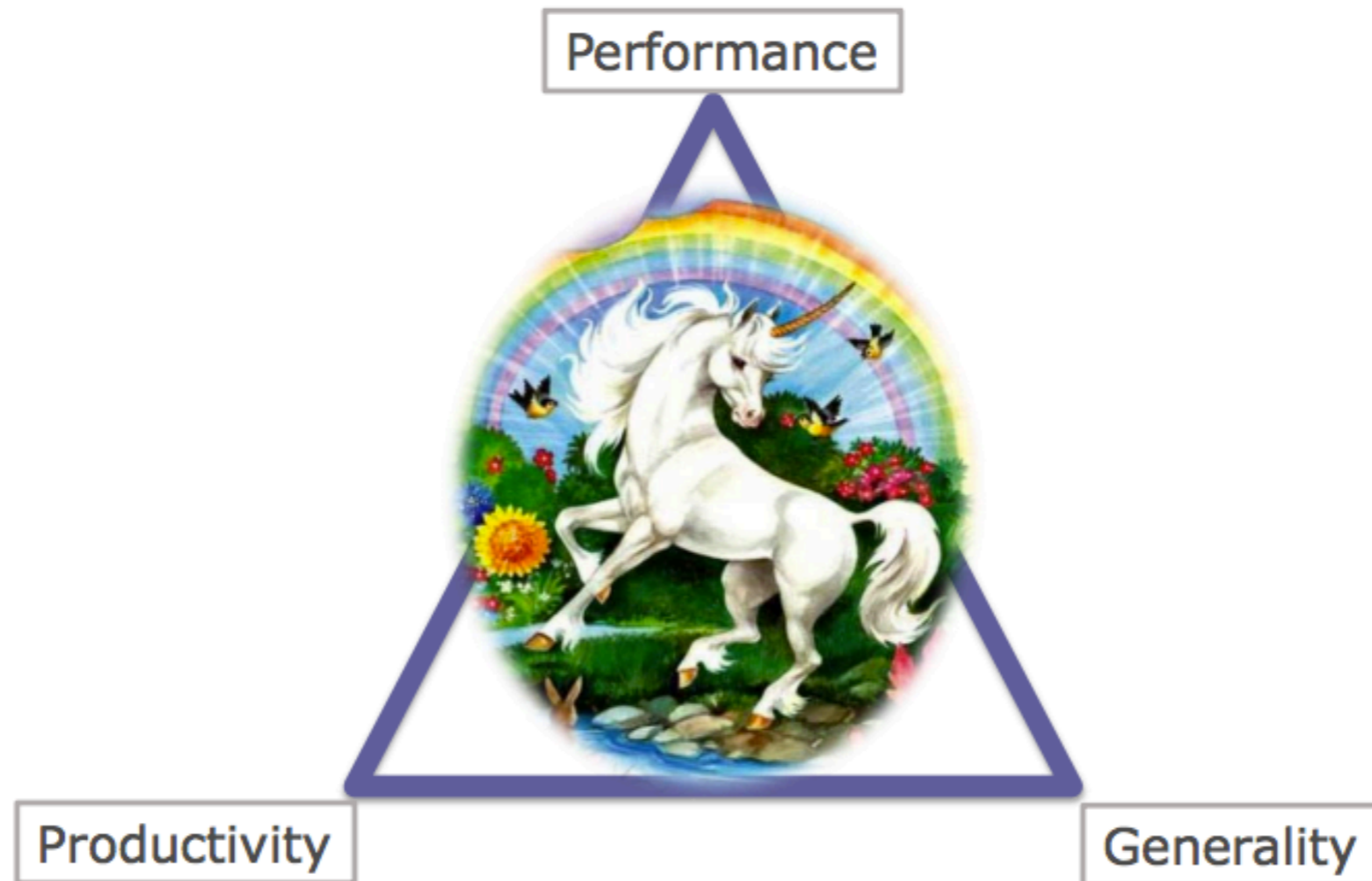


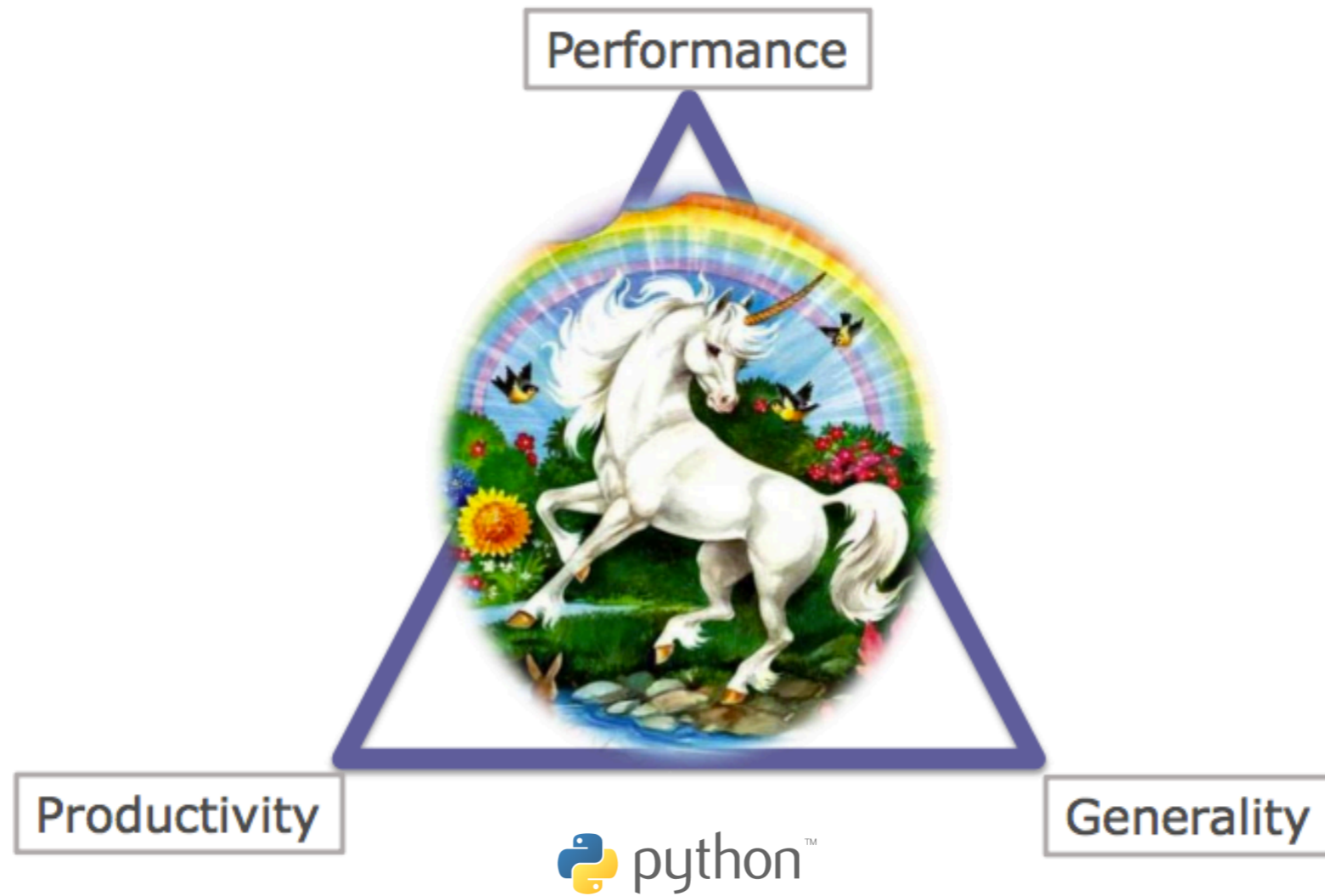
domain-specific SMT solving for neural network verification (or anything else)

Lindsey Kuper
UC Santa Cruz



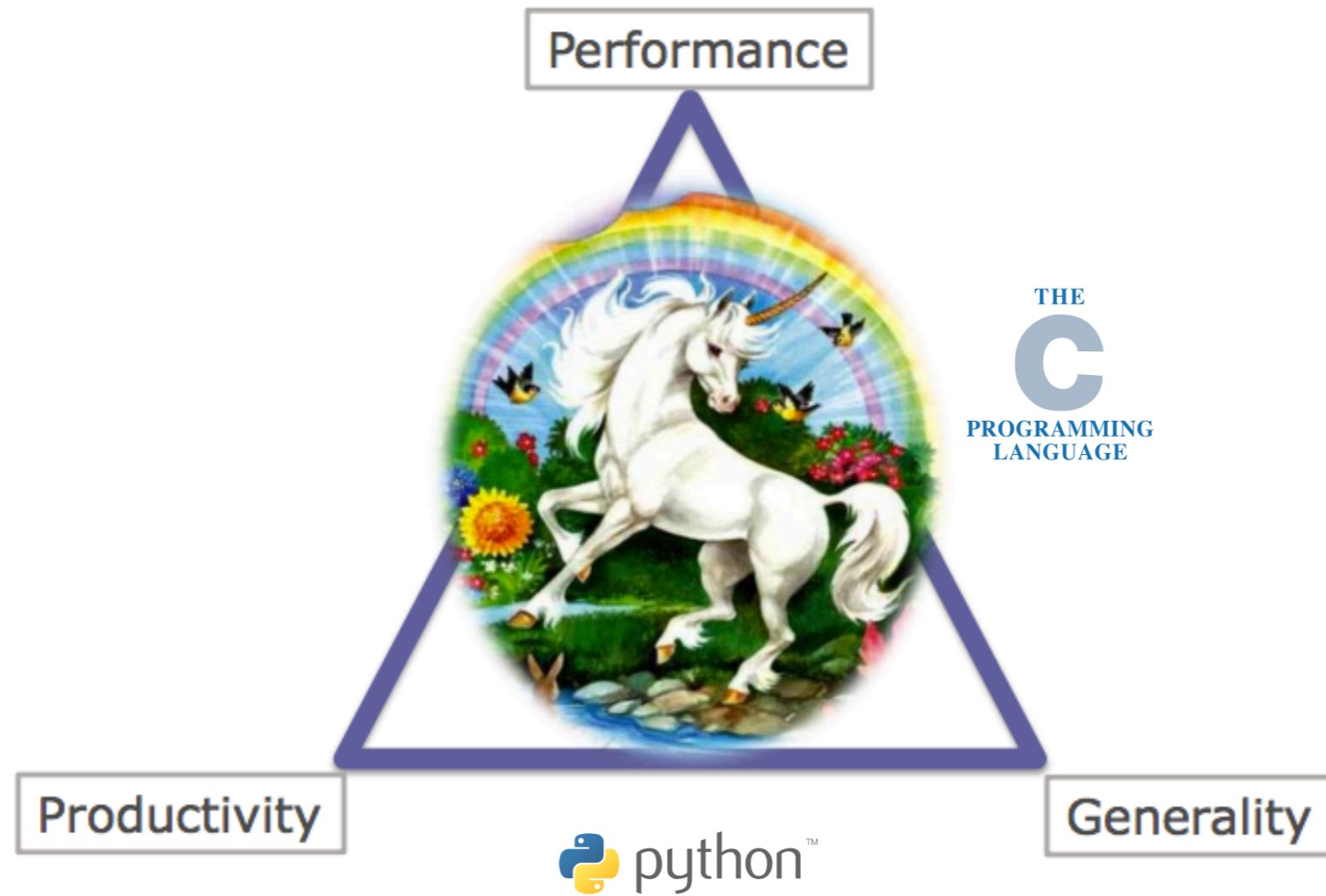
[Olukotun et al., 2012]

**sometimes it's worth trading generality
for productivity + performance**



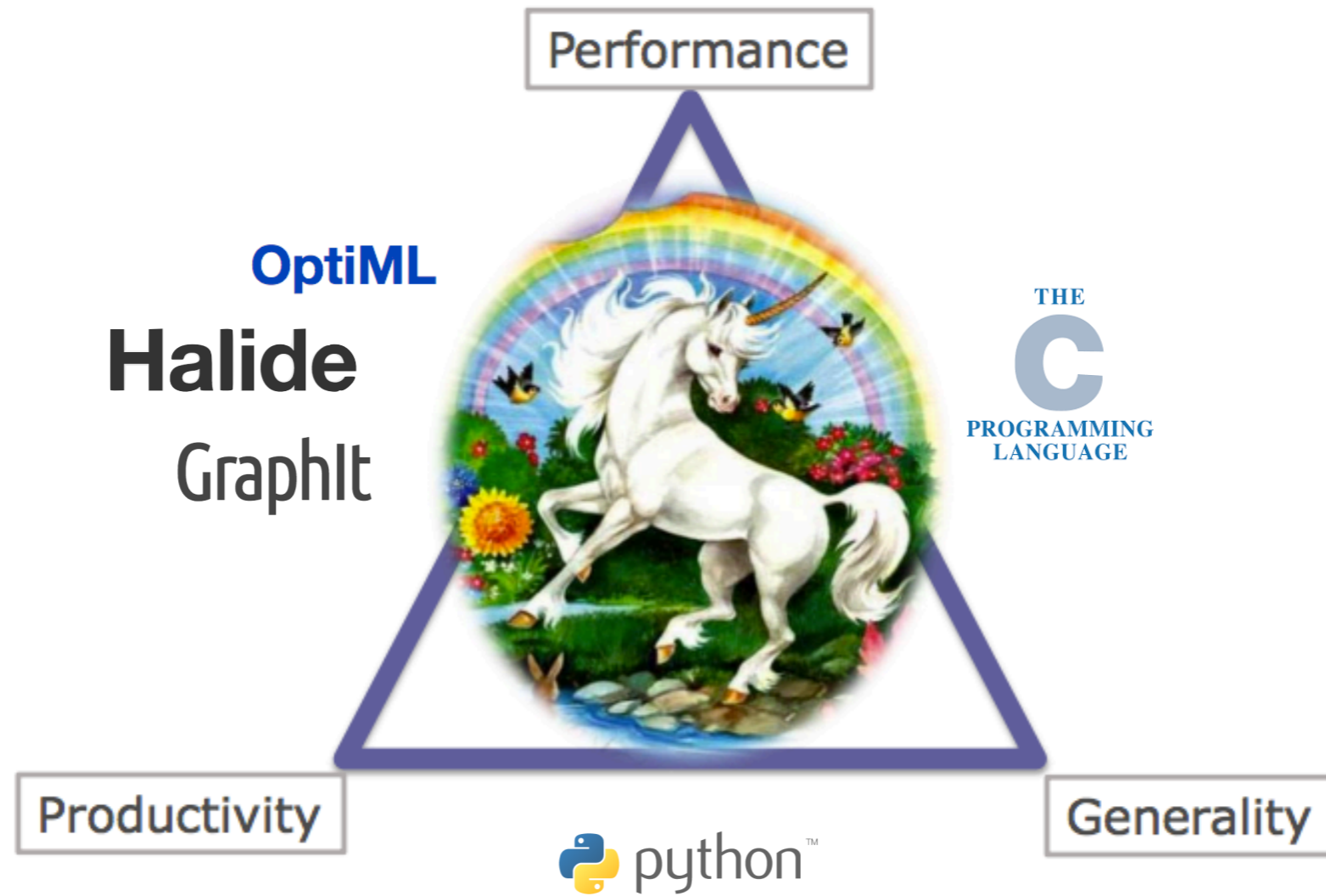
[Olukotun et al., 2012]

**sometimes it's worth trading generality
for productivity + performance**



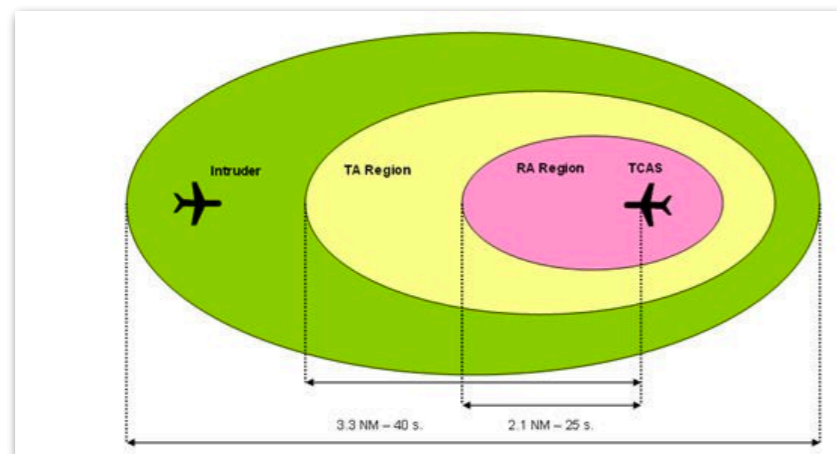
[Olukotun et al., 2012]

**sometimes it's worth trading generality
for productivity + performance**



[Olukotun et al., 2012]

**sometimes it's worth trading generality
for productivity + performance**



[Julian et al., 2016]

Policy Compression for Aircraft Collision Avoidance Systems

Kyle D. Julian*, Jessica Lopez[†], Jeffrey S. Brush[‡], Michael P. Owen[‡] and Mykel J. Kochenderfer*

*Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, 94305

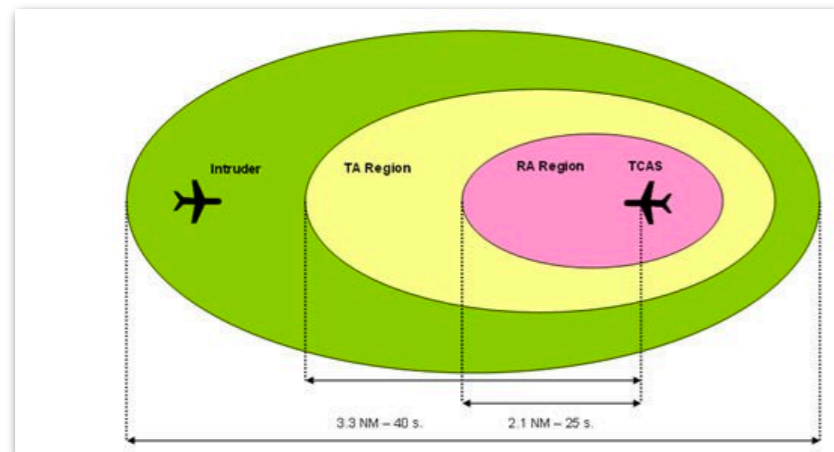
[†]Applied Physics Laboratory, Johns Hopkins University, Laurel, MD, 20723

[‡]Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA, 02420

Abstract—One approach to designing the decision making logic for an aircraft collision avoidance system is to frame the problem as Markov decision process and optimize the system using dynamic programming. The resulting strategy can be represented as a numeric table. This methodology has been used in the development of the ACAS X family of collision avoidance systems for manned and unmanned aircraft. However, due to the high dimensionality of the state space, discretizing the state variables can lead to very large tables. To improve storage efficiency, we propose two approaches for compressing the lookup table. The first approach exploits redundancy in the table. The table is decomposed into a set of lower-dimensional tables, some of which can be represented by single tables in areas where the lower-dimensional tables are identical or nearly identical with respect to a similarity metric. The second approach uses a deep neural network to learn a complex non-linear function approximation of the table. With the use of an asymmetric loss function and a

is extremely large, requiring hundreds of gigabytes of floating point storage. A simple technique to reduce the size of the score table is to downsample the table after dynamic programming. To minimize the deterioration in decision quality, states are removed in areas where the variation between values in the table are smooth. This allows the table to be downsampled with only minor impact on overall decision performance. The downsampling reduces the size of the table by a factor of 180 from that produced by dynamic programming. For the rest of this paper, we refer to the downsampled ACAS Xu horizontal table as our baseline, original table.

Even after downsampling, the current table requires over 2GB of floating point storage. Discretized score tables like this have been compressed with Gaussian processes [6] and



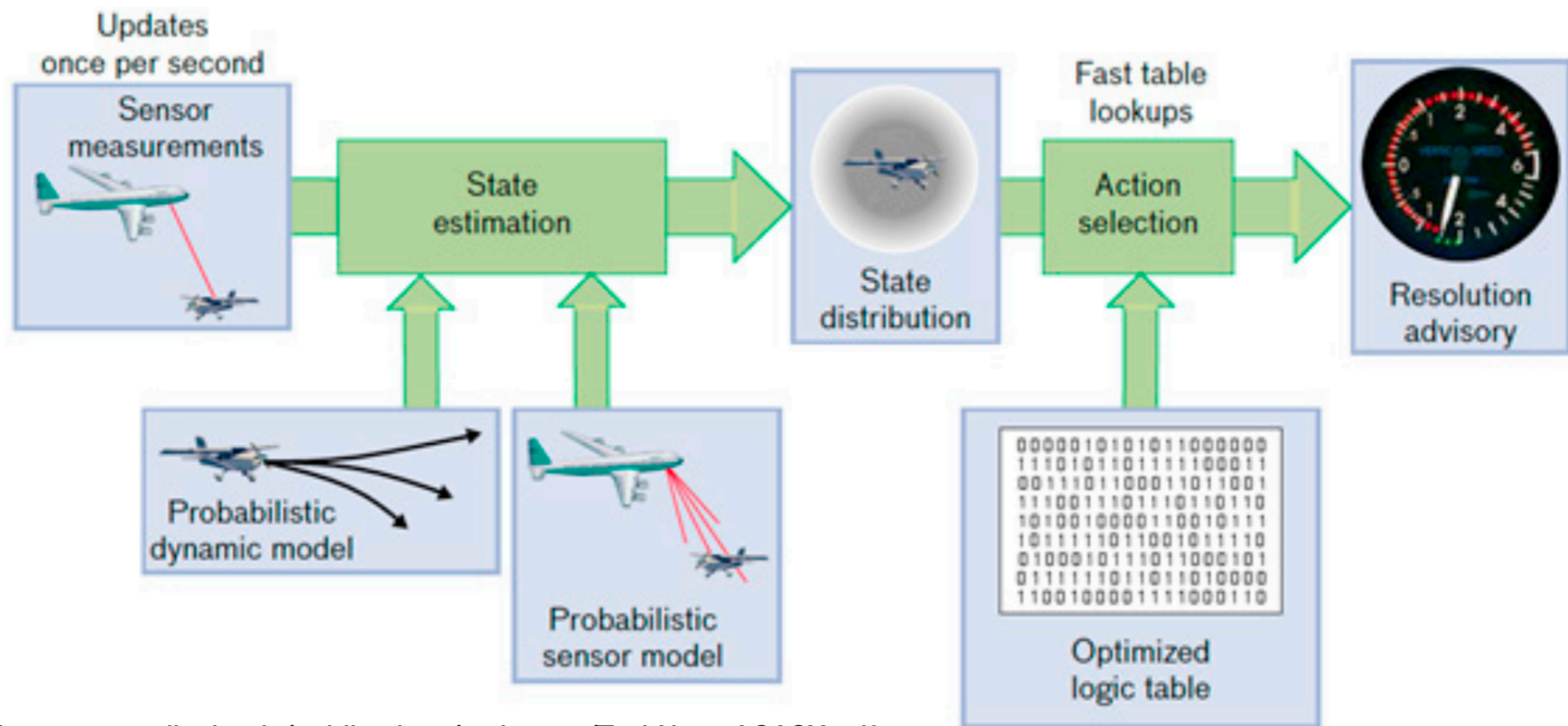
[Julian et al., 2016]

Policy Compression for Aircraft Collision Avoidance Systems

to a similarity metric. The second approach uses a deep neural network to learn a complex non-linear function approximation of the table. With the use of an asymmetric loss function and a preserving the relative preferences of the possible advisories for each state. As a result, the table can be approximately represented by only the parameters of the network, which reduces the required storage space by a factor of 1000. Simulation studies show that system performance is very similar using either concrete representation. Although there are significant certification concerns with neural network representations, which may be addressed in the future, these results indicate a promising way

input: sensor data
once per second

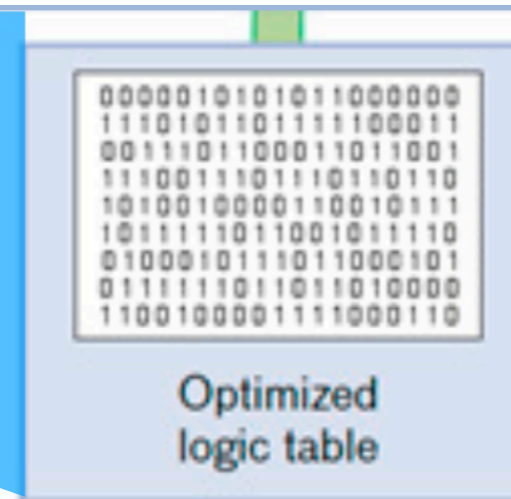
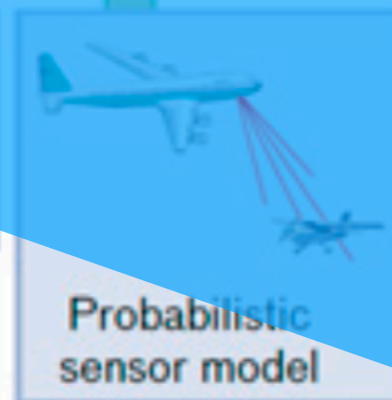
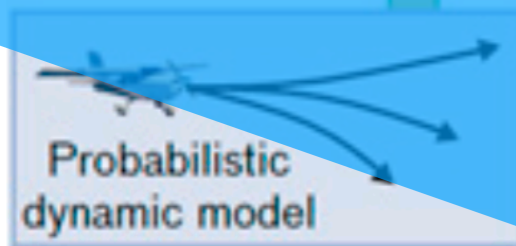
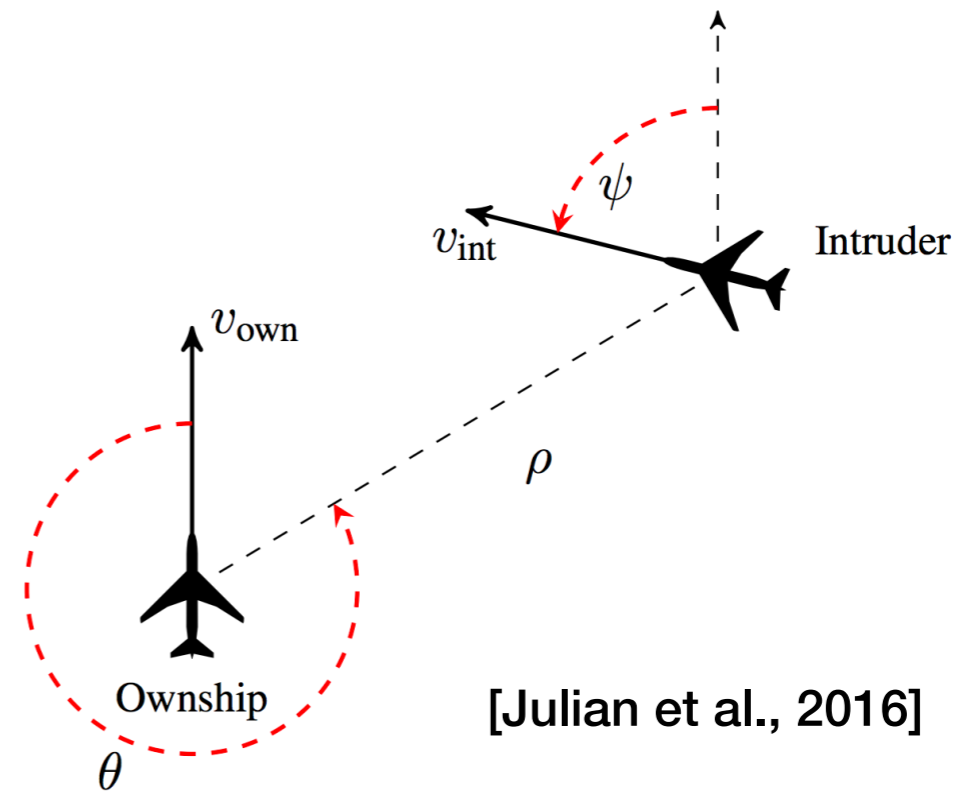
output: one of five resolution advisories
(COC, weak left, weak right, strong left, strong right)



Source: www.ll.mit.edu/publications/technotes/TechNote_ACASX.pdf

~120M 7-dimensional states

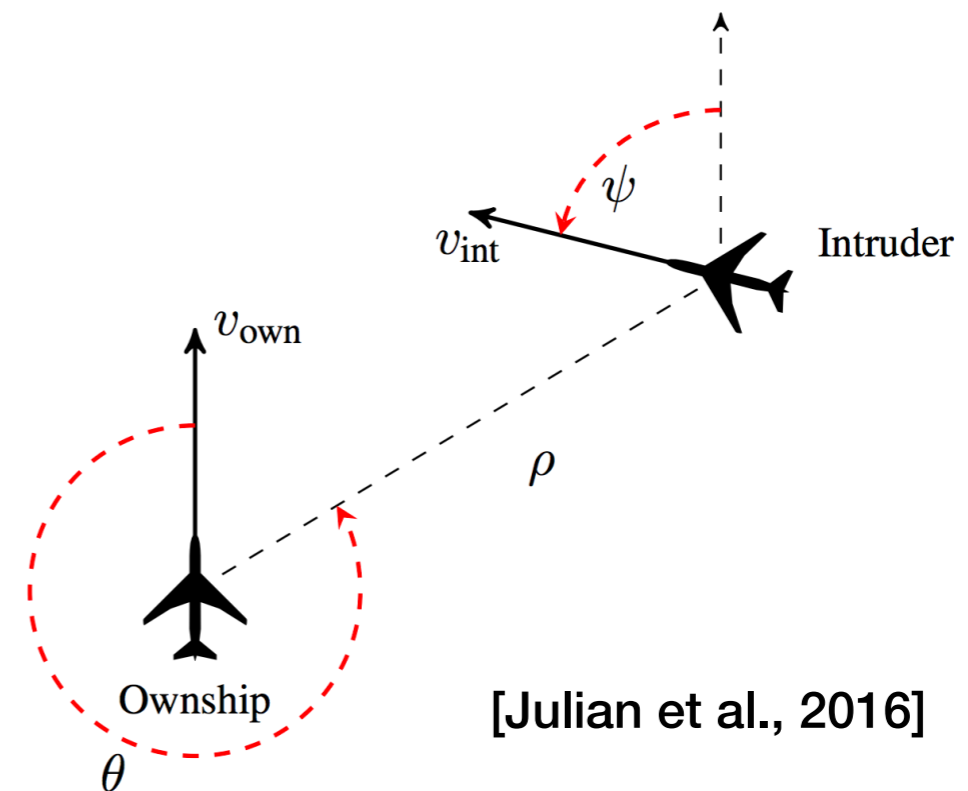
ρ : distance from ownship to intruder
 θ : angle to intruder
 ψ : heading angle of intruder
 v_{own} : speed of ownship
 v_{int} : speed of intruder
 τ : time until loss of vertical separation
 a_{prev} : previous advisory



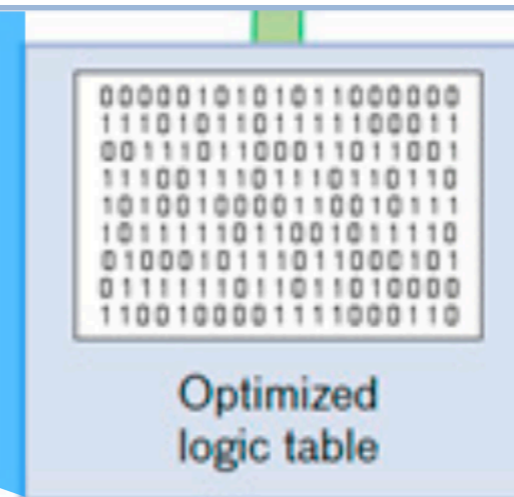
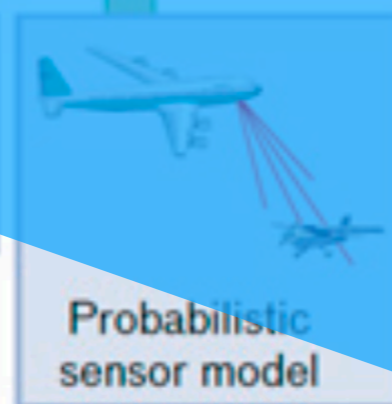
Source: www.ll.mit.edu/publications/technotes/TechNote_ACASX.pdf

~120M 7-dimensional states

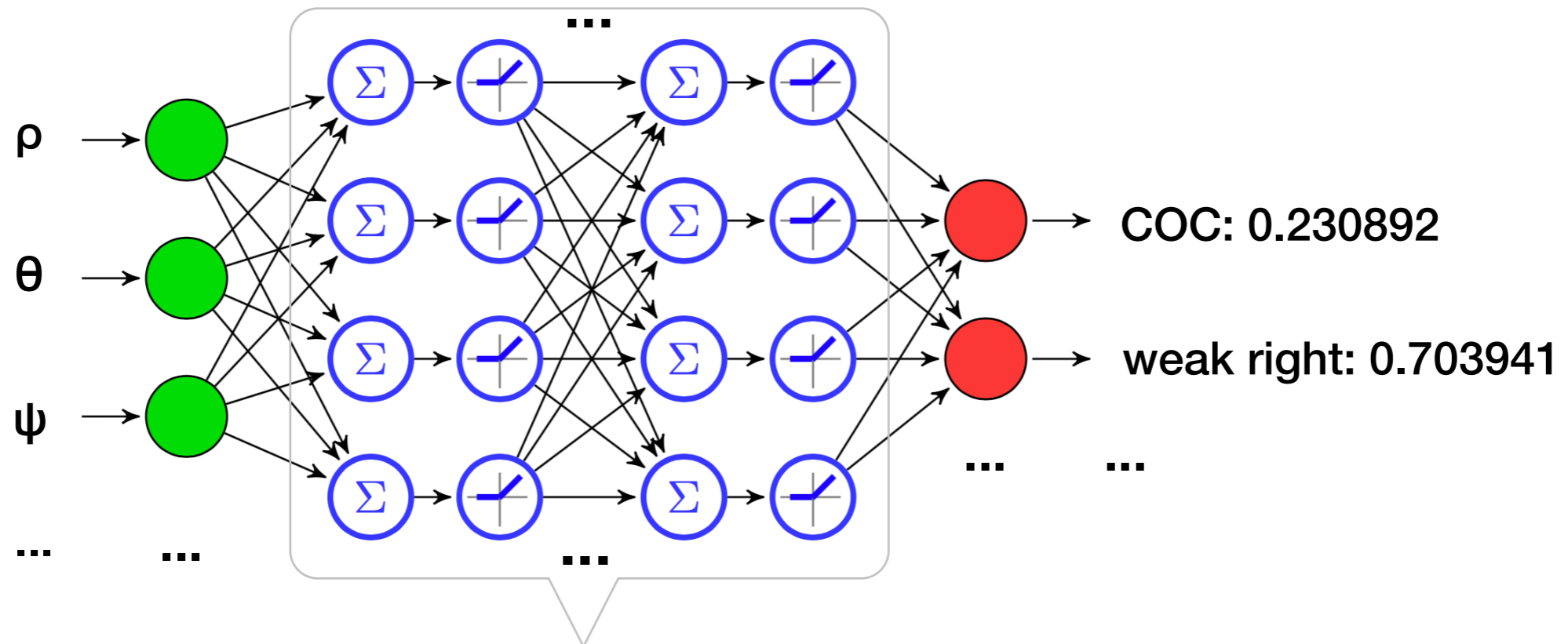
ρ : distance from ownship to intruder
 θ : angle to intruder
 ψ : heading angle of intruder
 v_{own} : speed of ownship
 v_{int} : speed of intruder
 τ : time until loss of vertical separation
 a_{prev} : previous advisory



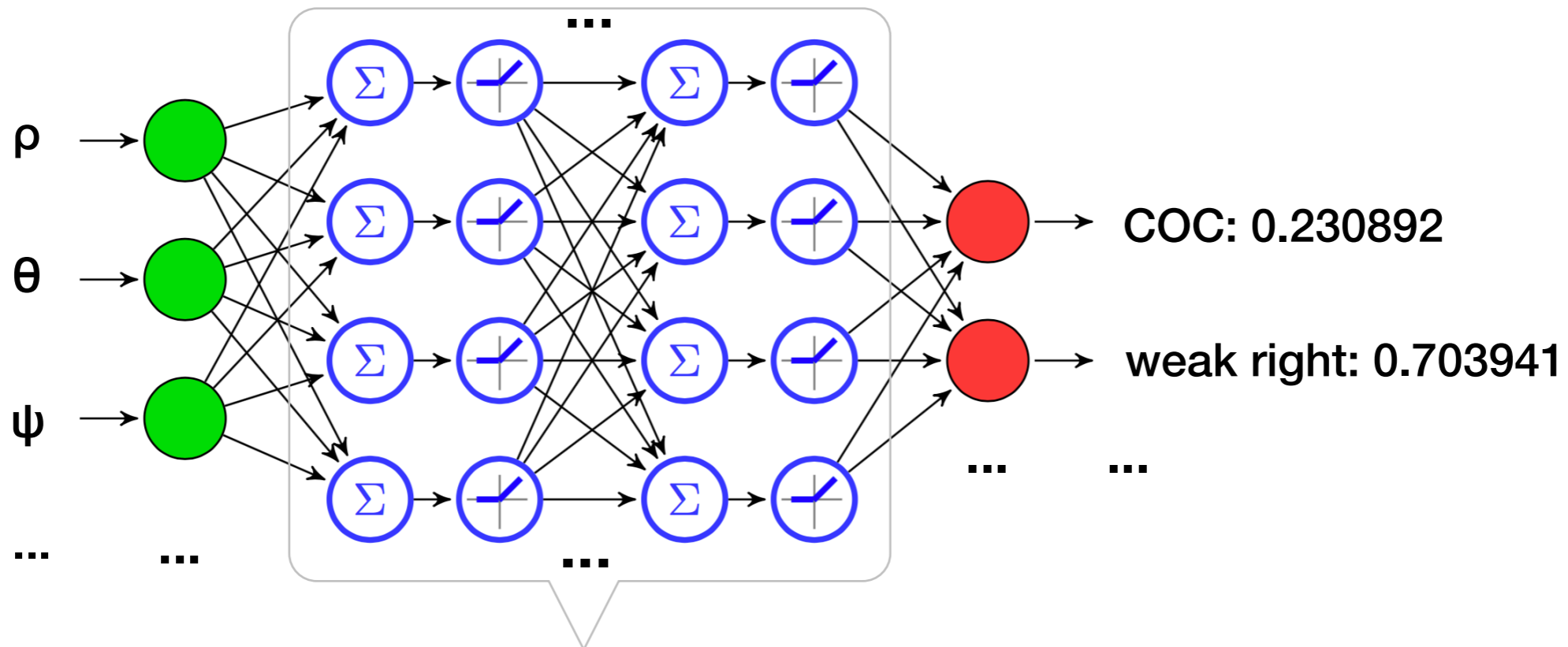
**needs 100s of GBs of storage—
too big to fit in memory on verified hardware**



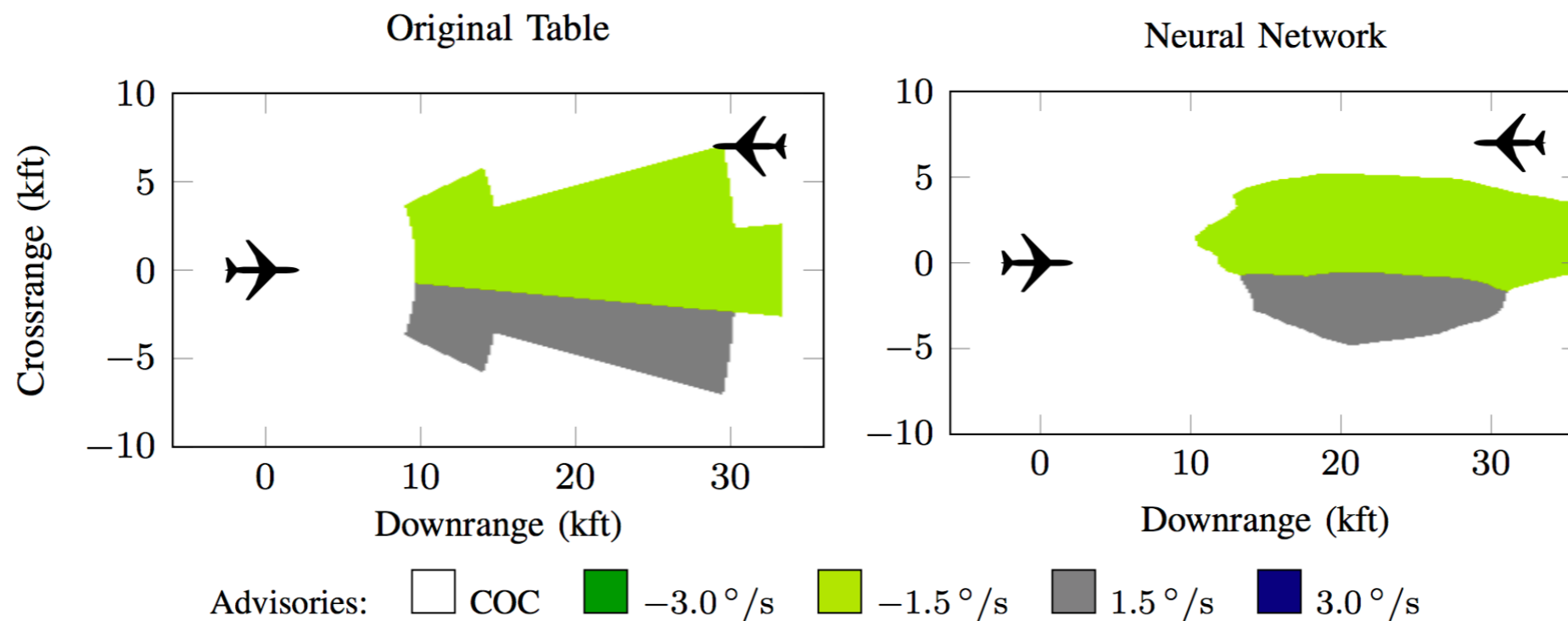
Source: www.ll.mit.edu/publications/technotes/TechNote_ACASX.pdf



9 fully-connected layers with ReLU activations ($f(x) = \max(0, x)$)



9 fully-connected layers with ReLU activations ($f(x) = \max(0, x)$)

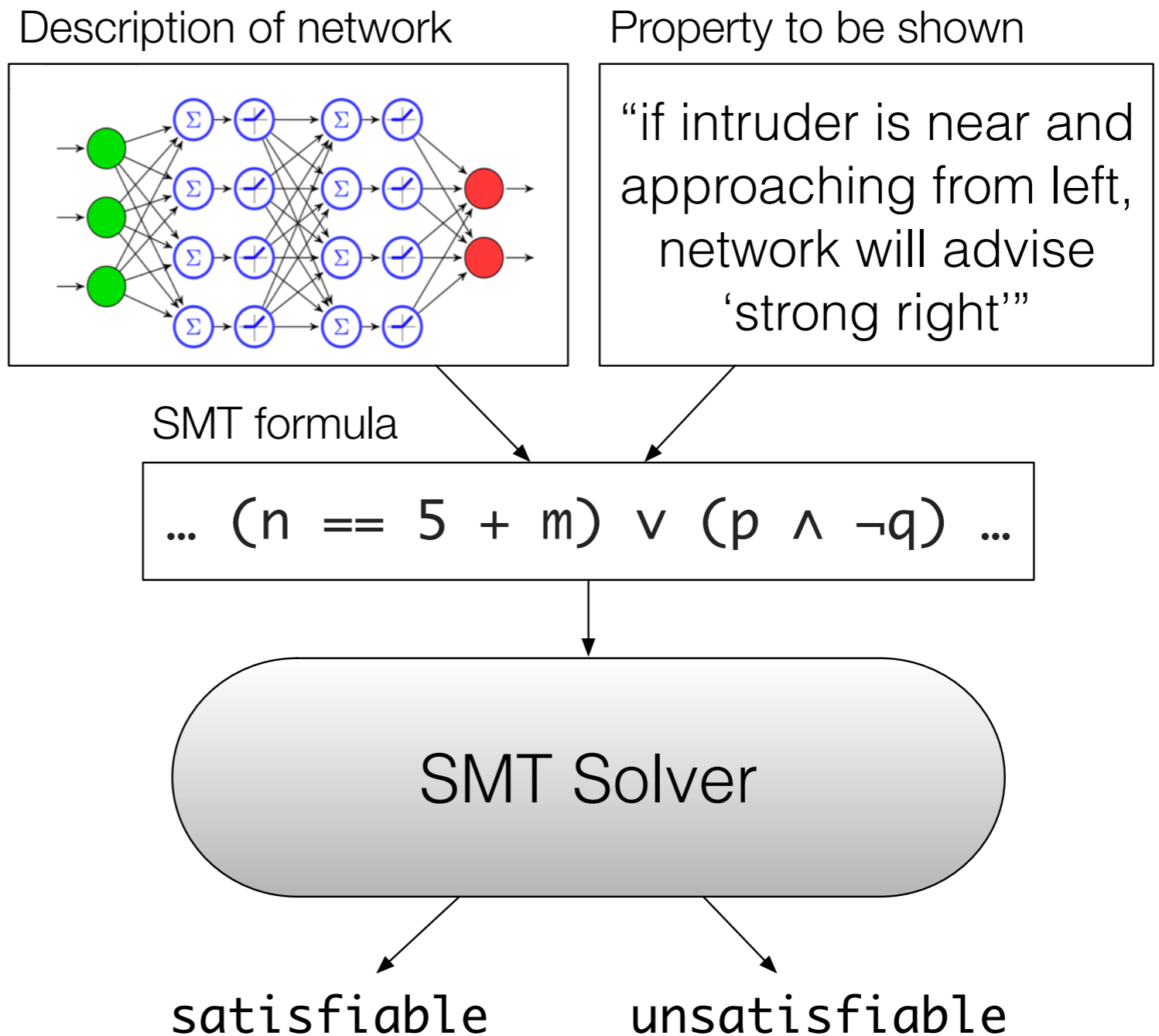


[Julian et al., 2016]

verification strategy

SAT: determine if a Boolean formula (containing only Boolean variables, parens, \wedge , \vee , \neg) is satisfiable

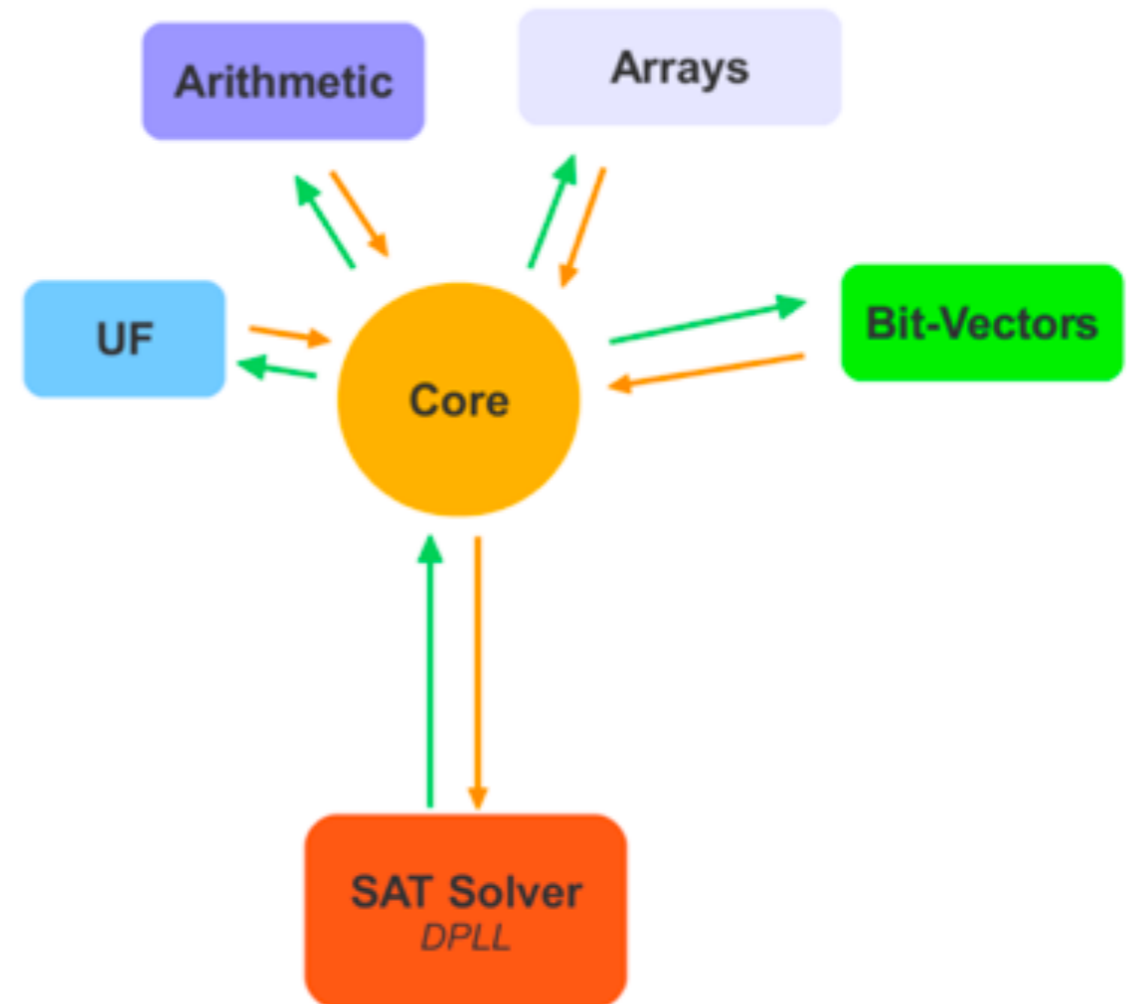
SMT: determine satisfiability of a formula with respect to some theory (e.g., theory of linear real arithmetic)



the virtues of laziness

eager approach: convert whole SMT formula to SAT formula immediately, then solve with SAT solver

lazy approach: use *theory solvers*, each specific to a particular theory

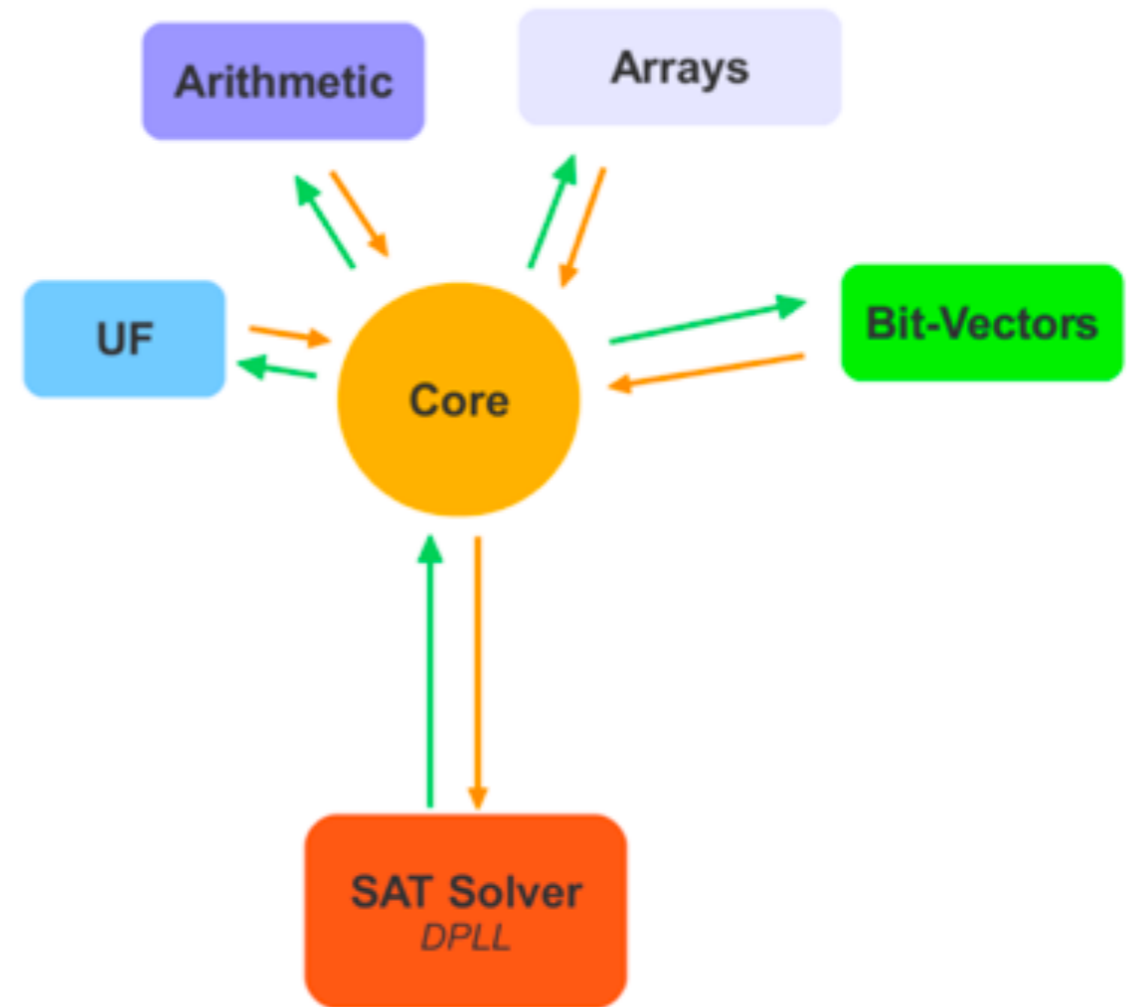


Source: fm.csl.sri.com/SSFT16/slides.pdf

the virtues of laziness

eager approach: convert whole SMT formula to SAT formula immediately, then solve with SAT solver

lazy approach: use *theory solvers*, each specific to a particular theory



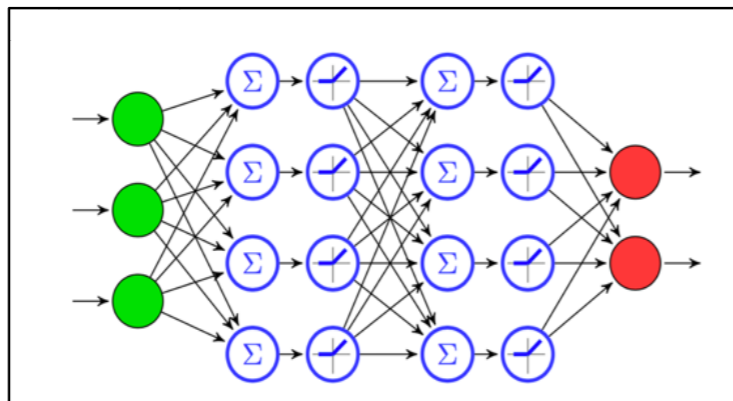
Source: fm.csl.sri.com/SSFT16/slides.pdf

key idea:
to exploit domain knowledge and unlock efficiency,
use a theory solver specifically for handling neural networks

lazily handling ReLU activations

[Katz et al., 2017]

Description of network



Property to be shown

“if intruder is near and approaching from left, network will advise ‘strong right’”

SMT formula

... $(n == 5 + m) \vee (p \wedge \neg q)$...

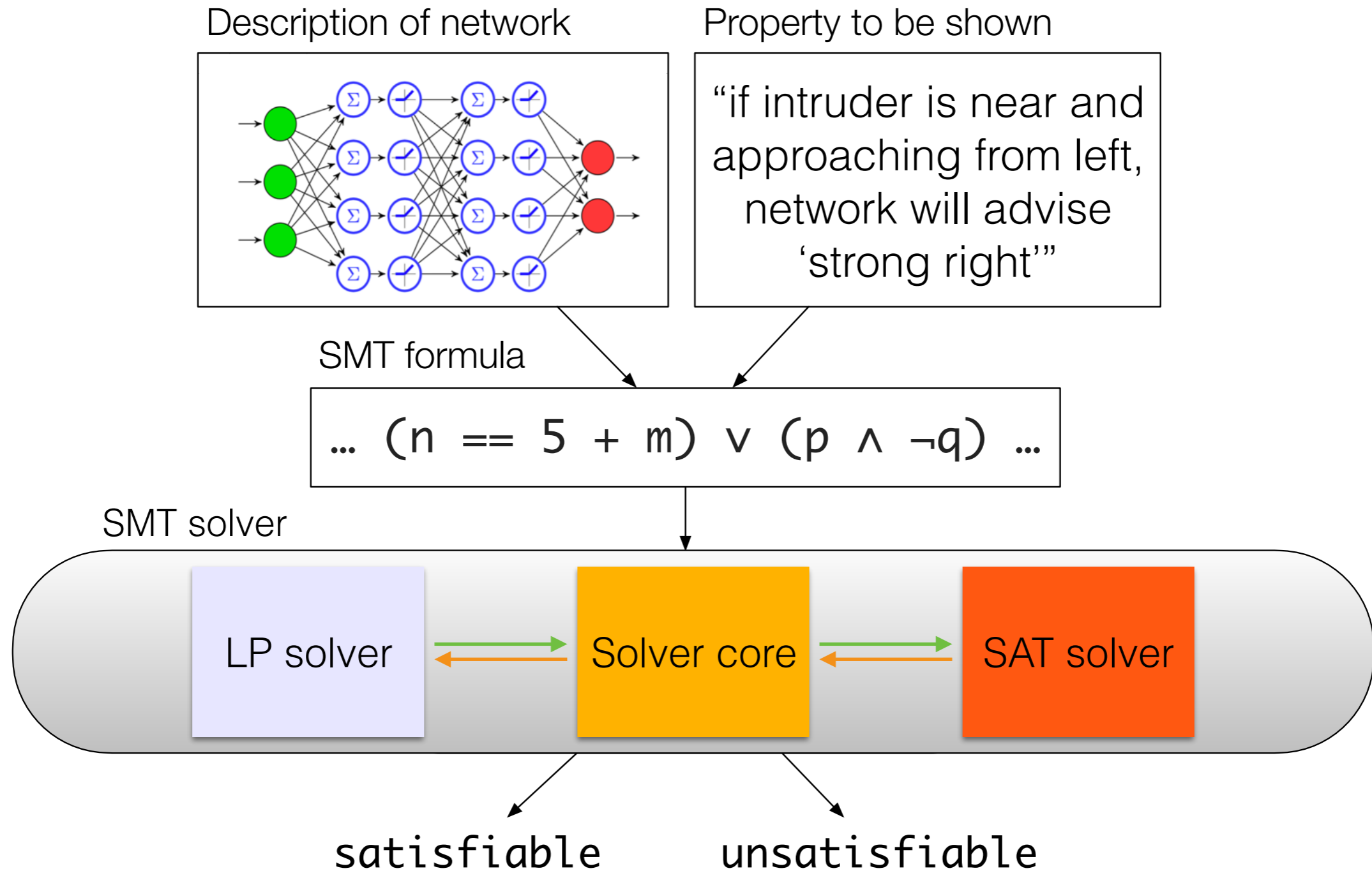
SMT Solver

satisfiable

unsatisfiable

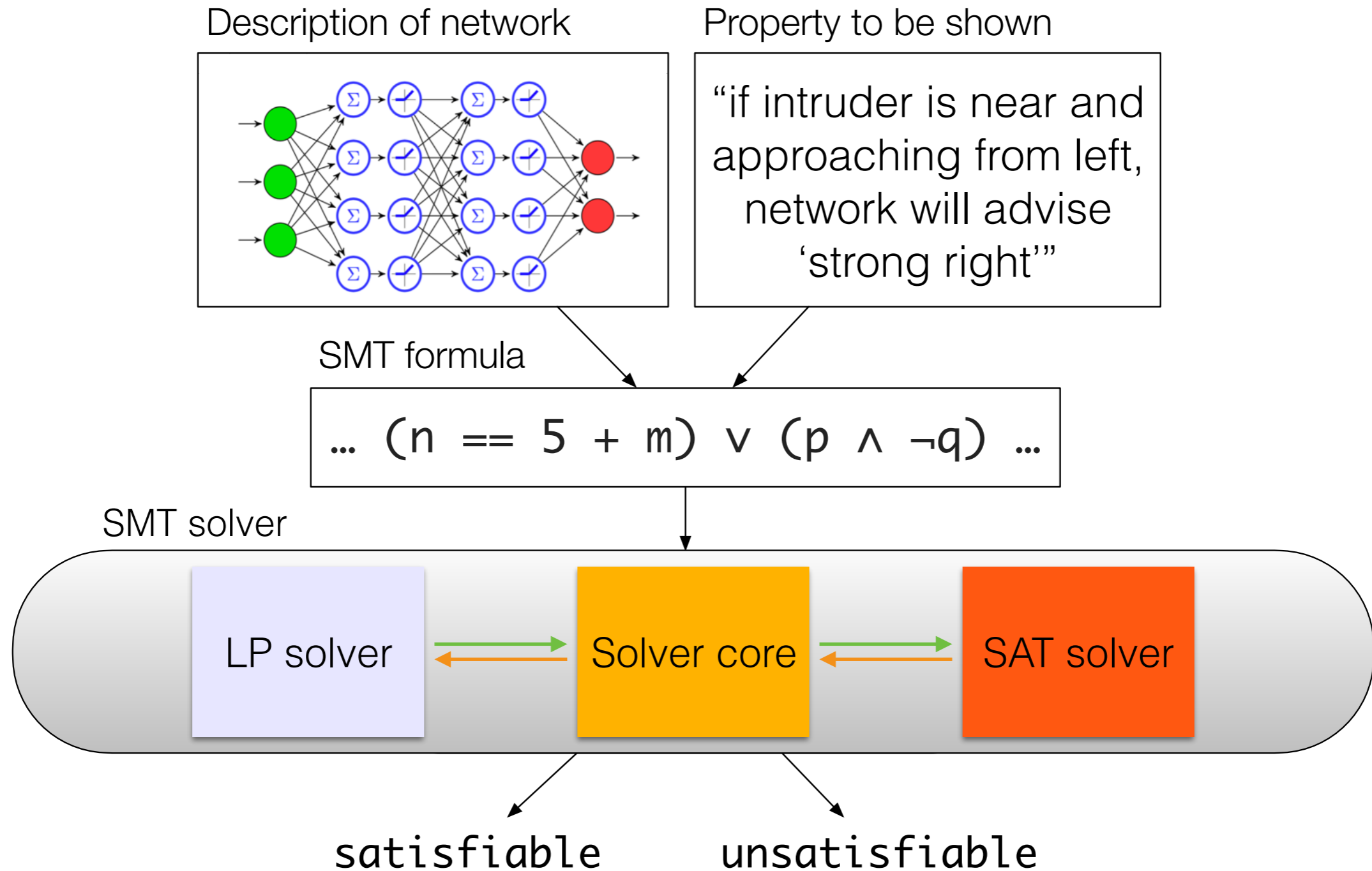
lazily handling ReLU activations

[Katz et al., 2017]



lazily handling ReLU activations

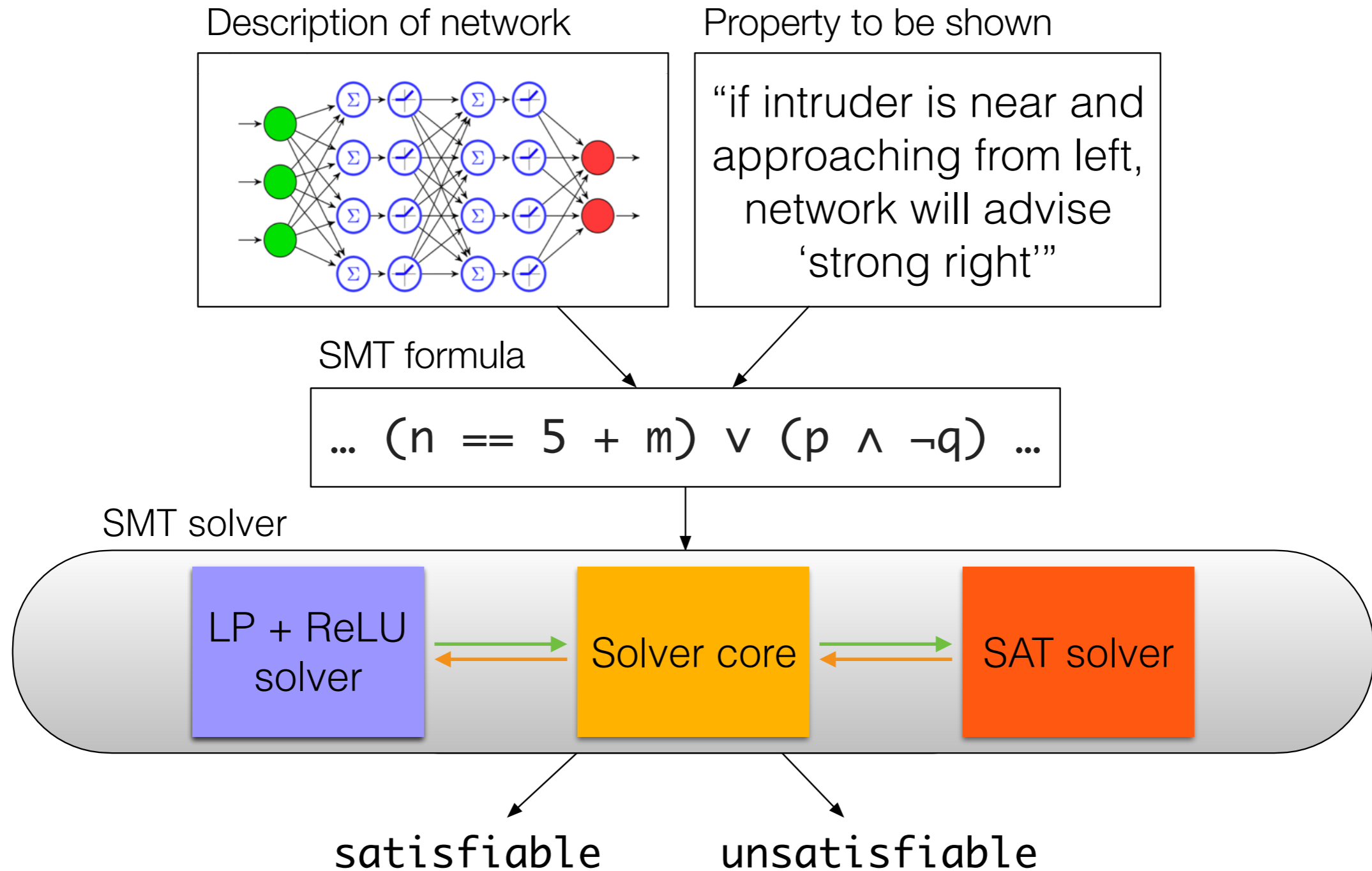
[Katz et al., 2017]



ReLU constraints like $y = \max(0, x)$ can only be encoded as disjunctions:
 $(x \geq 0 \wedge y = x) \vee (x < 0 \wedge y = 0)$

lazily handling ReLU activations

[Katz et al., 2017]



ReLU constraints like $y = \max(0, x)$ can only be encoded as disjunctions:
 $(x \geq 0 \wedge y = x) \vee (x < 0 \wedge y = 0)$

lazily handling ReLU activations

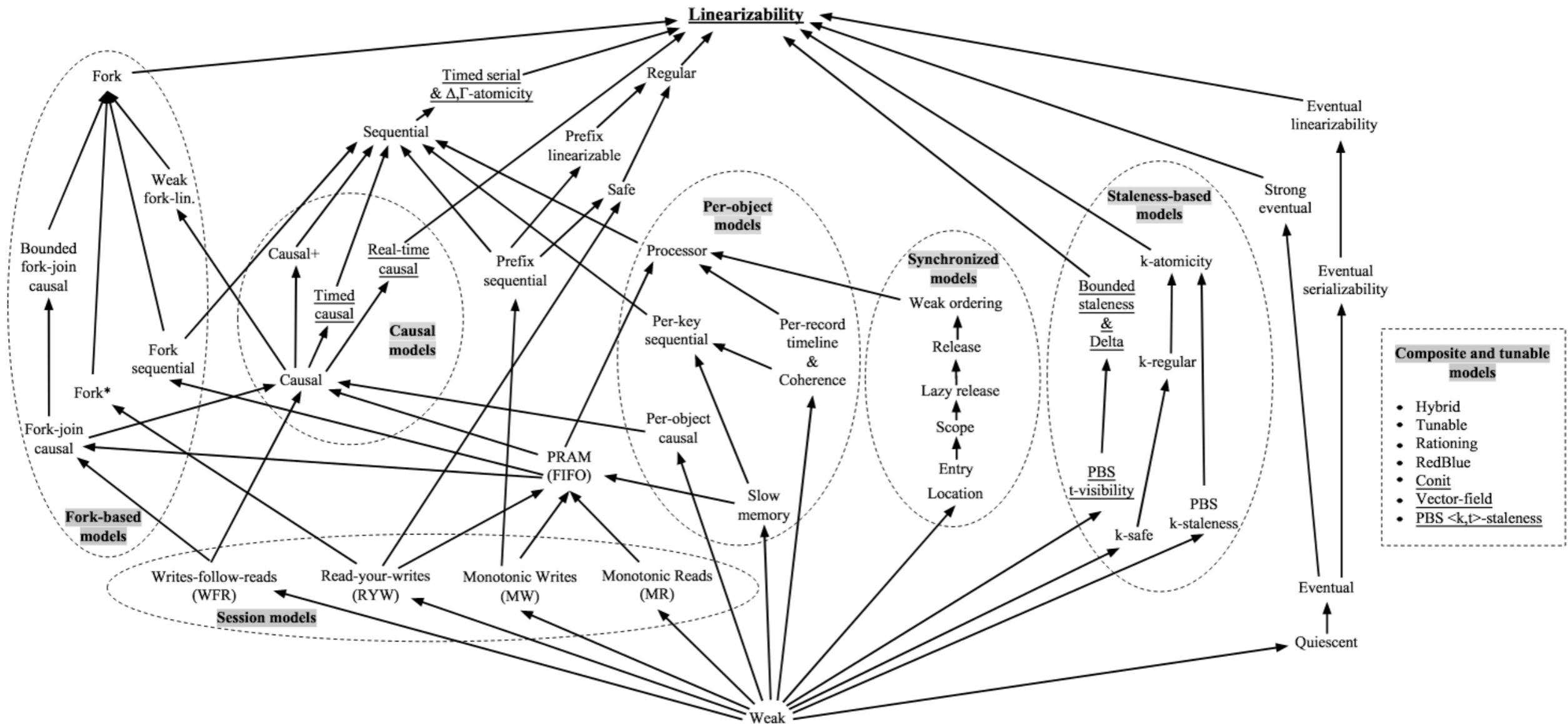
[Katz et al., 2017]

Property description	Does it hold?	Solver time	Max. ReLU split depth (out of 300)
“if intruder is directly ahead and is moving towards ownship, network will not advise COC”	✓	7.8h	22
“if intruder is near and approaching from left, network advises ‘strong right’”	✓	5.4h	46
“if intruder is sufficiently far away, network advises COC”	✓	50h	50
“for large vertical separation and previous ‘weak left’ advisory, network will either advise COC or continue advising ‘weak left’”	✗	11h	69

...and more

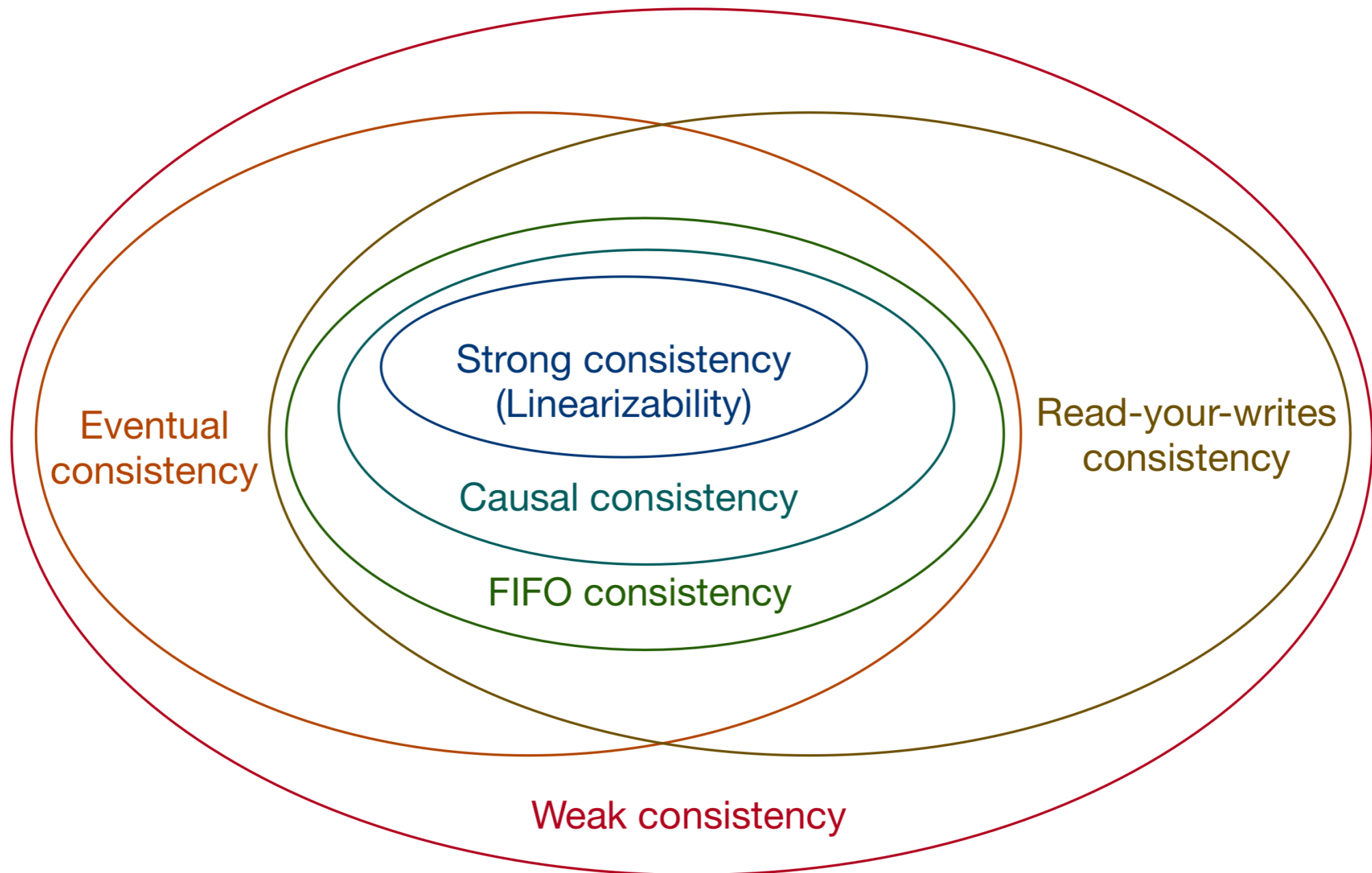
ReLU constraints like $y = \max(0, x)$ can only be encoded as disjunctions:
 $(x \geq 0 \wedge y = x) \vee (x < 0 \wedge y = 0)$

the distributed consistency model zoo



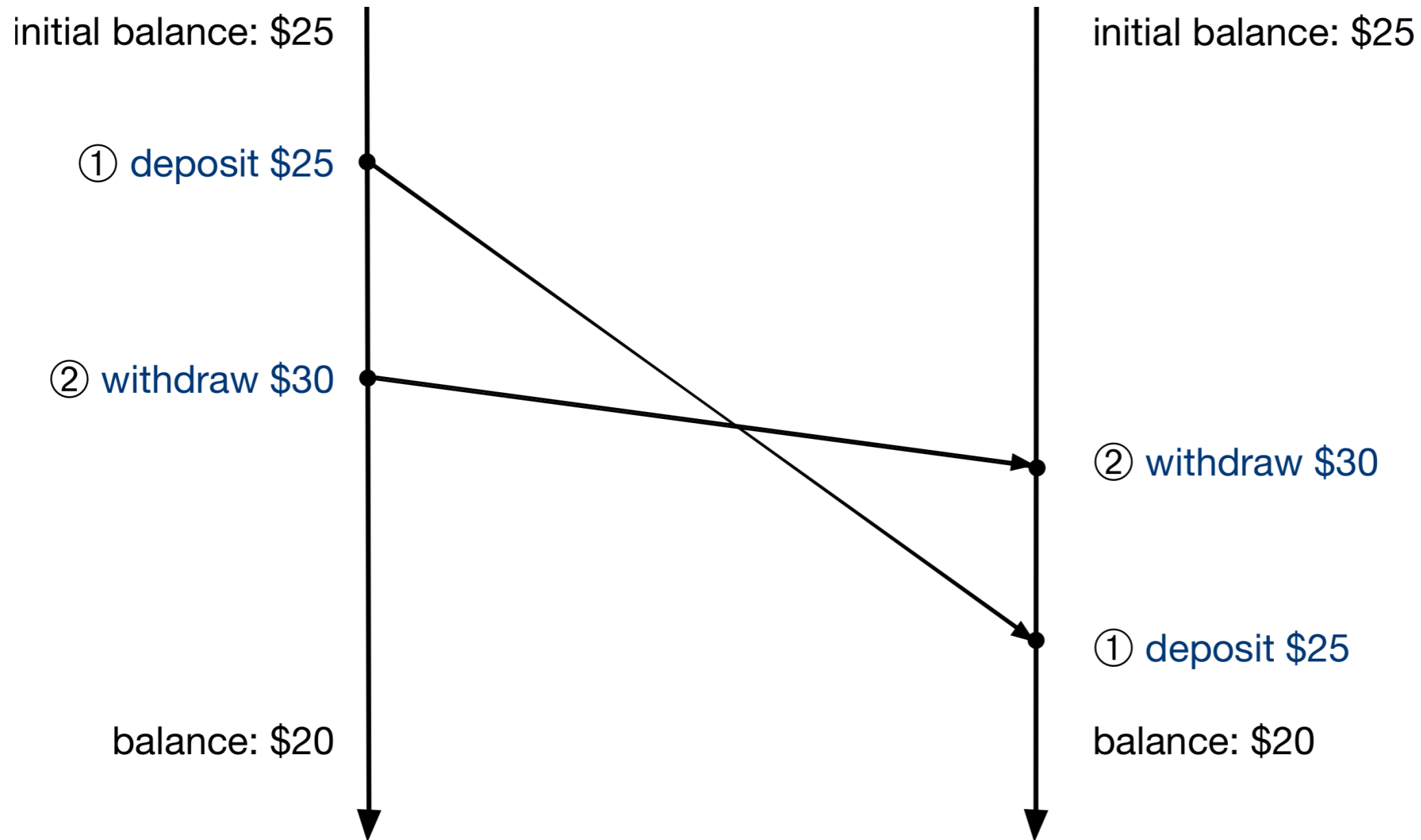
[Paolo Viotti and Marko Vukolić, [Consistency in Non-Transactional Distributed Storage Systems](#)]

the distributed consistency model zoo

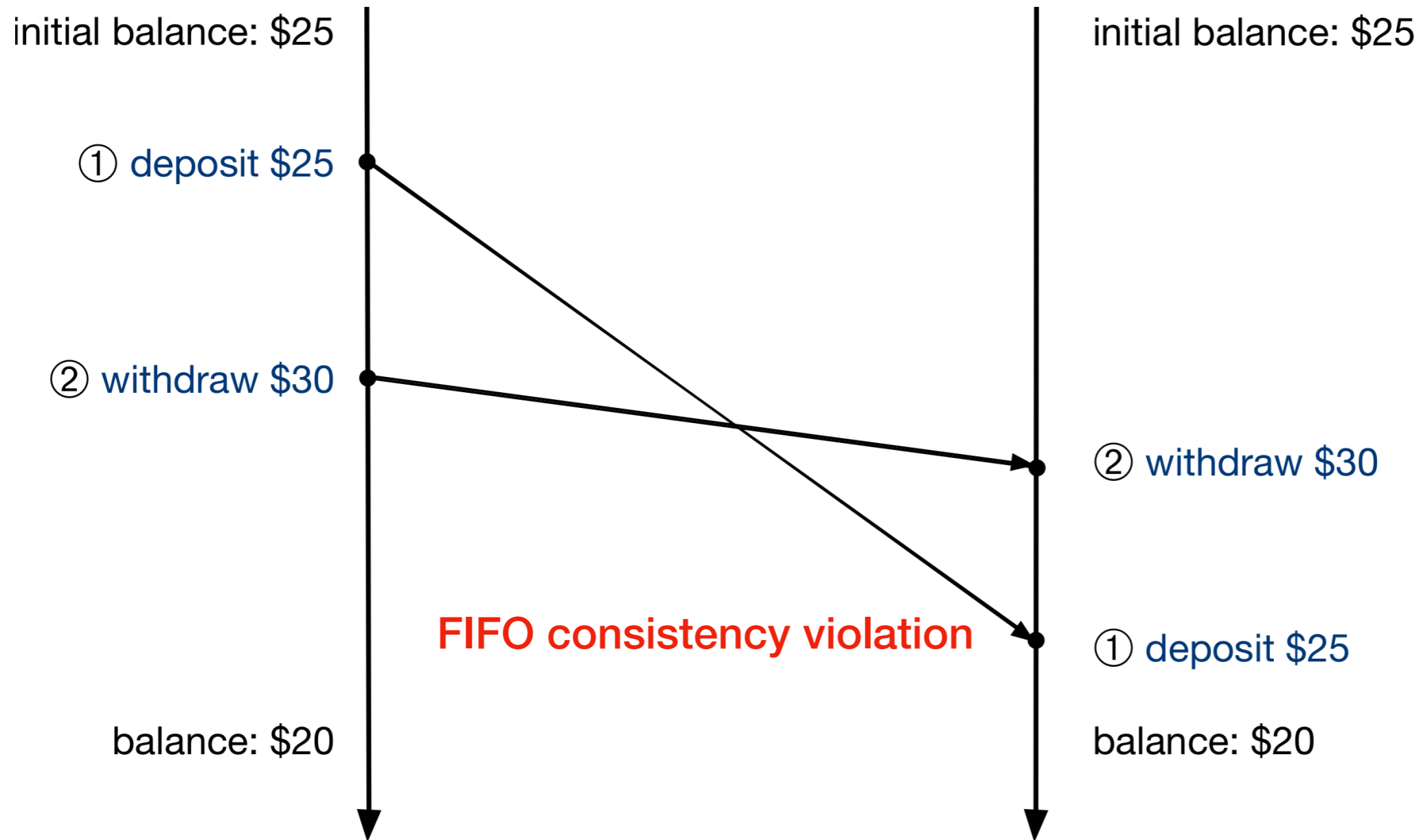


(smaller regions admit fewer executions)

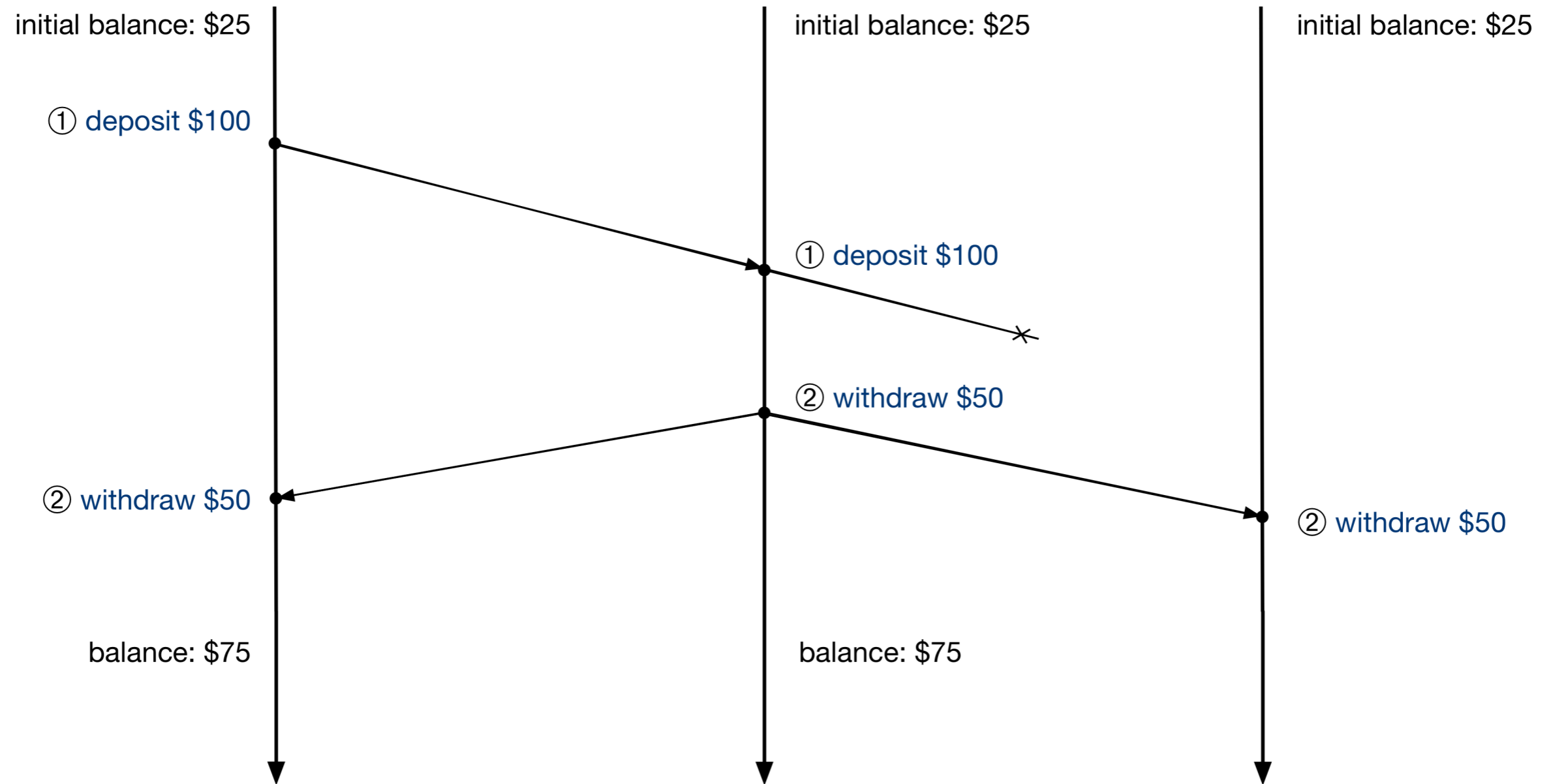
name that consistency bug/feature!



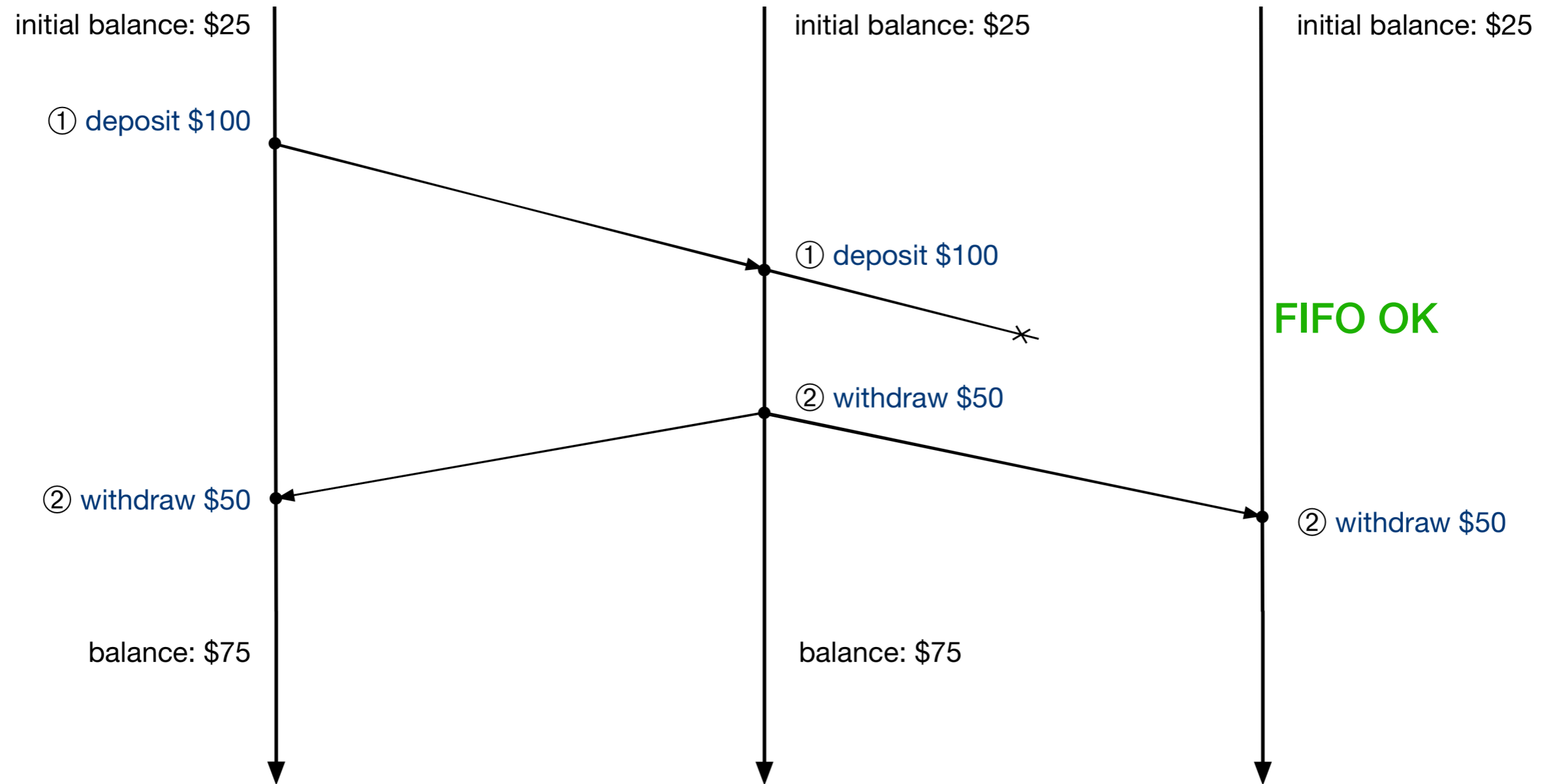
name that consistency bug/feature!



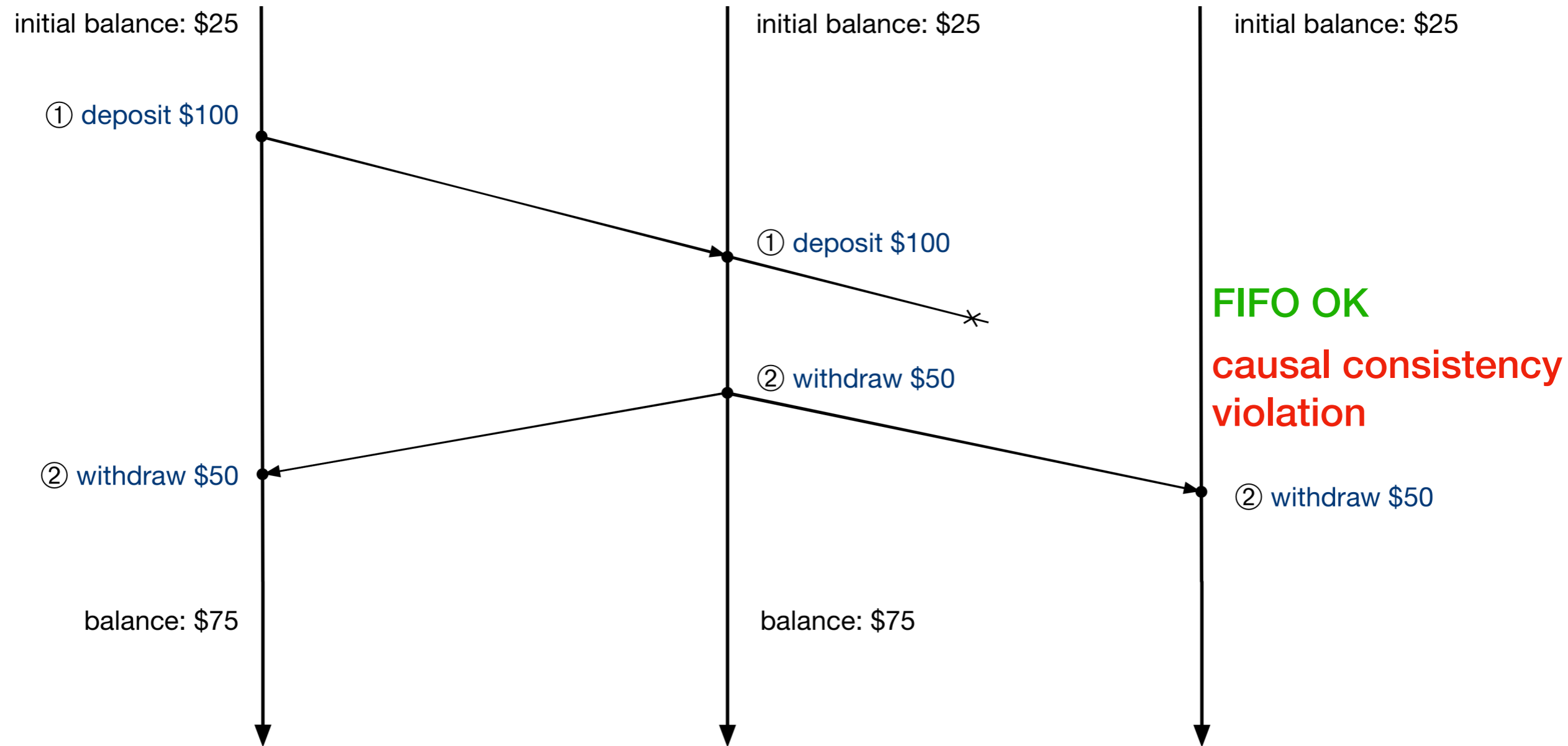
name that consistency bug/feature!



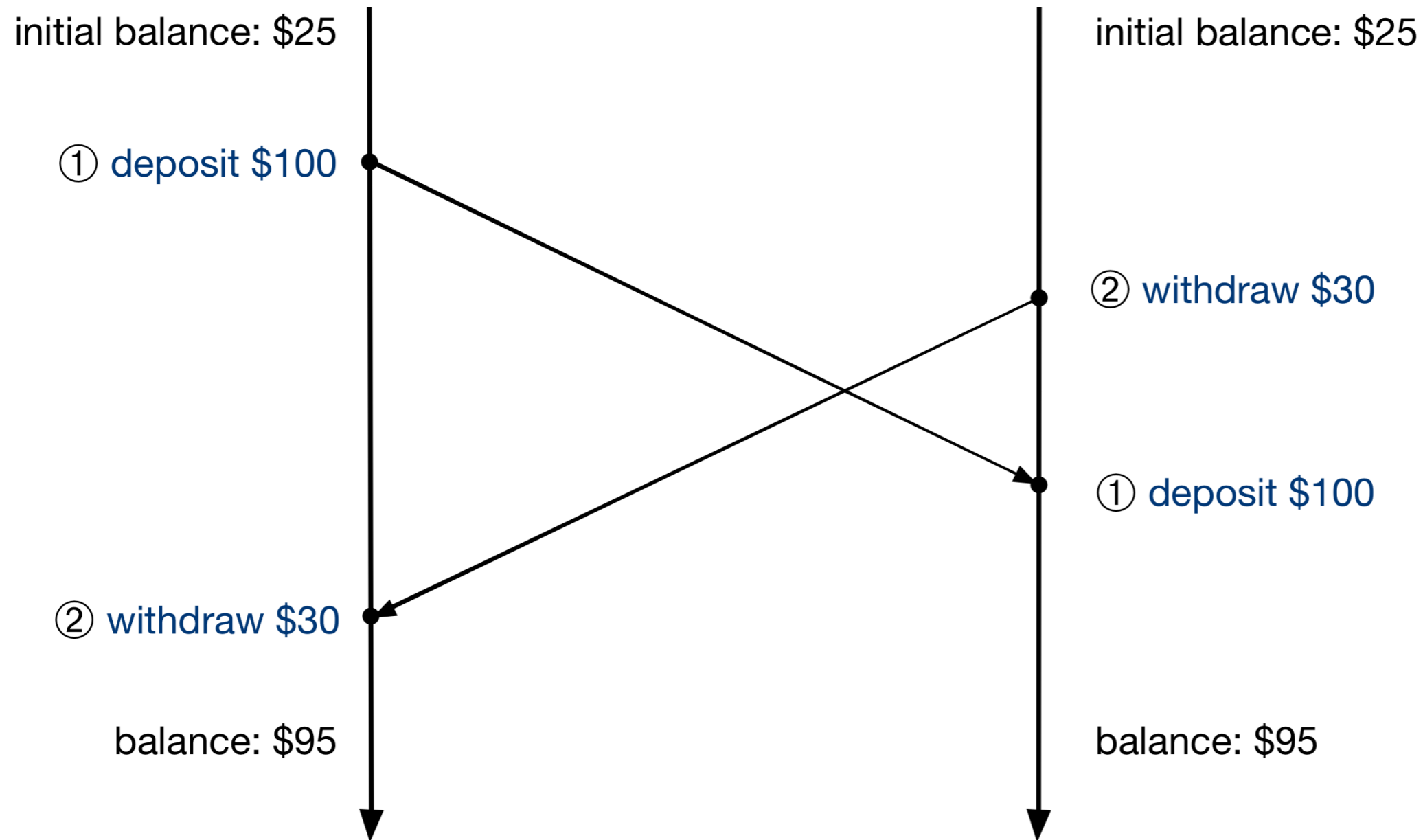
name that consistency bug/feature!



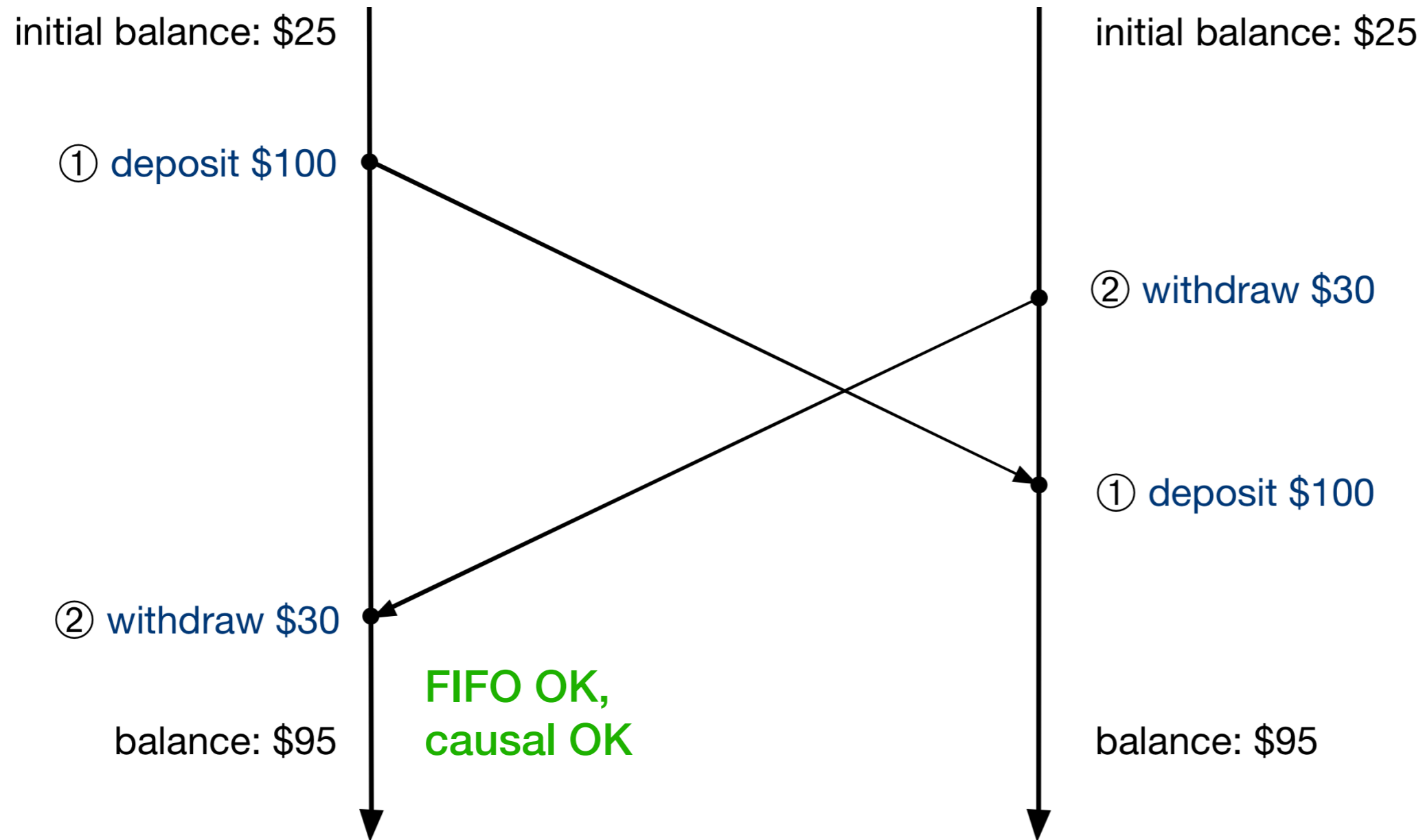
name that consistency bug/feature!



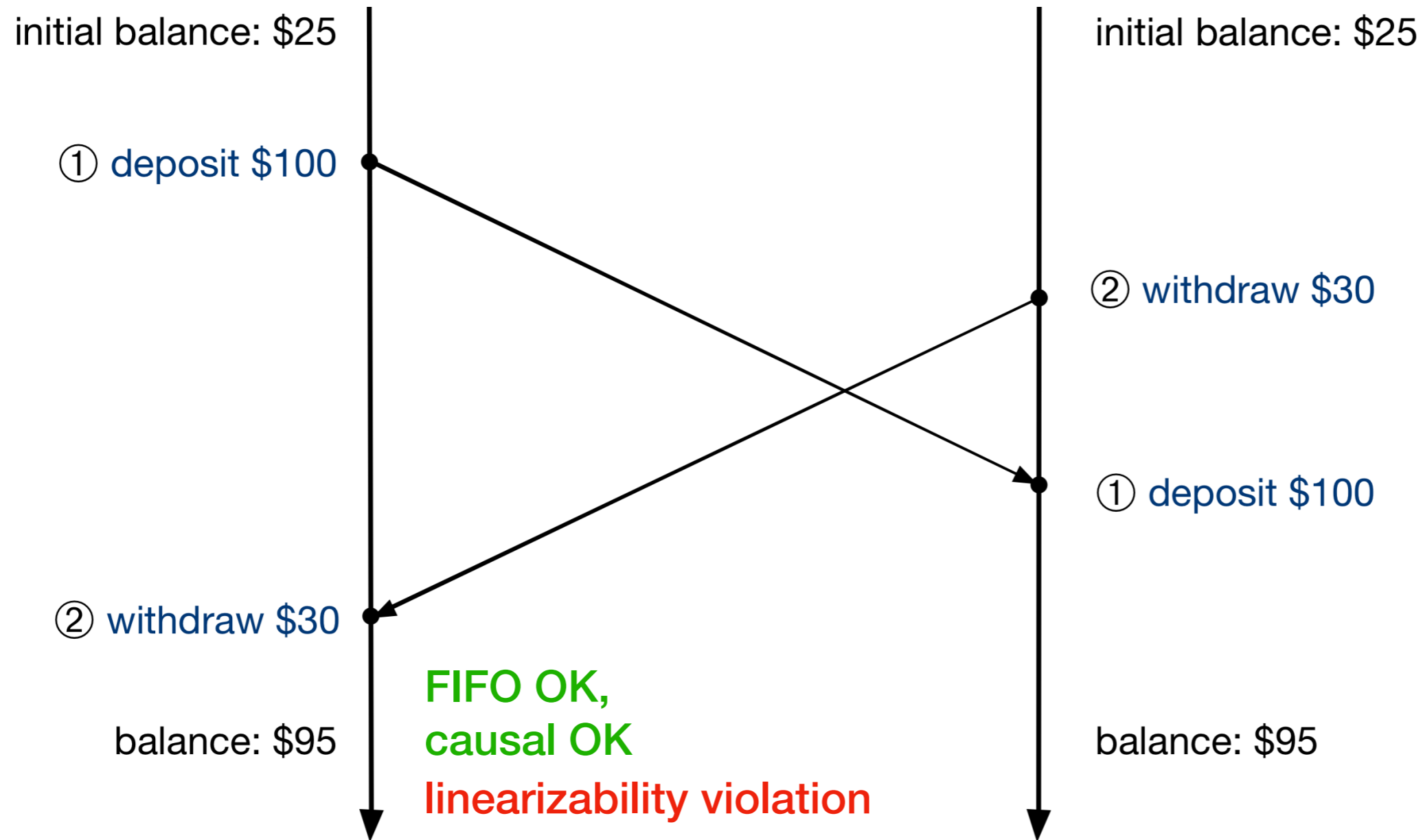
name that consistency bug/feature!



name that consistency bug/feature!



name that consistency bug/feature!



consistency contracts

[Sivaramakrishnan et al., 2015]

$$\begin{aligned}\psi &\in \text{Contract} & ::= & \forall(x : \tau).\psi \mid \forall x.\psi \mid \pi \\ \tau &\in \text{EffType} & ::= & \text{Op} \mid \tau \vee \tau \\ \pi &\in \text{Prop} & ::= & \text{true} \mid R(x, y) \mid \pi \vee \pi \\ & & & \mid \pi \wedge \pi \mid \pi \Rightarrow \pi \\ R &\in \text{Relation} & ::= & \text{vis} \mid \text{so} \mid \text{sameobj} \mid = \\ & & & \mid R \cup R \mid R \cap R \mid R^+\end{aligned}$$

$x, y, \hat{\eta} \in \text{EffVar}$

$\text{Op} \in \text{OpName}$

consistency contracts

[Sivaramakrishnan et al., 2015]

a contract is a first-order logic formula
universal quantification over EffVars allowed

ψ	\in	Contract	$::=$	$\forall(x : \tau).\psi \mid \forall x.\psi \mid \pi$
τ	\in	EffType	$::=$	$\text{Op} \mid \tau \vee \tau$
π	\in	Prop	$::=$	$\text{true} \mid R(x, y) \mid \pi \vee \pi$ $\mid \pi \wedge \pi \mid \pi \Rightarrow \pi$
R	\in	Relation	$::=$	$\text{vis} \mid \text{so} \mid \text{sameobj} \mid =$ $\mid R \cup R \mid R \cap R \mid R^+$

$x, y, \hat{\eta} \in \text{EffVar}$

$\text{Op} \in \text{OpName}$

consistency contracts

[Sivaramakrishnan et al., 2015]

$\psi \in \text{Contract} ::= \forall(x : \tau).\psi \mid \forall x.\psi \mid \pi$ a contract is a first-order logic formula
universal quantification over EffVars allowed

$\tau \in \text{EffType} ::= \text{Op} \mid \tau \vee \tau$

$\pi \in \text{Prop} ::= \text{true} \mid R(x, y) \mid \pi \vee \pi$
 $\mid \pi \wedge \pi \mid \pi \Rightarrow \pi$ see: Sebastian Burckhardt's book

$R \in \text{Relation} ::= \text{vis} \mid \text{so} \mid \text{sameobj} \mid =$
 $\mid R \cup R \mid R \cap R \mid R^+$

$x, y, \hat{\eta} \in \text{EffVar}$

$\text{Op} \in \text{OpName}$

consistency contracts

[Sivaramakrishnan et al., 2015]

$\psi \in \text{Contract} ::= \forall(x : \tau).\psi \mid \forall x.\psi \mid \pi$ a contract is a first-order logic formula
universal quantification over EffVars allowed

$\tau \in \text{EffType} ::= \text{Op} \mid \tau \vee \tau$

$\pi \in \text{Prop} ::= \text{true} \mid R(x, y) \mid \pi \vee \pi$
 $\mid \pi \wedge \pi \mid \pi \Rightarrow \pi$ see: Sebastian Burckhardt's book

$R \in \text{Relation} ::= \text{vis} \mid \text{so} \mid \text{sameobj} \mid =$
 $\mid R \cup R \mid R \cap R \mid R^+$

$x, y, \hat{\eta} \in \text{EffVar}$

$\text{Op} \in \text{OpName}$

example contracts for bank account operations

($\hat{\eta}$ is the current operation/effect)

for withdraw: $\forall(a : \text{withdraw}).$
 $\text{sameobj}(a, \hat{\eta}) \Rightarrow a = \hat{\eta} \vee \text{vis}(a, \hat{\eta}) \vee \text{vis}(\hat{\eta}, a)$

for getBalance: $\forall(a : \text{deposit}), (b : \text{withdraw}), (c : \text{deposit} \vee \text{withdraw}).$
 $(\text{vis}(a, b) \wedge \text{vis}(b, \hat{\eta}) \Rightarrow \text{vis}(a, \hat{\eta}))$
 $\wedge ((\text{so} \cap \text{sameobj})(c, \hat{\eta}) \Rightarrow \text{vis}(c, \hat{\eta}))$

consistency contracts

[Sivaramakrishnan et al., 2015]

$\psi \in \text{Contract} ::= \forall(x : \tau).\psi \mid \forall x.\psi \mid \pi$ a contract is a first-order logic formula
universal quantification over EffVars allowed

$\tau \in \text{EffType} ::= \text{Op} \mid \tau \vee \tau$

$\pi \in \text{Prop} ::= \text{true} \mid R(x, y) \mid \pi \vee \pi$
 $\mid \pi \wedge \pi \mid \pi \Rightarrow \pi$ see: Sebastian Burckhardt's book

$R \in \text{Relation} ::= \text{vis} \mid \text{so} \mid \text{sameobj} \mid =$
 $\mid R \cup R \mid R \cap R \mid R^+$

$x, y, \hat{\eta} \in \text{EffVar}$

$\text{Op} \in \text{OpName}$

example contracts for bank account operations

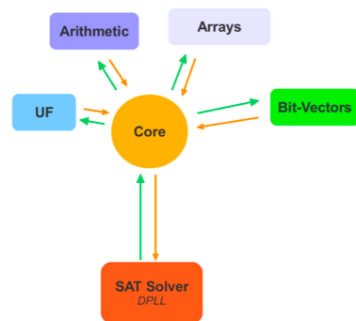
($\hat{\eta}$ is the current operation/effect)

for **withdraw**: $\forall(a : \text{withdraw}).$
 $\text{sameobj}(a, \hat{\eta}) \Rightarrow a = \hat{\eta} \vee \text{vis}(a, \hat{\eta}) \vee \text{vis}(\hat{\eta}, a)$

for **getBalance**: $\forall(a : \text{deposit}), (b : \text{withdraw}), (c : \text{deposit} \vee \text{withdraw}).$
 $(\text{vis}(a, b) \wedge \text{vis}(b, \hat{\eta}) \Rightarrow \text{vis}(a, \hat{\eta}))$
 $\wedge ((\text{so} \cap \text{sameobj})(c, \hat{\eta}) \Rightarrow \text{vis}(c, \hat{\eta}))$

(what's the contract for **deposit**? why?)

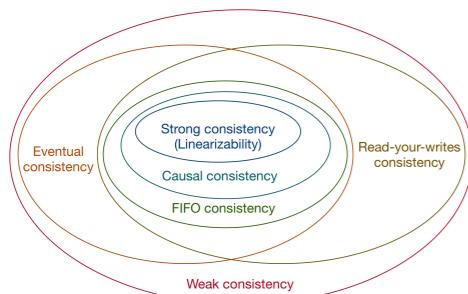
takeaways



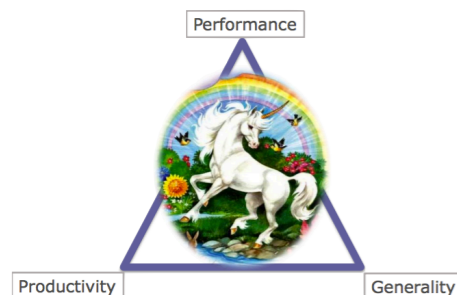
specialized theory solvers are an old idea

domain-specific solvers = high-level theory solvers

one domain of interest: *consistency-aware* solvers
existing contract languages a possible starting point



for now, PL folk can bravely dig into solver internals



in the long run: democratize solver development!
“Delite for domain-specific solvers”