Towards an Abstraction for Verifiable Credentials and Zero Knowledge Proofs

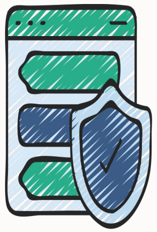
November 2023

Mark Moir
Architect
Oracle Labs
mark.moir@oracle.com

About us

- Full-time at Oracle Labs:
  - Harold Carr (https://www.linkedin.com/in/haroldcarr)
    - Blockchain fault tolerance and scalability, Infiniband transport, distributed systems, logic circuit simulation, …
  - Mark Moir (https://www.linkedin.com/in/markmoir)
    - Blockchain fault tolerance and scalability, synchronization primitives, concurrent algorithms, formal verification, …
- Intern, summer 2023: Christoph Braun (Karlsruhe Institute of Technology)

Using Verifiable Credentials and Zero Knowledge Proofs to balance privacy and accountability



- Collecting too much information creates liability, compliance and privacy issues
- Collecting too little precludes accountability
- Verifiable Credentials and Zero Knowledge Proofs can help to strike an effective balance
- But, many VC projects and standards don't enable such use of ZKPs and/or are tied too tightly to specific cryptography libraries, limiting choice and progress

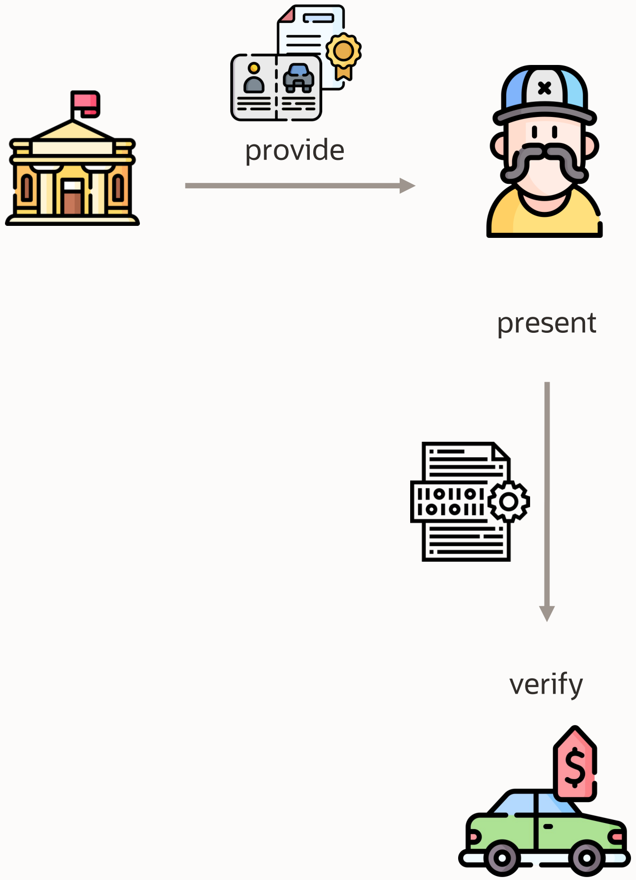
Icon attributions on last slide: 1, 2

Overview of our work

- We are:
  - Learning about these technologies, projects, standards efforts
  - Demonstrating (internally) potential of technologies
  - Developing an abstraction to decouple VCs from underlying cryptography
- So far we have:
  - Defined an initial abstraction, embodied by a Swagger API
  - Implemented it over DockNetwork crypto (https://github.com/docknetwork/crypto)
  - Built a Docker container that serves the API to facilitate experimentation by us and others
  - Developed a Car Share Service demo involving Driver's License and Monthly Subscription credentials, keeps hirer anonymous, enables identification by Authority if needed
- Currently continuing our work and:
  - Revising and generalizing abstraction based on lessons learned so far
  - Sharing our experience and opinions so far
  - Seeking feedback, engagement

Verifiable Credentials: basic roles

Issuer issues credential that includes attributes (e.g., about Holder)



Holder presents "something" to Verifier

Verifier verifies the presented "something"

See last slide for icon attributions

Strawman approach: Use traditional digital signatures

- Issuer signs message that includes attributes
- Holder presents message and signature to Verifier
- Verifier verifies signature

- Privacy implications
  - Holder must reveal entire message to enable verification
  - Holder presents same signature every time, enabling correlation, tracking, etc.
- Regulations, best practices require compliance with Data Minimization principle:
  - Collect only necessary information

Using Zero Knowledge Proofs

- Issuer signs message that includes attributes
- Holder presents ~~message and signature~~ proof of knowledge of Issuer's signature
- Verifier verifies ~~signature~~ proof

- Zero Knowledge Proofs
  - Prove knowledge of something (e.g., Issuer's signature) *without* disclosing it
  - Different proof each time; prevents unwanted correlation
  - Reveal selected attributes, hide others
  - Prove properties about something without disclosing it, e.g.:
    - Predicates such as DOB > 20 years ago
    - Set (non)membership,
    - Many others

Zero Knowledge Proofs

	Zero-knowledge Proofs:	est. ca.
Privacy	*Proof of Knowledge of Signature (selective disclosure)*	2004
Privacy	*Range Proofs (range membership)*	2008
Privacy	*Cryptographic Accumulators (set membership)*	2008
Accountability	*Verifiable Encryption (encrypted disclosure)*	1998

Zero Knowledge Proofs

	Zero-knowledge Proofs:	est. ca.
Privacy	• *Proof of Knowledge of Signature (selective disclosure)*	2004
	• *Range Proofs (range membership)*	2008
	• *Cryptographic Accumulators (set membership)*	2008
Accountability	• *Verifiable Encryption (encrypted disclosure)*	1998
	• ***and composites of those!***	**2016/2019 (in theory!)**

Zero Knowledge Proofs

		Zero-knowledge Proofs:	est. ca.
Privacy		• *Proof of Knowledge of Signature (selective disclosure)*	2004
		• *Range Proofs (range membership)*	2008
		• *Cryptographic Accumulators (set membership)*	2008
Accountability		• *Verifiable Encryption (encrypted disclosure)*	1998
		• ***and composites of those!***	**2016/2019 (in theory!)**

Not as easy as one might think!

Zero Knowledge Proofs

Commit-and-prove technology enables composing different ZKPs without compromising ZK

	Zero-knowledge Proofs:	est. ca.
Privacy	• *Proof of Knowledge of Signature (selective disclosure)*	2004
Privacy	• *Range Proofs (range membership)*	2008
Privacy	• *Cryptographic Accumulators (set membership)*	2008
Accountability	• *Verifiable Encryption (encrypted disclosure)*	1998
	• ***and composites of those!***	**2016/2019 (in theory!)**

Not as easy as one might think!

LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs

Matteo Campanelli[1], Dario Fiore[1], and Anaïs Querol[1,2]

[1] IMDEA Software Institute
[2] Universidad Politécnica de Madrid

matteo.campanelli@imdea.org
dario.fiore@imdea.org
anais.querol@imdea.org

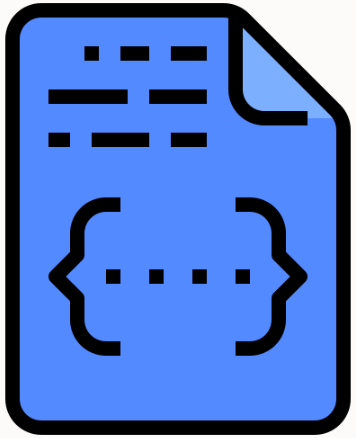
Full Version

Abstract. We study the problem of building non-interactive proof systems modularly by linking small specialized "gadget" SNARKs in a lightweight manner. Our motivation is both theoretical and practical. On the theoretical side, modular SNARK designs would be flexible and reusable. In practice, specialized SNARKs have the potential to be more efficient than general-purpose schemes, on which most existing works have focused. If a computation naturally presents different "components" (e.g. one arithmetic circuit and one boolean circuit), a general-purpose scheme would homogenize them to a single representation with a subsequent cost in performance. Through a modular approach one could instead exploit the nuances of a computation and choose the best gadget for each component.
Our contribution is LegoSNARK, a "toolbox" (or framework) for commit-and-prove zkSNARKs (CP-SNARKs) that includes:

IACR Cryptol. ePrint Arch. 2019

Practical implementation based on Commit-and-Prove, 2021-2023

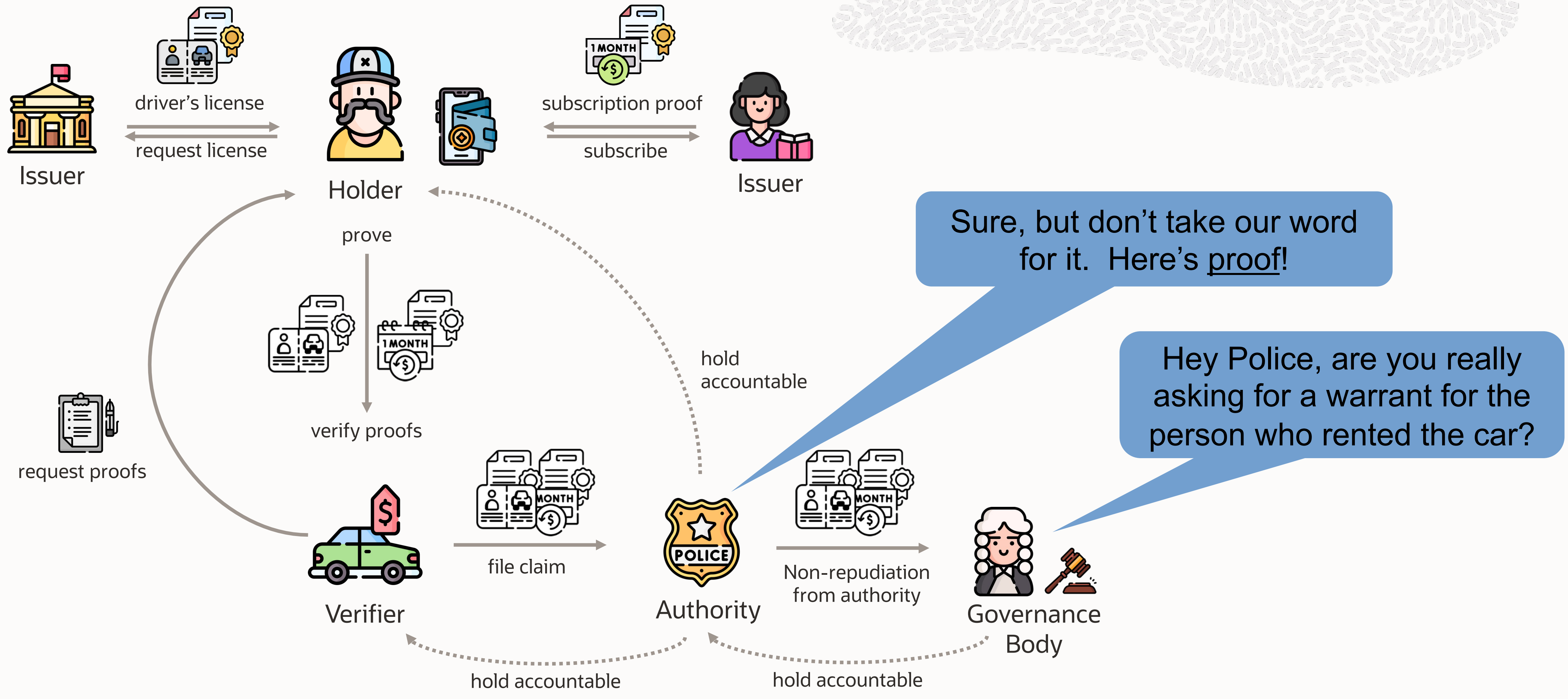
- Open source DockNetwork crypto library uses commit-and-prove to implement “proof system” that enables composing any/all of:
  - Efficient BBS+ signatures, with Selective Disclosure
  - Range proofs (e.g., DOB > 20 years ago)
  - Cryptographic accumulators (supporting privacy-preserving revocation, for example)
  - Other ZKPs (any zk-SNARK that can be expressed in R1CS, e.g., created using Circom)
  - Verifiable encryption
  - More (secret sharing, distributed key generation, …)



DockNetwork crypto library

See last slide for icon attributions (3)

Example use case - Accountability in privacy-preserving authentication for services



Icons by Freepik from flaticon.com

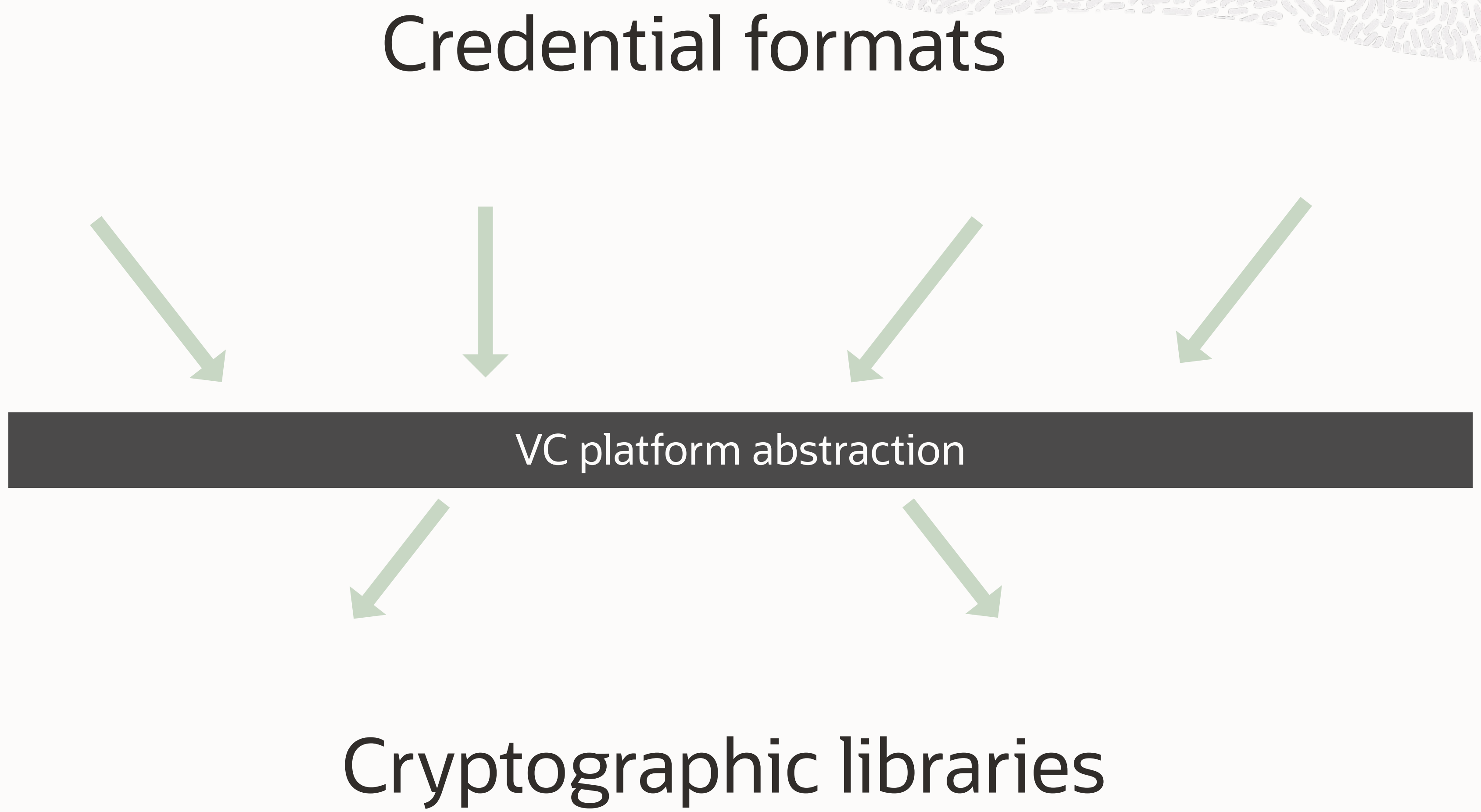
Implementing our use case requires

- Mapping credential contents to “messages” to be signed
  - Different for different attributes (e.g., special “reversible encoding” for encryptable attributes)
- Targeting DockNetwork library
- Defining use case requirements via library calls to create a “proof system” from “statements” and “meta-statements”
- Providing relevant “witnesses” to library calls in order to create proofs that can be verified

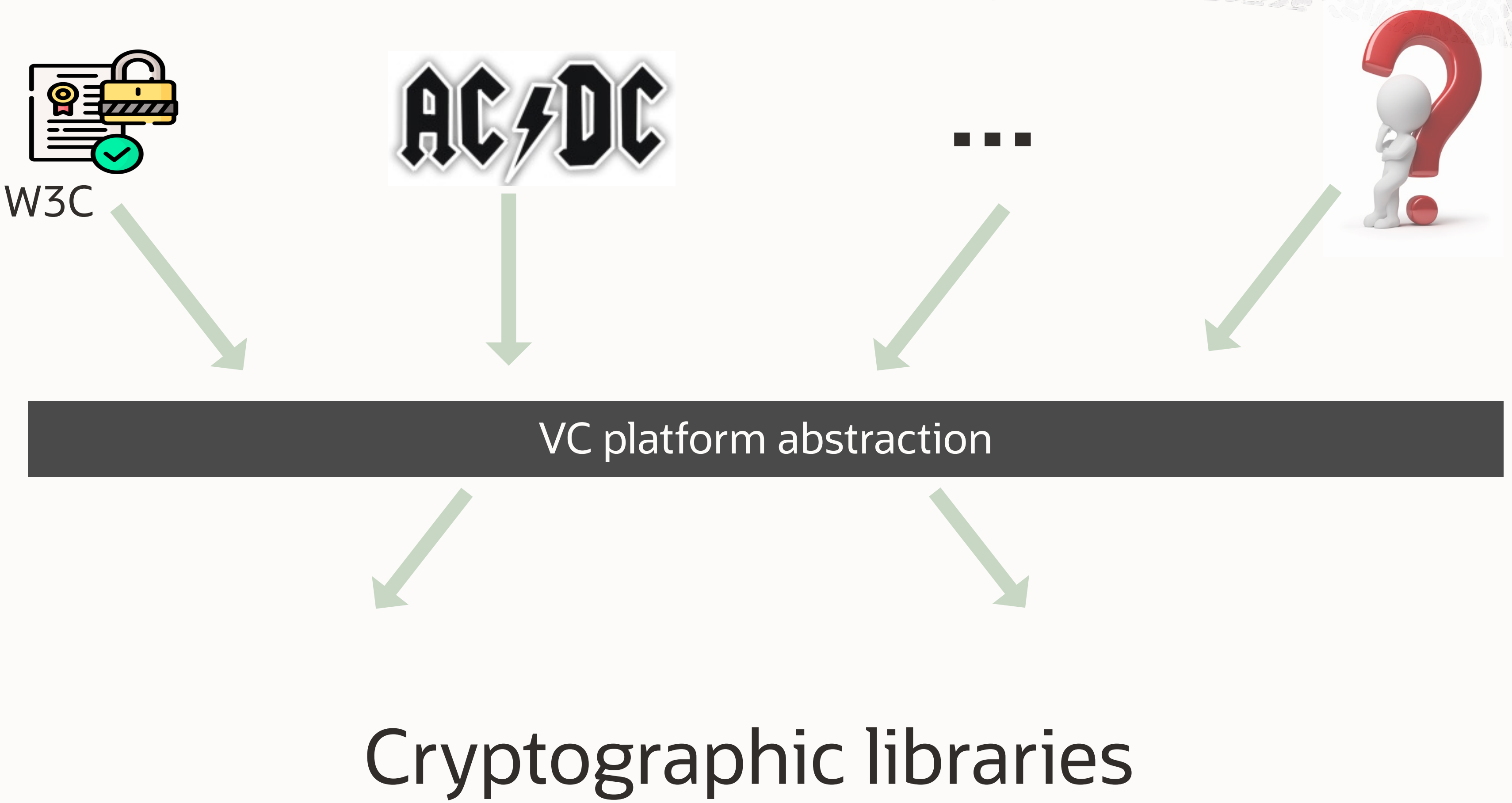
Motivation for an abstraction

- How can people (not just developers with expertise to use specialized cryptography libraries) express/understand/audit requirements?
- Make it easier to implement and understand a specific use case, without requiring low-level knowledge and expertise
- Avoid committing to a specific credential format
- This motivated us to explore an abstraction to separate:
  - Credential format
  - Expression of requirements and functionality
  - Low-level implementation details

Our abstraction

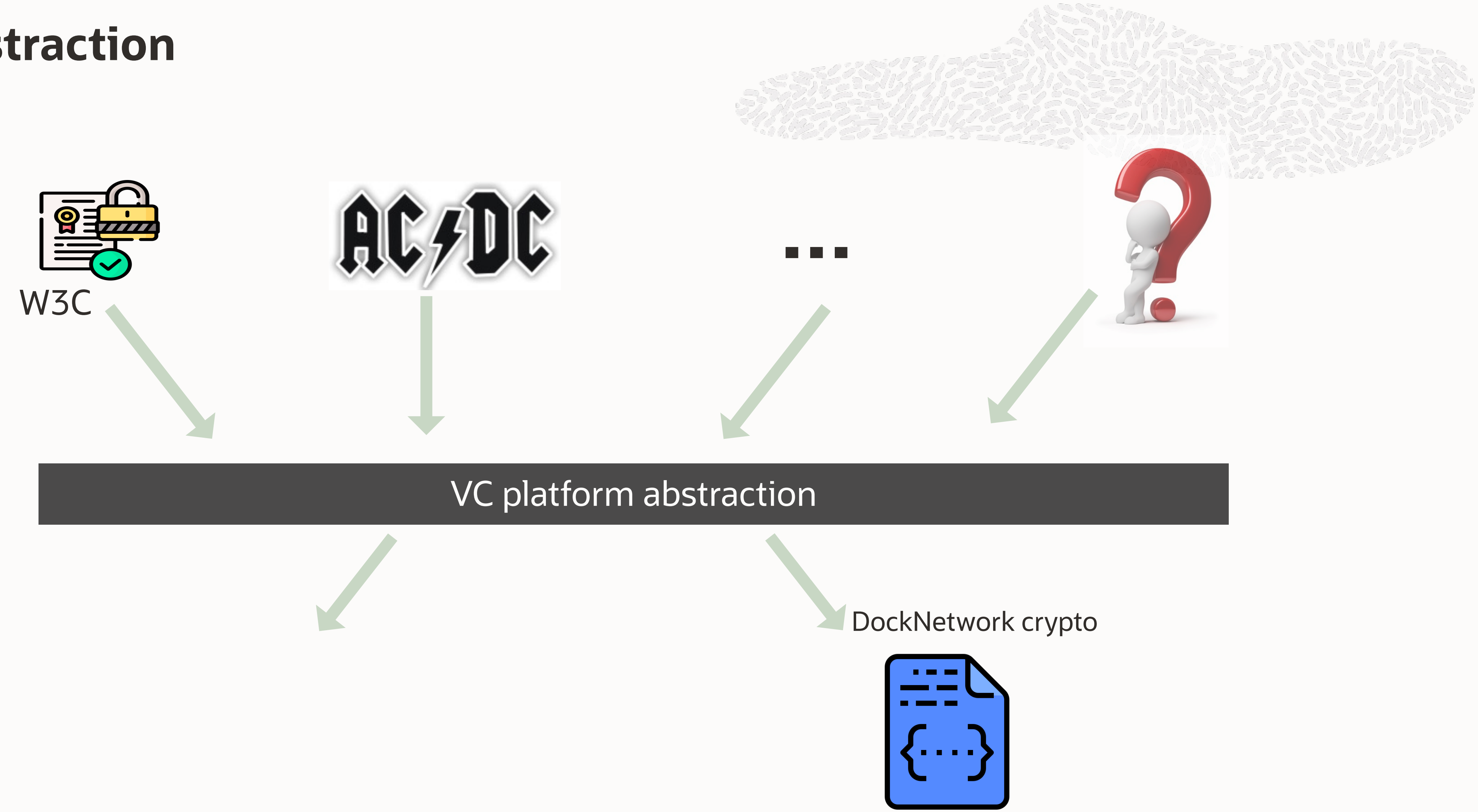


Our abstraction



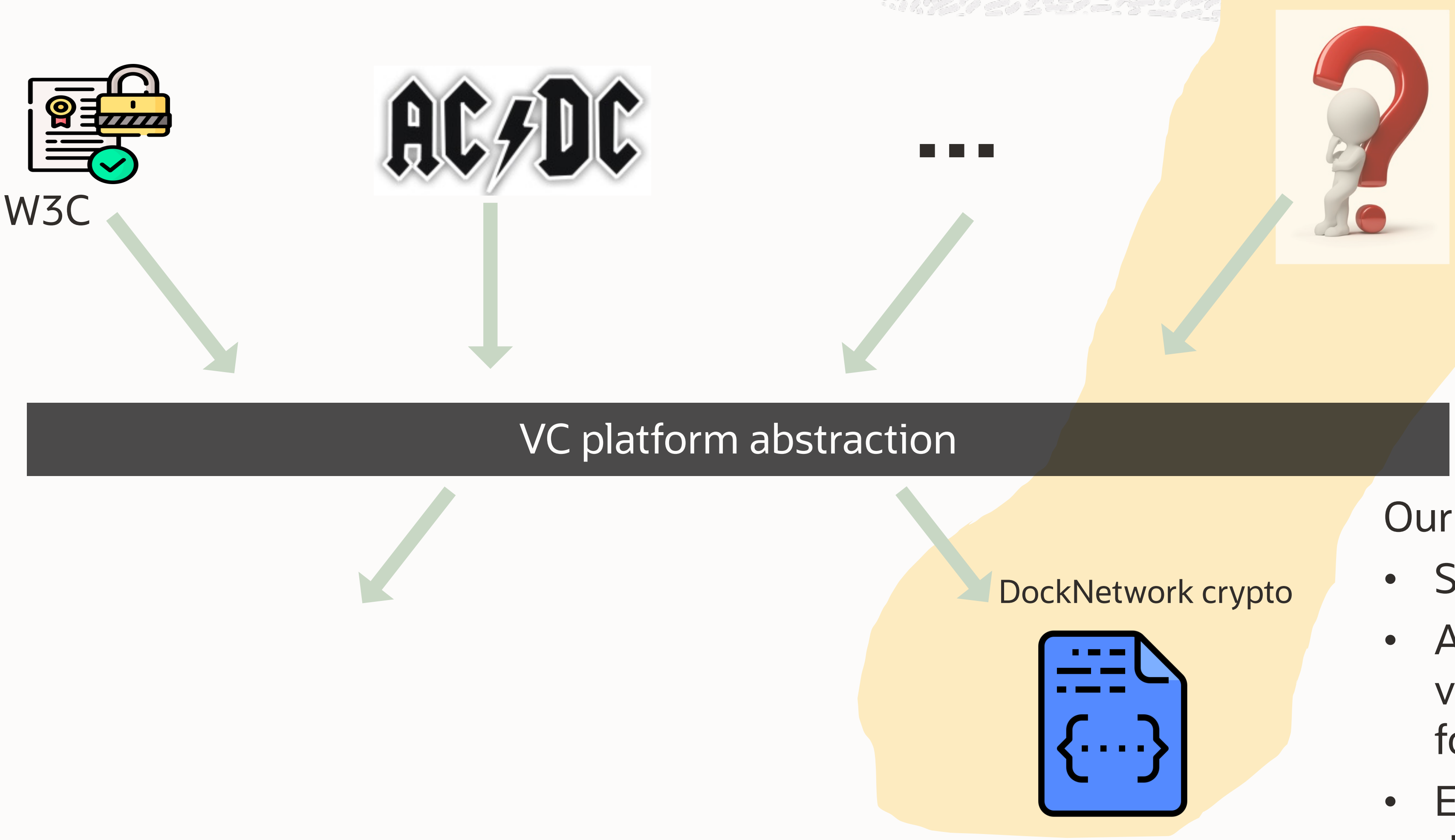
See last slide for icon attributions (4)

Our abstraction



See last slide for icon attributions (3,4)

Our abstraction

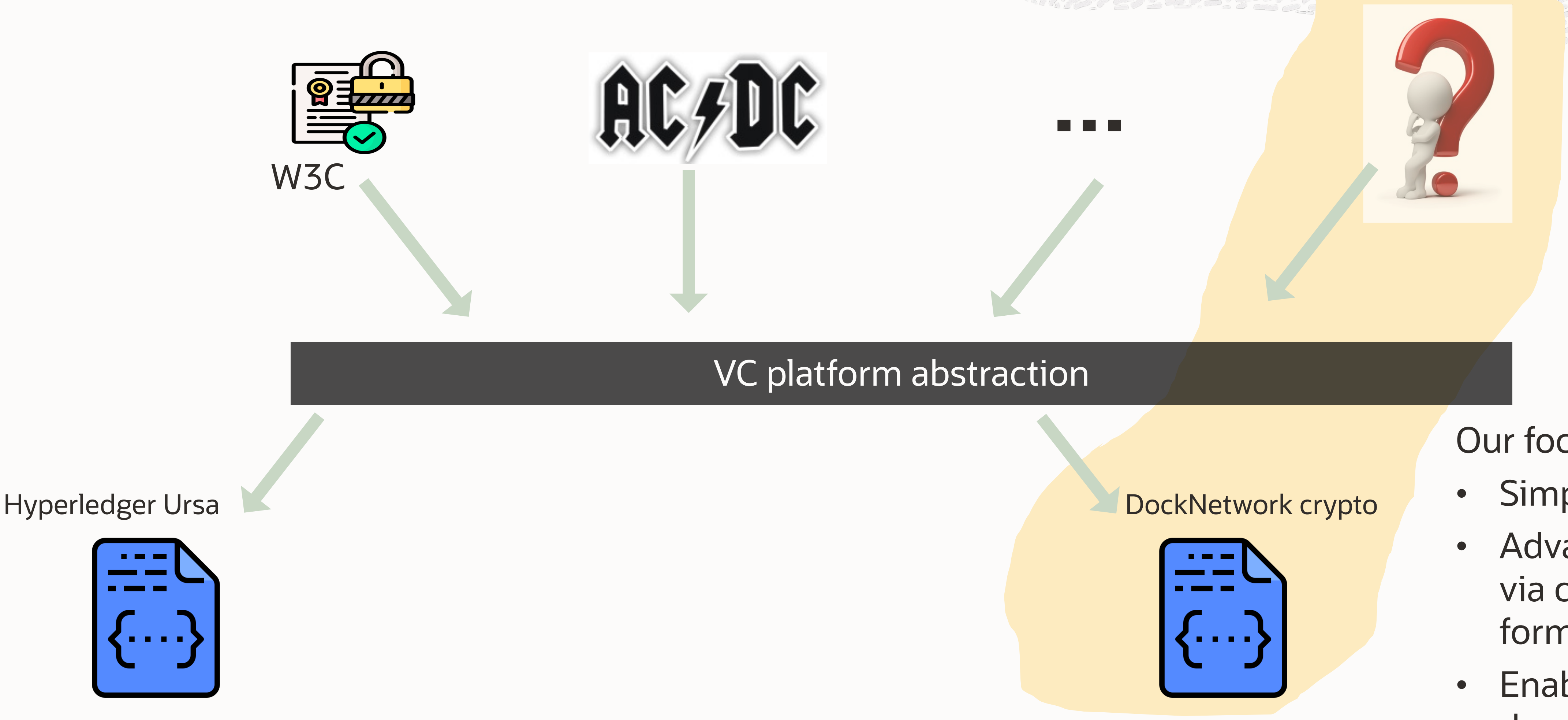


Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction

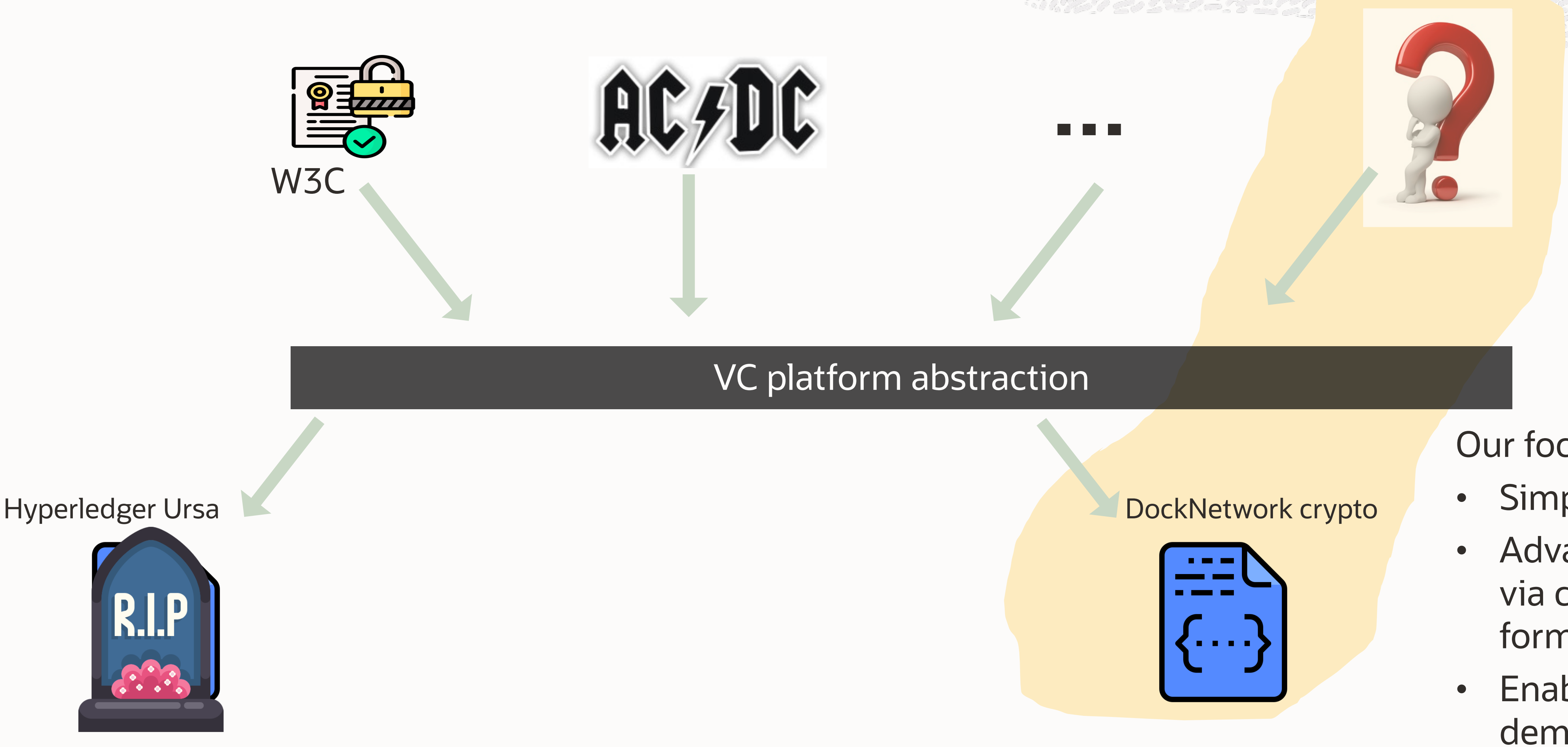


Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction

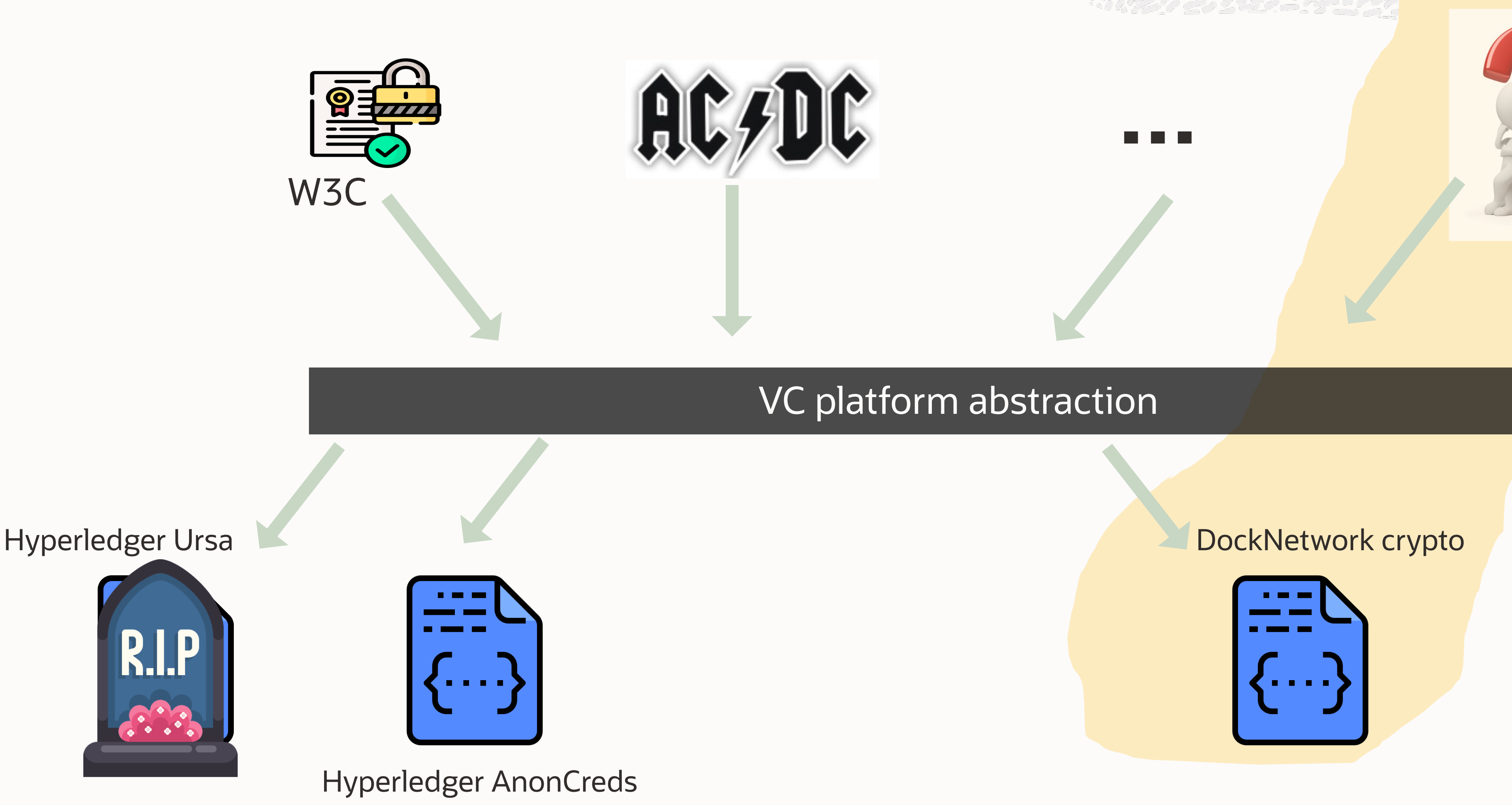


Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction

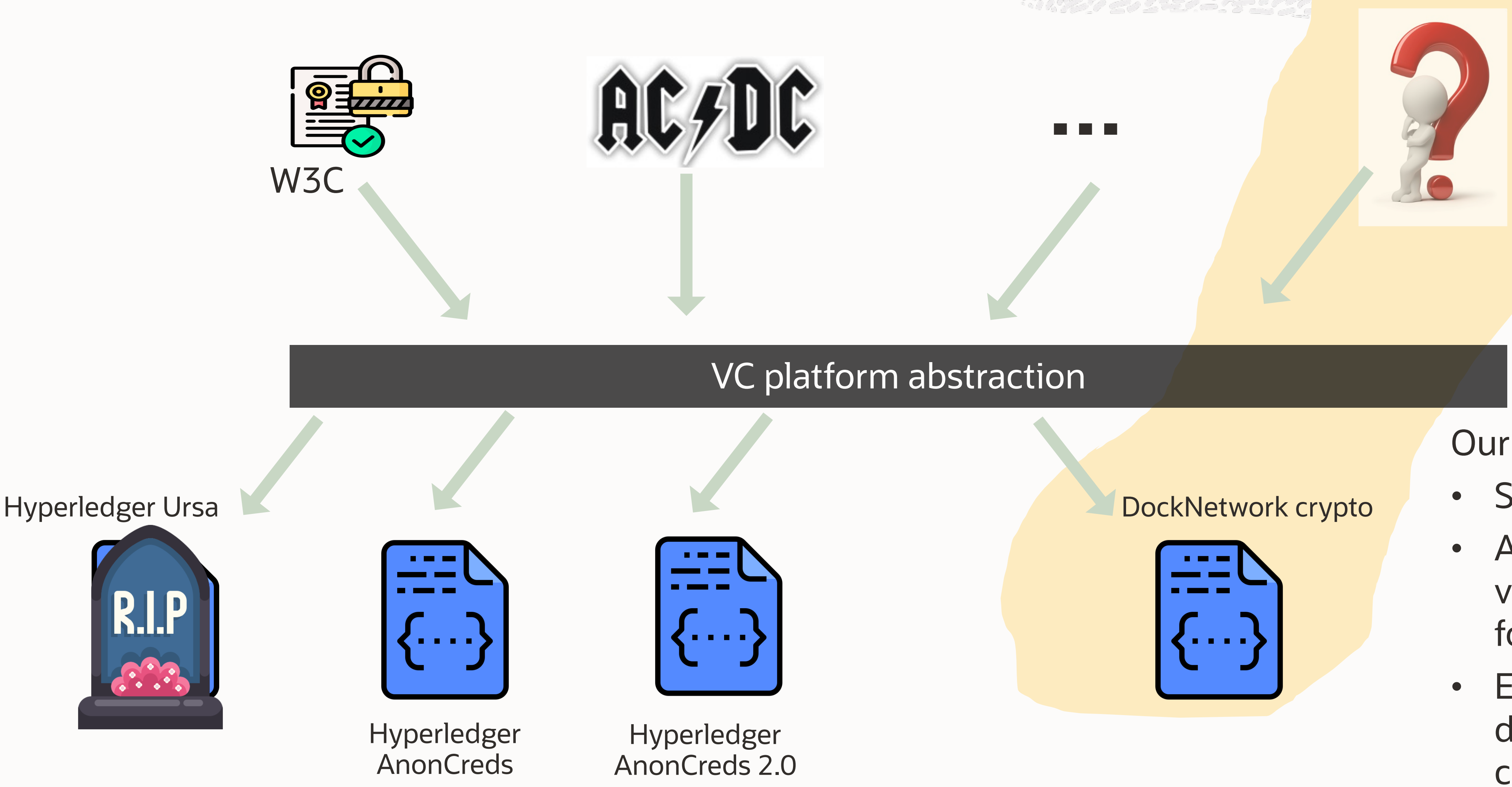


Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction

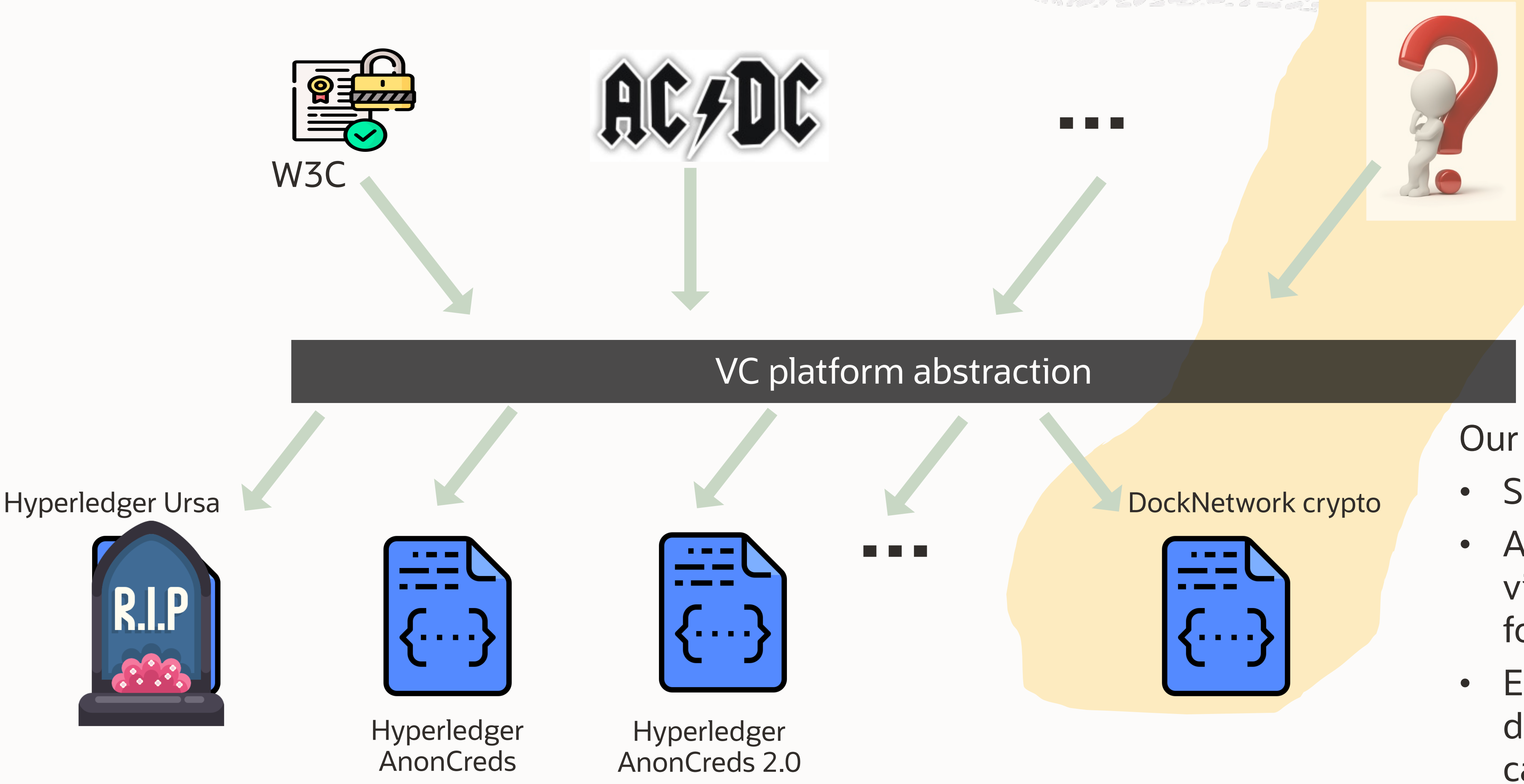


Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction



Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Not all libraries are equal, but…

- Common functionality can be abstracted, enabling use case descriptions to be reused
- Abstraction enables unsupported features to be ignored with warning (“Loose” mode) when creating a presentation proof, or fail with error (“Strict” mode)
- Useful for testing during development, decoupling use cases from specific underlying libraries
- <u>Note</u>: it is **not** a goal to enable interoperability between credentials signed using different underlying cryptography libraries

Caveat: Work in progress

- We are sharing an update on work in progress, seeking feedback and engagement, and hoping our efforts may be useful beyond helping internal product groups
- Even preparing this presentation has resulted in minor changes to our abstraction, some not yet implemented
- We have not yet implemented/used all features provided by DockNetwork crypto; doing so is not a goal in its own right

Simple, artificial credential format

- We think of a “Credential Format Designer” role. CFD decides on format, rules, policy, etc. for what can be in a credential. Could be W3C, AnonCreds, etc.
- Independent of our abstraction. CFD defines how to map their credential format to the abstraction
- To enable experimentation and demonstration, we played this role by defining a simple credential format
- ALMOST JWT, but requires metadata that is part of credential signed

Example driver's license credential in our simple format

```
{ "metaDataForSimpleFormat" : {
      "purpose" : "DriverLicense",
      "version" : "1.0"
  },
  "sdAttrsForSimpleFormat": {
      "ssn":                          "123-12-1234",
      "eyeColor":                     "Brown",
      "daysBornAfterJan_1_1900":  37852,
      "height":                       180,
      "idForRev":                     "abcdef0123456789abcdef0123456789"
  }
}
```

"Credential Format Designer" requires a metadata attribute represented as a JSON object, and a flat list of values (string or int).

The abstraction in more detail (1/2)

- Abstraction is agnostic to credential format (e.g., W3C, AnonCreds, …)
- Various “opaque types” that user receives from interface and sends back (perhaps indirectly), with examples of included information when implemented over DockNetwork crypto library:
    - `SignerPublicSetupData`
        - Public key, signature parameters
    - `SignerSecretKey`
        - Secret key
    - `DataForVerifier`
        - Proof, values of revealed attributes
    - `AuthorityPublicSetupData`
        - Chunk size, commitment generators, encryption generators, encryption key, Groth16ProvingKey, …
    - `AuthoritySecretKey`
        - Groth16SecretKey
    - `AuthorityDecryptionKey`
        - Groth16DecryptionKey
- Note: above is “aspirational”: current implementation does not so clearly separate abstraction from DockNetwork crypto types yet

The abstraction in more detail (2/2)

- Data formats defined by abstraction
  - **Data values** (`DVText` or `DVNat`)
    - Represent values in credential in some defined order
  - **Field Dispositions** (`FDText`, `FDEncryptableText` or `FDNat`)
    - Describe attributes in same order as Values
    - `FDncryptableText` enables special encoding required for decryption
  - **ProofRequest**
    - Defines what disclosures and properties Holder and Verifier agree on. AKA Presentation Request, etc.
  - **SharedParams**
    - `ProofRequest` refers to parameter values symbolically, `SharedParams` provides values
    - Makes `ProofRequest` more readable, reusable with different parameters
  - **DecryptRequest**
    - Identifies credential and attribute to be decrypted, contains `AuthoritySecretKey` and `AuthorityDecryptionKey`

Values for example driver license credential


```
{"values":
  [ {"contents": "CredentialMetadata (fromList [(\"purpose\",DVText
\"DriverLicense\"),(\"version\",DVText \"1.0\")])",  "tag": "DVText"}
  , {"contents": 37852,                              "tag": "DVInt"}
  , {"contents": "Brown",                            "tag": "DVText"}
  , {"contents": 180,                                "tag": "DVInt"}
  , {"contents": "abcdef0123456789abcdef0123456789", "tag": "DVText"}
  , {"contents": "123-12-1234",                      "tag": "DVText"}
  ]
}
```

Values for example driver license credential

```
{"values":
  [ {"contents": "CredentialMetadata (fromList [(\"purpose\",DVText
\"DriverLicense\"),(\"version\",DVText \"1.0\")])",  "tag": "DVText"}
  , {"contents": 37852,                                 "tag": "DVInt"}
  , {"contents": "Brown",                               "tag": "DVText"}
  , {"contents": 180,                                   "tag": "DVInt"}
  , {"contents": "abcdef0123456789abcdef0123456789", "tag": “DVText"}
  , {"contents": "123-12-1234",                         "tag": "DVText"}
  ]
}
```

Note: The simple credential format we defined for experimentation requires metadata, encoded into a `DVText` as the first value, i.e., the attribute with index 0 in the list. The structure, contents, and encoding of this value are determined by the "Credential Format Definer", independent of the abstraction.

Field dispositions for example driver license credential

```
{"fieldDispositions":
  [ "FDText"
  , "FDNat"
  , "FDText"
  , "FDNat"
  , "FDText"
  , "FDEncryptableText"
  ]
}
```

Field dispositions for example driver license credential

```
{"fieldDispositions":
  [ "FDText"
  , "FDNat"
  , "FDText"
  , "FDNat"
  , "FDText"
  , "FDEncryptableText"
  ]
}
```

Note: `FDEncryptableText` indicates to Issuer to sign this attribute (Social Security Number) for verifiable encryption.

Proof Request for example driver license and subscription use case (1/2)

```
{"driverLicense”:
  { "issuerPK"      : "dlIssuerPKList"
  , "sigParams"     : "dlSigParams"
  , "disclosed"     : [0]
  , "inAccum"       : [[4, "dlCurrentAccum"]]
  , "notInAccum"    : []
  , "inRange"       : [[1, ["minBDdays", "maxBDdays", "provingKey"]]]
  , "encryptedFor" : [[5, "commonAuthorityPK"]]
  , "equalTo"       : [[5, ["subscriptionCred", 2]]]
  }
, …
```

Proof Request for example driver license and subscription use case (1/2)

```
{"driverLicense”:
  { "issuerPK"      : "dlIssuerPKList"
  , "sigParams"     : "dlSigParams"
  , "disclosed"     : [0]
  , "inAccum"       : [[4, "dlCurrentAccum"]]
  , "notInAccum"    : []
  , "inRange"       : [[1, ["minBDdays", "maxBDdays", "provingKey"]]]
  , "encryptedFor"  : [[5, "commonAuthorityPK"]]
  , "equalTo"       : [[5, ["subscriptionCred", 2]]]
  }
, …
```

Note:

- These are indices into the list of Values; metadata is 0, Social Security Number is 5.
- Our abstraction does not (and should not) impose a requirement on credential formats to associate names with attribute values; those who want to do so can easily translate via a thin layer above the abstraction.

Proof Request for example driver license and subscription use case (2/2)

```
...
,"subscriptionCred":
  { "issuerPK"      : "subIssuerPKList"
  , "sigParams"     : "subSigParams"
  , "disclosed"     : [0, 4]
  , "inAccum"       : [[1, "subCurrentAccum"]]
  , "notInAccum"    : []
  , "inRange"       : [[3, ["minValiddays", "maxValiddays", "provingKey"]]]
  , "encryptedFor" : [[2, "commonAuthorityPK"]]
  , "equalTo"       : []
  }
}
```

Shared Params for example driver license and subscription use case

```
{ "dlIssuerPK"       :  "[150,80,83,...,126,153]"
, "dlSigParams"      : "{\"g1\":[129,172,243,...,5,224]]}"
, "authorityPubData":
       "{\"chunkBitSize\":8,\"chunkedCommGens\":\"{\\\"G\\\":[147,54,…"
, "maxBDdays"        :   999999999999
, "minBDdays"        :   37696
, …
}
```

DecryptRequests for example driver license and subscription use case


```
[ { "credLabel": "subscriptionCred"
  , "attrIndex": 2
  , "sk"        : "[85,218,36,215,5,193,110,77,65,9,220,70,43,64,147,39,62,…]"}
  , "dk"        : "{\"V_0\":[130,155,40,218,133,4,173,154,175, …}"
  }
]
```


Some advantages abstraction could achieve

- Separate (expression and understanding of) use case requirements from technical details of underlying cryptography libraries
- Enable reports/summaries in various levels of detail
  - User requests to disclose a verifiably encrypted attribute: likely a mistake. But conceivably intentional, so we allow it, but could generate a warning in a summary of requirements
  - Similarly for revealing accumulator members (thus leaking correlatable info)
- Facilitate switching underlying cryptography libraries for various reasons:
  - Health of project, easier performance comparison, better performance, more rigorous security proofs, stronger security guarantees (e.g., post-quantum), more favourable license, …
- Could enable formal proofs of privacy properties about use cases, based on assumptions about guarantees made by underlying cryptography
  - Such assumptions could be formally proved for specific underlying cryptography libraries, or assumed to be true based on pen and paper proofs
- …

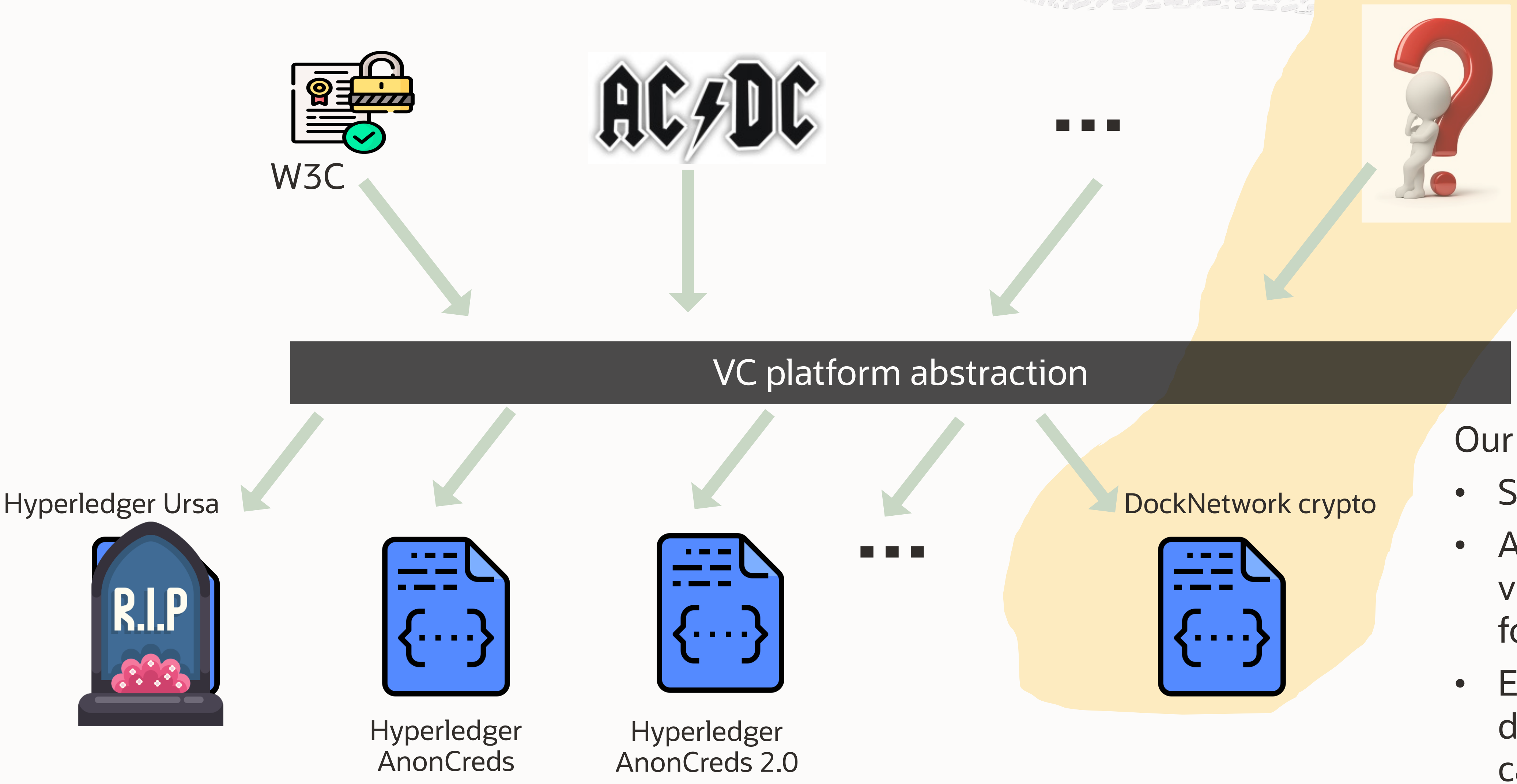
Notes

- No attribute names, can be done at higher level
- But credential names are useful and do not impose a requirement on credential format
- No “opinion” on what features should be used, etc.
- We haven’t done anything with signing blinded attributes yet

Partial list of things we think should live above the abstraction

- VC format (JSON, JSON-LD, …)
- Attribute names (if any)
- Negotiation of presentation requirements
- Communication protocols
- VC contents
  - Example: questions such as whether AnonCreds requires a link secret could/should be be separate from underlying cryptography
- Rules, policies, roles
  - Example: AnonCreds v2 assumes Authority = Issuer
    - We don't think this should be assumed (Authority could be Police, some other government entity, some legal entity, etc., determined by use case)
    - Regardless, such decisions should be separate from underlying cryptography
- …

Our abstraction

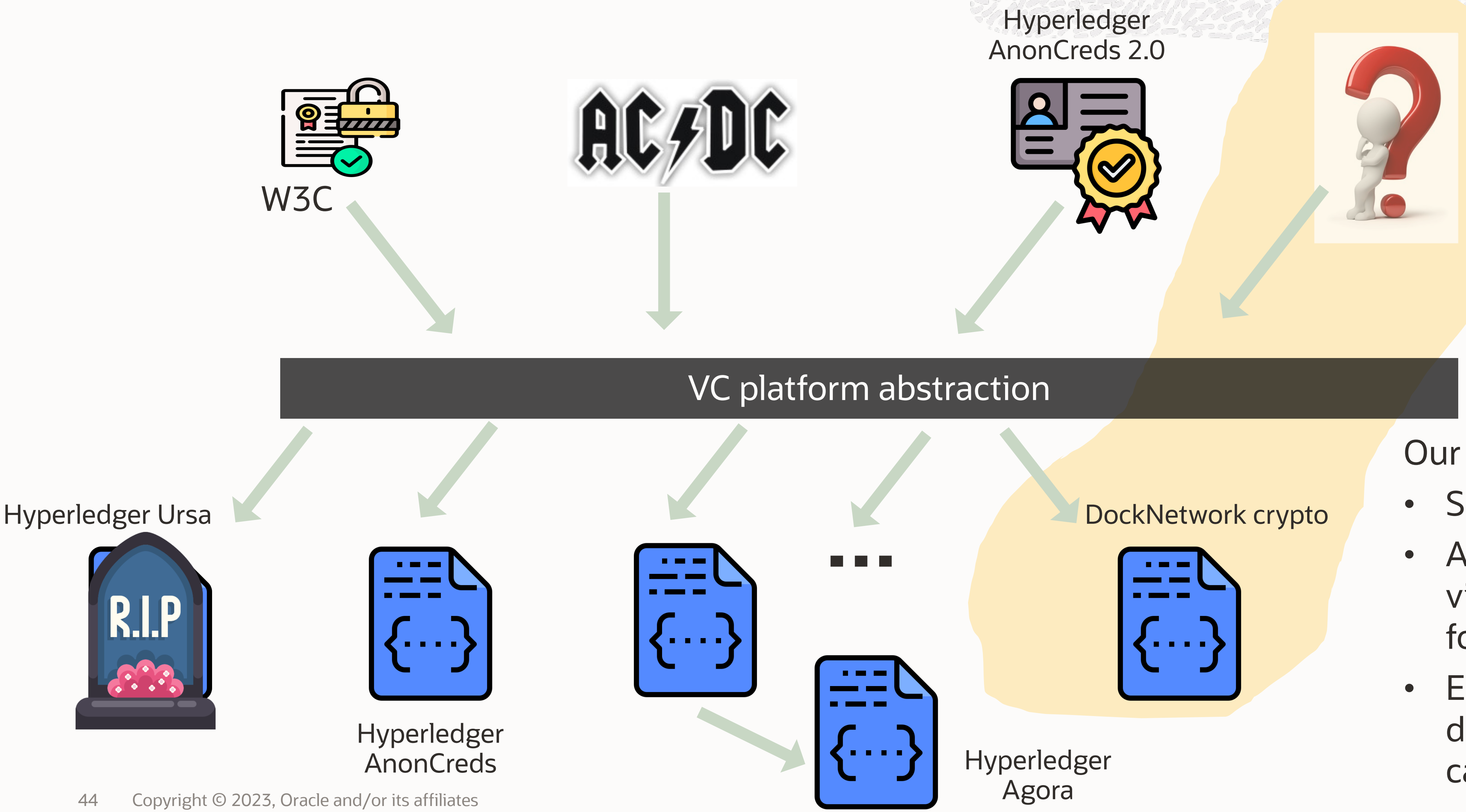


Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction and AnonCreds 2.0



Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4,5)

Summary

- We are:
  - Learning about technologies, projects, and standards efforts around Verifiable Credentials and Zero Knowledge Proofs
  - Demonstrating (internally) potential of these technologies
  - Developing an abstraction to decouple VCs from underlying cryptography
- This has led us to some opinions and ideas that we think can be beneficial beyond our organisation
- Therefore we are sharing our experience and opinions so far, seeking feedback, engagement

Thank you

Mark Moir, Architect, Oracle Labs

mark.moir@oracle.com

www.linkedin.com/in/markmoir

Icon attributions

All icons by Freepik from flaticon.com, except:

1. Private message icons created by juicy_fish – Flaticon
2. Arrest icons created by shmai – Flaticon
3. Icon by ultimatearm
4. By Denmokin - Own work, Public Domain
5. Professional icons created by Uniconlabs - Flaticon