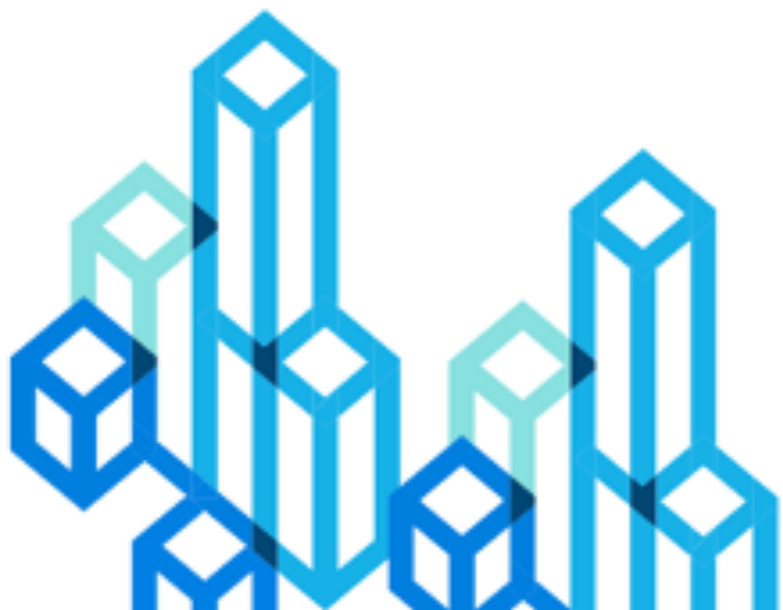
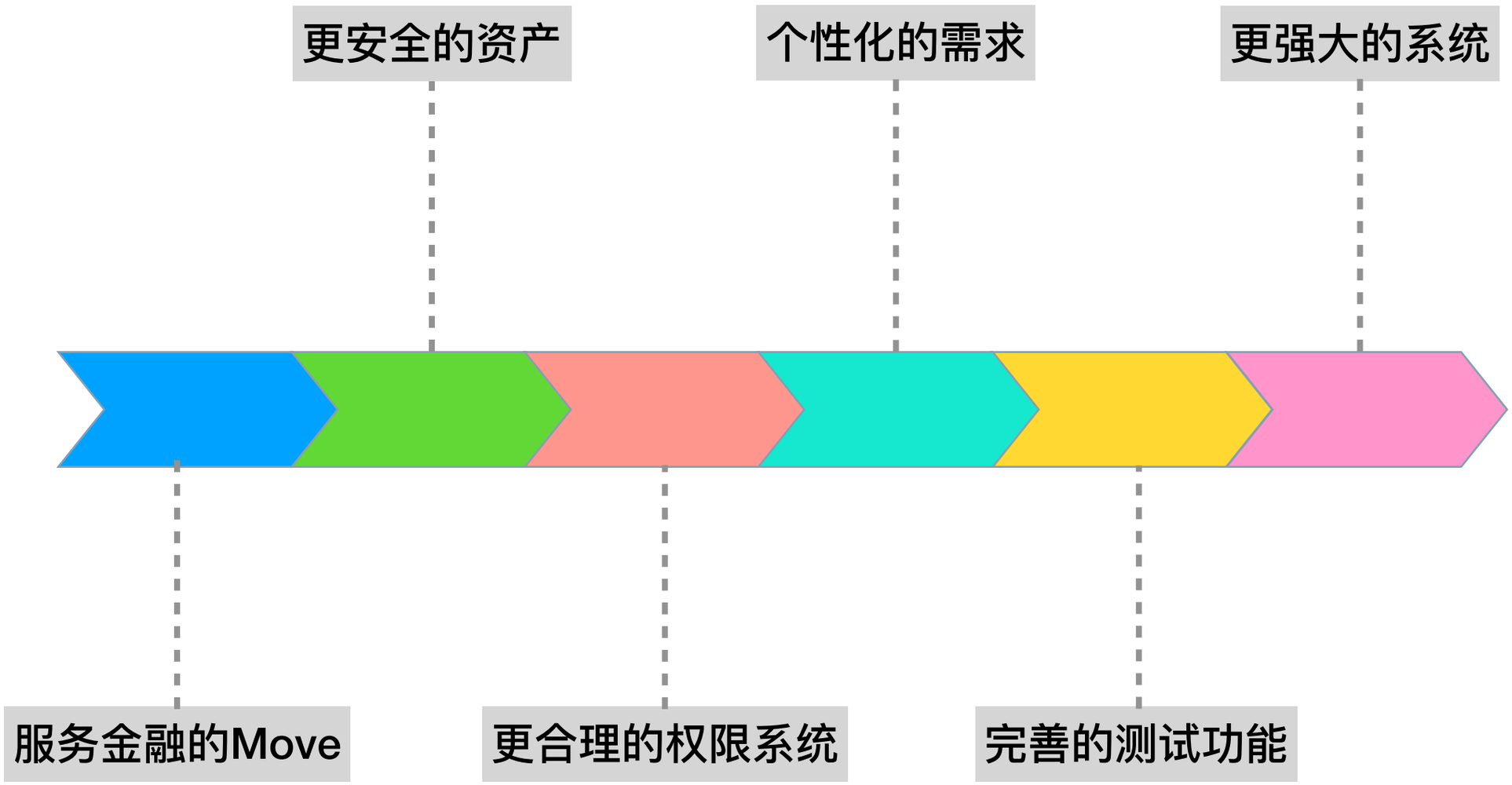
为什么我们需要新的智能合约语言Move?

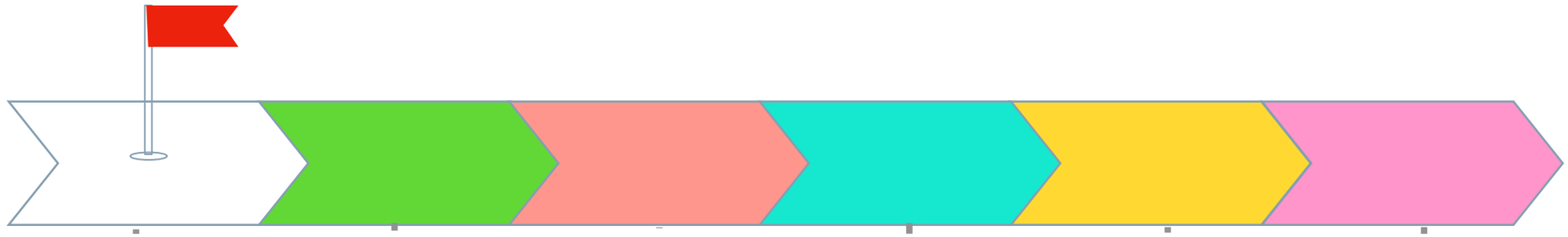
- 邓启明 Starcoin核心开发者



Agenda



服务金融的Move



服务金融的Move

进化的智能合约

Move介绍

更安全的资产

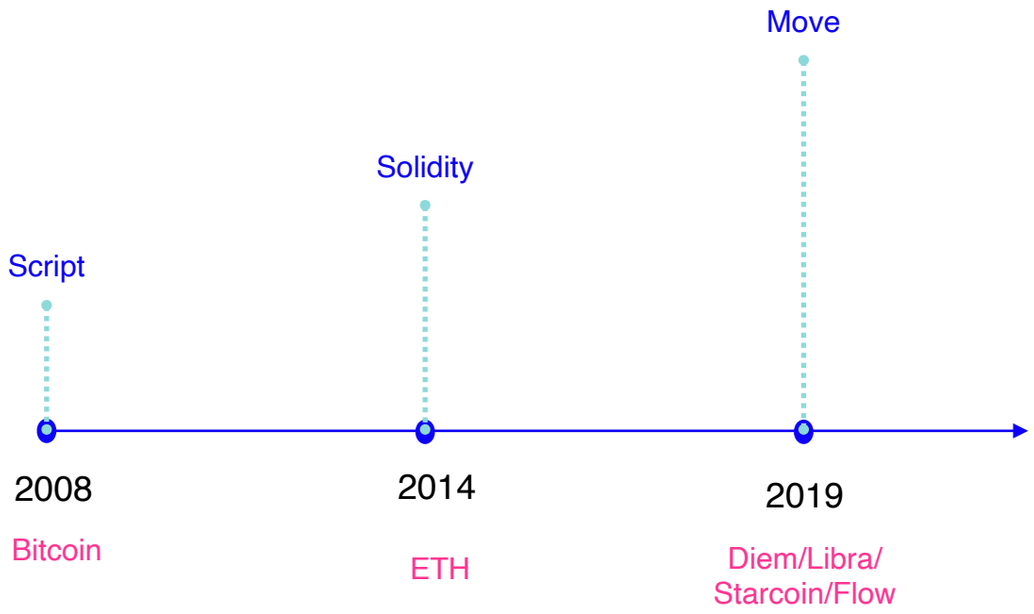
更合理的权限系统

个性化的需求

完善的测试功能

更强大的系统

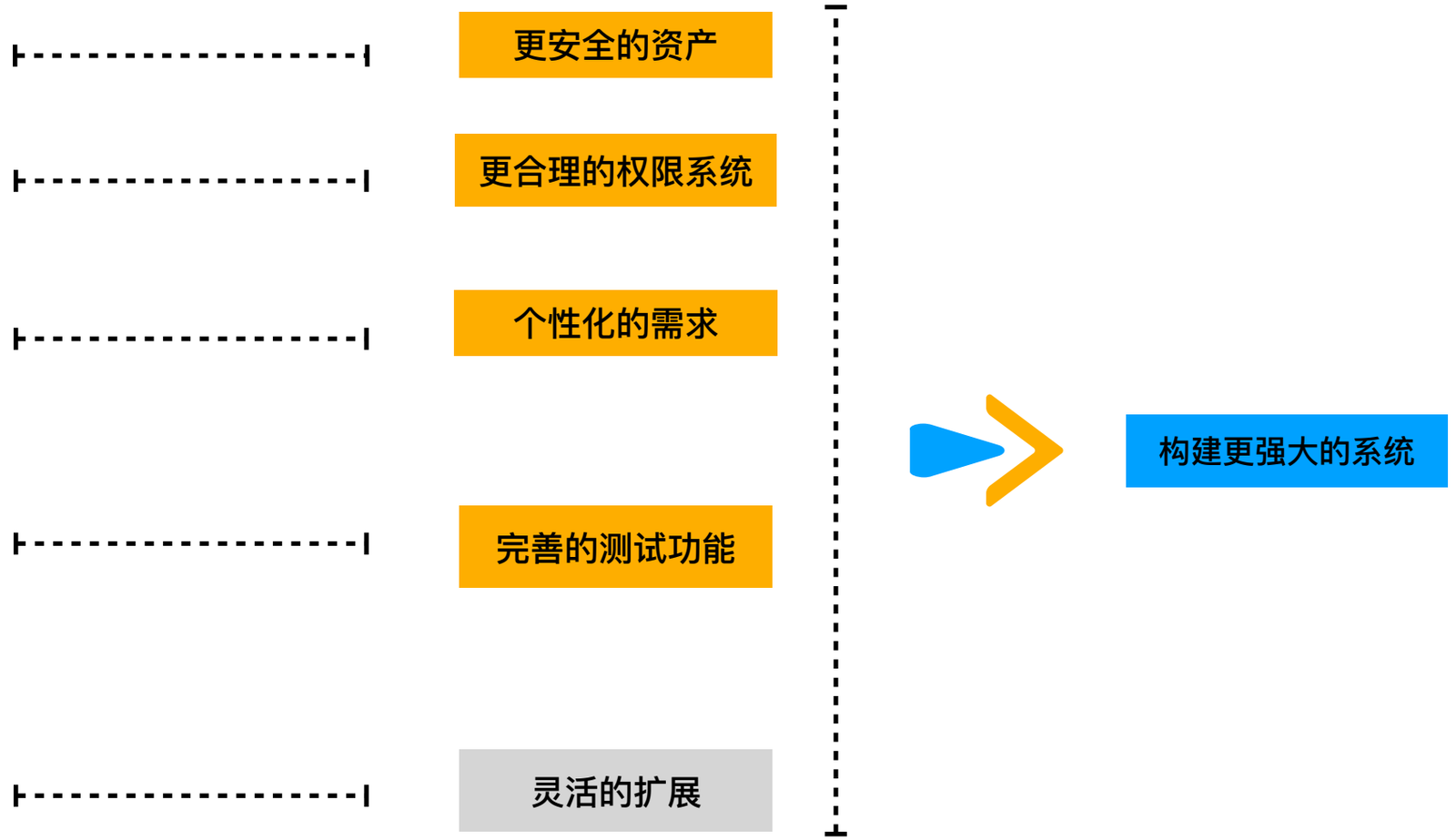
进化的智能合约



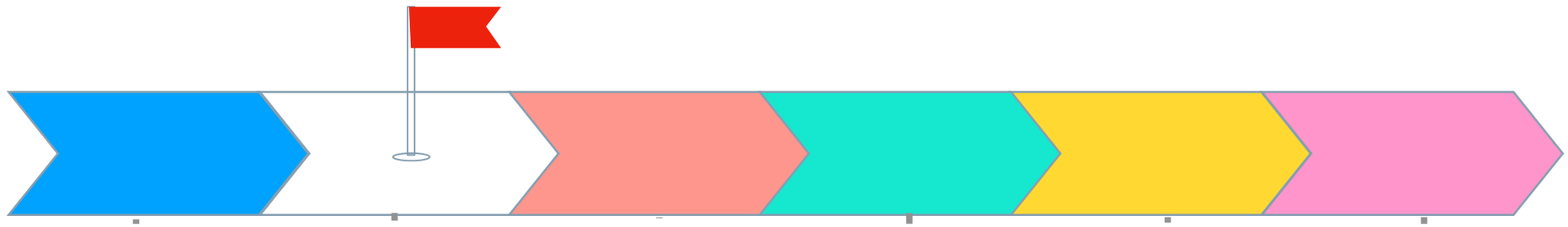
- Script
  - 表达能力不足，比较原始
- Solidity图灵完备，表达能力好
  - 安全性不足
  - 很难构建复杂系统
  - 任何修改都要更新合约
- Move图灵完备，线性逻辑的语言，针对金融场景做了安全加固，安全性高，模块化等等

Move介绍

- Ability
- Visibility
- Script & Module
- Unit Test
- 形式化验证
- 泛型编程



更安全的资产



服务金融的Move

进化的智能合约

Move介绍

更安全的资产

资源 & 信息

Ability

更合理的权限系统

个性化的需求

完善的测试功能

更强大的系统

资源 & 信息

信息

资源

- Move线性逻辑语言
- 区分资源类型与普通类型(信息)
- 资源类型，让资产更安全

信息

信息

- 信息，是一个科学术语。
- 信息是物质存在的一种方式、形态或运动形态，也是事物的一种普遍属性。
- 信息依据载体的不同被分为4大类：文字、图形（图像）、声音、视频。
- 信息可以廉价复制，可以广泛传播。

- 普遍属性
- 可复制

资源

资源

- 资源（经济学名词）。
- 资源是指任何一种有形或者无形、可利用性有限的物体，或者是任何有助于维持生计的事物。
- 资源主要具有三种特性： 实用性、数量（常常依据的是其可利用性）以及在生产其他资源方面的用途。

- 有价值
- 有限

资源 vs 信息

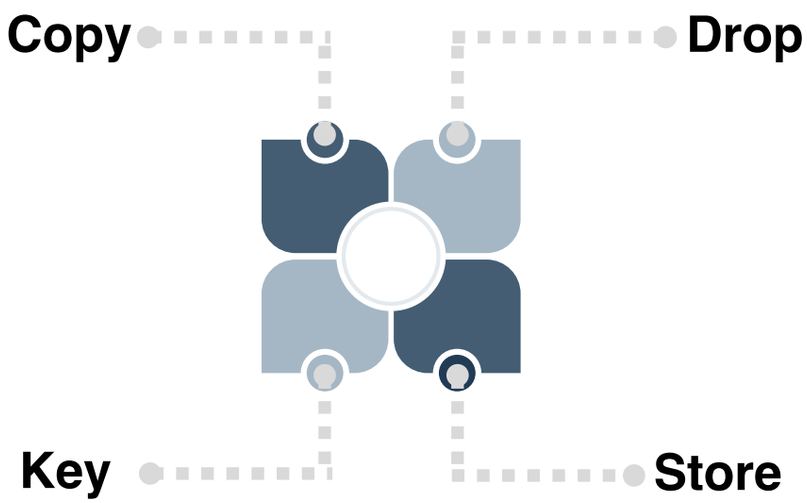
信息

资源

- 信息传递依靠可复制性。例如，微博。
- 价值传递必须突破可复制性。例如，古董。
- 「可复制性」与「价值实现」的矛盾。例如，盗版。

- Move如何解决这个问题?

Ability



- Copy：可复制
- Drop：可丢弃（凭空消失/垃圾回收）
- Key：可检索
- Store：可存储

- 信息：可存储、可复制 ...
- 资源：可存储、不可复制 ...

- 怎么用?

关键字has

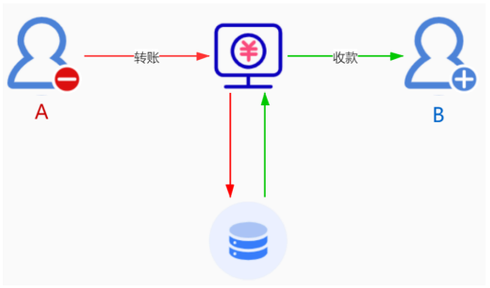
- 例子

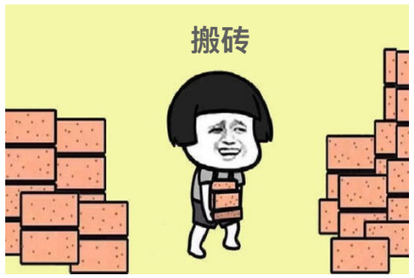
```
struct Test has key, copy, drop, store { }    // has
```

- 关键字：has
- Ability可任意组合
- 基本数据类型有所有4种Ability
- 默认不具备任何Ability

- 简单灵活
- 安全可靠
- 开箱即用

资源的move语义





Solidity VS Move

- Solidity：数字加减操作，没有事务保障
- Move：资源的所有权转移
  - 不能复制
  - 传输过程中不能修改
  - 能存储

- 买卖本质上是所有权转移
- 安全可靠

不安全的资产

```
function _transfer(address _from, address _to, uint256 _value) internal returns (bool) {
    require(_to != address(0));
    uint256 oldFromVal = balances[_from];     // 1
    require(_value > 0 && oldFromVal >= _value);
    uint256 oldToVal = balances[_to];    // 2
    uint256 newToVal = oldToVal + _value;
    require(newToVal > oldToVal);
    uint256 newFromVal = oldFromVal - _value;
    balances[_from] = newFromVal;       // 3
    balances[_to] = newToVal;      // 4          如果 _from == _to 被覆盖

    assert((oldFromVal + oldToVal) == (newFromVal + newToVal));
    emit Transfer(_from, _to, _value);

    return true;
}
```

- 信息，可复制
- 传输过程可修改
- 大值覆盖小值
- 更多安全隐患，比如溢出

- 无限增发的漏洞

更安全的资产

```
module Token {
    struct Token has store {  // 1. 定义 Token
        amount: u128
    }

    public fun withdraw(token: &mut Token, amount: u128): Token {
        token.amount -= amount;                                    // 2. 取款
        Token {
            amount
        }
    }

    public fun deposit(to_token:&mut Token, coins: Token) {  // 3. 存款
        let Token { amount } = coins;
        token.amount += amount;
    }
}
```

定义资产

```
module Account {
    use 0x1::Token;
    use 0x1::Singer;

    struct Account has key {   // 4. 账号
        balance: Token,
    }

    public fun transfer(account: &singer, receiver: address, amount: u128) {  // 5. 转账
        let myself = Singer::address_of(account);
        let balance = borrow_global_mut<Account>(myself).balance;  // 6. 获取账号 mut instance
        let coins = Token::withdraw(balance, amount); // 7. 取款
        let receiver_balance = borrow_global_mut<Account>(receiver).balance; // 8. 获取 mut
        Token::deposit(receiver_balance, coins);  // 9. 存款
    }
}
```

转移资产

- Token：通过Ability达到只能store，不能copy和drop
- 所有权转移：过程中不能修改，只能转移所有权

更多安全场景

- 道路千万条，安全第一条

- 资产
- NFT
- 闪电贷
- DEX
- 其他场景

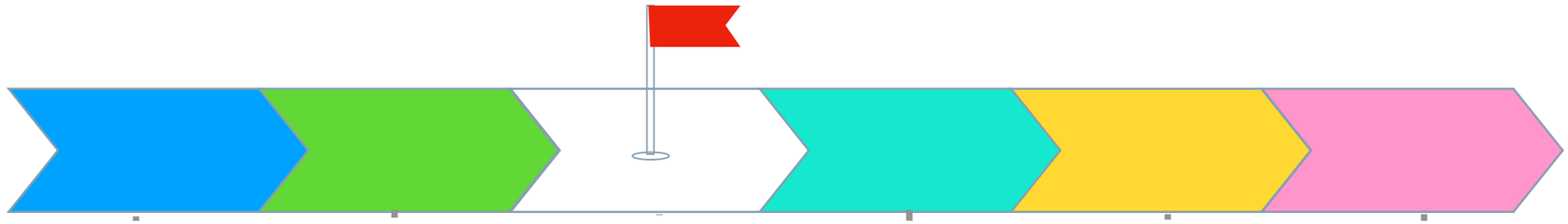
- 贴近真实的金融场景
- 降低开发者安全门槛

非同质化代币NFT

```
module NFTExample {
    use 0x1::Signer;
    use 0x1::Vector;

    struct NFT has key, store { name: vector<u8> }    // 1. 定义了一个可以 store，不能 copy 和 drop 的 struct

    struct UniqIdList has key, store {     // 2. 模拟了一个注册中心，用于保证 name 的唯一
        data: vector<vector<u8>>
    }

    public fun initialize(account: &signer) {    // 3. 初始化注册中心
        move_to(account, UniqIdList {data: Vector::empty<vector<u8>>()});
    }

    public fun new(account: &signer, name: vector<u8>): NFT acquires UniqIdList {    // 4. 发 NFT
        let account_address = Signer::address_of(account);
        let exist = Vector::contains<vector<u8>>(&borrow_global<UniqIdList>(account_address).data, &name);
        assert(!exist, 1);
        let id_list = borrow_global_mut<UniqIdList>(account_address);
        Vector::push_back<vector<u8>>(&mut id_list.data, copy name);
        NFT { name }
    }

    public fun destroy(account_address: address, nft: NFT) acquires UniqIdList {    // 5. 销毁 NFT
        let NFT { name } = nft;
        let (exist, index) = Vector::index_of<vector<u8>>(&borrow_global<UniqIdList>(account_address).data, &name);
        assert(!exist, 2);
        let id_list = borrow_global_mut<UniqIdList>(account_address);
        Vector::remove<vector<u8>>(&mut id_list.data, index);
    }
}
```

- 独一无二
  - 不可复制
  - 注册中心
- 不可分割
  - 不可修改
- 可存储

更合理的权限系统



服务金融的Move

进化的智能合约

Move介绍

更安全的资产

资源 & 信息

Ability

更合理的权限系统

友元可见性

个性化的需求

完善的测试功能

更强大的系统

真实的案例

- 单合约达到Maximum gas limit，部署不上去
- 不能返回Struct，也不能访问Struct
  - 导致合约很难拆分（ABI V2已经改善）
- 合约拆分之后，合约之间通过public函数调用，对外暴露不必要的函数

- 这是真实的案例，很难实现更庞大的系统

访问控制

修饰符	当前类	同一包内	子孙类(同一包)	子孙类(不同包)	其他包
public	Y	Y	Y	Y	Y
protected	Y	Y	Y	Y/N（说明）	N
default	Y	Y	Y	N	N
private	Y	N	N	N	N

- 类可见
- 包可见
- 任意可见

- **Java**

函数可见性

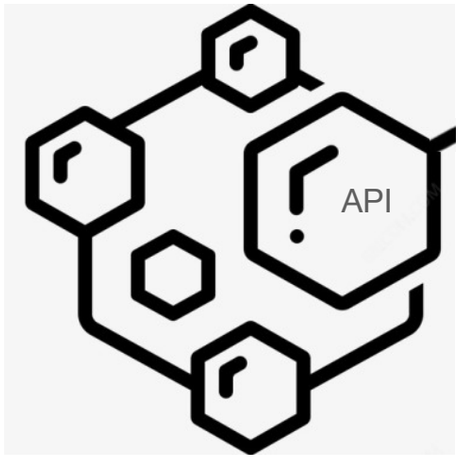
可见性	访问权限
private	module
friend	address
public	all

- 默认private
- 减少不必要的方法暴露
- 构建更复杂的系统

```
address 0x2 {
  module A {
    // friend declaration via fully qualified module name
    friend 0x2::B;   // 6

    // friend declaration via module alias
    use 0x2::C;
    friend C;  // 7

    fun i_am_private() {    // 1
      // other functions in the same module can also call friend functions
      bar();
    }

    public fun i_am_public() {   // 2
      // other functions in the same module can also call friend functions
      bar();
    }

    public(script) fun i_am_script() {}   // 3

    public(friend) fun bar() {}    // 4

    public(friend) fun foo() {   // 5
      // a friend function can call other non-script functions in the same module
      i_am_private();
      i_am_public();
      bar();
    }
  }
}
```

- Move

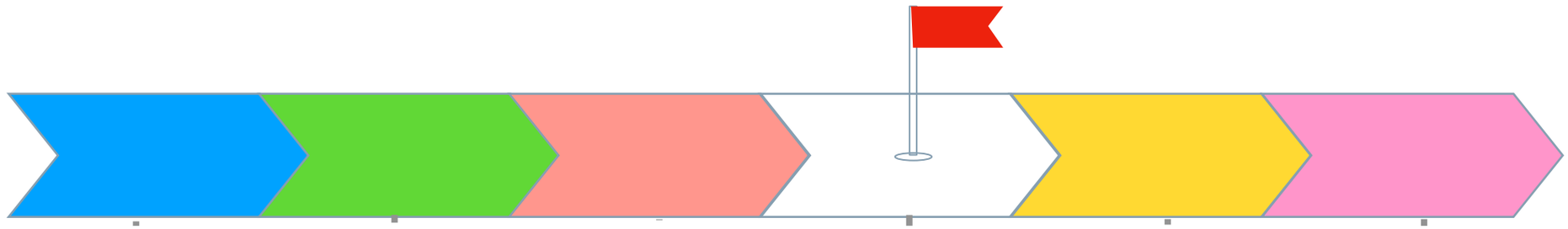
更庞大的系统



- 轻松部署
- 模块化组装
- 只对外提供必要的入口

编写功能更强大的合约

个性化的需求



服务金融的Move

进化的智能合约

Move介绍

更安全的资产

资源 & 信息

Ability

更合理的权限系统

友元可见性

个性化的需求

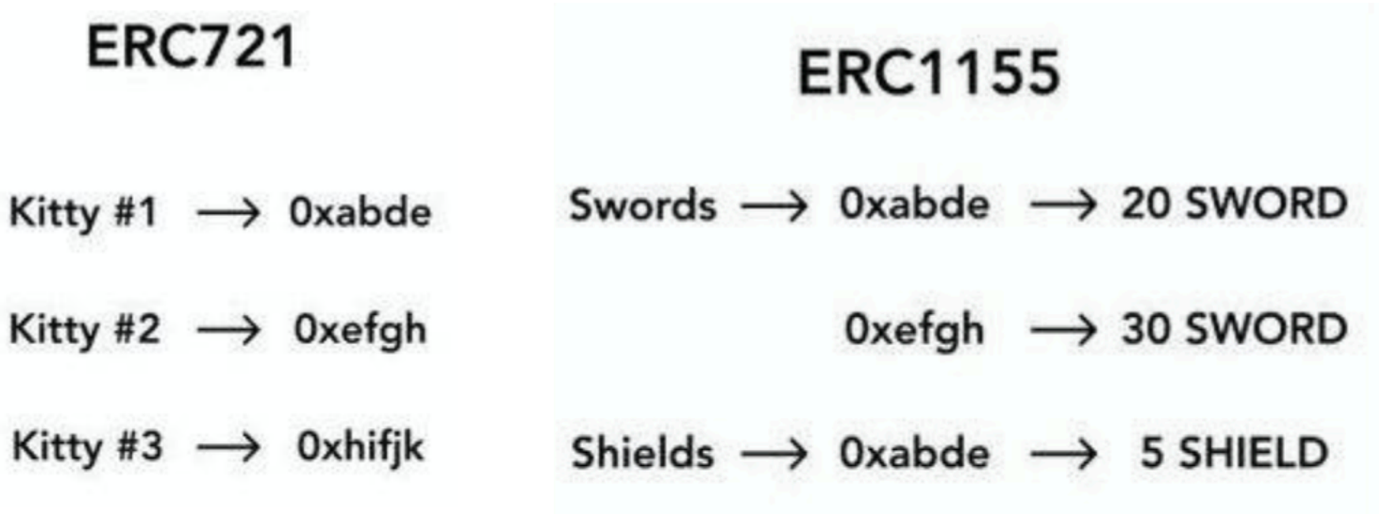
个性化的闪电贷

灵活的Script

完善的测试功能

更强大的系统

从ERC721到ERC1155



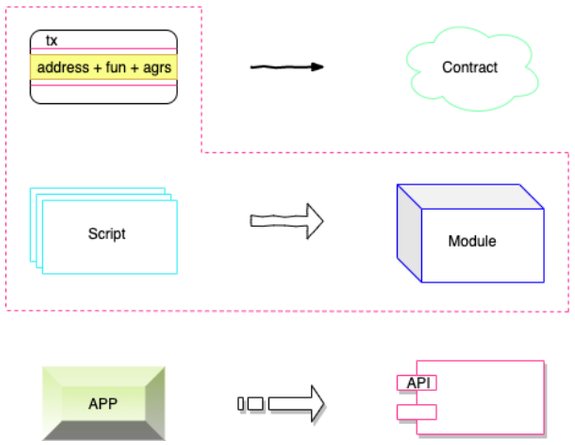
- 数量
- 游戏装备

1. ERC721不能批量操作NFT，批量操作只能通过发起批量交易实现，成本太高

2. 交易只能调用链上合约定义好的单个方法：address + fun + args

3. 交易不具备编程能力，要满足新的需求，只能部署新的合约

灵活的Script



- Module：部署在链上的合约
- Script：交易的Body
- Script：可编程的交易，自由封装逻辑

- **Module & Script**

可编程的Script

```
module NFTExample {
    use 0x1::Signer;
    use 0x1::Vector;

    struct NFT has key, store { name: vector<u8> }    // 1. 定义了一个可以store，不能copy和drop的struct

    struct UniqIdList has key, store {     // 2. 模拟了一个注册中心，用于保证name的唯一
        data: vector<vector<u8>>
    }

    public fun initialize(account: &signer) {    // 3. 初始化注册中心
        move_to(account, UniqIdList {data: Vector::empty<vector<u8>>()});
    }

    public fun new(account: &signer, name: vector<u8>): NFT acquires UniqIdList {    // 4. 发NFT
        let account_address = Signer::address_of(account);
        let exist = Vector::contains<vector<u8>>(&borrow_global<UniqIdList>(account_address).data, &name);
        assert(!exist, 1);
        let id_list = borrow_global_mut<UniqIdList>(account_address);
        Vector::push_back<vector<u8>>(&mut id_list.data, copy name);
        NFT { name }
    }

    public fun destroy(account_address: address, nft: NFT) acquires UniqIdList {    // 5. 销毁NFT
        let NFT { name } = nft;
        let (exist, index) = Vector::index_of<vector<u8>>(&borrow_global<UniqIdList>(account_address).data, &name);
        assert(!exist, 2);
        let id_list = borrow_global_mut<UniqIdList>(account_address);
        Vector::remove<vector<u8>>(&mut id_list.data, index);
    }
}
```

```
Script {
    use 0x2::NFTExample::{Self, NFT};
    use 0x1::Vector;
    use 0x1::Signer;

    fun main(account:signer) {  // 交易可编程，批量创建和销毁NFT
        let nft1 = NFTExample::new(&account, x"1");
        let nft2 = NFTExample::new(&account, x"2");

        let nfts = Vector::empty<NFT>();
        Vector::push_back(&mut nfts, nft1);
        Vector::push_back(&mut nfts, nft2);

        let account_address = Signer::address_of(&account);
        let nft1 = Vector::pop_back(&mut nfts);
        NFTExample::destroy(account_address, nft1);
        let nft2 = Vector::pop_back(account_address, &mut nfts);
        NFTExample::destroy(nft2);
        Vector::destroy_empty(nfts);
    }
}
```

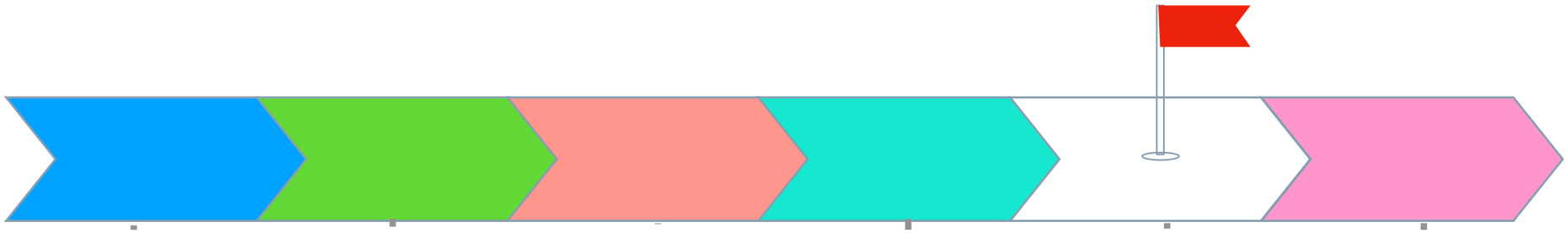
- 类似ERC721
- 批量操作NFT

1. Script：交易可编程

2. 把Module（链上）当“数据库”，在Script（交易）组装逻辑

3. Move：不升级合约，轻松实现个性化的需求

完善的测试功能



服务金融的Move

进化的智能合约

Move介绍

更安全的资产

资源 & 信息

Ability

更合理的权限系统

友元可见性

个性化的需求

个性化的闪电贷

灵活的Script

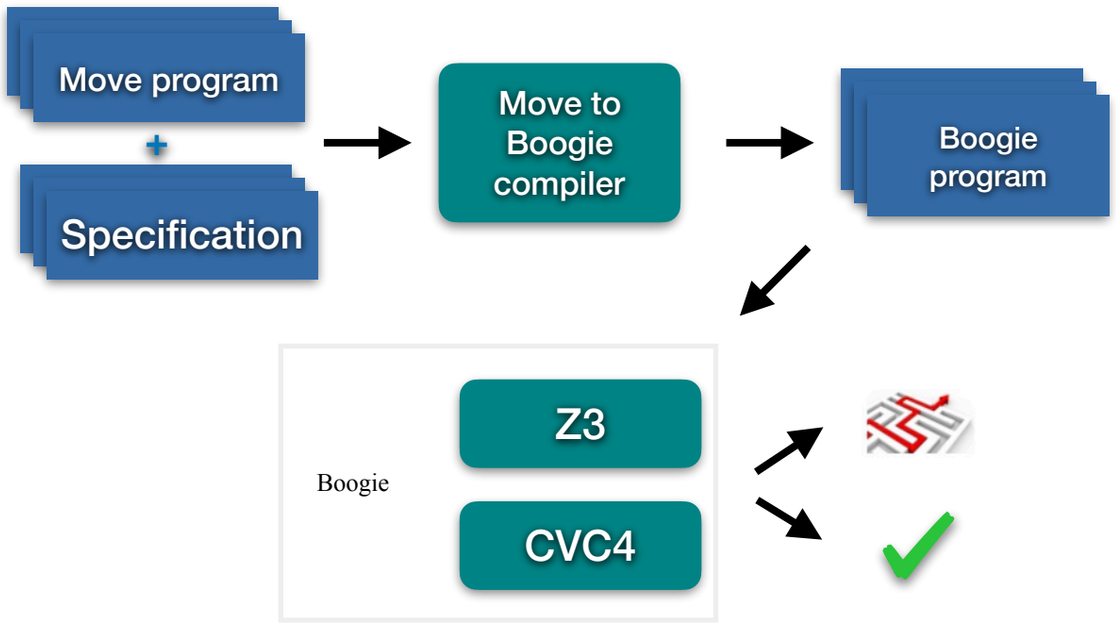
完善的测试功能

形式化验证

Unit Test

更强大的系统

形式化验证



- 早期检测错误
  - 溢出
  - 除0
  - 等等
  - 穷尽

- **Move Prover**

关键字spec

```
module M {
    struct Counter has key,store {
        value: u8,
    }

    public fun increment(a: address) acquires Counter {
        let r = borrow_global_mut<Counter>(a);
        r.value = r.value + 1;
    }

    spec fun increment {  // 1
        ...
    }

    spec module { // 2
        ...
    }
}
```

- spec module
- spec fun
- 其他

spec module

```
spec <target> {
    pragma <name> = <literal>;// pragma用于配置
}

spec module {
    pragma verify = true; // false, default, do not verify specs in this module
    pragma aborts_if_is_strict = true;
}
```

- verify
- aborts_if_is_strict

spec fun

```
public fun initialize(account: &signer, id: u8) {
    move_to(account, ChainId { id });
}

spec fun initialize {
    aborts_if exists<ChainId>(Signer::spec_address_of(account)); // 1
    ensures exists<ChainId>(Signer::spec_address_of(account));   // 2
}
```

- 终止条件：aborts_if
- 保证条件：ensures

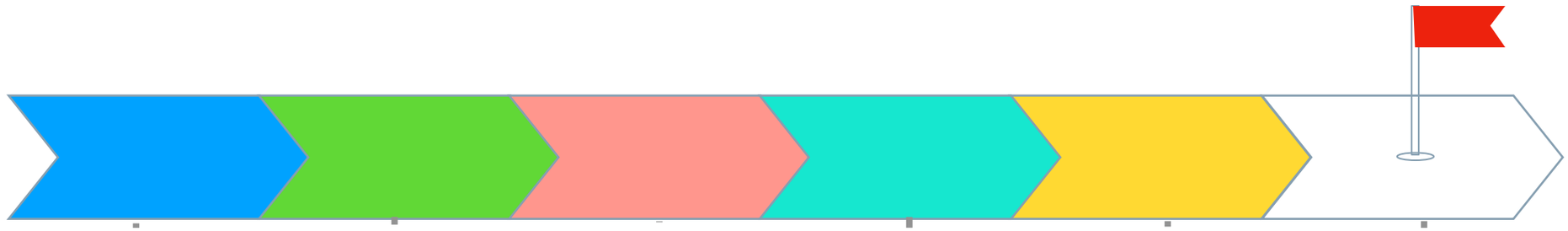
Unit Test

- #[test]
- 定义变量
- 校验结果
  - expected_failure
  - assert

```
#[test(account = @0x1)]   // 1. 定义 UT 函数，创建 account 变量
#[expected_failure(abort_code = 26119)]  // 2. 校验交易状态
fun test_deposit_not_exist(account: &signer) {
    let account_address = Signer::address_of(account);
    Vault::deposit<STCVaultPoolA::VaultPool, STC::STC>(&account_address, Token::zero<STC::STC>());
}
```

- 提供功能性测试
- 语法简洁

Agenda



服务金融的Move

进化的智能合约

Move介绍

更安全的资产

资源 & 信息

Ability

更合理的权限系统

友元可见性

个性化的需求

个性化的闪电贷

灵活的Script

完善的测试功能

形式化验证

Unit Test

更强大的系统

Starcoin x Move

Starcoin

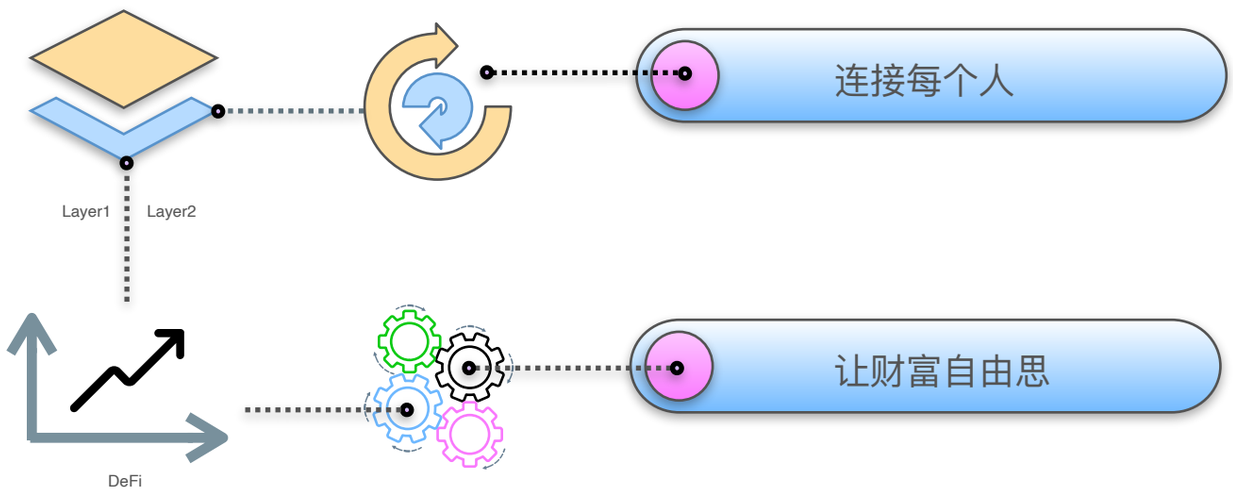
简单易学的Move

- **copy**
- **store**
- **key**
- **drop**
- **has**
- fun
- **public**
- **friend**
- let
- mut
- **script**
- **module**
- use
- struct
- assert
- abort
- if-else
- while

- 简单易学
- 安全可靠
- 灵活多变
- 贴近真实的金融场景

构建更强大的系统

Starcoin x Move

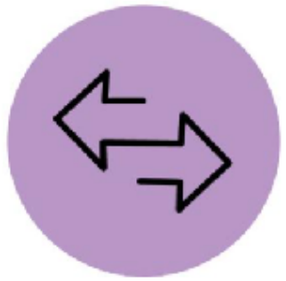


- 开放的Layer1
  - 任何人可参与的PoW共识
- 可扩展的Layer2
  - 让交互更简洁，触达更多生活场景
- 安全的DeFi
  - 让资产自动增长

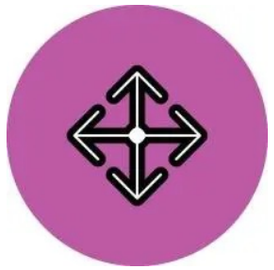
DeFi基础设施

NFT协议

时间：2021.8



StarSwap
去中心化交易所
https://www.movelang.io/t/topic/57
时间：2021.9



IDO

时间：2021.9

StarOracle
去中心化预言机
时间：2021.10

Bridge
数字资产跨链服务
时间：2021.10

结束

- 招聘
  - 对Move感兴趣
  - 对布道感兴趣

Q & A

麦思博(msup)有限公司是一家面向技术型企业的培训咨询机构，携手2000余位中外客座导师，服务于技术团队的能力提升、软件工程效能和产品创新迭代，超过3000余家企业续约学习，是科技领域占有率第1的客座导师品牌，msup以整合全球领先经验实践为己任，为中国产业快速发展提供智库。

高可用架构主要关注互联网架构及高可用、可扩展及高性能领域的知识传播。订阅用户覆盖主流互联网及软件领域系统架构技术从业人员。 高可用架构系列社群是一个社区组织，其精神是“分享+交流”，提倡社区的人人参与，同时从社区获得高质量的内容。