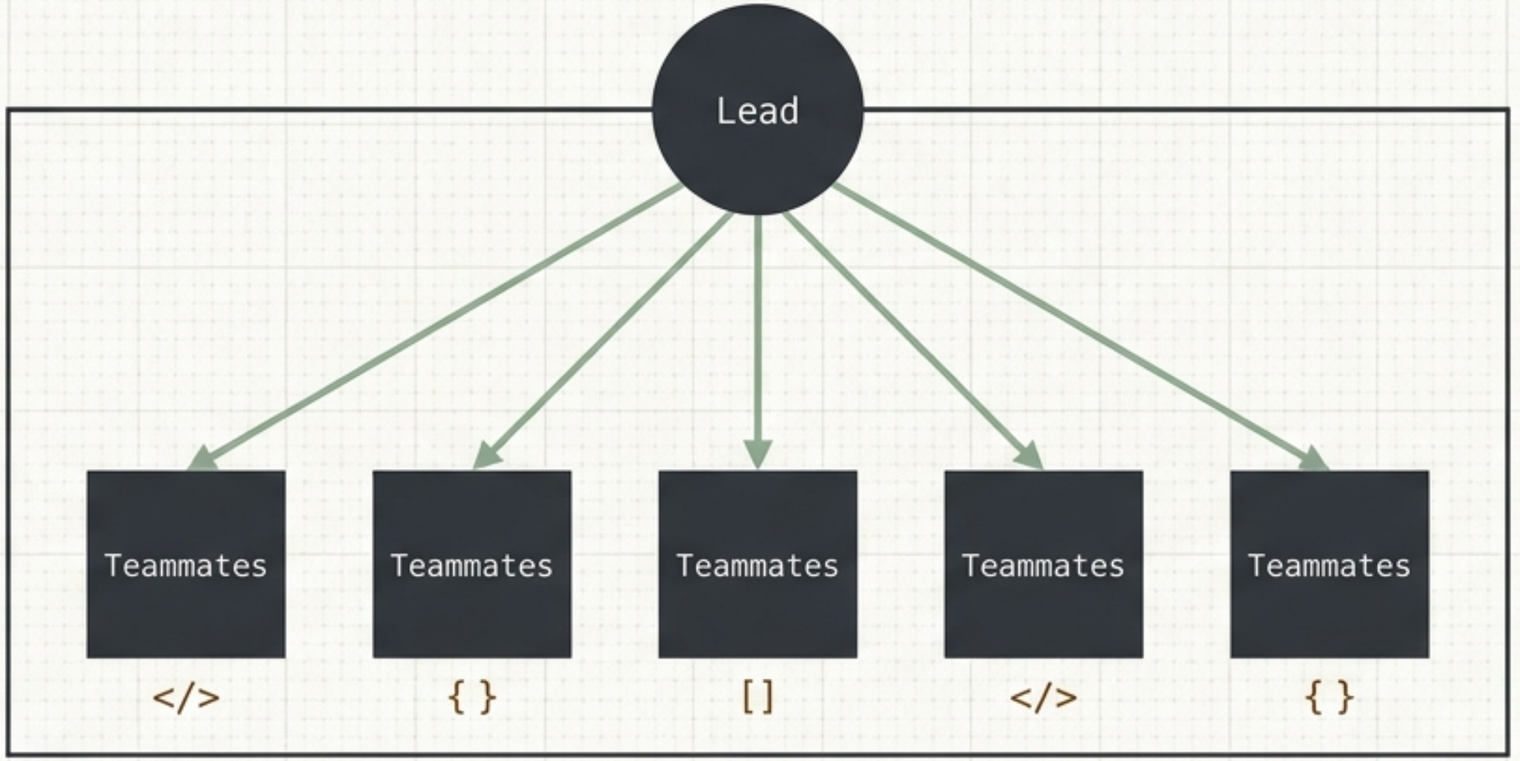
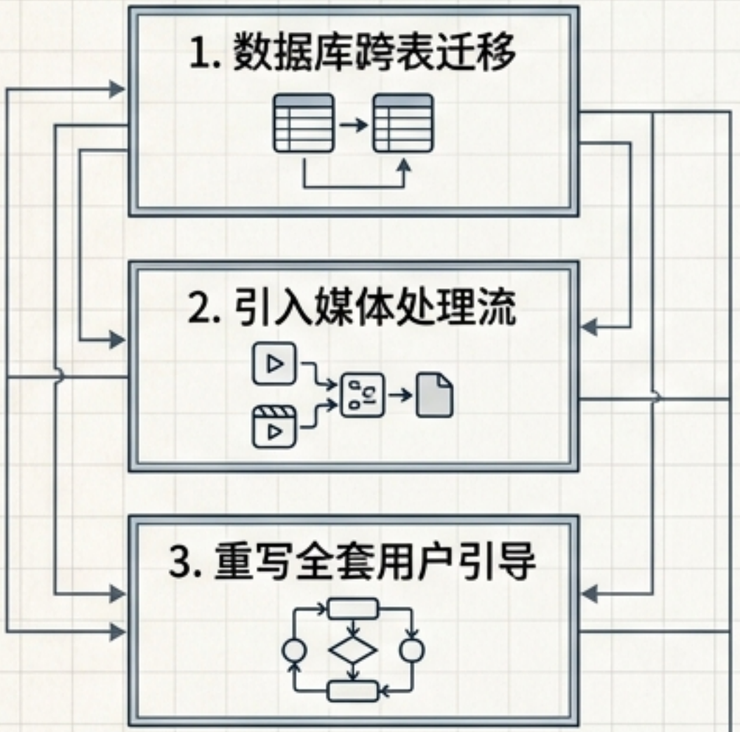
五个 AI 同时开工的协同蓝图

基于 Agent Team 的软件工程全自动化调度实战解剖



单体 AI 的工程死胡同：物理极限

复杂工程负载

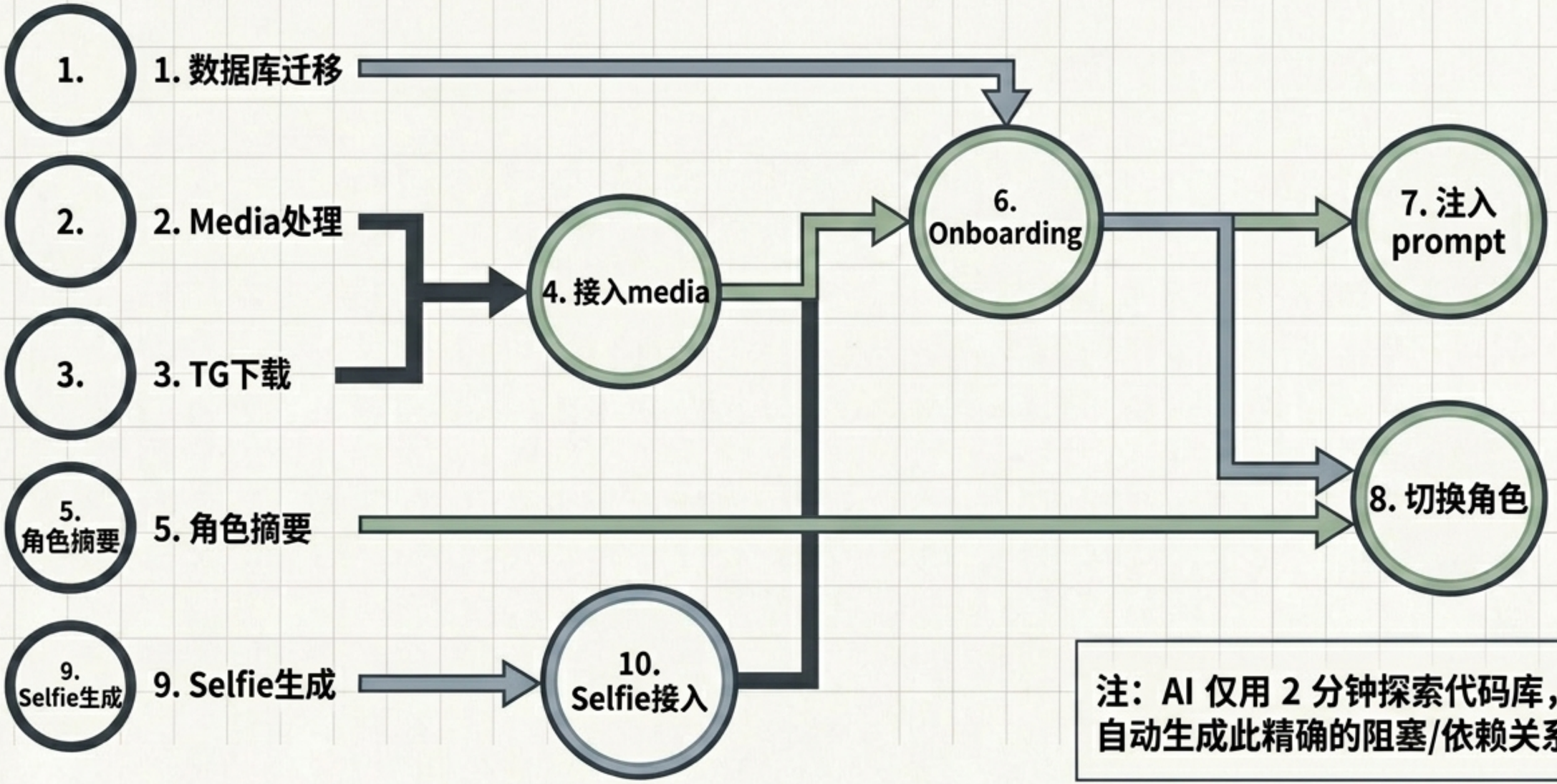


物理上下文瓶颈

大任务拆解后后的串行执行已经失效。

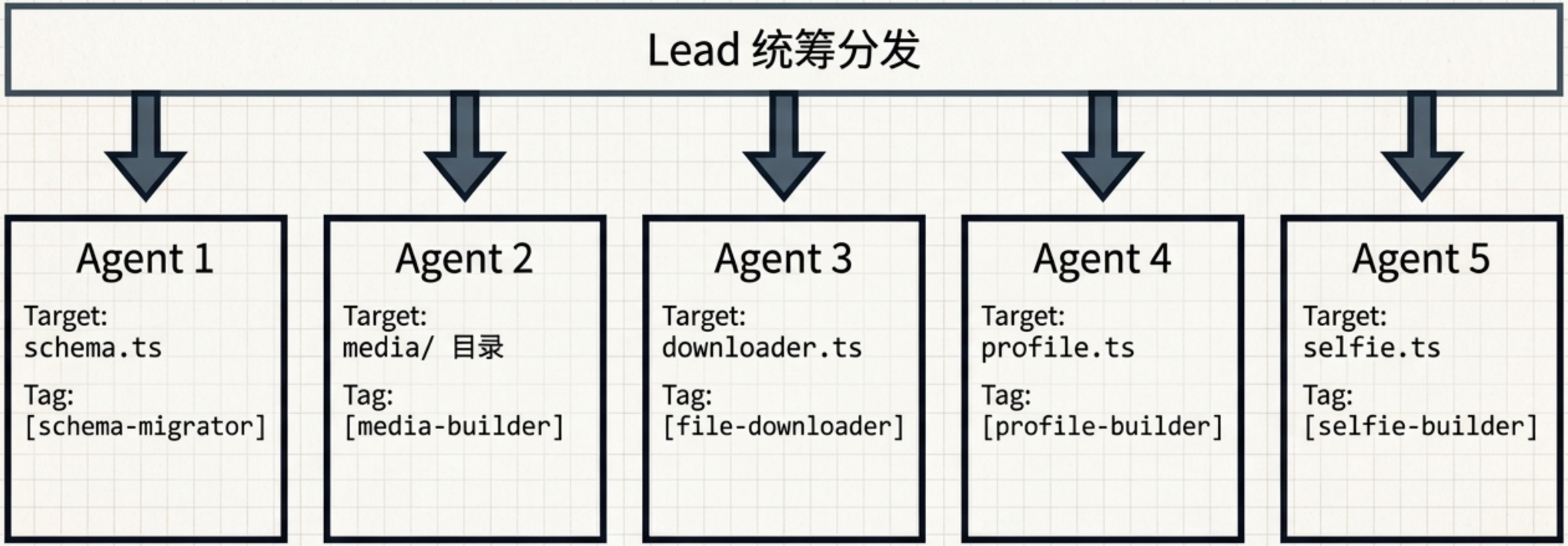
范式转移：三层协同指挥架构

AI 眼中的工程蓝图：构建依赖 DAG



注：AI 仅用 2 分钟探索代码库，即自动生成此精确的阻塞/依赖关系网。

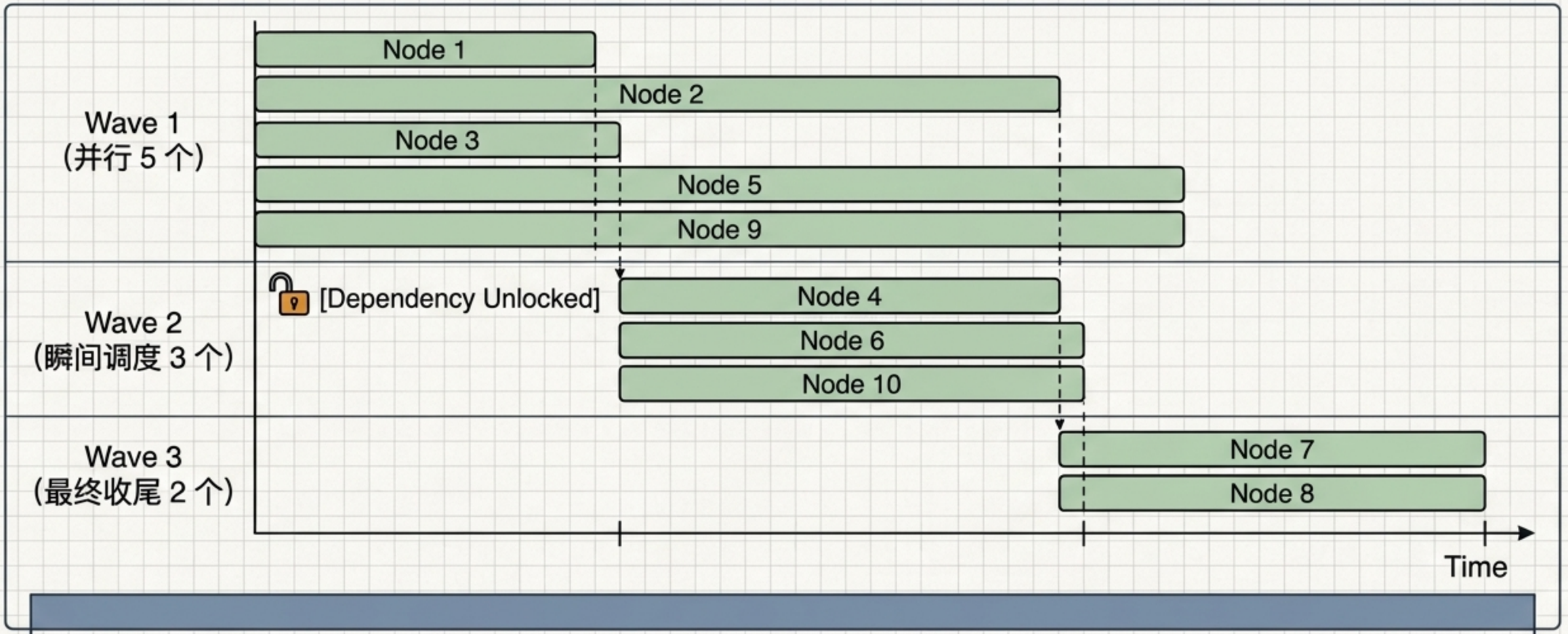
第一波攻击：绝对的上下文沙盒



文件所有权（File Ownership）

每个 Teammate 的权限被严格锁定在分配的文件与目录中。这是彻底消灭 Git 代码冲突、实现真并行的物理级保障。

动态调度：解开依赖锁



Lead 持续监听 DAG 状态。不需要人类干预，上游任务一旦完成，下游的阻塞即刻解除并自动分配给新启动的 Agent。

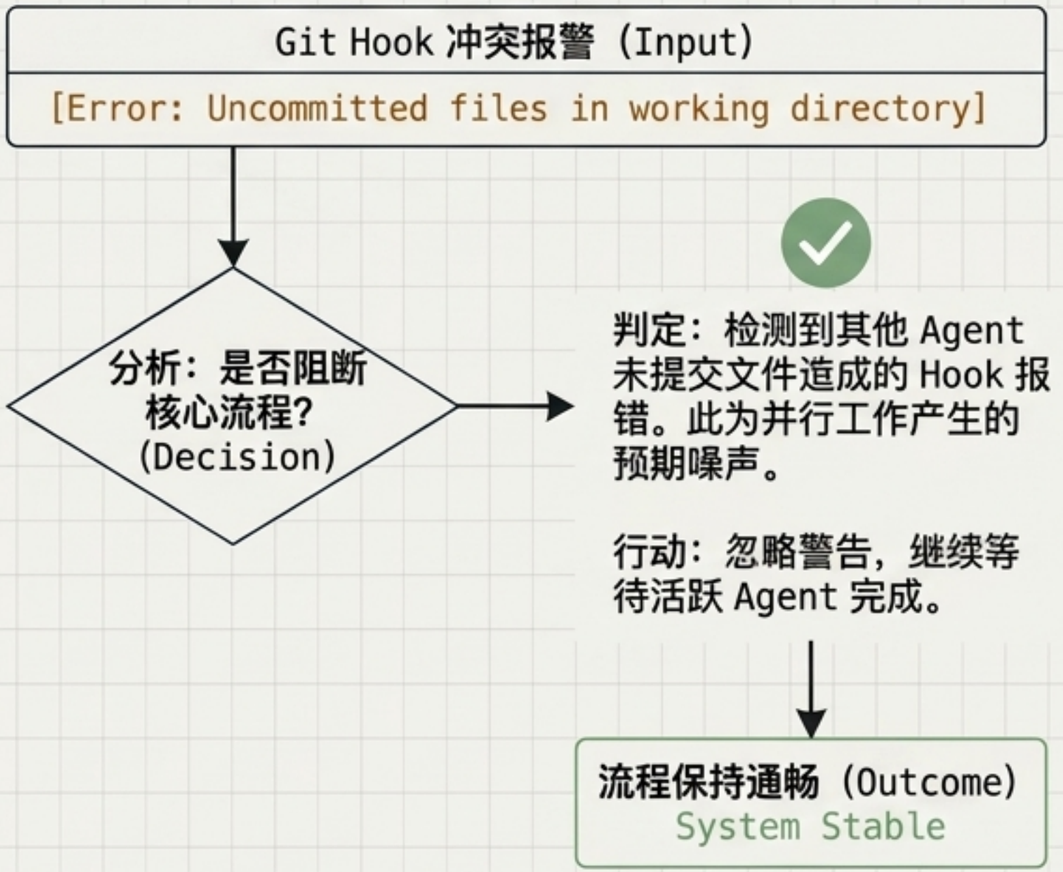
鲁棒性测试（1）：过滤噪音 vs 保持专注

系统警报

```
> git status
> [Error: Uncommitted files in working directory]
> [Warning: Concurrent file access detected]
> [Error: Uncommitted files in working directory]
> [Info: Agent 2 accessing media/]
> [Error: Uncommitted files in working directory]
> [Warning: Potential conflict in schema.ts]
> [Error: Uncommitted files in working directory]
> [Info: Agent 4 modifying profile.ts]
> [Error: Uncommitted files in working directory]
```

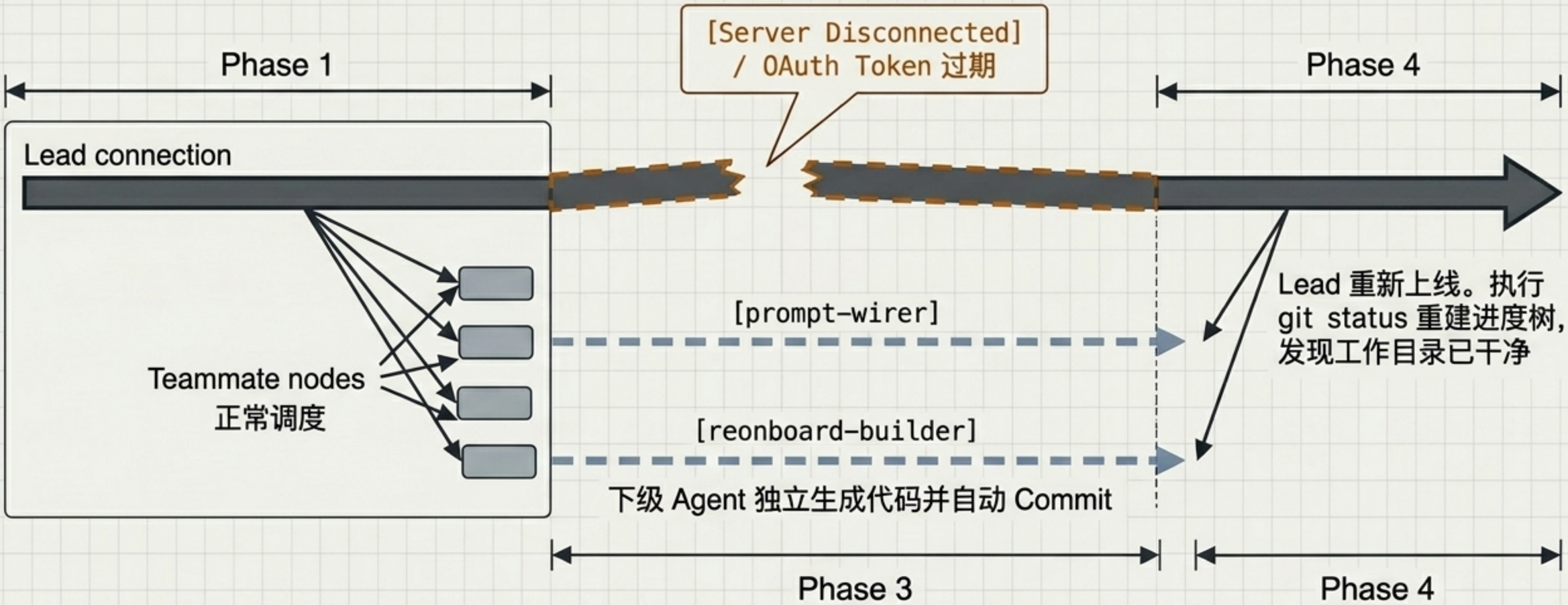
5 个 Agent 并发写代码，触发全局环境警告

Lead 的决策逻辑



核心能力：优秀的调度器懂得区分真正的系统故障与预期的环境噪音。

鲁棒性测试（2）：断线后的幽灵运作



独立运行属性：Teammate 不依赖轮询。上级掉线，下级依然会自主完成任务并提交代码。

闭环交付：最终的工程收尾

全自动扫尾协议（Post-Execution Protocol）

- ☑ **Typecheck**：运行 pnpm typecheck。检测到并自动修复两个 TS18048 错误
- ☑ **Test**：运行全量测试套件，确认无新增失败用例
- ☑ **Code Review**：扫描所有新生成文件的导出依赖正确性
- ☑ **Docs Sync**：自动派发额外 Subagent，并行更新开发文档与 TODO 列表

1 个计划输入	10 个子任务	10+ 个核心文件	跨 3 层架构	1 次会话内交付

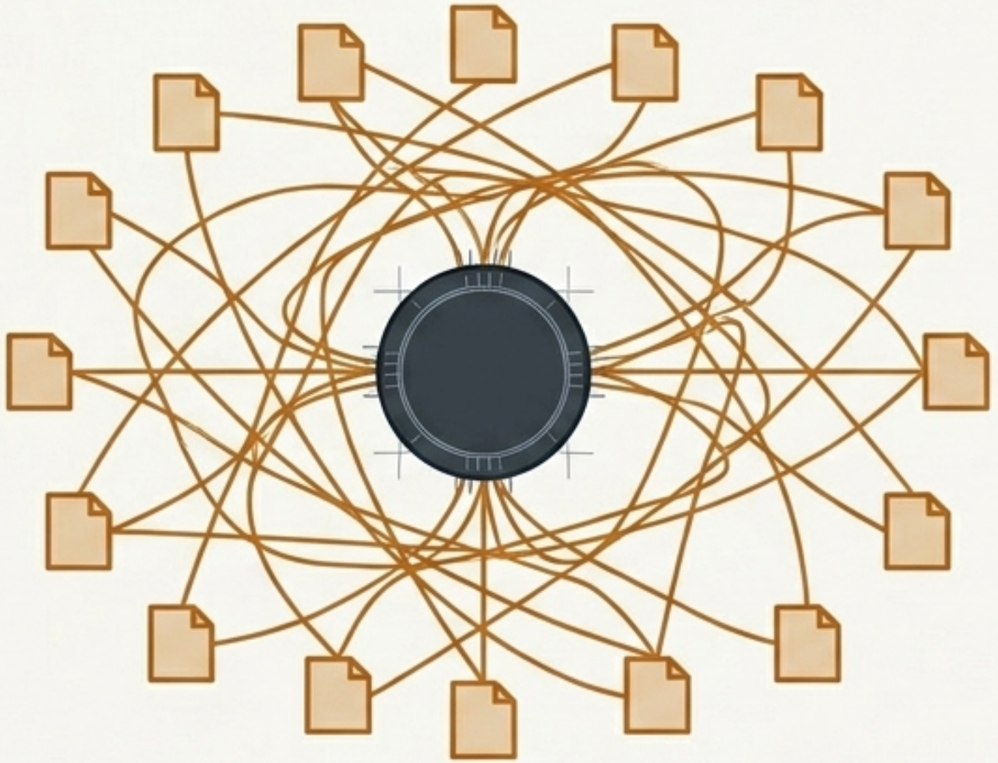
决策矩阵：何时应该启用 Agent Team?

评估维度	启用 Team 模式 （并行多 Agent）	保持单体模式 （单 Agent 串行）
任务形态	>3 个可并行的独立子任务，存在清晰的依赖关系图	完全串行，无并行空间，或仅修改 1-2 个文件
工程跨度	跨越 3 个以上不同目录或独立系统模块	修改高度集中在单一模块或局部逻辑
上下文需求	全局细节超出单一窗口极限，需按模块隔离	单窗口足以覆盖，且需要深度共享的调试状态
核心挑战	范围庞大、需要系统性重构与多点接入	复杂的单点逻辑推演（如仿真步进调试）

Team 模式的本质不是取代单体，而是降维处理工程规模度问题。

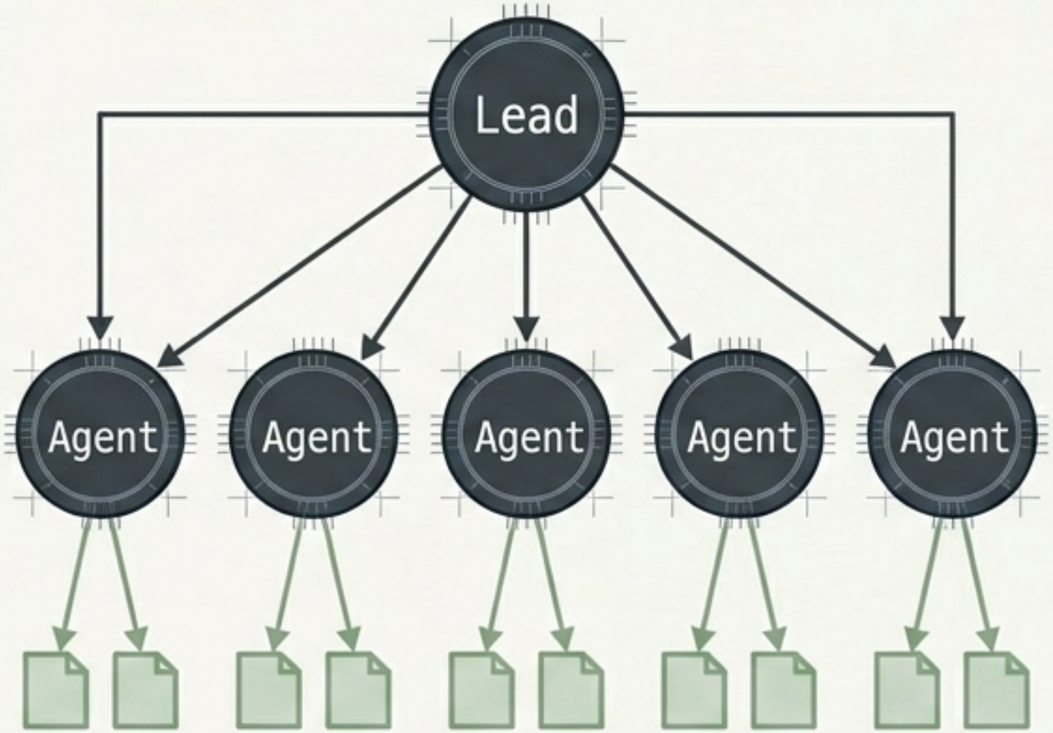
核心洞察：效率只是表象，隔离才是本质

单体模式的认知灾难



单体 AI 加载全局代码，逻辑极易交叉污染。

Team 模式的认知隔离



Agent 只聚焦专属模块，Lead 只维护依赖图。
最经典的系统工程分工法则。

你的全新定位：管理者的管理者

过去（结对编程）

人类 + AI =
逐行代码审查与逻辑微操。

现在（协同编排）

你（Manager's Manager）
定义商业逻辑、指定大体计划、确立验收标准

Lead（项目经理）
维护依赖图、调度计算资源、处理系统报错

Teammates（干将）
纯粹的执行与底层代码编写

好的分工不是为了让每个人干得更快，而是让每个人只需要关心自己的部分。
我全程没有审查任何一段具体的代码——我只负责定目标，剩下的，交给 AI 团队。