CTSRD

CRASH-worthy Trustworthy Systems Research and Development

CHERI

A Hybrid Capability-System Architecture for Scalable Software Compartmentalization

Robert N.M. Watson[*], Jonathan Woodruff[*], Peter G. Neumann[†], Simon W. Moore[*], Jonathan Anderson[‡], David Chisnall[*], Nirav Dave[†], Brooks Davis[†], Khilan Gudka[*], Ben Laurie[§], Steven J. Murdoch[¶], Robert Norton[*], Michael Roe[*], Stacey Son, and Munraj Vadera[*]

[*]University of Cambridge, [†]SRI International, [‡]Memorial University,
[§]Google UK Ltd, [¶]University College London

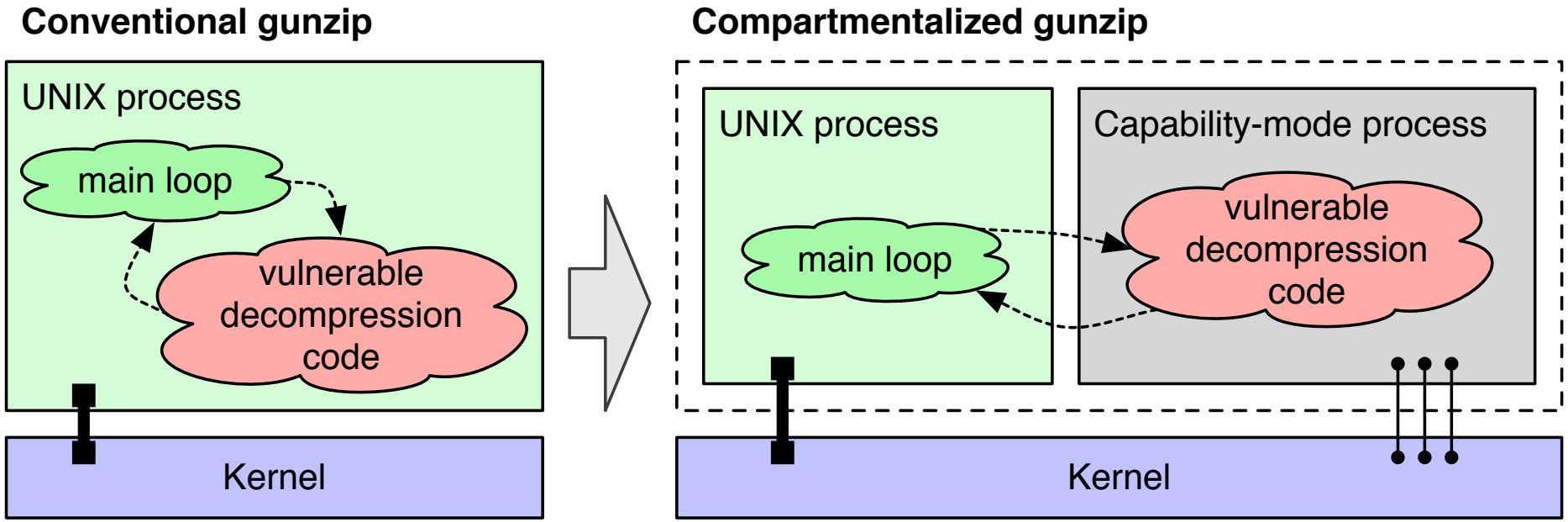
IEEE Symposium on Security and Privacy
18 May 2015

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.

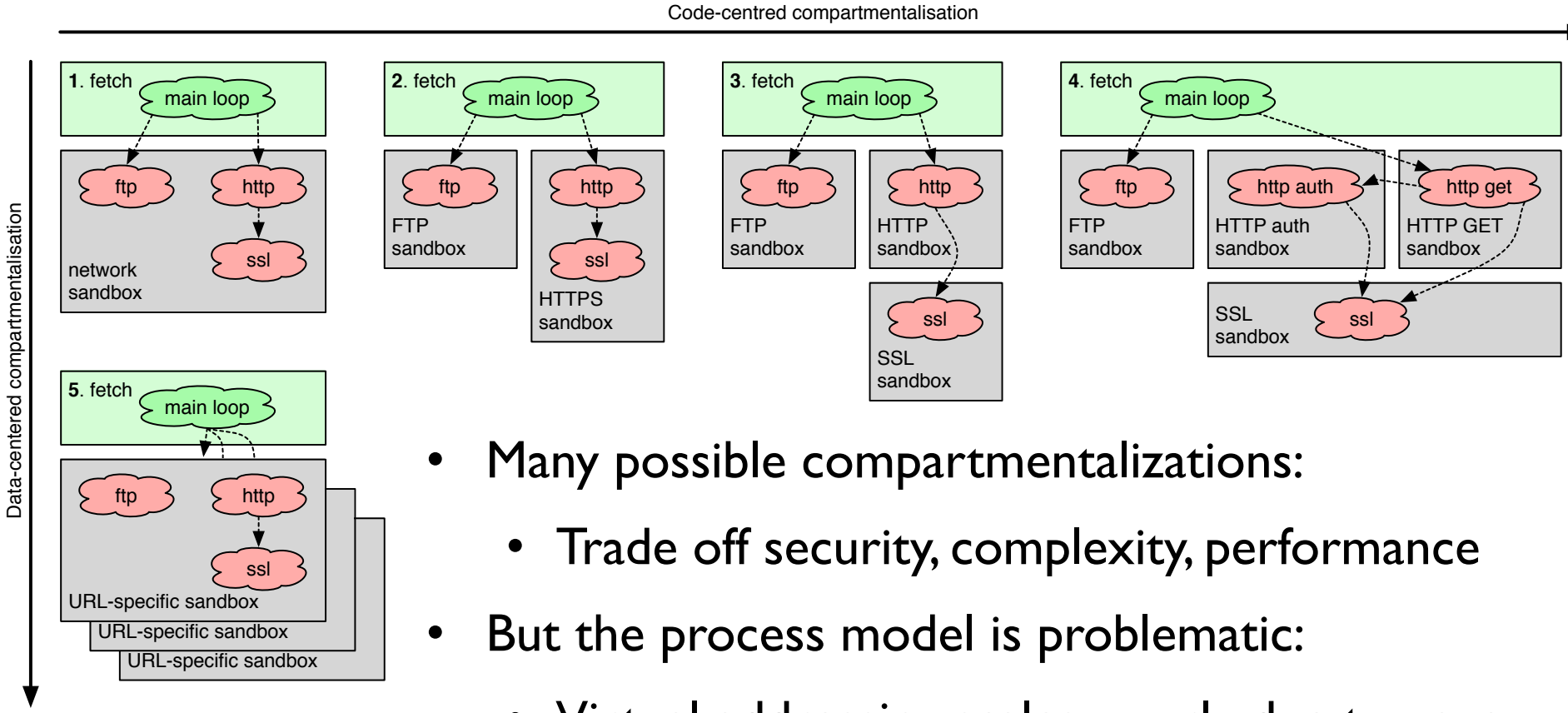
SRI International

University of Cambridge

Application compartmentalization



Application compartmentalization mitigates vulnerabilities by decomposing applications into **isolated compartments** delegated **limited rights**



- Many possible compartmentalizations:
  - Trade off security, complexity, performance
- But the process model is problematic:
  - Virtual addressing scales poorly due to page tables, Translation Look-aside Buffer (TLB)
  - Multiple address spaces and Inter-Process Communication (IPC) are hard to program
- Quite poor for **library compartmentalization** due to memory-centered APIs (e.g, zlib)

CHERI capability model

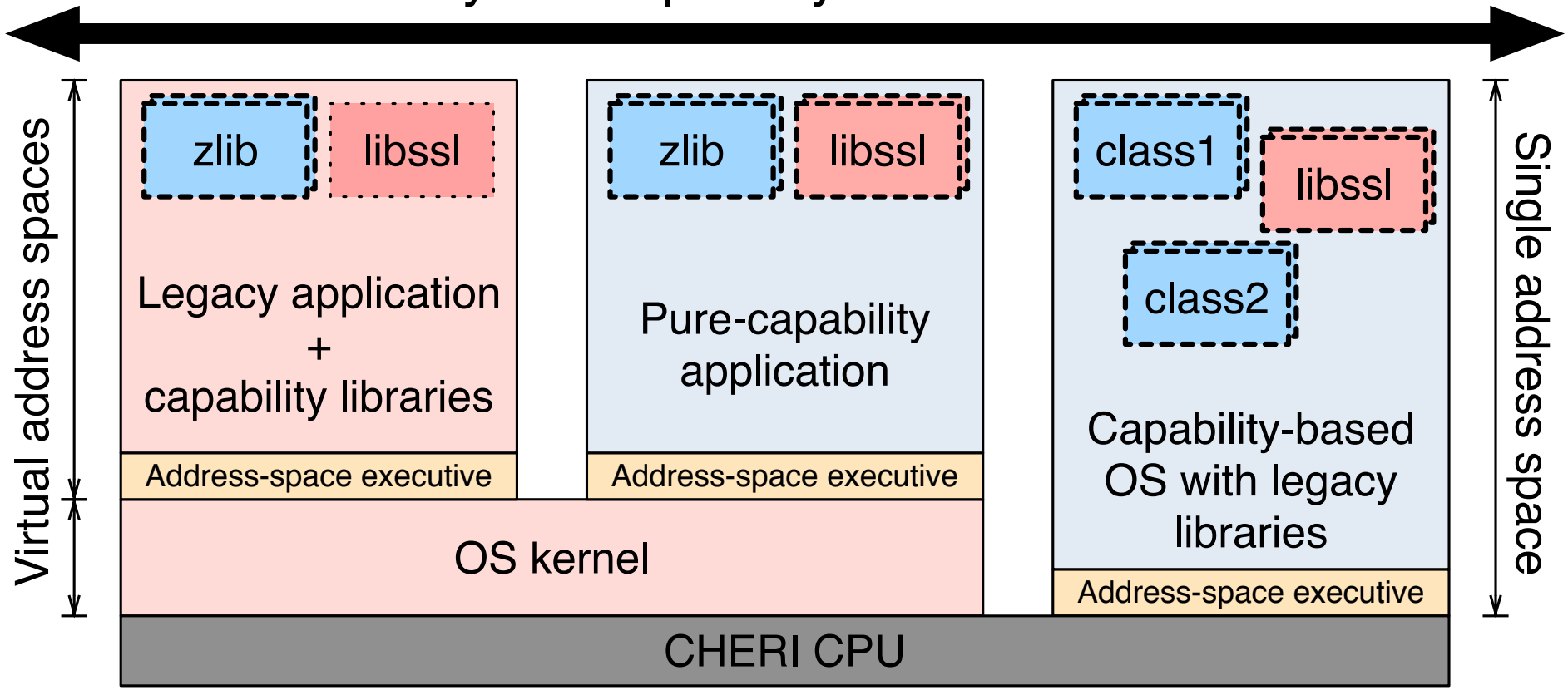
- **ISCA 2014**: Fine-grained, in-address-space memory protection via a capability model
  - **Capabilities** replace pointers for data references
  - **Capability registers** and **tagged memory** enforce strong pointer and control-flow integrity, bounds checking
  - Hybrid model composes naturally with an MMU
- **ASPLOS 2015**: Compiler support for capabilities
  - Converge **fat-pointer** and **capability** models
  - C pointers compiled into capabilities with various ABIs
- **Can we build efficient compartmentalization over CHERI memory protection ?**

Virtual memory vs. capabilities

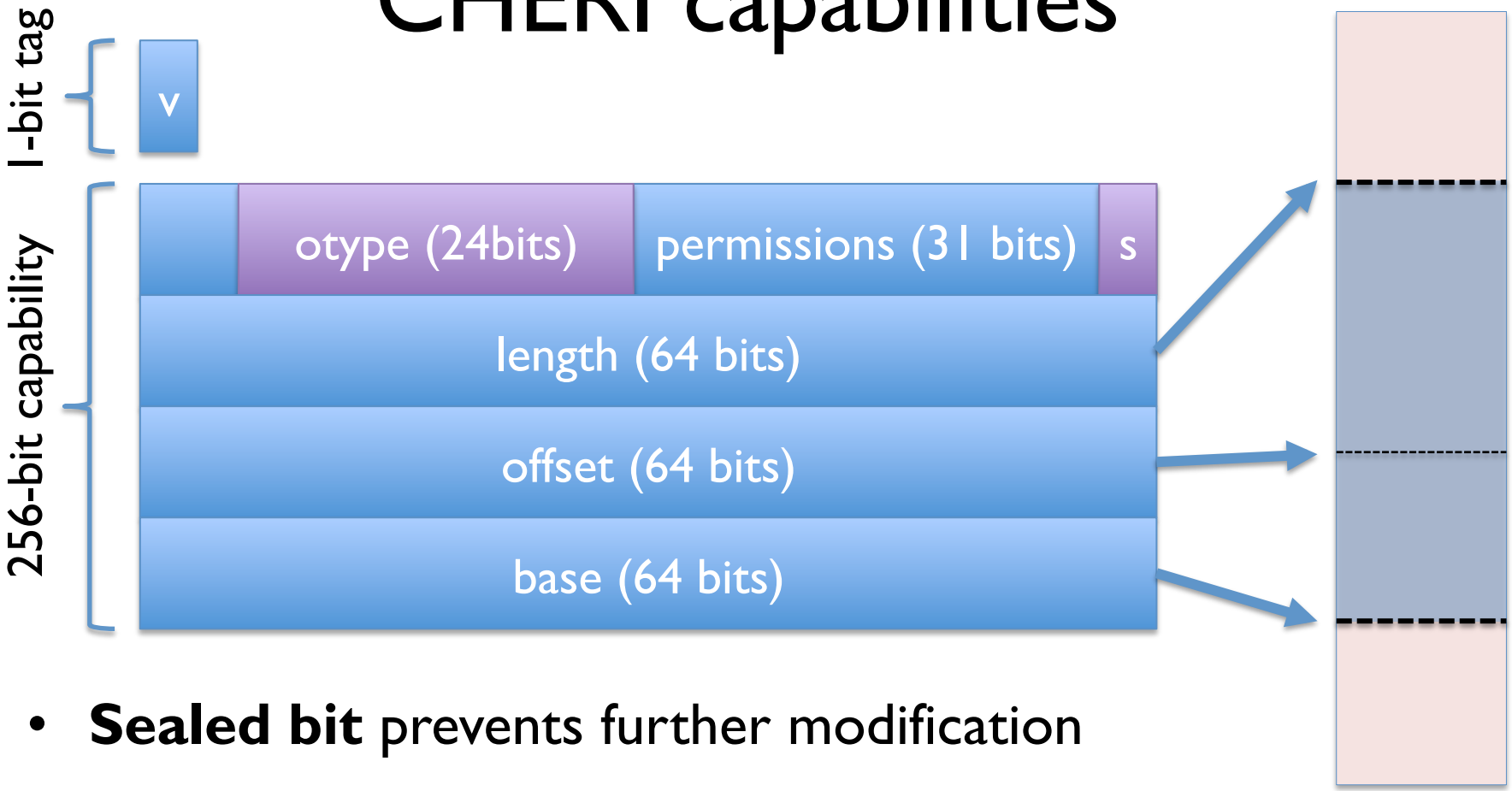
	Virtual Memory	Capabilities
Protects	Virtual addresses and pages	References (pointers) to C code, data structures
Hardware	MMU, TLB	Capability registers, tagged memory
Costs	TLB, page tables, lookups, shootdowns	Per-pointer overhead, context switching
Compartment scalability	Tens to hundreds	Thousands or more
Domain crossing	IPC	Function calls
Optimization goals	Isolation, full virtualization	Memory sharing, frequent domain transitions

CHERI hybridizes the models: pick two!

Hybrid capability/MMU OSes

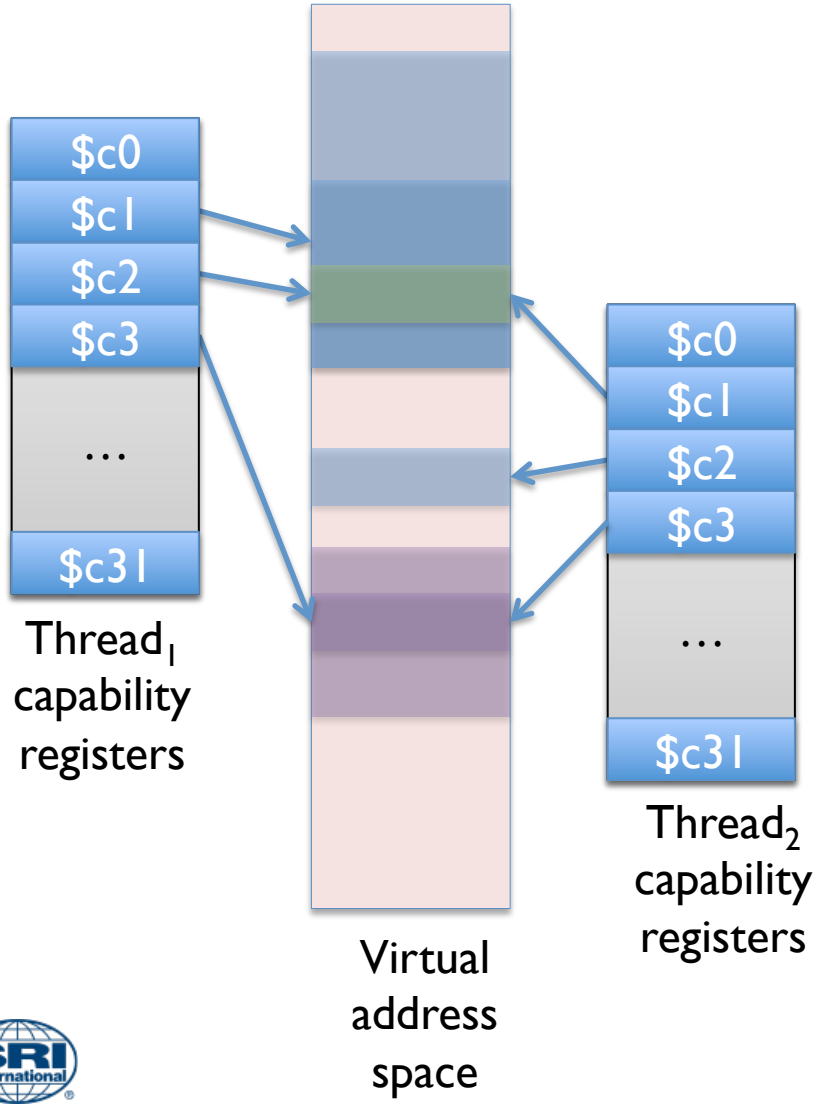


CHERI capabilities



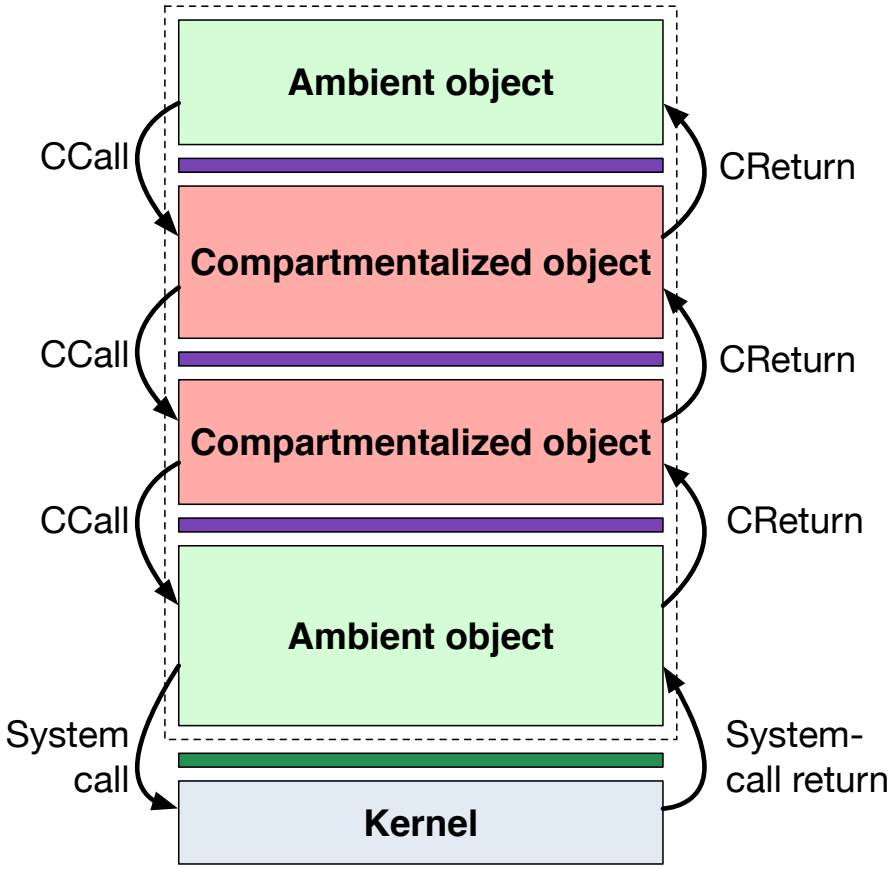
- **Sealed bit** prevents further modification
- **Object types** atomically link code, data capabilities
- **CCall/CReturn** instructions provide hardware-assisted, software-defined domain transitions

CheriBSD object capabilities



- In-process o**bject-capability model**
- libcheri loads and links **classes**, instantiates **objects**
- Per-thread capability register file describes its **protection domain**
- **Domain transition** within threads via register-file transformation
- **CCall/CReturn** exception handlers unseal capabilities, allow delegation
- **Trusted stack** provides reliable software-defined return, recovery
- Many other software-defined models possible; e.g., asynchronous closures

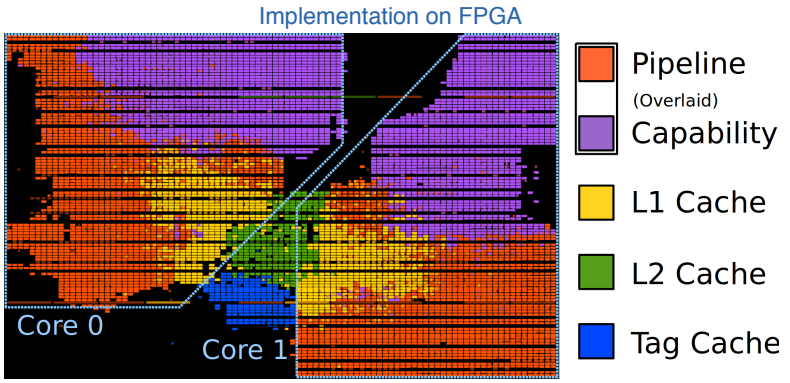
Object-capability call/return

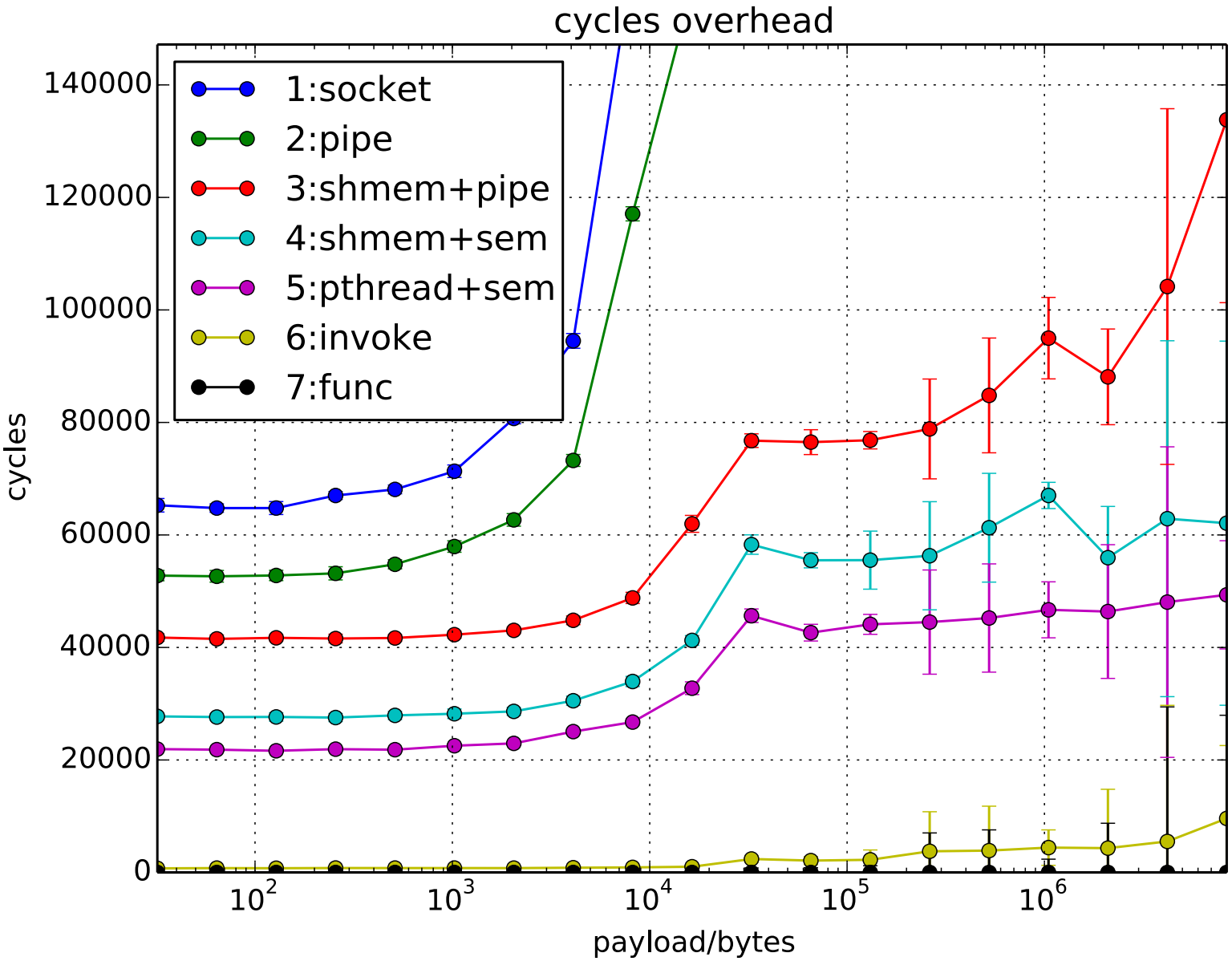


- Initial registers after execve() grant ambient authority
- Synchronous function-like call eases application/library adaptation
- CCall/CReturn ABI clears unused registers to prevent leakage
- Only authorized system classes can make system calls
- Constant overhead to function-call cost

CHERI hardware/software prototypes

- Bluespec FPGA prototype
  - 64-bit MIPS + CHERI ISA
  - Pipelined, L1/L2 caches, MMU
  - Synthesizes at ~100MHz
- Capability-aware software
  - CheriBSD OS
  - CHERI clang/LLVM compiler
  - Adapted applications
- Open-source release




cycles overhead
1:socket
2:pipe
3:shmem+pipe
4:shmem+sem
5:pthread+sem
6:invoke
7:func
cycles
payload/bytes
0
20000
40000
60000
80000
100000
120000
140000
10^2
10^3
10^4
10^5
10^6

Application implications

Pros

- Single address-space programming model
- Referential integrity matches programmer model
- Modest work to insert protection-domain boundaries
- Objects permit mutual distrust
- Constant (low) overhead relative to function calls even with large memory flows

Cons

- Still have to reason about the security properties
- Shared memory is more subtle than copy semantics
- Capability overhead in data cache is real and measurable
- ABI subtleties between MIPS and CHERI compiled code
- Lower overhead raises further cache side-channel concerns

Conclusions

- Hybrid object-capability model over memory capabilities
  - Software-defined, fine-grained, in-address-space compartmentalization
  - Cleanly extends the MMU-based process model
  - Targets C-language userspace TCBs
  - Non-IPC model supports library compartmentalization
  - Orders of magnitude more efficient compartmentalization that conventional designs
- Open-source reference implementation, ISA specification:

  http://www.cheri-cpu.org/