Visual Data Management System

github.com/IntelLabs/vdms
aperturedata.io

Vishakha Gupta, ApertureData Inc.

Luis Remis, Intel Labs

Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

Metadata -> Relational Database, Graph Database

Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

Metadata -> Relational Database, Graph Database

Service for storing the images -> HTTP Server, PACS

Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

Metadata -> Relational Database, Graph Database

Service for storing the images -> HTTP Server, PACS

Library for preprocessing -> OpenCV

Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

Metadata -> Relational Database, Graph Database

Service for storing the images -> HTTP Server, PACS

Library for preprocessing -> OpenCV

Typical Machine Learning (ML) Workload

Visual Workload: Metadata + Visual Data

Metadata -> Relational Database, Graph Database

Service for storing the images -> HTTP Server, PACS

Library for preprocessing -> OpenCV

Very tailored set of scripts

ML requires a new type of data management

Unified

Easy to use

AI ready

Efficient and evolving

Next Generation of Data Challenges

Primarily Visual
e.g. images, feature vectors, videos

Large scale
Rich with information
Individually large
Frequently time sensitive

Next Generation of Data Challenges

Primarily Visual
e.g. images, feature vectors, videos

Large scale
Rich with information
Individually large
Frequently time sensitive

Machine learning or data science usages

Expect easy integration
Repetitive data preprocessing
Ever evolving
Frequent network transfers

Next Generation of Data Challenges

Primarily Visual
e.g. images, feature vectors, videos

Large scale
Rich with information
Individually large
Frequently time sensitive

VDMS is designed to address these

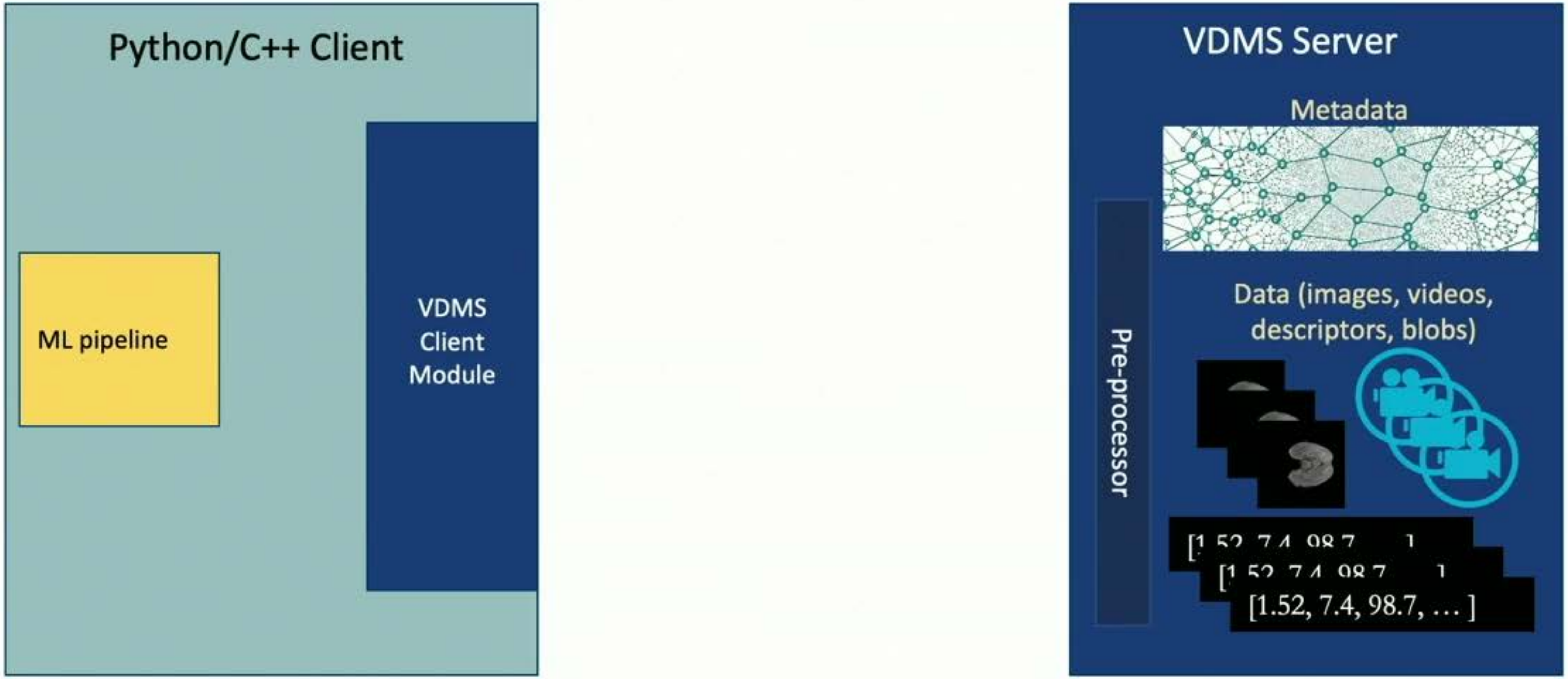
Machine learning or data science usages

Expect easy integration
Repetitive data preprocessing
Ever evolving
Frequent network transfers

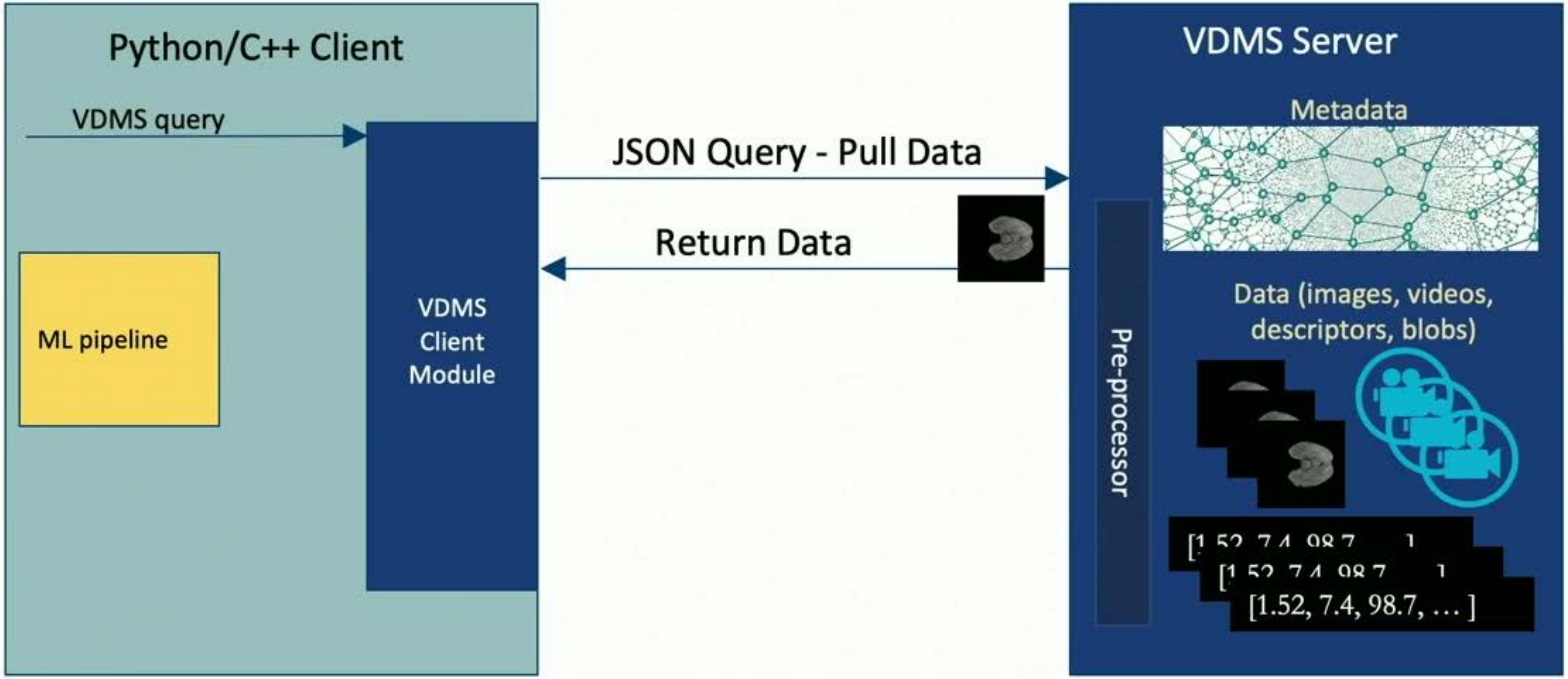
VDMS Capabilities

- Efficient completion of complex metadata queries
  - Metadata stored in (persistent) memory
  - Using our in-house Graph Database (now ACID compliant)
- Efficient visual data retrieval
  - Images can be stored in image format designed for analytics
    - Threshold, crop, resize, or basic augmentation on images on the server side.
  - Visual Descriptors can be stored, and similarity search (KNN) performed on the fly.
    - Using different mechanism to index and compute distances
  - Video can be stored/retrieved.
- Straightforward client API to enable both metadata and data retrieval
  - Queries submitted as JSON (using Python or C++)

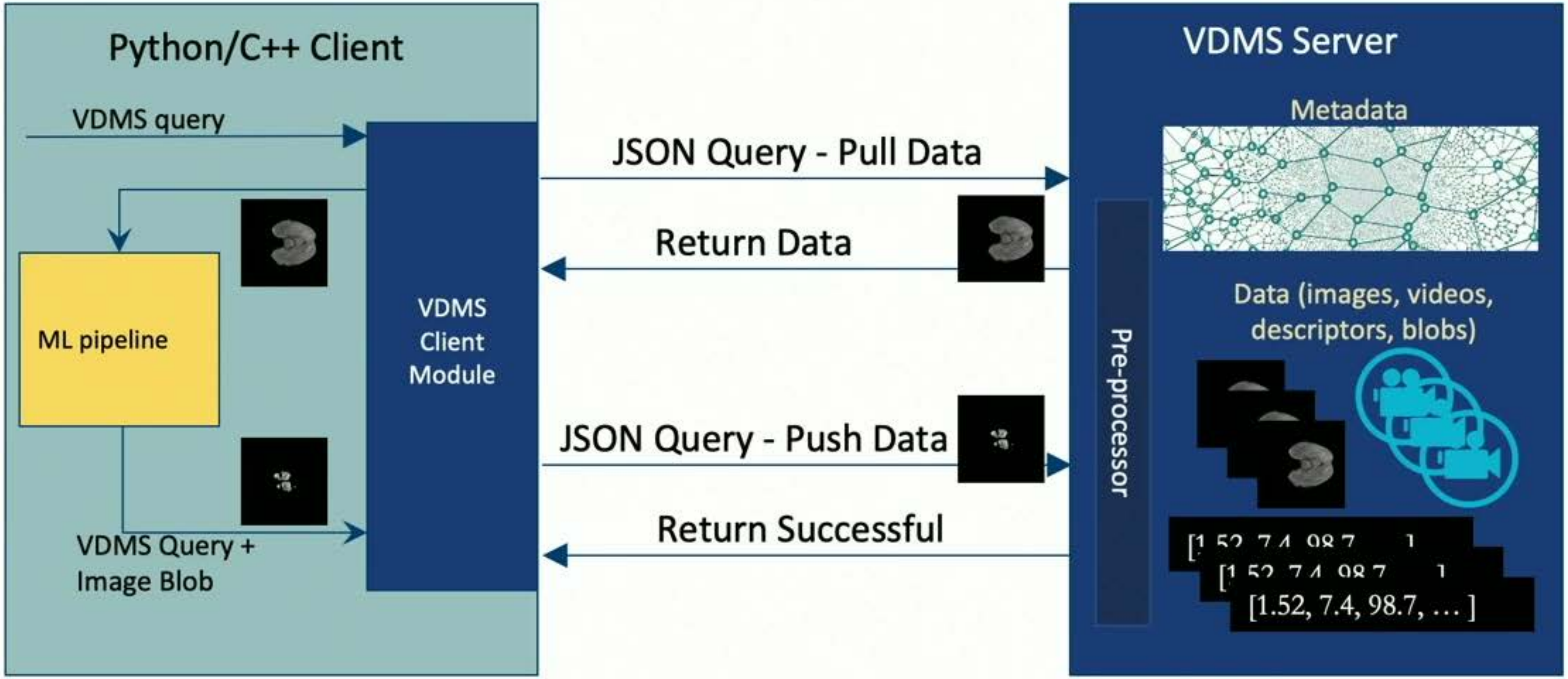
VDMS Pipeline Example



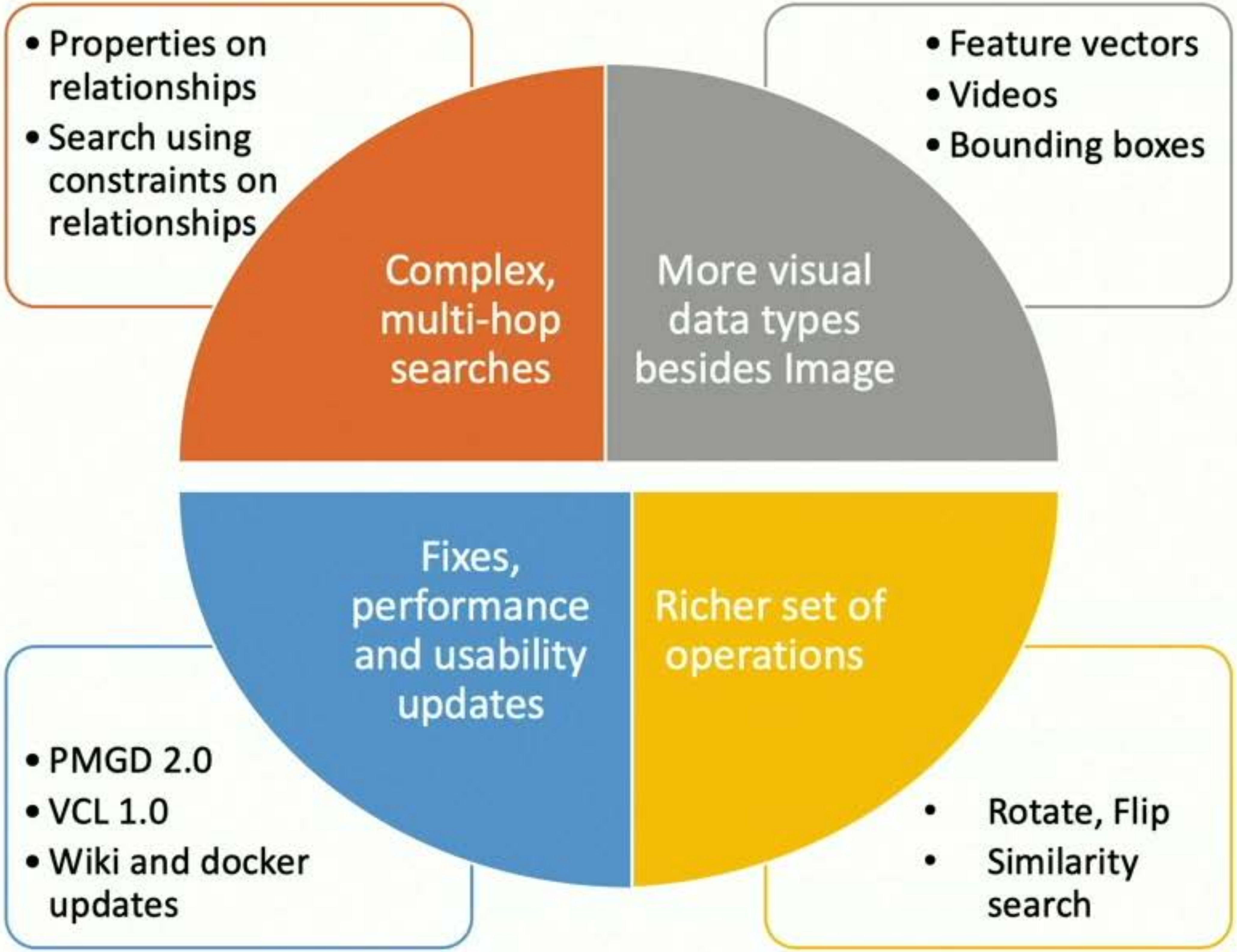
VDMS Pipeline Example



VDMS Pipeline Example



VDMS Release 2.0



Complex, multi-hop searches

- Properties on relationships
- Search using constraints on relationships

More visual data types besides Image

- Feature vectors
- Videos
- Bounding boxes

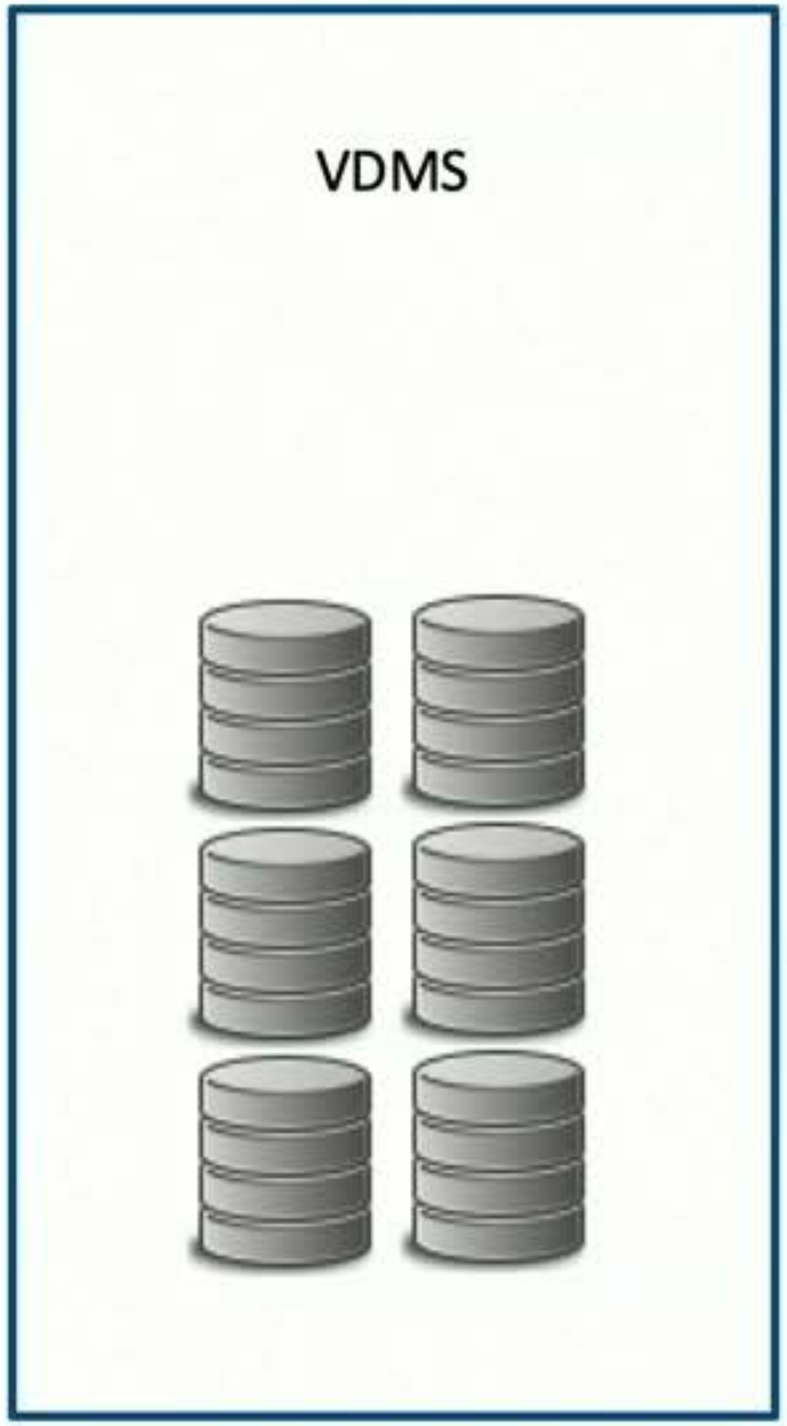
Fixes, performance and usability updates

- PMGD 2.0
- VCL 1.0
- Wiki and docker updates

Richer set of operations

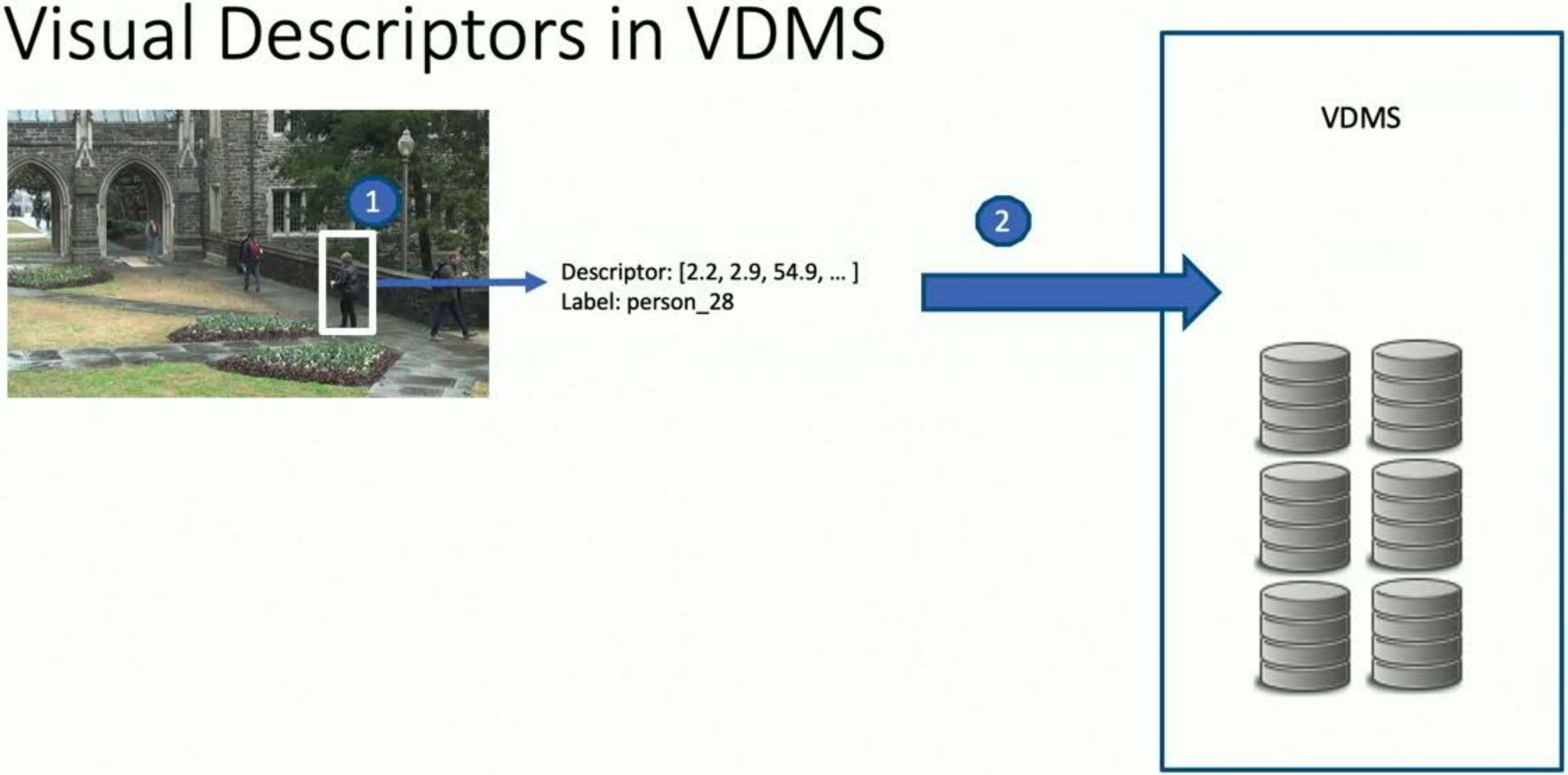
- Rotate, Flip
- Similarity search

Visual Descriptors in VDMS



Novel solution for persistent Feature Vector storage, indexing, and search

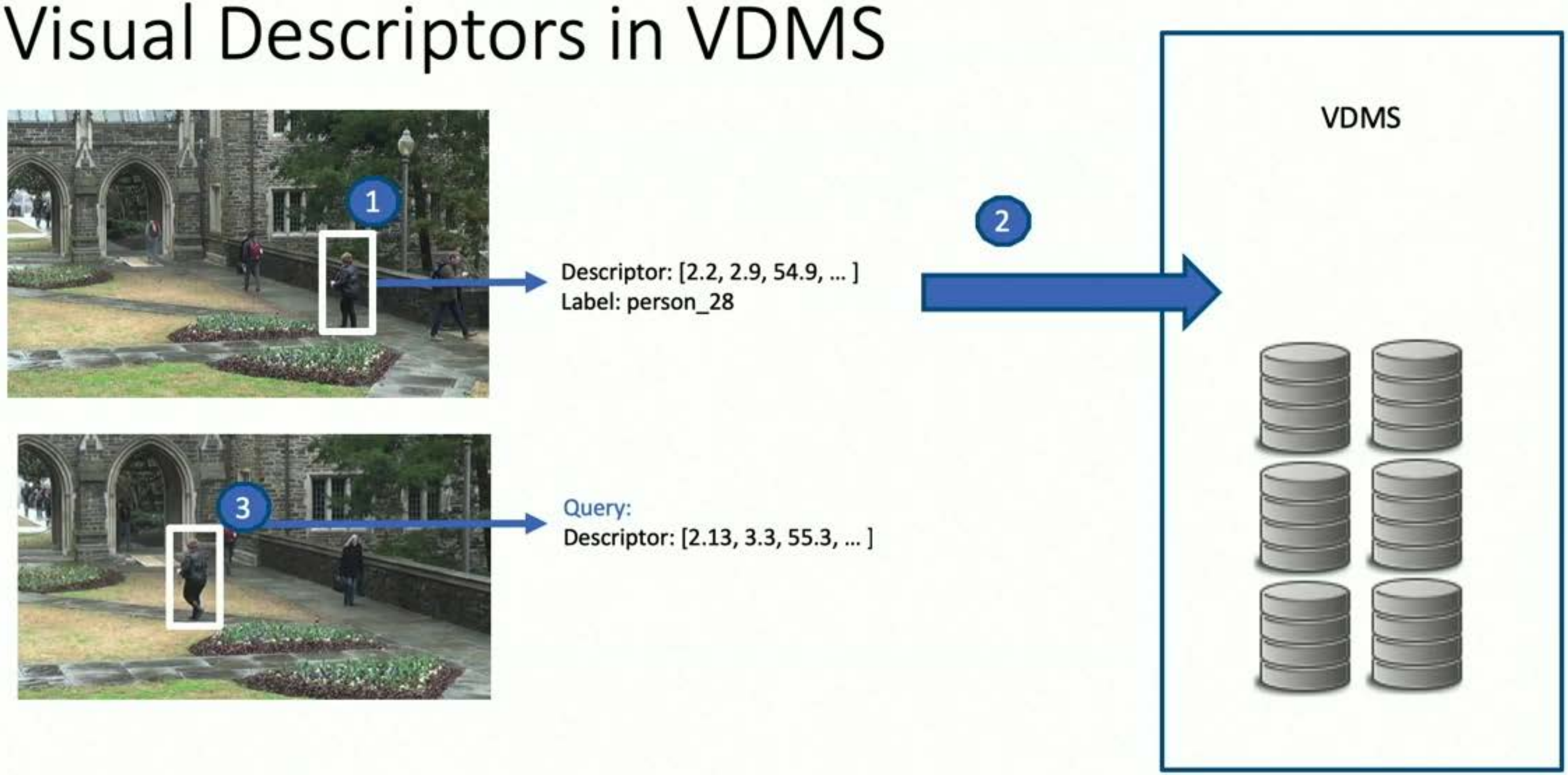
Visual Descriptors in VDMS



Descriptor: [2.2, 2.9, 54.9, ...]
Label: person_28

Novel solution for persistent Feature Vector storage, indexing, and search

Visual Descriptors in VDMS

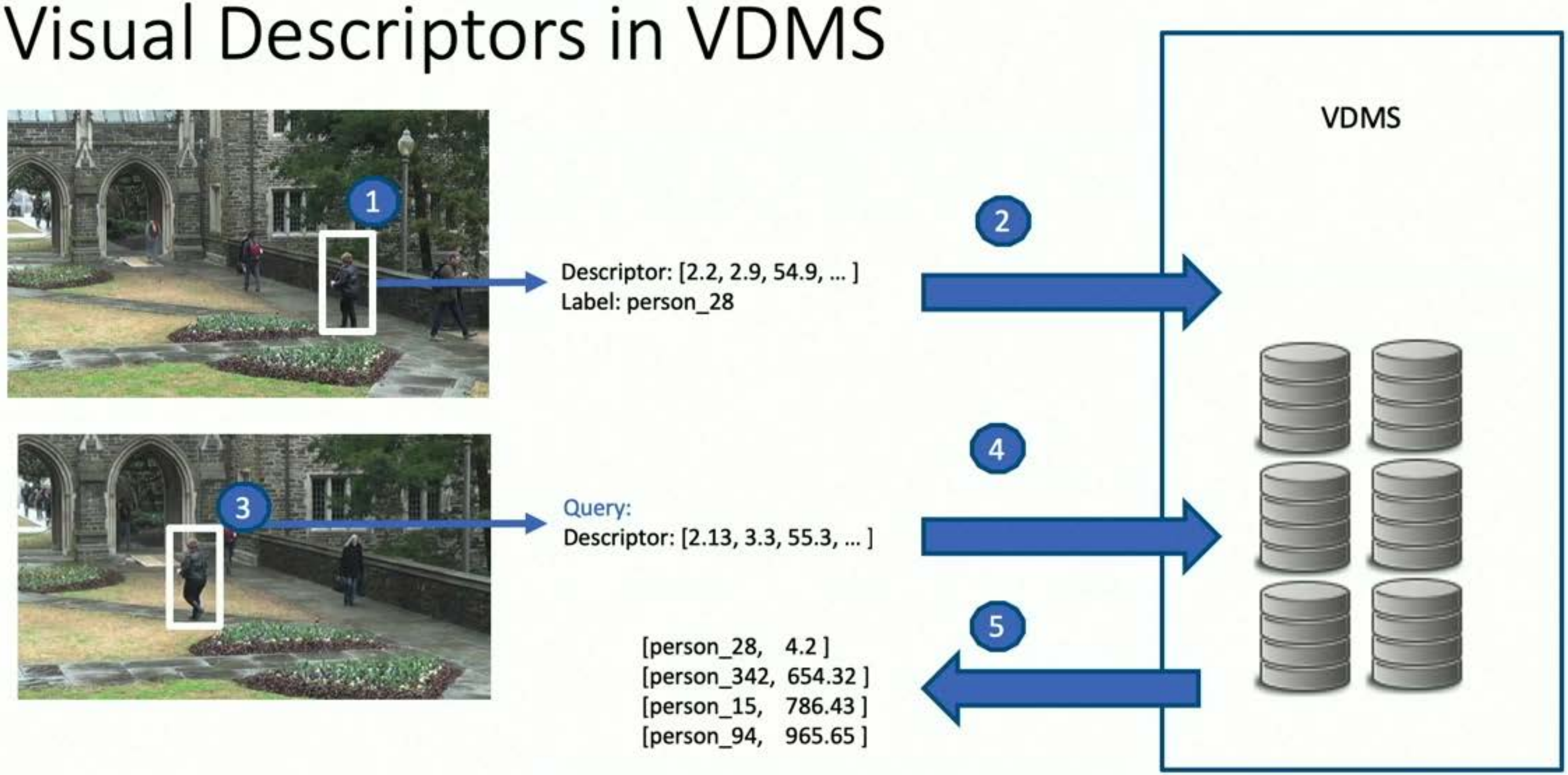


Descriptor: [2.2, 2.9, 54.9, …]
Label: person_28

Query:
Descriptor: [2.13, 3.3, 55.3, …]

Novel solution for persistent Feature Vector storage, indexing, and search

Visual Descriptors in VDMS



Descriptor: [2.2, 2.9, 54.9, …]
Label: person_28

Query:
Descriptor: [2.13, 3.3, 55.3, …]

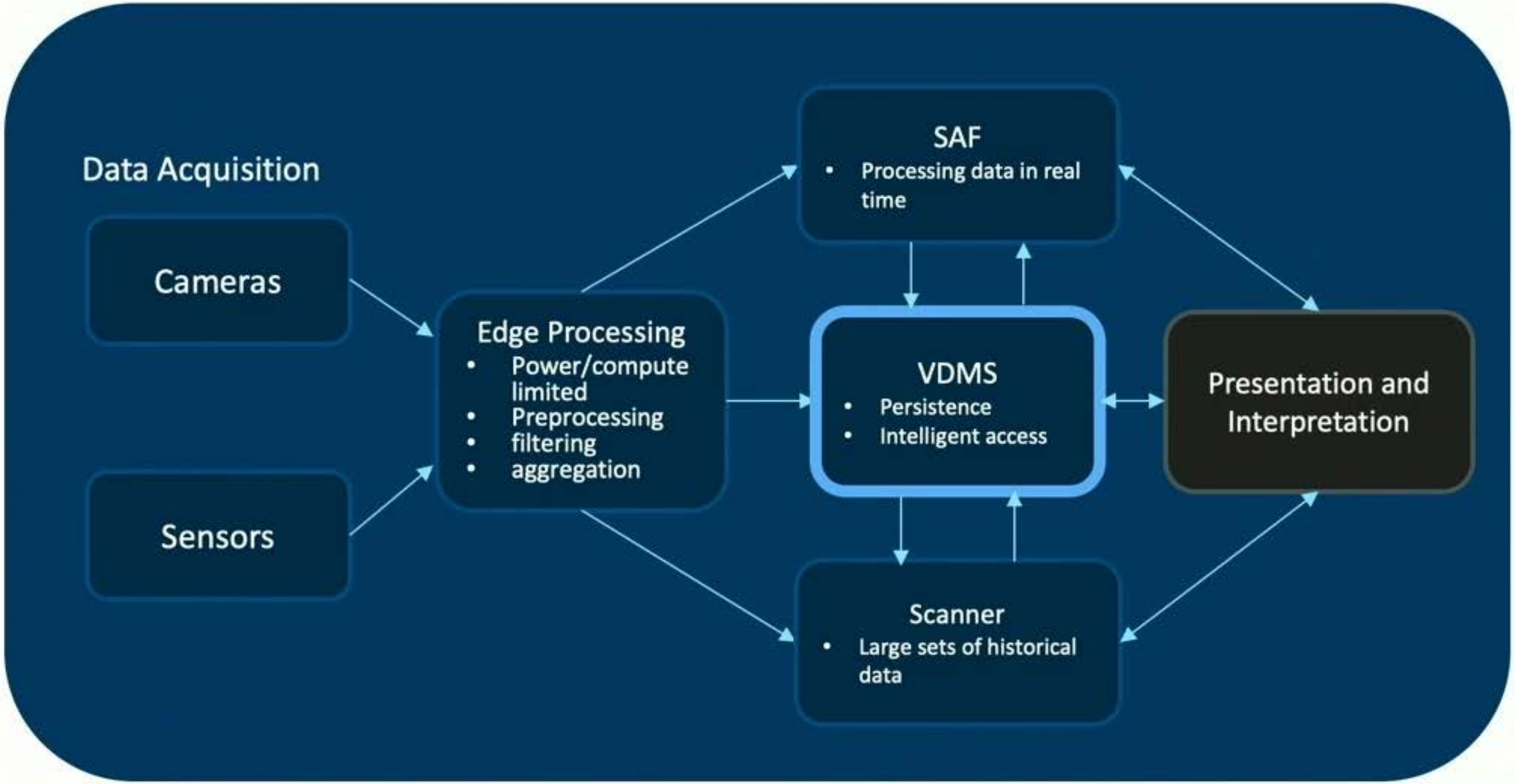
```
[person_28,    4.2 ]
[person_342,  654.32 ]
[person_15,   786.43 ]
[person_94,   965.65 ]
```

Novel solution for persistent Feature Vector storage, indexing, and search

Visual ML Pipeline

Collaboration with Intel Labs

Integration with other Research Project



Data Acquisition

Cameras

Sensors

Edge Processing
- Power/compute limited
- Preprocessing
- filtering
- aggregation

SAF
- Processing data in real time

VDMS
- Persistence
- Intelligent access

Scanner
- Large sets of historical data

Presentation and Interpretation

More Information

- https://aperturedata.io
- https://github.com/IntelLabs/vdms

- **VDMS: Efficient Big-Visual-Data Access for Machine Learning Workloads**
  Luis Remis, Vishakha Gupta-Cledat, et. Al.
  Systems for Machine Learning Workshop @ NIPS 2018
- **Addressing the dark side of vision research: Storage**
  Vishakha Gupta-Cledat, Luis Remis, el al.
  ATC HotStorage 2017

Aperture Data

intel

snowflake

The Snowflake Engine

Northwest Database Society (NWDS)

Annual Meeting 2019

Torsten Grabs - Product Management - torsten.grabs@snowflake.com

Who we are

Founded: August 2012

Mission: The data warehouse for the cloud

HQ in downtown San Mateo (south of San Francisco) with engineering offices in Bellevue, WA, and Berlin, Germany

1000+ employees, ~150 engs (and hiring…)

- Founders: Benoit Dageville, Thierry Cruanes, Marcin Zukowski
- CEO: Bob Muglia

GA in 2015

Raised over $900M across series A-F

Our Product

The Snowflake Elastic Data Warehouse, or “Snowflake”

- Multi-tenant, transactional, secure, highly scalable, elastic
- Implemented from scratch (no Hadoop, Postgres etc.)

Currently runs in the Amazon cloud (AWS) and Microsoft Azure

Serves millions of queries per day over 10s of petabyte of data

1500+ active customers, growing fast

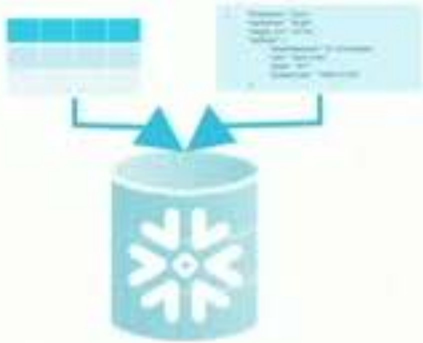
Our Vision for a Cloud Data Warehouse

Data warehouse as a service

No infrastructure to manage, no knobs to tune

Multidimensional elasticity

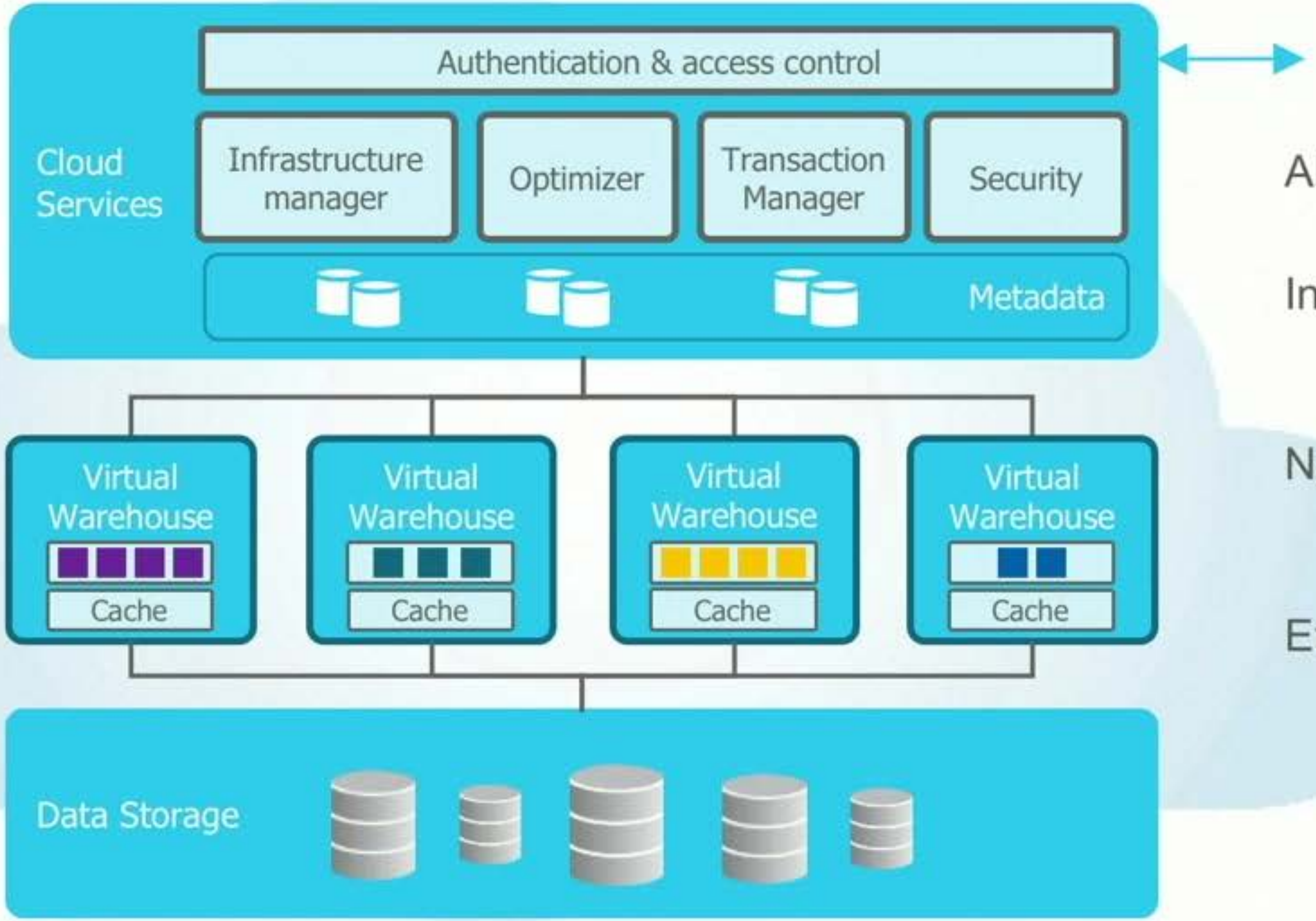
On-demand scalability data, queries, users



All business data

Native support for relational + semi-structured data

Multi-cluster Shared-data Architecture



All data in one place

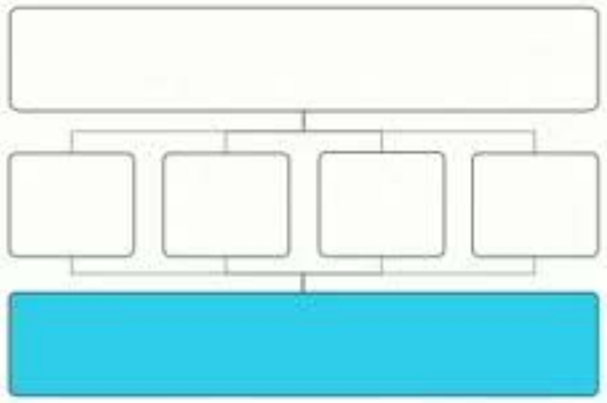
Independently scale storage and compute

No unload / reload to shut off compute

Every virtual warehouse can access all data

snowflake

Data Storage Layer



Stores table data and query results

Uses cloud-based blob storage in AWS or Azure

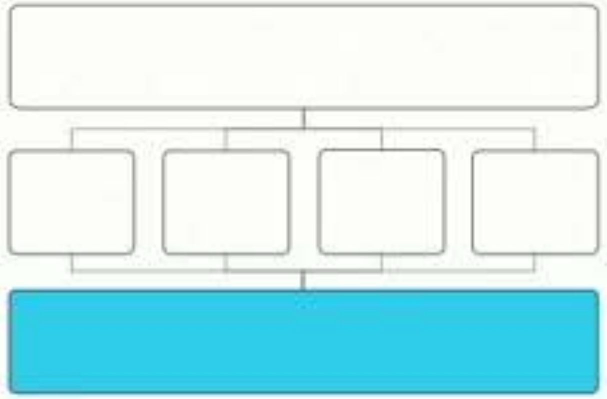
- Object store (key-value) with HTTP(S) PUT/GET/DELETE interface
- High availability, extreme durability (11-9)

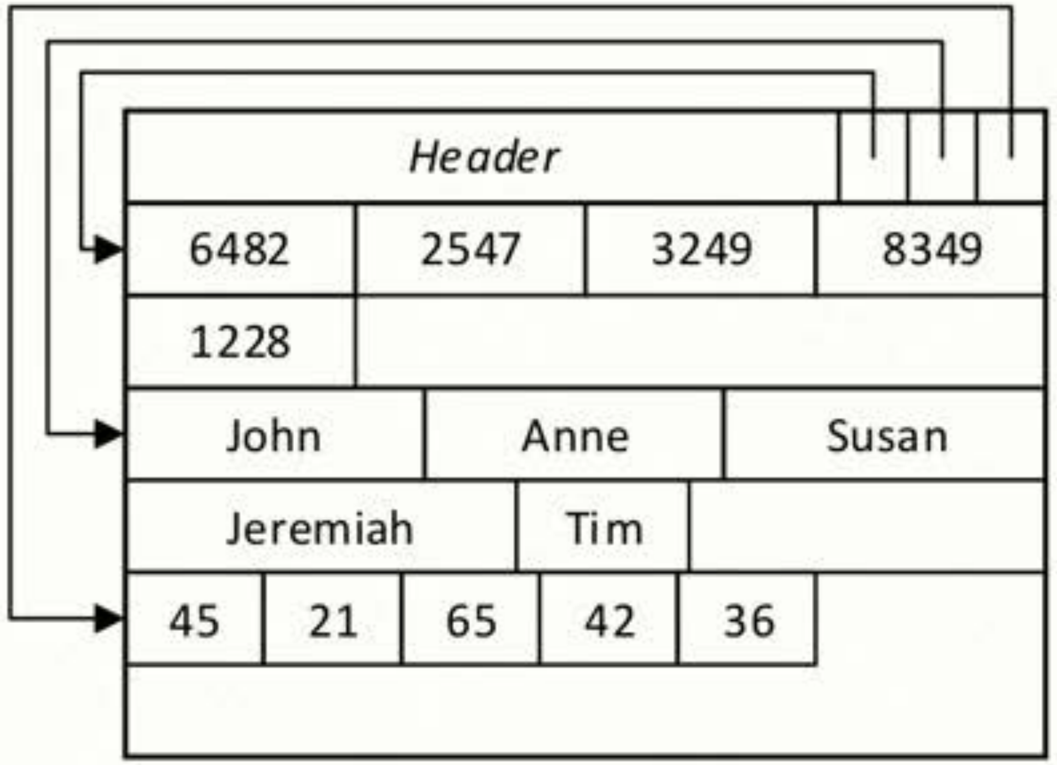
Some important differences w.r.t. local disks

- Performance (sure…)
- No update-in-place, objects must be written in full
- But: can read parts (byte ranges) of objects

Strong influence on table file format and concurrency control

Table Files





Snowflake uses PAX [Ailamaki01] aka hybrid columnar storage for table files

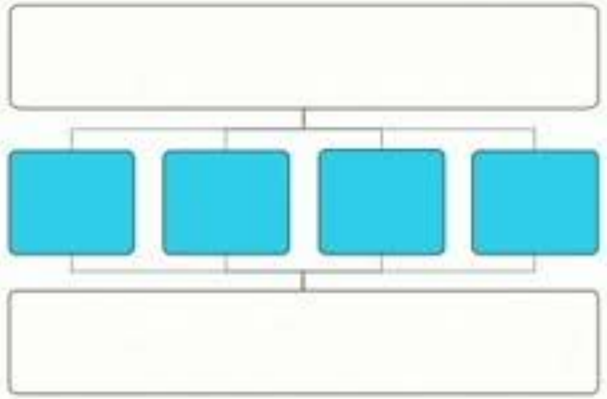
Tables horizontally partitioned into large immutable files (~16 MB each)

- Updates add or remove entire files
- Values of each column grouped together and compressed
- Queries read header + columns they need
- Old table versions retained for time travel

Metadata stored in a transactional key-value store (not blob storage)

- Which table consists of which blob storage objects
- Optimizer statistics, lock tables, transaction logs etc.
- Part of Cloud Services layer (see later)

Virtual Warehouse



VW = Cluster of cloud compute VM instances called worker nodes

Pure compute resources

- Created, destroyed, resized on demand
- Users may run multiple VW at same time
- Each VW has access to all data but isolated performance
- Users may shut down *all* VWs when they have nothing to run

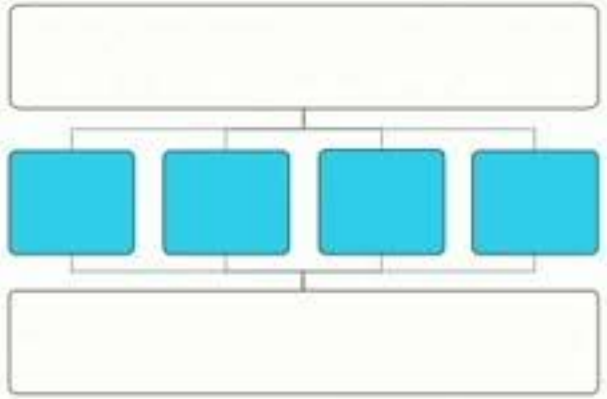
T-Shirt sizes: XS to 4XL

- Users do not know which type or how many VM instances
- Service and pricing can evolve independent of cloud platform

Each worker node maintains local table cache

- Collection of table files i.e. cloud storage objects accessed in past
- Shared across concurrent and subsequent worker processes
- Assignment of table files to nodes using consistent hashing

Execution Engine



Columnar [MonetDB, C-Store, many more]

- Effective use of CPU caches, SIMD instructions, and compression

Vectorized [Zukowski05]

- Operators handle batches of a few thousand rows in columnar format
- Avoids materialization of intermediate results

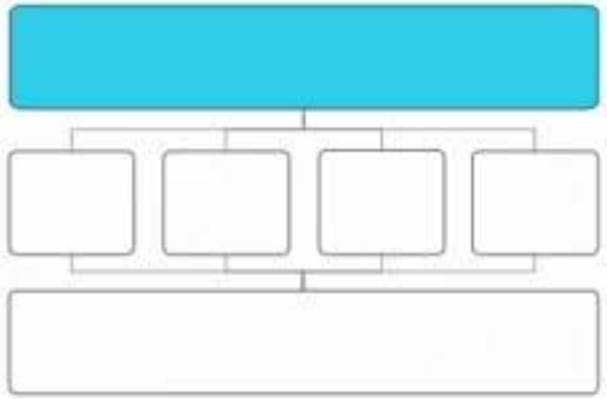
Push-based [Neumann11]

- Operators push results to downstream operators (no Volcano iterators)
- Removes control logic from tight loops
- Works well with DAG-shaped plans

No transaction management, no buffer pool

- But: most operators (join, group by, sort) can spill to disk and recurse

Cloud Services



Collection of services

- Access control, query optimizer, transaction manager etc.

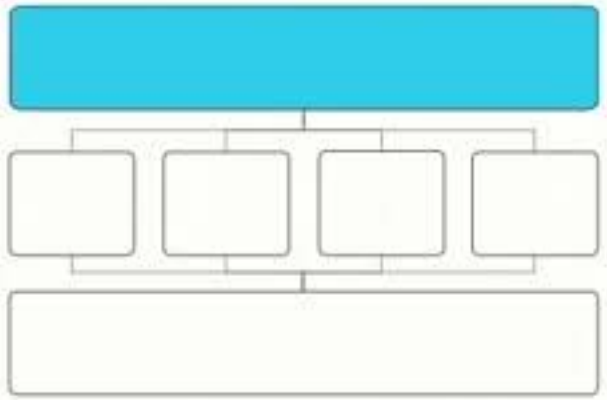
Heavily multi-tenant (shared among users) and always on

- Improves utilization and reduces administration

Each service replicated for availability and scalability

- Hard state stored in transactional key-value store

Concurrency Control



Designed for analytic workloads

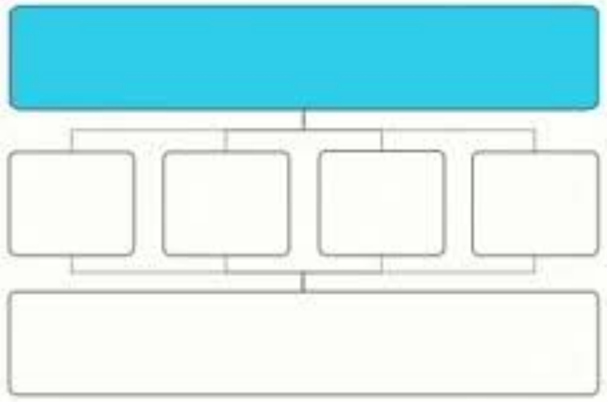
- Large reads, bulk or trickle inserts, bulk updates

Snapshot Isolation (SI) [Berenson95]

SI based on multi-version concurrency control (MVCC)

- DML statements (insert, update, delete, merge) produce new table versions of tables by adding or removing whole files
- Natural choice because table files in cloud storage are immutable
- Additions and removals tracked in metadata (key-value store)

Versioned snapshots used also for time travel and cloning



Pruning

Database adage: The fastest way to process data? Don't.

- Limiting access only to relevant data is key aspect of query processing

Traditional solution: B^+-trees and other indices

- Poor fit for us: random accesses, high load time, manual tuning

Snowflake approach: pruning

- AKA small materialized aggregates [Moerkotte98], zone maps [Netezza], data skipping [IBM]
- Per file min/max values, #distinct values, #nulls, bloom filters etc.
- Use metadata to decide which files are relevant for a given query
- Smaller than indices, more load-friendly, no user input required

Ongoing Challenges

Scale

- Support thousands of concurrent users, some of which do *weird* things
- Metadata layer is becoming huge
- Customer data is becoming huge
- More cloud regions across the globe
- Categorizing and handling failures automatically is very hard
- *Automation* is key to keeping operations lean

Serverless computing paradigm

Continuous and low latency data ingestion

Data sharing and collaboration over data

Lots of other work left to do

- SQL performance improvements.
- Stronger integration with 3rd party tools
- Self-service model
- Multi-account manageability
- Data visualization

It’s a wrap

Snowflake is an enterprise-ready data warehouse as a service

- Novel multi-cluster, shared-data architecture
- Highly elastic and available
- Semi-structured and schema-less data at the speed of relational data
- Pure SaaS experience

Rapidly growing user base and data volume

Lots of challenging work left to do

Veritas: Overlaying Distributed Database Applications over Blockchains

Donald Kossmann

Microsoft Research

Value Prop of Blockchain
=
Proof for Digital Transactions

Transactions in the Real World

- **All Transactions require Proof: Witnesses and/or Receipts**
  - getting married (best man + ring)
  - buying a house (notary + contract)
  - drinking alcohol (driver's licence)

- **Why do we need Proof?**
  - transactions have *conditions* and come with *rights* & *accountabilities*
    - getting married: „*I am married to you! Please, be nice to me!*"
    - buying a house: „*I am the rightful owner of the house! I am allowed to live here.*"
    - birth: „*I become Donald Kossmann.*" - drinking: „*I am Donald Kossmann!*"
  - Witnesses and receipts provide *proof*: *Proof = Trust*

Transactions in the Digital World

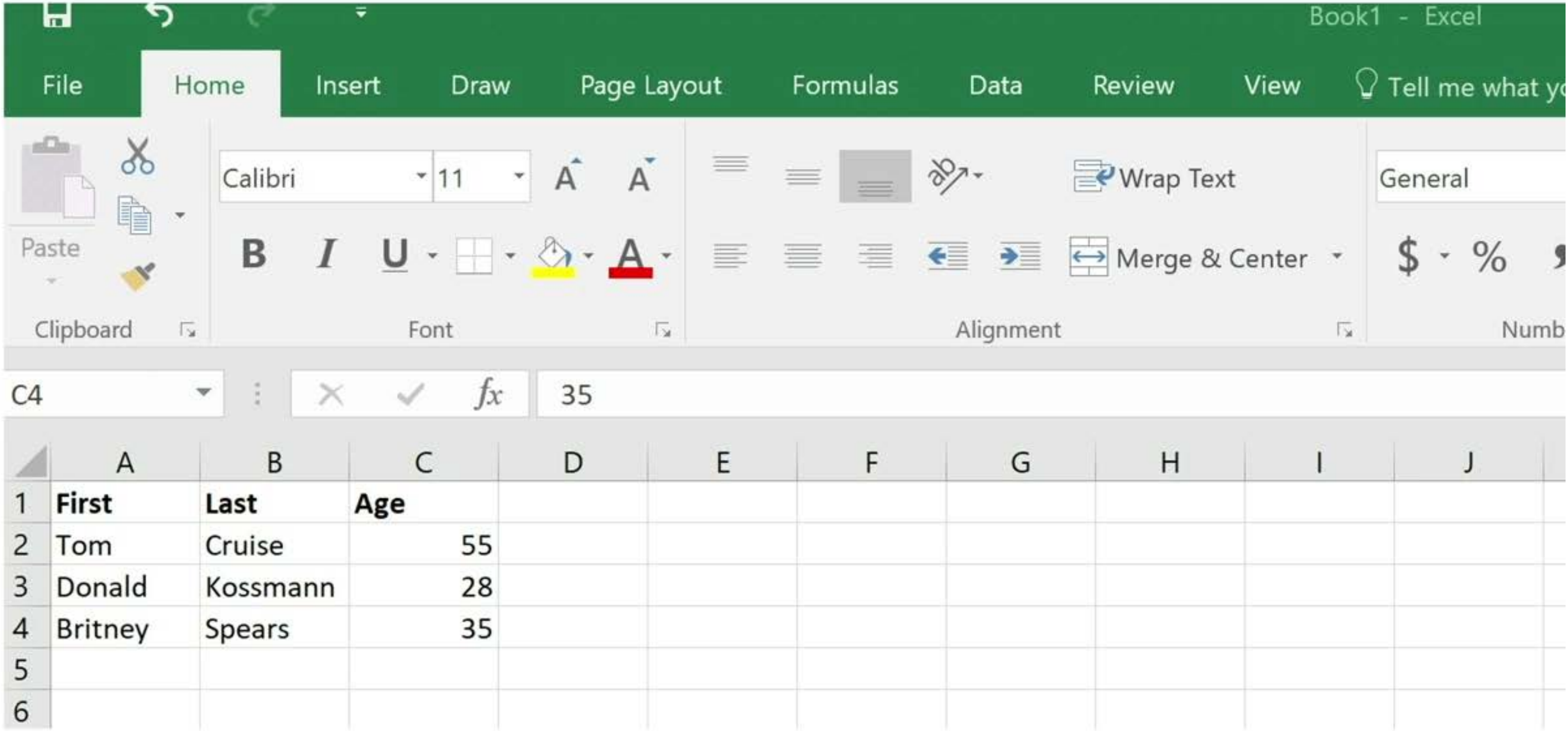
PC Era: If you are alone, you do not need trust (proof)

- user owns and controls all data; user trusts herself

Cloud Era / Connected World: Trust is needed

- users collaborate and share data (e.g., for AI)
- news gets hacked
- users need to verify data before making decisions: How?
  - proof (receipts & witnesses) in the digital world!
  - (Or we are stuck with trusted brands such as Facebook, ...)

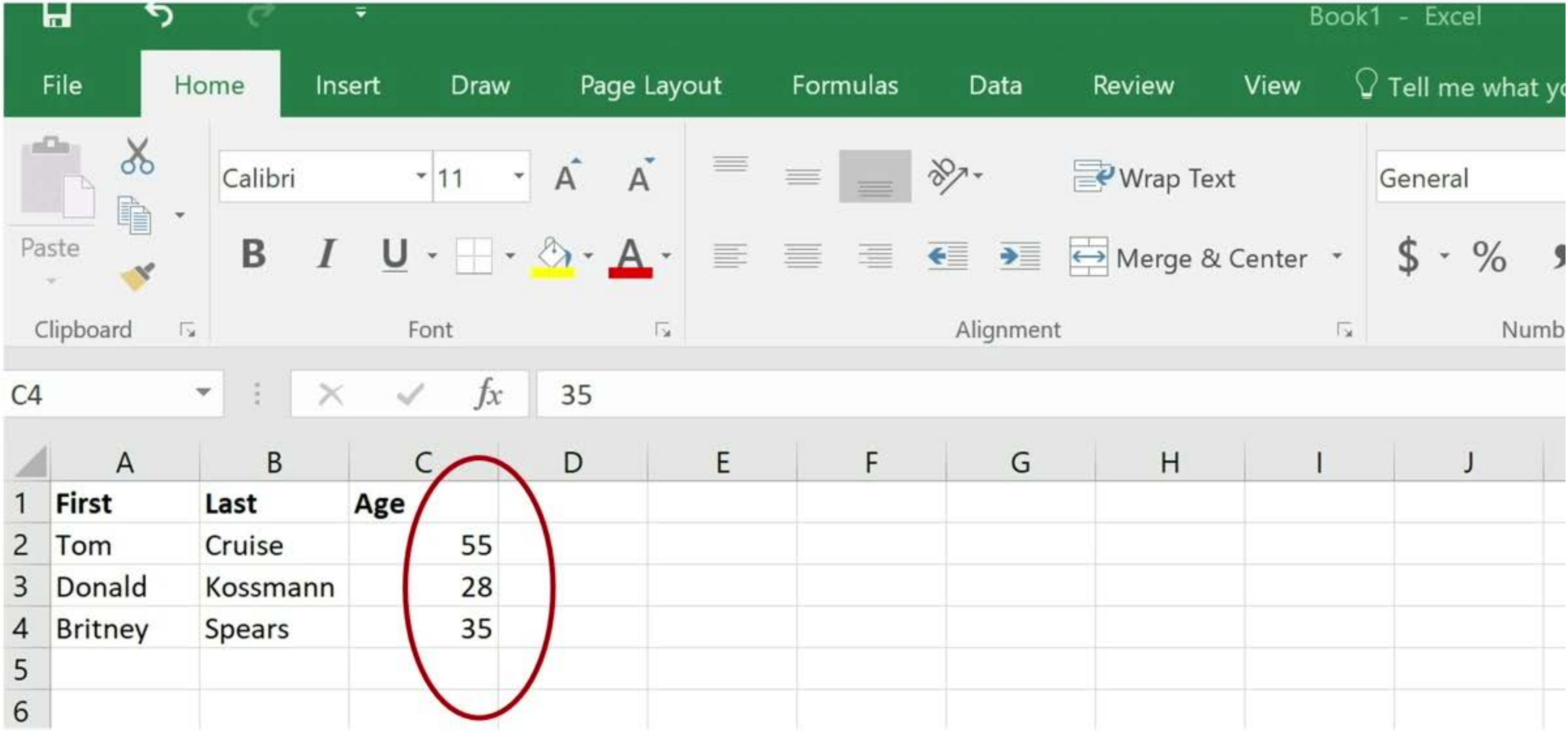
Where is the Trust Button?



C4 | *fx* 35

	A	B	C	D	E	F	G	H	I	J
1	**First**	**Last**	**Age**							
2	Tom	Cruise	55							
3	Donald	Kossmann	28							
4	Britney	Spears	35							
5										
6										

Where is the Trust Button?



	A	B	C	D	E	F	G	H	I	J
1	**First**	**Last**	**Age**							
2	Tom	Cruise	55							
3	Donald	Kossmann	28							
4	Britney	Spears	35							
5										
6										

The Dream: Automate Trust

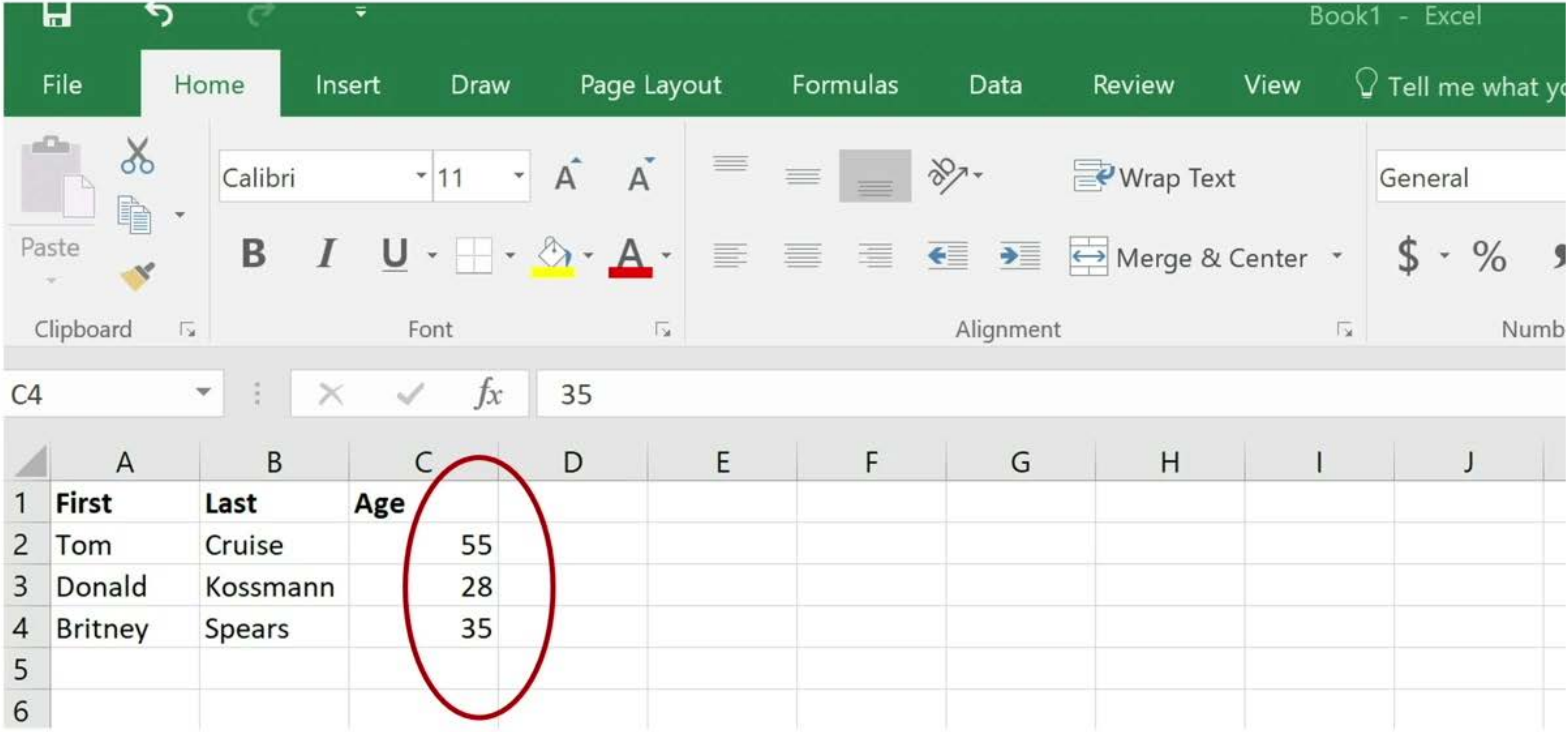
Create Proof in the Connected Digital World

- trace data and transactions
- every document comes with proof (verification)
- I can prove that I did the right thing

Blockchain is a nice building block, but not enough

- Issues: Integration, Performance, Privacy, …
- Challenge: Retrofit trust into existing applications

Where is the Trust Button?



C4

35

	A	B	C	D	E	F	G	H	I	J
1	**First**	**Last**	**Age**							
2	Tom	Cruise	55							
3	Donald	Kossmann	28							
4	Britney	Spears	35							
5										
6										

The Dream: Automate Trust

Create Proof in the Connected Digital World

- trace data and transactions
- every document comes with proof (verification)
- I can prove that I did the right thing

Blockchain is a nice building block, but not enough

- Issues: Integration, Performance, Privacy, ...
- Challenge: Retrofit trust into existing applications

Overview

Blockchain 101

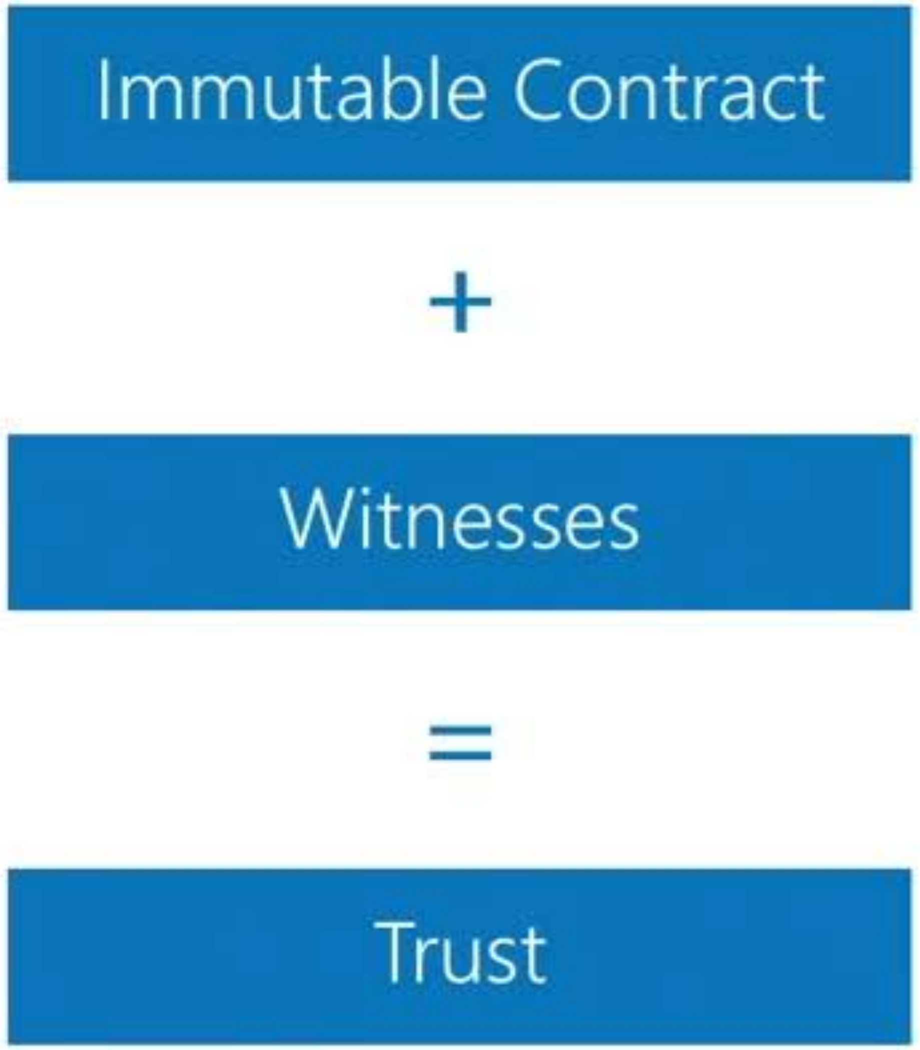
Veritas: Integrating Proof into Databases

Example: Decentralized ID

Blockchain 101

Idea 1: Crypto to make transactions **immutable** and **atomic**.

Idea 2: *Consensus protocol* to commit a transaction. **Community** verifies all transactions.



Immutable Contract

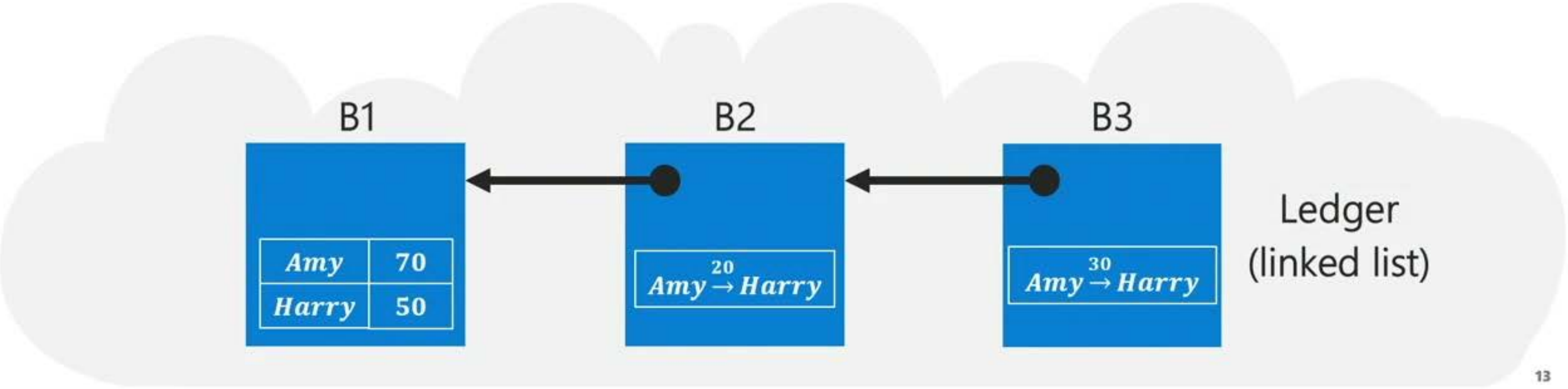
+

Witnesses

=

Trust

BLOCKCHAIN 101: UNTRUSTED LEDGER



B1

Amy	70
Harry	50

B2

$Amy \xrightarrow{20} Harry$

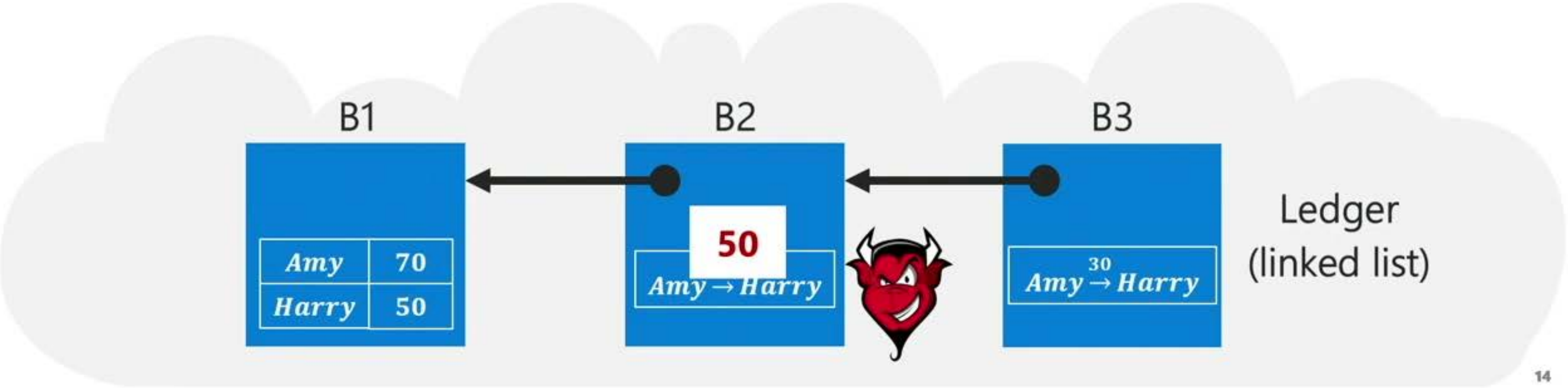
B3

$Amy \xrightarrow{30} Harry$

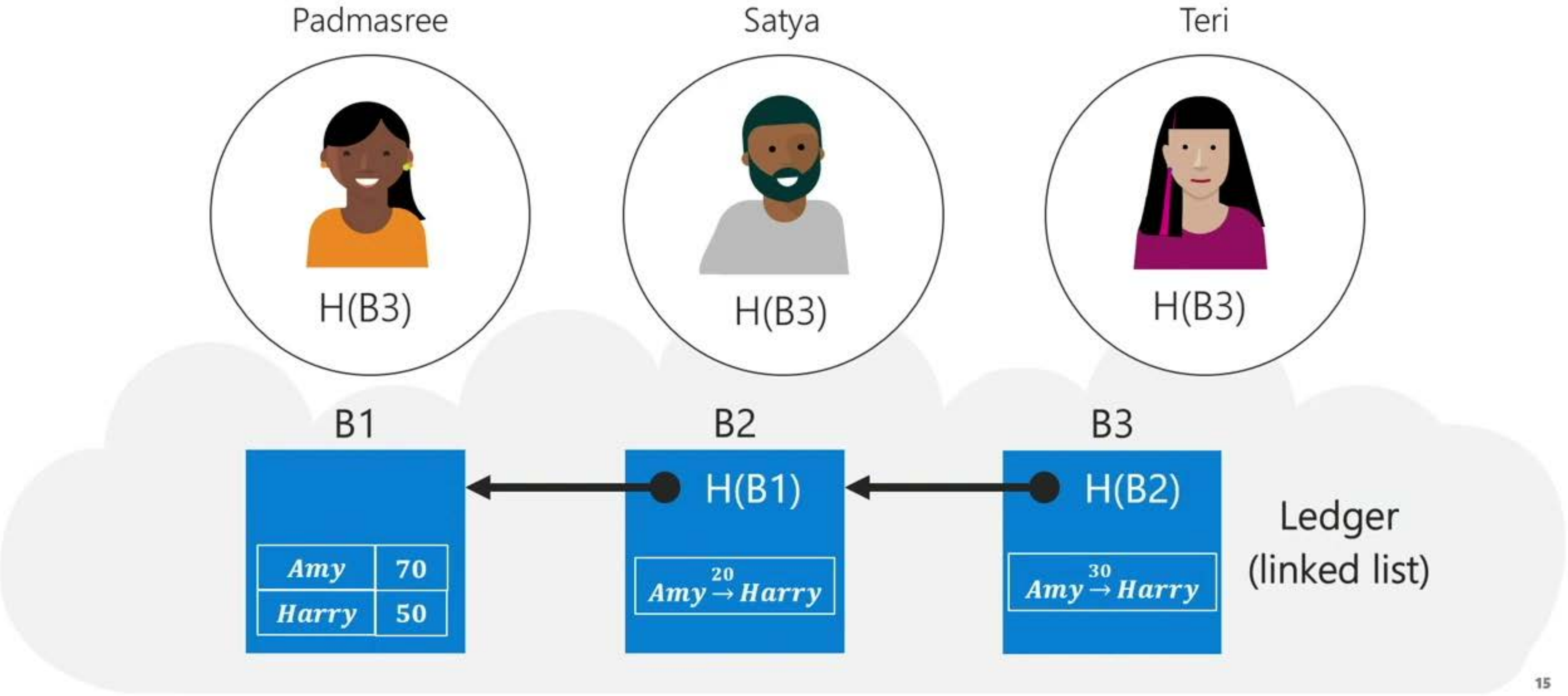
Ledger
(linked list)

BLOCKCHAIN 101: UNTRUSTED LEDGER

Problem: No protection against greedy Harry!



BLOCKCHAIN 101



Padmasree

H(B3)

Satya

H(B3)

Teri

H(B3)

B1

Amy	70
Harry	50

B2

H(B1)

$Amy \xrightarrow{20} Harry$

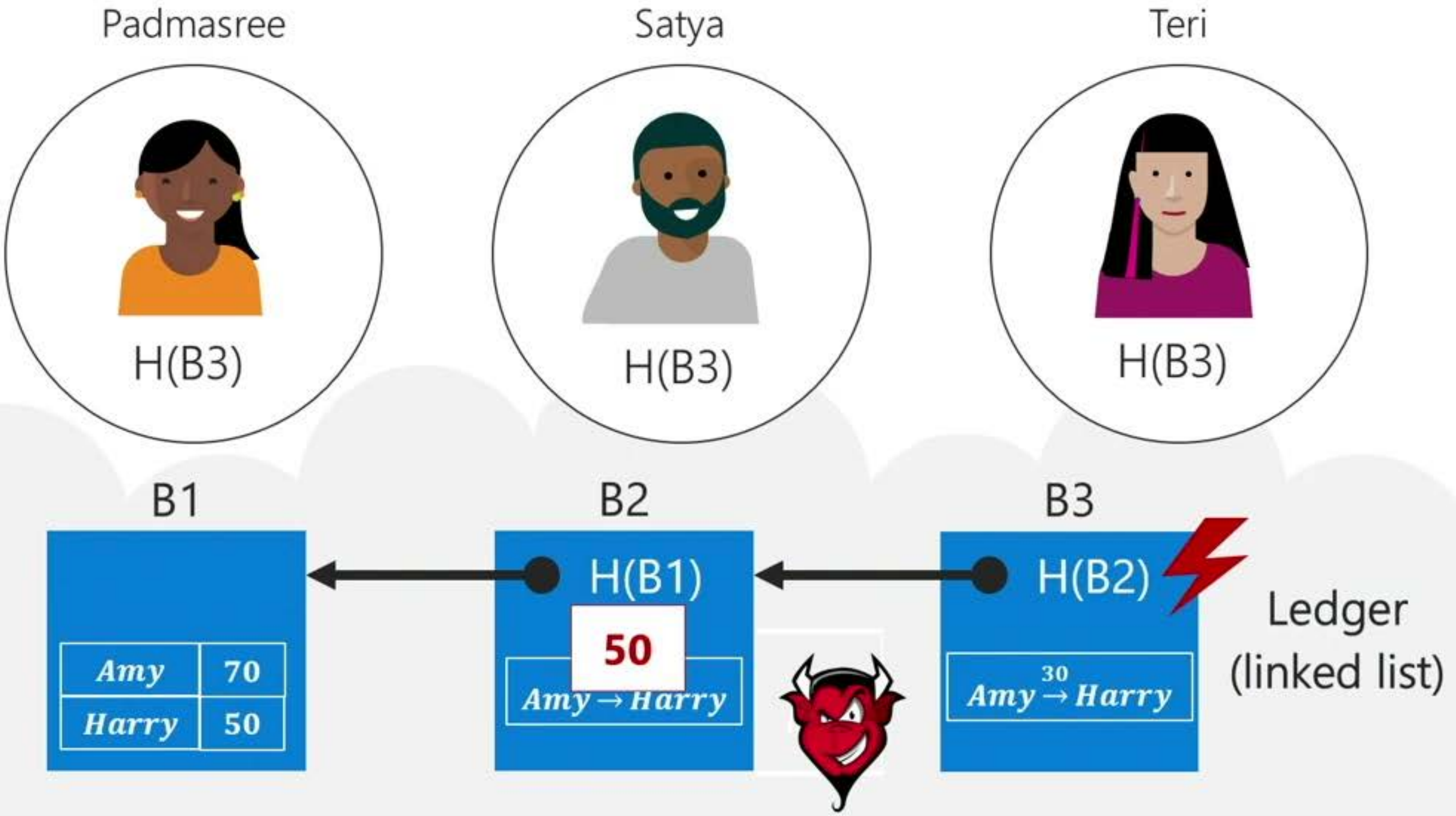
B3

H(B2)

$Amy \xrightarrow{30} Harry$

Ledger
(linked list)

BLOCKCHAIN 101



Padmasree

H(B3)

Satya

H(B3)

Teri

H(B3)

B1

Amy	70
Harry	50

B2

H(B1)

50

$Amy \rightarrow Harry$

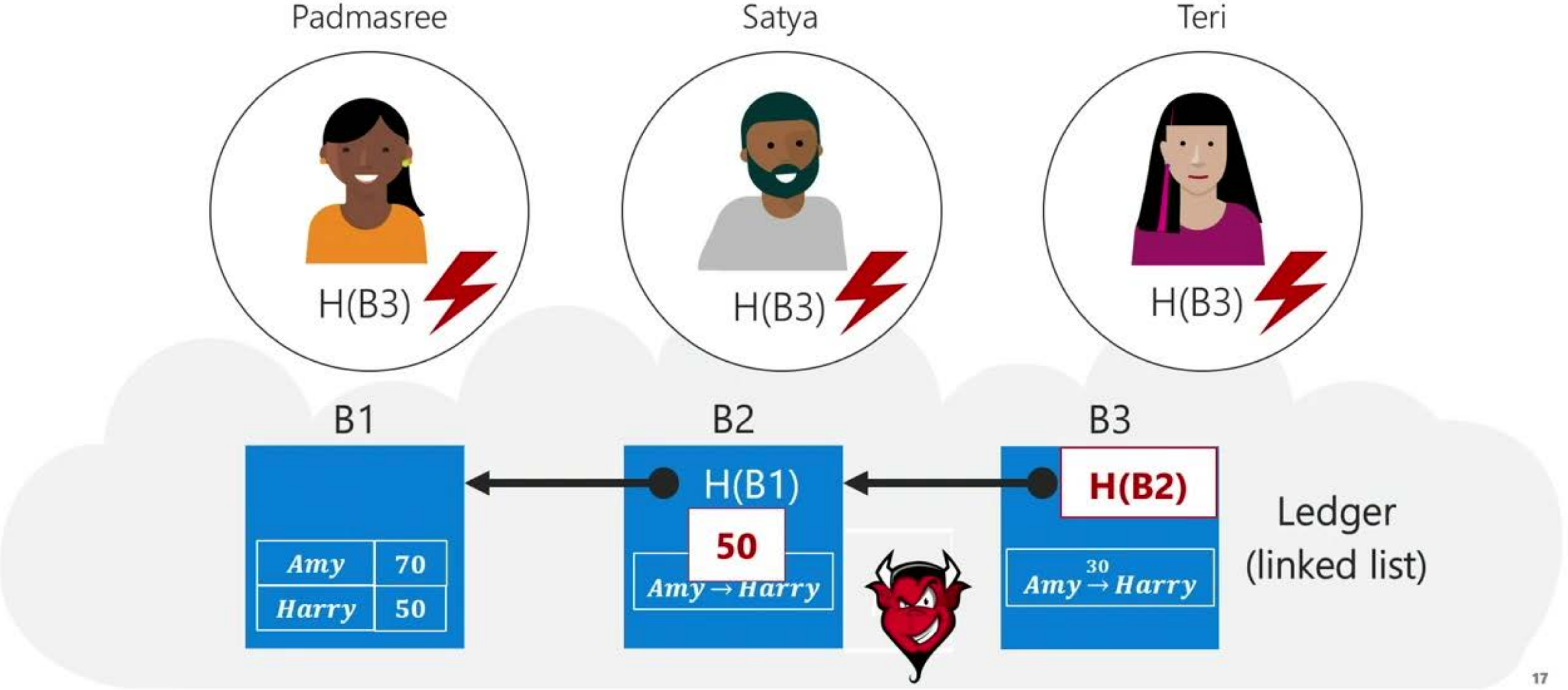
B3

H(B2)

$Amy \xrightarrow{30} Harry$

Ledger
(linked list)

BLOCKCHAIN 101



Padmasree

Satya

Teri

H(B3)

H(B3)

H(B3)

B1

B2

B3

Amy	70
Harry	50

H(B1)

50

$Amy \rightarrow Harry$

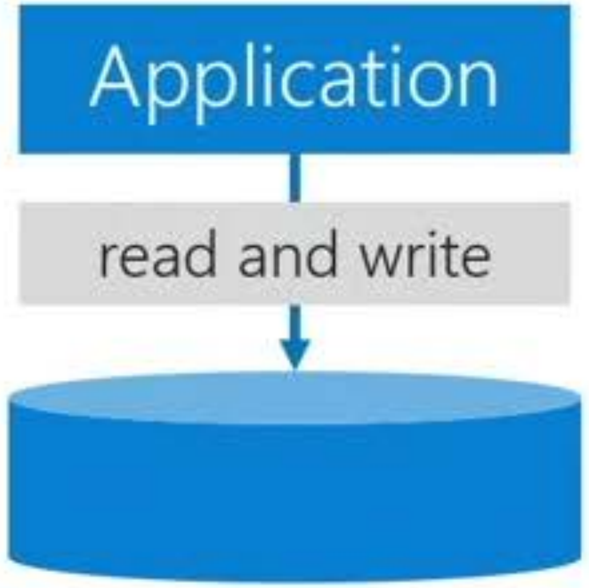
H(B2)

$Amy \xrightarrow{30} Harry$

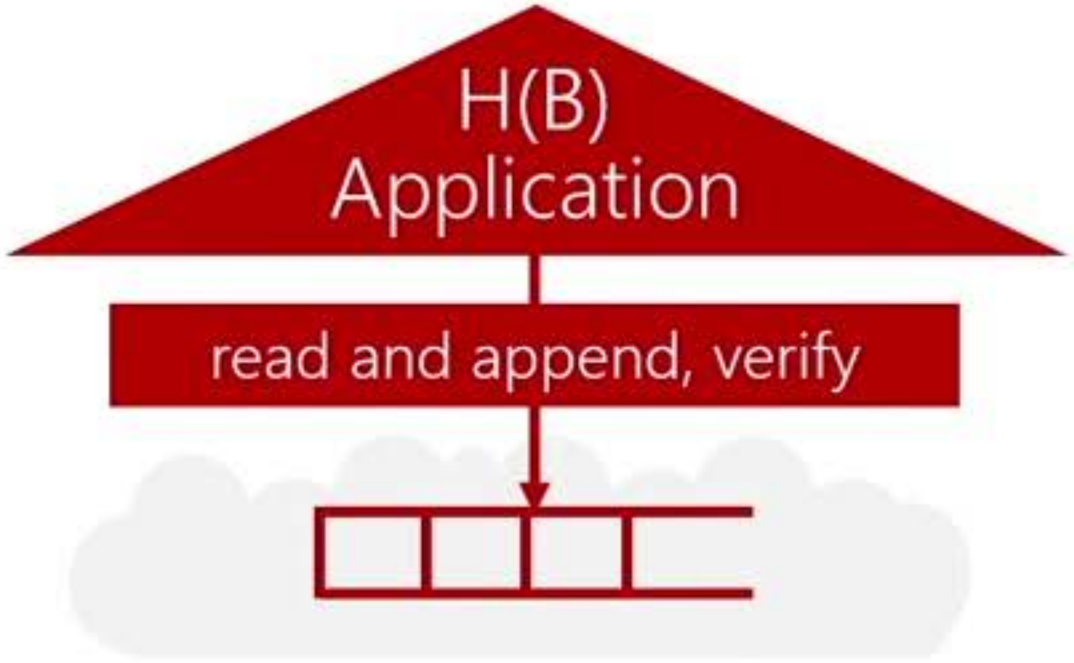
Ledger
(linked list)

TRADITIONAL VS. BLOCKCHAIN

Traditional



Blockchain



Traditional IT Systems

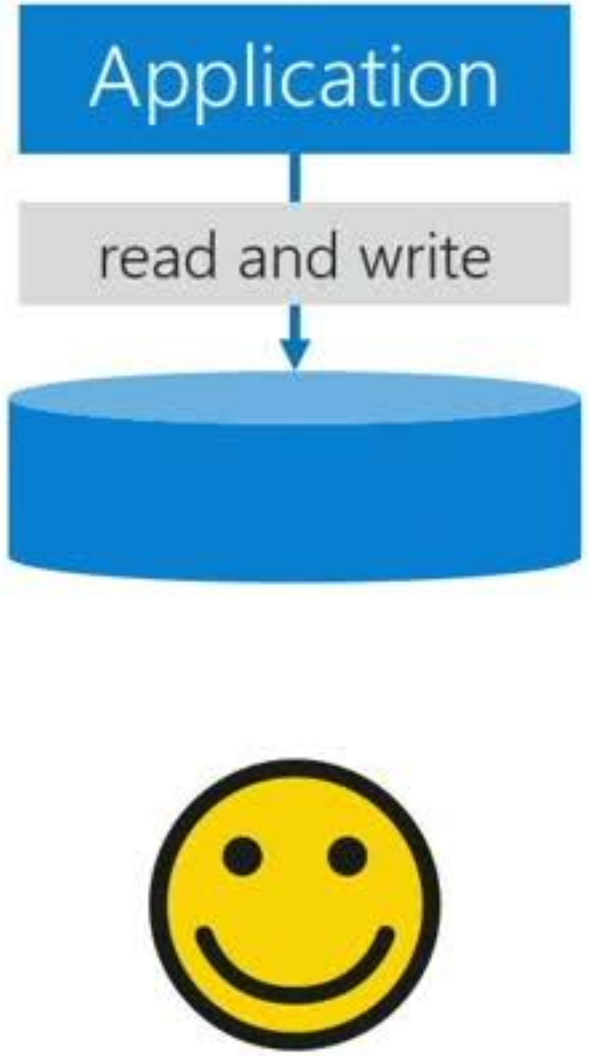
- Productivity: great abstractions
- Security: proven technology
- Performance: millions ops/sec
- Standardization
- but no proofs

Blockchain

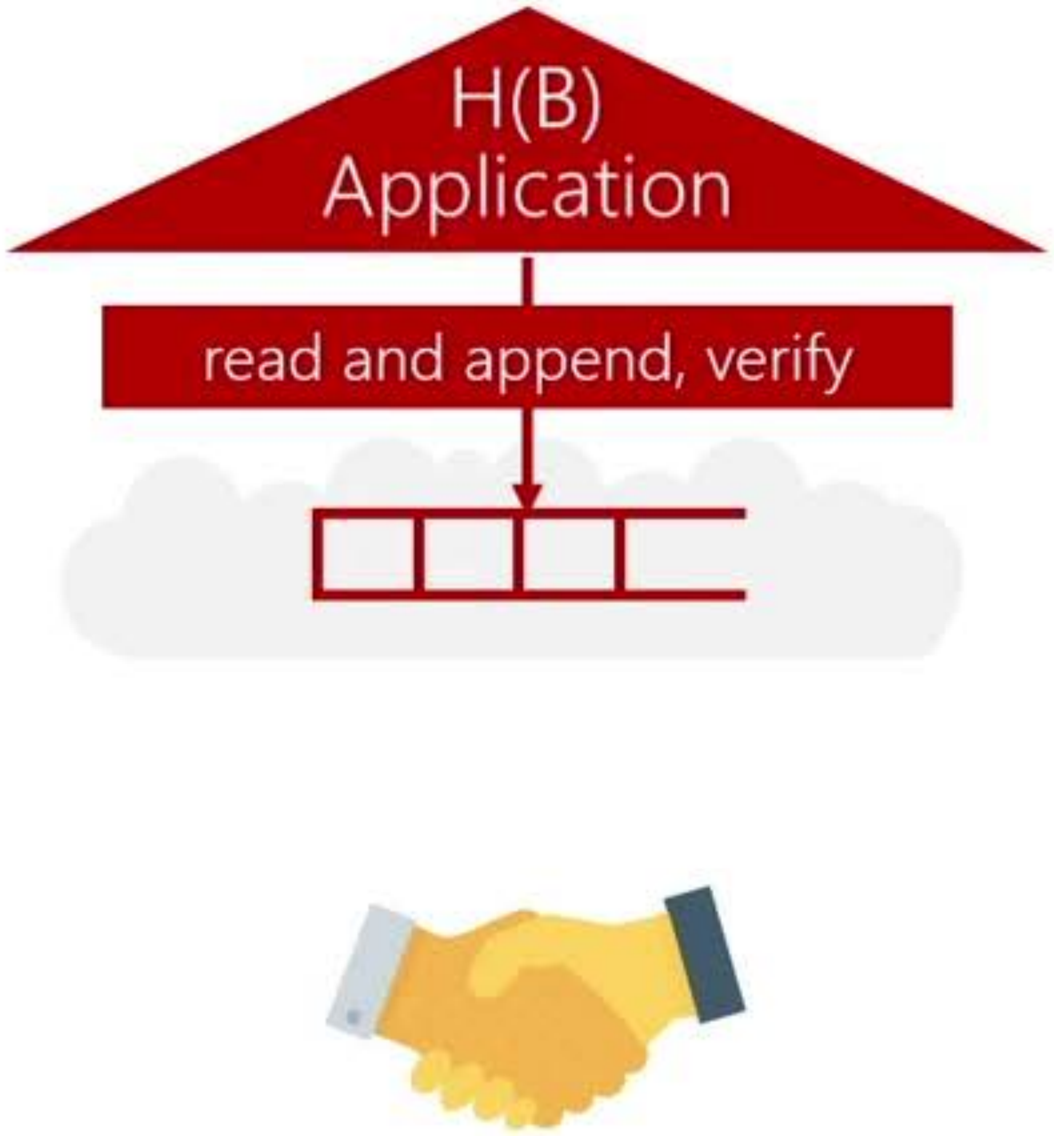
- reinvent the wheel
- but proofs

VERITAS

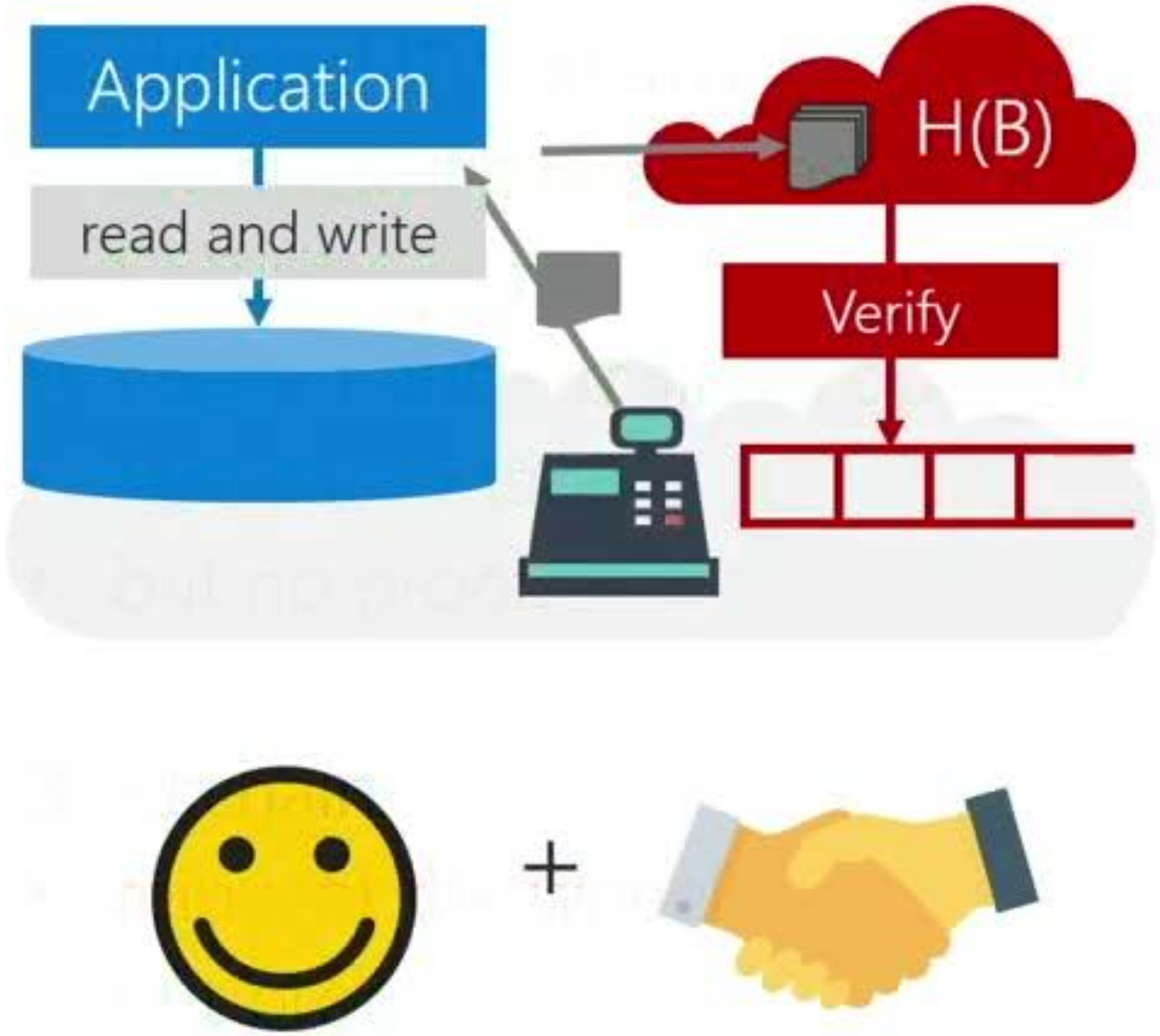
Traditional



Blockchain



Veritas

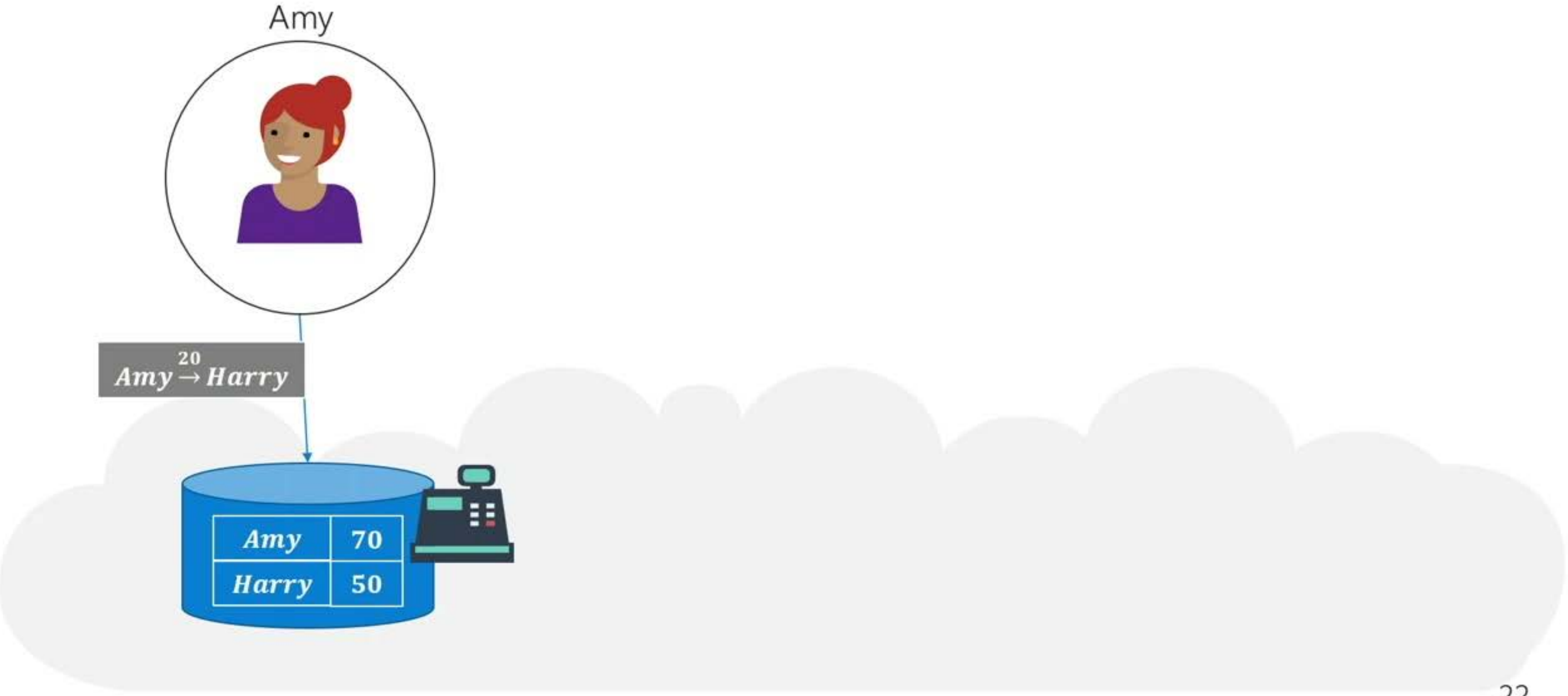


Veritas

- Separate Generation of Contracts (Idea 1) from Verification (Idea 2)
  - Digital Register generates receipts embedded in application
  - [illegible] Service collects receipts and verifies log, separate from application
- Trust Speed of Blockchains without Rebuilding Applications

Veritas

VERITAS

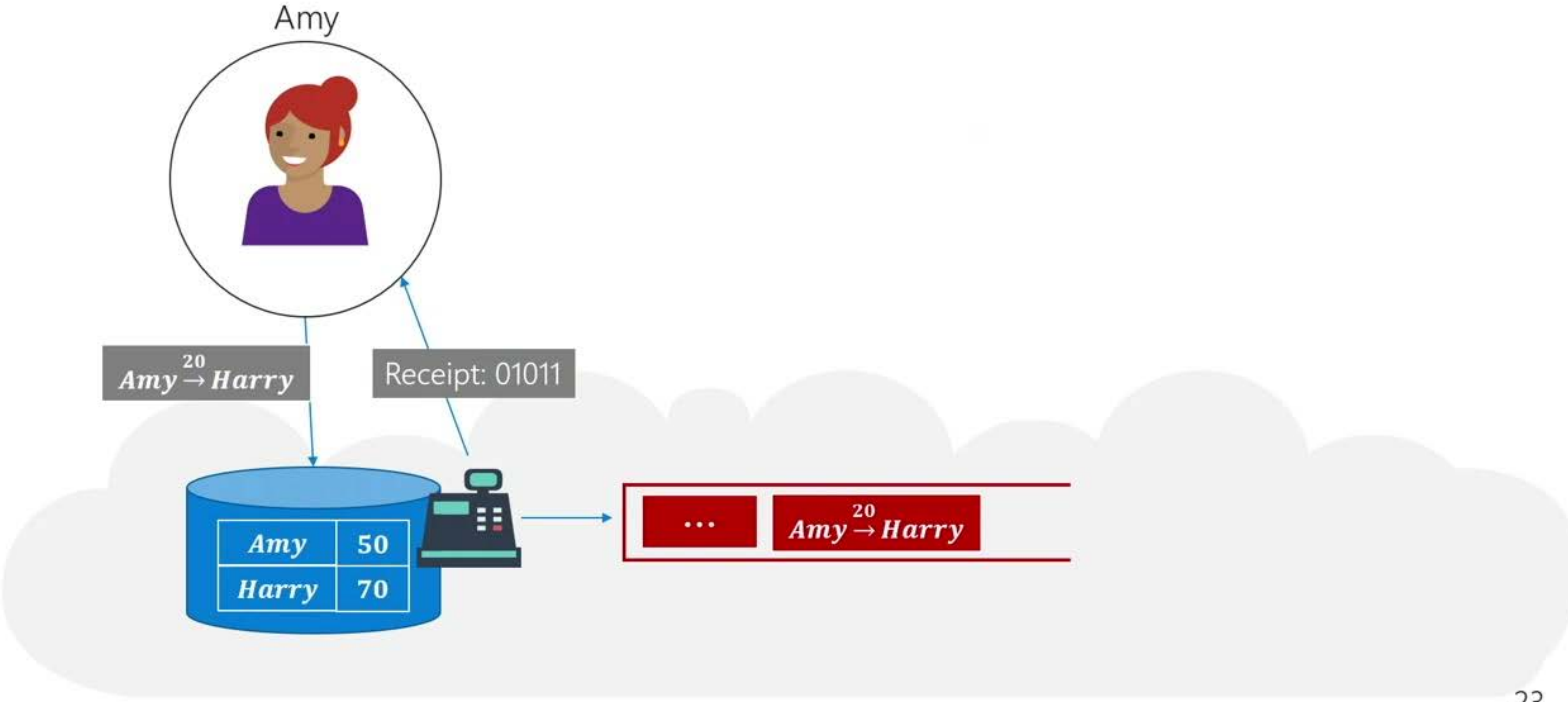


Amy

$Amy \xrightarrow{20} Harry$

Amy	70
$Harry$	50

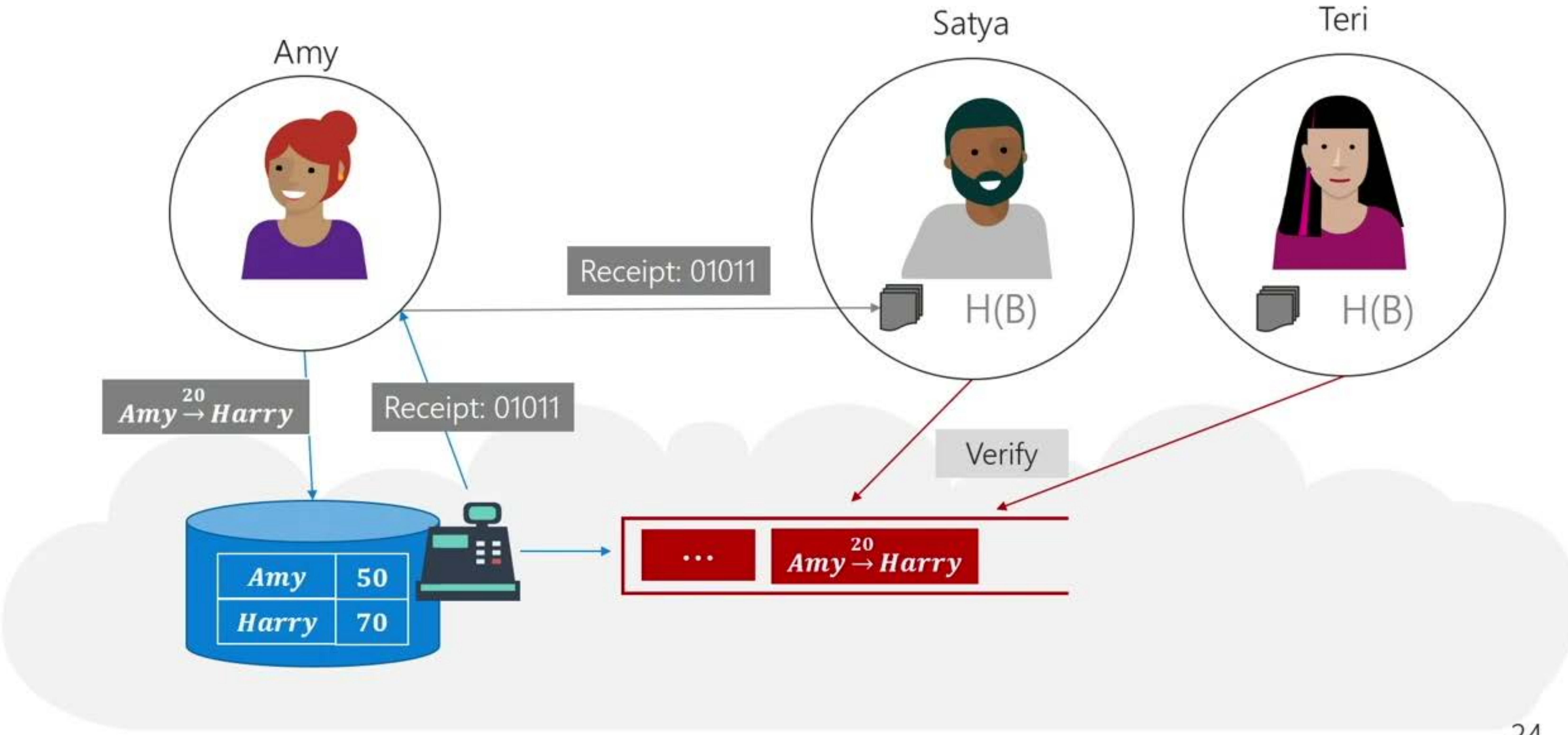
VERITAS



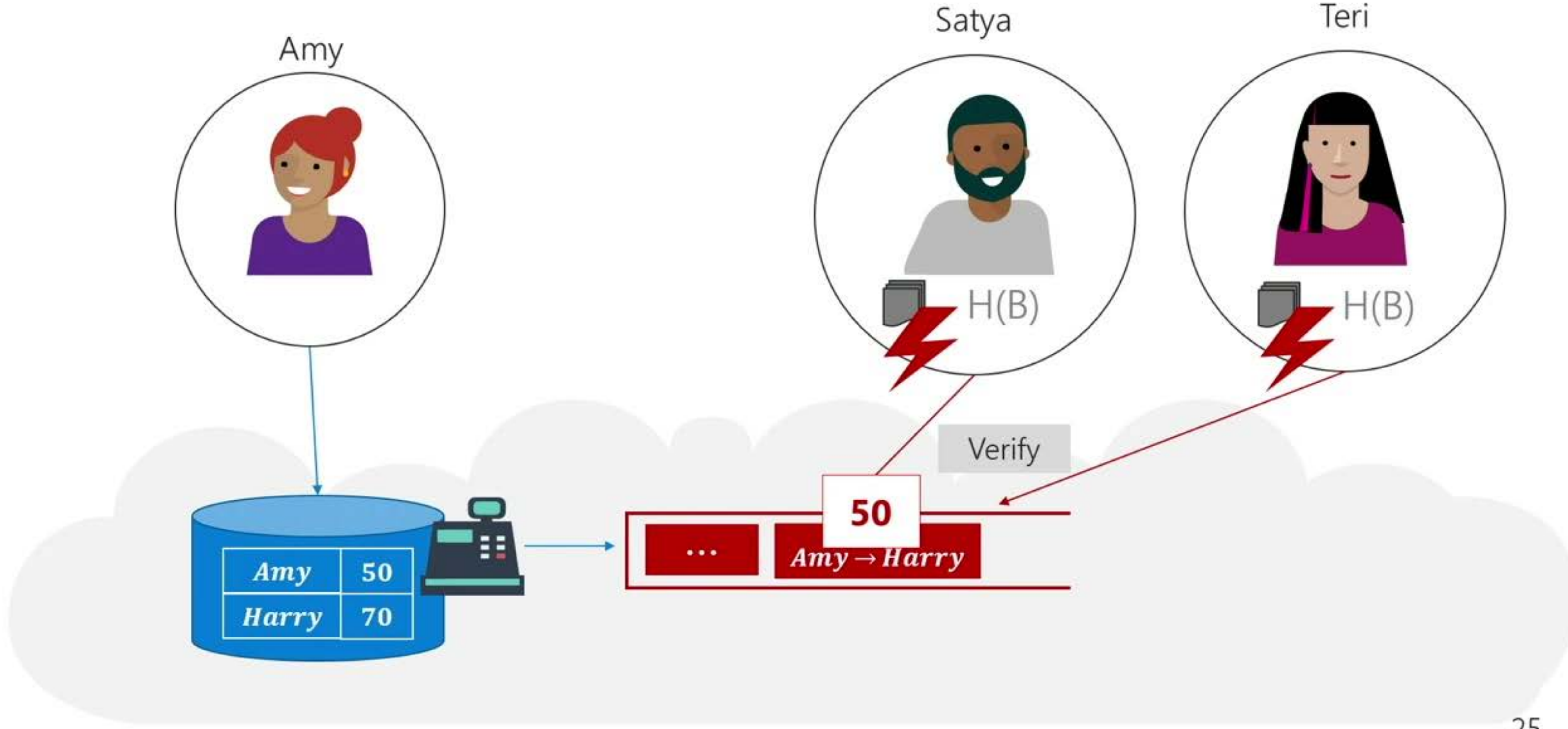
Amy
$Amy \overset{20}{\rightarrow} Harry$
Receipt: 01011
…
$Amy \overset{20}{\rightarrow} Harry$

Amy	50
Harry	70

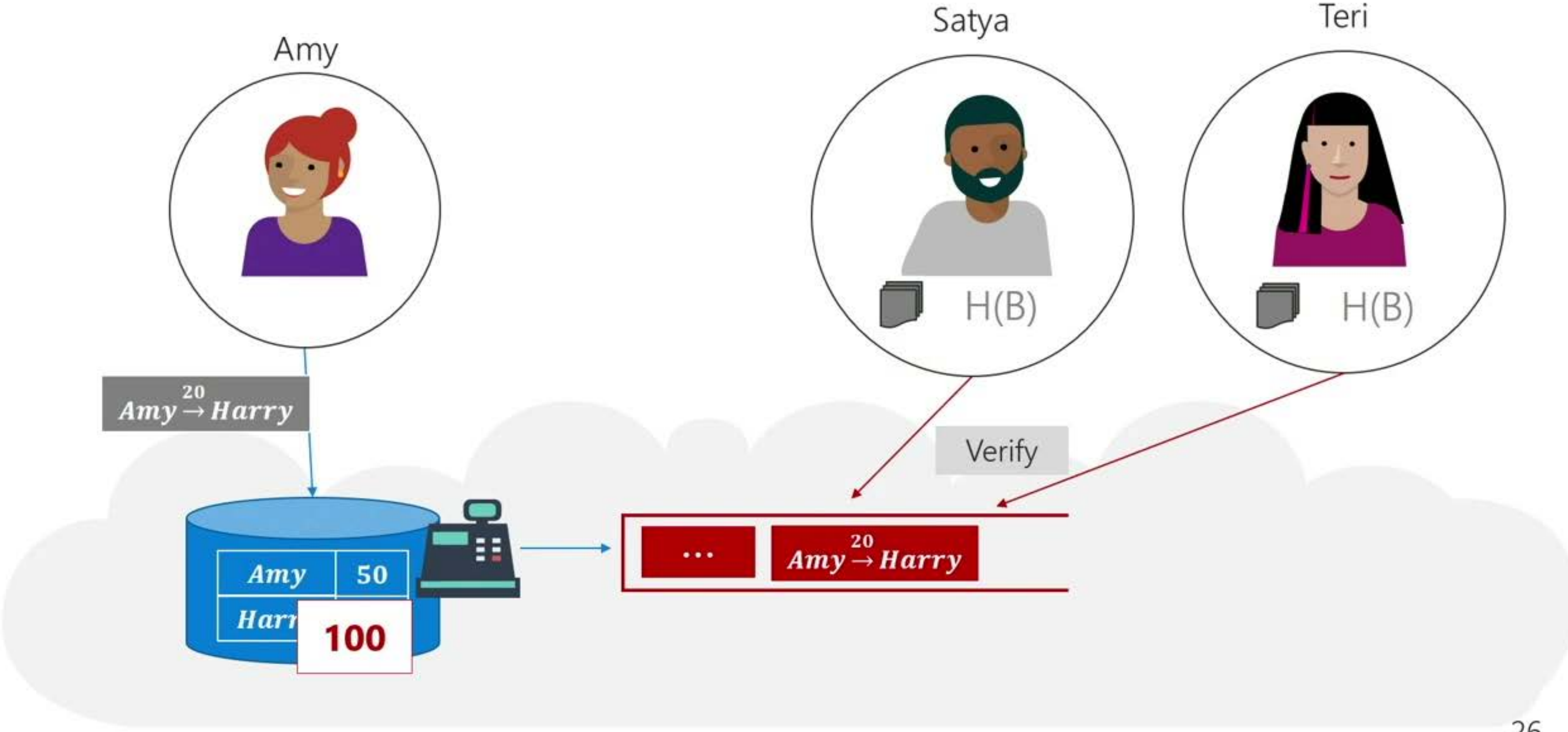
VERITAS


Amy
Satya
Teri
Receipt: 01011
H(B)
H(B)
$Amy \overset{20}{\rightarrow} Harry$
Receipt: 01011
Verify
Amy 50
Harry 70
...
$Amy \overset{20}{\rightarrow} Harry$

VERITAS

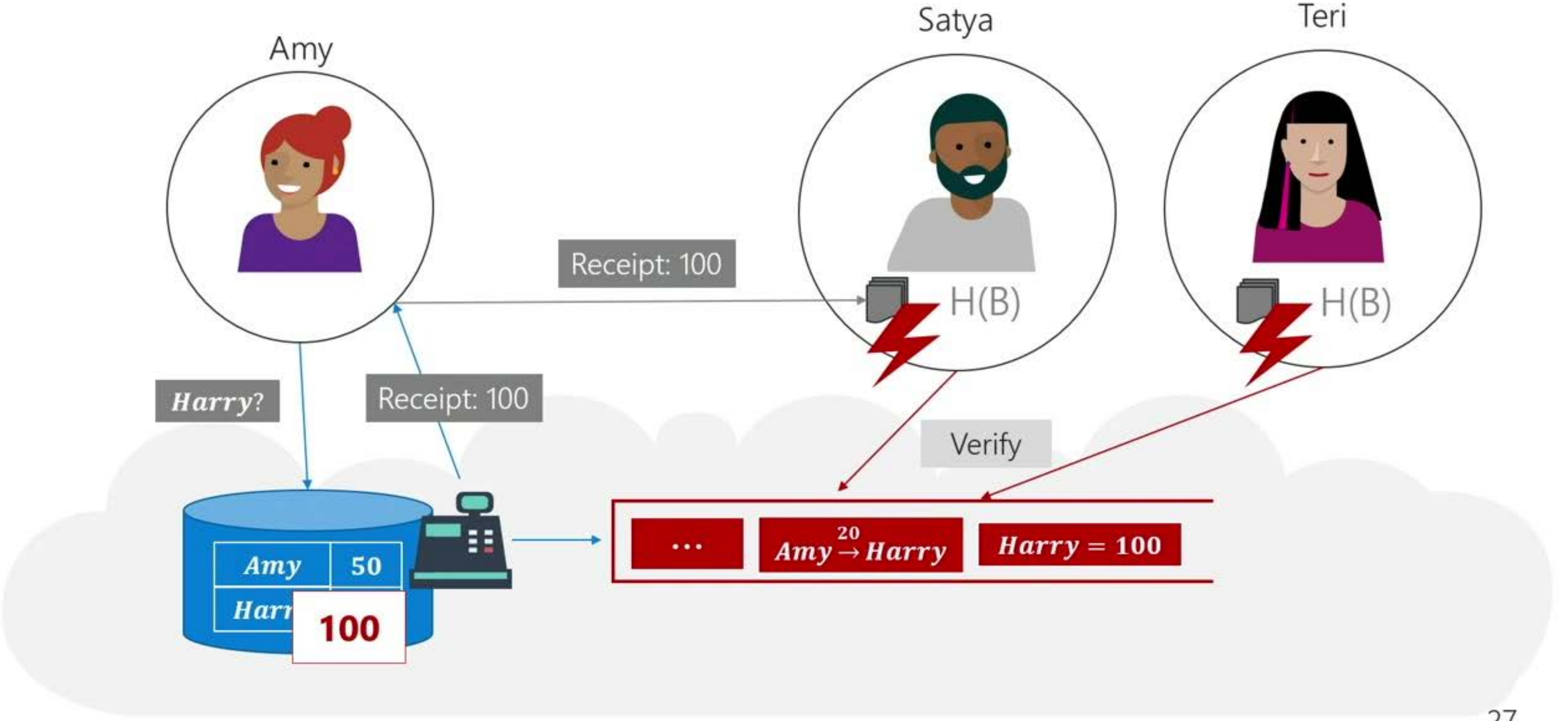


VERITAS



Amy
Satya
Teri
H(B)
H(B)
$Amy \xrightarrow{20} Harry$
Verify
Amy
50
Harr
100
...
$Amy \xrightarrow{20} Harry$

VERITAS



Amy
Satya
Teri
Receipt: 100
H(B)
H(B)
Harry?
Receipt: 100
Verify
Amy 50
Harry 100
…
$Amy \xrightarrow{20} Harry$
$Harry = 100$

Example: Decentralized IDs

Acknowledgments:

Daniel Buchner, Esha Ghosh, Rahee Ghosh, Ankur Jain, Srinath Setty, Henry Tsai

"On the Internet, nobody knows you're a dog."

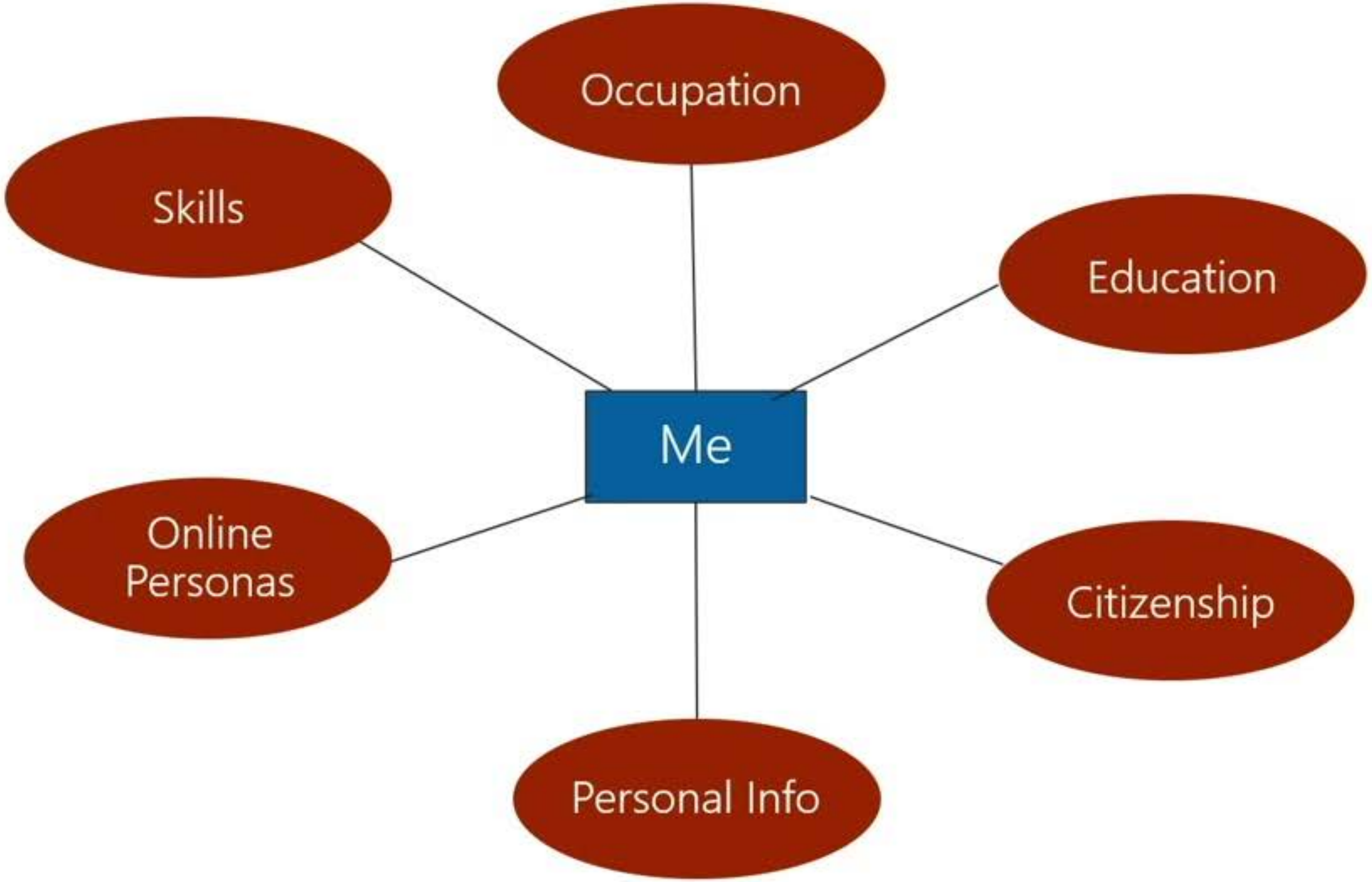
DID: Decentralized (Digital) Identifiers

Transactions in the Real World

- **All Transactions require Proof: Witnesses and/or Receipts**
  - getting married (best man + ring)
  - buying a house (notary + contract)
  - drinking alcohol (driver's licence)

- **Why do we need Proof?**
  - transactions have *conditions* and come with *rights* & *accountabilities*
    - getting married: *„I am married to you! Please, be nice to me!“*
    - buying a house: *„I am the rightful owner of the house! I am allowed to live here.“*
    - birth: *„I become Donald Kossmann.“* - drinking: *„I am Donald Kossmann!“*
  - Witnesses and receipts provide *proof*: *Proof = Trust*

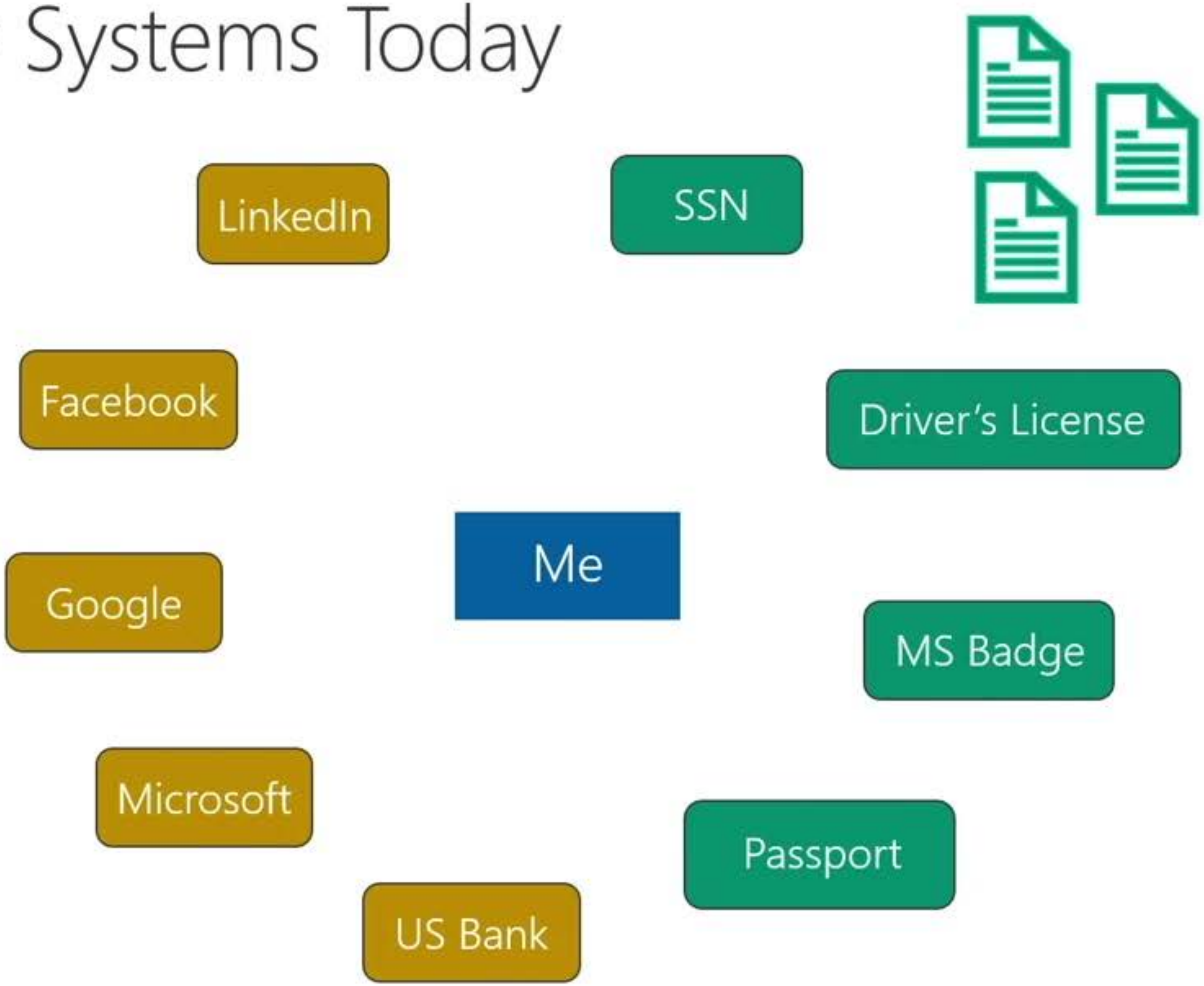
Identity Problem



Claims and Credentials

- Enter the country
    - Citizenship
- Drive a vehicle
    - Driving skills
- Drink alcohol
    - Age
- Enter this building
    - Work @ MS

Identity Systems Today



Diplomas
Birth certificate
...

Identity Systems Today: Challenges

- Lack of ownership and control over identifiers
  - Centralized root of trust
- Patchwork of multiple identifiers
  - Management complexity
  - Integration complexity (e.g., Mint)
- Non-cryptographic "proofs" of claims
  - Identity theft
- Privacy
  - Example: Establishing my age with DL reveals my location

Decentralized Identifiers (DIDs)

A self-owned identity which can be used to securely and privately store all elements of our identity and establish claims and credentials.

Decentralized Identity Foundation (DIF)

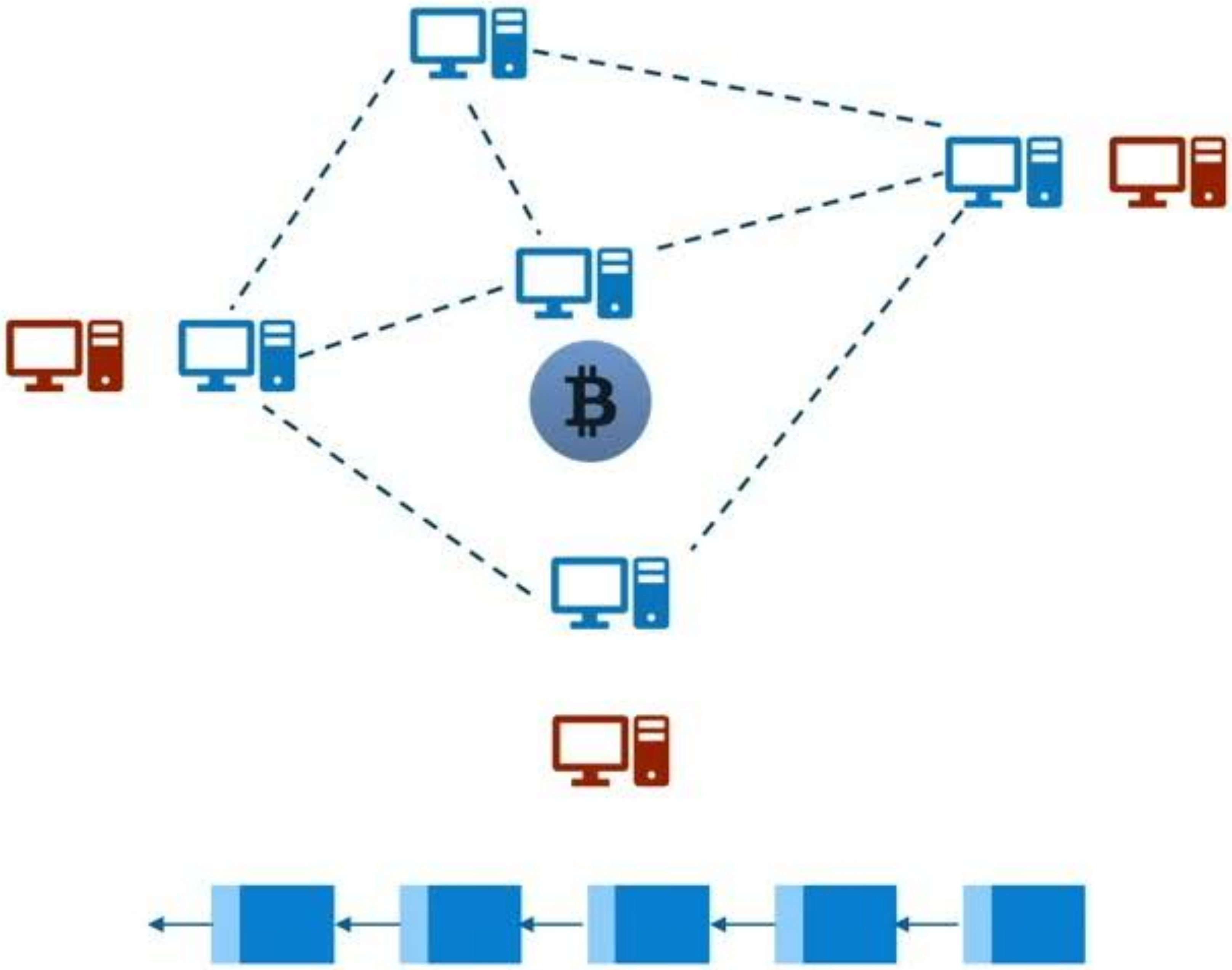
uport
BLOCKSTACK
sovrin
Microsoft
IBM
LEARN ING MA CHINE
HYPERLEDGER
civic
accenture
DANUBE TECH
RSA
evernym
mooti
aetna
r3.
authenteq
BLOCKCHAIN FOUNDRY
Validated ID
EQUINIX
Ockam
NuID
Meta
GS1
DIID
nuggets
TRUSTED KEY
ZINC
ONTology
LUXE
VERIDIUMID
pillar.
dominode
enigma
JOLOCOM
datum
onfido
Auth0
1KOSMOS BlockID
GAMECREDITS
SECURE KEY
lifeID
CENTINAL
IDENTOS
diwala
<sitekit>
Juru
remme
TRANSMUTE
Finema
GATACA
WeBank
微众银行

Decentralized Identifiers (DIDs)

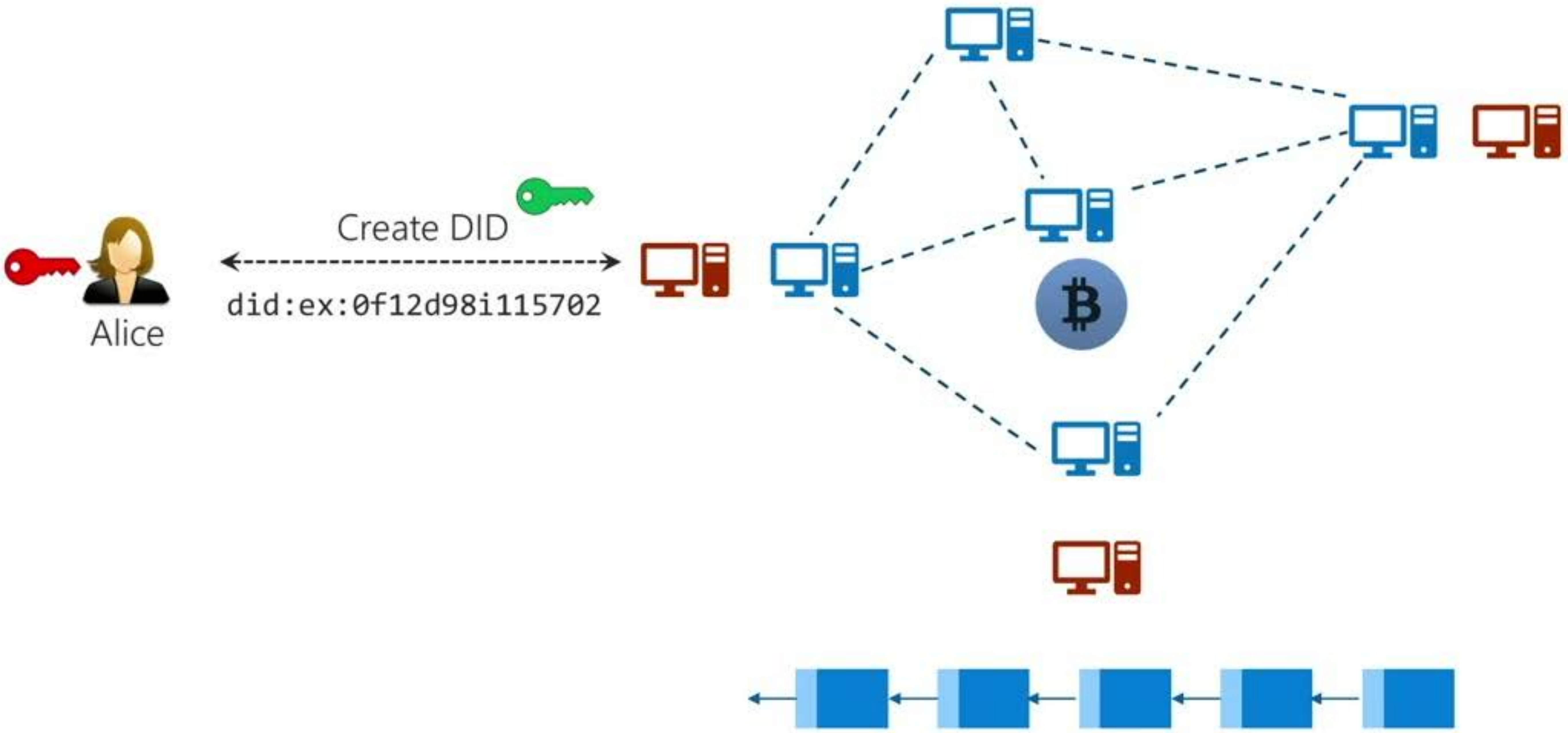
A self-owned identity which can be used to securely and privately store all elements of our identity and establish claims and credentials.

Sidetree DID protocol

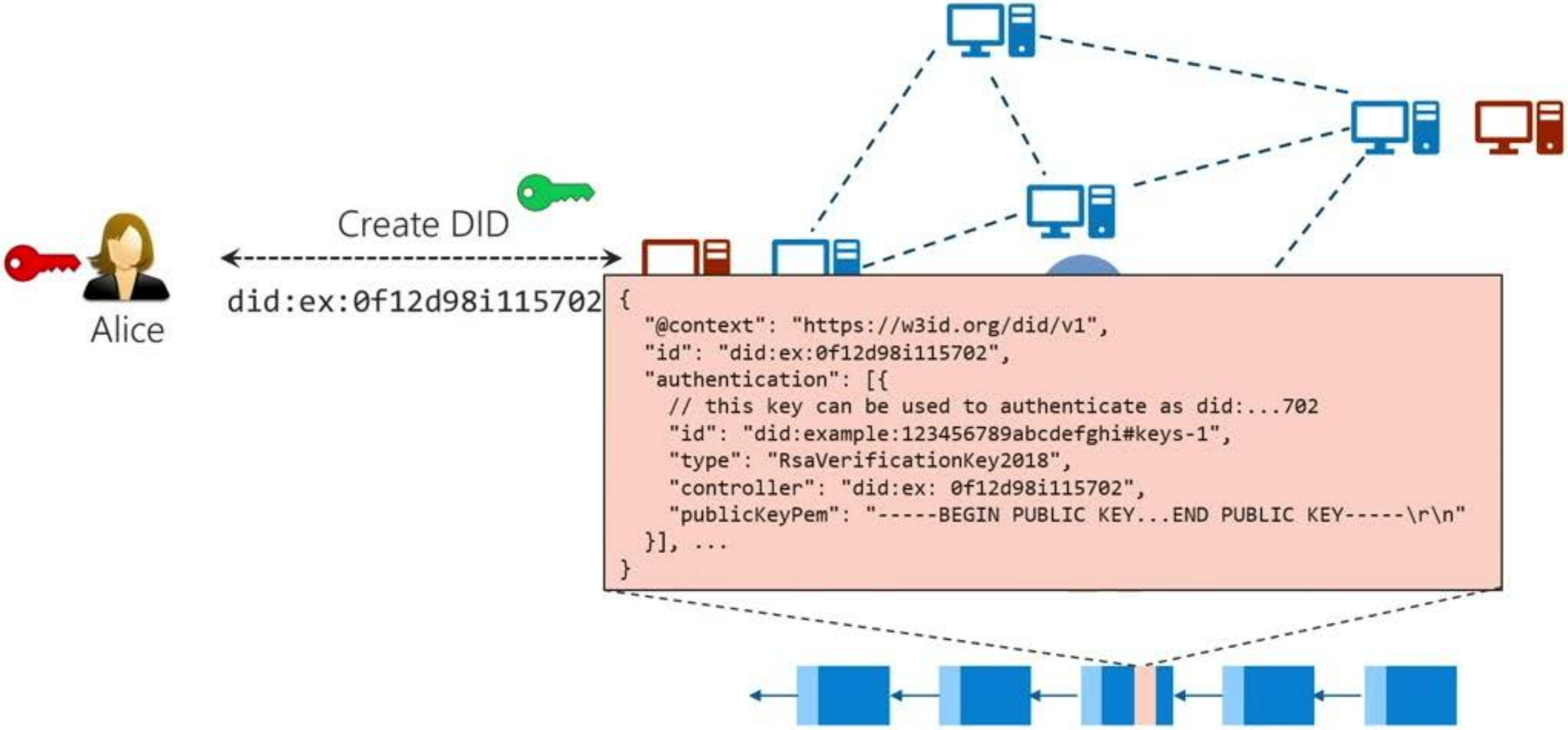
Sidetree DID protocol



Sidetree DID protocol

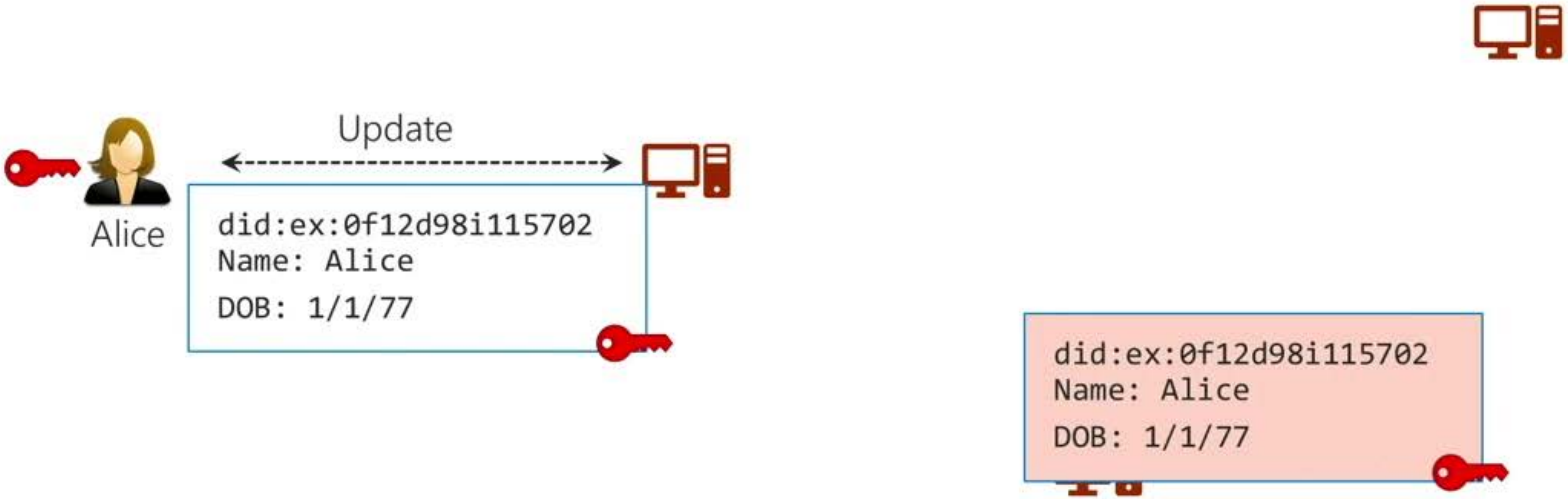


Sidetree DID protocol



```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:ex:0f12d98i115702",
  "authentication": [{
    // this key can be used to authenticate as did:...702
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:ex: 0f12d98i115702",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }], ...
}
```

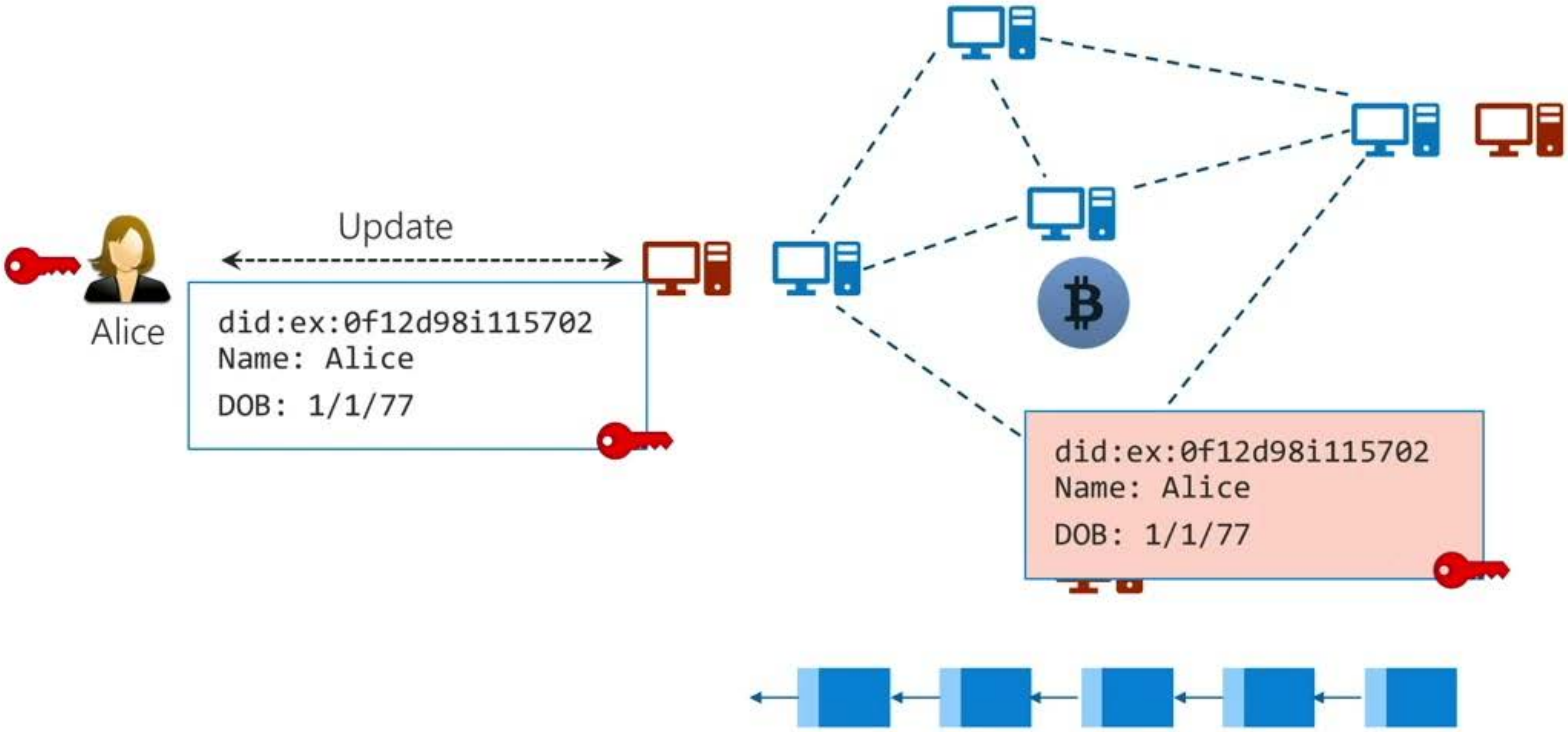
DID-based Claims



```
did:ex:0f12d98i115702
Name: Alice
DOB: 1/1/77
```

```
did:ex:0f12d98i115702
Name: Alice
DOB: 1/1/77
```

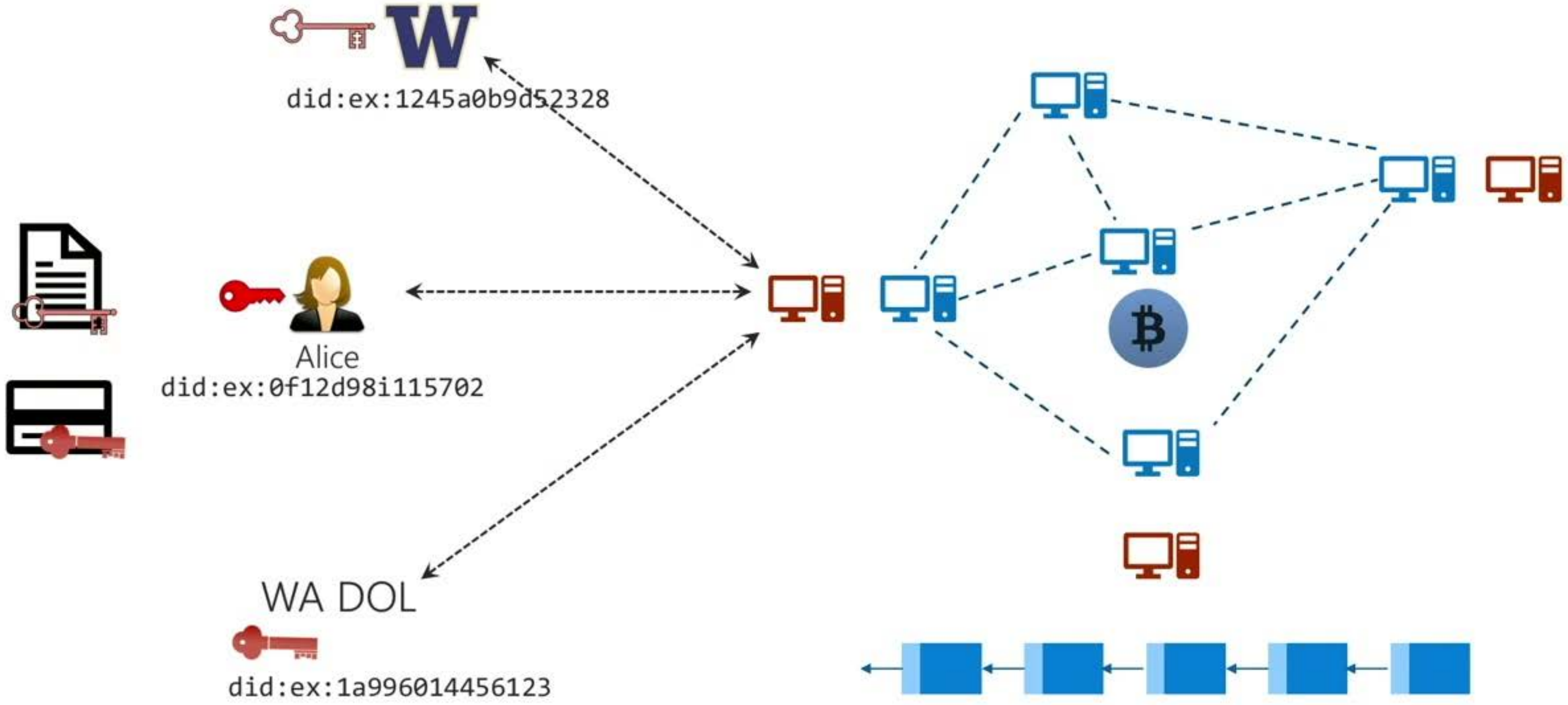
DID-based Claims



did:ex:0f12d98i115702
Name: Alice
DOB: 1/1/77

did:ex:0f12d98i115702
Name: Alice
DOB: 1/1/77

DID-based Claims



DID-based claims: Proofs

sk_{Alice}

Alice

Bob the Barman

WA USA WASHINGTON

DRIVER LICENSE

4d LIC# SAMPJA185800

1 SAMPLE

2 JOHN A

DONOR

3 DOB 01/06/1958

4a ISS 01/06/2015

8 123 STREET ADDRESS
YOUR CITY WA 99999-0000

15 SEX M

16 HGT 5'-08"

12 RESTRICTIONS
NONE

18 EYES BLU

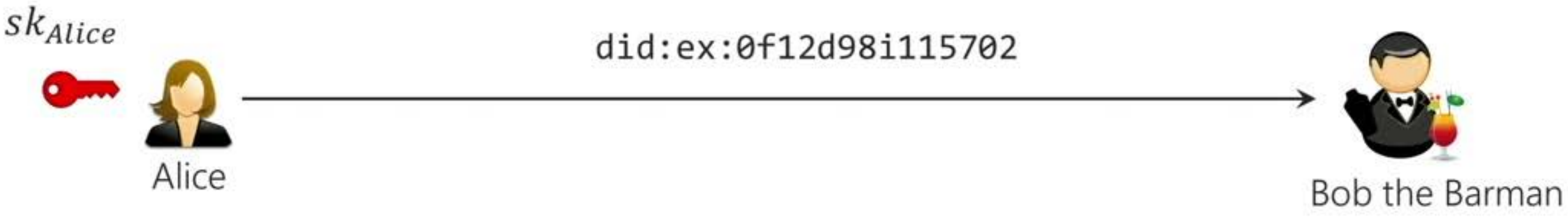
17 WGT 165

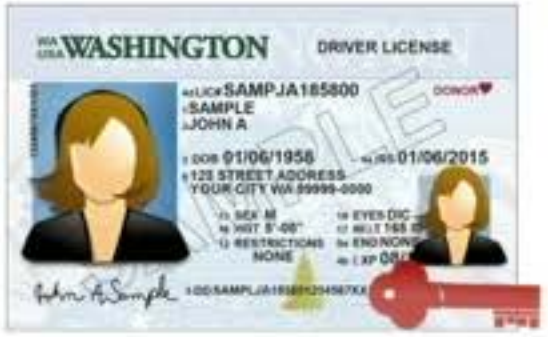
9a END NONE

4b EXP 08/

5 DD SAMPLJA185

DID-based claims: Proofs





WA USA WASHINGTON DRIVER LICENSE

4d LIC# SAMPJA185800 DONOR♥

1 SAMPLE

2 JOHN A

3 DOB 01/06/1958 4a ISS 01/06/2015

8 123 STREET ADDRESS
YOUR CITY WA 99999-0000

15 SEX M 18 EYES BLU

16 HGT 5'-08" 17 WGT 165 lb

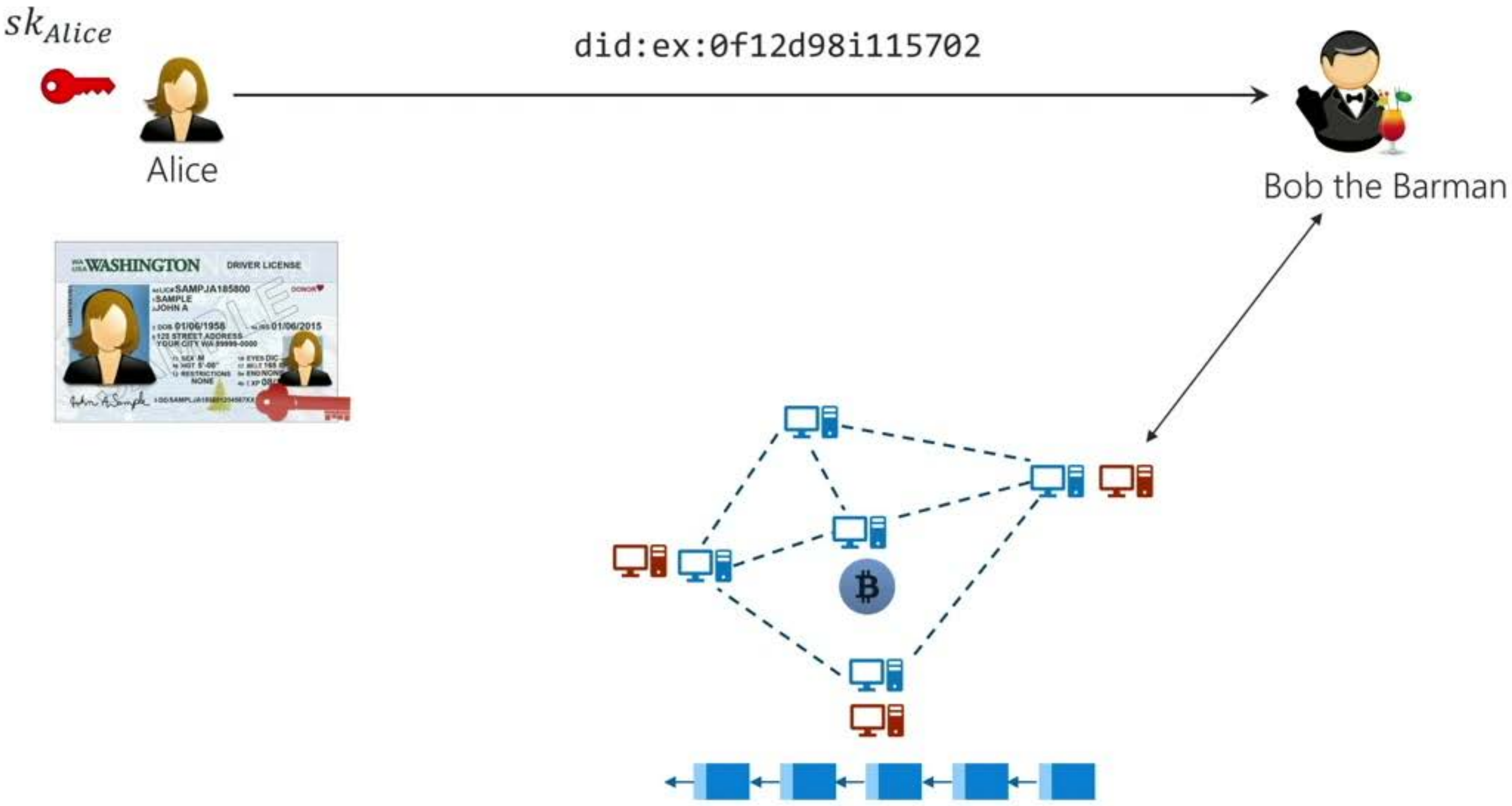
12 RESTRICTIONS NONE 9a END NONE

4b EXP 08/

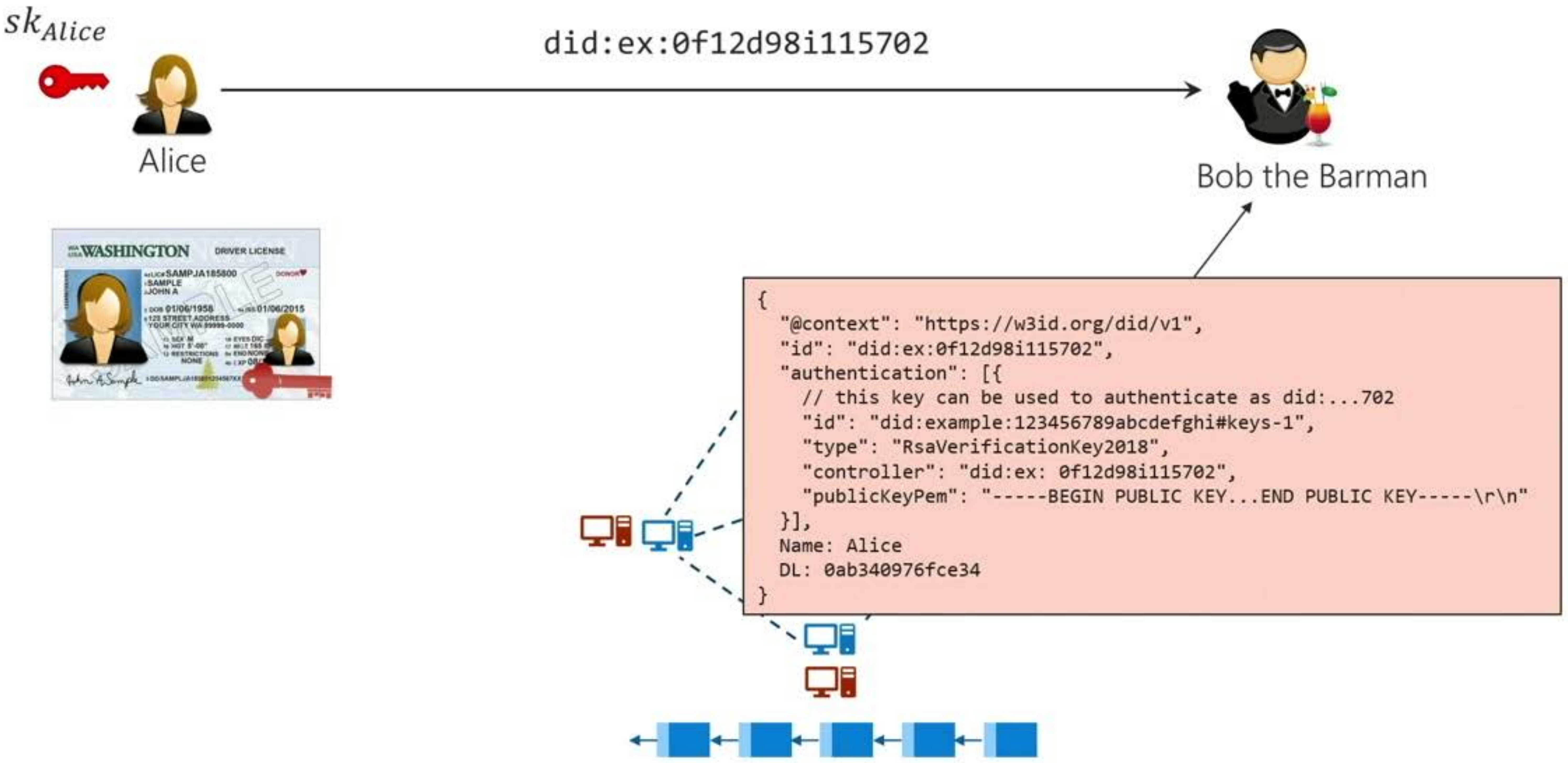
5 DD SAMPLJA185801234567XX

John A Sample

DID-based claims: Proofs

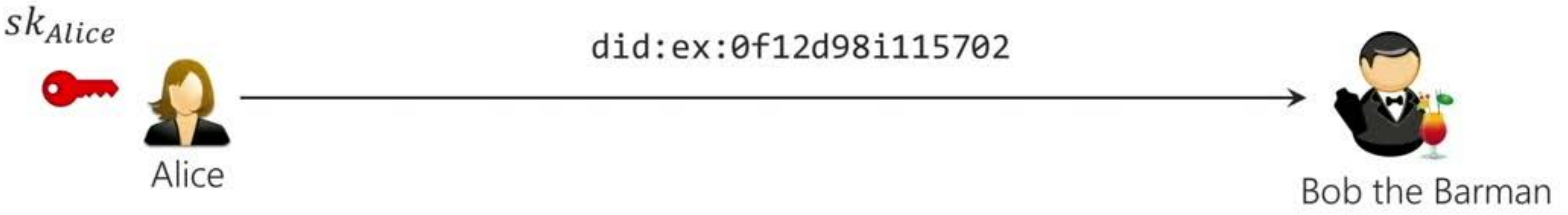


DID-based claims: Proofs



```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:ex:0f12d98i115702",
  "authentication": [{
    // this key can be used to authenticate as did:...702
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "controller": "did:ex: 0f12d98i115702",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],
  Name: Alice
  DL: 0ab340976fce34
}
```

DID-based claims: Proofs



sk_{Alice}

Alice

`did:ex:0f12d98i115702`

Bob the Barman

WA USA WASHINGTON DRIVER LICENSE

4d LIC# SAMPJA185800

1 SAMPLE

2 JOHN A

DONOR♥

3 DOB 01/06/1958

4a ISS 01/06/2015

8 123 STREET ADDRESS
YOUR CITY WA 99999-0000

15 SEX M

16 HGT 5'-08"

12 RESTRICTIONS
NONE

18 EYES BLU

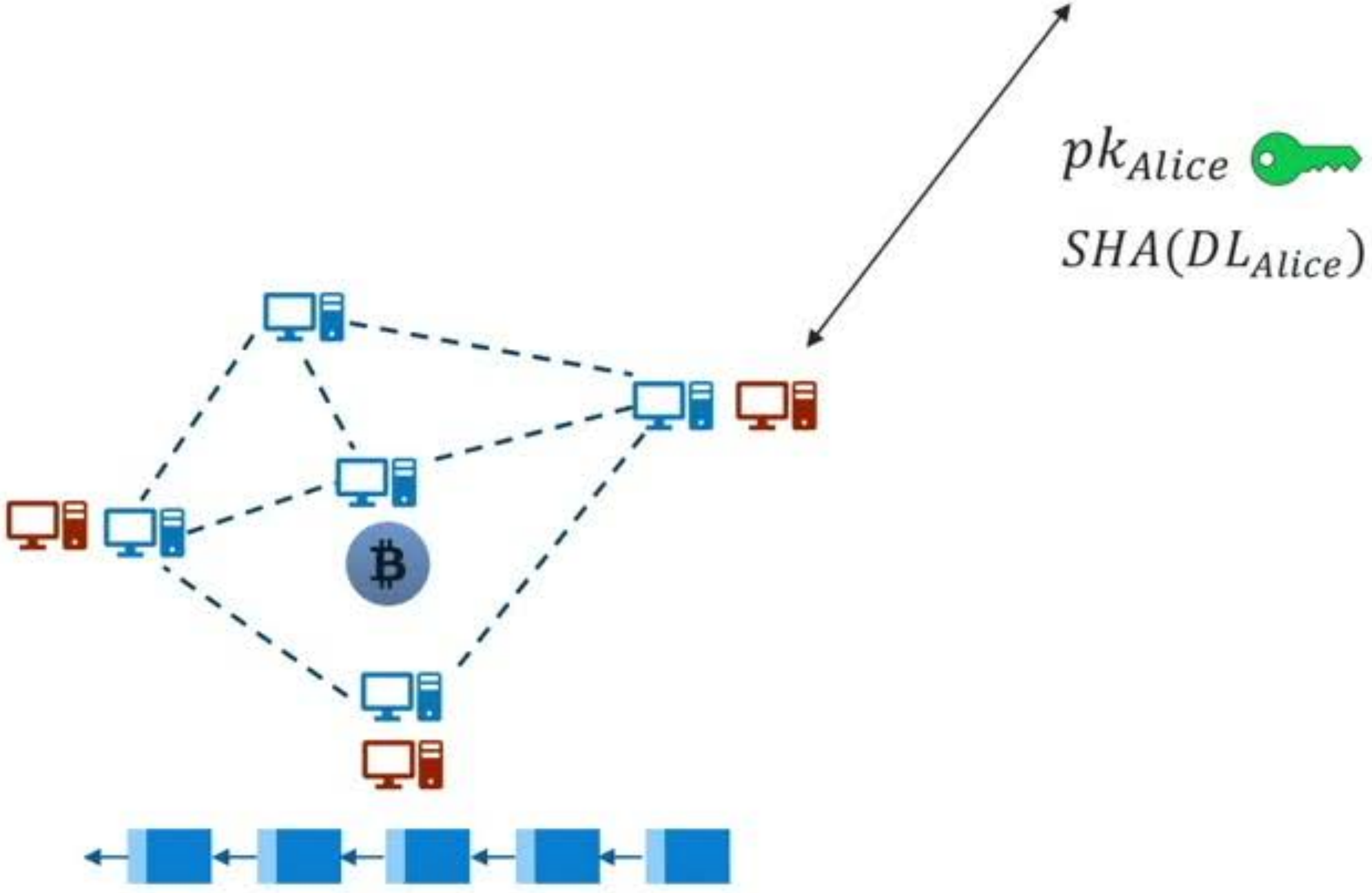
17 WGT 165

9a END NONE

4b EXP 08/

John A Sample

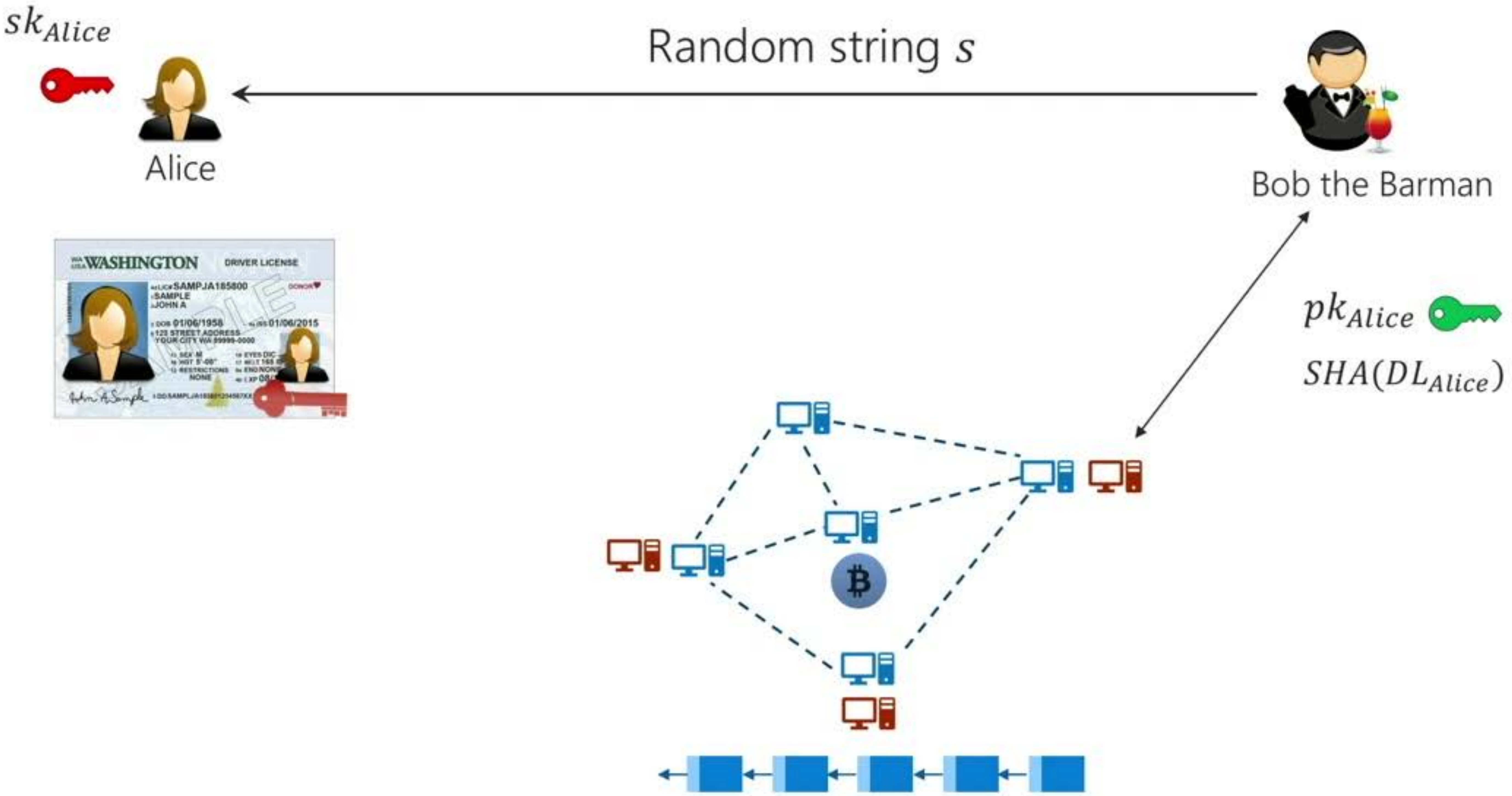
5 DD SAMPLJA1958



pk_{Alice}

$SHA(DL_{Alice})$

DID-based claims: Proofs



sk_{Alice}

Alice

Random string s

Bob the Barman

pk_{Alice}

$SHA(DL_{Alice})$

WA USA WASHINGTON DRIVER LICENSE

LIC# SAMPJA185800

SAMPLE

JOHN A

DOB 01/06/1958

ISS 01/06/2015

123 STREET ADDRESS

YOUR CITY WA 99999-0000

SEX M

HGT 5'-08"

RESTRICTIONS NONE

EYES DIC

WGT 165

END NONE

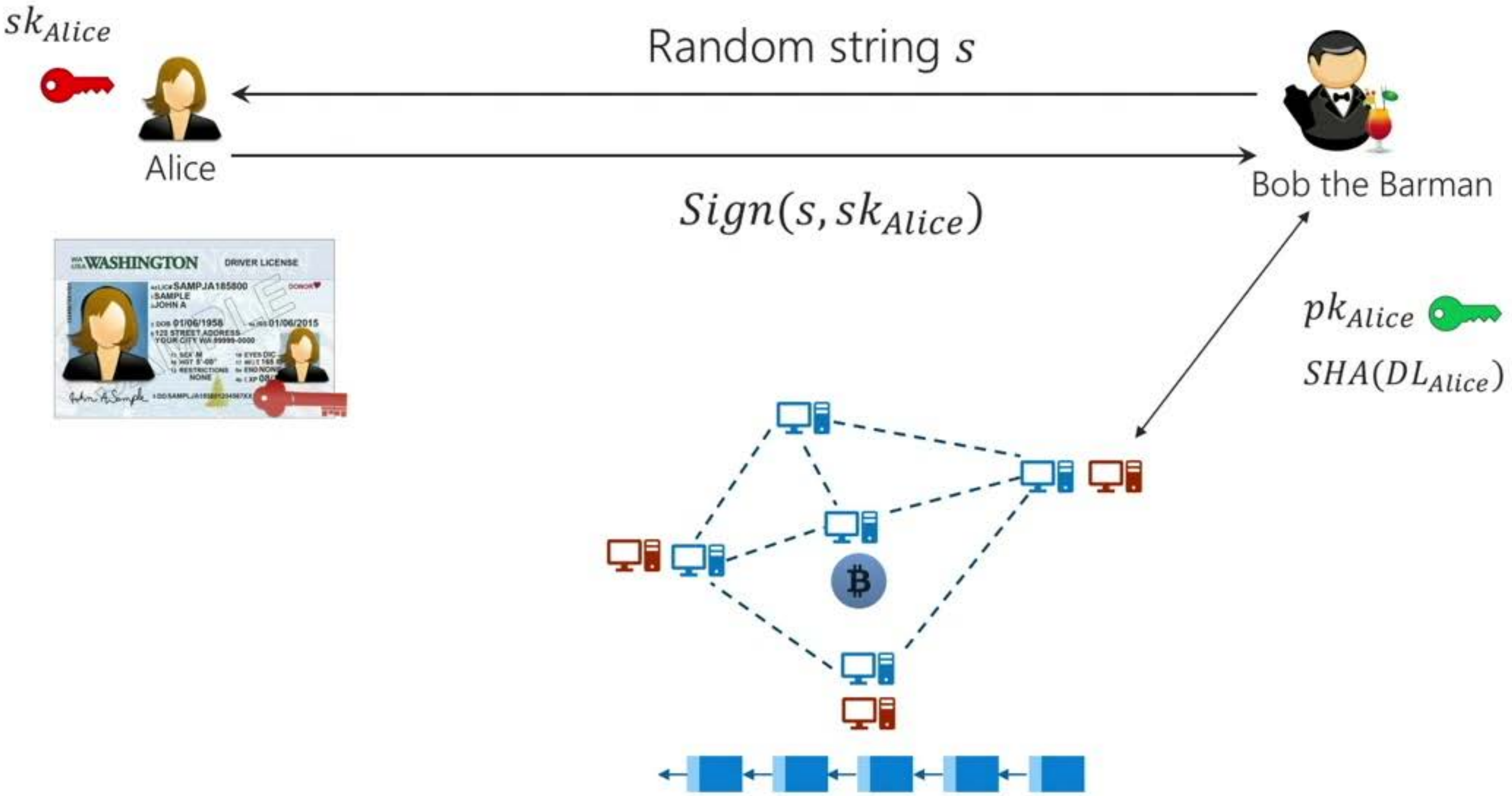
EXP 08

DONOR

SAMPLE

John A Sample

DID-based claims: Proofs



sk_{Alice}

Alice

Random string s

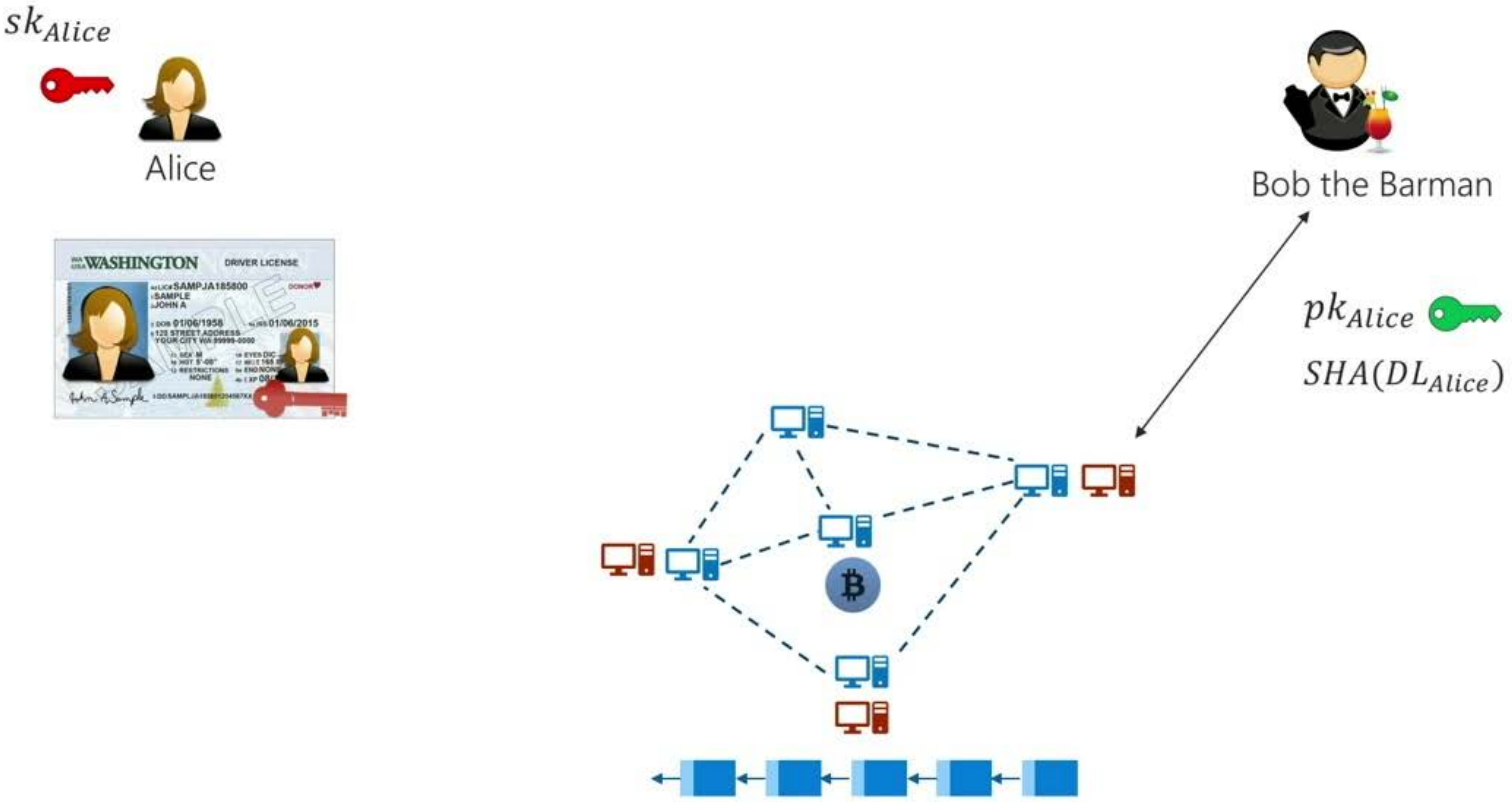
$Sign(s, sk_{Alice})$

Bob the Barman

pk_{Alice}

$SHA(DL_{Alice})$

DID-based claims: Proofs



sk_{Alice}

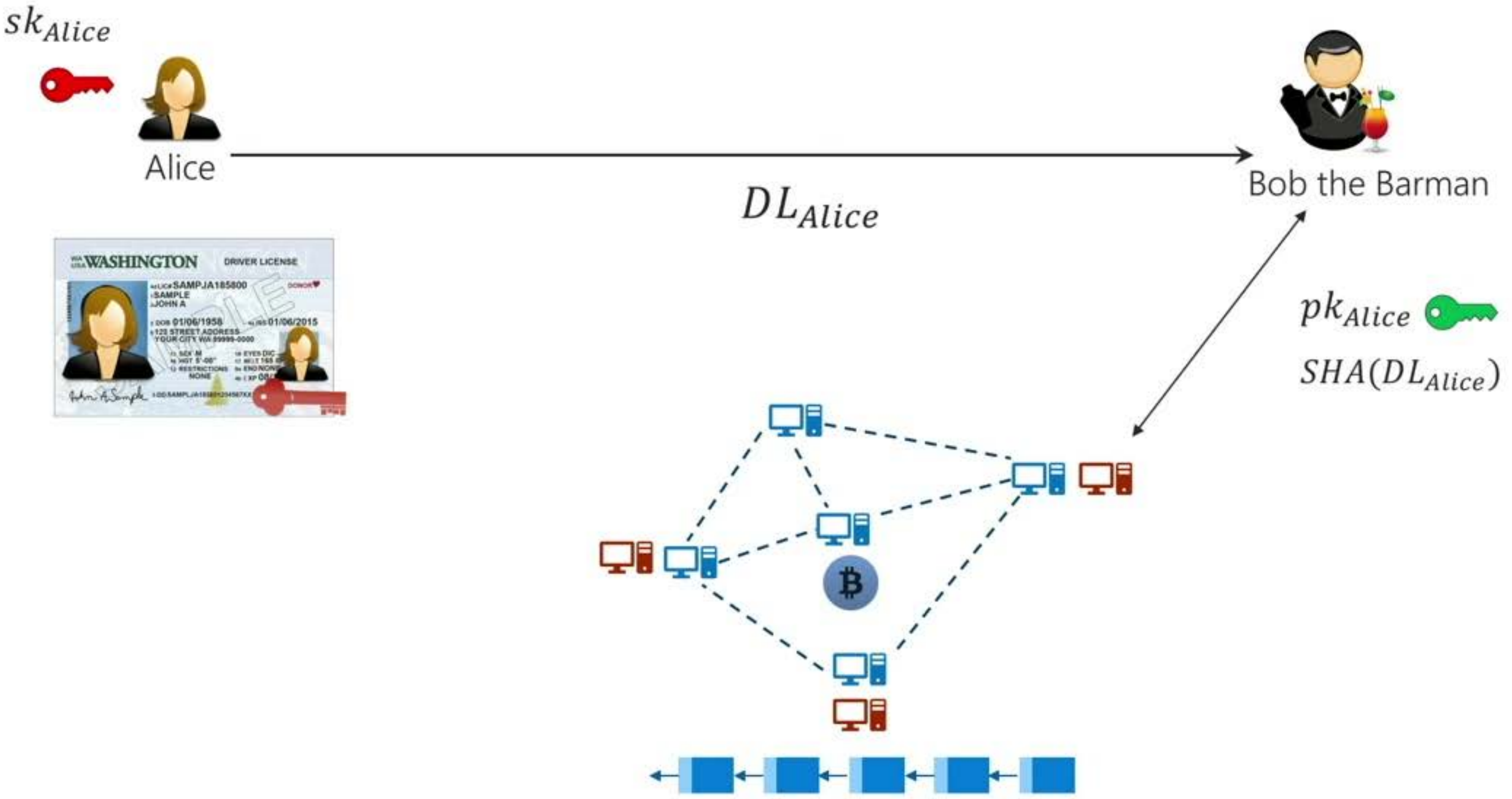
Alice

Bob the Barman

pk_{Alice}

$SHA(DL_{Alice})$

DID-based claims: Proofs



sk_{Alice}

Alice

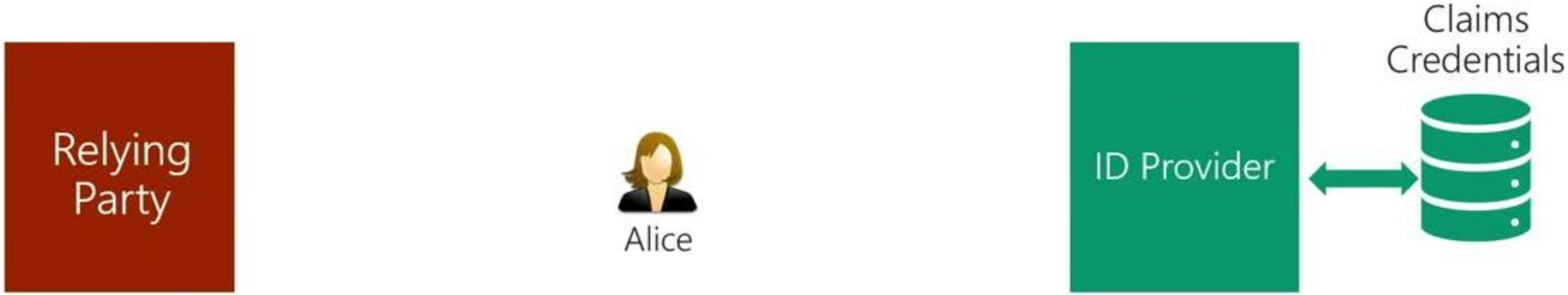
DL_{Alice}

Bob the Barman

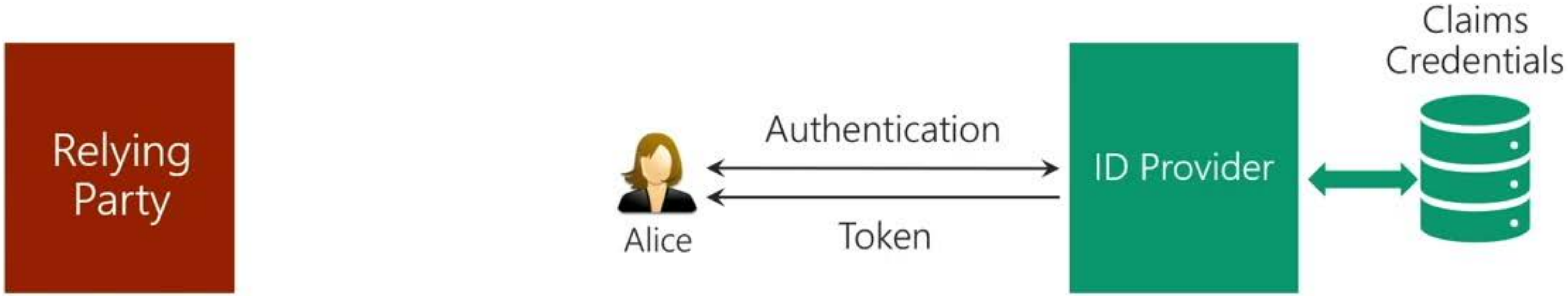
pk_{Alice}

$SHA(DL_{Alice})$

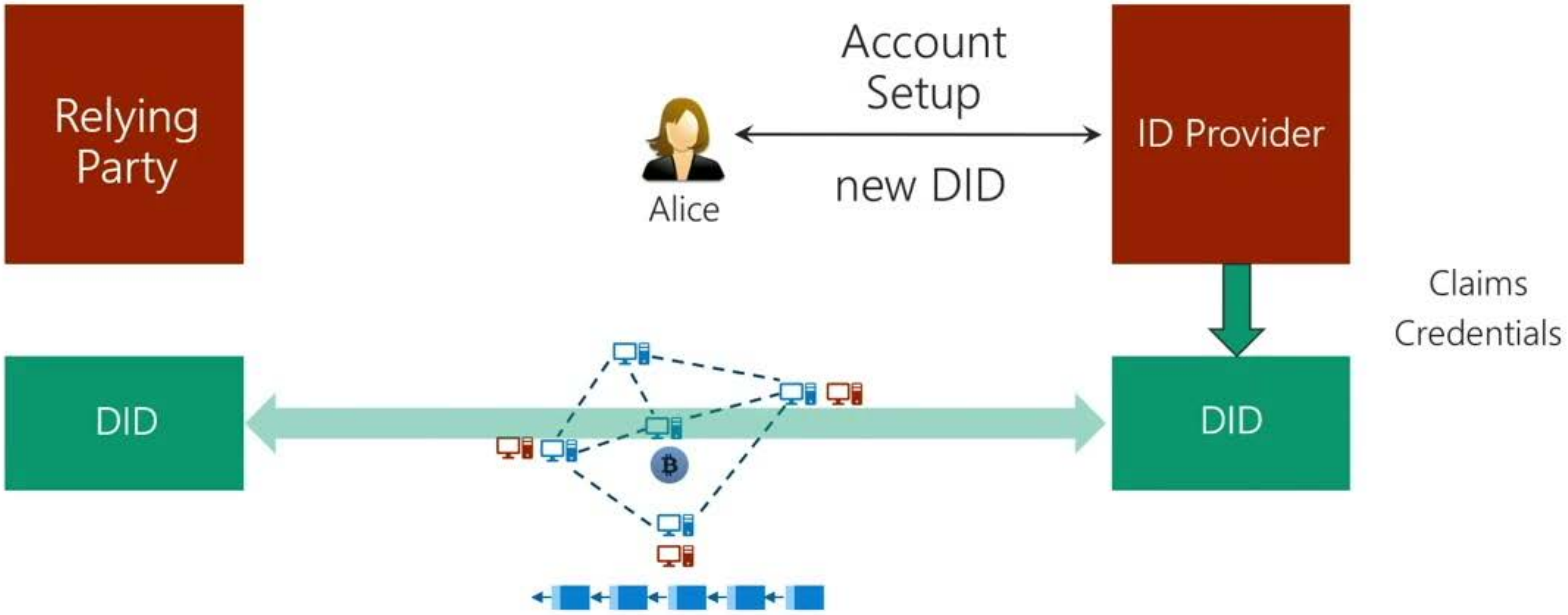
Retrofitting OpenId



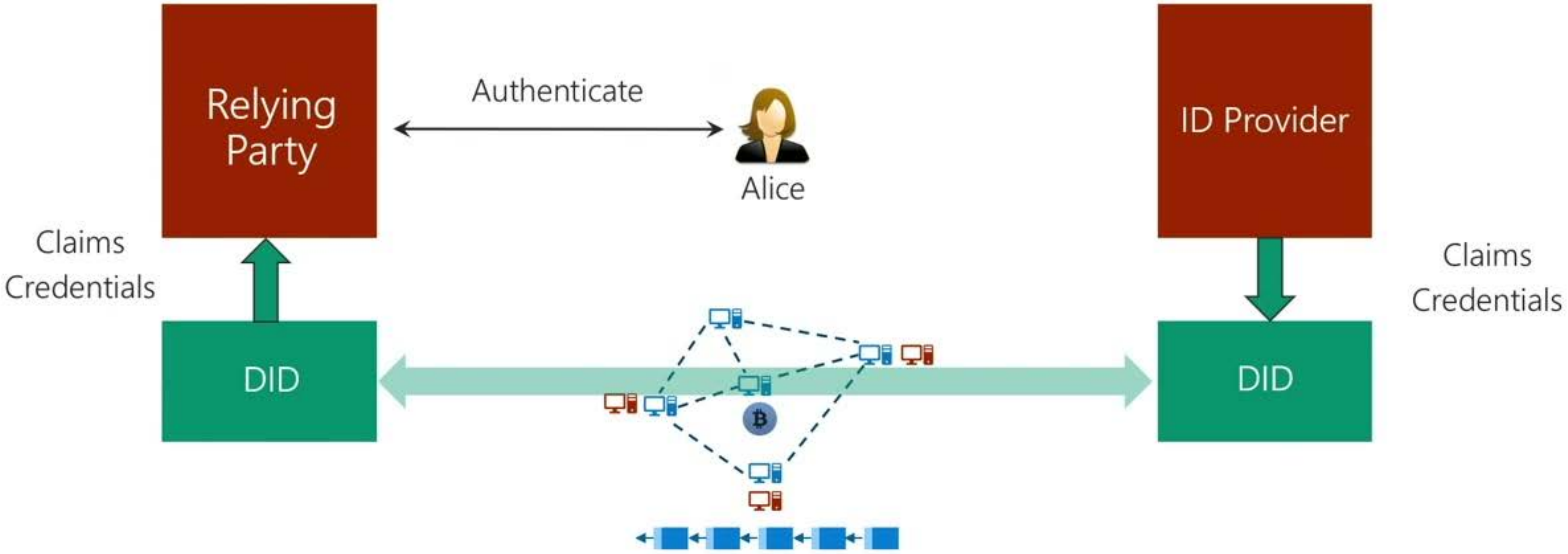
Retrofitting OpenId



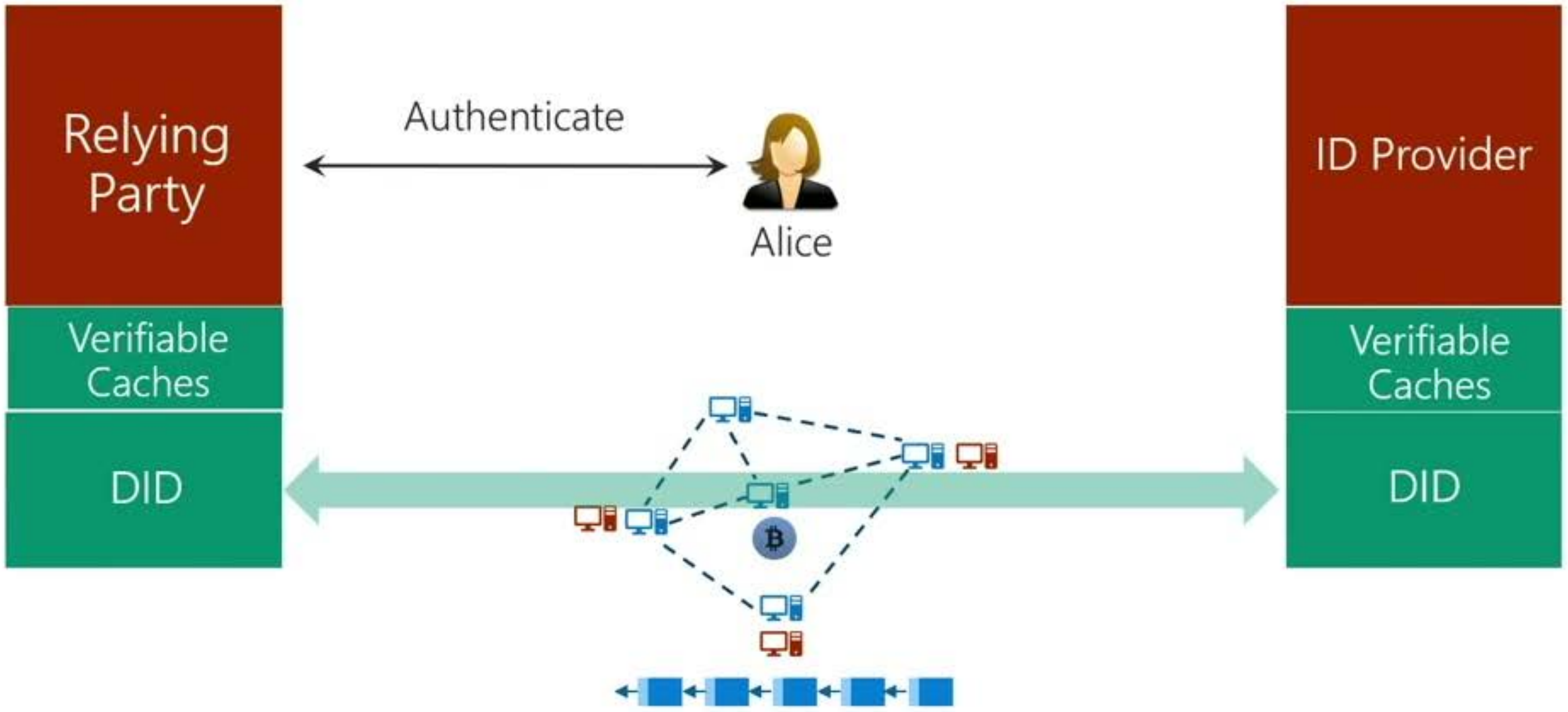
Retrofitting OpenId



Retrofitting OpenId



Retrofitting OpenId



Relying Party

Verifiable Caches

DID

Authenticate

Alice

ID Provider

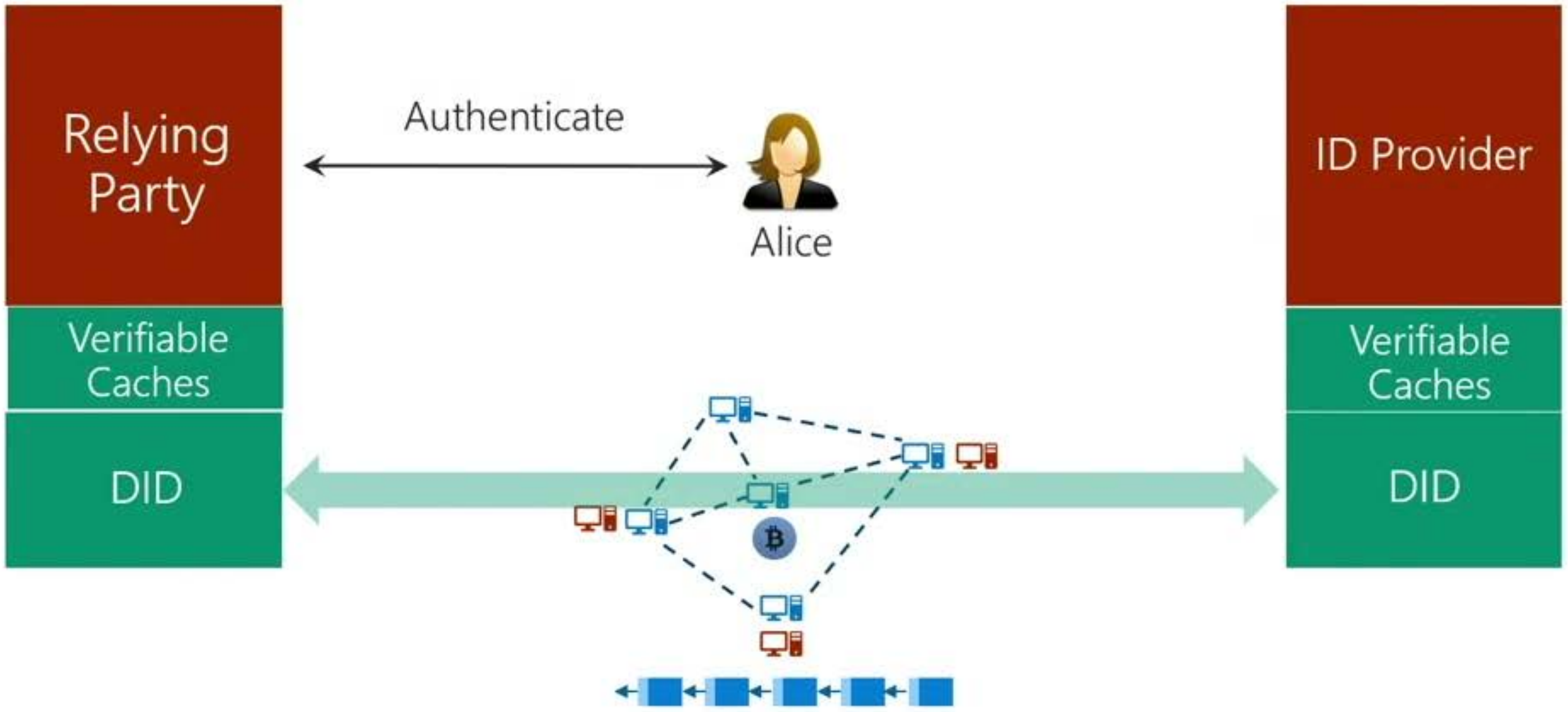
Verifiable Caches

DID

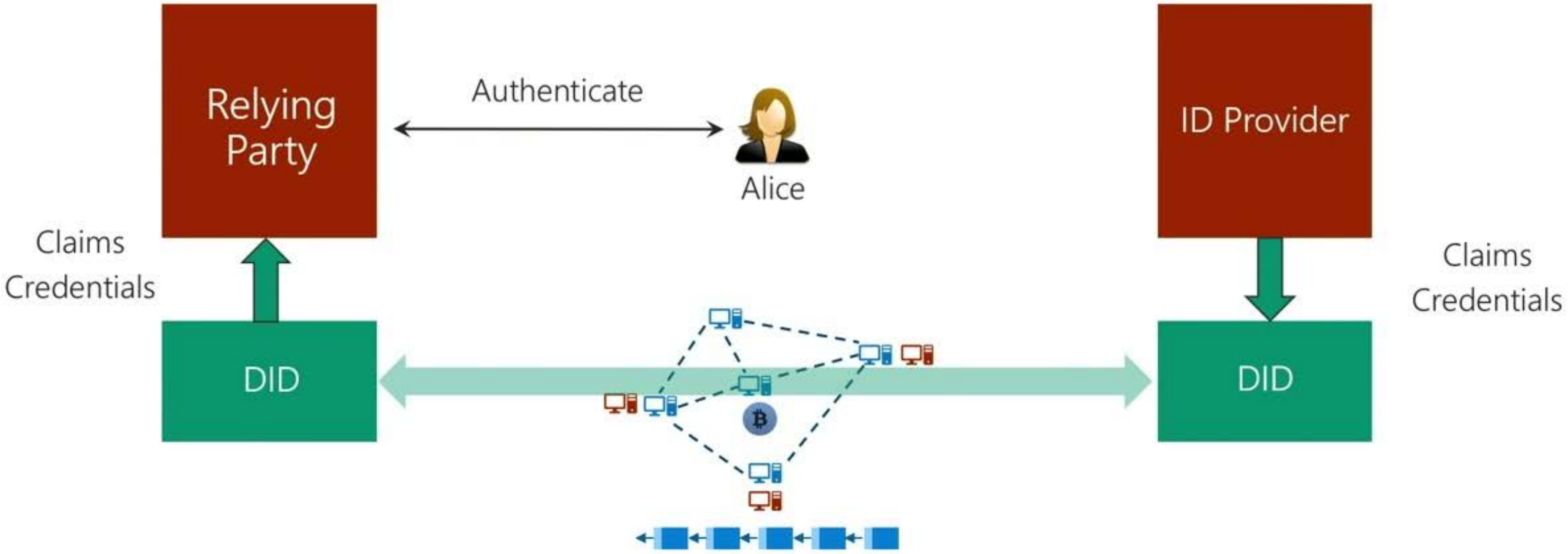
Conclusions

- Blockchains
  - proofs of digital transactions
  - Limitations – abstractions, performance
- Veritas
  - Retrofit verifiability to existing systems
  - Overlay on blockchains for consensus
- Decentralized Ids

Retrofitting OpenId



Retrofitting OpenId



Relying Party

Authenticate

Alice

ID Provider

Claims

Credentials

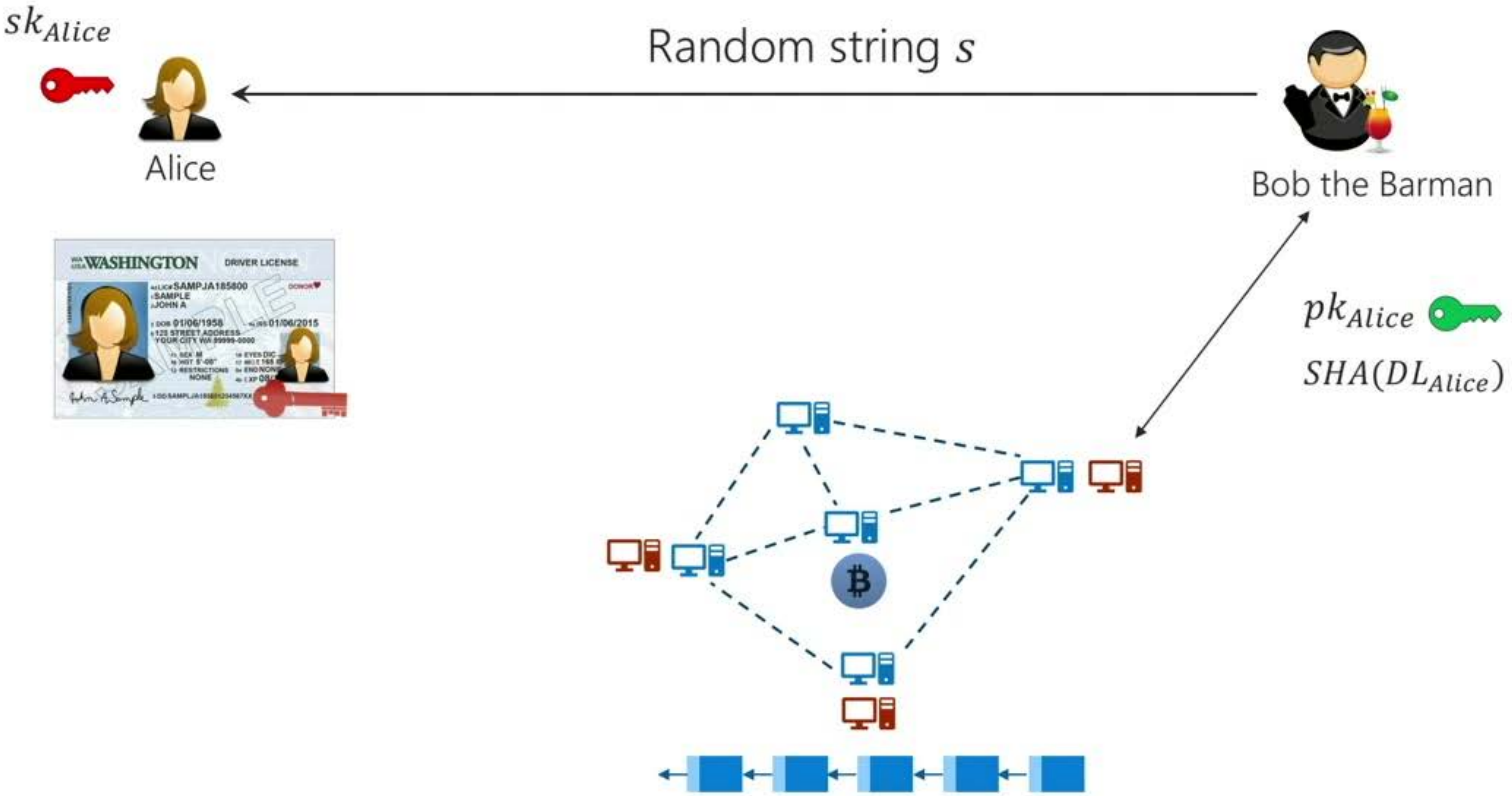
DID

DID

Claims

Credentials

DID-based claims: Proofs



sk_{Alice}

Alice

Random string s

Bob the Barman

pk_{Alice}

$SHA(DL_{Alice})$