RAVEN: Reinforcement Learning for Generating Verifiable Run-time Requirement Enforcers for MPSoCs

*Khalil Esper, *Jan Spieck, Pierre-Louis Sixdenier, Stefan Wildermann, Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

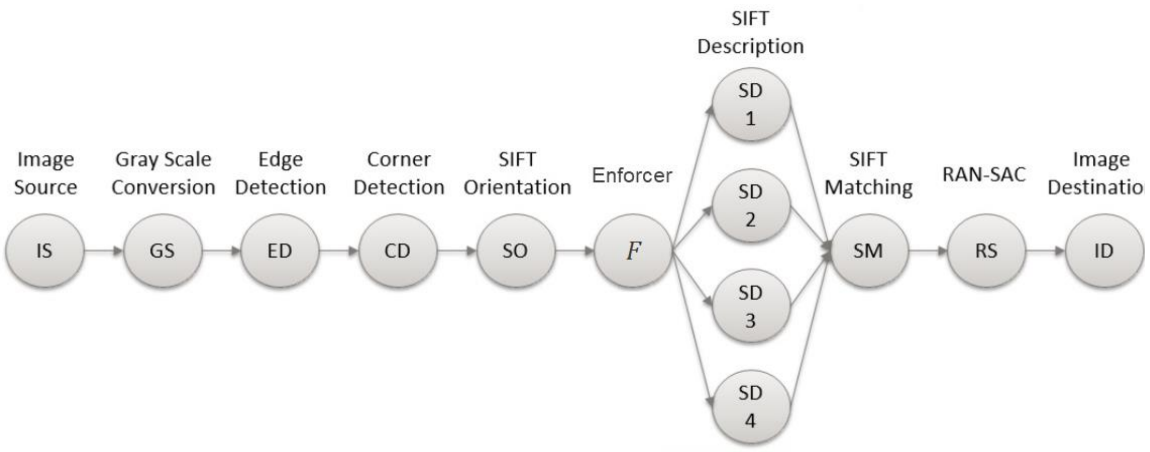
Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2023)

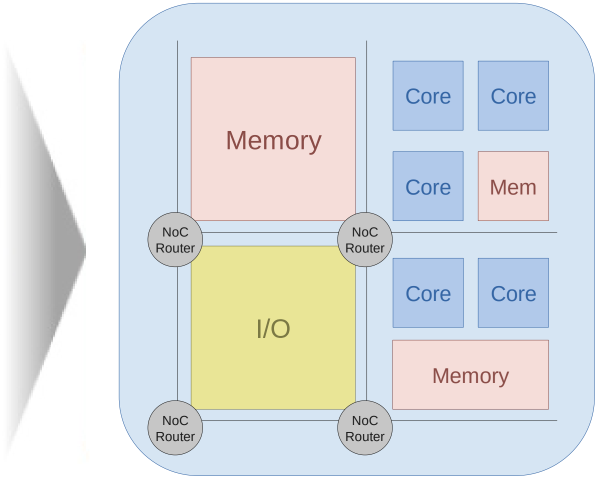
Outline

Introduction

Motivation

- Many core architecture
- Application via actor graph
- A set of non-functional requirements φ on execution properties o
  - E.g., latency, energy or throughput requirements
  - specified using intervals

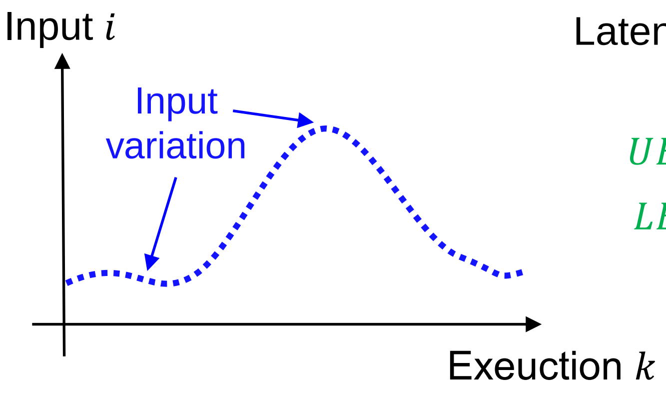


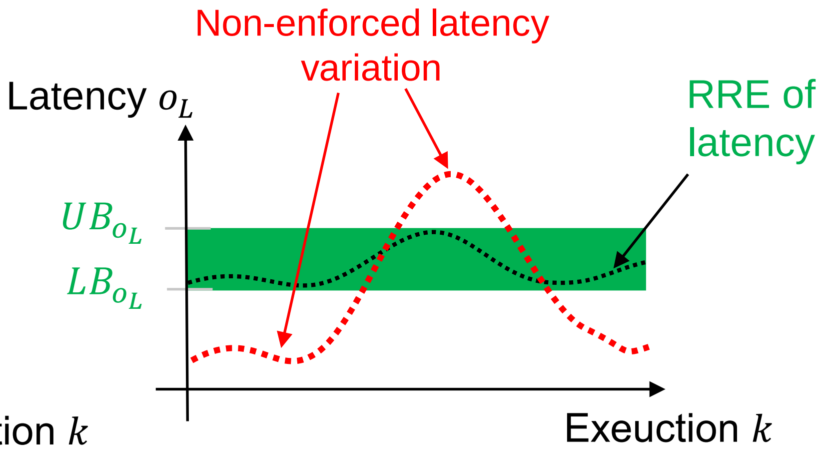


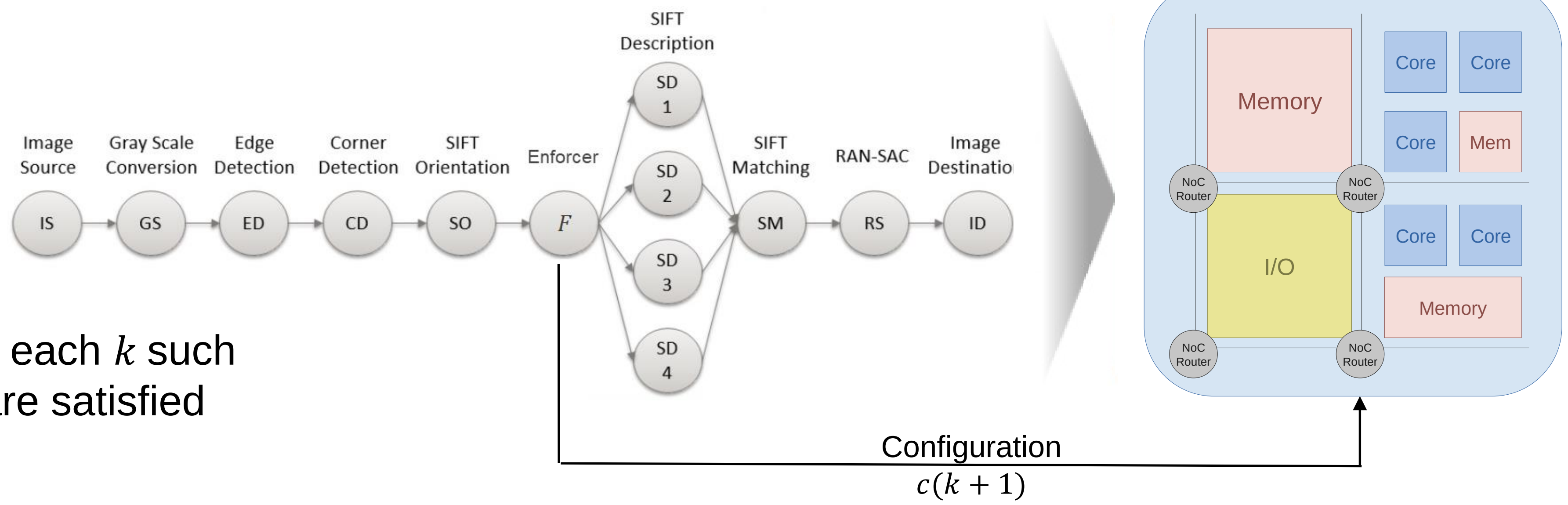
Introduction

Runtime requirement enforcement (RRE)

- Uncertainty in the environment
  - E.g., variation in input $i(k) \in I$
- Run-time Requirement Enforcement (RRE)
  - via configurations $c \in C$:
    - n: number of cores
    - m: DVFS level
    - Goal: choose $< n, m >$ at each k such that the requirements φ are satisfied



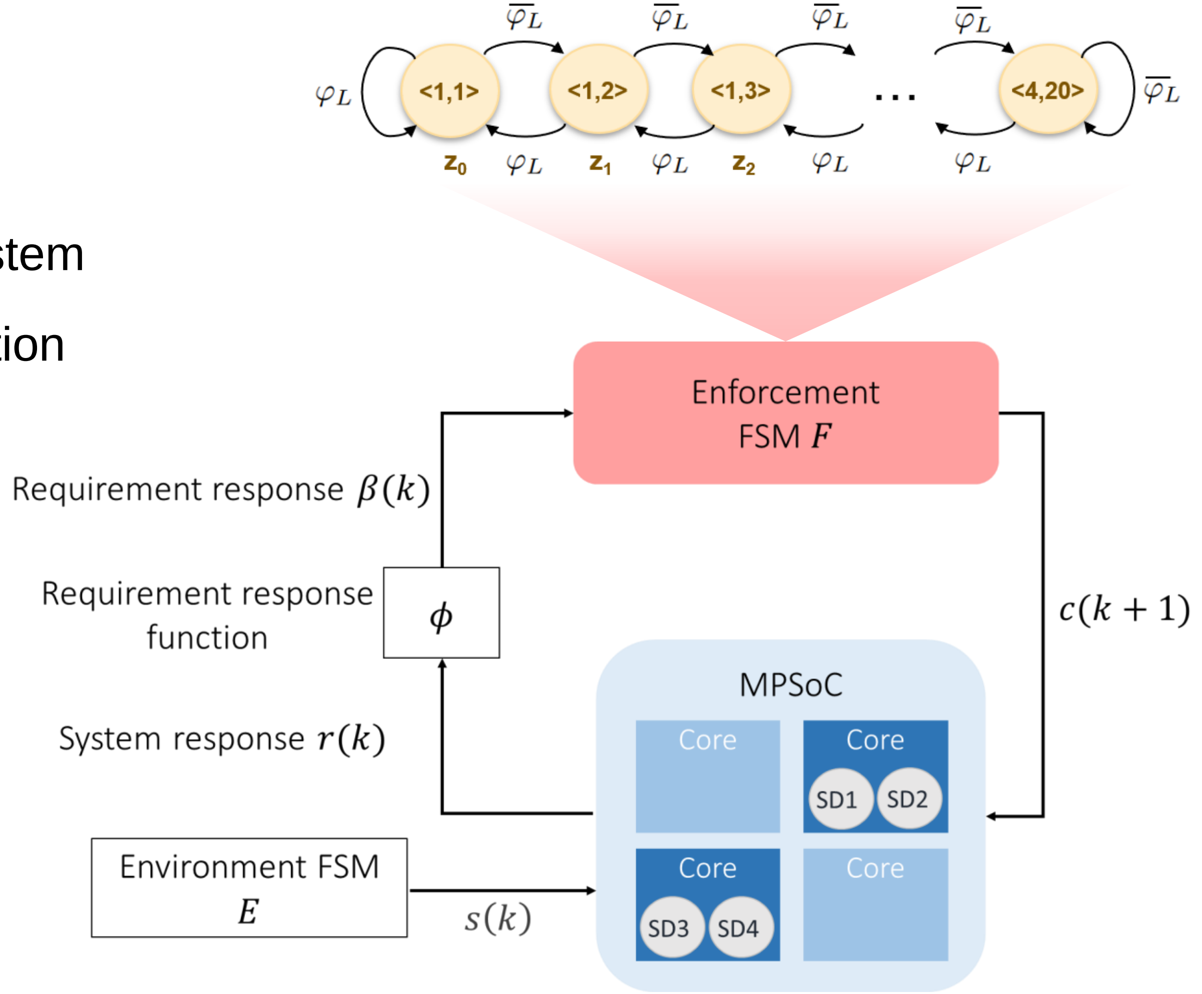




Introduction

Runtime requirement enforcement (RRE) using enforcement FSMs

- Enforcement FSM F determines a configuration $c(k+1)$ for the next execution $k+1$
  - Based on the kth requirement response $\beta(k)$ of the system
- Environment FSM describes the environment input variation
- Compare enforcer strategies based on verification goals VG:
  - defined over requirements φ
    - Strict: e.g., $AG(\varphi)$: φ should always hold
    - Loose: e.g., $S_{=?}[\neg\varphi]$: the steady-state probability of violating φ



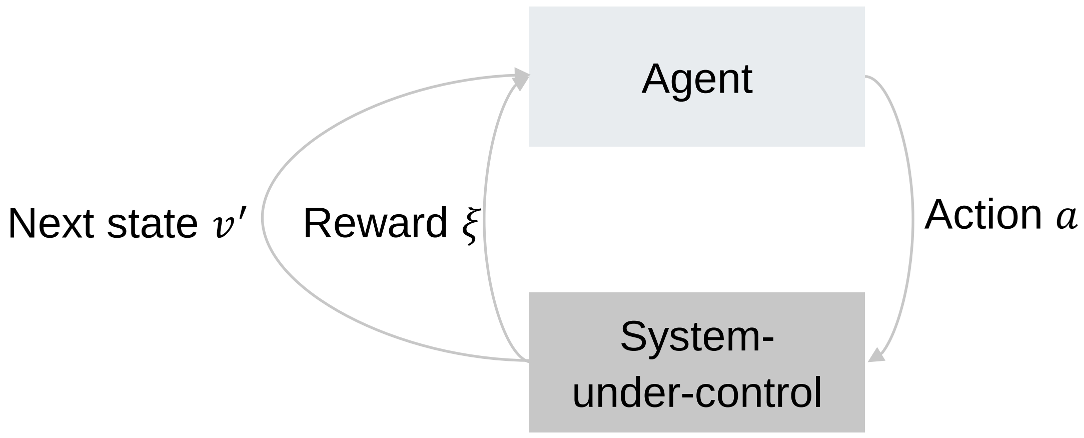
Introduction

Reinforcement Learning

Reinforcement Learning (RL) is a Machine Learning paradigm

- Goal: maximize a cumulative reward by learning actions

- The System-under-control resides in a state $v \in \Upsilon$
- Based on which the agent then selects an action $a \in A$ (according to its internal policy π)
- Transitions to successor state: $v' \in \Upsilon$
- Receives a reward signal $\xi : \Upsilon \times A \rightarrow \mathbb{R}$



Introduction

Q-Learning

How good is a state-action pair?

– An action-value function $Q^{\pi}: \Upsilon \times A \rightarrow \mathbb{R}$

– Predicts cumulated reward on the long run

Q-table	
State-action	Value
v_0, a_0	1
v_0, a_1	2
v_1, a_0	-1
…	…

Q-Learning:

– Learns action-value function, i.e., Q-function

– Until terminal state or maximum iterations

– Q-table stores all the values of Q-function

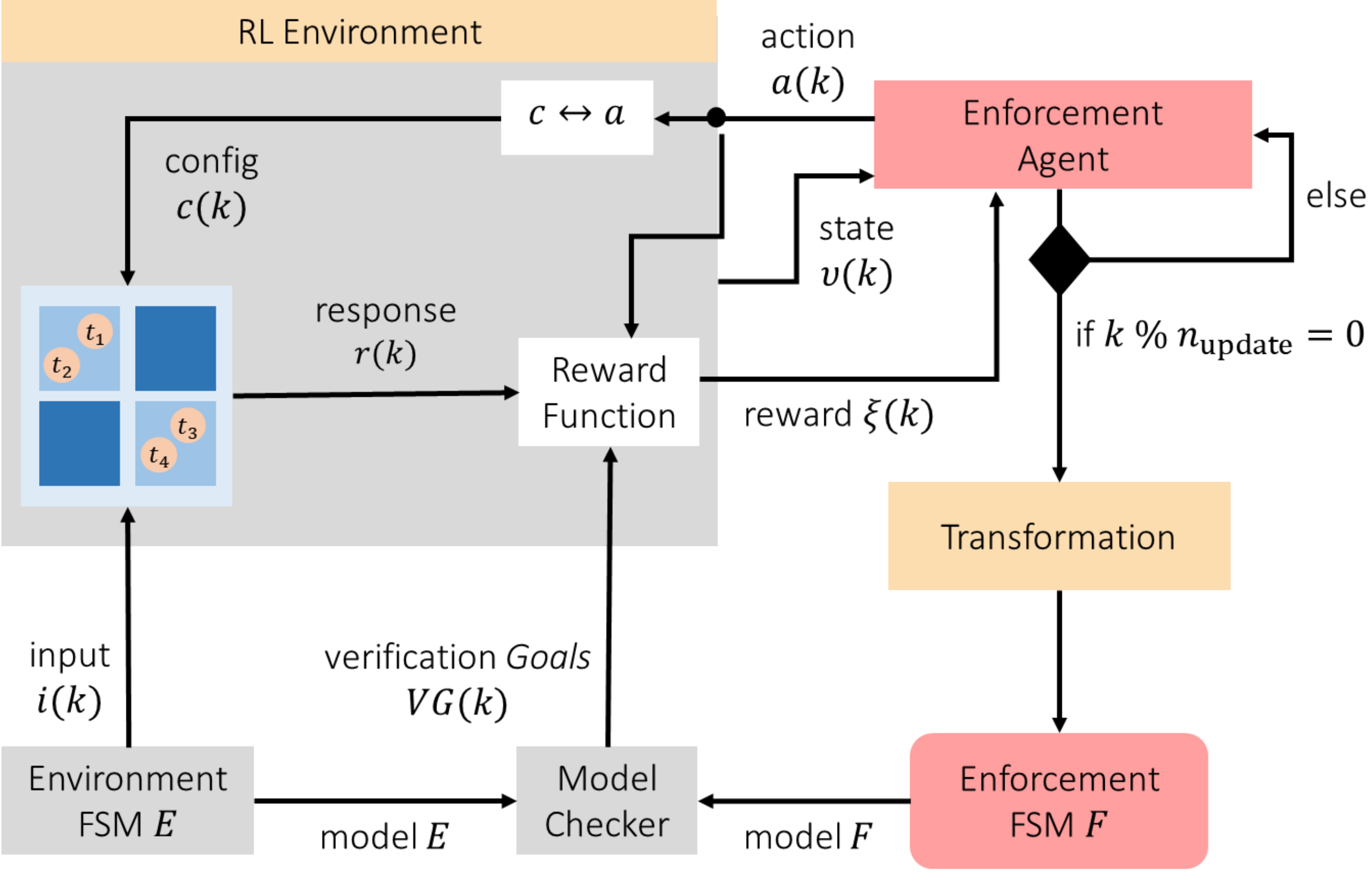
Choose an action → Calculate reward → Update Q-table

Reinforcement Learning for Generating Verifiable RRE

Training phase

Goal: learn an enforcement strategy that optimizes a given set of verification goals VG

- An action $a(k)$: a configuration $c(k) \in C$
- A state $\upsilon \in \Upsilon = B \times C$: a configuration $c \in C$ and a requirement response $\beta \in B$



Reinforcement Learning for Generating Verifiable RRE

Training phase

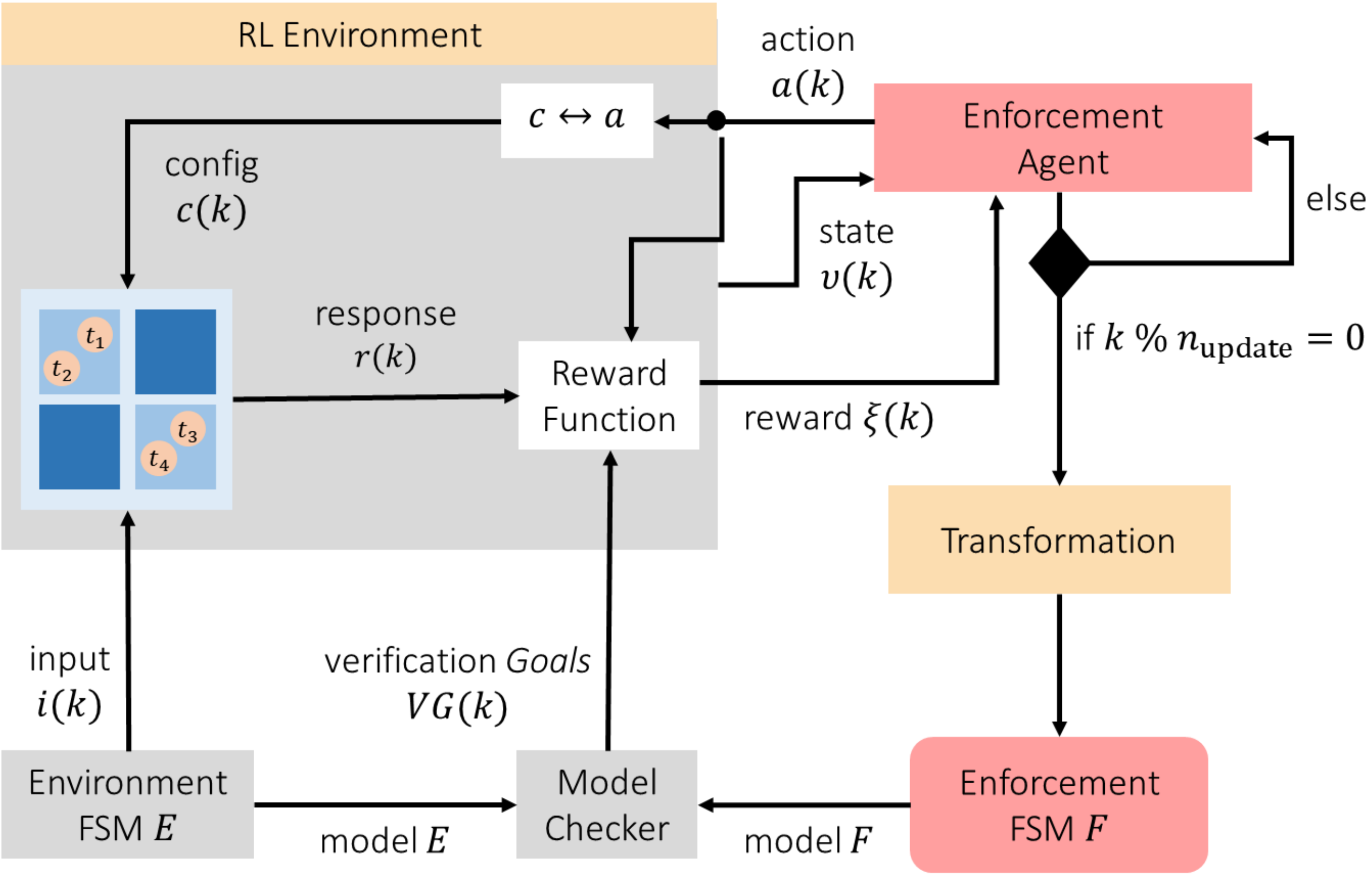
A reward function $\xi_\eta\big(a(k)\big) = \eta \,.\, \xi_{\text{sur}}(k) + (1-\eta)\,.\, \xi_{\text{ver}}(k)$

– Feedback about the requirements satisfaction

– A weighted sum of:

1. A verification reward $\xi_{\text{ver}}(k)$: from the model checker after transforming the enforcement agent into an enforcement FSM every n_{update} iterations

2. And a surrogate reward $\xi_{\text{sur}}(k)$: estimation of verification goals at each k based on the processed input history



Reinforcement Learning for Generating Verifiable RRE

Transformation

How to transform the enforcement agent (i.e., the Q-table) into an enforcer FSM?

- One unique enforcement state per configuration $\zeta: C \leftrightarrow Z$
- Best action per state $\varrho: \Upsilon \rightarrow A$ (for Q-learning: $\varrho(v) = \mathrm{argmax}_{a \in A}\, Q(v, a)$)

<u>Example:</u>

Two configurations $C = \{c_0, c_1\}$ and one verification goal $VG_L \coloneqq S_{=?}[\varphi_L]$ based on a latency requirement φ_L

Q-Table

States Υ	Q-Values $Q(v,a)$	
$v = (\beta, c)$	$a_0 = c_0$	$a_1 = c_1$
$v_0 = (\overline{\varphi_L}, c_0)$	0.71	0.34
$v_1 = (\overline{\varphi_L}, c_1)$	0.56	0.21
$v_2 = (\varphi_L, c_0)$	0.62	0.99
$v_3 = (\varphi_L, c_1)$	0.29	0.35

$\varrho(v)$ $\Rightarrow$

Transformation

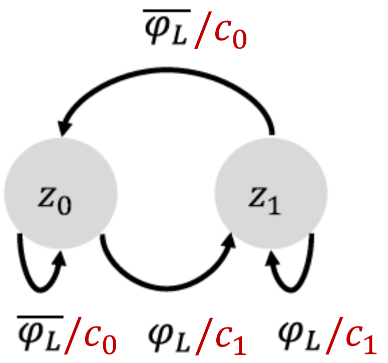
States Υ	Best action
v	$\varrho(v)$
$(\overline{\varphi_L}, c_0)$	a_0
$(\overline{\varphi_L}, c_1)$	a_0
(φ_L, c_0)	a_1
(φ_L, c_1)	a_1

$\zeta(c)$ $\Rightarrow$

Trans. Relation δ
$(\beta, \zeta(c), \zeta(a))$
$(\overline{\varphi_L}, z_0, z_0)$
$(\overline{\varphi_L}, z_1, z_0)$
(φ_L, z_0, z_1)
(φ_L, z_1, z_1)

$\Rightarrow$

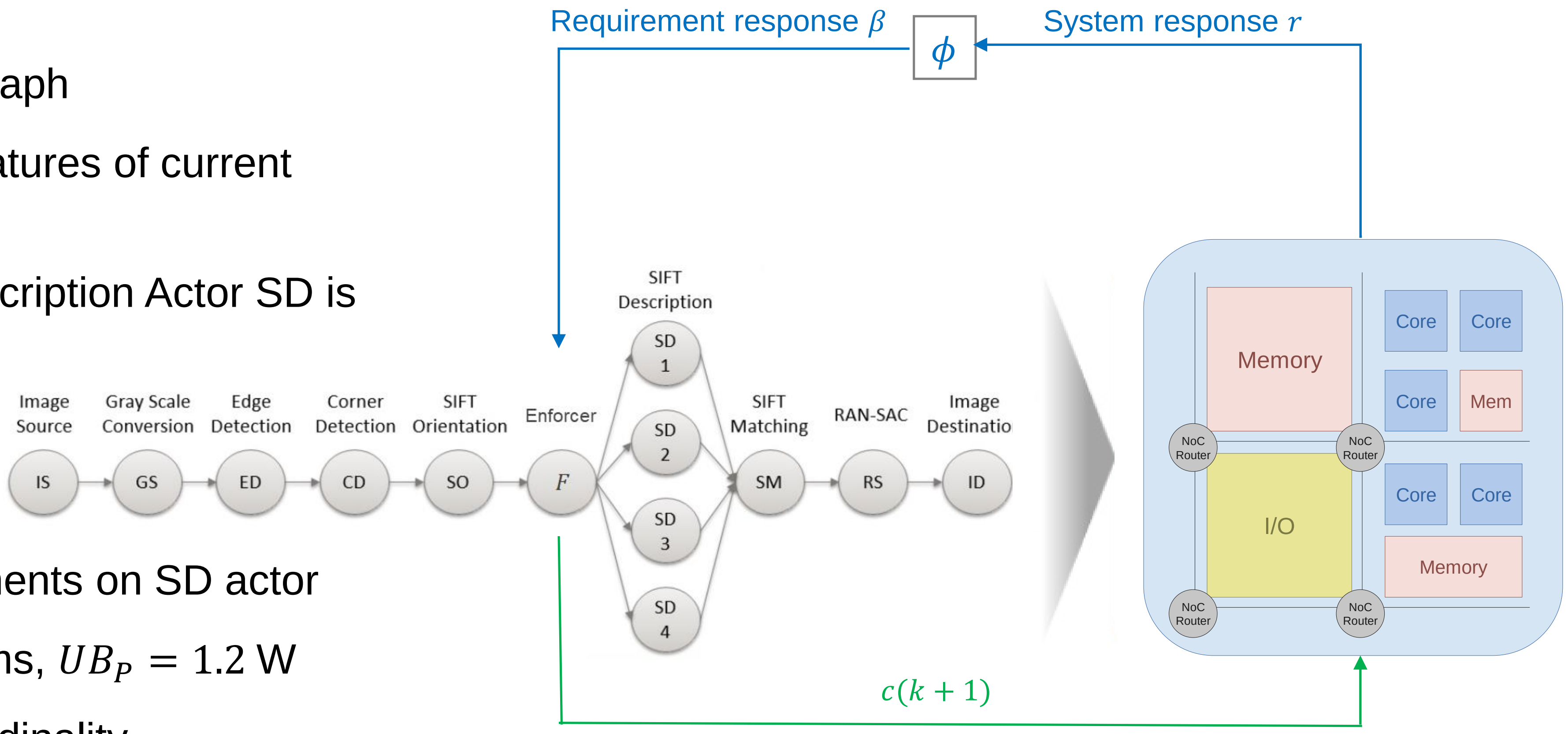
Enforcer FSM



Verification Results

Use case

- Object detection application
  - SIFT algorithm via actor graph
  - Input i is the number of features of current image $i(k)$
  - Latency $o_L(k)$ of SIFT Description Actor SD is input-dependent



- Latency and power requirements on SD actor
  - Upper bounds: $UB_L = 40$ ms, $UB_P = 1.2$ W
- Configuration space of a cardinality $|C| = |n| \cdot |m| = 4 \cdot 20 = 80$

Verification Results

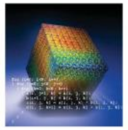
Verification results for Race-To-Idle (RTI), F_1 (1-step enforcer FSM), F_2 (8-step enforcer FSM), and $F_{\mathrm{rl}_0}, F_{\mathrm{rl}_1}, F_{\mathrm{rl}_2}$ (synthesized RL-based enforcer FSMs using our approach) based on a latency upper bound (deadline) $UB_L = 40$ ms, and a power upper bound $UB_P = 1.2$ W

Loose enforcement								
Requirement φ	**Latency φ_L**				**Power φ_P**			
VG	$P_{=?}[G^{\leq 3}\neg\varphi_L]$				$P_{=?}[G^{\leq 3}\neg\varphi_P]$			
Enforcer	RTI	F_1	F_2	F_{rl_0}	RTI	F_1	F_2	F_{rl_0}
Verification result	0	0.427	0.041	0	1	0.256	0.389	0
VG	$S_{=?}[\neg\varphi_L]$				$S_{=?}[\neg\varphi_P]$			
Enforcer	RTI	F_1	F_2	F_{rl_1}	RTI	F_1	F_2	F_{rl_1}
Verification result	0	0.5	0.121	0.173	1	0.445	0.591	0.435

Strict enforcement				
Requirement φ	**Latency φ_L**			
VG	$AG(\varphi_L)$			
Enforcer	RTI	F_1	F_2	F_{rl_2}
Verification result	true	false	false	true

Conclusion

- We presented a technique using RL for automatically generating verifiable feedback-based RRE enforcers
- First, the enforcement agent learns an enforcement strategy based on a representative input sequence at design time
- Then, the learned enforcement strategy is transformed into a verifiable enforcement FSM that can handle unseen input data at run-time
- We apply the approach to generate controllers that increase the probability of satisfying a given set of verification goals compared to related work, as can be verified by model checkers



CRC/Transregio 89
Invasive Computing

www.invasic.de

Thanks!