I4: Incremental Inference of Inductive Invariants for Verification of Distributed Protocols

Presenter: Kai Ma <ksqsf@mail.ustc.edu.cn>

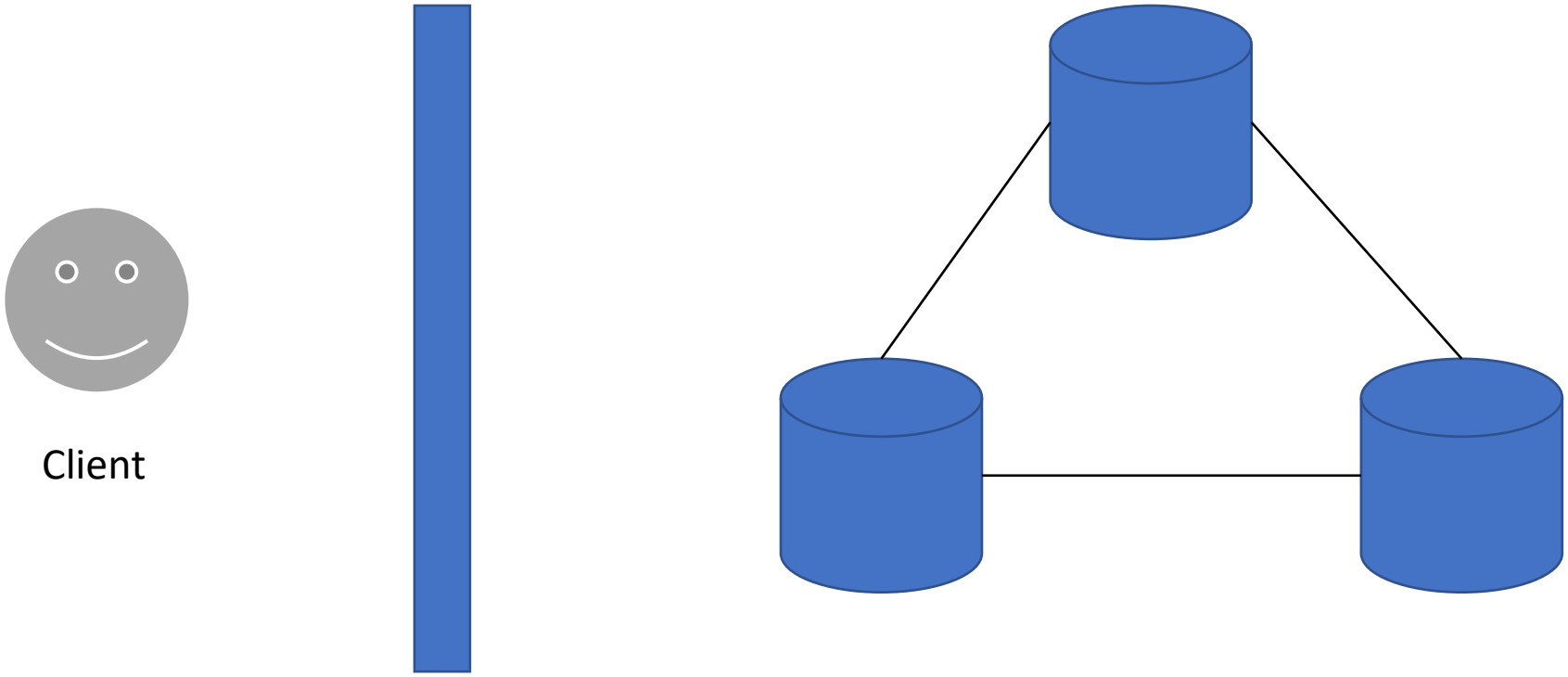
Dec. 4th, 2019

To verify is human;
To prove, divine.

Agenda

- Background
  - Challenge: *write* a correct distributed system
  - Challenge: *prove* a distributed protocol is correct
  - What does *invariant* mean?
  - What does *inductive* mean?
- Introduction to I4
  - Motivation: *incremental*
  - Methodology: *inference*
  - Effectiveness

Distributed Systems, in a Nutshell

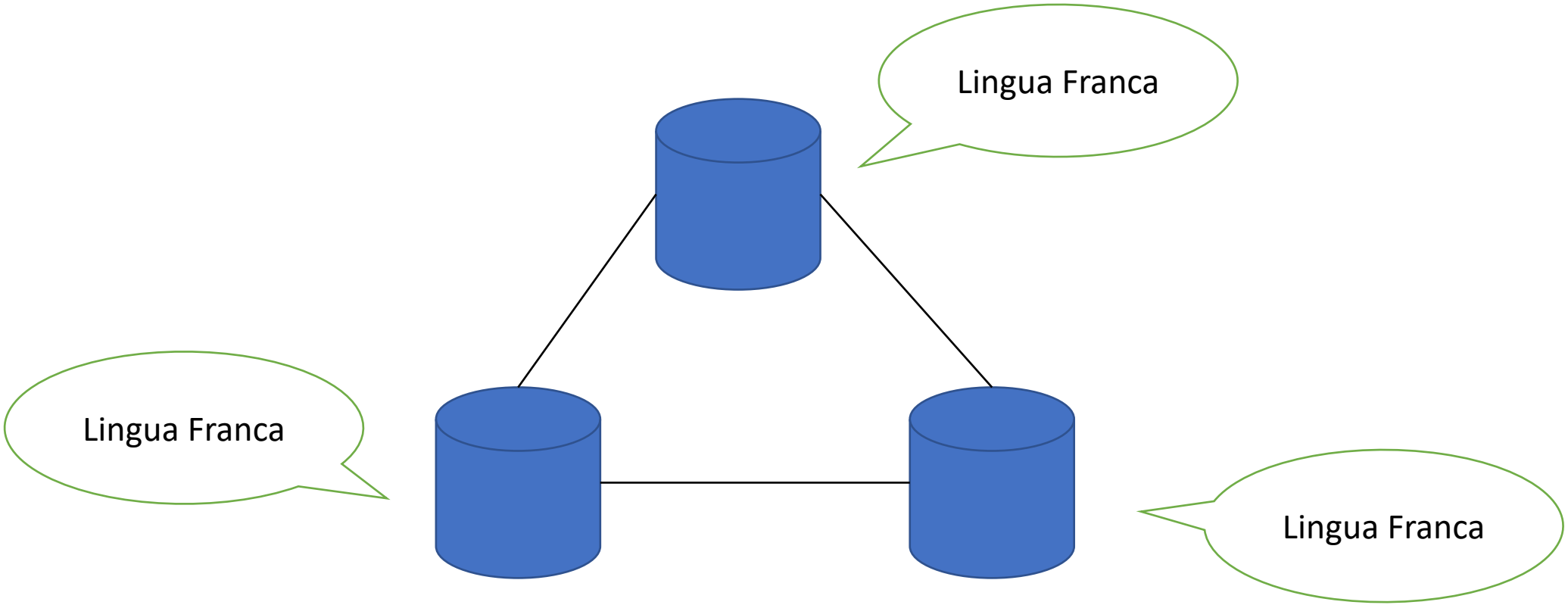


Client

“The Fourth Wall”

Servers / Clusters / Data Centers / …

Distributed Protocols



Mutual Exclusion

- At any time, there should be less than one computer using the printer…

$$\forall i, j \in P, (c[i] = \text{True} \land c[j] = \text{True}) \implies i = j$$

Dijkstra's Algorithm

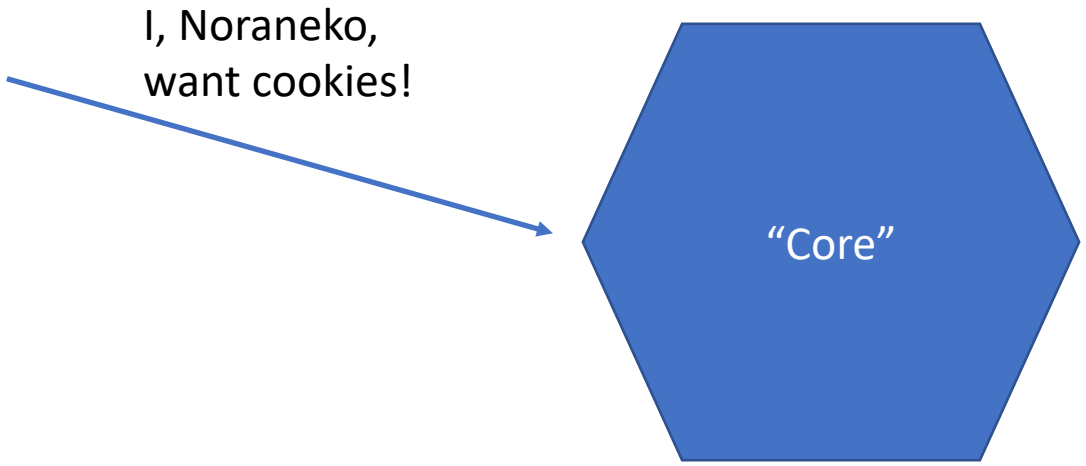
- Wow, impressive
- Is it correct?

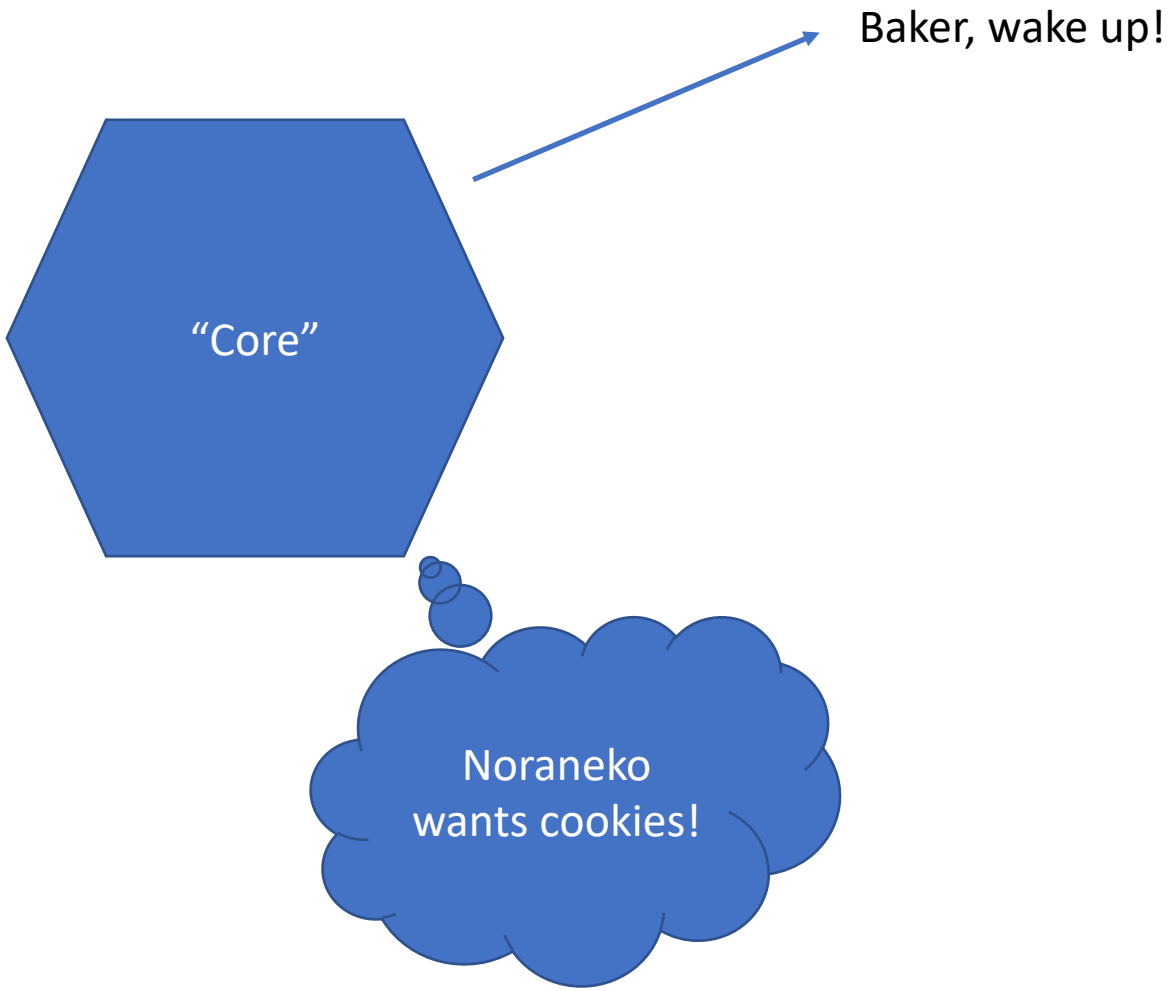
```
public void run() {
    spin:
    while (true) {
        spinning[id] = true;
        spinning = spinning;
        while (candidate != id) {
            ready[id] = false;
            ready = ready;
            if (!spinning[candidate]) {
                candidate = id;
            }
        }
        ready[id] = true;
        ready = ready;
        for (int i = 0; i < ready.length; i++) {
            if (i != id && ready[i]) {
                continue spin;
            }
        }
        criticalSection.run();
        ready[id] = false;
        ready = ready;
        spinning[id] = false;
        spinning = spinning;
    }
}
```

What should we do?

- More testing
  - increases our confidence in the software deployed, but we will never know if there are more bugs to discover…
- Prove our system has no bugs!
  - What do you mean by “no bugs”?
  - Aaaaand, what do you mean by “prove”?

- **Insight:** a server is driven by messages


I, Noraneko,
want cookies!
“Core”


Baker, wake up!
“Core”
Noraneko
wants cookies!

Modeling the whole system

- The state space of the whole system = Cartesian product of all its components
- **State Machines**
  - Initial states
    - One of $s_{00}, s_{01}, s_{02}, \ldots$
  - State transitions
    - Functions $f: S \to S$
- Desirable properties
  - **Safety**: a system never does what it isn't supposed to do
  - **Correctness**: a system does what it is supposed to do
  - **Liveness**: a system always makes progress

Invariants

- Some properties are desired
  - A digital telephone line should never be hijacked
  - A bomb should never explode when it's not okay
- If, for every reachable state s, $P(s)$ holds, then P is called an *invariant*
- Write down what you want as invariants
- Prove them
- If you can't, then your protocol/algorithm/… is likely to be wrong!

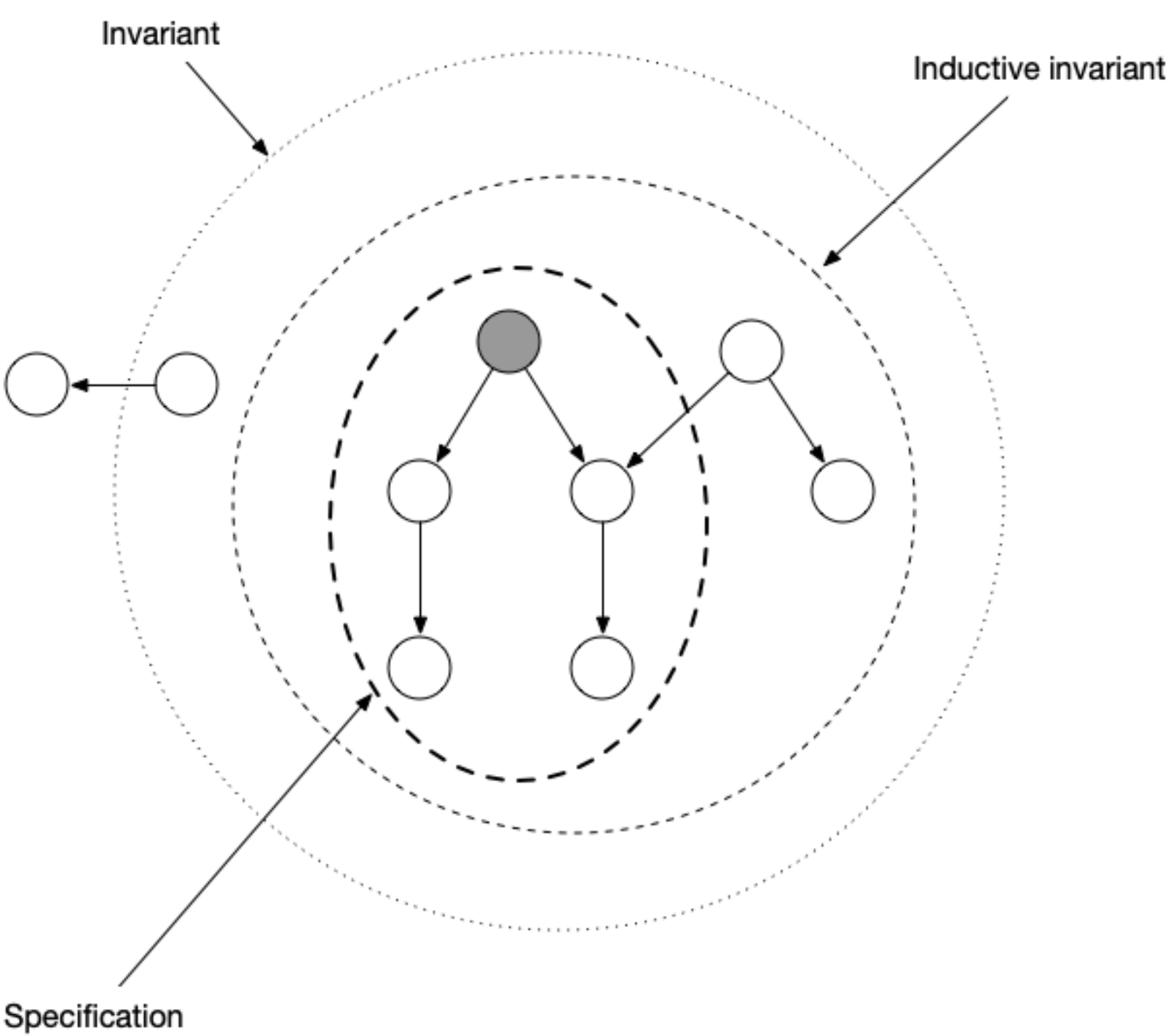
How to prove?

- Note that a system can often have
  - infinitely many states,
  - infinitely many transitions,
  - and infinitely many (wrong) behaviors and outcomes
- Let's try to make the properties "transitive"
  - If the property P holds for initial states
  - For all transitions $s \rightarrow t$, if we can prove $P(t)$ when $P(s)$ holds
  - Then property P holds for all states
- Such invariants are "inductive"

Induction

- Peano axioms, fifth
- If
  - P(0)
  - if P(i) then P(i+1)
- Then
  - For all natural number n, P(n) holds

- Structural induction
- If
  - P(s0) for all initial states s0
  - if P(s) then P(t) when s->t
- Then
  - For all reachable state s, P(s) holds


Invariant
Inductive invariant
Specification

Model Checking

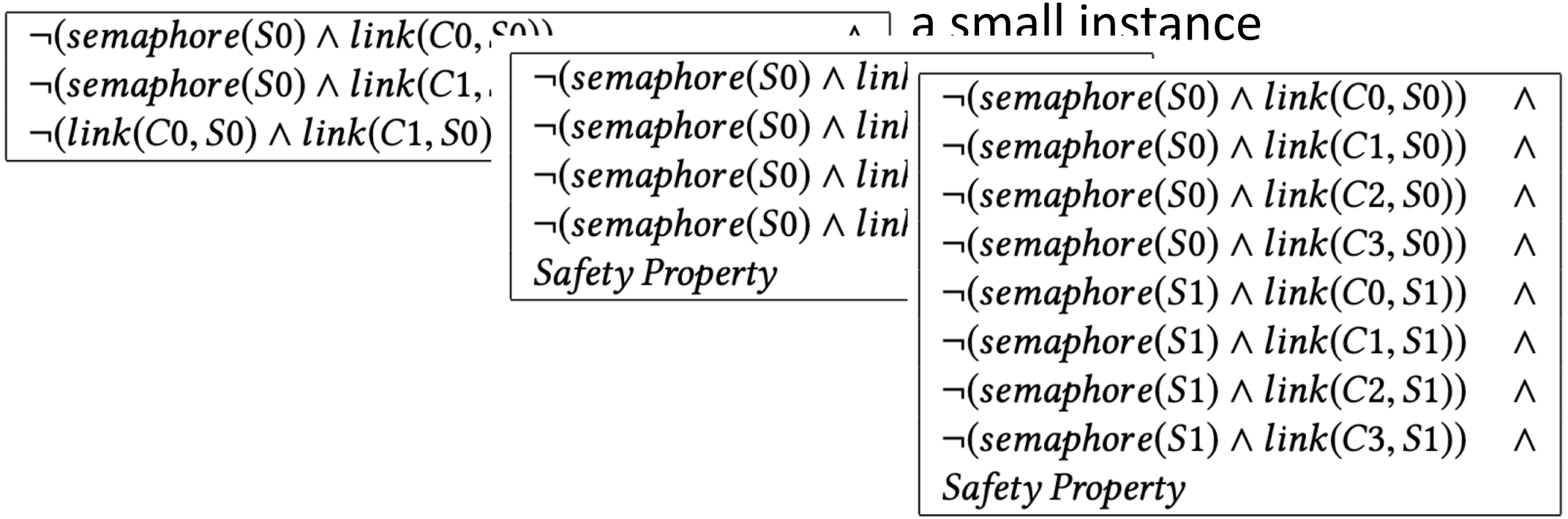
- Given a specification, check if the properties are satisfied
- Approach 1: Brute force
  - Enumerate all possible states and verify invariants on them one by one
  - E.g.: TLC
- Approach 2: SAT-based
  - Turn to logic calculi for help
  - E.g.: IC3, Ivy

Facts about Model Checking

- There are infinitely many servers, clients, states, variables, values…
- Given a **finite** model, we can enumerate all possible states
  - The size of the state space *exponentially* grows with the size of the model
  - Only tractable for finite and *small* models
  - Yes, we cannot handle infinity
- Given **inductive** invariants, SAT-based provers cannot handle **quantifiers** nicely
  - Domains are usually *infinite*
  - More often than not, these fragments of logic are *undecidable*

Observation

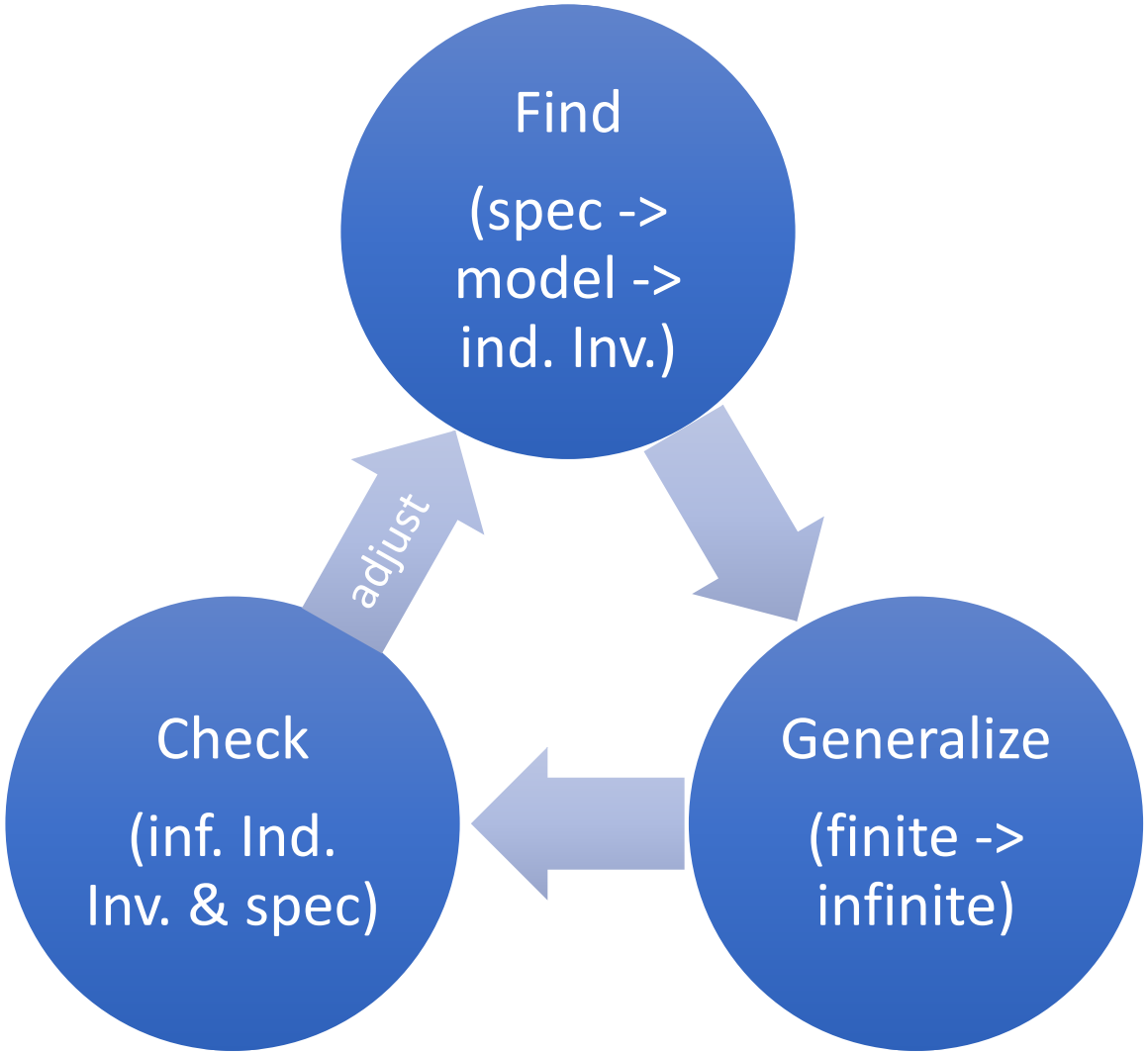
- Observe the inductive invariants of finite and small models
  - The size grows, but not the complexity
  - High regularity



Insight


Find an inductive invariant for a small case
Generalize
Check

Automate


Find
(spec -> model -> ind. Inv.)
Generalize
(finite -> infinite)
Check
(inf. Ind. Inv. & spec)
adjust

Some Problems

- How big should the small case be?
  - Unanswered, but the intuition is there
- Even small cases are intractable!
  - Add axioms to guide the checker (concretization)
- What if the checker fails to prove the inductive invariant?
  - Included *instance-specific* clauses -> prune them
  - Model too small so the generalized invariant is not inductive -> increase size

Methodology

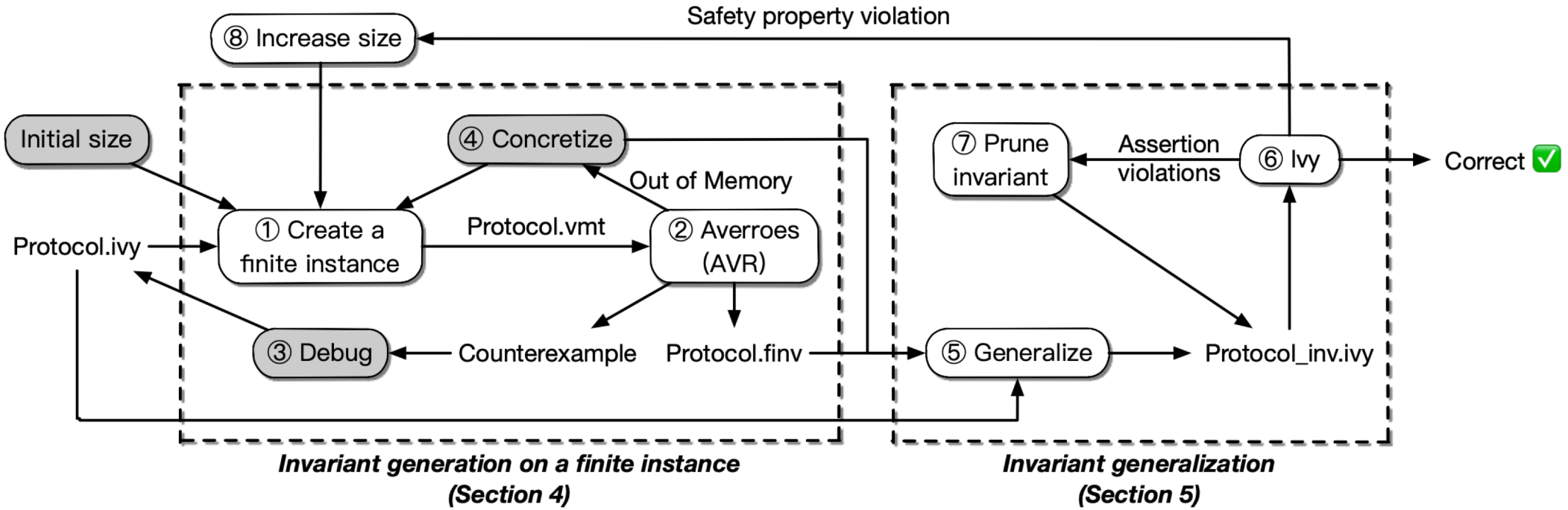


Figure 1. Flow of I4. White boxes are fully automated, while gray boxes denote manual effort.

Demonstration

Protocol	Traditional approach	Ivy	I4
Lock server	500 lines (Verdi)	<1 hour	Automated
Distributed lock	A few days (IronFleet)	A few hours	< 5 min

Demonstration

Protocol	F	M	G	total
Lock server	0.02	0.0	0.8	0.8
Leader election in ring	4.0	0.1	2.0	6.1
Distributed lock protocol	30.6	53.3	75.5	159.5
Chord ring maintenance	386.1	218.5	24.3	628.9
Learning switch	2.9	0.8	6.9	10.7
Database chain replication	4.2	2.3	6.2	12.6
Two-Phase Commit	2.6	0.1	1.6	4.3

Table 4. Runtime results (in seconds). F is the time required to find the finite inductive invariant; M is the time is takes to minimize the finite inductive invariant; and G is the time to generalize the clauses and perform invariant pruning.

Conclusion

- Regularity matters
- By combining existing technology, we can achieve a high level of automation in verification of distributed protocols