CLAUD CODE ARCHITECTURE 이해

Claude Code Architecture

다이어그램으로 보는 에이전트 하네스(Agentic Harness) 구조

터미널 기반 에이전틱 코딩 도구의 내부 동작 원리, 공식 문서 기반 심층 분석

Choi, WooHyung
Prin. Solutions Architect
AWS Korea
whchoi@amazon.com

Agenda

8개 레이어로 분해한 Claude Code 아키텍처

01	Claude Code 개요	에이전트 하네스란 무엇인가
02	전체 아키텍처 맵	여덟 개 레이어 한눈에 조망
03	The Agentic Loop	컨텍스트-행동-검증 3단계 루프
04	Models & Tools	추론 엔진과 다섯 가지 내장 도구
05	에이전트 구동 + Skills	4파일과 SKILL.md 표준 - 로딩, 구조, 호환
06	Input Layer	인터페이스, 실행 환경, 세션, 권한
07	Knowledge Layer	컨텍스트 윈도우와 메모리, Skills
08	Execution & Multi-Agent	도구 실행, 캐시, 서브에이전트 위임
09	Observability & Integration	Hooks 라이프사이클과 MCP 통합
10	프리미티브 선택과 종합	Skills/Subagents/MCP/Hooks 결정, End-to-End, 요약

01

Claude Code 개요

Claude Code 개요

LLM을 코딩 에이전트로 만드는 에이전트 하네스

AGENTIC HARNESS

Claude Code는 Claude 모델을 감싸는 에이전트 하네스.

모델에 도구(tools), 컨텍스트 관리, 실행 환경을 제공해 언어 모델을 실제로 코드를 작성하고 검증하는 코딩 에이전트로 전환.

TERMINAL-NATIVE

터미널 네이티브

CLI에서 직접 동작하며 모든 CLI 도구와 결합, IDE/CI-CD/웹으로 확장

CODEBASE-AWARE

코드베이스 이해

프로젝트 전체 파일, git 상태, 의존 관계를 에이전틱 검색으로 파악

TOOL-DRIVEN

도구 기반 행동

파일 편집, 셸 실행, 웹 검색 등 도구로 실제 작업을 수행하고 반복

HUMAN-IN-LOOP

사람 중심 통제

수정/실행 전 명시적 권한, 체크포인트로 모든 변경을 되돌리기 가능

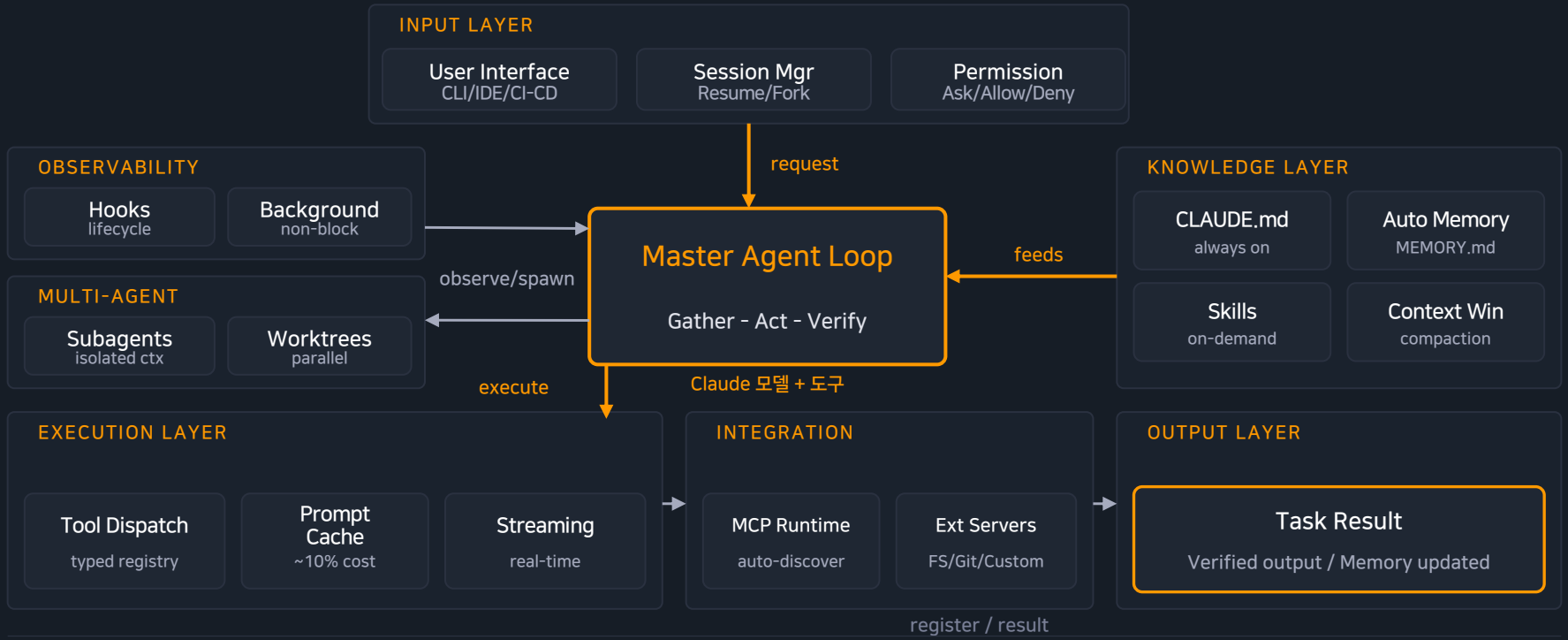
3 phase	5종	4개	3개
에이전틱 루프 단계	내장 도구 카테고리	확장 레이어	실행 환경

02

전체 아키텍처 맵

전체 아키텍처 맵

Master Agent Loop을 중심으로 한 여덟 개 레이어



03

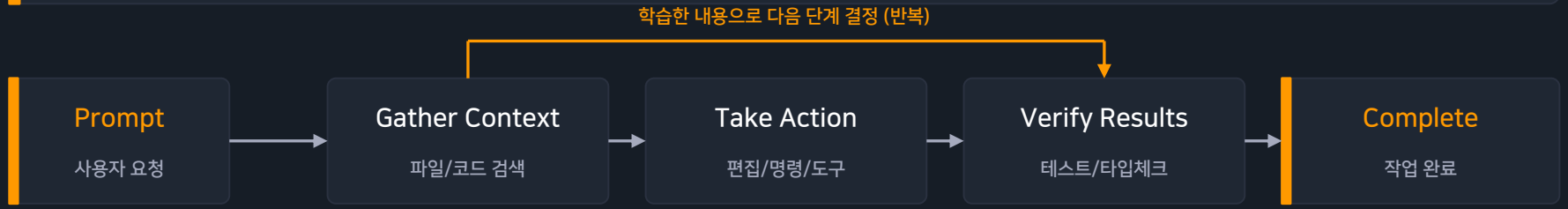
Agent Loop

The Agentic Loop

모든 작업을 관통하는 3단계 핵심 루프

CORE LOOP

사용자 프롬프트가 들어오면 Claude는 컨텍스트 수집(Gather Context), 행동(Take Action), 결과 검증(Verify Results)을 반복.
단계는 서로 섞이며 작업 성격에 따라 루프가 적응적으로 동작.



질문형 작업
컨텍스트 수집만으로 응답

버그 수정
세 단계를 여러 번 반복

리팩터링
검증 단계에 더 많은 비중

Gather
맥락을 파악

Act
도구로 행동

Verify
결과를 확인

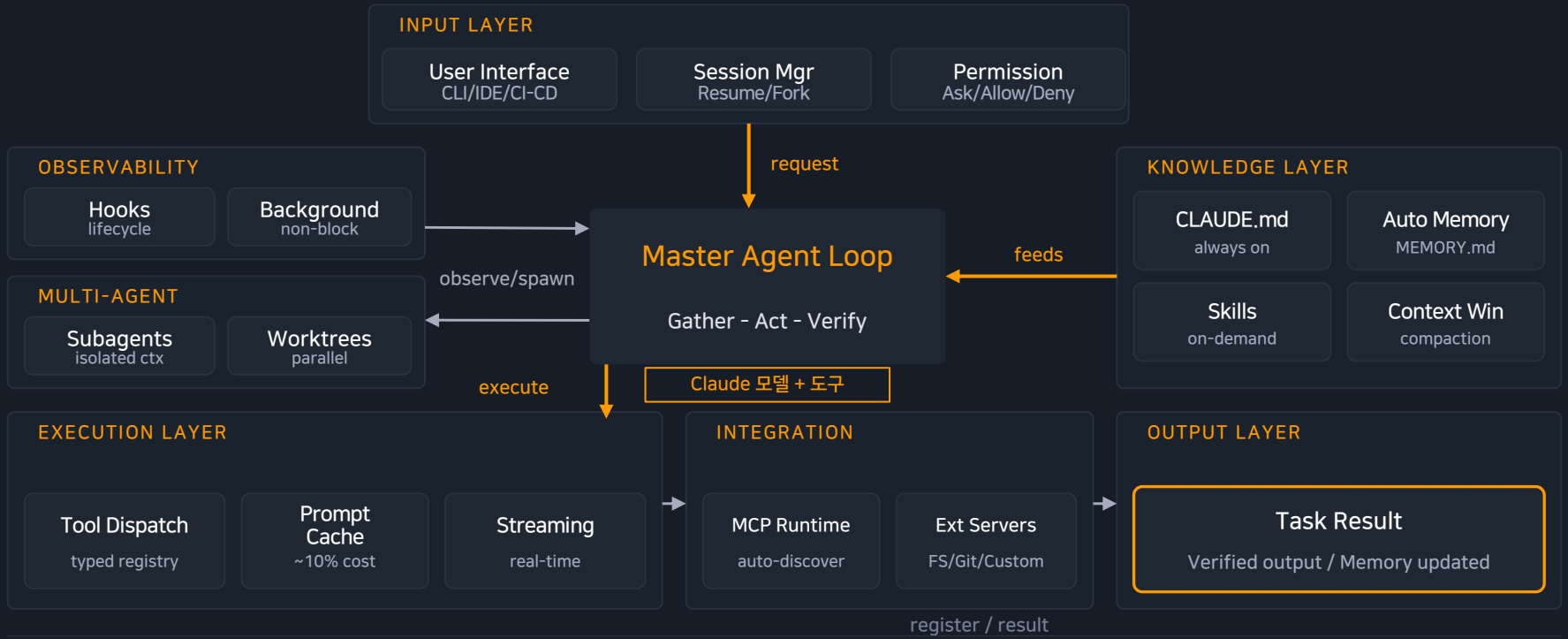
Esc
언제든 중단/재지시

04

Models & Tools

전체 아키텍처 맵

Master Agent Loop을 중심으로 한 여덟 개 레이어



Models & Tools

루프를 움직이는 두 축, 추론하는 모델과 행동하는 도구

MODELS - 추론 엔진

Sonnet은 대부분의 코딩 작업을, Opus는 복잡한 아키텍처 추론을 담당. `/model` 명령으로 세션 중 전환

TOOLS - 행동 수단

도구가 없으면 텍스트만 생성. 도구로 읽고 편집하고 실행하며, 각 결과가 루프로 피드백

FILE	SEARCH	EXECUTION	WEB	CODE INTEL
파일 작업	검색	실행	웹	코드 인텔리전스
읽기, 편집, 생성, 이름변경과 재구성	패턴/정규식 검색, 코드베이스 탐색	셸 명령, 서버 기동, 테스트, git	웹 검색, 문서 가져오기, 에러 조회	타입 오류, 정의 이동, 참조 (플러그인)

Sonnet
대부분의 코딩 작업

Opus
복잡한 추론/설계

5 + α
내장 도구 + 오케스트레이션

/model
세션 중 모델 전환

05

Agent 구동 및 Skills

에이전트를 움직이는 4가지 파일

CLAUDE.md / SKILL.md / Prompt / Hook - 무엇이고 언제 쓰는가

FOUR INPUTS

에이전트의 행동은 네 가지 입력으로 제어.

무엇을(컨텍스트), 어떻게(절차), 지금(지시), 자동으로(이벤트) - 각각 로딩 시점과 용도가 다름.

MEMORY

CLAUDE.md

프로젝트 영속 컨텍스트.
규칙, 명령, 구조를 매 세션 자동 로드.
항상 읽혀야 할 때 사용

SKILL

SKILL.md

재사용 절차서.
설명이 매칭될 때만 본문 로드.
반복되는 방법(how-to)을 담을 때 사용

PROMPT

Prompt

지금의 일회성 지시. 눈앞의 작업 하나를 처리하는 단발성 요청에 사용

HOOK

Hook

settings.json의 결정적 코드.
이벤트에 자동 실행
- 린트, 시크릿 차단, 로깅에 사용

항상
CLAUDE.md - 매 턴 로드

필요 시
SKILL.md - 설명 매칭

지금
Prompt - 일회성 지시

자동
Hook - 이벤트 트리거

무엇이 언제 컨텍스트에 들어오나

로딩 시점과 트리거 비교 - 항상 / 필요할 때 / 지금 / 자동

CLAUDE.md 항상	SKILL.md 필요할 때	Prompt 지금	Hook 자동
로드 시점 세션 시작 + 매 턴	로드 시점 설명 매칭 시 본문	로드 시점 사용자 입력 시점	로드 시점 라이프사이클 이벤트
트리거 자동 (조건 없음)	트리거 description 매칭	트리거 수동 (사람)	트리거 PreToolUse 등 이벤트
대표 예시 규칙, 명령, 구조	대표 예시 PDF 폼 채우기 절차	대표 예시 getUser 리팩터링	대표 예시 편집 시 lint 실행

세션 시작
CLAUDE.md 진입

설명 매칭
SKILL.md 트리거

사용자 입력
Prompt 발생

이벤트
Hook 발동

흔한 함정과 베스트 프랙티스

각 파일의 GOTCHA - 잘못 쓰면 이렇게 됩니다

GOTCHAS

강력한 만큼 함정도 분명. 네 가지만 피하면 에이전트가 안정적으로 동작.

BLOAT

비대화 주의

비대해지면 컨텍스트 낭비.
매 턴 로드되므로 꼭 필요한 것만 간결하게 유지

WEAK DESC

설명이 트리거

설명이 약하면 영영 호출 안 됨.
사람용이 아니라 매칭을 위해 작성

VAGUE

모호함 금지

모호한 지시는 모호한 결과.
목표, 제약, 완료 기준을 분명히 명시

BAD EXIT

격리 테스트

실제 셸로 실행됨.
잘못된 exit code가 에이전트를 멈출 수 있어 격리 환경에서 먼저 검증

간결하게
CLAUDE.md 유지

매칭 우선
SKILL.md 설명

명시적으로
Prompt 작성

격리 검증
Hook 테스트

SKILL.md - 범용 스킬 표준

재사용 가능한 온디맨드 능력을 여는 오픈 표준

OPEN STANDARD

AI 코딩 에이전트에 재사용 가능한 온디맨드 능력을 부여하는 오픈 표준.

Anthropic이 공개했고 규격은 agentskills.io에 있으며, Cursor / GitHub / JetBrains 등 여러 도구가 같은 포맷을 채택.

SKILL.md (예: review-pr)

```
---
name: review-pr
description: Reviews PRs for
  bugs, security, quality.
  Use when reviewing PRs.
---
# Review Checklist
1. Check security vulns
2. Verify test coverage
```

`name`	그대로 /slash-command 가 됨
`description`	작업과 매칭되면 자동 트리거
`body`	매칭될 때만 컨텍스트에 로드

Anthropic	16+	agentskills.io	온디맨드
오픈 표준 공개	채택 도구	공개 규격	매칭 시 로드

Progressive Disclosure

3단계 점진적 로딩 - 수백 개 스킬, 컨텍스트 부풀림 제로

3-LEVEL LOADING

스킬은 세 단계로 점진적으로 로드. 평소엔 이름과 설명만 읽고, 작업에 매칭되면 본문을, 본문이 요구할 때만 스크립트와 참조를 로드.
스킬이 수백 개여도 컨텍스트 부풀림이 없음.

LEVEL 1

이름 + 설명

에이전트가 항상 읽는 메타데이터.
약 30-50 토큰만 점유

LEVEL 2

SKILL.md 본문

작업과 매칭되면 전체 지시를 로드.
5,000 토큰 미만 권장

LEVEL 3

스크립트 / 참조

본문이 요구할 때만 scripts 와 references 를 온디맨드 로드

30-50 토큰
Level 1 메타데이터

5,000 미만
Level 2 본문

온디맨드
Level 3 자원

0 bloat
평소 컨텍스트

스킬 구조와 프론트매터

디렉터리 레이아웃과 핵심 메타데이터 필드

디렉터리 구조

```
.claude/skills/review-pr/
   SKILL.md          (필수: 지시)
   scripts/          (실행 코드)
      lint.sh    scan.py
   references/       (문서, 선택)
      api-docs.md
   assets/           (템플릿, 선택)
      report-template.md
```

핵심 프론트매터 필드

`name`	그대로 /slash-command 가 됨
`description`	자동 매칭 트리거 (가장 중요)
`allowed_tools`	도구 접근 권한 제어
`agent`	서브에이전트로 실행
`hooks`	라이프사이클 콜백 연결

500줄
SKILL.md 이하 권장

5,000 토큰
지시 이하 권장

scripts/
무거운 로직 분리

Git
팀과 스킬 공유

Skills vs Configs vs MCP

무엇을 언제 - 포맷은 표준, 경로는 도구별

	Skills SKILL.md	Configs CLAUDE.md	MCP 외부 연결
무엇	온디맨드 전문성	항상 켜진 컨텍스트	외부 도구 / 데이터
로드 시점	트리거 시 로드	매 세션 로드	시작 시 연결
이식성	16+ 도구 (오픈 포맷)	단일 도구	오픈 프로토콜
용도	반복 워크플로	프로젝트 규칙	API, DB, 서비스

표준 포맷
모든 도구 동일

경로별
.claude / .cursor / .agents

16+
채택 도구

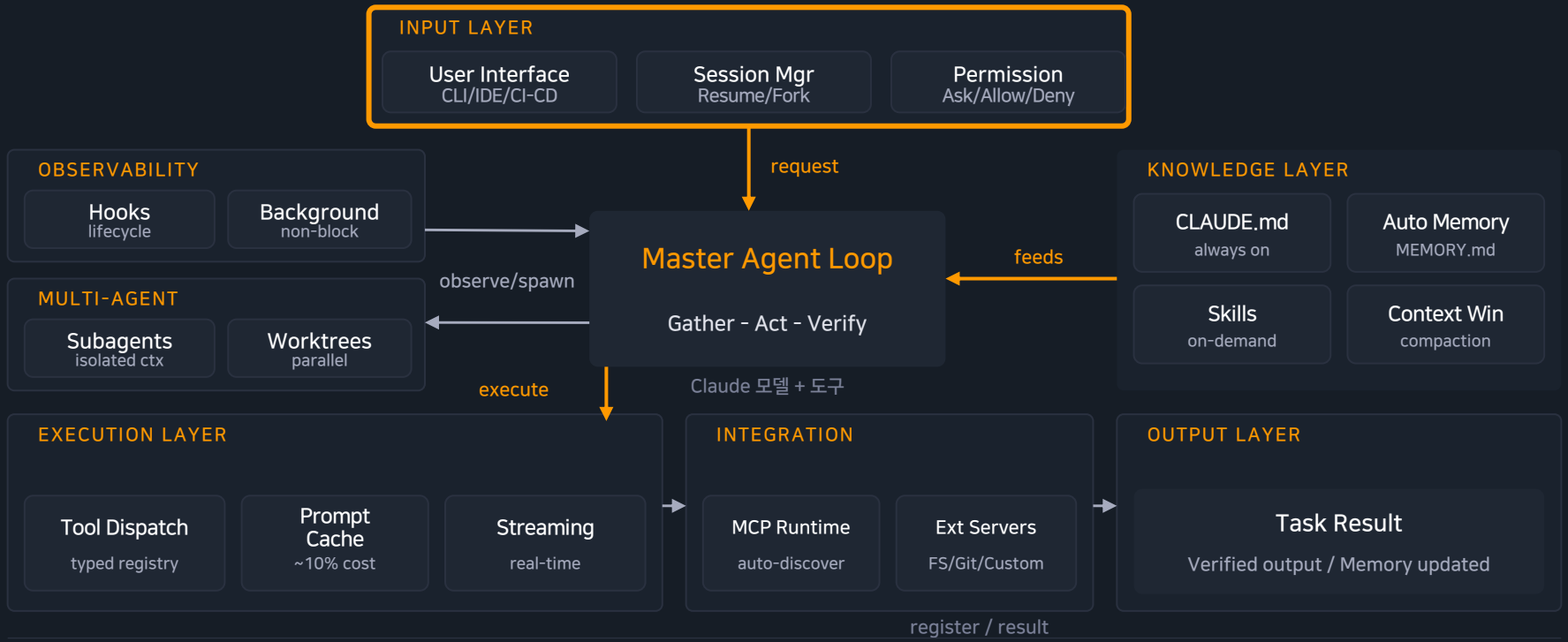
상호 보완
Skills + Config + MCP

06

Input Layer

전체 아키텍처 맵

Master Agent Loop을 중심으로 한 여덟 개 레이어



Input Layer - 인터페이스와 실행 환경

어디서 실행하고 어떻게 상호작용하는가

SAME LOOP, EVERYWHERE

에이전틱 루프와 도구, 기능은 어디서나 동일.

달라지는 것은 코드가 실행되는 위치(실행 환경)와 사용자가 상호작용하는 방식(인터페이스).

실행 환경 (Execution Environments)

Local
내 머신에서 실행, 파일/도구 전체 접근 (기본값)

Cloud
Anthropic 관리 VM에서 실행, 작업 오프로드

Remote Control
브라우저로 제어하되 내 머신에서 로컬 실행

인터페이스 (Interfaces)

- 터미널 (CLI)
- 데스크톱 앱
- VS Code / JetBrains
- claude.ai/code (웹)
- Remote Control
- Slack
- CI-CD (GitHub Actions)
- tag @claude (PR)

Local
기본 실행 환경

동일
루프/도구는 어디서나 같음

8+
접근 인터페이스

JSONL
세션은 로컬에 영구 저장

Sessions & Permissions

세션 연속성과 안전장치, 무엇을 허용할지 통제

세션 연속성 (Resume / Fork)

원본 유지 + 분기
git worktree로 병렬 세션 동시 실행

CHECKPOINTS

편집 전 파일 스냅샷 자동 저장. Esc 두 번으로 되감기.
git과 별개이며 파일 변경만 커버 (외부 부수효과는 체크포인트 불가).

권한 모드 (Shift+Tab으로 순환)

Default
파일 편집/셸 명령 전 매번 확인

Auto-accept edits
편집/일반 FS 명령 자동, 그 외 확인

Plan mode
읽기 전용 도구만, 계획 후 승인

Auto mode
백그라운드 안전성 검사 (research preview)

.claude/settings.json 으로 특정 명령 사전 허용, 조직에서 개인까지 정책 스코프 계층

Resume
같은 세션 이어가기

Fork
분기해 새 세션

Esc Esc
파일 변경 되감기

4 modes
권한 레벨 순환

07

Knowledge Layer

전체 아키텍처 맵

Master Agent Loop을 중심으로 한 여덟 개 레이어

2. Knowledge Layer - 컨텍스트와 메모리

무엇이 컨텍스트 윈도우로 들어오는가

CONTEXT SOURCES

컨텍스트 윈도우는 대화 히스토리, 파일 내용, 명령 출력, CLAUDE.md, 자동 메모리, 로드된 Skills, 시스템 지시를 담음.

영구적으로 적용할 규칙은 대화가 아니라 CLAUDE.md에 둠.

CLAUDE.md
항상 로드되는 규칙

200줄
세션 시작 시 메모리 로드

on-demand
Skills 본문 지연 로드

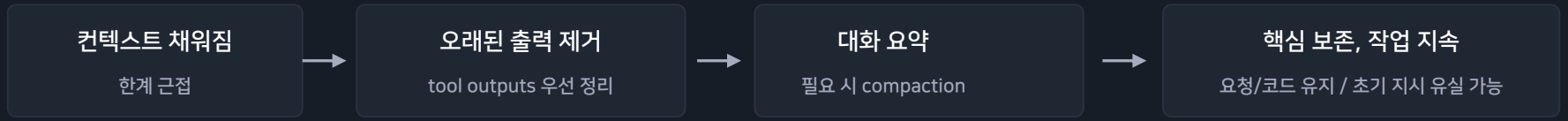
/context
컨텍스트 사용량 점검

Context Window 관리

가득 차면 어떻게 되는가, 자동 컴팩션 흐름

AUTO-COMPACTION

컨텍스트가 한계에 가까워지면 Claude Code가 자동으로 관리합니다. 오래된 도구 출력을 먼저 비우고, 필요하면 대화를 요약합니다. 요청과 핵심 코드 스니펫은 보존되지만 초기 상세 지시는 유실될 수 있습니다.



Compact Instructions
CLAUDE.md 섹션으로 보존 지정

/compact focus
요약 초점을 직접 지정

MCP 도구는 deferred
tool search로 온디맨드, 이름만 소비

출력 먼저
오래된 tool output 정리

요약
대화 compaction

CLAUDE.md
영구 규칙은 여기에

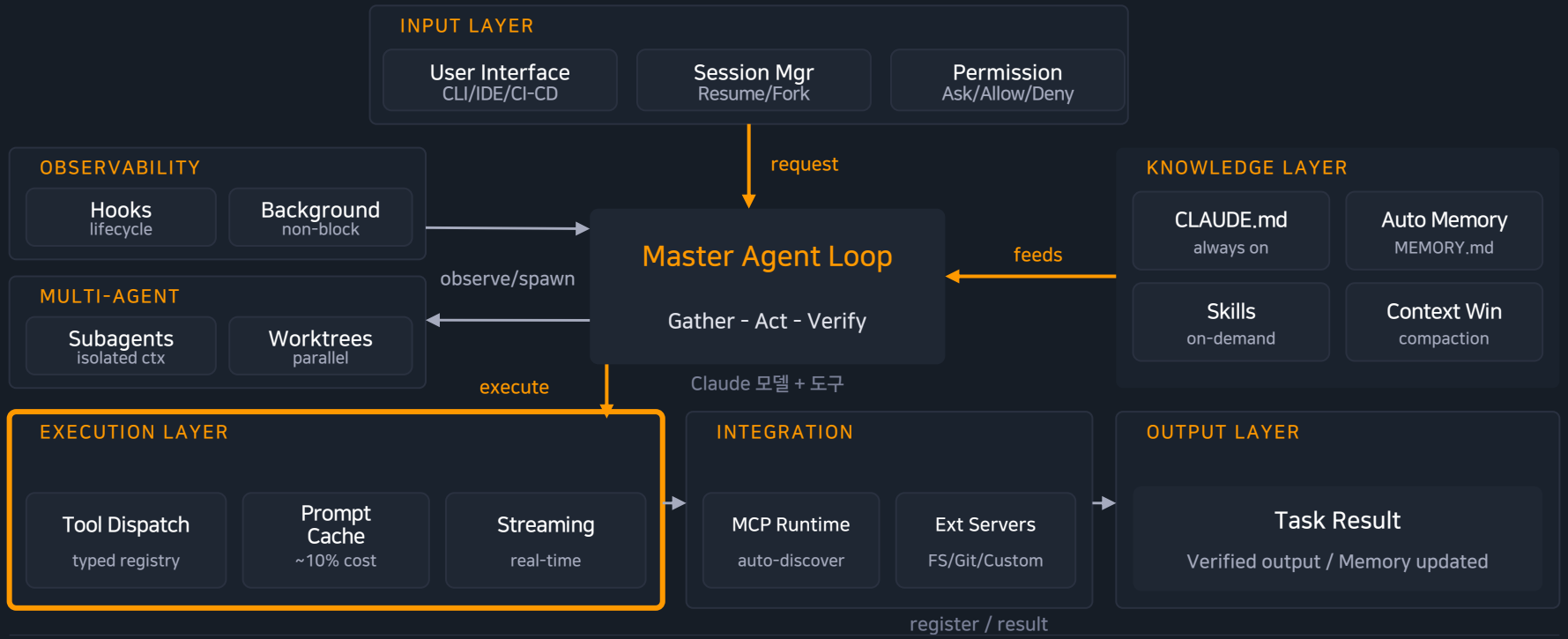
/mcp
서버별 컨텍스트 비용

08

Execution & Multi-Agent Layer

전체 아키텍처 맵

Master Agent Loop을 중심으로 한 여덟 개 레이어



Execution Layer - 도구 실행

행동을 실제 결과로 바꾸는 실행 파이프라인

TOOL EXECUTION

Take Action 단계에서 모델은 구조화된 도구 호출을 내보내고, 핸들러가 이를 해석해 실행합니다. 결과는 다시 루프로 피드백되며, 프롬프트 캐시와 스트리밍이 비용과 응답성을 끌어올립니다.

Prompt Cache (5분, 1시간)
안정적 프리픽스 재사용, 캐시 읽기 비용 약 10%

Streaming Runtime
실시간 출력, 독립적 도구 호출은 병렬 실행

tool_use
구조화된 도구 호출

핸들러
도구별 실행 라우팅

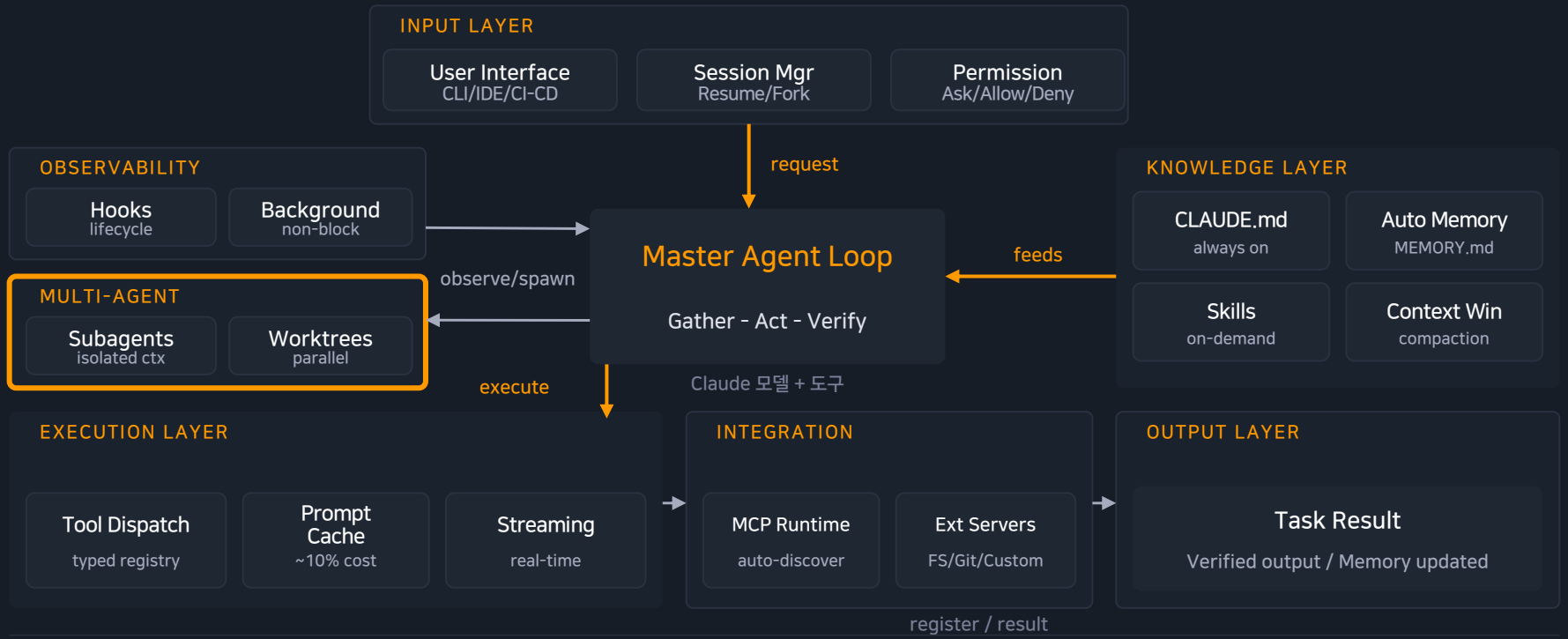
~10%
캐시 읽기 비용

병렬
독립 도구 동시 실행

출처: Anthropic Claude Docs (code.claude.com), Prompt Caching

전체 아키텍처 맵

Master Agent Loop을 중심으로 한 여덟 개 레이어



Multi-Agent Layer - 서브에이전트

작업을 위임하고 컨텍스트를 격리한다

DELEGATE, DON'T BLOAT

메인 에이전트가 계획과 통합을 맡고, 특화된 서브에이전트가 각자 자체 컨텍스트에서 작업을 수행합니다. 끝나면 요약만 반환해 메인 컨텍스트를 오염시키지 않습니다.

Main
계획/통합 담당

격리
서브에이전트 자체 컨텍스트

병렬
동시 실행, 실패 격리

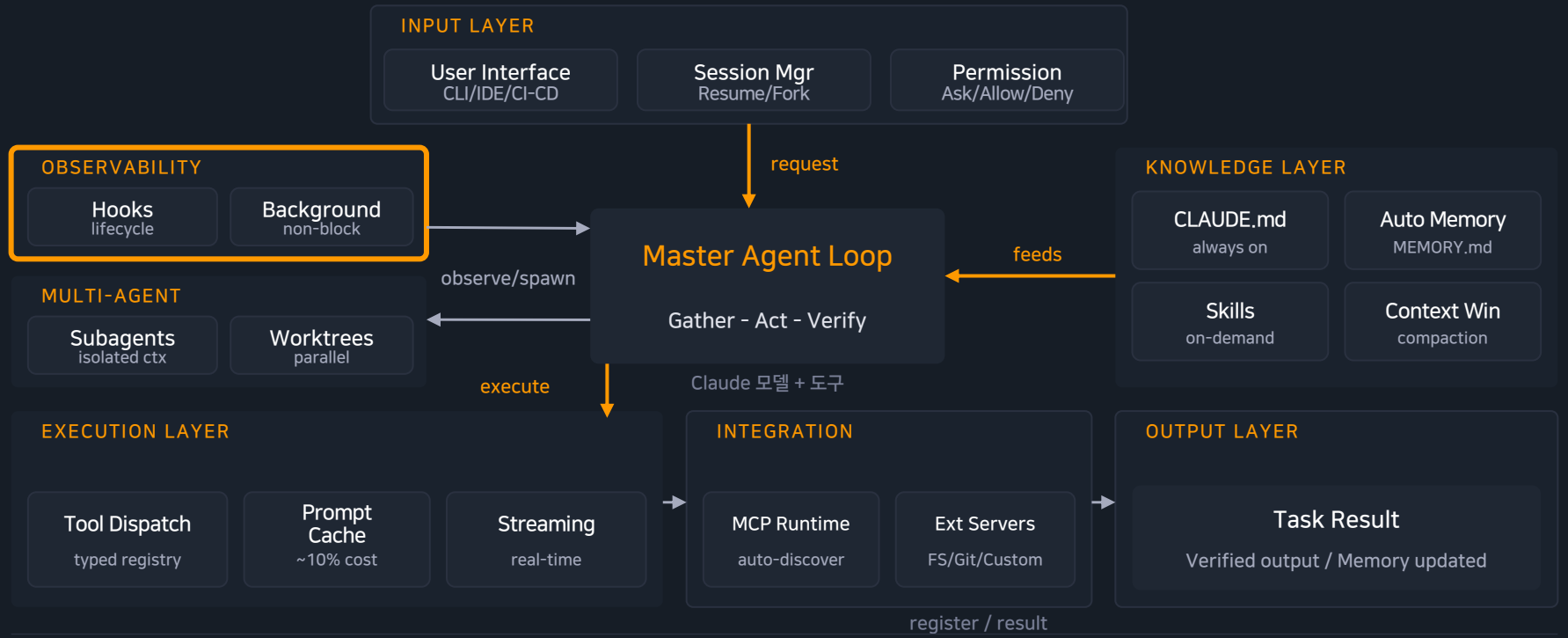
/agents
서브에이전트 구성

09.

Observability/ Integration Layer

전체 아키텍처 맵

Master Agent Loop을 중심으로 한 여덟 개 레이어



Observability Layer - Hooks

라이프사이클 이벤트로 워크플로를 자동화/통제

EVENT-DRIVEN

Hooks는 Claude Code의 특정 시점에 실행되는 셸 명령 또는 HTTP 핸들러.

이벤트가 발생하면 JSON 컨텍스트가 전달되고, 핸들러는 검사 후 결정(허용/차단)을 반환할 수 있음.

세션 1회	SessionStart →	SessionEnd	
턴 1회	UserPromptSubmit →	Stop →	StopFailure
도구 호출마다	PreToolUse →	PostToolUse →	PostToolUseFailure

그 외 이벤트: SubagentStart / SubagentStop / PermissionRequest / Notification / PreCompact / PostCompact

---

3 cadence
세션/턴/도구 호출

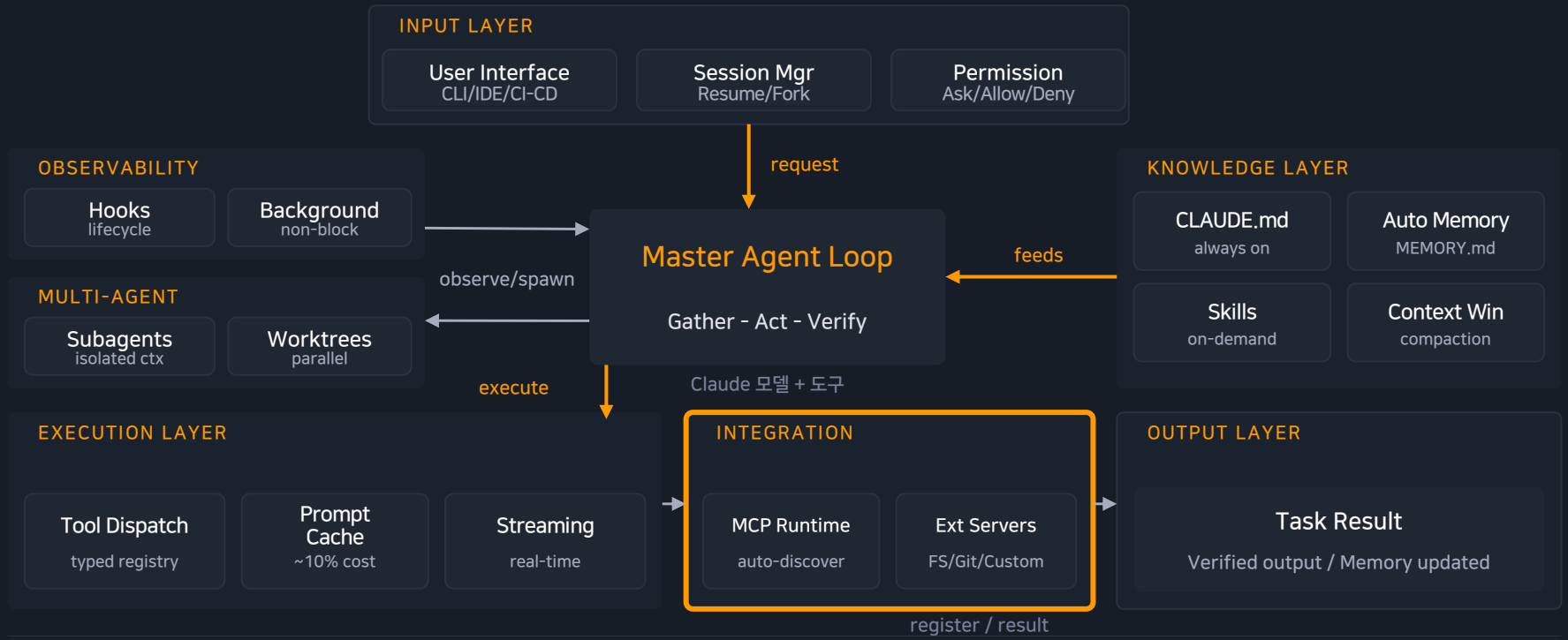
PreToolUse
실행 전 검증/차단

PostToolUse
포맷/테스트/로깅

12 events
전체 라이프사이클

전체 아키텍처 맵

Master Agent Loop을 중심으로 한 여덟 개 레이어

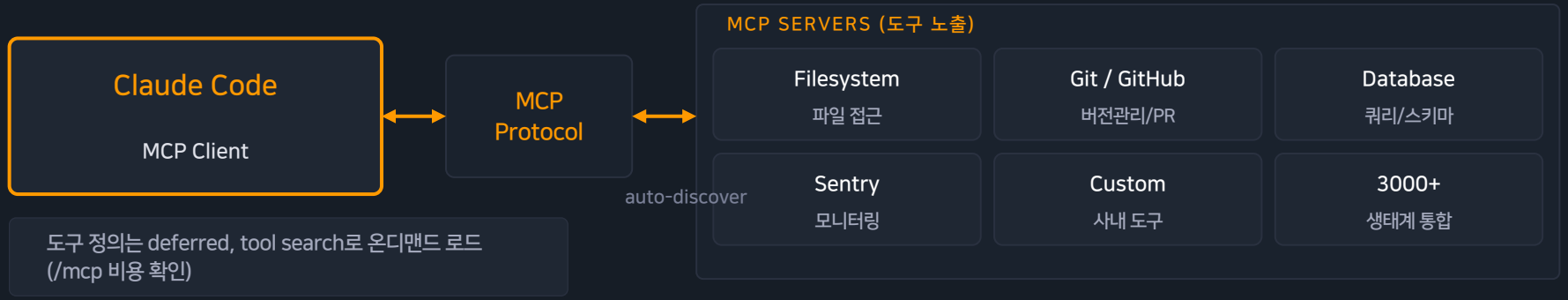


Integration Layer - MCP

외부 도구와 데이터를 표준 프로토콜로 연결

MODEL CONTEXT PROTOCOL

MCP는 AI 모델이 외부 도구와 데이터 소스에 접근하는 표준 프로토콜입니다. MCP 서버가 도구를 노출하면 Claude(MCP 클라이언트)가 이를 자동 발견해 호출합니다.



표준
도구/데이터 연결 프로토콜

auto-discover
서버 도구 자동 발견

3000+
통합 가능 서비스 생태계

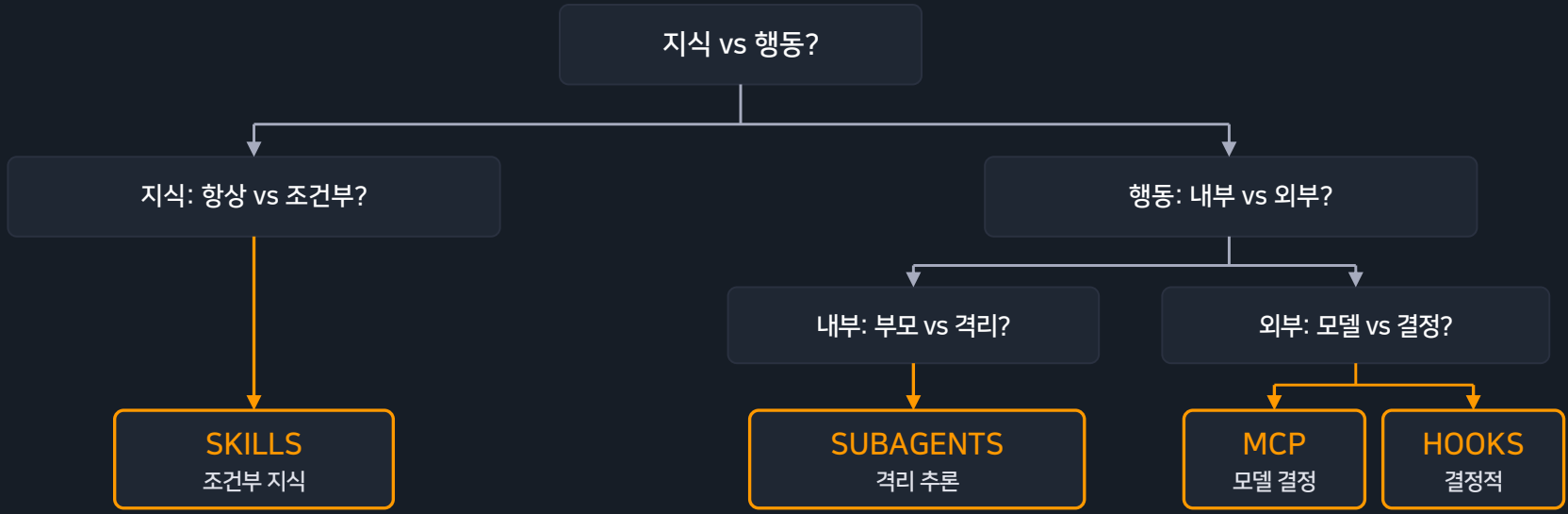
/mcp
서버별 컨텍스트 비용

10

프리미티브 선택

어떤 프리미티브를 쓸까

Skills / Subagents / MCP / Hooks - 결정 트리



항상 로드되는 지식은 CLAUDE.md, 부모 컨텍스트 내부 작업은 표준 도구 - 별도 프리미티브가 아닙니다

SKILLS
조건부 지식

SUBAGENTS
격리 추론/위임

MCP
외부 시스템 (모델 결정)

HOOKS
결정적 강제/자동화

Skills vs Subagents vs MCP vs Hooks

4가지 프리미티브 비교 매트릭스

	Skills	Subagents	MCP	Hooks
목적	조건부 작업 지식	격리된 추론 / 위임	모델이 호출하는 외부	결정적 강제 / 자동화
트리거	모델이 컨텍스트로 판단	모델이 명시적 위임	모델이 호출 시점 결정	이벤트 (PreToolUse 등)
실행 컨텍스트	같은 대화 컨텍스트	격리 샌드박스	외부 프로세스	로컬 훅 러너
실패 영향	낮음 (오용 시 환각)	중간 (격리 누수/부풀림)	높음 (네트워크/인증/장애)	높음 (흐름 차단/손상)
대표 예시	`pr-review-checklist`	`code-auditor`	`github, postgres`	`pre-tool-security-scan`

Low
Skills 지연

Medium
Subagents 지연

Med-High
MCP 지연

Low-Med
Hooks 지연

안티패턴과 설계 원칙

흔한 실패와 해결책, 그리고 프리미티브 설계 원칙

DESIGN PRINCIPLES

가장 단순한 프리미티브부터 시작하세요 (Skills, Subagents, MCP, Hooks 순).

컨텍스트는 가볍게, 경계는 명확하게, 강제는 결정적으로, 그리고 관찰하고 측정하며 개선.

SKILLS

부풀림 주의

거대 문서 덤프는 컨텍스트 부풀림.
작게, 정밀하게, 조건부로 유지

SUBAGENTS

격리/위임 절제

느슨한 격리와 과도한 위임 회피.
명확한 계약, 최소 입출력, 강한 스키마

MCP

외부 신뢰성

불안정 외부와 인증 실패, 도구 환각.
타임아웃, 재시도, 좋은 설명으로 보완

HOOKS

안전한 자동화

치명적 동작 차단과 무한 루프 주의.
안전 기본값, 멱등성, 철저한 테스트

단순함 우선
가장 단순한 것부터

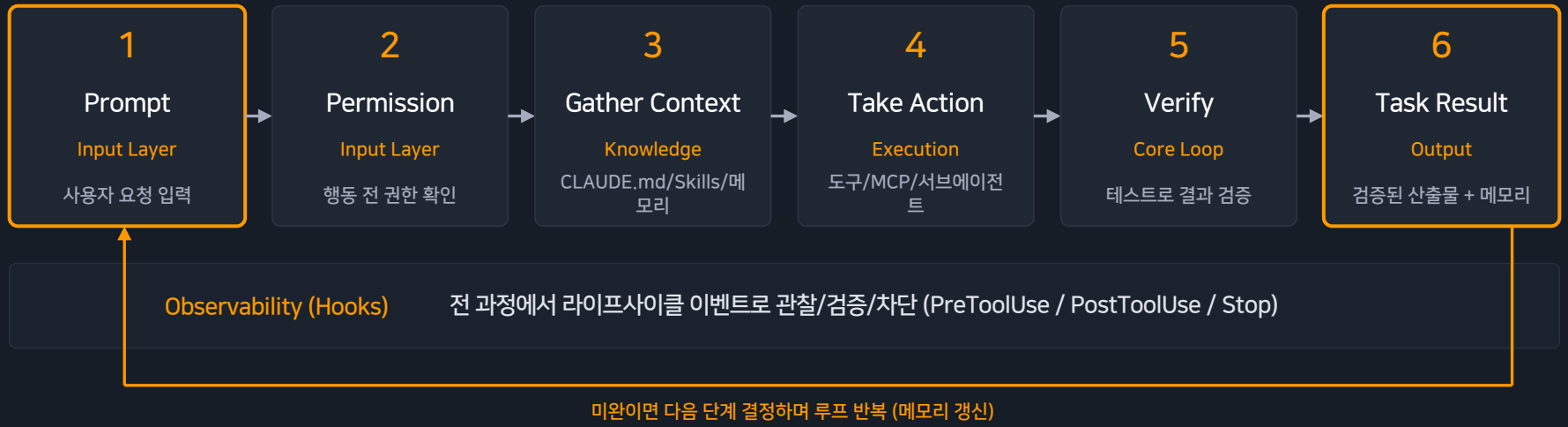
가벼운 컨텍스트
필요할 때만 로드

명확한 경계
입출력/계약 정의

결정적 강제
규칙은 Hooks로

종합 플로우 - End to End

요청에서 검증된 결과까지, 전체 레이어를 한 흐름으로



Input-Output
단일 에이전틱 흐름

feeds
지식이 매 단계 공급

hooks
전 과정 관측/통제

verified
검증된 결과 + 메모리

핵심 요약

여덟 개 레이어, 하나의 루프로 기억하기

CORE
Agentic Loop
수집-행동-검증을 적응적으로 반복

INPUT
진입/세션/권한
어디서나 같은 루프, 명시적 권한 통제

KNOWLEDGE
컨텍스트/메모리
CLAUDE.md/메모리/Skills + 자동 압축

EXECUTION
도구 실행/캐시
tool_use 디스패치, 캐시/스트리밍 최적화

MULTI-AGENT
서브에이전트
위임과 컨텍스트 격리, 병렬/실패 격리

OBSERVABILITY
Hooks
라이프사이클 이벤트로 결정론적 통제

INTEGRATION
MCP
표준 프로토콜로 외부 도구/데이터 연결

OUTPUT
Task Result
검증된 산출물 + 갱신된 메모리

---

1 loop
수집-행동-검증

2 축
모델 + 도구

4 확장
Skills/MCP/Hooks/Subagents

사람 통제
권한 + 체크포인트

References

공식 문서와 출처

ANTHROPIC DOCS

Claude Code 공식 문서

code.claude.com/docs - 작동 원리, Memory, Skills, Hooks, Sub-agents, MCP / Prompt Caching

SKILL STANDARD

SKILL.md 오픈 표준

agentskills.io - 스킬 규격, 프론트매터, 디렉터리 구조, 도구 호환성

PROTOCOL

Model Context Protocol

modelcontextprotocol.io - MCP 아키텍처, 도구, 인증 규격

Thank you!

Choi, WooHyung / Prin. Solutions Architect / AWS Korea

whchoi@amazon.com
https://linkedin.com/in/woohyungchoi