Reduction Techniques for Parameterized Model Checking and Synthesis of Fault-Tolerant Distributed Algorithms

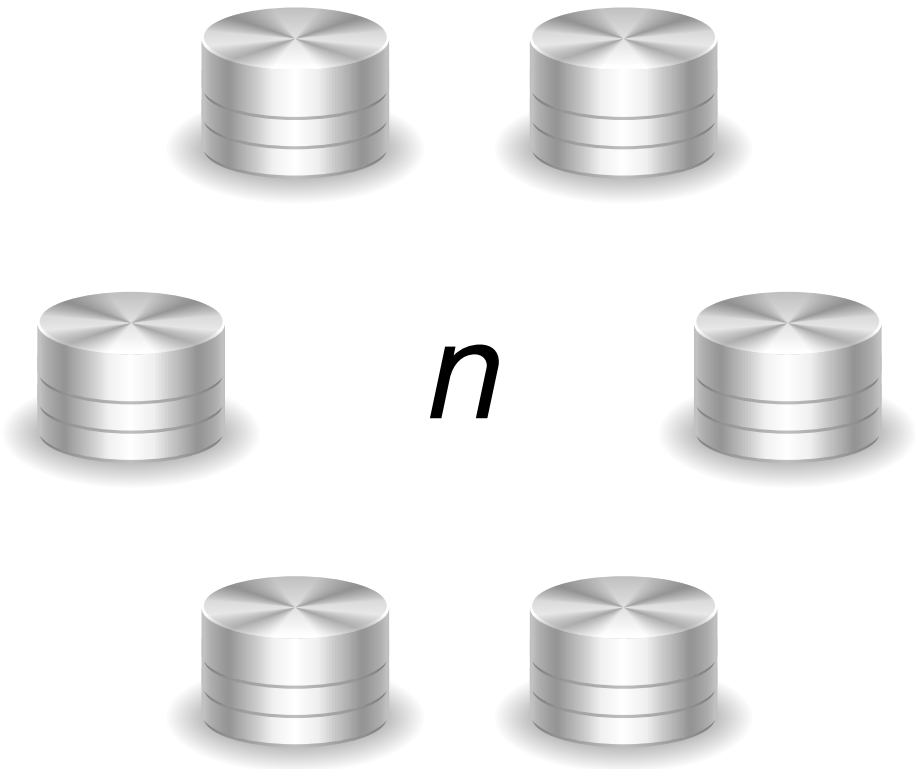
Marijana Lazić

PhD defense, Vienna, June 14, 2019

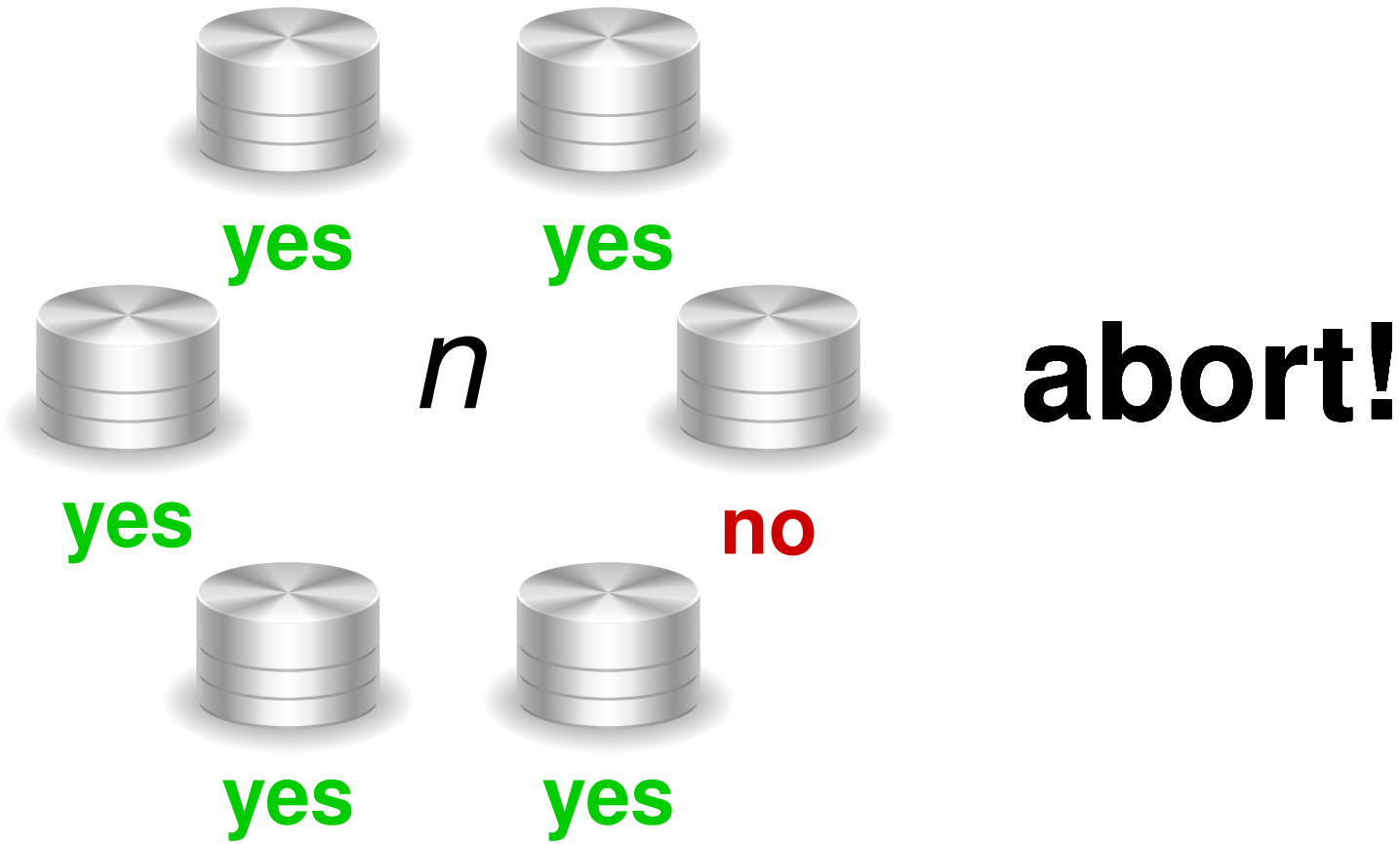
Distributed systems are everywhere

distributed algorithms at the core

Atomic commit in a distributed database

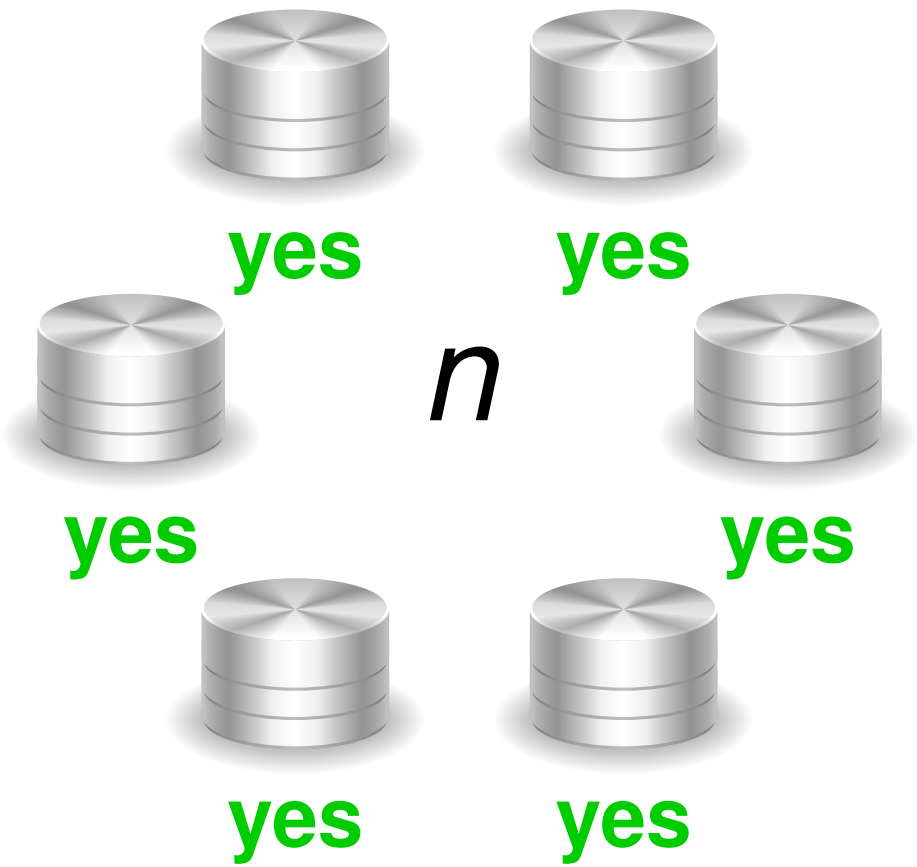


Atomic commit in a distributed database



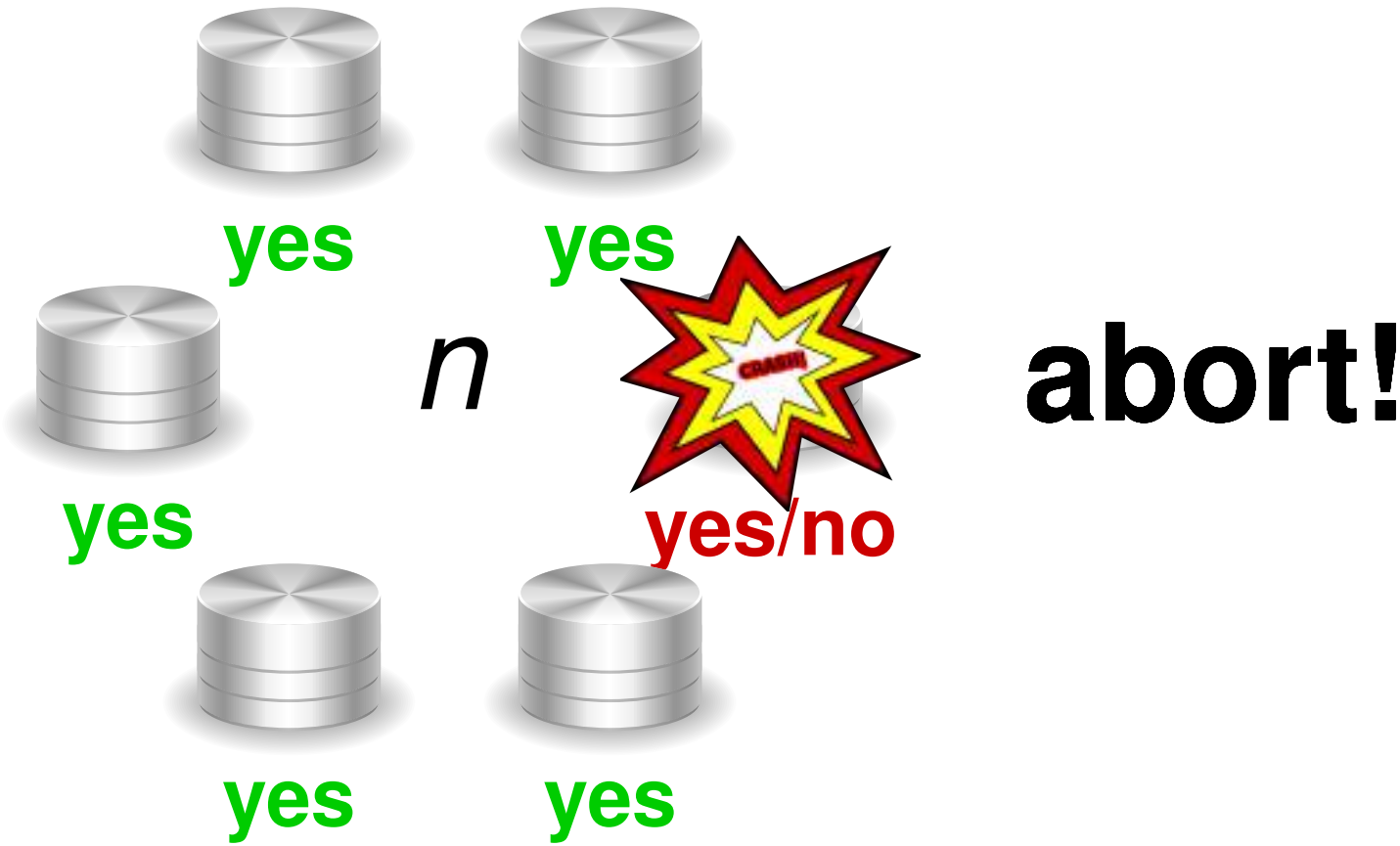
abort!

Atomic commit in a distributed database

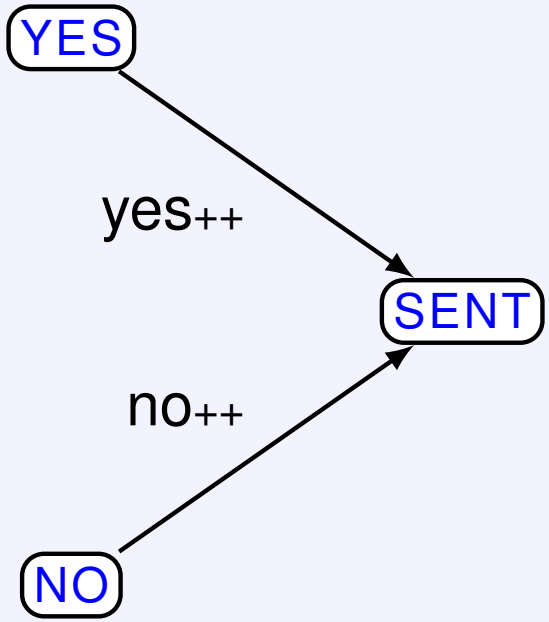


commit!

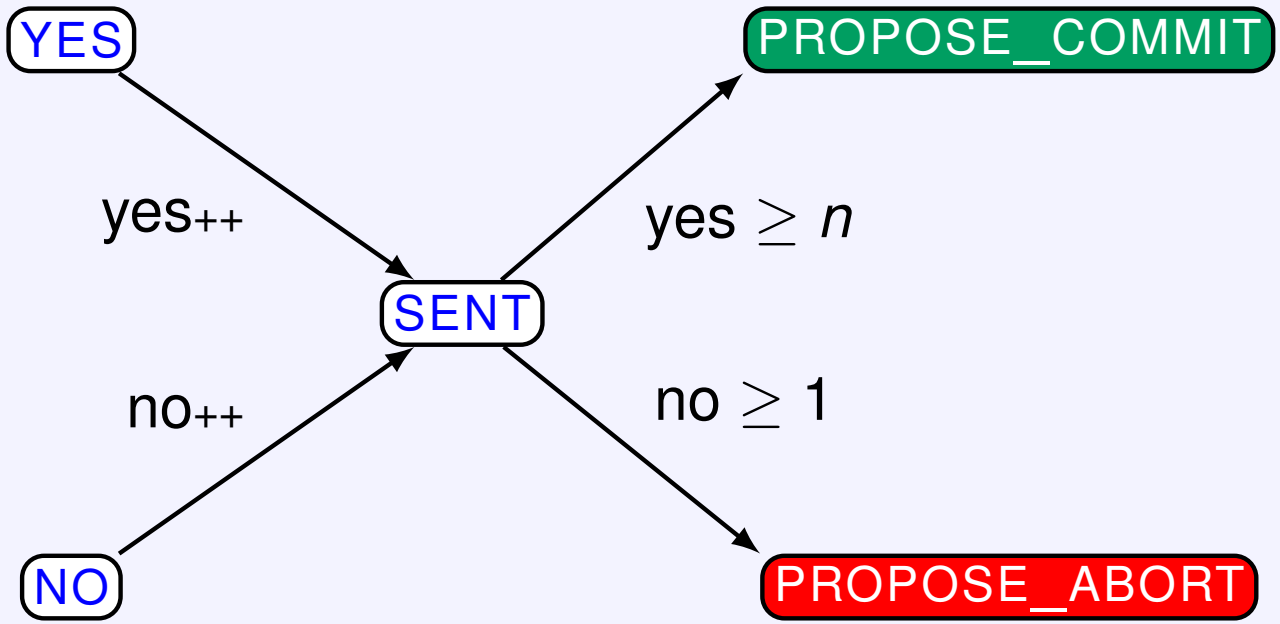
Atomic commit in a distributed database



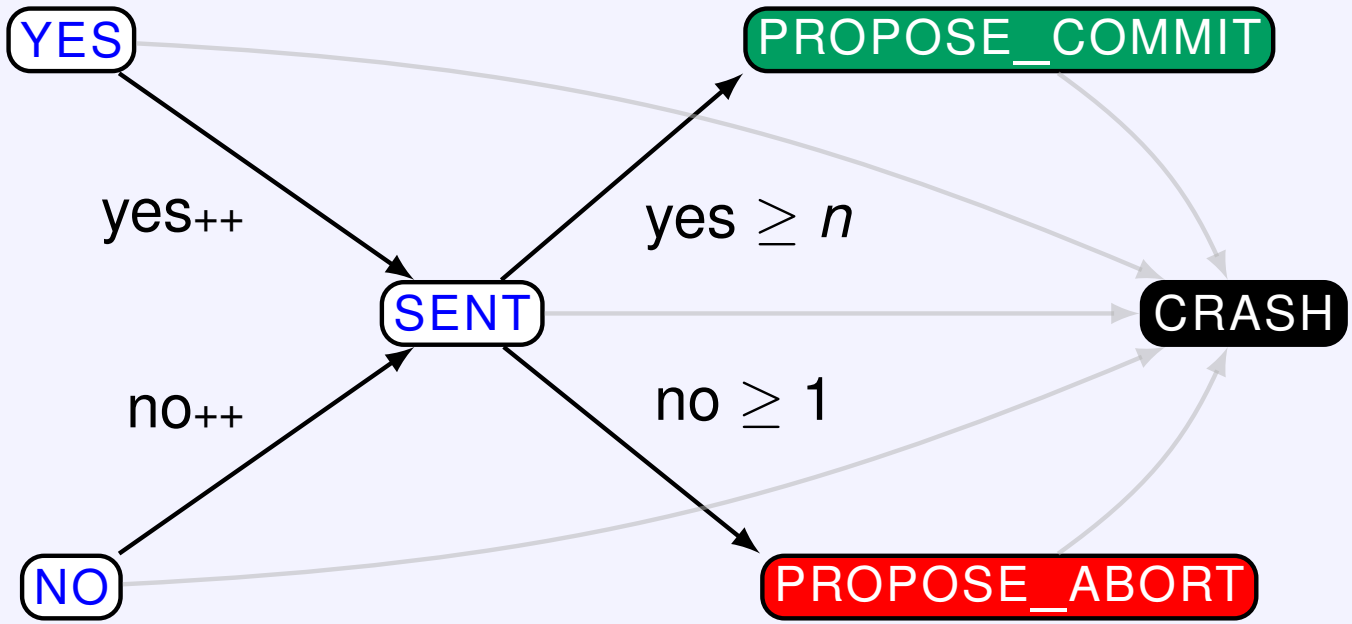
Non-blocking atomic commit [Raynal'97, Guerraoui'01]



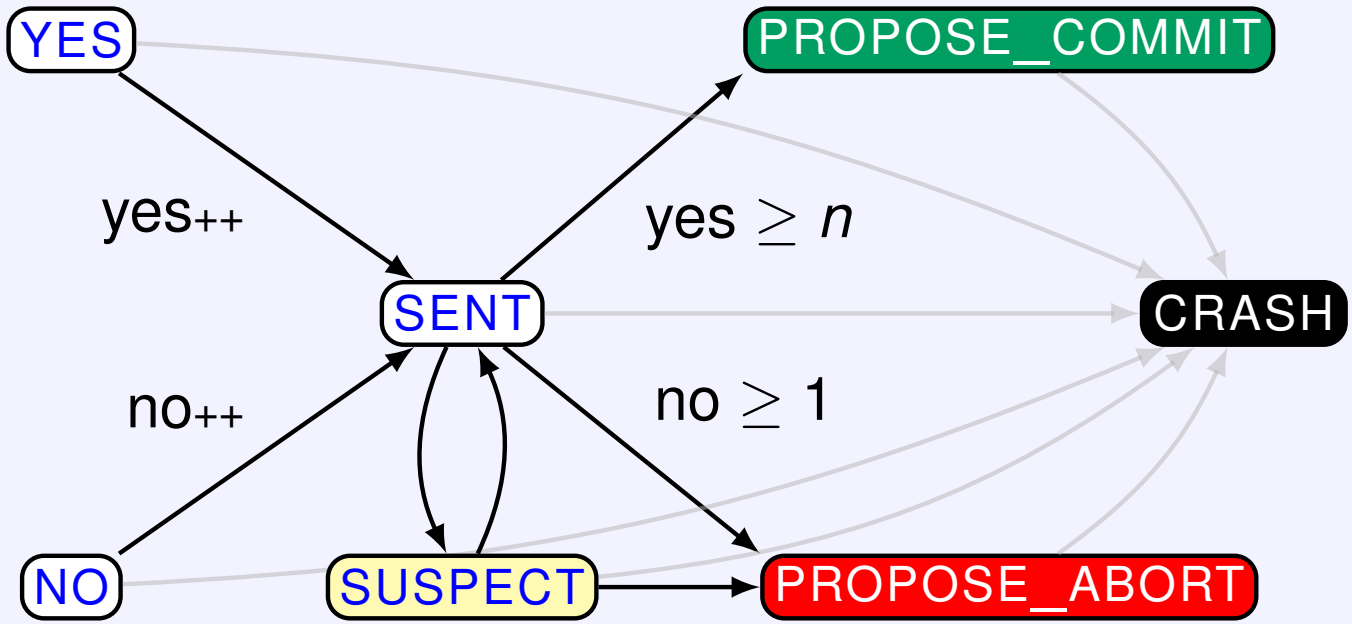
Non-blocking atomic commit [Raynal'97, Guerraoui'01]



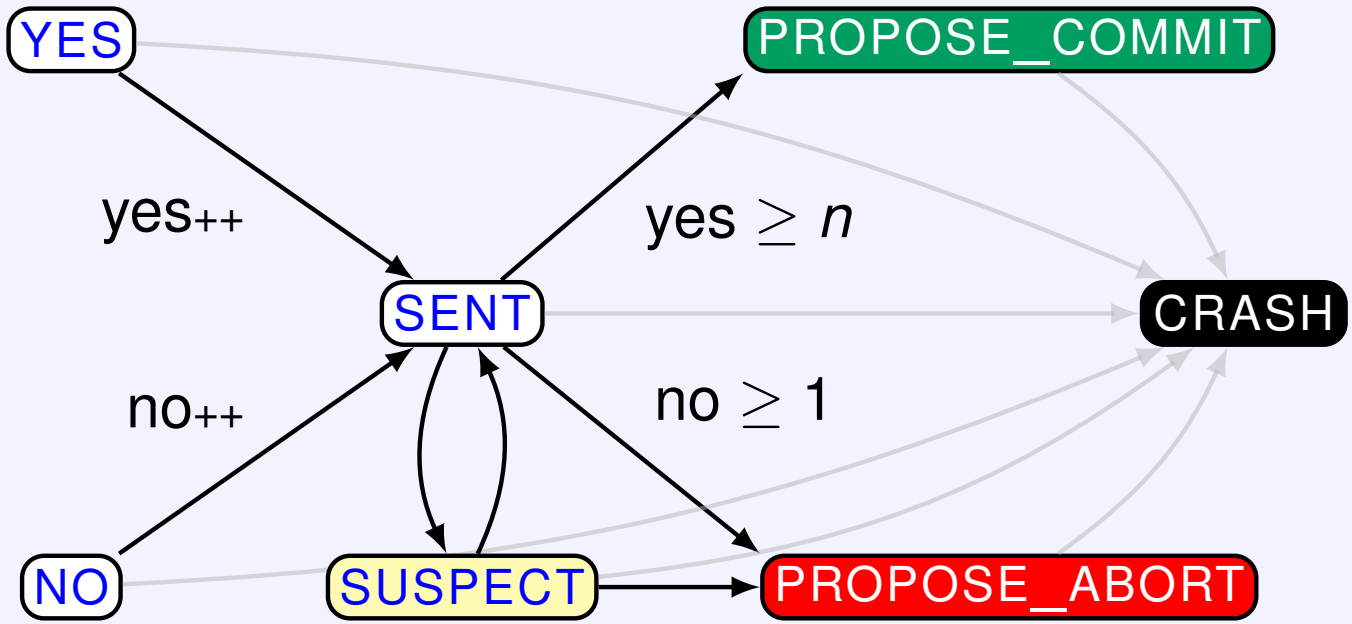
Non-blocking atomic commit [Raynal'97, Guerraoui'01]



Non-blocking atomic commit [Raynal'97, Guerraoui'01]



Non-blocking atomic commit [Raynal'97, Guerraoui'01]



Commit-Validity: if a process proposes commit, then all process voted Yes

Non-Triviality: if all processes vote Yes,
and if no process crashes and no process suspects a crash,
then eventually every process proposes commit

(and more...)

Fault-tolerant systems

safety critical systems: cars, planes, etc.

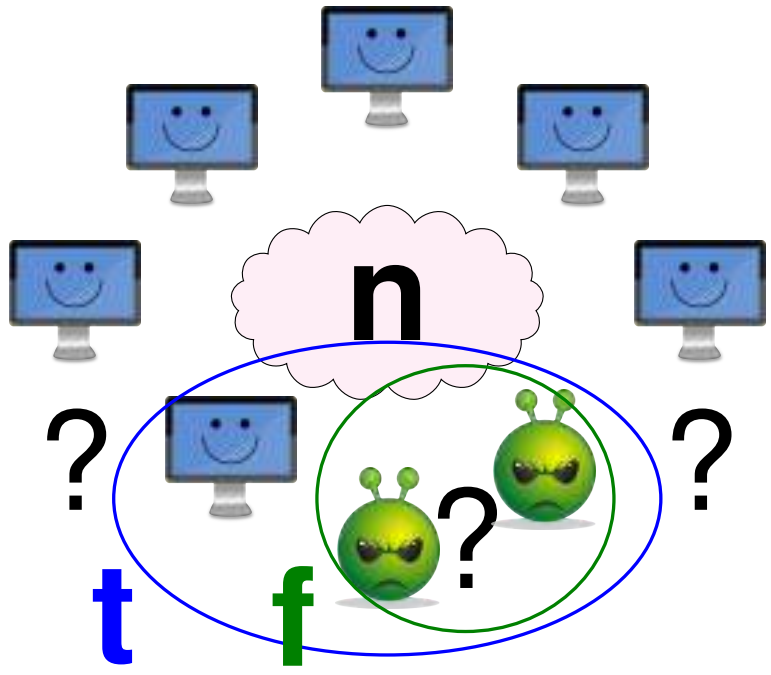
- rare but dangerous faults
- 3 to 7 processes

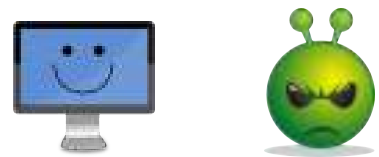
datacenters: thousands of computers

- faults happen every day
- 100–10,000 processes

Parameterized model checking and synthesis

Parameterized Verification



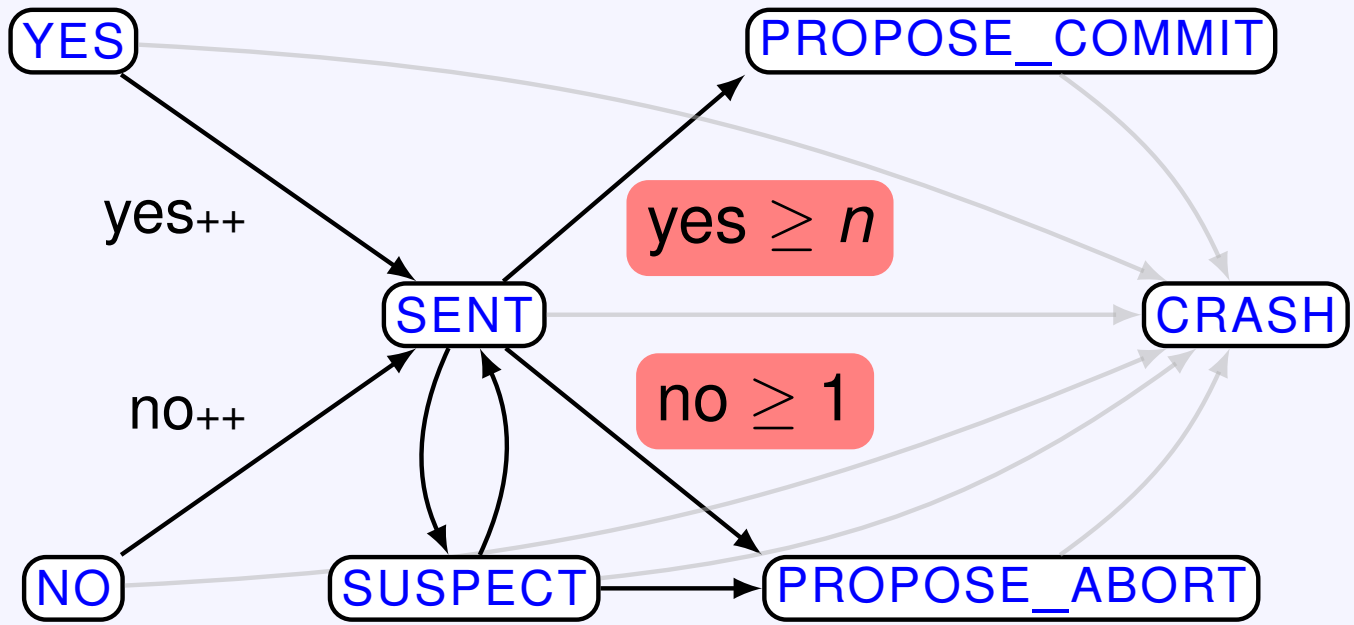


correct or faulty

$\forall n, t, f$ with $n > 3t$ and $t \geq f \geq 0$.

$$\underbrace{P(n,t) \parallel P(n,t) \parallel \ldots \parallel P(n,t)}_{\text{n-f correct}} \parallel \underbrace{\text{Faulty} \parallel \ldots \parallel \text{Faulty}}_{\text{f faulty}} \models \textit{Safety} \wedge \textit{Liveness}$$

In Focus: Threshold-based algorithms



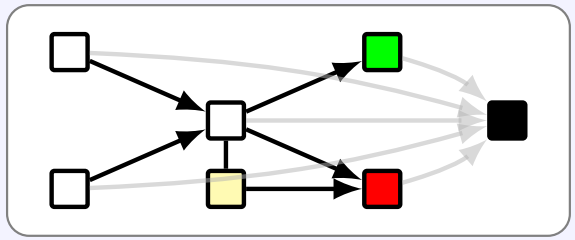
yes $\geq$ *n* and *no* $\geq$ 1 are threshold guards

More threshold guards...

Reliable broadcast	$x \geq t + 1$ and $x \geq n - t$	[Chandra, Toueg'96]
Byzantine agreement	$x \geq \lceil \frac{n}{2} \rceil + 1$	[Bracha, Toueg'85]
Condition-based consensus	$x \geq n - t$ and $x \geq \lceil \frac{n}{2} \rceil + 1$	[Mostéfaoui, Mourgaya, Parvedy, Raynal'03]
Consensus in one communication step	$x \geq n - t$ and $x \geq n - 2t$	[Brasileiro, Greve, Mostéfaoui, Raynal'03]
Byzantine one-step consensus	$x \geq \lceil \frac{n+3t}{2} \rceil + 1$	[Song, van Renesse'08]

In general, there is a resilience condition, e.g., $n > 3t$, $n > 7t$

Problems addressed in this thesis

$\forall n.$ n copies of  $\models \varphi$

VERIFICATION
Is an algorithm correct?

FMSD'17

POPL'17

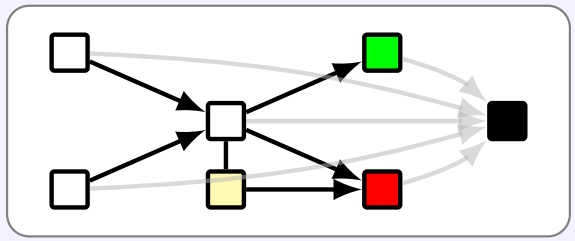
SYNTHESIS
Which thresholds make an algorithm correct?

OPODIS'17

RANDOMIZATION
Is a randomized algorithm correct?

CONCUR'19

Problems addressed in this thesis

$\forall n.$ n copies of  $\models \varphi$

VERIFICATION
Is an algorithm correct?

FMSD'17

POPL'17

SYNTHESIS

RANDOMIZATION

Can safety be violated?

Checking reachability:

(I) Counting processes

(II) Acceleration

(III) Bounded model checking

[Konnov, Lazić, Veith, Widder: FMSD'17]

(I) Counting processes

Threshold guards (e.g., yes $\geq n$) do not make use of process ids

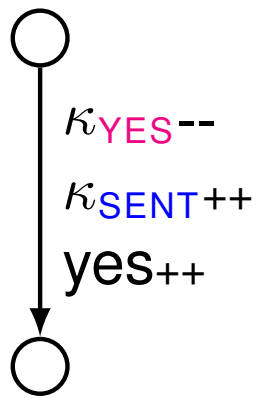
These transitions are indistinguishable:

A transition by a single process:

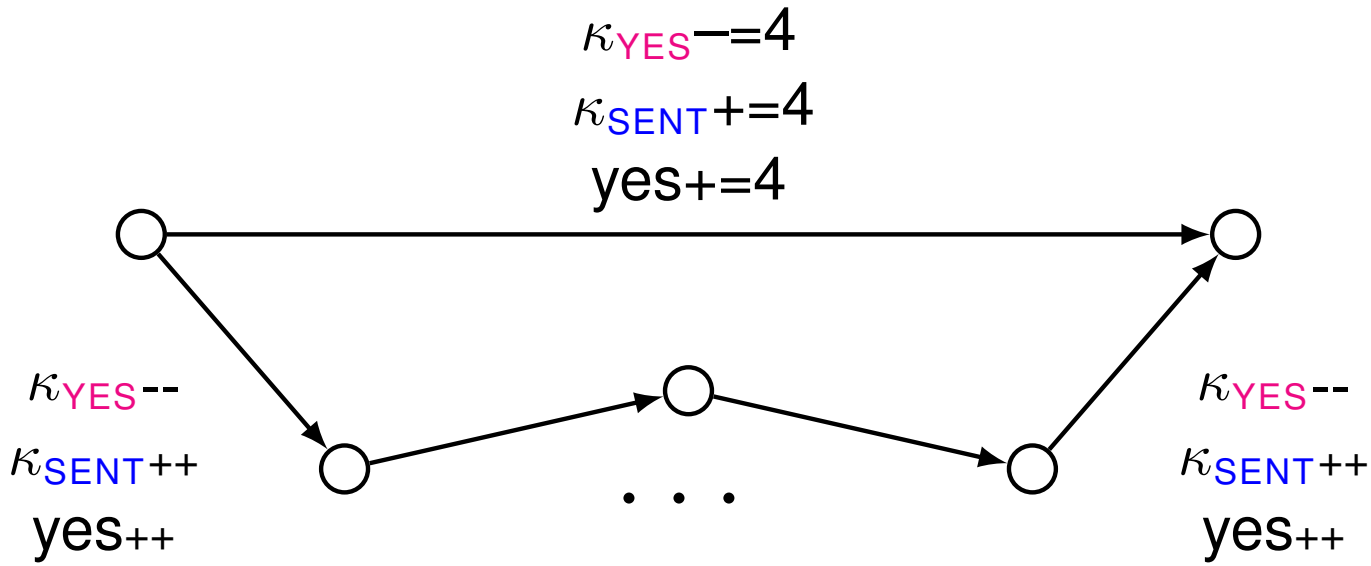
$$\left\{\kappa_{\text{YES}} = 4 \wedge \kappa_{\text{SENT}} = 1 \wedge \text{yes} = 0 \wedge \text{no} = 1\right\}$$

$$\kappa_{\text{YES}}\text{--} \;;\; \kappa_{\text{SENT}}\text{++} \;;\; \text{yes}\text{++} \;;$$

$$\left\{\kappa_{\text{YES}} = 3 \wedge \kappa_{\text{SENT}} = 2 \wedge \text{yes} = 1 \wedge \text{no} = 1\right\}$$



(II) Acceleration



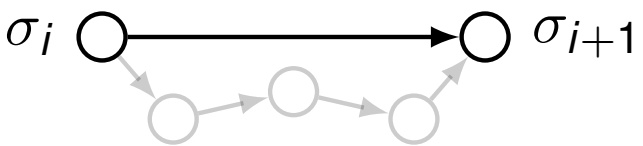
Acceleration factor can be any natural number δ

We allow unbounded number of processes to execute in one step

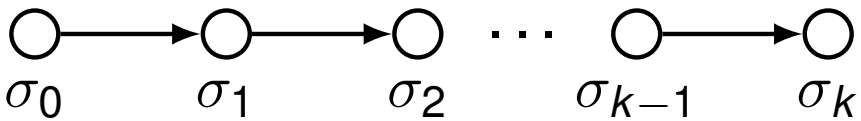
(III) Bounded Model Checking with SMT

A transition by δ_i processes (in linear integer arithmetic):

$$T(\sigma_i, \sigma_{i+1}, \delta_i) = \begin{bmatrix} \kappa_{\text{YES}}^{i+1} = \kappa_{\text{YES}}^{i} - \delta_i \wedge \\ \kappa_{\text{SENT}}^{i+1} = \kappa_{\text{SENT}}^{i} + \delta_i \wedge \\ \text{yes}^{i+1} = \text{yes}^{i} + \delta_i \end{bmatrix}$$



Execution:



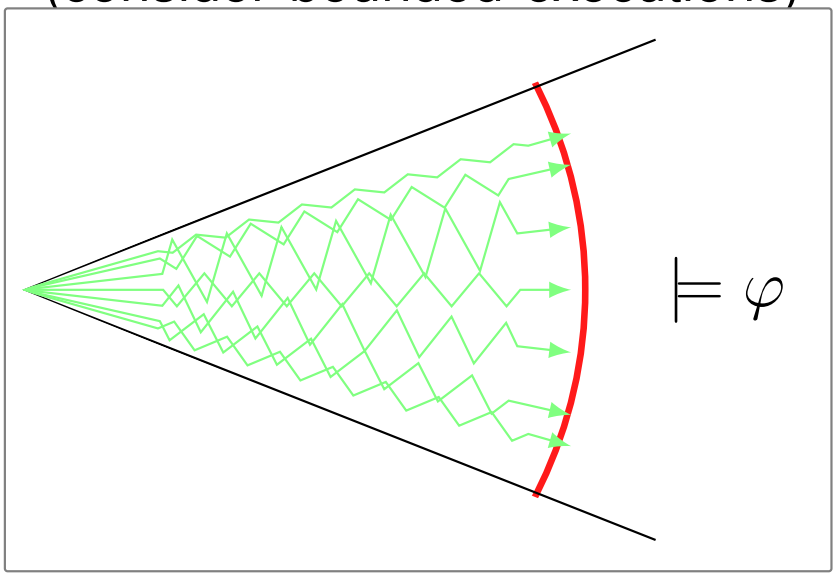
SMT formula: $T(\sigma_0, \sigma_1, \delta_0) \wedge T(\sigma_1, \sigma_2, \delta_1) \wedge \cdots \wedge T(\sigma_{k-1}, \sigma_k, \delta_{k-1}) \wedge \text{Spec}$

how long should the executions be?

Completeness of Bounded Model Checking

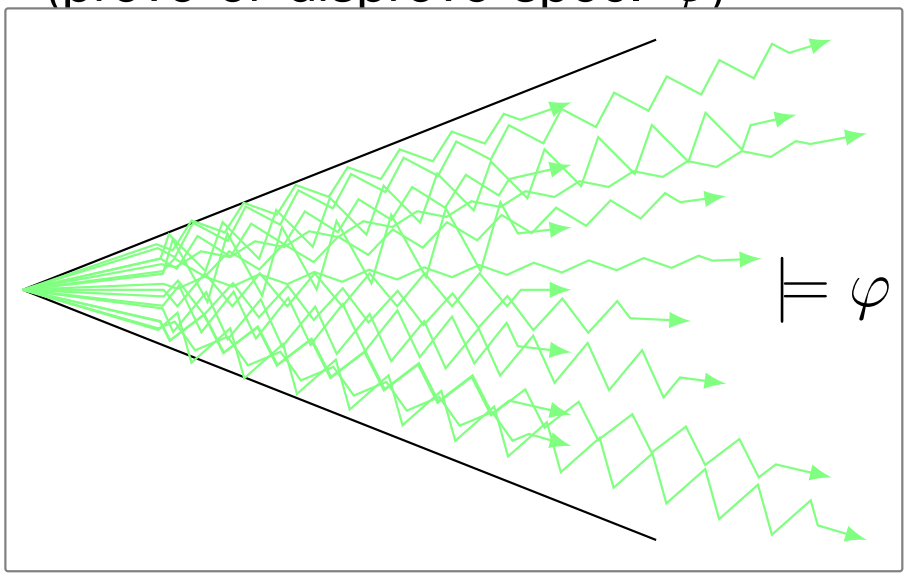
What we **can** do:

(consider bounded executions)



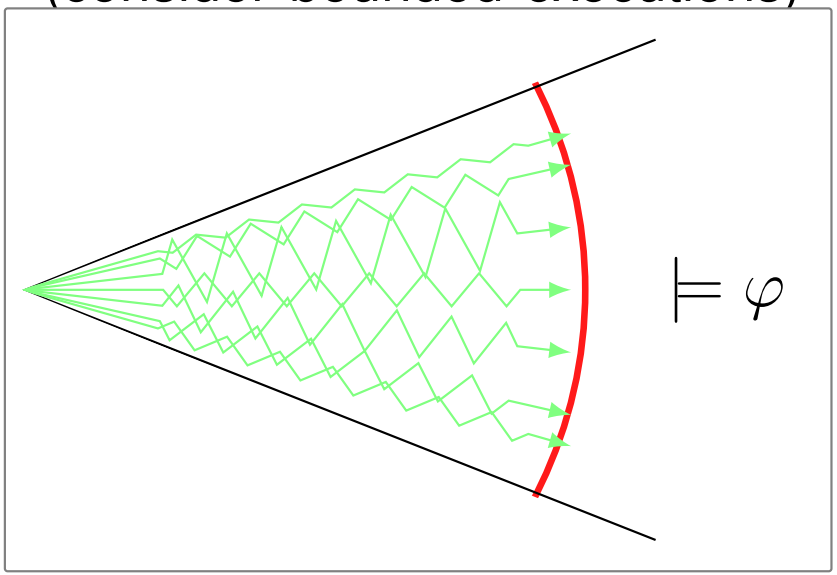
What we **want** to do:

(prove or disprove spec. φ)



Completeness of Bounded Model Checking

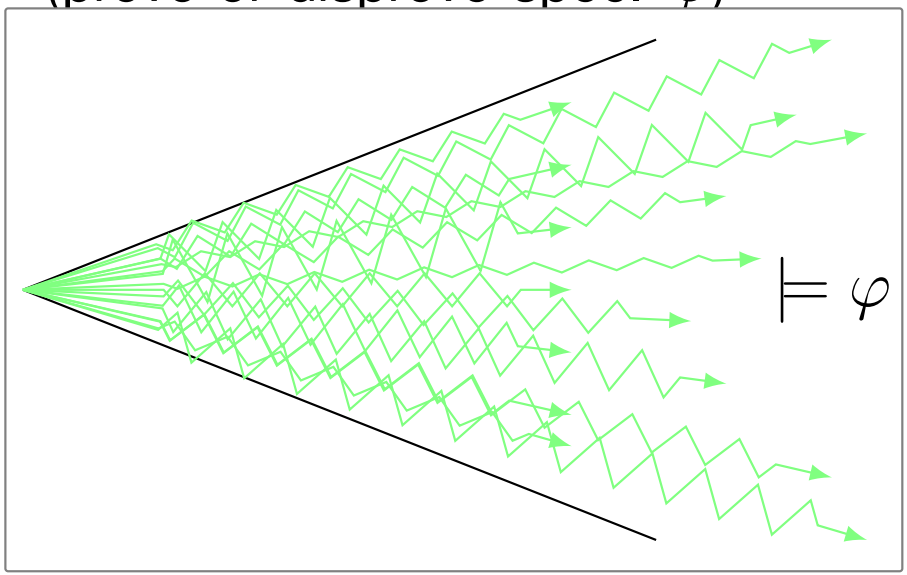
What we **can** do:
(consider bounded executions)



iff

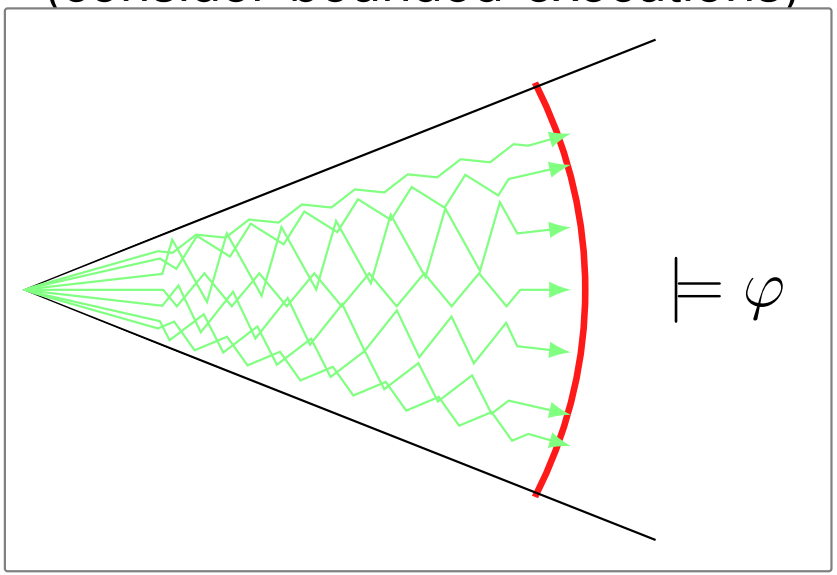
?

What we **want** to do:
(prove or disprove spec. φ)



Completeness of Bounded Model Checking

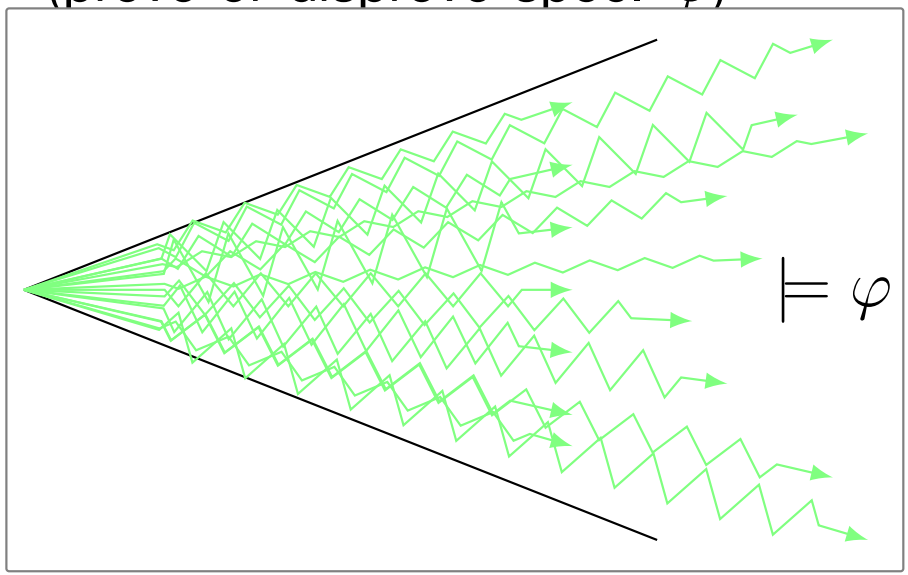
What we **can** do:
(consider bounded executions)



iff

!

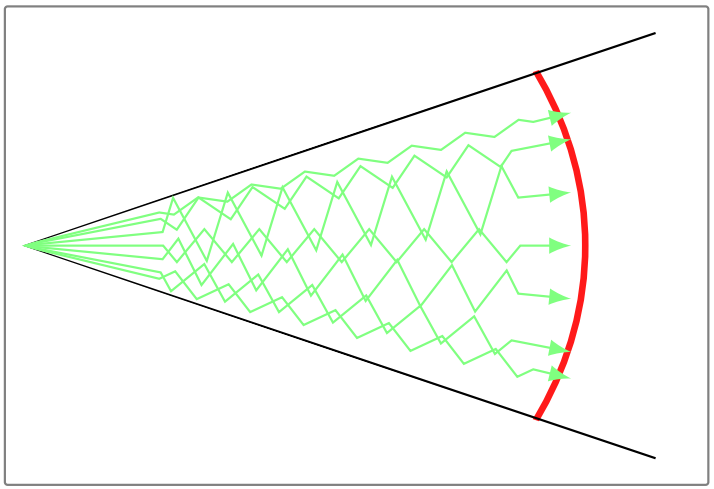
What we **want** to do:
(prove or disprove spec. φ)



Complete and efficient BMC for:

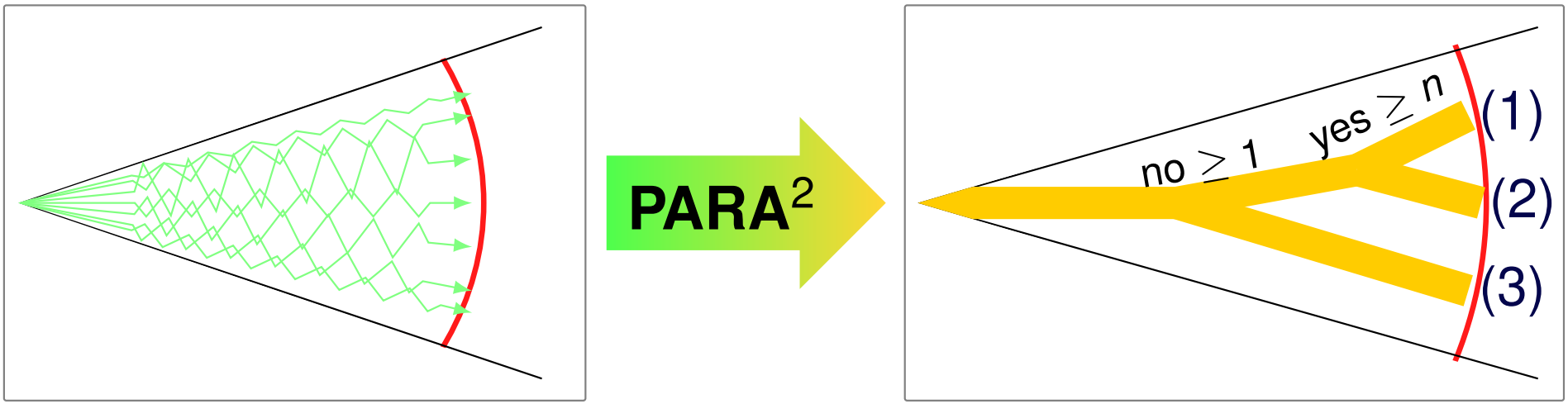
- reachability properties [Konnov, Lazić, Veith, Widder: FMSD'17]
- safety and liveness properties [Konnov, Lazić, Veith, Widder: POPL'17]

Searching all bounded executions is inefficient



PARAmeterized PAth Reduction & Acceleration

Searching all bounded executions is inefficient



The argument contains:

- reordering transitions (e.g., x_{++}; z_{++}; x_{++} becomes x_{++}; x_{++}; z_{++})

- acceleration (e.g., x_{++}; x_{++}; z_{++} becomes x += 2; z_{++})

FMSD'17

Reachability

a bad state is never reached

a **counterexample** is
a finite path

propositional formulas
specify bad states

POPL'17

Safety & Liveness

e.g., **good eventually happens**

a **counterexample** is
an infinite path

shortening is not obvious

temporal formulas
specify bad executions

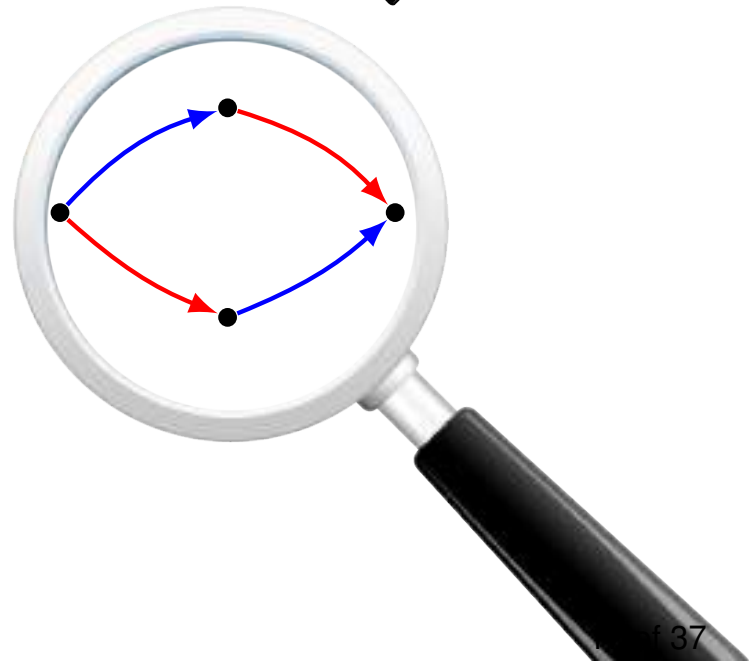
Safety and liveness: three major challenges

A) A temporal logic for bad executions

$$\mathbf{E}\left(\varphi_1 \wedge \mathbf{F}\,\mathbf{G}\left(\varphi_2 \vee \varphi_3\right)\right)$$

B) Enumerating shapes of counterexamples

C) Property specific PARA^2



(A) Specifying bad executions

Non-Triviality:

if all processes vote Yes,

and if no process crashes and no process suspects a crash

then eventually every process proposes commit

(A) Specifying bad executions

- **Non-Triviality:**
  - ~~if~~ all processes vote Yes,
  - and ~~if~~ no process crashes and no process suspects a crash
  - ~~then eventually every process proposes commit~~
  - and at least one process never proposes commit

(A) Specifying bad executions

Non-Triviality:

~~if~~ all processes vote Yes,

and ~~if~~ no process crashes and no process suspects a crash,

~~then eventually every process proposes commit~~

and at least one process never proposes commit

$$\mathbf{E}\left(\kappa_{\mathrm{NO}} = 0 \;\wedge\; \mathbf{G}\left(\kappa_{\mathrm{CRASH}} = 0 \wedge \kappa_{\mathrm{SUSPECT}} = 0\right) \wedge \mathbf{G}\left(\kappa_{\mathrm{YES}} \neq 0 \vee \kappa_{\mathrm{NO}} \neq 0 \vee \kappa_{\mathrm{SENT}} \neq 0 \vee \kappa_{\mathrm{ABORT}} \neq 0\right) \wedge \mathbf{G}\,\mathbf{F}\,\psi_{\mathrm{fair}}\right)$$

(A) Specifying bad executions

Non-Triviality:

~~if~~ all processes vote Yes,

and ~~if~~ no process crashes and no process suspects a crash,

~~then eventually every process proposes commit~~

and at least one process never proposes commit

$$\mathbf{E}\Big(\kappa_{\text{NO}} = 0 \;\wedge\; \mathbf{G}\big(\kappa_{\text{CRASH}} = 0 \wedge \kappa_{\text{SUSPECT}} = 0\big) \wedge \mathbf{G}\big(\kappa_{\text{YES}} \neq 0 \vee \kappa_{\text{NO}} \neq 0 \vee \kappa_{\text{SENT}} \neq 0 \vee \kappa_{\text{ABORT}} \neq 0\big) \;\wedge\; \mathbf{G}\,\mathbf{F}\,\psi_{\text{fair}}\Big)$$

Propositional formulas:

(1) $\bigwedge_{\ell \in S} \kappa_\ell = 0$

(2) $\bigvee_{\ell \in S} \kappa_\ell \neq 0$

(3) $\bigwedge_{S \subseteq T} \bigvee_{\ell \in S} \kappa_\ell \neq 0$

(4) Bool(Guards) $\rightarrow$ (1) $\wedge$ (2) $\wedge$ (3)

Temporal formulas:

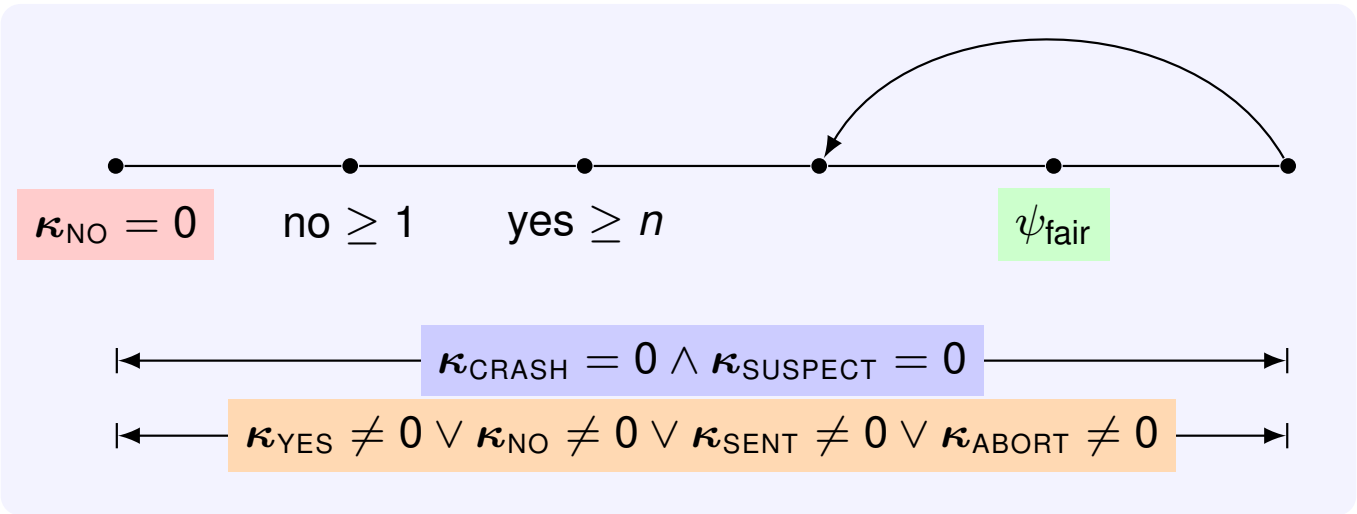
$\psi ::= \mathit{prop} \mid \mathbf{G}\,\psi \mid \mathbf{F}\,\psi \mid \psi \wedge \psi$

We call this fragment ELTL_{FT}

(B) Finitely many lasso shapes

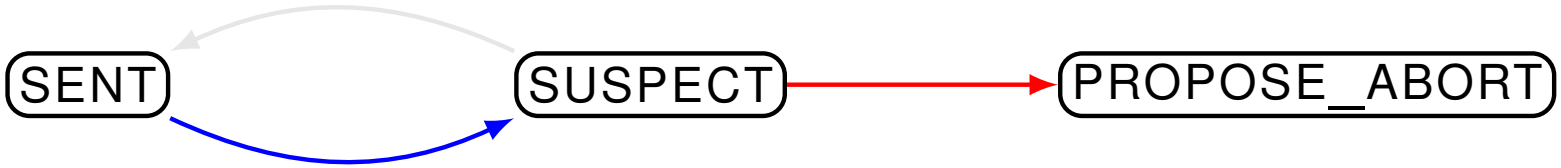
For an ELTL_{FT} formula, construct a (finite) set of lasso shapes, e.g.,

$$\mathbf{E}\Big(\kappa_{\text{NO}} = 0 \;\wedge\; \mathbf{G}\big(\kappa_{\text{CRASH}} = 0 \wedge \kappa_{\text{SUSPECT}} = 0\big) \wedge \mathbf{G}\big(\kappa_{\text{YES}} \neq 0 \vee \kappa_{\text{NO}} \neq 0 \vee \kappa_{\text{SENT}} \neq 0 \vee \kappa_{\text{ABORT}} \neq 0\big) \;\wedge\; \mathbf{G}\,\mathbf{F}\,\psi_{\text{fair}}\Big)$$



There are 4 more lasso shapes for this formula, depending on the guards

(C) Property-specific mover analysis



$$\models \mathbf{G}\,(\kappa_{\text{SUSPECT}} \neq 0)$$

$$\left[\begin{array}{c}\kappa_{\text{SENT}}\text{--}\\ \kappa_{\text{SUSPECT}}\text{++}\end{array}\right]\left[\begin{array}{c}\kappa_{\text{SUSPECT}}\text{--}\\ \kappa_{\text{ABORT}}\text{++}\end{array}\right]$$

$$\left\{\begin{array}{c}\kappa_{\text{SENT}} = 1\\ \kappa_{\text{SUSPECT}} = 1\\ \kappa_{\text{ABORT}} = 0\end{array}\right\} \qquad \left\{\begin{array}{c}\kappa_{\text{SENT}} = 0\\ \kappa_{\text{SUSPECT}} = 1\\ \kappa_{\text{ABORT}} = 1\end{array}\right\}$$

$$\left[\begin{array}{c}\kappa_{\text{SUSPECT}}\text{--}\\ \kappa_{\text{ABORT}}\text{++}\end{array}\right]\left[\begin{array}{c}\kappa_{\text{SENT}}\text{--}\\ \kappa_{\text{SUSPECT}}\text{++}\end{array}\right]$$

$$\not\models \mathbf{G}\,(\kappa_{\text{SUSPECT}} \neq 0)$$

(C) Solution - Classifying threads

$$\mathbf{G}\,(\kappa_{\text{SUSPECT}} \neq 0)$$

1) thread **ends** in ■

2) thread **begins** in ■

Move only threads of specific types

(C) Solution - Classifying threads

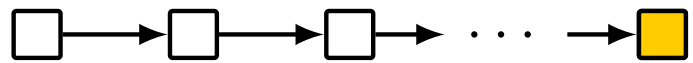
$\mathbf{G}\,(\kappa_{\text{SUSPECT}} \neq 0)$

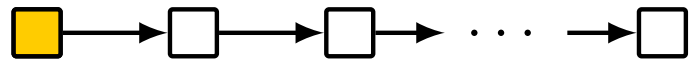
$\mathbf{G}\,(\bigvee_{\ell \in S} \kappa_\ell \neq 0)$

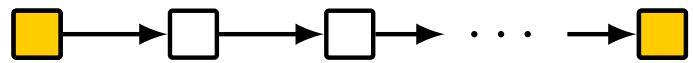
1) thread **ends** in 

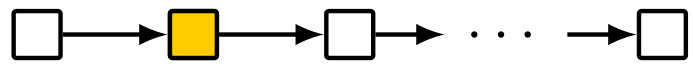
2) thread **begins** in 

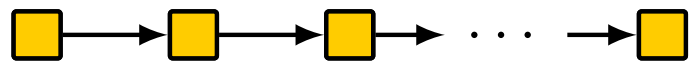
Move only threads of specific types

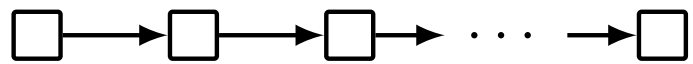
1) 

2) 

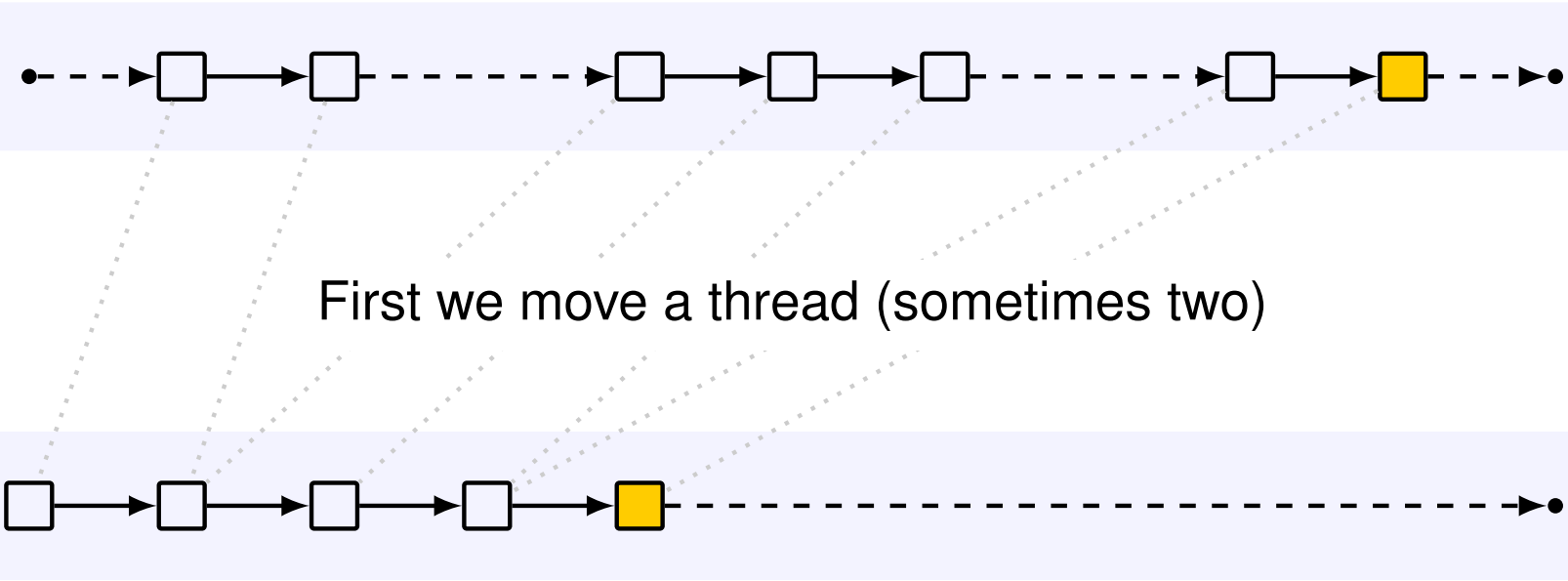
3) 

4) 

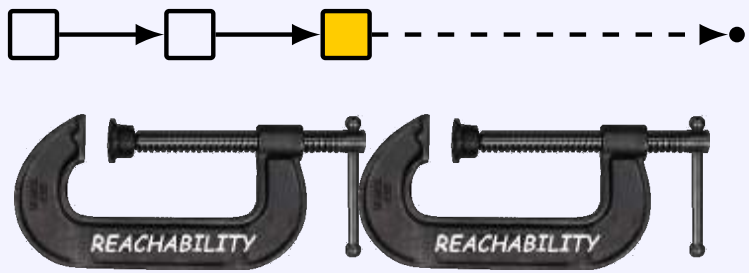
5) 

6) 

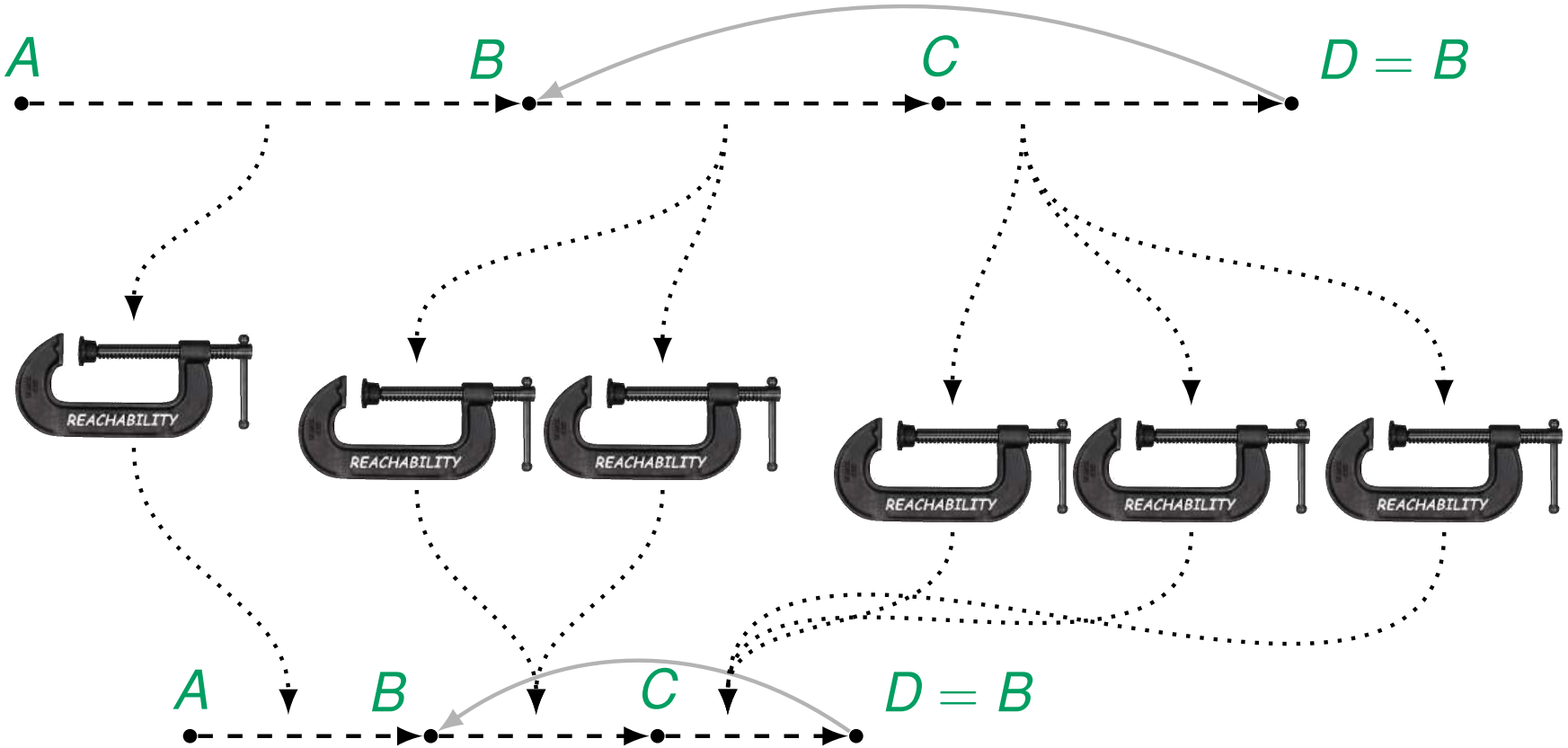
Shortening using threads



Then we apply shortening from [FMSD'17] to the thread and to the rest



Short counterexamples for safety or liveness



Safety & liveness (POPL'17)

Every lasso can be transformed into a bounded one. The bound depends on the process code and the specification, not the parameters.

VERIFICATION

SYNTHESIS

Which thresholds make an algorithm correct?

OPODIS'17

RANDOMIZATION

Different threshold guards for one sketch

```
local myval_i ∈ {0,1}          -- did process i receive bcast?

while true do
 if myval_i = 1 and not sent ECHO before
 then send ECHO to all

 if received ECHO from at least t + 1 distinct processes
    and not sent ECHO before
 then send ECHO to all

 if received ECHO from at least n - t distinct processes
 then accept
od
```

resilience: of $n > 3t$ processes, $f \leq t$ processes are Byzantine

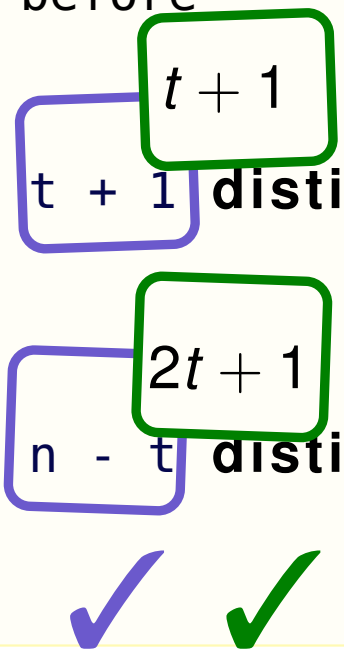
Different threshold guards for one sketch

```
local myval_i ∈ {0,1}          -- did process i receive bcast?

while true do
 if myval_i = 1 and not sent ECHO before
 then send ECHO to all

 if received ECHO from at least t + 1 distinct processes
    and not sent ECHO before
 then send ECHO to all

 if received ECHO from at least n - t distinct processes
 then accept
od
```



resilience: of $n > 3t$ processes, $f \leq t$ processes are Byzantine

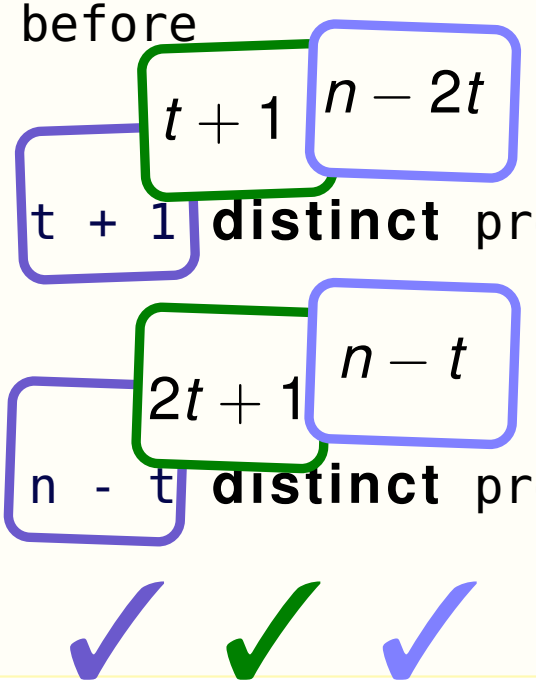
Different threshold guards for one sketch

```
local myval_i ∈ {0,1}          -- did process i receive bcast?

while true do
 if myval_i = 1 and not sent ECHO before
 then send ECHO to all

 if received ECHO from at least t + 1 distinct processes
    and not sent ECHO before
 then send ECHO to all

 if received ECHO from at least n - t distinct processes
 then accept
od
```



resilience: of $n > 3t$ processes, $f \leq t$ processes are Byzantine

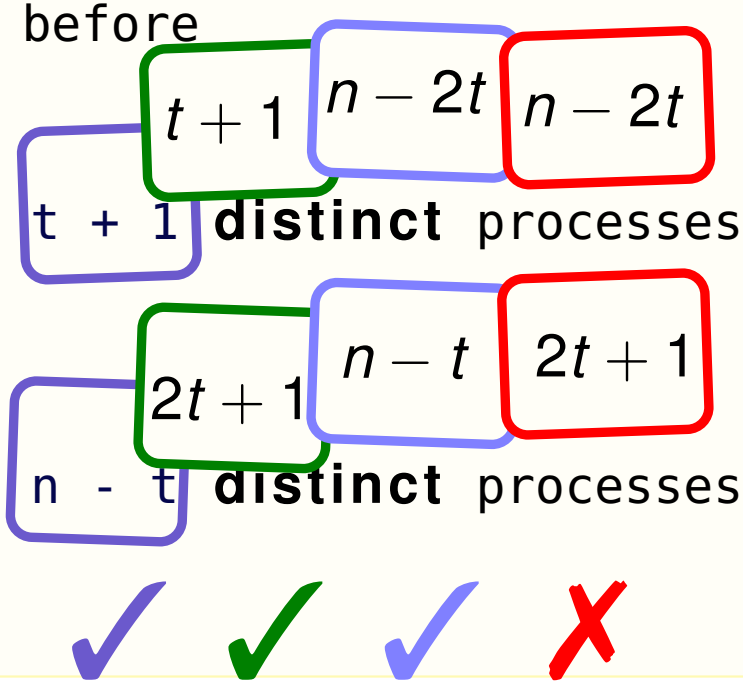
Different threshold guards for one sketch

```
local myval_i ∈ {0,1}          -- did process i receive bcast?

while true do
 if myval_i = 1 and not sent ECHO before
 then send ECHO to all

 if received ECHO from at least t + 1 distinct processes
    and not sent ECHO before
 then send ECHO to all

 if received ECHO from at least n - t distinct processes
 then accept
od
```



resilience: of $n > 3t$ processes, $f \leq t$ processes are Byzantine

Find thresholds automatically?

```
local myval_i ∈ {0,1}          -- did process i receive bcast?

while true do
 if myval_i = 1 and not sent ECHO before
 then send ECHO to all

 if received ECHO from at least ? distinct processes
    and not sent ECHO before
 then send ECHO to all

 if received ECHO from at least ? distinct processes
 then accept
od
```

$?_1 \cdot n + ?_2 \cdot t + ?_3$

$?_4 \cdot n + ?_5 \cdot t + ?_6$

resilience: of $n > 3t$ processes, $f \leq t$ processes are Byzantine

Synthesis loop

Find $?_1, \ldots, ?_k \in \mathbb{Q}$

coefficients

counterexample

solution

Generator
infinite
search space

Verifier
Byzantine MC

Synthesis loop

Find $?_1, \ldots, ?_k \in \mathbb{Q}$

coefficients

counterexample

solution

Generator
infinite
search space

Verifier
Byzantine MC

Termination?	sane guards $\Rightarrow$ bounded search space
Efficiency?	Generator learns from counterexamples

Termination: Sane guards in the interval $[0, n]$

`if received` $\frac{n}{2}$ `messages...` ✓ — wait for a majority

`if received` $t+1$ `messages...` ✓ — wait for a correct process

`if received` $2n$ `messages...` ✗

`if received` -5 `messages...` ✗

Theorem

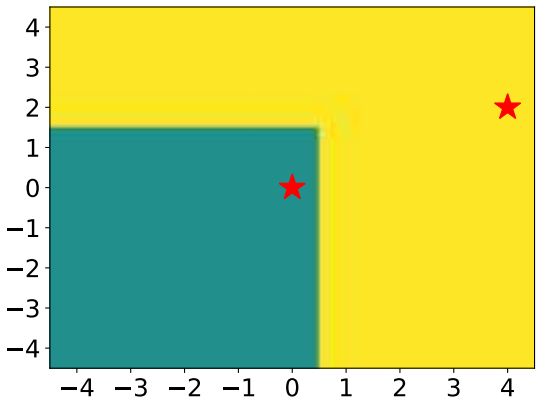
Assume: $n > \sum_{1 \le i \le k} \delta_i \cdot t_i$ and $\forall i.\ t_i \ge 0$ — resilience cond.

$$0 \ \le\ \mathbf{?}_a n + \sum_{1 \le i \le k} \mathbf{?}_{b_i} \cdot t_i + \mathbf{?}_c \ \le\ n$$ — threshold

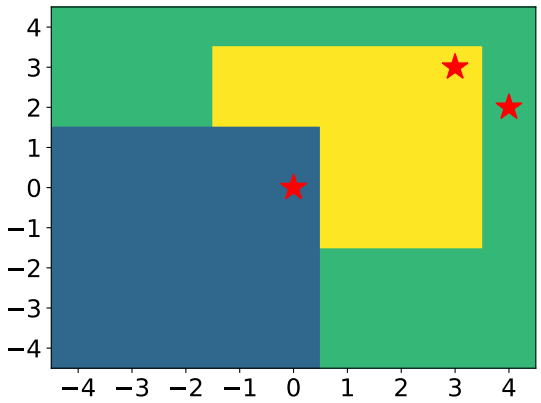
Then:

$$\begin{cases} 0 \le \mathbf{?}_a \le 1 \\ -B_i \le \mathbf{?}_{b_i} \le B_i & \text{for } B_i = \delta_i + 1 \text{ and } 1 \le i \le k \\ -C \le \mathbf{?}_c \le C & \text{for } C = k + 1 + 2(\delta_1 + \cdots + \delta_k) \end{cases}$$

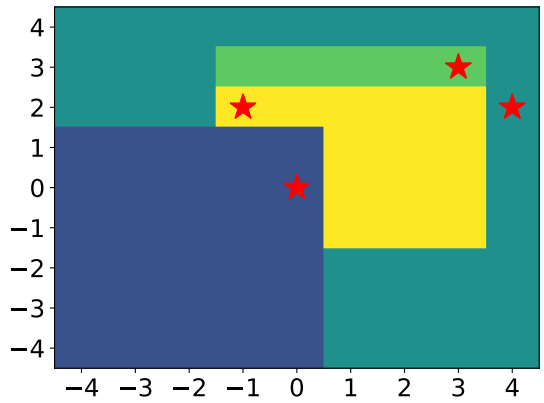
Efficiency: Learning from counterexamples



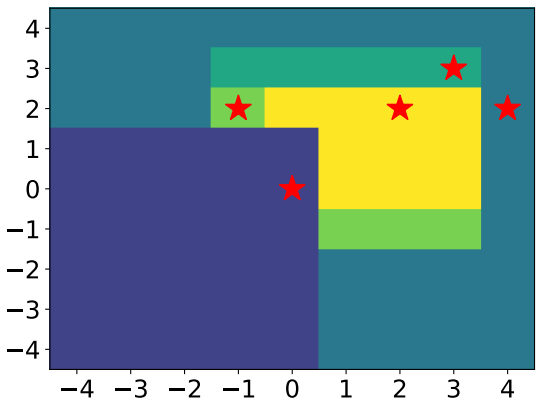
1. unforgeability



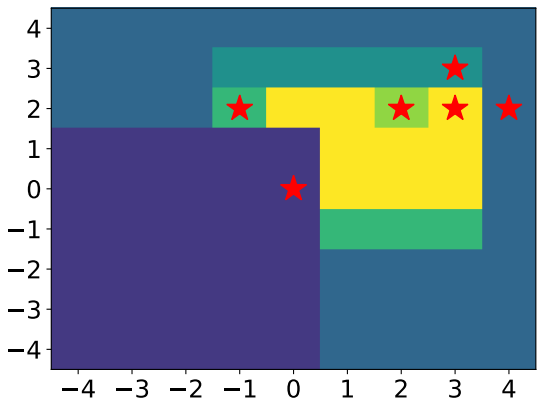
2. sanity



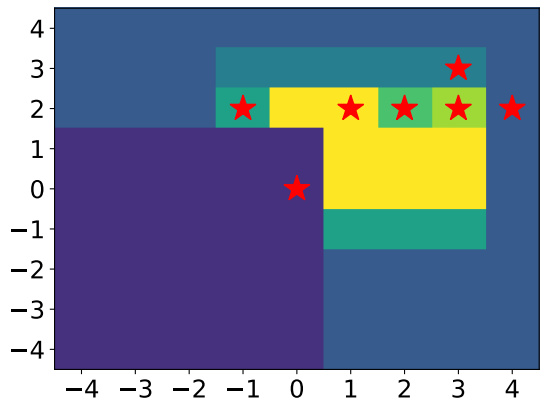
3. correctness



4. sanity



5. relay



6. relay

VERIFICATION

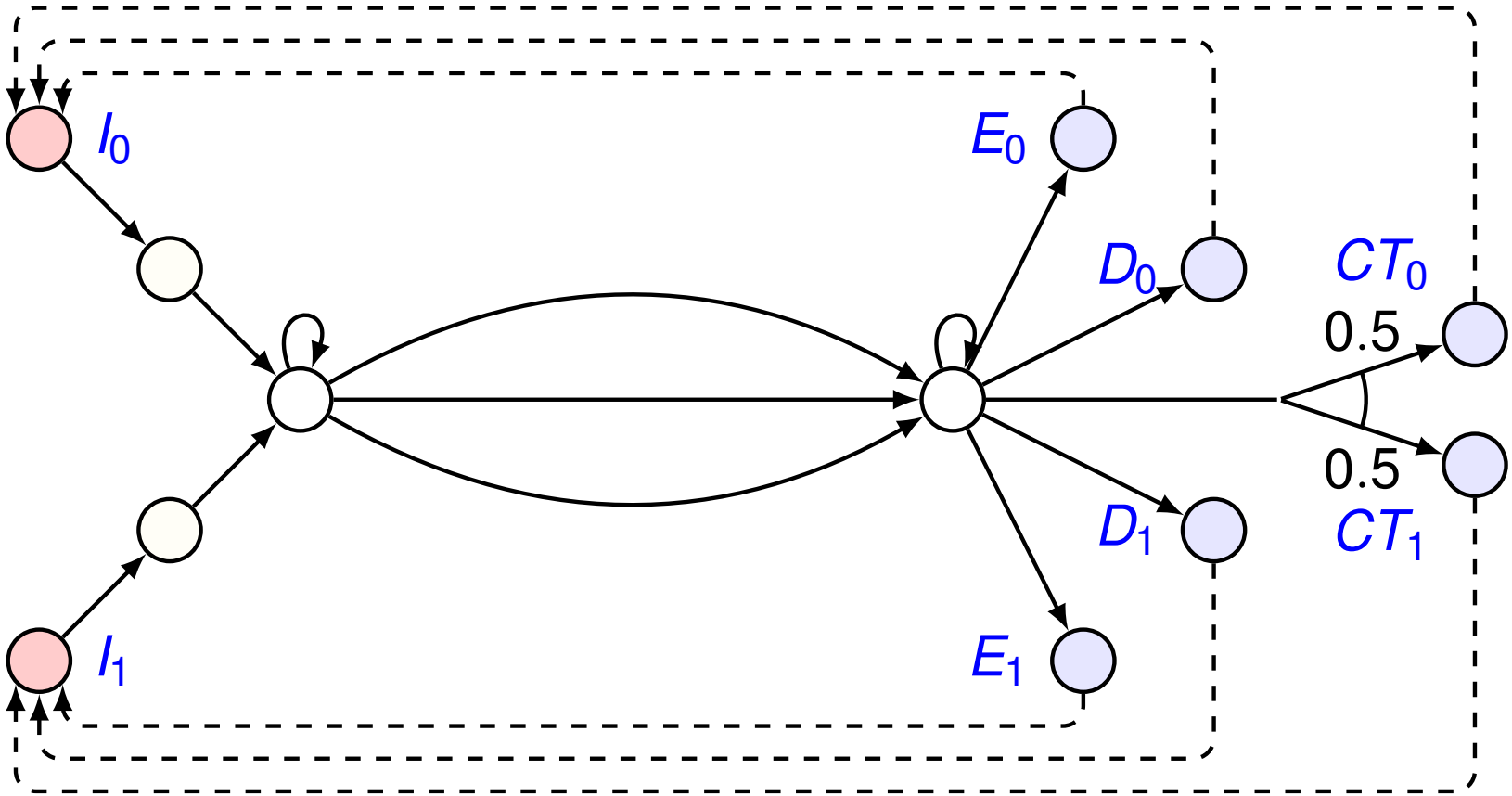
SYNTHESIS

RANDOMIZATION

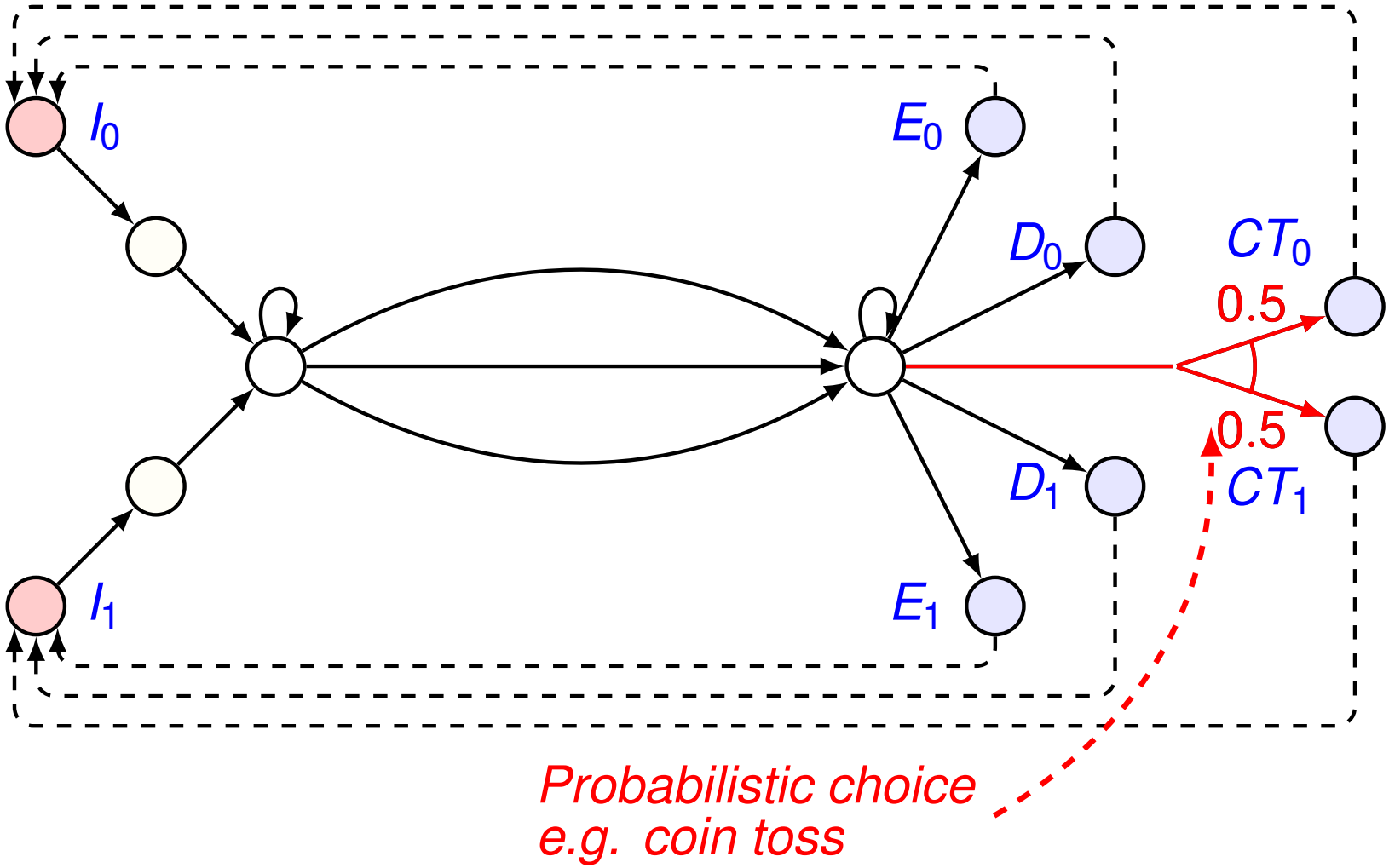
Is a randomized algorithm correct?

CONCUR'19

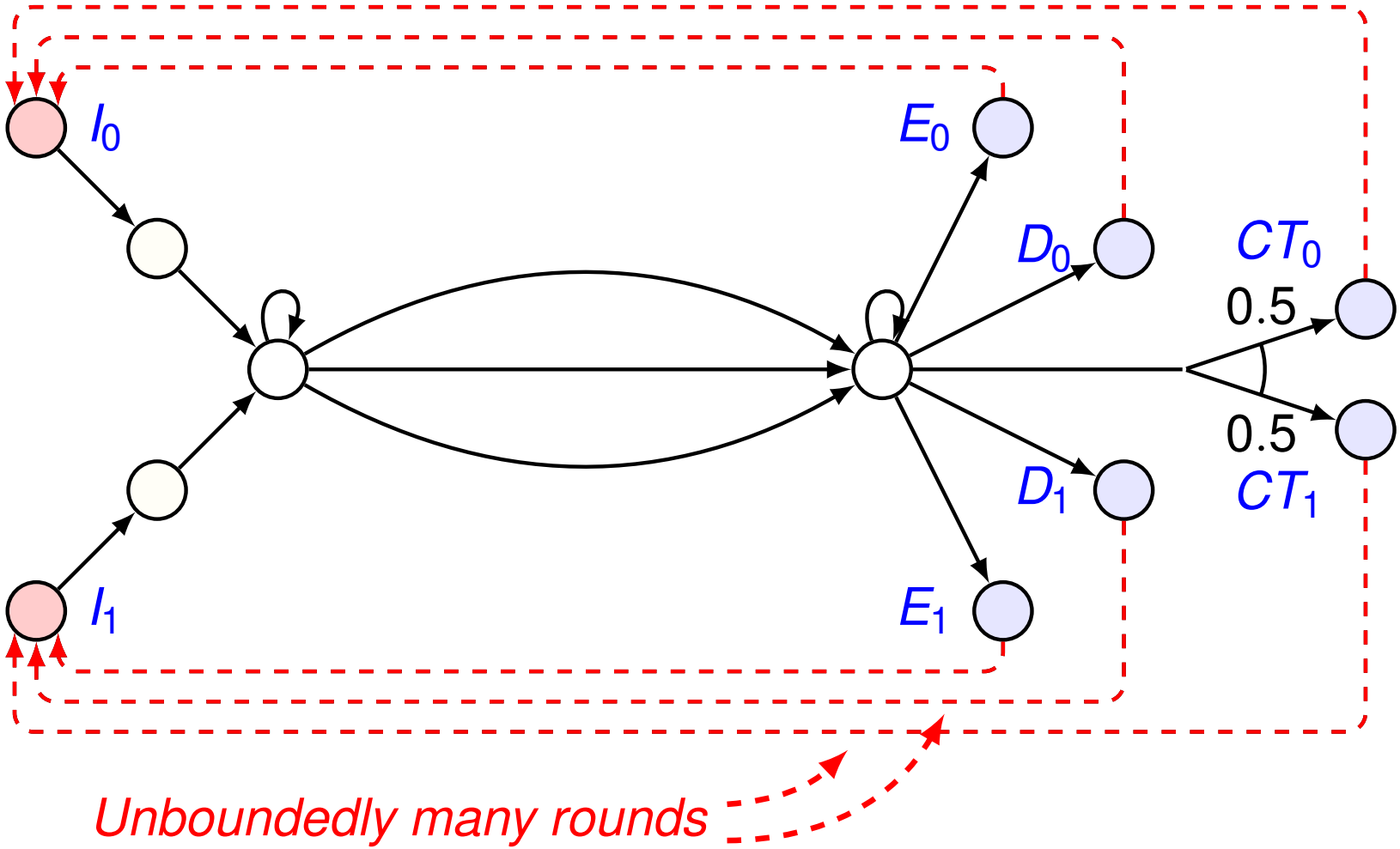
Probabilistic Threshold Automata (PTA)



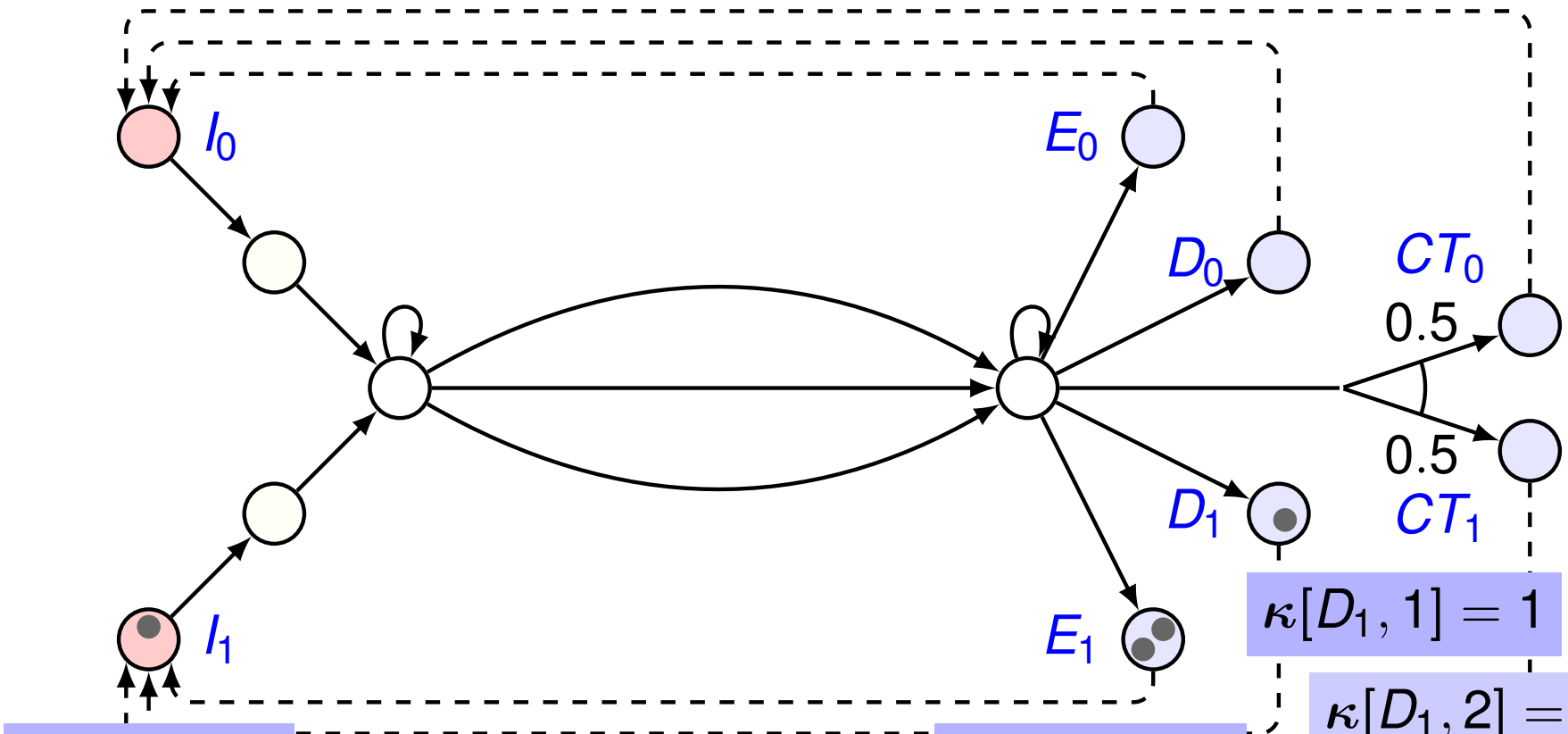
Probabilistic Threshold Automata (PTA)



Probabilistic Threshold Automata (PTA)



Probabilistic Counter System for a PTA



Specifications in LTL$_{-X}$ with counters

Agreement : No two correct processes decide differently

$$\forall k \in \mathbb{N}_0,\ \forall k' \in \mathbb{N}_0.\ \mathbf{A}\ \big(\mathbf{F}\ \kappa[D_v, k] > 0 \quad \rightarrow \quad \mathbf{G}\ \kappa[D_{1-v}, k'] = 0\big)$$

Termination : Under every round-rigid adversary, with probability 1 every correct process eventually decides

$$\mathbb{P}_s\big(\bigvee_{k\in\mathbb{N}_0}\ \bigvee_{v\in\{0,1\}}\ \mathbf{G}\ \bigwedge_{\ell\in\mathcal{L}\setminus\{D_v\}}\ \kappa[\ell, k] = 0\big) = 1$$

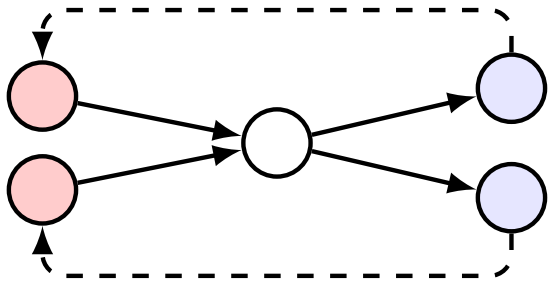
Reduction to verification [POPL'17]

Specifications

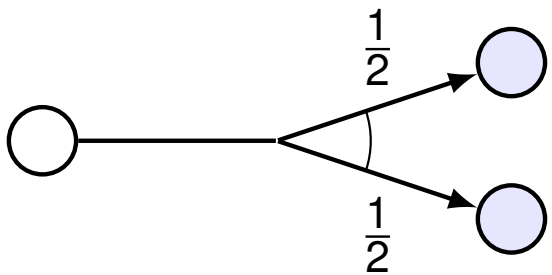
$$\forall k,\ \forall k'.\ \mathbf{A}\ \varphi[k, k']$$

$$\mathbb{P}_s\big(\varphi[k]\big) = 1$$

Unboundedly many rounds



Probabilistic choice



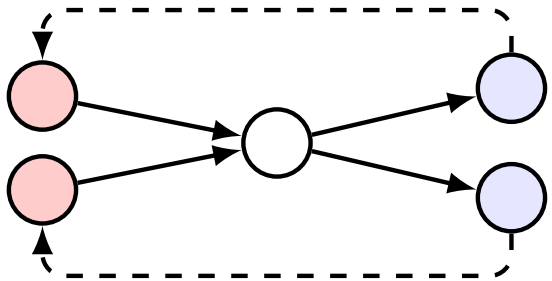
Reduction to verification [POPL'17]

Specifications

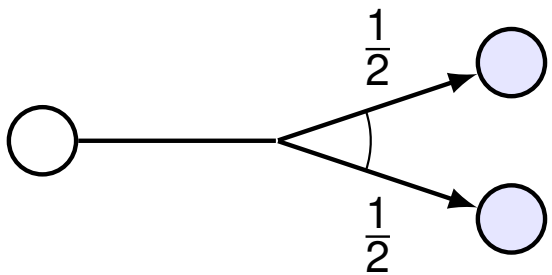
$$\forall k,\ \forall k'.\ \mathbf{A}\ \varphi[k,k']$$

$$\mathbb{P}_{\mathsf{s}}\big(\varphi[k]\big) = 1$$

Unboundedly many rounds



Probabilistic choice



One-round specs

$$\forall k.\ \mathbf{A}\ \psi[k]$$

Reduction to verification [POPL'17]

Specifications

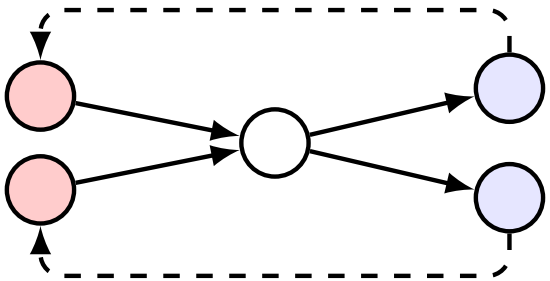
$$\forall k,\ \forall k'.\ \mathbf{A}\ \varphi[k,k']$$

$$\mathbb{P}_{\mathsf{s}}\big(\varphi[k]\big) = 1$$

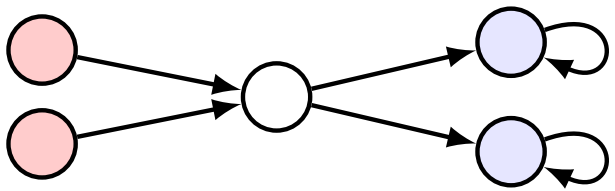
One-round specs

$$\forall k.\ \mathbf{A}\ \psi[k]$$

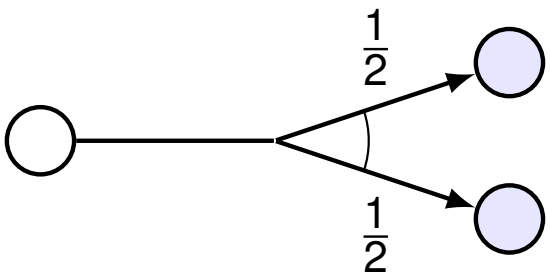
Unboundedly many rounds



One-round system



Probabilistic choice



Reduction to verification [POPL'17]

Specifications

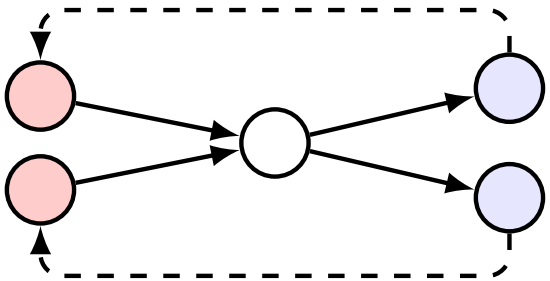
$\forall k,\ \forall k'.\ \mathbf{A}\ \varphi[k,k']$

$\mathbb{P}_s\big(\varphi[k]\big) = 1$

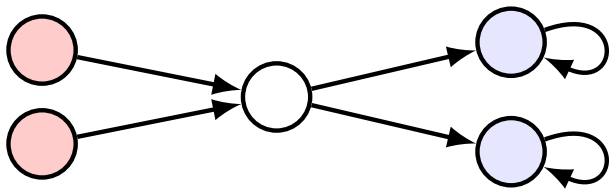
One-round specs

$\forall k.\ \mathbf{A}\ \psi[k]$

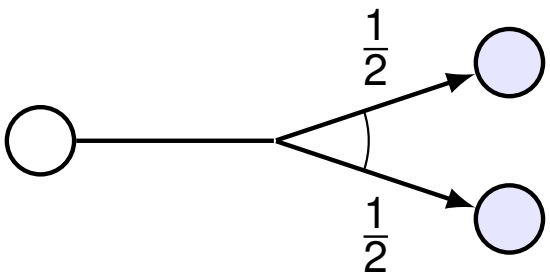
Unboundedly many rounds



One-round system



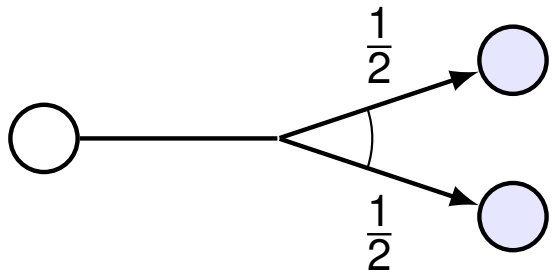
Probabilistic choice



Property-specific analysis

Reduction to verification [POPL'17]

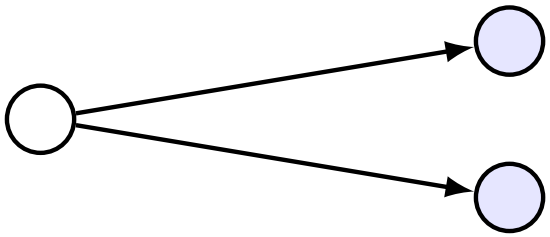
Probabilistic choice



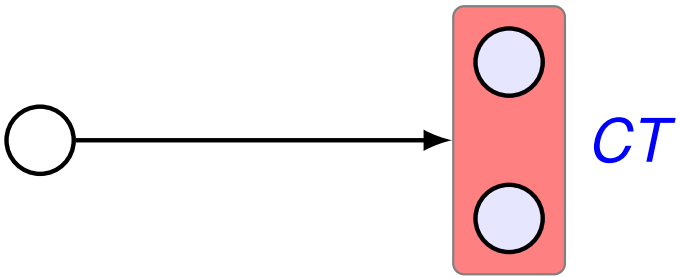
$\forall k,\ \forall k'.\ \mathbf{A}\ \varphi[k,k']$

$\mathbb{P}_s(\varphi[k]) = 1$

Non-determinism



Abstracting Coin Tosses



Conclusions

VERIFICATION
Is an algorithm correct?

FMSD'17

POPL'17

SYNTHESIS
Which thresholds make an algorithm correct?

OPODIS'17

RANDOMIZATION
Is a randomized algorithm correct?

CONCUR'19

References

Nathalie Bertrand, Igor Konnov, Marijana Lazic, and Josef Widder, *Verification of Randomized Distributed Algorithms under Round-Rigid Adversaries*, HAL **hal-01925533** (2018).

Idan Berkovits, Marijana Lazić, Giuliano Losa, Oded Padon, and Sharon Shoham, *Verification of threshold-based distributed algorithms by decomposition to decidable logics*, CoRR **abs/1905.07805** (2019), (to appear at CAV 2019).

Cezara Drăgoi, Marijana Lazić, and Josef Widder, *Communication-closed layers as paradigm for distributed systems: A manifesto*, Sinteza, 2018, pp. 131–138.

Éric Goubault, Marijana Lazić, Jérémy Ledent, and Sergio Rajsbaum, *Wait-free solvability of equality negation tasks*, (submitted to DISC 2019).

Igor Konnov, Marijana Lazić, Helmut Veith, and Josef Widder, *Para^2 : Parameterized path reduction, acceleration, and SMT for reachability in threshold-guarded distributed algorithms*, Formal Methods in System Design **51** (2017), no. 2, 270–307.

Igor Konnov, Marijana Lazić, Helmut Veith, and Josef Widder, *A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms*, POPL, 2017, pp. 719–734.

Marijana Lazić, Igor Konnov, Josef Widder, and Roderick Bloem, *Synthesis of distributed algorithms with parameterized threshold guards*, OPODIS, LIPIcs, vol. 95, 2017, pp. 32:1–32:20.

Future work

VERIFICATION

- symmetry-breaking
- bounded message delays
- relative process speeds

SYNTHESIS

- synthesizing automata
- improved learning
- understanding thresholds

RANDOMIZATION

- arbitrary adversaries
- more automation
- general probabilities

Future work

VERIFICATION

- symmetry-breaking
- bounded message delays
- relative process speeds

SYNTHESIS

- synthesizing automata
- improved learning
- **understanding thresholds**

RANDOMIZATION

- arbitrary adversaries
- more automation
- general probabilities

[Berkovits, Lazić, Losa, Padon, Shoham: CAV'19]

Future work

VERIFICATION

- symmetry-breaking
- bounded message delays
- relative process speeds

SYNTHESIS

- synthesizing automata
- improved learning
- **understanding thresholds**

RANDOMIZATION

- arbitrary adversaries
- more automation
- general probabilities

[Berkovits, Lazić, Losa, Padon, Shoham: CAV'19]

Thank you!

Schemas - encoding representatives

Schema: $\{pre\}\ actions\ \{post\}$

e.g. $\{\}(\text{YES}\rightarrow\text{SENT})^{\delta_0}, (\text{NO}\rightarrow\text{SENT})^{\delta_1}, (\text{SENT}\rightarrow\text{COMMIT})^{\delta_2}, \ldots \{\text{yes} \geq n\} \ldots$

Can this schema violate **commit-validity**?

(if a process proposes commit, then all process have voted Yes)

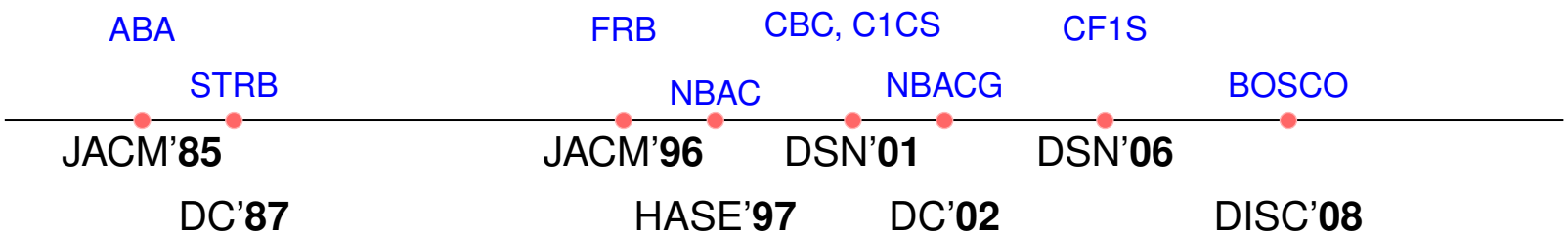
SMT solver tries to find: parameters n,
acceleration factors δ_i,
counters $\kappa^i_{\text{YES}}, \kappa^i_{\text{COMMIT}}, \ldots$

UNSAT $\rightsquigarrow$ the schema does not violate the property

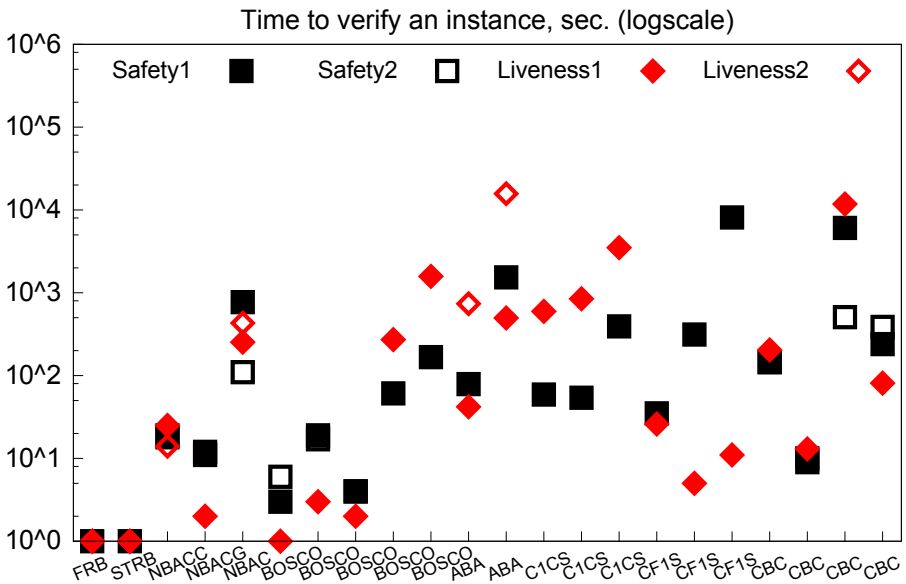
SAT $\rightsquigarrow$ gives a counterexample

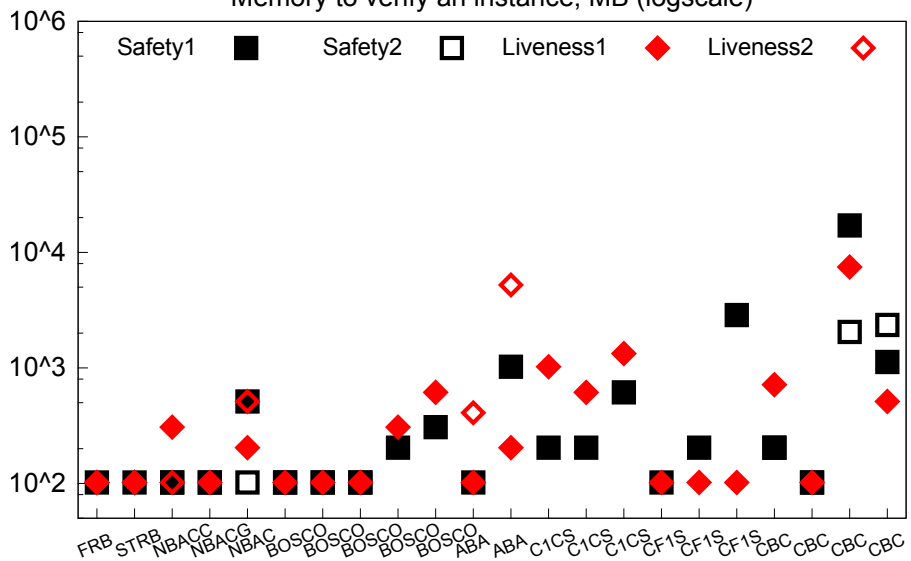
Experiments

we have verified 10 parameterized F-T distributed algorithms:



[forsyte.at/software/bymc]





We have synthesized

reliable broadcast, hybrid broadcast, and

BOSCO (one-step consensus) using

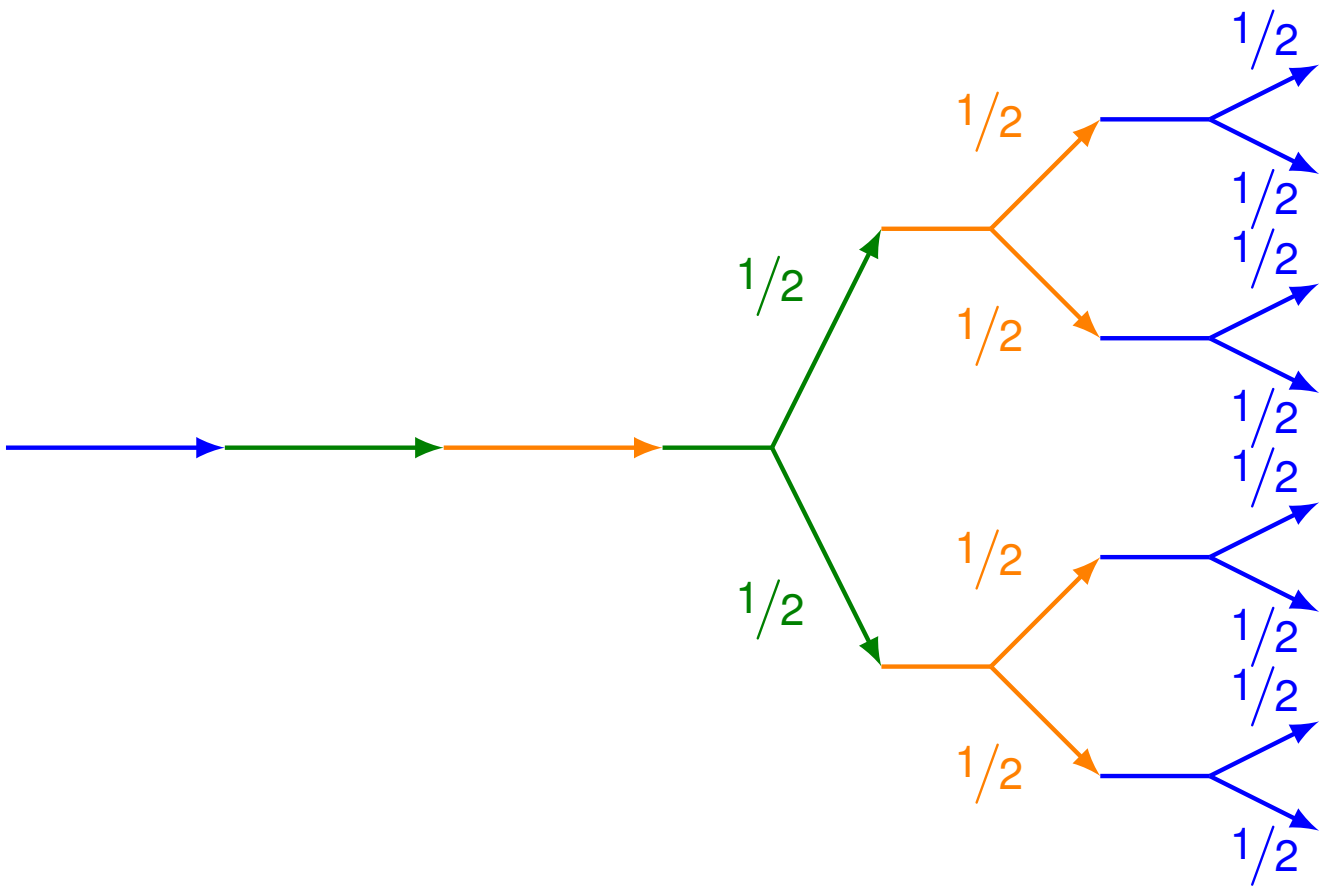
&

©VSC / Claudia Blaas-Schenner

[forsyte.at/software/bymc]

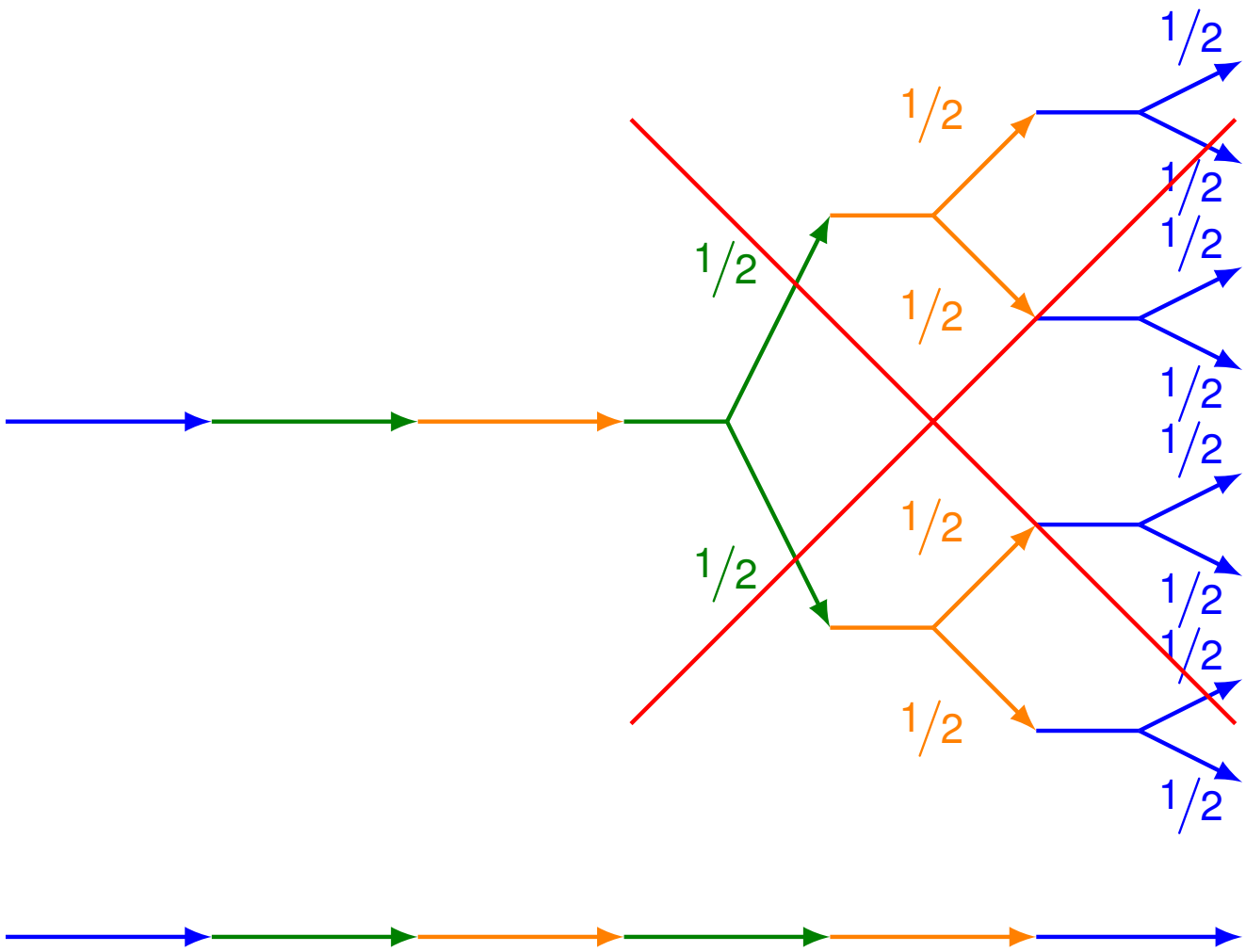
SYNTHESIZED
Correct-By-Construction

Probabilistic Reasoning



All outcomes have probability $p \geq (1/2)^n > 0$

Probabilistic Reasoning



All outcomes have probability $p \geq (1/2)^n > 0$

Capture that some processes toss a coin