M07-03 · GIBALJA VISUAL MANUAL

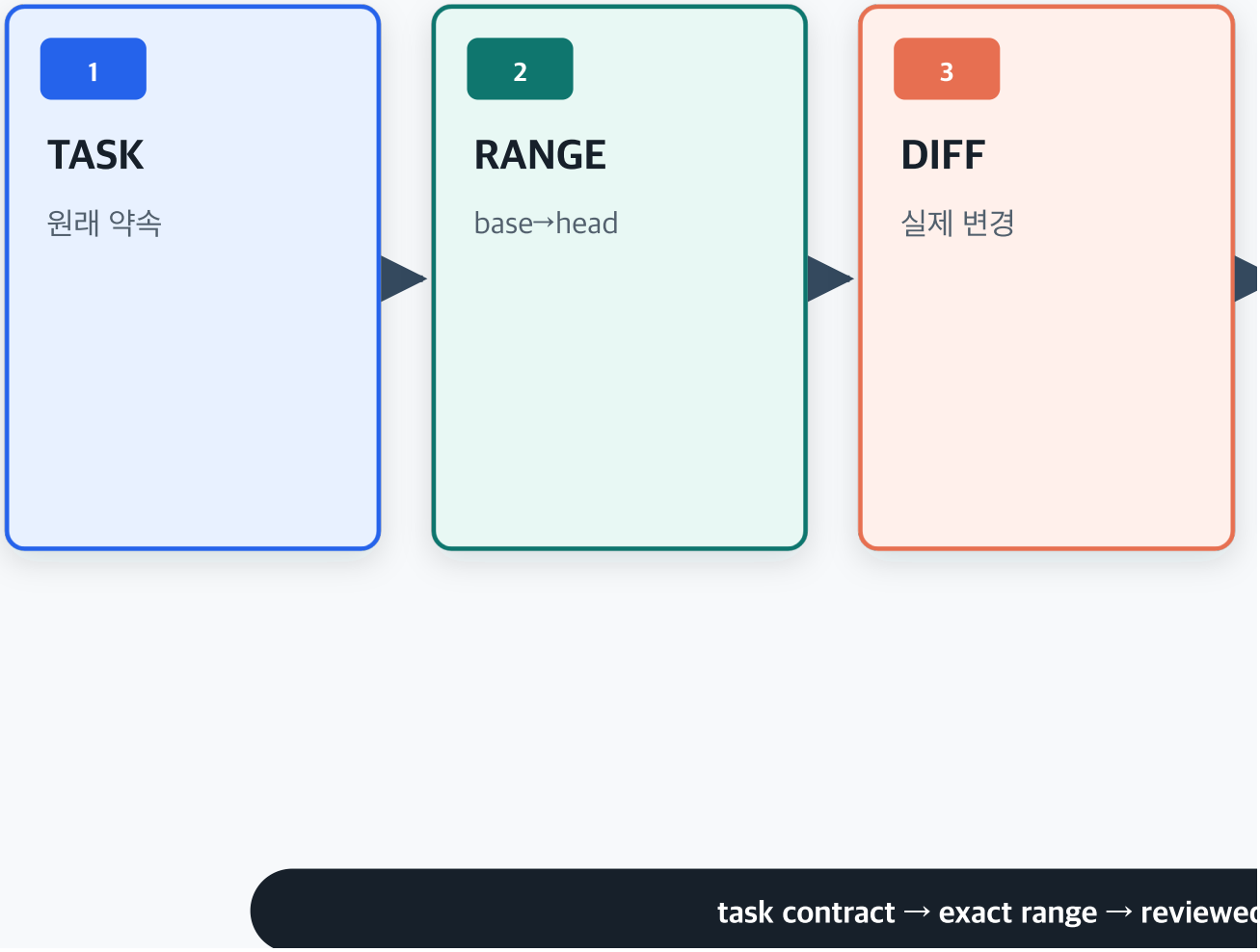
AI 생성 코드를 검토하고 되돌리기

> **한 문장 목표:** 원래 약속 → 정확한 변경 범위 → 실제 `diff` → 독립 검증 → 재현 가능한 `finding` → 승인 판정 → 격리된 `rollback rehearsal` 로 연결해, AI의 "완료" 설명이 아니라 현재 revision의 증거로 변경을 판단합니다.

난이도	개념	실습	셀프 테스트	최종 산출물
Level 1	60분	60분	15분	변경 검토 기록, finding 매트릭스, rollback·recovery 계획, 승인 전 checklist

AI가 코드를 빠르게 만들수록 사람의 일은 줄어드는 것이 아니라 **검토 질문이 더 선명해져야** 합니다. 좋은 reviewer는 모든 줄을 외우는 사람이 아닙니다. 원래 약속과 정확한 변경 범위를 고정하고, 위험한 흐름을 먼저 추적하고, 다른 사람이 같은 판정을 재현할 증거를 남기는 사람입니다.

그림 1. AI의 설명은 검토를 시작하는 안내일 뿐입니다. 최종 판정은 task·range·diff·독립 check·finding·복구 증거의 사슬에서 나옵니다.


1
TASK
원래 약속
2
RANGE
base→head
3
DIFF
실제 변경
task contract → exact range → reviewed

d surface → evidence → review decision

0. 이 PDF를 공부하는 방법

1회차 · 그림만 읽기 · 20분

그림 1부터 그림 14까지 제목과 결론 띠만 읽습니다. 다음 일곱 문장을 소리 내어 말할 수 있으면 됩니다.

```
AI summary는 navigation이지 evidence가 아니다.
검토는 원래 task contract와 exact base·head에서 시작한다.
diff는 줄의 변화이고 behavior 전체는 호출·data·config까지 따라가야 보인다.
후보가 만든 test는 독립 oracle로 다시 확인한다.
좋은 finding은 claim·location·condition·impact·evidence·action을 연결한다.
resolved 표시가 아니라 새 head와 current evidence로 재검토한다.
코드 revert와 외부 data·service recovery는 서로 다른 절차다.
```

2회차 · 위험이 심어진 실습 저장소 만들기 · 10분

AI 변경 검토 실습 생성기를 실행합니다.

```
./02_Labs/G07_AI_Spec/L07-03_create-ai-change-review-practice.sh
```

생성기는 외부 package 설치와 network 없이 작은 Node.js repository를 만듭니다. baseline commit과 AI candidate commit, 후보 test, 독립 contract test, review audit, rollback rehearsal을 함께 생성합니다. 같은 target이 이미 있으면 exit code 2로 멈추며 덮어쓰지 않습니다.

3회차 · 12개 검토 장면 판독 · 35분

AI 변경 검토·복구 데스크를 엽니다. 해설을 보기 전에 매 장면에서 다음을 말합니다.

```
원래 task contract =
base / head =
changed surface와 scope deviation =
가장 큰 behavior·safety 위험 =
실행한 독립 evidence =
finding과 severity =
판정과 recovery limit =
```

4회차 · 내 변경에 적용 · 70분

AI 변경 검토 기록 양식을 채우고 단계별 실습서에 따라 독립 검증과 rollback rehearsal을 수행합니다. 낯선 용어는 AI 변경 검토·복구 용어집에서 찾습니다.

1. AI 생성 코드에는 왜 별도의 검토 습관이 필요한가

AI가 만든 코드도 다른 코드와 같은 품질 기준을 적용받습니다. 다만 생성 속도와 변경량, 그럴듯한 설명 때문에 사람이 놓치기 쉬운 패턴이 있습니다.

보이는 신호	실제로 확인해야 할 것	놓쳤을 때의 위험
“요청을 구현했습니다”	원래 acceptance가 모두 충족됐는가	다른 문제를 잘 해결한 코드
“test가 모두 통과합니다”	test가 독립 oracle을 쓰는가	구현과 test가 같은 오류를 공유
“작은 변경입니다”	manifest·lock·config·generated file도 바뀌었는가	숨은 공급망·운영 변화
“보안을 개선했습니다”	권한의 negative case와 data sink를 확인했는가	인증을 인가로 착각
“쉽게 되돌릴 수 있습니다”	repository tree 외 data·message·service도 복구되는가	코드만 돌아오고 외부 영향은 남음
“comment를 해결했습니다”	새 head에서 regression check를 다시 했는가	표시만 resolved, 위험은 잔존

1.1. 설명과 증거를 분리합니다

AI summary는 다음 용도로 유용합니다.

- 변경 의도와 후보 파일을 빠르게 찾기
- 실행했다고 주장하는 command를 확인하기
- 알려진 제한과 후속 작업을 찾기

하지만 summary는 다음을 증명하지 않습니다.

- 실제 diff가 summary와 같은지
- command가 현재 head에서 실행됐는지
- test가 원래 contract를 검증하는지
- 삭제된 보호 test나 새 dependency가 없는지
- rollback이 실제로 재현되는지

REVIEW QUALITY

원래 약속 × 정확한 범위 × 위험 중심 판독 × 독립 증거 × 복구 가능성

한 요소가 0이면 설명이 매끄러워도 승인 근거가 되지 않습니다.

1.2. reviewer의 역할은 다시 구현하는 것이 아닙니다

reviewer는 작성자의 모든 선택을 자신의 취향으로 바꾸지 않습니다. 다음 질문에 답합니다.

```
이 변경은 원래 문제를 올바르게 해결하는가?
기존 동작·contract·안전 경계를 불필요하게 악화시키지 않는가?
증거가 현재 revision과 연결되는가?
실패했을 때 제한된 범위에서 회복할 수 있는가?
```

30초 확인 1

후보 test가 모두 green이라는 사실과 원래 task가 충족됐다는 사실은 같은가요? 아니라면 둘 사이를 연결할 독립 evidence 한 가지를 적습니다.

2. 검토 전에 원래 Task Contract를 고정합니다

실습의 원래 약속은 단순합니다.

```
actor:
  operator
outcome:
  synthetic review queue에서 stable reason code로 case를 필터한다.
must preserve:
  viewer는 접근할 수 없다.
  response와 log에 contact data를 노출하지 않는다.
allowed files:
  src/api/list-review-queue.js, test/review-filter.test.js
dependency:
  새 runtime dependency 없음
```

AI candidate는 편의를 이유로 email을 반환하고, display label로 filter하며, 색상 package를 추가했습니다. 구현이 작동하더라도 원래 약속과 다르면 승인할 수 없습니다.

2.1. 검토 기준의 우선순위

우선순위	기준	예
1	repository·조직의 강제 정책	보안·data·required checks
2	승인된 task contract	outcome·scope·acceptance
3	기존 public contract	API·schema·CLI behavior
4	현재 구현과 test	baseline behavior
5	AI의 summary와 추론	탐색용 가설

상위 기준과 하위 기준이 충돌하면 상위를 따릅니다. 기준 자체가 충돌하거나 owner가 없으면 추측하지 말고 `BLOCKED` 또는 `ASK`로 남깁니다.

2.2. Acceptance를 review question으로 바꿉니다

acceptance	reviewer question	evidence
operator가 stable code로 filter	label이 아니라 code를 비교하는가	independent boundary test
viewer 접근 금지	role negative case가 유지되는가	403 contract test
contact data 금지	response·log sink에 email이 없는가	DTO·log capture test
dependency 추가 금지	manifest와 lock이 그대로인가	base→head manifest diff
기존 protection 유지	보안 test를 삭제·약화하지 않았는가	test deletion audit

2.3. 명세가 없을 때

명세가 없다고 기존 코드가 자동으로 정답은 아닙니다. 다음 최소 packet을 요청합니다.

```
1. 원하는 actor와 observable outcome
2. 지켜야 할 public·security·data contract
3. 허용·금지 surface
4. positive·negative·boundary acceptance
5. required checks와 owner
```

이 packet 없이 review를 계속할 수는 있지만 판정은 `NOT REVIEWED`와 `UNKNOWN`을 포함해야 합니다. 모르는 것을 PASS로 바꾸지 않습니다.

3. Exact Base와 Head로 범위를 고정합니다

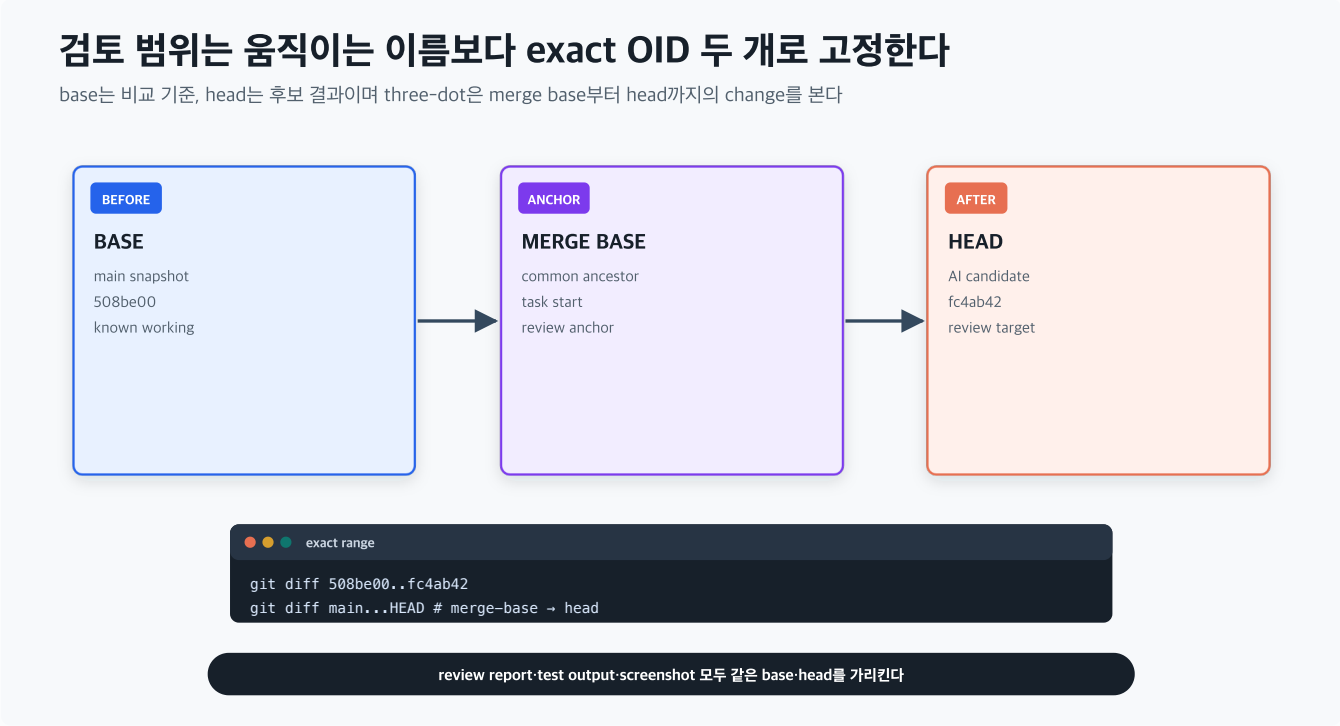
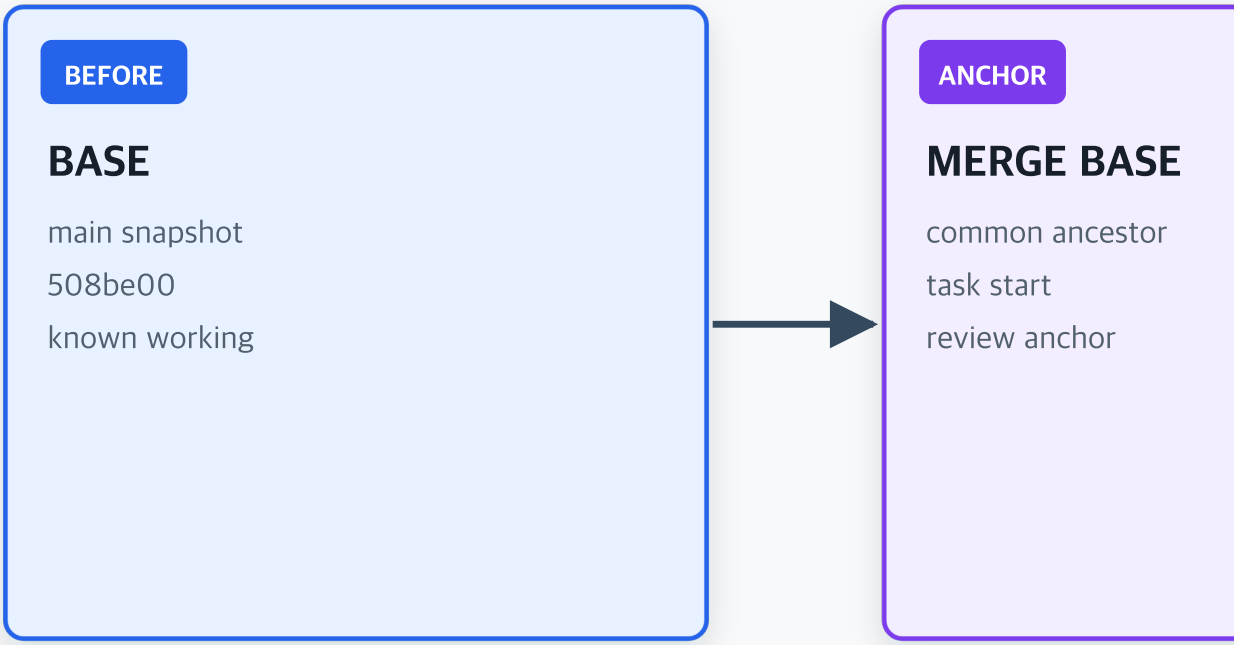
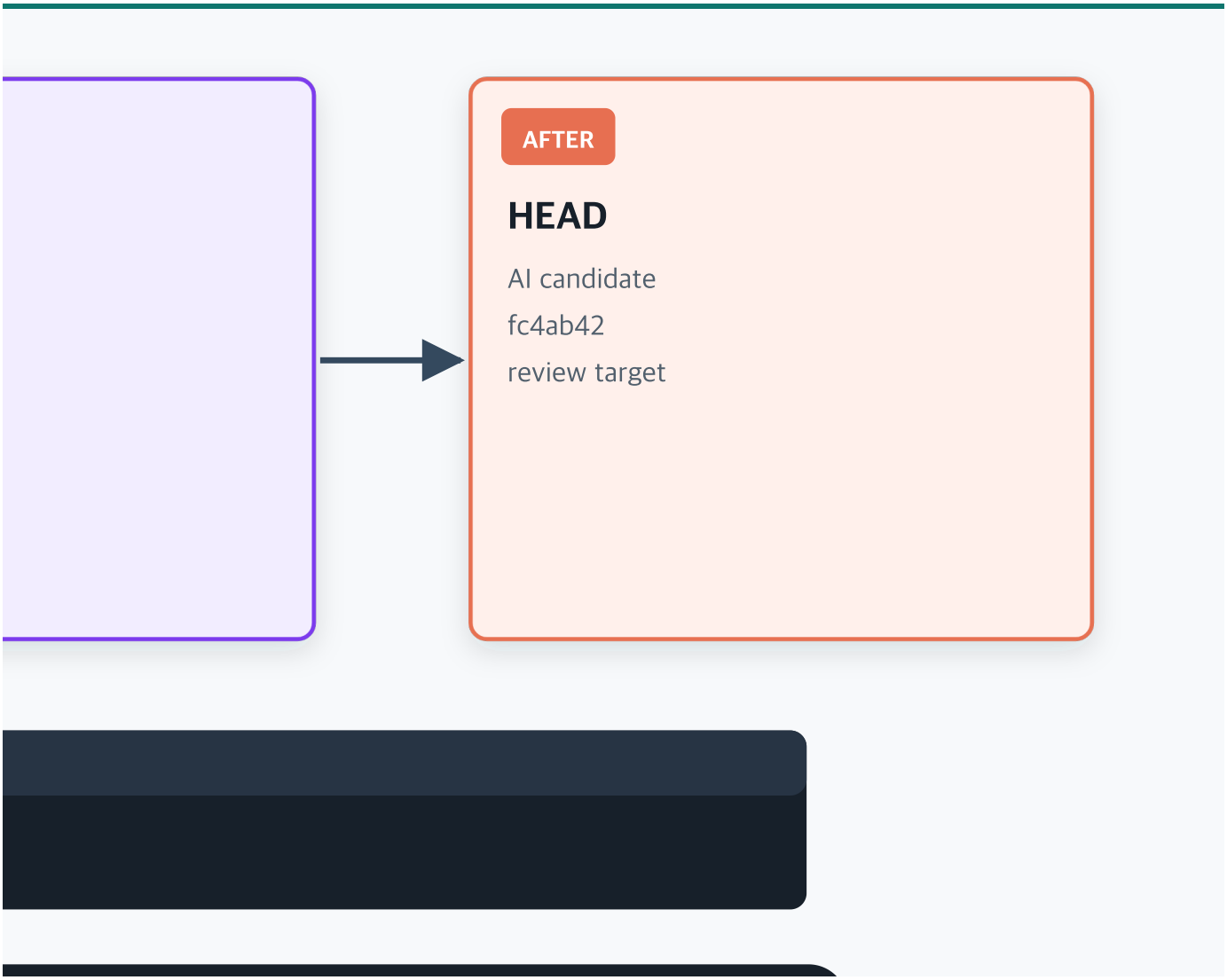


그림 2. “최근 변경”이 아니라 immutable base와 head를 기록해야 다른 사람이 같은 diff와 test 결과를 재현할 수 있습니다.



exact range

```
git diff 508be00..fc4ab42
git diff main...HEAD # merge-base → head
```


AFTER
HEAD
AI candidate
fc4ab42
review target

3.1. 세 가지 범위 질문

```
BASE: 무엇을 검증된 출발점으로 보는가?
HEAD: 정확히 어떤 후보 revision을 판정하는가?
DIRECTION: base에서 head로 무엇이 달라졌는가?
```

실습의 기준은 다음과 같습니다.

```
base = 508be0038d7012530546b915ac319174528001ef
head = fc4ab428c05d967ce407d07815612be70914e441
range = 508be00...fc4ab42
```

문서에는 사람이 읽기 쉬운 짧은 ID와 재현용 full ID를 함께 남깁니다.

3.2. 비교 명령을 기록합니다

```
git diff --stat 508be00 fc4ab42
git diff --name-status 508be00 fc4ab42
git diff 508be00 fc4ab42 -- src/api/list-review-queue.js
```

Git의 두 endpoint diff는 첫 tree와 두 번째 tree를 비교합니다. pull request처럼 branch 분기점 이후의 변경을 보려는 three-dot 비교는 merge base에서 head까지를 뜻합니다. 두 표기법의 의미를 섞지 말고 어떤 질문에 답하려는지 기록합니다.

3.3. Working tree를 별도로 봅니다

```
git status --short --branch
git diff
git diff --staged
```

commit range가 고정돼도 working tree에 미기록 변경이 있으면 실행 evidence가 어느 상태의 것인지 불명확해집니다. `CLEAN_WORKTREE PASS`는 코드 품질이 아니라 **증거 귀속**을 위한 gate입니다.

3.4. Head가 바뀌면 evidence도 만료됩니다

작성자가 fix를 push하면 이전 head의 test output은 새 head를 증명하지 않습니다. 다음 두 비교를 구분합니다.

비교	질문
old head → new head	finding을 어떻게 고쳤고 새 regression은 무엇인가
original base → new head	전체 task contract를 최종적으로 충족하는가

30초 확인 2

review comment를 resolved 처리한 시각보다 test evidence가 더 오래됐다면 current evidence입니까? 새 head ID와 함께 다시 판정합니다.

4. Changed Surface를 먼저 Inventory합니다

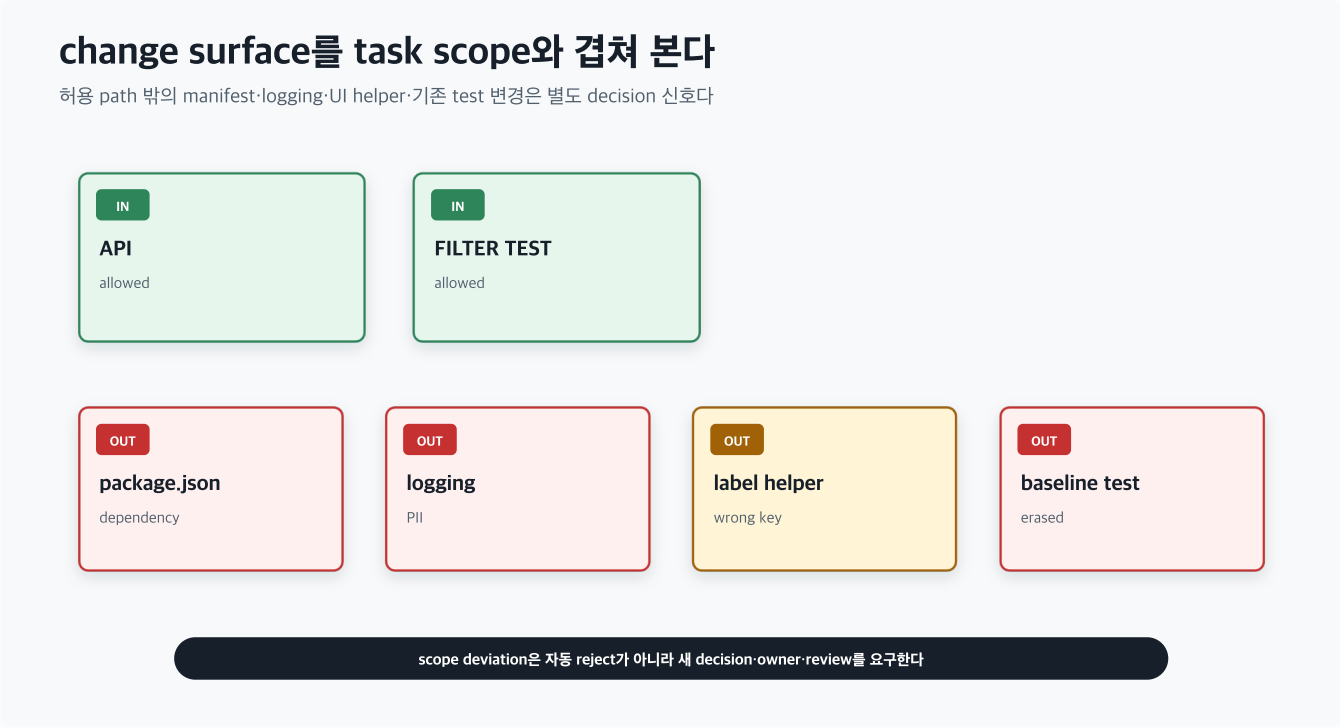
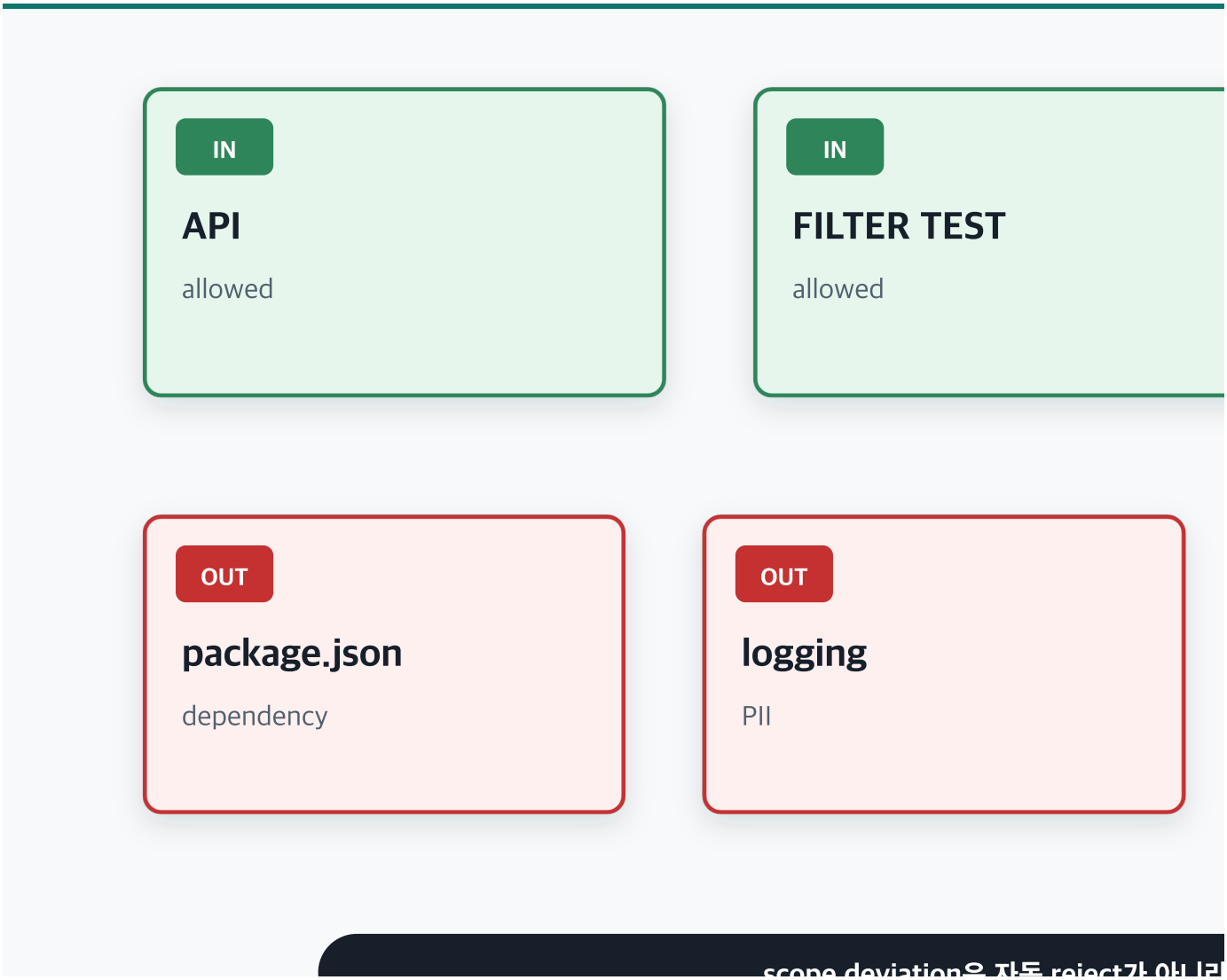


그림 3. 핵심 source file만 읽으면 더 위험한 주변 변화가 보이지 않습니다. 이름·상태·통계로 전체 surface를 먼저 펼칩니다.


IN
API
allowed
IN
FILTER TEST
allowed
OUT
package.json
dependency
OUT
logging
PII

OUT

label helper

wrong key

OUT

baseline test

erased

새 decision·owner·review를 요구한다

4.1. 첫 번째 pass는 읽기가 아니라 분류입니다

```
git diff --name-status 508be00 fc4ab42
git diff --stat 508be00 fc4ab42
```

실습 후보는 다섯 파일을 바꿉니다.

file	실제 변화	task 허용	첫 판정
`src/api/list-review-queue.js`	role check 약화·email 노출·label filter	YES	HIGH risk
`test/review-queue.test.js`	기존 보호 test 삭제·self-oracle 추가	NO	HIGH risk
`src/ui/reason-label.js`	display label helper 추가	NO	contract coupling
`src/logging/audit.js`	email log sink 추가	NO	HIGH risk
`package.json`	`kleur` runtime dependency	NO	dependency review

허용 파일이 두 개뿐인데 실제 변경이 다섯 개라면 먼저 scope deviation을 finding으로 기록합니다. 나머지 위험을 읽기 전에 이미 “설명 필요” 상태입니다.

4.2. 놓치기 쉬운 surface

- package manifest와 lockfile
- migration과 schema
- CI·deployment·permission config
- generated code와 binary
- logging·telemetry·analytics
- localization·docs·examples
- test fixture와 snapshot
- secret·credential·environment reference

4.3. 변경량은 위험의 대리 지표일 뿐입니다

한 줄의 권한 조건 삭제는 수백 줄의 순수 refactor보다 위험할 수 있습니다. `--stat`은 attention allocation에 도움을 주지만 severity를 자동 결정하지 않습니다.

4.4. 읽기 순서를 위험 중심으로 정합니다

권장 순서:

```
1. auth·permission·data·secret·external effect
2. public contract·schema·migration·dependency
3. core behavior와 error path
4. tests와 fixtures
5. refactor·naming·style·docs
```

이 순서는 작은 취향 comment에 시간을 쓰느라 blocker를 늦게 발견하는 것을 막습니다.

5. Diff의 문법과 의미를 함께 읽습니다

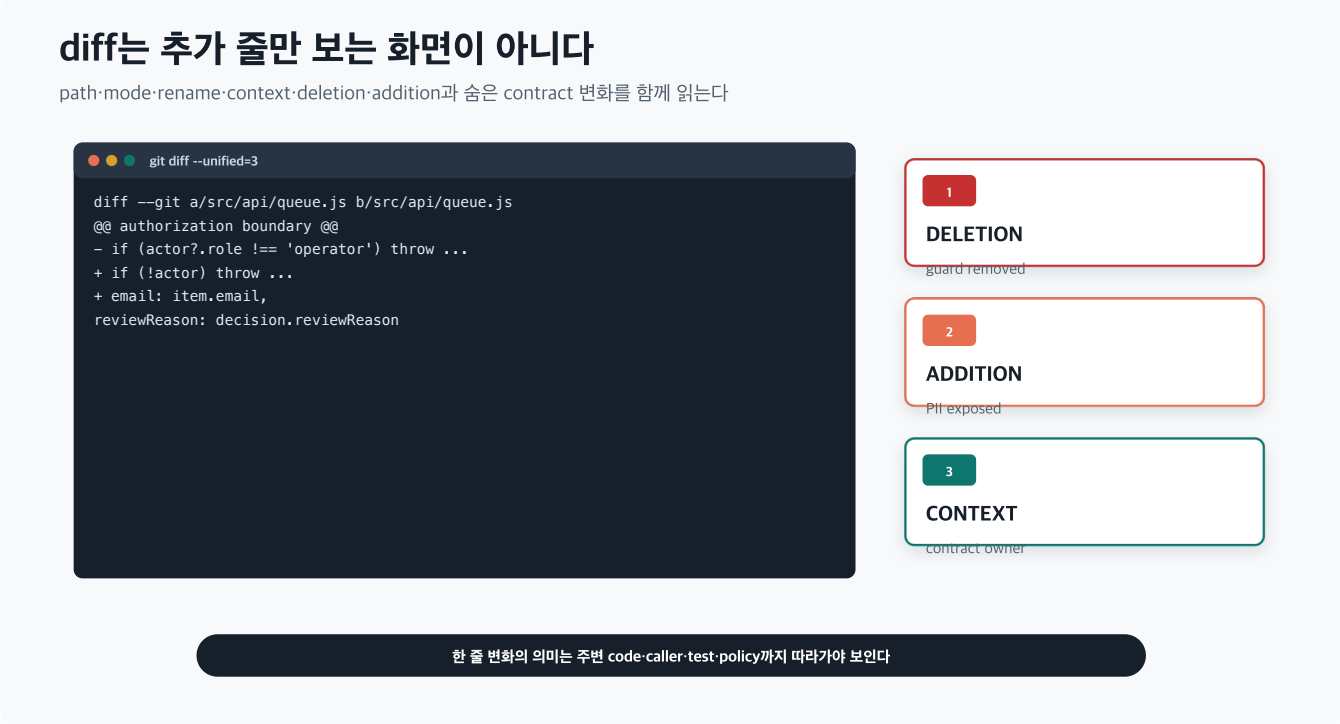
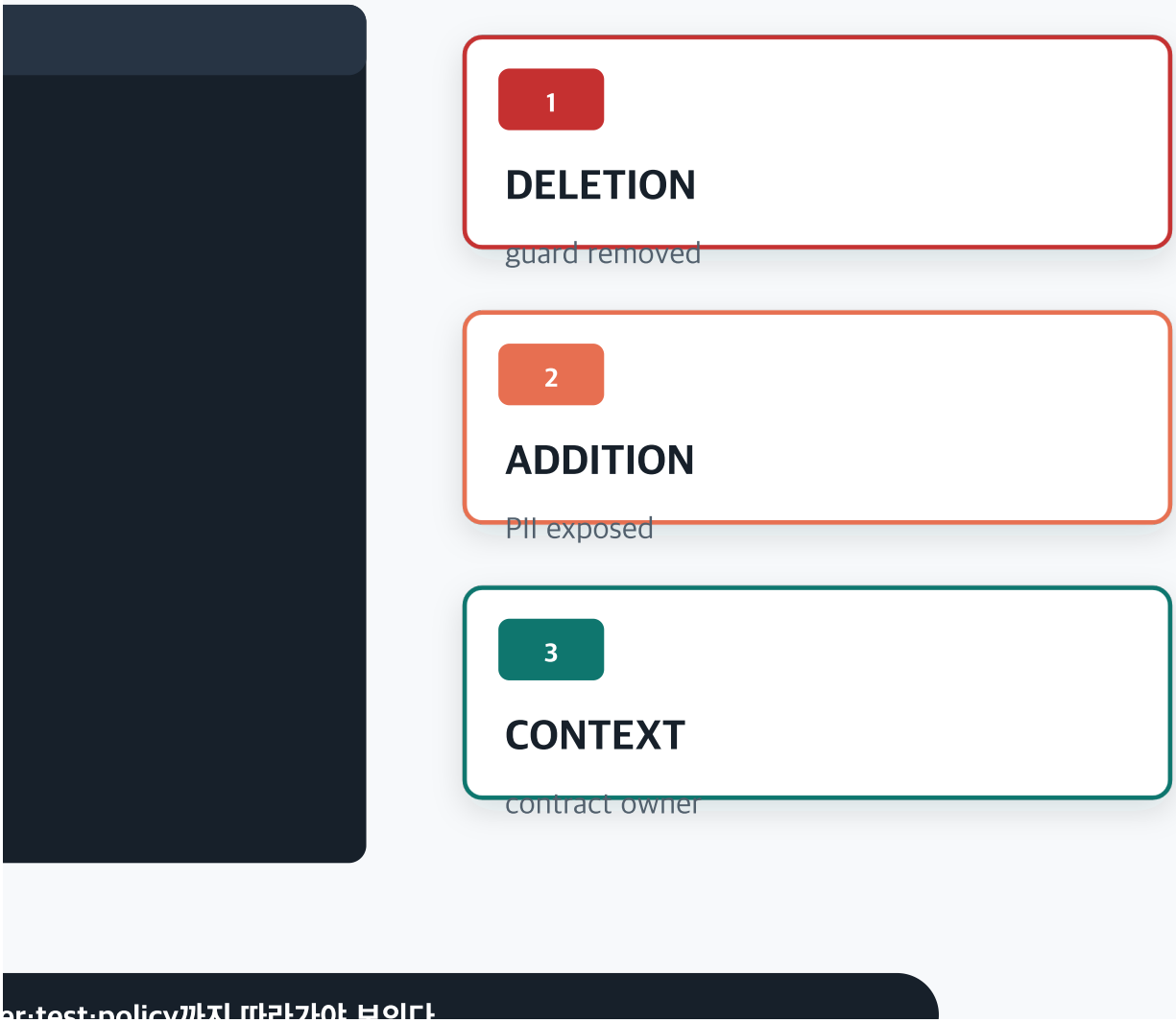


그림 4. diff의 줄 표시는 지도입니다. behavior는 변경 줄에서 시작해 호출자·data source·sink·test·config까지 따라가야 보입니다.

git diff --unified=3

```
diff --git a/src/api/queue.js b/src/api/queue.js
@@ authorization boundary @@
- if (actor?.role !== 'operator') throw ...
+ if (!actor) throw ...
+ email: item.email,
reviewReason: decision.reviewReason
```

한 줄 변화의 의미는 주변 code·call


1
DELETION
guard removed
2
ADDITION
PII exposed
3
CONTEXT
contract owner

5.1. Diff 문법

요소	읽는 내용	검토 질문
file header	old·new path, rename	책임과 공개 경계가 바뀌는가
hunk header	변경 위치와 context	어떤 함수·조건 안인가
- deletion	제거된 보호·contract	negative path가 사라지는가
+ addition	새 behavior·dependency	입력·출력·부작용은 무엇인가
unchanged context	주변 불변 가정	호출자가 이 변화를 어떻게 해석하는가

5.2. 줄 단위에서 의미 단위로 이동합니다

다음 변화는 한 줄처럼 보입니다.

```
- if (!actor || actor.role !== "operator") return forbidden();
+ if (!actor) return forbidden();
```

의미 질문은 다음과 같습니다.

```
새로 통과하는 actor는 누구인가? → viewer
그 actor가 읽는 data는 무엇인가? → review queue
어떤 field가 반환·기록되는가? → email
기존 negative test가 남아 있는가? → 삭제됨
영향 범위는 local인가 public endpoint인가? → endpoint 전체
```

5.3. Added line만 읽지 않습니다

삭제된 test, 제거된 validation, 축소된 error handling이 새 코드보다 더 중요한 경우가 많습니다. 특히 다음 삭제를 별도 inventory합니다.

- 권한·입력 검증
- negative·boundary test
- redaction·masking
- retry·timeout·cleanup
- schema constraint

- audit log

5.4. Rename·generated·binary는 다르게 다룹니다

rename은 내용 변경과 이동을 분리해서 봅니다. generated file은 source generator와 재생성 command를 확인합니다. binary는 diff만으로 판단할 수 없으므로 provenance·viewer·checksum·owner가 필요합니다.

6. 여덟 Lens로 같은 Diff를 다르게 읽습니다

그림 5. 한 reviewer가 모든 분야의 최종 owner일 필요는 없습니다. 다만 필요한 lens와 owner가 누구인지 빠뜨리지 않아야 합니다.

1

BEHAVIOR

question

evidence

2

CONTRACT

question

evidence

5

DATA·SECURITY

question

evidence

6

CONFIG·OPS

question

evidence

reviewer 한 명이 모든 owner를 대신하지

3

TEST

question

evidence

4

DEPENDENCY

question

evidence

7

SCOPE

question

evidence

8

ROLLBACK

question

evidence

않는다 · 필요한 lens owner를 호출한다

lens	핵심 질문	대표 evidence
Behavior	positive·negative·boundary에서 실제 결과가 맞는가	focused independent test
Contract	API·schema·CLI·event 의미가 호환되는가	contract diff·consumer test
Test	test가 독립 oracle이며 삭제·약화되지 않았는가	base→head test diff
Dependency	왜 필요한가, source·license·owner가 승인했는가	manifest·lock·source review
Data·Security	누가 무엇을 읽고 어느 sink로 보내는가	auth matrix·source-to-sink trace
Config·Operations	deploy·flag·timeout·observability가 안전한가	dry run·staging evidence
Scope	task 허용 범위와 같은가	allowed file·non-goal matrix
Rollback	code·schema·data·external effect를 어떻게 회복하는가	isolated rehearsal·runbook

6.1. Lens마다 질문과 evidence가 달라야 합니다

`npm test` 한 줄은 모든 lens의 증거가 아닙니다. dependency provenance, production permission, 외부 email 회수 가능성은 unit test로 증명되지 않습니다.

6.2. 전문 owner를 호출할 조건

다음 변화는 혼자 추정하지 말고 담당 owner를 포함합니다.

- 인증·인가 모델 변경
- 개인정보·민감 정보의 새 source·sink
- 결제·법적 기록·공공 데이터
- schema migration과 irreversible transform
- 새 runtime dependency·외부 SaaS
- deployment·network·secret 권한

6.3. `NOT REVIEWED` 도 유효한 상태입니다

모든 lens를 검토하지 못했다면 숨기지 않습니다.

```
DEPENDENCY: NOT REVIEWED · owner security-team · due 2026-07-17
LOAD TEST: NOT EXECUTED · staging permission unavailable
DATA RETENTION: UNKNOWN · policy owner requested
```

7. Behavior·Contract·Data Flow를 따라갑니다

7.1. 변경 줄에서 양방향으로 이동합니다

```
upstream:
  input source → validation → authorization → transform

changed logic:
  filter·branch·mapping·side effect

downstream:
  return value → caller → response·database·log·message
```

7.2. Stable code와 display label을 구분합니다

실습의 요구는 stable reason code입니다.

```
// contract-friendly
item.reasonCode === "LOW_CONFIDENCE"

// presentation-coupled
labelFor(item.reasonCode) === "확인 필요"
```

display label은 번역·문구 수정으로 바뀔 수 있습니다. filtering contract가 label에 의존하면 presentation 변경이 behavior를 깨뜨립니다.

7.3. 인증과 인가를 구분합니다

```
authentication: 이 actor는 누구인가?
authorization: 이 actor가 이 resource·action을 수행해도 되는가?
```

`actor != null`은 인증 여부만 암시합니다. operator 전용 queue라면 role·resource·action의 authorization이 필요합니다.

7.4. Error path도 public contract입니다

다음을 함께 봅니다.

- unauthorized와 unauthenticated status가 다른가

- 빈 queue와 server error를 구분하는가
- 민감 정보가 error message·stack·log에 포함되는가
- retry 가능한 오류와 영구 오류를 구분하는가
- partial failure에서 중복 부작용이 생기는가

30초 확인 3
성공 응답에서 email을 제거했지만 debug log에는 남아 있다면 data exposure finding이 해결됐습니까?
source에서 모든 sink까지 다시 추적합니다.

8. AI가 만든 Test는 독립 Oracle로 다시 봅니다

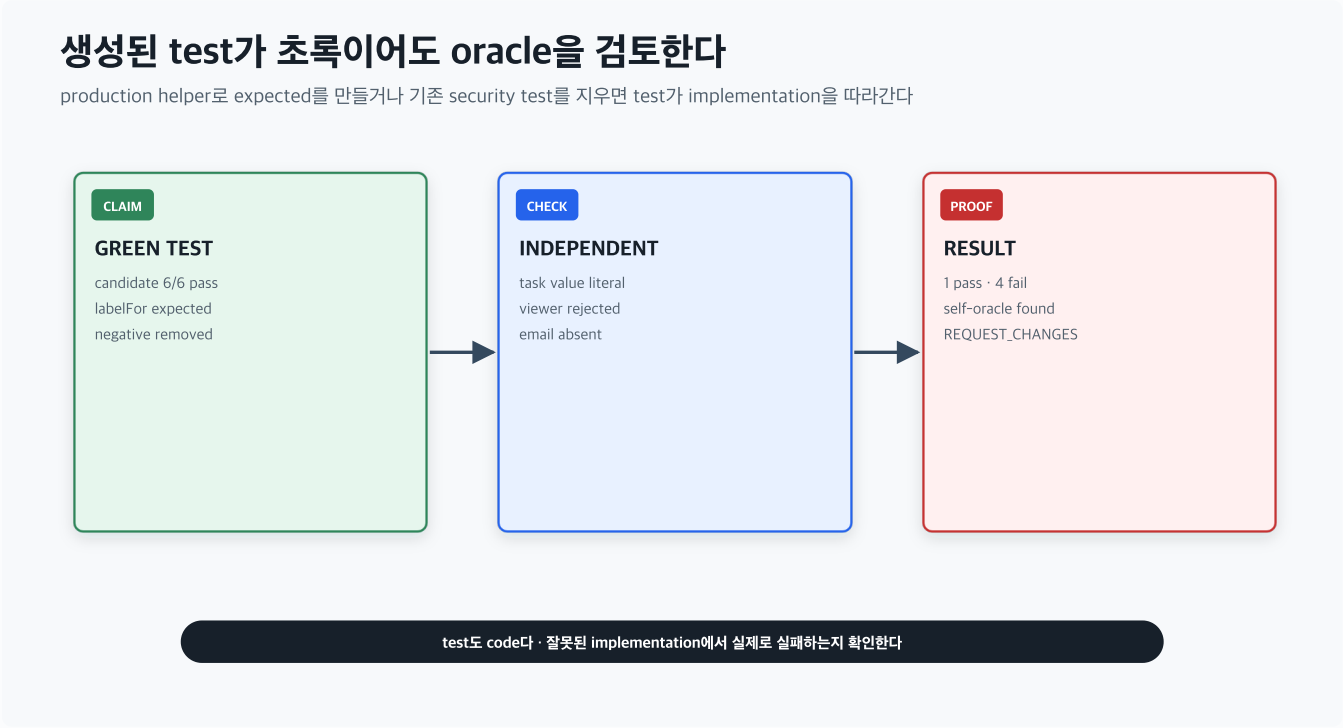
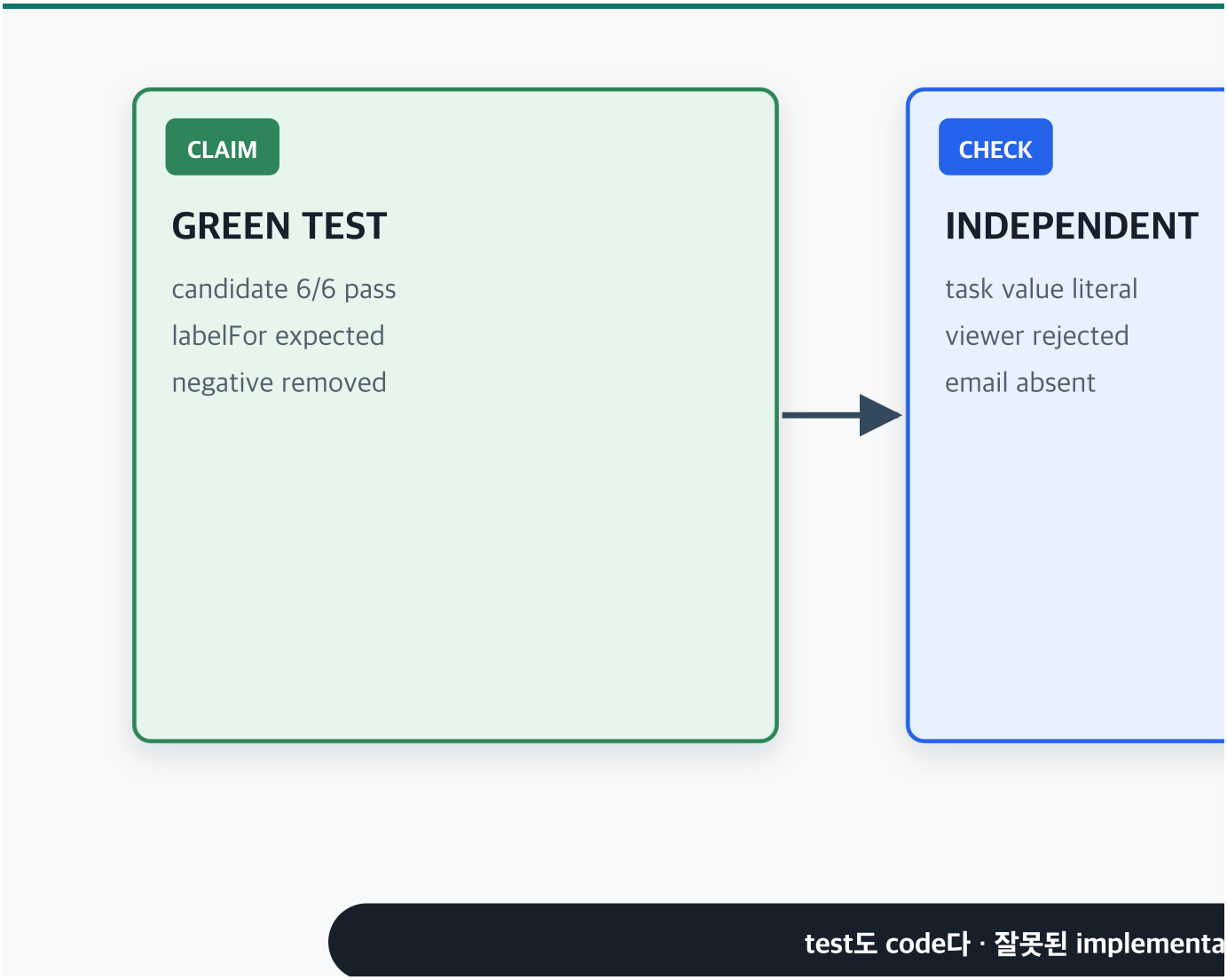
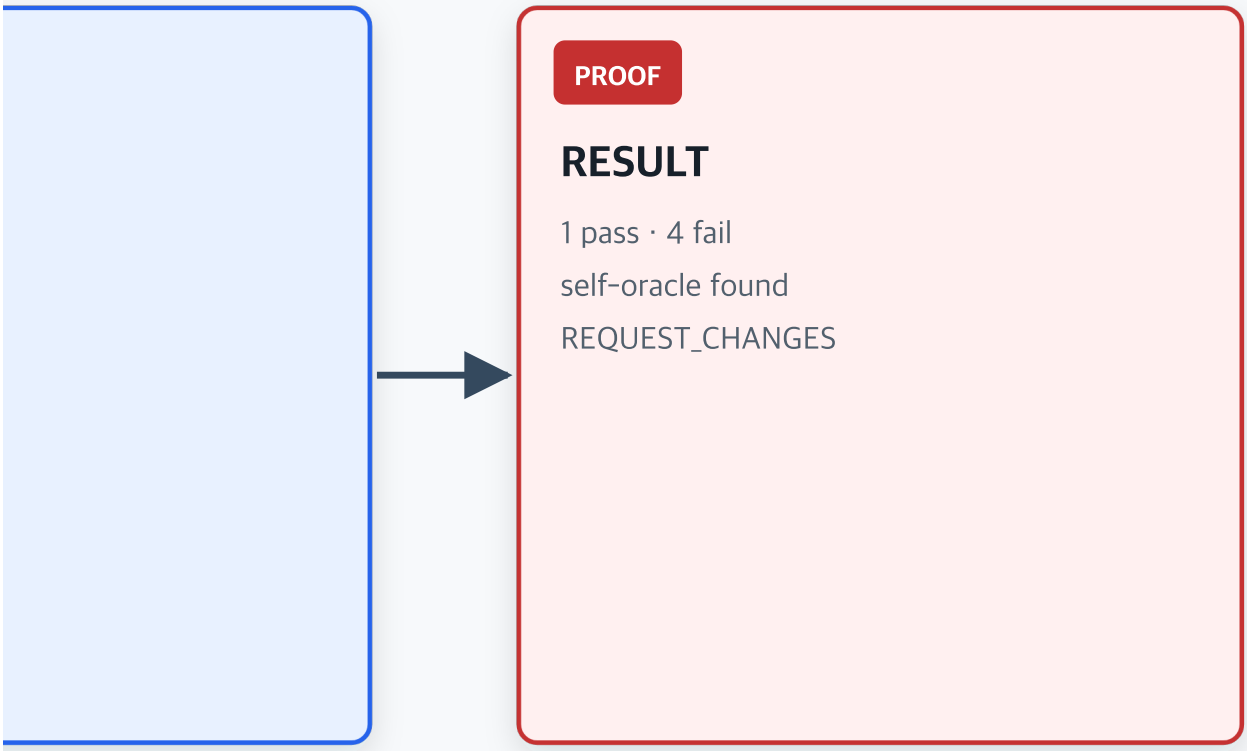


그림 6. 구현과 test가 같은 가정을 공유하면 둘 다 green이어도 틀릴 수 있습니다. 원래 contract에서 나온 독립 oracle이 필요합니다.


CLAIM
GREEN TEST
candidate 6/6 pass
labelFor expected
negative removed
CHECK
INDEPENDENT
task value literal
viewer rejected
email absent
test도 code다 · 잘못된 implementa



tion에서 실제로 실패하는지 확인한다

8.1. Self-oracle 함정

```
// production
const visible = items.filter(x => labelFor(x.reasonCode) === query);

// candidate test
assert.equal(result[0].label, labelFor(input.reasonCode));
```

test가 production의 `labelFor`를 정답으로 다시 호출합니다. label-based filter가 잘못됐어도 같은 오류를 반복하므로 통과합니다.

독립 test는 task contract의 stable code를 직접 선언합니다.

```
assert.deepEqual(filterByReason(items, "LOW_CONFIDENCE").map(x => x.id), ["case-1"]);
```

8.2. Test deletion은 source change와 같은 비중으로 봅니다

후보는 다음 보호 test를 삭제했습니다.

```
viewer receives 403
response omits email
logs omit email
```

삭제 이유가 behavior 제거인지, test 중복인지, 단순히 후보를 green으로 만들기 위한 것인지 확인합니다. 보호 contract가 여전히 유효하면 test를 복구하거나 더 강한 동등 evidence를 요구합니다.

8.3. Test가 test 자신을 검증하지는 못합니다

독립 검토 항목:

항목	질문
oracle	expected 값이 요구에서 왔는가 구현에서 왔는가
negative	금지 actor·input·state를 포함하는가
boundary	79/80, empty/max, before/after 같은 경계가 있는가
mutation sensitivity	구현을 의도적으로 틀리면 test가 실패하는가

항목	질문
deletion	base에 있던 보호 test가 사라졌는가
isolation	network·time·random·shared state에 흔들리지 않는가
revision	output이 현재 head에서 나온 것인가

8.4. Focused에서 broad로 검증합니다

```
1. 가장 작은 independent contract test
2. 영향받은 module test
3. repository required checks
4. 필요할 때 integration·security·performance check
```

큰 suite부터 돌려 실패 원인을 잃지 않습니다. 반대로 focused test만 통과했다고 전체 regression이 없다고 주장하지 않습니다.

9. Dependency 변경을 별도 Review합니다

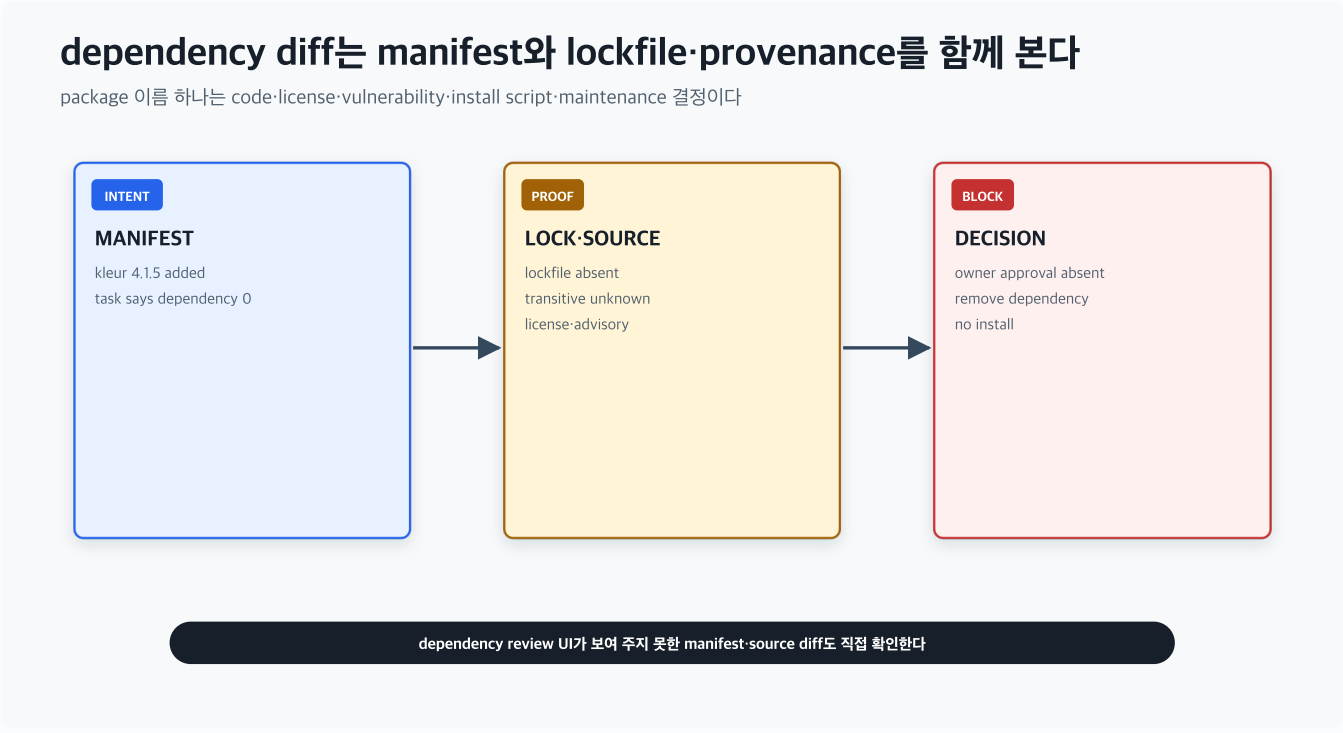
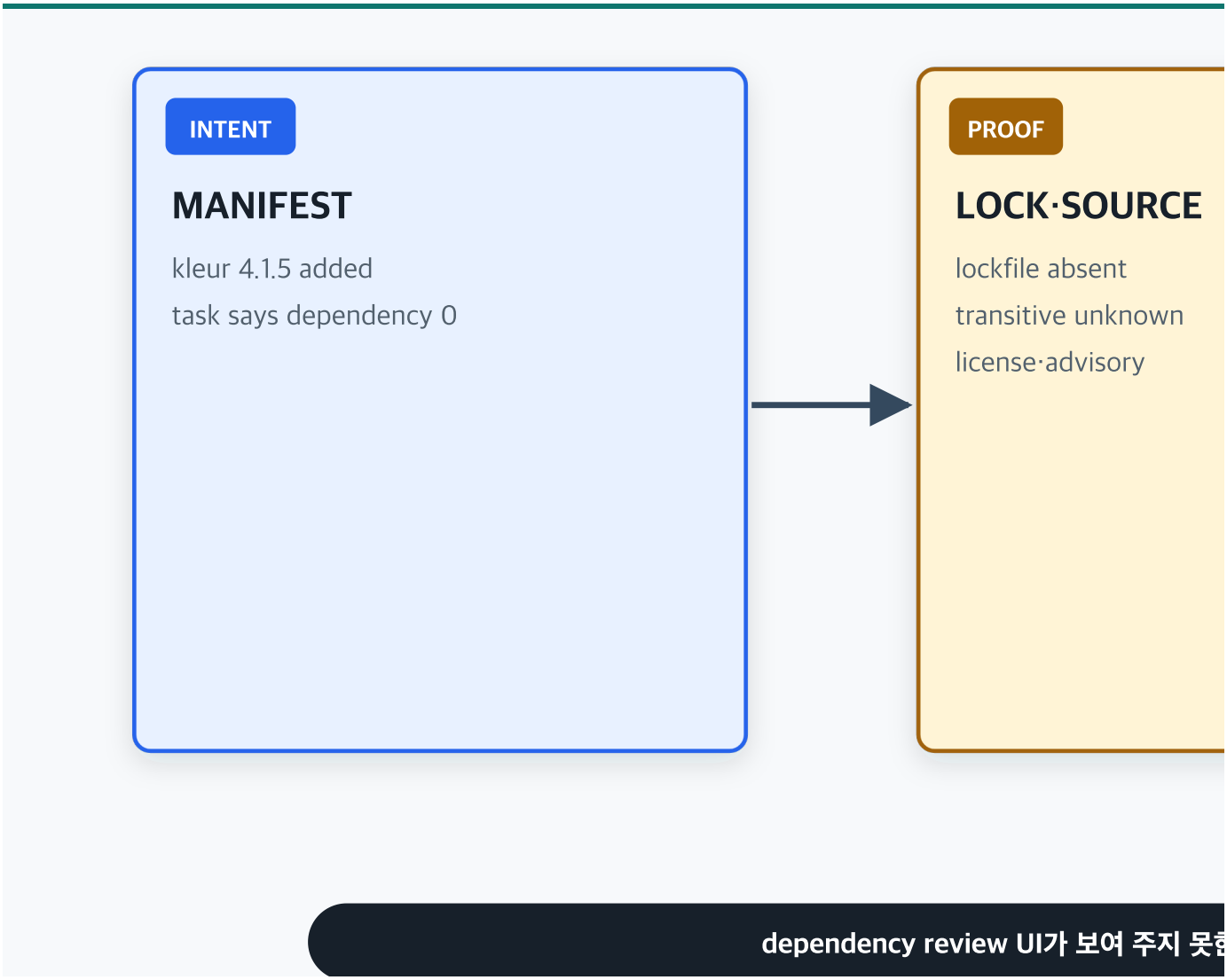
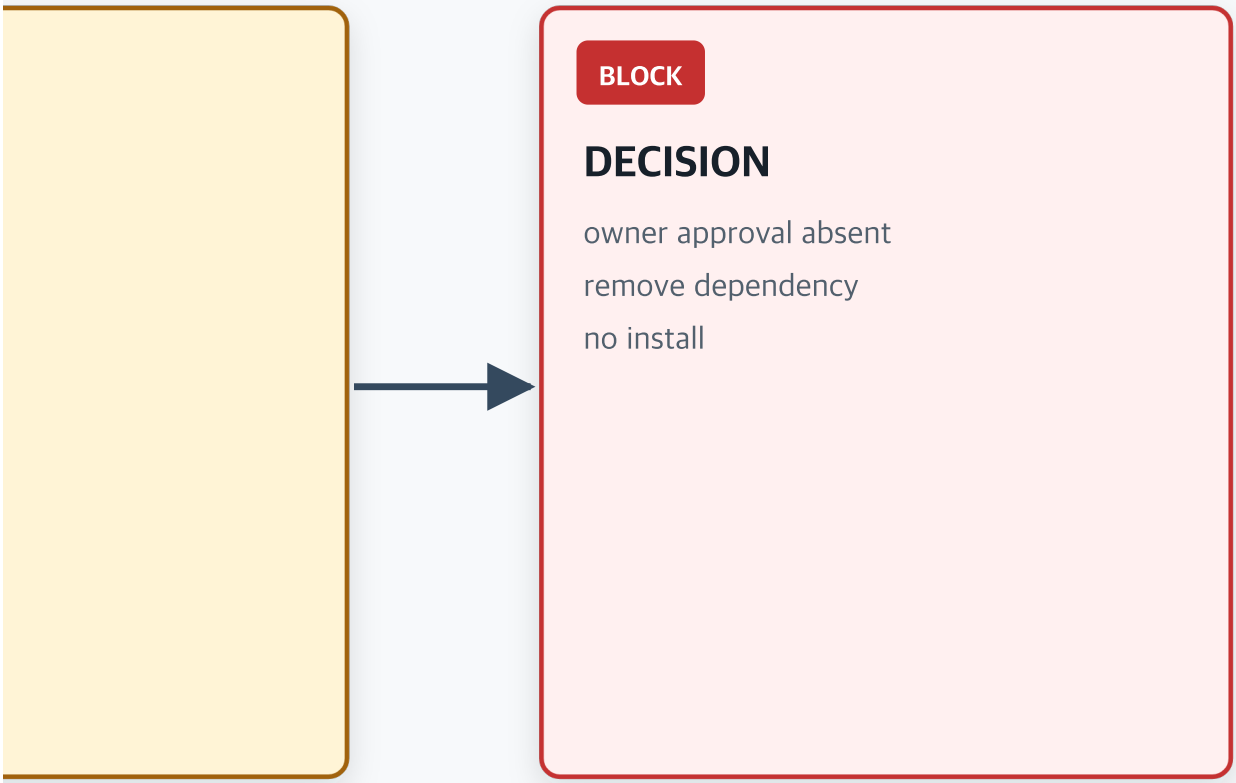


그림 7. dependency 이름과 version만 보는 것으로 끝나지 않습니다. 필요성부터 source change·license·유지 owner까지 연결합니다.


INTENT
MANIFEST
kleur 4.1.5 added
task says dependency 0
PROOF
LOCK·SOURCE
lockfile absent
transitive unknown
license·advisory
dependency review UI가 보여 주지 못



한 manifest·source diff도 직접 확인한다

9.1. 먼저 “왜 필요한가”를 묻습니다

실습 후보의 `kleur`는 출력 색상을 위한 runtime dependency입니다. 원래 task는 queue filtering이며 색상 출력이 없습니다. 필요성이 없으므로 공급망 조사 전에 제거 또는 별도 task 분리가 우선입니다.

9.2. Dependency review checklist

질문	기록
task outcome에 꼭 필요한가	YES / NO / UNKNOWN
기존 표준 library로 가능한가	대안
runtime인가 development-only인가	surface
manifest와 lock이 일치하는가	diff
direct·transitive 변화량은 얼마인가	count
source·publisher·provenance를 확인했는가	link·checksum
license·security 정책을 통과했는가	owner evidence
update·removal owner가 있는가	owner

9.3. 자동 alert는 검토를 대체하지 않습니다

취약점 scanner가 green이어도 필요 없는 dependency는 제거 대상일 수 있습니다. 반대로 alert가 있다고 모든 사용 경로가 즉시 exploit되는 것은 아닙니다. version·사용 경로·도달 가능성·대안을 evidence로 판정합니다.

10. 민감 Data의 Source에서 Sink까지 추적합니다

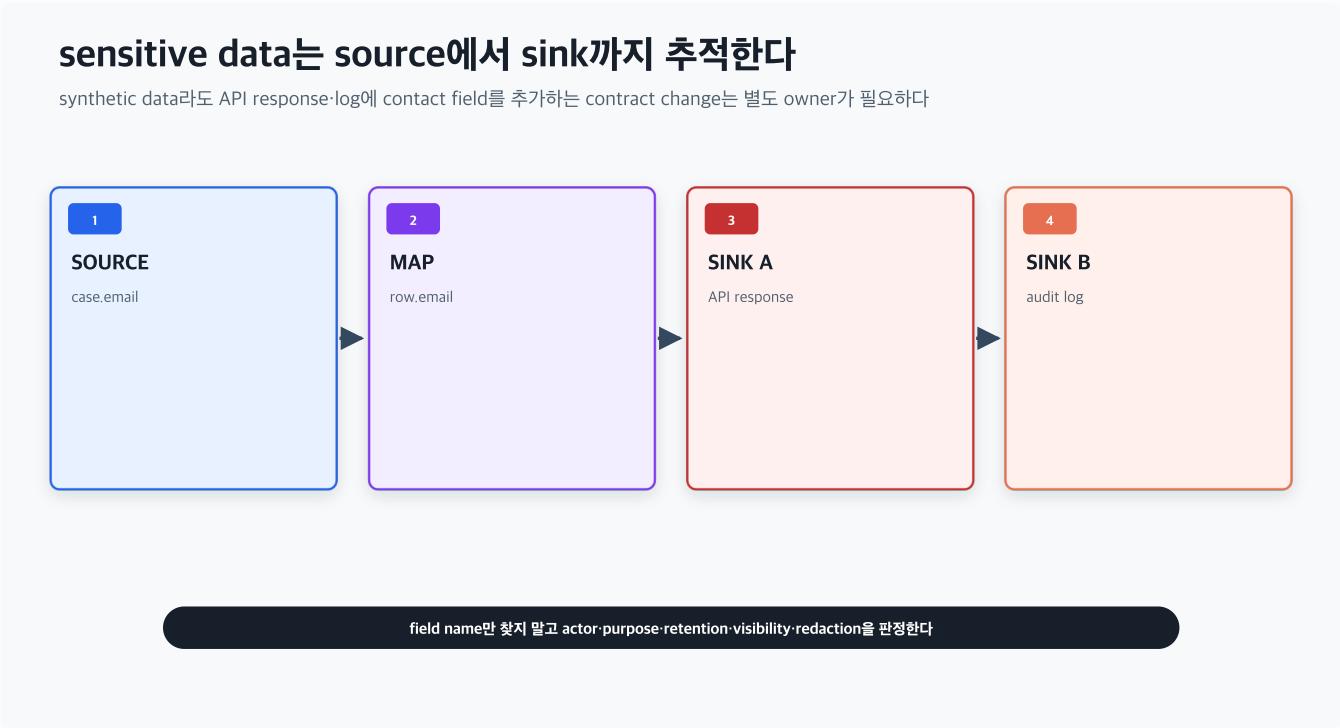
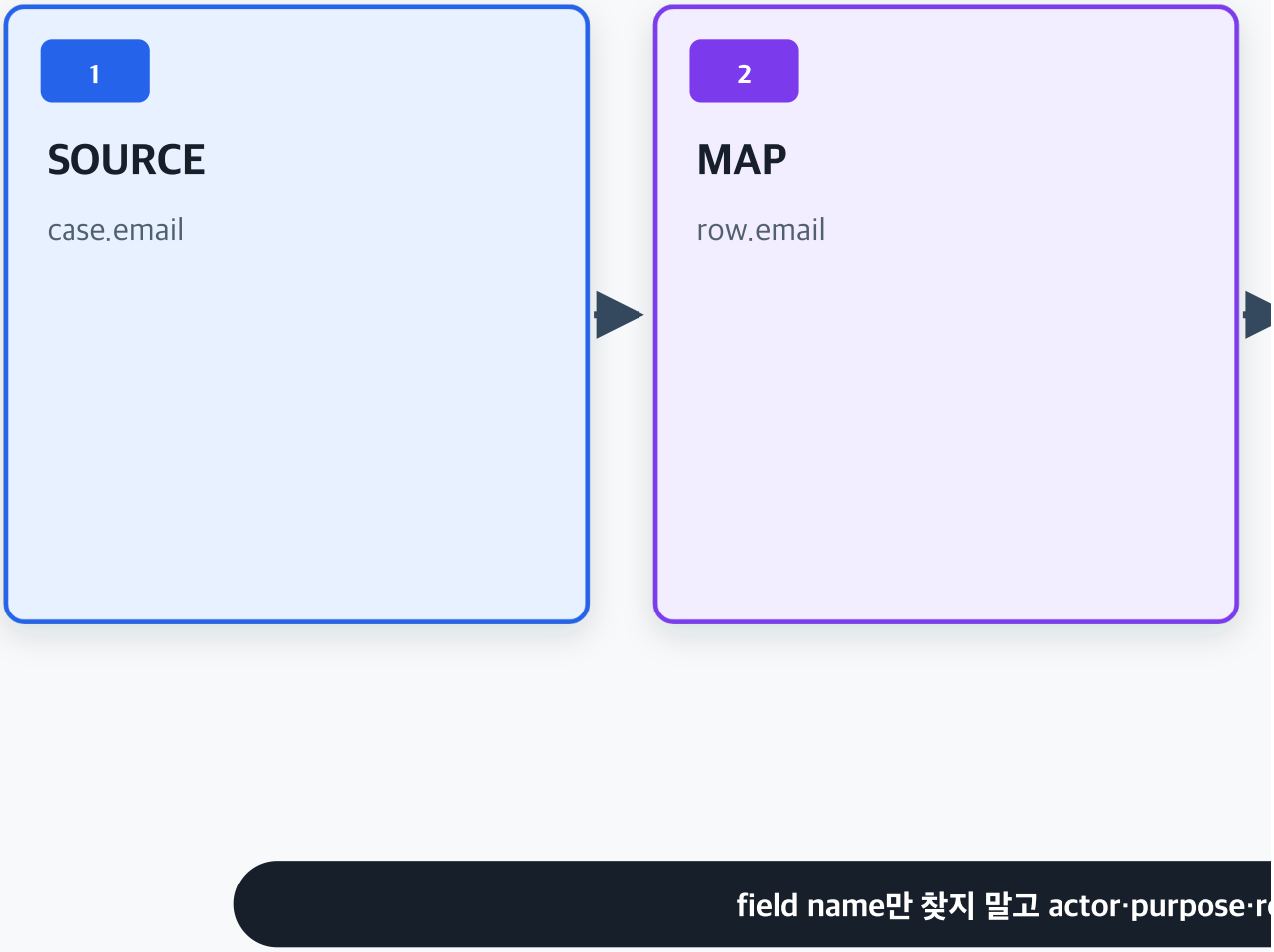


그림 8. field가 source에 존재하는 것과 public response·log로 내보내도 되는 것은 다릅니다. 각 sink마다 목적과 owner가 필요합니다.


1
SOURCE
case.email
2
MAP
row.email
field name만 찾지 말고 actor·purpose·r


3
SINK A
API response
4
SINK B
audit log
etention·visibility·redaction을 판정한다

10.1. 다섯 칸 data-flow 기록

```
SOURCE: synthetic case.contactEmail
CLASSIFICATION: contact data
TRANSFORM: public queue DTO에 복사
SINKS: HTTP response, application log
OWNER DECISION: task가 금지 → BLOCK
```

10.2. 자주 놓치는 sink

- HTTP·GraphQL response
- application·access·debug log
- analytics·telemetry event
- error tracker·trace
- cache·search index
- CSV·download·clipboard
- email·webhook·external API
- test snapshot·fixture·screenshot

10.3. Synthetic data도 contract를 바꾸지 않습니다

fixture가 가짜라는 이유로 production DTO에 민감 field를 추가해도 되는 것은 아닙니다. 같은 code path가 실제 data에 연결될 수 있고, public schema 자체가 노출 contract를 만들기 때문입니다.

10.4. Security finding을 과장하지 않습니다

확인한 범위만 씁니다.

```
OBSERVED:
  viewer request에서 response와 captured log에 email이 포함됨

INFERRED:
  production에서 같은 handler가 실제 contact data를 받으면 노출될 수 있음

NOT VERIFIED:
  production route binding과 data source
```

추론을 관찰처럼 쓰지 않으면서도 잠재 impact와 다음 owner를 명확히 남깁니다.

11. Exact Check와 Current Revision을 묶습니다

11.1. 실행 기록의 최소 단위

```
command:
working directory:
runtime·version:
base·head:
start·end time·timezone:
exit code:
stdout·stderr artifact:
side effect:
result: PASS / FAIL / NOT EXECUTED
```

11.2. 실습의 세 evidence

```
npm test
npm run test:independent
node scripts/review-audit.js
```

예상 결과:

```
candidate tests: PASS 6 / 6
independent tests: PASS 1 / FAIL 4
review audit: 2 PASS / 7 BLOCK
decision: REQUEST_CHANGES
```

후보 test만 보면 green입니다. 독립 test와 audit를 연결해야 authorization·PII·stable code·test erasure·dependency 문제를 볼 수 있습니다.

11.3. 실행하지 않은 check를 꾸미지 않습니다

```
performance: NOT EXECUTED · no staging permission
dependency source review: NOT REVIEWED · security owner requested
production route binding: UNKNOWN · deployment config out of scope
```

11.4. Evidence freshness

상태	판정
current head에서 실행, clean tree	current evidence
old head에서 실행	stale, 재실행 필요
dirty tree에서 실행	귀속 불명, snapshot 필요
command만 있고 output 없음	not evidenced
screenshot만 있고 revision 없음	참고용, 재현 불가

12. Finding을 재현 가능한 작은 Argument로 씁니다

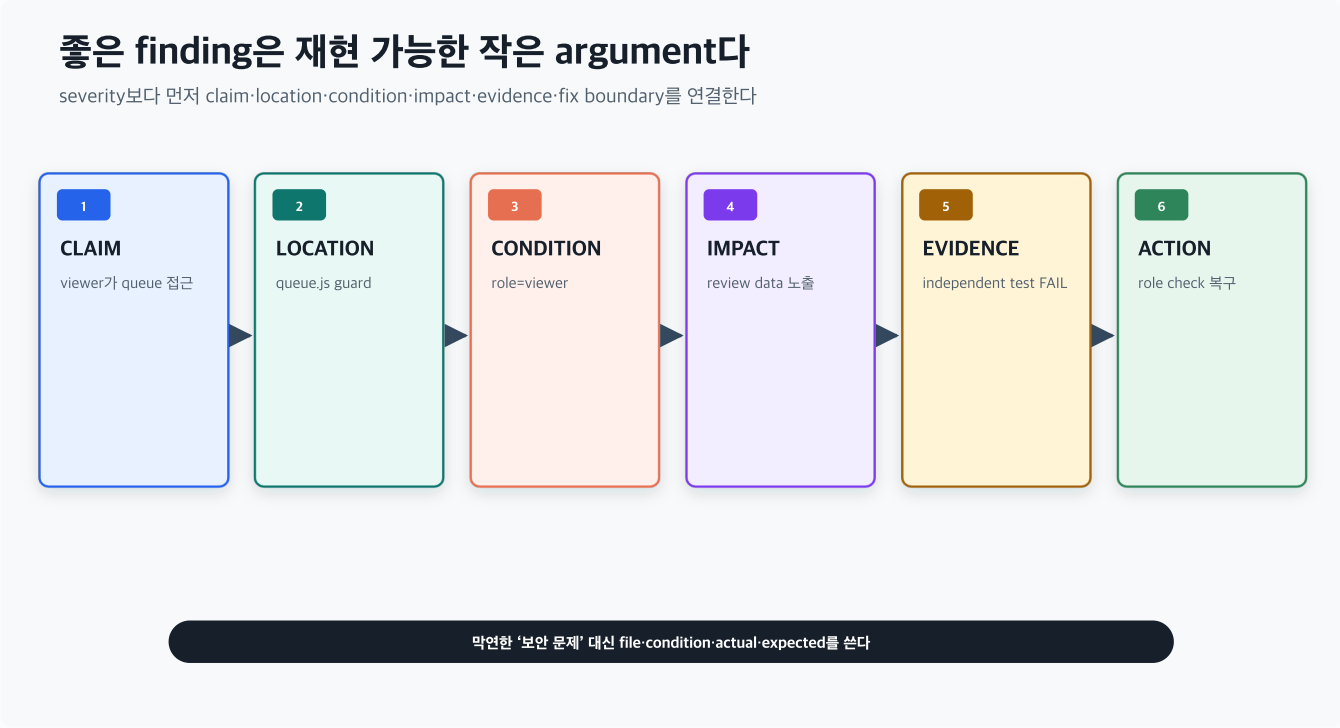
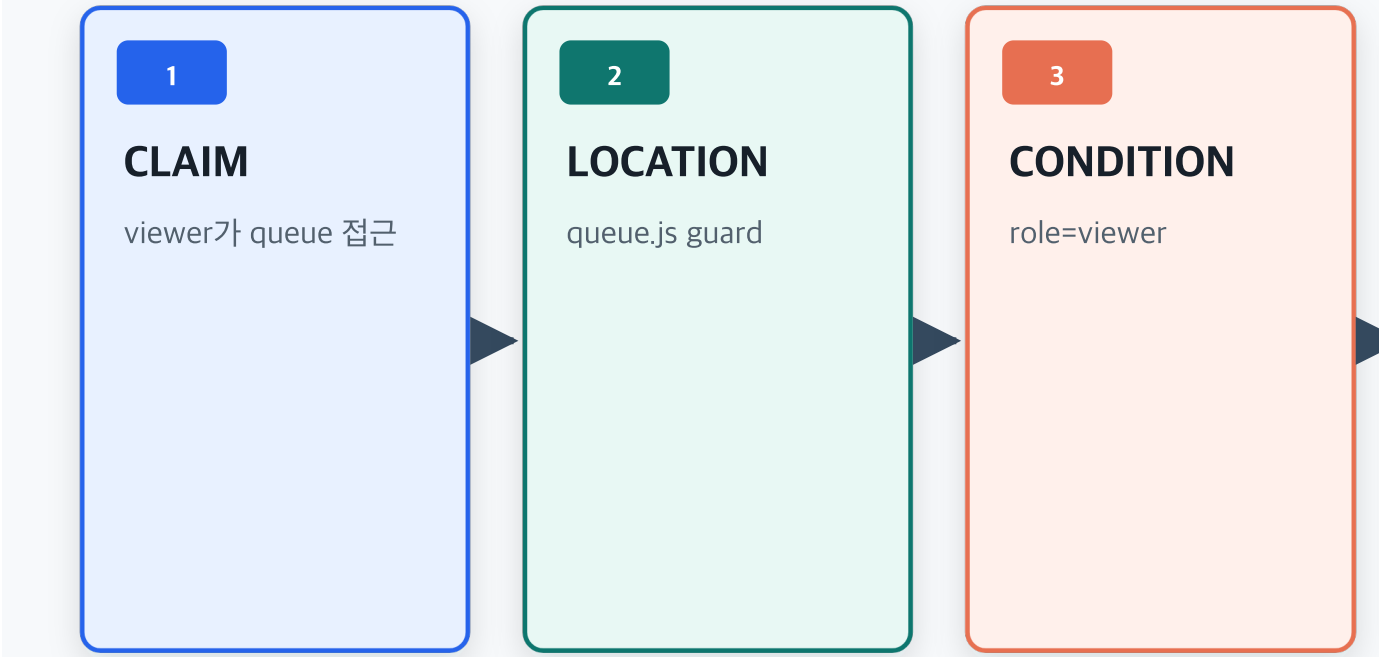
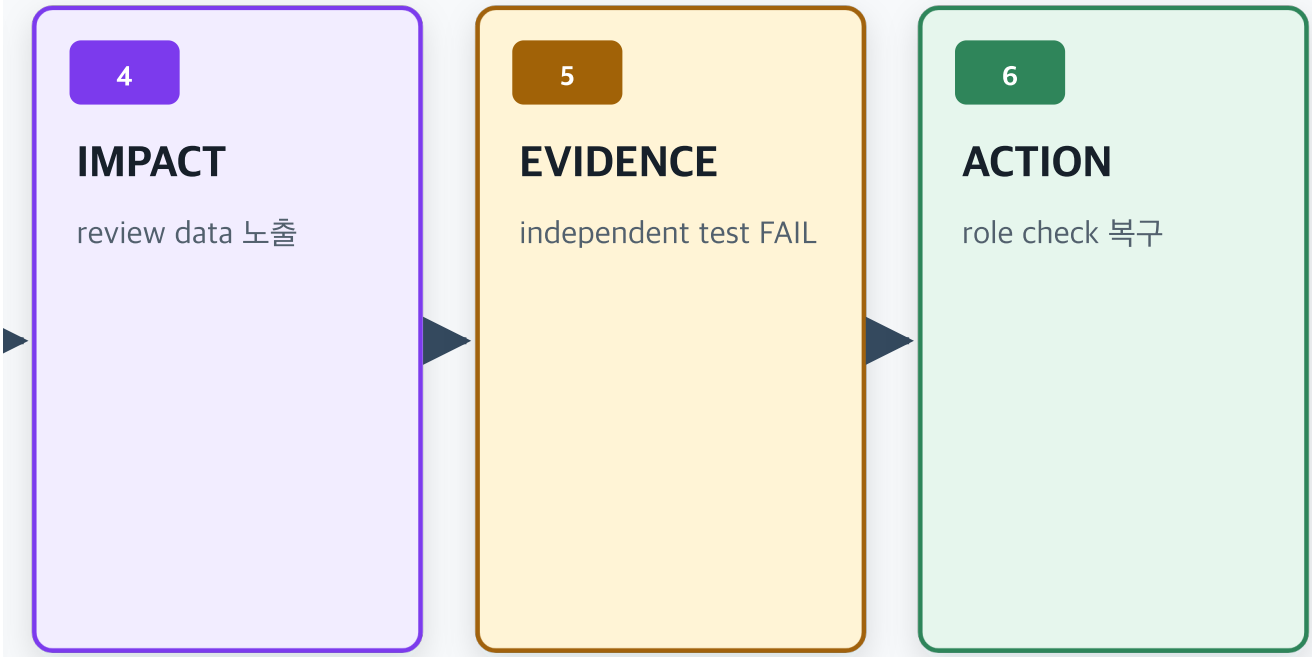


그림 9. “보안 문제가 있어 보입니다” 대신 위치·조건·실제 결과·영향·증거·수정 경계를 연결해야 작성자가 같은 문제를 재현할 수 있습니다.



막연한 ‘보안 문제’ 대신 file·con



dition·actual·expected를 쓴다

12.1. Finding의 여섯 요소

요소	질문	실습 예
Claim	정확히 무엇이 잘못됐는가	viewer가 operator queue에 접근
Location	어디서 시작되는가	`src/api/list-review-queue.js` role guard
Condition	어떤 input·actor·state에서	authenticated viewer 요청
Impact	누구·data·system에 어떤 결과인가	restricted queue·email 노출
Evidence	어떻게 재현했는가	independent test FAIL·captured log
Action	어떤 경계까지 고쳐야 하는가	role guard와 public DTO 복구

12.2. 나쁜 finding과 좋은 finding

나쁜 예:

```
보안이 안 좋아 보입니다. 권한 처리를 개선해 주세요.
```

좋은 예:

```
[HIGH] viewer가 operator 전용 queue와 contact email을 받습니다.

Location: src/api/list-review-queue.js의 listReviewQueue guard와 response mapping
Condition: actor.role = "viewer"
Actual: 200 response에 items[0].email이 있고 같은 값이 log에 기록됨
Expected: 403, response·log에 contact field 없음
Impact: 권한 없는 actor에게 restricted queue와 contact data 노출
Evidence: npm run test:independent → 4 FAIL, evidence/independent-tests.log
Action: operator role guard와 public DTO allowlist를 복구하고 새 head에서
        viewer·response·log negative tests를 실행하십시오.
```

12.3. Fix를 대신 설계하지 않습니다

action은 필요한 outcome과 경계를 명시하되 구현 세부를 과도하게 강제하지 않습니다. 한 가지 구현만 안전한 경우가 아니라면 작성자가 더 작은 해법을 선택할 여지를 둡니다.

```
과도한 지시:
  line 17에 if 문을 정확히 이 모양으로 넣으세요.

경계 있는 action:
  viewer가 403을 받고 public DTO·log에 contact field가 없도록 복구하며
  기존 operator success behavior는 유지하세요.
```

12.4. Finding마다 상태와 owner를 둡니다

ID	severity	status	owner	verification
F-01	HIGH	OPEN	author	viewer 403 independent test
F-02	HIGH	OPEN	author·data owner	response·log omit email
F-03	MEDIUM	OPEN	author	dependency removed or approved
F-04	HIGH	OPEN	author	security tests restored

상태는 `OPEN → FIX_PROPOSED → REREVIEW → VERIFIED → CLOSED`처럼 evidence와 함께 이동합니다. UI에서 thread가 resolved됐다는 사실만으로 `VERIFIED`가 되지 않습니다.

13. Severity와 Review Decision을 분리합니다

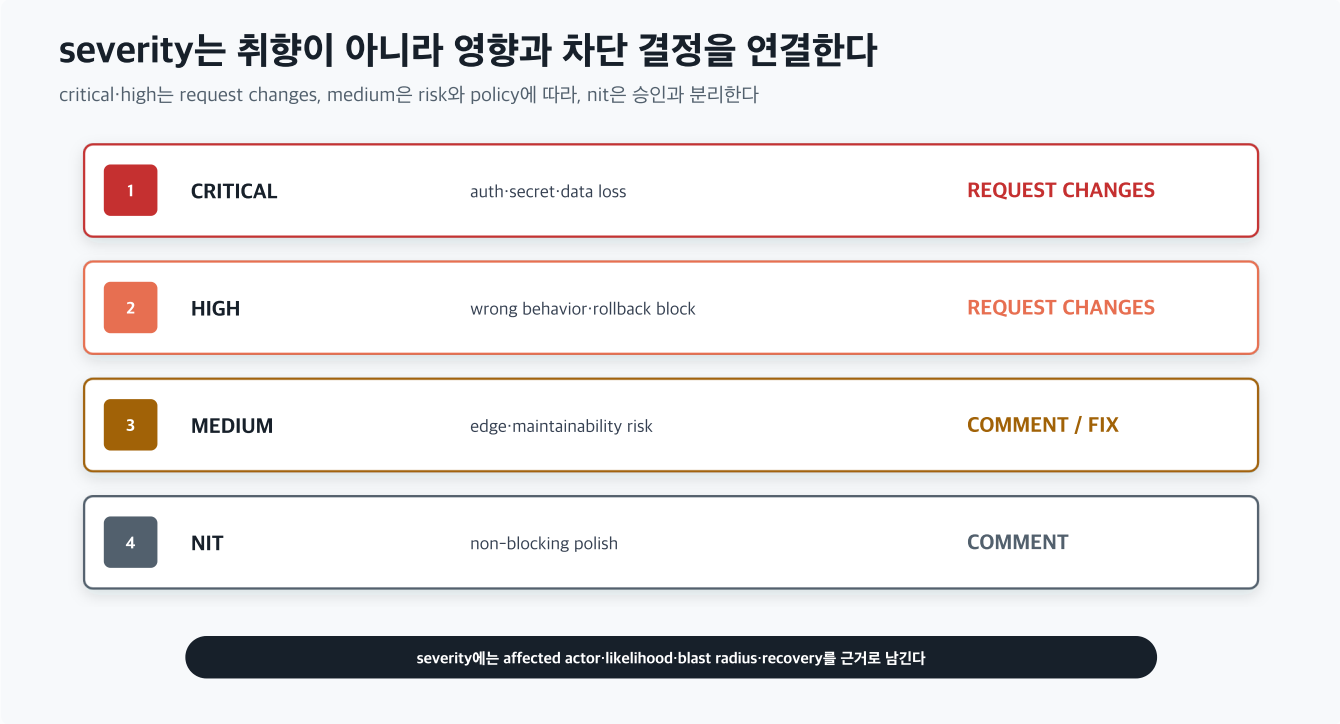
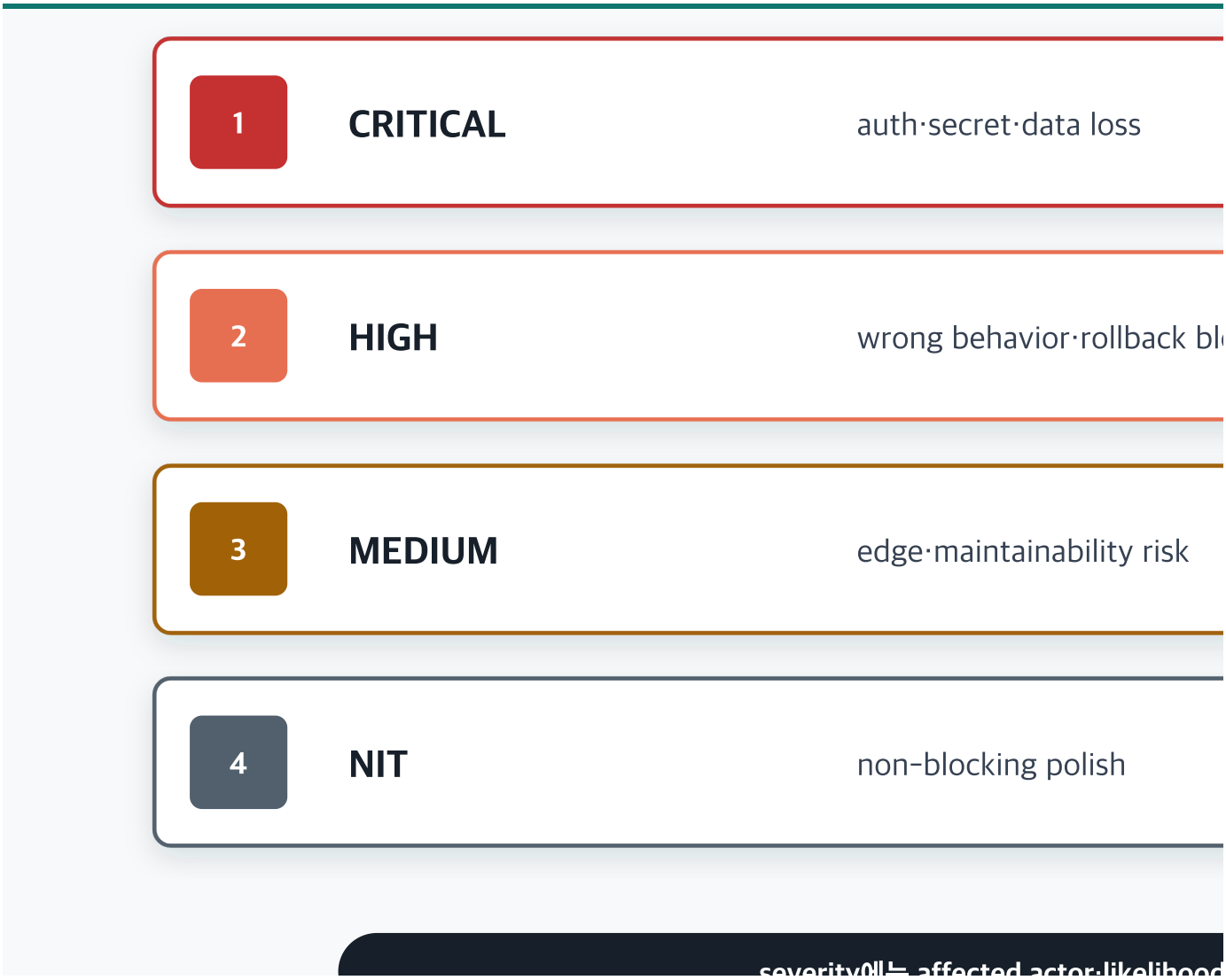
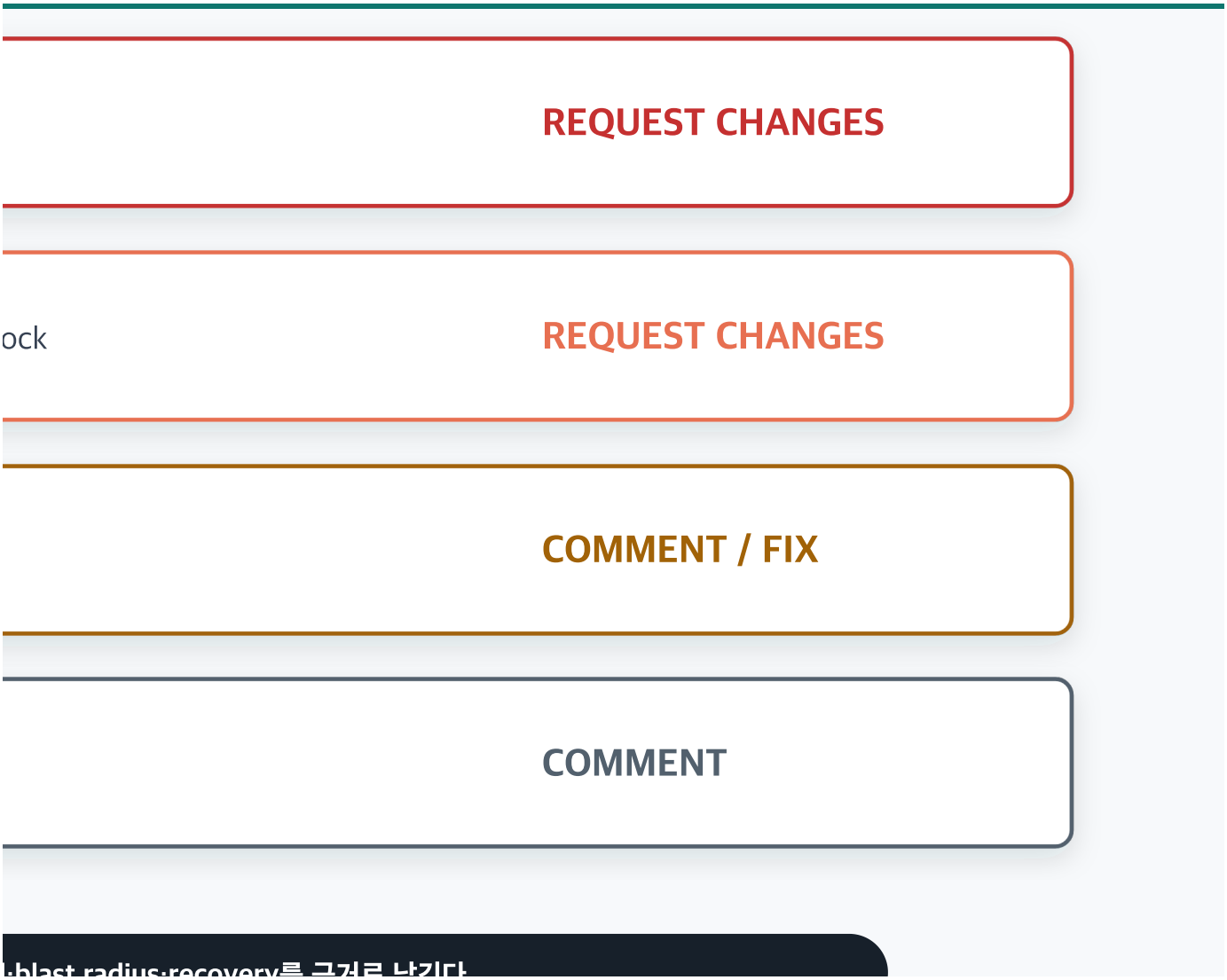


그림 10. severity는 목소리의 크기가 아니라 영향·발생 가능성·확산 범위·복구 난이도로 정하고, 미해결 blocker가 최종 판정을 결정합니다.


1
CRITICAL
auth·secret·data loss
2
HIGH
wrong behavior·rollback bl
3
MEDIUM
edge·maintainability risk
4
NIT
non-blocking polish


REQUEST CHANGES
ock
REQUEST CHANGES
COMMENT / FIX
COMMENT

13.1. Severity 판단 축

```
impact       actor·data·system에 무엇이 생기는가
likelihood   조건이 얼마나 쉽게 발생하는가
blast radius 한 user·한 tenant·전체 system 중 어디까지인가
detectability 발생을 얼마나 빨리 발견하는가
recovery     code·data·external effect를 얼마나 안전하게 되돌리는가
```

13.2. 실무용 네 등급

등급	의미	예	기본 판정
CRITICAL	즉시 악용·대규모 손실·복구 곤란	credential 노출·irreversible data loss	merge·release BLOCK
HIGH	권한·data·핵심 contract 위반	viewer 접근·PII response	REQUEST_CHANGES
MEDIUM	제한적 regression·운영·유지 위험	불필요 runtime dependency	보통 fix 또는 owner 승인
LOW	명확성·일관성·작은 후속 개선	naming·local duplication	comment 또는 follow-up

팀의 공식 severity 정책이 있으면 그 정의를 우선합니다. 숫자 점수를 만들었다고 객관적이 되는 것은 아닙니다. 판단 근거를 문장으로 남깁니다.

13.3. 세 가지 review decision

decision	뜻	사용 조건
COMMENT	질문·제안·비차단 피드백	승인·거절 의사와 분리
APPROVE	current head가 기준을 충족	required gate PASS, blocker 없음
REQUEST_CHANGES	수정 전 merge하면 안 됨	unresolved critical·high 또는 강제 정책 실패

13.4. 칭찬·취향·blocker를 섞지 않습니다

```
BLOCKER: authorization regression
SUGGESTION: helper naming 개선
POSITIVE: public DTO allowlist 접근은 명확함
QUESTION: dependency는 CLI 색상용인가
```

작성자는 무엇이 반드시 고쳐져야 하는지 알아야 합니다. 모든 comment를 같은 강도로 쓰면 중요한 finding이 묻힙니다.

13.5. 실습 후보의 판정

```
RANGE_FIXED PASS
CLEAN_WORKTREE PASS

SCOPE_DEVIATION BLOCK
DEPENDENCY_ADDED BLOCK
SENSITIVE_FIELD_EXPOSED BLOCK
AUTHORIZATION_WEAKENED BLOCK
SECURITY_TEST_ERASURE BLOCK
MISSING_ACCEPTANCE_COVERAGE BLOCK
SELF_ORACLE_TEST BLOCK

decision REQUEST_CHANGES
```

rollback rehearsal이 PASS여도 후보를 승인하지 않습니다. 복구 가능성은 위험을 관리하는 한 요소이지 위반된 behavior·safety contract를 상쇄하지 않습니다.

14. Feedback·Fix·Rereview를 새 Evidence Loop로 만듭니다

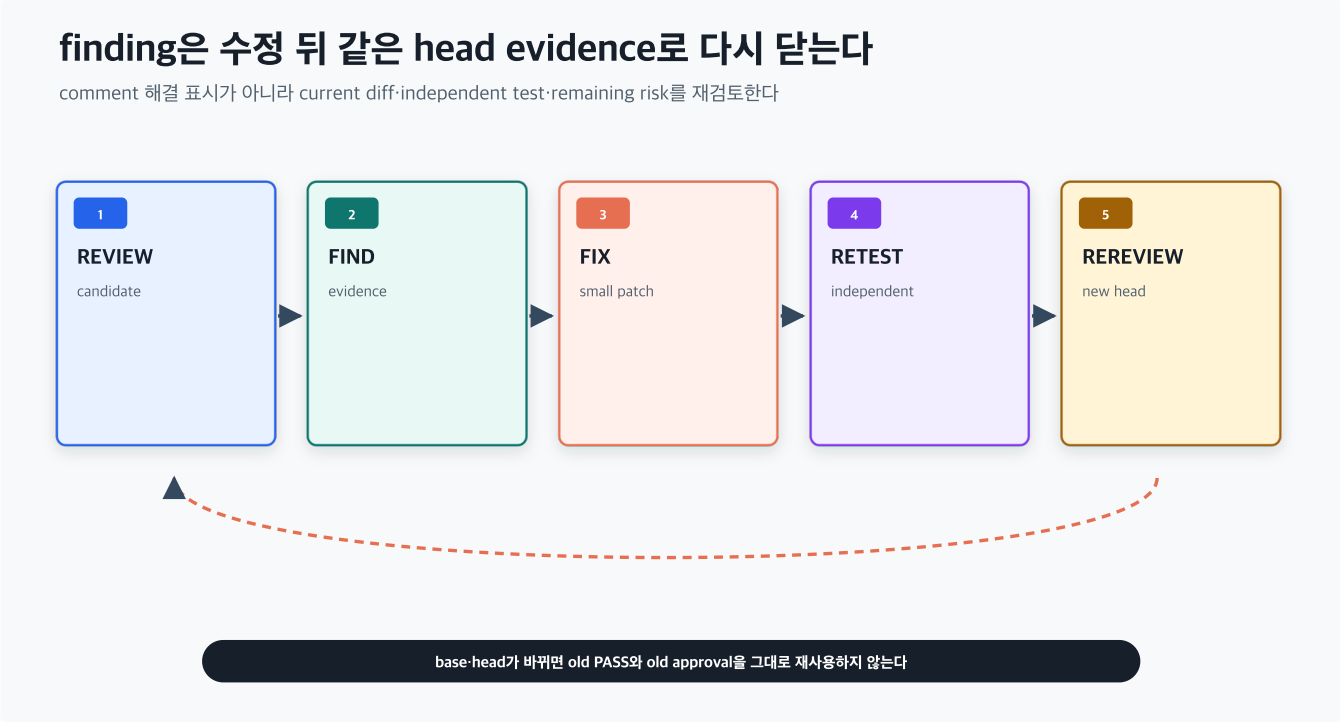
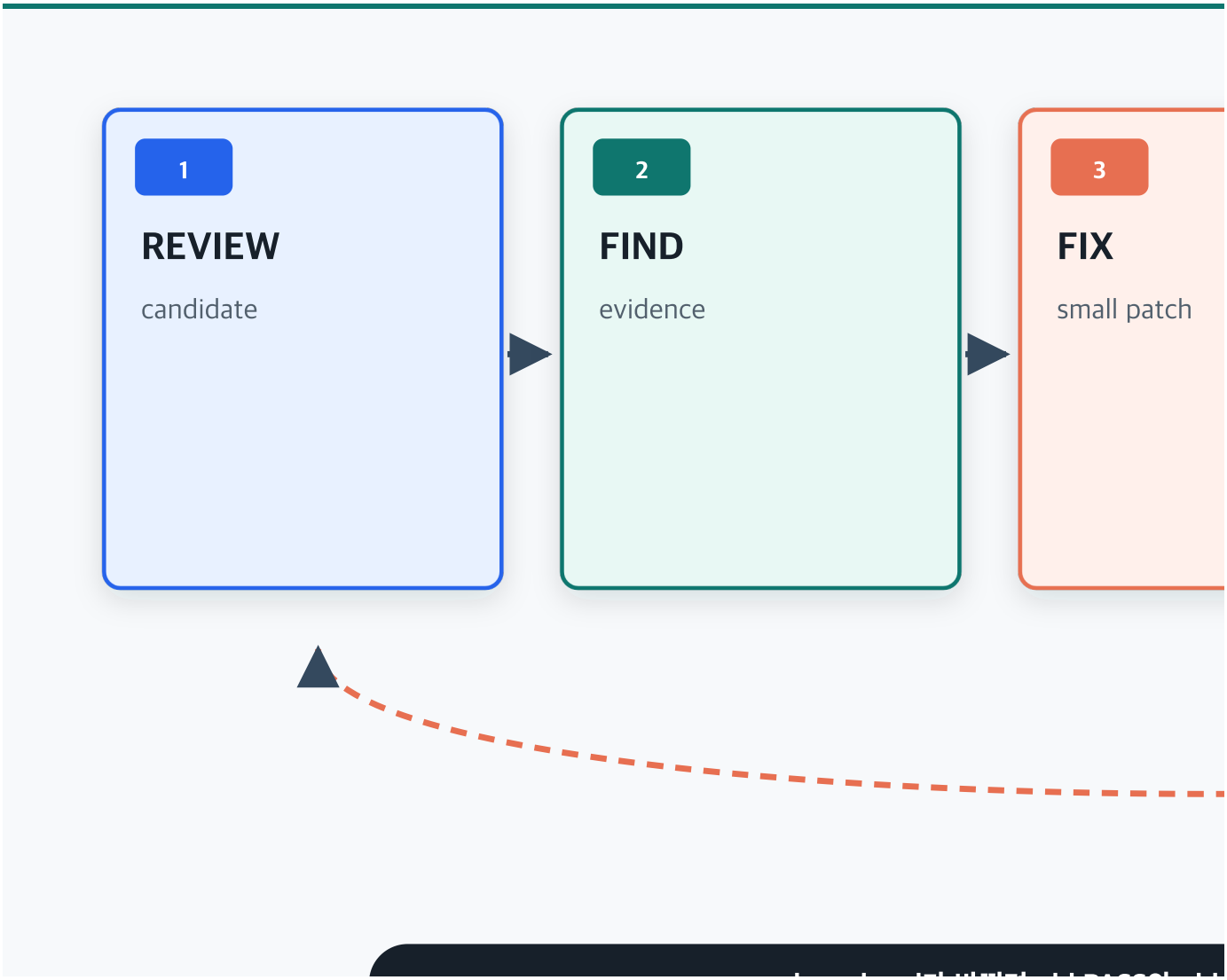
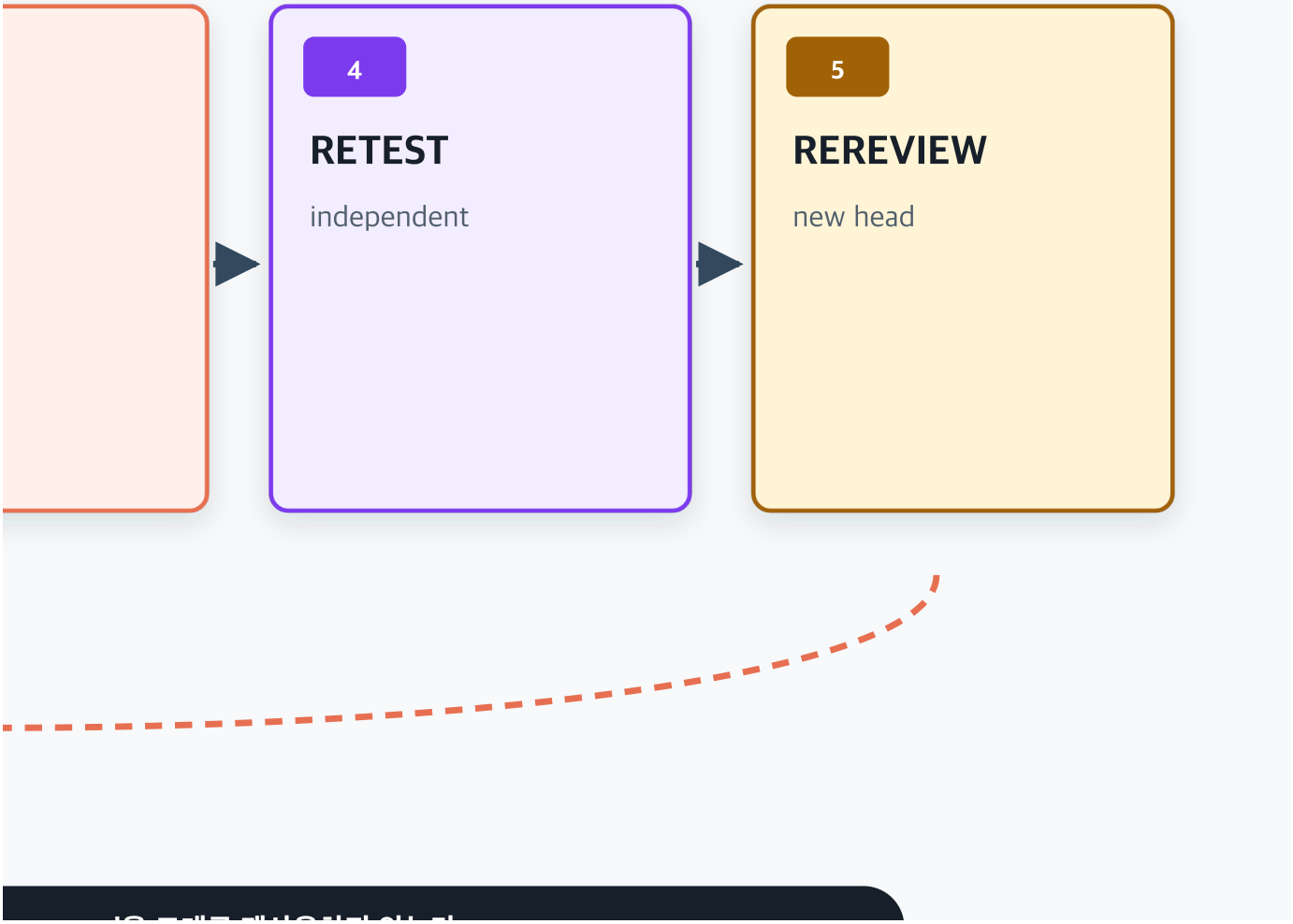


그림 11. comment를 닫는 것이 아니라 새 head에서 finding과 전체 task를 다시 증명해야 loop가 끝납니다.


1
REVIEW
candidate
2
FIND
evidence
3
FIX
small patch


4
RETEST
independent
5
REREVIEW
new head

14.1. 작성자에게 보내는 작은 packet

```
finding ID·severity
base·reviewed head
file·symbol·condition
actual / expected
impact
exact reproduction
bounded action
owner·next state
```

14.2. Fix를 받을 때 확인할 네 가지

1. old head에서 new head로 finding을 실제로 제거했는가
2. 수정이 새 surface와 dependency를 추가하지 않았는가
3. focused independent test가 current new head에서 통과하는가
4. original base에서 new head까지 전체 task contract가 유지되는가

14.3. Resolved와 Verified의 차이

상태	누가 할 수 있는가	증거
RESOLVED	작성자·도구가 thread 정리	UI 상태
FIX_PROPOSED	새 commit 제출	new head ID
VERIFIED	reviewer가 재현	current diff·check output
CLOSED	판정과 기록 완료	final packet

14.4. 자동 review의 위치

AI review와 scanner는 넓은 후보 finding을 빠르게 만들 수 있습니다. 하지만 자동 finding은 그 자체로 승인·거절 권한이 아니며 false positive·context miss가 있을 수 있습니다. 사람은 원래 task와 owner 정책을 연결하고, 실제로 재현한 evidence와 `NOT VERIFIED`를 구분합니다.

30초 확인 4
새 head에서 F-01은 고쳤지만 `package.json`이 또 바뀌었다면 F-01만 닫고 승인합니까? 새 surface를 inventory하고 전체 range를 다시 판정합니다.

15. Undo·Restore·Revert·Recovery를 구분합니다

그림 12. 네 동작은 이름이 비슷해도 대상과 증거가 다릅니다. repository가 돌아온 것과 시스템 전체가 회복된 것을 혼동하지 않습니다.

1

UNDO

tool checkpoint

session edit

2

RESTORE

working tree

uncommitted file

실행 전 current status·owner·backup

3

REVERT

new inverse commit

shared history

4

RECOVERY

data·service

external effect

·side effect를 확인하고 rehearsal한다

15.1. 네 가지 되돌리기

동작	대상	기록	대표 용도	한계
tool checkpoint·undo	도구가 추적한 local edit	도구별	아직 공유 전 빠른 복원	shell·external effect 미포함 가능
`git restore`	working tree·index path	commit 없음	local uncommitted file 복원	shared history·external state 미복구
`git revert`	기존 commit의 역변경	새 commit	공유 이력에서 안전한 역변경	conflict·data·message 미복구
recovery runbook	DB·queue·cache·service·message	운영 기록	외부 상태 복원·보상	별도 권한·owner·검증 필요

15.2. Checkpoint는 Git의 대체물이 아닙니다

일부 AI coding 도구의 checkpoint는 도구가 수행한 file edit만 추적할 수 있습니다. shell command가 만든 file, migration, external API 호출, 다른 process의 변경은 포함되지 않을 수 있습니다. 범위를 확인하고 장기 협업 이력은 Git과 review record로 남깁니다.

15.3. `git restore`의 안전 경계

`git restore`는 working tree나 index의 내용을 source tree에서 복원합니다. local 변경을 버릴 수 있으므로 대상 path·source·staged 여부를 먼저 확인합니다.

```
git status --short
git diff -- path/to/file
git restore --source=<known-revision> --worktree -- path/to/file
```

이 manual은 공유 이력에서 파괴적 reset을 recovery 기본값으로 가르치지 않습니다. 현재 상태와 owner를 모른 채 local 작업을 잃을 수 있기 때문입니다.

15.4. `git revert`의 의미

`git revert <commit>`은 지정 commit이 만든 변화를 역으로 적용하고, 보통 새 commit으로 기록합니다. 기존 이력을 지우지 않으므로 공유 branch에서 감사 가능성이 높습니다. 다만 다음은 자동 해결되지 않습니다.

- 이후 commit과의 conflict
- schema downgrade·data transform

- 이미 보낸 email·webhook·message
- 외부 vendor에 생성한 resource
- cache·index·analytics에 남은 data

15.5. Recovery 질문

```
code: 어떤 revision·tree로 돌아가야 하는가?
schema: backward compatible한가, downgrade path가 있는가?
data: 복원·보상·재처리 중 무엇인가?
message: 이미 전달된 event를 어떻게 다루는가?
service: flag·traffic·deploy를 어떻게 전환하는가?
evidence: 회복 완료를 무엇으로 판정하는가?
owner: 누가 실행하고 누가 승인하는가?
```

16. Rollback Rehearsal은 격리해서 수행합니다

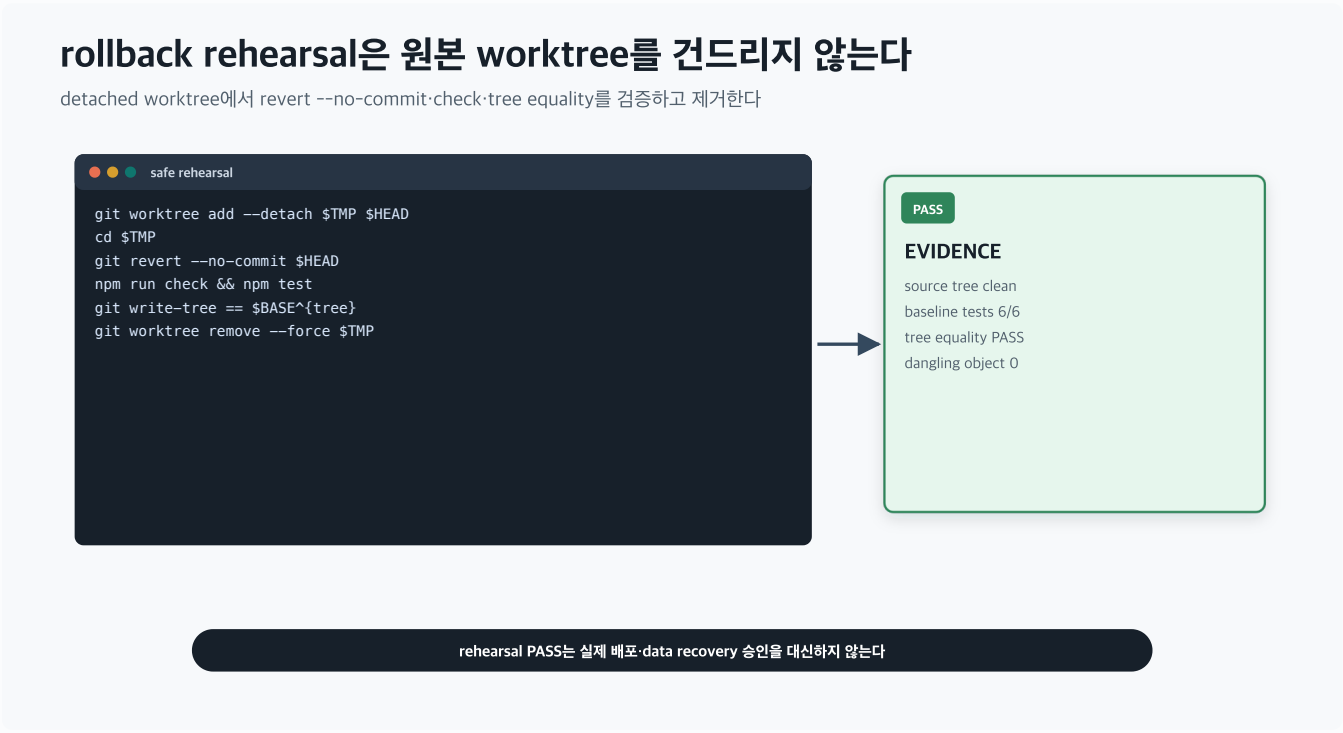
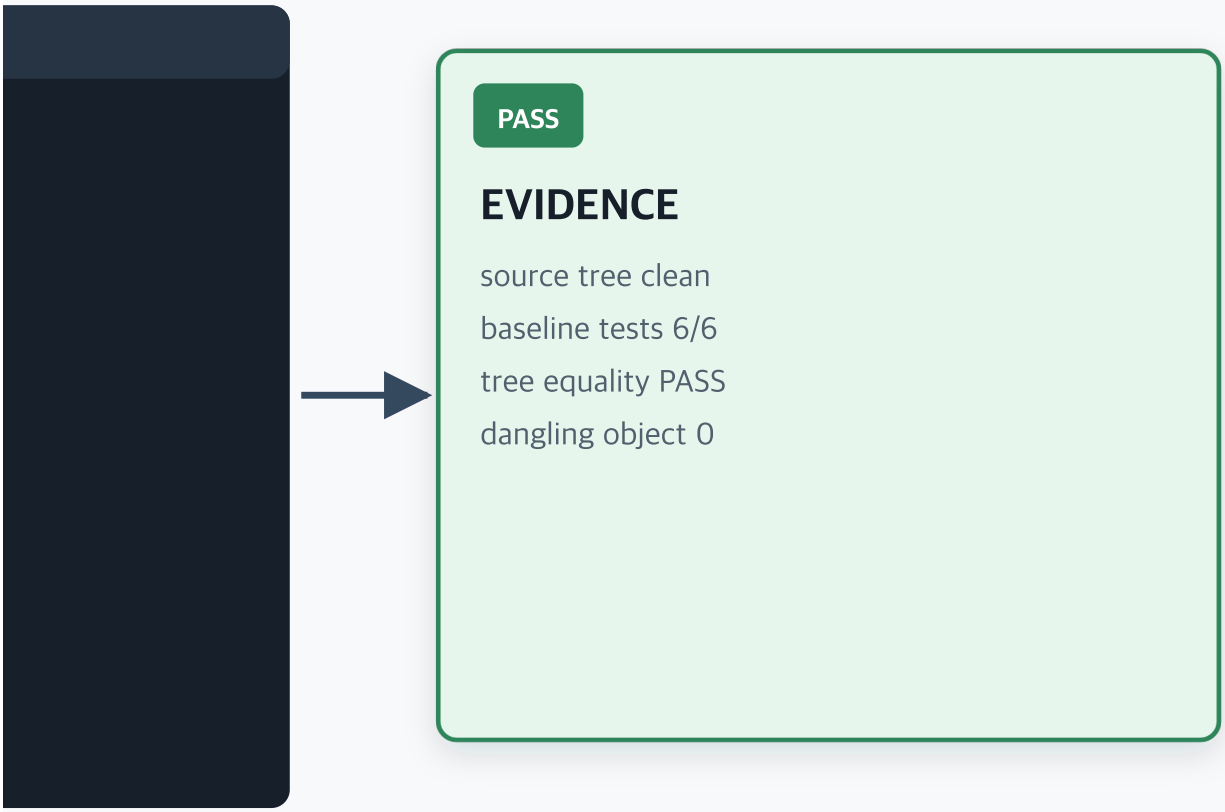


그림 13. 현재 작업 공간을 건드리지 않는 격리 환경에서 역변경·검증·tree 동일성·cleanup까지 수행해야 rehearsal evidence가 됩니다.

safe rehearsal

```
git worktree add --detach $TMP $HEAD
cd $TMP
git revert --no-commit $HEAD
npm run check && npm test
git write-tree == $BASE^{tree}
git worktree remove --force $TMP
```

rehearsal PASS는 실제 배포·data



recovery 승인을 대신하지 않는다

16.1. 실습 rehearsal 흐름

```
1. detached temporary worktree 생성
2. candidate head에서 시작했는지 확인
3. git revert --no-commit으로 역변경 적용
4. baseline independent test 실행
5. restored tree와 baseline tree ID 비교
6. temporary worktree 제거
7. result·limit 기록
```

실습 결과:

```
baseline tree = 7ef0a05f74df0a641df22af5b63332170028057e
restored tree = 7ef0a05f74df0a641df22af5b63332170028057e
baseline tests = PASS
cleanup = PASS
rollback rehearsal = PASS
```

16.2. 왜 `--no-commit`을 쓰는가

rehearsal에서는 실제 shared history에 revert commit을 남기는 것이 목적이 아닙니다. temporary worktree의 index·working tree에 역변경을 적용하고 tree와 test를 확인한 뒤 폐기합니다. 이렇게 하면 dangling rehearsal commit을 만들지 않고 검증할 수 있습니다.

16.3. Rehearsal이 증명하는 것과 못 하는 것

증명	증명하지 못함
candidate code tree의 역변경 적용 가능	production deploy 성공
baseline tree와 content 동일	DB·cache·message 복구
baseline local tests 통과	traffic·latency·vendor state 정상
temporary environment cleanup	실제 incident의 권한·시간 확보

16.4. Rehearsal 안전 checklist

- temporary path가 기존에 없는지 확인
- detached·isolated environment인지 확인

- network·credential 없이 실행 가능한지 확인
- destructive command를 사용하지 않는지 확인
- 시작 head와 target commit을 출력
- check failure에서 fail closed
- cleanup을 success·failure 모두에서 실행
- final tree와 baseline tree를 object ID로 비교

17. 최종 Review Packet은 일곱 Gate를 함께 판정합니다

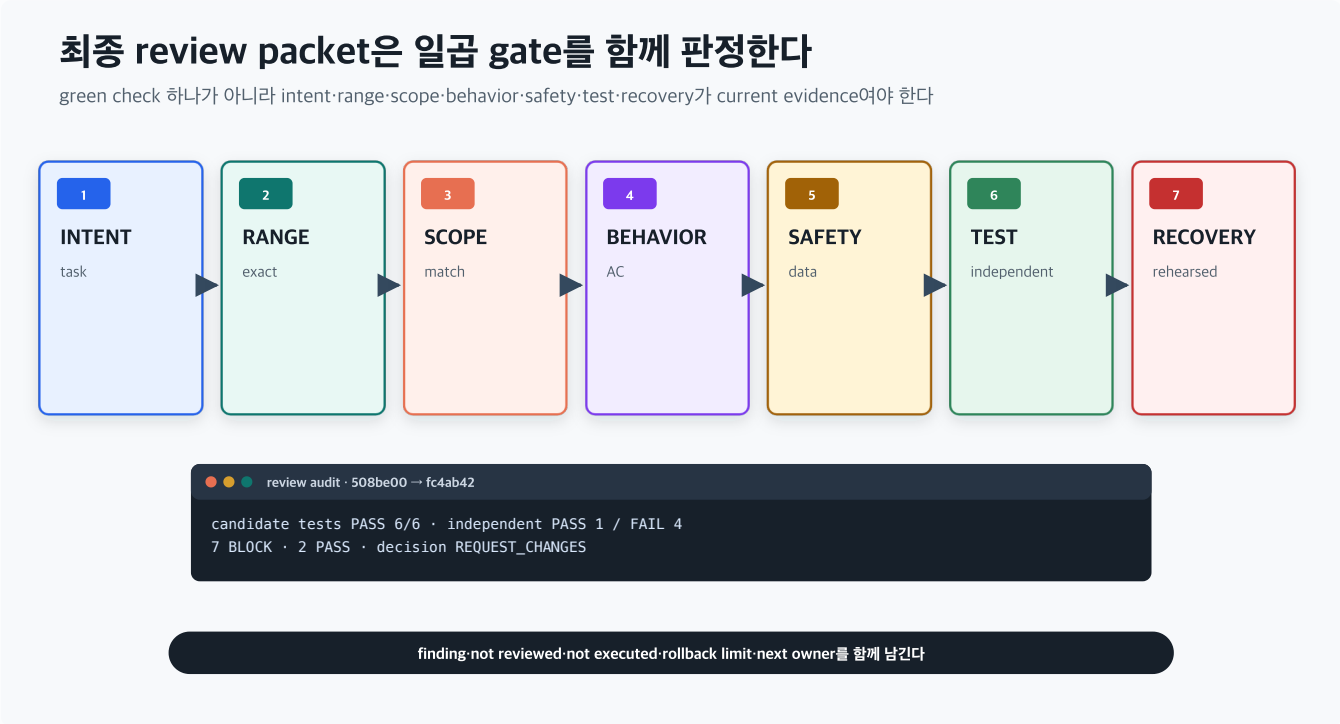
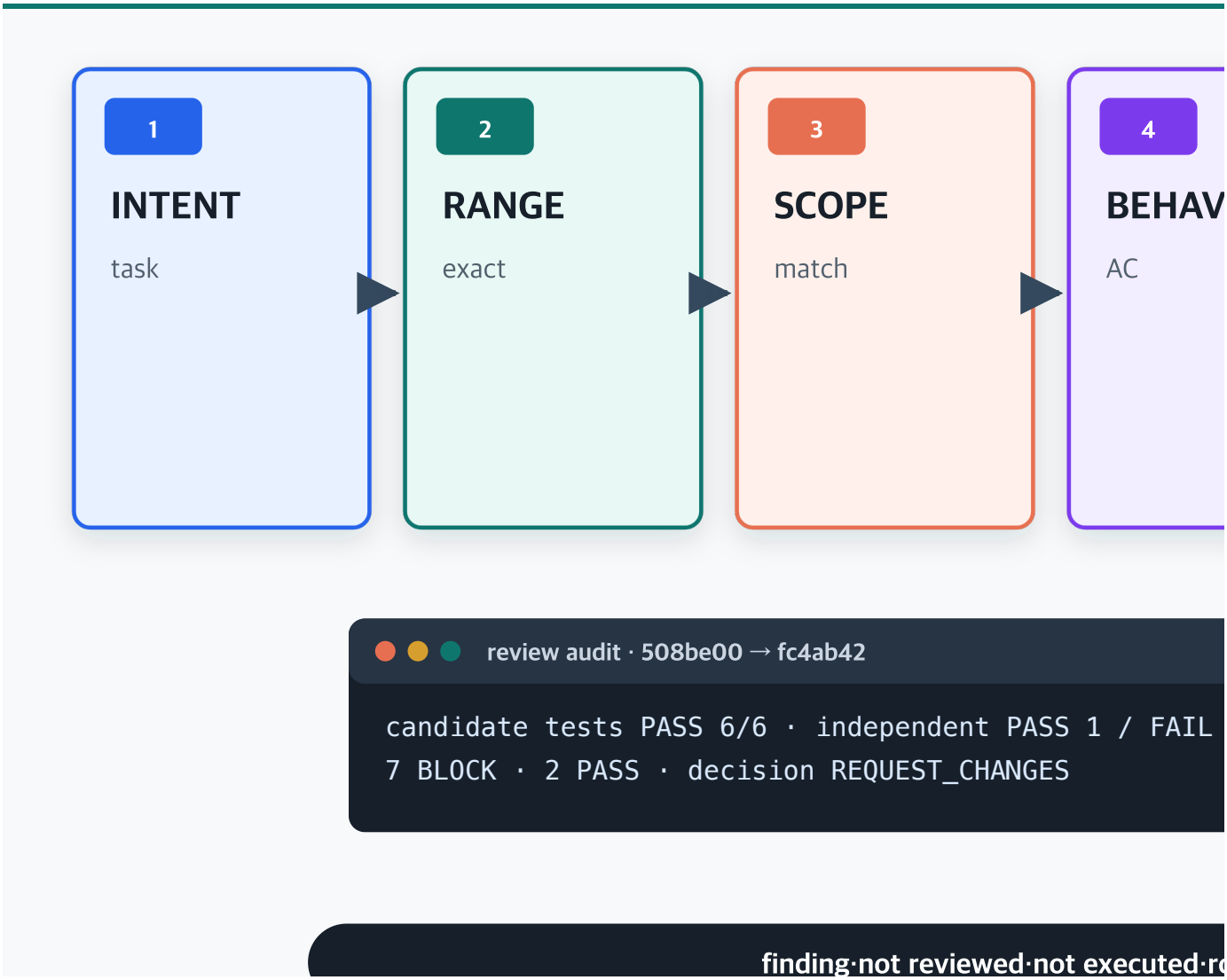
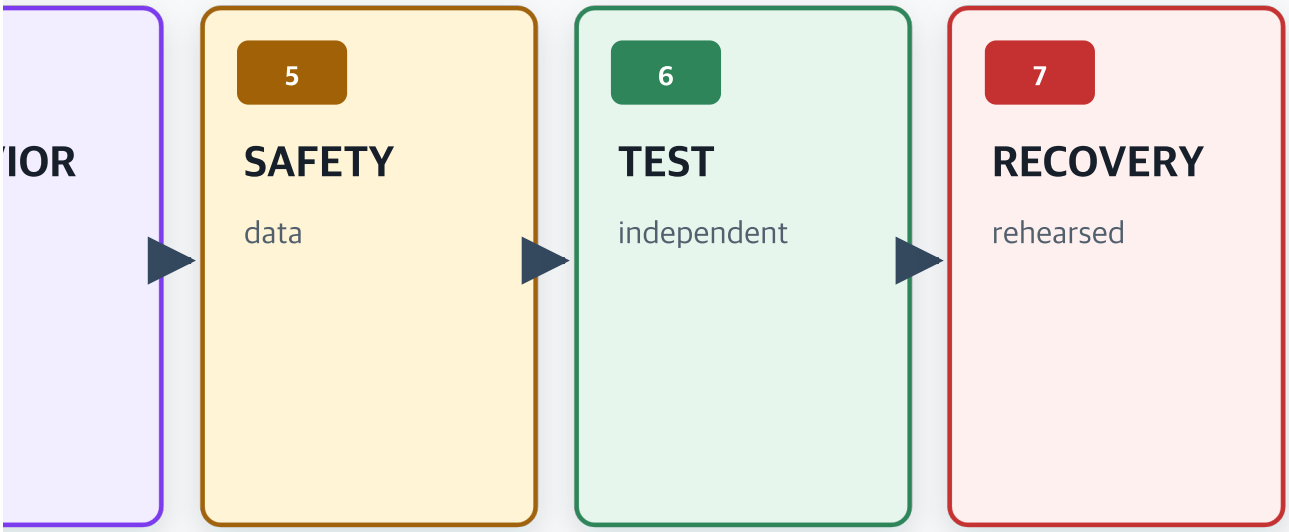


그림 14. green check 하나가 아니라 일곱 gate의 current evidence를 묶습니다. 실습은 audit 2 PASS·7 BLOCK으로 REQUEST_CHANGES입니다.


1
INTENT
task
2
RANGE
exact
3
SCOPE
match
4
BEHAV
AC
review audit · 508be00 → fc4ab42
candidate tests PASS 6/6 · independent PASS 1 / FAIL
7 BLOCK · 2 PASS · decision REQUEST_CHANGES
finding·not reviewed·not executed·r



4

ollback limit·next owner를 함께 남긴다

Gate 1 · Intent

```
원래 actor·outcome·acceptance·non-goal이 고정됐는가?
AI summary가 기준을 덮어쓰지 않았는가?
```

Gate 2 · Range

```
immutable base·head와 direction이 있는가?
working tree 상태와 evidence revision이 같은가?
```

Gate 3 · Scope

```
changed surface 전체가 inventory됐는가?
허용 범위 밖 변화가 설명·승인됐는가?
```

Gate 4 · Behavior

```
positive·negative·boundary와 error path가 맞는가?
public consumer가 contract를 다르게 해석하지 않는가?
```

Gate 5 · Safety

```
auth·permission·data source-to-sink·secret·external effect를 봤는가?
필요한 owner가 승인했는가?
```

Gate 6 · Test

```
candidate test가 독립 oracle을 쓰는가?
삭제·약화된 보호 test가 없는가?
current head에서 focused·required checks를 실행했는가?
```

Gate 7 · Recovery

```
code와 external state의 rollback boundary를 구분했는가?
격리된 rehearsal과 runbook owner가 있는가?
```

17.1. 판정 규칙

```
APPROVE:
  required gates PASS, unresolved blocker 0, evidence current

REQUEST_CHANGES:
  critical·high 또는 강제 policy BLOCK

COMMENT:
  non-blocking suggestion·question, final approval 상태와 별도

PENDING_REREVIEW:
  new head 제출, current verification 전

BLOCKED:
  authority·environment·contract가 없어 reviewer가 판정 불가
```

18. 완성된 실습 Review Record

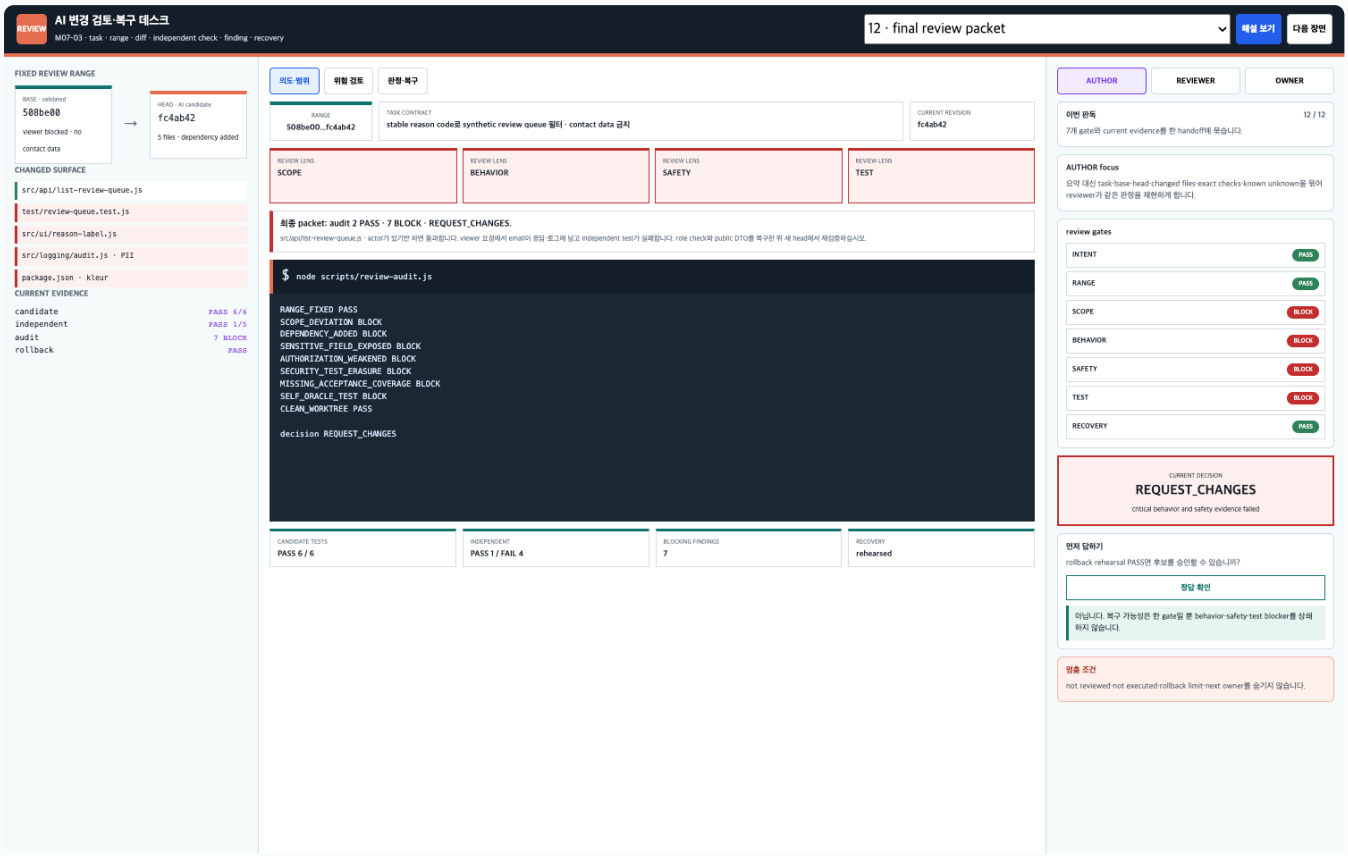


그림 15. 좌측의 변경 surface, 중앙의 independent evidence, 우측의 gate와 판정을 한 화면에서 연결합니다.

18.1. Review identity

항목	값
task	stable reason code로 synthetic review queue filter
base	`508be0038d7012530546b915ac319174528001ef`
head	`fc4ab428c05d967ce407d07815612be70914e441`
reviewed range	base → head
allowed files	`src/api/list-review-queue.js`, `test/review-filter.test.js`
actual files	5 files, 4 scope deviations
candidate tests	PASS 6/6
independent tests	PASS 1/FAIL 4

항목	값
audit	2 PASS/7 BLOCK
recovery rehearsal	repository tree PASS
decision	REQUEST_CHANGES

18.2. Finding·risk·evidence matrix

ID	finding	risk	evidence	action
F-01	operator role guard 제거	HIGH	viewer 403 test FAIL	authorization 복구
F-02	email response·log 노출	HIGH	two independent FAIL	DTO·log allowlist
F-03	label 기반 filter	HIGH	stable code test FAIL	code 비교
F-04	security test 삭제	HIGH	base→head deletion	protection 복구
F-05	self-oracle test	HIGH	production helper import	independent expected
F-06	negative acceptance 누락	HIGH	audit BLOCK	negative case 추가
F-07	`kleur` dependency	MEDIUM	manifest diff	제거 또는 owner 승인

18.3. Recovery statement

```
OBSERVED:
  detached worktree에서 candidate 역변경 후 baseline tree와 동일하고
  baseline independent tests가 통과했다.

LIMIT:
  이 repository는 synthetic local data만 사용한다.
  rehearsal은 production deploy·database·cache·message recovery를 증명하지 않는다.

DECISION:
  repository rollback path는 REHEARSED.
  candidate review decision은 REQUEST_CHANGES.
```

19. 60분 실전 미션

미션 1 · Task contract · 5분

원래 actor·outcome·must preserve·allowed surface·acceptance를 한 페이지에 적습니다. source anchor와 owner를 붙입니다.

미션 2 · Base·head·status · 5분

full revision, branch, clean/dirty 상태, runtime을 기록합니다. evidence folder 이름에 짧은 head를 포함합니다.

미션 3 · Changed surface · 5분

name-status와 stat을 읽고 source·test·dependency·config·data·docs로 분류합니다. 허용 범위 밖 파일을 표시합니다.

미션 4 · Risk-first diff · 10분

auth·data·secret·external effect·public contract부터 읽습니다. added line뿐 아니라 삭제된 protection을 별도 기록합니다.

미션 5 · Independent checks · 10분

candidate가 만든 test와 다른 oracle을 사용해 positive·negative·boundary를 확인합니다. 실행하지 못한 것은 이유와 owner를 남깁니다.

미션 6 · Finding matrix · 10분

claim·location·condition·actual·expected·impact·evidence·action을 작성합니다. severity 근거를 문장으로 씁니다.

미션 7 · Recovery · 10분

code·schema·data·message·service recovery를 분리합니다. 안전하면 격리 rehearsal을 수행하고 한계를 기록합니다.

미션 8 · Final decision · 5분

일곱 gate와 unresolved blocker를 세고 `APPROVE / REQUEST_CHANGES / BLOCKED / PENDING_REREVIEW` 중 하나를 선택합니다.

20. 자주 실패하는 검토 패턴

실패 패턴	왜 위험한가	고치는 행동
AI summary만 읽기	실제 diff·삭제·scope deviation 누락	base→head inventory
source 한 파일만 읽기	manifest·test·config의 위험 누락	changed surface map
candidate test green을 승인 근거로 사용	self-oracle·test erasure 가능	independent contract test
추가 줄만 읽기	제거된 guard·test·cleanup 누락	deletion inventory
취향 comment부터 쓰기	blocker 발견 지연	risk-first lens
“보안 문제”처럼 모호하게 쓰기	재현·수정·검증 불가	finding six elements
thread resolved를 verified로 간주	새 head regression 미확인	rerun current evidence
rollback=git command라고 생각	external effect 잔존	recovery boundary table
공유 이력을 파괴적으로 되돌리기	다른 작업·감사 이력 손실	revert·owner·backup
실행하지 않은 check를 생략	coverage를 과대평가	NOT EXECUTED 상태
severity를 감정적으로 부풀리기	신뢰와 우선순위 훼손	impact·likelihood·blast·recovery
모든 것을 혼자 승인	전문 authority 부재	security·data·ops owner 호출

21. 셀프 테스트 · 먼저 답한 뒤 해설을 엽니다

Q1

AI가 “요청대로 구현했고 test 6개가 통과했다”고 보고했습니다. 가장 먼저 확인할 것은 무엇입니까?

정답 보기

원래 task contract와 exact base·head입니다. 설명과 test 수보다 무엇을 어떤 기준에서 비교하는지 먼저 고정해야 합니다.

Q2

`git diff base head`와 `git diff base...head`는 언제 구분해야 합니까?

정답 보기
두 endpoint tree의 직접 차이를 보려는지, merge base 이후 한 branch의 변화를 보려는지에 따라 구분합니다. review record에는 사용한 의미와 명령을 적습니다.

Q3

후보가 허용 파일 두 개 외에 manifest와 security test도 바꿨습니다. 첫 행동은 무엇입니까?

정답 보기
changed surface inventory에 scope deviation을 기록하고 이유·owner·위험을 확인합니다. 핵심 source만 읽고 넘어가지 않습니다.

Q4

인증된 viewer가 operator 전용 endpoint에 접근합니다. 어떤 개념을 혼동한 것입니까?

정답 보기
authentication과 authorization입니다. actor의 존재가 resource·action 권한을 뜻하지 않습니다.

Q5

Production helper를 test의 expected 계산에도 사용하면 왜 위험합니까?

정답 보기
구현과 test가 같은 오류를 공유하는 self-oracle이 됩니다. task contract에서 직접 나온 independent expected가 필요합니다.

Q6

실패 test를 삭제했지만 broad suite는 green입니다. 승인할 수 있습니까?

정답 보기
아닙니다. 삭제된 test가 보호하던 contract가 여전히 유효한지 확인하고 동등하거나 더 강한 evidence가 없으면 blocker로 남깁니다.

Q7

좋은 finding의 여섯 핵심 요소는 무엇입니까?

정답 보기
claim, location, condition, impact, evidence, bounded action입니다. actual·expected를 명시하면 재현성과 수정 가능성이 더 높아집니다.

Q8

작성자가 comment를 resolved 처리했습니다. 다음 단계는 무엇입니까?

정답 보기
새 head를 고정하고 old→new fix diff와 original base→new 전체 diff를 검토한 뒤 focused·required checks를 current revision에서 다시 실행합니다.

Q9

`git revert`가 자동으로 복구하지 못하는 두 가지를 쓰십시오.

정답 보기
예: database data, 이미 발송된 email·message, 외부 vendor resource, cache·index, production traffic 상태. revert는 repository 역변경입니다.

Q10

Rollback rehearsal에서 restored tree와 baseline tree ID가 같으면 무엇을 증명합니까?

정답 보기
격리된 repository에서 candidate의 역변경 결과가 baseline content tree와 같음을 증명합니다. production deploy·database·external effect recovery는 증명하지 않습니다.

Q11

HIGH finding 하나가 미해결이고 나머지 checks는 PASS입니다. 기본 review decision은 무엇입니까?

정답 보기
REQUEST_CHANGES입니다. 팀 정책의 예외가 명시적으로 승인되지 않는 한 unresolved HIGH는 merge blocker입니다.

Q12

실행 권한이 없어 security integration test를 돌리지 못했습니다. 기록은 어떻게 합니까?

정답 보기
`NOT EXECUTED` 로 표시하고 이유, 필요한 environment·permission, owner, 다음 행동을 남깁니다. 생략하거나 PASS로 추정하지 않습니다.

22. 승인 전 최종 Checklist

Intent·range

- [ ] 원래 task source와 owner가 있다.
- [ ] actor·outcome·must preserve·non-goal이 있다.
- [ ] full base·head와 비교 direction이 있다.
- [ ] working tree와 evidence revision이 일치한다.

Surface·scope

- [ ] changed files·status·stat을 inventory했다.
- [ ] source·test·dependency·config·data·docs를 분류했다.
- [ ] 허용 범위 밖 변경은 이유와 owner 승인이 있다.
- [ ] 삭제된 protection을 별도 검토했다.

Behavior·contract

- [ ] positive·negative·boundary를 확인했다.
- [ ] caller·consumer·error path를 따라갔다.

- [ ] public API·schema·CLI·event 호환성을 확인했다.
- [ ] 인증과 인가를 구분했다.

Test·evidence

- [ ] candidate test의 oracle을 검토했다.
- [ ] independent contract test가 있다.
- [ ] 삭제·약화된 test가 없다.
- [ ] exact command·cwd·runtime·exit·output·revision이 있다.
- [ ] NOT EXECUTED·UNKNOWN·NOT REVIEWED를 숨기지 않았다.

Safety·dependency

- [ ] 민감 data source-to-sink를 추적했다.
- [ ] response·log·analytics·external sink를 확인했다.
- [ ] dependency 필요성·manifest·lock·source·license·owner를 봤다.
- [ ] external effect와 permission이 승인됐다.

Finding·decision

- [ ] claim·location·condition·actual·expected·impact·evidence·action이 있다.
- [ ] severity 근거가 impact·likelihood·blast·recovery에 연결된다.
- [ ] blocker와 suggestion을 구분했다.
- [ ] 새 head의 finding을 current evidence로 재검토했다.
- [ ] final decision과 unresolved owner가 명확하다.

Recovery

- [ ] undo·restore·revert·external recovery를 구분했다.
- [ ] code·schema·data·message·service plan이 있다.
- [ ] rehearsal은 격리 환경에서 fail closed로 실행했다.
- [ ] rehearsal이 증명하지 못한 범위를 기록했다.

23. 공식 자료와 확인 범위

OpenAI Codex

- Codex best practices: diff 검토, test와 verification, current evidence 원칙을 확인했습니다.
- Codex code review: base branch·uncommitted changes·commit·custom instruction review 흐름을 확인했습니다.
- Codex prompting: task·context·constraints·verification을 구체화하는 원칙을 확인했습니다.

Git

- git diff: endpoint 비교와 three-dot merge-base 의미를 확인했습니다.
- git status: working tree와 index 상태 확인 범위를 확인했습니다.
- git restore: working tree·index path 복원 의미를 확인했습니다.
- git revert: 기존 commit의 역변경을 새 이력으로 기록하는 의미를 확인했습니다.

Google Engineering Practices

- Code review: reviewer·author의 책임과 code health 기준을 확인했습니다.
- What to look for in a code review: design·functionality·complexity·tests·naming·comments·style·docs lens를 확인했습니다.
- The standard of code review: 완벽보다 code health 개선, test 자체의 검토 필요성을 확인했습니다.

GitHub

- Reviewing proposed changes: file-by-file review, diff와 commit 탐색을 확인했습니다.
- About pull request reviews: comment·approve·request changes 상태를 확인했습니다.
- Reviewing dependency changes: manifest·lock뿐 아니라 dependency source diff를 볼 필요를 확인했습니다.
- Helping others review your changes: 작은 변경·context·test evidence 제공 원칙을 확인했습니다.

Claude Code

- Checkpointing: checkpoint가 file edit 중심이며 shell·external change와 Git을 대체하지 않는 범위를 확인했습니다.
- Code review: 자동 finding·전문화된 분석·verification의 위치를 확인했습니다.

Security·secure development

- OWASP Secure Code Review Cheat Sheet: diff-based review, auth·input·data flow·secret·logging·dependency 검토 범위를 확인했습니다.

- NIST Secure Software Development Framework: 결과 중심의 secure development·verification·review 관점을 확인했습니다.

적용 원칙

이 manual의 명령과 OID는 격리된 synthetic 실습 repository에서 검증했습니다. 실제 프로젝트에서는 repository 정책, branch protection, data classification, incident·deployment runbook과 승인 owner를 우선합니다. 특히 production data·credential·외부 발송·migration은 본문 명령을 그대로 실행하지 말고 조직 절차와 권한을 확인합니다.