

CHERI

A Hybrid Capability-System Architecture for Fine-grained Memory Protection and Compartmentalization

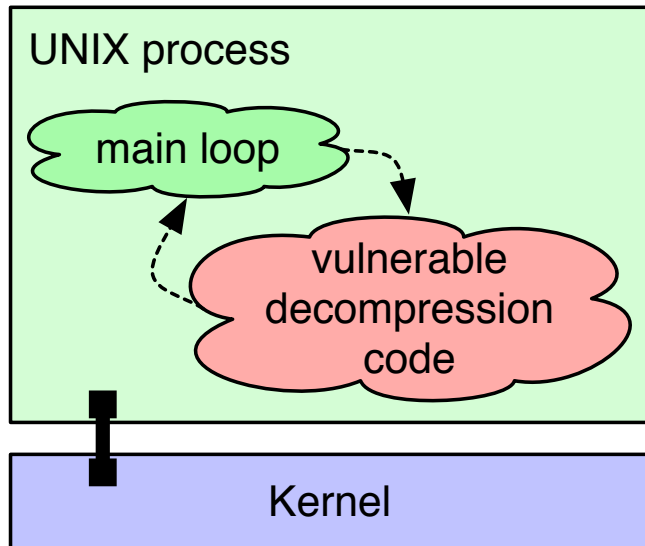
Robert N.M. Watson, Peter G. Neumann, Jonathan Woodruff, Simon W. Moore,
Jonathan Anderson, Ruslan Bukin, David Chisnall, Nirav Dave, Brooks Davis,
Khilan Gudka, Alexandre Joannou, Chris Kitching, Ben Laurie, A.Theo Markettos,
Alan Mujumdar, Steven J. Murdoch, Robert Norton, Michael Roe, Colin Rothwell,
Stacey Son, Munraj Vadera, and Bjoern Zeeb

University of Cambridge, SRI International

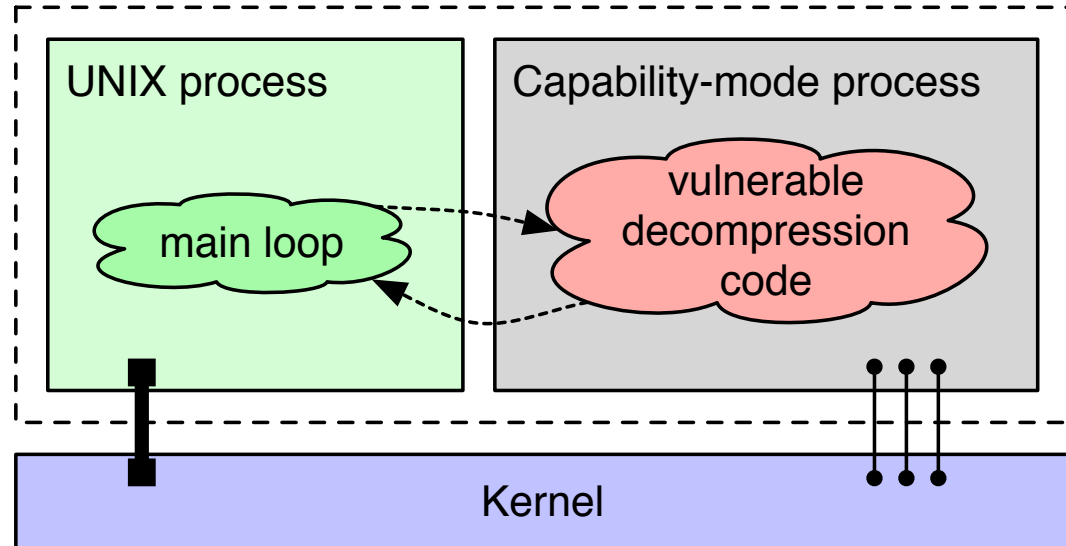
May 2015

Application compartmentalization

Conventional gunzip



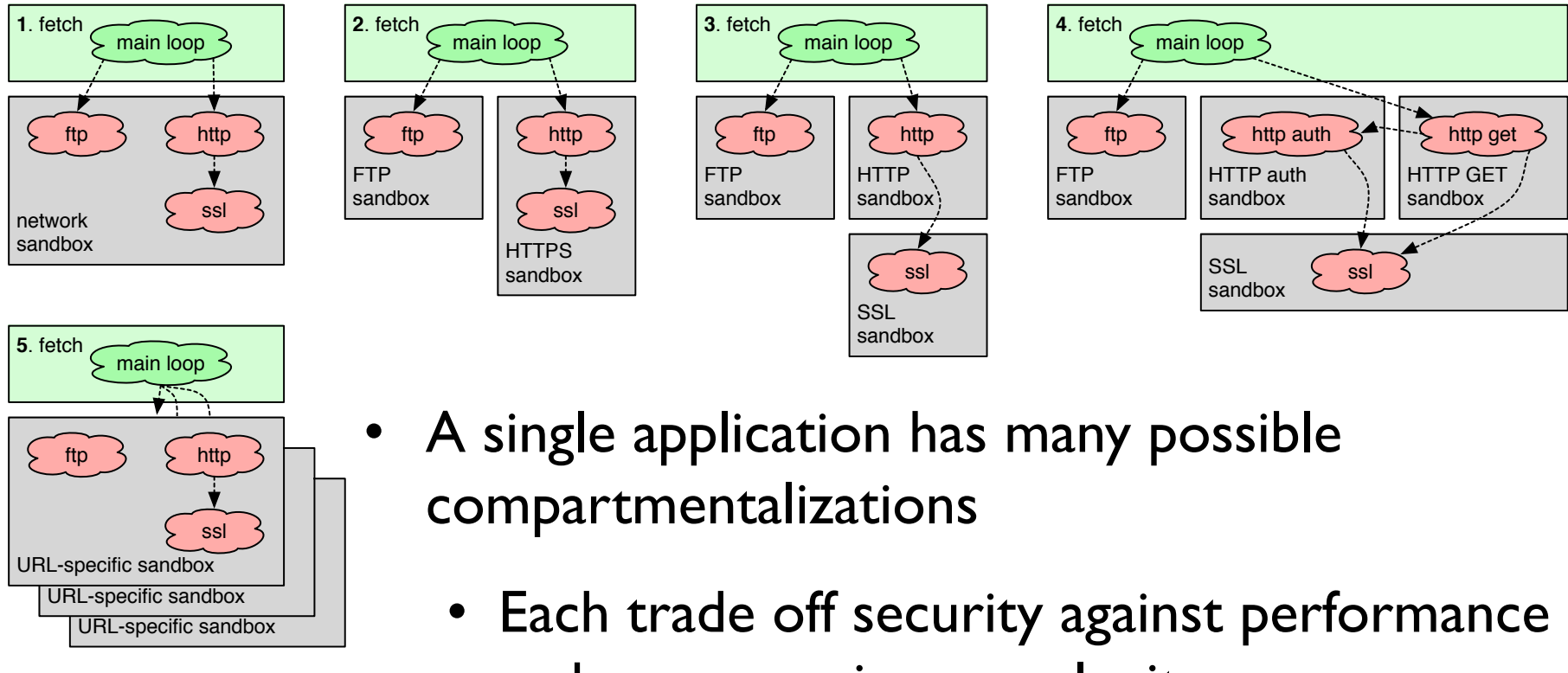
Compartmentalized gunzip



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

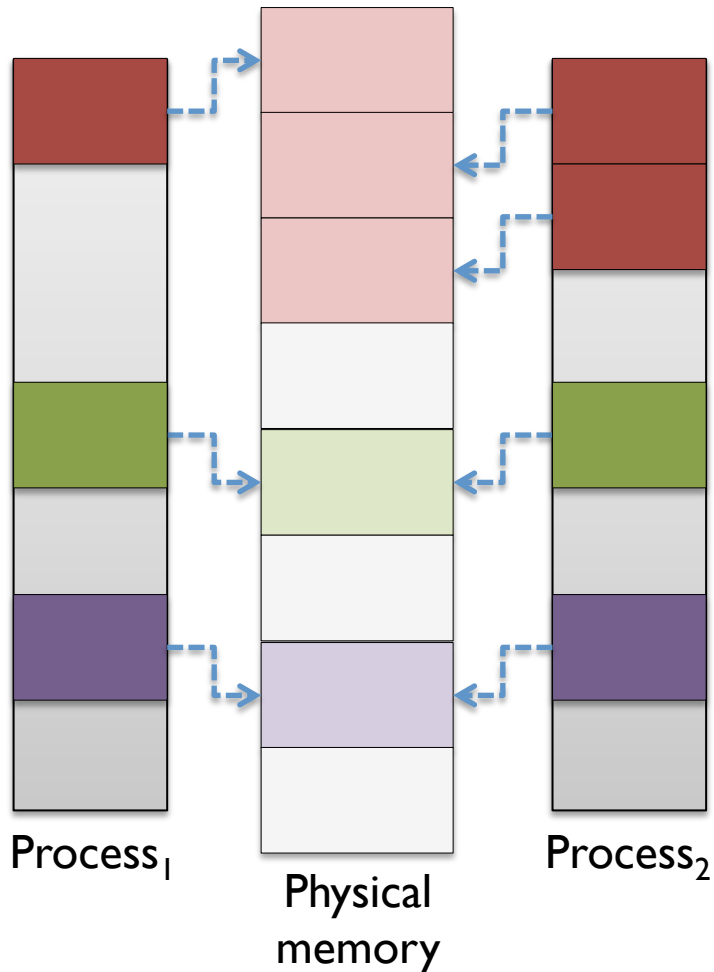
Compartmentalization vision





- A single application has many possible compartmentalizations
 - Each trade off security against performance and programming complexity
- But the process model is problematic:
 - Limited simultaneous-process scalability
 - Multi-address-space programming model

The process-model consensus



- Coarse-grained process isolation
 - Inter-program robustness
 - Multi-user access control
- Memory Management Unit (MMU)
 - Page tables control per-process virtual-to-physical mappings
 - Translation Look-aside Buffer (TLB)
- Bridged via Inter-Process Communication (IPC) and other kernel services (e.g., filesystem)
- **Inefficient and inadequate foundation for granular memory protection**
- **Inefficient and hard-to-program foundation for granular software compartmentalization**

If you could revise the
fundamental principles of
computer system design
to improve security...

...what would you change?

CHERI capability model

- **ISCA 2014**: Fine-grained, in-address-space memory protection
 - **Capabilities** replace pointers for data references
 - **Data-pointer integrity, control-flow integrity, bounds checking**
 - **Hybrid model** lives side-by-side with a conventional MMU
- **ASPLOS 2015**: Explore and refine C-language compatibility
 - Converge **fat-pointer** and **capability** models
 - Develop **binary-compatibility models**
- **Oakland 2015**: Software compartmentalization with CHERI
 - **Object-capability model** implemented over hardware capabilities
 - Efficient, in-address-space **software-defined domain transition**

CHERI MEMORY PROTECTION

Revisiting RISC in an age of risk

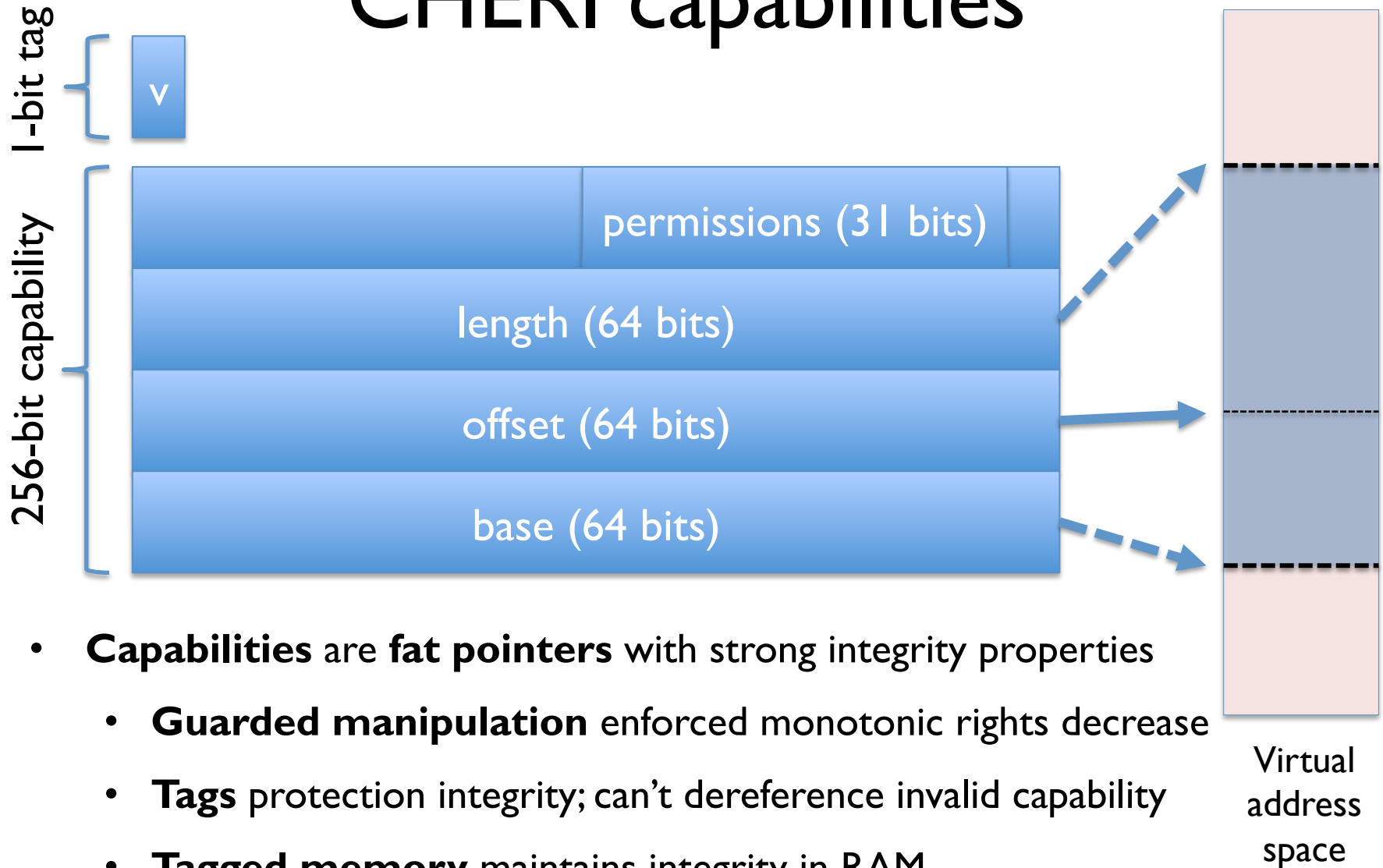
- Fine-grained, capability-based in-address-space protection
 - Deconflate virtualization and protection – retain MMU
 - Implement spatial protection (e.g., bounds checking)
 - Foundation for temporal protection (e.g., GC)
 - Foundation for fine-grained compartmentalization
- A RISC approach
 - Instructions are for compilers
 - Unprivileged fast paths, software slow paths
 - Prototype on 64-bit MIPS, but more broadly applicable

Virtual memory vs. capabilities

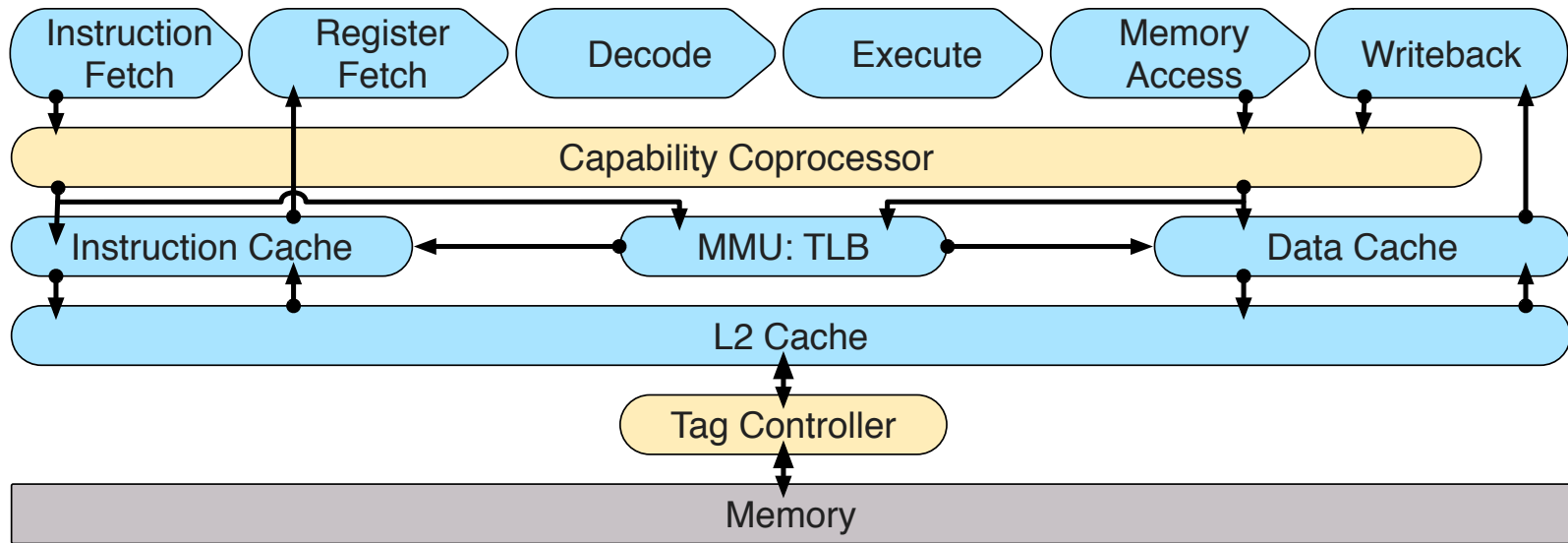
	Virtual Memory	Capabilities
Protects	Virtual addresses and pages	References (pointers) to C code, data structures
Hardware	MMU, TLB	Capability registers, tagged memory
Costs	TLB, page tables, lookups, shutdowns	Per-pointer overhead, context switching
Compartment scalability	Tens to hundreds	Thousands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

CHERI hybridizes these models: pick two!

CHERI capabilities

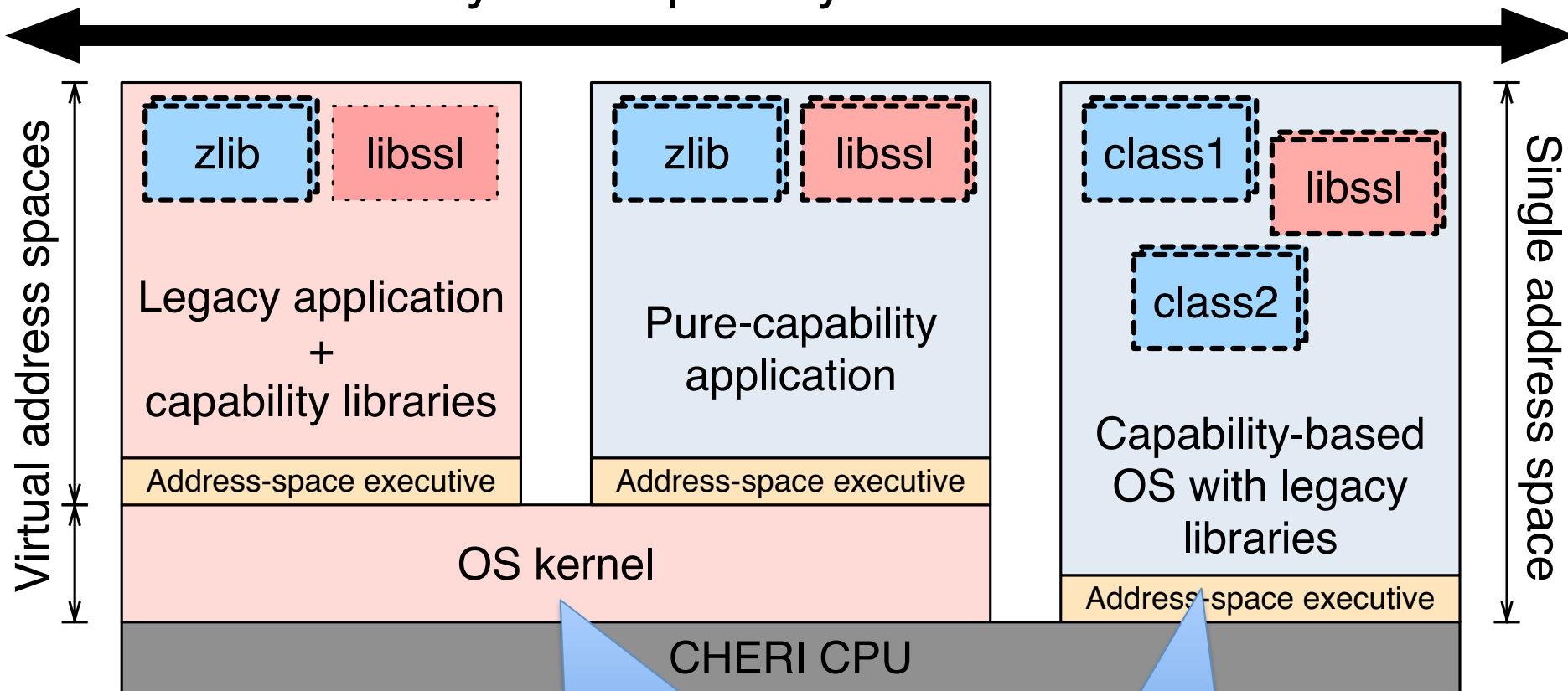


Capability extensions to pipeline



- Capability coprocessor provides capability registers, instructions
- Interposes on legacy MIPS load/store instructions, instruction fetch
- Processing 'before' MMU makes capabilities address-space relative
- Tag controller associates tags with in-memory capabilities
- Under the hood: memory partitioned, with a region holding all tags

Hybrid capability/MMU OSeS



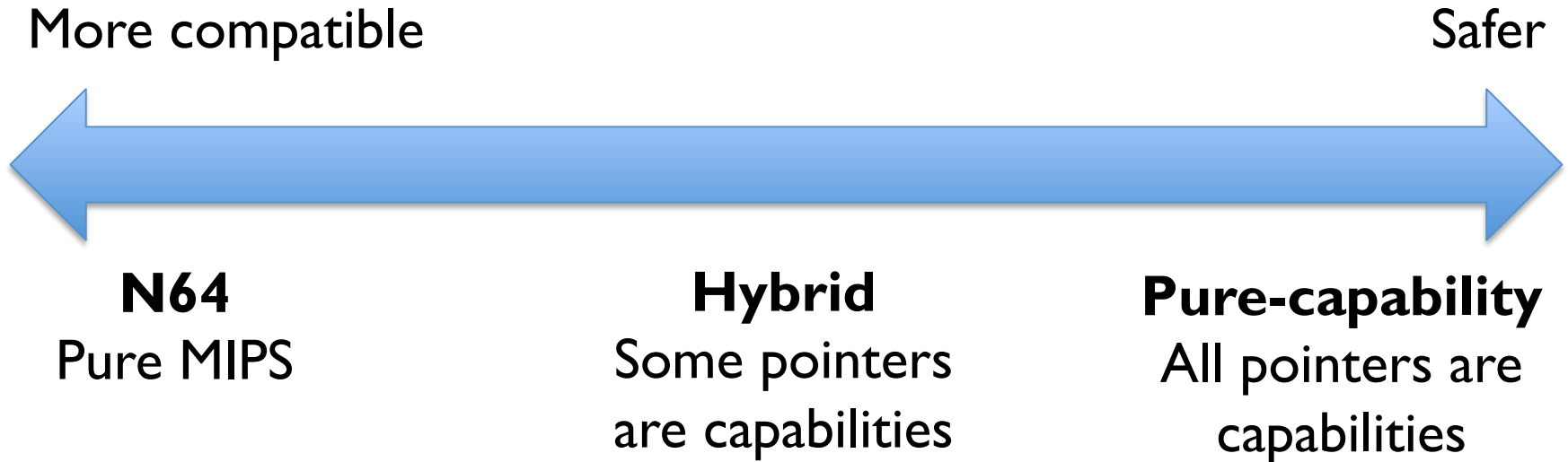
Our focus: hybridizing conventional virtual-memory security and in-address-space capability systems

Single-address-space systems are also possible on CHERI

C-language memory protection

- Compiler implements C pointers in terms of capabilities
- Strong pointer integrity and use limits
 - Tag detects any pointer corruption
 - Bounds checking, with subsetting
 - Permissions constrain use (e.g., read-only, W^X)
- Protects data and control flow integrity
 - Out-of-bounds data access eliminated
 - Data-pointer confusion eliminated
 - Support for accurate garbage collection

Binary compatibility



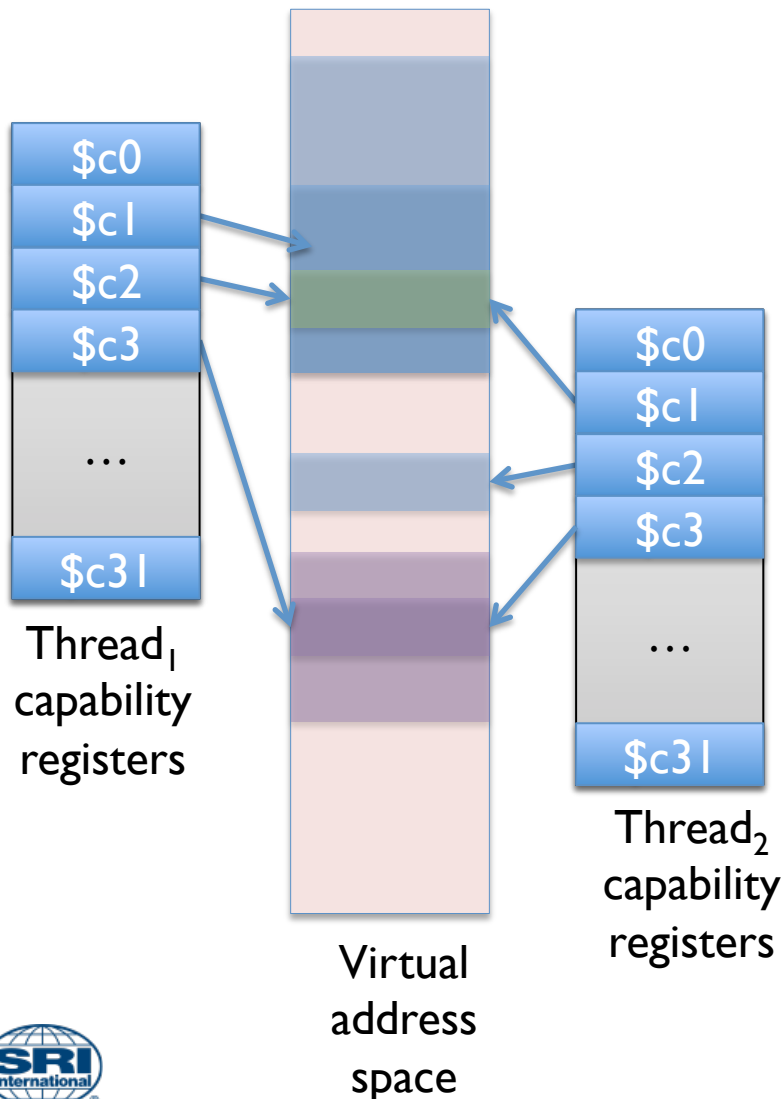
- MIPS code lives side-by-side with CHERI code
- Incremental adoption options – e.g., shifting to capabilities for return addresses, just stack pointers, etc.

CHERI OS considerations

- Prototyped using the FreeBSD operating system (+/- 4 KLoC)
 - Changes for user memory protection, compartmentalization
- Process model extended for tagged capabilities
 - Register-file setup and maintenance (exec, switch, thread create)
 - Virtual-memory support for physical tags
 - Signal-handling, debugging extensions
- Fine-grained, in-address-space object-capability security model
 - Kernel CCall/CReturn exception handlers
 - System calls blocked from non-system classes
 - Userspace compartmentalization runtime

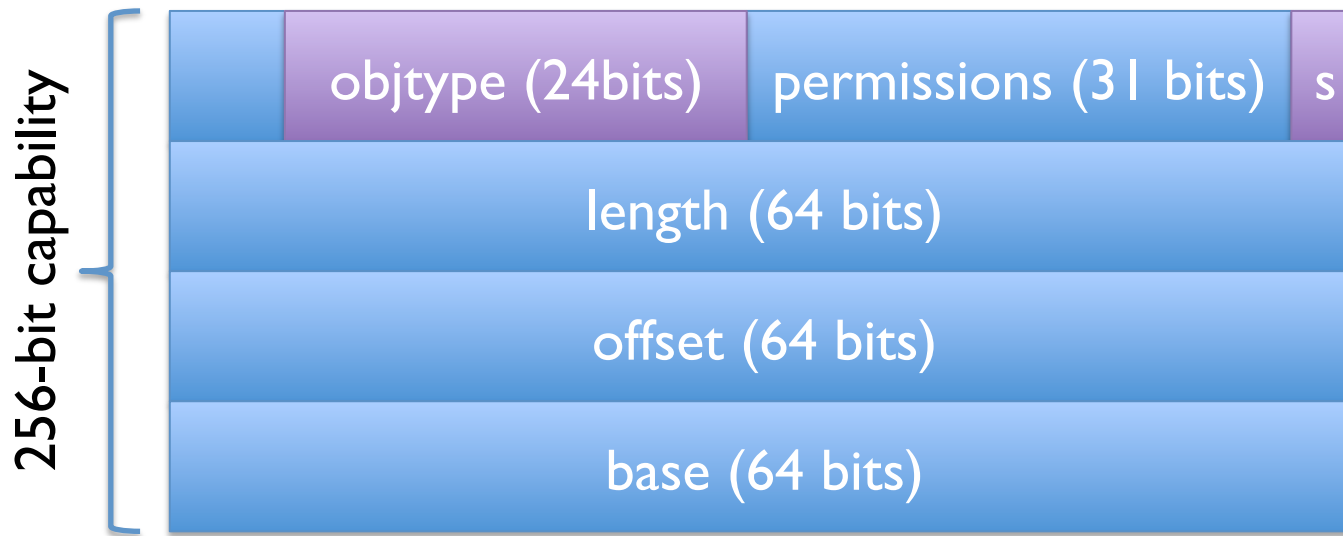
CHERI COMPARTMENTALIZATION

CheriBSD object capabilities



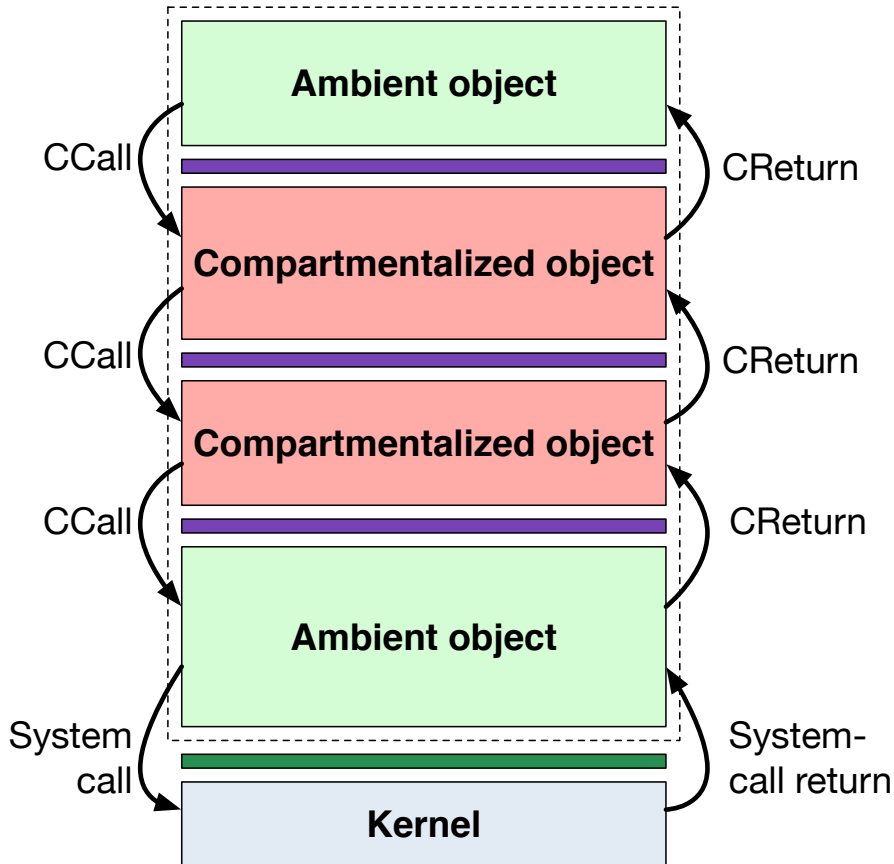
- In-process **object-capability model**
 - Per-thread capability register file describes its **protection domain**
 - **Domain transition** within threads via register-file transformation
 - Object capabilities support strong **encapsulation, mutual distrust**
- libcheri implements **classes, objects**
 - **CCall/CReturn** exception handlers unseal capabilities
 - Capabilities passed via call, return
 - **Trusted stack** provides reliable software-defined return, recovery

Supporting object capabilities



- **Sealed bit** prevents modification, dereferencing: **encapsulation**
- **Object types** atomically link **code**, **data** pairs to create **objects**
- **Object invocation** mechanism **unseals** capability pairs
 - Userspace TCB manages object-type namespace

Object-capability call/return



- Initial registers after `execve()` grant ambient authority
- Synchronous function-like call fits current application/library programming styles
- CCall/CReturn ABI clears unused registers to prevent data or capability leakage
- Only authorized system classes can make system calls
- Constant overhead to function-call cost

Application implications

Pros

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

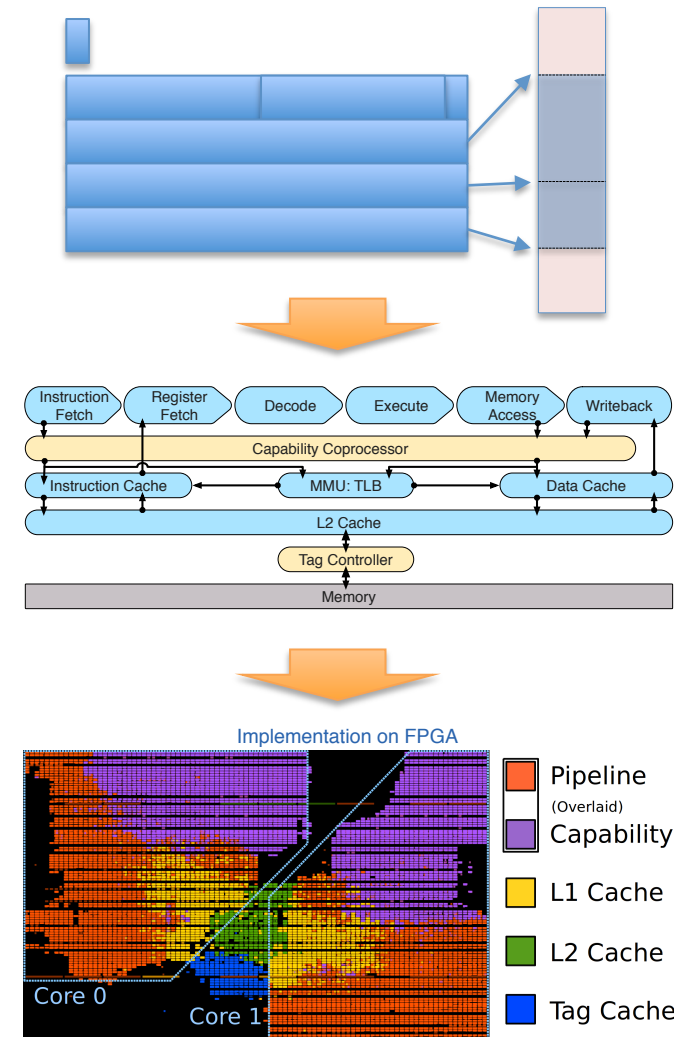
Cons

- 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

CHERI COMPARTMENTALIZATION PERFORMANCE

CHERI hardware/software prototypes

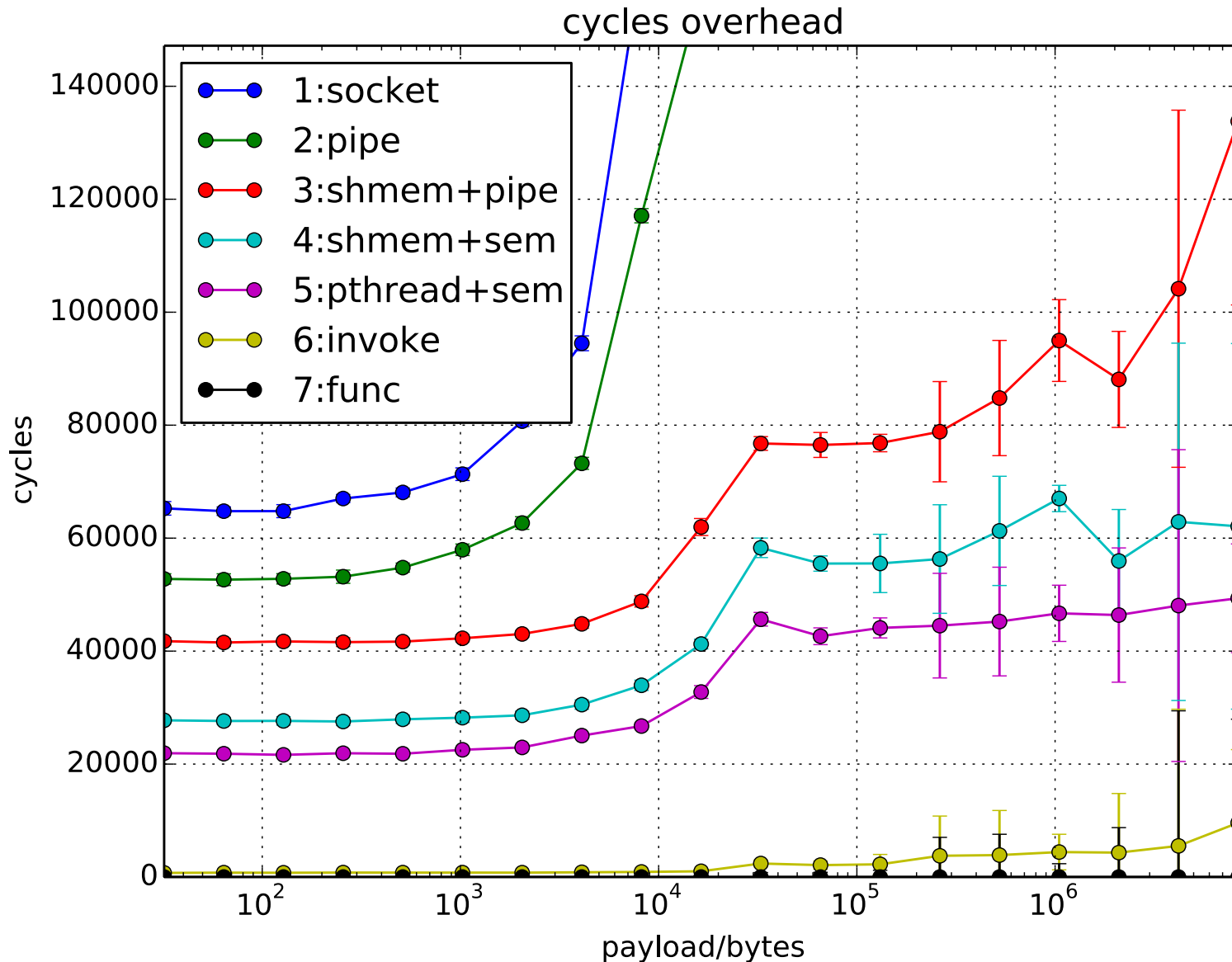
- Bluespec FPGA prototype
 - 64-bit MIPS + CHERI ISA
 - Pipelined, L1/L2 caches, MMU
 - Synthesizes at ~100MHz
 - Realistic (modest) illustration of what could be accomplished in silicon designs
- Capability-aware software
 - CheriBSD OS
 - CHERI clang/LLVM compiler
 - Adapted applications
- Open-source release



Updated ISCA/ASPLOS
results looking at the cache-
footprint question?

Include 128-bit simulation

Domain-switching overhead



Library compartmentalization

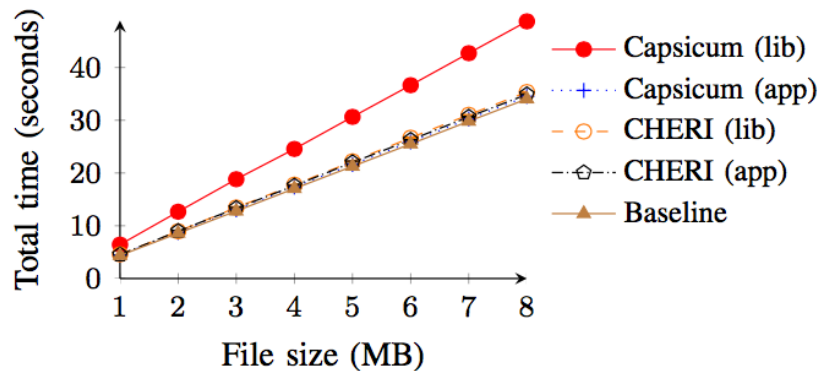


Fig. 12. Compression time for `gzip` with various sandboxing mechanisms

creation and destruction to instantiate and destroy compartments; (2) the amortized and indirect costs of additional virtual address spaces, such as TLB contention and kernel-data-structure footprint on the cache; and (3) the cost of protection-domain switching and implied copying (or MMU operations) to pass data between compartments using IPC.

In some cases, these overheads are negligible or amortized. For example, Capsicum allows processes to acquire access to capabilities while operating with ambient authority, and then to enter capability mode without creating an additional process. For simple program structures, such as in `tcpdump`, Capsicum sandboxing adds only a few extra system calls to limit future

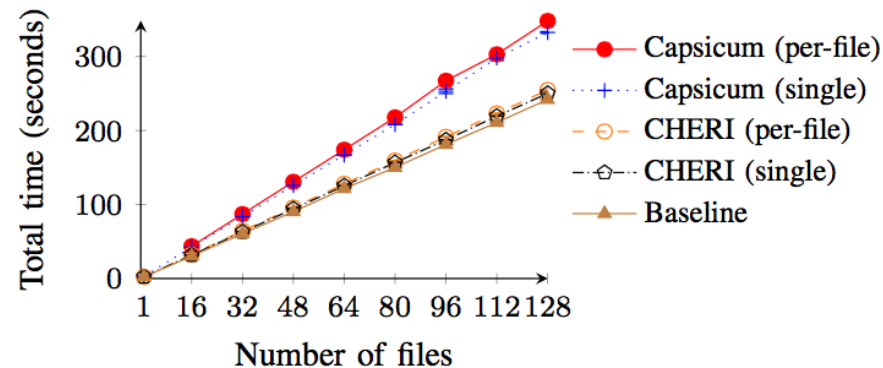


Fig. 13. Sandbox creation overhead: time taken to compress varying number of files of size 500,000 bytes with different `zlib` implementations

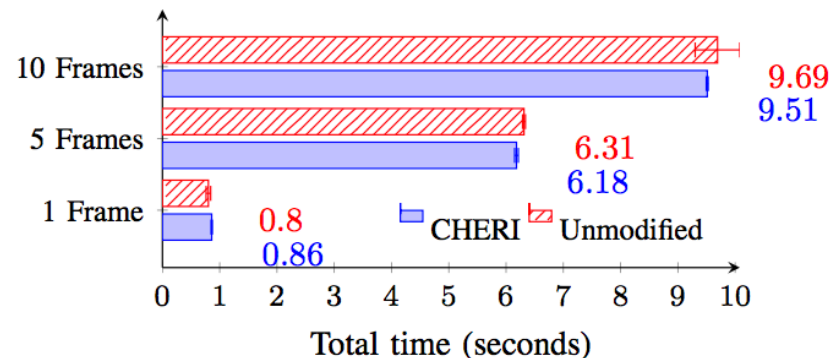


Fig. 14. `gif2png` with unmodified and CHERI `zlib` implementation

ONGOING REFINEMENT TO THE CHERI MODEL

Hardware and software refinement

- Architecture performance / cost
 - Re-converge register files
 - 128-bit compressed capabilities
 - Explore and evaluate application to a non-MIPS ISAs
- Software model and compatibility
 - Compiler and ABI refinement
 - Software compartmentalization models
 - Hybrid system call layer
 - Complete capability-ABI FreeBSD version
 - Reliable and accurate garbage collection

Capability compression

$\text{mask} = 0\text{xffffffffffffffff} \ll \text{exp}$

$\text{top} = (\text{pointer} + (\text{toTop} \ll \text{exp})) \& \text{mask}$

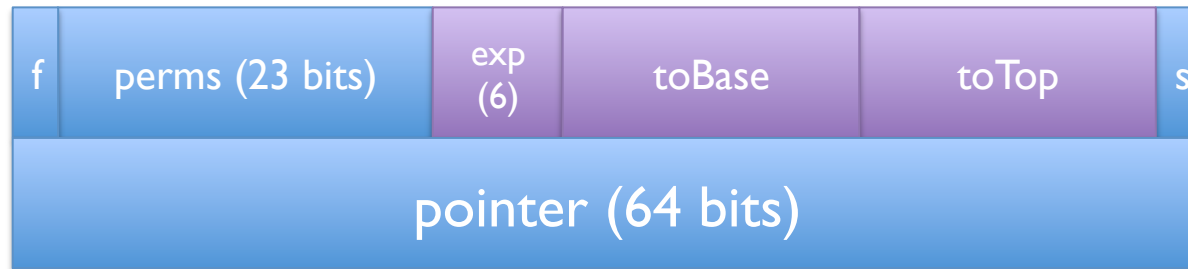
$\text{base} = (\text{pointer} + (\text{toBase} \ll \text{exp})) \& \text{mask}$

1-bit tag

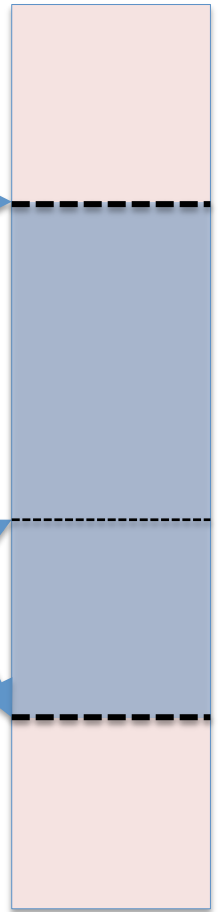


128-bit

capability



- Exchange precision for reduced register size, cache footprint
 - Floating-point style representation
 - Unlike prior schemes, support pointer “out of bounds” for C
 - Important: retain monotonicity for safe delegation!
- Great care must be taken with security—performance trade offs



Virtual
address
space

Papers and reports

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. IEEE Security and Privacy 2015.

Beyond the PDP-11: Processor support for a memory-safe C abstract machine.

David Chisnall, Colin Rothwell, Brooks Davis, Robert N.M. Watson, Jonathan Woodruff, Simon W. Moore, Peter G. Neumann and Michael Roe. ASPLOS 2015.

The CHERI capability model: Revisiting RISC in an age of risk.

Jonathan Woodruff, Robert N. M. Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert Norton, and Michael Roe. ISCA 2014.

Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set

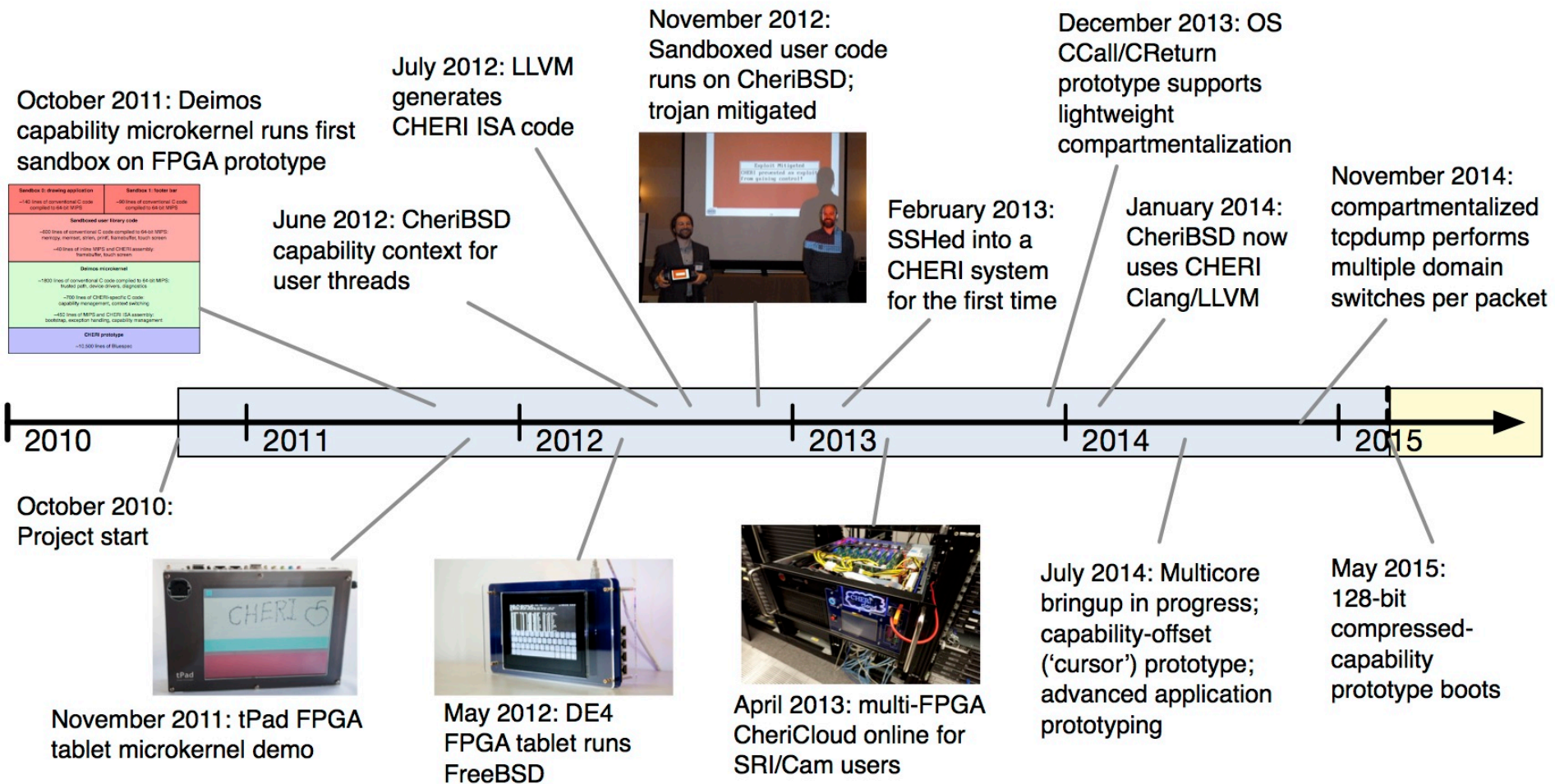
Architecture. Robert N.M. Watson, Peter G. Neumann, Jonathan Woodruff, Jonathan Anderson, David Chisnall, Brooks Davis, Ben Laurie, Simon W. Moore, Steven J. Murdoch, and Michael Roe. UCAM-CL-TR-864, Cambridge, December 2014.

Conclusions

- Hybrid capability model for fine-grained, in-address-space memory protection
 - Supplements current MMU
 - Targets C-language TCBs
- Object-capability model for granular, in-address-space compartmentalization
 - Software-defined domain-transition model
 - Reference-oriented programmer model restored
 - Efficient non-IPC communication with protection
- Open-source reference implementation, ISA specification:

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

Q&A



BACKUP SLIDES

Domain-switch pseudocode

CCall

CReturn

TODO

CCall - cycle-by-cycle analysis

(Including user and kernel space)

Module	Phase	#instr.	#cycles
Caller	Setup call, clear unused argument registers	22 *	42 *
libcheri	Save callee-save regs, push call frame	30 **	34 **
Kernel	Receive trap	13	28
	Validate CCall args	79 * **	79 * **
	Push trusted stack, unseal Ccall args.	31	41 **
	Clear non-argument registers	4	4
	Exit kernel	7	12
Sandbox	Set up sandbox	33	59
		219	299

* Further opportunities for hardware optimization 35

** Will further reduce with smaller registers, converged register file

CReturn cycle-by-cycle analysis

(Including user and kernel space)

Module	Phase	#instr.	#cycles
Sandbox	Exit sandbox	12	16
Kernel	Receive trap	13	31
	Validate return capability	7	7
	Pop trusted stack	26	41
	Clear non-return registers	4	4
	Exit kernel	7	7
libcheri	Pop call frame, restore regs.	28	52
Caller	Back in caller	1	1
		99	159

* Further opportunities for hardware optimization 36

** Will further reduce with smaller registers, converged register file

