CTSRD

CRASH-worthy Trustworthy Systems Research and Development

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, Michael Roe

University of Cambridge, SRI International, Google

ISCA 2014 — 18 June 2014

Approved for public release. 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/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.

SRI International

UNIVERSITY OF CAMBRIDGE

Memory Safety Crisis

~82% of exploited vulnerabilities in 2012

— Software Vulnerability Exploitation Trends, Microsoft

How are processors responding?

Memory Safety Deprecation & Demand

Burroughs B5000

IBM System/38

Consensus on paged virtual memory

AMD64

1960 1970 1980 1990 2000 2010

RISC

Intel MPX

Cambridge CAP computer

Security becoming marketable

Hardware support for fine-grained memory safety was the future!

We've Built A Real Open Source System

- CHERI processor + peripherals on FPGA
- Extension of FreeBSD OS (*Including Capsicum software capabilities*)
- Clang & LLVM

Capability: Unforgeable token of authority.

Some “capabilities”:

- File descriptors (Capsicum, L4)
- Segment descriptors (CAP, Intel iAPX)
- Pointers in a virtual machine (Java, .Net)
- Bounded pointers (M-Machine)

CHERI Capabilities are Unforgeable Fat Pointers

Fat Pointer = Base + Length + Permissions

Build a RISC Capability Machine

- Single-cycle instructions
- Load/Store architecture
- Compiler & OS manage capabilities

Build A Useful RISC Capability Machine

- Keep page-table for virtualisation and backward compatibility
- Constrain existing loads and stores with implied capability register
- Integrate with 64-bit MIPS ISA
  (*Applicable to any RISC ISA*)

Paged Memory

- OS managed
- Enables swapping
- Centralized
- Allows revocation

Address validation

Capabilities

- Compiler managed
- Precise
- Can be delegated
- Many domains

Pointer safety

Paged Memory + Capabilities

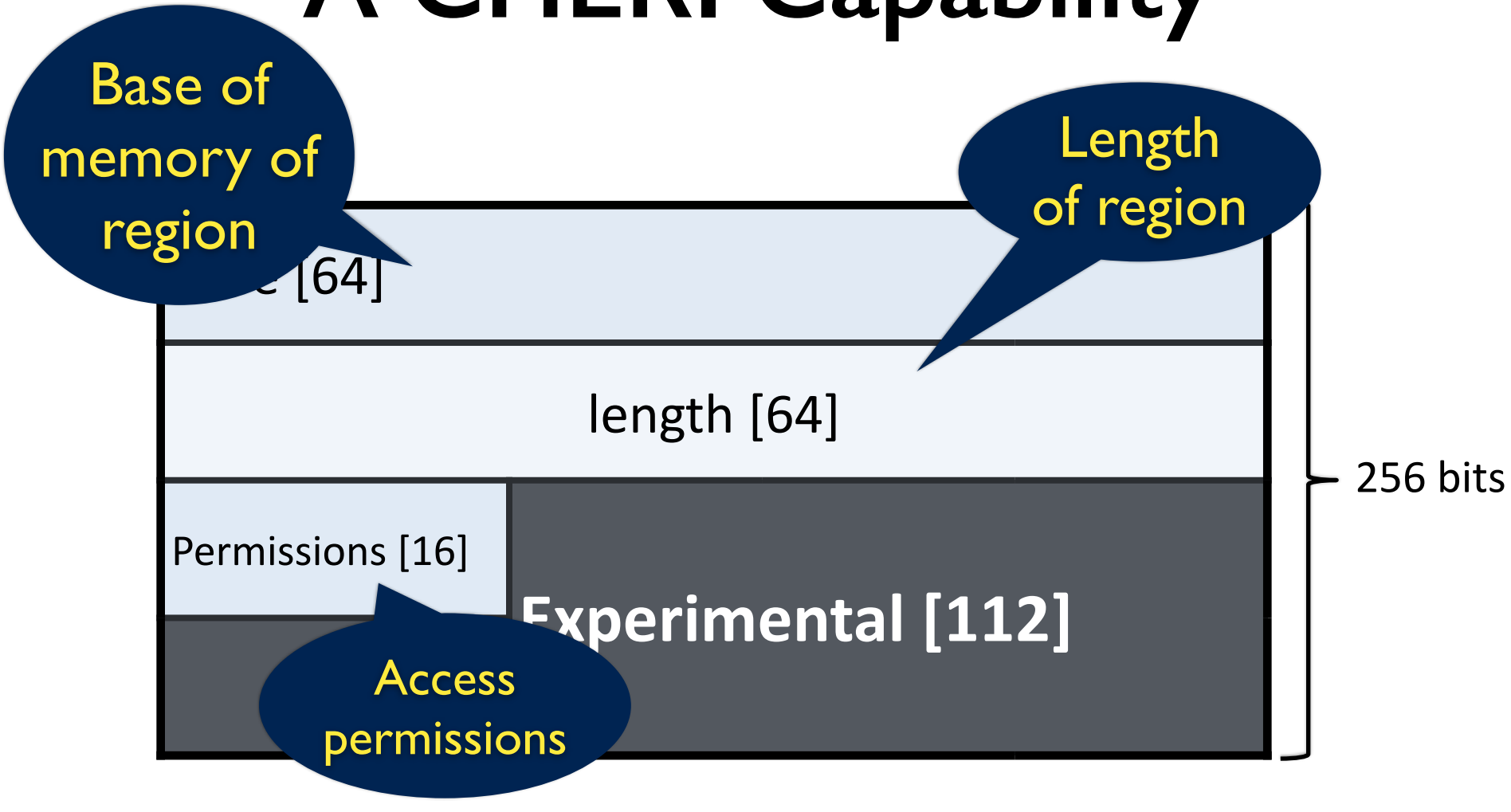
- OS managed
- Enables swapping
- Centralized
- Allows revocation

Address validation

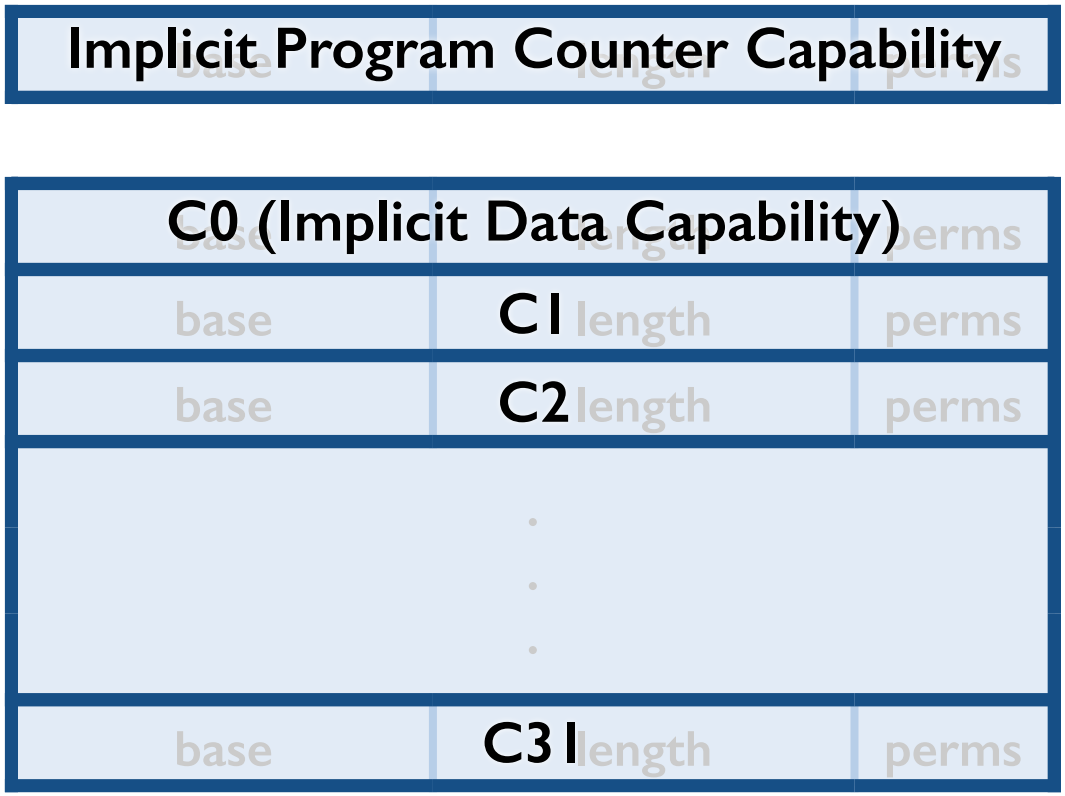
- Compiler managed
- Precise
- Can be delegated
- Many domains

Pointer safety

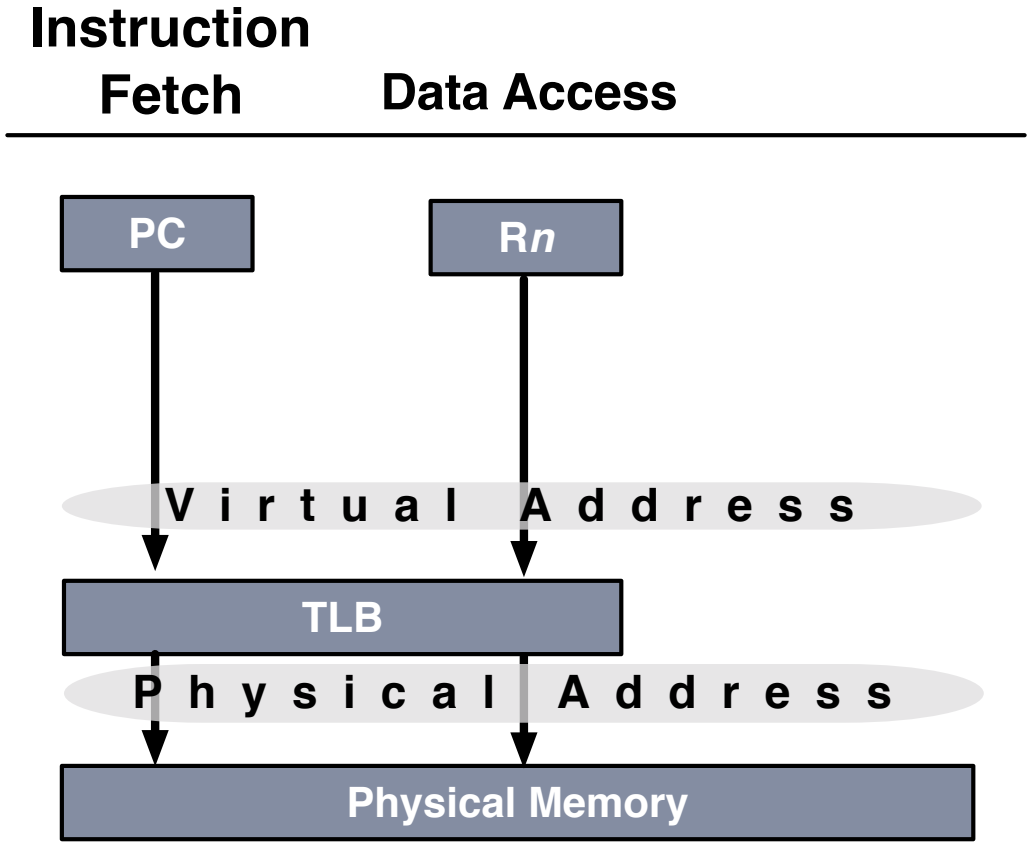
A CHERI Capability



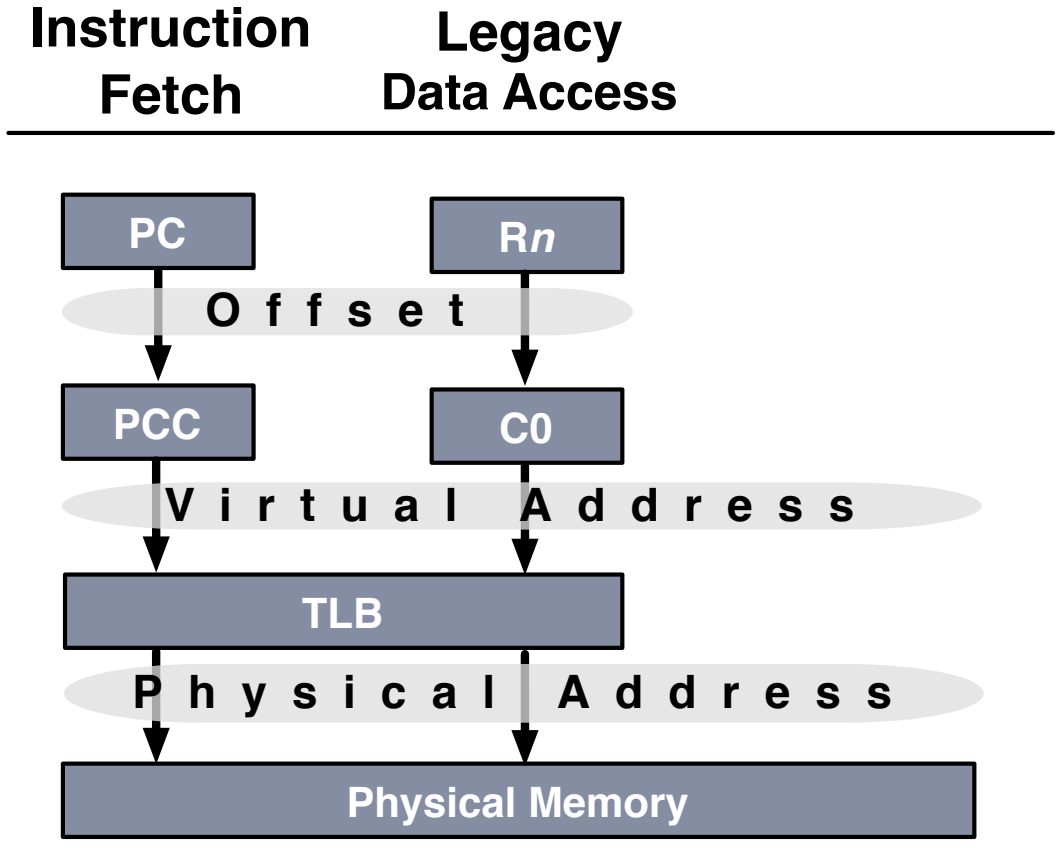
Capability Register File
Both Implicit and Explicit Use



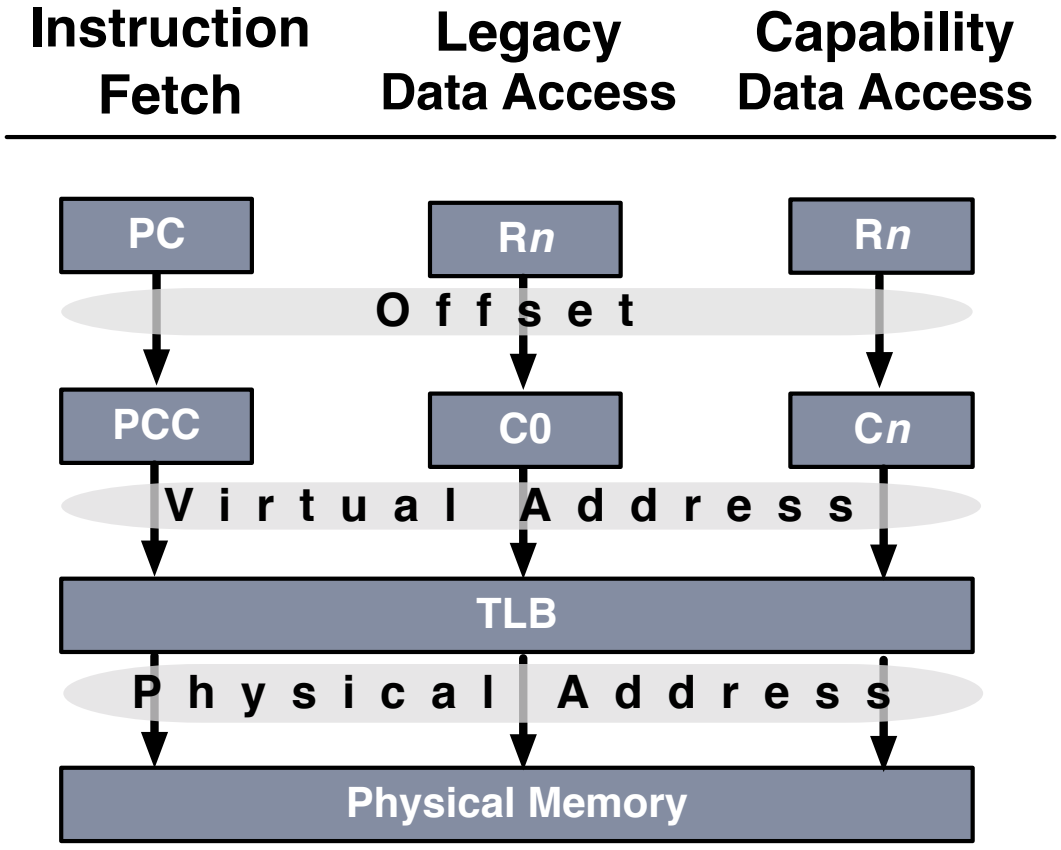
Address calculation



Address calculation



Address calculation



Capabilities can Replace Pointers

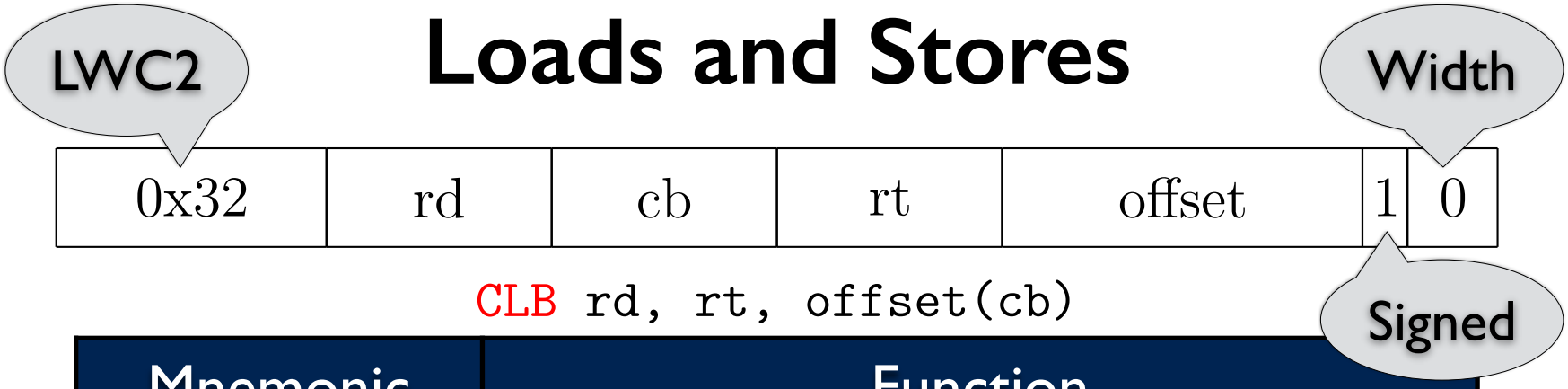
- Unprivileged capability manipulation instructions
- Capability loads and stores for all required memory operations

Capability Transformations Strictly Reduce Privilege

Mnemonic	Function
CIncBase	Increase base and decrease length
CSetLen	Reduce length
CAndPerm	Restrict permissions
CClearTag	Invalidate a capability register

Full Instruction Set Reference: www.cl.cam.ac.uk/techreports/UCAM-CL-TR-850.pdf

Complete Set of Capability Loads and Stores

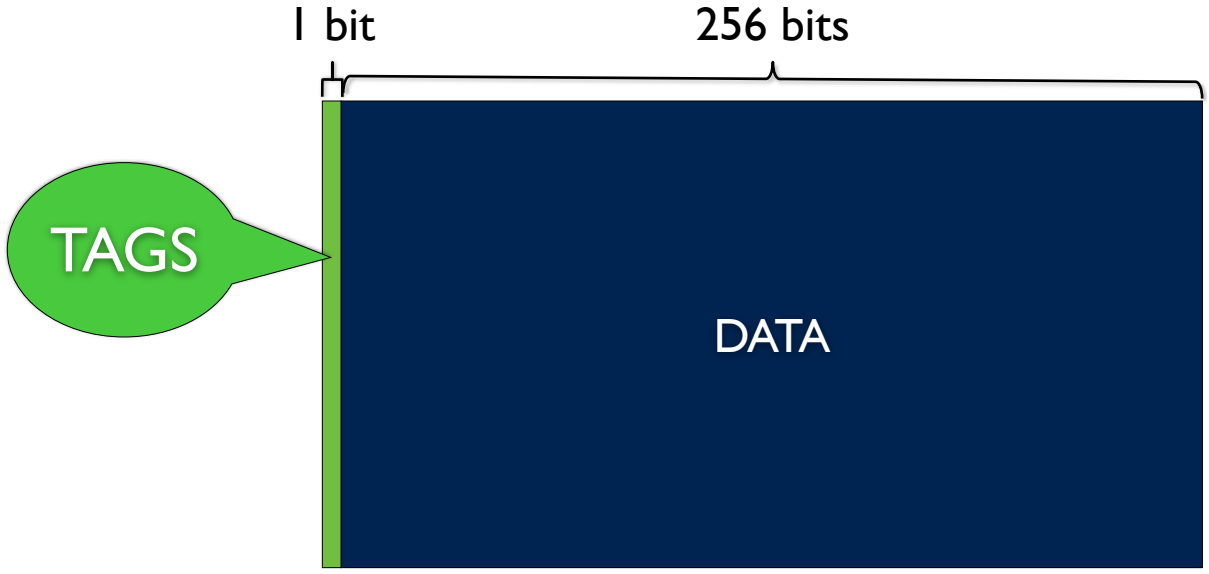



```
CLB rd, rt, offset(cb)
```

Mnemonic	Function
CSC	Store capability register
CLC	Load capability register
CL[BHWD][U]	Load byte-double via capability register
CS[BHWD]	Store byte-double via capability register
CLLD	Load-linked via capability register
CSCD	Store conditional via capability register

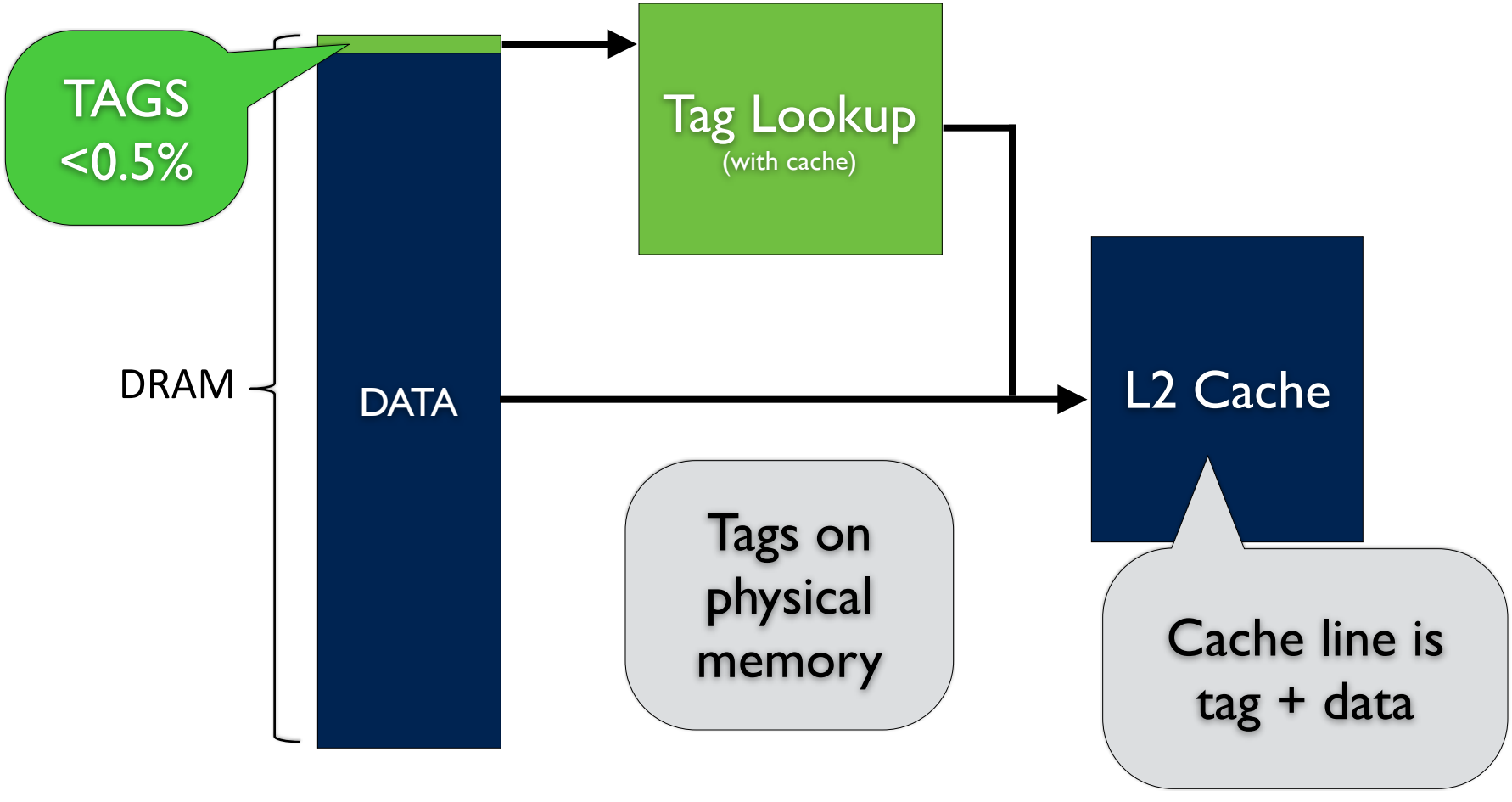
Full Instruction Set Reference: www.cl.cam.ac.uk/techreports/UCAM-CL-TR-850.pdf

Tags to Protect Capabilities in Memory



Capabilities on the stack and in data structures

Tag Table in Commodity DRAM



OS Support is Simple

- Preserve per-process capability state
- Deliver capability exception signals

*Result: **Capability machine in each address space***

C-language Support is Straightforward

- Clang extension to **implement pointers as capabilities**
- `__capability` qualifier on pointers
- Used almost like any other pointer
  (no subtraction)

C-language Support is Straightforward

```
__capability char *myString =
  (__capability char*)malloc(size);



myString[size] = ‘d’;
```

```
jalr          malloc

cincbase$c1, $c0, returnValue
csetlen $c1, $c1, size

csb      ‘d’, size, 0($c1)
```

C-language Support is Straightforward

```
__capability char *myString =
  (__capability char*)malloc(size);



myString[size] = ‘d’;
```

```
jalr          malloc

cincbase$c1, $c0, returnValue
csetlen $c1, $c1, size

csb      ‘d’, size, 0($c1)
```

Run-time Trap!

Program Compartmentalisation is Flexible

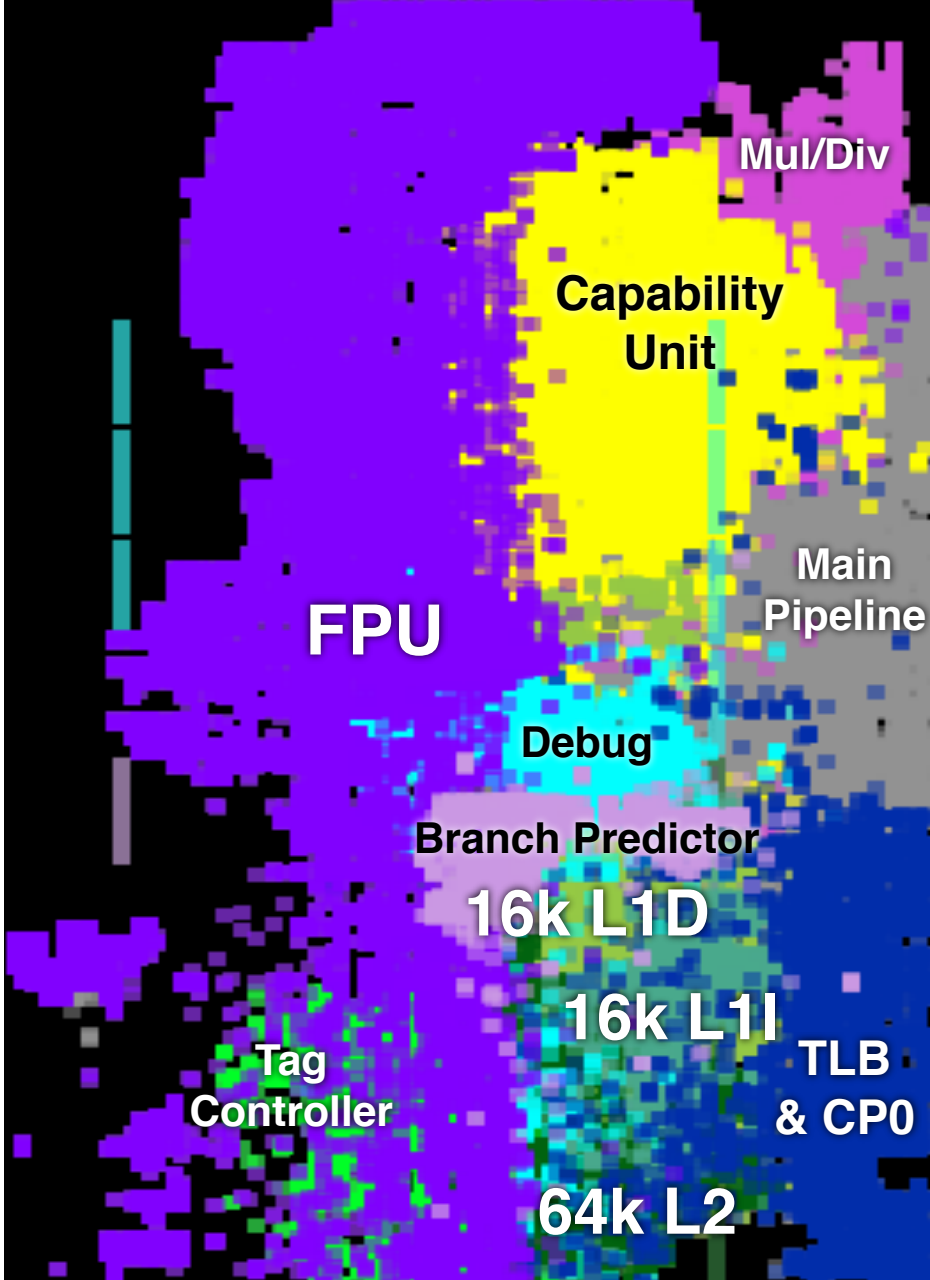
- Coarse-grained using C0, the implicit data capability
- Fine-grained using native capability addressing

CHERI is Built on BERI

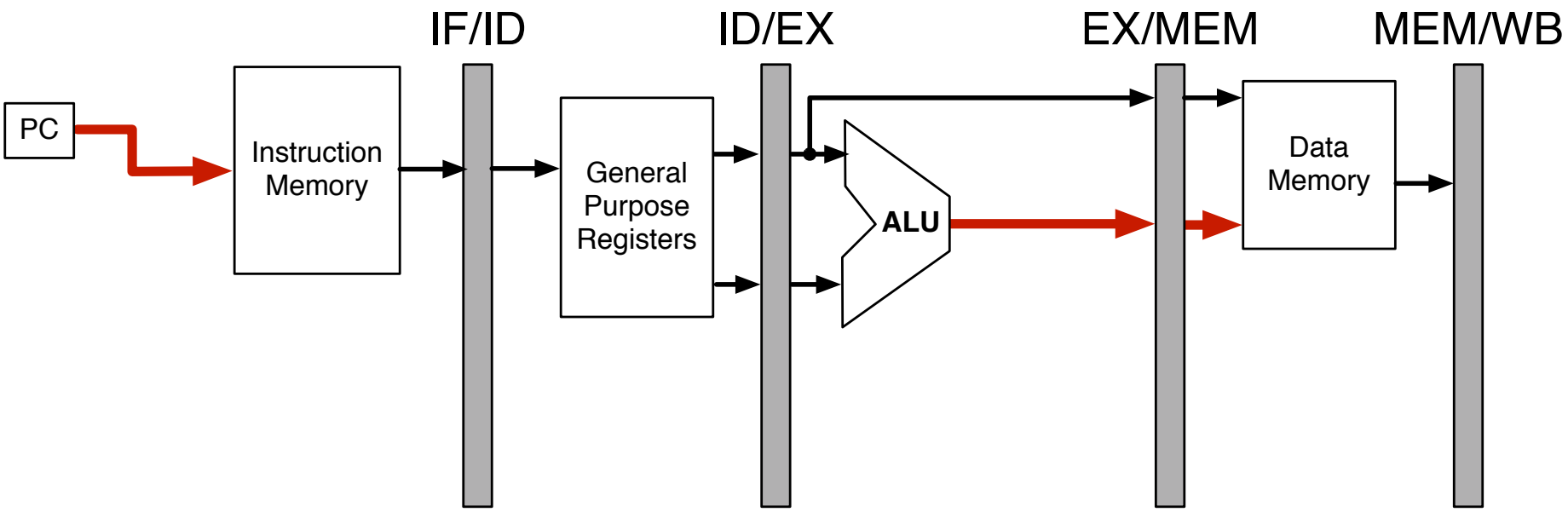
"Bluespec Extensible RISC Implementation"

- 64-bit MIPS R4000 ISA
- 6-stage pipeline
- Single issue, in order
- >100 MHz on Altera Stratix IV

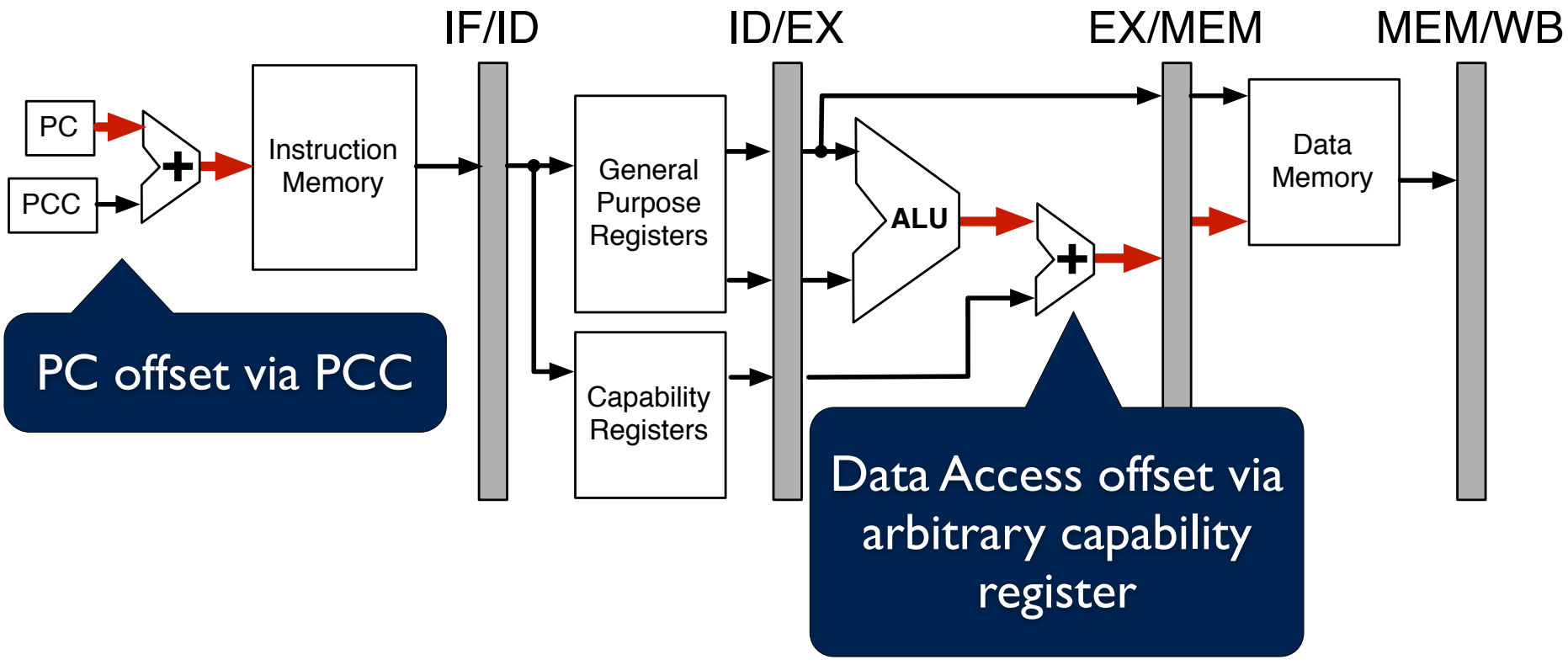
Open source at www.beri-cpu.org



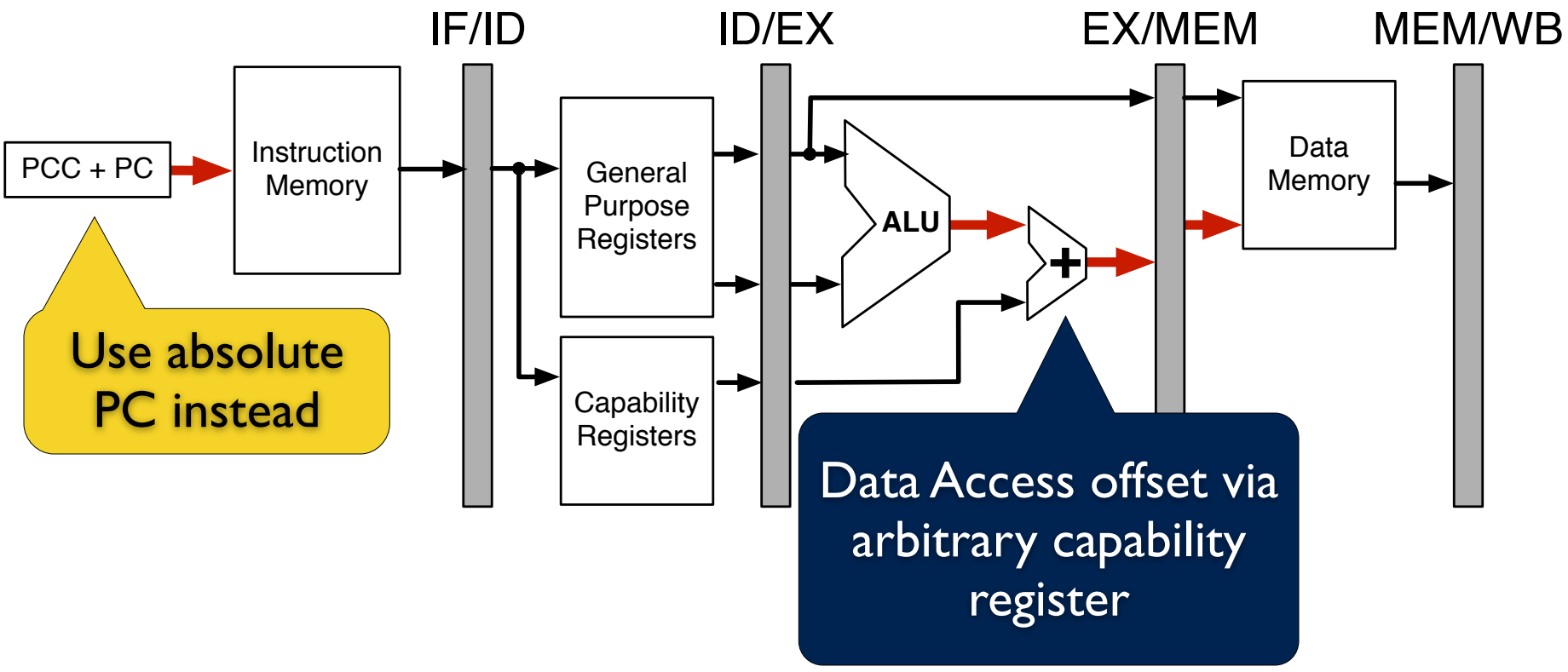
Address Calculation Pipeline



Address Calculation Pipeline



Address Calculation Pipeline



See Paper for Limit Study

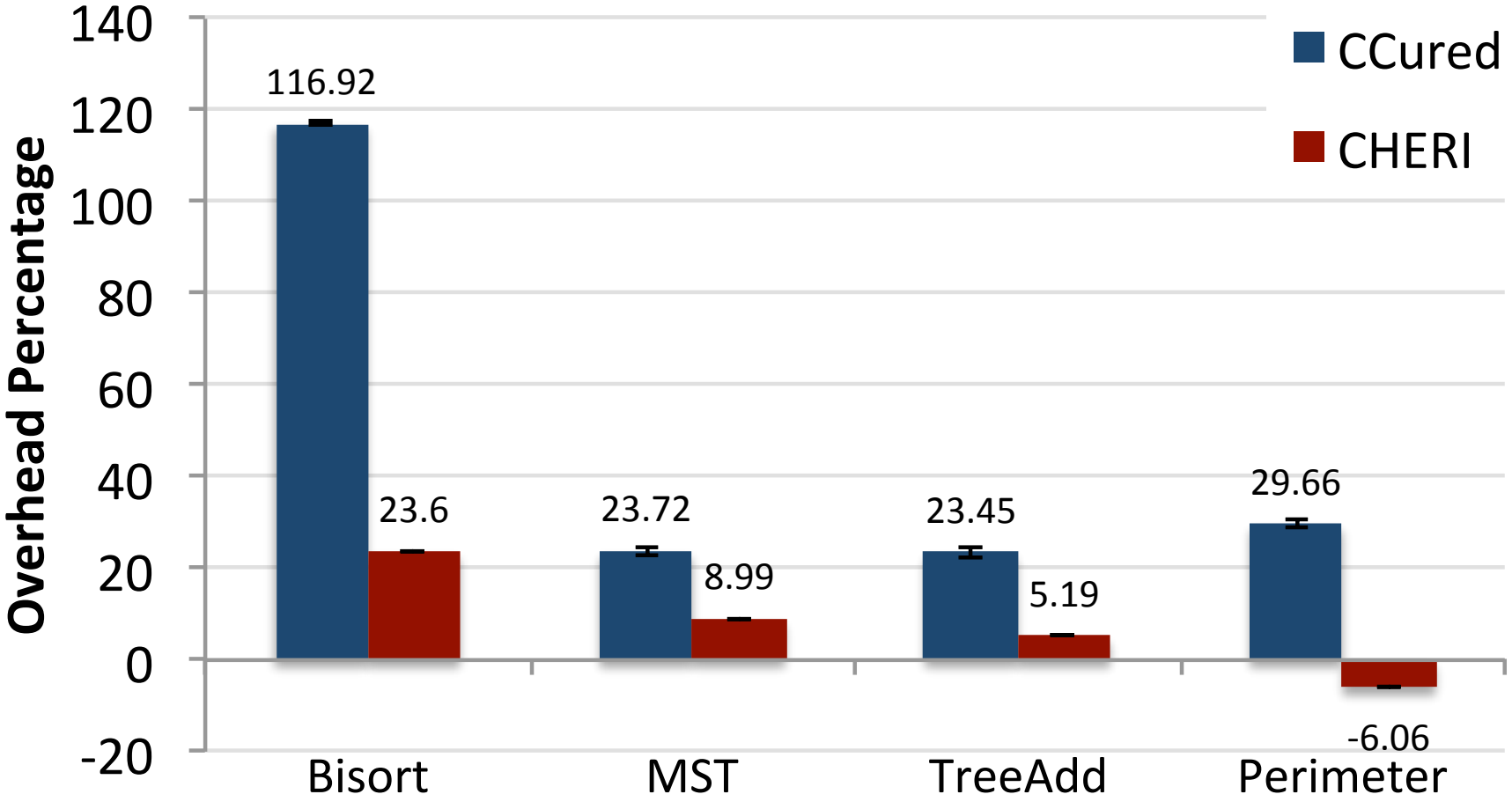
Conclusions:

- CHERI is competitive
- Our capability size is the only notable overhead
- A hypothetical 128-bit CHERI has leading performance

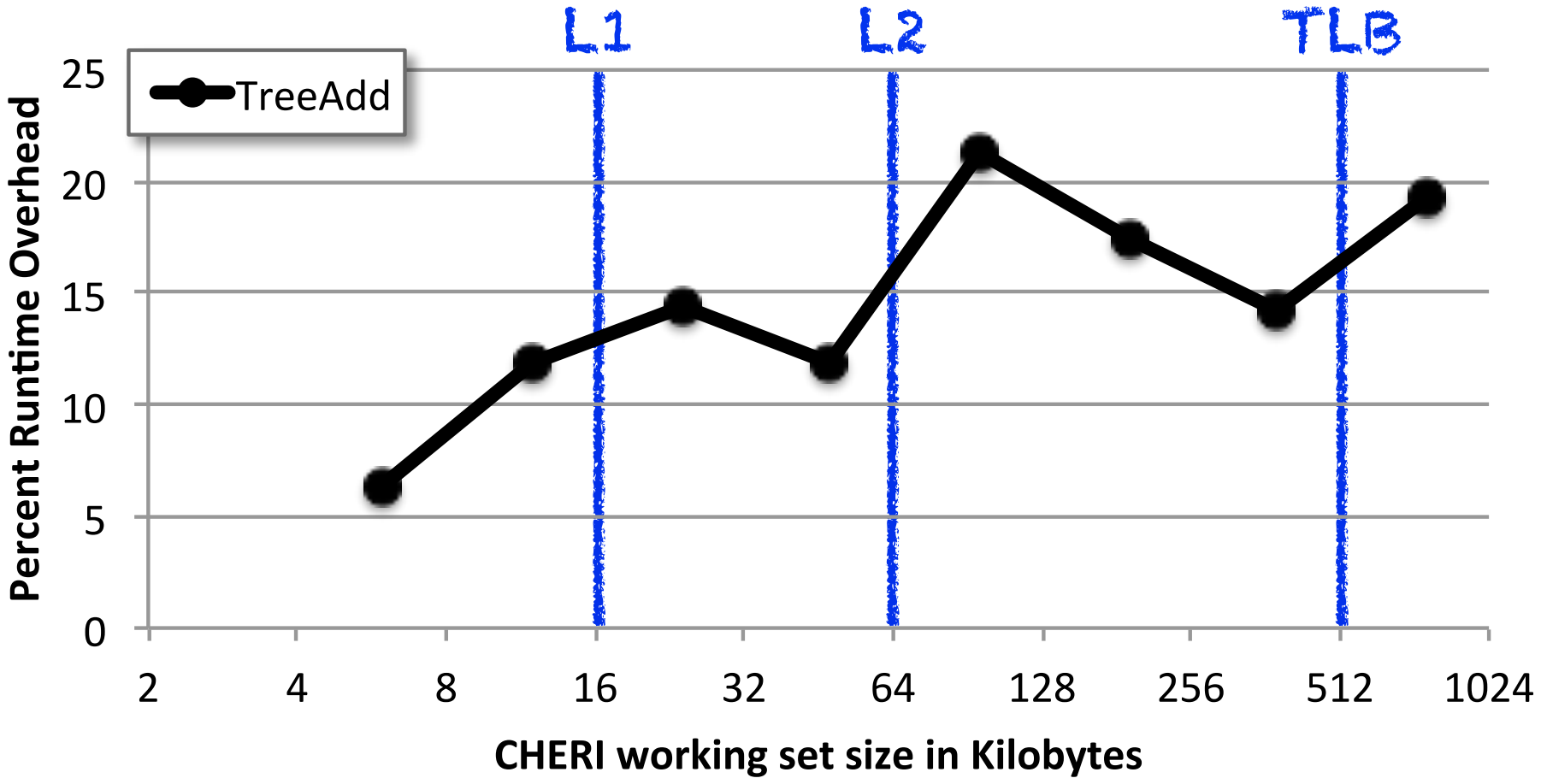
CHERI vs. CCured

- Running in userspace under FreeBSD on CHERI FPGA prototype
- We ported CCured, an automatic memory-safe transform for C

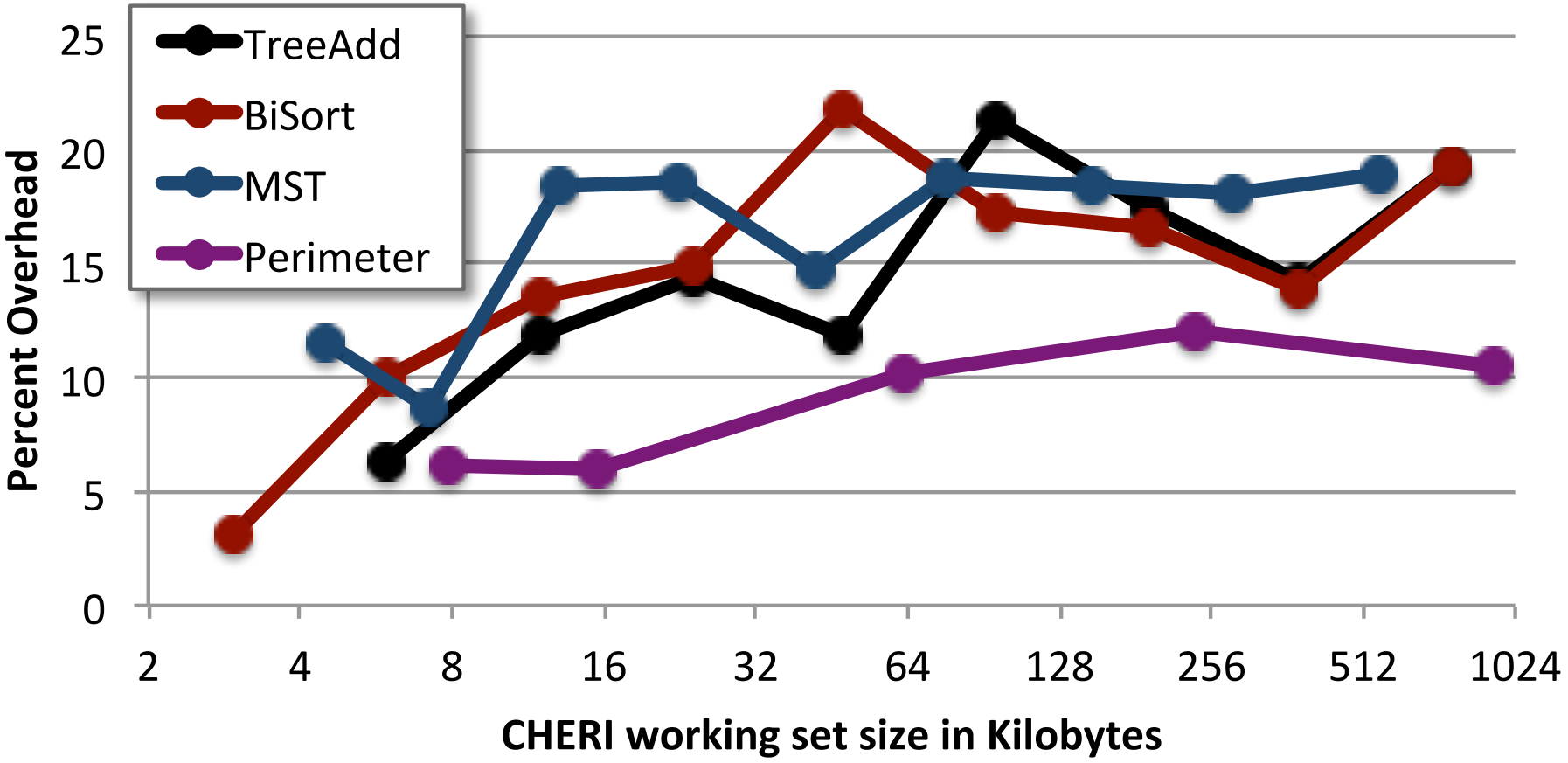
Olden Bounds-checking



Protection Slowdown vs. Working Set Size



Protection Slowdown vs. Working Set Size



Conclusions

- Memory safety needs hardware support
- Current approaches are too weak or too disruptive
- A hybrid capability approach is compatible and scalable

Questions?

CHERI & SoC RTL, LLVM, & FreeBSD are open source! www.cheri-cpu.org

Thanks to DARPA and Google for support!