



# Algorand AVM 1.0 Smart Contracts

Sept 27, 2021

# Table of Contents

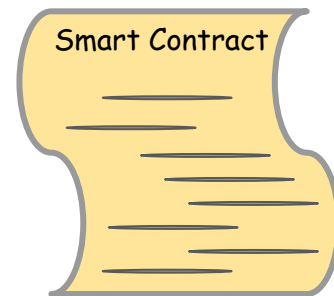
- Brief AVM 1.0 Features
- Overview of Smart Contracts and Smart Signatures
- Transaction Execution Approval Language (TEAL)
- Developing Smart Contracts with Python
- Reach Applications

# AVM 1.0 Release Features

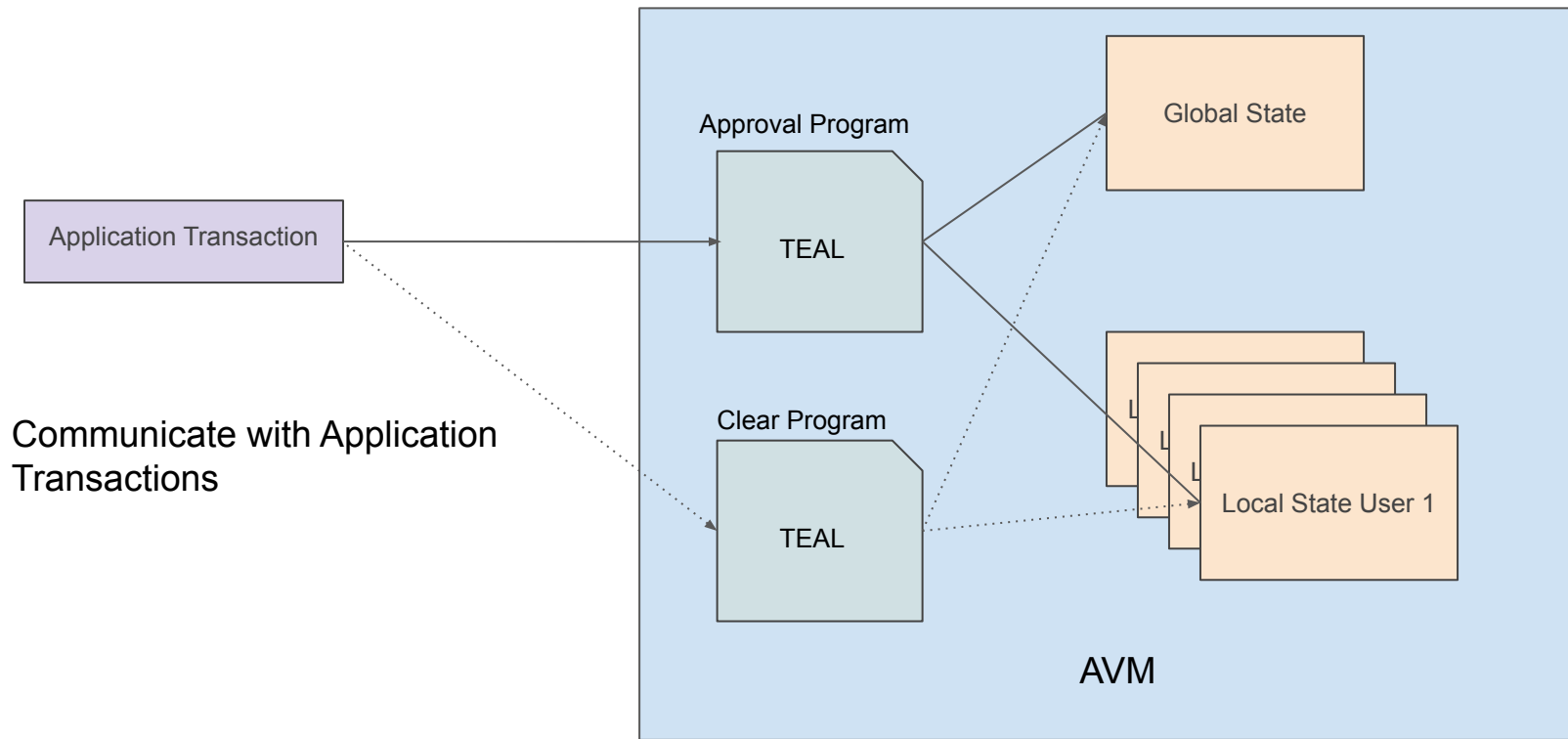
- 50 Smart Contract Optins (10 create)
- Pool Smart Contract Budget
- Inner Transactions
- Over 15 new opcodes - including Log support
- ABI spec with SDK support - In a couple of weeks
- Revamped Developer Site Launch This week.

# Algorand Smart Contracts and Smart Signatures

- **Smart Contracts and Smart Signatures**
  - Smart Contracts - On-chain Global and Local Storage
    - Escrow Accounts
    - On-chain dApps
  - Smart Signatures - Used to sign transactions
    - Delegate
- **Combinable with Other Algorand Technology**
  - Atomic Transfers
  - Algorand Assets
  - Combine Smart Contracts with Smart Signatures
- **Transaction Execution Approval Language**
  - The contract logic is written in TEAL
  - Python Enabled Compiler (PyTEAL)

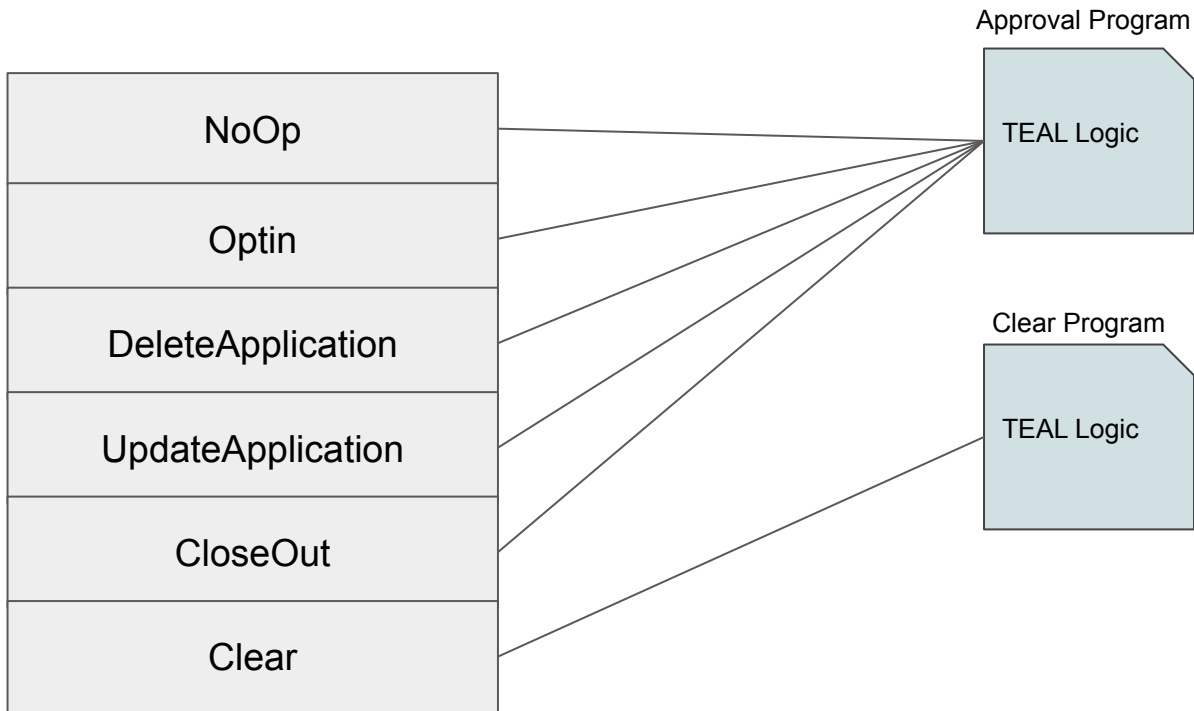


# Smart Contracts aka Apps



# Transaction Sub-Types for Application

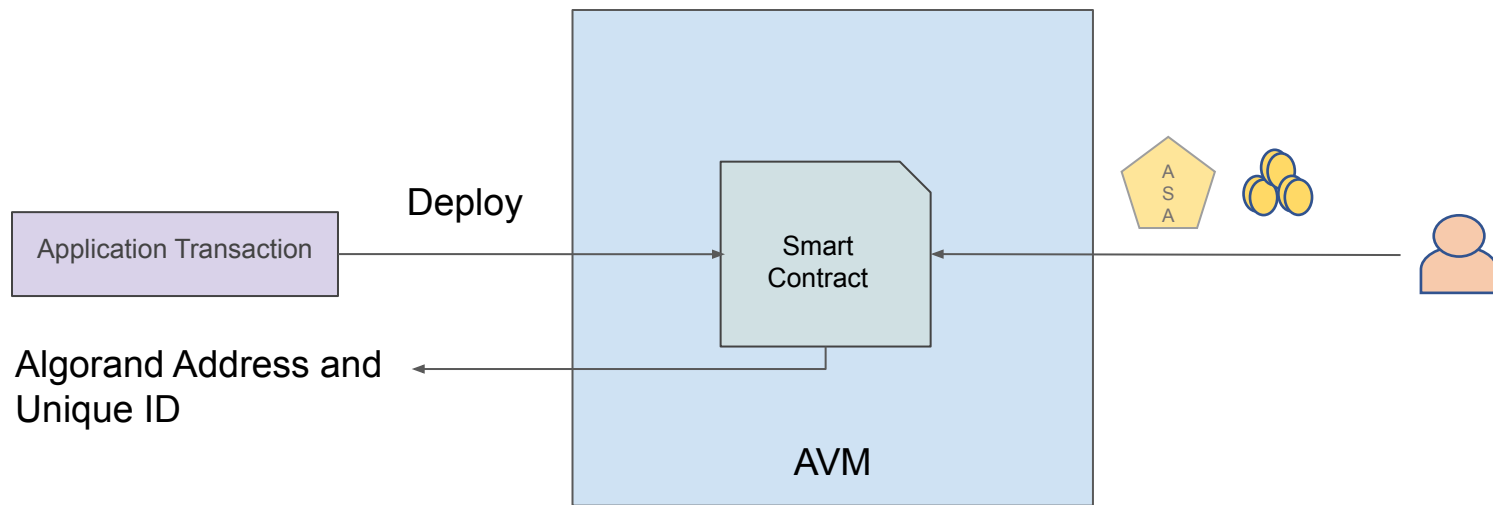
Used to Communicate With Smart Contract



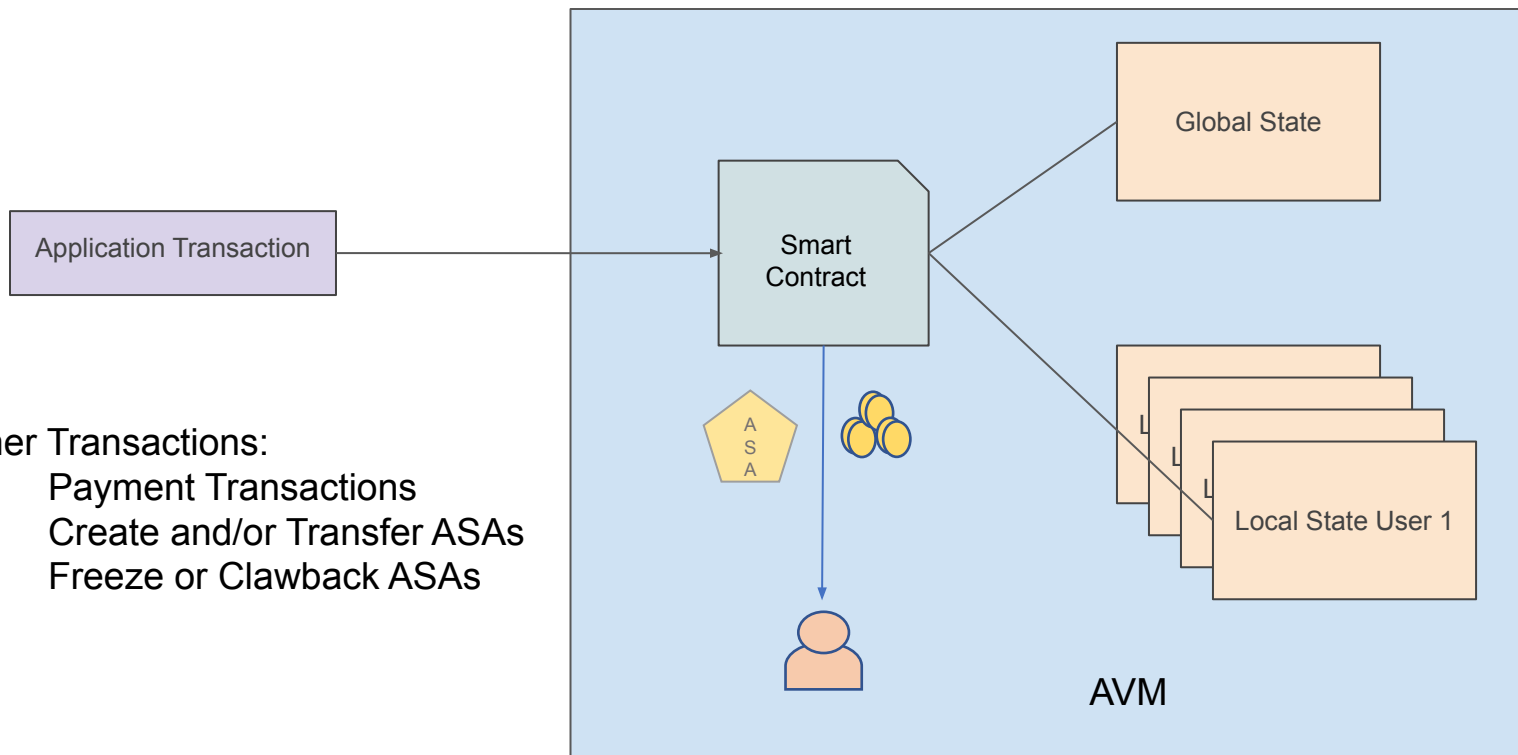
Graceful

Non-Graceful  
ie Will clear  
regardless

# Smart Contract Escrow



# Smart Contract Escrow



## Inner Transactions:

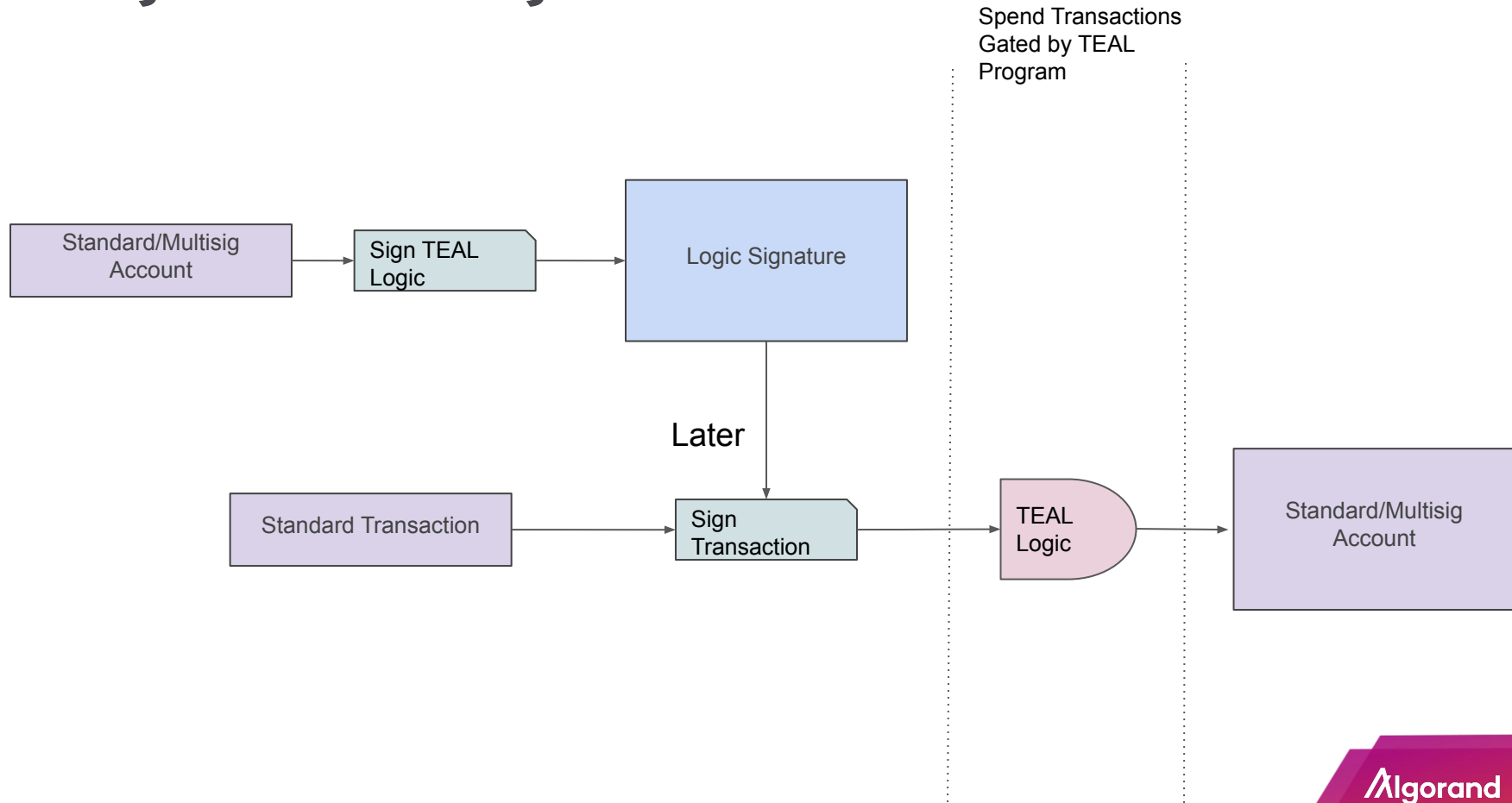
- Payment Transactions
- Create and/or Transfer ASAs
- Freeze or Clawback ASAs



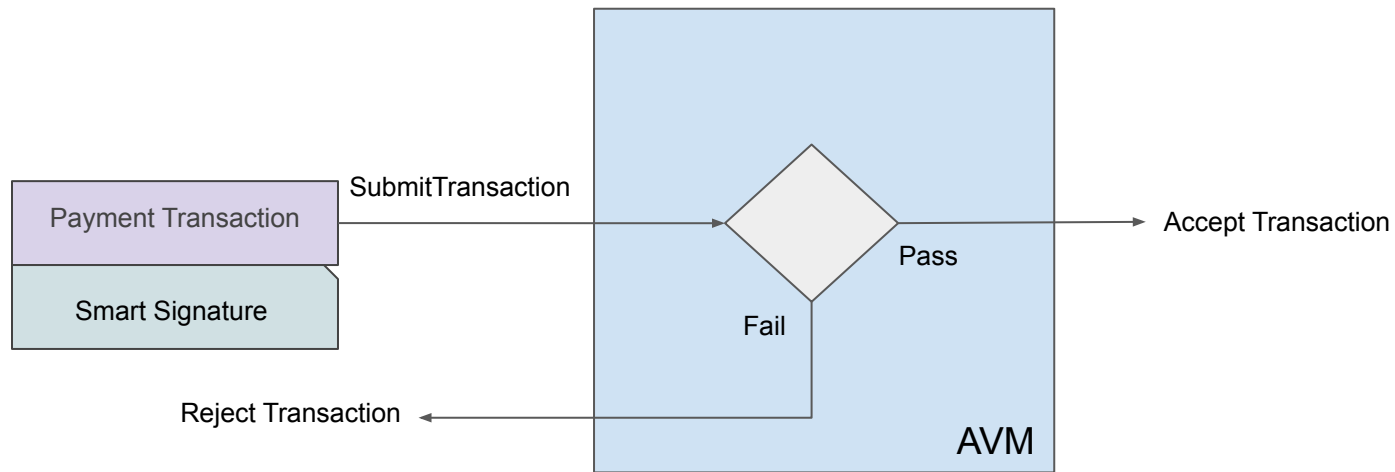
# Stateful Contract Usages

- Escrows
- 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

# Delegate Smart Signature



# Smart Signature



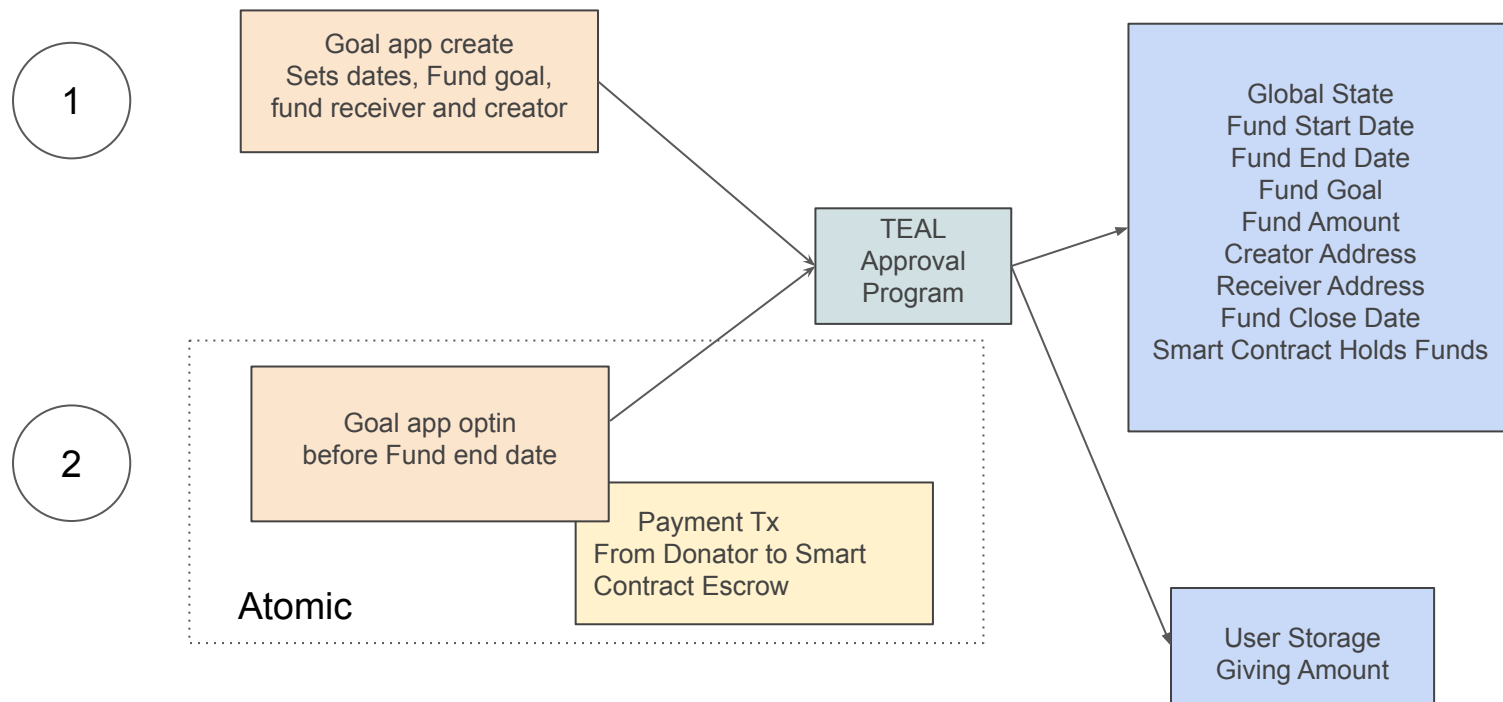
# Smart Signature Usages

- **Delegated**
  - Recurring Payments
  - Limited Account Authority



## Combining Algorand Technologies

# Combining Technologies - Crowdfunding



# Payout

3

Goal app call  
receive Funds if Fund End  
date has passed and fund  
goal achieved

4

Goal app 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  
Fund Amount  
Escrow Address  
Creator Address  
Receiver Address  
Fund Close Date

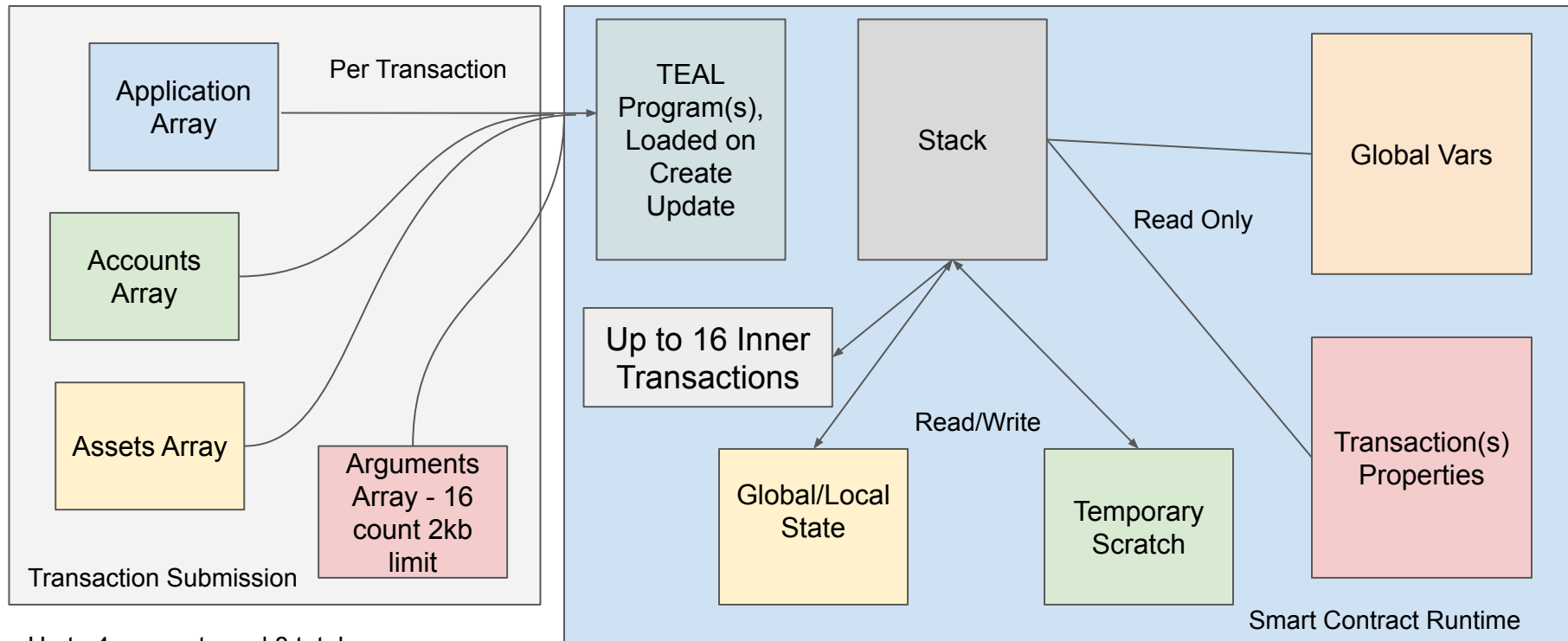
User Storage  
Giving Amount



# Runtime Architecture



# Smart Contract Runtime Architecture

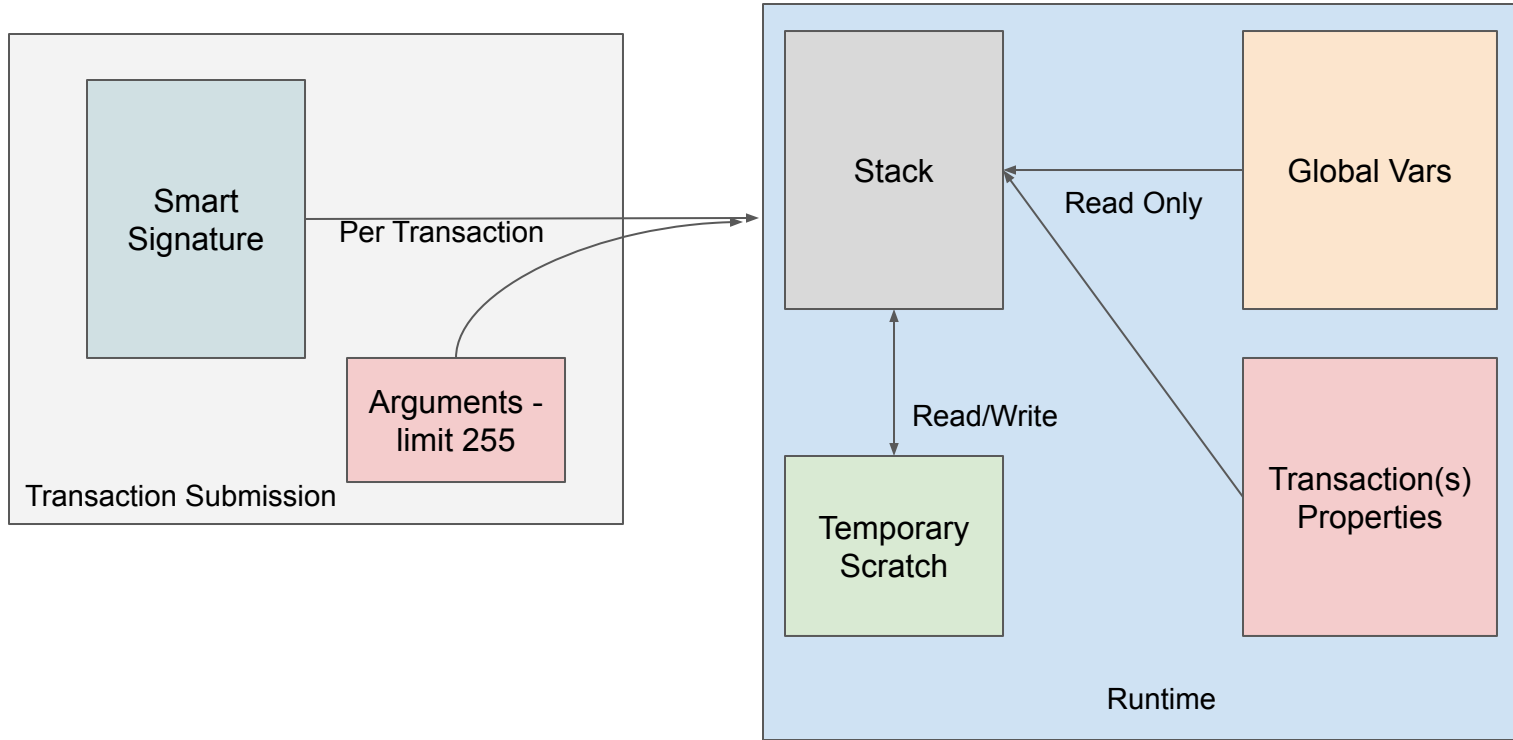


Up to 4 accounts and 8 total  
for Accounts, Application,  
Asset Arrays

Global State - 64 KV pairs -  
128 total bytes per KV, key  
limited to 64 bytes

Local State - 16 KV pairs -  
128 total bytes per KV, key  
limited to 64 bytes

# Smart Signature Runtime Architecture

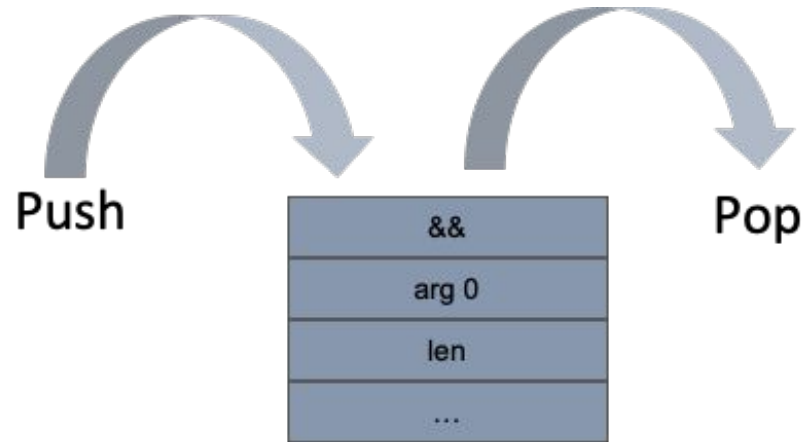




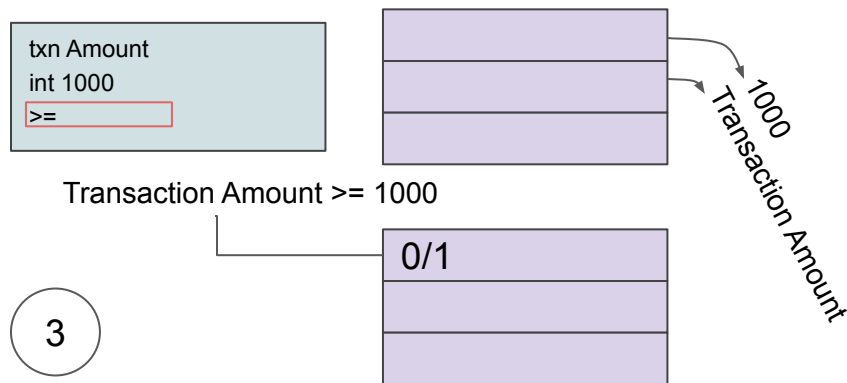
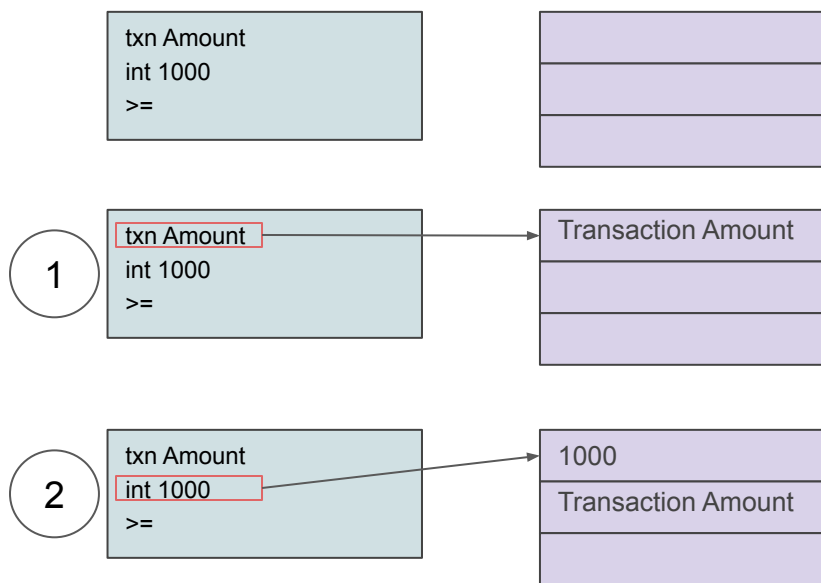
# Transaction Execution Approval Language (TEAL)

# TEAL - Transaction Execution Approval Language

- Bytecode based stack language
- Turing Complete
- Looping and Subroutines
- Returns True or False
- SDK Support
- > 130 Opcodes
- Access to ASA/Algo Balances
- Read all Transactions in a Group
- Smart Contract - Global/Local
- Smart Contract - 16 inner transactions
  - Asset transfers, creation, deletion
  - Asset configuration, revoking and freezing
  - Payment transfers
- 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

## app\_local\_get

- Opcode: 0x62
- Pops: ... *stack*, {uint64 A}, {[[]byte B}
- Pushes: any
- read from account specified by Txn.Accounts[A] from local state of the current application key B  
=> value
- LogicSigVersion >= 2
- Mode: Application

params: account index, state key. Return: value. The value is zero if the key does not exist.

[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





## Branching Demo - b, bnz, bz

# 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  
==
```

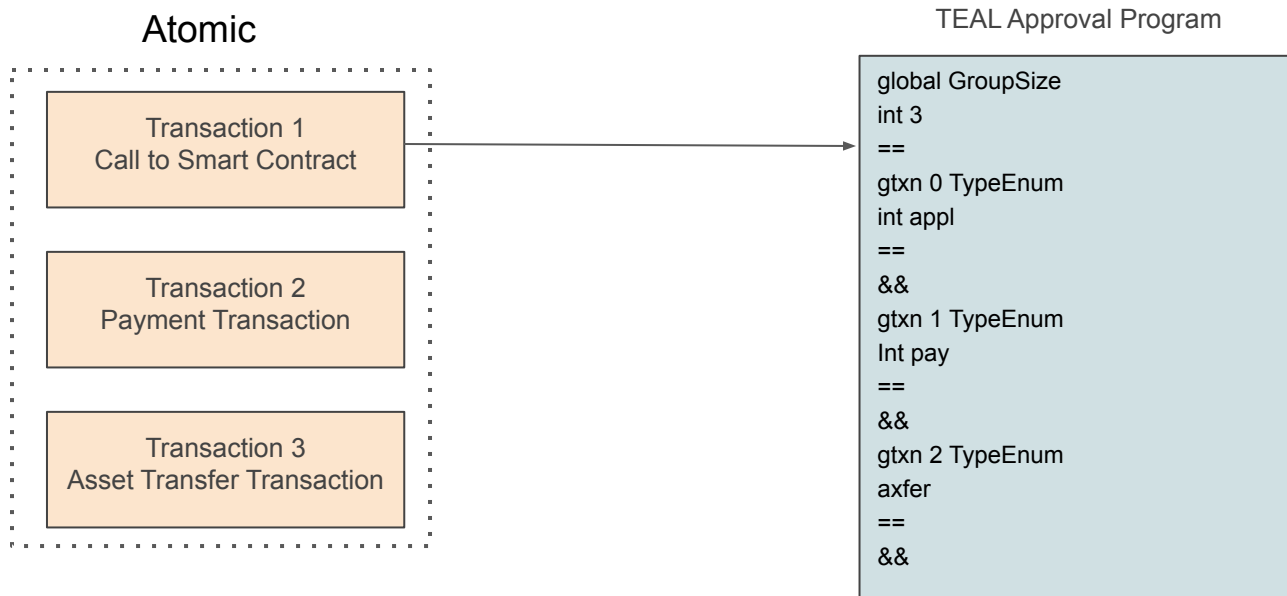
# Application Transaction Sub-Types - Stateful

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





## Atomic Demo

# Asset Check

Transaction 1  
Call to Stateful Contract

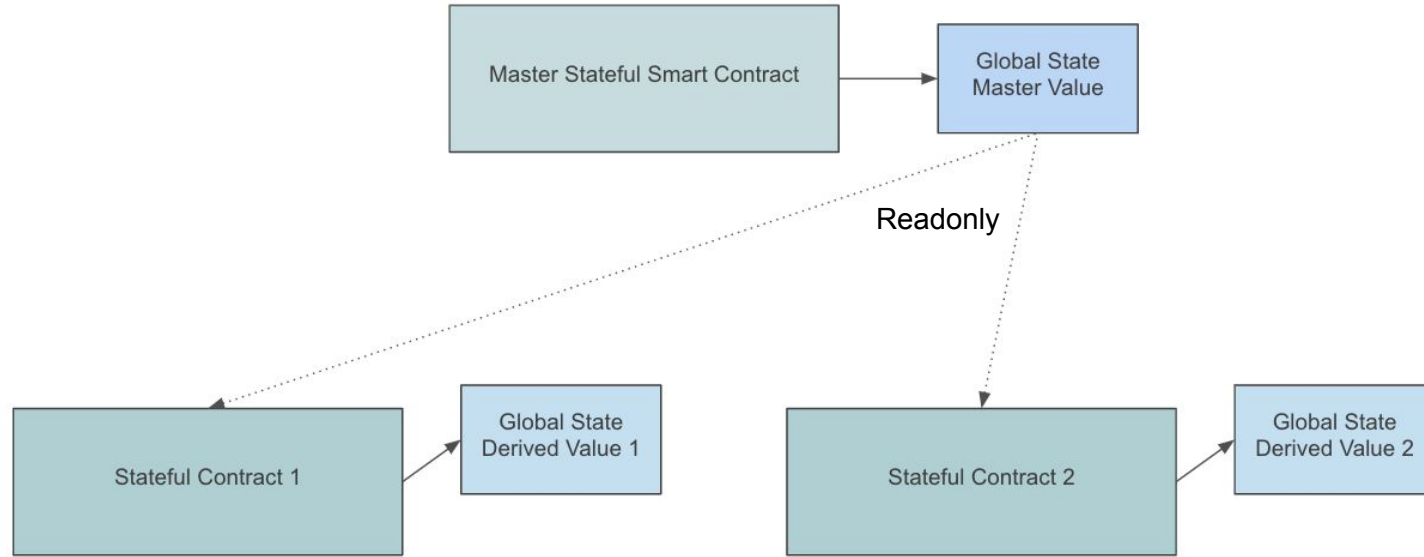
TEAL Approval Program

```
int 0 //Sender of tx  
int 2 //asset id  
asset_holding_get AssetBalance  
pop //assume they have the asset  
int 1  
>=
```

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

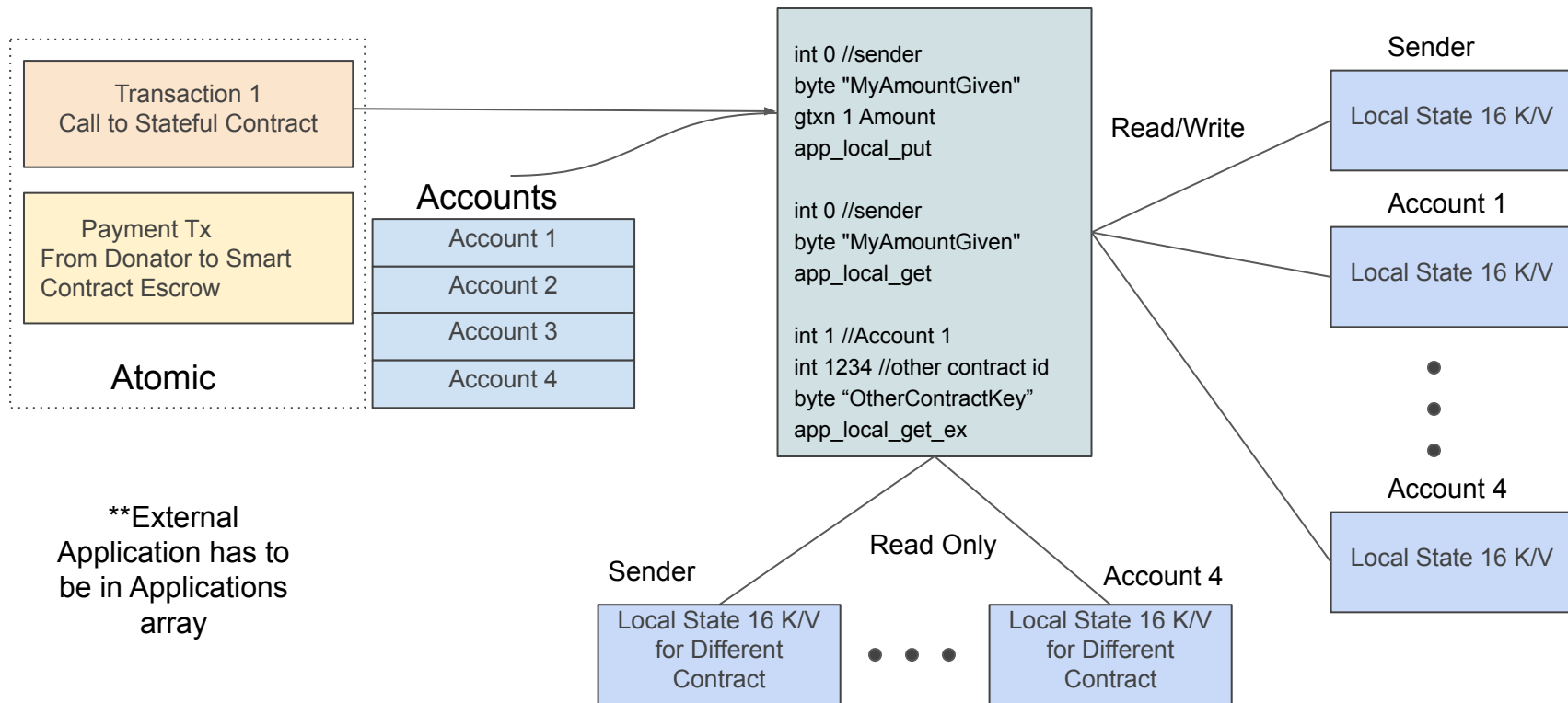


# Read Global State From Another Contract



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

# Reading and Writing Local State



# Loops and Subroutines

- **Constraint -opcode budget**
  - 700 opcode per call
  - Can be pooled with up to 16 atomic application transactions
- **Constraint - file size**
  - 2k - 8k for total for both programs
  - Upping pages increase minimum balance

```
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
- **Smart contract pays fees**
  - Fees eligible for pooling
  - Fees are default min balance
- Shows up as inner transactions within application transaction
- Recipient must be in the accounts array
- Accounts can be rekeyed to smart contract giving the spending authority to the contract but account must be in accounts array to spend from it
- Supports all asset transactions
- Supports payment transactions

```
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
```



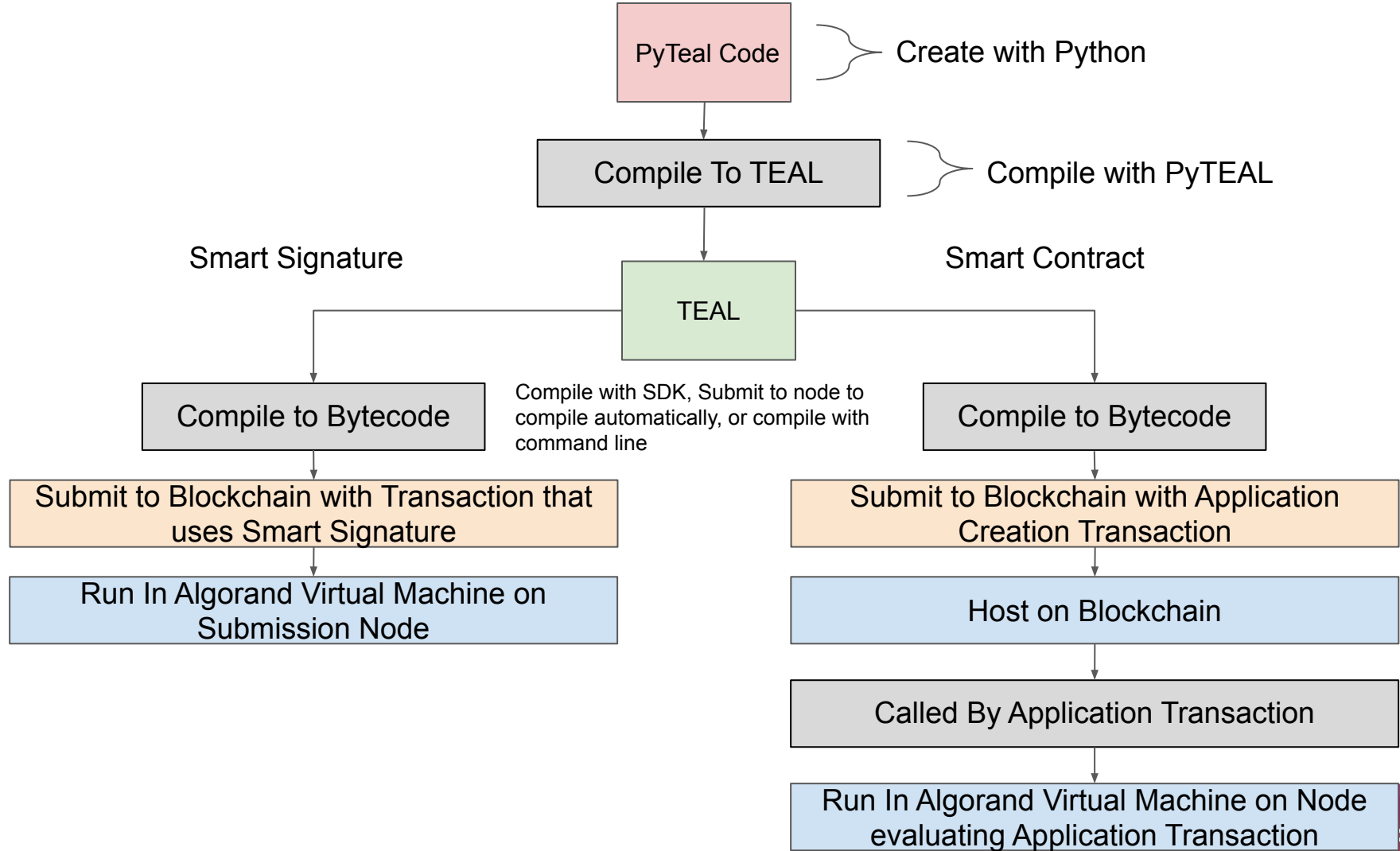
# Inner Transaction Demo

# 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))
```

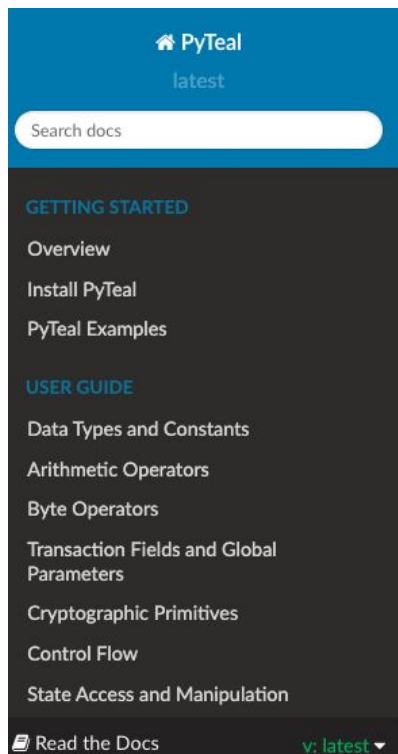
Returns

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





# PyTeal Documentation



The sidebar of the PyTeal documentation website. It features a blue header with the 'PyTeal' logo and a 'latest' version indicator. Below the header is a search bar labeled 'Search docs'. The main menu is divided into two sections: 'GETTING STARTED' and 'USER GUIDE'. Under 'GETTING STARTED', there are links for 'Overview', 'Install PyTeal', and 'PyTeal Examples'. Under 'USER GUIDE', there are links for 'Data Types and Constants', 'Arithmetic Operators', 'Byte Operators', 'Transaction Fields and Global Parameters', 'Cryptographic Primitives', 'Control Flow', and 'State Access and Manipulation'. At the bottom of the sidebar, there is a link to 'Read the Docs' and a version selector showing 'v: latest' with a dropdown arrow.

- PyTeal latest
- Search docs
- GETTING STARTED
  - Overview
  - Install PyTeal
  - PyTeal Examples
- USER GUIDE
  - Data Types and Constants
  - Arithmetic Operators
  - Byte Operators
  - Transaction Fields and Global Parameters
  - Cryptographic Primitives
  - Control Flow
  - State Access and Manipulation
- Read the Docs v: latest

[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.

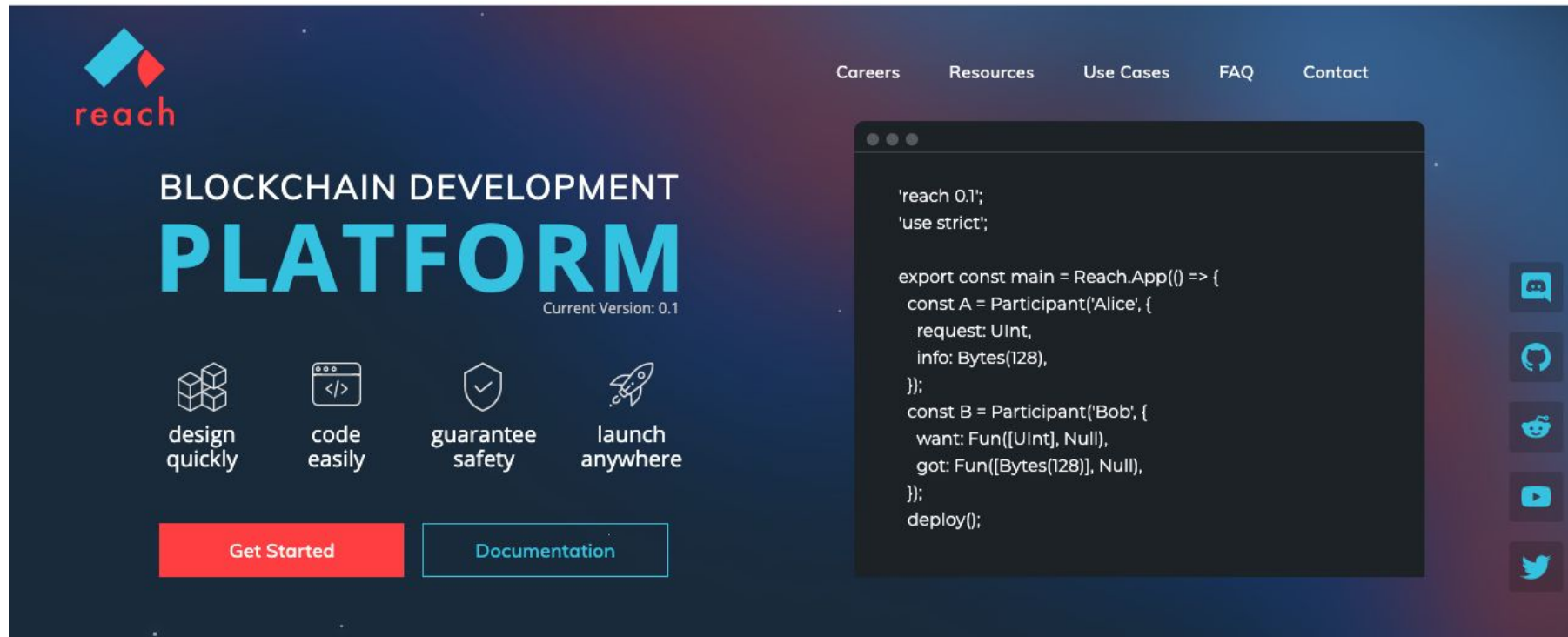
## Getting Started

[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, there 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, there are two buttons: a red 'Get Started' button and a light blue 'Documentation' button. On the right side of the page, there 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

# Reach Documentation

← prev up next →

v.0.1.2

▼ Reach: The Safest and Easiest DApp Programming Language

- 1 Overview
- 2 Tutorial
- 3 Guide
- 4 Workshop
- 5 Reference
- Index

ON THIS PAGE:

Reach: The Safest and Easiest DApp Programming Language

## Reach: The Safest and Easiest DApp Programming Language

by Jay McCarthy <jay@reach.sh>

Reach is a domain-specific language for building decentralized applications ([DApps](#)).

This set of documents contains everything you need to know about Reach.

- [The overview](#) briefly introduces the basic ideas of Reach. It can be helpful to get some context before diving into other sections.
- [The tutorial](#) is a directed series of steps to create a simple [DApp](#). You should start here if you've never used Reach before and want to start at the beginning.
- [The guide](#) continues [the overview](#) by discussing the key ideas and concepts used throughout Reach. You will often want to visit the guide to learn the background and "why" of some topic after you get started using it.

# Resources

- **Discord:** <https://discord.gg/YgPTCVk>
- 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>