Design and Analysis of Time-Critical Systems

Cache Analysis and Predictability

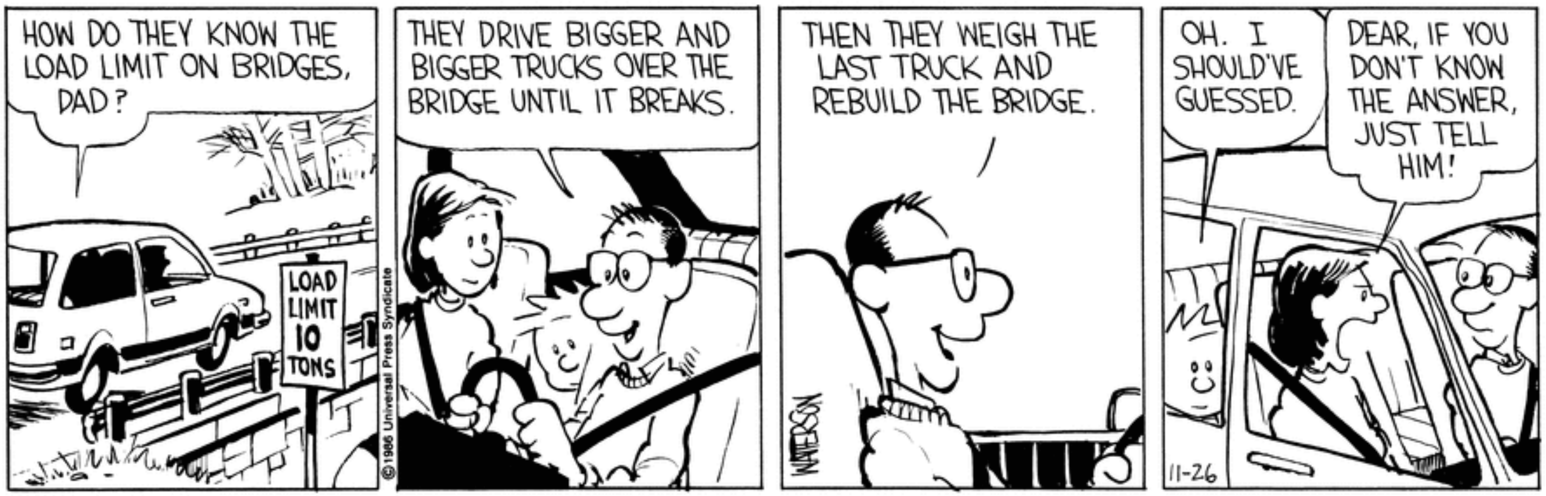
Jan Reineke @ SAARLAND UNIVERSITY

COMPUTER SCIENCE

ACACES Summer School 2017

Fiuggi, Italy

An Analogy



Calvin and Hobbes *by Bill Watterson*
November 26, 1986

Can We Predict Software Behavior?

- Security:
  - Does this app leak my private data?
  - Are buffer overflow attacks possible?
- Verification:
  - Can this program crash?
  - Does it behave according to specification?
- Timing Analysis:
  - What is the maximum runtime of this program?
  - Important Subproblems:
    - Which address will be accessed?
    - **Will this memory access result in a cache hit?**

Prediction is Impossible!

Alan Turing:
„The halting problem
is undecidable."

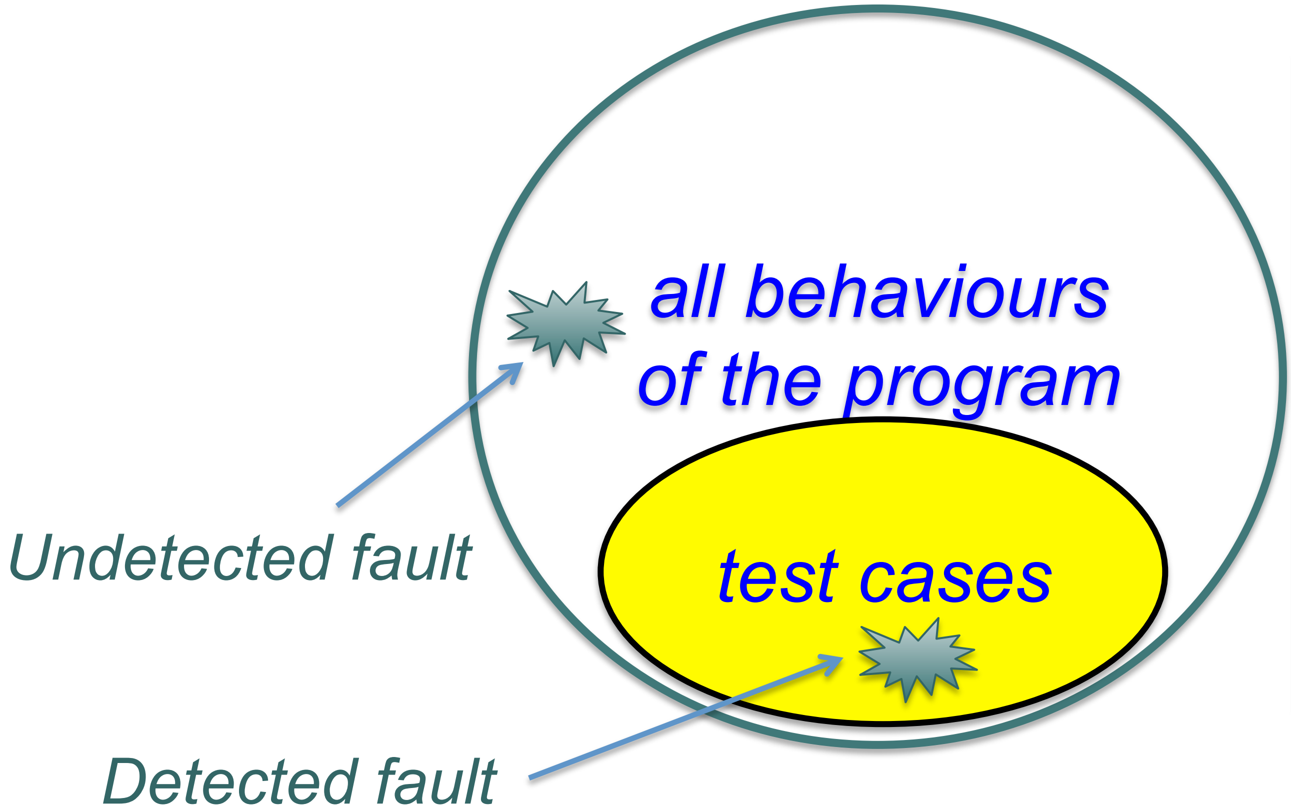
Alan Turing

Henry Gordon Rice:
„Any non-trivial property of programs
is undecidable."

Approximative Solutions

Underapproximation: Testing

Test the program on a finite subset of its inputs:

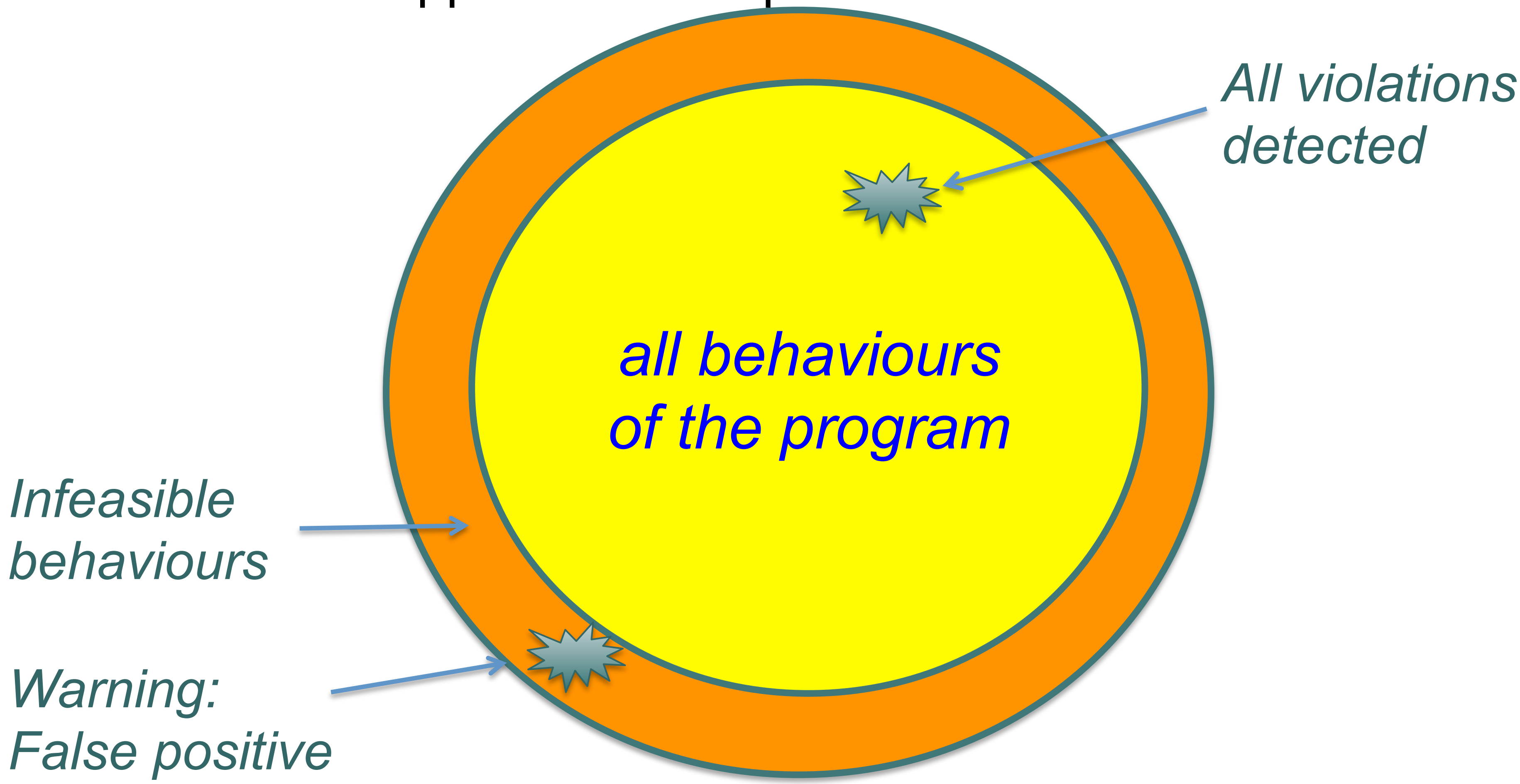


"Program testing can be used to show the presence of bugs, but never to show their absence!"

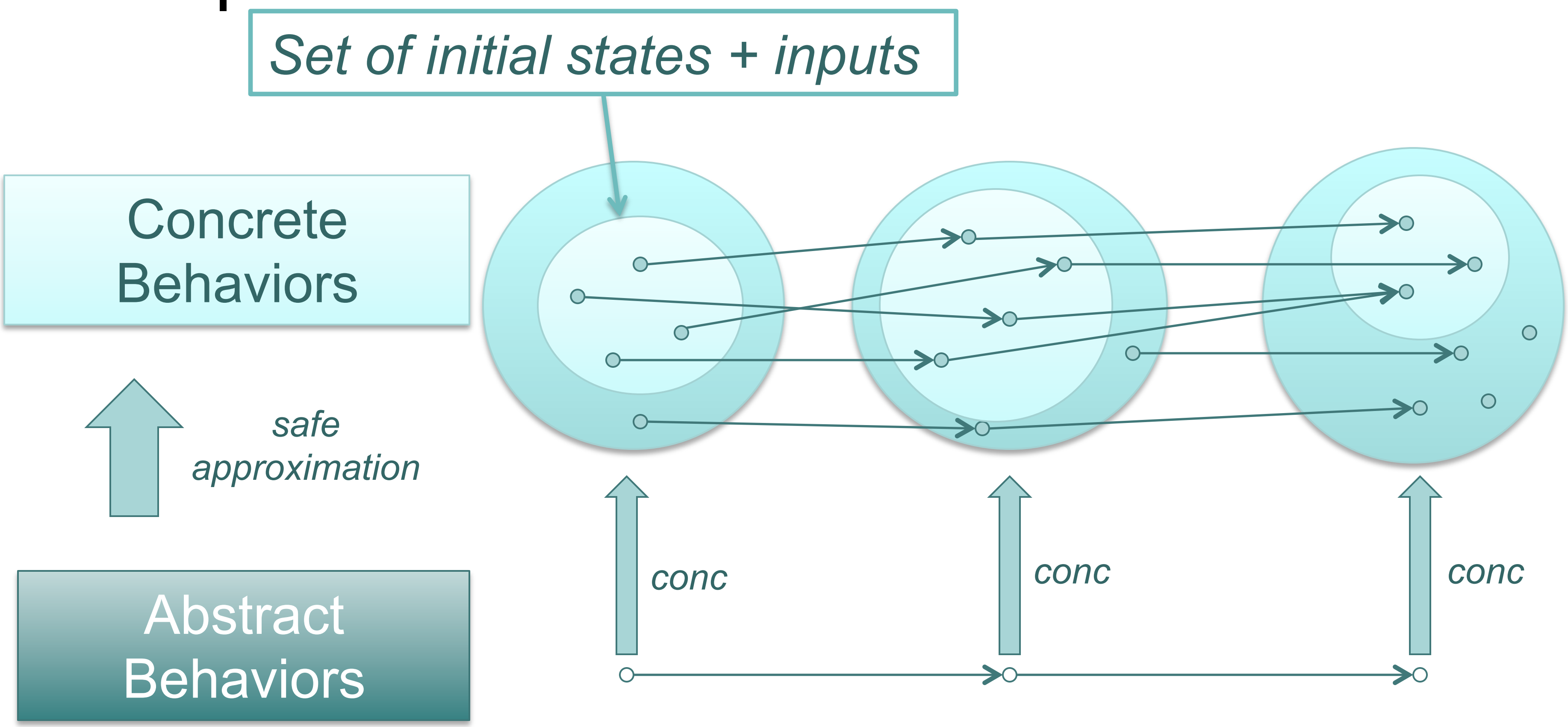
Dijkstra (1970)

Sound Static Program Analysis (Abstract Interpretation)

Over-approximate all possible behaviors:

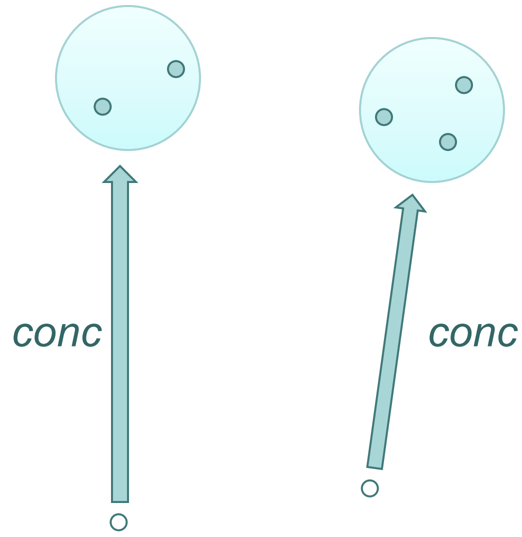


Static Analysis in a Nutshell



How to achieve „safe approximation“?

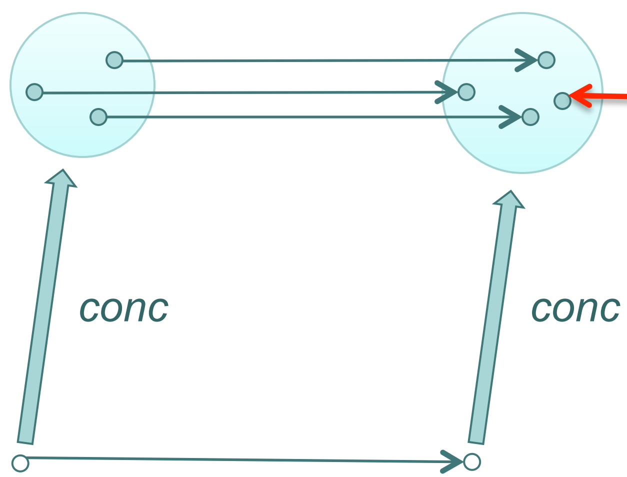
1. Every abstract state $s^\#$ represents a set *conc*($s^\#$) of concrete states:



How to achieve „safe approximation“?

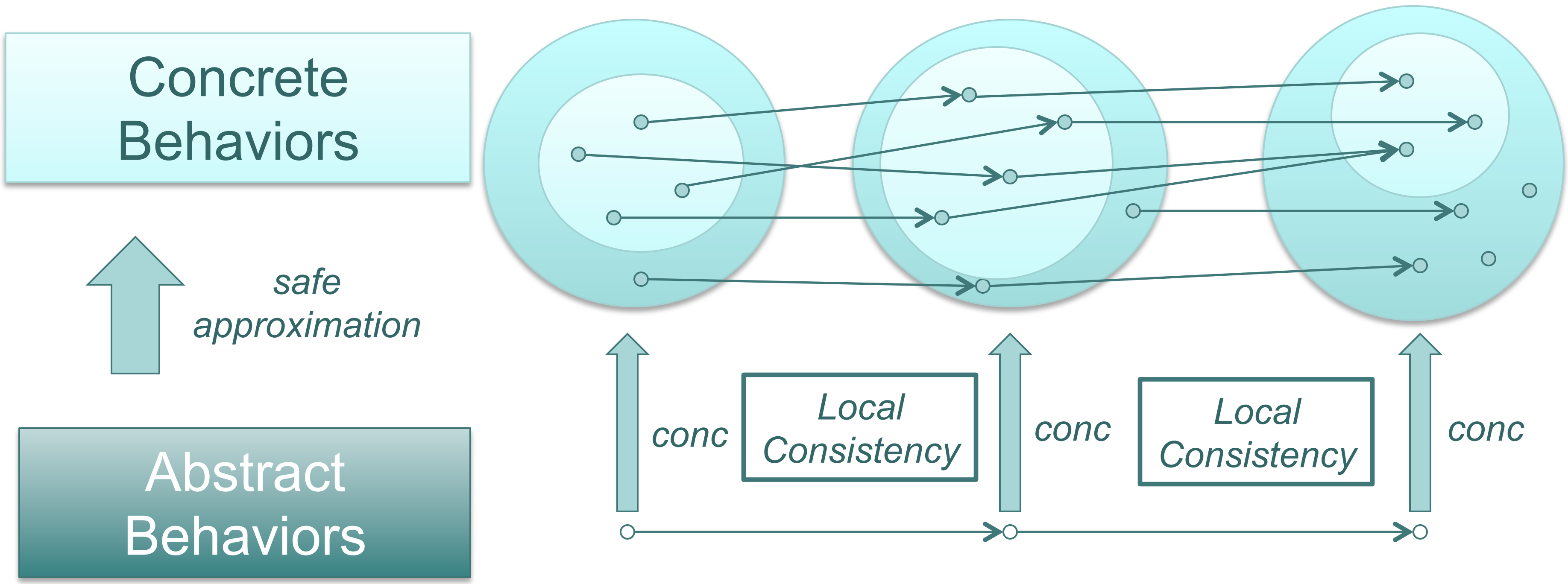
2. ***Local Consistency***:

„abstract successors“ over-approximate „concrete successors“



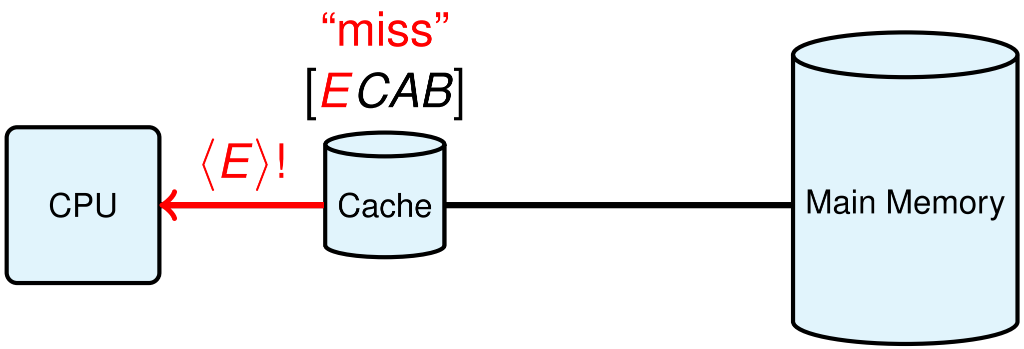
may represent more states → ***imprecision***

How to achieve „safe approximation“?



Application: Static Cache Analysis

Cache: Small, but fast memory that buffers part of main memory



Goal of static cache analysis is to classify memory accesses as **guaranteed cache hits**

Concrete Cache States: Least-Recently-Used Replacement

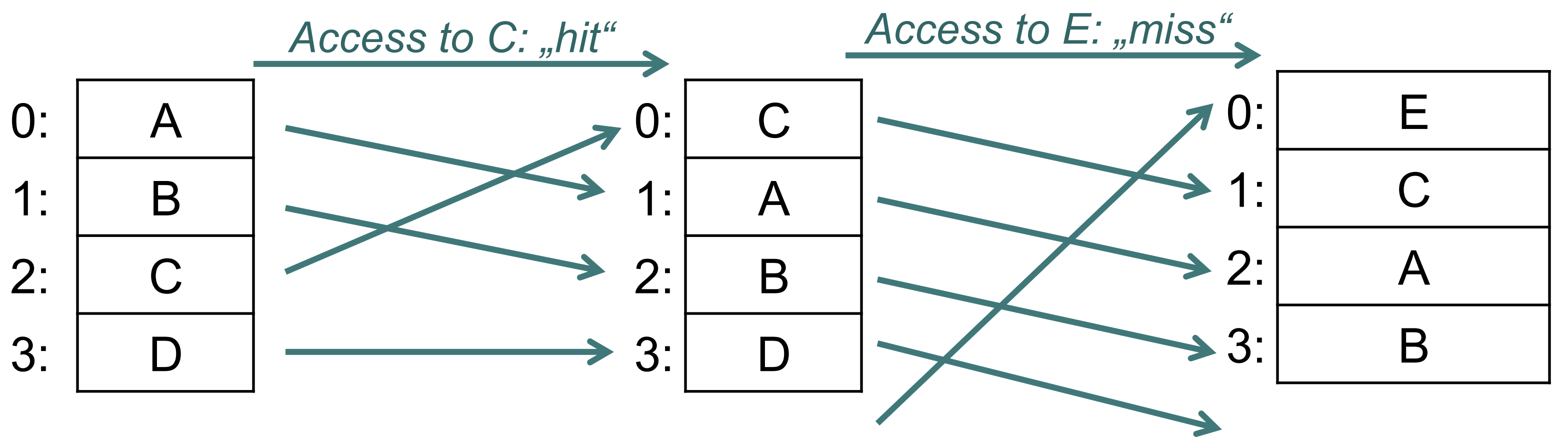
Notion of Age

0:	A	*most-recently-used*
1:	B	
2:	C	
3:	D	*least-recently-used*

0:	E	*most-recently-used*
1:	B	
2:	F	
3:	D	*least-recently-used*

Concrete Cache Behavior: Least-Recently-Used Replacement

How do concrete cache states change upon memory accesses?



Brainstorming: Predicting Cache Hits

Want to soundly predict cache hits.

How to compactly represent sets of concrete states for this purpose?

Abstract "Must" Cache States

Compactly represent sets of concrete states:

0:	{}
1:	{B}
2:	{A,C}
3:	{D}

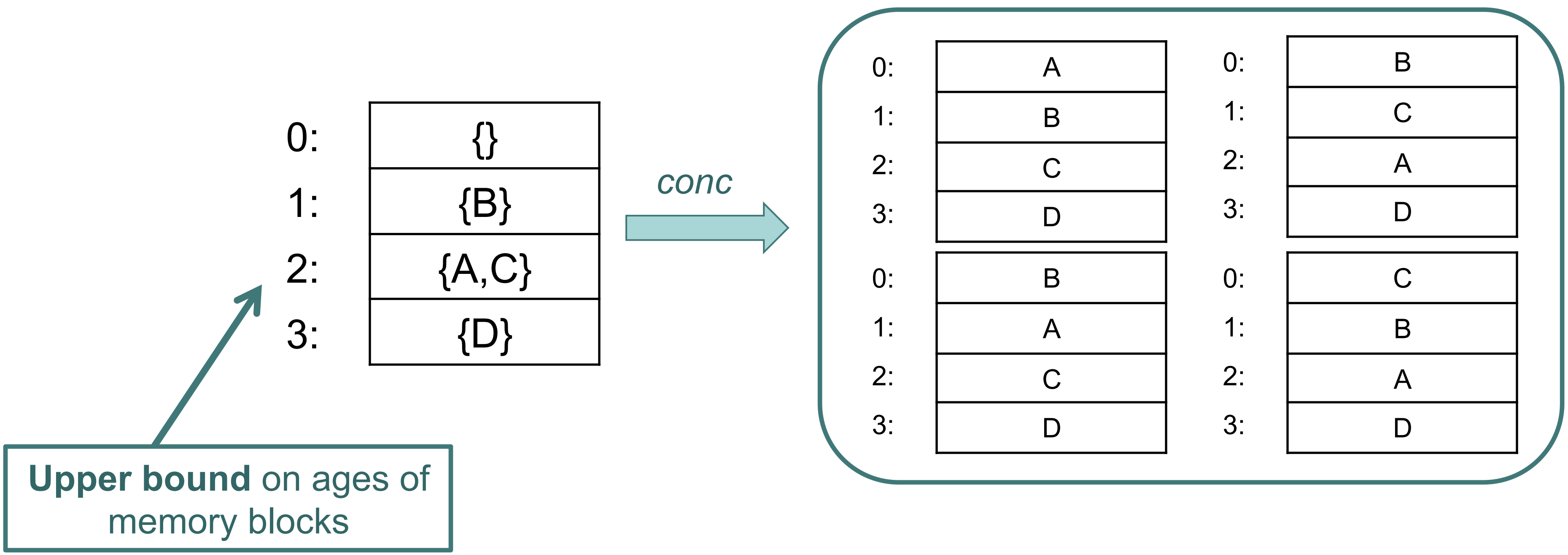
conc

Concrete states in which:
B's age is at most 1,
A's and C's age is at most 2
and D's age is at most 3.

Upper bound on ages of memory blocks

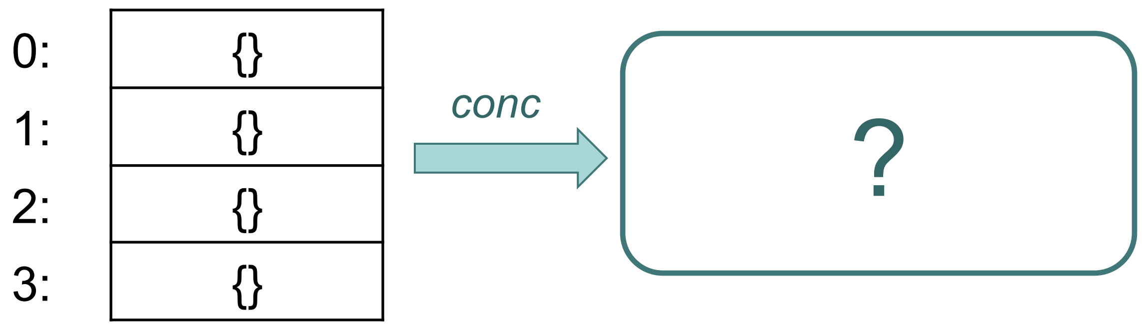
Abstract “Must” Cache States

Compactly represent sets of concrete states:



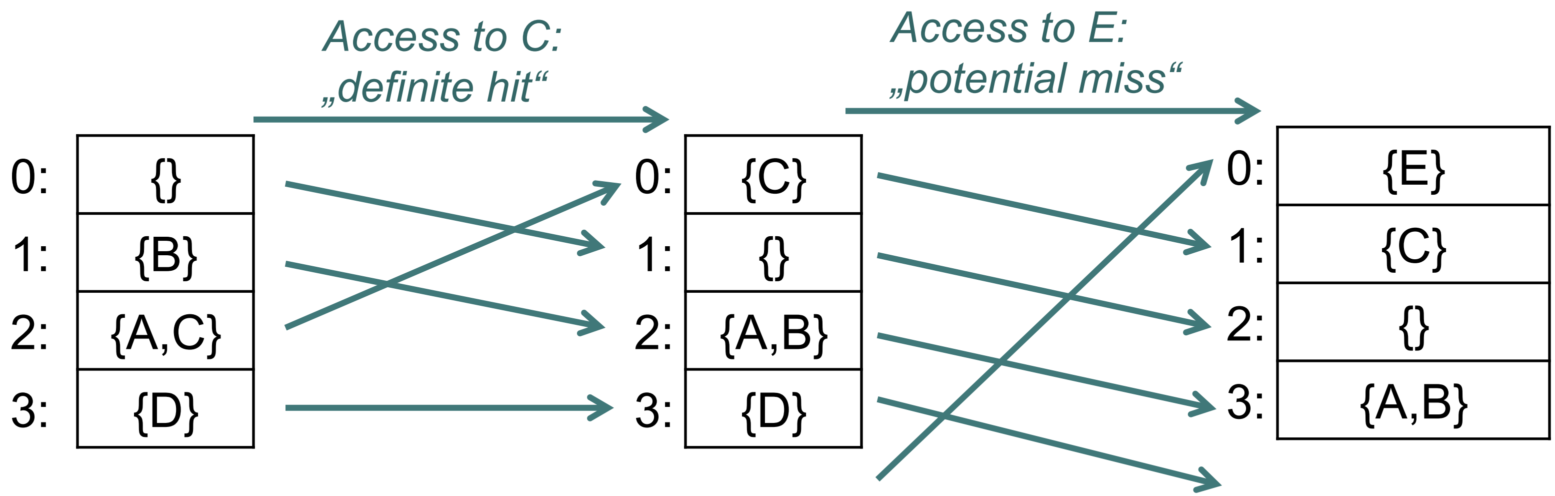
Abstract “Must” Cache States

Compactly represent sets of concrete states:

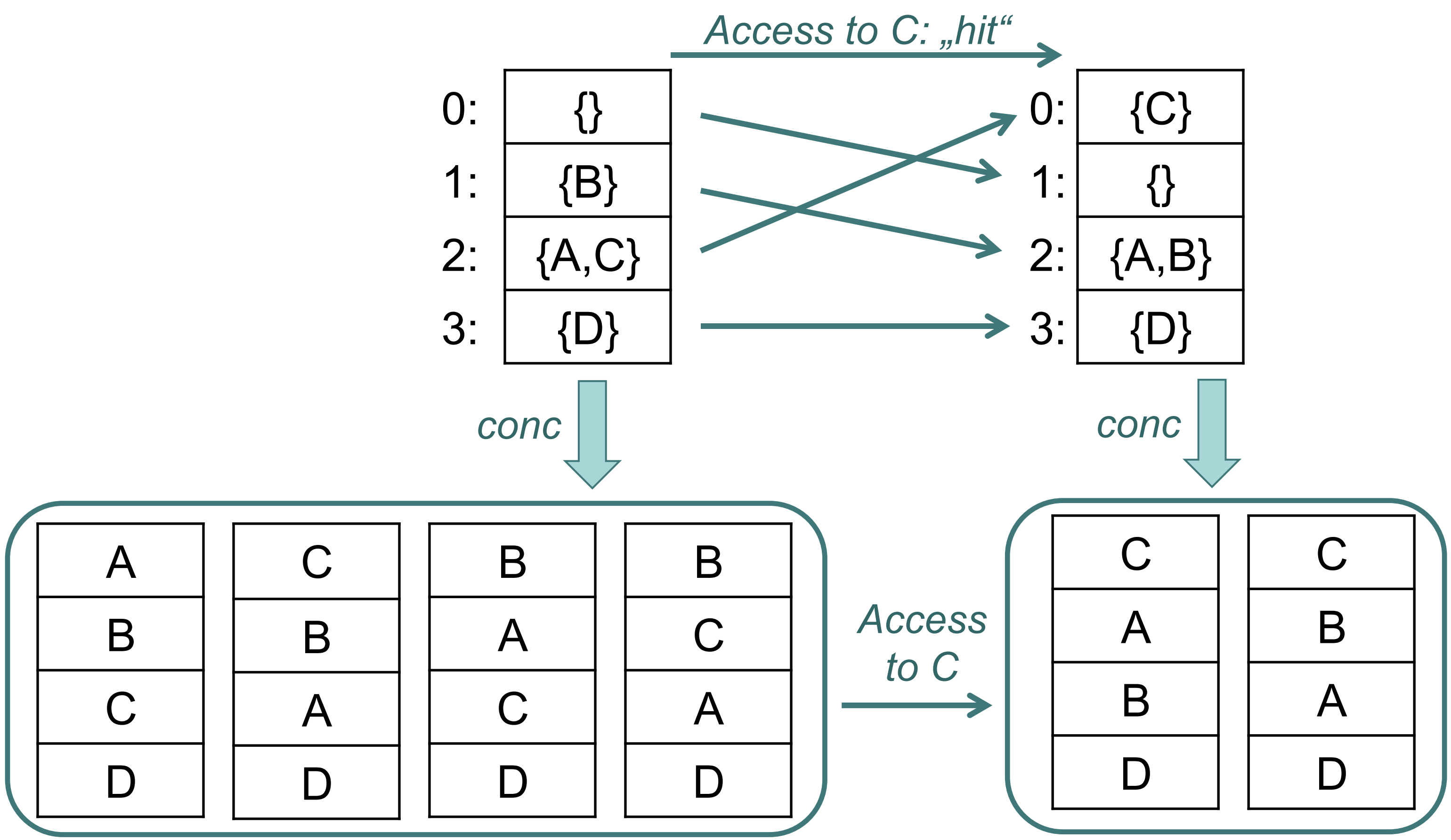


Abstract “Must” Cache Behavior

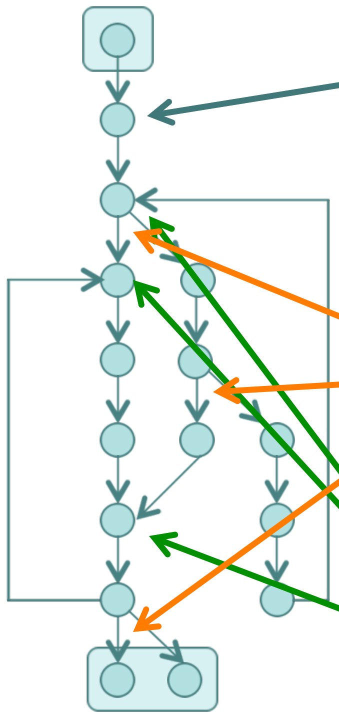
How do abstract cache states change upon memory accesses?



Abstract "Must" Cache Behavior: Local Consistency



Integration with Pipeline Analysis



During microarchitectural analysis, analysis states are pairs:

(pipeline state, cache state)

Upon uncertainty, e.g. whether a cache hit or miss happens, "split".

If two analysis states agree on the pipeline state, the cache states are "joined".

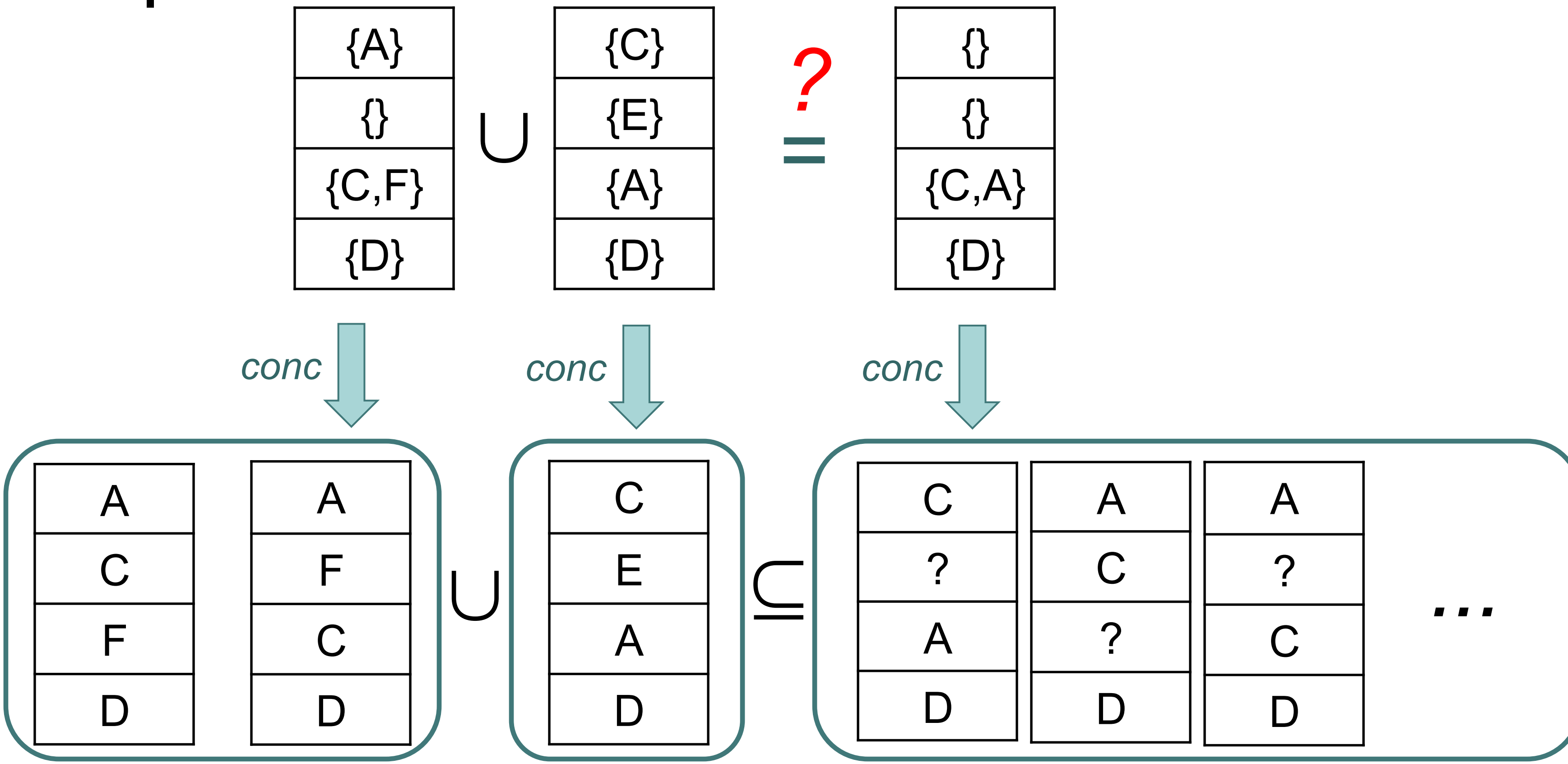
Properties of the Join

- *Soundness*:
  Must *not lose* any concrete states:

  $$conc(A) \subseteq conc(A \cup B)$$
  $$conc(B) \subseteq conc(A \cup B)$$

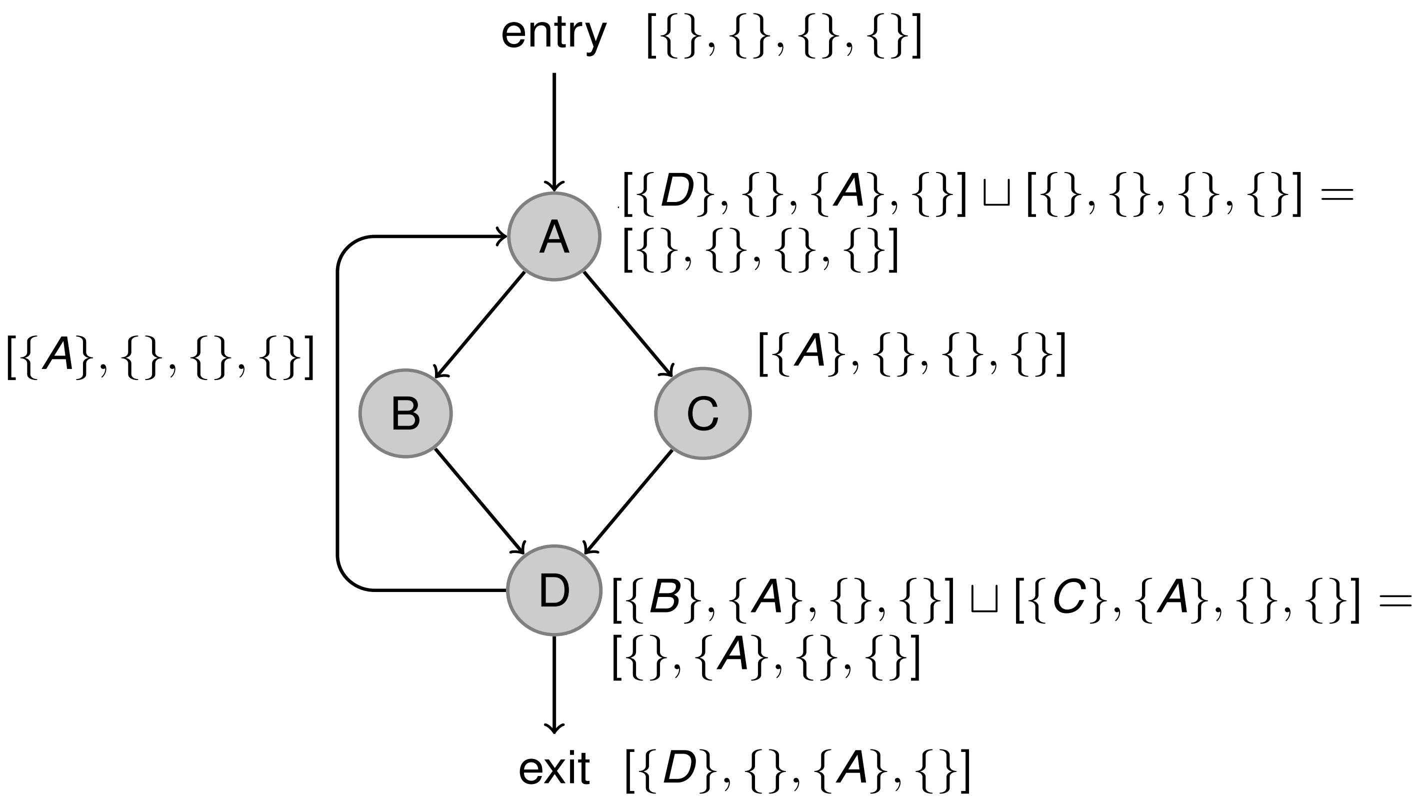
- *Precision*:
  Want the "best" representation that is sound, i.e. the one that represents the *fewest* concrete states.

Join for Abstract "Must" Cache States



→ Take the maximum of the age bounds!

Example Analysis: A Loop



No hits can be predicted! Why?

Context-sensitive Analysis: Virtual Loop Peeling (~~Unrolling~~)

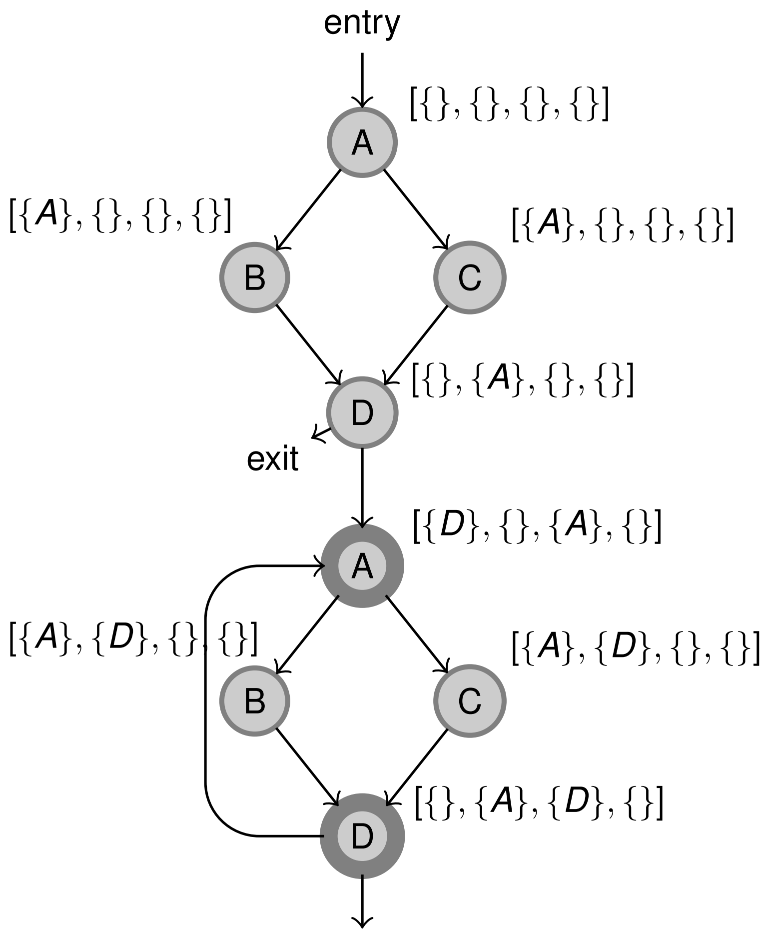
The problem:

- The first iteration of a loop will always result in cache misses
- Similarly for the first execution of a recursive function

A solution:

- Distinguish the first iteration of a loop from others
- Distinguish function calls by calling context

Context-sensitive Analysis: Example



- Accesses to A and D are provably hits after the first iteration
- Accesses to B and C can still not be classified, but they can miss at most once
  → Need Persistence Analysis

***Note**: Loop is only peeled virtually, i.e. during analysis not in the binary!*

Brainstorming: Predicting Cache Misses

Want to soundly predict cache misses.

How to compactly represent sets of concrete states for this purpose?

Predicting Cache Misses: Abstract "May" Cache States

0:	{A}
1:	{B, D}
2:	{C, E, F}
3:	{}

conc

Concrete states in which:
A's age is at least 0,
B's and D's age is at least 1, and
C's, E's, and F's age is at least 2.

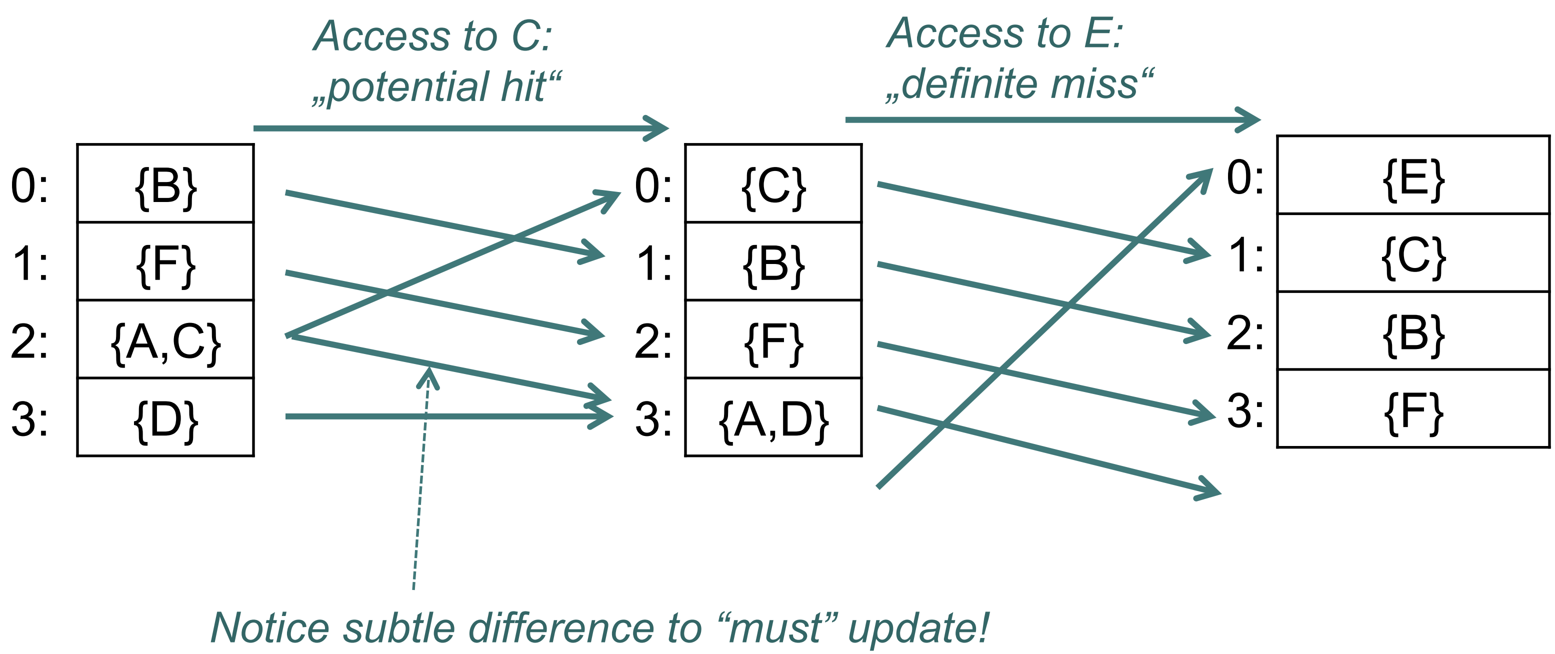
Lower bound on ages of memory blocks

Blocks with a lower bound greater or equal to the cache size are guaranteed not to be in the cache.

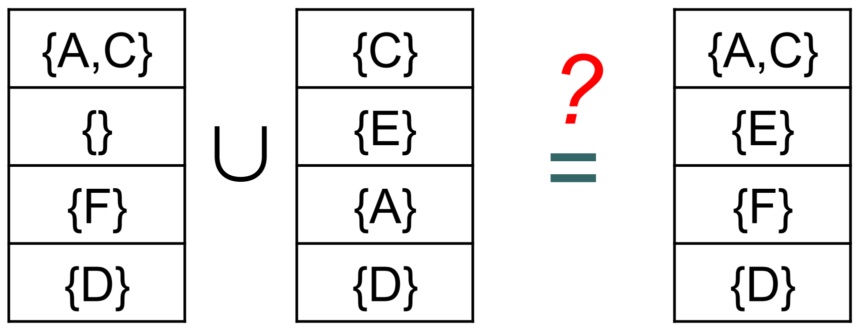
How is this useful in WCET analysis?

Abstract “May” Cache Behavior

How do abstract cache states change upon memory accesses?

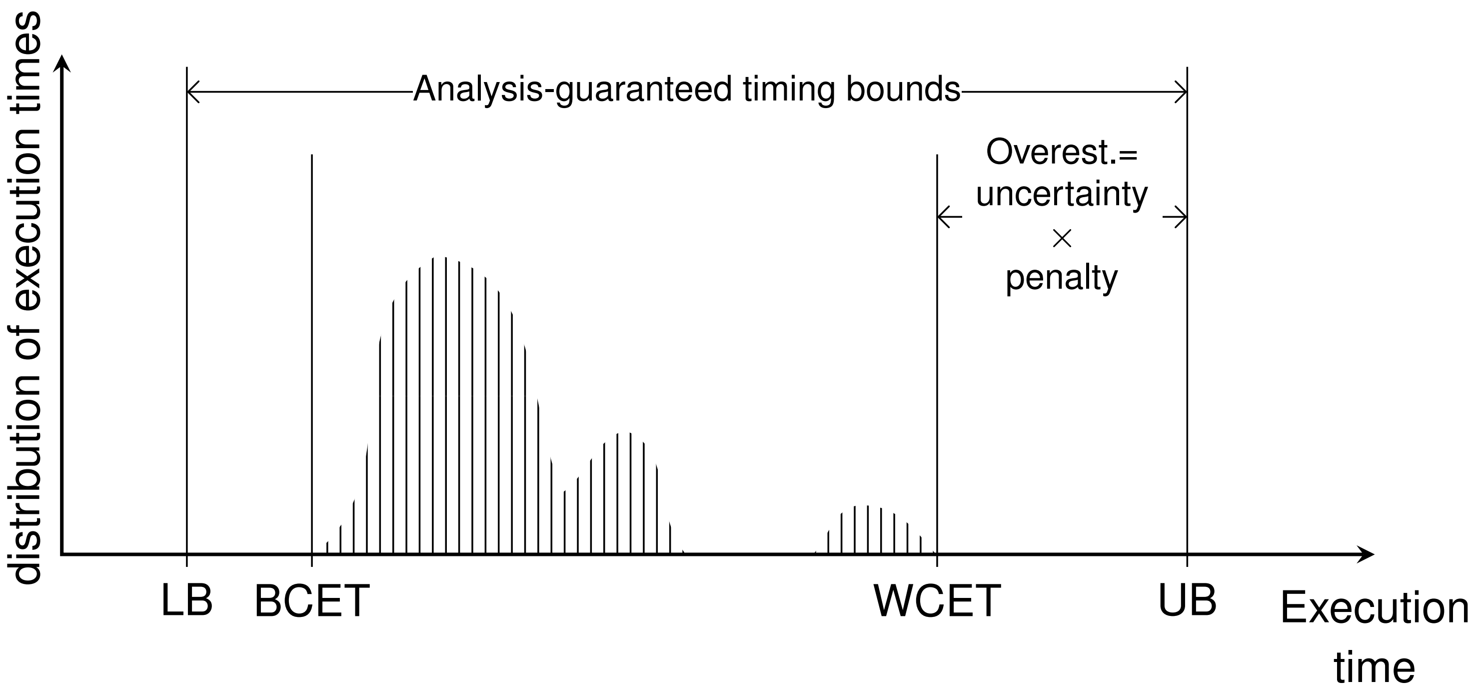


Join for Abstract “May” Cache States



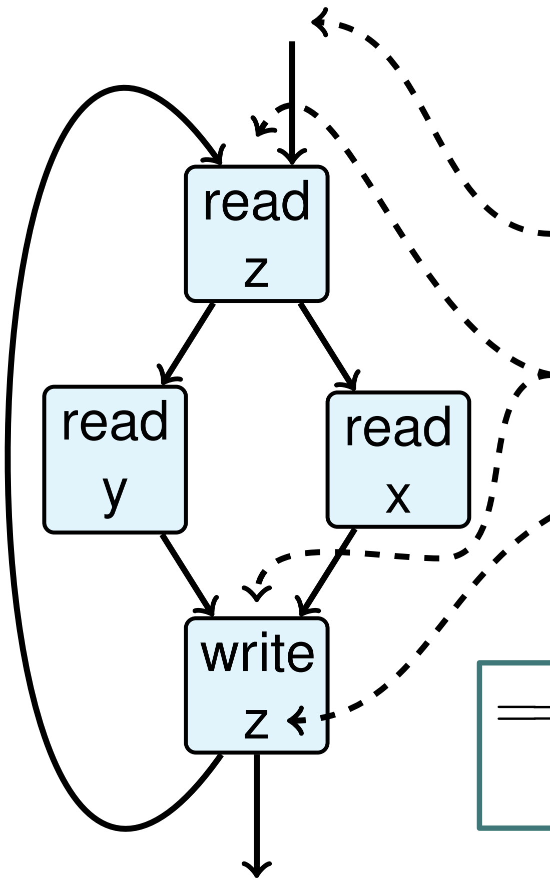
→ Take the minimum of the age bounds!

Timing Predictability
Uncertainty in WCET Analysis



- Precision of WCET analysis determined by amount of uncertainty
- Uncertainty in cache analysis depends on replacement policy

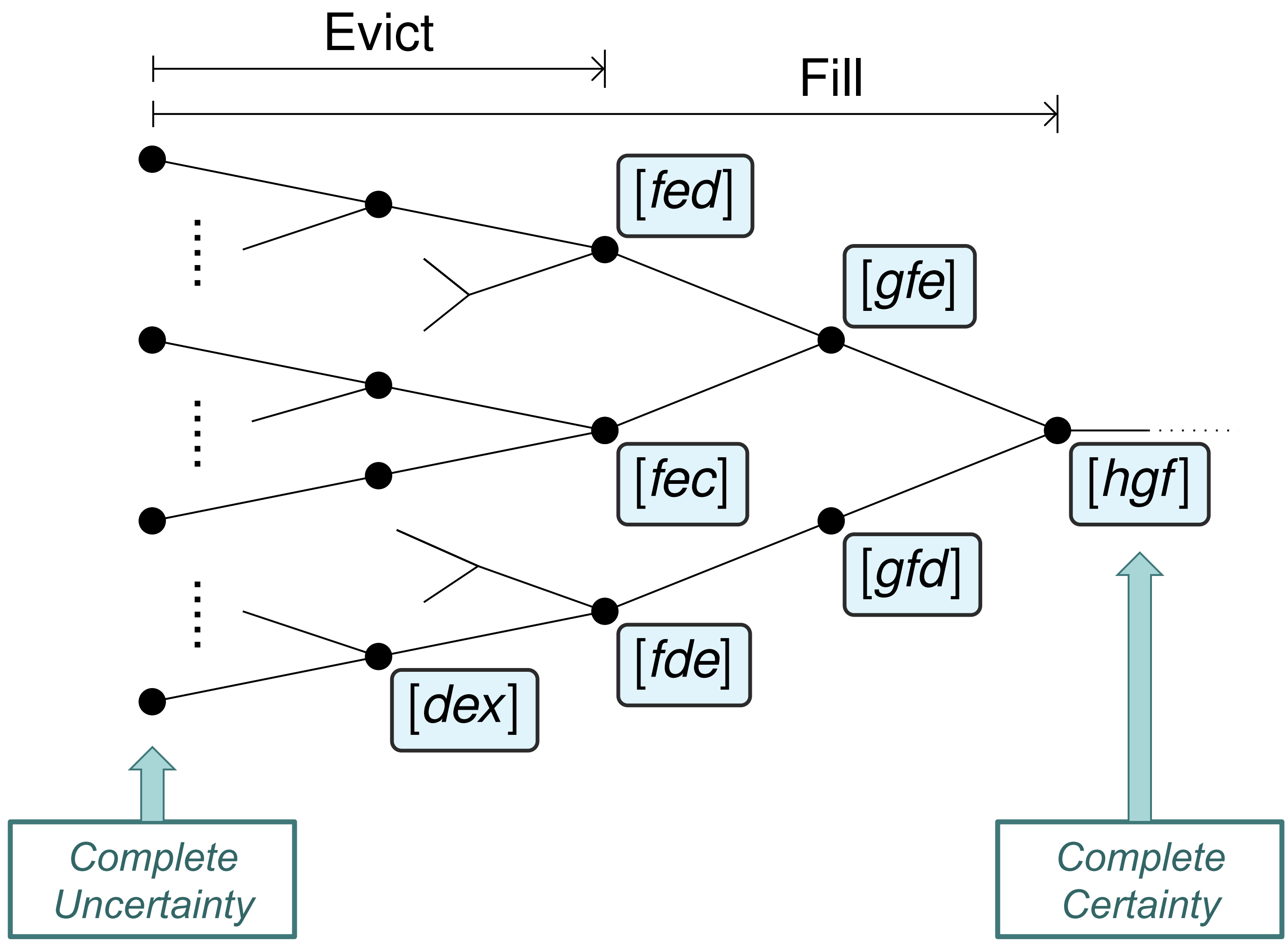
Uncertainty in Cache Analysis



1. Initial cache contents unknown.
2. Different paths lead to these points.
3. Cannot resolve address of *z*.

$\Longrightarrow$ Amount of uncertainty determined by ability to recover information

Cache Predictability Metrics



Interpretation of Predictability Metrics

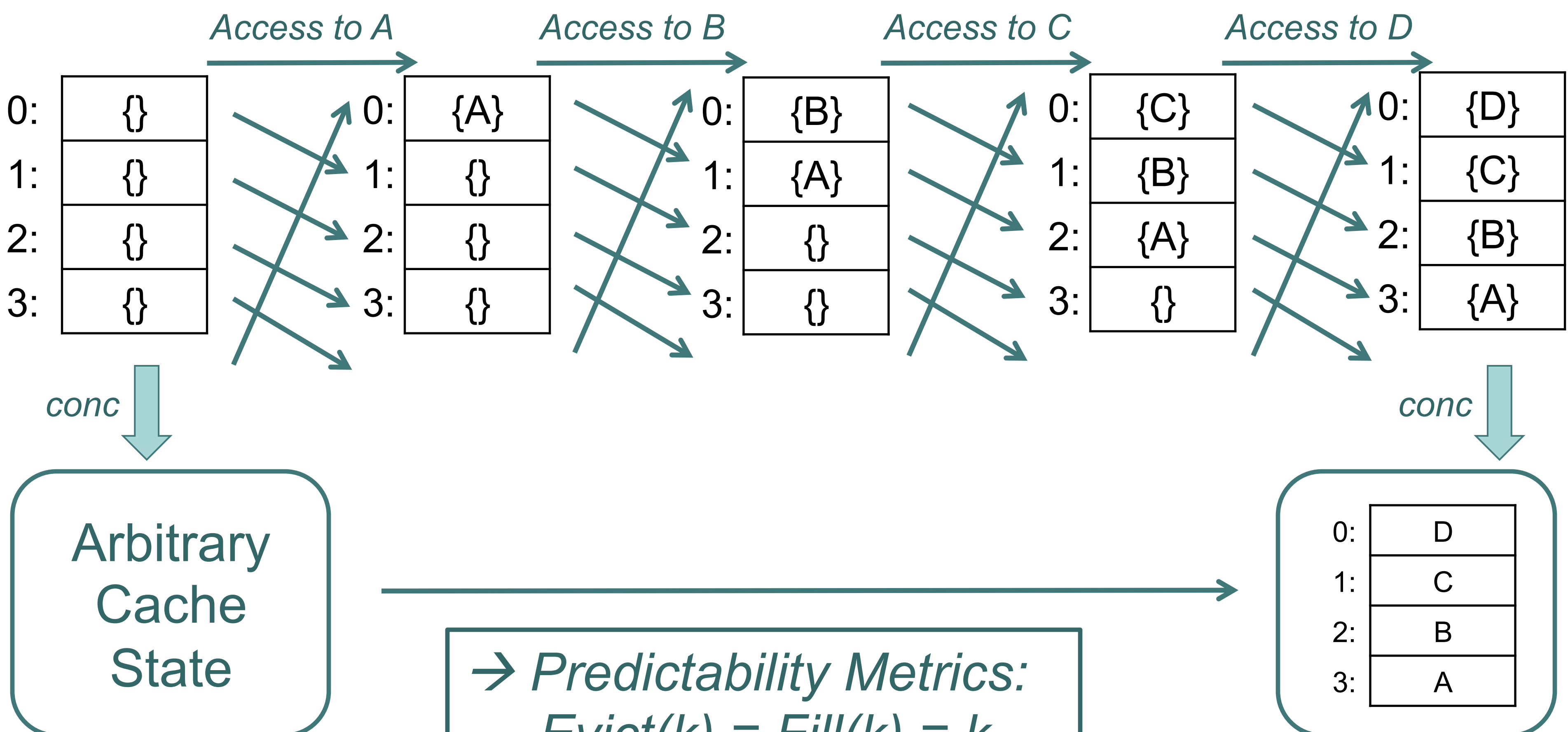
- *Evict*:
  - Number of accesses until any analysis can start predicting guaranteed cache misses
- *Fill*:
  - Number of accesses until any analysis can precisely predict any access as hit or miss

→ The two metrics bound the precision of ***any*** static cache analysis.
Can thus serve as benchmark for analyses.

Popular Cache Replacement Policies

- Least-Recently-Used (LRU) used in
  INTEL PENTIUM I, MIPS 24K/34K, AMD OPTERON
- First-In First-Out (FIFO) used in
  MOTOROLA POWERPC 56X, INTEL XSCALE, ARM9, ARM11
- Not Most-Recently-Used (NMRU) used in
  INTEL ITANIUM
- Pseudo-LRU (PLRU) used in
  INTEL PENTIUM II-IV, INTEL CORE, INTEL XEON, POWERPC 75X

Observation: LRU Replacement is „Robust“

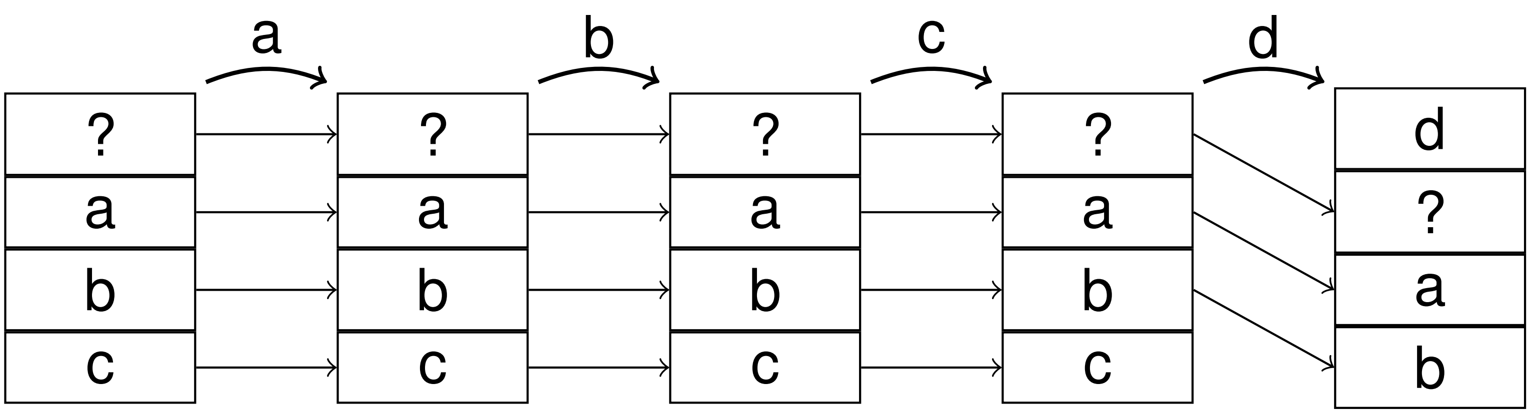


→ *Predictability Metrics:*
Evict(k) = Fill(k) = k

where k = associativity

Predictability Metrics: FIFO

- Like LRU in the miss case
- But: “Ignores” hits



In the worst case k-1 hits and k misses: (k = associativity)

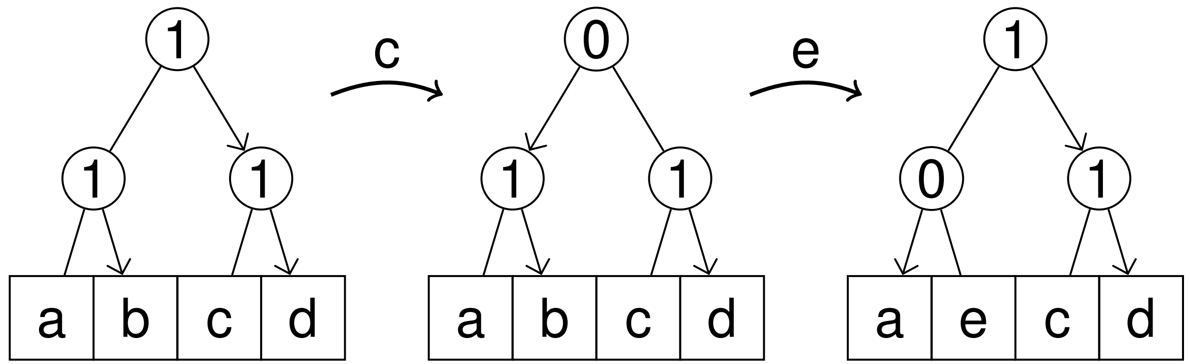
→ Evict(k) = 2k-1

Another k accesses to obtain complete knowledge:

→ Fill(k) = 3k-1

Predictability Metrics: PLRU

Tree bits point to block to be replaced



Accesses „rejuvenate“ neighborhood

- Active blocks keep their (inactive) neighborhood in the cache

Analysis yields:

- $\text{Evict}(k) = k/2 \log_2 k + 1$
- $\text{Fill}(k) = k/2 \log_2 k + k - 1$

Evaluation of Policies

Policy	Evict(k)	Fill(k)
LRU	k	k
FIFO	2k-1	3k-1
NMRU	2k-2	3k-4
PLRU	k/2 log k + 1	k/2 log k + k-1

1. LRU is **optimal** w.r.t. metrics
→ Use LRU when you have the choice.

2. How to build static analyses that are optimal w.r.t. metrics? → [SIGMETRICS, LCTES, SAS, WCET, ECRTS]

Conclusions

- Cache Analysis for Least-Recently-Used:
  - Efficiently represents sets of cache states by bounding the age of memory blocks
    - From above: “must” analysis
    - From below: “may” analysis
  - Requires context sensitivity for precision
- Cache Predictability Metrics:
  - Equate predictability with “forgetfulness”
  - LRU is particularly “forgetful”

Literature

Abstract Interpretation:
- Cousot, Cousot: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, In: Proceedings POPL, 1977

LRU:
- Ferdinand, Wilhelm: Efficient and precise cache behavior prediction for real-time systems, Real-Time Systems, 1999

Predictability:
- Reineke, Grund, Berg, Wilhelm: Timing predictability of cache replacement policies, Real-Time Systems 37(2), 2007

Other Policies:
- Reineke, Grund: Relative competitive analysis of cache replacement policies, In: Proceedings LCTES, 2008

FIFO:
- Grund, Reineke: Abstract interpretation of FIFO replacement, In: Proceedings SAS, 2009
- Grund, Reineke: Precise and efficient FIFO-replacement analysis based on static phase detection, In: Proceedings ECRTS, 2010
- Guan, Yang, Lv, Yi: FIFO cache analysis for WCET estimation: a quantitative approach, In: Proceedings DATE, 2013

Persistence:
- Cullmann: Cache persistence analysis: Theory and practice, ACM TECS, 2013

NMRU:
- Guan, Lv, Yi, Yu: WCET analysis with MRU cache: challenging LRU for predictability, ACM TECS, 2014

Survey:
- Lv, Guan, Reineke, Wilhelm, Yi: A survey on static cache analysis for real-time systems, LITES 3(1), 2016