생성형 AI 기반 RAG 개발자 과정

Text Splitter 청크 최적화

Session 12 / 33 — Day 2

검색 정확도의 숨은 70% — 청크 분할 전략

박수현 · (사)한국정보공학기술사회 · 2026

LLM 이론 · LangChain · ChromaDB CRUD · Agent — 4일 핸즈온

이번 시간을 마치면

✂️ 분할 방식

- **Character** Splitter
- **Recursive** Splitter (권장)
- Recursive 가 자연 경계를 따르는 원리

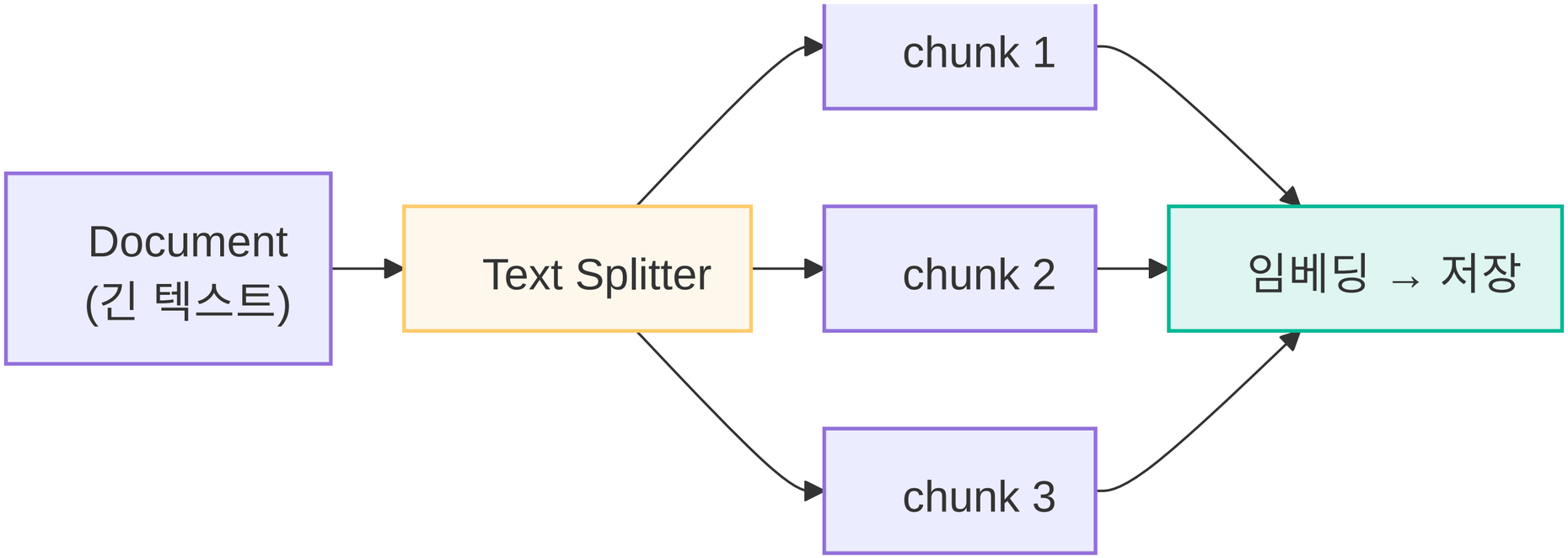
🎛️ 두 제어 변수

- **chunk size** — 조각 크기
- **overlap** — 경계 겹침
- chunk **메타데이터**(chunk_id)

> 🎯 목표: "작게=정밀 / 크게=문맥" 트레이드오프를 코드로 제어한다.

출처: Day 2 Text Splitter

분할 단계 — Loader 직후, 임베딩 직전



💡 검색은 **chunk 단위**로 수행된다 → chunk 가 곧 검색의 최소 단위.

과대도 과소도 부적합한 chunk 크기

❌ 과대할 때

- 한 chunk 에 여러 주제 → **잡음** 혼입
- 검색 정확도 저하, context 낭비

❌ 과소할 때

- 문장이 끊겨 **문맥 손상**
- 지시어("이것/그것")의 참조 대상 소실

✅ 적정할 때

- 질문 하나에 답할 **최소 충분량**
- 정확·저비용·고속

문서	권장 chunk
규정·매뉴얼	작게 (400~600)
서술·논문	크게 (800~1200)

> 🔑 Day1 S6 의 "질문 하나에 답할 만큼" 기준을 구체적인 수치로 확정하는 단계.

청크 경계 결정 기준

- **Character**: 단일 구분자(`\n\n` 등)로만 분할 — 단순하지만 크기 불균일·문장 절단 발생
- **Recursive**: `\n\n`(문단) → `\n`(줄) → ` `(단어) 순으로 **자연 경계**를 우선 적용 → 실무 기본은 **RecursiveCharacterTextSplitter**

권장 분할기 — Recursive (스니펫)

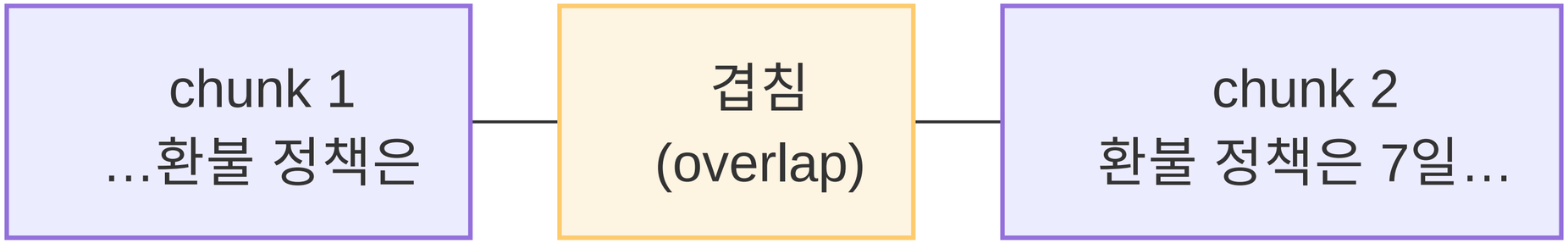
```
from langchain_text_splitters import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(
    chunk_size=500,        # 조각당 최대 글자
    chunk_overlap=80,      # 인접 조각 겹침 (경계 보존)
)
chunks = splitter.split_documents(docs)
print(len(chunks), chunks[0].metadata)
```

- `split_documents` → 메타데이터(source·page) **자동 승계**
- `chunk_size` / `chunk_overlap` 만 조정해 튜닝
- 토큰 과금(OpenAI) 정밀 관리 → `.from_tiktoken_encoder(...)` 로 **토큰 단위** 분할

> 📁 비교 예제: `2.langchain/7.RAG/2.loaders/2.3_chunking.py` (Character · Recursive · tiktoken 3종 비교)

출처: Day 2 Text Splitter

overlap 으로 경계 문장 보존



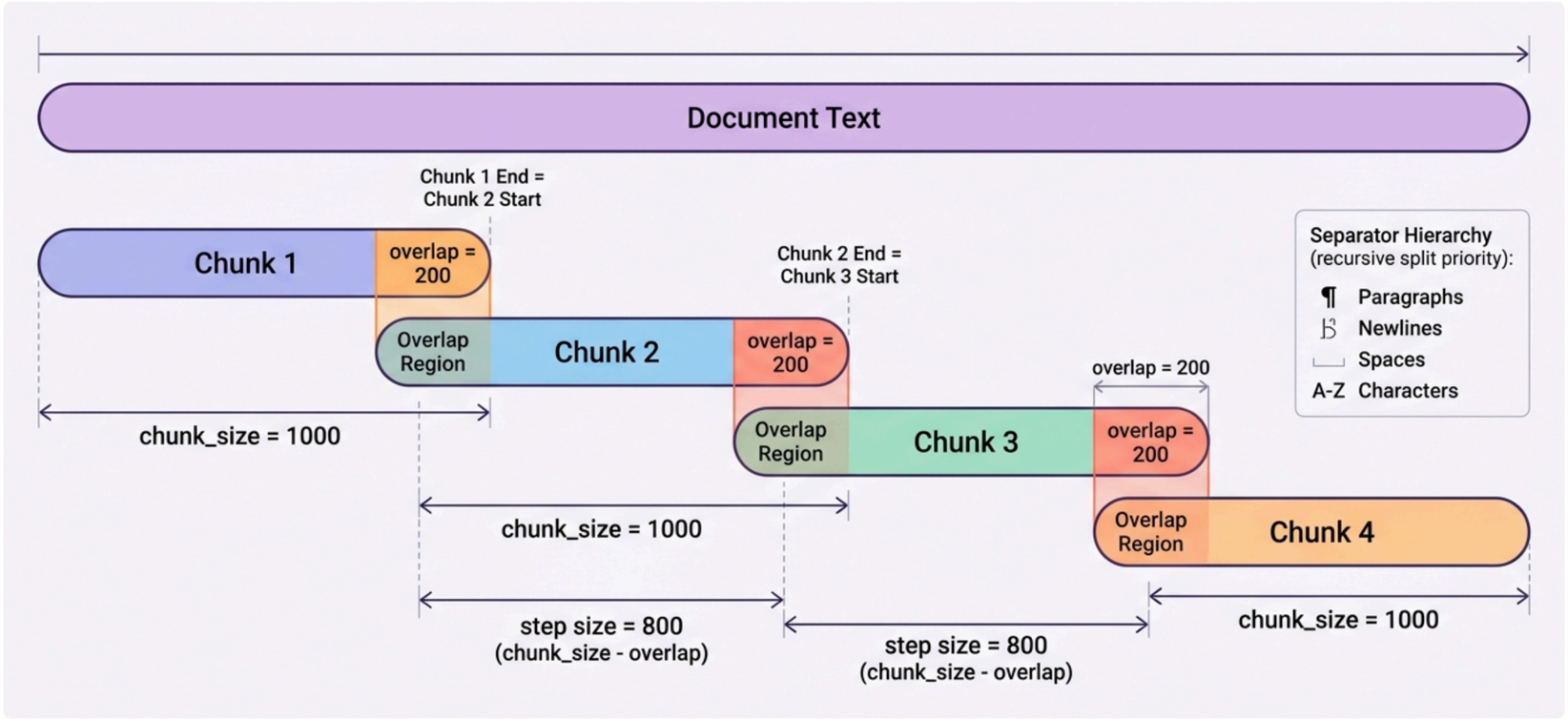
overlap 이 없으면

- chunk 경계에서 문장이 **분절**
- 핵심 정보가 두 chunk 로 분산되어 검색 실패

overlap 이 있으면

- 한 chunk 의 **끝 N글자**가 다음 chunk 앞에 재수록
- 경계 문장을 **양쪽 chunk 에 모두** 포함
- 통상 chunk size 의 **10~20%**

chunk_size / overlap 슬라이딩 윈도우



RecursiveCharacterTextSplitter · chunk_size/overlap 슬라이딩 윈도우, 경계에서 겹쳐 문맥 손실 방지

조각별 출처 정보 부여 (스니펫)

```
for i, c in enumerate(chunks):
    c.metadata["chunk_id"] = i          # 몇 번째 조각인지
    # source·page 는 split_documents 가 이미 승계

print(chunks[5].metadata)
# {'source': 'faq.pdf', 'page': 2, 'chunk_id': 5}
```

- `source` + `page` + `chunk_id` → **정확한 출처 인용**의 기반
- Day2 RAG Chain(S16)·Day3 출처 표시(S22)에서 그대로 활용

> 📁 참고: `2.langchain/7.RAG/2.loaders/2.4_metadata.py`

실전 튜닝 — 증상별 조정 기준

증상	조정
답이 단편적·문맥 부족	chunk size ↑, overlap ↑
검색에 엉뚱한 내용 섞임	chunk size ↓ (더 정밀하게)
경계에서 정보 누락	**overlap ↑** (10→20%)
표·코드가 깨짐	문서별 **분할 기준** 조정

> 💡 고정된 정답은 없다 → **문서 단위 실험**으로 결정. 동일 질문 세트로 답 품질을 비교하며 size·overlap 을 조정한다.

핵심 5가지

1. ✅ 검색은 **chunk 단위** → 분할이 RAG 품질의 주요 변수
2. ✅ **Recursive** = 문단→문장→단어 자연 경계 (실무 기본)
3. ✅ **chunk size**: 작게=정밀 / 크게=문맥
4. ✅ **overlap**(10~20%)으로 경계 문장 보존
5. ✅ `source·page·chunk_id` 메타데이터 = **출처 추적**의 기반