Lossy Compression for Worst-Case Execution Time Analysis of PLRU Caches

David Griffin, Benjamin Lesage

Alan Burns, Rob Davis

RTS York

THE UNIVERSITY of York

Introduction

- Traditional Abstract Interpretation approach
  - Identify a value that is interesting
  - Find an approximation of that value
  - Define map between concrete / abstract states

Introduction

- Problems
  - No guarantee that an interesting value is a useful value
  - No guarantee approximation doesn't discard important information

Lossy Compression

- Lossy Compression is the art of choosing to lose information of little value to the goal
  - Example: MP3 compression discards audio data that would be impossible to hear
  - Caveat: Not typically applied to state spaces...

Lossy Compression

- Effectively, Lossy Compression is already used in Abstract Interpretation
  - Takes big state space, makes it smaller by discarding/approximating information
- However, it is not explicitly mentioned or used...

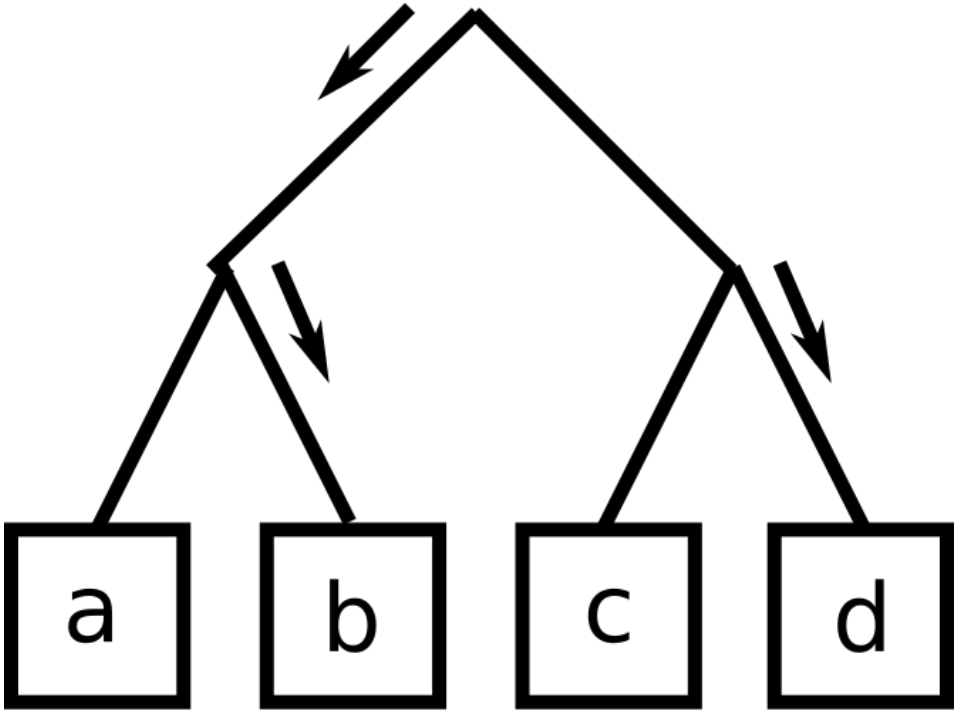
Lossy Compression

- Lossy Compression approach
  - Write down the different types of information in the system
  - Experiment/Reason about what would happen if each type were discarded/approximated
  - Discard Information from states until state space is of a manageable size

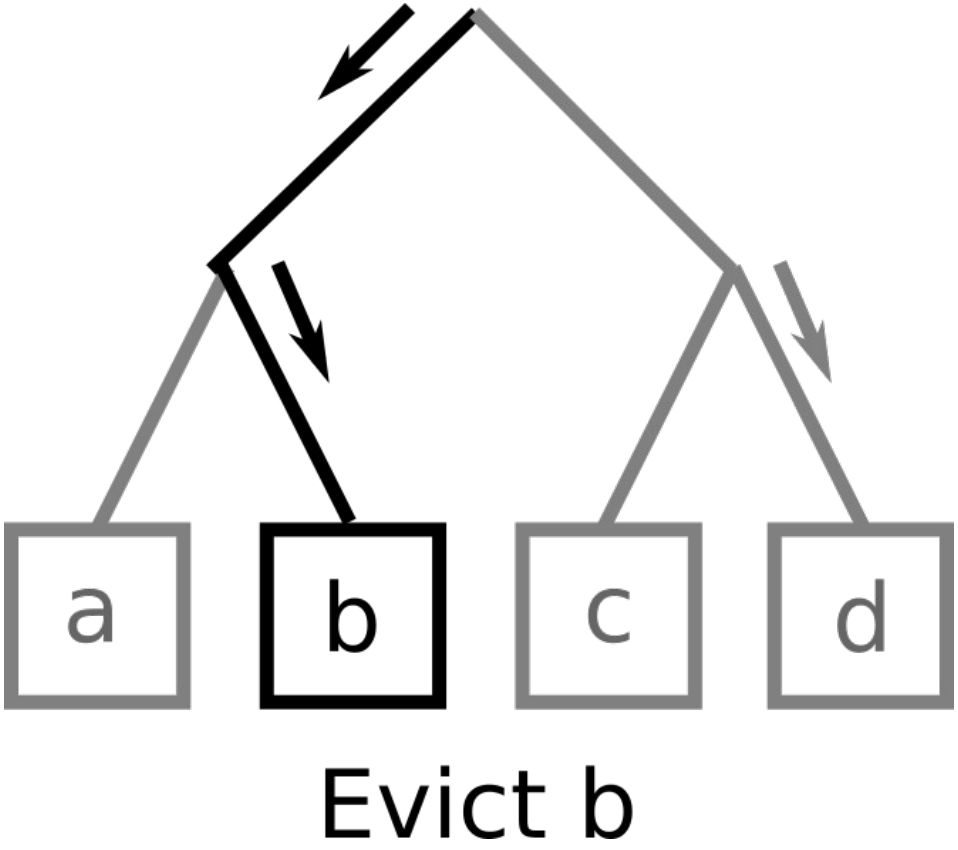
PLRU Cache

- PLRU cache is commonly used
  - "nearly" as good as LRU
  - Less expensive to implement
- Based on a binary tree
- Uses much smaller silicon area

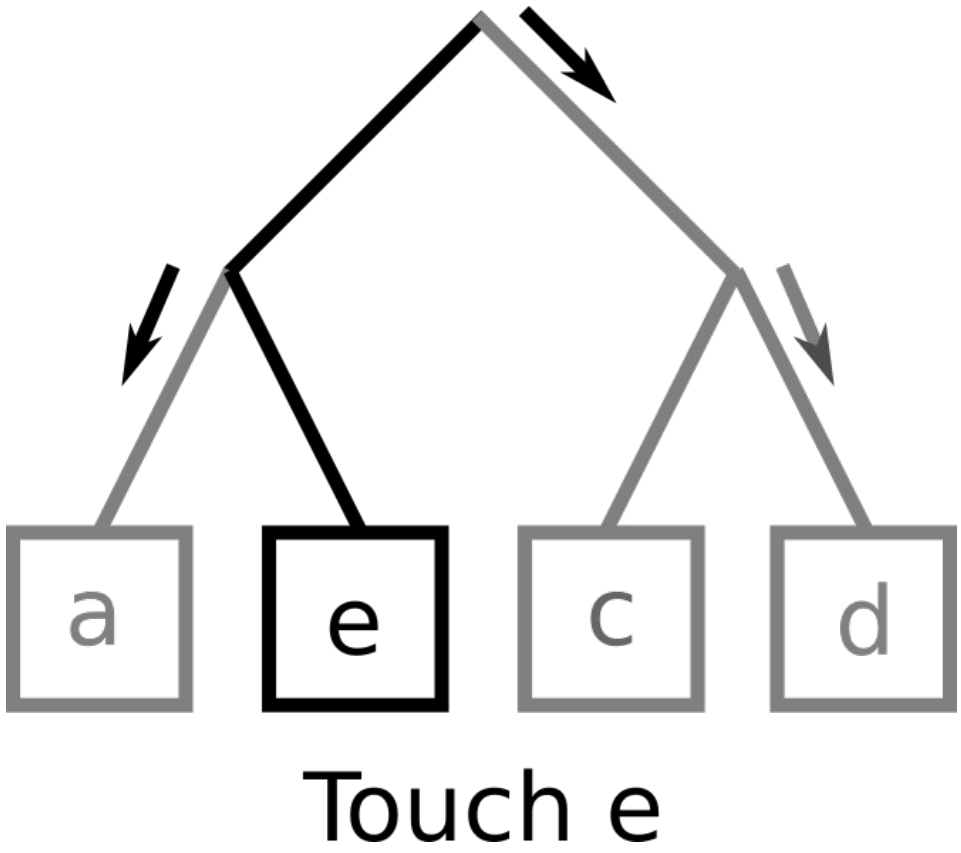
PLRU Cache



PLRU Cache



PLRU Cache



PLRU Cache

- Algorithm

```
if classify(cs, memloc) == Miss:
  evict(cs, memloc)
touch(cs, memloc)
```

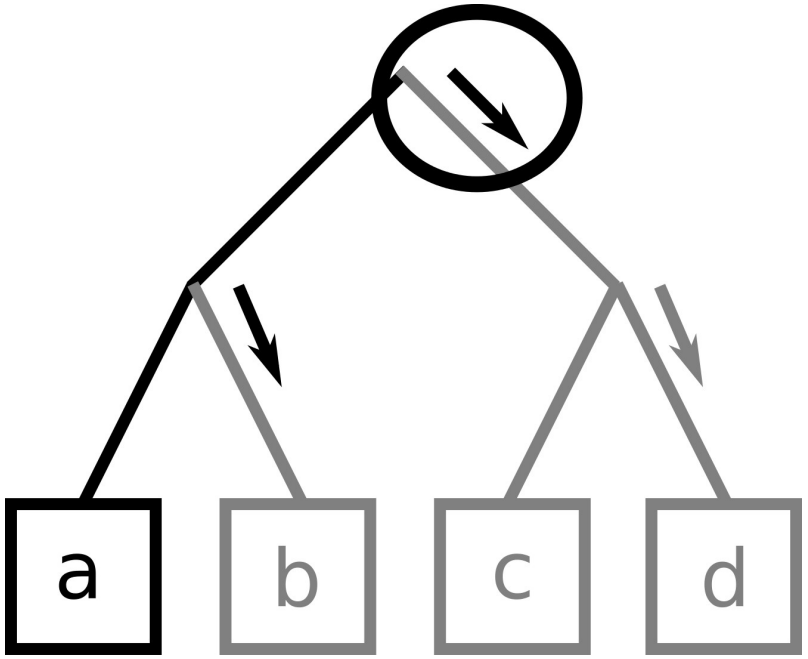
- Nearly always behaves like LRU

PLRU Cache

- "Nearly Always" is a problem for WCET
- Corner cases where PLRU behaves very differently to LRU
  - Element kept in cache that hasn't been accessed
  - Elements evicted quicker than in LRU

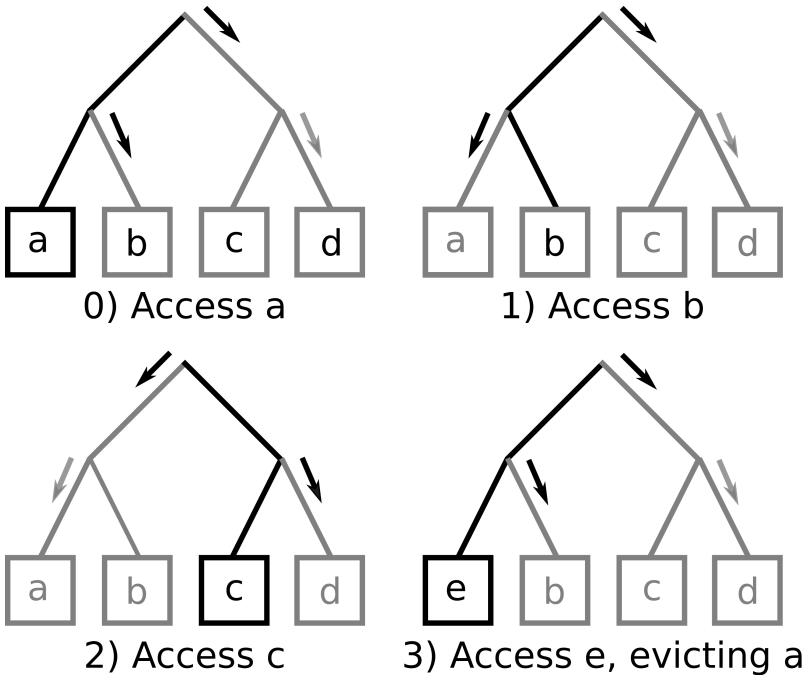
Memory Block Protection

- If **a** is accessed, it's shared pointer with **b** is set away
- If **a** is repeatedly accessed, **b** is *never evicted*

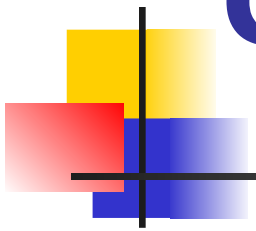


Speedy Eviction

- log(n) pointers protect any single element
- So an element can be evicted in log(n) + 1 accesses
  - If these are the right accesses

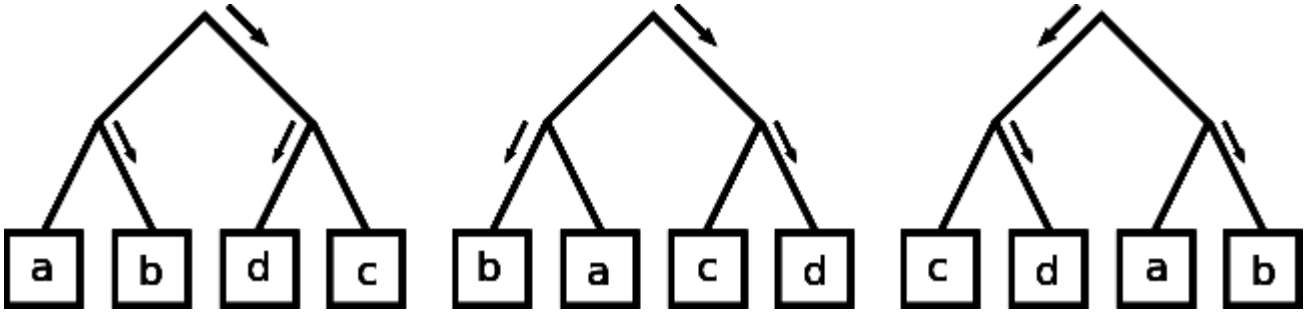


Current Techniques



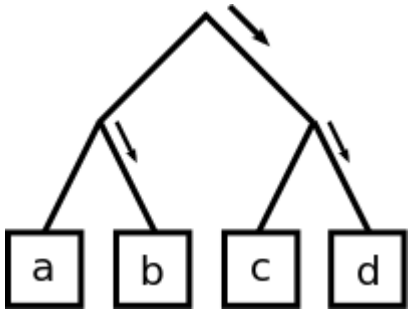
- Grunde & Reineke's Potential Leading Zeroes approach
  - Gives a (partial) Must analysis
  - No May analysis
- Collecting Semantics
  - Expensive for large problems

Collecting Semantics



- Some cache states have the same behaviour

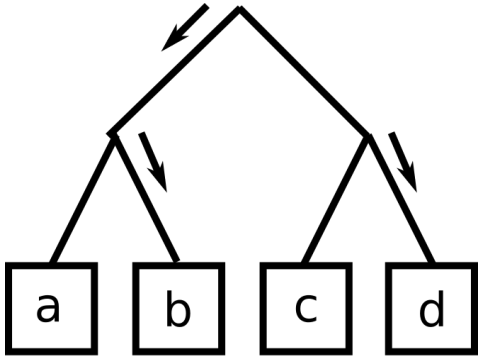
Collecting Semantics



- A behaviour can be "named" by flipping all pointers to a fixed direction
- Value of information lost: 0

Information in PLRU Cache State

- 3 Types of Information
  - Cache Lines
  - Tree Structure
  - Pointers
- 3 Operations in algorithm
  - Classify, Evict, Touch

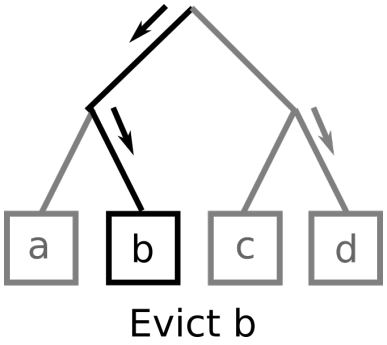


Information Inside Cache States

- Classify
  - Determines if Eviction should happen
  - Uses Cache Lines as input
  - If uncertain, would have to consider both the possibilities of performing an eviction or not performing an eviction

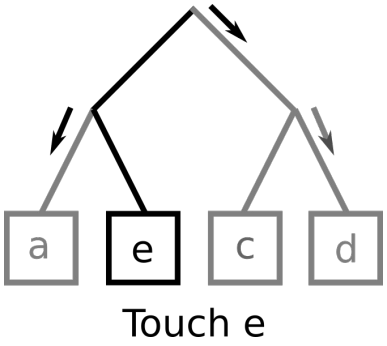
Information Inside Cache States

- Evict
  - Follows pointers and replaces the pointed at element with a new element
  - Uses Pointers and Tree structure as input; Overwrites Cache Lines
  - If Pointer/Tree Structure uncertain, could have to consider each element of cache being evicted



Information Inside Cache States

- Touch
  - Sets all pointers on path to cache line away
  - Uses Tree Structure as input; Overwrites Pointers
  - If Tree Structure uncertain, could have to consider cache element in any position



Information Inside Cache States

- Every cache access will perform Classify and Touch
- But not every access will perform Evict
  - When optimised for cache, expect > 90% hit rate

Information Inside Cache States

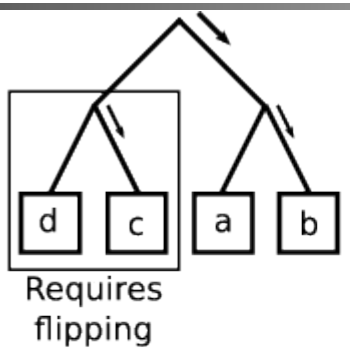
	Usage Freq	Overwrite Freq	Worst case uncertainty
Cache Lines	High	Low	2 (2n)
Pointers	Low	High	n
Tree Structure	High	High	n

- So Pointers are used infrequently and overwritten frequently
  - Good candidate tor discarding

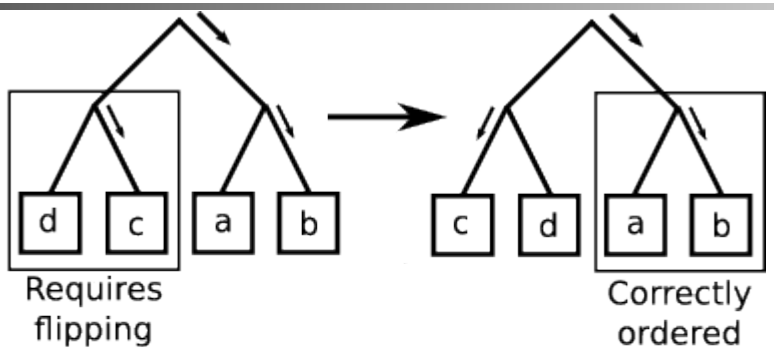
Revisiting Behaviour Naming

- As pointers may now be unknown, need to revisit how behaviours are named
- Instead of flipping trees based on pointer, do so based on cache lines and tree structure
- Accomplished by a recursive sort

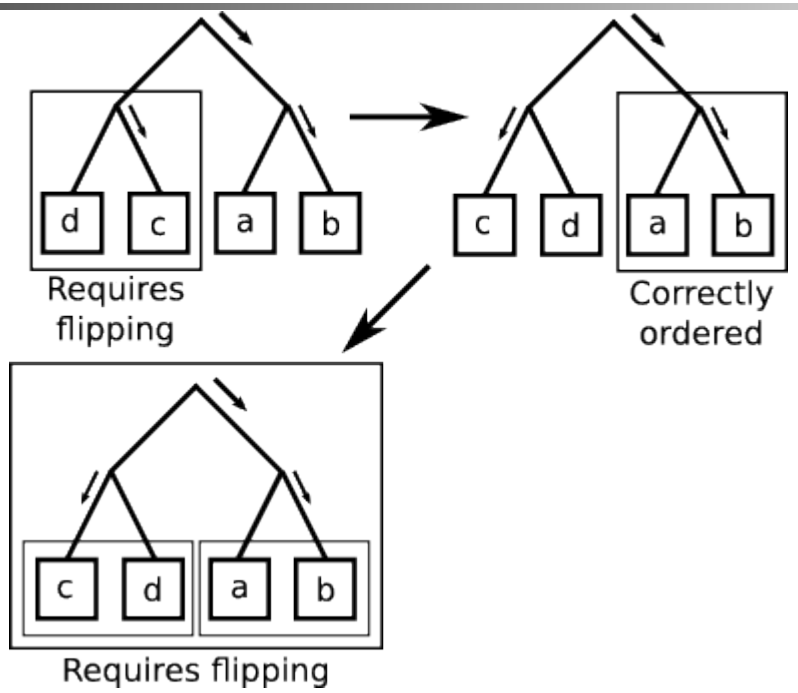
Revisiting Behaviour Naming



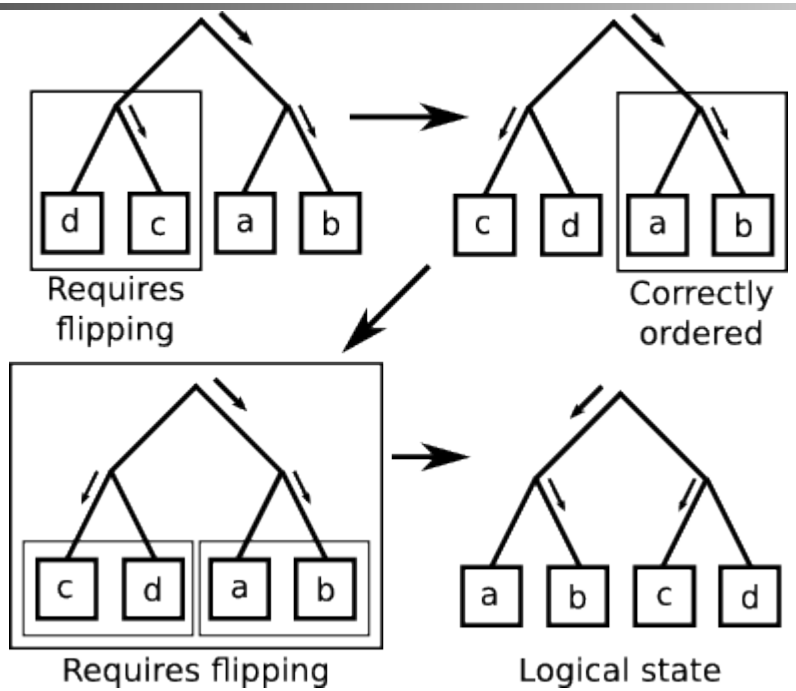
Revisiting Behaviour Naming



Revisiting Behaviour Naming



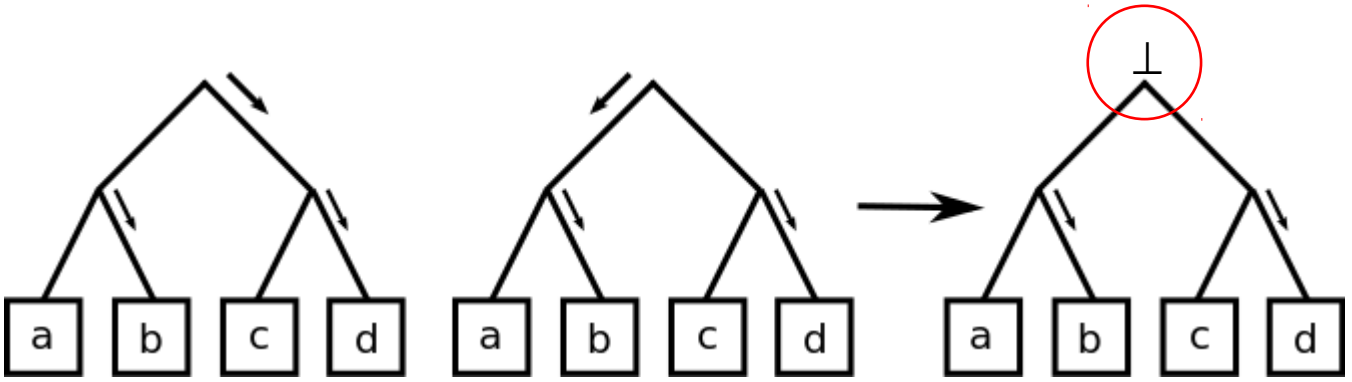
Revisiting Behaviour Naming



- This new name is the cache signature

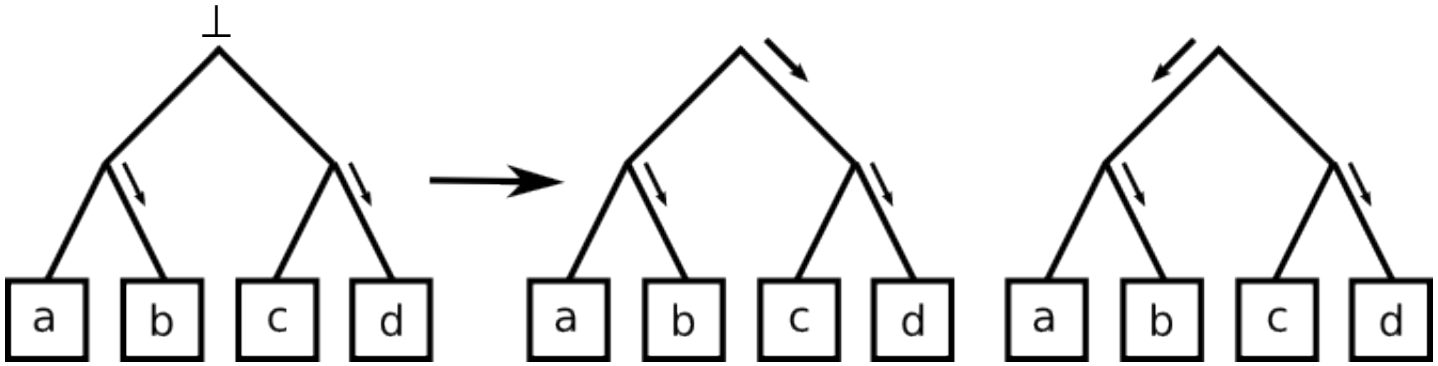
Full Tree Analysis

- Merge all cache states with the same signature
- If Pointers differ, set conflicting Pointers to be ⊥ (Unknown)



Full Tree Analysis

- When encountering an unknown pointer, consider both possibilities



Full Tree Analysis

- To perform classification, classify on each state being considered
  - If a Must in all states, Must overall
  - If a Must in some states, May overall
  - If a Must in no states, Miss overall

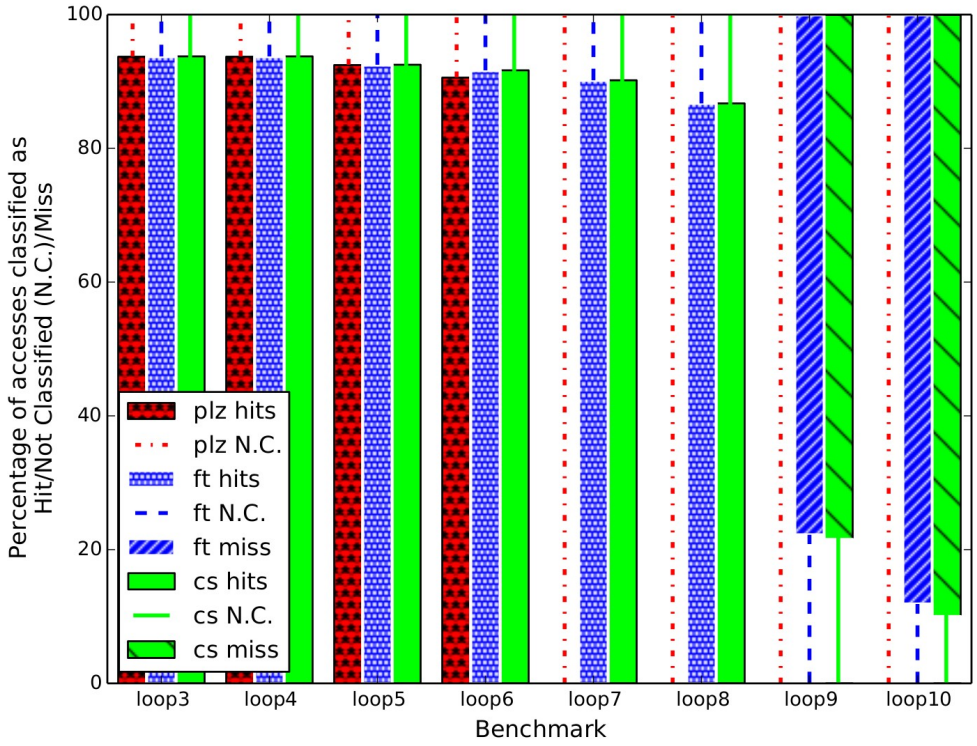
Evaluation

- Must analysis evaluated against Grund and Reineke's PLRU-plz analysis [17]
- Must/May analysis evaluated against Collecting Semantics

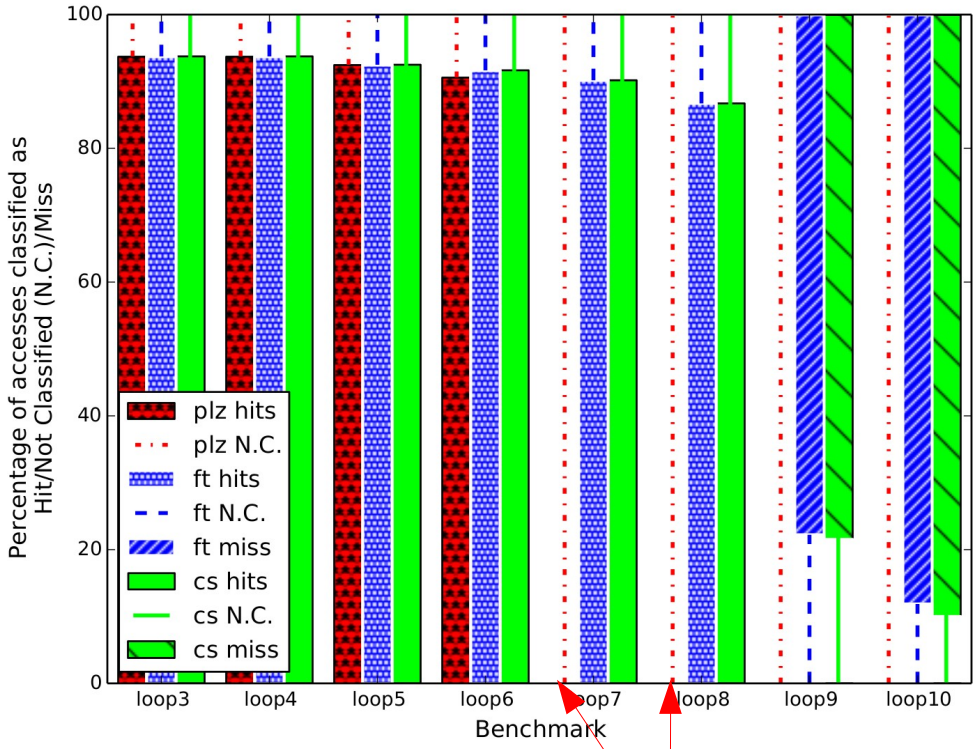
Evaluation

- Synthetic Benchmarks from Grund and Reineke [17]
- Loop(n): Loop of n different memory accesses, repeated 16 times
- Random(n): 100 random memory accesses from range 1..n

Evaluation

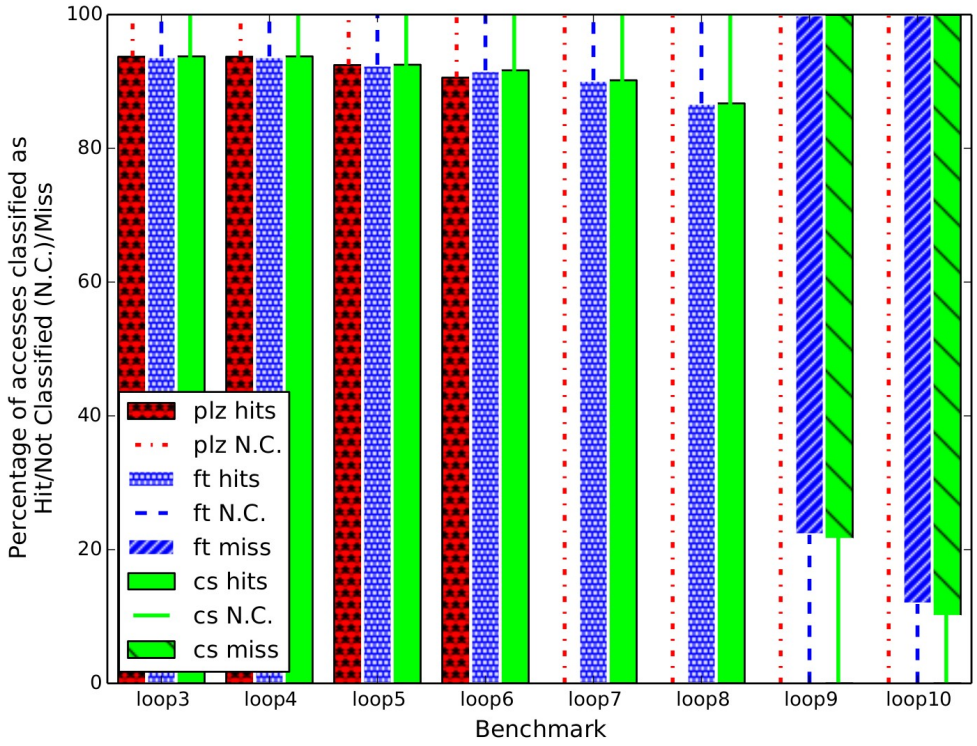

Percentage of accesses classified as Hit/Not Classified (N.C.)/Miss
100
80
60
40
20
0
plz hits
plz N.C.
ft hits
ft N.C.
ft miss
cs hits
cs N.C.
cs miss
loop3
loop4
loop5
loop6
loop7
loop8
loop9
loop10
Benchmark

Evaluation



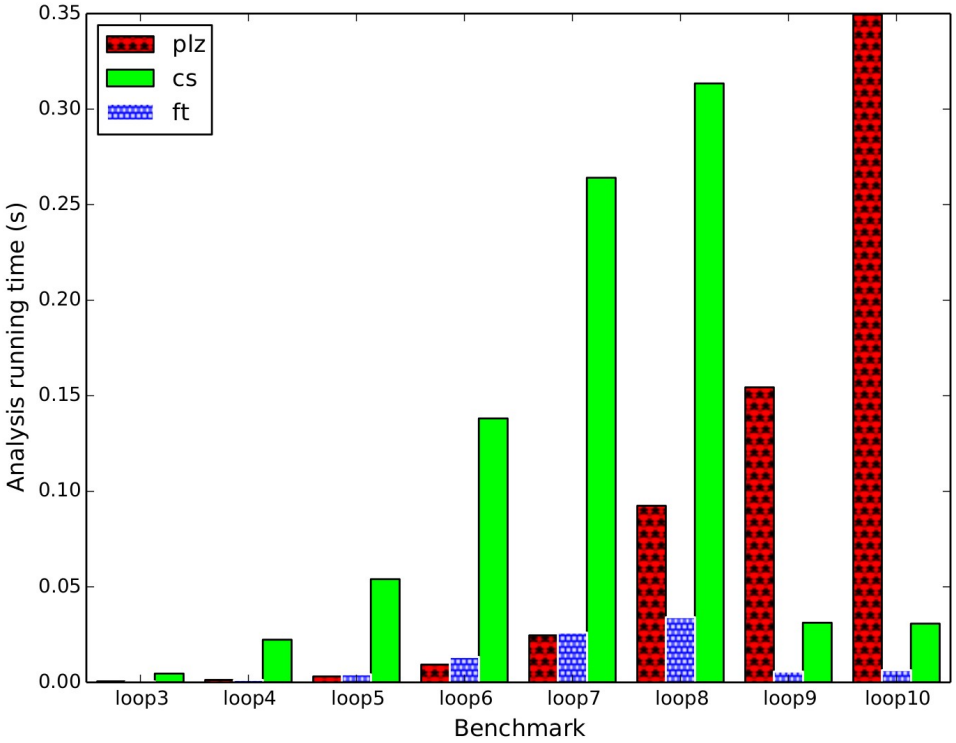
plz cannot analyse these

Evaluation



ft achieves almost the same results as cs

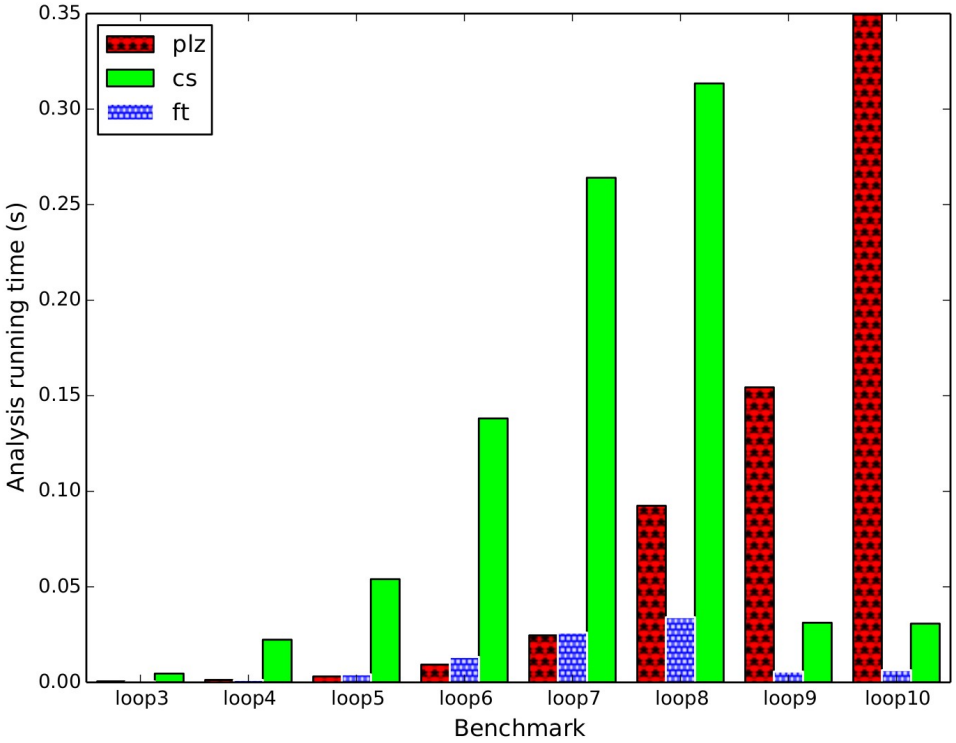
Evaluation



RTS York

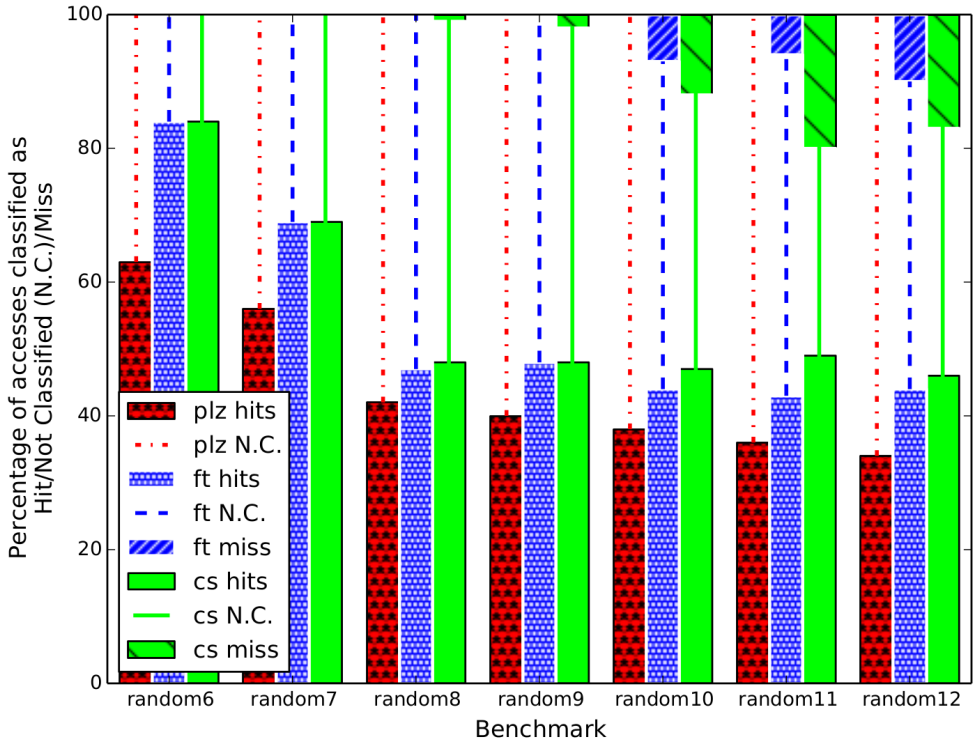
THE UNIVERSITY of York

Evaluation

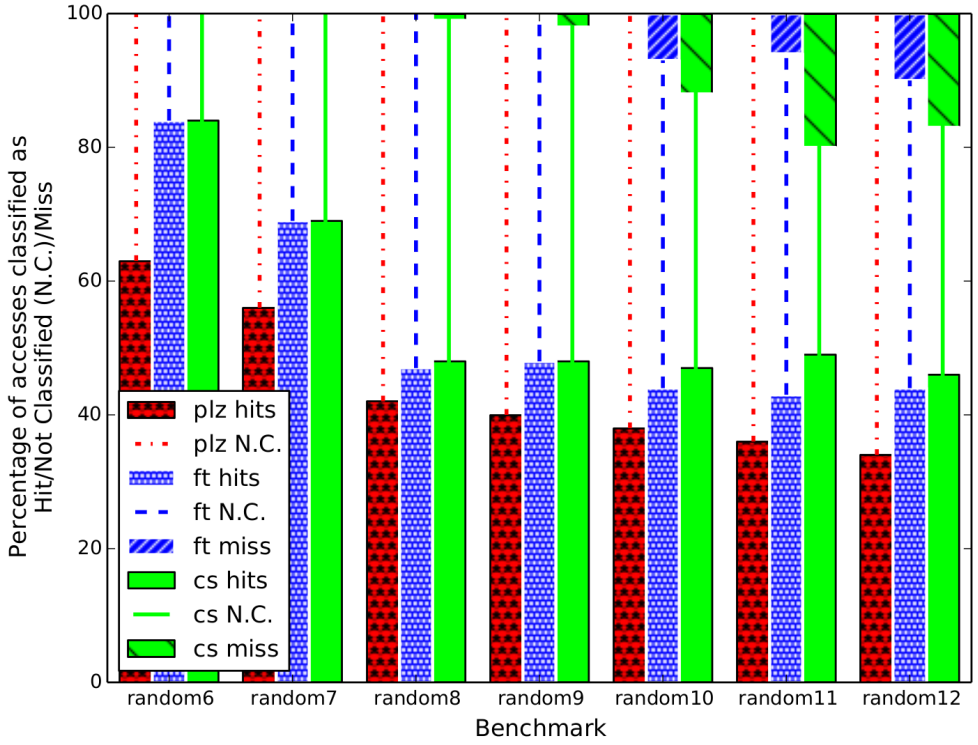


ft is faster than cs

Evaluation

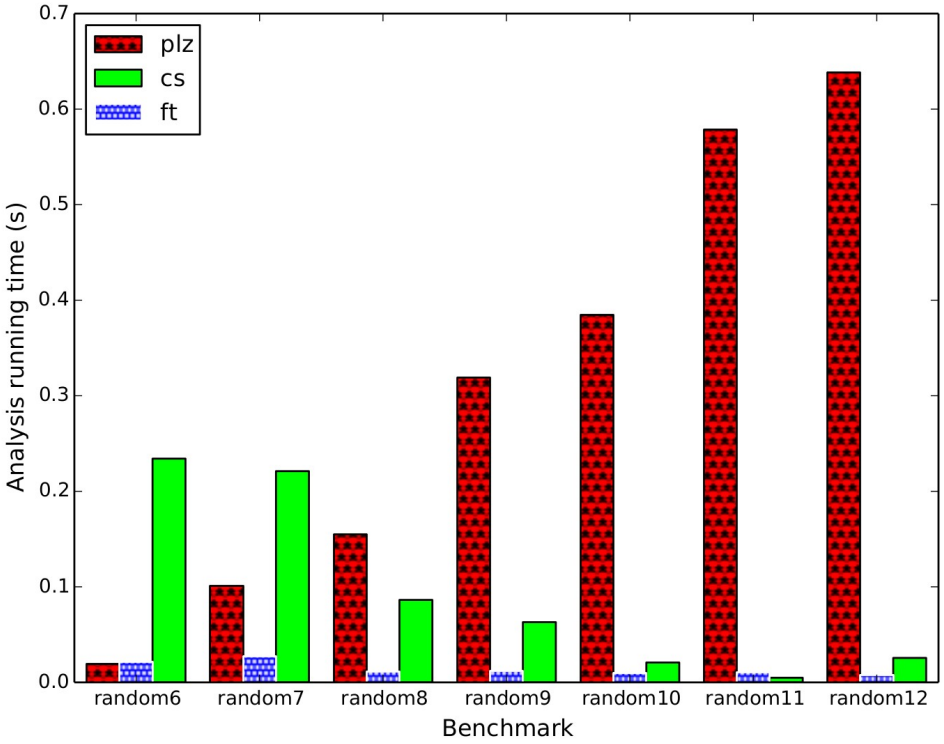

Percentage of accesses classified as Hit/Not Classified (N.C.)/Miss
100
80
60
40
20
0
plz hits
plz N.C.
ft hits
ft N.C.
ft miss
cs hits
cs N.C.
cs miss
random6
random7
random8
random9
random10
random11
random12
Benchmark

Evaluation



Similar results, but May analysis is more pessimistic

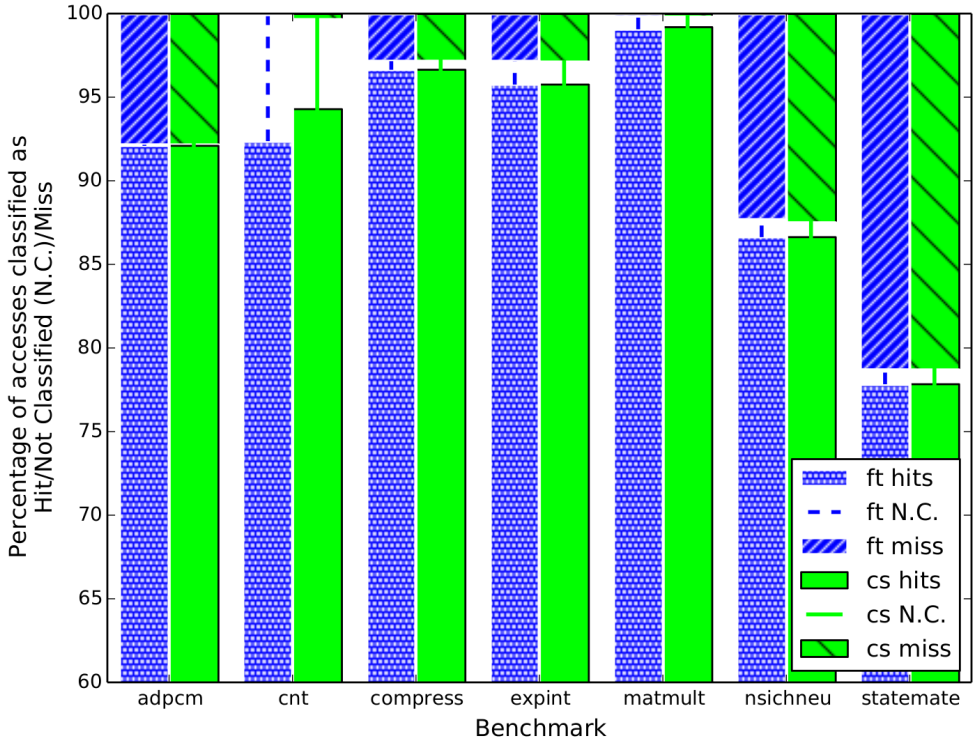
Evaluation


plz
cs
ft
Analysis running time (s)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
random6
random7
random8
random9
random10
random11
random12
Benchmark

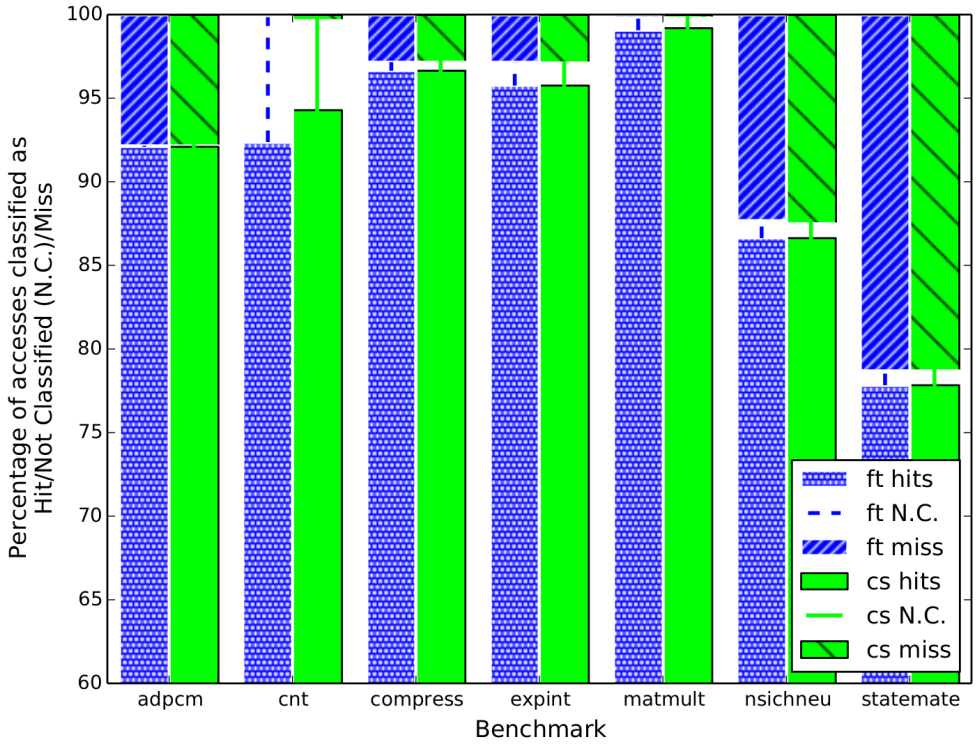
Evaluation

- Mälardalen + PapaBench
  - Compiled for MIPS
  - Interrogated by Heptane analyser
  - Multipath, but no path constraints
- 8-way cache, 32 byte line size
  - 256 byte cache
- PLRU-plz was unable to analyse these benchmarks due to memory usage

Evaluation

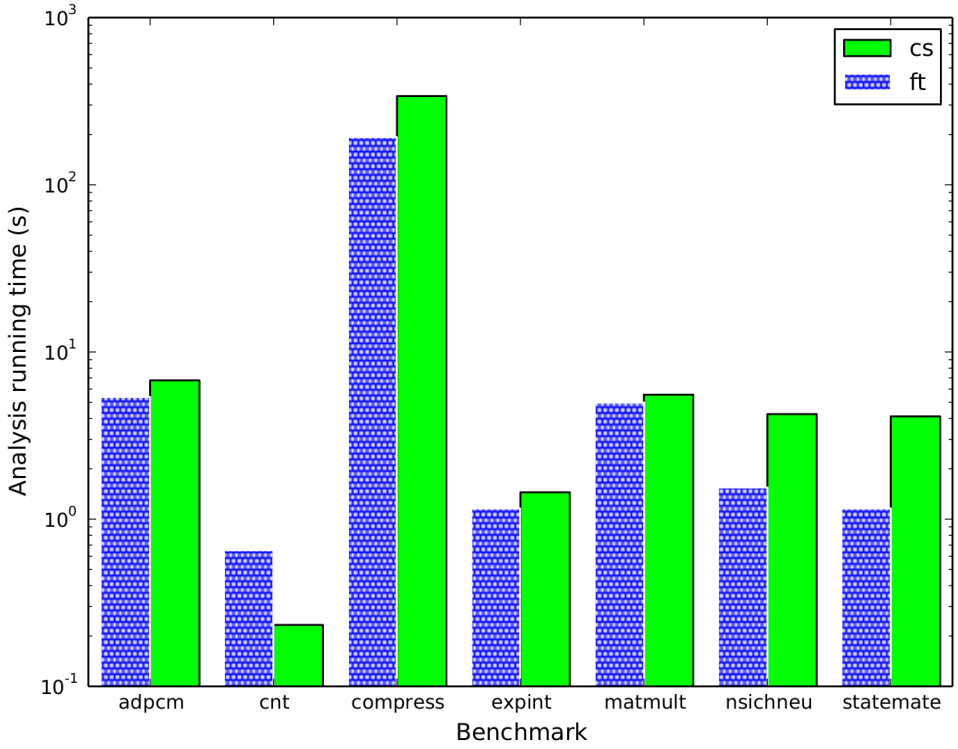

Percentage of accesses classified as Hit/Not Classified (N.C.)/Miss
100
95
90
85
80
75
70
65
60
adpcm
cnt
compress
expint
matmult
nsichneu
statemate
Benchmark
ft hits
ft N.C.
ft miss
cs hits
cs N.C.
cs miss

Evaluation

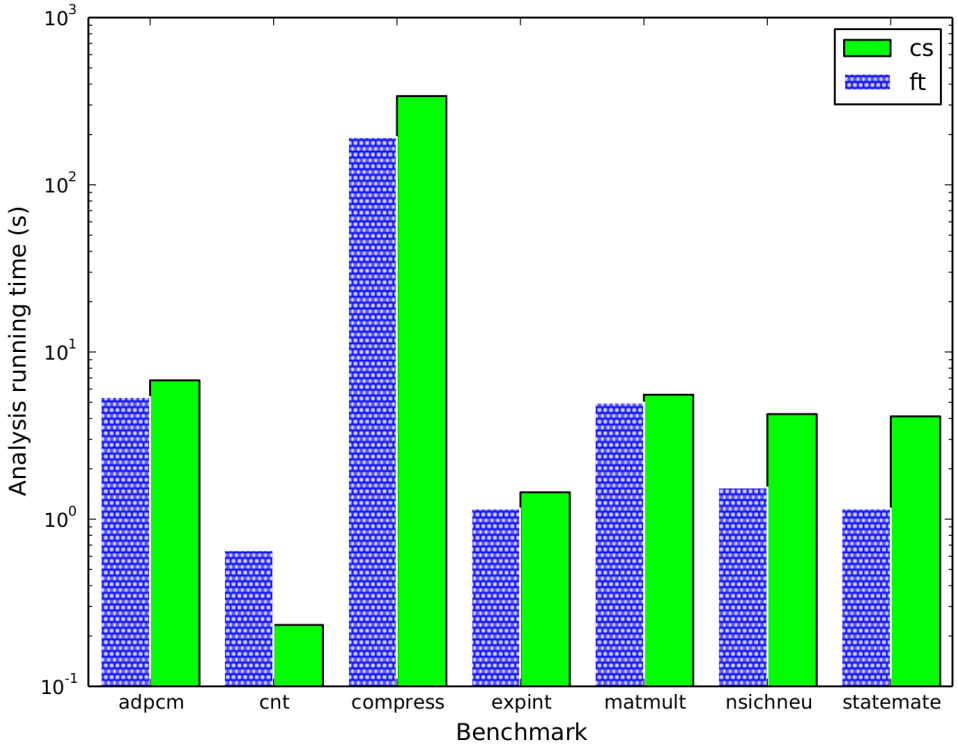


Similar accuracy on 'real' benchmarks

Evaluation

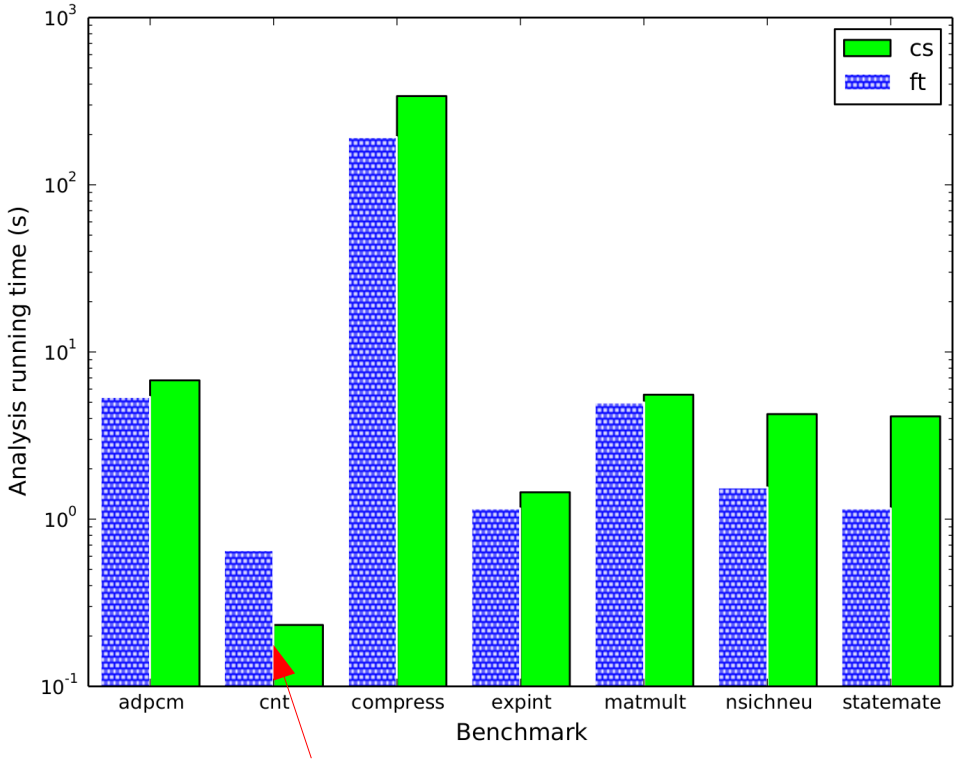

cs
ft
Analysis running time (s)
10^3
10^2
10^1
10^0
10^{-1}
adpcm
cnt
compress
expint
matmult
nsichneu
statemate
Benchmark

Evaluation



Normally faster

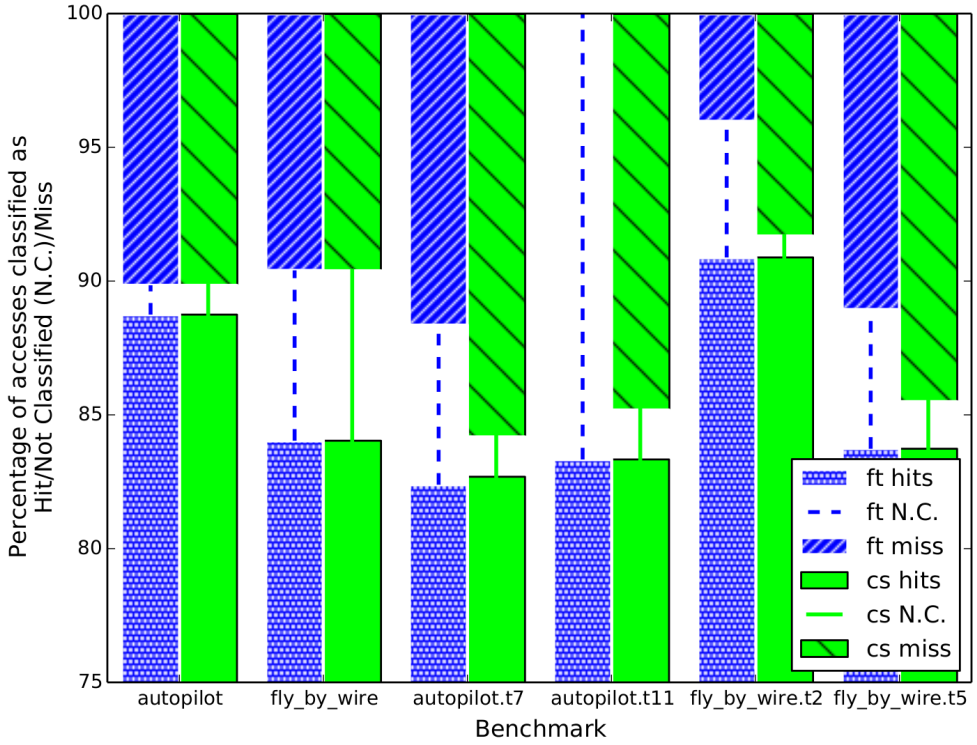
Evaluation



Normally faster

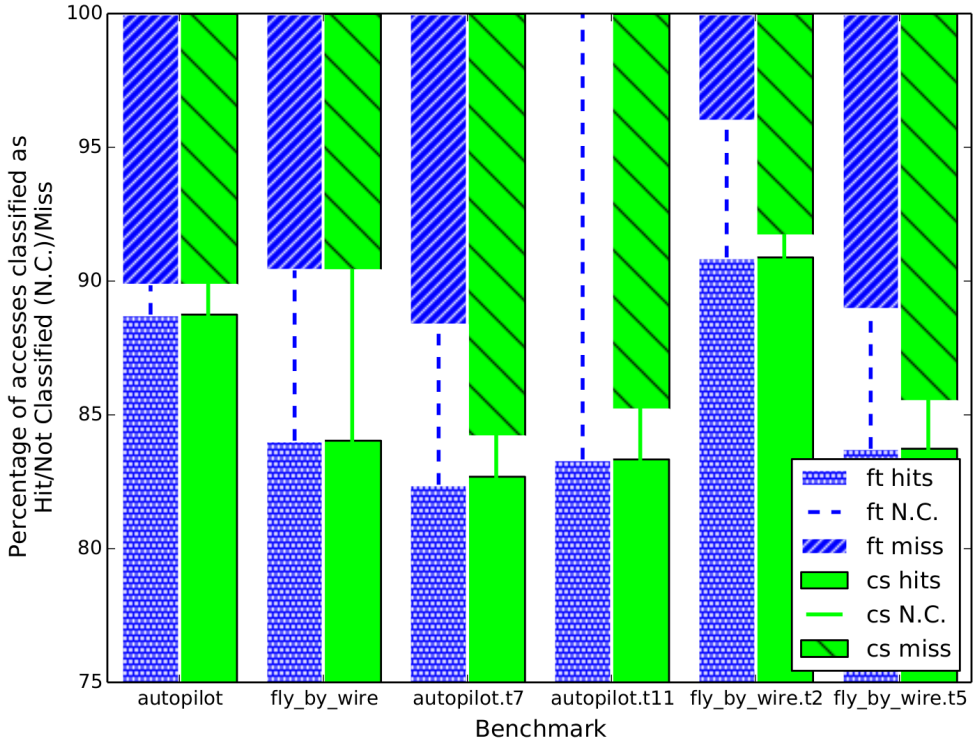
… but not on smaller benchmarks

Evaluation


Percentage of accesses classified as Hit/Not Classified (N.C.)/Miss
100
95
90
85
80
75
autopilot
fly_by_wire
autopilot.t7
autopilot.t11
fly_by_wire.t2
fly_by_wire.t5
Benchmark
ft hits
ft N.C.
ft miss
cs hits
cs N.C.
cs miss

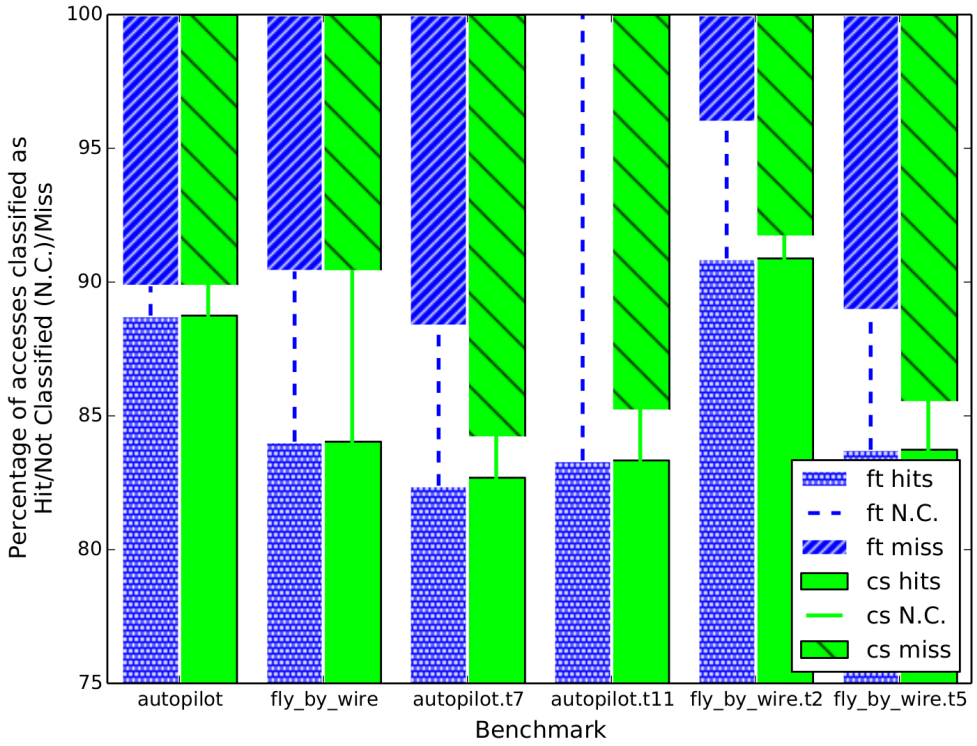
Evaluation

Good accuracy on larger benchmarks



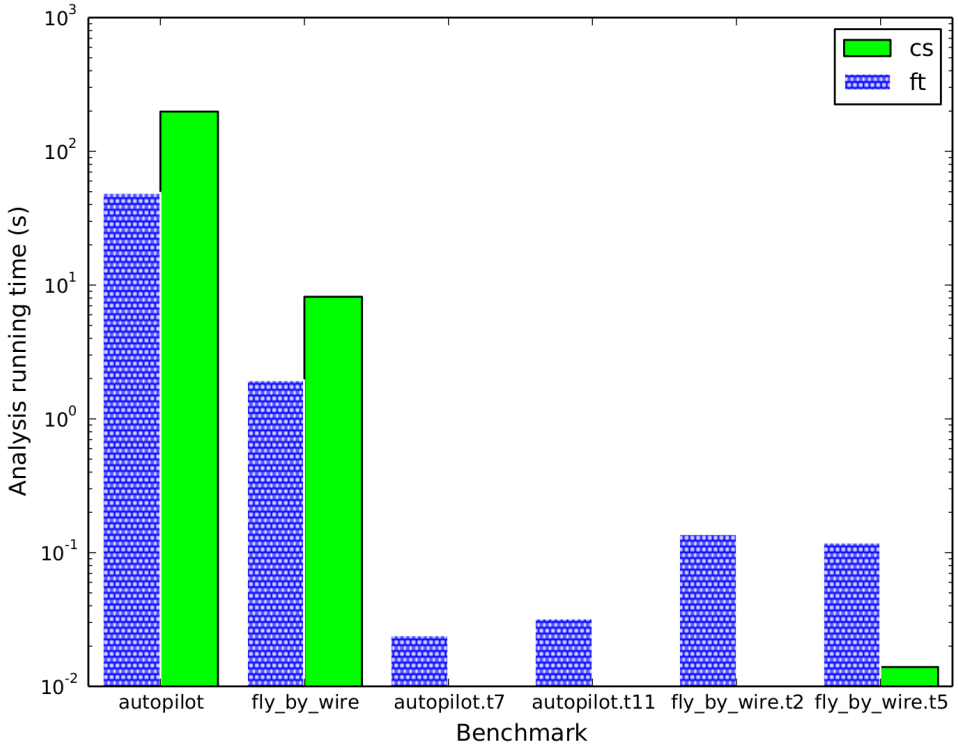
Evaluation

Good accuracy on larger benchmarks



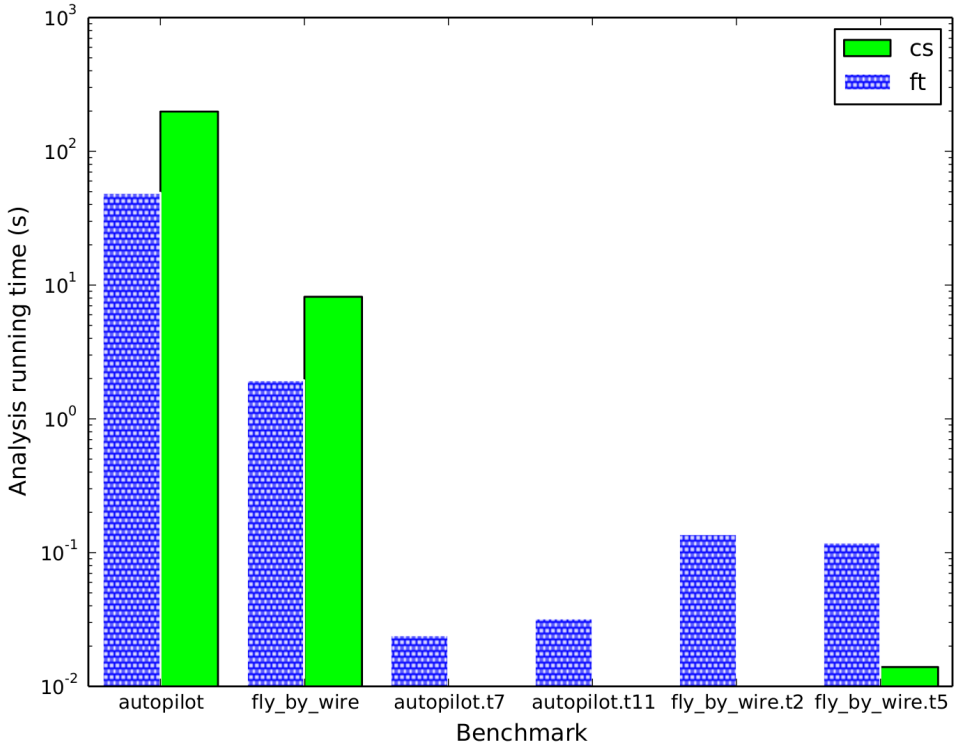
… but more pessimistic on shorter ones

Evaluation


cs
ft
Analysis running time (s)
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
autopilot
fly_by_wire
autopilot.t7
autopilot.t11
fly_by_wire.t2
fly_by_wire.t5
Benchmark

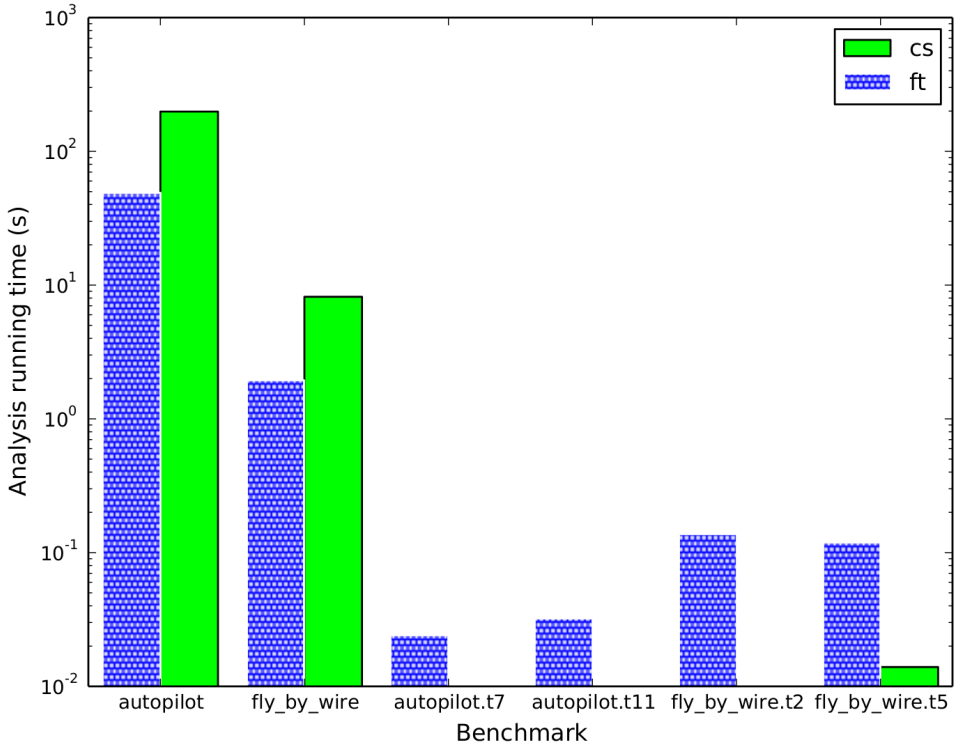
Evaluation

Faster on bigger benchmarks



Evaluation

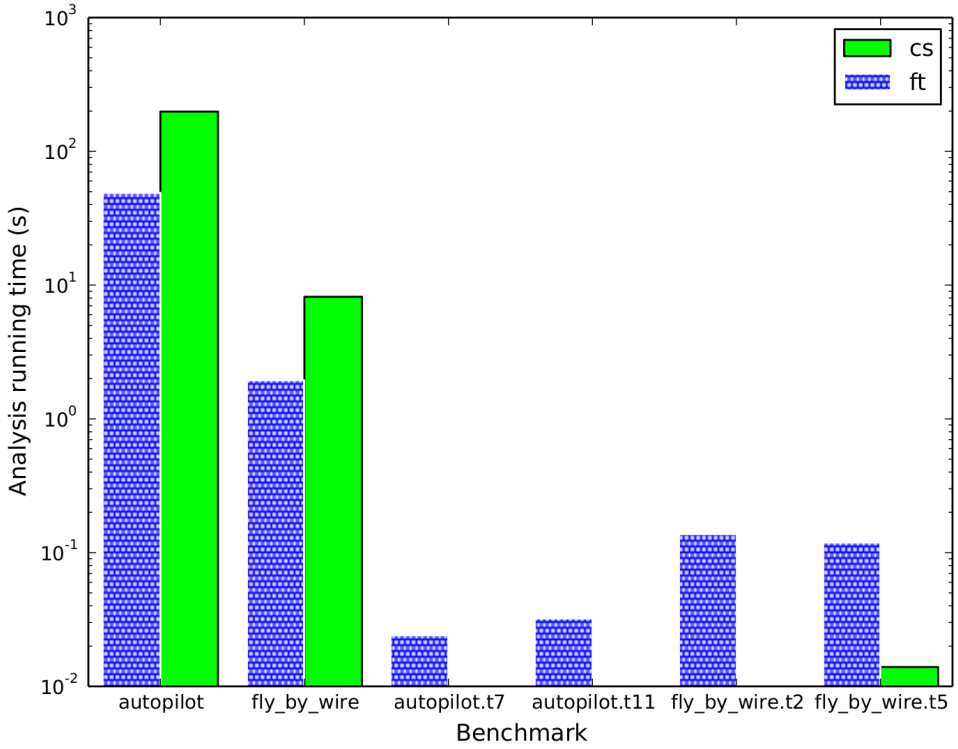
Faster on bigger benchmarks



… but slower on shorter ones.

Evaluation

Faster on bigger benchmarks



… but slower on shorter ones.

Pessimism = Slow

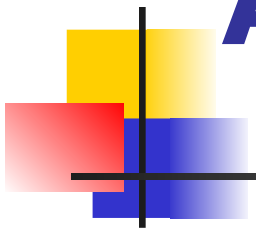
RTS York

THE UNIVERSITY of York

Conclusion

- Full Tree analysis is able to provide a fast and accurate PLRU cache analysis
- Lossy Compression can be useful in deriving an abstraction for use in abstract interpretation
  - (Not just for PLRU caches – see talk tomorrow on Random Replacement Caches)

Any Questions?



RTS York

THE UNIVERSITY of York

PROXIMA