Nion Orchestration Internship Assessment

Problem Statement

You are building a simplified version of Nion's orchestration engine. Nion is an **AI Program Manager agent** that helps teams track action items, risks, issues, and decisions across projects. It observes communications (emails, Slack, meetings), extracts relevant items, and responds to questions about project status.

Given an input message, your program should analyze it and print the complete orchestration map showing how the message flows through the L1 → L2 → L3 hierarchy.

Background

Nion uses a three-tier architecture:

Layer	Role	Description
L1	Orchestrator	Ingests and parses the message, reasons about intent, identifies gaps, selects strategy, generates plan
L2	Coordinator	Receives directions from L1, coordinates multiple L3 agents, aggregates results
L3	Agent	Executes specific tasks, returns results

Visibility Rules

Each layer has restricted visibility:

Layer	Can See
L1	L2 domains + Cross-Cutting agents only
L2: TRACKING_EXECUTION	Its own L3 agents + Cross-Cutting agents
L2: COMMUNICATION_COLLABORATION	Its own L3 agents + Cross-Cutting agents
L2: LEARNING_IMPROVEMENT	Its own L3 agents + Cross-Cutting agents

L1 does **not** see individual L3 agents directly. It delegates to L2, which coordinates the appropriate L3 agents.

Available Components

L2 Domain Agents (Visible to L1)

Domain	Purpose
`TRACKING_EXECUTION`	Extraction, validation, and tracking of items
`COMMUNICATION_COLLABORATION`	Q&A, reporting, delivery
`LEARNING_IMPROVEMENT`	Learning from instructions

Cross-Cutting Agents (Visible to L1 and all L2 domains)

Cross Cutting Agent	Purpose
`knowledge_retrieval`	Retrieves context from database - project info, stakeholder details, historical data
`evaluation`	Validates outputs before delivery - checks accuracy, tone, completeness

L3 Agents by L2 Domain

L2: TRACKING_EXECUTION (only visible to this L2)

L3 Agent	Purpose
`action_item_extraction`	Extracts action items from message content, infers owners and due dates
`action_item_validation`	Validates action items have required fields, flags missing info
`action_item_tracking`	Tracks action items to completion, provides status snapshots
`risk_extraction`	Extracts risks from message content, assesses likelihood and impact
`risk_tracking`	Tracks risks, provides risk snapshots

`issue_extraction`	Extracts issues/problems from message content, assesses severity
`issue_tracking`	Tracks issues to resolution, provides issue snapshots
`decision_extraction`	Extracts decisions from message content, identifies decision maker
`decision_tracking`	Tracks decisions to implementation

L2: COMMUNICATION_COLLABORATION (only visible to this L2)

L3 Agent	Purpose
`qna`	Formulates responses to questions, handles both direct answers and gap-aware responses
`report_generation`	Creates formatted reports (status reports, summaries, digests)
`message_delivery`	Sends messages via appropriate channels (email, Slack, Teams)
`meeting_attendance`	Captures meeting transcripts, generates meeting minutes

L2: LEARNING_IMPROVEMENT (only visible to this L2)

L3 Agent	Purpose
`instruction_led_learning`	Learns from explicit instructions, stores SOPs and rules

---

Your Task

Write a program that:

1. Accepts an input message (JSON)
2. Performs L1 reasoning (ingestion, intent analysis, planning) and delegates to L2 agent
3. Prints the full orchestration map showing L1 plan followed by L2/L3 execution
4. Respects visibility rules (L1 only delegates to L2/Cross-Cutting, L2 coordinates its own L3 agents)

Input Format

```
{
  "message_id": "MSG-001",
  "source": "email",
  "sender": {
    "name": "Sarah Chen",
```

```
    "role": "Product Manager"
  },
  "content": "The customer demo went great! They loved it but asked if we could add real-time notifications and a dashboard export feature. They're willing to pay 20% more and need it in the same timeline. Can we make this work?",
  "project": "PRJ-ALPHA"
}

```

Expected Output Format

```
================================================================================
NION ORCHESTRATION MAP
================================================================================
Message: <message_id>
From: <sender_name> (<sender_role>)
Project: <project>

================================================================================
L1 PLAN
================================================================================
[TASK-XXX] → L2:<DOMAIN> or L3:<cross-cutting-agent>
  Purpose: <description>
  Depends On: <optional dependencies>

<... repeat for all planned tasks ...>

================================================================================
L2/L3 EXECUTION
================================================================================

[TASK-XXX] L2:<DOMAIN>
  └→ [TASK-XXX-A] L3:<agent>
      Status: COMPLETED
      Output:
        • <output line 1>
        • <output line 2>

[TASK-XXX] L3:<cross-cutting-agent> (Cross-Cutting)
  Status: COMPLETED
  Output:
    • <output line 1>
    • <output line 2>

<... repeat for all executed tasks ...>

================================================================================

```

Sample Input/Output

Input:

```
{
  "message_id": "MSG-001",
  "source": "email",
  "sender": {
    "name": "Sarah Chen",
    "role": "Product Manager"
```

```
  },
  "content": "The customer demo went great! They loved it but asked if we could add real-time notifications and a dashboard export feature. They're willing to pay 20% more and need it in the same timeline. Can we make this work?",
  "project": "PRJ-ALPHA"
}

```

Output:

(Note: L3 entries showing responses are just dummy placeholders and can be generated randomly)

```
================================================================================
NION ORCHESTRATION MAP
================================================================================
Message: MSG-001
From: Sarah Chen (Product Manager)
Project: PRJ-ALPHA

================================================================================
L1 PLAN
================================================================================
[TASK-001] → L2:TRACKING_EXECUTION
  Purpose: Extract action items from customer request

[TASK-002] → L2:TRACKING_EXECUTION
  Purpose: Extract risks from scope change request

[TASK-003] → L2:TRACKING_EXECUTION
  Purpose: Extract decision needed

[TASK-004] → L3:knowledge_retrieval (Cross-Cutting)
  Purpose: Retrieve project context and timeline

[TASK-005] → L2:COMMUNICATION_COLLABORATION
  Purpose: Formulate gap-aware response
  Depends On: TASK-001, TASK-002, TASK-003, TASK-004

[TASK-006] → L3:evaluation (Cross-Cutting)
  Purpose: Evaluate response before sending
  Depends On: TASK-005

[TASK-007] → L2:COMMUNICATION_COLLABORATION
  Purpose: Send response to sender
  Depends On: TASK-006

================================================================================
L2/L3 EXECUTION
================================================================================

[TASK-001] L2:TRACKING_EXECUTION
  └→ [TASK-001-A] L3:action_item_extraction
      Status: COMPLETED
      Output:
        • AI-001: "Evaluate real-time notifications feature"
          Owner: ? | Due: ? | Flags: [MISSING_OWNER, MISSING_DUE_DATE]
        • AI-002: "Evaluate dashboard export feature"
          Owner: ? | Due: ? | Flags: [MISSING_OWNER, MISSING_DUE_DATE]

[TASK-002] L2:TRACKING_EXECUTION
  └→ [TASK-002-A] L3:risk_extraction
```

```
      Status: COMPLETED
      Output:
        • RISK-001: "Adding 2 features with same timeline"
          Likelihood: HIGH | Impact: HIGH
        • RISK-002: "Scope creep for 20% revenue increase"
          Likelihood: MEDIUM | Impact: MEDIUM

[TASK-003] L2:TRACKING_EXECUTION
  └─► [TASK-003-A] L3:decision_extraction
      Status: COMPLETED
      Output:
        • DEC-001: "Accept or reject customer feature request"
          Decision Maker: ? | Status: PENDING

[TASK-004] L3:knowledge_retrieval (Cross-Cutting)
  Status: COMPLETED
  Output:
    • Project: PRJ-ALPHA
    • Current Release Date: Dec 15
    • Days Remaining: 20
    • Code Freeze: Dec 10
    • Current Progress: 70%
    • Team Capacity: 85% utilized
    • Engineering Manager: Alex Kim
    • Tech Lead: David Park

[TASK-005] L2:COMMUNICATION_COLLABORATION
  └─► [TASK-005-A] L3:qna
      Status: COMPLETED
      Output:
        • Response: "Great news on the demo! For the feature request:

          WHAT I KNOW:
          • Current timeline: Dec 15 (code freeze Dec 10)
          • Team capacity: 85% utilized
          • Progress: 70% complete

          WHAT I'VE LOGGED:
          • 2 action items for feature evaluation
          • 2 risks flagged (timeline + scope creep)
          • 1 pending decision

          WHAT I NEED:
          • Complexity estimates from Engineering (Alex Kim / David
Park)
          • Go/no-go decision from leadership

          I cannot assess feasibility without Engineering input on
          whether 2 new features can fit in 20 days at 85% capacity."

[TASK-006] L3:evaluation (Cross-Cutting)
  Status: COMPLETED
  Output:
    • Relevance: PASS
    • Accuracy: PASS
    • Tone: PASS
    • Gaps Acknowledged: PASS
    • Result: APPROVED

[TASK-007] L2:COMMUNICATION_COLLABORATION
  └─► [TASK-007-A] L3:message_delivery
      Status: COMPLETED
      Output:
        • Channel: email
        • Recipient: Sarah Chen
        • CC: Alex Kim (Engineering Manager)
        • Delivery Status: SENT

================================================================
===========

```

Test Cases

Test Case 1: Simple Status Question

```
{
  "message_id": "MSG-101",
  "source": "slack",
  "sender": { "name": "John Doe", "role": "Engineering Manager" },
  "content": "What's the status of the authentication feature?",
  "project": "PRJ-BETA"
}
```

Test Case 2: Feasibility Question (New Features)

```
{
  "message_id": "MSG-102",
  "source": "email",
  "sender": { "name": "Sarah Chen", "role": "Product Manager" },
  "content": "Can we add SSO integration before the December release?",
  "project": "PRJ-ALPHA"
}
```

Test Case 3: Decision/Recommendation Request

```
{
  "message_id": "MSG-103",
  "source": "email",
  "sender": { "name": "Mike Johnson", "role": "VP Engineering" },
  "content": "Should we prioritize security fixes or the new dashboard?",
  "project": "PRJ-GAMMA"
}
```

Test Case 4: Meeting Transcript

```
{
  "message_id": "MSG-104",
  "source": "meeting",
  "sender": { "name": "System", "role": "Meeting Bot" },
  "content": "Dev: I'm blocked on API integration, staging is down. QA: Found 3 critical bugs in payment flow. Designer: Mobile mockups ready by Thursday. Tech Lead: We might need to refactor the auth module.",
  "project": "PRJ-ALPHA"
}
```

Test Case 5: Urgent Escalation

```
{
  "message_id": "MSG-105",
  "source": "email",
  "sender": { "name": "Lisa Wong", "role": "Customer Success Manager" },
  "content": "The client is asking why feature X promised for Q3 is still not delivered. They're threatening to escalate to legal. What happened?",
  "project": "PRJ-DELTA"
}
```

Test Case 6: Ambiguous Request

```
{
  "message_id": "MSG-106",
  "source": "slack",
```

```
  "sender": { "name": "Random User", "role": "Unknown" },
  "content": "We need to speed things up",
  "project": null
}

```

Tools & Submission

You may use any AI coding tools to help you build this (Cursor, Claude Code, GitHub Copilot, ChatGPT, etc.) and/or agentic framework (langchain / crewai)

Submit:

1. Source code
2. README with setup instructions
3. Sample output for each test case

Good luck!