R265: Advanced Topics in Computer Architecture

Background to Memory Safety

Prof. Simon W. Moore



UNIVERSITY OF CAMBRIDGE
Computer Science & Technology

Copyright © Simon W. Moore, 2026

Overview

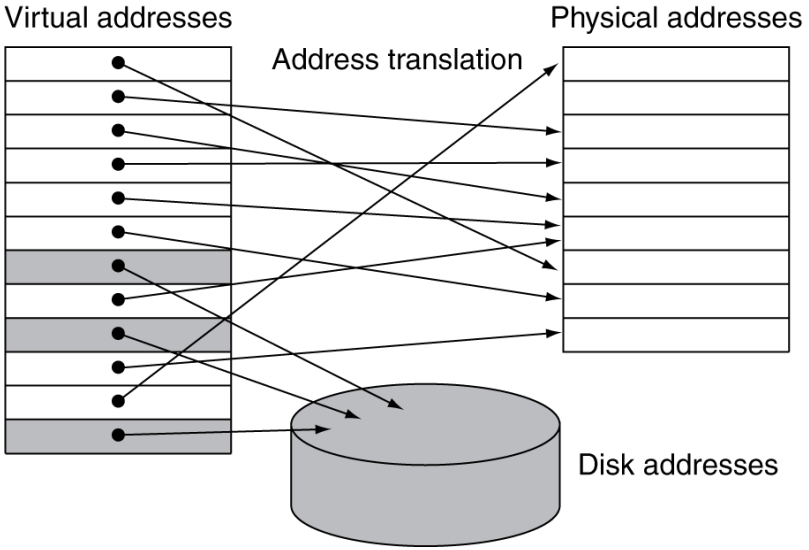
- Refresher on page-based virtual memory
- Fine grained memory safety using CHERI capabilities
- Papers this week:
  - Managing virtual memory in multi-core systems
  - Virtual memory for virtual machines
  - CHERI-Morello (CHERI on ARM)

Introducing virtual memory

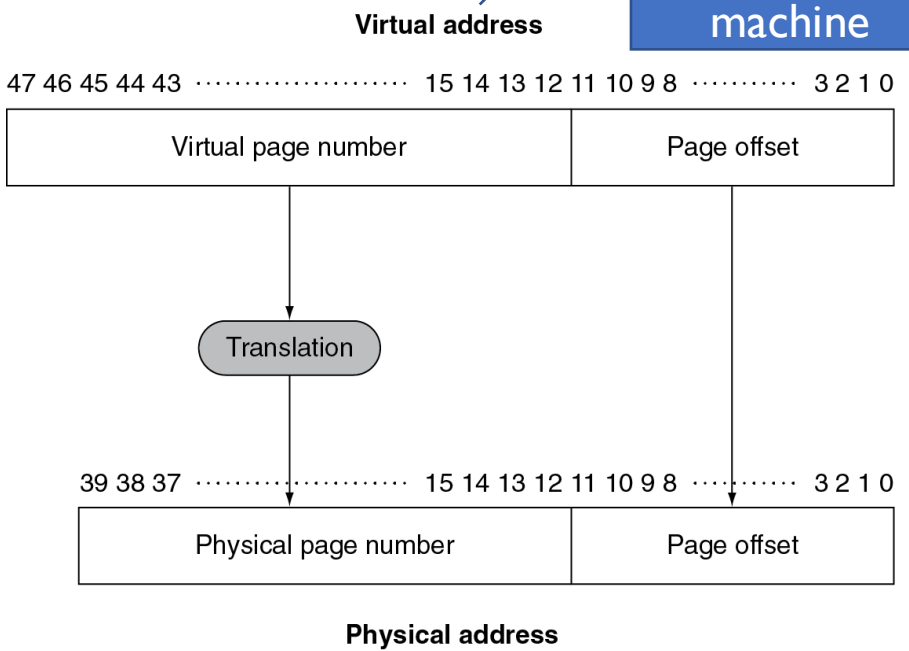
- Give each application the illusion that it has its own address space
- Processor + OS translates virtual to physical addresses
  - Memory broken into blocks or **pages** (e.g. 4KiB) or **superpages** (e.g. 4MiB)
  - Each translation fault is called a **page fault**
- Aids memory allocation
- Also provides memory protection
  - Permission bits per page determine what is shared and what is not
- Also allows rarely used pages to be migrated to disk: **paging**
  - Gives the illusion of a bigger memory
  - Can be useful to tolerate long running applications with memory leaks
  - But slow if paging to magnetic disk and risks wearing out flash disk (SSD)
- Can have virtual memory without paging (e.g. most smart phones)

Concept of address translation

- Fixed-size pages: e.g., 4KiB page or 4MiB superpage



example for 64-bit machine



Page Tables

- Stores virtual to physical translation information
  - Array of page table entries (PTEs), indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE can refer to a location in swap space on disk
  - Software **Page Fault Handler** invoked

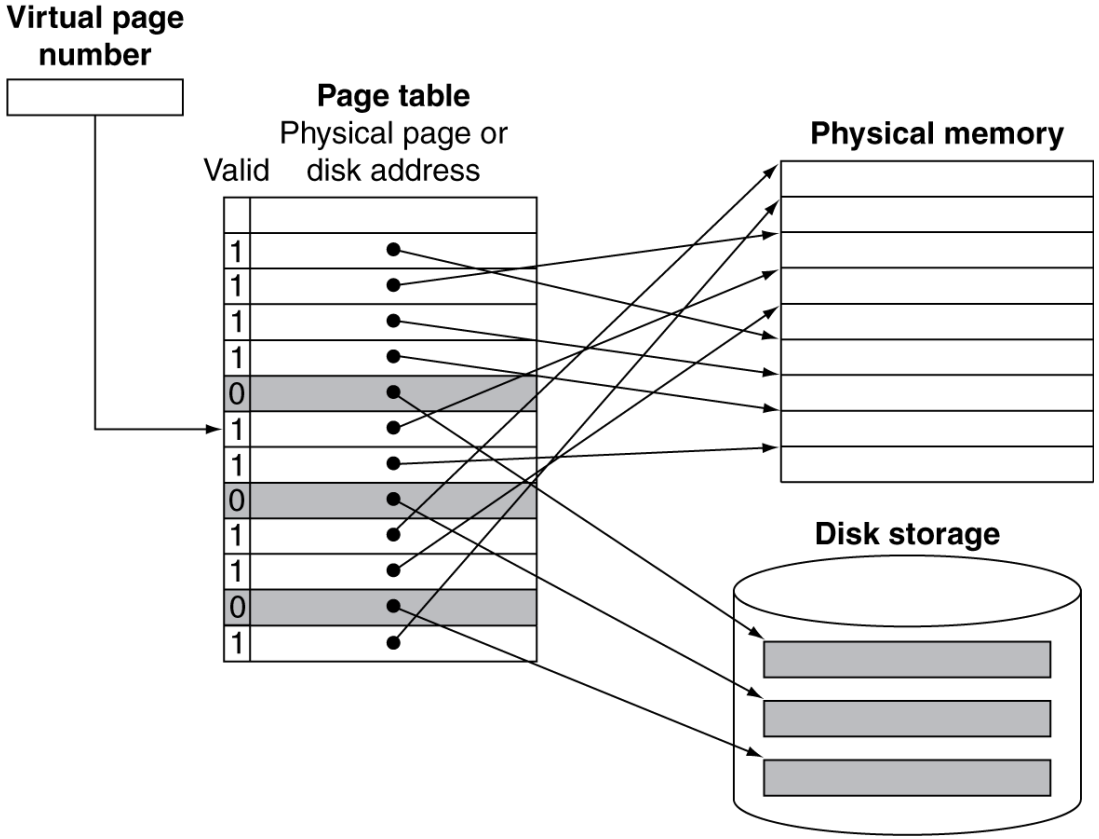
Page Fault Handler

- Page fault taken when PTE indicates that page is on disk
  - Software (the OS) has to get involved at this point
- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction

Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Handled by OS code
  - Can take millions of clock cycles
- Try to minimize page fault rate
  - Fully associative placement (any page on disk can be mapped to any page in memory)
  - Smart replacement algorithms

Mapping Pages to Disk

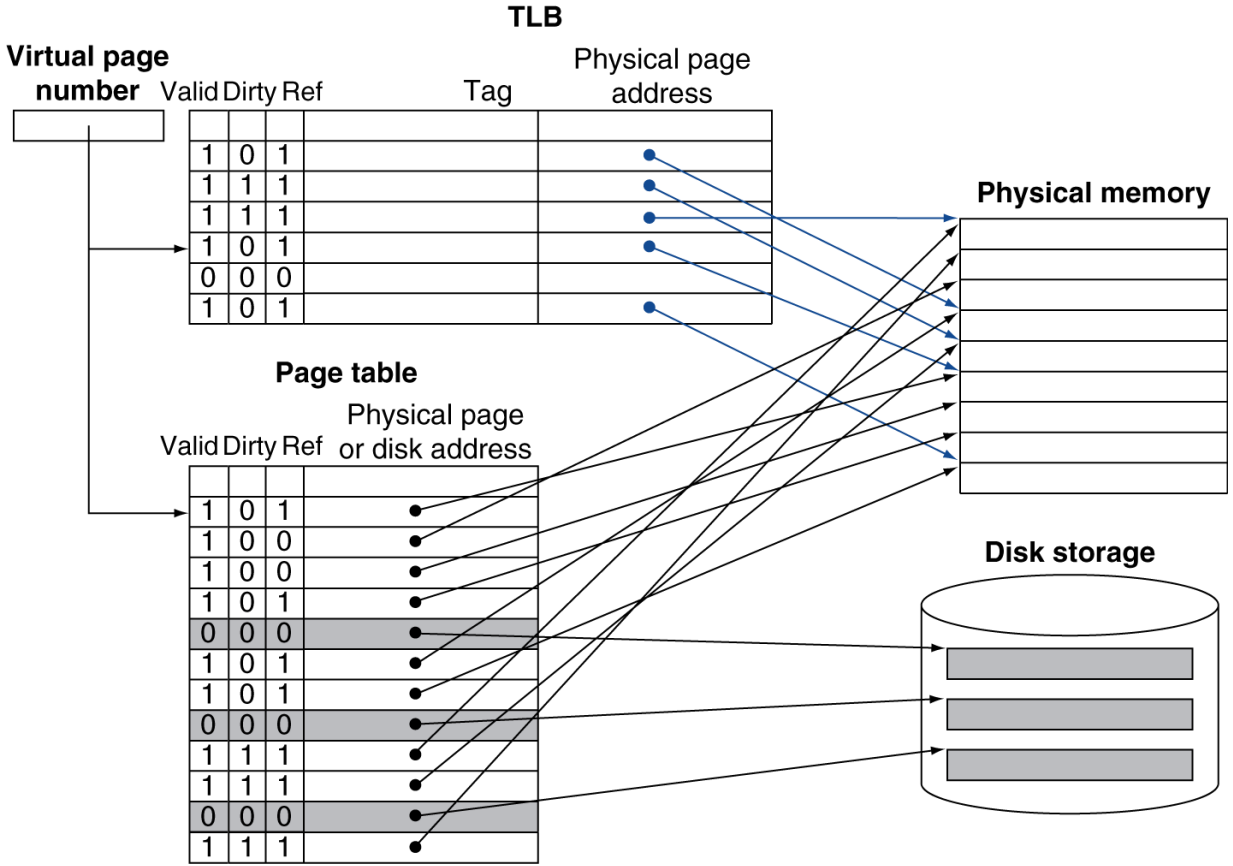


- DRAM becomes cache of the disk
- Disk accesses are slow, so careful management is critical, e.g. dirty bit – if not set then don't write the page back to disk
- Page table allows fully associative mapping
- LRU replacement policy

Translation Look-aside buffer (TLB) = cache of PTEs

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB) = cache of PTEs
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - TLB lookup usually performed before L1 cache access
  - TLB misses could be handled by hardware or software
    - Older processors like MIPS handled in software
    - Modern processors all have hardware page table walkers

Fast Translation Using a TLB



TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
    - Usually worth it for increased performance
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

Hardware page table walker

- TLB miss causes the hardware page table walker to search for the PTE, i.e. the TLB is a cache of the page table
  - PTE found gets put into the TLB
  - PTE not found results in an exception
- Must recognize TLB miss before destination register overwritten
  - Typically memory operations are replayed (re-run) after the TLB miss has been performed

Fine grained memory protection with CHERI capabilities



UNIVERSITY OF CAMBRIDGE

Computer Science & Technology

Challenge

- *"Buffer overflows have not objectively gone down in the last 40 years. The impact of buffer overflows have if anything gone up."*
  Ian Levy, Technical Director at NCSC, 2022

- Matt Miller from MS Response Center @ BlueHat 2019:
  - From 2006 to 2018, year after year, 70% MSFT CVEs are memory safety bugs

Motivation – The Eternal War in Memory*

- Many security vulnerabilities exploit memory safety violations







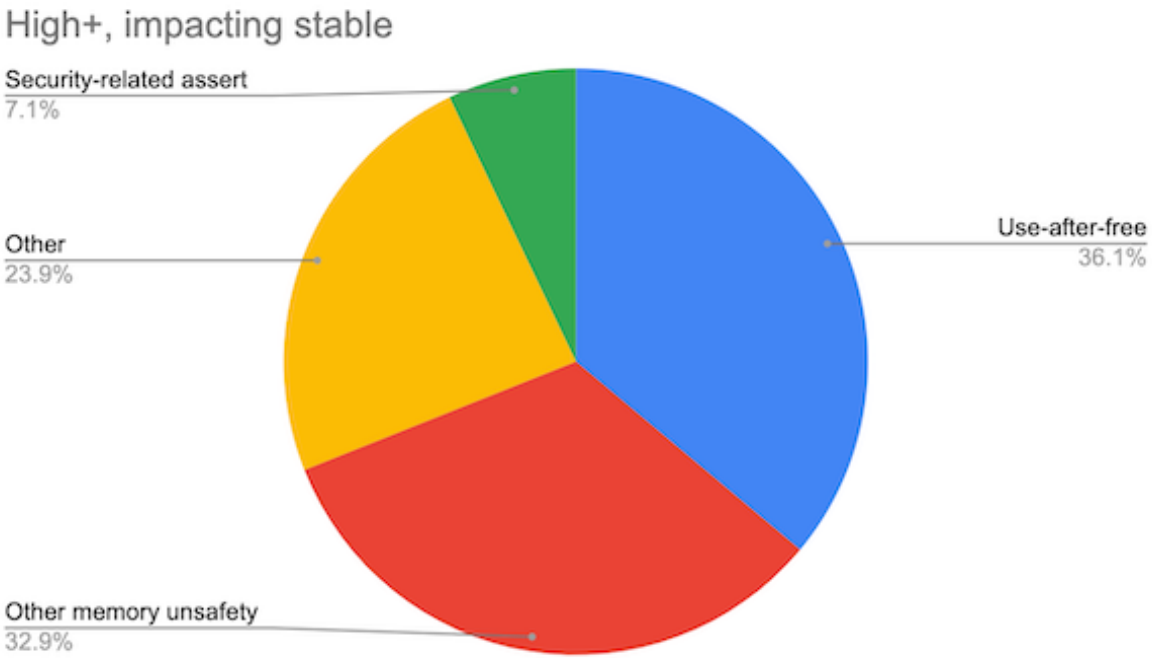


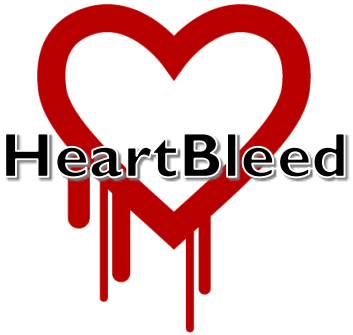
* Title based on Oakland 2013 paper: SoK: Eternal War in Memory, László Szekeres, Mathias Payer, Tao Wei, Dawn Song

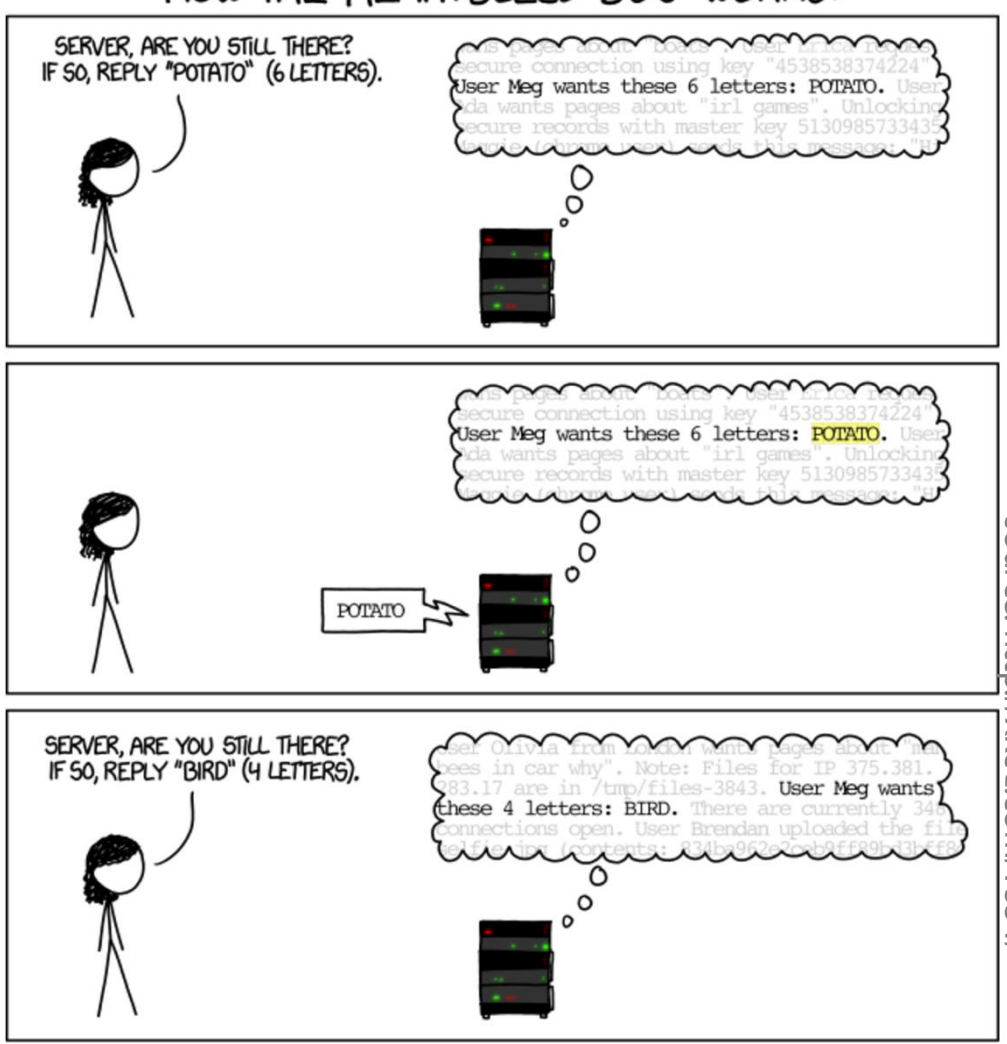
Example – Chromium Browser Safety

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

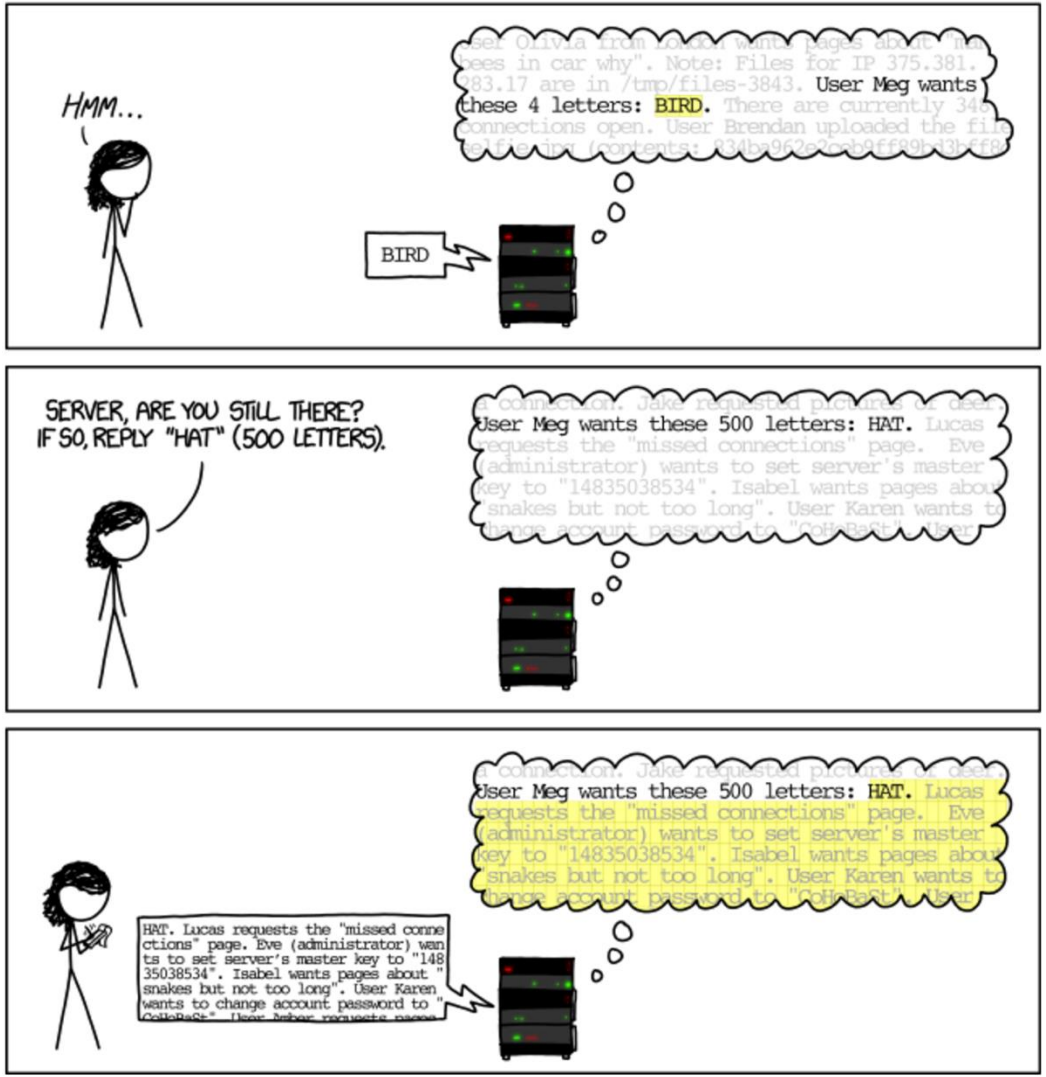
www.chromium.org/Home/chromium-security/memory-safety



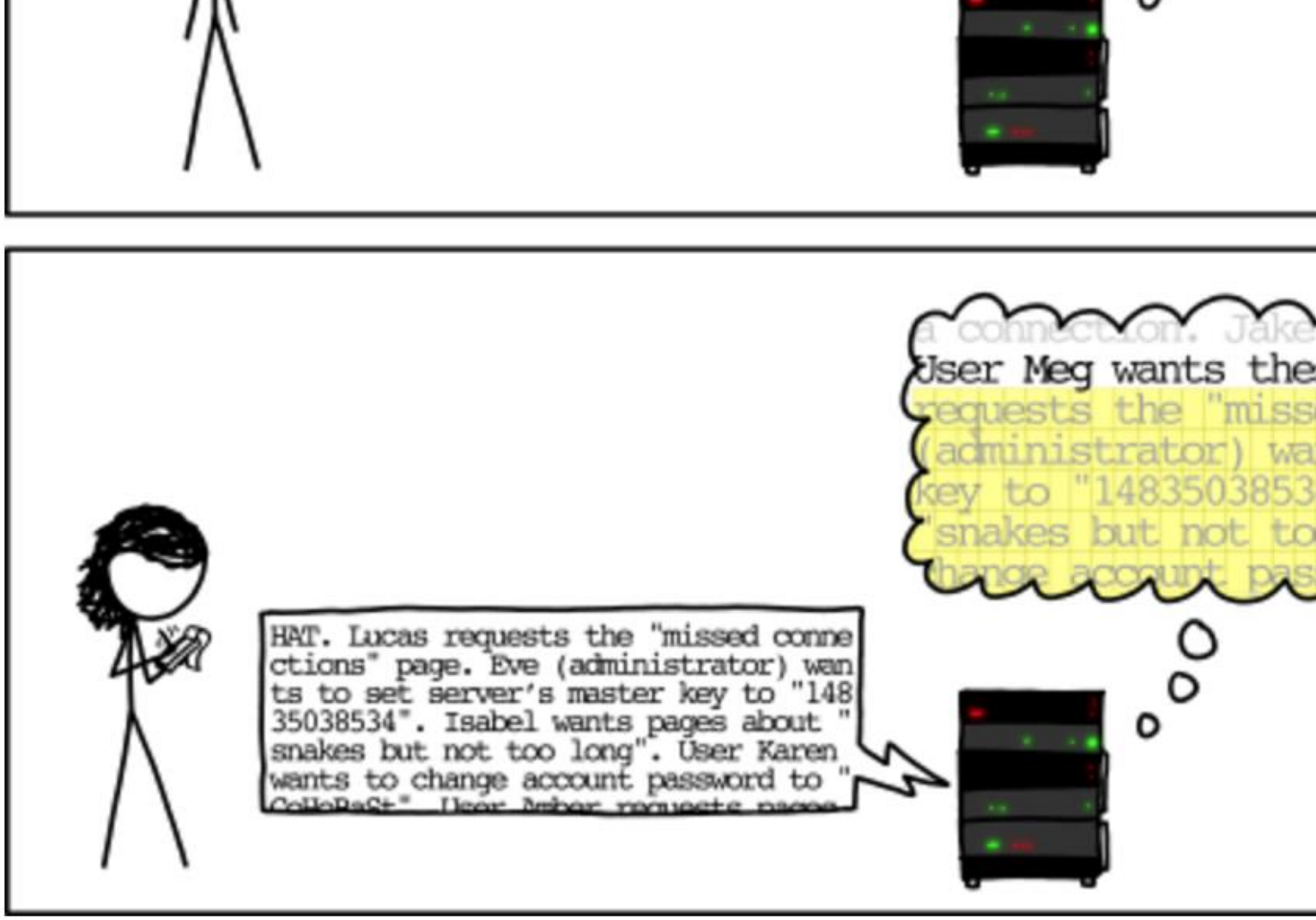




source: http://xkcd.com/1354/



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


User Meg wants the
requests the "miss
(administrator) wa
key to "1483503853
snakes but not to
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?

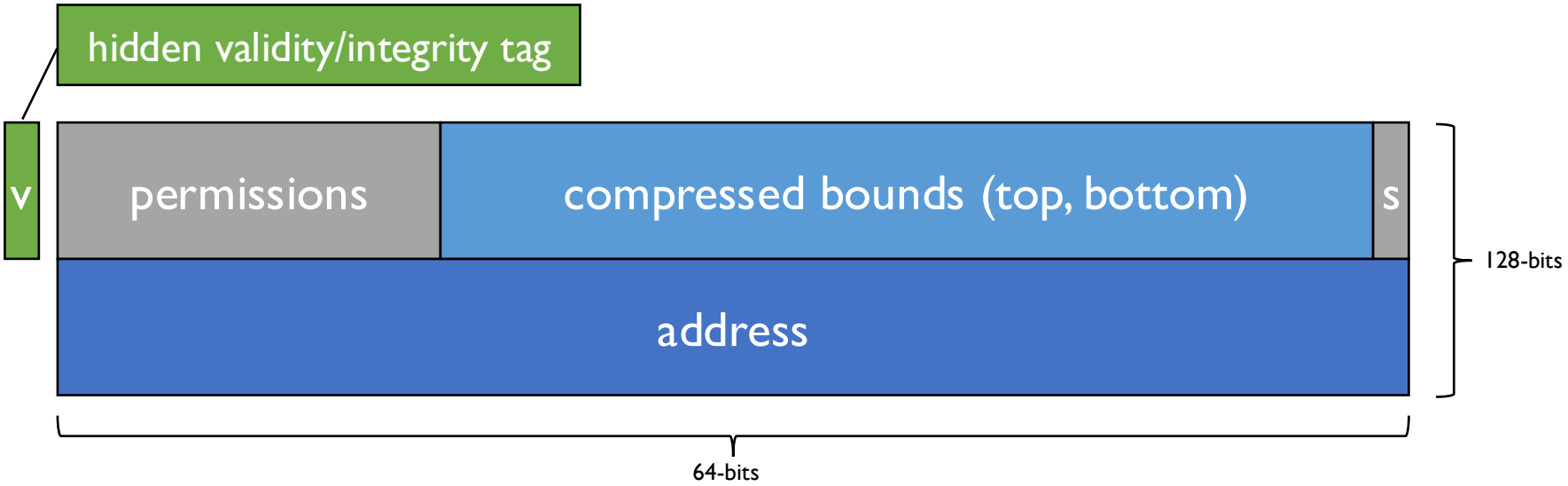
- Classical answer:
  - The programmer forgot to check the bounds of the data structure being read
  - Fix the vulnerability in hindsight – one line fix:
    if (1+2+payload+16 **>** s->s3->rrec.length) **return** 0;
- Our answer:
  - Preserve bounds information during compilation
  - Use hardware (CHERI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

CHERI – A Modern Capability Architecture

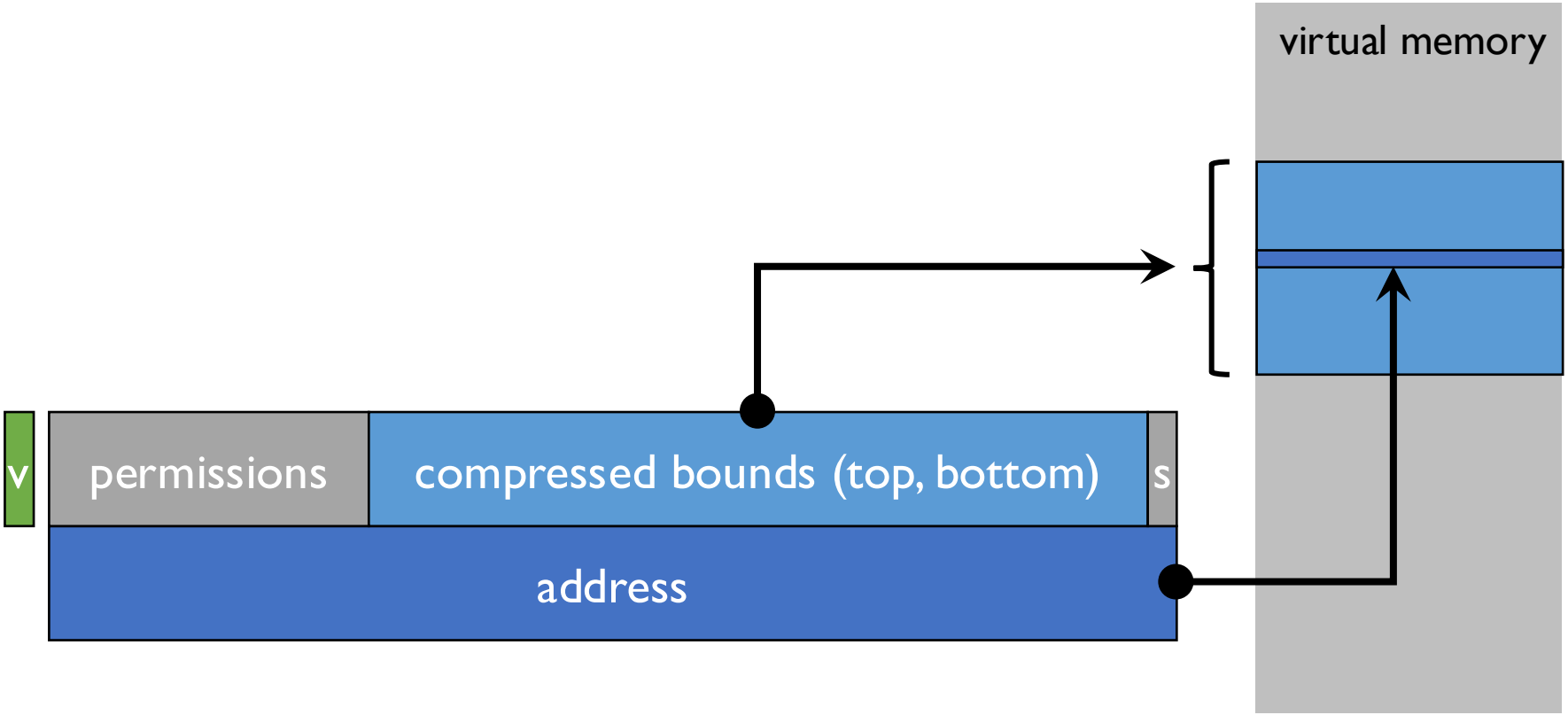
CHERI = Capability Hardware Enhanced RISC Instructions

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**



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

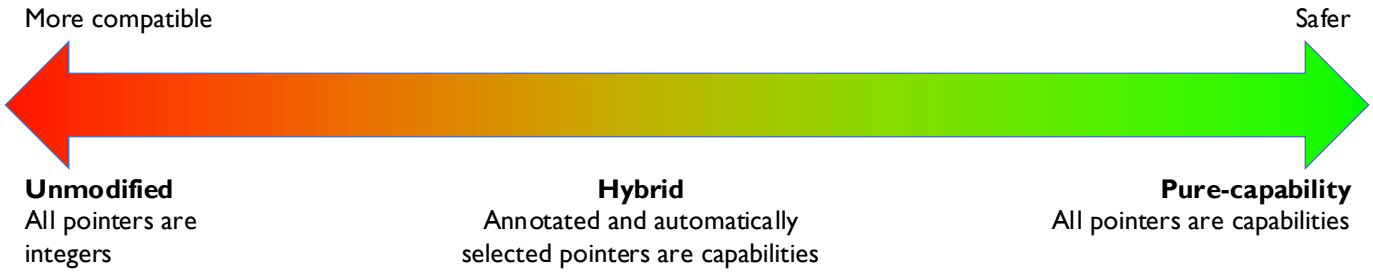
critical property for security

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



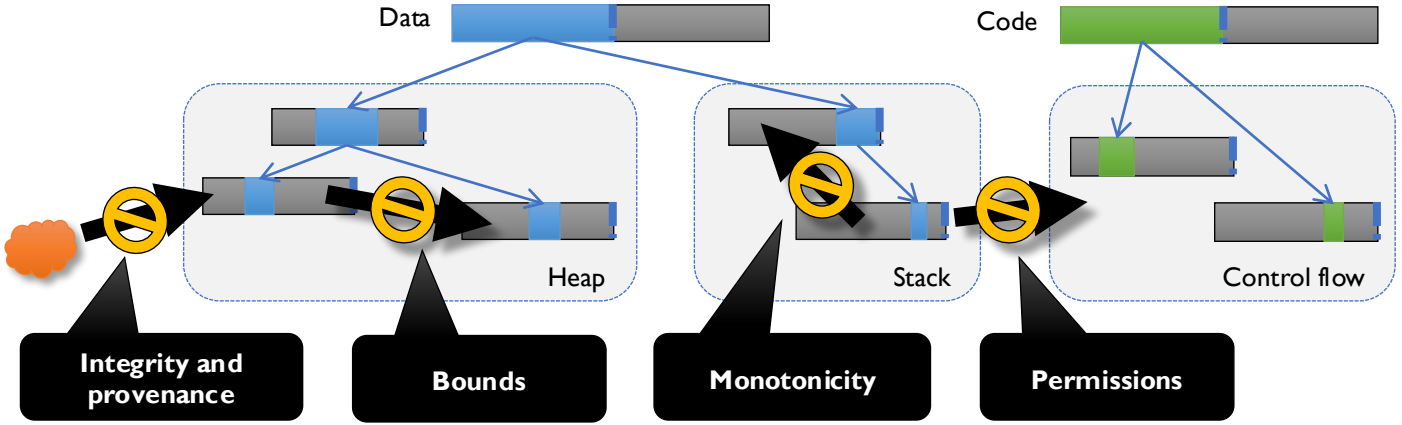
- **Source and binary compatibility**: **C-language idioms,** multiple **ABIs**
  - **Unmodified code**: Existing code runs without modification
  - **Hybrid code**: E.g., used in return addresses, for annotated data/code pointers, for specific types, stack pointers, etc.

    … 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** 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

Summary of Capability Protections

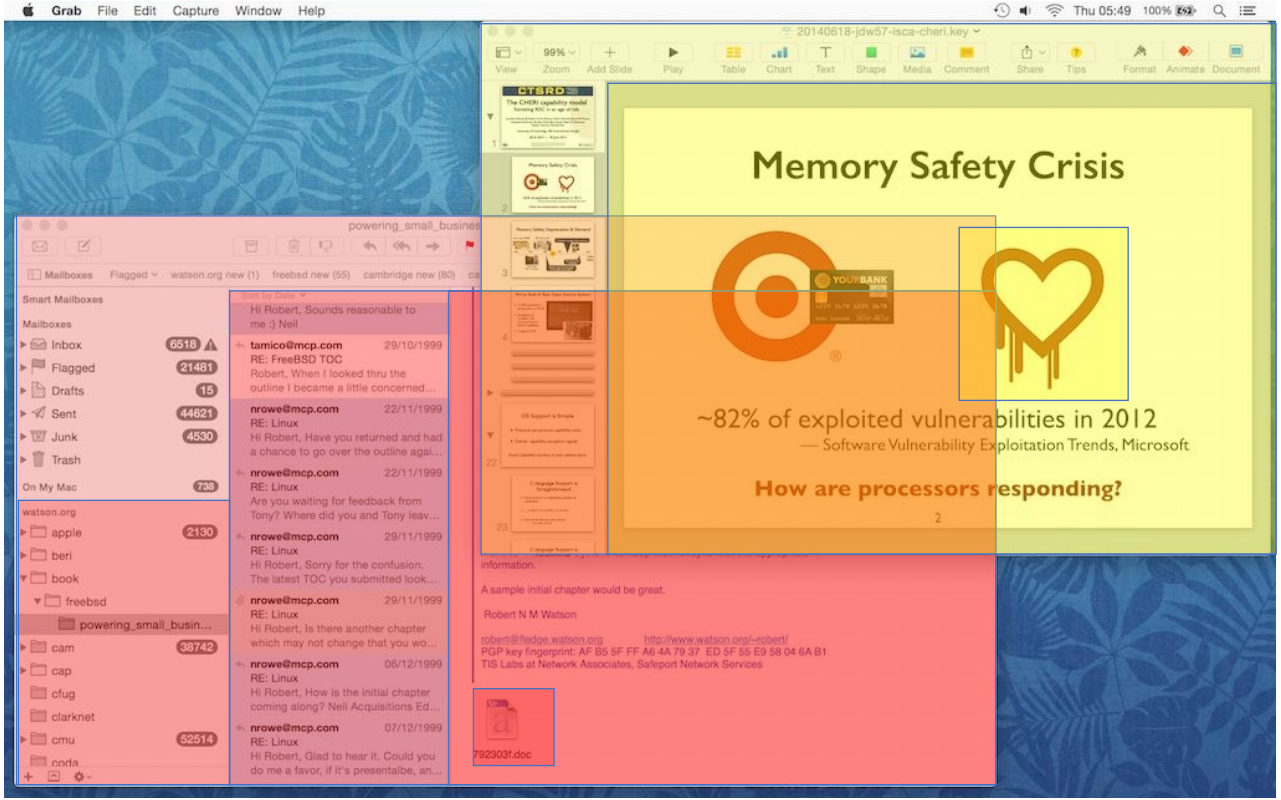


- **Valid userspace pointer set** – pointers not generated using derivation rules are not part of the valid provenance tree and will not be dereferenceable
- **Pointer privilege reduction** – capabilities allow pointers to carry specific privileges, which can be minimized with OS, compiler, and linker 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)
- CInvoke 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)

Compartmentalisation Illustrated



CHERI: Conclusions

- 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**
  - 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, Huawei



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

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

Reading

1. *Avoiding TLB Shootdowns through Self-invalidating TLB Entries*. PACT 2017.
   DOI: 10.1109/PACT.2017.38
   https://ieeexplore.ieee.org/abstract/document/8091251
2. *NEVE: Nested Virtualization Extensions for ARM*. SOSP 2017.
   DOI: 10.1145/3132747.3132754
   https://dl.acm.org/doi/10.1145/3132747.3132754
3. *The Arm Morello Evaluation Platform—Validating CHERI-Based Security in a High-Performance System*. IEEE Micro 2023.
   DOI: 10.1109/MM.2023.3264676
   https://ieeexplore.ieee.org/document/9895591

- Optional background material:
  - *It Is Time to Standardize Principles and Practices for Software Memory Safety*. Communications of the ACM 68, 2 (February 2025), 40–45.
    DIO: 10.1145/3708553
    https://cacm.acm.org/opinion/it-is-time-to-standardize-principles-and-practices-for-software-memory-safety/