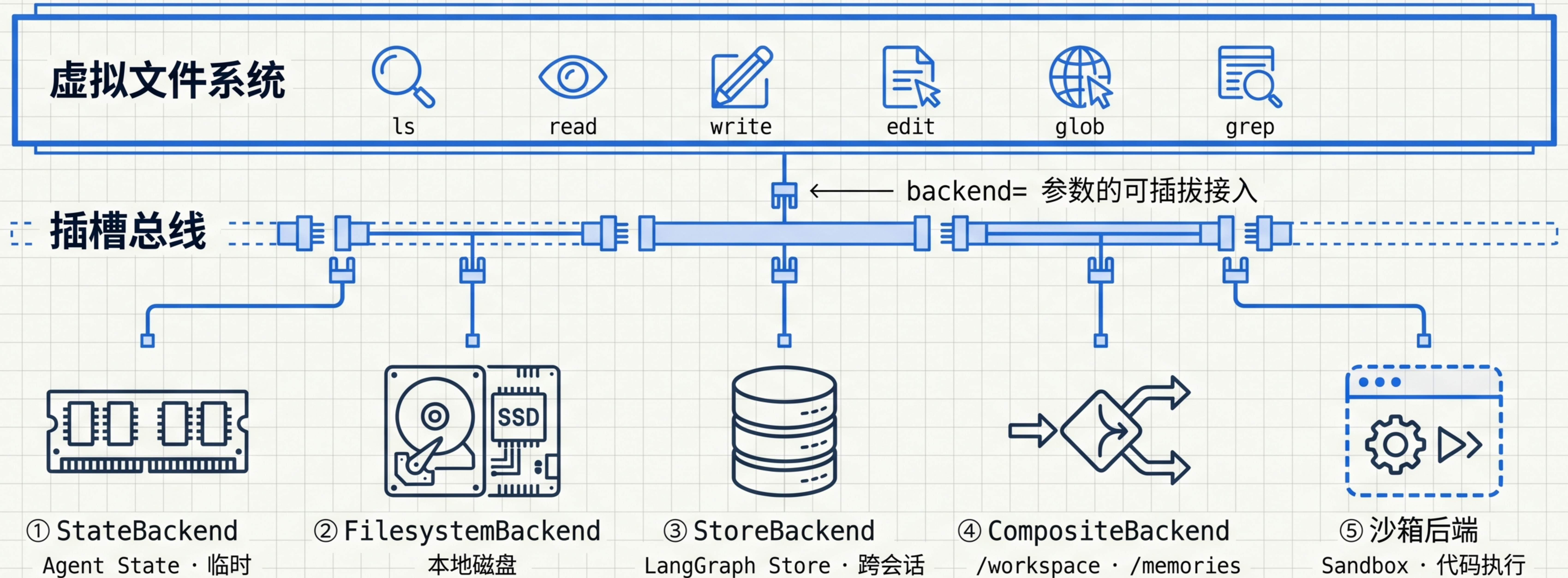


# Deep Agents 实战

## 第 5 讲：可插拔存储后端



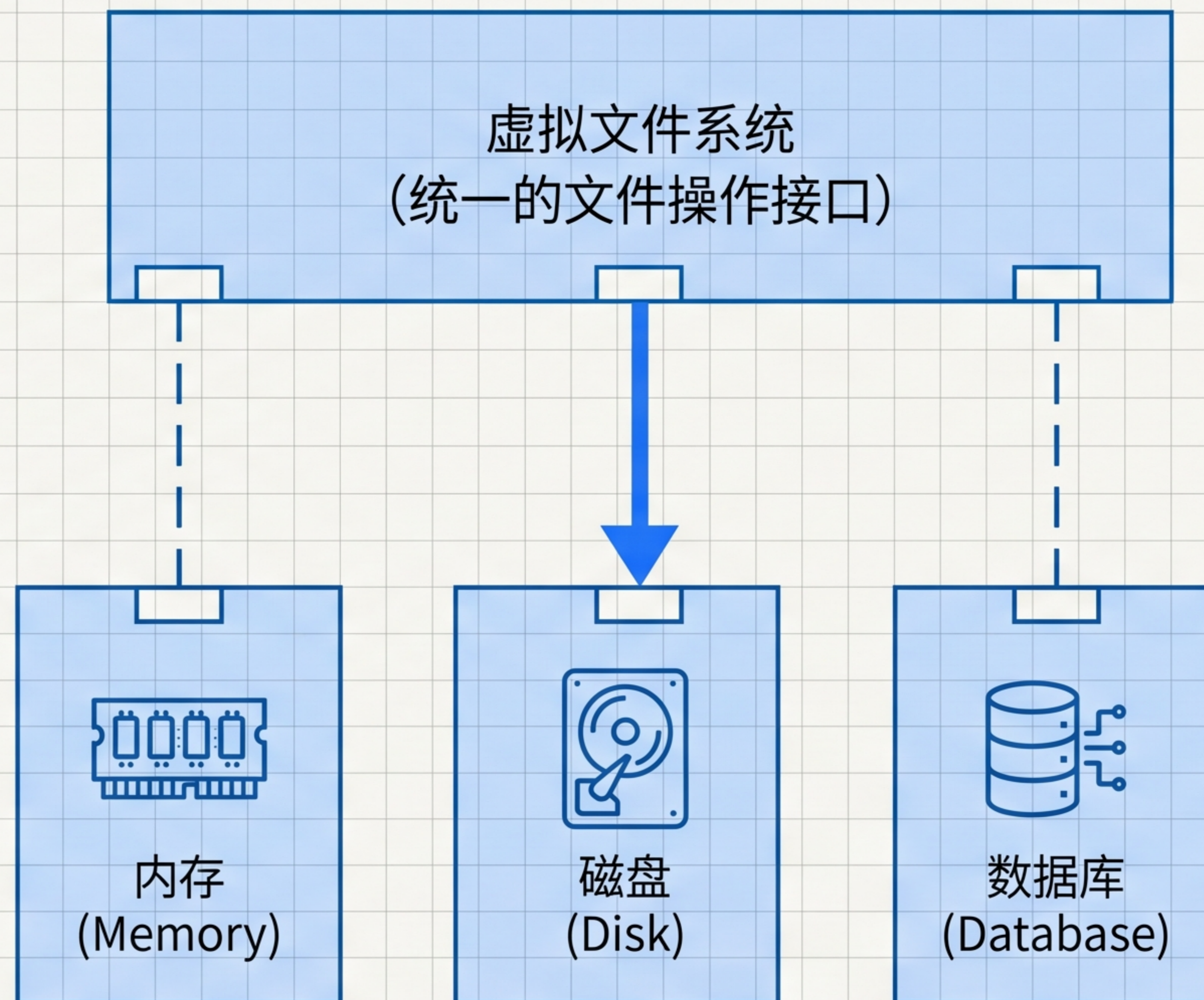
# 什么是存储后端?

## 虚拟文件系统背后的可插拔存储层

虚拟文件系统 = 统一的文件操作接口 (ls、read、write...)

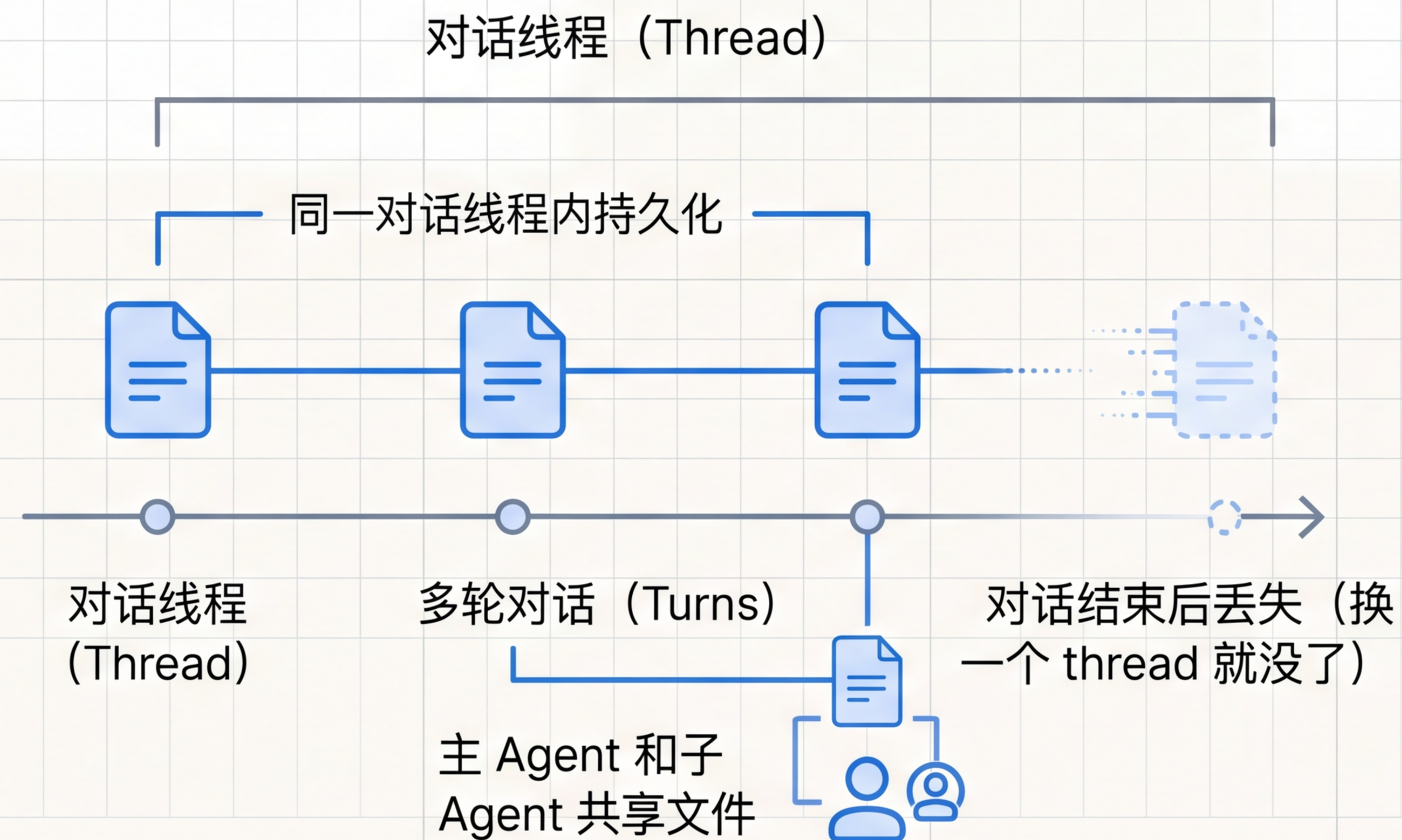
后端 = 文件实际存到哪里 (内存? 磁盘? 数据库?)

可插拔设计: 同一套工具, 切换后端即切换存储策略



# StateBackend: 临时存储 (默认)

存在 Agent State 中的'草稿纸'



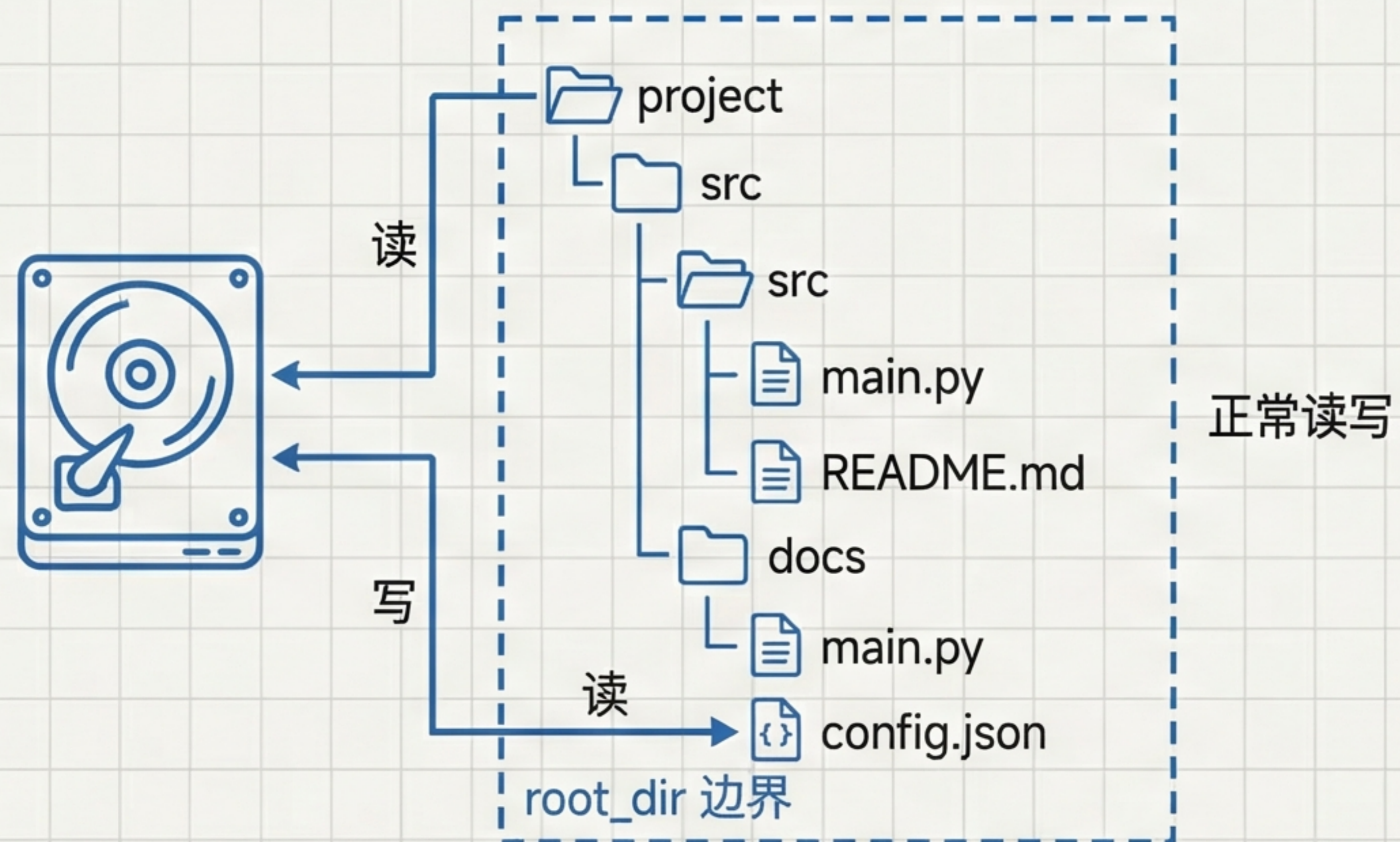
Deep Agent 接入 backend 标准写法

```
1 from deepagents import create_deep_agent
2 from deepagents.backends import StateBackend
3
4 agent = create_deep_agent(
5     model="google_genai:gemini-3.1-pro-preview",
6     backend=StateBackend(),
7 )
```

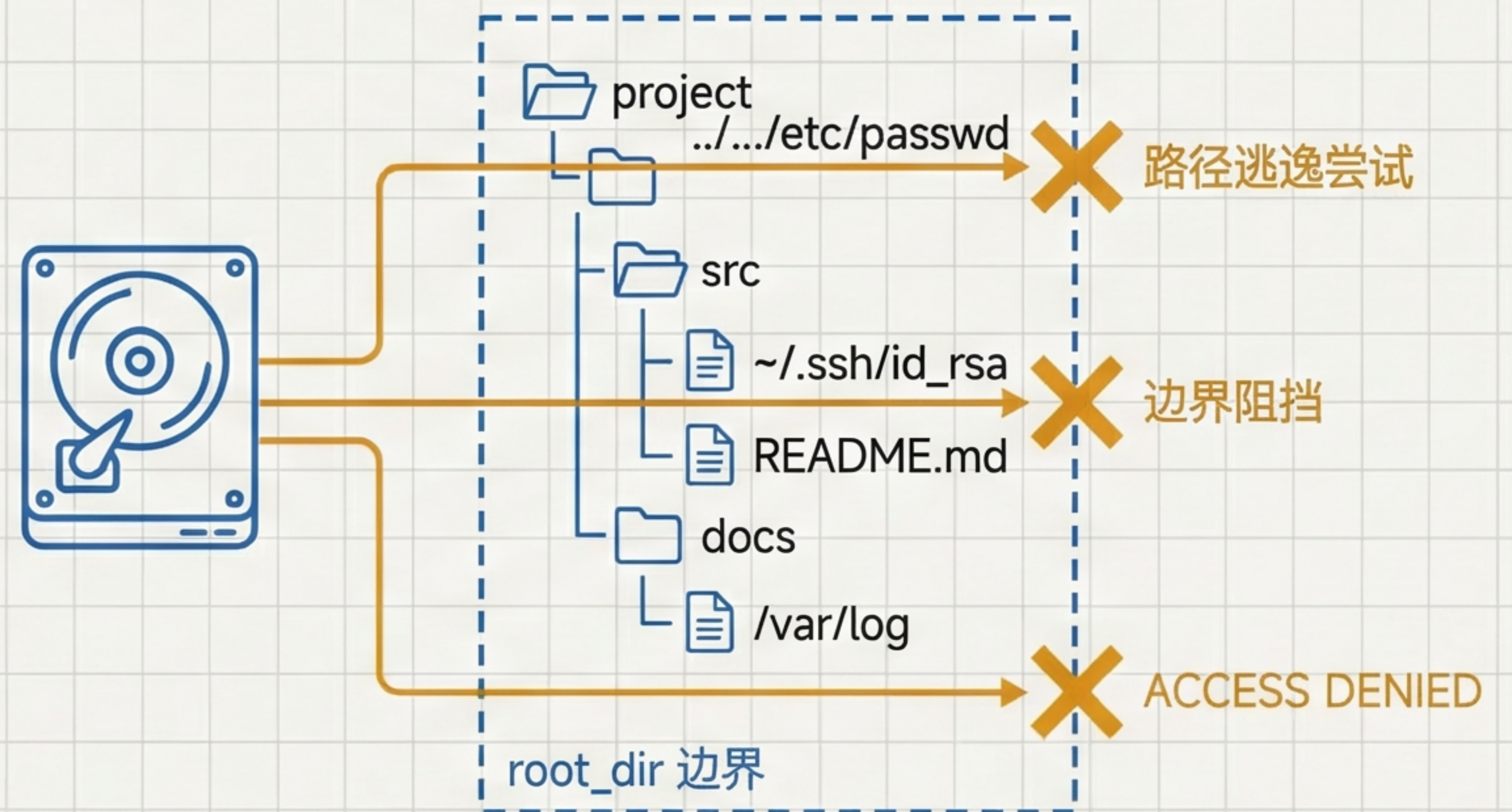
# FilesystemBackend: 本地磁盘

直接操作项目文件, virtual\_mode 保驾护航

允许的操作 (Within root\_dir)



被阻止的操作 (Path Escape Attempts)



- root\_dir 指定 Agent 可访问的根目录
- virtual\_mode=True 启用路径沙箱 (阻止 ..、~ 等路径逃逸)
- 文件修改是永久的、不可逆的

```
FilesystemBackend(root_dir="/home/user/project", virtual_mode=True)
```

# LocalShellBackend: 本地 Shell 执行

比 FilesystemBackend 多一个 execute 工具



- execute 直接在宿主机运行 Shell 命令，无任何沙箱隔离
- 支持 timeout (默认 120s)、max\_output\_bytes (默认 100,000)、env 配置

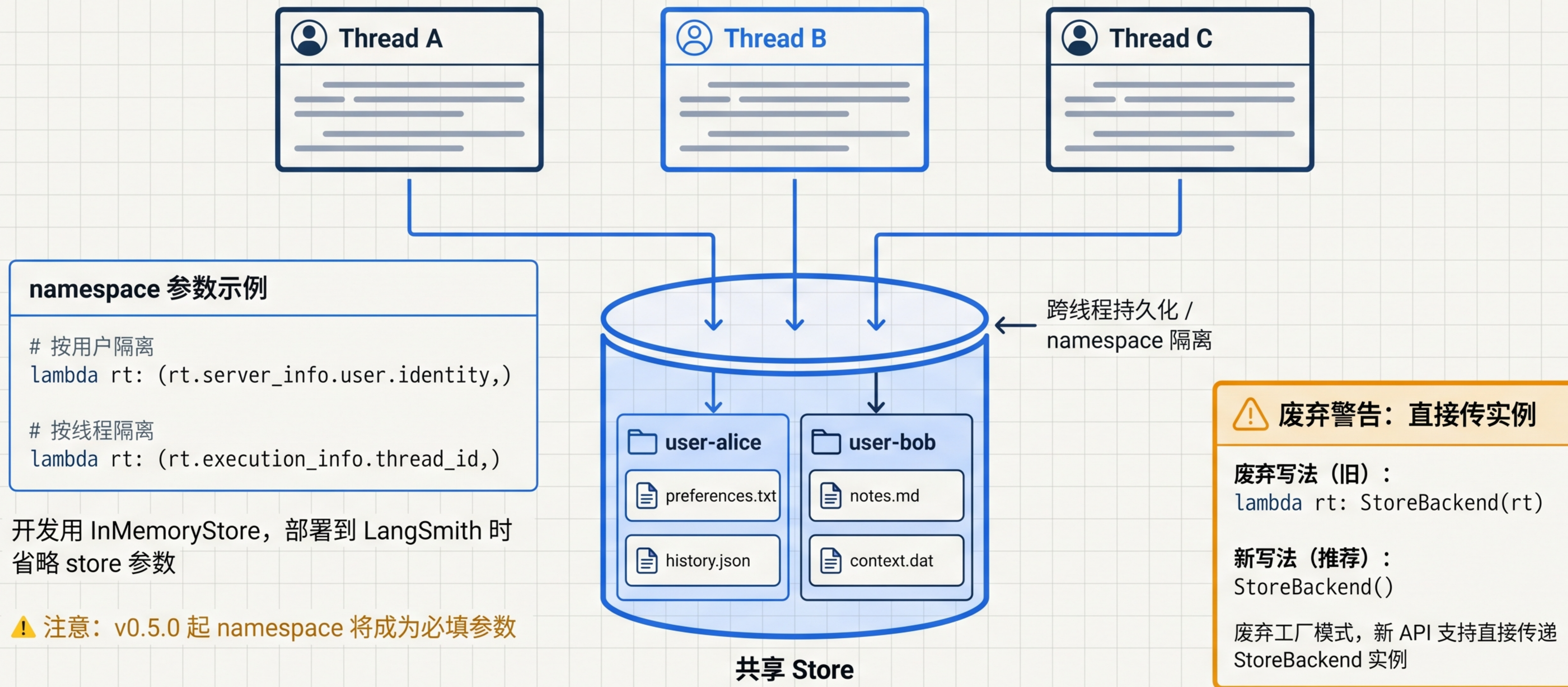
**⚠ 极高风险: 可执行任意命令, 含删除文件、外传数据**

适用场景: 本地开发机的编程助手 (你完全信任 Agent 时)

生产环境替代方案: 使用沙箱后端 (本讲后续介绍)

# StoreBackend: 跨会话持久化

基于 LangGraph Store, 不同对话都能访问



# StoreBackend : namespace 三种模式

## 用 Runtime 对象控制数据隔离粒度

### 1 按用户隔离 (最常用)

```
backend = StoreBackend(  
    namespace=lambda rt:  
    (rt.server_info.user.identity),  
)
```

→ 每个用户独立存储, 互不干扰

适用: 多用户 SaaS、个人偏好存储

### 2 按对话线程隔离

```
backend = StoreBackend(  
    namespace=lambda rt:  
    (rt.execution_info.thread_id),  
)
```

→ 每次对话独立存储, 会话结束  
数据留存


适用: 每次会话需要独立工作区

### 3 按助手隔离 (所有用户共享)

```
backend = StoreBackend(  
    namespace=lambda rt:  
    (rt.server_info.assistant_id),  
)
```

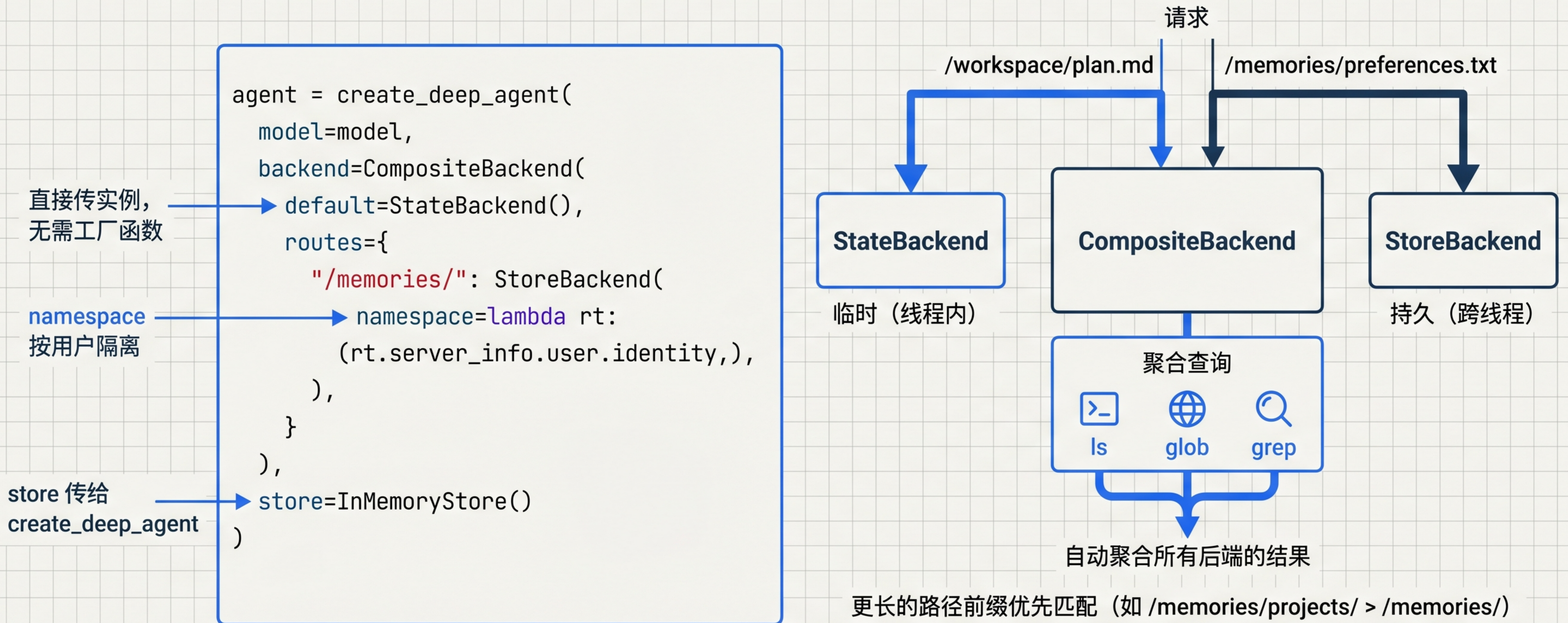
→ 同一助手的所有用户共享数据

适用: 共享知识库、产品文档

 v0.5.0 起 namespace 为必填参数

# CompositeBackend：路径路由

不同路径走不同后端，兼顾临时与持久



# 五种后端代码速查

把 backend= 参数换成这些即可

## ① StateBackend (默认, 无需传参)

```
from deepagents.backends import StateBackend
agent = create_deep_agent(
    model="google_genai:gemini-3.1-pro-preview",
    backend=StateBackend(), # 省略也等价于默认
)
```

## ② FilesystemBackend (本地磁盘)

```
from deepagents.backends import FilesystemBackend
backend=FilesystemBackend(
    root_dir=".", ← # 根目录 (绝对路径)
    virtual_mode=True, ← # 开启路径沙箱
)
```

## ③ LocalShellBackend (本地 Shell, 谨慎使用)

```
from deepagents.backends import LocalShellBackend
backend=LocalShellBackend(
    root_dir=".", ← # 根目录
    env={"PATH": "/usr/bin:/bin"}, ← # 环境配置
)
```

## ④ StoreBackend (跨会话持久化)

```
from deepagents.backends import StoreBackend
backend=StoreBackend(
    namespace=lambda rt: (rt.server_info.user.identity,), ← # 命名空间
)
# create_deep_agent 还需传 store=InMemoryStore()
```

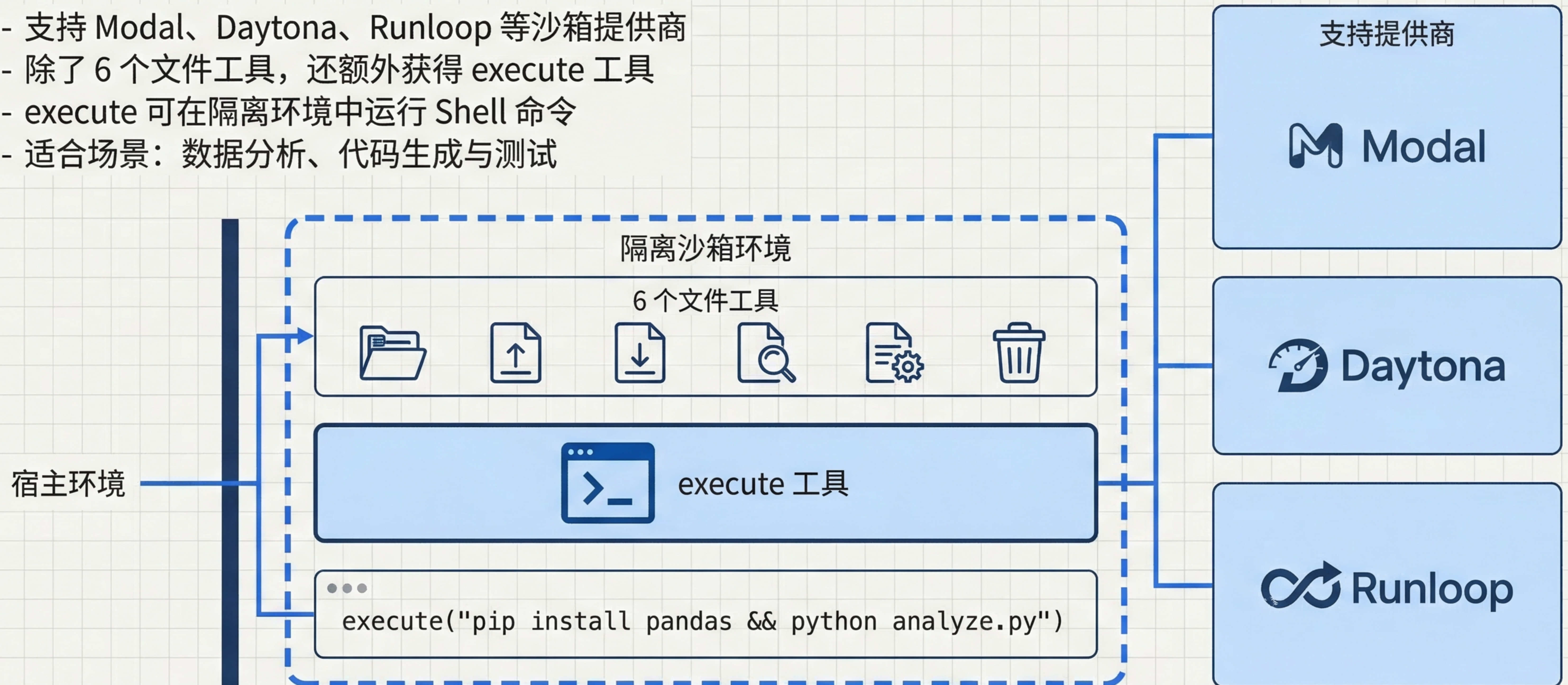
## ⑤ CompositeBackend (混合路由)

```
from deepagents.backends import CompositeBackend
backend=CompositeBackend(
    default=StateBackend(), ← # 默认后端
    routes={"/memories/": StoreBackend( ← # 路由配置
        namespace=lambda rt: (rt.server_info.user.identity,),
    )},
)
```

# 沙箱后端：安全代码执行







隔离环境 + 额外的 execute 工具

- 支持 Modal、Daytona、Runloop 等沙箱提供商
- 除了 6 个文件工具，还额外获得 execute 工具
- execute 可在隔离环境中运行 Shell 命令
- 适合场景：数据分析、代码生成与测试



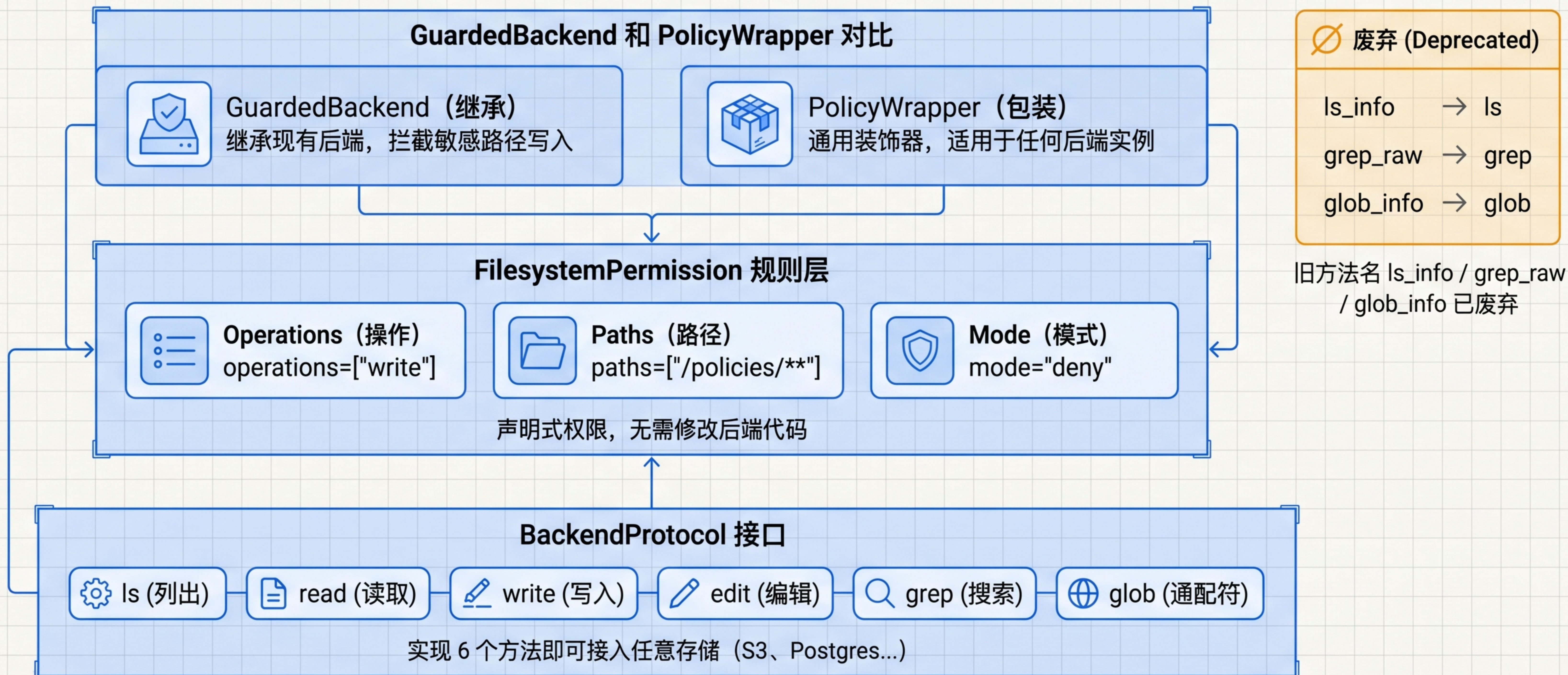
# 后端选择指南

根据场景选择最合适的存储策略

	场景	推荐后端	关键特点
	学习和实验	StateBackend (默认)	零配置, 自动清理
	本地编程助手 (只读写文件)	FilesystemBackend	root_dir + virtual_mode 保护
	本地编程助手 (需执行命令)	LocalShellBackend	文件 + execute, 仅限开发机
	需要跨会话记忆	CompositeBackend	混合 State + Store
	需要执行代码 (生产安全)	沙箱后端	隔离执行, API 密钥不入沙箱
	生产部署	StoreBackend / Composite	持久化 + namespace 隔离

# 自定义后端与安全策略

BackendProtocol + FilesystemPermission + PolicyWrapper



# 本讲回顾

-  1. 后端是虚拟文件系统的可插拔存储层——同一套工具，切换后端即切换存储策略
-  2. 六种后端：State（临时）→ Filesystem（本地磁盘）→ LocalShell（本地 Shell）→ Store（持久化）→ Composite（混合路由）→ 沙箱（隔离执行）
-  3. StoreBackend 新 API：直接传实例，namespace 工厂隔离多用户数据
-  4. namespace 三种粒度：按用户 / 按线程 / 按助手
-  5. 权限控制：FilesystemPermission 声明式；PolicyWrapper 自定义策略
-  6. BackendProtocol 方法名更新：ls / grep / glob（旧名已废弃）

下一讲 →

**第 6 讲：任务规划与中间件**

下一讲：任务规划与中间件——  
让 Agent 学会拆解复杂任务