Verification of Real-Time Systems
Caches in WCET Analysis

Jan Reineke

Department of Computer Science
Saarland University
Saarbrücken, Germany

Advanced Lecture, Summer 2015

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. Caches

2. Cache Analysis for Least-Recently-Used
   - Challenges
   - Formalization of LRU and Logical Abstraction
   - Must Analysis
   - May Analysis
   - Remaining Challenges

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

4. Summary

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. Caches

2. Cache Analysis for Least-Recently-Used
   - Challenges
   - Formalization of LRU and Logical Abstraction
   - Must Analysis
   - May Analysis
   - Remaining Challenges

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

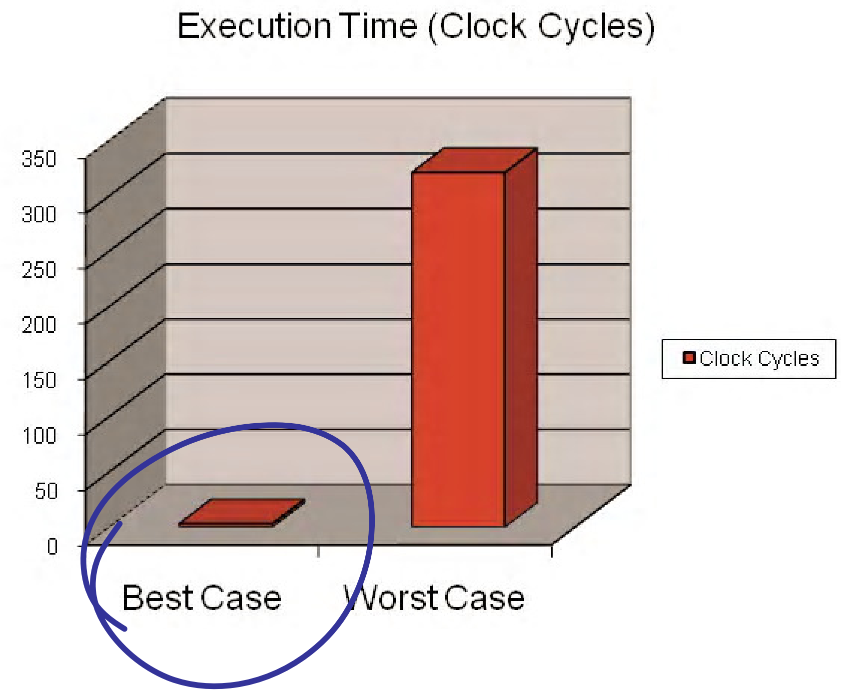
4. Summary

Cache vs Memory Performance

x=a+b; →

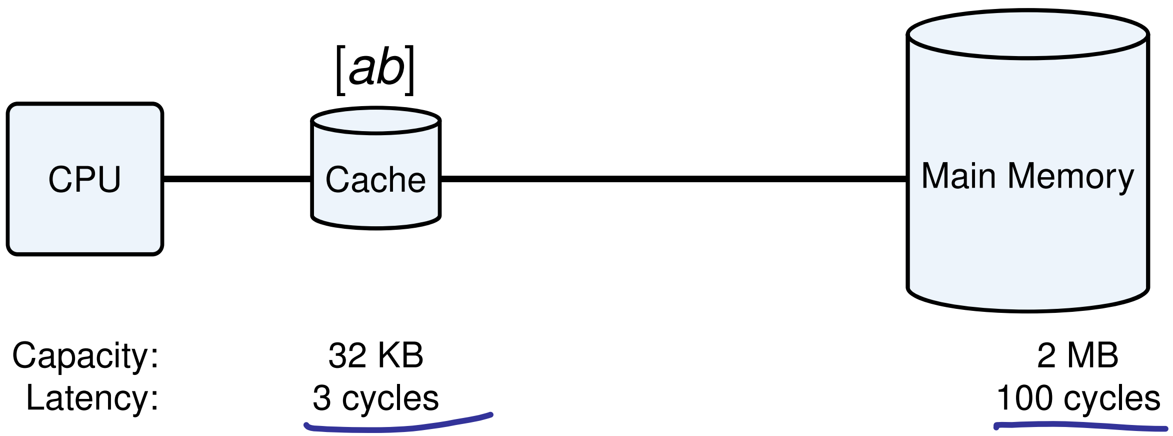
```
LOAD   r2, _a
LOAD   r1, _b
ADD    r3,r2,r1
```

Execution time on Motorola PowerPC 755:



Caches

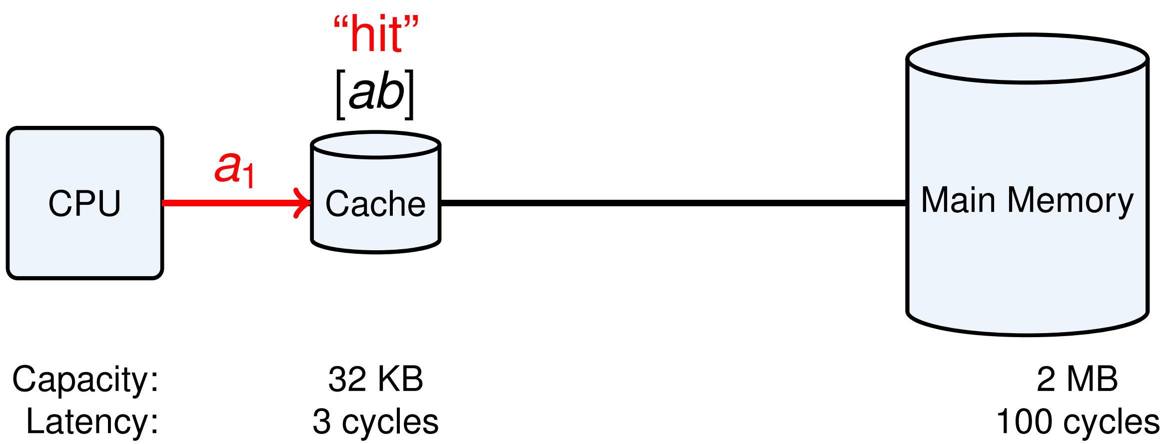
- How they work:
  - dynamically
  - managed by replacement policy



- Why they work: *principle of locality*
  - spatial
  - temporal

Caches

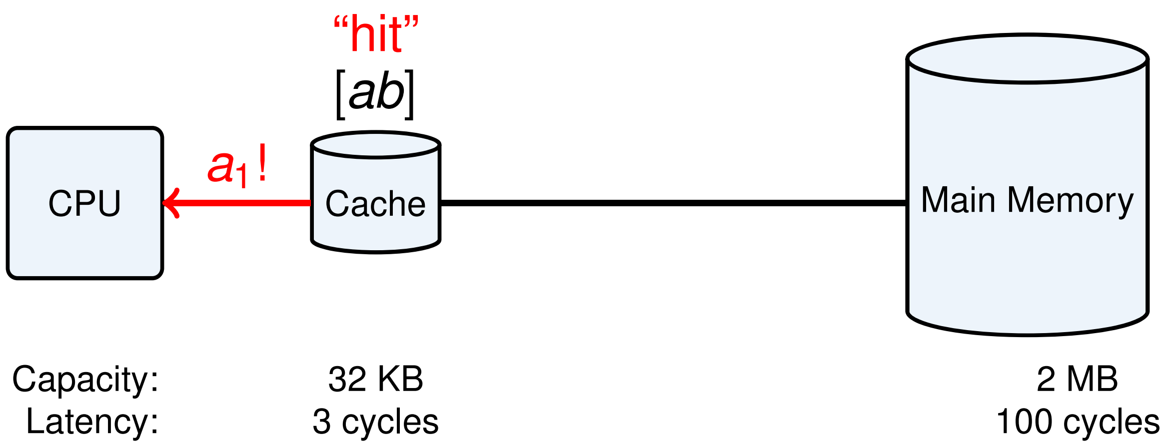
- How they work:
  - dynamically
  - managed by replacement policy



- Why they work: *principle of locality*
  - spatial
  - temporal

Caches

- How they work:
  - dynamically
  - managed by replacement policy

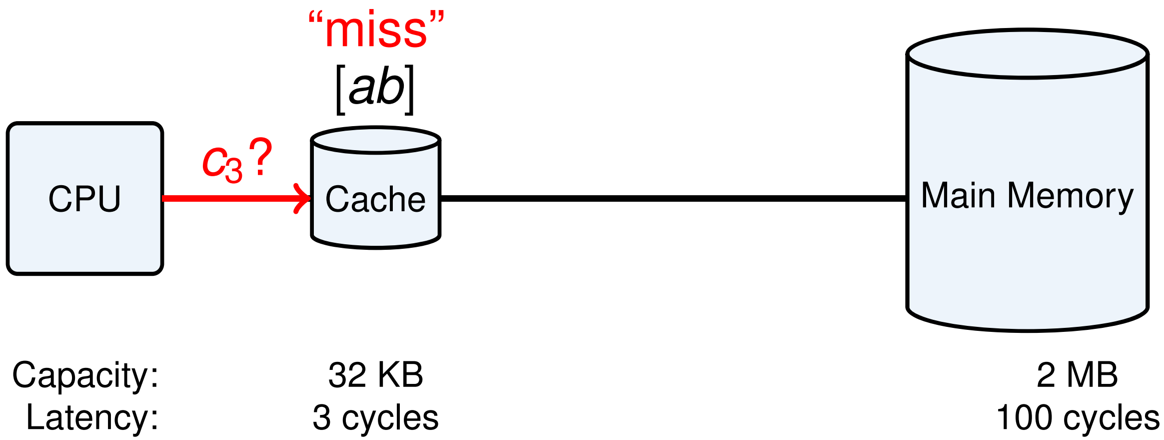


- Why they work: *principle of locality*
  - spatial
  - temporal

Caches

SAARLAND UNIVERSITY
COMPUTER SCIENCE

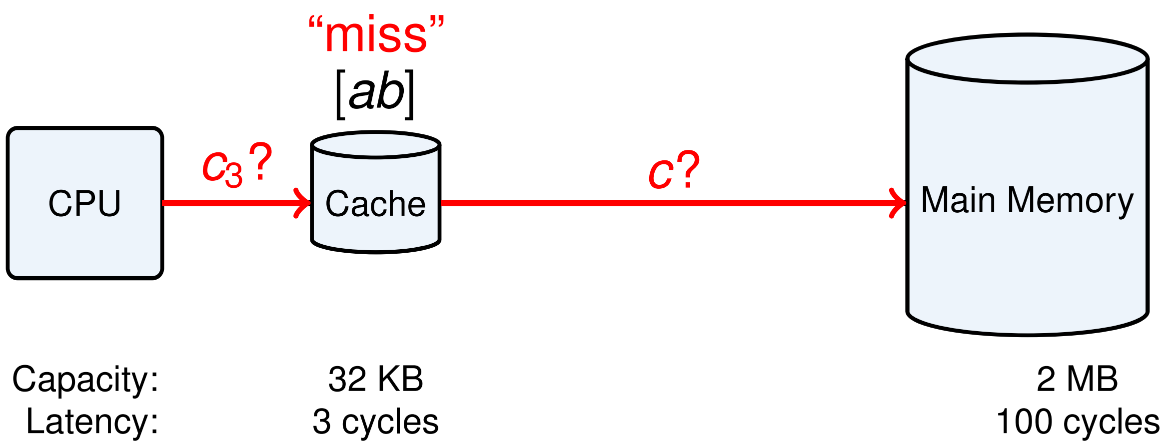
- How they work:
  - dynamically
  - managed by replacement policy



- Why they work: *principle of locality*
  - spatial
  - temporal

Caches

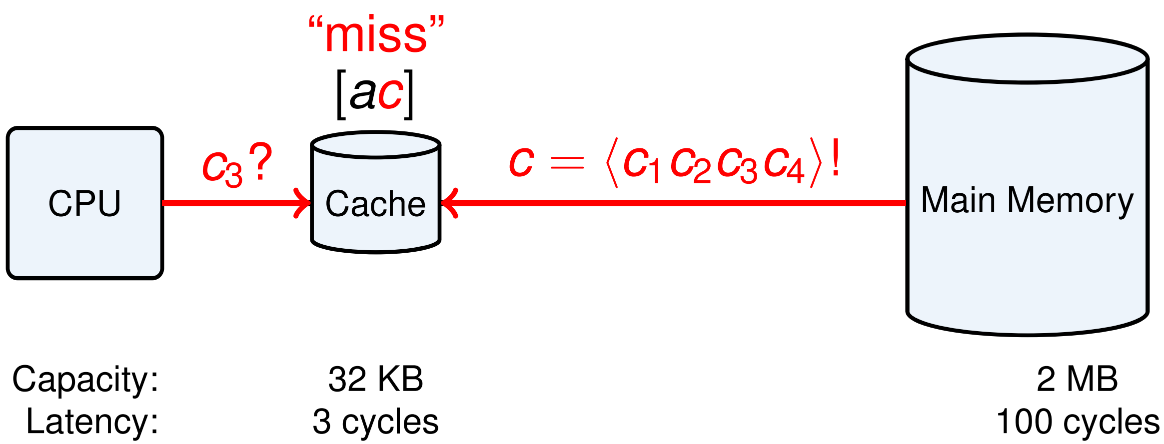
- How they work:
  - dynamically
  - managed by replacement policy



- Why they work: *principle of locality*
  - spatial
  - temporal

Caches

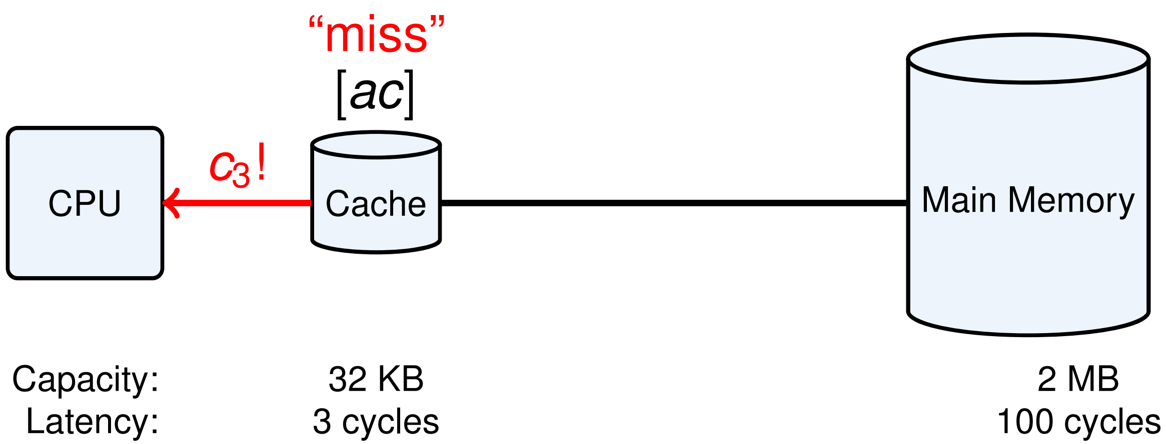
- How they work:
  - dynamically
  - managed by replacement policy



- Why they work: *principle of locality*
  - spatial
  - temporal

Caches

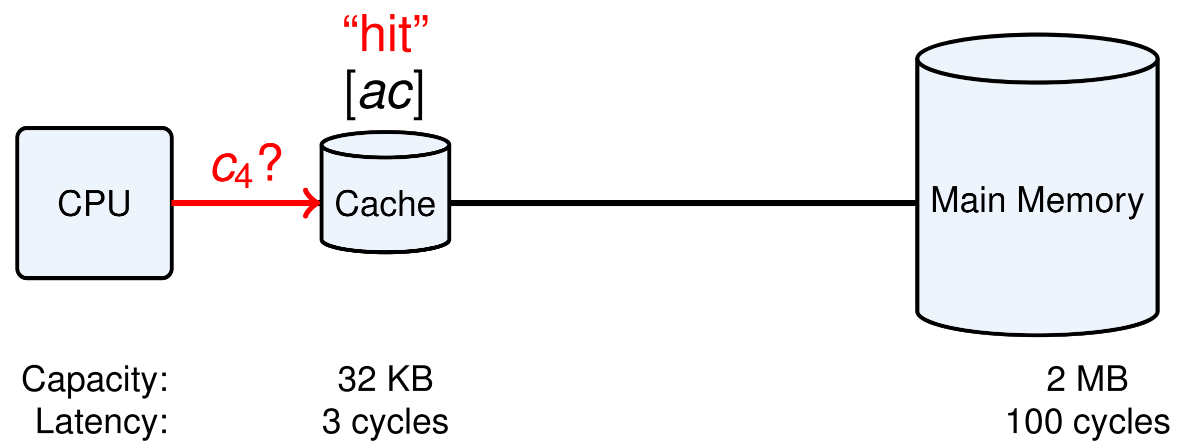
- How they work:
  - dynamically
  - managed by replacement policy



- Why they work: *principle of locality*
  - spatial
  - temporal

Caches

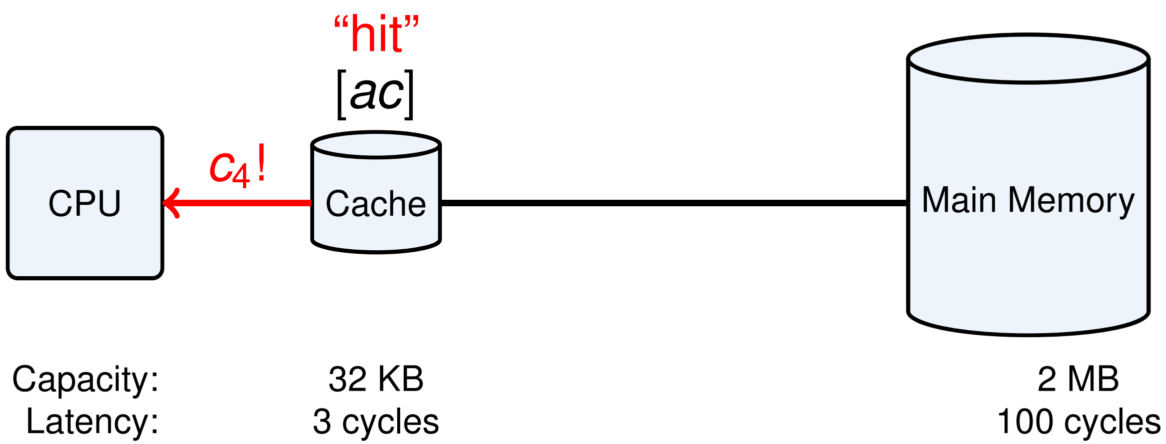
- How they work:
  - dynamically
  - managed by replacement policy



- Why they work: *principle of locality*
  - spatial
  - temporal

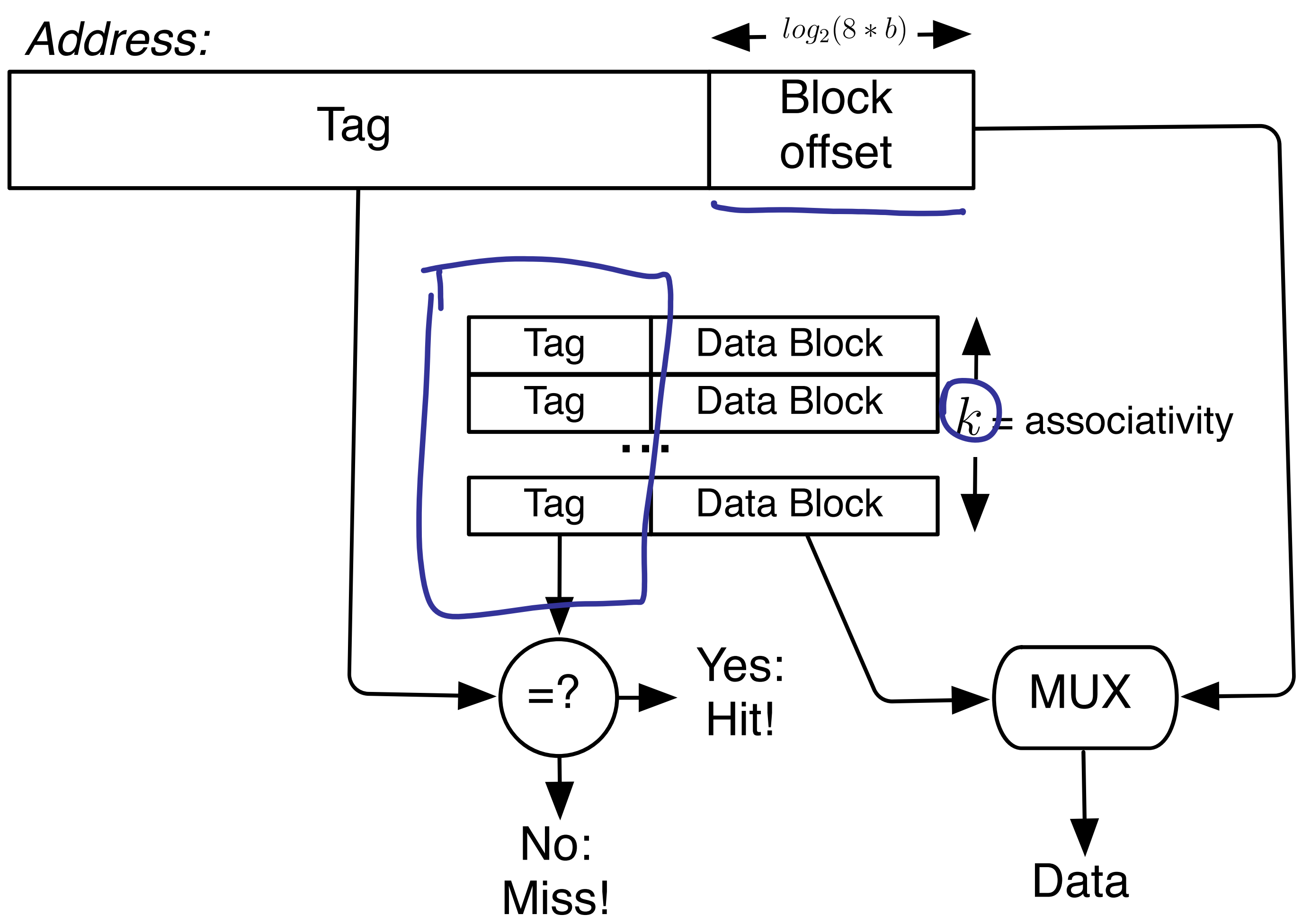
Caches

- How they work:
  - dynamically
  - managed by replacement policy

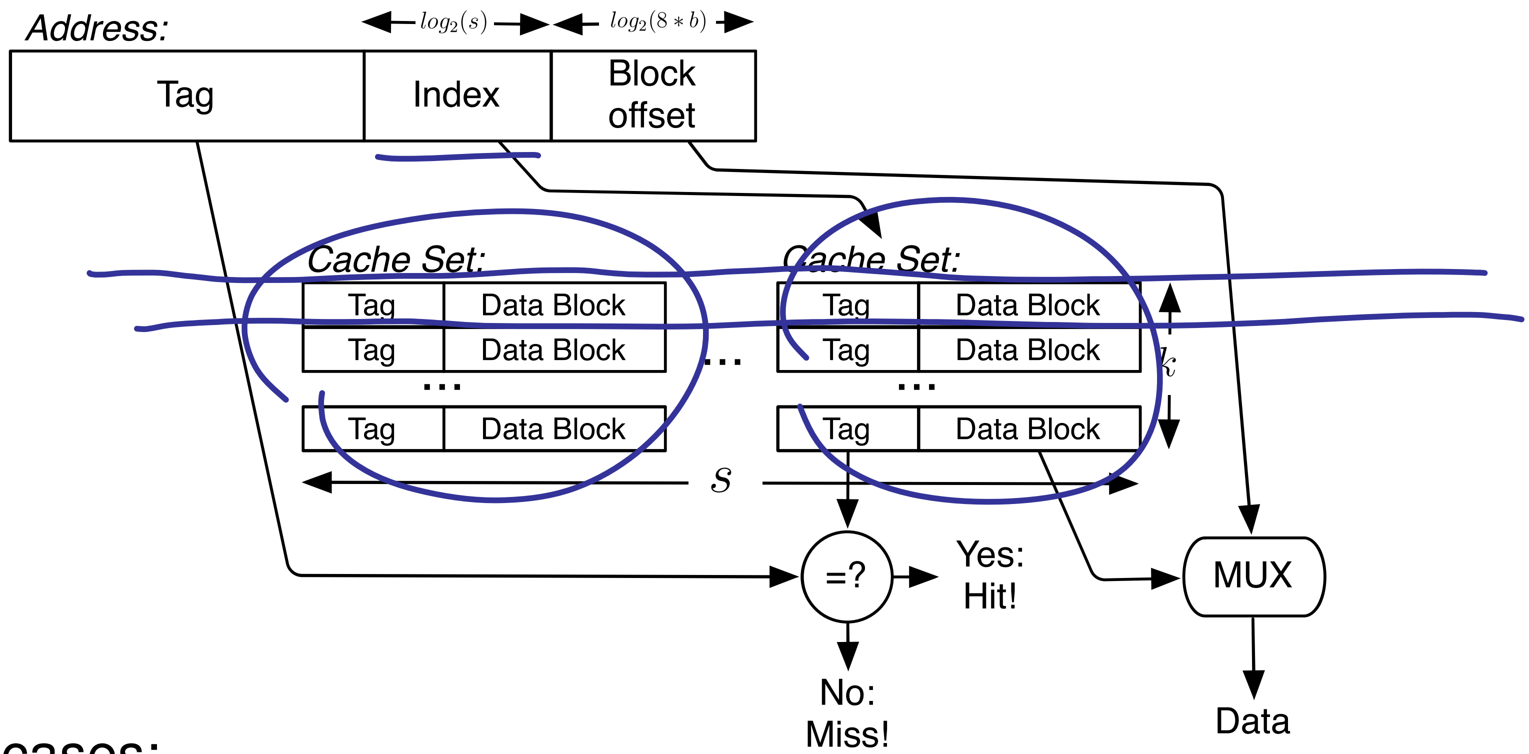


- Why they work: *principle of locality*
  - spatial
  - temporal

Fully-Associative Caches



Set-Associative Caches



Special cases:

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

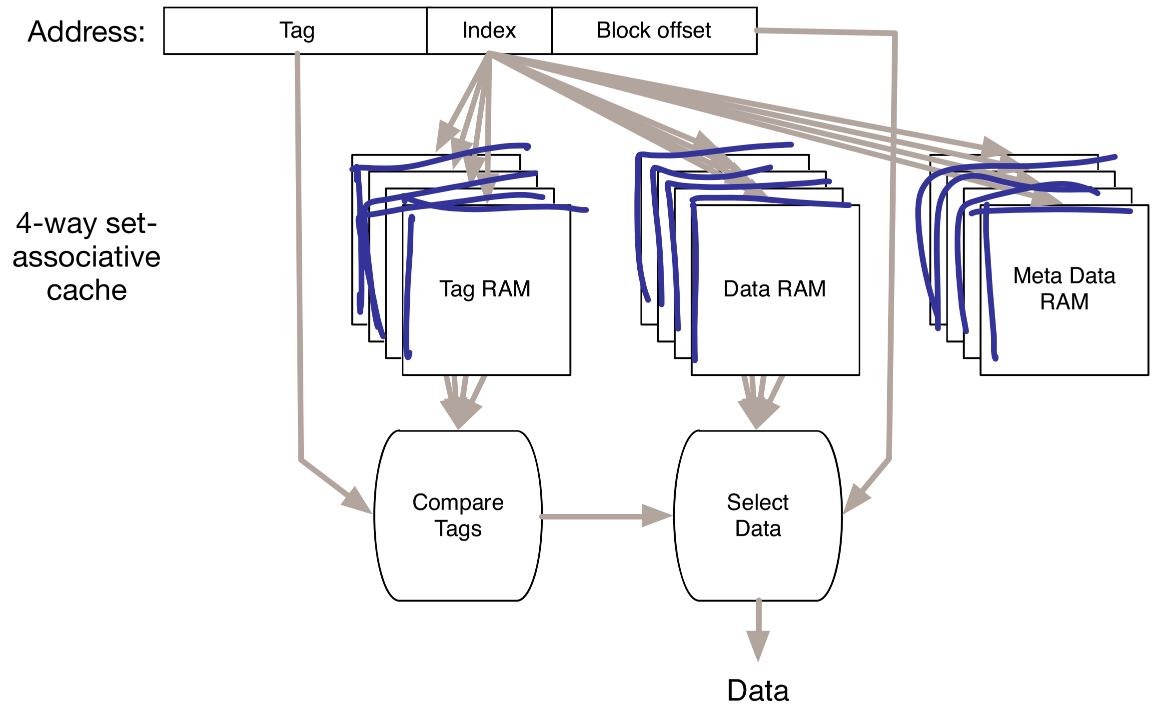
Cache Replacement Policies

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:
$\longrightarrow$ Set-associative caches are compositions of fully-associative caches.

“Physical” Cache Implementation



- One Tag RAM per cache way: enables parallel lookup
- “Meta Data” RAM maintains additional information:
  - Replacement policy status
  - Data caches: “dirty” bits
  - Shared caches: cache coherence information

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1 Caches

2 Cache Analysis for Least-Recently-Used
- Challenges
- Formalization of LRU and Logical Abstraction
- Must Analysis
- May Analysis
- Remaining Challenges

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

4 Summary

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. Caches

2. Cache Analysis for Least-Recently-Used
   - Challenges
   - Formalization of LRU and Logical Abstraction
   - Must Analysis
   - May Analysis
   - Remaining Challenges

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

4. Summary

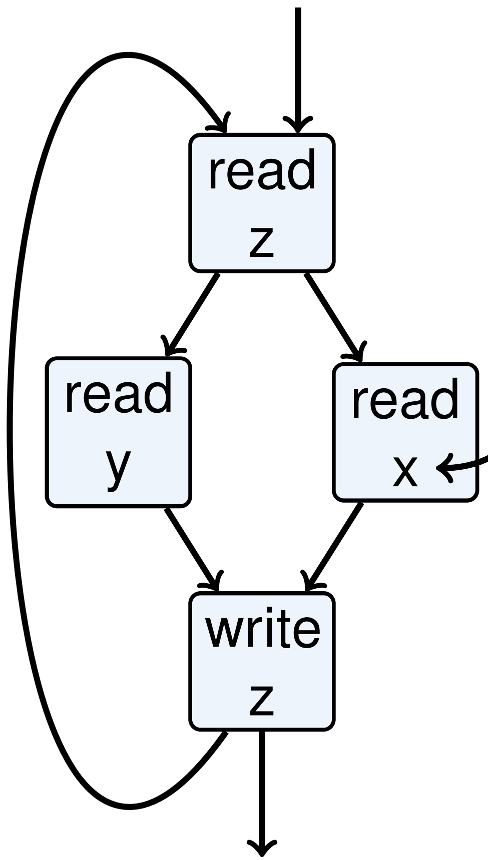
Cache Analysis

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
   - Special case: Persistence analysis

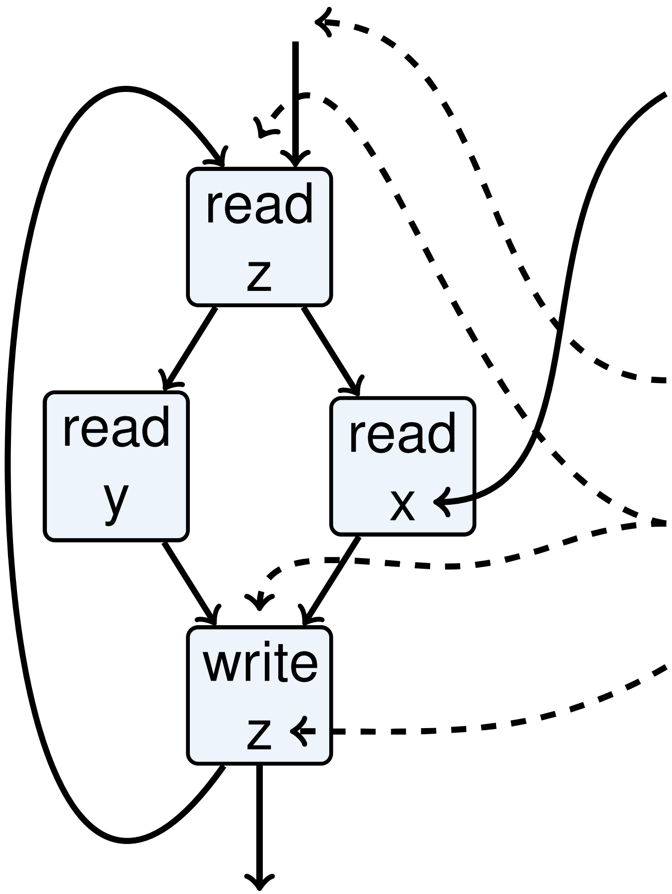
- Cache analyses mostly for LRU
- In practice: FIFO, PLRU, ...

Challenges for Cache Analysis



Always a cache hit/always a miss?

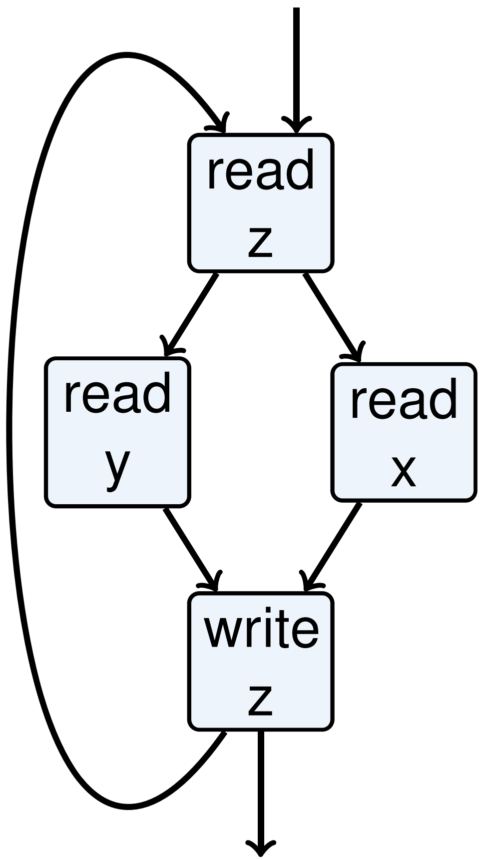
Challenges for Cache Analysis



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

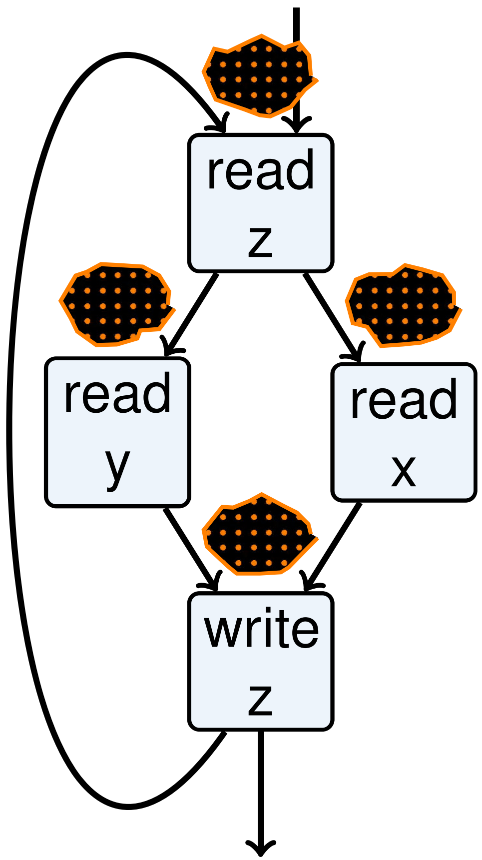


Collecting Semantics =
set of states at each program point that any execution may encounter there

Two approximations:

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

Deriving Invariants about Cache States using Abstract Interpretation



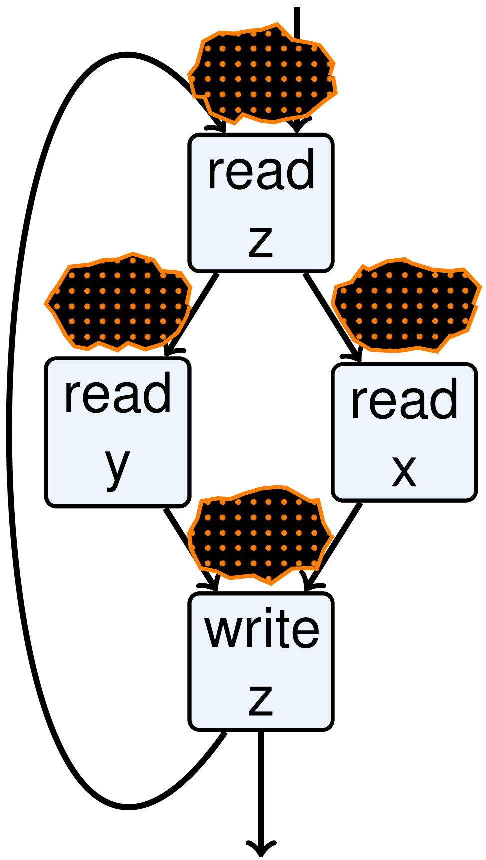
Collecting Semantics =
set of states at each program point that any execution may encounter there

Two approximations:

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

Deriving Invariants about Cache States using Abstract Interpretation

SAARLAND UNIVERSITY COMPUTER SCIENCE

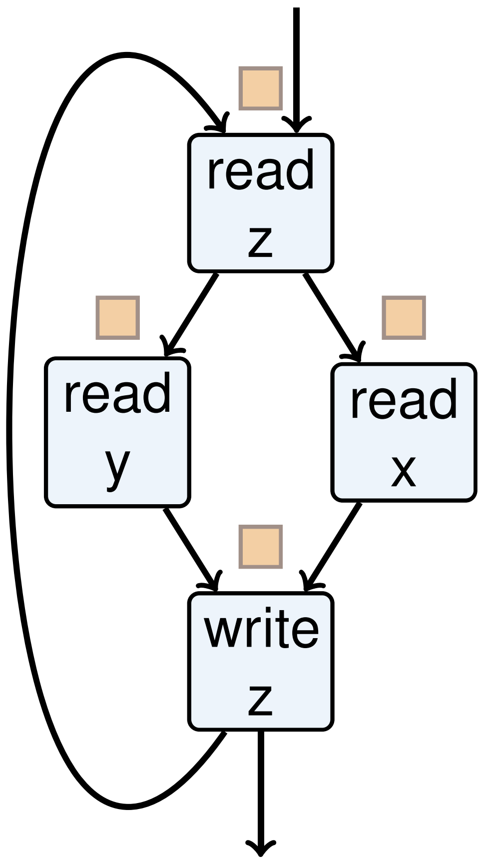


Collecting Semantics =
set of states at each program point that any execution may encounter there

Two approximations:

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

Deriving Invariants about Cache States using Abstract Interpretation

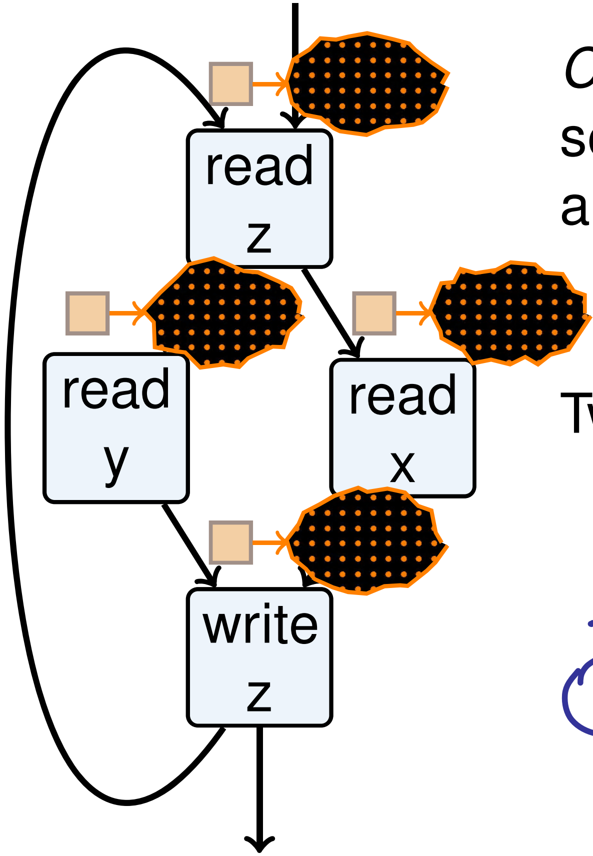


Collecting Semantics =
set of states at each program point that any execution may encounter there

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

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. Caches
2. Cache Analysis for Least-Recently-Used
   - Challenges
   - Formalization of LRU and Logical Abstraction
   - Must Analysis
   - May Analysis
   - Remaining Challenges
3. Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis
4. Summary

Least-Recently-Used (LRU): Concrete Domain

Concrete states of k-way fully-associative cache captured by three functions corresponding to *tag*, *data*, and *meta data* RAMs:

"physical"

$$\begin{aligned} tag &: \{1, \ldots, k\} \rightarrow \mathcal{B}, \quad \text{MEMORY BLOCKS} \\ data &: \{1, \ldots, k\} \rightarrow \mathbb{B}^n, \\ meta &: \{1, \ldots, k\} \rightarrow \{0, \ldots, \underline{k-1}\}, \end{aligned}$$

where *meta* captures the "age" of the memory block stored in the i^{th} cache line. This is the only LRU-specific aspect.

Least-Recently-Used (LRU): A First Abstraction

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. Cache analysis is interested in *which* memory blocks are cached, not *what data* they hold $\longrightarrow$ abstract from *data*.
2. For predicting hits and misses, it is irrelevant in which *physical* cache line a memory block is cached. Only the relative ages of different cache blocks are important.

A “logical” LRU cache state can be captured by

$$age : \mathcal{B} \rightarrow \{0, \ldots, k-1, \underline{k}\},$$

where uncached blocks obtain age k.

Least-Recently-Used (LRU): A First Abstraction

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. Cache analysis is interested in *which* memory blocks are cached, not *what data* they hold $\longrightarrow$ abstract from *data*.
2. For predicting hits and misses, it is irrelevant in which *physical* cache line a memory block is cached. Only the relative ages of different cache blocks are important.

A “logical” LRU cache state can be captured by

$$age : \mathcal{B} \to \{0, \ldots, k-1, k\},$$

where uncached blocks obtain age k.
Logical state be obtained from “physical” cache states as follows:

$$\alpha(tag, data, meta) := \lambda b \in \mathcal{B} : \begin{cases} meta(i) & : \text{if } tag(i) = b \\ k & : \text{if } tag(i) \neq b \; \forall i \in \{1, \ldots, k\} \end{cases}$$

Least-Recently-Used (LRU): Logical Behavior

SAARLAND UNIVERSITY COMPUTER SCIENCE

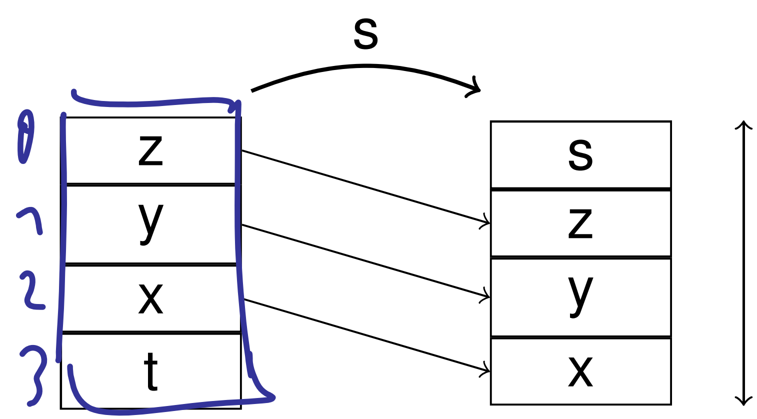
$$up(age, b) := \lambda b' \in \mathcal{B} : \begin{cases} 0 & : \text{if } b' = b \\ age(b') & : \text{if } age(b) \leq age(b') \\ age(b') + 1 & : \text{if } age(b) > age(b') \end{cases}$$

Least-Recently-Used (LRU): Logical Behavior

SAARLAND UNIVERSITY
COMPUTER SCIENCE

$$up(age, b) := \underline{\lambda b'} \in \mathcal{B} : \begin{cases} 0 & : \textit{if } b' = b \\ age(b') & : \textit{if } age(b) \leq age(b') \\ age(b') + 1 & : \textit{if } age(b) > \underline{age(b')} \end{cases}$$

"Cache Miss":

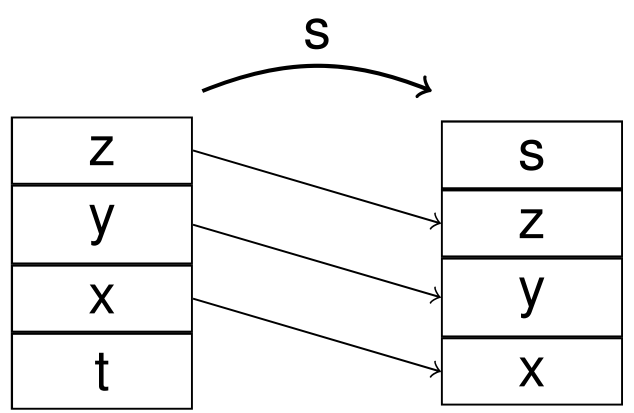


LRU has notion of age

Least-Recently-Used (LRU): Logical Behavior

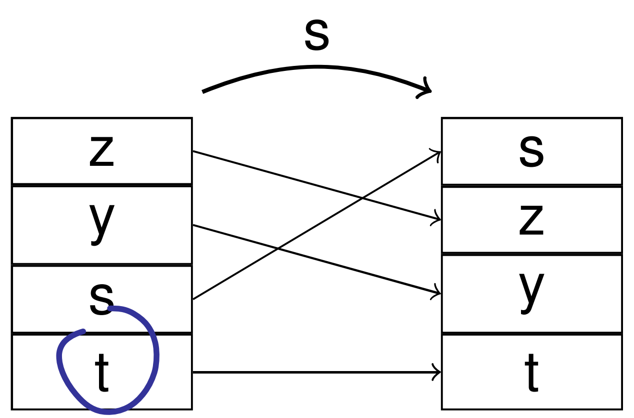
$$up(age, b) := \lambda b' \in \mathcal{B} : \begin{cases} 0 & : \text{if } b' = b \\ age(b') & : \text{if } age(b) \leq age(b') \\ age(b') + 1 & : \text{if } age(b) > age(b') \end{cases}$$

"Cache Miss":



LRU has notion of age

"Cache Hit":



Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. Caches
2. Cache Analysis for Least-Recently-Used
   - Challenges
   - Formalization of LRU and Logical Abstraction
   - Must Analysis
   - May Analysis
   - Remaining Challenges
3. Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis
4. Summary

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
{}	1
{s,t}	2
{}	age 3

. . . and its interpretation:

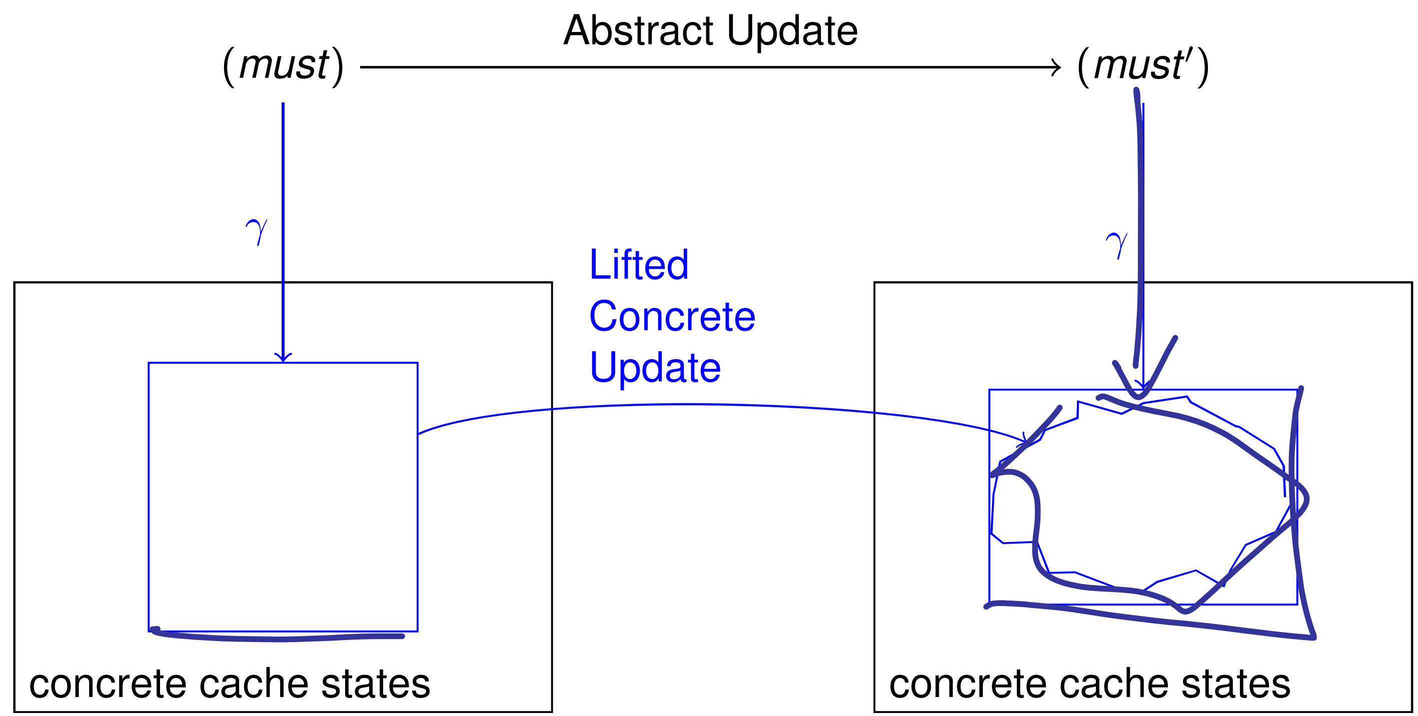
Describes the set of all concrete cache states in which x, s, and t occur,

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

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

0 1 2 3

Sound Update – Local Consistency



LRU: Must-Analysis: Update

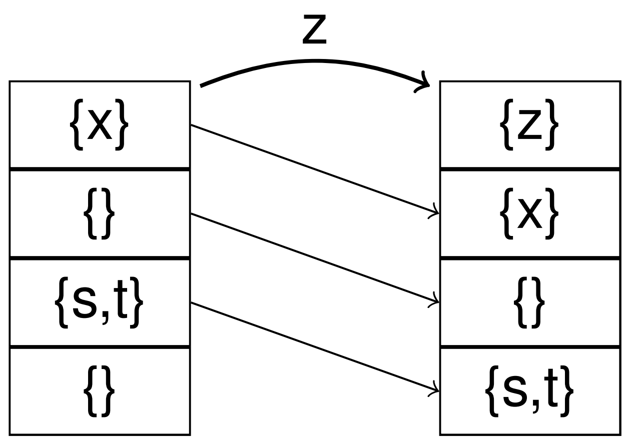
Formally:

$$up_{must}(\widehat{age}, b) := \lambda b' \in \mathcal{B} : \begin{cases} 0 & : \textit{if } b' = b \\ age(b') & : \textit{if } \widehat{age}(b) \leq \widehat{age}(b') \\ age(b') + 1 & : \textit{if } \widehat{age}(b) > \widehat{age}(b') \end{cases}$$

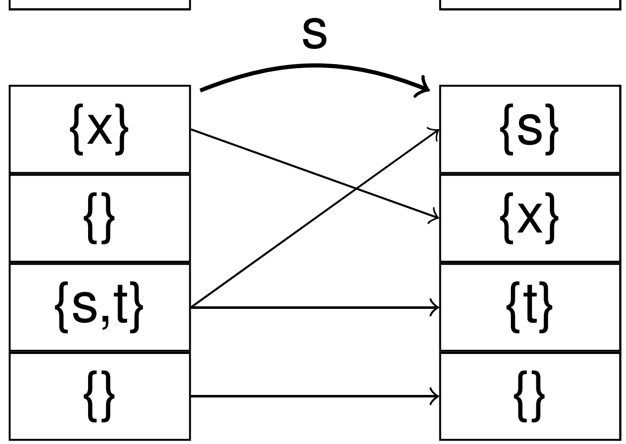
LRU: Must-Analysis: Update

SAARLAND UNIVERSITY
COMPUTER SCIENCE

"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 (ensures γ is monotone):

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

Formally: $age \sqcup age' := \underline{\lambda b \in \mathcal{B}} : \max\{age(b), age'(b)\}$

Graphically:

$$\begin{array}{|c|}\hline \{a\} \\ \hline \{\} \\ \hline \{c,f\} \\ \hline \{d\} \\ \hline \end{array} \sqcup \begin{array}{|c|}\hline \{c\} \\ \hline \{e\} \\ \hline \{a\} \\ \hline \{d\} \\ \hline \end{array} = \begin{array}{|c|}\hline \{\} \\ \hline \{\} \\ \hline \{a,c\} \\ \hline \{d\} \\ \hline \end{array}$$

"INTERSECTION" + "MAXIMUM AGE"

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \max\{age(b), age'(b)\}$

Graphically:

$$\begin{array}{|c|}\hline \{a\} \\ \hline \{\} \\ \hline \{c,f\} \\ \hline \{d\} \\ \hline \end{array} \quad \bigsqcup \quad \begin{array}{|c|}\hline \{c\} \\ \hline \{e\} \\ \hline \{a\} \\ \hline \{d\} \\ \hline \end{array} \quad = \quad \begin{array}{|c|}\hline \{\} \\ \hline \{\} \\ \hline \{a,c\} \\ \hline \{d\} \\ \hline \end{array}$$

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \max\{age(b), age'(b)\}$

Graphically:

$$\begin{array}{|c|}\hline \{a\} \\ \hline \{\} \\ \hline \{c,f\} \\ \hline \{d\} \\ \hline \end{array} \;\bigsqcup\; \begin{array}{|c|}\hline \{c\} \\ \hline \{e\} \\ \hline \{a\} \\ \hline \{d\} \\ \hline \end{array} \;=\; \begin{array}{|c|}\hline \{\} \\ \hline \{\} \\ \hline \{a,c\} \\ \hline \{d\} \\ \hline \end{array}$$

LRU: Must-Analysis: Join

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \max\{age(b), age'(b)\}$

Graphically:

{a}		{c}		{}
{}	$\bigsqcup$	{e}	$=$	{}
{c,f}		{a}		{a,c}
{d}		{d}		{d}

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \max\{age(b), age'(b)\}$

Graphically:

$$\begin{array}{|c|}\hline \{a\} \\ \hline \{\} \\ \hline \{c,f\} \\ \hline \{d\} \\ \hline \end{array} \bigsqcup \begin{array}{|c|}\hline \{c\} \\ \hline \{e\} \\ \hline \{a\} \\ \hline \{d\} \\ \hline \end{array} = \begin{array}{|c|}\hline \{\} \\ \hline \{\} \\ \hline \{a,c\} \\ \hline \{d\} \\ \hline \end{array}$$

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \max\{age(b), age'(b)\}$

Graphically:

{a}		{c}		{}
{}	$\sqcup$	{e}	$=$	{}
{c,f}		{a}		{a,c}
{d}		{d}		{d}

LRU: Must-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \max\{age(b), age'(b)\}$

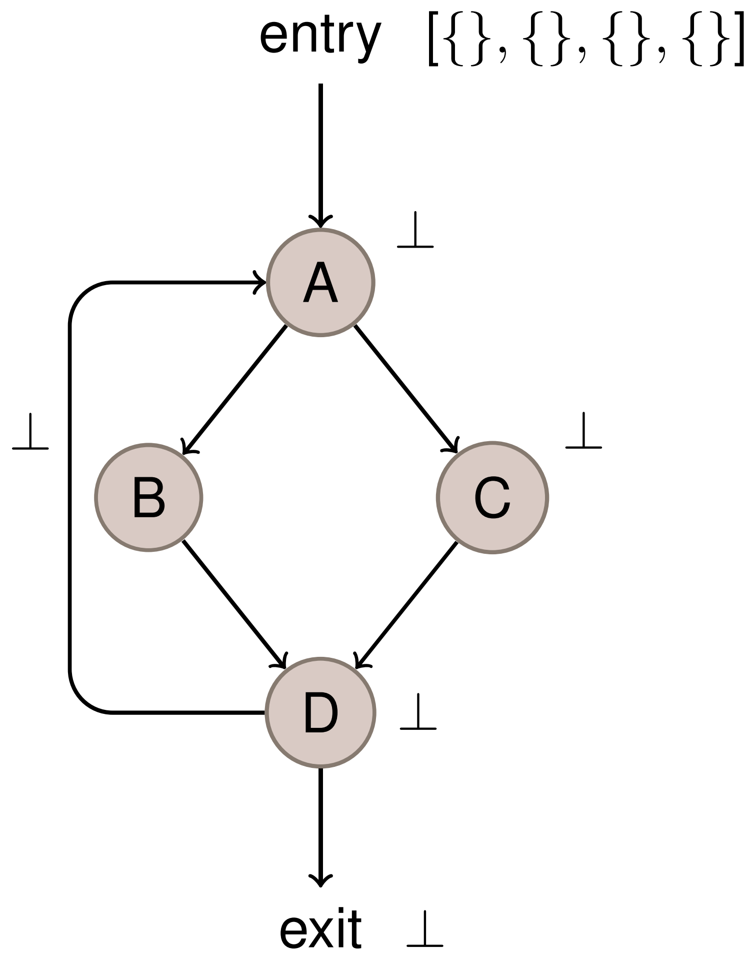
Graphically:

$$\begin{array}{|c|}\hline \{a\} \\ \hline \{\} \\ \hline \{c,f\} \\ \hline \{d\} \\ \hline \end{array} \sqcup \begin{array}{|c|}\hline \{c\} \\ \hline \{e\} \\ \hline \{a\} \\ \hline \{d\} \\ \hline \end{array} = \begin{array}{|c|}\hline \{\} \\ \hline \{\} \\ \hline \{a,c\} \\ \hline \{d\} \\ \hline \end{array}$$

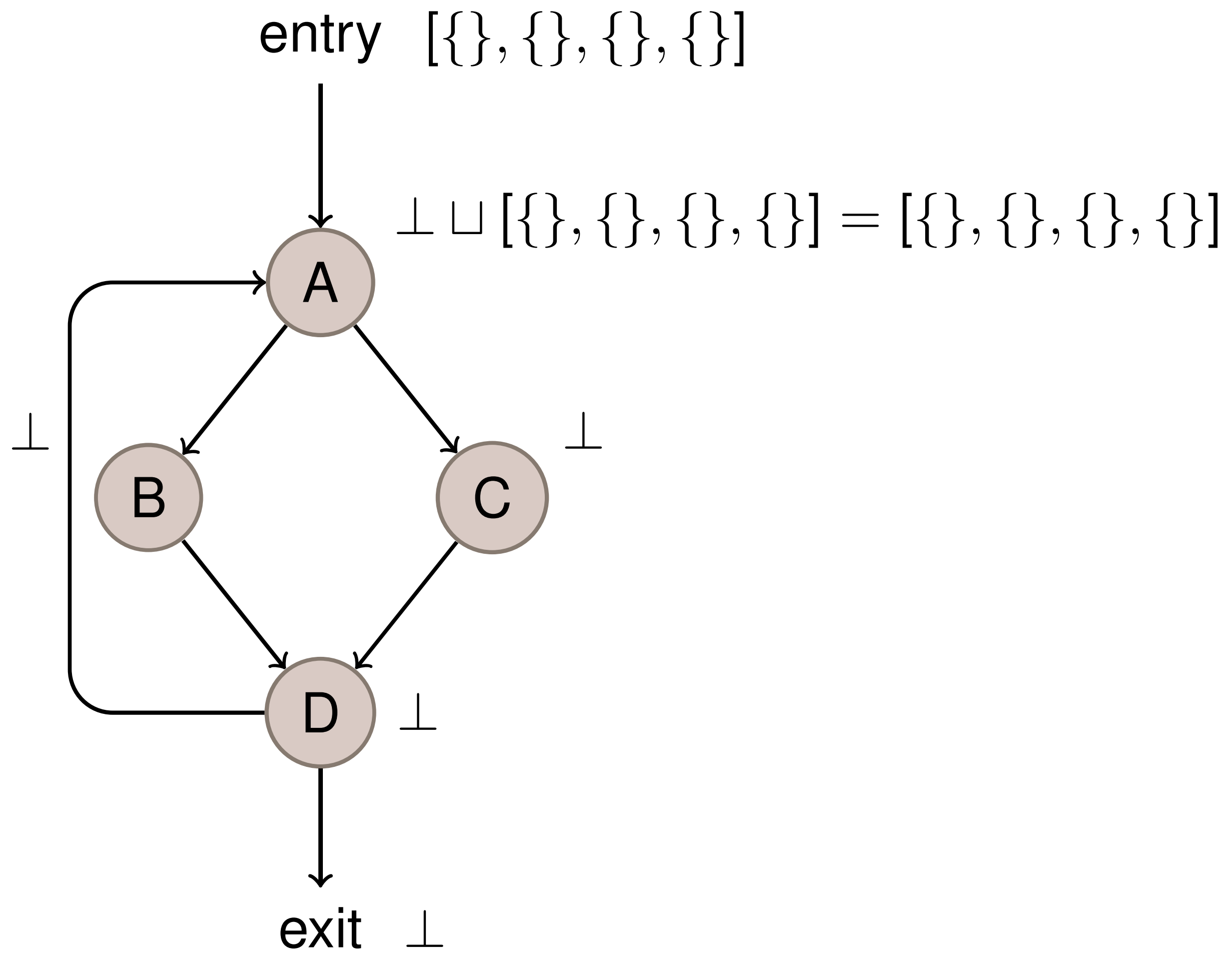
How many memory blocks can be in the must-cache? $\tilde{age}(\cdot) < k$

What about the ascending chain condition? ✓

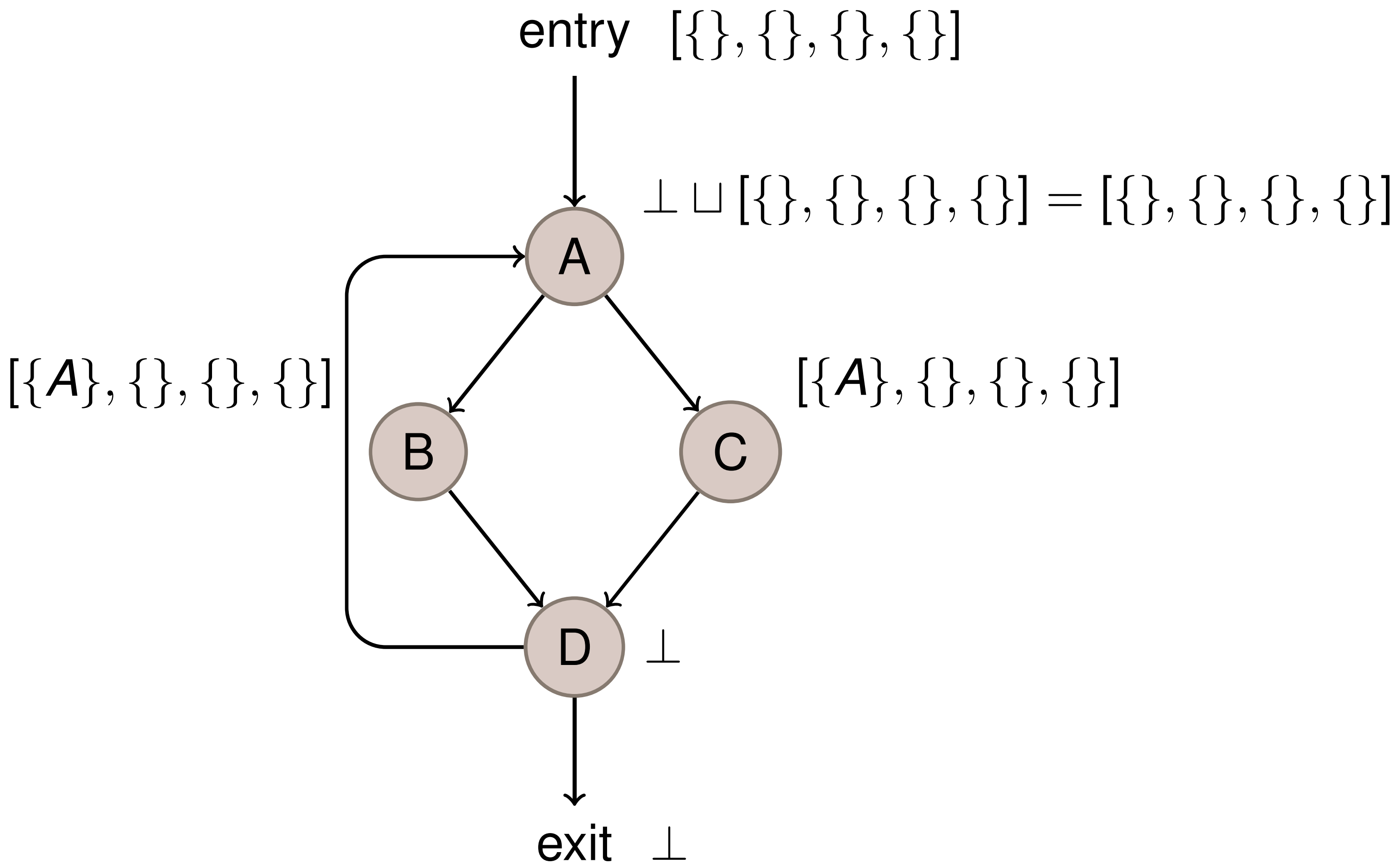
Example: Must-Analysis



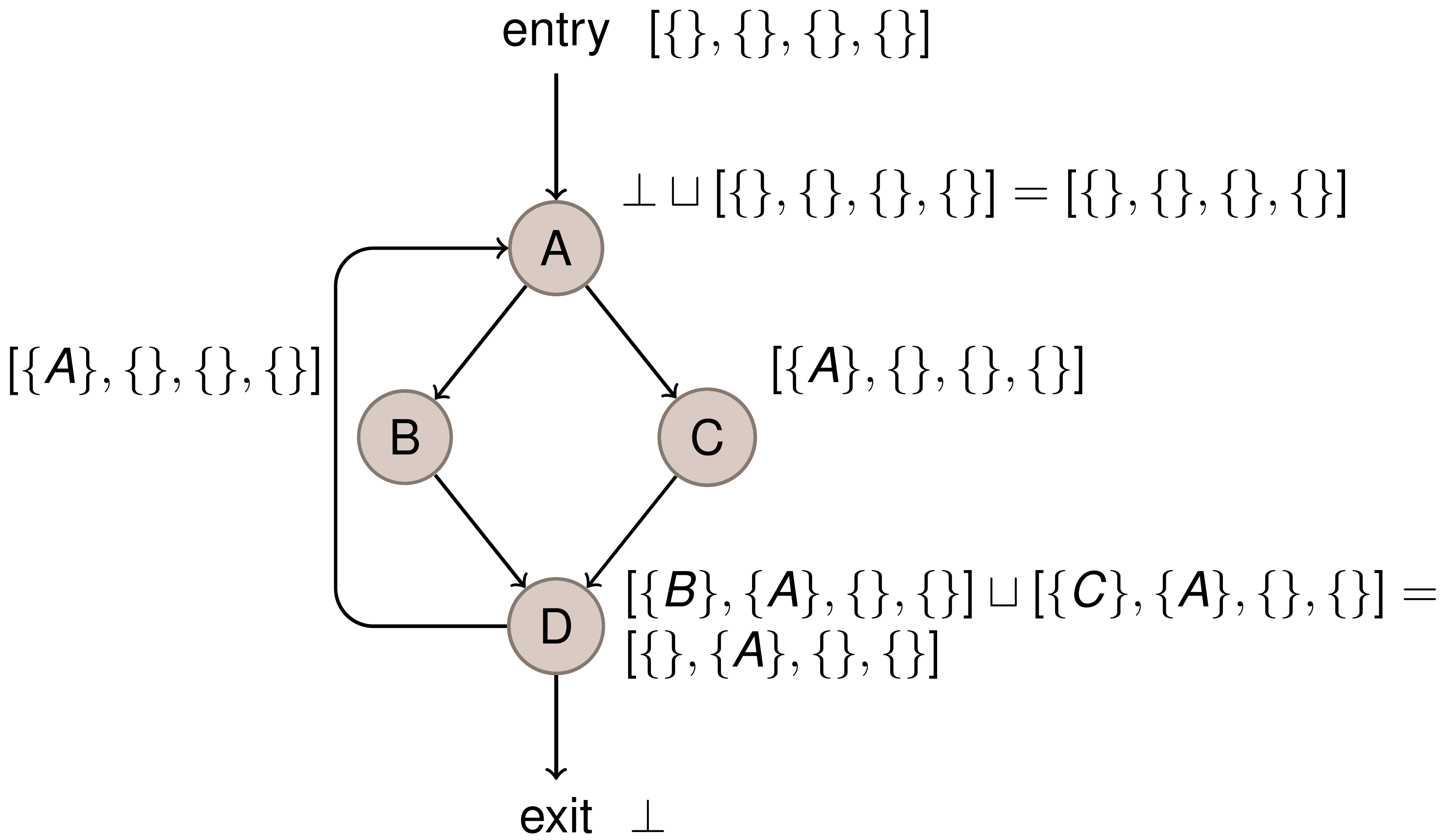
Example: Must-Analysis



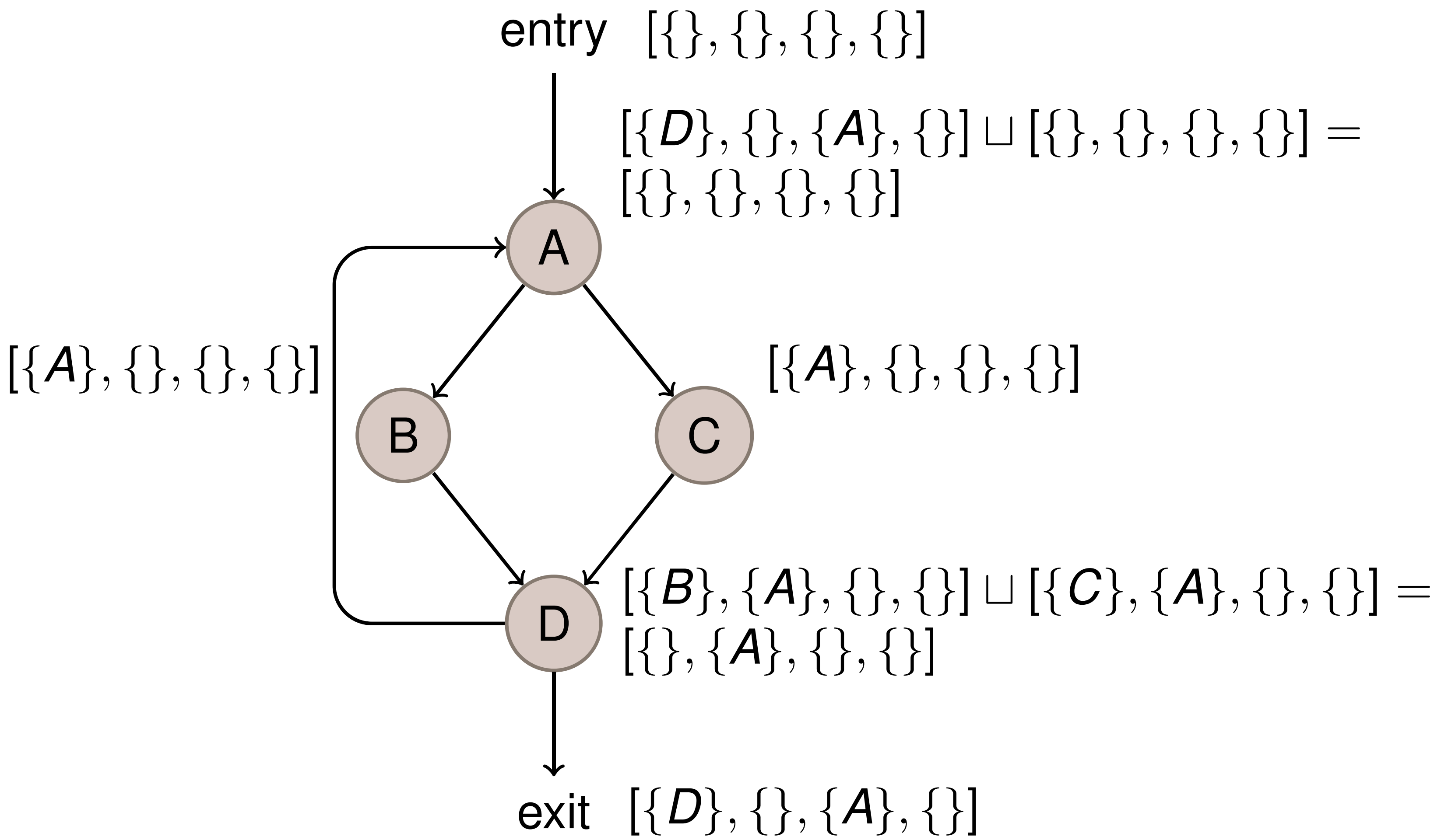
Example: Must-Analysis



Example: Must-Analysis



Example: Must-Analysis



No cache hits can be predicted :-(

Context-Sensitive Analysis/Virtual Loop-Unrolling

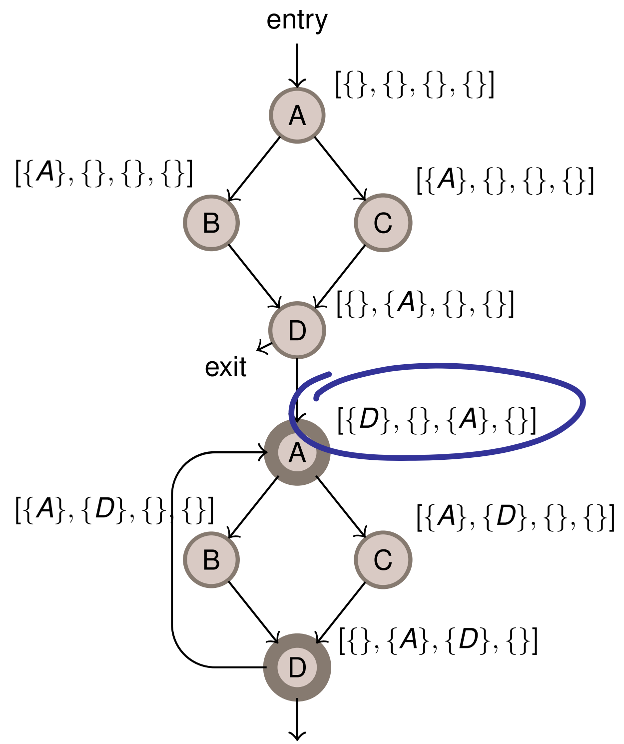
SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Problem:
  - The first iteration of a loop will always result in cache misses.
  - Similarly for the first execution of a function.
- Solution:
  - Virtually Peel 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



Outline

1 Caches

2 Cache Analysis for Least-Recently-Used
- Challenges
- Formalization of LRU and Logical Abstraction
- Must Analysis
- **May Analysis**
- Remaining Challenges

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

4 Summary

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

Abstract state:

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

...and its interpretation:

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

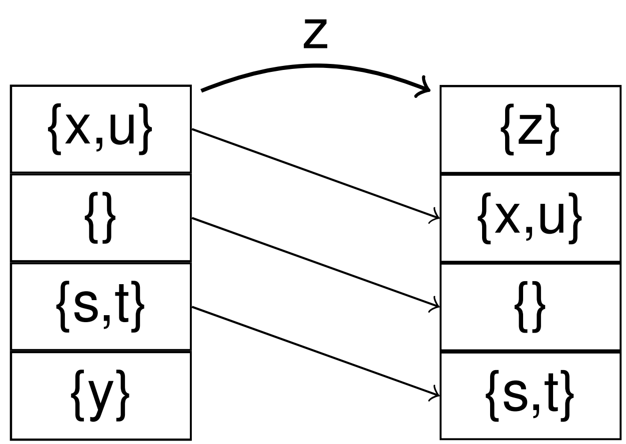
SAARLAND UNIVERSITY
COMPUTER SCIENCE

Formally:

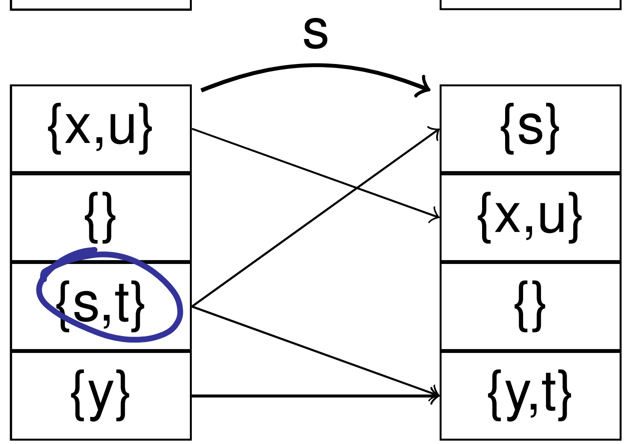
$$up_{may}(\widehat{age}, b) := \lambda b' \in \mathcal{B} : \begin{cases} 0 & : \textit{if } b' = b \\ age(b') & : \textit{if } \widehat{age}(b) < \widehat{age}(b') \\ age(b') + 1 & : \textit{if } \widehat{age}(b) \geq \widehat{age}(b') \neq k \\ k & : \textit{if } \widehat{age}(b') = k \end{cases}$$

LRU: May-Analysis: Update

“Definite Cache Miss”:



“Potential Cache Hit”:



Why does *t age* in the second case?

LRU: May-Analysis: Join

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \min\{age(b), age'(b)\}$

Graphically:

$$\begin{array}{|c|}\hline \{a,b\} \\ \hline \{\} \\ \hline \{c,f\} \\ \hline \{d\} \\ \hline \end{array} \sqcup \begin{array}{|c|}\hline \{c\} \\ \hline \{e\} \\ \hline \{a\} \\ \hline \{d\} \\ \hline \end{array} = \begin{array}{|c|}\hline \{a,b,c\} \\ \hline \{e\} \\ \hline \{f\} \\ \hline \{d\} \\ \hline \end{array}$$

"UNION" + "MINIMAL AGES"

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \min\{age(b), age'(b)\}$

Graphically:

{a,b}		{c}		{a,b,c}
{}	$\sqcup$	{e}	$=$	{e}
{c,f}		{a}		{f}
{d}		{d}		{d}

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \min\{age(b), age'(b)\}$

Graphically:

{a,b}		{c}		{a,b,c}
{}	$\bigsqcup$	{e}	$=$	{e}
{c,f}		{a}		{f}
{d}		{d}		{d}

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \min\{age(b), age'(b)\}$

Graphically:

$$\begin{array}{|c|}\hline \{a,b\} \\ \hline \{\} \\ \hline \{c,f\} \\ \hline \{d\} \\ \hline \end{array} \sqcup \begin{array}{|c|}\hline \{c\} \\ \hline \{e\} \\ \hline \{a\} \\ \hline \{d\} \\ \hline \end{array} = \begin{array}{|c|}\hline \{a,b,c\} \\ \hline \{e\} \\ \hline \{f\} \\ \hline \{d\} \\ \hline \end{array}$$

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \min\{age(b), age'(b)\}$

Graphically:

$$\begin{array}{|c|}\hline \{a,b\} \\ \hline \{\} \\ \hline \{c,f\} \\ \hline \{d\} \\ \hline \end{array} \quad \bigsqcup \quad \begin{array}{|c|}\hline \{c\} \\ \hline \{e\} \\ \hline \{a\} \\ \hline \{d\} \\ \hline \end{array} \quad = \quad \begin{array}{|c|}\hline \{a,b,c\} \\ \hline \{e\} \\ \hline \{f\} \\ \hline \{d\} \\ \hline \end{array}$$

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \min\{age(b), age'(b)\}$

Graphically:

$$\begin{array}{|c|}\hline \{a,b\} \\ \hline \{\} \\ \hline \{c,f\} \\ \hline \{d\} \\ \hline \end{array} \sqcup \begin{array}{|c|}\hline \{c\} \\ \hline \{e\} \\ \hline \{a\} \\ \hline \{d\} \\ \hline \end{array} = \begin{array}{|c|}\hline \{a,b,c\} \\ \hline \{e\} \\ \hline \{f\} \\ \hline \{d\} \\ \hline \end{array}$$

LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative (ensures γ is monotone):

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

Formally: $age \sqcup age' := \lambda b \in \mathcal{B} : \min\{age(b), age'(b)\}$

Graphically:

$$\begin{array}{|c|}\hline \{a,b\} \\ \hline \{\} \\ \hline \{c,f\} \\ \hline \{d\} \\ \hline \end{array} \;\sqcup\; \begin{array}{|c|}\hline \{c\} \\ \hline \{e\} \\ \hline \{a\} \\ \hline \{d\} \\ \hline \end{array} \;=\; \begin{array}{|c|}\hline \{a,b,c\} \\ \hline \{e\} \\ \hline \{f\} \\ \hline \{d\} \\ \hline \end{array}$$

How many memory blocks can be in the must-cache?
What about the ascending chain condition?

Outline

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1 Caches

2 Cache Analysis for Least-Recently-Used
- Challenges
- Formalization of LRU and Logical Abstraction
- Must Analysis
- May Analysis
- Remaining Challenges

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

4 Summary

Accounting for Uncertainty about Memory Addresses

For instruction-cache analysis, there is no uncertainty about the accessed memory blocks. But what about data accesses?

So far: $up(\widehat{age}, x) : \underline{\widehat{\text{Cache}} \times \mathcal{B}} \rightarrow \widehat{\text{Cache}}$.

With uncertainty: $up(\widehat{age}, X) : \widehat{\text{Cache}} \times \underline{2^{\mathcal{B} \cup \{\perp\}}} \rightarrow \widehat{\text{Cache}}$.

Accounting for Uncertainty about Memory Addresses

For instruction-cache analysis, there is no uncertainty about the accessed memory blocks. But what about data accesses?

So far: $\underline{up(\widehat{age}, x)} : \widehat{Cache} \times \mathcal{B} \rightarrow \widehat{Cache}$.

With uncertainty: $up(\widehat{age}, X) : \widehat{Cache} \times 2^{\mathcal{B} \cup \{\perp\}} \rightarrow \widehat{Cache}$.

How to define $up(\widehat{age}, X)$?

$$up(\widehat{age}, X) := \bigsqcup \{ up(\widehat{age}, x) \mid x \in X \}$$

Accounting for Uncertainty about Memory Addresses

For instruction-cache analysis, there is no uncertainty about the accessed memory blocks. But what about data accesses?

So far: $up(\widehat{age}, x) : \widehat{\text{Cache}} \times \mathcal{B} \to \widehat{\text{Cache}}$.

With uncertainty: $up(\widehat{age}, X) : \widehat{\text{Cache}} \times 2^{\mathcal{B} \cup \{\perp\}} \to \widehat{\text{Cache}}$.

How to define $up(\widehat{age}, X)$?

$$up(\widehat{age}, X) := \bigsqcup \left\{ up(\widehat{age}, x) \mid x \in X \right\} \text{ where } up(\widehat{age}, \perp) := \widehat{age}$$

Practical effect?

Extension to Set-Associative Caches

One view: Set-associative cache = array of fully-associative caches

$$up_{set}(\widehat{s}, x) := \widehat{s}[index(x) \mapsto up(\widehat{s}(index(x)), x)],$$

where $index(x)$ determines the set number of block x.

Uncertainty can be accounted for as described before.

$\longrightarrow$ Excursion: Relational Cache Analysis

Extension to Set-Associative Caches

SAARLAND UNIVERSITY COMPUTER SCIENCE

One view: Set-associative cache = array of fully-associative caches

$$up_{set}(\widehat{s}, x) := \widehat{s}[index(x) \mapsto up(\widehat{s}(index(x)), x)],$$

where $index(x)$ determines the set number of block x.

Uncertainty can be accounted for as described before.

Effect of address uncertainty on results?

$\longrightarrow$ Excursion: Relational Cache Analysis

Data Caches: Write-through vs Write-back

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Two different *write policies*:

Write-through: Every write goes directly to main memory.

Write-back: Cache keeps track of “dirty” cache lines and writes back changes to main memory only when cache line is evicted.

What is preferable in terms of performance/analysis?