Towards Scalable Verification of Deep Reinforcement Learning

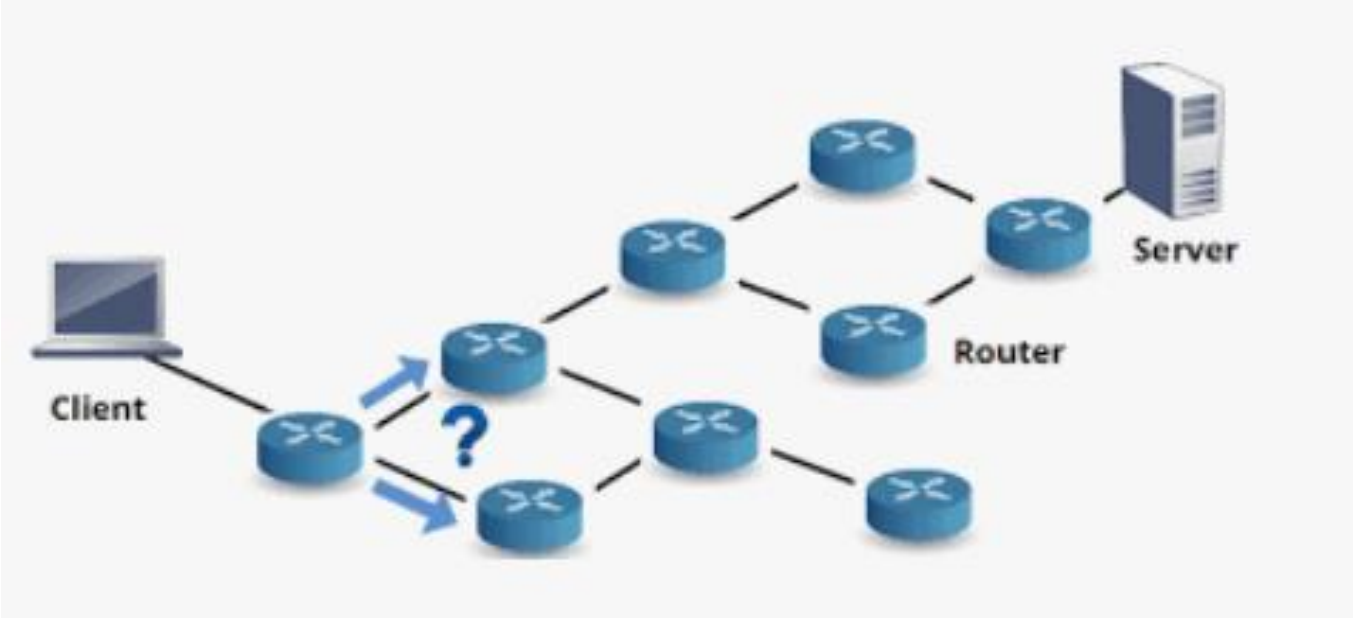
October 2021

Guy Amir Michael Schapira Guy Katz

THE HEBREW UNIVERSITY OF JERUSALEM

Traditionally

Computer and networked systems are handcrafted by domain-specific experts



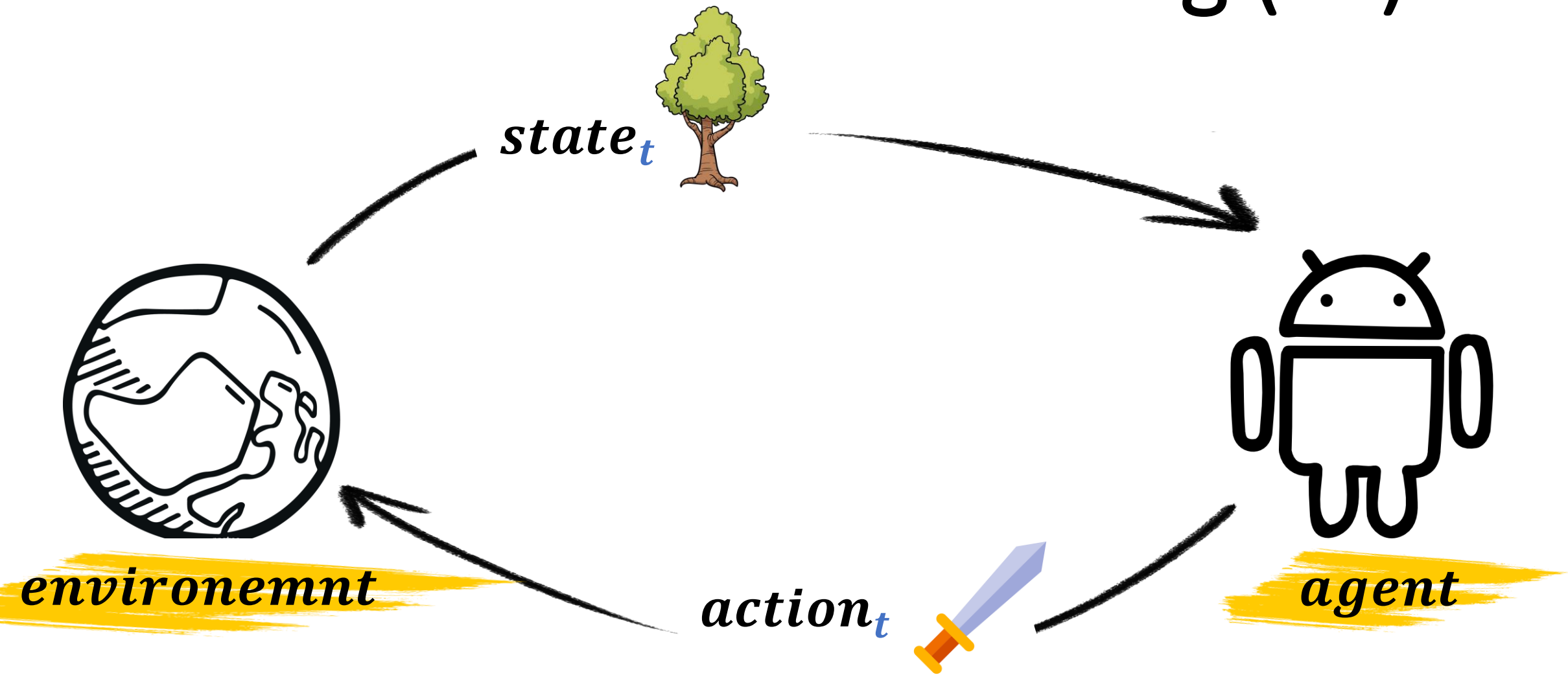
An Emerging Alternative

Deep Reinforcement Learning (DRL) solutions

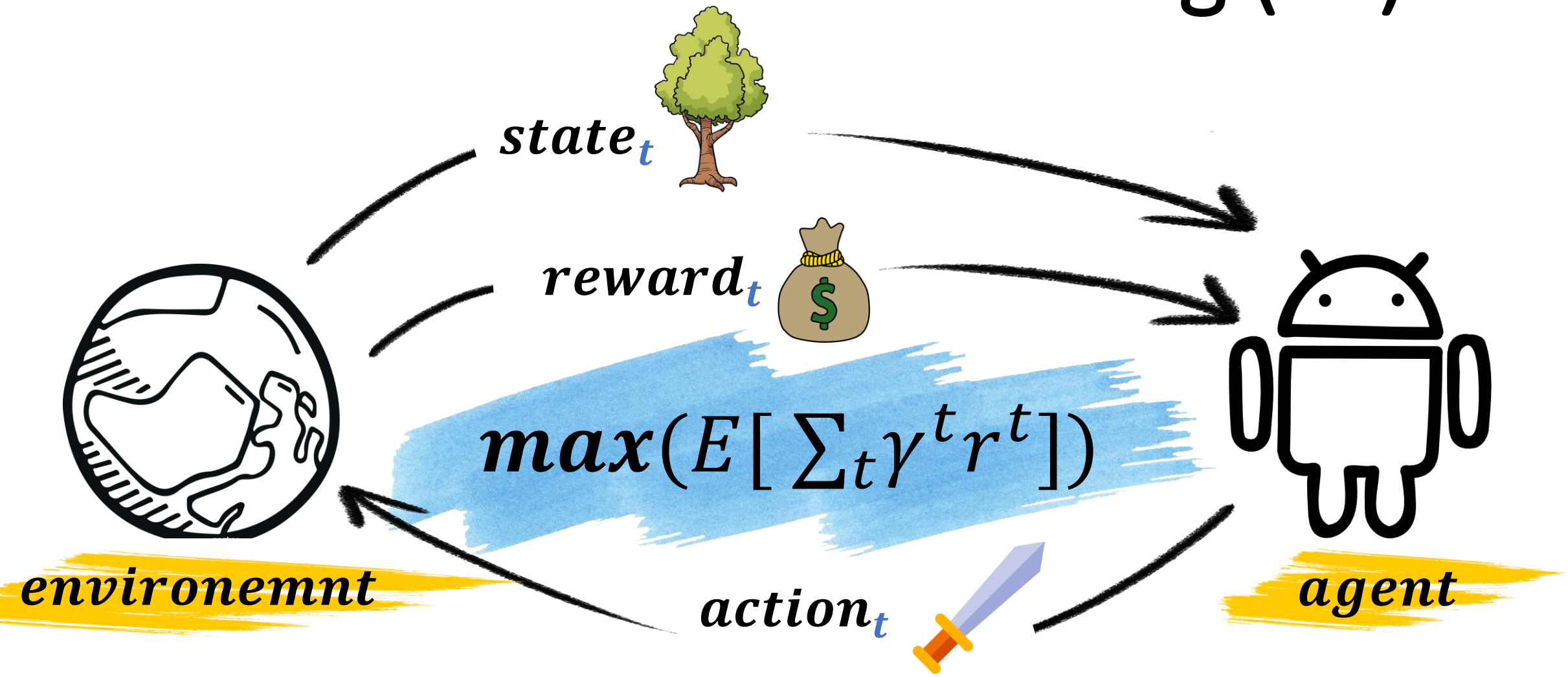
Outperform the state-of-the-art in various contexts

System	Application Domain
Aurora [29]	congestion control
NeuroCuts [40]	packet classification
[51]	SQL optimization
NEO [49]	SQL optimization
DeepRM [44]	resource allocation
[72]	resource allocation
[42]	resource & power management
[36]	compiler phase ordering
[52]	device placement
Placeto [2]	device placement
Decima [48]	spark cluster job scheduling
Pensieve [46]	adaptive video streaming
AuTO [11]	traffic optimizations

Reinforcement Learning (RL)

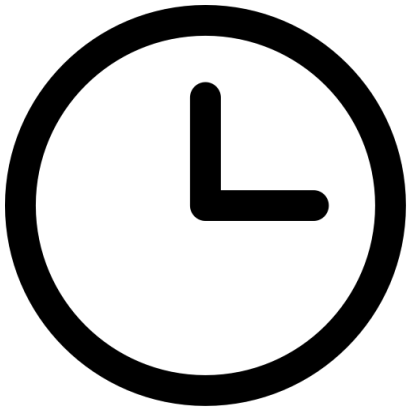


Reinforcement Learning (RL)

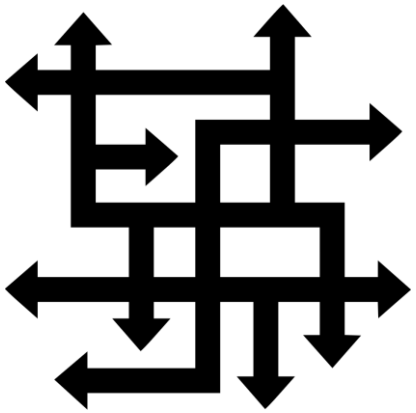


Reinforcement Learning (RL)

Infinite Runs



Complex Policies



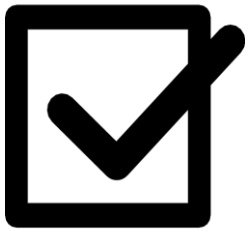
Communication Domain

But…

How do we know a Deep Neural Network trained via Reinforcement Learning is safe?

"Testing shows the presence, not the absence of bugs"

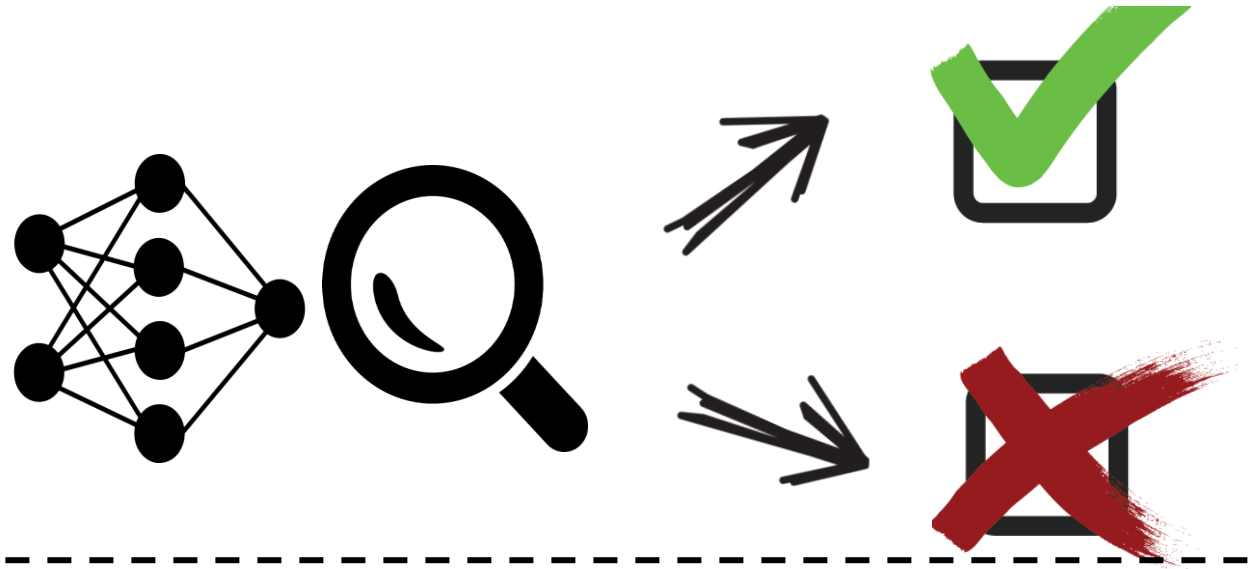
Dijkstra, 1969



***Challenge:** These "black boxes" need to be formally verified for correct behavior*

Our approach: Formal Verification!

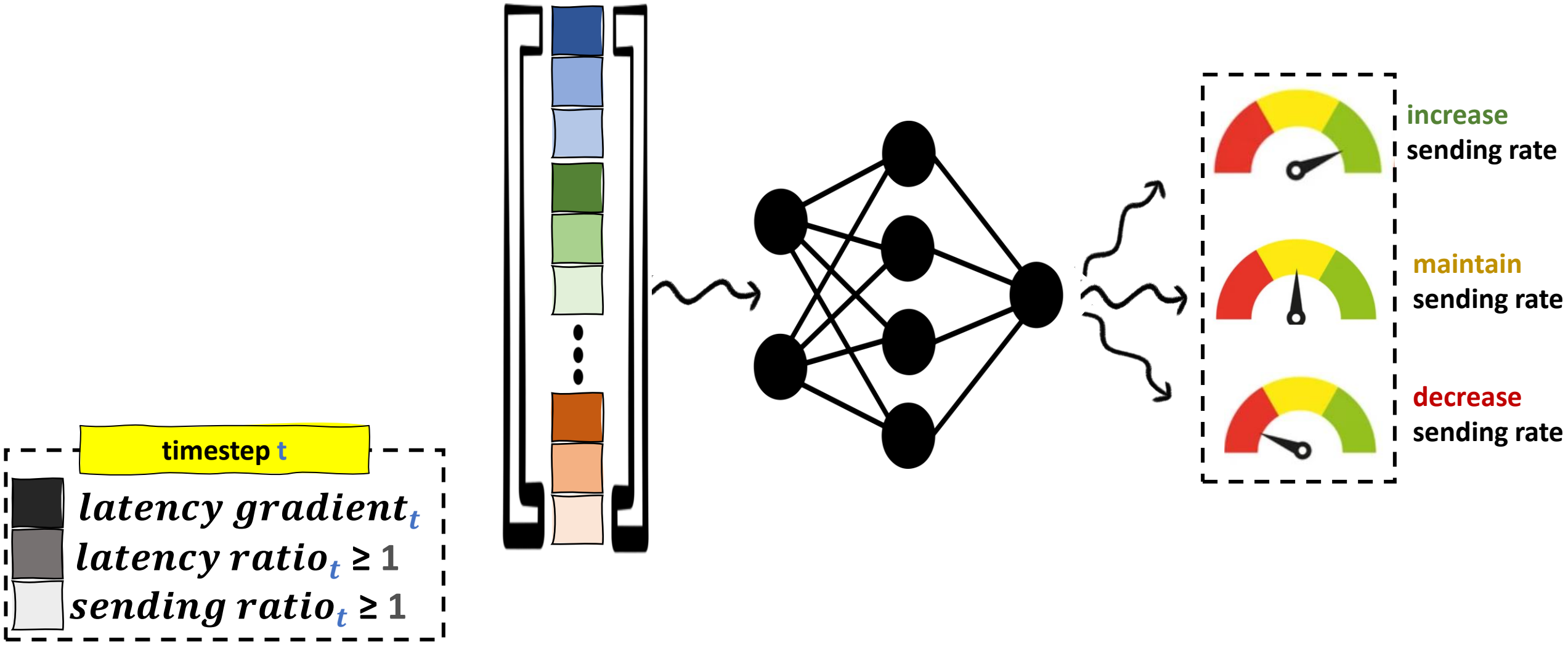
Provably guarantee that a learned policy meets our requirements, or identify concrete violations (bugs)



+ counter-example

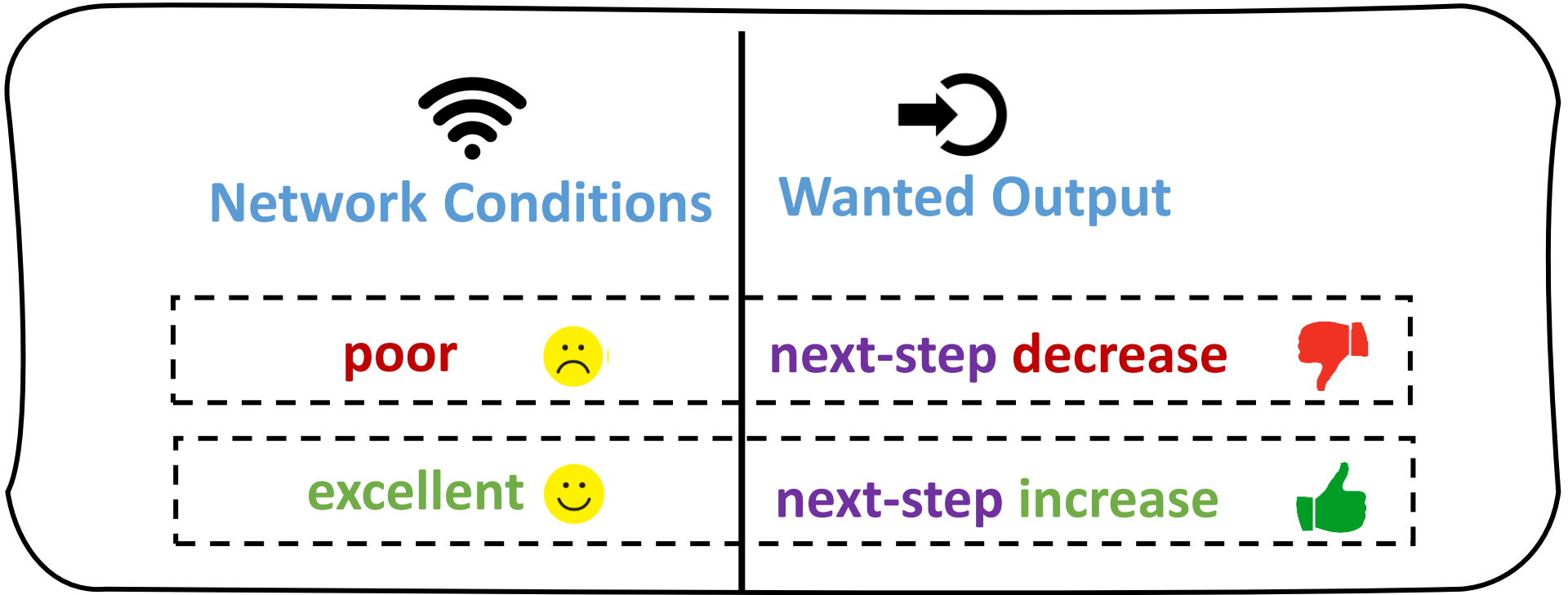
Example: The Aurora Congestion Controller

[Jay, Rotman, et al., ICML 2019]



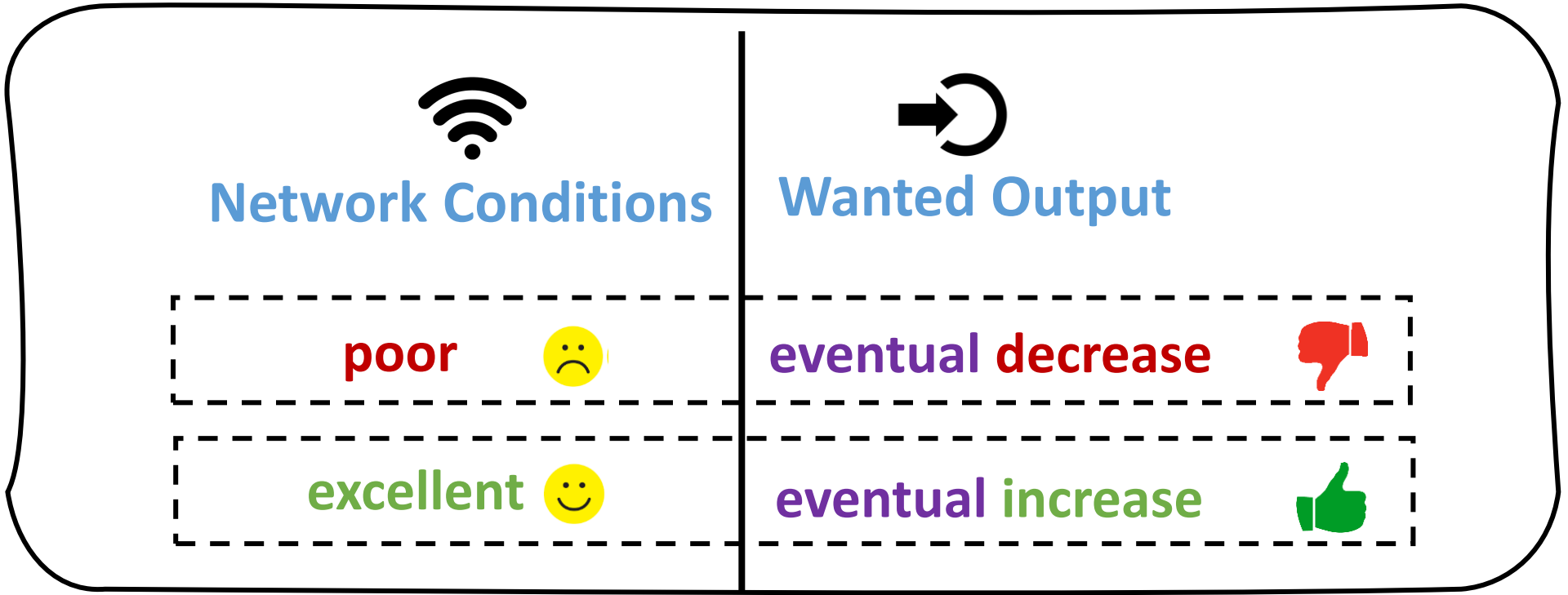
Aurora Safety Properties

Safety **- *"Something *bad* never happens"***
(finite-long violations)

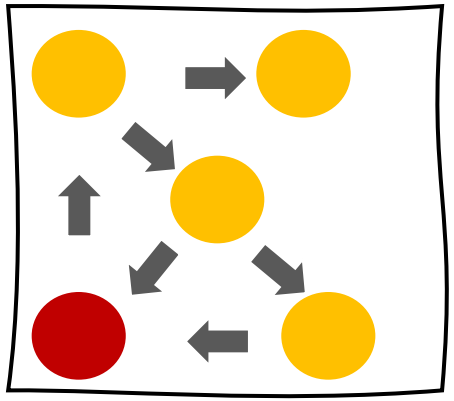


Aurora Liveness Properties

Liveness **- *"Something good eventually happens"***
(infinite-long violations)

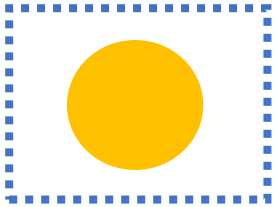


Our Verification Strategy

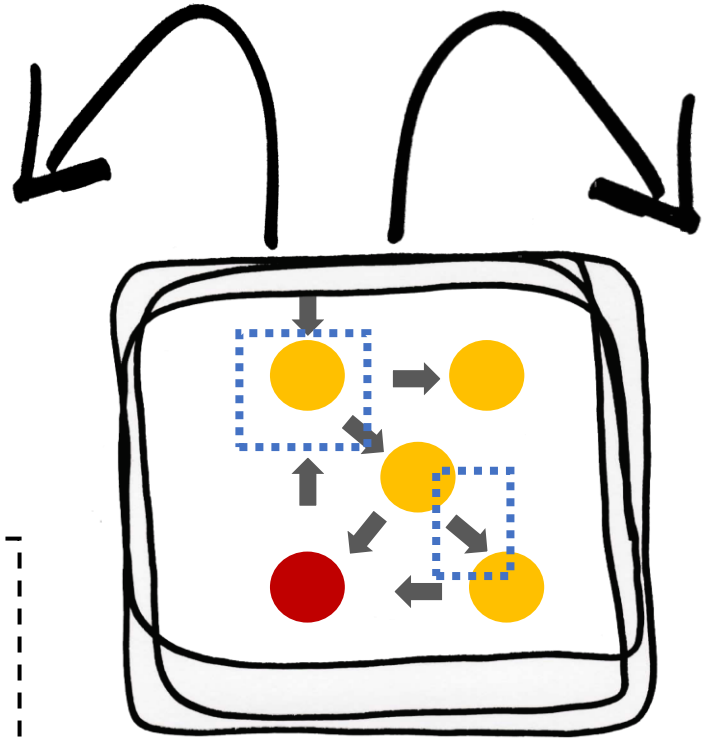


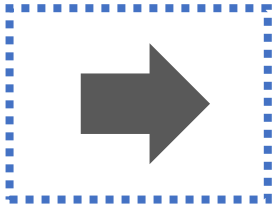
Defining a **state graph**
& **transition function**
[Eliyahu-Kazak-Katz-Schapira, SIGCOMM 2021]

Transition System Graph

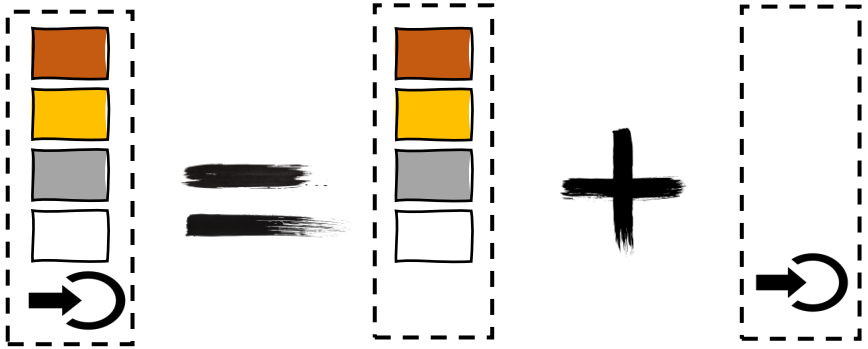


Defining a $state_t$

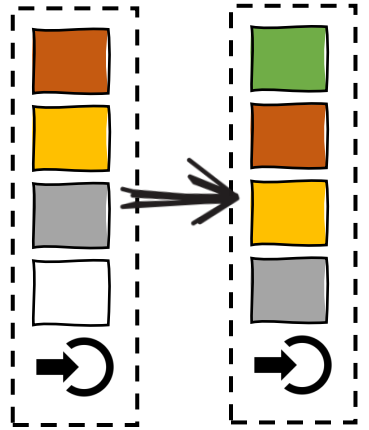




Defining a $transition_{t,t'}$

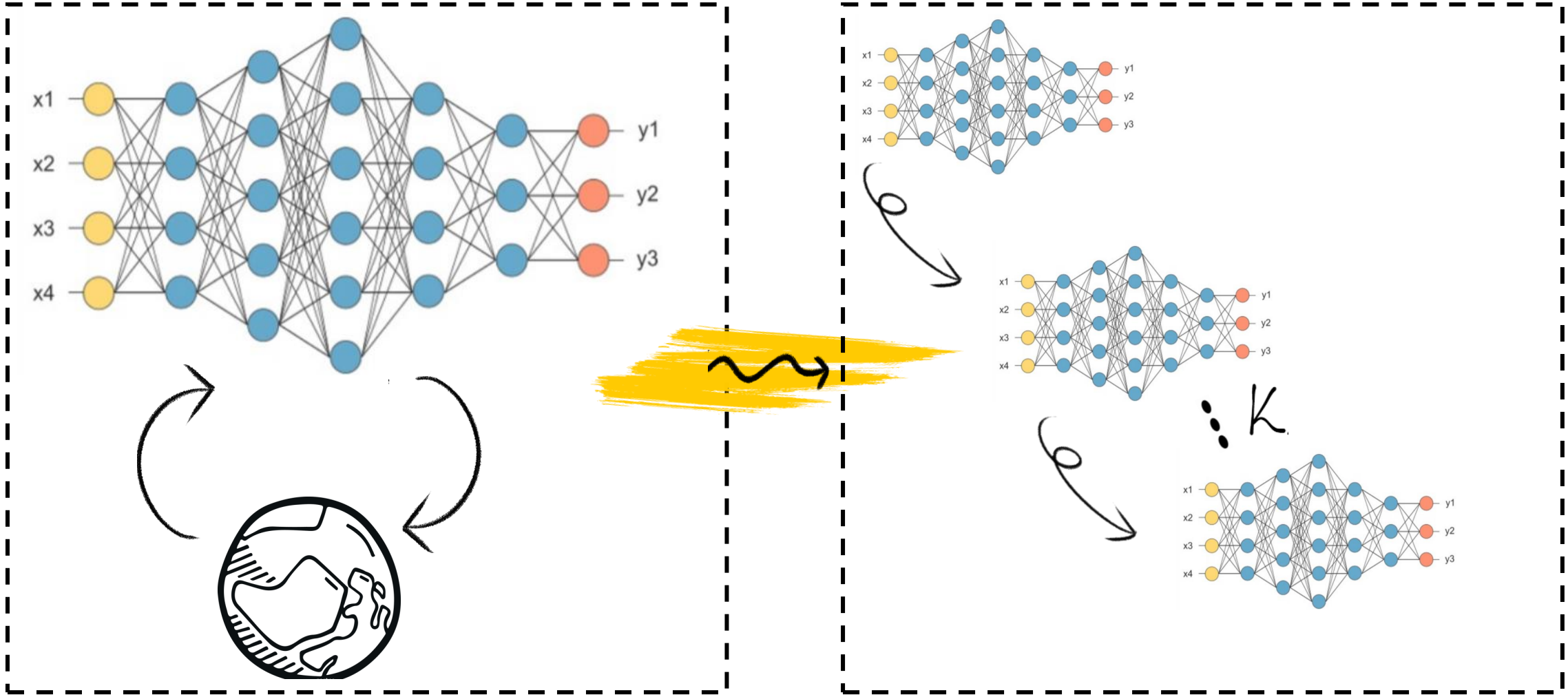


$state_t$ $input_t$ $output_t$

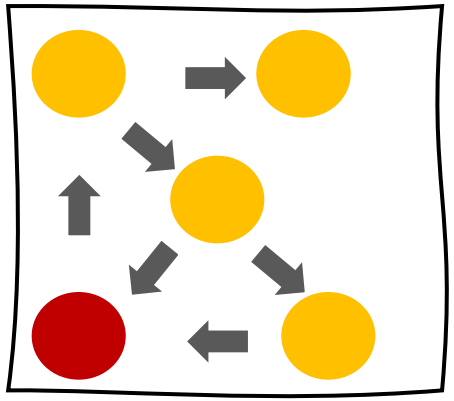


$state_t$ $state_{t'}$

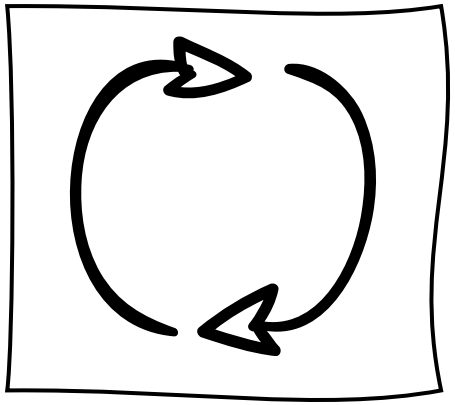
Encoding Multiple Transitions



Our Verification Strategy



Defining a **state graph**
& **transition function**
[Eliyahu-Kazak-Katz-Schapira, SIGCOMM 2021]



Running a **portfolio approach** for checking
k-long **violations** or **k**-long **provable runs**
[Amir-Schapira-Katz, FMCAD 2021]

Bounded Model Checking (BMC)

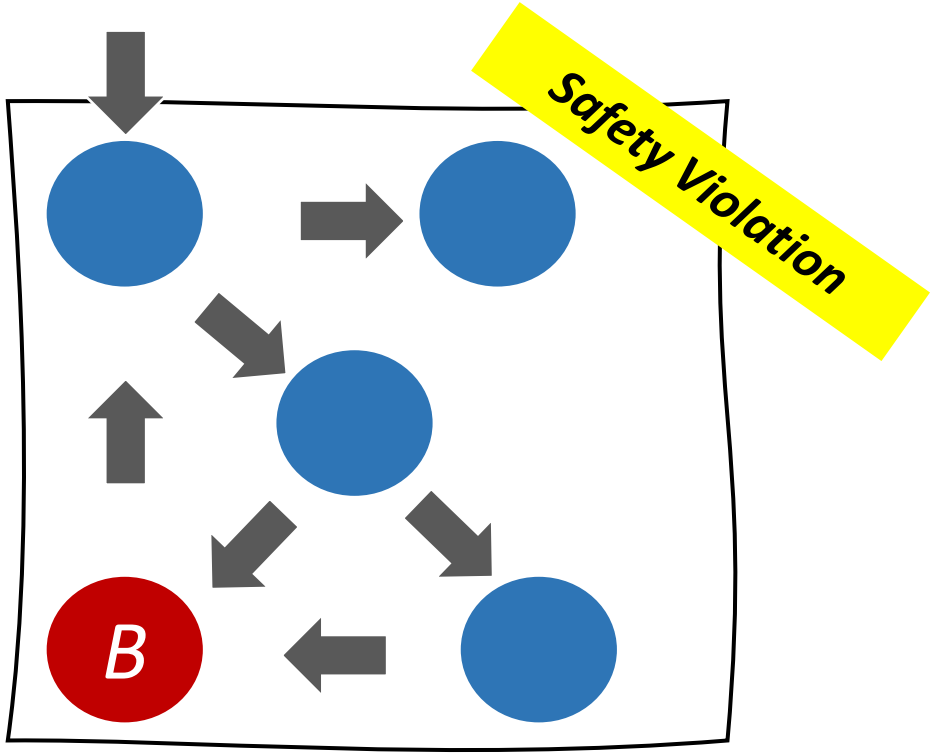
Bounded Model Checking

A method for checking violations of properties, for a given number of k steps

Bounded Model Checking (BMC)

Bounded Model Checking

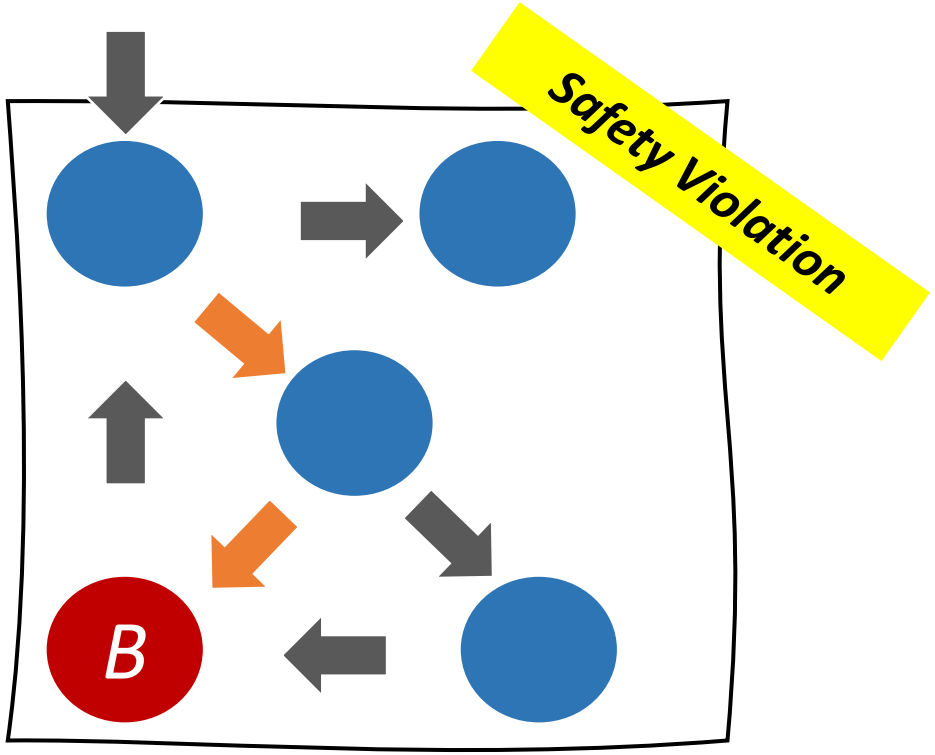
A method for checking violations of properties, for a given number of k steps

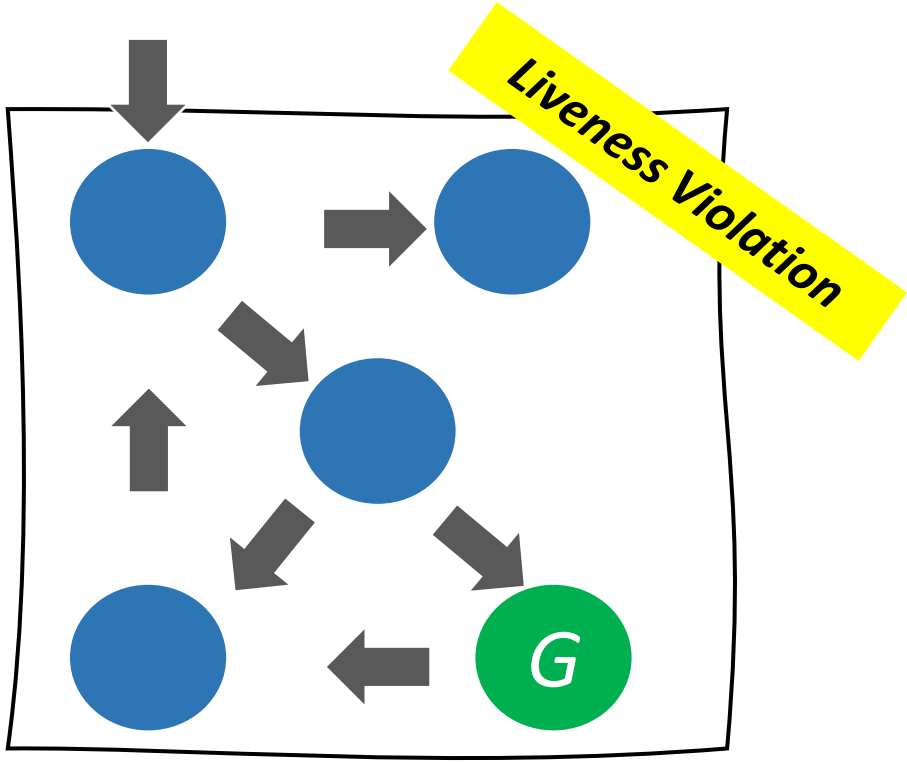


Bounded Model Checking (BMC)

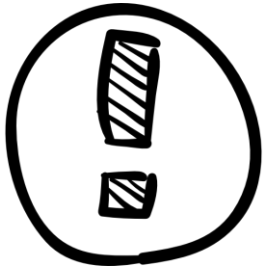
Bounded Model Checking

A method for checking violations of properties, for a given number of k steps

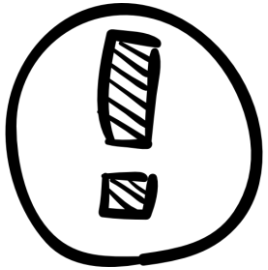




BMC Setbacks

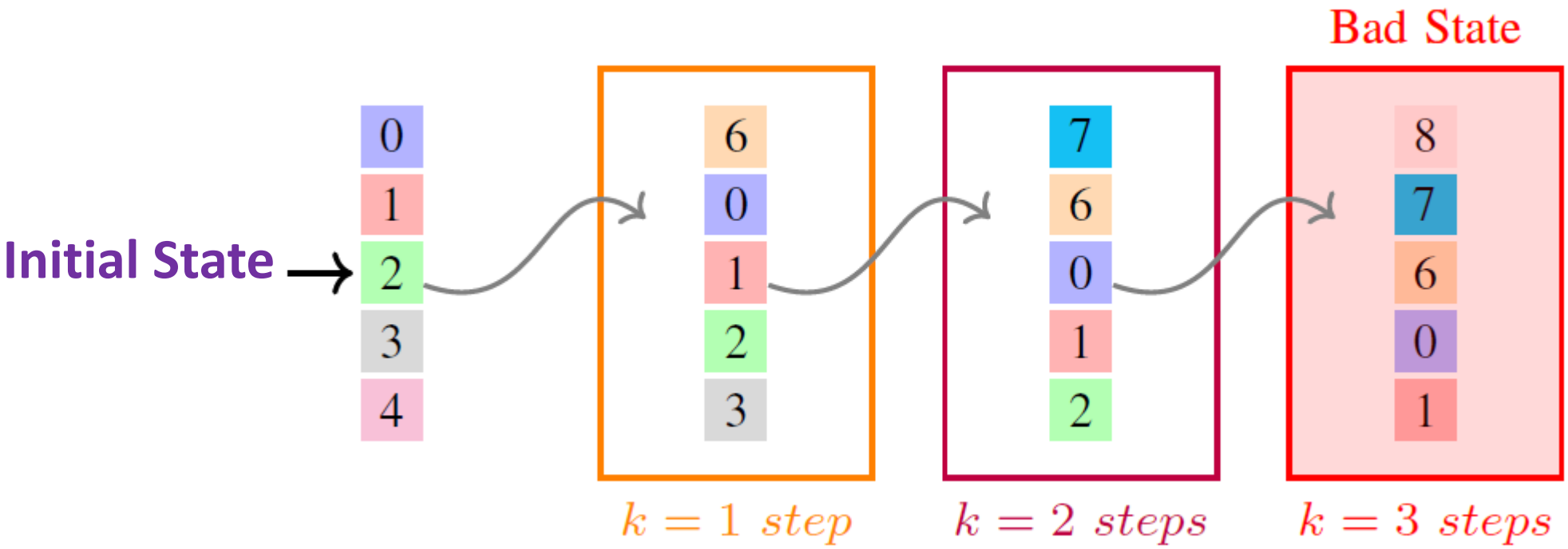


We can't prove that any properties hold

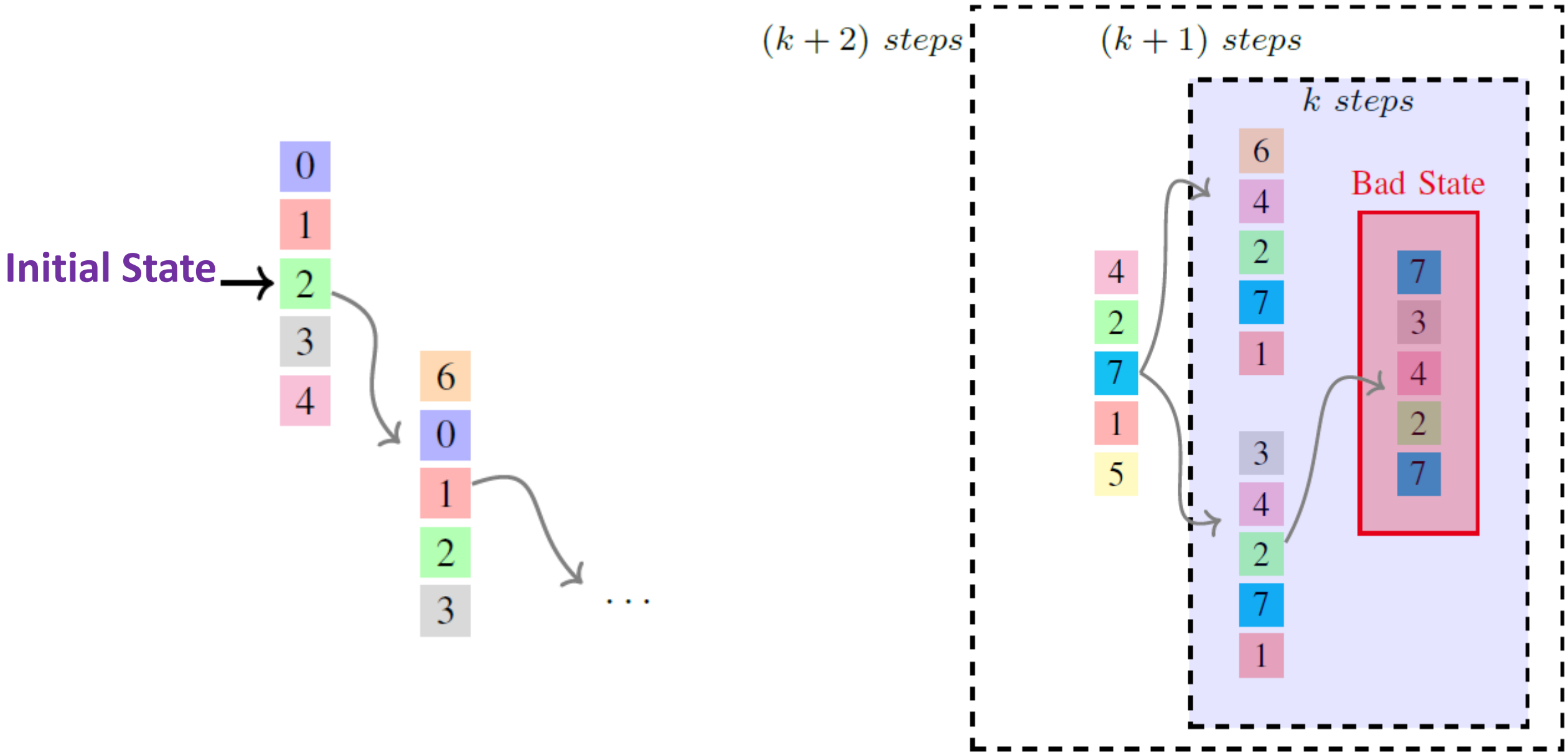


We can't analyze complex properties

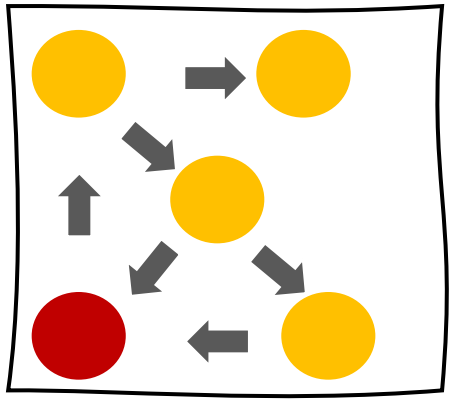
BMC


Bad State
Initial State
0
1
2
3
4
6
0
1
2
3
7
6
0
1
2
8
7
6
0
1
$k = 1$ step
$k = 2$ steps
$k = 3$ steps

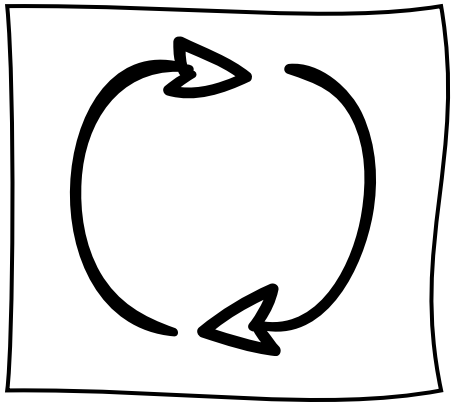
K-Induction


$(k+2)$ steps
$(k+1)$ steps
k steps
Initial State
Bad State
0
1
2
3
4
6
0
1
2
3
...
4
2
7
1
5
6
4
2
7
1
3
4
2
7
1
7
3
4
2
7

Our Verification Strategy

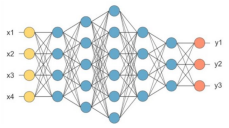


Defining a **state graph**
& **transition function**
[Eliyahu-Kazak-Katz-Schapira, SIGCOMM 2021]

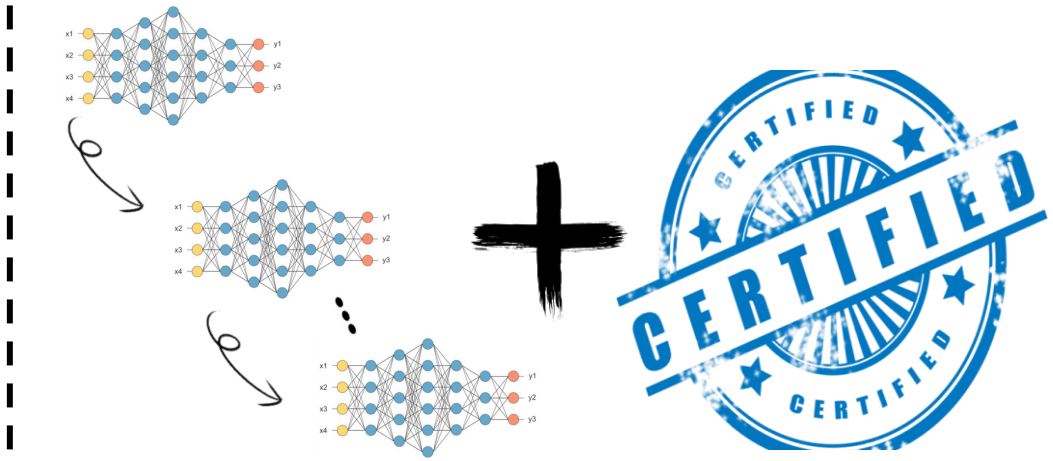


Running a **portfolio approach** for checking
k-long **violations** or **k**-long **provable runs**
[Amir-Schapira-Katz, FMCAD 2021]

Portfolio Approach

input:  + property

initialization: k=1 step



emulate k steps

correctness → K-Induction

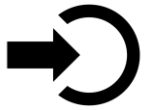
bug → BMC → exit

?

increment: k ++

Aurora Properties

Network Conditions



Desired Output

	Network Conditions	Desired Output	
Property 1	excellent ☺	eventual change 👍👎	✓
Property 2	excellent ☺	eventual increase 👍	✗ k=2
Property 3	poor ☹	next-step decrease 👎	✗ k=1
Property 4	poor ☹	eventual decrease 👎	✓

WhiRL 2.0 - Techniques

K-Induction

Invariant Inference

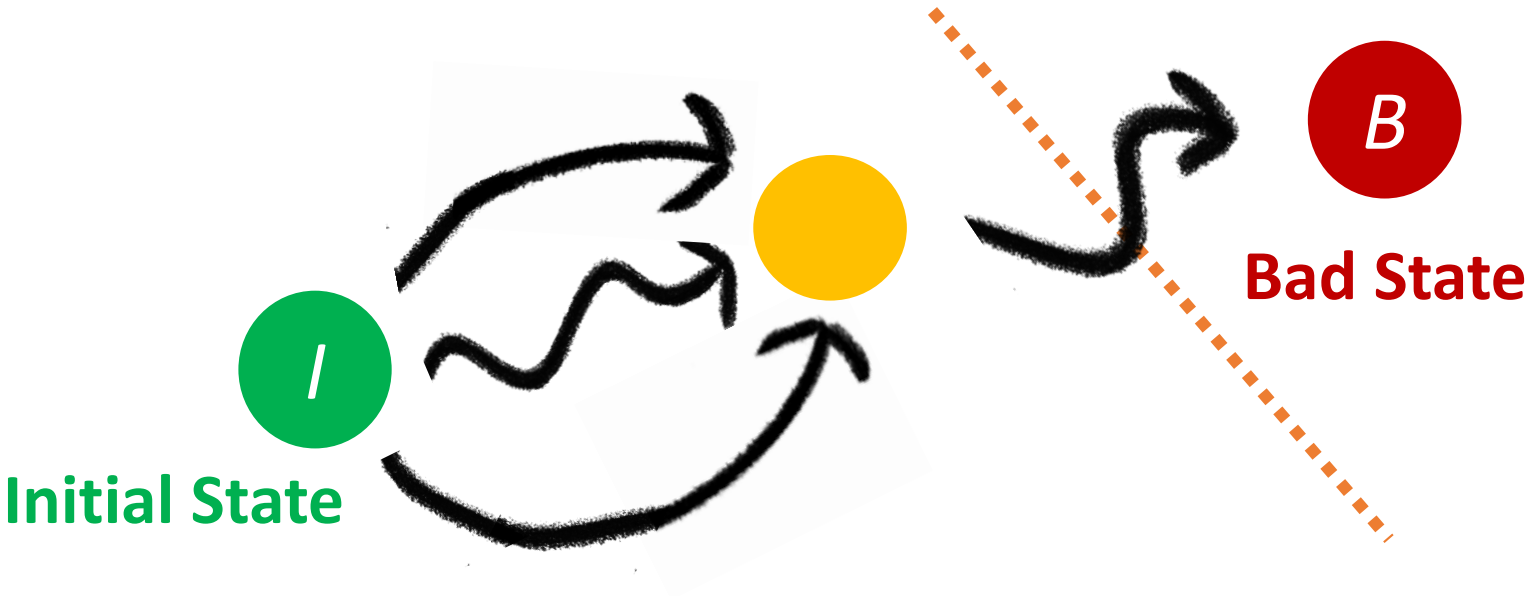
Abstraction

Invariant Inference

Invariant

A partition of the state space S into two disjoint sets S_1 and S_2 such that:

$$s_1 \in S_1 \wedge s_2 \in S_2 \rightarrow (s_1, s_2) \notin transtion\ T$$



Invariant Inference

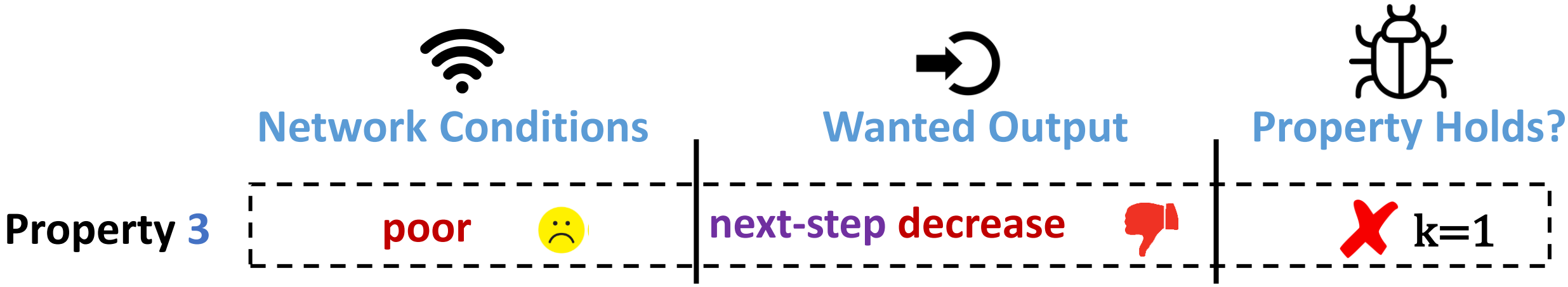
Templates:

- ✓ use *monotonicity* of properties
- ✓ fix *inputs* or *outputs*
- ✓ conduct a *binary search* on the non-fixed variables
- ✓ *dynamic*: user-chosen values

Strategy: search for the *"2nd best" behavior*

Invariant Inference - Aurora

- **Execution of *invariance inference algorithms* based on the *templates***
- **For example, an invariant is found, *based on the following violated property***



Invariant Inference - Aurora

- **Originally, "*poor*" network conditions:**

$$2 \leq sending\_ratio_t$$

- **We can search for the *worst-case sending ratio* for the output to decrease:**

$$output_t < 0$$

Invariant Inference - Aurora

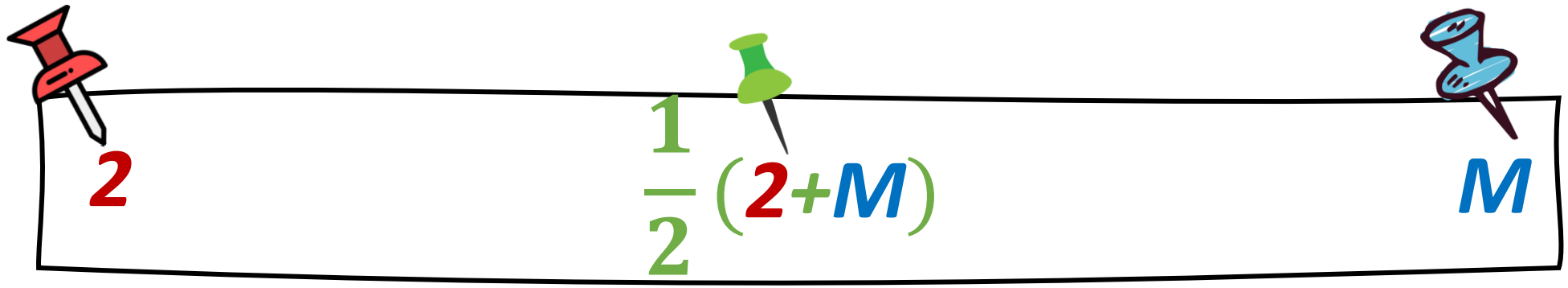
- **Initialization**: $0 \leq output, \quad 2 \leq sending\_ratio_t \leq M_{user}$
- **Iterate**:
  - binary-search the $sending\_ratio_t$ lower bound
  - call a $verifier$ on the middle point
  - update $sending\_ratio_t$ accordingly
- **Return**: lower bound on worst case $sending\_ratio_t$

Invariant Inference - Aurora

$sending\_ratio_t$ lower bound for *SAT* → **2**

$sending\_ratio_t$ lower bound for *UNSAT* → **M**

$$verifier\{sending_{ratio_t} \in [2, M]\} \rightarrow SAT$$

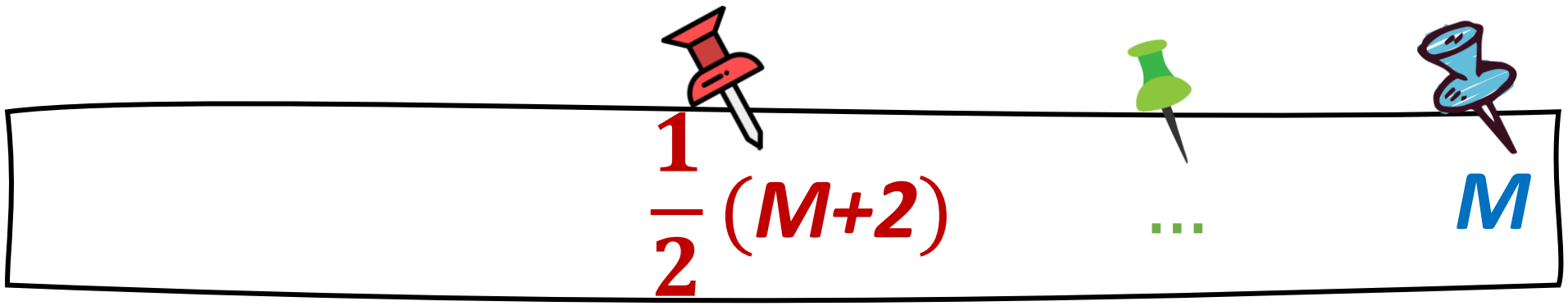


$$verifier\{sending_{ratio_t} \in [\frac{1}{2}(2+M), M]\} \rightarrow SAT$$

Invariant Inference - Aurora

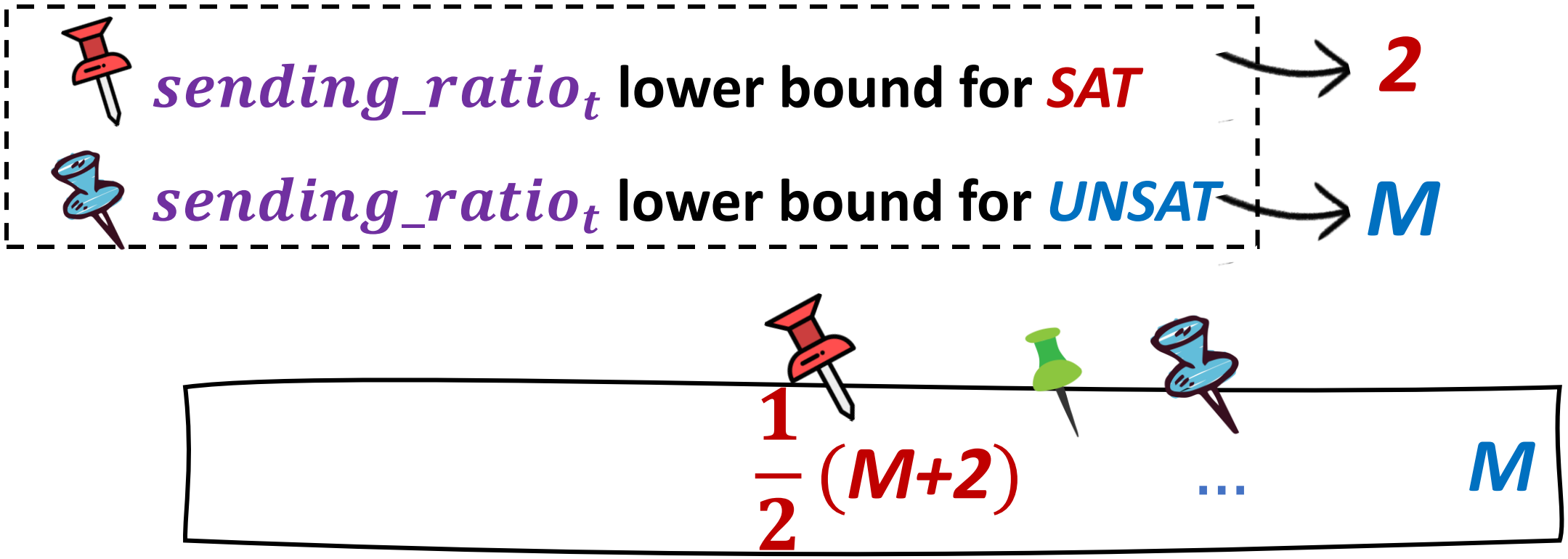
$sending\_ratio_t$ lower bound for *SAT* → *2*

$sending\_ratio_t$ lower bound for *UNSAT* → *M*



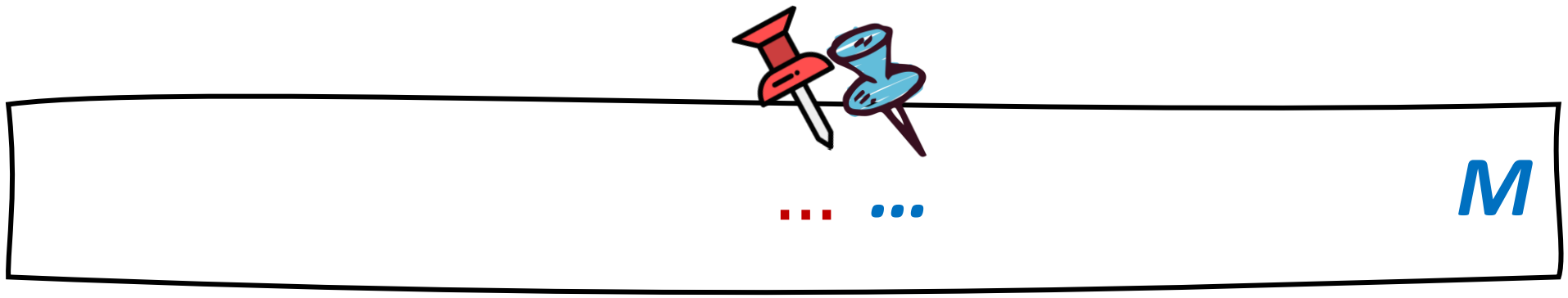
$$verifier\{sending_{ratio_t} \in [\frac{1}{2}(\frac{1}{2}(M+2)+M), M]\} \rightarrow UNSAT$$

Invariant Inference - Aurora



Invariant Inference - Aurora

after $log(M)$ iterations:



***return*:**  **= lower bound on worst-case $sending\_ratio_t$**

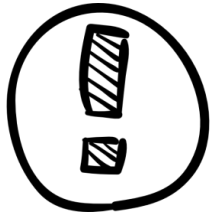
Techniques

K-Induction

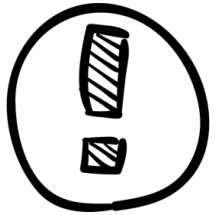
Invariant Inference

~~*Abstraction*~~

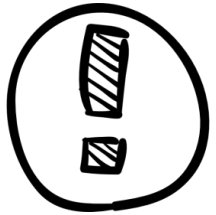
See paper for…



Abstraction techniques for generalization



Methods for identifying undesirable policies



Modules for improving interpretability

[Amir-Schapira-Katz, FMCAD 2021]

Summary

- A (first?) method for proving properties of RL-driven systems
- Automatic invariant inference of "2nd best" properties, in chosen scenarios
- Explainability and interpretability of bad policies

Future Steps

Improve scalability

Focus on generalization

Questions