Instruction Caches in Static WCET Analysis of Artificially Diversified Software

Joachim Fellmuth, Thomas Göthel, Sabine Glesner

Technische Universität Berlin
Software and Embedded Systems Engineering

ECRTS 2018, Barcelona

SESE

berlin

Motivation

Cyber-physical systems (CPS) omnipresent

- **safety-critical**, **hard real-time** requirements
- highly interconnected $\rightarrow$ **large attack surface**

Motivation

Cyber-physical systems (CPS) omnipresent

- **safety-critical**, **hard real-time** requirements
- highly interconnected $\rightarrow$ **large attack surface**

Security important development aspect

- unsafe languages enable **code-reuse attacks**

$\rightarrow$ use knowledge of system's memory

Motivation

Cyber-physical systems (CPS) omnipresent

- **safety-critical**, **hard real-time** requirements
- highly interconnected $\rightarrow$ **large attack surface**

Security important development aspect

- unsafe languages enable **code-reuse attacks**

$\rightarrow$ use knowledge of system's memory

Artificial software diversity

- hides memory layout (e.g. randomize instruction addresses)
- copes with unknown attack types
- WCET-aware security increase possible [FHPG17]

Problem

Problem: **Uncertainty** induces **pessimism** in WCET analysis

- timing impact of some randomized diversification techniques unpredictable
  [DDNS12, LHBF14, Coh93, FSA97, WMHL12]
- **WCET hardware analyses** rely on full knowledge of the program addresses
- state-of-the-art **cache analyses** not able to produce upper bound estimate for **all program variants**
  [HJR11, ZK15, BC08, Cul13, LGR+16]
- *All miss* has to be assumed as **worst-case** cache behavior

Proposed Solution

Our goal: Efficient **WCET cache analysis** for diverse programs

- **powerful diversity approach**
- reuse **established analyses**, compatible with **IPET**
- **tight** worst-case estimate **over all variants**

Approach:

- **relocation and reordering** of code fragments
- **introduce uncertainty** into WCET cache analysis
- aggregate results for all variants **per basic block**

Outline

Outline

Artificial software diversity

Semantically equivalent program **variants**

$\rightarrow$ different program layout in memory

$\rightarrow$ exploit compiler decisions to obtain variants

Artificial software diversity

Semantically equivalent program **variants**

→ different program layout in memory

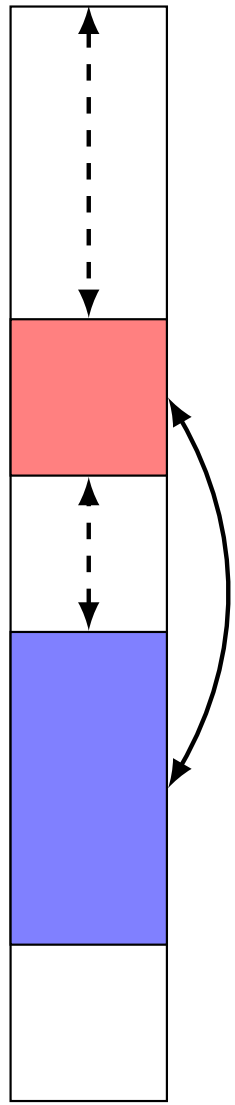
→ exploit compiler decisions to obtain variants

We use **relocation** and **reordering** of rearrangeable code parts (fragments)

→ no changes to instructions (code size) and CFG

→ predictable behavior over all variants

→ covers the entire instruction memory



Artificial software diversity

Semantically equivalent program **variants**

→ different program layout in memory

→ exploit compiler decisions to obtain variants

We use **relocation** and **reordering** of rearrangeable code parts (fragments)

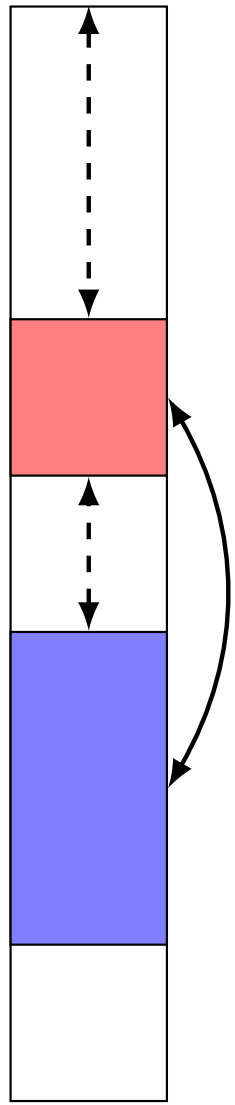
→ no changes to instructions (code size) and CFG

→ predictable behavior over all variants

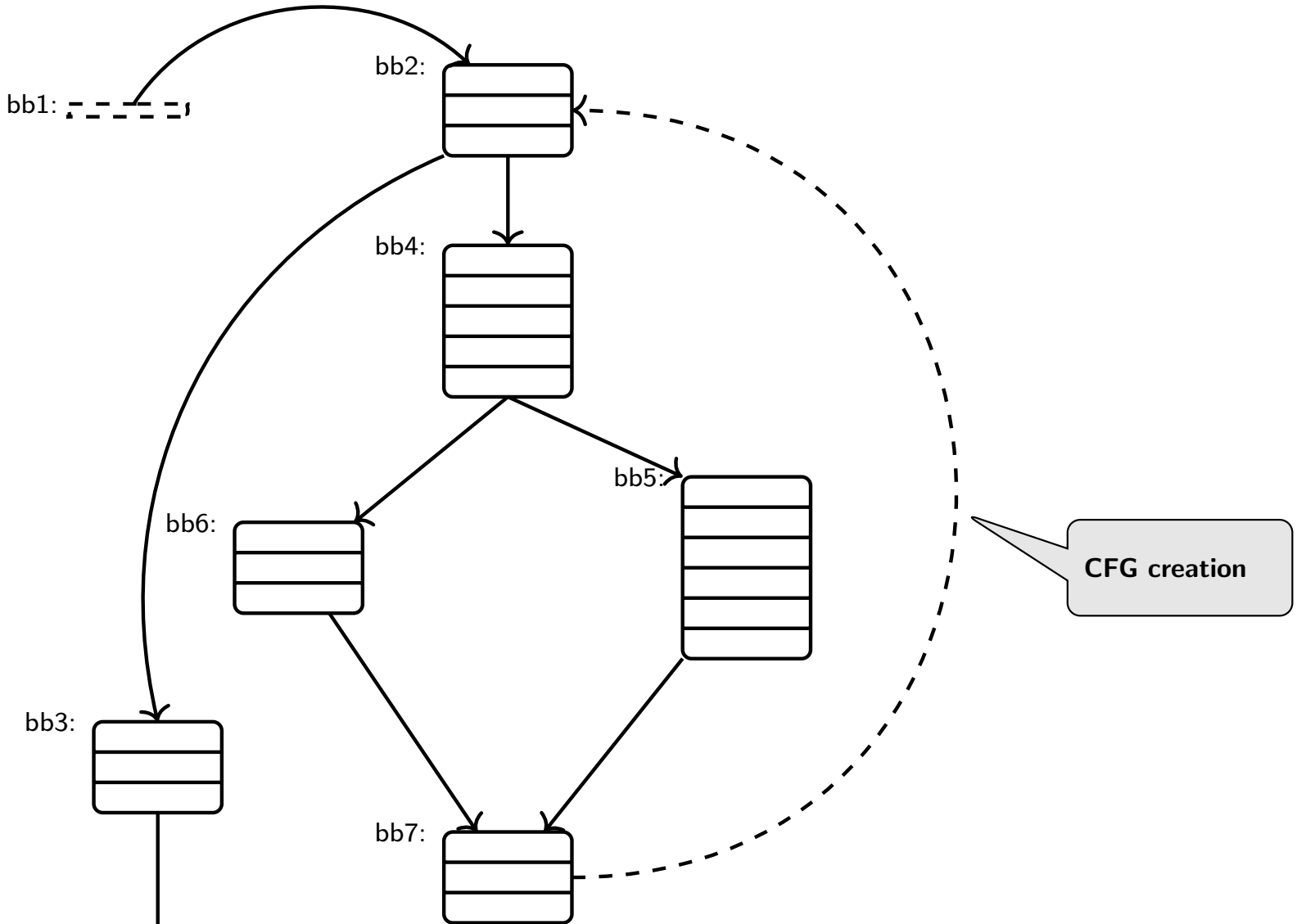
→ covers the entire instruction memory

Different **fragment granularities** possible

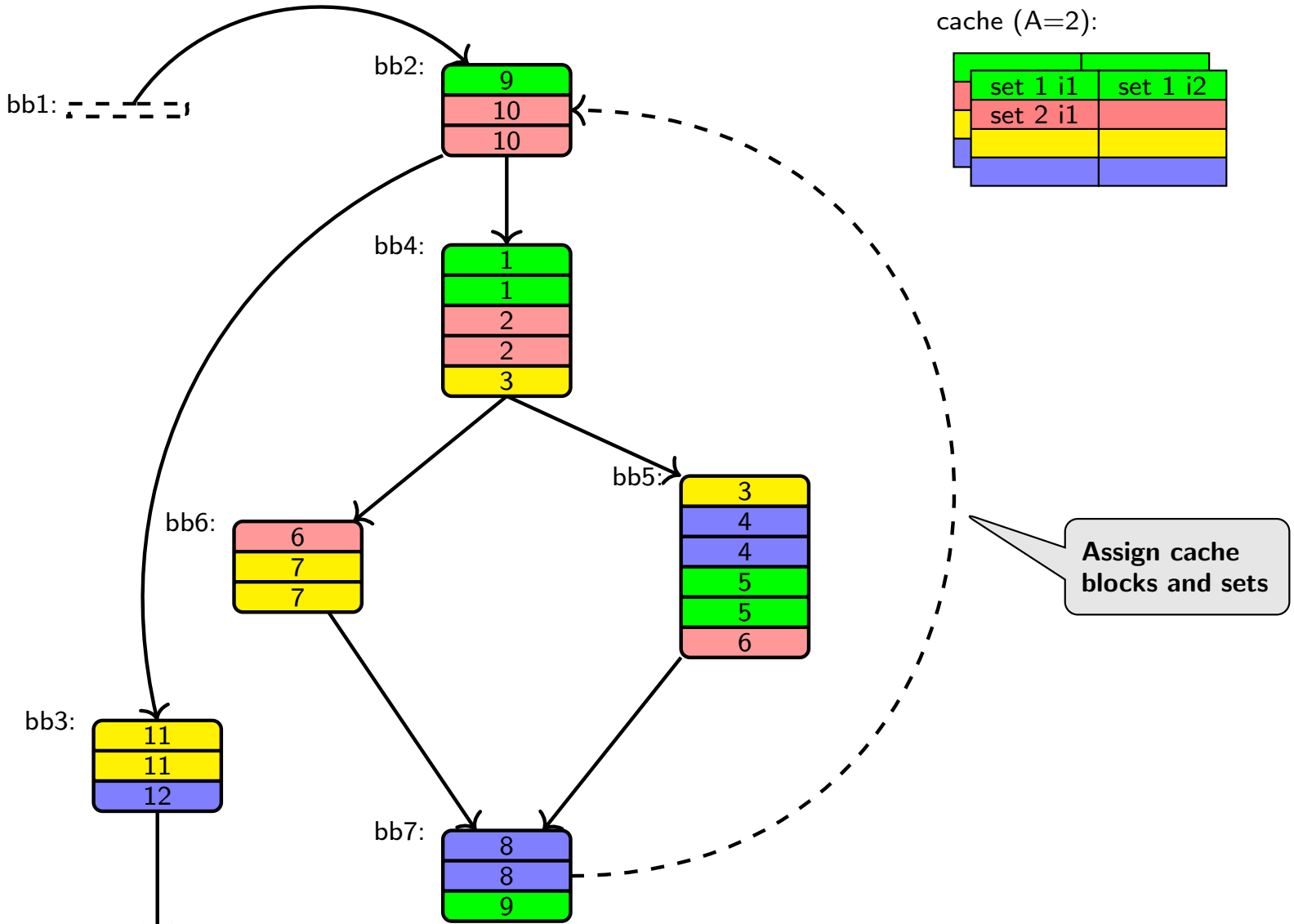
→ **segment-, function, block level**



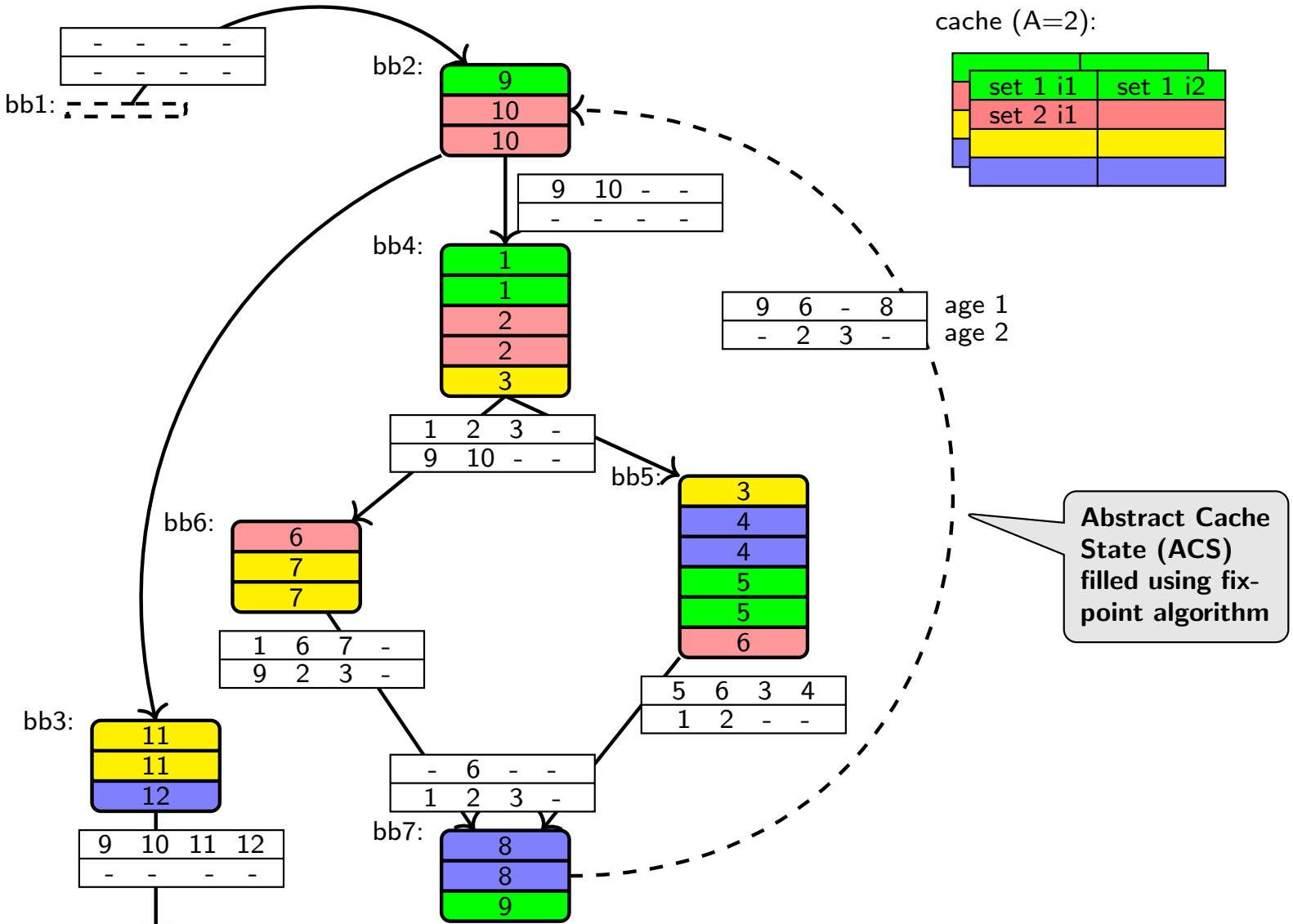
Cache analysis: Must (LRU)



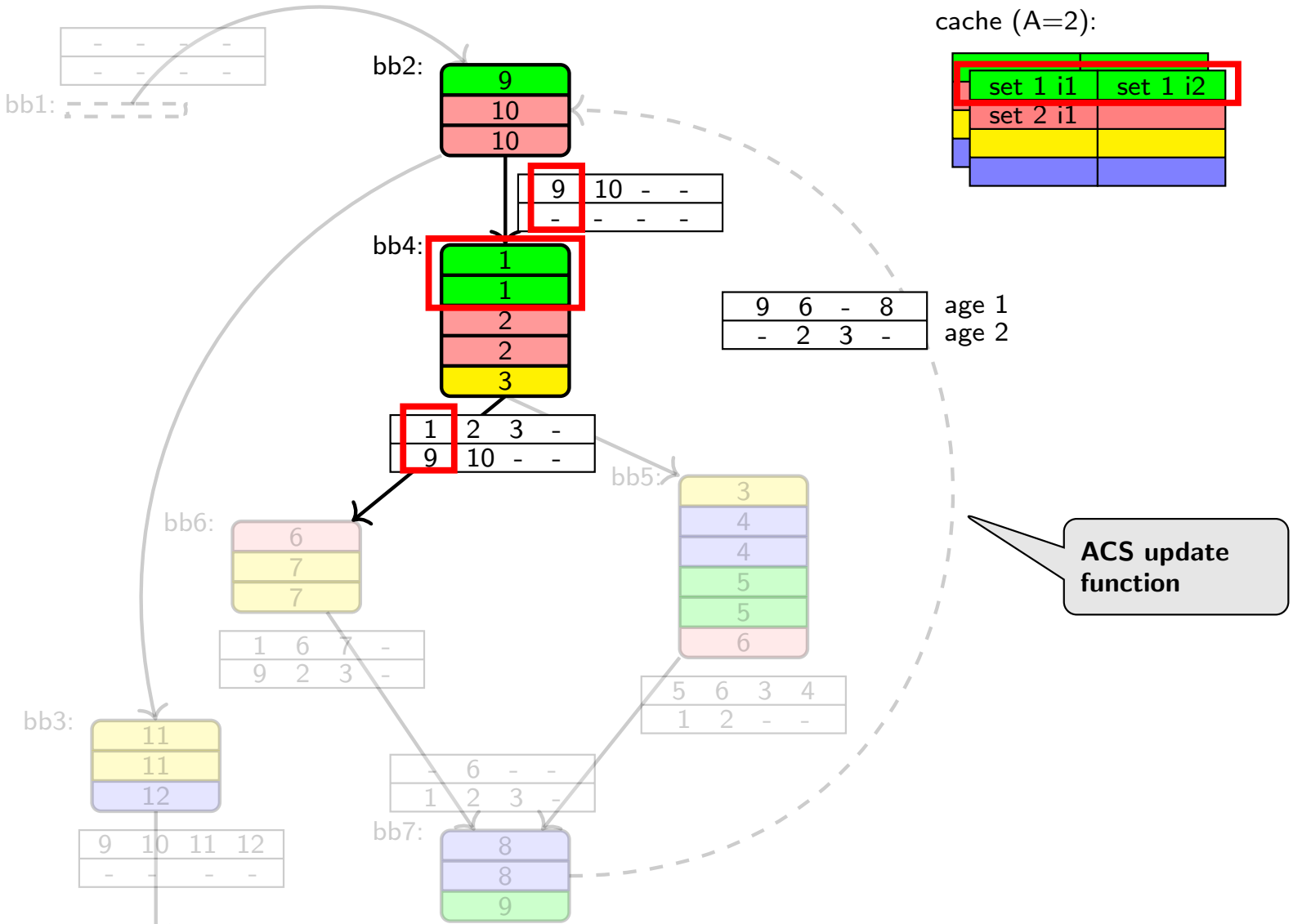
Cache analysis: Must (LRU)



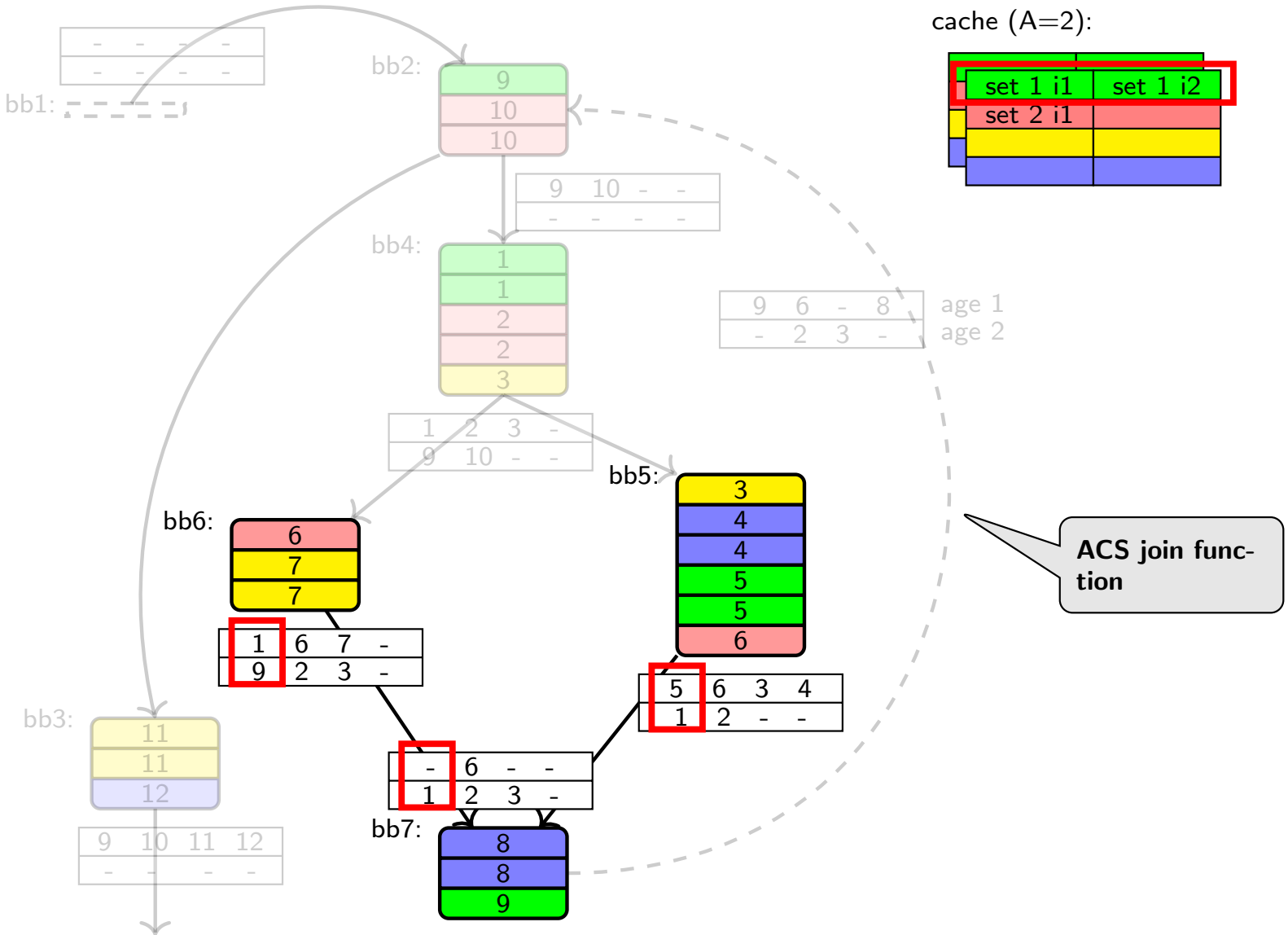
Cache analysis: Must (LRU)



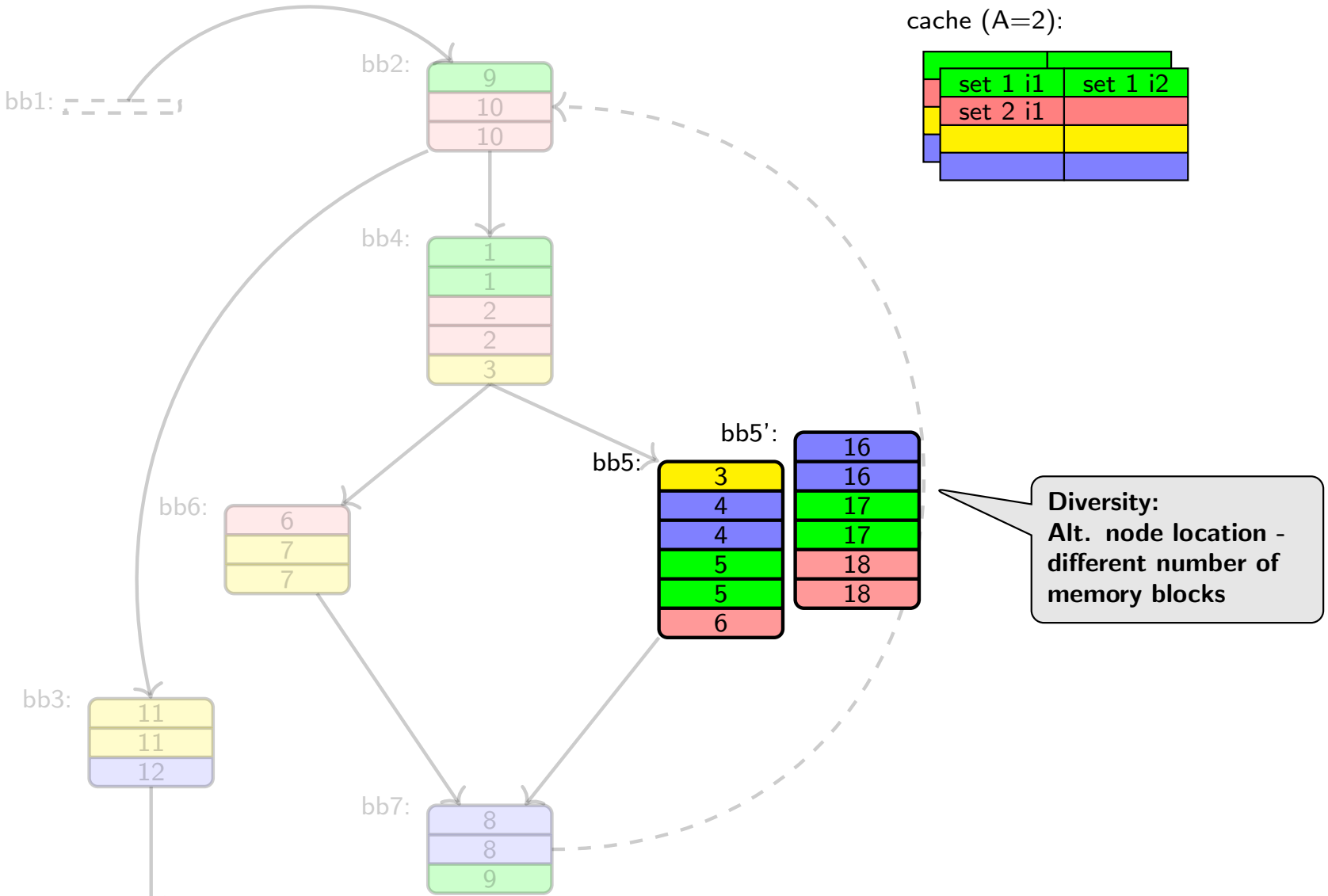
Cache analysis: Must (LRU)



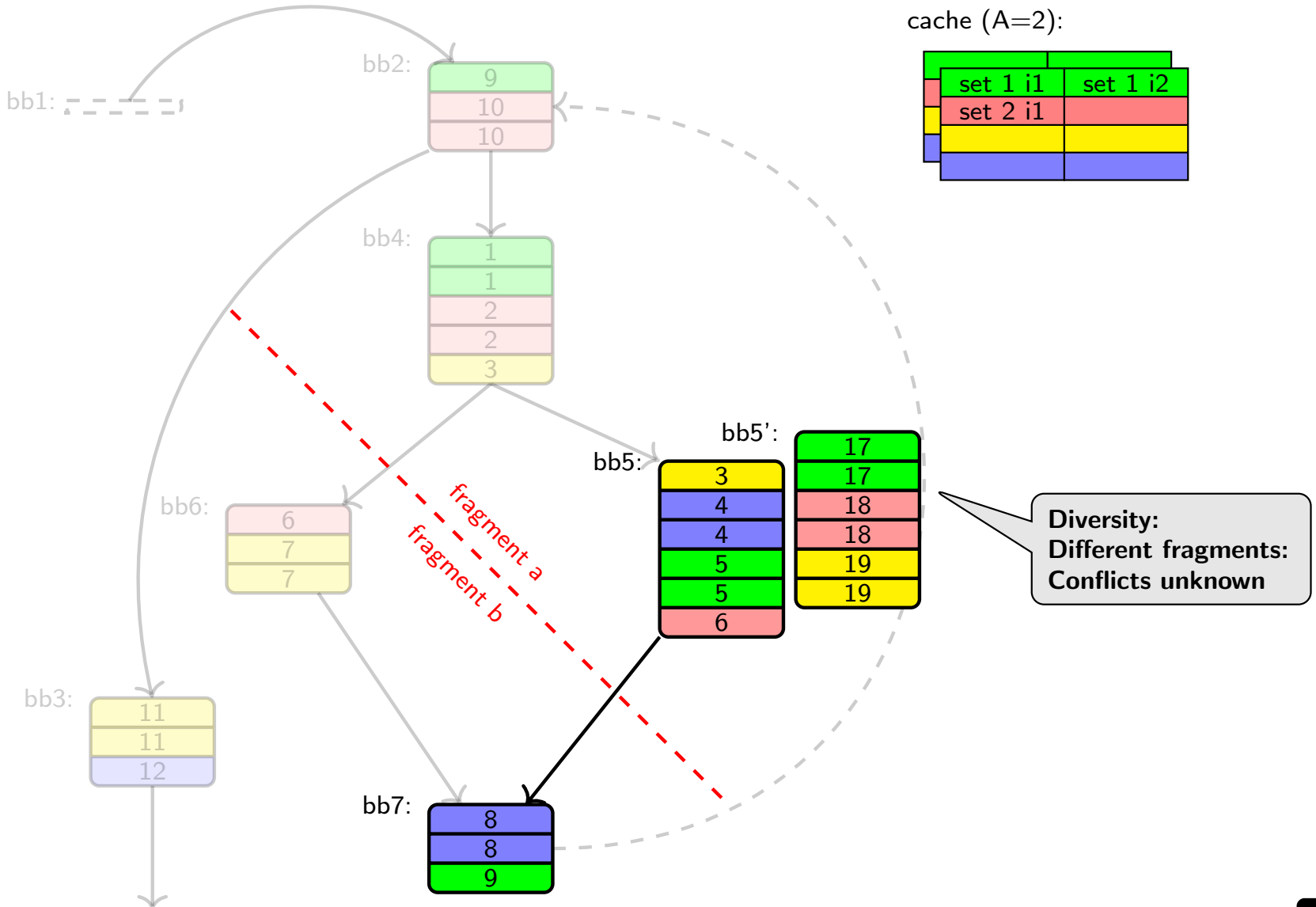
Cache analysis: Must (LRU)



Cache analysis: Must (LRU)



Cache analysis: Must (LRU)



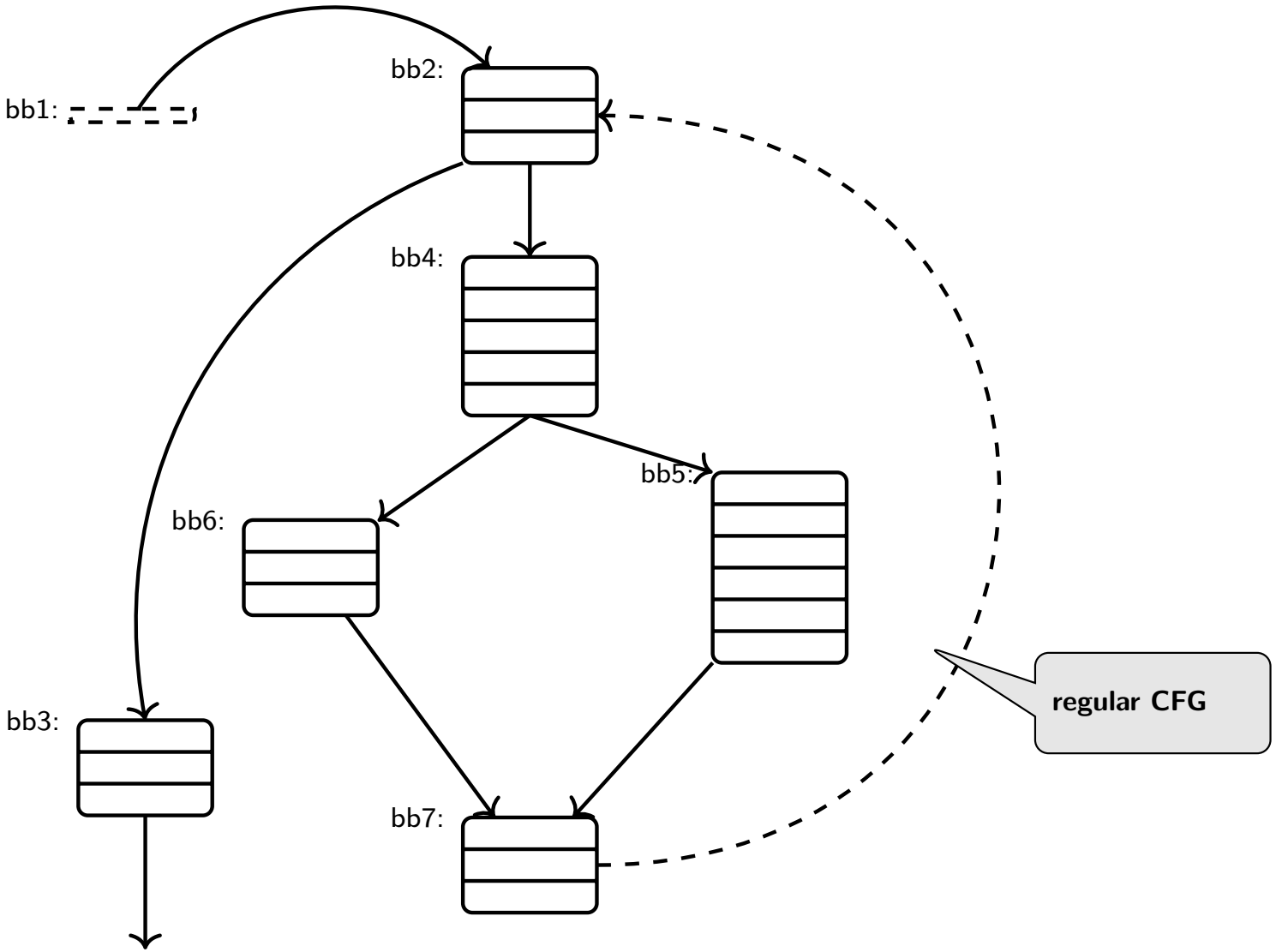
Outline

Cache Analysis for diverse Programs

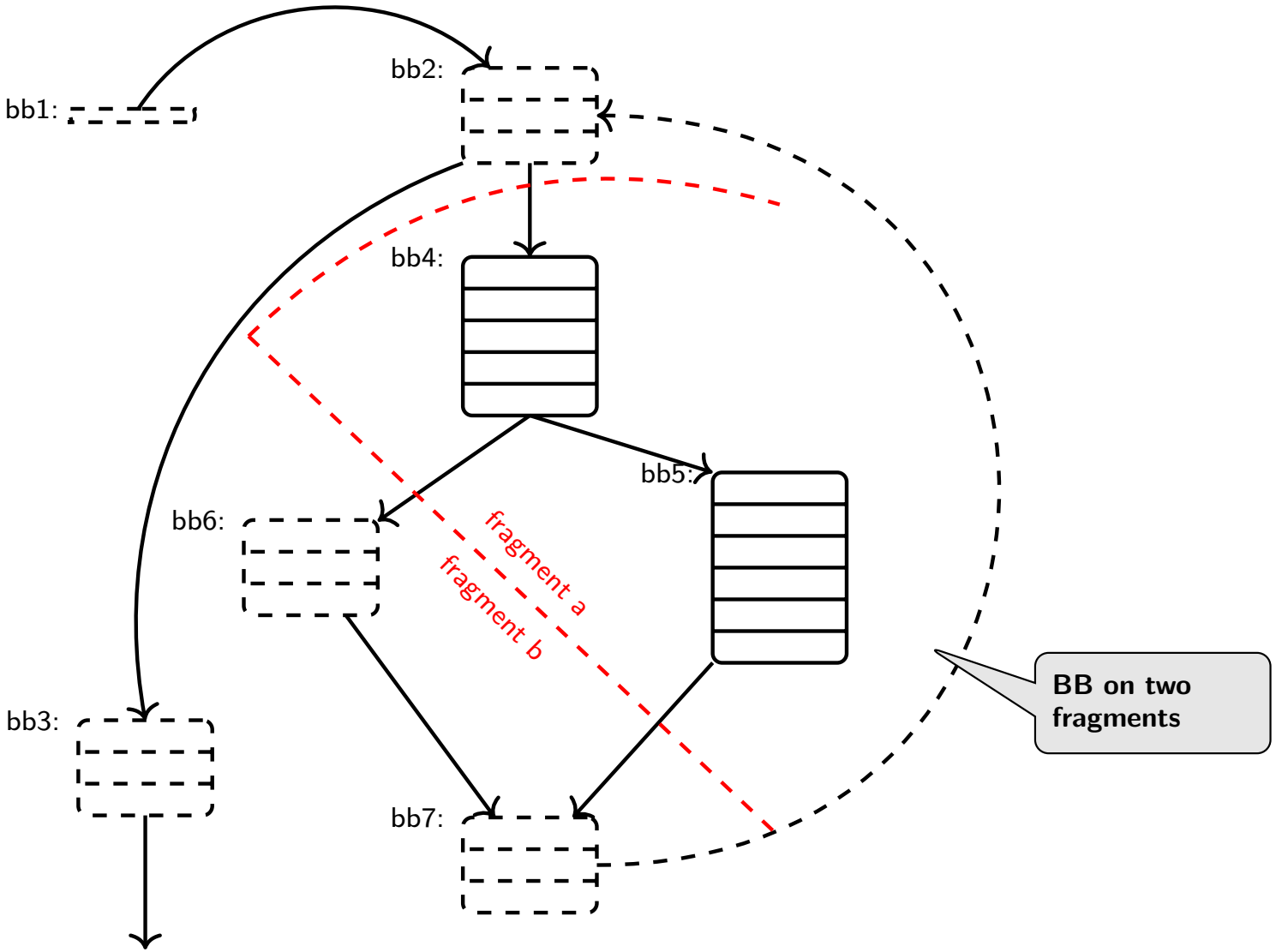
Uncertainty of relocation and reordering in Abstract Cache State

- every basic block (BB) belongs to a fragment
- distances to BB are only known within this fragment
  → one "virtual" cache per fragment
  → regular cache replacement within fragment
  → cache contents of other fragments subject to worst-case
- behavior depends on offset within set
  → cache behavior is equal over all sets
  → fragment starts at 0 plus offset
  → one cache representation for each possible offset

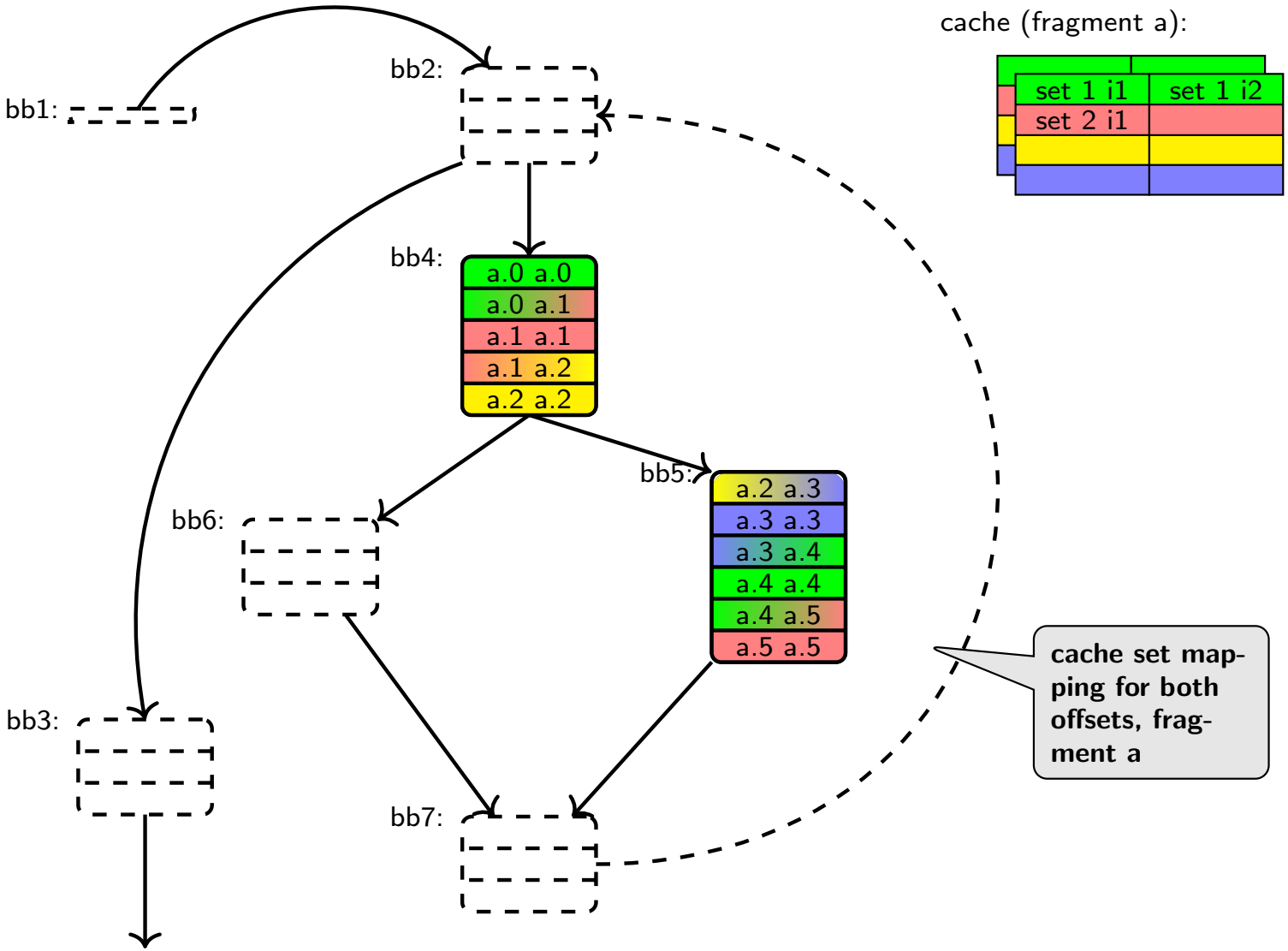
Location-tolerant Must analysis



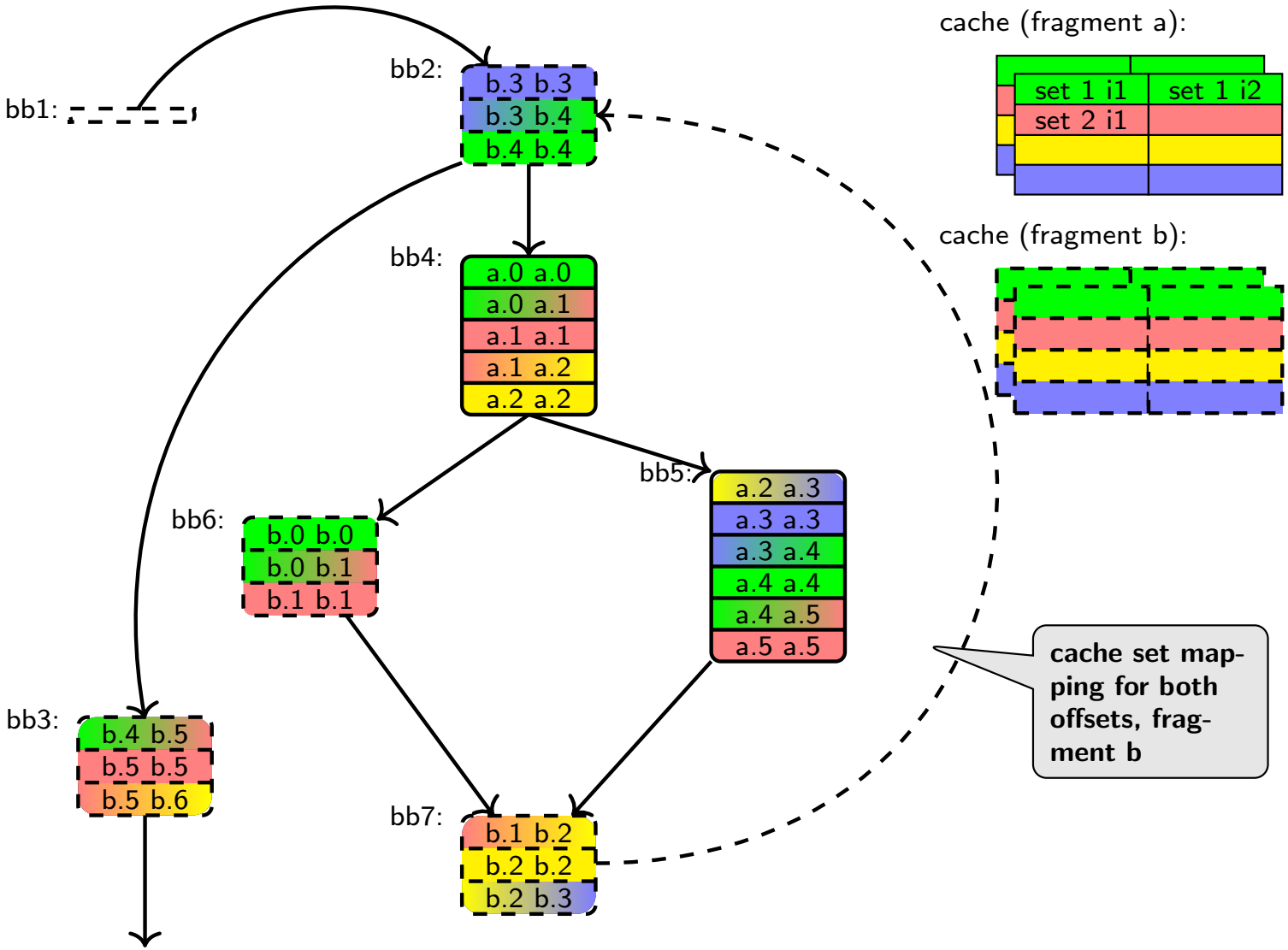
Location-tolerant Must analysis



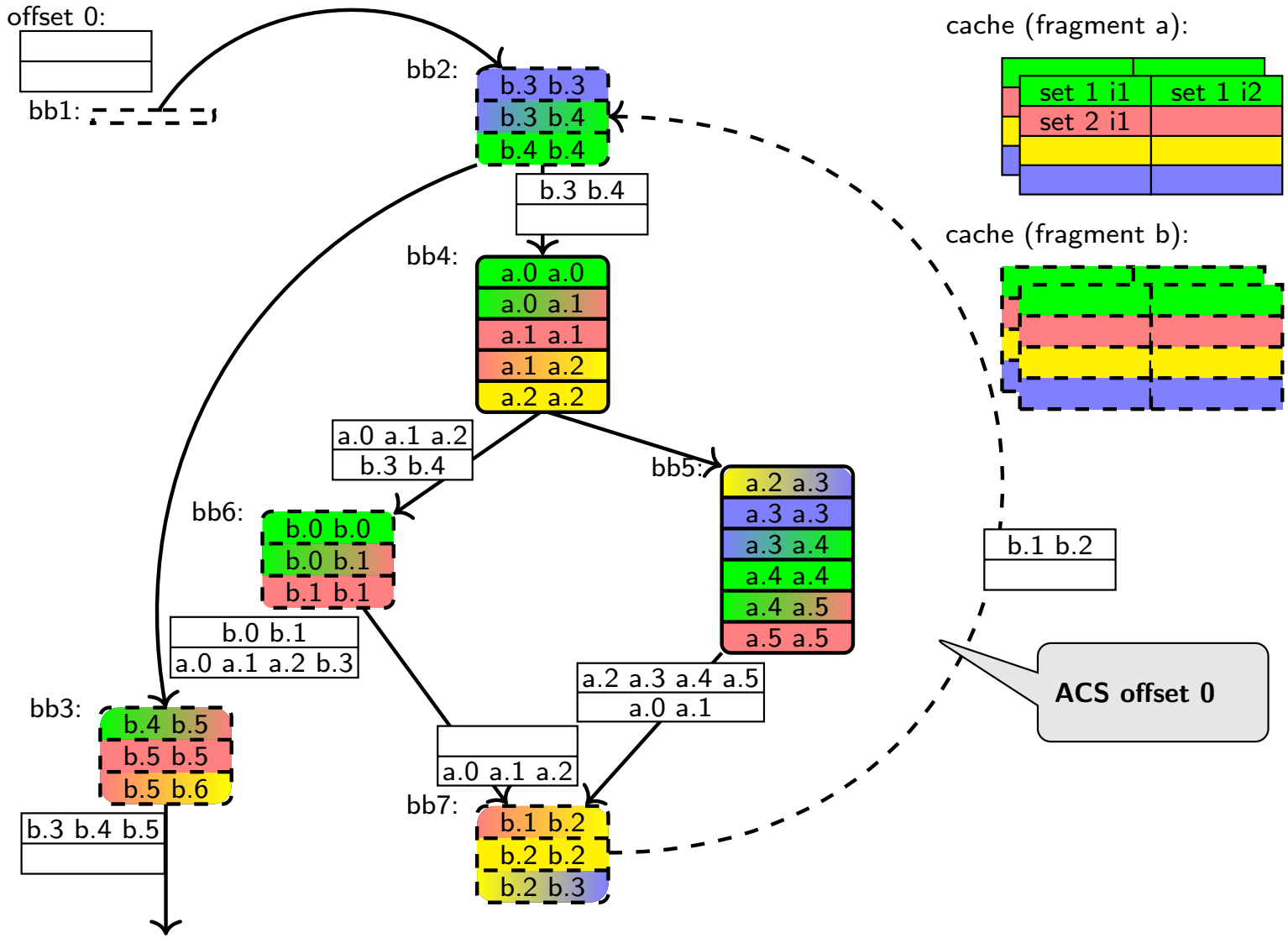
Location-tolerant Must analysis



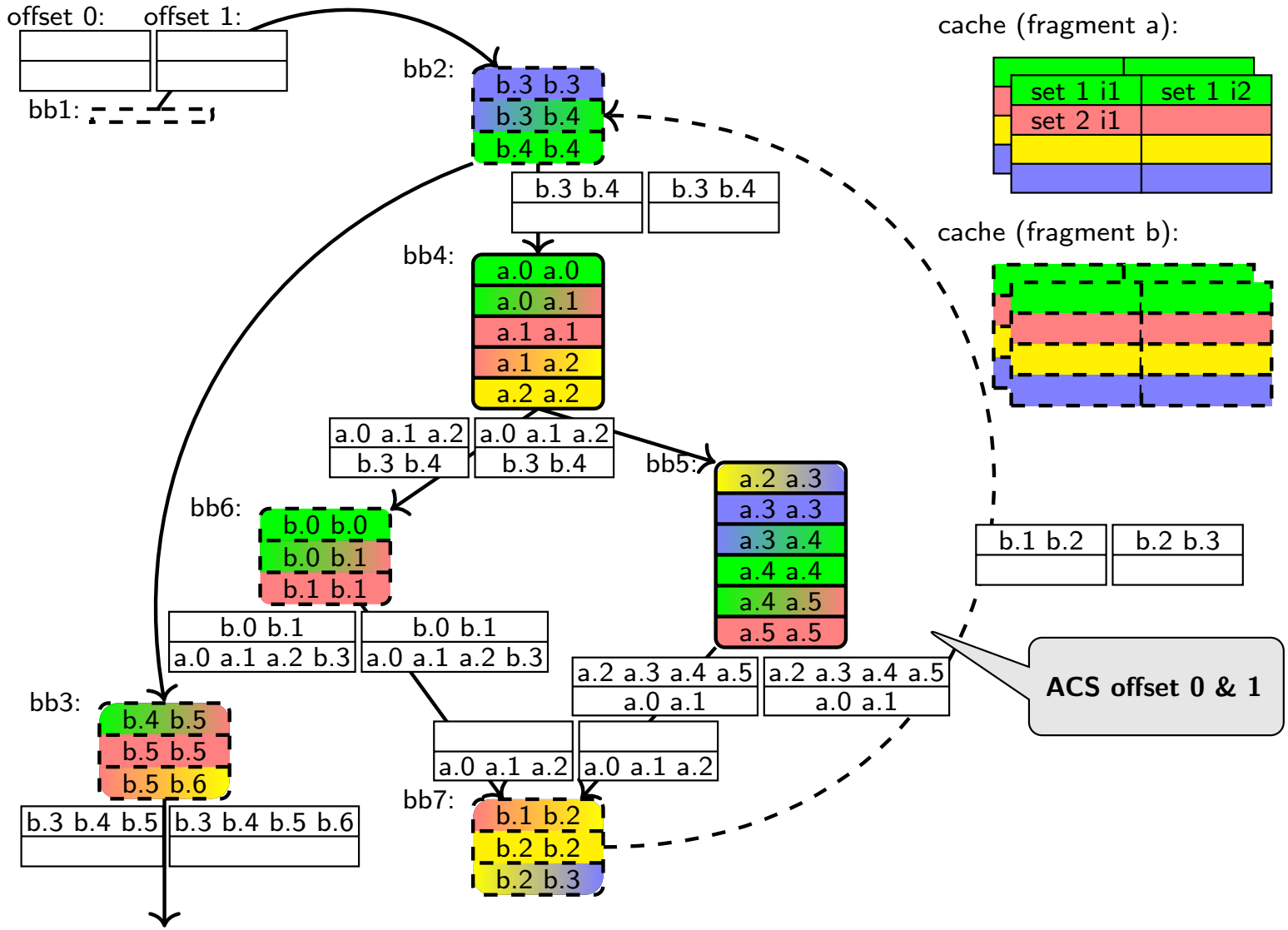
Location-tolerant Must analysis



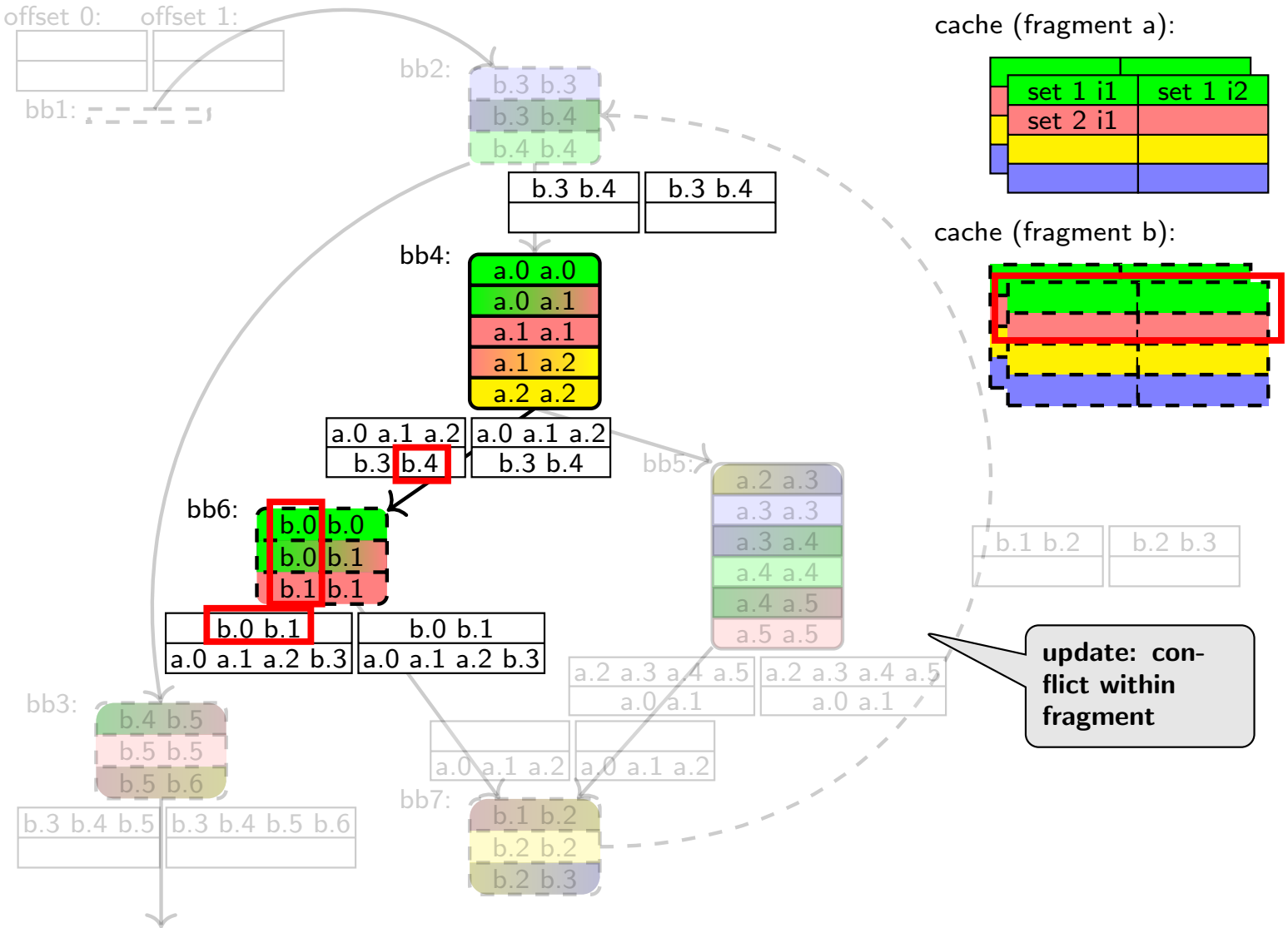
Location-tolerant Must analysis



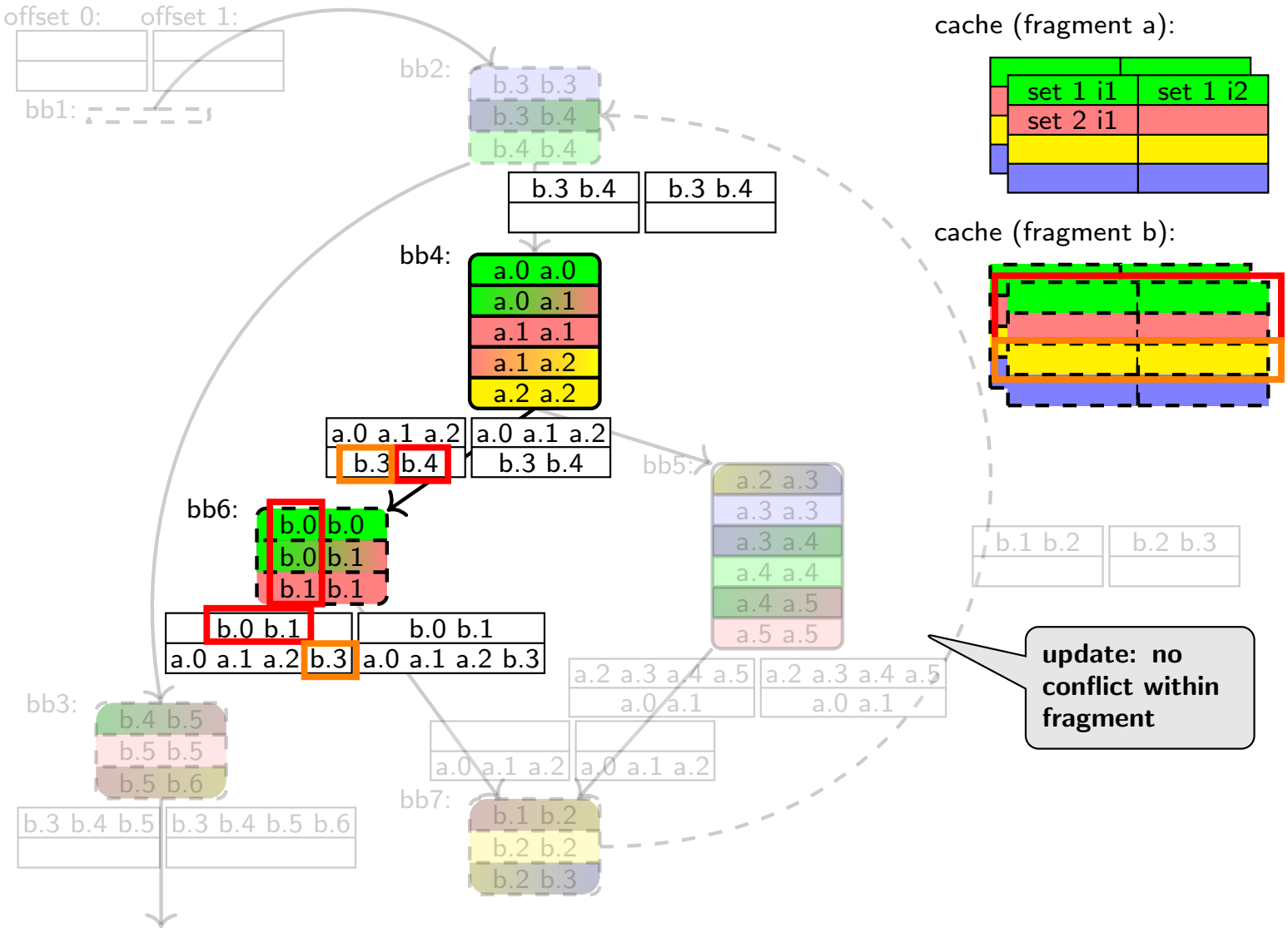
Location-tolerant Must analysis



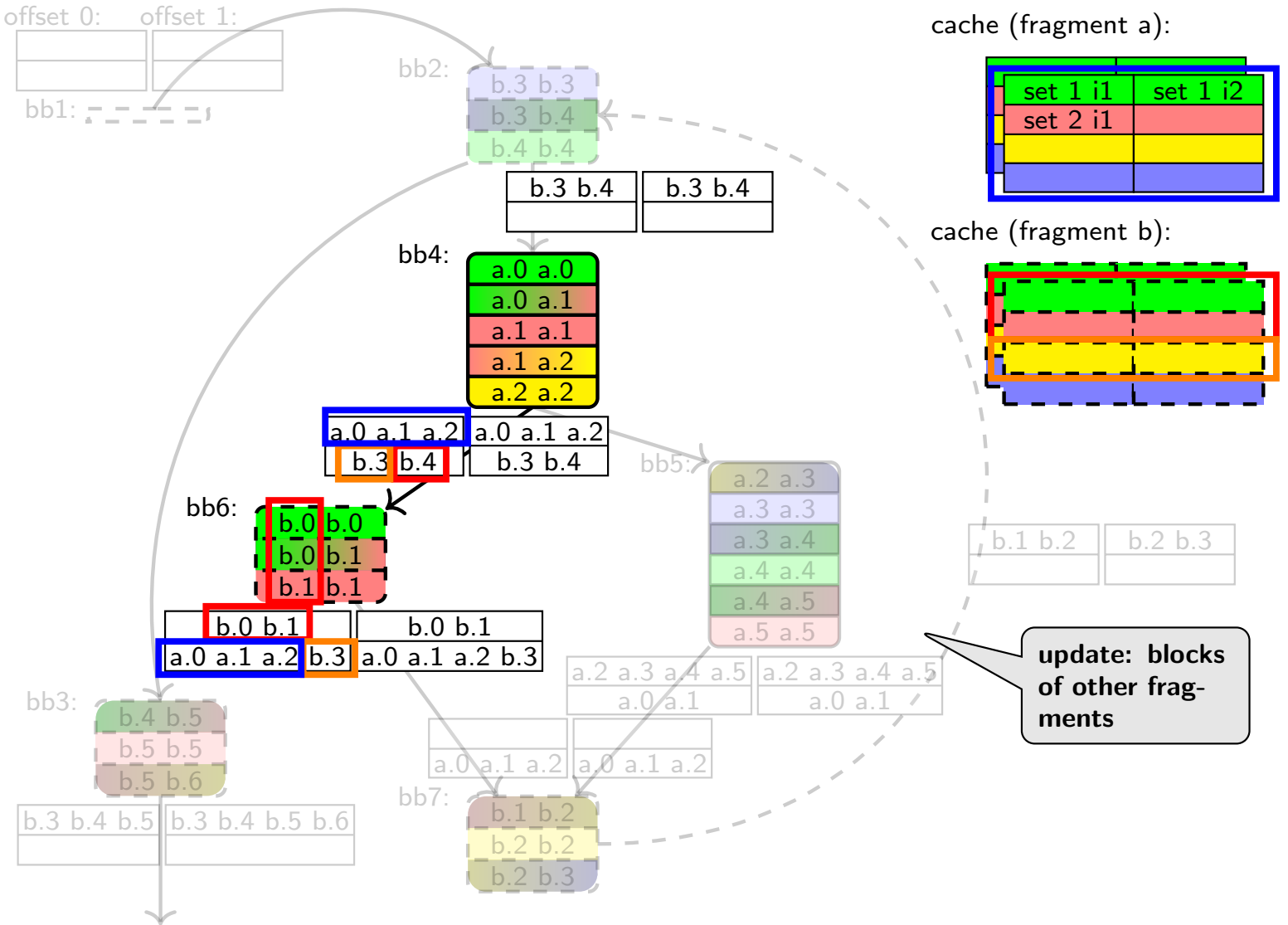
Location-tolerant Must analysis



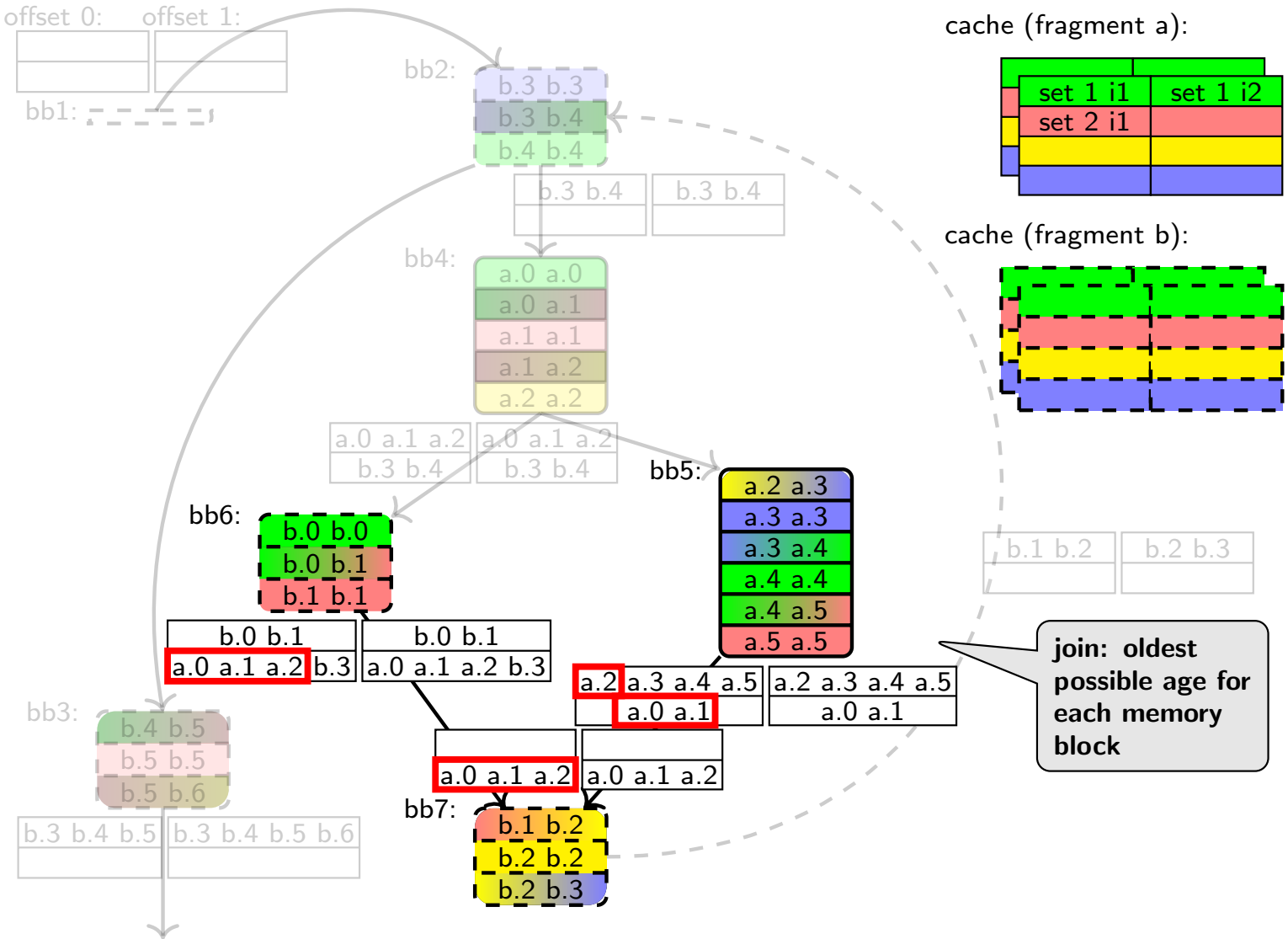
Location-tolerant Must analysis



Location-tolerant Must analysis



Location-tolerant Must analysis



Further analyses

Must analysis update uses most of addressing information

- within fragment: regular update
- outside fragment: worst-case aging may not cause eviction

Approach can be applied similarly to *may* and *persistence* analyses

→ Ballabriga's multi-level persistence analysis [BC08]

→ Allows for *first miss* classification with respect to enclosing loops

ACS results aggregated to find worst case timing **per basic block**

Outline

Experimental Setup

Implemented in **OTAWA** (I.R.I.T. Labs, Toulouse)

- analysis entirely **static**
- **framework** intended for extension in research

Experimental Setup

Implemented in **OTAWA** (I.R.I.T. Labs, Toulouse)

- analysis entirely **static**
- **framework** intended for extension in research

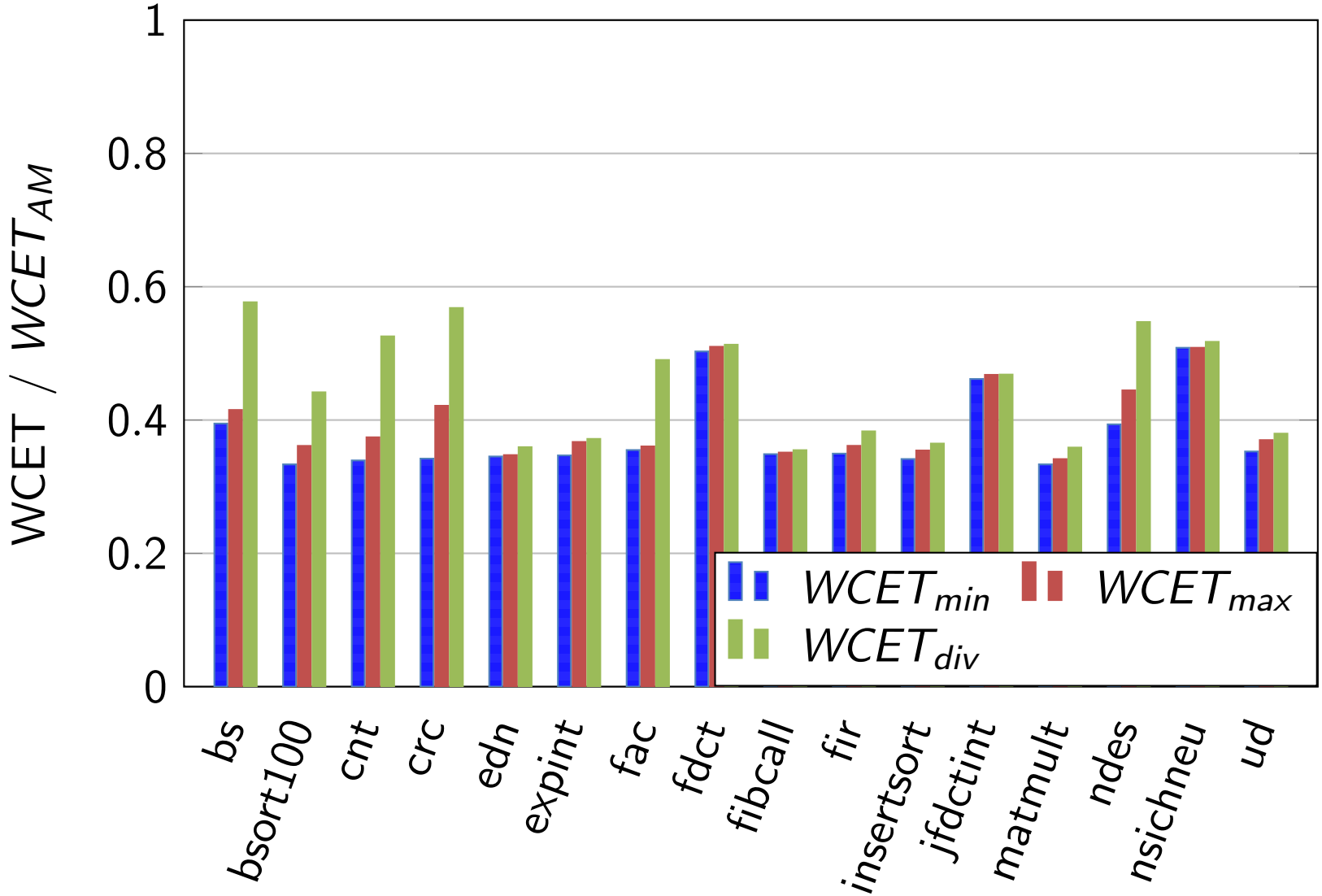
Tested on **Mälardalen benchmarks**

- widely used WCET benchmark suite
- **program collection**, cover all aspects of WCET analysis

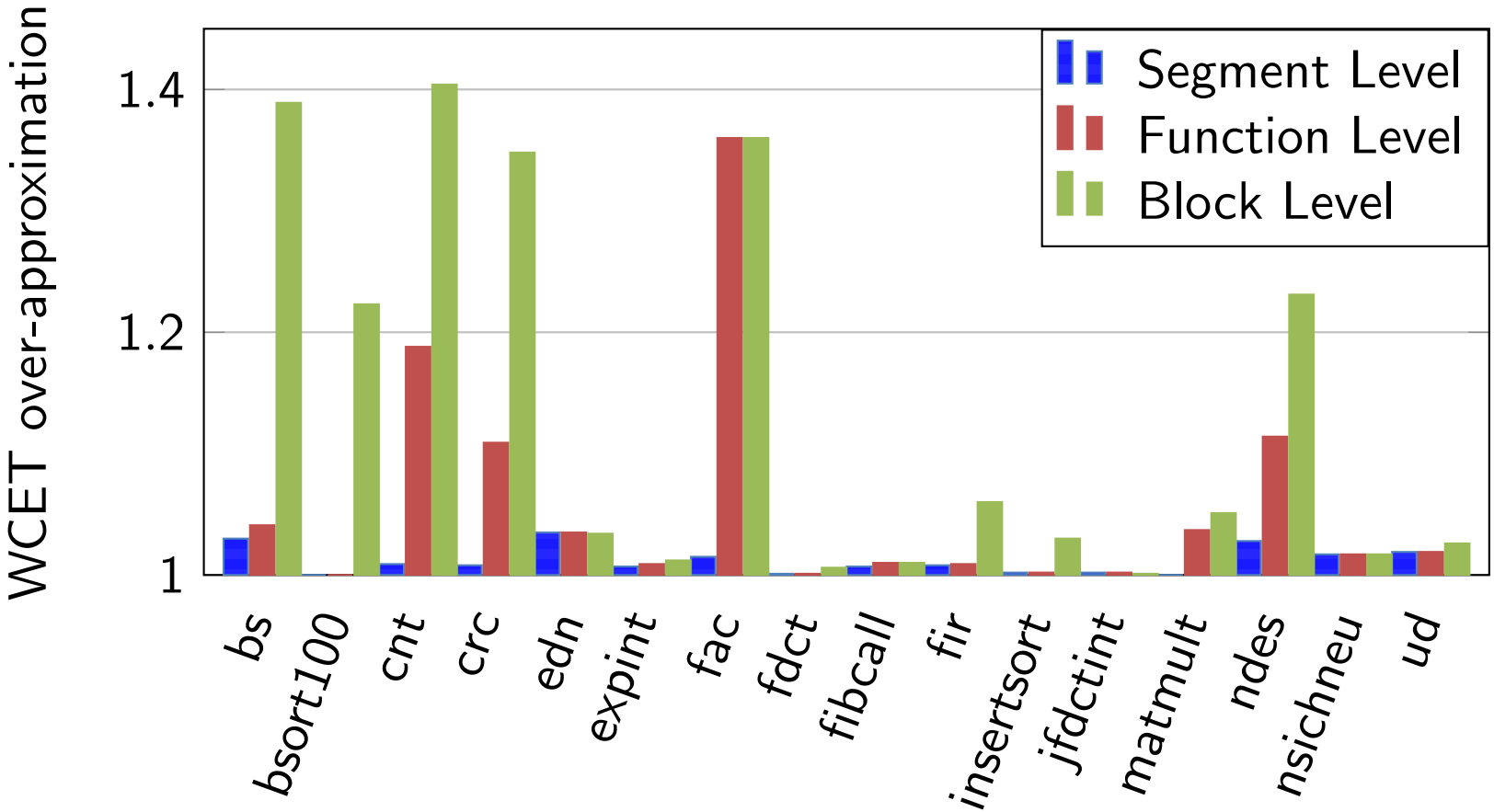
Experiments conducted with random benchmark variants

- cache sizes, associativity and granularity varied
- diverse WCET estimate compared to non-diverse analysis
- $WCET_{AM}$ base line of all estimations

Results: Precision



Results: Impact of granularity



Outline

Conclusion

Contribution: static cache analysis **for diversified programs**

- **tight estimates** with respect to uncertainties of diversity
- **major improvement** to assuming *all miss*
- acceptable analysis time
- supports powerful **relocation and reordering** diversification
- applicable to other aspects of **varying memory layout**: e.g. dynamic libraries, redundancy concepts

Conclusion

Contribution: static cache analysis **for diversified programs**

- **tight estimates** with respect to uncertainties of diversity
- **major improvement** to assuming *all miss*
- acceptable analysis time
- supports powerful **relocation and reordering** diversification
- applicable to other aspects of **varying memory layout**: e.g. dynamic libraries, redundancy concepts

Outlook:

- more WCET aspects: multi-level caches, branch prediction
- investigate architectures with timing anomalies
- optimize fragmentation using cache analysis results

Conclusion

Contribution: static cache analysis **for diversified programs**

- **tight estimates** with respect to uncertainties of diversity
- **major improvement** to assuming *all miss*
- acceptable analysis time
- supports powerful **relocation and reordering** diversification
- applicable to other aspects of **varying memory layout**: e.g. dynamic libraries, redundancy concepts

Q & A

Outlook:

- more WCET aspects: multi-level caches, branch prediction
- investigate architectures with timing anomalies
- optimize fragmentation using cache analysis results