# The Reuse Cache

## Downsizing the Shared Last-Level Cache

**Jorge Albericio**[1], Pablo Ibáñez[2], Víctor Viñals[2], and José M. Llabería[3]

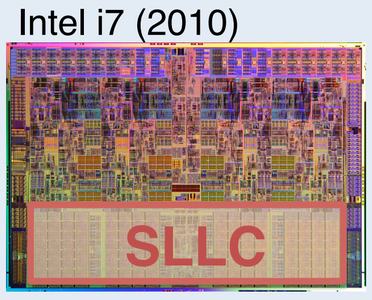
1 UNIVERSITY OF TORONTO

2 Universidad Zaragoza 1542

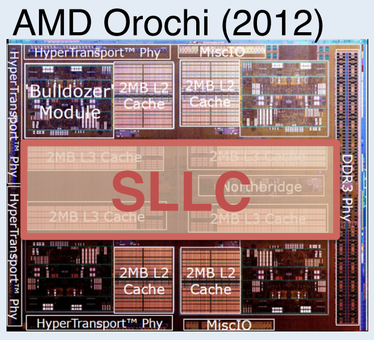
3 UPC

## Motivation (I)

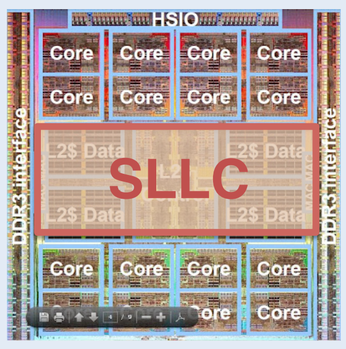
Intel i7 (2010)



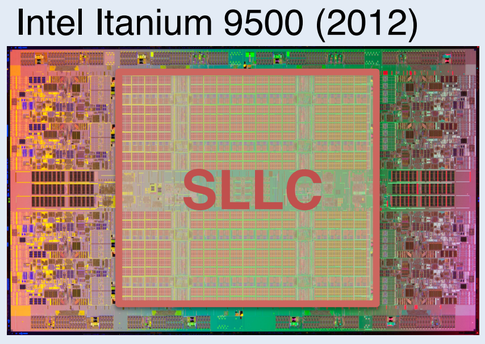

AMD Orochi (2012)




Fujitsu SPARC VIIIfx (2011)




Intel Itanium 9500 (2012)




SLLC is a big part of modern CMPs

**SLLC data**

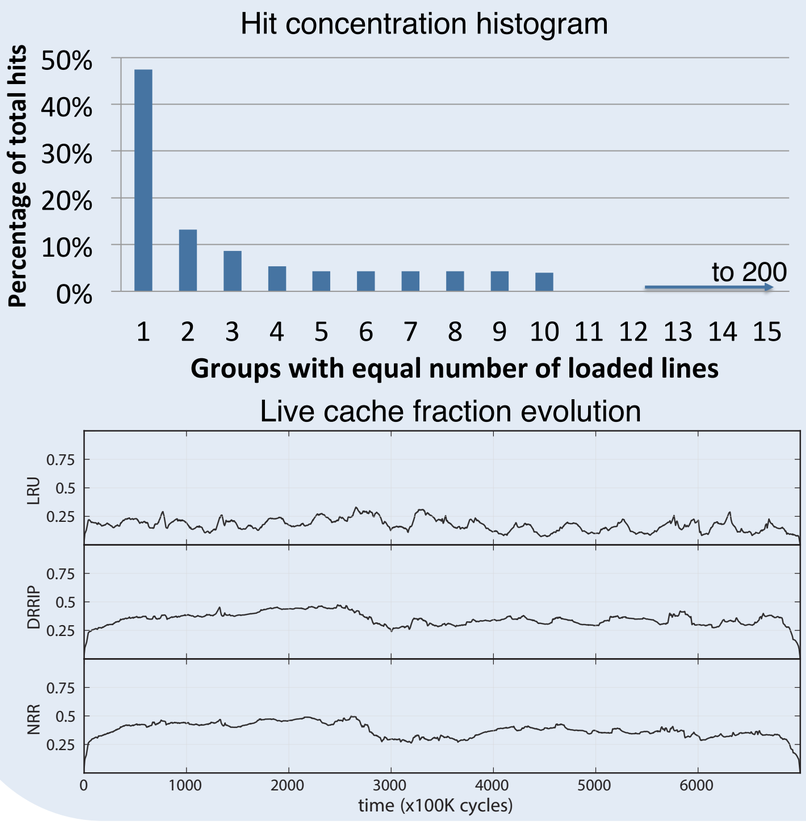
Many hits

Zero hits

A big fraction of the SLLC contents is dead

Opportunity for a **smaller SLLC** that stores only reused data

## Motivation (II)




Hits concentrated in a few lines

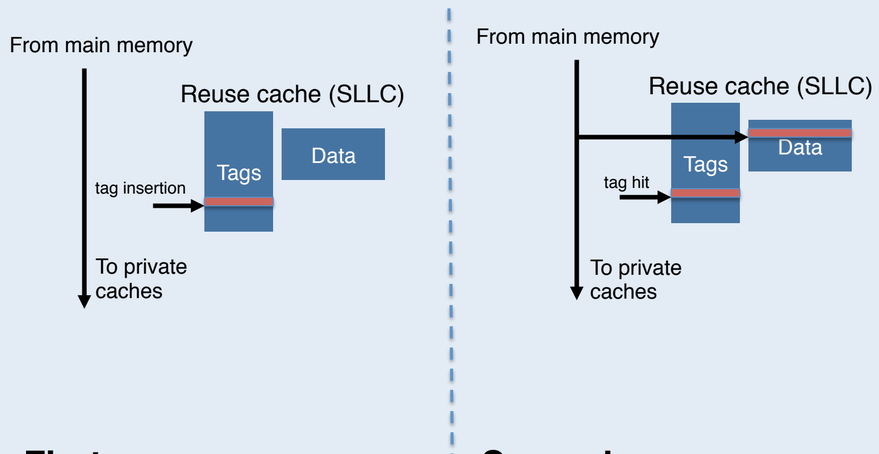
- 5% of all the inserted lines concentrate all the hits
- 0.5% of all the inserted lines concentrate close to 50% of hits

The fraction of live SLLC lines is very small (10-30%)

- On average 78% of lines won't receive any additional access

## Idea

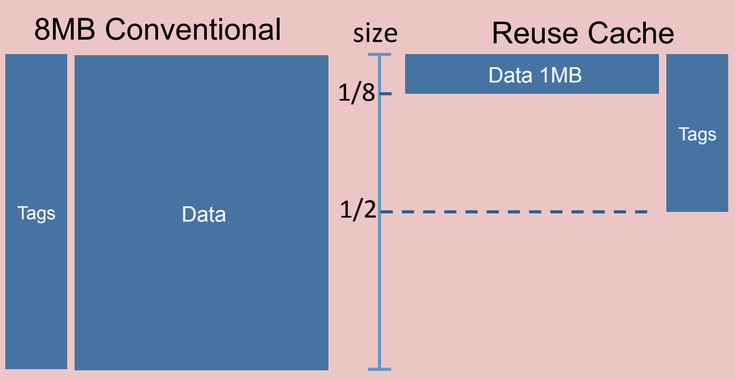
Data is stored only when reuse is detected




**First access:**
Miss → only tag insertion

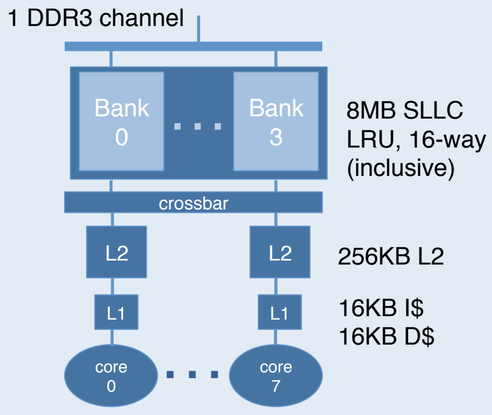
**Second access:**
Tag hit → data insertion (reuse detected)

## Results




**Same average performance* with 84% area savings**

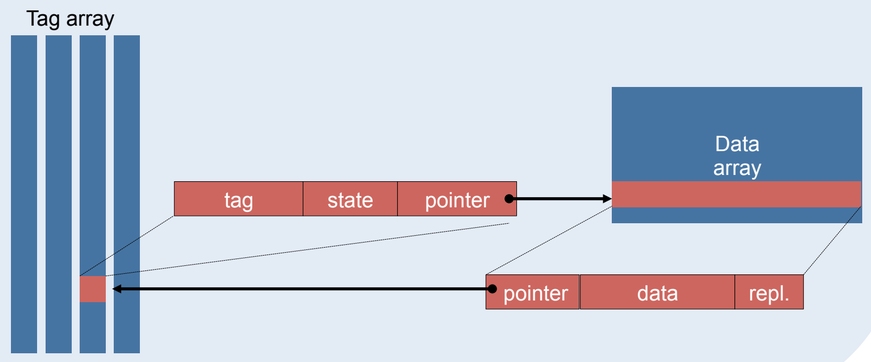
## Methodology




- Simulation: Simics + ruby
- Area and latency: CACTI 6.5
- Workloads
  - Multiprogrammed: 100 SPEC CPU 2006 mixes
  - Parallel: from PARSEC and SPLASH2

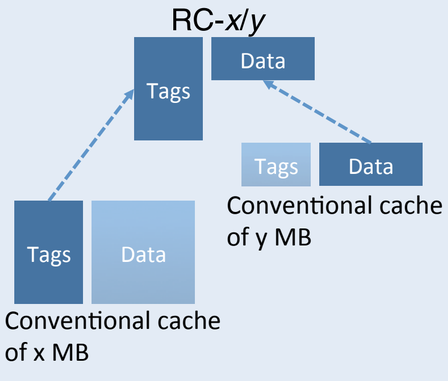
## Organization

- Tag array
  - Entries have associated data or not
  - Set-associative
  - Forward pointers: to indicate corresponding data array entry
- Data array
  - Only stores reused lines
  - Fully- or set-associative
  - Reverse pointers: update tag array entry




## Terminology

RC-*x*/*y* = Reuse cache with tags equivalent to *x* MB, *y* MB of data




## Size exploration