Formal modeling at Informal Systems

Manuel Bravo <manuel@informal.systems>

What we do

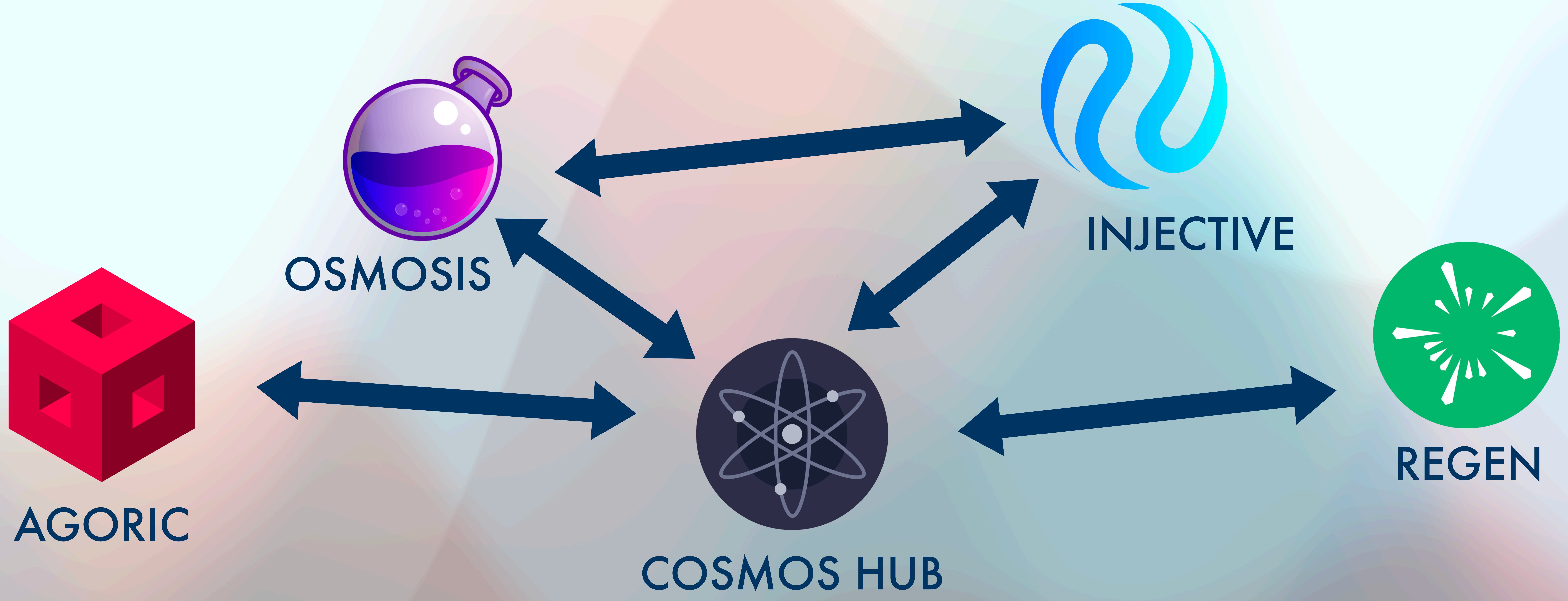
What we do

We help to build confidence in the Cosmos ecosystem (aka the interchain)

Wait, what's the interchain?

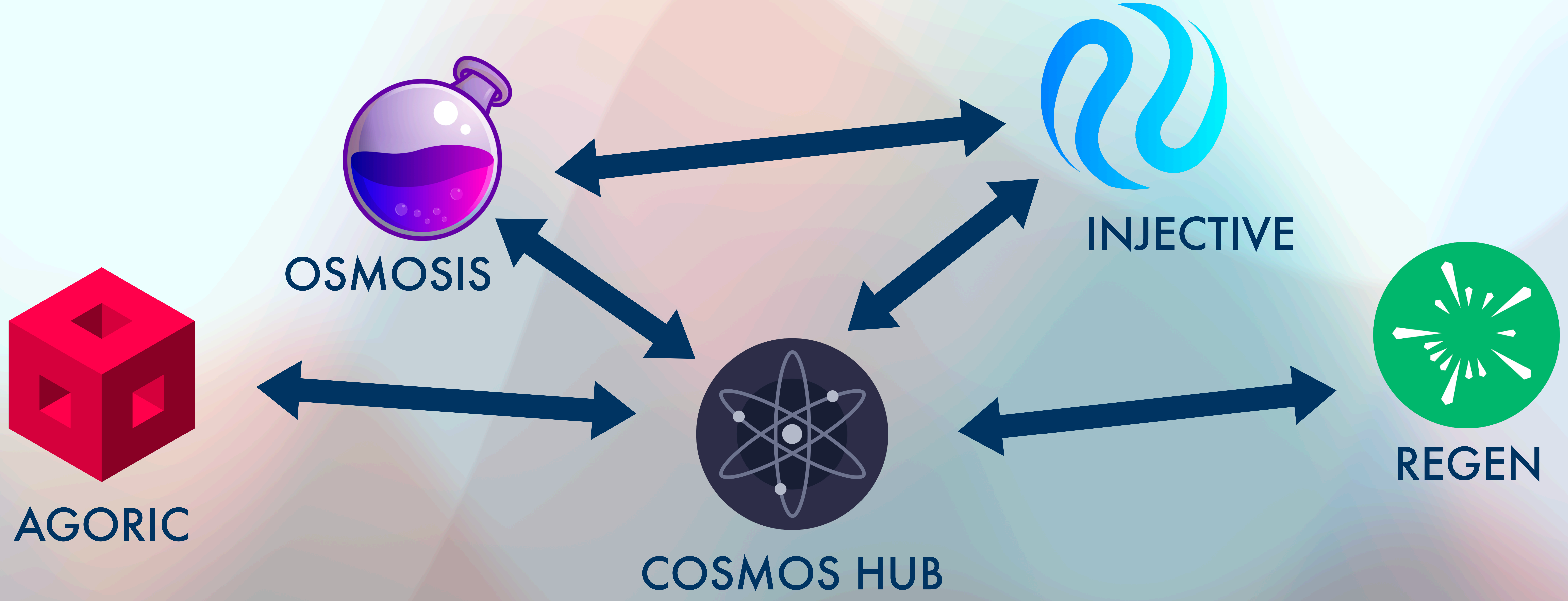
The interchain: the Internet of Blockchains

A network of blockchains able to communicate with each other in a decentralized way



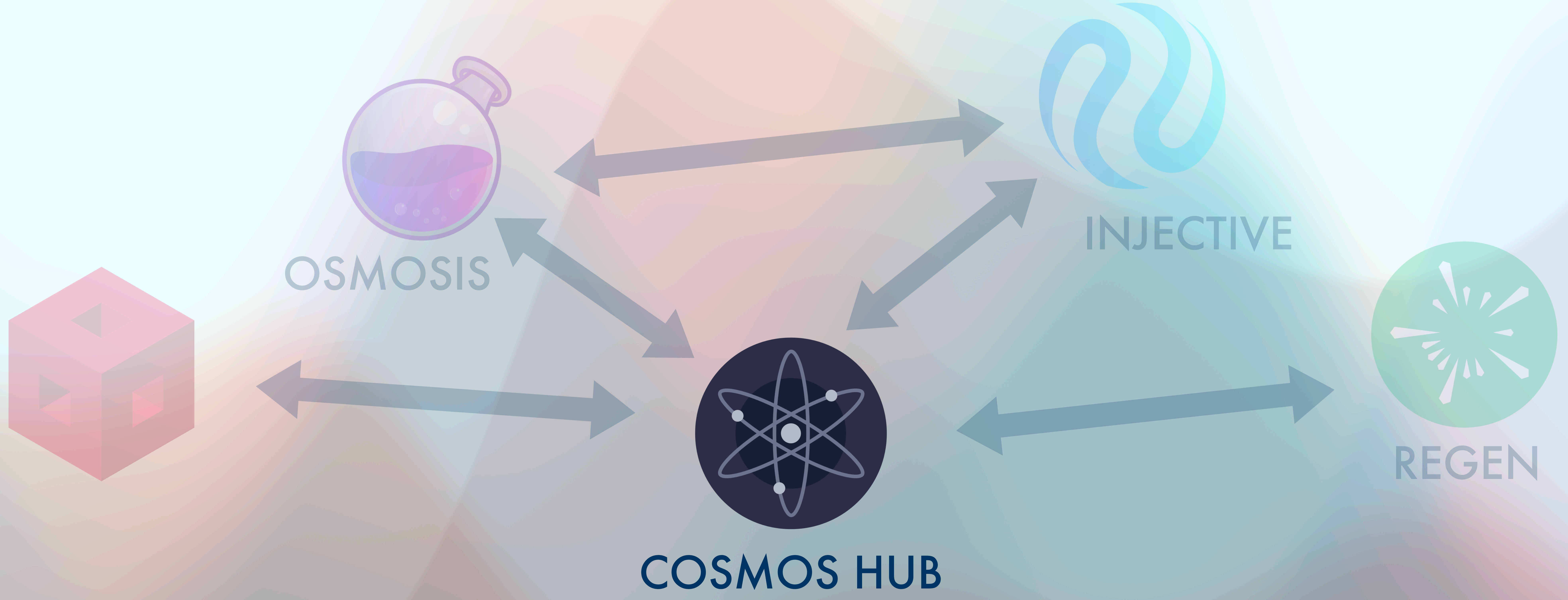
The interchain: the Internet of Blockchains

248+ apps and services

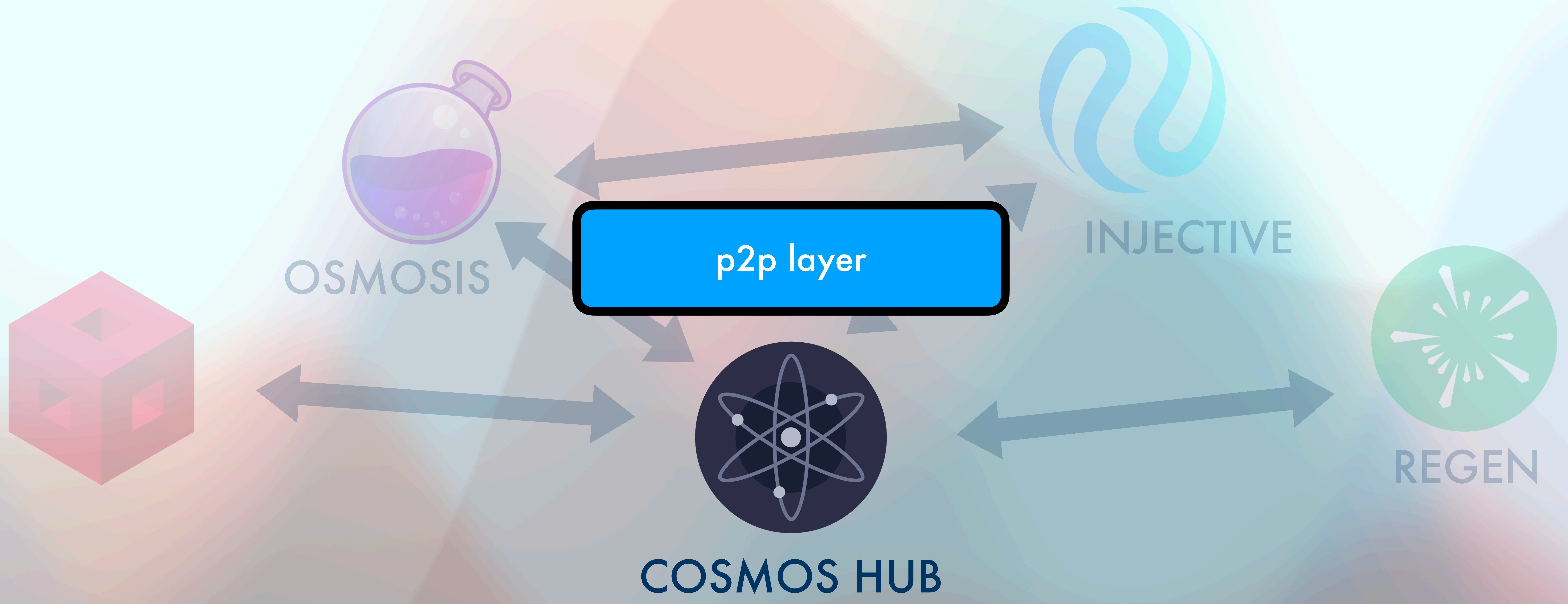


Distributed protocols everywhere!

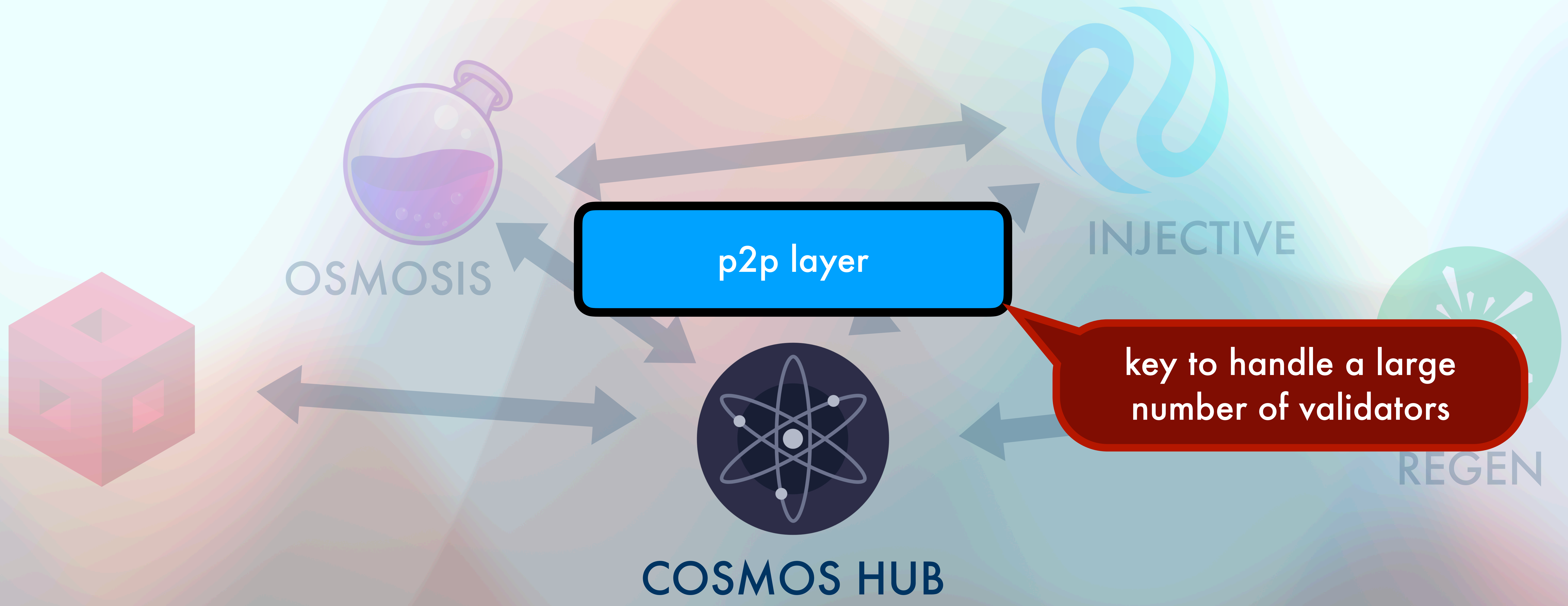
The interchain: the Internet of Blockchains



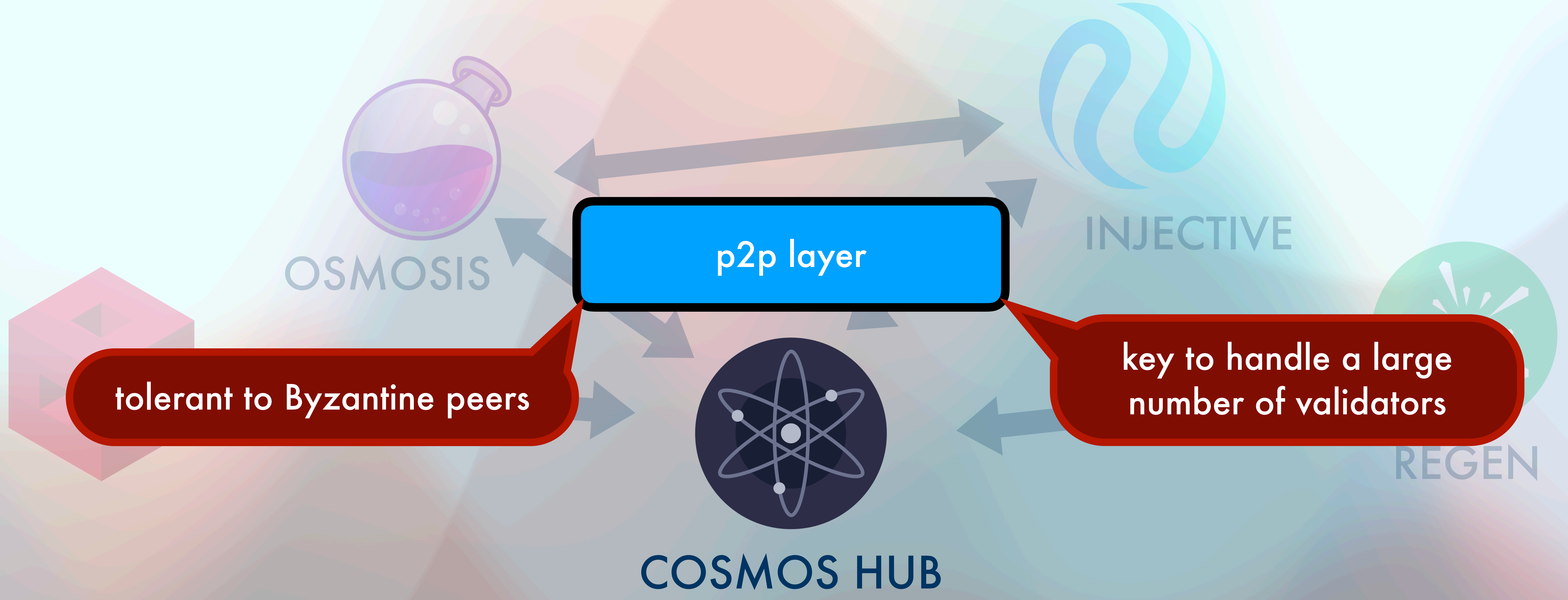
The interchain: the Internet of Blockchains



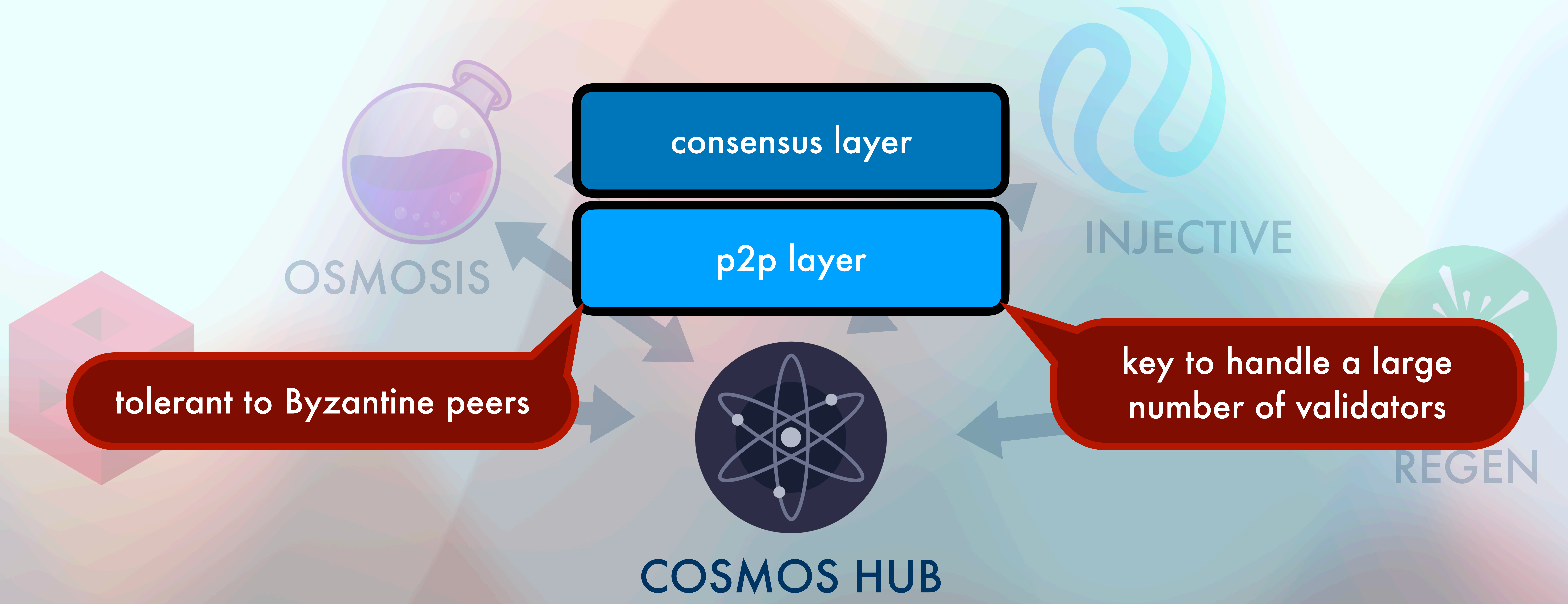
The interchain: the Internet of Blockchains



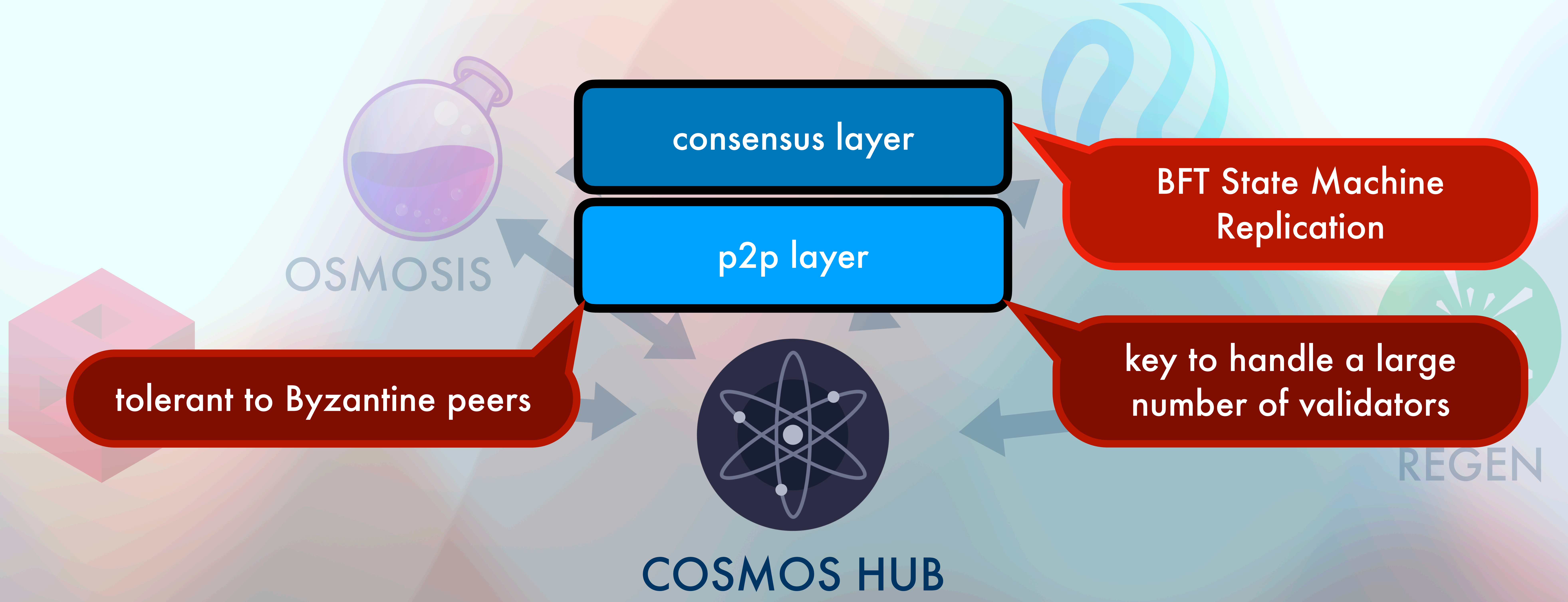
The interchain: the Internet of Blockchains



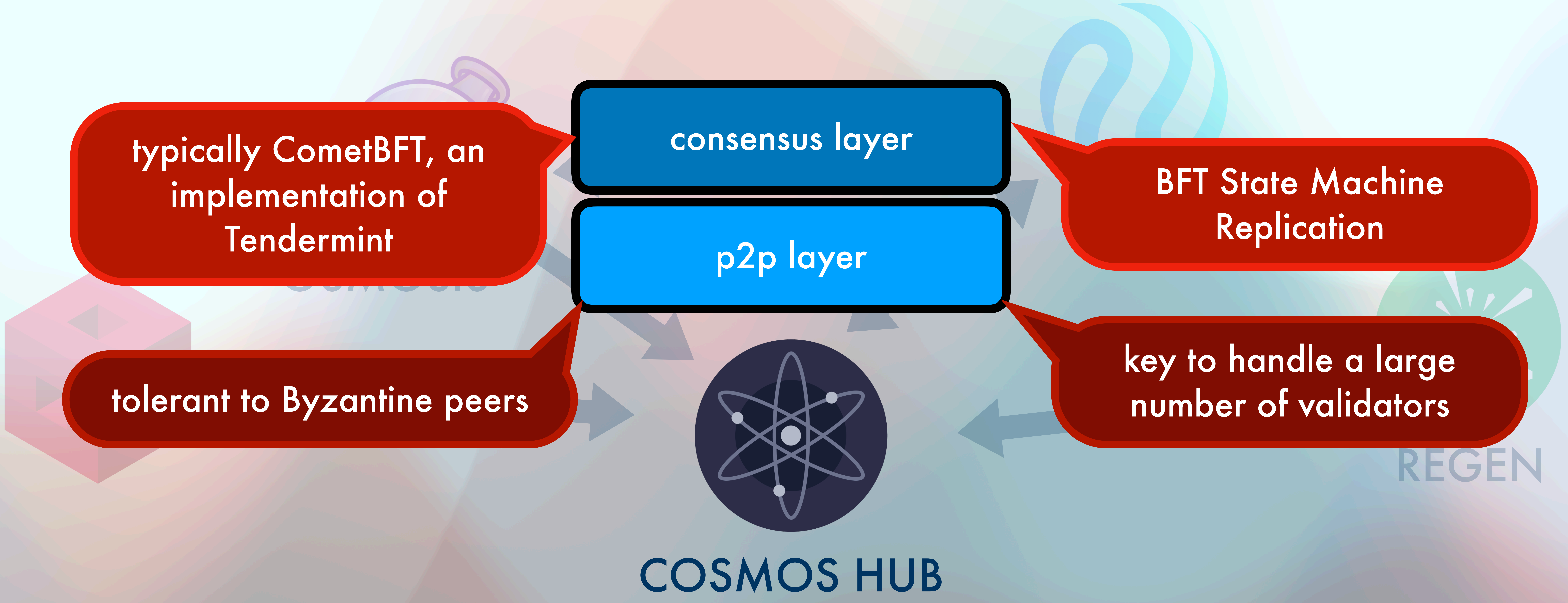
The interchain: the Internet of Blockchains



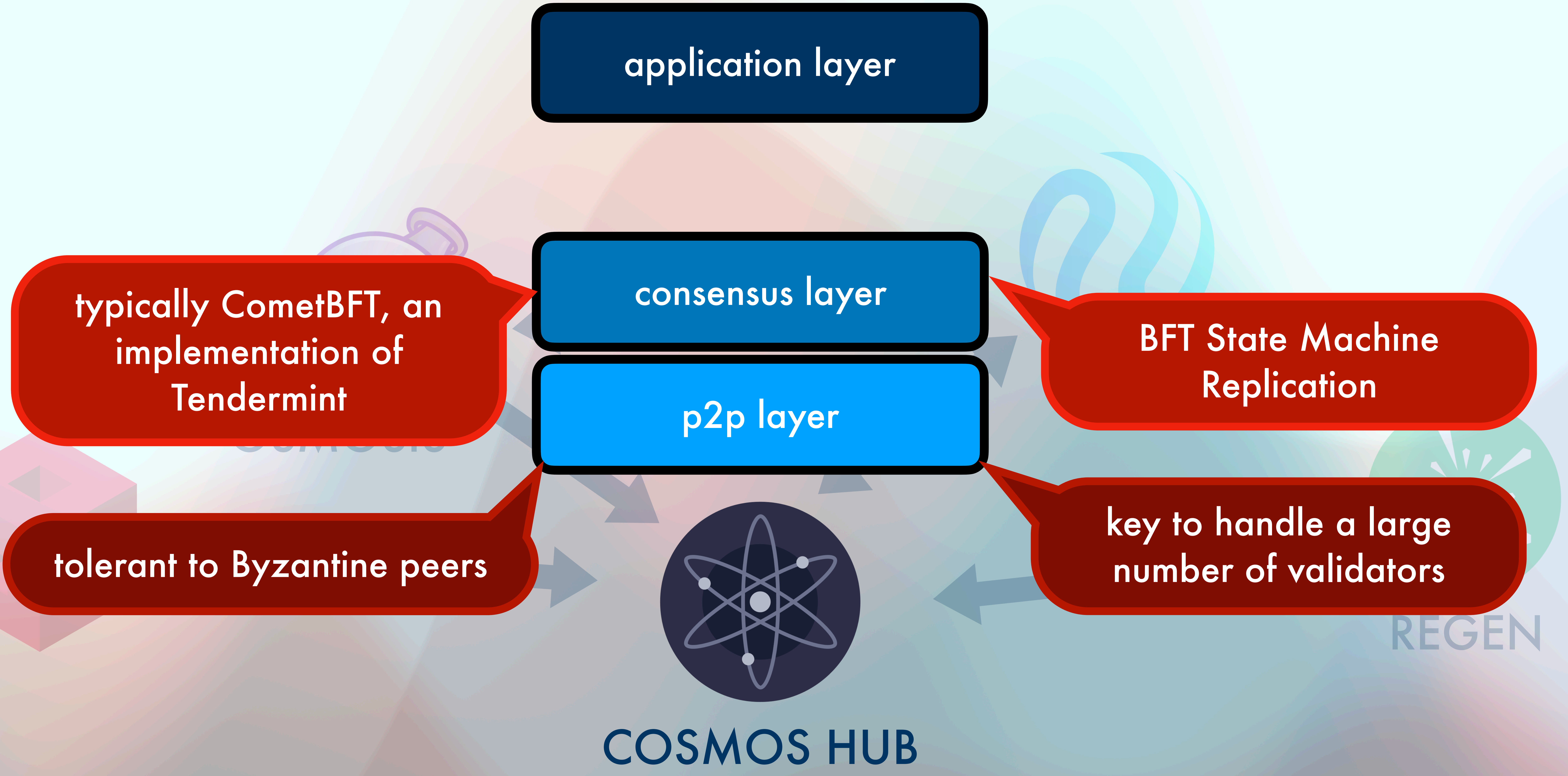
The interchain: the Internet of Blockchains



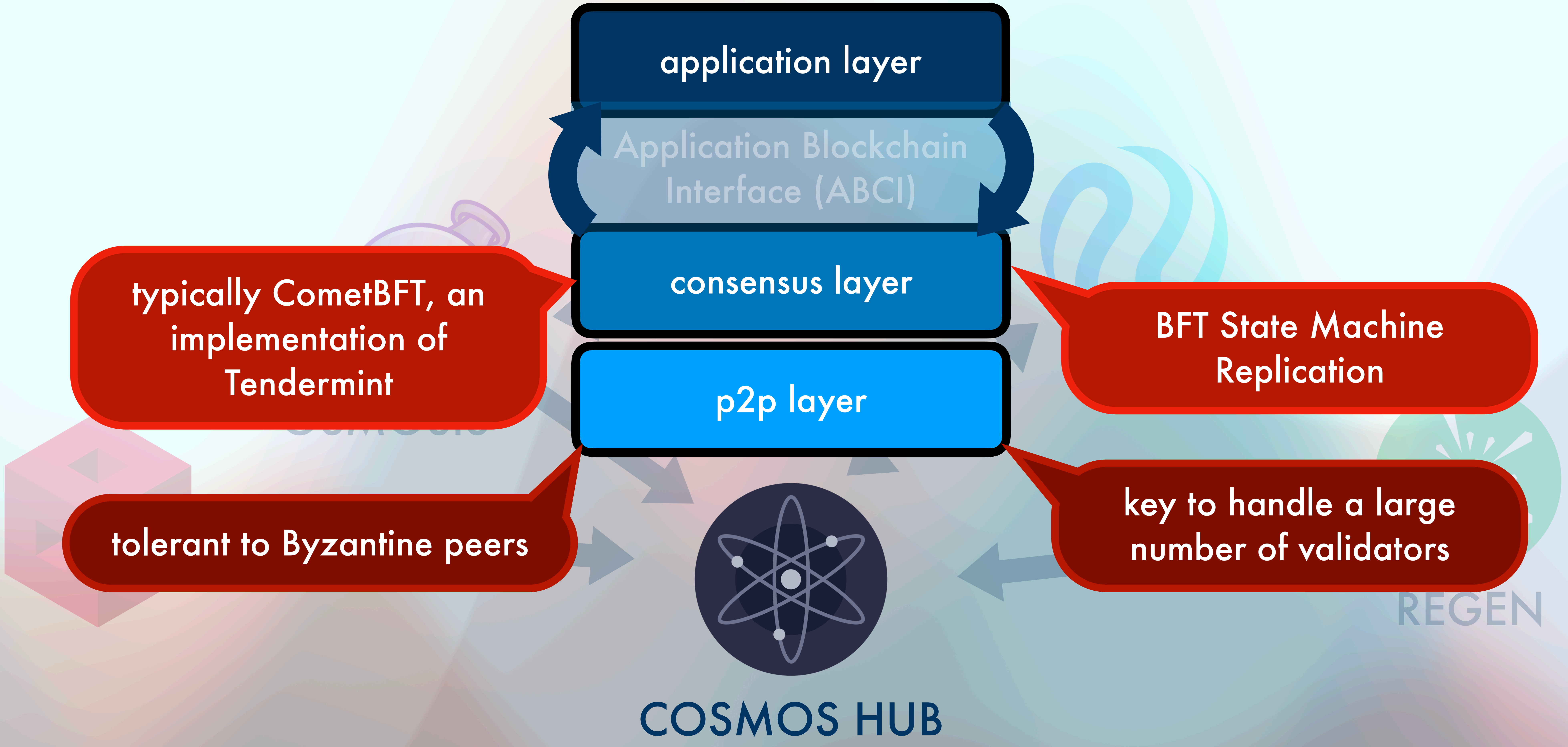
The interchain: the Internet of Blockchains



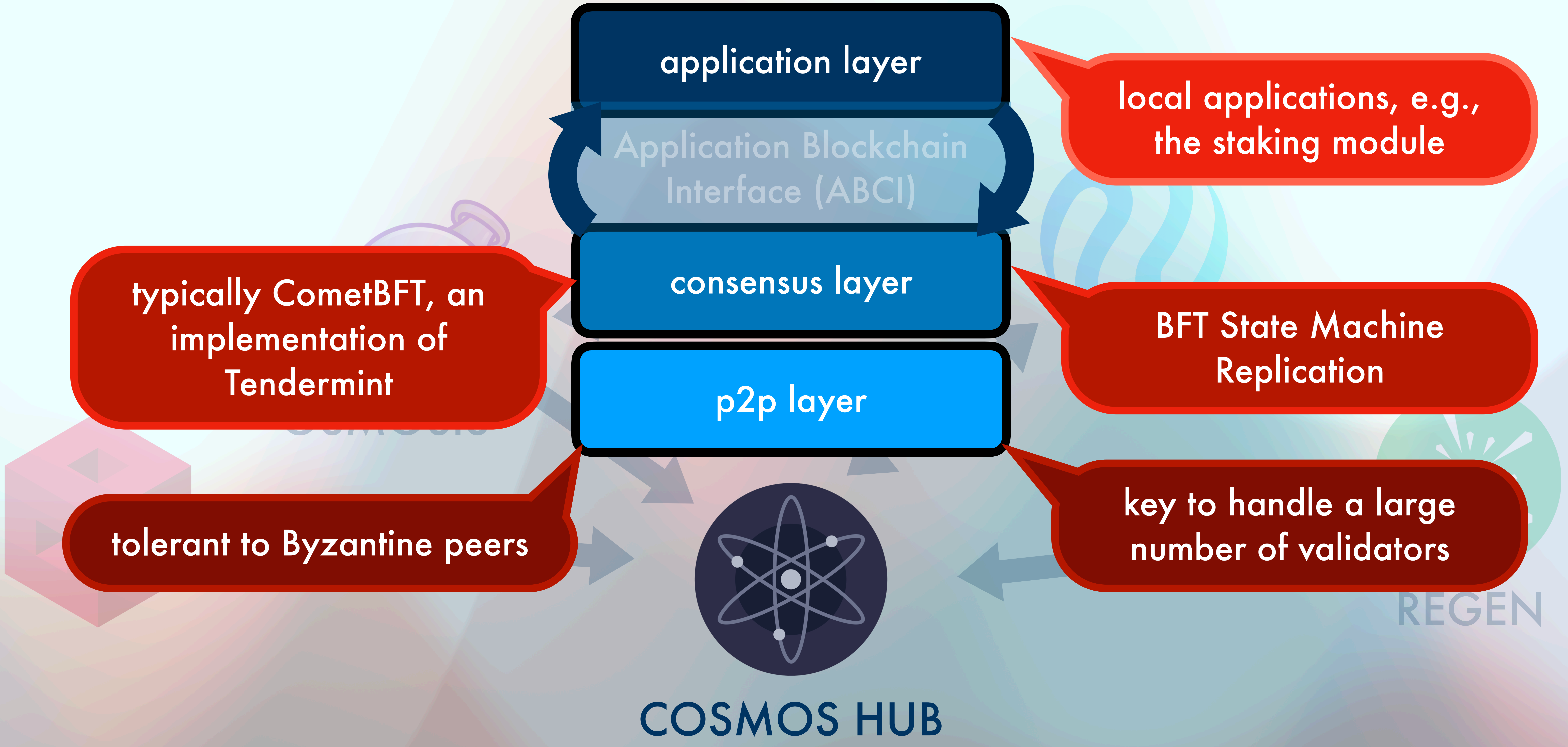
The interchain: the Internet of Blockchains



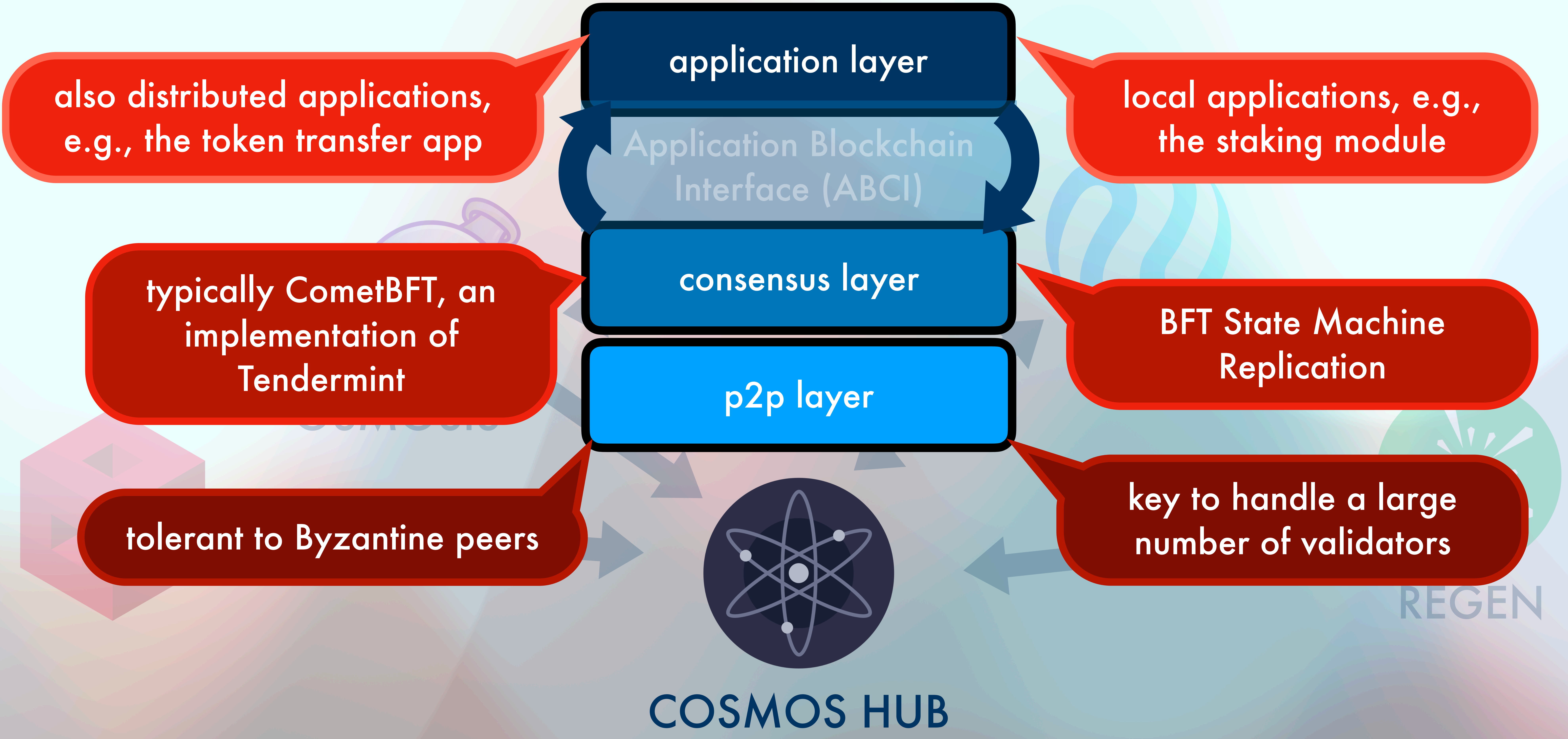
The interchain: the Internet of Blockchains



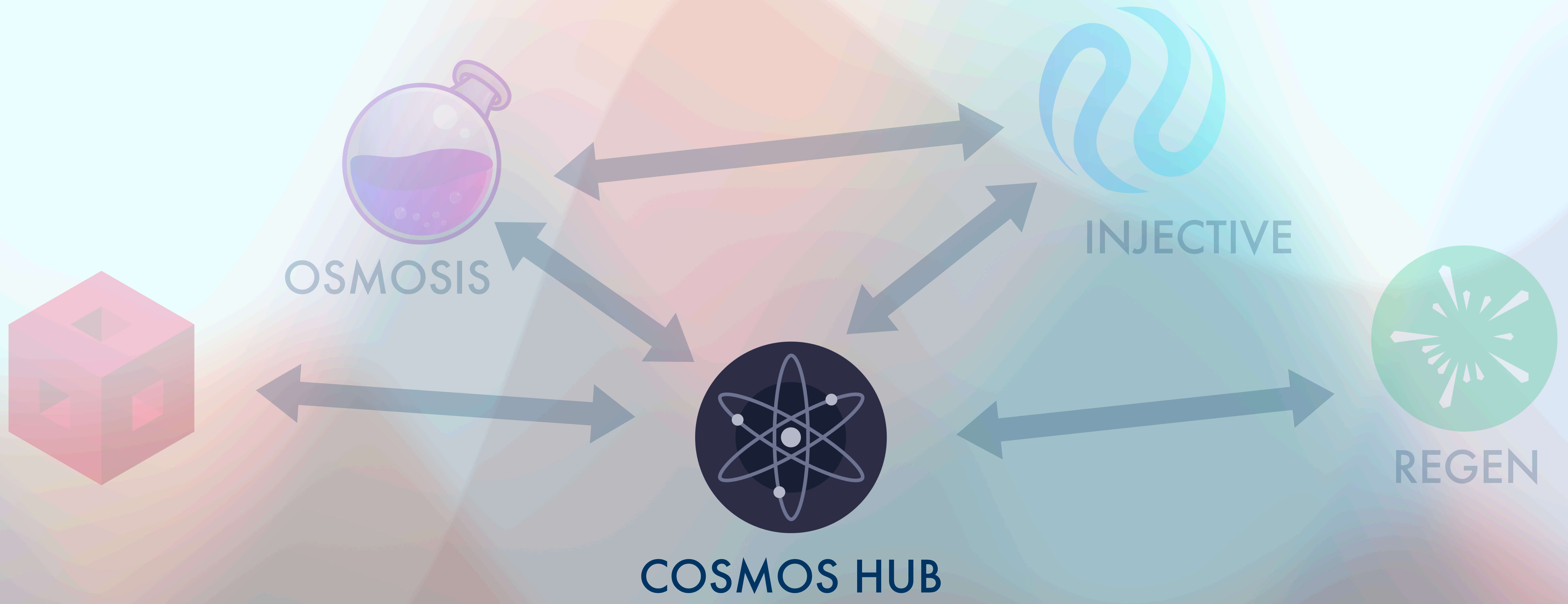
The interchain: the Internet of Blockchains



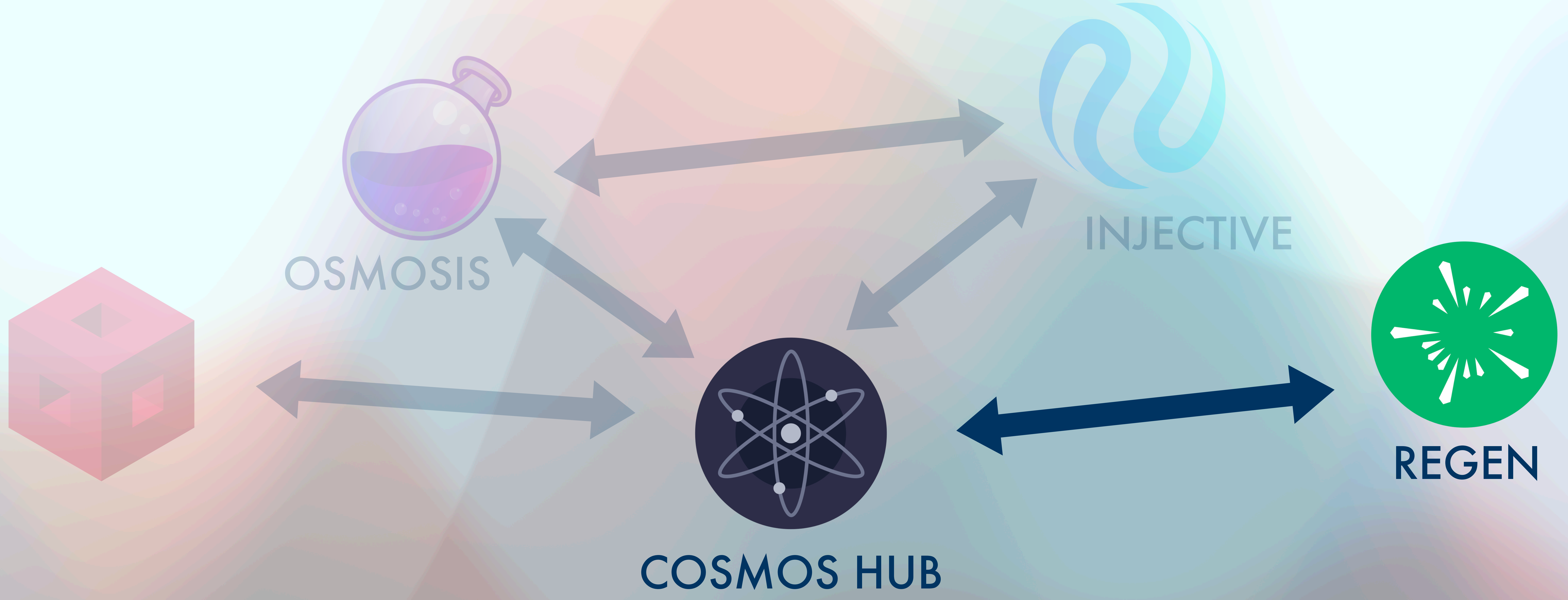
The interchain: the Internet of Blockchains



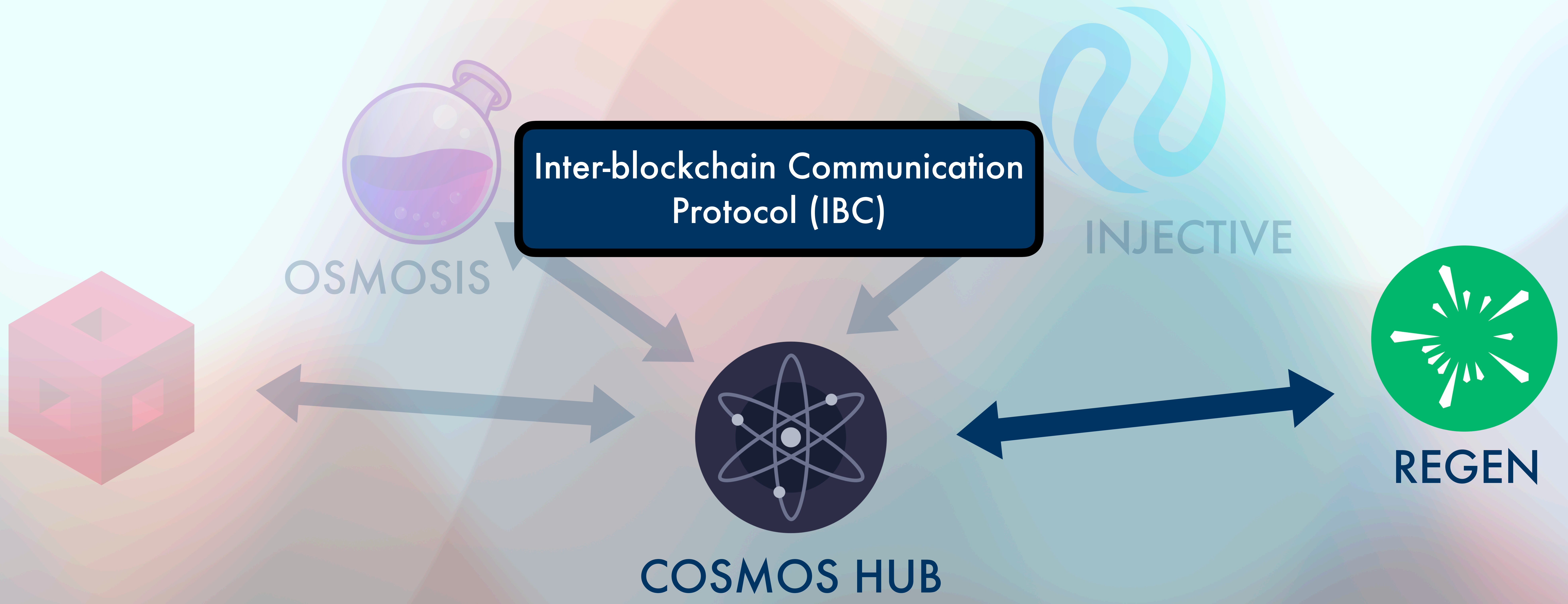
The interchain: the Internet of Blockchains



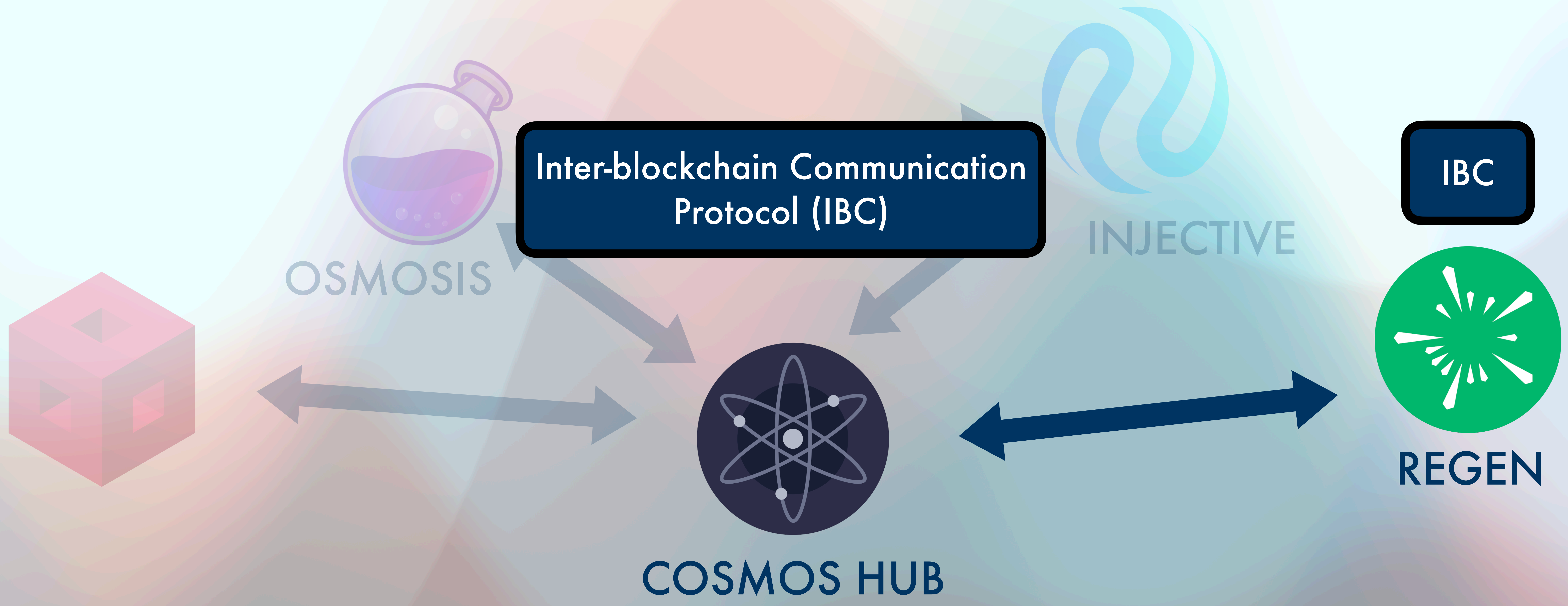
The interchain: the Internet of Blockchains



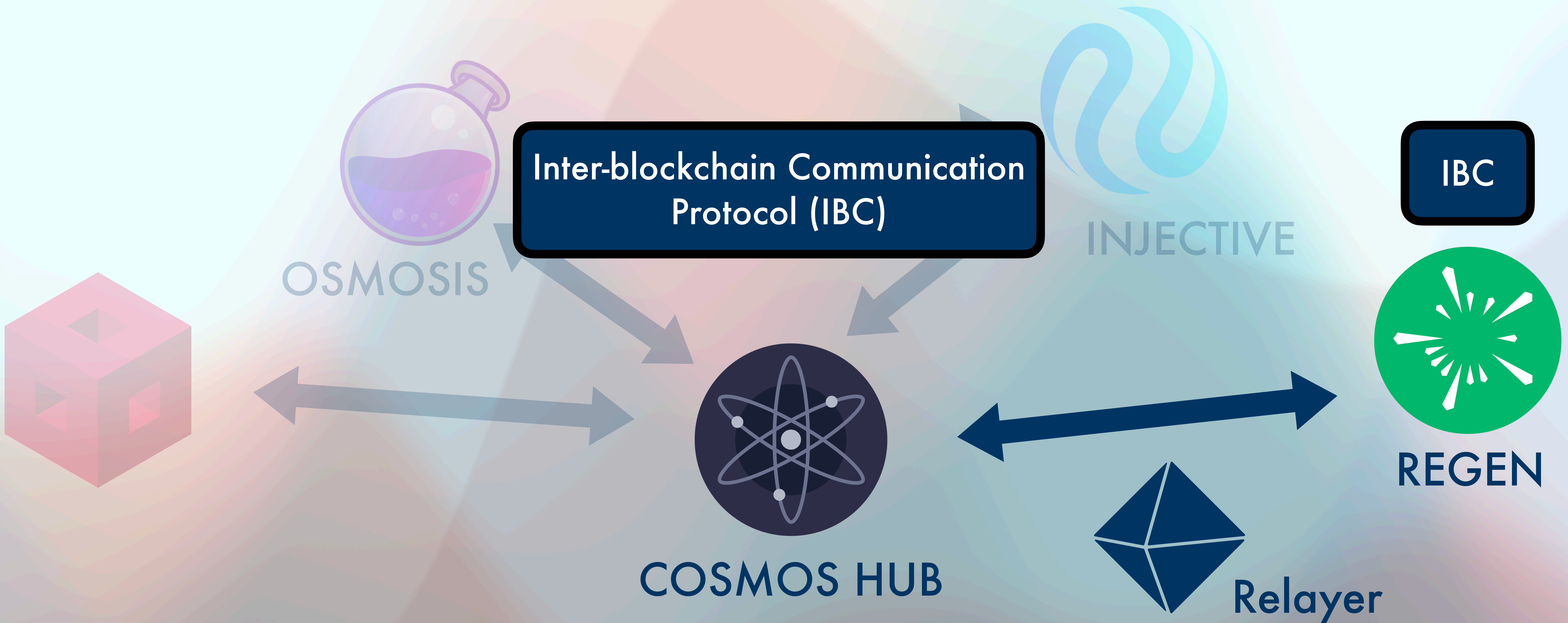
The interchain: the Internet of Blockchains



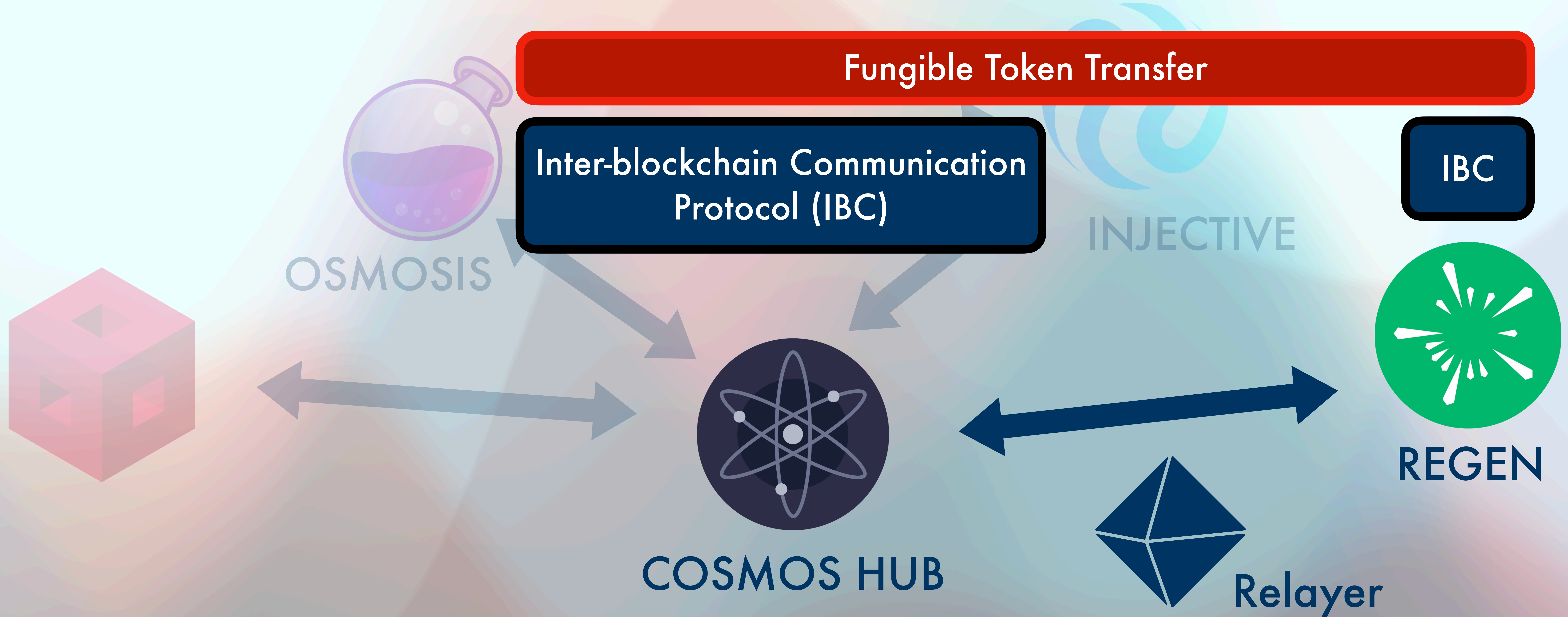
The interchain: the Internet of Blockchains



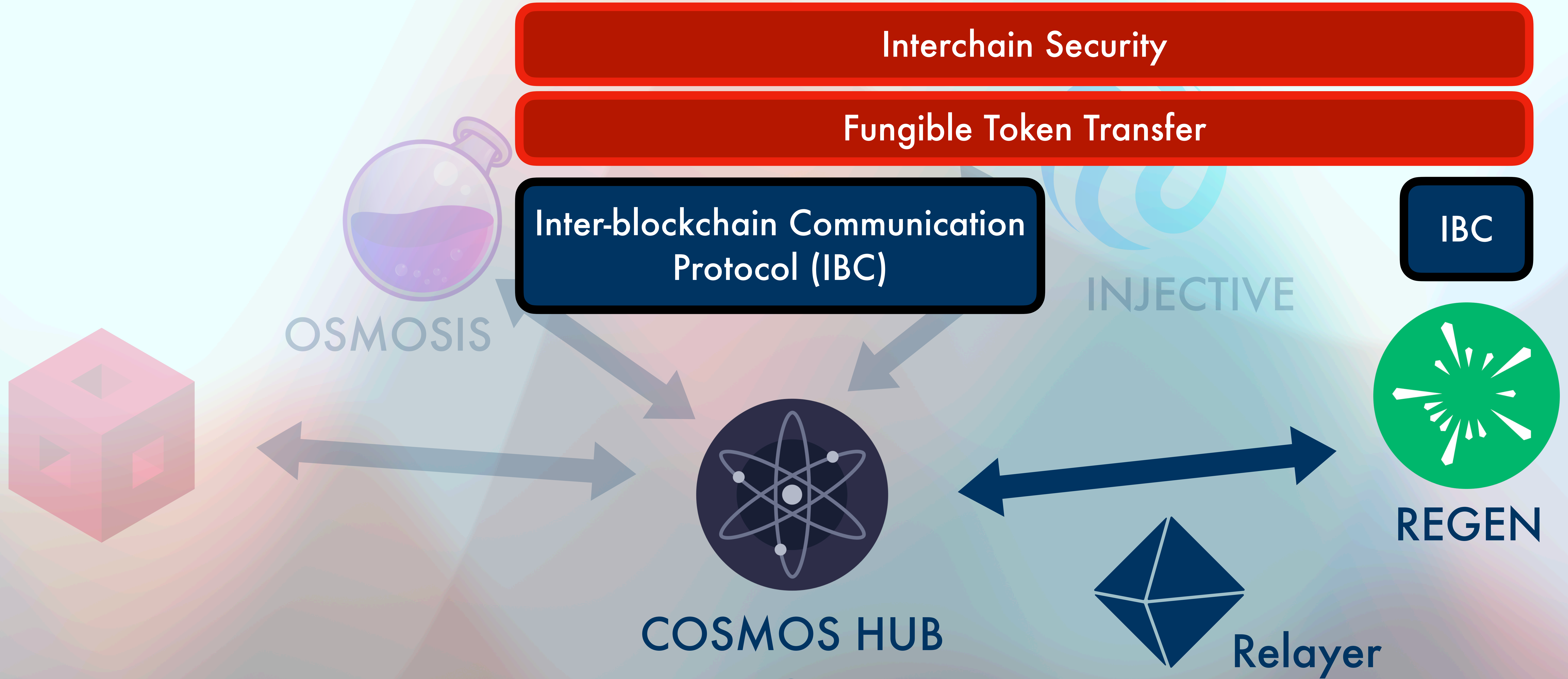
The interchain: the Internet of Blockchains



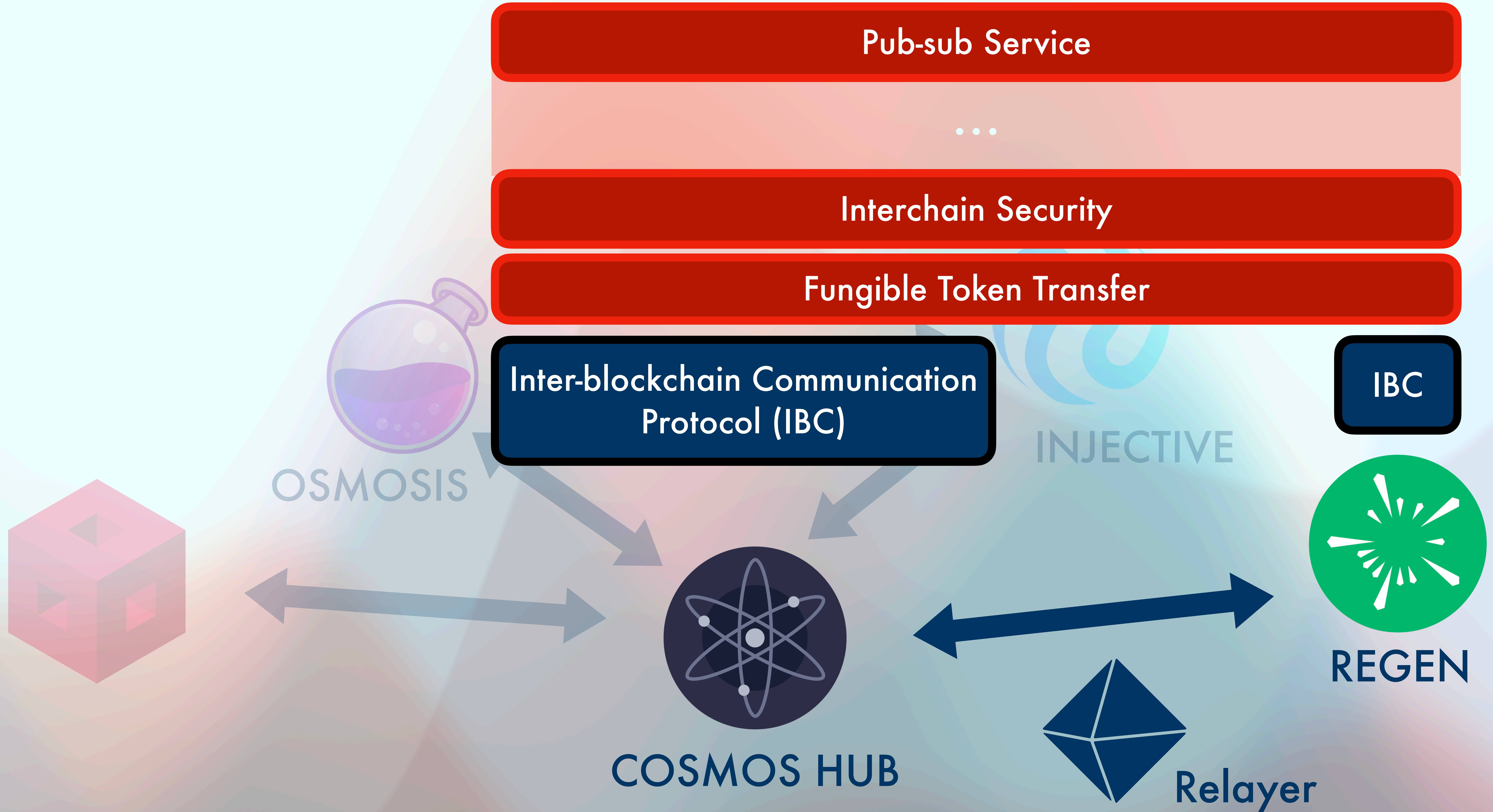
The interchain: the Internet of Blockchains



The interchain: the Internet of Blockchains



The interchain: the Internet of Blockchains



What we do

We help to build confidence in the Cosmos ecosystem (aka the interchain)

What we do

We help to build confidence in the Cosmos ecosystem (aka the interchain)

How we do it

What we do

We help to build confidence in the Cosmos ecosystem (aka the interchain)

How we do it

1. Stewarding critical software components

What we do

We help to build confidence in the
Cosmos ecosystem (aka the interchain)

How we do it

1. Stewarding critical software components
2. Conducting security audits to key projects

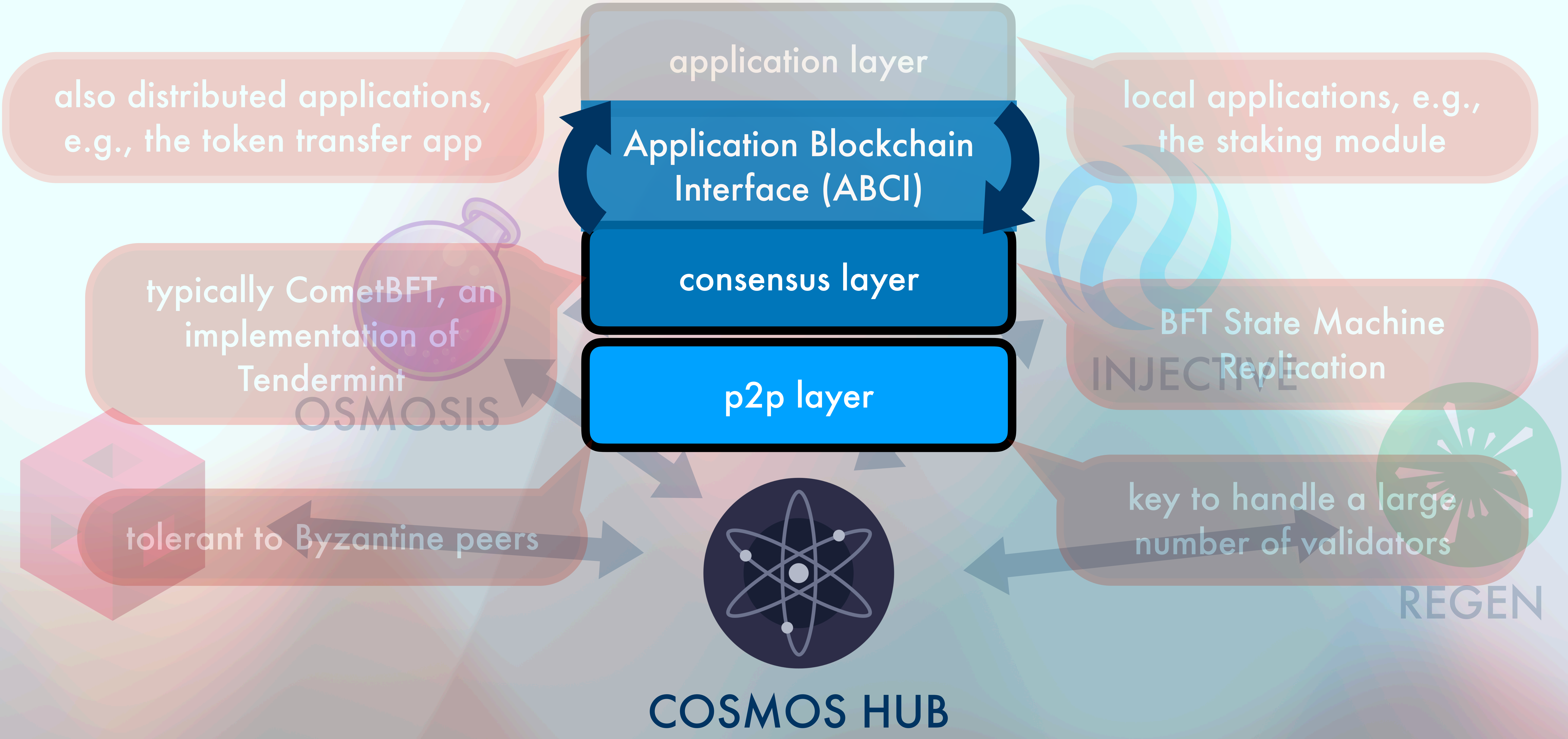
What we do

We help to build confidence in the
Cosmos ecosystem (aka the interchain)

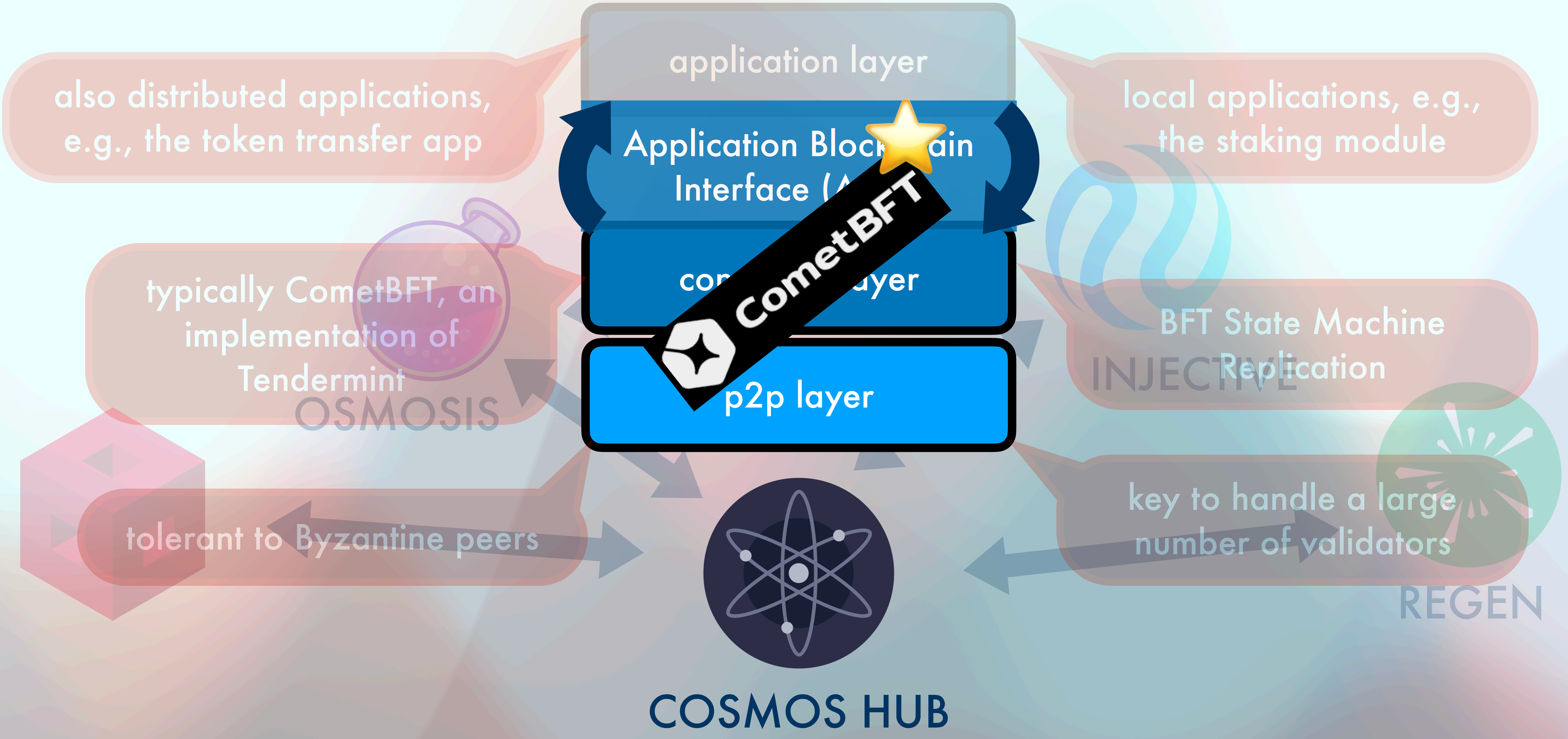
How we do it

1. Stewarding critical software components
2. Conducting security audits to key projects
3. Securing chains by reliably validating

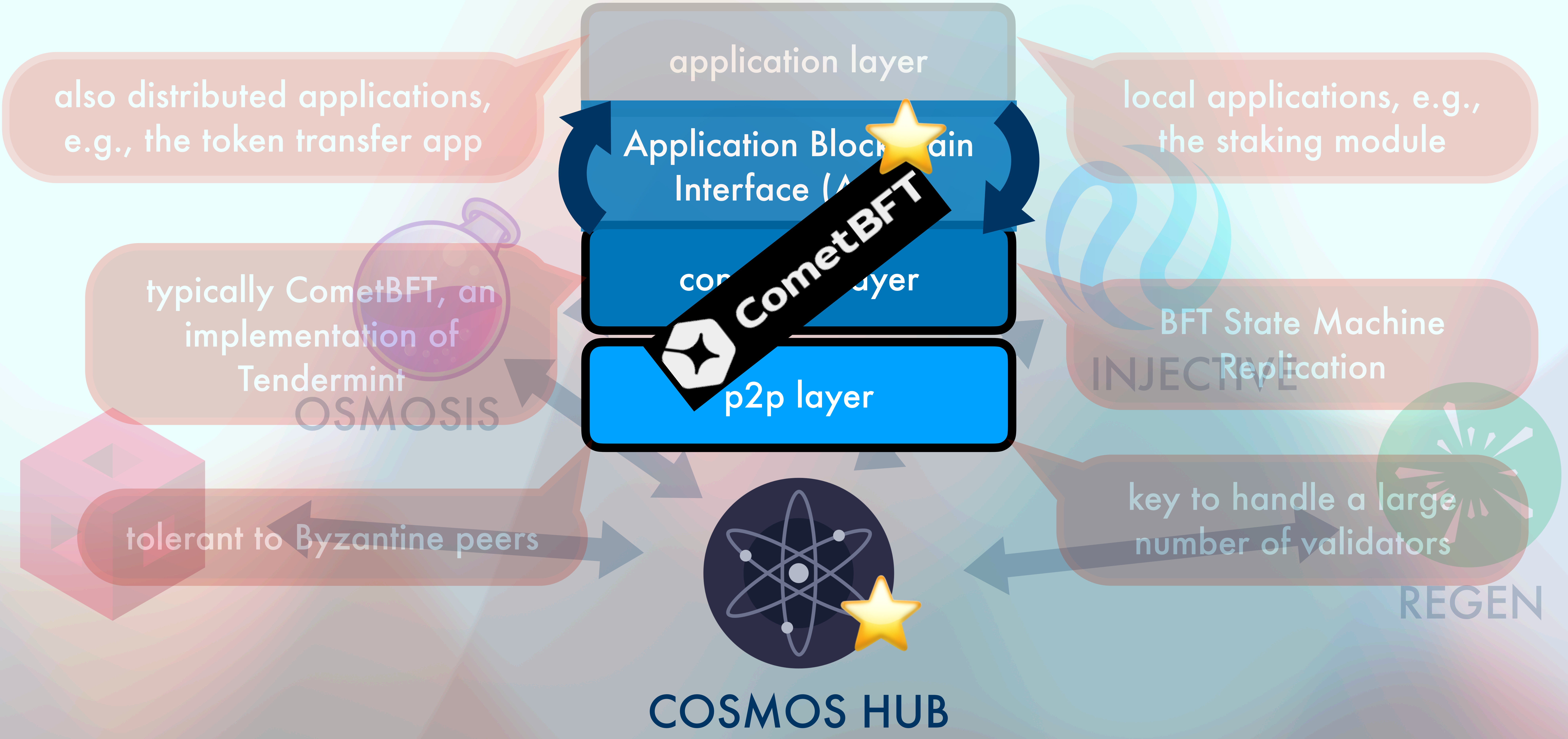
How we do it: #1 stewarding



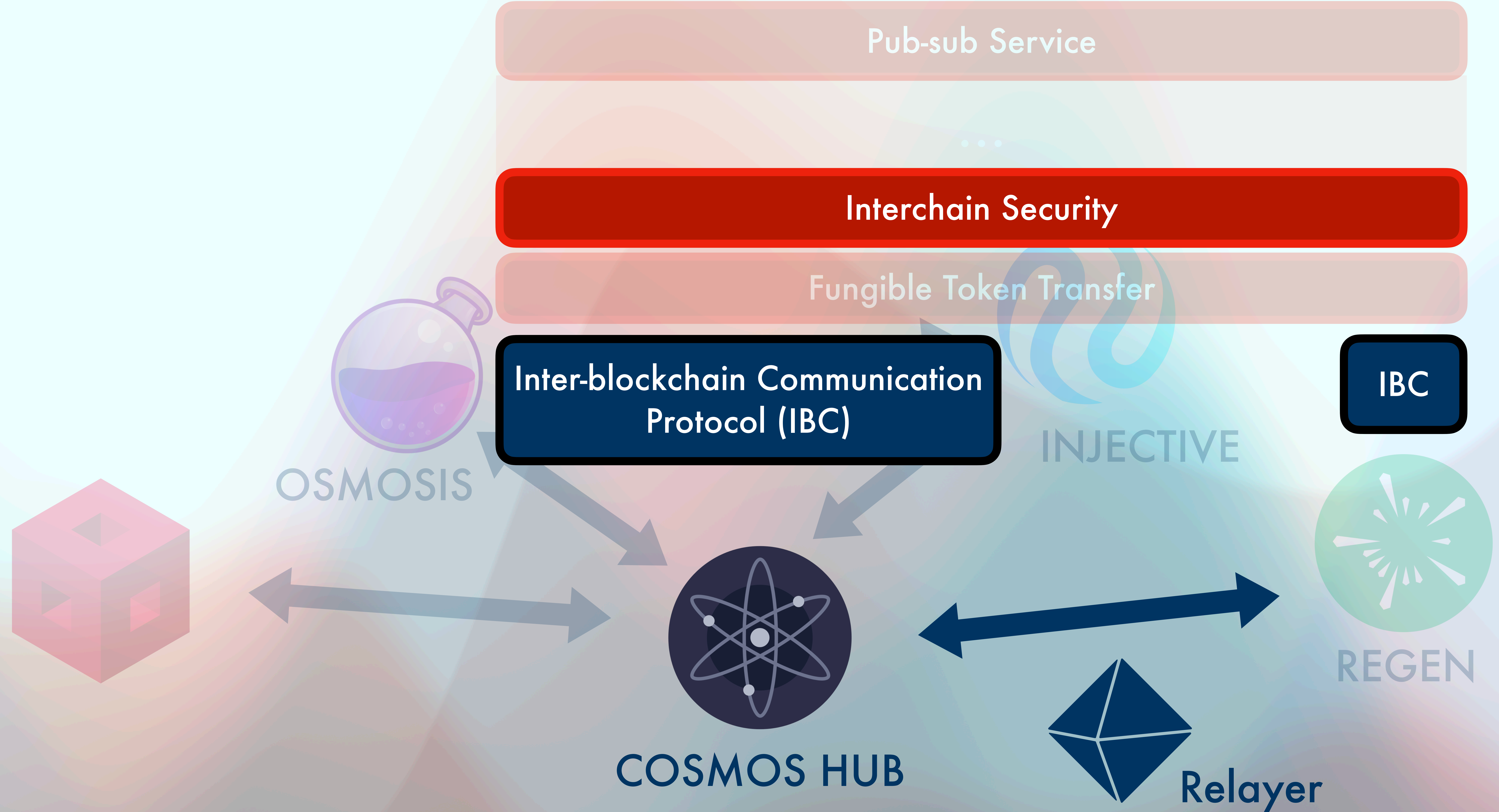
How we do it: #1 stewarding



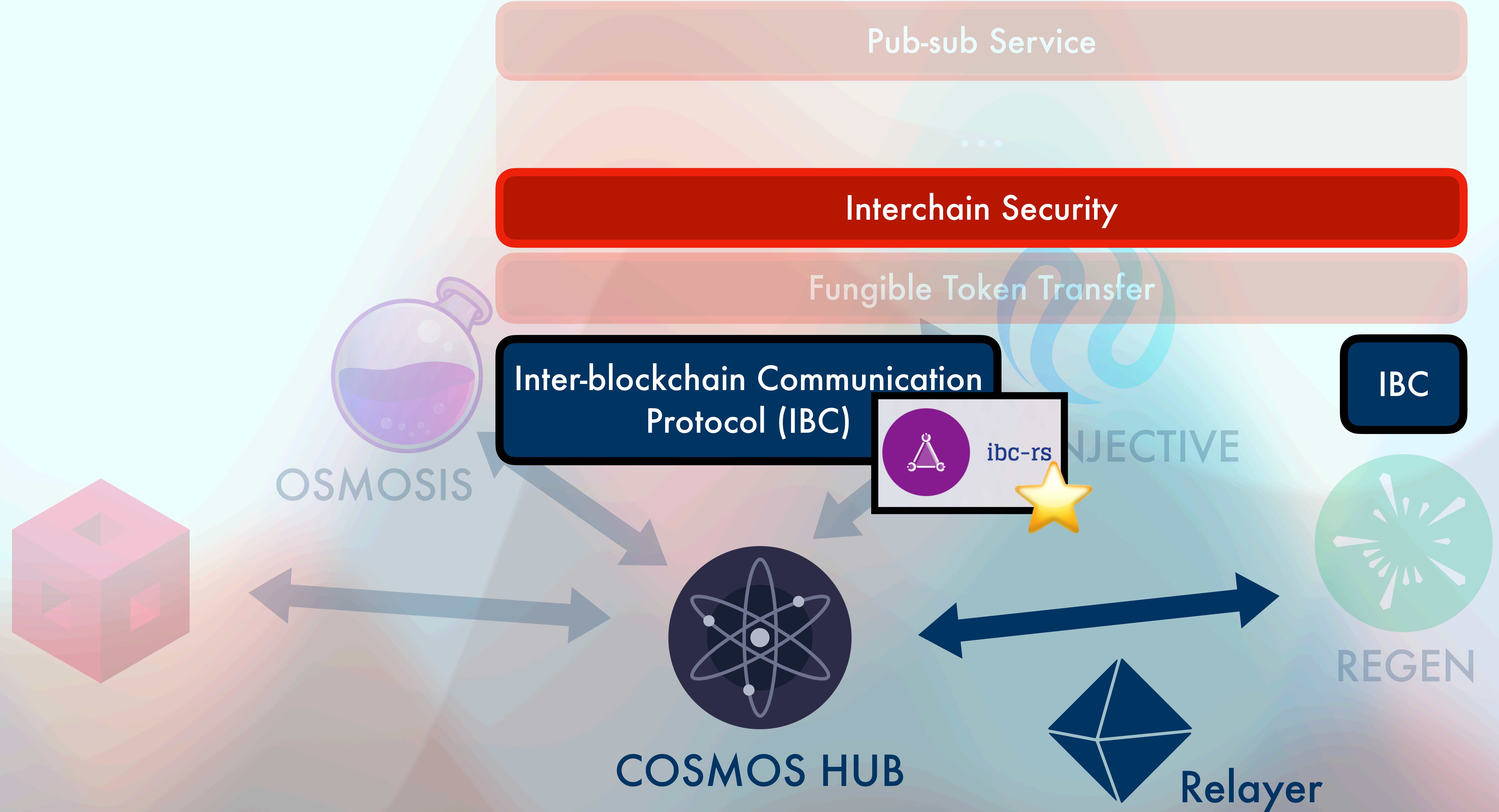
How we do it: #1 stewarding



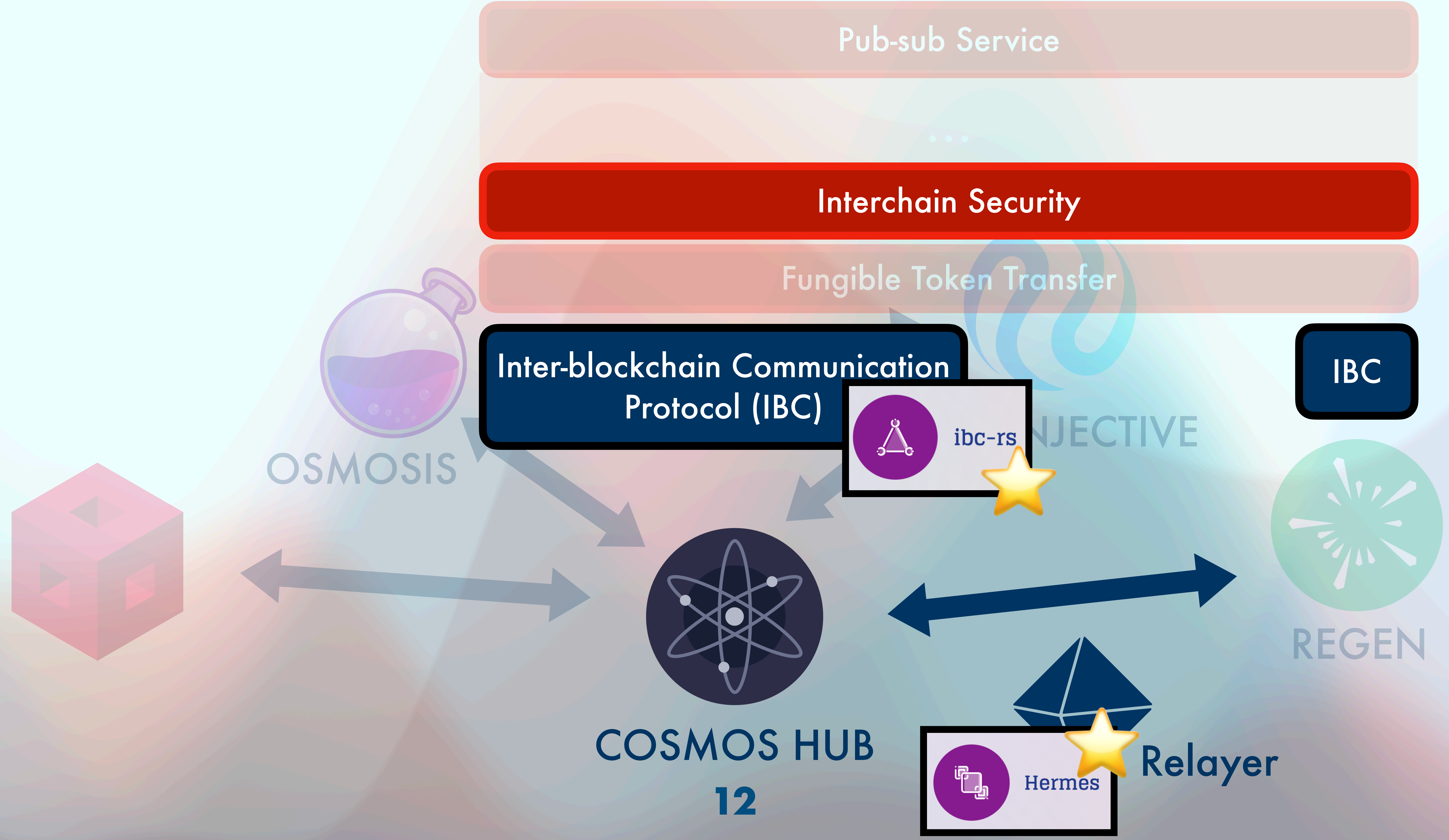
How we do it: #1 stewarding



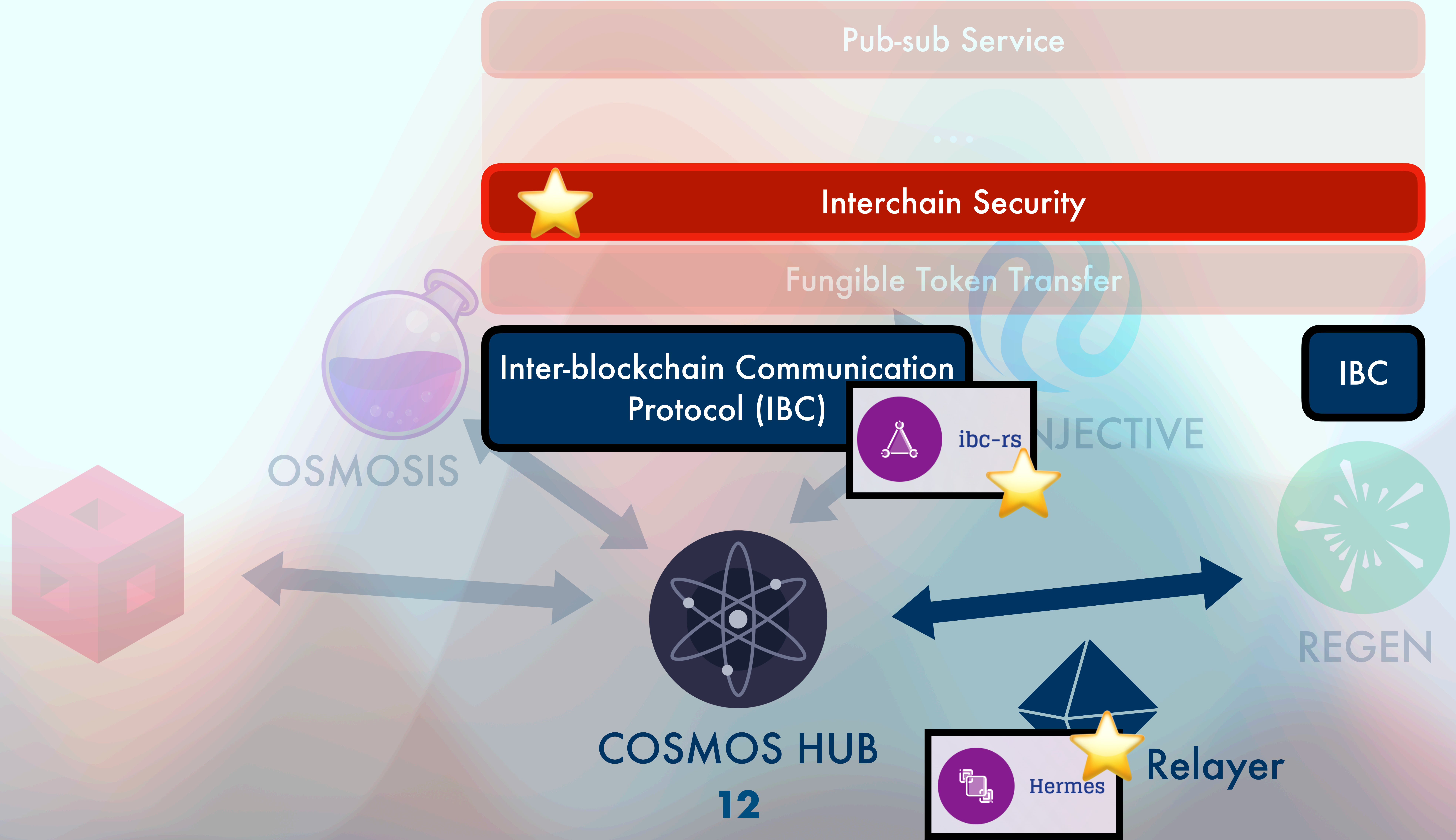
How we do it: #1 stewarding



How we do it: #1 stewarding



How we do it: #1 stewarding



How we do it: #2 security audits

How we do it: #2 security audits

We offer code review, and protocol design services: protocol design, formalization and analysis

How we do it: #2 security audits

We offer code review, and protocol design services: protocol design, formalization and analysis

We leverage formal methods and tools to make distributed systems secure and resilient

How we do it: protocol design service

How we do it: protocol design service

E.g., a client has a protocol and want help formalizing it

How we do it: protocol design service

E.g., a client has a protocol and want help formalizing it

We produce a specification artifact that includes a formal specification with a set of desired properties and make assumptions explicit

How we do it: protocol design service

How we do it: protocol design service

E.g., a client has a formalized protocol and want help checking its correctness

How we do it: protocol design service

E.g., a client has a formalized protocol and want help checking its correctness

We deliver a correctness artifact produced via different methods that provide different levels of trust

How we do it: protocol design service

E.g., a client has a formalized protocol and want help checking its correctness

We deliver a correctness artifact produced via different methods that provide different levels of trust

Simulations using executable specs, model-checking or pencil-and-paper mathematical analysis

How we do it: #3 securing chains

How we do it: #3 securing chains

Proof-of-Stake validation and IBC relaying on major networks

How we do it: #3 securing chains

Proof-of-Stake validation and IBC relaying on major networks

This means that we participate in consensus on major blockchains and relay packets between them for chain interoperability

Our approach

Our approach

We adopt the “model first” approach

Our approach

We adopt the “model first” approach

Where formal models are first-class artifacts in the software development process

Our approach

We adopt the “model first” approach

Where formal models are first-class artifacts in the software development process

We apply it both in the projects we steward, and in the security audits that involve protocol design and analysis

Formal modeling and protocol analysis at Informal Systems

How we've done it so far

How we’ve done it so far

PODC/DISC style

Proposer-Based Time - Part I

System Model

Time and Clocks

[PBTS-CLOCK-NEWTON.0]

There is a reference Newtonian real-time `t` (UTC).

Every correct validator `V` maintains a synchronized clock `C_V` that ensures:

[PBTS-CLOCK-PRECISION.0]

There exists a system parameter `PRECISION` such that for any two correct validators `V` and `W`, and at any real-time `t`, `|C_V(t) - C_W(t)| < PRECISION`

Message Delays

We do not want to interfere with the Tendermint timing assumptions. We will postulate a timing restriction, which, if satisfied, ensures that liveness is preserved.

In general the local clock may drift from the global time. (It may progress faster, e.g., one second of clock time might take 1.005 seconds of real-time). As a result the local clock and the global clock may be measured in different time units. Usually, the message delay is measured in global clock time units. To estimate the correct local timeout precisely, we would need to estimate the clock time duration of a message delay taking into account the clock drift. For simplicity we ignore this, and directly postulate the message delay assumption in terms of local time.

How we've done it so far

PODC/DISC style

Proposer-Based Time - Part I

System Model

Time and Clocks

[PBTS-CLOCK-NEWTON.0]

There is a reference Newtonian real-time `t` (UTC).

Every correct validator `V` maintains a synchronized clock `C_V` that ensures:

[PBTS-CLOCK-PRECISION.0]

There exists a system parameter `PRECISION` such that for any two correct validators `V` and `W`, and at any real-time `t`, `|C_V(t) - C_W(t)| < PRECISION`

Message Delays

We do not want to interfere with the Tendermint timing assumptions. We will postulate a timing restriction, which, if satisfied, ens[...] liveness is preserved.

In general the local clock may drift from the global time. (It may progress faster, e.g., one second of clock time might take 1.005 se[...] real-time). As a result the local clock and the global clock may be measured in different time units. Usually, the message delay is m[...] in global clock time units. To estimate the correct local timeout precisely, we would need to estimate the clock time duration of a m[...] delay taking into account the clock drift. For simplicity we ignore this, and directly postulate the message delay assumption in term[...] time.

TLA+

```
\* lines 36-46
\* [PBTS-ALG-NEW-PREVOTE.0]
UponProposalInPrevoteOrCommitAndPrevote(p) ==
  \E v \in ValidValues, t \in Timestamps, vr \in RoundsOrNil:
    /\ step[p] \in {"PREVOTE", "PRECOMMIT"} \* line 36
    /\ LET msg ==
         AsMsg([type |-> "PROPOSAL", src |-> Proposer[round[p]],
                round |-> round[p], proposal |-> Proposal(v, t), validRound |-> vr]) IN
       /\ <<p, msg>> \in receivedTimelyProposal \* updated line 36
       /\ LET PV == { m \in msgsPrevote[round[p]]: m.id = Id(Proposal(v, t)) } IN
         /\ Cardinality(PV) >= THRESHOLD2 \* line 36
         /\ evidence' = PV \union {msg} \union evidence
    /\  IF step[p] = "PREVOTE"
       THEN \* lines 38-41:
         /\ lockedValue' = [lockedValue EXCEPT ![p] = v]
         /\ lockedRound' = [lockedRound EXCEPT ![p] = round[p]]
         /\ BroadcastPrecommit(p, round[p], Id(Proposal(v, t)))
         /\ step' = [step EXCEPT ![p] = "PRECOMMIT"]
       ELSE
         UNCHANGED <<lockedValue, lockedRound, msgsPrecommit, step>>
      \* lines 42-43
    /\ validValue' = [validValue EXCEPT ![p] = v]
    /\ validRound' = [validRound EXCEPT ![p] = round[p]]
    /\ UNCHANGED <<round, decision, msgsPropose, msgsPrevote,
                   localClock, realTime, receivedTimelyProposal, inspectedProposal,
                   beginConsensus, endConsensus, lastBeginConsensus, proposalTime, proposalReceivedTime>>
    /\ action' = "UponProposalInPrevoteOrCommitAndPrevote"
```

Why it isn't working

Why it isn't working

Only experts, with very a specific background can do it

Why it isn't working

Only experts, with very a specific background can do it

We want engineers and auditors to formalize and analyze their own protocols

Quint: more than a modern specification language

Quint

Quint

An executable specification language design for usability

Quint

An executable specification language design for usability

Combines the robust theoretical basis of TLA - it is in a way a new skin for TLA+

Quint

An executable specification language design for usability

Combines the robust theoretical basis of TLA - it is in a way a new skin for TLA+

With state-of-the-art static analysis and development tooling

The team

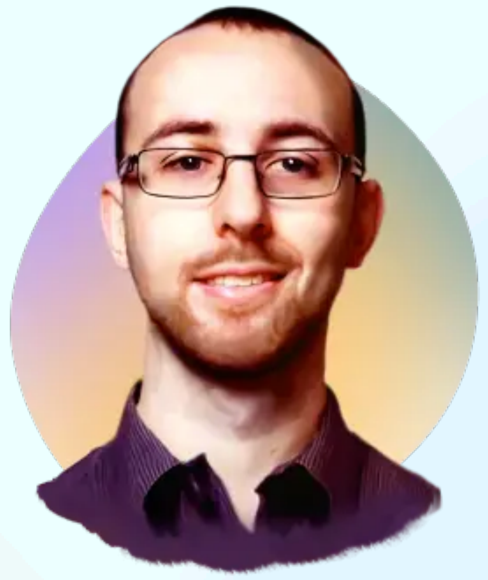
Igor Konnov

Thomas Pani

Gabriela Moreira

Shon Feder

Jure Kukovec



The Quint language

Design principles

Design principles

Least surprise: copy syntax from mainstream languages

Design principles

Least surprise: copy syntax from mainstream languages

Easy to read: keeps the set of ASCII control characters to minimum, eliminates ambiguity, types

Design principles

Least surprise: copy syntax from mainstream languages

Easy to read: keeps the set of ASCII control characters to minimum, eliminates ambiguity, types

Easy to write and parse: a small set of syntactic rules (250 LOC)

Design principles

Least surprise: copy syntax from mainstream languages

Easy to read: keeps the set of ASCII control characters to minimum, eliminates ambiguity, types

Easy to write and parse: a small set of syntactic rules (250 LOC)

(Mostly) **compatible with TLA+**

Design principles

Least surprise: copy syntax from mainstream languages

Easy to read: keeps the set of ASCII control characters to minimum, eliminates ambiguity, types

Easy to write and parse: a small set of syntactic rules (250 LOC)

(Mostly) **compatible with TLA+**

Command line-first: IDEs change, CLI tools stay

Cheatsheet

Comments

// one line

/* multiple
lines */

Basic types

bool – Booleans

int – signed big integers

str – string literals

type *Name* = *otherType*
type alias, starts with upper-case

Literals

false true

123 123_000 0x12abcd

"Quint": str, a string

Int: Set[int] – all integers

Nat: Set[int] – all non-negative integers

Bool = Set(false, true)

Records

{ *name*: *str*, *age*: *int* }

Sets - core data structure!

Set[*T*] – type: set with elements of type T

Set(1, 2, 3) – new set, contains its arguments

1.to(4) – new set: Set(1, 2, 3, 4)

1.in(*S*) – true, if the argument is in *S*

S.contains(1) – the same

S.subseteq(*T*) – true, if all elements of *S* are in *T*

S.union(*T*) – new set: elements in *S* or in *T*

S.intersect(*T*) – new set: elements both in *S* and in *T*

S.exclude(*T*) – new set: elements in *S* but not in *T*

S.map(*x* => 2 * *x*) – new set: elements of S are transformed by expression

S.filter(*x* => *x* > *0*) – new set: leaves the elements of S that satisfy condition

S.exists(*x* => *x* > 10) – true, if some element of *S* satisfies condition

S.forall(*x* => *x* <= 10) – true, if all elements of *S*

Maps - key/value bindings

a -> *b* – type: binds keys of type *a* to values of type *b*

Map(1 -> 2, 3 -> 6) – binds keys 1, 3 to values 2, 6

S.mapBy(*x* => 2 * *x*) – binds keys in *S* to expressions

M.keys() – the set of keys

M.get(*key*) – get the value bound to *key*

M.set(*k*, *v*) – copy of *M*: but binds *k* to *v*, if *k* has a value

M.put(*key*, *v*) – copy of *M*: but (re-)binds *k* to *v*

M.setBy(*k*, *(old* => *old* + *1))* as *M*.set(*k*, *v*) but *v* is computed via anonymous operator with *old* == *M*.get(*k*)

S.setOfMaps(*T*) – new set: contains **all maps** that bind elements of *S* to elements of *T*

Set((1, 2), (3, 6)).setToMap() new map: bind the first elements of tuples to the second elements

Tuples

(*str*, *int*, *bool*)
tuple type

Lists - use Set, if you can

List[*T*] – type: list with elements of type T

[1, 2, 3] – new list, contains its arguments in order

List(1, 2, 3) – the same

range(*start*, *end*) – new list [*start*, *start* + 1, …, *end* – 1]

length(L) – the number of elements in the list *L*

L[*i*] – ith element, if 0 <= *i* < length(*L*)

L.concat(*K*) – new list: start with elements of *L*, continue with elements of *K*

L.append(*x*) – new list: just L.concat([x])

L.replaceAt(*i*, *x*) – *L*'s copy but the ith element is set to *x*

L.slice(*s*, *e*) – new list: [*L*[*s*], …, *L*[*e* – 1]]

L.select(*x* > *5*) – new list: leaves the elements of *L* that satisfy condition

L.foldl(*i*, (*s*, *x*) => *x* + *s*) go over elements of *L in order*, apply expression, continue with

Highlights

Highlights

- Layered language: it introduces "modes", which are similar in spirit to TLA+ levels, but more refined

Highlights

- Layered language: it introduces "modes", which are similar in spirit to TLA+ levels, but more refined

- Types are built-in

Highlights

- Layered language: it introduces "modes", which are similar in spirit to TLA+ levels, but more refined
- Types are built-in
- Folds instead of recursive operators

Highlights

- Layered language: it introduces "modes", which are similar in spirit to TLA+ levels, but more refined
- Types are built-in
- Folds instead of recursive operators
- Isolates non-determinism

Layered language

Pure and stateful definitions

Layered language

Pure and stateful definitions

```
pure val MAX_UINT = 2^256 - 1

pure def sumOverBalances(balances) = {
   balances.keys().fold(0,
     (sum, a) => sum + balances.get(a))
}
var state: Erc20State
val totalSupplyInv = isTotalSupplyCorrect(state)
```

Layered language

Pure and stateful definitions

```
pure val MAX_UINT = 2^256 - 1

pure def sumOverBalances(balances) = {
    balances.keys().fold(0,
       (sum, a) => sum + balances.get(a))
}
var state: Erc20State
val totalSupplyInv = isTotalSupplyCorrect(state)
```

pure = stateless definitions

Layered language

Pure and stateful definitions

```
pure val MAX_UINT = 2^256 - 1

pure def sumOverBalances(balances) = {
    balances.keys().fold(0,
       (sum, a) => sum + balances.get(a))
}
var state: Erc20State
val totalSupplyInv = isTotalSupplyCorrect(state)
```

pure = stateless definitions

var = state variable

Layered language

Pure and stateful definitions

```
pure val MAX_UINT = 2^256 - 1

pure def sumOverBalances(balances) = {
    balances.keys().fold(0,
       (sum, a) => sum + balances.get(a))
}
var state: Erc20State
val totalSupplyInv = isTotalSupplyCorrect(state)
```

pure = stateless definitions

var = state variable

val = stateful definition

Layered language

Actions

Layered language

Actions

```
action submit(tx: Transaction): bool = all {
      mempool' = mempool.union(Set(tx)),
      erc20State' = erc20State,
      lastTx' = tx,
  }
```

Layered language

Actions

```
action submit(tx: Transaction): bool = all {
      mempool' = mempool.union(Set(tx)),
      erc20State' = erc20State,
      lastTx' = tx,
  }
```

Used to make state transitions

Layered language

Actions

```
action submit(tx: Transaction): bool = all {
      mempool' = mempool.union(Set(tx)),
      erc20State' = erc20State,
      lastTx' = tx,
  }
```

Used to make state transitions

Layered language

Runs

Layered language

Runs

```
run transferFromWhileApproveInFlightTest = {
  all {
    erc20State' = newErc20("alice", 91),
    mempool' = Set(), lastTx' = NoneTx,
  } // alice sets a high approval for bob
  .then(submit(ApproveTx("alice", "bob", 92)))
  // bob immediately initiates his transaction
  then(submit(TransferFromTx("bob", "alice", "eve", 54)))
  // alice changes her mind and lowers her approval to bob
 ...
}
```

Layered language

Runs

```
run transferFromWhileApproveInFlightTest = {
  all {
    erc20State' = newErc20("alice", 91),
    mempool' = Set(), lastTx' = NoneTx,
  } // alice sets a high approval for bob
  .then(submit(ApproveTx("alice", "bob", 92)))
  // bob immediately initiates his transaction
  then(submit(TransferFromTx("bob", "alice", "eve", 54)))
  // alice changes her mind and lowers her approval to bob
 ...
}
```

A run represents a finite execution

Layered language

Runs

```
run transferFromWhileApproveInFlightTest = {
  all {
    erc20State' = newErc20("alice", 91),
    mempool' = Set(), lastTx' = NoneTx,
  } // alice sets a high approval for bob
  .then(submit(ApproveTx("alice", "bob", 92)))
  // bob immediately initiates his transaction
  then(submit(TransferFromTx("bob", "alice", "eve", 54)))
  // alice changes her mind and lowers her approval to bob
 ...
}
```

A run represents a finite execution

Sequence of actions

Layered language

Runs

```
run transferFromWhileApproveInFlightTest = {
  all {
    erc20State' = newErc20("alice", 91),
    mempool' = Set(), lastTx' = NoneTx,
  } // alice sets a high approval for bob
  .then(submit(ApproveTx("alice", "bob", 92)))
  // bob immediately initiates his transaction
  then(submit(TransferFromTx("bob", "alice", "eve", 54)))
  // alice changes her mind and lowers her approval to bob
 ...
}
```

A run represents a finite execution

Sequence of actions

Think about tests: unit tests and property-based

Layered language

Temporal

Layered language

Temporal

```
temporal noOverheat =
  always(temperature <= 100)

temporal eventuallyOff =
  eventually(not(heatingOn))
```

Layered language

Temporal

```
temporal noOverheat =
  always(temperature <= 100)

temporal eventuallyOff =
  eventually(not(heatingOn))
```

Temporal properties

Layered language

Temporal

```
temporal noOverheat =
  always(temperature <= 100)

temporal eventuallyOff =
  eventually(not(heatingOn))
```

Temporal properties

Describe infinite executions

Types are built-in

Types are built-in

```
// type aliases
type Address = str
type Uint = int
// variables must have a type annotation
var mempool: Set[Transaction]
// operators may have a type annotation
pure def isUint(i: int): bool =
  (0 <= i and i <= MAX_UINT)
```

Types are built-in

```
// type aliases
type Address = str
type Uint = int
// variables must have a type annotation
var mempool: Set[Transaction]
// operators may have a type annotation
pure def isUint(i: int): bool =
  (0 <= i and i <= MAX_UINT)
```

```
// a record type
type Erc20State = {
  // a map of addresses to amounts
  balanceOf: Address -> Uint,
  // the sum of all balances
  totalSupply: Uint,
  // a map of pairs to amounts
  allowance: (Address, Address) -> Uint,
  // the address of the contract creator
  owner: Address,
}
```

Folds instead of recursive operators

Folds instead of recursive operators

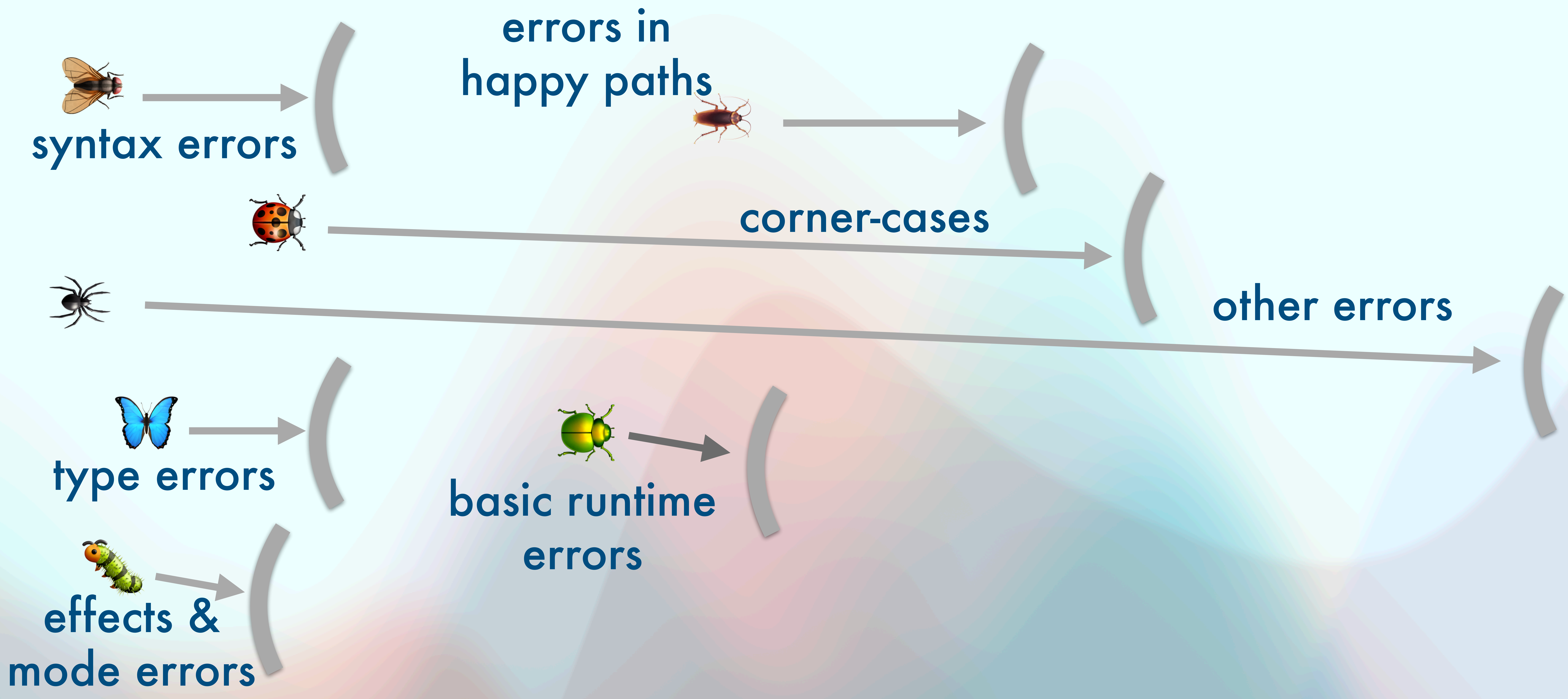
```
Iteration over sets
//iterates in some order
//always terminates
//size(..) iterations
pure def sumOverBalances(balances) = {
   balances.keys().
fold(0, (sum, a) => sum + balances.get(a))
```

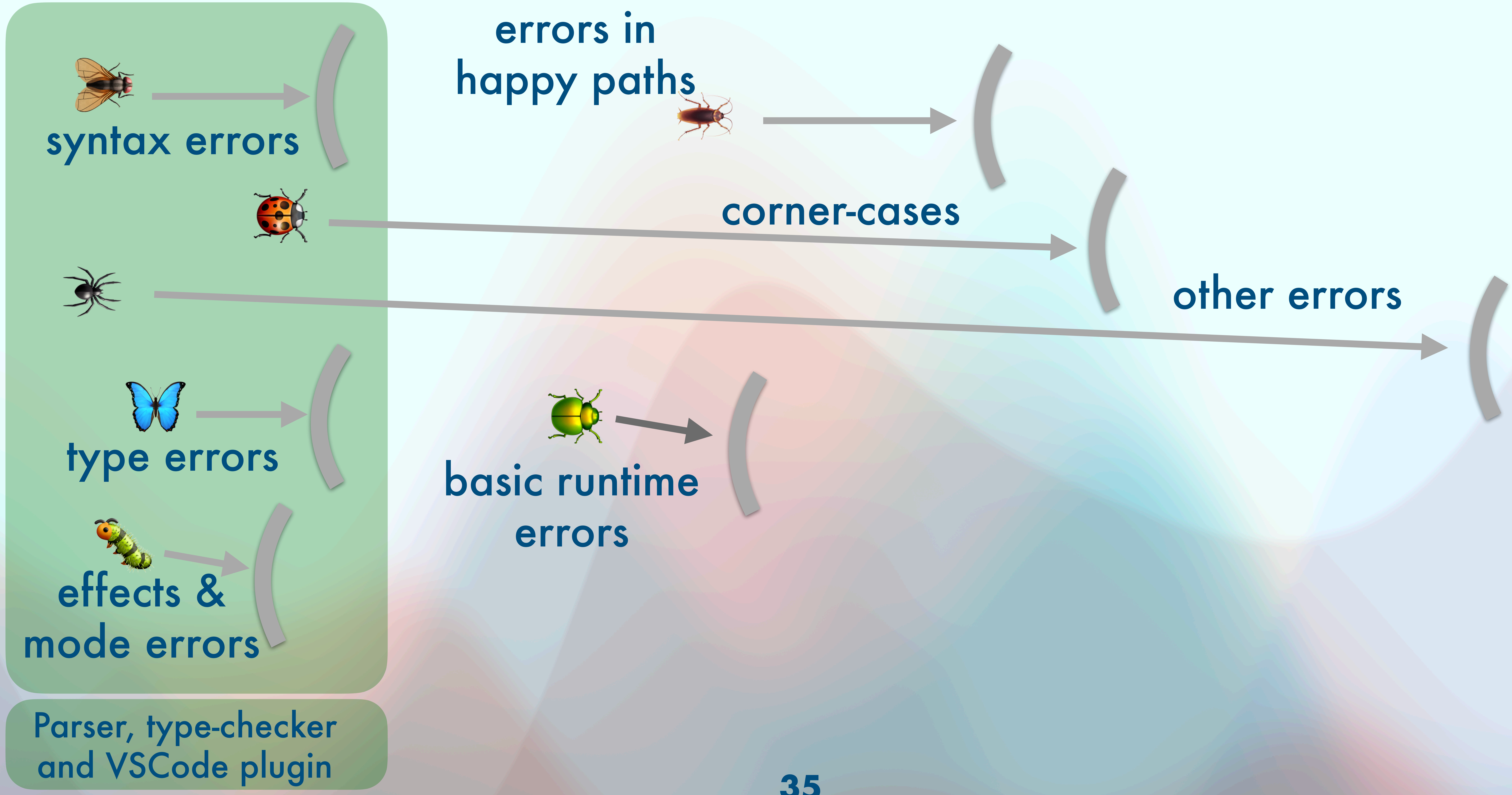
Folds instead of recursive operators

```
Iteration over sets
//iterates in some order
//always terminates
//size(..) iterations
pure def sumOverBalances(balances) = {
    balances.keys().
fold(0, (sum, a) => sum + balances.get(a))
```

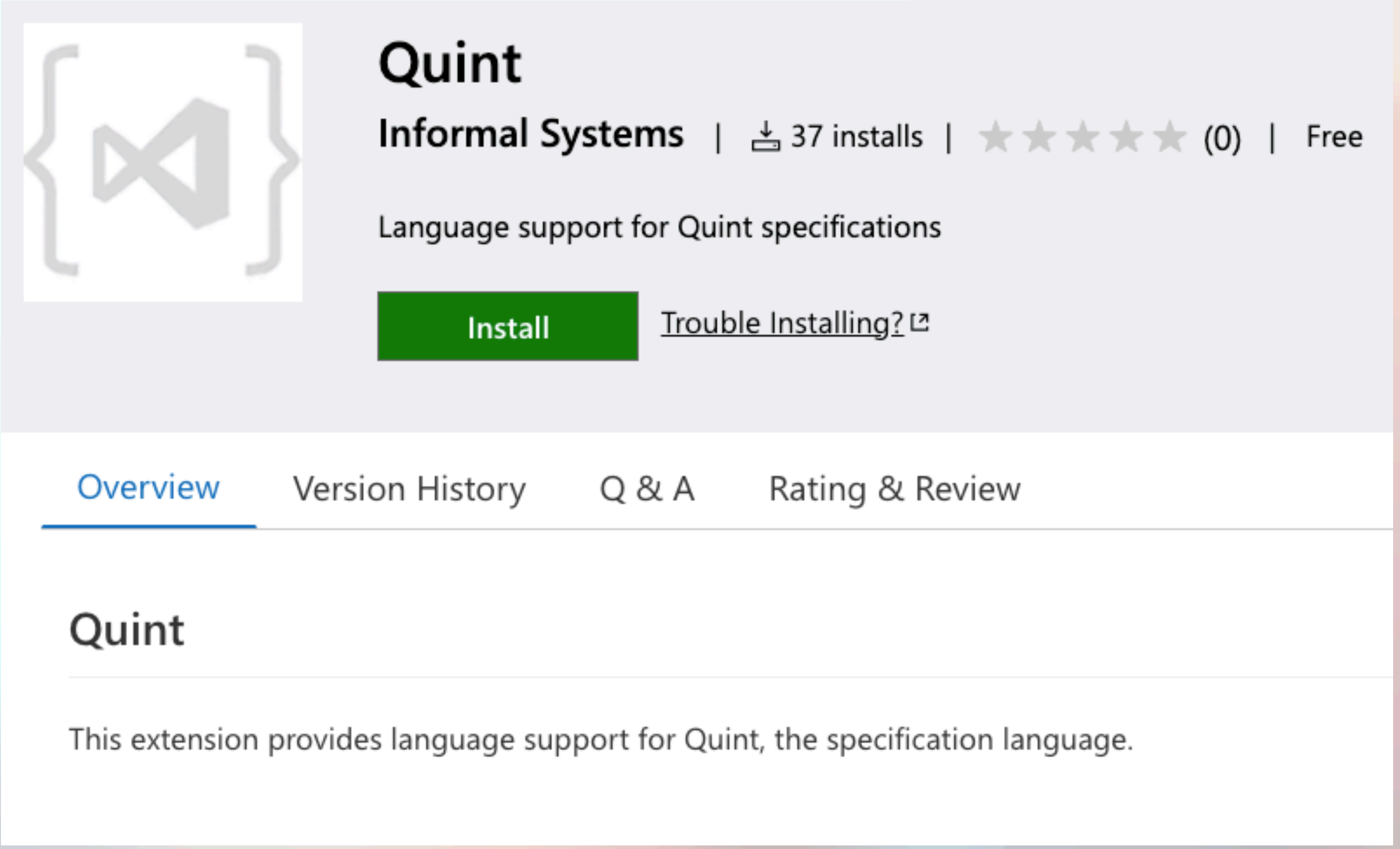
```
Iteration over lists
//always terminates
//len(..) iterations
pure def simpleHash(word) =
  word.foldl(0, (i, j) => i + j) % BASE
```

Quint tools

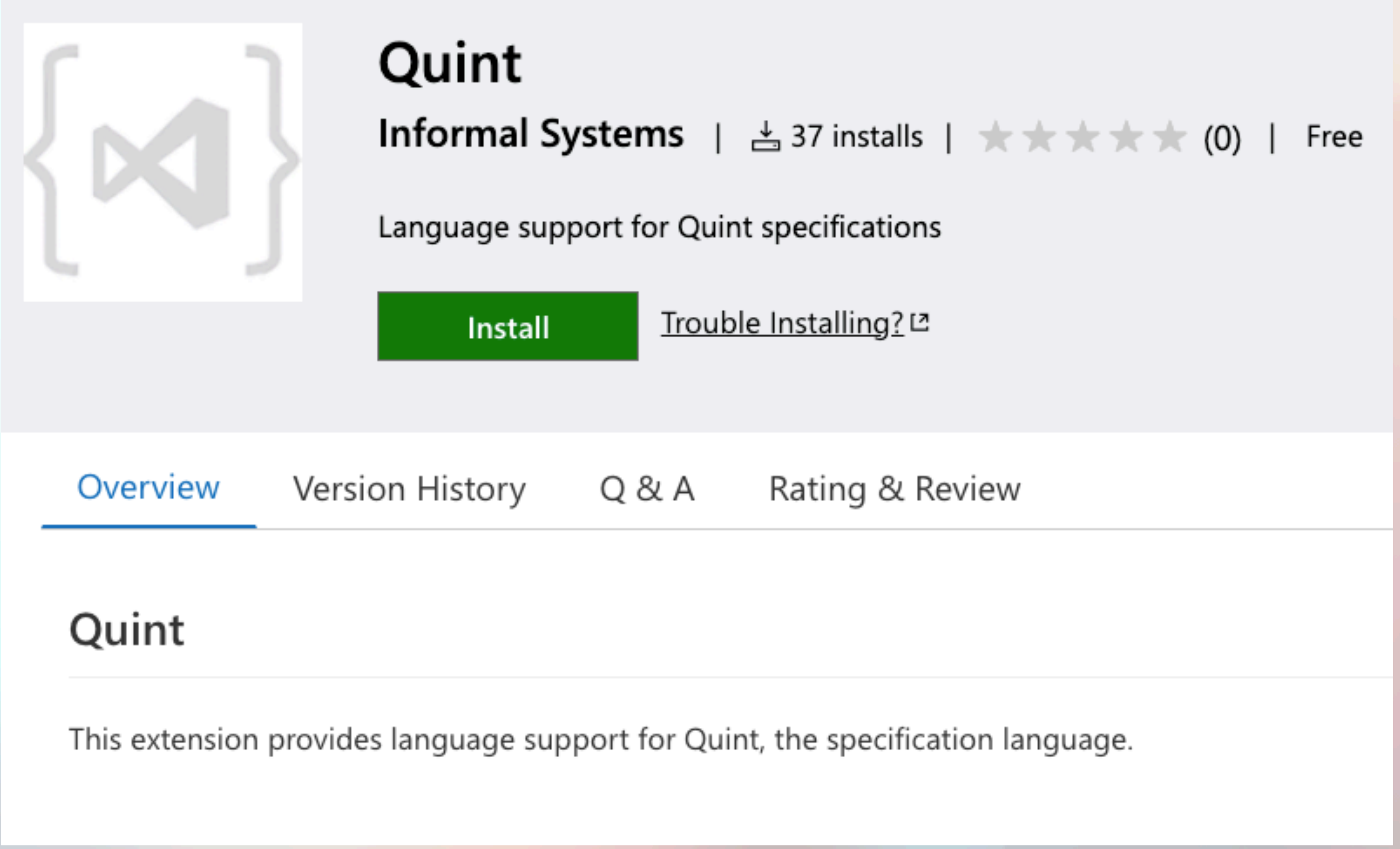

errors in
happy paths
syntax errors
corner-cases
other errors
type errors
basic runtime
errors
effects &
mode errors


errors in
happy paths
syntax errors
corner-cases
other errors
type errors
basic runtime
errors
effects &
mode errors
Parser, type-checker
and VSCode plugin

Parser and VSCode plugin

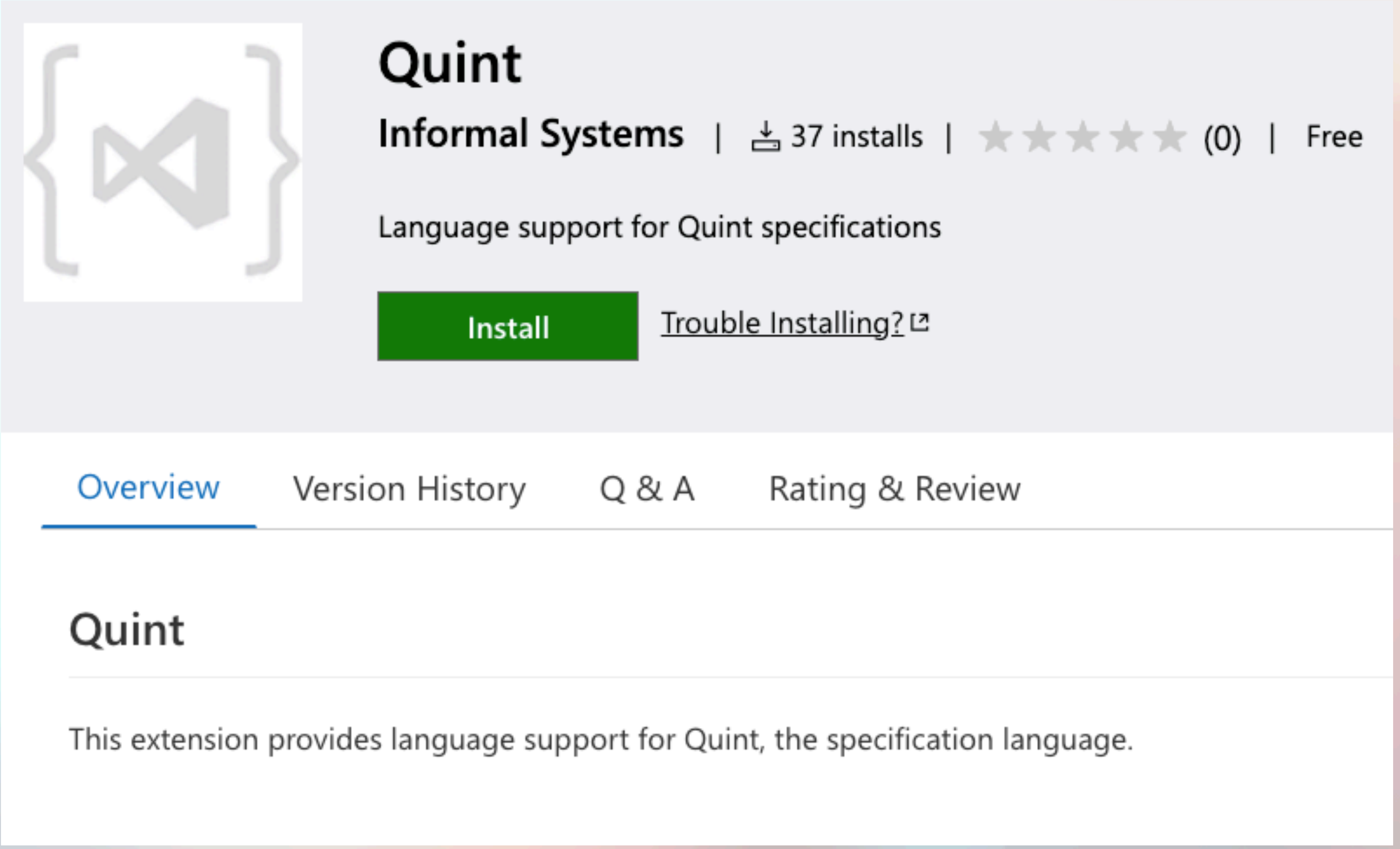


Parser and VSCode plugin



Instant feedback on

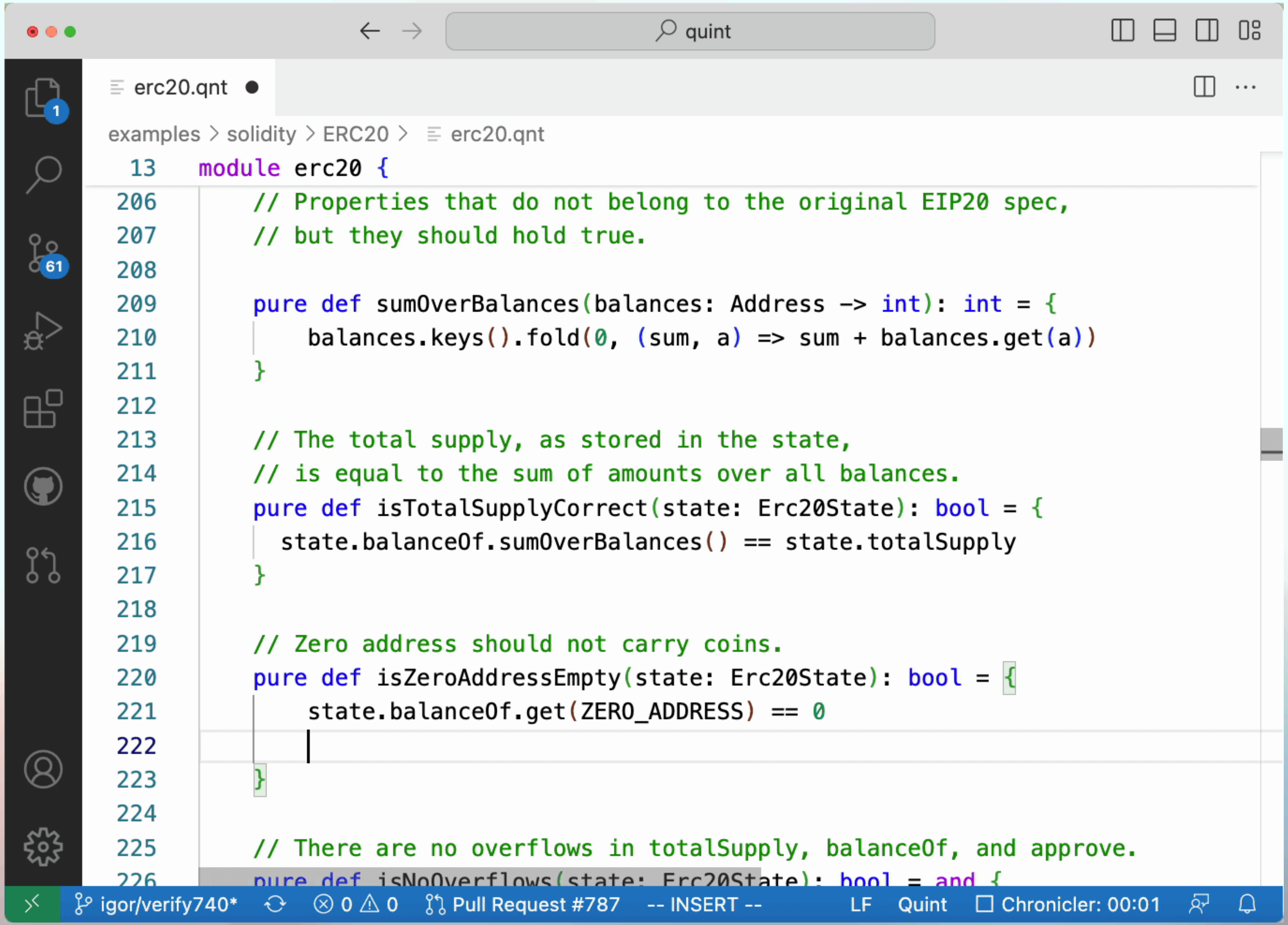
Parser and VSCode plugin



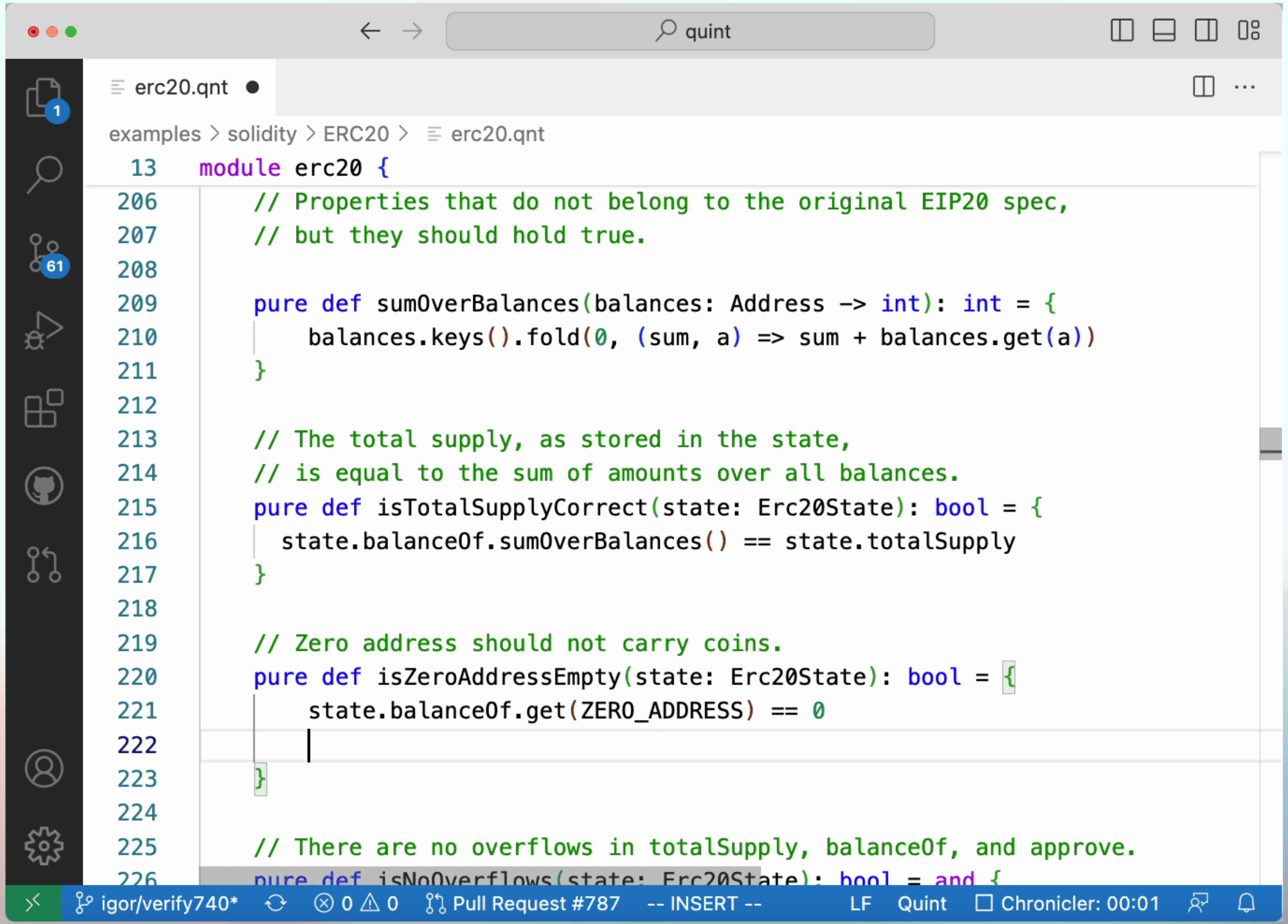
Instant feedback on

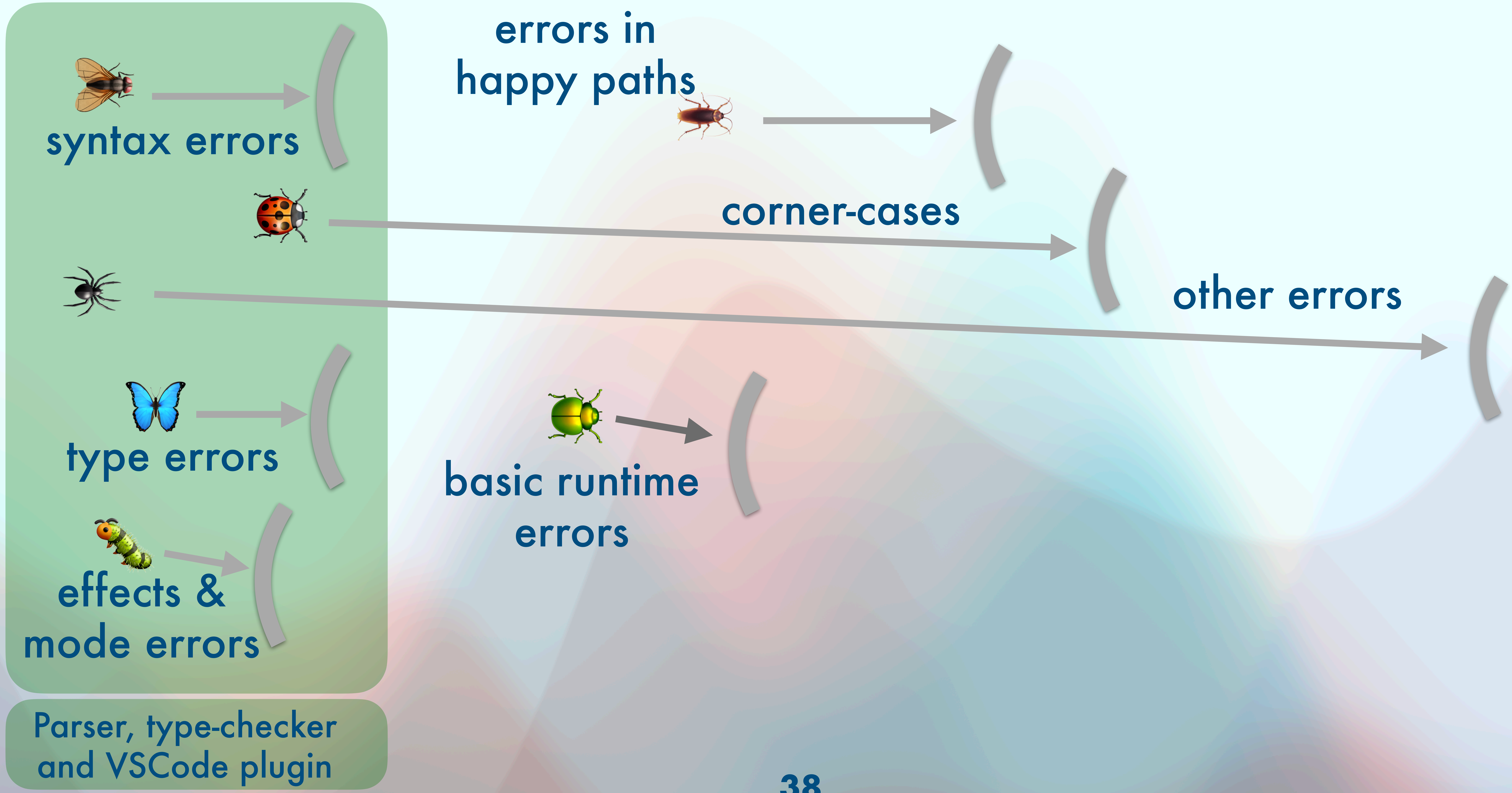
syntax, types, mode and effects errors

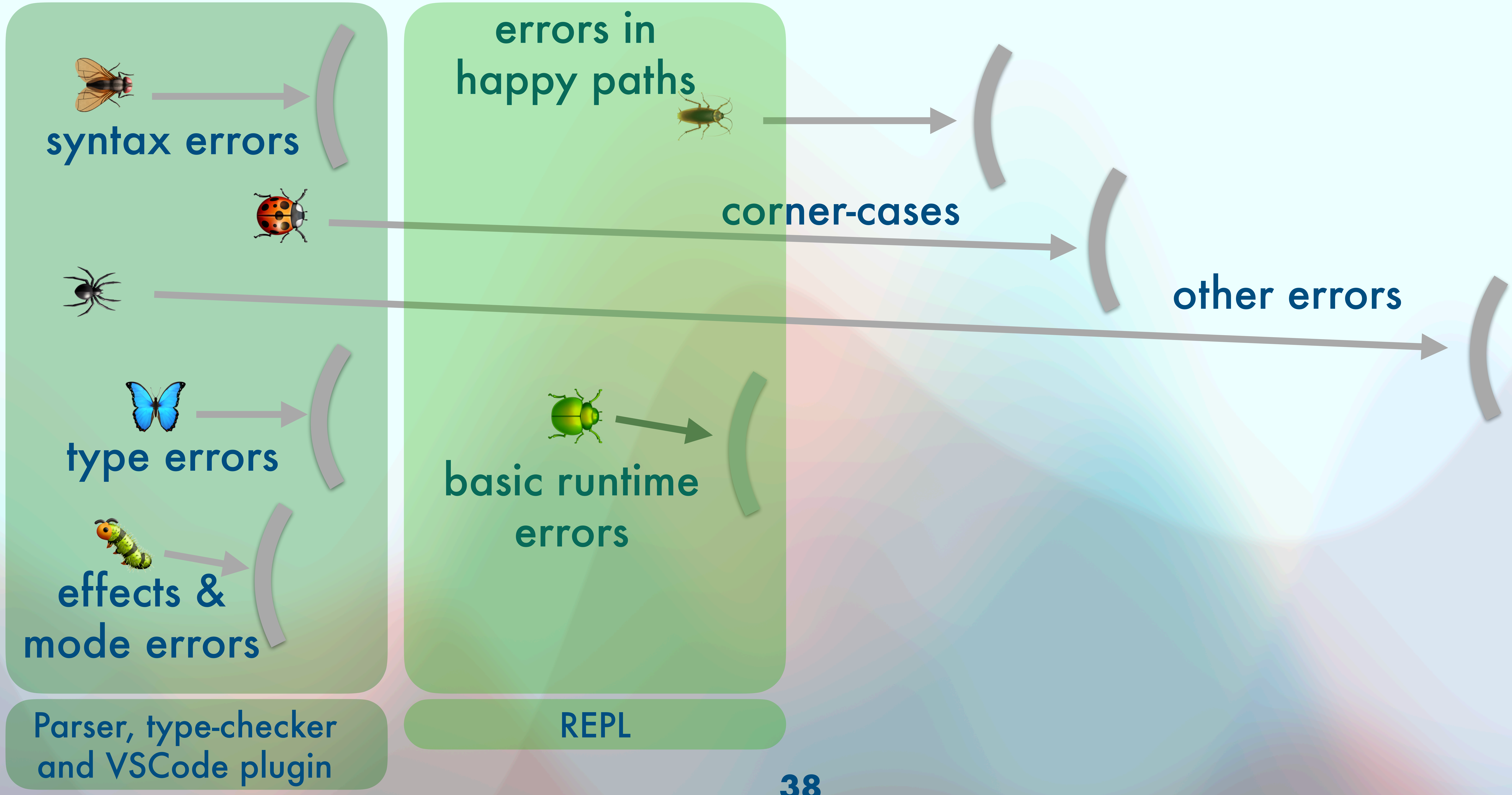
Type checker 🦋



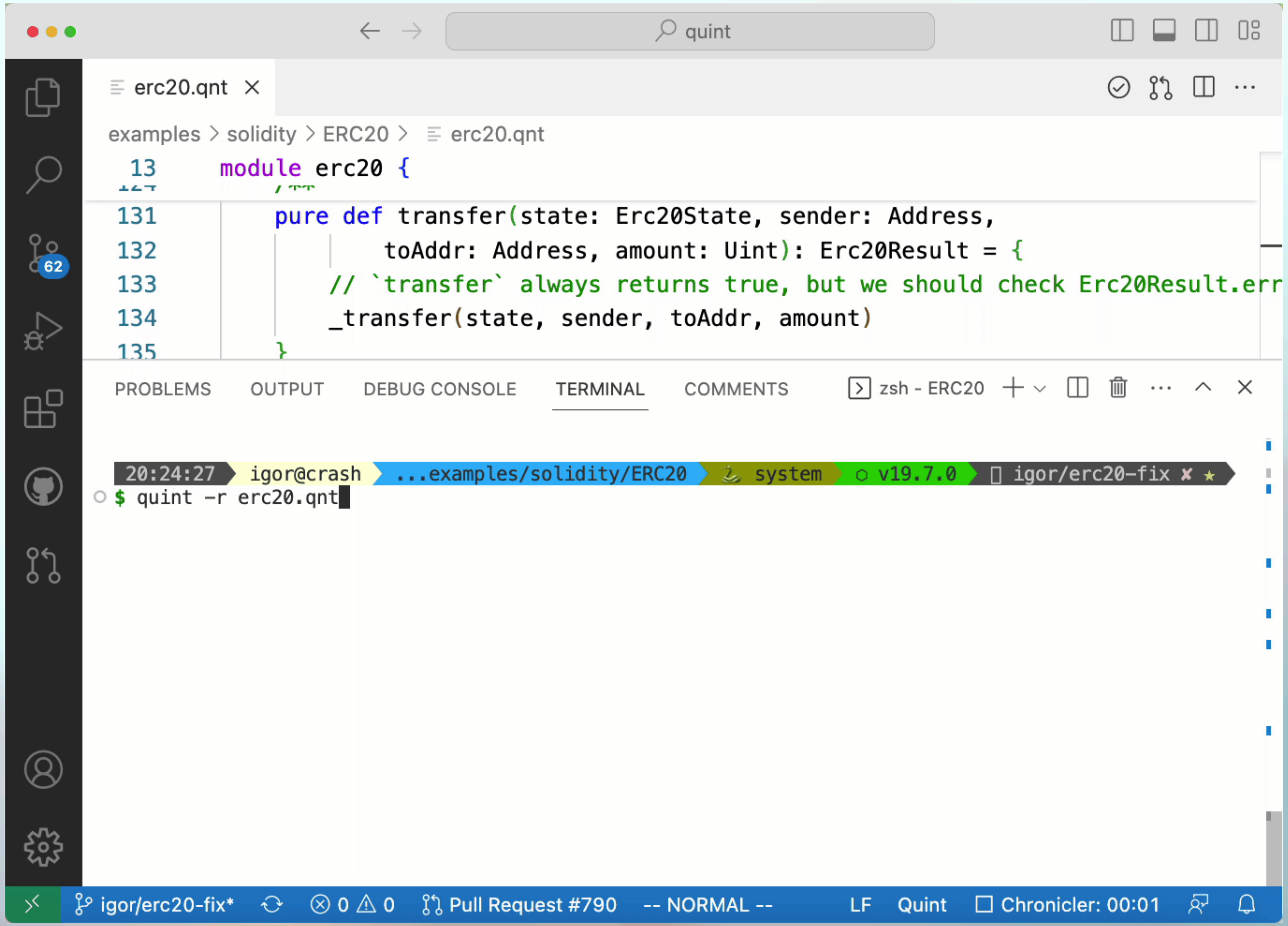
Type checker 🦋



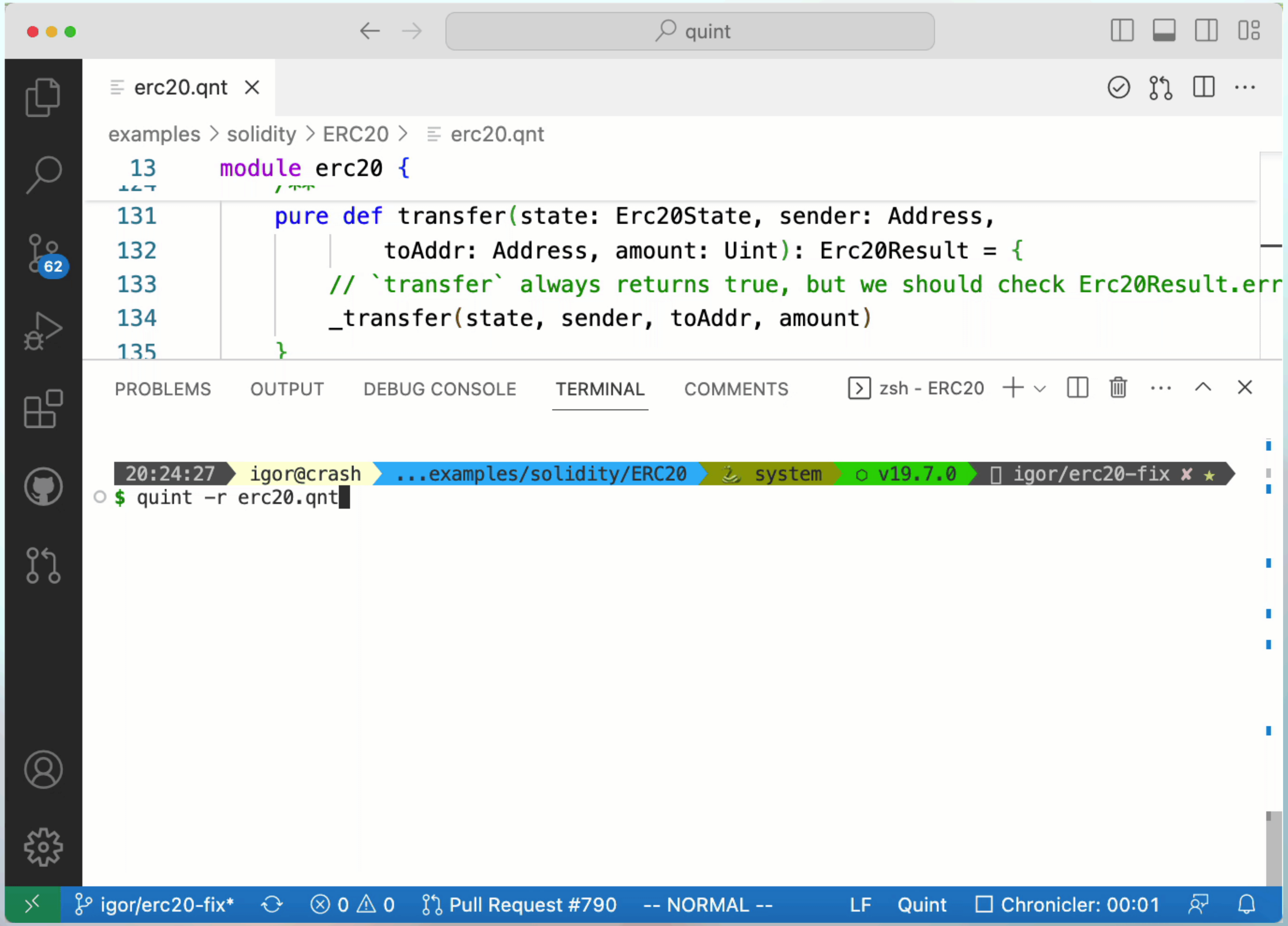

syntax errors
errors in
happy paths
corner-cases
other errors
type errors
basic runtime
errors
effects &
mode errors
Parser, type-checker
and VSCode plugin


syntax errors
type errors
effects &
mode errors
errors in
happy paths
basic runtime
errors
corner-cases
other errors
Parser, type-checker
and VSCode plugin
REPL

REPL: read-eval-print loop

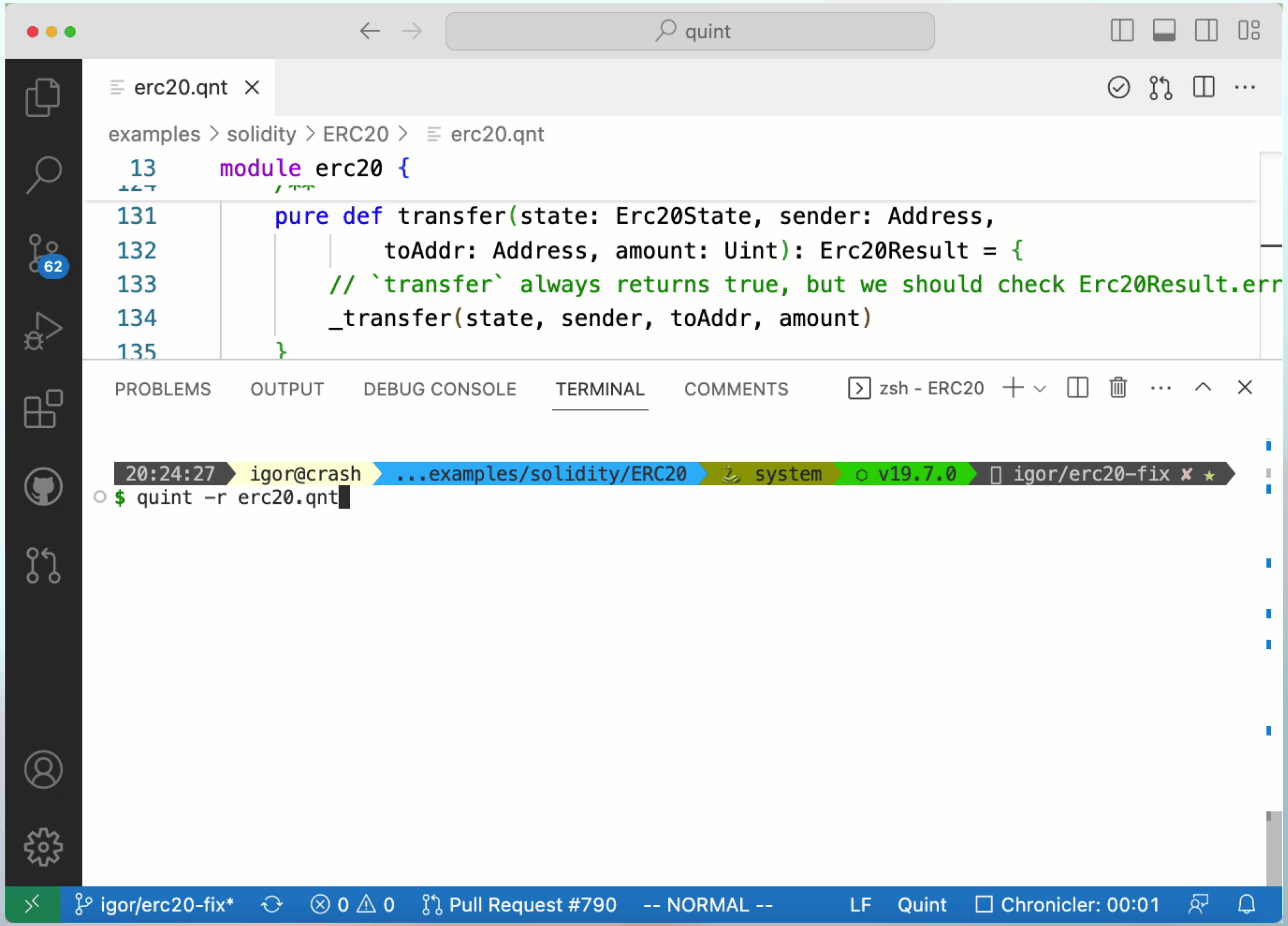


REPL: read-eval-print loop



Interactive learning

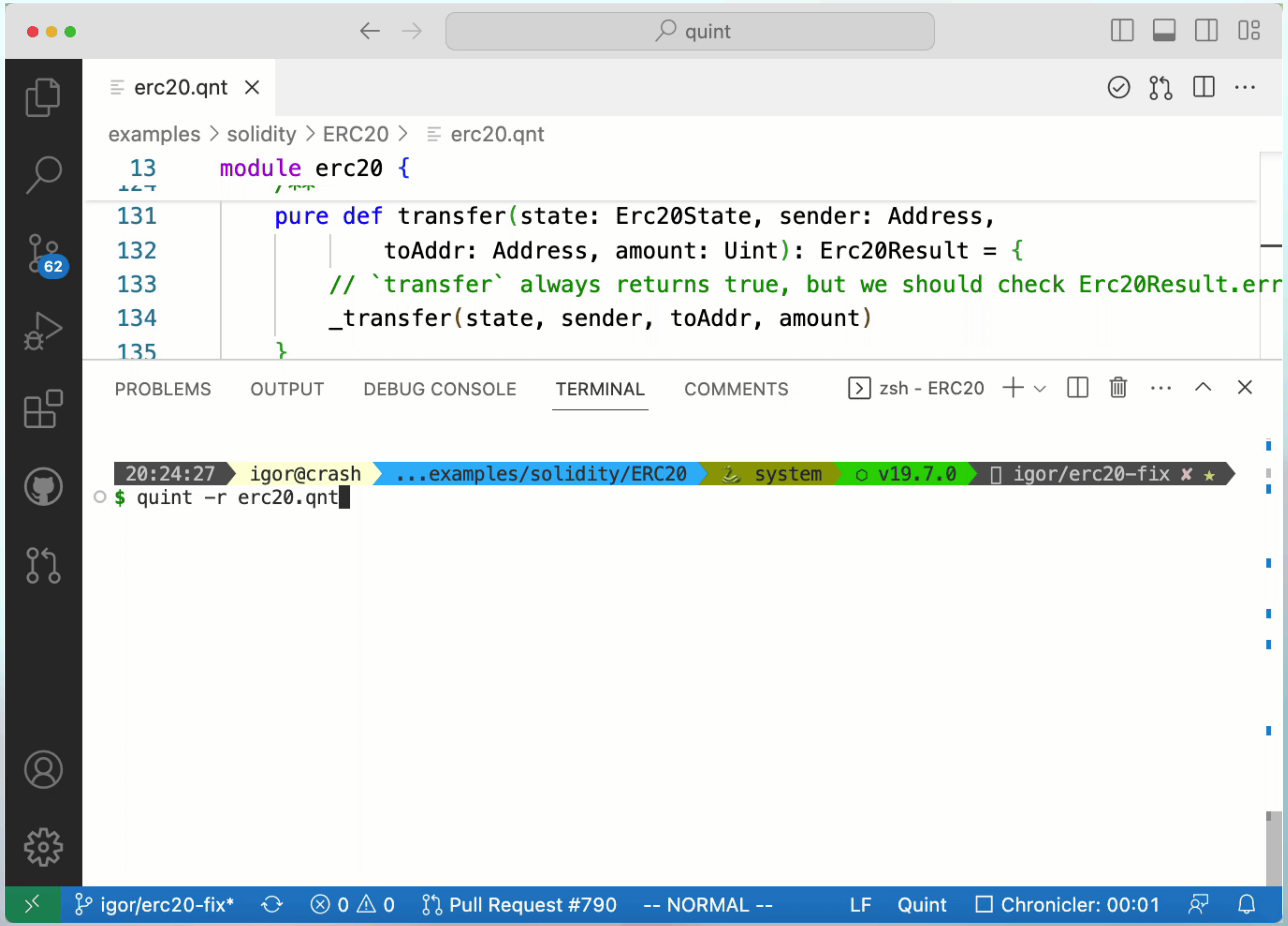
REPL: read-eval-print loop



Interactive learning

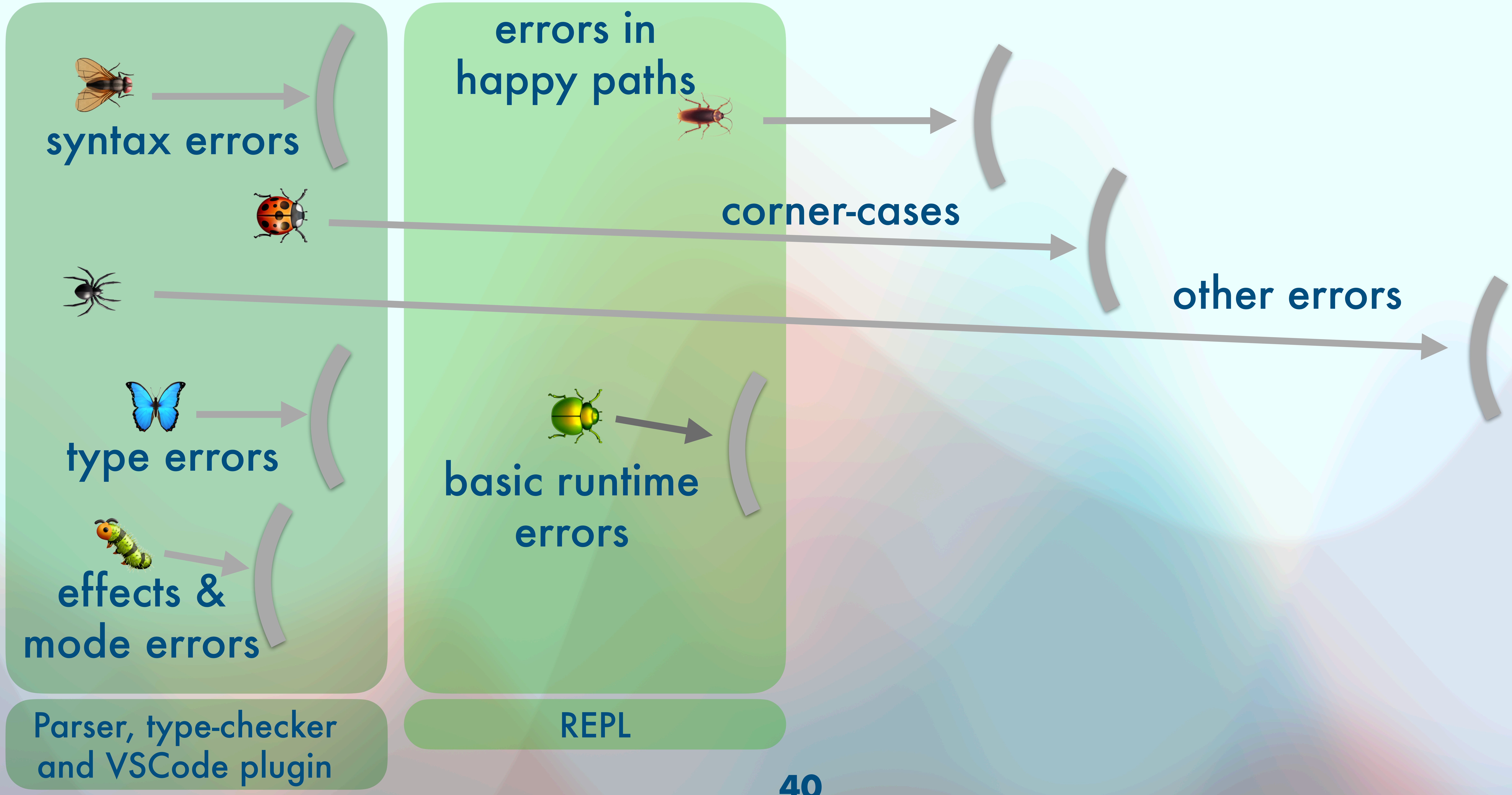
Step-by-step debugging

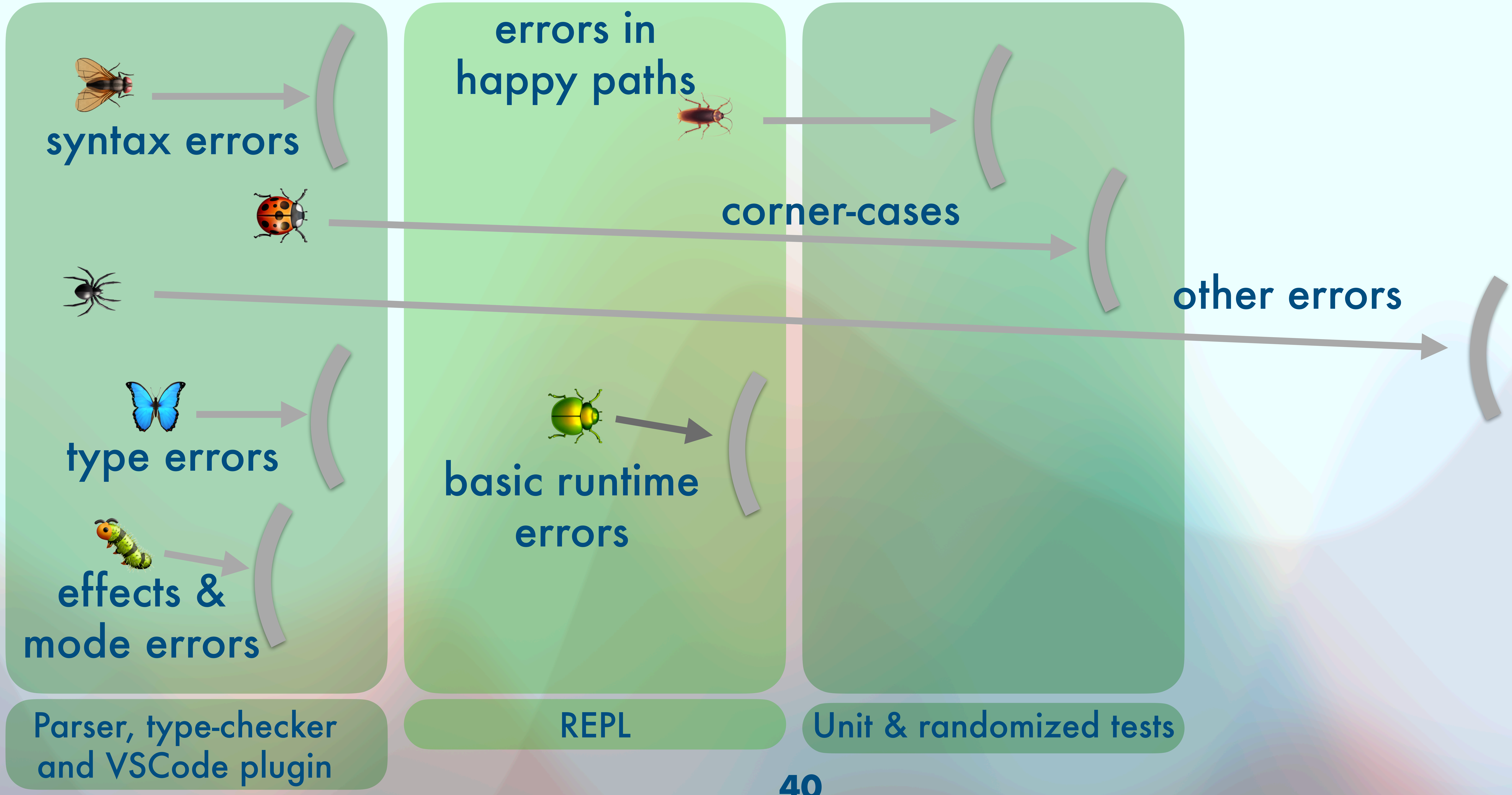
REPL: read-eval-print loop



Interactive learning

Step-by-step debugging


errors in happy paths
syntax errors
corner-cases
other errors
type errors
basic runtime errors
effects & mode errors
Parser, type-checker and VSCode plugin
REPL


errors in happy paths
syntax errors
corner-cases
other errors
type errors
basic runtime errors
effects & mode errors
Parser, type-checker and VSCode plugin
REPL
Unit & randomized tests

Random simulator

Random simulator

We can use the **run** command to execute a Quint specification via random simulation similar to stateful property-based testing

Random simulator

We can use the **run** command to execute a Quint specification via random simulation similar to stateful property-based testing

In other words, check invariants in **—max-samples** random executions up to **—max-steps** each

Random simulator

We can use the **run** command to execute a Quint specification via random simulation similar to stateful property-based testing

In other words, check invariants in **—max-samples** random executions up to **—max-steps** each

```
quint run —invariant=myInvariant —verbosity=3 myspec.qnt
```

Random simulator

Random simulator

Two special actions:

Random simulator

Two special actions:

- **init**: modifies all state variables, reads none

Random simulator

Two special actions:

- **init**: modifies all state variables, reads none
- **step**: modifies all state variables, may read some

Random simulator

Two special actions:

- **init**: modifies all state variables, reads none
- **step**: modifies all state variables, may read some

Ways of inserting non-determinism

Random simulator

Two special actions:

- **init**: modifies all state variables, reads none
- **step**: modifies all state variables, may read some

Ways of inserting non-determinism

- oneOf(S) randomly selects a set element

Random simulator

Two special actions:

- **init**: modifies all state variables, reads none
- **step**: modifies all state variables, may read some

Ways of inserting non-determinism

- oneOf(S) randomly selects a set element
- any { A1, ..., An } randomly selects an action

Random simulator

Two special actions:

- **init**: modifies all state variables, reads none
- **step**: modifies all state variables, may read some

Ways of inserting non-determinism

- oneOf(S) randomly selects a set element
- any { A1, ..., An } randomly selects an action

```
action step =
 any {
  nondet sender = oneOf(ADDR)
  nondet amount = oneOf(AMOUNTS)
  nondet toAddr = oneOf(ADDR)
  any {
   DepositTx(sender, amount),
   TransferTx(sender, toAddr, amount),
   ...
  }
 }
```

Random simulator

Random simulator

The simulator tries to find the shortest trace that violates the invariant

Random simulator

The simulator tries to find the shortest trace that violates the invariant

If it finds one, it outputs the trace

Random simulator

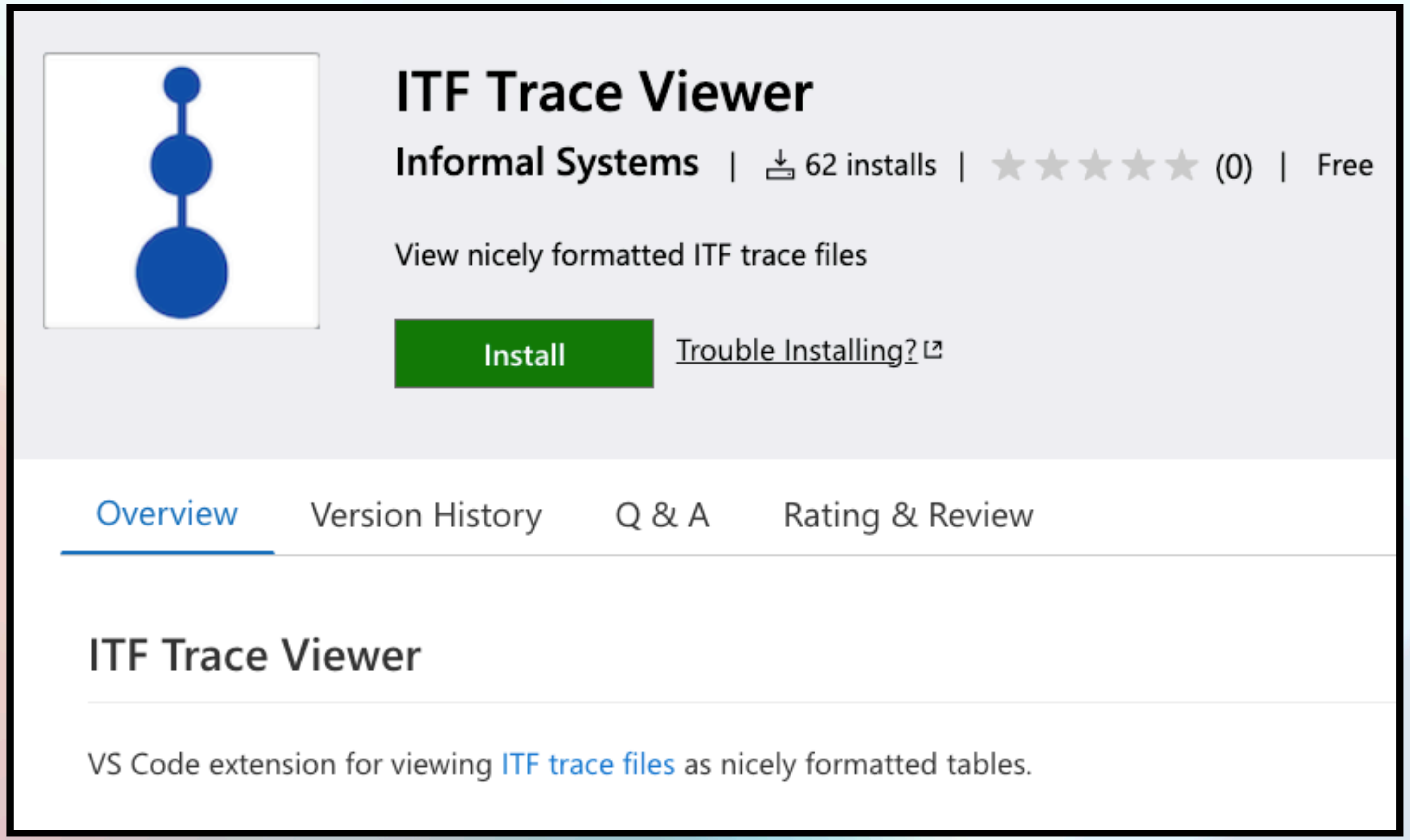
The simulator tries to find the shortest trace that violates the invariant

If it finds one, it outputs the trace

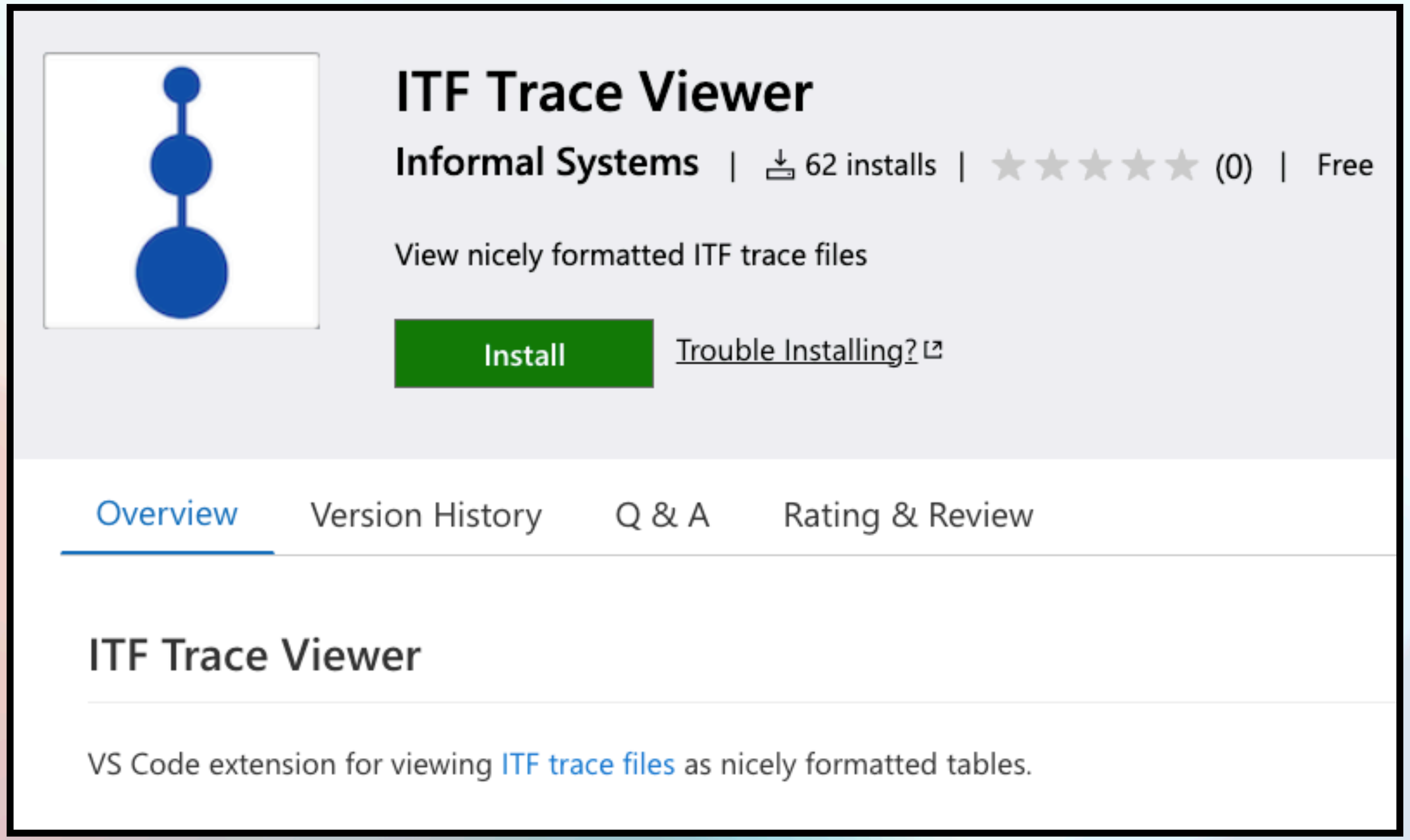
If it does not find a violating trace, it outputs the longest sample trace that the simulator has found during the execution

Trace viewer

Trace viewer

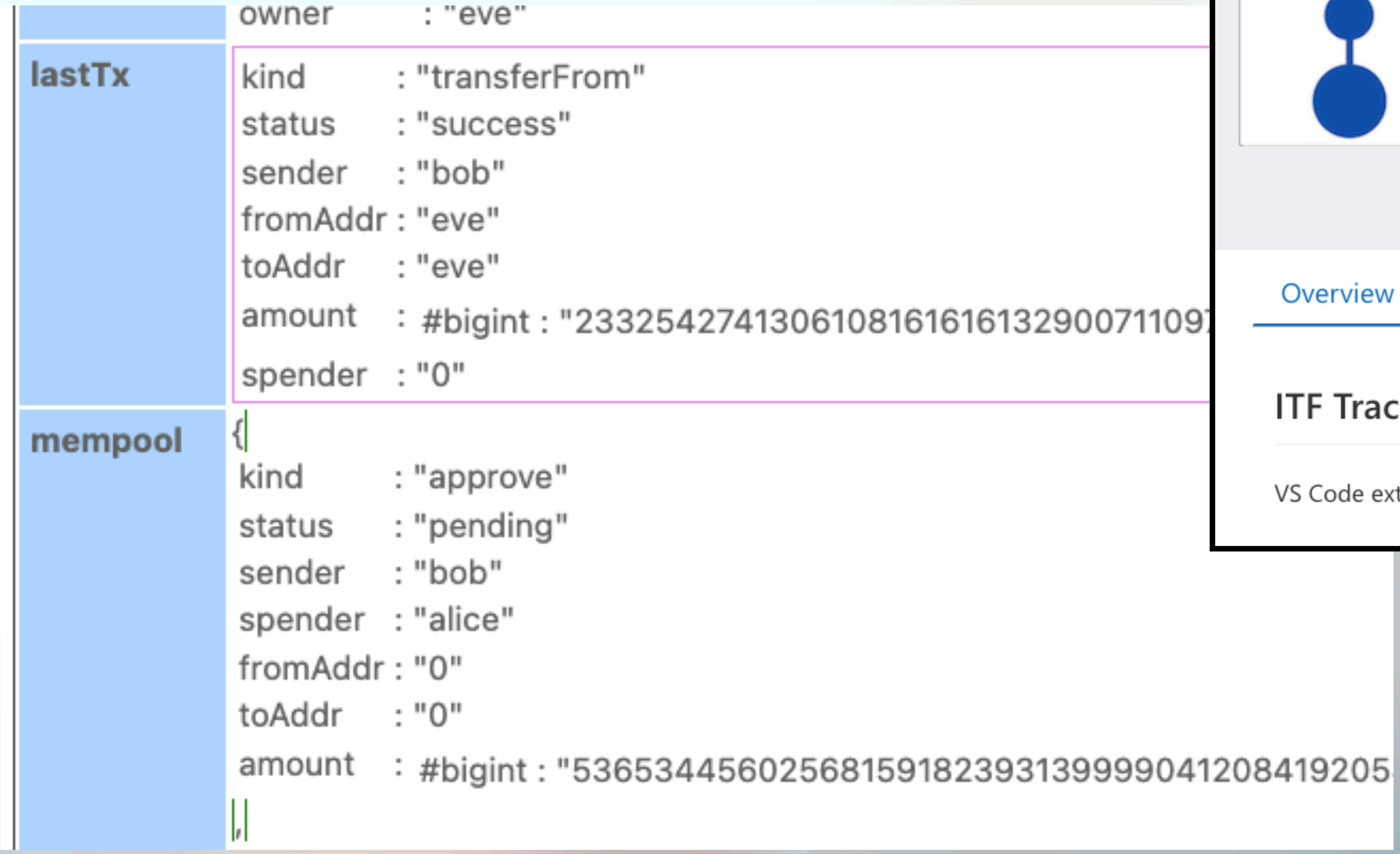


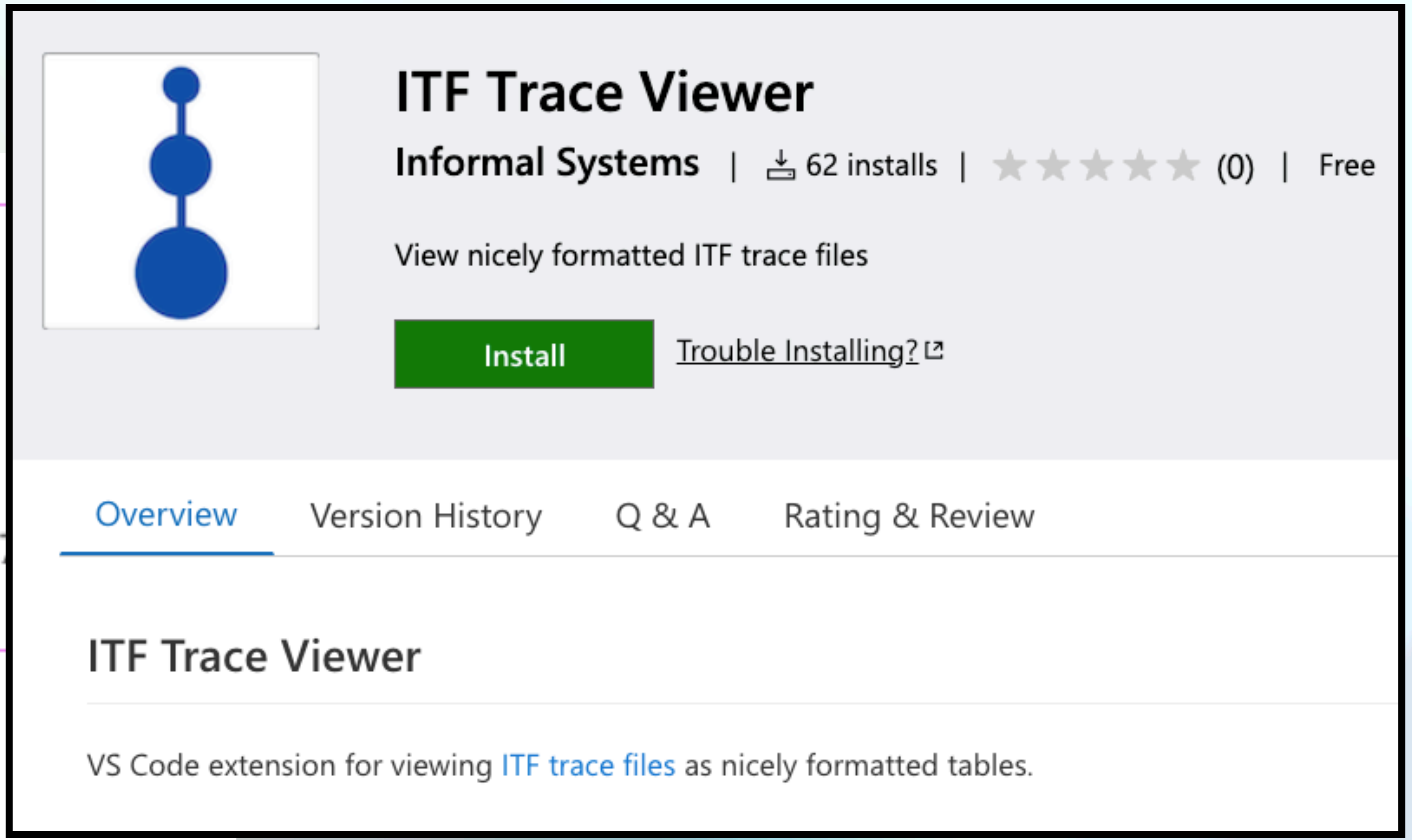
Trace viewer



Hernán Vanzetto

Trace viewer





Hernán Vanzetto

Testing framework

Testing framework

We can use the **test** command to run tests (run operators) against a Quint specification

Testing framework

We can use the **test** command to run tests (run operators) against a Quint specification

Unit tests and property-based tests

Testing framework

We can use the **test** command to run tests (run operators) against a Quint specification

Unit tests and property-based tests

Easy to use with continuos integration

Testing framework

We can use the **test** command to run tests (run operators) against a Quint specification

Unit tests and property-based tests

Easy to use with continuos integration

```
run transferFromWhileApproveInFlightTest = {
 all {
   erc20State' = newErc20("alice", 91),
   mempool' = Set(), lastTx' = NoneTx,
 } // alice sets a high approval for bob
 .then(submit(ApproveTx("alice", "bob", 92)))
 // bob immediately initiates his transaction
 then(submit(TransferFromTx("bob", "alice", "eve", 54)))
 // alice changes her mind and lowers her approval to bob
 ...
}
```

Testing framework

We can use the **test** command to run tests (run operators) against a Quint specification

Unit tests and property-based tests

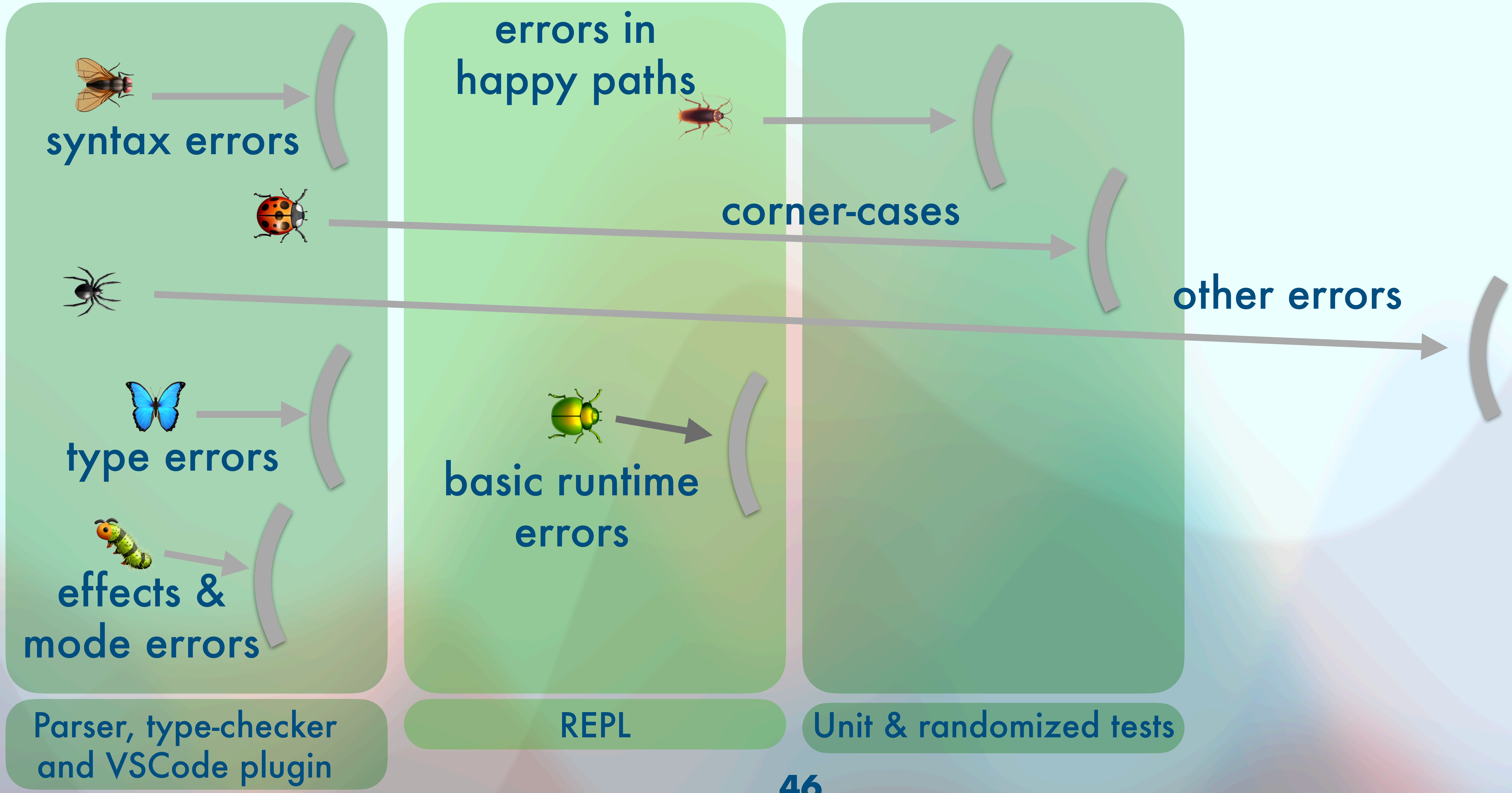
Easy to use with continuos integration

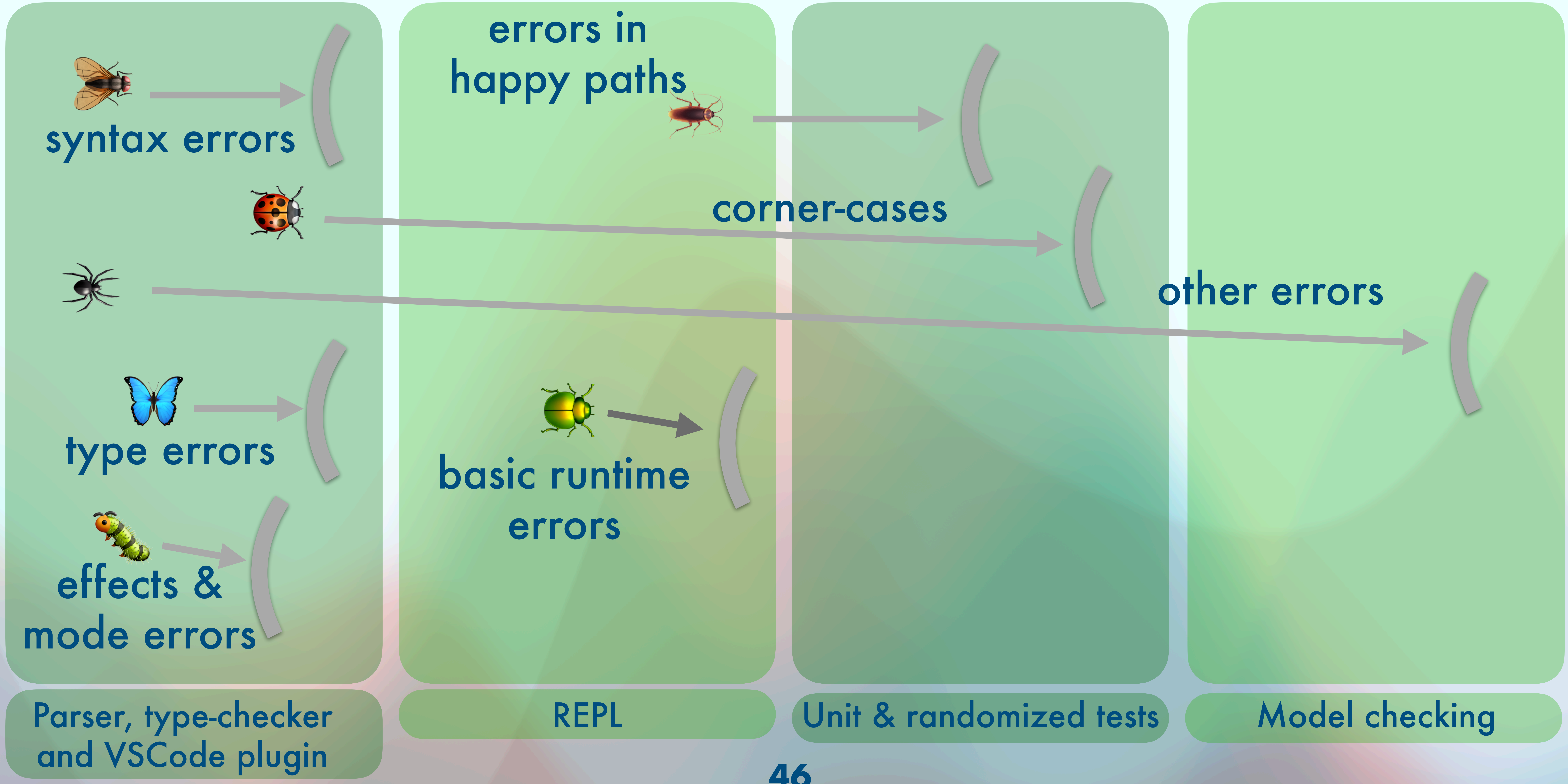
```
run transferFromWhileApproveInFlightTest = {
  all {
    erc20State' = newErc20("alice", 91),
    mempool' = Set(), lastTx' = NoneTx,
  } // alice sets a high approval for bob
  .then(submit(ApproveTx("alice", "bob", 92)))
  // bob immediately initiates his transaction
  then(submit(TransferFromTx("bob", "alice", "eve", 54)))
  // alice changes her mind and lowers her approval to bob
```

```
$ quint test --main=erc20Tests erc20.qnt

  erc20Tests
    ok transferTest passed 10000 test(s)

  1 passing (895ms)
```


errors in
happy paths
syntax errors
corner-cases
other errors
type errors
basic runtime
errors
effects &
mode errors
Parser, type-checker
and VSCode plugin
REPL
Unit & randomized tests


errors in
happy paths
syntax errors
corner-cases
other errors
type errors
basic runtime
errors
effects &
mode errors
Parser, type-checker
and VSCode plugin
REPL
Unit & randomized tests
Model checking

About model-checking Quint specifications

Apalache integration

Apalache integration

The Quint team is working on integrating the Apalache model checker to verify Quint specifications

Apalache integration

The Quint team is working on integrating the Apalache model checker to verify Quint specifications

Goal: check invariants for all executions up to —max-steps

Apalache integration

The Quint team is working on integrating the Apalache model checker to verify Quint specifications

Goal: check invariants for all executions up to —max-steps

Apalache is our in-house symbolic model checker

Apalache integration

The Quint team is working on integrating the Apalache model checker to verify Quint specifications

Goal: check invariants for all executions up to —max-steps

Apalache is our in-house symbolic model checker

The Quint team has already been able to check a Tendermint Quint specification!

When and how we use Quint

When we use it

When we use it

To design new protocols or features from scratch

When we use it

To design new protocols or features from scratch

To formalize existing protocols: from code or documentation

When we use it

To design new protocols or features from scratch

To formalize existing protocols: from code or documentation

To find bugs in existing implementations (audits)

When we use it

To design new protocols or features from scratch

To formalize existing protocols: from code or documentation

To find bugs in existing implementations (audits)

Quint specs have shown potential for onboarding as well

Conclusions

Conclusions

Conclusions

Our goal is that "anyone" can formalize and check their protocols

Conclusions

Our goal is that "anyone" can formalize and check their protocols

The Quint language and the tools around it aim at enabling this

Conclusions

Our goal is that "anyone" can formalize and check their protocols

The Quint language and the tools around it aim at enabling this

By having a syntax that's similar to programming languages and providing an experience similar to what software development looks for engineers

Thanks!

manuel@informal.systems