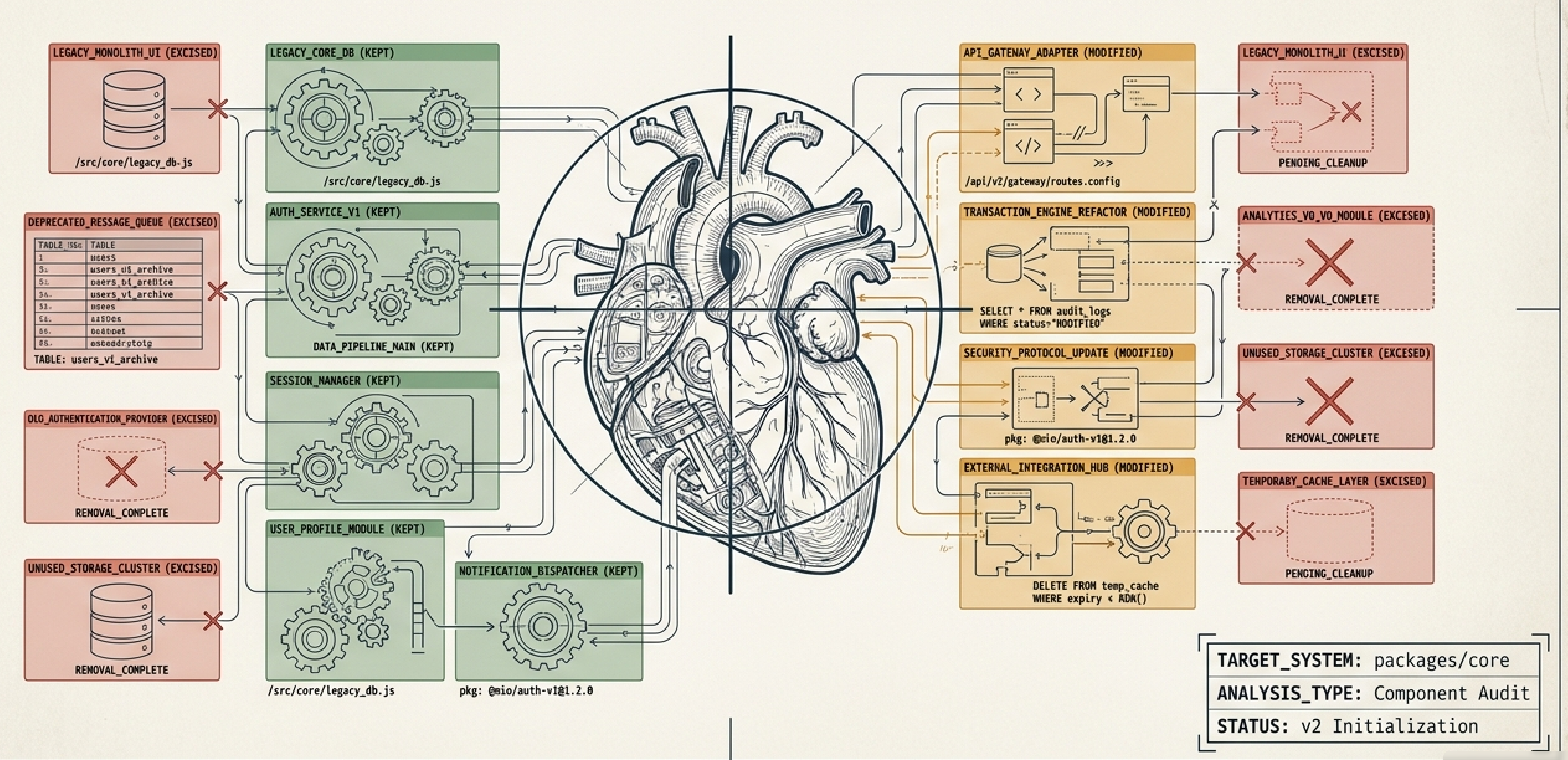
Mio v1 技术尸检报告：一场器官移植手术

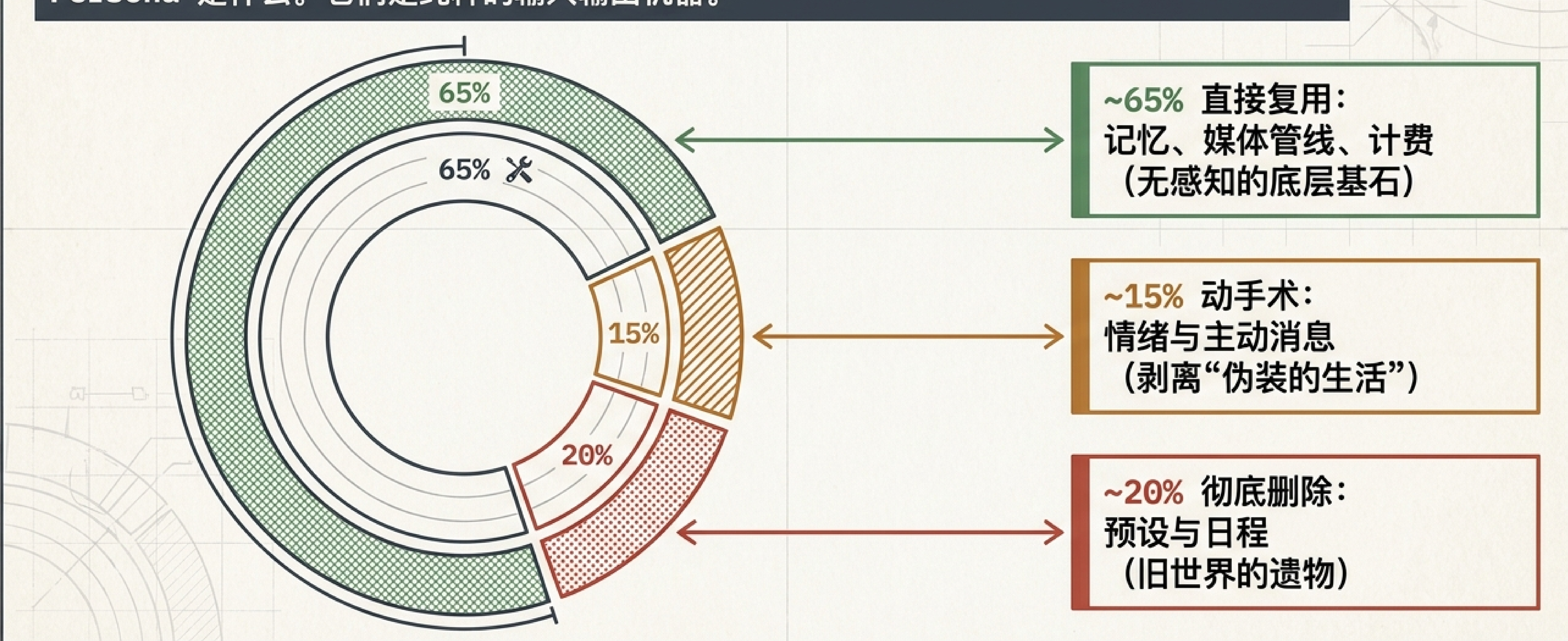
核心架构的保留、改造与切除



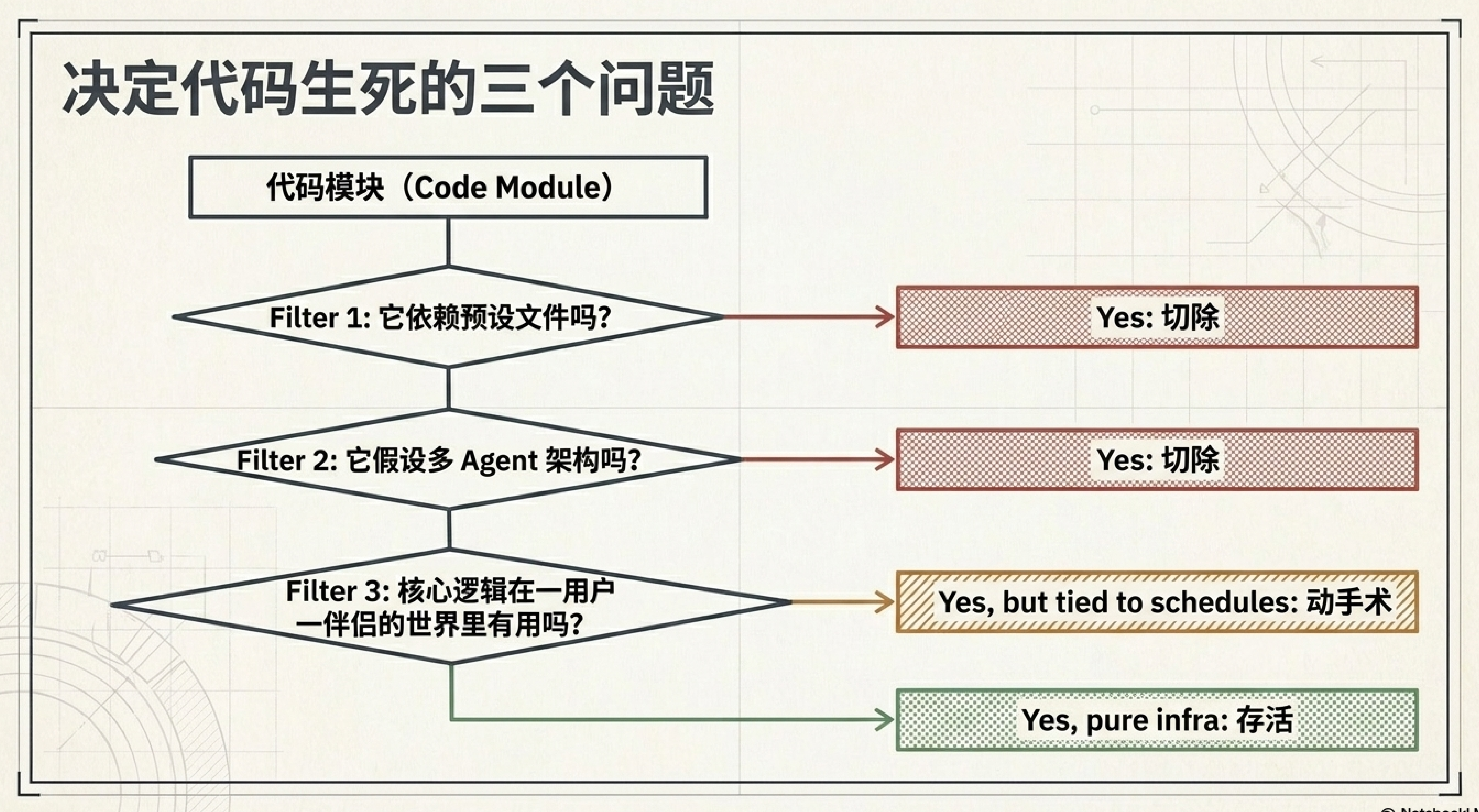
NotebookLM

重建 ≠ 全部重写

当追踪 packages/core 的 import 路径时，答案出乎意料：大部分底层模块根本不在意 Persona 是什么。它们是纯粹的输入输出机器。



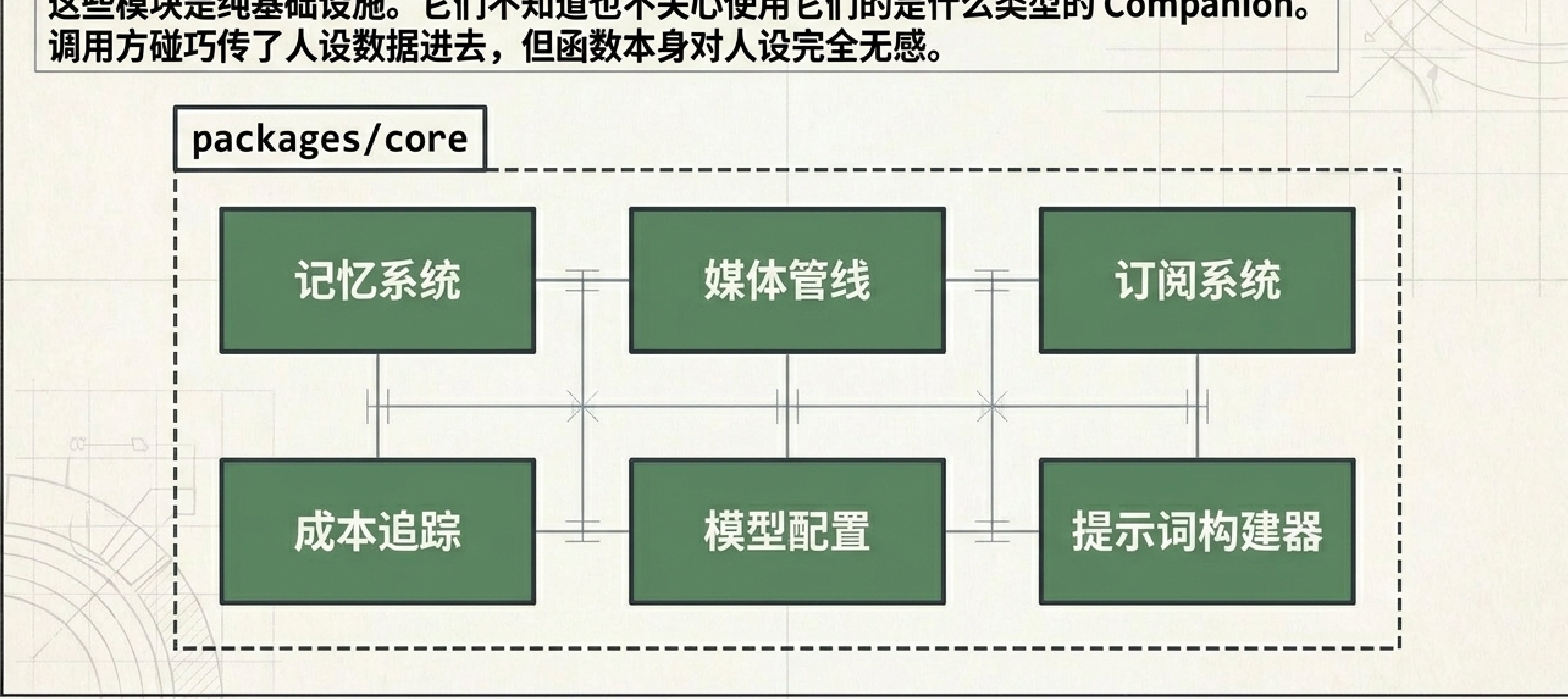
NotebookLM



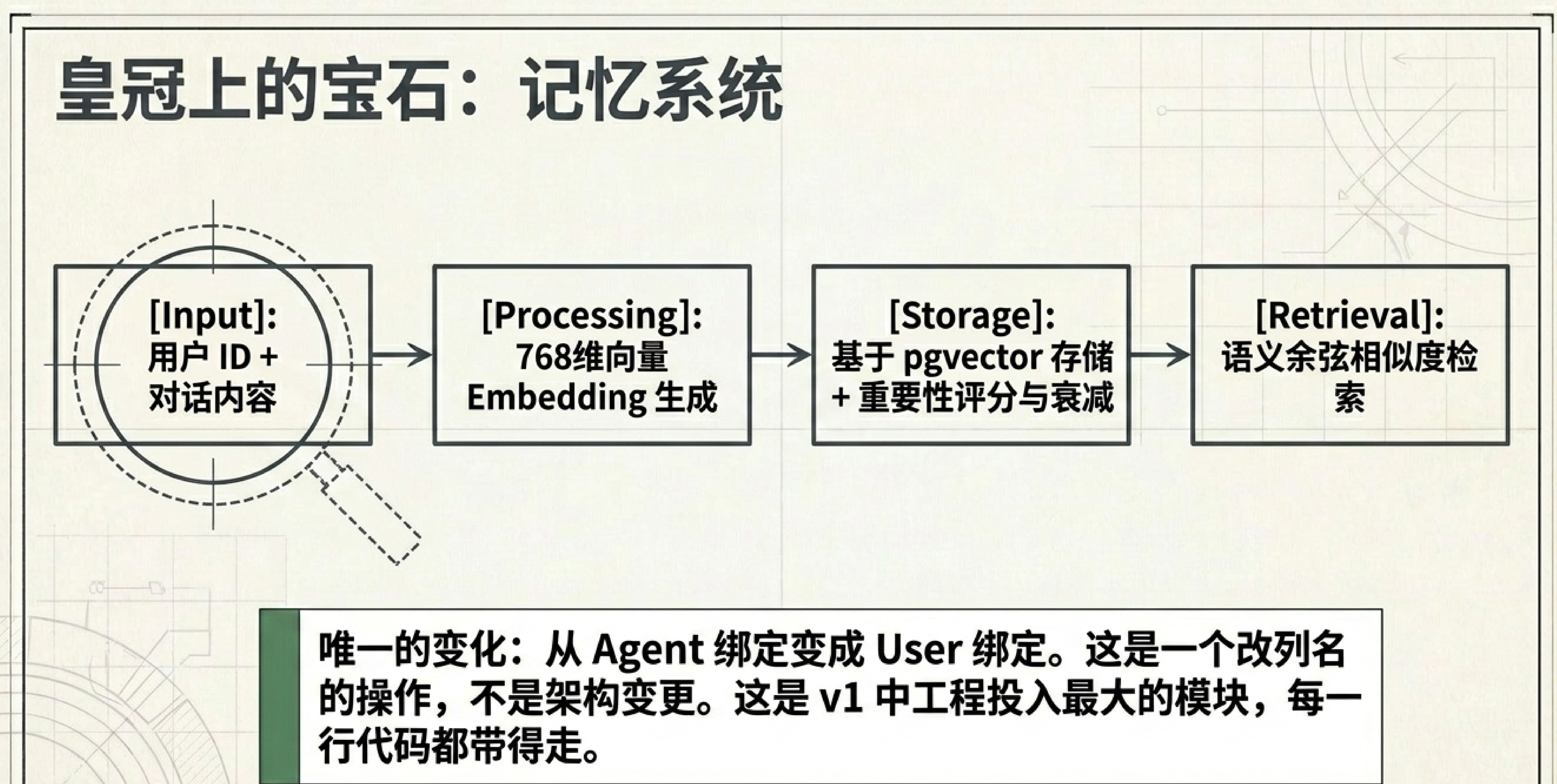
决定代码生死的三个问题
代码模块（Code Module）
Filter 1: 它依赖预设文件吗?
Yes: 切除
Filter 2: 它假设多 Agent 架构吗?
Yes: 切除
Filter 3: 核心逻辑在一用户一伴侣的世界里有用吗?
Yes, but tied to schedules: 动手术
Yes, pure infra: 存活
NotebookLM

存活者：无感知的底层基石

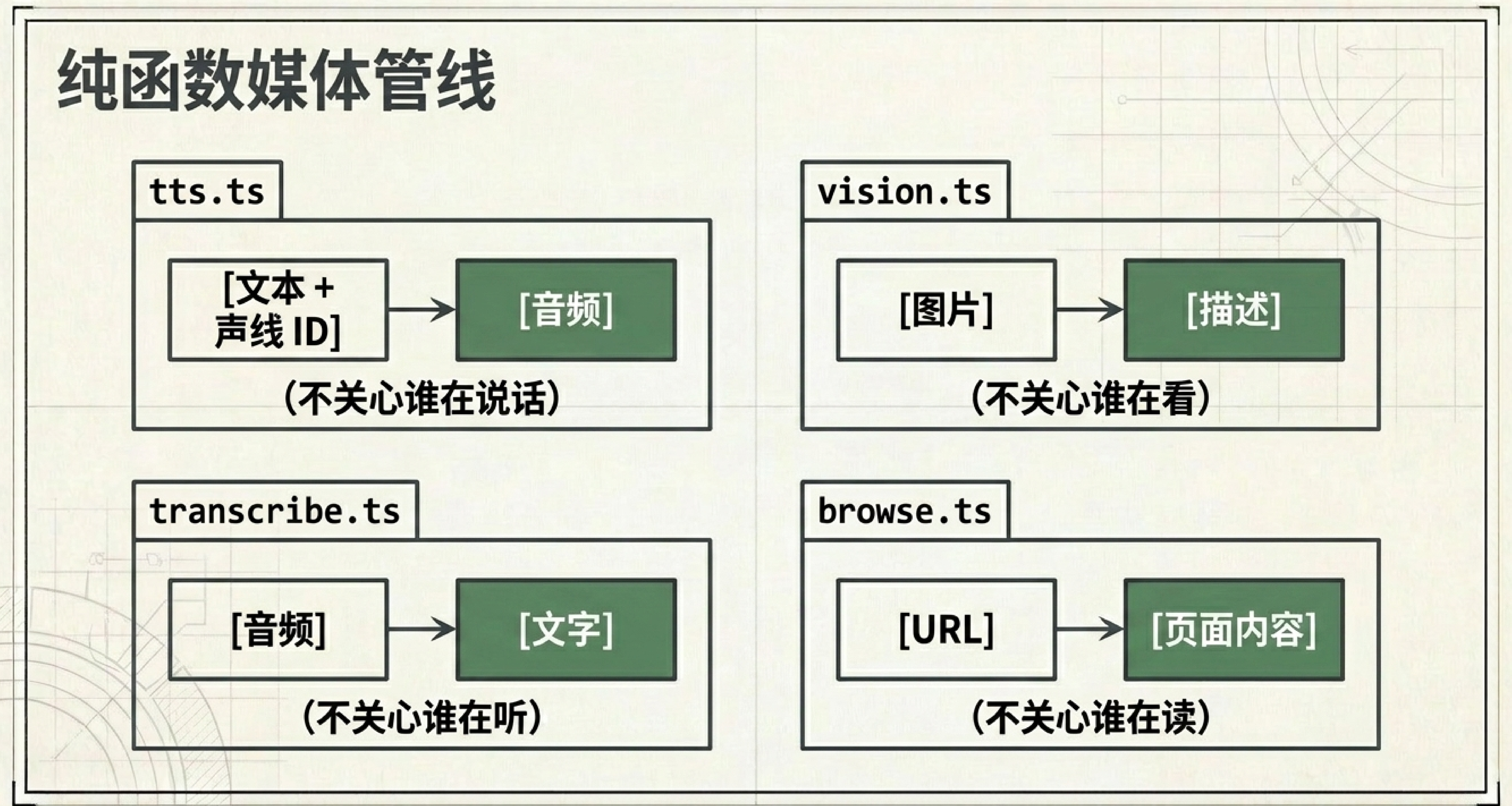
这些模块是纯基础设施。它们不知道也不关心使用它们的是什么类型的 Companion。
调用方碰巧传了人设数据进去，但函数本身对人设完全无感。



NotebookLM



NotebookLM


纯函数媒体管线
tts.ts
[文本 + 声线 ID]
[音频]
(不关心谁在说话)
vision.ts
[图片]
[描述]
(不关心谁在看)
transcribe.ts
[音频]
[文字]
(不关心谁在听)
browse.ts
[URL]
[页面内容]
(不关心谁在读)
NotebookLM

账房与调度台

`core/cost/`

成本追踪

记录每次 LLM 调用、TTS 合成、视觉分析的 token 数和美元成本。纯会计逻辑，支撑单位经济学。

`core/subscription/`

订阅系统

知道 free/starter/pro/max 层级和每日额度。完全独立于 Companion 层。

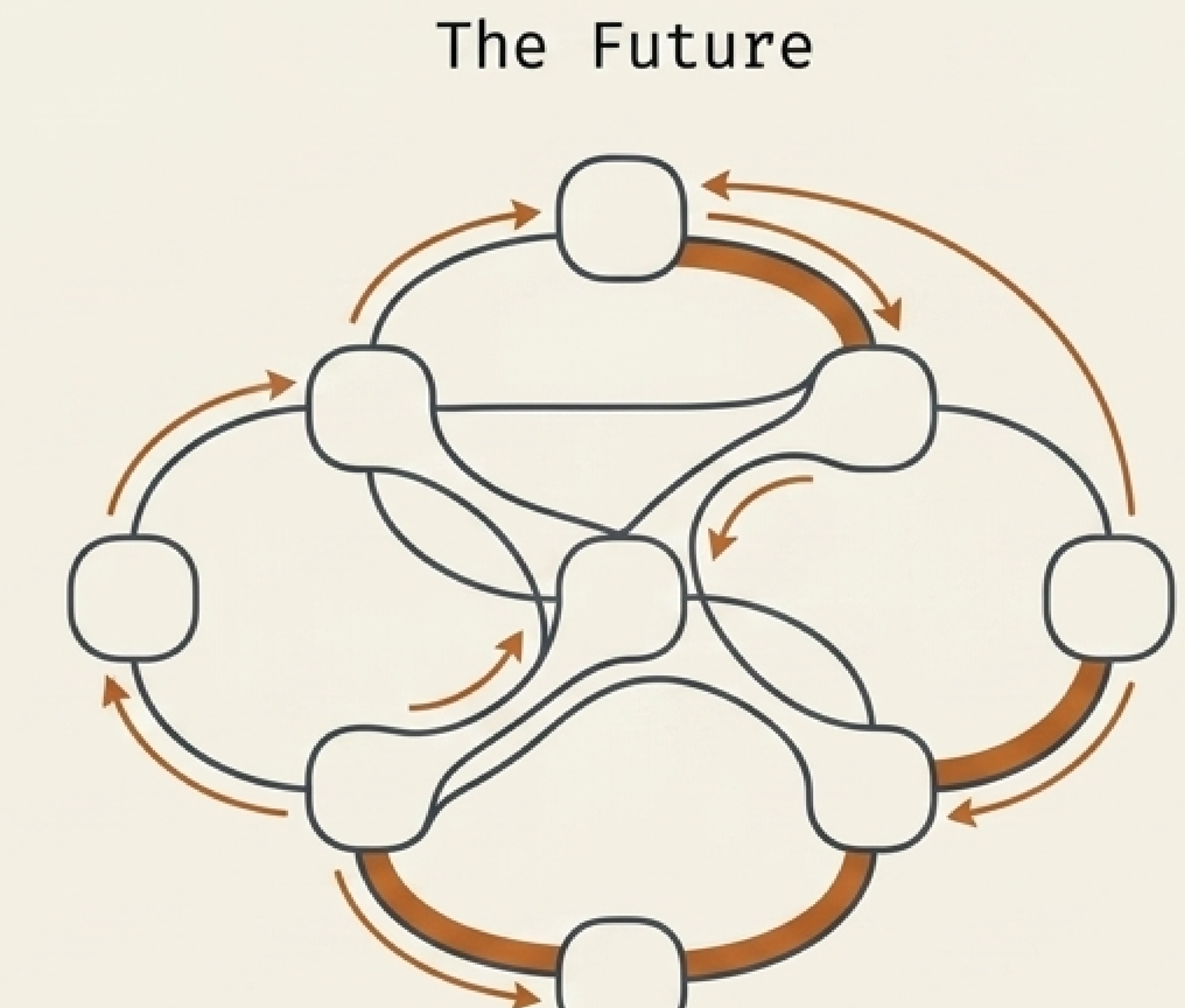
`models.ts`

模型配置

定义可用 LLM、定价、上下文窗口。无人设依赖的纯路由规则。

NotebookLM

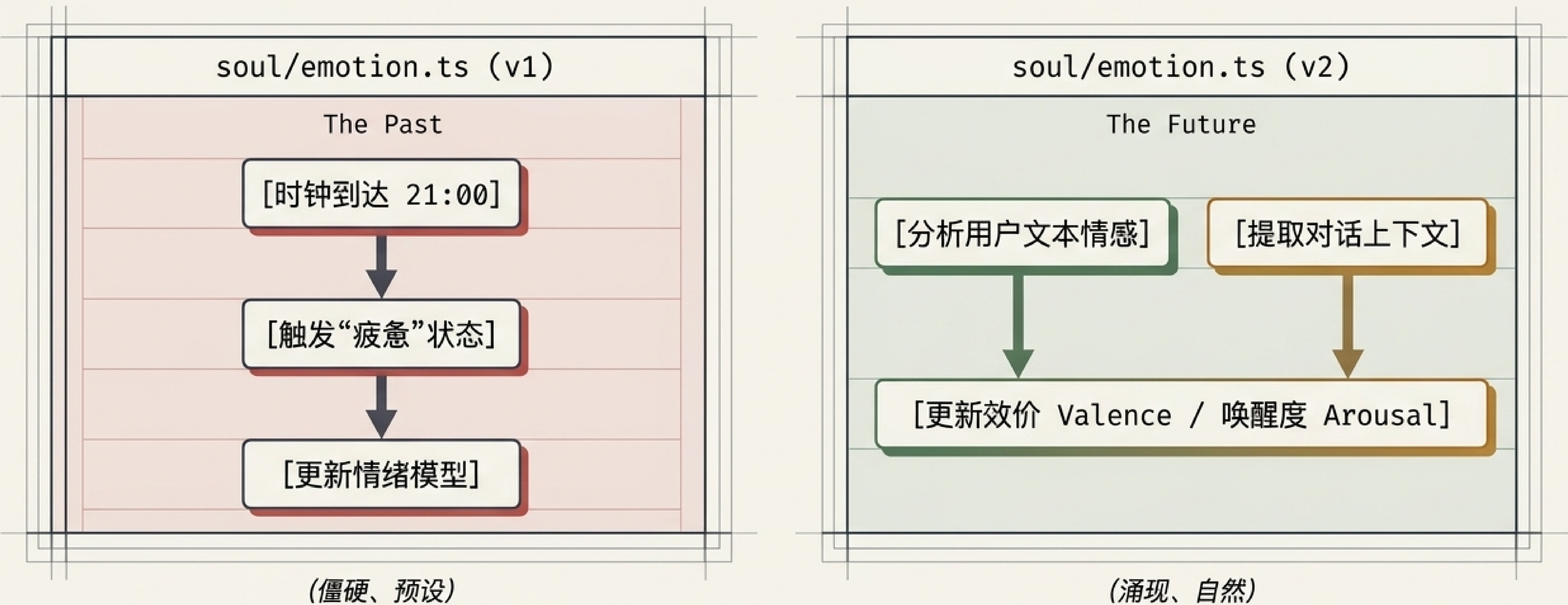
剥离“伪装的生活”



核心逻辑是扎实的，但带着 v1 对“人类作息”的执念。系统会在晚上“觉得累”，仅仅是因为日程表上这么写，而不是基于真实的对话动态。
治疗方案：保留核心模型，彻底拔掉所有伪造生命的时间表触发器。

NotebookLM

情绪引擎：从日程驱动到交互驱动



（僵硬、预设）

（涌现、自然）

情绪模型留下，假装有生活的触发器走人。

NotebookLM

主动消息：更简单，更诚实

时间感知（Time）

知道几点，但不装有日程。

（'晚上了，今天过得怎么样？'）

记忆驱动（Memory）

从存储中提取未完话题。

（'你上次说要去面试，结果如何？'）

情绪延续（Emotion）

读取上一次交流的效价状态。

（'昨天聊完感觉你心情不太好，好点了吗？'）

单纯关心（Gap）

检测对话间隔的纯逻辑。

（'好几天没聊了，想你了。'）

NotebookLM

旧世界的遗物 (~20%)

Core Insight

不写悼词。它们在人格驱动的世界里完成了使命，但那个世界已经不存在了。数百行精心打磨的背景故事、设定文件和照片生成器，全部删除。

The Paradigm Shift

在 v2 中，性格从对话中自然涌现，而不是从 JSON 文件中读取。

```
/root
|-- /config
|     |-- settings.json
|-- /personas
|     |-- /barista
|     |     |-- backstory.json
|     |     |-- portrait_gen.py
|     '-- /student
|           |-- settings.yaml
|           |-- memories.db
'-- /dialogue_engine
      '-- legacy_triggers.ts
```

NotebookLM

切除清单：不写悼词，只写原因

所有预设文件 (Preset Files)

成都咖啡师 或 学姐 的身份定义。

死因：性格不再被硬编码。

参考图片 (Reference Images)

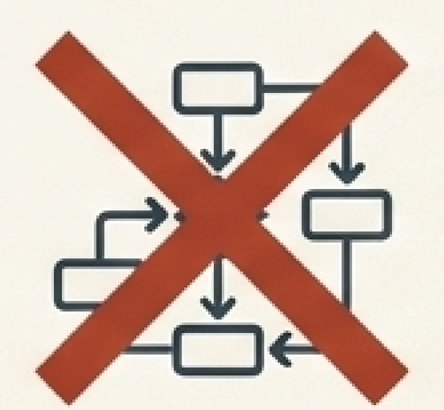
用于自拍生成的照片。

死因：一个不假装是人类的 Companion 不需要伪造的脸。

日程系统 (schedule-*.ts)

假装上班和做瑜伽。

死因：对留存贡献最小。用户的依赖来源于记忆，而不是虚假的作息。



关系进化状态机 (evolution-*.ts)

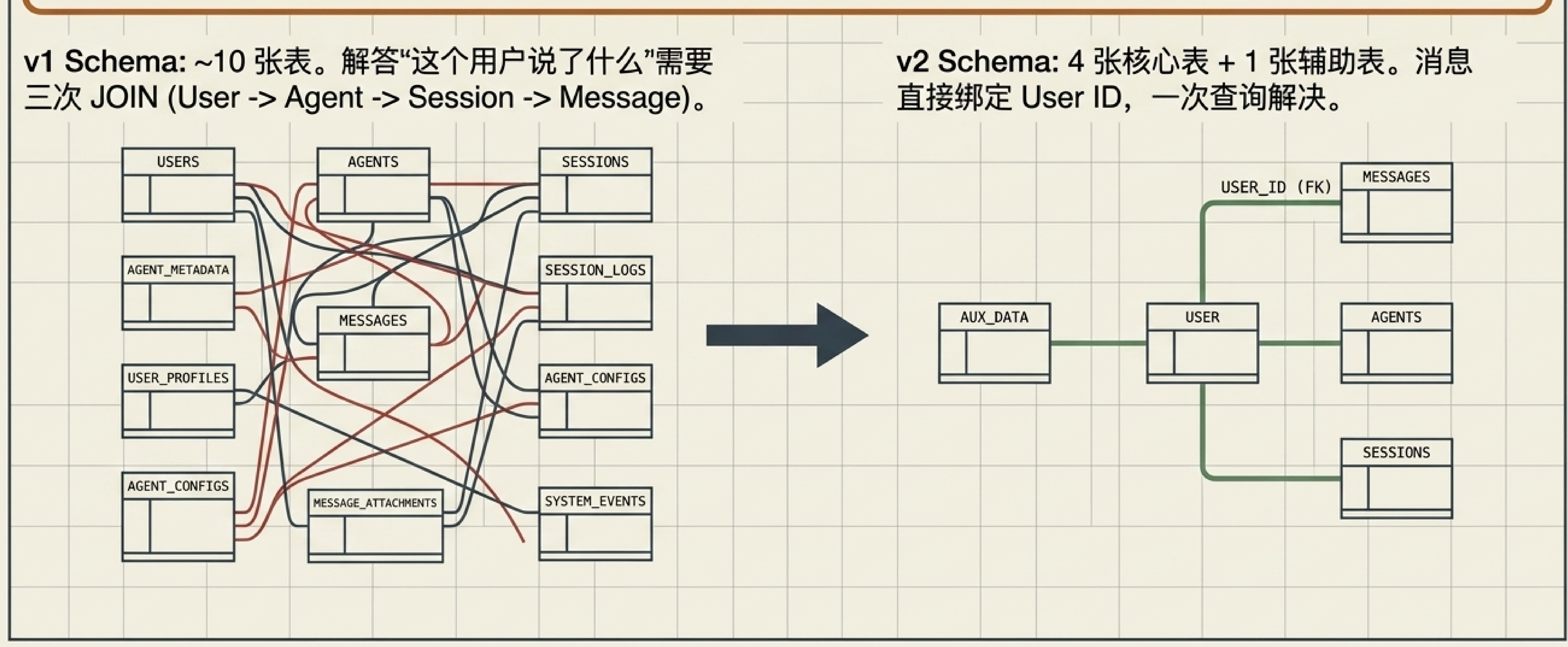
显式的关系阶段推进。

死因：过度工程。记忆和温度自然会加深关系，不需要状态机。

NotebookLM

架构坍缩：从 10 张表到 4 张表

整个多 Agent 基础设施，最终坍缩成了一个优雅的外键约束。



NotebookLM

消灭多 Agent 的终极约束

```
CONSTRAINT one_companion_per_user UNIQUE (user_id)
```

被消灭的表	死因
agents	一用户一伴侣，无需多个角色实例。
sessions	消息变为每个用户的单条扁平流。
channel_bindings	回归单平台 Native App。
onboarding_states	状态活在对话流中，无需分支状态机。
telegram_allowlist	放弃跨平台。
account_link_tokens	放弃跨平台。

NotebookLM

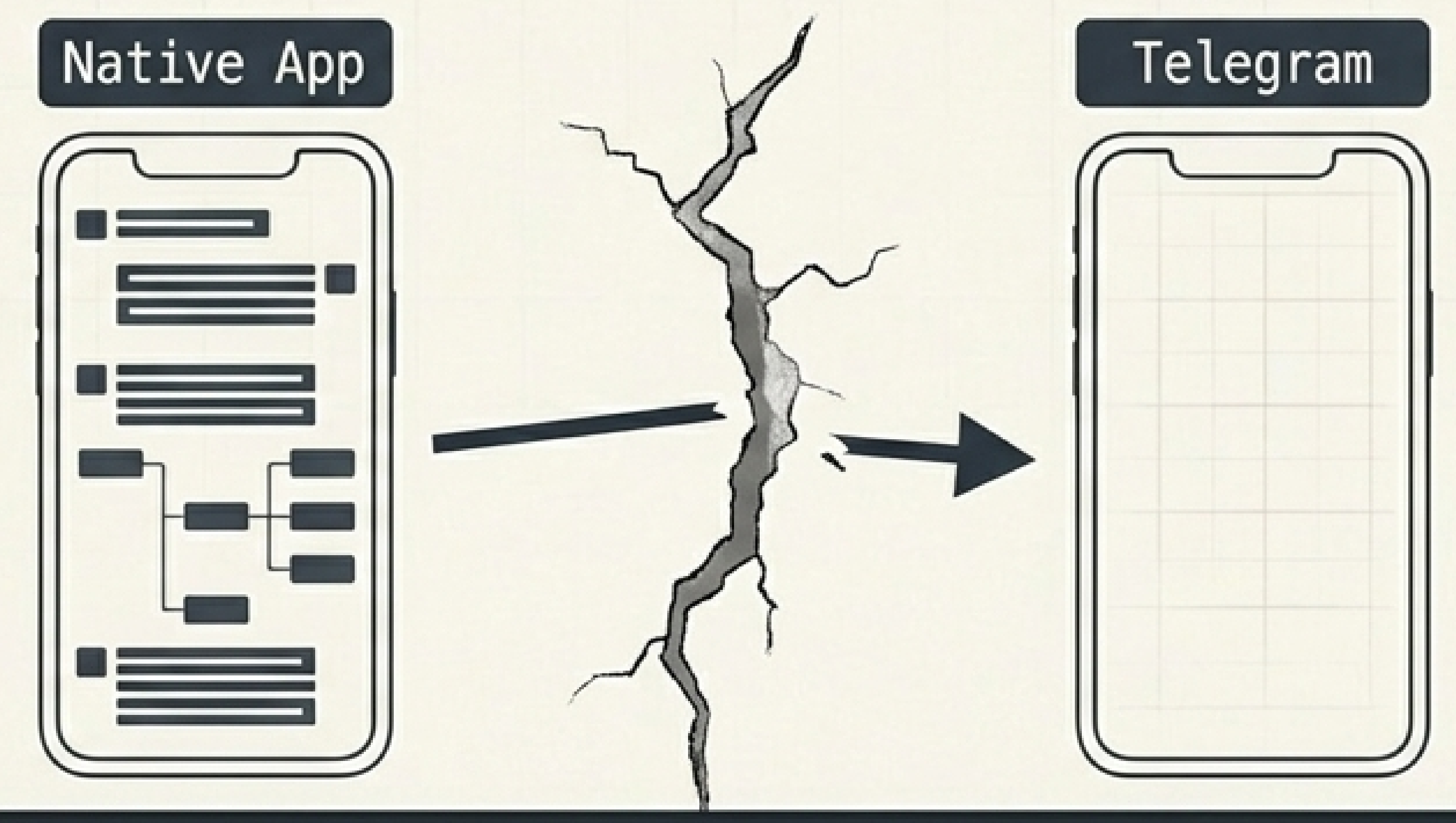
实时通信：WebSocket 替代 SSE

	[SSE] ⚠️	[WebSocket] ✅
流向 (Directionality)	单向，需另开 HTTP 发送。	天然双向，为未来实时语音铺路。
主动消息 （Proactive）	需客户端轮询或独立通道。	服务器推流和回复走统一通道。
移动端支持 (Native App)	React Native 需 Polyfill，重连有边缘情况。	Expo 支持成熟，生态完善。
连接健康检测 (Heartbeat)	依赖脆弱的应用层 Keepalive。	原生 ping/pong 帧。

NotebookLM

平台转移：为什么要砍掉 Telegram?

砍掉它不是因为它不好，而是新架构让它毫无意义。



致命问题：聊天历史不会同步到 Telegram UI。Companion 什么都知道，但对话记录在第三方平台上不可见。这不是精简版，这是一个彻底坏掉的体验。

如果跨平台触达真的重要，系统级的 Widget 比在别人的平台上做一个没有记忆界面的 Bot 更具战略意义。

NotebookLM

技术栈演进图谱

	v1		v2
前端 (Frontend)	Next.js Web + Telegram	→	Expo / React Native (完全替换) (Modified)
动画 (Animation)	CSS	→	React Native Skia (Modified)
服务端 (Backend)	Hono	→	Hono (保留) (Kept)
实时通信 (Comm)	SSE	→	WebSocket (Modified)
数据库引擎 (DB)	Supabase + Drizzle	→	Supabase + Drizzle (保留) (Kept)
向量搜索 (Vector)	pgvector	→	pgvector (保留) (Kept)

NotebookLM

Mio v2 的第一天

System Loaded (The Head Start)

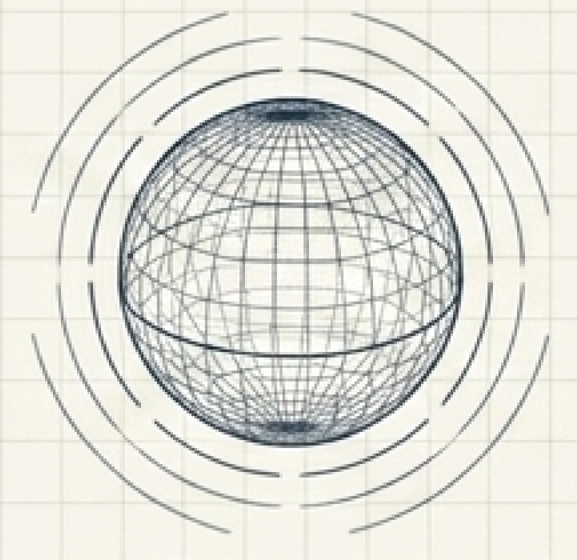
- [OK] 带 pgvector 的语义记忆系统
- [OK] 完整纯函数媒体管线
- [OK] 计费与订阅系统
- [OK] 提示词引擎

Pending Build (The Next Steps)

- [EXEC] 部署 4 张表的新 Schema
- [EXEC] 搭建 WebSocket 服务端
- [EXEC] 构建对话式 Onboarding
- [EXEC] 启动 Expo 客户端

NotebookLM

“最难的部分不是写代码，而是克制住不去重建那些已经能用的东西。”



下一步：搭起 Expo，接通 WebSocket，看着光球第一次脉动。

NotebookLM