** 0;
- Our answer:
  - Preserve bounds information during compilation
  - Use hardware (CHERI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

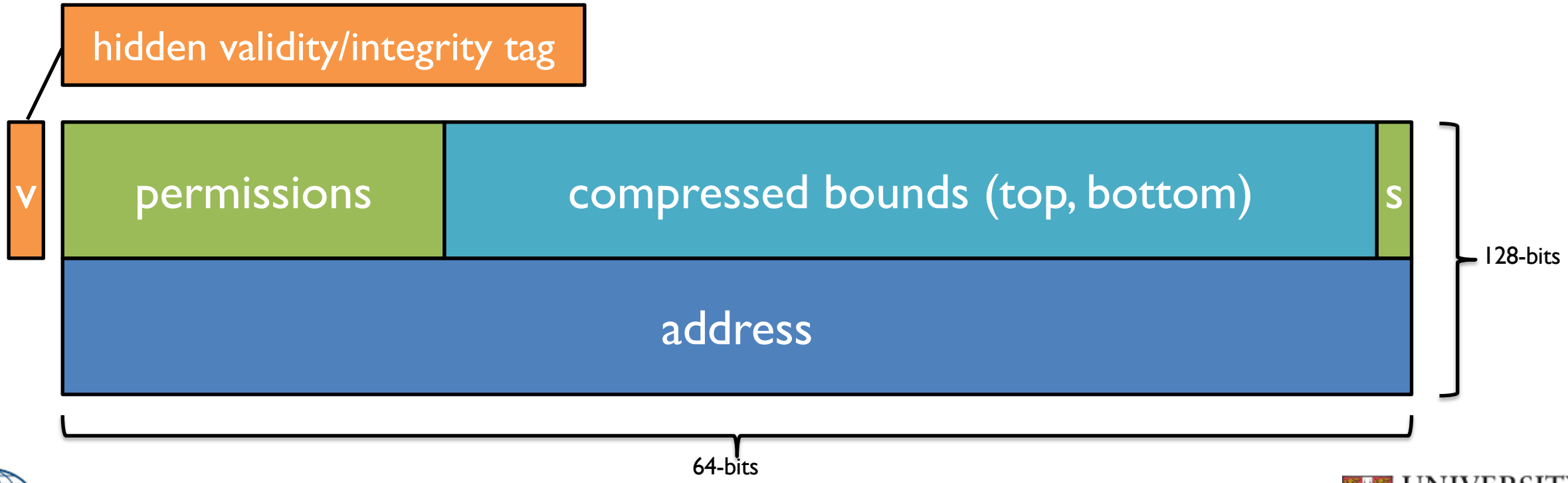
UNIVERSITY OF CAMBRIDGE

Principles CHERI helps to uphold

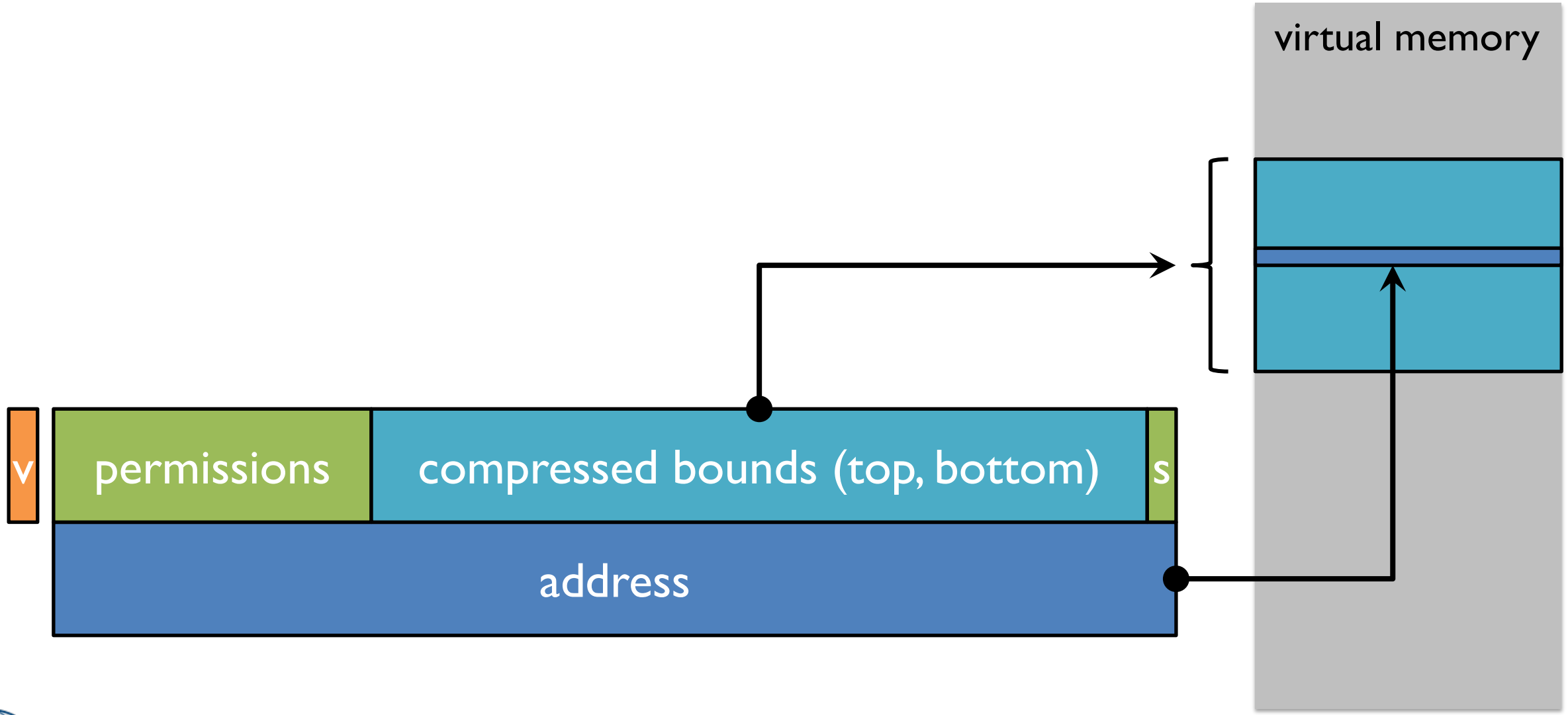
- The **principle of intentional use**
  - Ensure that software runs the way the programmer intended, not the way the attacker tricked it
  - Approach: guaranteed pointer integrity & provenance, with efficient dynamic bounds checking
- The **principle of least privilege**
  - Reduce the attack surface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

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

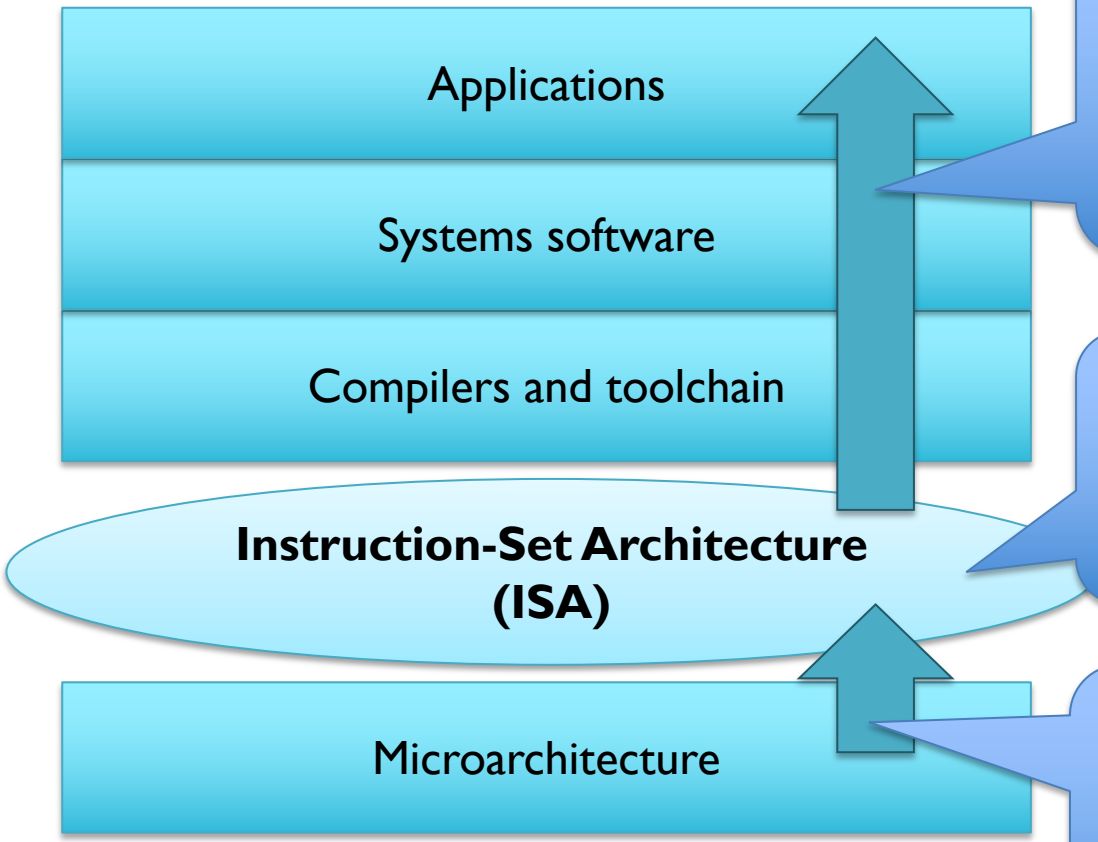
- CHERI Capability = bounds checked pointer with integrity
- Held in memory and in extended registers



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**

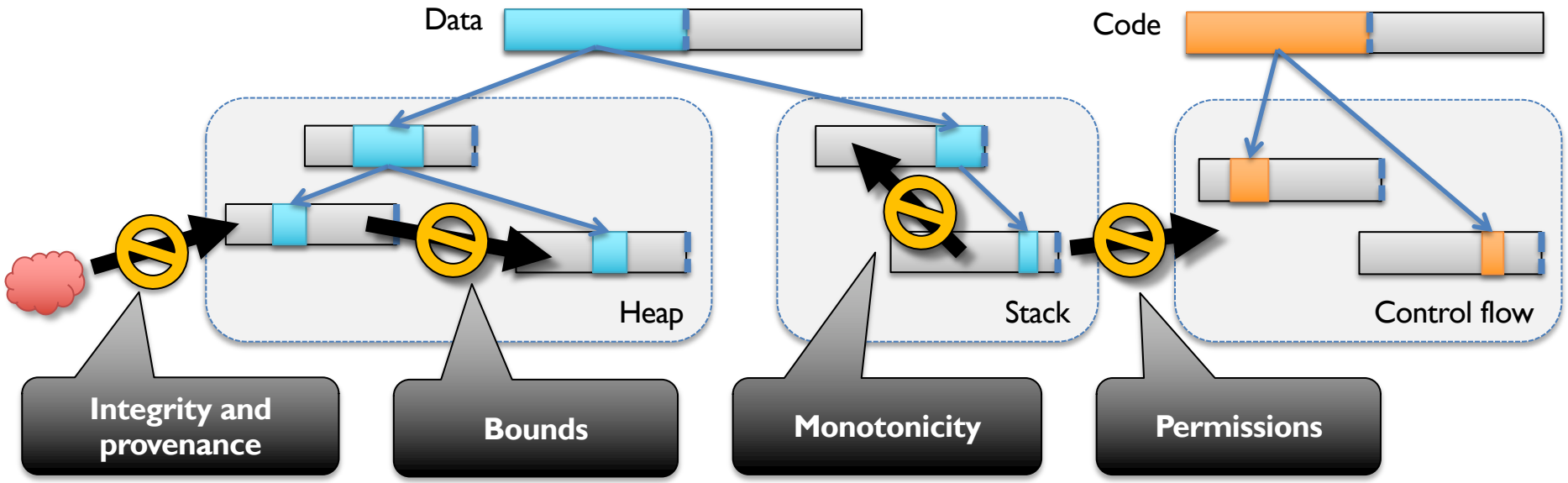
SRI International

UNIVERSITY OF CAMBRIDGE

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)

Summary of Capability Memory Protection



Valid userspace pointer set – pointers not generated using derivation rules are not part of the valid provenance tree and will not be dereferenceable

Pointer privilege reduction – capabilities allow pointers to carry specific privileges, which can be minimized with OS, compiler, and linker support

Foundation for higher-level models such as **software compartmentalization**

Compartmentalization scalability

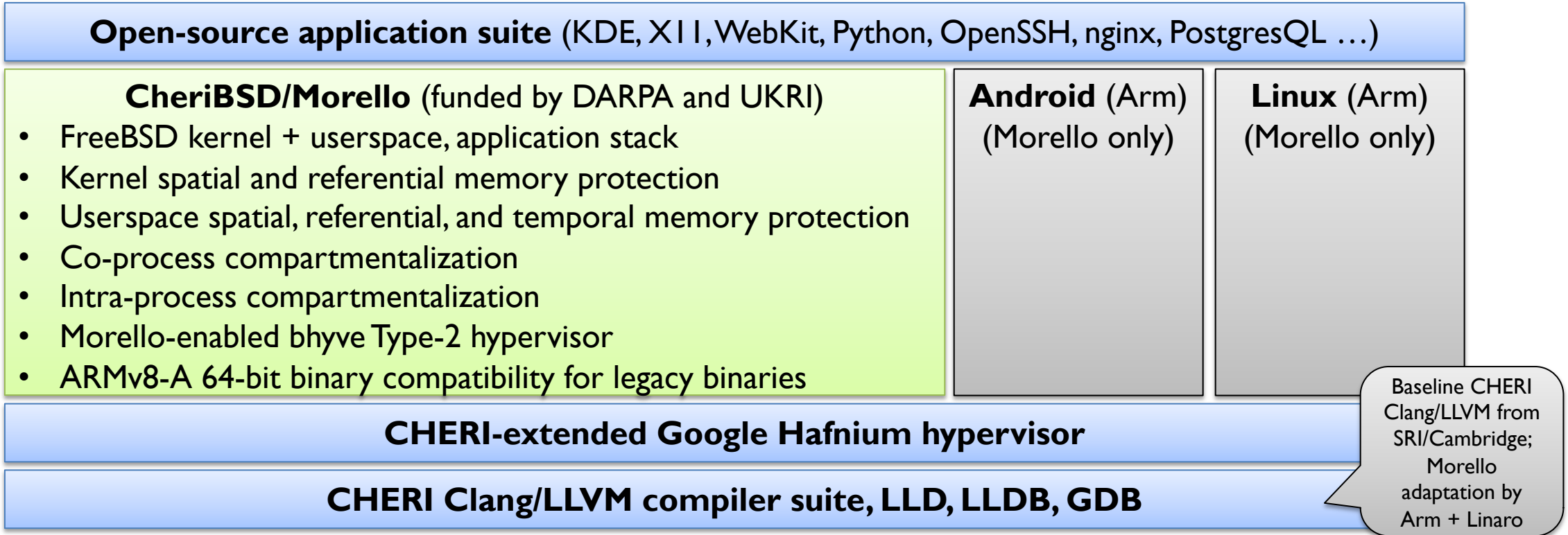
- CHERI dramatically improves **compartmentalization scalability**
  - More compartments
  - More frequent and faster domain transitions
  - Faster shared memory between compartments

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

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

CHERI prototype software stack

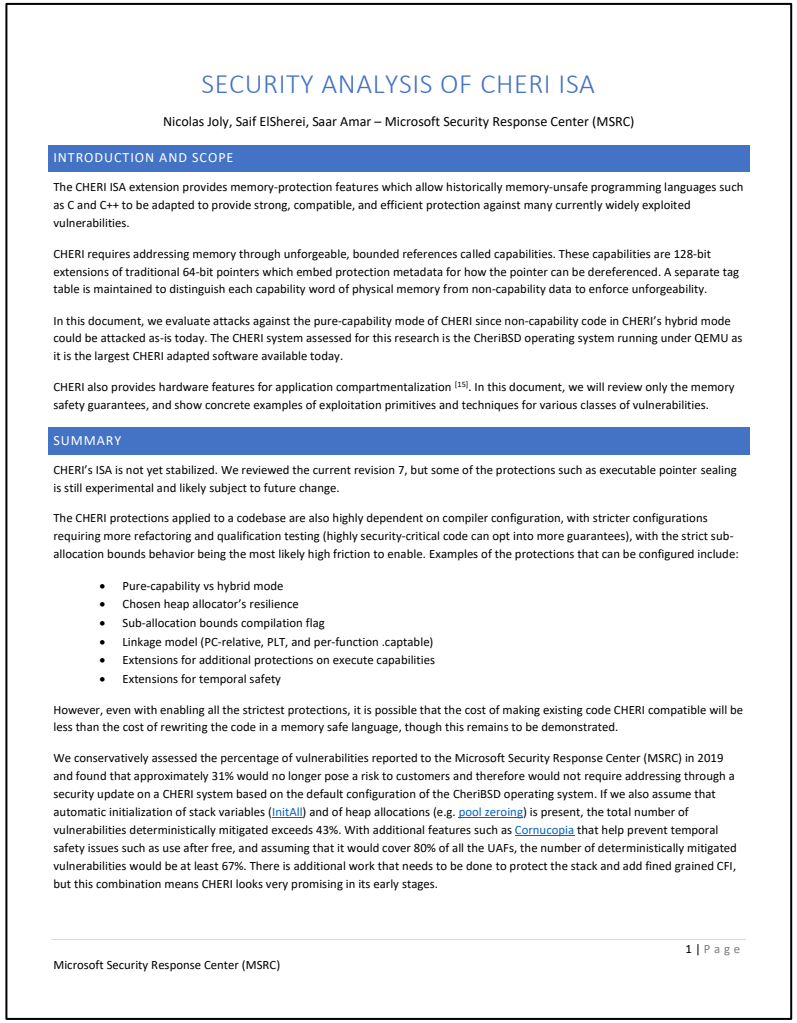
- **Complete open-source software stack** from bare metal up: compilers, toolchain, debuggers, hypervisor, OS, applications – all demonstrating CHERI
  - Rich CHERI feature use, but fundamentally incremental/hybridized deployment
  - Aim: Mature and highly useful research and development platform for Morello



CheriBSD

- FreeBSD adapted for CHERI
- New ABI: CheriABI
  - All pointers are CHERI capabilities (pure-capability)
- New compat ABI: freebsd64
  - Supports conventional FreeBSD and hybrid (some pointers are capabilities) programs
- Kernel can be hybrid or pure-capability
  - Large diff, due to hybrid support (all userspace pointers annotated)
- Panfrost GPU support, drm, etc working on Ruslan's desk
- Andy has bhyve for Morello patches

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 Response 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

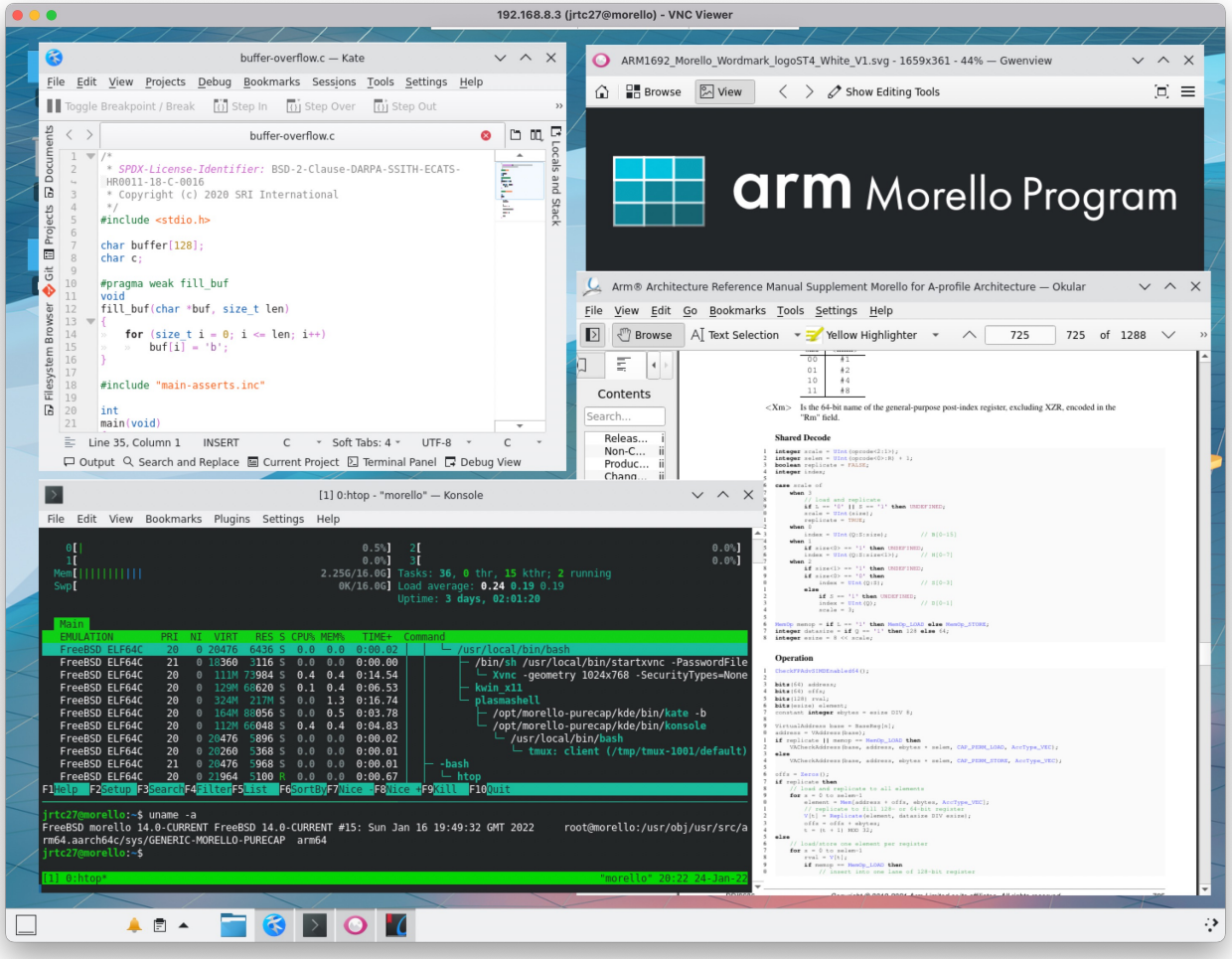
3-month CHERI Desktop UKRI pilot study

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

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

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

CHERI desktop ecosystem study: Key outcomes



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

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:**
  - https://cheri-cpu.org/
  - https://cheribsd.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

arm
Morello Program

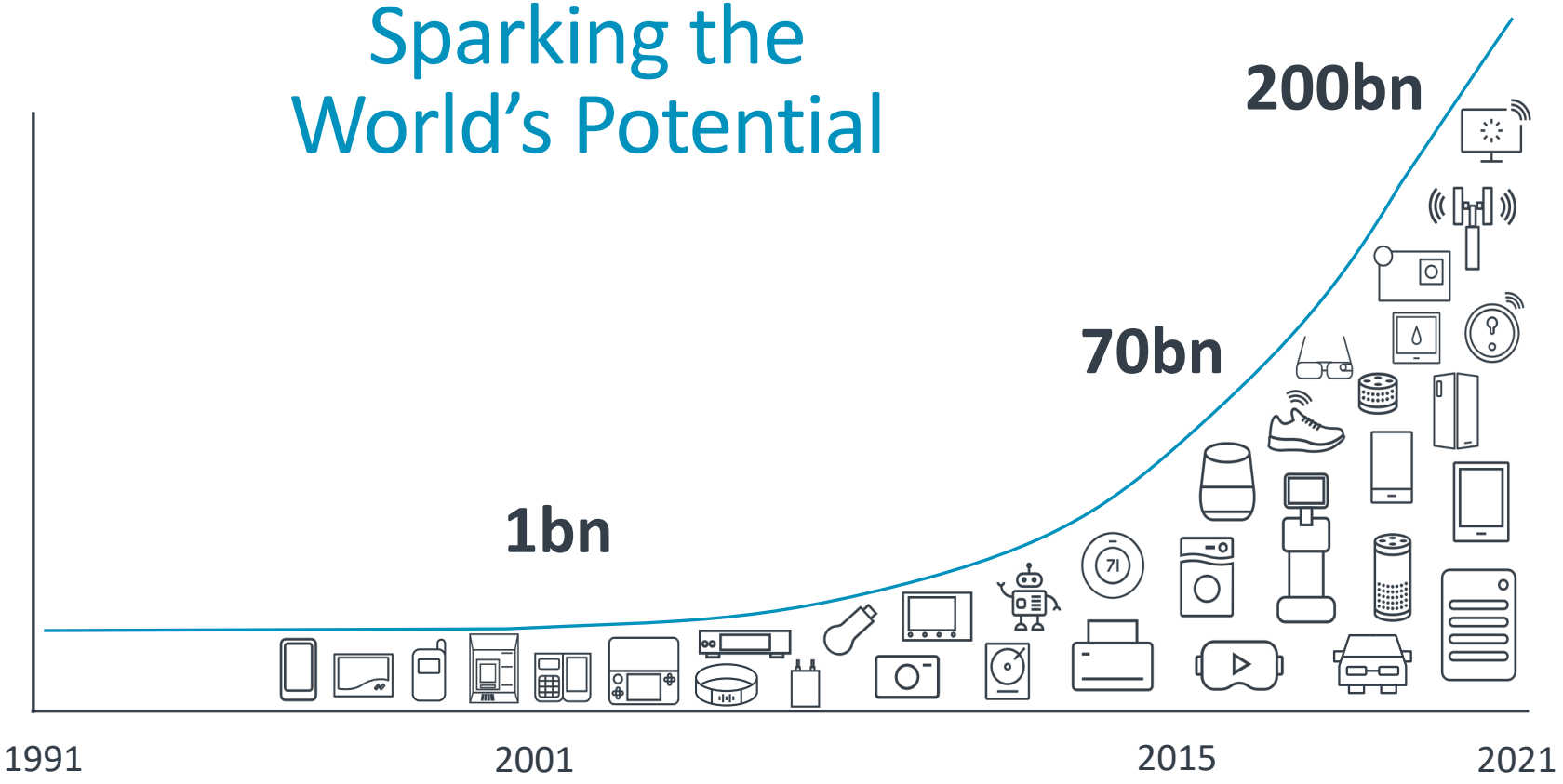
Digital Security by Design – Morello Program

Led by Arm Ltd from their HQ in the UK

UK Research and Innovation

Arm is Everywhere Compute Happens

Sparking the World's Potential



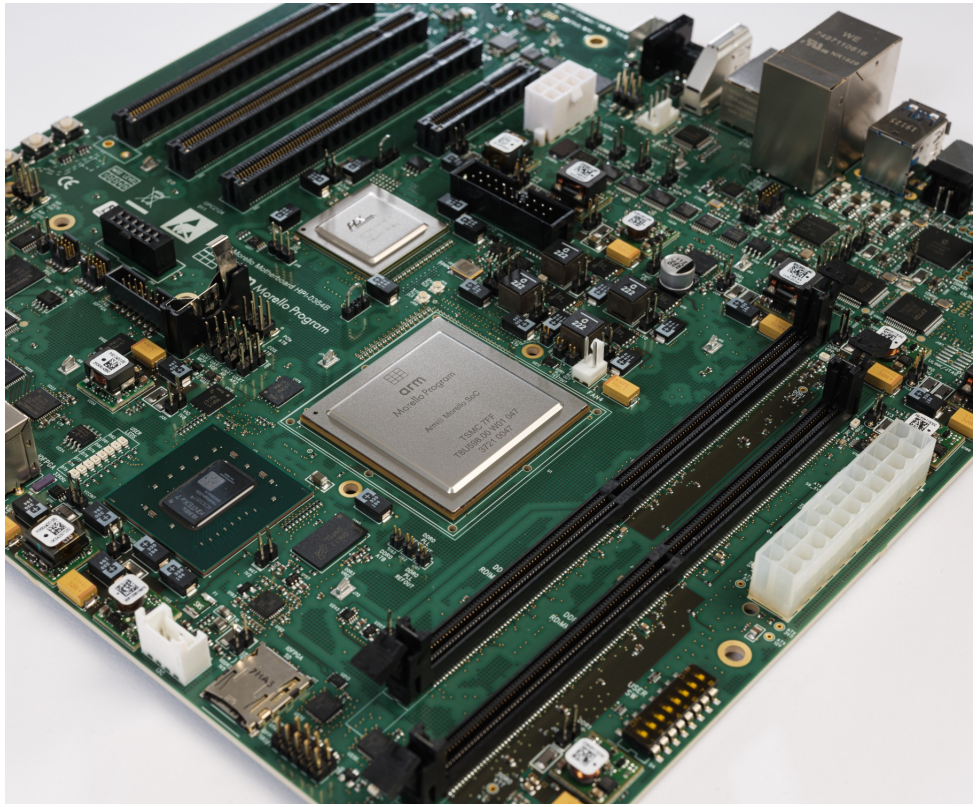
2000+
Active licenses, growing by 100+ every year

600+ licensees
Industry leaders and high-growth start-ups; chip companies and OEMs

200bn+
Arm-based chips shipped to-date

25bn
Arm-based chips shipped in FY 2020 (April 2020-Mar 2021)

Morello Demonstrator Board



What Do We Want To Achieve?

- Security vulnerabilities are a huge problem – CHERI has a lot of promise in solving them
- Arm has very interested partners in CHERI – but we need to do a lot of investigation
- Morello is a prototype architecture that allows the experimentation we need to get feedback from

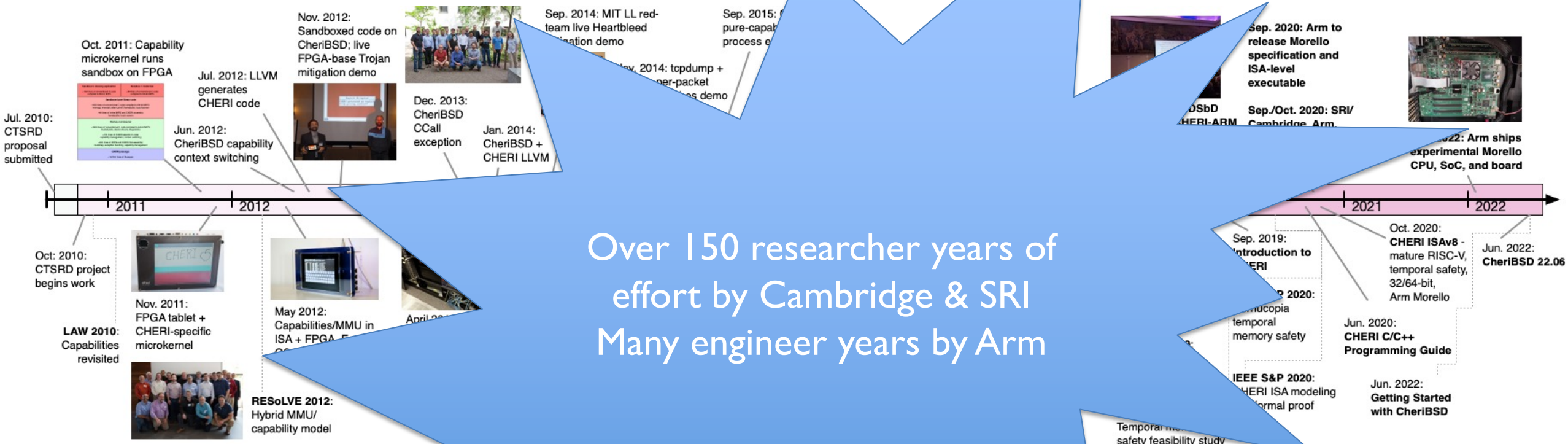
Success looks like:

- People find compelling uses of the Morello prototype in addressing security challenges
- This leads to CHERI technologies being deployed in Arm processors, and potentially other architectures
  - The CHERI technology is not patented, and any architecture can use it

Further details:

- https://www.arm.com/architecture/cpu/morello

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

FreeBSD upstreaming

- CheriBSD tracks FreeBSD main branch
  - We upstream obvious improvements to FreeBSD
- Should we upstream CHERI support?
  - Would show a community commitment
  - Allow CI to cover CHERI-specific issues
  - Lack of a non-prototype architecture is a drawback

FreeBSD ports upstreaming

- CheriBSD-ports will track FreeBSD main
- We build hybrid packages with LOCALBASE=/usr/local64
  - >10K broken/blocked packages
- What sort of ports changes would be acceptable?
  - LOCALBASE!=/usr/local?
  - Excess depends with DOCS disabled?

Implications for the FreeBSD Project

- CheriBSD is the only practical development environment for CHERI software
  - Integrated base system and use of clang were critical!
  - Ports has allowed us to build ~8k packages for CheriABI
    - We hope to build many more (aiming for 100MLoC in the next 18 months)
- How can we maintain and exploit this lead?

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**
- FreeBSD plays a major role in the success of CHERI. Let's keep it going!