Learning Cache Models by Measurements

Jan Reineke

joint work with Andreas Abel

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Uppsala University

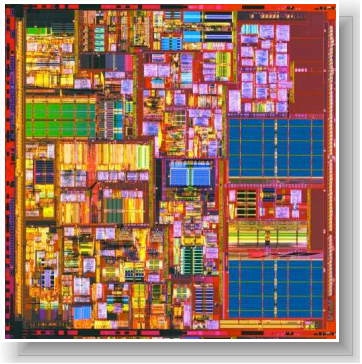
December 20, 2012

The Timing Analysis Problem

```
// Perform the convolution.
for (int i=0; i<10; i++) {
  x[i] = a[i]*b[j-i];
  // Notify listeners.
  notify(x[i]);
}
```

Embedded Software

+



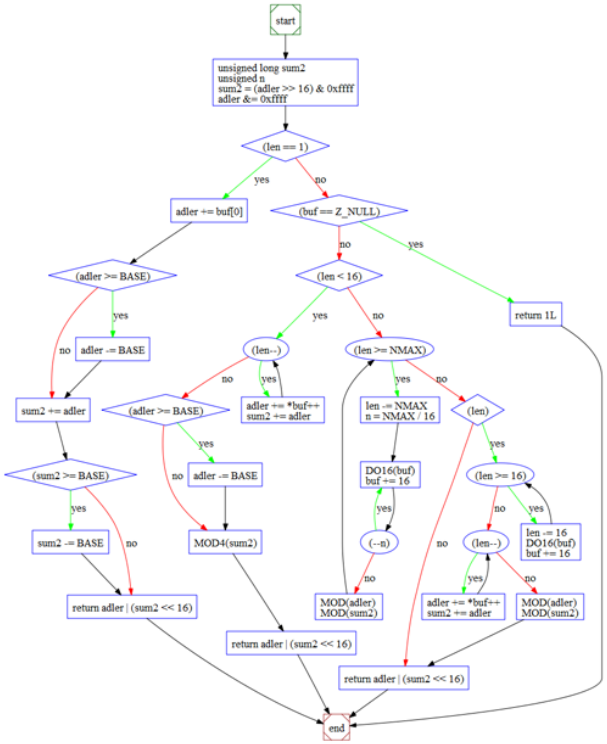
Microarchitecture

?

Timing Requirements

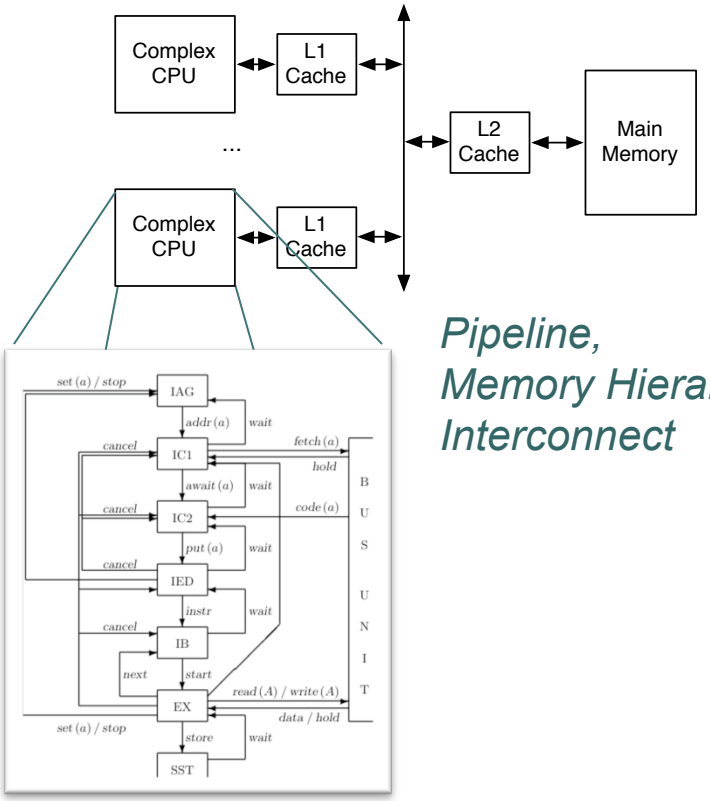
What does the execution time of a program depend on?

Input-dependent control flow



+

Microarchitectural State



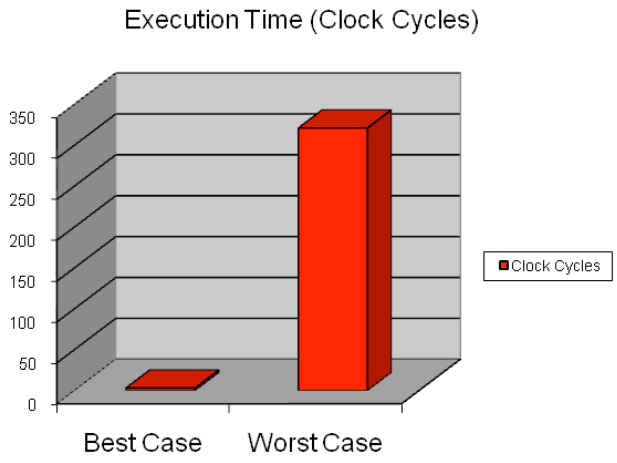
Pipeline, Memory Hierarchy, Interconnect

Example of Influence of Microarchitectural State

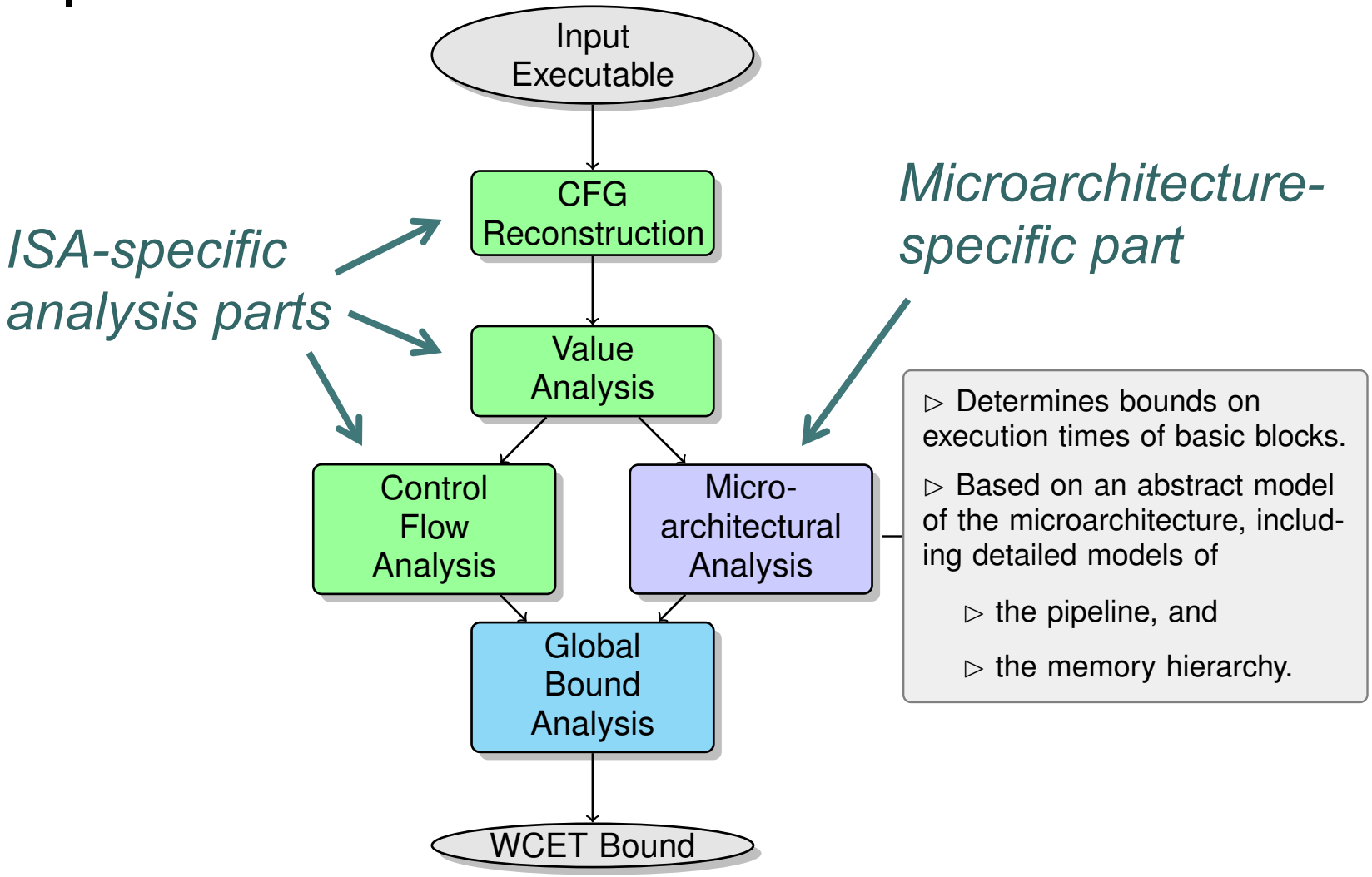
x=a+b; →

```
LOAD   r2, _a
LOAD   r1, _b
ADD    r3,r2,r1
```

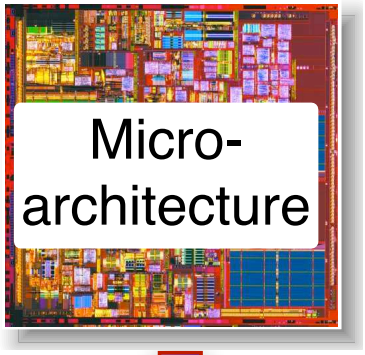
Motorola PowerPC 755



Typical Structure of Timing Analysis Tools



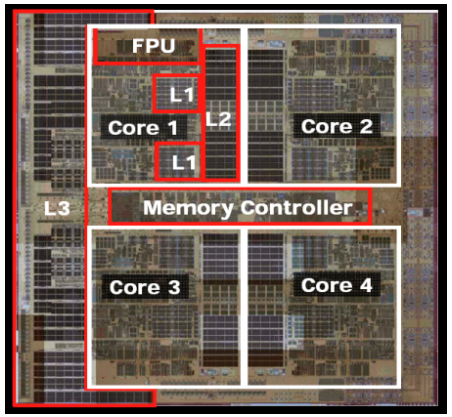
Construction of Timing Models

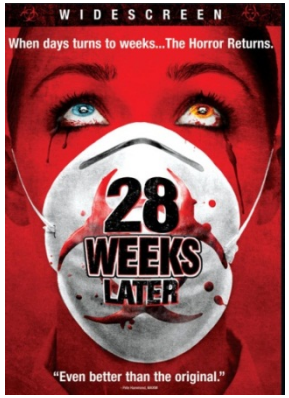


Timing Model

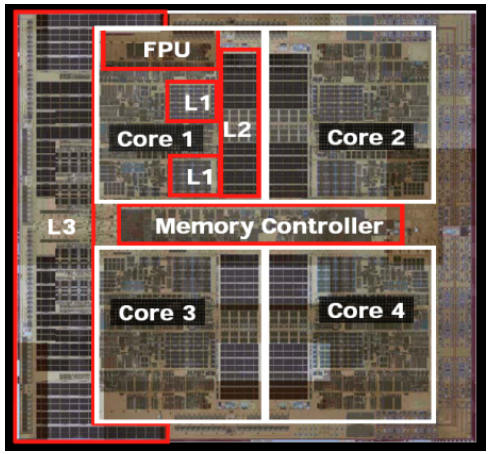
Needs to accurately model all aspects of the microarchitecture that influence execution time.

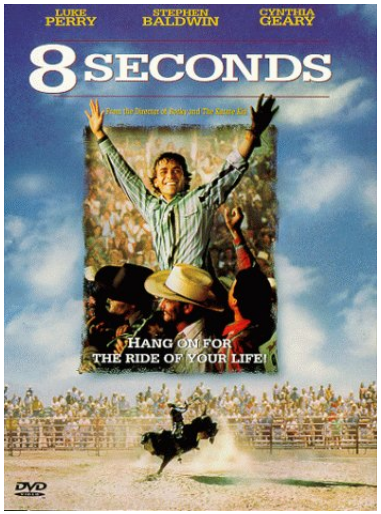
Construction of Timing Models **Today**





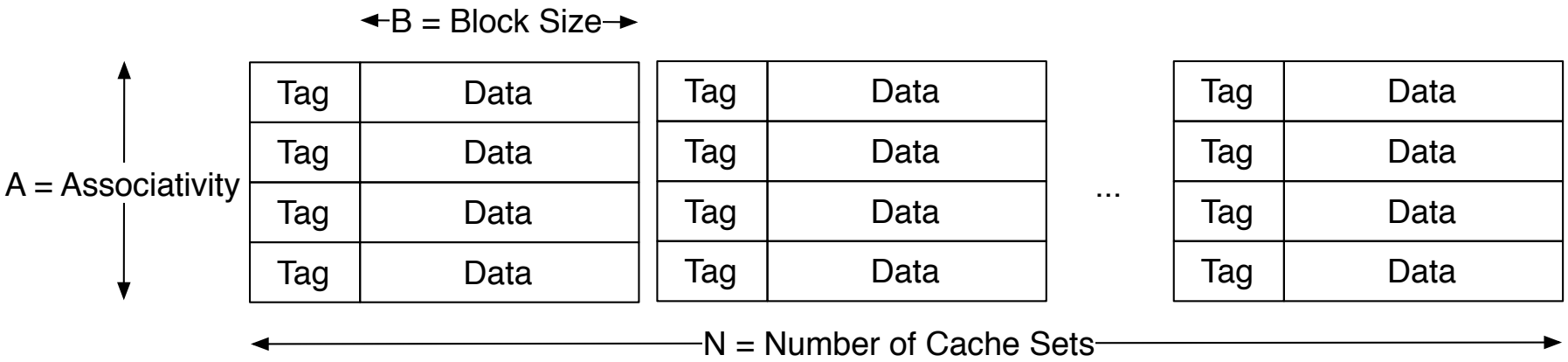
Construction of Timing Models **Tomorrow**





Cache Models

- Cache Model is important part of Timing Model
- Caches have simple interface: loads+stores
- Can be characterized by a few parameters:
  - ABC: associativity, block size, capacity
  - Replacement policy



High-level Approach

- Generate memory access sequences
- Determine number of cache misses on these sequences
  - using performance counters, or
  - by measuring execution times.
- Infer property from measurement results.

Warm-up I: Inferring the Cache Capacity

Basic idea:

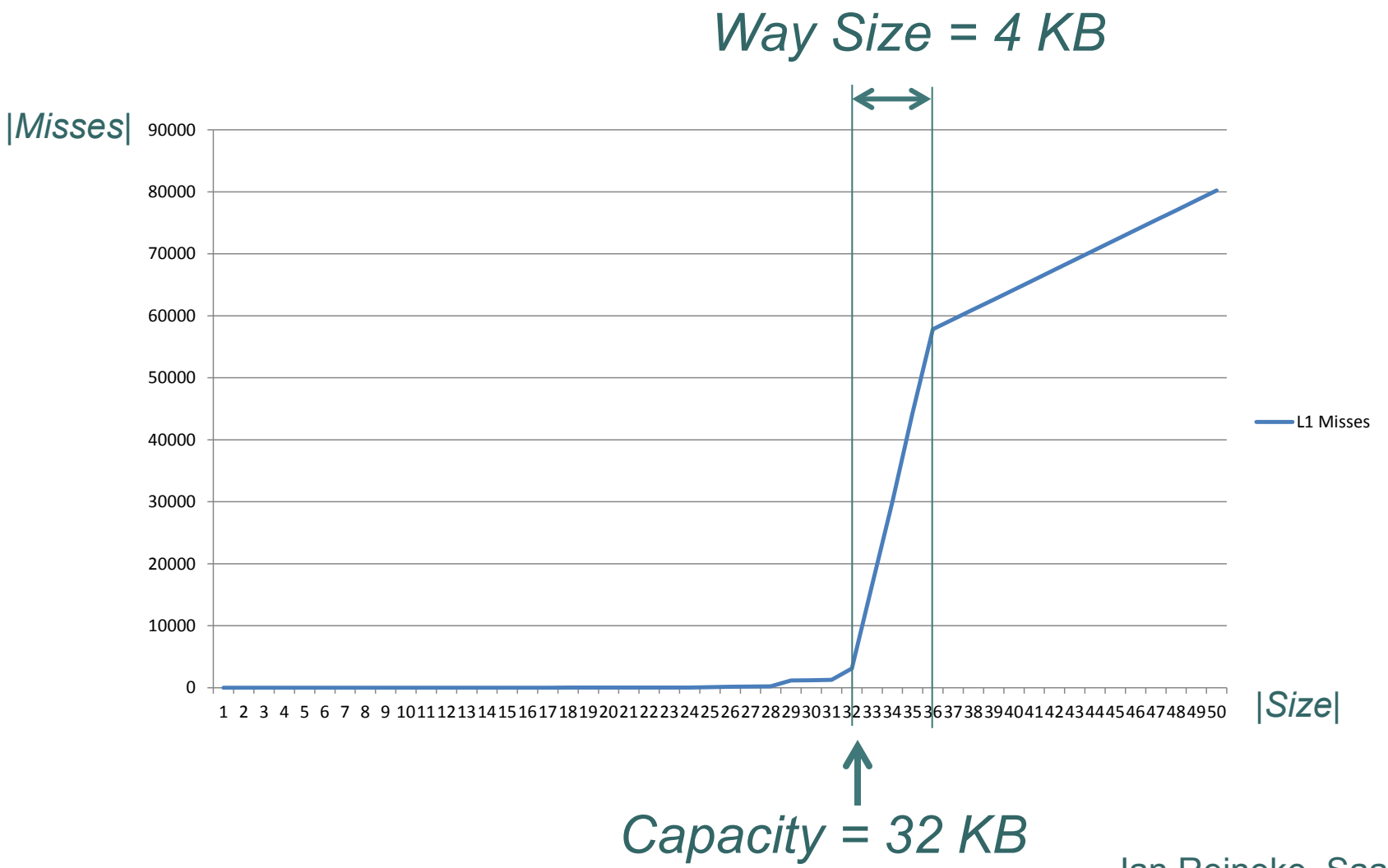
- Access sets of memory blocks A once.
- Then access A again and count cache misses.
- If A fits into the cache, no misses will occur.

Notation: *Preparatory accesses* *Accesses to measure*

$$\mathbf{measure}(\vec{p}, \vec{a})$$

Measure with $\vec{p} = \vec{a} = \langle 0, \ldots, size \rangle$.

Example: Intel Core 2 Duo E6750, L1 Data Cache



Warm-up II: Inferring the Block Size

Given: way size W and associativity A

Wanted: block size B

$$\vec{p} = \langle 0, W, 2 \cdot W, \ldots, A/2 \cdot W \rangle$$
$$\vec{q} = \langle B', B' + W, B' + 2 \cdot W, \ldots, B' + (A/2 + 1) \cdot W \rangle$$

$$\textbf{measure}_C(\vec{p} \cdot \vec{q}, (\vec{p} \cdot \vec{q})^n) = \begin{cases} 0 & \text{if } B' > B \\ n \cdot (A+1) & \text{otherwise} \end{cases}$$

Inferring the Replacement Policy

There are infinitely many conceivable replacement policies…

- ➔ For any set of observations, multiple policies remain possible.
- ➔ Need to make some assumption on possible policies to render inference possible.

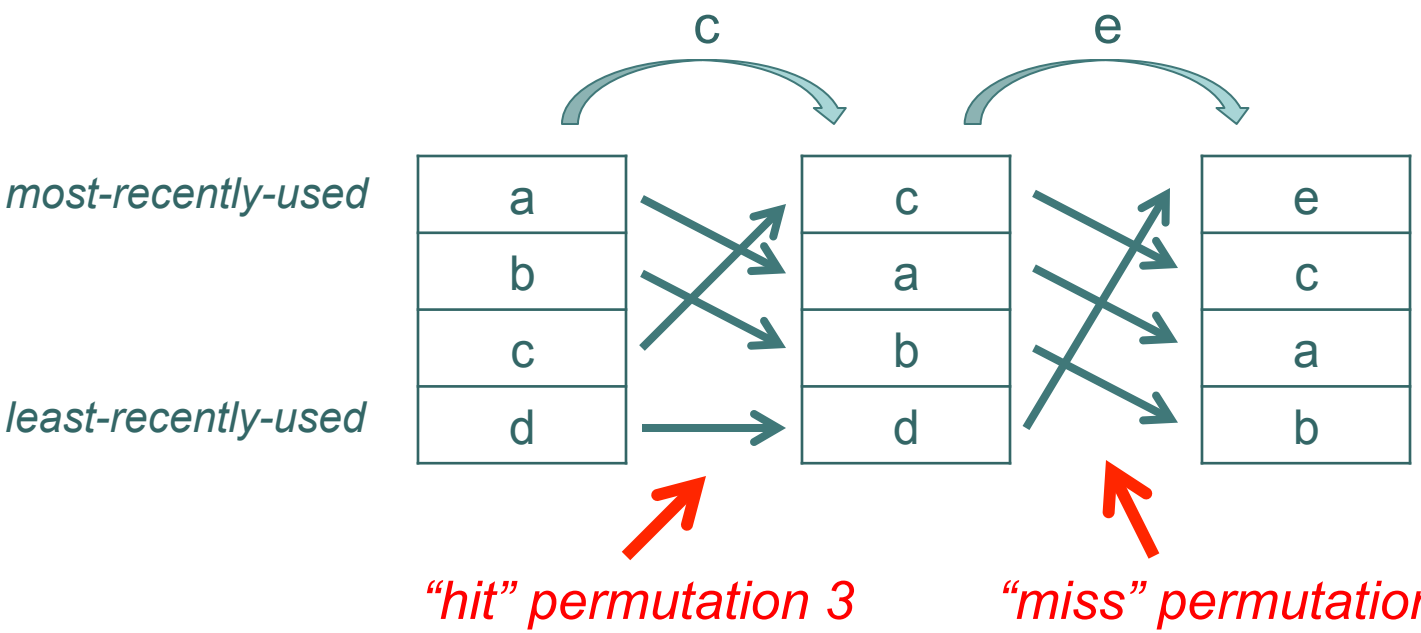
A Class of Replacement Policies: Permutation Policies

- Permutation Policies:
  - Maintain total order on blocks in each cache set
  - Evict the greatest block in the order
  - Update the order based on the position of the accessed block in the order
  - Can be specified by A+1 permutations
    - Associativity-many “hit” permutations
    - one “miss” permutation
- Examples: LRU, FIFO, PLRU, …

Permutation Policy LRU

LRU (least recently used):

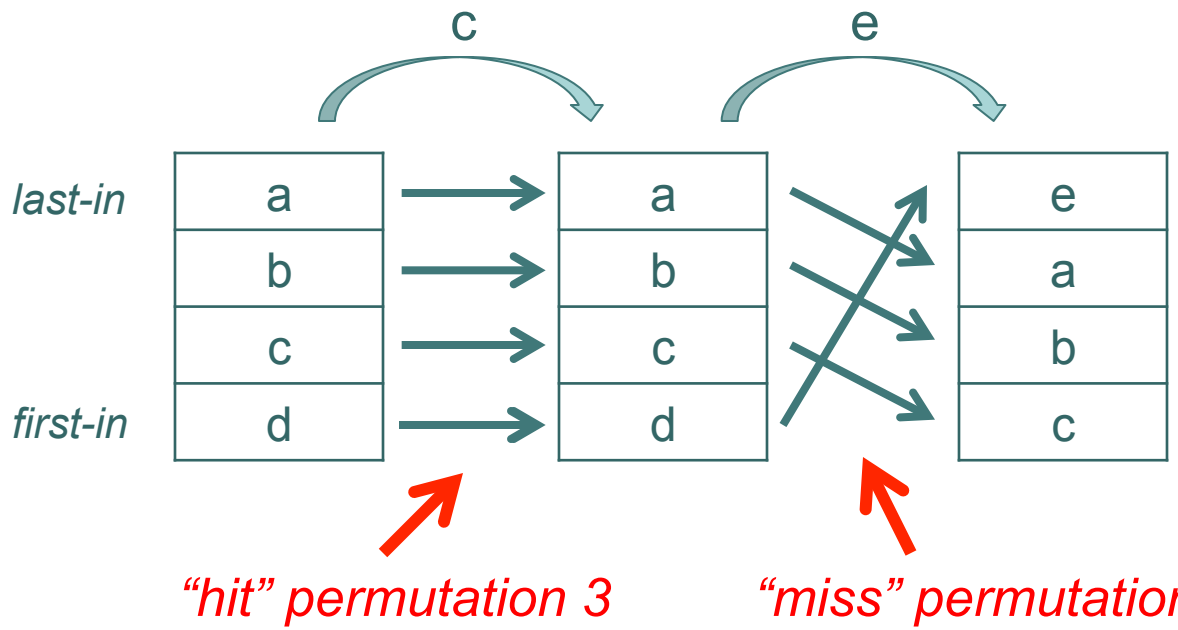
order on blocks based on recency of last access



Permutation Policy FIFO

FIFO (first-in first-out):

order on blocks based on order of cache misses



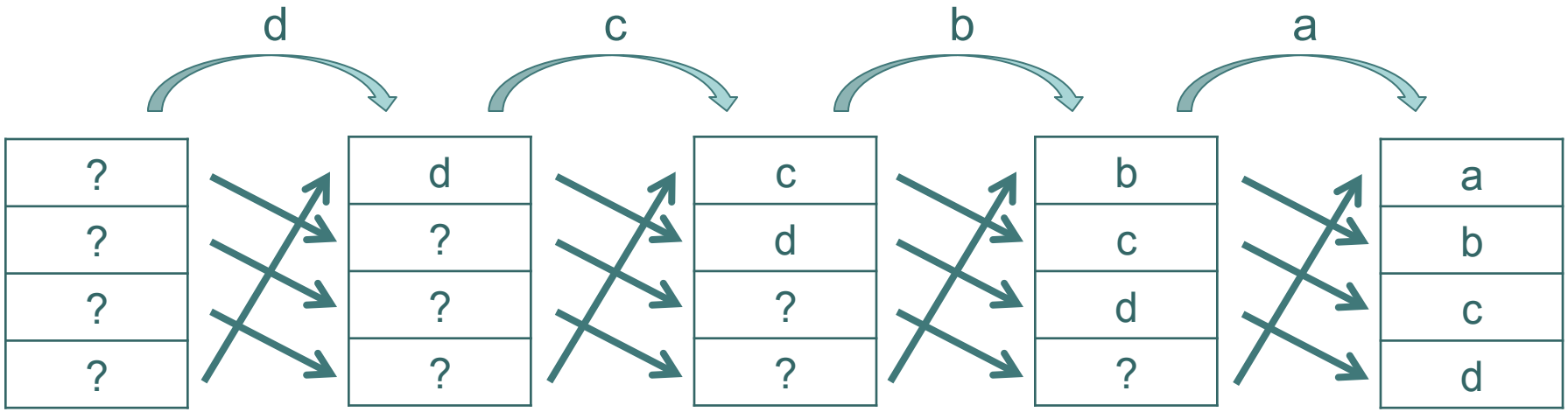
Inferring Permutation Policies

Strategy to infer permutation i:

1) Establish a known cache state **s**
2) Trigger permutation i
3) “Read out” the resulting cache state **s’**.
   Deduce permutation from **s** and **s’**.

1) Establishing a known cache state

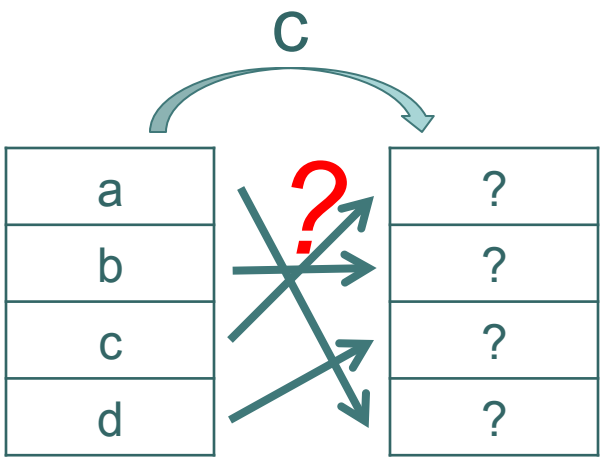
Assume “miss” permutation of FIFO, LRU, PLRU:



2) Triggering permutation i

Simply access i^{th} block in cache state.

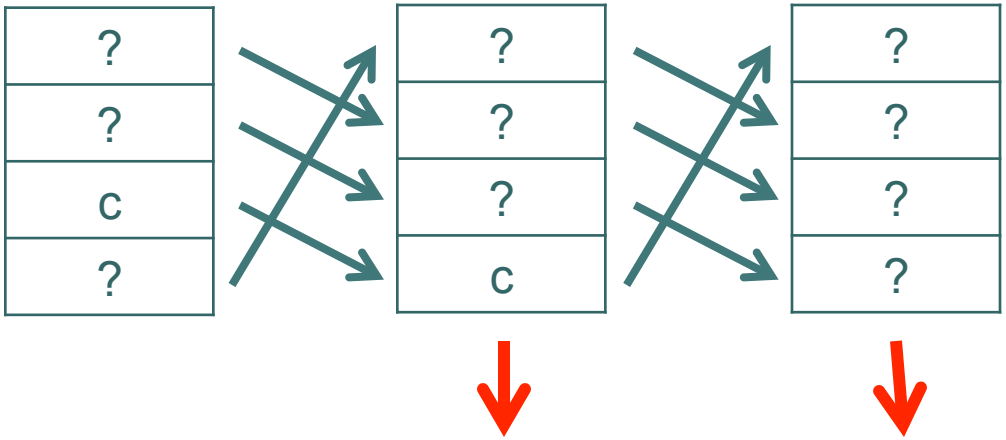
E,g, for i = 3, access c:



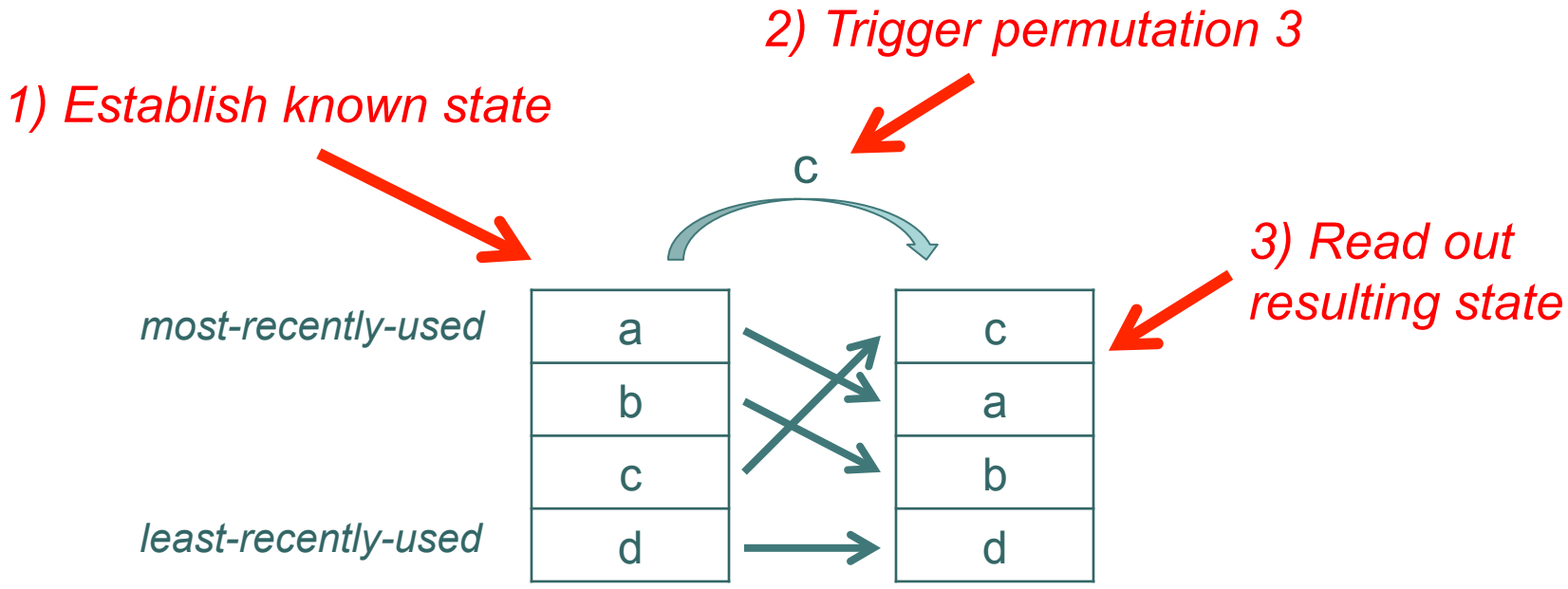
3) Reading out a cache state

Exploit “miss” permutation to determine position of each of the original blocks:

If position is j then A-j+1 misses will evict the block, but A-j misses will not.



Inferring Permutation Policies: Example of LRU, Permutation 3



Implementation Challenges

- Interference
- Prefetching
- Instruction Caches
- Virtual Memory
- L2+L3 Caches:
  - Strictly-inclusive
  - Non-inclusive
  - Exclusive
- Shared Caches:
  - Coherence

Experimental Results

Undocumented variant of LRU

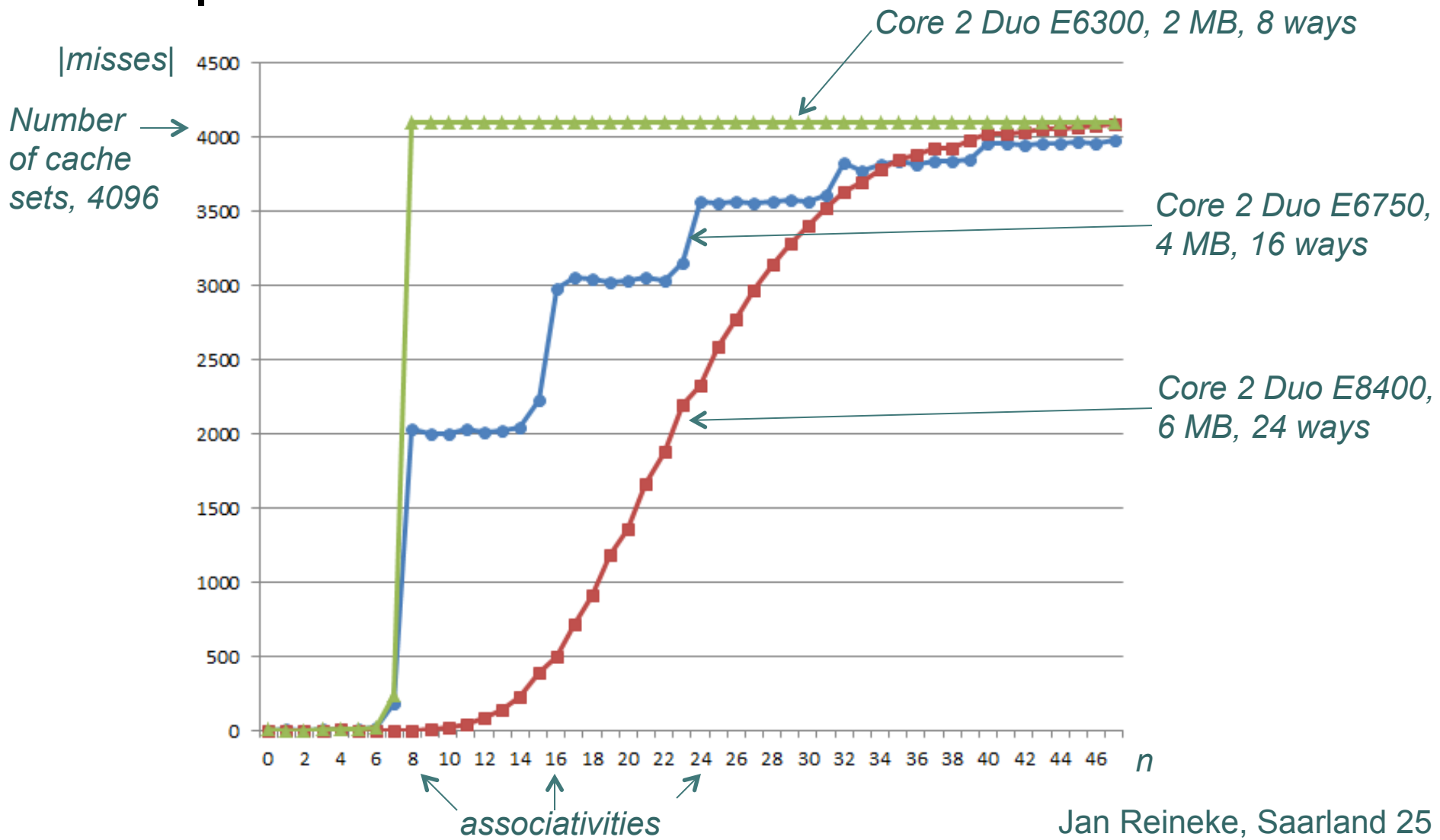
Surprising measurements ;-)

Architecture	L1 Data			L1 Instruction			L2 Unified		
	Size	Assoc.	Policy	Size	Assoc.	Policy	Size	Assoc.	Policy
Intel Atom D525	24kB	6	ATOM	32kB	8	PLRU	512kB	8	PLRU
Intel Pentium 3 900	16kB	4	PLRU	16kB	4	PLRU	256kB	8	PLRU
Intel Core 2 Duo E6300	32kB	8	PLRU	32kB	8	PLRU	2048kB	8	PLRU
Intel Core 2 Duo E6750	32kB	8	PLRU	32kB	8	PLRU	4096kB	16	Section 6.2
Intel Core 2 Duo E8400	32kB	8	PLRU	32kB	8	PLRU	6144kB	24	Section 6.2
Intel Core i5 460M	32kB	8	PLRU	32kB	4	Section 6.2	256kB	8	PLRU
Intel Xeon W3550	32kB	8	PLRU	32kB	4	Section 6.2	256kB	8	PLRU
AMD Athlon 64 X2 4850e	64kB	2	LRU	64kB	2	LRU	512kB	16	Section 6.2
AMD Opteron 8360SE	64kB	2	LRU	64kB	2	LRU	512kB	16	Section 6.2

Nehalem introduced "L0" micro-op buffer

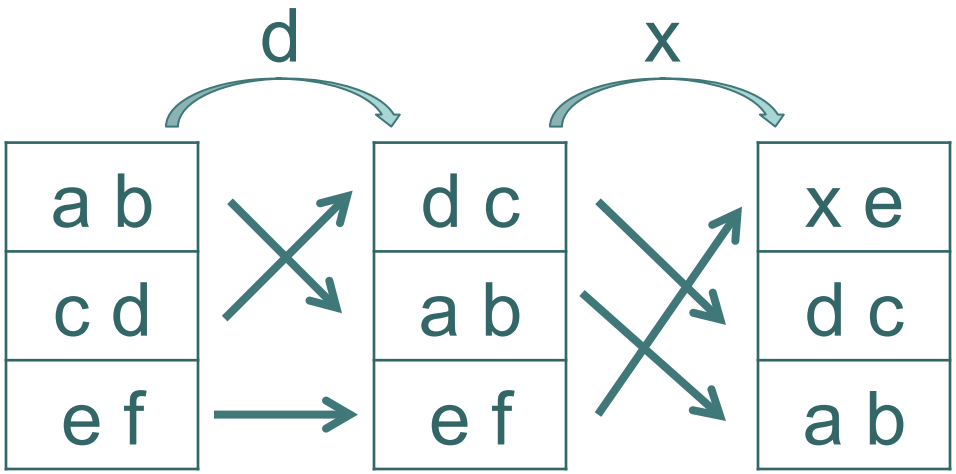
Exclusive hierarchy

L2 Caches on some Core 2 Duos



Replacement Policy of the Intel Atom D525

Discovered to our knowledge undocumented "hierarchical" policy:



ATOM = LRU(3, LRU(2))

PLRU(2k) = LRU(2, PLRU(k))

Conclusions and Future Work

- Inference of cache models I
- More general class of replacement policies, e.g. by *inferring canonical register automata*.
- Shared caches in multicores, coherency protocols, etc.
- Deal with other architectural features: translation lookaside buffers, branch predictors, prefetchers
- Infer *abstractions* instead of "*concrete*" models

Questions?

Measurement-based Modeling of the Cache Replacement Policy
A. Abel and J. Reineke
RTAS, 2013 (to appear).