# *Til Depth Do Us Part: Determination and Commitments*

JOE HELLERSTEIN

vp/distinguished scientist, AWS

prof. of the graduate school, UC Berkeley

aws

Berkeley
UNIVERSITY OF CALIFORNIA

Tandem TR 81.3

**The Transaction Concept:**
**Virtues and Limitations**

Jim Gray
Tandem Computers Incorporated
19333 Vallco Parkway, Cupertino CA, 95014

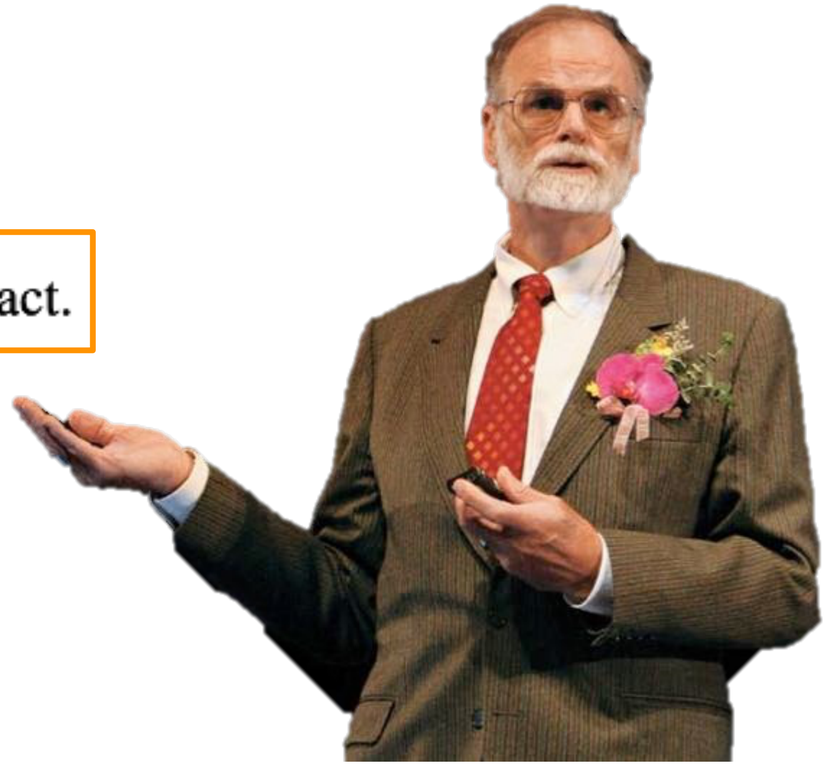
## INTRODUCTION: What is a transaction?

The transaction concept derives from contract law.

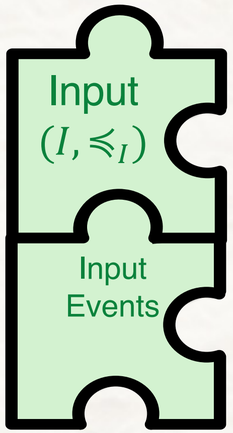
t, two or more parties
the joint signature of a
document or by some other act (as simple as a handshake or nod). If the parties are rather suspicious of one another or just want to be safe, they appoint an intermediary (usually called an escrow officer to coordinate the commitment of the transaction.

The Christian wedding ceremony gives a good example of such a contract.

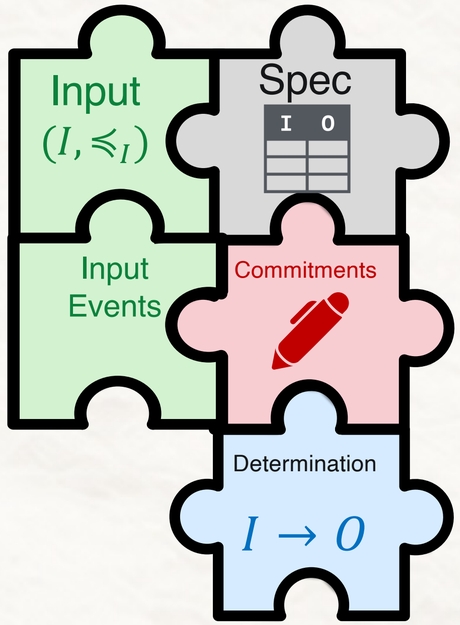
minister first asks if anyone has any objections to the marriage; he then asks the bride and groom if they agree to the marriage. If they both say, "I do", he pronounces them man and wife.



Appeared in Proceedings of Seventh International Conference on Very Large Databases, Sept. 1981. Published by Tandem Computers Incorporated.


Input
$(I, \preccurlyeq_I)$
Input
Events


Do you, Alice, take Bob to be your lawfully wedded spouse?




Only one option remains:

$I \to O$. *Determined.*

Two possible *commitments:*

each rules out some futures.

*Ambiguous* spec!

Multiple outputs per input.

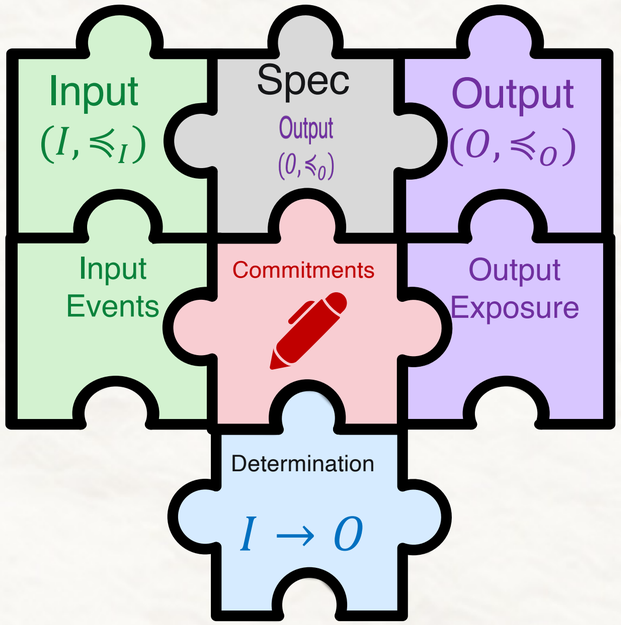
**Alice's admissible futures: a Spec**

| INPUT | OUTPUT |
|---|---|
| ~~"Do you …?"~~ | ~~Bob~~ |
| ~~"Do you …?"~~ | ~~Carol~~ |
| ~~"Do you …?"~~ | ~~Dan~~ |

**Bob**

**¬Bob**

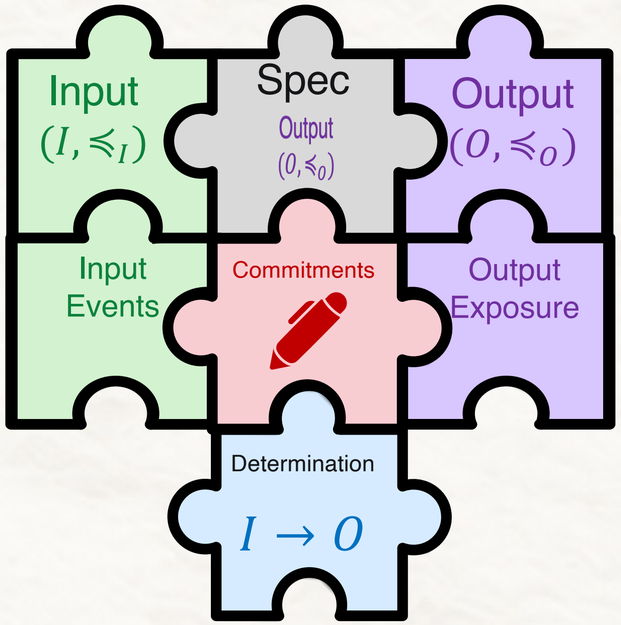




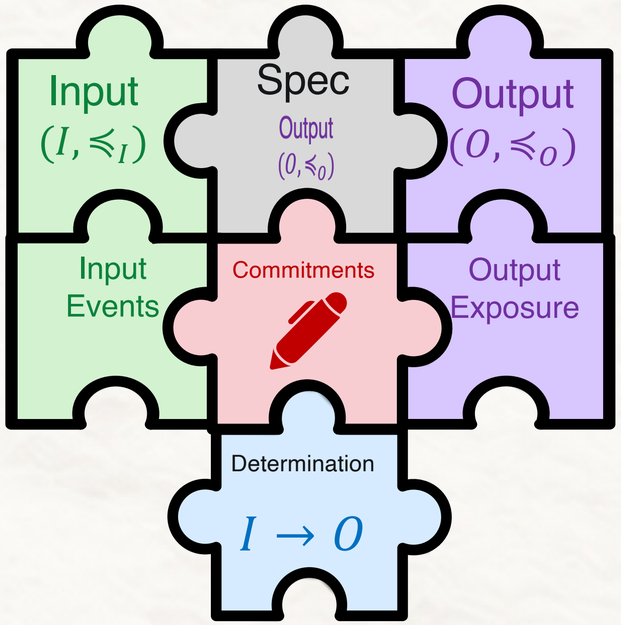

Output *exposure*

**ALICE: I DO**




Input
$(I, \preccurlyeq_I)$
Spec
Output
$(O, \preccurlyeq_O)$
Output
$(O, \preccurlyeq_O)$
Input
Events
Commitments
Output
Exposure
Determination
$I \to O$

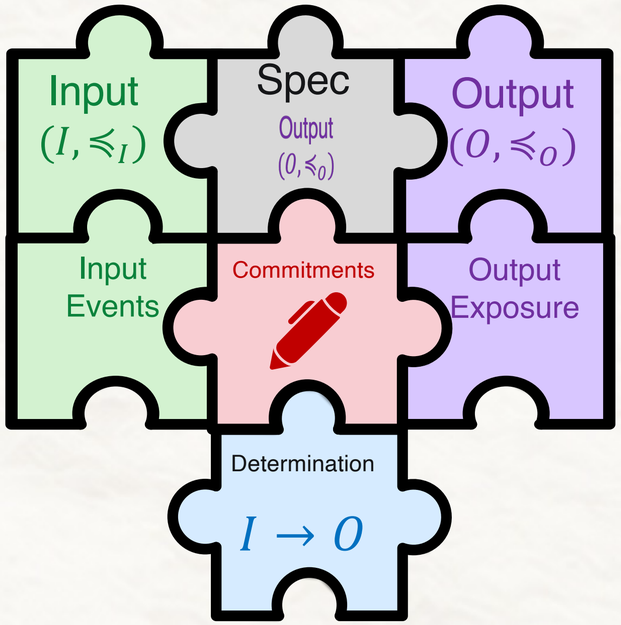

*crickets*
BOB: I DO




*Computation* to decide.

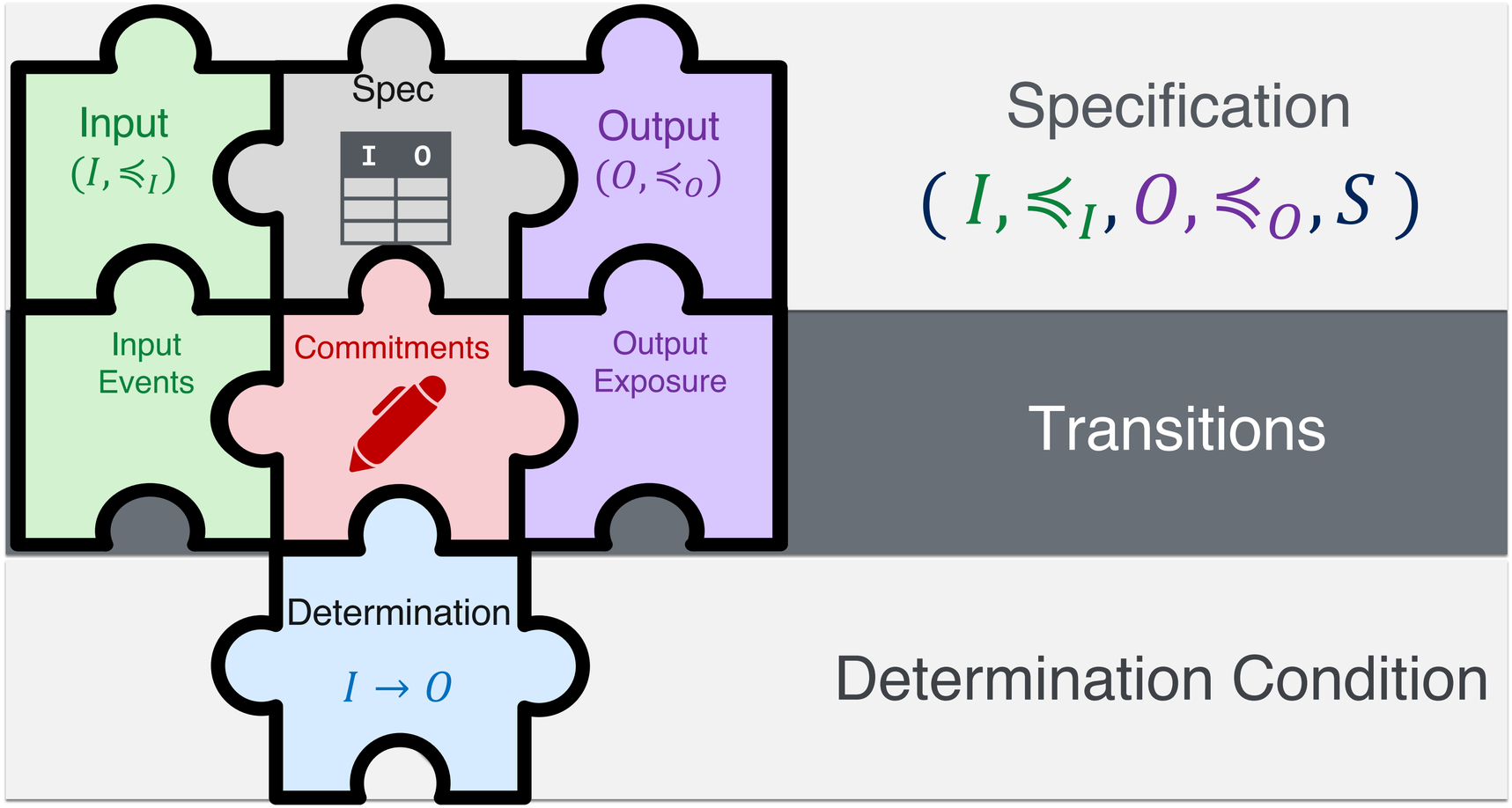
```
pct == 100 → MARRIED;
```




Input
$(I, \preccurlyeq_I)$
Spec
Output
$(O, \preccurlyeq_O)$
Output
$(O, \preccurlyeq_O)$
Input
Events
Commitments
Output
Exposure
Determination
$I \to O$


I now pronounce you married

# A Formal Framework



# Determination: Why Should I Care?

Ambiguity/Non-Determinism is *inherent* in concurrent/distributed systems

- *Execution* is non-deterministic (ordering, timeout, failure)
- *Semantics* are often *ambiguous by design* (serializability, leader election, …)

Committing to one legal output among many is a primary cost!

- Rounds of Consensus, Lock Acquisitions, Queue Insertion, …




But when precisely do we need these mechanism? Why? How often?

We need a principled theory … and practical tools!

# Formal Specs

# Semantic Specs: $I \rightarrow O$ mappings

A semantic spec is a *relation* in $I \times O$

*Determined:* satisfies FD $I \rightarrow O$

| INPUT | OUTPUT |
|---|---|
| "Do you …?" | 👨🏽 **Bob** |
| ~~"Do you …?"~~ | ~~👩🏾 **Carol**~~ |
| ~~"Do you …?"~~ | ~~👨🏿 **Dan**~~ |
| … | **…** |

$\not\preceq_O$

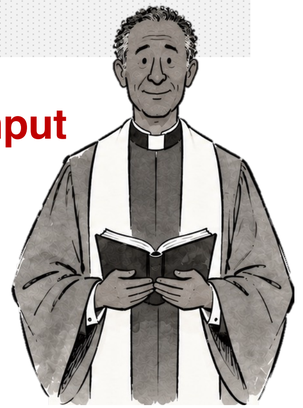


Semantic Commitments: **Choosing a Legal Output**

For ordered inputs, even determined specs may have to commit a *seal*.

| INPUT | OUTPUT |
|---|---|
| {Alice: Y} | **Timeout…** |
| ~~{Alice: Y, Bob: Y}~~ | ~~**I pronounce…**~~ |
| {Alice: N} | **Call it off…** |
| {Alice: N, Bob: Y} | **Call it off…** |
| … | **…** |

$\preceq_I$ $\not\preceq_O$



Execution Commitments: **Sealing Input**

# From Framework to Theory

- **DECISION:** Are commitments intrinsically *required* to determine outcomes?
- **COMPLEXITY:** How many commitments for a given protocol?
- **PROVENANCE:** Which commitments affected a particular outcome? What if…

**Determination Theory**

studies the structure and cost of *determining* outcomes

from non-deterministic executions

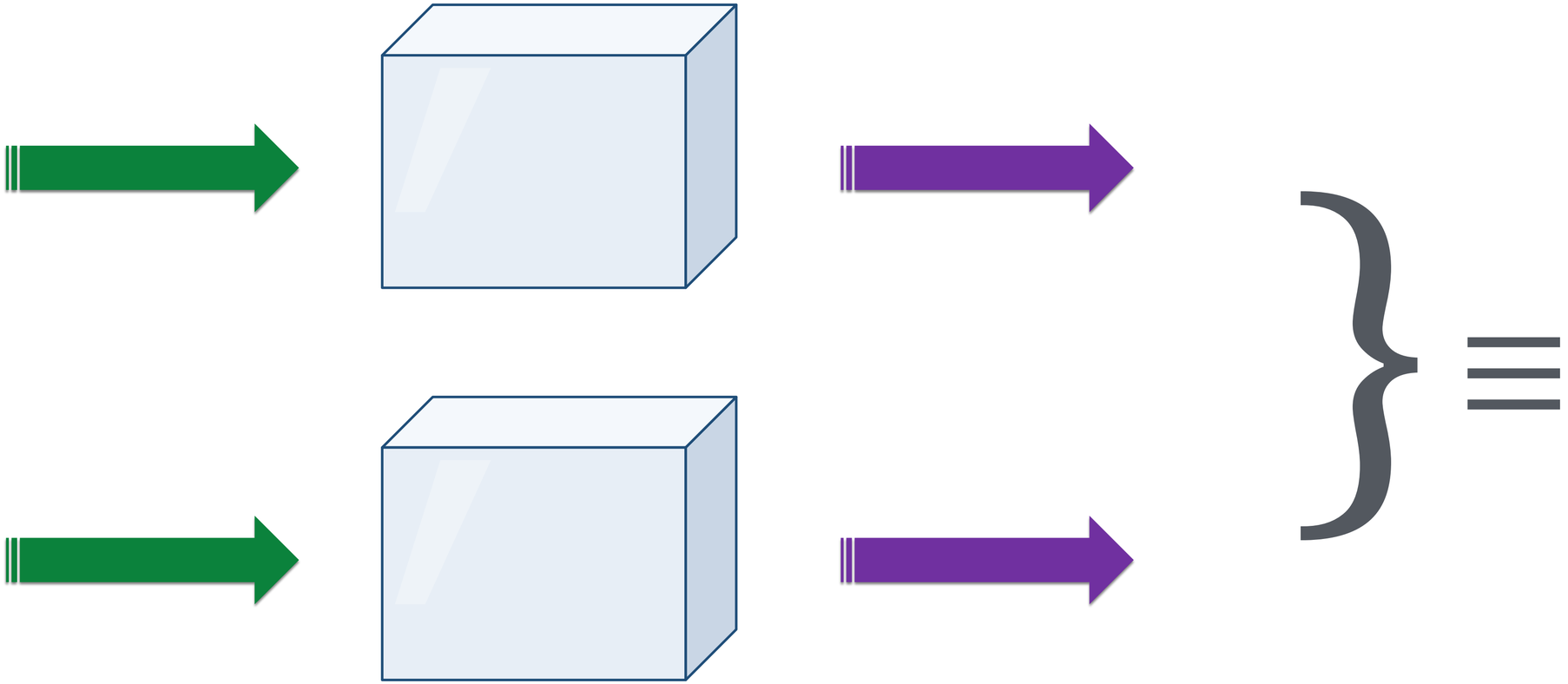
and ambiguous specifications.

# From Framework to Theory

- **DECISION:** Are commitments intrinsically *required* to determine outcomes?
- **COMPLEXITY:** How many commitments for a given protocol?
- **PROVENANCE:** Which commitments affected a particular outcome? What if…

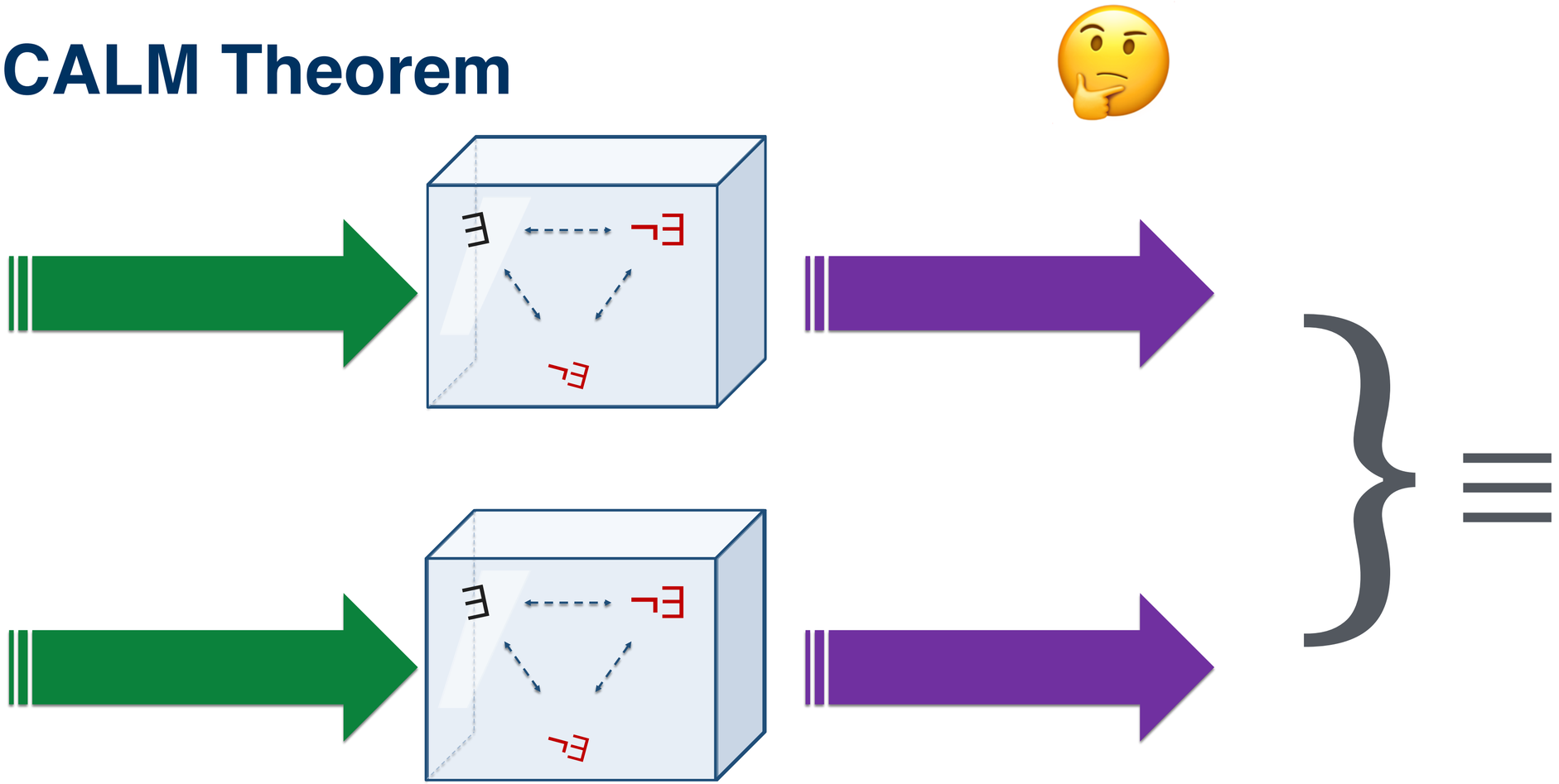
# DECISION: Is Coordination Avoidable

# CALM: Consistency As Logical Monotonicity



Coordination-Free Consistency *iff* Spec is Monotone

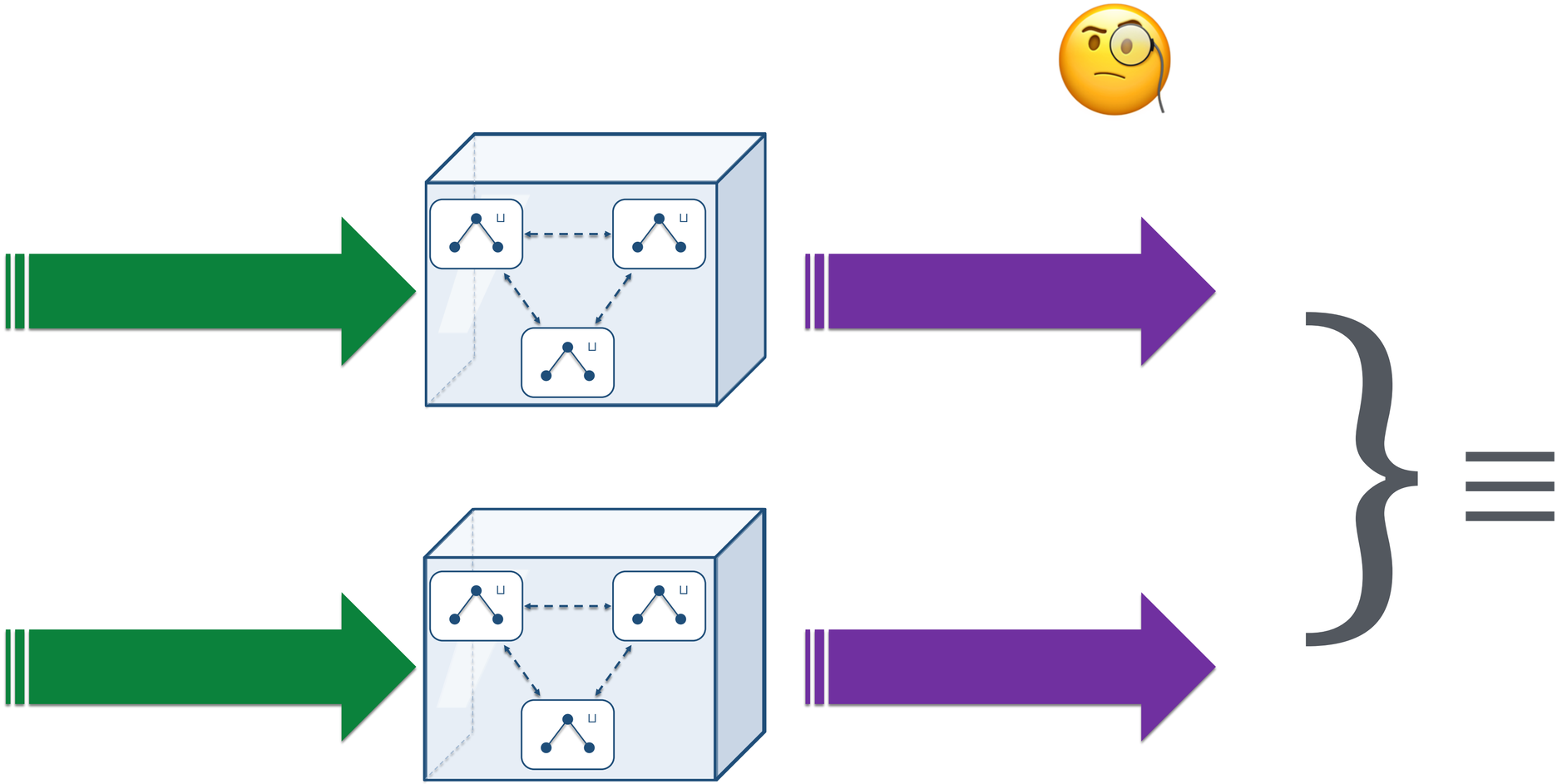
# Original CALM Theorem




Ameloot, et al 2011: a Network of Logic Programs (relational transducers)

Coordination-Free Consistency *iff* Spec is Monotone Datalog

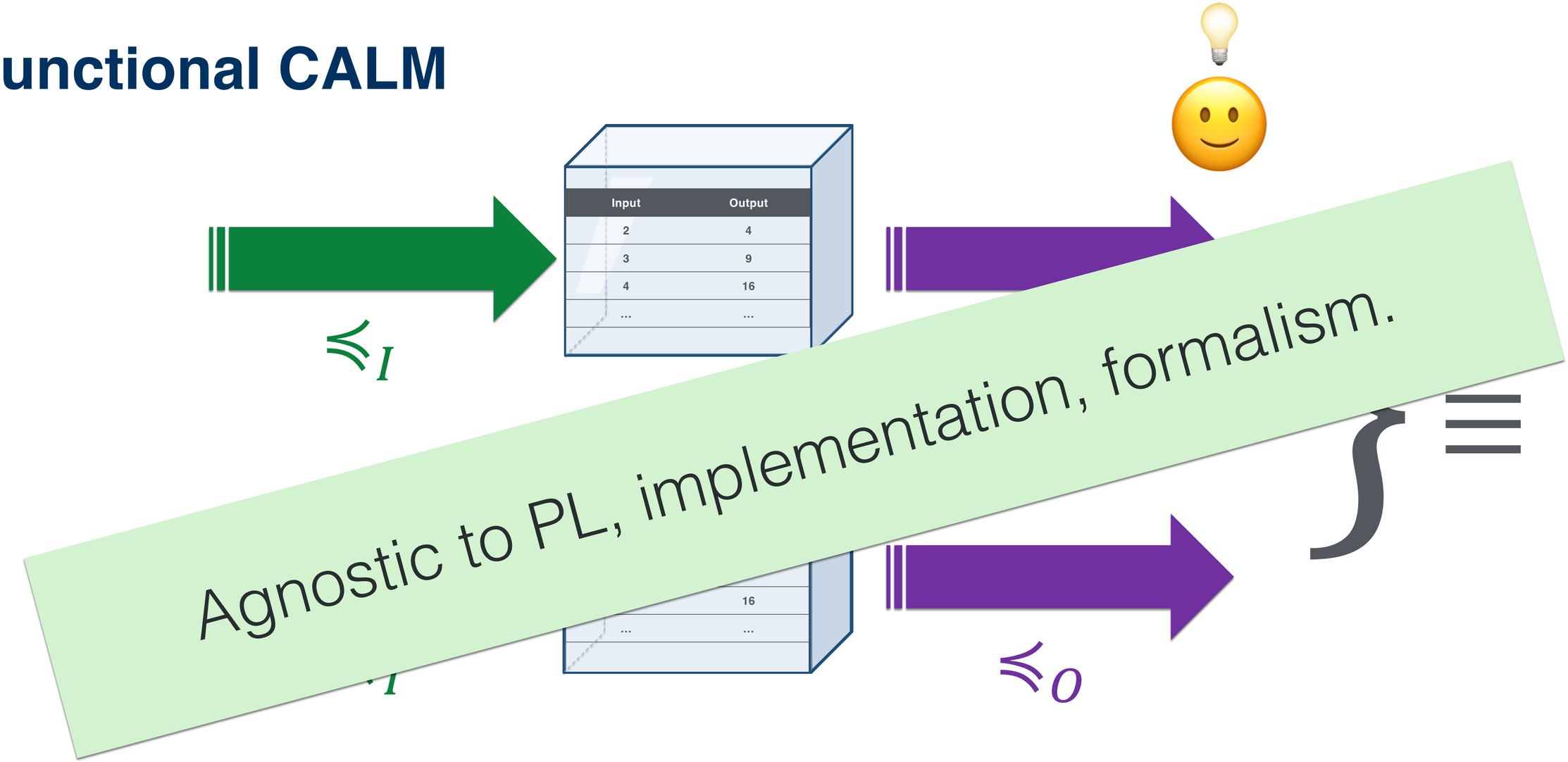
# CRDTs



Shapiro, et al 2011: Conflict-Free Replicated Data Types

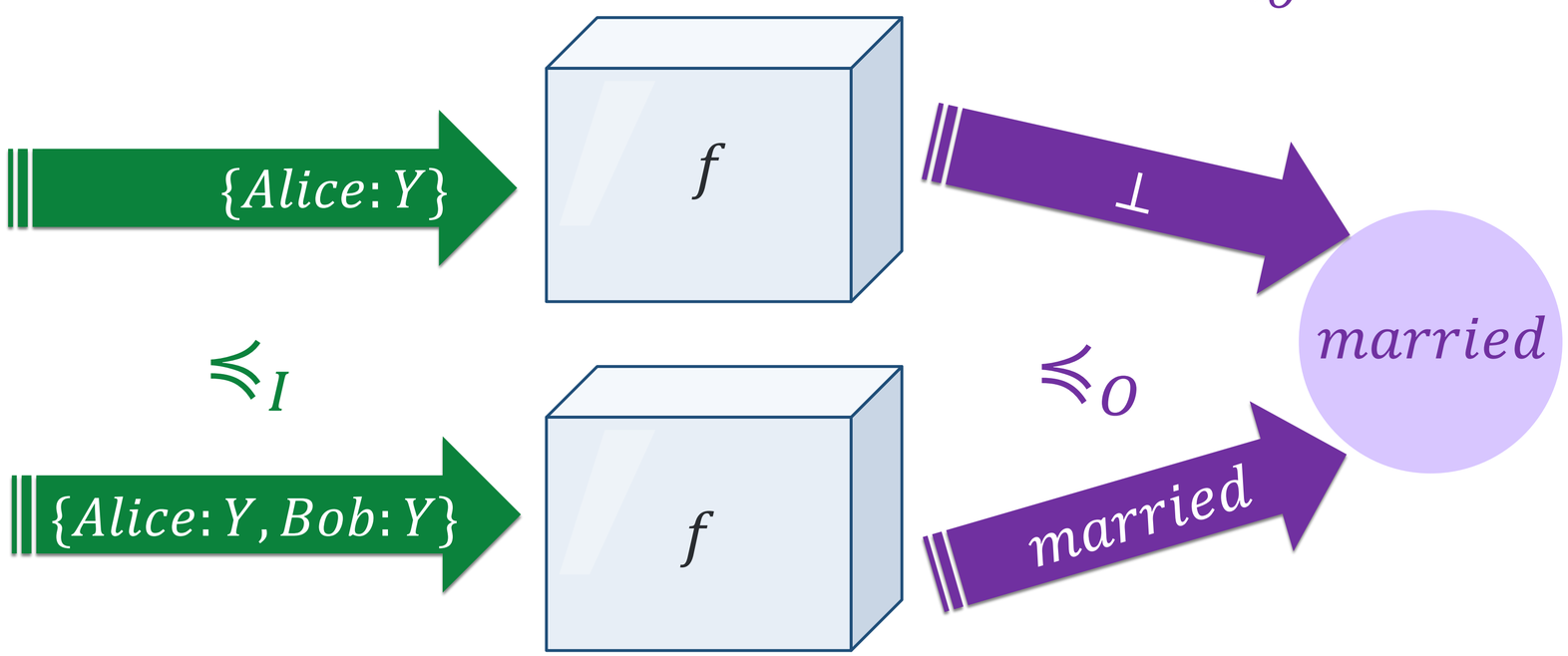
Coordination-Free Consistency *if* a Join Semilattice

# Functional CALM



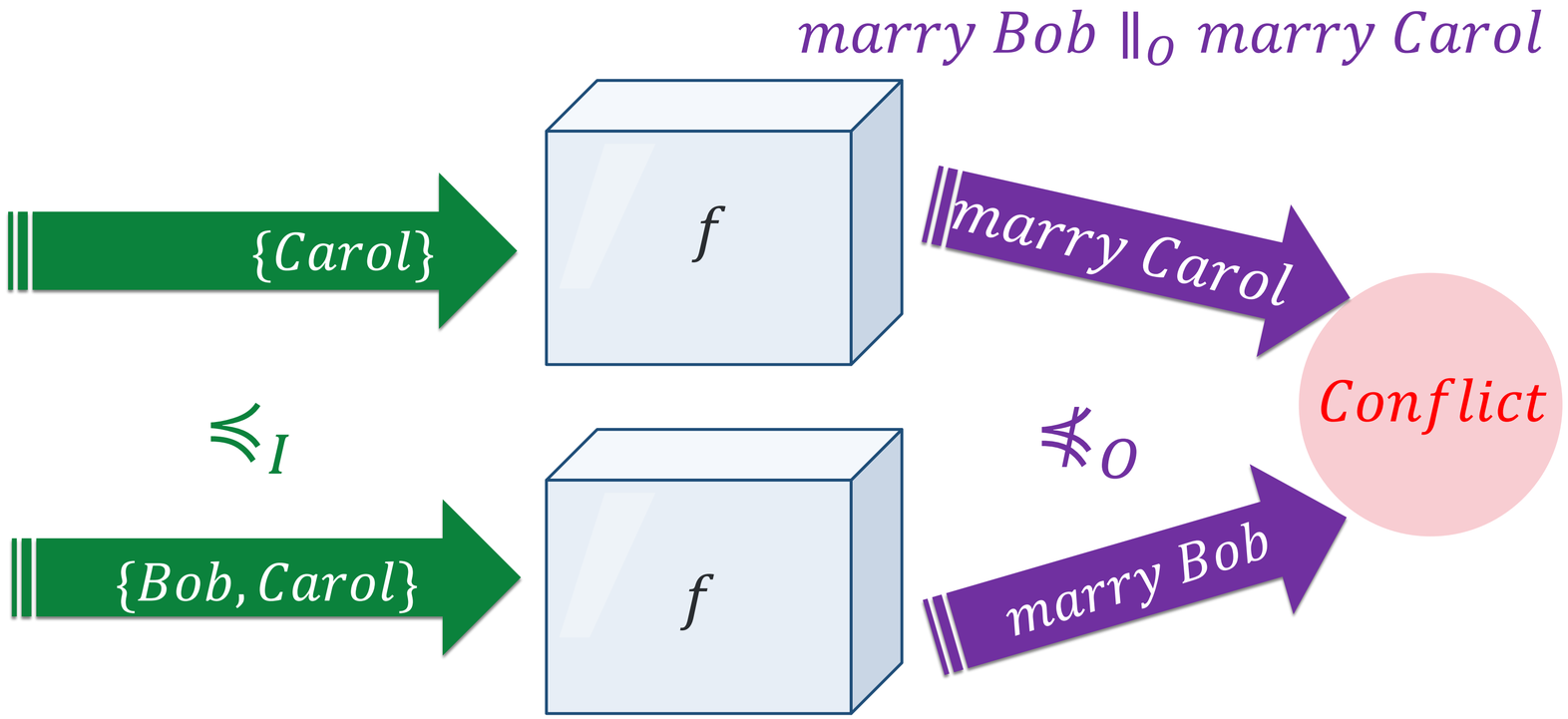

Coordination-Free Consistency *iff* Spec is *any* Monotone function

# Functional CALM




- $f$ monotone $\rightarrow f(j) \succcurlyeq_O f(i)$; nodes can expose output $f(\cdot)$ at will
  - Common refinement: $f(j) \succcurlyeq_O f(i)$.

# Functional CALM




- $f$ monotone $\rightarrow f(j) \succcurlyeq_O f(i)$; nodes can expose output $f(\cdot)$ at will
  - Common refinement: $f(j) \succcurlyeq_O f(i)$.
- $f$ not monotone $\rightarrow$ output exposure of $f(\cdot)$ requires commitment
  - The two nodes conflict: $f(j) \not\succcurlyeq_O f(i)$ ! No common refinement.
  - Must agree to commit at a common input ($Bob$ or $Carol$).

# Complete CALM: Handling Ambiguous Specs

Two conditions:

- **Sealing Input:** ***Lifted monotonicity (Hoare Preorder)***
  - as input grows, there always exists some bigger output
  - $i \preccurlyeq_I j, o \in S(i) \rightarrow \exists o' \in S(j)\ s.t.\ o' \succcurlyeq_O o$
- **Choosing Outputs:** ***Directedness (order theory)***
  - at each $i$, every pair of outputs has a common upper bound
  - $\forall i \in I\ o_1, o_2 \in S(i) \rightarrow \exists o_3 \in S(i) \quad s.t. o_3 \succcurlyeq_O \{o_1, o_2\}$

Ambiguity: a *relational* spec in $I \times O$

| INPUT | | OUTPUT |
|---|---|---|
| "Do you …?" | $\not\preccurlyeq_O$ | 👨🏽 **Bob** |
| "Do you …?" | | 👩🏽 **Carol** |
| "Do you …?" | | 👨🏾 **Dan** |
| … | | **…** |

# Complete CAP

- **A corollary**: CAP holds *iff* spec is monotone
- **Generalization of** Gilbert-Lynch beyond Impossibility:
  - Both True and False cases of CAP, depending on the choice of spec!

- Quibbles reduce to the nature of the spec design
  - *No more quibbling about the “truth” of a theorem!*

# FROM THEORY TO PRACTICE

## COORDINATION CHECKS IN YOUR COMPILER

# Hydro: Distributed Programming in Rust

**COMPILE-TIME** PREVENTION OF [illegible]EMS



Process Location Type

Cluster Location Type (SPMD)

Typed transport protocols

src/two_pc.rs

```
pub fn count_yes<'a>(
    votes: Stream<Vote, Cluster<'a, Participant>> ,
    leader: &Process<'a, Leader>
) -> Singleton<usize, Process<'a, Leader>> {
    votes
        .send(leader, TCP.retry_on_fail().bincode())
        .fold(q!(|| 0), q!(|n, v| if v == Vote::yes {*n += 1}))
}
```

Pure Hydro *guarantees*

**eventual determinism**

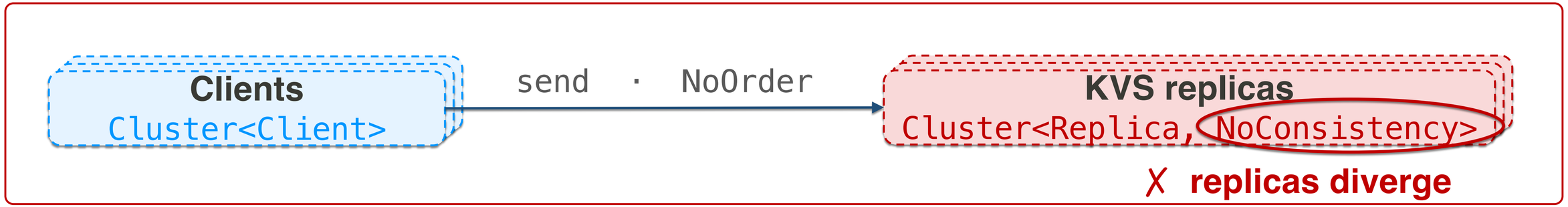
at compile time

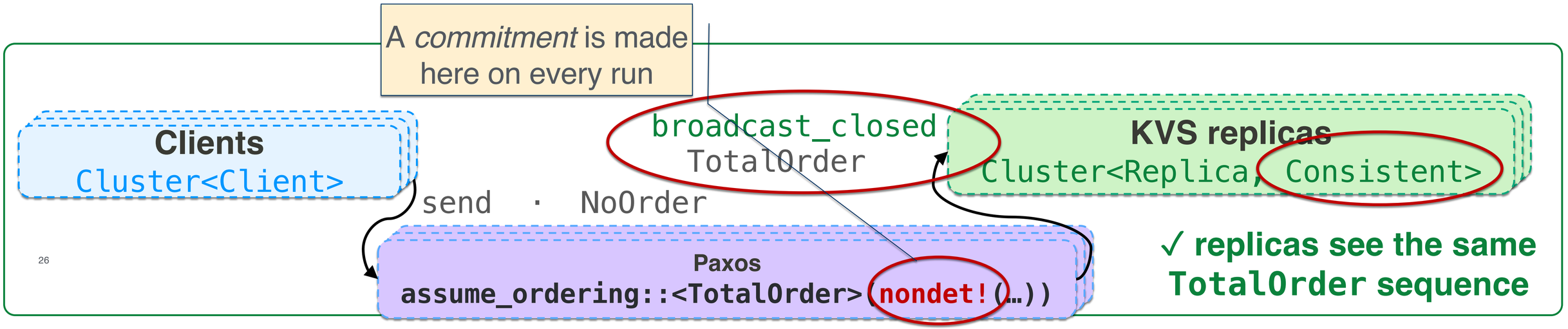
One function, many machines

```
error: `fold` requires `ExactlyOnce` delivery, but this stream is `AtLeastOnce`
= help: prove the closure is idempotent with `idempotent = manual_proof!(…)` or
acknowledge the non-determinism with `assume_retries(nondet!(…))`
```

# Beyond CALM: Proper Coordination

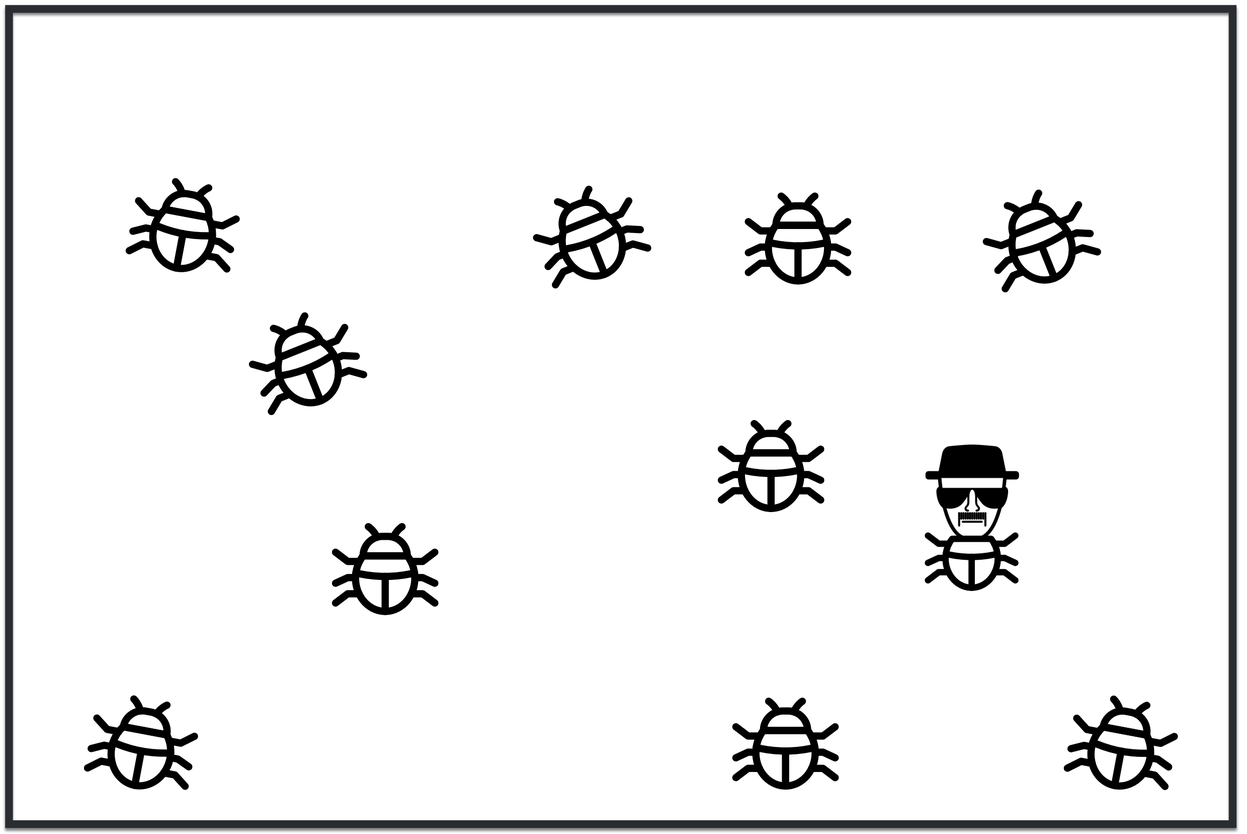
- Hydro analysis for **proper coordination** in the type system



# Automated Reasoning in a Nutshell

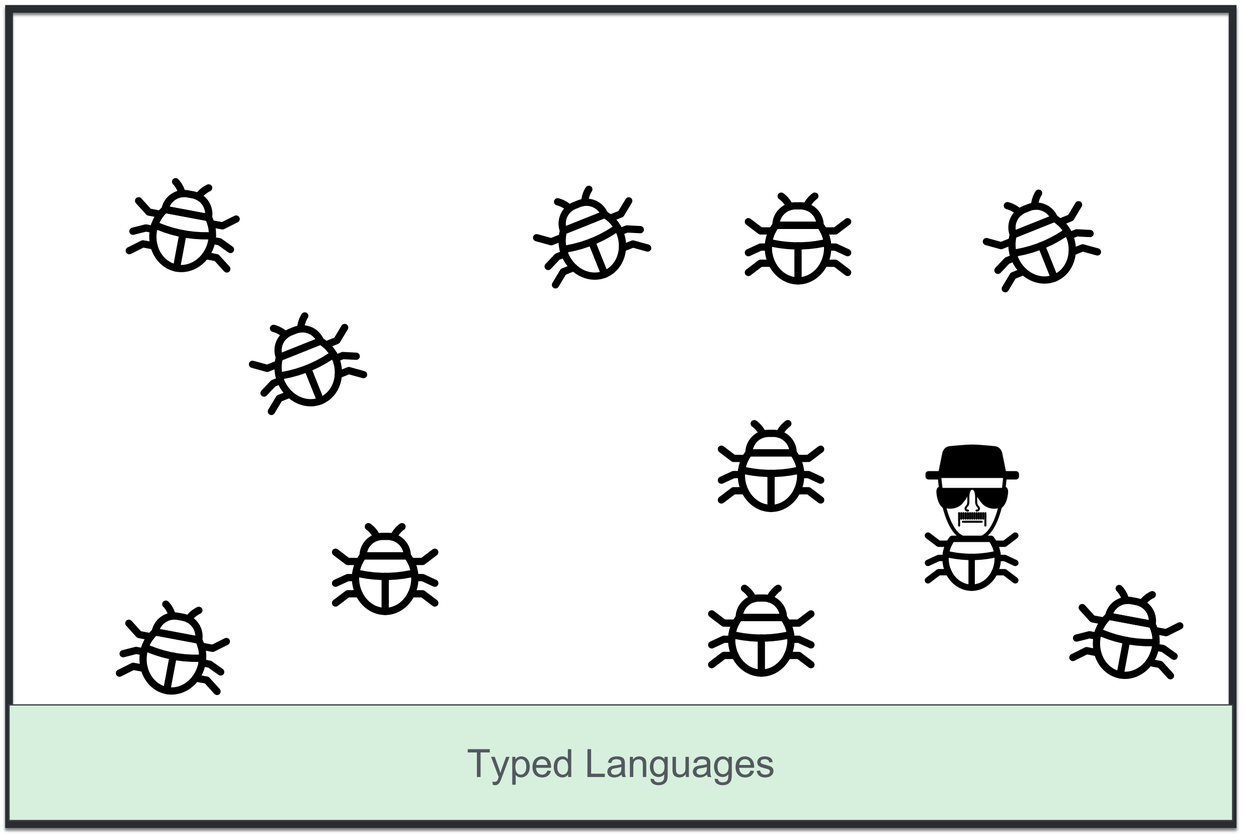
Ceiling: Proof of Correctness



Floor: Assembly code

# Automated Reasoning in a Nutshell

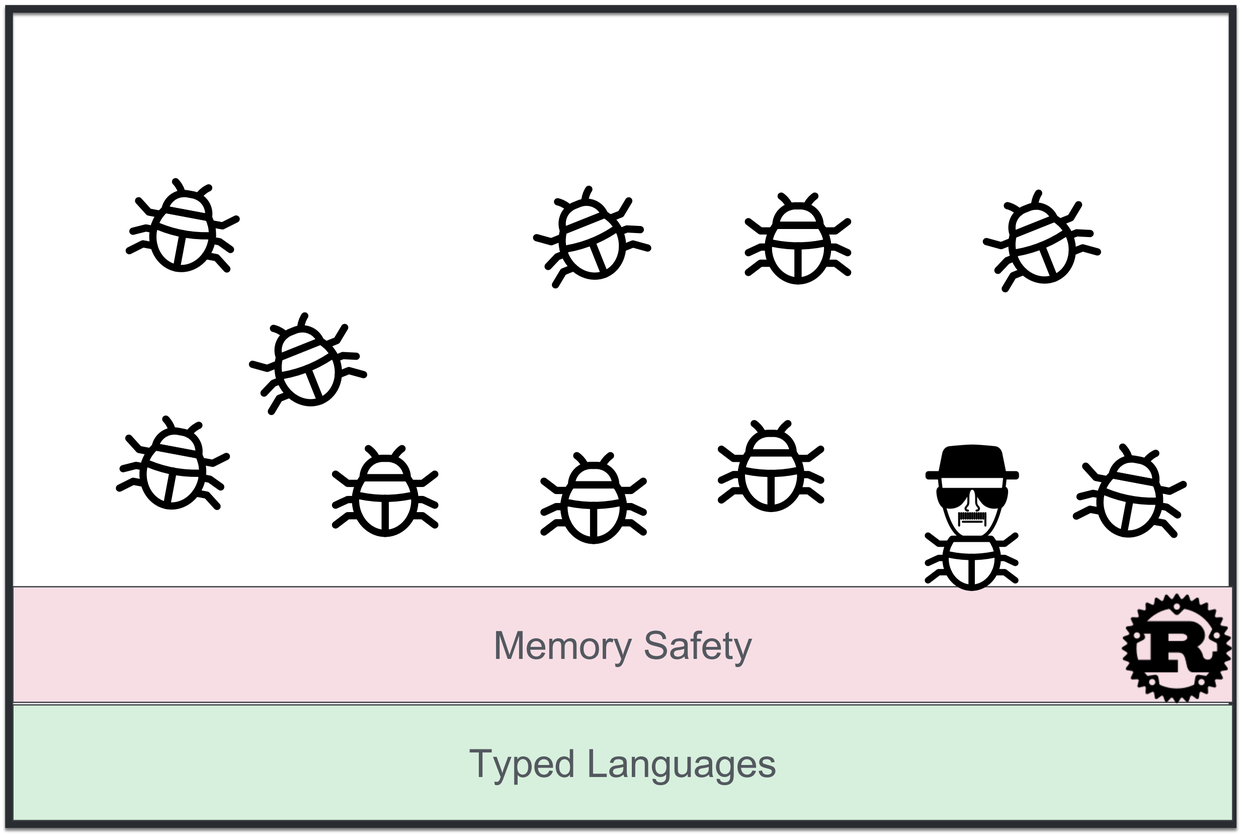
Ceiling: Proof of Correctness




Floor: Assembly code

# Automated Reasoning in a Nutshell

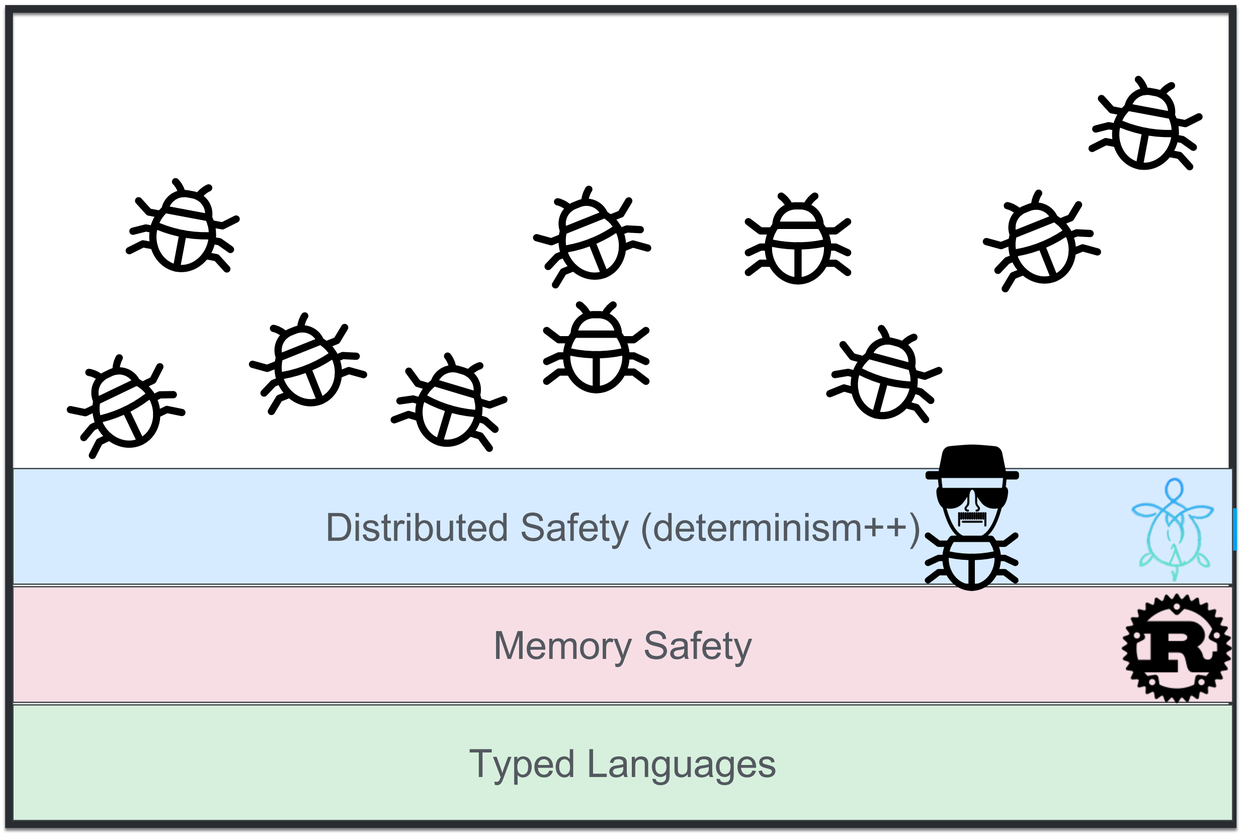
Ceiling: Proof of Correctness




Floor: Assembly coe

# Automated Reasoning in a Nutshell

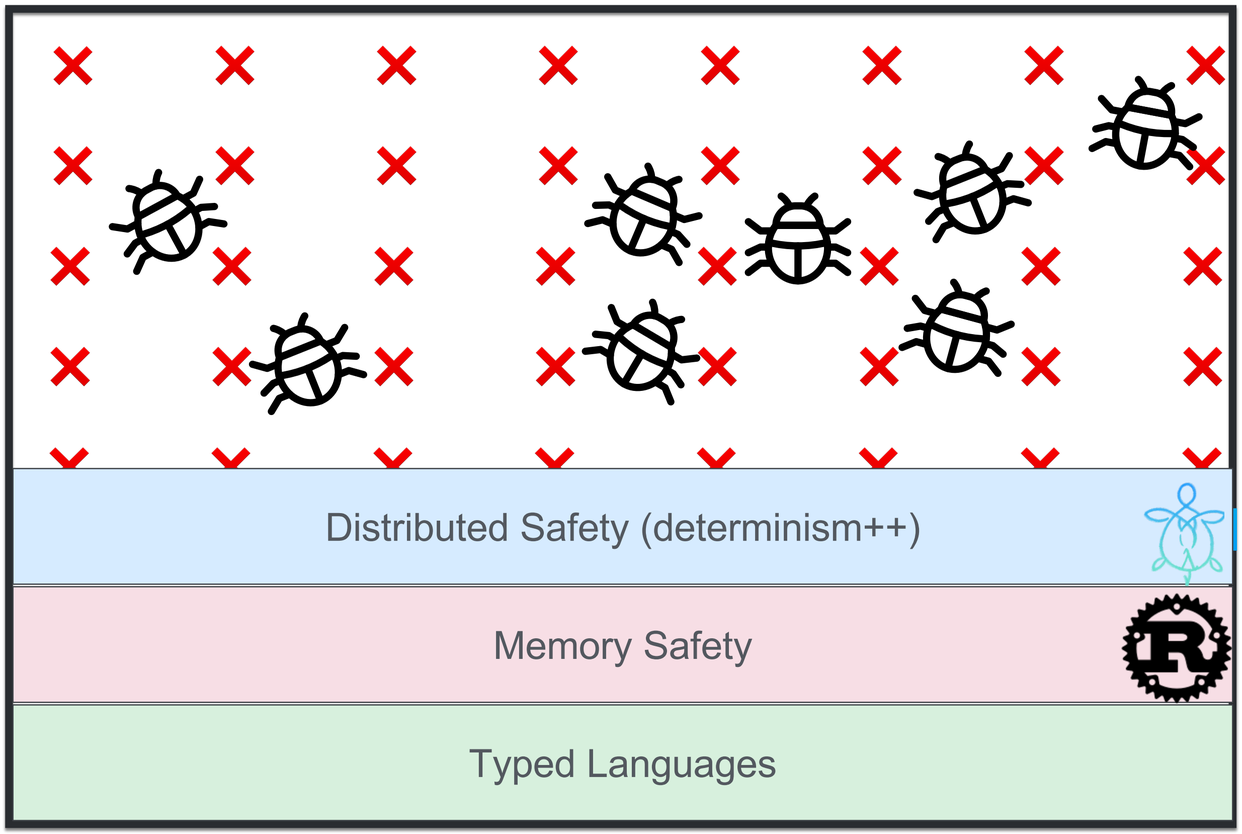
Ceiling: Proof of Correctness




Floor: Assembly code

# Automated Reasoning in a Nutshell

Ceiling: Proof of Correctness

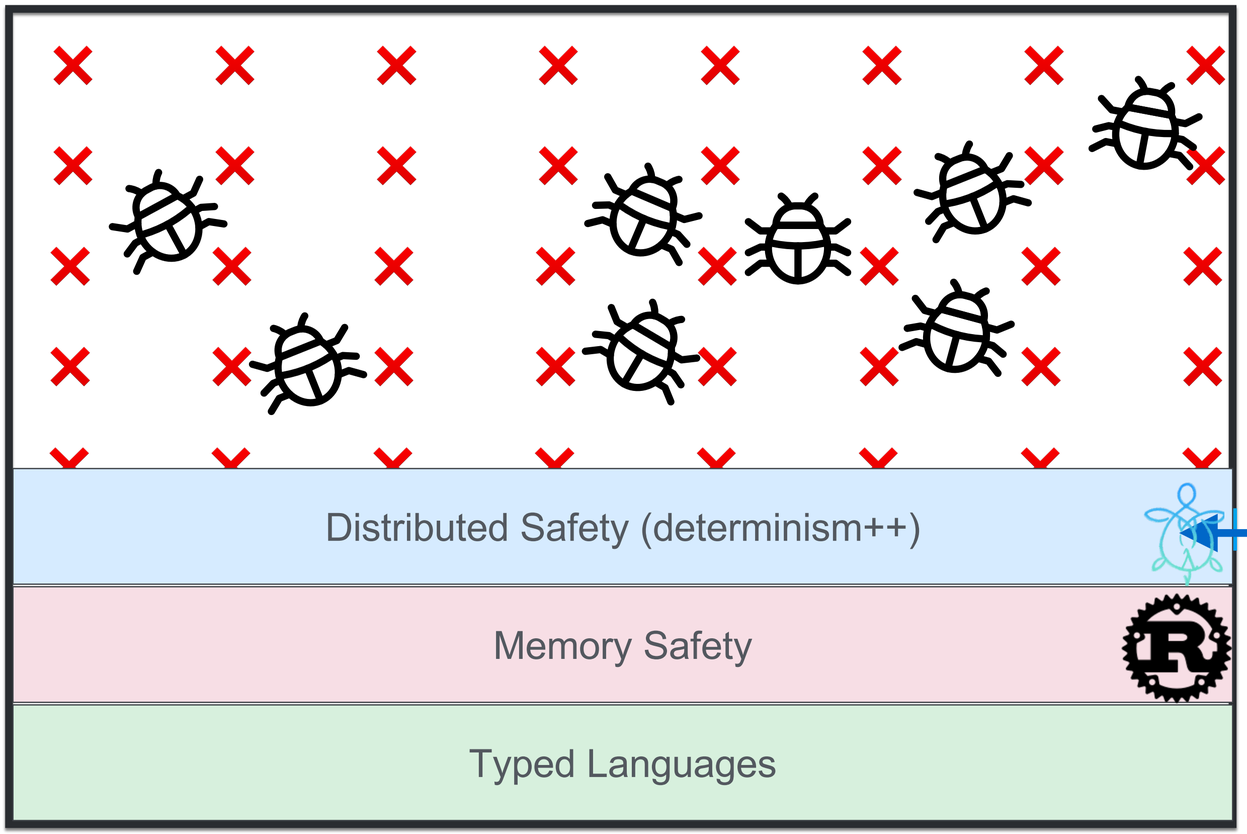



Floor: Assembly Code

Testing:
Finite Model Checking, Fuzzing

# Automated Reasoning in a Nutshell

Ceiling: Proof of Correctness




Floor: Assembly code

Testing:
Model Checking, Fuzzing

Hydro
sim

The code *is* the spec.

Unit tests are the invariants.

# A Broader Perspective on Declarativity

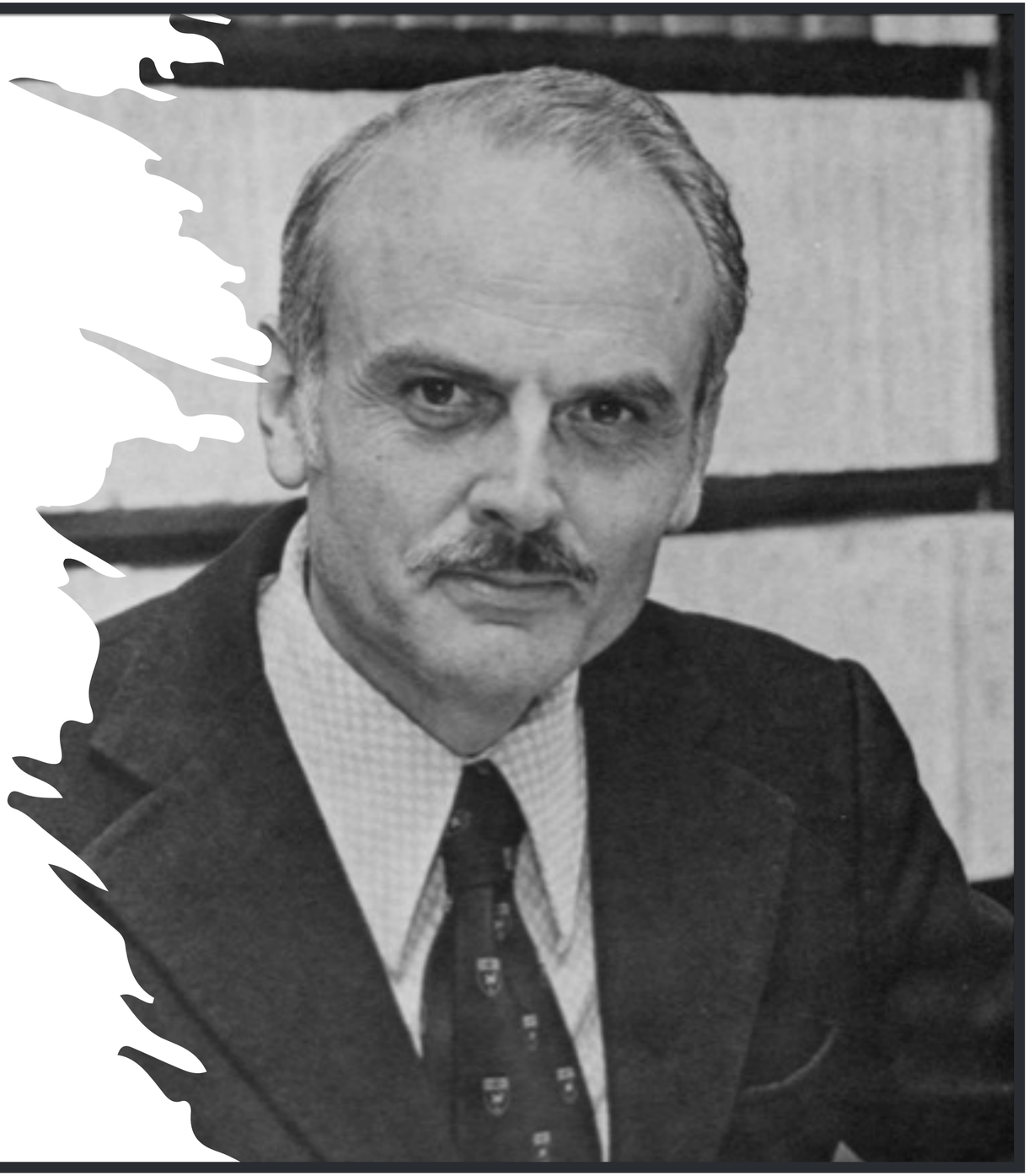
Codd Award Talk SIGMOD 2023

JOE HELLERSTEIN

# Codd’s Lesson

Formalize specification;

automate implementation.



# The Narrow Waist



# DEEPER THEORY

## Determination Complexity

- Orthogonal to computation
- Examples
  - Replicated State Machines: 1 commitment
  - Transactions: linear commitments
- Metrics
  - Work: #commitments
  - **Depth**: #parallel rounds of commitment

## Determination Provenance

- Example:
  - Why did my account balance go below 0?
- Record *commitments* in a log
- Explain output wrt which commitments:
  - **Why? Why not?**
  - **What if? How likely?**

# Takeaways

Yes, we're hiring at AWS.

- **Ambiguity** introduces significant costs in computing
  - Agreement, Tokens, Locks, Bids, Moves, Rounds, …
- **Determination Theory** gives that cost a *formal & practical* frame
  - Settles classic distributed systems questions, including Complete CALM and CAP
  - Checkable at compile time! Hydro type-checks proper coordination in real Rust programs
- **Determination Complexity** is orthogonal to computation
  - A new axis, with new tradeoffs
- **Determination Provenance** explains behavior of real, nondeterministic systems
  - Why, Why Not, and weighted responsibility

*fin*