CTSRD

CRASH-WORTHY
TRUSTWORTHY
SYSTEMS
RESEARCH AND
DEVELOPMENT

Efficient Tagged Memory

Alexandre Joannou, Jonathan Woodruff, Simon W. Moore, Robert Kovacsics, Hongyan Xia, Robert N. M. Watson, David Chisnall, Michael Roe, Brooks Davis, Peter G. Neumann, Edward Napierala, John Baldwin, A. Theodore Markettos, Khilan Gudka, Alfredo Mazzinghi, Alexander Richardson, Stacey Son and Alex Bradbury

University of Cambridge and SRI International

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.

SRI International

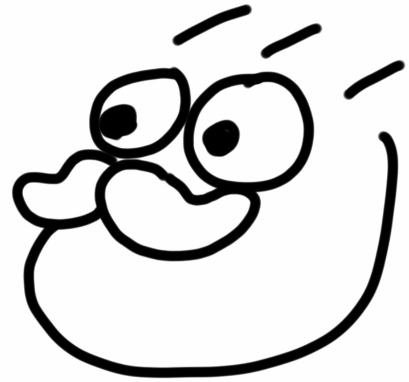
UNIVERSITY OF CAMBRIDGE

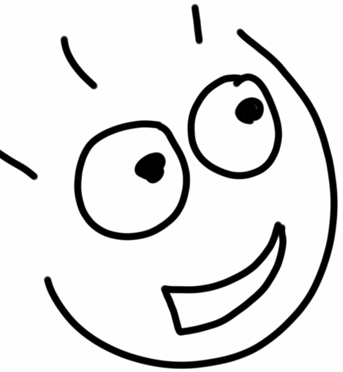
Johnny Proposes Tagged Memory

1-bit Tag Per Word!

- Tag pointers for integrity?
- Tag allocated memory?
- Data flow tracking?
- Watchpoints on any word?

Only 1-bit per word!



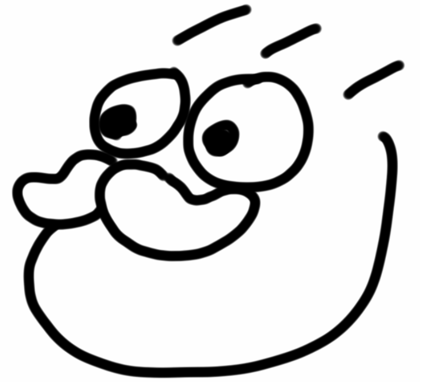


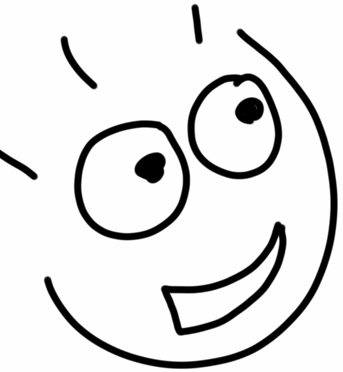
Johnny Proposes Tagged Memory

1-bit Tag Per Word!

- Tag pointers for integrity?
- Tag allocated memory?
- Data flow tracking?
- Watchpoints on any word?

Only 1-bit per word!





Non-standard Memory?

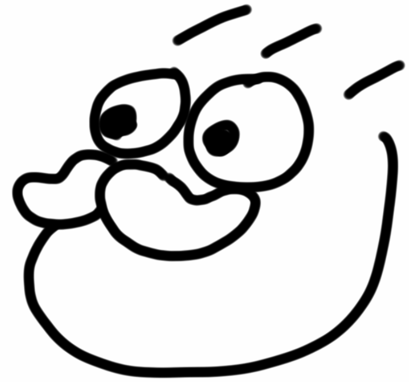
- Custom cache width is possible
- Registers could preserve the bit
- But custom DRAM is a non-starter

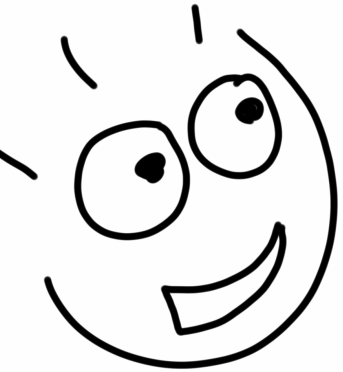
  We can't even afford ECC!
- **Security must be free!**

Johnny Proposes Tagged Memory

A Tag Table in DRAM!

- Put table in standard DRAM
- It will be really small (1-bit per word!)
- Emulate wider memory, fetch tag and data on cache miss
- Keep them together on-chip

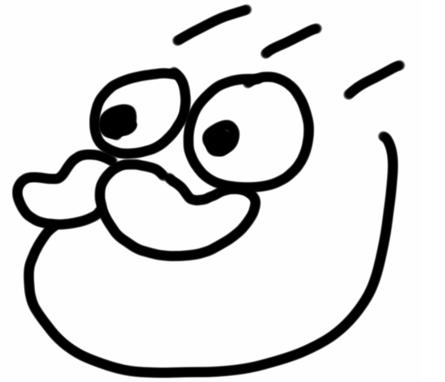


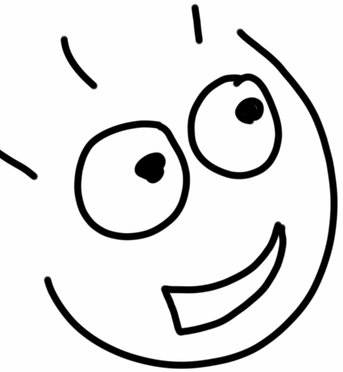


Johnny Proposes Tagged Memory

A Tag Table in DRAM!

- Put table in standard DRAM
- It will be really small (1-bit per word!)
- Emulate wider memory, fetch tag and data on cache miss
- Keep them together on-chip





Double the Memory Accesses?

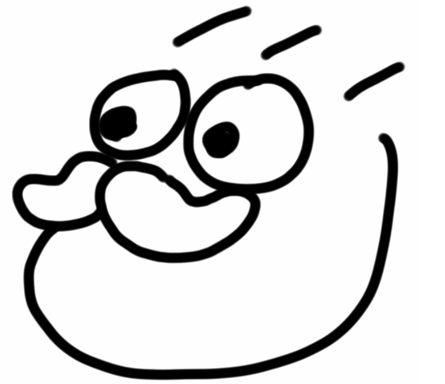
- Access both the table and the data on every cache miss?
- **No**

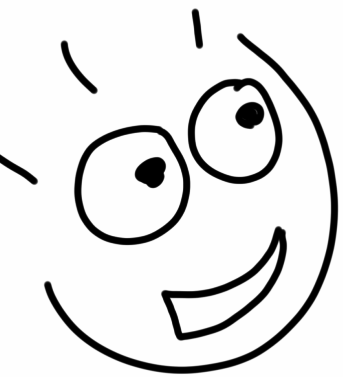
Johnny Proposes Tagged Memory

A Cache for the Tag Table!

- Use a dedicated cache for the tags!
- It will hold tags for loads of data

  (*1-bit per word*! Covers megabytes of data!)
- Only do DRAM table lookup on a miss



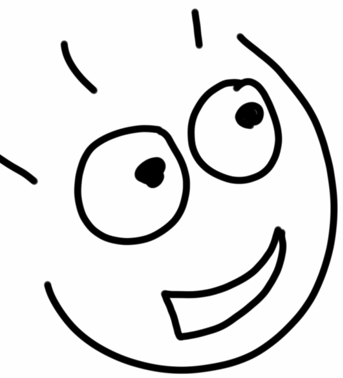


Johnny Proposes Tagged Memory

A Cache for the Tag Table!

- Use a dedicated cache for the tags!
- It will hold tags for loads of data.

  (*1-bit per word*! Covers megabytes of data!)
- Only do DRAM table lookup on a miss.



Last-level Caches Aren't that Effective

- This is logically a last-level cache
- LLC has low hit-rates: 40-60% for SPEC

  We only see accesses that have missed in primary caches…
- **+50% memory accesses isn't going to fly**

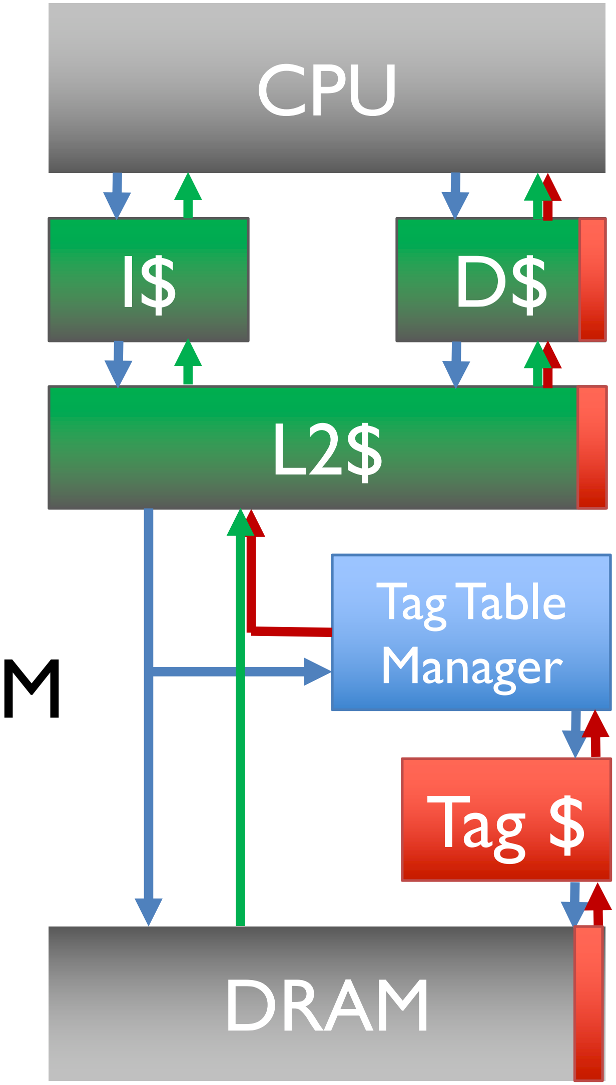
The Tagged Memory Challenge

1. Add 1 bit per word of memory

2. Make it “free”

Re-cap Simple Tag Hierarchy

- Store tags with data in cache hierarchy
- Tag controller does tag table lookup on DRAM access
- Cache lines of tags from DRAM

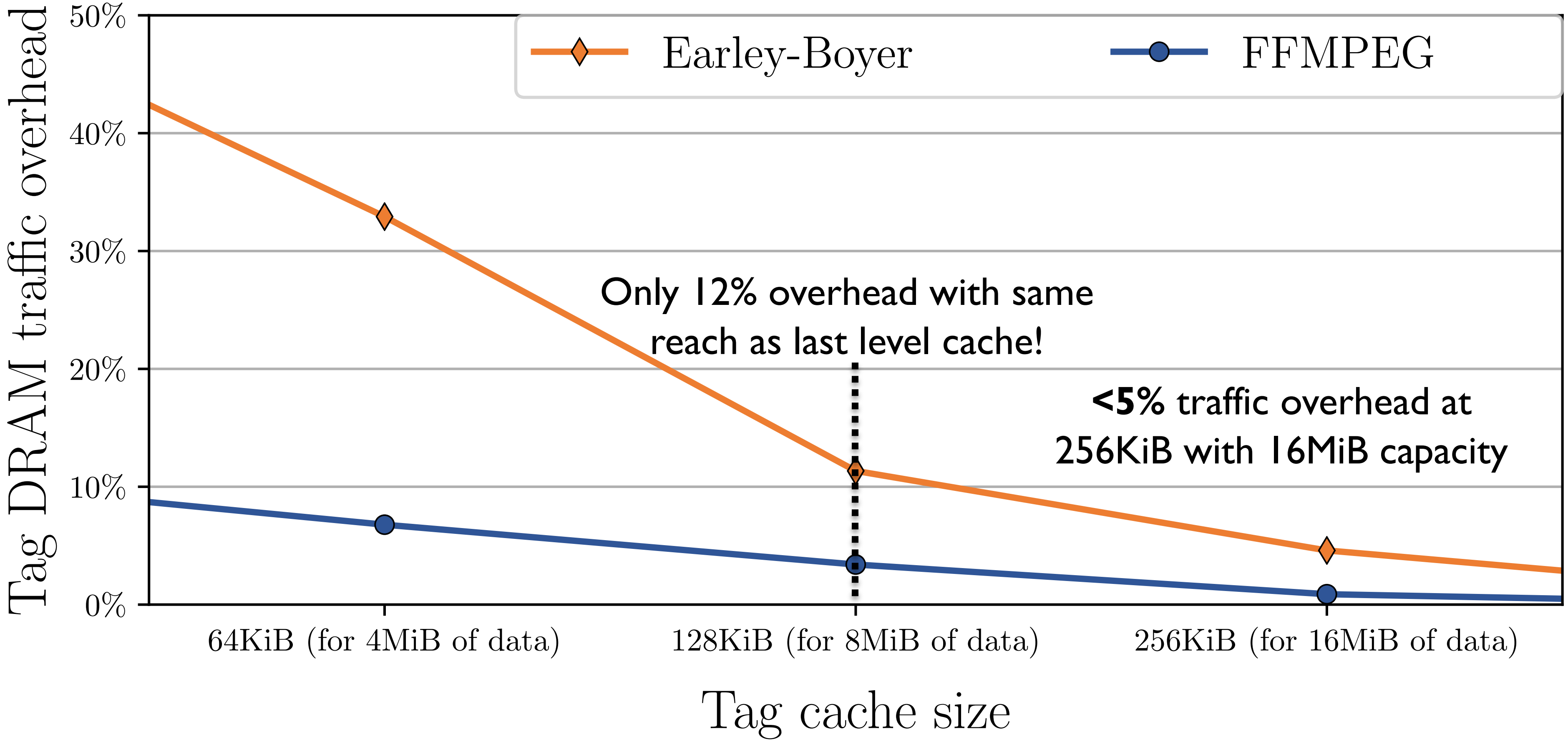


An Experiment in Gem5

- Trace all DRAM accesses
- Replay against a tag controller + cache model
- Measure tag-cache hit-rate
  - Using ARMv8 Gem5
  - Google v8 engine running Earley-Boyer Octane (x3)
  - FFMPEG
  - 4-core, 8MiB L3 with prefetching

Tag Table Cache Properties

DRAM traffic overhead vs. tag cache size, 64-byte lines



Only 12% overhead with same reach as last level cache!

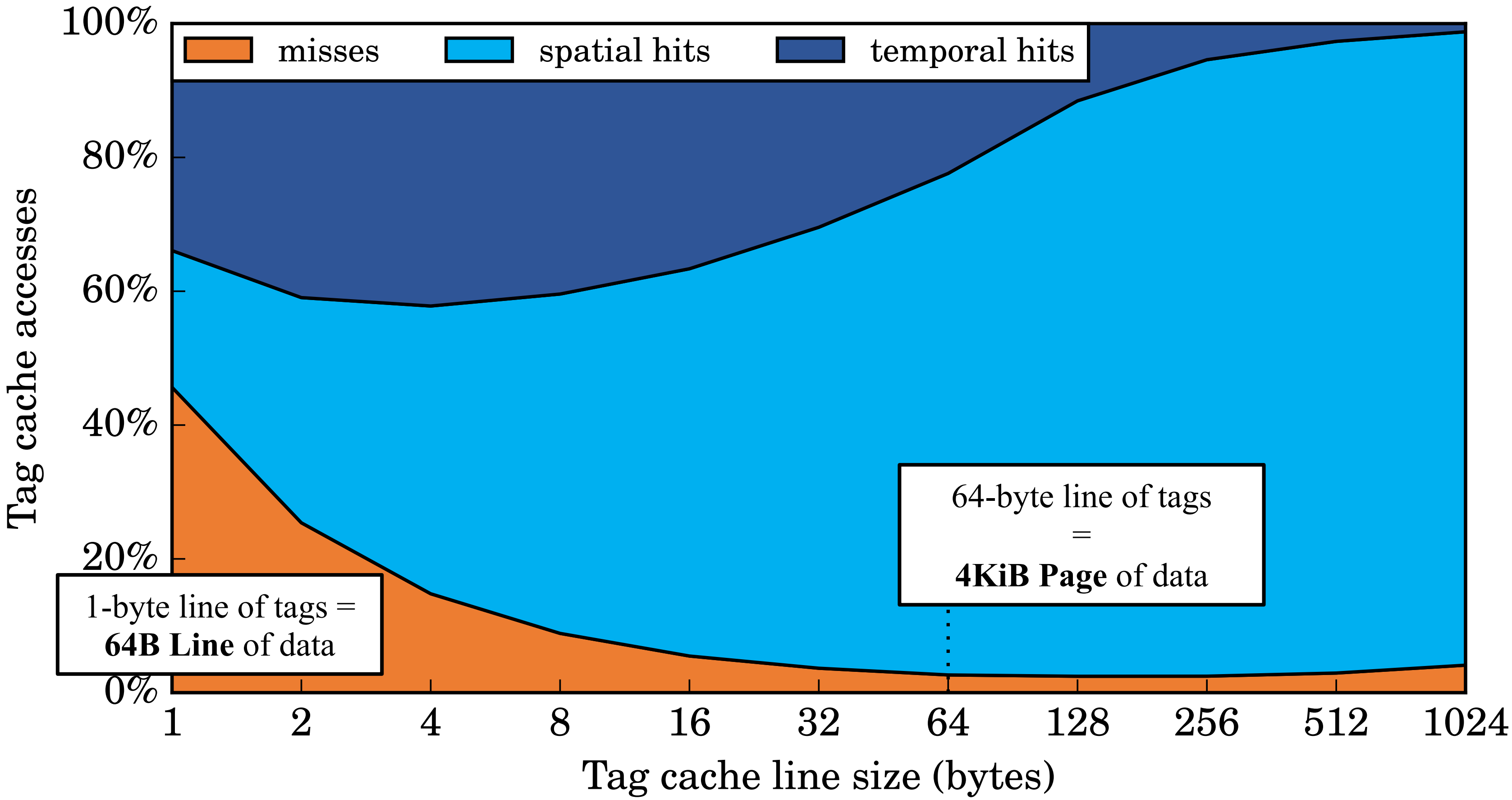
<5% traffic overhead at 256KiB with 16MiB capacity

Why is tag cache more effective than a traditional last-level cache?

Tag Table Cache Locality Analysis

Temporal and Spatial Hits vs. Line Size

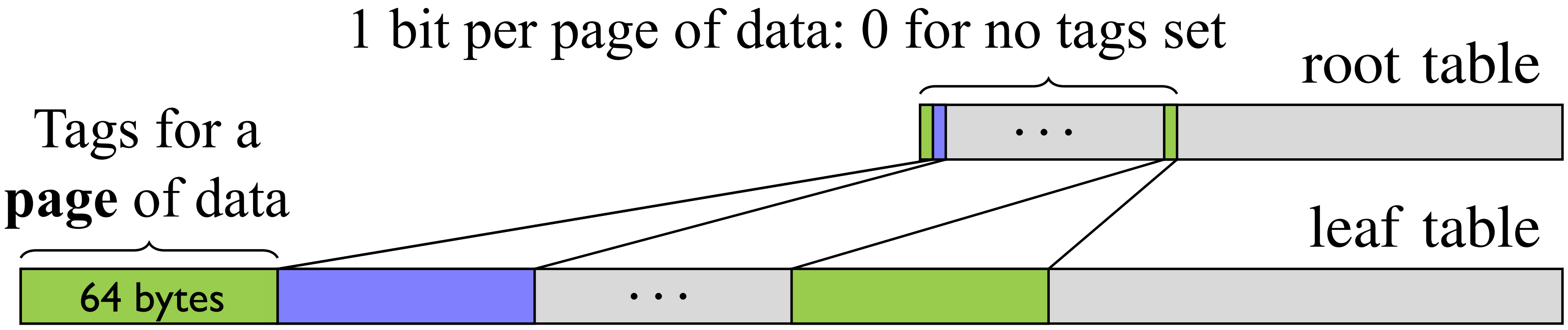
for Earley-Boyer, 256KiB tag cache, 8-way set associative



1-byte line of tags = **64B Line** of data

64-byte line of tags = **4KiB Page** of data

Tag Compression


1 bit per page of data: 0 for no tags set
root table
Tags for a **page** of data
leaf table
64 bytes
. . .
. . .

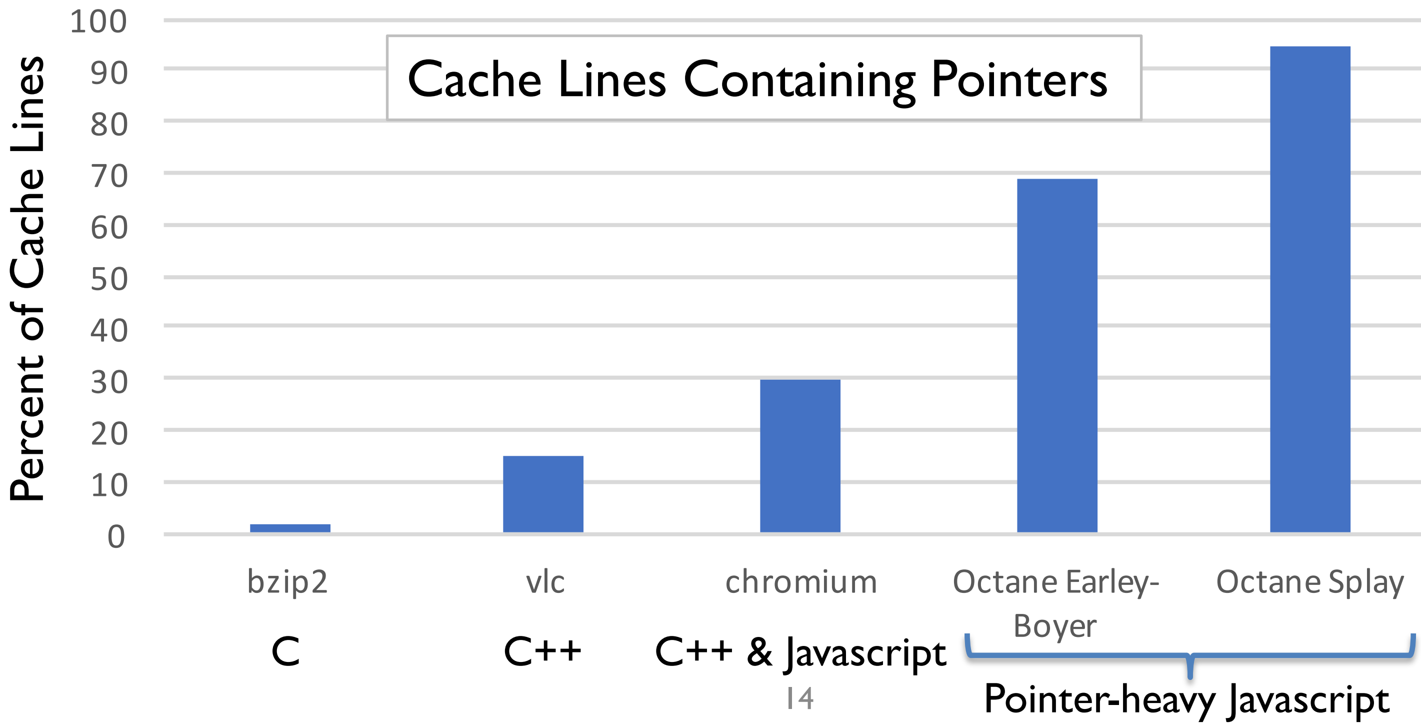
- 2-level tag table
- Each bit in the **root** level indicates all zeros in a **leaf** group
- Reduces tag cache footprint
- Amplifies cache capacity

Use-case 1: Pointer Integrity

- All **virtual addresses** are tagged

  All words that match successful TLB translations

- Similar to our CHERI FPGA implementation

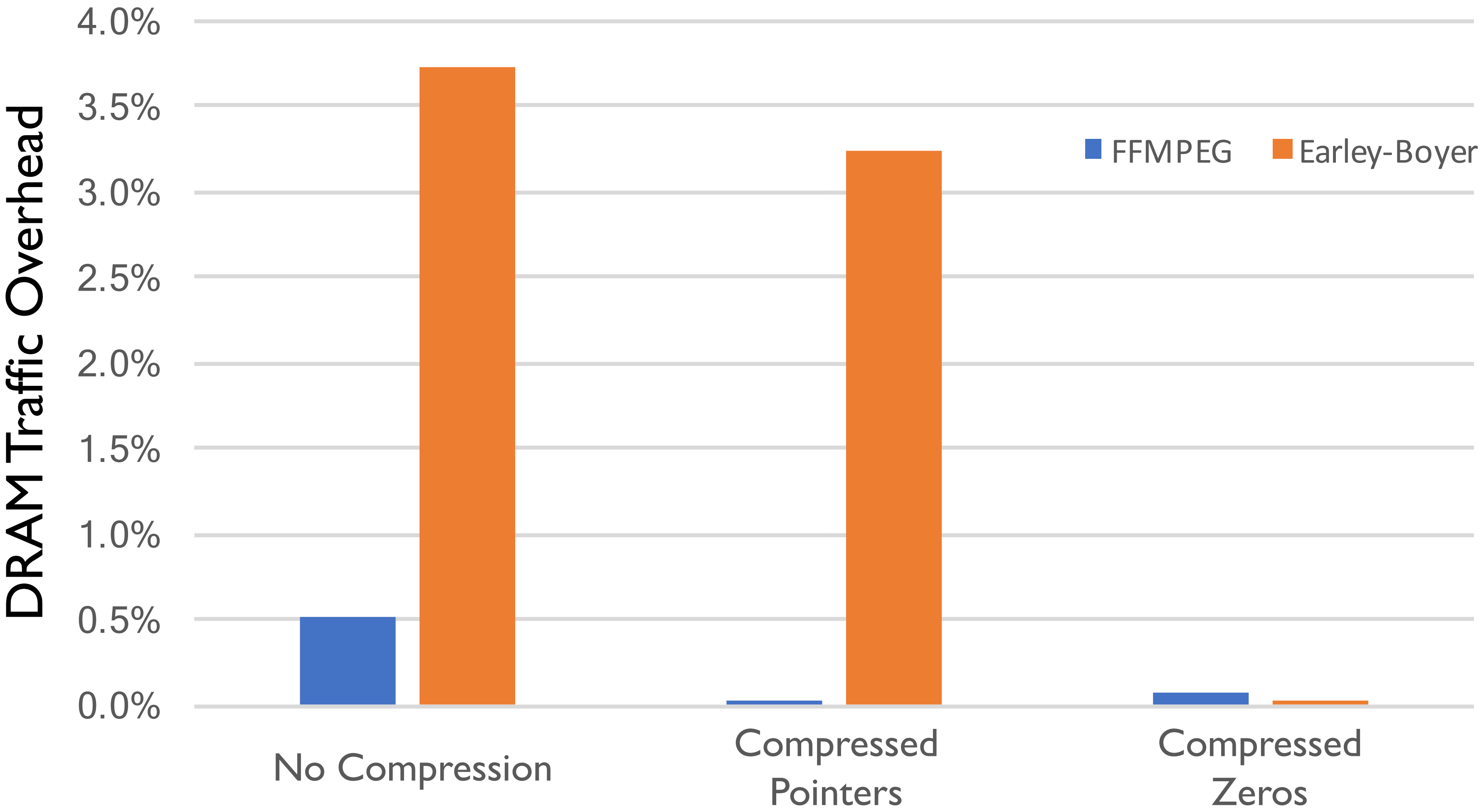


Use-case 2: Zero Elimination

- Tag cache lines that contain zeros
- Eliminate zero cache lines from DRAM traffic
- Can we eliminate more data traffic than the tag table generates?
  - **1.5-2.5%** of lines in DRAM traffic are all zero

    (in our workloads)
  - If we use less than 1% for table traffic, we improve performance!

Overhead with Compression



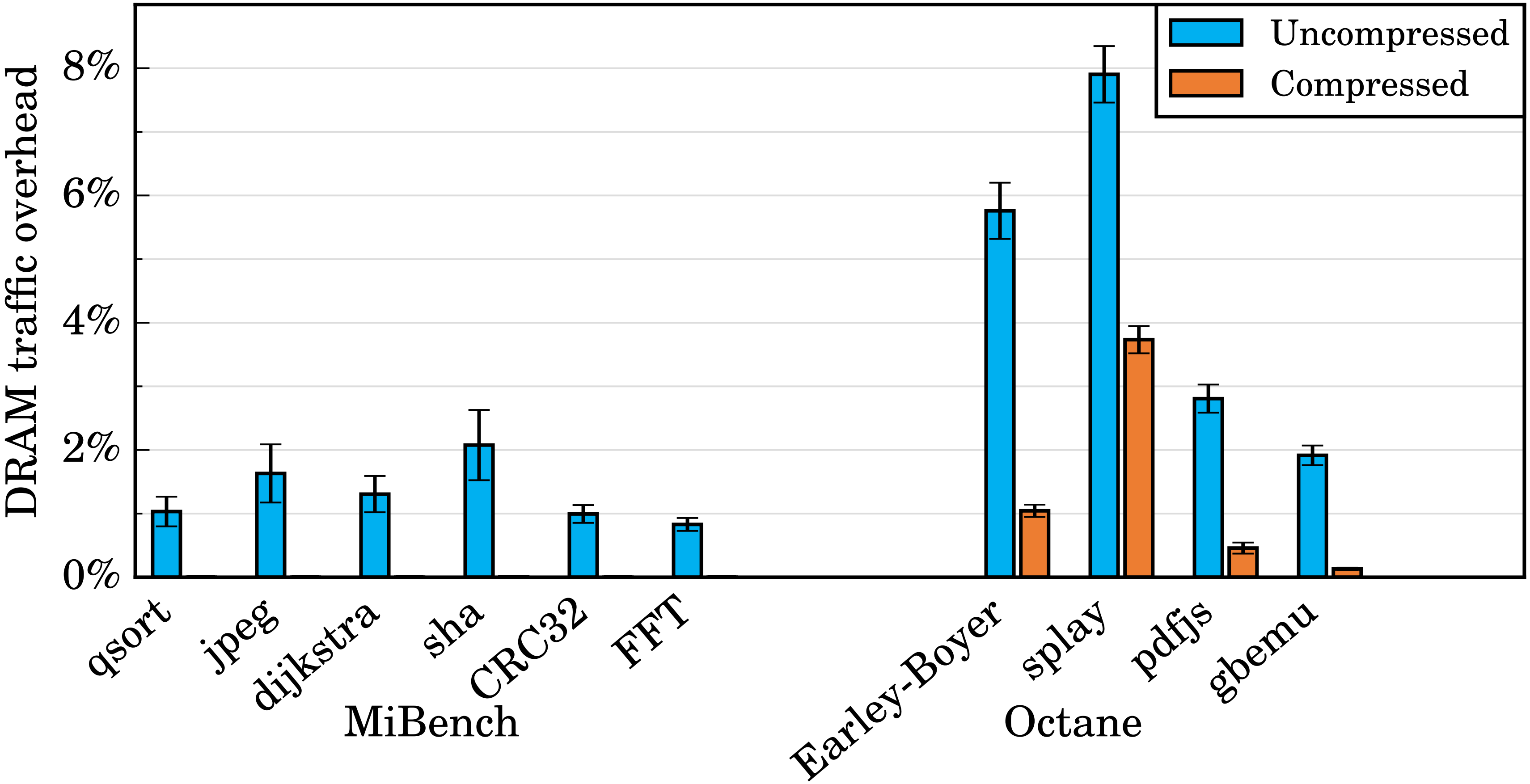
CHERI FPGA Implementation

- 64-bit MIPS implementation with tagged pointers
- 256KiB, 4-way set associative L2 cache
- Parameterizable hierarchical tag controller backed by 32KiB 4-way associative tag cache

Benchmarks in Hardware

DRAM Traffic Overhead in FPGA Implementation

Note: MiBench overheads with compression are approximately zero



Things We've Learned

- A tag table caches extremely well

  Spatial locality pays off for very wide lines

- Simple compression works well for sparse tags
- Single-bit tags in standard memory can require nearly **zero** overhead in the common case

  Pointer tags + zero line elimination could actually net reduce memory accesses for most cases!

Questions?

Jonathan.Woodruff@cl.cam.ac.uk