Ranking Formal Specifications using LLMs

Mike He, Zhendong Ang, Ankush Desai, Aarti Gupta

SPLASH/LMPL'25, October 15

$$\text{Impl} \models \text{Spec}$$

The heart of formal reasoning

Impl ⊨ Spec

Hardware

`request |=> ##[1:3] ack`

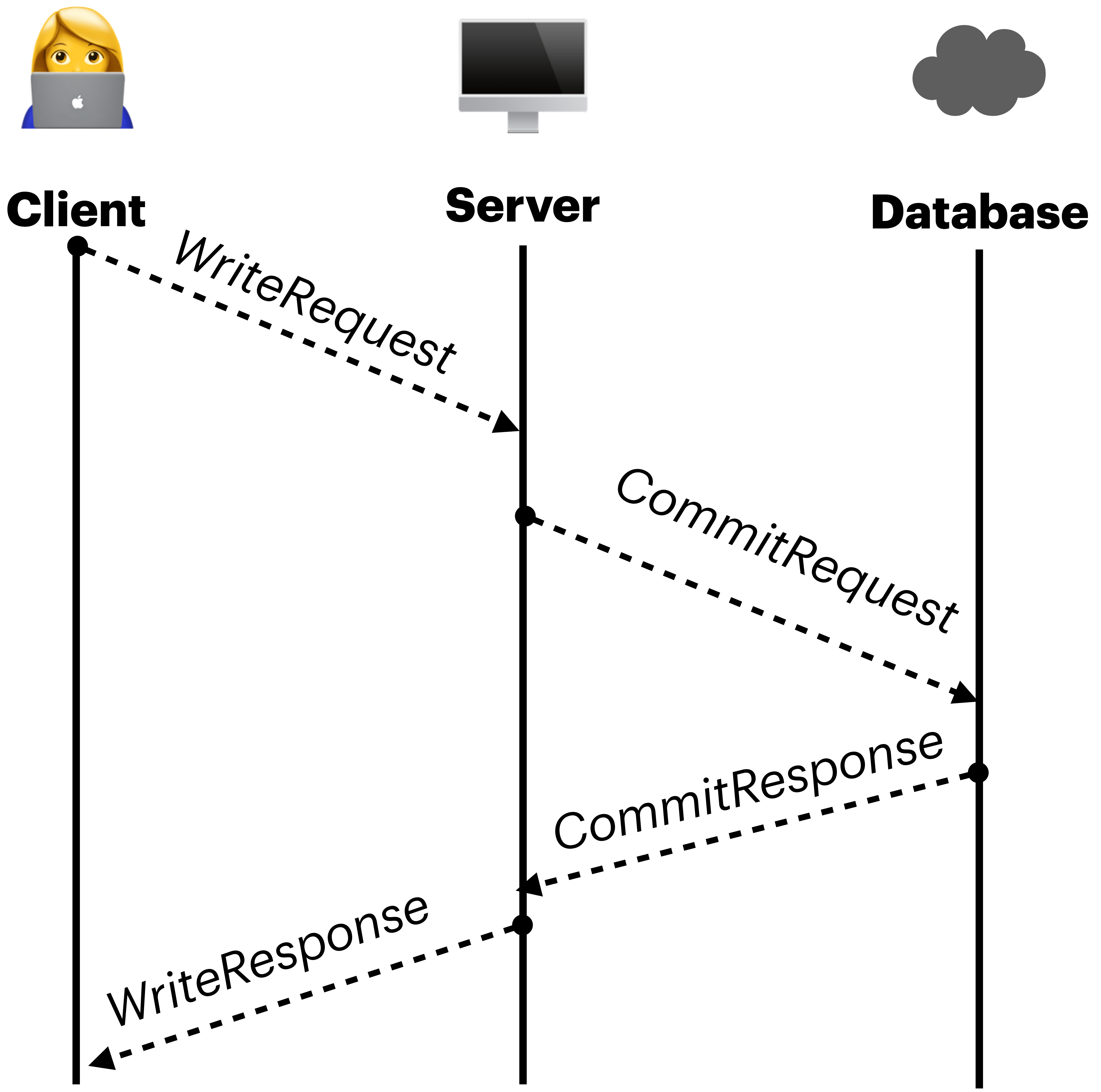
Code

`{P} S {Q}`

Distributed Systems

```
forall t: Txn.
  !(t.commit & t.abort)
```

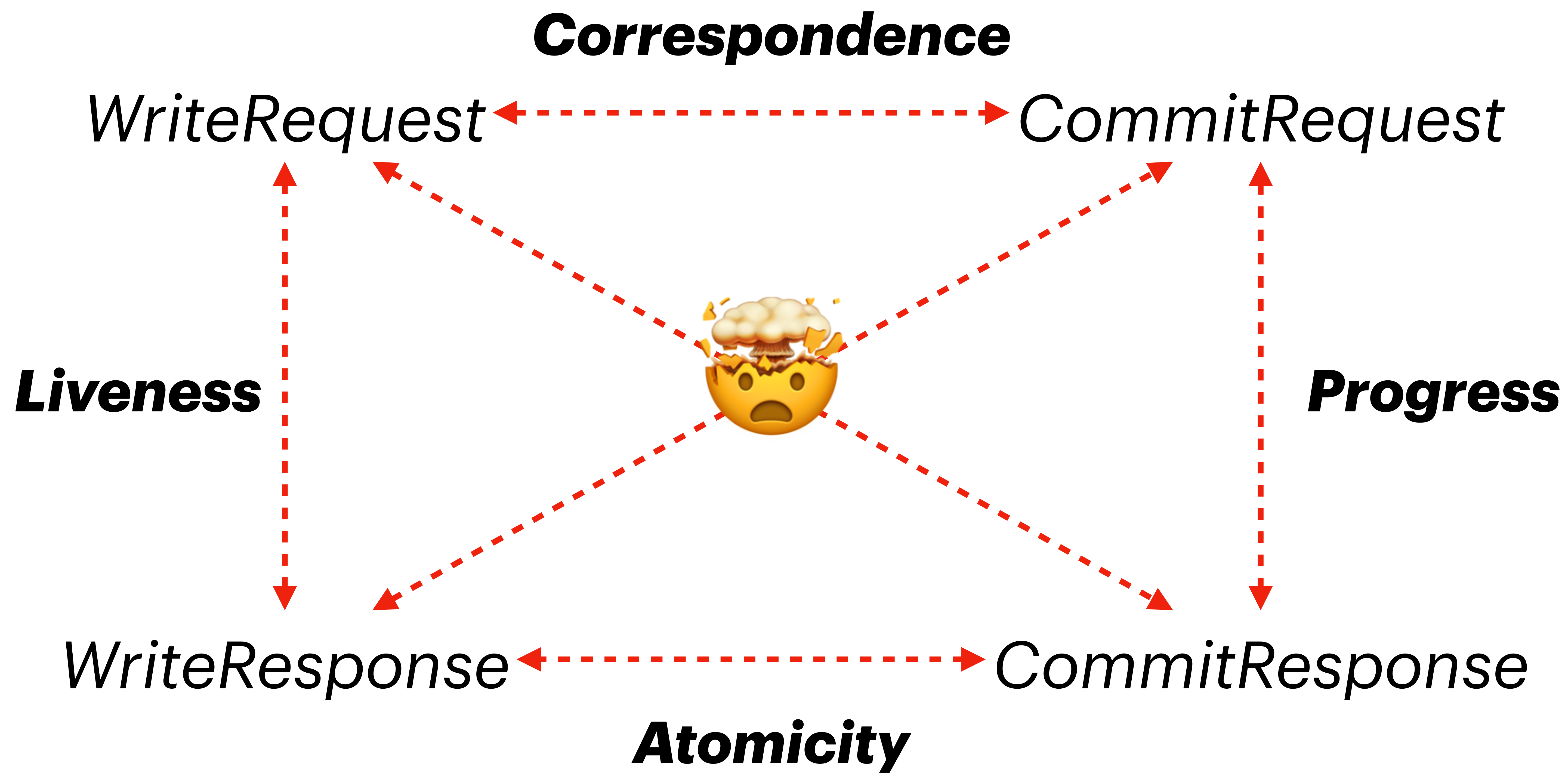
Impl ⊨ Spec



☁️ Distributed Systems

```
forall t: Txn.
  !(t.commit & t.abort)
```

Impl ⊨ Spec



Correspondence
WriteRequest
CommitRequest
Liveness
Progress
WriteResponse
CommitResponse
Atomicity

Dynamic trace-based techniques for mining specifications

Execution Traces

Likely Invariants

Hold across ALL traces

e.g.,

Dynamic trace-based techniques for mining specifications

Too many of them!

Likely Invariants

Dynamic trace-based techniques for mining specifications

Too many of them!

e.g., for the Raft [1] protocol:

Dinv [2]: ≈**1M** properties from state traces

PInfer [3]: ≈**100** properties from message logs

Likely Invariants

New challenge for developers using these tools
Identifying meaningful specifications

[1]: D. Ongaro, J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 305–319.
[2]: S. Grant, H. Cech, I. Beschastnikh, "Inferring and asserting distributed system invariants," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 1149–1159.
[3]: GitHub - p-org/P at experimental/pinfer –- github.com

Potential approaches for identifying meaningful specifications

	Meaningful Specs?	*Scalable?*
Hire an expert	✅	❌
Deductive verifiers	—	❌
Model checking	—	—
LLMs?	?	?

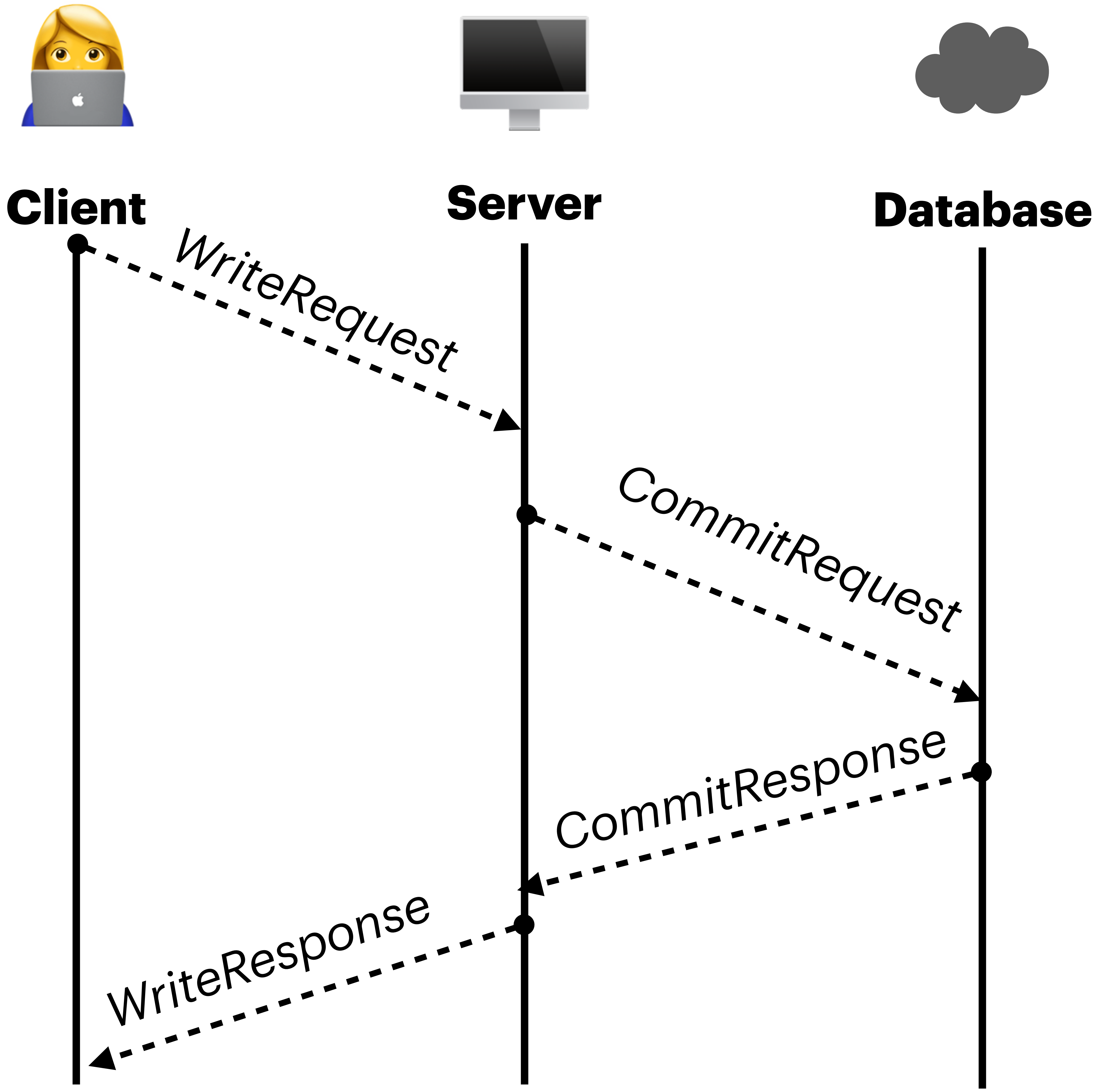
We aim to investigate these two aspects

✅: **Yes**

—: **To some degree**

❌: **No**

Background: P modeling language

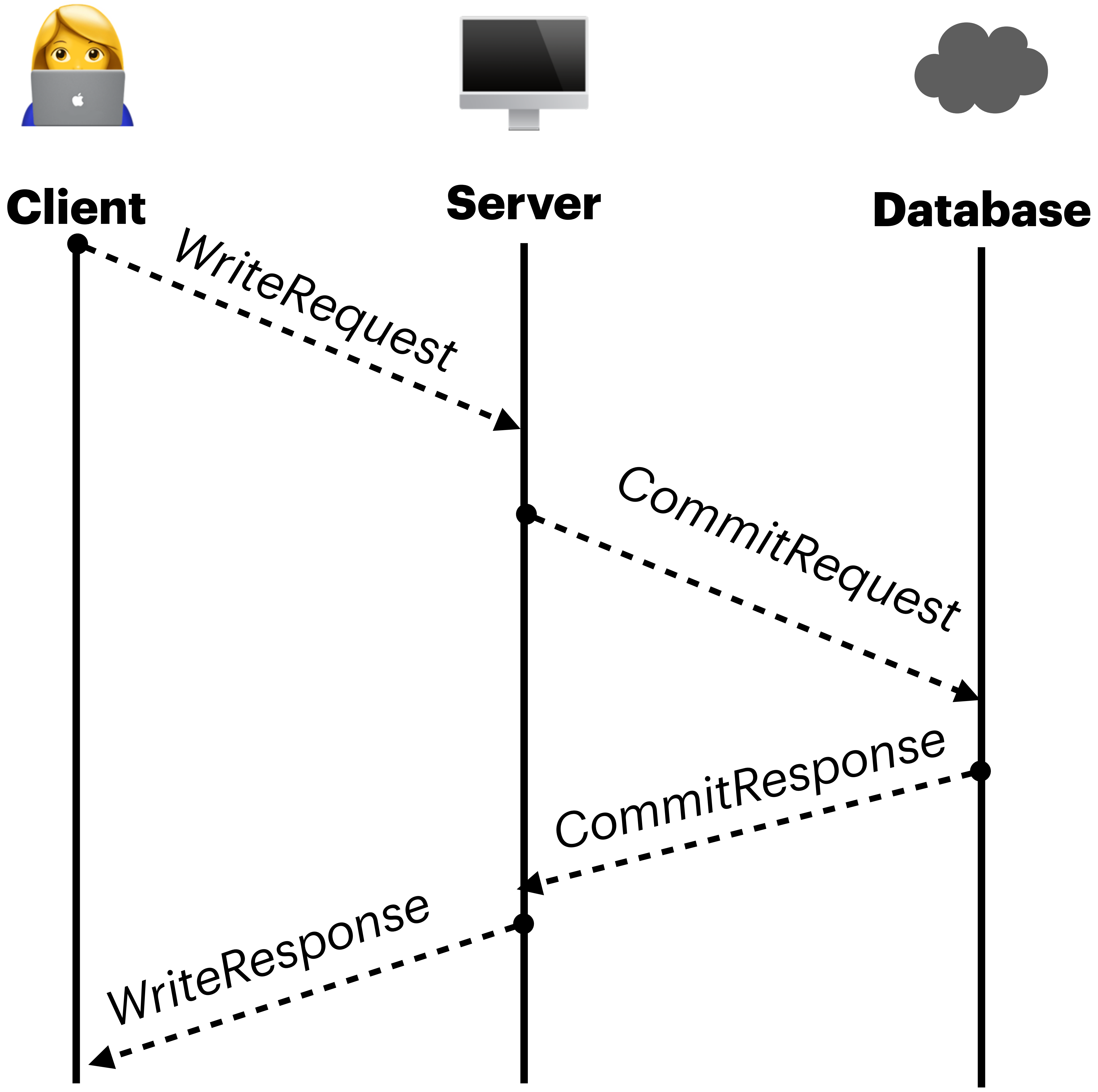


```
// Client machine
machine Client {
  var server: Server;
  start state SendRequest {
    entry {
      send server, WriteRequest, (reqId=genId(), client=this, key=someKey, value=someValue);
      goto WaitForResponse;
    }
  }
  state WaitForResponse { ... }
}

// Server machine
machine Server {
  var database: Database;
  start state Serving {
    on WriteRequest do (payload: (reqId: tId, client:Client, key:string, value:int)) {
      send database, CommitRequest, payload;
    }
    on CommitResponse do (result: (reqId: tId, status:Status, client:Client)) {
      if (result.status == COMMITTED)
        send result.client, WriteResponse, (reqId=result.reqId, success=true);
      else
        send result.client, WriteResponse, (reqId=result.reqId, success=false);
    }
  }
}

// Database machine (not shown)
```

Background: Specifications over messages

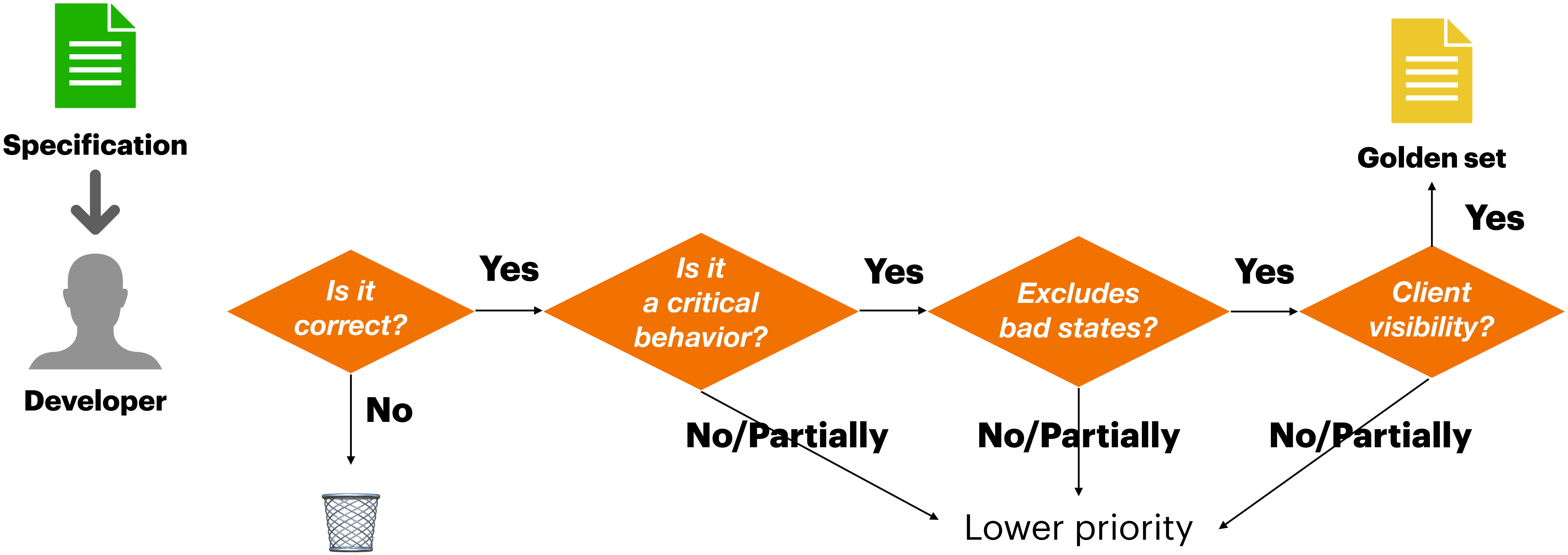


$$\forall e_0 : \mathit{WriteRequest}, e_1 : \mathit{WriteResponse} \,.$$
$$e_0 \,.\, \mathit{reqId} = e_1 \,.\, \mathit{reqId} \rightarrow e_0 \prec e_1$$

$$\forall e_0 : \mathit{WriteResponse} \,.\, \exists e_1 : \mathit{WriteRequest} \,.$$
$$e_1 \prec e_0 \wedge e_0 \,.\, \mathit{reqId} = e_1 \,.\, \mathit{reqId}$$

$\prec$ *denotes the traditional happened-before relation*

What is a meaningful specification ?



Rating Framework

Four-metric rating framework

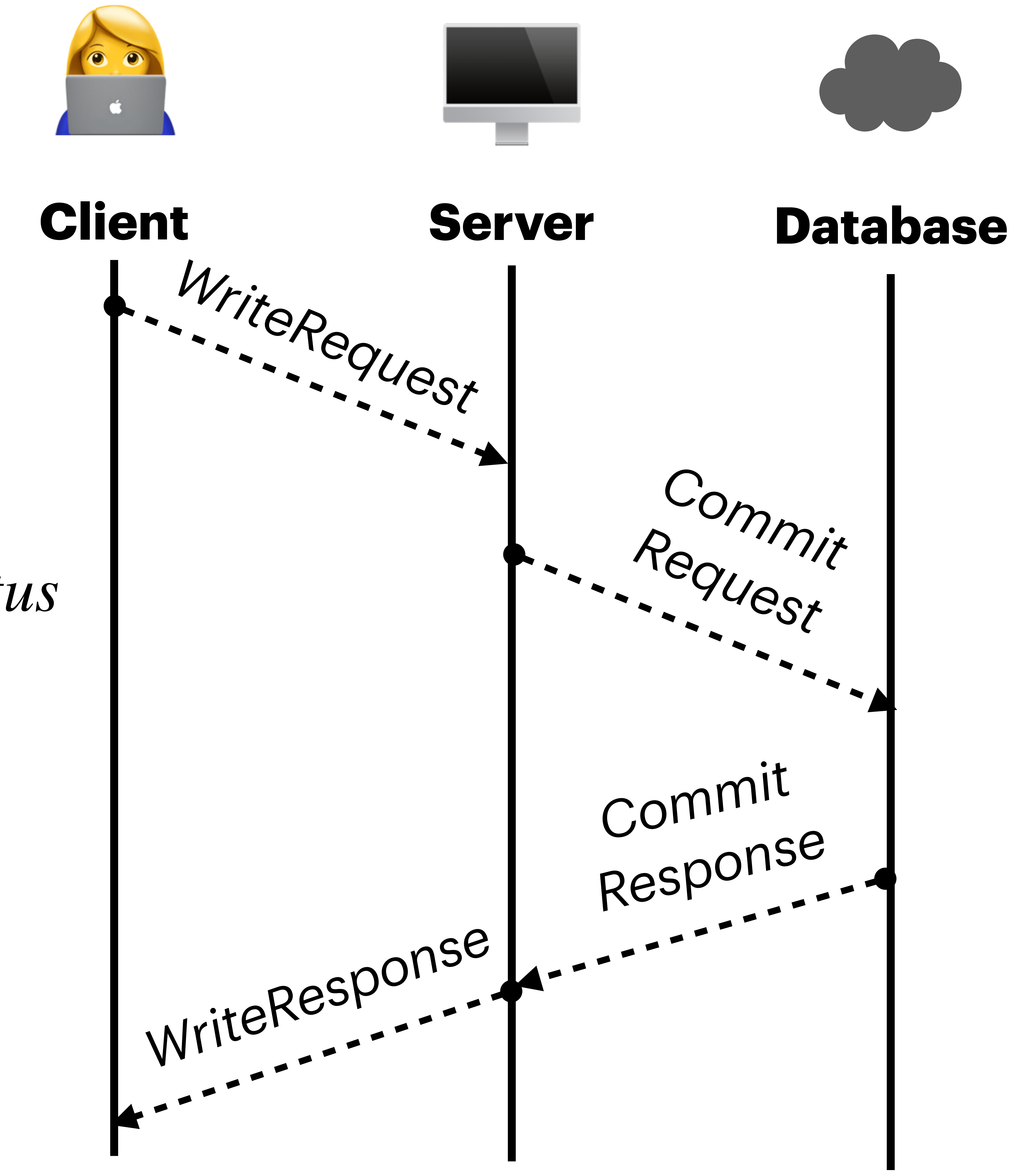
Is it correct?	Is it a critical behavior?	Excludes bad states?	Client visibility?
Generalizability How much does the specification **generalizes** to unseen (correct) behaviors	**Criticality** How **severe** is a specification violation (e.g., service outage)	**Distinguishability** *How well does a specification* ***separates*** *good and bad behaviors*	**Visibility** How directly a violation is **visible** to end users
Deprioritize specs that may cause false violations		Deprioritize specs that may allow bad behaviors	

What is a meaningful specification ?

- **Generalizability**
- **Criticality**
- **Distinguishability**
- **Visibility**

$$\forall e_0 : WriteResponse, e_1 : WriteResponse\,.\; e_0\,.\,reqId = e_1\,.\,reqId \rightarrow e_0\,.\,status = e_1\,.\,status$$

Consistency of committed writes



What is a meaningful specification ?

Generalizability

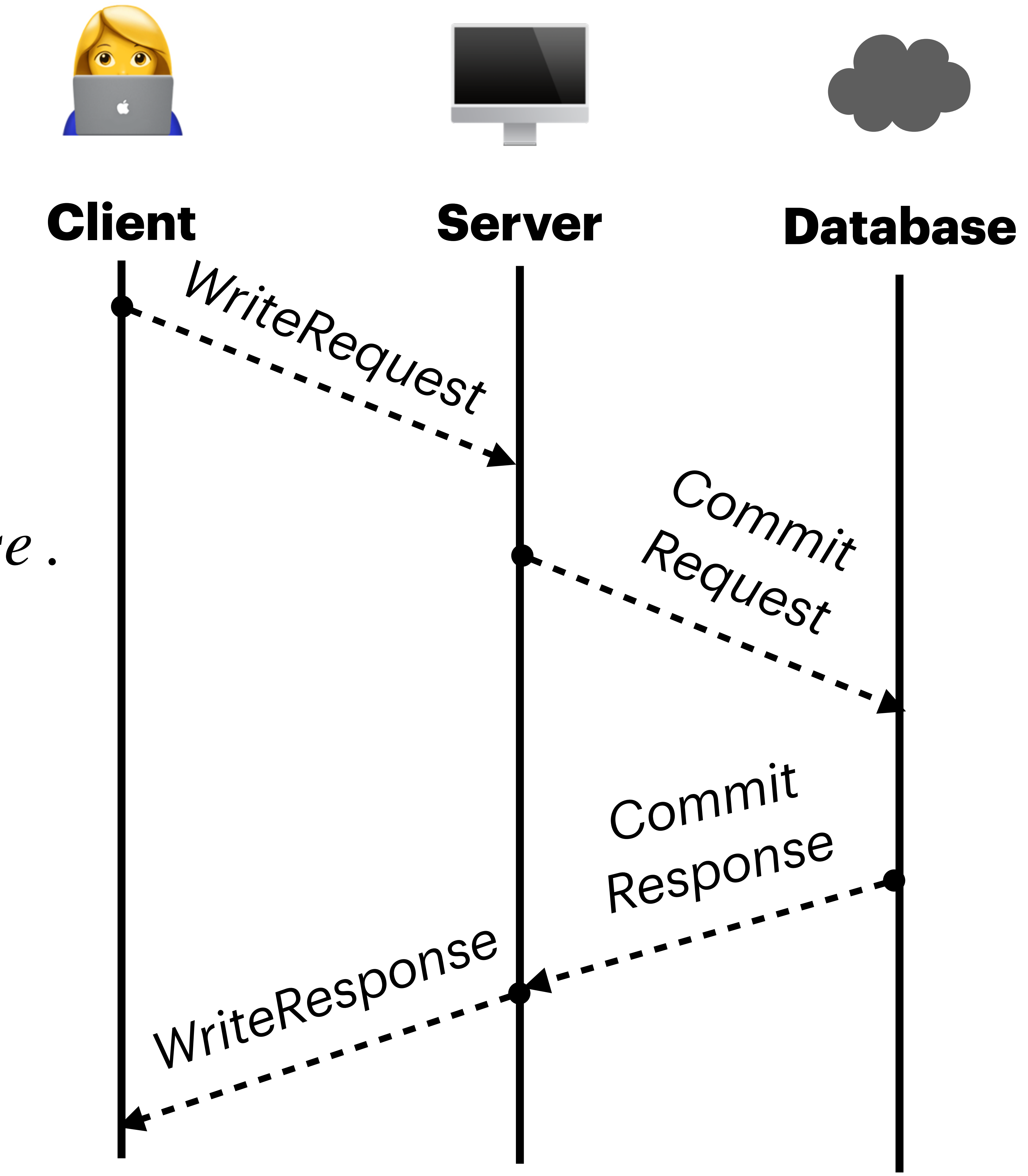
Criticality

Distinguishability

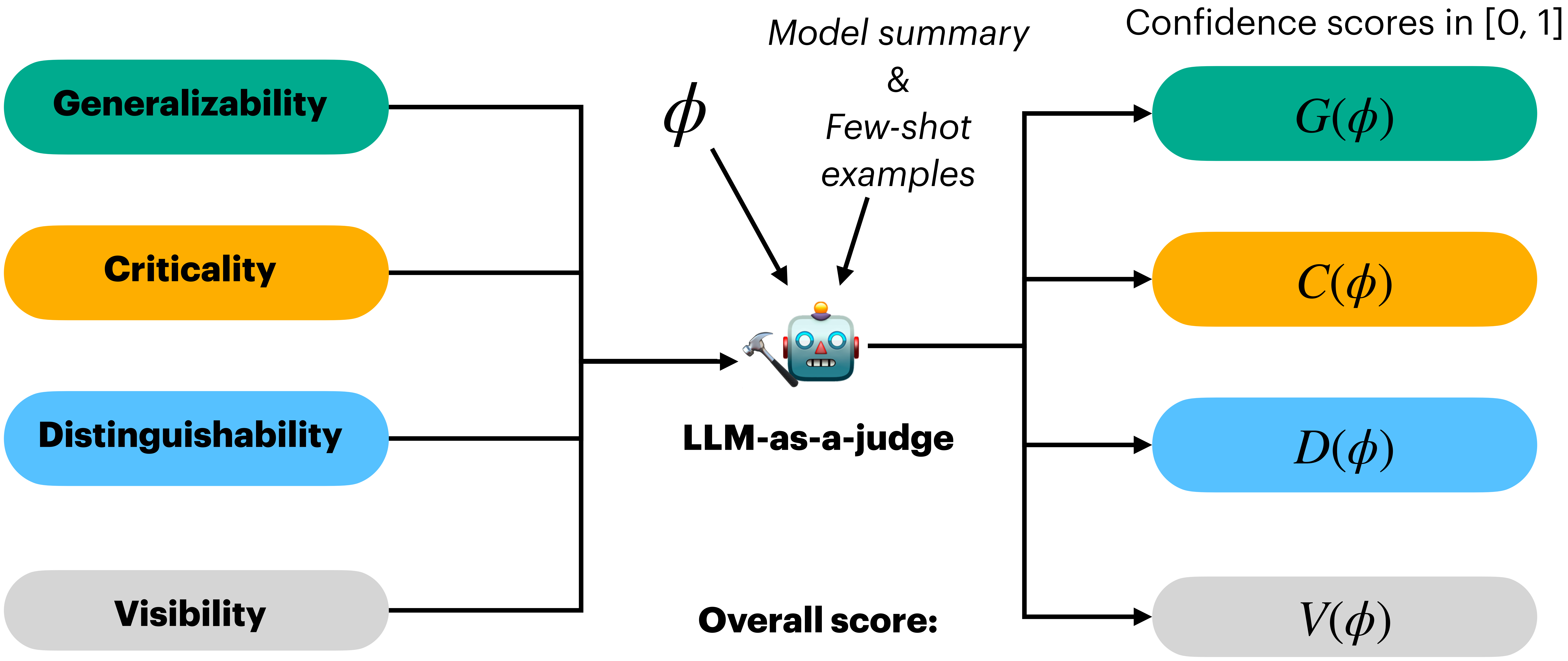
Visibility

$$\forall e_0 : CommitResponse, e_1 : CommitResponse\ .\ e_0\,.\,reqId = e_1\,.\,reqId \rightarrow e_0 = e_1$$

Uniqueness of commit response per request



What is a meaningful specification ?



$$S(\phi) = \lambda_1 \cdot \sqrt{G(\phi) \cdot C(\phi)} + \lambda_2 \cdot D(\phi) + \lambda_3 \cdot V(\phi)$$

Benchmarks

Protocols:

14 distributed protocols, including 11 from established works and 3 from proprietary services

Input specifications (for ranking):

Specifications learned by PInfer (and Model Checked against P models)

Goal specifications:

Established works: taken from the original papers (e.g., The Part-Time Parliament [1]);
Proprietary protocols: written by development teams

[1] Lamport, L. (1998). The part-time parliament. *ACM Trans. Comput. Syst., 16(2), 133–169.*

Benchmarks Setup

Evaluation Criteria:

Take the top-k ranked input specifications, and check the coverage against the goals

$$S(\phi) = \lambda_1 \cdot \sqrt{G(\phi) \cdot C(\phi)} + \lambda_2 \cdot D(\phi) + \lambda_3 \cdot V(\phi)$$

$$0.8 \qquad\qquad\qquad\qquad 0.1 \qquad\qquad 0.1$$

Preliminary Results of Ranking with LLMs

input specs — Top-k

	k=10	k=20	k=30	k=40	k=50	Cost
2PC (46)	2/2	2/2	2/2	2/2	2/2	$0.07
Chain (33)	3/5	5/5	5/5	5/5	5/5	$0.15
Raft (58)	3/5	5/5	5/5	5/5	5/5	$0.12
Toy Consensus (28)	1/1	1/1	1/1	1/1	1/1	$0.07
Distributed Lock (76)	1/1	1/1	1/1	1/1	1/1	$0.20
Whitelist (40)	1/1	1/1	1/1	1/1	1/1	$0.07
Lock Server (35)	1/1	1/1	1/1	1/1	1/1	$0.08
Paxos (46)	2/2	2/2	2/2	2/2	2/2	$0.11
Ring Leader (27)	1/1	1/1	1/1	1/1	1/1	$0.08
Sharded KV (19)	1/1	1/1	1/1	1/1	1/1	$0.06
Vertical Paxos (94)	2/2	2/2	2/2	2/2	2/2	$0.21
ClockBound (37)	2/3	3/3	3/3	3/3	3/3	$0.09
DynamoDB-LE (54)	1/5	1/5	3/5	3/5	5/5	$0.15
2PC-CC (89)	2/7	3/7	5/7	7/7	7/7	$0.26
Overall	23/37	29/37	33/37	35/37	37/37	$1.71
Coverage (%)	62.2%	78.4%	89.2%	94.6%	100.0%	-

Established protocols (2PC through Vertical Paxos)

Proprietary protocols (ClockBound, DynamoDB-LE, 2PC-CC)

Each colored cell:
#covered/#goals ***(Pass@1)***

All goals are covered in
top-10 ranked specifications

Some highlights:

1. **All** specifications are covered within **top-50**
2. Many are covered **within top-10 (60%+)**
3. Costs are **negligible**

Comparisons to Baseline Ranking Approach

Baseline:
Ranks specification formulas by Term Frequency (lowest to highest), treating predicates as “words”.

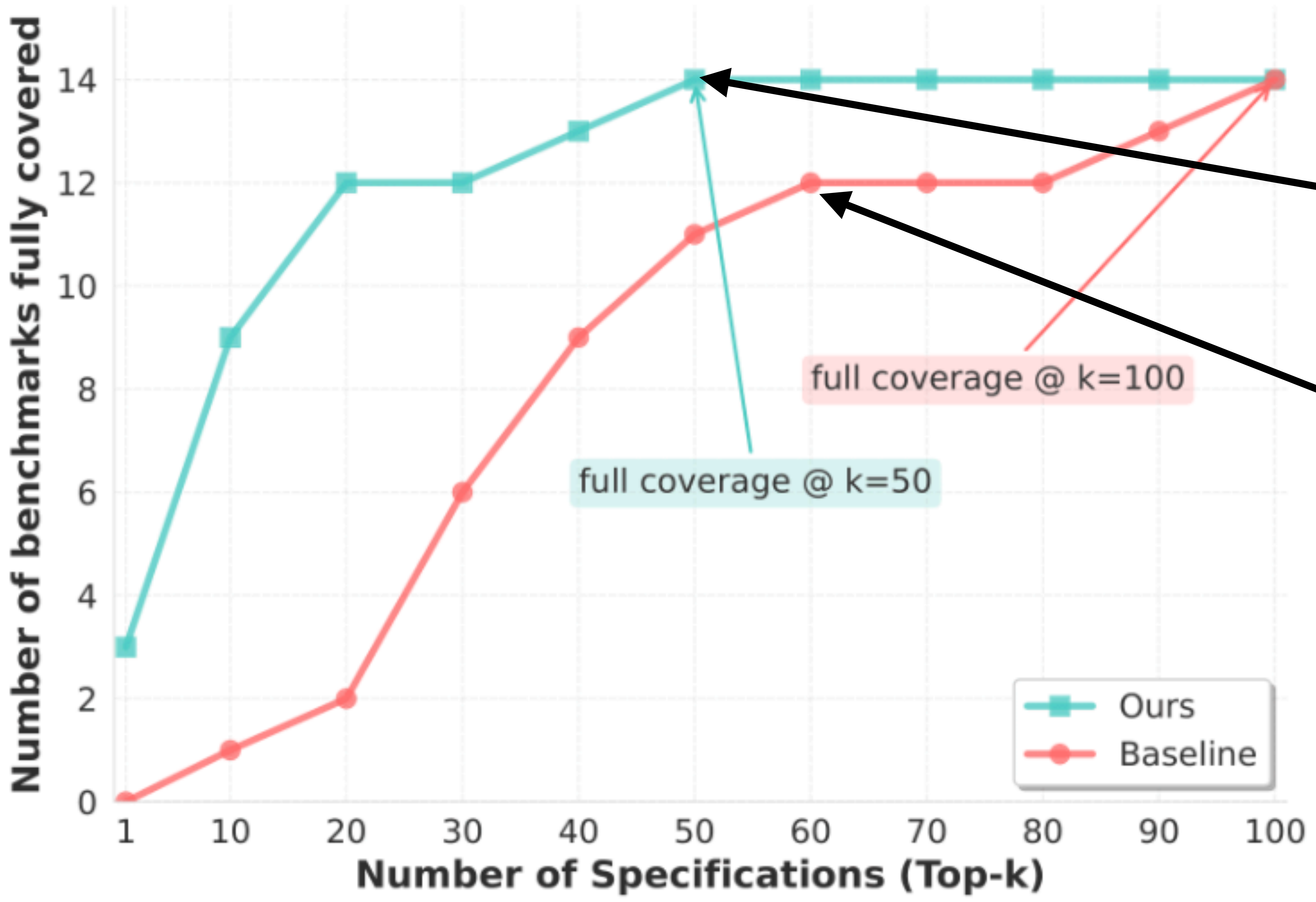


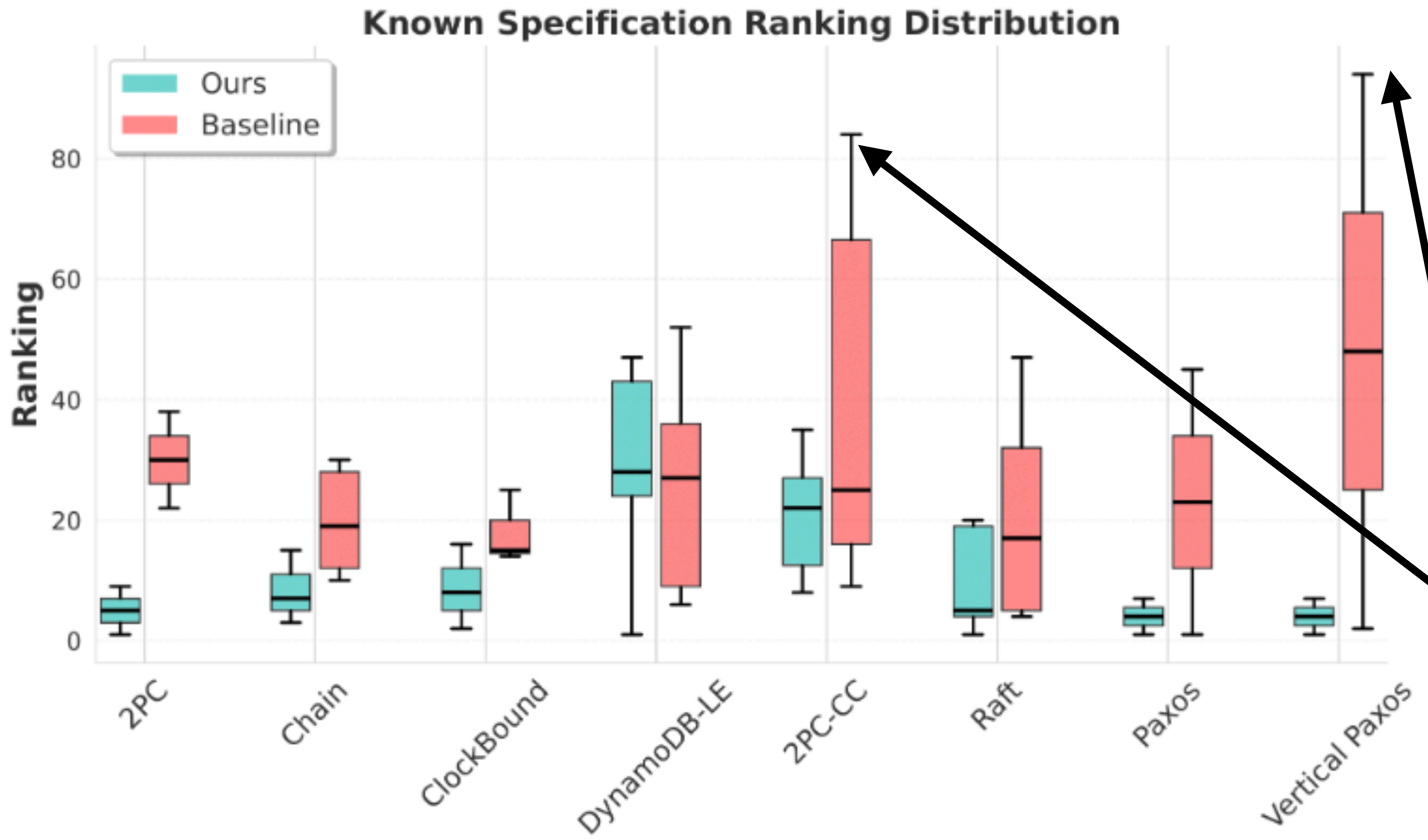
Figure 4. Number of benchmarks whose target specifications are all found within top-k using different approaches.

LLM achieves **full coverage at k=50** (as shown in the previous slide)

The baseline approach **reaches bottleneck around k=60**

Where does baseline get stuck?

Comparisons to Baseline Ranking Approach



(a) Ranking distributions of the target specifications

Ranking distribution (for benchmarks with multiple specifications)

Vertical Paxos (94)	2PC-CC (89)

Baseline struggled with Vertical Paxos and the proprietary *2PC-CC* protocols

Reason: predicate frequency can fail to capture semantics of the formula to the protocol

Summary and Proposed Future Efforts

Hopefully I convinced you that …

	Meaningful Specs?	*Scalable?*
LLMs	☑️	☑️

***Bonus**: “soundness” is guaranteed (by the mining algorithms)*

Summary and Proposed Future Efforts

How can this whole thing be important?

Helps identify the specifications from a large, noisy set

Can be useful in verified program synthesis with LLMs

Extending to program verification by identifying (mutually) inductive invariants

How can we make it better?

Refine the rating framework for domain-specific tasks

Tune Hyper-parameters for computing overall scores (to better align with ground truth)

Integrate to a verification workflow (e.g., with PVerifier[1, 2]!) after identifying important specs.

[1] https://github.com/p-org/P

[2] Mora, F., Desai, A., Polgreen, E., & Seshia, S. (2023). Message Chains for Distributed System Verification. *Proc. ACM Program. Lang., 7(OOPSLA2).*

Q & A

More about the P ecosystem: p-org.github.io/P/

Prompts in the P repo: experimental/pinfer/Src/PInfer/Scripts

Position paper: dl.acm.org/doi/10.1145/3759425.3763386

Backup Slides: Ranking with Vanilla LLM?

Model: Claude Sonnet 4

Established Protocols (Pass @ 1):

Covered within top-10
(6/11)↑

2PC, Toy Consensus,
Distributed Lock, Lock Server,
Sharded KV, Ring Leader Election

Our workflow: 9/11↑

Not covered within top-10
(5/11)↓

Chain (3/5) Raft (3/5)
Firewall (0/1) Paxos (1/2)
Vertical Paxos (1/2)

Our workflow: 2/11↓

Backup Slides: Ranking with Vanilla LLM?

Model: Claude Sonnet 4

Proprietary Protocols (Pass @ 1):

Covered within top-50
(1/3)↑

ClockBound

Our workflow: 3/3 ↑

Not covered within top-50
(2/3)↓

2PC-CC (4/7)
DynamoDB-LE (1/5)

Our workflow: 0/3 ↓

Backup Slides: Example Prompts

Prompt of the description of the Ranking Framework

<scoring_metrics>
<generalization_score>
<definition>Measures likelihood that specification holds on ALL possible executions (0.0 to 1.0)</definition>
<scoring_guidelines>
Score 1.0: Universal correctness property that applies to every possible execution
Score 0.5-0.9: Generally applicable with reasonable constraints
Score 0.0: Non-generalizable, can relate irrelevant events across different contexts

<example_high_score>
Specification: ∀e0: eAbortTrans ∀e1: eCommitTrans :: (e0.payload.transId != e1.payload.transId)
Score: 1.0
Reasoning: This is universally true - abort and commit can never occur for the same transaction in ANY execution
</example_high_score>
<example_medium_score>
….
<example_low_score>
…

Backup Slides: Example Prompts

Prompt of specification formula syntax and semantics

Here is the description of PInfer specifications:
<pinfer_specifications>
Specifications discovered by PInfer are first-order logic formulas over P events. The formulas are in the form of $\forall+\exists*$ (i.e., at least 1 forall-quantified variable and 0 or more existentially quantified variables).

<synax_of_pinfer_specifications>
…
</synax_of_pinfer_specifications>
<semantics_of_specifications>
…
</semantics_of_specifications specifications>
</pinfer_specifications>

Backup Slides: Example Prompts

Prompt of the Task

<task_overview>
PInfer learns specifications from event traces. Your job is to:
1. Analyze each specification's properties
2. Score it across four metrics: Generalization, Criticality, Distinguishability, and Visibility
3. Rank specifications from highest to lowest score
4. Output the top-k specifications as requested
</task_overview>

<common_mistakes_to_avoid>
- Don't assign high generalization scores to specifications without meaningful payload relationships
- Don't overvalue trivial temporal orderings that are implementation artifacts
- Don't ignore the real-world impact of violations on end users
- Don't score based on specification complexity rather than importance
- Don't assume all specifications with similar syntax have similar importance
</common_mistakes_to_avoid>

Thank you!

More about P and PVerifier: p-org.github.io/P/advanced/psemantics/

Prompts in the P repo: experimental/pinfer/Src/PInfer/Scripts

Position paper: dl.acm.org/doi/10.1145/3759425.3763386