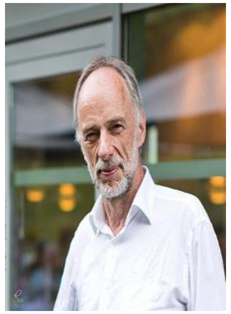
From TVLA to IVY

Mooly Sagiv

REPS at 60

Edinburgh, September 11, 2016

The TVLA Team

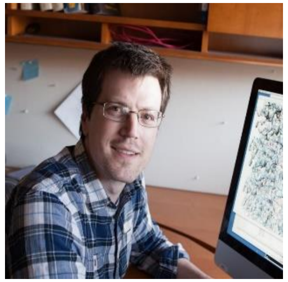









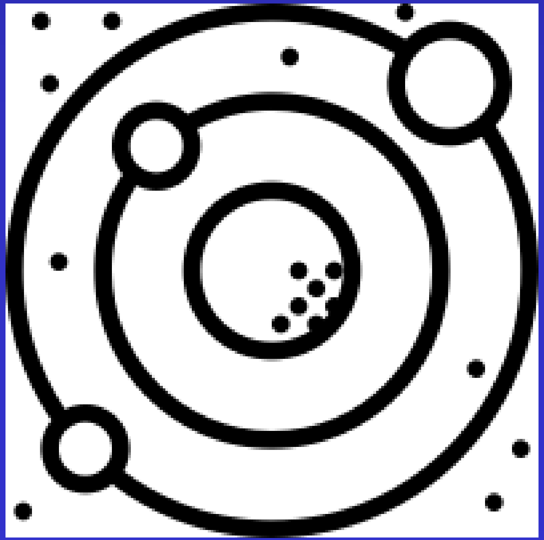


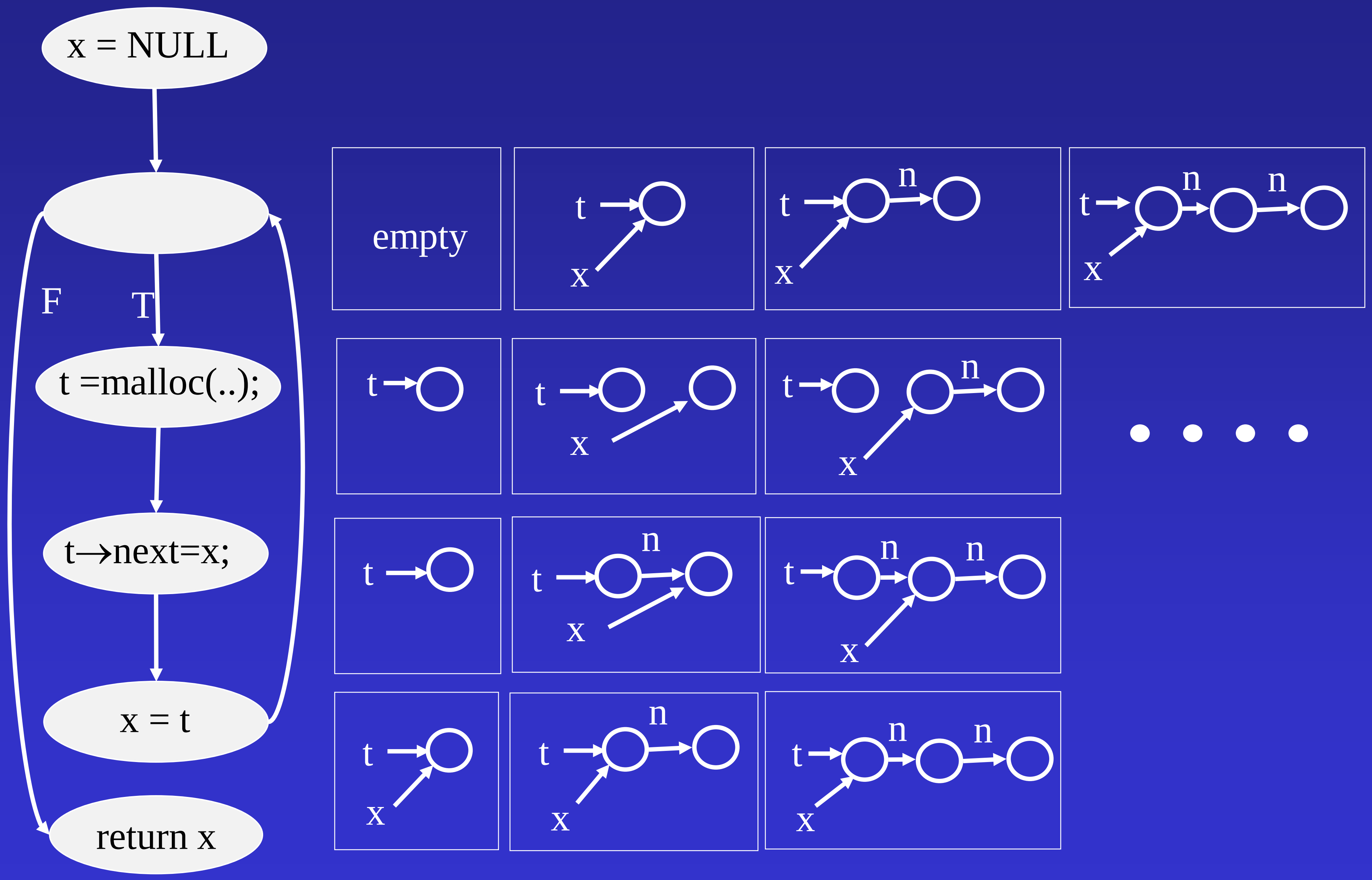



The TVLA Principles

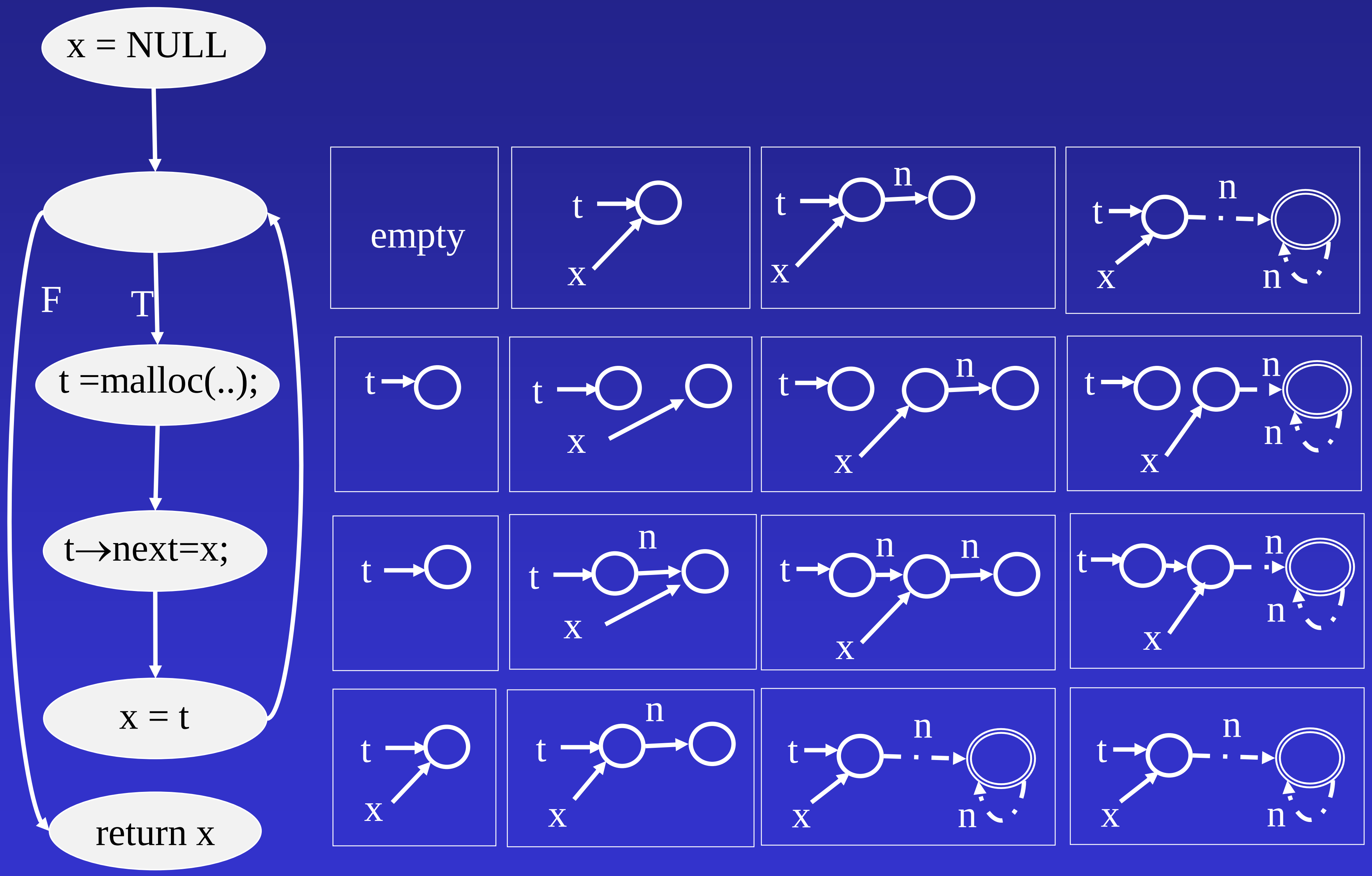
- Concrete semantics expressed as evolving relations/graphs in FO^{TC}
  - Celebrate unboundedness
  - No arithmetic
- Abstract Interpretation with Canonical Partially Disjunctive Abstraction
- Effective heuristics for automatic postcondition calculation
  - Kleene evaluation
  - Focus
  - Coerce
  - Differencing



Example: Concrete Interpretation



Example: Abstract Interpretation



Lessons learned

- FO^{TC} is powerful
- But FO^{TC} reasoning is complicated
  - Decidability of implications
  - Scalability of disjunctive abstractions
  - TVLA heuristics are effective for experts and specialized domains
    - Effective for shape analysis of common data structures and graduate students
    - But hard to apply by non experts and complicated clients
    - CEGAR has limited power

The IVY System

Kenneth McMillan

Microsoft Research

Oded Padon

TEL AVIV UNIVERSITY

http://microsoft.github.io/ivy/

IVY Principles

- Concrete semantics expressed as evolving relations/graphs in Effectively Propositional Logic (EPR)
  - Explore locality of updates
  - Simulate FOTC via differencing
- Interactive interference of conjunctive invariants
- [Abstract interpretation is coming]

Ivy: Safety Verification by Interactive Generalization

Artifact * AEC * Evaluated * PLDI * Consistent * Complete * Well Documented * Easy to Reuse

Oded Padon, Kenneth McMillan, Aurojit Panda, Sharon Shoham




TEL AVIV UNIVERSITY

Microsoft Research

Berkeley UNIVERSITY OF CALIFORNIA

TEL AVIV UNIVERSITY

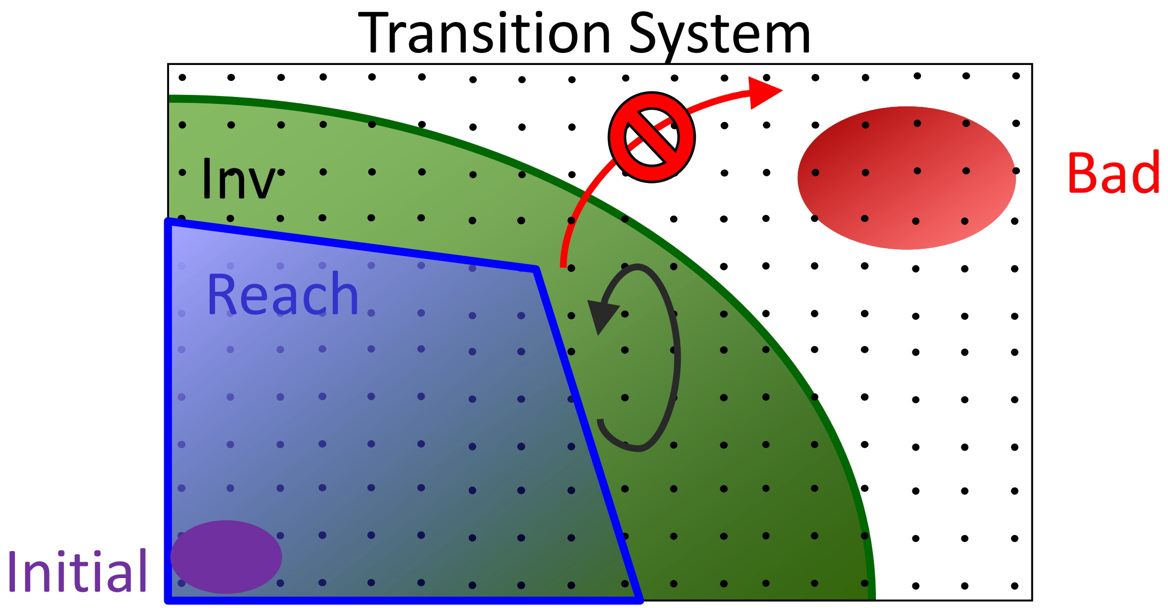
PLDI 2016

http://microsoft.github.io/ivy/

Ivy: Safety Verification by Interactive Generalization

- Verification of distributed systems
- Modeling infinite-state systems in a way which allows decidable automated reasoning (EPR)
- Interactive discovery of inductive invariants

Safety of Transition Systems



System S is safe if no bad state is reachable

System S is safe iff there exists an inductive invariant Inv s.t.:

Init $\subseteq$ *Inv (Initiation)*

if $\sigma \in$ *Inv and* $\sigma \rightarrow \sigma'$ *then* $\sigma' \in$ *Inv (Consecution)*

Inv $\cap$ *Bad* = $\varnothing$ *(Safety)*

Challenges for Deductive Verification

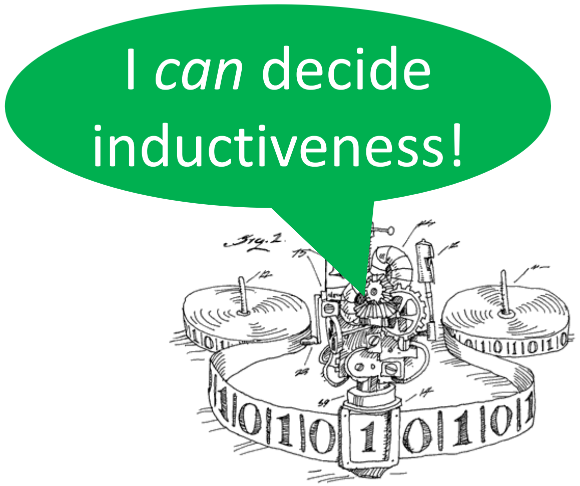
1. Formal specification:
    - Modeling the system
    - Formalizing the safety property
2. Inductive Invariants
    - Hard to specify manually
    - Hard to infer automatically
3. Deduction – Checking inductiveness
    - Undecidability of implication checking
        - Unbounded state, arithmetic, quantifier alternation

Existing Approaches for Verification of Infinite-State Systems

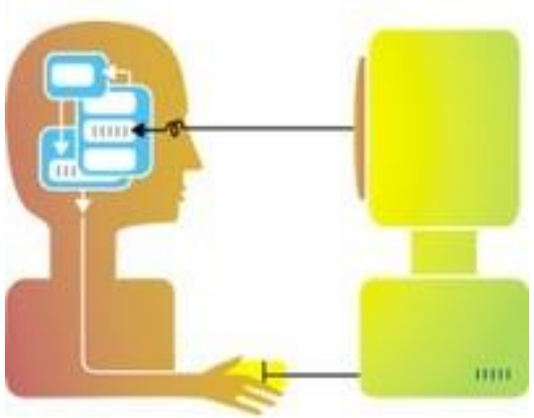
- Automated invariant inference
  - Abstract Interpretation
  - Ultimately limited due to undecidability
- Use SMT for deduction with manual program annotations (e.g. Dafny)
  - Requires programmer effort to provide inductive invariants
  - SMT solver may diverge (matching loops, arithmetic)
- Interactive theorem provers (e.g. Coq, Isabelle/HOL)
  - Programmer provides inductive invariant and proves it
  - Huge effort (10-100 lines of proof per line of code)

Usually opaque when failing

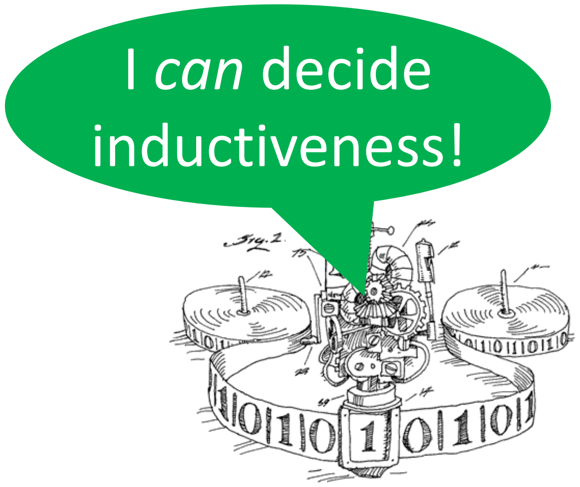
Our Approach in Ivy



- Restrict the specification language for decidability
  - Deduction is decidable with SAT solvers
  - Challenge: verify complex systems using a restricted language
    - Solution: domain specific axioms
- Finding inductive invariants (still undecidable):
  - Combine automated techniques with human guidance
  - Graphical user interaction
  - Key: generalization from counterexamples to induction
  - Decidability allows reliable automated checks



Relational Modeling Language (RML)

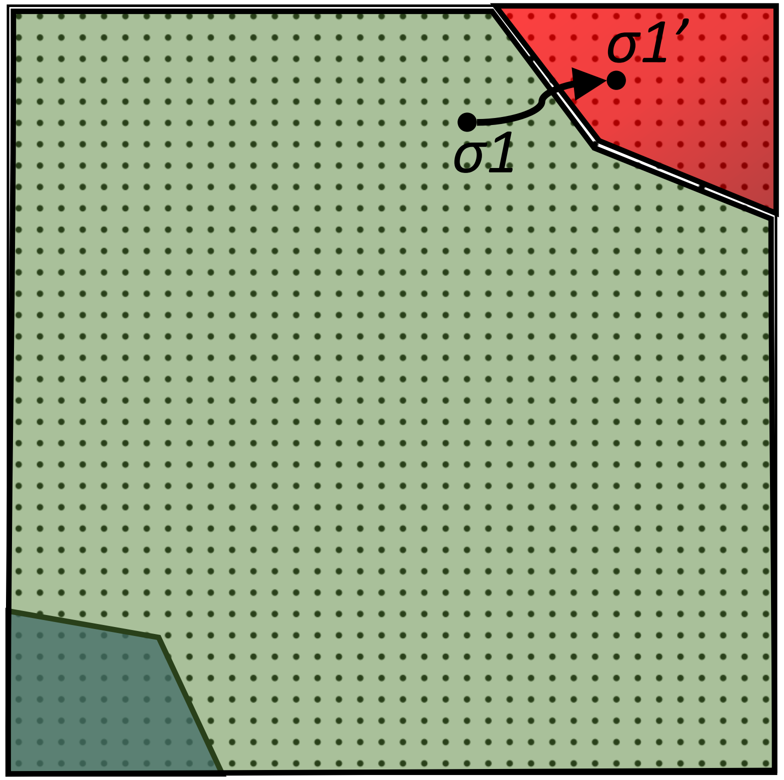


- Designed to make verification tasks decidable
    - Yet expressive enough to model systems
- Turing-Complete
- Universally quantified inductive invariants are decidable to check
- System state described by finite (unbounded) relations
- No numerics
- Simple (quantifier-free) updates
- Universally quantified axioms (domain specific)
    - Total orders, partial orders, lists, trees, rings, quorums, …

Languages for verification

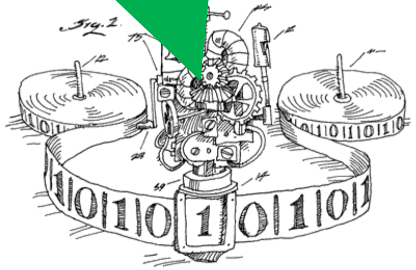
Language	Executable	Deduction	Expresiveness
C, Java, Python…	☑	Undecidable	Turing-Complete
Dafny	☑	Undecidable	Turing-Complete
SMV	☒	Decidable Verification of Temporal properties	Finite-state
Ivy	in progress	Decidable (EPR)	Turing-Complete
Alloy	☒	Undecidable	Turing-Complete
Coq, Isabelle/HOL	☑	Manual	Turing-Complete
TLA+	☒	Manual	Turing-Complete

Invariant Inference In Ivy



Inv = ¬Bad

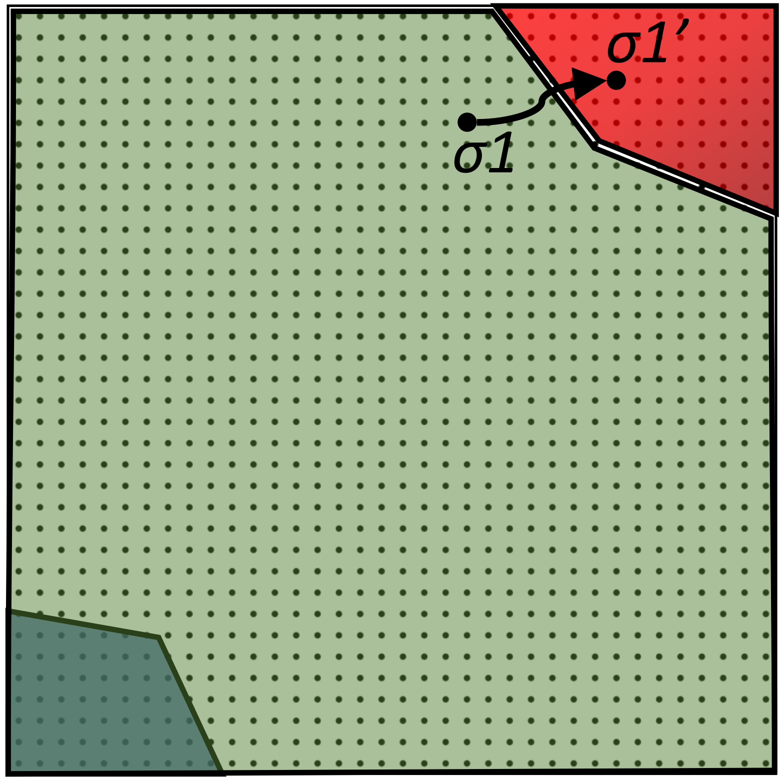
I *can* decide inductiveness!



Check Inductiveness

Counterexample To Induction (CTI)

Invariant Inference In Ivy



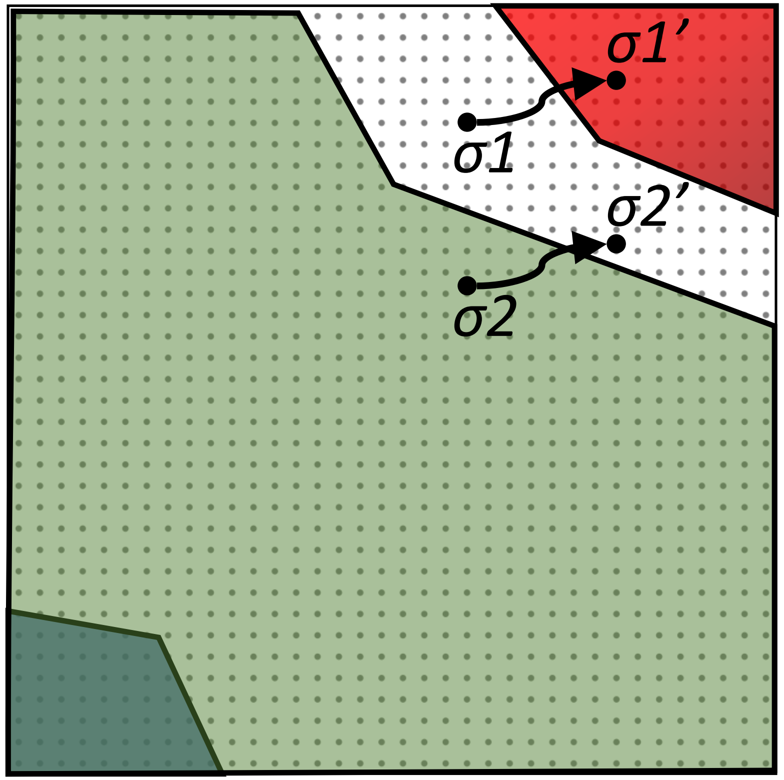
Inv = ¬Bad

σ1,σ1’ – CTI

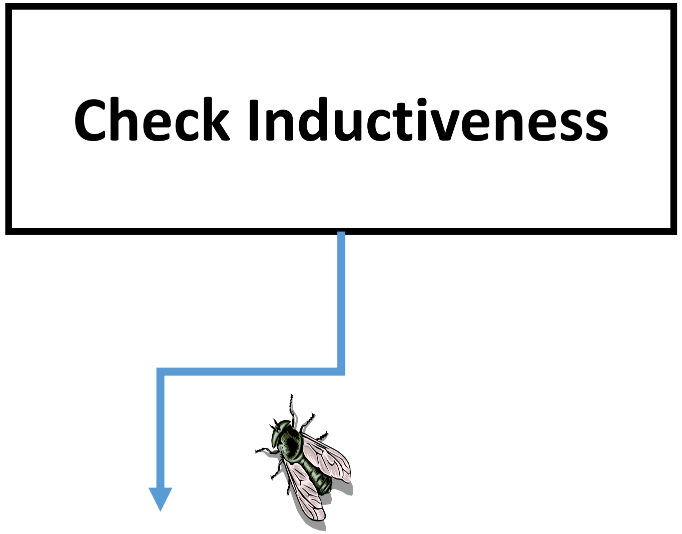
Generalize from CTI

φ(σ1,σ1’)

Invariant Inference In Ivy

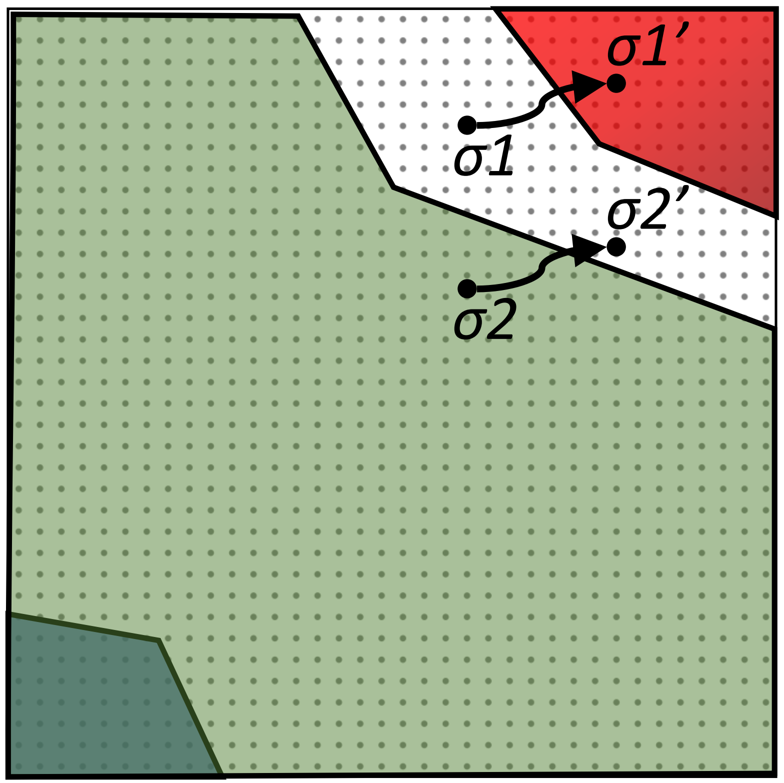


$Inv = \neg Bad \wedge \varphi(\sigma1,\sigma1')$



Counterexample To Induction (CTI)

Invariant Inference In Ivy



Inv = ¬Bad ∧ φ(σ1,σ1’)

σ2,σ2’ – CTI

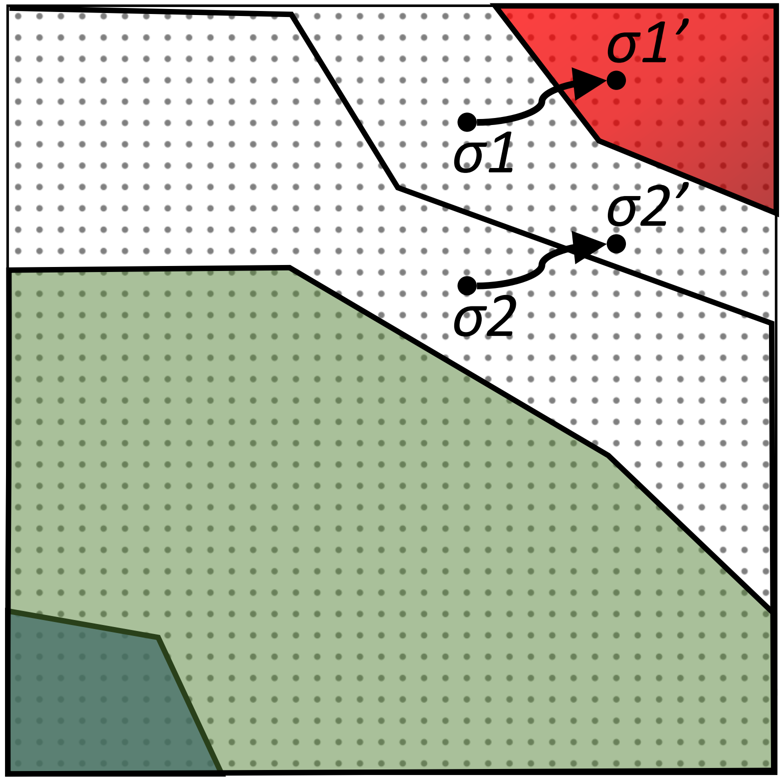
↓

Generalize from CTI

↓

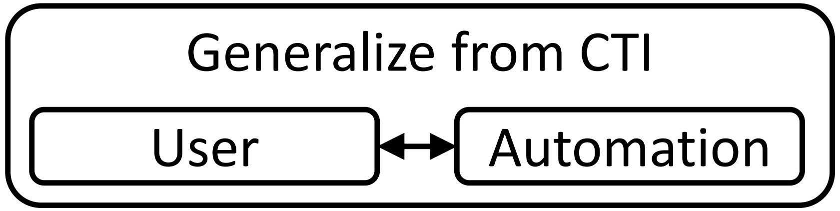
φ(σ2,σ2’)

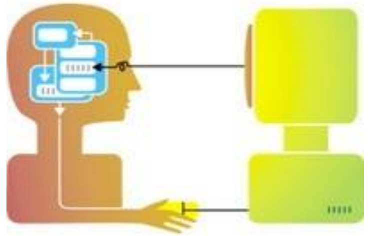
Invariant Inference In Ivy



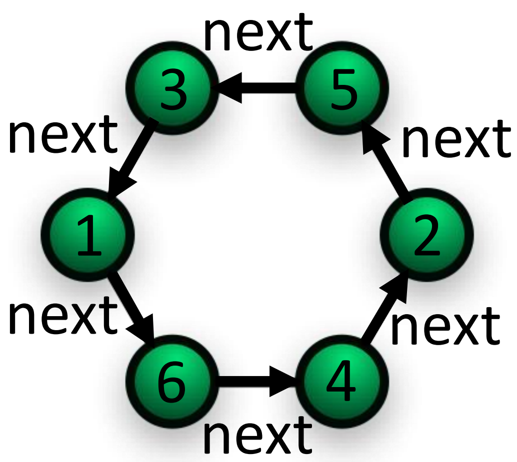
Inv = ¬Bad ∧ φ(σ1,σ1') ∧ φ(σ2,σ2')

- Key challenge for invariant inference: generalization
- Ivy's approach: put the user in the loop *interactive generalization*





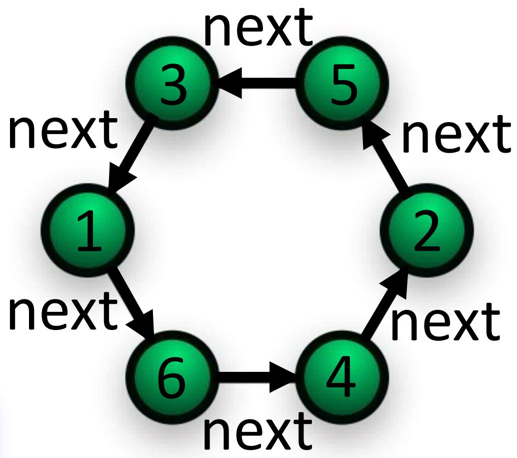
Example: Leader Election in a Ring



- Nodes are organized in a ring
- Each node has a unique numeric id
- Protocol:
  - Each node sends its id to the next
  - A node that receives a message passes it (to the next) if the id in the message is higher than the node's own id
  - A node that receives its own id becomes the leader
- Theorem:
  - The protocol selects at most one leader

[CACM'79] E. Chang and R. Roberts. *An improved algorithm for decentralized extrema-finding in circular configurations of processes*

Example: Leader Election in a Ring



- Nodes are organized in a ring
- Each node has a unique numeric id
- Protocol:
  - Each
  - A no) if the id in the mes
  - A no
- Theore
  - The

Proposition: This algorithm detects one and only one highest number.

Argument: By the circular nature of the configuration and the consistent direction of messages, any message must meet all other processes before it comes back to its initiator. Only one message, that with the highest number, will not encounter a higher number on its way around. Thus, the only process getting its own message back is the one with the highest number.

[CACM'79] E. Chang and R. Roberts. *An improved algorithm for decentralized extrema-finding in circular configurations of processes*

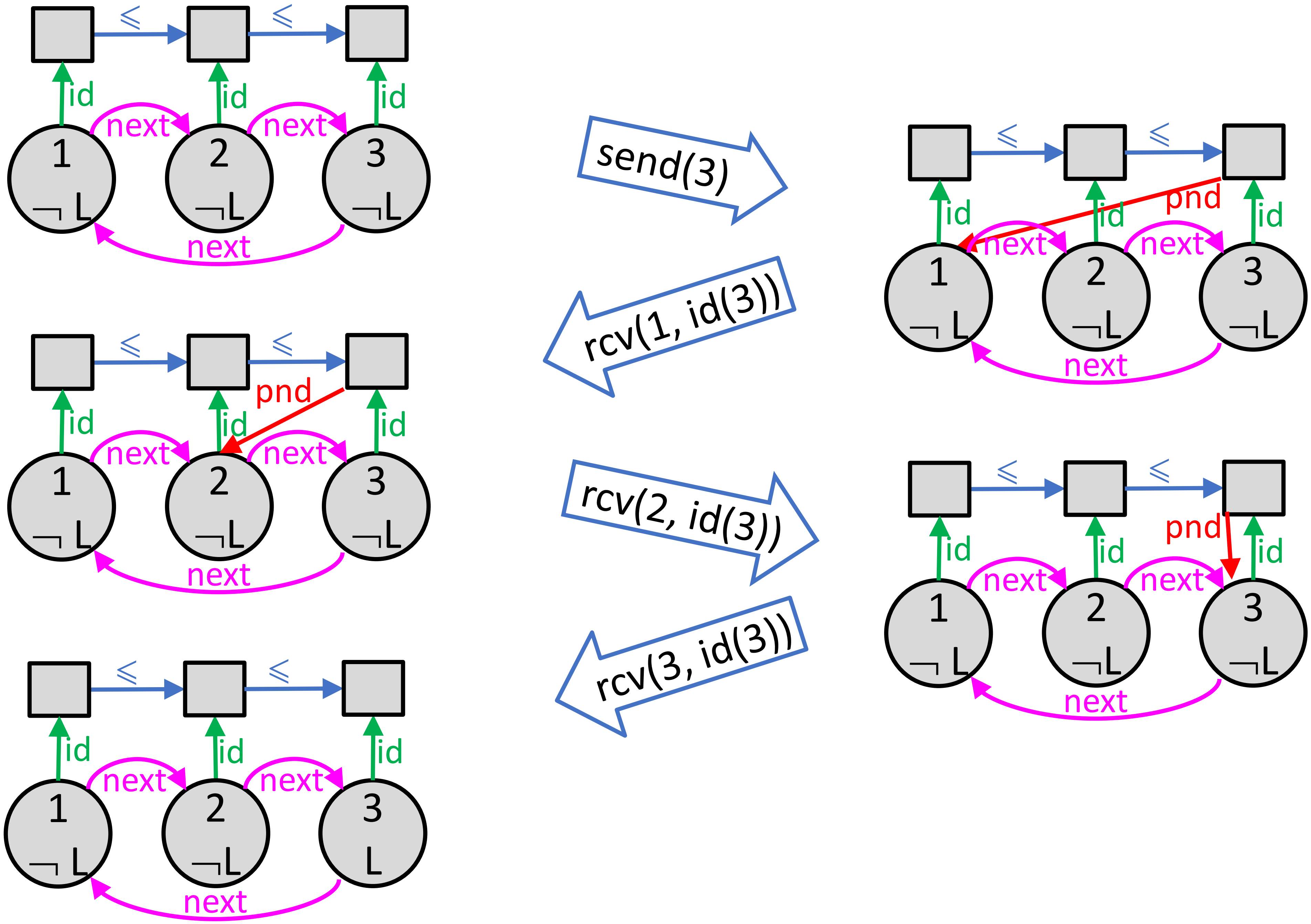
Leader Election Protocol (RML)

- ⩽ (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node → ID – relate a node to its unique id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

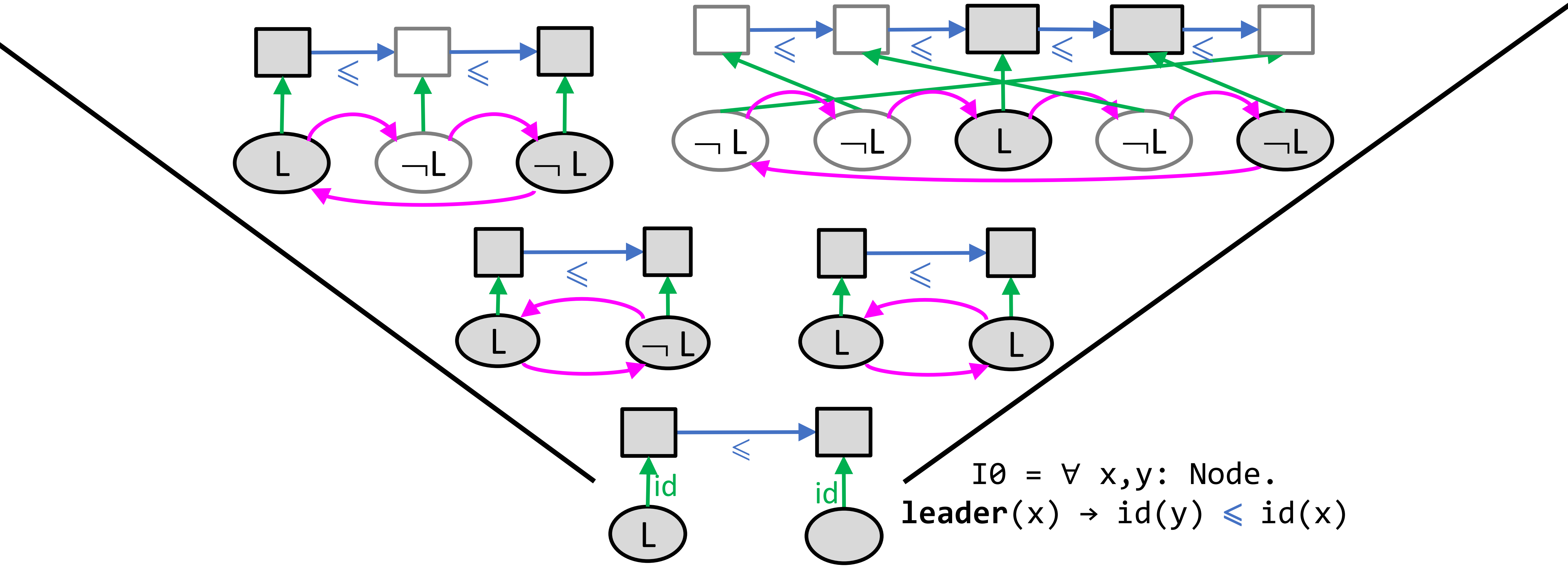
```
action send(n: Node) = {
  "s := next(n)";
  pending(id(n),s) := true
}
```

```
action receive(n: Node, m: ID) = {
  requires pending(m, n);
  pending(m, n) := false;
  if id(n) = m then
    // found leader
    leader(n) := true
  else if id(n) ⩽  m then
    // pass message
    "s := next(n)";
    pending(m, s) := true
}
```

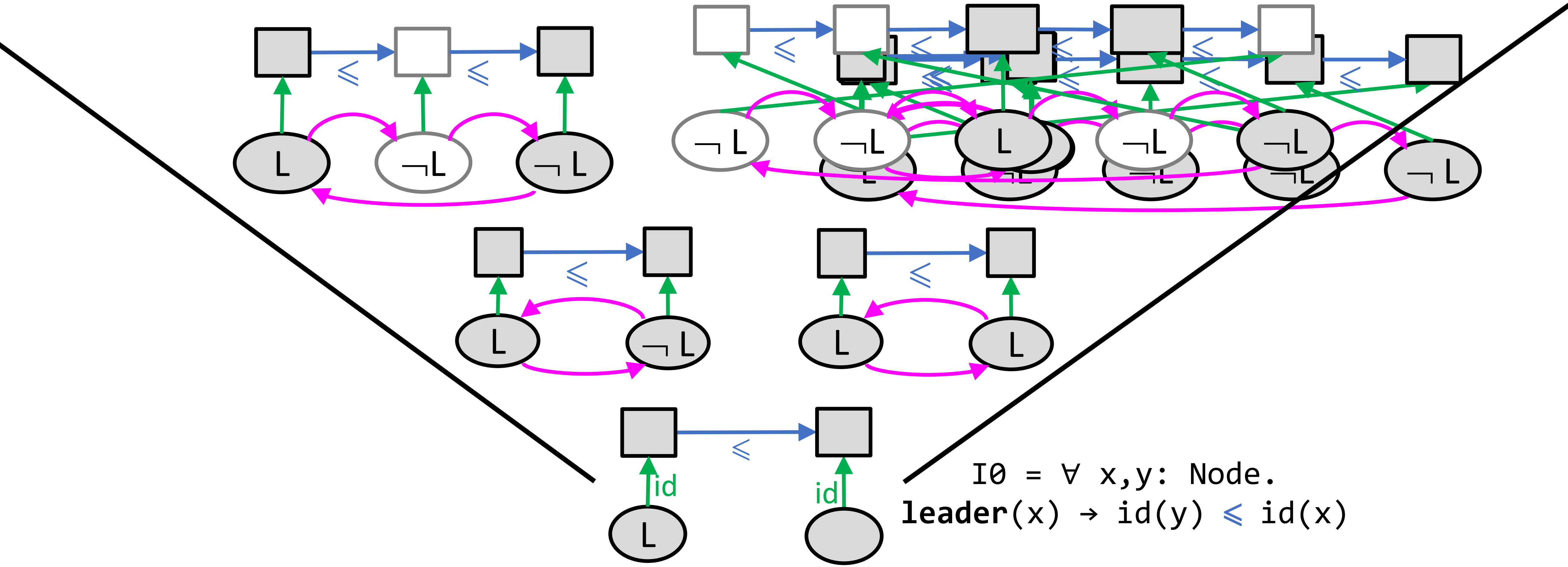
```
protocol = (send | receive)*

next(a)=b ↔ ∀x: Node. x=a ∨ x=b ∨ btw(a,b,x)

assert I0 = ∀ x,y: Node. leader(x) → id(y) ⩽ id(x)
```



send(3)
rcv(1, id(3))
rcv(2, id(3))
rcv(3, id(3))
pnd
id
next
⩽
1
2
3
¬L
L



$$I0 = \forall\, x,y\colon \mathrm{Node}.\ \mathbf{leader}(x) \rightarrow id(y) \leqslant id(x)$$



$$I0 = \forall\ x,y\text{: Node.}$$
$$\textbf{leader}(x) \rightarrow id(y) \leqslant id(x)$$

Inductive Invariant for Leader Election

- $\leqslant$ (ID, ID) – total order on node id's
- **btw** (Node, Node, Node) – the ring topology
- **id**: Node → ID – relate a node to its id
- **pending**(ID, Node) – pending messages
- **leader**(Node) – leader(n) means n is the leader

Safety property: `I0`

`I0 = ∀ x,y: Node. leader(x) → id(y) ⩽ id(x)`

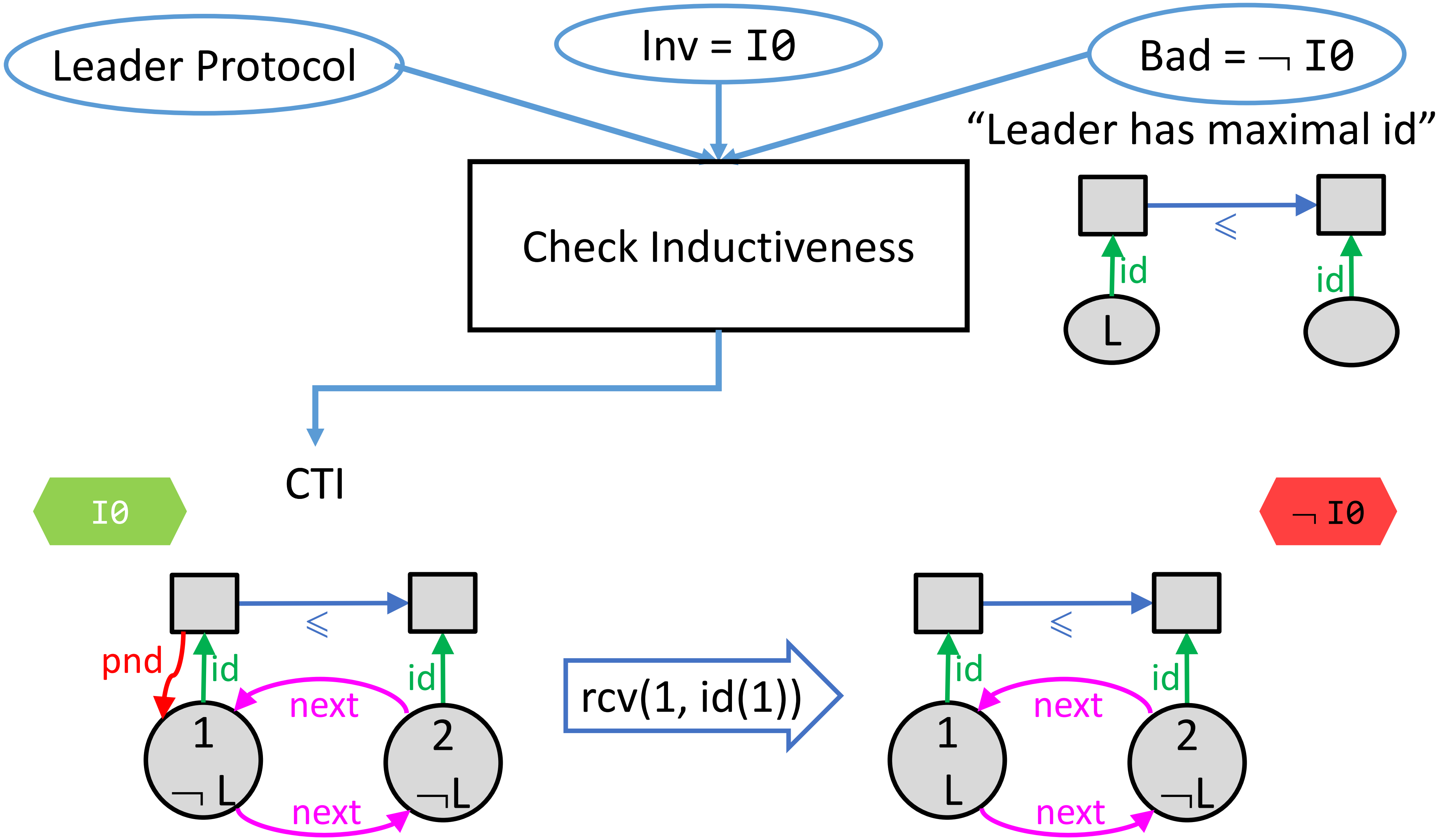
Inductive invariant: `Inv = I0 ∧ I1`

`I1`

`I`

How can we find an inductive invariant without knowing it?

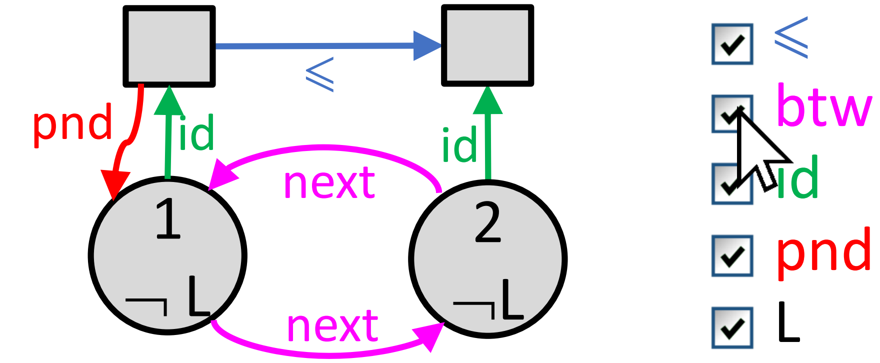
Ivy: Check Inductiveness (1)



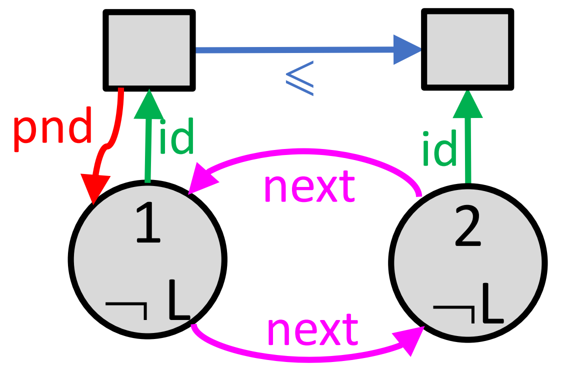
Ivy: Generalize from CTI (1)

1. Each node sends its id to the next
2. A node that receives a message passes it (to the next in the ring) if the id in the message is higher than the node's own id
3. A node that receives its own id becomes the leader

Ivy: Generalize from CTI (1)

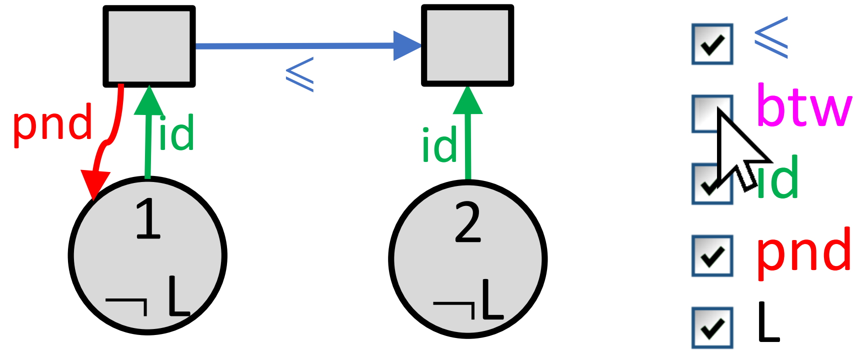


Ivy: Generalize from CTI (1)

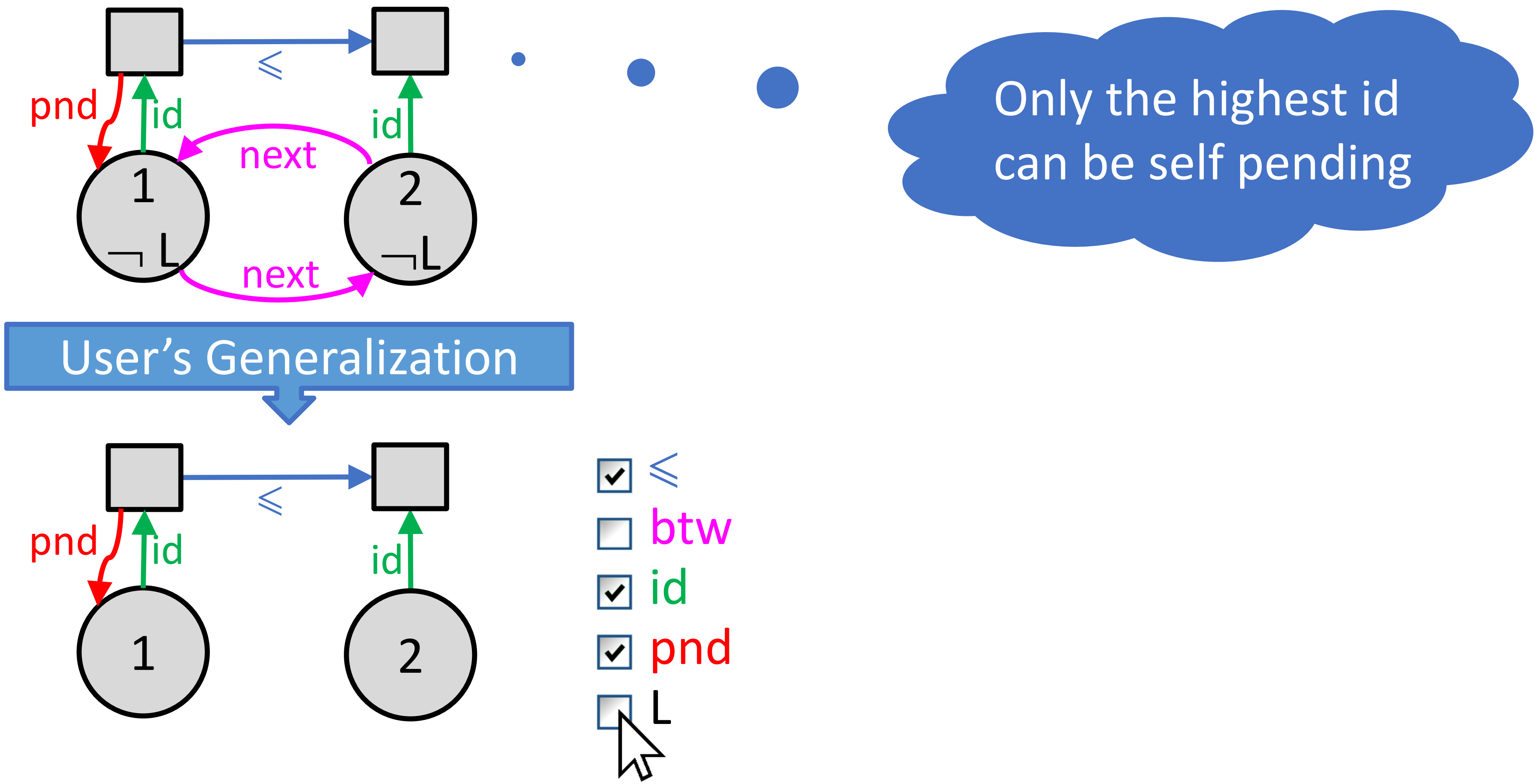


Only the highest id can be self pending

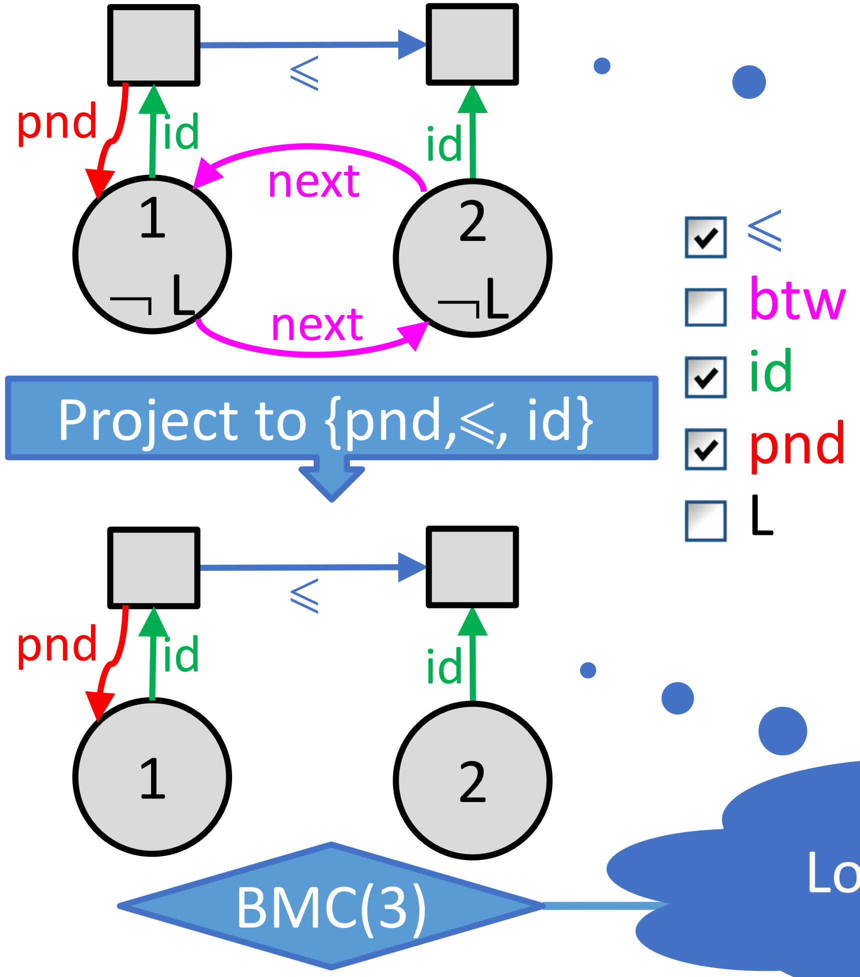
User's Generalization



Ivy: Generalize from CTI (1)



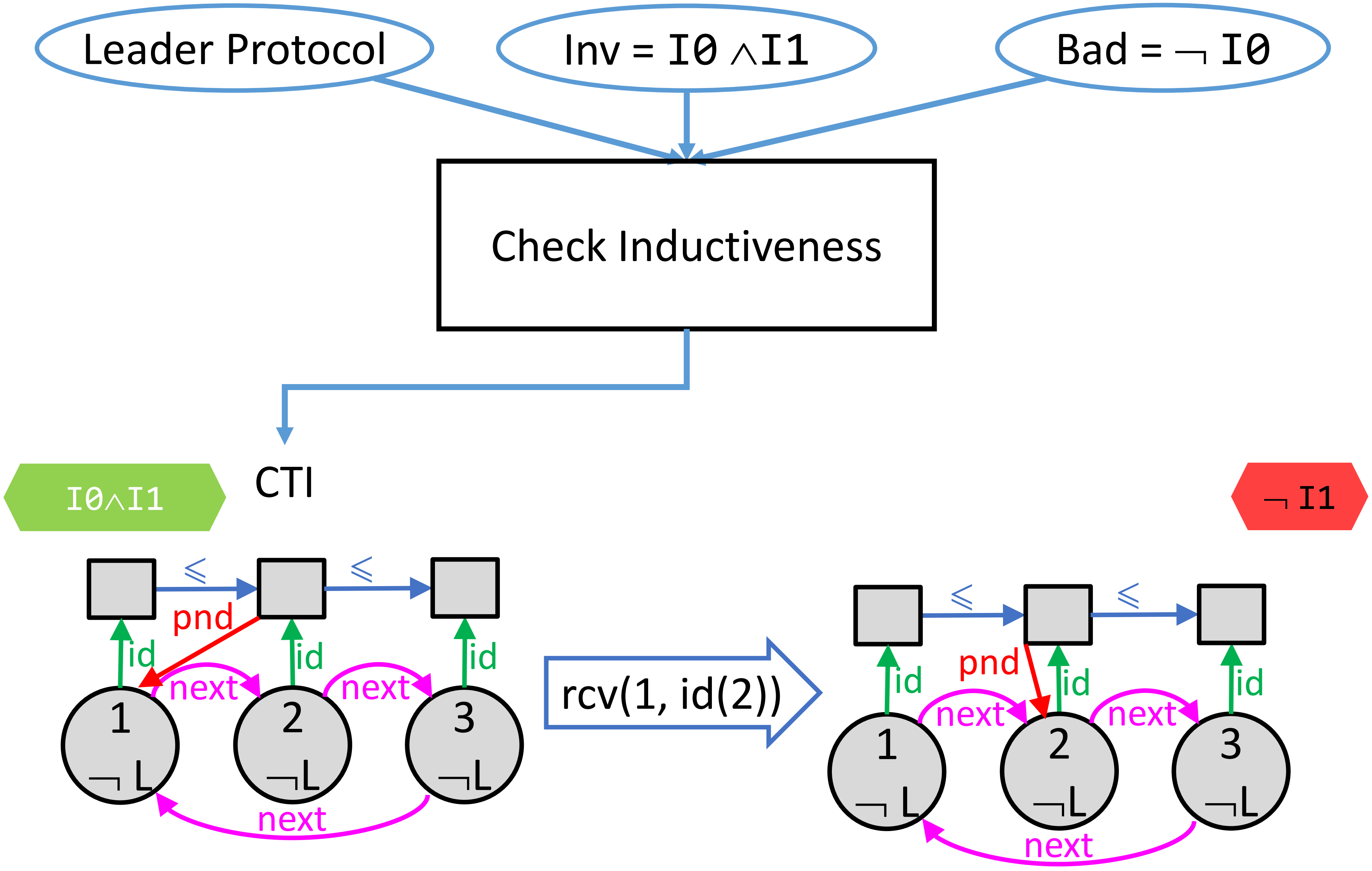
Ivy: Generalize from CTI (1)



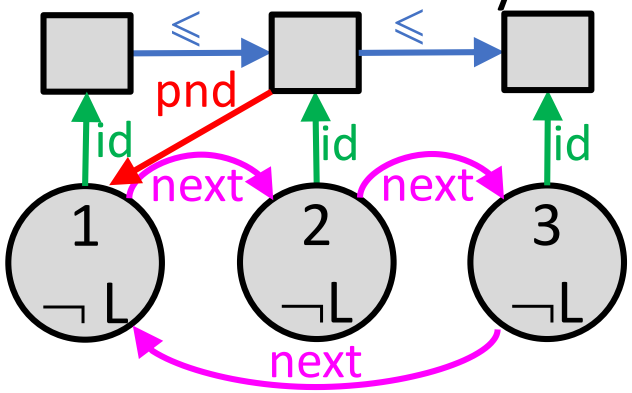
Only the highest id can be self pending

Looks good, add to the invariant as I1

Ivy: Check Inductiveness (2)



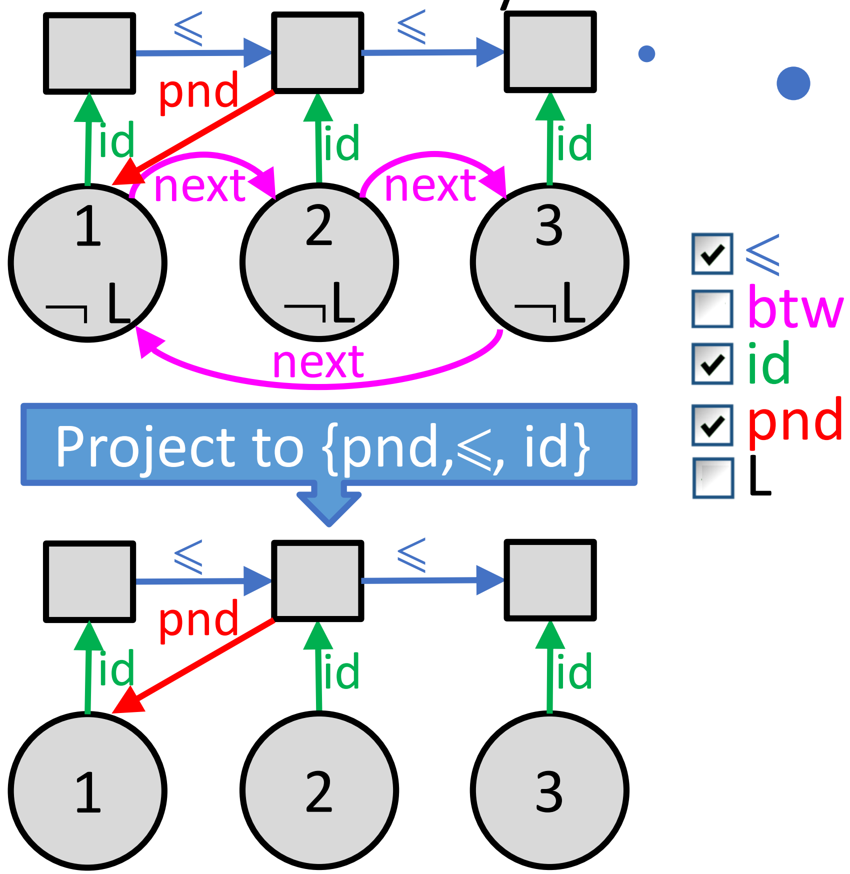
Ivy: Generalize from CTI (2)



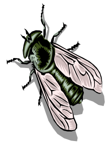
Cannot bypass nodes with higher ids

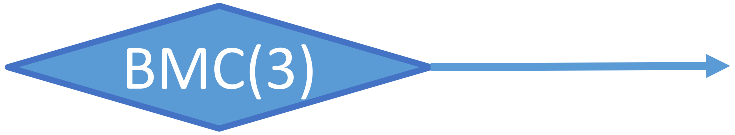
1. Each node sends its id to the next
2. A node that receives a message passes it (to the next in the ring) if the id in the message is higher than the node's own id
3. A node that receives its own id becomes the leader

Ivy: Generalize from CTI (2)



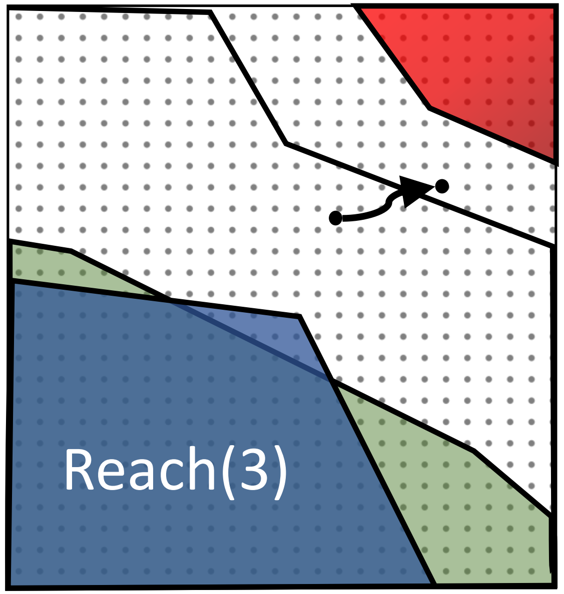
Cannot bypass nodes with higher ids

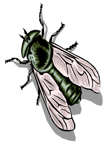


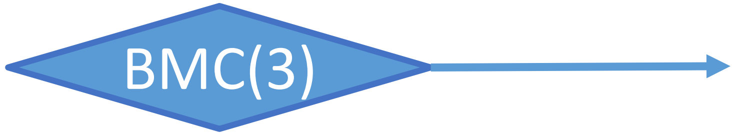


Counterexample Trace

Ivy: Generalize from CTI (2)

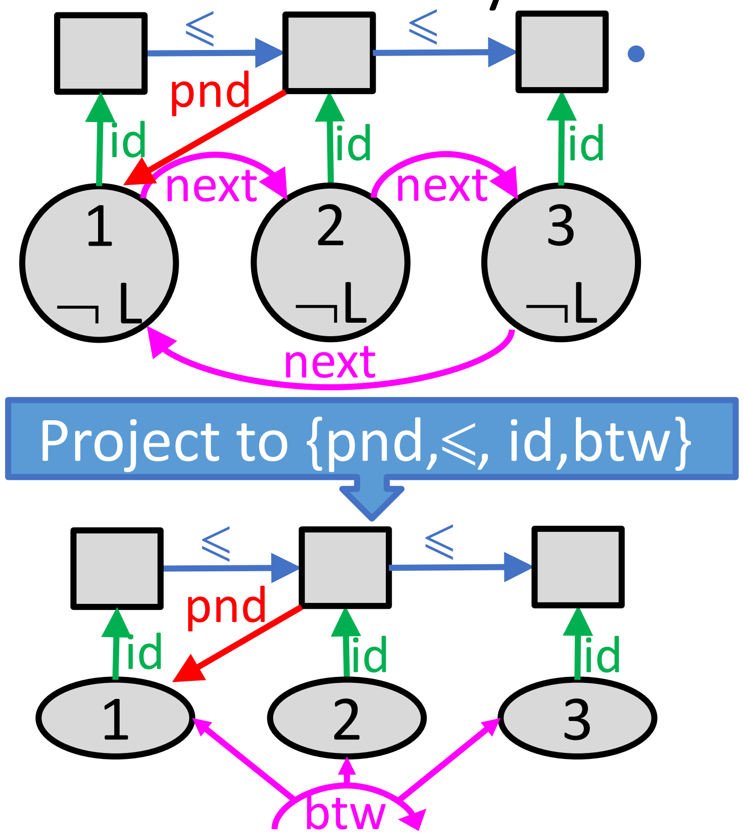






Counterexample Trace

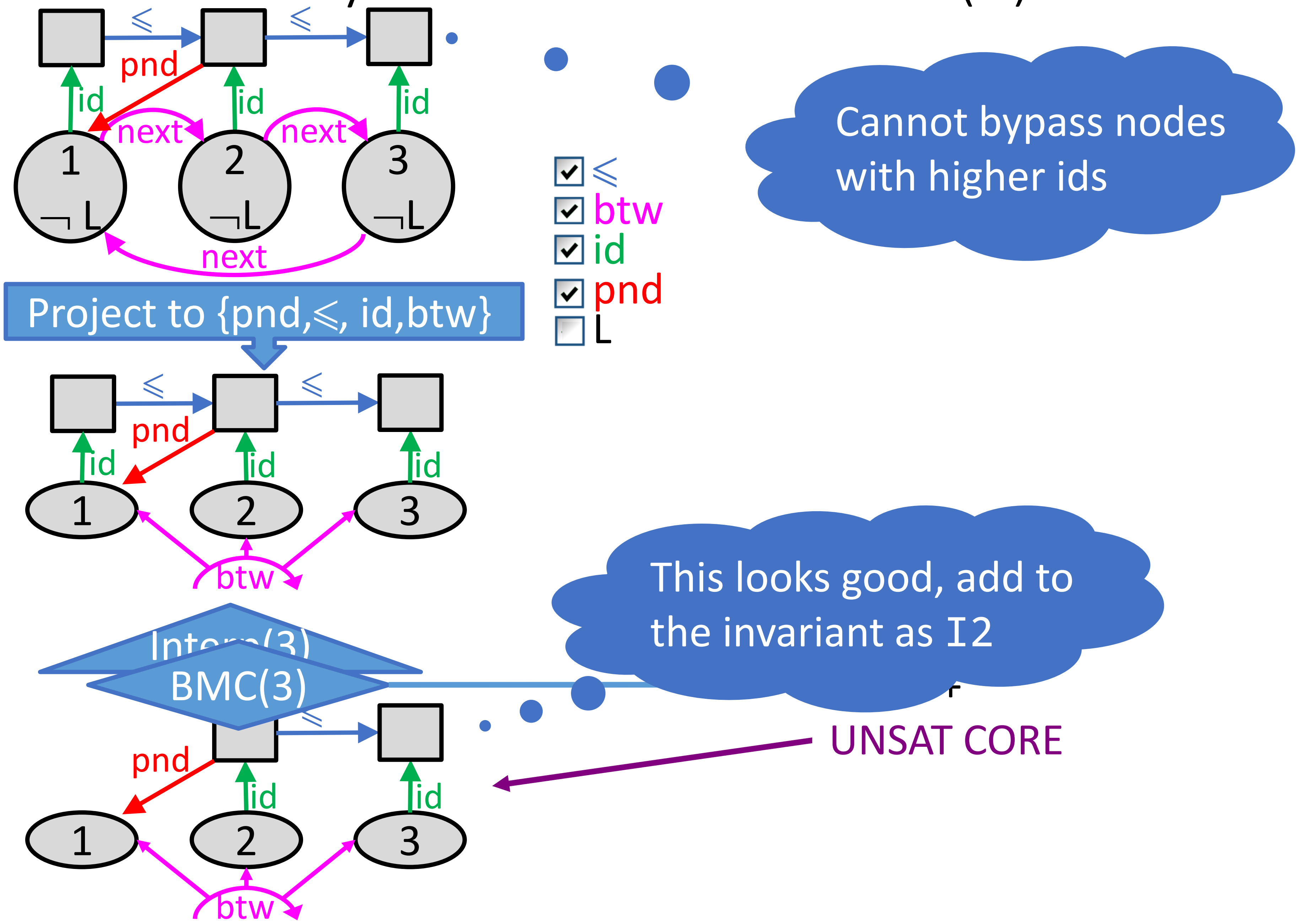
Ivy: Generalize from CTI (2)



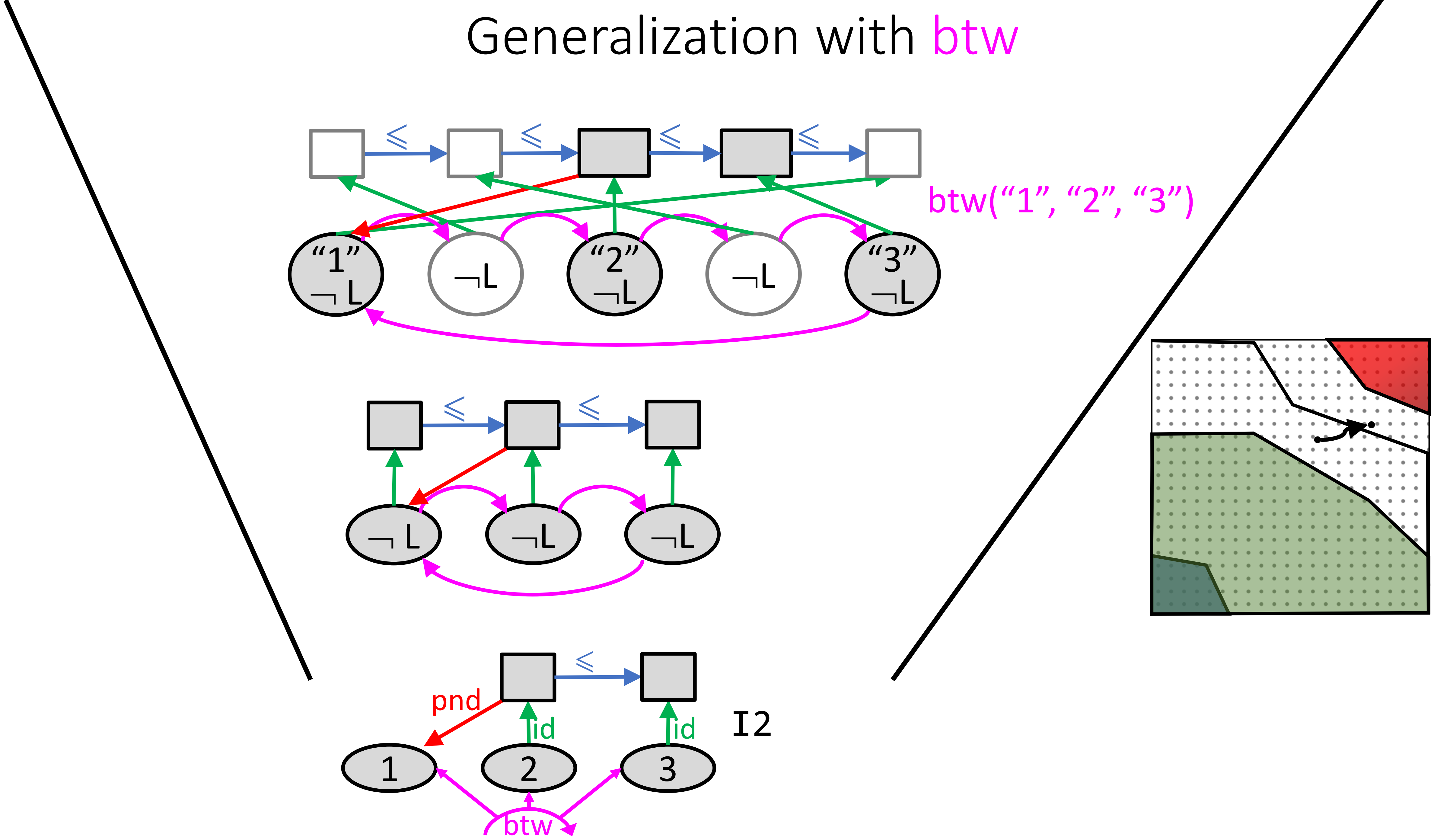
☑ ⩽
☑ btw
☑ id
☑ pnd
☐ L

Cannot bypass nodes with higher ids

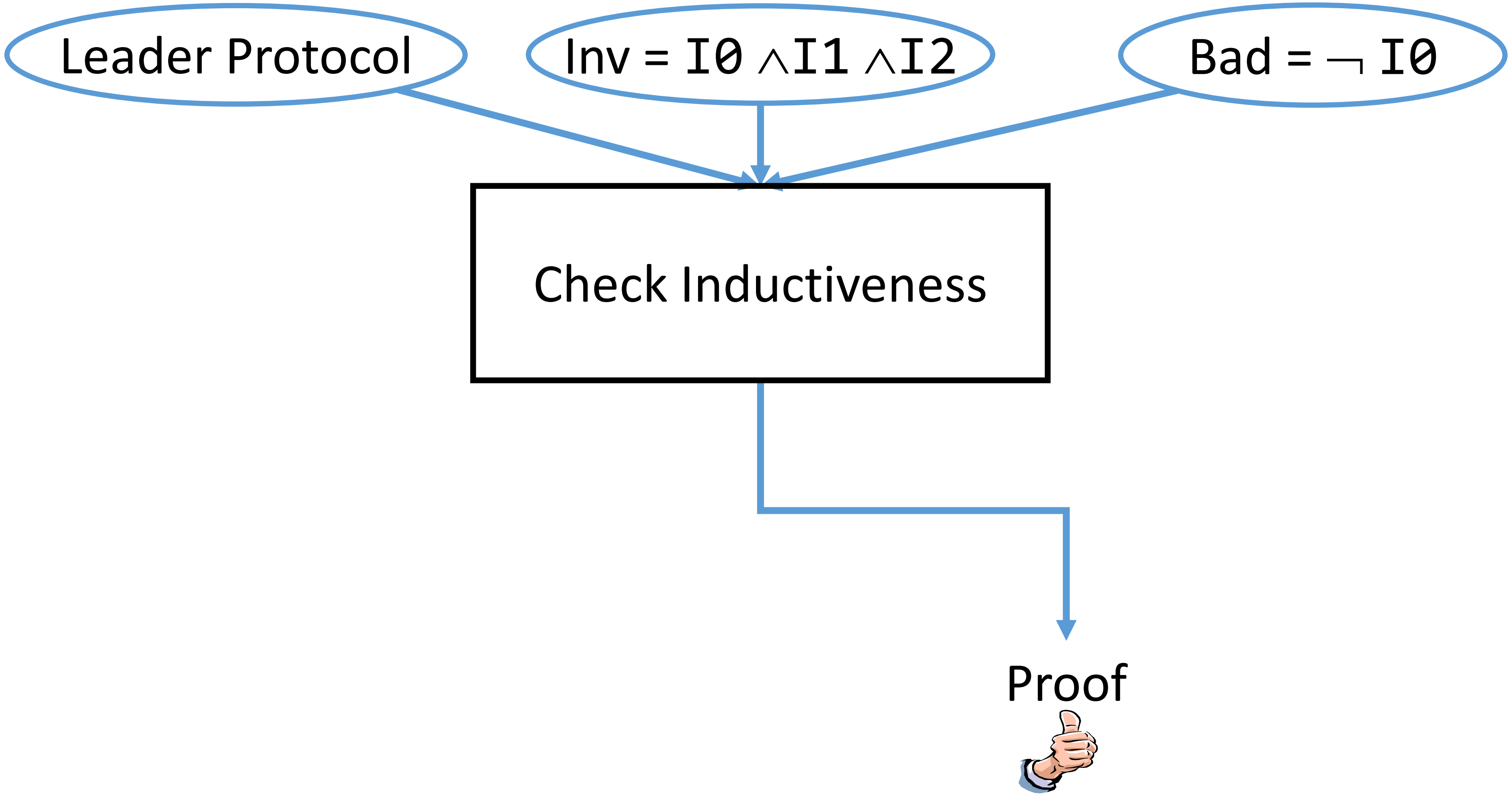
Ivy: Generalize from CTI (2)

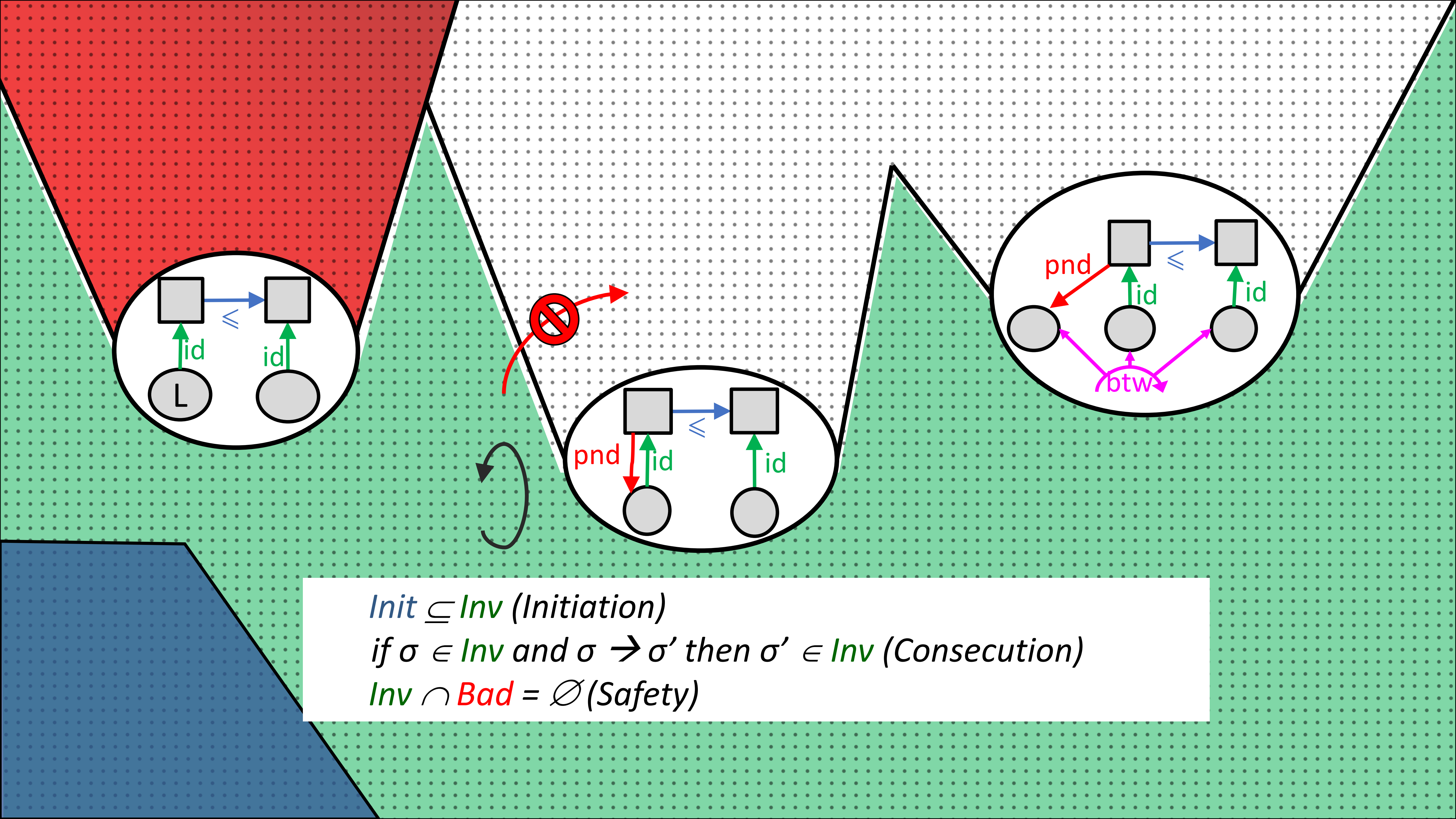


Generalization with btw



Ivy: Check Inductiveness (3)





$Init \subseteq Inv$ (Initiation)

if $\sigma \in Inv$ and $\sigma \rightarrow \sigma'$ then $\sigma' \in Inv$ (Consecution)

$Inv \cap Bad = \emptyset$ (Safety)

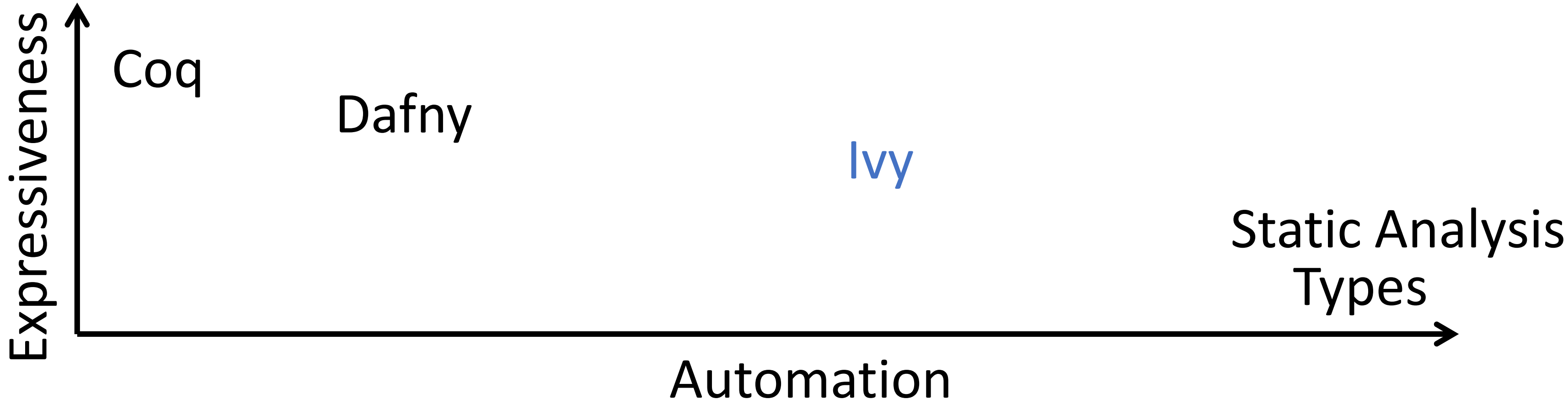
Completeness and Interaction Complexity

- Any generalization from CTI adds one universally quantified clause
- A universally quantified invariant in CNF with N clauses, can be obtained by the user in N generalization steps
  - Assuming the user is optimal
- If the user is sub-optimal, backtracking (weakening) may be needed

Verified Protocols

Protocol	Model Types	Relations & Functions	Property (# Literals)	Invariant (# Literals)	CTI Gen. Steps
Leader in Ring	2	5	3	12	3
Learning Switch	2	5	11	18	3
DB Chain Replication	4	13	11	35	7
Chord	1	13	35	46	4
Lock Server 500 Coq lines [Verdi]	5	11	3	21	8 (1h)
Distributed Lock 1 week [IronFleet]	2	5	3	26	12 (1h)
Paxos	Work in progress				
Raft					

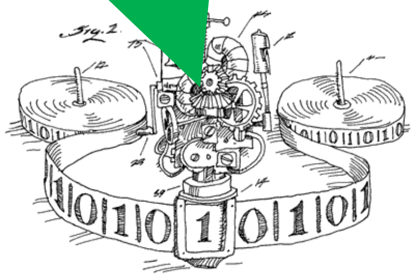
Expressiveness vs. Automation



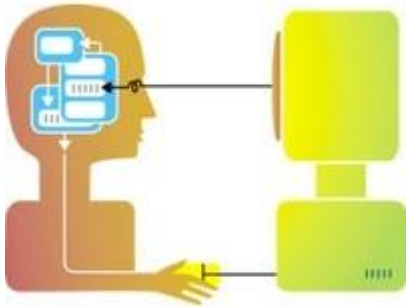
	Coq	Dafny	Ivy	Fully Automatic Static Analysis
Invariant	User	User	User + System	System
Deduction	User	System (Z3) + “User”	System (EPR Z3)	System

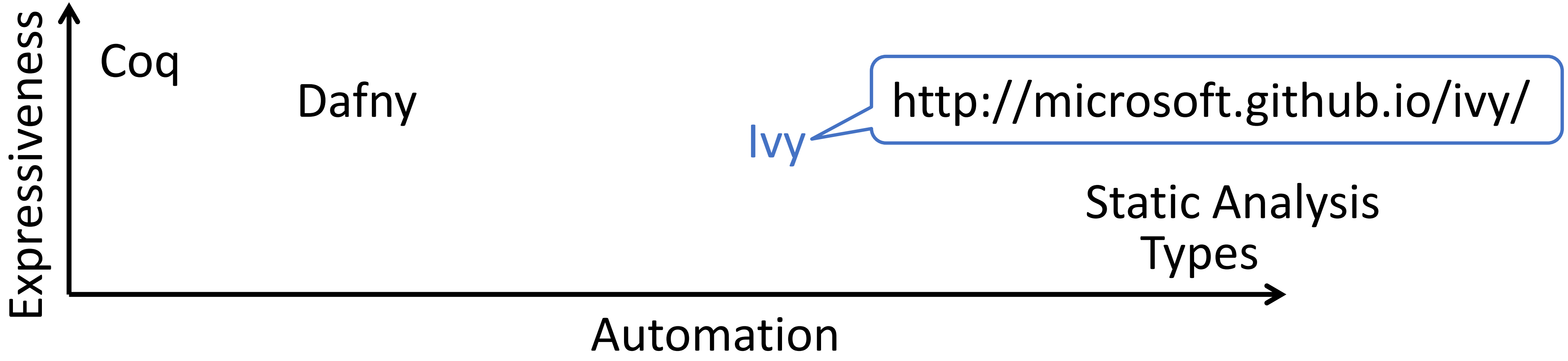
Summary

I *can* decide inductiveness!



- RML – modeling language that makes deduction decidable
  - Many systems can be verified (axioms for orders, trees, rings, …)
- Interactive generalization for finding inductive invariants
- Application to the domain of distributed protocols
- User intuition and machine heuristics complement each other:
  - User has intuition that leads to *better generalizations*
  - Machine is better at finding bugs and corner cases
- Interactive process assists user to gain intuition about the protocol





4 Lessons Learned from Tom

- Spoonfeed the reader
- Look the other way
- Think deeply
- Dedication/Dedication/Dedication

Thanks

James Cheney

Xavier Rival