CheriABI

Hardware enforced memory safety for FreeBSD

Brooks Davis, Robert N. M. Watson, Alexander Richardson,
Peter G. Neumann, Simon W. Moore, John Baldwin, David Chisnall,
James Clarke, Nathaniel Wesley Filardo, Khilan Gudka, Alexandre Joannou, Ben Laurie,
A. Theodore Markettos, J. Edward Maste, Alfredo Mazzinghi,
Edward Napierala, Robert Norton, Michael Roe, Peter Sewell, Stacey Son,
Jonathan Woodruff

SRI International, University of Cambridge, Microsoft Research, Google, Inc

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 contracts FA8750-10-C-0237 (“CTSRD”) and HR0011-18-C-0016 (“ECATS”). 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.

Punchline: it really does work

- Full FreeBSD operating system with spatial and referential memory safety
  - Covers programs, libraries, and linkers
  - Kernel access to user memory
- Performance is generally acceptable
- Significant 3rd-party software works: PostgreSQL database, Webkit

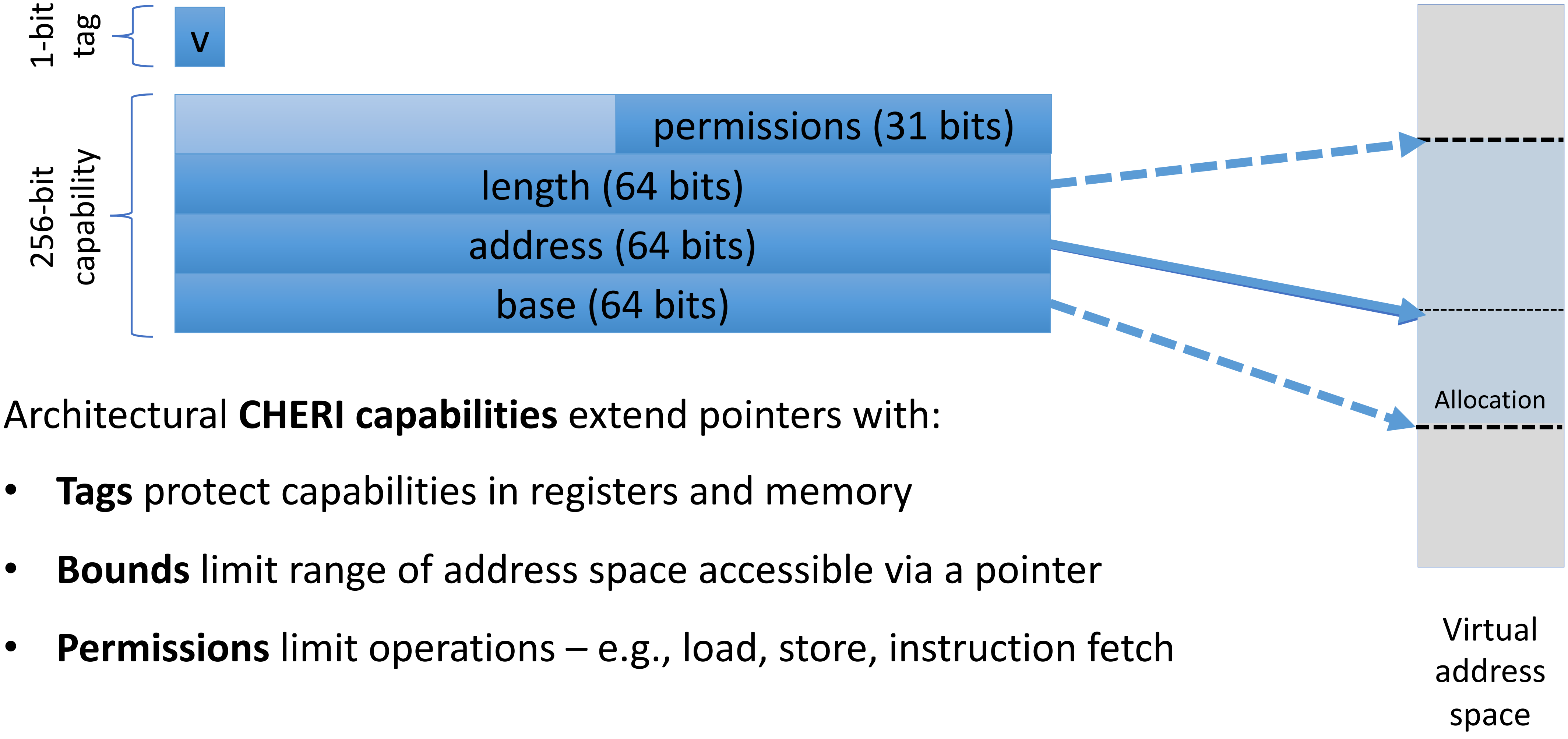
Introduction to CHERI

- CHERI introduces a new register type: the **capability**
  - In addition to integer and floating point
- CHERI capabilities grant access to bounded regions of virtual address space
  - Protected by tags

Watson, et al. **CHERI: a research platform deconflating hardware virtualization and protection.** RESoLVE 2012.

Woodruff, et al. **The CHERI capability model: Revisiting RISC in an age of risk**. ISCA 2014.

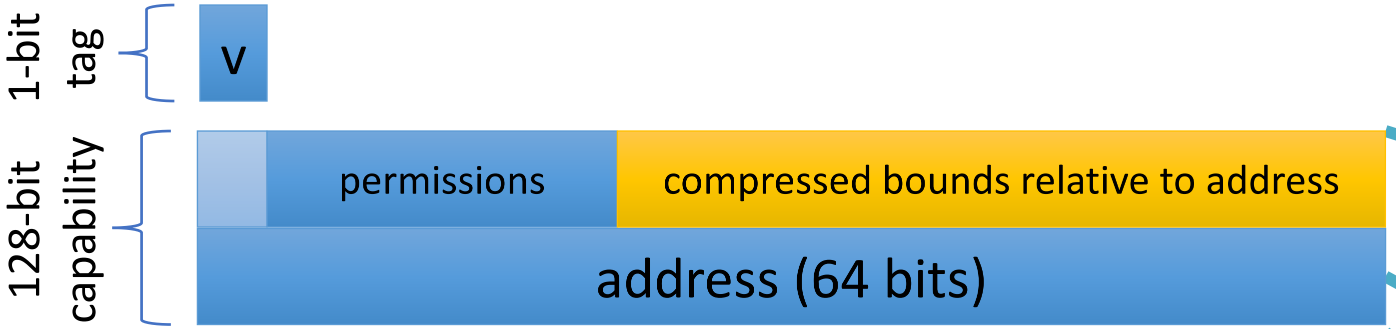
Architectural CHERI capabilities

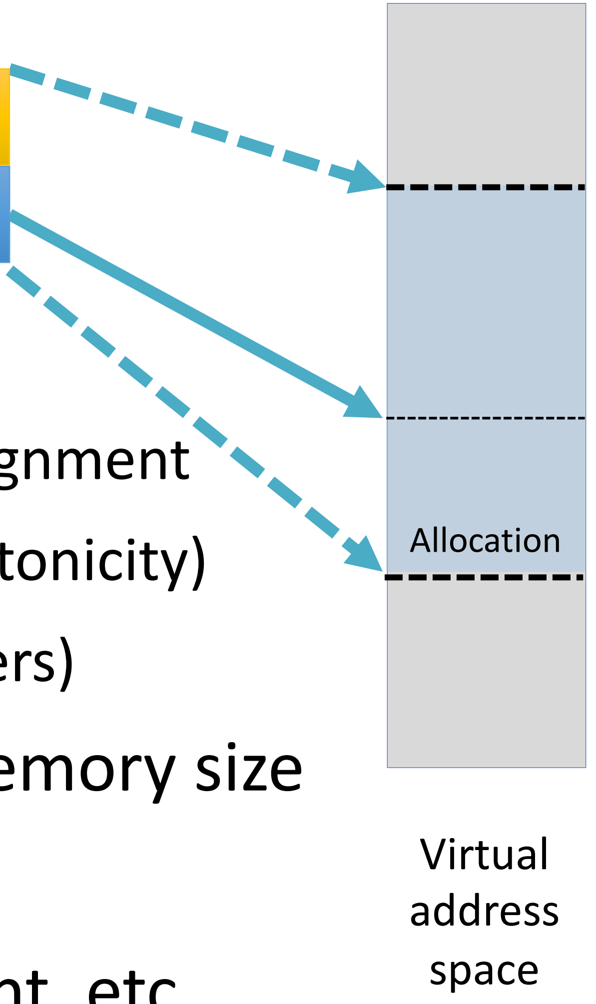


Architectural **CHERI capabilities** extend pointers with:

- **Tags** protect capabilities in registers and memory
- **Bounds** limit range of address space accessible via a pointer
- **Permissions** limit operations – e.g., load, store, instruction fetch

128-bit compressed capabilities





- **Compress bounds** relative to 64-bit virtual address
  - Floating-point bounds mechanism constrains bounds alignment
  - Security properties maintained (e.g., provenance, monotonicity)
  - Strong C-language support (e.g., for out-of-bound pointers)
- DRAM tag density from 0.4% to 0.8% of physical memory size
- Full prototype with full software stack on FPGA
- Implications for memory allocators, object alignment, etc

CHERI memory operation

- All memory access via CHERI capabilities
  - Explicit (new instructions):
    - Capability load, store, branch, jump
  - Implicit (legacy MIPS ISA):
    - via Default Data Capability (DDC) or Program Counter Capability (PCC)

CHERI capability manipulation

- Capabilities are used and manipulated in capability registers with capability instructions
  - Manipulations are monotonic (can only reduce bounds and permissions)
  - `CAndPerm cd, cb, rt`
  - `CSetAddr cd, cs, rs`
- Capabilities can be stored in memory, protected by tags
  - Non-capability stores clear tags

Capabilities as C pointers

- CHERI capabilities are designed for use as C pointers
  - Allowed to be out of bounds between dereferences
  - Can store 64-bit integers (untagged)
  - No protection tables or privileged operations
- Two compilation modes:
  - Hybrid: `__capability` annotation applied to select pointers
  - Pure-capability: all pointers are capabilities

Chisnall, et al. **Beyond the PDP-11: Processor support for a memory-safe C abstract machine.** ASPLOS 2015.

CheriABI: Pure-capability process environment

- Built on CheriBSD (FreeBSD modified for CHERI)
- All program pointers are capabilities
  - Including syscall arguments and return values
- Goal: Bounds are minimized
  - C-language objects
  - Pointers provided by the kernel
- Goal: run pure-capability programs with simple recompilation

Watson, et al. **CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization.** Oakland 2015.

Chisnall, et al. **CHERI-JNI: Sinking the Java security model into the C.** ASPLOS 2017.

Implementation: kernel

- CheriABI is implemented as a compat layer (i.e. freebsd32)
- The kernel is a hybrid CHERI-C program
  - Pointers to userspace are annotated with `__capability` and are capabilities.
  - Select data structures (e.g. `struct iovec`, signal bits) converted to store capabilities.
- All userspace access via capabilities
  - Capability aware versions of userspace access functions: `copyin_c/copyout_c/fueword_c`, etc
  - Non "_c" verisons return error for CheriABI processes
  - Capabilities not copied to/from userspace by default
    - Special copyincap/copyoutcap used to ensure copy is intentional

Implementation: userspace

- Libraries live in /usr/libcheri
  - Built before programs
- Programs can compile and link as legacy, hybrid, or pure-capability
- Almost-full support for external LLVM toolchain for mips64

Abstract capabilities

How should the systems programmer **think** about bounds?

New concept: *abstract capability*

- Set of permissions of the process
- Tracks ghost state across swapping, etc
- Constructed and maintained by a collaboration of the kernel and language runtime

System startup

Power-on state

Registers

DDC	RWX 0x0 - 0xFF…FF
PCC	RWX 0x0 - 0xFF…FF
C1-31	NULL

Memory

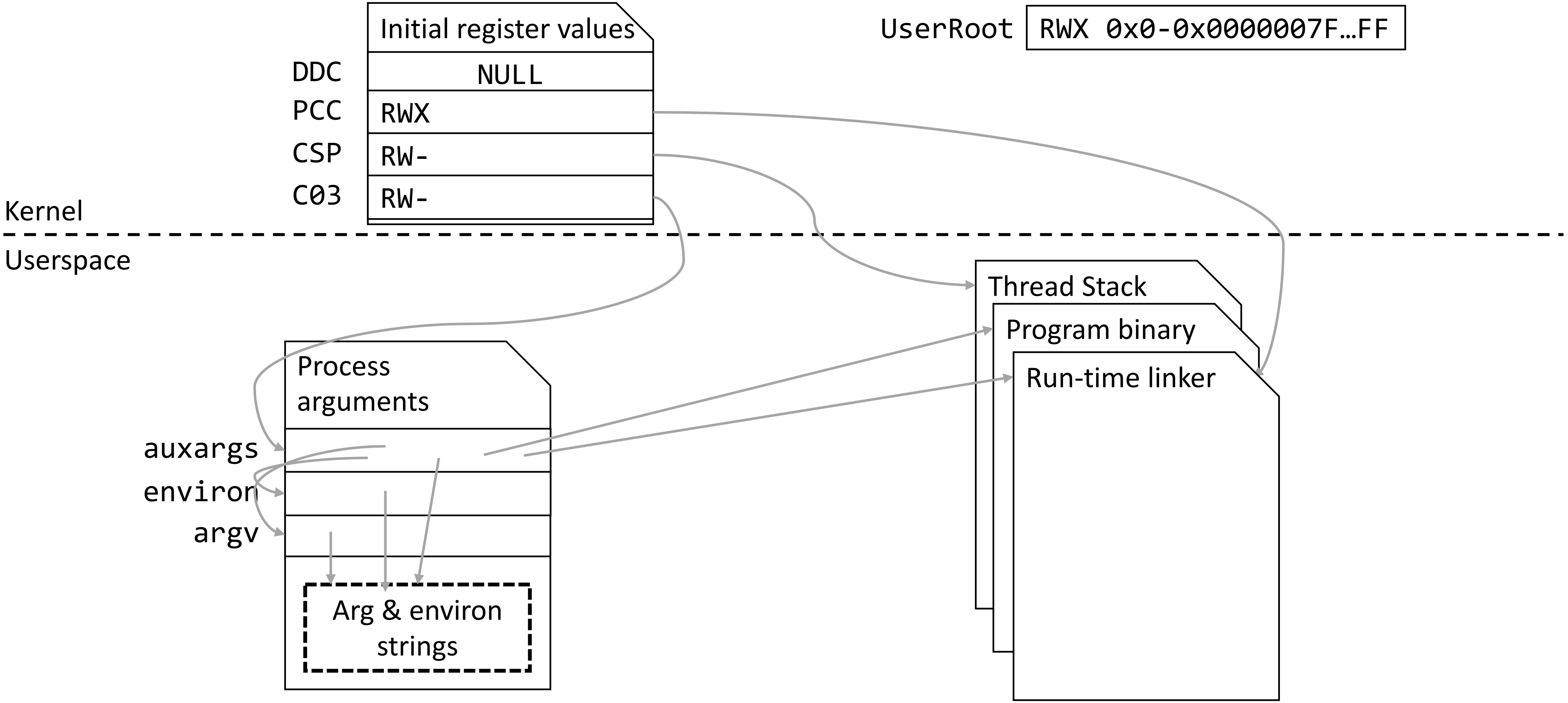
All tags clear

Early boot

DDC	RW- 0x0 - 0xFF…FF
PCC	R-X 0x0 - 0xFF…FF
C1-31	*Working set*

UserRoot	RWX 0x0-0x0000007F…FF
SwapRoot	RWX 0x0 - 0xFF…FF

Execve



Initial register values
DDC
NULL
PCC
RWX
CSP
RW-
C03
RW-
UserRoot
RWX 0x0-0x0000007F…FF
Kernel
Userspace
Thread Stack
Program binary
Run-time linker
Process arguments
auxargs
environ
argv
Arg & environ strings

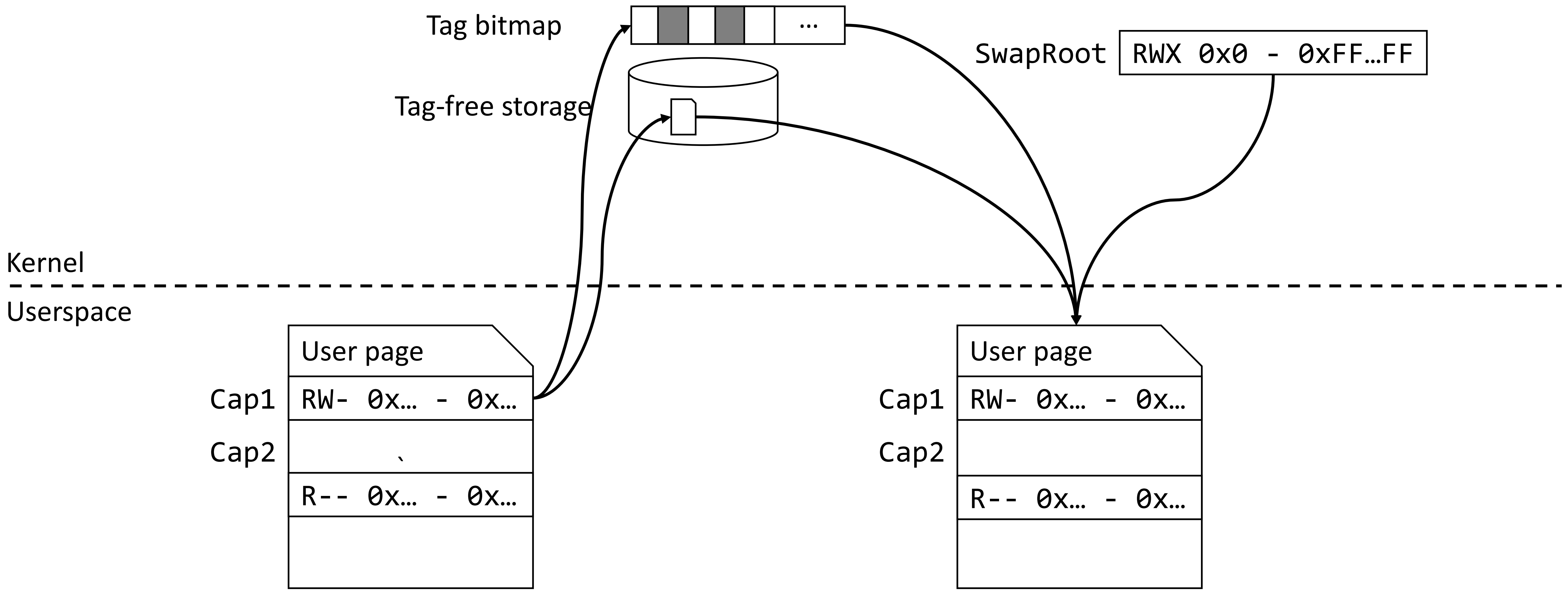
Virtual-memory system

- Programmer visible:
  - Provides capabilities to newly mapped regions via `mmap()` and `shmat()`
  - Alters and frees mappings
- Abstract capability maintenance:
  - Ensures correct virtual to physical mappings
  - Preserves stored capabilities in swapped pages

Virtual-memory system: mmap

- mmap() allocates virtual address space and changes mappings
- In CheriABI returns a bounded pointer
  - Imprecise mapping requests rejected
  - User must round-up unpresentable requests
- Permissions are set based on page permissions
  - PROT_MAX() extension allows PROT_NONE mappings for reservation

Virtual-memory system: swap



Run-time linker

- Loads and links dynamic libraries
- Resolves symbols and synthesizes capabilities
- Jumps to program entry point

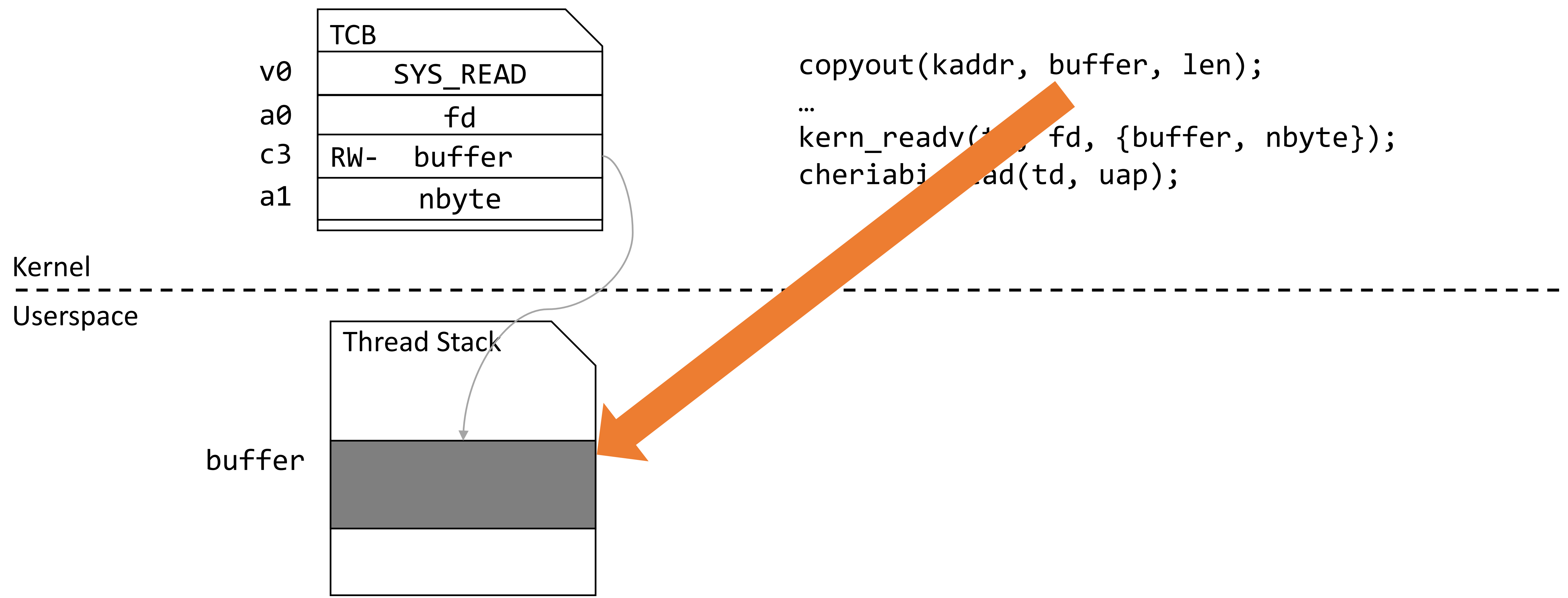
- Provides on-demand loading of libraries and supports exception handling

C runtime

- Objects allocated by `malloc()` are bounded to requested size
- `realloc()` adjusts bounds or allocates new storage
- Thread-local storage is bounded
  - Currently to per-thread storage
- Compiler generated code sets bounds on stack, automatic, and global objects as required

System calls

```
read(fd, buffer, nbyte);
```



Kernel code changes: read()

```
int user_read(struct thread *td, int fd, void * __capability buf,
    size_t nbyte)
{
        struct uio auio;
        kiovec_t aiov;
        if (nbyte > IOSIZE_MAX)
                return (EINVAL);
        IOVEC_INIT_C(&aiov, buf, nbyte);
        auio.uio_iov = &aiov;
        …
        return (kern_readv(td, fd, &auio));
}
```

Called by sys_read() and cheriabi_read()

New init macro for struct iovec

Required changes: pointer alignment

```
 label_done:
         if (metadata_thp_madvise()) {
                 /* Set NOHUGEPAGE after unmap to avoid kernel defrag
                    . */
-                assert(((uintptr_t)addr & HUGEPAGE_MASK) == 0 &&
+                assert(__builtin_is_aligned(addr, HUGEPAGE) &&
                     (size & HUGEPAGE_MASK) == 0);
                 pages_nohuge(addr, size);
         }
```

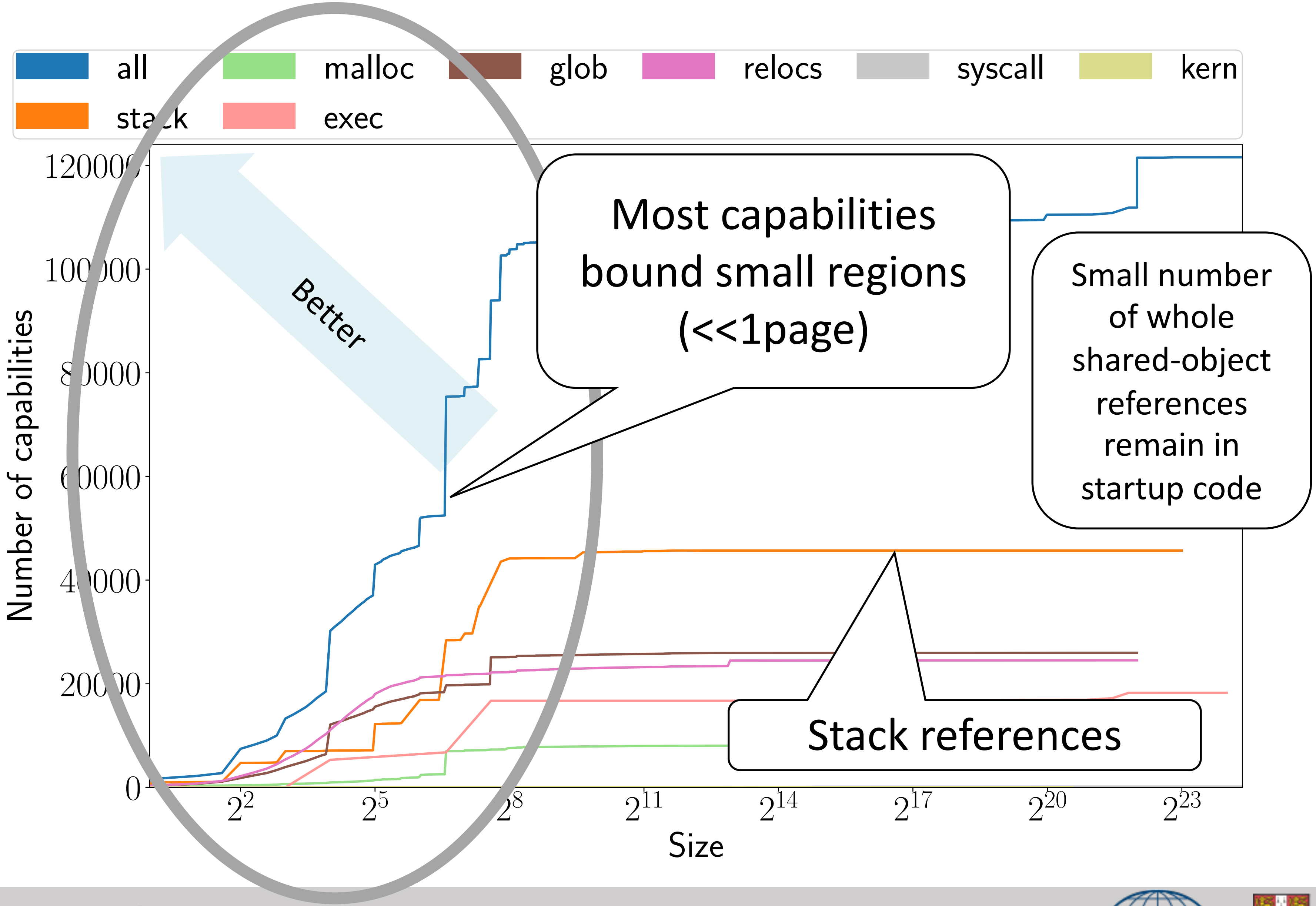
Required changes: pointer provenance

```
        if ((nstrings = realloc(we->we_strings, we->we_nbytes)) ==
            NULL) {
                error = WRDE_NOSPACE;
                goto cleanup;
        }
        for (i = 0; i < vofs; i++)
-               if (we->we_wordv[i] != NULL)
-                       we->we_wordv[i] += nstrings - we->we_strings
    ;
+               if (we->we_wordv[i] != NULL) {
+                       we->we_wordv[i] = nstrings +
+                           (we->we_wordv[i] - we->we_strings);
+               }
        we->we_strings = nstrings;
```

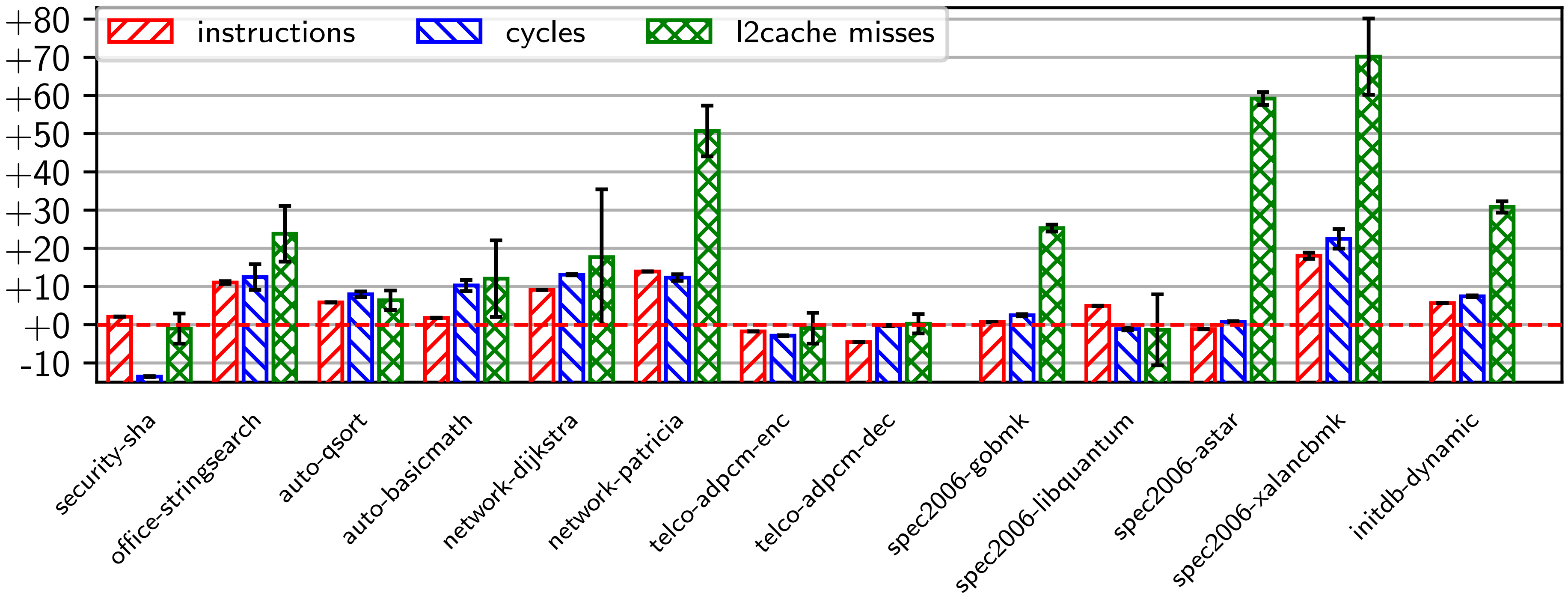
Required changes: summary

- Userspace: 1% (~200) of files required changes
  - Concentrated in libraries
  - Most programs require no changes
- Kernel: <6% of files (~750) required changes
  - Pervasive changes to `iovec`, signal handlers, network interface `ioctl` handlers
  - A pure-capability kernel could reduce changes
- Many changes improve code quality
  - We have upstreamed many to FreeBSD (compat32 improvements, etc)

Capability bounds minimization (OpenSSL)



Performance



- Micro-benchmark performance generally acceptable
  - <10% overhead in most cases
  - Graph excludes crypto and bit-manipulation outliers

Reflections on using FreeBSD for CheriABI

- Good:
  - Well-abstracted process ABI infrastructure
    - SysV stack ABI somewhat baked in
  - Central, generated system call tables, stubs, etc
  - Single, hackable build system
- Bad
  - Centralized `copyin`/`copyout` for `ioctl` divorces copy from types
  - Tests require ports/packages (kyua)
    - No easy way to build kyua static

Work in progress

- Porting ISA from MIPS64 to RISC-V
- New compressed capability format
- Temporal memory safety
- Make CheriABI the default ABI
  - Add a compat/freebsd64
- Pure-capability kernel

Future work on FreeBSD

- More compatX cleanup
  - Code deduplication
  - Remove separate syscalls.master
- Rework ioctl interface
  - Konrad Witaszczyk (def@) is working in this area
- Refactor use of initial stack for arguments
  - Needed for CheriABI, likely helpful for ASLR
- Upstream CHERI/CheriABI support
  - Hardware platform required, but hopefully coming

Conclusions

- Full UNIX-like operating system with spatial and referential memory safety
  - Covers programs, libraries, and linkers
  - Kernel access to user memory
- Some fundamental operating system changes required
  - Generally non-disruptive
- 3rd-party software works: PostgreSQL database, Webkit

Further Reading

http://cheri-cpu.org/

Watson, et al., **Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 7),** Technical Report UCAM-CL-TR-927, Computer Laboratory, Cambridge UK, October 2018.

Davis, et al., **CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment (Extended Version),** Technical Report UCAM-CL-TR-932, Computer Laboratory, Cambridge UK, January 2019.

Woodruff, et al., **CHERI Concentrate: Practical Compressed Capabilities,** IEEE Transactions on Computers, (forthcoming).

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA).
The views, opinions and/or findings expressed are those of the author 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.