Typesafe Client for Smart Contracts

Parsing smart contracts and generating TypeScript definitions

Albert Groothedde
Architect Developer Experience
@alber70g
github.com/alber70g

Kadena
Proof of Work Scalable Blockchain
@kadena
github.com/kadena-io
/kadena-community

KADENA

Albert Groothedde

Architect Developer Experience
@alber70g
github.com/alber70g

KADENA

Albert Groothedde

Architect Developer Experience
@alber70g
github.com/alber70g

@kadena/client

@kadena/client

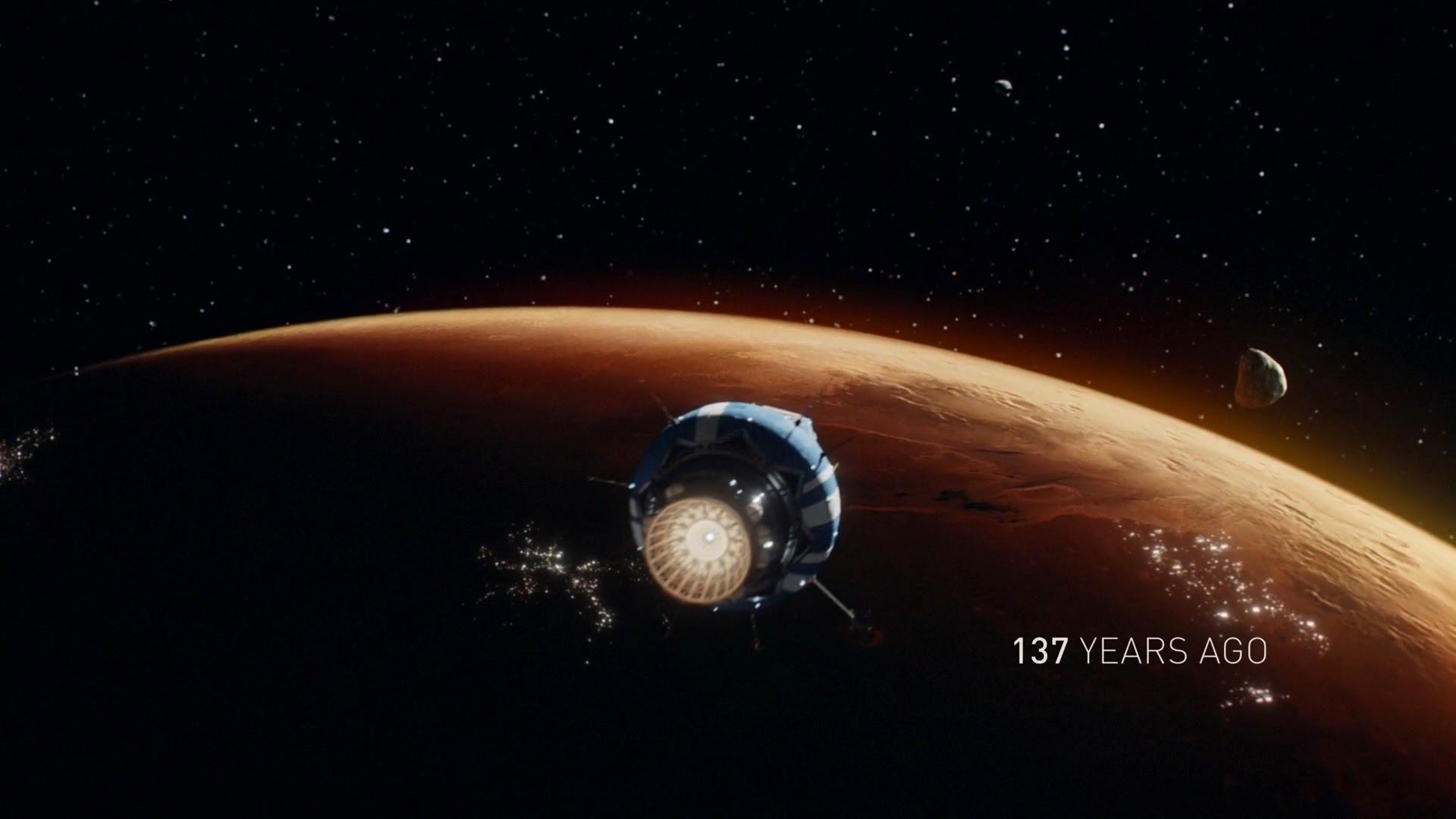
build transactions

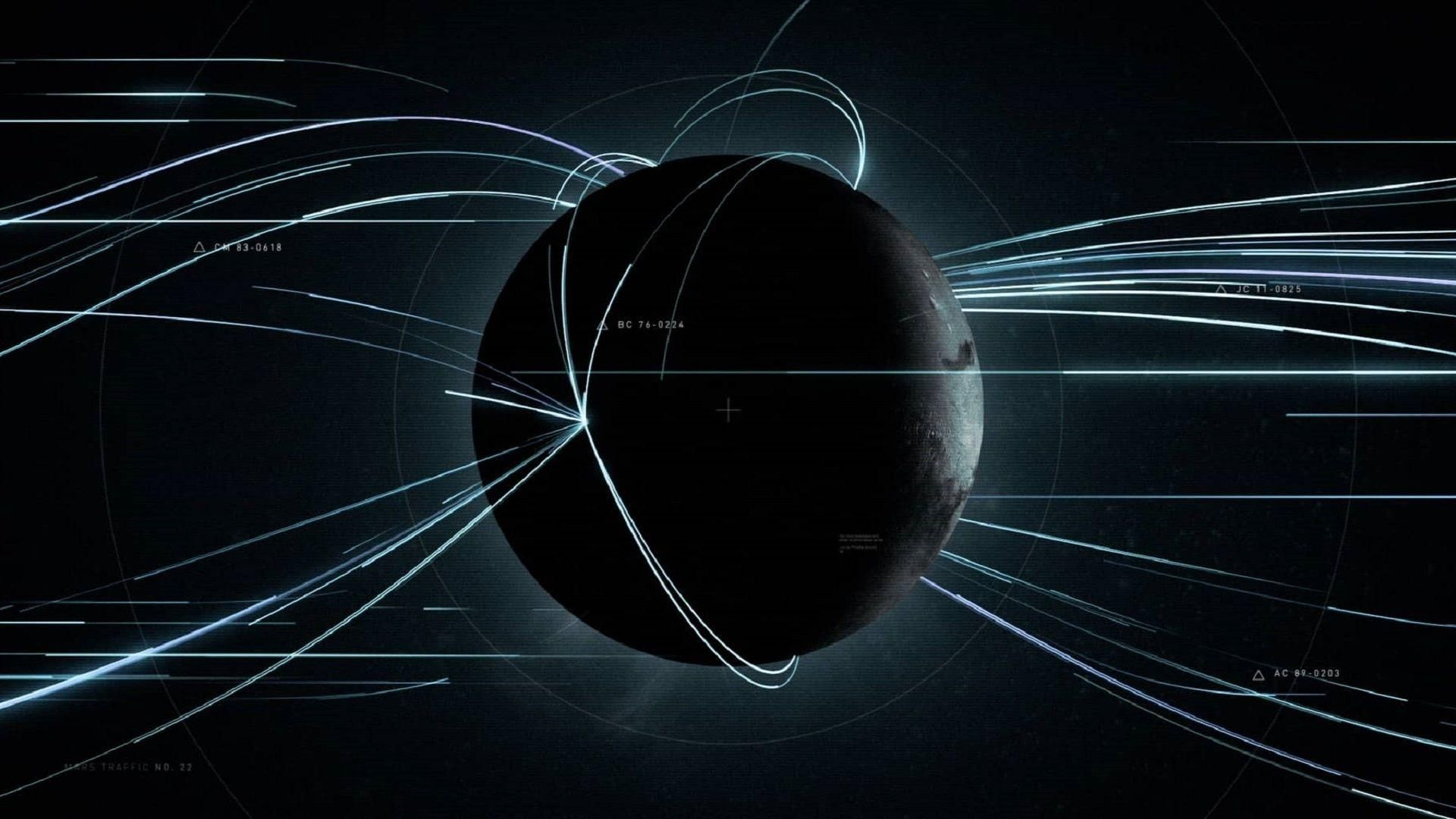
sign transactions

submit transactions


MORN
158

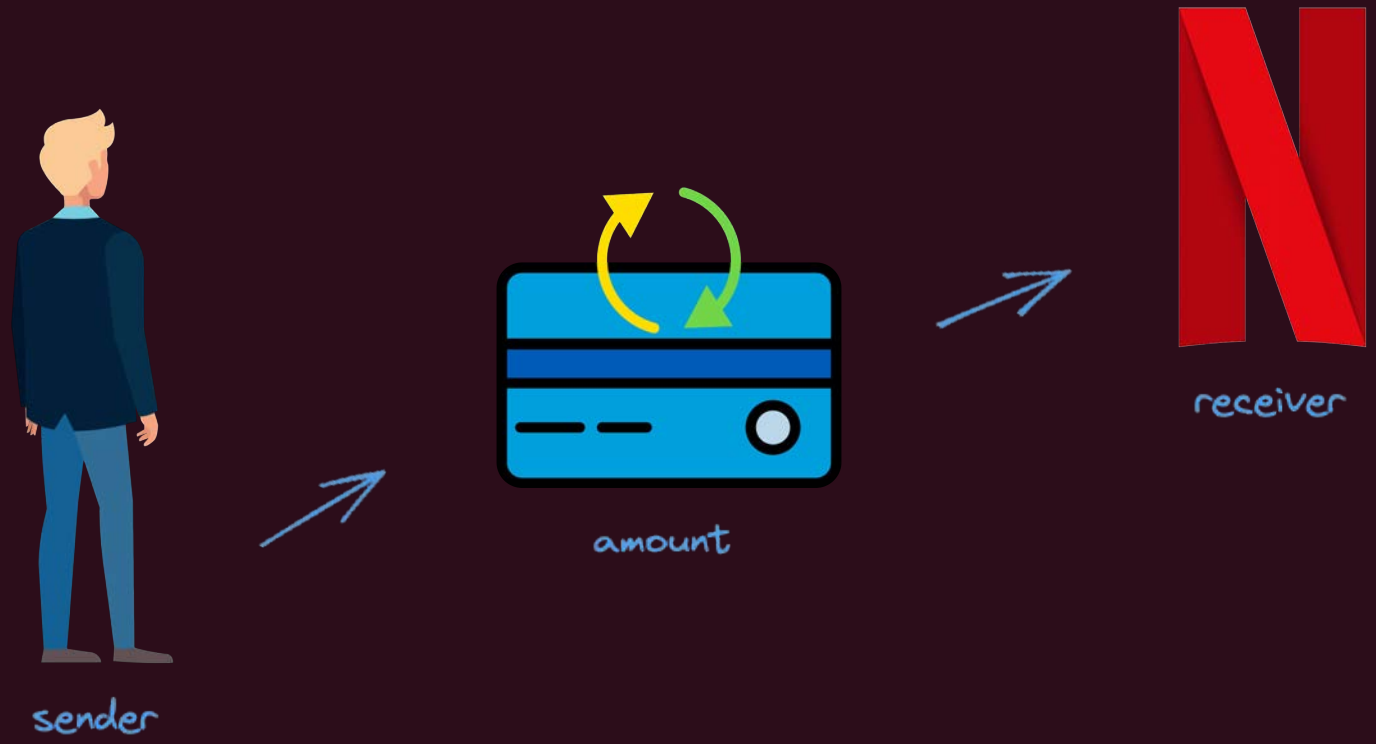

2
TYPHO


137 YEARS AGO

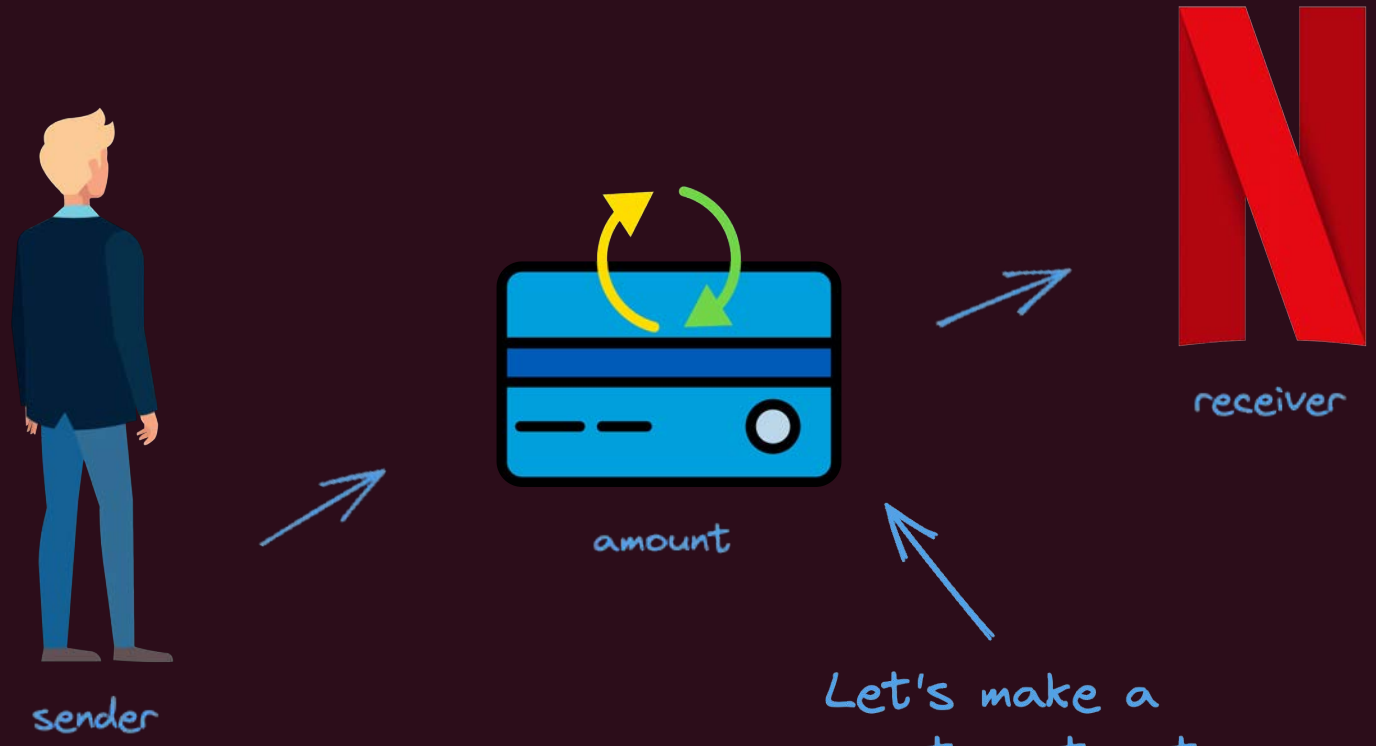

CM 83-0618
BC 76-0224
JC 11-0825
AC 89-0203
MARS TRAFFIC NO. 22


THE
EXPANSE

Subscription process



Subscription process



Subscription on StreamTV

```
subscribe( account:  string, months:  integer )
```

Subscription on StreamTV

```
subscribe( account:  string, months:  integer )
```

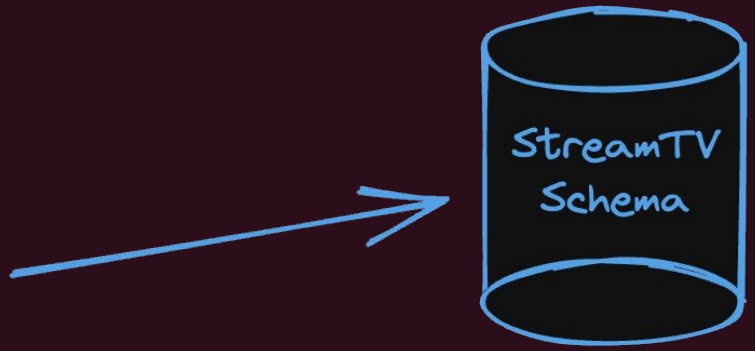
Subscription on StreamTV

```
subscribe( account:  string, months:  integer )

write-record( { key:  account, value:  months } )
```

Subscription on StreamTV

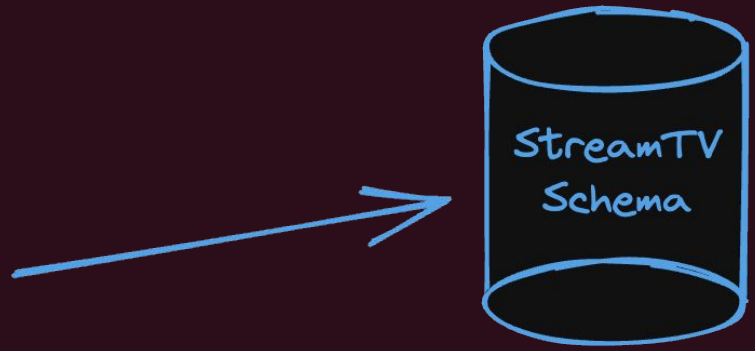
subscribe(account: string, months: integer)

write-record({ key: account, value: months })



Subscription on StreamTV

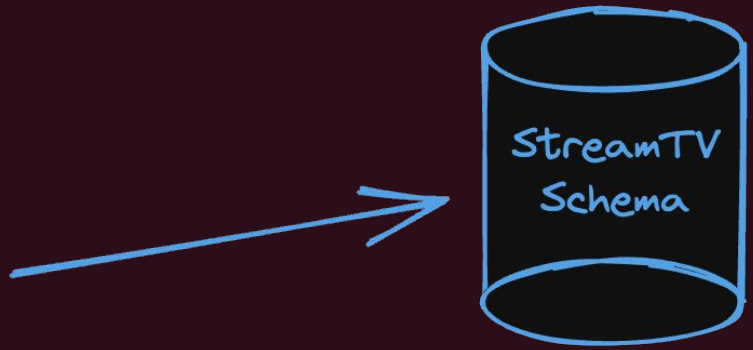
```
subscribe( account:  string, months:  integer )

write-record( { key:  account, value:  months } )

coin.transfer( account,  StreamTV,  months * 4.99 )
```



Subscription on StreamTV

```
subscribe( account:  string, months:  integer )

write-record( { key:  account, value:  months } )

coin.transfer( account,  StreamTV,  months * 4.99 )
```

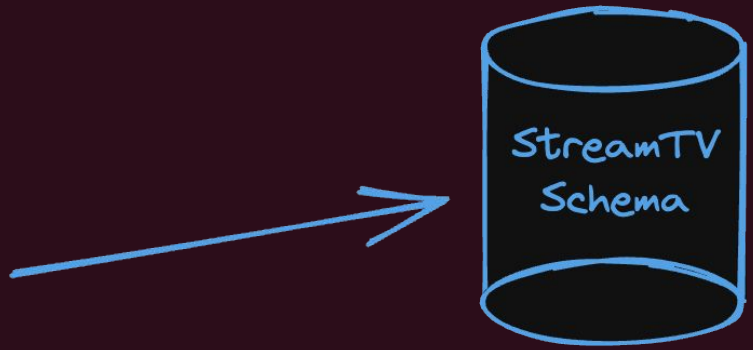
sender



Subscription on StreamTV

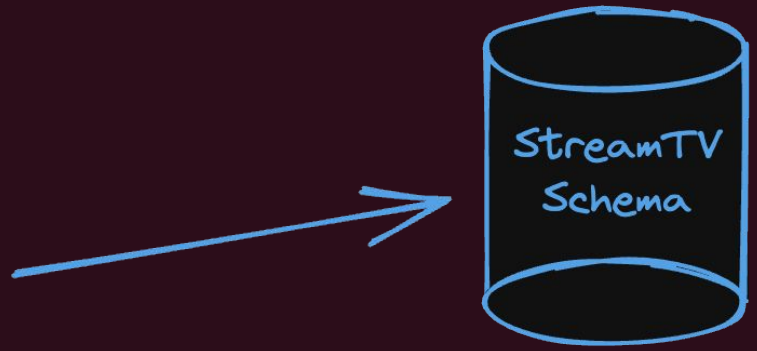
```
subscribe( account:  string, months:  integer )

write-record( { key:  account, value:  months } )

coin.transfer( account,  StreamTV,  months * 4.99 )
```

sender receiver



Subscription on StreamTV

```
subscribe( account:  string, months:  integer )

write-record( { key:  account, value:  months } )

coin.transfer( account,  StreamTV,  months * 4.99 )
```

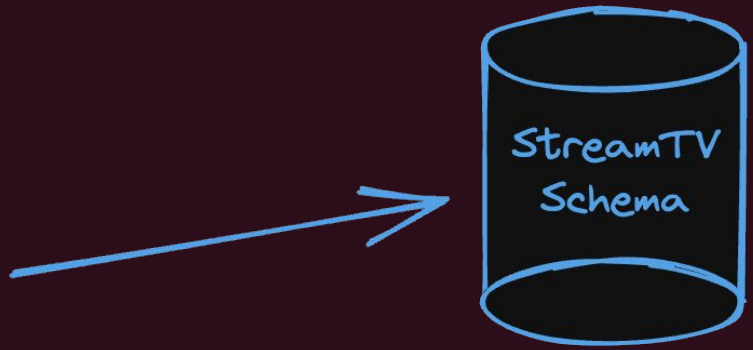


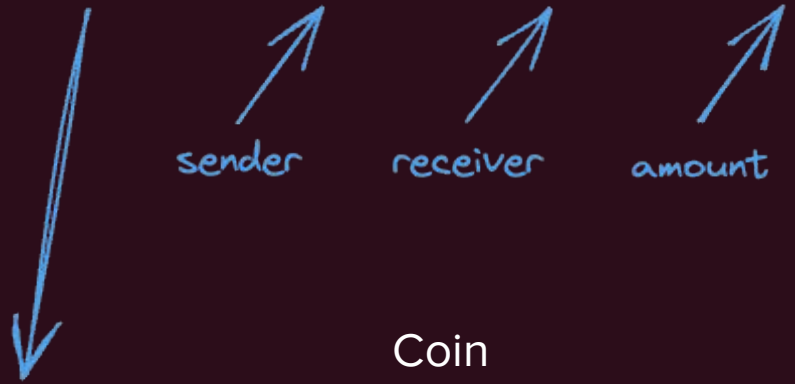
Subscription on StreamTV

subscribe(account: string, months: integer)

write-record({ key: account, value: months })

coin.transfer(account, StreamTV, months * 4.99)





Coin

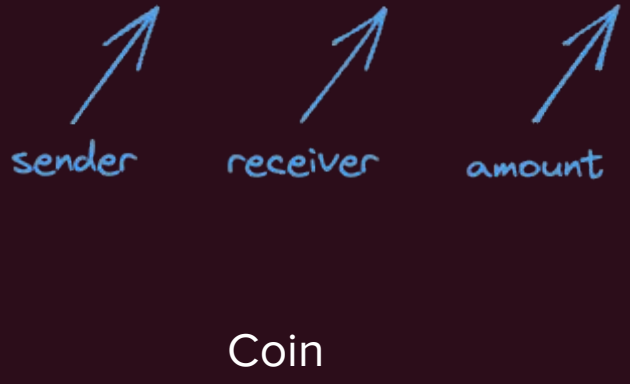
transfer(sender: string, receiver: string, amount: decimal)

Subscription on StreamTV

subscribe(account: string, months: integer)

write-record({ key: account, value: months })

coin.transfer(account, StreamTV, months * 4.99)



Coin

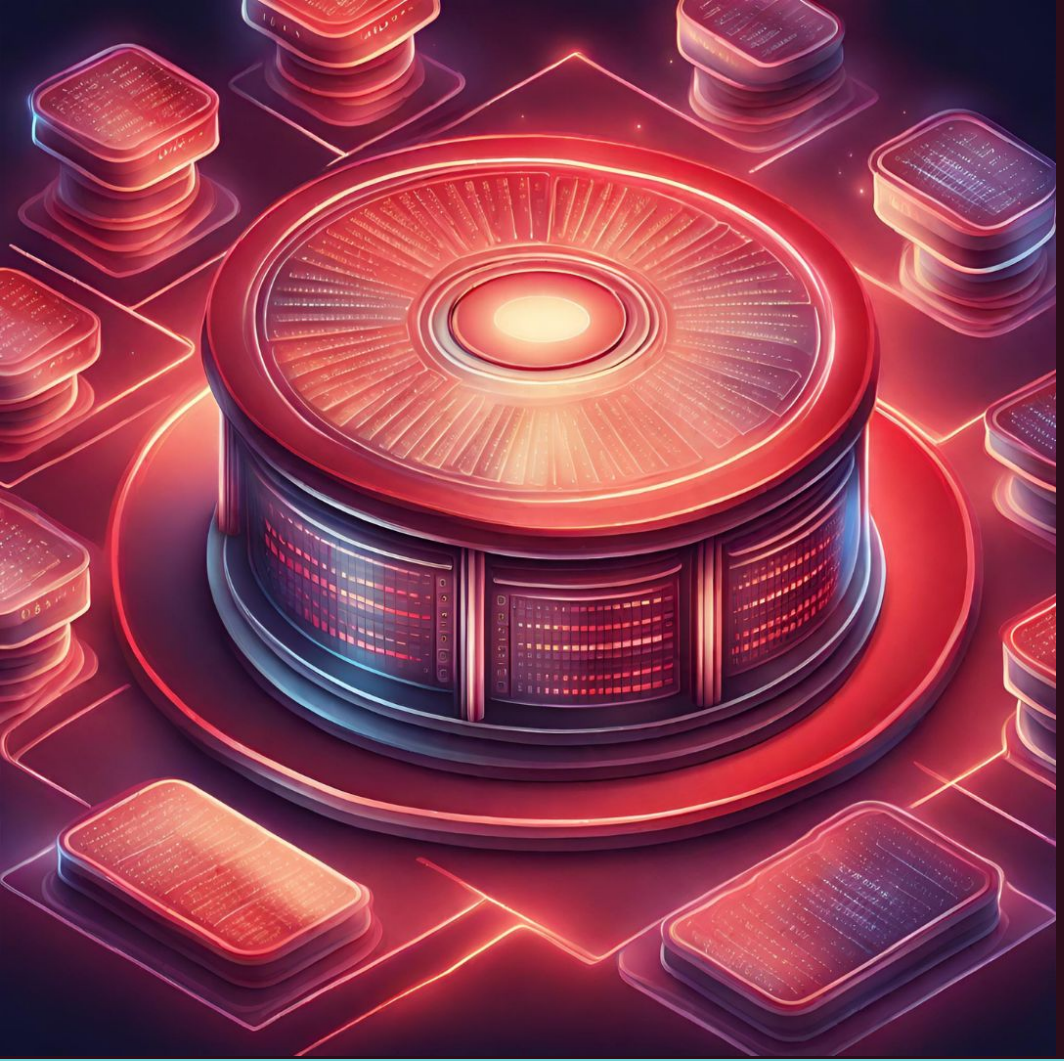
transfer(sender: string, receiver: string, amount: decimal)

Smart contract is...

A piece of code that's stored on a **decentralized platform**

Smart contract contains...

a **definition of a schema**
and **functions** to interact
with the schema



Smart contract contains...

a function that can

restrict the operations

on the records in that schema

In short,
smart contract is...

A set of **secure**,
serverless **functions**,

that can interact with a
database on a
decentralized platform.

A piece of code that's stored on a **decentralized platform**.

It contains a **definition of a schema** and **functions** to interact with the schema.

A function can **restrict the operations** on the records in that schema.

In short

A set of secure, serverless functions,
that can interact with a database on a decentralized platform.

PACT

PACT

Turing Incomplete

No recursion

No loops

Turing Incomplete

No recursion

No loops

Model checking

Formal verification

Turing Incomplete

No recursion

No loops

Model checking

Formal verification

```
(defun transfer:string (sender:string receiver:string amount:decimal)
  @model [ (property conserves-mass)
           (property (> amount 0.0))
           (property (valid-account sender))
           (property (valid-account receiver))
           (property (≠ sender receiver)) ]
```

Turing Incomplete

No recursion

No loops

Model checking

Formal verification

```
(defun transfer:string (sender:string receiver:string amount:decimal)
  @model [ (property conserves-mass)
           (property (> amount 0.0))
           (property (valid-account sender))
           (property (valid-account receiver))
           (property (≠ sender receiver)) ]
```

$852,250,000

could have been saved by Kadena.

savedby.kadena.network

Ask your favorite DeFi project why they're not building on Kadena.

View Projects ↘

Project	REKT Date	Amount Lost	Analysis	Remedy (hover for more info)
Compound	2021-10-04	$ 147,000,000	1	Pact ScalablePOW
Vee Finance	2021-09-21	$ 34,000,000	1	Pact
Cream Finance	2021-08-30	$ 18,800,000		Pact
X-Token	2021-08-30	$ 4,500,000		Pact
Punk Protocol	2021-08-13	$ 8,950,000		Pact FV
Poly Network	2021-08-11	$ 611,000,000	1	Pact
Popsicle Finance	2021-08-04	$ 20,000,000		Pact
THORChain	2021-07-26	$ 8,000,000	1	Pact ScalablePOW

A piece of code that's stored on a **decentralized platform**.

It contains a **definition of a schema** and **functions** to interact with the schema.

A function can **restrict the operations** to the records in that schema.

In short

A set of secured, serverless functions, on a decentralized platform.

A piece of code that's stored on a **decentralized platform**.

It contains a **definition of a schema** and **functions** to interact with the schema.

A function can **restrict the operations** to the records in that schema.

In short

A set of secured, serverless functions, on a decentralized platform.

```
(module coin GOVERNANCE
  ; .........

  ; ---------------------------------------------------------------------------
  ; Schemas and Tables

  (defschema coin-schema
    @doc "The coin contract token schema"
    @model [ (invariant (≥ balance 0.0)) ]

    balance:decimal
    guard:guard)

  (deftable coin-table:{coin-schema})
```

```
(module coin GOVERNANCE
  ; .........

  ; ---------------------------------------------------------------------------
  ; Schemas and Tables

  (defschema coin-schema
    @doc "The coin contract token schema"
    @model [ (invariant (≥ balance 0.0)) ]

    balance:decimal
    guard:guard)

  (deftable coin-table:{coin-schema})
```

```
(module coin GOVERNANCE
  ; .........

  ; ---------------------------------------------------------------------------
  ; Schemas and Tables

  (defschema coin-schema
    @doc "The coin contract token schema"
    @model [ (invariant (≥ balance 0.0)) ]
                        <-------------  id: string
    balance:decimal
    guard:guard)

  (deftable coin-table:{coin-schema})
```

```
(module coin GOVERNANCE
  ; .........

  ; ------------------------------------------------------------------------
  ; Schemas and Tables

  (defschema coin-schema
    @doc "The coin contract token schema"
    @model [ (invariant (≥ balance 0.0)) ]

    balance:decimal
    guard:guard)

  (deftable coin-table:{coin-schema})
```

id: string

Key

id	balance	guard
albert	133.7	canAccess(args): boolean
john	103	canAccess(args): boolean

```
(module coin GOVERNANCE
  ; .........

  ; ------------------------------------------------------------------------
  ; Schemas and Tables

  (defschema coin-schema
    @doc "The coin contract token schema"
    @model [ (invariant (≥ balance 0.0)) ]

    balance:decimal
    guard:guard)

  (deftable coin-table:{coin-schema})
```

id: string

Key	Value	
id	**balance**	**guard**
albert	133.7	canAccess(args): boolean
john	103	canAccess(args): boolean

```
(module coin GOVERNANCE
  ; .........

  ; ---------------------------------------------------------------------------
  ; Schemas and Tables

  (defschema coin-schema
    @doc "The coin contract token schema"
    @model [ (invariant (≥ balance 0.0)) ]
                          <---------- id: string
    balance:decimal
    guard:guard)

  (deftable coin-table:{coin-schema})
```

Key

Value

id	balance	guard
albert	133.7	key(keys-all, 554…)
john	103	key(keys-one, a…, b…)

A piece of code that's stored on a **decentralized platform**.

It contains a **definition of a schema** and **functions to interact with the schema**.

A function can **restrict the operations** to the records in that schema.

In short

A set of secured, serverless functions, on a decentralized platform.

A piece of code that's stored on a **decentralized platform**.

It contains a **definition of a schema** and **functions to interact with the schema**.

A function can **restrict the operations** to the records in that schema.

In short

A set of secured, serverless functions, on a decentralized platform.

```
(defun transfer:string (sender:string receiver:string amount:decimal)
  (enforce (!= sender receiver)
    "sender cannot be the receiver of a transfer")

  (validate-account sender)
  (validate-account receiver)

  (enforce (> amount 0.0)
    "transfer amount must be positive")

  (enforce-unit amount)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
(defun transfer:string (sender:string receiver:string amount:decimal)
  (enforce (!= sender receiver)
    "sender cannot be the receiver of a transfer")

  (validate-account sender)
  (validate-account receiver)

  (enforce (> amount 0.0)
    "transfer amount must be positive")

  (enforce-unit amount)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
(defun transfer:string (sender:string receiver:string amount:decimal)
  (enforce (!= sender receiver)
    "sender cannot be the receiver of a transfer")

  (validate-account sender)
  (validate-account receiver)

  (enforce (> amount 0.0)
    "transfer amount must be positive")

  (enforce-unit amount)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
(defun transfer:string (sender:string receiver:string amount:decimal)
  (enforce (!= sender receiver)
    "sender cannot be the receiver of a transfer")

  (validate-account sender)
  (validate-account receiver)

  (enforce (> amount 0.0)
    "transfer amount must be positive")

  (enforce-unit amount)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
(defun transfer:string (sender:string receiver:string amount:decimal)
  (enforce (!= sender receiver)
    "sender cannot be the receiver of a transfer")

  (validate-account sender)
  (validate-account receiver)

  (enforce (> amount 0.0)
    "transfer amount must be positive")

  (enforce-unit amount)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
(defun transfer:string (sender:string receiver:string amount:decimal)
  (enforce (!= sender receiver)
    "sender cannot be the receiver of a transfer")

  (validate-account sender)
  (validate-account receiver)

  (enforce (> amount 0.0)
    "transfer amount must be positive")

  (enforce-unit amount)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

A piece of code that's stored on a **decentralized platform**.

It contains a **definition of a schema** and **functions to interact with the schema**.

A function can **restrict the operations** to the records in that schema.

In short

A set of secured, serverless functions, on a decentralized platform.

A piece of code that's stored on a **decentralized platform**.

It contains a **definition of a schema** and **functions to interact with the schema**.

A function can **restrict the operations** to the records in that schema.

In short

A set of secured, serverless functions, on a decentralized platform.

```
(defun transfer:string (sender:string receiver:string amount:decimal)
  (enforce (!= sender receiver)
    "sender cannot be the receiver of a transfer")

  (validate-account sender)
  (validate-account receiver)

  (enforce (> amount 0.0)
    "transfer amount must be positive")

  (enforce-unit amount)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
(defun transfer:string (sender:string receiver:string amount:decimal)
  (enforce (!= sender receiver)
    "sender cannot be the receiver of a transfer")

  (validate-account sender)
  (validate-account receiver)

  (enforce (> amount 0.0)
    "transfer amount must be positive")

  (enforce-unit amount)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

with-capability is a function with 2 arguments

1. the capability (aka: guard)

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

with-capability is a function with 2 arguments

1. the capability (aka: guard)
2. code

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

with-capability is a function with 2 arguments

1. the capability (aka: guard)
2. code

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

Key

Value

id	balance	guard
albert	133.7	`key(keys-all, 554…)`
john	103	`key(keys-one, a…, b…)`

with-capability is a function with 2 arguments

1. the capability (aka: guard)
2. code

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

```
(coin.transfer "albert" "john" 133.7)
```

id	balance	guard
albert	133.7	key(keys-all, 554…)
john	103	key(keys-one, a…, b…)

with-capability is a function with 2 arguments

1. the capability (aka: guard)
2. code

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

```
(coin.transfer "albert" "john" 133.7)
```

id	balance	guard
albert	133.7	key(keys-all, 554…)
john	103	key(keys-one, a…, b…)

with-capability is a function with 2 arguments

1. the capability (aka: guard)
2. code

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

```
(coin.transfer "albert" "john" 133.7)
```

id	balance	guard
albert	133.7	key(keys-all, 554…)
john	103	key(keys-one, a…, b…)

with-capability is a function with 2 arguments

1. the capability (aka: guard)
2. code

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

```
(coin.transfer "albert" "john" 133.7)
```

id	balance	guard
albert	0	key(keys-all, 554…)
john	236.7	key(keys-one, a…, b…)

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

```
{
  code: '(coin.transfer "albert" "john" 133.7)',
  signers: [{
    "pubKey":"albert-pubkey",
    "clist":[
      {
        "name":"coin.TRANSFER","args":
        ["albert","john", {"decimal":"133.7"}]
      }
  ]}
}
```

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

```
{
  code: '(coin.transfer "albert" "john" 133.7)',
  signers: [{
    "pubKey":"albert-pubkey",
    "clist":[
      {
        "name":"coin.TRANSFER","args":
        ["albert","john", {"decimal":"133.7"}]
      }
  ]}
}
```

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

```
{
  code: '(coin.transfer "albert" "john" 133.7)',
  signers: [{
    "pubKey":"albert-pubkey",
    "clist":[
      {
        "name":"coin.TRANSFER","args":
        ["albert","john", {"decimal":"133.7"}]
      }
  ]}
}
```

```
(with-capability (TRANSFER sender receiver amount)
  (debit sender amount)
  (with-read coin-table receiver
    { "guard" := g }

    (credit receiver g amount))
  )
)
```

```
{
  code: '(coin.transfer "albert" "john" 133.7)',
  signers: [{
    "pubKey":"albert-pubkey",
    "clist":[
      {
        "name":"coin.TRANSFER","args":
        ["albert","john", {"decimal":"133.7"}]
      }
  ]}
}
```

```
{
  code: '(coin.transfer "albert" "john" 133.7)',
  signers: [{
    "pubKey":"albert-pubkey",
    "clist":[
      {
        "name": "coin.TRANSFER",
        "args": ["albert","john", {"decimal":"133.7"}]
      }
  ]},
  sigs: [{
    "sig": "0b4fce9e70d7347b15282cb...cf73f3e2dfdfdcf99af9569d06"
  }]
}
```

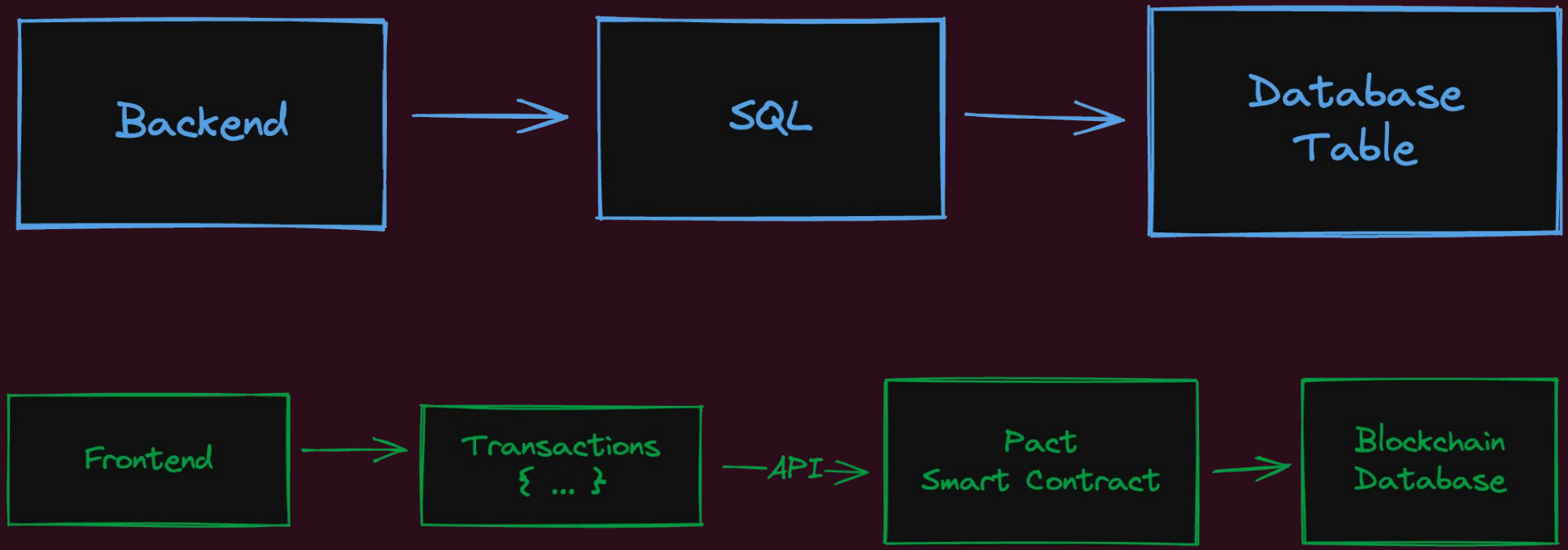
```
{
  code: '(coin.transfer "albert" "john" 133.7)',
  signers: [{
    "pubKey":"albert-pubkey",
    "clist":[
      {
        "name": "coin.TRANSFER",
        "args": ["albert","john", {"decimal":"133.7"}]
      }
  ]},
  sigs: [{
    "sig": "0b4fce9e70d7347b15282cb...cf73f3e2dfdfdcf99af9569d06"
  }]
}
```

```
{
  code: '(coin.transfer "albert" "john" 133.7)',
  signers: [{
    "pubKey":"albert-pubkey",
    "clist":[
      {
        "name": "coin.TRANSFER",
        "args": ["albert","john", {"decimal":"133.7"}]
      }
  ]},
  sigs: [{
    "sig": "0b4fce9e70d7347b15282cb...cf73f3e2dfdfdcf99af9569d06"
  }]
}
```

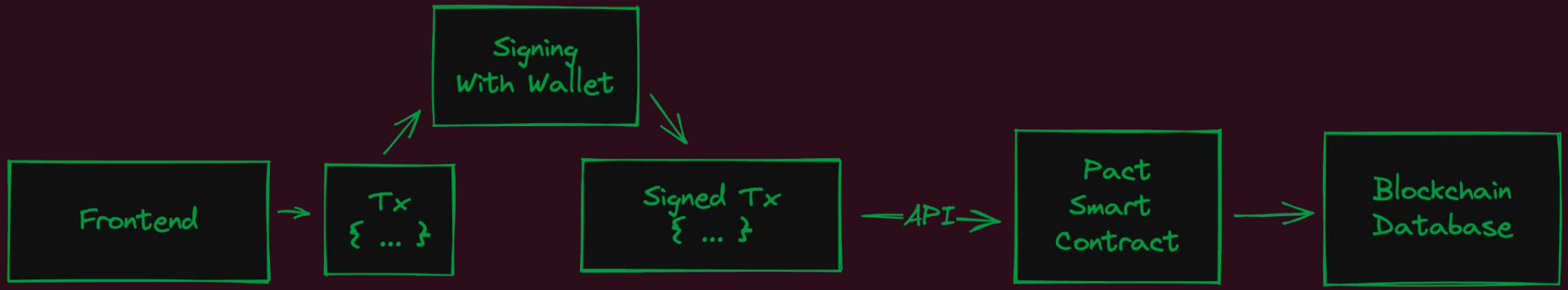
Why a Javascript Client?

Why a javascript client

Why a javascript client

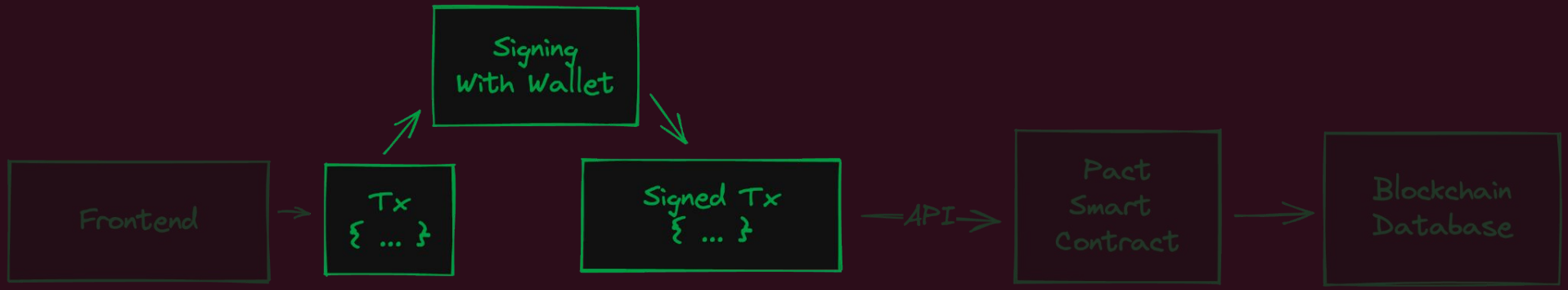


Why a javascript client



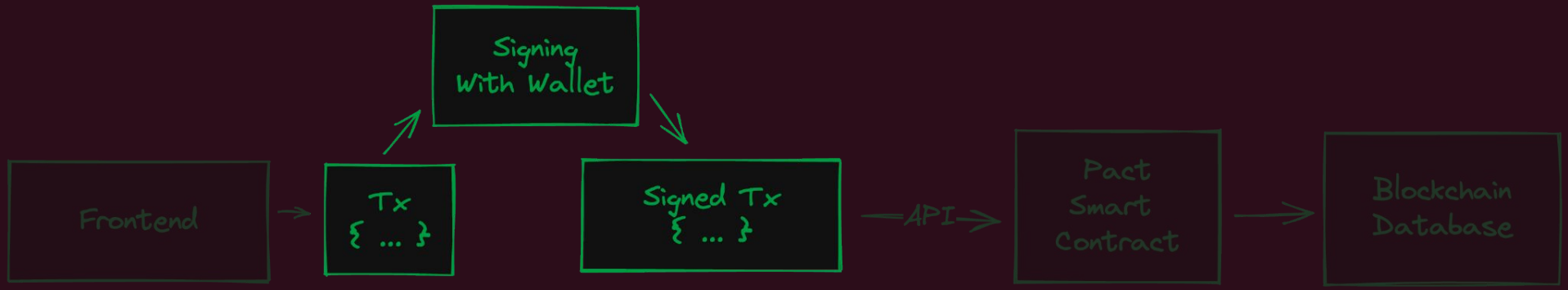
Why a javascript client

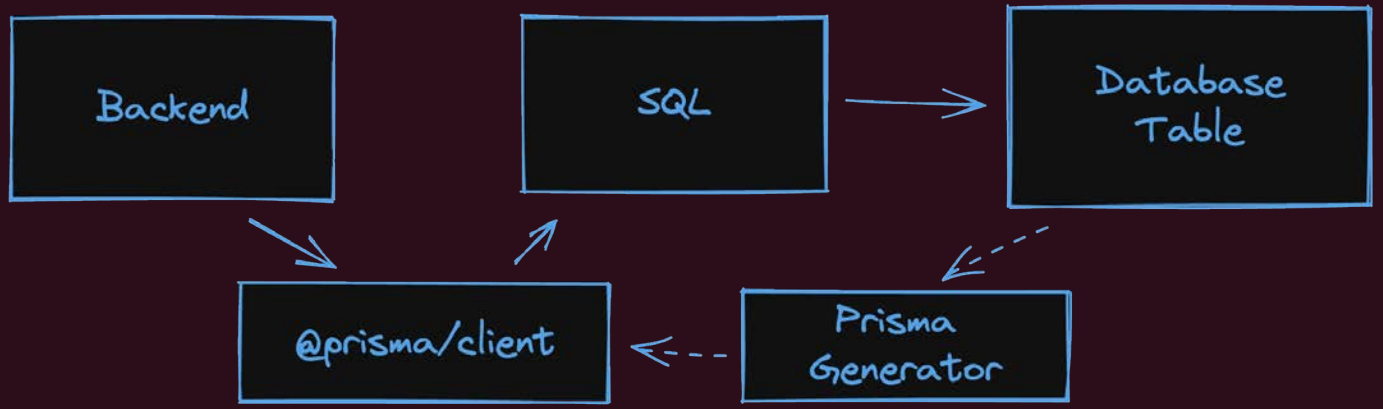
- Build a transaction
- Sign the transaction
- Send it to the blockchain
- Wait for it to be validated

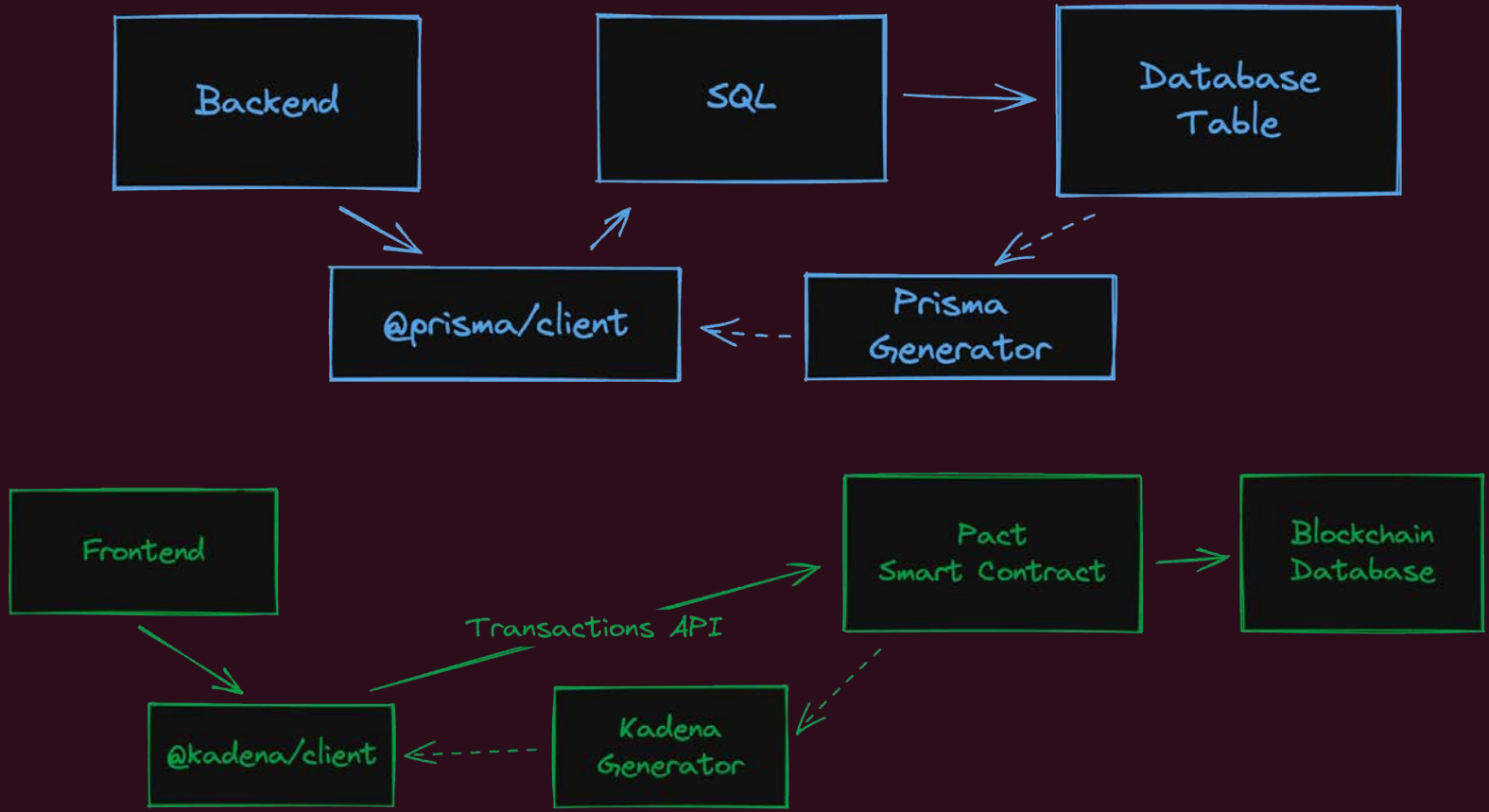


Why a javascript client

- Build a transaction
  - writing the code `(coin.transfer "albert" "john" 133.7)`
  - adding the relevant authorizations `(coin.TRANSFER x y amount)`
  - setting the correct context parameters (e.g. networkId, chainId, sender, etc...)




Backend
SQL
Database Table
@prisma/client
Prisma Generator


Backend
SQL
Database Table
@prisma/client
Prisma Generator
Frontend
Pact Smart Contract
Blockchain Database
Transactions API
@kadena/client
Kadena Generator

```
const transaction = Pact.builder
```

```
const transaction = Pact.builder
  .execution(Pact.modules.coin.transfer(sender, receiver, amount))      You, 1 second ago • U
```

```
transfer-create
transfer
TRANSFER-mgr
TRANSFER_XCHAIN-mgr
```

```
(property) "transfer": (se
    capability: ICapabilit
}
```

Rest Endpoint

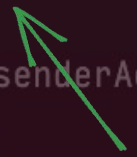
```
const transaction = Pact.builder
  .execution(Pact.modules.coin.transfer(sender, receiver, amount))
```

```
(property) "transfer": (sender: string, receiver: string, amount: IPactDecimal) ⇒ string & {
    capability: ICapability_transfer & ICapability_Coin_GAS;
}
```

```
const transacti
  .execution(Pact.modules.coin.transfer(sender, receiver, amount))
  .addSigner(keyFromAccount(sender), (withCapability) ⇒ [
    withCapability('coin.GAS'),
    withCapability('coin.TRANSFER', sender, receiver, amount),
  ])
  .setMeta({ c                    ccount: sender })
  .setNetworkId(NETWORK_ID)
  .createTransaction();
```

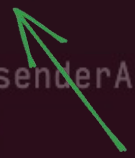
Rest Endpoint

```
const transaction = Pact.builder
  .execution(Pact.modules.coin.transfer(sender, receiver, amount))
  .addSigner(keyFromAccount(sender), (withCapability) => [
    withCapability('coin.GAS'),
    withCapability('coin.TRANSFER', sender, receiver, amount),
  ])
  .setMeta({ chainId: '0', senderAccount: sender })
  .setNetworkId(NETWORK_ID)
  .createTransaction();
```



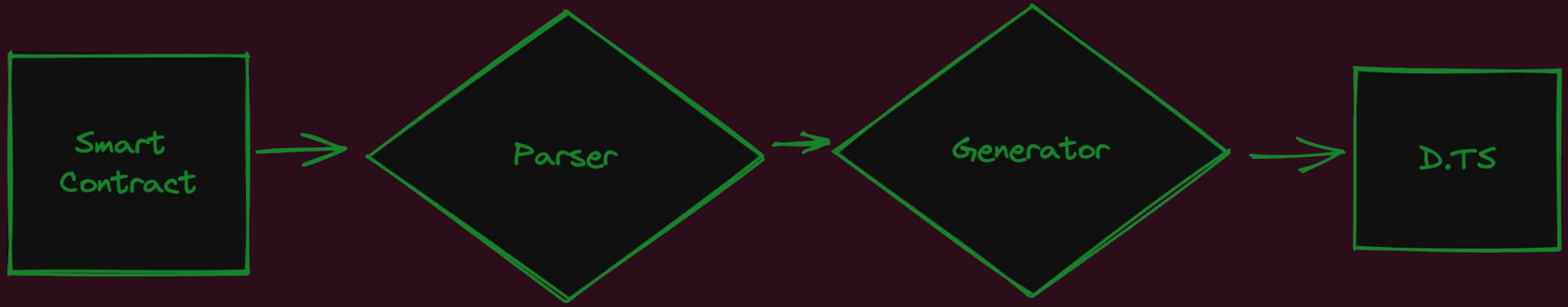
Authorization
e.g. JWT Token

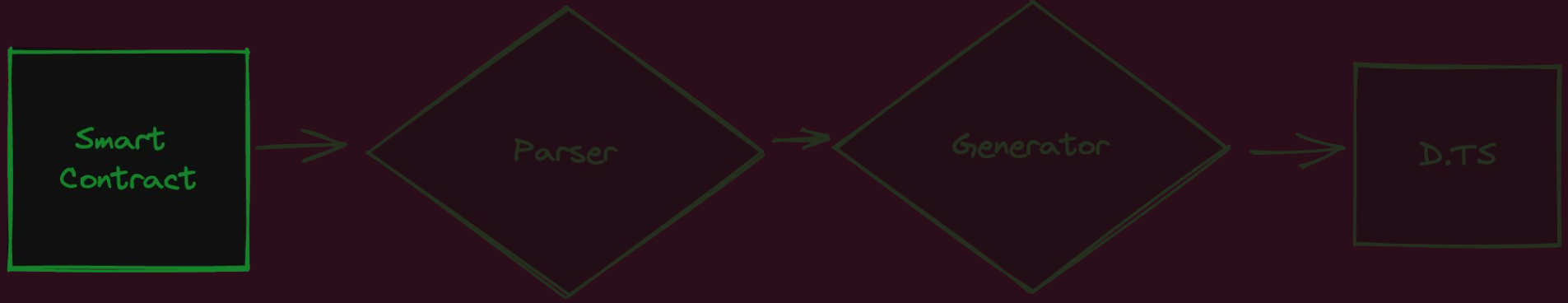
```
const transaction = Pact.builder
  .execution(Pact.modules.coin.transfer(sender, receiver, amount))
  .addSigner(keyFromAccount(sender), (withCapability) => [
    withCapability('coin.GAS'),
    withCapability('coin.TRANSFER', sender, receiver, amount),
  ])
  .setMeta({ chainId: '0', senderAccount: sender })
  .setNetworkId(NETWORK_ID)
  .createTransaction();
```

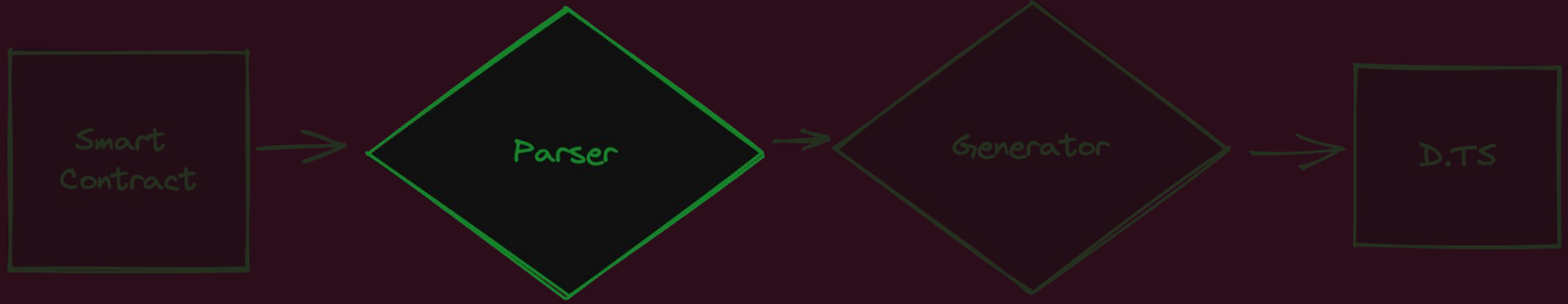


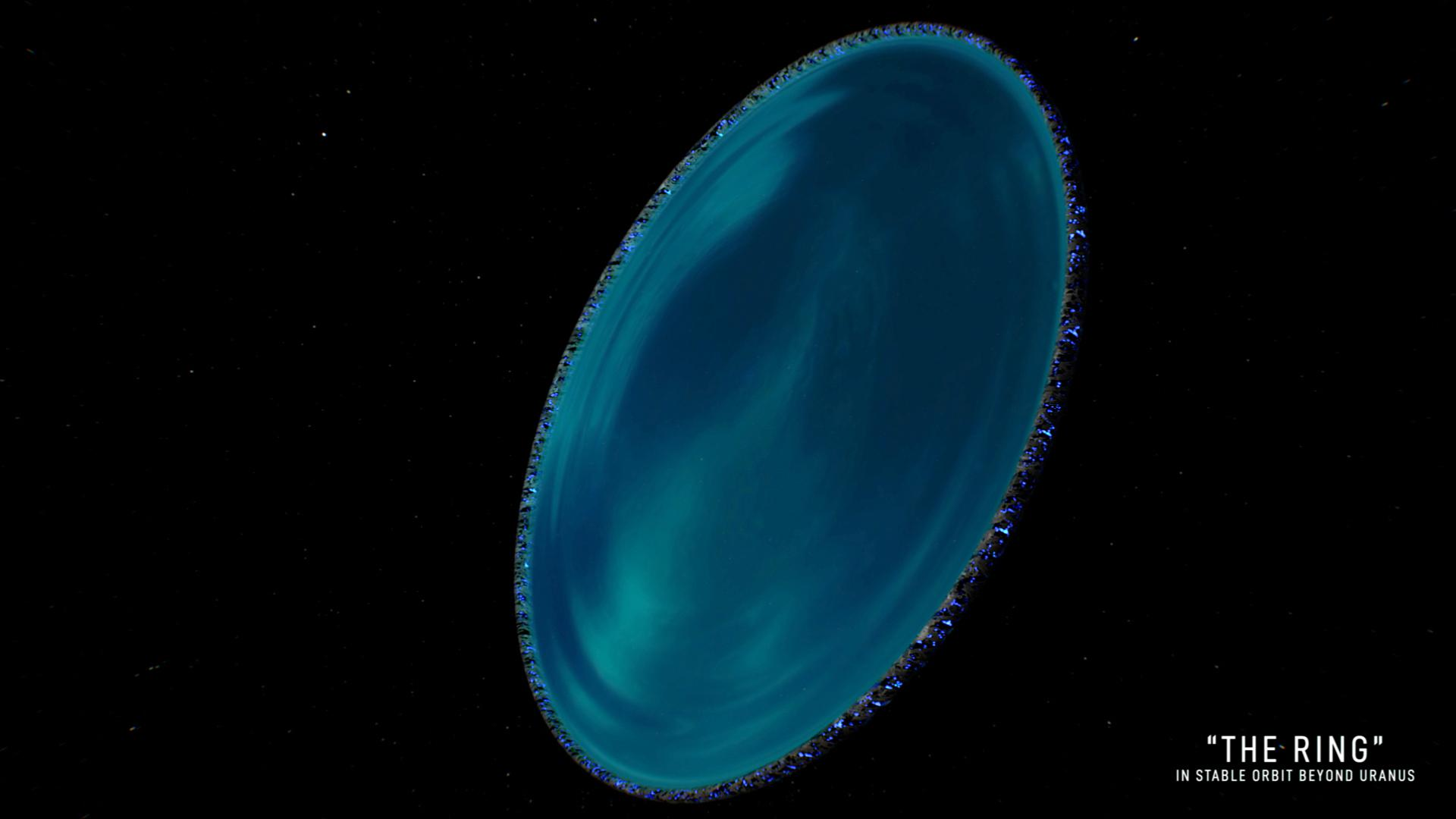
Authorization
e.g. JWT Token

```
const transaction = Pact.builder
  .execution(Pact.modules.coin.transfer(sender, receiver, amount))
  .addSigner(keyFromAccount(sender), (withCapability) => [
    withCapability('coin.GAS'),
    withCapability('coin.TRANSFER', sender, receiver, amount),
  ])
  .setMeta({ chainId: '0', senderAccount: sender })
  .setNetworkId(NETWORK_ID)
  .createTransaction();
```


Smart Contract
Parser
Generator
D.TS


Smart Contract
Parser
Generator
D.TS


Smart Contract
Parser
Generator
D.TS


"THE RING"
IN STABLE ORBIT BEYOND URANUS

Smart Contract

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

Parser Output

```
{
  "kind": "defun",
  "name": "transfer",
  "returnType": "string",
  "parameters": [
    { "name": "sender", "type": "string" },
    { "name": "receiver", "type": "string" },
    { "name": "amount", "type": "decimal" }
  ],
}
```

Smart Contract

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

Parser Output

```
{
  "kind": "defun",
  "name": "transfer",
  "returnType": "string",
  "parameters": [
    { "name": "sender", "type": "string" },
    { "name": "receiver", "type": "string" },
    { "name": "amount", "type": "decimal" }
  ],
}
```

Smart Contract

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

Parser Output

```
{
  "kind": "defun",
  "name": "transfer",
  "returnType": "string",
  "parameters": [
    { "name": "sender", "type": "string" },
    { "name": "receiver", "type": "string" },
    { "name": "amount", "type": "decimal" }
  ],
}
```

Smart Contract

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

Parser Output

```
{
  "kind": "defun",
  "name": "transfer",
  "returnType": "string",
  "parameters": [
    { "name": "sender", "type": "string" },
    { "name": "receiver", "type": "string" },
    { "name": "amount", "type": "decimal" }
  ],
}
```

Smart Contract

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

Parser Output

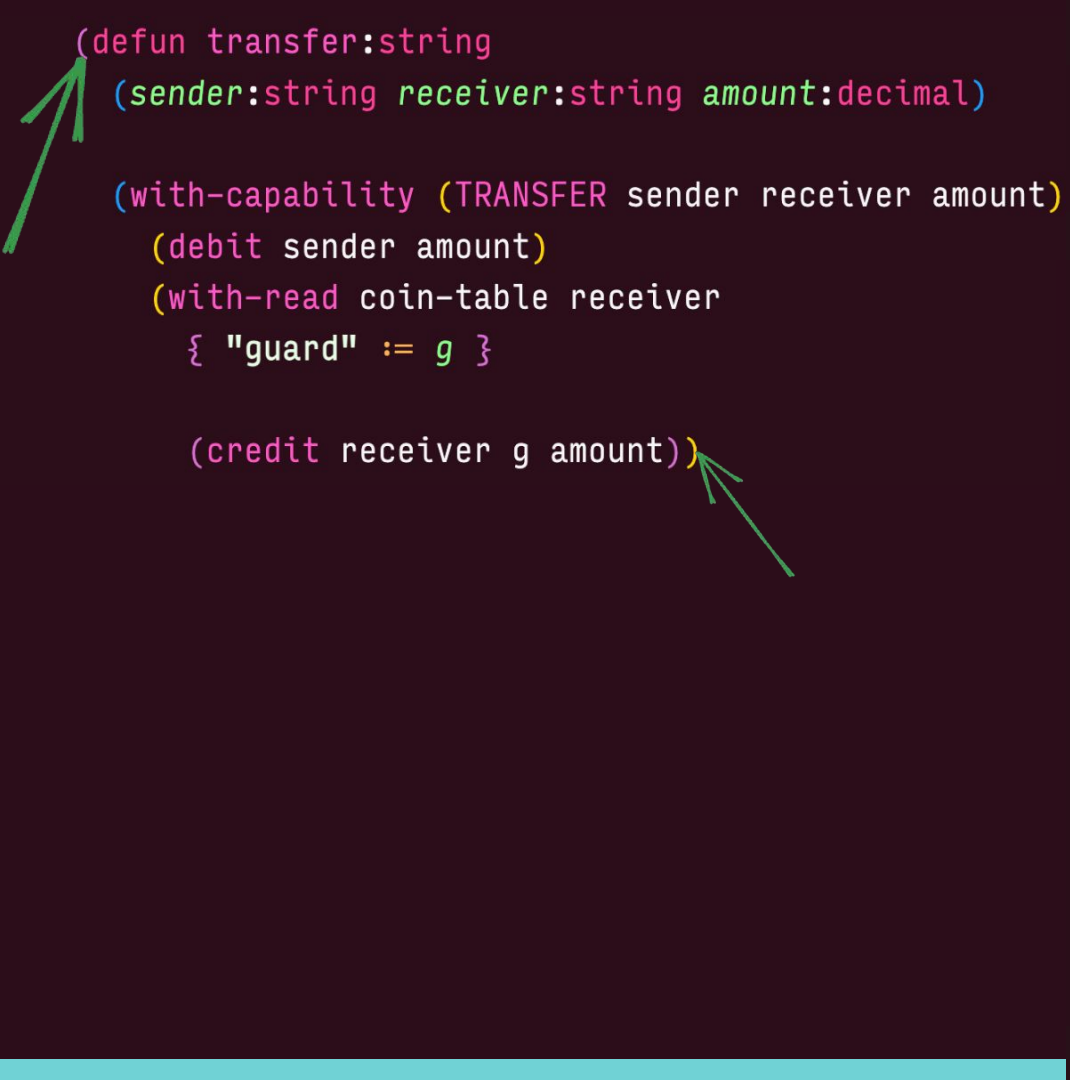
```
{
  "kind": "defun",
  "name": "transfer",
  "returnType": "string",
  "parameters": [
    { "name": "sender", "type": "string" },
    { "name": "receiver", "type": "string" },
    { "name": "amount", "type": "decimal" }
  ],
}
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```



```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

No it's not JQuery

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
{
  "kind": "defun",
  "name": "transfer",
  "returnType": "string",
  "parameters": [
    { "name": "sender", "type": "string" },
    { "name": "receiver", "type": "string" },
    { "name": "amount", "type": "decimal" }
  ],
}
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))

  {
    "kind": "defun",
    "name": "transfer",
    "returnType": "string",
    "parameters": [
      { "name": "sender", "type": "string" },
      { "name": "receiver", "type": "string" },
      { "name": "amount", "type": "decimal" }
    ],
  }
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))

  {
    "kind": "defun",
    "name": "transfer",
    "returnType": "string",
    "parameters": [
      { "name": "sender", "type": "string" },
      { "name": "receiver", "type": "string" },
      { "name": "amount", "type": "decimal" }
    ],
  }
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))

  {
    "kind": "defun",
    "name": "transfer",
    "returnType": "string",
    "parameters": [
      { "name": "sender", "type": "string" },
      { "name": "receiver", "type": "string" },
      { "name": "amount", "type": "decimal" }
    ],
  }
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
{
  "kind": "defun",
  "name": "transfer",
  "returnType": "string",
  "parameters": [
    { "name": "sender", "type": "string" },
    { "name": "receiver", "type": "string" },
    { "name": "amount", "type": "decimal" }
  ],
}
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
{
  "kind": "defun",
  "name": "transfer",
  "returnType": "string",
  "parameters": [
    { "name": "sender", "type": "string" },
    { "name": "receiver", "type": "string" },
    { "name": "amount", "type": "decimal" }
  ],
}
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))

  {
    "kind": "defun",
    "name": "transfer",
    "returnType": "string",
    "parameters": [
      { "name": "sender", "type": "string" },
      { "name": "receiver", "type": "string" },
      { "name": "amount", "type": "decimal" }
    ],
  }
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))


  {
    "kind": "defun",
    "name": "transfer",
    "returnType": "string",
    "parameters": [
      { "name": "sender", "type": "string" },
      { "name": "receiver", "type": "string" },
      { "name": "amount", "type": "decimal" }
    ],
  }
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

```
(defun transfer:string
  (sender:string receiver:string amount:decimal)

  (with-capability (TRANSFER sender receiver amount)
    (debit sender amount)
    (with-read coin-table receiver
      { "guard" := g }

      (credit receiver g amount))
```

```
export const method = <T extends IParser>(
  type: 'defun' | 'defcap' | 'defpact',
  bodyParser: T = skipTheRest as T,
) ⇒
  block(
    $('kind', id(type)),
    $('name', atom),
    maybe($('returnType', typeRule)),
    block(
      maybe(
        repeat(
          $('parameters',
            seq($('name', atom),
              $('type', maybe(typeRule))
            )),
        ),
      ),
    ),
    maybe(id('@doc')),
    maybe($('doc', str)),
    bodyParser,
  );
```

Parser Output


```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "functionCalls": {
        "internal": [],
        "external": []
      },
      "allExtractedCaps": []
    }],
    "usedModules": []
}}
```

Parser Output


```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "functionCalls": {
        "internal": [],
        "external": []
      },
      "allExtractedCaps": []
    }],
    "usedModules": []
}}
```


Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
    }
  }
}
```

Parser Output

```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "functionCalls": {
        "internal": [],
        "external": []
      },
      "allExtractedCaps": []
    }],
    "usedModules": []
}}
```

Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
    }
  }
}
```

Parser Output

```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "functionCalls": {
        "internal": [],
        "external": []
      },
      "allExtractedCaps": []
    }],
    "usedModules": []
}}
```

Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
```

Parser Output

```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "functionCalls": {
        "internal": [],
        "external": []
      },
      "allExtractedCaps": []
    }],
    "usedModules": []
}}
```

Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
```

Parser Output

```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "functionCalls": {
        "internal": [],
        "external": []
      },
      "allExtractedCaps": []
    }],
    "usedModules": []
}}
```

Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
```

Parser Output

```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "functionCalls": {
        "internal": [],
        "external": []
      },
      "allExtractedCaps": []
    }],
    "usedModules": []
}}
```

Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
```

Parser Output

```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "functionCalls": {
        "internal": [],
        "external": []
      },
      "allExtractedCaps": []
    }],
    "usedModules": []
}}
```

Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
```

Parser Output


```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "functionCalls": {
        "internal": [],
        "external": []
      },
      "allExtractedCaps": []
    }],
    "usedModules": []
}}
```


Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
```

Parser Output


```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "withCapabilities": [ "TRANSFER" ],
      "allExtractedCaps": [
          {
            "name": "TRANSFER",
            "fullModuleName": "coin",
            "reason": "with-capability",
            "origin": "transfer",
            "capability": {
              "kind": "defcap",
              "name": "TRANSFER",
              "returnType": "bool",
```


Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
```

Parser Output


```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "withCapabilities": [ "TRANSFER" ],
      "allExtractedCaps": [
          {
            "name": "TRANSFER",
            "fullModuleName": "coin",
            "reason": "with-capability",
            "origin": "transfer",
            "capability": {
              "kind": "defcap",
              "name": "TRANSFER",
              "returnType": "bool",
```


Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
```

Parser Output


```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "withCapabilities": [ "TRANSFER" ],
      "allExtractedCaps": [
          {
            "name": "TRANSFER",
            "fullModuleName": "coin",
            "reason": "with-capability",
            "origin": "transfer",
            "capability": {
              "kind": "defcap",
              "name": "TRANSFER",
              "returnType": "bool",
```


Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
```

Parser Output


```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "withCapabilities": [ "TRANSFER" ],
      "allExtractedCaps": [
          {
            "name": "TRANSFER",
            "fullModuleName": "coin",
            "reason": "with-capability",
            "origin": "transfer",
            "capability": {
              "kind": "defcap",
              "name": "TRANSFER",
              "returnType": "bool",
```


Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
  )
  .addSigner(
    signFor('
    signFor('')
  ]);
```

signFor(capabilityName: "coin.TRANSFER", sen
string, receiver: string, amount: IPactDecima
1/2
ICap

coin.GAS
coin.TRANSFER

Parser Output


```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "withCapabilities": [ "TRANSFER" ],
      "allExtractedCaps": [
          {
            "name": "TRANSFER",
            "fullModuleName": "coin",
            "reason": "with-capability",
            "origin": "transfer",
            "capability": {
              "kind": "defcap",
              "name": "TRANSFER",
              "returnType": "bool",
```


Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
  )
  .addSigner(
    signFor('
    signFor('')
  ]);
```

signFor(capabilityName: "coin.TRANSFER", sen
string, receiver: string, amount: IPactDecima
1/2
ICap

coin.GAS
coin.TRANSFER

Parser Output


```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "withCapabilities": [ "TRANSFER" ],
      "allExtractedCaps": [
          {
            "name": "TRANSFER",
            "fullModuleName": "coin",
            "reason": "with-capability",
            "origin": "transfer",
            "capability": {
              "kind": "defcap",
              "name": "TRANSFER",
              "returnType": "bool",
```


Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) ⇒ string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
  )
  .addSigner('sender-key', (
    signFor('coin.GAS'),
    signFor('coin.TRANSFER', 'sender', 'receiver', {})
  ]);
```

```
signFor(capabilityName: "coin
string, receiver: string, amou
ICap
1/2
```

Parser Output


```
{ "coin": {
    "kind": "module",
    "name": "coin",
    "governance": "GOVERNANCE",
    "functions": [{
      "kind": "defun",
      "name": "transfer",
      "parameters": [
        { "name": "sender", "type": "string" },
        { "name": "receiver", "type": "string" },
        { "name": "amount", "type": "decimal" }
      ],
      "withCapabilities": [ "TRANSFER" ],
      "allExtractedCaps": [
          {
            "name": "TRANSFER",
            "fullModuleName": "coin",
            "reason": "with-capability",
            "origin": "transfer",
            "capability": {
              "kind": "defcap",
              "name": "TRANSFER",
              "returnType": "bool",
```


Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) => string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
  )
  .addSigner('sender-key', (signFor) => [
    signFor('coin.GAS'),
    signFor('coin.TRANSFER', 'sender', 'receiver', {
      decimal: '133.7',
    }),
  ]);
```

Generated ts definition

```ts
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) => string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

Client Code

```ts
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
  )
  .addSigner('sender-key', (signFor) => [
    signFor('coin.GAS'),
    signFor('coin.TRANSFER', 'sender', 'receiver', {
      decimal: '133.7',
    }),
  ]);
```

Generated ts definition

```
interface ICapability_Coin_GAS {
  (name: 'coin.GAS'): ICap;
}
interface ICapability_transfer {
  (capabilityName: "coin.TRANSFER",
    sender: string, receiver: string, amount: IPactDecimal): ICap
}
declare module '@kadena/client' {
  export interface IPactModules {
    "coin": {
      "transfer": (
        sender: string, receiver: string, amount: IPactDecimal
      ) => string
        & { capability : ICapability_transfer & ICapability_Coin_GAS }
```

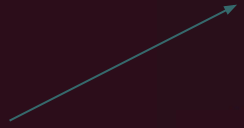
Client Code

```
Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
  )
  .addSigner('sender-key', (signFor) => [
    signFor('coin.GAS'),
    signFor('coin.TRANSFER', 'sender', 'receiver', {
      decimal: '133.7',
    }),
  ]);
```

Transaction building with

Recap

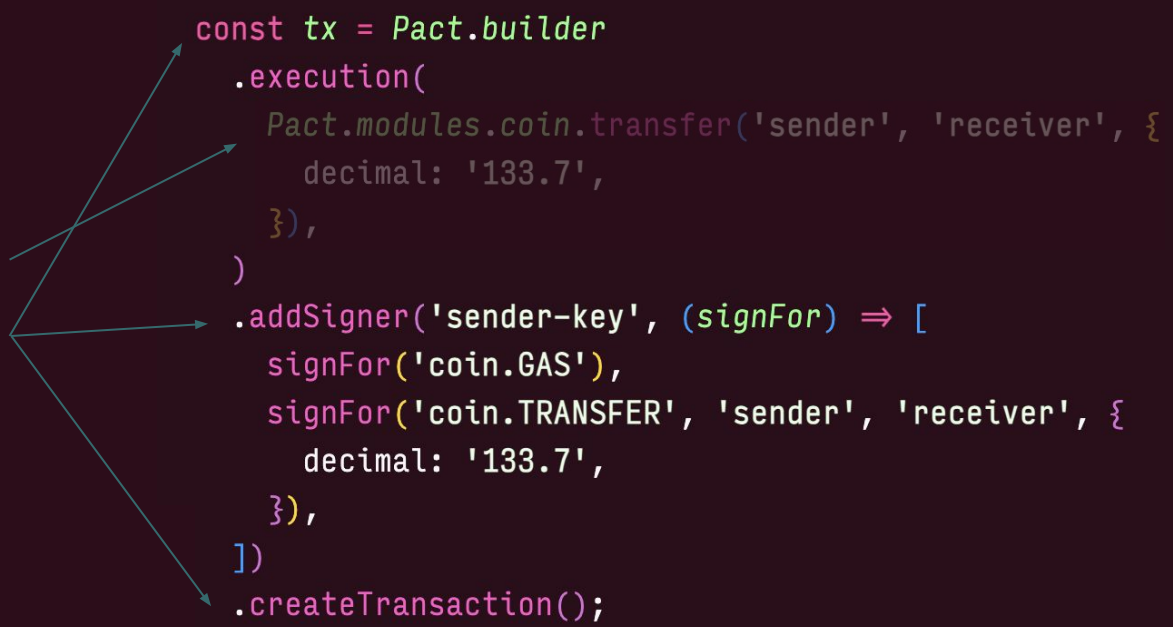
- Expression builder



```
Pact.modules.coin.transfer('sender', 'receiver', {
  decimal: '133.7',
}),
```

Recap

- Expression builder
- Transaction builder

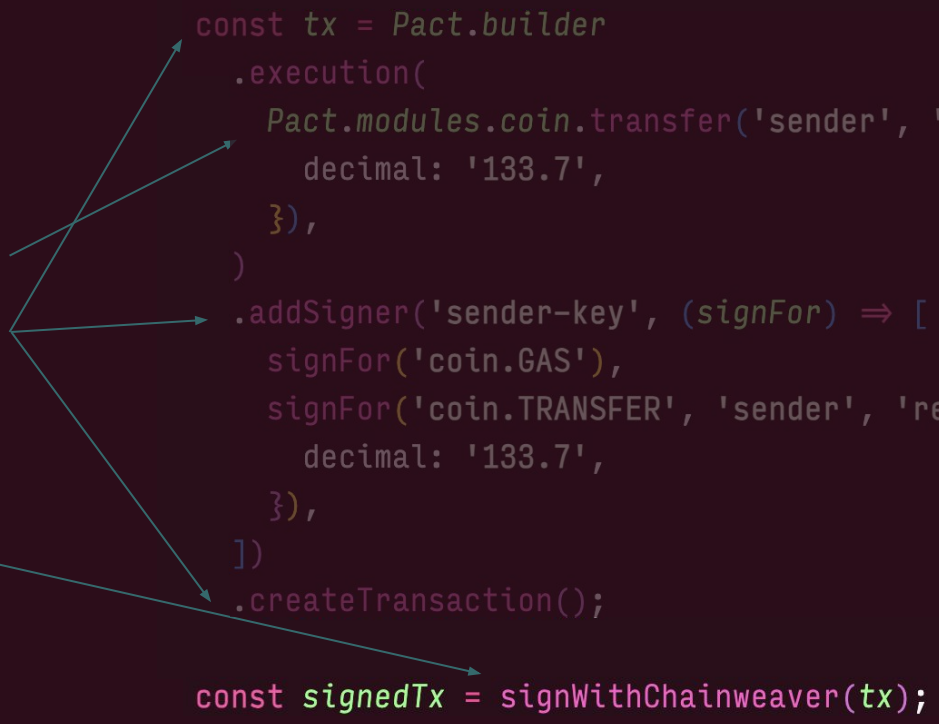


```
const tx = Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
  )
  .addSigner('sender-key', (signFor) => [
    signFor('coin.GAS'),
    signFor('coin.TRANSFER', 'sender', 'receiver', {
      decimal: '133.7',
    }),
  ])
  .createTransaction();
```

Recap

- Expression builder
- Transaction builder

- Signing utilities



Recap

- Expression builder
- Transaction builder
- Signing utilities
- Client for Blockchain

```
const tx = Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
  )
  .addSigner('sender-key', (signFor) => [
    signFor('coin.GAS'),
    signFor('coin.TRANSFER', 'sender', 'receiver', {
      decimal: '133.7',
    }),
  ])
  .createTransaction();

const signedTx = signWithChainweaver(tx);
const { submit, listen } = createClient();
```

Recap

- Expression builder
- Transaction builder

```
const tx = Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
  )
  .addSigner('sender-key', (signFor) => [
    signFor('coin.GAS'),
    signFor('coin.TRANSFER', 'sender', 'receiver', {
      decimal: '133.7',
    }),
  ])
  .createTransaction();
```

- Signing utilities

```
const signedTx = signWithChainweaver(tx);
```

- Client for Blockchain
  - `submit` tx to the blockchain
  - `listen` for it to be verified

```
const { submit, listen } = createClient();

const finishedTx = await listen(await submit(signedTx));
console.log('Finished!', finishedTx);
```

Recap

- Expression builder
- Transaction builder
- Signing utilities
- Client for Blockchain
  - `submit` tx to the blockchain
  - `listen` for it to be verified

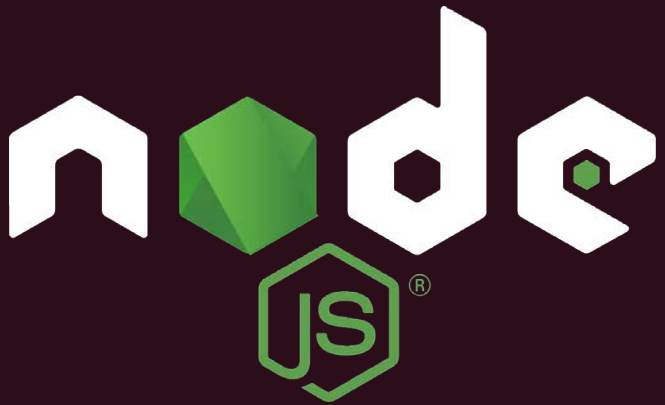
```
const tx = Pact.builder
  .execution(
    Pact.modules.coin.transfer('sender', 'receiver', {
      decimal: '133.7',
    }),
  )
  .addSigner('sender-key', (signFor) => [
    signFor('coin.GAS'),
    signFor('coin.TRANSFER', 'sender', 'receiver', {
      decimal: '133.7',
    }),
  ])
  .createTransaction();

const signedTx = signWithChainweaver(tx);
const { submit, listen } = createClient();

const finishedTx = await listen(await submit(signedTx));
console.log('Finished!', finishedTx);
```

Signing Transactions

Koala Wallet

WalletConnect

Blockchain Client



Frontend
is closer to
Blockchain
than you
thought

Signing Transactions

Koala Wallet

WalletConnect

KADENA

Thank you!

Albert Groothedde
Architect Developer Experience
@alber70g
github.com/alber70g

Kadena
Proof of Work Scalable Blockchain
@kadena
github.com/kadena-io
/kadena-community