M04-04 · GIBALJA VISUAL MANUAL

외부 API와 비동기 작업 설계하기

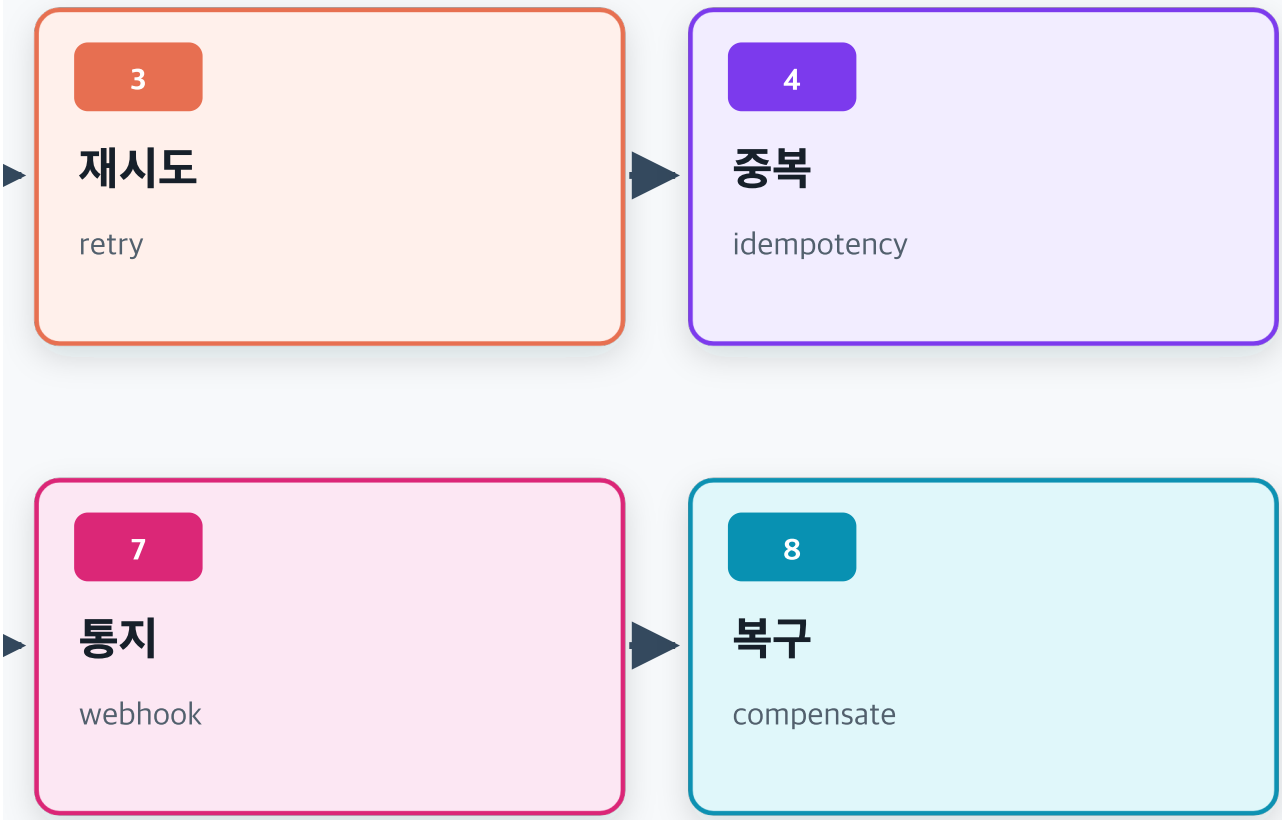
> **한 문장 목표:** 외부 시스템에 요청을 보냈다는 사실만 기록하지 않고, 언제 무엇이 완료됐는지, 응답을 잃었을 때 무엇을 조회할지, 같은 요청·message·event가 반복돼도 왜 결과가 한 번만 생기는지, 부분 실패를 어떻게 복구할지를 하나의 연계 계약으로 작성합니다.

난이도	개념	실습	셀프 테스트	최종 산출물
Level 1	60분	60분	15분	연계 계약서, job 상태도, retry·중복 방지표, webhook·보상 시나리오

결제 API가 2초 뒤 timeout됐습니다. 이것은 “결제 실패”가 아닙니다. 요청이 도착하지 않았을 수도 있고, 결제가 완료됐지만 응답만 잃었을 수도 있습니다. 좋은 계약은 `idempotency key`와 외부 결제 ID로 기존 결과를 확인하고, 자동 재시도·상태 조회·운영자 확인 중 다음 행동을 결정합니다.

그림 1. 외부 연계는 호출 한 번이 아니라 여덟 경계의 계약입니다. 어느 경계가 끝났는지 알아야 다음 행동이 안전해집니다.


1
호출
request
2
시간
timeout
5
접수
job
6
전달
queue
완료 계약 = business result + 상태 조회 +



+ 실패·중복·복구 + end-to-end evidence

0. 이 PDF를 공부하는 방법

1회차 · 그림만 읽기 · 20분

각 그림의 제목과 검은 결론 띠만 읽습니다. 다음 일곱 문장을 소리 내어 말합니다.

```
202는 완료가 아니라 접수다.
timeout은 실패가 아니라 결과 불명일 수 있다.
retry는 오류 처리 코드가 아니라 정책이다.
같은 의도는 같은 결과에 연결한다.
queue와 webhook은 같은 message를 다시 줄 수 있다.
outbox는 발행 틈을 줄이지만 consumer 중복은 남는다.
보상도 실패할 수 있는 별도 workflow다.
```

2회차 · 한 사례로 끝까지 추적하기 · 25분

결제·보고서 생성·AI 문서 분석 중 하나를 고릅니다. `request → external call → job → message → webhook → final result`에 같은 업무 ID를 적습니다.

3회차 · 실습기에서 실패 만들기 · 50분

외부 연계·비동기 작업 스튜디오를 열어 14개 시나리오를 실행합니다. 한 값만 바꾸고 결과가 왜 달라졌는지 말합니다.

4회차 · 내 기능 계약하기 · 40분

외부 API·비동기 작업 연계 명세서를 채웁니다. 마지막에 셀프 테스트를 풀고 정답과 비교합니다.

1. 외부 연계가 어려운 이유는 상대 시스템이 밖에 있기 때문입니다

우리 데이터베이스 transaction은 성공했는데 외부 문자 발송이 실패할 수 있습니다. 외부 결제는 성공했는데 우리 응답 저장이 실패할 수 있습니다. queue가 message를 두 번 전달할 수 있고 webhook 응답이 늦어 공급자가 같은 event를 다시 보낼 수 있습니다.

한 줄 요구	빠진 계약
결제 API를 호출합니다	timeout 뒤 결제 여부를 어떻게 확인합니까

한 줄 요구	빠진 계약
실패하면 세 번 retry합니다	어떤 실패를 누가 언제까지 다시 보냅니까
오래 걸리면 비동기로 합니다	접수·진행·완료·실패를 어디서 봅니까
webhook으로 알려 줍니다	누가 보냈는지와 재전송을 어떻게 검증합니까
queue가 알아서 처리합니다	중복·순서 역전·poison message는 어떻게 다룹니까
실패하면 rollback합니다	이미 끝난 외부 side effect를 무엇으로 보상합니까

30초 확인 1
외부 API timeout을 곧바로 업무 실패로 저장하면 어떤 중복이 생길 수 있습니까?

2. 먼저 여덟 계약 경계를 찾습니다

그림 1의 여덟 칸은 특정 표준의 architecture가 아닙니다. 기획자가 외부 연계에서 빠뜨리기 쉬운 결정을 찾기 위한 **YEONCORE 학습용 검수 프레임**입니다.

gate	결정 질문	남길 증거
1 호출	무엇을 어떤 인증·schema로 요청합니까	request ID·external operation
2 시간	한 번과 전체를 언제 중단합니까	attempt·elapsed·deadline
3 재시도	어떤 실패를 누가 몇 번 다시 보냅니까	reason·next attempt
4 중복	같은 업무 의도를 어떻게 식별합니까	key·fingerprint·result
5 접수	즉시 완료입니까, job 접수입니까	status·Location·job ID
6 전달	message가 반복·역전되면 어떻게 합니까	message ID·sequence·ack
7 통지	polling·webhook·event 중 무엇입니까	event ID·signature·delivery
8 복구	부분 실패를 어떻게 맞춥니까	outbox·DLQ·compensation

기획자의 한 문장 계약

```
[trigger]가 [operation]을 요청하면 [completion boundary]까지 기다린다.
[timeout / transient failure]이면 [retry policy]를 적용하되 [idempotency evidence]로 중복을 막는다.
접수 뒤에는 [job / message / event ID]로 상태를 추적하고,
[terminal failure / partial completion]이면 [DLQ / compensation / manual reconcile]로 복구한다.
```

3. 동기와 비동기는 업무 완료 경계로 구분합니다

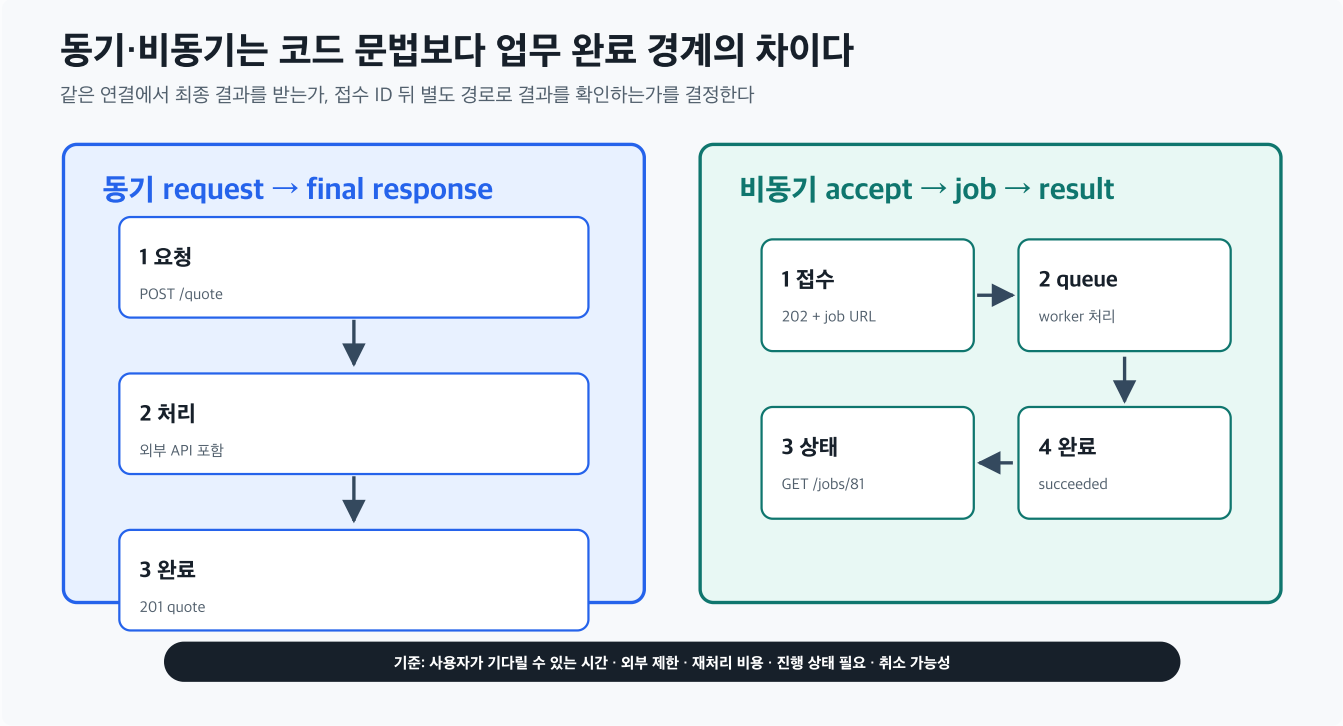
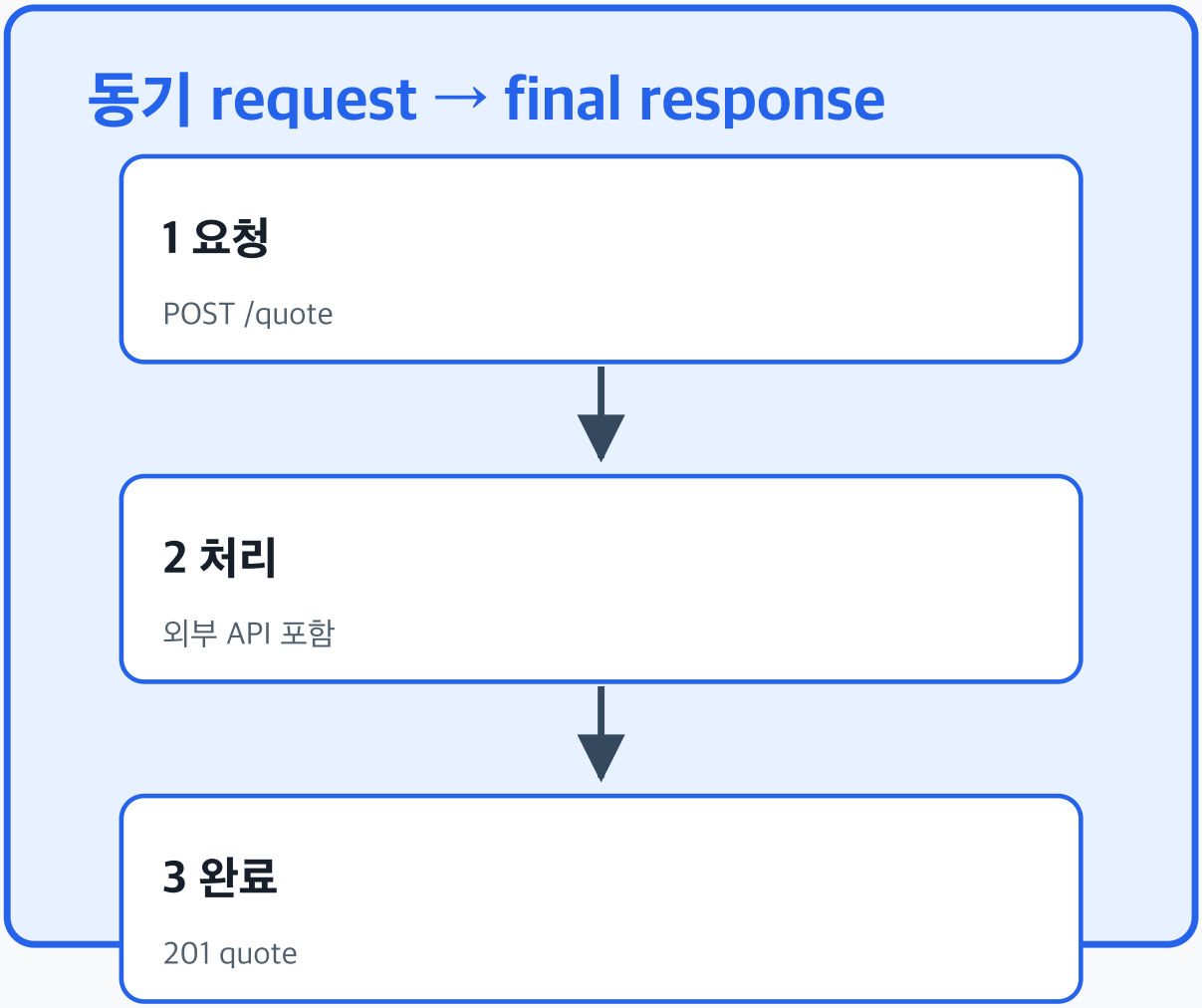
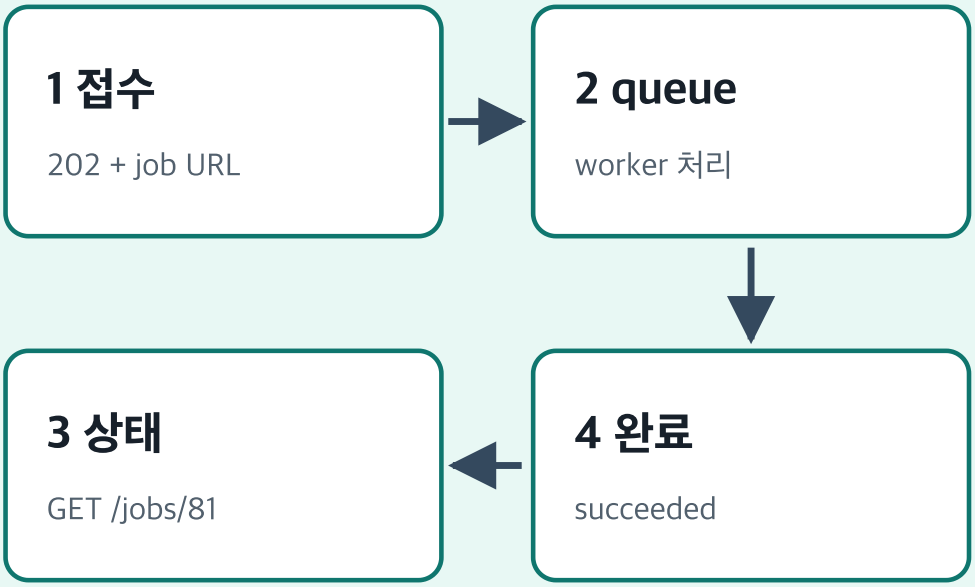


그림 2. `async/await` 문법이 아니라 client가 같은 교환에서 최종 업무 결과를 받는지가 핵심입니다.


동기 request → final response
1 요청
POST /quote
2 처리
외부 API 포함
3 완료
201 quote

비동기 accept → job → result



3.1. 동기 request-response

client는 같은 요청의 응답에서 최종 결과를 받습니다. 외부 호출까지 포함한 전체 시간이 사용자·gateway·server의 deadline 안에 들어와야 합니다.

```
POST /quotes HTTP/1.1
Content-Type: application/json

{"productId":"prd_81","quantity":2}

HTTP/1.1 201 Created
Location: /quotes/quo_81
Content-Type: application/json

{"quoteId":"quo_81","status":"ready","amount":42000}
```

3.2. 비동기 request-accept-process-result

초기 요청은 검증과 접수만 끝냅니다. 별도 worker가 나중에 처리하고 client는 job resource·webhook·event·stream으로 최종 결과를 확인합니다.

```
POST /exports HTTP/1.1
Idempotency-Key: exp_81

HTTP/1.1 202 Accepted
Location: /jobs/job_81
Retry-After: 3
Content-Type: application/json

{"jobId":"job_81","status":"queued","result":null}
```

RFC 9110의 `202 Accepted`는 요청이 처리를 위해 접수됐지만 처리가 완료되지 않았고 실제 처리가 시작되지 않을 수도 있음을 뜻합니다. `Location`과 `Retry-After`를 포함하는 위 형태는 학습용 연계 계약 패턴이며 모든 202 응답에 RFC가 강제하는 형식은 아닙니다.

3.3. 비동기로 바꿀 판단 기준

질문	동기 쪽	비동기 쪽
최종 결과 시간	짧고 예측 가능	길거나 분산이 큼
사용자 기다림	즉시 결정 필요	진행 상태로 충분

질문	동기 쪽	비동기 쪽
외부 rate limit	낮은 영향	queue로 흡수 필요
취소·진행률	단순	job 상태 필요
재시작	요청 전체 재시도	checkpoint·worker 재개
트래픽 폭주	곧바로 downstream에 전달	queue가 완충

30초 확인 2

비동기 API가 202를 반환한 직후 화면에 “완료”라고 표시하면 왜 틀렸습니까?

4. timeout 뒤에는 네 가지 결과가 가능합니다

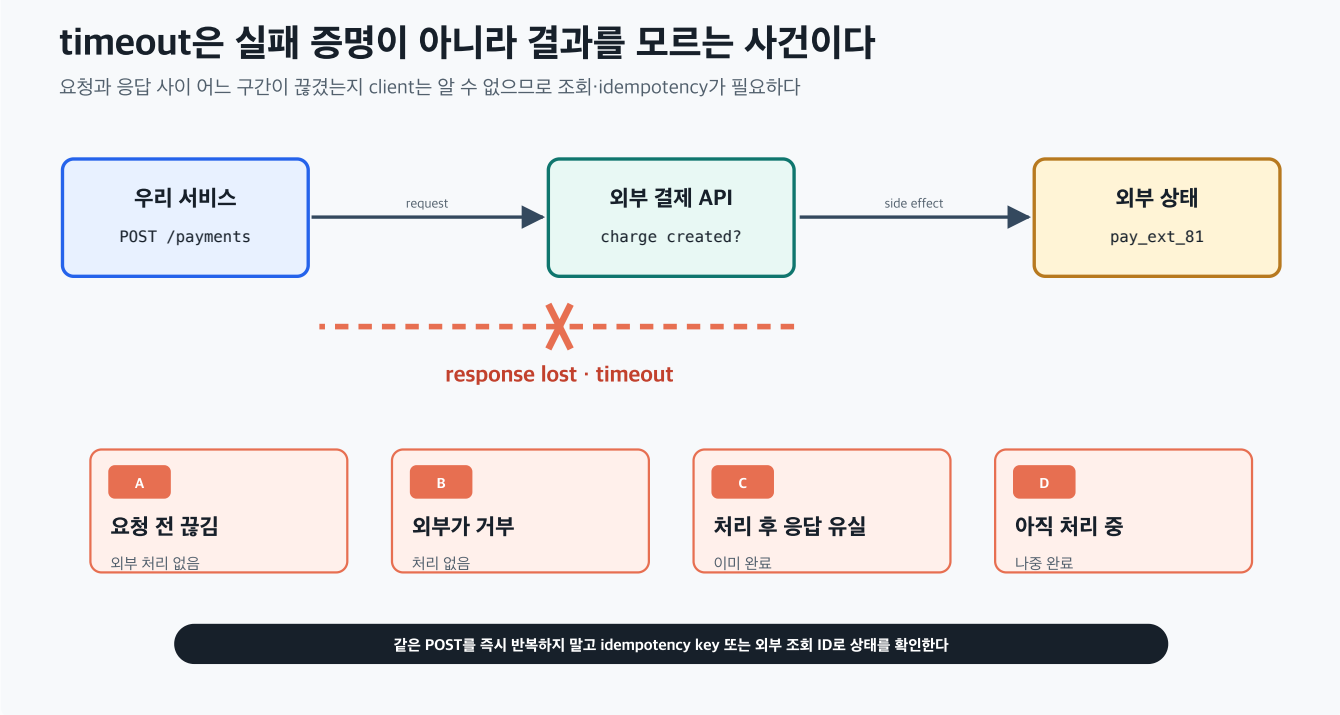


그림 3. client가 본 timeout 하나가 외부 시스템에서는 서로 다른 네 상태일 수 있습니다.


우리 서비스
POST /payments
request
외부 결
charge
response lost · timeout
A
요청 전 끊김
외부 처리 없음
B
외부가 거부
처리 없음
같은 POST를 즉시 반복하지 말고 idempoten


제 API
created?
side effect
외부 상태
pay_ext_81
C
처리 후 응답 유실
이미 완료
D
아직 처리 중
나중 완료
ncy key 또는 외부 조회 ID로 상태를 확인한다

4.1. timeout이 말해 주는 것

정해 둔 시간 안에 필요한 응답을 받지 못했다는 사실입니다. 외부 업무가 실패했다는 증명이 아닙니다.

실제 위치	외부 side effect	client가 본 결과	안전한 다음 행동
요청 전 연결 실패	없음	timeout·network error	조건부 retry
외부가 요청 거부	없음	응답 유실 가능	status·audit 조회
외부 처리 중	미정	timeout	같은 key 상태 확인
외부 완료·응답 유실	이미 있음	timeout	기존 resource 조회

4.2. 결제 사례

```
나쁜 규칙  : timeout → failed 저장 → 새 POST
위험       : 첫 결제는 완료됐는데 두 번째 결제가 생성됨

좋은 규칙  : payment_intent=ord_81, key=pay_ord_81 저장
             → 같은 key 조회 또는 재요청
             → 외부 payment ID와 우리 상태 reconcile
```

4.3. unknown을 별도 상태로 둡니다

`failed`와 `unknown`은 다릅니다. unknown은 자동 retry보다 먼저 조회·조정(reconciliation)이 필요할 수 있습니다.

상태	뜻	사용자 문구	내부 행동
failed	실패가 확인됨	처리하지 못했습니다	수정·보상·재요청
unknown	결과를 아직 모름	처리 상태를 확인 중입니다	외부 조회·reconcile
pending	외부 처리 중	처리 중입니다	polling·webhook 대기
succeeded	완료 증거 있음	완료됐습니다	resource 표시

5. timeout은 전체 예산으로 설계합니다

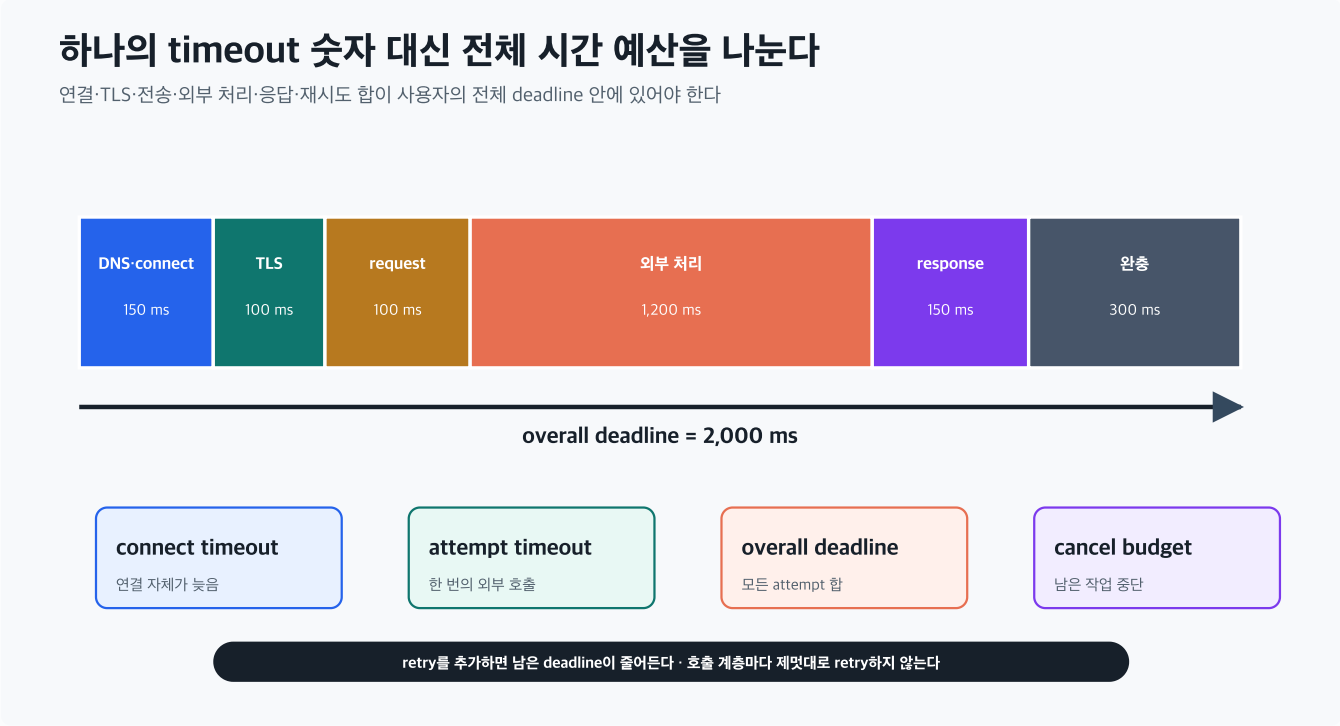
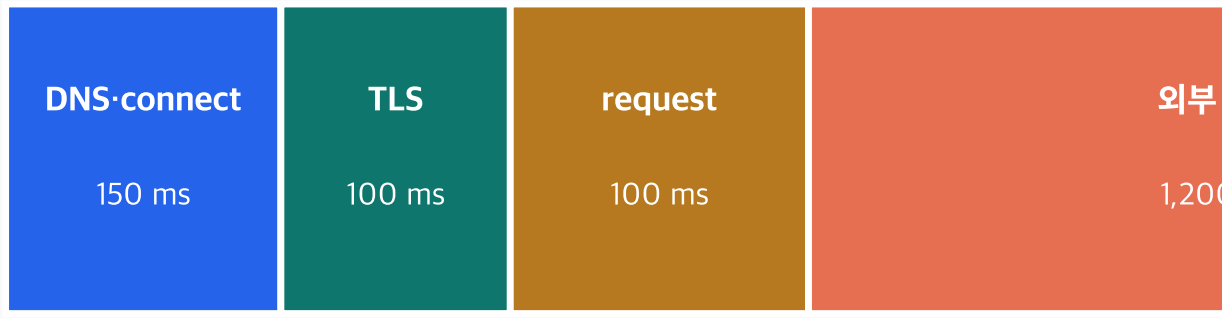


그림 4. 한 번의 socket timeout만 정하면 연결·처리·재시도 합이 사용자 deadline을 넘기기 쉽습니다.



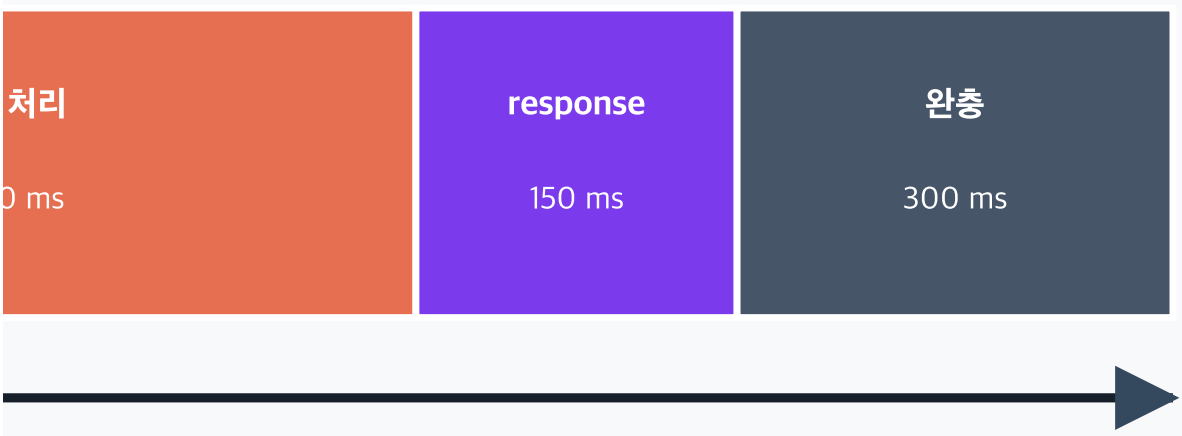
overall deadlin

connect timeout

연결 자체가 늦음

attempt timeout

한 번의 외부 호출



e = 2,000 ms

overall deadline

모든 attempt 합

cancel budget

남은 작업 중단

5.1. 네 개 시간을 나눕니다

시간	질문	예시
connect timeout	연결·TLS를 언제 포기합니까	250 ms
attempt timeout	외부 호출 한 번을 언제 중단합니까	1,500 ms
overall deadline	모든 attempt를 합해 언제 끝냅니까	4,000 ms
cancellation budget	남은 작업 정리 시간을 확보했습니까	300 ms

숫자는 예시입니다. 실제 값은 외부 서비스 지연 분포, 네트워크, 사용자 경험, 비용, 공급자 SLA를 관찰해 정합니다.

5.2. 단계별 예산표

단계	예산	실제 p95	초과 시 행동	책임자
연결·TLS				
요청 전송				
외부 처리				
응답 수신				
retry 대기				
결과 저장				

5.3. retry amplification을 계산합니다

gateway, service A, service B가 각각 세 번 retry하면 최악의 경우 아래 계층 호출이 크게 불어날 수 있습니다. 그래서 retry 주체와 최대 attempt를 계약으로 한 곳에 둡니다.

```
호출 계층마다 3 attempts
gateway 3 × service A 3 × service B 3 = 최대 27회 downstream attempt
```

30초 확인 3

overall deadline이 2초인데 첫 attempt가 1.9초 걸렸다면 같은 timeout으로 두 번째 attempt를 시작해도 됩니까?

6. retry는 안전성·일시성·예산으로 결정합니다

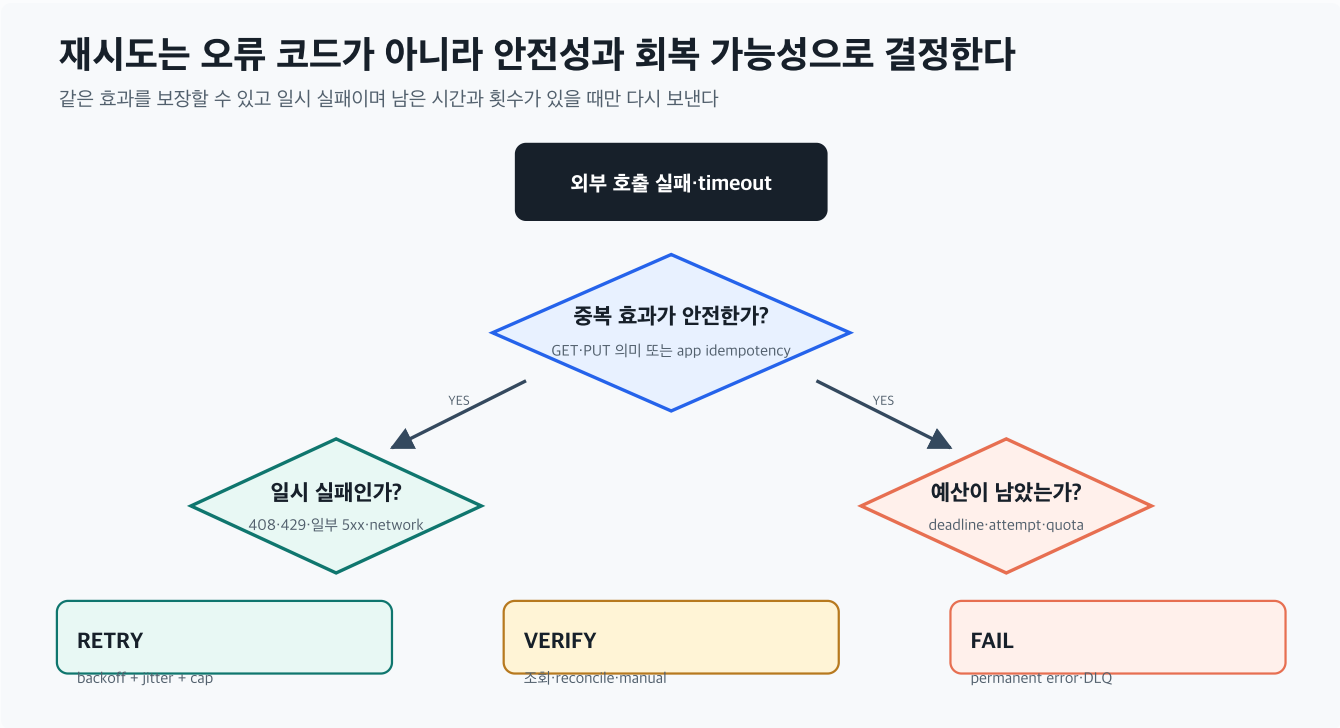
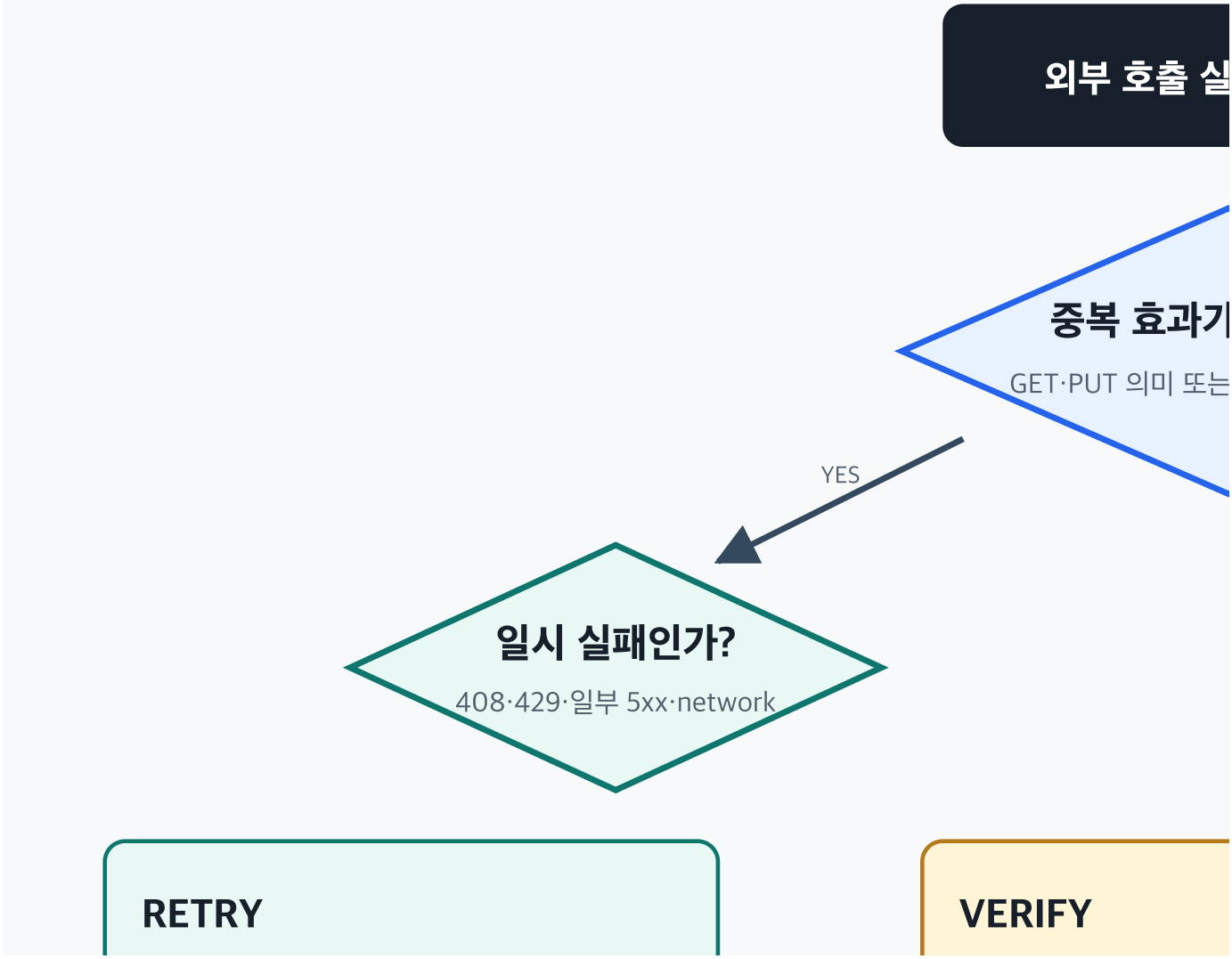
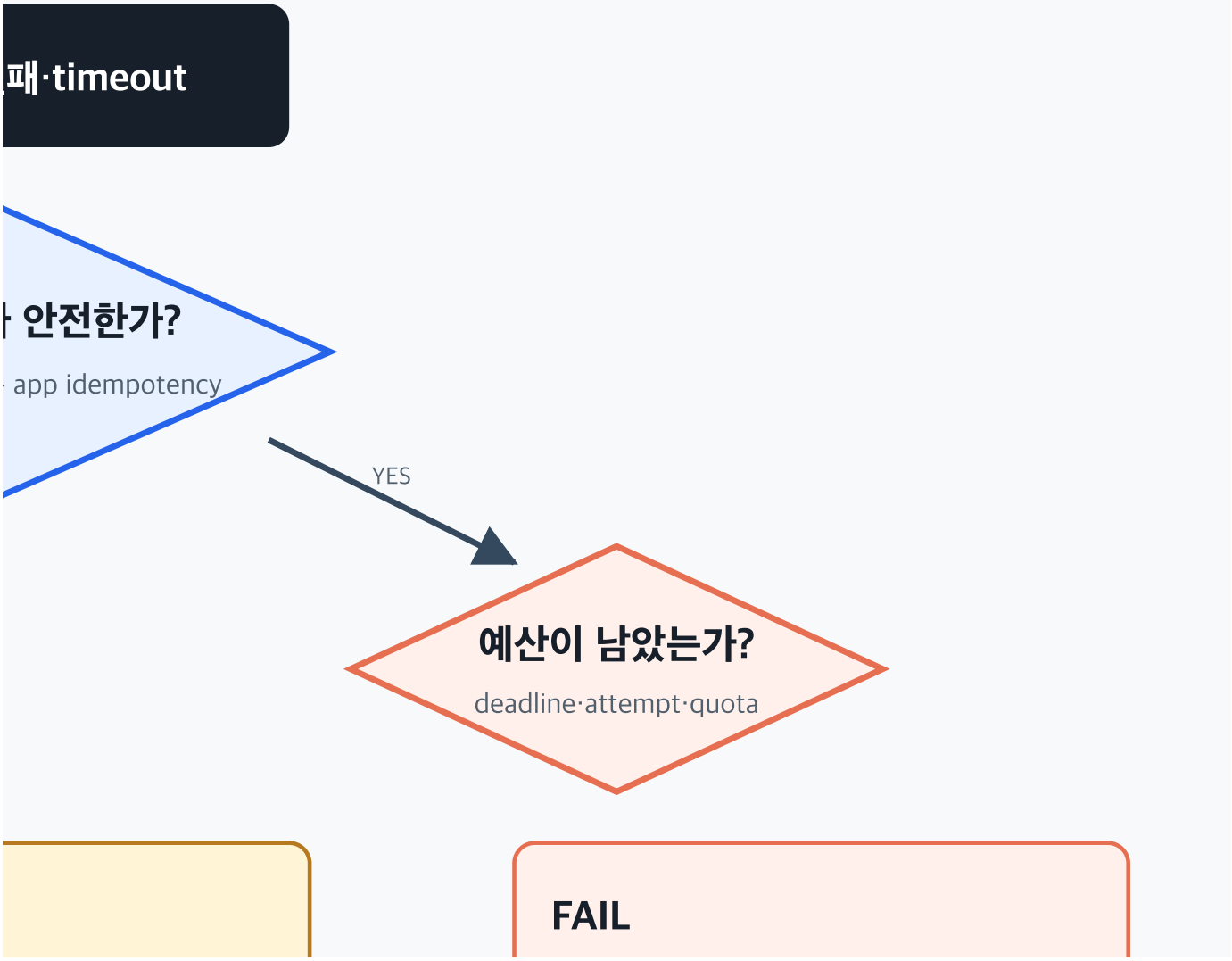


그림 5. “5xx면 retry”처럼 한 조건만 보지 않습니다. 같은 효과가 안전하고 회복 가능하며 예산이 남아야 합니다.


외부 호출 실
중복 효과기
GET·PUT 의미 또는
YES
일시 실패인가?
408·429·일부 5xx·network
RETRY
VERIFY


패·timeout
안전한가?
app idempotency
YES
예산이 남았는가?
deadline·attempt·quota
FAIL

6.1. HTTP method 의미와 업무 안전성을 구분합니다

RFC 9110에서 safe method는 읽기 성격이고, idempotent method는 같은 의도로 여러 번 요청해도 의도된 server 효과가 한 번과 같다는 의미입니다. 응답 코드와 로그까지 완전히 같다는 뜻은 아닙니다.

method	RFC 의미	일반 자동 retry 판단
GET·HEAD·OPTIONS·TRACE	safe이며 idempotent	일시 실패·예산 안에서 가능
PUT·DELETE	idempotent	전제조건·외부 effect 확인 뒤 가능
POST	기본적으로 idempotent 아님	application idempotency 계약 필요

특정 `POST /payments`가 key와 fingerprint로 중복을 막는다면 그 operation은 application 수준에서 retry-safe하게 만들 수 있습니다. 이것을 모든 POST의 일반 성질로 확대하면 안 됩니다.

6.2. 실패 분류표

신호	보통 의미	자동 retry	주의
connect failure	요청 미도달 가능	조건부	실제 전송 여부를 client가 모를 수 있음
408	요청 timeout	조건부	server 처리 여부 계약 확인
429	rate limit	조건부	`Retry-After`·quota 존중
502·503·504	일시 upstream 장애 가능	조건부	모든 5xx가 일시 오류는 아님
400·401·403·404·409·422	입력·인증·정책·상태 문제	보통 안 함	값·권한·현재 상태를 바꿔야 함
schema parsing failure	영구 오류 가능	안 함	DLQ 또는 수정 필요

RFC 6585의 `429 Too Many Requests` 응답은 조건 설명을 담고 `Retry-After`를 포함할 수 있습니다. `Retry-After`가 없을 때의 기본 backoff도 client 계약에 적습니다.

6.3. retry policy 문장

```
caller = integration worker
retryable = connect error, 429, 502, 503, 504
non-retryable = 400, 401, 403, 404, 409, 422, schema error
attempts = initial 1 + retry 2
deadline = 10 seconds
delay = exponential backoff + full jitter, cap 4 seconds
idempotency = required for all mutating operations
exhausted = job.failed + problem detail + operator alert
```

7. backoff와 jitter로 재시도 폭주를 줄입니다

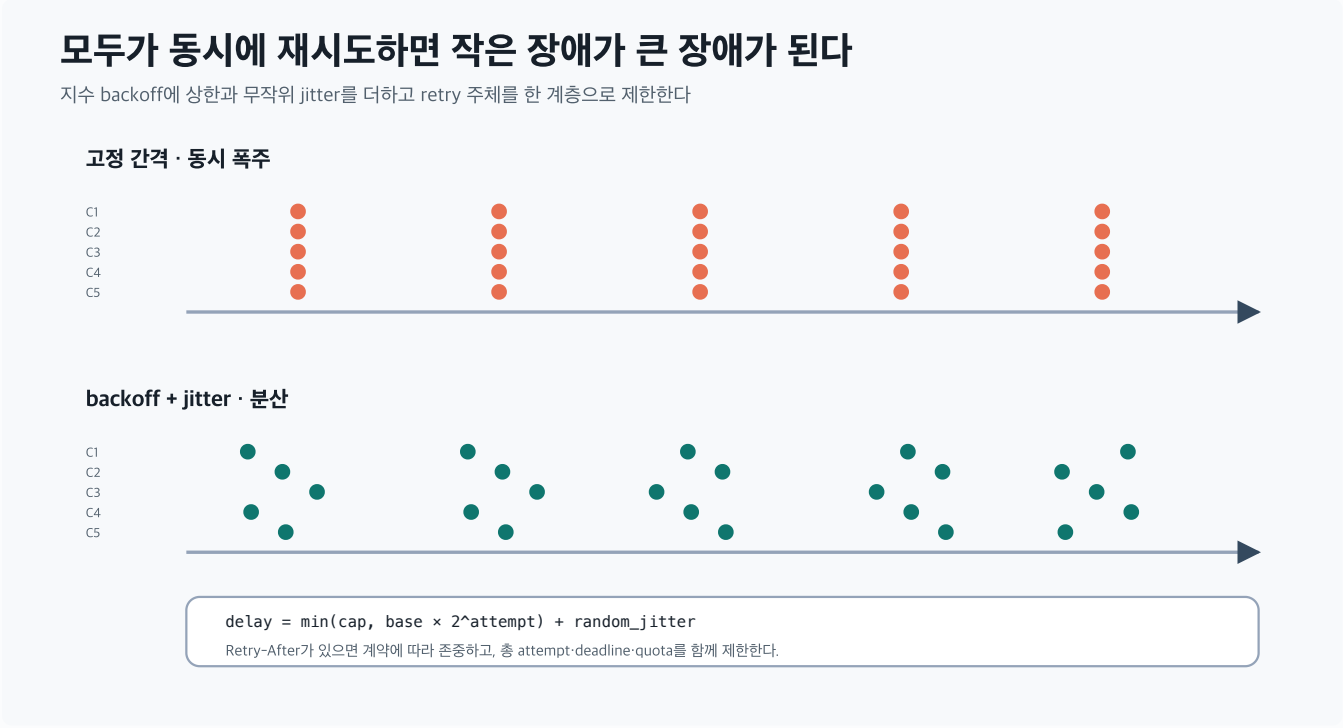
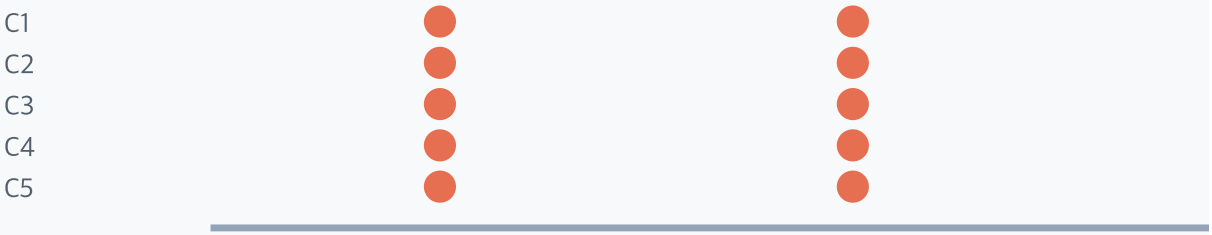


그림 6. 같은 간격은 수많은 client를 다시 같은 순간에 모읍니다. jitter는 시점을 흩어 놓습니다.

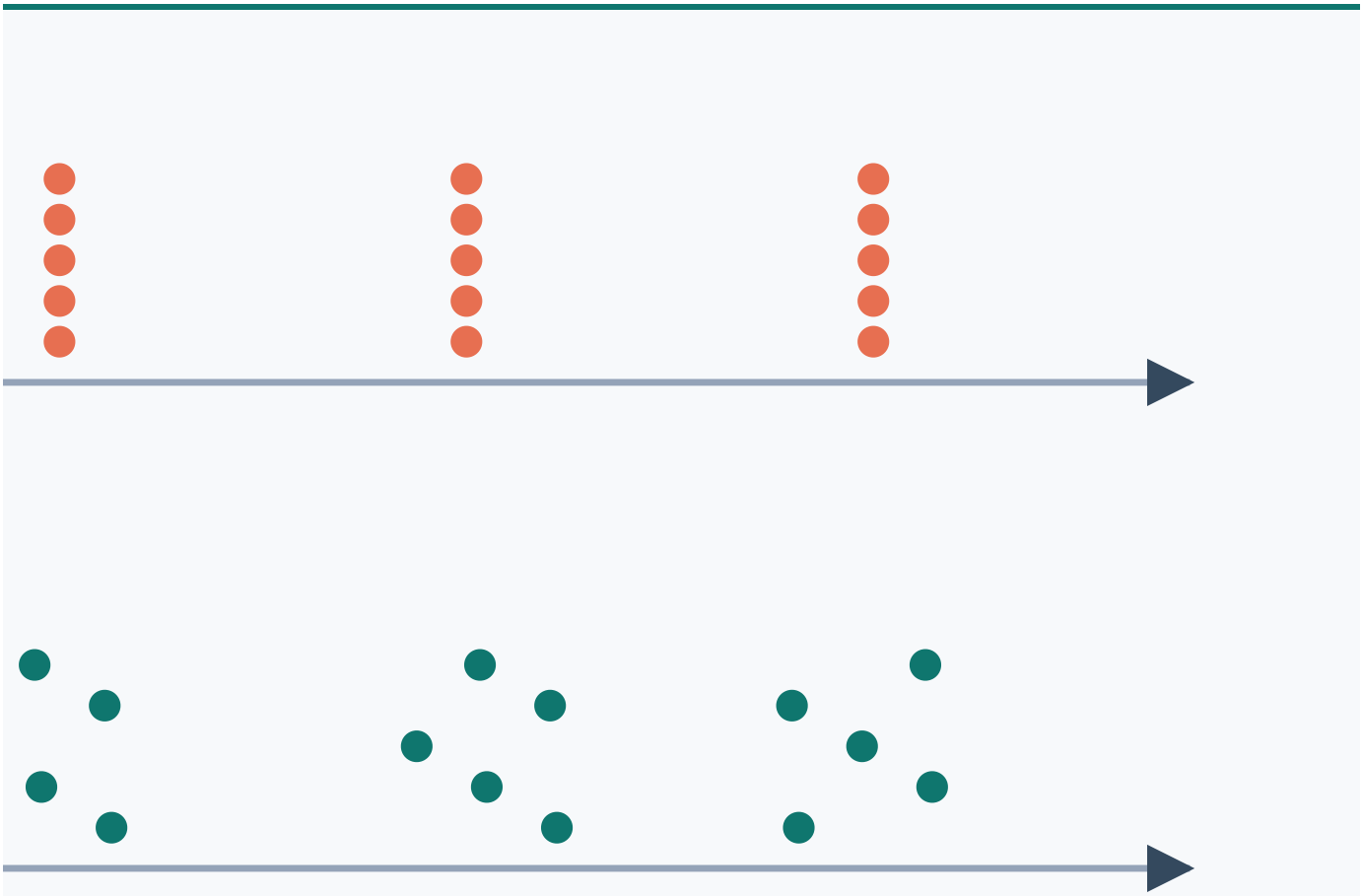
고정 간격 · 동시 폭주



backoff + jitter · 분산


```
delay = min(cap, base × 2^attempt) + random_jitt
```

Retry-After가 있으면 계약에 따라 존중하고, 총 attempt·deadline·quota



ter

를 함께 제한한다.

7.1. 학습용 계산식

```
base delay       = 250 ms
exponential      = base × 2^attempt
cap              = 4,000 ms
scheduled delay  = min(cap, exponential) + random jitter
```

실제 SDK의 retry mode와 jitter 방식은 제품마다 다릅니다. 직접 구현하기 전에 client library·SDK의 공식 retry 정책과 중복 계층을 확인합니다.

7.2. 재시도 예산표

attempt	실패	기본 delay	jitter 범위	남은 deadline	실행
1	503	0 ms	-	10,000 ms	완료
2	503	500 ms	0~500 ms	8,900 ms	예정
3	429	`Retry-After: 3`	0~500 ms	5,100 ms	계약 확인
4	-	-	-	-	cap 초과·중단

7.3. circuit breaker와 retry를 혼동하지 않습니다

retry는 현재 요청을 다시 시도합니다. circuit breaker는 실패가 계속될 때 새 호출을 잠시 차단해 downstream과 caller를 보호하는 운영 정책입니다. 두 정책의 상태·시간·fallback을 따로 정의합니다.

8. idempotency는 key 하나보다 넓은 계약입니다

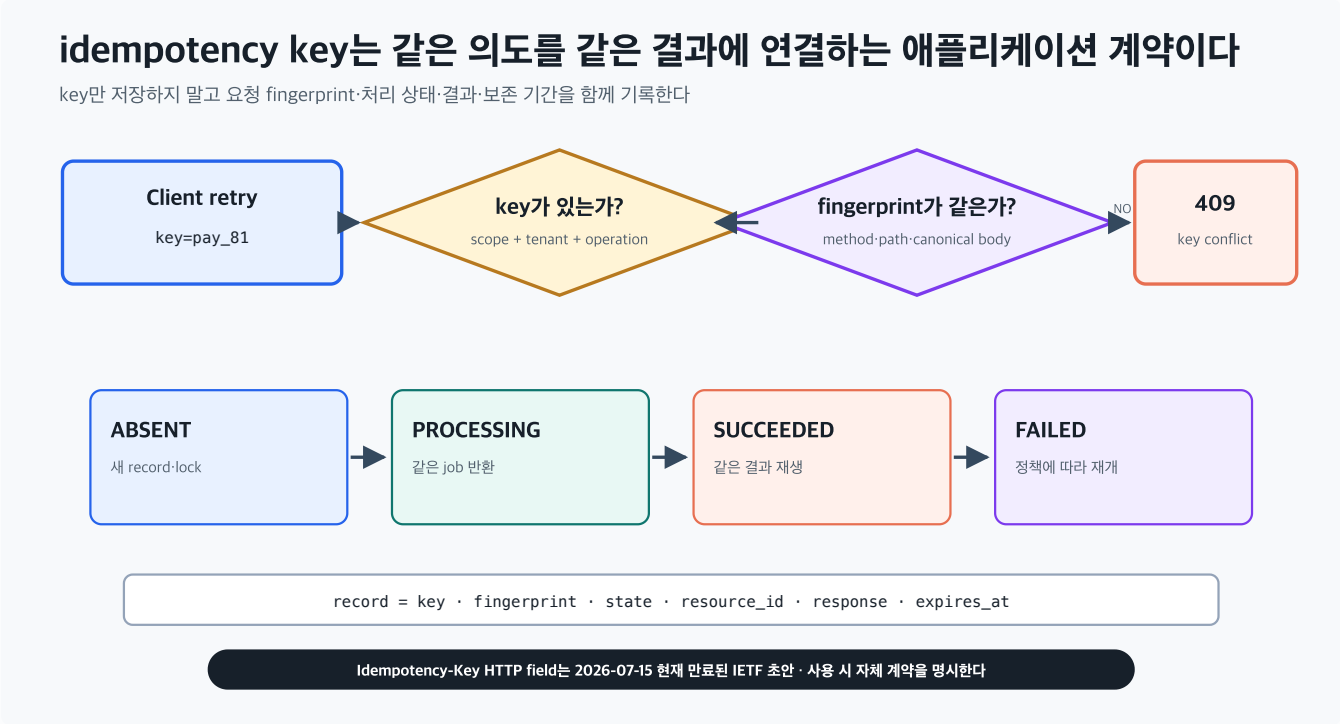
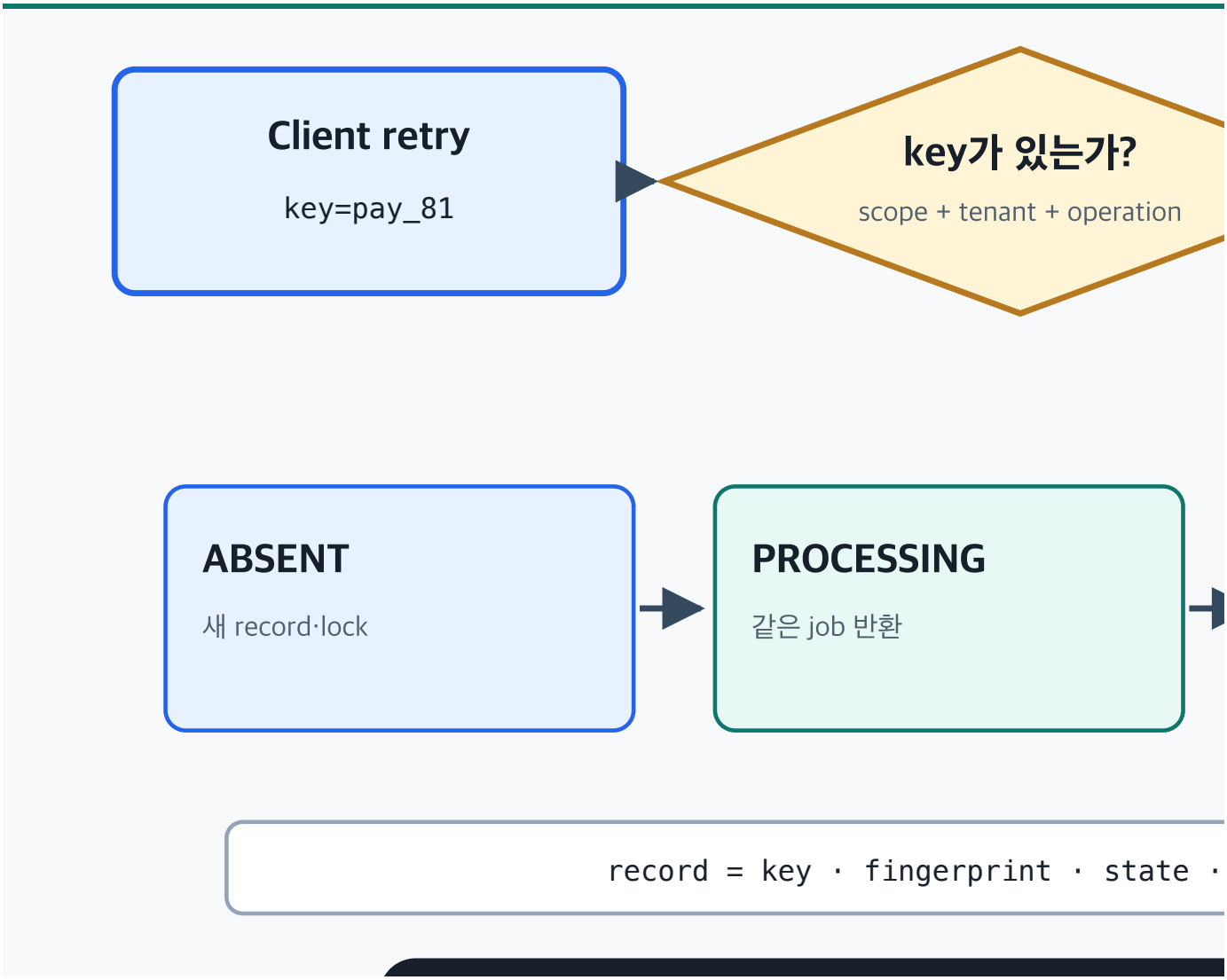
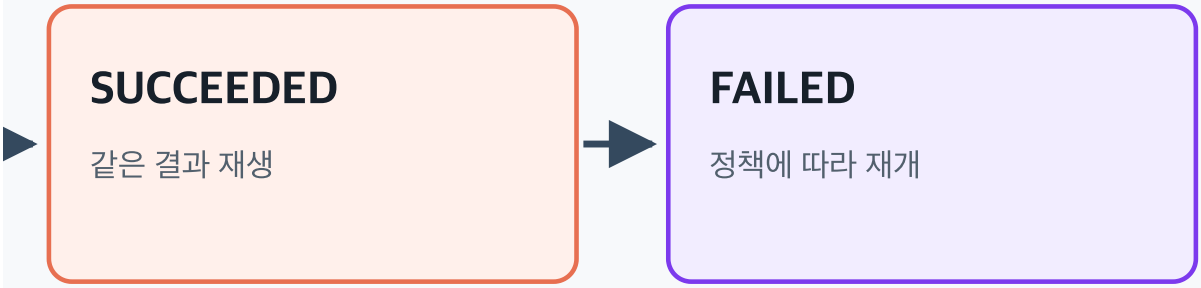


그림 7. 같은 key가 같은 요청이면 기존 상태·결과를 재생하고, 다른 요청이면 충돌로 거부합니다.


Client retry
key=pay_81
key가 있는가?
scope + tenant + operation
ABSENT
새 record·lock
PROCESSING
같은 job 반환
record = key · fingerprint · state ·




```
resource_id · response · expires_at
```

8.1. 2026-07-15 현재 표준 상태

`Idempotency-Key` HTTP header는 널리 쓰이는 구현 관행이지만, 확인일 현재 IETF `draft-ietf-httpapi-idempotency-key-header-07` 은 **Expired Internet-Draft**입니다. 보편 RFC처럼 말하지 말고 provider·consumer 사이의 자체 계약으로 이름·형식·scope·보존 시간을 정의합니다.

8.2. 최소 dedupe record

```
{
  "key": "pay_ord_81",
  "scope": "tenant_1:POST:/payments",
  "fingerprint": "sha256:canonical-request",
  "state": "processing",
  "resourceId": "pay_ext_81",
  "httpStatus": 202,
  "responseRef": "/jobs/job_81",
  "createdAt": "2026-07-15T20:15:00+09:00",
  "expiresAt": "2026-07-16T20:15:00+09:00"
}
```

8.3. 네 가지 요청 처리

key 상태	fingerprint	행동	응답 예
없음	새 요청	원자적으로 record 만들고 처리	202·201
processing	같음	두 번째 실행 금지·기존 job 반환	202
succeeded	같음	기존 결과 재생	200·201
존재	다름	key 충돌	409 problem

8.4. 보존 기간이 짧으면 어떤 일이 생깁니까

dedupe record가 삭제된 뒤 늦은 retry가 오면 새 요청으로 보일 수 있습니다. client retry window, 공급자 webhook retry 기간, 업무 취소 기간보다 보존이 짧지 않은지 검토합니다.

30초 확인 4
같은 idempotency key에 금액만 다른 결제 요청이 오면 기존 결과를 반환해야 합니까?

9. 202 뒤에는 job resource가 필요합니다

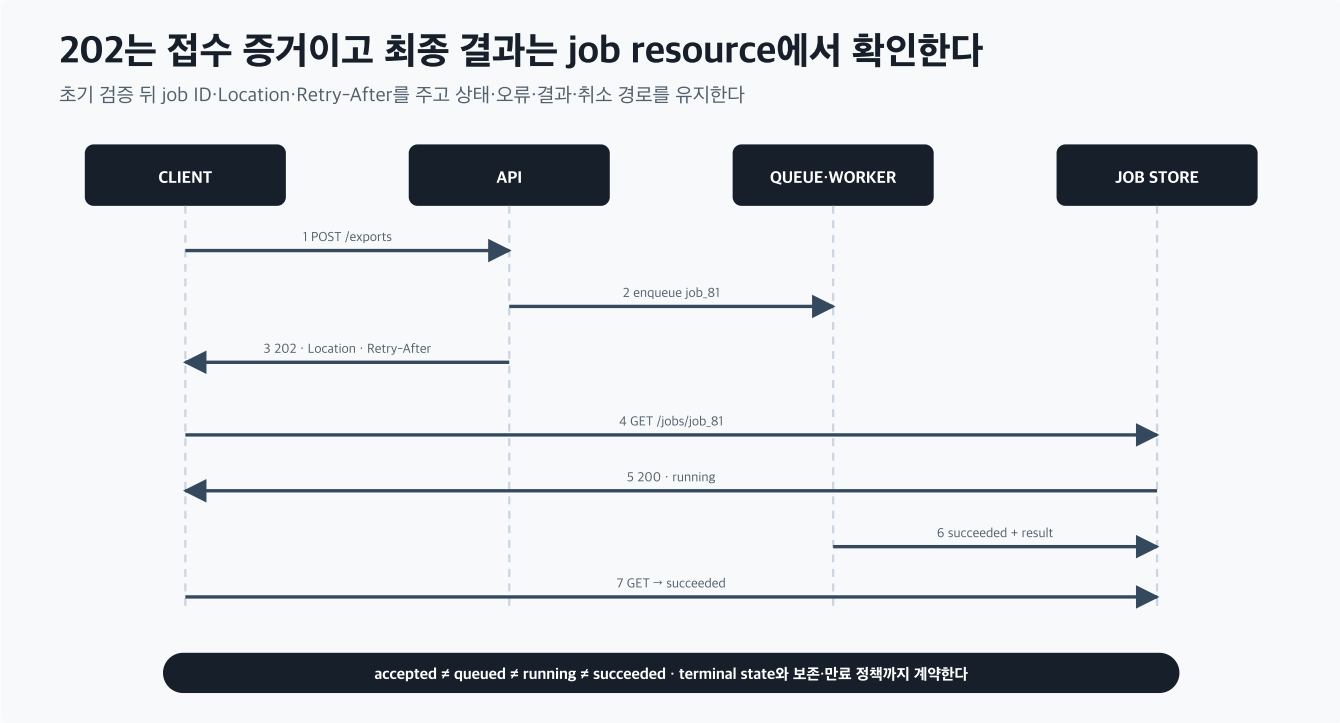
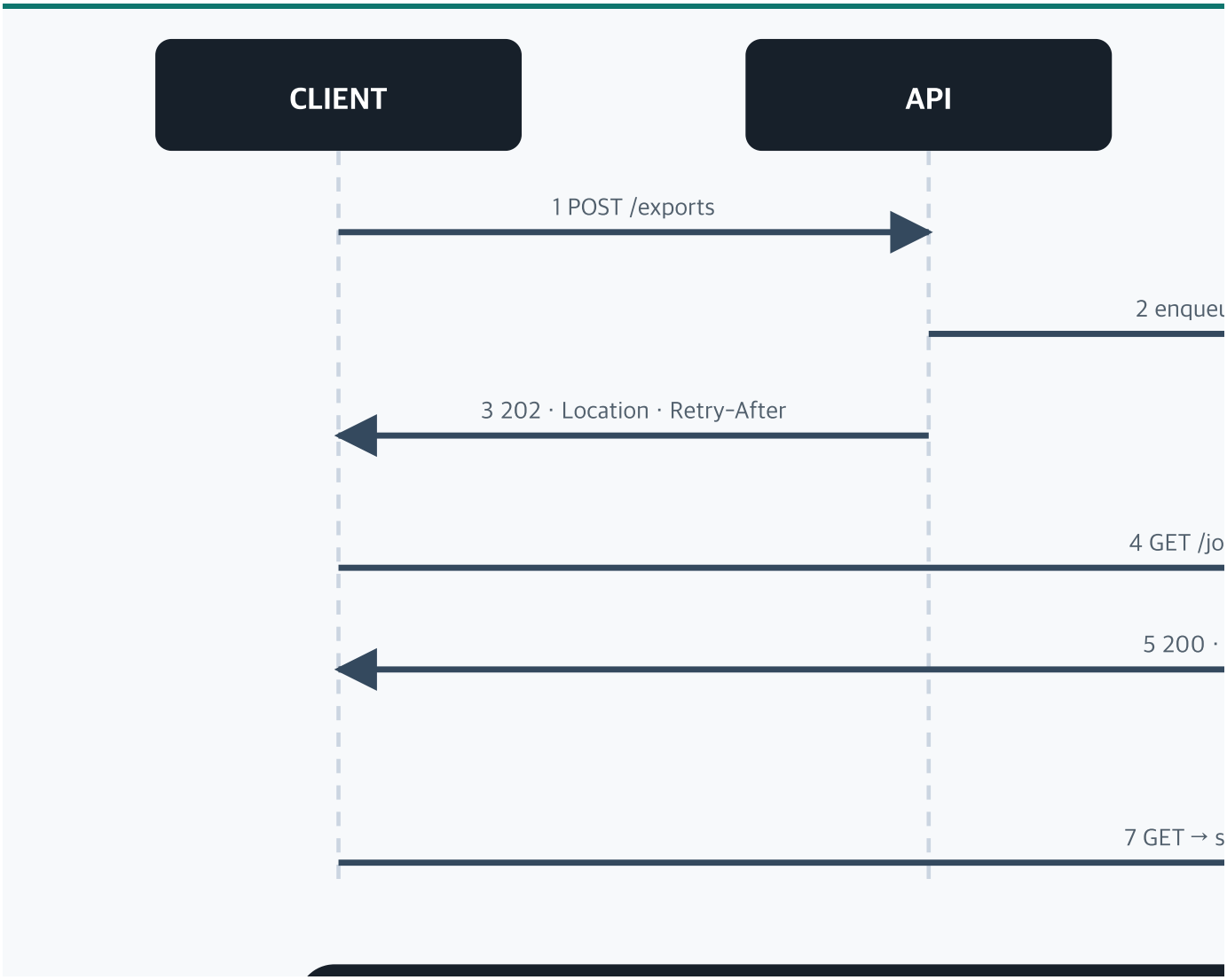
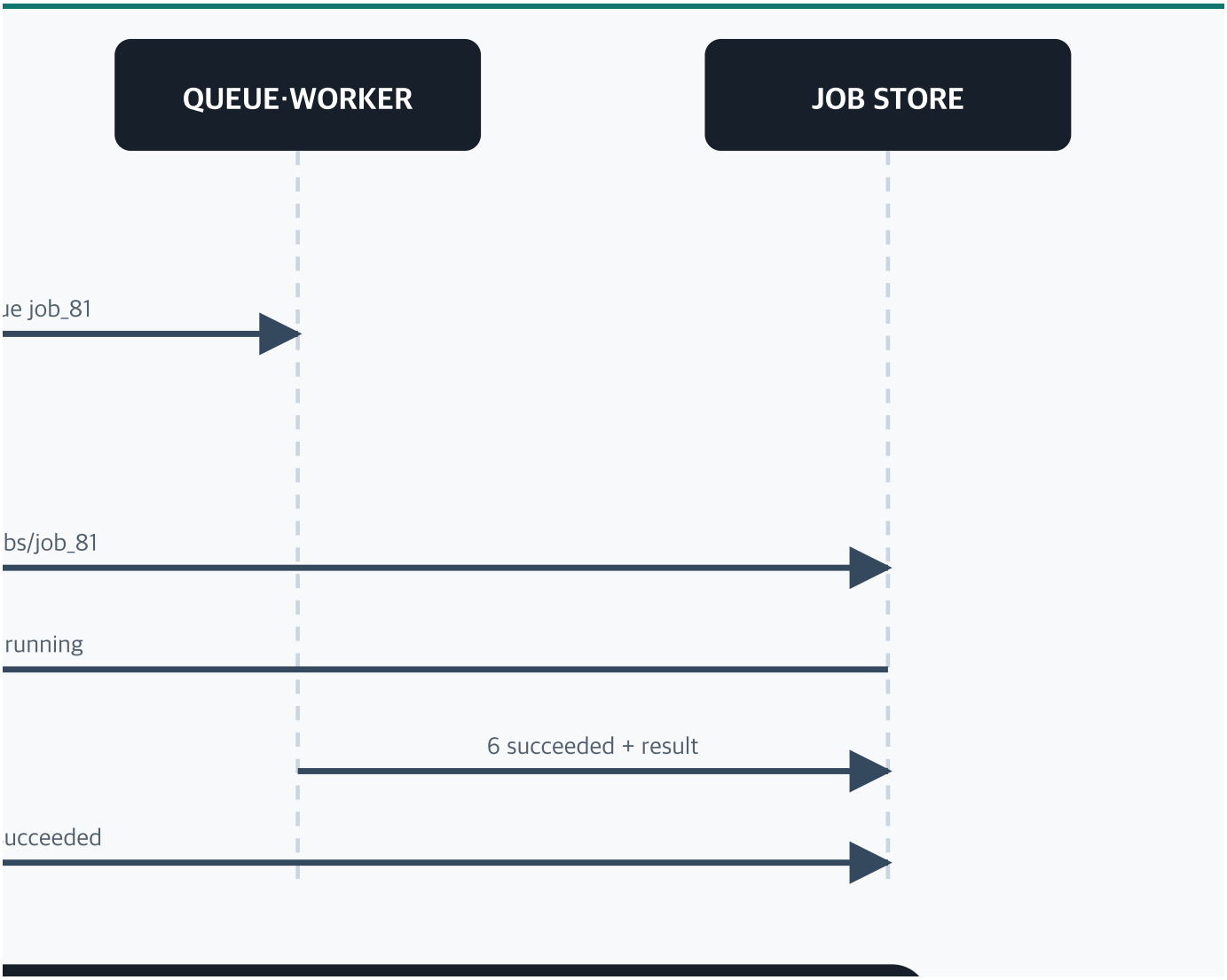


그림 8. 접수 응답, queue 처리, job 상태 저장, 최종 결과 조회가 서로 다른 시점에 일어납니다.


CLIENT
API
1 POST /exports
2 enqueu
3 202 · Location · Retry-After
4 GET /jo
5 200 ·
7 GET → s


QUEUE·WORKER
JOB STORE
ue job_81
bs/job_81
running
6 succeeded + result
ucceeded

9.1. 접수 전에 할 일

202를 빨리 반환한다는 이유로 기본 검증을 worker까지 미루지 않습니다.

접수 전에 확인	접수 뒤 worker에서 확인
인증·권한	외부 시스템 현재 상태
schema·필수값	큰 파일 처리
idempotency key·fingerprint	모델 추론·보고서 생성
명백한 업무 불가 조건	재시도 가능한 외부 호출
job record·queue enqueue 가능성	진행률·checkpoint

9.2. job 응답 예

```
{
  "jobId": "job_81",
  "type": "monthly_export",
  "status": "running",
  "createdAt": "2026-07-15T20:15:00+09:00",
  "updatedAt": "2026-07-15T20:15:08+09:00",
  "attempt": 1,
  "progress": {"completed": 38, "total": 100},
  "result": null,
  "error": null,
  "expiresAt": "2026-07-22T20:15:00+09:00",
  "links": {
    "self": "/jobs/job_81",
    "cancel": "/jobs/job_81"
  }
}
```

9.3. 오류는 job 안의 terminal result입니다

비동기 작업이 실패했을 때 초기 POST 응답을 과거로 돌아가 500으로 바꿀 수 없습니다. job resource에 `failed`와 machine-readable error를 기록합니다. HTTP 오류 표현이 필요하면 RFC 9457 Problem Details 구조를 재사용할 수 있습니다.

```
{
  "status": "failed",
  "error": {
    "type": "https://api.example.com/problems/export-source-unavailable",
    "title": "원본 데이터를 읽을 수 없습니다",
    "status": 503,
    "detail": "외부 원본이 retry budget 안에 회복되지 않았습니다",
    "instance": "/jobs/job_81"
  }
}
```

10. job 상태와 전이를 계약합니다

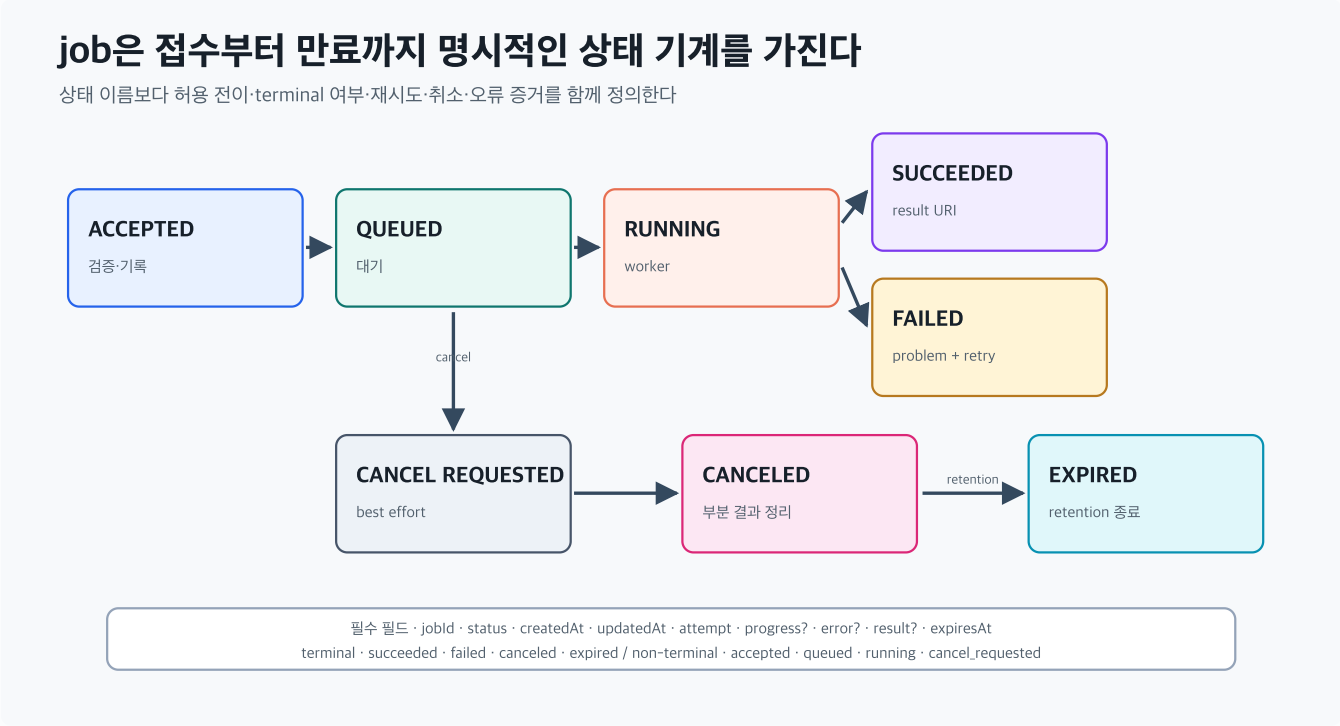
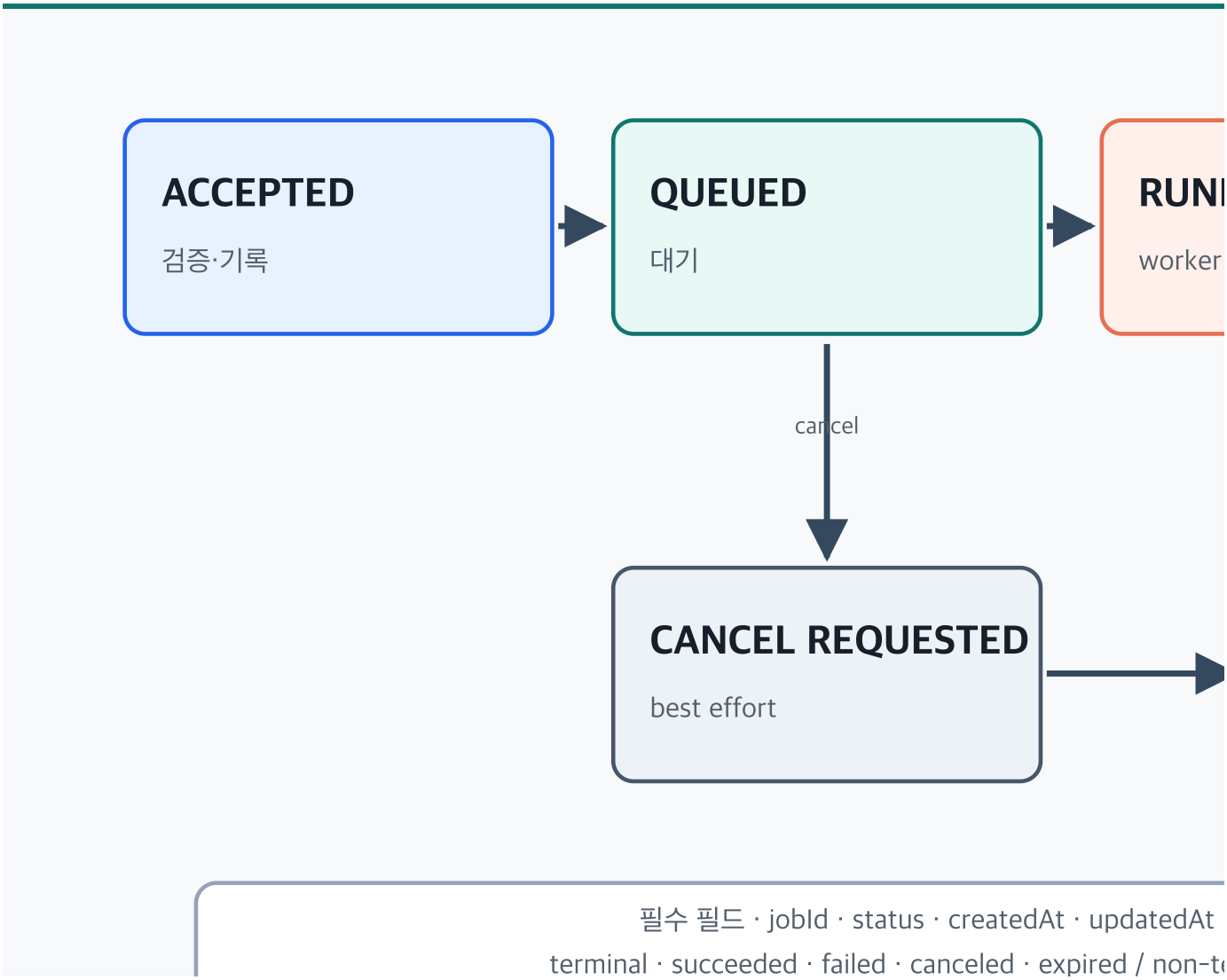
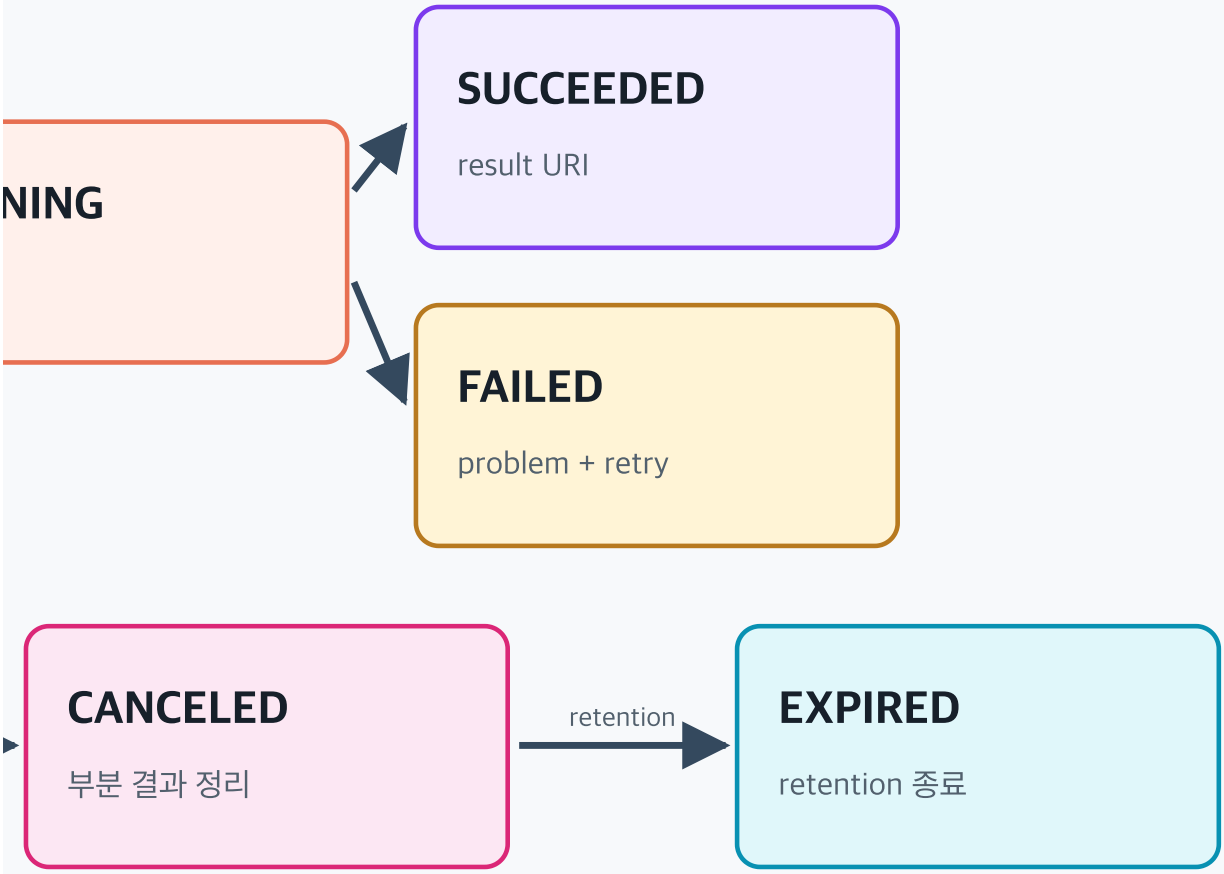


그림 9. 상태 이름만 나열하지 말고 어떤 상태에서 어디로 갈 수 있는지와 terminal 여부를 정의합니다.


ACCEPTED
검증·기록
QUEUED
대기
RUNI
worker
cancel
CANCEL REQUESTED
best effort
필수 필드 · jobId · status · createdAt · updatedAt
terminal · succeeded · failed · canceled · expired / non-t



· attempt · progress? · error? · result? · expiresAt

erminal · accepted · queued · running · cancel_requested

10.1. 권장 학습 상태

이 상태 집합은 범용 표준이 아니라 연계 명세를 시작하기 위한 학습 모델입니다. 제품의 실제 상태와 맞춥니다.

상태	terminal	사용자 의미	허용 다음 전이
accepted	아니요	요청 접수	queued·failed·canceled
queued	아니요	실행 대기	running·failed·canceled
running	아니요	처리 중	succeeded·failed·cancel_requested
cancel_requested	아니요	취소 시도 중	canceled·succeeded·failed
succeeded	예	완료	expired
failed	예 또는 정책별	실패	retry job·expired
canceled	예	취소 완료	expired
expired	예	조회 보존 종료	없음

10.2. 취소는 best effort일 수 있습니다

외부 결제가 이미 완료됐거나 worker가 취소 신호를 받기 전에 마지막 단계를 끝낼 수 있습니다. `cancel_requested`와 `canceled`를 구분하고, 취소가 불가능한 지점 뒤에는 보상 행동을 정의합니다.

10.3. progress는 거짓 정밀도를 피합니다

실제 총량을 모르면 `37%`를 만들지 않습니다. `phase=extracting`, `processed=38`, `total=null`처럼 관찰 가능한 정보를 제공합니다.

11. polling·webhook·event·stream을 선택합니다

그림 10. 통지 채널은 client가 받을 수 있는 연결, 허용 지연, 구독 수, 보안 운영 능력으로 고릅니다.

POLLING

client → status

방화벽 단순 · 지연·부하

`GET /jobs/{id}`

WEBHOOK

provider → callback

빠른 push · 서명·재전송

`POST callback`

browser·callback 불가

POLLING

B2B·빠른 통지

WEBHOOK + POLLING

어떤 채널도 유실이 0은 아니다 · event ID·dedu

EVENT

broker → consumers

다수 구독 · 운영 복잡

`topic / queue`

STREAM

open connection

실시간 · 연결 관리

`SSE / WebSocket`

다수 내부 consumer

EVENT

초단위 진행 화면

STREAM

pe·status 조회 fallback으로 최종 상태를 맞춘다

채널	시작 방향	잘 맞는 상황	반드시 정할 실패 경로
polling	client → status API	browser·방화벽 단순 환경	poll 간격·429·만료
webhook	provider → callback	B2B·빠른 결과 통지	서명·재전송·SSRF·fallback
event	producer → broker → consumer	내부 다수 구독·fan-out	schema·ordering·DLQ
SSE·WebSocket	열린 연결	실시간 진행 화면	재연결·last event·fallback

11.1. polling 계약

```
status URL       /jobs/{jobId}
initial interval 2 seconds
backoff          2 → 4 → 8 seconds, cap 15 seconds
429 handling     Retry-After 우선
terminal         succeeded / failed / canceled / expired
retention        7 days
fallback         지원 화면에서 job ID 조회
```

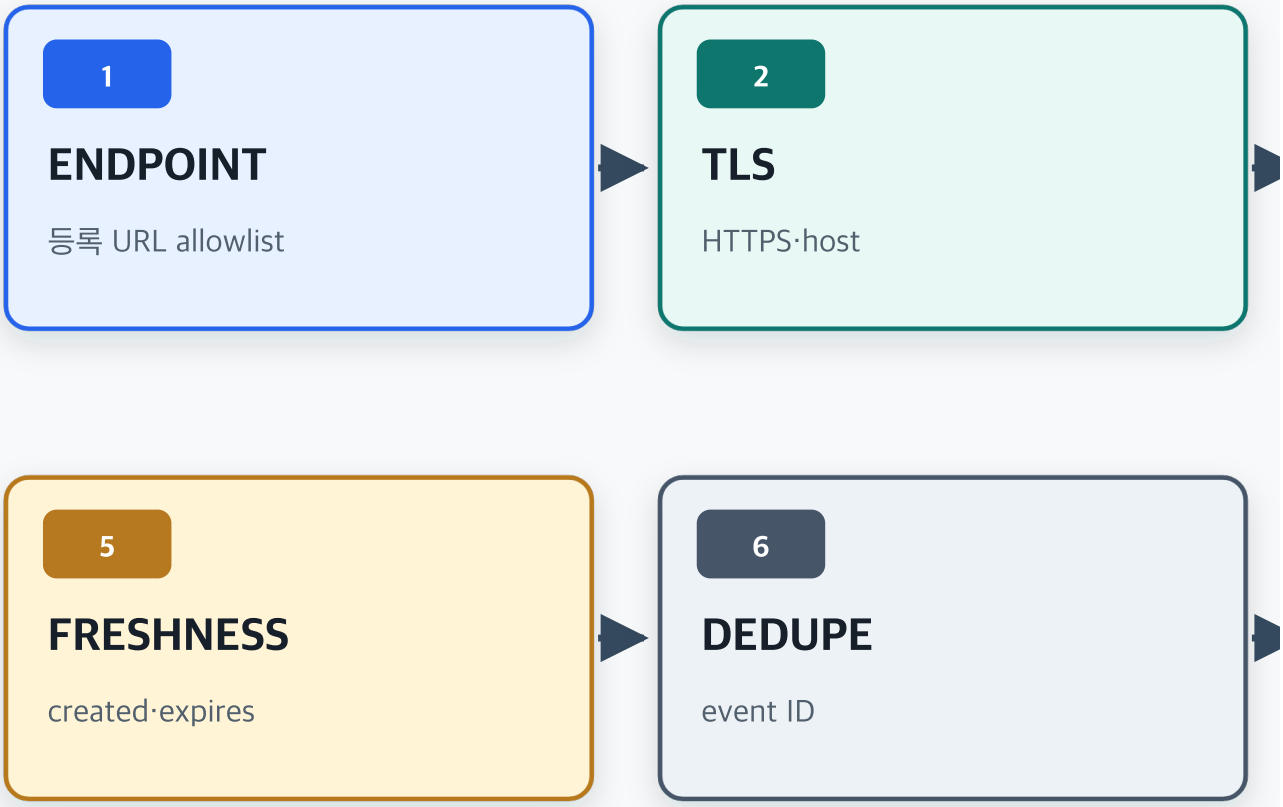
11.2. webhook만 믿지 않습니다

webhook이 유실되거나 receiver가 오래 중단될 수 있습니다. 최종 상태를 조회할 polling API나 reconciliation batch를 함께 둡니다.

30초 확인 5
사내 방화벽 안의 client가 inbound callback을 받을 수 없다면 어떤 채널이 가장 단순합니까?

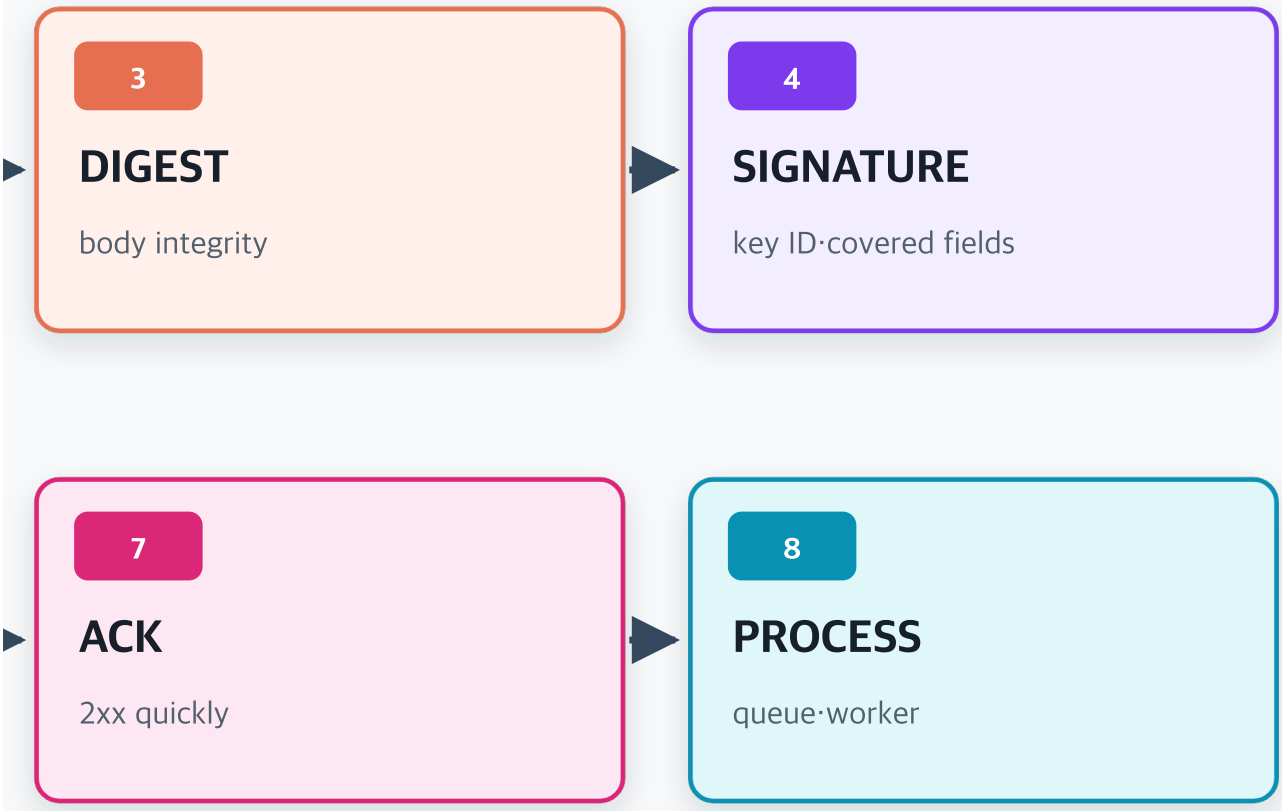
12. webhook은 보안 검증 뒤 빠르게 접수합니다

그림 11. webhook body를 바로 업무 처리하지 않습니다. 진위와 재전송을 검증하고 빠르게 응답한 뒤 내부 queue로 넘깁니다.



서명만으로 충분하지 않다 · body digest·서명 대상·time

사용자 제공 callback URL은 SSRF 경계다 · sche



estamp window·key rotation·replay store를 계약한다

eme·DNS·IP·redirect·private range를 검증한다

12.1. 서명 검증 계약

RFC 9421은 HTTP message component에 digital signature 또는 MAC을 적용하는 표준 메커니즘을 정의합니다. RFC 9530의 `Content-Digest`는 content digest를 전달합니다. digest만으로는 공격자가 body와 digest를 함께 바꾸는 일을 막지 못하므로 TLS·signature와 결합합니다.

공급자가 자체 HMAC header를 쓴다면 다음을 정확히 계약합니다.

항목	질문
key ID	어떤 secret·public key를 선택합니까
covered fields	method·path·content digest·timestamp 중 무엇을 묶습니까
canonicalization	서명 base string을 어떻게 만듭니까
freshness	허용 시간 차와 clock skew는 얼마입니까
rotation	새·이전 key가 함께 유효한 기간은 얼마입니까
replay	event ID·nonce를 얼마 동안 저장합니까

12.2. 응답과 업무 처리를 분리합니다

```
1. HTTPS endpoint 수신
2. body size·content type 제한
3. digest·signature·timestamp 검증
4. event ID dedupe record 생성
5. queue에 안전하게 접수
6. 2xx 빠른 응답
7. worker가 업무 side effect 실행
8. 최종 상태 reconcile
```

12.3. callback URL은 SSRF 경계입니다

사용자가 webhook URL을 입력할 수 있으면 server-side request forgery(SSRF) 위험이 생깁니다. HTTPS scheme, 허용 domain, DNS 재해석, private·loopback·link-local IP, redirect, port, credential 포함 URL을 검토합니다. 단순 문자열 prefix 검사로 끝내지 않습니다.

12.4. webhook schema 예

OpenAPI 3.2는 root `webhooks`로 API provider가 독립적으로 보낼 수 있는 incoming request를 기술하고, `callbacks`는 parent API operation 때문에 발생하는 out-of-band request를 기술합니다.

```
webhooks:
  exportCompleted:
    post:
      parameters:
        - in: header
          name: Event-Id
          required: true
          schema: { type: string }
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ExportCompleted'
      responses:
        '204': { description: Event accepted }
        '401': { description: Signature invalid }
        '409': { description: Event already processed }
```

13. queue는 중복 전달과 poison message를 전제로 합니다

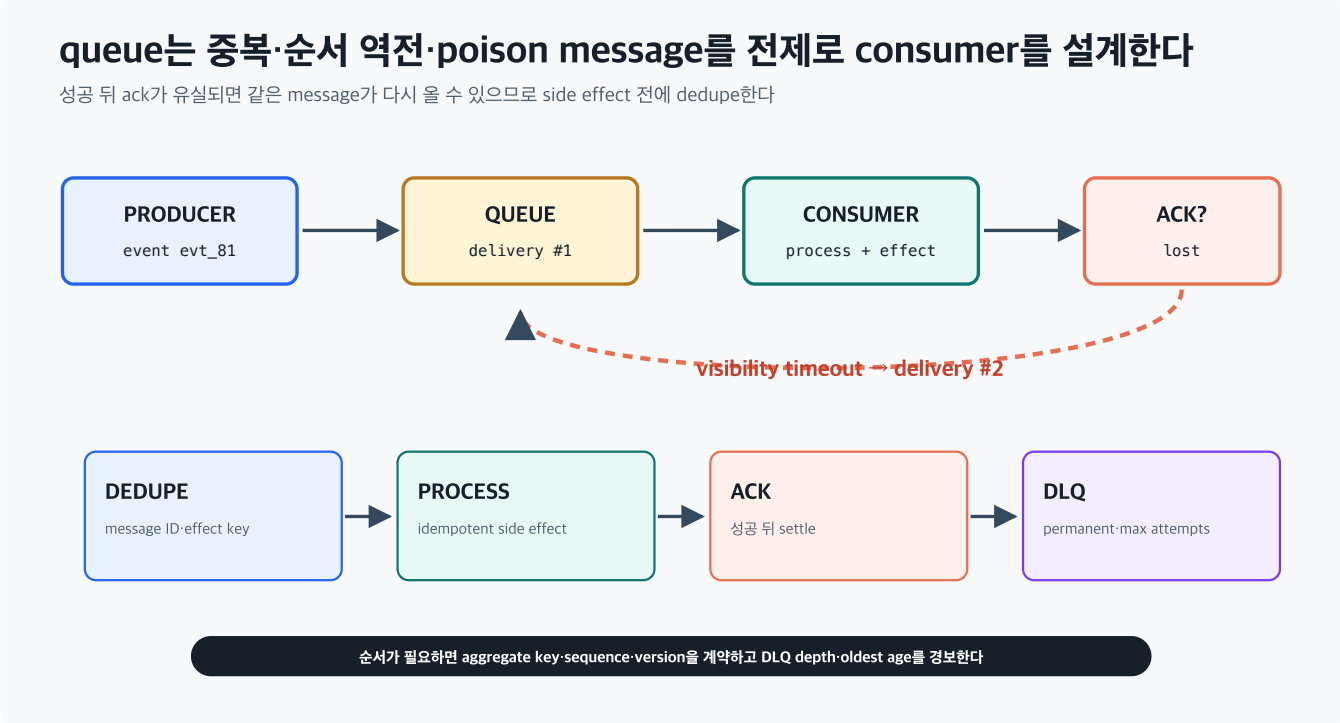
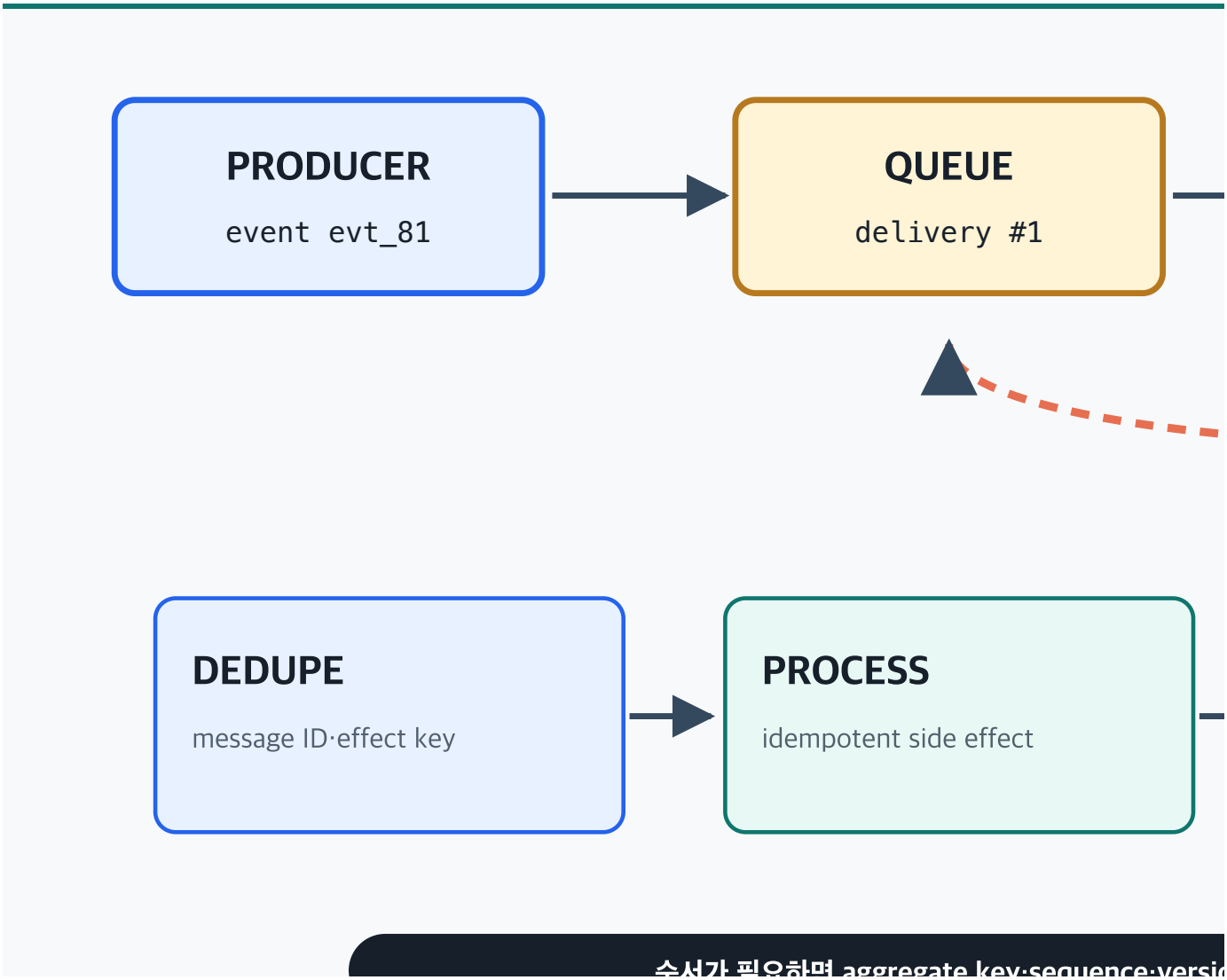
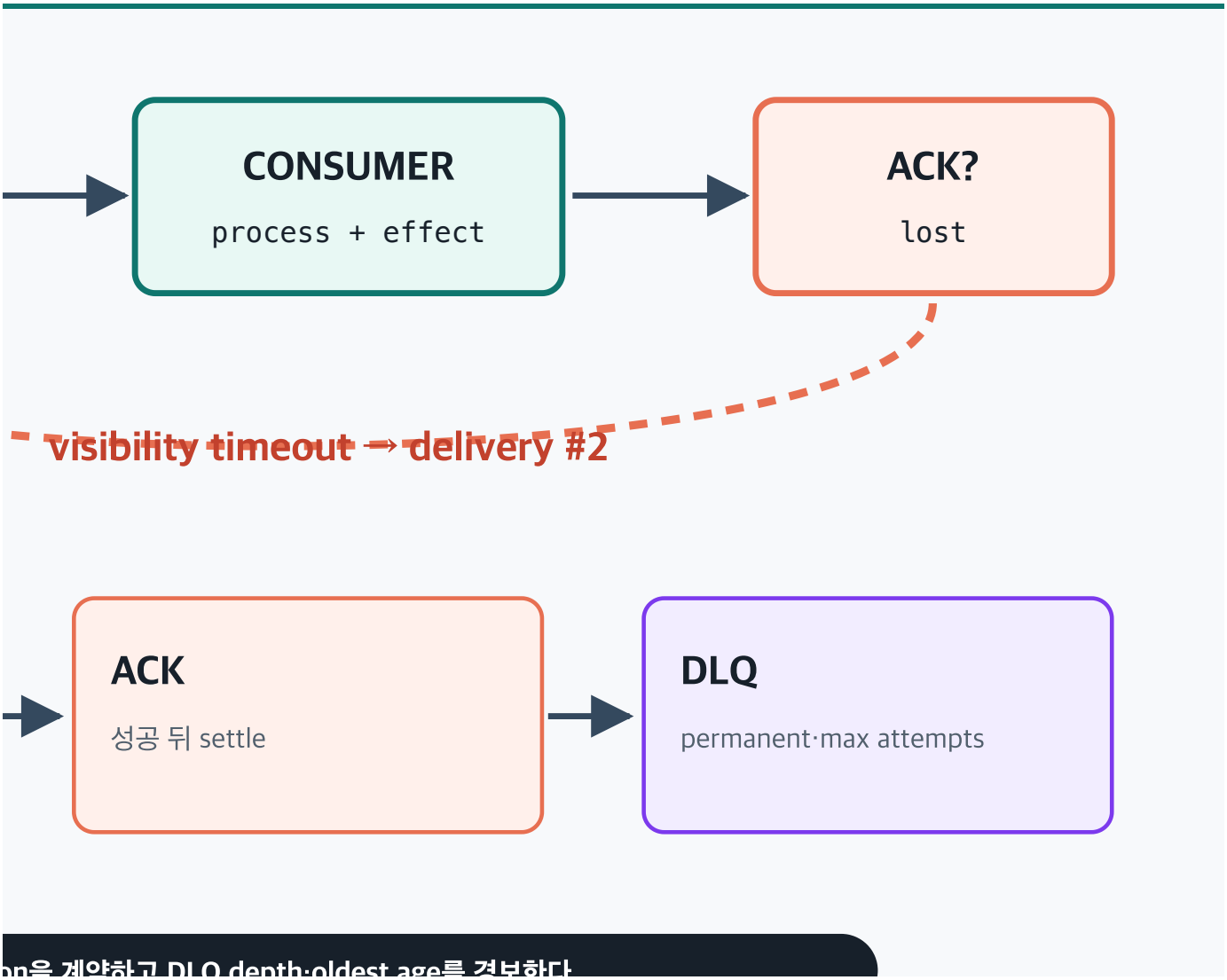


그림 12. worker가 side effect를 끝낸 뒤 ack를 잃으면 같은 message가 다시 전달될 수 있습니다.


PRODUCER
event evt_81
QUEUE
delivery #1
DEDUPE
message ID·effect key
PROCESS
idempotent side effect


CONSUMER
process + effect
ACK?
lost
visibility timeout → delivery #2
ACK
성공 뒤 settle
DLQ
permanent·max attempts

13.1. 전송 제품의 보장을 확인합니다

많은 managed queue는 at-least-once 전달을 사용해 같은 message가 다시 올 수 있습니다. 모든 broker가 같은 보장·순서·visibility·settlement를 제공하는 것은 아니므로 선택한 제품의 공식 문서를 확인합니다.

```
receive → dedupe check → business effect → effect record → ACK
                                  ↓ failure
                            visibility 만료
                                  ↓
                              redelivery
```

13.2. consumer는 idempotent해야 합니다

operation	중복 위험	방지 증거
`set status=paid`	낮지만 version 충돌 가능	aggregate version
`balance += 10000`	두 번 더할 수 있음	ledger transaction ID
email 발송	같은 메일 반복	campaign+recipient+event ID
AI 모델 호출	비용·결과 중복	job ID·provider request ID
파일 생성	같은 파일 여러 개	deterministic object key

13.3. poison message는 계속 retry하지 않습니다

schema가 깨졌거나 참조 데이터가 영구 삭제된 message는 같은 입력으로 회복되지 않습니다. permanent failure와 최대 delivery count를 판단해 dead-letter queue(DLQ)로 격리합니다.

DLQ 운영 항목	기준
max delivery	업무·비용·지연에 맞는 횟수
alert	depth > 0 또는 oldest age 임계값
payload 보호	개인정보·secret 최소화·암호화
redrive	원인 수정·승인·새 attempt ID
폐기	보존 기간·감사·업무 owner 승인

13.4. 순서는 별도 계약입니다

`created → paid → canceled`가 `canceled → paid`로 도착하면 최종 상태가 뒤집힐 수 있습니다. aggregate key, sequence, event version, expected current version을 사용합니다.

```
{
  "eventId": "evt_83",
  "aggregateId": "order_81",
  "aggregateVersion": 7,
  "type": "order.canceled"
}
```

30초 확인 6
worker가 외부 메일을 보낸 뒤 ack 전에 죽었습니다. 다시 전달된 message에서 무엇을 먼저 확인해야 합니까?

14. event envelope과 전달 보장은 다른 문제입니다

CloudEvents 1.0.2는 event를 일관된 형식으로 기술하기 위한 specification입니다. `id`, `source`, `specversion`, `type` 같은 공통 context를 제공하지만, broker의 retry·ordering·exactly-once business effect까지 자동으로 보장하지 않습니다.

```
{
  "specversion": "1.0",
  "id": "evt_81",
  "source": "https://orders.example.com",
  "type": "com.example.order.paid.v1",
  "subject": "orders/order_81",
  "time": "2026-07-15T20:15:00+09:00",
  "datacontenttype": "application/json",
  "data": {"orderId":"order_81","amount":42000}
}
```

14.1. command와 event

종류	시제	목적	이름 예
command	해 달라는 요청	특정 행동 수행	`GenerateInvoice`
event	이미 일어난 사실	구독자에게 알림	`InvoiceGenerated`

14.2. schema evolution

필드를 갑자기 삭제·의미 변경하지 않습니다. producer와 consumer 배포 시차를 고려해 additive change, version, compatibility test, deprecation window를 계약합니다. AsyncAPI 3.0은 channel·operation·message를 문서화하는 표준 형식을 제공합니다.

15. DB 저장과 event 발행의 틈을 다룹니다

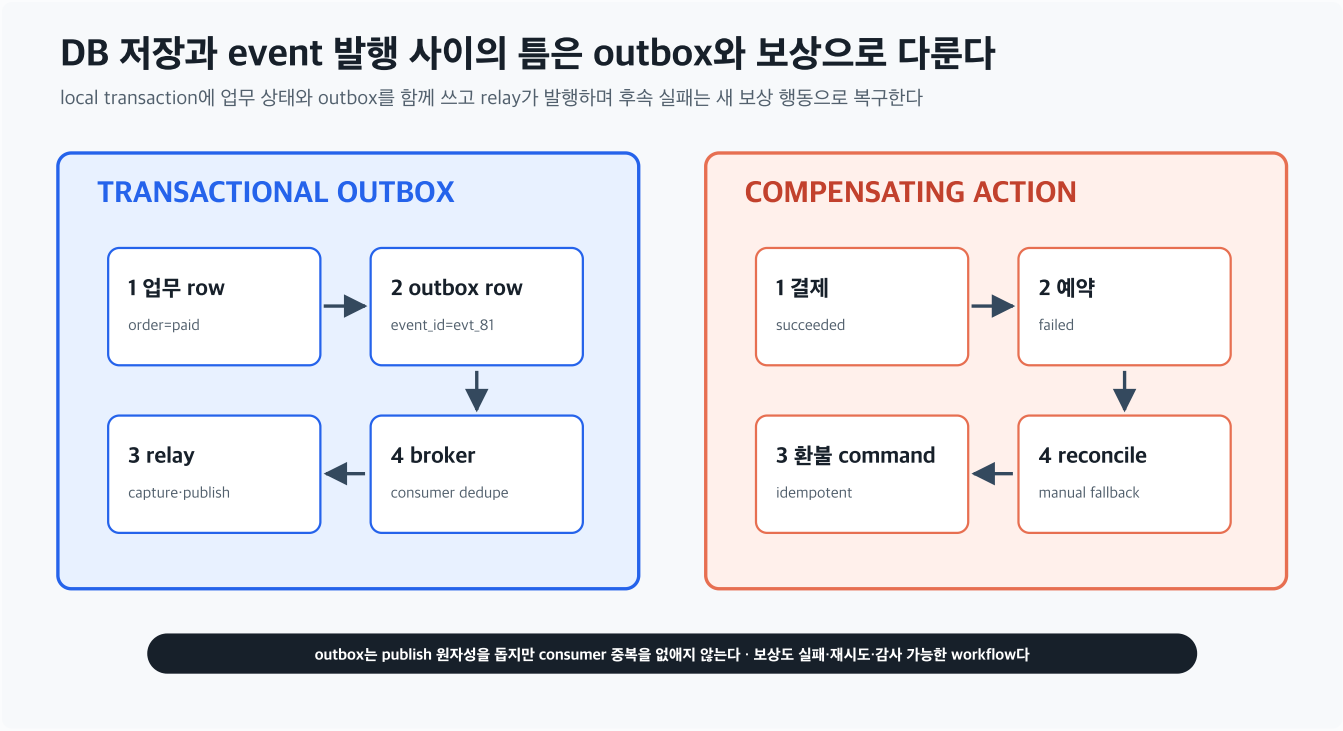
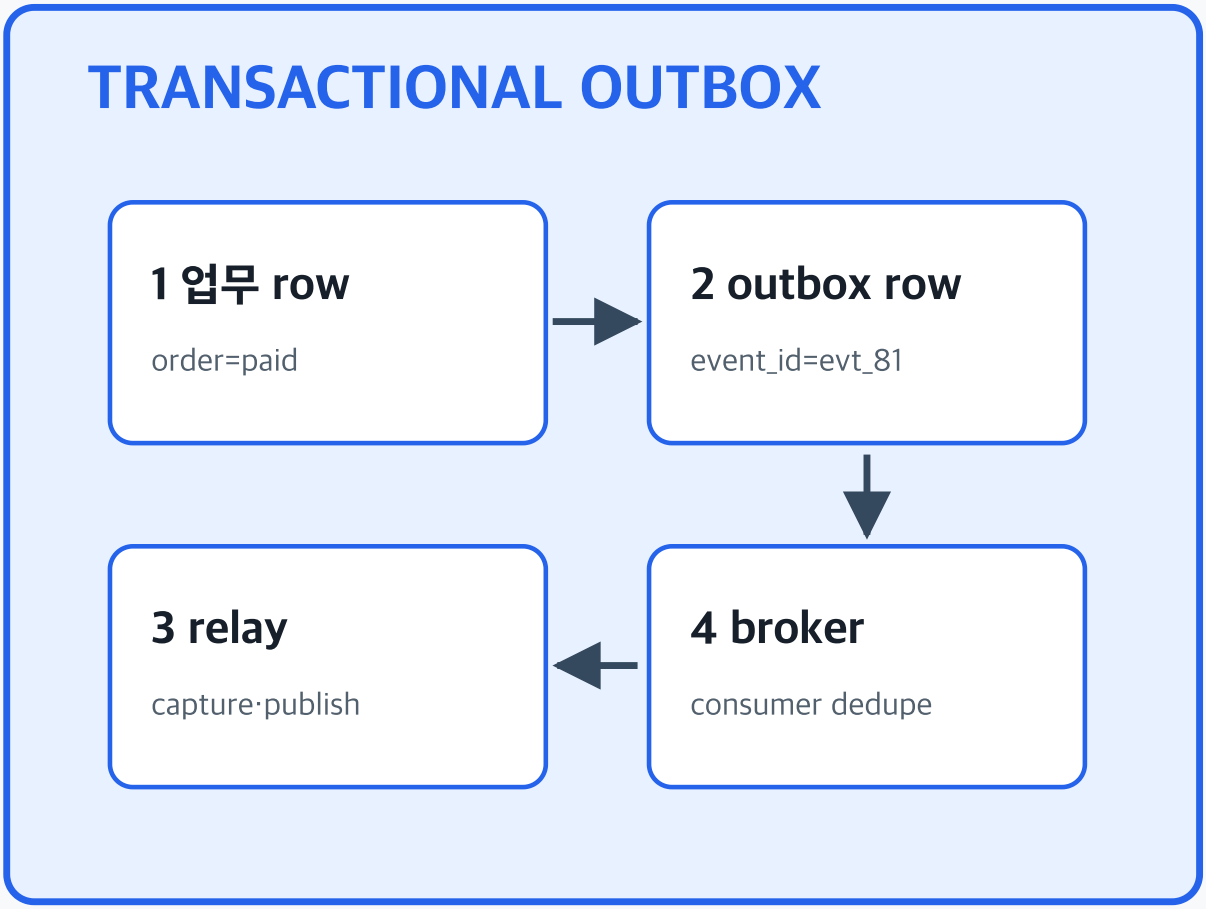
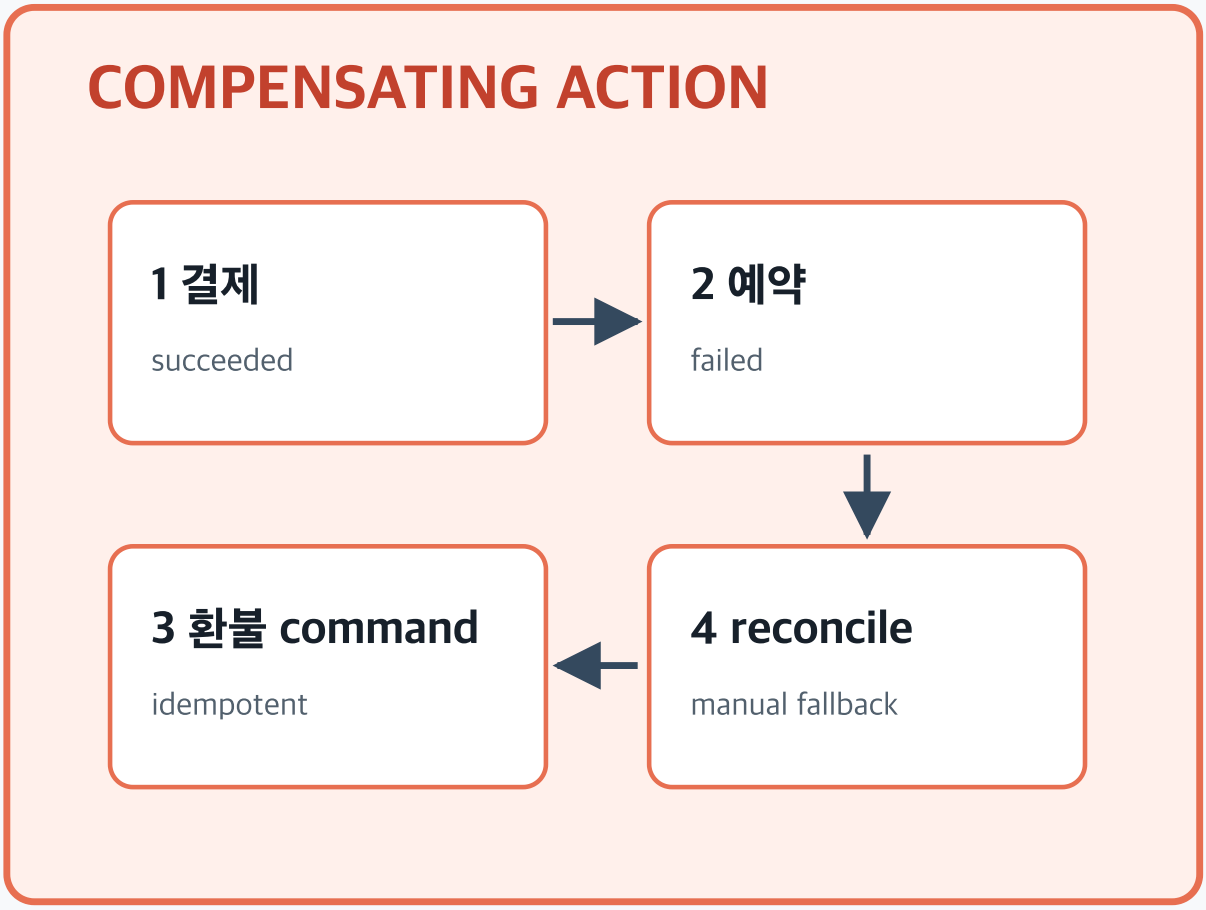


그림 13. 왼쪽은 DB·event 발행 틈을 줄이는 outbox, 오른쪽은 이미 끝난 side effect를 새 행동으로 맞추는 보상입니다.



outbox는 publish 원자성을 돕지만 consumer 중복을 없(



애지 않는다 · 보상도 실패·재시도·감사 가능한 workflow다

15.1. dual write 위험

```
나쁜 흐름 A  DB commit 성공 → publish 실패 → 업무는 바뀌었지만 event 없음
나쁜 흐름 B  publish 성공 → DB rollback → 존재하지 않는 업무 event가 전달됨
```

15.2. transactional outbox

업무 row와 outbox row를 같은 local transaction에 기록합니다. relay 또는 change data capture가 outbox를 broker로 전달합니다.

```
BEGIN
  UPDATE orders SET status='paid' WHERE id='order_81';
  INSERT outbox(event_id, aggregate_id, type, payload)
  VALUES ('evt_81','order_81','order.paid.v1','{...}');
COMMIT
```

outbox는 DB와 event 기록의 원자성을 돕습니다. relay가 같은 event를 다시 publish하거나 consumer가 다시 받을 가능성까지 없애지는 않으므로 event ID dedupe가 여전히 필요합니다.

15.3. outbox 운영 증거

지표	무엇을 찾는가
unpublished count	발행되지 않은 outbox 누적
oldest unpublished age	relay가 멈췄는지
publish attempts	broker 장애·poison payload
duplicate publish	consumer dedupe 작동
cleanup lag	outbox 보존·삭제 지연

16. 부분 실패는 보상 가능한 workflow로 설계합니다

16.1. compensation은 시간 되돌리기가 아닙니다

항공권 결제 후 좌석 예약이 실패했다고 과거 결제 기록을 지우지 않습니다. `refund requested → refund succeeded`라는 새 업무 행동을 실행하고 감사 이력을 남깁니다.

원 단계	완료 증거	뒤 단계 실패	보상 행동	보상 실패
결제 승인	payment ID	예약 실패	환불 command	운영자 reconcile
쿠폰 차감	ledger ID	주문 생성 실패	쿠폰 복원 entry	고객 지원
파일 공개	object version	DB 저장 실패	공개 취소·삭제	격리·alert
AI 작업 과금	provider usage ID	결과 저장 실패	결과 재조회	비용 조정

16.2. 보상도 idempotent해야 합니다

refund command가 timeout돼 다시 실행될 수 있습니다. 원 거래 ID와 compensation ID를 연결하고 같은 보상 의도가 두 번 실행되지 않도록 합니다.

```
{
  "workflowId": "trip_81",
  "originalEffectId": "pay_ext_81",
  "compensationId": "refund_trip_81",
  "state": "compensating",
  "attempt": 2,
  "nextActionAt": "2026-07-15T20:20:00+09:00"
}
```

30초 확인 7
보상 transaction이 실패하면 원 업무를 단순히 failed로 끝내도 됩니까?

17. 끝까지 같은 ID로 관측합니다

외부 연계는 여러 process와 시간이 끊어져 실행됩니다. “요청 로그는 성공인데 결과가 없다”를 피하려면 시작·접수·전달·실행·완료를 같은 상관 식별자로 연결합니다.

ID	범위	질문
request ID	HTTP 한 요청	이 교환에서 무슨 일이 있었나
idempotency key	같은 업무 의도	새 실행인가 기존 의도인가
job ID	비동기 작업 수명	현재 상태와 결과는 무엇인가
message ID	한 전달 단위	중복·ack·DLQ는 무엇인가
event ID	한 업무 사실	이미 처리한 event인가
correlation ID	여러 단계 업무	한 workflow의 모든 단계는 무엇인가
trace ID	분산 실행 경로	어느 service·span에서 늦었나
external resource ID	공급자 결과	외부 실제 상태는 무엇인가

W3C Trace Context는 `traceparent`와 `tracestate`로 분산 trace context를 전달하는 표준입니다. 외부 경계에서 받은 값은 신뢰 경계 정책에 따라 검증·재생성하고, secret·개인정보를 tracing metadata에 넣지 않습니다.

17.1. 최소 event log

```
{
  "event": "integration_attempt_finished",
  "requestId": "req_81",
  "correlationId": "corr_order_81",
  "traceId": "4bf92f3577b34da6a3ce929d0e0e4736",
  "idempotencyKeyHash": "sha256:...",
  "jobId": "job_81",
  "externalProvider": "payment-demo",
  "externalResourceId": "pay_ext_81",
  "attempt": 2,
  "elapsedMs": 1820,
  "outcome": "unknown",
  "nextAction": "reconcile"
}
```

token, API key, webhook secret, 원문 개인정보, 전체 결제정보는 로그에 남기지 않습니다.

17.2. 시작보다 완료를 관측합니다

지표	질문
enqueue-to-completion latency	queue 대기까지 포함한 업무 지연은 얼마인가
queue depth·oldest age	worker가 따라잡지 못하는가
job terminal ratio	접수 후 끝나지 않은 job이 있는가
retry attempts·exhausted	일시 장애가 확대되는가
dedupe hit·conflict	중복 요청과 key 오용이 있는가
webhook verification failure	공격·key rotation 문제가 있는가
DLQ depth·age	실패 작업이 방치되는가
compensation pending	부분 완료가 고객에게 남아 있는가

18. 실습 · 연계 계약 8-gate를 판정합니다

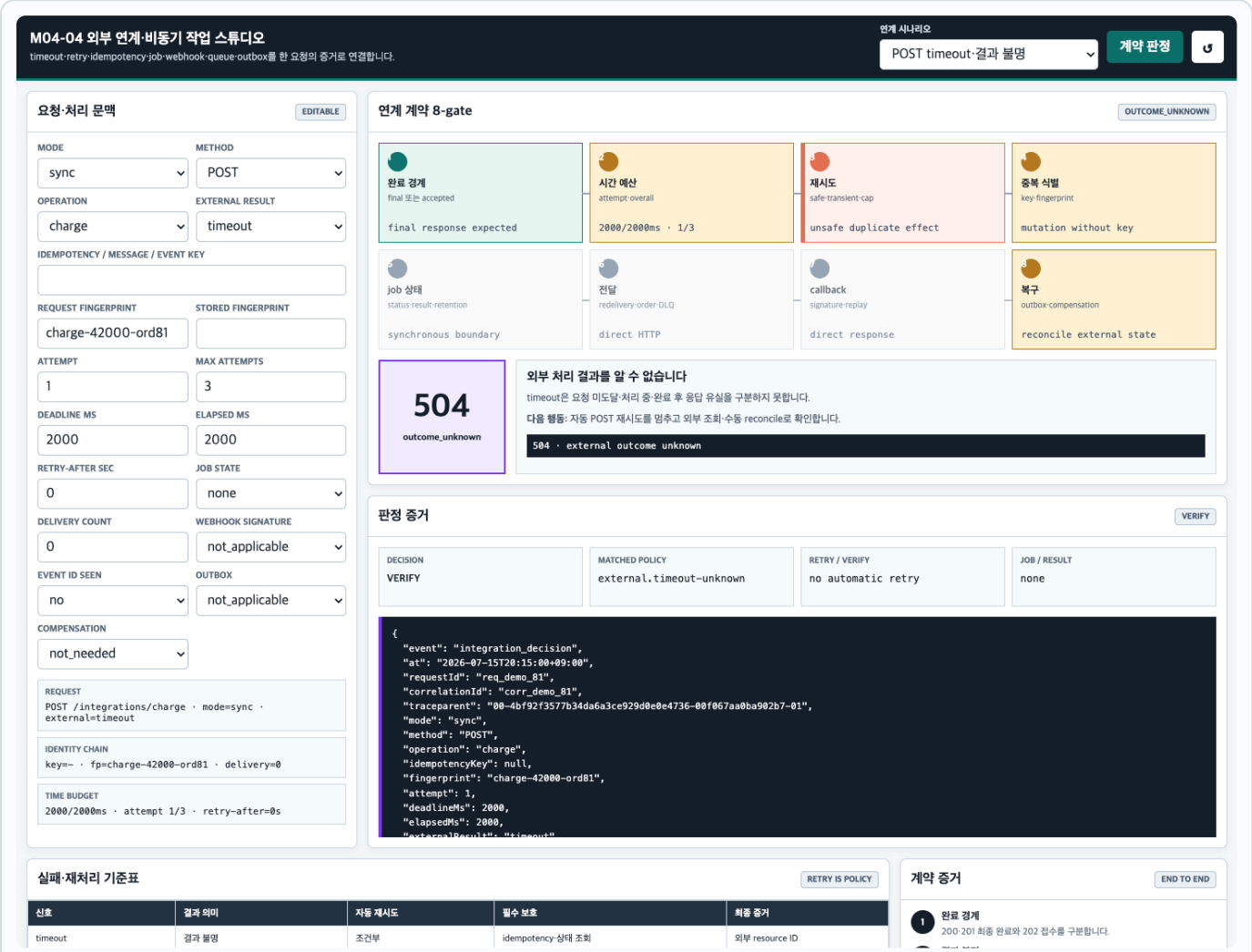


그림 14. 같은 입력에서 key·attempt·job·delivery·signature 한 값만 바꾸며 판정 경계가 이동하는지 확인합니다.

18.1. 실습 목표

1. POST timeout이 `failed`가 아니라 `outcome_unknown`이 되는 이유를 설명합니다.
2. 429·503가 있어도 중복 안전성과 budget이 없으면 retry하지 않습니다.
3. 같은 key·같은 fingerprint와 같은 key·다른 fingerprint를 구분합니다.
4. 202·queue redelivery·webhook replay·outbox pending·compensation을 판정합니다.

18.2. 실행 순서

1. 단계별 실습 안내를 엽니다.
2. `POST timeout·결과 불명`을 실행합니다.

3. idempotency key를 `pay_81`로 바꾸고 다시 판정합니다.
4. `같은 key·다른 요청`과 `queue 중복 전달`을 비교합니다.
5. `정상 서명 webhook`과 `재전송 webhook`을 비교합니다.
6. `DB 완료·outbox 발행 대기`와 `부분 완료·보상 필요`를 실행합니다.

18.3. 실습 기록

scenario	status·outcome	failed·warn gate	자동 실행 여부	다음 행동	최종 증거
POST timeout					
429					
duplicate conflict					
202 accepted					
queue redelivery					
webhook replay					
outbox pending					
compensation					

19. 기발자의 연계 의사결정표

19.1. 회의에서 반드시 묻는 질문

영역	질문	나쁜 답	완료 증거
완료	202 뒤 무엇이 완료입니까	비동기라 알아서 됩니다	terminal state·result
시간	attempt·overall timeout은 얼마입니까	30초 하나입니다	단계별 budget
retry	누가 어떤 오류만 재시도합니까	실패하면 세 번입니다	retry matrix
중복	같은 업무 의도를 어떻게 찾습니까	UUID를 씁니다	scope·key·fingerprint·TTL
job	상태·취소·만료는 무엇입니까	pending·done입니다	transition table

영역	질문	나쁜 답	완료 증거
webhook	진위·재전송·SSRF는 어떻게 막습니까	HTTPS입니다	signature·dedupe·URL policy
queue	중복·순서·poison을 어떻게 다룹니까	queue가 보장합니다	consumer·DLQ contract
저장·발행	DB commit과 event publish 틈은요	거의 동시에 합니다	outbox evidence
복구	외부 side effect 뒤 실패하면요	rollback합니다	compensation workflow
관측	접수 뒤 완료되지 않은 일을 어떻게 찾습니까	로그를 봅니다	SLI·alert·correlation

19.2. AI 생성 연계 설계 검수

AI에게 “retry를 넣어 줘”라고만 요청하지 않습니다.

```
아래 외부 연계의 완료 경계·timeout budget·retryable failure·idempotency scope·job 상태·
webhook verification·queue redelivery·DLQ·outbox·compensation·관측 ID를 표로 설계하라.
HTTP 표준과 provider 자체 계약을 구분하고, timeout을 실패로 단정하지 말라.
각 자동 재시도에는 중복 안전성·최대 attempt·overall deadline·terminal 행동을 적어라.
```

AI 결과에서 다음 문장을 찾으면 멈춰 검증합니다.

위험 문장	확인할 사실
exactly once를 보장합니다	어느 범위의 전달·처리·side effect인가
timeout이면 retry합니다	결과 불명·중복 안전·budget은 있는가
202면 성공입니다	접수 성공인가 업무 완료인가
webhook secret으로 암호화합니다	MAC·signature·encryption을 혼동하지 않는가
queue 순서가 보장됩니다	partition·key·consumer concurrency 조건은 무엇인가
outbox로 중복이 없어집니다	relay·consumer dedupe는 남는가

20. 한 장 요약

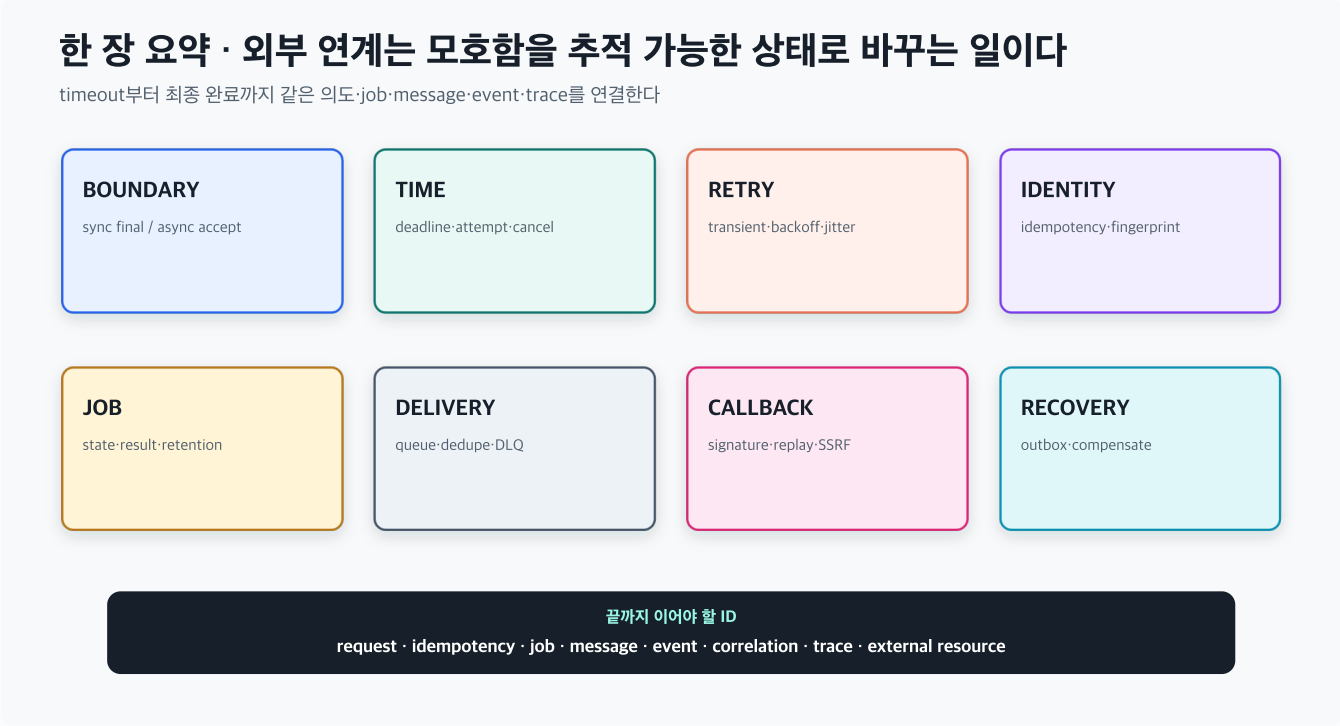


그림 15. timeout부터 최종 완료까지 같은 의도와 실행 증거를 이어 붙이면 외부 연계의 모호함을 관리할 수 있습니다.

BOUNDARY

sync final / async accept

TIME

deadline·attempt·cancel

JOB

state·result·retention

DELIVERY

queue·dedupe·DLQ

끝까지 이

request · idempotency · job · message · ev

RETRY

transient·backoff·jitter

IDENTITY

idempotency·fingerprint

CALLBACK

signature·replay·SSRF

RECOVERY

outbox·compensate

어야 할 ID

ent · correlation · trace · external resource

기억할 여덟 문장

1. 동기·비동기는 **업무 완료 경계**로 구분합니다.
2. timeout은 결과 불명일 수 있으므로 실패로 단정하지 않습니다.
3. retry는 안전성·일시성·budget을 만족할 때 한 계층에서 제한합니다.
4. idempotency는 key·fingerprint·state·result·retention의 계약입니다.
5. 202는 접수이며 job resource가 최종 상태를 보관합니다.
6. webhook과 queue의 재전송을 전제로 consumer side effect를 중복 방지합니다.
7. outbox는 DB·event 틈을 줄이고 compensation은 부분 완료를 새 행동으로 맞춥니다.
8. request·job·message·event·trace·external ID로 시작부터 terminal까지 관측합니다.

최종 산출물

산출물	파일·위치	완료 조건
외부 연계 계약	T04-04	actor·auth·schema·timeout·retry·완료 경계
job 상태도	T04-04	전이·terminal·취소·만료·result
retry·중복 방지표	T04-04	오류·budget·key·fingerprint·TTL
webhook·queue 계약	T04-04	signature·dedupe·ack·DLQ·ordering
장애·보상 기록	실습 기록	unknown·partial failure·reconcile evidence

21. 셀프 테스트

문제 1

동기·비동기를 코드 문법이 아니라 업무 계약으로 구분하는 질문을 씁니다.

문제 2

외부 결제 POST가 timeout됐을 때 가능한 네 상태와 첫 행동을 씁니다.

문제 3

attempt timeout과 overall deadline의 차이를 설명합니다.

문제 4

POST를 application 수준에서 retry-safe하게 만드는 idempotency record 필드 다섯 가지를 씁니다.

문제 5

같은 idempotency key에 다른 fingerprint가 들어왔을 때 응답과 행동을 씁니다.

문제 6

202 응답과 job resource에 각각 어떤 정보를 넣어야 합니까?

문제 7

429·503를 자동 retry하기 전 확인할 세 조건을 씁니다.

문제 8

webhook에서 signature가 유효해도 event ID를 확인해야 하는 이유를 씁니다.

문제 9

queue consumer가 side effect 뒤 ack 전에 죽었을 때 생기는 일과 방지 방법을 씁니다.

문제 10

transactional outbox가 해결하는 문제와 해결하지 않는 문제를 하나씩 씁니다.

문제 11

compensation이 database rollback과 다른 이유를 씁니다.

문제 12

하나의 비동기 workflow를 끝까지 추적할 ID 여섯 가지를 씁니다.

답안 메모

구분	내 답의 핵심어
완료 경계	
결과 불명	

구분	내 답의 핵심어
timeout·retry budget	
idempotency·dedupe	
job·webhook·queue	
outbox·보상·관측	

22. 셀프 테스트 정답과 해설

정답 1

“같은 request-response 교환에서 최종 업무 결과를 받는가, 접수 ID 뒤 별도 채널에서 최종 상태를 확인하는가”로 구분합니다. `async/await` 사용 여부만으로 판단하지 않습니다.

정답 2

요청 미도달, 외부 거부, 처리 중, 처리 완료 후 응답 유실이 가능합니다. 자동 새 POST 전에 같은 key·외부 조회 ID·job 상태로 기존 결과를 확인합니다.

정답 3

attempt timeout은 외부 호출 한 번을 기다리는 한도입니다. overall deadline은 retry 대기와 여러 attempt, 결과 저장을 포함한 전체 업무 한도입니다.

정답 4

key, scope, request fingerprint, processing state, result·resource reference, retention·expiresAt 중 다섯 가지 이상입니다. key 문자열만 저장해서는 다른 요청 충돌을 찾기 어렵습니다.

정답 5

기존 결과를 반환하지 않고 `409` 같은 conflict로 거부합니다. 새 의도라면 새 key를 쓰고, 같은 의도라면 원 요청을 복원합니다.

정답 6

202에는 접수 결과, job ID, status URL, 현재 상태, 다음 poll 힌트를 둡니다. job resource에는 상태·시간·attempt·진행·result·structured error·취소 link·만료를 둡니다.

정답 7

같은 효과를 여러 번 요청해도 안전한지, 오류가 일시적인지, attempt·overall deadline·quota가 남았는지 확인합니다. `Retry-After`와 backoff·jitter도 적용합니다.

정답 8

유효하게 서명된 같은 event가 공급자의 retry나 공격자의 replay로 다시 올 수 있습니다. event ID·timestamp·nonce를 보존해 업무 side effect를 한 번만 실행합니다.

정답 9

visibility timeout 뒤 같은 message가 다시 전달될 수 있습니다. consumer가 message·effect ID로 dedupe하고 side effect를 idempotent하게 만든 뒤 성공 증거가 저장된 후 ack합니다.

정답 10

outbox는 업무 DB 변경과 발행할 event 기록 사이의 dual write 틈을 줄입니다. relay 중복 publish나 consumer 중복 전달·side effect까지 자동으로 없애지는 않습니다.

정답 11

외부 side effect는 같은 local transaction으로 과거로 되돌릴 수 없습니다. compensation은 환불·복원처럼 새 업무 행동을 실행하고 그 자체의 retry·idempotency·실패·감사를 관리합니다.

정답 12

request ID, idempotency key, job ID, message ID, event ID, correlation ID, trace ID, external resource ID 중 여섯 가지 이상입니다. 각 ID의 scope가 서로 다름을 함께 설명해야 합니다.

23. 최종 제출 체크리스트와 공식 근거

- [ ] 동기 최종 완료와 비동기 접수를 분리했습니다.
- [ ] timeout을 failed·unknown·pending과 구분했습니다.
- [ ] connect·attempt·overall·cancel budget을 정의했습니다.
- [ ] retryable·non-retryable 오류와 retry 주체를 정했습니다.
- [ ] backoff·jitter·attempt cap·deadline이 있습니다.
- [ ] mutating operation의 idempotency scope·key·fingerprint·TTL이 있습니다.
- [ ] 같은 key·같은 요청과 같은 key·다른 요청의 행동이 다릅니다.
- [ ] 202 response에 job ID·status URL·현재 상태가 있습니다.
- [ ] job 상태·허용 전이·terminal·취소·만료를 정의했습니다.
- [ ] polling·webhook·event·stream 선택과 fallback이 있습니다.
- [ ] webhook signature·digest·freshness·replay·key rotation을 검수합니다.

- [ ] 사용자 callback URL의 SSRF 방어를 검수합니다.
- [ ] queue 중복·순서 역전·poison message를 테스트합니다.
- [ ] DLQ depth·oldest age·redrive에 owner와 alert가 있습니다.
- [ ] event schema version과 compatibility 정책이 있습니다.
- [ ] DB·event dual write에 outbox 또는 동등한 복구 전략이 있습니다.
- [ ] 부분 완료의 idempotent compensation과 manual reconcile이 있습니다.
- [ ] 시작·완료·실패를 correlation·trace·external ID로 연결합니다.
- [ ] secret·token·개인정보 원문을 message·log에 넣지 않습니다.

이 교재는 다음 공식·권위 자료를 2026-07-15에 확인해 학습자용으로 재구성했습니다.

- RFC 9110 · HTTP Semantics: safe·idempotent method, 202 Accepted, Location, Retry-After, HTTP status 의미
- RFC 6585 · Additional HTTP Status Codes: 429 Too Many Requests와 Retry-After
- RFC 9457 · Problem Details for HTTP APIs: machine-readable terminal error
- OpenAPI Specification 3.2.0: callbacks·webhooks·response 계약
- RFC 9421 · HTTP Message Signatures: HTTP component signature·MAC
- RFC 9530 · Digest Fields: Content-Digest·Repr-Digest와 한계
- IETF Idempotency-Key draft status: 만료된 Internet-Draft 상태 확인
- CloudEvents Specification 1.0.2: 공통 event context와 protocol binding
- AsyncAPI Specification 3.0.0: channel·operation·message-driven API 기술
- W3C Trace Context: traceparent·tracestate 전파
- AWS Builders' Library · Timeouts, retries and backoff with jitter: retry amplification·backoff·jitter 운영 원리
- AWS Builders' Library · Making retries safe with idempotent APIs: client intent·idempotent API 설계
- Azure Architecture Center · Asynchronous Request-Reply: 202·status endpoint·polling·cancel pattern
- Azure Architecture Center · Background jobs: redelivery·ordering·poison message·DLQ·관측
- Debezium · Outbox Event Router: transactional outbox 구현 사례
- OWASP · SSRF Prevention Cheat Sheet: webhook·callback URL 검증
- Azure Architecture Center · Compensating Transaction: 부분 완료의 보상·재개·수동 복구

M04-04 완료
이제 "API를 호출한다"나 "비동기로 처리한다"를 한 줄 요구로 남기지 않습니다. timeout 뒤 결과 불명, 제한된 retry, idempotency, 202 job, webhook·queue 재전송, outbox, DLQ, compensation, end-to-end evidence를 하나의 연계 명세로 설명하고 검수할 수 있습니다.