Persistence Analysis Reloaded

Christoph Cullmann

Department of Computer Science
Saarland University

AbsInt Angewandte Informatik GmbH

July 2009

Motivation

How to do precise cache analysis for this loop?

```
void alternatingLoop (int maxRounds)
{
  while (int i = 0; i < maxRounds; ++i) {
    if (someThingUnknown()) {
      accessA ();
    } else {
      accessB ();
    }
  }
}
```

Cache & Analysis Parameters

SAARLAND UNIVERSITY
COMPUTER SCIENCE

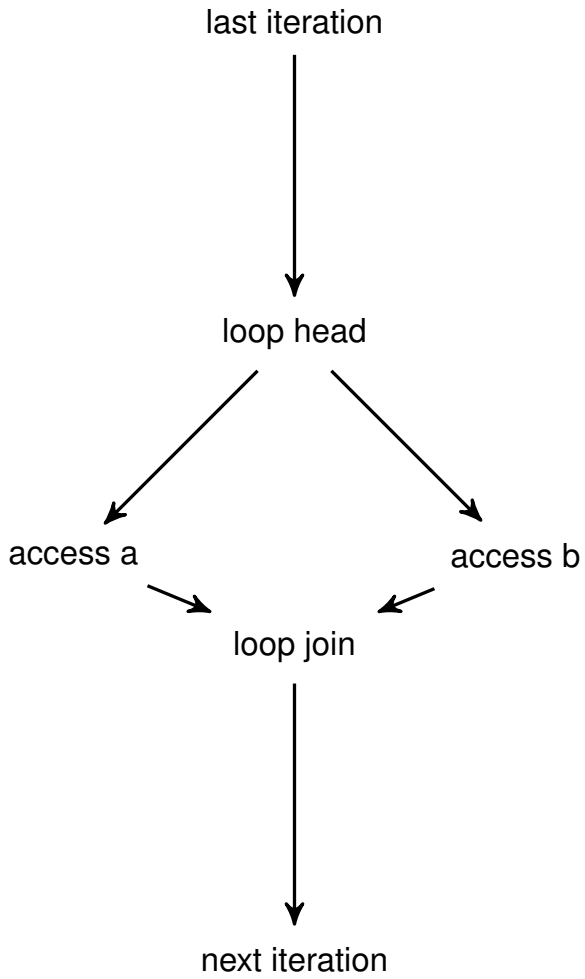
Cache Parameters

- LRU Replacement Policy
- 2-way associative

Analysis Parameters

- Analysing only one set, as the sets are independent.
- accessA will read cache line a, accessB cache line b.
- a and b map to same set, the analysed one.

Simplified Control Flow for Example Loop



First Idea: Use Must Analysis

Must Analysis Basics

- Under-approximation of cache contents.
- Maps cache lines to their maximal age in the cache.
- Allows to classify sure-hits.

Must Analysis Theory

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Update Function

$$U_{must}(m,x) = \begin{cases} l_1 \mapsto \{x\} \\ l_i \mapsto m(l_{i-1}) \mid i = 2 \,..\, h-1 \\ l_h \mapsto m(l_{h-1}) \cup (m(l_h) \setminus \{x\}) \\ l_i \mapsto m(l_i) \mid i = h+1 \,..\, A \end{cases} \quad \text{if } \exists l_h : x \in m(l_h)$$

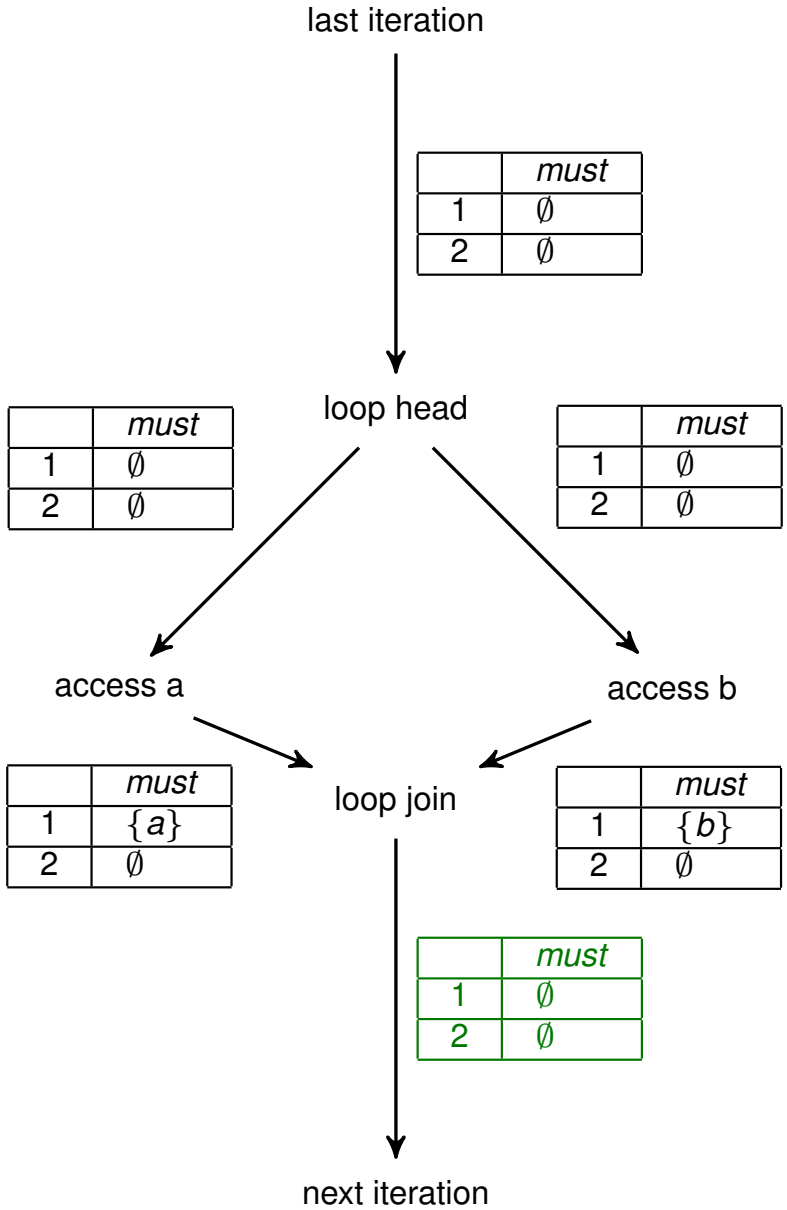
$$\begin{cases} l_1 \mapsto \{x\} \\ l_i \mapsto m(l_{i-1}) \mid i = 2 \,..\, A \end{cases} \quad \text{otherwise}$$

Must Analysis Theory

Join Function

$$J_{must}(m, m') = l_i \mapsto \{x \mid \exists l_a, l_b : x \in m(l_a) \wedge x \in m'(l_b) \wedge i = \max(a, b)\}$$

Must Analysis applied to Example



Results of Must Analysis

Results

- No accesses to cache line a or b classified as sure-hits.
- Therefore: Later Analysis must assume both miss and hit case.

Second Idea: Use May Analysis

SAARLAND UNIVERSITY
COMPUTER SCIENCE

May Analysis Basics

- Over-approximation of cache contents.
- Maps cache lines to their minimal age in the cache.
- Allows to classify sure-misses.

May Analysis Theory

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Update Function

$$U_{may}(m, x) = \begin{cases} l_1 \mapsto \{x\} \\ l_i \mapsto m(l_{i-1}) \mid i = 2 \,..\, h \\ l_{h+1} \mapsto m(l_{h+1}) \cup (m(l_h) \setminus \{x\}) \\ l_i \mapsto m(l_i) \mid i = h+2 \,..\, A \end{cases} \quad \text{if } \exists l_h : x \in m(l_h)$$

$$\begin{cases} l_1 \mapsto \{x\} \\ l_i \mapsto m(l_{i-1}) \mid i = 2 \,..\, A \end{cases} \quad \text{otherwise}$$

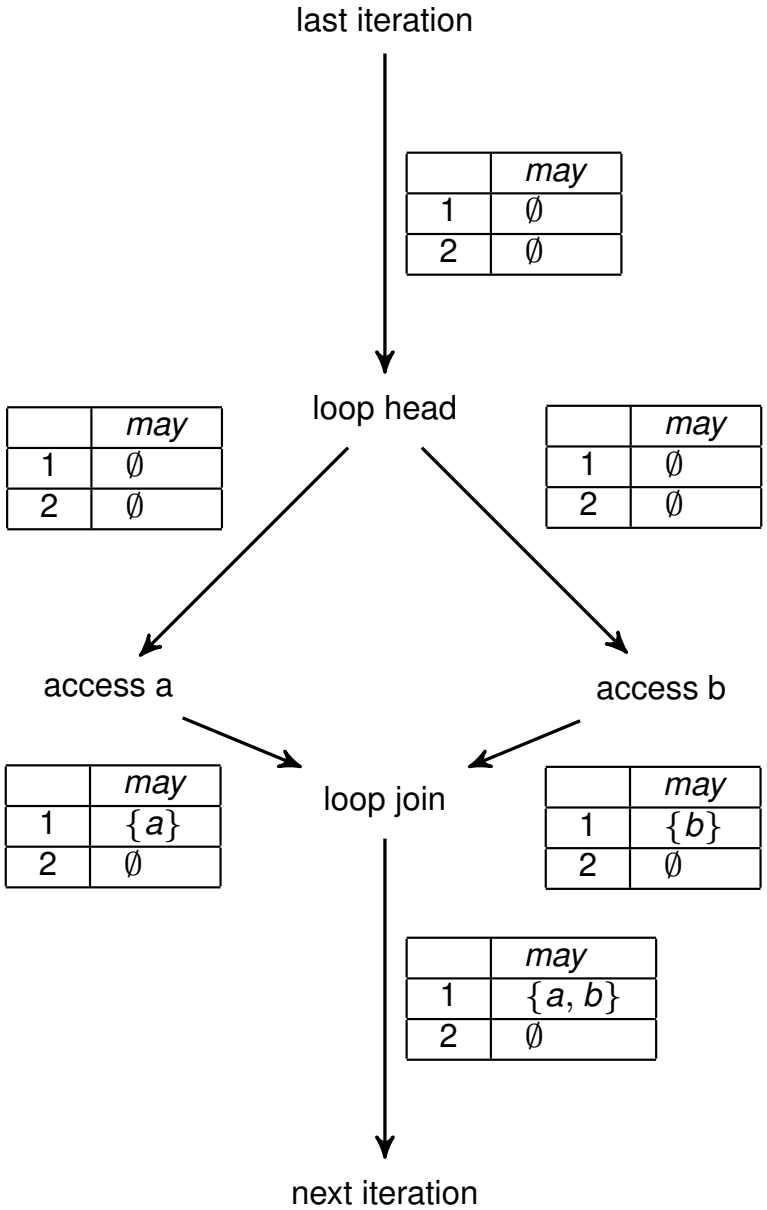
May Analysis Theory

SAARLAND UNIVERSITY
COMPUTER SCIENCE

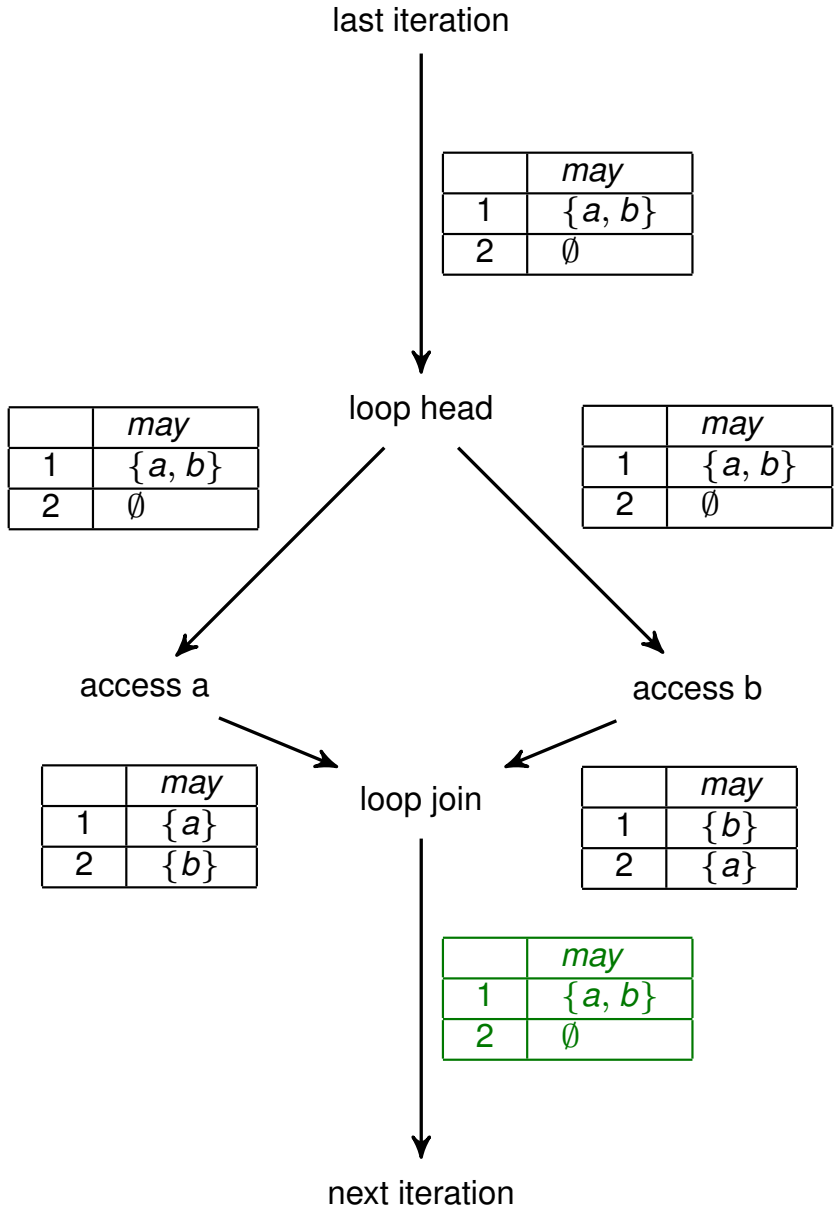
Join Function

$$J_{may}(m, m') = l_i \mapsto \begin{array}{ll} & \{x \mid \exists l_a, l_b : x \in m(l_a) \\ & \quad \wedge x \in m'(l_b) \wedge i = \min(a, b)\} \\ \cup & \{x \mid x \in m(l_i) \wedge \nexists l_a : x \in m'(l_a)\} \\ \cup & \{x \mid x \in m'(l_i) \wedge \nexists l_a : x \in m(l_a)\} \end{array}$$

Example - May Analysis, 1st Iteration



Example - May Analysis, Fixpoint



Results of May Analysis

Results

- No accesses to cache line a or b classified as sure-misses.
- Therefore: Same as after must analysis, later Analysis must assume both miss and hit case.

Last Escape: Using Persistence Analysis

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Basic Idea

- Persistence analysis tries to calculate if a cache line can no be evicted in a given scope.
- Using this persistence classification, only the first access to such a classified cache line will be eventually a miss, all following a hit.

Intuition for Example

- Natural scope: The loop itself.
- Only 2 cache lines of the analysed set read.
- The associativity is 2.
- Both cache lines should be persistent!

Persistence Analysis of Ferdinand

Basics of the Analysis

- Based on must analysis.
- Uses same aging as must analysis.
- Union with maximization of ages instead of intersection as join function.
- Introduction of an additional age, to keep track of lines possibly evicted.

Persistence Analysis Theory

Update Function

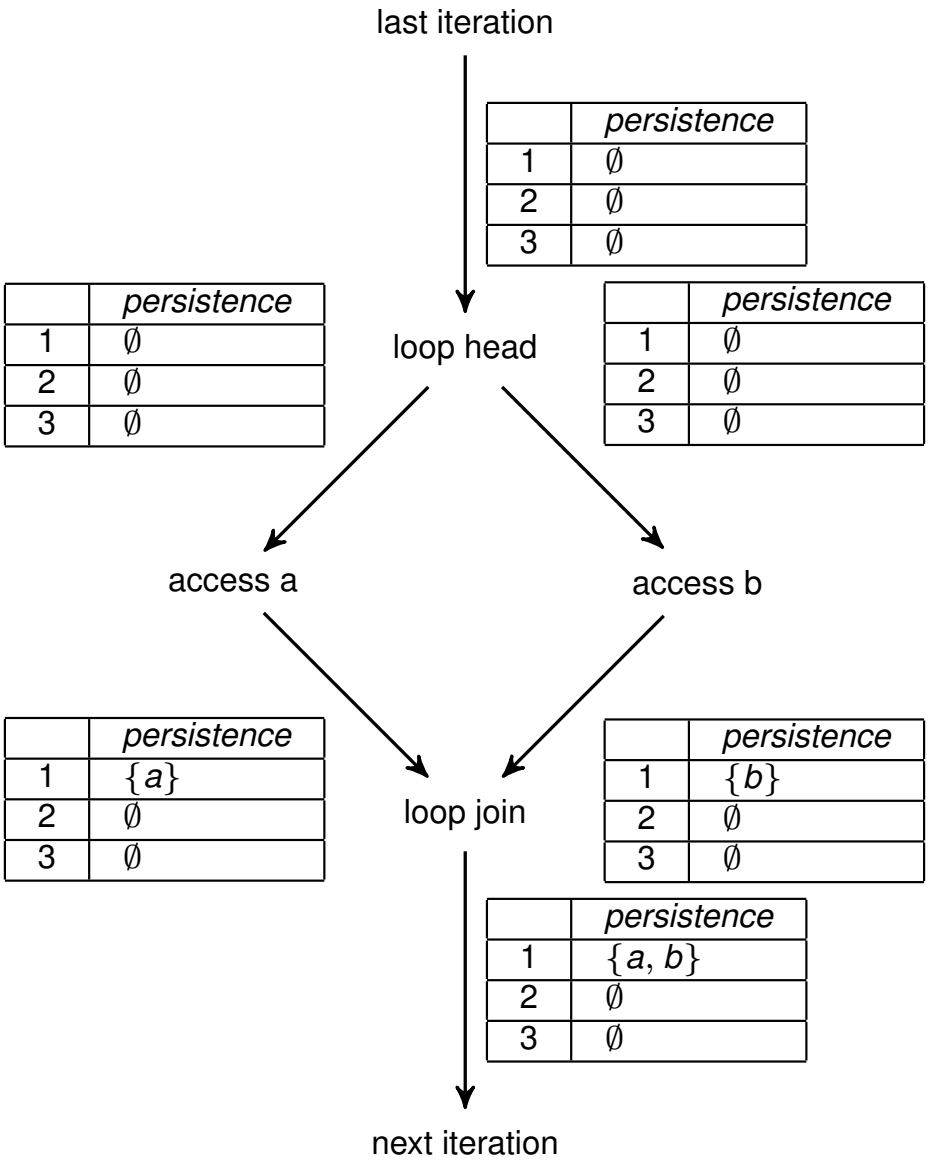
$$U_{pers}(m,x) = \begin{cases} l_1 \mapsto \{x\} \\ l_i \mapsto m(l_{i-1}) \mid i = 2 \,..\, h-1 \\ l_h \mapsto m(l_{h-1}) \cup (m(l_h) \setminus \{x\}) & \text{if } \exists h \in \{1,...,A\} : x \in m(l_h) \\ l_i \mapsto m(l_i) \mid i = h+1 \,..\, A \\ l_{A+1} \mapsto m(l_{A+1}) \\ \\ l_1 \mapsto \{x\} \\ l_i \mapsto m(l_{i-1}) \mid i = 2 \,..\, A & \text{otherwise} \\ l_{A+1} \mapsto m(l_A) \cup (m(l_{A+1}) \setminus \{x\}) \end{cases}$$

Persistence Analysis Theory

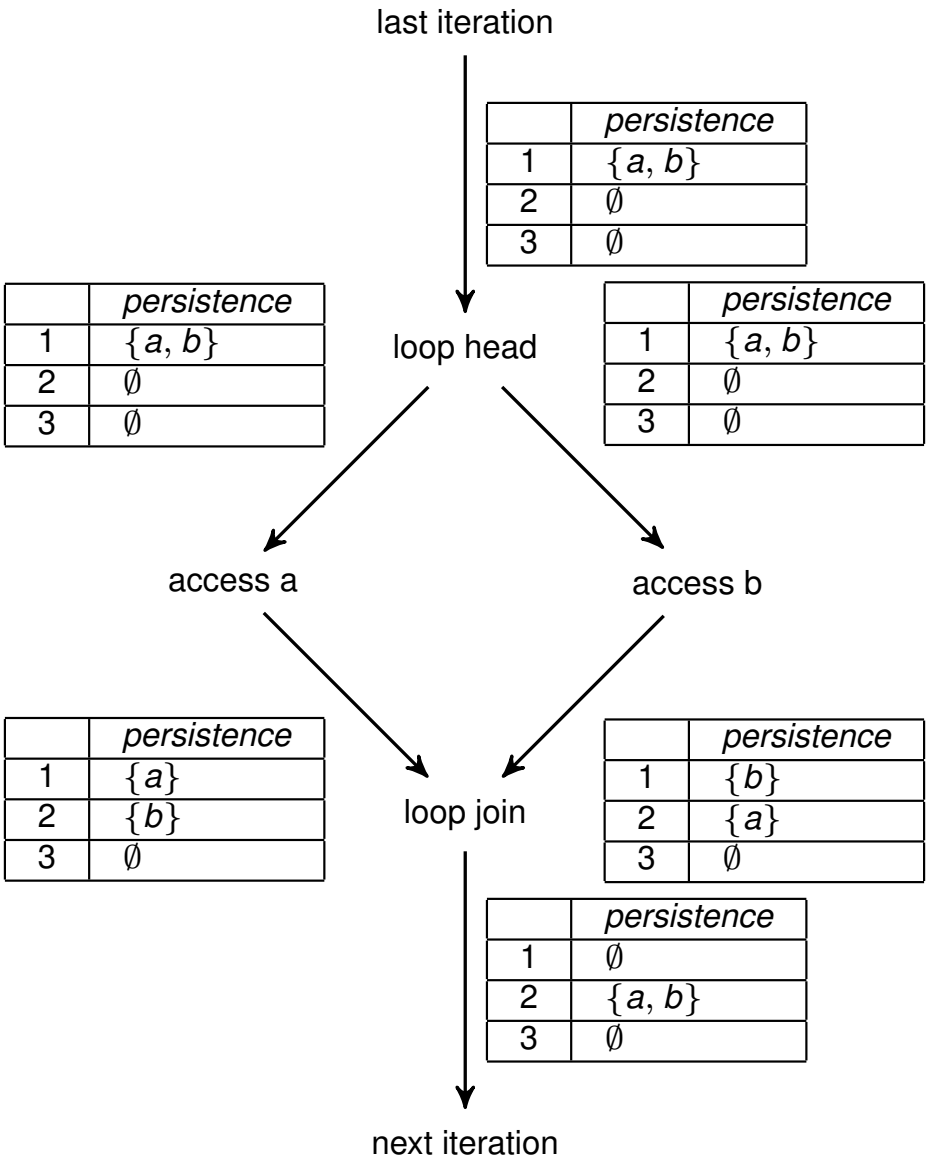
Join Function

$$J_{pers}(m, m') = l_i \mapsto \begin{array}{ll} & \{x \mid \exists l_a, l_b : x \in m(l_a) \\ & \quad \wedge x \in m'(l_b) \wedge i = \max(a, b)\} \\ \cup & \{x \mid x \in m(l_i) \wedge \nexists l_a : x \in m'(l_a)\} \\ \cup & \{x \mid x \in m'(l_i) \wedge \nexists l_a : x \in m(l_a)\} \end{array}$$

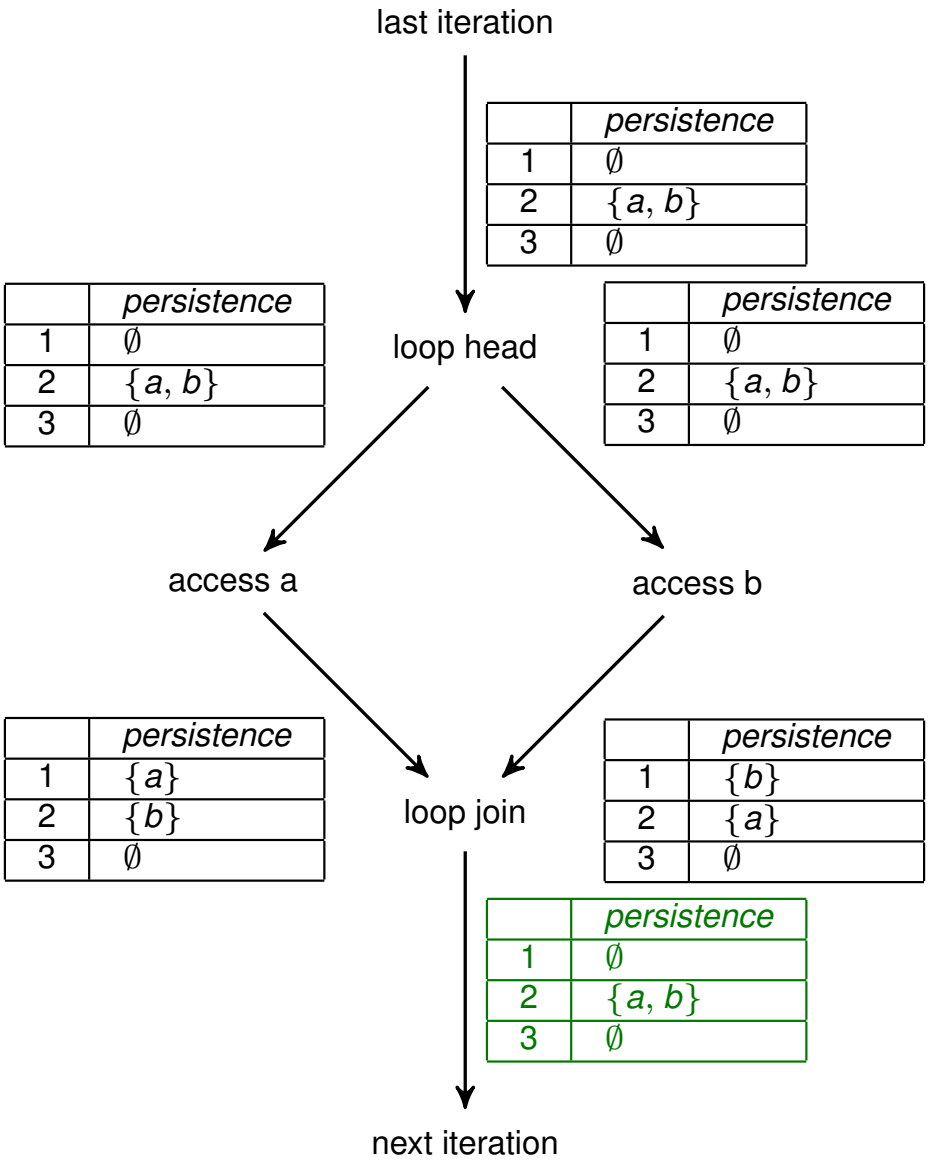
Example - Persistence Analysis, 1st Iteration



Example - Persistence Analysis, 2nd Iteration



Example - Persistence Analysis, Fixpoint



Results of Persistence Analysis

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Results

- Both cache line a and cache line b can not be evicted inside the loop (they never get the age 3).
- Both accesses can be classified as persistent.

Then, where is the problem?

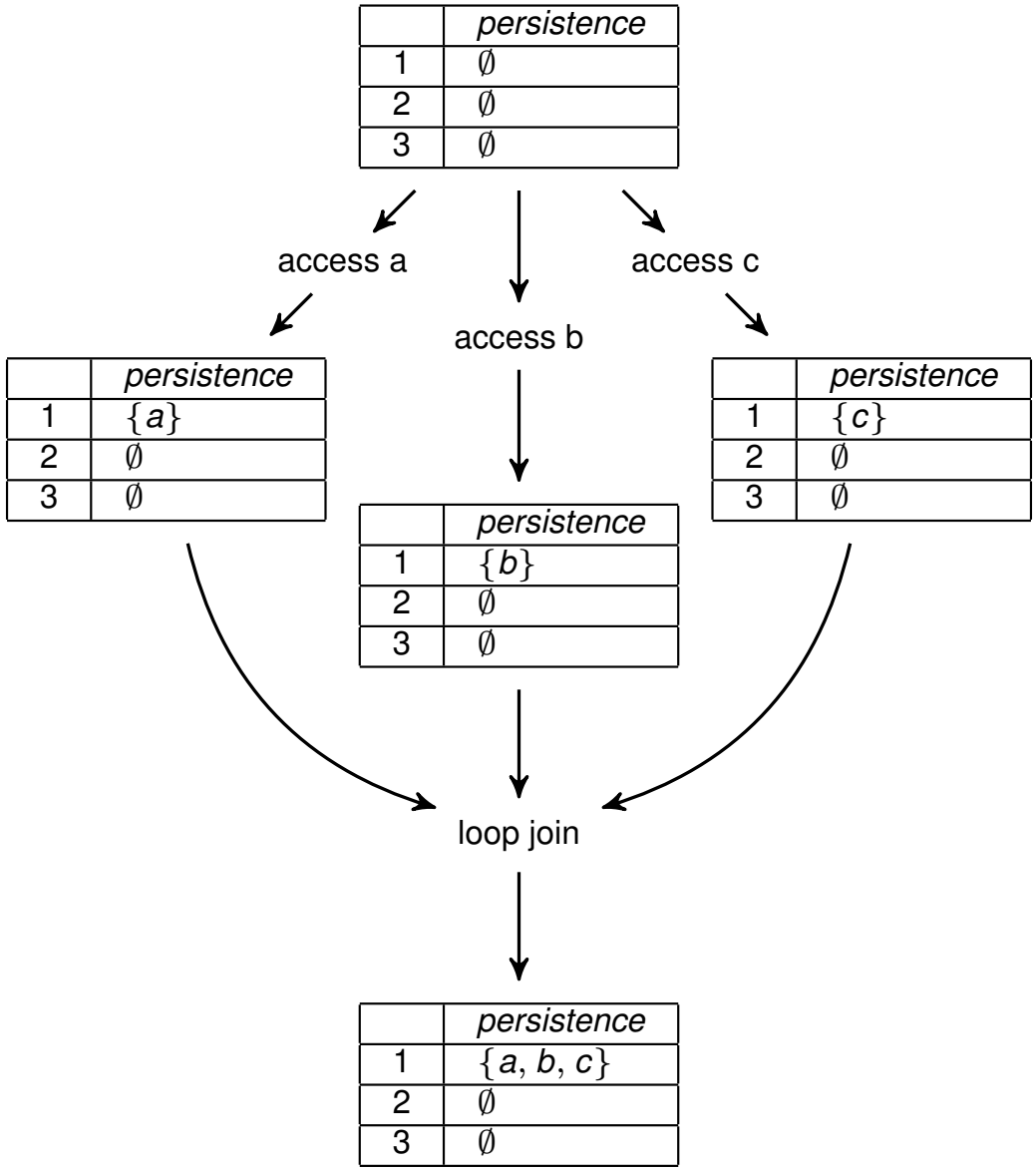
Problem with analysis discovered by Hugues Cassé.

Problem found by Hugues Cassé

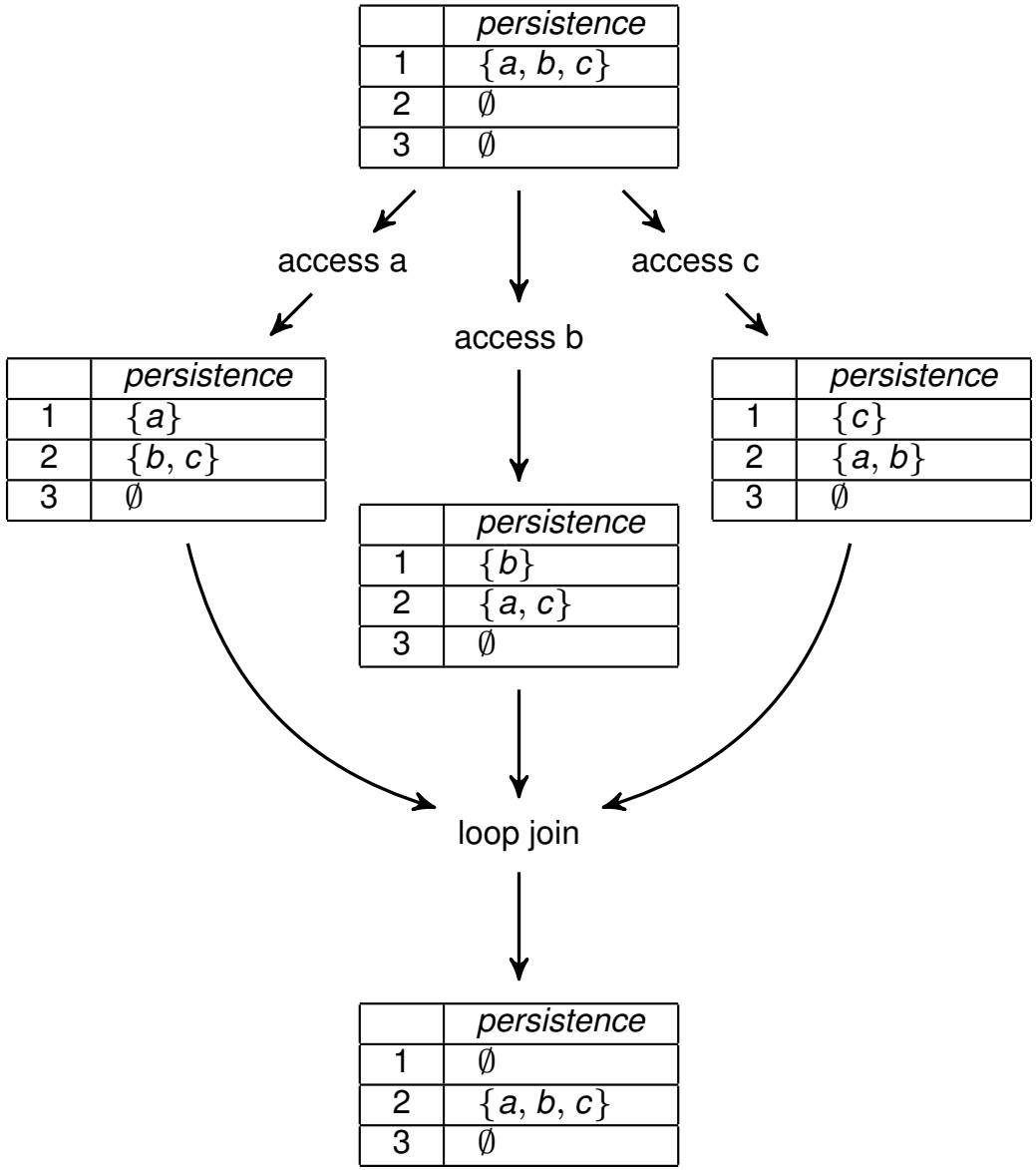
Addition of third possibility to our example:

```
void alternatingLoop (int maxRounds)
{
  while (int i = 0; i < maxRounds; ++i) {
    switch (someThingUnknown()) {
      case a:
        accessA (); break;
      case b:
        accessB (); break;
      default:
        accessC (); break;
    }
  }
}
```

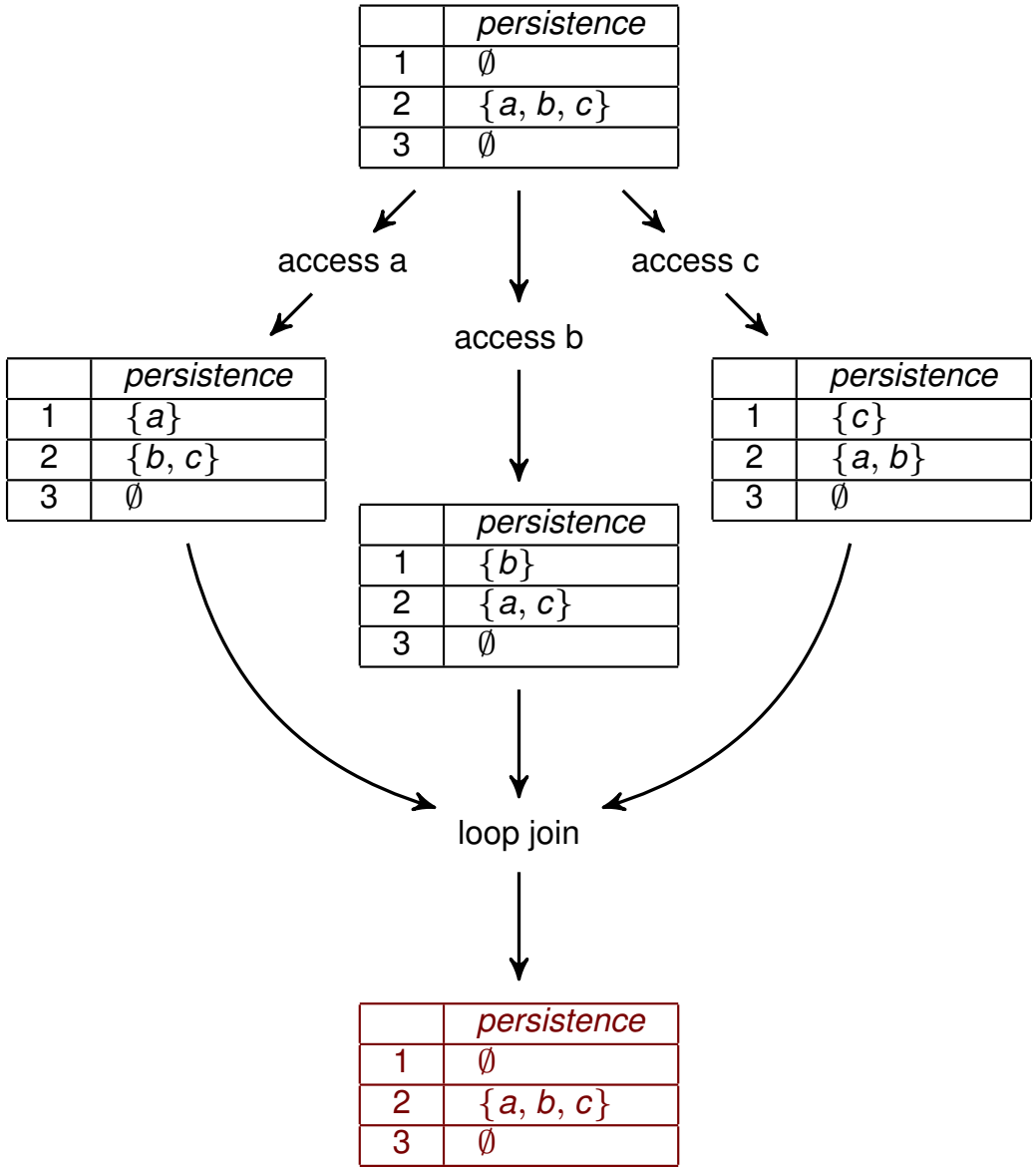
Problem Hugues Cassé, 1st Iteration



Problem Hugues Cassé, 2nd Iteration



Problem Hugues Cassé, Fixpoint



Results of Persistence Analysis

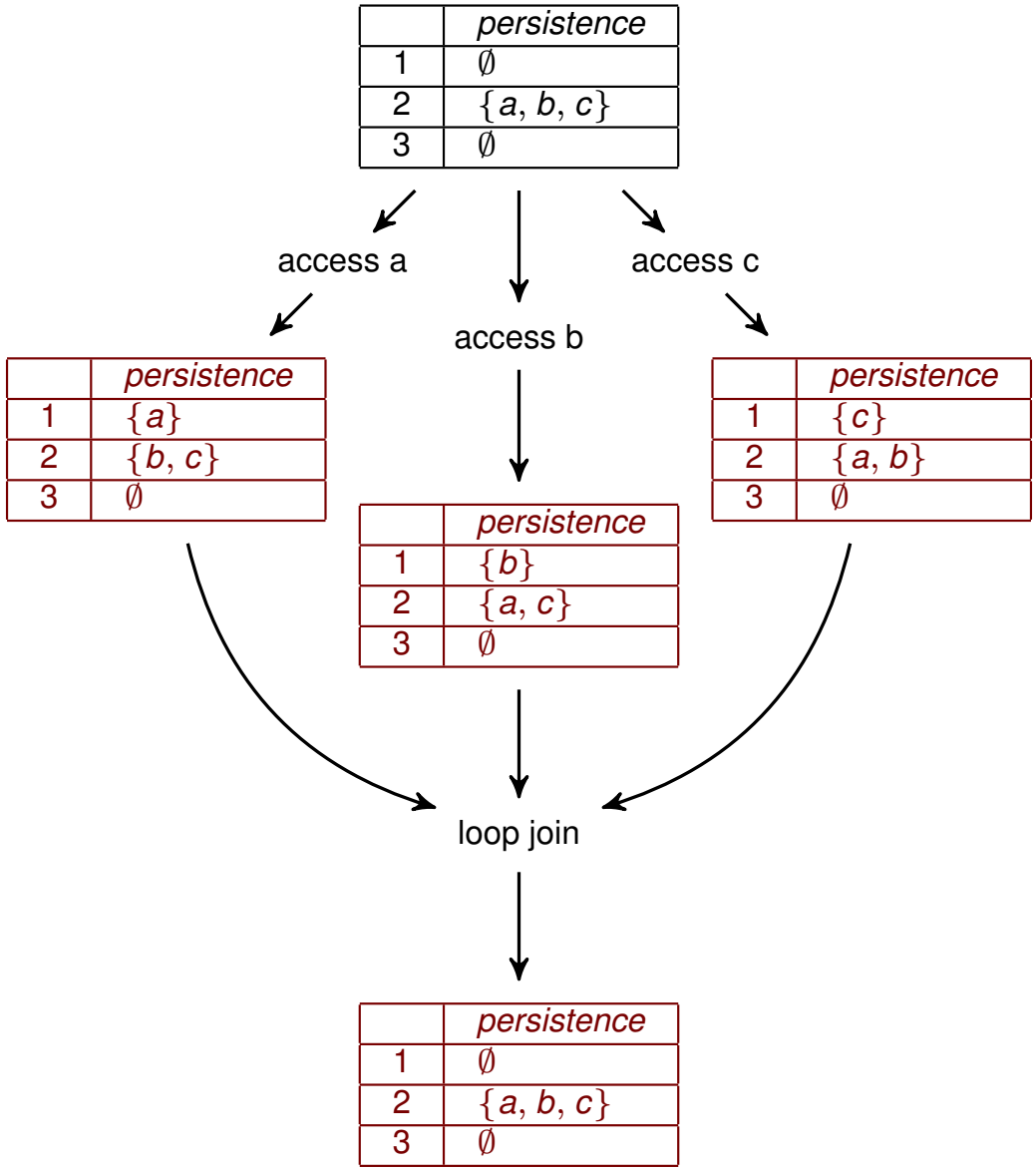
Problematic:

- All three cache lines a, b and c can not be evicted inside the loop (they never get the age 3)
- All three accesses can be classified as persistent
- This is wrong, as three elements don't fit in the 2 element large set!

Where is the error?

- Aging is not correct, persistence can't use the same aging as must analysis!
- Reason: persistence analysis is no under-approximation of the cache, no guarantee to be in the cache

Problem Hugues Cassé, Aging Error

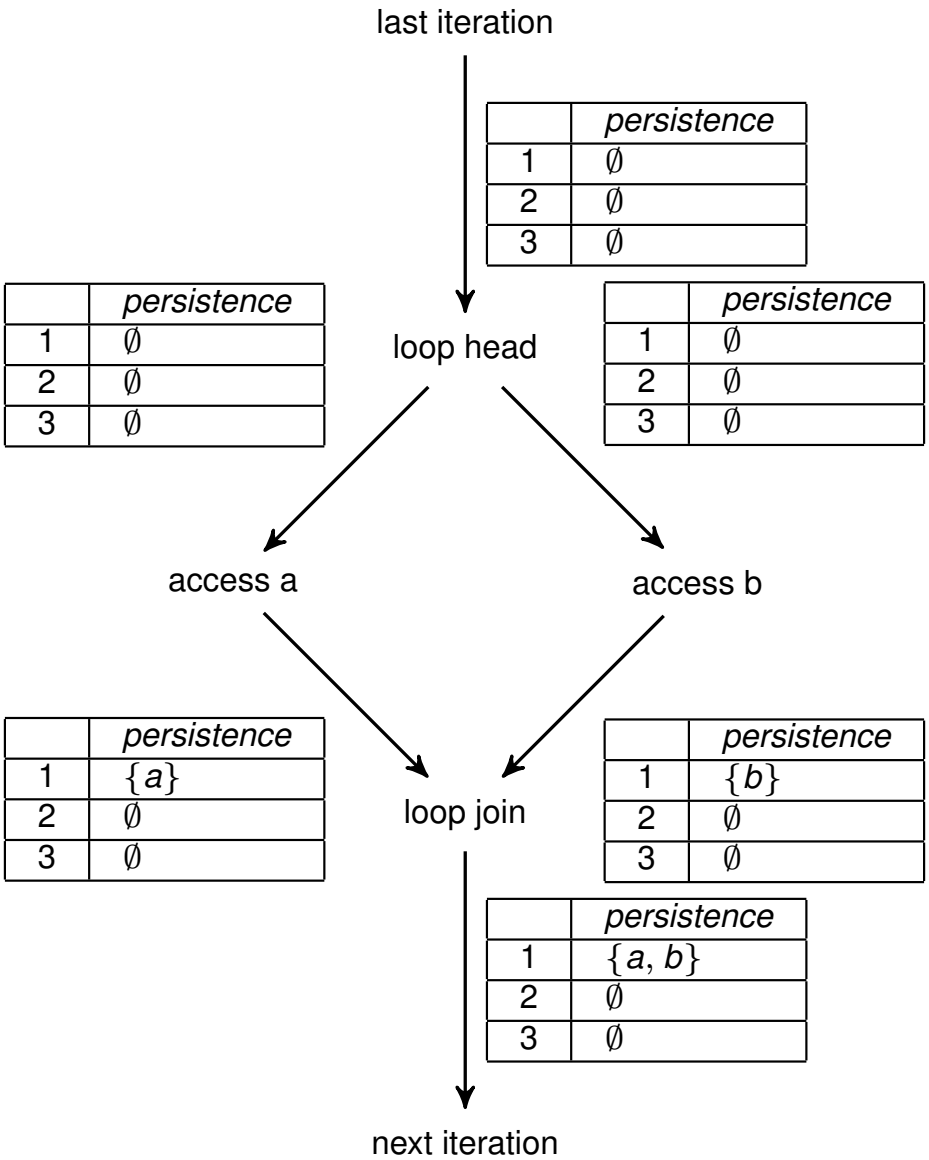


First Possible Solution

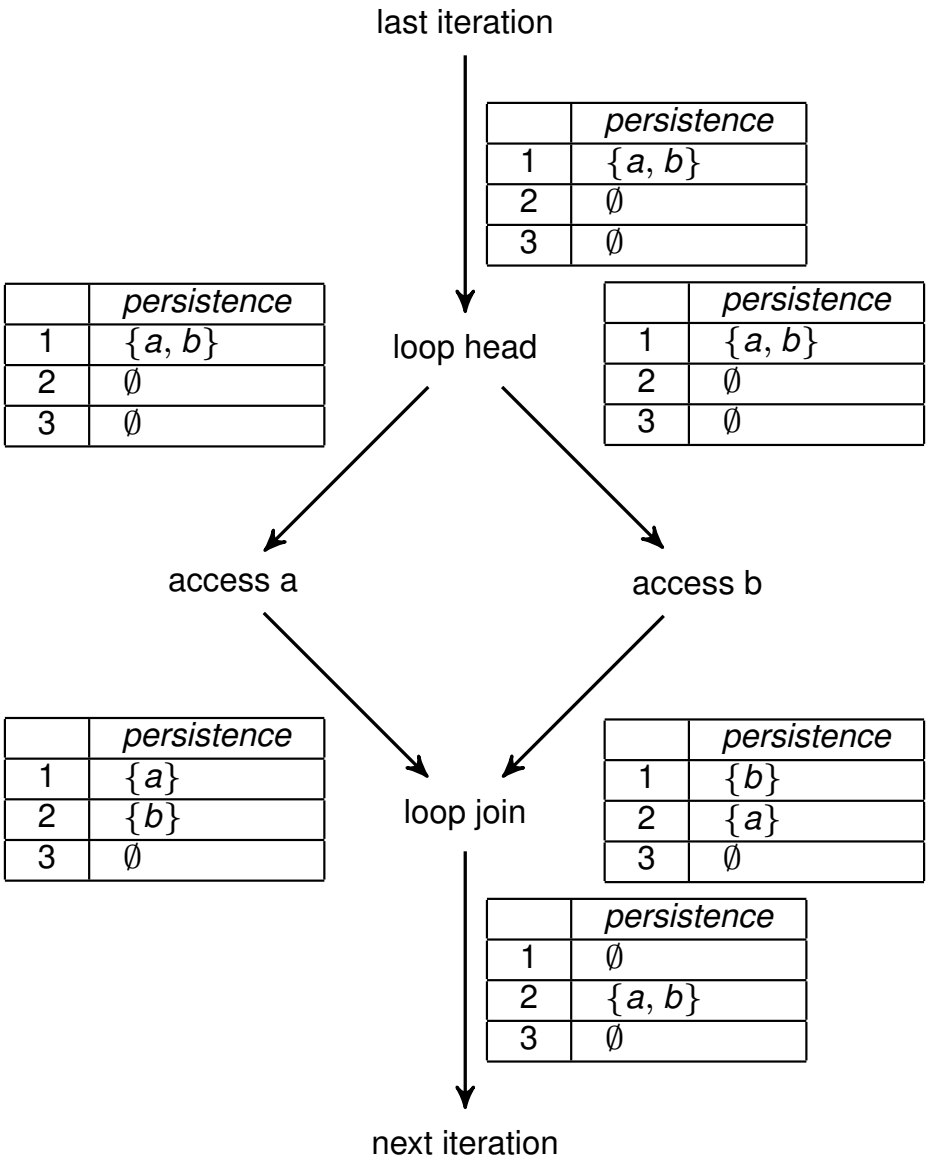
Fixing the Aging

- On access to any element, all other elements will age. This is needed, as we need maximal ages.
- This fixes the problem shown by Cassé
- But: this does not even allow the first example to work

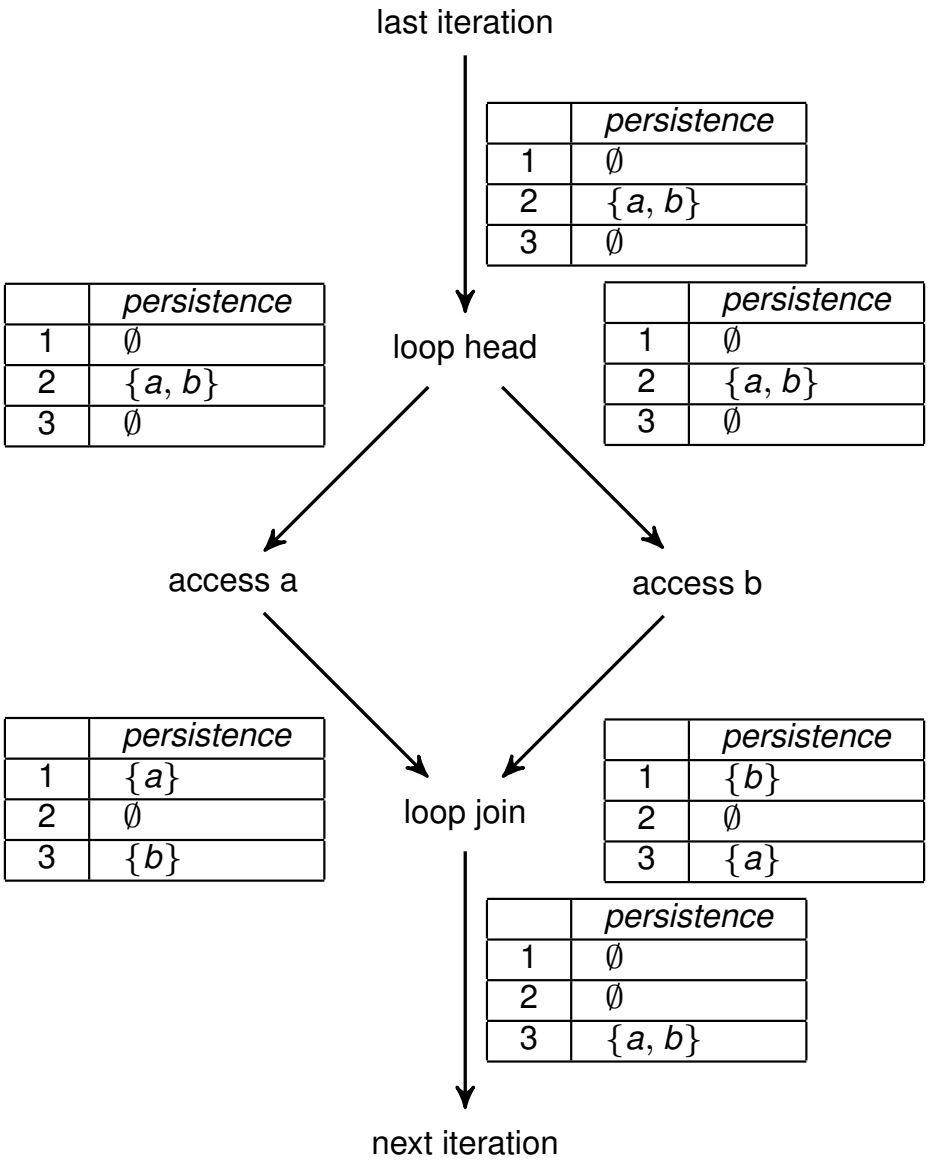
Example - New Aging, 1st Iteration



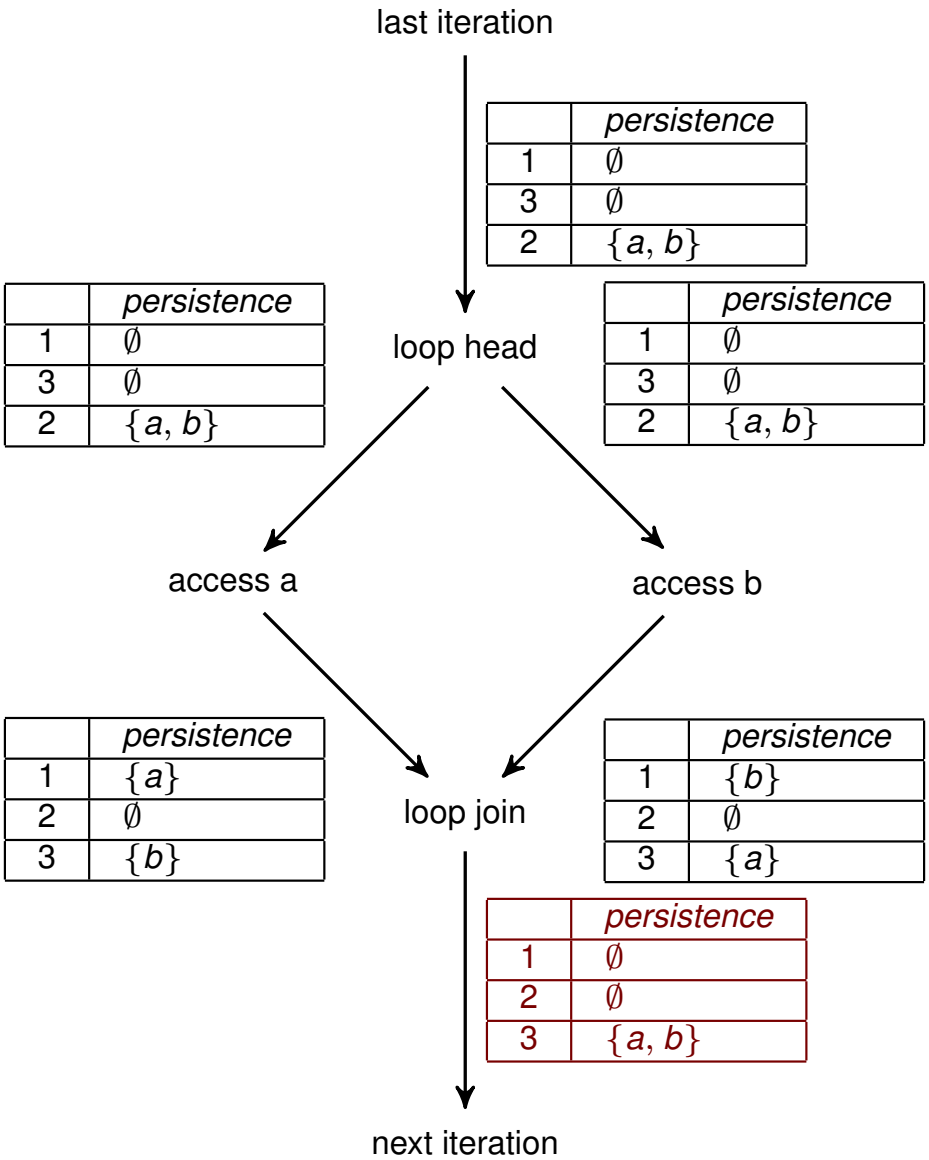
Example - New Aging, 2nd Iteration



Example - New Aging, 3rd Iteration



Example - New Aging, Fixpoint



Second Possible Solution

Introducing New Analysis

- Learn out of the error of the old analysis.
- Not must-analysis based, but may-analysis based.

New Analysis - Introduction

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Basics

- Parallel may- and may-max-analysis.
  may-max ($\widehat{may}$) = may with maximal ages, additional age for possibly evicted elements.
- Using of the may-analysis to bound the number of elements in cache.
- Use of the may-max-analysis to calculate the possible evictions.
- Important: Eviction only possible, if the cache is full.
  This works only for LRU!

Update Function

$$U_{pers}((m,\hat{m}),x) = (U_{may}(m,x),\ U_{\widehat{may}}(\hat{m},m,x))$$

$$U_{\widehat{may}}(\hat{m},m,x) = \begin{cases} l_1 \mapsto \{x\} \\ l_i \mapsto \hat{m}(l_{i-1}) \setminus \{x\} \mid i = 2\,..\,A \\ l_{A+1} \mapsto (\hat{m}(l_{A+1}) \cup \hat{m}(l_A)) \setminus \{x\} \end{cases} \quad \text{if mayevict}(m,x)$$

$$\begin{cases} l_1 \mapsto \{x\} \\ l_i \mapsto \hat{m}(l_{i-1}) \setminus \{x\} \mid i = 2\,..\,A-1 \\ l_A \mapsto (\hat{m}(l_A) \cup \hat{m}(l_A - 1)) \setminus \{x\} \\ l_{A+1} \mapsto \hat{m}(l_{A+1}) \setminus \{x\} \end{cases} \quad \text{otherwise}$$

$$\text{mayevict}(m,x) = (|\{y \mid y \neq x \wedge \exists l_i : y \in m(l_i)\}| \geq A)$$

New Analysis Theory

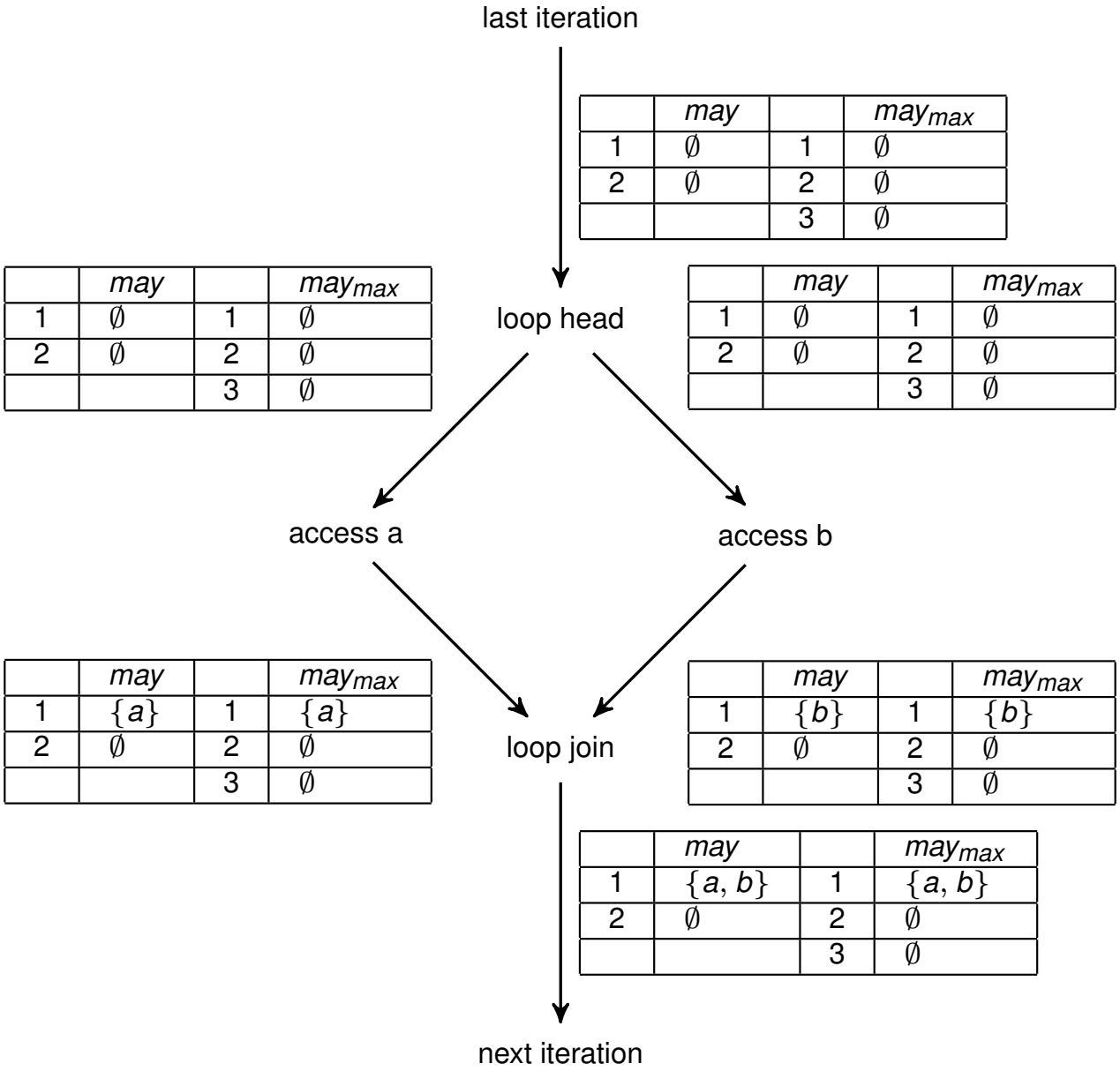
Join Function

$$J_{pers}((m,\hat{m}),(m',\hat{m}')) \quad = \quad (J_{may}(m,m'),\ J_{\widehat{may}}(\hat{m},\hat{m}'))$$

$$J_{\widehat{may}}(\hat{m},\hat{m}') \quad = \quad l_i \mapsto \begin{array}{ll} & \{x \mid \exists l_a, l_b : x \in \hat{m}(l_a) \wedge x \in \hat{m}'(l_b) \\ & \quad \wedge i = \max(a,b)\} \\ \cup & \{x \mid x \in \hat{m}(l_i) \wedge \nexists l_a : x \in \hat{m}'(l_a)\} \\ \cup & \{x \mid x \in \hat{m}'(l_i) \wedge \nexists l_a : x \in \hat{m}(l_a)\} \end{array}$$

$$\mid i = 1 \,..\, A+1$$

Example - New Analysis, 1st Iteration



	may		may_{max}
1	∅	1	∅
2	∅	2	∅
		3	∅

	may		may_{max}
1	∅	1	∅
2	∅	2	∅
		3	∅

	may		may_{max}
1	∅	1	∅
2	∅	2	∅
		3	∅

	may		may_{max}
1	$\{a\}$	1	$\{a\}$
2	∅	2	∅
		3	∅

	may		may_{max}
1	$\{b\}$	1	$\{b\}$
2	∅	2	∅
		3	∅

	may		may_{max}
1	$\{a, b\}$	1	$\{a, b\}$
2	∅	2	∅
		3	∅

Example - New Analysis, 2nd Iteration

SAARLAND UNIVERSITY
COMPUTER SCIENCE

last iteration

	may		may_{max}
1	$\{a, b\}$	1	$\{a, b\}$
2	$\emptyset$	2	$\emptyset$
		3	$\emptyset$

loop head

	may		may_{max}
1	$\{a, b\}$	1	$\{a, b\}$
2	$\emptyset$	2	$\emptyset$
		3	$\emptyset$

	may		may_{max}
1	$\{a, b\}$	1	$\{a, b\}$
2	$\emptyset$	2	$\emptyset$
		3	$\emptyset$

access a

access b

	may		may_{max}
1	$\{a\}$	1	$\{a\}$
2	$\{b\}$	2	$\{b\}$
		3	$\emptyset$

loop join

	may		may_{max}
1	$\{b\}$	1	$\{b\}$
2	$\{a\}$	2	$\{a\}$
		3	$\emptyset$

	may		may_{max}
1	$\{a, b\}$	1	$\emptyset$
2	$\emptyset$	2	$\{a, b\}$
		3	$\emptyset$

next iteration

Example - New Analysis, Fixpoint

last iteration

	may		may_{max}
1	$\{a, b\}$	1	$\emptyset$
2	$\emptyset$	2	$\{a, b\}$
		3	$\emptyset$

loop head

	may		may_{max}
1	$\{a, b\}$	1	$\emptyset$
2	$\emptyset$	2	$\{a, b\}$
		3	$\emptyset$

	may		may_{max}
1	$\{a, b\}$	1	$\emptyset$
2	$\emptyset$	2	$\{a, b\}$
		3	$\emptyset$

access a

access b

	may		may_{max}
1	$\{a\}$	1	$\{a\}$
2	$\{b\}$	2	$\{b\}$
		3	$\emptyset$

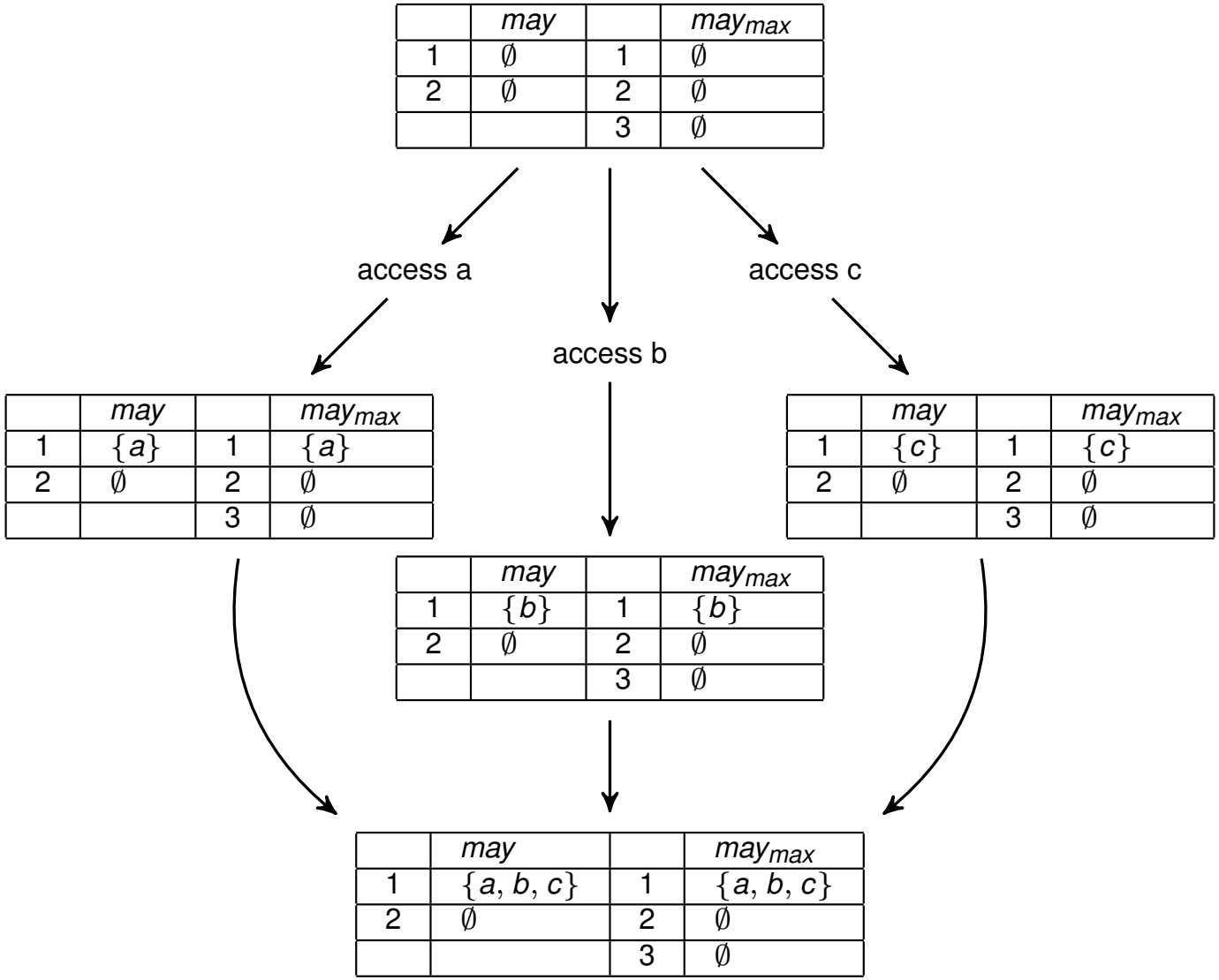
loop join

	may		may_{max}
1	$\{b\}$	1	$\{b\}$
2	$\{a\}$	2	$\{a\}$
		3	$\emptyset$

	may		may_{max}
1	$\{a, b\}$	1	$\emptyset$
2	$\emptyset$	2	$\{a, b\}$
		3	$\emptyset$

next iteration

Problem Hugues Cassé, 1st Iteration



	may		may_{max}
1	$\emptyset$	1	$\emptyset$
2	$\emptyset$	2	$\emptyset$
		3	$\emptyset$

	may		may_{max}
1	$\{a\}$	1	$\{a\}$
2	$\emptyset$	2	$\emptyset$
		3	$\emptyset$

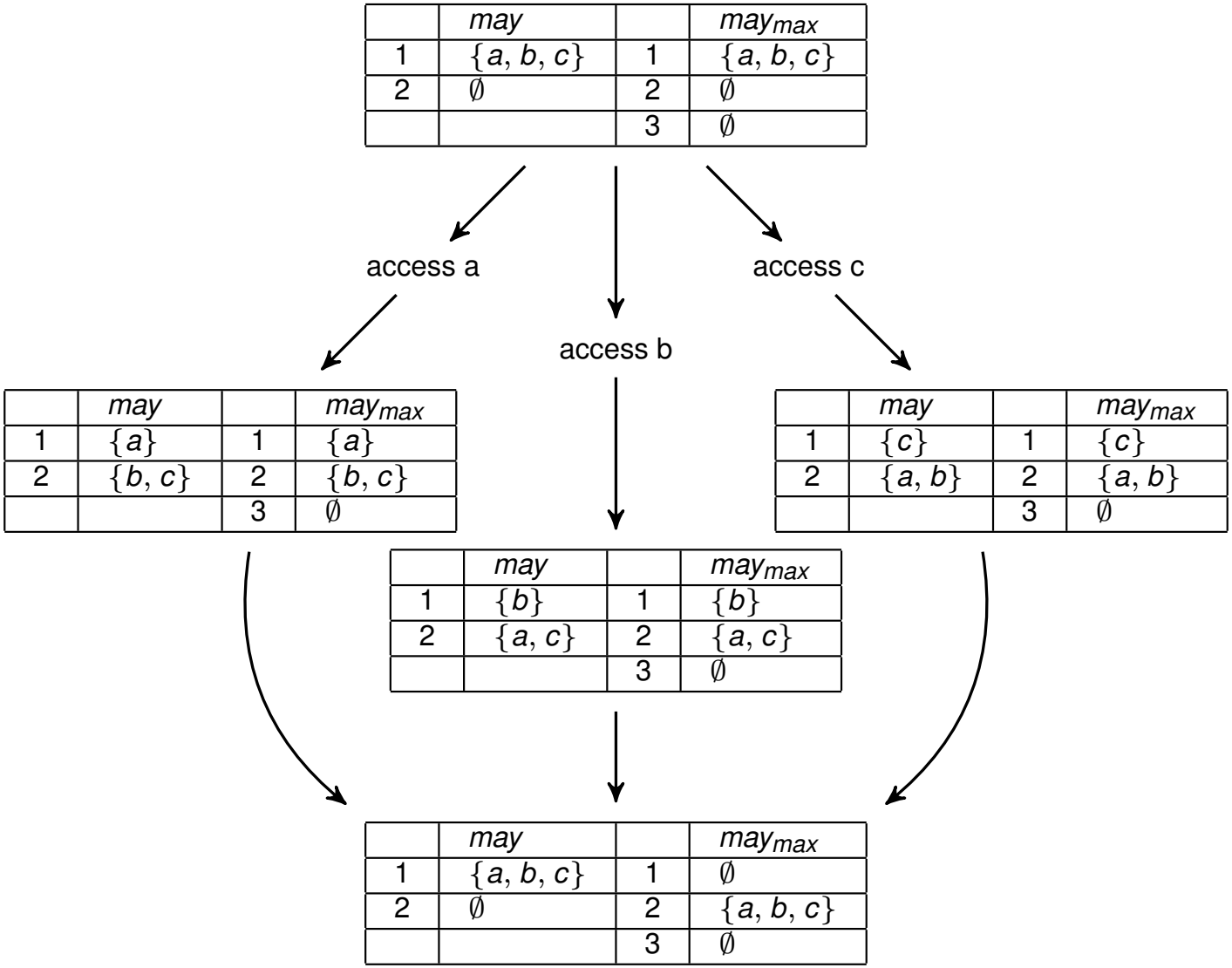
	may		may_{max}
1	$\{b\}$	1	$\{b\}$
2	$\emptyset$	2	$\emptyset$
		3	$\emptyset$

	may		may_{max}
1	$\{c\}$	1	$\{c\}$
2	$\emptyset$	2	$\emptyset$
		3	$\emptyset$

	may		may_{max}
1	$\{a, b, c\}$	1	$\{a, b, c\}$
2	$\emptyset$	2	$\emptyset$
		3	$\emptyset$

Problem Hugues Cassé, 2nd Iteration

SAARLAND UNIVERSITY
COMPUTER SCIENCE



	may		may_{max}
1	$\{a, b, c\}$	1	$\{a, b, c\}$
2	$\emptyset$	2	$\emptyset$
		3	$\emptyset$

	may		may_{max}
1	$\{a\}$	1	$\{a\}$
2	$\{b, c\}$	2	$\{b, c\}$
		3	$\emptyset$

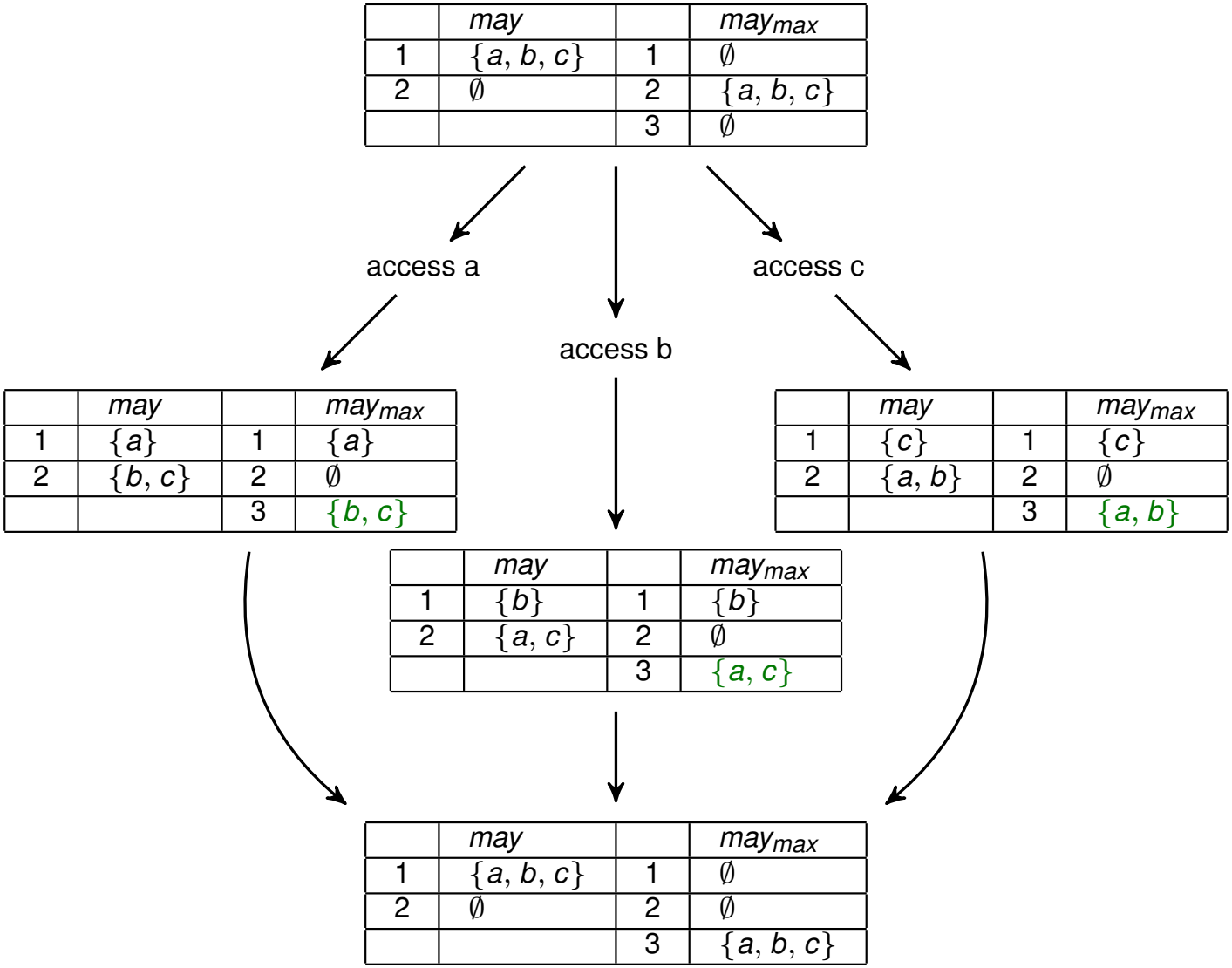
	may		may_{max}
1	$\{b\}$	1	$\{b\}$
2	$\{a, c\}$	2	$\{a, c\}$
		3	$\emptyset$

	may		may_{max}
1	$\{c\}$	1	$\{c\}$
2	$\{a, b\}$	2	$\{a, b\}$
		3	$\emptyset$

	may		may_{max}
1	$\{a, b, c\}$	1	$\emptyset$
2	$\emptyset$	2	$\{a, b, c\}$
		3	$\emptyset$

Problem Hugues Cassé, 3rd Iteration

SAARLAND UNIVERSITY
COMPUTER SCIENCE



	may		may_{max}
1	$\{a, b, c\}$	1	$\emptyset$
2	$\emptyset$	2	$\{a, b, c\}$
		3	$\emptyset$

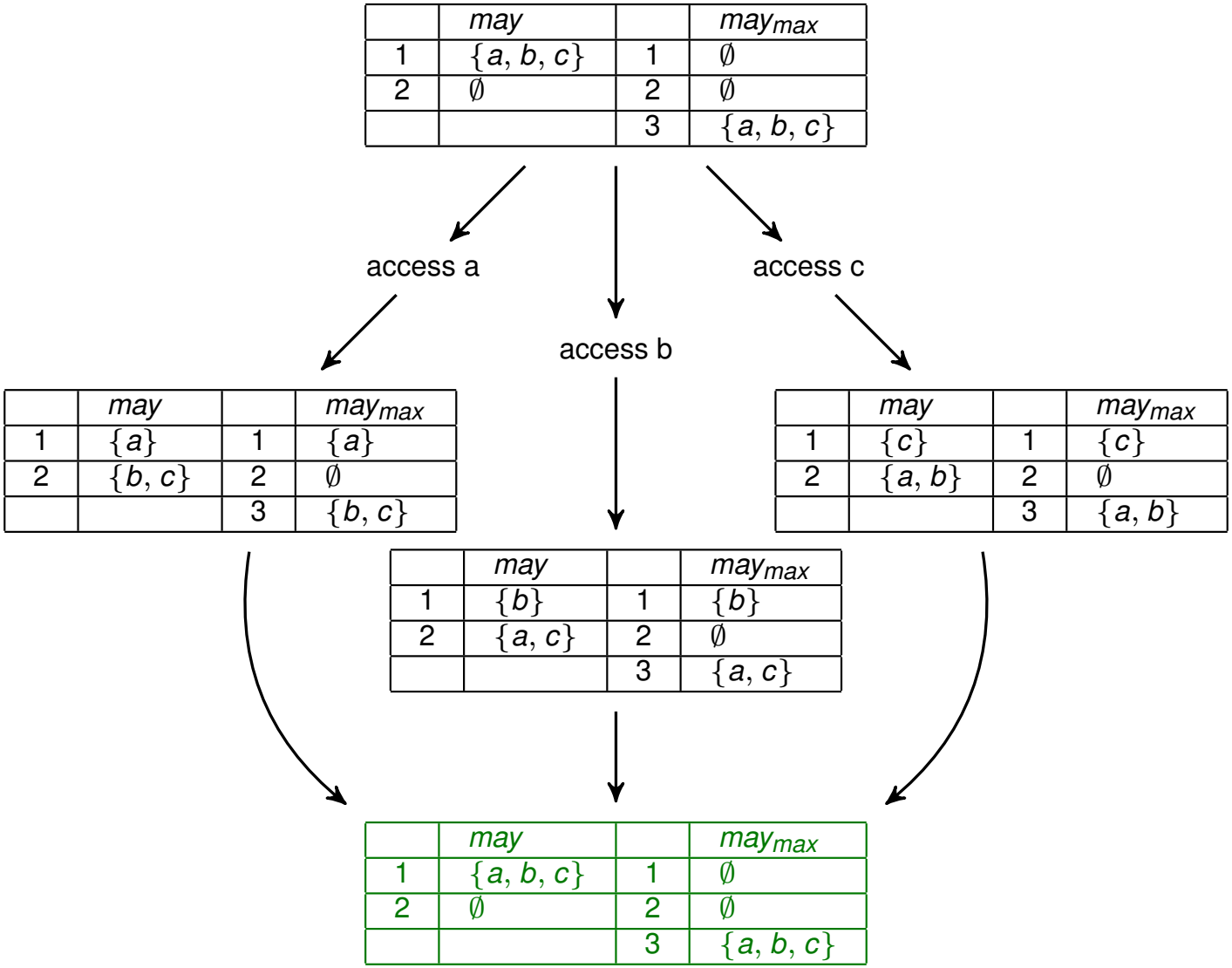
	may		may_{max}
1	$\{a\}$	1	$\{a\}$
2	$\{b, c\}$	2	$\emptyset$
		3	$\{b, c\}$

	may		may_{max}
1	$\{b\}$	1	$\{b\}$
2	$\{a, c\}$	2	$\emptyset$
		3	$\{a, c\}$

	may		may_{max}
1	$\{c\}$	1	$\{c\}$
2	$\{a, b\}$	2	$\emptyset$
		3	$\{a, b\}$

	may		may_{max}
1	$\{a, b, c\}$	1	$\emptyset$
2	$\emptyset$	2	$\emptyset$
		3	$\{a, b, c\}$

Problem Hugues Cassé, Fixpoint



Summary

New Analysis Works

- It allows successful persistence analysis for example.
- It solves the problem of Hugues Cassé.

Outlook

Future Work: Evaluation

- Analysis works for constructed examples.
- What is the gain in precision of the WCET estimate for real software?
- Research:
  - Evaluation on benchmark programs and real industry tasks.
  - Evaluation on current processors:
    MPC755 or MPC7448 (partial locked cache), MPC603e

Outlook

Future Work: Unsharp Accesses

- Extension to allow the handling of unsharp accesses.
- Draft implementation using one place holder element already works.
- Research:
  - How does this extension work out on real software (data caches)?
  - Would it make sense, to introduce different place holders for different accesses?

Outlook

Future Work: Persistence Scopes

- Persistence uses scopes, e.g., the loop in the example.
- Improve: Allow nested persistence scopes.
  (As shown by Clément Ballabriga and Hugues Cassé)
- Research:
  - How much more precision do nested scopes allow?
  - How to chose good persistence scopes automatically?
  - Are there optimal scopes?

Questions?