PUB: Path Upper-Bounding for Measurement-Based Probabilistic Timing Analysis

Leonidas Kosmidis, Jaume Abella, Franck Wartel,
Eduardo Quiñones, Antoine Colin, Francisco J. Cazorla

Madrid, July 11th, ECRTS 2014
www.proxima-project.eu

This project and the research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7 / 2007-2013] under grant agreement 611085

Outline

- Motivation and problem description
- Introduction to MBPTA
- Time Randomised Caches and associated properties
- PUB
- Results
- Conclusion

Outline

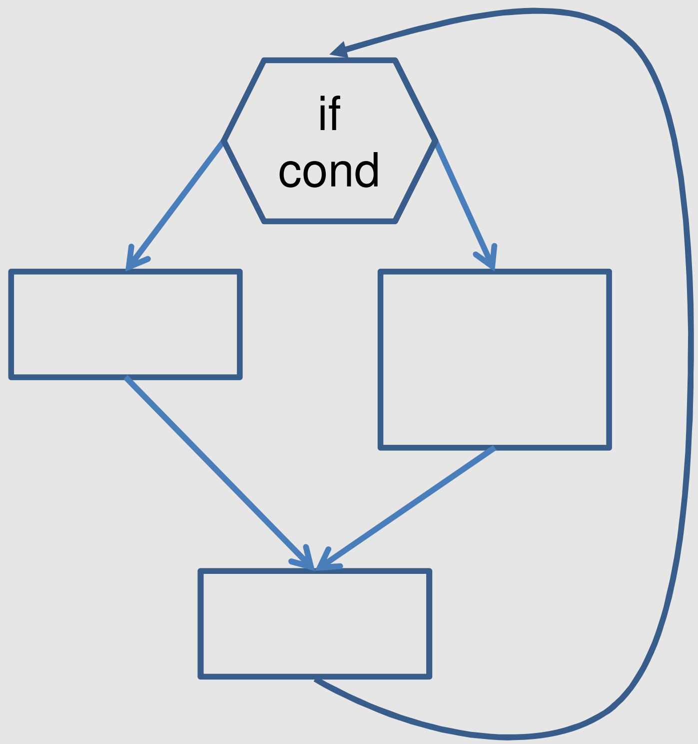
- Motivation and problem description
- Introduction to MBPTA
- Time Randomised Caches and associated properties
- PUB
- Results
- Conclusion

Motivation

- Modern Safety-Critical Real-Time Systems (CRTS) require more computing power
- More computational power is delivered by
  - More complex SW
  - More complex HW: **caches**
- Worst Case Execution Time (WCET) must be derived
- Measurement-Based Probabilistic Timing Analysis (MBPTA)
  - Trustworthy WCET estimates on complex hardware (e.g. multiple levels of caches)
  - Some properties required to emanate from the HW

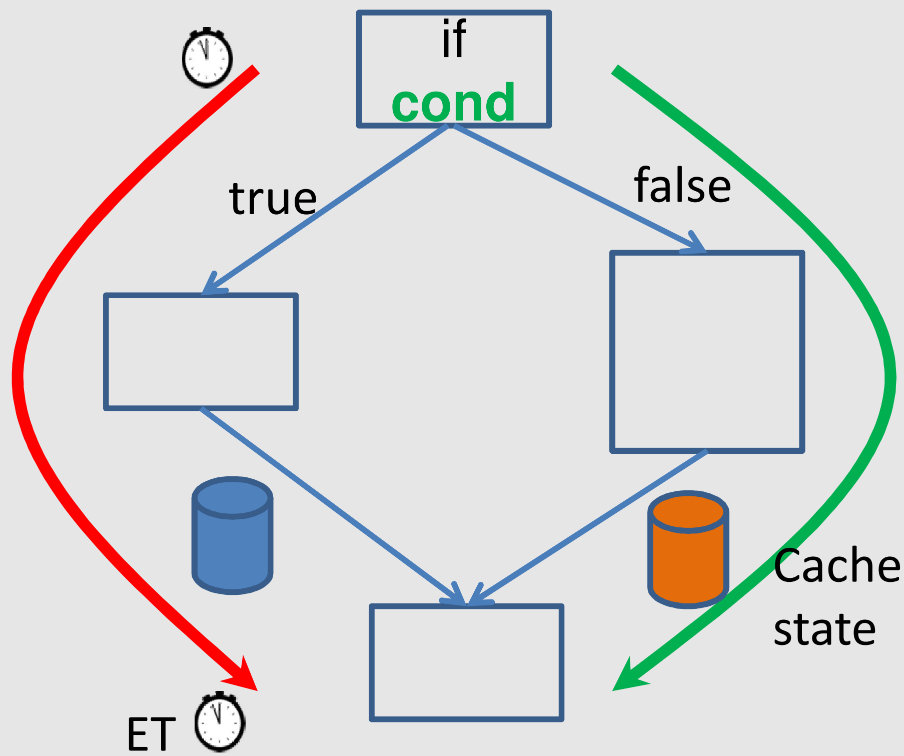
The problem

- Traditional MPBTA provides a pWCET upper-bound of exercised paths at analysis time
- Deriving the pWCET for the program → user has to provide inputs exercising paths leading to highest execution times (Worst-Case Path or WCP)



Conditional control-flow constructs (CFC)

- Complicate Analysis
- In the general case only a subset of the branches of a CFC are going to be captured in the observations
- Impact of the unobserved branches:
  - May be longer execution time that observed branches
  - May leave the stateful resources (e.g. cache) in a *worse state* than observed branches → longer Exec. Time of following code

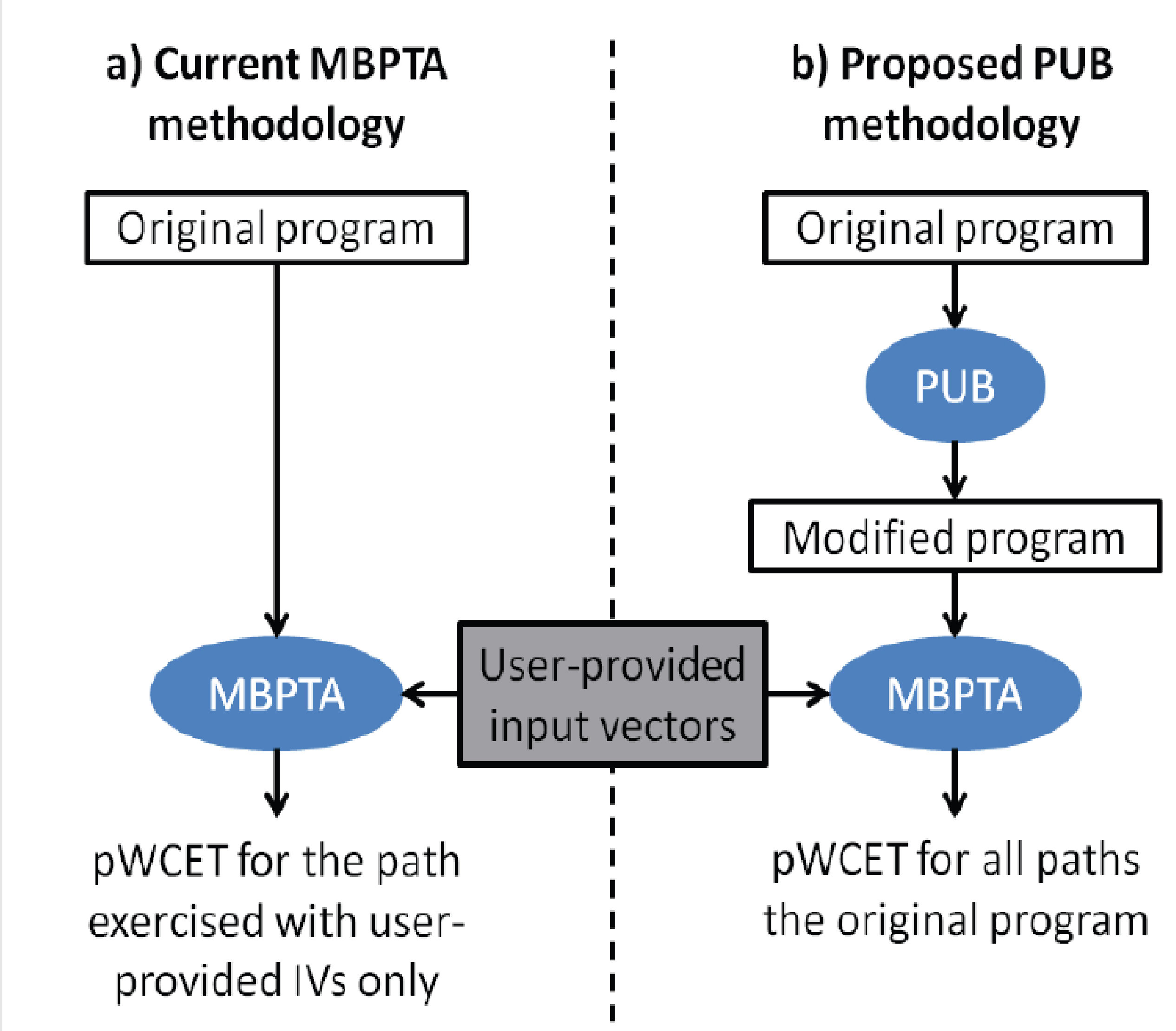


PUB solution

- Path Upper-Bounding (PUB)
  - Relaxes the input requirement from user:
    - No need to provide input vectors exercising WCP
    - Required to have loop iteration bounds (already a hard problem)
  - Provides a pWCET estimate that upper-bounds any path, even when input vectors don't exercise WCP
  - Simplifies Timing Analysis and broadens applicability of MPBTA

How?

- How:
  - Creating an extended version of the original program for analysis
  - Unmodified program is used for deployment



Outline

- Motivation and problem description
- Introduction to MBPTA
- Time Randomised Caches and associated properties
- PUB
- Results
- Conclusion

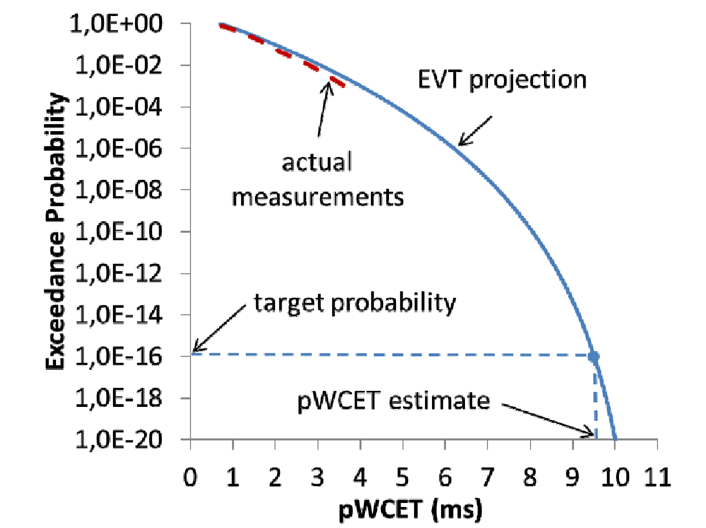
Measurement-Based PTA

- PTA aims at reducing dependence of software timing behaviour on execution history
- How is this done:
  - **Selectively** introduces **randomisation** into the **timing behaviour** of the **hardware** and/or software [1]
    - Jittery resources with high impact on pWCET and hard-to-track state are randomised (e.g. cache)
  - Control of input-data dependent jitter
  - Functional behaviour is left unchanged
- MBPTA
  - Collects execution time of end-to-end runs (observations)
  - Applying Extreme Value Theory (EVT)

[1] Kosmidis et al, Measurement-Based Probabilistic Timing Analysis to Buffer Resources, WCET 2013

Measurement-Based PTA Requirements

- MBPTA requirements:
  - Inherited from EVT:
    - Observations have to be modeled by independent and identically distributed (i.i.d.) random variables [1]
  - Its own requirements [2]
    - At analysis time events affecting execution time need to match or upper-bound deterministically or probabilistically those events at deployment
  - No need to compute that probability [2], unlike SPTA that requires it



[1] L. Cucu et al, Measurement-Based Probabilistic Timing Analysis for Multi-path Programs, *ECRTS 2012*

[2] F. Cazorla et al, Upper-bounding program execution time with extreme value theory," in *WCET* 2013

Outline

- Motivation and problem description
- Introduction to MBPTA
- Time Randomised Caches and associated properties
- PUB
- Results
- Conclusion

MBPTA and Time Randomised caches

- Time-Randomised (TR) Caches [1]
  - Implement Random-Placement and Random-Replacement
  - Provide a probability for each access to be a hit/miss
  - *Decouple addresses from placement and replacement*

- *TR caches provide some properties*
  - Next we review them and show how to use them
  - Those properties do not necessarily hold for time-deterministic caches

[1] *Kosmidis et al, A Cache design for Probabilistically Analysable Real-Time Systems, DATE 2013*

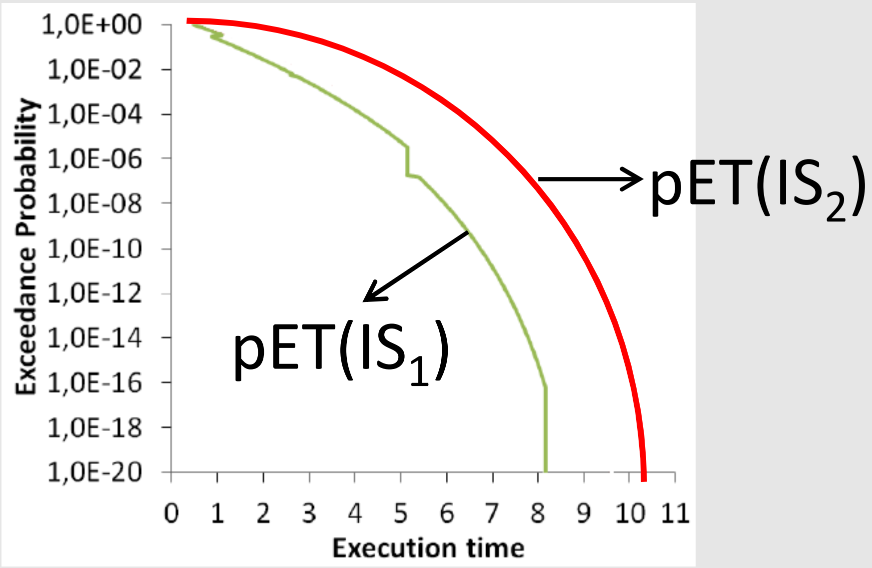
Theorem 1

- *Given an Instruction Sequence (IS) the introduction of any access at any point of the sequence increases its pET.*

IS_1 = A B C B … F

IS_2 = A B ***X*** C B … F

→ $pET(IS_2) >= pET(IS_1)$



PROXIMA

Theorem 1 (cnt'd)

IS_1 = A B C B … F

IS_2 = A B ***X*** C B … F

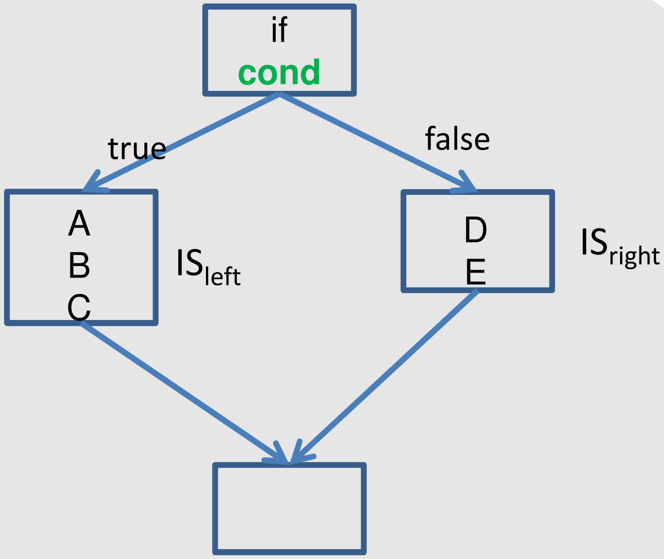
Intuition:
- If ***X*** is a hit, pET increases by a hit latency. Cache state remains the same.
- If ***X*** is a miss, pET increases by a miss latency. Perhaps later ***X'***, which was a miss, becomes a hit. Overall, pET increases at least by a hit latency.

Proof in the paper

- (Probabilistic) cache state (PCS) after executing IS_1 and IS_2 differ due to access to ***X***
  - For the sake of this presentation assume that program finishes
  - Proof showing that extra accesses increase pET despite effects in PCS also in the paper

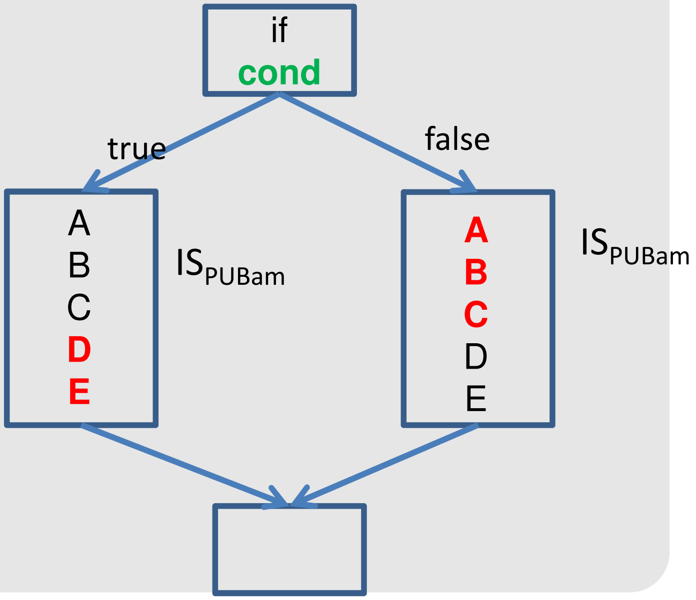
Theorem 1 and CFC Upper-Bounding

- Create a sequence IS_{PUBam} so that
  - $pET(IS_{PUBam}) >= pET(IS_{left})$, and
  - $pET(IS_{PUBam}) >= pET(IS_{right})$



- Straightforward solution
  - $IS_{PUBam} = IS_{left} \cup Is_{right}$
  - In the example: IS_{PUB} = A B C D E

- Based on Theorem 1, it does not matter actual path executed
  - pET through both paths upperbounds both original paths



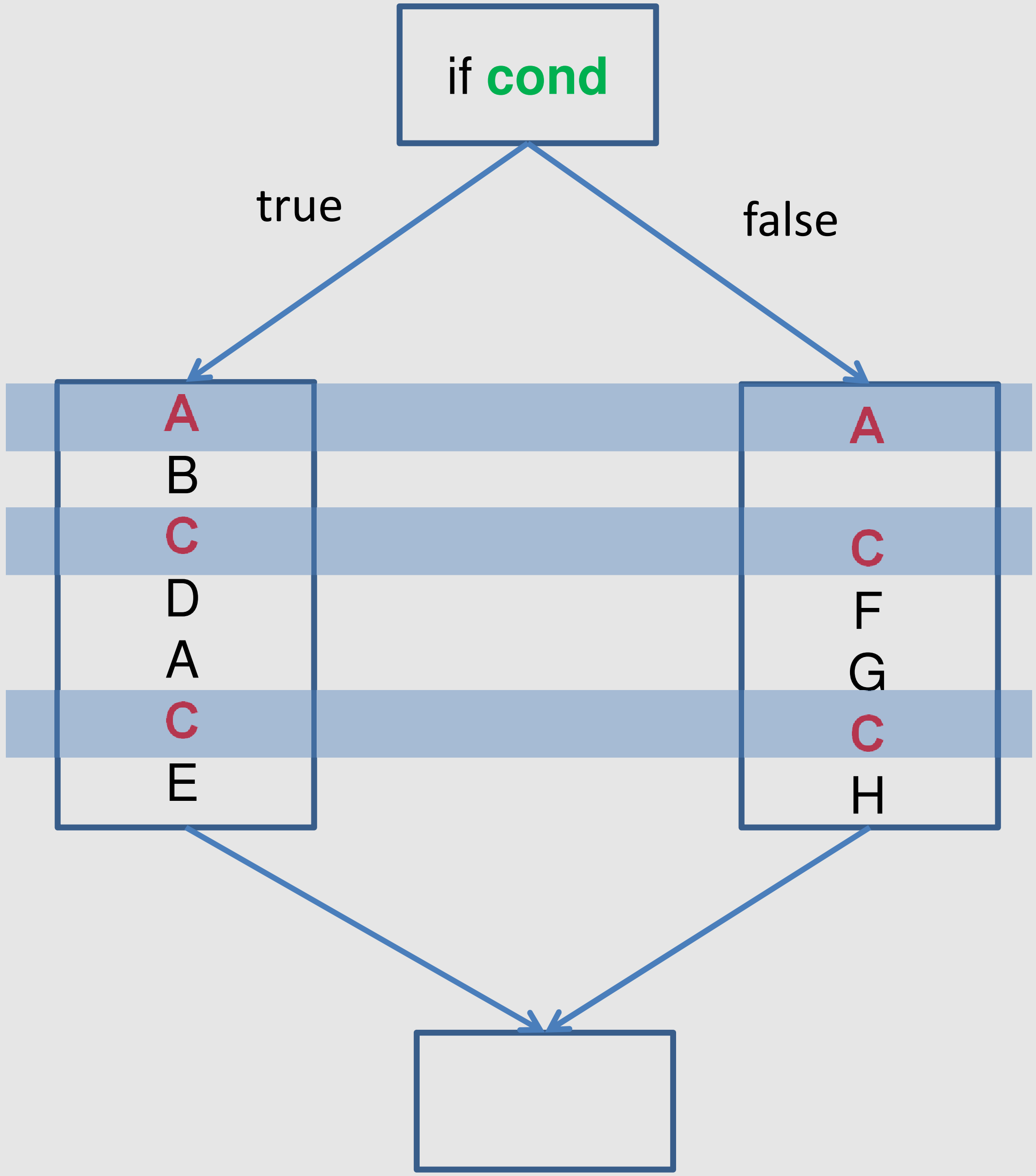
Refining PUBam

- PUBam (address merging)
  - No need to replicate all code
  - IS_{PUBam} must include both IS_{left} and IS_{right}

- Identify repeated sequences in IS_{left} and IS_{right} and avoid replicating them
  - They are already replicated across paths

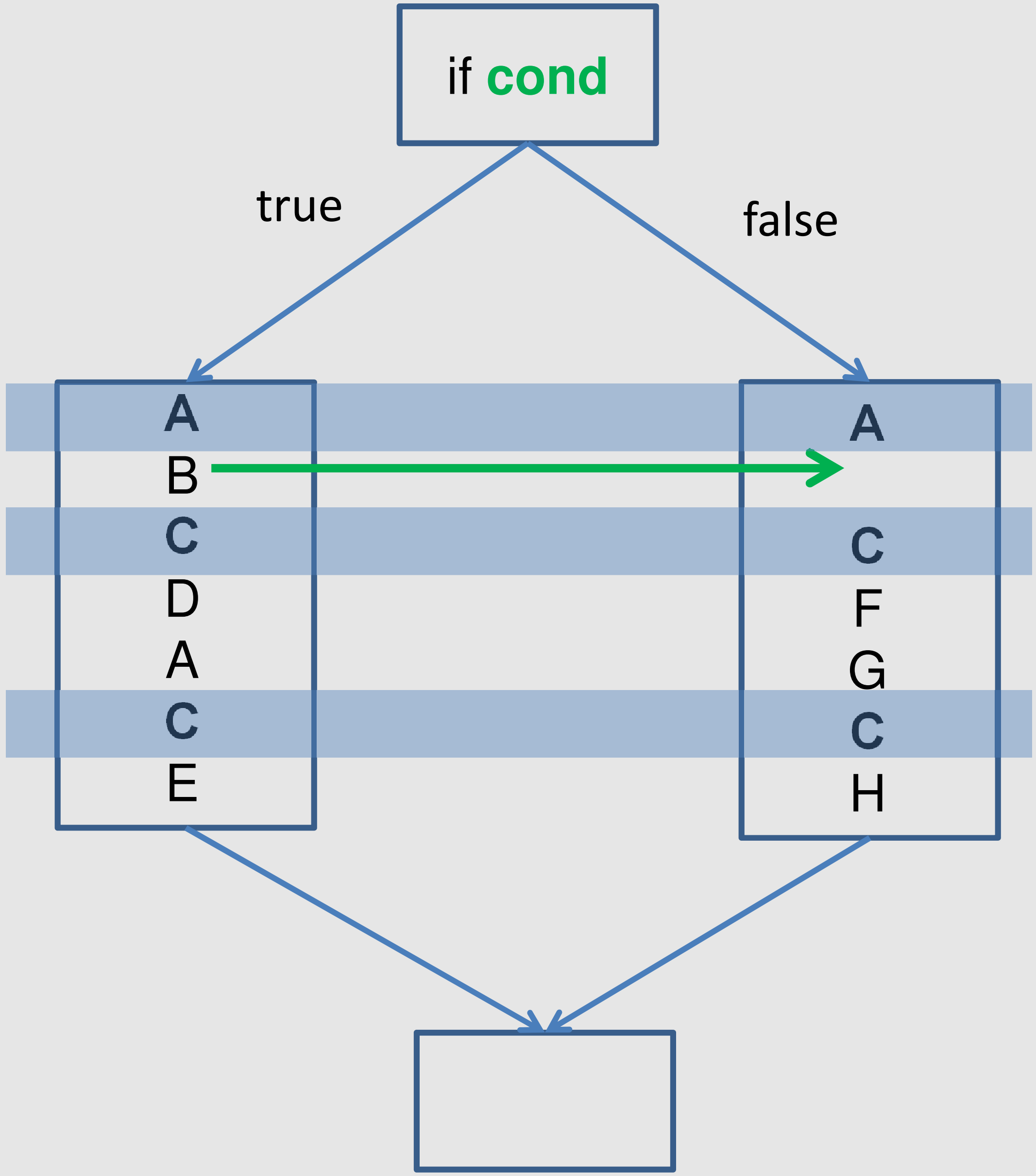
PUBam example

- Identify the longest common access pattern



PUBam example

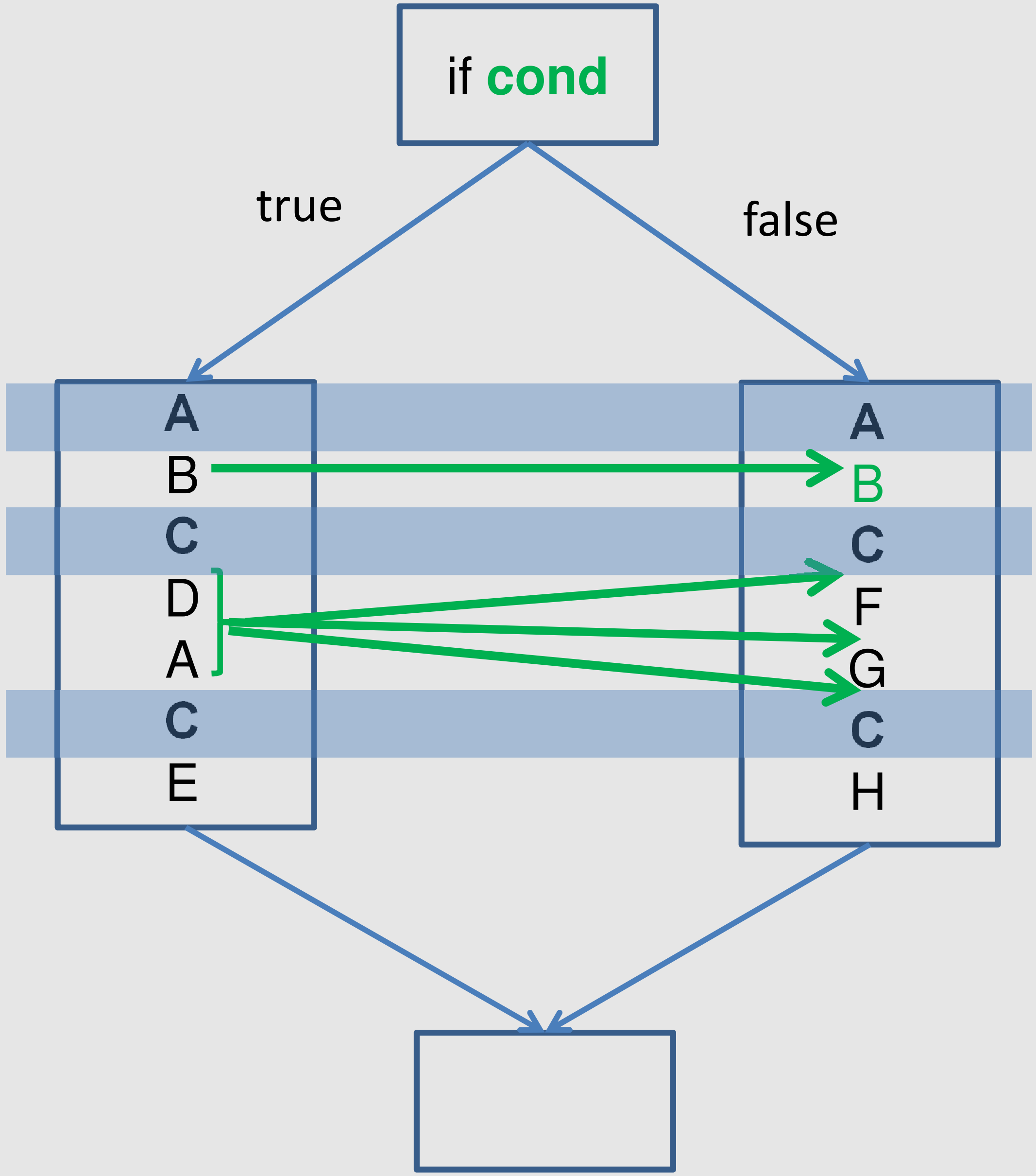
- Identify the longest common access pattern
- Introduce the non common accesses to each path, preserving their relative ordering



PROXIMA

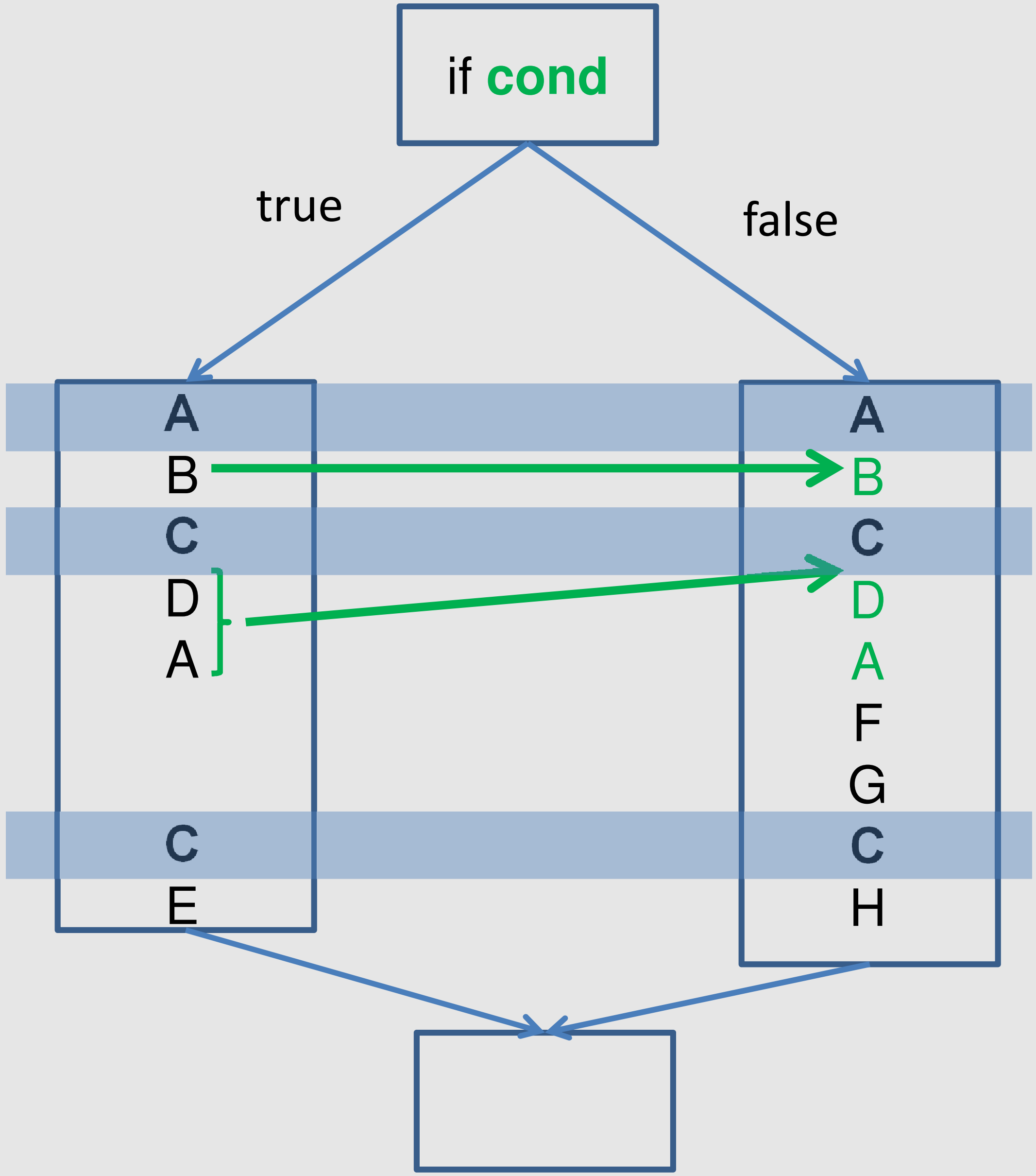
PUBam example

- Identify the longest common access pattern
- Introduce the non common accesses to each path, preserving their relative ordering



PUBam example

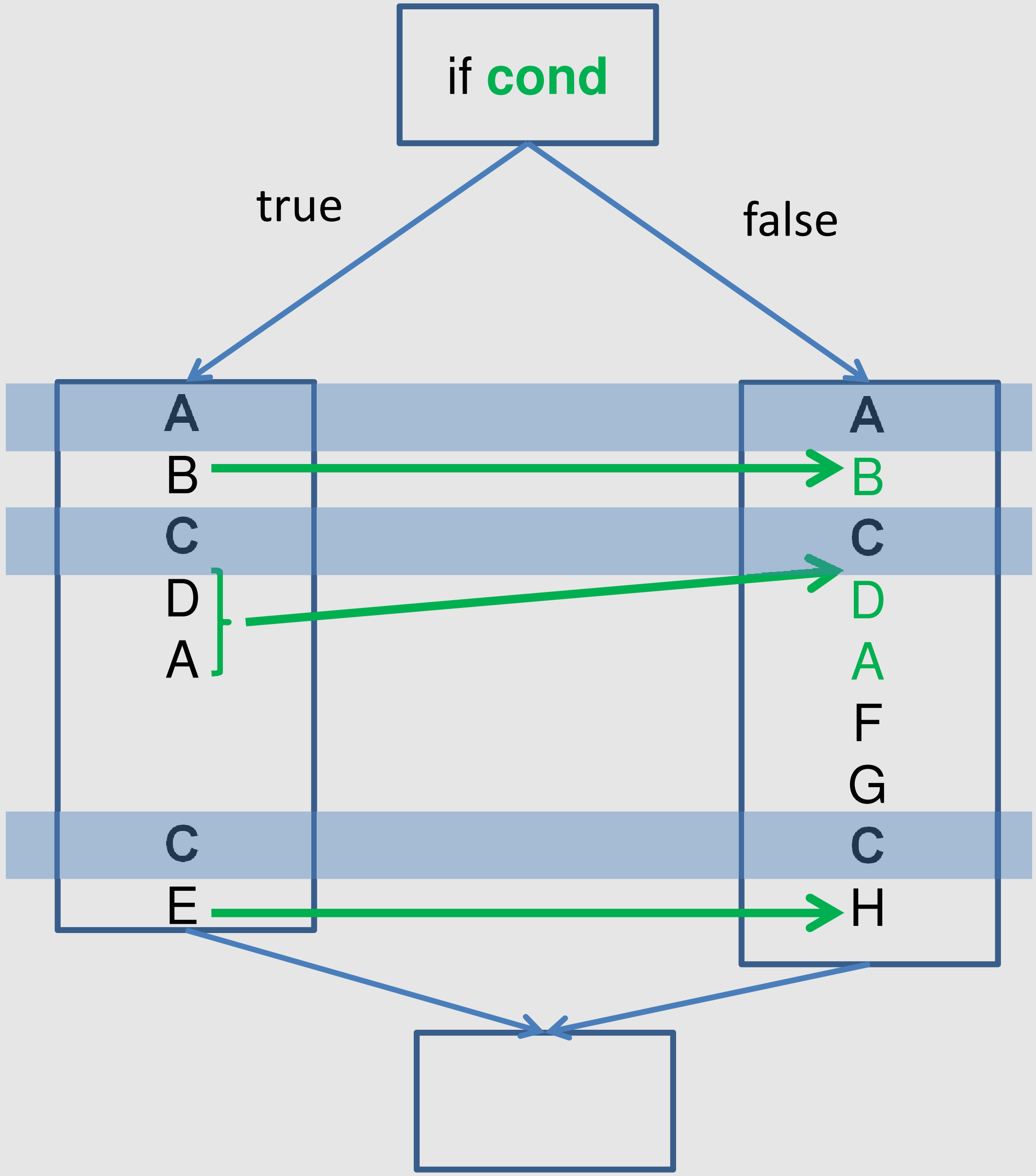
- Identify the longest common access pattern
- Introduce the non common accesses to each path, preserving their relative ordering



PROXIMA

PUBam example

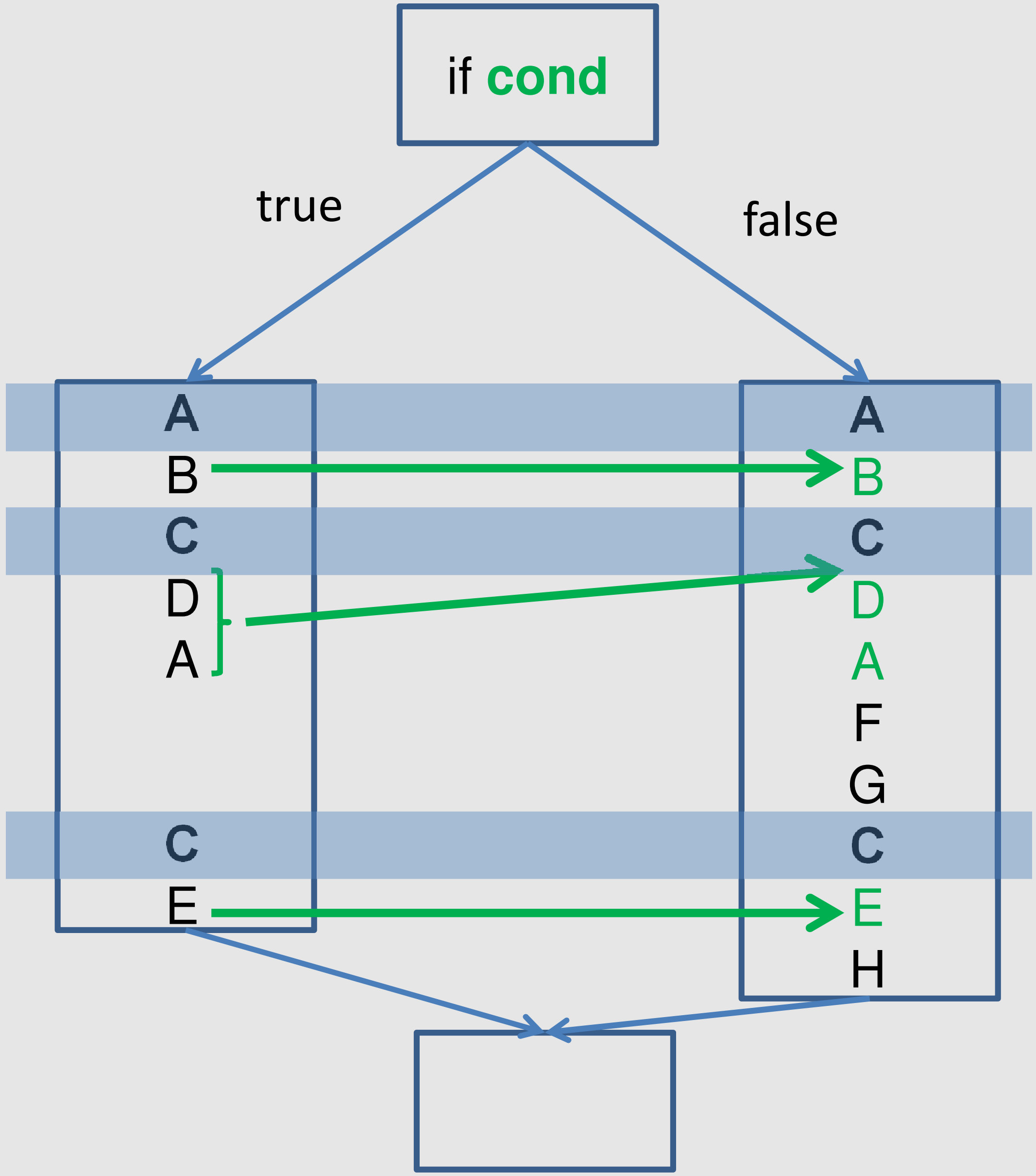
- Identify the longest common access pattern
- Introduce the non common accesses to each path, preserving their relative ordering



PROXIMA

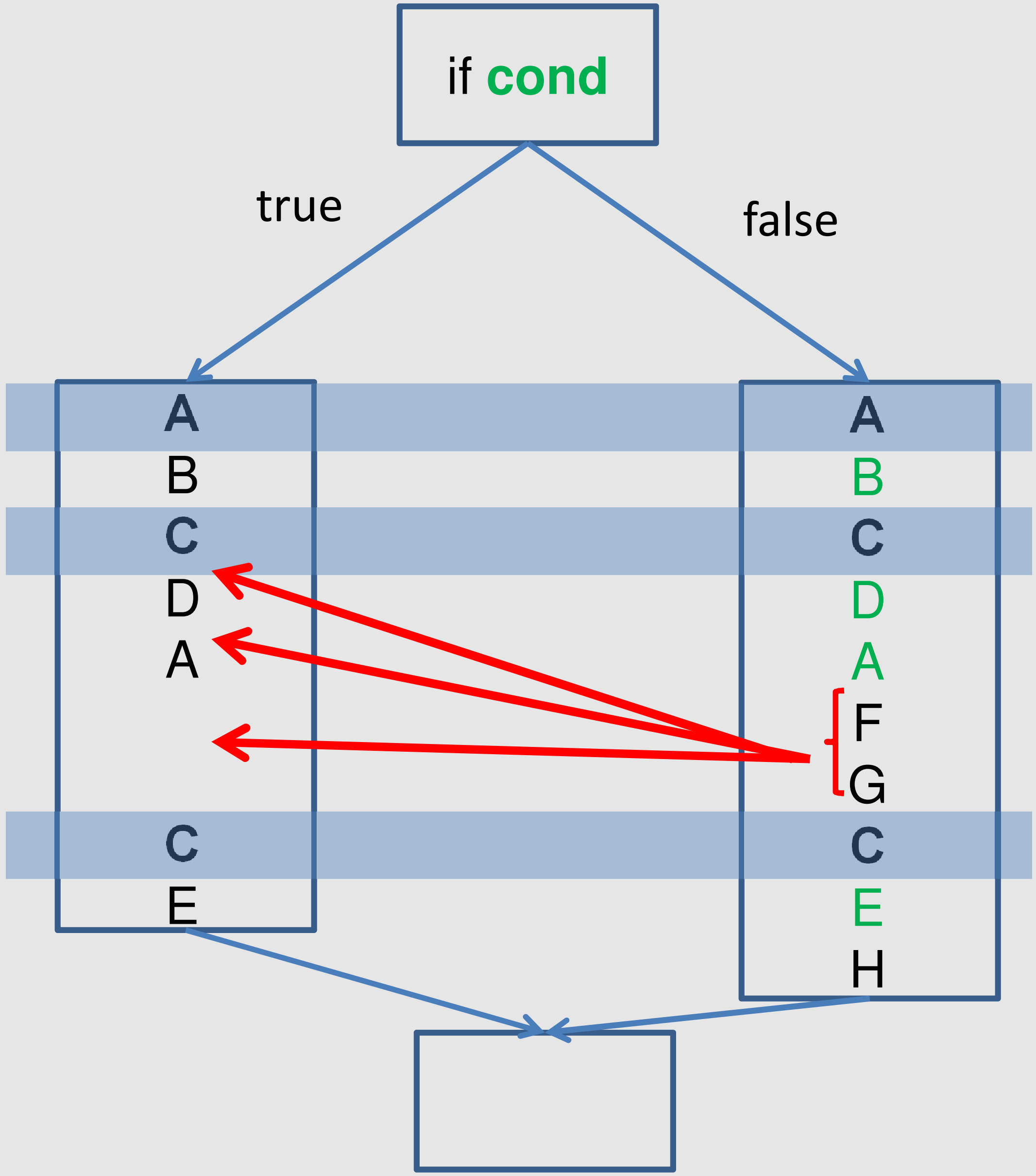
PUBam example

- Identify the longest common access pattern
- Introduce the non common accesses to each path, preserving their relative ordering



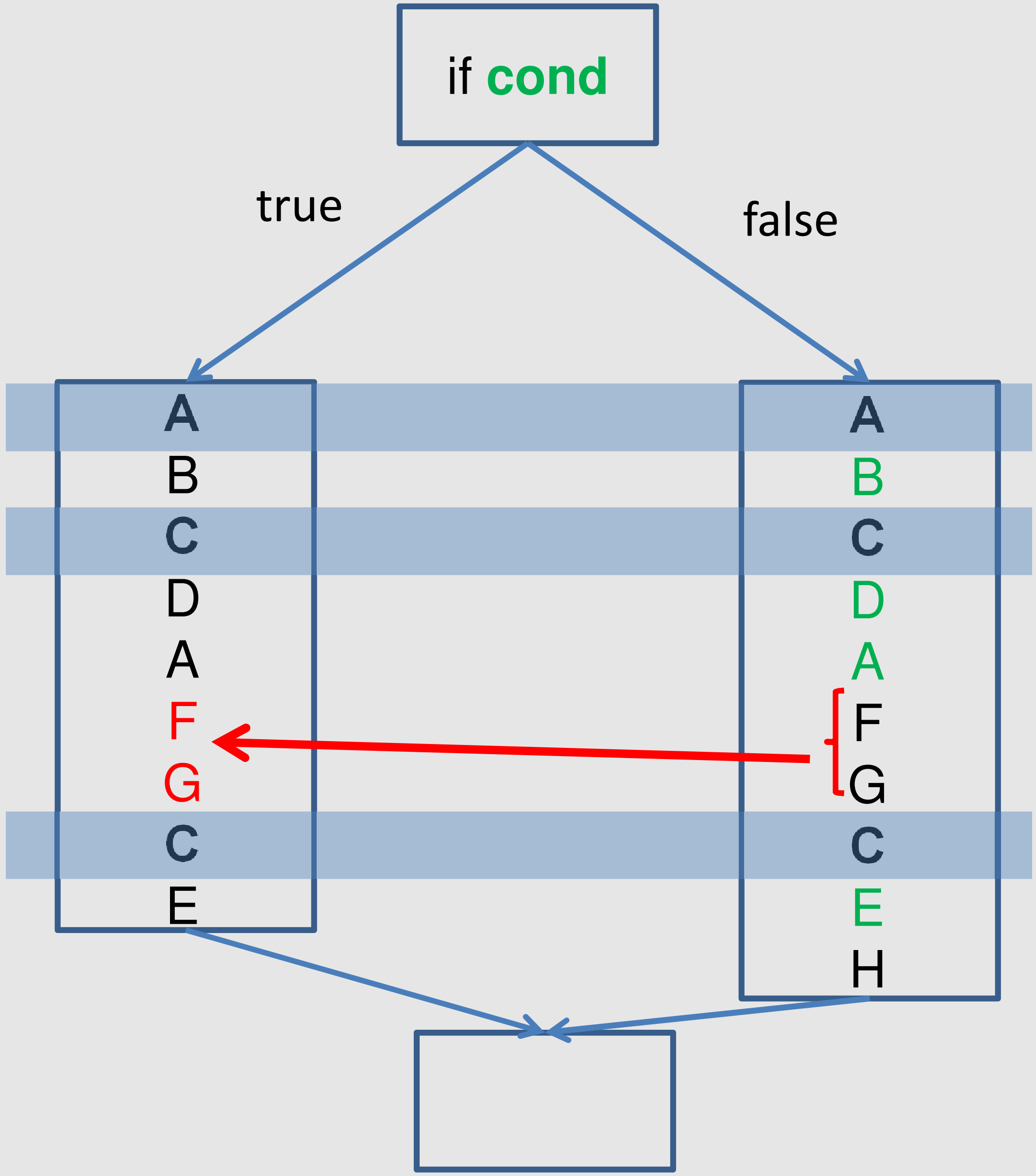
PUBam example

- Identify the longest common access pattern
- Introduce the non common accesses to each path, preserving their relative ordering



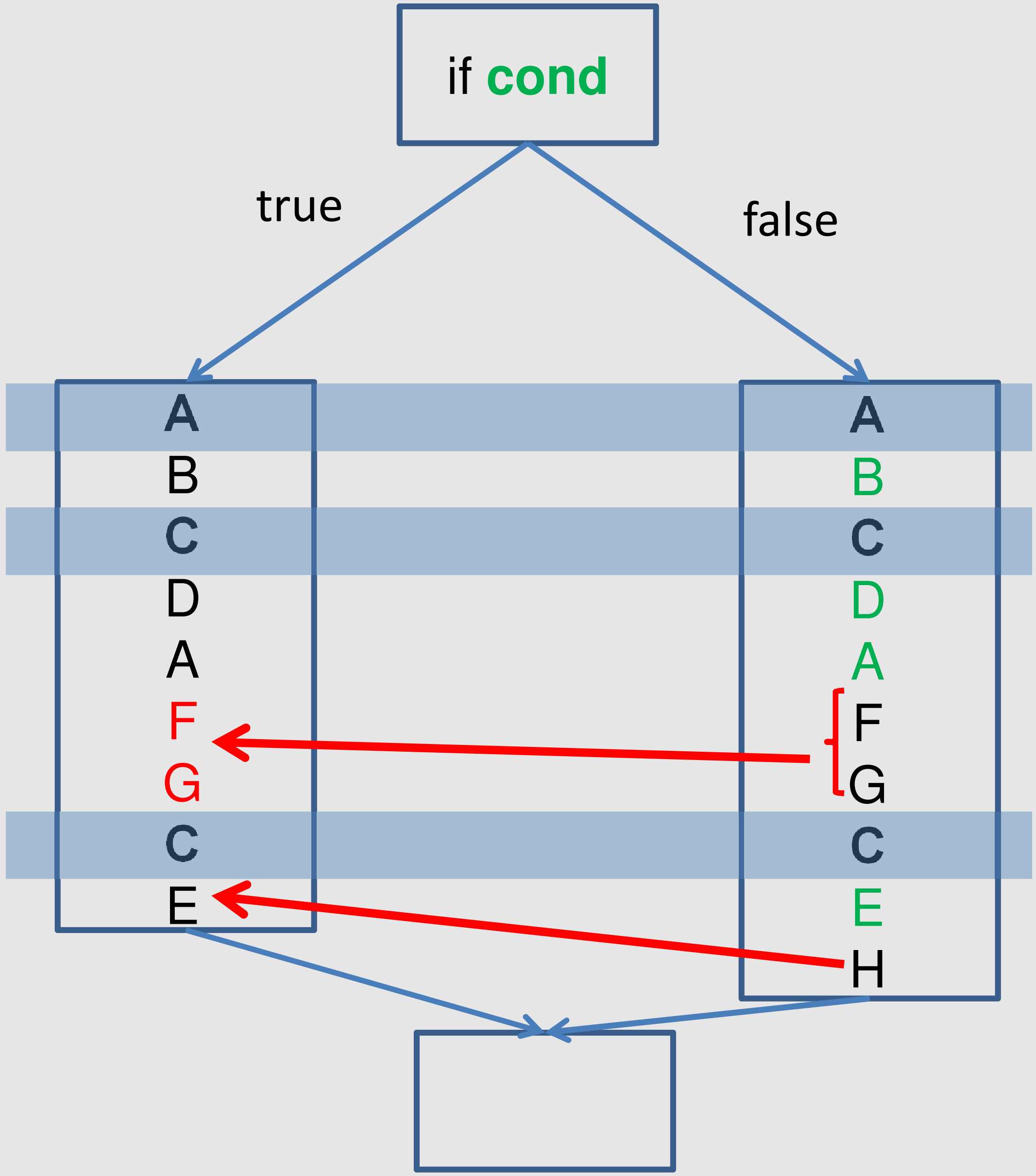
PUBam example

- Identify the longest common access pattern
- Introduce the non common accesses to each path, preserving their relative ordering



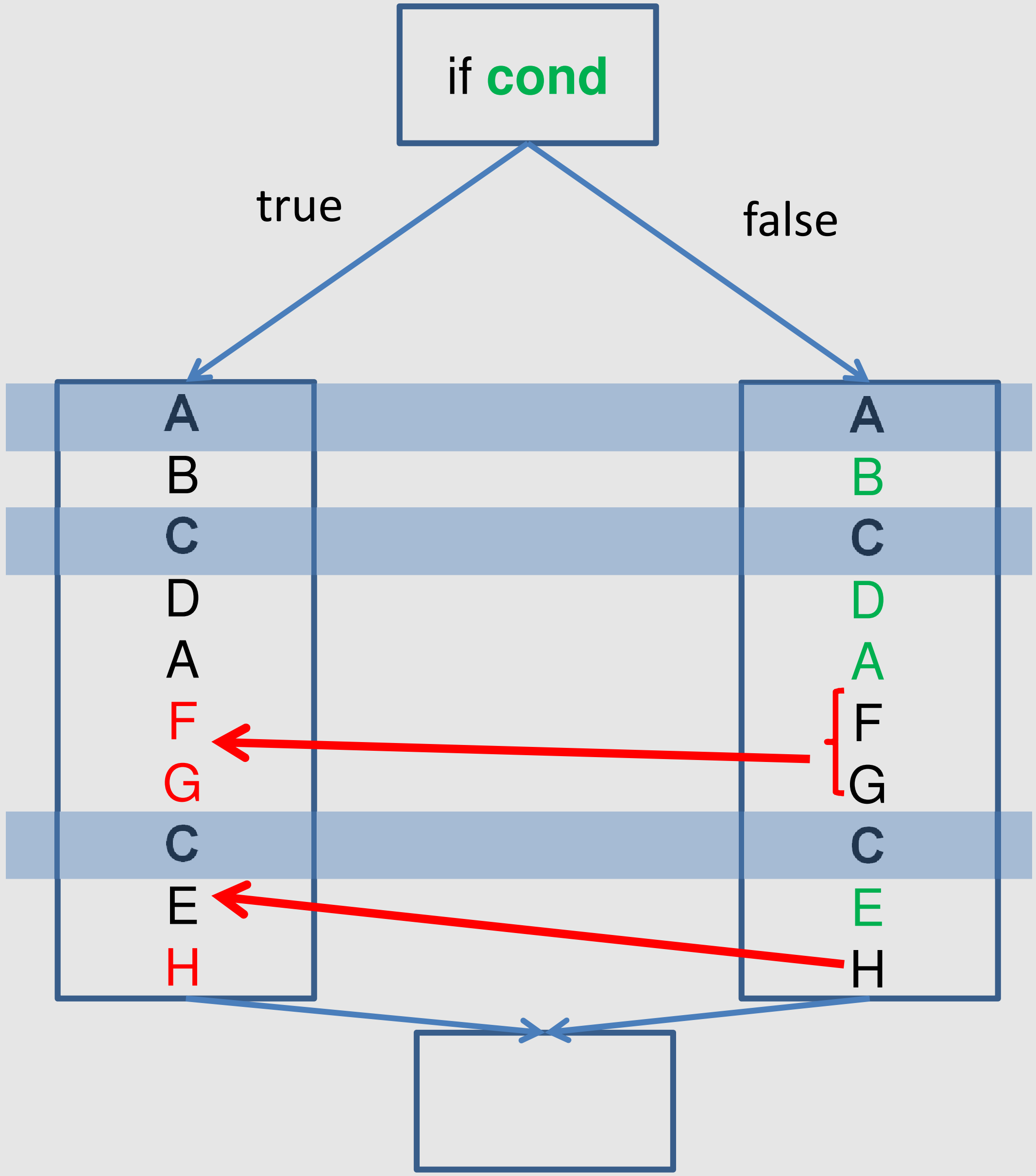
PUBam example

- Identify the longest common access pattern
- Introduce the non common accesses to each path, preserving their relative ordering



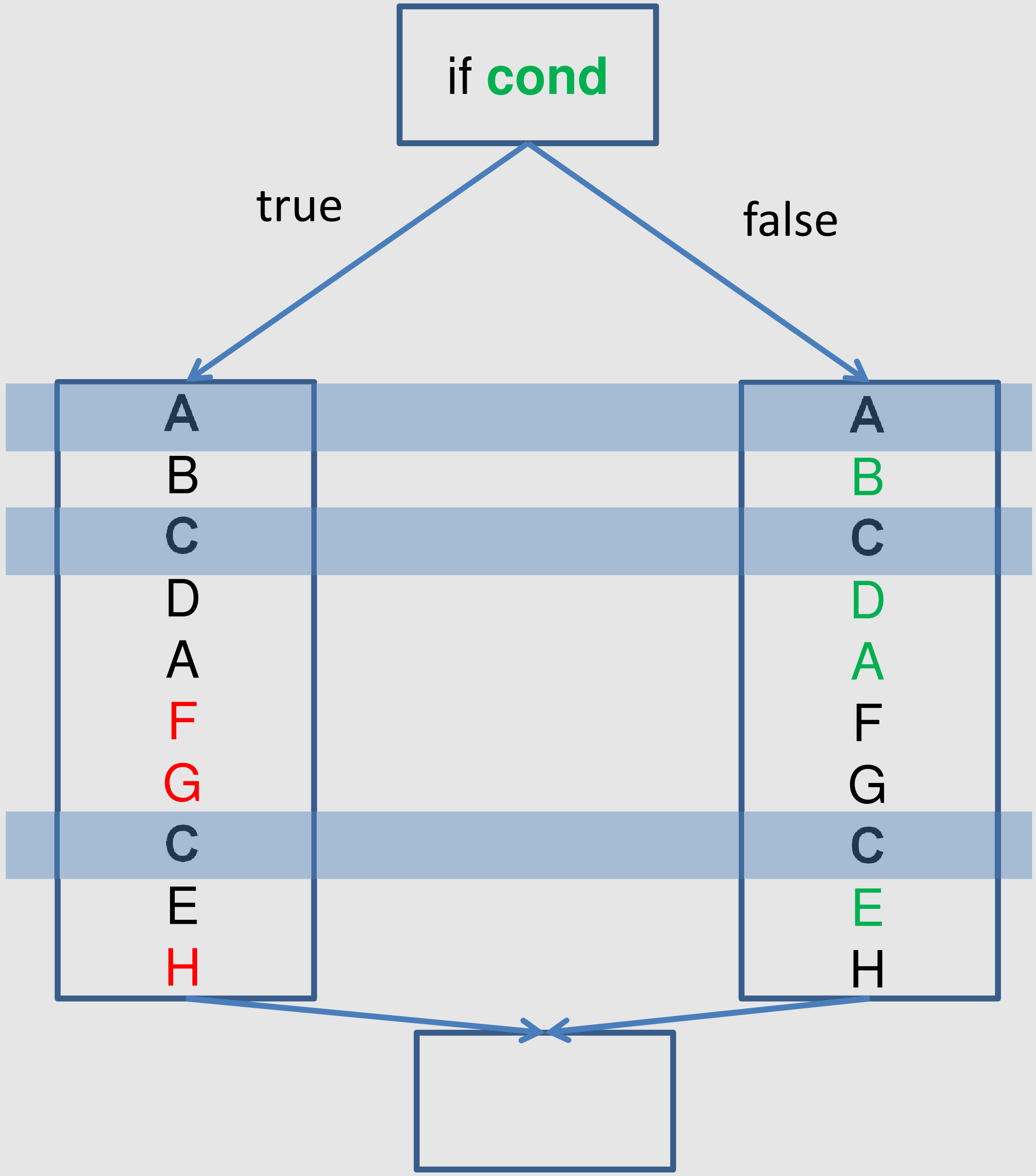
PUBam example

- Identify the longest common access pattern
- Introduce the non common accesses to each path, preserving their relative ordering



PUBam example

- Identify the longest common access pattern
- Introduce the non common accesses to each path, preserving their relative ordering



Different Control Flow Structures

- `If-then`:
  - Same process, assuming that the false branch is empty
- `Switch` (if-then-elsif-…-elsif):
  - More than 2 branches, apply the process for each branch
  - IS_{PUBam} must upper-bound ***ALL*** paths
- `Nested Conditionals`:
  - Apply recursively starting from the inner most branch
- `Function calls in conditionals`:
  - Dummy function accessing the same addresses or
  - PUB Address Aging
  - If the function is called with the same inputs, assume it as common pattern

Outline

- Motivation and problem description
- Introduction to MBPTA
- Time Randomised Caches and associated properties
- PUB
- Results
- Conclusion

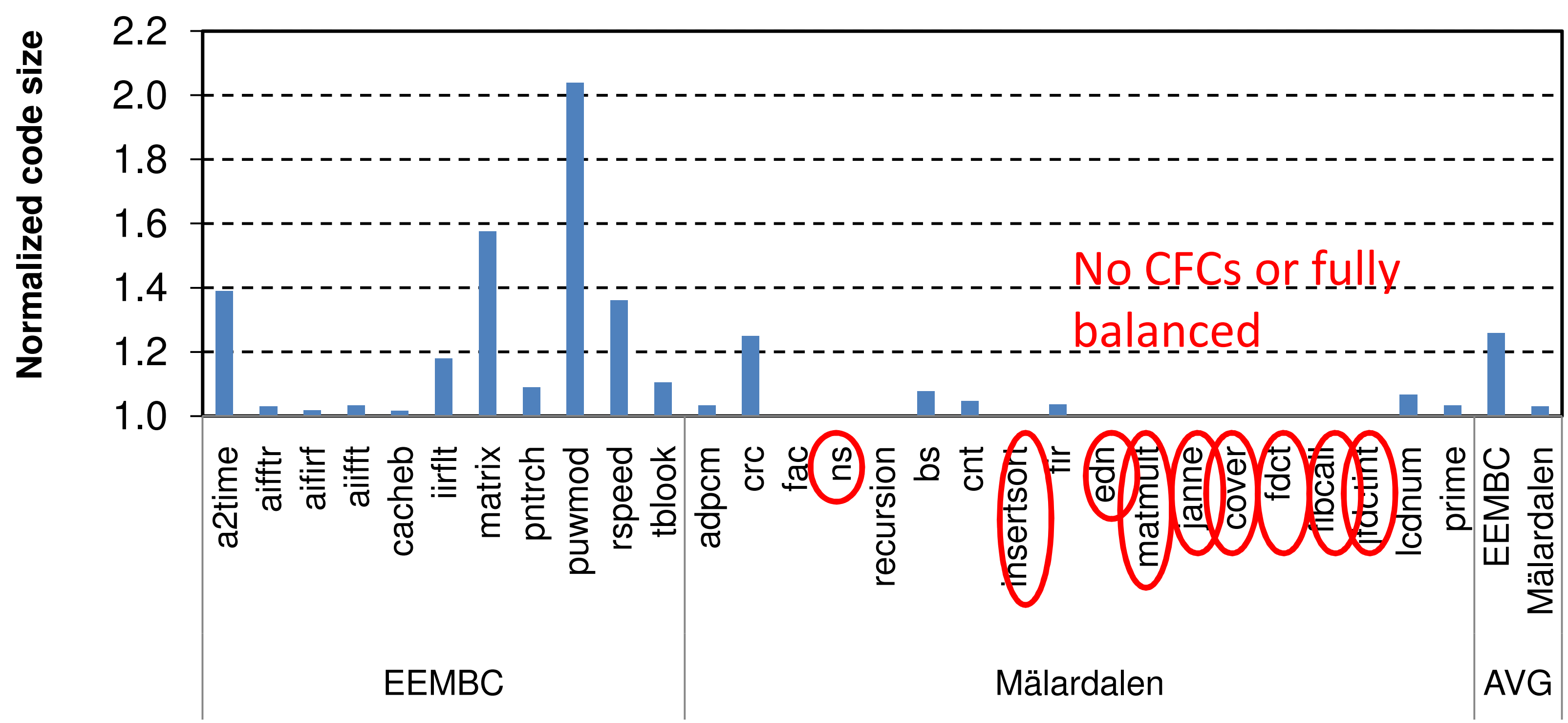
Experimental Setup

- LEON 4-like pipelined processor
- Time-randomised Instruction and Data caches
  - 8KB, 8 way set-associative, 16b line size
  - Hit latency 1 cycle, miss penalty 100 cycles
  - Write-back data cache, dirty evictions cause pipeline stall
- Malardalen and EEMBC automotive benchmarks
  - For some benchmarks we know the worst case input vectors:
    - Malardalen: bs, cnt, insertsort, fir, edn, matmult, janne, cover, fdct, fibcall, jfdctint, lcdnum, prime
- PUBam with standard MBPTA methodology [1]

[1] L. Cucu et al, Measurement-Based Probabilistic Timing Analysis for Multi-path Programs, *ECRTS 2012*

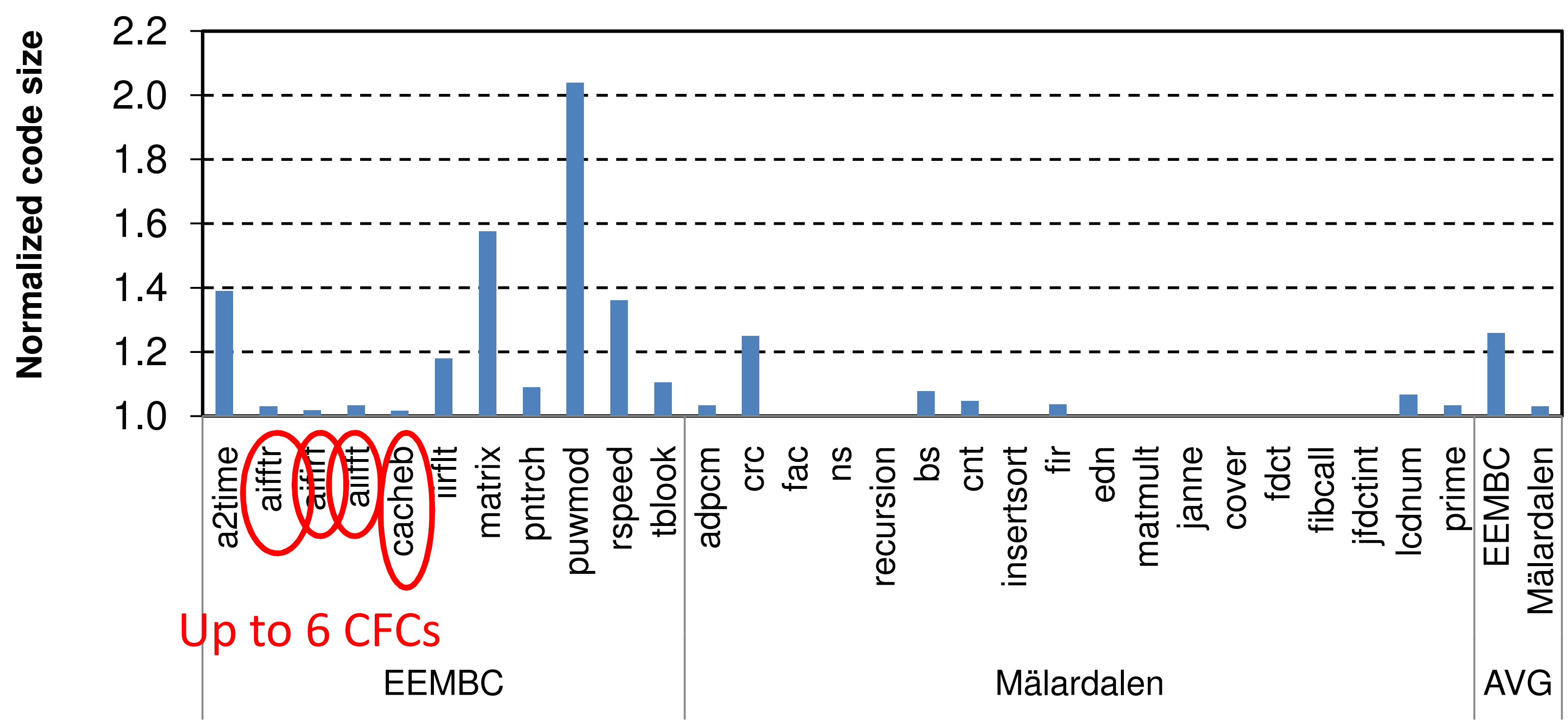
Results

- Impact on Code Size



Results

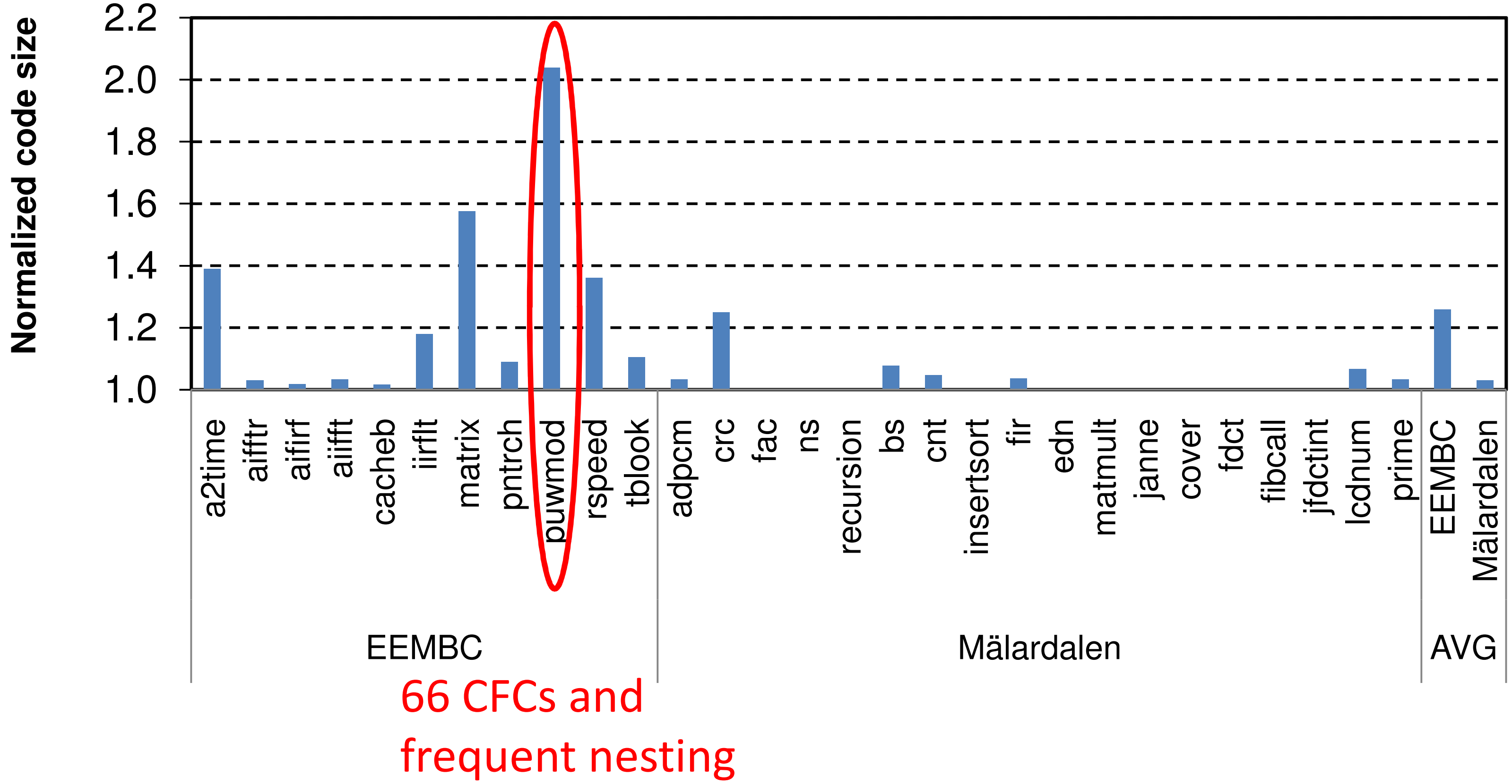
- Impact on Code Size



PROXIMA

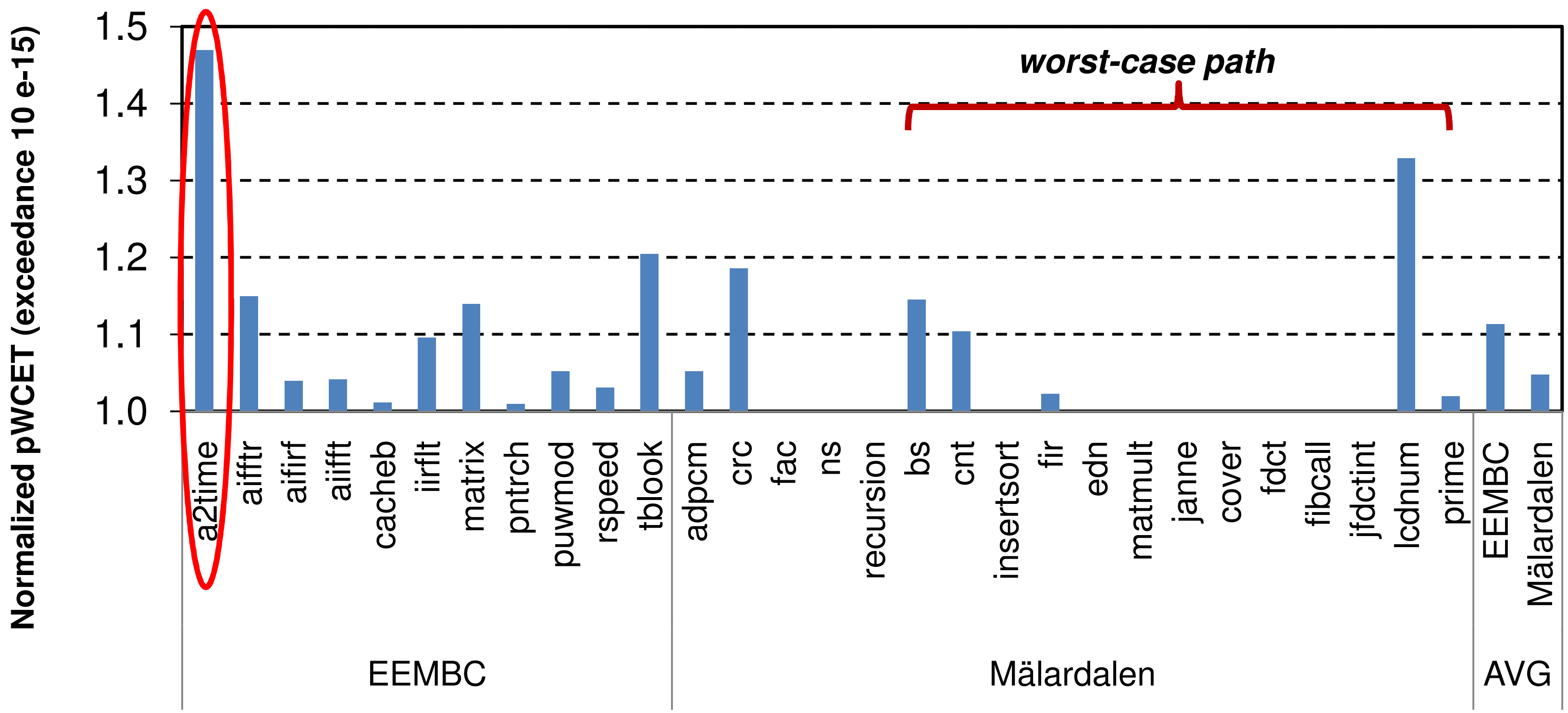
Results

- Impact on Code Size



Results

- Impact on pWCET with respect to MBPTA with user provided input vectors

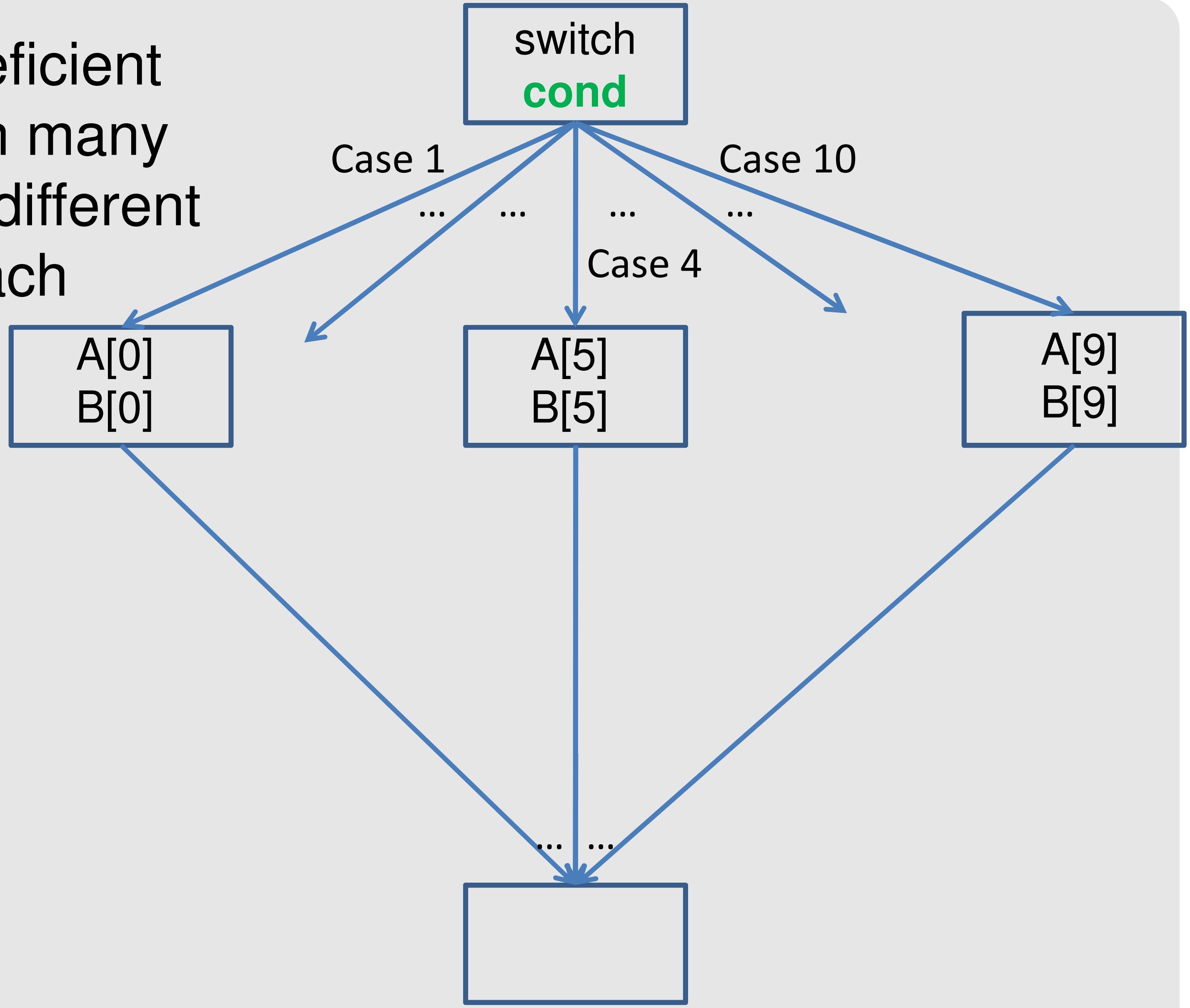


Outline

- Motivation and problem description
- Introduction to MBPTA
- Time Randomised Caches and associated properties
- PUB
- Results
- PUBaa
- Conclusion

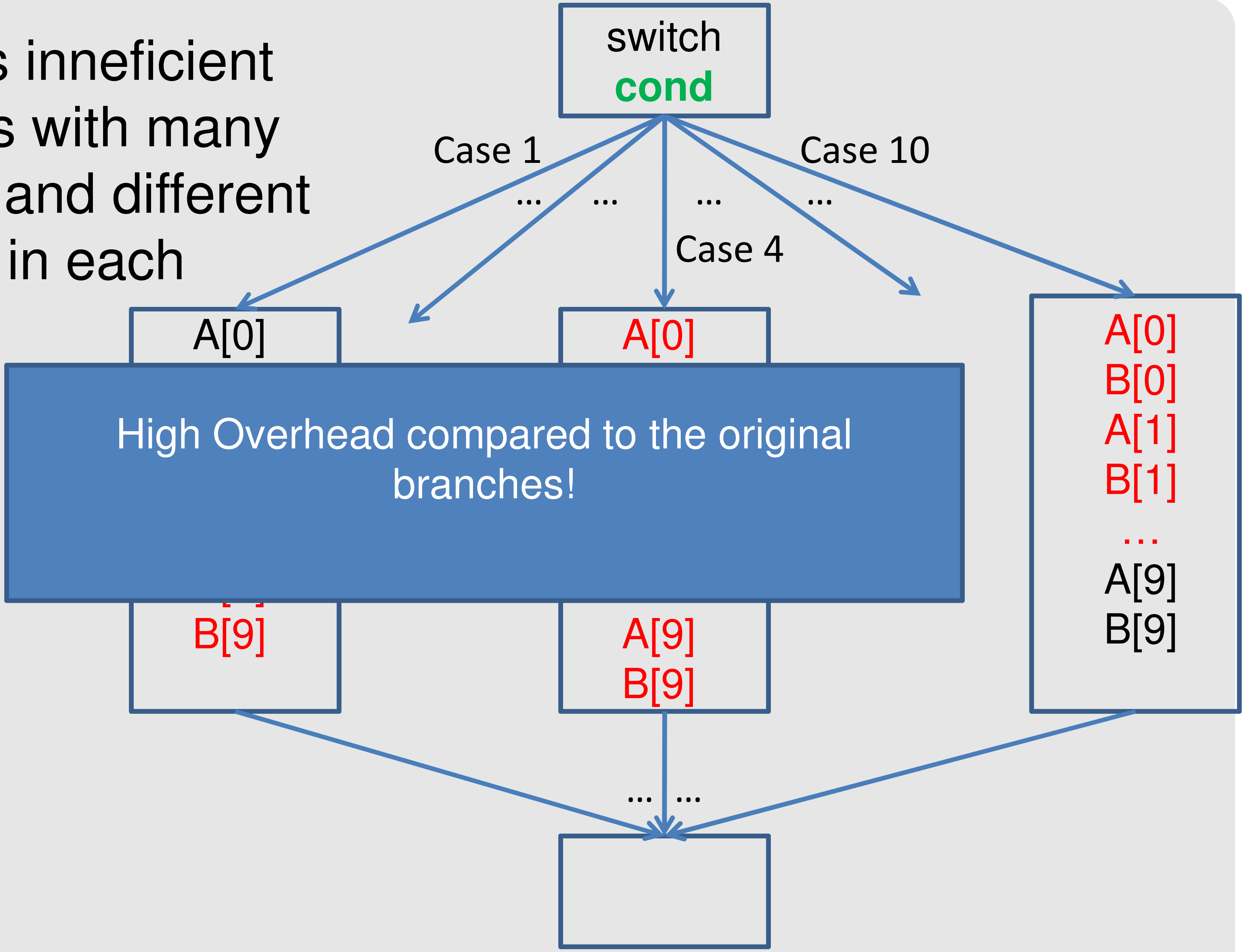
PUBam Inefficiency

- PUBam is inneficient with CFCs with many branches and different accesses in each branch



PUBam Inefficiency

- PUBam is inneficient with CFCs with many branches and different accesses in each branch



Theorem 2

- Given an instruction sequence (IS), if any access is replaced by a unique access (U), the pET of the new sequence can only increase (or remain the same)
  - Unique access: a unique access causes a miss and does not bring any benefit (e.g., its data are never reused)

IS_1 = A B C B … F

IS_2 = A B ***U*** B … F

→ $pET(IS_2) >= pET(IS_1)$

Theorem 2 (cnt'd)

IS_1 = A B C B … F

IS_2 = A B ***U*** B … F

Intuition:

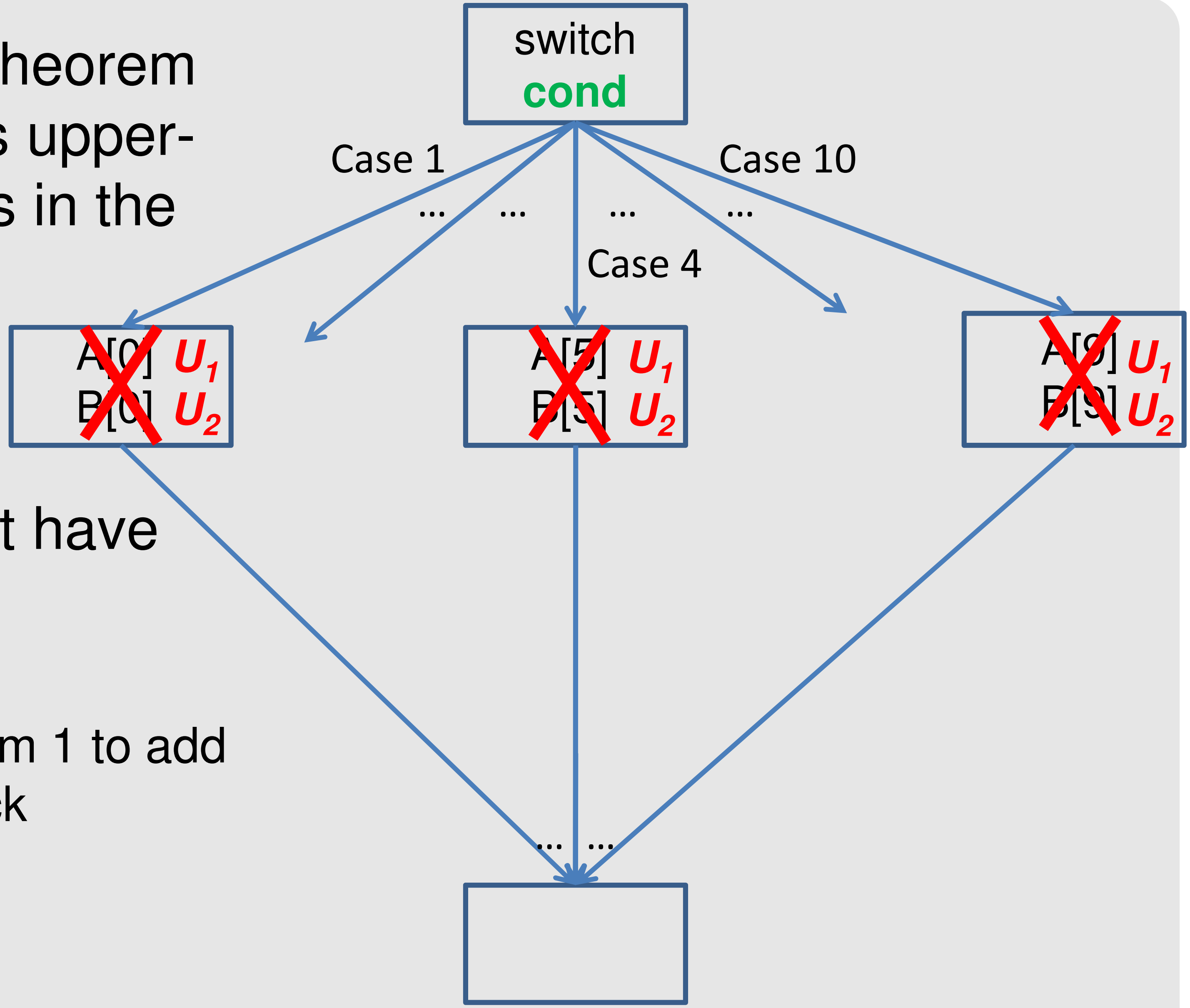
- If access replaced ***C*** is a hit, pET increases as we replace a hit by a miss. ***U*** does not bring any benefit for later instructions
- If access replaced ***C*** is a miss, pET remains the same. ***U*** does not bring any benefit for later instructions.

Proof in the paper

PUBaa (address aging)

- According to Theorem 2 now all paths upper-bound all paths in the original code

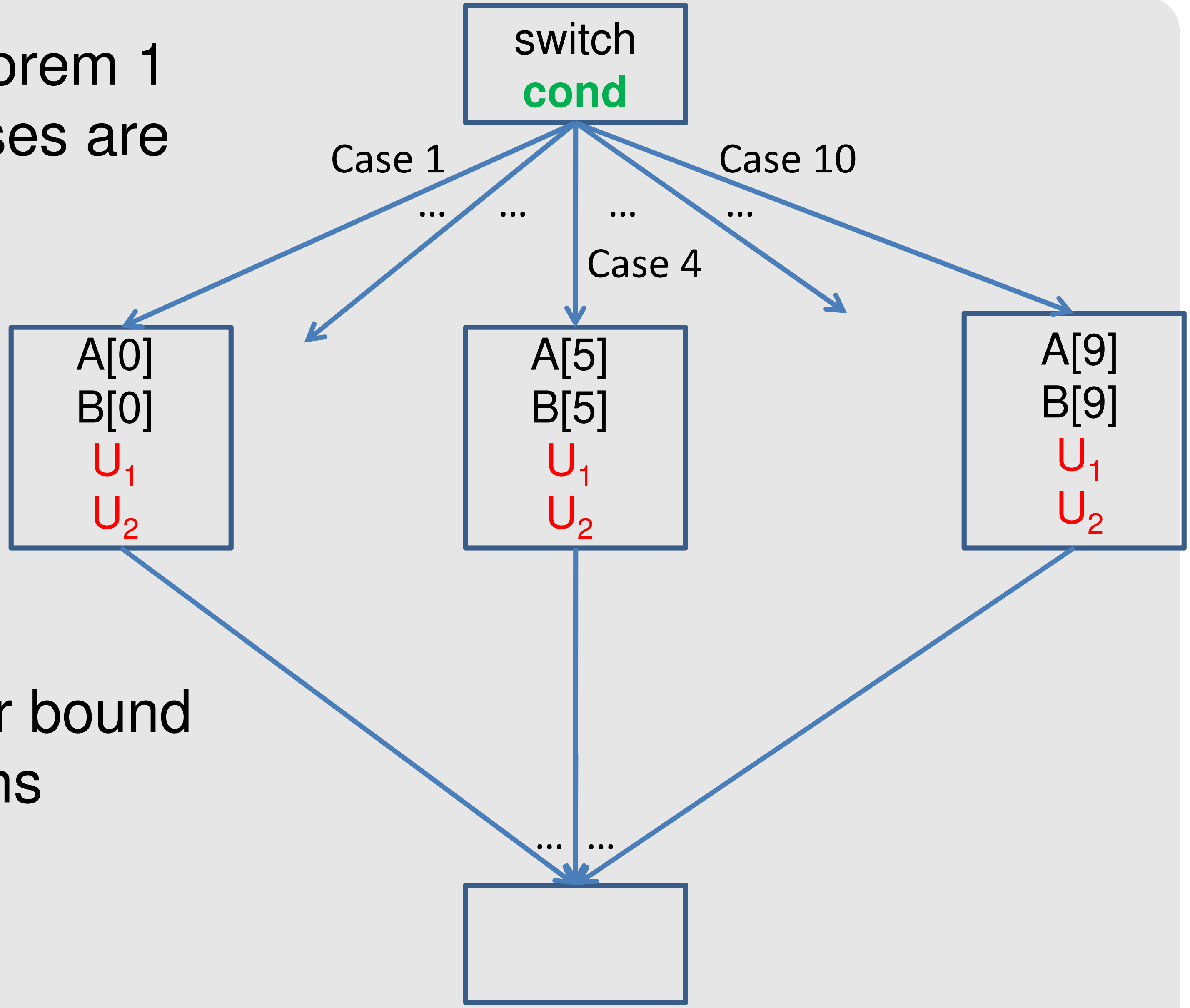
- But they do not have their original functionality
  - Apply Theorem 1 to add accesses back



PUBaa (address aging)

- Based on Theorem 1 original accesses are added

- All paths upper bound all original paths



Comparison of PUB variants

- PUBam

  Efficient with CFCs with few paths: if-then-else, if-then

  - Few accesses
  - Similar accesses in each branch
  - High imbalance

- PUBaa

  - Efficient when paths in CFC are many (switch statements)

Orthogonal and complementary to each other!

Implementation

- PUBam: Introduced code must not modify the functionality of the program or generate any exceptions
  - Introduce loads to a non-modifiable register (r0 in SPARC/MIPS) or
  - Use any free register
- PUBaa: Unique accesses
  - Non-repeated accesses to a dummy data structure or
  - HW support: instruction that always misses and fetches nothing

Implementation

- Core latency:
  - Leon4 like processor, with fixed core-instruction latency
  - Paths are also balanced with the core instructions from other paths using a non-modifiable register for result
- Code Alignment:
  - Keep same cache-line alignment during balancing paths
  - Reuse-distance is affected by cache line size
  - Size of inserted code is exact multiple of cache line size
- PUB solution for instruction caches described in the paper
  - Relies mostly on Theorem 1
  - Reuses code introduced by PUBam and PUBaa to reduce its overhead

Outline

- Motivation and problem description
- Introduction to MBPTA
- Time Randomised Caches and associated properties
- PUB
- Results
- PUBaa
- Conclusion

Conclusions

- Traditional MBPTA is based on user provided input vectors to determine pWCET
- In this paper we propose PUB
  - Builds on properties of Time-Randomised Caches
  - Upper-bounds the execution time of all program paths, using with a single input-vector
  - Creates an extended version of the program, which works on top of the traditional MBPTA
  - The unmodified binary is used for deployment
- 5% and 11% slowdown on average compared to pWCET computed with MBPTA for Malardalen and EEMBC.

PROXIMA

PUB: Path Upper-Bounding for Measurement-Based Probabilistic Timing Analysis

Leonidas Kosmidis, Jaume Abella, Franck Wartel,
Eduardo Quiñones, Antoine Colin, Francisco J. Cazorla

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

CONSEJO SUPERIOR DE INVESTIGACIONES CIENTÍFICAS

AIRBUS

RAPITA SYSTEMS LTD

Madrid, July 11th, ECRTS 2014
www.proxima-project.eu

This project and the research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7 / 2007-2013] under grant agreement 611085