Standardizing Commit-and-Prove ZK

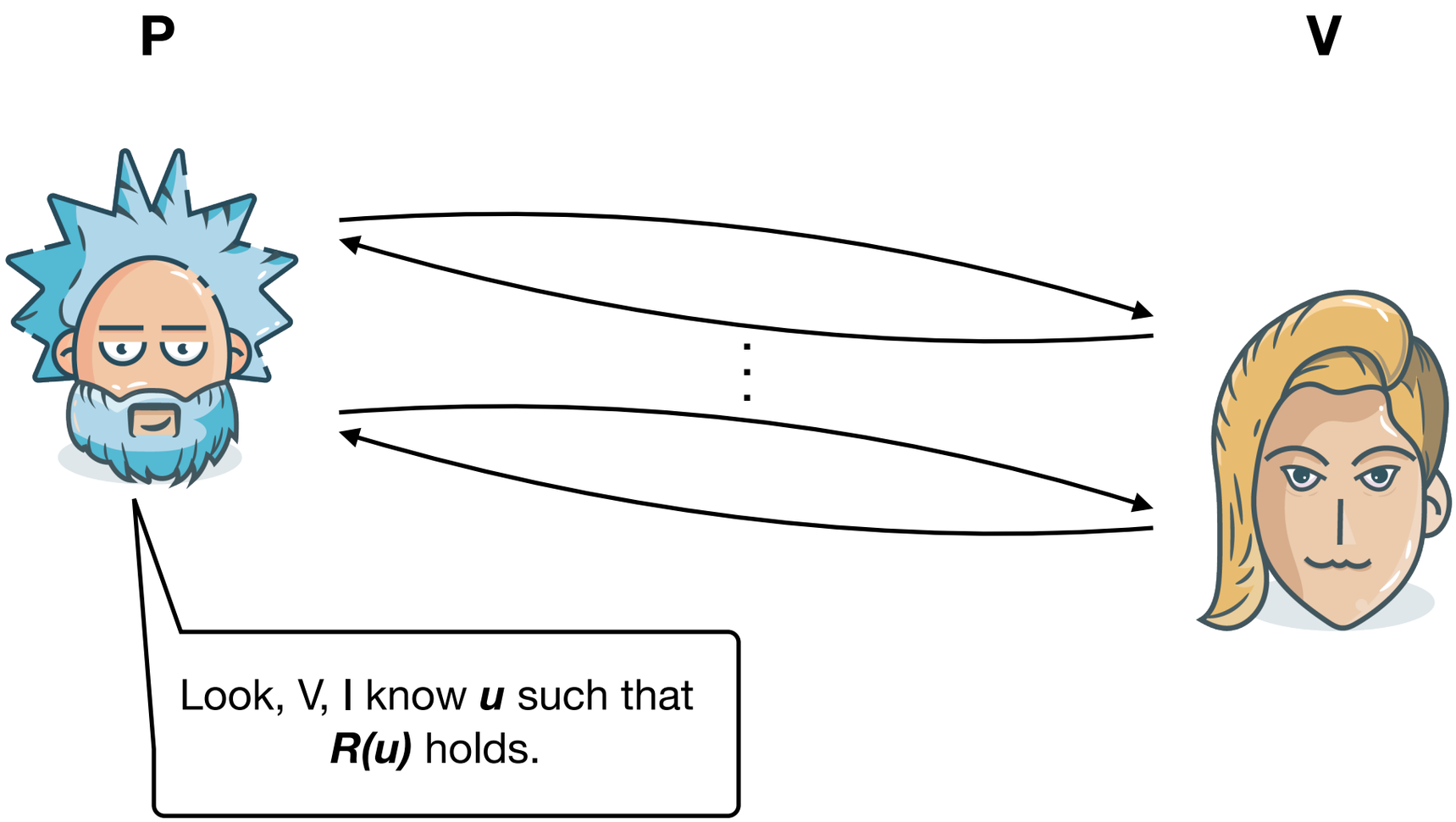
Daniel Benarroch
QEDIT

Matteo Campanelli
IMDEA Software Institute, Madrid

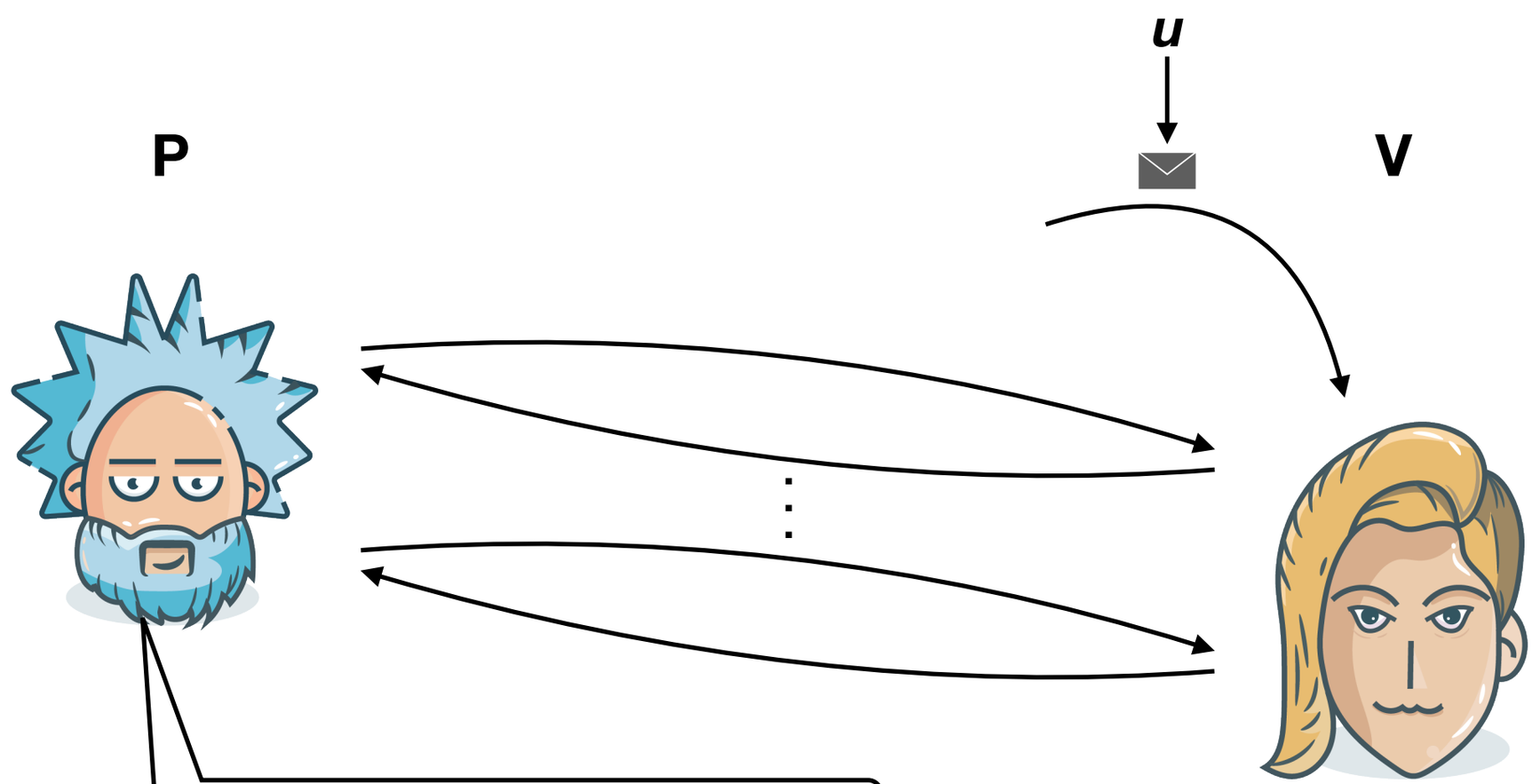
Dario Fiore
IMDEA Software Institute, Madrid

2nd ZKProof Workshop

ZK



Commit-and-Prove (CP) ZK



Look, V, I know ***u*** such that ***R(u)*** holds.

And, by the way,

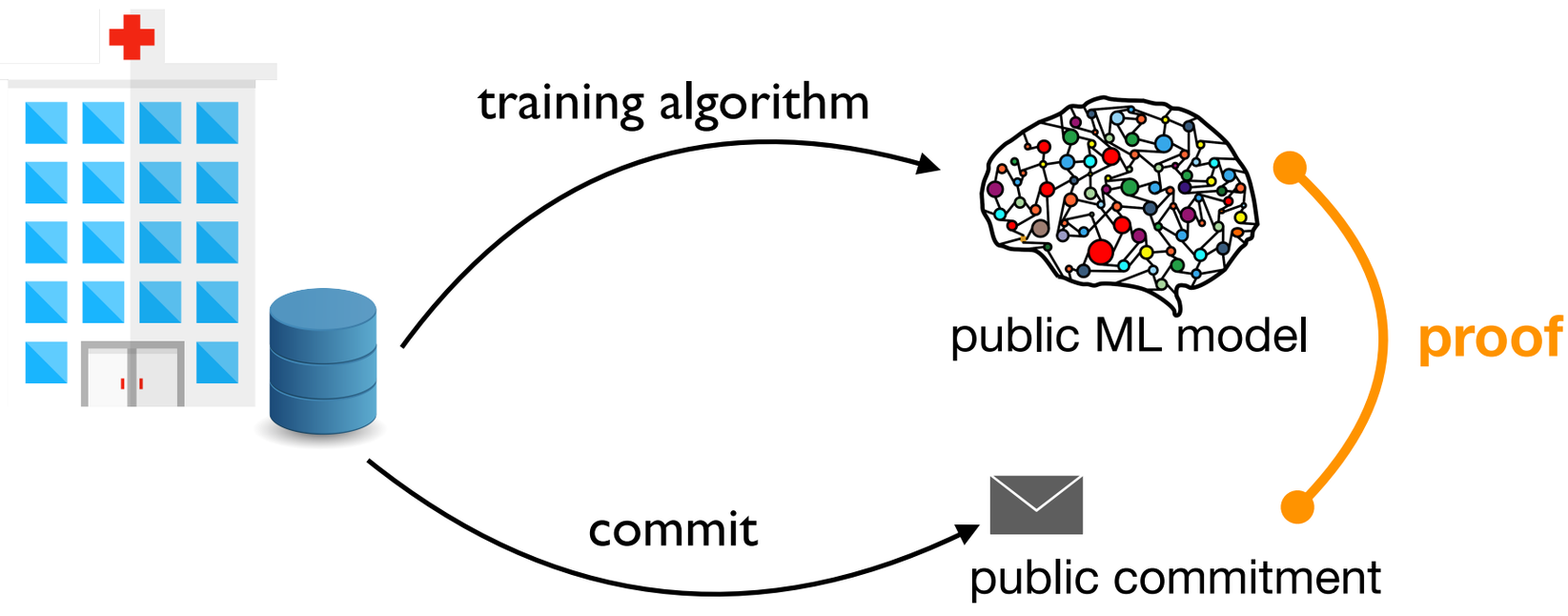
<u>Commit-and-Prove ZKP:</u>

A ZKP for the relation

$$R_{com}(c, u) := R(u) \ \ \texttt{AND} \ \ c \xrightarrow{\texttt{opens}} u$$

Motivation: Soundness + Integrity

One example (from [WZCPS18])



More:

CP in several applications presented in this workshop

Composition of proof systems [**C**FQ19,Folklore]

My goal:

throwing things at you re CP standards.

Caveats on the focus:

Applications

$\boxed{\text{Abstractions}}$

Implementation

Non-Interactive case

Standardizing CP

Why?

What?

How (much)?

Standardizing CP

Why?

What?

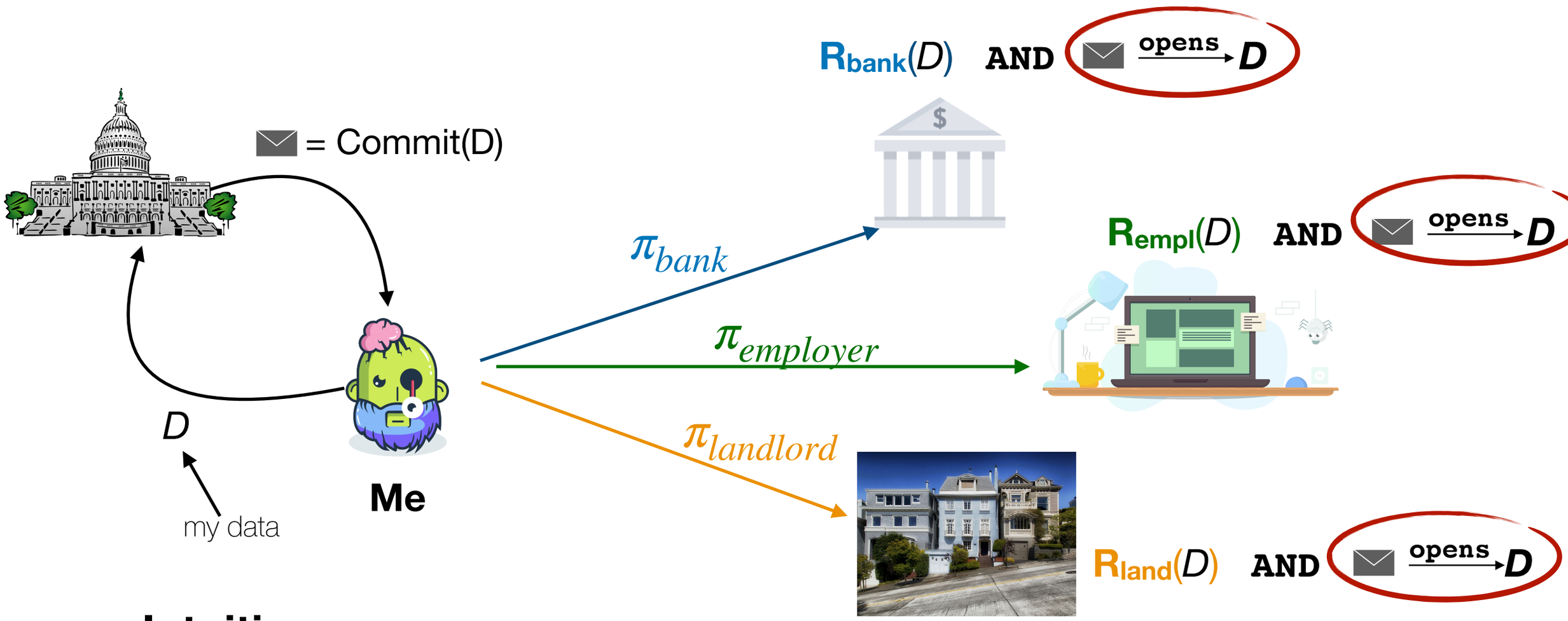
How (much)?

Why Discussing a Standard for CP?

- **Extensive usage**
  - → Typical reasons to standardize (maximize compatibility, etc.)
- **Idiosyncratic reasons**
  - CP requires a *particular type of interoperability*

NEXT: *let's give an example*

Interoperability and CP



Intuition:

Different CPZK operate on the *same* representation (the commitment).

This representation is part of the relation.

Standardizing CP

Why?

What?

How (much)?

Step 0:
One Single Notion for CPZK

An arbitrary relation for CP:

R_{com}(ck, c, u) := R(u) AND "c opens to u w.r.t. ck"

There exist two notions in literature; they treat ck differently*

Notion (A) **(ck output of KG)** [Geppetto]	**Notion (B)** **(ck input of KG)** [**C**FQ19,Lipmaa16,~EG14]
(ck, srs) <- ZKCP.KeyGen(R)	*ck <- Com.Setup()* ⋮ *srs <- ZKCP.KeyGen(ck, R)*

* **NB**: this distinction makes sense for systems with trusted setup.
Recall KG syntax: *srs <- ZK.KeyGen(R)*

Comparing Notions

May be the notion worth standardizing

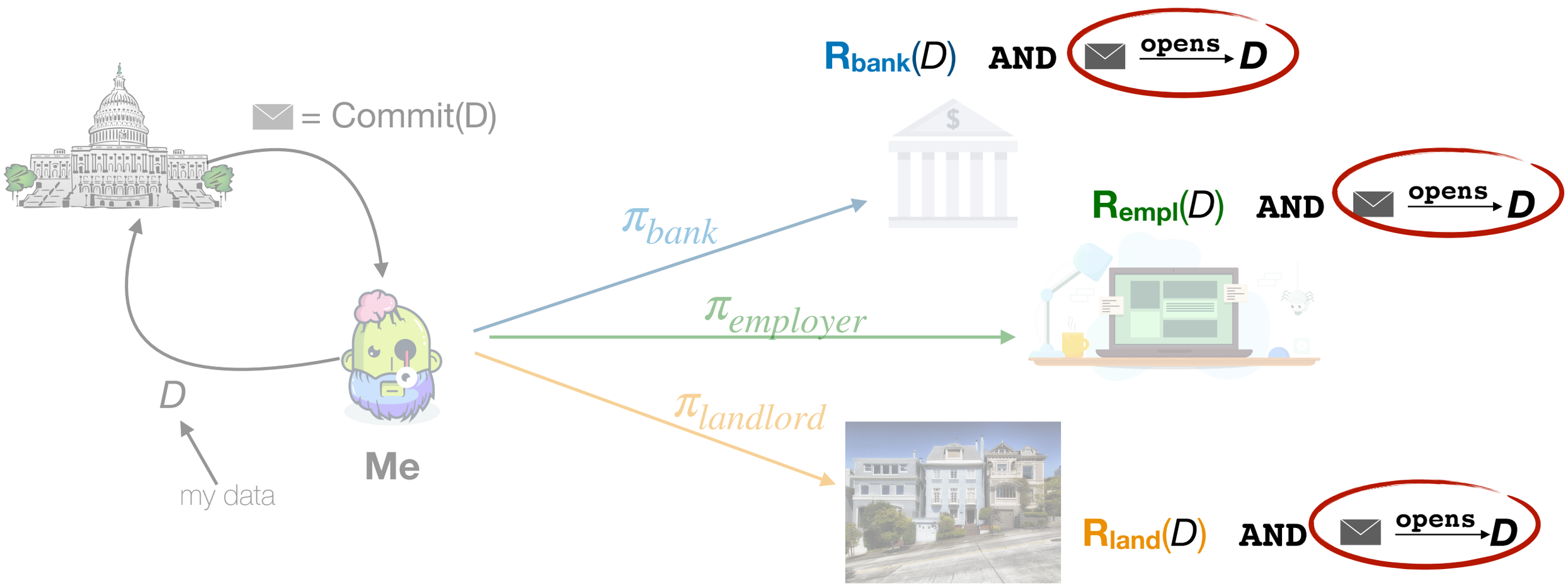
Notion (A) (ck output of KG)	**Notion (B)** (ck input of KG)
(ck, srs) <- ZKCP.KeyGen(R)	srs <- ZKCP.KeyGen(ck, R)
Commitment depends on R and on scheme.	**Decouples commitment, R and scheme.**
Quite specific.	**Enables nice applications for CP** (e.g. commit-ahead-of-time, etc.)

In the remainder of this presentation
I will assume (B) as a CP notion to standardize

What to standardize?

Plausibly**, commitments.** Why?

1. CPZK ~ ZK + Commitment



2. Commitments are the **"interoperability bottleneck"** *and* they are part of the relation. → At least we may need to agree on their syntax.

A Commitment Syntax

GOAL: A syntax for  $\xrightarrow{\textbf{opens}}$ ***D***

As for CP, different notions of "opening" are possible. Let's agree on one.

A possible syntax:

$$\mathsf{VfyCom}(ck, c, D, o) \to b \in \{0,1\}$$

"c opens to data D through opening o w.r.t. ck"

"Verification"-flavored
(opening always carries *D*)

Contrast this with (the more common)

$$\mathsf{Open}(ck, c, o) \to D \in \{0,1\}^*$$

"Reconstruction"-flavored
(need to be able to recompute D from (c,o))

A Definition for CP* [CDQ19]

$R(\ x\ ,\ u\ ,\ \omega\ )$

Public input — Committed witness — "free" witness

Def. *A CP-NIZK for relation R and commitment scheme Com is a NIZK for the relation* $R_{com} := (ck, R)$ s.t.

$$R_{com}(\ x, c, u, o, \omega\ ) := \text{“}\ R(\ x, u, \omega\ ) = 1 \quad \wedge\ \mathrm{VfyCom}(ck, c, u, o) = 1\ \text{”}$$

CP syntax.

KeyGen(ck, R) $\rightarrow$ srs = (ek, vk)

Prove(ek, x, c, u, o, ω) $\rightarrow \pi$

Ver(vk, x, c, π) $\rightarrow$ 0/1

*credits to Dario Fiore for the slide.

Standardizing CP

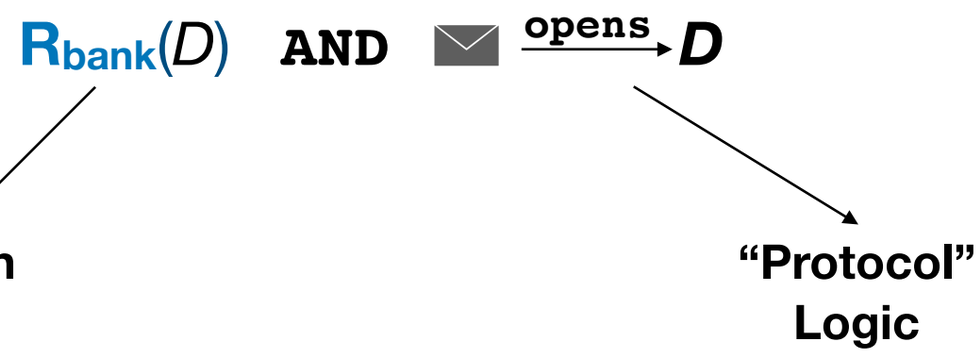
Why?

What?

How (much)?

CP and Relation Representation

A Commit-and-prove relation shows **separation of concerns:**

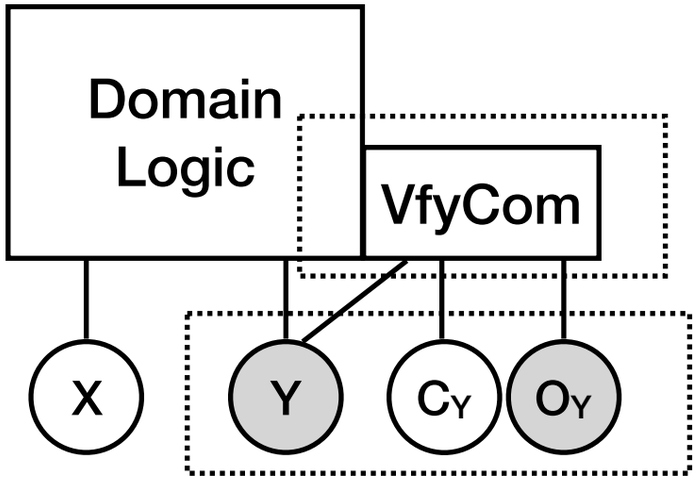


Q: Could CP be standardized at the level of **relation representation?**

CP and Relation Representation (cont.)

```
define_rel(
     name: myComAndProveRel,
     vars:(
        X: Field,
        comY: Commitment(Field) ),
     constraints:
        /* Domain logic R(X,Y) */
)
```

Could compile automatically to R* that:
1. shows opening of Y
2. applies domain logic on domain input values (X,Y)



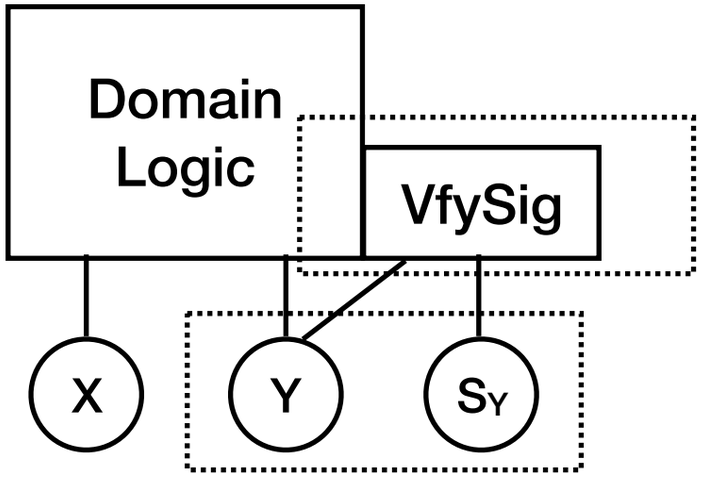
Q: Could we do the same for other cryptographic constructs?

Not only Commit-and-Prove

```
define_rel(
     name: mySigAndProveRel,
     vars:(
        X: Field,
        sigY: Sign(Field) ),
      constraints:
        /* Domain logic R(X,Y) */
)
```

Could compile automatically to R* that:

1. verifies signature on Y
2. applies domain logic
   on domain input values (X,Y)



Final Remarks/Qs

- What **abstraction(s)** for CP?
  - def. should **decouple commitment and relation** (or should it?)
  - Current proposal:
    - modularity
    - enable “nice” properties (commit-ahead-of-time, etc.)
- Standardizing commitments? Which **syntax**?
- Should we **separate “domain” and “protocol” logic**?
- Standardizing applications, implementation, etc…?

Thanks!