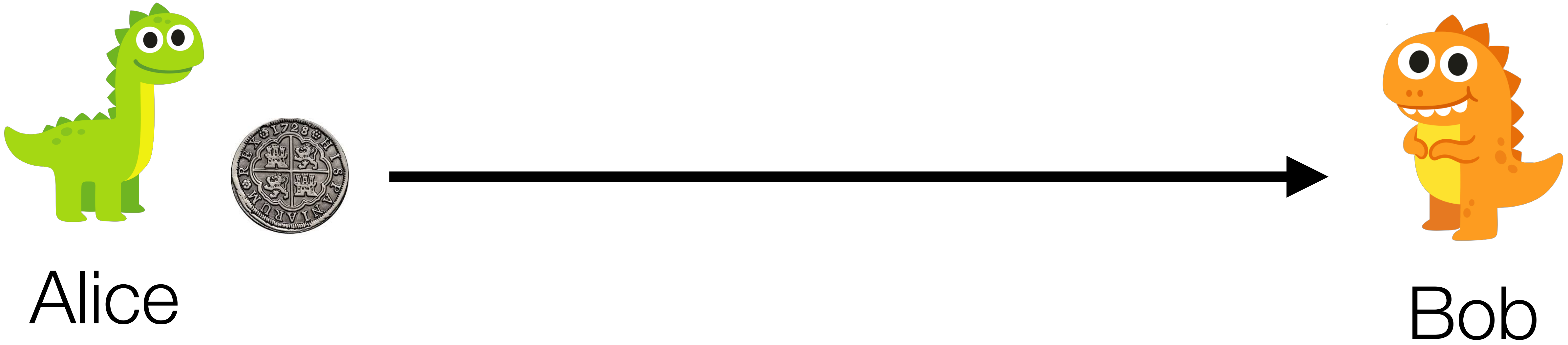
Smart Contracts as Authorized Production Rules

Ben Lippmeier (with Amos Robinson and Andrae Muys)
FP-Syd 2019/4/24

Alice

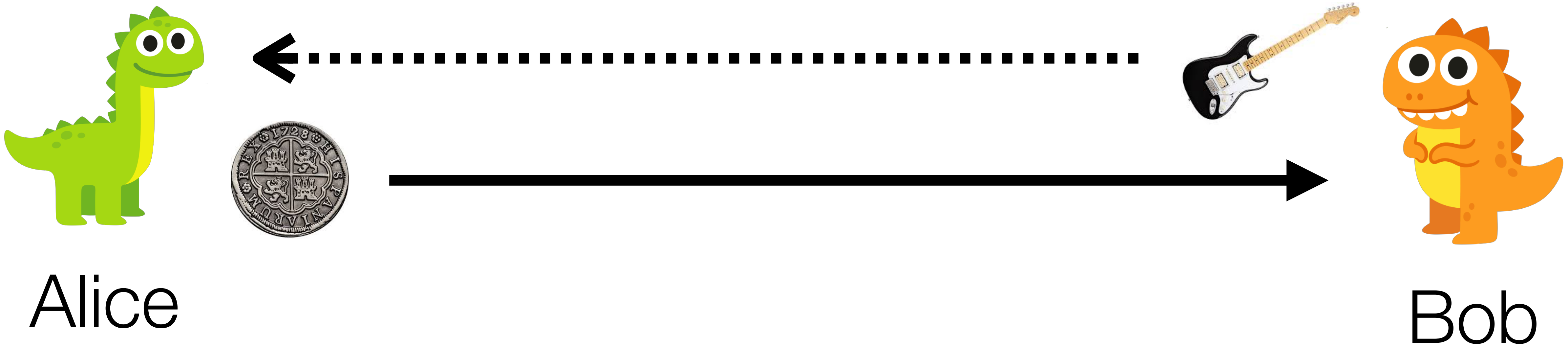
Alice

Bob



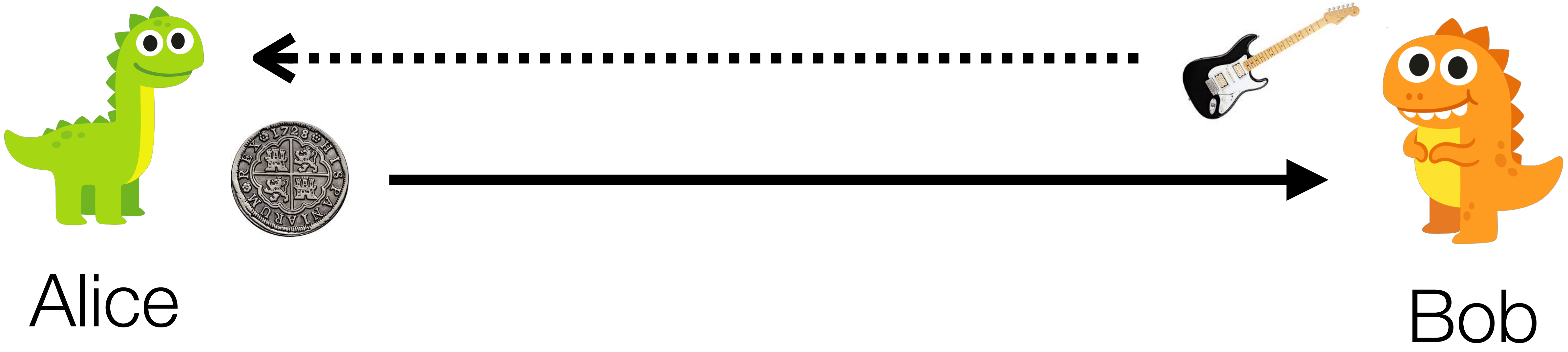
Alice

Bob

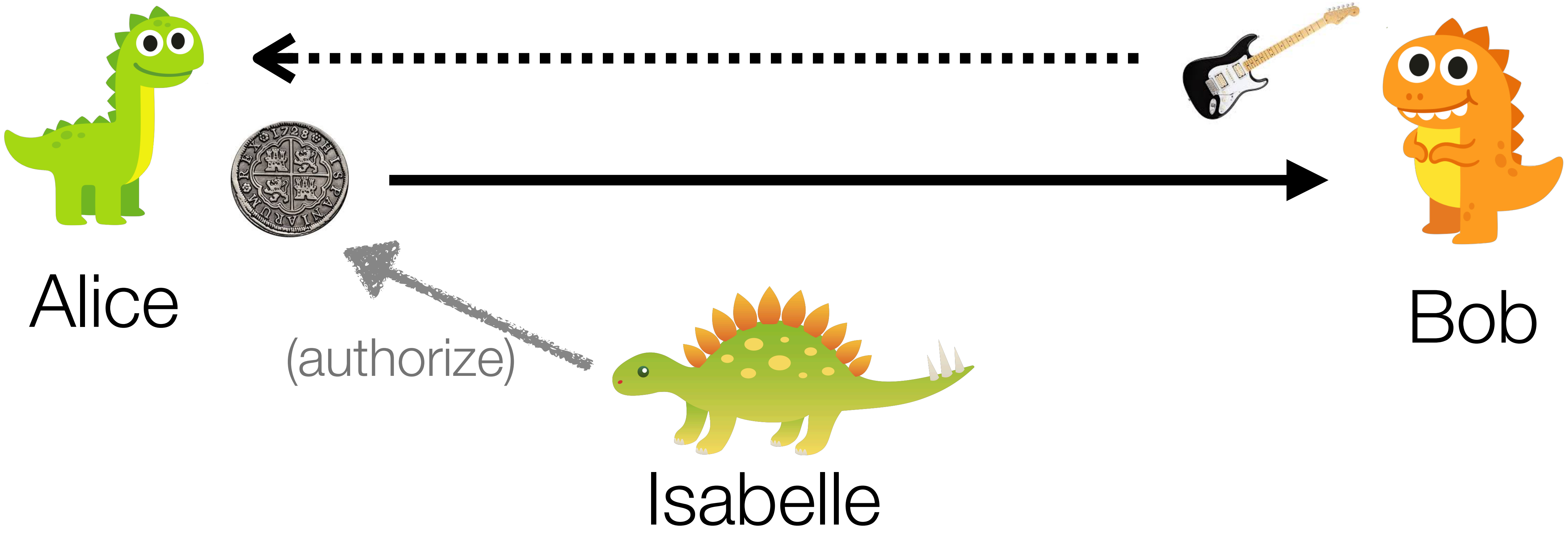


Alice

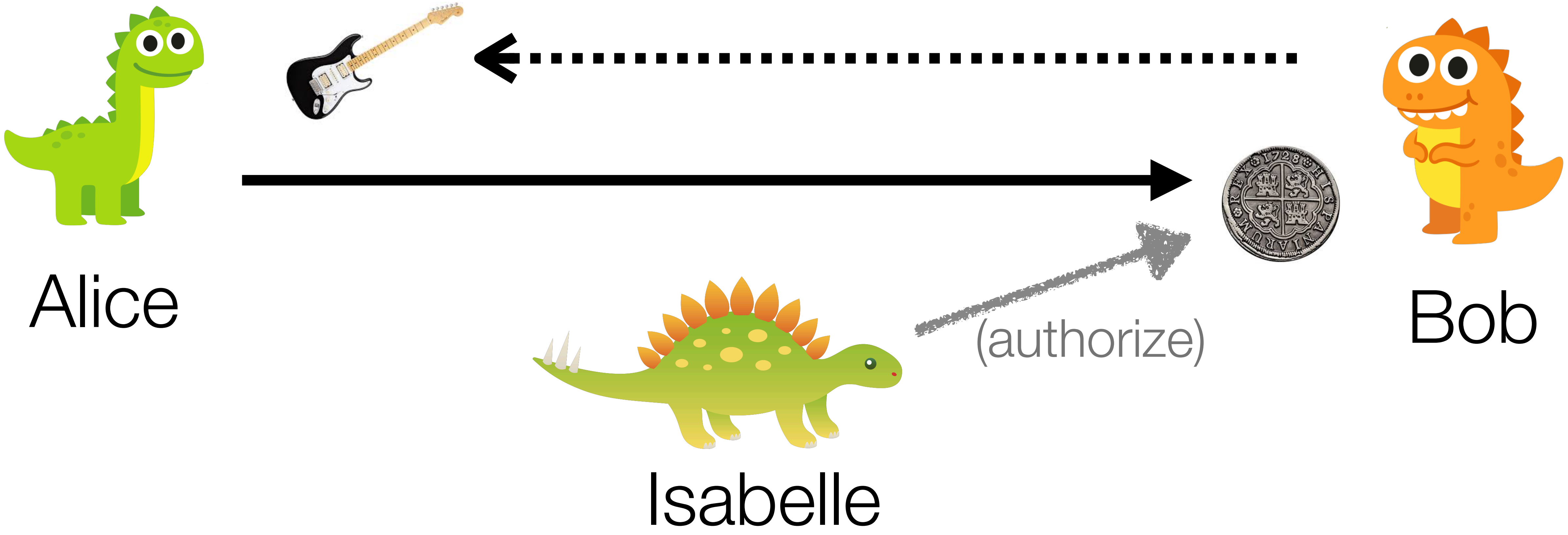
Bob



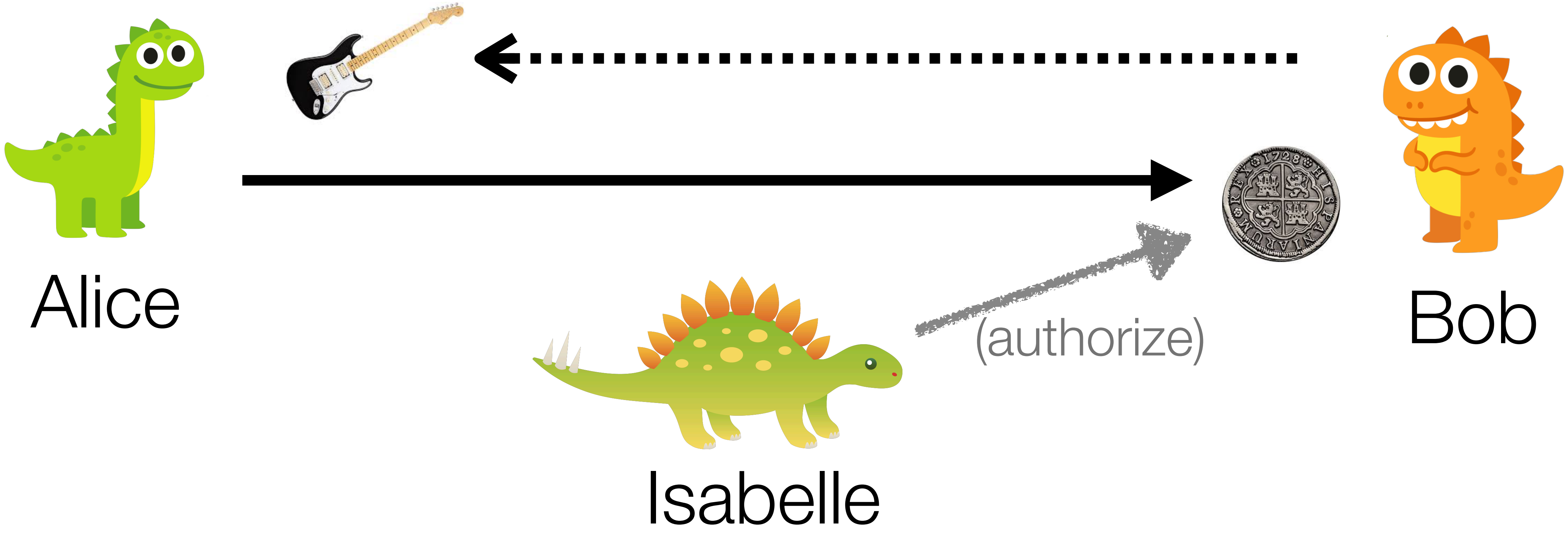
```
fact Coin    [holder: Party]

fact Offer   [id:     Symbol, terms:    Text,
              giver:  Party,  receiver: Party]

fact Accept  [id:     Symbol, accepter: Party]
```



```
fact Coin   [holder: Party,  issuer:   Party]

fact Offer  [id:     Symbol, terms:    Text,
             giver:  Party,  receiver: Party]

fact Accept [id:     Symbol, accepter: Party]
```



```
fact Coin    [holder: Party,   issuer:    Party]

fact Offer   [id:     Symbol, terms:     Text,
              giver:  Party,   receiver: Party]

fact Accept  [id:     Symbol, accepter: Party]
```



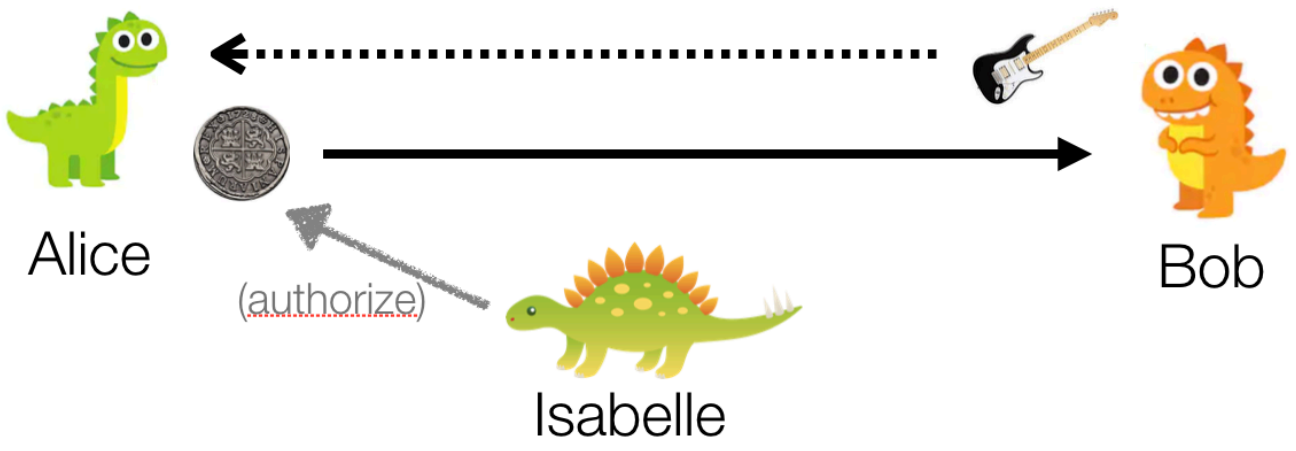
Alice

Bob

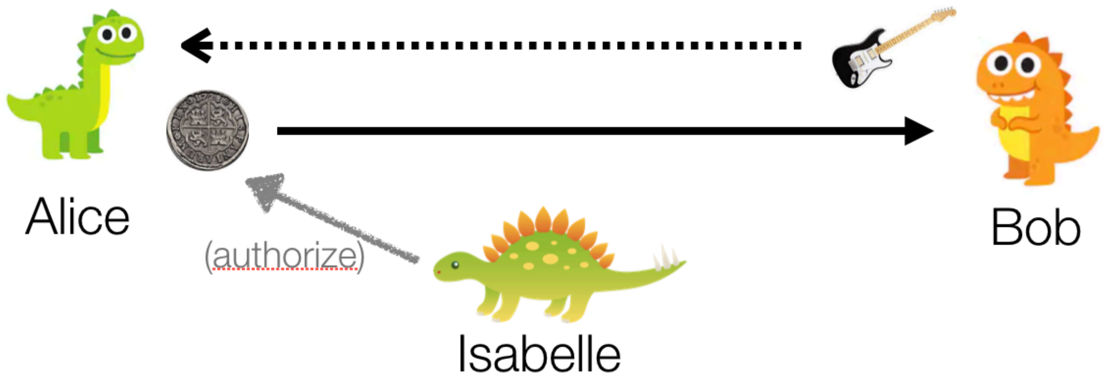
(authorize)

Isabelle

- Any fact can be stated with a party's own authority.
- Any existing fact that carries a single party's authority can be deleted by that party acting alone.
- Ensuring that coin facts always carry the authority of multiple parties means they cannot be unilaterally created, deleted, or transferred (updated).

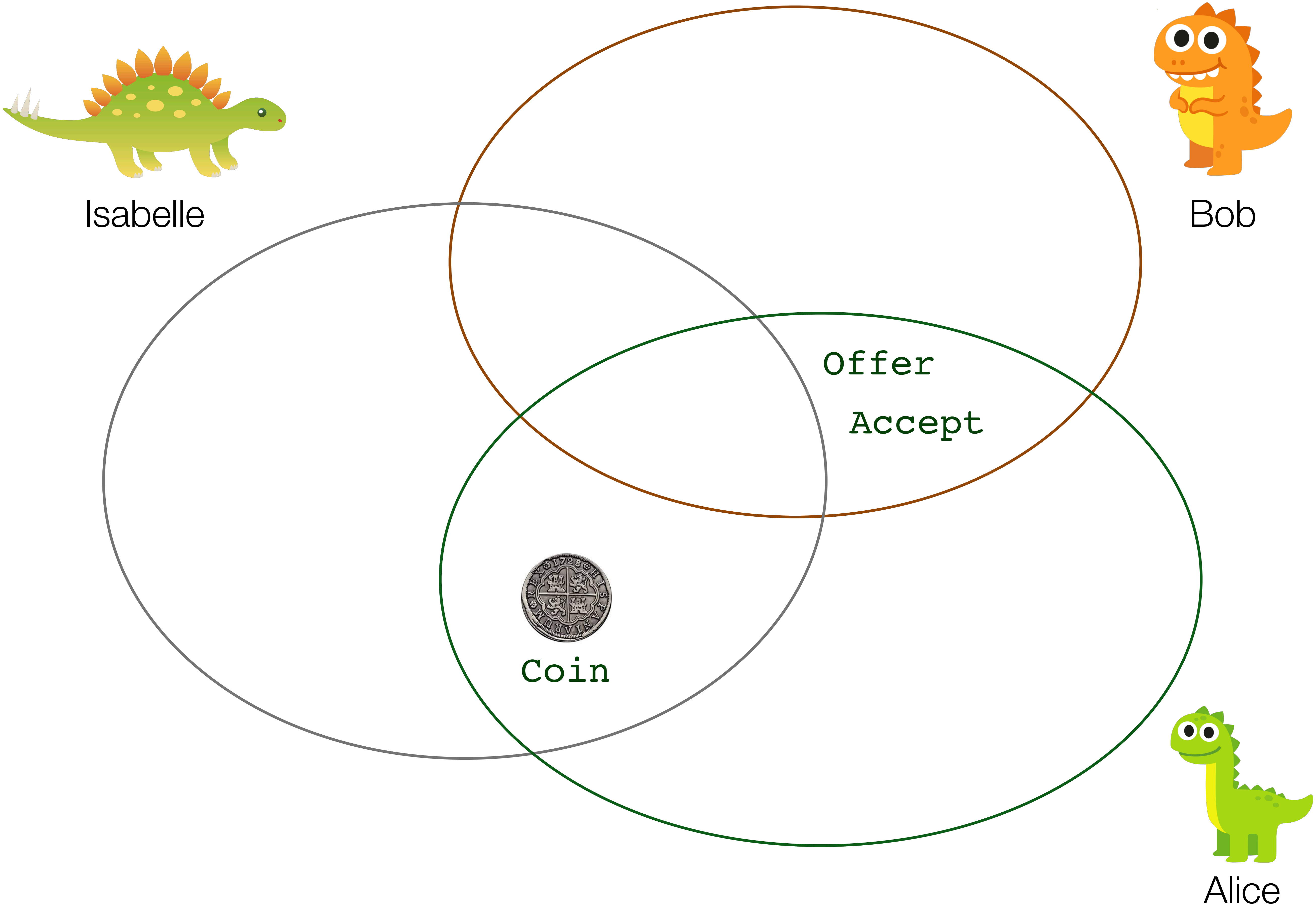


```
rule  transfer
await Offer  [id = ?i, giver = ?g, receiver = ?a]
      gain {g}
  and Accept [id = i,  accepter  = a]
      gain {a}
  and Coin   [issuer = ?s, holder = g]
      gain {s, g}
to
  say Coin   [issuer = s,  holder = a]
   by {s, a} use {'transfer}
```



```
Coin   [holder = !Alice, issuer = !Isabelle]
       by {!Alice, !Isabelle}    use {'transfer}

Offer  [id    = '1234,  terms = "for one guitar",
        giver = !Alice, receiver = !Bob]
       by {!Alice}  obs {!Bob}   use {'transfer}

Accept [id    = '1234,  accepter = !Bob]
       by {!Bob}    obs {!Alice} use {'transfer}
```

```
Coin   [holder = !Bob, issuer = !Isabelle]
       by {!Bob, !Isabelle} use {'transfer}
```



Isabelle

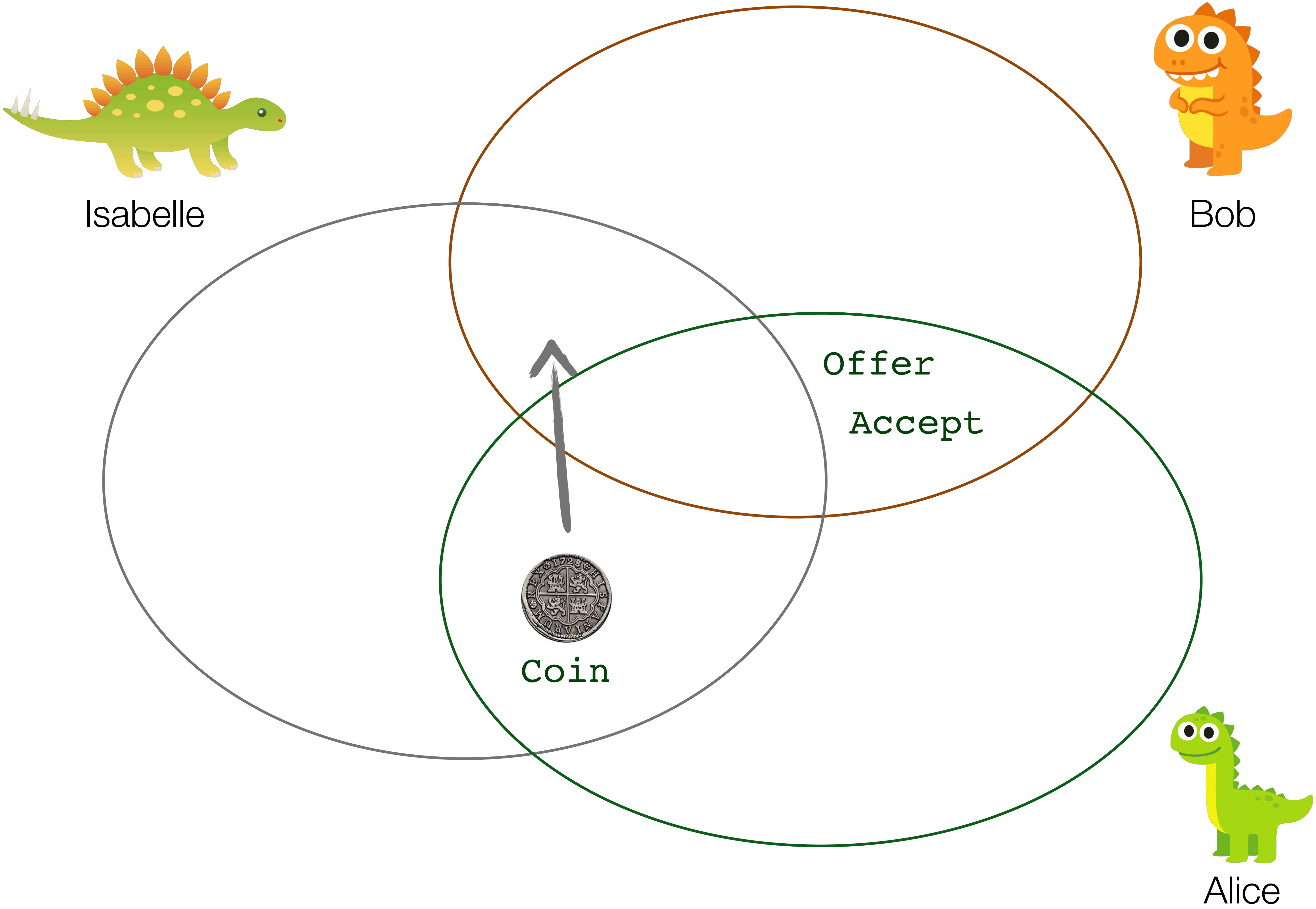
Bob

Offer

Accept

Coin

Alice



Isabelle

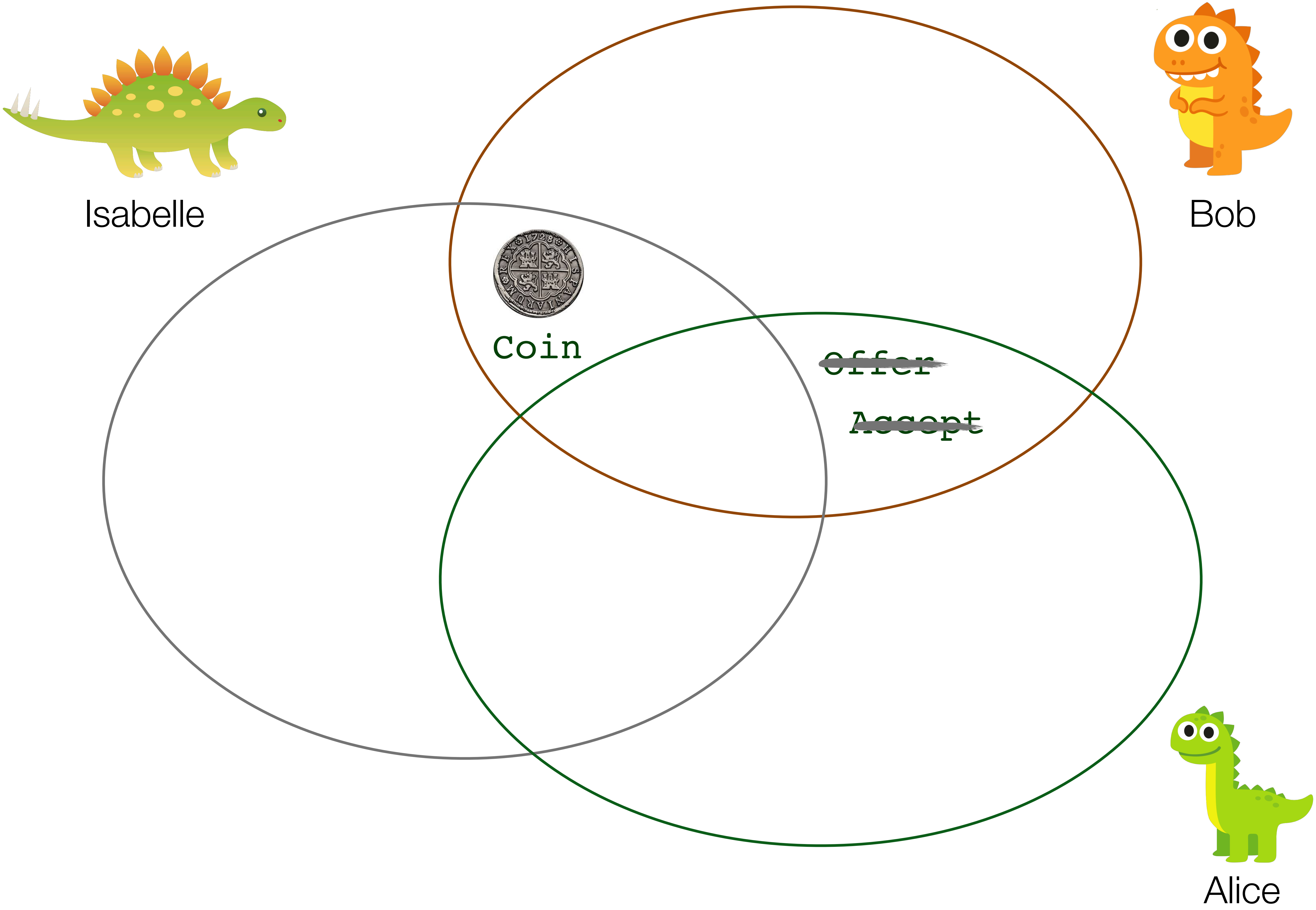
Bob

Offer

Accept

Coin

Alice



Isabelle

Bob

Coin

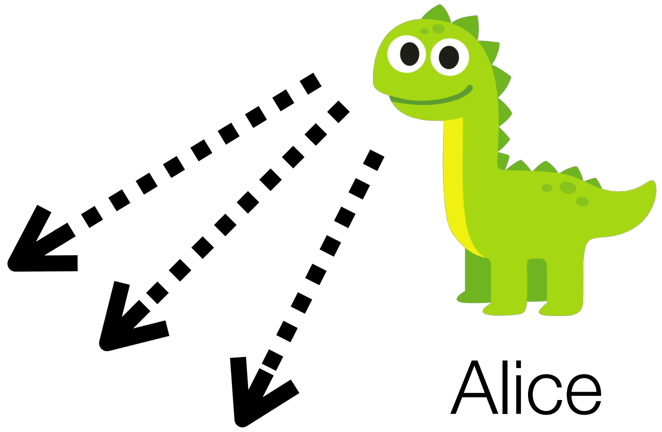
~~Offer~~

~~Accept~~

Alice

```
Transaction
{ ident: ... fresh number ...
, rule:  transfer
, spent:
    Coin   [holder = !Alice, issuer = !Isabelle]
           by {!Alice, !Isabelle}    use {'transfer}

    Offer  [id    = '1234,  terms = "for one guitar",
            giver = !Alice, receiver = !Bob]
           by {!Alice}  obs {!Bob}   use {'transfer}

    Accept [id    = '1234,  accepter = !Bob]
           by {!Bob}    obs {!Alice} use {'transfer}

, new:
    Coin   [holder = !Bob, issuer = !Isabelle]
           by {!Bob, !Isabelle} use {'transfer}
}
```

```
Transaction
{ ident: ... fresh number ...
, rule:  transfer
, spent:
    Coin   [holder = !Alice, issuer = !Isabelle]
           by {!Alice, !Isabelle}    use {'transfer}

    Offer  [id     = '1234,  terms = "for one guitar",
            giver = !Alice, receiver = !Bob]
           by {!Alice}  obs {!Bob}   use {'transfer}

    Accept [id     = '1234,  accepter = !Bob]
           by {!Bob}    obs {!Alice} use {'transfer}

, new:
    Coin   [holder = !Bob, issuer = !Isabelle]
           by {!Bob, !Isabelle} use {'transfer}
}
```



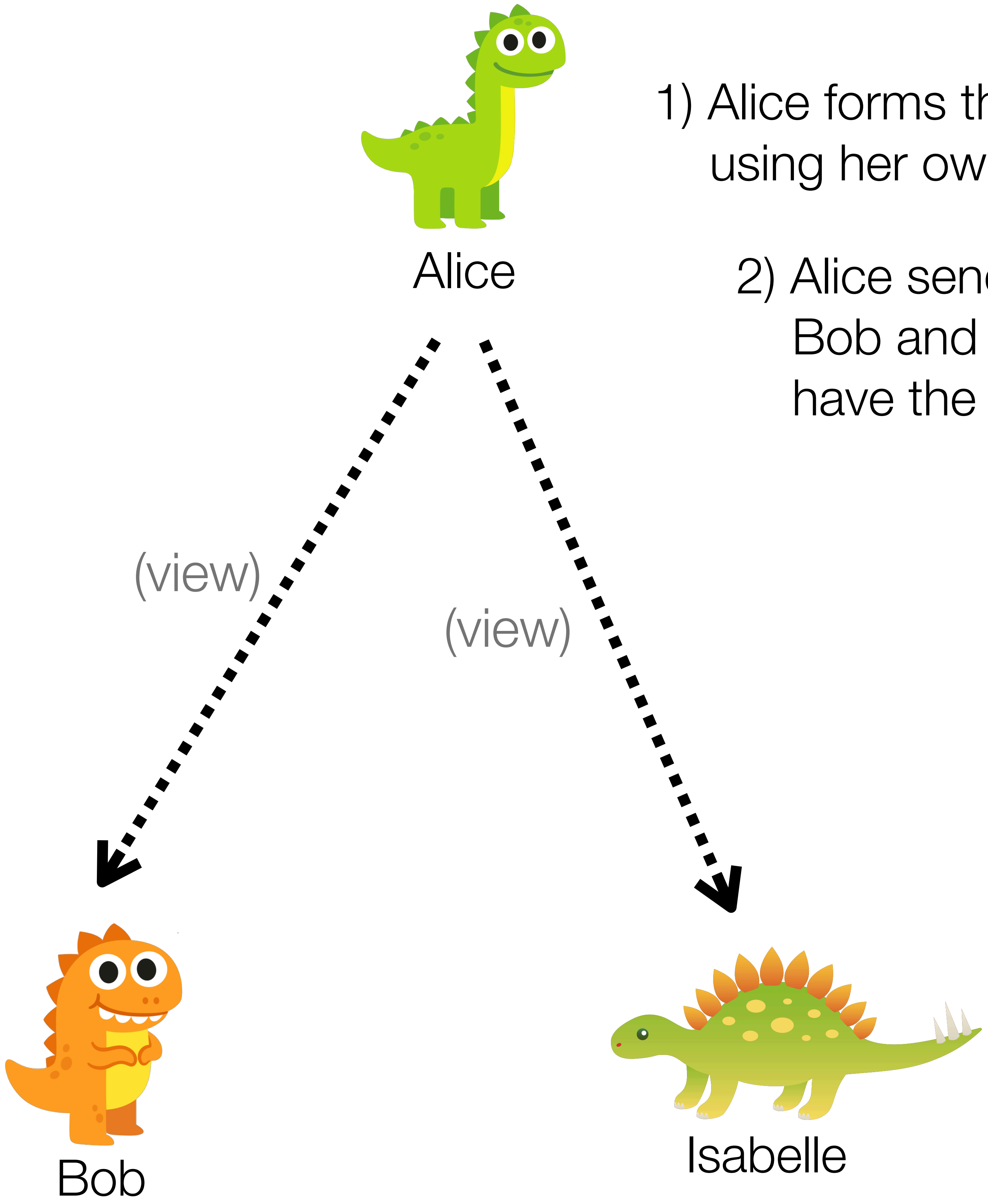
```
Transaction
{ ident: ... fresh number ...
, rule:  transfer
, spent:
    Coin   [holder = !Alice, issuer = !Isabelle]
           by {!Alice, !Isabelle}    use {'transfer}

    Offer  [id     = '1234,  terms = "~~for one guitar~~",
            ~~giver = !Alice, receiver~~ = !Bob]
           by {!Alice}  obs {!Bob}   use {'transfer}

    Accept [id     = '1234,  ~~accepter = !Bob]~~
           ~~by {!Bob}~~      obs {!Alice} use {'transfer}

, new:
    Coin   [holder = !Bob, issuer = !Isabelle]
           by {!Bob, !Isabelle} use {'transfer}
}
```

```
Transaction
{ ident: ... fresh number ...
, rule:  transfer
, spent:
    Coin   [holder = !Alice, issuer = !Isabelle]
           by {!Alice, !Isabelle}    use {'transfer}


,   HASH[fact_2, salt_2]


,   HASH[fact_3, salt_3]


, new:
    Coin   [holder = !Bob, issuer = !Isabelle]
           by {!Bob, !Isabelle} use {'transfer}
}
```

```
                    HASH[transaction]
Transaction
{ ident: ... fresh number ...
, rule:  transfer
, spent:
    Coin   [holder = !Alice, issuer = !Isabelle]
           by {!Alice, !Isabelle}    use {'transfer}


,   HASH[fact_2, salt_2]


,   HASH[fact_3, salt_3]


, new:
    Coin   [holder = !Bob, issuer = !Isabelle]
           by {!Bob, !Isabelle} use {'transfer}
}
```

1) Alice forms the complete transaction, using her own copy of the store.

2) Alice sends restricted views to Bob and Isabelle. All three views have the same transaction hash.



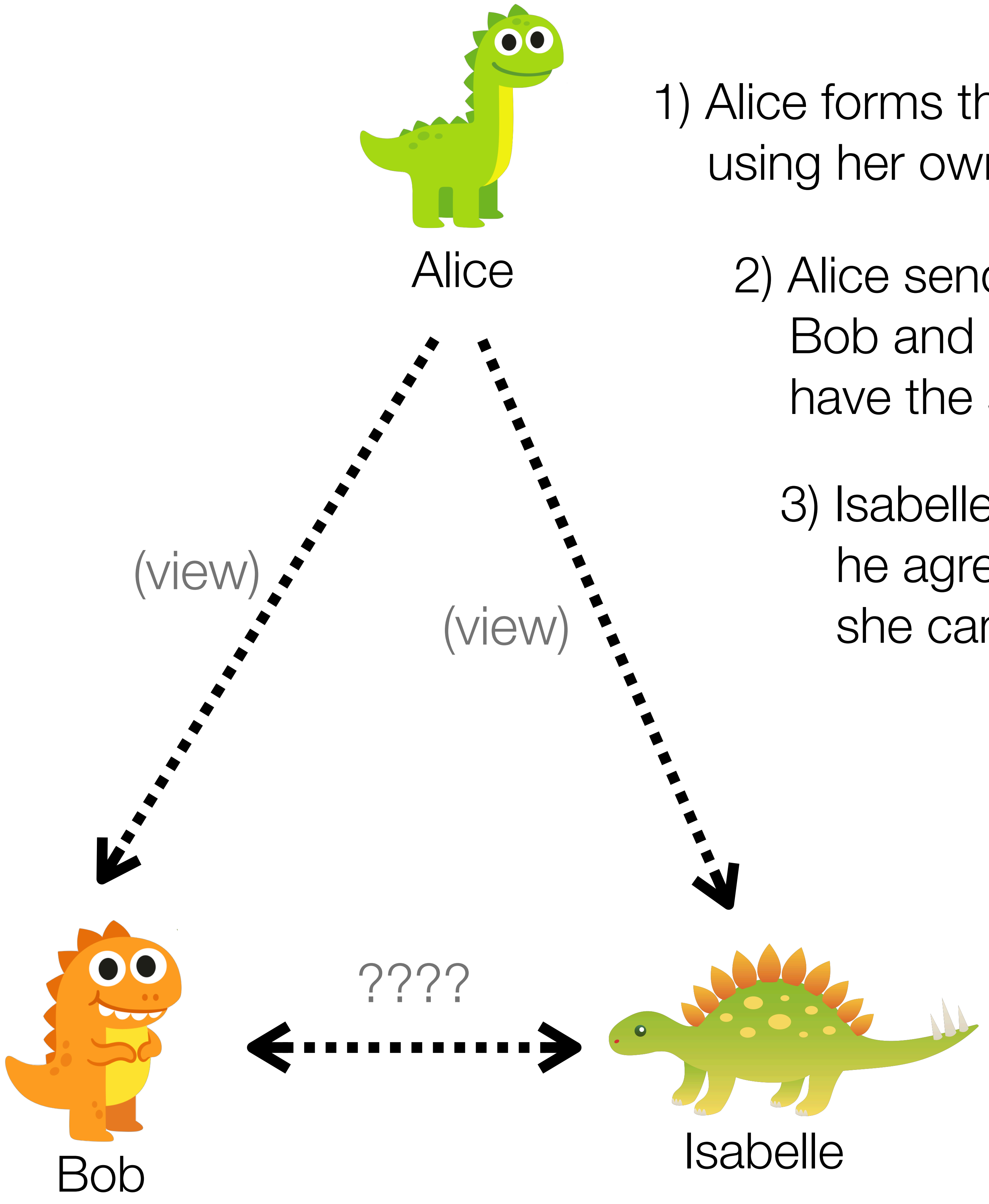
Alice

(view)

(view)

Bob

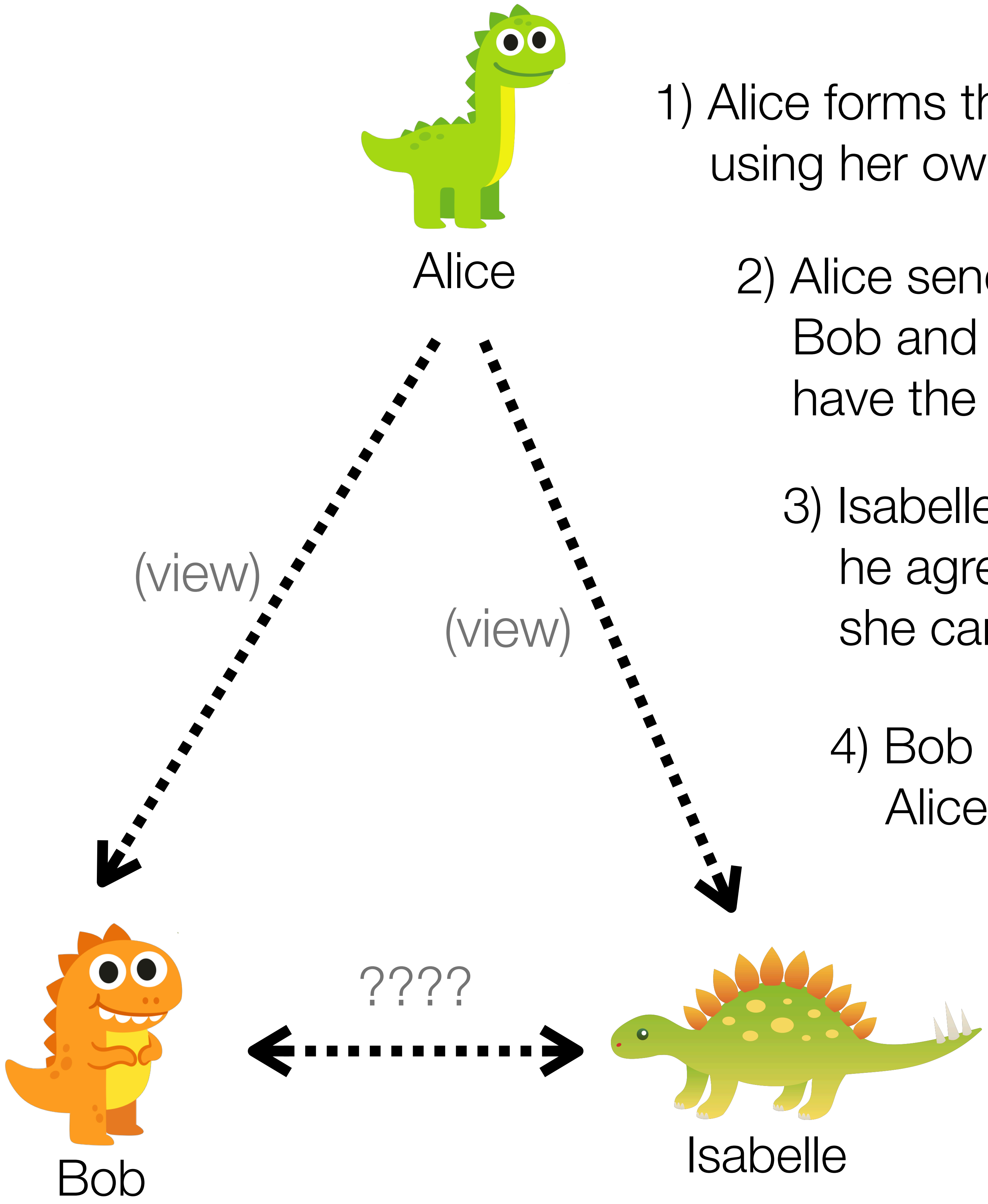
Isabelle



1) Alice forms the complete transaction, using her own copy of the store.

2) Alice sends restricted views to Bob and Isabelle. All three views have the same transaction hash.

3) Isabelle can confirm with Bob that he agrees to the transaction, even though she cannot see the terms of the Offer.



1) Alice forms the complete transaction, using her own copy of the store.

2) Alice sends restricted views to Bob and Isabelle. All three views have the same transaction hash.

3) Isabelle can confirm with Bob that he agrees to the transaction, even though she cannot see the terms of the Offer.

4) Bob can confirm with Isabelle that, Alice really has a coin to transfer to him.

Useful Theorems

FRAME CONDITION

IF a rule executes and generates some transaction

THEN we can execute the same rule with just the input facts that are listed in that transaction.

This lets the parties in the system re-execute the complete transaction views they receive to check their consistency.

AUTHORITY FLOW

IF a rule produces a fact that is authorized by some party.

THEN is also matched on a fact that was also authorized by the same party.

This tells us that the parties that submit transactions do not have any special rights.

Facts are given meaning by the rules only, not the people "running" the system

STORE WEAKENING

IF a rule executes and generates some transaction.

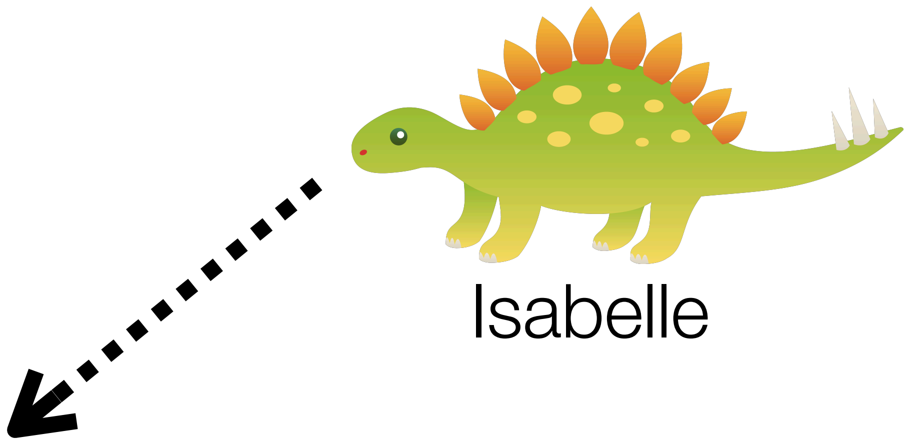
THEN it will do the same even when there are extra facts added to the store that the submitting party cannot see.

This is necessary for our semantics to make sense in an open system. Rule firing should not be inhibited by data you cannot see.

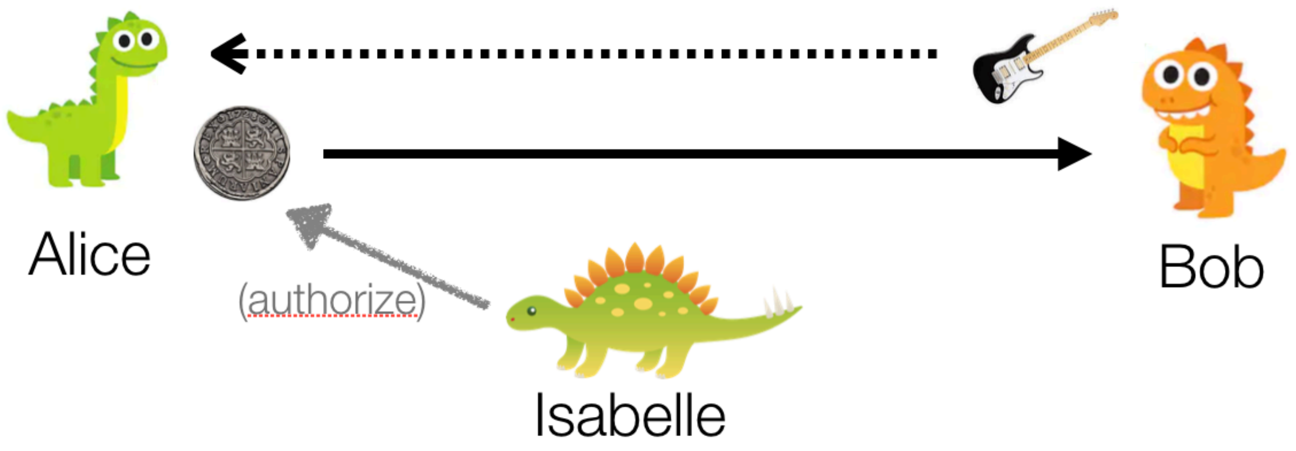
Rule Upgrade

```
rule  upgrade
await Coin   [issuer = ?s, holder = ?h]   gain {s,h}
  and LetsUpgrade [rules = ?rs]   gain {!Operator}
  and YeahOk      [party = s, rules = rs] gain {s}
  and YeahOk      [party = h, rules = rs] gain {h}
to
  say Coin   [issuer = s,  holder = h]
   by {s, h} use rs
```

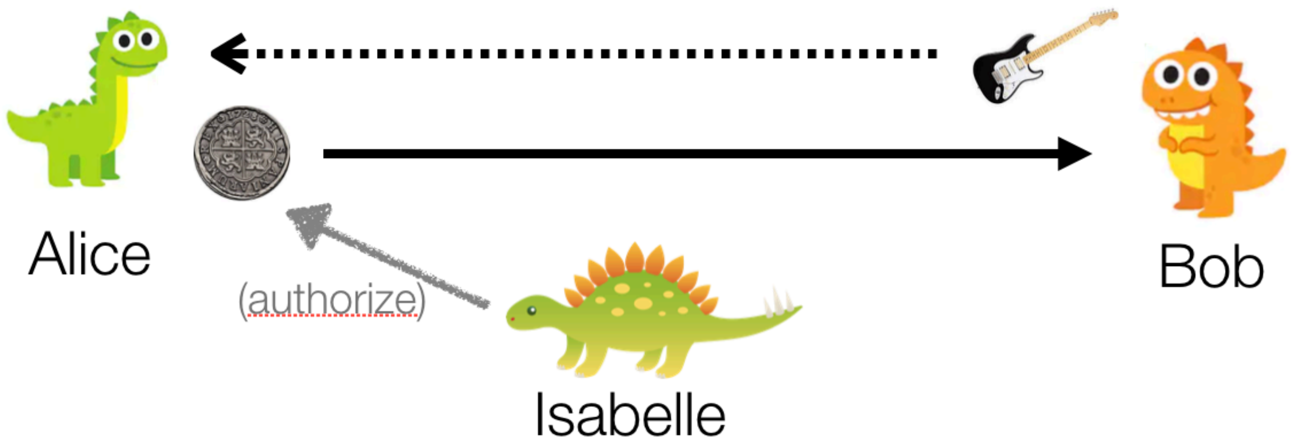
Rule Splitting



```
Transaction
{ ident: ... fresh number ...
, rule:  'transfer
, spent:
    Coin   [holder = !Alice, issuer = !Isabelle]
           by {!Alice, !Isabelle}   use {'transfer}

    Offer  [id    = '1234,  terms = "for one guitar",
           giver = !Alice, receiver = !Bob]
           by {!Alice}  obs {!Bob}   use {'transfer}

    Accept [id    = '1234,  accepter = !Bob]
           by {!Bob}    obs {!Alice} use {'transfer}

, new:
    Coin   [holder = !Bob, issuer = !Isabelle]
           by {!Bob, !Isabelle} use {'transfer}
}
```



```
rule  transfer
await Offer  [id = ?i, giver = ?g, receiver = ?a]
      gain {g}
  and Accept [id = i,   accepter  = a]
      gain {a}
  and Coin   [issuer = ?s, holder = g]
      gain {s, g}
to
  say Coin   [issuer = s,  holder = a]
   by {s, a} use {'transfer}
```



```
rule  transfer
await Offer  [id = ?i, giver = ?g, receiver = ?a]
      gain {g}
  and Accept [id = i,   accepter  = a]
      gain {a}
  and Coin   [issuer = ?s, holder = g]
      gain {s, g}
to
  say Coin   [issuer = s,  holder = a]
   by {s, a} use {'transfer}
```

split

```
rule  agree
await Offer  [id = ?i, giver = ?g, receiver = ?a]
      gain {g}
  and Accept [id = i,  accepter  = a]
      gain {a}
to
  say  Agreed [giver = g, receiver = a]
      by {g, a} obs {!Isabelle} use {doTransfer}


rule  doTransfer
await Agreed [giver = ?g, receiver = ?a]
      gain {g, a}
  and Coin   [issuer = ?s, holder = g]
      gain {s, g}
to
  say Coin   [issuer = s,  holder = a]
      by {s, a} use {doTransfer}
```