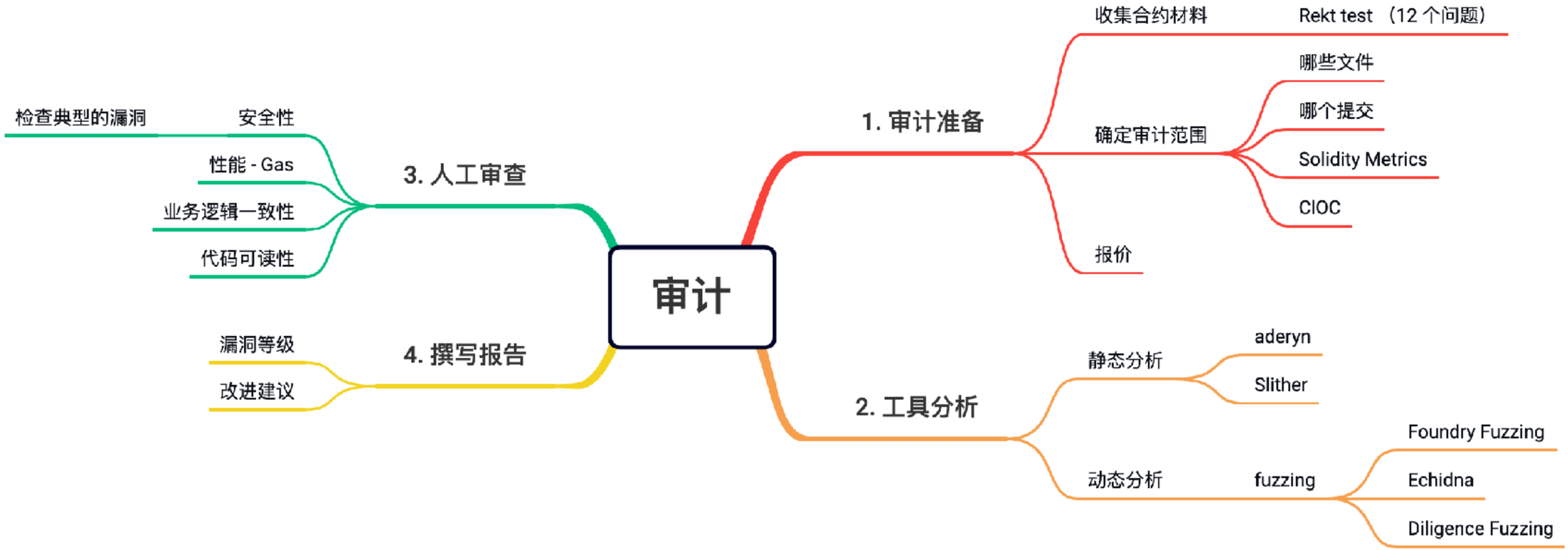
Solidity 安全

安全是一个过程，而非一个产品。它是一种思维方式，必须贯穿软件开发过程的方方面面

安全全过程

- 部署前：文档、编码、测试、审计
- 运行中：监控
- 事故分析
  - 分析资金流，尝试定位
  - 迁移

合约审计


审计
1. 审计准备
收集合约材料
Rekt test （12 个问题）
确定审计范围
哪些文件
哪个提交
Solidity Metrics
CIOC
报价
2. 工具分析
静态分析
aderyn
Slither
动态分析
fuzzing
Foundry Fuzzing
Echidna
Diligence Fuzzing
3. 人工审查
检查典型的漏洞
安全性
性能 - Gas
业务逻辑一致性
代码可读性
4. 撰写报告
漏洞等级
改进建议

审计准备

- 完整的工程、详细的文档、完备的测试
  - Rekt Test: https://blog.trailofbits.com/2023/08/14/can-you-pass-the-rekt-test/
- 工具(初评代码): 初步了解代码量、代码结构
  - CLOC： https://github.com/AlDanial/cloc
  - Solidity Metrics (VS Code extensions)： 直接生成分析报告
    - https://marketplace.visualstudio.com/items?itemName=tintinweb.solidity-metrics
  - Solidity Visual developer (VS Code extensions)： 分析调用关系
    - https://marketplace.visualstudio.com/items?itemName=tintinweb.solidity-visual-auditor

静态分析

- 在不执行程序代码的情况下，分析程序的源代码、AST、中间表示或二进制代码来发现潜在的漏洞。
  - 控制流分析：分析程序所有可能的执行路径
  - 数据流分析：跟踪变量、如：检查可能的溢出
  - 规则检测：根据预定义的规则集检查代码,
  - …

静态分析

- Slither : https://github.com/crytic/slither
  - 开源、灵活强大、可自定义检查器、生态活跃
  - 误报有点多
- Aderyn： https://github.com/Cyfrin/aderyn
  - 精确性更好、新秀、傻瓜使用

动态分析

- 通过模拟环境来运行代码，观察各种输入下的表现，来发现潜在的漏洞或性能问题。
- 方法：模糊测试 (Fuzzing) - 基于属性的测试方法 、代码覆盖率分析
- 模糊测试 , 优秀的 Fuzzer 很关键
  - Foundry Fuzzing
  - Echidna
  - Diligence Fuzzing （云平台）: https://fuzzing-docs.diligence.tools/

人工审查

- 一个优秀的合约审计工程师：
  - 熟悉EVM特性、常见协议、常见安全问题
    - https://github.com/ZhangZhuoSJTU/Web3Bugs
  - 跟踪安全事件、了解新攻击点
    - https://substack.com/@blockthreat
    - https://solodit.xyz/
    - https://rekt.news/

审计报告

- 模板
  - https://github.com/Cyfrin/audit-report-templating

运行中：监控

- 及时知晓问题（消息通知）及处理问题（自动化处理）
- 监控大额资金的变化
- 监控权限的转移
- 监控关键参数的修改
- …

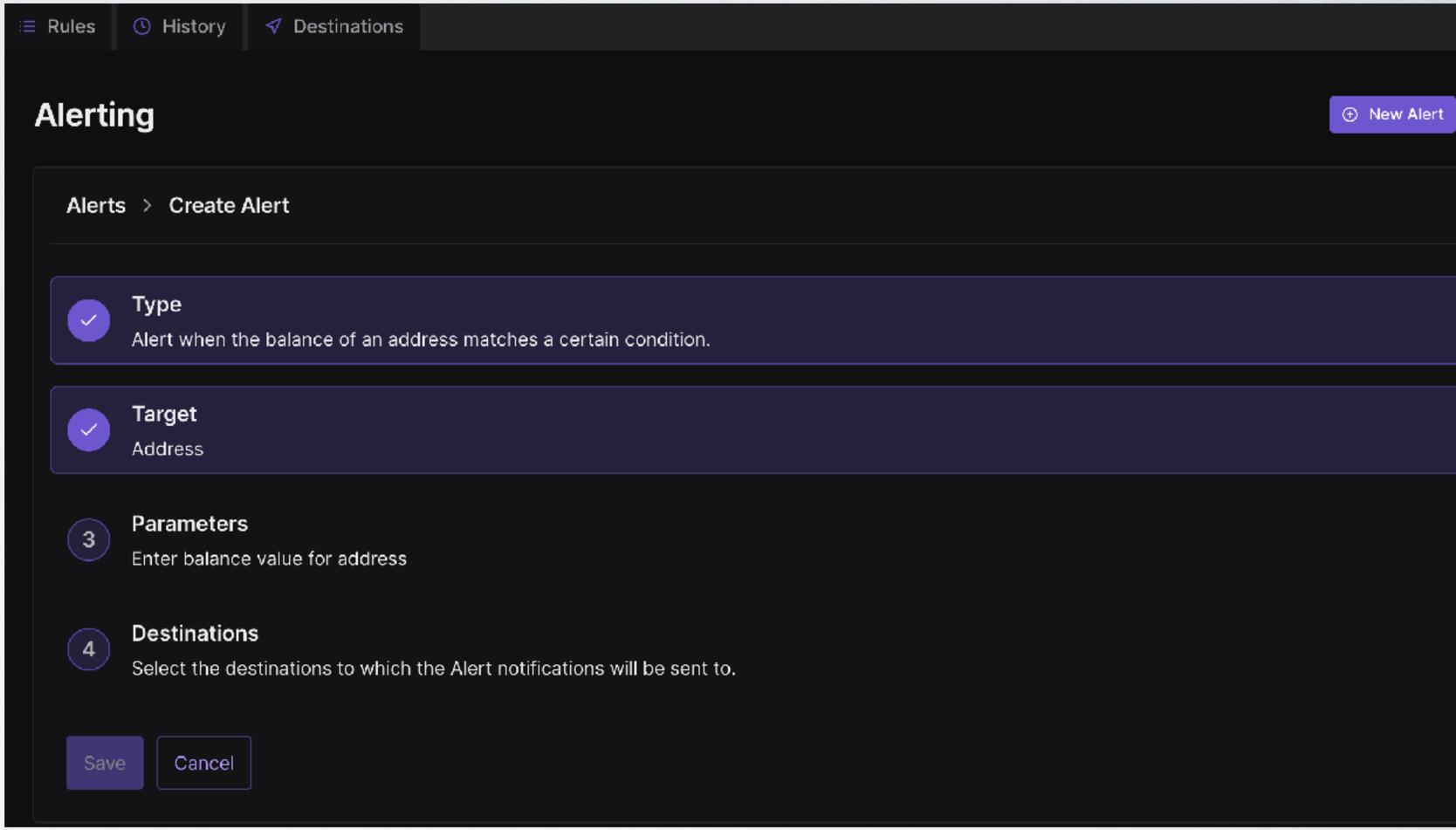
自动监控可以促进快速响应，越快响应，损害越小

运行中：监控

- 自建监控
- 很多第三方节点服务提供 WebHook 服务
- Tenderly Monitoring(Alerts)： https://tenderly.co/monitoring
- Alchemy: https://www.alchemy.com/webhooks
- QuickNode: https://www.quicknode.com/webhooks
- Forta （Detection Bots）： https://app.forta.network/bots

TENDERLY ALERTING

- 三个组件：
- Triggers： 12 种特定链上事件
- Targets： 感兴趣的监控地址
- Destinations：警报发送到哪



https://learnblockchain.cn/article/20223

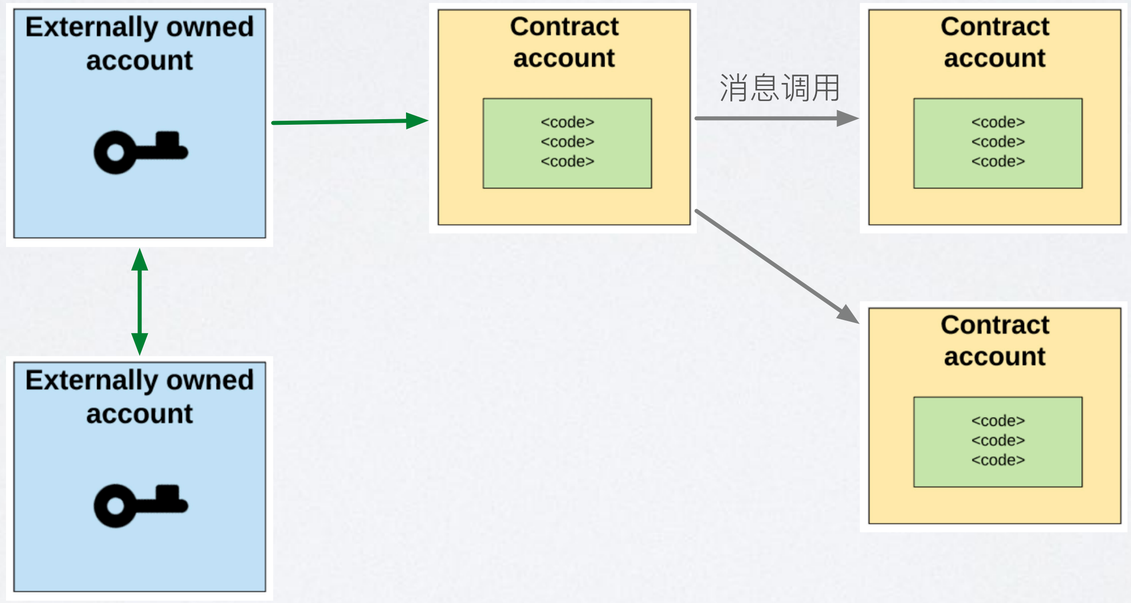
监控合约并自动化执行

应对：周期任务、紧急情况

合约只能被动响应

1. 由 EOA 支付Gas
2. 一次发起一笔交易

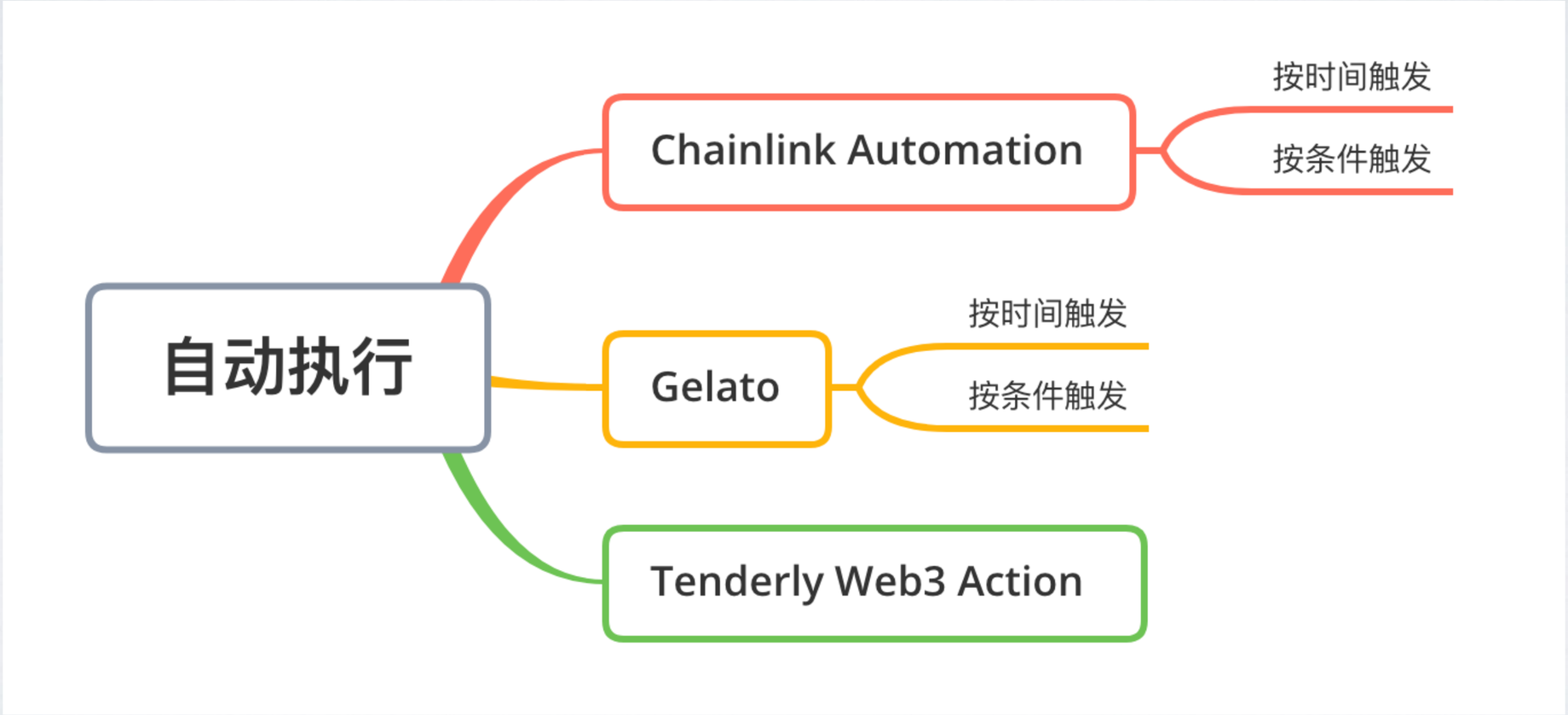
合约账户只能被动响应、没有自动运行



合约自动化执行

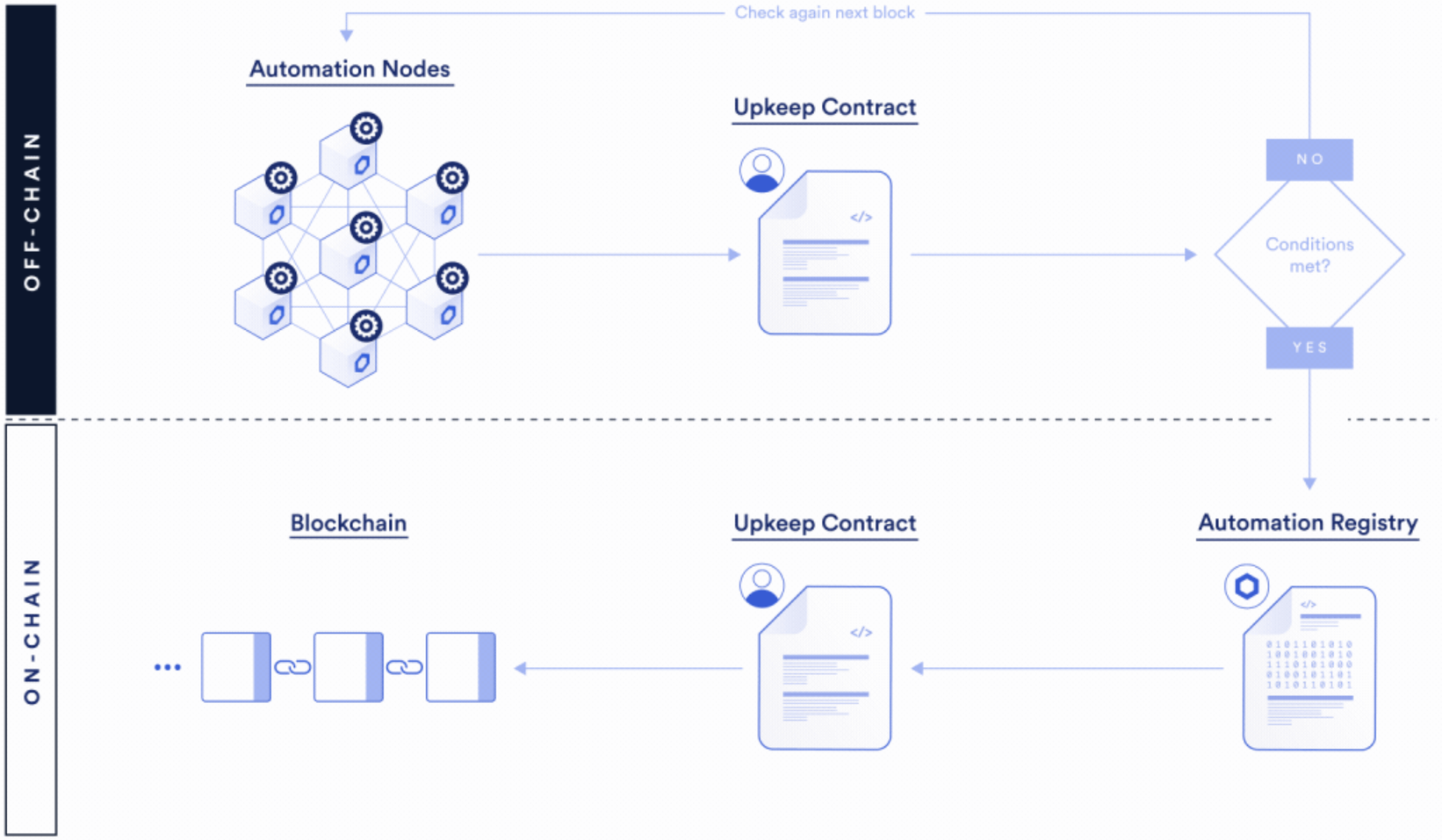
- 如何实现周期任务/定时任务/条件任务?
- 编写后端程序，常驻后端执行
- 主要问题：单点故障、热钱包泄漏

运行中：合约自动化执行


自动执行
Chainlink Automation
按时间触发
按条件触发
Gelato
按时间触发
按条件触发
Tenderly Web3 Action

CHAINLINK AUTOMATION 服务

运行动画：

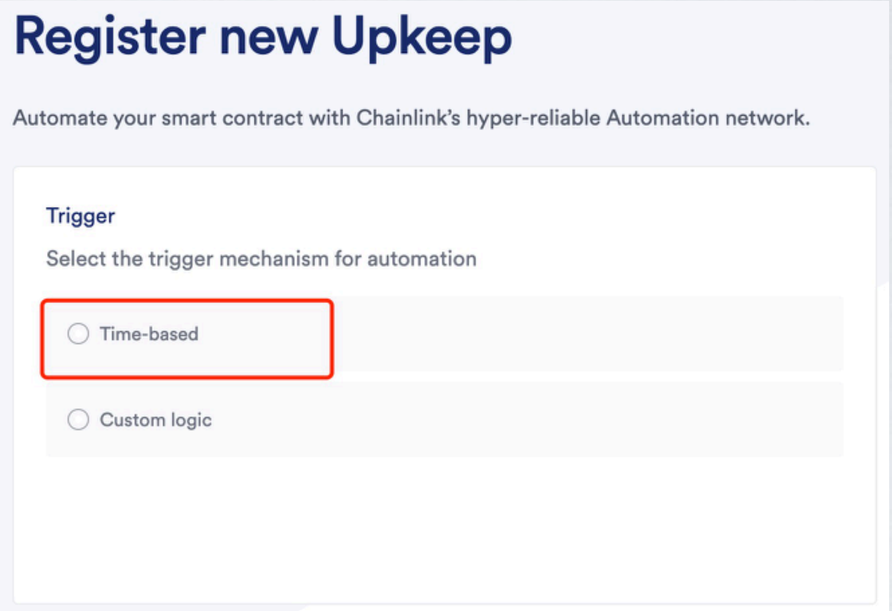


Chainlink Automation

- 超可靠和去中心化的自动化平台
- 支持三种方式自动执行合约函数
  - 根据时间， 类似设置定位任务的逻辑，定时执行自己的合约
  - 按条件执行，需编写 Upkeep 合约 (继承 AutomationCompatibleInterface.sol)， 参考文档
    - checkUpKeep() - 检查是否需要执行
    - performUpKeep() - 执行
  - 按日志出发执行， 实现 ILogAutomation.sol， 参考文档
    - checkLog() - 检查是否有匹配的日志
    - performUpKeep() - 执行

ChainLink Automation- 按时间执行

- 开发步骤：
- 在 https://automation.chain.link/ 注册
- 选择“Time-based”
- 填入要执行的合约地址
- 填入时间周期
- 选择“Custom-logic”
- 填入要执行的Upkeep合约地址



ChainLink Automation - 按条件执行

- 编写 UpKeep 合约处理进行逻辑判断及调用
  - checkUpkeep (判断条件)
  - performUpkeep(执行)

ChainLink Automation - 按条件执行

- 案例：当 Bank 存款大于 2 个 Token 时，自动转出

代码 : hello_foundry/AutoCollectUpKeep.sol

任务地址：https://automation.chain.link/sepolia/27157894517625986686394990837133401010366092253963320486987630466239869405442

Gelato Functions

- 按时间执行, 无需代码 （automated-transaction）
- 按链上条件执行 （Solidity Functions） ， checker 合约
- 按链下条件执行 （Typescript Functions）

事故分析

- Tenderly Debugger
- https://phalcon.blocksec.com/explorer
  - https://www.youtube.com/watch?v=eXeirKUy1XA
  - https://www.youtube.com/watch?v=uiqCrhIU0To
- Foundry Transaction Replay Trace / Debugger
  - cast run： 本地环境中运行一个已发布的交易，并打印出跟踪。

练习题

- 修改 Bank 合约：
  - 用户可以通过 deposit 进行存款（先Approve） — 之前已经实现
  - (自动化)当存款超过 x 时，转移一半的存款的到指定的地址，如 Owner.

可使用 ChainLink Automation 、Gelato 、

https://decert.me/quests/072fccb4-a976-4cf9-933c-c4ef14e0f6eb