CTSRD

CRASH-WORTHY TRUSTWORTHY SYSTEMS RESEARCH AND DEVELOPMENT

Winning the war in memory using CHERI capabilities

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

Hesham Almatary, Jonathan Anderson, John Baldwin, Hadrien Barrel, Ruslan Bukin, David Chisnall, 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, Kyndylan Nienhuis, Robert Norton-Wright, Philip Paeps, Lucian Paul-Trifu, Alex Richardson, Michael Roe, Colin Rothwell, 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

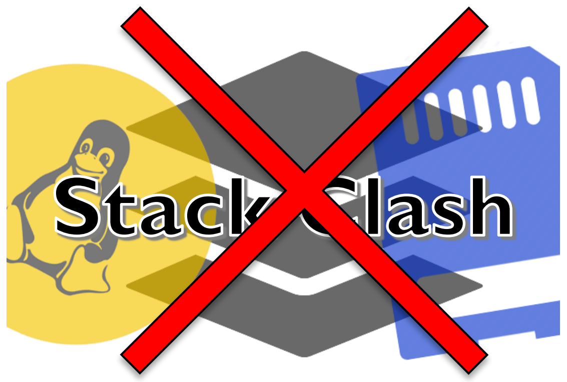
SILM Summer School, Inria, Rennes – 11 July 2019

SRI International

UNIVERSITY OF CAMBRIDGE

Motivation – The Eternal War in Memory*

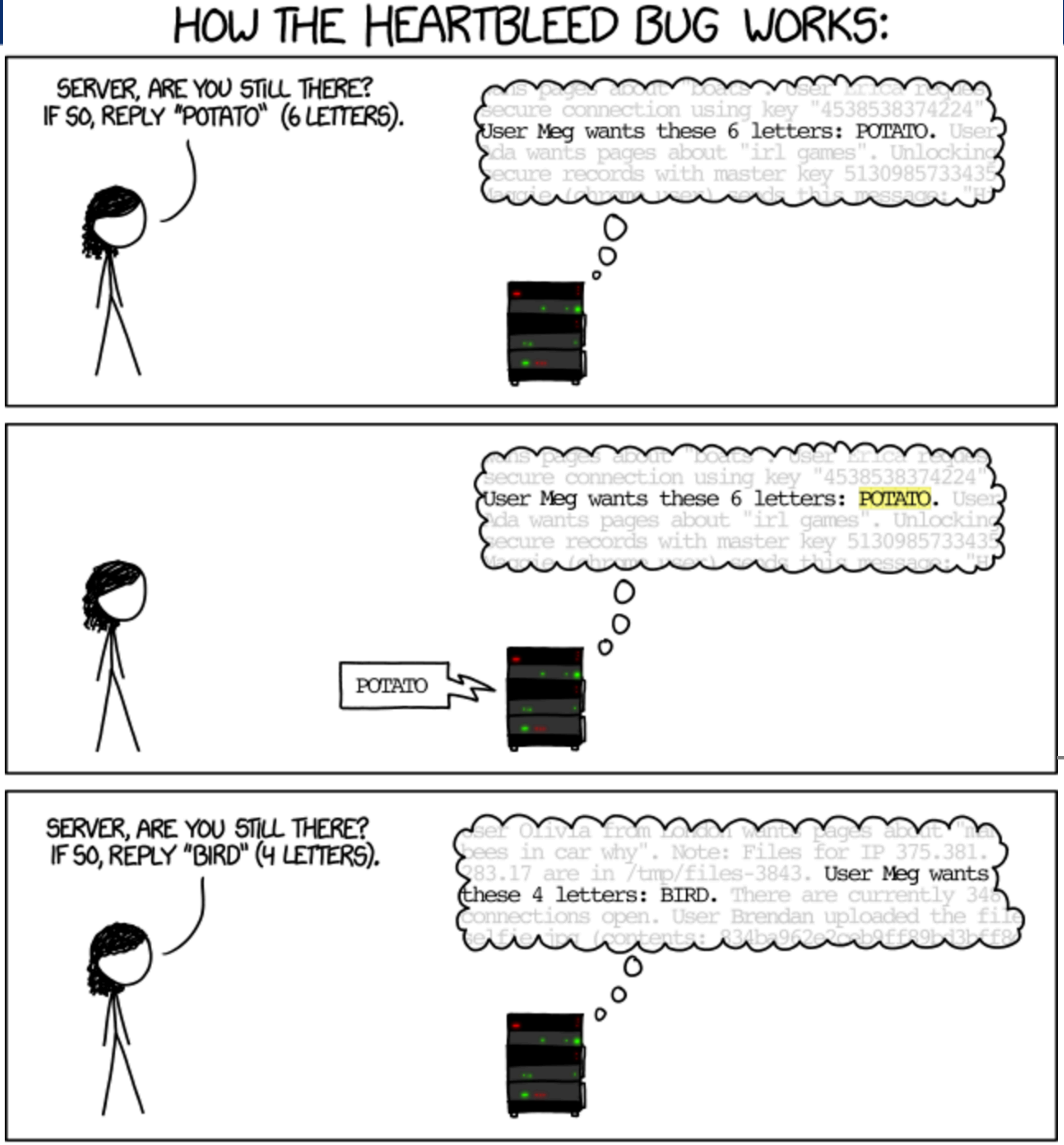
- Many security vulnerabilities exploit memory safety violations



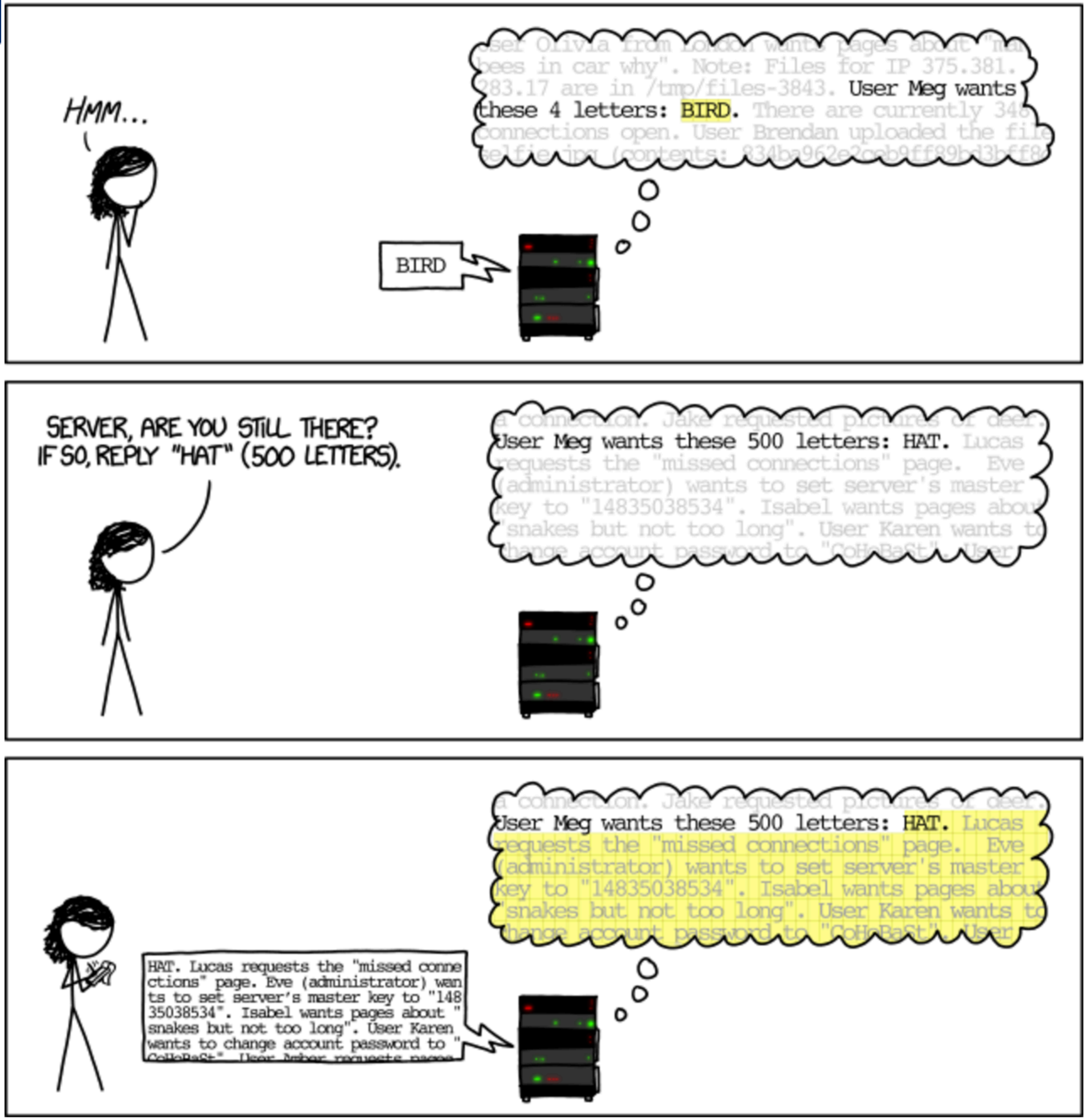
* Title based on Oakland 2013 paper: SoK: Eternal War in Memory, László Szekeres, Mathias Payer, Tao Wei, Dawn Song

Example 1

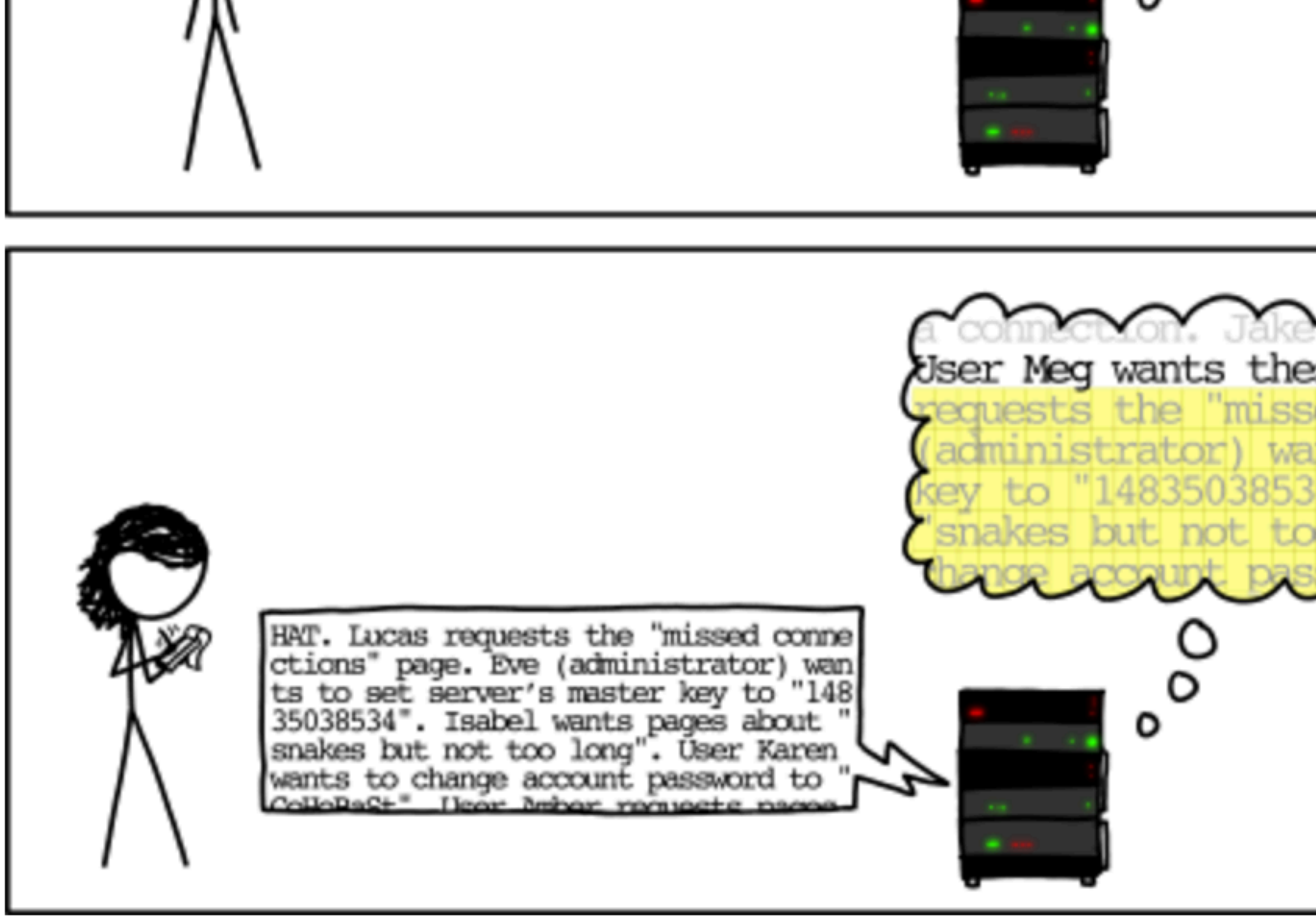
HeartBleed


HOW THE HEARTBLEED BUG WORKS:
SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
...ans pages about "boats". User Erica reques... secure connection using key "4538538374224". User Meg wants these 6 letters: POTATO. User ...da wants pages about "irl games". Unlocking secure records with master key 5130985733435... Meggie (chrome user) sends this message: "H...
POTATO
SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
User Olivia from London wants pages about "ma... bees in car why". Note: Files for IP 375.381. 283.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 34... connections open. User Brendan uploaded the file selfie.jpg (contents: 834ba962e2ceb9ff89bd3bff8...

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


HMM...
User Olivia from London wants pages about "man bees in car why". Note: Files for IP 375.381.283.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 348 connections open. User Brendan uploaded the file selfie.jpg (contents: 834ba962e2ceb9ff89bd3bff8
BIRD
SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User Amber requests pages

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


a connection. Jake
User Meg wants the
requests the "miss
(administrator) wa
key to "1483503853
"snakes but not to
hange account pas
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 " CoHoBaSt". User Amber requests pages

Went wrong? How do we do better?

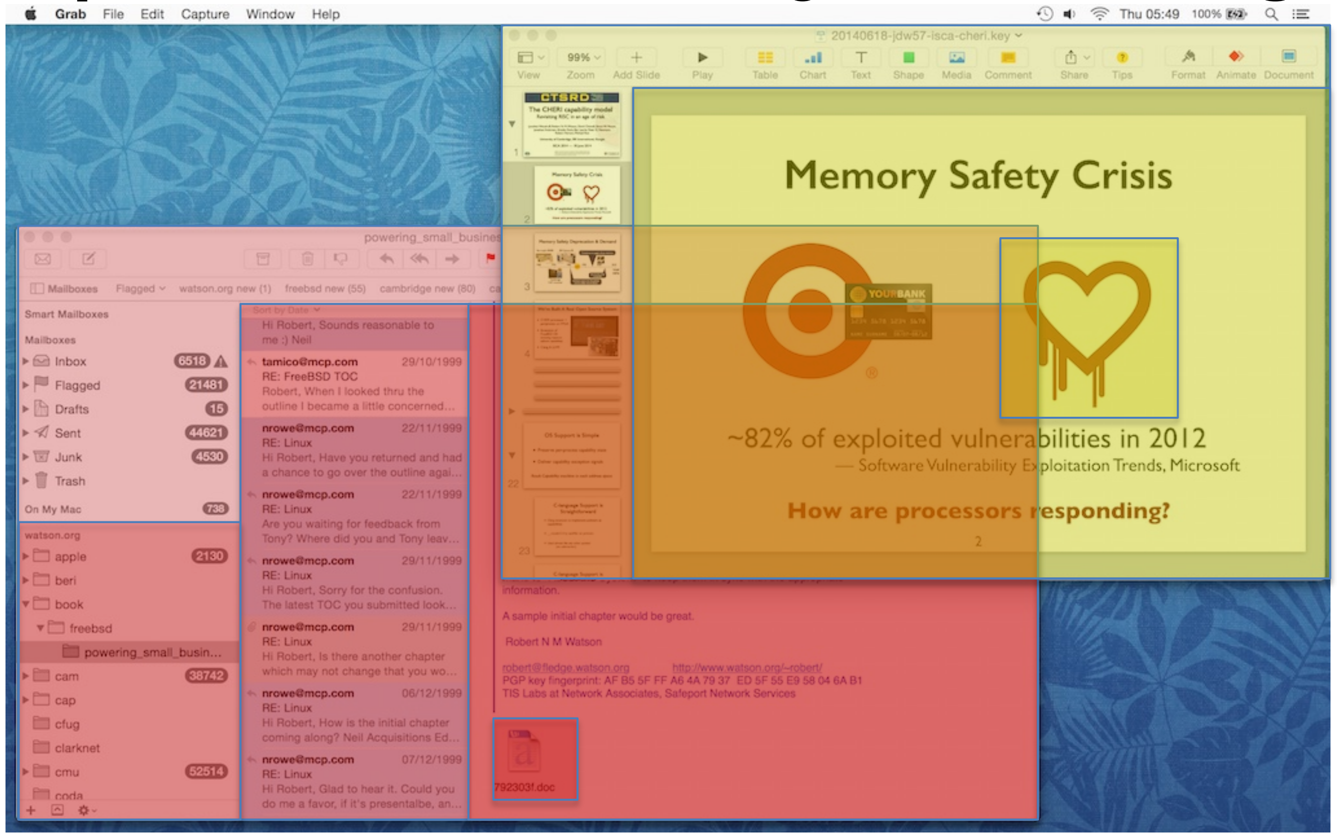
- Classical answer:
  - The programmer forgot to check the bounds of the data structure being read
  - Fix the vulnerability in hindsight – one line fix:
    if (1+2+payload+16 **>** s->s3->rrec.length) **return** 0;
- Our answer:
  - Preserve bounds information during compilation
  - Use hardware (CHERI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

Application-level least privilege

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



Hi Robert, Sounds reasonable to me :) Neil

tamico@mcp.com 29/10/1999
RE: FreeBSD TOC
Robert, When I looked thru the outline I became a little concerned...

nrowe@mcp.com 22/11/1999
RE: Linux
Hi Robert, Have you returned and had a chance to go over the outline agai...

nrowe@mcp.com 22/11/1999
RE: Linux
Are you waiting for feedback from Tony? Where did you and Tony leav...

nrowe@mcp.com 29/11/1999
RE: Linux
Hi Robert, Sorry for the confusion. The latest TOC you submitted look...

nrowe@mcp.com 29/11/1999
RE: Linux
Hi Robert, Is there another chapter which may not change that you wo...

nrowe@mcp.com 06/12/1999
RE: Linux
Hi Robert, How is the initial chapter coming along? Neil Acquisitions Ed...

nrowe@mcp.com 07/12/1999
RE: Linux
Hi Robert, Glad to hear it. Could you do me a favor, if it's presentalbe, an...

information.

A sample initial chapter would be great.

Robert N M Watson

robert@fledge.watson.org http://www.watson.org/~robert/
PGP key fingerprint: AF B5 5F FF A6 4A 79 37 ED 5F 55 E9 58 04 6A B1
TIS Labs at Network Associates, Safeport Network Services

Memory Safety Crisis

~82% of exploited vulnerabilities in 2012
— Software Vulnerability Exploitation Trends, Microsoft

How are processors responding?

2

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

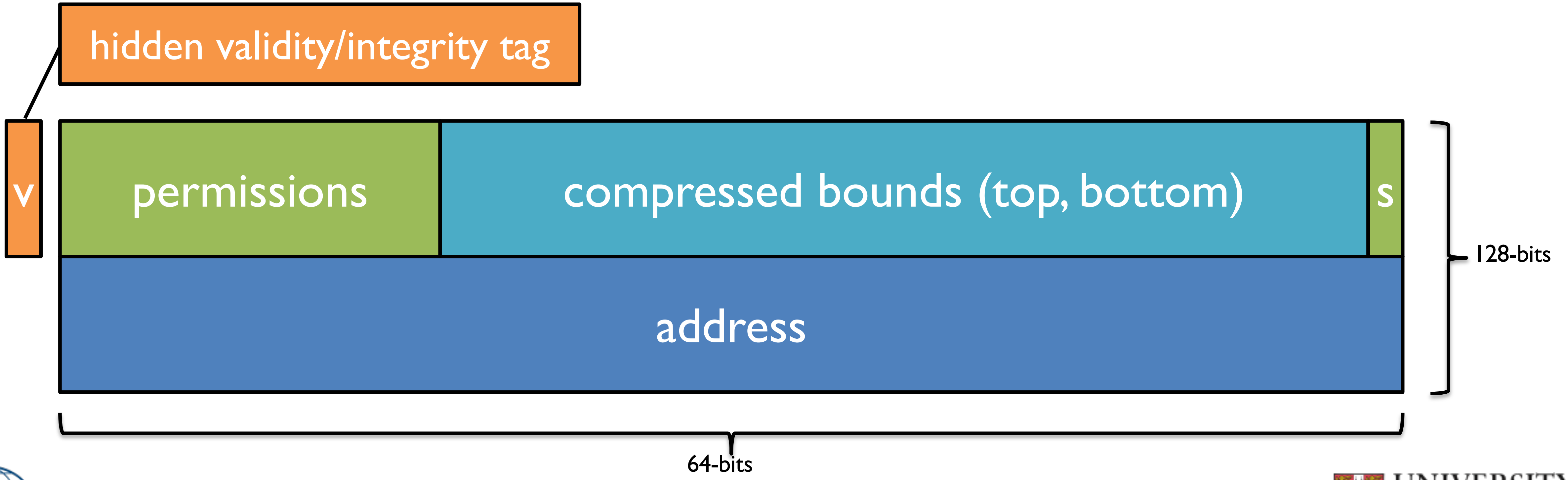
Principles CHERI helps to uphold

- 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

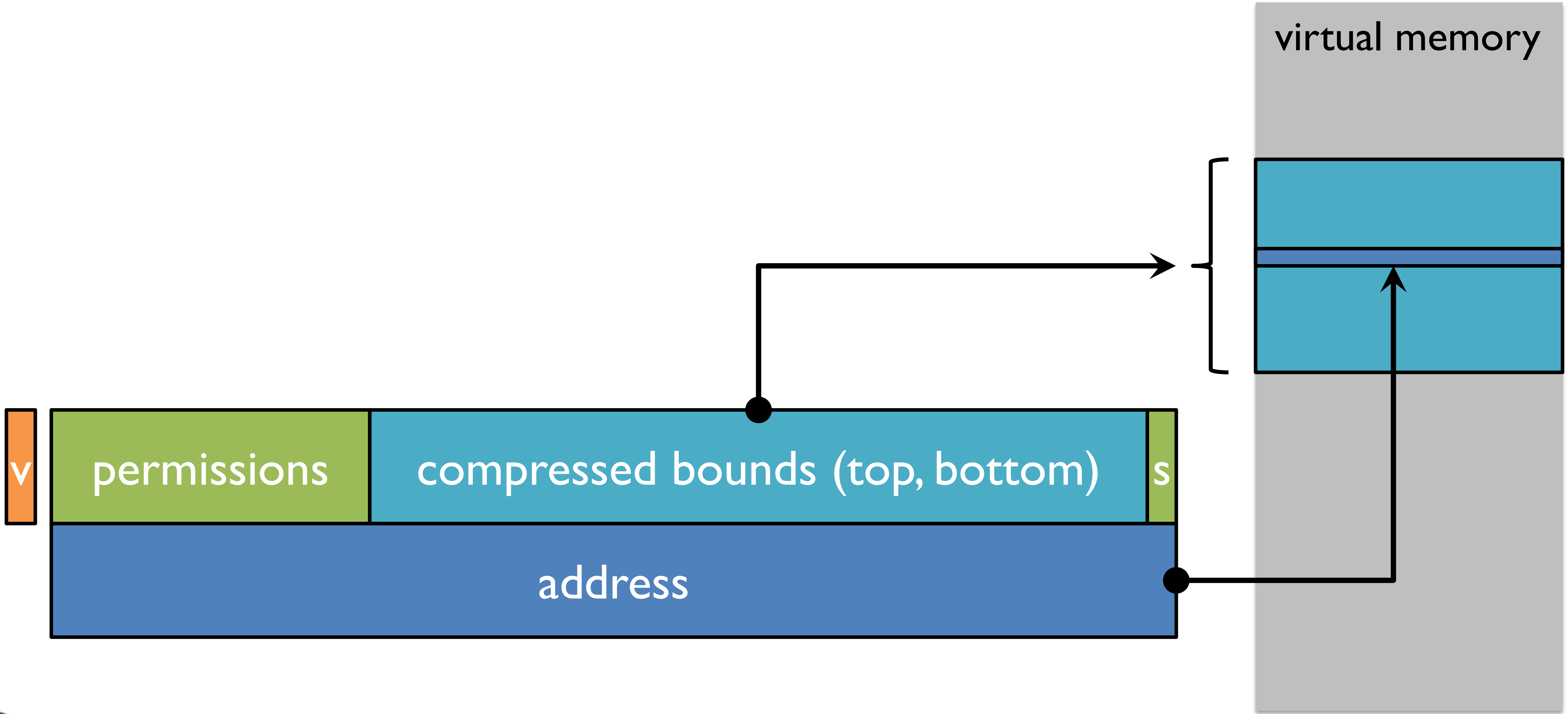
CHERI HARDWARE ARCHITECTURE

A new type – the **Capability**

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


hidden validity/integrity tag
v
permissions
compressed bounds (top, bottom)
s
address
128-bits
64-bits

A new type – the **Capability**



New Instructions

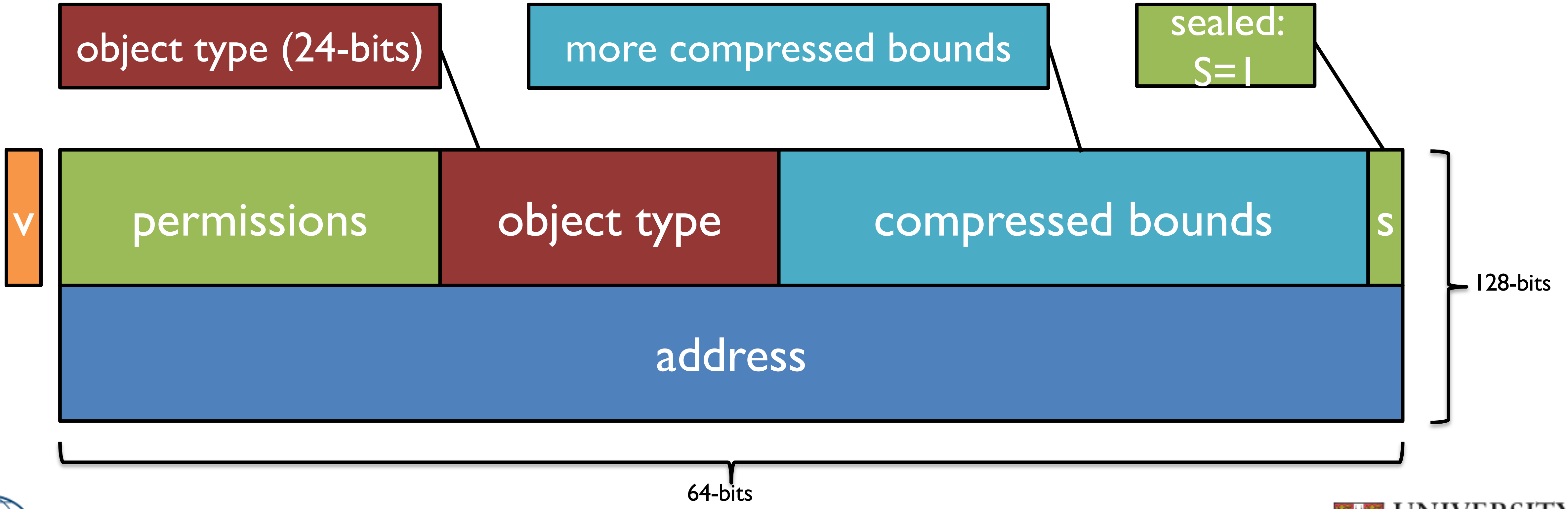
- Memory access
  - Loads and stores via a bounds checked capability
  - Exception if address is out of range
- Guarded manipulation of capabilities
  - Decrease bounds
  - Decrease permissions
  - Adjust the address
  - Extract/test fields

monotonic decrease in rights guaranteed by formally verified hardware

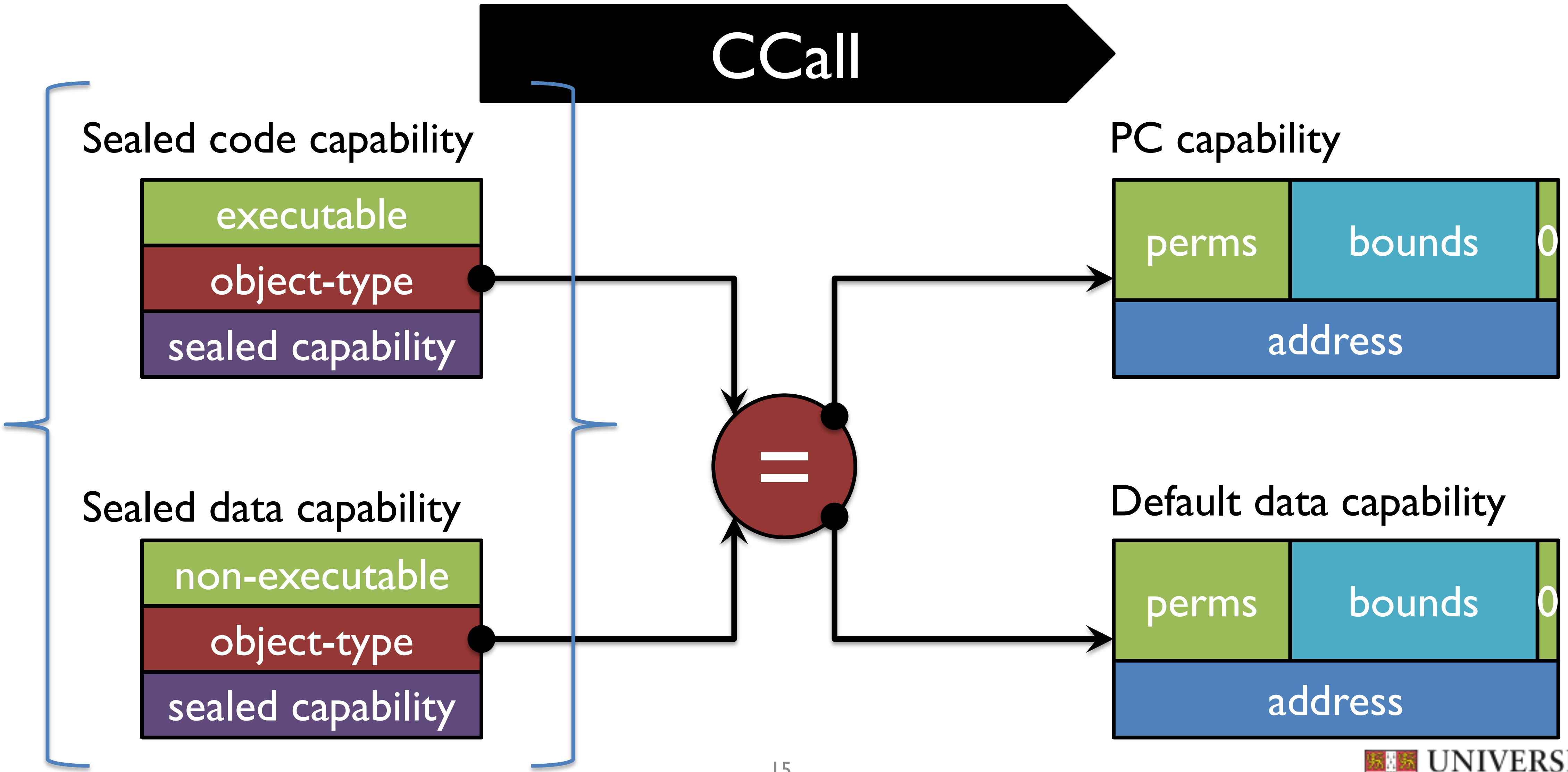
critical property for security

Sealed Capabilities for Compartmentalization

- Sealed capabilities are none dereferencable capabilities
- Have to be unsealed (e.g. inside a compartment) before use



Calling a Compartment

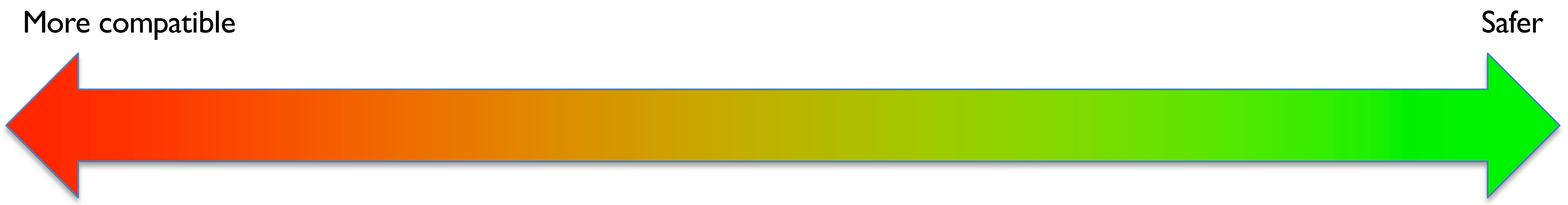


SOFTWARE MODELS

Background to CHERI Software Models

- Machine-level capabilities and instructions provide the building blocks on which new abstract capability software models can be built
- Analogy:
  - Machine-level translation lookaside buffer (TLB) and page table walker enables the OS to represent virtual memory
  - Virtual memory can then be used in different ways to impose new security features, e.g. guard pages

Low-level CHERI software models



Unmodified
All pointers are integers

Hybrid
Annotated and automatically selected pointers are capabilities

Pure-capability
All pointers are capabilities

- **Source and binary compatibility**: **C-language idioms,** multiple **ABIs**
  - **Unmodified code**: Existing code runs without modification
  - **Hybrid code**: E.g., used in return addresses, for annotated data/code pointers, for specific types, stack pointers, etc.

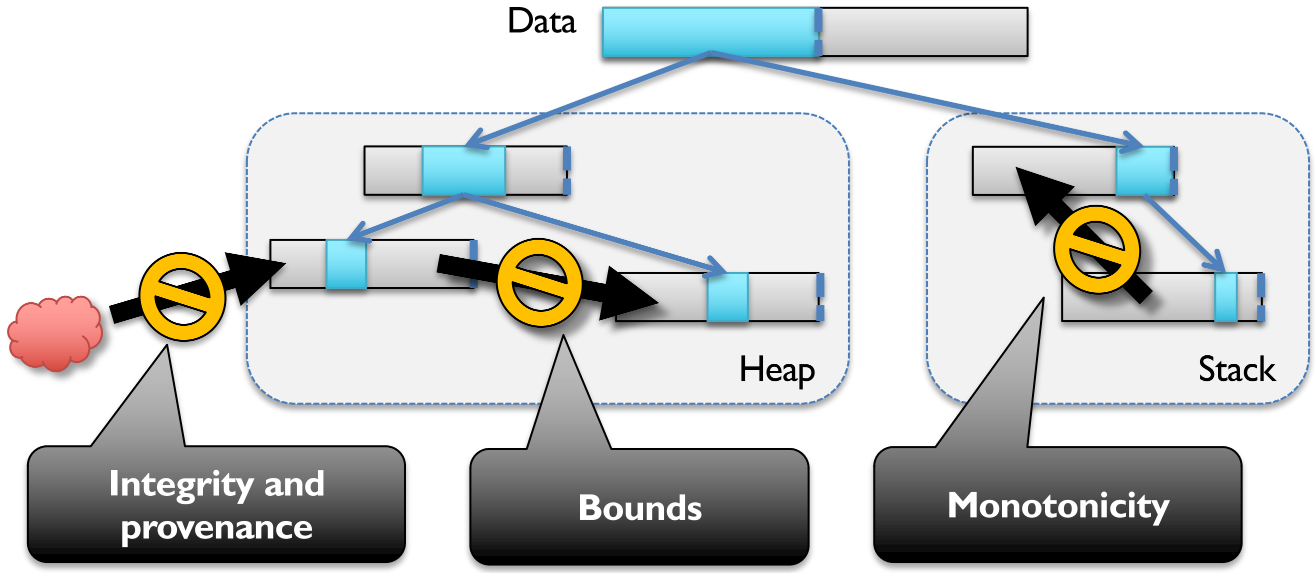
    … But "hybrid" is a spectrum: many different choices for manual and automatic selection of integers vs. capabilities, API and ABI impacts
  - **Pure-capability code**: Ubiquitous data- and data-pointer protection. Not interoperable with legacy code due to changed pointer size.
- **CHERI Clang/LLVM compiler prototype** generates code for all three

Pure Capability Code → Needs CheriABI

- CheriABI
  - Compatibility layer to the OS
  - Allows capabilities to be used in place of pointers
  - A bit like a 32-bit compatibility layer for a 64-bit OS
- Result – we can now recompile large corpuses of C code into a pure capability form with virtually no code changes
- Award winning paper at ASPLOS 2019:
  CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment

Capabilities for Bounds Checking and Integrity

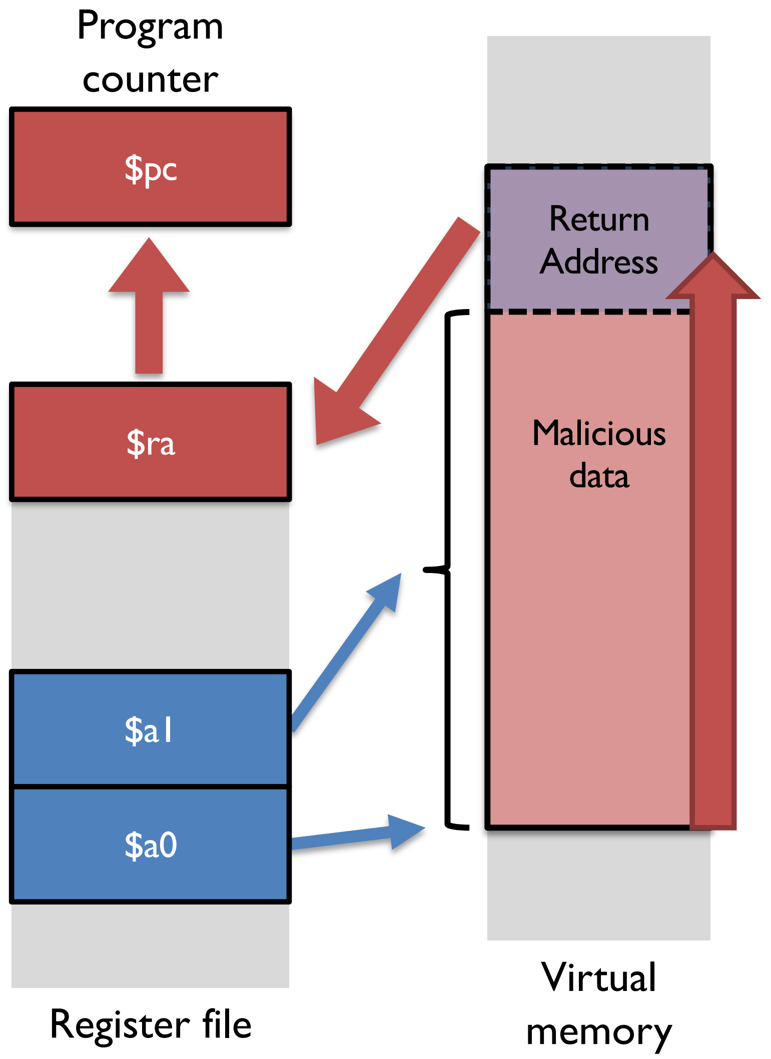
- Pure capability code – all pointers become capabilities
- Compiler + malloc( ) derive bounds for objects
- Strong pointer provenance and integrity properties (validity tag)



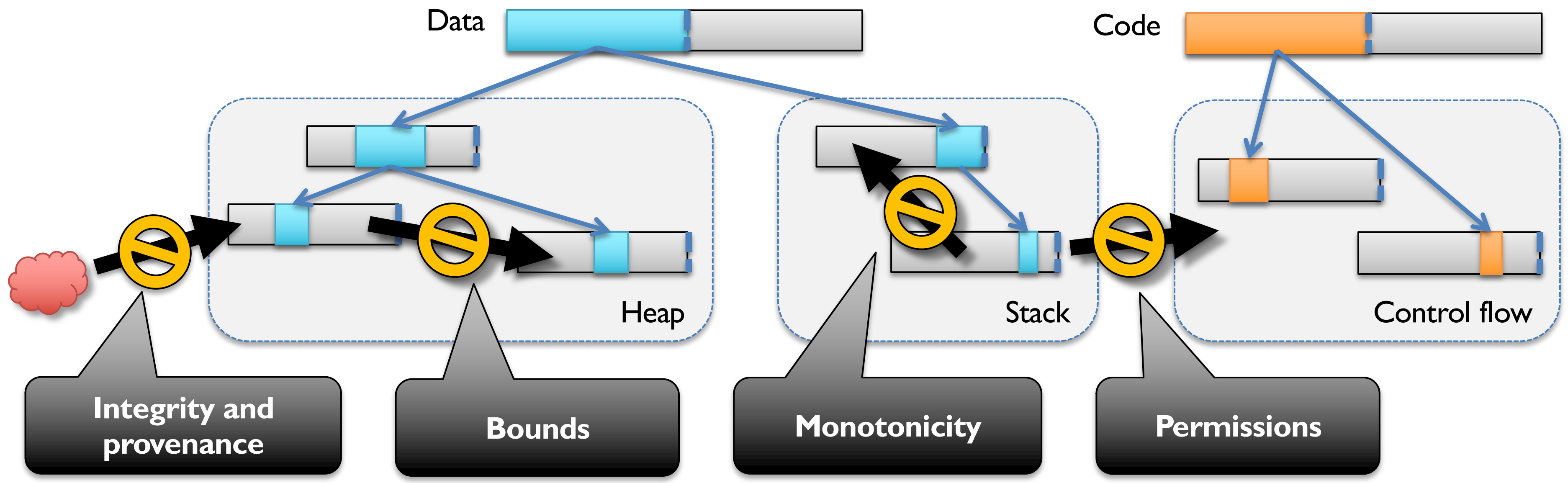
- Mitigates buffer overflow/overread vulnerabilities with no code change!

Capabilities for Control-Flow Robustness

- Capabilities used for return addresses
- Integrity bit mitigates code reuse attacks:
  - ROP – Return Oriented Programming
  - JOP – Jump Oriented Programming
- Much better than current statistical technique ASLR (Address Space Layout Randomisation)



Summary of Capability Protections



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

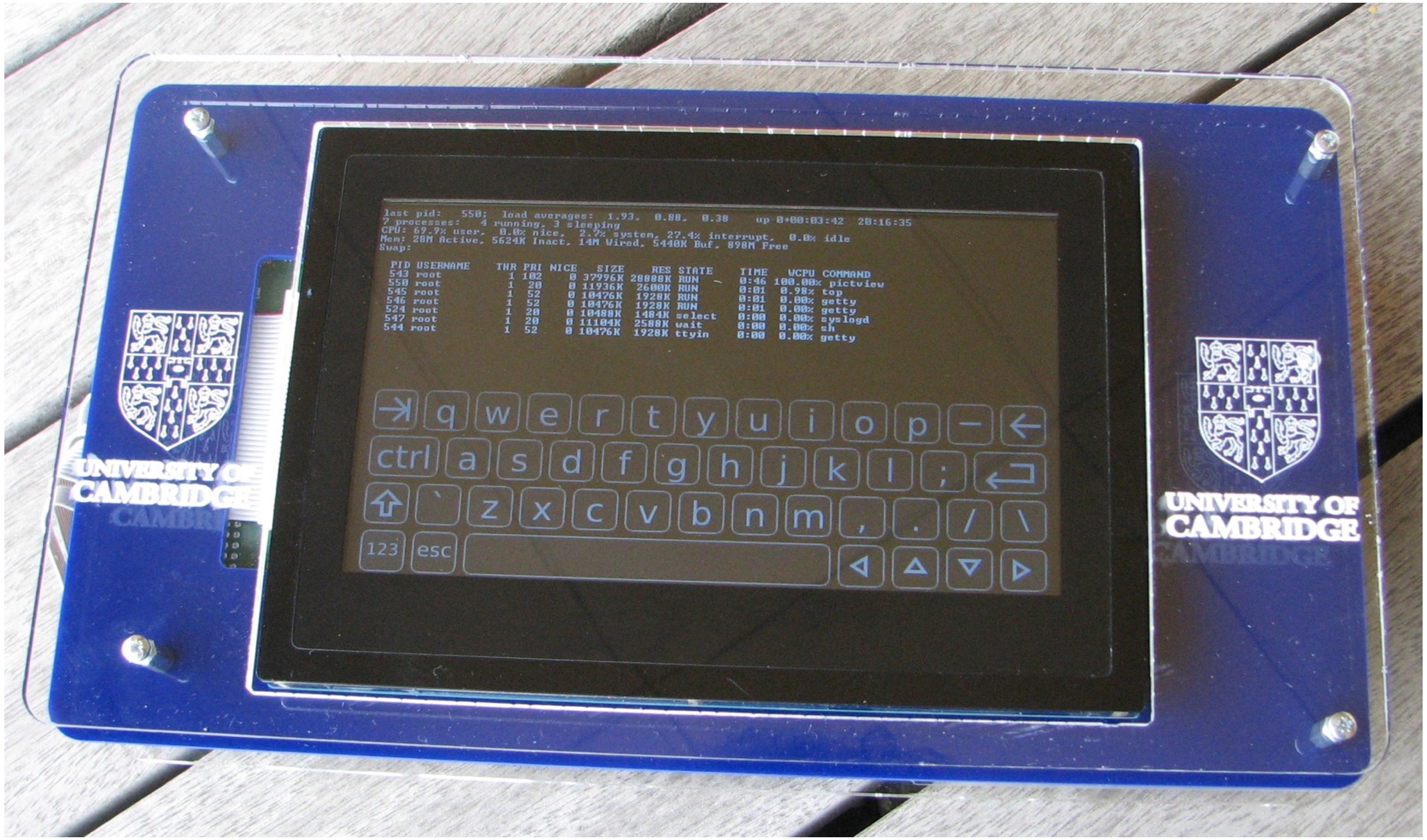
Foundation for higher-level models such as **software compartmentalization**

Compartmentalisation

- Compartment can be described using a sealed pair of capabilities: (program counter, default data capability)
- CCall providing the domain transition
- Allows a number of abstract software models:
  - Library compartmentalisation, e.g. of risky or legacy (non-cap.) code
  - Process-based compartmentalisation within an application can be replaced by much more efficient capability-based protection
    - Same virtual address space (more efficient TLB usage)
    - Very similar software model (easy to port code)

RESULTS

First we made it work – Demo tablet platform



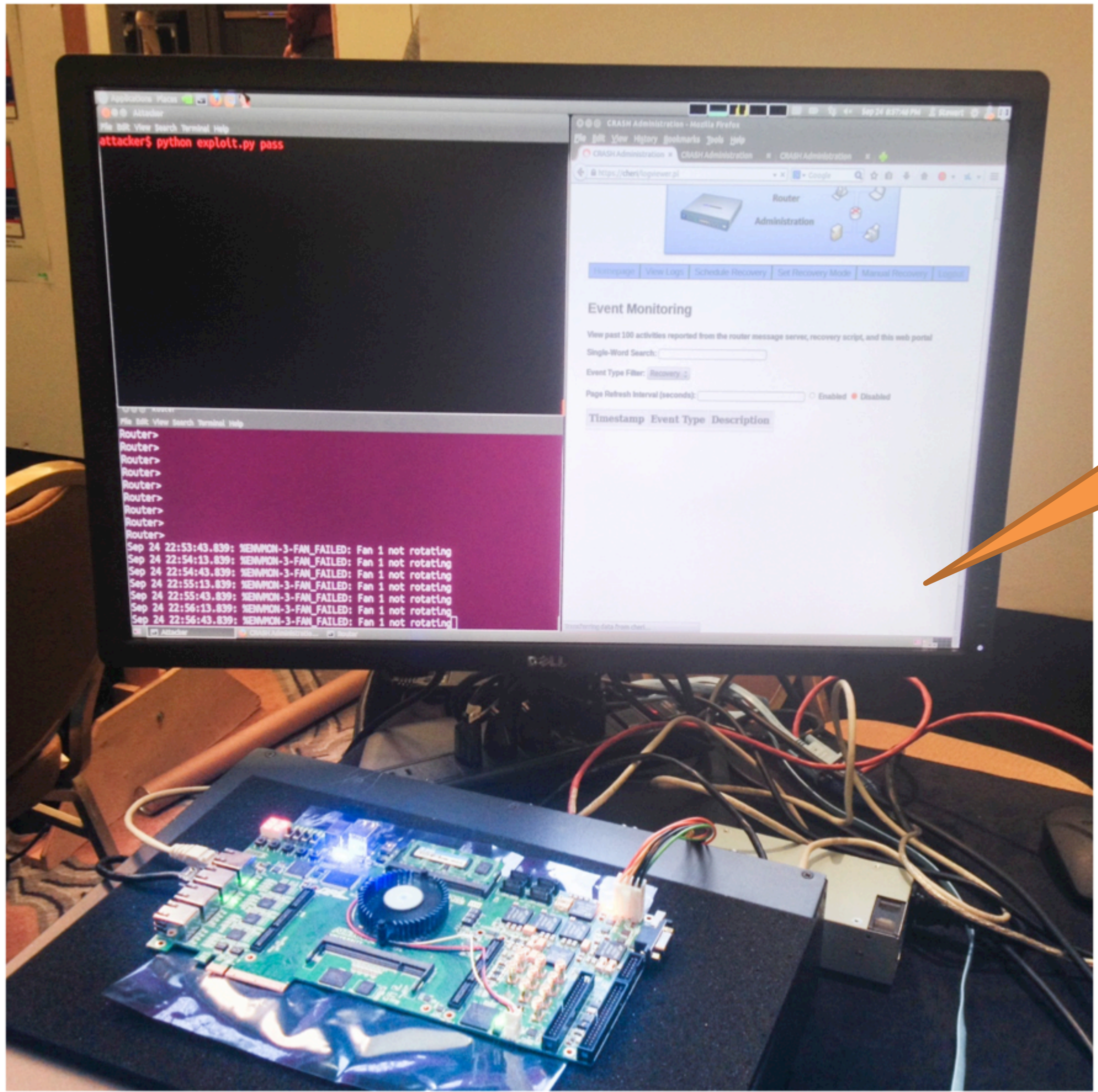
```
last pid:   550;  load averages:  1.93,  0.88,  0.38    up 0+00:03:42  20:16:35
7 processes:   4 running, 3 sleeping
CPU: 69.9% user,  0.0% nice,  2.7% system, 27.4% interrupt,  0.0% idle
Mem: 28M Active, 5624K Inact, 14M Wired, 5440K Buf, 898M Free
Swap:

  PID USERNAME     THR PRI NICE   SIZE    RES STATE    TIME   WCPU COMMAND
  543 root           1 102    0 37996K 28888K RUN      0:46 100.00% pictview
  550 root           1  20    0 11936K  2600K RUN      0:01  0.98% top
  545 root           1  52    0 10476K  1928K RUN      0:01  0.00% getty
  546 root           1  52    0 10476K  1928K RUN      0:01  0.00% getty
  524 root           1  20    0 10488K  1484K select   0:00  0.00% getty
  547 root           1  20    0 11104K  2588K wait     0:00  0.00% syslogd
  544 root           1  52    0 10476K  1928K ttyin    0:00  0.00% sh
```

SRI International

UNIVERSITY OF CAMBRIDGE

Red Team Evaluation by MIT Lincoln Labs



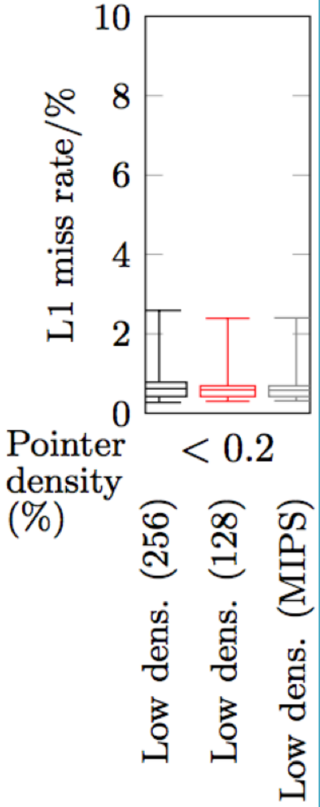
```
attacker$ python exploit.py pass
```

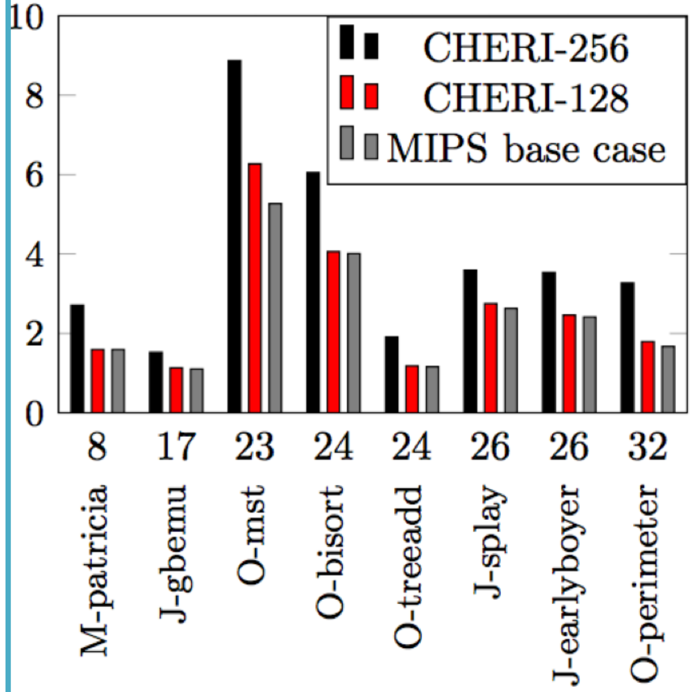
```
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Router>
Sep 24 22:53:43.839: %ENVMON-3-FAN_FAILED: Fan 1 not rotating
Sep 24 22:54:13.839: %ENVMON-3-FAN_FAILED: Fan 1 not rotating
Sep 24 22:54:43.839: %ENVMON-3-FAN_FAILED: Fan 1 not rotating
Sep 24 22:55:13.839: %ENVMON-3-FAN_FAILED: Fan 1 not rotating
Sep 24 22:55:43.839: %ENVMON-3-FAN_FAILED: Fan 1 not rotating
Sep 24 22:56:13.839: %ENVMON-3-FAN_FAILED: Fan 1 not rotating
Sep 24 22:56:43.839: %ENVMON-3-FAN_FAILED: Fan 1 not rotating
```

CHERI mitigates Heartbleed exploit!

Memory-protection performance

Collection of low pointer-density benchmarks from MiBench





High pointer-density benchmarks

(M) MiBench

(O) Olden

(J) Octane JavaScript

L1 cache miss rate for CHERI 256, CHERI-128, and MIPS

CheriABI: A full pure-capability OS userspace

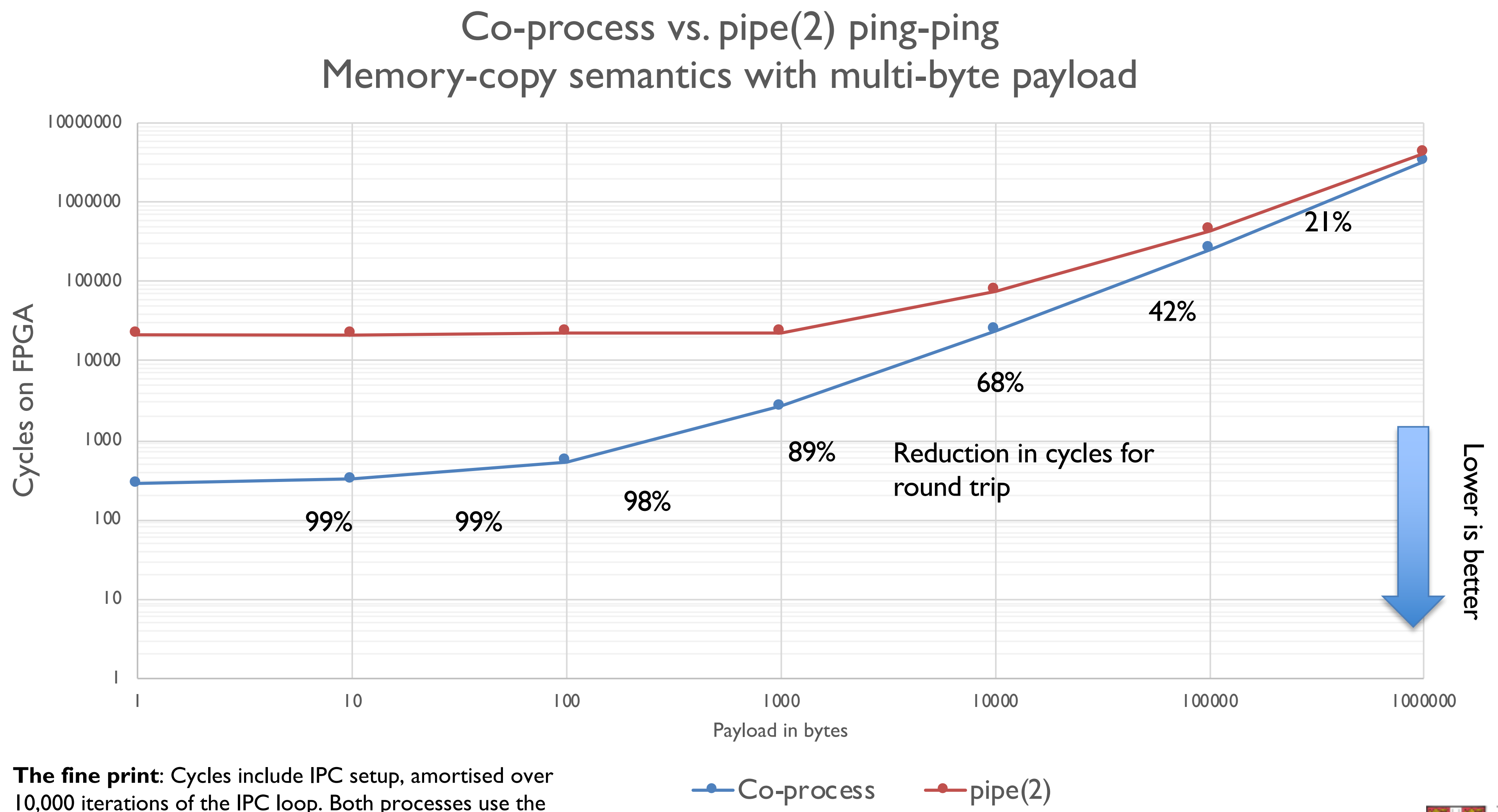
- Complete memory- and pointer-safe FreeBSD C/C++ userspace
  - **System libraries**: crt/csu, libc, zlib, libxml, libssl, …
  - **System tools and daemons**: echo, sh, ls, openssl, ssh, sshd, …
  - **Applications**: PostgreSQL, nginx; bringing up WebKit (C++)
- **Valid provenance**, **minimized privilege** for **pointers, implied VAs**
  - Userspace capabilities originate in **kernel-provided roots**
  - Compiler, allocators, run-time linker, etc., **refine** bounds and perms
- Trading off **privilege minimization**, **monotonicity**, **API conformance**
  - Typically in memory management – realloc(), mmap() + mprotect()

Evaluating memory-protection compatibility

- Prototyping approach:
  - "pure-capability" **C compiler** (Clang/LLVM)
  - **full OS** (FreeBSD) that use capabilities for all explicit or implied userspace pointers
- Observations:
  - **Little or no software modification** (BSD base system + utilities)
  - Small changes to source files for 34 of 824 programs, 28 of 130 libraries
  - Overall: modified ~200 of ~20,000 user-space C files/header

CHERI vs. Process-based Compartmentalization

(Early IPC ping-pong microbenchmark results)



The fine print: Cycles include IPC setup, amortised over 10,000 iterations of the IPC loop. Both processes use the pure-capability ABI using 256-bit capabilities. 272-entry TLB, 32K L1 I-Cache, 32K L1 D-Cache, 256K L2 Cache.

CURRENT RESEARCH DIRECTIONS

Generalising CHERI support for many ISAs

- 64-bit MIPS for pragmatic reason: needed a 64-bit RISC ISA in late 2010
- Generic CHERI support doesn't mean that all implementations need to be identical
    - E.g. **portable virtual-memory semantics** and **UNIX process model** despite (quite) different MMUs across architectures

- **Architectural abstraction**: Lift CHERI properties above ISA
- **Architectural localization**: E.g., ISA choices, opcode approaches, exceptions, page tables, … → **architecture-specific specifications**

- Currently working on **CHERI-ARM** and **CHERI-RISC-V** variants
- Currently exploring interaction with virtualization for servers

Portability implications for software

- **CHERI Clang/LLVM**
  - Modest pointer/capability abstraction improvements in front-end and IR
  - Adapt target back-ends to teach them about capability code generation
  - Optimize for architecture-specific code generation
  - Optimize for available microarchitectures
- **CheriBSD** (CHERI support in FreeBSD)
  - More clear machine-independent / machine-independent split
  - Shift to hybrid capability C in the kernel to improve machine independence
  - Various MD kernel updates: boot code, exceptions, PMAP, …
  - Clean up APIs, header separation, architecture abstraction
  - Various userspace updates: rtld, libcheri, CRT/CSU, …

Many other research questions

- Can we efficiently impose CHERI protection mechanisms on I/O devices and accelerators?
  - See Thunderclap work on I/O security (http://thunderclap.io/)
- Does CHERI make managed languages (e.g. Rust) safer or faster (e.g. through efficient dynamic checks)?
- Does fine-grained compartmentalisation help mitigate fault injection attacks?
- Can CHERI help to mitigate speculative execution attacks?
- Can CHERI be used for enclaves?

Conclusions

- CHERI provides the hardware with more semantic knowledge of what the programmer intended
  - Toward the **principle of intentional use**
- 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**
  - Large performance improvement over process-based compartmentalisation
- **Working with industry to bring the technology to market**
- Thanks to sponsors: DARPA, ARM, Google, EPSRC, HEIF, Isaac Newton Trust, Thales E-Security, HP Labs

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

Simon.Moore@cl.cam.ac.uk
Computer Science & Technology

Additional Topics

1. Our verification and test strategy
2. How to build efficient tagged memory
3. Compressed capabilities
4. How CHERI helps to mitigate speculative execution attacks
5. THUNDERCLAP: The Perils of Peripherals