

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 n° 249100.

The logo for PROARTIS features the word "PROARTIS" in a bold, black, sans-serif font. A blue swoosh underline starts under the 'P' and curves upwards and to the right, passing behind the 'ARTIS' part of the word. The entire logo is positioned above a thin horizontal line.

PROARTIS

Hardware Architectural Solutions

Leonidas Kosmidis, Jaume Abella, Eduardo Quiñones, Francisco J. Cazorla

Barcelona Supercomputing Center (BSC-CNS)

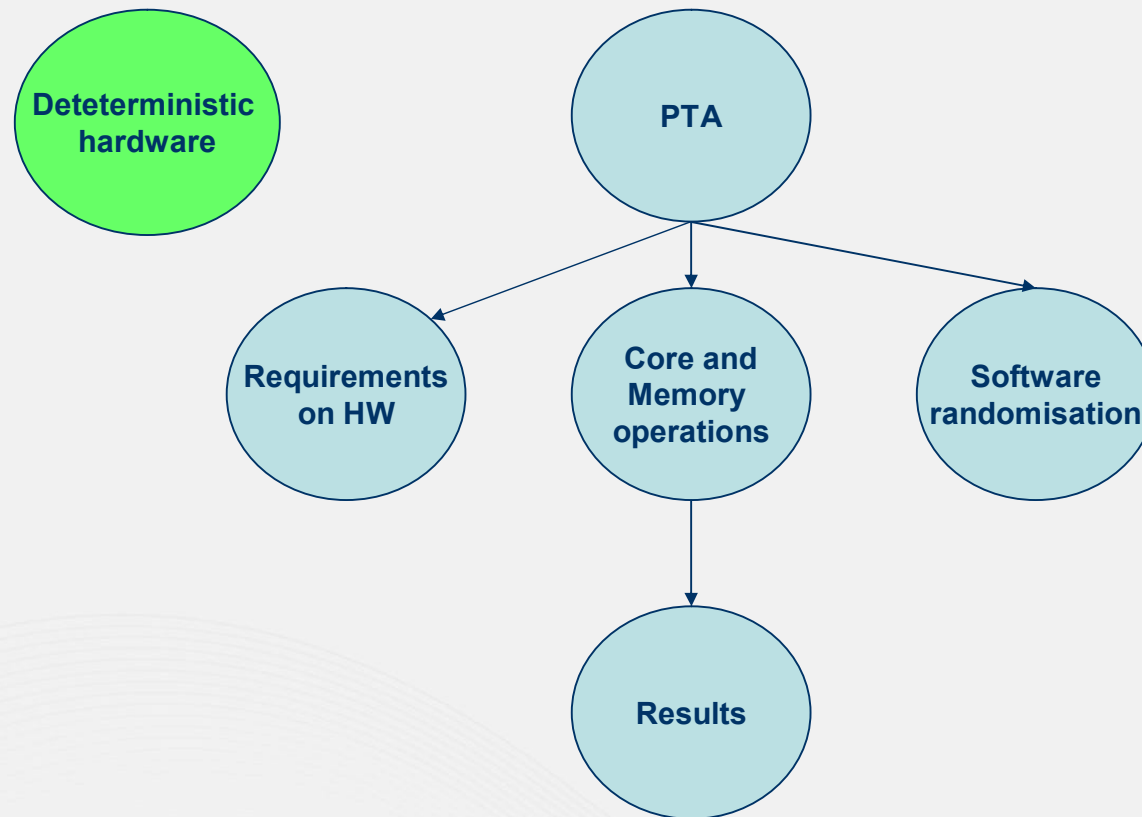
22 January 2013



Berlin

www.proartis-project.eu

Outline



Conventional Hardware

- *Behaviour is deterministic*
- *Deterministic → Easy to predict?*
 - Detail Information about EH required
 - Interaction between HW components → hard to predict
- *Dependence on EH*
 - Dependence on non-functional aspects of the applications and HW
 - E.g., particular addresses where code/data are mapped in memory
 - E.g., data-dependent latencies of some resources
 - Those aspects may be unknown at analysis time
 - Program location in memory
 - Relative location of the OS
 - Location in memory of function stack, heap, etc.
 - Pointers, data dependent indices for arrays, etc.
 - Data-dependent latencies for ALUs

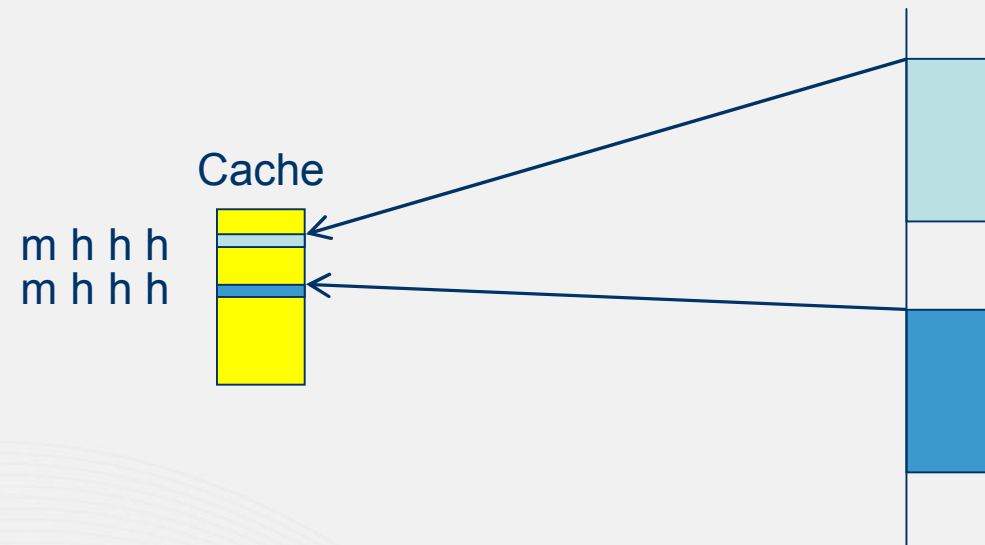
Example

- *What if vect1 and vect2 addresses cannot be determined?*
 - We cannot determine whether accesses will hit or miss in most of the cases

```
Void foo(int * vect1, int * vect2, int size) {  
    for (int i=0; i<size; i++) {  
        vect1[i] = vect1[i] + vect2[i];  
    }  
}
```

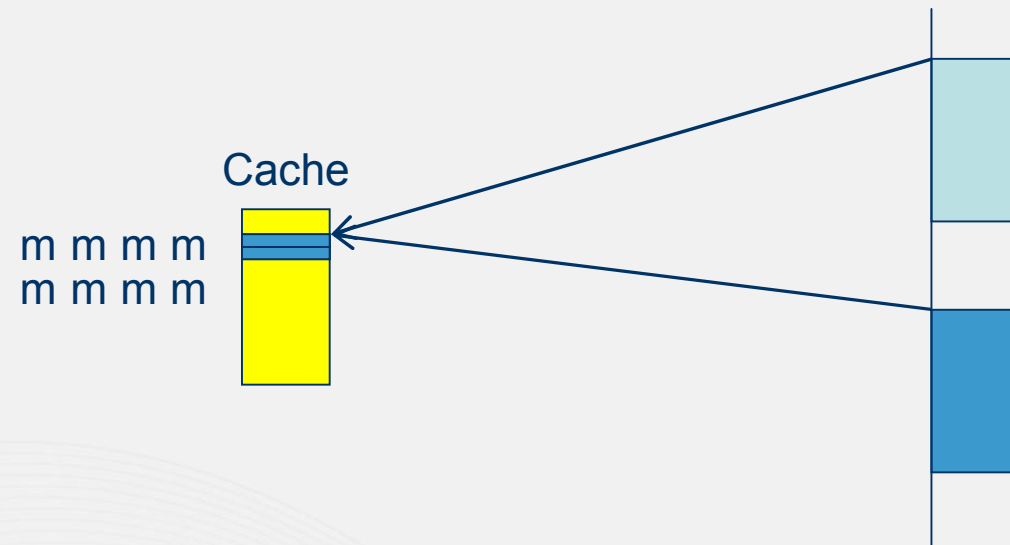
Example (2)

- *Potential (nice) behaviour*



Example (3)

- *Potential (ugly) behaviour*



Example (3)

- Effect of unknown addresses: STA

A _{mru}	B _{lru}
C _{mru}	D _{lru}
E _{mru}	F _{lru}
G _{mru}	H _{lru}
I _{mru}	J _{lru}
K _{mru}	L _{lru}
M _{mru}	N _{lru}
O _{mru}	P _{lru}

(a) initial state

∅ _{mru}	A _{lru}
∅ _{mru}	C _{lru}
∅ _{mru}	E _{lru}
∅ _{mru}	G _{lru}
∅ _{mru}	I _{lru}
∅ _{mru}	K _{lru}
∅ _{mru}	M _{lru}
∅ _{mru}	O _{lru}

(b) 1 unknown address

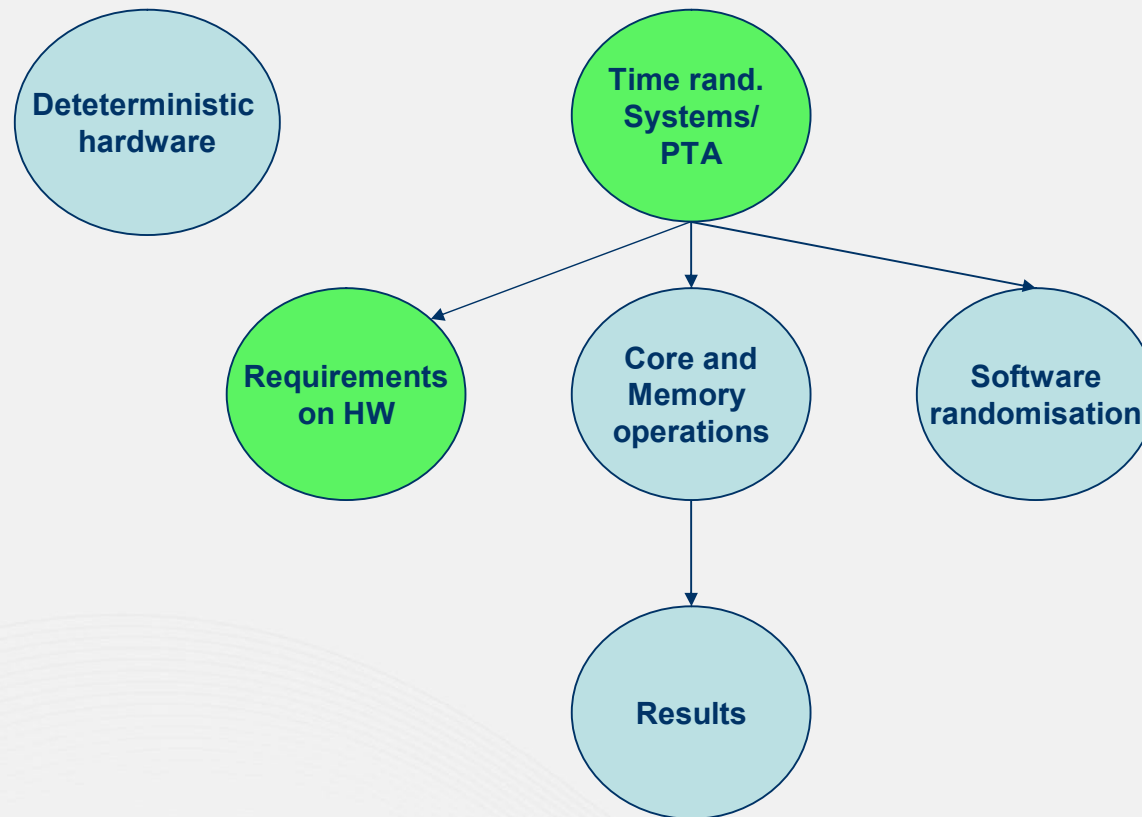
∅ _{mru}	∅ _{lru}
∅ _{mru}	∅ _{lru}
∅ _{mru}	∅ _{lru}
∅ _{mru}	∅ _{lru}
∅ _{mru}	∅ _{lru}
∅ _{mru}	∅ _{lru}
∅ _{mru}	∅ _{lru}
∅ _{mru}	∅ _{lru}

(c) 2 unknown addresses

Example (4)

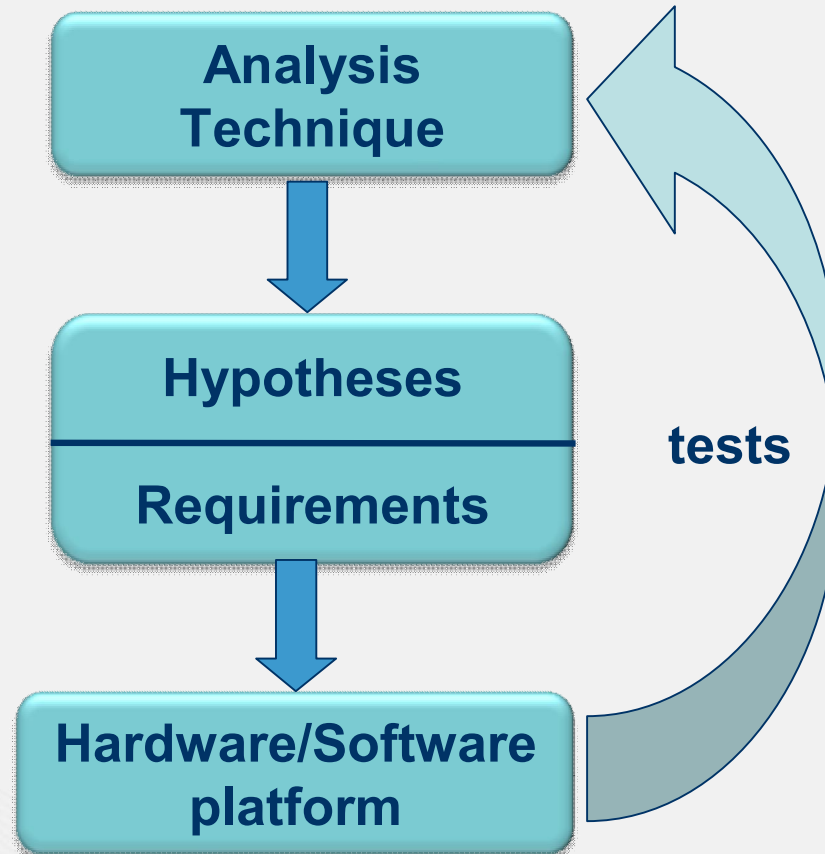
- *STA:*
 - Invalidate a complete cache way on every access!!
- *MBTA:*
 - We may not observe any conflict among vect1 and vect2 in our measurements
 - Conflicts can occur systematically in the system once deployed
- *Deterministic platforms, in the presence of unknowns*
 - Pessimistic WCET estimates, or
 - Tough to derive WCET estimates

Outline



Probabilistic Timing Analysis: Recap

- Requirements: i.i.d
 - ETP → i.i.d



Random variables

- *In a random variable, each value has an associated probability*

➤ $v = \begin{pmatrix} v_1 & v_2 & \cdots & v_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$ in which $\sum_{j=1,n} p_j = 1$



$v = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{pmatrix}$

- **Independent and identically distribution (i.i.d.)**

- Values are independent of previous occurrences
- Probabilities associated to each value remains the same

- ***The timing behaviour of each processor instruction can be modelled by a random variable***

- The Execution Time Profile (ETP) is the implementation of a random variable that describes the timing behaviour of processor instructions

$I = \begin{pmatrix} t_1 & t_2 & \cdots & t_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix} \Rightarrow \text{ETP}(I) = \{\vec{t}, \vec{p}\} = \{\{t_1, t_2, \dots, t_n\}, \{p_1, p_2, \dots, p_n\}\},$
in which $\sum_{i=1}^n p_i = 1$

Prob. Approach: Requirements on the HW

- *PTA techniques require that the events under analysis, can be modelled with i.i.d. random variables:*
 - Executions times (ET) for MBPTA
 - Instruction latencies for SPTA
- *The Existence of an ETP ensures that event under consideration (ET or latencies) have an actual probability of occurrence,*
 - This is a sufficient and necessary condition to achieve the desired probabilistic i.i.d. behaviour

Type of resources

- *Jitterless resources*
 - No response time variation
- *Jittery resources*
 - Dependence on input data
 - **Strategy** Force them to respond in its worst-case response time
 - Worst-Case Execution Mode (MERASA project)
 - Dependence on initial execution conditions
 - **Strategy** Force the initial execution conditions to be the same at analysis and deployment time
 - Dependence on execution history generated within the execution
 - **Strategy** Provide a random timing behaviour of the response time

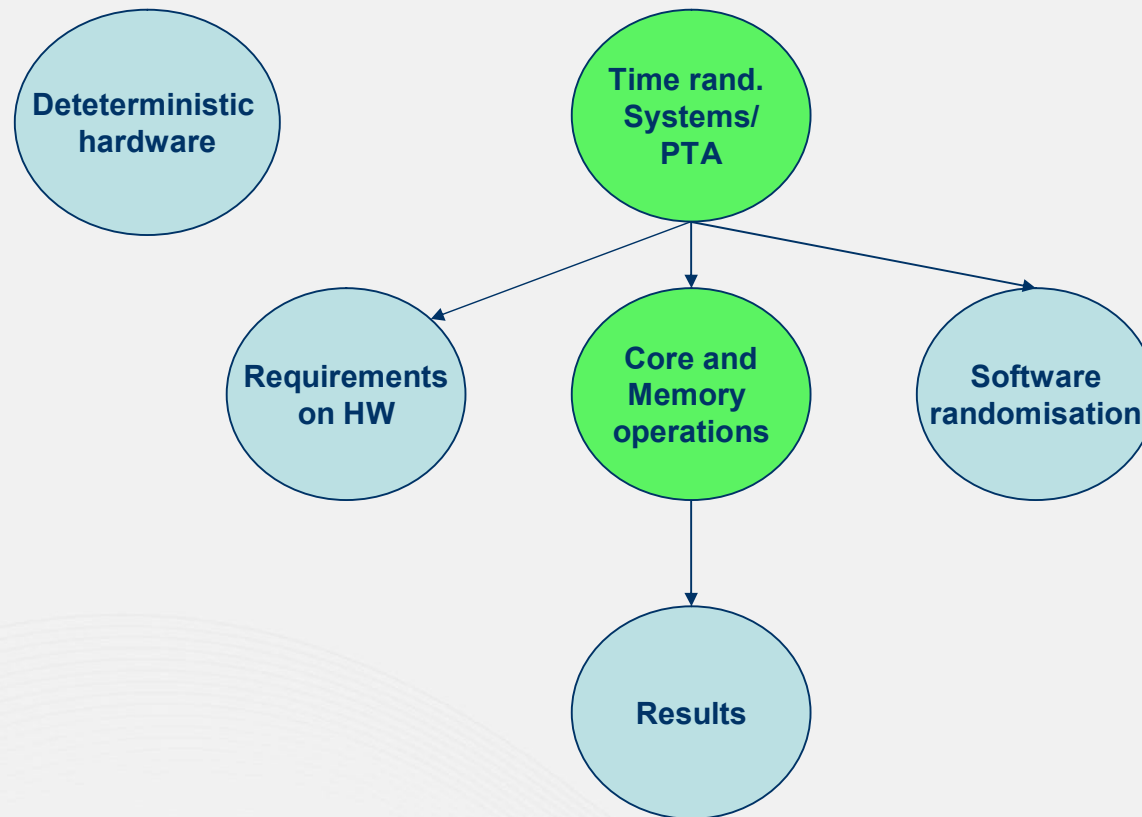
Type of resources

- ⋮ *The response time of each processor resource can be described with an ETP*
 - Jitterless
 - $ETP(R_i) = \{\vec{t}_i, \vec{p}_i\} = \{\{t\}, \{1\}\}$
 - Bounded from above
 - $ETP(R_i) = \{\vec{t}_i, \vec{p}_i\} = \{\{t_{min}, t_1, t_2, t_{max}\}, \{0, 0, 0, 1\}\}$
 - Time randomised
 - $ETP(R_i) = \{\vec{t}_i, \vec{p}_i\} = \{\{t_{min}, t_1, t_2, t_{max}\}, \{p_{min}, p_1, p_2, p_{max}\}\}$

Deterministic and time-randomised resources

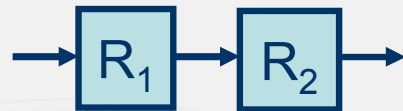
- *Deterministic and probabilistic processor resources can coexist in a PTA-friendly processor*
 - Multiple levels of random cache design, buffers, branch predictors

Outline

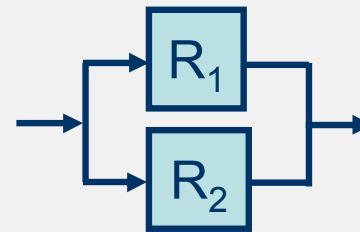


From resources to instructions

- The ETP of a processor instruction is computed by convolving the ETP of the resources it accesses
 - $ETP(I) = ETP(R_1) \otimes ETP(R_2) = \{\vec{t_1} \otimes \vec{t_2}, \vec{p_1} \otimes \vec{p_2}\}$
- Accesses to resources in sequential or parallel order
 - In sequential, latencies are added and probabilities are multiplied
 - In parallel, latencies takes the maximum and probabilities are multiplied



$$\begin{aligned}
 ETP(R_1) &= \{1,3,5\}, \{0.3,0.4,0.3\} \\
 ETP(R_2) &= \{1\}, \{1.0\} \\
 \mathbf{ETP(I)} &= \mathbf{\{2,4,6\}, \{0.3,0.4,0.3\}}
 \end{aligned}$$



$$\begin{aligned}
 ETP(R_1) &= \{1,4\}, \{0.3,0.6\} \\
 ETP(R_2) &= \{2,3\}, \{0.3,0.7\} \\
 \mathbf{ETP(I)} &= \mathbf{\{2,3,4\}, \{0.12,0.28,0.6\}}
 \end{aligned}$$

From resources to instructions

- *Two type of instructions considered:*
 - Core operations
 - Memory operations
- *Core Operations*
 - For this talk we assume an architecture similar to the NGMP
 - All instructions have fixed latency (1 cycle)
 - Full-bypasses provide back-to-back execution

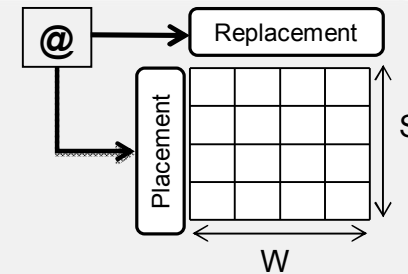
Cache operations: PROARTIS cache design

- *Remove sources of determinism*

- Placement
- Replacment

- *PROARTIS cache designs*

- Random placement and random replacement
- Can conflicts occur? Yes, but with a given probability
- Fulfils i.i.d. as required by MBPTA
 - MBPTA needs execution time to behave as a random variable
 - This occurs if execution times are independent and identically distributed
- Much less information required
- Robust in front of changes
- Applied to IL1, DL1, L2, TLBs, etc.

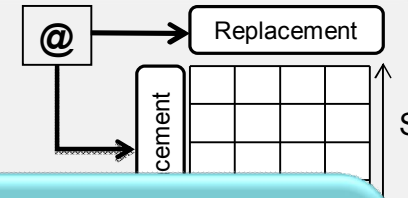


Cache operations: PROARTIS cache design

- *Remove sources of determinism*

- Placement

- Replacement



- In a random DM cache:

- In a hard-real time system you assume the worst
 - For any @a and @b, they can be mapped onto the same set → It is close not to have a cache

- **WRONG!!!**

- @a can evict @b with a known probability
 - Cache retains a probabilistic state
 - We benefit from that probabilistic state

Random Replacement: Evict on Access (FA)

- *Random replacement*



- A new line is randomly evicted on every access
- **Independence across evictions**
- Prob. of a cache line to be evicted $P_{miss}=1/W$; $P_{hit}=(W-1)/W$
- P_{hit} depends on how many accesses are performed between consecutive accesses to a given address A.
 - Reuse distance
 - History dependent!!!!
 - PTA analysis work on a per-path basis
 - One run → one path.
 - Combine paths

$$P(hit) = \left(\frac{N-1}{N} \right)^K$$

$$P(hit) = \begin{cases} \left(\frac{N-(K-1)-1}{N-(K-1)} \right)^K & \text{if } K < N \\ 0 & \text{if } K \geq N \end{cases}$$

Random Replacement: Evict on miss (FA)

- *Random replacement*



- A new line is randomly evicted on every miss
- **Independence across evictions**
- Prob. of a cache line to be evicted $P_{miss}=1/W$; $P_{hit}=(W-1)/W$
- P_{hit} depends on how many accesses are performed between consecutive accesses to a given address A and their hit prob
 - The higher the miss probability of $\{B_i\}$ the lower the survivability of A_j
- Example:
 - Sequence: $A_i B_{i+1} \dots B_{j-1} A_j$

$$P_{hit_{A_j}} = \left(\frac{W-1}{W} \right)^{\sum_{k=i+1}^{j-1} P_{miss_{B_k}}}$$

Random Replacement: Evict on miss (FA)

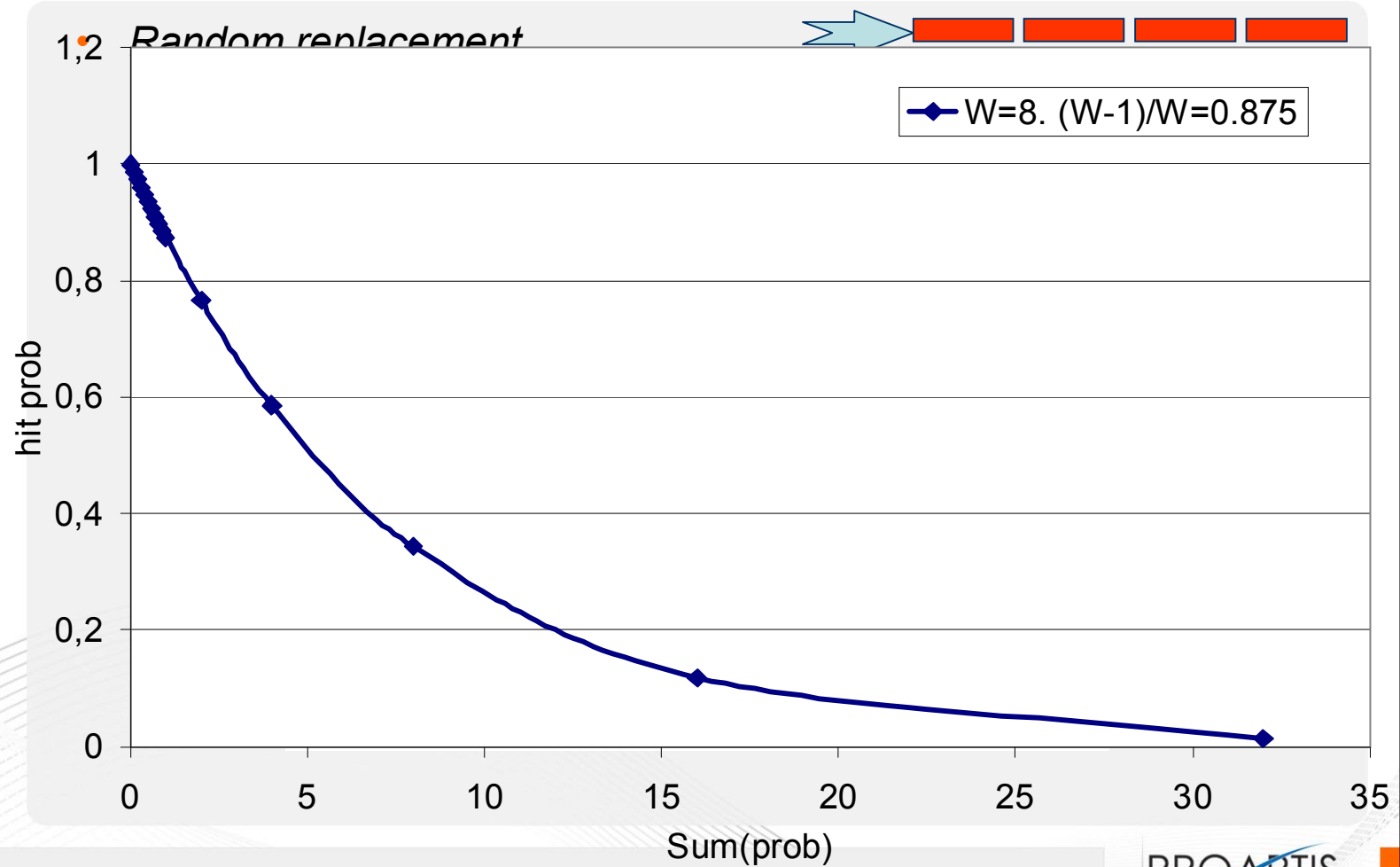
- *Random replacement*



- A new line is randomly evicted on every miss
- **Independence across evictions**
- Prob. of a cache line to be evicted $P_{miss}=1/W$; $P_{hit}=(W-1)/W$
- P_{hit} depends on how many accesses are performed between consecutive accesses to a given address A and their hit prob
 - The higher the miss probability of $\{B_i\}$ the lower the survivability of A_j
- Example:
 - Sequence: $A_i B_{i+1} \dots B_{j-1} A_j$

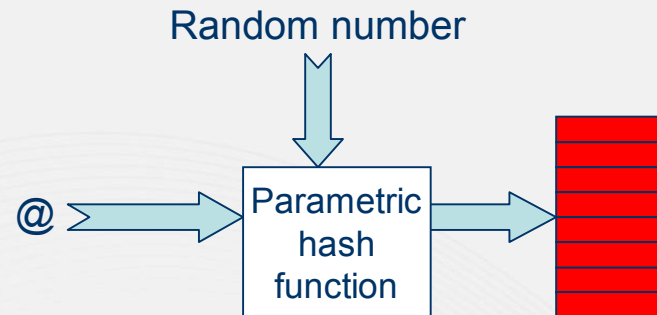
$$P_{hit_{A_j}} = \left(\frac{W-1}{W} \right)^{\sum_{k=i+1}^{j-1} P_{miss_{B_k}}}$$

Random Replacement: Evict on miss (FA)



Random Placement (DM)

- *If placement policy assign two addresses to the same set → the will collide systematically.*
- *Random placement:*
 - On every run a random number pass to a parametric hash function
 - Conflicts during a run are deterministic,
 - but random across runs (as needed)
 - Phit(A) depends on how many accesses are between consecutive accesses to A and whether those may reuse data
 - Those parameters are independent of the absolute address of data!!



$A, B_1, B_2, \dots, B_q, A$

$$P_{hit_A} = \frac{(S-1)^q \cdot S}{S^{q+1}} = \left(\frac{S-1}{S}\right)^q$$

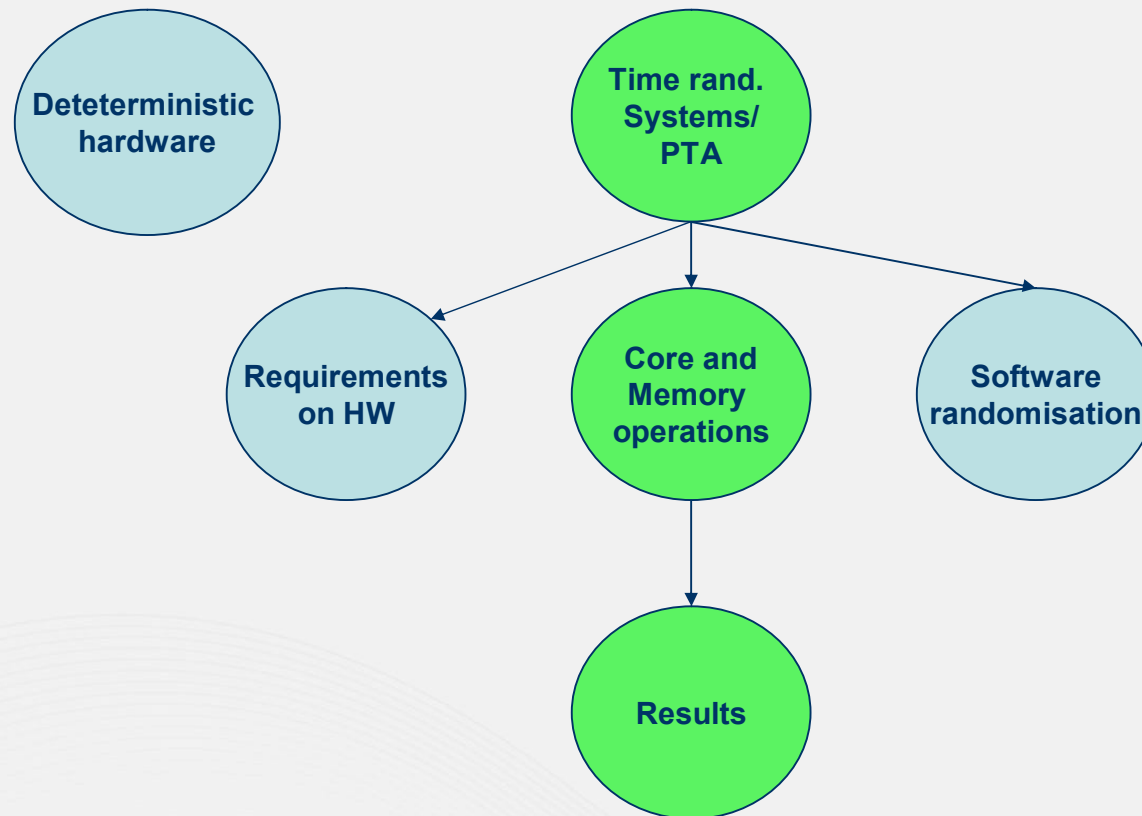
Leonidas Kosmidis, Jaume Abella, Eduardo Quiñones, Francisco Cazorla
"A Cache Design for Probabilistic Real-Time Systems". DATE 2013

Random Placement + Replacement

- *PROARTIS* cache designs
 - Random placement and replacement provide **true probabilistic behaviour**
 - Fulfils i.i.d. as required by MBPTA
 - Much less information required
 - Robust in front of changes
 - Applied to IL1, DL1, L2, TLBs, etc.

$$P_{miss_A}(SA[S, W]) = P_{miss_A}(DM[S]) \cdot P_{miss_A}(FA[W])$$

Outline



Results of the iid tests (MBPTA)

- *~ hundreds of runs*
- *Hypothesis testing*
- *Identically distribution:*
 - We apply the two-sample Kolmogorov-Smirnov test
 - H_0 is that the both samples are identically distributed
 - H_1 is that the samples are not identically distributed.
 - Significance level $\alpha = 0.05$
 - Outcome of the test: p-value.
 - $p\text{-value} > \alpha$ the null hypothesis cannot be rejected,
 - Both samples are identically distributed
- *Independence:*
 - Runs test for randomness
 - $\alpha = 0.05$: If the outcome of the test (Z-value) $> 1.96 \rightarrow$ indicates non-randomness

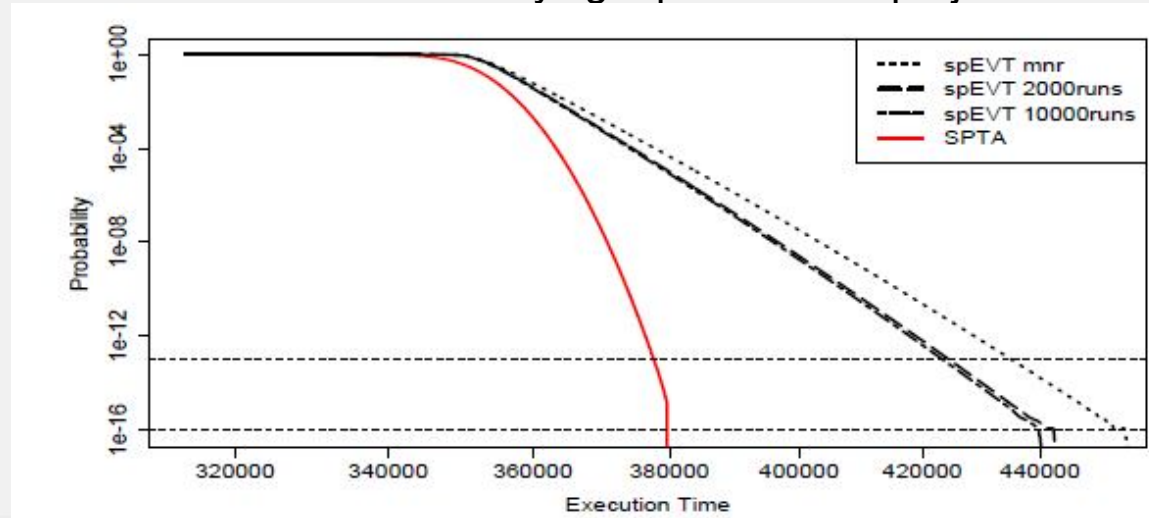
Results of the iid tests

- *Core architecture with fixed latency*
- *Various Cache setup with different no. of Sets and Ways*

Benchmarks	1w-256s DM	8w-32s SA	32w-8s SA	256w-1s FA
a2time	0.63 / 0.43	0.90 / 0.64	0.88 / 0.75	0.53 / 0.57
aifft	0.07 / 0.98	0.01 / 0.93	0.74 / 0.92	0.59 / 0.74
aifirf	0.39 / 0.83	0.27 / 0.84	0.76 / 0.28	0.21 / 0.36
aiifft	0.23 / 0.81	0.11 / 0.70	0.19 / 0.53	0.05 / 0.34
cacheb	0.53 / 0.48	0.51 / 0.40	0.27 / 0.97	1.20 / 0.36
canrdr	0.17 / 0.53	0.21 / 0.39	1.27 / 0.12	0.23 / 0.29
iirflt	1.27 / 0.84	0.11 / 0.80	0.05 / 0.49	0.71 / 0.75
puwmod	0.17 / 0.47	0.37 / 0.89	0.41 / 0.96	0.51 / 0.92
rspeed	0.33 / 0.89	0.33 / 0.27	0.25 / 0.24	1.24 / 0.60
tblook	0.51 / 0.35	0.47 / 0.93	0.67 / 0.91	0.03 / 0.54
ttsprk	0.82 / 0.13	0.63 / 0.73	0.19 / 0.92	1.12 / 0.80

MBPTA: results on EEMBC

- *MBPTA projections are tight to those provided by SPTA*
 - Recall SPTA is a safe and very tight probabilistic projection

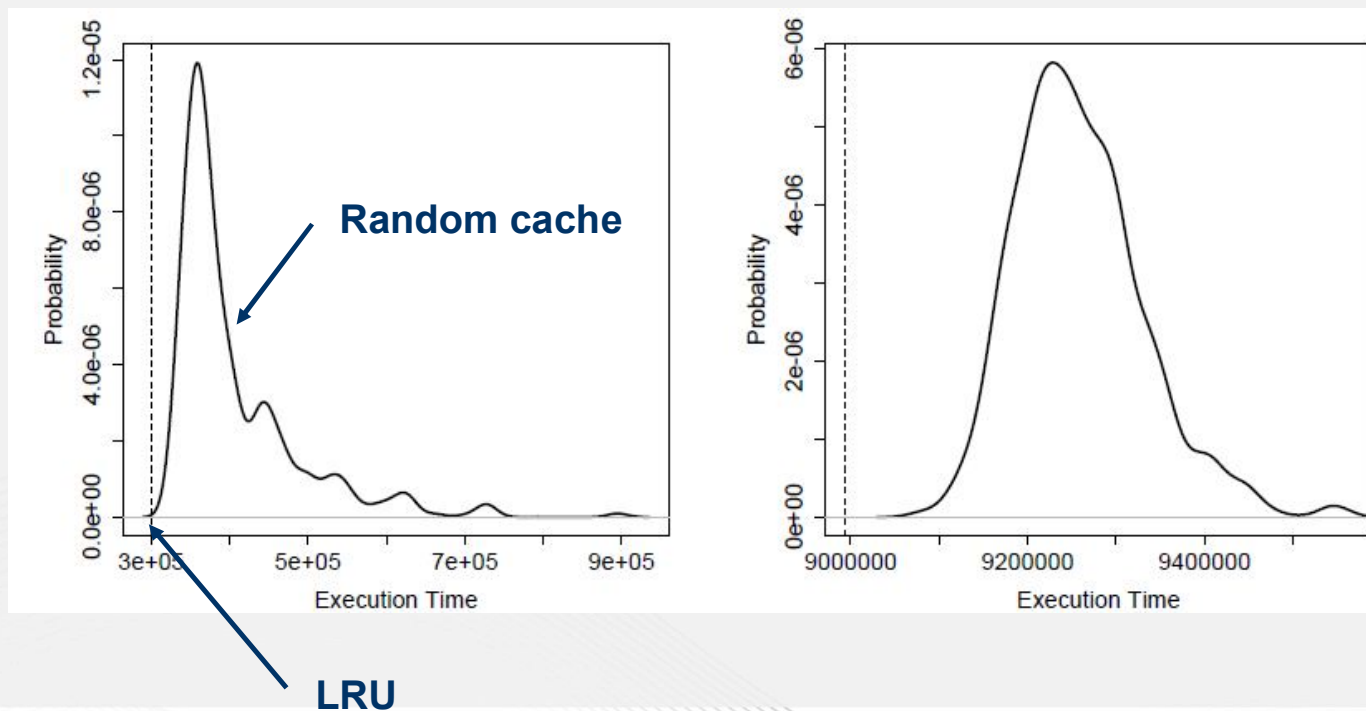


- A

probability	AI	AT	CA	CN	PU	RS	TB	TT
10^{-13}	9%	5%	6%	5%	5%	2%	2%	6%
10^{-16}	15%	7%	8%	7%	7%	3%	3%	9%

Average performance

- *Average performance of the deterministic architecture around 20% better than with randomised architecture*



Average performance

- *LRU+modulo vs. RR+RP cache*

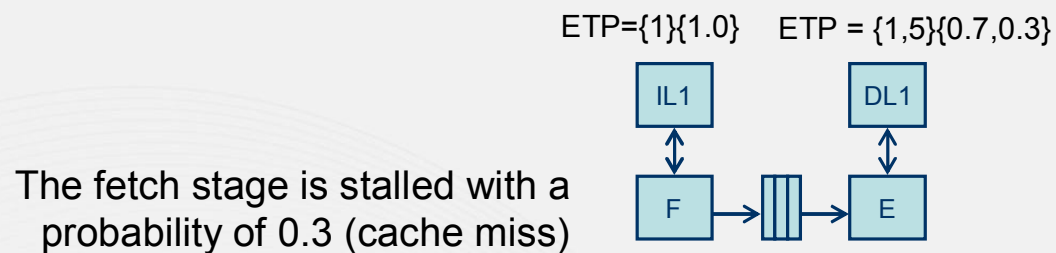
IPC OF RP+RR AND MODULO+LRU CACHES IN OUR ARCHITECTURE

	1w-256s DM	8w-32s SA	32w-8s SA	256w-1s FA
RP+RR	0.234	0.585	0.615	0.627
LRU+modulo	0.613	0.665	0.687	0.698

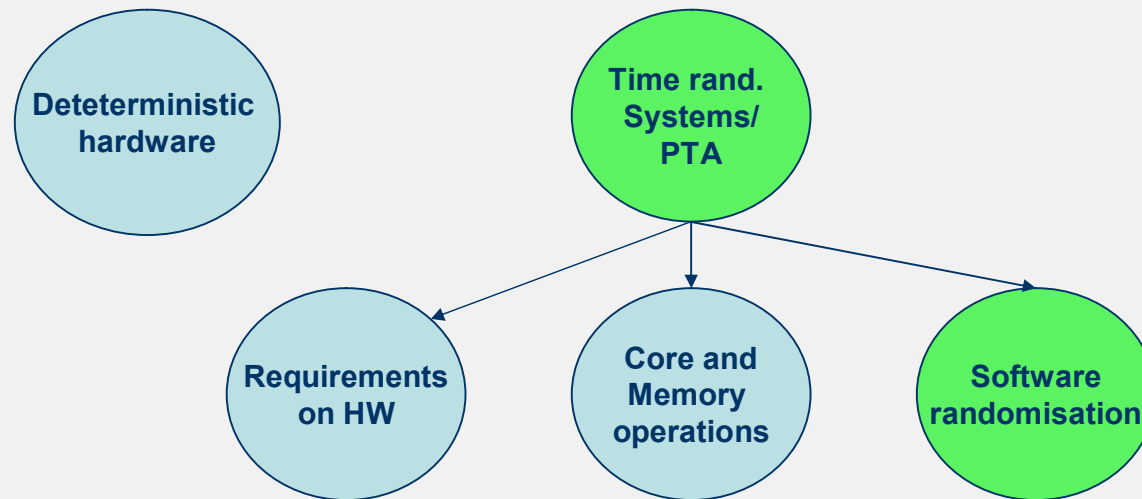
- *High performance degradation for DM caches*
- *Smaller performance degradation for SA caches*
 - 11%

Core operations: buffers

- *Buffers may get full and create pipeline stalls that propagate backwards*
- *If all processor resources generate probabilistic jitter:*
- *For a given sequence of events*
 - The resulting ET have a given probability
 - Buffers do not change this probability but increase the ET



Outline



Software Random Placement

- *Can we achieve something similar by SW? Yes*
 - Random replacement already exists in real processors, but not random placement
 - Place randomly objects in memory
 - They will be mapped into random cache sets, as for HW random placement
 - Limitations
 - Placing randomly things requires indirections (cost in performance)
 - Small objects require many indirections, thus performance cost is high (and WCET estimates are higher)
 - Large objects offer a lower level of randomisation, so MBPTA delivers worse WCET estimates

Leonidas Kosmidis, Charlie Curtsinger, Eduardo Quinones, Jaume Abella, Emery Berger, Francisco J. Cazorla
"Probabilistic Timing Analysis on Conventional Cache Designs". DATE 2013

Conclusions

- *Deterministic and probabilistic processor resources can coexist in a PTA-friendly processor*
- *Design “90%” compatible with Leon3/4 Designs*
 - **Except they do not implement a random placement policy**

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 n° 249100.

The logo for PROARTIS features the word "PROARTIS" in a bold, black, sans-serif font. A blue curved line starts from the left, passing through the 'O' and 'A', and ending to the right of the 'S'. A thin vertical line is positioned to the left of the 'P', and a thin horizontal line is positioned below the 'S'.

PROARTIS

Hardware Architectural Solutions

Leonidas Kosmidis, Jaume Abella, Eduardo Quiñones, Francisco J. Cazorla

Barcelona Supercomputing Center (BSC-CNS)

22 January 2013



Berlin

www.proartis-project.eu