Timing analysis and timing predictability

Caches in WCET Analysis

Reinhard Wilhelm[1] Jan Reineke[2]

[1]Saarland University, Saarbrücken, Germany

[2]University of California, Berkeley, USA

January 18, 2011

SAARLAND UNIVERSITY

COMPUTER SCIENCE

Outline

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1 Caches

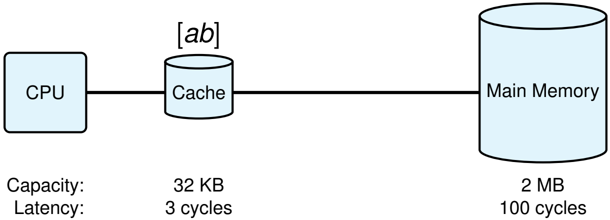
2 Cache Analysis for Least-Recently-Used

3 Beyond Least-Recently-Used
- Predictability Metrics
- Relative Competitiveness
- Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary

Caches

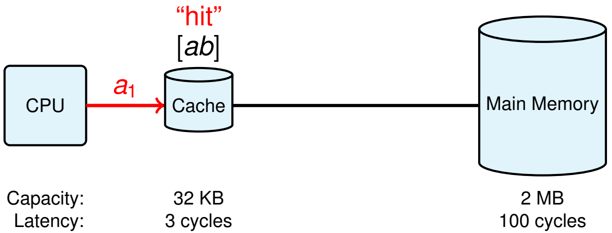
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

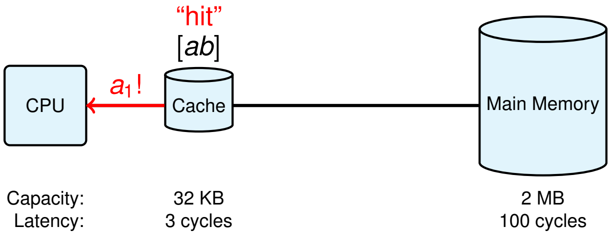
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

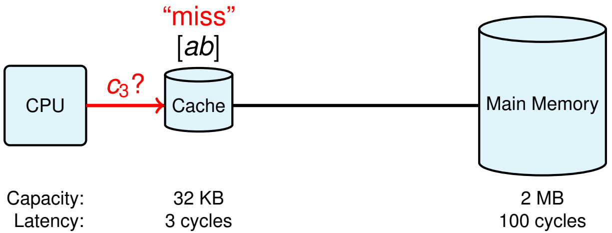
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

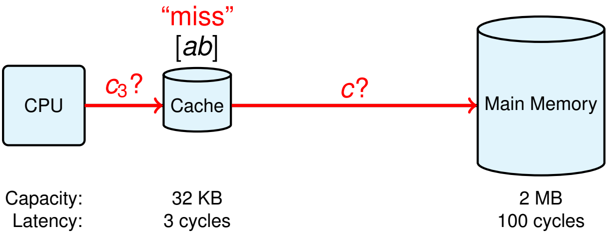
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

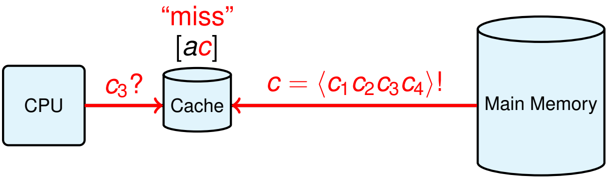
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



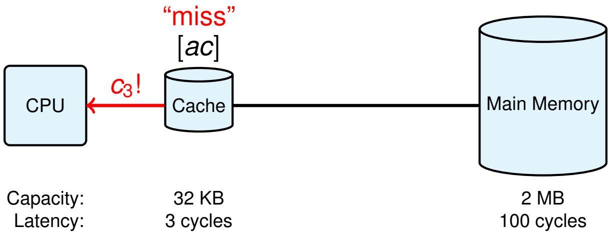
Capacity:	32 KB	2 MB
Latency:	3 cycles	100 cycles

- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

SAARLAND UNIVERSITY
COMPUTER SCIENCE

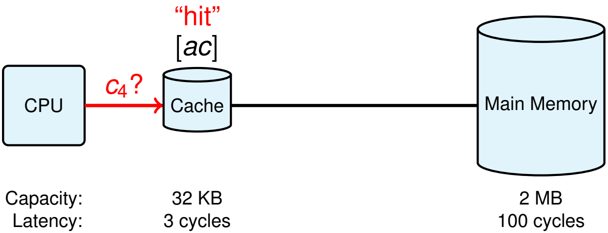
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Caches

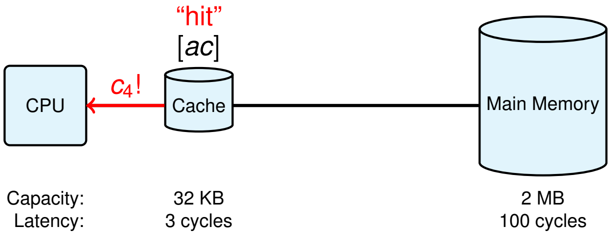
- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory



- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

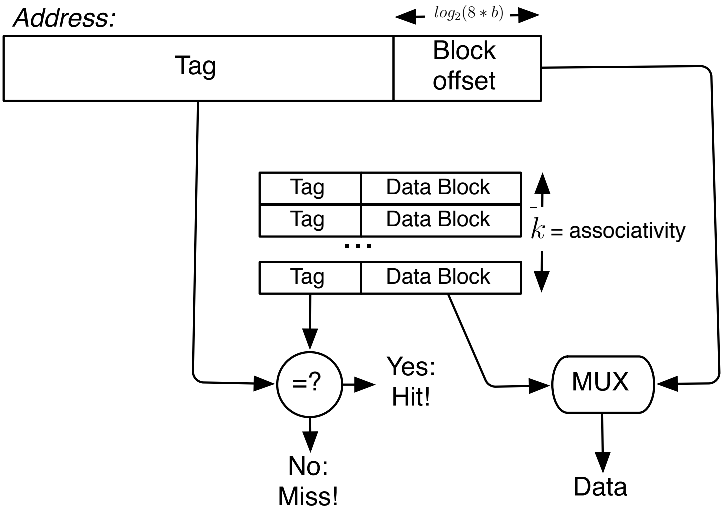
Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

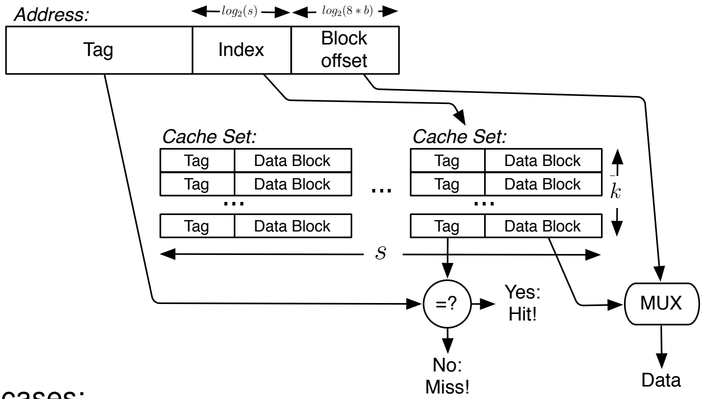


- Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

Fully-Associative Caches



Set-Associative Caches



Special cases:

- direct-mapped cache: only one line per cache set
- fully-associative cache: only one cache set

Cache Replacement Policies

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Least-Recently-Used (LRU) used in
  INTEL PENTIUM I and MIPS 24K/34K
- First-In First-Out (FIFO or Round-Robin) used in
  MOTOROLA POWERPC 56X, INTEL XSCALE, ARM9, ARM11
- Pseudo-LRU (PLRU) used in
  INTEL PENTIUM II-IV and POWERPC 75X
- Most-Recently-Used (MRU) as described in literature

Each cache set is treated independently:
⟶ Set-associative caches are compositions of fully-associative caches.

Outline

Cache Analysis

Two types of cache analyses:

1. Local guarantees: classification of individual accesses
   - May-Analysis $\longrightarrow$ Overapproximates cache contents
   - Must-Analysis $\longrightarrow$ Underapproximates cache contents
2. Global guarantees: bounds on cache hits/misses

- Cache analyses almost exclusively for LRU
- In practice: FIFO, PLRU, ...

Abstract Interpretation in Timing Analysis

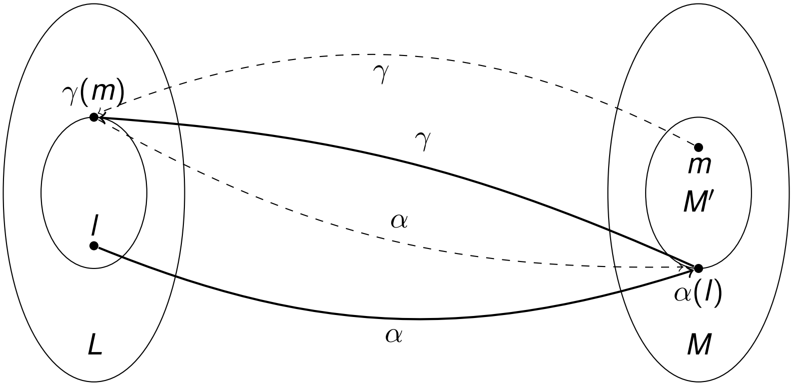
SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Abstract interpretation is always based on the semantics of the analyzed language.
- A semantics of a programming language that talks about time needs to incorporate the execution platform!
- Static timing analysis is thus based on such a semantics.

Galois Connection

- Abstraction function α
- Concretization function γ

$\Rightarrow \forall m' \in M' : \gamma(m') = \gamma(m)$

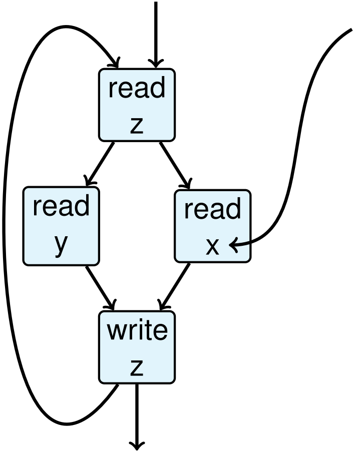


Abstract Interpretation in Timing Analysis

Determines:

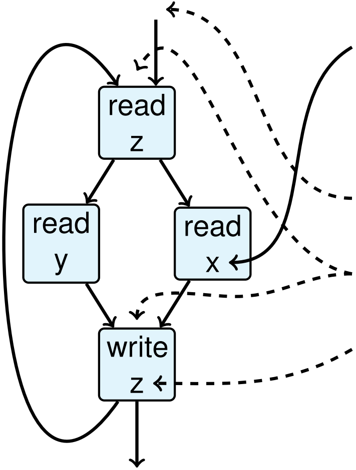
1. invariants about the values of variables (in registers, on the stack)
   - to compute loop bounds
   - to eliminate infeasible paths
   - to determine effective memory addresses
2. invariants on architectural execution state
   - Cache contents $\Rightarrow$ predict hits and misses
   - Pipeline states $\Rightarrow$ predict or exclude pipeline stalls

Challenges for Cache Analysis



Always a cache hit/always a miss?

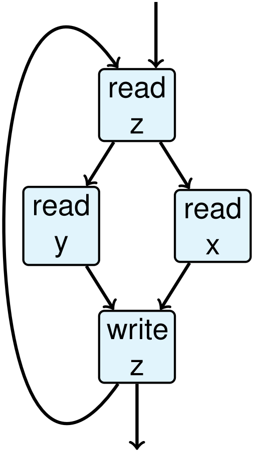
Challenges for Cache Analysis



Always a cache hit/always a miss?

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

Deriving Invariants about Cache States using Abstract Interpretation

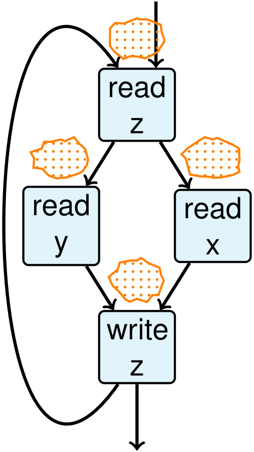


Collecting Semantics =
set of states at each program point that any execution may encounter there

Two approximations:

	Collecting Semantics	uncomputable
$\subseteq$	Cache Semantics	computable
$\subseteq$	γ(Abstract Cache Sem.)	efficiently computable

Deriving Invariants about Cache States using Abstract Interpretation

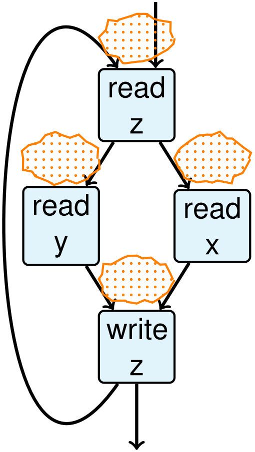


Collecting Semantics =
set of states at each program point that any execution may encounter there

Two approximations:

	Collecting Semantics	uncomputable
$\subseteq$	Cache Semantics	computable
$\subseteq$	γ(Abstract Cache Sem.)	efficiently computable

Deriving Invariants about Cache States using Abstract Interpretation

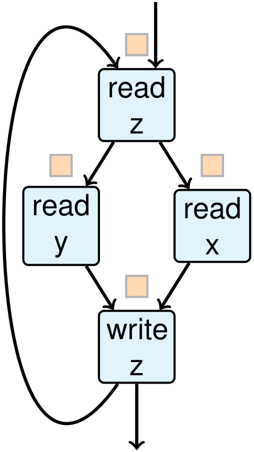


Collecting Semantics =
set of states at each program point that any execution may encounter there

Two approximations:

	Collecting Semantics	uncomputable
$\subseteq$	Cache Semantics	computable
$\subseteq$	γ(Abstract Cache Sem.)	efficiently computable

Deriving Invariants about Cache States using Abstract Interpretation

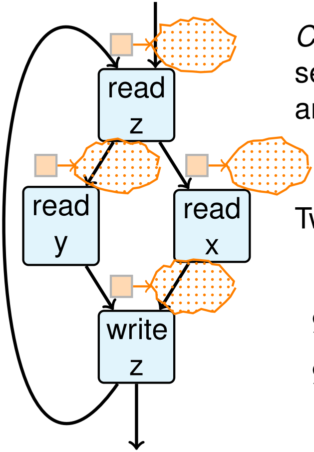


Collecting Semantics =
set of states at each program point that any execution may encounter there

Two approximations:

	Collecting Semantics	uncomputable
$\subseteq$	Cache Semantics	computable
$\subseteq$	γ(Abstract Cache Sem.)	efficiently computable

Deriving Invariants about Cache States using Abstract Interpretation



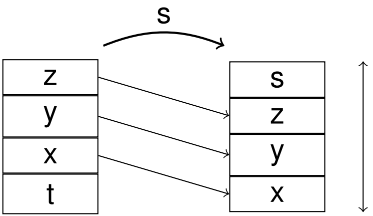
Collecting Semantics =
set of states at each program point that any execution may encounter there

Two approximations:

	Collecting Semantics	uncomputable
$\subseteq$	Cache Semantics	computable
$\subseteq$	γ(Abstract Cache Sem.)	efficiently computable

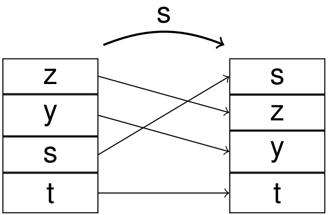
Least-Recently-Used (LRU): Concrete Behavior

“Cache Miss”:



LRU has notion of age

“Cache Hit”:



LRU: Must-Analysis: Abstract Domain

- Used to predict *cache hits*.
- Maintains *upper bounds on ages* of memory blocks.
- Upper bound $\leq$ associativity $\longrightarrow$ memory block definitely cached.

Example

Abstract state:

{x}	age 0
{}	
{s,t}	
{}	age 3

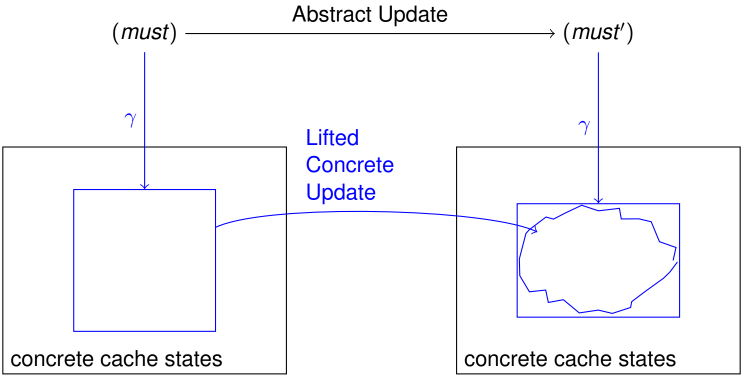
...and its interpretation:

Describes the set of all concrete cache states in which x, s, and t occur,

- x with an age of 0,
- s and t with an age not older than 2.

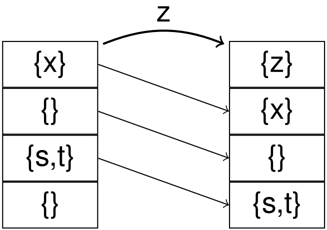
$$\gamma([\{x\},\{\},\{s,t\},\{\}]) = \{[x,s,t,a],[x,t,s,a],[x,s,t,b],\ldots\}$$

Sound Update – Local Consistency

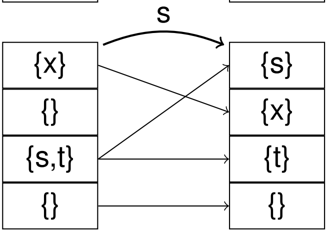


LRU: Must-Analysis: Update

"Potential Cache Miss":



"Definite Cache Hit":



Why does *t* not age in the second case?

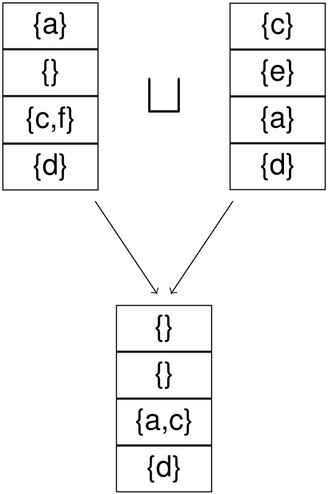
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Intersection + Maximal Age"



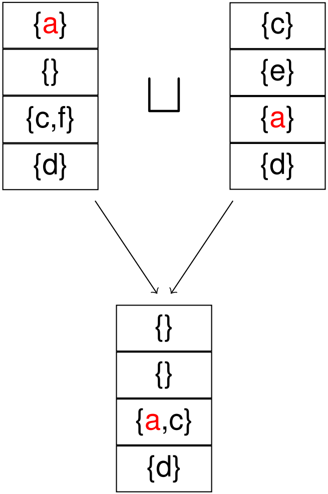
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Intersection + Maximal Age"



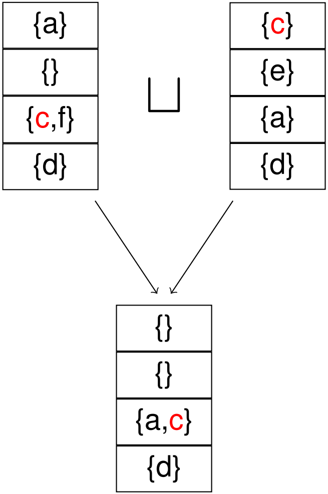
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

“Intersection + Maximal Age”



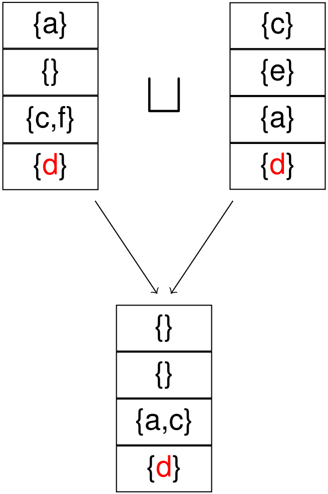
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Intersection + Maximal Age"



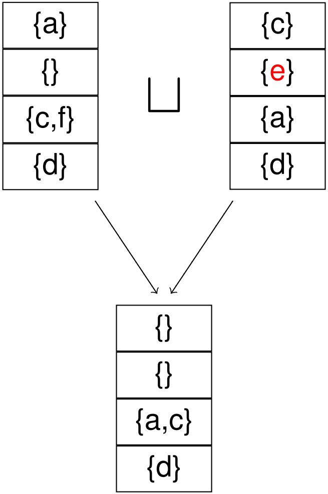
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Intersection + Maximal Age"



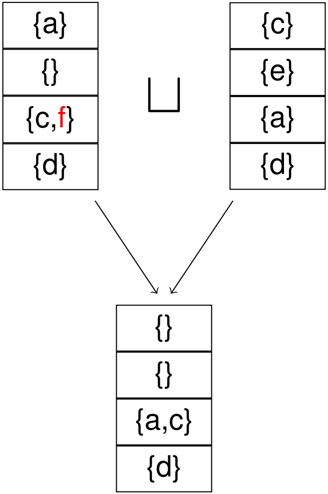
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Intersection + Maximal Age"

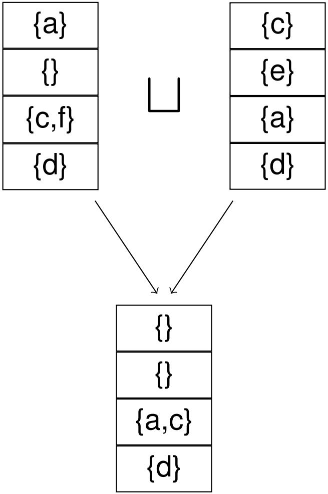


Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

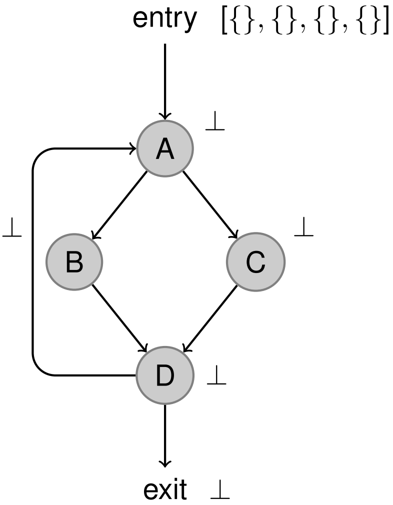
- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$



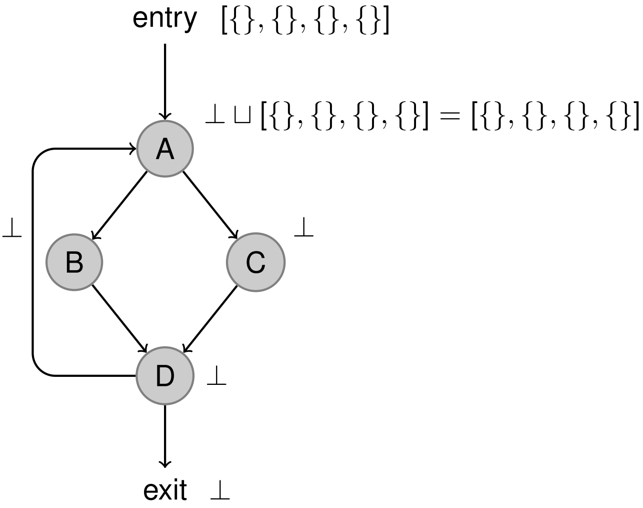
"Intersection + Maximal Age"

How many memory blocks can be in the must-cache?

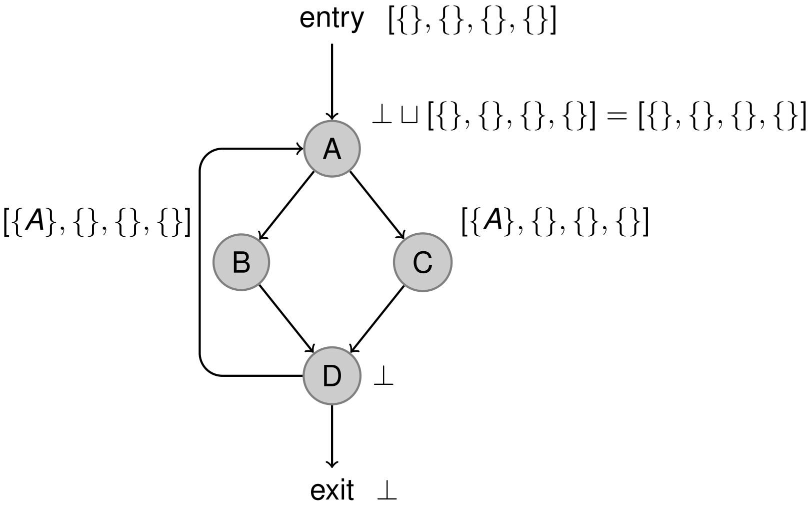
Example: Must-Analysis



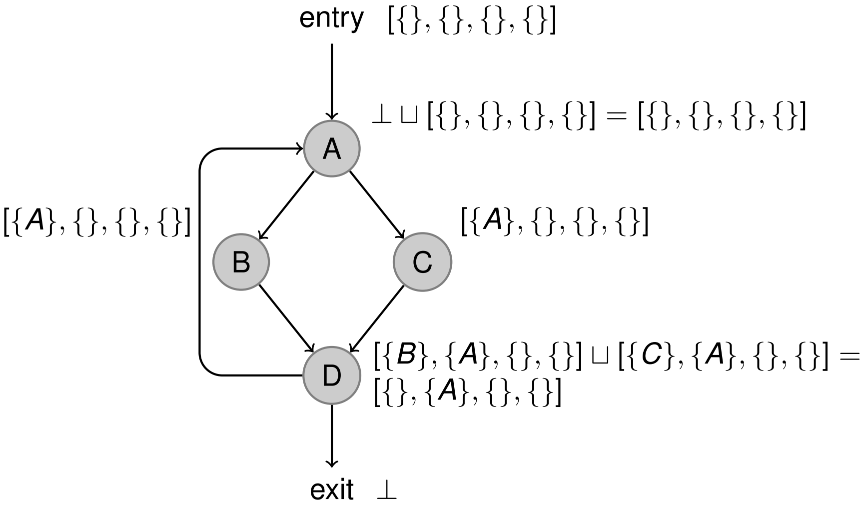
Example: Must-Analysis



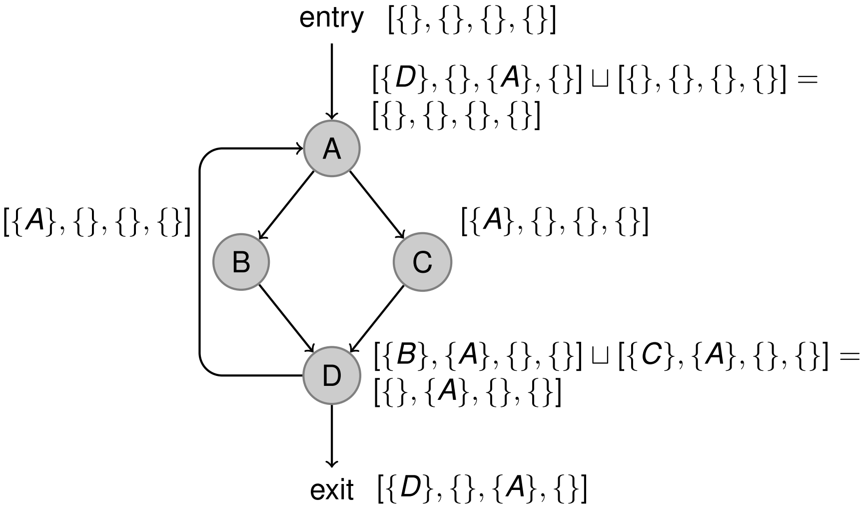
Example: Must-Analysis



Example: Must-Analysis



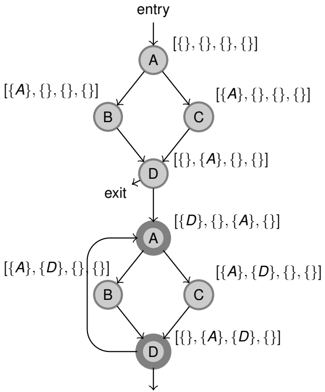
Example: Must-Analysis



No cache hits can be predicted :-(

Context-Sensitive Analysis/Virtual Loop-Unrolling

- Problem:
  - The first iteration of a loop will always result in cache misses.
  - Similarly for the first execution of a function.
- Solution:
  - Virtually Unroll Loops: Distinguish the first iteration from others
  - Distinguish function calls by calling context.



Virtually unrolling the loop once:

- Accesses to *A* and *D* are provably hits after the first iteration
- Accesses to *B* and *C* can still not be classified. Within each execution of the loop, they may only miss once.
  $\longrightarrow$ Persistence Analysis

LRU: May-Analysis: Abstract Domain

- Used to predict *cache misses*.
- Maintains *lower bounds on ages* of memory blocks.
- Lower bound $\geq$ associativity
  $\longrightarrow$ memory block definitely *not* cached.

Example

...and its interpretation:

Abstract state:

{x,y}	age 0
{}	
{s,t}	
{u}	age 3

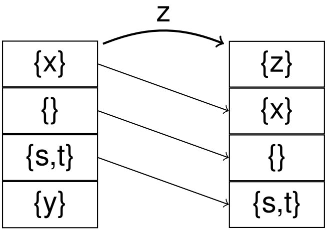
Describes the set of all concrete cache states in which no memory blocks except *x*, *y*, *s*, *t*, and *u* occur,

- *x* and *y* with an age of at least 0,
- *s* and *t* with an age of at least 2,
- *u* with an age of at least 3.

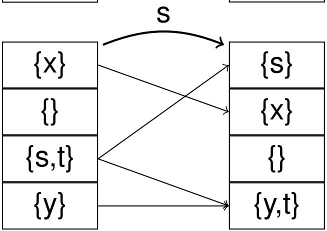
$$\gamma([\{x,y\},\{\},\{s,t\},\{u\}]) = \{[x,y,s,t],[y,x,s,t],[x,y,s,u],\ldots\}$$

LRU: May-Analysis: Update

“Definite Cache Miss”:



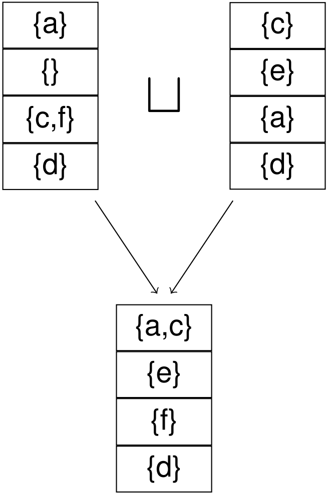
“Potential Cache Hit”:



Why does *t* age in the second case?

LRU: May-Analysis: Join

Need to combine information where control-flow merges.



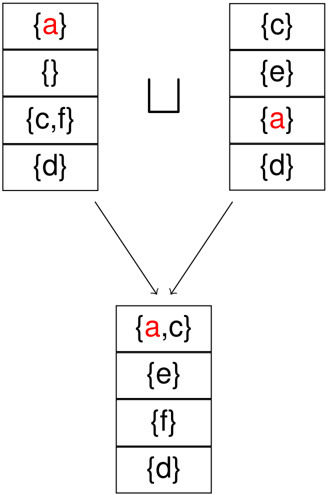
Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Union + Minimal Age"

LRU: May-Analysis: Join

Need to combine information where control-flow merges.



Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Union + Minimal Age"

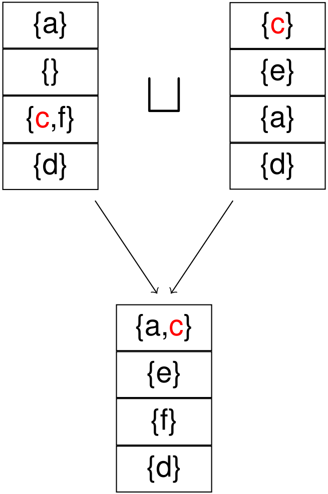
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

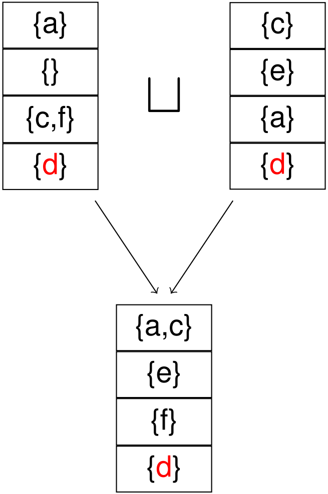
- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Union + Minimal Age"



LRU: May-Analysis: Join

Need to combine information where control-flow merges.



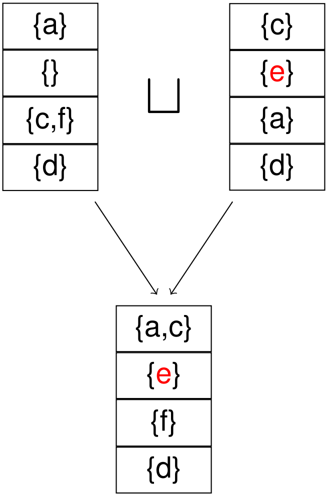
Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

“Union + Minimal Age”

LRU: May-Analysis: Join

Need to combine information where control-flow merges.



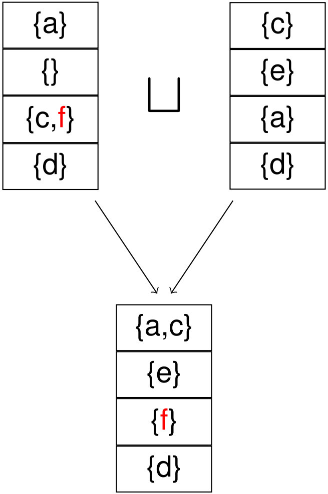
Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Union + Minimal Age"

LRU: May-Analysis: Join

Need to combine information where control-flow merges.



Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

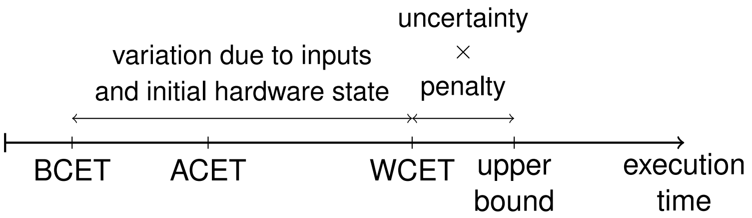
"Union + Minimal Age"

Outline

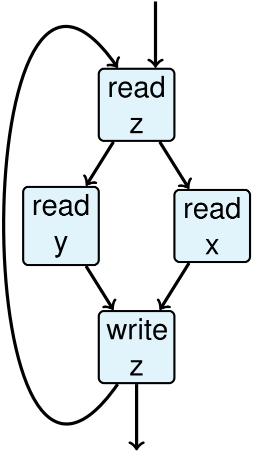
Outline

Uncertainty in WCET Analysis

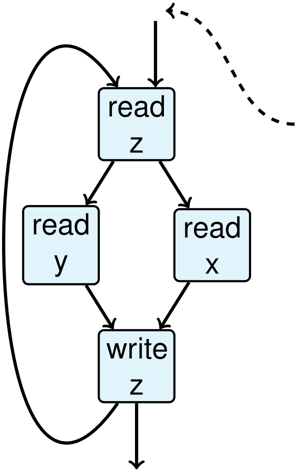
- Amount of uncertainty determines precision of WCET analysis
- Uncertainty in cache analysis depends on replacement policy



Uncertainty in Cache Analysis

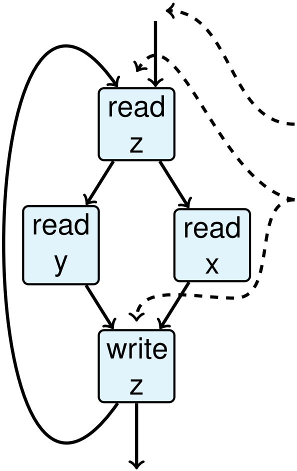


Uncertainty in Cache Analysis



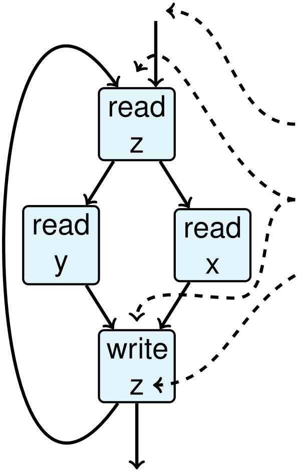
1. Initial cache contents unknown.

Uncertainty in Cache Analysis



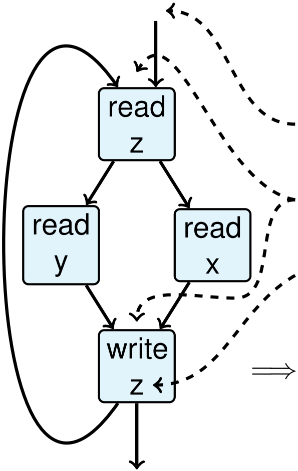
1. Initial cache contents unknown.
2. Need to combine information.

Uncertainty in Cache Analysis



1. Initial cache contents unknown.
2. Need to combine information.
3. Cannot resolve address of *z*.

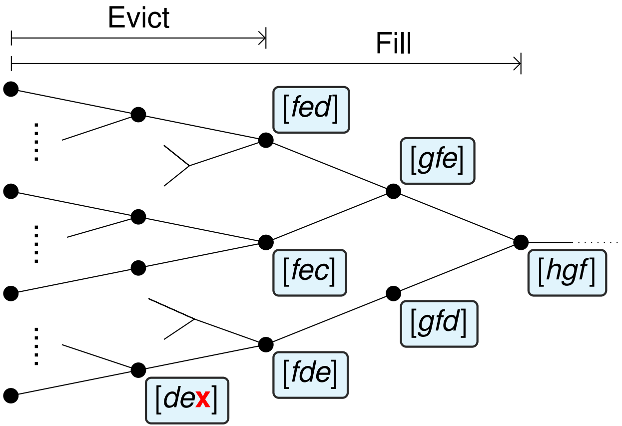
Uncertainty in Cache Analysis



1. Initial cache contents unknown.
2. Need to combine information.
3. Cannot resolve address of *z*.

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

Predictability Metrics



Sequence: $\langle$a, . . . , e, f, g, h$\rangle$

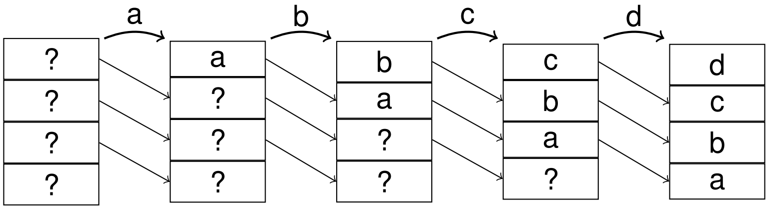
Meaning of Metrics

- Evict
  - Number of accesses to obtain *any may*-information.
  - I.e. when can an analysis predict any cache misses?
- Fill
  - Number of accesses to complete *may*- and *must*-information.
  - I.e. when can an analysis predict each access?

$\longrightarrow$ Evict and Fill bound the precision of *any* static cache analysis.
Can thus serve as a benchmark for analyses.

Evaluation of Least-Recently-Used

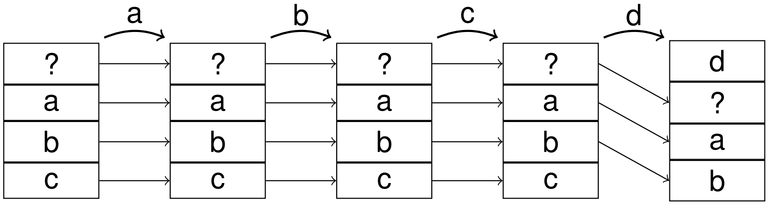
- LRU "forgets" about past quickly:
  - cares about most-recent access to each block only
  - order of previous accesses irrelevant



- In the example: Evict $=$ Fill $=$ 4
- In general: Evict(k) $=$ Fill(k) $=$ k, where k is the associativity of the cache

Evaluation of First-In First-Out (sketch)

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

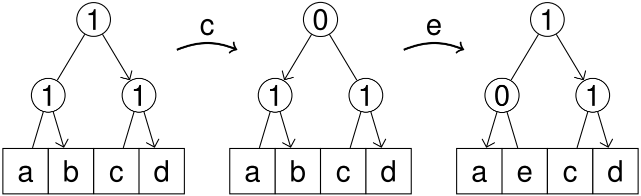


- In the worst-case $k - 1$ hits and k misses: (k = associativity)
  $\longrightarrow \text{Evict}(k) = 2k - 1$
- Another k accesses to obtain complete knowledge:
  $\longrightarrow \text{Fill}(k) = 3k - 1$

Evaluation of Pseudo-LRU (sketch)

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Tree-bits point to block to be replaced



- Accesses “rejuvenate” neighborhood
  - Active blocks keep their (inactive) neighborhood in the cache
- Analysis yields:
  - $\text{Evict}(k) = \frac{k}{2}\log_2 k + 1$
  - $\text{Fill}(k) = \frac{k}{2}\log_2 k + k - 1$

Evaluation of Policies

Policy	Evict(k)	Fill(k)	Evict(8)	Fill(8)
LRU	k	k	8	8
FIFO	$2k - 1$	$3k - 1$	15	23
MRU	$2k - 2$	$\infty/3k - 4$	14	$\infty/20$
PLRU	$\frac{k}{2}\log_2 k + 1$	$\frac{k}{2}\log_2 k + k - 1$	13	19

- LRU is optimal w.r.t. metrics.
- Other policies are much less predictable.

$\longrightarrow$ Use LRU if predictability is a concern.

- How to obtain *may*- and *must*-information within the given limits for other policies?

Outline

Relative Competitiveness

- Competitiveness (Sleator and Tarjan, 1985):
  worst-case performance of an online policy *relative to the optimal offline policy*
  - used to evaluate online policies
- Relative competitiveness (Reineke and Grund, 2008):
  worst-case performance of an online policy *relative to another online policy*
  - used to derive local and global cache analyses

Definition – Relative Miss-Competitiveness

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Notation

$$m_{\mathbf{P}}(p, s) \quad = \quad \textit{number of misses that policy } \mathbf{P} \textit{ incurs on access sequence } s \in M^* \textit{ starting in state } p \in C^{\mathbf{P}}$$

Definition – Relative Miss-Competitiveness

Notation

$m_{\mathbf{P}}(p, s)$ = *number of misses that policy* **P** *incurs on access sequence* $s \in M^*$ *starting in state* $p \in C^{\mathbf{P}}$

Definition (Relative miss competitiveness)

Policy **P** is (k, c)-miss-competitive relative to policy **Q** if

$$m_{\mathbf{P}}(p, s) \leq k \cdot m_{\mathbf{Q}}(q, s) + c$$

for all access sequences $s \in M^*$ and cache-set states $p \in C^{\mathbf{P}}, q \in C^{\mathbf{Q}}$ that are compatible $p \sim q$.

Definition – Relative Miss-Competitiveness

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Notation

$$m_{\mathbf{P}}(p, s) \quad = \quad \textit{number of misses that policy } \mathbf{P} \textit{ incurs on access sequence } s \in M^* \textit{ starting in state } p \in C^{\mathbf{P}}$$

Definition (Relative miss competitiveness)

Policy **P** is (k, c)-miss-competitive relative to policy **Q** if

$$m_{\mathbf{P}}(p, s) \leq k \cdot m_{\mathbf{Q}}(q, s) + c$$

for all access sequences $s \in M^*$ and cache-set states $p \in C^{\mathbf{P}}, q \in C^{\mathbf{Q}}$ that are compatible $p \sim q$.

Definition (Competitive miss ratio of **P** relative to **Q**)

The smallest k, s.t. **P** is (k, c)-miss-competitive rel. to **Q** for some c.

Example – Relative Miss-Competitiveness

P is $(3,4)$-miss-competitive relative to **Q**.
If **Q** incurs x misses, then **P** incurs at most $3 \cdot x + 4$ misses.

Example – Relative Miss-Competitiveness

P is $(3, 4)$-miss-competitive relative to **Q**.
If **Q** incurs x misses, then **P** incurs at most $3 \cdot x + 4$ misses.

Best: **P** is $(1, 0)$-miss-competitive relative to **Q**.

Example – Relative Miss-Competitiveness

P is $(3, 4)$-miss-competitive relative to **Q**.
If **Q** incurs x misses, then **P** incurs at most $3 \cdot x + 4$ misses.

Best: **P** is $(1, 0)$-miss-competitive relative to **Q**.

Worst: **P** is not-miss-competitive (or ∞-miss-competitive) relative to **Q**.

Example – Relative Hit-Competitiveness

P is $(\frac{2}{3}, 3)$-hit-competitive relative to **Q**.
If **Q** has x hits, then **P** has at least $\frac{2}{3} \cdot x - 3$ hits.

Example – Relative Hit-Competitiveness

P is $(\frac{2}{3}, 3)$-hit-competitive relative to **Q**.
If **Q** has x hits, then **P** has at least $\frac{2}{3} \cdot x - 3$ hits.

Best: **P** is $(1, 0)$-hit-competitive relative to **Q**.
Equivalent to $(1, 0)$-miss-competitiveness.

Example – Relative Hit-Competitiveness

P is $(\frac{2}{3}, 3)$-hit-competitive relative to **Q**.
If **Q** has x hits, then **P** has at least $\frac{2}{3} \cdot x - 3$ hits.

Best: **P** is $(1, 0)$-hit-competitive relative to **Q**.
Equivalent to $(1, 0)$-miss-competitiveness.

Worst: **P** is $(0, 0)$-hit-competitive relative to **Q**.
Analogue to ∞-miss-competitiveness.

Local Guarantees: $(1, 0)$-Competitiveness

Let **P** be $(1, 0)$-competitive relative to **Q**:

$$m_{\mathbf{P}}(p, s) \leq 1 \cdot m_{\mathbf{Q}}(q, s) + 0$$
$$\Leftrightarrow m_{\mathbf{P}}(p, s) \leq m_{\mathbf{Q}}(q, s)$$

Local Guarantees: $(1, 0)$-Competitiveness

Let **P** be $(1, 0)$-competitive relative to **Q**:

$$m_{\mathbf{P}}(p, s) \leq 1 \cdot m_{\mathbf{Q}}(q, s) + 0$$
$$\Leftrightarrow m_{\mathbf{P}}(p, s) \leq m_{\mathbf{Q}}(q, s)$$

1. If **Q** “hits”, so does **P**, and
2. if **P** “misses”, so does **Q**.

Local Guarantees: $(1,0)$-Competitiveness

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Let **P** be $(1,0)$-competitive relative to **Q**:

$$m_{\mathbf{P}}(p,s) \leq 1 \cdot m_{\mathbf{Q}}(q,s) + 0$$
$$\Leftrightarrow m_{\mathbf{P}}(p,s) \leq m_{\mathbf{Q}}(q,s)$$

1. If **Q** "hits", so does **P**, and
2. if **P** "misses", so does **Q**.

As a consequence,

1. a *must*-analysis for **Q** is also a *must*-analysis for **P**, and
2. a *may*-analysis for **P** is also a *may*-analysis for **Q**.

Global Guarantees: (k, c)-Competitiveness

Given: Global guarantees for policy **Q**.
Wanted: Global guarantees for policy **P**.

Global Guarantees: (k, c)-Competitiveness

Given: Global guarantees for policy **Q**.
Wanted: Global guarantees for policy **P**.

1. Determine competitiveness of policy **P** relative to policy **Q**.

$$\mathbf{m_P \leq k \cdot m_Q + c}$$

Global Guarantees: (k, c)-Competitiveness

Given: Global guarantees for policy **Q**.
Wanted: Global guarantees for policy **P**.

1. Determine competitiveness of policy **P** relative to policy **Q**.

   $$\mathbf{m_P \leq k \cdot m_Q + c}$$

2. Compute global guarantee for task T under policy **Q**.

   $$\mathbf{m_Q(T)}$$

Global Guarantees: (k, c)-Competitiveness

Given: Global guarantees for policy **Q**.
Wanted: Global guarantees for policy **P**.

1. Determine competitiveness of policy **P** relative to policy **Q**.

   $\mathbf{m_P} \leq \mathbf{k} \cdot \mathbf{m_Q} + \mathbf{c}$

2. Compute global guarantee for task T under policy **Q**.

   $\mathbf{m_Q(T)}$

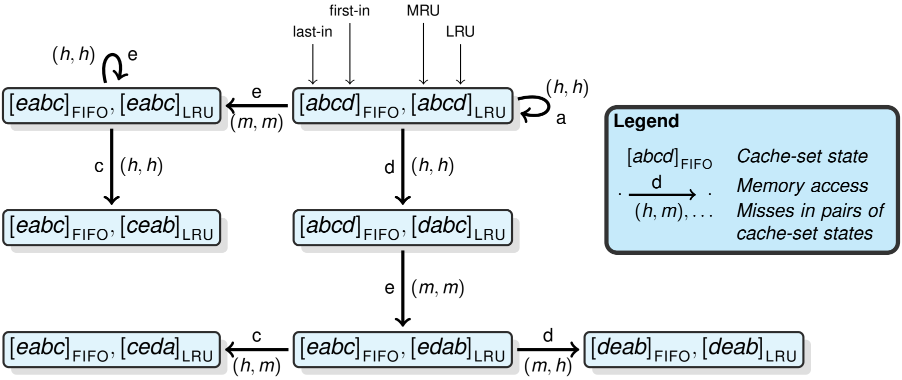
3. Calculate global guarantee on the number of misses for **P** using the global guarantee for **Q** and the competitiveness results of **P** relative to **Q**.

   $\mathbf{m_P} \leq \mathbf{k} \cdot \mathbf{m_Q} + \mathbf{c} \quad \mathbf{m_Q(T)} \quad = \quad \mathbf{m_P(T)}$

Relative Competitiveness: Automatic Computation

SAARLAND UNIVERSITY
COMPUTER SCIENCE

P and **Q** (here: FIFO and LRU) induce transition system:



Competitive miss ratio = maximum ratio of misses in policy **P** to misses in policy **Q** in transition system

Transition System is ∞ Large

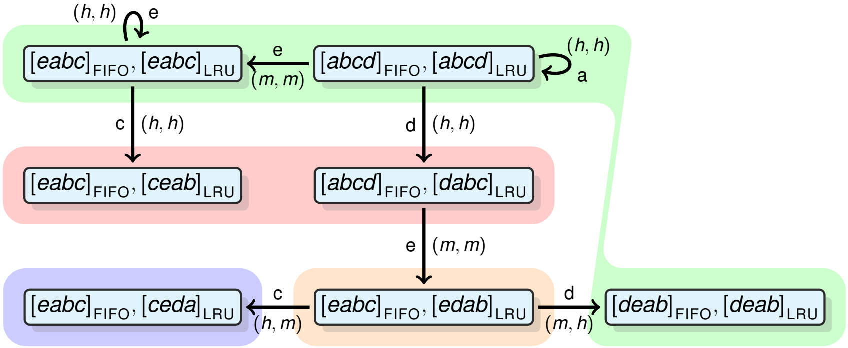
Problem: The induced transition system is ∞ large.

Observation: Only the *relative positions* of elements matter:

$$
\begin{tikzcd}
{[abc]_{\mathrm{LRU}}, [bde]_{\mathrm{FIFO}}} \arrow[r, phantom, "\approx"] \arrow[d, "c"', "{(h, m)}"] & {[fgl]_{\mathrm{LRU}}, [ghm]_{\mathrm{FIFO}}} \arrow[d, "l"', "{(h, m)}"] \\
{[cab]_{\mathrm{LRU}}, [cbd]_{\mathrm{FIFO}}} \arrow[r, phantom, "\approx"] & {[lfg]_{\mathrm{LRU}}, [lgh]_{\mathrm{FIFO}}}
\end{tikzcd}
$$

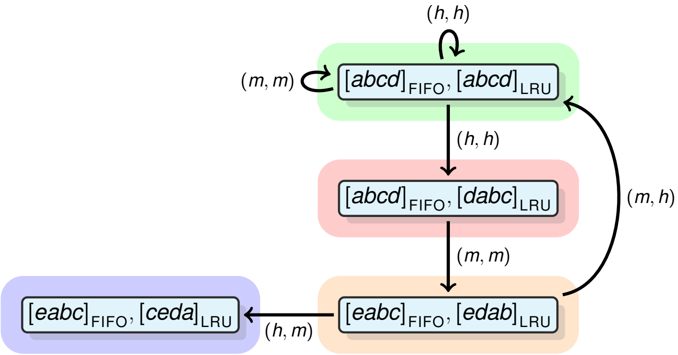
Solution: Construct *finite* quotient transition system.

≈-Equivalent States in Running Example



Finite Quotient Transition System

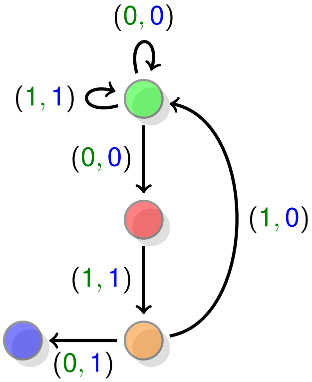
Merging $\approx$-equivalent states yields a finite quotient transition system:



Competitive Ratio = Maximum Cycle Ratio

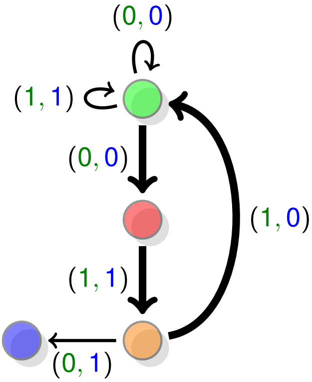
SAARLAND UNIVERSITY
COMPUTER SCIENCE

Competitive miss ratio =
maximum ratio of misses in policy **P** to misses in policy **Q**



Competitive Ratio = Maximum Cycle Ratio

Competitive miss ratio =
maximum ratio of misses in policy **P** to misses in policy **Q**



Maximum cycle ratio = $\frac{0+1+1}{0+1+0} = 2$

Tool Implementation

- Implemented in Java, called Relacs
- Interface for replacement policies

- Fully automatic
- Provides example sequences for competitive ratio and constant

- Analysis usually practically feasible up to associativity 8
  - limited by memory consumption
  - depends on similarity of replacement policies

Online version:

`http://rw4.cs.uni-sb.de/~reineke/relacs`

Generalizations

Identified patterns and proved generalizations by hand.

Aided by example sequences generated by tool.

Generalizations

Identified patterns and proved generalizations by hand.
Aided by example sequences generated by tool.

Previously unknown facts:

$\text{PLRU}(k)$ is $(1, 0)$ comp. rel. to $\text{LRU}(1 + log_2 k)$,
$\longrightarrow$ LRU-*must*-analysis can be used for PLRU

Generalizations

Identified patterns and proved generalizations by hand.
Aided by example sequences generated by tool.

Previously unknown facts:

$\text{PLRU}(k)$ is $(1,0)$ comp. rel. to $\text{LRU}(1+log_2 k)$,

$\longrightarrow$ LRU-*must*-analysis can be used for PLRU

$\text{FIFO}(k)$ is $(\frac{1}{2}, \frac{k-1}{2})$ hit-comp. rel. to $\text{LRU}(k)$, whereas

$\text{LRU}(k)$ is $(0,0)$ hit-comp. rel. to $\text{FIFO}(k)$, but

Generalizations

Identified patterns and proved generalizations by hand.
Aided by example sequences generated by tool.

Previously unknown facts:

PLRU(k) is $(1, 0)$ comp. rel. to LRU($1 + log_2 k$),
$\longrightarrow$ LRU-*must*-analysis can be used for PLRU

FIFO(k) is $(\frac{1}{2}, \frac{k-1}{2})$ hit-comp. rel. to LRU(k), whereas
LRU(k) is $(0, 0)$ hit-comp. rel. to FIFO(k), but

LRU($2k - 1$) is $(1, 0)$ comp. rel. to FIFO(k), and
LRU($2k - 2$) is $(1, 0)$ comp. rel. to MRU(k).
$\longrightarrow$ LRU-*may*-analysis can be used for FIFO and MRU
$\longrightarrow$ optimal with respect to predictability metric Evict

Generalizations

Identified patterns and proved generalizations by hand.
Aided by example sequences generated by tool.

Previously unknown facts:

$$\mathrm{PLRU}(k) \text{ is } (1,0) \text{ comp. rel. to } \mathrm{LRU}(1+log_2 k),$$

$\longrightarrow$ LRU-*must*-analysis can be used for PLRU

$$\mathrm{FIFO}(k) \text{ is } (\tfrac{1}{2}, \tfrac{k-1}{2}) \text{ hit-comp. rel. to } \mathrm{LRU}(k), \text{ whereas}$$
$$\mathrm{LRU}(k) \text{ is } (0,0) \text{ hit-comp. rel. to } \mathrm{FIFO}(k), \text{ but}$$

$$\mathrm{LRU}(2k-1) \text{ is } (1,0) \text{ comp. rel. to } \mathrm{FIFO}(k), \text{ and}$$
$$\mathrm{LRU}(2k-2) \text{ is } (1,0) \text{ comp. rel. to } \mathrm{MRU}(k).$$

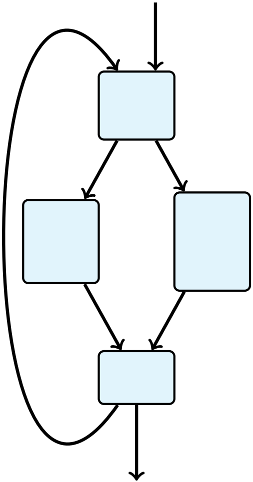
$\longrightarrow$ LRU-*may*-analysis can be used for FIFO and MRU
$\longrightarrow$ optimal with respect to predictability metric Evict

FIFO-*may*-analysis used in the analysis of the branch target buffer of the MOTOROLA POWERPC 56X.

Outline

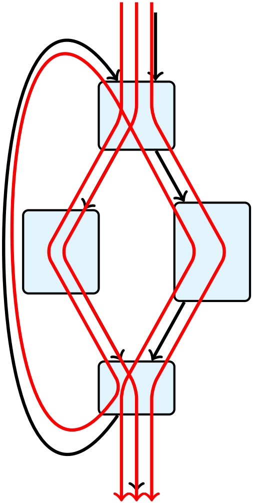
Measurement-Based Timing Analysis

- Run program on a number of inputs and initial states.
- Combine measurements for basic blocks to obtain WCET estimation.
- Sensitivity Analysis demonstrates this approach may be dramatically wrong.

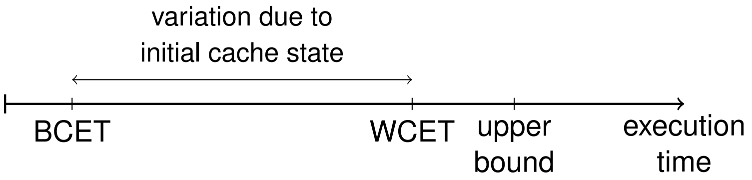


Measurement-Based Timing Analysis

- Run program on a number of inputs and initial states.
- Combine measurements for basic blocks to obtain WCET estimation.
- Sensitivity Analysis demonstrates this approach may be dramatically wrong.



Influence of Initial Cache State



Definition (Miss sensitivity)

Policy **P** is (k, c)-miss-sensitive if

$$m_{\mathbf{P}}(q, s) \leq k \cdot m_{\mathbf{P}}(q', s) + c$$

for all access sequences $s \in M^*$ and cache-set states $q, q' \in C^{\mathbf{P}}$.

Sensitivity Results

Policy	2	3	4	5	6	7	8
LRU	$1,2$	$1,3$	$1,4$	$1,5$	$1,6$	$1,7$	$1,8$
FIFO	$2,2$	$3,3$	$4,4$	$5,5$	$6,6$	$7,7$	$8,8$
PLRU	$1,2$	$-$	∞	$-$	$-$	$-$	∞
MRU	$1,2$	$3,4$	$5,6$	$7,8$	MEM	MEM	MEM

- LRU is optimal. Performance varies in the least possible way.
- For FIFO, PLRU, and MRU the number of misses may vary strongly.
- Case study based on simple model of execution time by Hennessy and Patterson (2003):
  WCET may be 3 times higher than a measured execution time for 4-way FIFO.

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Summary

Cache Analysis for Least-Recently-Used

. . . efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

. . . requires context-sensitivity for precision.

Summary

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Cache Analysis for Least-Recently-Used

...efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

...requires context-sensitivity for precision.

Predictability Metrics

...quantify the predictability of replacement policies.

$\longrightarrow$ LRU is the most predictable policy.

Summary

Cache Analysis for Least-Recently-Used

...efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

...requires context-sensitivity for precision.

Predictability Metrics

...quantify the predictability of replacement policies.

$\longrightarrow$ LRU is the most predictable policy.

Relative Competitiveness

...allows to derive guarantees on cache performance,

...yields first *may*-analyses for FIFO and MRU.

Summary

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Cache Analysis for Least-Recently-Used

…efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

…requires context-sensitivity for precision.

Predictability Metrics

…quantify the predictability of replacement policies.

$\longrightarrow$ LRU is the most predictable policy.

Relative Competitiveness

…allows to derive guarantees on cache performance,

…yields first *may*-analyses for FIFO and MRU.

Sensitivity Analysis

…determines the influence of initial state on cache performance.

Summary

Cache Analysis for Least-Recently-Used

...efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

...requires context-sensitivity for precision.

Predictability Metrics

...quantify the predictability of replacement policies.

$\longrightarrow$ LRU is the most predictable policy.

Relative Competitiveness

...allows to derive guarantees on cache performance,

...yields first *may*-analyses for FIFO and MRU.

Sensitivity Analysis

...determines the influence of initial state on cache performance.

Thank you for your attention!

Summary

Cache Analysis for Least-Recently-Used

... efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

... requires context-sensitivity for precision.

Predictability Metrics

... quantify the predictability of replacement policies.

$\longrightarrow$ LRU is the most predictable policy.

Relative Competitiveness

... allows to derive guarantees on cache performance,

... yields first *may*-analyses for FIFO and MRU.

Sensitivity Analysis

... determines the influence of initial state on cache performance.

Thank you for your attention!

Cousot, P. and Cousot, R. (1977).
Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In *POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, New York, NY, USA. ACM Press.

Ferdinand, C. and Wilhelm, R. (1999).
Efficient and precise cache behavior prediction for real-time systems.
Real-Time Systems, 17(2-3):131–181.

Reineke, J. and Grund, D. (2008a).
Relative competitive analysis of cache replacement policies.
In *LCTES '08: Proceedings of the 2008 ACM SIGPLAN-SIGBED conference on Languages, compilers, and tools for embedded systems*, pages 51–60, New York, NY, USA. ACM.

Reineke, J. and Grund, D. (2008b).
Sensitivity of cache replacement policies.

Reports of SFB/TR 14 AVACS 36, SFB/TR 14 AVACS.
ISSN: 1860-9821, http://www.avacs.org.

Reineke, J., Grund, D., Berg, C., and Wilhelm, R. (2007).
Timing predictability of cache replacement policies.
Real-Time Systems, 37(2):99–122.

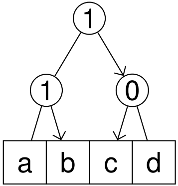
Most-Recently-Used – MRU

MRU-bits record whether line was recently used

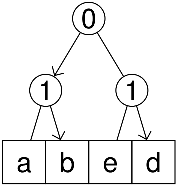
$$\begin{tikzcd}
{[abcd]_{0101}} \arrow[loop right, "b{,}d"] \arrow[d, bend right, "e"'] \\
{[ebcd]_{1101}} \arrow[loop right, "e{,}b{,}d"] \arrow[d, bend right, "c"'] \\
{[ebcd]_{0010}} \arrow[loop right, "c"]
\end{tikzcd}$$

$\longrightarrow$ Never converges

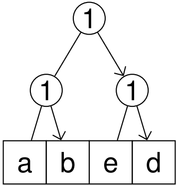
Pseudo-LRU – PLRU



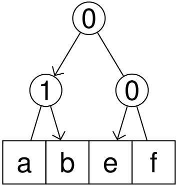
Initial cache-set state $[a, b, c, d]_{110}$.



After a miss on *e*. State: $[a, b, e, d]_{011}$.



After a hit on *a*. State: $[a, b, e, d]_{111}$.



After a miss on *f*. State: $[a, b, e, f]_{010}$.

Hit on *a* “rejuvenates” neighborhood; “saves” *b* from eviction.

May- and Must-Information

$$
\begin{aligned}
May^{\mathbf{P}}(s) &:= \bigcup_{p \in C^{\mathbf{P}}} CC_{\mathbf{P}}(update_{\mathbf{P}}(p, s)) \\
Must^{\mathbf{P}}(s) &:= \bigcap_{p \in C^{\mathbf{P}}} CC_{\mathbf{P}}(update_{\mathbf{P}}(p, s))
\end{aligned}
$$

$$
\begin{aligned}
may^{\mathbf{P}}(n) &:= \left| May^{\mathbf{P}}(s) \right|, \text{where } s \in S^{\neq} \subsetneq M^*, |s| = n \\
must^{\mathbf{P}}(n) &:= \left| Must^{\mathbf{P}}(s) \right|, \text{where } s \in S^{\neq} \subsetneq M^*, |s| = n
\end{aligned}
$$

$S^{\neq}$: set of finite access sequences with pairwise different accesses

Definitions of Metrics

$$\begin{aligned} \mathrm{Evict}^{\mathbf{P}} &:= \min\left\{n \mid may^{\mathbf{P}}(n) \leq n\right\}, \\ \mathrm{Fill}^{\mathbf{P}} &:= \min\left\{n \mid must^{\mathbf{P}}(n) = k\right\}, \end{aligned}$$

where k is **P**'s associativity.

Relation: Pred. Metrics $\leftrightarrow$ Rel. Competitiveness

Let $P(k)$ be $(1, 0)$-miss-competitive relative to policy $Q(l)$, then

(i) $Evict^P(k) \geq Evict^Q(l)$,

(ii) $mls^P(k) \geq mls^Q(l)$.

Alternative Pred. Metrics $\leftrightarrow$ Rel. Competitiveness

Let l be the smallest associativity, such that $\mathrm{LRU}(l)$ is $(1,0)$-miss-competitive relative to $P(k)$. Then

$$\text{Alt-Evict}^{P}(k) = l.$$

Let l be the greatest associativity, such that $P(k)$ is $(1,0)$-miss-competitive relative to $\mathrm{LRU}(l)$. Then

$$\text{Alt-mls}^{P}(k) = l.$$

Size of Transition System

SAARLAND UNIVERSITY
COMPUTER SCIENCE

$$\underbrace{2^{l+l'}}_{\substack{\text{status bits}\\ \text{of } \mathbf{P} \text{ and } \mathbf{Q}}} \cdot \underbrace{\sum_{i=0}^{k} \binom{k}{i}}_{\text{non-empty lines in } \mathbf{P}} \cdot \underbrace{\sum_{i'=0}^{k'} \binom{k'}{i'}}_{\text{non-empty lines in } \mathbf{Q}} \cdot \underbrace{\sum_{j=0}^{\min\{i,i'\}} \binom{i}{j}\binom{i'}{j} j!}_{\substack{\text{number of overlappings}\\ \text{in non-empty lines}}}$$

$$\begin{aligned} \sum_{j=0}^{\min\{k,k'\}} \binom{k}{j}\binom{k'}{j} j! &\leq k! \cdot k'! \sum_{j=0}^{\min\{k,k'\}} \frac{1}{(k-j)!j!(k'-j)!} \\ &\leq k! \cdot k'! \sum_{j=0}^{\infty} \frac{1}{j!} = e \cdot k! \cdot k'! \end{aligned}$$

This can be bounded by

$$2^{l+l'+k+k'} \leq |(C_k^l \times C_{k'}^{l'})/\approx| \leq 2^{l+l'+k+k'} \cdot \underbrace{e \cdot k! \cdot k'!}_{\text{bound on number of overlappings}}$$

Compatible States

$$
\begin{tikzcd}
\boxed{i^{\mathbf{P}} = [\bot\bot\bot\bot]_{\mathbf{P}}} \arrow[d, squiggly, "update_{\mathbf{P}}(i^{\mathbf{P}}, s)"] \arrow[r, phantom, "\approx"] & \boxed{i^{\mathbf{Q}} = [\bot\bot\bot\bot]_{\mathbf{Q}}} \arrow[d, squiggly, "update_{\mathbf{Q}}(i^{\mathbf{Q}}, s)"] \\
\boxed{p} \arrow[r, phantom, "\approx"] & \boxed{q}
\end{tikzcd}
$$

$(1, 0)$-Competitiveness and May/Must-Analyses

Let **P** be $(1, 0)$-competitive relative to **Q**, then

$$\begin{tikzcd}
\boxed{p} \arrow[d, "m_{\mathbf{P}}(p, \langle x \rangle) = 1"'] \arrow[rr, phantom, "\approx"] \arrow[drr, phantom, "\Longrightarrow"] & & \boxed{q} \arrow[d, "m_{\mathbf{Q}}(q, \langle x \rangle) = 1"] \\
\boxed{p'} \arrow[rr, phantom, "\approx"] & & \boxed{q'}
\end{tikzcd}$$

$(1,0)$-Competitiveness and May/Must-Analyses

$$
\begin{tikzcd}
C^{\mathbf{P}} \arrow[r, phantom, "\approx"] \arrow[d, "S"'] & C^{\mathbf{Q}} \arrow[d, "S"] \\
P \arrow[r, phantom, "\approx"] \arrow[d, "\forall p \in P : m_{\mathbf{P}}(p, \langle x \rangle) = 1"'] \arrow[dr, phantom, "\Longrightarrow"] & Q \arrow[d, "\forall q \in Q : m_{\mathbf{Q}}(q, \langle x \rangle) = 1"] \\
P' \arrow[r, phantom, "\approx"] & Q'
\end{tikzcd}
$$

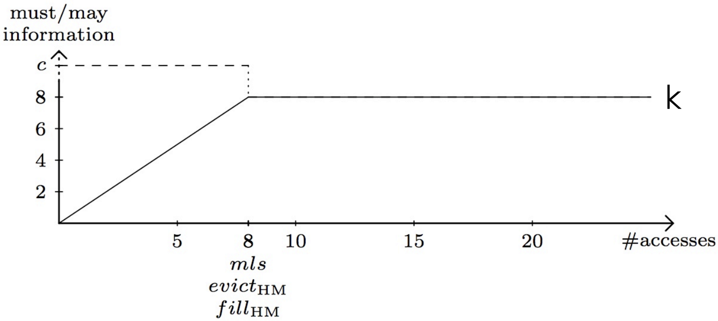
Case Study: Impact of Sensitivity

- Simple model of execution time from Hennessy & Patterson (2003)
- CPI_{hit} = Cycles per instruction assuming cache hits only
- $\frac{\text{Memory accesses}}{\text{Instruction}}$ including instruction and data fetches

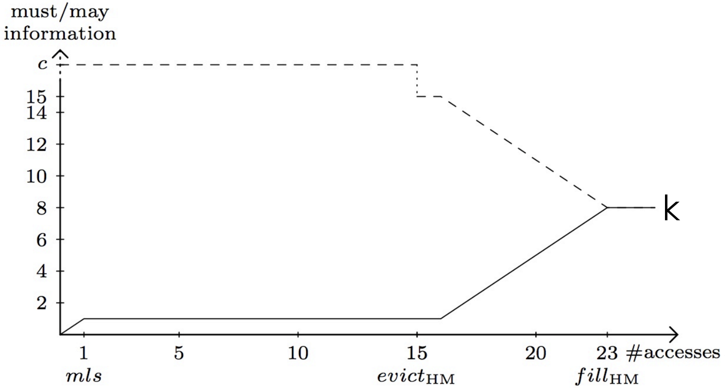
$$\frac{T_{wc}}{T_{meas}} = \frac{\text{CPI}_{hit} + \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate}_{wc} \times \text{Miss penalty}}{\text{CPI}_{hit} + \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate}_{meas} \times \text{Miss penalty}}$$

$$= \frac{1.5 + 1.2 \times 0.20 \times 50}{1.5 + 1.2 \times 0.05 \times 50} = \frac{13.5}{4.5} = 3$$

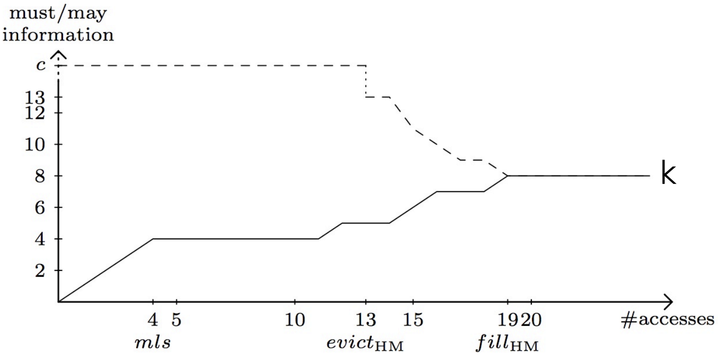
Evolution of May- and Must-Information for LRU



Evolution of May- and Must-Information for FIFO



Evolution of May- and Must-Information for PLRU



Evolution of May- and Must-Information for MRU