Proof-Carrying Data
and Hearsay Arguments from Signature Cards

Alessandro Chiesa and Eran Tromer

Innovations in Computer Science 2010
January 6, 2010

Motivation

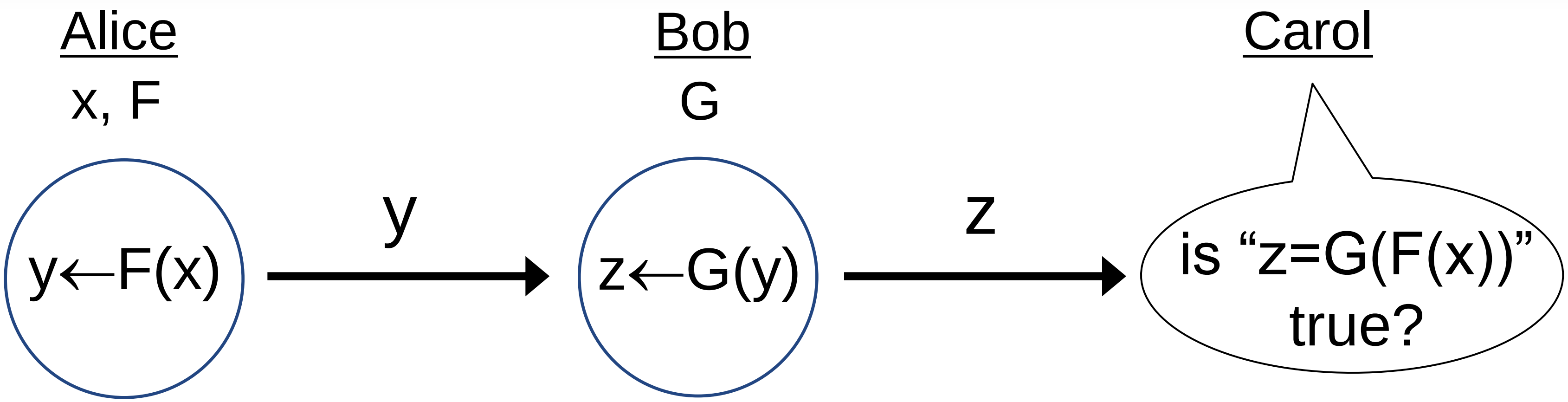
System and protocol security often fail when assumptions about software, platform, and environment are violated.

bug attacks	bugs	physical side channels
architectural side channels	trojans	tampering
hardware trojans		

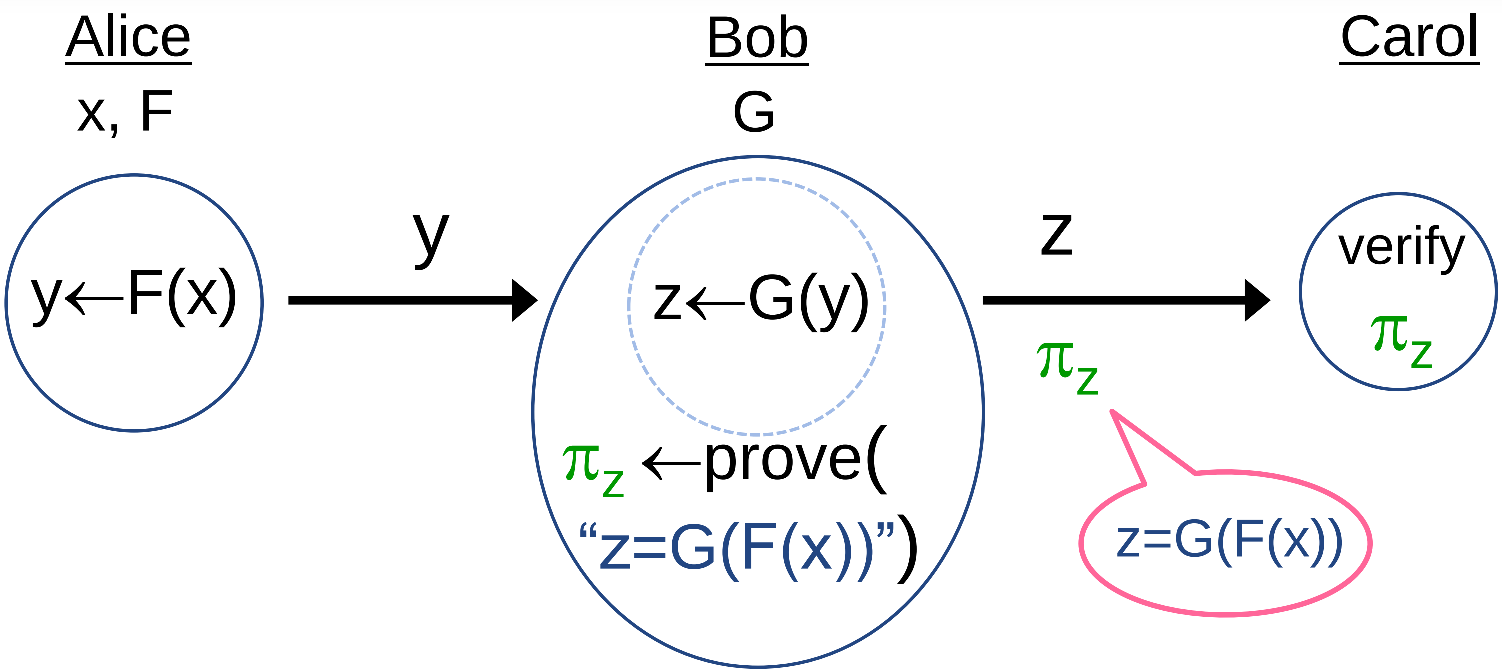
High-level goal

Ensure properties of a distributed computation when parties are mutually untrusting, faulty, leaky & malicious.

Example: 3-party correctness



Example: computationally-sound (CS) proofs [Micali 94]

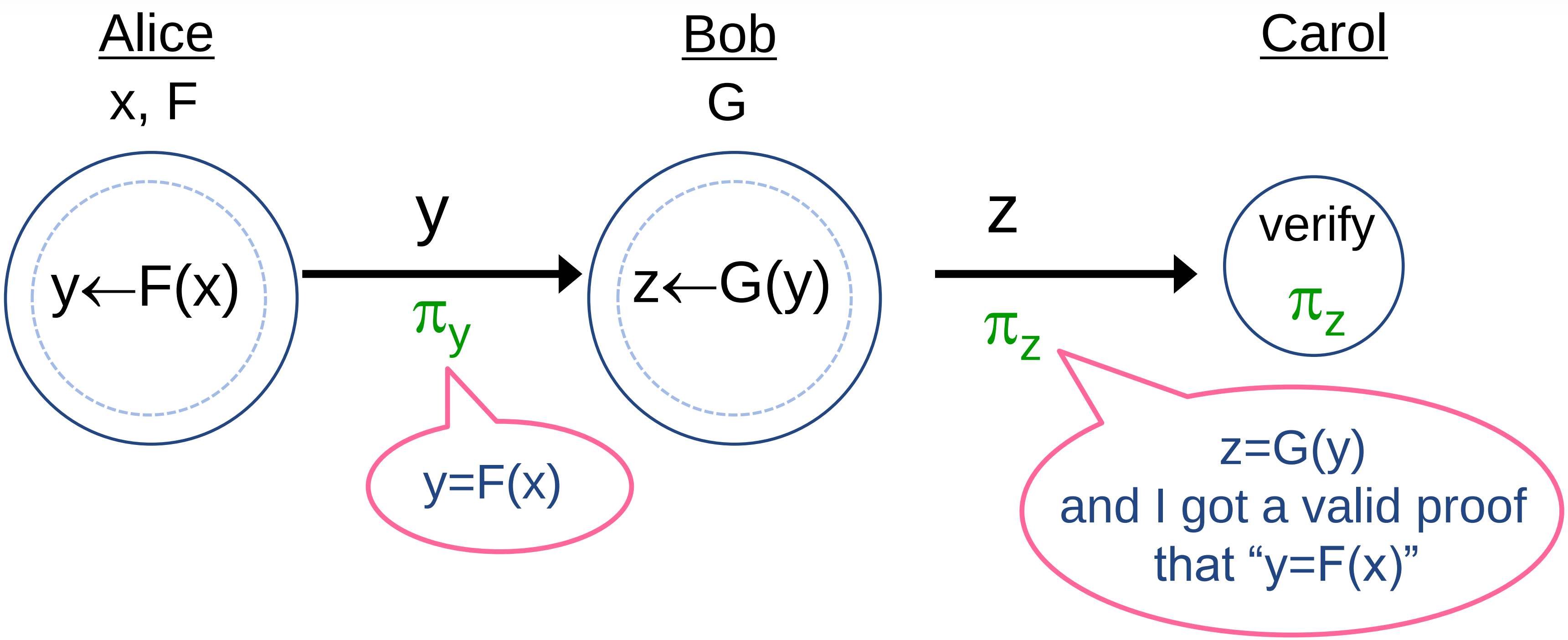


Bob can generate a proof string that is:

- Tiny (polylogarithmic in his own computation)
- Efficiently verifiable by Carol

However, now Bob recomputes everything...

Example: Proof-Carrying Data [Chiesa Tromer 09] following Incrementally-Verifiable Computation [Valiant 08]



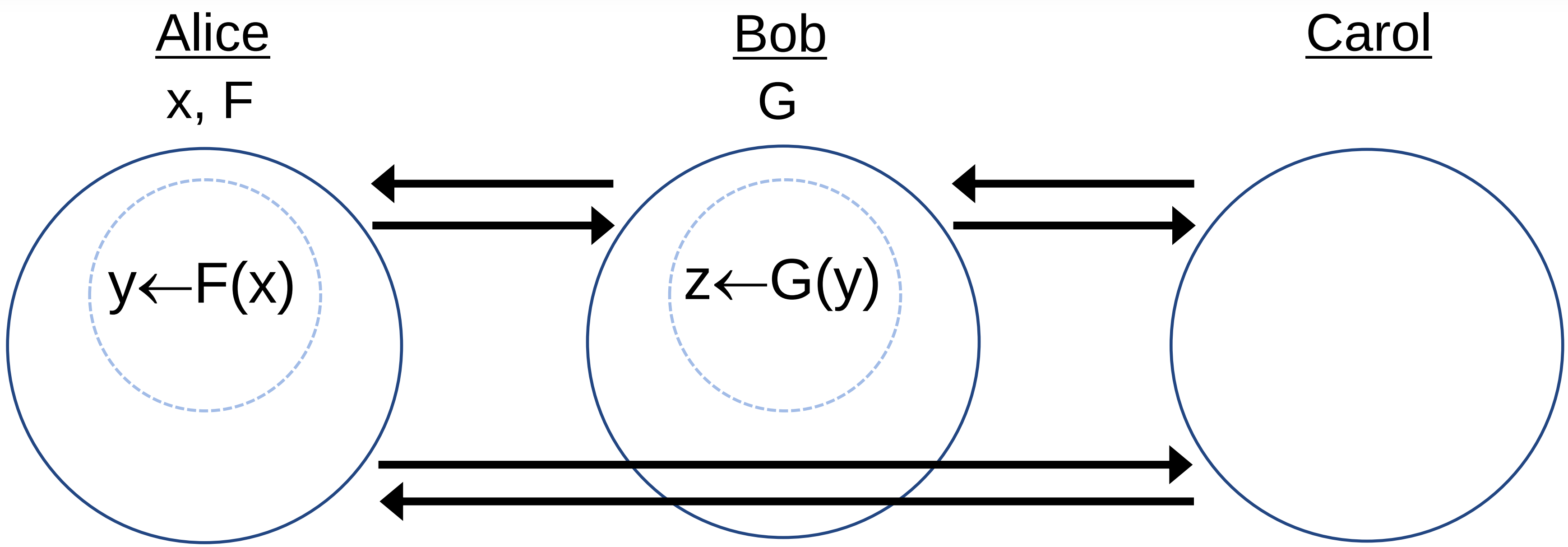
Each party prepares a proof string for the next one.

Each proof is:

- Tiny (polylogarithmic in party's own computation).
- Efficiently verifiable by the next party.

Related work:

Secure multiparty computation [GMW87][BGW88][CCD88]



But:

- computational blowup is polynomial in the whole computation, and not in the local computation
- does not preserve communication graph
- parties and computation must be fixed in advance

CSAIL MIT

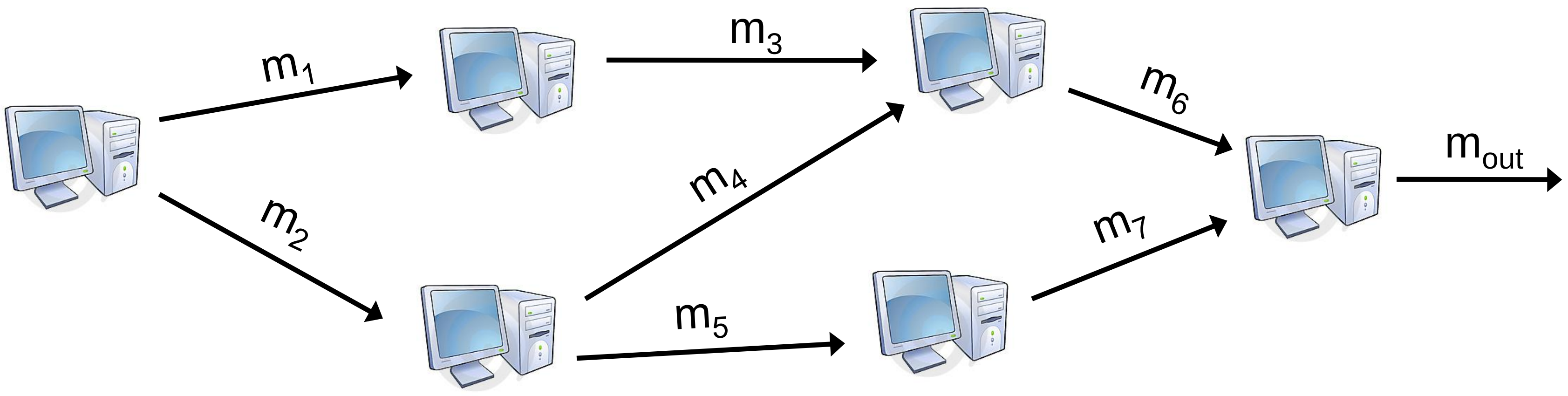
Generalizing:

The Proof-Carrying Data framework

Generalizing: distributed computations

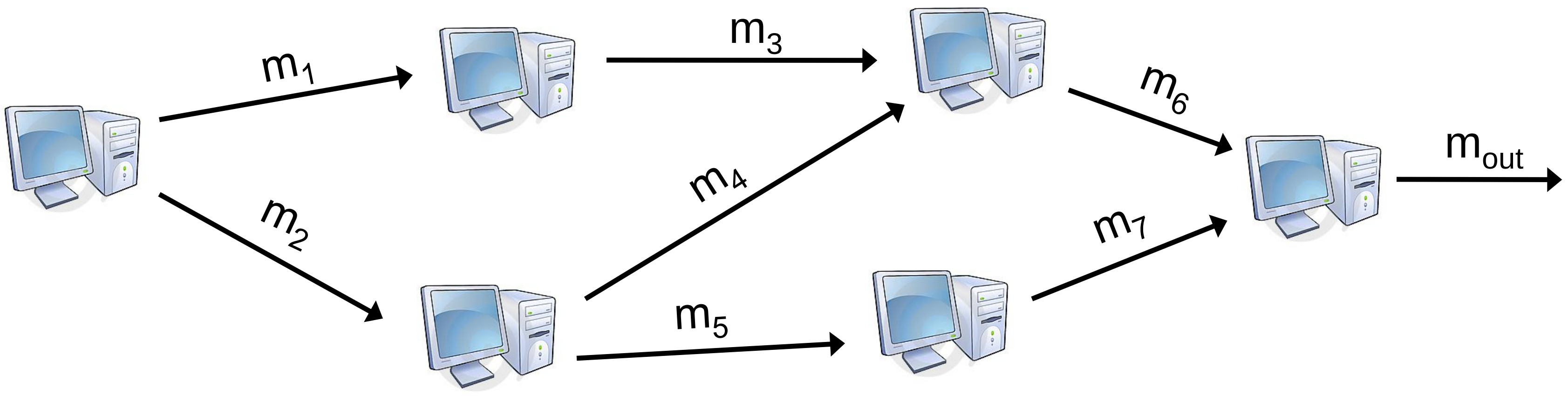
Distributed computation?

Parties exchange messages and perform computation.



Generalizing: arbitrary interactions

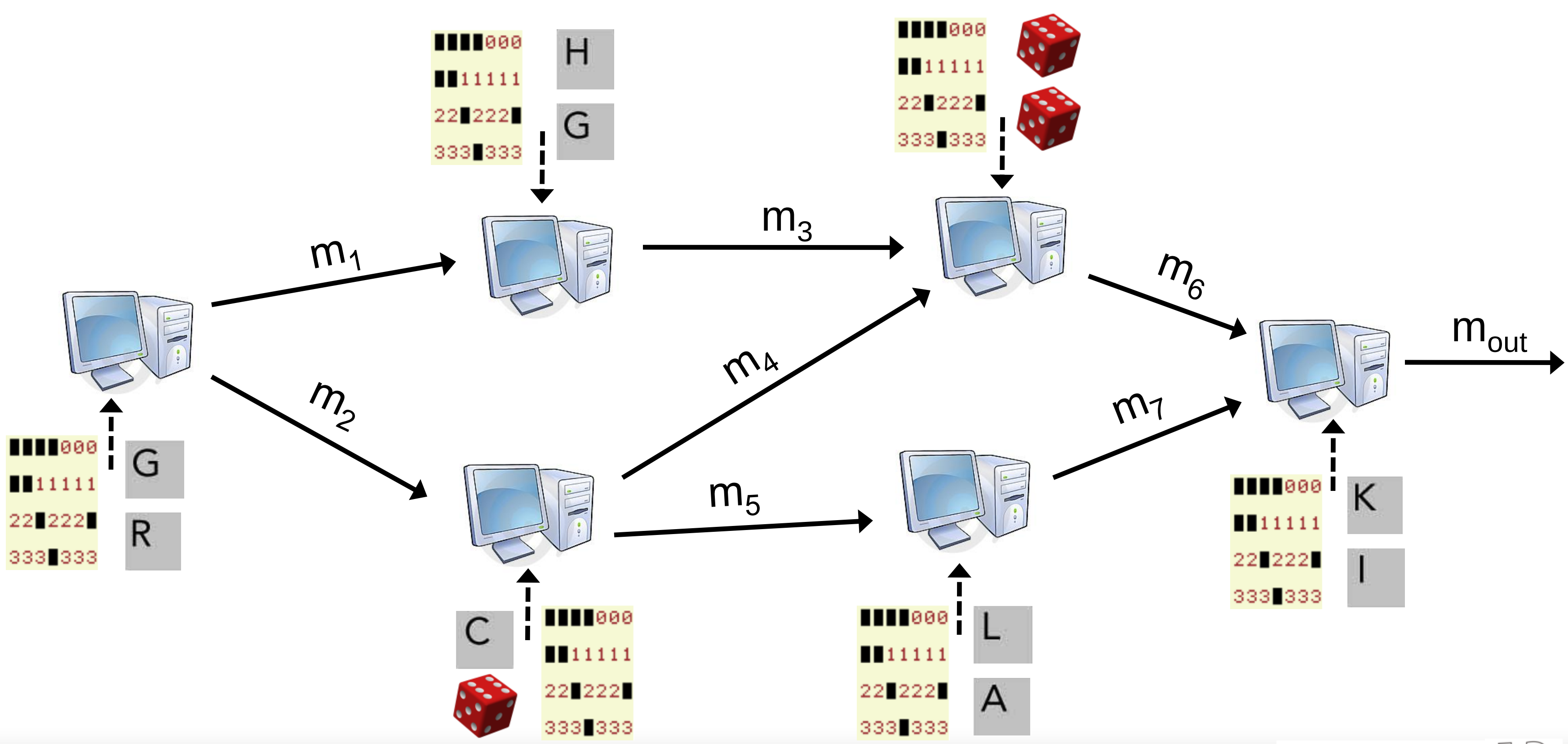
- Arbitrary interactions
  - communication graph over time is any DAG



Generalizing: arbitrary interactions

- Computation and graph are determined on the fly
  - by each party's local inputs:

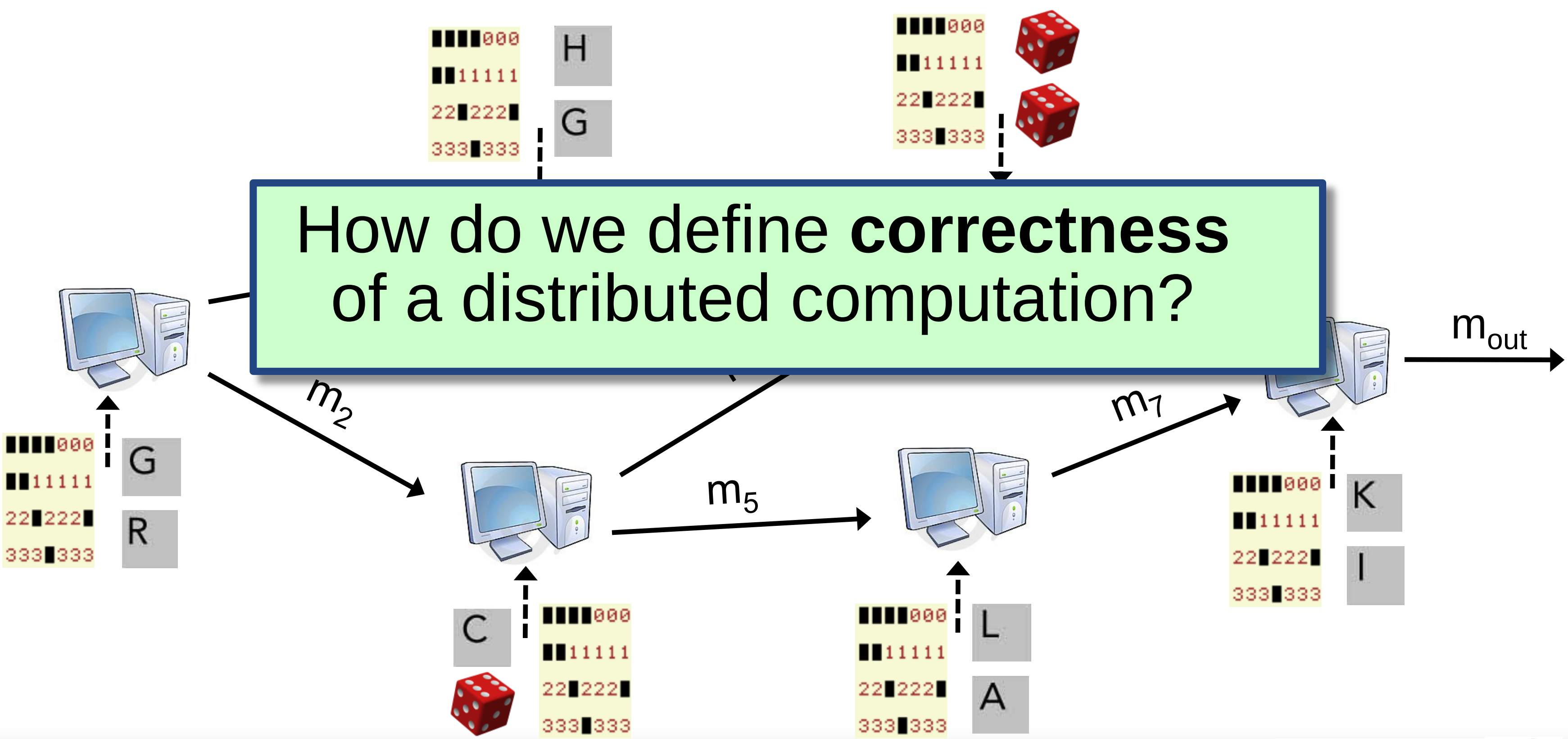
human inputs randomness program



Generalizing: arbitrary interactions

- Computation and graph are determined on the fly
  - by each party's local inputs:

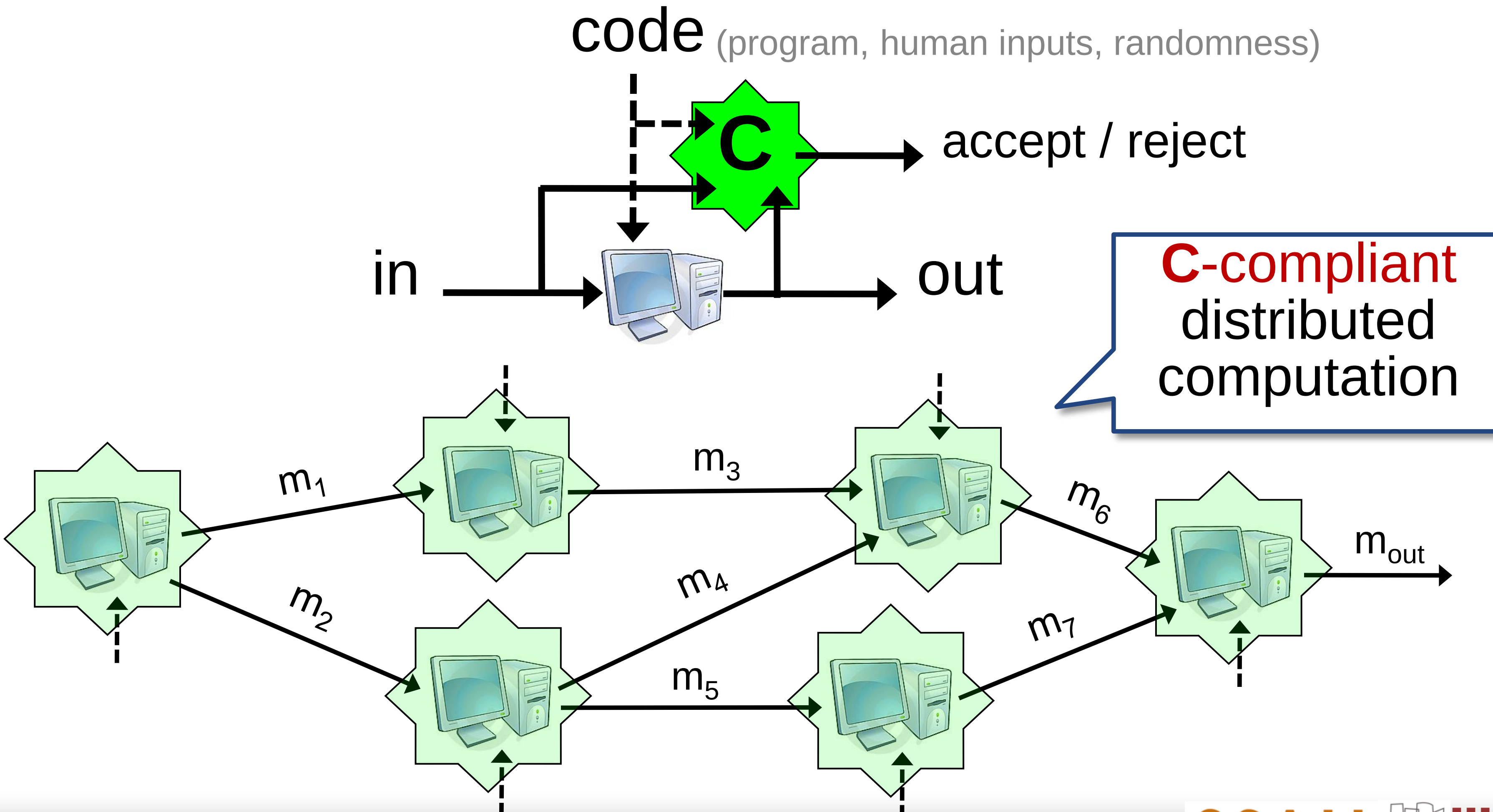
human inputs randomness program



How do we define **correctness** of a distributed computation?

C-compliance

correctness is a **compliance predicate** **C**(in,code,out) that must be locally fulfilled at every node



C-compliance

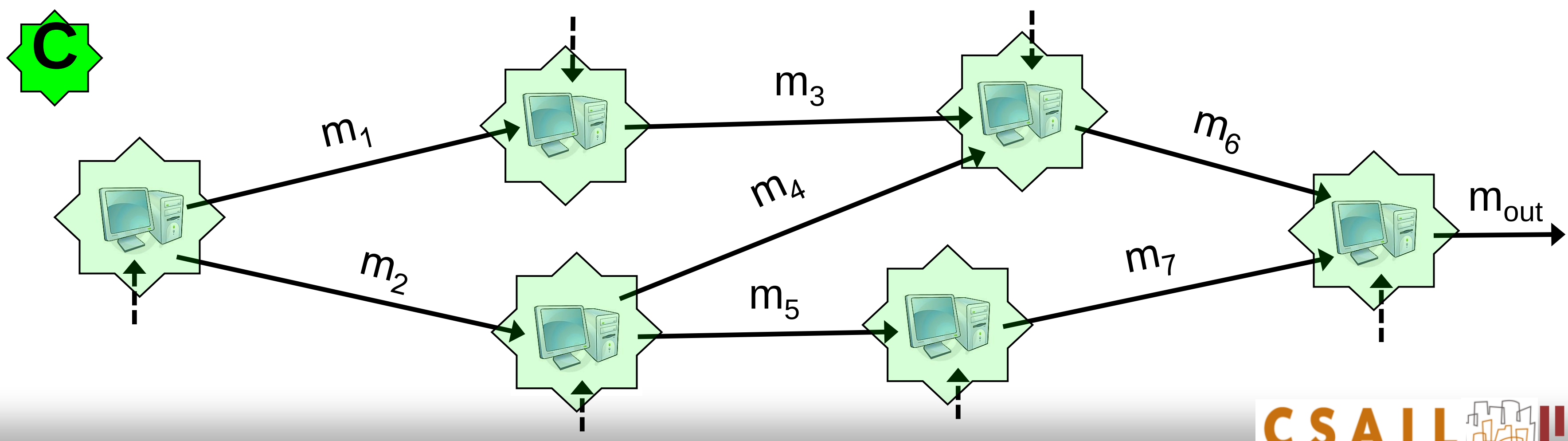
correctness is a **compliance predicate** **C**(in,code,out)
that must be locally fulfilled at every node

Some examples:

C = "none of the inputs are labeled `secret`"

C = "the code was digitally signed by the sysadmin, and executed correctly"

C = "the code is type-safe and the output is indeed the result of running the code"



CSAIL MIT

Goals

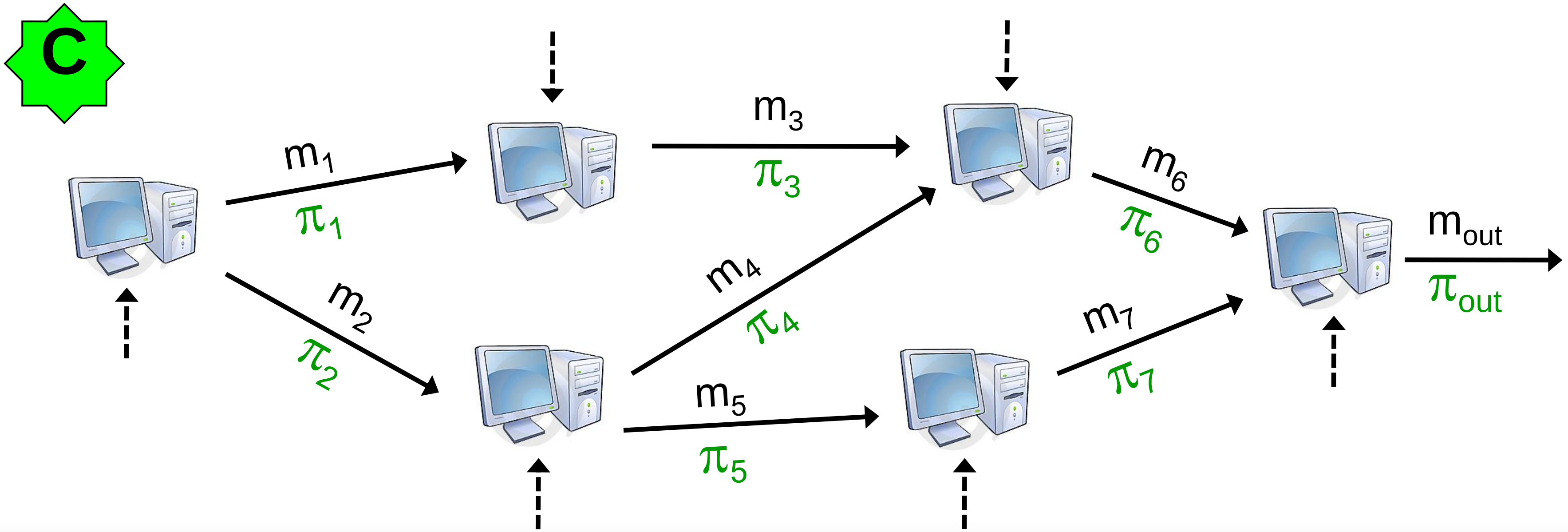
Ensure **C**-compliance while **respecting** the original distributed computation.

- Allow for any interaction between parties
- Preserve parties' communication graph
  - no new channels
- Allow for dynamic computations
  - human inputs, indeterminism, programs
- Blowup in computation and communication is local and polynomial

Dynamically augment computation with proofs strings

In PCD, messages sent between parties are augmented with concise proof strings attesting to their "compliance".

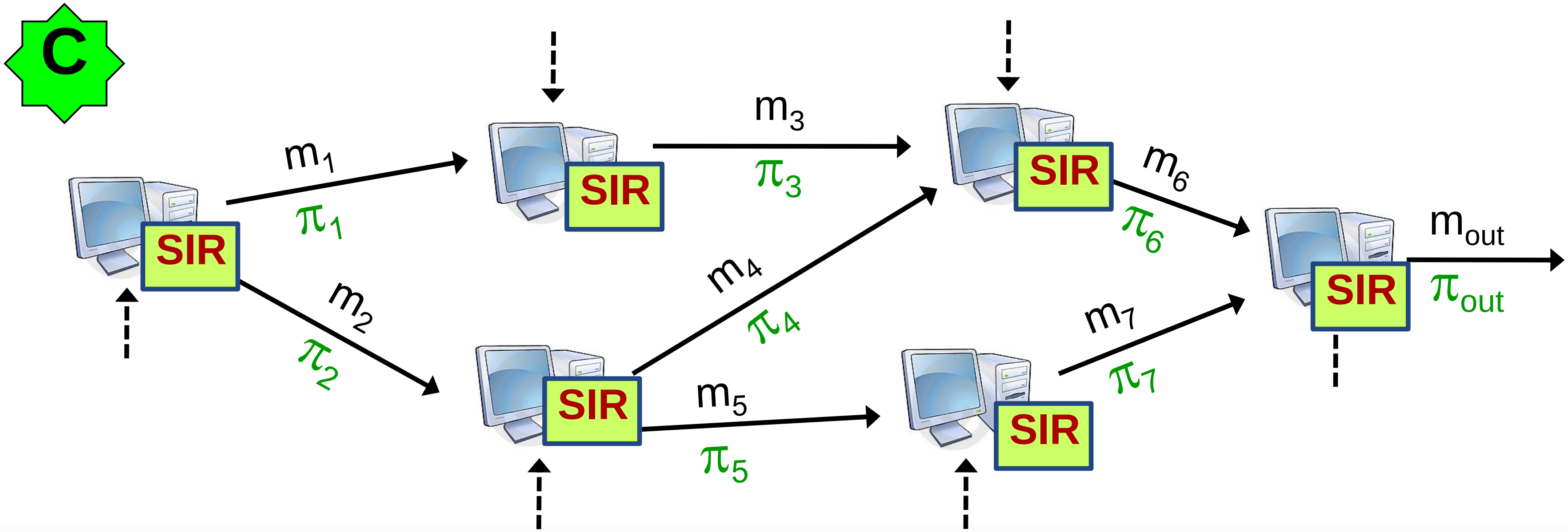
Distributed computation evolves like before, except that each party also generates on the fly a proof string to attach to each output message.



Model

Every node has access to a simple, fixed, stateless trusted functionality -- essentially, a signature card.

- Signed-Input-and-Randomness (SIR) oracle



Model

Every node has access to a simple, fixed, trusted functionality -- essentially, a signature card.

- Signed-Input-and-Randomness (SIR) oracle

x — input string; s — length →

SIR_{SK}

$r \leftarrow \{0,1\}^s$

$\sigma \leftarrow SIGN(SK,(x,r))$

← r — random string; σ — signature on (x,r)

Similar assumptions:
[Hofheinz Müller-Quade Unruh 05]

— **VK**

CSAIL MIT

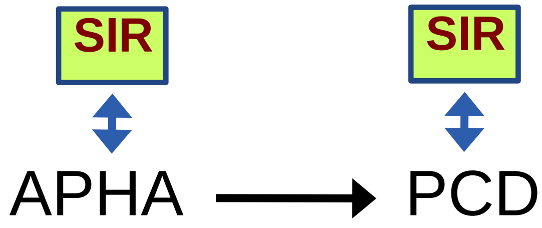
Sample application: type safety

C(in,code,out) verifies that
code is type-safe & out=code(in)

- Using PCD, type safety can be maintained
  - even if underlying execution platform is untrusted
  - even across mutually untrusting platforms
- Type safety is very expressive
  - Can express any computable property
  - Extensive literature on types that can be verified efficiently (at least with heuristic completeness, which is good enough)
  - E.g., can do certain forms of confidentiality via IFC

Our results

Overview of Results



Proof-Carrying Data (PCD):

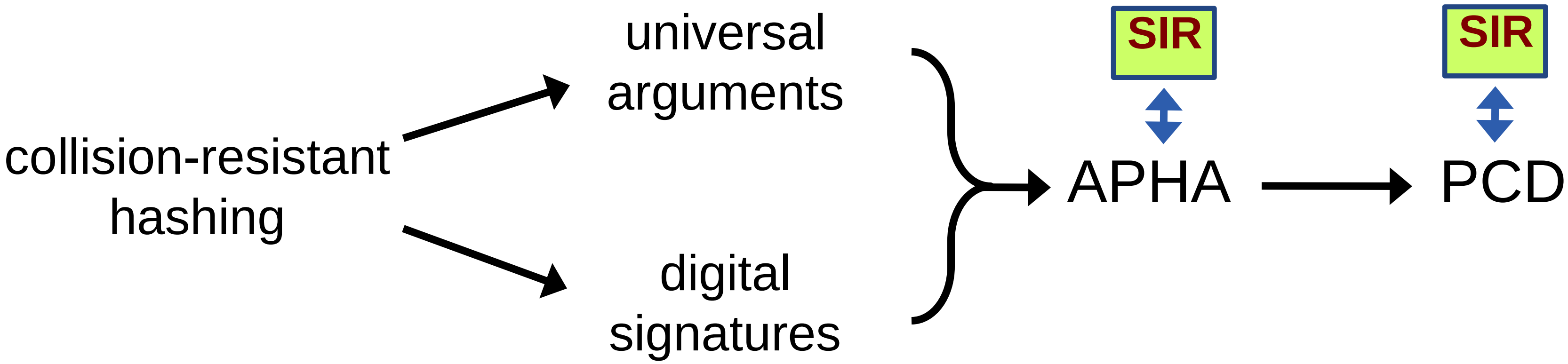
- **C**-compliance
- Aggregate proof strings to generate new ones
- Simpler interface hides implementation details

Assisted-Prover Hearsay-Arguments (APHA):

- Very strong variant of non-interactive CS proofs / arguments of knowledge (for NP)
- Proof system for a “single step”

CSAIL MIT

Overview of Results



Need:

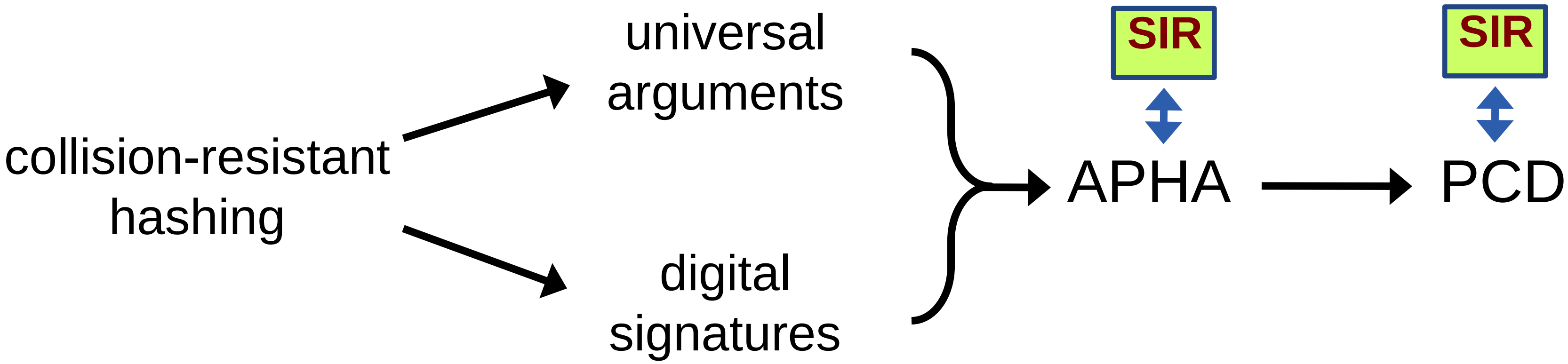
- Universal arguments (CS proofs) that are public-coin and constant-round [Barak Goldreich 02] [Micali 94]
- Signature schemes that are strongly unforgeable

generic (from UOWHFs): [Goldreich 04]

efficient: [Boneh Shen Waters 02]

Both exist if Collision Resistant Hash schemes exist.

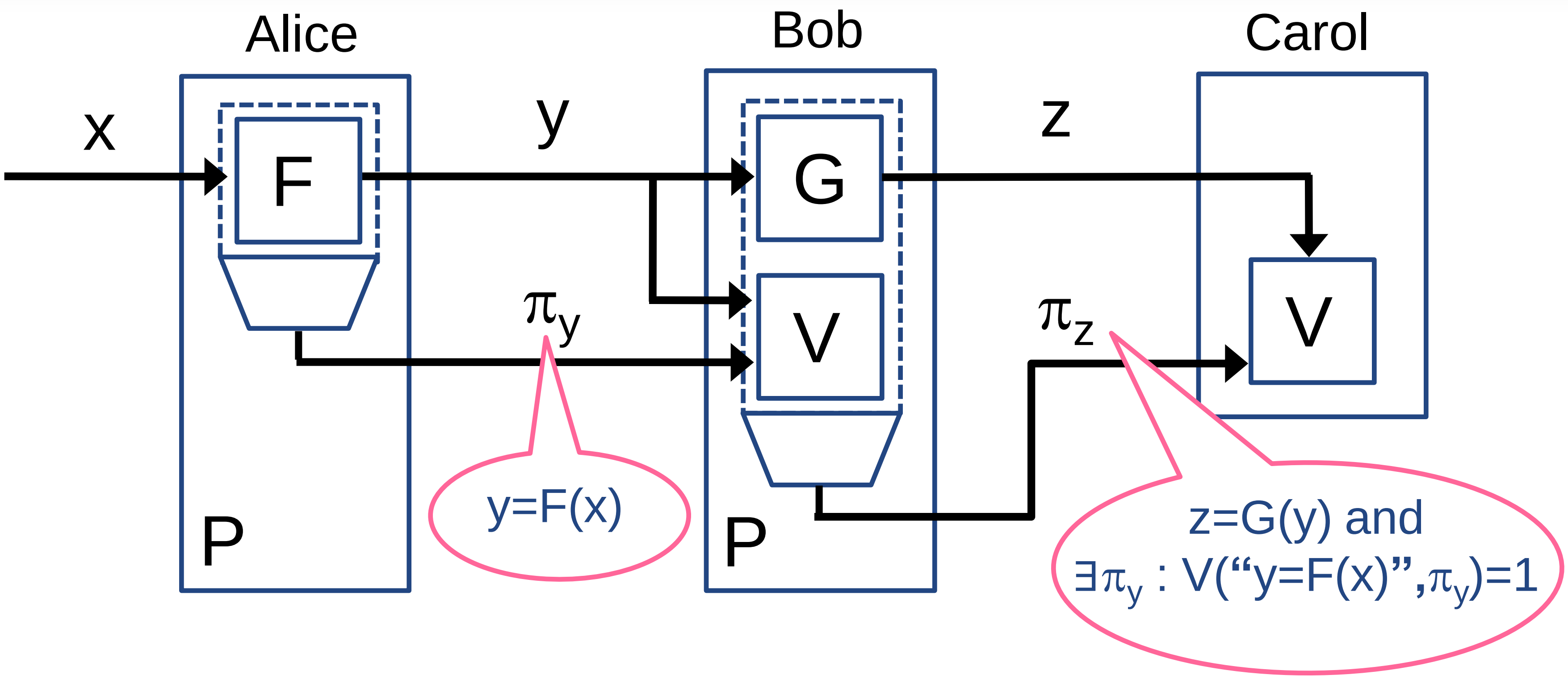
Rest of this talk



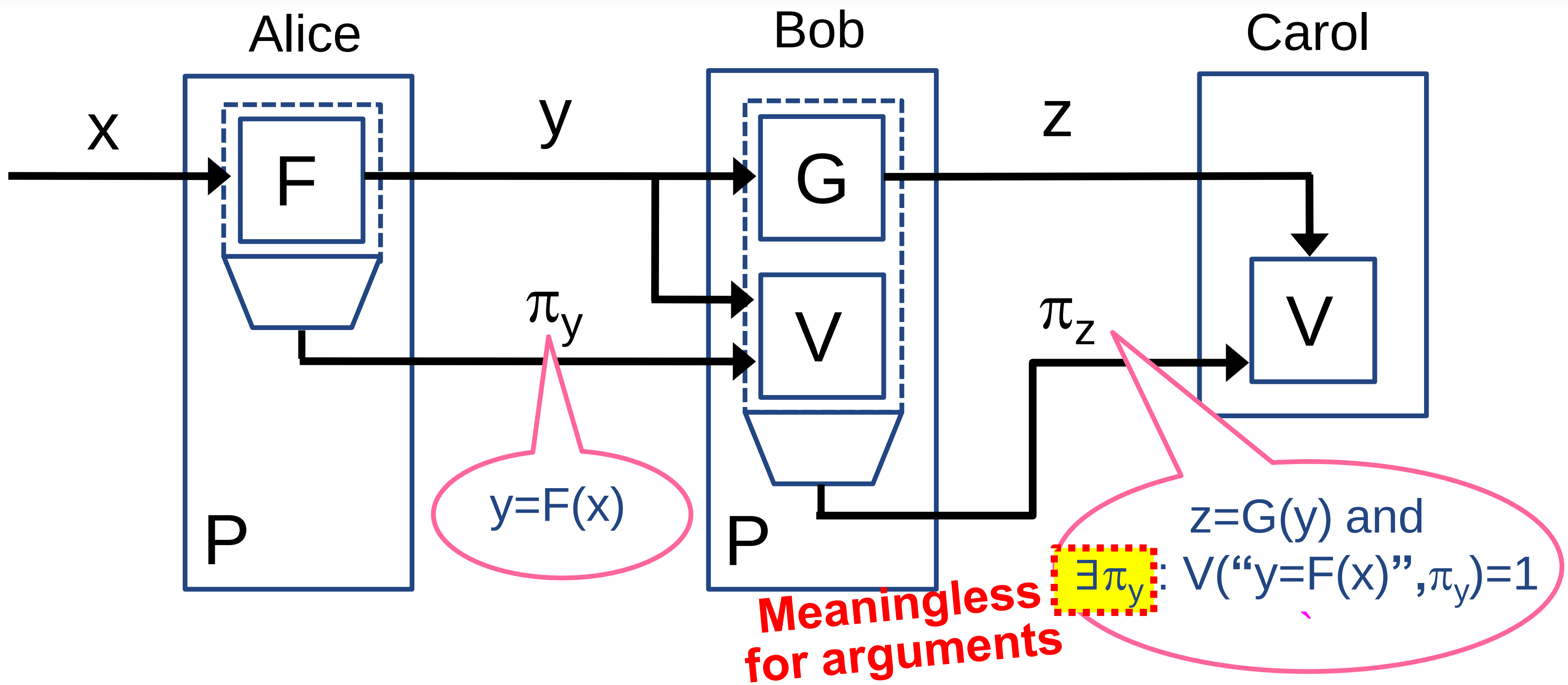
Rest of this talk:

- Intuition on how to aggregate proofs in “F and G” example

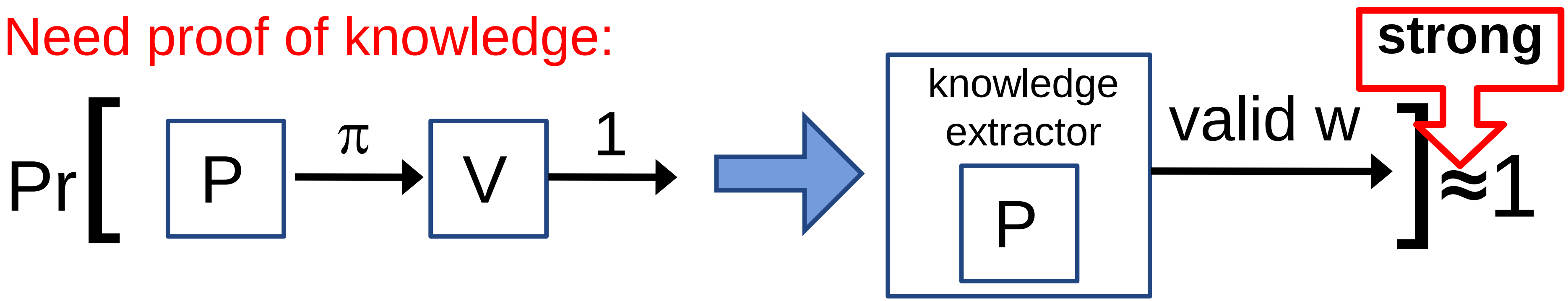
Proof aggregation



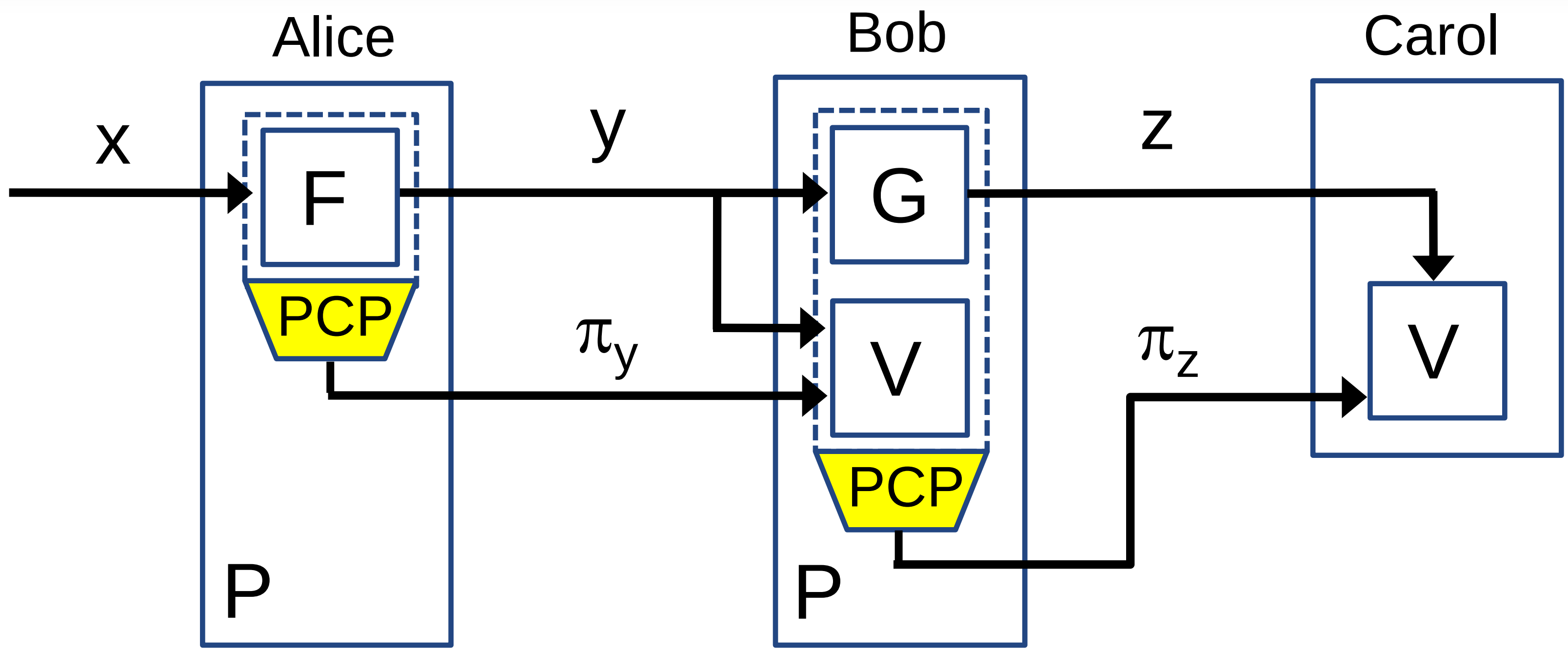
Soundness vs. proof of knowledge



Need proof of knowledge:



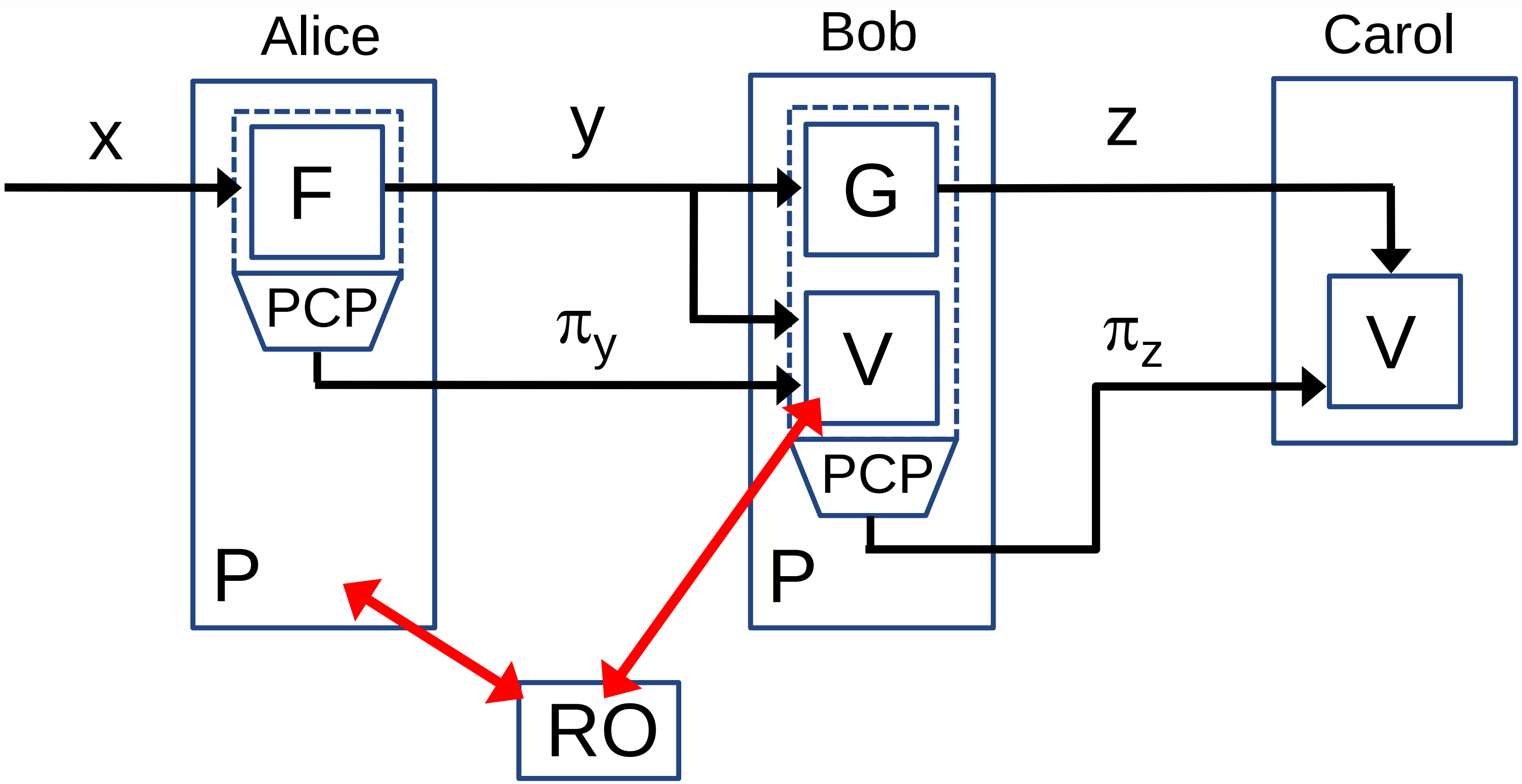
Must use PCPs for compression



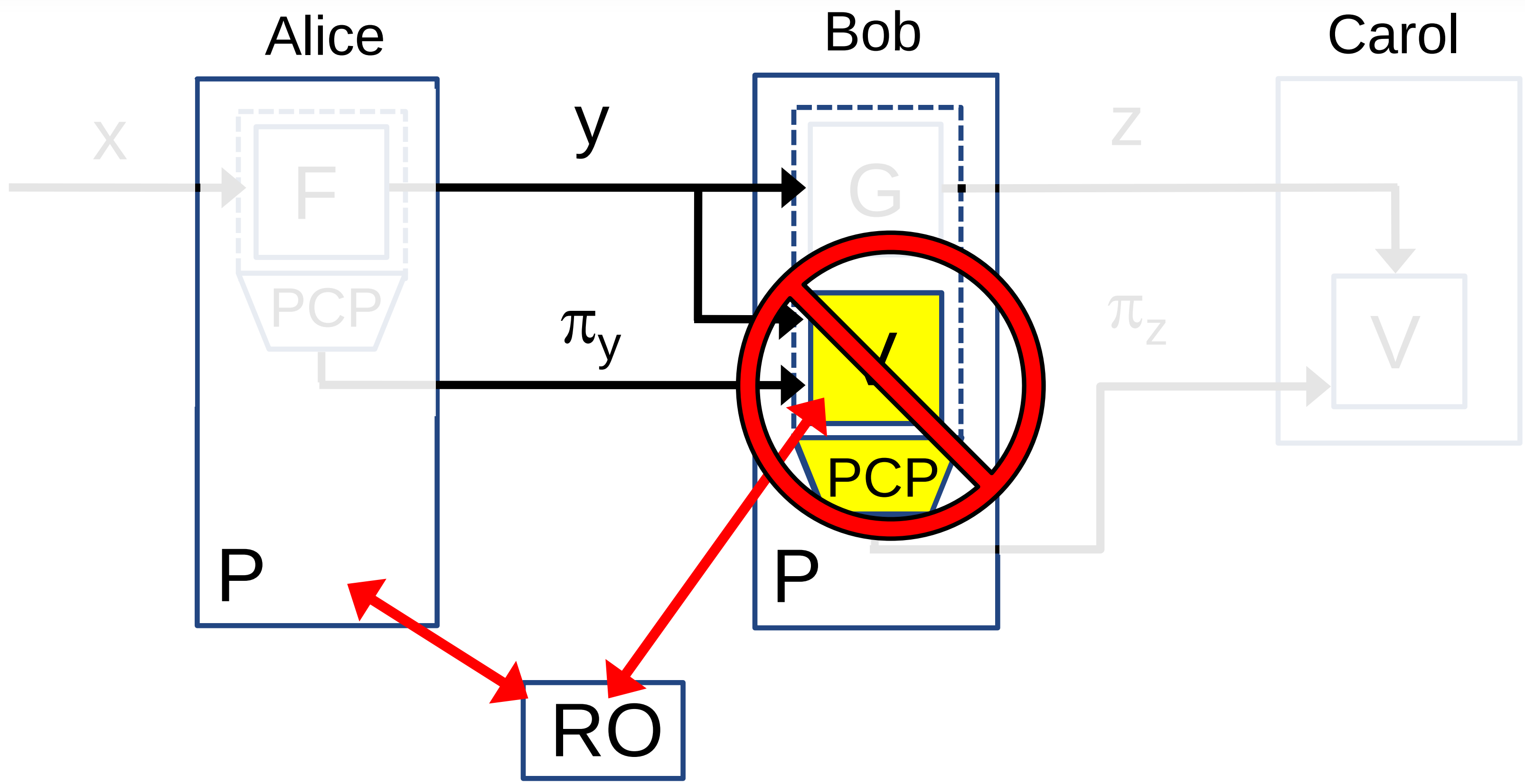
- Probabilistically Checkable Proofs (PCPs) used to generate concise proof strings.

(And there is evidence this is inherent [Rothblum Vadhan 09].)

Must use oracles for non-interactive proof of knowledge

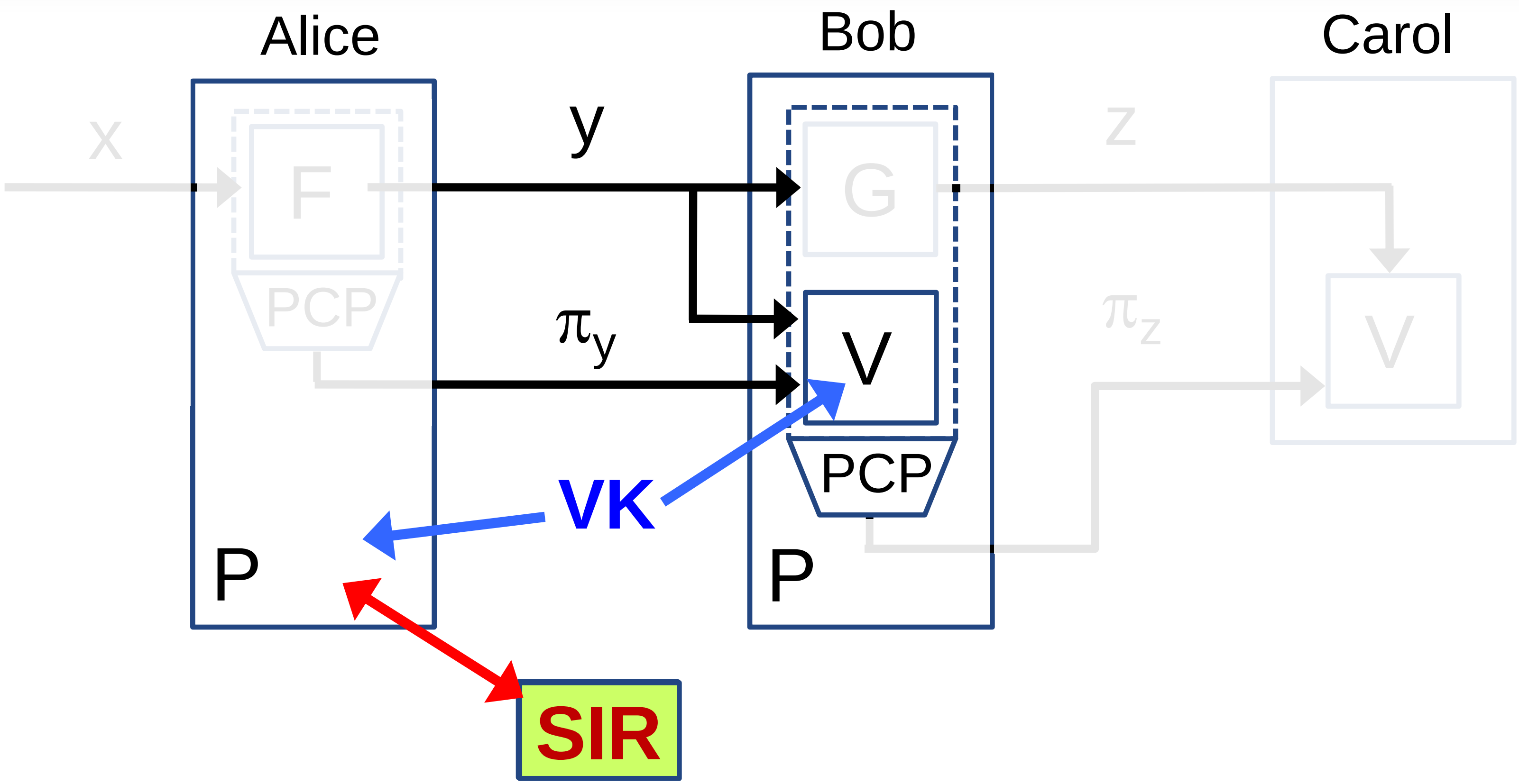


PCP vs. oracles conflict



- PCP theorem does **not** relativize [Fortnow '94], not even with respect to a RO [Chang et al. '92]
- this precluded a satisfying proof of security in [Valiant '08]

Our solution:
Public-key crypto to the rescue



Oracle signs answers using public-key signature:

- answers are verifiable without accessing oracle
- asymmetry allows us to break “PCP vs. oracle” conflict, and recursively aggregate proofs

Sketch of remaining constructions

Constructing APHAs:

≈ Fiat-Shamir heuristic

- Start with universal arguments
- De-interactivize by replacing public-coin messages with oracle queries
- Add signature to statement to force witness query (≈ [Chandran et al. 08])
- Prove a very strong PoK by leveraging the weak PoK of UA

Generalizing to PCD:

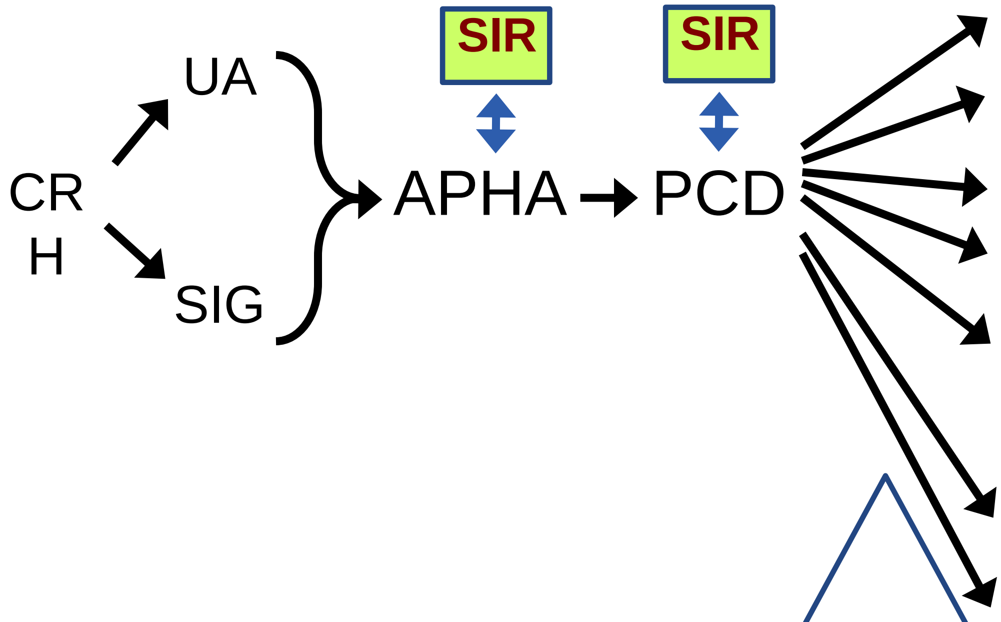
- Handle distributed-computation DAG,
  using APHA for the proofs along each edge.
- **C**-compliance: use fixed aggregation rule to reason about arbitrary computation by proving statements of the form:

  C(in,code,out)=1 & "each input carries a valid APHA proof string"

CSAIL MIT

Discussion

Applications



- type safety
- fault isolation & accountability
- multilevel security
- financial systems
- distributed dynamic program analysis
- proof-carrying <u>code</u>
- antispam email policies

Security design reduces to "**compliance engineering**": write down a suitable compliance predicate **C**.

Proof-Carrying Data:
Conclusions and open problems

Contributions

- Framework for securing distributed computations between parties that are mutually untrusting and potentially faulty, leaky, and malicious.
- Explicit construction, under standard generic assumptions, in a "signature cards" model.
- Suggested applications.

Ongoing and future work

- Reduce requirement for signature cards, or prove necessity.
- Add zero-knowledge constructions.
- Achieve Practicality (PCPs are notorious for "polynomial" overheads).
- Identify and implement applications.