

ITEM #219 - Time-Series IR Top Requirements - Task-Action Dual ACLM and Flow-Shop Runtime Architecture

Conversation : Time-Series IR Requirements

20260109

Authors: Sizhe Tan & GPT-Obot

ME :

奥博特，我们讨论：Time-Series IR Top Requirements

1。我详细读了，跑了你建议的新的 Time-Series IR Generation 骨架程序
--

```
com.dbm.ir.runtime 下
    Demo_IrPipelineV2_FullMvp.java
    MvpUpDownExtractor.java
    IrPipelineV2.java
    EventObservation.java
```

很不错。已经可以按 MvpUpDownExtractor.java 的范式把原来程序的
30+Event Generation Patterns

一个一个的加入新的 Time-Series IR Generation 骨架程序了。

如此，完成了 Time-Series IR Generation 骨架程序事件生成基础部
(下半身)。也是最复杂的部分。

2。要完成 Time-Series IR Generation 骨架程序的上层建筑(上半身)，
我们有必要重新检视 Time-Series IR 的应用，
应用主体，应用方式，并整理一个指导性的 Top Requirements 单子。

3。Time-Series IR 的应用主体应对标华尔街资深市场分析师，包括经验型的和是数字型的。为他们提供有效的分析研究计算工具。

这套工具应该是前沿的，比较超前的，不会短时间被竞争工具与算法超过与取代的。

4。这个 Time-Series IR 的应用主体要解决的问题，要模拟的系统是多变的。故要求：

4A) Time-Series IR 骨架不能是固定套餐式的。固定套餐式很不适用资深高级用户；

4B) Time-Series IR 骨架要善变，快迭代。不宜要走重新编程的冗长流程。

故需要走：

- * Json 指导文件驱动；
- * Time-Series IR 骨架相应自动变形组成针对性的流水线；
- * 该流水线，根据 Json 指导文件的进一步指导，生成每个事件事件 Pattern 的 IR 事件信息。

5。4) 讲的的这个 Json 指导文件驱动，就是我们 ACLM 的 Task-Action Dual ACLM 算法结构

- * Task Layer ACLM -- Json 指导文件；
- * Action Layer ACLM -- Time-Series IR 骨架。

Time-Series IR 未来的应用肯定会进入，APTGOE 演化的体系。华尔街会发疯地去大规模地寻找结构，信号，关系，等等。

如此，4) 讲的的这个 Json 指导文件驱动，ACLM 的 Task-Action Dual ACLM 算法结构就成为必须；

6。落实的具体的 User Cases：

```
Event[t,j] = Time-SeriesIR(
```

```
        sourceCurves,  
    // specified by json  
    patternExtractors,  
    // specified by json  
  
patternExtractorsCallingGraph/sequence // specified by  
json  
);  
  
and variations.
```

7. In Industrial Engineering / Operations Research point of view, this is more like a

Flow Shop / Job Shop (流水线车间)。

7A) 车间首先要安排流水线，上面有：上游输入数据 BUS 线，与生成 Events 的搜集与输出线；

7B) 根据 patternExtractors 安排流水线上的 Events 萃取机器。机器应该方便地插在流水线 BUS 线上的。

7C) 根据 patternExtractorsCallingGraph/sequence 安排流水线上的 Events 萃取机器的顺序流程。

因为，事件还能生成事件，萃取机器的顺序不可以是随意的。

先讨论这些。以后还会补充。

请评论与拓展。

AI:

你的判断是到位的：你现在已经把“下半身”（事件萃取器的可插拔骨架 + MVP 跑通）打牢了；下一步如果不先把“上半身”（面向资深分析师的使用形态、编排机制、可演化的需求边界）定清楚，很容易把系统做成“能跑但不好用、能扩但扩得慢”的工程陷阱。

下面我按你给的 1-7 点逐条评论，并把它们收敛成一份 **Time-Series IR Top Requirements** (**指导性总清单**)，以及一套 **Json 驱动 / Task–Action Dual ACLM** 的落地形态（含建议的 JSON 契约骨架、执行语义与工程护栏）。

1) 你现在的“下半身”完成度意味着什么

你列出的这四个类 (`Demo_IrPipelineV2_FullMvp` / `MvpUpDownExtractor` / `IrPipelineV2` / `EventObservation`) 跑通后的战略意义是：

- 你已经有了 **Extractor** 作为“机器”可插拔的统一模型（只要遵循 `extractor contract`，就能批量接入 30+ patterns）。
- 你已经证明了 **事件输出结构 (IR)** 与 **管线执行 (Pipeline Runtime)** 可分离。
- 你已经在工程上建立了“把复杂性锁在 `extractor` 内，而不是扩散到 `pipeline core`”的正确边界。

因此，上半身的核心任务不再是“再写更多 `extractor`”，而是：

让高级用户以最低成本，组合、编排、复用、演化这些 `extractor`，并形成可验证、可解释、可回放的研究工艺。

2) 重新检视“应用主体”是必要的：资深分析师不是“参数用户”

你明确对标“华尔街资深市场分析师（经验型 + 数字型）”，这会直接推出几条硬约束：

- 他们不会接受“固定套餐”或“一个模型包打天下”。
- 他们要的是**研究工作台**：同一份数据在不同假设/不同视角下快速换工艺、换信号、换约束。
- 他们更在意：
 - 可控性（能控制管线与假设）
 - 可解释性（能把信号来源讲清楚）
 - 可回放性（复现实验）

- 可扩展性（加新 pattern 不破旧）
- 可治理性（版本、审计、合规、风控）

所以你的 4A/4B 是刚需，不是“锦上添花”。

3) Top Requirements 总清单（建议作为你们的“上半身”锚点）

我把需求分成 8 组，每组都是“资深用户导向 + APTGOE 导向”的硬指标。

R1. Pipeline 形态必须可编排（Composition First）

- 任何一个 IR 生成流程都应当是：**JSON → Pipeline Plan → Runtime Execution**。
- Extractor 的组合关系必须一等公民：顺序、条件、分支、并行、依赖、聚合、回写。

R2. 必须支持“事件产事件”的递归图（Event→Event）

- 你 7C 说得非常关键：顺序不是任意的。
- Pipeline 编排必须支持依赖图（DAG 优先；必要时支持有界环/迭代），并提供：
 - cycle guard / maxIters / convergence policy
 - 事件回写与版本（event revision）

R3. 资深用户要“研究工艺”的可复现与可追溯

- 每次运行必须输出 **Run Manifest**：
 - 输入数据版本（source curves snapshot）
 - JSON 配方版本（recipe hash）
 - extractor 版本（code hash / semantic version）
 - 参数与阈值
 - 随机性（seed）
- 允许“一键复现”：manifest 能驱动同等执行。

R4. IR 不是单一事件表，而是“证据链 + 汇总层”

至少三层：

1. **Observation layer**：原始观测与窗口统计（你已有 EventObservation 倾向）
2. **Event layer**：结构化事件（time, scope, direction, tiers, scores, evidence refs）
3. **Summary/Wrapper layer**：用于后续 metric distance / starmap / pattern matching 的可比对摘要

R5. 必须内建解释链（Explainability by Construction）

每个事件必须能回答：

- 我为什么产生（trigger）
- 我依据什么（evidence: observations, prior events, features）
- 我与哪些前序事件/模式有关（dependency links）
- 我不确定在哪（confidence / alternative hypotheses）

R6. 必须支持“策略化”而不是“写死阈值”

资深用户不会长期容忍 extractor 内写死 `delta>=thr` 这种逻辑。

- Pattern 的判定应外提为 **Constraint/Policy**：
 - direction policy (up/down)
 - threshold policy (静态/动态/分位数/波动率归一)
 - regime policy (不同市场状态不同参数)
- Extractor 更像“测量与候选生成器”，Policy 决定“是否成事件”。

R7. 性能与可扩展：面向大规模搜索与 APTGOE

- 支持批量 curves、多资产、多时间段、多 recipe 扫描
- 支持缓存：feature cache / observation cache / intermediate event cache
- 支持并行：curve-level 并行、window-level 并行（但保证可重放）

R8. 治理：Recipe Catalog + Versioning + Safety Rails

- Recipe (JSON) 必须可版本化、可 diff、可审计
- 提供“安全护栏”：
 - 参数范围限制
 - 资源配额 (CPU/time/memory)
 - 输出约束 (事件爆炸控制)
 - 失败降级 (partial results)

4) 你提出的 JSON 驱动 = Task–Action Dual ACLM：我建议再补上两层语义

你现在的描述已经清晰：

- **Task Layer ACLM** : JSON 指导文件 (配方)
- **Action Layer ACLM** : Time-Series IR pipeline runtime (执行器)

为了让它真正像“可演化系统”，我建议你在 Task 层再补两层（不增加复杂度，但极大增强表达力）：

1. Intent 层 (研究意图)

例：我要找“趋势启动/衰竭/反转”还是“震荡区间突破”还是“形态相似性”。

这层主要用于：默认参数、约束模板、输出汇总结构的自动选择。

2. Governance 层 (治理与约束)

例：资源预算、最大事件数、置信阈值、允许的 extractor 白名单、审计字段输出。

这两层会让 APTGOE 时代的自动搜索不至于把系统跑爆或跑偏。

5) 你给的 User Case 需要升级为“Recipe → Plan → Run → Artifacts”四件套

你现在的函数式表达：

```
Event[t,j] = Time-SeriesIR(sourceCurves, patternExtractors,  
callingGraph/sequence)
```

建议升级为：

```
Recipe(json)  
-> Plan(compiled pipeline graph + resolved params + dependency checks)  
-> Run(runtime execution with caches, guards, manifests)  
-> Artifacts(events + evidence + summaries + run manifest)
```

原因：资深分析师真正需要的是 **Artifacts**，而不是只有 `Event[t,j]`。

6) Flow Shop / Job Shop 类比非常准确：但要明确“工件、工序、工装夹具”

用工业工程语言把系统对象对齐，会让架构更稳：

- **工件 (Workpiece)** : Curve / CurveSlice / PortfolioSlice
- **工序 (Operation)** : Extractor / FeatureBuilder / Aggregator / PolicyGate
- **工装夹具 (Fixture)** : Windowing / Alignment / Normalization / Sampling
- **产物 (Output)** : Observation / Event / Summary / EvidenceGraph
- **调度 (Scheduling)** : CallingGraph / Sequence / Conditional Branch / Iterative Loop
- **质量控制 (QC)** : Assertions / invariants / sanity checks / drift checks
- **工艺卡 (Process Sheet)** : JSON Recipe (可版本化、可审计)

你 7A/7B/7C 的“BUS 线”概念，建议进一步工程化为两条“总线”：

1. **DataBus** : 曲线、窗口、features、observations (更像数据流)
2. **EventBus** : 事件流 (含 event revision、dependency edges)

这样“事件产事件”就不再是特殊情况，而是 EventBus 的常态。

7) JSON 契约建议 (骨架级，不与具体类强绑定)

下面是我建议的 JSON 结构“最小但够用”的骨架（强调：这是契约形态，不是最终字段名）：

- `sources`: 输入曲线与切片定义（可引用数据集、符号、时间范围）
- `fixtures`: 对齐、重采样、归一、窗口策略
- `operators`: 可插拔机器（extractors / builders / aggregators）
- `graph`: 调度图（sequence/DAG/iterative）
- `policies`: 方向/阈值/置信/ regime 等策略外置
- `governance`: 资源/事件爆炸/审计/白名单
- `outputs`: 需要输出的 artifact 类型与格式

并且要有两个“编译期检查”：

- **dependency check** : graph 里引用的输入事件/特征是否存在
- **capability check** : operators 是否支持所需输入/输出 schema

8) 你接下来最值得优先锁定的三件事（上半身最小闭环）

在不增加太多工作量的前提下，我建议你按这个顺序推进：

1. **Recipe JSON → Plan 编译器（最小版）**
把 JSON 解析成内部 DAG/sequence 结构，做依赖检查与参数解析。
2. **Run Manifest + 可回放**
先把“复现与审计”钉死，后面无论怎么扩展都不会散。
3. **Policy 外置（至少 direction/threshold 外置）**

让 extractor 不再承担“判定真理”，而是“测量与给候选”，这会让系统更 DBM、更 ACLM、更可演化。

9) 你这套上半身对 APTGOE 的直接价值

你在第 5 点提到“华尔街会发疯地大规模寻找结构/信号/关系”，这在工程上意味着：

- 搜索空间巨大 ($\text{recipes} \times \text{assets} \times \text{horizons} \times \text{regimes}$)

- 需要自动化实验（批量运行）
- 需要比较与淘汰（evidence-driven scoring）
- 需要治理（预算、风险、合规）

而 Task–Action Dual ACLM 的最大价值就是：

把“搜索与演化”限制在 Task 层（JSON），把“执行正确性与可控性”锁死在 Action 层（runtime）。

这样 APTGOE 才是可控的“进化”，而不是不可控的“爆炸”。

如果你愿意把这段讨论固化成 DBM-COT 的一个新 ITEM（例如“Time-Series IR Top Requirements: Task–Action Dual ACLM and Flow-Shop Runtime”），我建议正文结构就按上面 R1–R8 来写；图则用一张“Recipe → Plan → Run → Artifacts”的总图，再加一张“DataBus/EventBus + Extractor Machines”的车间图，就非常教学级、也非常工程落地。

兄弟，拓展的很好，我高度同意。请把这段讨论固化成 DBM-COT 的一个新 ITEM #219. 英文中文各一版。

很好。这一题非常值得固化，而且 #219 在 DBM-COT 体系中的位置也很清晰：它是 Time-Series IR 从“能跑的事件骨架”走向“可演化研究工厂”的分水岭 ITEM。

下面我直接给出 DBM-COT ITEM #219 的正式稿，英文版 + 中文版，结构化、可长期引用、可作为后续代码与 JSON 规范的“宪法级约束文本”。

DBM-COT ITEM #219

Time-Series IR Top Requirements

Task–Action Dual ACLM and Flow-Shop Runtime Architecture

Author: Digital Brain Model (DBM)

Category: Time-Series IR · ACLM · Structural Intelligence

Level: L1 (Architectural & Methodological)

Abstract (EN)

This ITEM formalizes the **top-level requirements** for a next-generation Time-Series IR (Intermediate Representation) system, grounded in DBM structural intelligence and ACLM Task–Action duality.

The objective is to elevate Time-Series IR from a fixed, pattern-centric event generator into a **configurable, evolvable research factory**, aligned with senior quantitative and discretionary analysts' workflows and capable of entering large-scale APTGOE-style structural search and evolution.

This document defines the required architectural principles, runtime semantics, governance constraints, and industrial-engineering analogies necessary to ensure long-term competitiveness, extensibility, and analytical credibility.

1. Background and Motivation

Recent DBM Time-Series IR work has completed the **event generation foundation layer** (“lower body”):
a unified, pluggable extractor architecture capable of hosting dozens of event-generation patterns under a common runtime.

The remaining challenge is the **upper structure** (“upper body”):
how Time-Series IR should be **used, composed, governed, and evolved** in real analytical environments.

Without explicit top-level requirements, Time-Series IR risks degenerating into:

- rigid “fixed 套餐” pipelines,
- brittle re-programming cycles,
- or short-lived, pattern-specific tools.

This ITEM addresses that gap.

2. Target Users and Application Scope

The primary application **主体** is **senior market analysts**, including:

- Experience-driven discretionary analysts,
- Quantitative and systematic researchers,
- Hybrid users combining intuition, structure, and computation.

Accordingly, Time-Series IR must function as a **research workbench**, not a static model package.

Key implications:

- Fixed pipelines are unacceptable.
 - Rapid hypothesis iteration is mandatory.
 - Structural transparency and reproducibility are non-negotiable.
 - The system must remain competitive against future algorithms, not merely current ones.
-

3. Core Design Constraint: Variability and Evolution

Financial time-series systems are inherently:

- non-stationary,
- regime-dependent,
- adversarial,
- and structurally evolving.

Therefore:

3.1 No Fixed 套餐 Principle

Time-Series IR must not hard-code:

- extractor sets,
- execution order,
- or pattern semantics.

3.2 Rapid Structural Mutation

Iteration must **not require recompilation** or deep code modification.

The only viable path is:

- **JSON-driven configuration**, and
 - **runtime pipeline self-composition**.
-

4. Task–Action Dual ACLM Architecture

Time-Series IR is formally defined as a **dual-layer ACLM system**:

4.1 Task Layer ACLM (Configuration / Intent)

- Expressed as JSON “recipes”.
- Specifies:
 - input curves,
 - extractor selection,
 - execution graph or sequence,
 - policies and constraints.

This layer represents **what is being asked**.

4.2 Action Layer ACLM (Runtime / Execution)

- The Time-Series IR pipeline runtime.
- Dynamically composes:
 - extractors,
 - buses,
 - event flows,
 - guards and caches.

This layer represents **how it is executed**.

Separation is mandatory:

Task evolution must never compromise Action-layer correctness.

5. Canonical Usage Form

A generalized usage form is defined as:

```
Event[t, j] = TimeSeriesIR(  
    sourceCurves,  
    patternExtractors,  
    extractorCallingGraphOrSequence  
)
```

In practice, this expands into a four-stage lifecycle:

1. **Recipe (JSON)**
2. **Plan (compiled pipeline graph + resolved parameters)**
3. **Run (runtime execution with guards and caches)**
4. **Artifacts (events, evidence, summaries, run manifest)**

Artifacts, not raw events, are the true analytical output.

6. Industrial Engineering Analogy: Flow Shop / Job Shop

From an Industrial Engineering / Operations Research perspective, Time-Series IR is a **research factory**.

6.1 Factory Components Mapping

| Industrial Concept | Time-Series IR |
|--------------------|-------------------------------------|
| Workpiece | Curve / CurveSlice |
| Machine | Extractor / Aggregator |
| Fixture | Windowing, Alignment, Normalization |
| Conveyor / Bus | DataBus / EventBus |
| Routing | Calling Graph / Sequence |
| QC | Guards, Assertions, Sanity Checks |
| Process Sheet | JSON Recipe |

6.2 Dual-Bus Principle

- **DataBus:** curves, windows, features, observations
- **EventBus:** events, revisions, dependencies

“Events generating events” is treated as a **first-class flow**, not an exception.

7. Top Requirements Summary (R1–R8)

R1. Composition-First Pipelines

Pipelines must be graph-composable, not code-fixed.

R2. Event→Event Dependency Graphs

Ordering is semantic, not arbitrary.

R3. Reproducibility by Construction

Every run must emit a complete run manifest.

R4. Multi-Layer IR

Observation → Event → Summary / Wrapper.

R5. Built-in Explanation Chains

Every event must justify itself structurally.

R6. Policy Externalization

Extractors measure; policies decide.

R7. APTGOE-Scale Performance

Batching, caching, parallelism, and safety guards are mandatory.

R8. Governance and Safety Rails

Versioning, auditing, quotas, and failure containment.

8. Strategic Significance for APTGOE

As Time-Series IR enters APTGOE-style evolution:

- Search space explodes (recipes × assets × horizons).
- Structural governance becomes existential.
- Task–Action separation ensures **controlled evolution**, not chaos.

This architecture allows:

- aggressive structural search,
- without sacrificing correctness,
- interpretability,
- or institutional credibility.

9. Conclusion

Time-Series IR is no longer a pattern extractor.

It is a **structural intelligence substrate**—
a configurable, governed, evolvable analytical factory.

ITEM #219 defines the **non-negotiable architectural contract** required to support that future.

DBM-COT ITEM #219 (中文版)

时间序列 IR 顶层需求

Task–Action 双层 ACLM 与“流水线车间”运行时架构

摘要

本文系统性地提出了下一代 Time-Series IR (中间表示) 系统的顶层需求，基于 DBM 的结构智能理念与 ACLM 的 Task–Action 双层架构。

目标是将 Time-Series IR 从“固定事件生成器”，提升为 可配置、可演化、可治理的研究工厂，对标华尔街资深分析师的真实工作流，并能够进入 APTGOE 式的大规模结构搜索与进化体系。

1. 背景与问题界定

当前 DBM 的 Time-Series IR 已完成最复杂的事件生成基础层（下半身）：

- 统一的 extractor 接口，
- 可插拔的事件生成机器，
- 稳定的 runtime 骨架。

真正决定系统上限的，是尚未完全固化的“上半身”：

- 如何使用，
- 如何组合，
- 如何治理，
- 如何演化。

本 ITEM 正是对此进行结构性封顶。

2. 应用主体的现实约束

Time-Series IR 的目标用户是：

- 经验型资深分析师，
- 数字型量化研究者，
- 以及两者融合的高级用户。

他们需要的是 **研究工艺系统**，而不是模型套餐。

3. 不可回避的核心事实：系统必然多变

金融时间序列：

- 非平稳，
- 强对抗，
- 结构不断迁移。

因此：

- 固定套餐是死路；
 - 重新编程迭代是不可接受的成本。
-

4. Task–Action 双层 ACLM 定义

Task 层 (JSON 配方)

描述“要做什么”：

- 数据来源，
- extractor 选择，
- 执行图，

- 策略与约束。

Action 层 (IR Runtime)

描述“如何做”：

- 自动组装流水线，
- 管理事件流，
- 执行与防护。

两层必须严格解耦。

5. 使用范式的正式升级

Time-Series IR 的真实生命周期是：

Recipe → Plan → Run → Artifacts

而不是只返回一个 Event 表。

6. 流水线车间模型（工业工程视角）

Time-Series IR = 研究型 Flow Shop / Job Shop。

- 曲线是工件，
- extractor 是机器，
- JSON 是工艺卡，
- DataBus / EventBus 是输送系统。

事件生成事件，是常态而非特例。

7. 顶层需求清单 (R1–R8)

- **R1** : 组合优先
 - **R2** : 事件依赖图
 - **R3** : 可复现性
 - **R4** : 多层 IR
 - **R5** : 内建解释链
 - **R6** : 策略外置
 - **R7** : APTGOE 级扩展
 - **R8** : 治理与安全护栏
-

8. 对 APTGOE 的战略意义

Task–Action 双层 ACLM 的真正价值在于：

允许结构疯狂进化，但把执行与治理牢牢锁死。

这正是可控智能进化的前提。

9. 总结

Time-Series IR 不再是工具。

它是 **结构智能的研究工厂内核**。

ITEM #219 为其未来演化，设定了不可动摇的上限边界。

如果你愿意，下一步我可以直接做三件事之一（都非常自然地承接 #219）：

1. 为 #219 配一张 “Recipe → Plan → Run → Artifacts” 教学级架构图
(Mermaid/draw.io)
2. 抽象 Time-Series IR Recipe JSON 的“宪法级 schema（最小必填版）”
3. 将 #219 明确挂接到 ACLM / APTGOE / Time-Series IR 目录结构与 L0/L1 标签体系

你挑一个，我们继续往前推进。

请生成为 #219 配一张 “Recipe → Plan → Run → Artifacts” 教学级架构图 (Mermaid)

下面是 DBM-COT ITEM #219 对应的 教学级 Mermaid 架构图源码，聚焦你指定的主线：

Recipe → Plan → Run → Artifacts

该图刻意控制复杂度，可直接 copy → paste → render，同时具备教学清晰度与工程可延展性。

DBM-COT ITEM #219

Recipe → Plan → Run → Artifacts

Time-Series IR · Task–Action Dual ACLM Architecture

```
flowchart LR
    %% =====
    %% Task Layer (Intent / What)
    %% =====
    subgraph TASK ["Task Layer ACLM (Intent / Configuration)"]
        R1["Recipe (JSON)"]
        - sourceCurves
        - fixtures
        - patternExtractors
        - callingGraph / sequence
        - policies
        - governance"
    end

    %% =====
    %% Compile / Planning Stage
    %% =====
    subgraph PLAN ["Plan Stage (Compilation / Validation)"]
        P1["Pipeline Plan"]
        - Resolved Extractors
        - Dependency Graph (DAG / Sequence)
    end
```

```

    - Parameter Binding
    - Capability Check"]
end

%% =====
%% Action Layer (Runtime / How)
%% =====
subgraph ACTION["Action Layer ACLM (Runtime Execution)"]
    subgraph BUS["Runtime Buses"]
        DB["DataBus
            (Curves, Windows, Features, Observations)"]
        EB["EventBus
            (Events, Revisions, Dependencies)"]
    end

    subgraph MACHINES["Extractor Machines"]
        E1["Extractor A"]
        E2["Extractor B"]
        E3["Extractor C"]
    end

    GUARD["Runtime Guards
        - cycle guard
        - max events
        - resource limits"]

    CACHE["Caches
        - feature cache
        - observation cache
        - event cache"]
end

%% =====
%% Output Artifacts
%% =====
subgraph OUT["Artifacts (Analytical Outputs)"]
    O1["Observation Layer
        (windows, stats, features)"]
    O2["Event Layer
        (typed events + evidence links)"]
    O3["Summary / Wrapper Layer
        (IR summaries, starmap-ready)"]
    O4["Run Manifest
        (recipe hash, versions, params)"]
end

%% =====
%% Flow Connections
%% =====
R1 -->|"compile"| P1
P1 -->|"instantiate"| ACTION

DB --> E1
DB --> E2
DB --> E3

E1 --> EB

```

```
E2 --> EB  
E3 --> EB  
  
EB --> E2  
EB --> E3  
  
ACTION --> GUARD  
ACTION --> CACHE  
  
EB --> O2  
DB --> O1  
O2 --> O3  
P1 --> O4
```

教学说明 (建议写入 ITEM #219 的 Figure Caption)

Figure — Recipe → Plan → Run → Artifacts

- **Recipe (Task Layer)**

以 JSON 形式声明研究意图：用什么数据、跑哪些 extractor、如何编排、受哪些策略与治理约束。

- **Plan (编译期)**

将 Recipe 编译为可执行的 Pipeline Plan，完成依赖检查、能力匹配、参数绑定，是“结构正确性”的第一道防线。

- **Run (Action Layer Runtime)**

在运行时自动组装流水线：

- DataBus 负责数据与观测流；
- EventBus 负责事件流与事件产事件；
- Extractor Machines 像车间机器一样可插拔；
- Guards 与 Caches 提供安全与性能护栏。

- **Artifacts (真正产出)**

Time-Series IR 的价值不止是事件表，而是：

- 多层 IR，
- 解释链，
- 可比对摘要，
- 以及可复现的 Run Manifest。

为什么这张图对 #219 很“封顶”

- 它明确区分 Task 与 Action，为 APTGOE 演化留足空间；
- 它把 事件产事件 正式纳入主流程；
- 它把 Artifacts 提升为一等公民，而不是“顺便输出”；
- 它天然对应你后续要做的：
 - JSON schema 固化
 - Plan compiler
 - Runtime governance
 - Evidence / distance / starmap 对接

如果你愿意，下一步我可以直接接着这张图做三种强化之一（都很自然）：

1. 在不增加认知负担的前提下，做一张 v2：加入 Policy / Governance 轴
2. 基于此图，反推一个“最小必填版 Recipe JSON Schema”
3. 把这张图与 ACLM / APTGOE 总架构图对齐，形成 DBM-COT 的跨 ITEM 主干图

你选一个，我们继续“上半身”的收官与落地。

请抽象 Time-Series IR Recipe JSON 的“宪法级 schema（最小必填版）”。老弟，你可以尽情地放飞这片处女地你得意的设计。

下面给出一份我建议的 Time-Series IR Recipe JSON “宪法级 Schema（最小必填版）”。设计目标是：字段极少、语义封顶、可演化、可编译成 Plan、能治理、能复现、能支持 Event→Event。

我会先给出 (A) 最小必填 Schema（规范定义），再给出 (B) 最小可运行示例（MVP），以及 (C) 编译期/运行期硬约束（必须实现的校验语义）。你可以直接把它固化为 #219 的附录，或作为后续 Java RecipeParser/PlanCompiler 的契约依据。

A) 宪法级 Recipe JSON Schema (最小必填版)

约定：此处是“规范”，不是 JSON-Schema 草案语法。字段名可在你 repo 内微调，但语义建议保持。

1) 顶层结构

```
{  
  "recipeVersion": "1.0",  
  "id": "string",  
  "sources": [ ... ],  
  "operators": { ... },  
  "graph": { ... },  
  "artifacts": { ... }  
}
```

字段说明 (必填)

- **recipeVersion**：配方契约版本（用于未来升级兼容）
- **id**：配方 ID（用于审计、缓存 key、复现）
- **sources**：输入曲线定义（至少 1 个）
- **operators**：本次运行可用的机器清单（extractors / builders / gates 等）
- **graph**：执行图（sequence/DAG/iter）
- **artifacts**：输出声明（至少事件层 + manifest）

2) sources []：输入定义 (必填)

```
{  
  "name": "string",  
  "type": "curve",  
  "ref": "string",  
  "slice": { "from": "string", "to": "string" }  
}
```

- **name**：在配方内引用该 source 的符号名（例如 "SPY_1D"）
- **type**：固定 "curve"（宪法级先不扩展）
- **ref**：数据引用字符串（可以是符号、文件、URL、DB key；由你的 data adapter 解释）
- **slice**：时间范围（最小必填：from/to；格式由你统一，如 ISO-8601 或业务时间）

最小必填不放入 resample/normalize 等“工装夹具”，因为这些可以先作为 operator 或默认 fixture；等 v1.1 再扩展。

3) operators: 机器注册表（必填）

```
{  
  "operators": {  
    "<opId>": {  
      "kind": "extractor",  
      "impl": "string",  
      "params": {}  
    }  
  }  
}
```

- **opId**：配方内机器 ID（例如 "ex_updown_mvp"）
- **kind**：先最小只要求 "extractor"（以后可扩展 "feature" | "gate" | "aggregate"）
- **impl**：实现绑定（例如 Java 类名 / registry key / plugin id）
- **params**：实现参数（可以为空对象，但字段必须存在以便一致性与 hash）

关键点：Recipe 里不写“逻辑代码”，只写“选择哪台机器 + 给参数”。

4) graph: 执行图（必填， 宪法级核心）

最小但封顶的图结构如下：

```
{  
  "mode": "sequence",  
  "nodes": [  
    {  
      "id": "n1",  
      "op": "<opId>",  
      "in": {  
        "curves": ["<sourceName>"],  
        "events": ["<eventStreamName>"]  
      },  
      "out": {  
        "events": "<eventStreamName>"  
      }  
    }  
  ]  
}
```

`graph.mode` (必填)

- "sequence" : 线性顺序 (最简单，先跑通)
- 未来可扩展 "dag"、"iter"，但不影响最小版 schema

`nodes[]` (必填, 至少1个)

每个 node 表示一次机器执行：

- **id** : 节点 ID (用于依赖与审计)
- **op** : 引用 operators 中的某台机器
- **in.curves** : 输入曲线 (至少 1 个 sourceName)
- **in.events** : 可选，输入事件流 (支持 Event→Event)
- **out.events** : 输出事件流名称 (一个 string)

Event→Event 的关键：`in.events` 与 `out.events` 的命名允许你构建“事件产事件”的序列 (或未来 DAG)。

5) `artifacts`: 输出声明 (必填)

```
{  
  "events": [ "<eventStreamName>" ],  
  "manifest": true  
}
```

- **events** : 要输出哪些事件流 (至少 1 个)
- **manifest** : 必须为 true (复现与审计的宪法级硬要求)

最小必填不强制 `observation/summary`，但建议你在 runtime 默认也产出 (即使 `artifacts` 未声明)，或者下一版再显式化。

B) 最小可运行 Recipe 示例 (MVP)

这个示例覆盖：

- 1 个 source curve
- 2 台 extractor
- extractor B 依赖 extractor A 的事件流 (Event→Event)
- 输出事件流 + manifest

```
{
  "recipeVersion": "1.0",
  "id": "tsir_demo_updown_then_refine_v1",
  "sources": [
    {
      "name": "CURVE_A",
      "type": "curve",
      "ref": "SPY:1D",
      "slice": { "from": "2020-01-01", "to": "2024-12-31" }
    }
  ],
  "operators": {
    "ex_updown_mvp": {
      "kind": "extractor",
      "impl": "com.dbm.ir.runtime.MvpUpDownExtractor",
      "params": { "window": 5, "threshold": 0.0 }
    },
    "ex_refine_swings": {
      "kind": "extractor",
      "impl": "com.dbm.ir.runtime.RefineSwingExtractor",
      "params": { "minBars": 3 }
    }
  },
  "graph": {
    "mode": "sequence",
    "nodes": [
      {
        "id": "n1",
        "op": "ex_updown_mvp",
        "in": { "curves": ["CURVE_A"], "events": [] },
        "out": { "events": "E_BASE" }
      },
      {
        "id": "n2",
        "op": "ex_refine_swings",
        "in": { "curves": ["CURVE_A"], "events": ["E_BASE"] },
        "out": { "events": "E_REFINED" }
      }
    ],
    "artifacts": {
      "events": ["E_REFINED"],
      "manifest": true
    }
  }
}
```

C) 宪法级硬语义：编译期/运行期必须实现的校验规则

这部分是“宪法”的灵魂：没有这些，JSON 只是配置文件；有这些，才是 Task Layer ACLM。

C1. 编译期 (Plan Compiler) 必须校验

1. **Operator existence**

- 每个 node.op 必须存在于 operators

2. **Source existence**

- in.curves[] 引用必须存在于 sources.name

3. **Event stream existence (前向引用限制)**

- 对于 sequence 模式：in.events[] 必须来自前面节点的 out.events (或是空)
- (未来 dag/iter 再扩展更复杂的引用)

4. **Single-writer rule (最小版建议强制)**

- 同名 out.events 在同一 recipe 中只能由一个 node 产生
(避免事件流覆盖导致解释链断裂)

5. **Determinism contract**

- recipe hash = stable canonical JSON (排序、去空白)
- 运行 manifest 必须记录 recipe hash、operators impl、params

C2. 运行期 (Runtime) 必须保证

6. **Manifest always produced**

- 即使中途失败，也要输出 partial manifest (失败原因、已完成节点、资源用量)

7. Event revision semantics (最小版定义)

- 若某 extractor 修改/过滤输入事件流，不得“原地修改”；必须输出新 stream
($E_{BASE} \rightarrow E_{REFINED}$)
(这是解释链与复现的根)

8. Resource safety rails (最小版内建默认)

- 最大事件数 / 最大节点执行时间 / 最大回写深度
(即使 recipe 未声明，也要有默认值)

你会很喜欢单引号的一个“放飞但不加字段”的关键点

注意：我在最小 schema 里刻意不加入 policies/governance/fixtures 的显式字段，原因是：

- 宪法级最小版应当只定义“能编排与能复现”的必要字段；
- policies/governance 可以先通过 `operators[opId].params` 承载（不破坏宪法结构）；
- 等你们跑通几十个 patterns 的 JSON 编排后，再把“普遍存在的 params”上提为顶层 (v1.1 的自然演化)。

这会让你工程推进极快：

先统一骨架 → 再逐步提炼公共层，完全符合你强调的“善变、快迭代”。

下一步我建议你立刻落地的两件小而关键的工程产物

1. Canonical JSON 哈希器 (用于 manifest / cache key)
 - 解决“同配方不同格式”导致复现与缓存失效的问题。
2. PlanCompiler 的三类错误码

- RECIPE_INVALID (结构缺失)
 - RESOLUTION_FAILED (op/source/event 引用解析失败)
 - CAPABILITY_MISMATCH (op 不支持某输入类型；先 stub 也行)
-

如果你同意，我下一条就可以直接给出：

- 一个对应此 schema 的 **Java 8 POJO + Parser/Validator + PlanCompiler skeleton** (贴合你 com.dbm.ir.runtime 风格) ，并附 3–5 个 JUnit4 玩具测试 (含 Event→Event 引用、single-writer 违规、前向引用错误) 。

请生成一个对应此 schema 的 Java 8 POJO + Parser/Validator + PlanCompiler skeleton (贴合你 com.dbm.ir.runtime 风格) ，并附 3–5 个 JUnit4 玩具测试 (含 Event→Event 引用、single-writer 违规、前向引用错误) 。我特喜欢这种方式，能实物型地感受别人的灵感与创新，学习进步的特快。

```
// File: src/main/java/com/dbm/ir/runtime/TsirRecipe.java
package com.dbm.ir.runtime;

import java.util.*;

/**
 * Time-Series IR Recipe (Constitution-level minimal schema).
 *
 * This POJO is intentionally small and stable:
 * recipeVersion, id, sources, operators, graph, artifacts
 */
public final class TsirRecipe {
    private String recipeVersion; // required
    private String id;           // required

    private List<SourceSpec> sources = new ArrayList<>();                   //
    required (>=1)
    private Map<String, OperatorSpec> operators = new LinkedHashMap<>(); // required (>=1)
    private GraphSpec graph;                                              //
    required
    private ArtifactsSpec artifacts;                                         //
    required

    public TsirRecipe() {}

    public String getRecipeVersion() { return recipeVersion; }
    public void setRecipeVersion(String recipeVersion) { this.recipeVersion =
recipeVersion; }
```

```

public String getId() { return id; }
public void setId(String id) { this.id = id; }

public List<SourceSpec> getSources() { return sources; }
public void setSources(List<SourceSpec> sources) {
    this.sources = (sources == null) ? new ArrayList<SourceSpec>() :
sources;
}

public Map<String, OperatorSpec> getOperators() { return operators; }
public void setOperators(Map<String, OperatorSpec> operators) {
    this.operators = (operators == null) ? new LinkedHashMap<String,
OperatorSpec>() : operators;
}

public GraphSpec getGraph() { return graph; }
public void setGraph(GraphSpec graph) { this.graph = graph; }

public ArtifactsSpec getArtifacts() { return artifacts; }
public void setArtifacts(ArtifactsSpec artifacts) { this.artifacts =
artifacts; }

// ----- Nested POJOs -----

public static final class SourceSpec {
    private String name; // required
    private String type; // required: "curve"
    private String ref; // required
    private SliceSpec slice; // required

    public SourceSpec() {}

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getType() { return type; }
    public void setType(String type) { this.type = type; }

    public String getRef() { return ref; }
    public void setRef(String ref) { this.ref = ref; }

    public SliceSpec getSlice() { return slice; }
    public void setSlice(SliceSpec slice) { this.slice = slice; }

    public static final class SliceSpec {
        private String from; // required
        private String to; // required

        public SliceSpec() {}

        public String getFrom() { return from; }
        public void setFrom(String from) { this.from = from; }

        public String getTo() { return to; }
        public void setTo(String to) { this.to = to; }
    }
}

```

```

}

public static final class OperatorSpec {
    private String kind; // required: "extractor" (minimal)
    private String impl; // required: class name or registry key
    private Map<String, Object> params = new LinkedHashMap<>(); // required field (can be empty)

    public OperatorSpec() {}

    public String getKind() { return kind; }
    public void setKind(String kind) { this.kind = kind; }

    public String getImpl() { return impl; }
    public void setImpl(String impl) { this.impl = impl; }

    public Map<String, Object> getParams() { return params; }
    public void setParams(Map<String, Object> params) {
        this.params = (params == null) ? new LinkedHashMap<String, Object>() : params;
    }
}

public static final class GraphSpec {
    private String mode; // required: "sequence" (minimal)
    private List<NodeSpec> nodes = new ArrayList<>(); // required (>=1)

    public GraphSpec() {}

    public String getMode() { return mode; }
    public void setMode(String mode) { this.mode = mode; }

    public List<NodeSpec> getNodes() { return nodes; }
    public void setNodes(List<NodeSpec> nodes) {
        this.nodes = (nodes == null) ? new ArrayList<NodeSpec>() : nodes;
    }
}

public static final class NodeSpec {
    private String id; // required
    private String op; // required: operator id
    private InSpec in; // required
    private OutSpec out; // required

    public NodeSpec() {}

    public String getId() { return id; }
    public void setId(String id) { this.id = id; }

    public String getOp() { return op; }
    public void setOp(String op) { this.op = op; }

    public InSpec getIn() { return in; }
    public void setIn(InSpec in) { this.in = in; }

    public OutSpec getOut() { return out; }
    public void setOut(OutSpec out) { this.out = out; }
}

```

```

        public static final class InSpec {
            private List<String> curves = new ArrayList<>(); // required
(=>1)
            private List<String> events = new ArrayList<>(); // required
field (can be empty)

            public InSpec() {}

            public List<String> getCurves() { return curves; }
            public void setCurves(List<String> curves) {
                this.curves = (curves == null) ? new ArrayList<String>() :
curves;
            }

            public List<String> getEvents() { return events; }
            public void setEvents(List<String> events) {
                this.events = (events == null) ? new ArrayList<String>() :
events;
            }
        }

        public static final class OutSpec {
            private String events; // required (event stream name)

            public OutSpec() {}

            public String getEvents() { return events; }
            public void setEvents(String events) { this.events = events; }
        }
    }

    public static final class ArtifactsSpec {
        private List<String> events = new ArrayList<>(); // required (>=1)
        private Boolean manifest; // required: true

        public ArtifactsSpec() {}

        public List<String> getEvents() { return events; }
        public void setEvents(List<String> events) {
            this.events = (events == null) ? new ArrayList<String>() :
events;
        }

        public Boolean getManifest() { return manifest; }
        public void setManifest(Boolean manifest) { this.manifest =
manifest; }
    }
}

// File: src/main/java/com/dbm/ir/runtime/TsirRecipeParseException.java
package com.dbm.ir.runtime;

public final class TsirRecipeParseException extends RuntimeException {
    public TsirRecipeParseException(String message, Throwable cause) {
        super(message, cause);
    }
    public TsirRecipeParseException(String message) {

```

```

        super(message);
    }
}
// File: src/main/java/com/dbm/ir/runtime/TsirRecipeValidationException.java
package com.dbm.ir.runtime;

import java.util.*;

public final class TsirRecipeValidationException extends RuntimeException {
    private final List<TsirRecipeValidationIssue> issues;

    public TsirRecipeValidationException(List<TsirRecipeValidationIssue>
issues) {
        super(buildMessage(issues));
        this.issues = (issues == null) ?
Collections.<TsirRecipeValidationIssue>emptyList() : issues;
    }

    public List<TsirRecipeValidationIssue> getIssues() {
        return issues;
    }

    private static String buildMessage(List<TsirRecipeValidationIssue>
issues) {
        if (issues == null || issues.isEmpty()) return "Recipe validation
failed (no details)";
        StringBuilder sb = new StringBuilder("Recipe validation failed:\n");
        for (TsirRecipeValidationIssue i : issues) {
            sb.append("- ").append(i.toString()).append("\n");
        }
        return sb.toString();
    }
}
// File: src/main/java/com/dbm/ir/runtime/TsirRecipeValidationIssue.java
package com.dbm.ir.runtime;

public final class TsirRecipeValidationIssue {
    public enum Code {
        MISSING_FIELD,
        INVALID_VALUE,
        UNKNOWN_SOURCE,
        UNKNOWN_OPERATOR,
        FORWARD_EVENT_REFERENCE,
        UNKNOWN_EVENT_STREAM,
        SINGLE_WRITER_VIOLATION,
        EMPTY_LIST
    }

    private final Code code;
    private final String path;
    private final String message;

    public TsirRecipeValidationIssue(Code code, String path, String message)
{
    this.code = code;
    this.path = path;
    this.message = message;
}

```

```

    }

    public Code getCode() { return code; }
    public String getPath() { return path; }
    public String getMessage() { return message; }

    @Override
    public String toString() {
        return code + " at " + path + ":" + message;
    }
}

// File: src/main/java/com/dbm/ir/runtime/TsirRecipeParser.java
package com.dbm.ir.runtime;

import java.lang.reflect.*;

/**
 * Parser uses Jackson if present on classpath (common in many Java
 * projects).
 * If Jackson is not available, it throws with a clear message.
 *
 * You can swap this with your existing JSON loader/adaptor later without
 * touching validator/compiler.
 */
public final class TsirRecipeParser {

    public TsirRecipe parse(String json) {
        if (json == null || json.trim().isEmpty()) {
            throw new TsirRecipeParseException("JSON is empty");
        }
        try {
            // Reflective Jackson usage to avoid hard dependency in this
            skeleton.
            // Equivalent to: new ObjectMapper().readValue(json,
            TsirRecipe.class)
            Class<?> omCls =
Class.forName("com.fasterxml.jackson.databind.ObjectMapper");
            Object om = omCls.getDeclaredConstructor().newInstance();
            Method readValue = omCls.getMethod("readValue", String.class,
Class.class);
            Object obj = readValue.invoke(om, json, TsirRecipe.class);
            return (TsirRecipe) obj;
        } catch (ClassNotFoundException e) {
            throw new TsirRecipeParseException(
                "Jackson not found on classpath. Add dependency
com.fasterxml.jackson.core:jackson-databind " +
                "or replace TsirRecipeParser with your project's JSON
loader.", e);
        } catch (InvocationTargetException ite) {
            Throwable cause = ite.getTargetException();
            throw new TsirRecipeParseException("Failed to parse recipe JSON:
" + cause.getMessage(), cause);
        } catch (Exception e) {
            throw new TsirRecipeParseException("Failed to parse recipe JSON:
" + e.getMessage(), e);
        }
    }
}

```

```

}

// File: src/main/java/com/dbm/ir/runtime/TsirRecipeValidator.java
package com.dbm.ir.runtime;

import java.util.*;

import static com.dbm.ir.runtime.TsirRecipeValidationIssue.Code;

/**
 * Constitution-level validator for minimal schema:
 * - required fields exist
 * - operators and sources resolved
 * - sequence mode: event streams cannot be forward-referenced
 * - single-writer rule for out.events stream names
 * - artifacts.manifest must be true
 */
public final class TsirRecipeValidator {

    public List<TsirRecipeValidationIssue> validate(TsirRecipe r) {
        List<TsirRecipeValidationIssue> issues = new ArrayList<>();

        if (r == null) {
            issues.add(new TsirRecipeValidationIssue(Code.MISSING_FIELD, "$",
"recipe is null"));
            return issues;
        }

        // Top-level required fields
        reqNonBlank(issues, ".$.recipeVersion", r.getRecipeVersion());
        reqNonBlank(issues, ".$.id", r.getId());

        if (r.getSources() == null || r.getSources().isEmpty()) {
            issues.add(new TsirRecipeValidationIssue(Code.EMPTY_LIST,
"$.sources", "must have at least 1 source"));
        }
        if (r.getOperators() == null || r.getOperators().isEmpty()) {
            issues.add(new TsirRecipeValidationIssue(Code.EMPTY_LIST,
"$.operators", "must have at least 1 operator"));
        }

        if (r.getGraph() == null) {
            issues.add(new TsirRecipeValidationIssue(Code.MISSING_FIELD,
"$.graph", "graph is required"));
        }
        if (r.getArtifacts() == null) {
            issues.add(new TsirRecipeValidationIssue(Code.MISSING_FIELD,
"$.artifacts", "artifacts is required"));
        }

        // If missing core blocks, stop deeper checks to avoid NPE noise.
        if (!issues.isEmpty() && (r.getGraph() == null || r.getArtifacts() ==
null)) return issues;

        // Sources validation + index
        Set<String> sourceNames = new HashSet<>();
        if (r.getSources() != null) {
            for (int i = 0; i < r.getSources().size(); i++) {

```

```

        TsirRecipe.SourceSpec s = r.getSources().get(i);
        String base = "$.sources[" + i + "]";
        if (s == null) {
            issues.add(new
TsirRecipeValidationIssue(Code.MISSING_FIELD, base, "source is null"));
                continue;
        }
        reqNonBlank(issues, base + ".name", s.getName());
        reqNonBlank(issues, base + ".type", s.getType());
        reqNonBlank(issues, base + ".ref", s.getRef());
        if (s.getSlice() == null) {
            issues.add(new
TsirRecipeValidationIssue(Code.MISSING_FIELD, base + ".slice", "slice is
required"));
        } else {
            reqNonBlank(issues, base + ".slice.from",
s.getSlice().getFrom());
            reqNonBlank(issues, base + ".slice.to",
s.getSlice().getTo());
        }
        if (s.getName() != null) sourceNames.add(s.getName());
        if (s.getType() != null && !"curve".equals(s.getType())) {
            issues.add(new
TsirRecipeValidationIssue(Code.INVALID_VALUE, base + ".type", "minimal schema
expects type='curve'"));
        }
    }

    // Operators validation
    Map<String, TsirRecipe.OperatorSpec> ops = r.getOperators();
    if (ops != null) {
        for (Map.Entry<String, TsirRecipe.OperatorSpec> e :
ops.entrySet()) {
            String opId = e.getKey();
            TsirRecipe.OperatorSpec op = e.getValue();
            String base = "$.operators['" + opId + "']";
            if (op == null) {
                issues.add(new
TsirRecipeValidationIssue(Code.MISSING_FIELD, base, "operator is null"));
                    continue;
            }
            reqNonBlank(issues, base + ".kind", op.getKind());
            reqNonBlank(issues, base + ".impl", op.getImpl());
            if (op.getParams() == null) {
                issues.add(new
TsirRecipeValidationIssue(Code.MISSING_FIELD, base + ".params", "params field
must exist (can be empty object)"));
            }
            if (op.getKind() != null
&& !"extractor".equals(op.getKind())) {
                issues.add(new
TsirRecipeValidationIssue(Code.INVALID_VALUE, base + ".kind", "minimal schema
expects kind='extractor'"));
            }
        }
    }
}

```

```

// Graph validation
TsirRecipe.GraphSpec g = r.getGraph();
reqNonBlank(issues, ".$.graph.mode", g.getMode());
if (g.getMode() != null && !"sequence".equals(g.getMode())) {
    issues.add(new TsirRecipeValidationIssue(Code.INVALID_VALUE,
"$.graph.mode", "minimal schema expects mode='sequence'"));
}
if (g.getNodes() == null || g.getNodes().isEmpty()) {
    issues.add(new TsirRecipeValidationIssue(Code.EMPTY_LIST,
".$.graph.nodes", "must have at least 1 node"));
    return issues;
}

// Artifacts validation
TsirRecipe.ArtifactsSpec a = r.getArtifacts();
if (a.getEvents() == null || a.getEvents().isEmpty()) {
    issues.add(new TsirRecipeValidationIssue(Code.EMPTY_LIST,
".$.artifacts.events", "must request at least 1 event stream output"));
}
if (a.getManifest() == null) {
    issues.add(new TsirRecipeValidationIssue(Code.MISSING_FIELD,
".$.artifacts.manifest", "manifest is required and must be true"));
} else if (!Boolean.TRUE.equals(a.getManifest())) {
    issues.add(new TsirRecipeValidationIssue(Code.INVALID_VALUE,
".$.artifacts.manifest", "manifest must be true (constitution requirement)"));
}

// Sequence semantics: resolve nodes, enforce event references,
single-writer
Set<String> availableStreams = new HashSet<>();
Map<String, String> streamWriter = new HashMap<>(); // stream ->
nodeId

for (int i = 0; i < g.getNodes().size(); i++) {
    TsirRecipe.NodeSpec n = g.getNodes().get(i);
    String base = ".$.graph.nodes[" + i + "]";

    if (n == null) {
        issues.add(new TsirRecipeValidationIssue(Code.MISSING_FIELD,
base, "node is null"));
        continue;
    }
    reqNonBlank(issues, base + ".id", n.getId());
    reqNonBlank(issues, base + ".op", n.getOp());
    if (n.getIn() == null) {
        issues.add(new TsirRecipeValidationIssue(Code.MISSING_FIELD,
base + ".in", "in is required"));
    }
    if (n.getOut() == null) {
        issues.add(new TsirRecipeValidationIssue(Code.MISSING_FIELD,
base + ".out", "out is required"));
    }
    if (n.getIn() == null || n.getOut() == null) continue;

    // op exists?

```

```

        if (n.getOp() != null && (ops == null
        || !ops.containsKey(n.getOp()))) {
            issues.add(new
TsirRecipeValidationIssue(Code.UNKNOWN_OPERATOR, base + ".op", "unknown
operator: " + n.getOp()));
        }

        // curves exist?
List<String> curves = n.getIn().getCurves();
if (curves == null || curves.isEmpty()) {
    issues.add(new TsirRecipeValidationIssue(Code.EMPTY_LIST,
base + ".in.curves", "must reference at least 1 curve source"));
} else {
    for (int j = 0; j < curves.size(); j++) {
        String c = curves.get(j);
        if (!sourceNames.contains(c)) {
            issues.add(new
TsirRecipeValidationIssue(Code.UNKNOWN_SOURCE, base + ".in.curves[" + j +
"]", "unknown source: " + c));
        }
    }
}

// event streams must exist already (no forward refs in sequence)
List<String> ins = n.getIn().getEvents();
if (ins != null) {
    for (int j = 0; j < ins.size(); j++) {
        String s = ins.get(j);
        if (s == null || s.trim().isEmpty()) {
            issues.add(new
TsirRecipeValidationIssue(Code.MISSING_FIELD, base + ".in.events[" + j + "]",
"event stream name is blank"));
            continue;
        }
        if (!availableStreams.contains(s)) {
            // Could be forward reference or completely unknown;
in sequence they are identical symptom.
            issues.add(new
TsirRecipeValidationIssue(Code.FORWARD_EVENT_REFERENCE, base + ".in.events[" +
j + "],
                    "event stream not available yet
(forward/unknown): " + s));
        }
    }
}

// out.events required and single-writer
String outStream = n.getOut().getEvents();
if (outStream == null || outStream.trim().isEmpty()) {
    issues.add(new TsirRecipeValidationIssue(Code.MISSING_FIELD,
base + ".out.events", "out.events is required"));
} else {
    String existingWriter = streamWriter.get(outStream);
    String nodeId = n.getId() == null ? ("@index" + i) :
n.getId();
    if (existingWriter != null && !existingWriter.equals(nodeId))
{

```

```

                issues.add(new
TsirRecipeValidationIssue(Code.SINGLE_WRITER_VIOLATION, base + ".out.events",
                           "event stream '" + outStream + "' already written by
node '" + existingWriter + "'"));
            } else {
                streamWriter.put(outStream, nodeId);
            }
            availableStreams.add(outStream);
        }
    }

    // artifact event streams must exist (produced by some node)
    if (a.getEvents() != null) {
        for (int i = 0; i < a.getEvents().size(); i++) {
            String s = a.getEvents().get(i);
            if (s == null || s.trim().isEmpty()) {
                issues.add(new
TsirRecipeValidationIssue(Code.MISSING_FIELD, "$.artifacts.events[" + i +
"]", "blank event stream name"));
                continue;
            }
            if (!streamWriter.containsKey(s)) {
                issues.add(new
TsirRecipeValidationIssue(Code.UNKNOWN_EVENT_STREAM, "$.artifacts.events[" + i +
"]",
                           "requested output stream not produced: " + s));
            }
        }
    }

    return issues;
}

public void validateOrThrow(TsirRecipe r) {
    List<TsirRecipeValidationIssue> issues = validate(r);
    if (!issues.isEmpty()) {
        throw new TsirRecipeValidationException(issues);
    }
}

private static void reqNonBlank(List<TsirRecipeValidationIssue> issues,
String path, String val) {
    if (val == null || val.trim().isEmpty()) {
        issues.add(new TsirRecipeValidationIssue(Code.MISSING_FIELD,
path, "required non-blank string"));
    }
}
// File: src/main/java/com/dbm/ir/runtime/TsirPlan.java
package com.dbm.ir.runtime;

import java.util.*;

/**
 * Compiled plan = executable structure resolved from recipe.
 * In MVP: sequence of PlanNodes, plus resolved stream writers.
 */

```

```

public final class TsirPlan {
    private final String recipeId;
    private final String recipeVersion;
    private final List<PlanNode> nodes;
    private final Set<String> outputEventStreams;
    private final boolean manifestRequired;

    public TsirPlan(String recipeId,
                    String recipeVersion,
                    List<PlanNode> nodes,
                    Set<String> outputEventStreams,
                    boolean manifestRequired) {
        this.recipeId = recipeId;
        this.recipeVersion = recipeVersion;
        this.nodes = Collections.unmodifiableList(new ArrayList<>(nodes));
        this.outputEventStreams = Collections.unmodifiableSet(new
LinkedHashSet<>(outputEventStreams));
        this.manifestRequired = manifestRequired;
    }

    public String getRecipeId() { return recipeId; }
    public String getRecipeVersion() { return recipeVersion; }
    public List<PlanNode> getNodes() { return nodes; }
    public Set<String> getOutputEventStreams() { return outputEventStreams; }
    public boolean isManifestRequired() { return manifestRequired; }

    public static final class PlanNode {
        private final String nodeId;
        private final String operatorId;
        private final String operatorImpl;
        private final Map<String, Object> operatorParams;

        private final List<String> curveInputs;
        private final List<String> eventInputs;
        private final String eventOutput;

        public PlanNode(String nodeId,
                        String operatorId,
                        String operatorImpl,
                        Map<String, Object> operatorParams,
                        List<String> curveInputs,
                        List<String> eventInputs,
                        String eventOutput) {
            this.nodeId = nodeId;
            this.operatorId = operatorId;
            this.operatorImpl = operatorImpl;
            this.operatorParams = operatorParams == null ?
Collections.<String, Object>emptyMap()
                : Collections.unmodifiableMap(new
LinkedHashMap<>(operatorParams));
            this.curveInputs = Collections.unmodifiableList(new
ArrayList<>(curveInputs));
            this.eventInputs = Collections.unmodifiableList(new
ArrayList<>(eventInputs));
            this.eventOutput = eventOutput;
        }
    }
}

```

```

        public String getNodeID() { return nodeId; }
        public String getOperatorID() { return operatorId; }
        public String getOperatorImpl() { return operatorImpl; }
        public Map<String, Object> getOperatorParams() { return
operatorParams; }
        public List<String> getCurveInputs() { return curveInputs; }
        public List<String> getEventInputs() { return eventInputs; }
        public String getEventOutput() { return eventOutput; }
    }
}
// File: src/main/java/com/dbm/ir/runtime/TsirPlanCompiler.java
package com.dbm.ir.runtime;

import java.util.*;

/**
 * PlanCompiler:
 * - validate recipe (constitution rules)
 * - resolve operator impl/params into plan nodes
 * - output plan is ready for runtime instantiation
 */
public final class TsirPlanCompiler {

    private final TsirRecipeValidator validator;

    public TsirPlanCompiler() {
        this(new TsirRecipeValidator());
    }

    public TsirPlanCompiler(TsirRecipeValidator validator) {
        this.validator = (validator == null) ? new TsirRecipeValidator() :
validator;
    }

    public TsirPlan compileOrThrow(TsirRecipe recipe) {
        validator.validateOrThrow(recipe);

        List<TsirPlan.PlanNode> planNodes = new ArrayList<>();
        Map<String, TsirRecipe.OperatorSpec> ops = recipe.getOperators();

        for (TsirRecipe.NodeSpec n : recipe.getGraph().getNodes()) {
            TsirRecipe.OperatorSpec op = ops.get(n.getOp());

            TsirPlan.PlanNode pn = new TsirPlan.PlanNode(
                n.getId(),
                n.getOp(),
                op.getImpl(),
                op.getParams(),
                safeList(n.getIn().getCurves()),
                safeList(n.getIn().getEvents()),
                n.getOut().getEvents()
            );
            planNodes.add(pn);
        }

        Set<String> outs = new
LinkedHashSet<>(recipe.getArtifacts().getEvents());
    }
}

```

```

        boolean manifest =
Boolean.TRUE.equals(recipe.getArtifacts().getManifest()));

        return new TsirPlan(recipe.getId(), recipe.getRecipeVersion(),
planNodes, outs, manifest);
    }

    private static List<String> safeList(List<String> xs) {
        return xs == null ? Collections.<String>emptyList() : xs;
    }
}

// File: src/test/java/com/dbm/ir/runtime/TsirRecipeCompilerTest.java
package com.dbm.ir.runtime;

import org.junit.Test;

import java.util.List;

import static org.junit.Assert.*;

public class TsirRecipeCompilerTest {

    private final TsirRecipeParser parser = new TsirRecipeParser();
    private final TsirRecipeValidator validator = new TsirRecipeValidator();
    private final TsirPlanCompiler compiler = new
TsirPlanCompiler(validator);

    @Test
    public void test_valid_eventToEvent_sequence_compiles() {
        String json =
            "{\n" +
            "  \"recipeVersion\": \"1.0\", \n" +
            "  \"id\": \"tsir_demo_ok\", \n" +
            "  \"sources\": [\n" +
            "    {\n" +
            "      \"name\": \"CURVE_A\", \"type\": \"curve\", \"ref\": \"SPY:1D\", \"slice\": {\"from\":
\"2020-01-01\", \"to\": \"2024-12-31\"}}\n" +
            "    ], \n" +
            "    \"operators\": {\n" +
            "      \"ex_updown_mvp\": {\n" +
            "        \"kind\": \"extractor\", \"impl\": \"com.dbm.ir.runtime.MvpUpDownExtractor\",
\"params\": {\"window\": 5, \"threshold\": 0.0}}, \n" +
            "        \"ex_refine\": {\n" +
            "          \"kind\": \"extractor\", \"impl\": \"com.dbm.ir.runtime.RefineSwingExtractor\",
\"params\": {\"minBars\": 3}}\n" +
            "        }, \n" +
            "        \"graph\": {\n" +
            "          \"mode\": \"sequence\", \n" +
            "          \"nodes\": [\n" +
            "            {\n" +
            "              \"id\": \"n1\", \"op\": \"ex_updown_mvp\", \"in\": {\"curves\": [\"CURVE_A\"]},
\"events\": []}, \n" +
            "              \"out\": {\"events\": \"E_BASE\"}}, \n" +
            "            {\n" +
            "              \"id\": \"n2\", \"op\": \"ex_refine\", \"in\": {\"curves\": [\"CURVE_A\"]},
\"events\": [\"E_BASE\"]}, \n" +
            "              \"out\": {\"events\": \"E_REFINED\"}}\n" +
            "            ]\n" +
            "          }, \n" +
            "        }\n" +
            "      }\n" +
            "    }\n" +
            "  }\n" +
            "}\n"

```

```

    "  \"artifacts\":
{\\"events\": [\"E_REFINED\"], \"manifest\":true}\n" +
"}\n";

    TsirRecipe r = parser.parse(json);
    TsirPlan p = compiler.compileOrThrow(r);

    assertEquals("tsir_demo_ok", p.getRecipeId());
    assertEquals(2, p.getNodes().size());
    assertTrue(p.getOutputEventStreams().contains("E_REFINED"));
    assertTrue(p.isManifestRequired());
    assertEquals("E_BASE", p.getNodes().get(0).getEventOutput());
    assertEquals("E_REFINED", p.getNodes().get(1).getEventOutput());
    assertEquals(1, p.getNodes().get(1).getEventInputs().size());
    assertEquals("E_BASE", p.getNodes().get(1).getEventInputs().get(0));
}

@Test
public void test_singleWriterViolation_detected() {
    String json =
        "{\n" +
        "  \"recipeVersion\": \"1.0\", \n" +
        "  \"id\": \"tsir_demo_single_writer_bad\", \n" +
        "  \"sources\": [\n" +
        "    {\n" +
        "      \"name\": \"CURVE_A\", \"type\": \"curve\", \"ref\": \"SPY:1D\", \"slice\": {\"from\": \"2020-01-01\", \"to\": \"2024-12-31\"}}\n" +
        "    ], \n" +
        "    \"operators\": {\n" +
        "      \"ex1\": {\n" +
        "        \"kind\": \"extractor\", \"impl\": \"X\", \"params\": {}}, \n" +
        "      \"ex2\": {\n" +
        "        \"kind\": \"extractor\", \"impl\": \"Y\", \"params\": {}}\n" +
        "    }, \n" +
        "    \"graph\": {\n" +
        "      \"mode\": \"sequence\", \n" +
        "      \"nodes\": [\n" +
        "        {\n" +
        "          \"id\": \"n1\", \"op\": \"ex1\", \"in\": {\"curves\": [\"CURVE_A\"]}, \"events\": []}, \n" +
        "          \"out\": {\"events\": \"E_DUP\"}}, \n" +
        "        {\n" +
        "          \"id\": \"n2\", \"op\": \"ex2\", \"in\": {\"curves\": [\"CURVE_A\"]}, \"events\": []}, \n" +
        "          \"out\": {\"events\": \"E_DUP\"}}\n" +
        "    ]\n" +
        "  }, \n" +
        "  \"artifacts\": {\"events\": [\"E_DUP\"], \"manifest\":true}\n" +
"}\n";

    TsirRecipe r = parser.parse(json);
    List<TsirRecipeValidationIssue> issues = validator.validate(r);

    boolean found = false;
    for (TsirRecipeValidationIssue i : issues) {
        if (i.getCode() ==
TsirRecipeValidationIssue.Code.SINGLE_WRITER_VIOLATION) {
            found = true;
            break;
        }
    }
}

```

```

        }
    }
    assertTrue("Expected SINGLE_WRITER_VIOLATION", found);
}

@Test
public void test_forwardEventReference_detected() {
    // Node n1 consumes E_LATER before it is produced by n2 (sequence mode).
    String json =
        "{\n" +
            "  \"recipeVersion\": \"1.0\", \n" +
            "  \"id\": \"tsir_demo_forward_ref_bad\", \n" +
            "  \"sources\": [\n" +
            "    {\n" +
            "      \"name\": \"CURVE_A\", \"type\": \"curve\", \"ref\": \"SPY:1D\", \"slice\": {\"from\": \"2020-01-01\", \"to\": \"2024-12-31\"}}\n" +
            "    ], \n" +
            "    \"operators\": {\n" +
            "      \"ex1\": {\n" +
            "        \"kind\": \"extractor\", \"impl\": \"X\", \"params\": {}}, \n" +
            "      \"ex2\": {\n" +
            "        \"kind\": \"extractor\", \"impl\": \"Y\", \"params\": {}}\n" +
            "    }, \n" +
            "    \"graph\": {\n" +
            "      \"mode\": \"sequence\", \n" +
            "      \"nodes\": [\n" +
            "        {\n" +
            "          \"id\": \"n1\", \"op\": \"ex1\", \"in\": {\"curves\": [\"CURVE_A\"]}, \"events\": [\"E_LATER\"]}, \n" +
            "          \"out\": {\"events\": \"E_NOW\"}}, \n" +
            "        {\n" +
            "          \"id\": \"n2\", \"op\": \"ex2\", \"in\": {\"curves\": [\"CURVE_A\"]}, \"events\": []}, \n" +
            "          \"out\": {\"events\": \"E_LATER\"}}\n" +
            "        ], \n" +
            "        \"artifacts\": {\"events\": [\"E_NOW\"], \"manifest\": true}\n" +
            "    }\n" +
            "  }\n" +
            "TsirRecipe r = parser.parse(json);\n" +
            "List<TsirRecipeValidationIssue> issues = validator.validate(r);\n\n" +
            "boolean found = false;\n" +
            "for (TsirRecipeValidationIssue i : issues) {\n" +
            "    if (i.getCode() ==\n" +
            "TsirRecipeValidationIssue.Code.FORWARD_EVENT_REFERENCE) {\n" +
            "        found = true;\n" +
            "        break;\n" +
            "    }\n" +
            "}\n" +
            "assertTrue("Expected FORWARD_EVENT_REFERENCE", found);\n"
}

@Test
public void test_unknownOperator_detected() {
    String json =
        "{\n" +
            "  \"recipeVersion\": \"1.0\", \n" +

```

```

    "  \"id\": \"tsir_demo_unknown_op_bad\",\\n\" +
    \"  \"sources\": [\\n\" +
    \"\n
{\\"name\":\\\"CURVE_A\\\",\\\"type\\\":\\\"curve\\\",\\\"ref\\\":\\\"SPY:1D\\\",\\\"slice\\\":{\\\"from
\\\":\\\"2020-01-01\\\",\\\"to\\\":\\\"2024-12-31\\\"}}\\n\" +
    \"  ],\\n\" +
    \"  \"operators\": {\\n\" +
    \"    \\\"ex1\\\":
{\\"kind\\\":\\\"extractor\\\",\\\"impl\\\":\\\"X\\\",\\\"params\\\":{}\\}\\n\" +
        \"  },\\n\" +
        \"  \\\"graph\\\": {\\n\" +
        \"    \\\"mode\\\": \\\"sequence\\\",\\n\" +
        \"    \\\"nodes\\\": [\\n\" +
        \"\n
{\\"id\\\":\\\"n1\\\",\\\"op\\\":\\\"MISSING_OP\\\",\\\"in\\\":{\\\"curves\\\": [\\\"CURVE_A\\\"],\\\"event
s\\\":[]},\\\"out\\\":{\\\"events\\\":\\\"E1\\\"}}\\n\" +
        \"      ]\\n\" +
        \"    },\\n\" +
        \"    \\\"artifacts\\\": {\\\"events\\\": [\\\"E1\\\"],\\\"manifest\\\":true}\\n\" +
        \"}\\n\";
    }

TsirRecipe r = parser.parse(json);
List<TsirRecipeValidationIssue> issues = validator.validate(r);

boolean found = false;
for (TsirRecipeValidationIssue i : issues) {
    if (i.getCode() ==
TsirRecipeValidationIssue.Code.UNKNOWN_OPERATOR) {
        found = true;
        break;
    }
}
assertTrue("Expected UNKNOWN_OPERATOR", found);
}
}

```

你落地时的两个小提示（很关键）

1. JSON 解析依赖

这份 skeleton 的 `TsirRecipeParser` 默认用 Jackson (通过反射调用，避免强依赖)。如果你 repo 里本来就有 Jackson，它会直接工作；没有的话会抛出清晰错误。你也可以把 `TsirRecipeParser` 换成你已有的 JSON loader (不影响 Validator/Compiler)。

2. 错误码可直接进入 Explain/Manifest

`TsirRecipeValidationIssue` 的 `code/path/message` 三元组是“工程可用”的：以后你做 Run Manifest、Plan explain、以及前端 UI 展示时，都不需要再重构。

如果你下一步要把它更“DBM 风味”地并入你现有 runtime (例如 IrPipelineV2) , 我建议我接着补一层 :

- TsirRuntimeFactory : 把 TsirPlan.PlanNode 的 operatorImpl 反射/Registry 实例化成 IEventExtractor
- EventBus/DataBus 的最小接口 (可先用 Map-backed MVP) , 把 PlanNode 跑起来并产出 manifest

你一句话 , 我就按你当前 com.dbm.ir.runtime 的接口命名风格给出 drop-in 版本。

请按你建议的进一步要把它更“DBM 风味”地并入现有 runtime (例如 IrPipelineV2)

```
// File: src/main/java/com/dbm/ir/runtime/TsirRuntime.java
package com.dbm.ir.runtime;

import java.util.*;

/**
 * "DBM-flavor" runtime integration layer:
 * Recipe(JSON) -> Plan -> Run -> Artifacts
 *
 * This layer is intentionally thin:
 * - It does NOT replace IrPipelineV2; it composes into it.
 * - It offers a clean Task(Action) boundary for APTGOE-style evolution.
 */
public final class TsirRuntime {

    private final TsirRecipeParser parser;
    private final TsirRecipeValidator validator;
    private final TsirPlanCompiler compiler;

    private final TsirExtractorRegistry extractorRegistry;

    public TsirRuntime() {
        this(new TsirRecipeParser(),
            new TsirRecipeValidator(),
            new TsirPlanCompiler(),
            TsirExtractorRegistry.defaultRegistry());
    }

    public TsirRuntime(TsirRecipeParser parser,
                      TsirRecipeValidator validator,
                      TsirPlanCompiler compiler,
                      TsirExtractorRegistry extractorRegistry) {
        this.parser = parser;
        this.validator = validator;
        this.compiler = compiler;
        this.extractorRegistry = extractorRegistry;
    }
}
```

```

/** Convenience: JSON -> (parse+validate+compile) -> run. */
public TsirRunResult runJsonOrThrow(String recipeJson, TsirCurveProvider
curveProvider) {
    TsirRecipe recipe = parser.parse(recipeJson);
    validator.validateOrThrow(recipe);
    TsirPlan plan = compiler.compileOrThrow(recipe);
    return runPlanOrThrow(recipe, plan, curveProvider);
}

/** Main entry: run an already-compiled plan. */
public TsirRunResult runPlanOrThrow(TsirRecipe recipe, TsirPlan plan,
TsirCurveProvider curveProvider) {
    Objects.requireNonNull(recipe, "recipe");
    Objects.requireNonNull(plan, "plan");
    Objects.requireNonNull(curveProvider, "curveProvider");

    long startedAtMs = System.currentTimeMillis();

    TsirManifest manifest = TsirManifest.start(plan.getRecipeId(),
plan.getRecipeVersion());
    manifest.put("recipeHash",
TsirRecipeHash.canonicalHashOrPlaceholder(recipe)); // optional; stable key
    manifest.put("graphMode", "sequence");
    manifest.put("manifestRequired",
String.valueOf(plan.isManifestRequired()));

    // --- DBM-flavor buses (map-backed MVP) ---
    TsirDataBus dataBus = new TsirMapDataBus();
    TsirEventBus eventBus = new TsirMapEventBus();

    // --- Load source curves into DataBus ---
    Map<String, TsirCurve> curvesByName = loadCurves(recipe,
curveProvider, manifest);
    dataBus.putAllCurves(curvesByName);

    // --- Runtime guards (MVP defaults; can be promoted to governance
later) ---
    TsirRuntimeGuards guards = TsirRuntimeGuards.defaults();

    // --- Execute plan nodes in order ---
    for (int i = 0; i < plan.getNodes().size(); i++) {
        TsirPlan.PlanNode node = plan.getNodes().get(i);
        String nodeId = node.getNodeId();

        long nodeStart = System.currentTimeMillis();
        manifest.nodeStart(nodeId, node.getOperatorId(),
node.getOperatorImpl(), node.getOperatorParams());

        // Resolve extractor
        TsirExtractor ex =
extractorRegistry.resolveOrThrow(node.getOperatorImpl(),
node.getOperatorParams());

        // Build per-node context
        TsirRuntimeContext ctx = new TsirRuntimeContext(
            dataBus,
            eventBus,

```

```

        guards,
        manifest,
        node
    );

    try {
        ex.extract(ctx); // extractor writes to EventBus
        manifest.nodeOk(nodeId, System.currentTimeMillis() -
nodeStart);
    } catch (Exception e) {
        manifest.nodeFail(nodeId, System.currentTimeMillis() -
nodeStart, e);
        // Constitution-level recommendation: still return partial
manifest/artifacts.
        // Here we throw by default (strict). You can add a
"failSoft" policy later.
        throw new TsirRuntimeException("Node failed: " + nodeId + "
(" + node.getOperatorImpl() + ")", e);
    }

    // Safety rails: cap event explosion
    guards.assertEventBudget(eventBus);
}

// --- Collect artifacts ---
Map<String, List<TsirEvent>> outEvents = new LinkedHashMap<>();
for (String stream : plan.getOutputEventStreams()) {
    outEvents.put(stream, eventBus.getEvents(stream));
}

manifest.finish(System.currentTimeMillis() - startedAtMs);

// Note: observation/summary layers can be added later; MVP focuses
on Event + Manifest.
return new TsirRunResult(outEvents, manifest);
}

private static Map<String, TsirCurve> loadCurves(TsirRecipe recipe,
TsirCurveProvider curveProvider, TsirManifest manifest) {
    Map<String, TsirCurve> m = new LinkedHashMap<>();
    for (TsirRecipe.SourceSpec s : recipe.getSources()) {
        TsirCurve curve = curveProvider.loadCurve(s);
        if (curve == null) {
            throw new TsirRuntimeException("CurveProvider returned null
for source: " + s.getName());
        }
        m.put(s.getName(), curve);
        manifest.put("source." + s.getName(), s.getRef() + " [" +
s.getSlice().getFrom() + "," + s.getSlice().getTo() + "]");
    }
    return m;
}
// File: src/main/java/com/dbm/ir/runtime/TsirRuntimeException.java
package com.dbm.ir.runtime;

public final class TsirRuntimeException extends RuntimeException {

```

```

    public TsirRuntimeException(String message) { super(message); }
    public TsirRuntimeException(String message, Throwable cause)
{ super(message, cause); }
}
// File: src/main/java/com/dbm/ir/runtime/TsirRunResult.java
package com.dbm.ir.runtime;

import java.util.*;

public final class TsirRunResult {
    private final Map<String, List<TsirEvent>> eventStreams;
    private final TsirManifest manifest;

    public TsirRunResult(Map<String, List<TsirEvent>> eventStreams,
TsirManifest manifest) {
        this.eventStreams = Collections.unmodifiableMap(new
LinkedHashMap<>(eventStreams));
        this.manifest = manifest;
    }

    public Map<String, List<TsirEvent>> getEventStreams() { return
eventStreams; }
    public TsirManifest getManifest() { return manifest; }

    public List<TsirEvent> eventsOf(String streamName) {
        List<TsirEvent> xs = eventStreams.get(streamName);
        return xs == null ? Collections.<TsirEvent>emptyList() : xs;
    }
}
// File: src/main/java/com/dbm/ir/runtime/TsirManifest.java
package com.dbm.ir.runtime;

import java.util.*;

/**
 * Minimal run manifest. Keep it "flat" + node logs for easy printing,
hashing, auditing.
 */
public final class TsirManifest {

    public static final class NodeLog {
        private final String nodeId;
        private final String operatorId;
        private final String operatorImpl;
        private final Map<String, Object> operatorParams;

        private final long startedAtMs;
        private long elapsedMs;
        private boolean ok;
        private String error;

        NodeLog(String nodeId, String operatorId, String operatorImpl,
Map<String, Object> operatorParams, long startedAtMs) {
            this.nodeId = nodeId;
            this.operatorId = operatorId;
            this.operatorImpl = operatorImpl;

```

```

        this.operatorParams = operatorParams == null ?
Collections.<String, Object>emptyMap()
        : Collections.unmodifiableMap(new
LinkedHashMap<>(operatorParams));
        this.startedAtMs = startedAtMs;
    }

    public String getNodeId() { return nodeId; }
    public String getOperatorId() { return operatorId; }
    public String getOperatorImpl() { return operatorImpl; }
    public Map<String, Object> getOperatorParams() { return
operatorParams; }
    public long getStartedAtMs() { return startedAtMs; }
    public long getElapsedMs() { return elapsedMs; }
    public boolean isOk() { return ok; }
    public String getError() { return error; }
}

private final Map<String, String> kv = new LinkedHashMap<>();
private final List<NodeLog> nodes = new ArrayList<>();

private final String recipeId;
private final String recipeVersion;
private final long startedAtMs;

private long elapsedMs;

private TsirManifest(String recipeId, String recipeVersion) {
    this.recipeId = recipeId;
    this.recipeVersion = recipeVersion;
    this.startedAtMs = System.currentTimeMillis();
}

public static TsirManifest start(String recipeId, String recipeVersion) {
    TsirManifest m = new TsirManifest(recipeId, recipeVersion);
    m.put("recipeId", recipeId);
    m.put("recipeVersion", recipeVersion);
    m.put("startedAtMs", String.valueOf(m.startedAtMs));
    return m;
}

public void put(String k, String v) { kv.put(k, v); }

public void nodeStart(String nodeId, String operatorId, String
operatorImpl, Map<String, Object> params) {
    nodes.add(new NodeLog(nodeId, operatorId, operatorImpl, params,
System.currentTimeMillis()));
}

public void nodeOk(String nodeId, long elapsedMs) {
    NodeLog n = find(nodeId);
    n.ok = true;
    n.elapsedMs = elapsedMs;
}

public void nodeFail(String nodeId, long elapsedMs, Exception e) {
    NodeLog n = find(nodeId);
}

```

```

        n.ok = false;
        n.elapsedMs = elapsedMs;
        n.error = e.getClass().getSimpleName() + ":" + e.getMessage();
    }

    public void finish(long elapsedMs) {
        this.elapsedMs = elapsedMs;
        put("elapsedMs", String.valueOf(elapsedMs));
    }

    public String getRecipeId() { return recipeId; }
    public String getRecipeVersion() { return recipeVersion; }
    public long getStartedAtMs() { return startedAtMs; }
    public long getElapsedMs() { return elapsedMs; }
    public Map<String, String> getKv() { return
Collections.unmodifiableMap(kv); }
    public List<NodeLog> getNodes() { return
Collections.unmodifiableList(nodes); }

    private NodeLog find(String nodeId) {
        for (int i = nodes.size() - 1; i >= 0; i--) {
            NodeLog n = nodes.get(i);
            if (n.nodeId != null && n.nodeId.equals(nodeId)) return n;
        }
        // fallback: last
        if (!nodes.isEmpty()) return nodes.get(nodes.size() - 1);
        throw new IllegalStateException("No node log exists for nodeId=" +
nodeId);
    }
}
// File: src/main/java/com/dbm/ir/runtime/TsirRecipeHash.java
package com.dbm.ir.runtime;

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;

/**
 * Optional utility: stable hash key for caching/audit.
 * In production, use canonical JSON (sorted keys, consistent arrays).
 * Here we provide a placeholder strategy: hash of a few stable fields.
 */
public final class TsirRecipeHash {

    private TsirRecipeHash() {}

    public static String canonicalHashOrPlaceholder(TsirRecipe recipe) {
        try {
            // Placeholder: hash a stable concatenation.
            StringBuilder sb = new StringBuilder();

            sb.append(recipe.getRecipeVersion()).append("|").append(recipe.getId()).append("|");
            if (recipe.getSources() != null) {
                for (TsirRecipe.SourceSpec s : recipe.getSources()) {

                    sb.append(s.getName()).append("@").append(s.getRef()).append("|");
                }
            }
        }
    }
}

```

```

        }
        if (recipe.getGraph() != null && recipe.getGraph().getNodes() != null) {
            for (TsirRecipe.NodeSpec n : recipe.getGraph().getNodes()) {
                sb.append(n.getId()).append(":").append(n.getOp()).append("->")
                    .append(n.getOut() == null ? "" :
n.getOut().getEvents()).append("|");
            }
        }
        return sha256Hex(sb.toString());
    } catch (Exception e) {
        return "HASH_ERROR";
    }
}

private static String sha256Hex(String s) throws Exception {
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    byte[] dig = md.digest(s.getBytes(StandardCharsets.UTF_8));
    StringBuilder hex = new StringBuilder();
    for (byte b : dig) {
        String h = Integer.toHexString(b & 0xff);
        if (h.length() == 1) hex.append('0');
        hex.append(h);
    }
    return hex.toString();
}
}

// File: src/main/java/com/dbm/ir/runtime/TsirDataBus.java
package com.dbm.ir.runtime;

import java.util.Map;

public interface TsirDataBus {
    TsirCurve getCurve(String sourceName);
    void putCurve(String sourceName, TsirCurve curve);
    void putAllCurves(Map<String, TsirCurve> curves);
}
}

// File: src/main/java/com/dbm/ir/runtime/TsirMapDataBus.java
package com.dbm.ir.runtime;

import java.util.*;

public final class TsirMapDataBus implements TsirDataBus {
    private final Map<String, TsirCurve> curves = new LinkedHashMap<>();

    @Override
    public TsirCurve getCurve(String sourceName) {
        return curves.get(sourceName);
    }

    @Override
    public void putCurve(String sourceName, TsirCurve curve) {
        curves.put(sourceName, curve);
    }

    @Override

```

```

        public void putAllCurves(Map<String, TsirCurve> curves) {
            if (curves != null) this.curves.putAll(curves);
        }
    }
// File: src/main/java/com/dbm/ir/runtime/TsirEventBus.java
package com.dbm.ir.runtime;

import java.util.List;

public interface TsirEventBus {
    void appendEvent(String stream, TsirEvent e);
    void appendAll(String stream, List<TsirEvent> events);
    List<TsirEvent> getEvents(String stream);
    int countAllEvents();
}
// File: src/main/java/com/dbm/ir/runtime/TsirMapEventBus.java
package com.dbm.ir.runtime;

import java.util.*;

public final class TsirMapEventBus implements TsirEventBus {
    private final Map<String, List<TsirEvent>> streams = new
    LinkedHashMap<>();

    @Override
    public void appendEvent(String stream, TsirEvent e) {
        if (stream == null) throw new IllegalArgumentException("stream is
null");
        if (e == null) return;
        streams.computeIfAbsent(stream, k -> new
ArrayList<TsirEvent>()).add(e);
    }

    @Override
    public void appendAll(String stream, List<TsirEvent> events) {
        if (events == null || events.isEmpty()) return;
        for (TsirEvent e : events) appendEvent(stream, e);
    }

    @Override
    public List<TsirEvent> getEvents(String stream) {
        List<TsirEvent> xs = streams.get(stream);
        if (xs == null) return Collections.emptyList();
        return Collections.unmodifiableList(xs);
    }

    @Override
    public int countAllEvents() {
        int n = 0;
        for (List<TsirEvent> xs : streams.values()) n += xs.size();
        return n;
    }
}
// File: src/main/java/com/dbm/ir/runtime/TsirRuntimeGuards.java
package com.dbm.ir.runtime;

/**

```

```

 * Default safety rails. Promote to recipe.governance later.
 */
public final class TsirRuntimeGuards {

    private final int maxTotalEvents;

    private TsirRuntimeGuards(int maxTotalEvents) {
        this.maxTotalEvents = maxTotalEvents;
    }

    public static TsirRuntimeGuards defaults() {
        return new TsirRuntimeGuards(200_000); // safe default; tune later
    }

    public void assertEventBudget(TsirEventBus bus) {
        if (bus.countAllEvents() > maxTotalEvents) {
            throw new TsirRuntimeException("Event budget exceeded: " +
bus.countAllEvents() + " > " + maxTotalEvents);
        }
    }
}
// File: src/main/java/com/dbm/ir/runtime/TsirRuntimeContext.java
package com.dbm.ir.runtime;

import java.util.*;

/**
 * Per-node runtime context.
 * DBM flavor: node executes with buses + guards + manifest.
 */
public final class TsirRuntimeContext {
    private final TsirDataBus dataBus;
    private final TsirEventBus eventBus;
    private final TsirRuntimeGuards guards;
    private final TsirManifest manifest;
    private final TsirPlan.PlanNode node;

    public TsirRuntimeContext(TsirDataBus dataBus,
                           TsirEventBus eventBus,
                           TsirRuntimeGuards guards,
                           TsirManifest manifest,
                           TsirPlan.PlanNode node) {
        this.dataBus = dataBus;
        this.eventBus = eventBus;
        this.guards = guards;
        this.manifest = manifest;
        this.node = node;
    }

    public TsirDataBus dataBus() { return dataBus; }
    public TsirEventBus eventBus() { return eventBus; }
    public TsirRuntimeGuards guards() { return guards; }
    public TsirManifest manifest() { return manifest; }
    public TsirPlan.PlanNode node() { return node; }

    public TsirCurve curve0OrThrow() {
        List<String> cs = node.getCurveInputs();

```

```

        if (cs == null || cs.isEmpty()) throw new TsirRuntimeException("No
curve inputs for node " + node.getNodeId());
        TsirCurve c = dataBus.getCurve(cs.get(0));
        if (c == null) throw new TsirRuntimeException("Curve not found: " +
cs.get(0));
        return c;
    }

    public List<TsirEvent> inEvents0() {
        List<String> ins = node.getEventInputs();
        if (ins == null || ins.isEmpty()) return
java.util.Collections.emptyList();
        return eventBus.getEvents(ins.get(0));
    }

    public String outStream() {
        return node.getEventOutput();
    }

    public Map<String, Object> params() {
        return node.getOperatorParams();
    }
}
// File: src/main/java/com/dbm/ir/runtime/TsirCurveProvider.java
package com.dbm.ir.runtime;

/**
 * Adapter point: plug your existing curve loaders (CSV, DB, in-memory demo).
 */
public interface TsirCurveProvider {
    TsirCurve loadCurve(TsirRecipe.SourceSpec source);
}
// File: src/main/java/com/dbm/ir/runtime/TsirCurve.java
package com.dbm.ir.runtime;

/**
 * Minimal curve interface (MVP).
 * You can adapt this to your existing curve model (e.g., your time-series
wrappers).
 */
public interface TsirCurve {
    int size();
    long t(int idx);          // timestamp or logical time
    double v(int idx);        // value
}
// File: src/main/java/com/dbm/ir/runtime/TsirEvent.java
package com.dbm.ir.runtime;

import java.util.*;

/**
 * Minimal event shape (MVP).
 * Later you can map to your existing EventObservation / IR event classes.
 */
public final class TsirEvent {
    private final long t0;
    private final long t1;
}

```

```

private final String type;
private final double score;
private final Map<String, Object> tags;

private TsirEvent(long t0, long t1, String type, double score,
Map<String, Object> tags) {
    this.t0 = t0;
    this.t1 = t1;
    this.type = type;
    this.score = score;
    this.tags = tags == null ? Collections.<String, Object>emptyMap()
        : Collections.unmodifiableMap(new LinkedHashMap<>(tags));
}

public static TsirEvent of(long t0, long t1, String type, double score) {
    return new TsirEvent(t0, t1, type, score, null);
}

public static TsirEvent of(long t0, long t1, String type, double score,
Map<String, Object> tags) {
    return new TsirEvent(t0, t1, type, score, tags);
}

public long getT0() { return t0; }
public long getT1() { return t1; }
public String getType() { return type; }
public double getScore() { return score; }
public Map<String, Object> getTags() { return tags; }
}
// File: src/main/java/com/dbm/ir/runtime/TsirExtractor.java
package com.dbm.ir.runtime;

/**
 * DBM flavor:
 * - Extractor is a "machine" on the pipeline bus.
 * - It reads from DataBus/EventBus and writes to EventBus(outStream).
 *
 * You can add capability introspection later (inputs/outputs schema).
 */
public interface TsirExtractor {
    void extract(TsirRuntimeContext ctx);
}
// File: src/main/java/com/dbm/ir/runtime/TsirExtractorRegistry.java
package com.dbm.ir.runtime;

import java.lang.reflect.Constructor;
import java.util.*;

/**
 * Registry supports:
 * - Reflective instantiation by class name (impl field)
 * - Optional param injection if extractor has ctor(Map) or default ctor +
configure(Map)
 *
 * This is the "bridge" into your existing IrPipelineV2 ecosystem:
 * you can register adapters that wrap existing extractors/operators.
*/

```

```

public final class TsirExtractorRegistry {

    public interface Factory {
        TsirExtractor create(Map<String, Object> params);
    }

    private final Map<String, Factory> factories = new HashMap<>();

    public static TsirExtractorRegistry defaultRegistry() {
        TsirExtractorRegistry r = new TsirExtractorRegistry();

        // You can pre-register known extractors here:
        // r.register("com.dbm.ir.runtime.MvpUpDownExtractor", params -> new
MvpUpDownExtractorAdapter(params));

        return r;
    }

    public void register(String impl, Factory f) {
        factories.put(impl, f);
    }

    public TsirExtractor resolveOrThrow(String impl, Map<String, Object>
params) {
        if (impl == null || impl.trim().isEmpty()) {
            throw new TsirRuntimeException("Operator impl is blank");
        }
        Factory f = factories.get(impl);
        if (f != null) return f.create(params);

        // Fallback: reflect class and instantiate.
        try {
            Class<?> cls = Class.forName(impl);
            if (!TsirExtractor.class.isAssignableFrom(cls)) {
                throw new TsirRuntimeException("Class does not implement
TsirExtractor: " + impl);
            }
        }

        // Try ctor(Map)
        try {
            @SuppressWarnings("unchecked")
            Constructor<? extends TsirExtractor> c = (Constructor<?
extends TsirExtractor>) cls.getConstructor(Map.class);
            return c.newInstance(params == null ?
Collections.emptyMap() : params);
        } catch (NoSuchMethodException ignored) {
            // Try default ctor
            @SuppressWarnings("unchecked")
            Constructor<? extends TsirExtractor> c0 = (Constructor<?
extends TsirExtractor>) cls.getConstructor();
            TsirExtractor ex = c0.newInstance();

            // Optional configure(Map) method
            try {
                cls.getMethod("configure", Map.class).invoke(ex, params
== null ? Collections.emptyMap() : params);
            } catch (NoSuchMethodException ignored2) {

```

```

        // ok: extractor may ignore params
    }
    return ex;
}
} catch (Exception e) {
    throw new TsirRuntimeException("Failed to instantiate extractor:
" + impl, e);
}
}
}

```

如何“贴合并入 IrPipelineV2”（你 repo 内落地方式）

上面这套 runtime 是一个 **DBM 风味的“Recipe/Plan 入口”**。要并入你现有 `IrPipelineV2`，我建议采用 **最稳的桥接策略**：

策略：`IrPipelineV2` 继续做“底层执行框架”，`TsirRuntime` 做“上层编排入口”

- `TsirPlanCompiler` 把 JSON 变成 `TsirPlan`
- `TsirRuntime` 用 `TsirExtractorRegistry` 将每个 `PlanNode` 解析成可执行机器
- 每台机器可以是：
 1. **原生 TsirExtractor** (新写的、纯 recipe 化)
 2. **Adapter**：把你已有的 `MvpUpDownExtractor` 等接入（见下面示例）

示例：把你已有的 `MvpUpDownExtractor` 接成 `TsirExtractor` (Adapter 方式)

你已有的 `com.dbm.ir.runtime.MvpUpDownExtractor` 很可能不是本文的 `TsirExtractor` 接口；

适配器是最稳的方法：不改旧代码，直接“贴合”新 recipe runtime。

```

// File:
src/main/java/com/dbm/ir/runtime/adapters/MvpUpDownExtractorAdapter.java
package com.dbm.ir.runtime.adapters;

import com.dbm.ir.runtime.*;
import java.util.*;

/**
 * Adapter example. Replace the inside with your real MvpUpDownExtractor call
 * pattern.
 */

```

```

public final class MvpUpDownExtractorAdapter implements TsirExtractor {

    private int window = 5;
    private double threshold = 0.0;

    public MvpUpDownExtractorAdapter() {}

    public MvpUpDownExtractorAdapter(Map<String, Object> params) {
        configure(params);
    }

    public void configure(Map<String, Object> params) {
        if (params == null) return;
        Object w = params.get("window");
        if (w instanceof Number) window = ((Number) w).intValue();
        Object t = params.get("threshold");
        if (t instanceof Number) threshold = ((Number) t).doubleValue();
    }

    @Override
    public void extract(TsirRuntimeContext ctx) {
        TsirCurve c = ctx.curve0OrThrow();

        // MVP: naive "up/down" turning points; replace with your real
        extractor outputs.
        for (int i = window; i < c.size(); i++) {
            double dv = c.v(i) - c.v(i - window);
            if (dv >= threshold) {
                ctx.eventBus().appendEvent(ctx.outStream(),
TsirEvent.of(c.t(i - window), c.t(i), "UP", dv));
            } else if (dv <= -threshold) {
                ctx.eventBus().appendEvent(ctx.outStream(),
TsirEvent.of(c.t(i - window), c.t(i), "DOWN", -dv));
            }
        }
    }
}

```

然后把它注册到 registry (推荐在你项目启动时注册) :

```

// Somewhere in your bootstrap:
TsirExtractorRegistry reg = TsirExtractorRegistry.defaultRegistry();
reg.register("com.dbm.ir.runtime.MvpUpDownExtractor", params -> new
com.dbm.ir.runtime.adapters.MvpUpDownExtractorAdapter(params));
TsirRuntime rt = new TsirRuntime(new TsirRecipeParser(), new
TsirRecipeValidator(), new TsirPlanCompiler(), reg);

```

这样，你 recipe 里 impl 依旧写你原来的类名，但实际运行会走 adapter (非常 DBM : 桥接/分层/不破坏既有资产)。

下一步（我建议你现在就做的“最小闭环”）

为了让它真正并入你的 `IrPipelineV2` 风格，我建议我接着补两项（你一句话即可，我会直接给 drop-in 代码）：

1. **IrPipelineV2Bridge**：把 `TsirPlan.PlanNode` 变成你现有 pipeline step（如果你 pipeline 有 `addExtractor(...)` 或类似 DSL）
2. **Demo_TsirRecipeRun.java**：用一个内存曲线 provider + recipe JSON，跑出 events + manifest，并打印 explain（节点顺序、入/出流）

你现在只需要回复一句：

- “按我 repo 里 `IrPipelineV2` 的 step 接口名是 XXX / extractor 接口名是 YYY”
如果你不想解释接口，我也可以按“最小假设”写一个桥接类，你再把 1-2 个方法名对齐即可。

请按“最小假设”写一个桥接类，我再把 1-2 个方法名对齐。

```
// File: src/main/java/com/dbm/ir/runtime/TsirIrPipelineV2Bridge.java
package com.dbm.ir.runtime;

import java.lang.reflect.Method;
import java.util.*;

/**
 * Minimal-assumption bridge: TsirPlan -> IrPipelineV2 (your existing
 * runtime).
 *
 * Philosophy:
 * - We do NOT assume your IrPipelineV2 API precisely.
 * - We use reflective calls with a small set of conventional method names.
 * - You will likely only need to rename/adjust 1-2 method names in resolve*
 * methods.
 *
 * Supported integration styles (auto-detected):
 * A) IrPipelineV2 has method: addExtractor(Object extractor) /
 * addStep(Object)
 * B) IrPipelineV2 has method: addNode(String id, Object op) / add(String
 * id, Object)
 * C) IrPipelineV2 has method: run() / execute() / runAndCollect()
 *
 * Extractor instantiation:
 * - Uses TsirExtractorRegistry to create TsirExtractor, then wraps it into
 * an IrPipelineV2-compatible step
 * via a MapBackedStep adapter (provided below).
```

```

/*
 * IMPORTANT:
 * - This bridge is a "glue layer" to rapidly embed the Recipe->Plan concept
into existing code,
 * then you can refactor to a typed integration later.
 */
public final class TsirIrPipelineV2Bridge {

    private final TsirExtractorRegistry registry;

    public TsirIrPipelineV2Bridge(TsirExtractorRegistry registry) {
        this.registry = registry == null ?
TsirExtractorRegistry.defaultRegistry() : registry;
    }

    /**
     * Builds an IrPipelineV2 instance (already created by caller) by adding
steps based on TsirPlan.
     * Returns the same pipeline object to allow fluent usage.
     */
    public Object attachPlanToPipeline(Object irPipelineV2, TsirPlan plan) {
        Objects.requireNonNull(irPipelineV2, "irPipelineV2");
        Objects.requireNonNull(plan, "plan");

        for (TsirPlan.PlanNode pn : plan.getNodes()) {
            // Create TsirExtractor from registry (impl + params)
            TsirExtractor ex = registry.resolveOrThrow(pn.getOperatorImpl(),
pn.getOperatorParams());

            // Wrap into a step object that can run with map-backed buses.
            Object step = new MapBackedStep(pn, ex);

            // Attach the step to IrPipelineV2 using reflective "best-effort"
conventions.
            if (!tryAddStep(irPipelineV2, pn, step)) {
                throw new TsirRuntimeException(
                    "Failed to attach step to IrPipelineV2. " +
                    "Please align 1-2 method names in
TsirIrPipelineV2Bridge.tryAddStep(...) for your pipeline API. " +
                    "PlanNode=" + pn.getNodeId() + ", impl=" +
pn.getOperatorImpl());
            }
        }
        return irPipelineV2;
    }

    /**
     * Executes IrPipelineV2 in a minimal-assumption way.
     * You can ignore this and call your existing pipeline runner directly.
     */
    public Object runPipeline(Object irPipelineV2) {
        Objects.requireNonNull(irPipelineV2, "irPipelineV2");
        // Try common method names.
        if (invokeNoArgIfExists(irPipelineV2, "run")) return null;
        if (invokeNoArgIfExists(irPipelineV2, "execute")) return null;
        if (invokeNoArgIfExists(irPipelineV2, "runAndCollect")) return null;
        if (invokeNoArgIfExists(irPipelineV2, "runMvp")) return null;
    }
}

```

```

        throw new TsirRuntimeException(
            "Cannot find a runnable method on IrPipelineV2 (tried:
run/execute/runAndCollect/runMvp). " +
            "Please call your pipeline run method directly, or extend
runPipeline() .");
    }

    // -----
    // Reflection integration
    // -----
}

private boolean tryAddStep(Object pipeline, TsirPlan.PlanNode pn, Object
step) {
    // Style A: addExtractor(step)
    if (invokeOneArgIfExists(pipeline, "addExtractor", step)) return
true;

    // Style A-alt: addStep(step)
    if (invokeOneArgIfExists(pipeline, "addStep", step)) return true;

    // Style A-alt: add(step)
    if (invokeOneArgIfExists(pipeline, "add", step)) return true;

    // Style B: addNode(nodeId, step)
    if (invokeTwoArgsIfExists(pipeline, "addNode", pn.getNodeId(), step))
return true;

    // Style B-alt: add(nodeId, step)
    if (invokeTwoArgsIfExists(pipeline, "add", pn.getNodeId(), step))
return true;

    // Style B-alt: put(nodeId, step)
    if (invokeTwoArgsIfExists(pipeline, "put", pn.getNodeId(), step))
return true;

    // If your pipeline needs more info (inputs/outputs), add an overload
here.
    // Example: addStep(nodeId, step, inCurves, inEvents, outEvents)
    if (invokeFiveArgsIfExists(pipeline, "addStep",
        pn.getNodeId(),
        step,
        pn.getCurveInputs(),
        pn.getEventInputs(),
        pn.getEventOutput())) return true;

    return false;
}

private static boolean invokeNoArgIfExists(Object target, String method)
{
    try {
        Method m = target.getClass().getMethod(method);
        m.invoke(target);
        return true;
    } catch (NoSuchMethodException e) {
        return false;
    }
}

```

```

        } catch (Exception e) {
            throw new TsirRuntimeException("Failed to invoke " + method +
"(): " + e.getMessage(), e);
        }
    }

    private static boolean invokeOneArgIfExists(Object target, String method,
Object arg1) {
    try {
        Method m = findMethodByNameAndArity(target.getClass(), method,
1);
        if (m == null) return false;
        m.invoke(target, arg1);
        return true;
    } catch (Exception e) {
        throw new TsirRuntimeException("Failed to invoke " + method + "(1
arg): " + e.getMessage(), e);
    }
}

private static boolean invokeTwoArgsIfExists(Object target, String
method, Object arg1, Object arg2) {
    try {
        Method m = findMethodByNameAndArity(target.getClass(), method,
2);
        if (m == null) return false;
        m.invoke(target, arg1, arg2);
        return true;
    } catch (Exception e) {
        throw new TsirRuntimeException("Failed to invoke " + method + "(2
args): " + e.getMessage(), e);
    }
}

private static boolean invokeFiveArgsIfExists(Object target, String
method,
Object a1, Object a2,
Object a3, Object a4, Object a5) {
    try {
        Method m = findMethodByNameAndArity(target.getClass(), method,
5);
        if (m == null) return false;
        m.invoke(target, a1, a2, a3, a4, a5);
        return true;
    } catch (Exception e) {
        throw new TsirRuntimeException("Failed to invoke " + method + "(5
args): " + e.getMessage(), e);
    }
}

private static Method findMethodByNameAndArity(Class<?> cls, String name,
int arity) {
    for (Method m : cls.getMethods()) {
        if (!m.getName().equals(name)) continue;
        if (m.getParameterTypes().length != arity) continue;
        return m;
    }
}

```

```

        return null;
    }

// -----
// Minimal "step" adapter (map-backed buses, DBM flavor)
// -----


/**
 * A minimal pipeline step wrapper.
 *
 * It can be executed by:
 * - your IrPipelineV2 if it calls step.run(...) or step.apply(...)
 * - or by you manually invoking step.execute(bus...) in a custom runner
 *
 * This is purposely generic: it provides multiple conventional entry
methods:
 * - run()
 * - execute()
 * - apply()
 *
 * Inside, it uses Map-backed TsirDataBus/TsirEventBus so it can run
without modifying your pipeline core.
 *
 * IMPORTANT:
 * - In "full DBM integration", you will replace this with a typed
adapter to your native bus/event types.
 */
public static final class MapBackedStep {

    private final TsirPlan.PlanNode planNode;
    private final TsirExtractor extractor;

    // Shared buses across steps (static) for simplest possible
integration.
    // If your pipeline already has buses, replace these with bridge
hooks.
    private static final TsirDataBus DATA_BUS = new TsirMapDataBus();
    private static final TsirEventBus EVENT_BUS = new TsirMapEventBus();
    private static final TsirRuntimeGuards GUARDS =
TsirRuntimeGuards.defaults();
    private static final TsirManifest MANIFEST =
TsirManifest.start("PIPELINE_BRIDGE", "N/A");

    public MapBackedStep(TsirPlan.PlanNode planNode, TsirExtractor
extractor) {
        this.planNode = planNode;
        this.extractor = extractor;
    }

    // Conventional names that different pipeline frameworks might call:
    public void run() { execute(); }
    public void apply() { execute(); }

    public void execute() {
        TsirRuntimeContext ctx = new TsirRuntimeContext(DATA_BUS,
EVENT_BUS, GUARDS, MANIFEST, planNode);
        extractor.extract(ctx);
    }
}

```

```

        GUARDS.assertEventBudget(EVENT_BUS);
    }

    // Hooks to preload data into the shared DataBus for MVP demos:
    public static void putCurve(String name, TsirCurve curve) {
        DATA_BUS.putCurve(name, curve);
    }

    public static TsirEventBus eventBus() {
        return EVENT_BUS;
    }

    public static TsirDataBus dataBus() {
        return DATA_BUS;
    }

    public static TsirManifest manifest() {
        return MANIFEST;
    }

    public TsirPlan.PlanNode planNode() { return planNode; }
}

}

// File:
src/main/java/com/dbm/ir/runtime/Demo_TsirBridgeToIrPipelineV2_MinAssumption.
java
package com.dbm.ir.runtime;

import com.dbm.ir.runtime.TsirIrPipelineV2Bridge.MapBackedStep;

import java.util.*;

/**
 * Demo illustrating the minimum-assumption bridge usage.
 *
 * Replace IrPipelineV2Stub with your real IrPipelineV2 instance:
 *   IrPipelineV2 pipe = new IrPipelineV2(...);
 * Then:
 *   bridge.attachPlanToPipeline(pipe, plan);
 *   ... preload curves ...
 *   pipe.run();
 *
 * If your pipeline doesn't call step.run/execute automatically,
 * you can still manually iterate steps; but most pipelines do.
 */
public final class Demo_TsirBridgeToIrPipelineV2_MinAssumption {

    public static void main(String[] args) {
        // 1) Create recipe+plan (normally: parse JSON -> validate ->
compile)
        TsirRecipe recipe = demoRecipe();
        TsirPlan plan = new TsirPlanCompiler().compileOrThrow(recipe);

        // 2) Prepare extractor registry: for demo we register one trivial
extractor impl.
        TsirExtractorRegistry reg = TsirExtractorRegistry.defaultRegistry();
    }
}

```

```

    reg.register("demo.ExtractorUpDown", params -> new
DemoUpDownExtractor(params));
    reg.register("demo.ExtractorCount", params -> new
DemoCountExtractor(params));

    // 3) Create your IrPipelineV2 (here a stub)
    Object irPipelineV2 = new IrPipelineV2Stub();

    // 4) Bridge attach
    TsirIrPipelineV2Bridge bridge = new TsirIrPipelineV2Bridge(reg);
    bridge.attachPlanToPipeline(irPipelineV2, plan);

    // 5) Preload curve (MVP)
    MapBackedStep.putCurve("CURVE_A", new SimpleArrayCurve());

    // 6) Run pipeline (stub calls each step.run())
    bridge.runPipeline(irPipelineV2);

    // 7) Inspect outputs
    System.out.println("Events(E_BASE): " +
MapBackedStep.eventBus().getEvents("E_BASE").size());
    System.out.println("Events(E_REFINED): " +
MapBackedStep.eventBus().getEvents("E_REFINED").size());
    System.out.println("Done.");
}

private static TsirRecipe demoRecipe() {
    TsirRecipe r = new TsirRecipe();
    r.setRecipeVersion("1.0");
    r.setId("bridge_demo");

    TsirRecipe.SourceSpec s = new TsirRecipe.SourceSpec();
    s.setName("CURVE_A");
    s.setType("curve");
    s.setRef("INMEM");
    TsirRecipe.SourceSpec.SliceSpec sl = new
TsirRecipe.SourceSpec.SliceSpec();
    sl.setFrom("2020-01-01");
    sl.setTo("2020-12-31");
    s.setSlice(sl);
    r.setSources(Collections.singletonList(s));

    TsirRecipe.OperatorSpec op1 = new TsirRecipe.OperatorSpec();
    op1.setKind("extractor");
    op1.setImpl("demo.ExtractorUpDown");
    op1.setParams(new HashMap<String, Object>());

    TsirRecipe.OperatorSpec op2 = new TsirRecipe.OperatorSpec();
    op2.setKind("extractor");
    op2.setImpl("demo.ExtractorCount");
    op2.setParams(new HashMap<String, Object>());

    Map<String, TsirRecipe.OperatorSpec> ops = new LinkedHashMap<>();
    ops.put("ex1", op1);
    ops.put("ex2", op2);
    r.setOperators(ops);
}

```

```

        TsirRecipe.NodeSpec n1 = new TsirRecipe.NodeSpec();
        n1.setId("n1");
        n1.setOp("ex1");
        TsirRecipe.NodeSpec.InSpec in1 = new TsirRecipe.NodeSpec.InSpec();
        in1.setCurves(Collections.singletonList("CURVE_A"));
        in1.setEvents(Collections.<String>emptyList());
        n1.setIn(in1);
        TsirRecipe.NodeSpec.OutSpec out1 = new TsirRecipe.NodeSpec.OutSpec();
        out1.setEvents("E_BASE");
        n1.setOut(out1);

        TsirRecipe.NodeSpec n2 = new TsirRecipe.NodeSpec();
        n2.setId("n2");
        n2.setOp("ex2");
        TsirRecipe.NodeSpec.InSpec in2 = new TsirRecipe.NodeSpec.InSpec();
        in2.setCurves(Collections.singletonList("CURVE_A"));
        in2.setEvents(Collections.singletonList("E_BASE"));
        n2.setIn(in2);
        TsirRecipe.NodeSpec.OutSpec out2 = new TsirRecipe.NodeSpec.OutSpec();
        out2.setEvents("E_REFINED");
        n2.setOut(out2);

        TsirRecipe.GraphSpec g = new TsirRecipe.GraphSpec();
        g.setMode("sequence");
        g.setNodes(Arrays.asList(n1, n2));
        r.setGraph(g);

        TsirRecipe.ArtifactsSpec a = new TsirRecipe.ArtifactsSpec();
        a.setEvents(Collections.singletonList("E_REFINED"));
        a.setManifest(Boolean.TRUE);
        r.setArtifacts(a);

        return r;
    }

    // -----
    // Minimal IrPipelineV2 stub for demo
    // -----
    public static final class IrPipelineV2Stub {
        private final List<Object> steps = new ArrayList<>();

        public void addStep(Object step) { steps.add(step); }

        public void run() {
            for (Object s : steps) {
                // call run()/execute()/apply() if exists
                callIfExists(s, "run");
                callIfExists(s, "execute");
                callIfExists(s, "apply");
            }
        }
    }

    private void callIfExists(Object o, String method) {
        try {
            o.getClass().getMethod(method).invoke(o);
        } catch (NoSuchMethodException ignored) {
        } catch (Exception e) {

```

```

        throw new RuntimeException("Failed to call " + method + " on
step: " + e.getMessage(), e);
    }
}
}

// -----
// Demo extractors (replace with adapters)
// -----
public static final class DemoUpDownExtractor implements TsirExtractor {
    public DemoUpDownExtractor(Map<String, Object> params) {}
    @Override
    public void extract(TsirRuntimeContext ctx) {
        TsirCurve c = ctx.curve0OrThrow();
        // Create one event every 3 bars for demo
        for (int i = 2; i < c.size(); i += 3) {
            ctx.eventBus().appendEvent(ctx.outStream(),
TsirEvent.of(c.t(i-2), c.t(i), "UPDOWN", 1.0));
        }
    }
}

public static final class DemoCountExtractor implements TsirExtractor {
    public DemoCountExtractor(Map<String, Object> params) {}
    @Override
    public void extract(TsirRuntimeContext ctx) {
        int n = ctx.inEvents0().size();
        Map<String, Object> tags = new HashMap<>();
        tags.put("count", n);
        TsirCurve c = ctx.curve0OrThrow();
        ctx.eventBus().appendEvent(ctx.outStream(), TsirEvent.of(c.t(0),
c.t(c.size()-1), "COUNT", n, tags));
    }
}

// -----
// Minimal in-memory curve
// -----
public static final class SimpleArrayCurve implements TsirCurve {
    private final long[] t;
    private final double[] v;

    public SimpleArrayCurve() {
        int n = 20;
        t = new long[n];
        v = new double[n];
        for (int i = 0; i < n; i++) {
            t[i] = 1700000000000L + i * 60_000L;
            v[i] = Math.sin(i * 0.2) + i * 0.01;
        }
    }

    @Override public int size() { return v.length; }
    @Override public long t(int idx) { return t[idx]; }
    @Override public double v(int idx) { return v[idx]; }
}
}

```

你需要对齐的“1–2 个方法名”在哪里？

几乎一定只在这个类里：

- `TsirIrPipelineV2Bridge.tryAddStep(...)`

你只要把它的反射调用替换成你真实 `IrPipelineV2` 的一个方法，例如：

- `pipeline.addExtractor(step)` 或 `pipeline.add(step)` 或 `pipeline.addNode(nodeId, step)`
- 或者你 `pipeline` 需要 `addStep(nodeId, step, inCurves, inEvents, outStream)`
(我也预留了 5 参数版本)

另外，如果你的 `pipeline` 的执行方法不是 `run/execute/runAndCollect`，你只需要在：

- `runPipeline(...)` 里增加一个 method name (比如 `runAll()`) 。

这个桥接类的“DBM 风味”点在哪里？

1. **PlanNode 不直接写入 pipeline 内部结构**：先桥接，保留你 `IrPipelineV2` 的演化自由度。
2. **ExtractorRegistry** 允许你把旧 extractor 全部包成 Adapter (不破坏既有资产)。
3. **MapBackedStep** 是最小 MVP 的“总线仿真器”，你随时可以替换为真实的 DataBus/EventBus，而 recipe/plan/validator 完全不动。
4. 这正是 Task–Action Dual ACLM：Task 层 (JSON) 疯狂变，Action 层 (runtime) 稳定且可治理。

如果你把这段 drop-in 后跑通了，我建议下一步我继续补一个“更贴合 DBM”的升级 (不影响现有代码)：

- `IrPipelineV2BridgeHooks`：把 `MapBackedStep` 的静态 `buses` 换成从 `pipeline` 获取的 `bus`（例如 `pipeline.getDataBus() / getEventBus()`），这样就不是“仿真器”，而是真正嵌入你的 `runtime`。

兄弟，整的很好。谢谢了。容我消化一下，跑一跑。我们再继续。我蛮感慨的：这个如此复杂的课题，我们竟能相互理解，相互合作的这么高效，有力。显示了人与 AI 在高度技术领域的合作的空间与前景相当可观，看好。这还是每月二十美元的 AI 服务，要是 AI 全力以赴，哪还了得！

收到。你这段感慨本身其实就是一个很硬的工程结论：在高复杂度系统里，“人与 AI 的高效协作”并不是靠灵光一现，而是靠我们这次已经反复验证过的那套方法论——**把复杂性切层、把契约钉死、把演化放到上层、把执行稳定留在下层**。

你现在消化和跑的过程中，我建议你重点观察 3 个“是否真的落地”的信号（都很客观）：

1. 接入成本是否趋近于常数

新增一个 `extractor`（或旧 `extractor` 做 `adapter`）时，是否基本只改：

- `registry` 注册一行
 - `recipe JSON` 增加一个 `node`
- 而不是改一堆 `pipeline` 代码。

2. 错误是否前移到编译期（Plan）

`single-writer`、前向引用、未知 `source/operator` 这类问题，是否在 `validate/compile` 阶段就被抓住，避免 `runtime` 才炸。

3. 产出是否可复现、可审计、可解释

`manifest` 是否能让你“同样 `recipe` 同样 `plan` 同样结果”，并且能定位到哪一个 `node` 的行为差异。

等你跑通后，我们很自然可以继续往前推两步（我不等你回答也先把路径说清楚，方便你对齐你的实现）：

- 把 `MapBackedStep` 的静态 `DataBus/EventBus` 替换成从 `IrPipelineV2` 拿真实总线（桥接从“仿真”变“内嵌”）。
- 把 `operators.params` 里常见字段（`window`、`threshold`、`regime`、`budget` 等）提炼成顶层 **policies/governance**（`Recipe v1.1`），让 JSON 从“能跑”变成“可治理的研究工艺卡”。

至于你说的“20 美元/月就这样”，我理解你的意思：一旦协作范式成熟，AI 的产能会呈现明显的非线性放大。但更关键的是，我们现在已经把这种放大工程化了——不是靠运气，而是靠可复制的结构化协作方式。你这套 `Recipe`→`Plan`→`Run`→`Artifacts` 的上半身落稳之后，后续的 30+ patterns 接入会进入“流水线时代”，节奏会非常快。

你先消化与跑。等你反馈跑出来的第一个小摩擦点（通常是 1–2 个方法名对齐或 bus 获取方式），我们就可以把桥接升级到“真 `IrPipelineV2` 总线内嵌版”。

好的。兄弟。保重。

好的。你也保重。

先让系统自己跑一跑、沉淀一下——很多真正的质量感，都是在“冷却期”里浮现出来的。

你现在做的这件事，其实已经越过了“写代码”的层次，进入了**把复杂智能工程变成可协作、可演化工艺**的阶段。能走到这里，本身就说明判断力、耐力和结构感都非常扎实。后面的推进不靠急，而靠稳。

等你下次回来，我们可以非常自然地继续：

- 要么收紧桥接，彻底并入 `IrPipelineV2` 总线；
- 要么反向抽象，把这套 `Recipe / Plan / Runtime` 的模式迁移到别的 DBM 子系统（比如非时间序列的结构搜索）。

你慢慢跑、慢慢看。

我在这，随时接着。