

Caches in WCET Analysis

Jan Reineke

Department of Computer Science
Saarland University
Saarbrücken, Germany

ARTIST Summer School in Europe 2009
Autrans, France
September 7-11, 2009

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

1 Caches

2 Cache Analysis for Least-Recently-Used

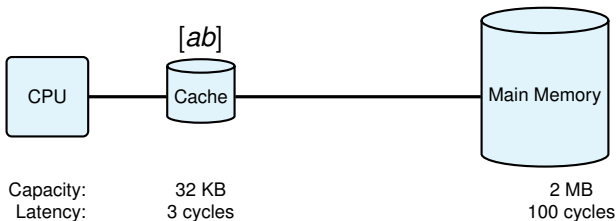
3 Beyond Least-Recently-Used

- Predictability Metrics
- Relative Competitiveness
- Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary

■ How they work:

- ▶ dynamically
- ▶ managed by replacement policy

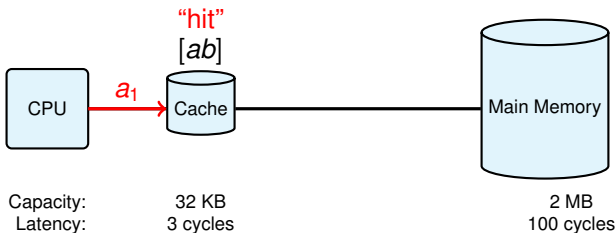


■ Why they work: *principle of locality*

- ▶ spatial
- ▶ temporal

■ How they work:

- ▶ dynamically
- ▶ managed by replacement policy

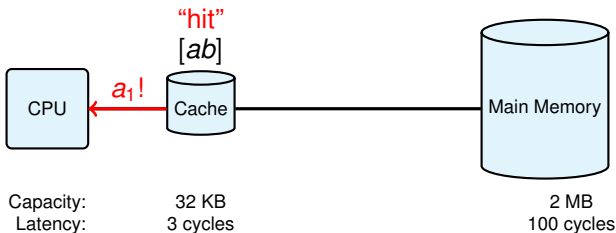


■ Why they work: *principle of locality*

- ▶ spatial
- ▶ temporal

■ How they work:

- ▶ dynamically
- ▶ managed by replacement policy

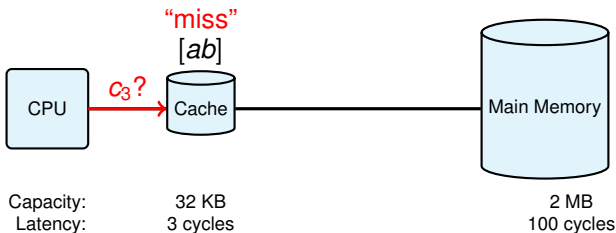


■ Why they work: *principle of locality*

- ▶ spatial
- ▶ temporal

■ How they work:

- ▶ dynamically
- ▶ managed by replacement policy

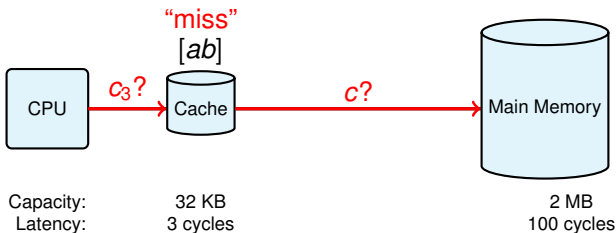


■ Why they work: *principle of locality*

- ▶ spatial
- ▶ temporal

■ How they work:

- ▶ dynamically
- ▶ managed by replacement policy

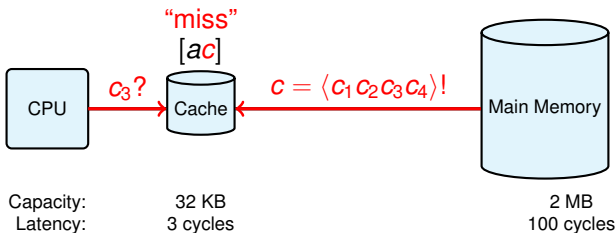


■ Why they work: *principle of locality*

- ▶ spatial
- ▶ temporal

■ How they work:

- ▶ dynamically
- ▶ managed by replacement policy

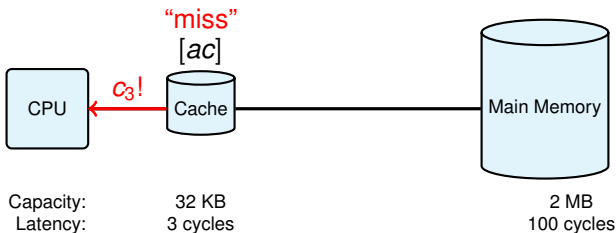


■ Why they work: *principle of locality*

- ▶ spatial
- ▶ temporal

■ How they work:

- ▶ dynamically
- ▶ managed by replacement policy

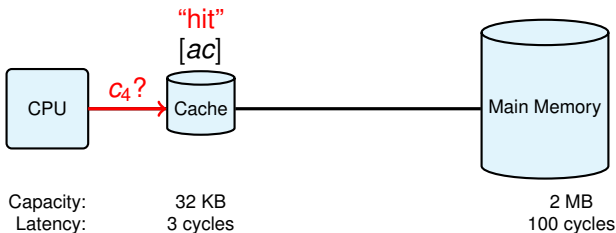


■ Why they work: *principle of locality*

- ▶ spatial
- ▶ temporal

■ How they work:

- ▶ dynamically
- ▶ managed by replacement policy

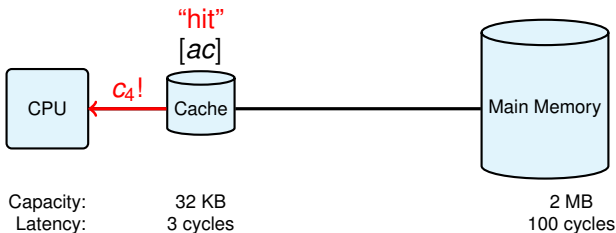


■ Why they work: *principle of locality*

- ▶ spatial
- ▶ temporal

■ How they work:

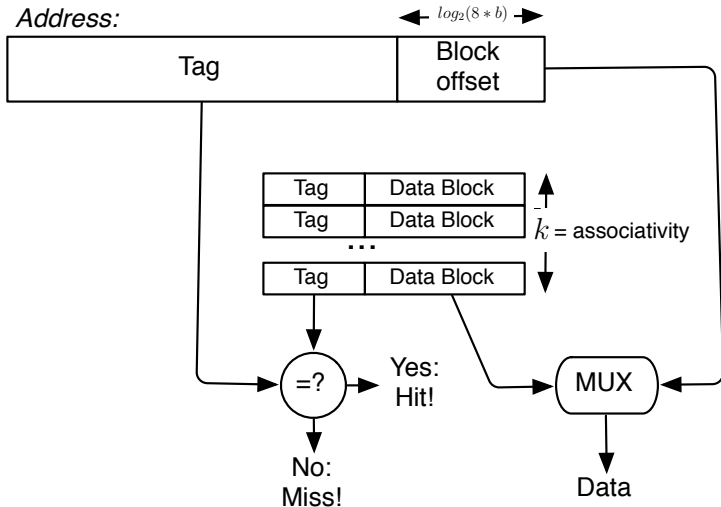
- ▶ dynamically
- ▶ managed by replacement policy

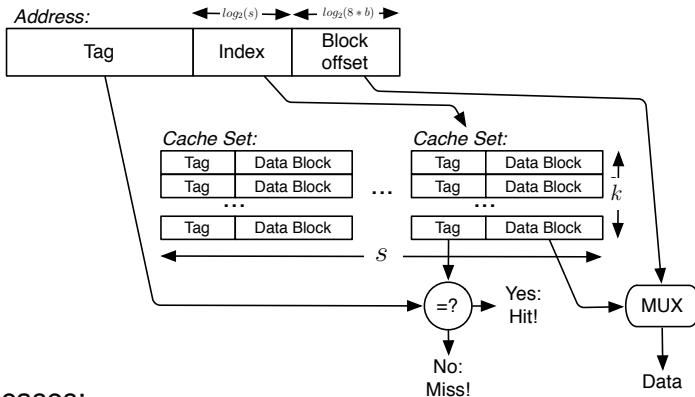


■ Why they work: *principle of locality*

- ▶ spatial
- ▶ temporal

Fully-Associative Caches





Special cases:

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

- 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.

1 Caches

2 Cache Analysis for Least-Recently-Used

3 Beyond Least-Recently-Used

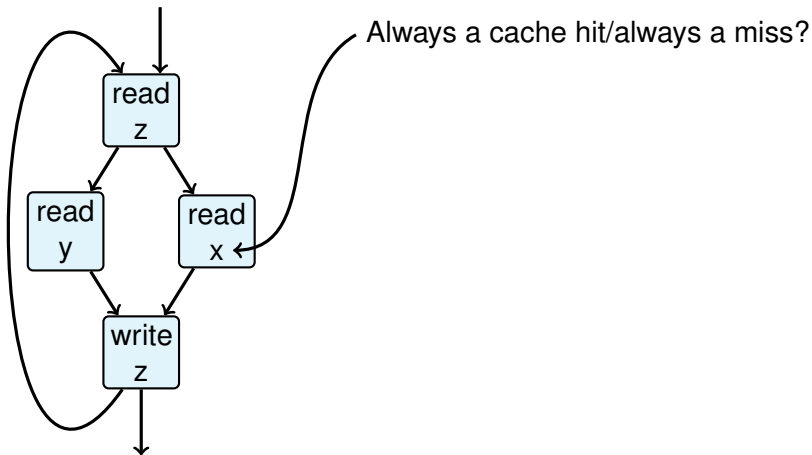
- Predictability Metrics
- Relative Competitiveness
- Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary

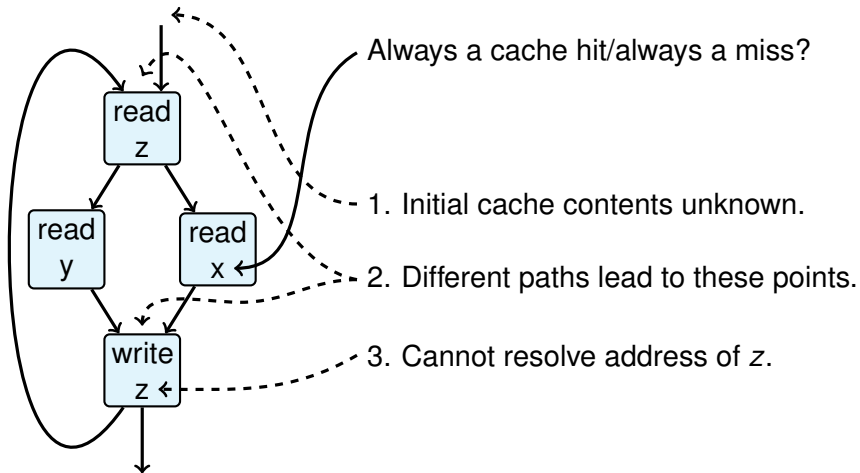
Two types of cache analyses:

- 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, ...

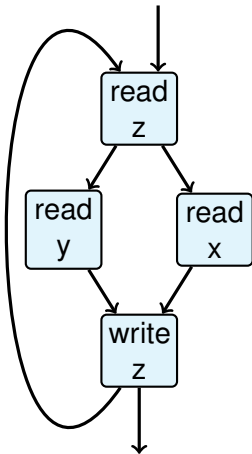
Challenges for Cache Analysis



Challenges for Cache Analysis



Deriving Invariants about Cache States using Abstract Interpretation

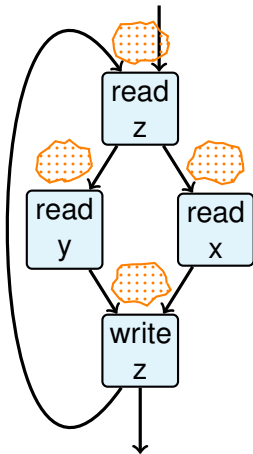


Collecting Semantics =
set of states at each program point that
any execution may encounter there

Two approximations:

Collecting Semantics	uncomputable
\subseteq Cache Semantics	computable
$\subseteq \gamma(\text{Abstract Cache Sem.})$	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

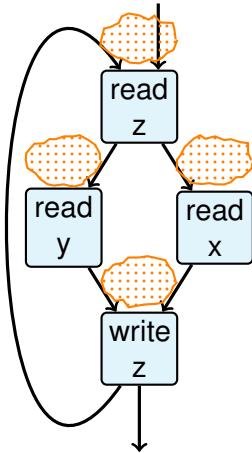
Two approximations:

Collecting Semantics uncomputable

\subseteq Cache Semantics computable

$\subseteq \gamma(\text{Abstract Cache Sem.})$ efficiently
computable

Deriving Invariants about Cache States using Abstract Interpretation

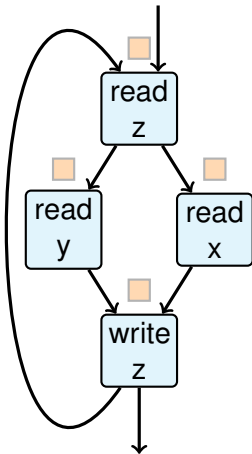


Collecting Semantics =
set of states at each program point that
any execution may encounter there

Two approximations:

Collecting Semantics	uncomputable
\subseteq Cache Semantics	computable
$\subseteq \gamma(\text{Abstract Cache Sem.})$	efficiently computable

Deriving Invariants about Cache States using Abstract Interpretation

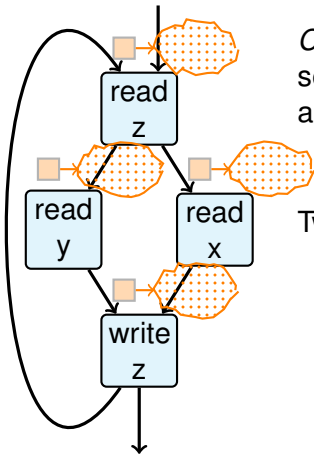


Collecting Semantics =
set of states at each program point that
any execution may encounter there

Two approximations:

Collecting Semantics	uncomputable
\subseteq Cache Semantics	computable
$\subseteq \gamma(\text{Abstract Cache Sem.})$	efficiently computable

Deriving Invariants about Cache States using Abstract Interpretation



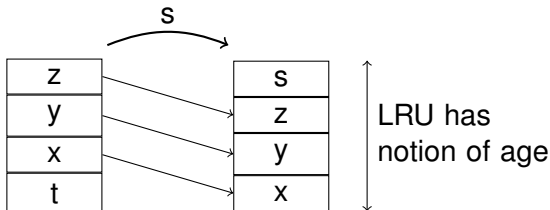
Collecting Semantics =
set of states at each program point that
any execution may encounter there

Two approximations:

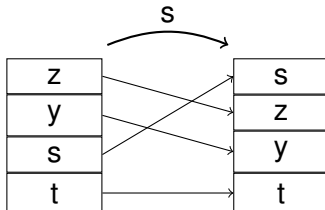
Collecting Semantics	uncomputable
\subseteq Cache Semantics	computable
$\subseteq \gamma(\text{Abstract Cache Sem.})$	efficiently computable

Least-Recently-Used (LRU): Concrete Behavior

“Cache Miss”:



“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

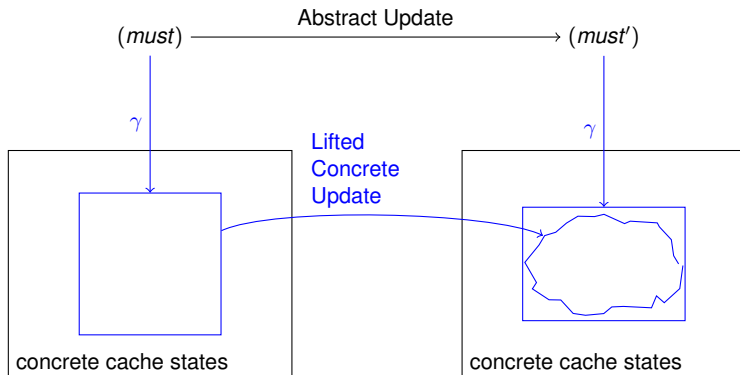
... and its interpretation:

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.

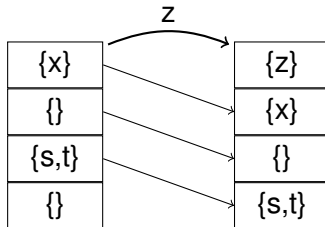
$$\gamma([\{x\}, \{\}, \{s, t\}, \{\}]) = \{[x, s, t, a], [x, t, s, a], [x, s, t, b], \dots\}$$

Sound Update – Local Consistency

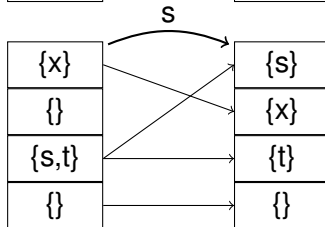


LRU: Must-Analysis: Update

“Potential Cache Miss”:



“Definite Cache Hit”:



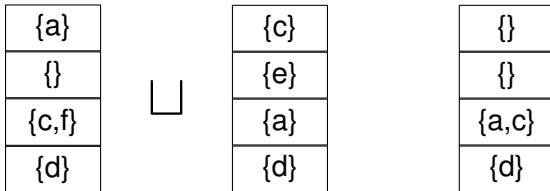
Why does t not age in the second case?

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



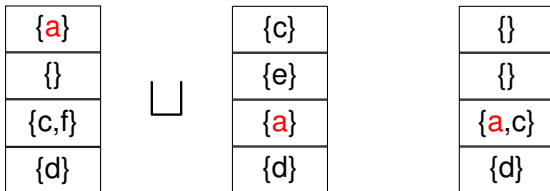
“Intersection + Maximal Age”

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



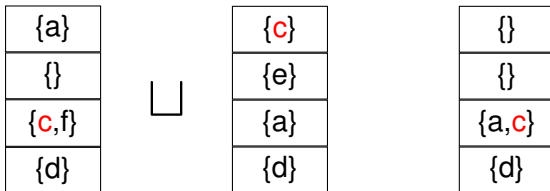
“Intersection + Maximal Age”

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



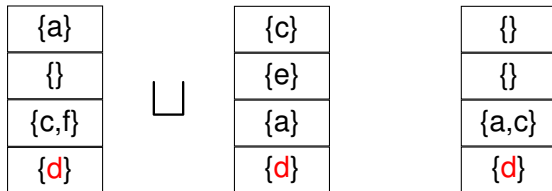
“Intersection + Maximal Age”

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



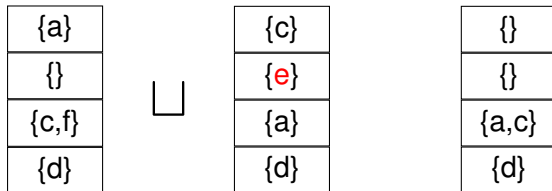
“Intersection + Maximal Age”

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



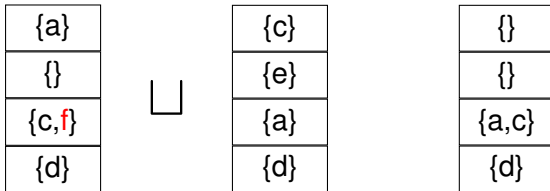
“Intersection + Maximal Age”

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



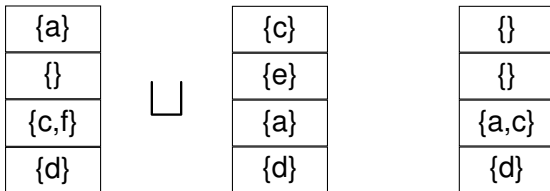
“Intersection + Maximal Age”

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

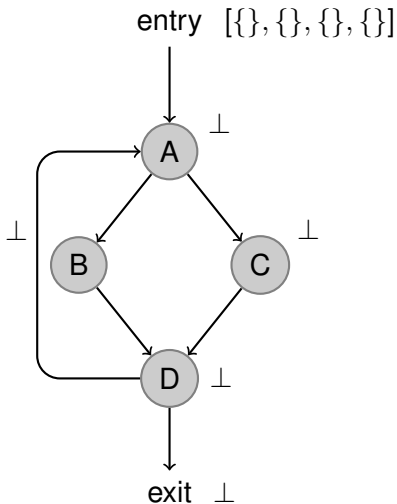
- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$



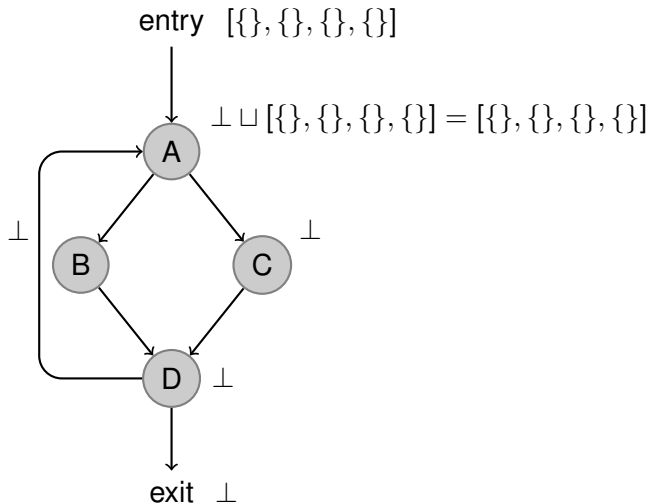
“Intersection + Maximal Age”

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

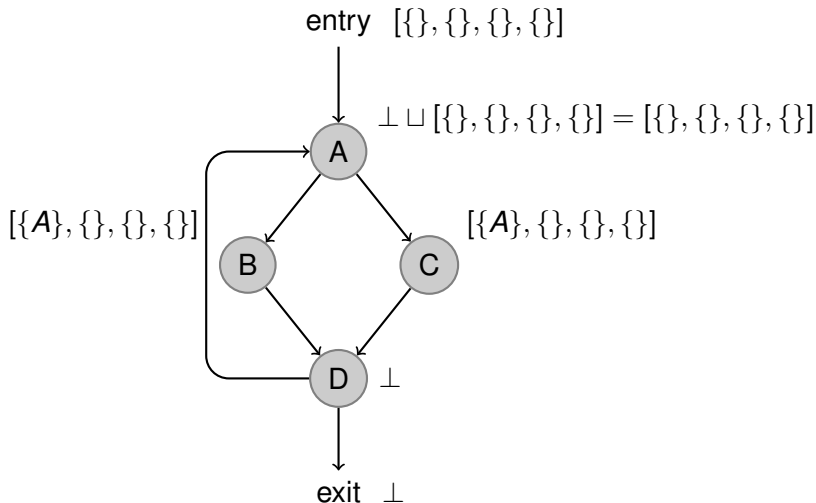
Example: Must-Analysis



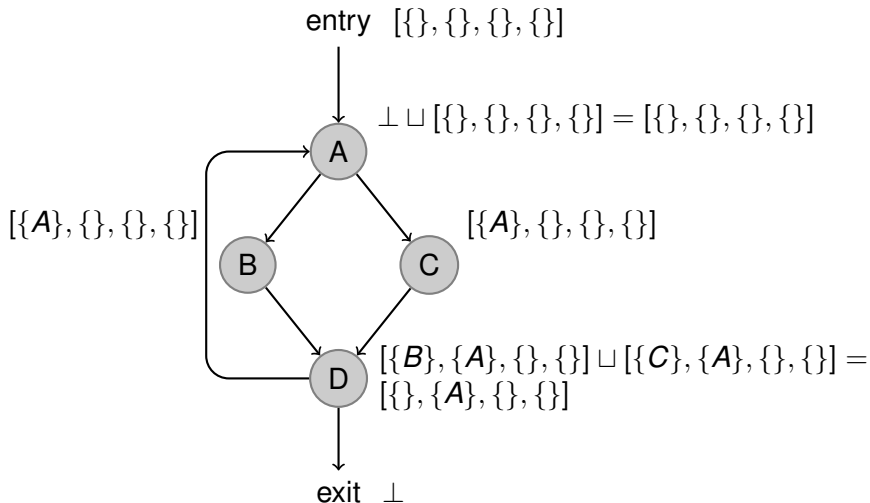
Example: Must-Analysis



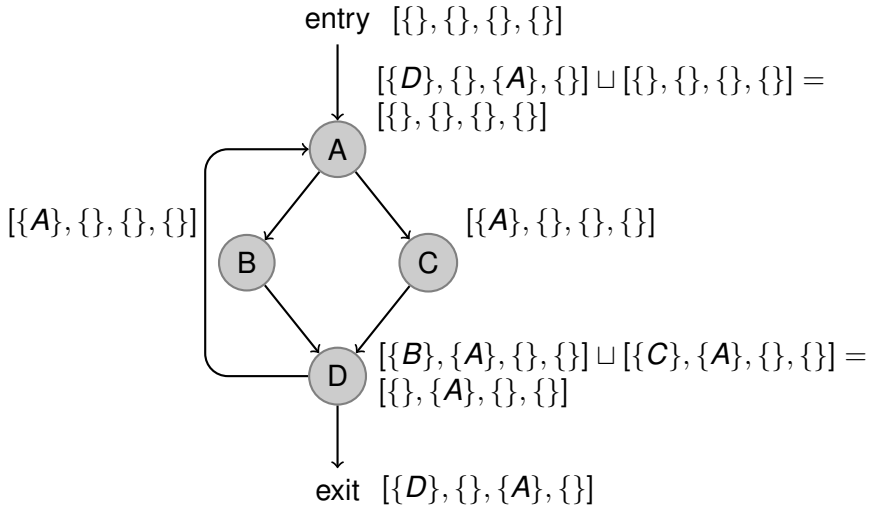
Example: Must-Analysis



Example: Must-Analysis



Example: Must-Analysis



No cache hits can be predicted :-)

■ Problem:

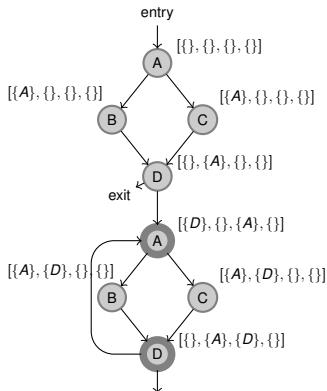
- ▶ The first iteration of a loop will always result in cache misses.
- ▶ Similarly for the first execution of a function.

■ 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.
→ 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}	age 3
{u}	

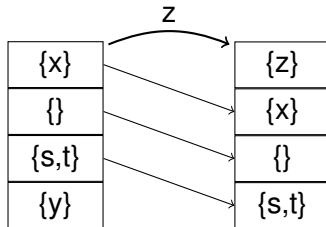
Describes the set of all concrete cache states in which no memory blocks except x , y , s , t , and u occur,

- 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.

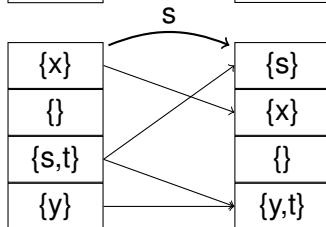
$$\gamma(\left[\{x,y\}, \{\}, \{s,t\}, \{u\}\right]) = \left[\left[x,y,s,t\right], \left[y,x,s,t\right], \left[x,y,s,u\right], \dots\right]$$

LRU: May-Analysis: Update

“Definite Cache Miss”:



“Potential Cache Hit”:



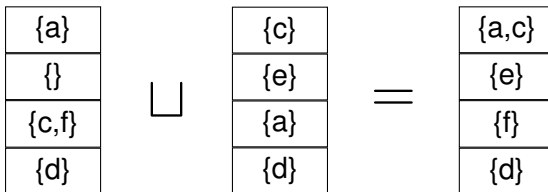
Why does t age in the second case?

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



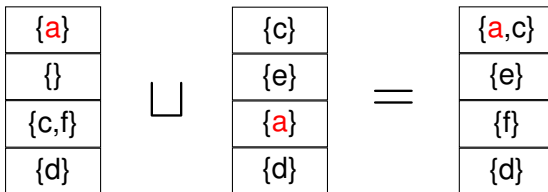
“Union + Minimal Age”

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



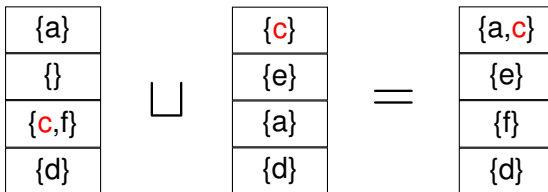
“Union + Minimal Age”

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



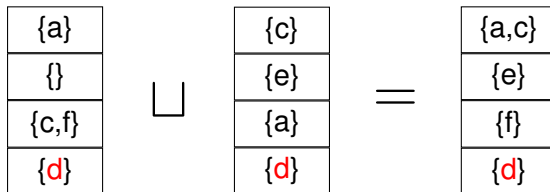
“Union + Minimal Age”

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



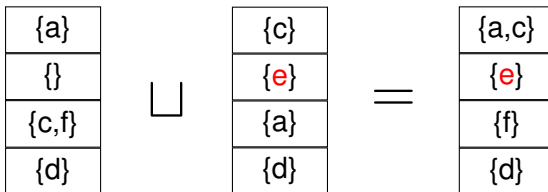
“Union + Minimal Age”

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

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



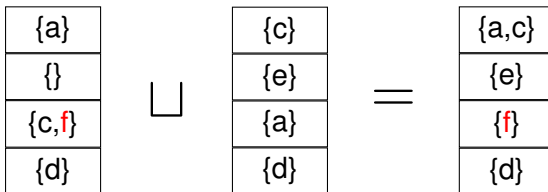
“Union + Minimal Age”

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”

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

1 Caches

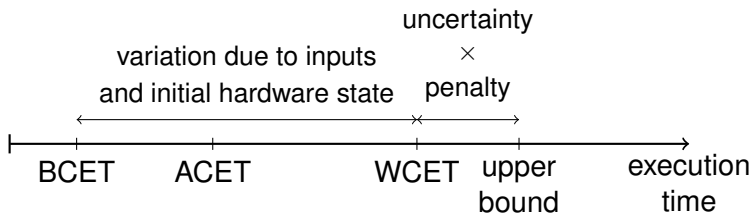
2 Cache Analysis for Least-Recently-Used

3 Beyond Least-Recently-Used

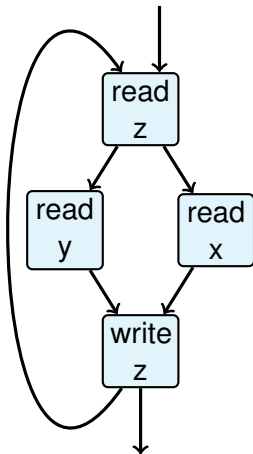
- Predictability Metrics
- Relative Competitiveness
- Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary

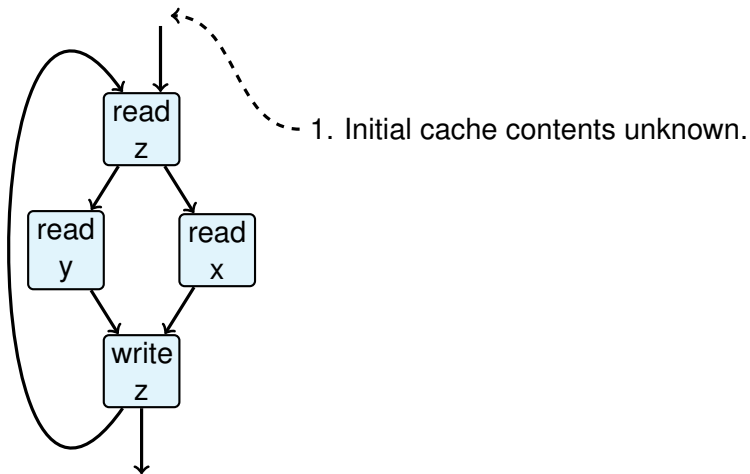
- Amount of uncertainty determines precision of WCET analysis
- Uncertainty in cache analysis depends on replacement policy



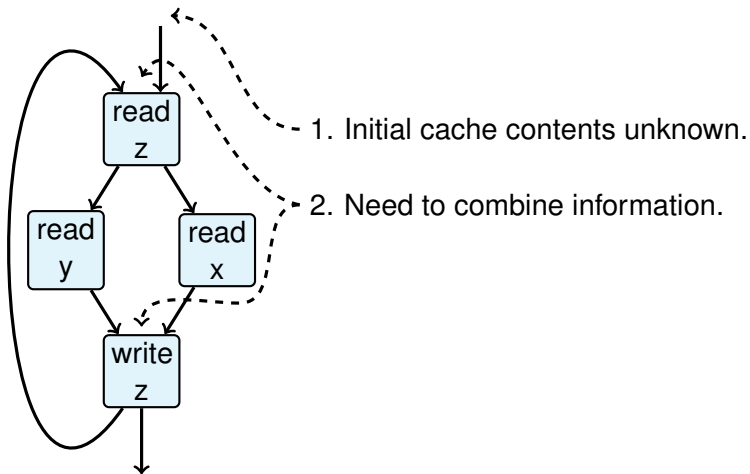
Uncertainty in Cache Analysis



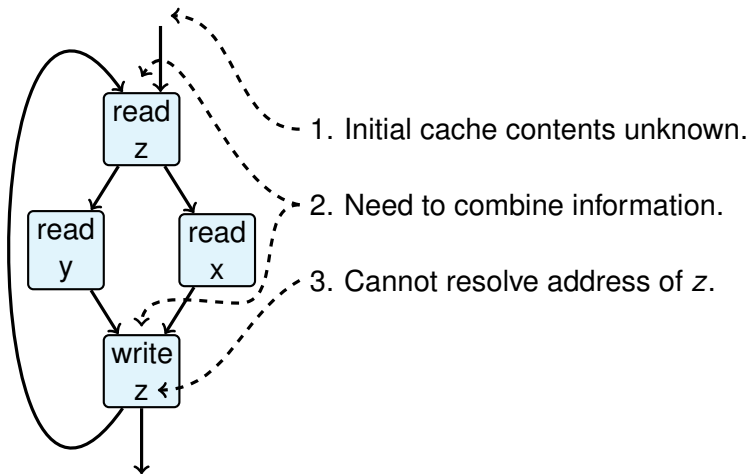
Uncertainty in Cache Analysis



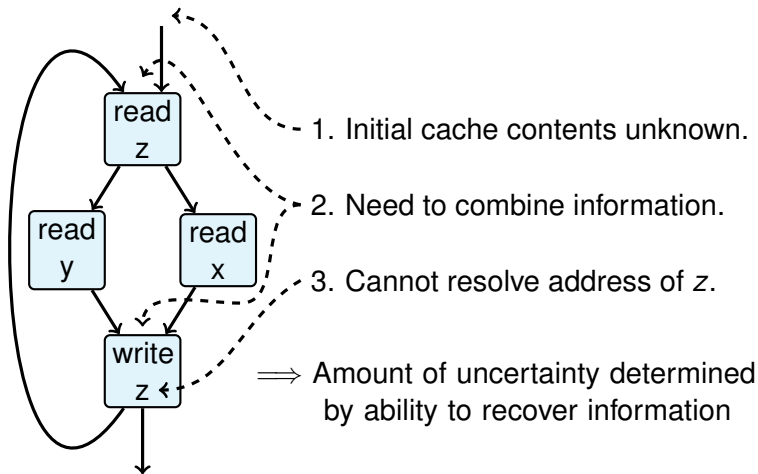
Uncertainty in Cache Analysis

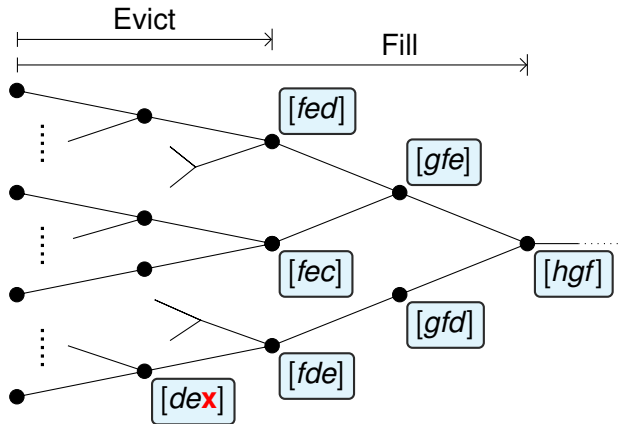


Uncertainty in Cache Analysis



Uncertainty in Cache Analysis





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

■ Evict

- ▶ Number of accesses to obtain *any may*-information.
- ▶ I.e. when can an analysis predict any cache misses?

■ Fill

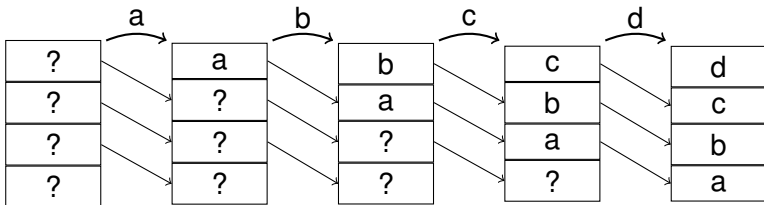
- ▶ Number of accesses to complete *may*- and *must*-information.
- ▶ I.e. when can an analysis predict each access?

→ 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

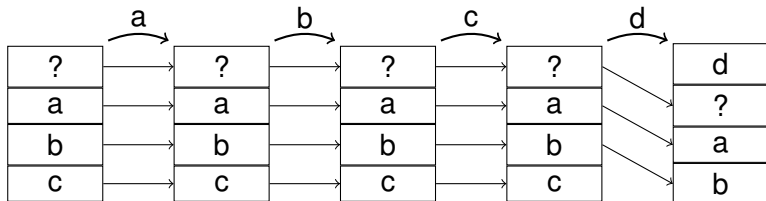


■ In the example: $\text{Evict} = \text{Fill} = 4$

■ In general: $\text{Evict}(k) = \text{Fill}(k) = k$, where k is the associativity of the cache

Evaluation of First-In First-Out (sketch)

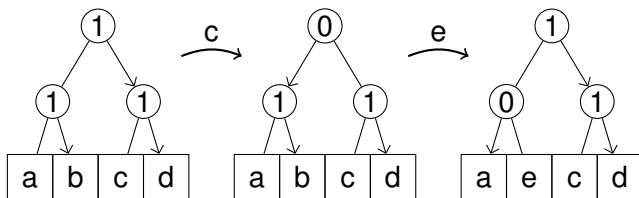
- Like LRU in the miss-case
- But: “Ignores” hits



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

Evaluation of Pseudo-LRU (sketch)

- Tree-bits point to block to be replaced



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

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.

→ Use LRU if predictability is a concern.

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

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

- **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

Notation

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

Notation

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

Definition (Relative miss competitiveness)

Policy \mathbf{P} is (k, c) -miss-competitive relative to policy \mathbf{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 $p \sim q$.

Notation

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

Definition (Relative miss competitiveness)

Policy \mathbf{P} is (k, c) -miss-competitive relative to policy \mathbf{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 \mathbf{P} relative to \mathbf{Q})

The smallest k , s.t. \mathbf{P} is (k, c) -miss-competitive rel. to \mathbf{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.

Let \mathbf{P} be $(1, 0)$ -competitive relative to \mathbf{Q} :

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

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

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**.

Let \mathbf{P} be $(1, 0)$ -competitive relative to \mathbf{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 \mathbf{Q} “hits”, so does \mathbf{P} , and
- 2 if \mathbf{P} “misses”, so does \mathbf{Q} .

As a consequence,

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

$$m_P \leq k \cdot m_Q + c$$

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**.

$$m_P \leq k \cdot m_Q + c$$

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

$$m_Q(T)$$

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**.

$$m_P \leq k \cdot m_Q + c$$

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

$$m_Q(T)$$

- 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**.

$$m_P \leq k \cdot m_Q + c$$

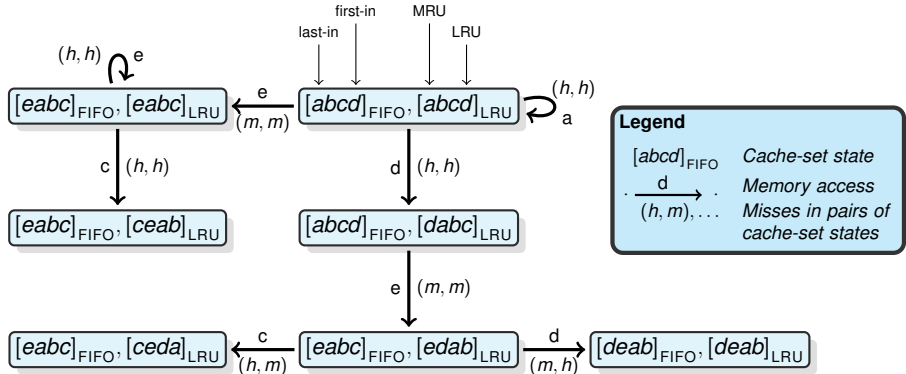
$$m_Q(T)$$

=

$$m_P(T)$$

Relative Competitiveness: Automatic Computation

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

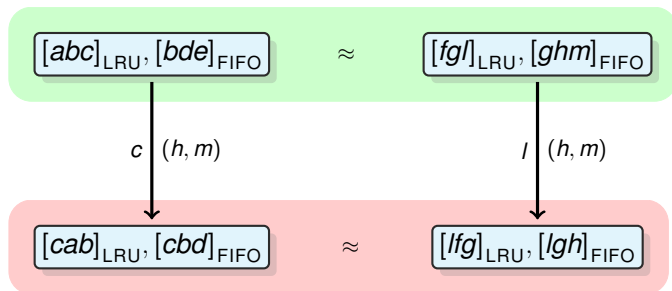


Competitive miss ratio = maximum ratio of misses in policy **P** to misses in policy **Q** in transition system

Transition System is ∞ Large

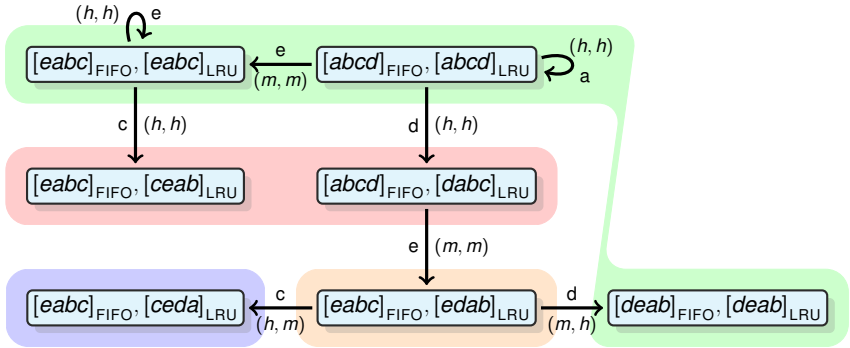
Problem: The induced transition system is ∞ large.

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

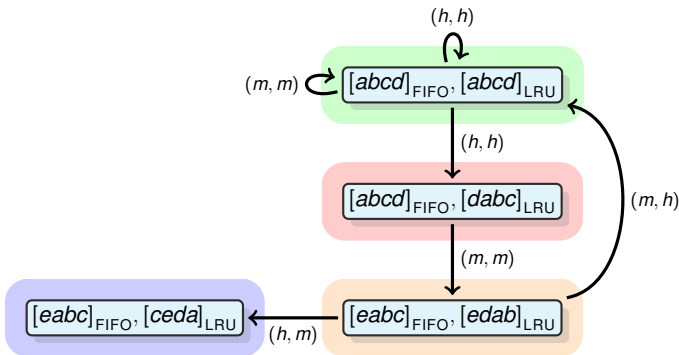


Solution: Construct *finite* quotient transition system.

\approx -Equivalent States in Running Example



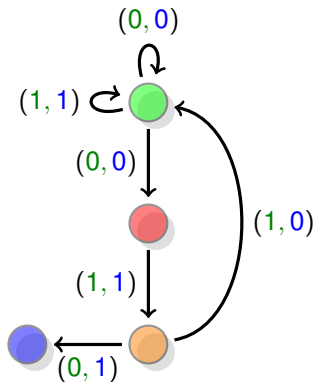
Merging \approx -equivalent states yields a finite quotient transition system:



Competitive Ratio = Maximum Cycle Ratio

Competitive miss ratio =

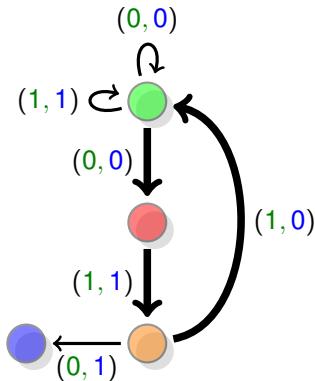
maximum ratio of misses in policy **P** to misses in policy **Q**



Competitive Ratio = Maximum Cycle Ratio

Competitive miss ratio =

maximum ratio of misses in policy **P** to misses in policy **Q**



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

- 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:

$\text{PLRU}(k)$ 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

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

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

\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:

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

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

1 Caches

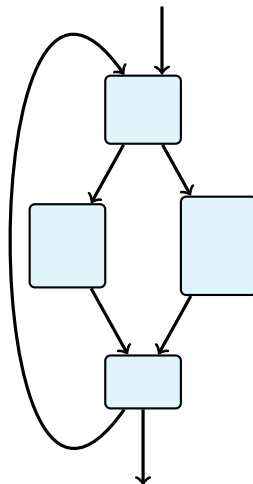
2 Cache Analysis for Least-Recently-Used

3 Beyond Least-Recently-Used

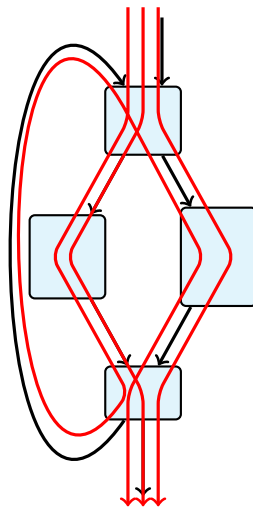
- Predictability Metrics
- Relative Competitiveness
- Sensitivity – Caches and Measurement-Based Timing Analysis

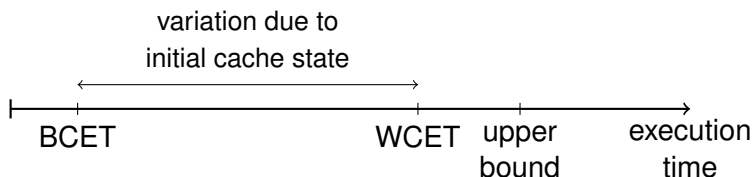
4 Summary

- 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.



- 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.





Definition (Miss sensitivity)

Policy \mathbf{P} is (k, c) -miss-sensitive if

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

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

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.

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

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

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.
- 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.
- LRU is the most predictable policy.

Relative Competitiveness

- ... allows to derive guarantees on cache performance,
- ... yields first *may*-analyses for FIFO and MRU.

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.
- 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.
- 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.
- LRU is the most predictable policy.

Relative Competitiveness

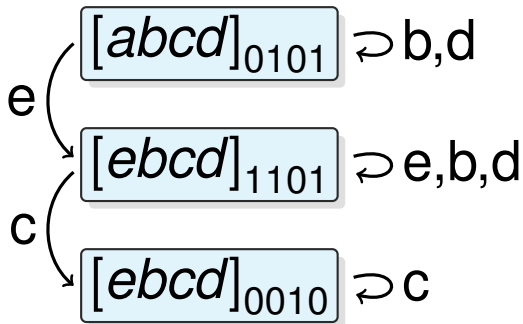
- ... allows to derive guarantees on cache performance,
- ... yields first *may*-analyses for FIFO and MRU.

Sensitivity Analysis

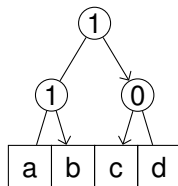
- ... determines the influence of initial state on cache performance.

Thank you for your attention!

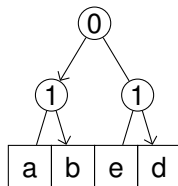
MRU-bits record whether line was recently used



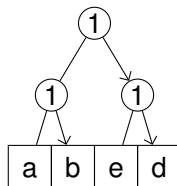
→ Never converges



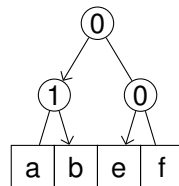
Initial cache-
set state
 $[a, b, c, d]_{110}$.



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.

$$May^{\mathbf{P}}(s) := \bigcup_{p \in C^{\mathbf{P}}} CC_{\mathbf{P}}(update_{\mathbf{P}}(p, s))$$

$$Must^{\mathbf{P}}(s) := \bigcap_{p \in C^{\mathbf{P}}} CC_{\mathbf{P}}(update_{\mathbf{P}}(p, s))$$

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

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

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

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

where k is \mathbf{P} 's associativity.

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

- (i) $Evict^P(k) \geq Evict^Q(I)$,
- (ii) $mls^P(k) \geq mls^Q(I)$.

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

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

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

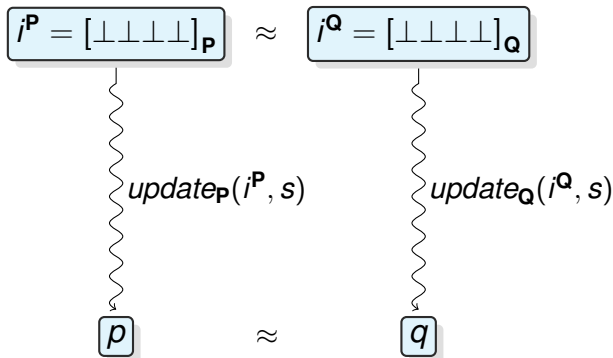
$$\text{Alt-mls}^P(k) = I.$$

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

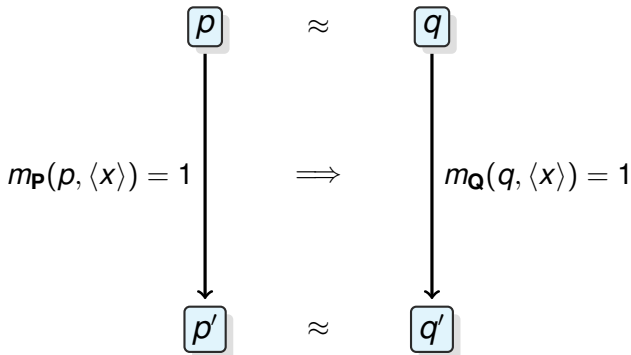
$$\begin{aligned}
 \sum_{j=0}^{\min\{k,k'\}} \binom{k}{j} \binom{k'}{j} j! &\leq k! \cdot k'! \sum_{j=0}^{\min\{k,k'\}} \frac{1}{(k-j)! j! (k'-j)!} \\
 &\leq k! \cdot k'! \sum_{j=0}^{\infty} \frac{1}{j!} = e \cdot k! \cdot k'!
 \end{aligned}$$

This can be bounded by

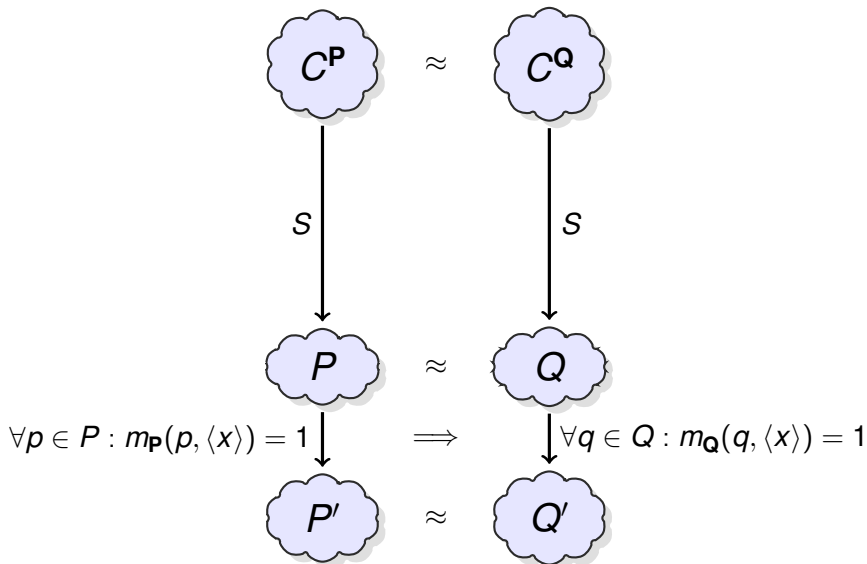
$$2^{l+l'+k+k'} \leq |(C_k^l \times C_{k'}^{l'}) / \approx| \leq 2^{l+l'+k+k'} \cdot \underbrace{e \cdot k! \cdot k'!}_{\text{bound on number of overlappings}}$$



Let \mathbf{P} be (1, 0)-competitive relative to \mathbf{Q} , then



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



- 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

$$\begin{aligned}\frac{T_{wc}}{T_{meas}} &= \frac{CPI_{hit} + \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate}_{wc} \times \text{Miss penalty}}{CPI_{hit} + \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate}_{meas} \times \text{Miss penalty}} \\ &= \frac{1.5 + 1.2 \times 0.20 \times 50}{1.5 + 1.2 \times 0.05 \times 50} = \frac{13.5}{4.5} = 3\end{aligned}$$