CHERI: A Modern Capability Micro-Architecture

Dr. Nathaniel Wesley Filardo

April 11, 2019

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 contract FA8750-10-C-0237. 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.

Introductions

I am...

- a postdoc at Cambridge University, working on CHERI.
- not speaking on behalf of my employer.
- only one of many of us on CHERI:
  Robert N. M. Watson, Simon W. Moore, Peter G. Neumann, Hesham Almatary, Jonathan Anderson, John Baldwin, Hadrien Barrel, Ruslan Bukin, David Chisnall, Nirav Dave, Brooks Davis, Lawrence Esswood, Khilan Gudka, Alexandre Joannou, Robert Kovacsics, Ben Laurie, A. Theo Markettos, J. Edward Maste, Alfredo Mazzinghi, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch, Edward Napierala, Robert Norton-Wright, Philip Paeps, Lucian Paul-Trifu, Alex Richardson, Michael Roe, Colin Rothwell, Hassen Saidi, Peter Sewell, Stacey Son, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Jonathan Woodruff, Hongyan Xia, Bjoern A. Zeeb

Introductions

This talk is...

- not a job talk.
- an experimental information sharing exercise.
- interruptable.
- choose your own adventure!

Introductions

You. . .

- find your home in ECE? CS? Others?
- undergrad? graduate? postdoc? faculty?
- focus on (micro)Architecture? Kernel/OS? Security?
- have heard of “iAPX 432”?

Outline

- Fixed bits of the talk:
  - Motivation for CHERI
  - Project Status
  - CHERI ISA Overview
- Modular lecture components:
  - Tags in Memory and Caches
  - Capabilities as Threats to Pipelines
  - FreeBSD ABI for CHERI: syscall, libc, . . . (ASPLOS'19)
  - Sealing and Controlled Amplification
  - Temporal safety and capability revocation
  - Why is CHERI not doomed to die? ("iAPX 432 again?")
  - Projects seeking students

Motivation

State of software security is “not great.”
Pretty sure that’s not controversial.

Motivation

We keep having the same kinds of problems:

- Stack smashing (Ye Olde Buffere Overflowe, ROP)
- Heap smashing (`malloc` gadgets, JOP)
- Unintended, overt disclosures (Heartbleed)
- Covert disclosures (Spectre variants $1 - \omega$)

Motivation

Many proposals to improve the world floating about:

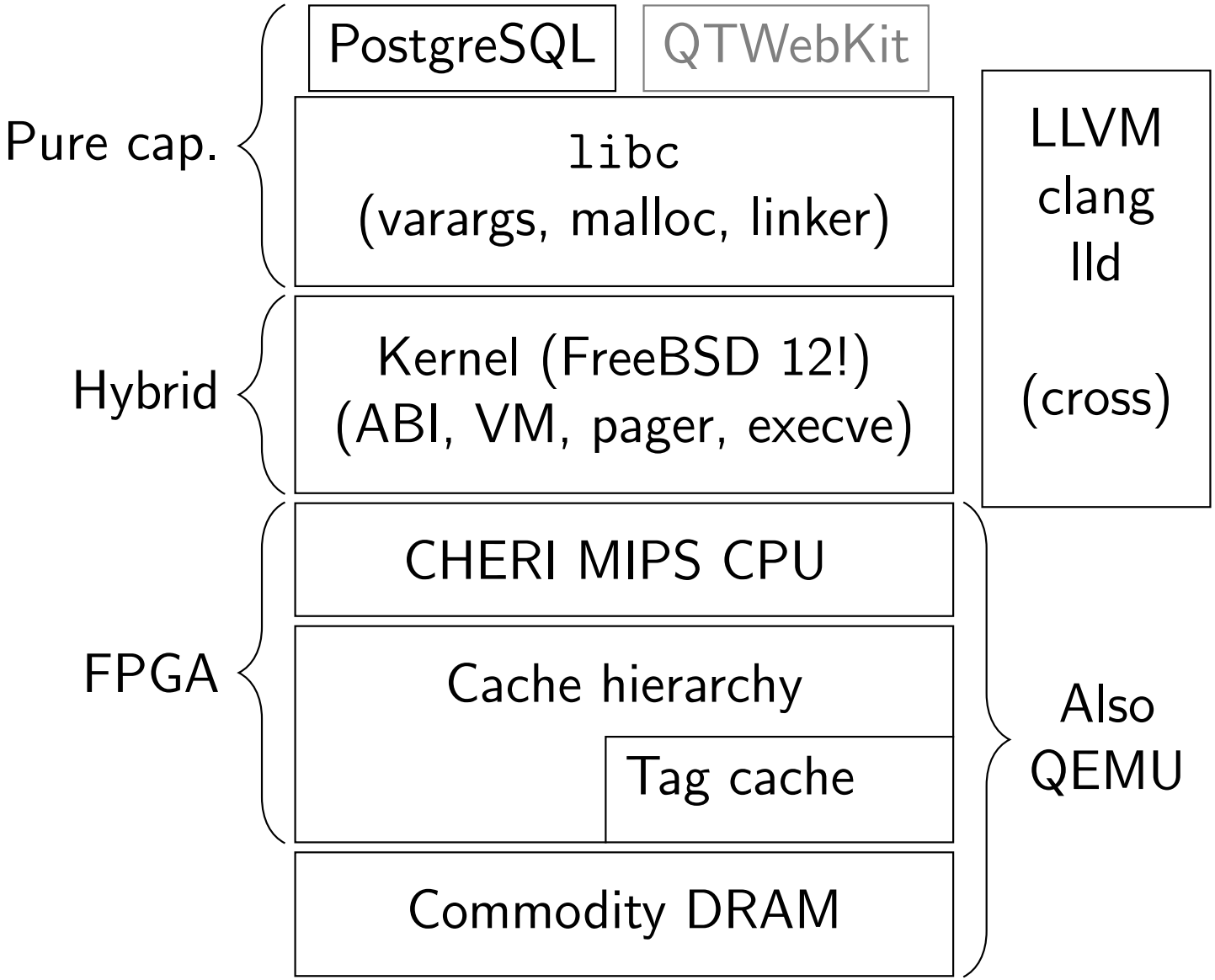
- Proof-Carrying Code
- Heavy emulation of fake systems (JVM, JS, wasm)
- Ever more layers of virtualization ("ring -1")
- Hardware-assisted isolation and attestation (SGX)
- Memory & pointer versioning (ADI, MTE)
- Remove speculation entirely from hardware
- Information flow tagging (PUMP)
- Hardware that directly understands object model (CHERI)

Motivation

We think CHERI is a unique point in the space:

- *directly* conveys some high-level notions in hardware
- *scales* better than current protection mechanisms
- retains broad *compatibility* with
  - languages (C)
  - OSes (FreeBSD)
  - applications (PostgreSQL)
- new hardware, but "just" cores and caches
- has a gradual software transition path, with *incremental gains* to be had
- exists, in FPGA, with large software stack & encouraging results so far

Project Status

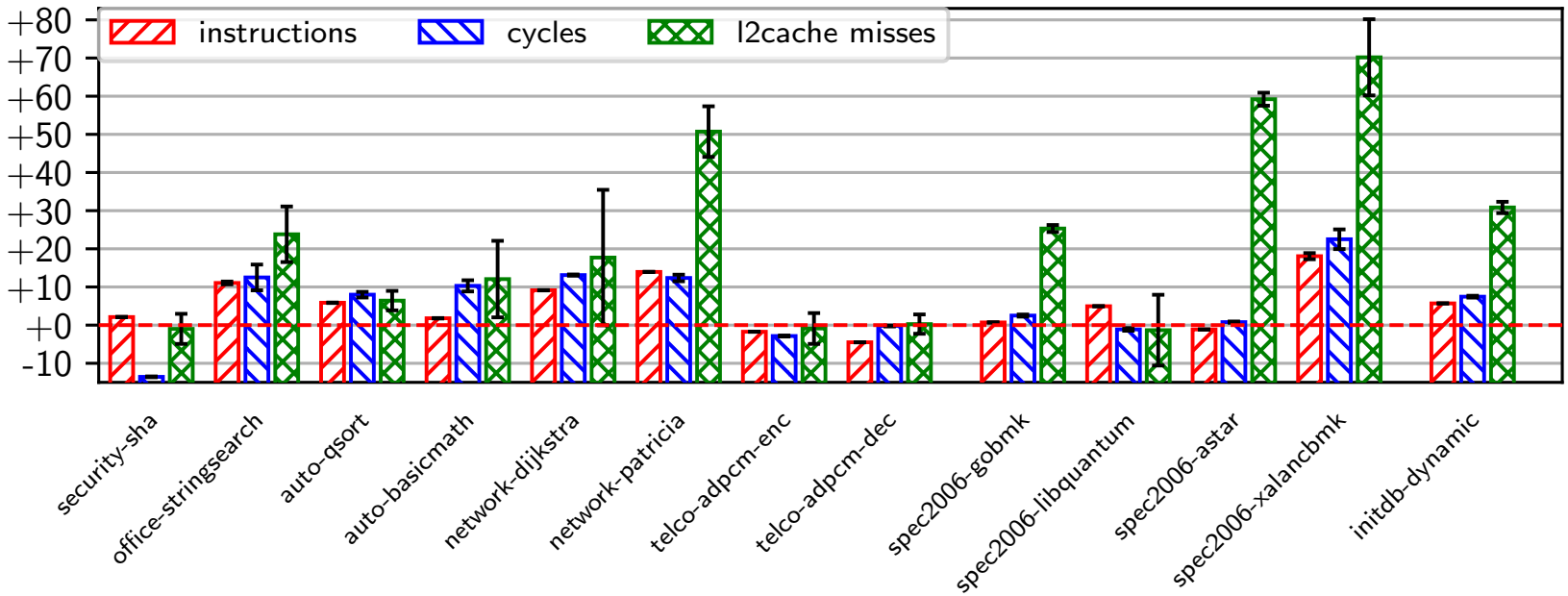


Project Status

- Much of FreeBSD user space is pure capability.
  - Everything from `bash` to OpenSSL.
- C++ support is WIP, focusing on QTWebKit.
  - Self-hosting LLVM in the fullness of time.
- Lots of details in ASPLOS'19 talk next week!

Project Status

MiBench, SPEC CPU2006, and PostgreSQL (ASPLOS'19):



ISA Overview

- Motivation for CHERI
- Project Status
- CHERI ISA Overview
- Modular lecture components

- When reasoning about code, big gains if we can *rule out* possible actions.
  - Type systems for memory safety
  - Object types for access control
  - Separation logic for local pointer graph reasoning

- When reasoning about code, big gains if we can *rule out* possible actions.
  - Type systems for memory safety
  - Object types for access control
  - Separation logic for local pointer graph reasoning
- Integer pointers are *terrible* for this!
  - Add any offset you like before dereferencing.
    - Maybe the MMU, if there is one, will stop you?
  - Any integer might be a pointer in disguise.
  - "Pointers" can even transit the network.

CHERI MIPS core has capability coprocessor.

- Exposes 32 general-purpose capability registers via new "load/store via capability" instructions.
- Some special-purpose capability registers:
  - Kernel-mode-only "scratch" capability registers
  - Program Counter Capability (PCC)
  - Default Data Capability (DDC)

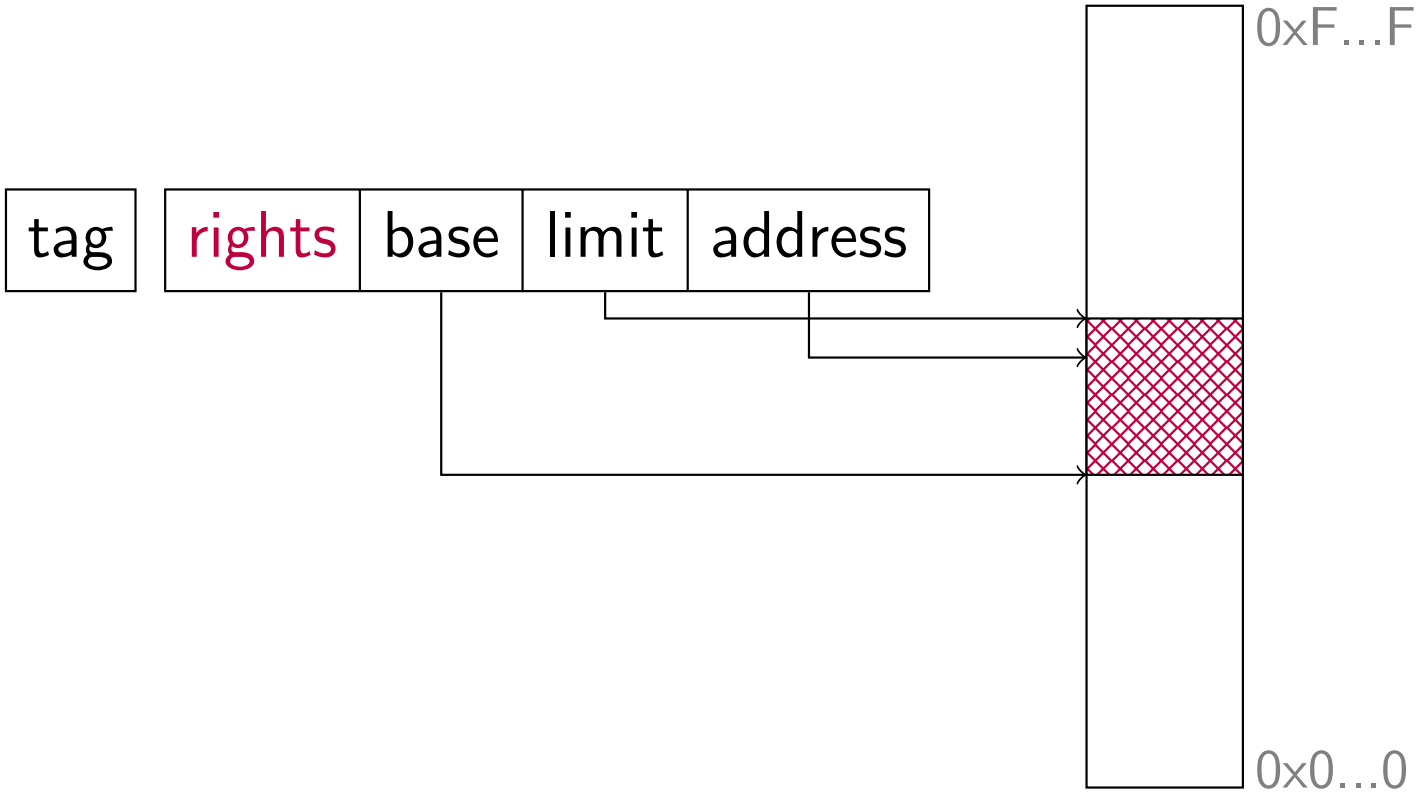
CHERI MIPS running MIPS64 instructions:

- Program counter is indirected through PCC.
- MIPS64 load/store instructions indirected through DDC.

```
MIPS64  ld  $1, 8($2)
 CHERI  cld $1, $2, 8($DDC)
```

- OK, so what is a capability?

Capabilities



A memory capability is an evolved "fat pointer":

- A current position, *address*.
- Bounds: *base* and *limit*, like fat pointer.
- A set of *rights* (R, W, X, LC, SC, ...) and some flags.
- A *tag* attesting its validity.

Capability Use

To *use* a capability, it has to be in a register (like pointer).

- DDC, PCC, or general capability register.
- Full parallel copy of MIPS load/stores + capabilities.
  - store double: `csd $sint, $offset, offset($cap)`
  - load capability: `clc $tcap, $offset, offset($cap)`
- Instructions *fault* if offsets take address out of bounds or `$cap`'s rights do not authorize the operation.

Capability Use

To *use* a capability, it has to be in a register (like pointer).

- DDC, PCC, or general capability register.
- Full parallel copy of MIPS load/stores + capabilities.
  - store double: `csd $sint, $offset, offset($cap)`
  - load capability: `clc $tcap, $offset, offset($cap)`
- Instructions *fault* if offsets take address out of bounds or `$cap`'s rights do not authorize the operation.

Contrast other capability machines:

- *No additional indirection*: only via MMU/TLB.
- No capability resolution table or scan:
  - Everything needed for TLB lookup is in registers.
  - Nothing "out there" to speculate or silently cache.
  - Principle Of Intentional Use:

    Don't use rights you have by mistake!

- Omnipotent capability in register(s) at power-on.
  All (tagged) capabilities trace *provenance* to this root.

Capability Provenance

- Omnipotent capability in register(s) at power-on.
  All (tagged) capabilities trace *provenance* to this root.
- Capabilities copied in registers & duplicated to RAM.
  `cmove $cd, $cs`, `csc $cs, 0($ct)`

Capability Provenance

- Omnipotent capability in register(s) at power-on.
  All (tagged) capabilities trace *provenance* to this root.
- Capabilities copied in registers & duplicated to RAM.
  `cmove $cd, $cs`, `csc $cs, 0($ct)`
- Can change address:
  Increment `cincoffset $cd, $delta, $cs`
  Assign `csetaddr $cd, $addr, $cs`

Capability Provenance

- Omnipotent capability in register(s) at power-on.
  All (tagged) capabilities trace *provenance* to this root.
- Capabilities copied in registers & duplicated to RAM.
  `cmove $cd, $cs`, `csc $cs, 0($ct)`
- Can change address:
  Increment `cincoffset $cd, $delta, $cs`
  Assign `csetaddr $cd, $addr, $cs`
- Can narrow bounds:
  `csetbounds $cd, $length, $cs` (`cd.base` = `cs.addr`)

Capability Provenance

- Omnipotent capability in register(s) at power-on.
  All (tagged) capabilities trace *provenance* to this root.
- Capabilities copied in registers & duplicated to RAM.
  `cmove $cd, $cs`, `csc $cs, 0($ct)`
- Can change address:
  Increment `cincoffset $cd, $delta, $cs`
  Assign `csetaddr $cd, $addr, $cs`
- Can narrow bounds:
  `csetbounds $cd, $length, $cs` (`cd.base` = `cs.addr`)
- Can remove rights (logical AND):
  `candperm $cd, $mask, $cs`

Capability Provenance

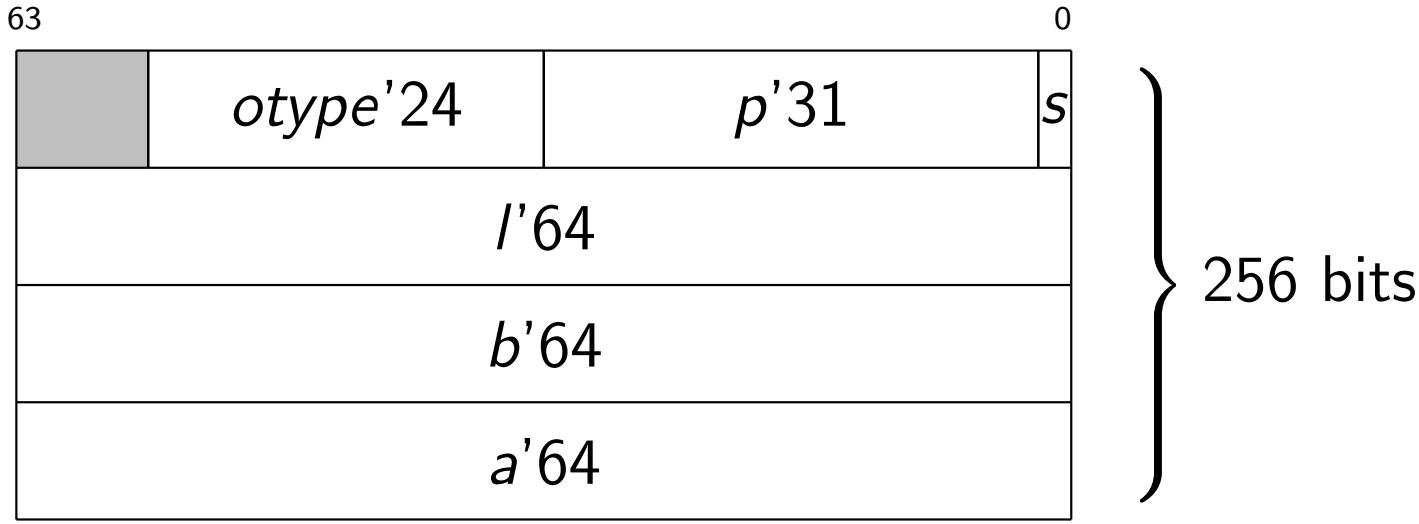
- Omnipotent capability in register(s) at power-on.
  All (tagged) capabilities trace *provenance* to this root.
- Capabilities copied in registers & duplicated to RAM.
  `cmove $cd, $cs`, `csc $cs, 0($ct)`
- Can change address:
  Increment `cincoffset $cd, $delta, $cs`
  Assign `csetaddr $cd, $addr, $cs`
- Can narrow bounds:
  `csetbounds $cd, $length, $cs` (`cd.base` = `cs.addr`)
- Can remove rights (logical AND):
  `candperm $cd, $mask, $cs`
- Invalid operations give rise to untagged values.
  (Makes our architects happier than traps would.)

- Capabilities live in regular memory.
  - Inspectble with regular load/store instructions.
  - Capability format, bits are not secret.
- Data stores to memory clear tag;
  bits stop being a capability.
- Can neither synthesize nor corrupt a tagged capability.

Execution confined to transitive closure of capability registers.

- Ambient actions all monotone non-increasing in rights.
- Some capabilities are opaque ("sealed"):
  - can be held
  - immutable
  - convey no authority until unsealed
- Controlled non-monotone changes to register file on control transfers
  - "(un)sealing rights" are capability-mediated
  - Some registers only available to kernel
  - Unseal some capability/ies by jumping to them

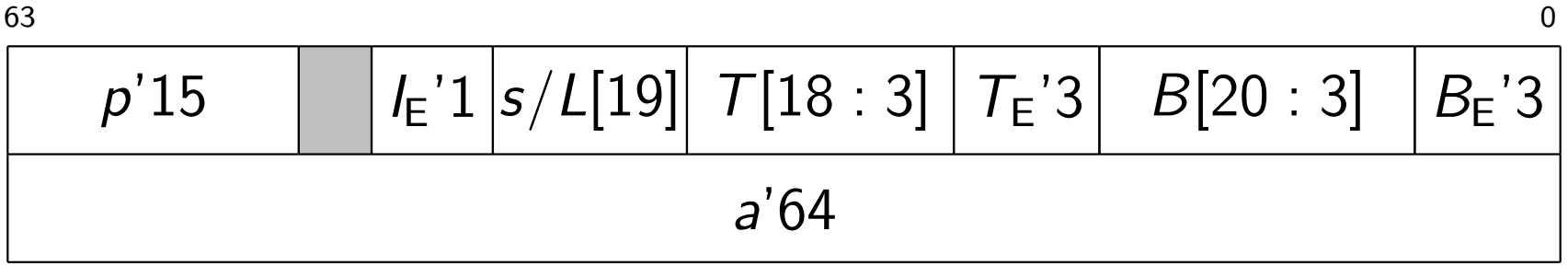
Original, CHERI-256 capability format:



otype: object type *p*: permissions *s*: sealed
l: object length *b*: base *a*: pointer address

- Direct encoding, so easily manipulated.
- Bulky and had to be aligned.
- Now have compressed format.

CHERI Concentrate 128 format:



p: permissions s: sealed a: pointer address
T: encoded limit B: encoded base $?_E$: exponent

- Smaller size and alignment requirements.
- Inspired by floating point representations.
- Doubles tag density to 1/128 bits (.8%).
- Not all bounds or (pointer,bounds) pairs are representable.

Lecture Modules

Vote early, vote often!

- Tags in Memory and Caches
- Capabilities as Threats to Pipelines
- FreeBSD ABI for CHERI: syscall, libc, . . . (ASPLOS'19)
- Sealing and Controlled Amplification
- Temporal safety and capability revocation
- Why is CHERI not doomed to die? ("iAPX 432 again?")
- Projects seeking students

Conclusion

- CHERI is a new and interesting architectural design.
  - Hybrid of capability and pointer-based designs
  - Largely compatible with existing languages, systems, and applications.
  - Large gain for comparatively modest changes?

Conclusion

- CHERI is a new and interesting architectural design.
  - Hybrid of capability and pointer-based designs
  - Largely compatible with existing languages, systems, and applications.
  - Large gain for comparatively modest changes?
- Hopefully whetted your appetite for more information.

`http://cheri-cpu.org`

Conclusion

- CHERI is a new and interesting architectural design.
  - Hybrid of capability and pointer-based designs
  - Largely compatible with existing languages, systems, and applications.
  - Large gain for comparatively modest changes?
- Hopefully whetted your appetite for more information.

  `http://cheri-cpu.org`
- We'd love to hear from you.
  - I am `nwf20@cl.cam.ac.uk`
  - `robert.watson@cl.cam.ac.uk` software-leaning PI
  - `simon.moore@cl.cam.ac.uk` hardware-leaning PI

CHERI Concentrate Regions