

Знайомство зі смарт- контрактами Algorand



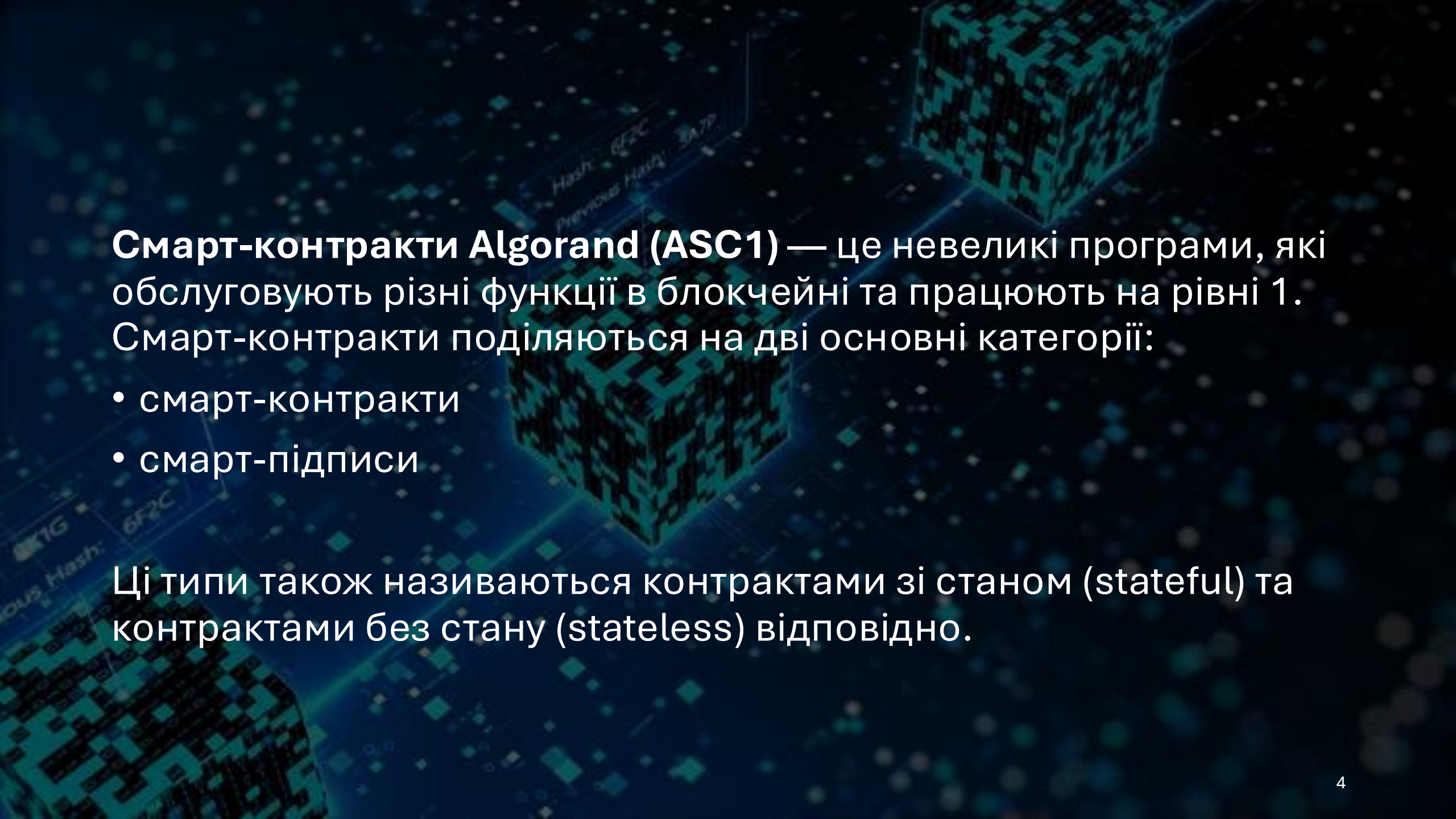
Лекція 2



План:

1. Смарт-контракти в Алгоранд.
Statefull та stateless смарт-контракти.
2. Огляд структури смарт-контрактів Algorand.
3. Огляд мови програмування PyTeal.
4. 'goal' CLI інструмент

Смарт-контракти — це програмні коди, які автоматично виконують угоди між сторонами на основі певних умов. Вони працюють на основі технології блокчейн і забезпечують надійність, прозорість та автоматизацію транзакцій без необхідності довіряти посередникам.




Смарт-контракти Algorand (ASC1) — це невеликі програми, які обслуговують різні функції в блокчейні та працюють на рівні 1. Смарт-контракти поділяються на дві основні категорії:

- смарт-контракти
- смарт-підписи


Ці типи також називаються контрактами зі станом (stateful) та контрактами без стану (stateless) відповідно.

В Algorand існує два основних типи смарт-контрактів:

- **Stateful смарт-контракти:** ці контракти містять логіку, яка зберігається на блокчейні та може бути викликана з будь-якого вузла на блокчейні Algorand. Виклик такого контракту виконується через транзакцію типу `Application Call`. Логіка контракту може модифікувати дані на глобальному рівні або на рівні конкретного акаунта. + ● ○
- **Stateless смарт-контракти (Smart Signatures):** це контракти, які використовуються для підпису транзакцій, наприклад, для делегування підпису. Логіка такого контракту зберігається на блокчейні як частина процесу затвердження транзакції, але вона не викликається віддалено. Кожна нова транзакція, яка використовує такий підпис, повинна знову подати логіку контракту.



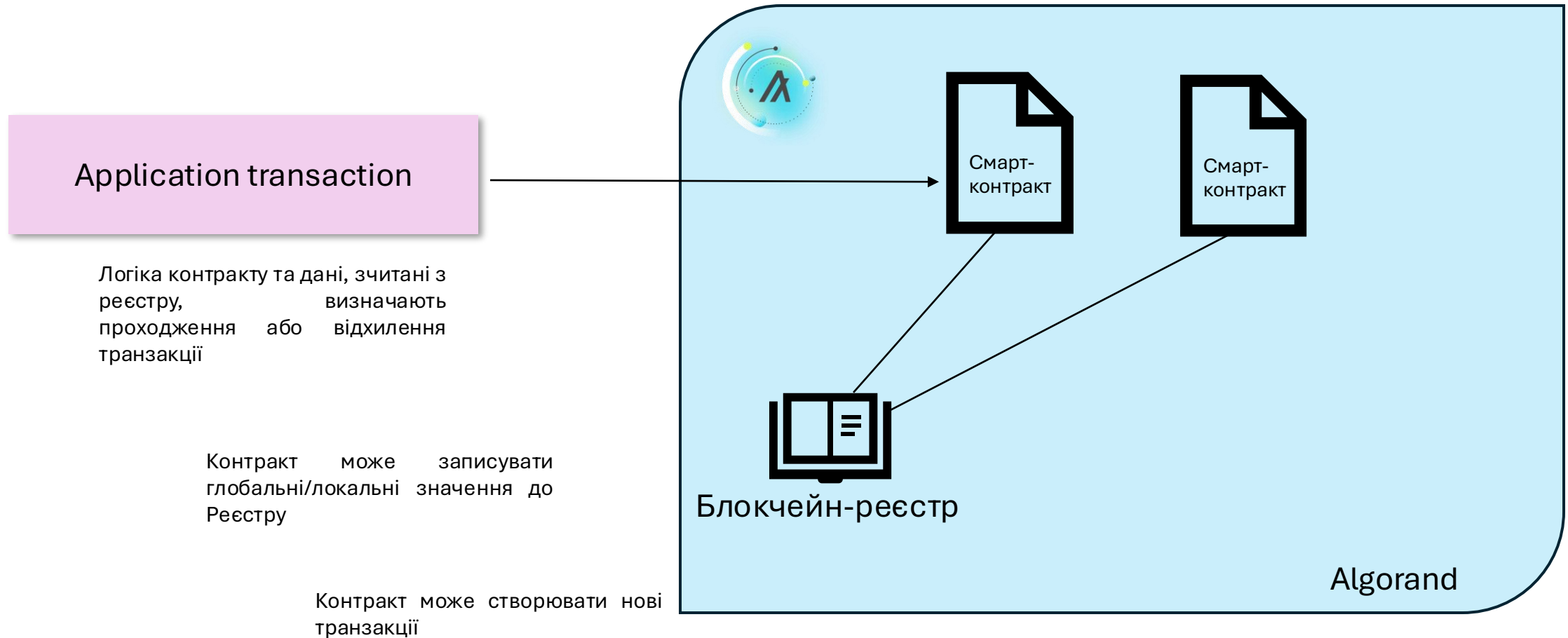
Що таке сма^т- конт^{ра}кти Algorand?



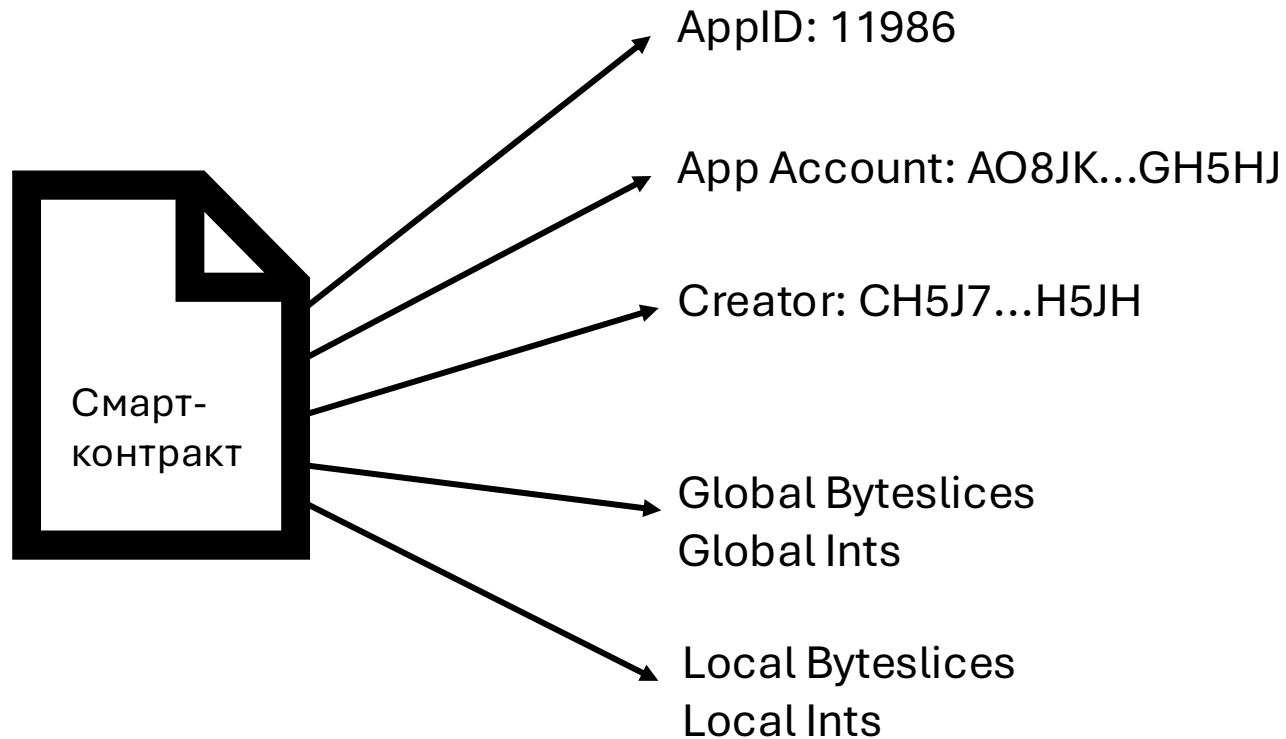
Сма^т-конт^{ра}кт - це

- невеликий фрагмент логіки, який існує на блокчейні,
- ініційований транзакцією,
- повертає значення True або False - підтвердження або невдачу транзакції,
- може генерувати активи та платіжні транзакції,
- може зберігати активи,
- написаний мовою TEAL або PyTeal або згенерований за допомогою фреймворку Reach.

Смарт-контракти. Високий рівень



Смарт-контракт в Алгоранд



Смарт-контракти Algorand

- **Смарт-контракти (stateful)** - це контракти, які після розгортання можна віддалено викликати з будь-якого вузла в блокчейні Algorand. Вони можуть зберігати дані в блокчейні та виконувати операції, які залежать від поточного стану контракту. Контракти з підтримкою стану часто використовуються для більш складних програм, які вимагають постійного зберігання та керування станом.
- Після розгортання екземпляр контракту в ланцюжку називається додатком (Application) і йому надається ідентифікатор додатка (Application Id).
- Ці програми запускаються транзакцією певного типу, яка називається транзакцією виклику програми (Application Call transaction). Ці ончейн-програми обробляють основну децентралізовану логіку dApp.

Смарт-контракти

Ключові характеристики контрактів із станом:


- Смарт-контракти можуть зберігати значення в блокчейні. Це сховище може бути глобальним, локальним або box storage.
- Додатки можуть змінювати стан, пов'язаний із програмою.
- Додатки можуть отримувати доступ до даних у мережі, таких як баланс рахунків, параметри конфігурації активів або час останнього блоку.
- Смарт-контракти можуть виконувати транзакції як частину виконання логіки (внутрішні транзакції).
- Додатки мають пов'язаний обліковий запис програми, який може зберігати баланси Algos або ASA і може використовуватися як рахунки умовного депонування в мережі.

Сховище смарт-контрактів

Глобальне сховище і бокси (boxes) пов'язані з самою програмою, тоді як локальне сховище пов'язано з кожним обліковим записом, який підключився до додатку. Глобальне та локальне сховище — це **пари ключ/значення**, які обмежені **128 байтами на пару**. Boxes — це ключові сегменти зберігання **до 32 Кб даних на бокс**.

Ключі зберігаються у вигляді байтових фрагментів (byte-array value), а *значення* зберігаються у вигляді байтових фрагментів або у вигляді uint64.





Глобальне сховище

- Може включати від 0 до 64 пар ключ/значення (key/value - k/v), загальний обсяг пам'яті яких становить 8 КБ.
- Обсяг глобального сховища виділяється в одиницях k/v і визначається під час створення контракту. Ця схема є незмінною після розгортання смарт-контракту (неможливо змінити кількість глобальних змінних).
- Кількість глобальних змінних впливає на мінімальний баланс для смарт-контракту.
- Може бути прочитано будь-яким викликом програми, яка має ідентифікатор додатку в масиві зовнішніх додатків.
- Значення в глобальне сховище може бути записано лише смарт-контрактом.
- Видаляється, коли видаляється програма, з якою сховище пов'язано.

+

•

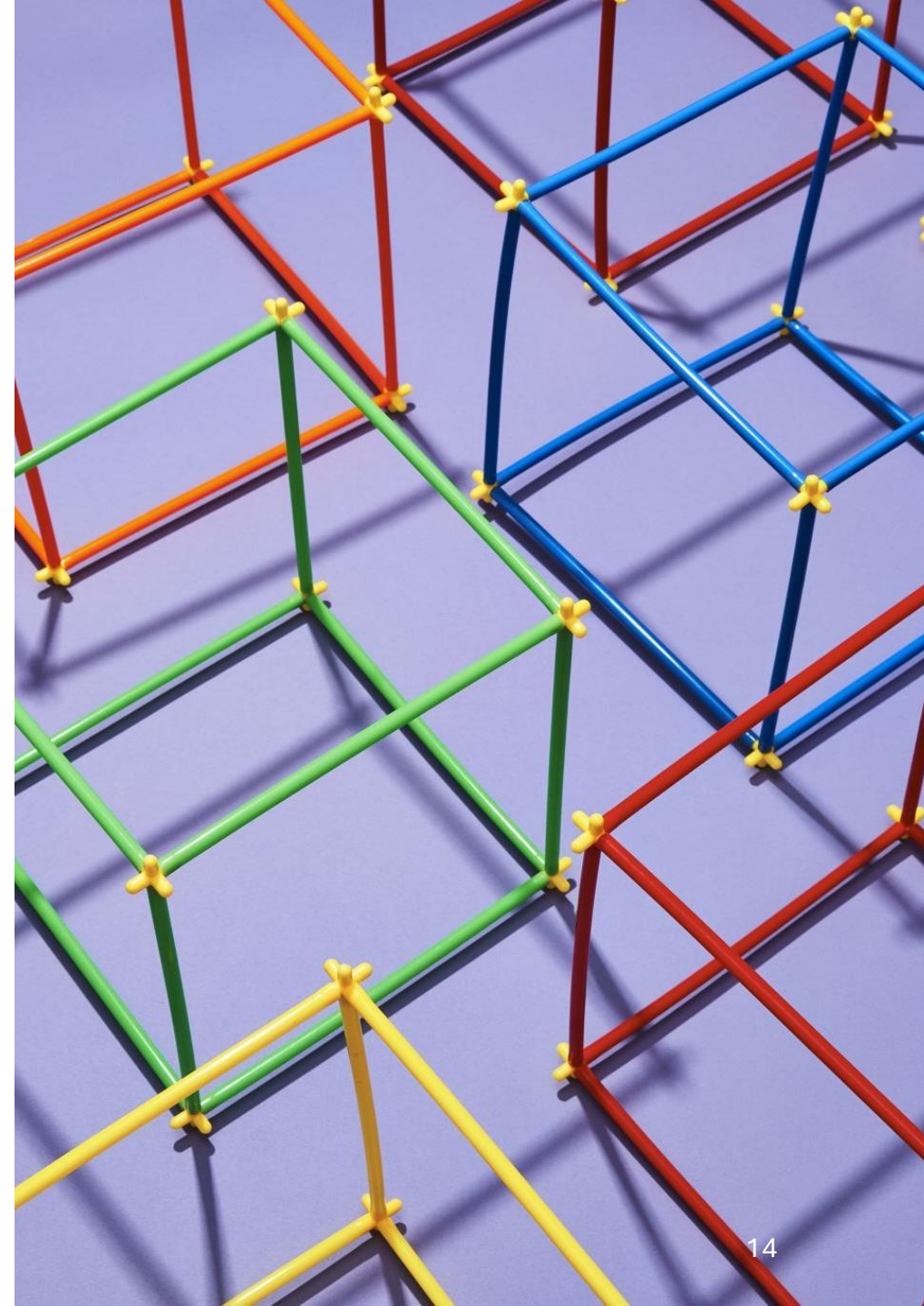
○

Локальне сховище

- Виділяється, коли обліковий запис X підключається до додатку A (надсилає транзакцію для підключення до додатку A).
- Може включати від 0 до 16 пар ключ/значення, загальний обсяг пам'яті яких становить 2 КБ.
- Обсяг локального сховища виділяється в одиницях k/v та визначається при створенні контракту. Це не може бути змінено пізніше.
- Кількість глобальних змінних впливає на мінімальний баланс для смарт-контракту.
- Дані з локального сховища можуть бути прочитані будь-яким викликом програми, яка містить додаток A в масиві зовнішніх додатків і обліковий запис X у масиві зовнішніх облікових записів.
- Редагується лише додатком A, але може бути видалено додатком A або користувачем X (за допомогою виклику ClearState транзакції).
- Обліковий запис X може запросити очищення локального стану за допомогою транзакції закриття (CloseOut).
- Обліковий запис X може очистити свій локальний стан для додатку A за допомогою транзакції очищення стану (ClearState), яка завжди буде успішною, навіть після видалення програми a.

Box Storage

- Додаток А може виділити стільки боксів, скільки йому потрібно в будь-який час.
- Бокси можуть мати будь-який розмір від 0 до 32 Кб.
- Імена боксів мають мати принаймні 1 байт, щонайбільше 64 байти, і мають бути унікальними в додатку А.
- Ім'я бокса та ідентифікатор додатку (appId) повинні бути вказані в масиві боксів додатку, що викликається.
- Якщо додаток видаляється, його бокси не видаляються. Бокси не можна буде змінити, але їх можна буде запитувати за допомогою SDK. Мінімальний баланс також буде заблоковано. (правильний план очищення — знайти бокси поза мережею та викликати додаток, щоб видалити всі його додатки, перш ніж видаляти сам додаток).



Вимоги до мінімального балансу для смарт-контракту¶

Під час створення чи підключення до смарт-контракту мінімальний баланс користувача буде збільшено. Сума, на яку він буде збільшений, залежатиме від обсягу сховища в мережі, яке використовується. Ця вимога до мінімального балансу пов'язана з обліковим записом, який створює смарт-контракт чи підключається до нього.

Це означає, що користувач повинен забезпечити **наявність додаткових Algo на своєму рахунку** для покриття цих витрат.

Збільшення мінімального балансу при створенні смарт-контракту

$$100,000 * (1 + \text{ExtraProgramPages}) + (25,000 + 3,500) * \text{schema.NumUint} + (25,000 + 25,000) * \text{schema.NumByteSlice}$$

100 000 - базова комісія microAlgo за кожен запитаний сторінку.
25 000 + 3 500 = 28 500 для кожного Uint у схемі глобального стану
25 000 + 25 000 = 50 000 для кожного фрагмента байтів у схемі глобального стану

Збільшення мінімального балансу при приєднанні до смарт-контракту

$$100,000 + (25,000 + 3,500) * \text{schema.NumUint} + (25,000 + 25,000) * \text{schema.NumByteSlice}$$

100 000 - базова комісія microAlgo за підключення
25 000 + 3 500 = 28 500 для кожного Uint у схемі локального стану
25 000 + 25 000 = 50 000 для кожного фрагмента байтів у схемі локального стану

Глобальне сховище фактично зберігається в обліковому записі творця додатку, тому цей обліковий запис відповідає за мінімальний баланс глобального сховища. Коли обліковий запис підключається до додатку, він відповідає за мінімальний баланс локального сховища.

Приклад


Дано смарт-контракт з 1 глобальною парою ключ-значення типу `byteslice` та 1 парою ключ-значення локального сховища типу `integer` і без додаткових сторінок.

Мінімальний баланс облікового запису, який створює додаток буде збільшено на 150 000 мікроалго.


$$100,000 + 50,000 = 150,000$$

Мінімальний баланс облікового запису, який приєднується до додатку, становить 128 500 `microAlgos`.

$$100,000 + 28,500 = 128,500$$

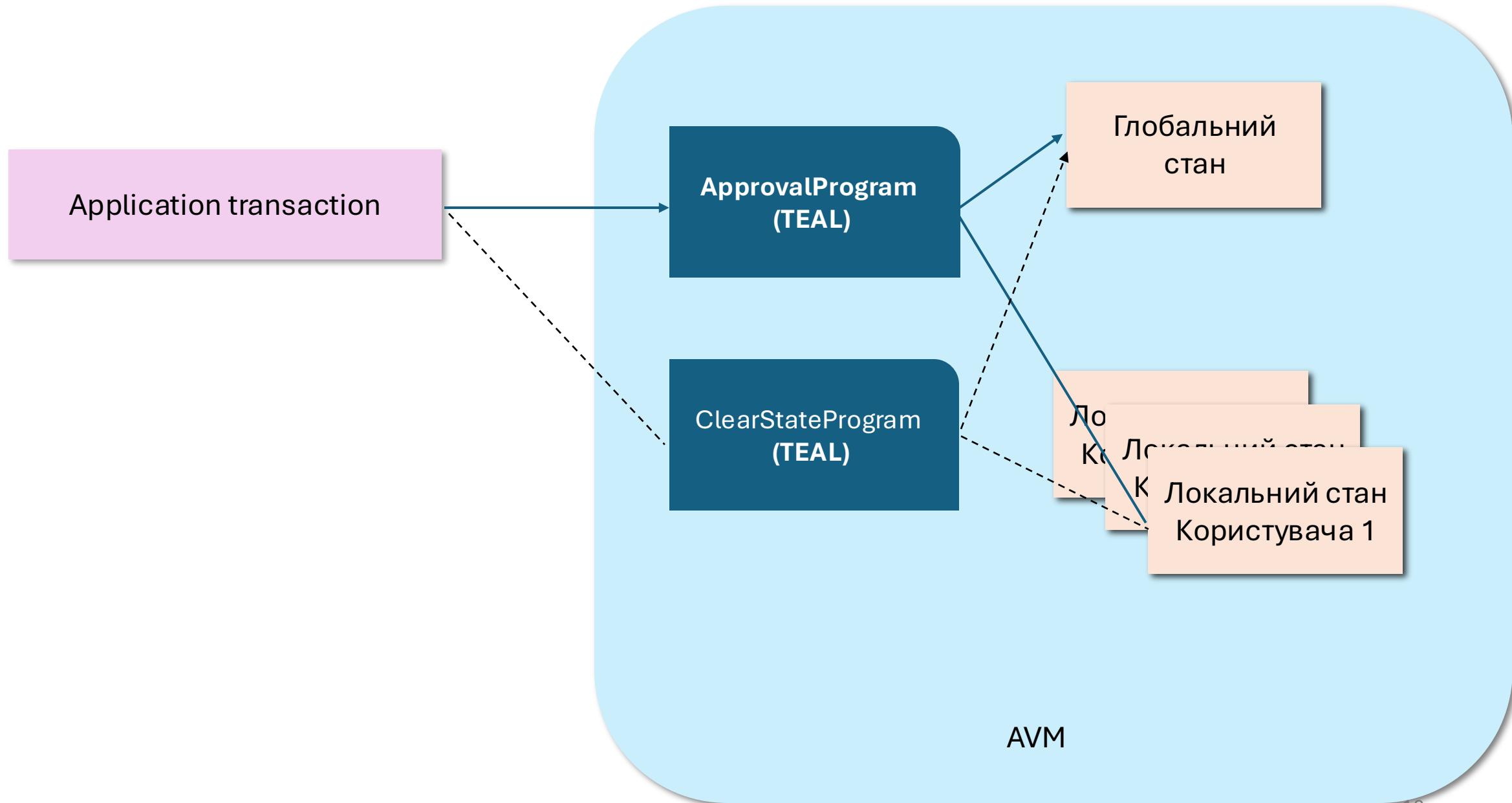


ЖИТТЄВИЙ ЦИКЛ СМАРТ- КОНТРАКТУ



Смарт-контракти реалізуються за допомогою двох програм:

- **ApprovalProgram** відповідає за обробку всіх викликів до контракту, за винятком виклику Clear. Ця програма відповідає за реалізацію більшості логіки додатку. Ця програма буде успішною, лише якщо після завершення програми в стеку залишиться одне ненульове значення або код операції, що повертається, викликається з позитивним значенням у верхній частині стека. Невдалий виклик ApprovalProgram смарт-контракту не є дійсною транзакцією і тому не записується до блокчейну.
- **ClearStateProgram** використовується для обробки облікових записів за допомогою виклику clear, щоб видалити смарт-контракт із запису балансу. Ця програма пройде або не пройде так само, як це робить ApprovalProgram.





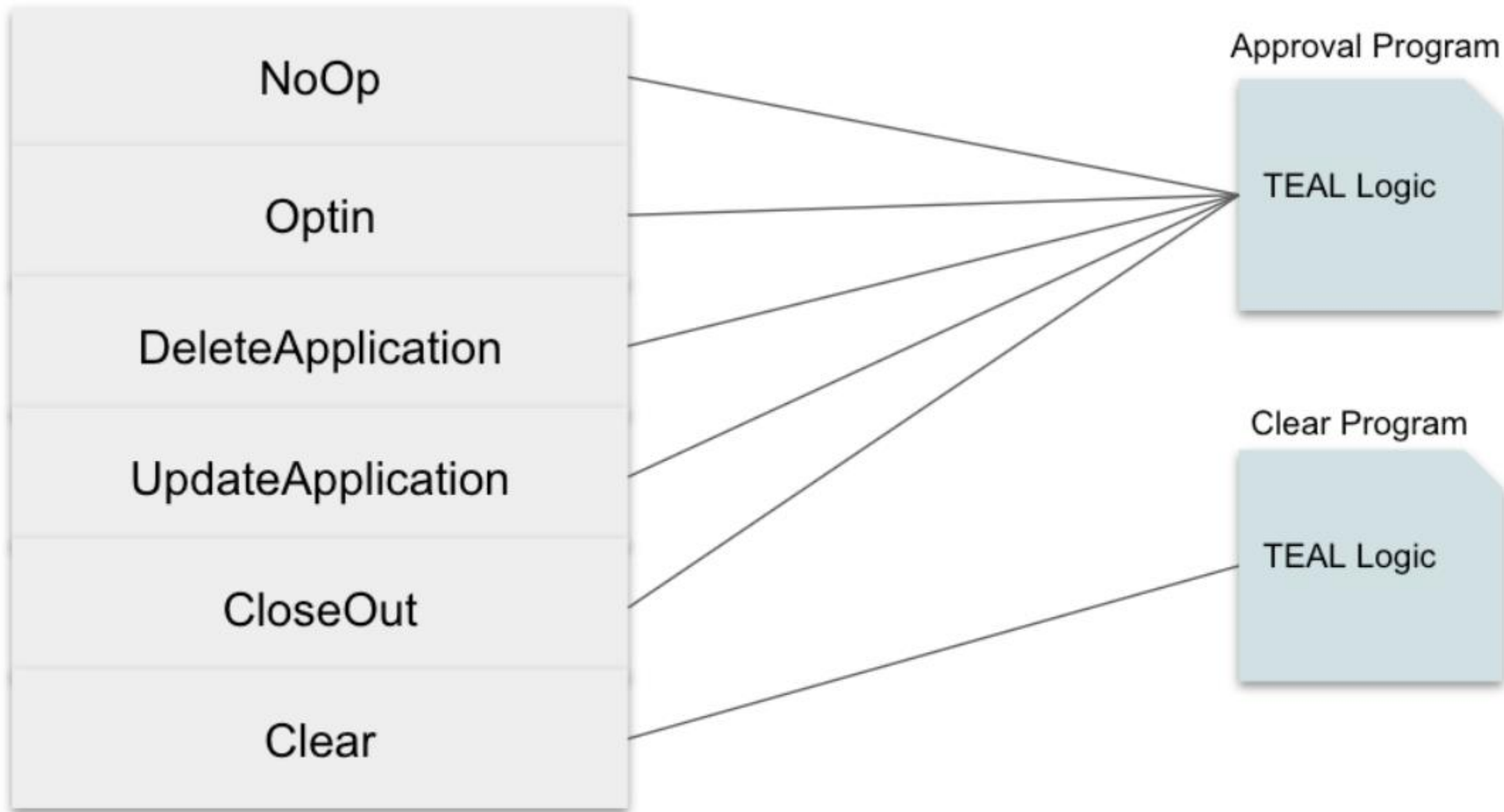
Application Transaction Types

Виклики смарт-контрактів реалізуються за допомогою транзакцій `ApplicationCall`.

Ці типи транзакцій є такими:

- `NoOp` – загальні виклики програми для виконання `ApprovalProgram`.
- `OptIn` – облікові записи використовують цю транзакцію, щоб розпочати участь у смарт-контракті. Участь дозволяє використовувати локальне сховище.
- `DeleteApplication` - транзакція для видалення додатку.
- `UpdateApplication` – транзакція для оновлення програм TEAL для контракту.
- `CloseOut` – облікові записи використовують цю транзакцію, щоб припинити свою участь у контракті. Цей виклик може завершитися помилкою на основі логіки TEAL, що не дозволить обліковому запису видалити контракт із запису балансу.
- `ClearState` — подібно до `CloseOut`, але транзакція завжди очищає контракт із запису балансу облікового запису незалежно від того, успішно чи не вдасться програмі.

Смарт-контракт з відстеженням стану повинен мати можливість виявляти будь-який тип виклику програми та відповідним чином виконувати певні гілки логіки TEAL.



+

•

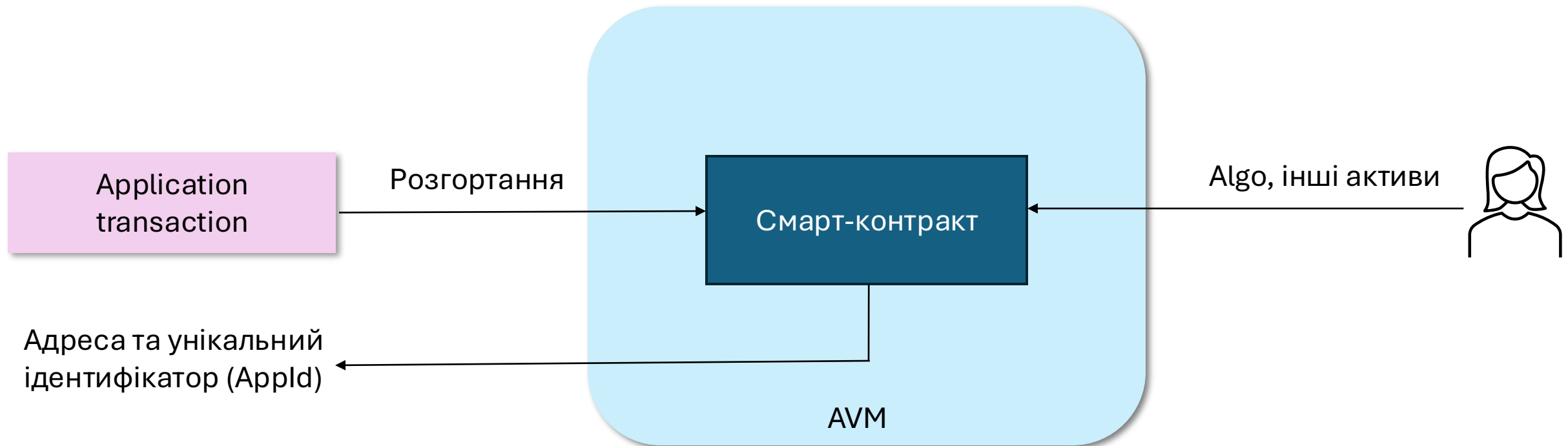
○

Рахунок умовного депонування

Після розгортання смарт-контракт має власну адресу та може функціонувати як **рахунок умовного депонування**.

- Рахунок умовного депонування - це рахунок, на якому кошти блокуються доти, доки не відбудеться якась заздалегідь визначена подія або не буде виконано певний набір умов. Умови, що визначають, коли кошти мають бути надіслані, закодовані і таким чином реалізуються логікою самого контрактного рахунку.
- Це усуває необхідність у централізованому органі, який визначатиме, чи виконано умову, а потім модеруватиме транзакцію.

Смарт-контракт як рахунок умовного депонування



Створення першого смарт-контракту

Мови написання смарт-контрактів

Algorand Smart Contracts (ASC1) можуть бути написані на двох основних мовах: Teal і PyTeal.

Teal (Transaction Execution Approval Language) - це низькорівнева мова для написання смарт-контрактів на Algorand. Вона заснована на стеках і працює безпосередньо на віртуальній машині Algorand (AVM). Teal дозволяє писати ефективні та оптимізовані контракти, але може бути складною для розуміння та використання.

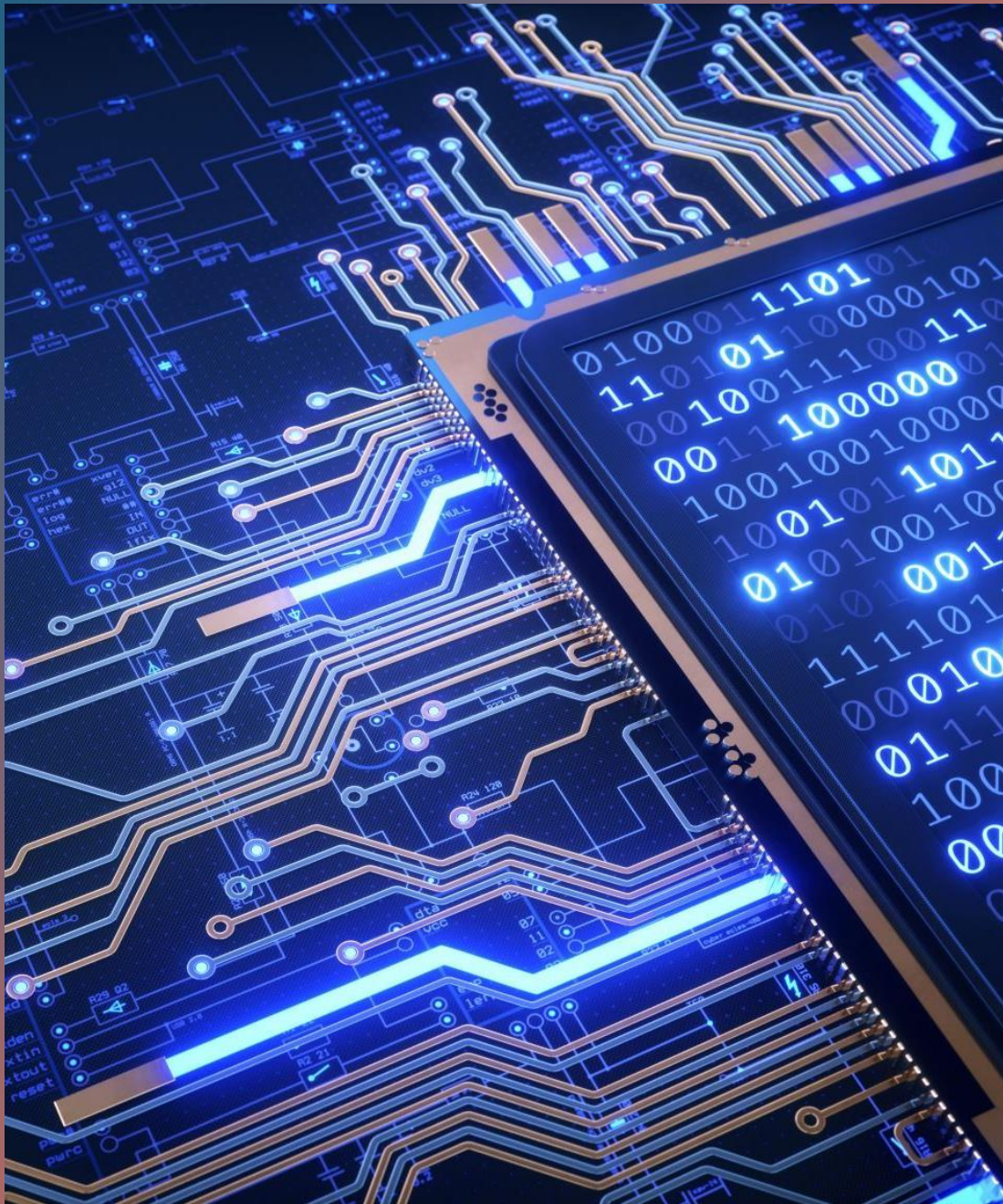
Приклад простого смарт-контракту на Teal:

```
// Program that approves if the transaction sender is a
  specific address

txn Sender

addr YOUR_ADDRESS_HERE

==
```

Teal - Transaction Execution Approval Language

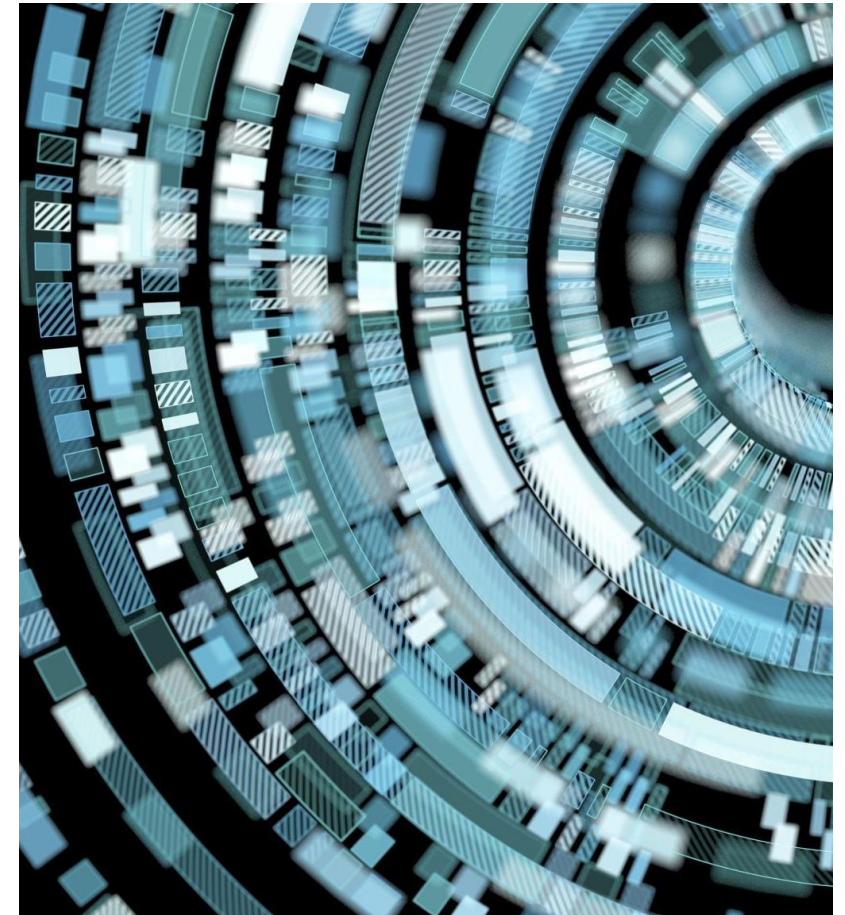
Основні характеристики:

- Мова стеку на основі байт-коду
- Повнота Тьюринга
- Має цикли та підпрограми
- У стеку можуть бути тільки Uint64 та Byte[]
- > 140 Opcodes
- Бібліотека PyTeal для написання смарт-контрактів на python

Algorand Virtual Machine (AVM)

AVM — це віртуальна машина, яка виконує смарт-контракти на блокчейні Algorand. Вона є центральним компонентом для реалізації логіки смарт-контрактів, написаних на мові TEAL (Transaction Execution Approval Language) або PyTeal (Python API для TEAL).

Ця віртуальна машина містить стековий механізм, який оцінює смарт-контракти та смарт-підписи відповідно до транзакцій, з якими вони викликаються. Програми або відхиляють транзакцію, або виконують зміни згідно з логікою і змістом транзакцій.



+

•

○

Algorand Virtual Machine (AVM)

- **Smart Contracts (Stateful):** Якщо виклик смарт-контракту завершується успішно, зміни записуються в блокчейн після підтвердження блоку. Якщо виклик не вдається, всі зміни, спричинені викликом, не будуть збережені на блокчейні.
- **Smart Signatures (Stateless):** Логіка смарт-підписів подається разом із транзакцією, і AVM оцінює її. Якщо логіка не виконується, транзакція не буде застосована.



PyTeal

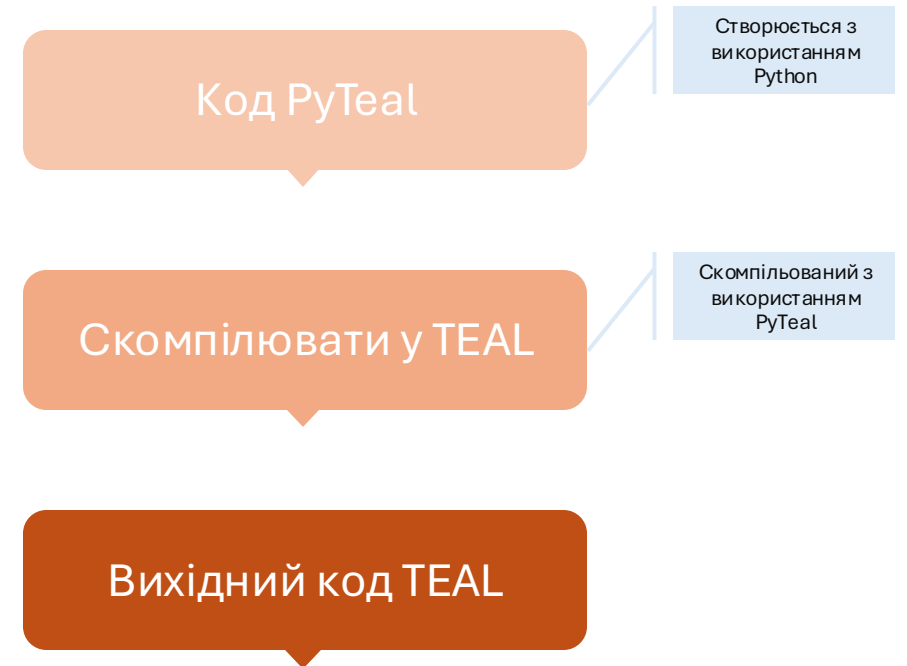


PyTeal — це прив'язка до мови Python для смарт-контрактів Algorand (ASC1).

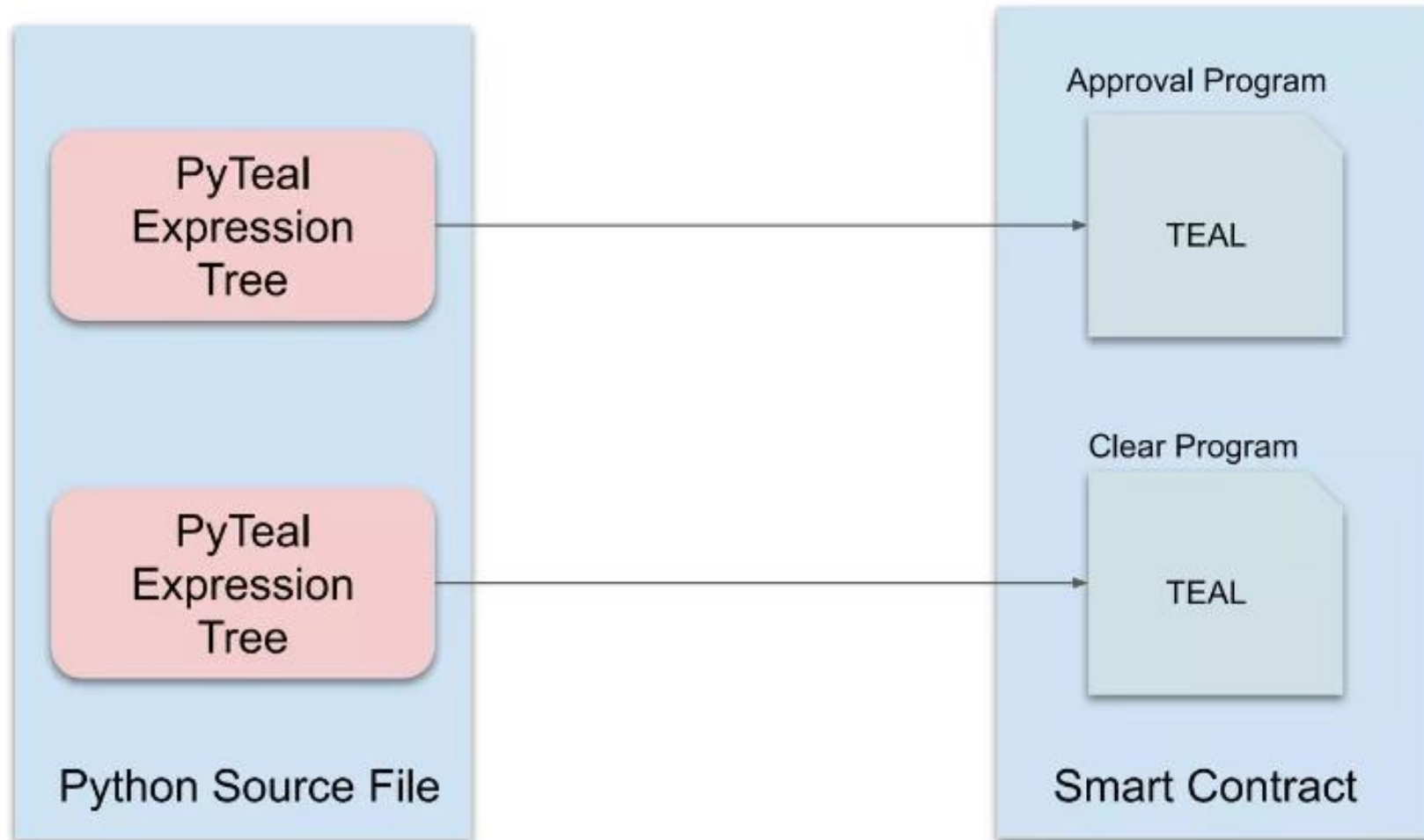
TEAL по суті є мовою асемблера. Завдяки PyTeal розробники можуть виражати логіку смарт-контрактів виключно за допомогою Python. PyTeal забезпечує абстракції високого рівня функціонального стилю програмування над TEAL і виконує перевірку типів під час створення.

PyTeal

- Бібліотека Python, яка створює код TEAL
- Вихідний код TEAL можна скомпілювати в байт-код і розгорнути в блокчейні
- Контракти створюються як дерева виразів PyTeal



PyTeal y TEAL



Перший смарт-контракт

Кожен файл PyTeal складається з двох основних функцій:

approval()	<pre>def approval(): program = Return(Int(1)) return compileTeal(program, Mode.Application, version=6)</pre>
clear()	<pre>def clear(): program = Return(Int(1)) return compileTeal(program, Mode.Application, version=6)</pre>

+

•

○

Перший сма^т- контракт

Код, написаний на PyTeal перетворюється у програму TEAL, яка складається з послідовності кодів операцій TEAL.

Наш `approval.teal` має таку структуру:

```
#pragma version 6  
int 1  
Return
```

Перший рядок визначає версію TEAL, яка використовується.

Другий рядок визначає ціле число 1 і поміщає його в стек.

Третій рядок повертає значення 1.

01

Коли TEAL повертає 1, це означає, що все гаразд і транзакція смарт-контракту дійсна

02

1 представляє значення, яке повертає вираз `Approve()`, який використовується в `PuTeal`

03

Коли код TEAL повертає число, відмінне від 1, це означає, що транзакція смарт-контракту недійсна, і смарт-контракт не виконується

'goal'

CLI інструмент

Після написання смарт-контракту, його необхідно задеплоїти на блокчейн Algorand. Це можна зробити за допомогою команди `goal` у середовищі Algorand Sandbox.

Інтерфейс командного рядка (command-line interface — CLI) **'goal'** — це потужний інструмент, наданий Algorand, який дозволяє розробникам і користувачам взаємодіяти з блокчейном Algorand. Він надає повний набір команд для керування обліковими записами, створення та взаємодії з транзакціями, компіляції та розгортання смарт-контрактів і моніторингу блокчейну.

Часто використовувані команди 'goal'

Команди для облікового запису:

- *goal account new*: створення нового облікового запису Algorand.
- *goal account list*: повертає список усіх облікових записів, якими керує вузол.
- *goal account balance -a [ADDRESS]*: повертає баланс конкретного рахунку.

Команди транзакцій:

- *goal clerk send*: створення платіжної транзакції.
- *goal clerk dryrun*: тестування транзакції або смарт-контракту, не надсилаючи їх у мережу.

Часто використовувані команди 'goal'

Команди для розумних контрактів:

- *goal clerk compile -o [OUTPUT_FILE] [SOURCE_FILE]*: компілює смарт-контракт TEAL.
- *goal app create [flags]*: створює новий додаток (розумний контракт).
- *goal app call*: виклик додатку, що вже існує.
- *goal app optin [flags]*: використовується для підключення облікового запису Algorand до певного додатку.
- *goal app info --app-id [id]*: перевірте основну інформацію для додатку.
- *goal app read --global --app-id [id]*: перевірте глобальне сховище додатку.

Приклади команд

Створення нового облікового запису


```
root@c17dae2797fd:~/testnetwork/Node# goal account new
Created new account with address 2YEK4W65H2UMTRFQJMU2KYJ46SPQ5GAQKNIQY5NCCI7JBDXUE3TMJGTSWQ
```

Перевірка балансу облікового запису

```
root@c17dae2797fd:~/testnetwork/Node# goal account balance -a 2YEK4W65H2UMTRFQJMU2KYJ46SPQ5GAQKNIQY5NCCI7JBDXUE3TMJGTSWQ
0 microAlgos
```

Пересилання грошей з одного рахунку на інший

```
root@c17dae2797fd:~/testnetwork/Node# goal clerk send --amount 123456789 --from $ADDR1 --to $ADDR2
Sent 123456789 MicroAlgos from account FXA6NGWY6MR2Y5Q7ZPFH7CC7OWHX5JAIED22235KH6RPK55GHUB5JH4054 to address JLFS5K27S74MN2HF4C56
7UBK64LCJNPVE4DNKWX4W3ULGTNUVZH5QH2ML4, transaction ID: A4E5V7STMRN326AVHYCWPTCKJ6VJ7EAQE6GN4U4EAMZUDACU3QEQ. Fee set to 1000
Transaction A4E5V7STMRN326AVHYCWPTCKJ6VJ7EAQE6GN4U4EAMZUDACU3QEQ still pending as of round 244427
Transaction A4E5V7STMRN326AVHYCWPTCKJ6VJ7EAQE6GN4U4EAMZUDACU3QEQ still pending as of round 244428
Transaction A4E5V7STMRN326AVHYCWPTCKJ6VJ7EAQE6GN4U4EAMZUDACU3QEQ committed in round 244429
```



Додаткові матеріали

- [Algorand Smart Contracts](#) (Презентація)
- [Smart Contracts Overview](#) (Відео)
- [What are smart contracts?](#) (Стаття)
- [Smart contracts and signatures](#) (офіційна документація Algorand)