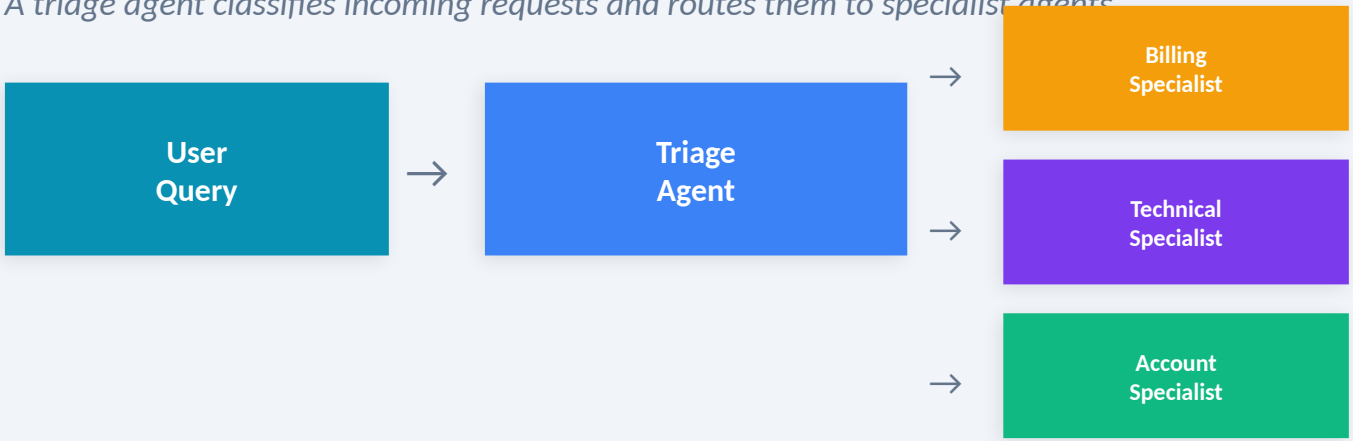
05

Advanced Patterns

Handoff · Magentic · Production Considerations

Handoff: Support Triage

A triage agent classifies incoming requests and routes them to specialist agents



When to Use

- Different input types need different expertise
- Keep specialists focused with lean prompts
- Routing logic = classification problem

Context: Structured HandoffContext

Structured handoff object (HandoffContext dataclass) passes context from triage to specialist. Contains: query, category, priority, extracted_info. Specialist does NOT see triage reasoning.

Handoff — Decision Guide

✓ When to Use

Different agents handle different workflow stages
Each stage requires specialized expertise
Responsibility needs to be clearly transferred
Next agent starts only when previous one finishes

★ Why Choose It

Clean ownership boundaries between agents
Ideal for workflows with tiered expertise
Each stage handled by the right specialist
Reduces back-and-forth and rework

✗ When NOT to Use

Agent order is already known upfront (use Sequential)
One agent can complete the entire task
Current agent lacks context to perform handoff
You need parallel processing (use Concurrent)

➤ Example

Adaptive Product Issue Escalation

Feedback → Hardware Diagnostics → Software Telemetry → Risk → Root Cause

Exercise: Support Triage Code

`exercises/07_handoff/01_support_triage.py`

Triage: parse() for Classification

```
class TriageDecision(BaseModel):
    category: str  # billing/technical/account
    priority: str  # low/medium/high
    reasoning: str
    extracted_info: dict  # order IDs, etc.

decision = client.chat.completions.parse(
    model=model,
    messages=[
```

Structured Handoff + Routing

```
@dataclass
class HandoffContext:
    customer_query: str
    category: str
    priority: str
    extracted_info: dict

# Specialist gets ONLY handoff context
handoff_text = (
    f"Query: {handoff.customer_query}\n"
```

Each Specialist Has Focused Tools

Billing	`lookup_order()`, `process_refund()`
Technical	`search_faq()`
Account	`search_faq()`

Magentic — Decision Guide

✓ When to Use

Agents activate based on findings, not a fixed order
Workflow is adaptive — next step depends on discoveries
Tasks have conditional dependencies
Problem needs both structure and flexibility

★ Why Choose It

Context-aware flow — only necessary agents activate
Reduces wasted work (no unnecessary reviews)
Deeper reasoning without forcing a rigid path
Great balance between efficiency and thoroughness

✗ When NOT to Use

Solution path is deterministic and known upfront
Tasks are simple, don't need multi-round reasoning
Speed is critical; can't afford planning overhead
Token budget is tight (manager + workers = many calls)

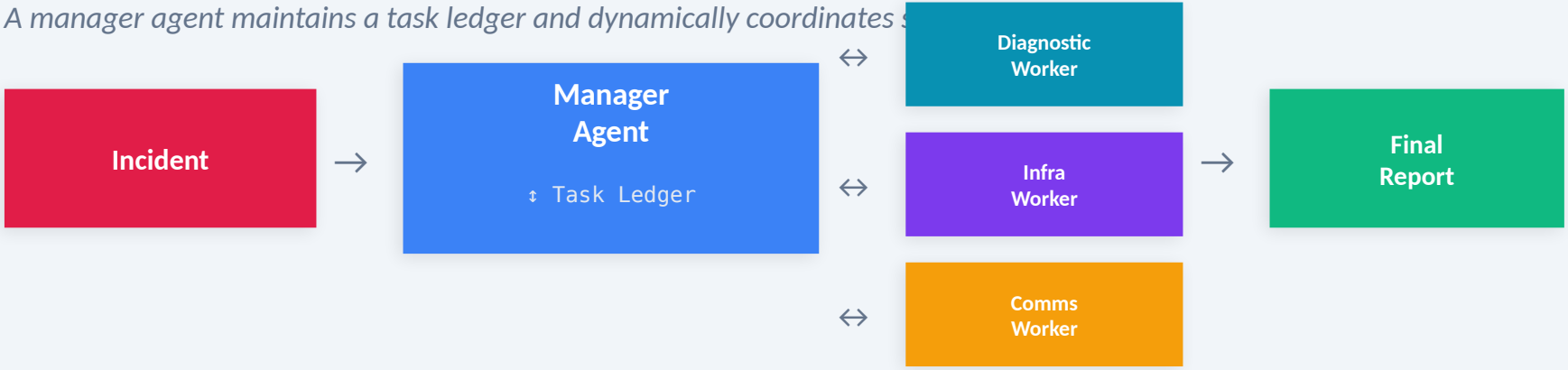
➤ Example

RFP Reviewer

Summary → Risk Assessment → Compliance (only if risk found) → Decision

Magentic: Incident Response

A manager agent maintains a task ledger and dynamically coordinates s



When to Use

- Complex multi-step tasks with plan changes
- Intermediate findings may add new tasks
- Central coordinator tracks progress

Context: Task Ledger + Task-Specific

Workers get TASK-SPECIFIC context only — incident description + their specific task. The manager controls information flow and maintains the Task Ledger (shared mutable dataclass). Plan can adapt: new tasks added based on findings.

Exercise: Incident Response — 4 Phases

`exercises/08_magentic/01_incident_response.py`

Phase 1: Manager Creates Plan

```
class IncidentPlan(BaseModel):
    assessment: str
    tasks: list[PlannedTask]
```

Phase 2: Workers Execute Tasks

```
for task in ledger.tasks:
    # Worker gets ONLY task context
    messages = [
      {"role": "system", "content": PROMPT},
```

Phase 3: Manager Adapts Plan

```
class AdaptedPlan(BaseModel):
    analysis: str
    new_tasks: list[PlannedTask]
    ready_for_report: bool
```

Phase 4: Final Incident Report

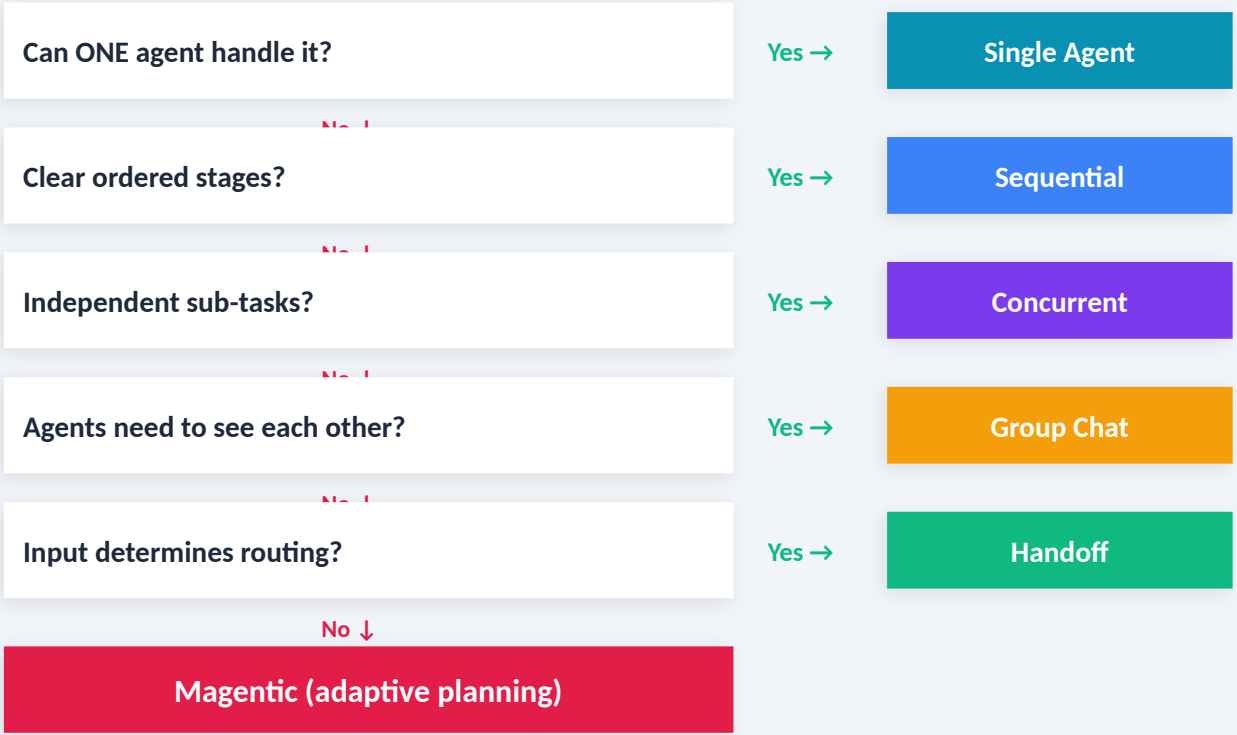
```
# Manager synthesizes ALL findings
all_findings = "\n".join(
    f"Task #{t.id} ({t.assigned_to}): "
    f"{t.result}"
```

Task Ledger = shared mutable dataclass with tasks[], findings[], add_task(), complete_task(). The central coordination artifact.

Choosing the Right Pattern — At a Glance

Pattern	Decision Style	Use When	When NOT To Use
Sequential	Fixed, staged	Steps depend on previous one	Steps aren't ordered
Concurrent	Parallel, independent	Tasks run in parallel	Tasks depend on each other
Group Chat	Collaborative dialogue	Agents reason together	Need strict step-by-step
Handoff	Expertise-based routing	Need specialized expertise	Flow is fixed upfront
Magentic	Adaptive, goal-driven	Plan evolves with outcomes	Path is deterministic
Single Agent	Direct	One prompt covers it	Need multiple perspectives

Choosing the Right Pattern



Pro Tip

Start with Single Agent.

Before choosing a multi-agent pattern, ask: can I solve this with one agent that has good tools and a clear system prompt?

If yes, stop there.

Context Management Strategies

Full History

Single Agent, Group Chat

Pass entire conversation list. Simple but can hit token limits.

Fresh per Stage

Sequential

Each agent gets clean context with only previous output. Token-efficient.

Structured Objects

Handoff

Pass a typed dataclass (HandoffContext) between agents. Explicit and robust.

Task-Specific

Magentic

Manager curates what each worker sees — only incident context + specific task.

Context strategy is as important as choosing the right pattern.

Handoff — Decision Guide

✓ When to Use

Different agents handle different workflow stages
Each stage requires specialized expertise
Responsibility needs to be clearly transferred
Next agent starts only when previous one finishes

★ Why Choose It

Clean ownership boundaries between agents
Ideal for workflows with tiered expertise
Each stage handled by the right specialist
Reduces back-and-forth and rework

✗ When NOT to Use

Agent order is already known upfront (use Sequential)
One agent can complete the entire task
Current agent lacks context to perform handoff
You need parallel processing (use Concurrent)

➤ Example

Adaptive Product Issue Escalation

Feedback → Hardware Diagnostics → Software Telemetry → Risk → Root Cause

Magentic — Decision Guide

✓ When to Use

Agents activate based on findings, not a fixed order
Workflow is adaptive — next step depends on discoveries
Tasks have conditional dependencies
Problem needs both structure and flexibility

★ Why Choose It

Context-aware flow — only necessary agents activate
Reduces wasted work (no unnecessary reviews)
Deeper reasoning without forcing a rigid path
Great balance between efficiency and thoroughness

✗ When NOT to Use

Solution path is deterministic and known upfront
Tasks are simple, don't need multi-round reasoning
Speed is critical; can't afford planning overhead
Token budget is tight (manager + workers = many calls)

➤ Example

RFP Reviewer

Summary → Risk Assessment → Compliance (only if risk found) → Decision

Choosing the Right Pattern — At a Glance

Pattern	Decision Style	Use When	When NOT To Use
Sequential	Fixed, staged	Steps depend on previous one	Steps aren't ordered
Concurrent	Parallel, independent	Tasks run in parallel	Tasks depend on each other
Group Chat	Collaborative dialogue	Agents reason together	Need strict step-by-step
Handoff	Expertise-based routing	Need specialized expertise	Flow is fixed upfront
Magentic	Adaptive, goal-driven	Plan evolves with outcomes	Path is deterministic
Single Agent	Direct	One prompt covers it	Need multiple perspectives