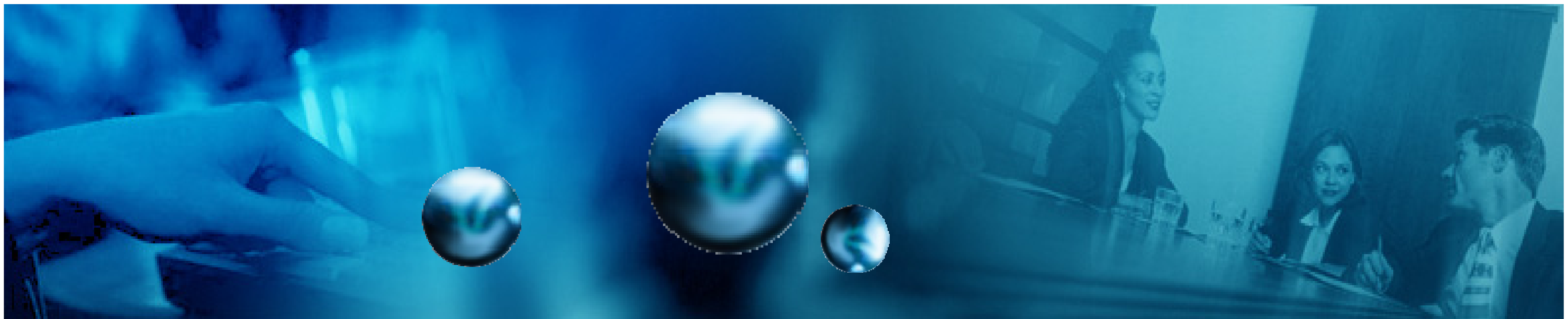
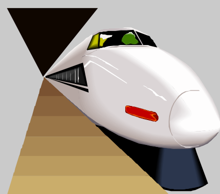
Predictability of Cache Replacement Policies



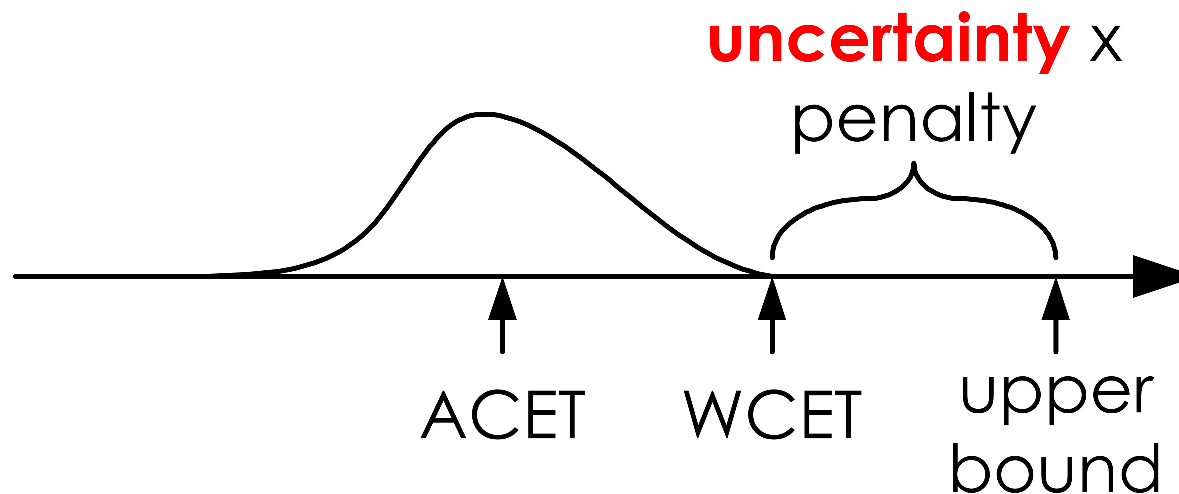
Jan Reineke - Daniel Grund
Christoph Berg - Reinhard Wilhelm
Saarland University



ARTIST2

Predictability in Timing Context

- Hard real-time systems
  - → Strict timing constraints
  - → Need to derive upper bounds on WCET



Cache Analysis

How to statically precompute cache contents:

- Must Analysis:

  For each program point (and calling context), find out which blocks are in the cache

- May Analysis:

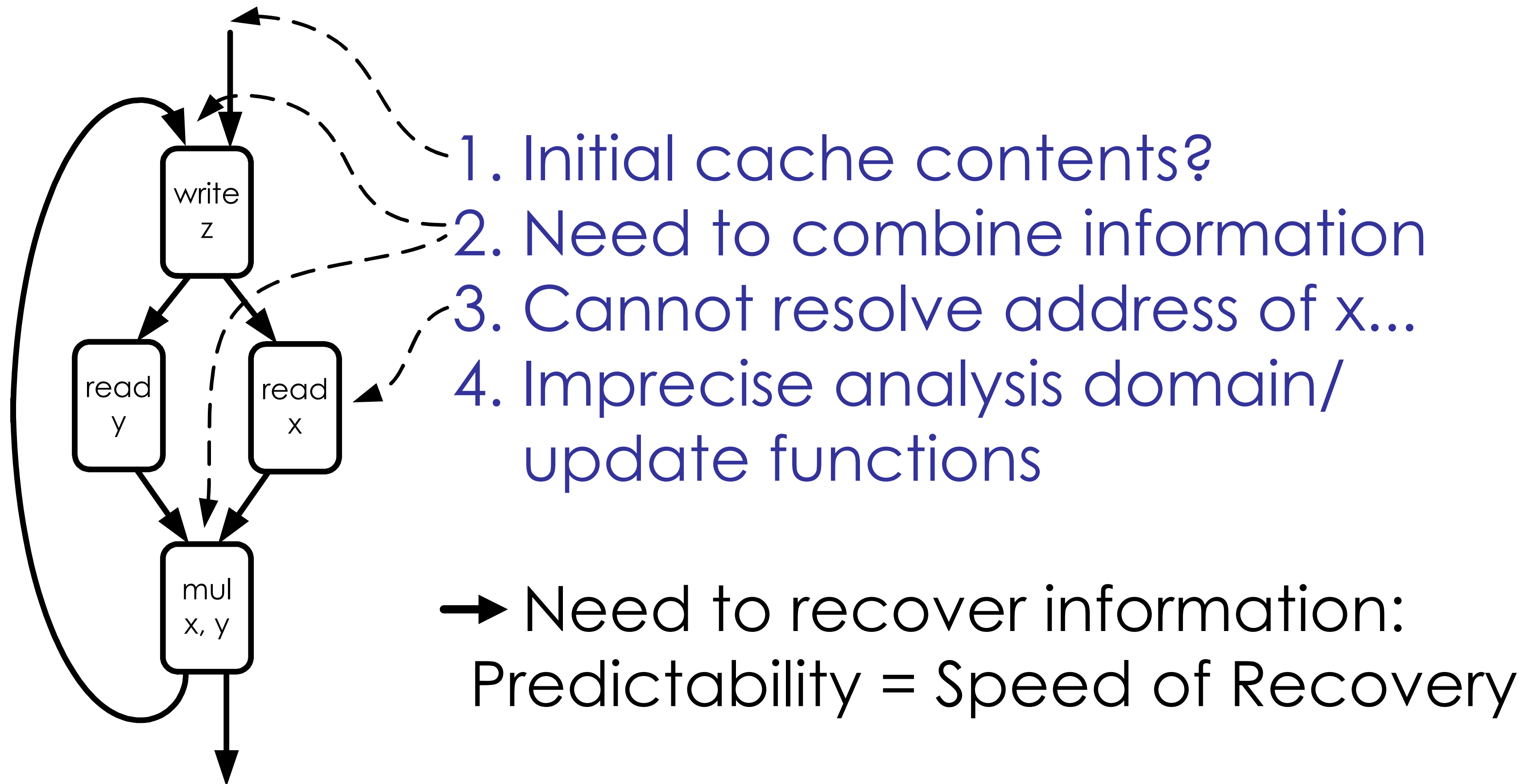
  For each program point (and calling context), find out which blocks may be in the cache

  Complement says what is not in the cache

Must-Cache and May-Cache- Information

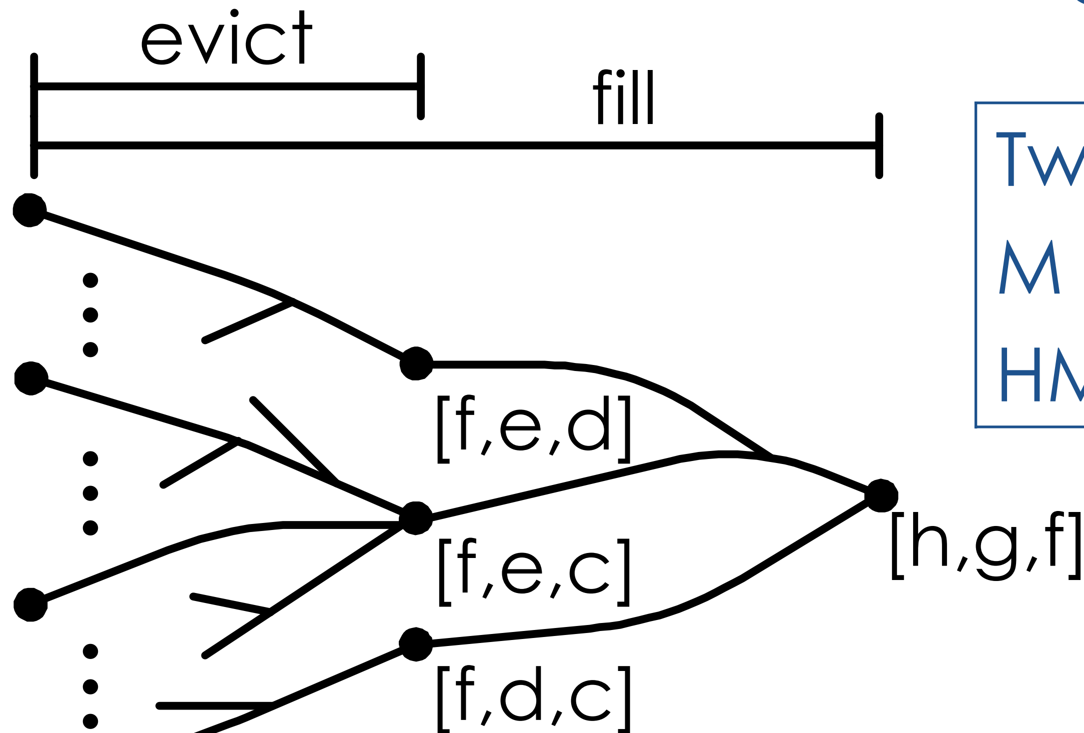
- Must Analysis determines safe information about cache hits
  Each predicted cache hit reduces the upper bound
- May Analysis determines safe information about cache misses
  Each predicted cache miss increases the lower bound

Uncertainty in Cache Analysis



Metrics of Predictability:

evict & fill



Two Variants:

M = Misses Only

HM

← pairwise different

Meaning of evict/fill - I

- Evict: *may*-information:
  - What is definitely not in the cache?
  - Safe information about Cache Misses
- Fill: *must*-information:
  - What is definitely in the cache?
  - Safe information about Cache Hits

Meaning of evict/fill - II

Metrics are independent of analyses:

→ evict/fill bound the precision of any static analysis!

→ Allows to analyze an analysis:

Is it as precise as it gets w.r.t. the metrics?

Replacement Policies

- LRU – Least Recently Used

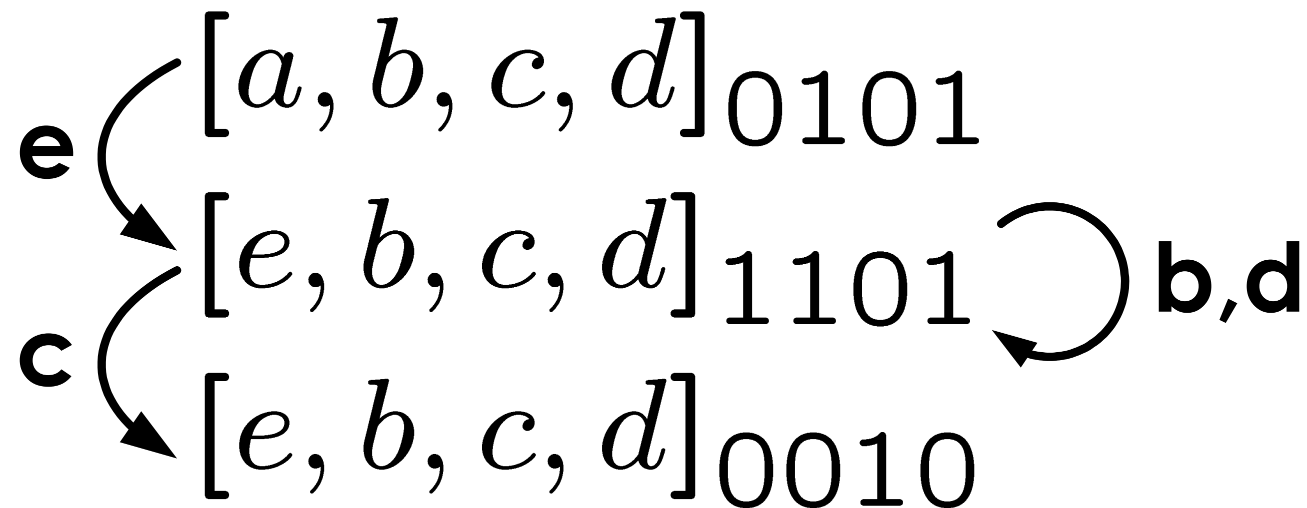
  Intel Pentium, MIPS 24K/34K
- FIFO – First-In First-Out (Round-robin)

  Intel XScale, ARM9, ARM11
- PLRU – Pseudo-LRU

  Intel Pentium II+III+IV, PowerPC 75x
- MRU – Most Recently Used

MRU - Most Recently Used

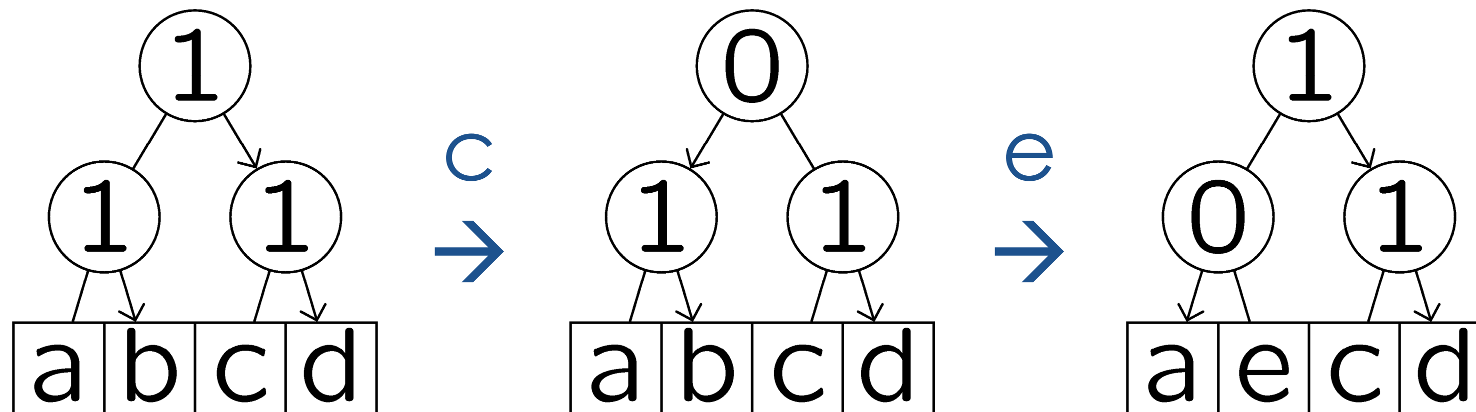
MRU-bit records whether line was recently used



Problem: never stabilizes

Pseudo-LRU

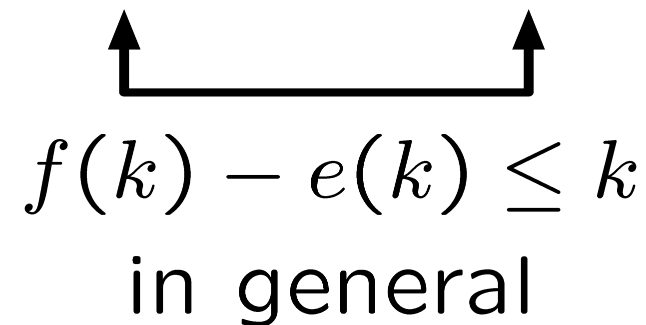
Tree maintains order:



Problem: accesses „rejuvenate“ neighborhood (d protected by c)

Results: tight bounds

Policy	$e_M(k)$	$f_M(k)$	$e_{HM}(k)$	$f_{HM}(k)$
LRU	k	k	k	k
FIFO	k	k	$2k-1$	$3k-1$
MRU	$2k-2$	$\infty/2k-4^{\S}$	$2k-2$	$\infty/3k-4^{\S}$
PLRU	$\left\{\begin{array}{c} 2k-\sqrt{2k} \\ 2k-\frac{3}{2}\sqrt{k} \end{array}\right\}$	$2k-1$	$\frac{k}{2}\log_2 k+1$	$\frac{k}{2}\log_2 k+k-1$



Generic examples prove tightness.

Results: instances for k=4,8

	$k=4$				$k=8$			
Policy	e_M	f_M	e_{HM}	f_{HM}	e_M	f_M	e_{HM}	f_{HM}
LRU	4	4	4	4	8	8	8	8
FIFO	4	4	7	11	8	8	15	23
MRU	6	$\infty/4$	6	$\infty/8$	14	$\infty/12$	14	$\infty/20$
PLRU	5	7	5	7	12	15	13	19

Question:

8-way PLRU cache, 256 sets,

straight-line code, 4 instructions per line

How many instructions to get *may*-information?

Future Work I

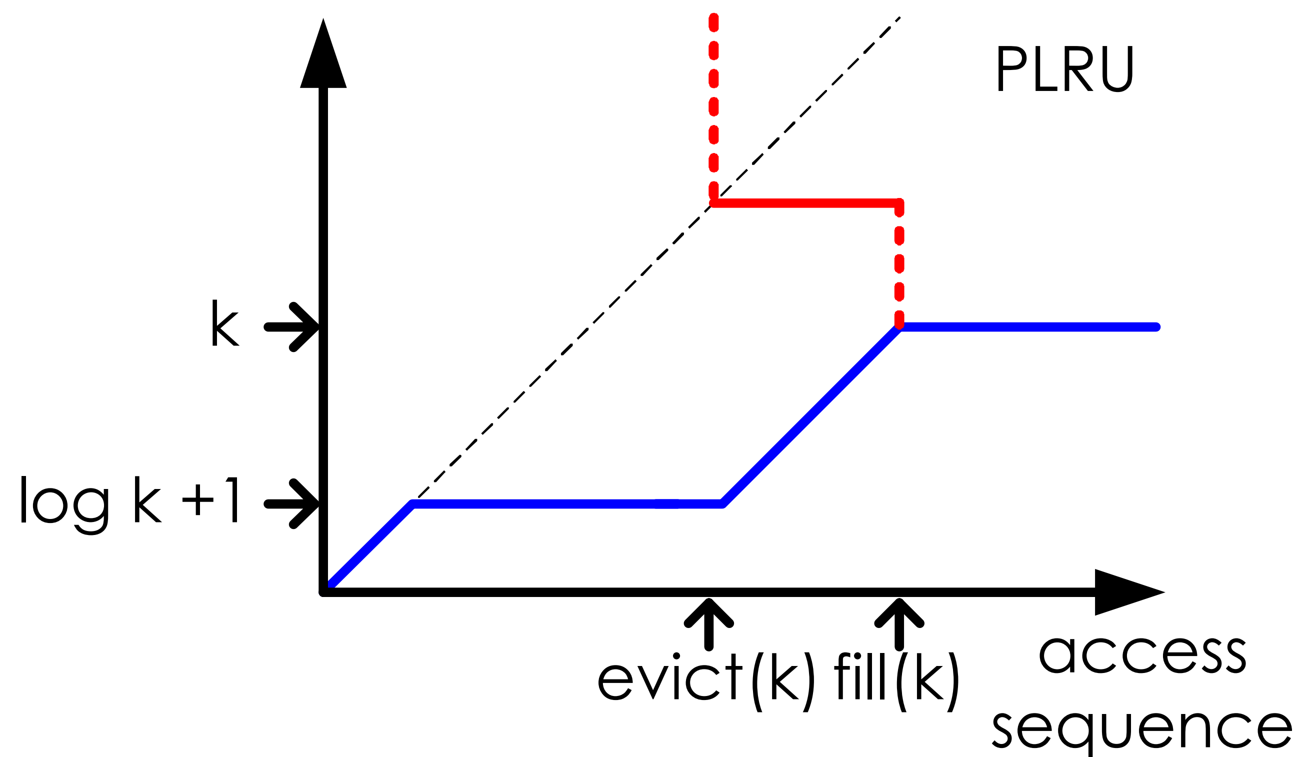
Beyond evict/fill:

- Evict/fill assume complete uncertainty
- What if there is only partial uncertainty?
- Other useful metrics?

Future Work I

Beyond evict/fill:

- Evolution of *may/must*-information:



Future Work II/III

Analyze cache analyses:

- Do they ever recover „perfect“ *may/must*-information?
- If so, within evict/fill accesses?

Develop precise and efficient analyses:

- Idea: Remember last *evict* accesses
- Problem: Accesses are not pairwise different in practice (... no cache hits ;-))

Future Work III

→ Simplify access sequences :

– <w x y y z> → <w x y z> !

– <x z y z> → <x y z> ?

Works for LRU, not for other policies in general?

Yields known LRU analysis after additional abstraction.