



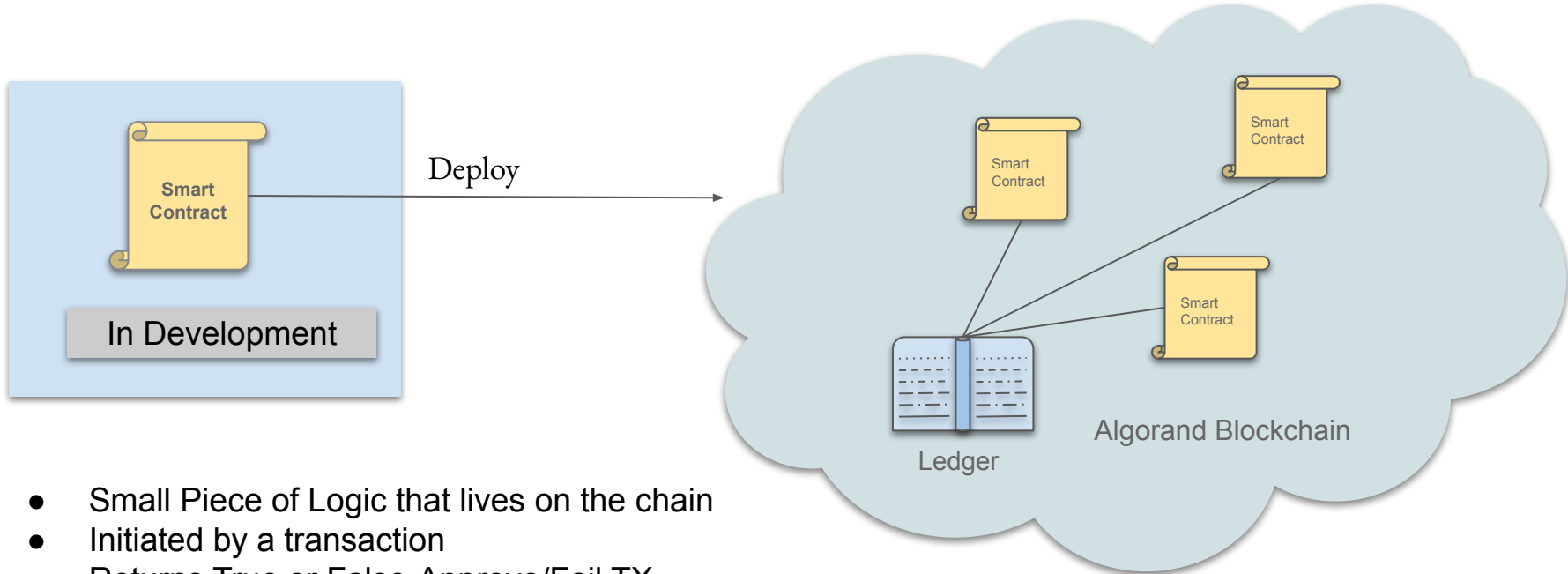
# Algorand AVM 1.0 Smart Contracts

Jason Weathersby, Sr Director Developer relations  
@JasonWeathersby

# Agenda

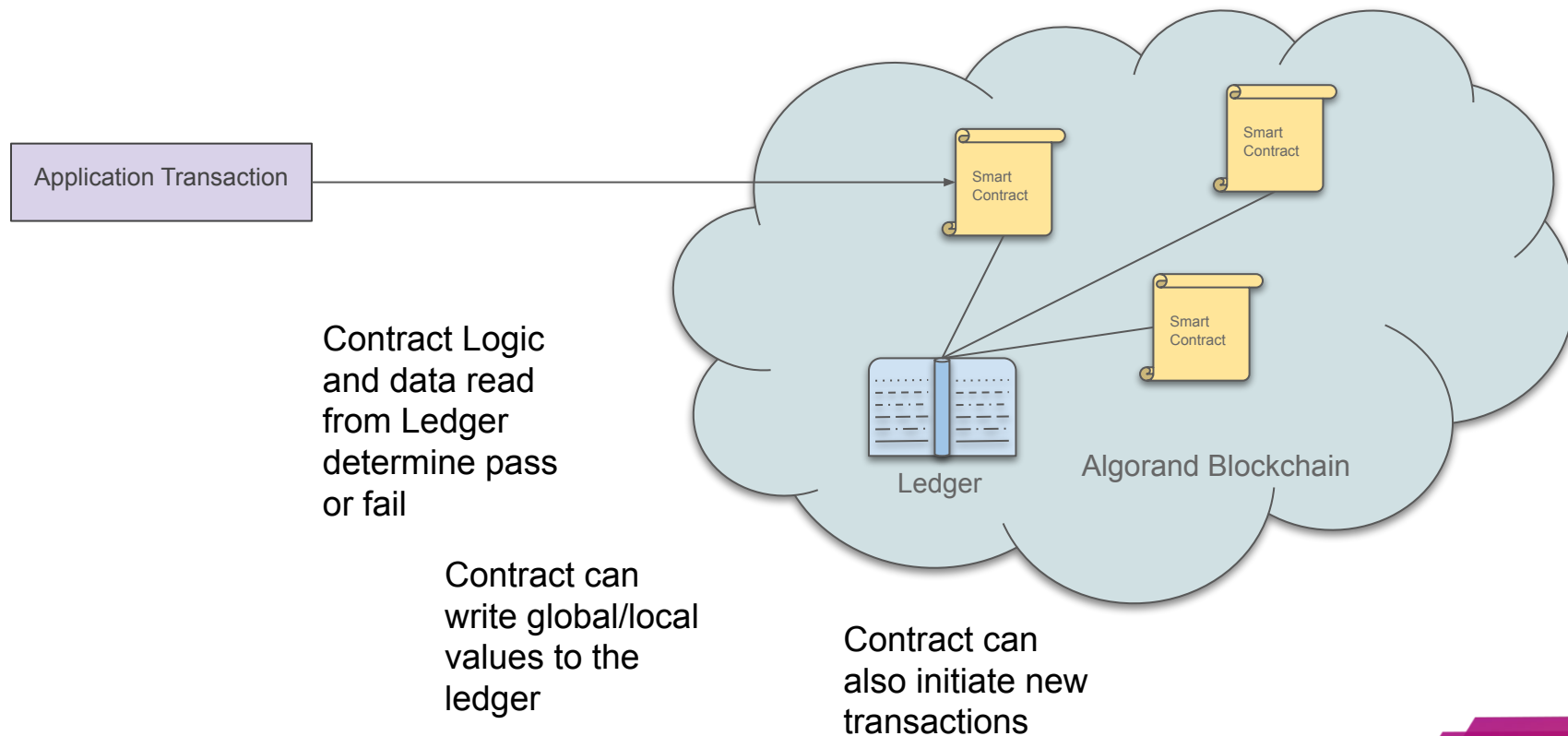
- Overview of Smart Contracts
- Transaction Execution Approval Language (TEAL)
- Developing Smart Contracts with Python

# What is a Smart Contract

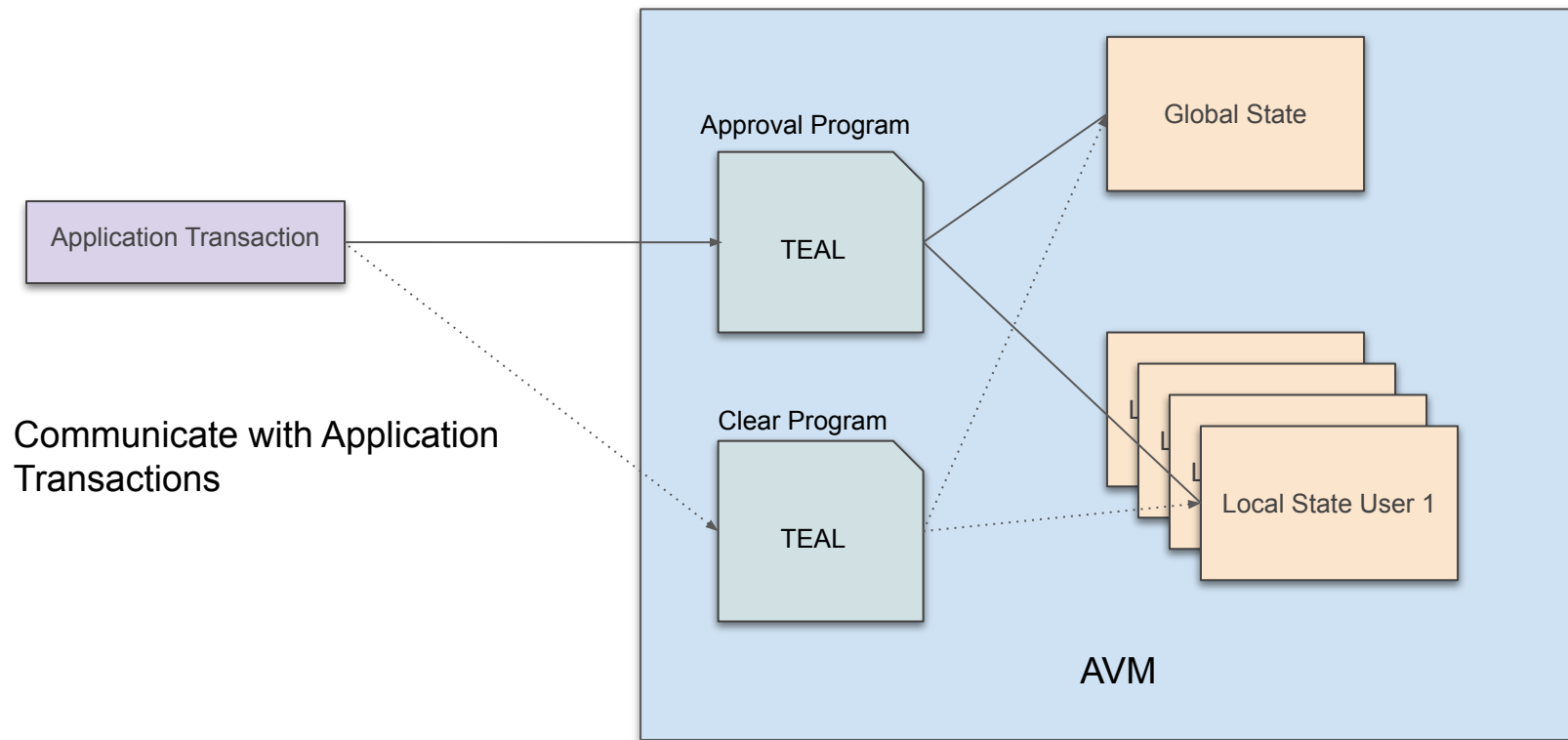


- Small Piece of Logic that lives on the chain
- Initiated by a transaction
- Returns True or False-Approve/Fail TX
- Can be grouped with other Transactions
- Written in TEAL or PyTEAL

# Smart Contract High Level

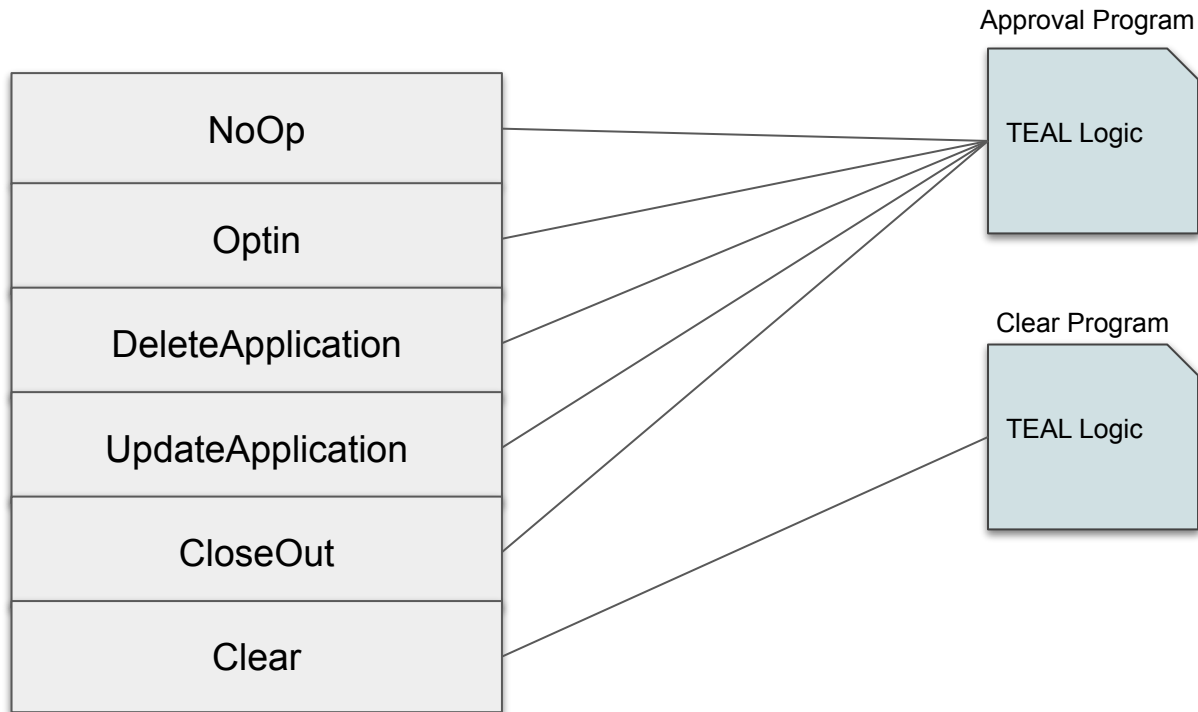


# Smart Contracts aka Apps



# Transaction Sub-Types for Application

Used to Communicate With Smart Contract



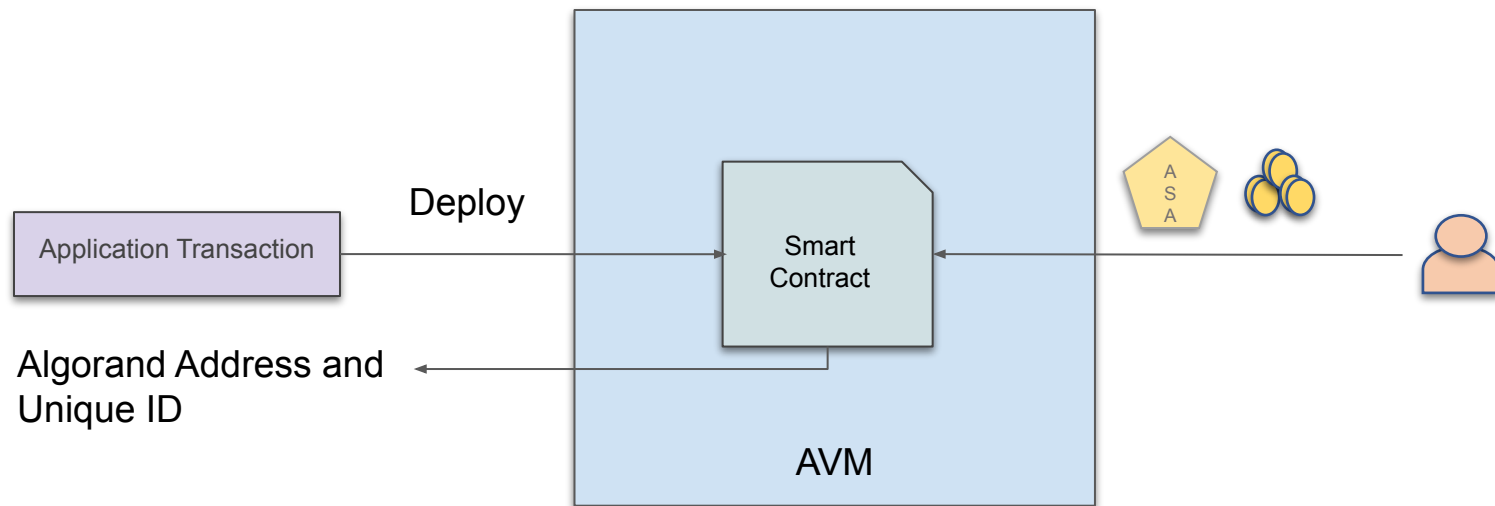
Graceful

Non-Graceful  
ie Will clear  
regardless



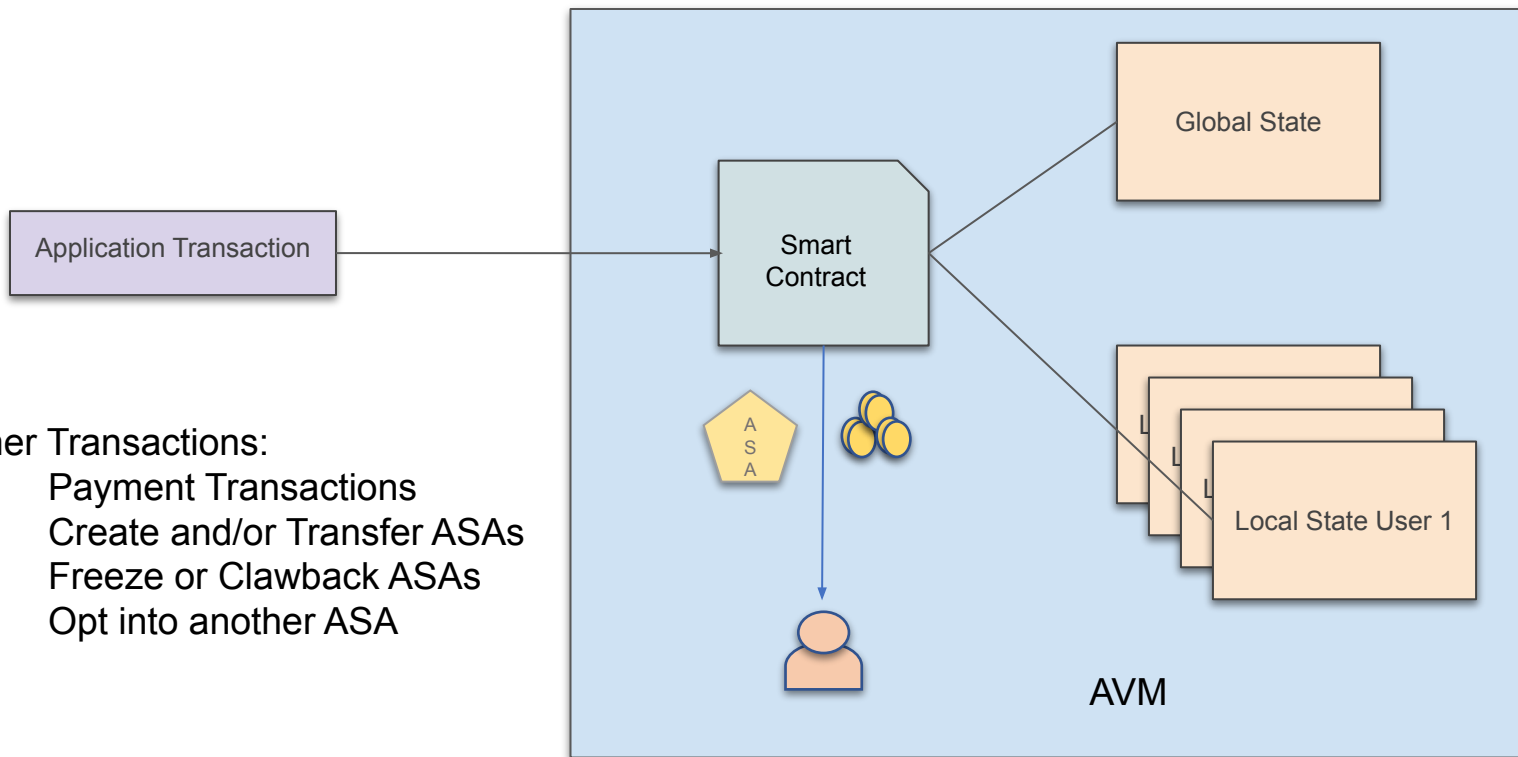
# Smart Contracts as Escrows

# Smart Contract Escrow





# Smart Contract Escrow



## Inner Transactions:

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

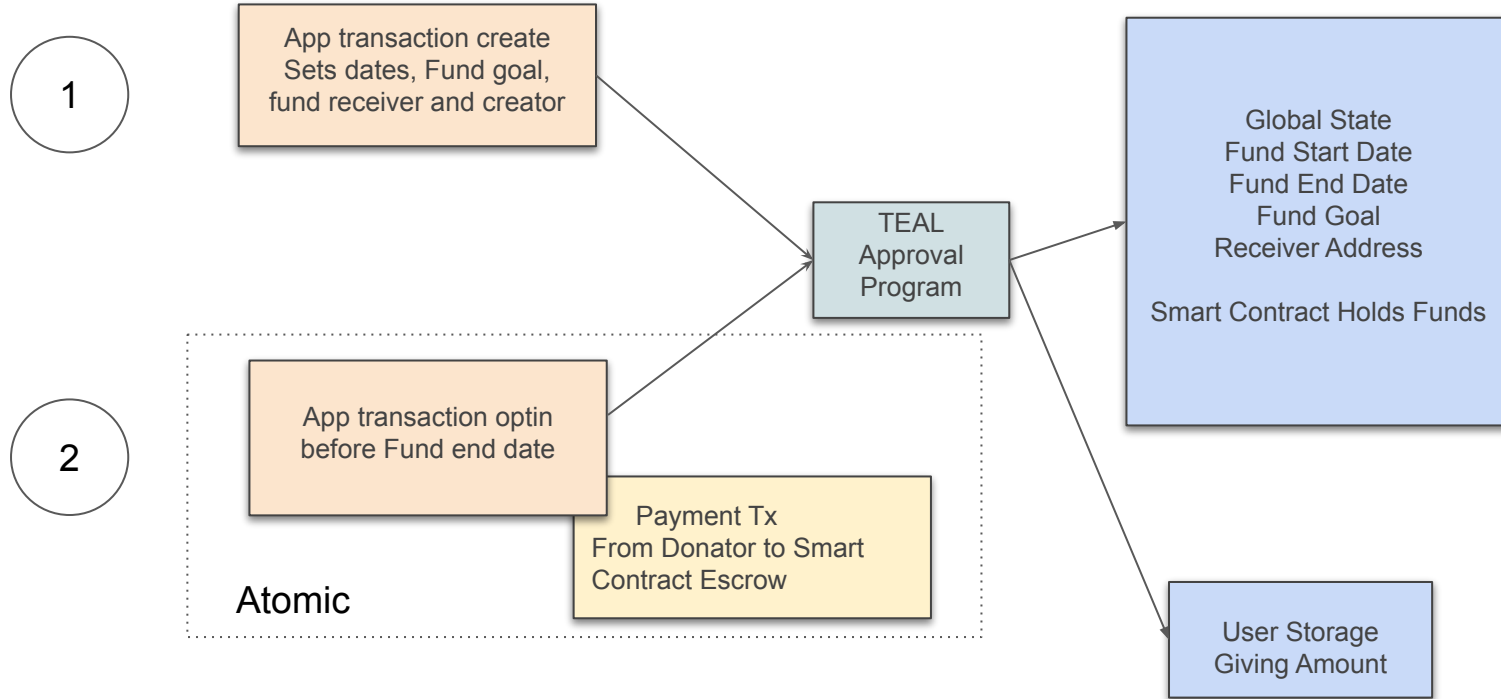
# Other Smart Contract Usages

- Defi Apps (AMM, lending, etc)
- Crowdfunding
- Voting
- Material Tracking
- Lottery
- Seating
- Anywhere a global variable is needed
- Anywhere a individual users values need to be manipulated



## Case Study - Crowdfunding

# Combining Technologies - Crowdfunding



# Payout

3

App transaction call  
receive Funds if Fund End  
date has passed and fund  
goal achieved

4

App transaction call  
Recover Donation if End  
date has passed and fund  
goal not achieved

TEAL  
Approval  
Program  
Uses Inner  
Transactions  
to Pay out or  
return funds

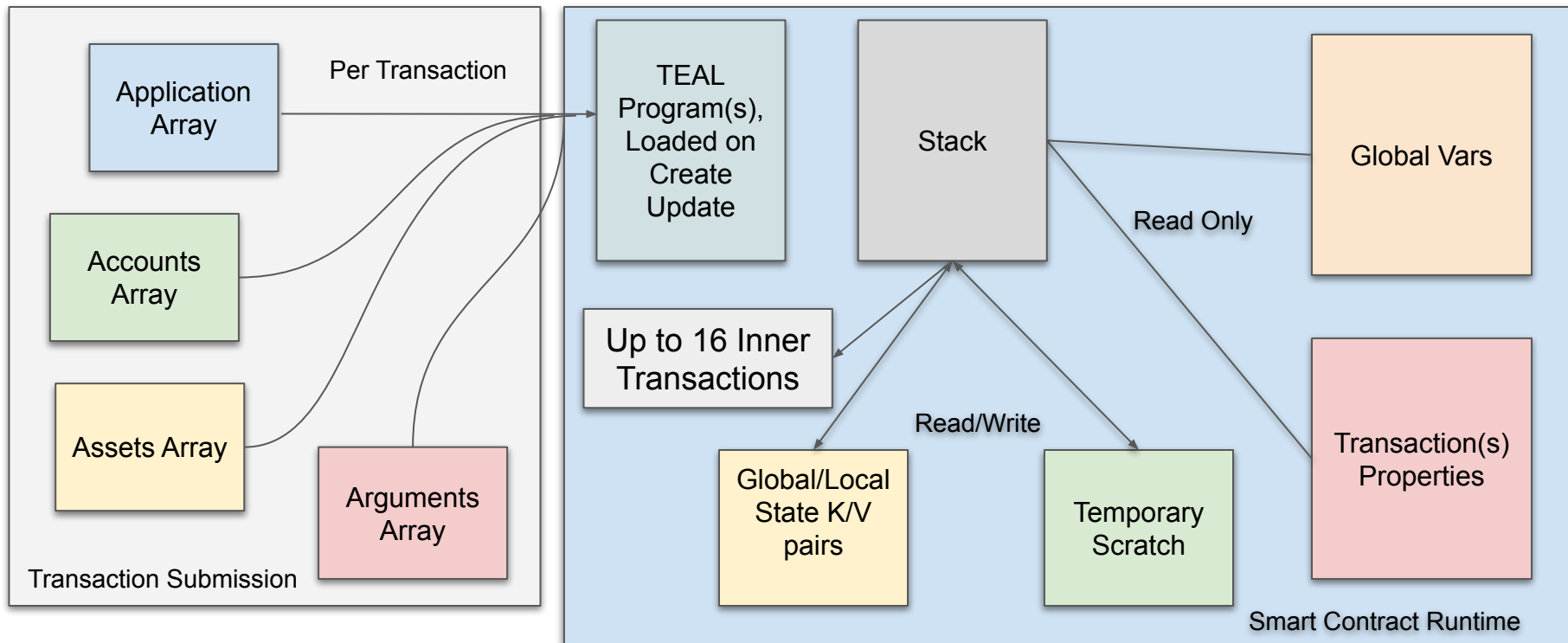
Global State  
Fund Start Date  
Fund End Date  
Fund Goal  
Receiver Address  
Fund Close Date  
Smart Contract Holds Funds

User Storage  
Giving Amount



# Runtime Architecture

# Smart Contract Runtime Architecture



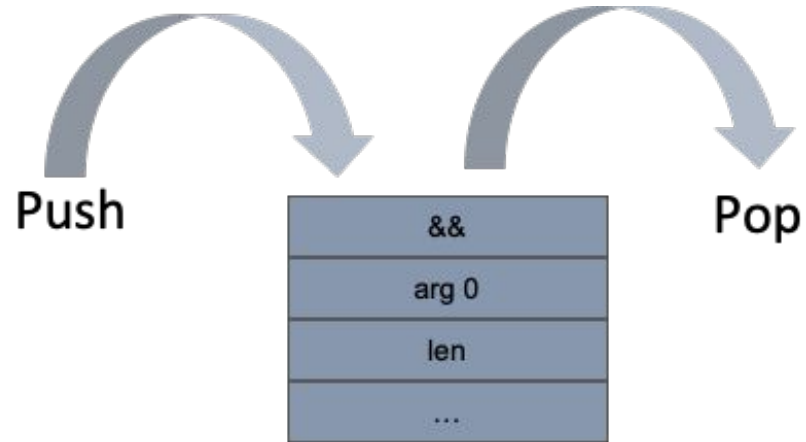


# Transaction Execution Approval Language (TEAL)

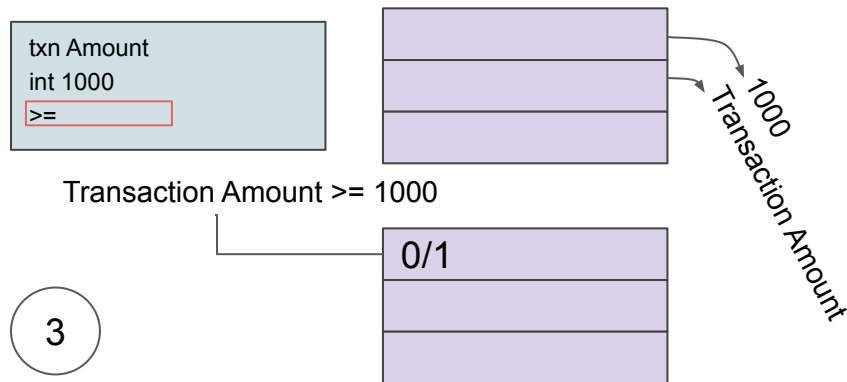
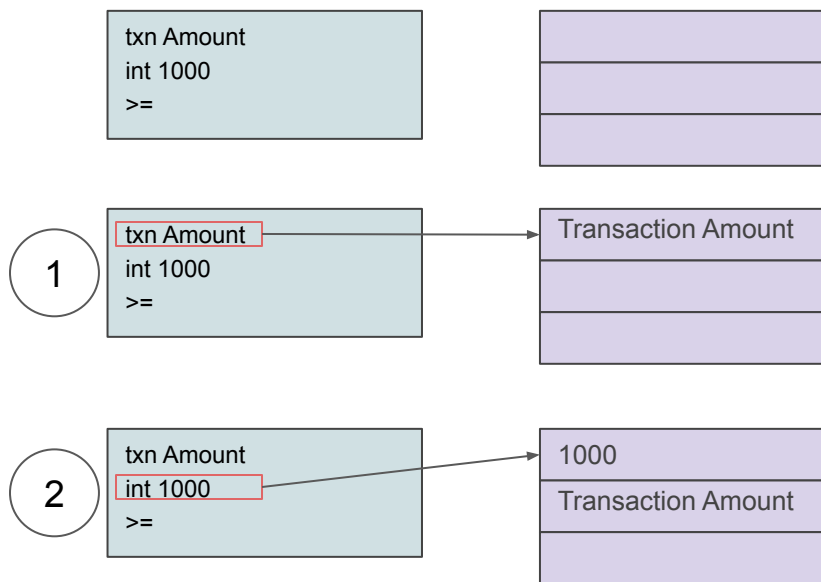


# TEAL - Transaction Execution Approval Language

- Bytecode based stack language
- Turing Complete
- Looping and Subroutines
- True or False - Positive value on stack
- > 130 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](#)

# Pseudo Operators

**int** //load an int onto stack

**byte** //Load bytes on stack

**addr** //Load Algorand address

## TEAL Approval Program

txn Receiver

addr HJLWACXDBOEH25KJB2WI2X5BHQOJ4LS2MRLNMHJ5ZZOBLJU7KGDJSHEU3I

==

.

byte "mystring"

txn ApplicationArgs 0

==



## State Demo

# Accessing Transaction Properties

- Sender
- Fee
- FirstValid
- FirstValidTime
- LastValid
- Note
- Lease
- Receiver
- Amount
- CloseRemainderTo
- VotePK
- SelectionPK
- VoteFirst
- VoteLast
- VoteKeyDilution
- Type
- TypeEnum
- XferAsset
- AssetAmount
- AssetSender
- AssetReceiver
- AssetCloseTo
- GroupIndex
- TxID
- ...

TEAL Approval  
Program

```
txn Amount  
int 1000  
>=
```

**\*\*over 60  
properties**

# Global Variables

Index	Name	Type	Notes
0	MinTxnFee	uint64	micro Algos
1	MinBalance	uint64	micro Algos
2	MaxTxnLife	uint64	rounds
3	ZeroAddress	[]byte	32 byte address of all zero bytes
4	GroupSize	uint64	Number of transactions in this atomic transaction group. At least 1
5	LogicSigVersion	uint64	Maximum supported TEAL version. LogicSigVersion $\geq$ 2.
6	Round	uint64	Current round number. LogicSigVersion $\geq$ 2.
7	LatestTimestamp	uint64	Last confirmed block UNIX timestamp. Fails if negative. LogicSigVersion $\geq$ 2.
8	CurrentApplicationID	uint64	ID of current application executing. Fails in LogicSigs. LogicSigVersion $\geq$ 2.
9	CreatorAddress	[]byte	Address of the creator of the current application. Fails if no such application is executing. LogicSigVersion $\geq$ 3.
10	CurrentApplicationAddress	[]byte	Address that the current application controls. Fails in LogicSigs. LogicSigVersion $\geq$ 5.
11	GroupID	[]byte	ID of the transaction group. 32 zero bytes if the transaction is not part of a group. LogicSigVersion $\geq$ 5.

# Checking Type of Transaction

Value	Constant name	Description
0	unknown	Unknown type. Invalid transaction
1	pay	Payment
2	keyreg	KeyRegistration
3	acfg	AssetConfig
4	axfer	AssetTransfer
5	afrz	AssetFreeze
6	appl	ApplicationCall

TEAL Approval  
Program

```
txn.TypeEnum  
int appl  
==
```



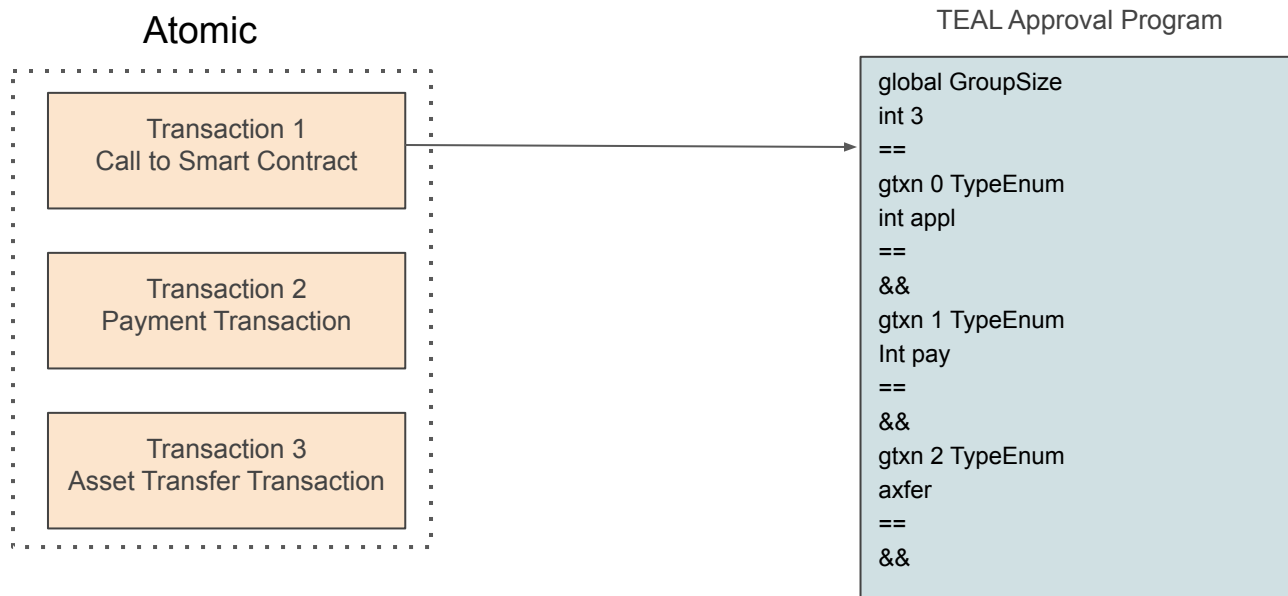
# Application Transaction Sub-Types - Stateful

0	NoOp	Only execute the <code>ApprovalProgram</code> associated with this application ID, with no additional effects.
1	OptIn	Before executing the <code>ApprovalProgram</code> , allocate local state for this application into the sender's account data.
2	CloseOut	After executing the <code>ApprovalProgram</code> , clear any local state for this application out of the sender's account data.
3	ClearState	Don't execute the <code>ApprovalProgram</code> , and instead execute the <code>ClearStateProgram</code> (which may not reject this transaction). Additionally, clear any local state for this application out of the sender's account data as in <code>CloseOutOC</code> .
4	UpdateApplication	After executing the <code>ApprovalProgram</code> , replace the <code>ApprovalProgram</code> and <code>ClearStateProgram</code> associated with this application ID with the programs specified in this transaction.
5	DeleteApplication	After executing the <code>ApprovalProgram</code> , delete the application parameters from the account data of the application's creator.

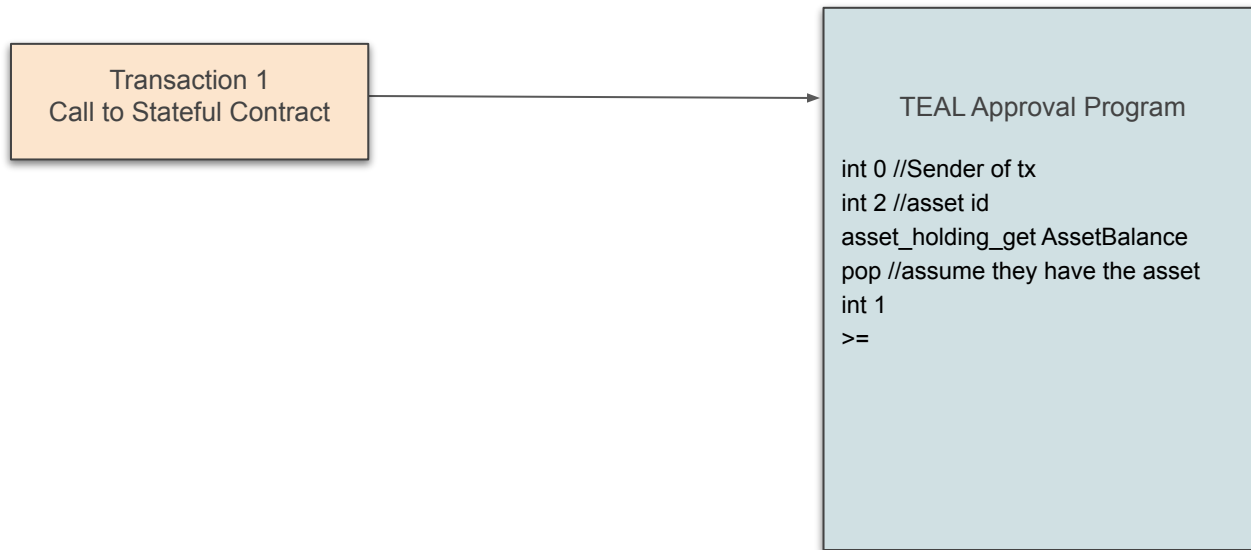
TEAL Approval  
Program

```
txn OnCompletion  
int NoOp  
==
```

# Atomic Transactions - gtxn vs txn

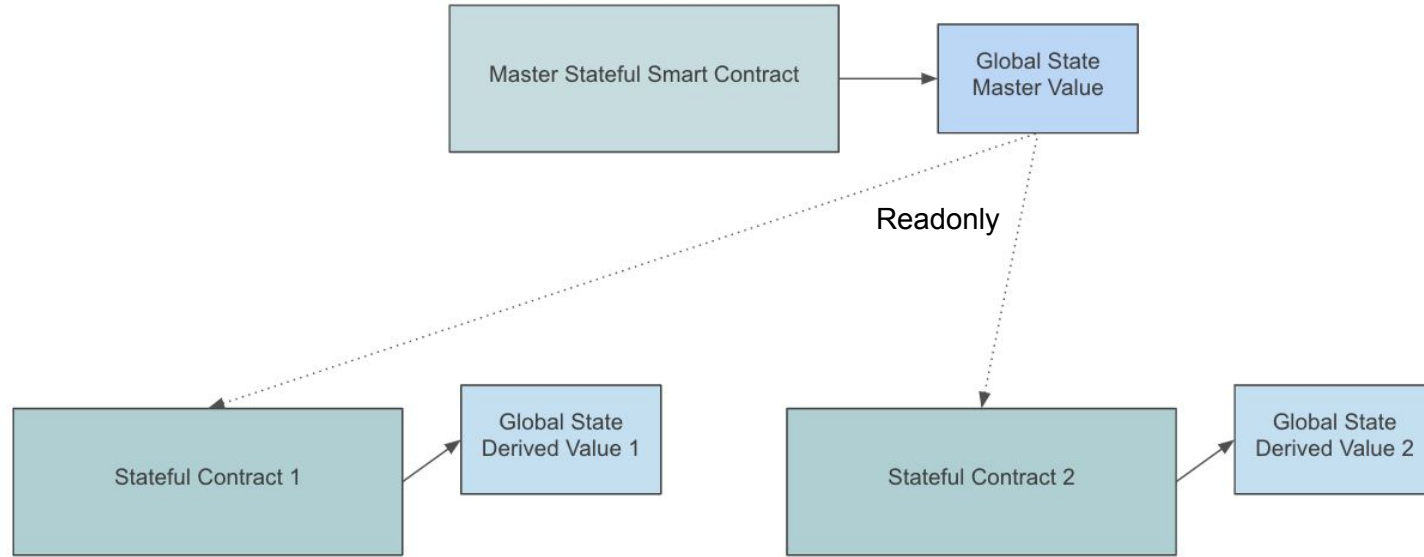


# Asset Check



**\*\*Asset and  
Account have to  
be in Assets and  
Accounts array**

# Read Global State From Another Contract



**\*\*External  
Application has to  
be in Applications  
array**

# Loops and Subroutines

- **Constraint -opcode budget**
  - 700 opcode per call
- **Constraint - file size**
  - 2k - 8k for total for both programs

```
main:
int 0 // initialize loop
loop:
callsub cal_mul
int 1
+ // increment by 1
dup
txn ApplicationArgs 0
btoi // it must be an integer
int 1
-
< // loop until app argument -1
bnz loop
```

```
cal_mul:
load 0
int 1
-
load 1
*
store 1
load 0
int 1
-
store 0
retsub
```



# Looping and Subroutine Demo

# Inner Transactions

- Up to 16 inner transactions
- Shows up as inner transactions within application transaction
- Recipient must be in the accounts array
- itxn opcodes used to initiate

```
handle_noop:  
txn ApplicationArgs 0  
byte "payme"  
==  
assert  
itxn_begin  
int pay  
itxn_field TypeEnum  
int 5000  
itxn_field Amount  
txn Sender  
itxn_field Receiver  
itxn_submit
```

Algorand

PyTeal



# PyTEAL - Python Library

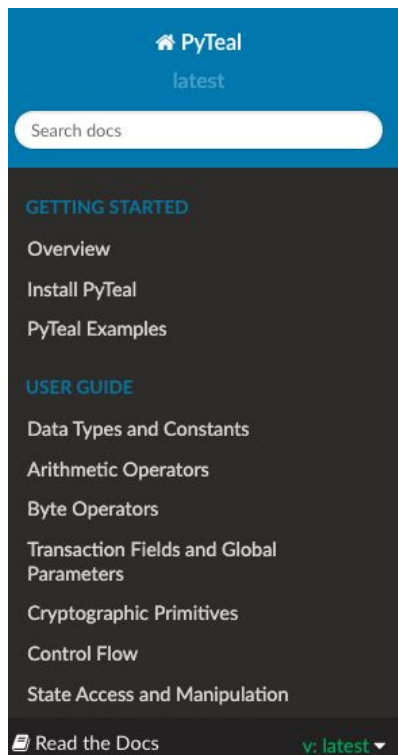
```
def bank_for_account(receiver):  
    is_payment = Txn.type_enum() == TxnType.Payment  
    is_single_tx = Global.group_size() == Int(1)  
    is_correct_receiver = Txn.receiver() == Addr(receiver)  
  
    return And(  
        is_payment,  
        is_single_tx,  
        is_correct_receiver  
    )  
  
if __name__ == "__main__":  
    program =  
    bank_for_account("ZZAF5ARA4MEC5PVDOP64JM5O5MQST63Q2KOY2FLYFL  
    XXD3PFSNJJBIAFZM")  
    print(compileTeal(program, Mode.Signature))
```

Returns



```
#pragma version 2  
txn TypeEnum  
int pay  
==  
global GroupSize  
int 1  
==  
&&  
txn Receiver  
addr  
ZZAF5ARA4MEC5PVDOP64JM5O5MQST63Q2KOY2FLYFLXXD3P  
FSNJJBIAFZM  
==  
&&
```

# PyTeal Documentation



[Docs](#) » [PyTeal: Algorand Smart Contracts in Python](#)

[Edit on GitHub](#)

## PyTeal: Algorand Smart Contracts in Python

PyTeal is a Python language binding for [Algorand Smart Contracts \(ASC1s\)](#).

Algorand Smart Contracts are implemented using a new language that is stack-based, called [Transaction Execution Approval Language \(TEAL\)](#). This is a non-Turing complete language that allows branch forwards but prevents recursive logic to maximize safety and performance.

However, TEAL is essentially an assembly language. With PyTeal, developers can express smart contract logic purely using Python. PyTeal provides high level, functional programming style abstractions over TEAL and does type checking at construction time.

The [User Guide](#) describes many useful features in PyTeal, and the complete documentation for every expression and operation can be found in the [PyTeal Package API documentation](#).

PyTeal hasn't been security audited. Use it at your own risk.

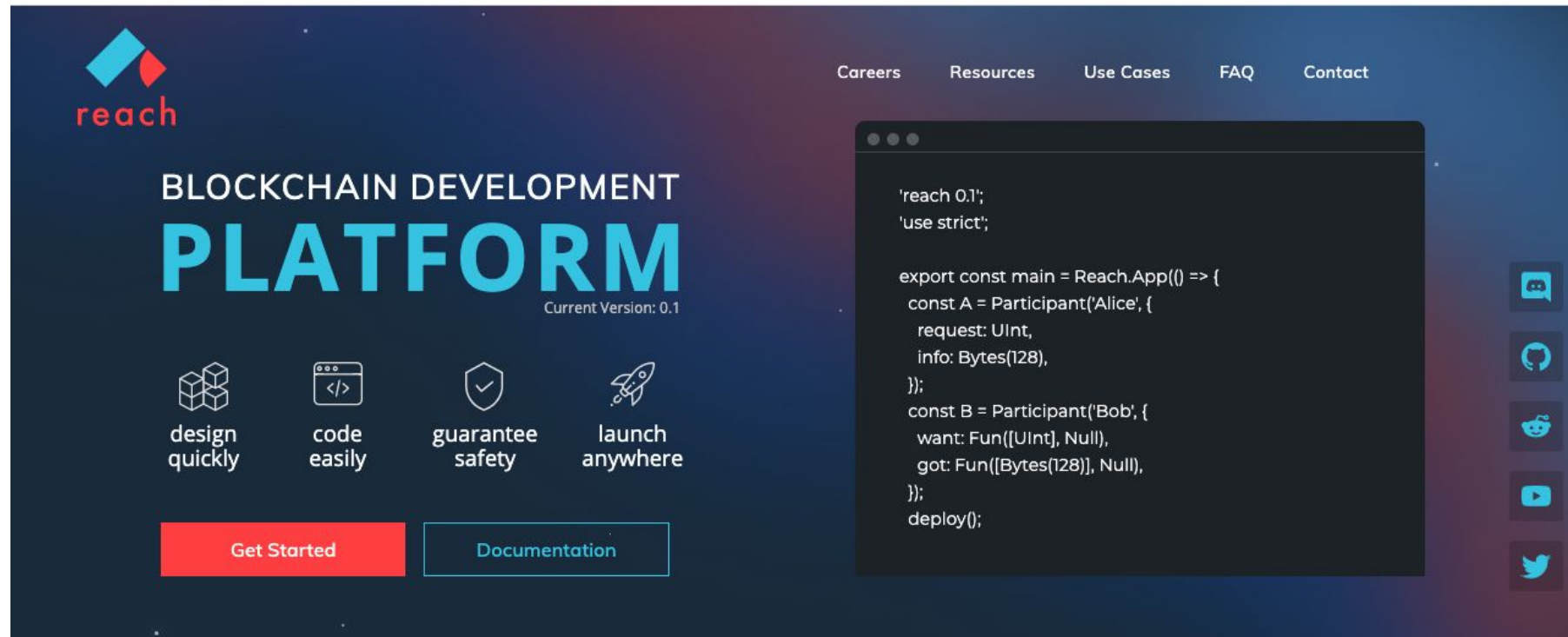
### Getting Started

[Documentation](#)



## PyTeal Demo

# Reach - Reach.sh



The screenshot shows the Reach.sh website homepage. The background is a dark blue gradient. In the top left is the Reach logo, which consists of a stylized 'R' made of two overlapping triangles (one blue, one red) and the word 'reach' in red lowercase letters. To the right of the logo is a navigation bar with links: 'Careers', 'Resources', 'Use Cases', 'FAQ', and 'Contact'. Below the logo, the text 'BLOCKCHAIN DEVELOPMENT' is in white, and 'PLATFORM' is in large, bold, light blue letters. Underneath 'PLATFORM' is the text 'Current Version: 0.1'. Below this are four icons in a row: a stack of cubes, a code editor window, a shield with a checkmark, and a rocket. Each icon has text below it: 'design quickly', 'code easily', 'guarantee safety', and 'launch anywhere'. At the bottom are two buttons: a red 'Get Started' button and a white 'Documentation' button with a blue border. On the right side of the page is a dark grey code editor window showing Reach code. To the right of the code editor are five social media icons: a chat bubble, GitHub, Reddit, YouTube, and Twitter.

**reach**

BLOCKCHAIN DEVELOPMENT  
**PLATFORM**  
Current Version: 0.1

design quickly   code easily   guarantee safety   launch anywhere

[Get Started](#)   [Documentation](#)

```
'reach 0.1';
'use strict';

export const main = Reach.App(() => {
  const A = Participant('Alice', {
    request: UInt,
    info: Bytes(128),
  });
  const B = Participant('Bob', {
    want: Fun([UInt], Null),
    got: Fun([Bytes(128)], Null),
  });
  deploy();
});
```

Chat, GitHub, Reddit, YouTube, Twitter

# Link To Presentation

- <https://github.com/algorand/smart-contracts/tree/master/devrel/www>

# 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>