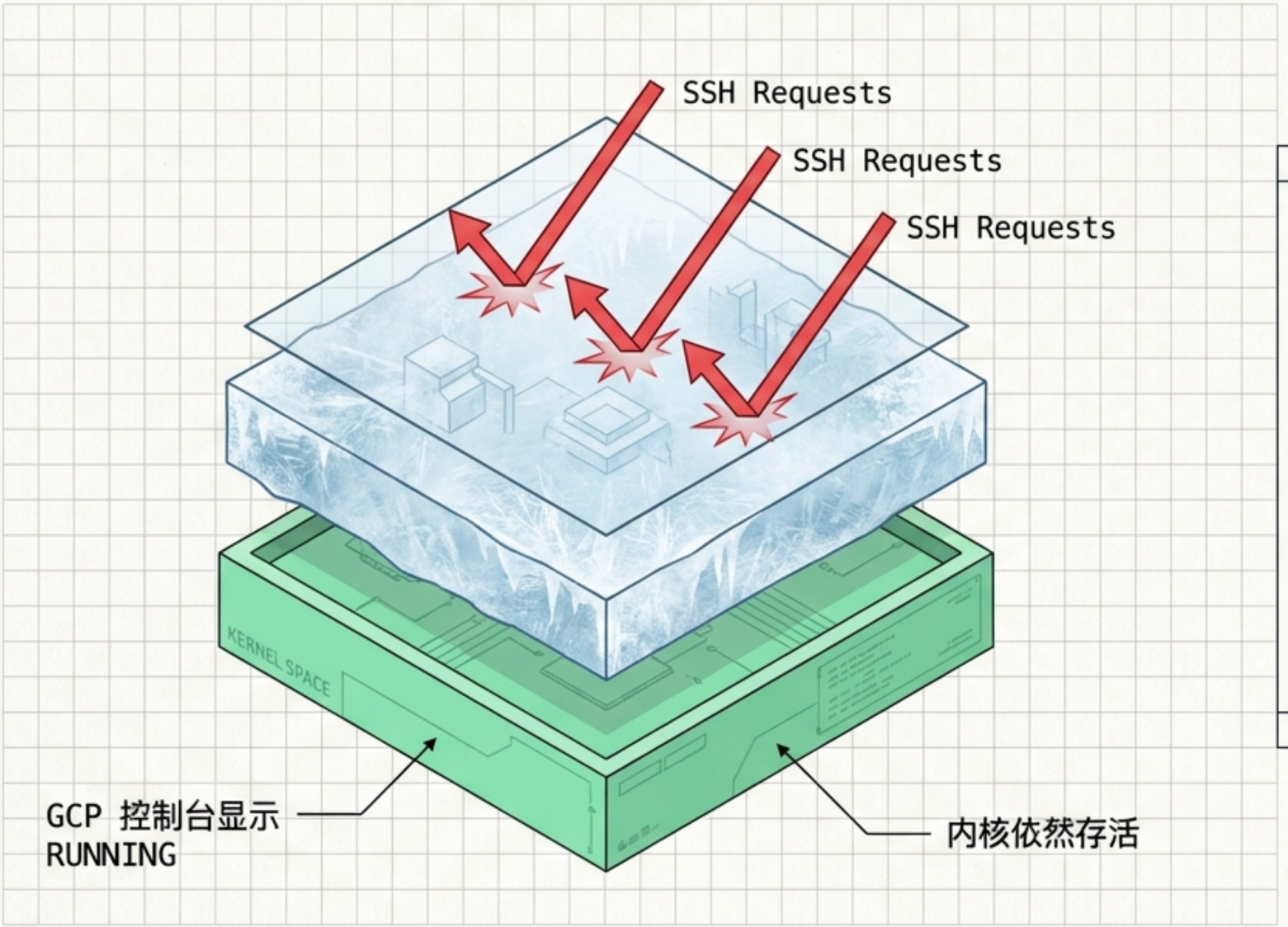
服务器又挂了：Cursor 的连环内存谋杀案

32GB GCP 开发机的二次死亡调查与法医级尸检报告

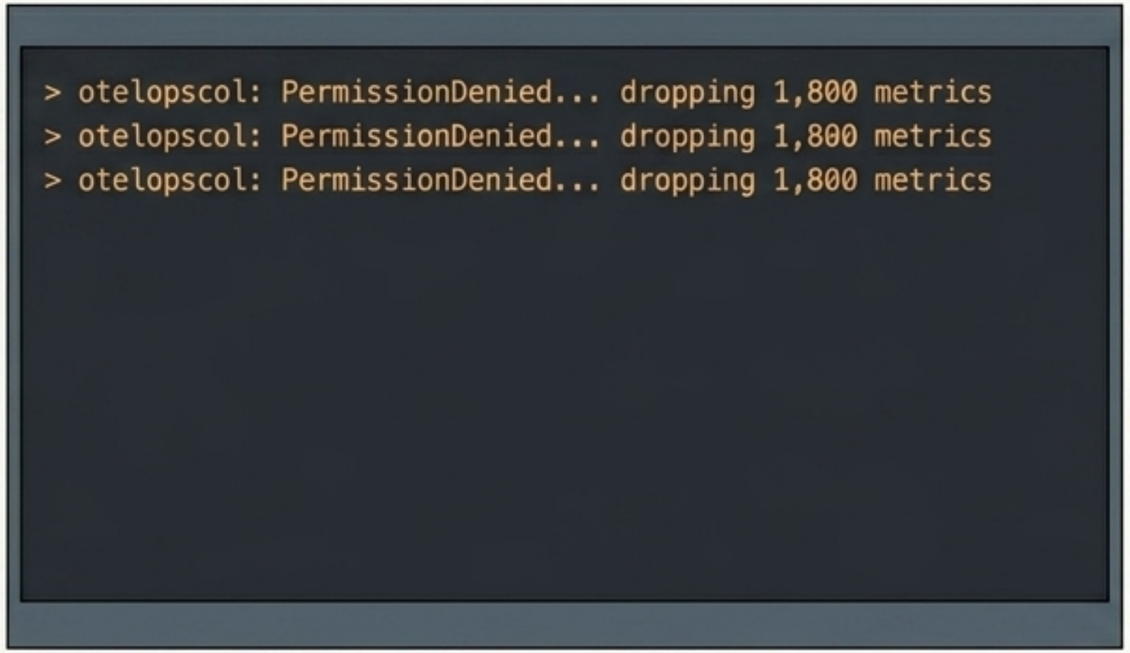
```
[CASE STATUS]: CLOSED
[ENVIRONMENT]: GCP e2-highmem-4 (32GB)
[UPTIME]: 15 HOURS BEFORE CRASH
```

薛定谔的服务器



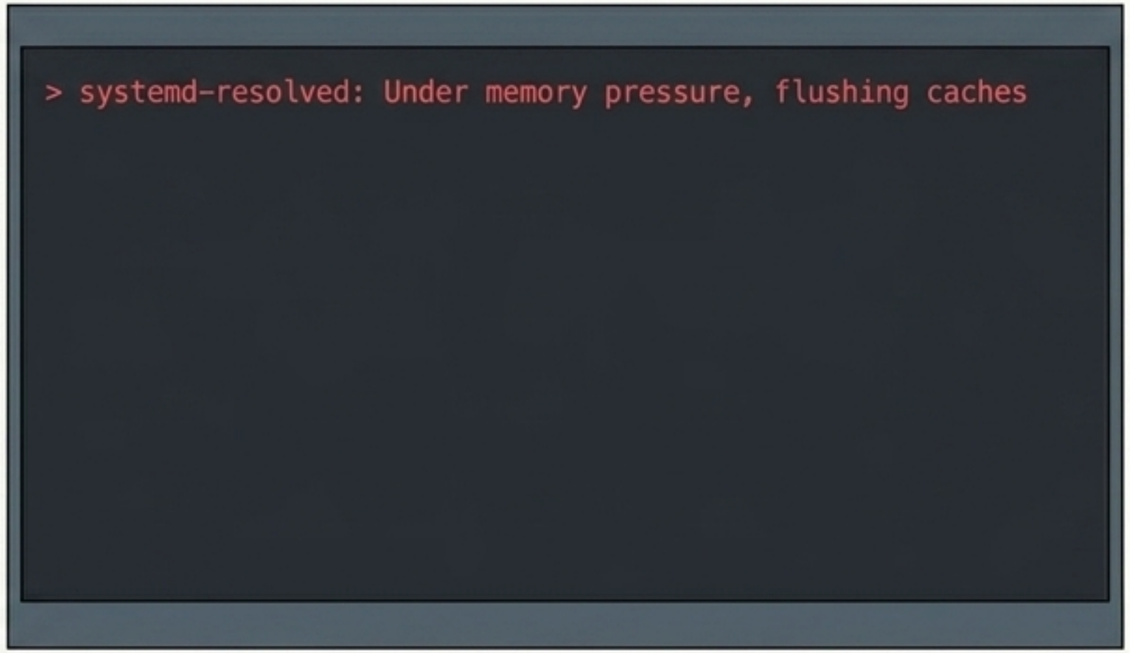
内核活着，但无人应答

症状重现：SSH 连接超时，死死卡在 banner exchange 阶段。就像第三篇里那台 16GB 机器的死亡回放，升级到 32GB 并没有阻止用户态的彻底冻结。



监控代理的反噬

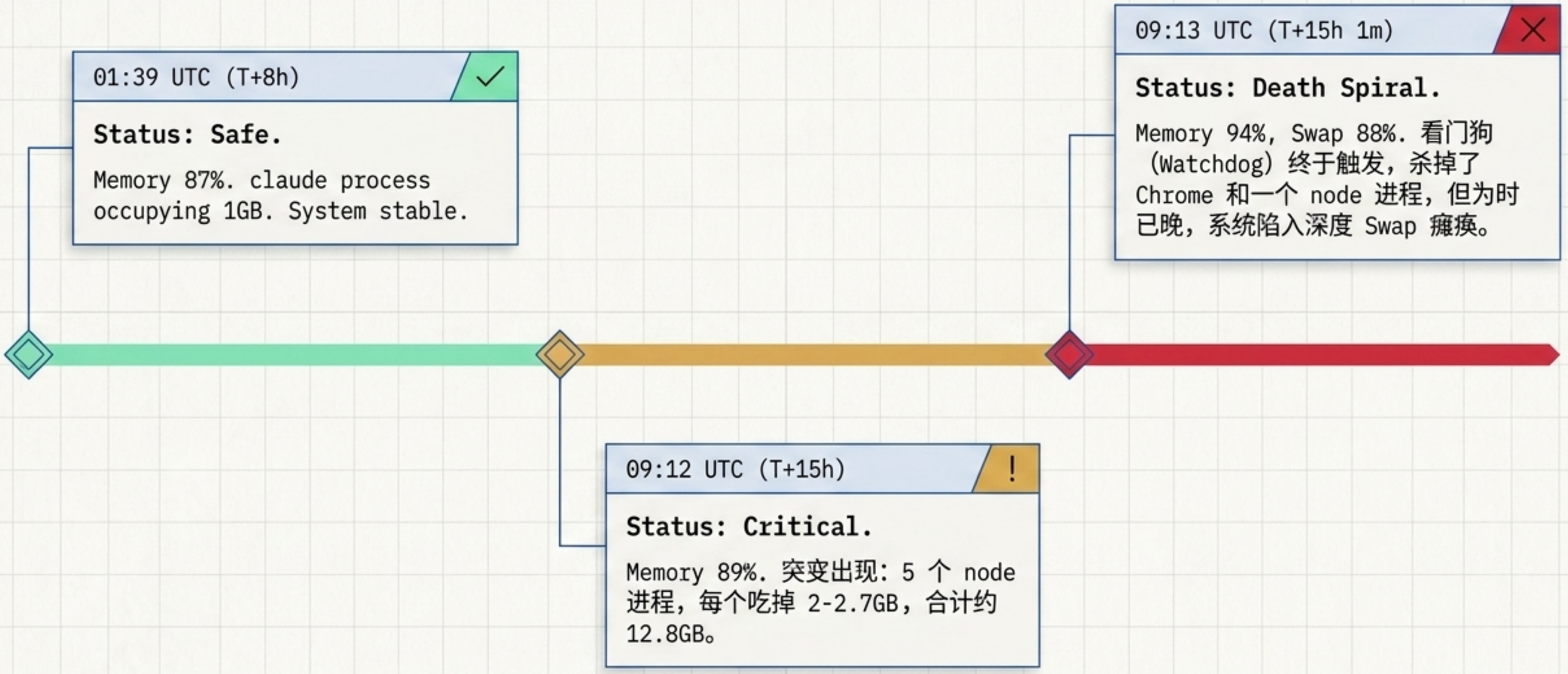
原本安装用于防止崩溃的 GCP Ops Agent，因 IAM 权限缺失，陷入疯狂的错误重试循环，反而消耗了大量磁盘 I/O 和内存，加速了系统的死亡。



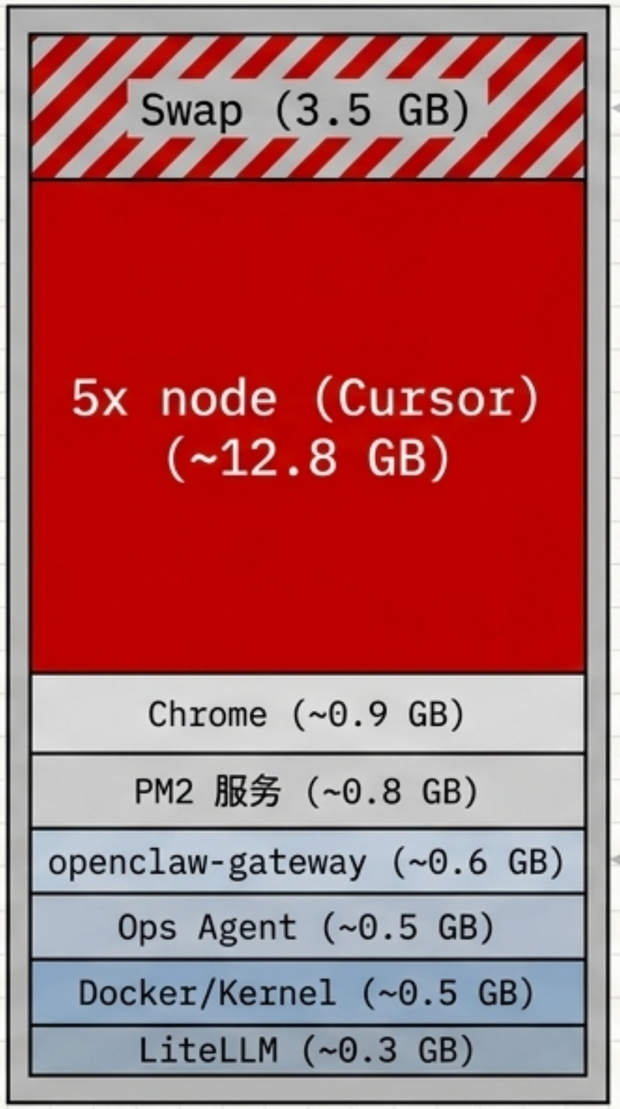
临终遗言

日志在输出这条内存高压警告后戛然而止，系统彻底陷入沉默。

Forensic Timeline



Memory Autopsy



32GB 的机器，光进程就吃掉了约 20GB。五个 node 进程占了将近一半的可用空间。

Suspect Comparison Matrix

Digital Identity Scan: Process & Resource Analysis

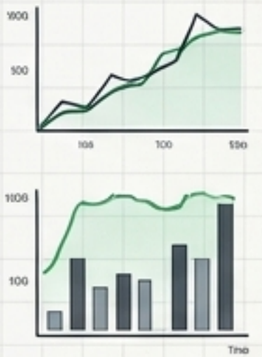
嫌疑人 A (Initial Bias)

- **Identity:** Claude Code
- **Process Name:** `claude`
- **Motive:** Known to use 1-2GB per session. 5 sessions = ~10GB+. Plausible.

FAILED: Watchdog logs did not name 'claude'

嫌疑人 B (True Culprit)

- **Identity:** Cursor 远程服务器
- **Process Name:** **node**
- **Actual Footprint:** 5 processes, ~12.8GB total.



CONFIRMED

真相：如果不看进程名，默认 'node = Claude Code'，修复方向就全错了。

Exploded View: Memory Leak Mechanics

server-main.js
缓存打开文件、撤销历史与搜索索引（从不释放）。

ptyHost
终端滚动缓冲区（从不截断）。

extensionHost
所有扩展的宿主。V8 垃圾回收懒惰，内存单调递增。

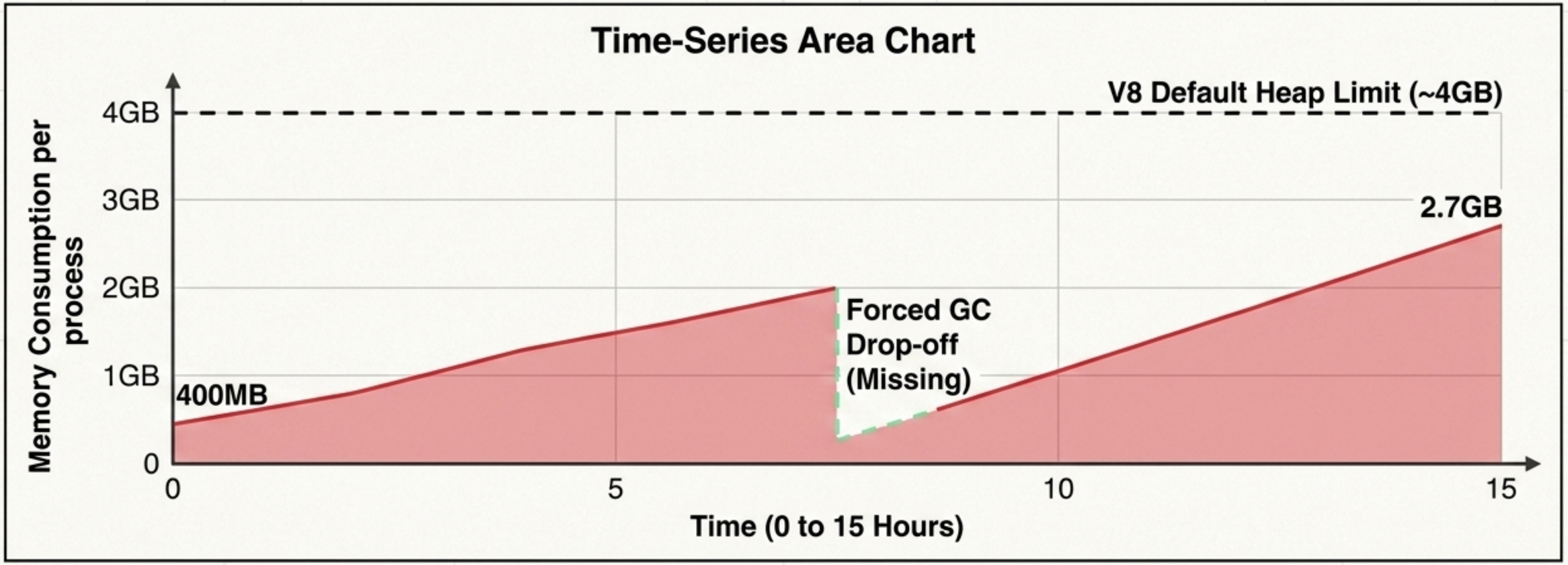
Cursor SSH Mode

fileWatcher
维护内存中的文件树（随仓库大小持续增长）。

tsserver
将整个项目的 AST 放入内存（重新解析但不收缩）。

这不是单一的泄漏，而是五个进程同时在执行无约束的内存囤积。

Memory Leak Timeline: V8 Heap Expansion



V8 的懒惰 GC 陷阱

默认未设置 `--max-old-space-size`。V8 只有在接近 4GB 极限的巨大内存压力下才会执行 major GC。配合 Cursor 远程服务，每个进程天生就是一个慢速内存泄漏点。

Environmental Hardening: System-Level Mitigations

Cut 1 - 权限加固（IAM Shield）

修复 Ops Agent 权限，授予 monitoring.metricWriter 和 logging.logWriter，彻底阻断错误日志的 I/O 洪流。

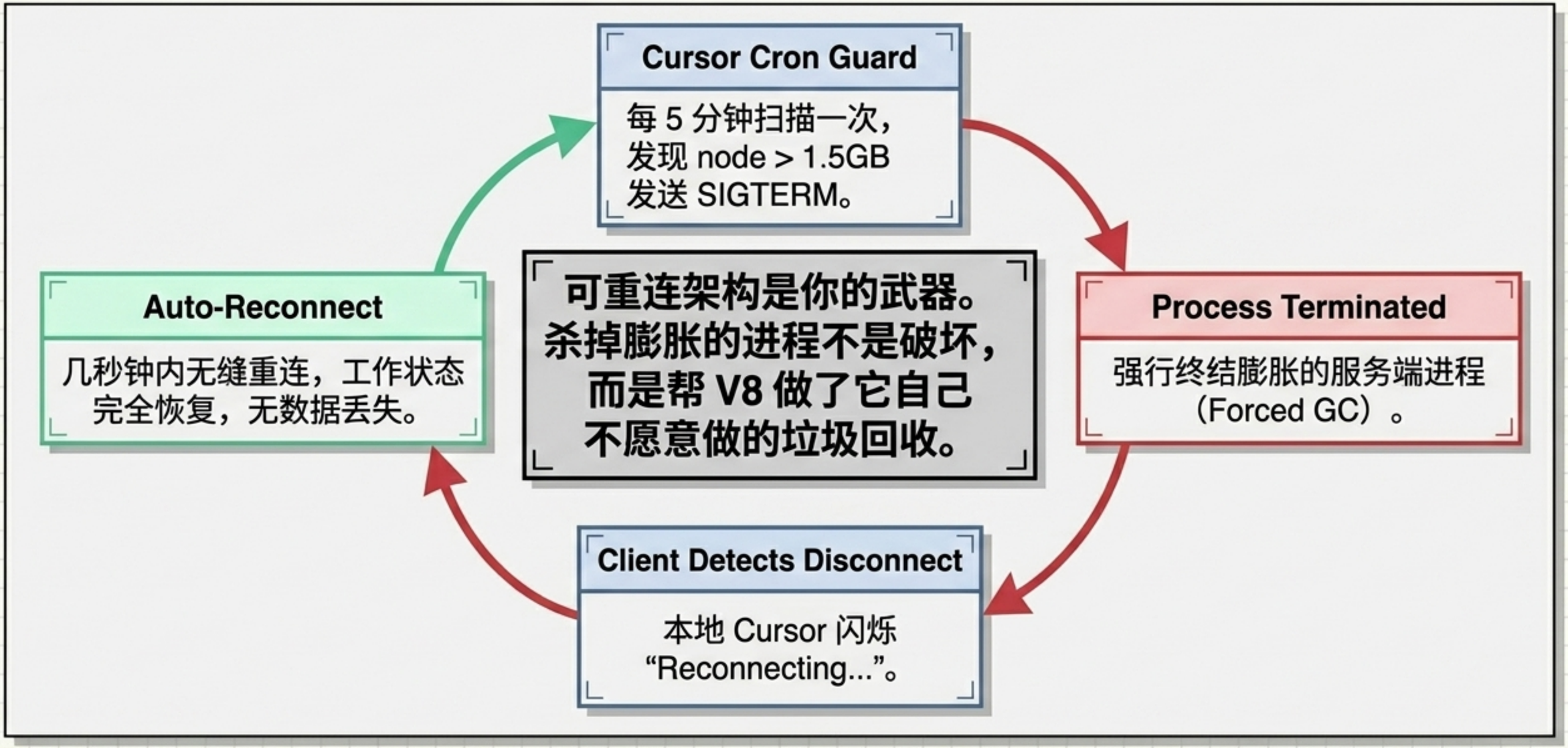
Cut 2 - 看门狗协议升级（Watchdog Protocol Matrix）

Old Protocol	New Protocol - Pre-emptive Strike
85%: Warning -> 92%: Kill Chrome (Result: System already drowning in Swap).	• 75%: 预警记日志 (Warn) • 80%: 杀 Chrome (Kill Chrome) • 85%: 猎杀占用 >1GB 的 node 进程 (Targeted Node Kill)

System Status: Mitigated

Executive Digital Forensics Dossier

The Reconnect Weapon: Architecture as Defensive Strategy



System Status: Mitigated

Executive Insights & Strategic Takeaways

监控本身可以成为问题

监控工具若无正确权限和资源限制，会反向吞噬系统资源（如 Ops Agent 的 I/O 洪流）。

进程名是诊断的真相

绝不依靠直觉假设。node 和 claude 的区别决定了调查方向，日志永远比偏见诚实。

激进利用无状态架构

如果服务端足够无状态（如 Cursor 远程架构），暴力重启就是最完美、最安全的内存回收机制。

32GB 只是买了缓冲时间，没有改变根本问题。

任何长时间运行且没有进程级内存上限的程序，最终都会把你给它的 RAM 填满。硬件无法解决软件的无度。

[FORENSICS_REPORT_END] // System Stable.