CTSRD

CRASH-worthy Trustworthy Systems Research and Development

CheriBSD

Hybrid-capability OS prototype

Robert N. M. Watson

Brooks Davis, Simon W. Moore, Peter G. Neumann, Jonathan Woodruff, Jonathan Anderson, Hadrien Barral, Ruslan Bukin, David Chisnall, Nirav Dave, Lawrence Esswood, Khilan Gudka, Alexandre Joannou, Chris Kitching, Ben Laurie, A. Theo Markettos, Alan Mujumdar, Steven J. Murdoch, Robert Norton, Philip Paeps, Alex Richardson, Michael Roe, Colin Rothwell, Hassen Saidi, Stacey Son, Munraj Vadera, Hongyan Xia, and Bjoern Zeeb

University of Cambridge, SRI International

CHERI Microkernel Workshop – 23 April 2016

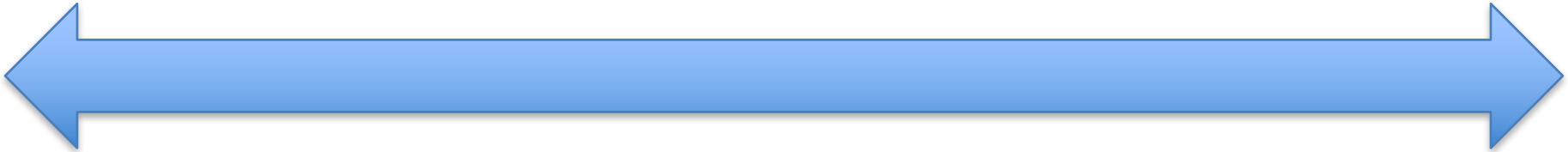
Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contracts FA8750-10-C-0237 ('CTSRD') and FA8750-11-C-0249 ('MRC2'). The views, opinions, and/or findings contained 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.

SRI International

University of Cambridge

CHERI software models

More compatible — Safer



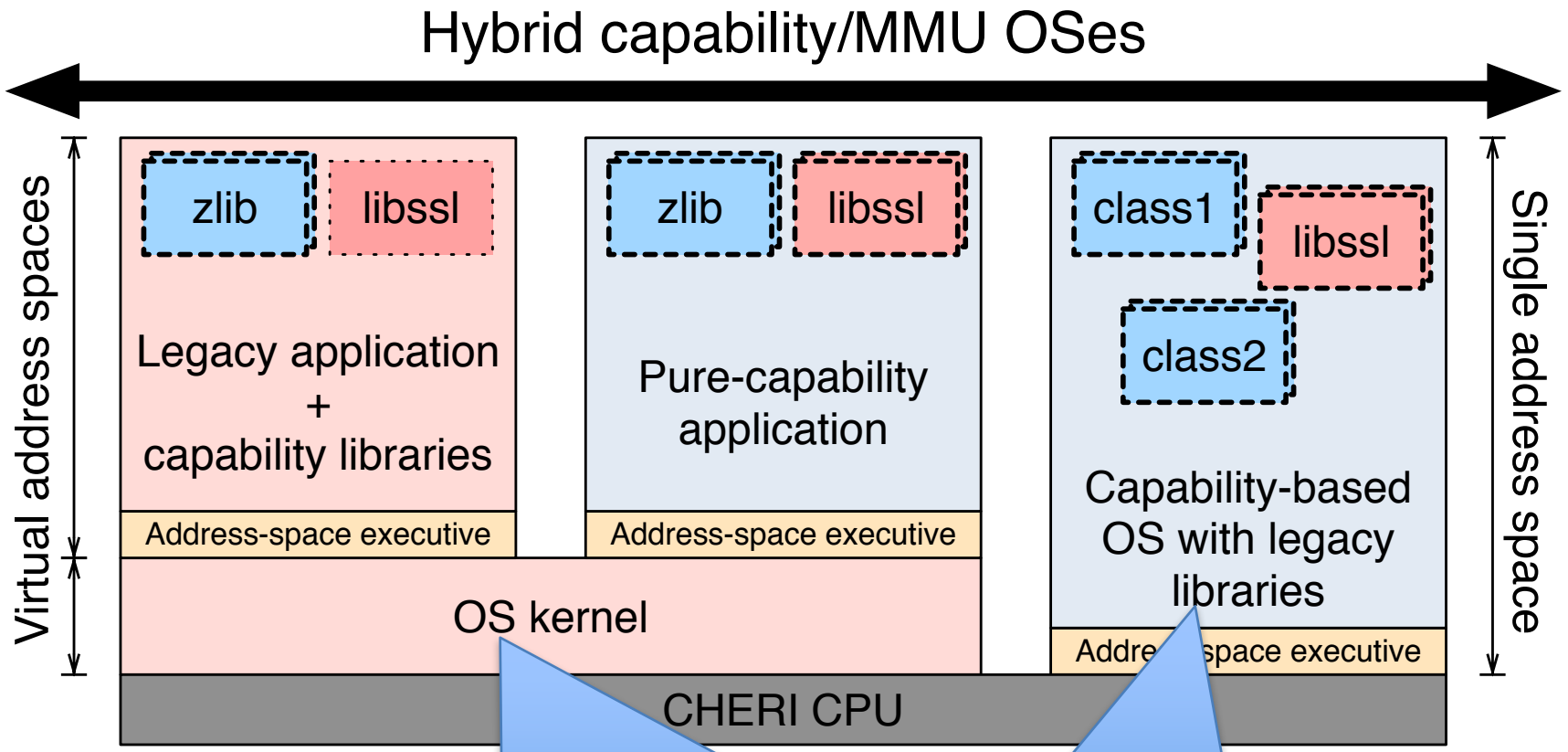
Unmodified
All pointers are registers

Hybrid
Annotated and automatically selected pointers are capabilities

Pure-capability
All code and data pointers are capabilities

- **Source and binary compatibility**: common **C-language idioms,** various **ABIs**
  - **Unmodified code**: Existing n64 code runs without modification
  - **Hybrid code**: e.g., used solely in return addresses, for annotated data/code pointers, for specific types, stack pointers, etc.; n64-interoperable.
  - **Pure-capability code**: ubiquitous data-pointer protection, strong Control Flow Integrity (CFI). Non-n64-interoperable.
- **CHERI Clang/LLVM prototype** generates code for all three

Software deployment models



Hybrid MMU-capability models: protection and compartmentalization within virtual address spaces

Single-address-space systems are possible but not yet our focus

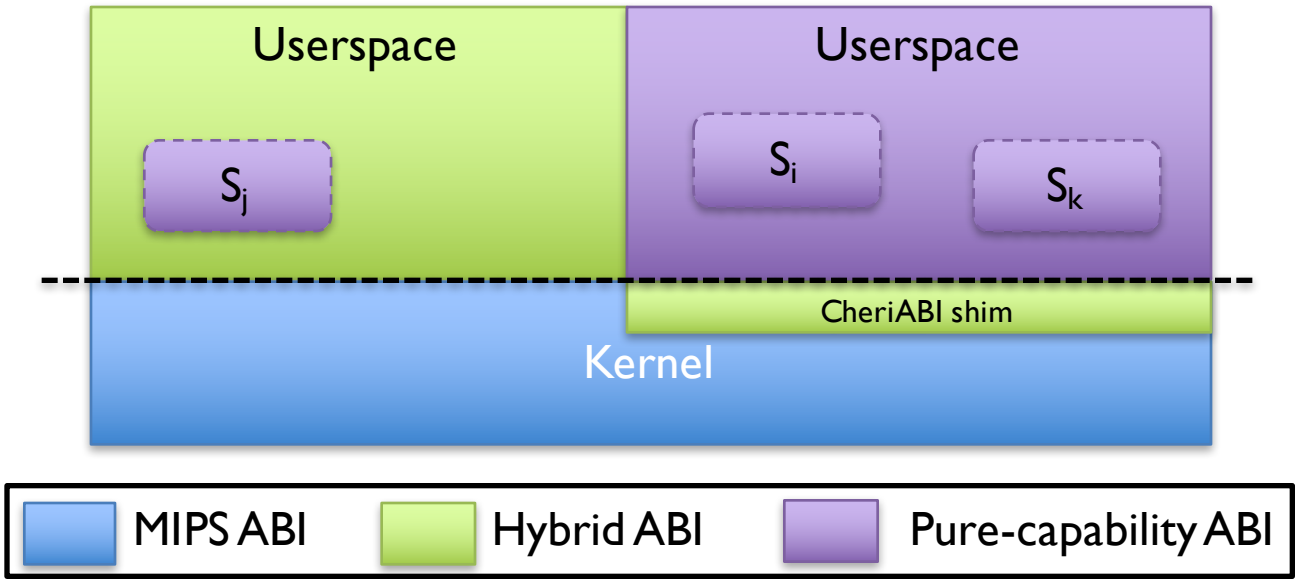
Key software hypotheses

- Viable composition of capability and MMU models for software
  - E.g., CHERI complements paged VM in practical systems
- CHERI capabilities can be usefully applied to program constructs
  - E.g., heap/stack allocations, code pointers, return addresses
  - E.g., kernel-provided memory mappings, static + run-time linking
- Strong binary and source-code compatibility; incremental deployment
  - E.g., selected libraries, applications within a larger system
- Platform for compartmentalization research
  - Libraries/applications are efficiently/easily compartmentalized
  - But also kernel code (in due course)

CheriBSD

- Based on open-source FreeBSD operating system
- "Minimalist" kernel adaptation
  - Process model, VM, debugging, signals support capabilities
  - E.g., thread state includes capability registers
  - E.g., tags preserved for swapped anonymous memory
  - Kernel actually compiled with MIPS, not CHERI, compiler
- Multiple process ABIs: hybrid MIPS and CheriABI
- Fine-grained, in-address-space compartmentalization model
  - Kernel-assisted domain transition, fault handling
  - libcheri object-capability runtime

Multiple process ABIs



- **64-bit MIPS ABI** supports highly compatible hybrid code execution, traditional pointer-based system calls
- **CheriABI** binaries/processes are pure-capability code throughout; system-call interface enforces user model

Demonstration applications

- Pure-capability libraries and applications
  - Pure-capability compilation of all key system libraries and increasing number of commands – e.g., OpenSSL, OpenSSH
  - Strong memory protection for heap, stack; control-flow integrity for minimally modified or unmodified applications
- Library compartmentalization
  - Transparent, efficient sandboxing of security-critical libraries
- tcpdump compartmentalization
  - Fine-grained: multiple domain transitions per packet
- Otherwise (essentially) unmodified userspace

Next directions

- Short-term: complete pure-capability userspace
  - CHERI-aware run-time linking, multithreading
  - Remainder of C (and C++) pure-capability userspace
  - LLDB debugger support
- Short-term: selected capability use and CFI within the kernel
  - E.g., in CheriABI, for user-originated pointers, network stack
- Longer-term: selectively compartmentalized kernel
  - CHERI-based microkernel within CheriBSD kernel
- Longer-term: non-volatile memory + capabilities
  - Semantics for tagged capabilities within filesystem objects

BACKUP SLIDES

Kernel Changes

Component	File	Lines +	Lines -
Machine-dependent headers	19	1424	11
CHERI initialization	2	49	4
Context management	2	392	10
Exception handling	3	574	90
Memory copying	2	122	0
Virtual memory	5	398	27
Object capabilities	2	883	0
System calls	2	76	0
CheriABI	6	2855	0
Signal delivery	3	327	71
Process monitoring/debugging	3	298	0
Kernel debugger	2	264	0