

The Ideal



Visualizing the Invisible: The Async RPC Failure Simulator

Async RPC Failure Simulator – 非同期RPCにおける設計誤りの再現シミュレーター

Synchronous / Ordered

The Reality

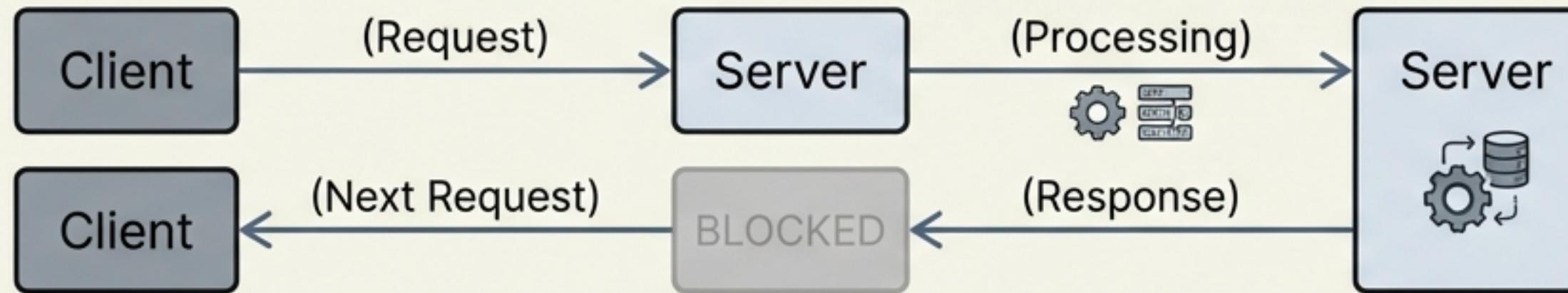


Asynchronous / Chaos

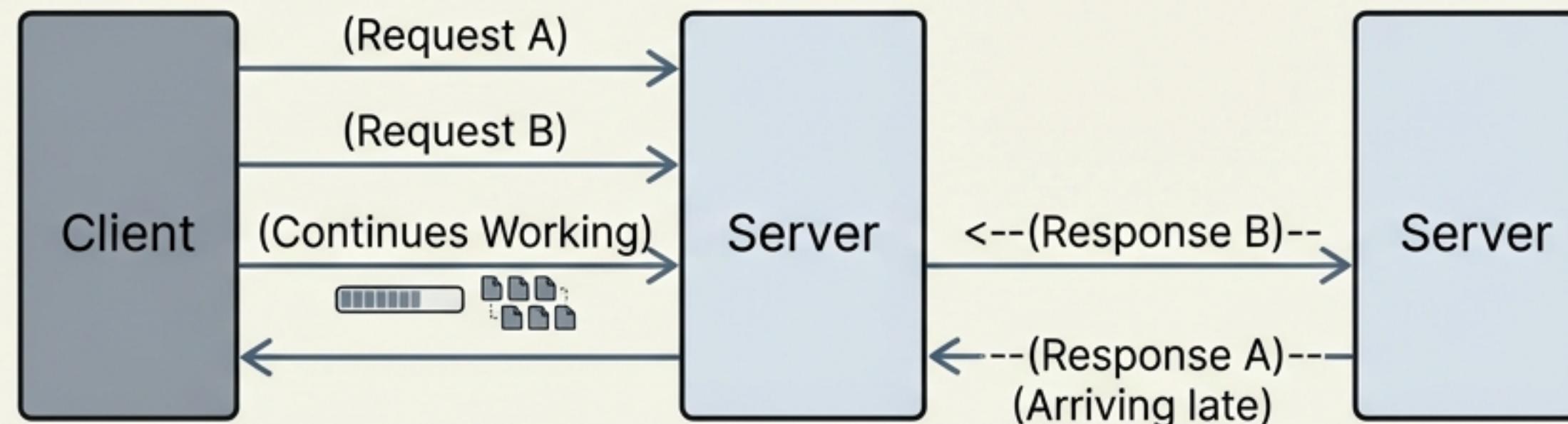
The Fragility of "Fire and Forget"

なぜ非同期RPCは難しいのか

Synchronous (Blocking)



Asynchronous (Non-Blocking)



問題点 (The Problem):

非同期RPC (Future/Promise) は強力ですが、リクエストとレスポンスの時軸がずれるため、管理が複雑化します。

重要な課題 (Key Challenge):

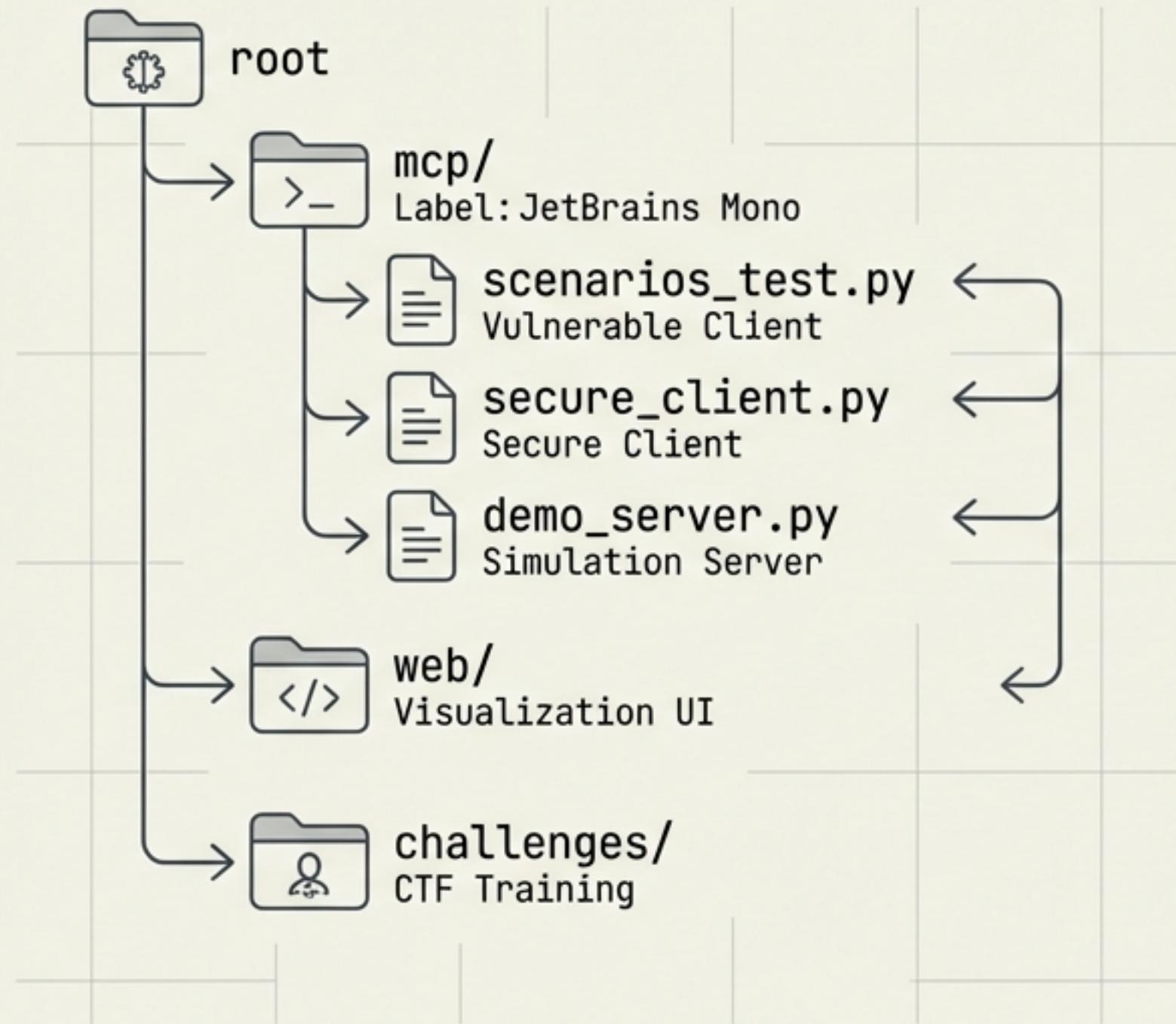
「Fire and forget (投げっぱなし)」は簡単ですが、「投げて、待って、タイムアウトして、正しく後始末する」ことは極めて困難です。

対象領域 (Target Domain):

Microservices, WebSockets, AI Agents (MCP Context)

Introducing the Laboratory

安全な失敗をシミュレートする実験環境



機能 (What it does):

- Request / Response の順不同 (Out-of-order execution)
- タイムアウト後の遅延レスポンス (Orphan responses)
- Pending台帳（待機リスト）の掃除と副作用

目的 (Goal):

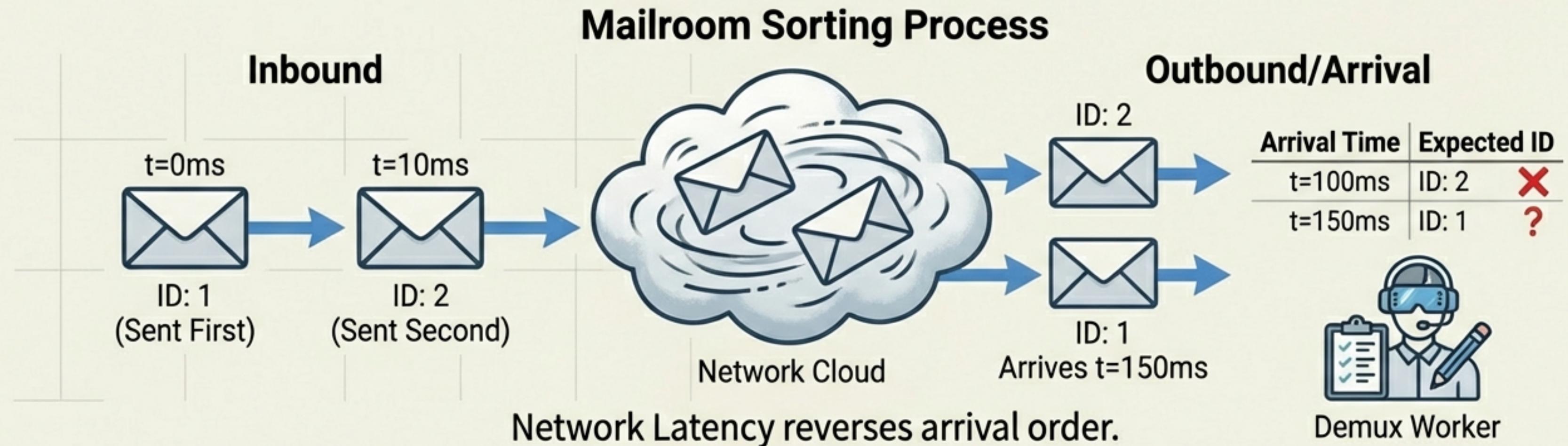
攻撃ツールではありません。設計ミスがセキュリティ事故につながる境界 (Boundary) を学ぶための教材です。

手法 (Method):

意図的に脆弱なクライアント (scenarios_test.py) と、対策済みのクライアント (secure_client.py) を比較実行し、挙動の差異を観測します。

Experiment 1: The Demux Problem

応答順序の逆転とID管理



シナリオ (The Scenario):

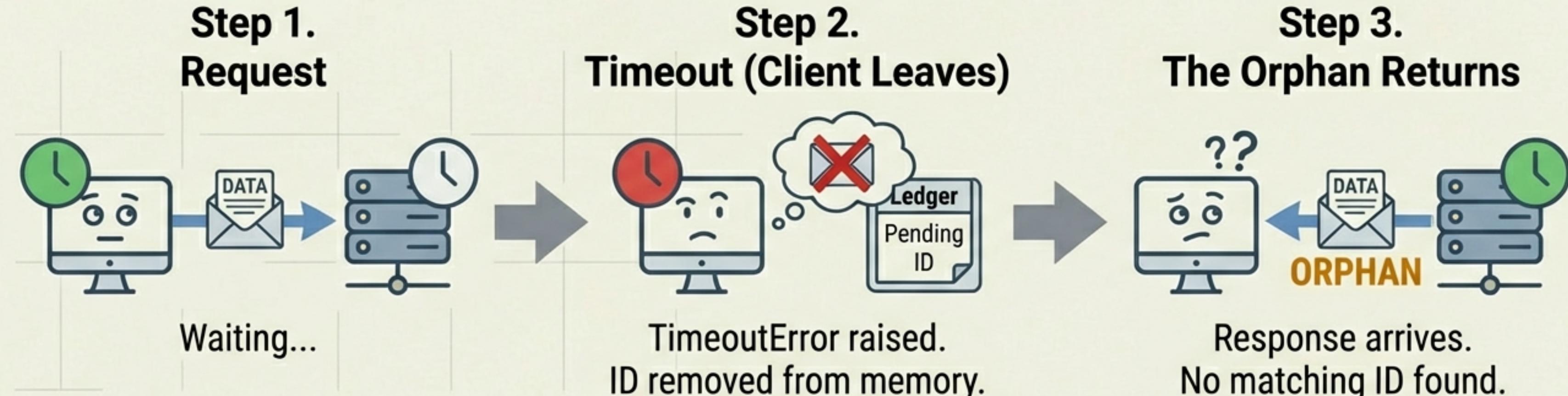
クライアントが複数のリクエスト (ID=1, ID=2) を連続して送信し、サーバーが逆順 (ID=2, ID=1) で応答を返すケース。

リスク (The Risk):

非同期処理では応答順序は保証されません。IDによる突き合わせ (Demultiplexing) が正しく実装されていないと、手紙を別人に渡すような「レスポンスの取り違え」が発生します。

Experiment 2: The Orphan Response

タイムアウト後の迷子レスポンス



定義 (Definition):

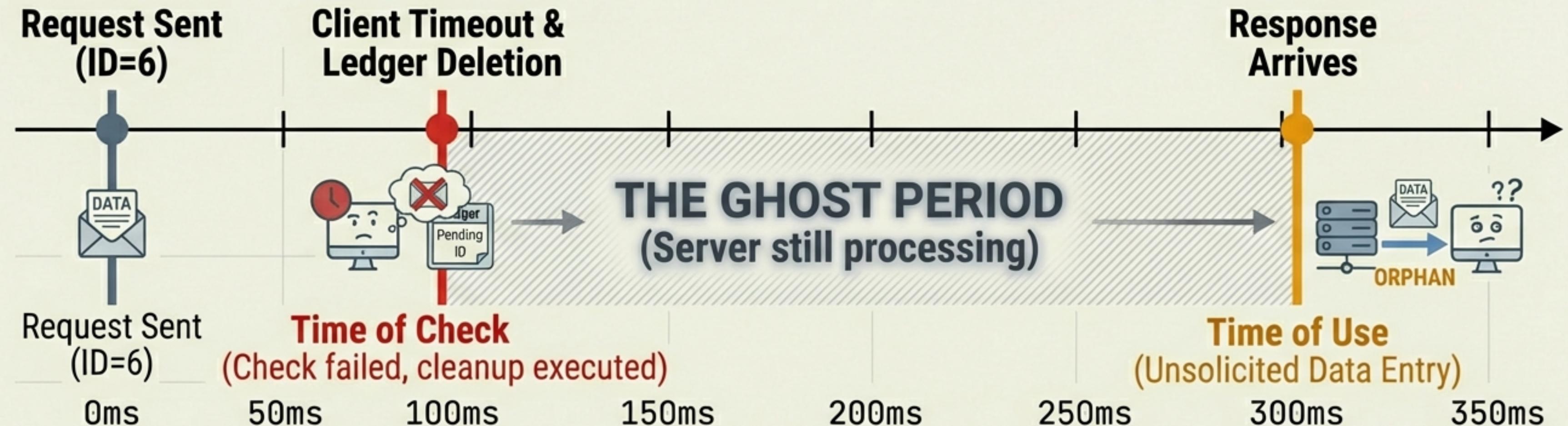
タイムアウト後に届いた「引き取り手のない」レスポンス。

結果 (The Result):

クライアントには対応するリクエストが存在しないため、このデータは「孤児 (Orphan)」となります。このデータの扱いがセキュリティの分かれ道です。

Anatomy of a Timing Bug (TOCTOU)

Time of Check to Time of Use - 危険な時間の隙間



インサイト (Key Insight):

サーバーはクライアントがタイムアウトしたことを知りません。

ギャップ (The Gap):

「pending台帳」からIDが消えた後にデータが届くため、システムはこのデータの扱い（保存するか？捨てるか？）をその場で判断しなければなりません。

Vulnerable Pattern: Storing the Ghosts

やってはいけない実装パターン



```
1 def _issue_id(self):
2     # Bad Practice: Predictable ID
3     self._next_id += 1 # ● Sequential ← VULNERABILITY
4     return self._next_id
5
6 def _reader_loop(self):
7     # ... receiving data ...
8     if fut is None:
9         # Bad Practice: Hoarding Orphans
10        self.orphan_responses.append(data) # ● Risk
```

**予測可能なID
(Sequential IDs):**
攻撃者が次のIDを予測し、
偽のレスポンスを注入可能
(Injection attacks)。

**Orphanの保存
(Hoarding Orphans):**
遅延したレスポンスを保存・
再利用することで、後続の無
関係なリクエストと取り違え
る原因になります
(Response Confusion)。

Secure Pattern: The Shield

堅牢な実装パターン



```
1 import secrets
2 def _issue_id(self):
3     # Secure: Cryptographically Strong Random
4     return secrets.token_hex(16) # ● Secure ← DEFENSE
5
6 def _reader_loop(self):
7     # ... receiving data ...
8     if fut is None:
9         # Secure: Log and Discard
10        log_security("WARN", f"Orphan: {id}")
11        # self.stats["orphans_discarded"] += 1
12        continue # ● Discard immediately ← DEFENSE
```

暗号論的乱数 (Random IDs):

128ビットの予測不可能なID (secrets.token_hex)により、偽装を防止します。

Orphanの破棄 (Discarding Orphans):

「待っていないデータ」は即座にログに残して破棄します。再利用は絶対にしません。

Vulnerable vs. Secure Implementation

設計方針の比較

	 Vulnerable (脆弱)	 Secure (堅牢)
Request ID Generation	連番 (Sequential) 1, 2, 3... → 予測可能 (Predictable)	暗号論的乱数 (CSPRNG) → 予測不可能 (Unpredictable)
Orphan Handling	Save to List (`append`) → 再利用リスク (Reuse Risk)	Discard & Log (`continue`) → 安全 (Safe)
Timeout Validation	No Check → DoS Risk	Min/Max Enforcement → Resource Protection
Error Logging	Standard Output	Separated Security Log (stderr)

From Simulation to Real-World Exploits

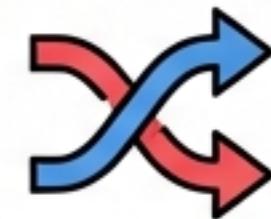
シミュレーターが再現する実際のCVE



Resource Exhaustion

HTTP/2 Rapid Reset
(CVE-2023-44487)

タイムアウト後もサーバー側で処理が継続し、リソースが枯渀する。Orphanの大量発生と同じ構造です。



Response Confusion

Apache mod_proxy Mix-up
(CVE-2020-11984)

バックエンド接続の管理ミスにより、別ユーザーのレスポンスが紛れ込む。Orphanの再利用に起因します。



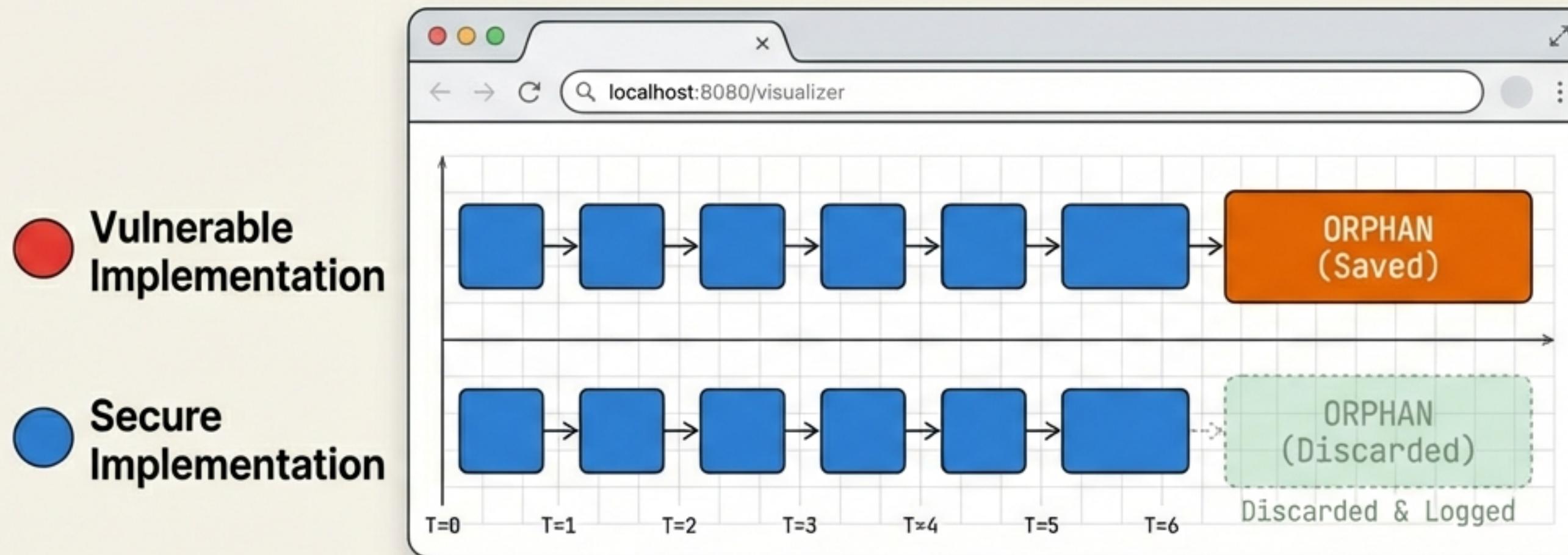
ID Prediction / Injection

CryptoNote Wallet
Vulnerability

予測可能なIDにより、攻撃者が偽のレスポンスを注入して資金を窃取。連番IDの危険性を示しています。

Seeing is Believing: The Web UI

可視化ツールによる直感的理解



Features:

- Orange Flow: 脆弱な実装。IDが連番で、Orphanが「保存」される様子。
- Green Flow: 堅牢な実装。IDがランダムで、Orphanが「破棄」される様子。

チームメンバーに非同期バグの危険性を説明するための教育用ツールです。

Challenge: Orphan Response Hijacking

実践演習 - CTF Mode

```
$ python exploit_template.py
[*] Step 1: Login as Guest... SUCCESS
[*] Step 2: Trigger Admin activity...
[Admin] Fetching secrets...
[*] Step 3: Waiting for Orphan...
[!] ORPHAN CAPTURED!
[*] Step 4: Extracting Flag...
[+] FLAG: FLAG{Orph4n_r3sp0ns3_h1j4ck3d_suc
c3ssfully}
```

- **Mission:**
`challenges/challenge1_orphan_hijack`
- **シナリオ:**
あなたはGuestユーザーです。タイミング攻撃を仕掛け、Adminユーザーの秘密情報(FLAG)を奪取してください。
- **学習目標:**
Orphan response を再利用することで、いかに簡単に「なりすまし」が可能かを体験的に学習します。

Security Best Practices Checklist

開発者が守るべき4つの原則

Use Cryptographic Random IDs

UUID v4 or CSPRNG (`secrets.token_hex`). Never use counters or timestamps.

Discard Orphans Immediately

Implement 'Log and Discard' policy for any unknown response ID. Never cache them.

Validate Timeout Ranges

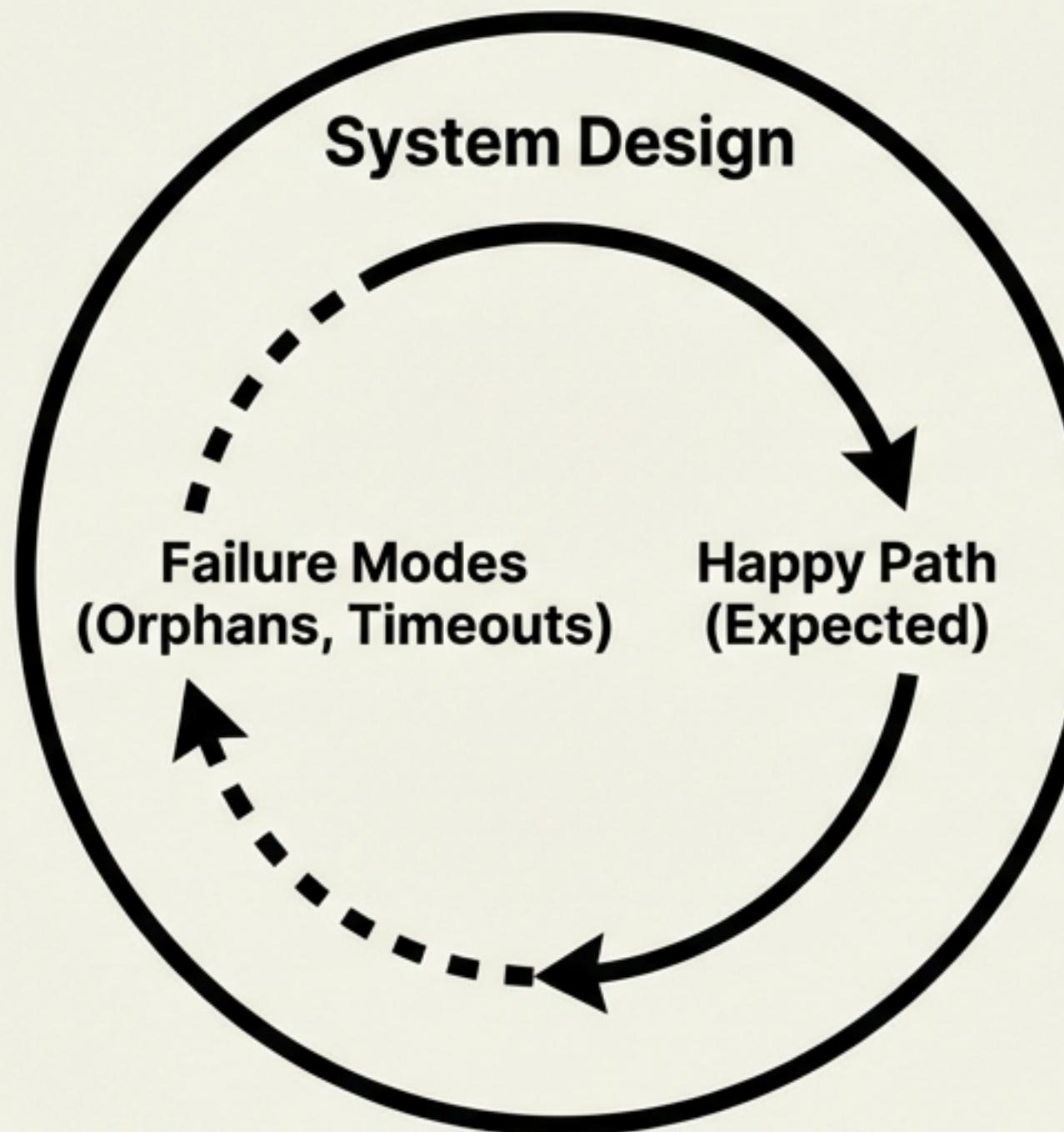
Enforce strict Min/Max timeout values to prevent Rapid Reset style attacks.

Separate Error Types

Explicitly distinguish between 'Protocol Errors' (retry) and 'Tool Errors' (don't retry).

Abnormalities are Normal

結論: バグではなく「性質」として扱う



Core Philosophy: Orphan responses are not bugs; they are an inherent property of async systems.

The Real Bug: The bug is not that they happen, but failing to handle them safely.

Conclusion: 「異常系」を「正常なフローの一部」として設計に組み込むことが、堅牢な非同期システムの条件です。

Start Experimenting

プロジェクトに参加する



[https://github.com/ipusiron/
async-rpc-failure-simulator](https://github.com/ipusiron/async-rpc-failure-simulator)

Project: Part of "生成AIで作るセキュリティツール200" (Day 112).

Action: Clone the repo, run `scenarios_test.py`, and witness the failure modes yourself.

Acknowledgments: Special thanks to ipusiron.