Лекция 12: AI Agents и Tools

От text generation към действия

Цели на лекцията

- От генериране към действия — защо agents?
- Архитектура на агенти — компоненти и loops
- ReAct — reasoning + acting
- Memory — как агентите помнят
- Planning и reflection — самокорекция
- Multi-agent системи
- Реални резултати и ограничения

Част 1: От Generation към Action

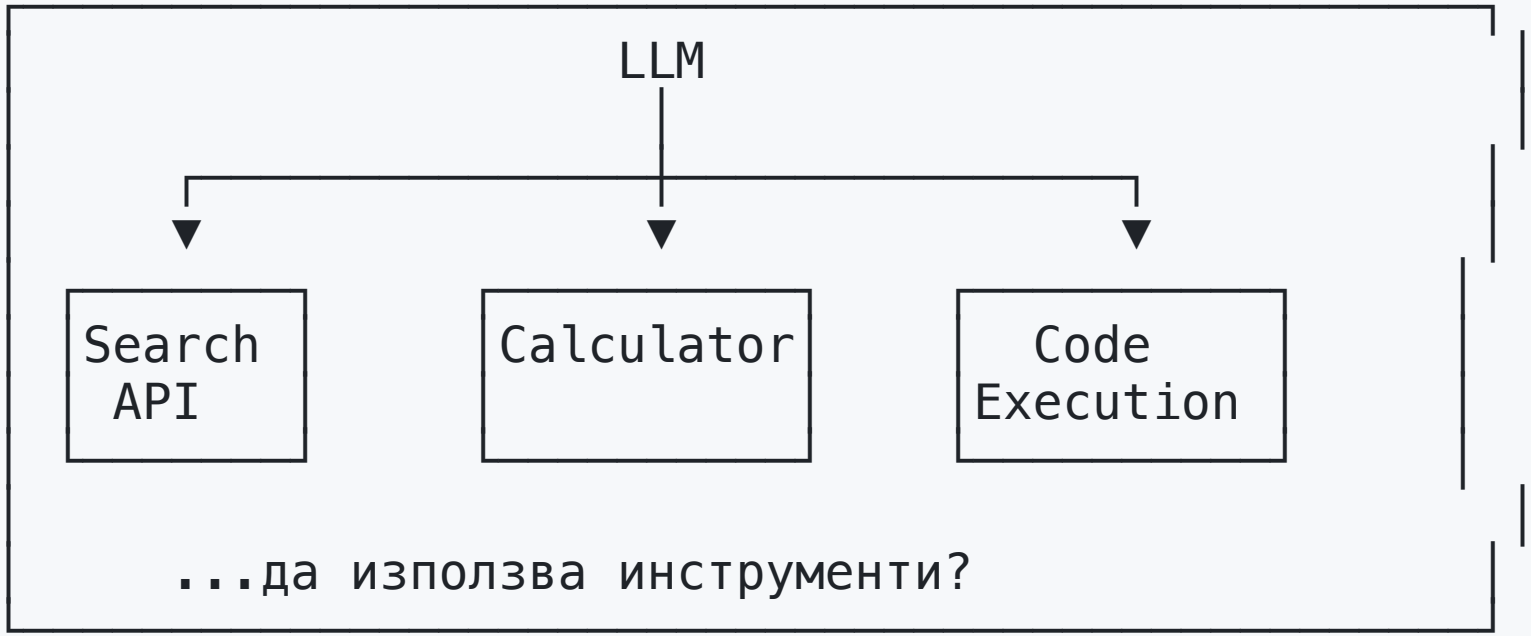
Ограничението на чистата генерация

```
User: Какво е времето в София днес?

LLM: Не мога да проверя текущото време,
     тъй като нямам достъп до интернет.
```

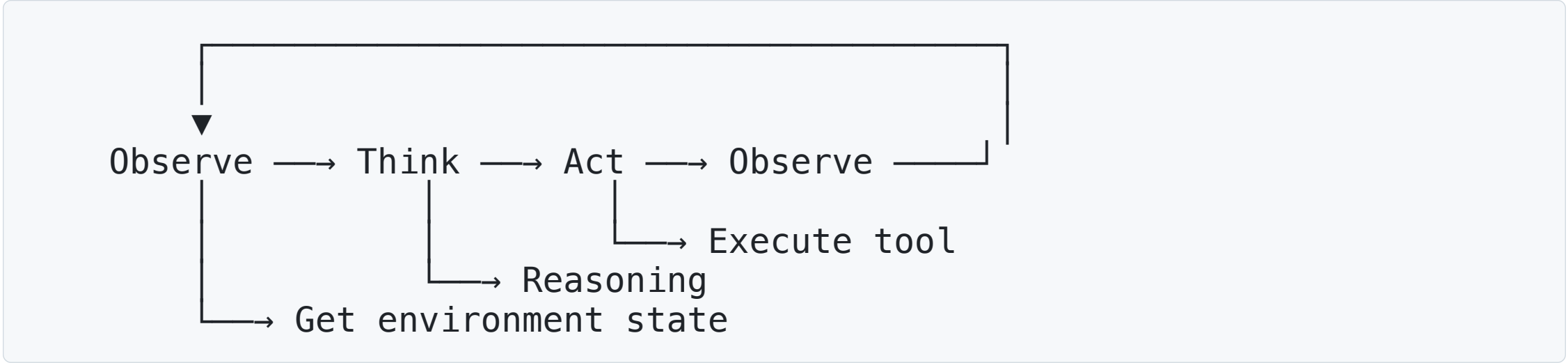
Моделът може само да генерира текст

- Няма достъп до интернет
- Няма достъп до файлове
- Няма възможност за изчисления
- Няма възможност за действия

Какво ако моделът можеше...

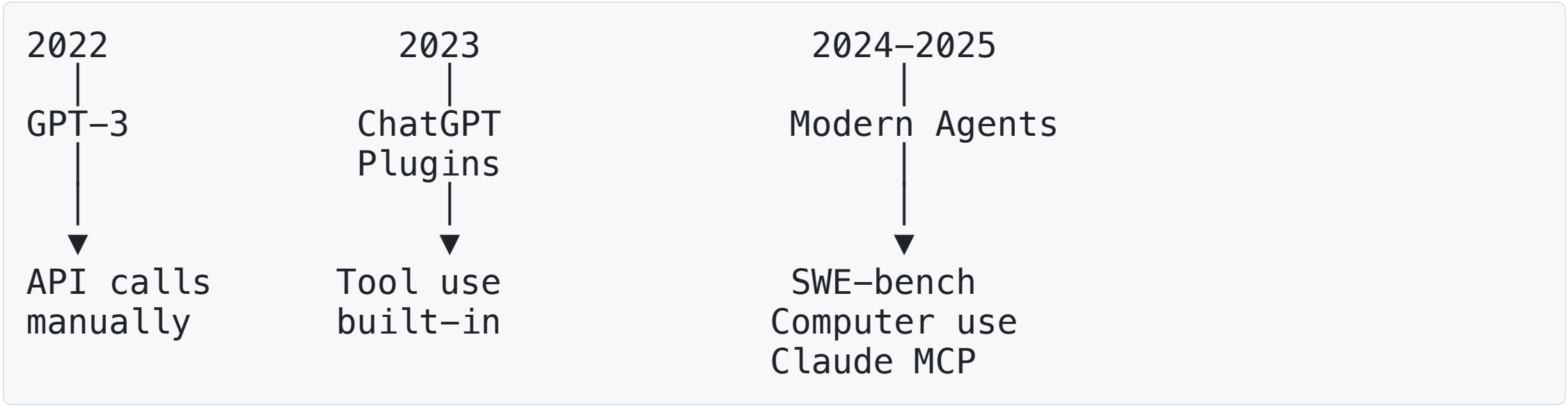


Agent Paradigm



Agent = LLM + Tools + Loop

Кратка история



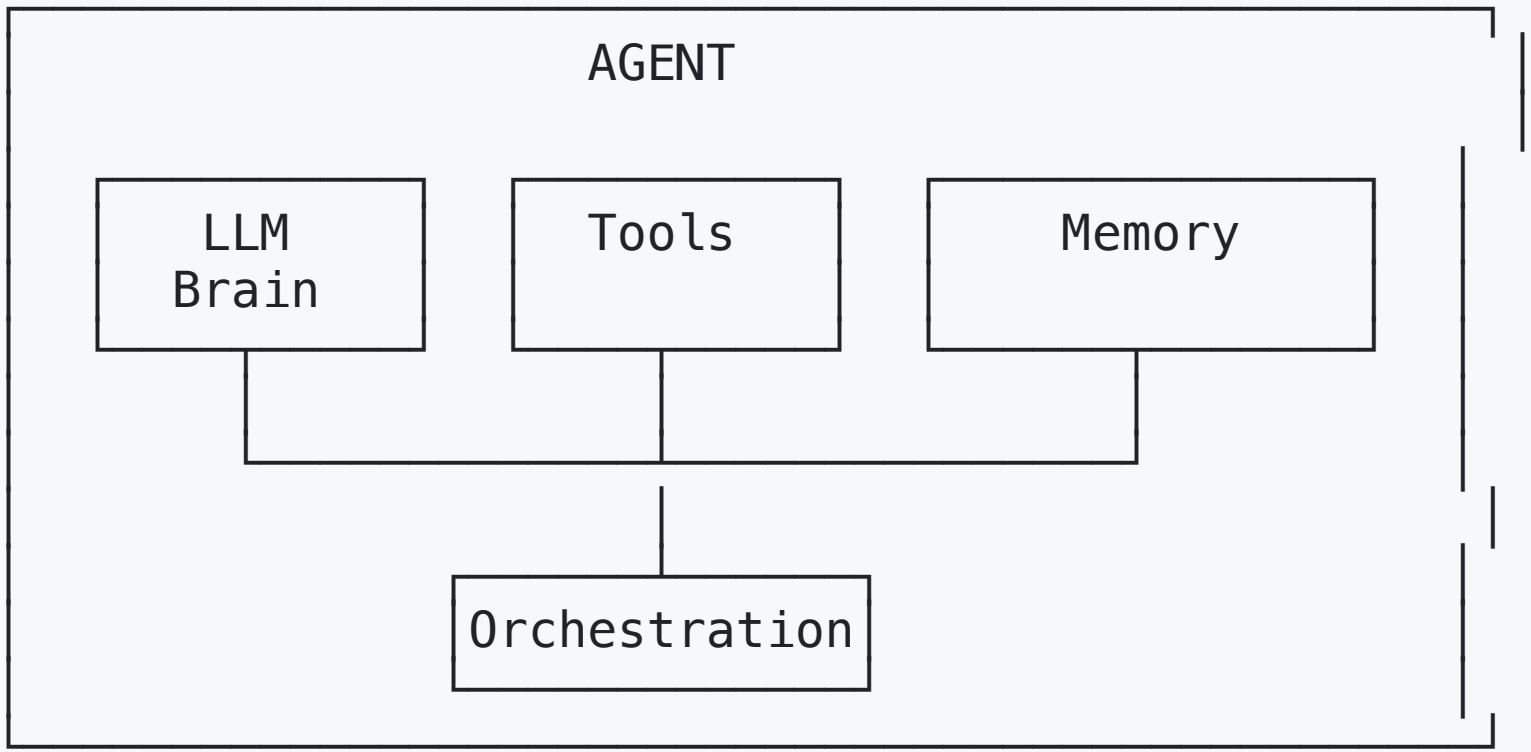
Защо 2023-2024 беше explозия на agents?

1. **По-способни модели** — следват сложни инструкции
2. **Function calling** — built-in API за tools
3. **По-дълъг context** — помнят повече
4. **По-добър reasoning** — правят по-малко грешки

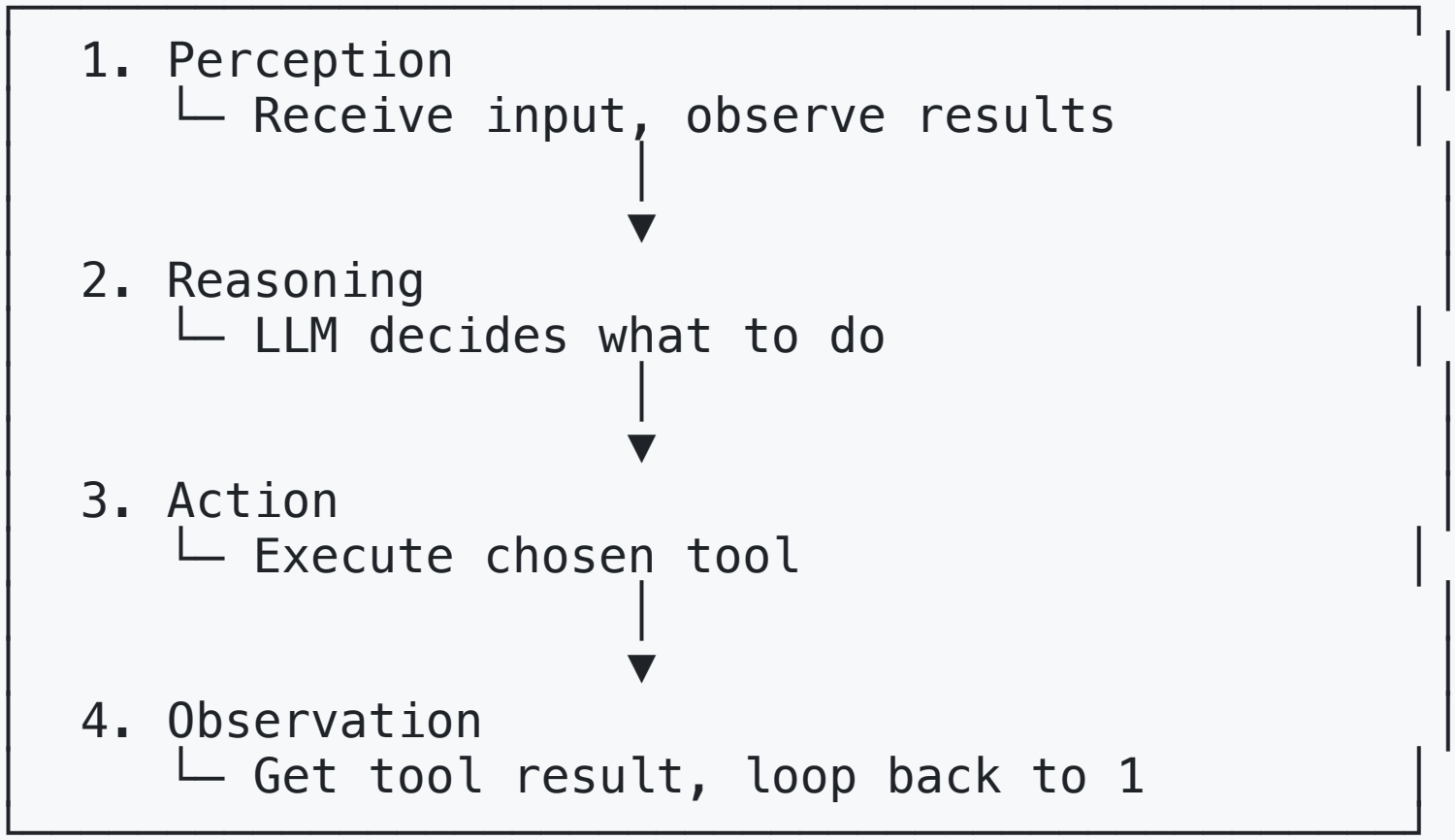
Но: Все още не са reliable в production

Част 2: Agent Architecture

Основни компоненти



Agent Loop


1. Perception
└ Receive input, observe results
2. Reasoning
└ LLM decides what to do
3. Action
└ Execute chosen tool
4. Observation
└ Get tool result, loop back to 1

Tool Definitions

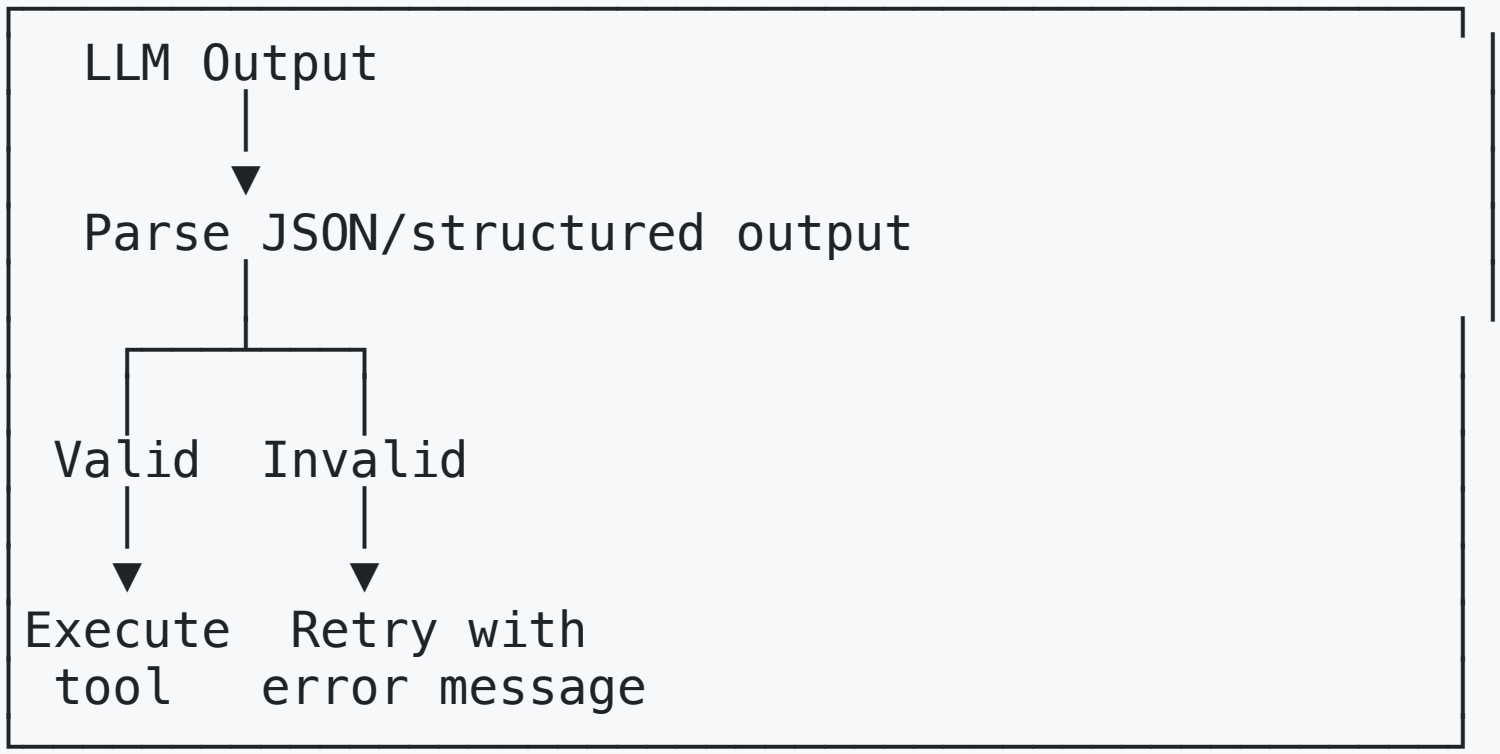
```
tools = [
    {
        "name": "search",
        "description": "Search the web for information",
        "parameters": {
            "query": {
                "type": "string",
                "description": "Search query"
            }
        }
    },
    {
        "name": "calculator",
        "description": "Perform math calculations",
        "parameters": {
            "expression": {"type": "string"}
        }
    }
]
```

Как моделът избира tool?

```
User: Колко е 15% от 847?

LLM reasoning:
- Трябва математическо изчисление
- Имам calculator tool
- Expression: "847 * 0.15"

LLM output:
{
    "tool": "calculator",
    "parameters": {"expression": "847 * 0.15"}
}
```

Моделът "разбира" кога кой tool е подходящ

Parsing и Error Handling



Пример: Simple Agent

```
User: Какво е населението на България и колко е 5% от него?

Agent:
  Thought: Нужна ми е информация за населението
  Action: search("население на България 2024")
  Observation: ~6.5 милиона души

  Thought: Сега да изчисля 5%
  Action: calculator("6500000 * 0.05")
  Observation: 325000

  Answer: България има ~6.5 милиона жители.
          5% от това са 325,000 души.
```

Част 3: ReAct Pattern

Какво е ReAct?

Reasoning + **Act**ing

```
Стандартен agent:
  User → Action → Observation → Action → Answer

ReAct agent:
  User → Thought → Action → Observation →
       → Thought → Action → Observation → Answer
```

Ключова разлика: Explicit reasoning преди всяко действие

ReAct структура

```
Question: Кой е режисьорът на филма, спечелил
          Оскар 2024 за най-добър филм?

Thought: Трябва да намеря кой филм спечели Оскар 2024
Action: search("Оскар 2024 най-добър филм")
Observation: "Oppenheimer" спечели Оскар за най-добър филм

Thought: Сега трябва да намеря режисьора
Action: search("Oppenheimer режисьор")
Observation: Christopher Nolan

Thought: Имам отговора
Answer: Christopher Nolan
```

Защо Thought помага?

Без Thought:

```
Action: search("Оскар 2024 режисьор") ← грешен query
```

С Thought:

```
Thought: Въпросът има две части:
         1) Кой филм спечели
         2) Кой е режисьорът
         Първо да намеря филма...
Action: search("Оскар 2024 най-добър филм") ← правилен query
```

ReAct vs Chain-of-Thought

CoT	ReAct
Reasoning в текст	Reasoning + Actions
Един изход	Много стъпки
Само модел knowledge	+ External knowledge
Може да hallucinate	Grounded в observations

ReAct = CoT + Tool Use + Grounding

Кога ReAct помага?

Помага:

- ✅ Multi-step въпроси
- ✅ Нужда от актуална информация
- ✅ Комбиниране на източници
- ✅ Verification на факти

Не помага:

- ❌ Прости въпроси
- ❌ Creative tasks
- ❌ Когато няма подходящи tools

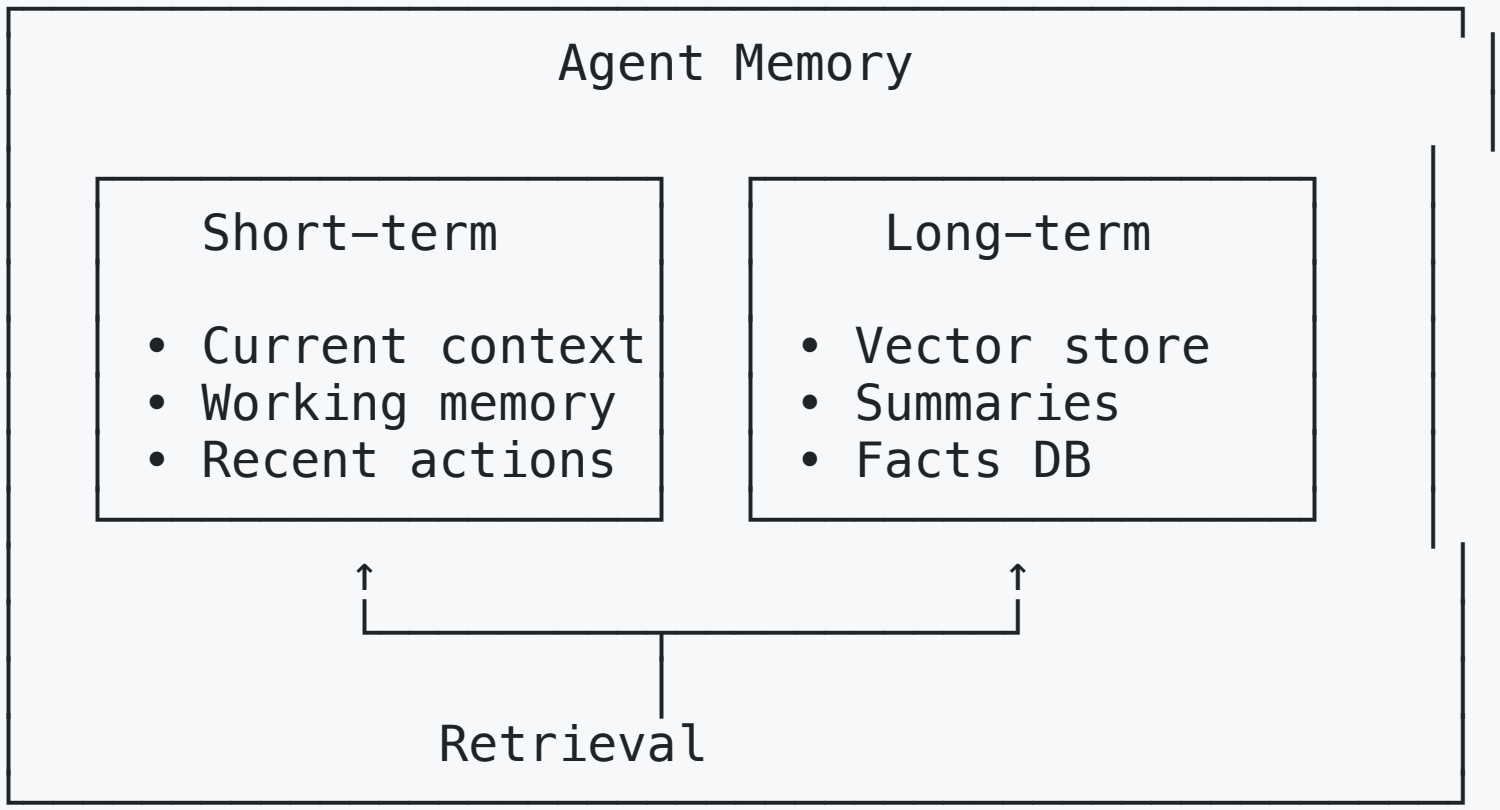
ReAct Failure Modes

```
1. Wrong tool selection:
   Thought: Да потърся...
   Action: calculator("население България") ← грешен tool

2. Reasoning loop:
   Thought: Да проверя отново...
   Action: search(same query)
   Thought: Да проверя пак... [∞]

3. Observation misinterpretation:
   Observation: "6.5 million (2021 estimate)"
   Thought: Населението е 6.5 ← miss-ва годината
```

Част 4: Agent Memory

Context Window ограничение

Types of Memory



Short-term Memory

Какво включва:

- Текущ conversation
- Последни N съобщения
- Working context за текущата задача

Стратегии:

```
1. Sliding window: Keep last N messages
2. Token budget: Keep until X tokens
3. Summarize old: Compress older messages
```

Long-term Memory

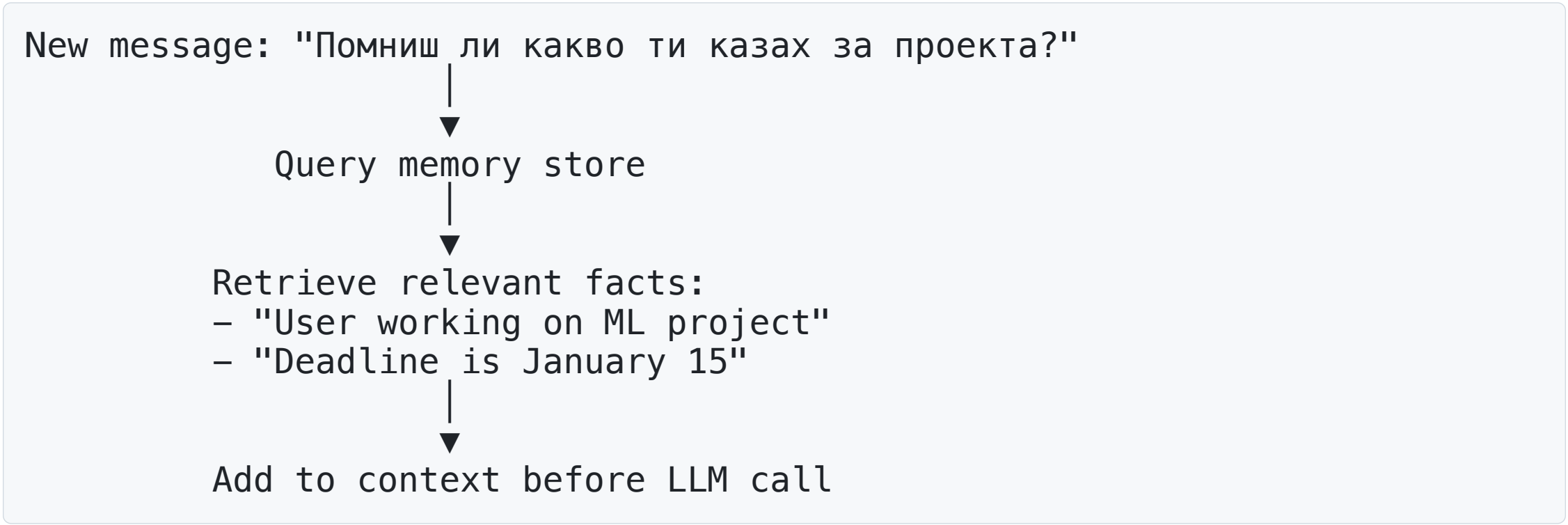
Какво включва:

- Facts научени от потребителя
- Минали разговори (summarized)
- Важни документи

Implementation:

```
User: Казвам се Иван, работя като програмист

Agent → Memory store:
  {
    "fact": "User name is Иван",
    "category": "personal"
  }
  {
    "fact": "User works as programmer",
```

Memory Retrieval



Episodic vs Semantic Memory

Episodic	Semantic
Конкретни събития	Общи факти
"Вчера говорихме за X"	"User предпочита Python"
Time-stamped	Без timestamp
Raw conversation	Extracted knowledge

Практика: Повечето agents използват и двете

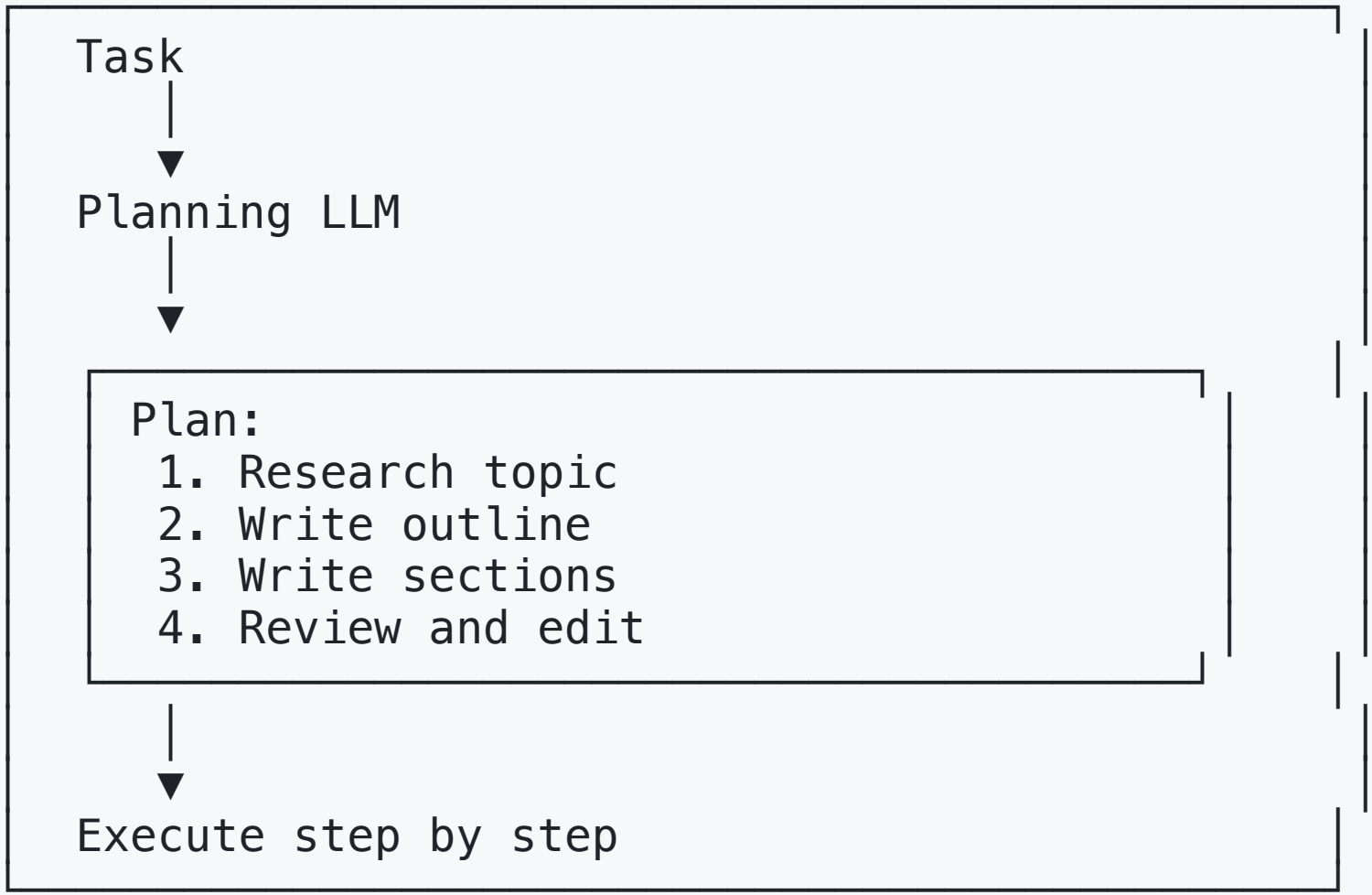
Memory Management

Част 5: Planning и Reflection

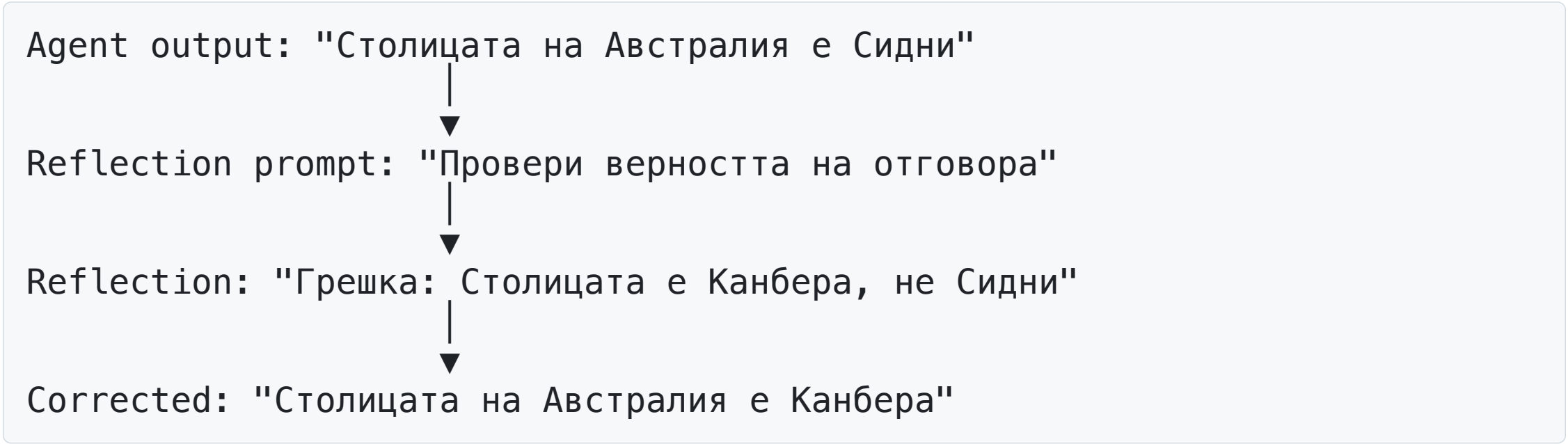
Защо Planning?

```
Task: "Напиши блог пост за машинно обучение,
       добави изображения и публикувай"

Без planning:
  Agent веднага започва да пише → забравя images

С planning:
  1. Напиши draft
  2. Генерирай/намери images
  3. Review и edit
  4. Публикувай
  → Следва структуриран план
```

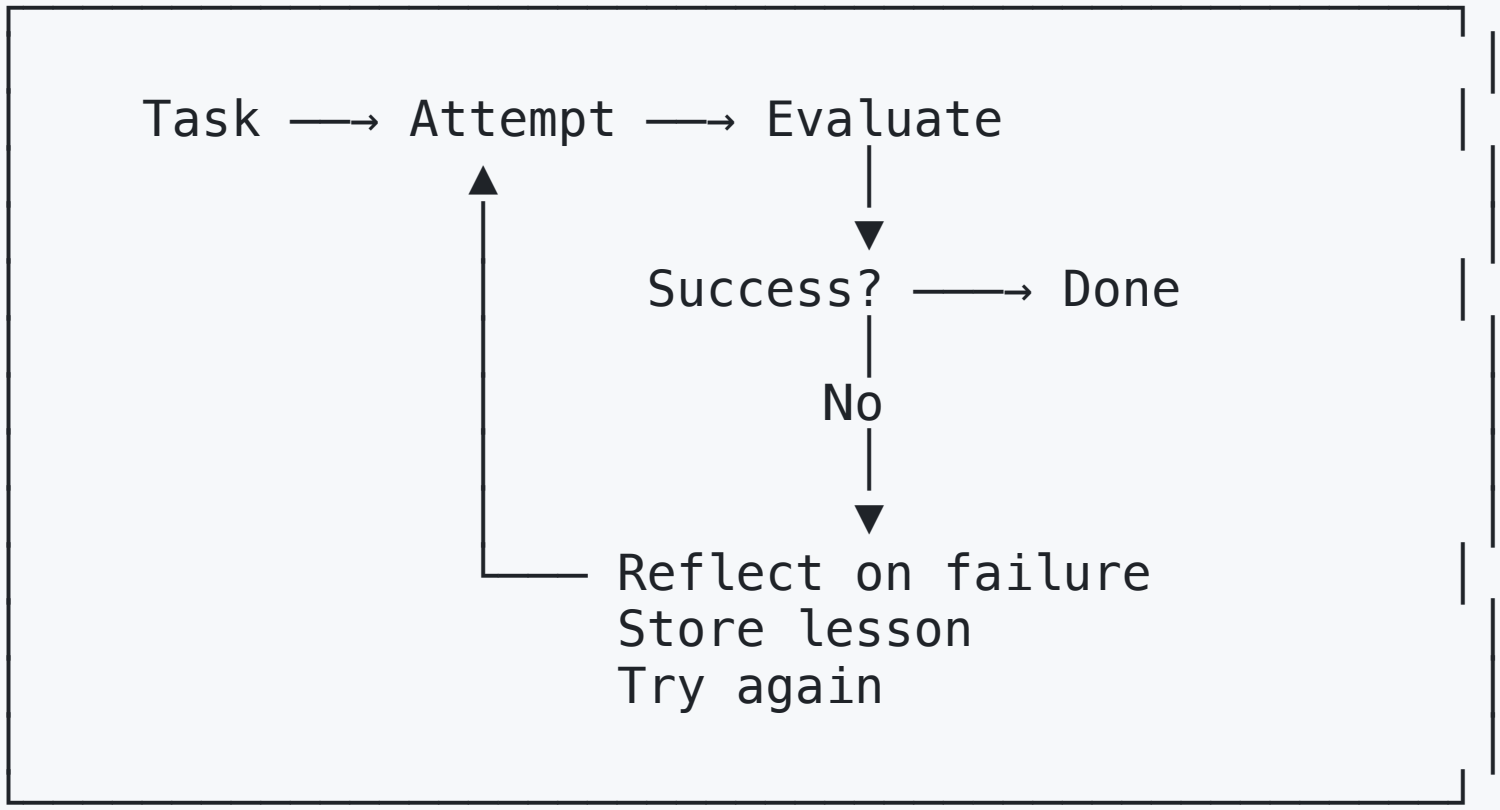
Plan-and-Execute Pattern



Reflection: Самокритика



Reflexion Pattern



Reflexion = Learn from mistakes

Self-Correction Loop

```
def solve_with_reflection(task, max_attempts=3):
    for attempt in range(max_attempts):
        solution = agent.solve(task)

        critique = agent.reflect(
            f"Провери това решение: {solution}"
        )

        if "correct" in critique.lower():
            return solution

        task = f"{task}\nПредишна грешка: {critique}"

    return solution  # Best effort
```

Предизвикателството

Проблем: Моделът не знае какво не знае

```
Agent: "Населението на България е 8 милиона"
Reflection: "Това звучи правилно" ← грешно!
```

Reflection работи по-добре за:

- Логически грешки
- Format violations
- Incomplete answers

Не толкова за:

- Factual errors (без external verification)

Част 6: Tools in Practice

Common Tool Categories

Категория	Примери	Use Case
Search	Web, Wikipedia	Information retrieval
Code	Python, SQL	Calculations, data
Files	Read, Write	Document processing
APIs	Weather, Stocks	Real-time data
Browser	Navigate, Click	Web automation

Tool Design Principles

Добър tool:

```
{
    "name": "get_weather",
    "description": "Get current weather for a city",
    "parameters": {
        "city": {"type": "string", "required": True},
        "units": {"type": "string", "enum": ["celsius", "fahrenheit"]}
    }
}
```

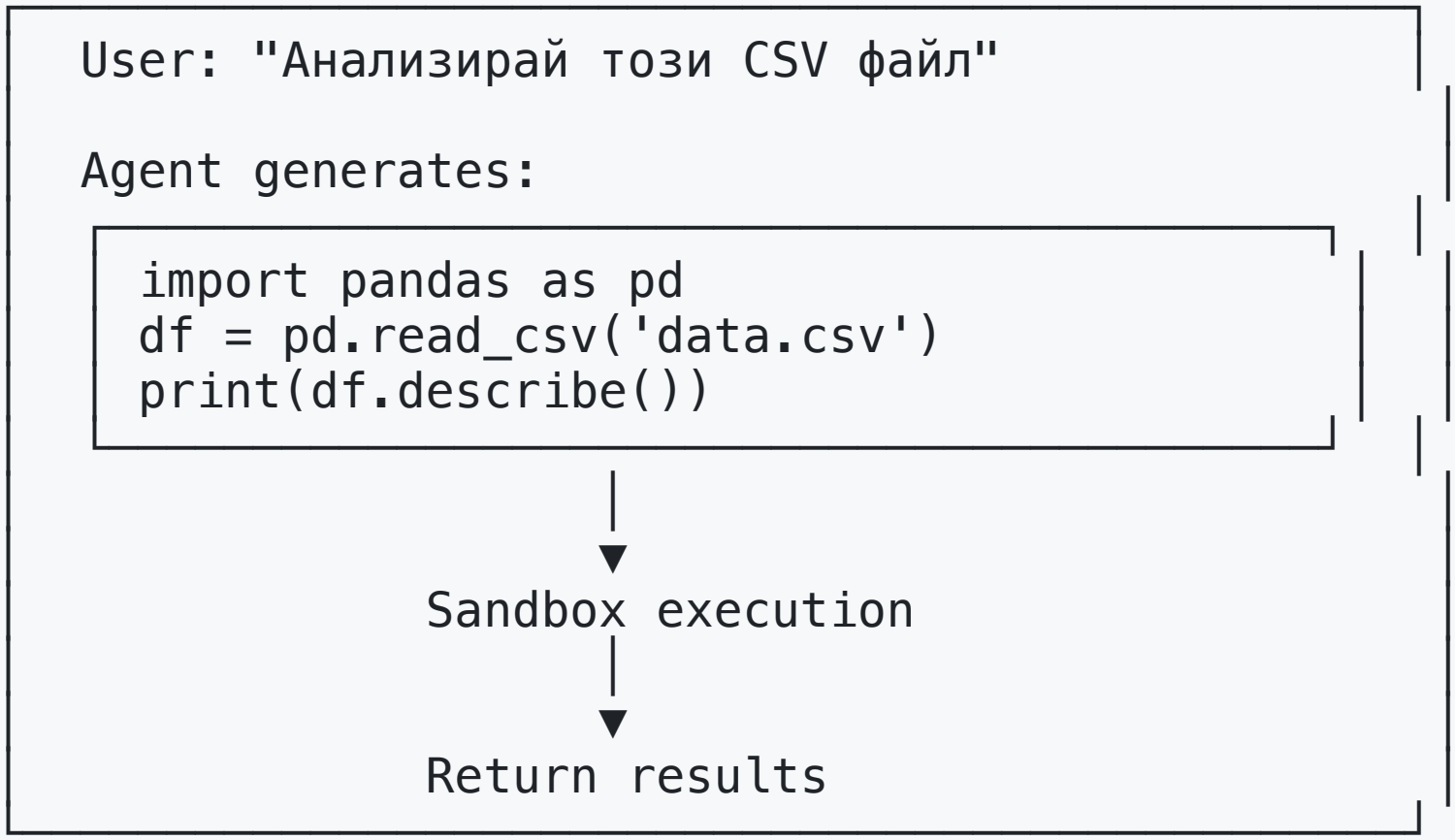
Лош tool:

```
{
    "name": "do_stuff",  # Неясно име
    "description": "Does things",  # Неясно описание
    "parameters": {}  # Липсват параметри
}
```

Tool Selection at Scale

```
Problem: 100+ tools → Model confused

Solutions:

1. Tool retrieval:
   Query → Embed → Find relevant tools → Use only those

2. Hierarchical tools:
   "Category: Data" → "Subcategory: SQL" → Specific tools

3. Tool recommendation:
   Few-shot examples of which tool for which task
```

Code Interpreters

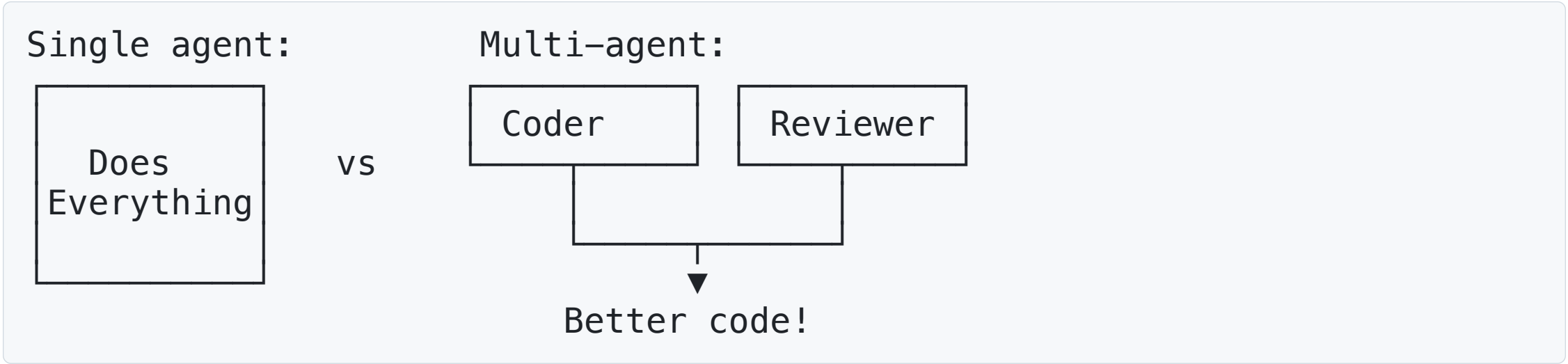


Security Considerations

```
┌──────────────────────────────────────────────────┐
│                  SECURITY                        │
│                                                  │
│  ✗ Arbitrary code execution                      │
│  ✗ File system access without limits             │
│  ✗ Network requests to anywhere                  │
│  ✗ Credential exposure                           │
│                                                  │
│  ✓ Sandboxed environments                        │
│  ✓ Whitelisted tools only                        │
│  ✓ Rate limiting                                 │
│  ✓ Input validation                              │
│  ✓ Output sanitization                           │
└──────────────────────────────────────────────────┘
```

Част 7: Multi-Agent Systems

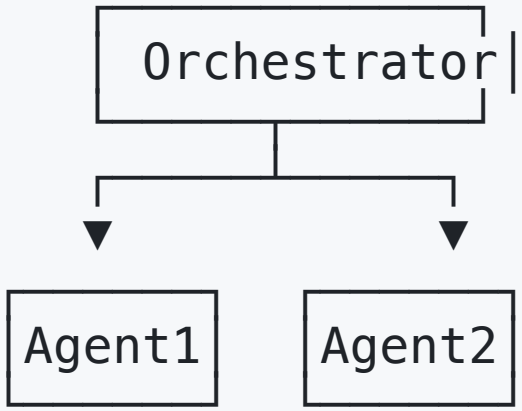
Защо повече от един agent?



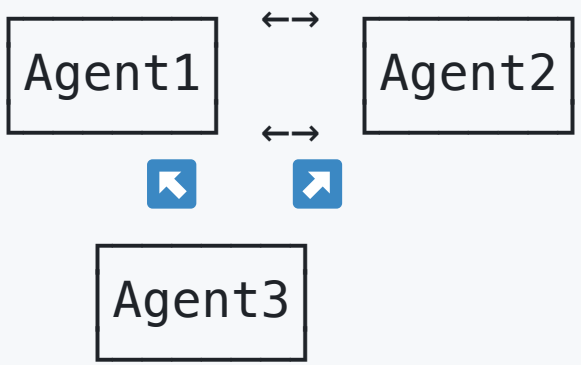
Ползи: Специализация, diverse perspectives, debate

Multi-Agent Architectures

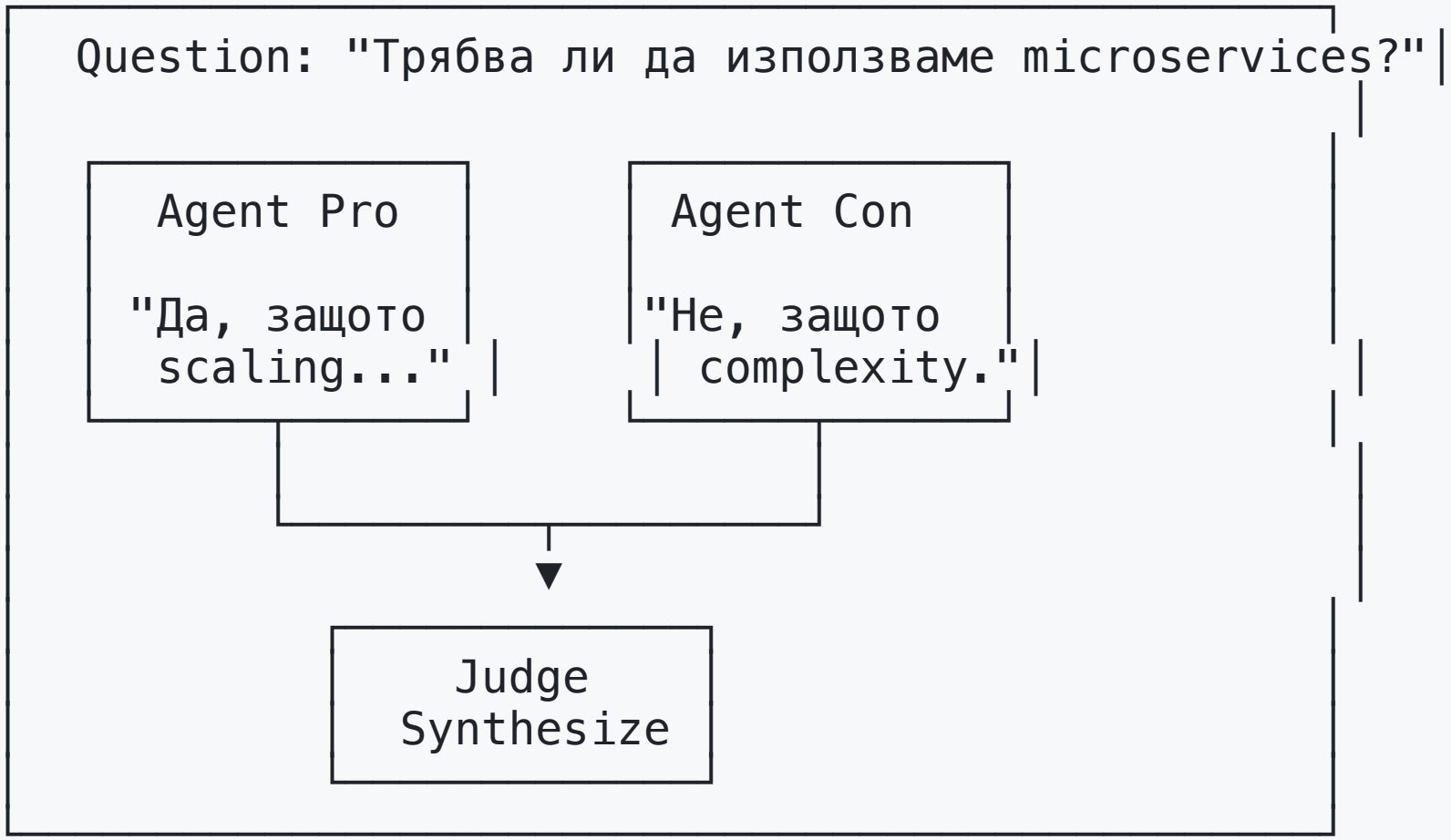
1. Hierarchical:



2. Peer-to-peer:



Debate Architecture



Practical Examples

Coding assistant:

Research team:

Coordination Challenges

Challenge	Problem	Mitigation
Communication	Agents misunderstand each other	Clear protocols
Overhead	Many LLM calls	Efficient routing
Conflicts	Disagreements	Resolution rules
Loops	Infinite back-and-forth	Max iterations

Reality: Multi-agent добавя complexity за marginal gains

Част 8: Results и Limitations

Benchmarks

Benchmark	Measures	Top Performance
SWE-bench	Code fixes	~50% (verified)
WebArena	Web tasks	~35%
GAIA	General assistant	~70% (Level 1)

Note: Numbers change fast, top models improve monthly

Къде Agents успяват

✅ Structured tasks with clear tools
"Search for X, summarize results"

✅ Well-defined workflows
"Parse this PDF, extract tables"

✅ Code generation with feedback
"Write code, run tests, fix errors"

✅ Information synthesis
"Research topic from multiple sources"

Къде Agents се провалят

❌ Long-horizon planning
Error accumulates over many steps

❌ Ambiguous tasks
"Make this better" → unclear actions

❌ Recovery from errors
One mistake → cascade of failures

❌ Novel situations
No training data for new tools

The Reliability Problem

```
Single step accuracy: 95%

Multi-step task (10 steps):
  0.95^10 = 59.8% success rate

Multi-step task (20 steps):
  0.95^20 = 35.8% success rate
```

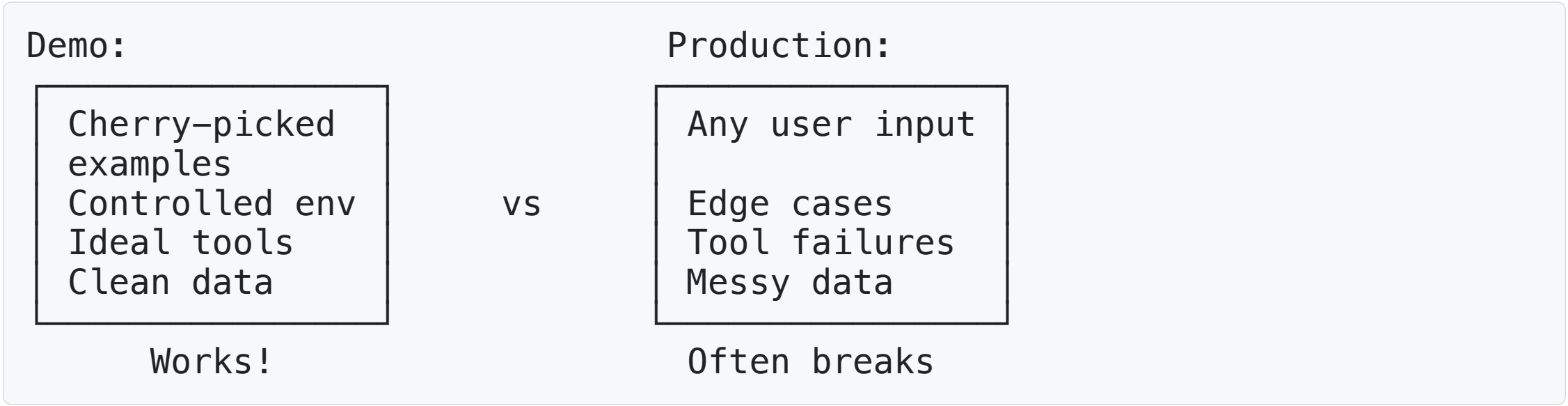
Малки грешки се натрупват експоненциално

Cost и Latency

```
Simple query:
  1 LLM call × $0.01 = $0.01, 1 second

Agent task (20 tool uses):
  ~40 LLM calls × $0.01 = $0.40, 60+ seconds
```

Metric	Simple LLM	Agent
Cost	1×	10-50×
Latency	1s	30s-5min
Reliability	High	Medium-Low

Demo vs Production Gap

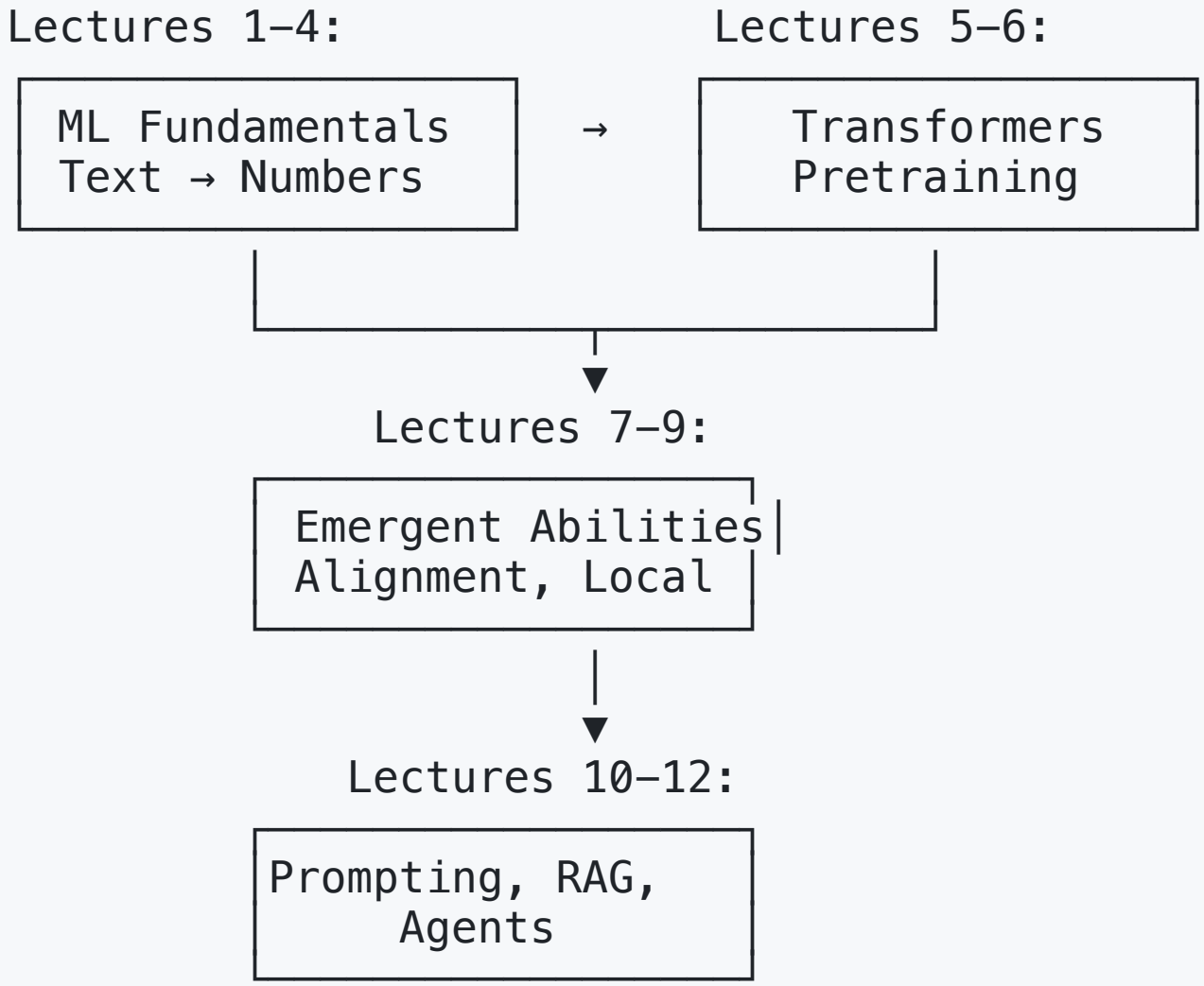


Обобщение

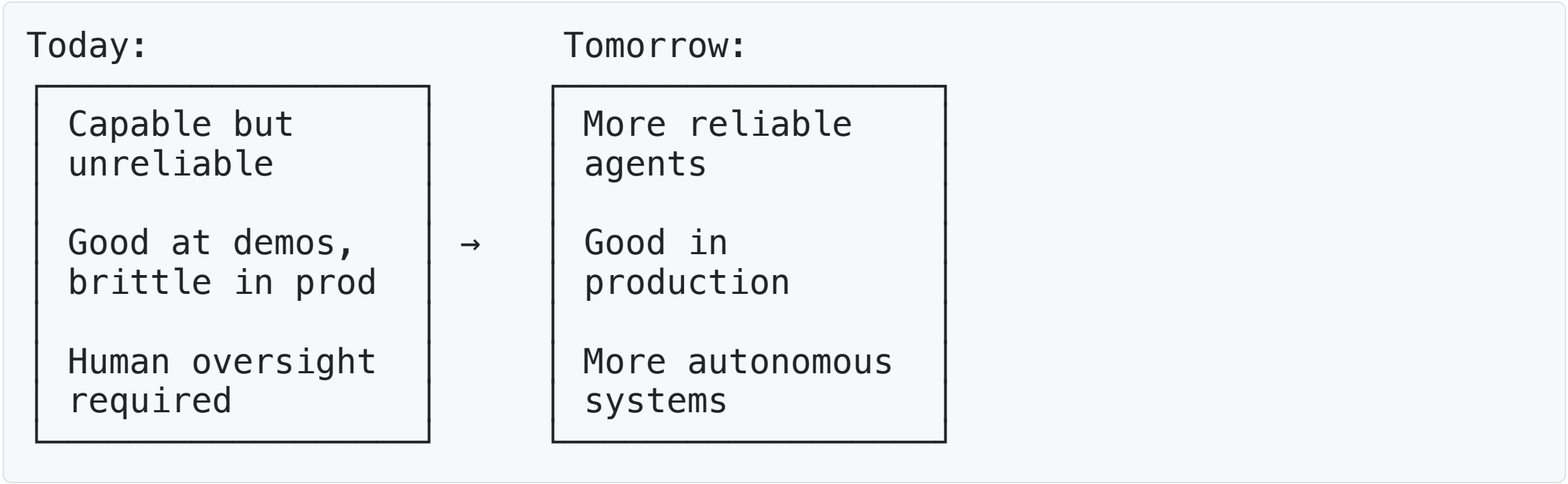
Ключови идеи

1. **Agents** = LLM + Tools + Loop → действия в света
2. **ReAct** = Reasoning + Acting → по-добър tool selection
3. **Memory** = Short-term + Long-term → дълги interactions
4. **Planning** = Decompose tasks → структуриран подход
5. **Reflection** = Self-critique → самокорекция (ограничена)
6. **Multi-agent** = Специализация → complexity trade-off
7. **Reality** = Capable but unreliable → use carefully

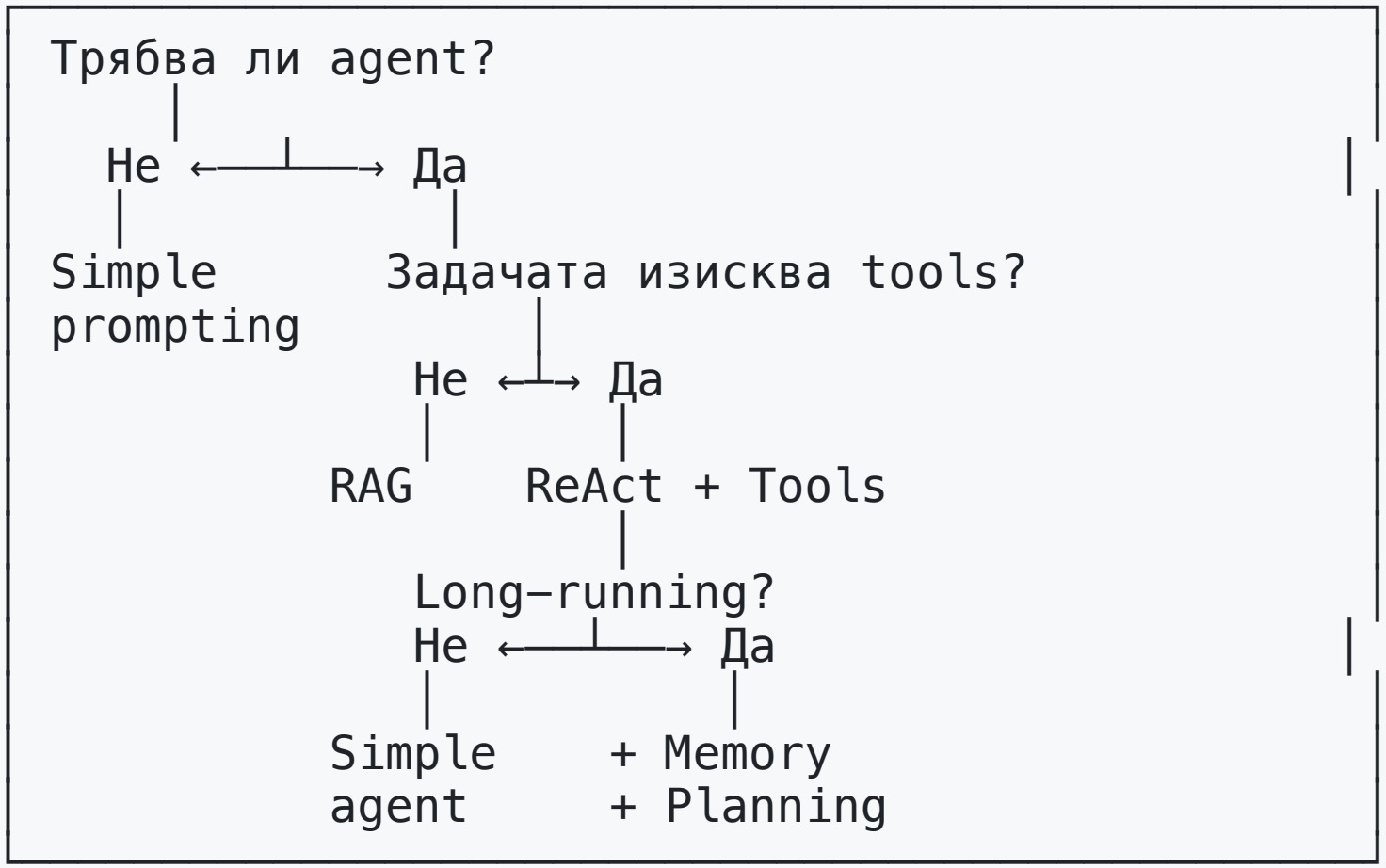
Course Arc


Lectures 1–4:
ML Fundamentals
Text → Numbers
→
Lectures 5–6:
Transformers
Pretraining
Lectures 7–9:
Emergent Abilities
Alignment, Local
Lectures 10–12:
Prompting, RAG,
Agents

What's Next for the Field?



Практически Framework



Ресурси

Papers:

- Yao et al. (2022) — ReAct
- Shinn et al. (2023) — Reflexion
- Park et al. (2023) — Generative Agents
- Wang et al. (2024) — Survey of LLM-based Agents

Tools:

- LangChain, LlamaIndex — Agent frameworks
- Claude MCP — Tool integration
- OpenAI Assistants API

Въпроси?