Algorand

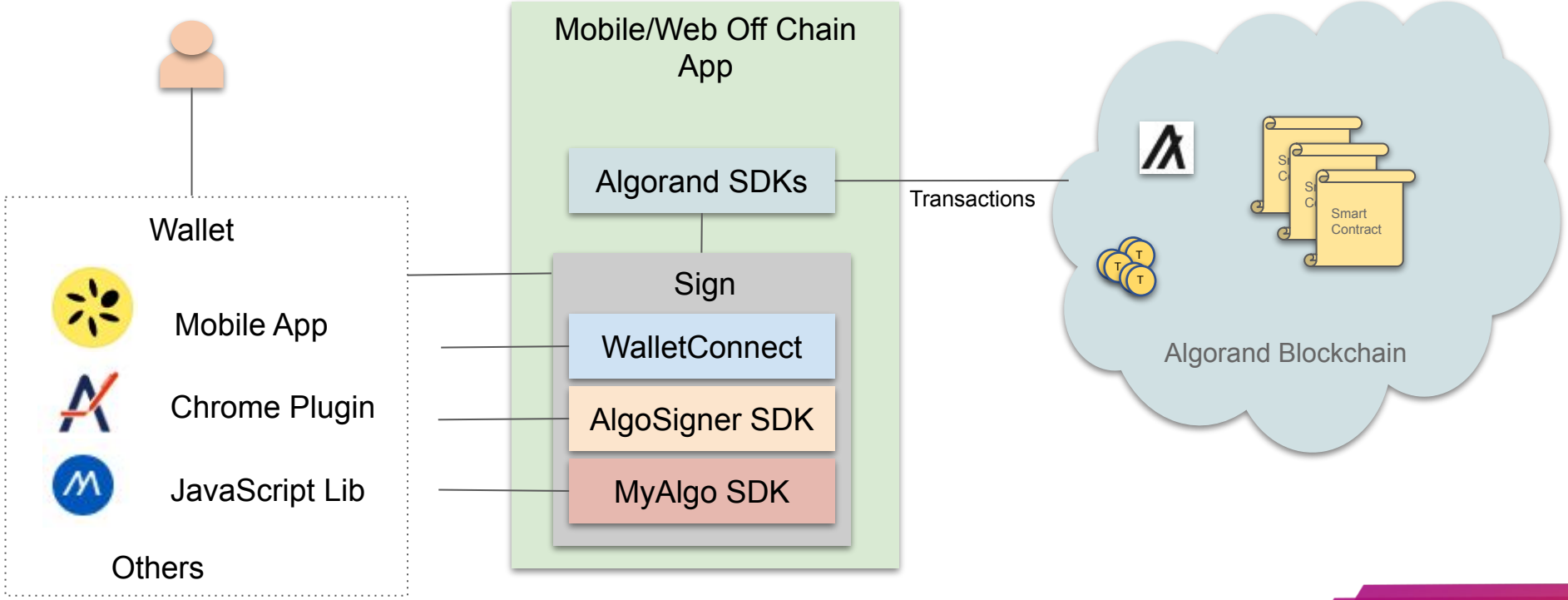
Algorand Smart Contracts

Jason Weathersby, Sr Director Developer Relations,
Algorand
@JasonWeathersby

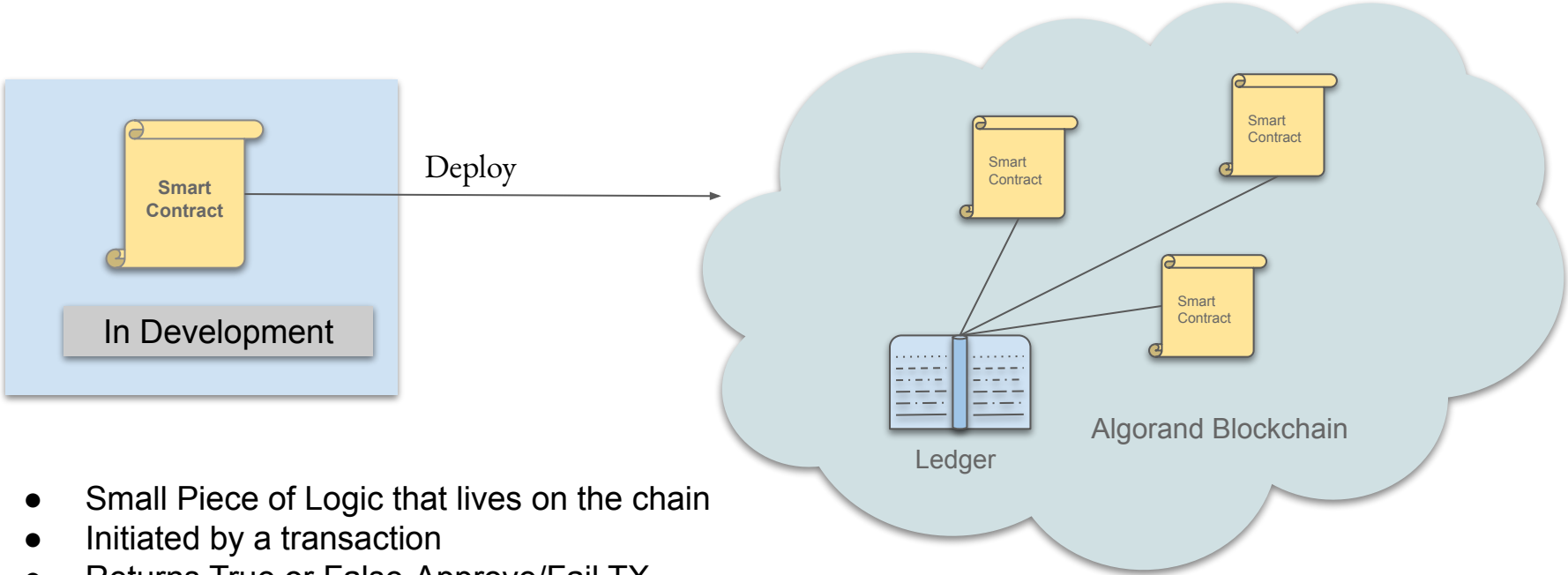
Agenda

- Overview of Smart Contracts
- Transaction Execution Approval Language (TEAL)
- Developing Smart Contracts with Python

Algorand Dapp Architecture

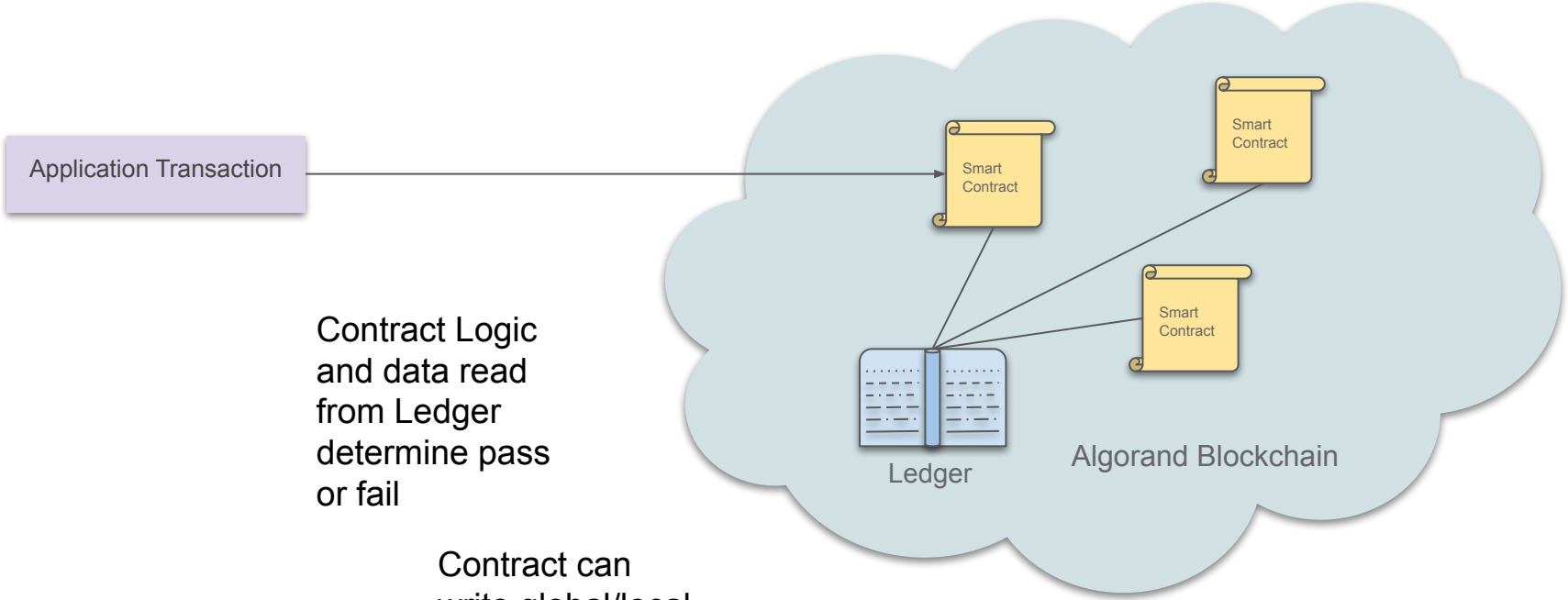


What is a Smart Contract



- Small Piece of Logic that lives on the chain
- Initiated by a transaction
- Returns True or False-Approve/Fail TX
- Can be grouped with other Transactions
- Written in TEAL or PyTeal or Generated By Reach Framework

Smart Contract High Level

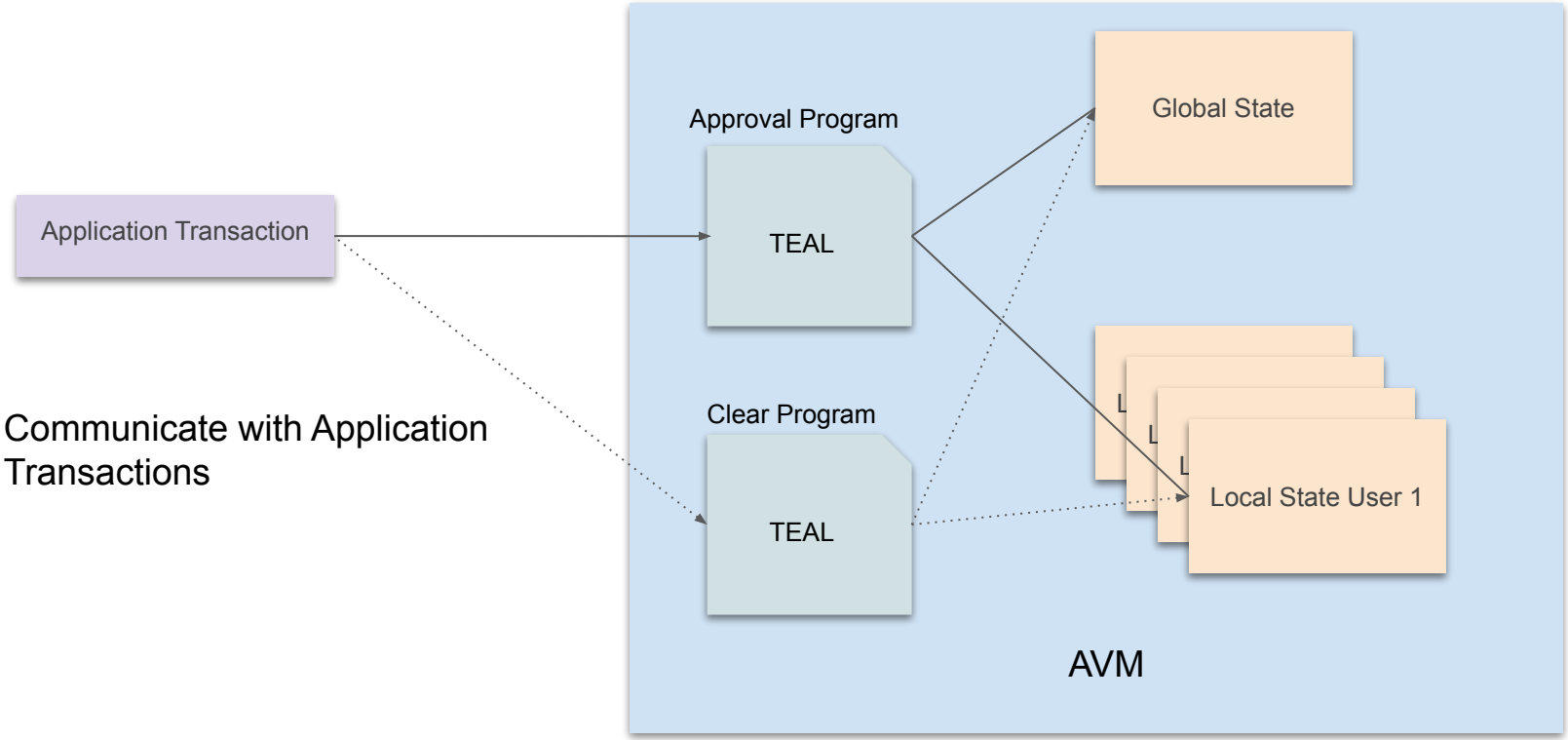


Contract Logic and data read from Ledger determine pass or fail

Contract can write global/local values to the ledger

Contract can also initiate new transactions

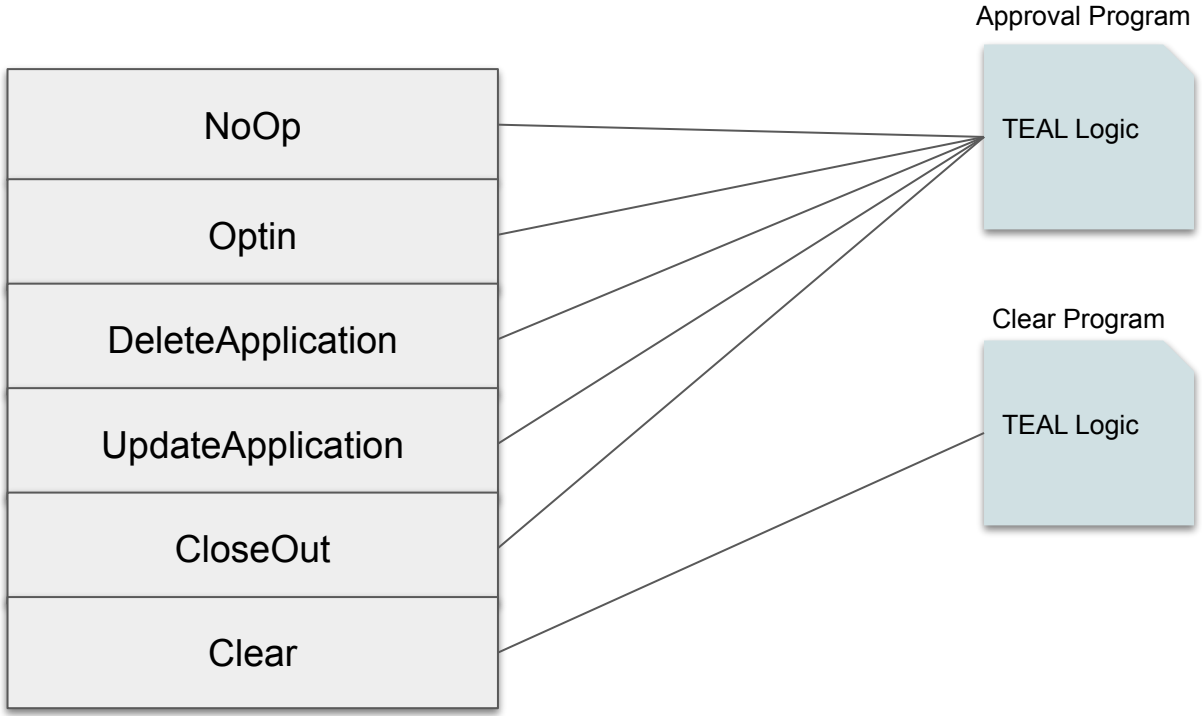
Smart Contracts aka Apps



Communicate with Application Transactions

Transaction Sub-Types for Application

Used to Communicate With Smart Contract



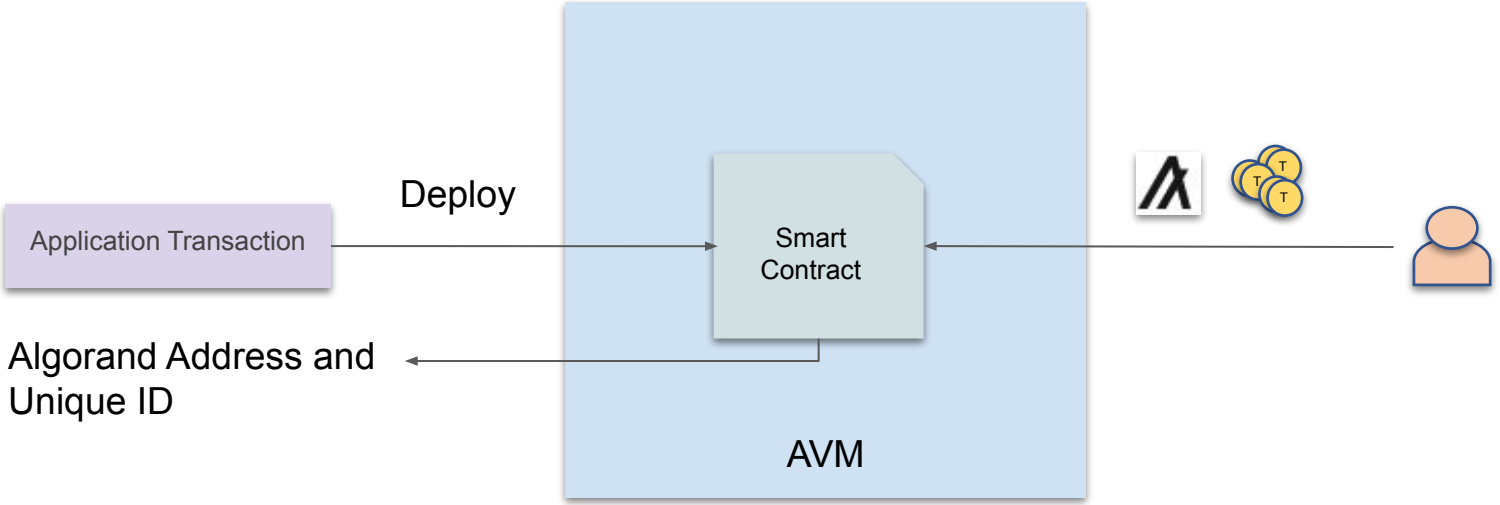
Graceful

Non-Graceful
ie Will clear
regardless

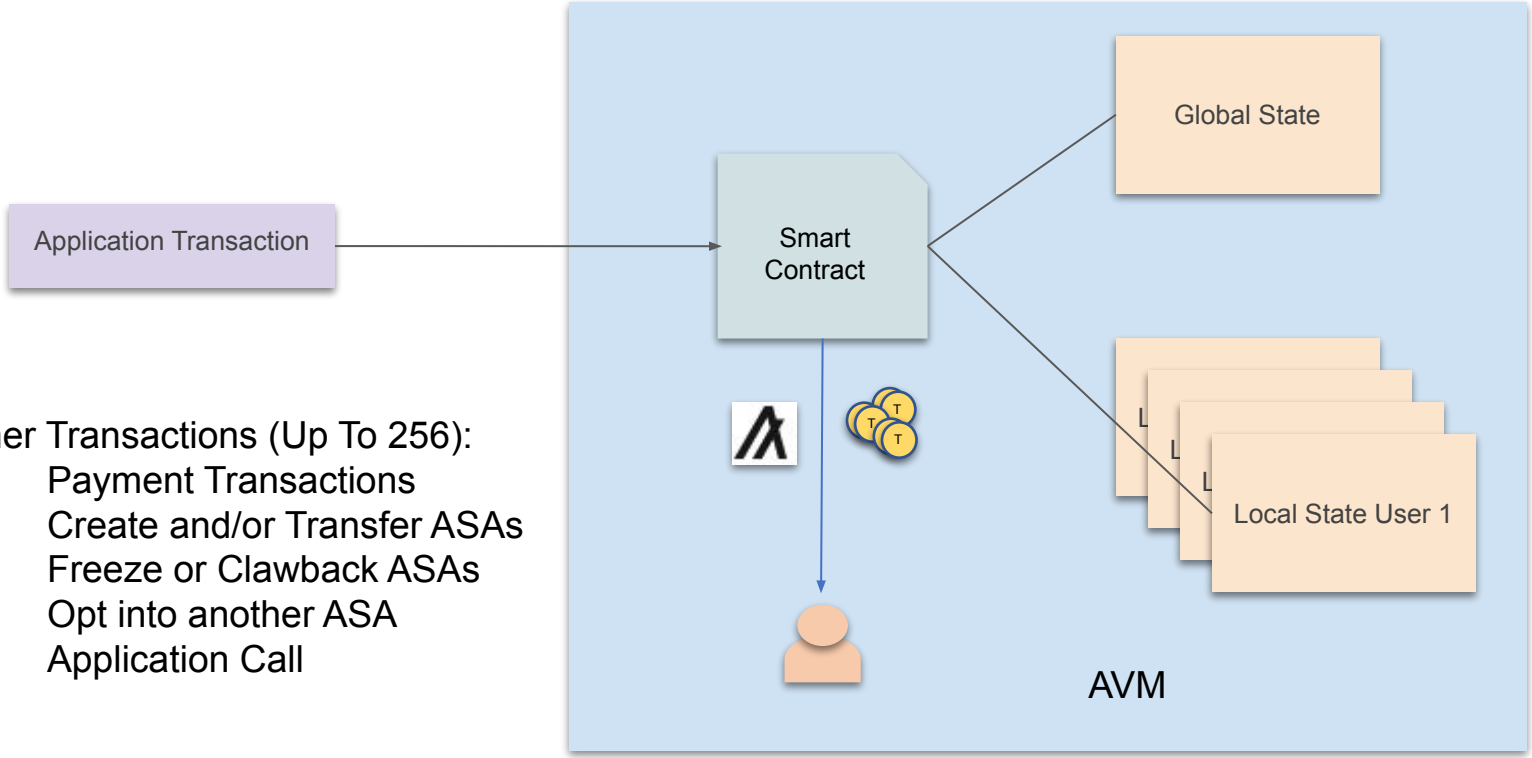
Algorand

Smart Contracts as Escrows

Smart Contract Escrow



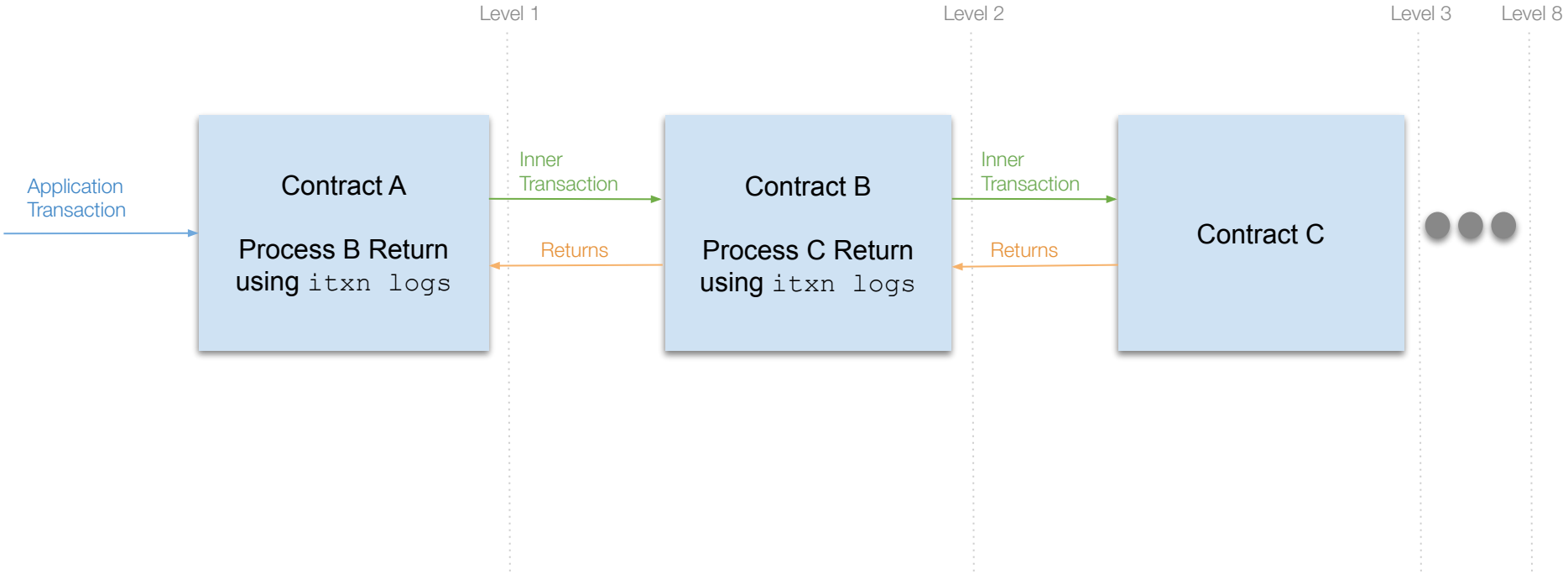
Smart Contract Escrow



Inner Transactions (Up To 256):

- Payment Transactions
- Create and/or Transfer ASAs
- Freeze or Clawback ASAs
- Opt into another ASA
- Application Call

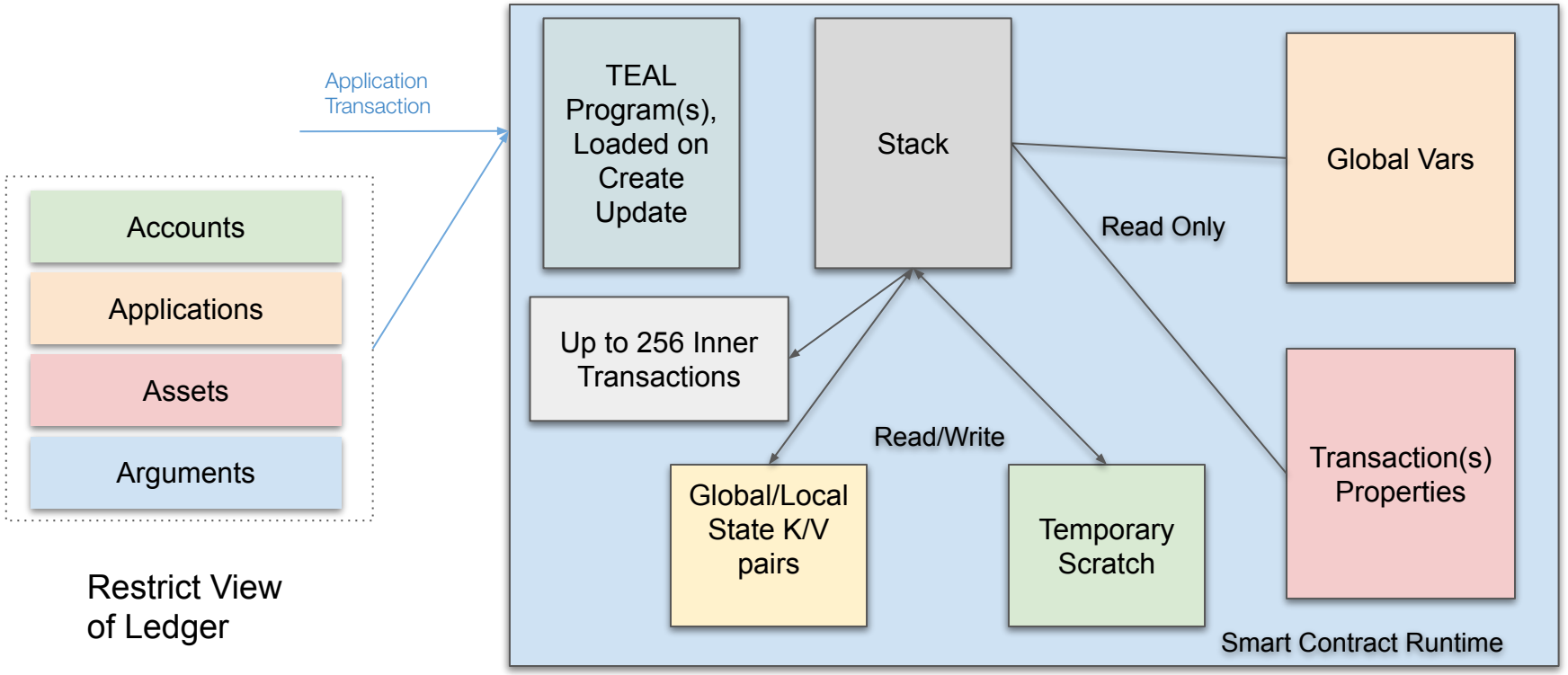
Smart Contract to Smart Contract Calling



Algorand

Runtime Architecture

Smart Contract Runtime Architecture



Algorand

Quick Glance At ABI

Application Binary Interface (ABI)

- Standard for Describing Methods, Interfaces and Contracts
- Described in JSON
- SDKs read and convert to proper smart contract calls, parameters and return types

```
1  {
2      "name":"add",
3      "desc":"Add x to Global Int",
4      "args":[ { "type":"uint64","name":"parm1", "desc":"first parameter"}],
5      "returns": {"type":"uint64"}
6  }
```

```
"add(uint64)uint64"
```

Method Signature

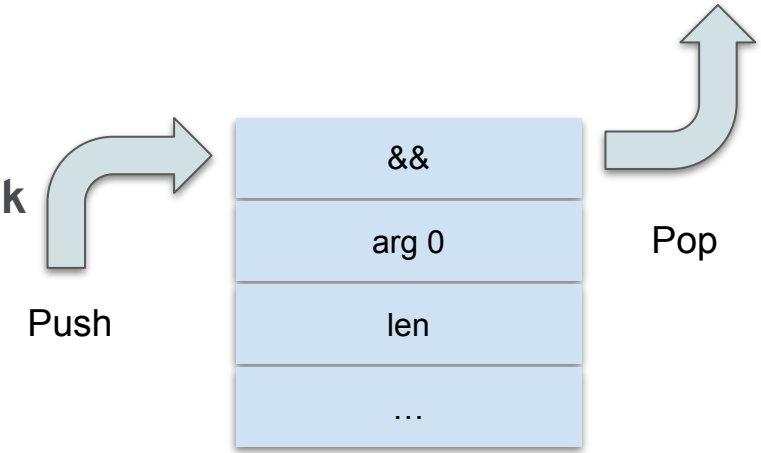
https://github.com/algorandfoundation/ARCs/blob/main/ARCs/arc-0004.md
https://developer.algorand.org/articles/contract-to-contract-calls-and-an-abi-come-to-algorand/
https://www.youtube.com/watch?v=ajAAy9d3B_0

Algorand

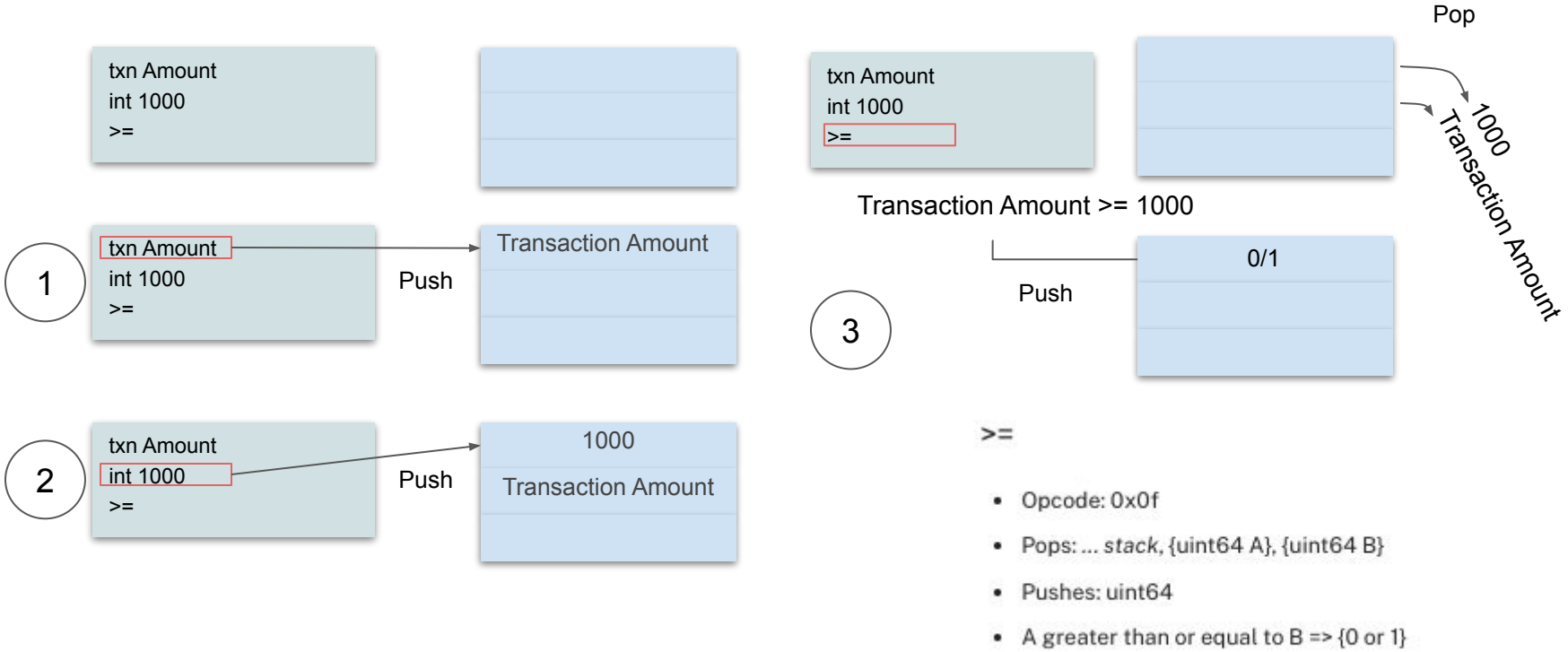
Transaction Execution Appoval Language (TEAL)

TEAL - Transaction Execution Approval Language

- Bytecode based stack language
- Turing Complete
- Looping and Subroutines
- Only Uint64 and Byte[] allowed on stack
- True or False - Positive value on stack
- > 140 Opcodes
- PyTeal library to write in python



Simple Stack Example



>=

- Opcode: 0x0f
- Pops: ... *stack*, {uint64 A}, {uint64 B}
- Pushes: uint64
- A greater than or equal to B => {0 or 1}

Opcodes

txn f

- Opcode: 0x31 {uint8 transaction field index}
- Pops: *None*
- Pushes: any
- push field F of current transaction to stack

`txn` Fields (see transaction reference):

Index	Name	Type	Notes
0	Sender	[]byte	32 byte address
1	Fee	uint64	micro-Algos

Opcode Reference Document

Algorand

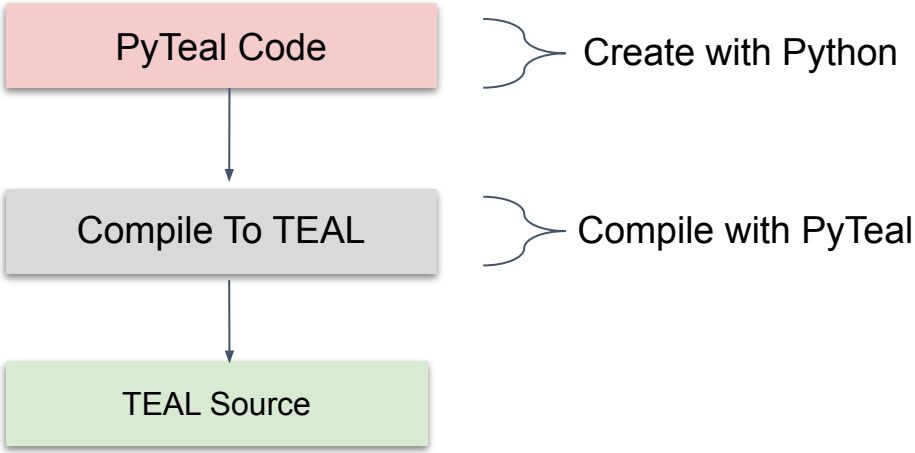
Simple Addition TEAL Demo

Algorand

PyTeal

PyTeal

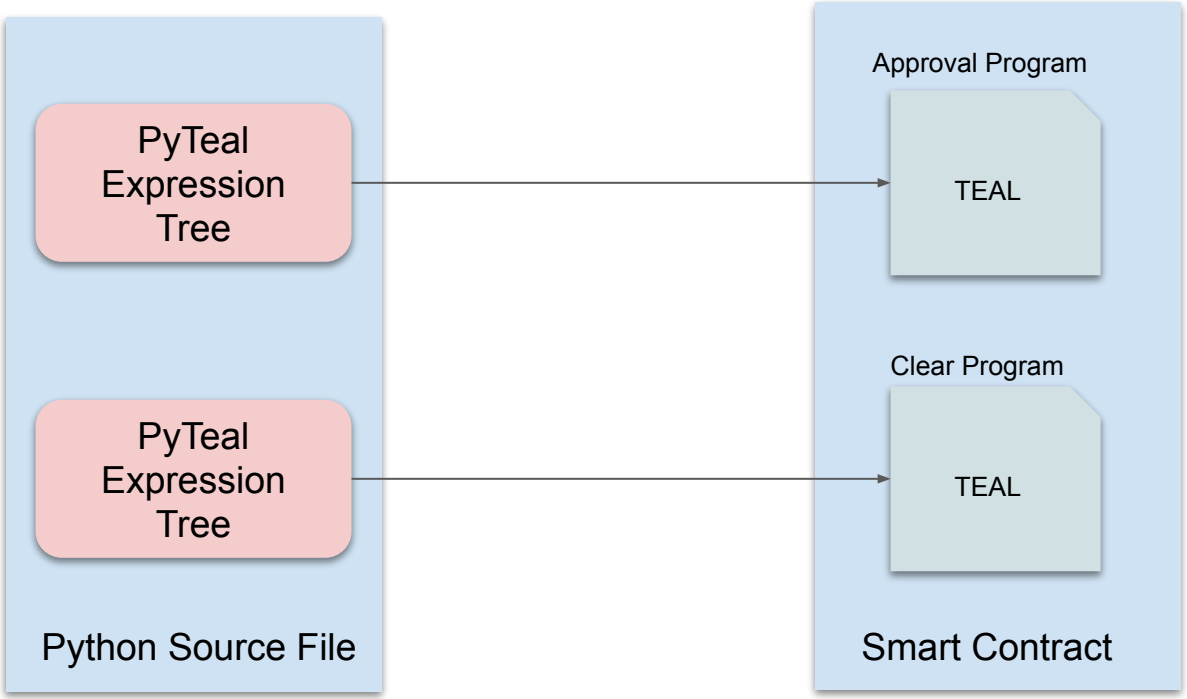
- **Python Library that produces TEAL**
- **TEAL compiled to bytecode and runs on Algorand AVM**
- **Contracts are created as PyTeal Expression Trees**



https://pyteal.readthedocs.io/en/stable/index.html

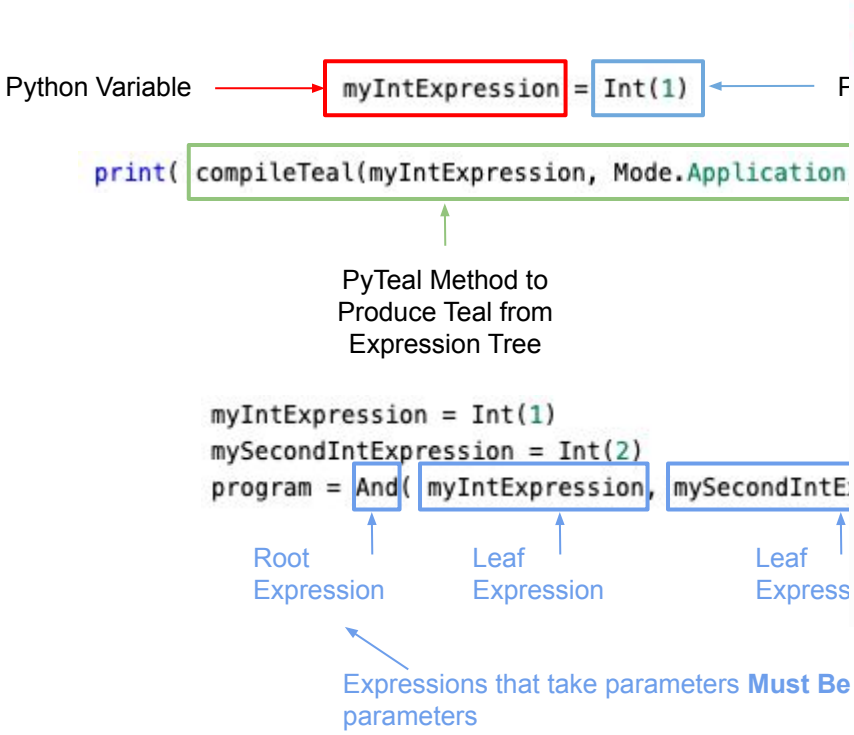
Algorand

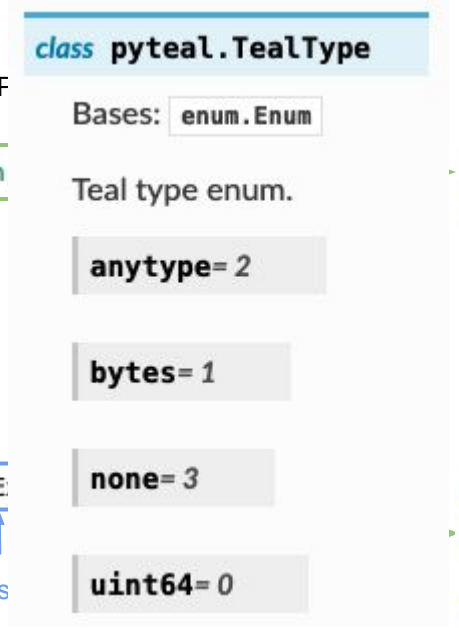
PyTeal to Teal



PyTeal Expressions

- PyTeal Object that compiles to TEAL
- Expressions are created using PyTeal Class methods
- Expression Trees are just a collection of connected Expressions



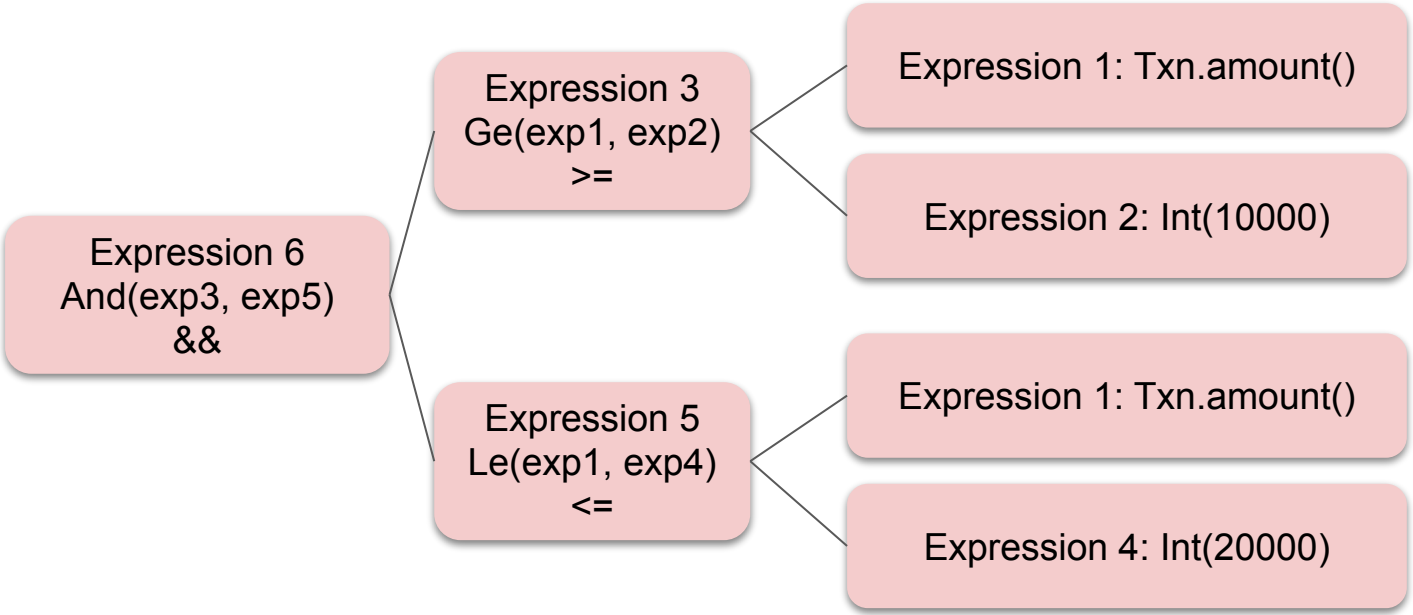



```
#pragma version 5
int 1
return
```

```
#pragma version 5
int 1
int 2
&&
return
```

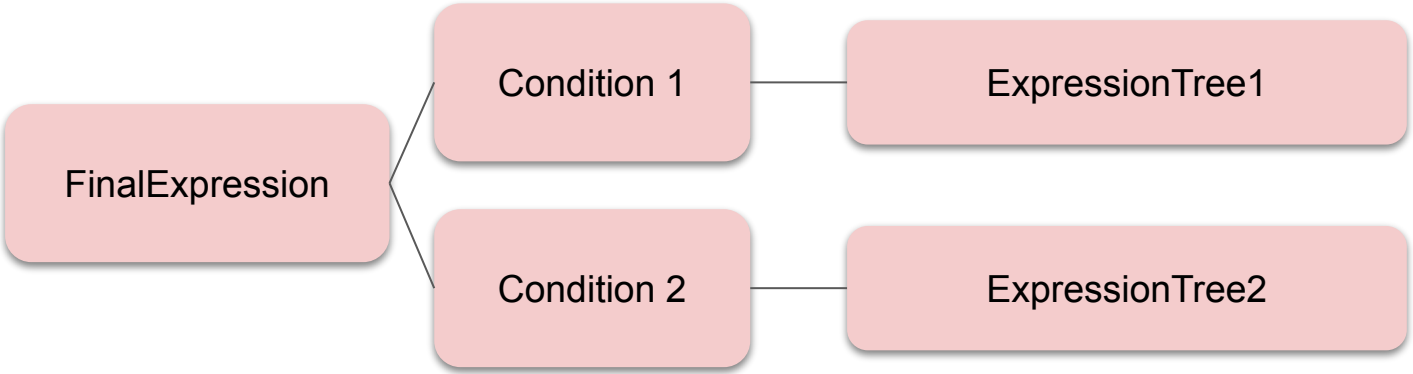
PyTeal Expression Tree Example

Transaction Amount >= 10000 and <= 20000




```
And(Txn.amount() >= Int(10000), Txn.amount() <= Int(20000))
```

PyTeal Expression Tree Control Flow

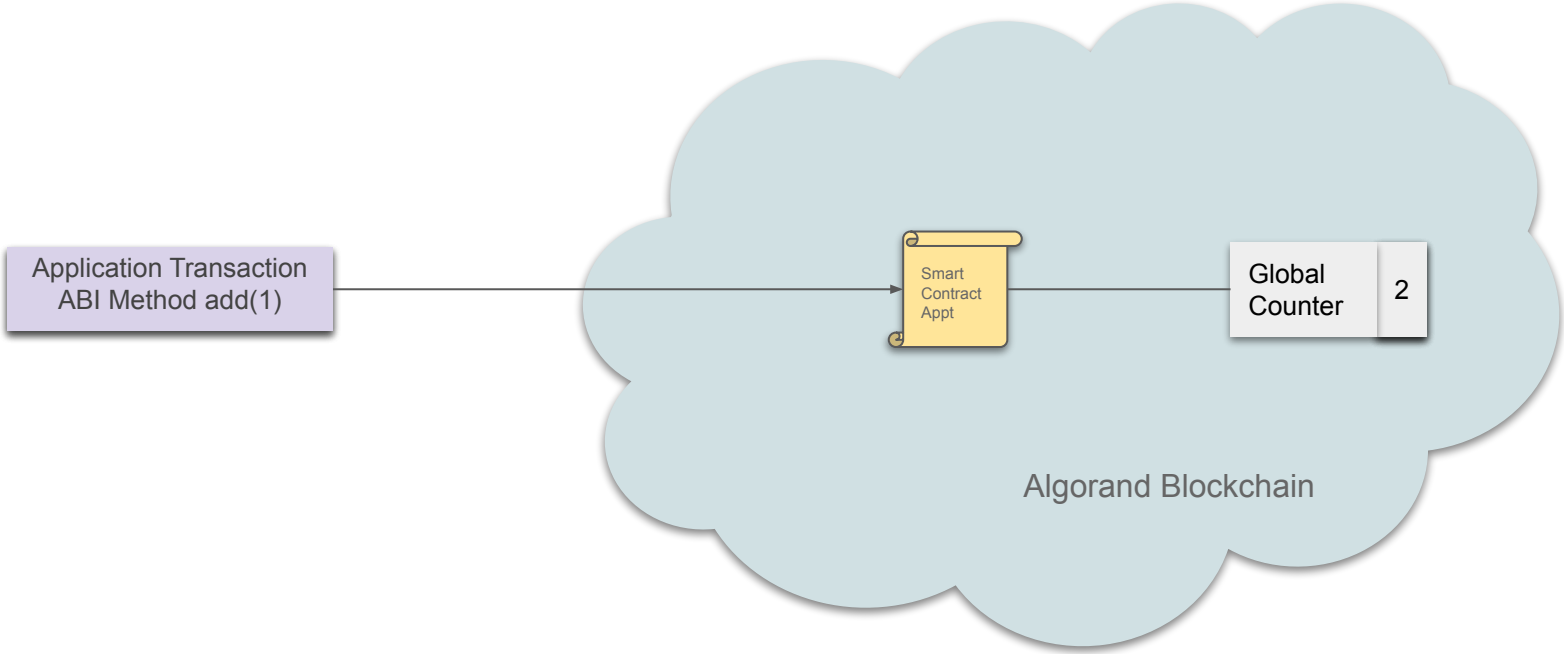



```
return Cond(
    [Int(1) == Int(0), tree1],
    [Int(1) == Int(1), tree2],
)
```

Algorand

PyTeal Example

Incrementing Counter



Transaction Properties

Operator	Type	Min TEAL Version	Notes
`Txn.sender()`	`TealType.bytes`	2	32 byte address
`Txn.fee()`	`TealType.uint64`	2	in microAlgos
`Txn.first_valid()`	`TealType.uint64`	2	round number
`Txn.last_valid()`	`TealType.uint64`	2	round number
`Txn.note()`	`TealType.bytes`	2	
`Txn.lease()`	`TealType.bytes`	2	
`Txn.receiver()`	`TealType.bytes`	2	32 byte address
`Txn.amount()`	`TealType.uint64`	2	in microAlgos
`Txn.close_remainder_to()`	`TealType.bytes`	2	32 byte address

```
Gtxn[0].sender()  #tran group size 0 based
Global.group_size() #get size of group
```

https://pyteal.readthedocs.io/en/stable/accessing_transaction_field.html#id1

Transaction Routing

```
Cond([test-expr-1, body-1],
     [test-expr-2, body-2],
     . . . )
```

First Success Breaks Out Of Cond()

All Expressions for body must eval to same Teal Type

```
# new router coming that will simplify this
return Cond(
    [Txn.application_id() == Int(0), handle_creation],
    [Txn.on_completion() == OnComplete.OptIn, Reject()],
    [Txn.on_completion() == OnComplete.CloseOut, Approve()],
    [Txn.on_completion() == OnComplete.UpdateApplication, Approve()],
    [Txn.on_completion() == OnComplete.DeleteApplication, Approve()],
    [Txn.on_completion() == OnComplete.NoOp, handle_noop]
)
```

Create and Initialize the Global counter to 0

```
Seq([
    App.globalPut(Bytes("creator"), Txn.sender()),
    Return(Int(1))
])
```

All Expressions But The Last Must Resolve to Teal Type of None

Not Allowed
Seq(Int(1), Int(1))

```
App.globalPut(Bytes("status"), Bytes("active")) # write a by
App.globalPut(Bytes("total supply"), Int(100)) # write a uin
```

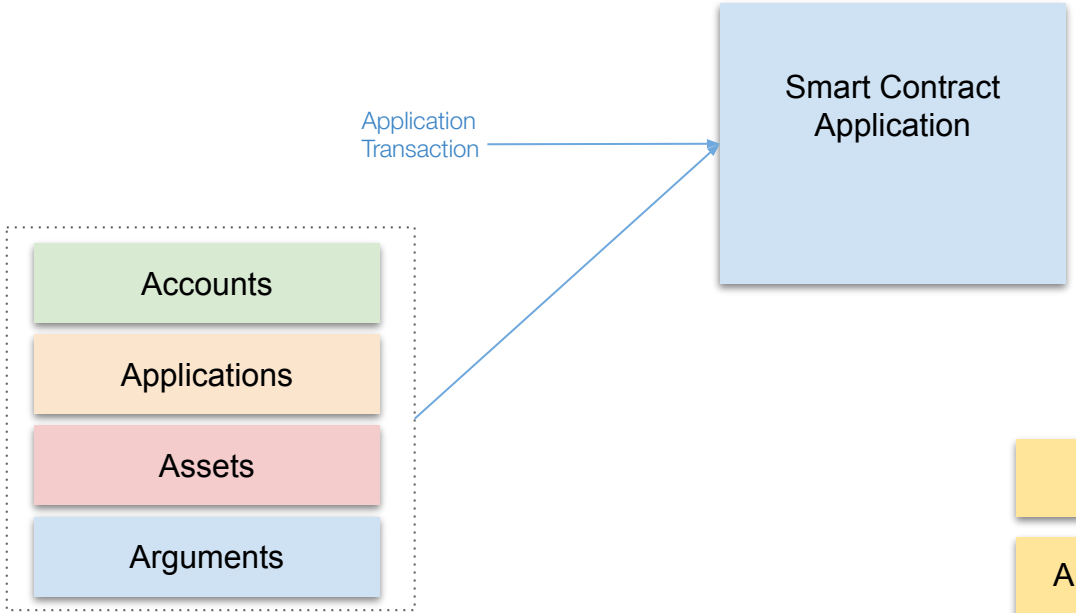
Only byte[] or uint64 Allowed

```
handle_creation = Seq(
    App.globalPut(Bytes("Count"), Int(0)),
    Approve()
)
```

https://pyteal.readthedocs.io/en/stable/control_structures.html#chaining-expressions-seq

https://pyteal.readthedocs.io/en/stable/state.html#writing-global-state

Application Call Arrays




```
Txn.accounts[0]
Txn.assets[0]
Txn.applications[0]
Txn.application_args[0]
```

Accounts[0] is Always Sender

Applications[0] is Always Current App

Arguments[0] Contains Meth Sig for ABI Calls

Switch on Specific ABI Method - call add or sub

```
Txn.application_args.length() # get the number of application arguments in the transaction
Txn.application_args[0] # get the first application argument
Txn.application_args[1] # get the second application argument
```

PyTeal Overloads Python Operators

`Mod(a, b)`	`a % b`	*a % b*, modulo operation	`Int(7) % Int(3)`
`Eq(a, b)`	`a == b`	*1* if *a* equals *b*, *0* otherwise	`Int(7) == Int(7)`
`Neq(a, b)`	`a != b`	*0* if *a* equals *b*, *1* otherwise	`Int(7) != Int(7)`

```
34  # new router coming that will simplify this
35  handle_noop = Cond(
36      [Txn.application_args[0] == MethodSignature("add(uint64)uint64"), Return(add(Btoi(Txn.application_args[1])))],
37      [Txn.application_args[0] == MethodSignature("sub(uint64)uint64"), Return(deduct(Btoi(Txn.application_args[1])))],
38  )
```

https://pyteal.readthedocs.io/en/stable/accessing_transaction_field.html?#transaction-array-fields

https://pyteal.readthedocs.io/en/stable/arithmetic_expression.html

Implement Add and Deduct

```
App.globalGet(Bytes("status"))
App.globalGet(Bytes("total supply"))
```

Do not let the Counter go below 0

```
If(test-expr, then-expr)
```

If(test-expr, then-expr, else-expr)

https://pyteal.readthedocs.io/en/stable/state.html#reading-global-state

https://pyteal.readthedocs.io/en/stable/control_structures.html#simple-branching-if

Add Function

```
4   """Basic Counter Application"""
5   #specifically for ABI, future enhancements will improve this
6   return_prefix = Bytes("base16", "0x151f7c75")  # Literally hash('return')[:4]
7
8   @Subroutine(TealType.uint64)
9   def add(x):
10      return Seq(
11          App.globalPut(Bytes("Count"), App.globalGet(Bytes("Count")) + x),
12          Log(Concat(return_prefix, Itob(App.globalGet(Bytes("Count"))))),
13          Approve()
14      )
```

ABI Return Marker

ABI Concat Return Marker With Updated Value

Deduct Function

```
16  @Subroutine(TealType.uint64)
17  def deduct(x):
18      return Seq(
19          If(App.globalGet(Bytes("Count")) >= x,
20              App.globalPut(Bytes("Count"), App.globalGet(Bytes("Count")) - x),
21          ),
22          Log(Concat(return_prefix, Itob(App.globalGet(Bytes("Count"))))),
23          Approve()
24      )
```

Handle Negative Number Check

Algorand

Compile the Smart Contract

Algorand

Call the Smart Contract

Call the Smart Contract from Python

```
33      # Create group txn
34      sp = client.suggested_params()
35      # contruct the ATC (Which supports ABI)
36      atc = AtomicTransactionComposer()
37      # Create signer object
38      signer = AccountTransactionSigner(pk)
39      # Construct the method object
40      meth1 = Method("add", [Argument("uint64")], Returns("uint64"))
41      meth2 = Method("sub", [Argument("uint64")], Returns("uint64"))
42      # Add a method call to the smart contract
43      atc.add_method_call(app_id, meth1, addr, sp, signer, method_args=[3])
44      atc.add_method_call(app_id, meth2, addr, sp, signer, method_args=[1])
45
46      # Execute the transaction
47      result = atc.execute(client, 3);
48      for result in result.abi_results:
49          print("ABI Return Value: ",result.return_value)
50
```

https://developer.algorand.org/docs/get-details/atc/

Algorand

Smart Contract Execution

Link To Presentation

-

Algorand

Resources

- **Discord:** https://discord.com/invite/84AActu3at
- Developer Portal (Documentation and Tutorials): https://developer.algorand.org/
- Forum: https://forum.algorand.org/
- GitHub: https://github.com/algorand
- OFFICE HOURS sign up: https://www.algorand.com/developers