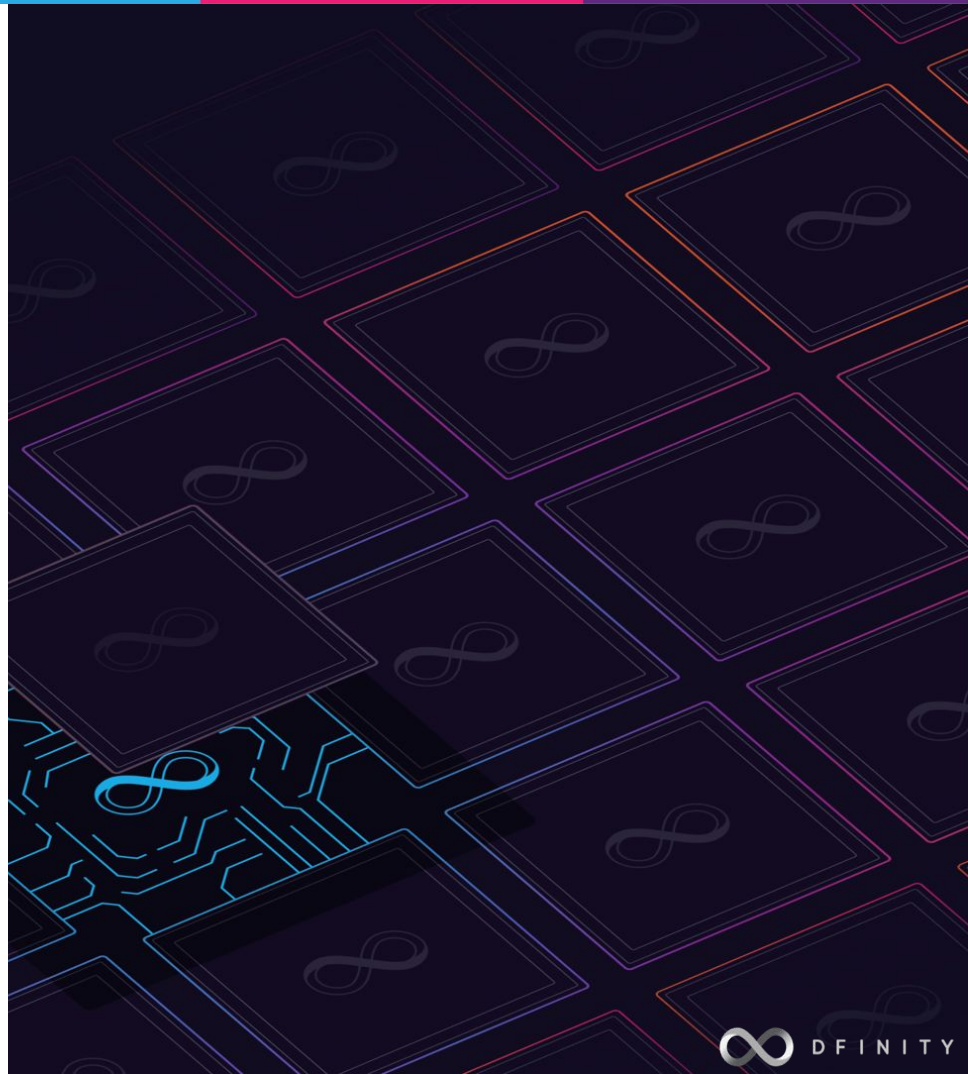
ICP区块链开发进阶课程

4. 整合 ICP 系统服务

主讲：Paul Liu - DFINITY 工程师

课程大纲

1. Motoko 语言进阶
2. Canister 开发进阶 I
3. Canister 开发进阶 II
4. 整合 ICP 系统服务
5. 项目实例分析

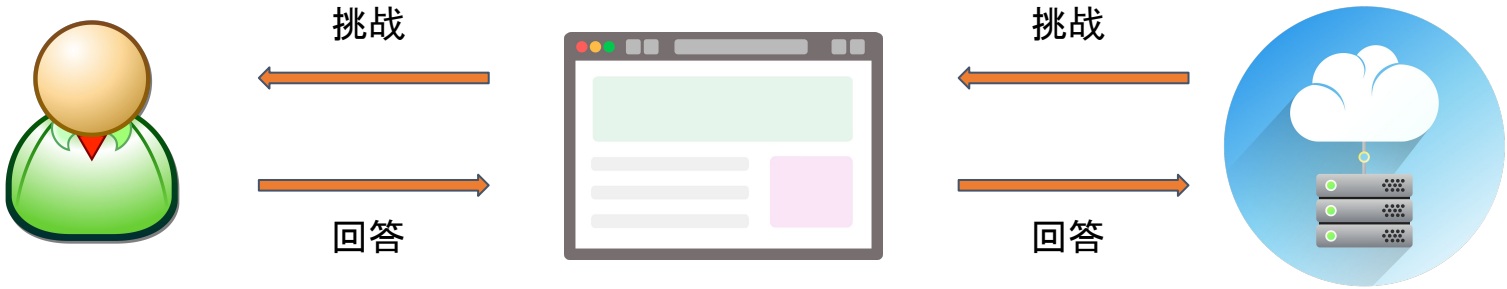



DFINITY
Internet Identity

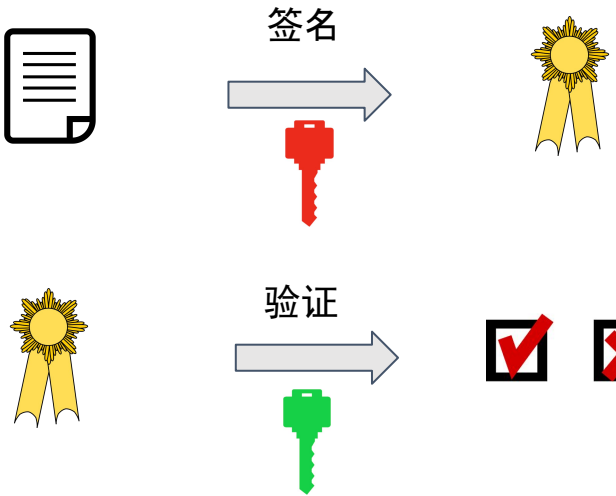
身份与权限

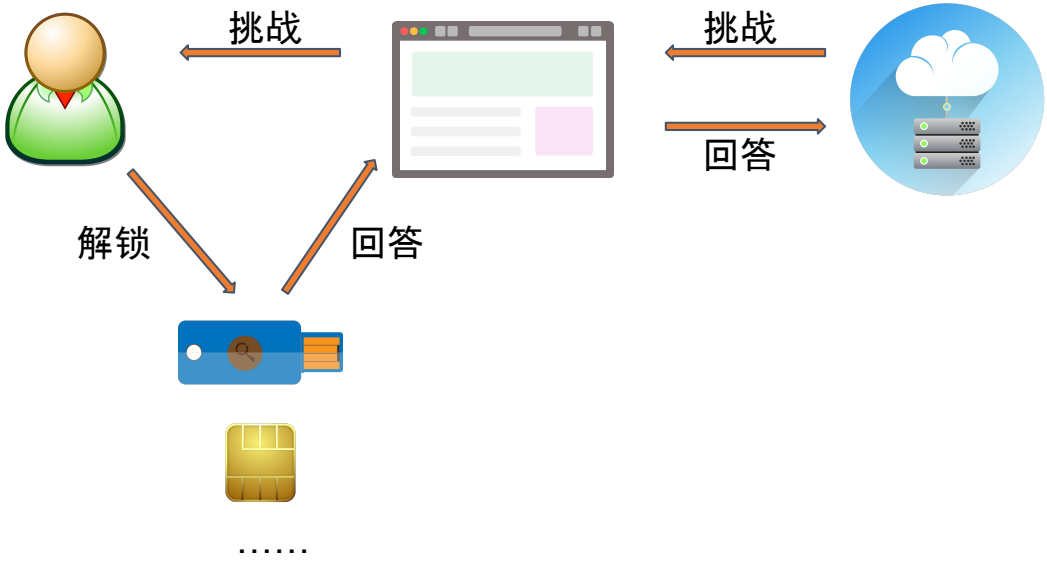
验证身份的方式

- What you know - 密码
- What you have - 钥匙
- What you are - 生物特征

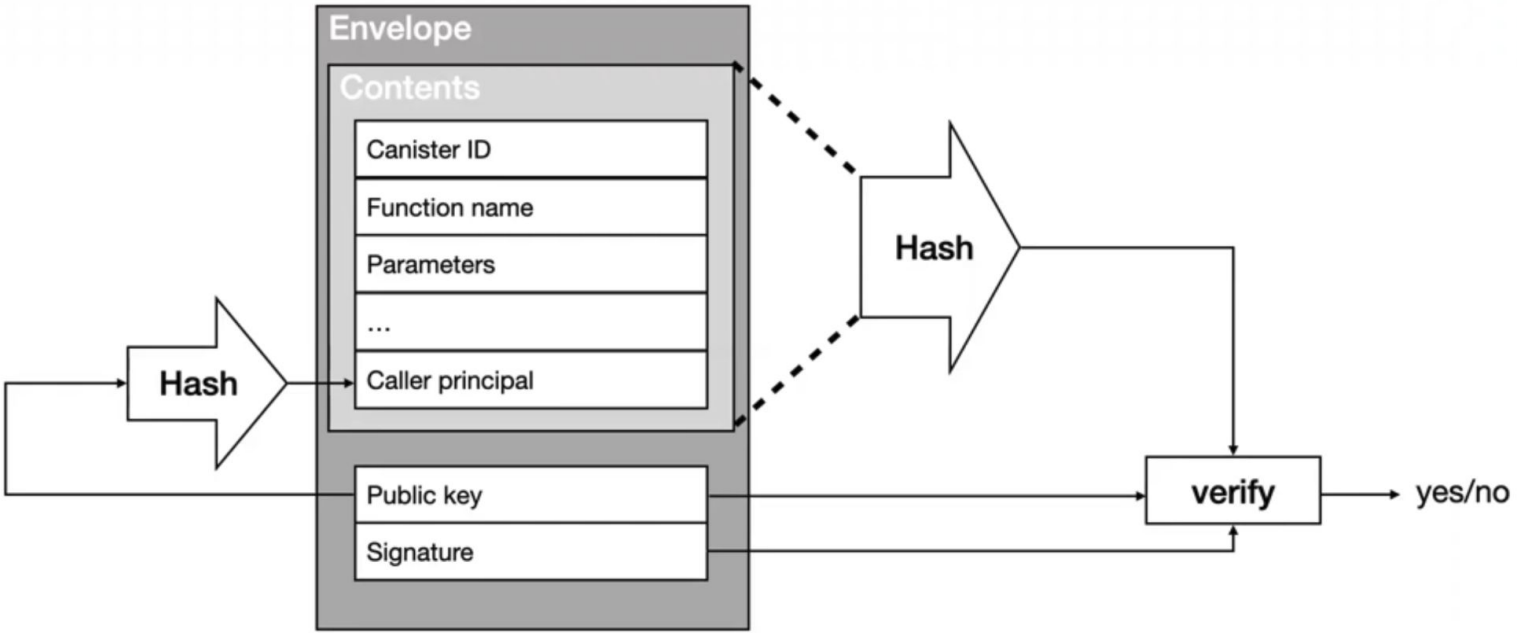


密码学数字签名

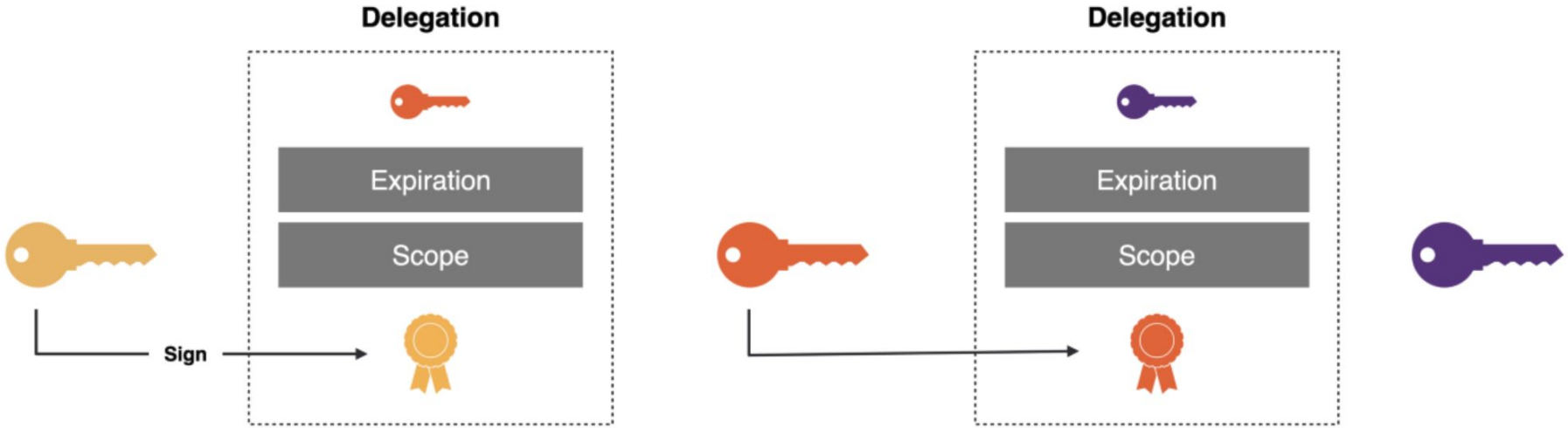




IC 用户签发消息

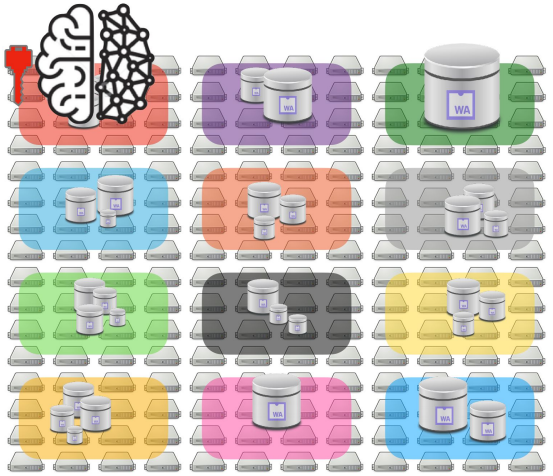

Envelope
Contents
Canister ID
Function name
Parameters
...
Caller principal
Hash
Hash
Public key
Signature
verify
yes/no

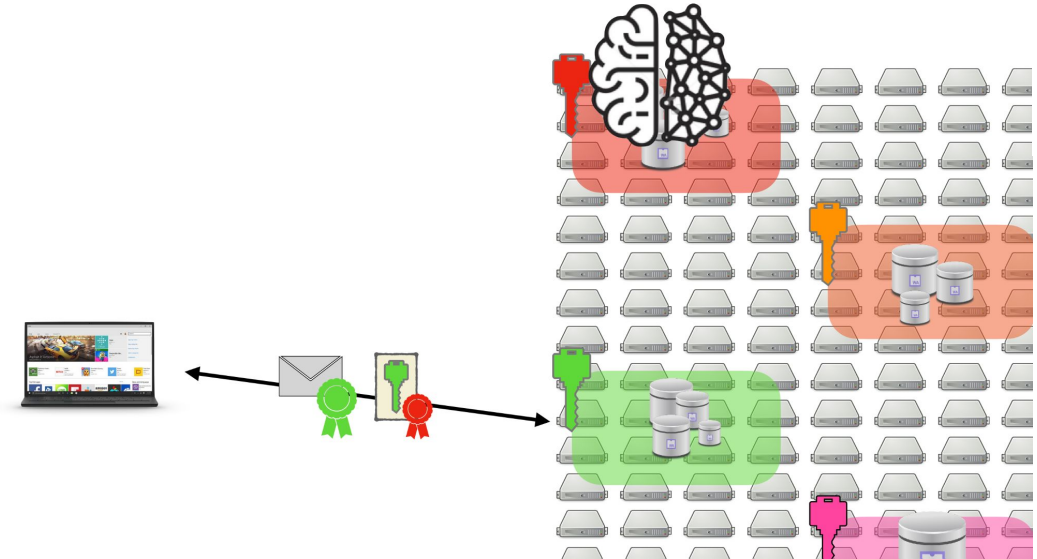
通过签名来授权使用 Principal



Canister 签名/证书/授权

- 所有的计算结果组成 Merkle Tree
- 子网对 Merkle Hash 签名
- Canister/用户/子网，可以通过签名相互授权



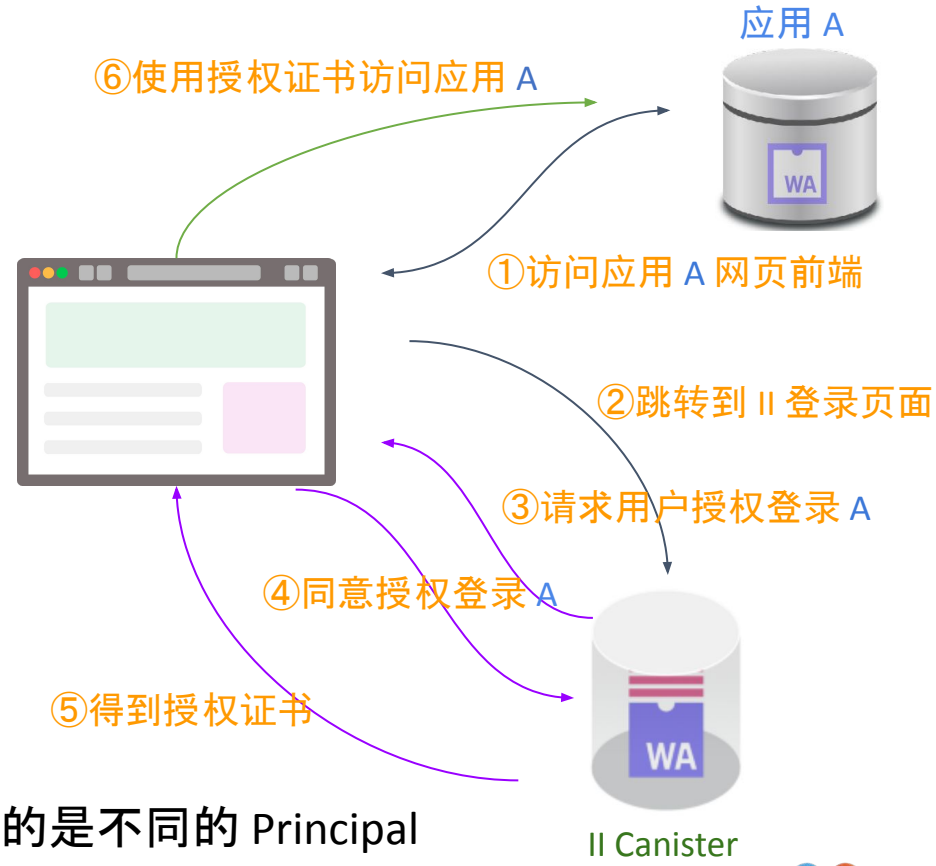


Internet Identity 解决方案

- 更安全的私钥管理
- 多设备登录
- 保护用户身份的私密性
- 身份 ≠ 私钥 (丢失可以替换)

用户使用 II 登录其它应用

- 生成随机的 session key
- 得到 II Canister 的授权证书
- 使用 II Canister 赋予的 Principal
- 同一个 II 账号在不同的应用中对应的是不同的 Principal



在前端集成 Internet Identity 登录服务

```
import { Actor, HttpAgent } from "@dfinity/agent";
import { AuthClient } from "@dfinity/auth-client";

const init = async () => {
  const authClient = await AuthClient.create();
  if (await authClient.isAuthenticated()) {
    handleAuthenticated(authClient);
  }
  renderIndex();

  const loginButton = document.getElementById(
    "loginButton"
  ) as HTMLButtonElement;
  loginButton.onclick = async () => {
    await authClient.login({
      onSuccess: async () => {
        handleAuthenticated(authClient);
      },
    });
  };
};
```

```
async function handleAuthenticated(authClient: AuthClient) {
  const identity = await authClient.getIdentity();

  const agent = new HttpAgent({ identity });
  console.log(process.env.CANISTER_ID);
  const whoami_actor = Actor.createActor<_SERVICE>(idlFactory, {
    agent,
    canisterId: process.env.CANISTER_ID as string,
  });
  renderLoggedIn(whoami_actor, authClient);
}

init();
```

使用 AuthClient 的示例：
https://github.com/krpeacock/auth-client-demo

DFINITY

ICP Ledger

什么是 Ledger Canister

- ICP 的铸造、转账、销毁
- 完全由 Canister 实现，属于应用层
- 交易记录以区块链形式保存，方便查询和验证
- 两种接口：Candid（链上）和 Rosetta API（链下，需要另外跑转接器）

Ledger 接口

```
service : {
  transfer : (TransferArgs) -> (TransferResult);
  account_balance : (AccountBalanceArgs) -> (Tokens) query;
  transfer_fee : (TransferFeeArg) -> (TransferFee) query;
  query_blocks : (GetBlocksArgs) -> (QueryBlocksResponse) query;
  symbol : () -> (record { symbol: text }) query;
  name : () -> (record { name: text }) query;
  decimals : () -> (record { decimals: nat32 }) query;
  archives : () -> (Archives) query;
}

type TransferResult = variant { Ok : BlockIndex;
                                Err : TransferError };

type AccountBalanceArgs = record { account: AccountIdentifier };
```

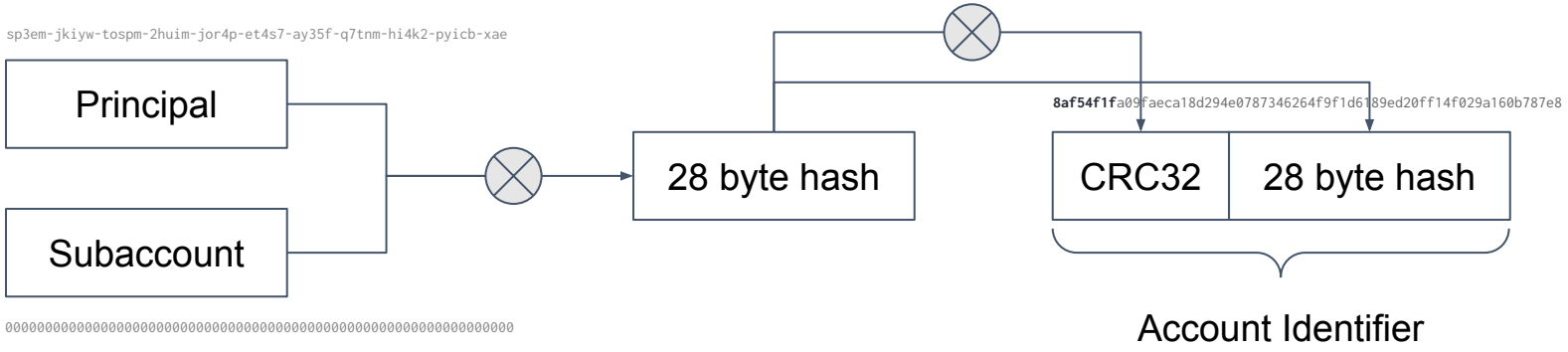
```
type Tokens = record { e8s : nat64 };

type TransferArgs = record {
      memo: Memo;
      fee: Tokens;
      from_subaccount: opt SubAccount;
      to: AccountIdentifier;
      created_at_time: opt TimeStamp;
};

// 32-byte array. The first 4 bytes is big-endian
// encoding of a CRC32 checksum.
type AccountIdentifier = blob;

// 32-byte byte array.
type SubAccount = blob;
```

https://github.com/dfinity/ic/tree/master/rs/rosetta-api/ledger_canister

DFINITY

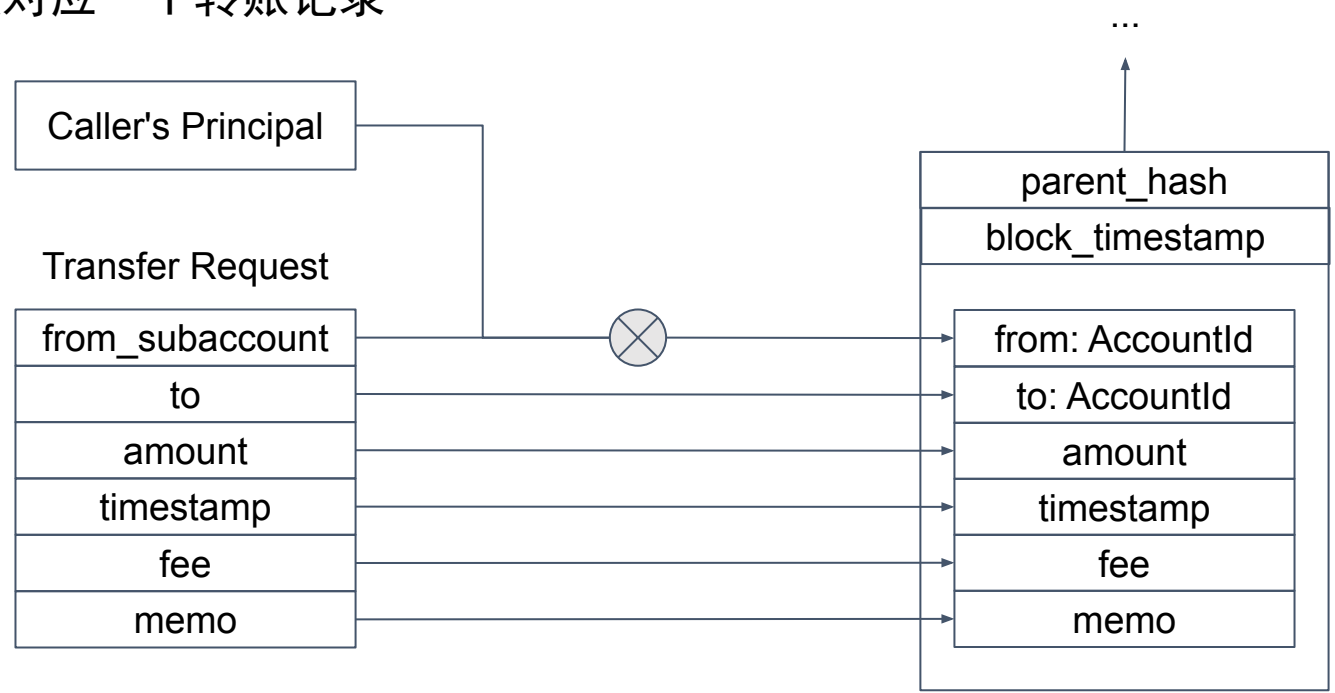
账户 ID 的生成机制

一个 Principal 可以控制多个账户

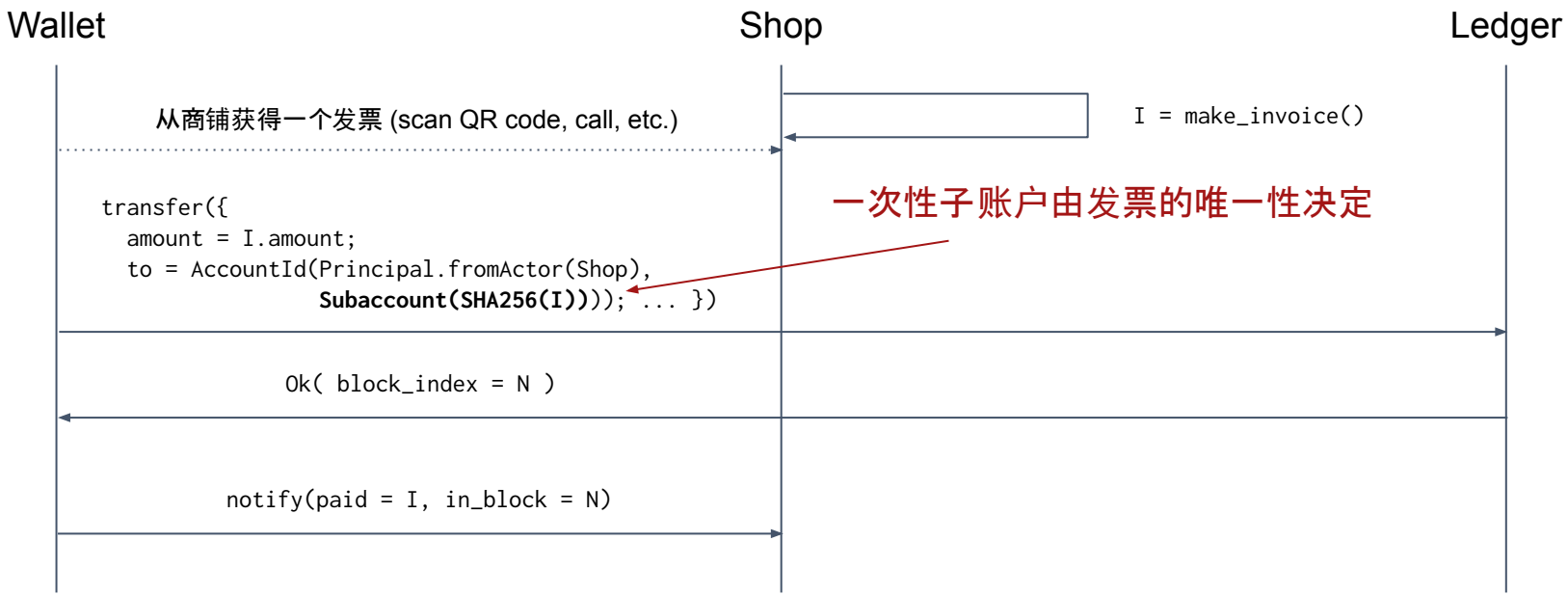


转账记录的区块

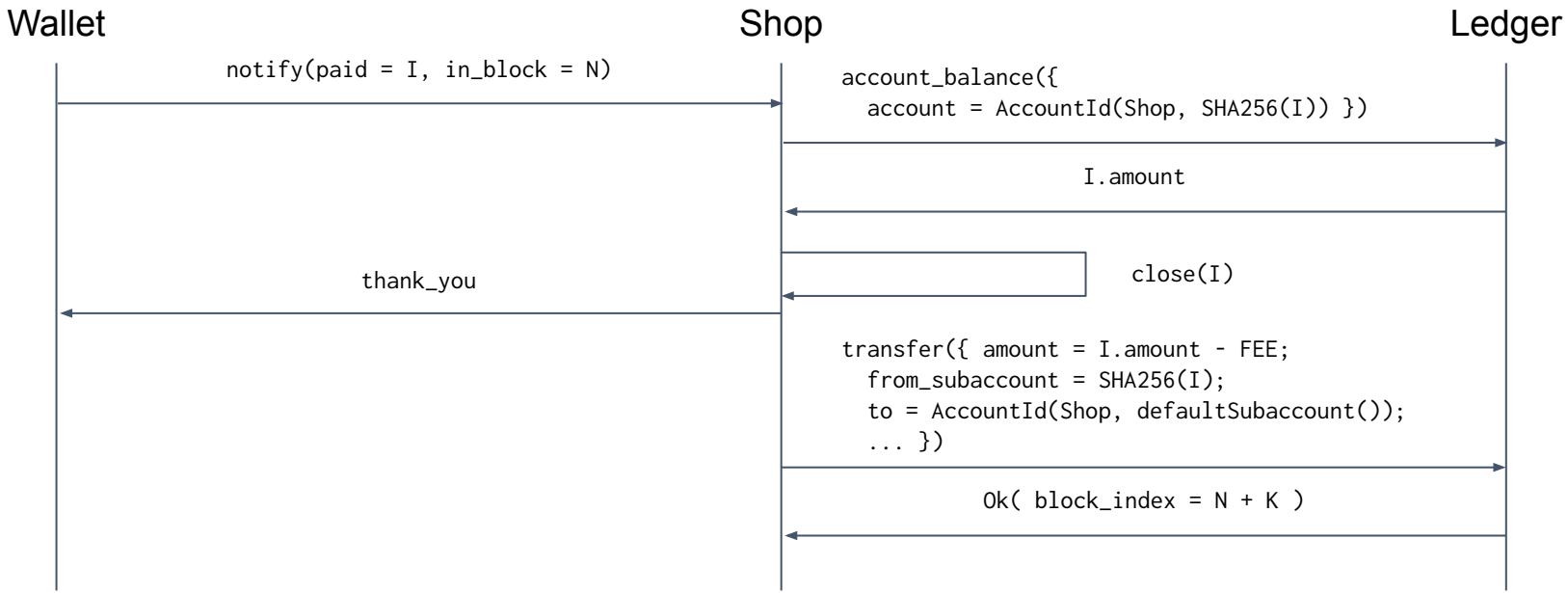
每个区块对应一个转账记录



转帐模式:使用一次性账户



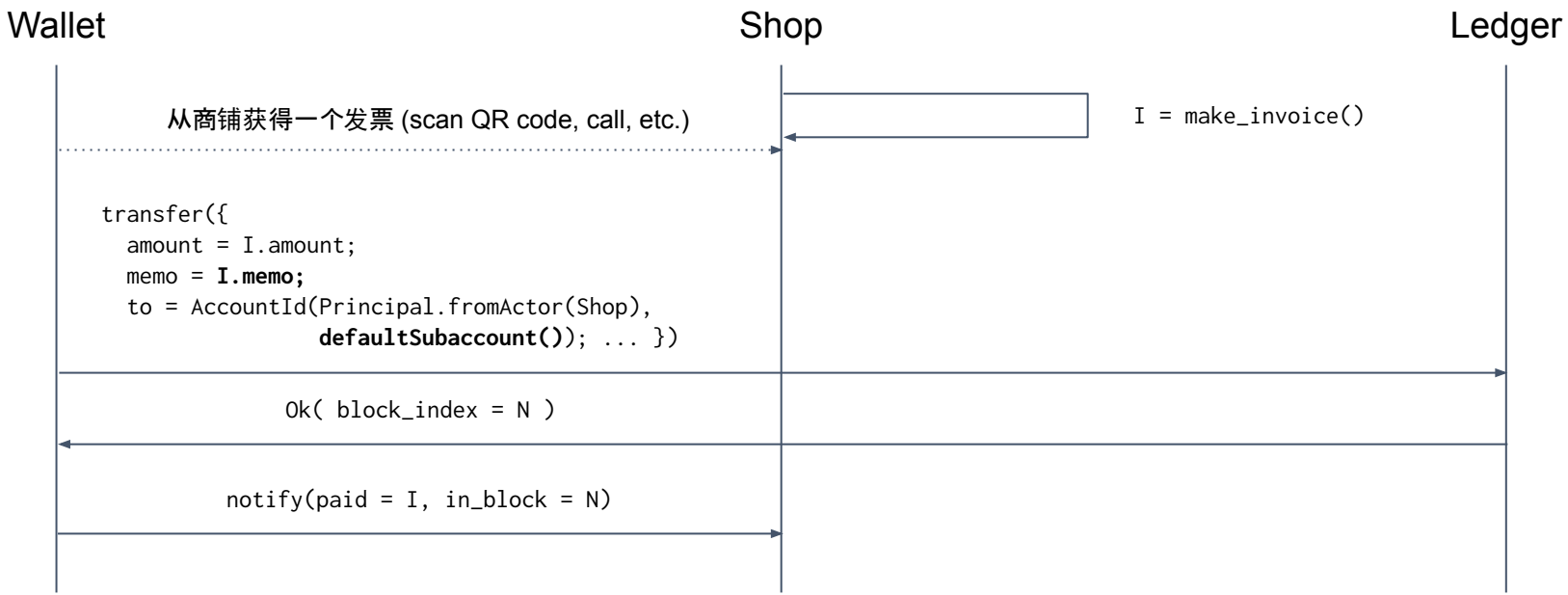
转帐模式:使用一次性账户(续)



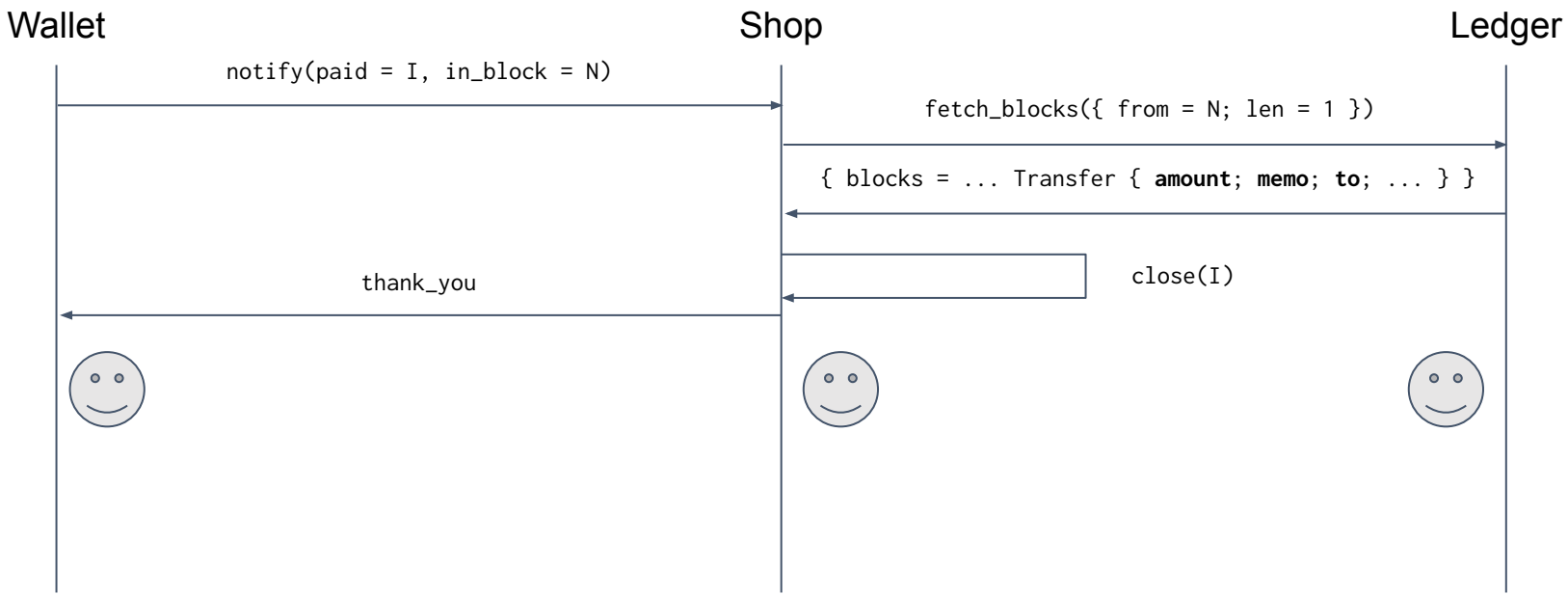
转帐模式:使用一次性账户(小结)

- 用户钱包不需要知道子账户逻辑，商铺可以直接将目标账户放在发票里面。
- 子账户只需要保持唯一性。可以不使用 SHA256，用计数器也可以。
- 商铺通过调用 account_balance 来确定转账成功。

转帐模式:使用账户关联信息(memo)



转帐模式:使用账户关联信息(续)



转帐模式:使用账户关联信息(续)

- 用户钱包与商铺之间的交互流程，在两种转账模式中是一致的。
  替换转账的实现方法不影响客户端的使用。
- 使用 memo 的模式只需要一次转账。
- 使用 memo 的模式需要商铺保留 memo 对应的发票记录。
- 商铺通过调用 fetch_blocks 来确认转账成功。

课程作业

实现一个简单的多人 Cycle 钱包：

1. 团队 N 个成员，每个人都可以用它控制和安装 canister。
2. 升级代码需要 M/N 成员同意。

要求：

- 只有提案通过后才执行对应的操作
- 简单的前端界面，允许查看当前的提案，canister 列表，小组成员
- 在前端整合 Internet Identity 登录

下一节：示例分析