

Timing-Predictability of Cache Replacement Policies



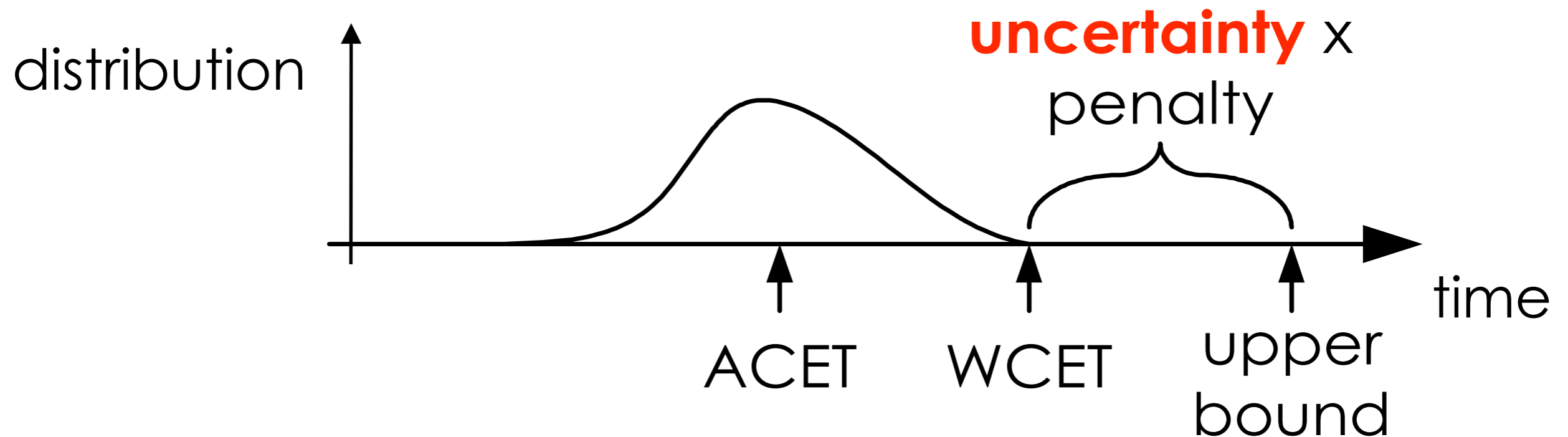
Jan Reineke - Daniel Grund
Christoph Berg - Reinhard Wilhelm

Dagstuhl, March 6th, 2007



Predictability in Timing Context

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

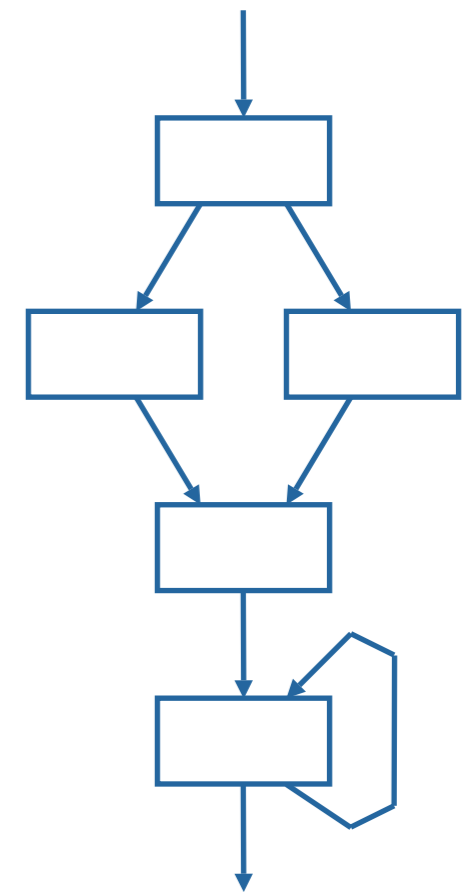


{W | A}CET = {Worst | Average}-Case Execution Time



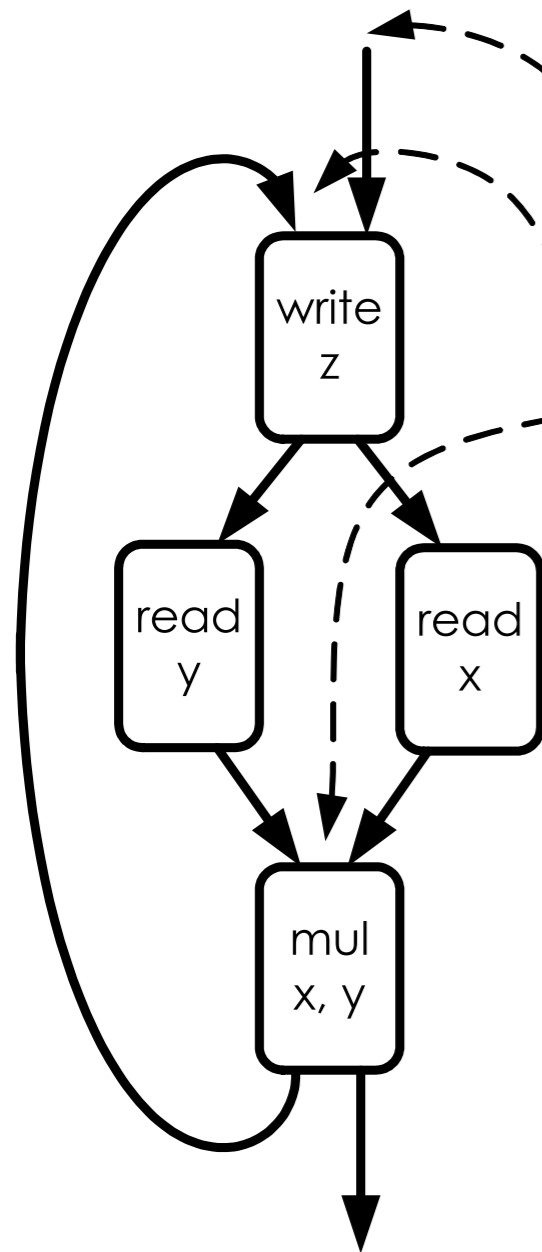
Cache Analysis: Goals

- Goal: classify accesses as hits or misses
 - May-Information:
For each program point (and calling context):
Which lines **may** be in the cache?
→ Classify misses
 - Must-Information:
For each program point (and calling context):
Which lines **must** be in the cache?
→ Classify hits
- Cache Miss Penalty: ~100 cycles





Uncertainty in Cache Analysis



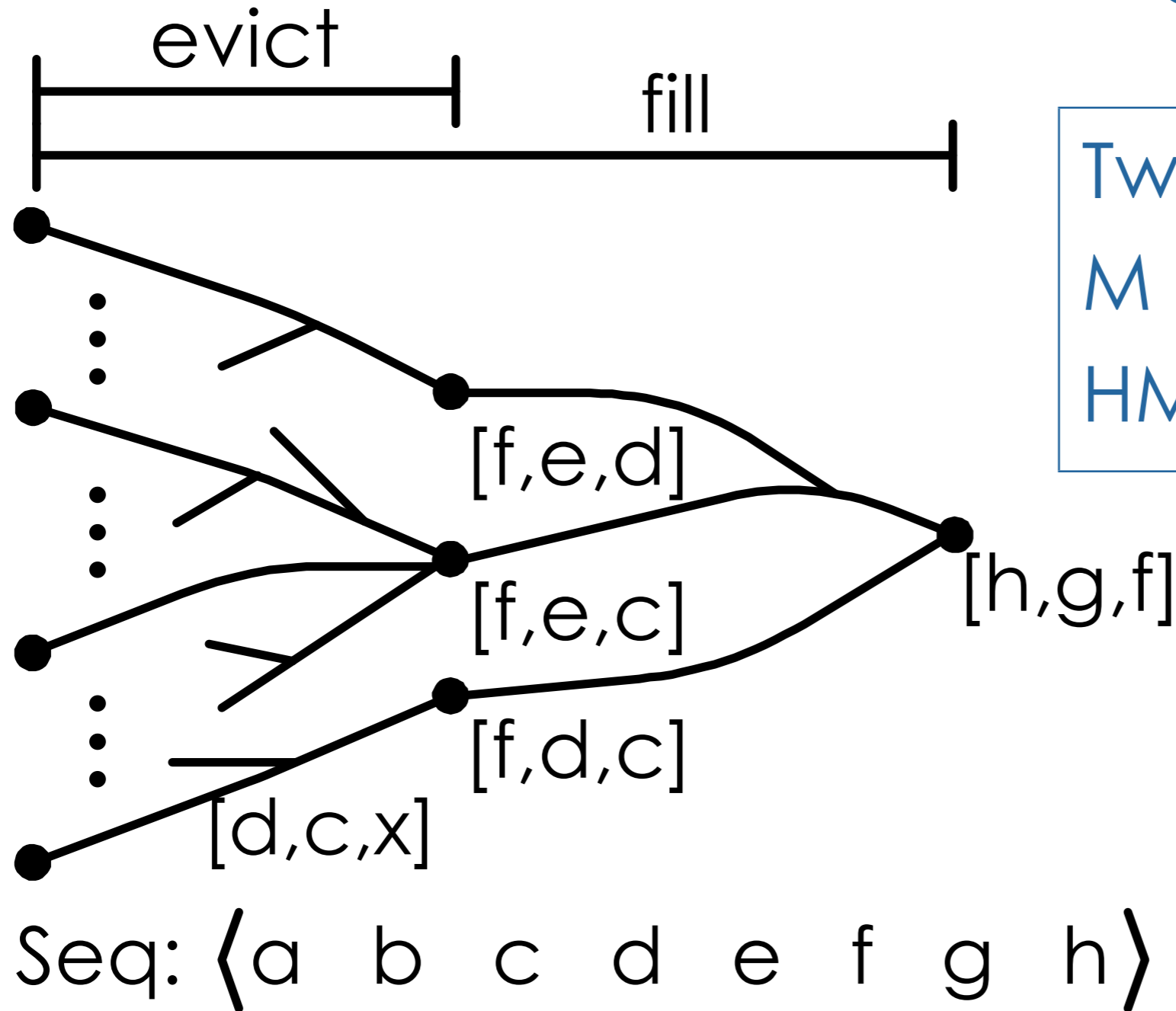
1. Initial cache contents?
2. Need to combine information
3. Cannot resolve address of x...

→ Need to recover information:
Predictability = Speed of Recovery



Metrics of Predictability:

evict & fill



Two Variants:
 M = Misses Only
 HM = Hits & Misses



Meaning of evict/fill - I

- Evict:
 - When do we gain any *may*-information?
 - Some information about Cache Misses
- Fill:
 - When do we gain precise *may*- and *must*-information?
 - Can precisely classify Cache Hits and Misses



Meaning of evict/fill - II

Metrics are independent of analyses:

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

→ Allows to analyze an analysis:

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



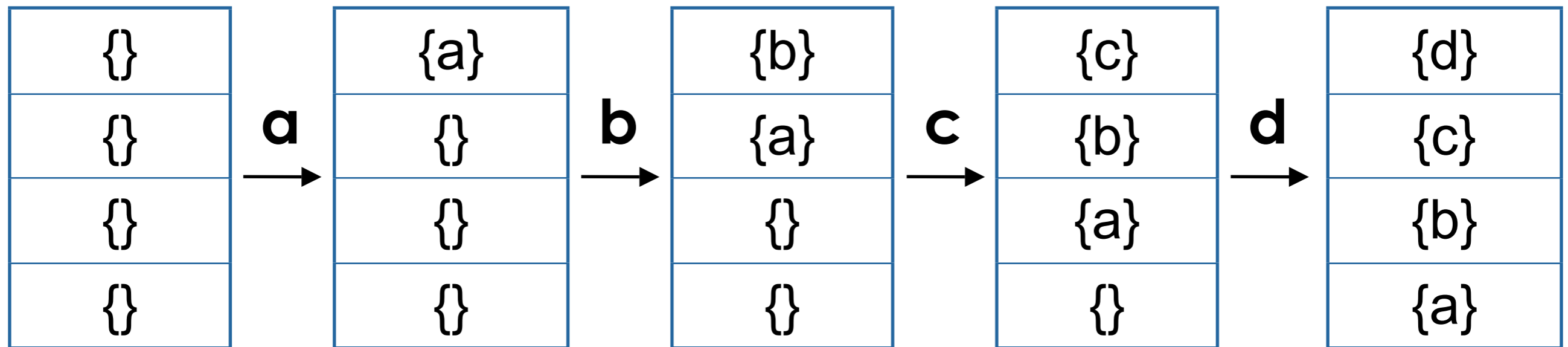
Replacement Policies

- 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

LRU - Least Recently Used

LRU is the simplest case:

After $i \leq k$ (associativity) we have exact must-information for i elements.

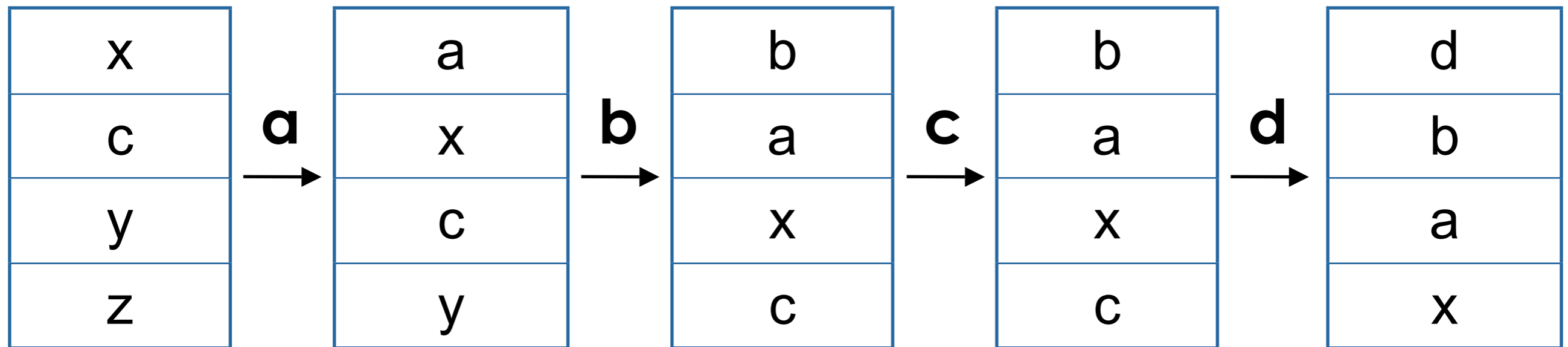


$$\rightarrow \text{evict}(k) = \text{fill}(k) = k$$



FIFO – First-In First-Out

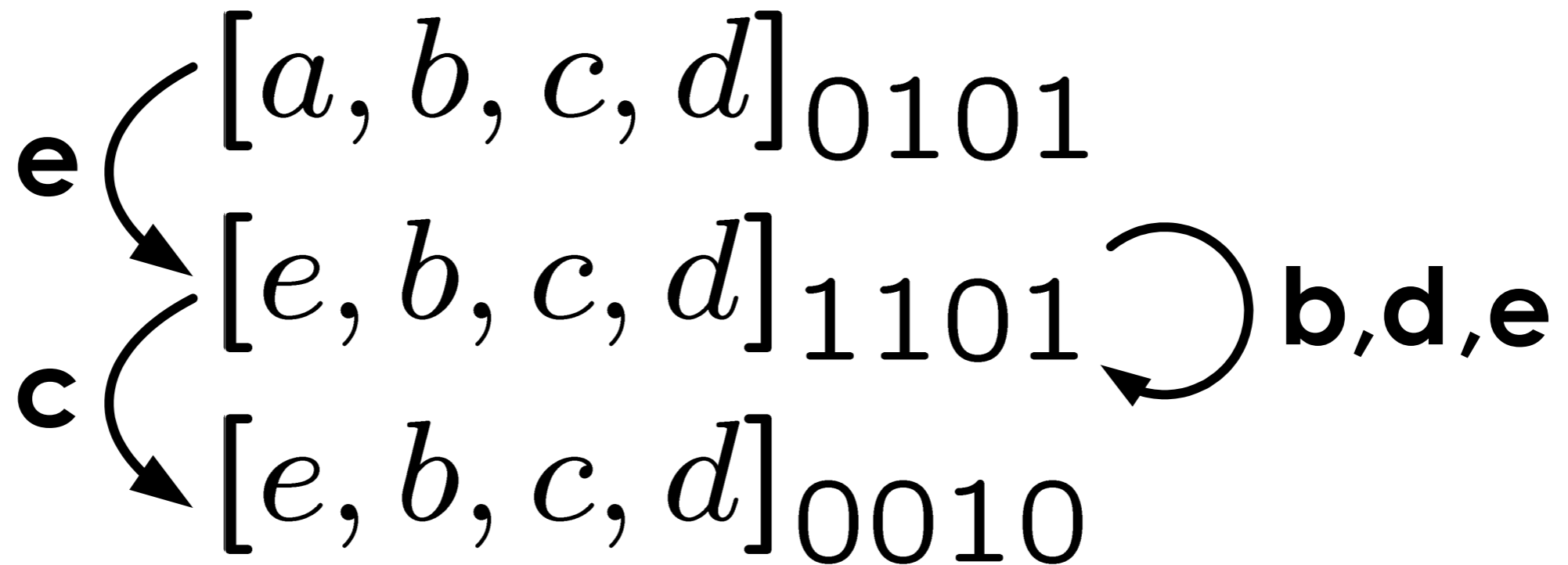
- Like LRU in the miss-case
- But hits do **not** change the state





MRU - Most Recently Used

MRU-bit records whether line was recently used

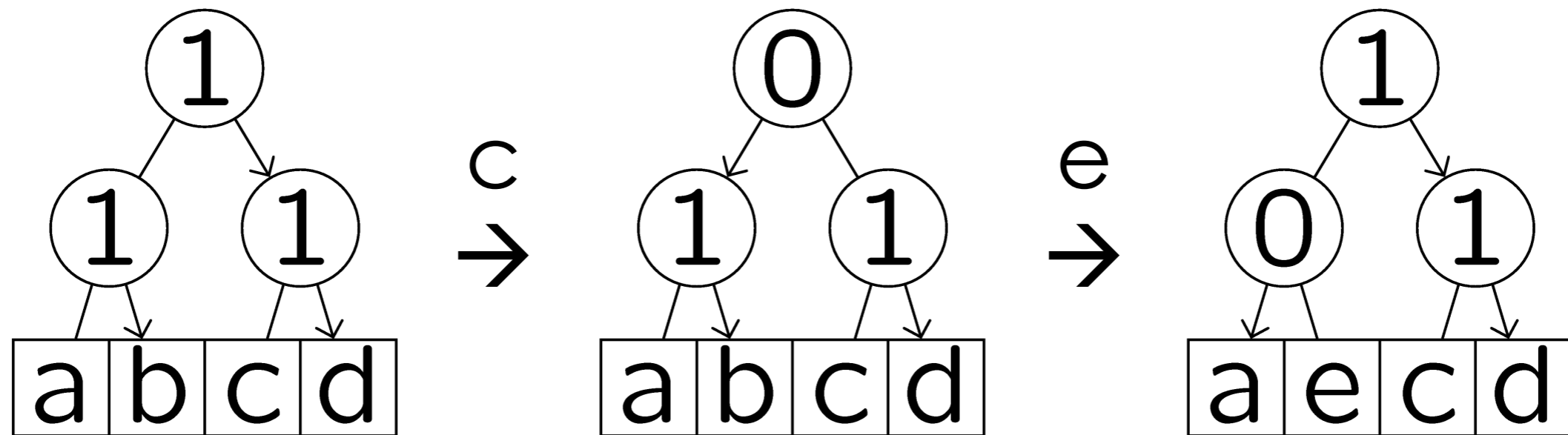


Problem: never stabilizes

**c „safe“
for 5 acc.**



Tree maintains order:



Problem: accesses „rejuvenate“ neighborhood



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	$\left\{ \begin{array}{l} 2k - \sqrt{2k} \\ 2k - \frac{3}{2}\sqrt{k} \end{array} \right\}$	$2k - 1$	$\frac{k}{2} \log_2 k + 1$	$\frac{k}{2} \log_2 k + k - 1$

Generic examples prove tightness.



Results: instances for $k=4,8$

Policy	$k = 4$				$k = 8$			
	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, 4 instructions per line

Assume equal distribution of instructions over 256 sets:

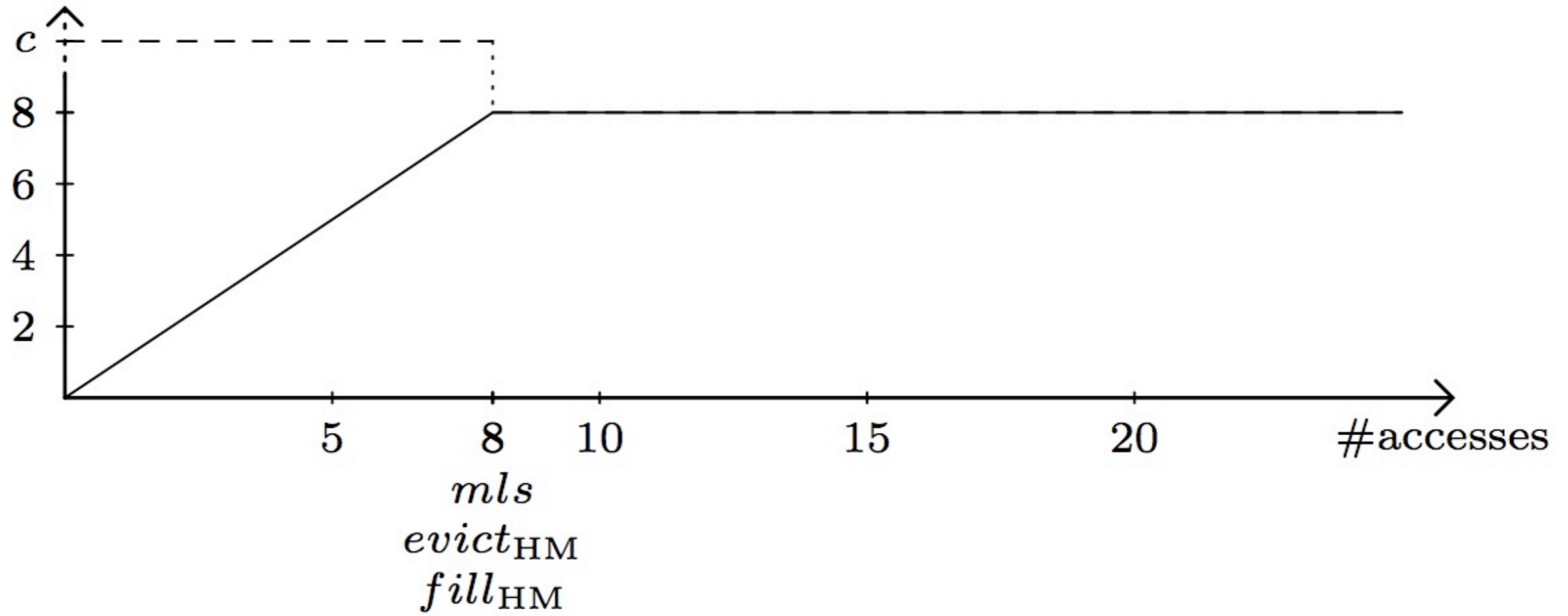
How long a straight-line code sequence is needed to obtain precise must-information?



Evolution of *may/must*-information

8-way LRU:

must/may
information

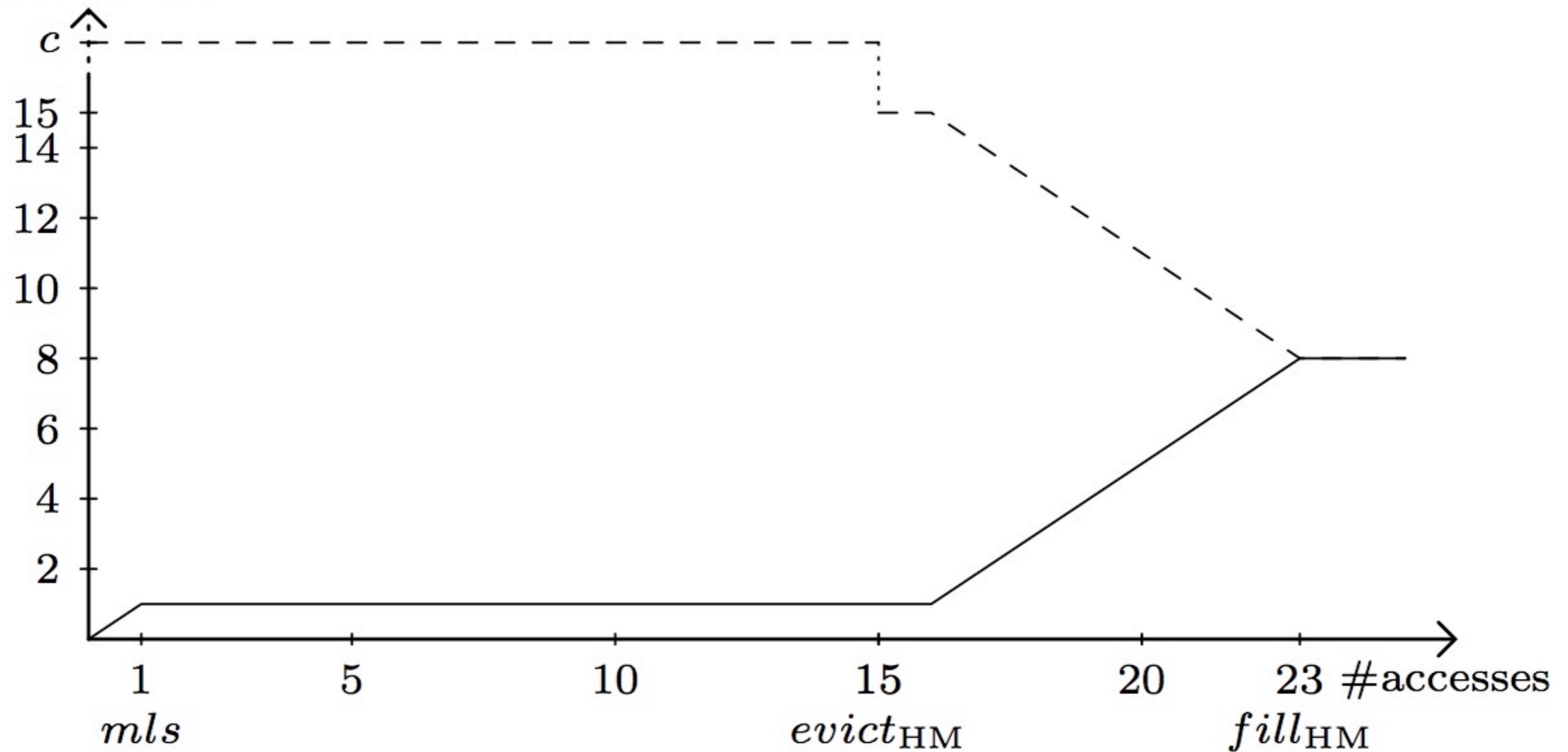




Evolution of may/must-information

8-way FIFO:

must/may
information



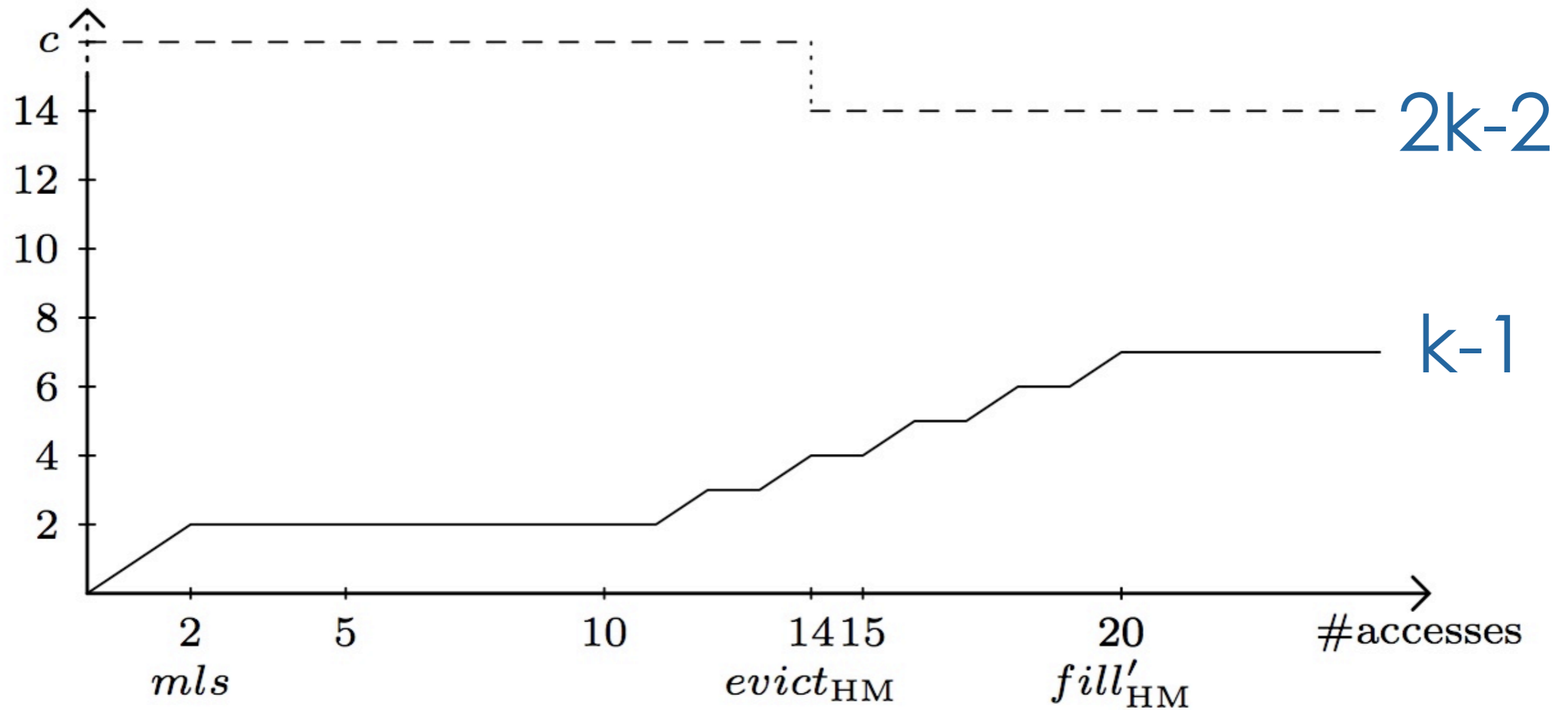
mls = minimal live-span



Evolution of may/must-information

8-way MRU:

must/may
information

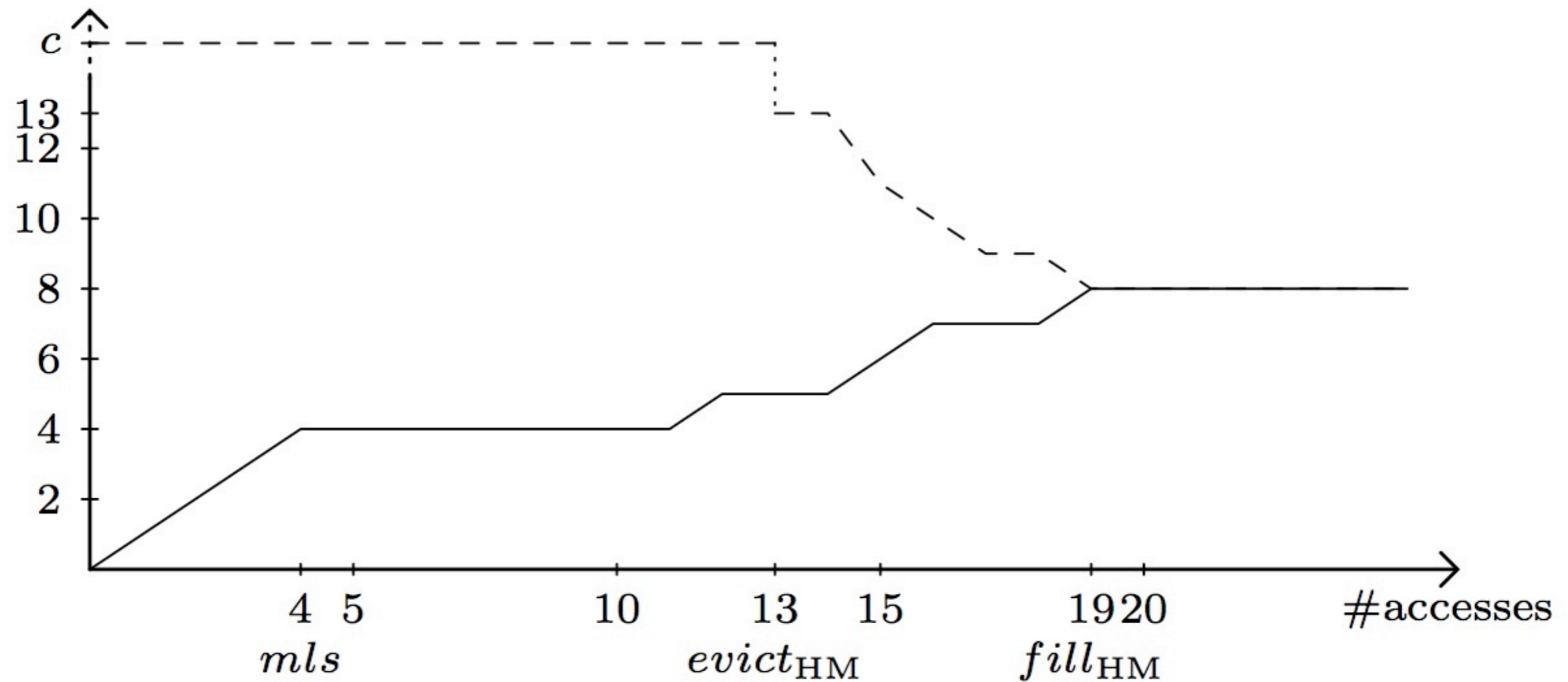




Evolution of may/must-information

8-way PLRU:

must/may
information





Conclusion

- First analytical results on the predictability of cache replacement policies
- LRU is perfect in terms of our predictability metrics
- FIFO and MRU are particularly bad, especially considering the evolution of must-information



Find **new** cache replacement policies

- Predictable
- Cheap to implement
- High (average-case) performance



- Earlier version: AVACS Tech. Report 11 (www.avacs.org)
- Current version under review at RTSJ



The End

