스프링 배치

배치?

일괄 처리라고도 하는 과정으로서
실시간으로 요청에 의해서 처리되는 방식이 아닌
일괄적으로 한꺼번에 대량의 프로세스를 처리하는 방식

ex/
차액정산 배치
사용자 거래금액 통계 배치
이자상환 안내 배치
이메일 대상 발송 배치
...

스프링 배치!

https://jojoldu.tistory.com/324

https://spring.io/projects/spring-batch

대용량 데이터 처리에 최적화되어 고성능을 발휘

효과적인 로깅, 통계 처리, **트랜잭션 관리 등 재사용 가능한 필수 기능**을 지원

수동으로 처리하지 않도록 **자동화**

예외 사항과 비정상 동작에 대한 방어 기능

비지니스 로직에 집중

배치 활용 예시 

요구사항

1) **30일 동안 이체 금액이 100만원 미만인 이체 횟수가 5건 미만**인 회원은 **상태를 휴면상태로 변경**해야 한다.
2) **회원의 수는 1천만 명**이다.
3) **거래 데이터는 회원 별로 생성**되어 있다.
4) **여신 서비스를 조회하여 대출이 없는 경우에만 휴면**으로 처리한다.

그냥 배치

1천만건 조회

대출 여부 조회

데이터 반환

여신 서비스

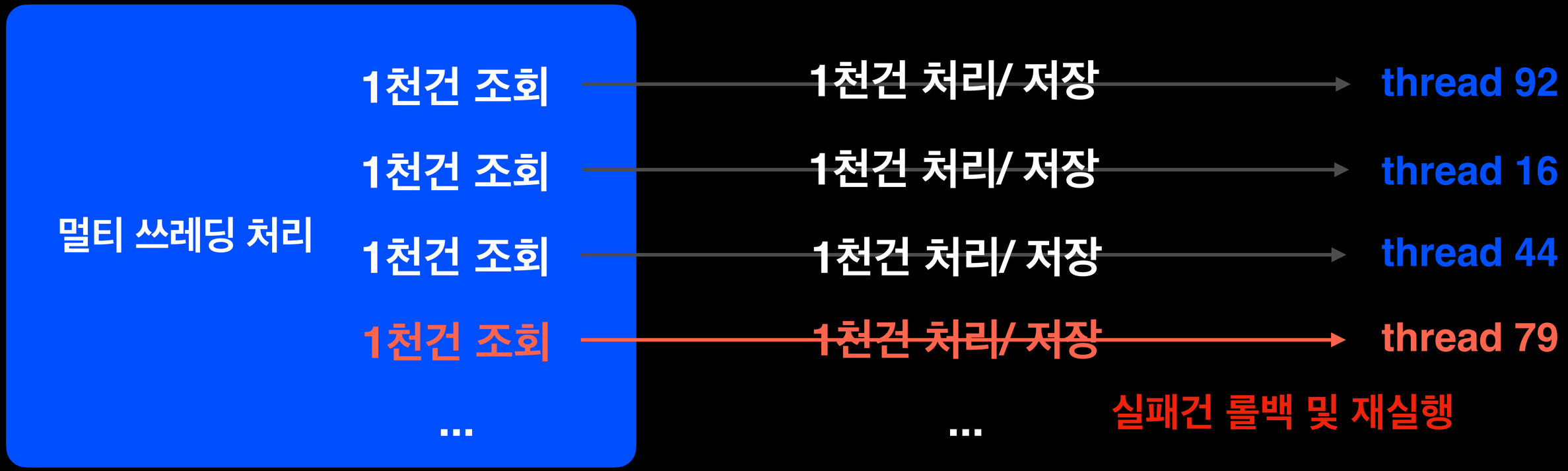
👨🏻‍✈️ 준하님
🥷🏿 슬아님
👮🏻‍♀️ 한기님
👨🏻‍🚒 소현님
👲🏽 진우님
....

스프링 배치

50 thread * 1천건 = 5만건씩 (200번) 처리

chunk = 1,000 thread = 50

스프링 배치



50 thread * 1천건 = 5만건씩 (200번) 처리

chunk = 1,000 thread = 50

10,000,000번 vs 200번

(IO 통신이 아닐 경우 매우 효과적)

실제 성능 비교

환경

h2 database
task (max/core pool size 50, throttle limit size 50, chunk 1000)

ex/ 일반 배치

```kotlin
val startTime = System.currentTimeMillis()

usersRepository
    // @description 조건에 해당하는 유저 데이터 리스트를 조회
    .getUserByLessThenTransactionCountFiveAndAmountOneHundredThousandBeforeThirtyDays()

    // @description 여신 대출 여부 조회
    .filter { !isExistsLoans(it.id!!) }

    // @description one by one, not bulk
    .map {
        it.setDormant()
        usersRepository.save(it)
    }
    .toList()

val milliseconds = System.currentTimeMillis() - startTime
val seconds = (milliseconds / 1000) % 60

println("${seconds}s${milliseconds % 1000}ms")
```

ex/ 스프링 배치

```kotlin
@Bean(최근_30일_동안_거래가_없는_회원_휴면_처리_스텝)
fun statusChangeUserWithoutTransactionThirtyDaysStep(
    @Qualifier(최근_30일_동안_거래가_없는_회원_조회_리더) reader: ItemReader<Users>,
    @Qualifier(여신_대출_존재하는_유저_휴면_처리_프로세서) processor: ItemProcessor<Users, Users>,
    @Qualifier(유저_데이터_저장_라이터) writer: ItemWriter<Users>,
    stepBuilderFactory: StepBuilderFactory,
): Step {
    return stepBuilderFactory
        .get(최근_30일_동안_거래가_없는_회원_휴면_처리_스텝)
        .chunk<Users, Users>(CHUNK_PAGE_SIZE)
        .reader(reader)
        .processor(processor)
        .writer(writer)

        // 실패건 BATCH_RETRY_COUNT 회 재시도
        // 실패한 CHUNK_PAGE_SIZE 건 롤백
        // 재시도 에러 기준 (TimeoutException, DeadlockLoserDataAccessException)

        .faultTolerant()
        .retryLimit(BATCH_RETRY_COUNT)
        .retry(TimeoutException::class.java)
        .retry(DeadlockLoserDataAccessException::class.java)
        .transactionManager(transactionManager)
        .taskExecutor(taskAsyncExecutor)

        .throttleLimit(THROTTLE_LIMIT_SIZE)
        .build()
}
```

1천명 유저 테스트

0s824ms vs 1s290ms

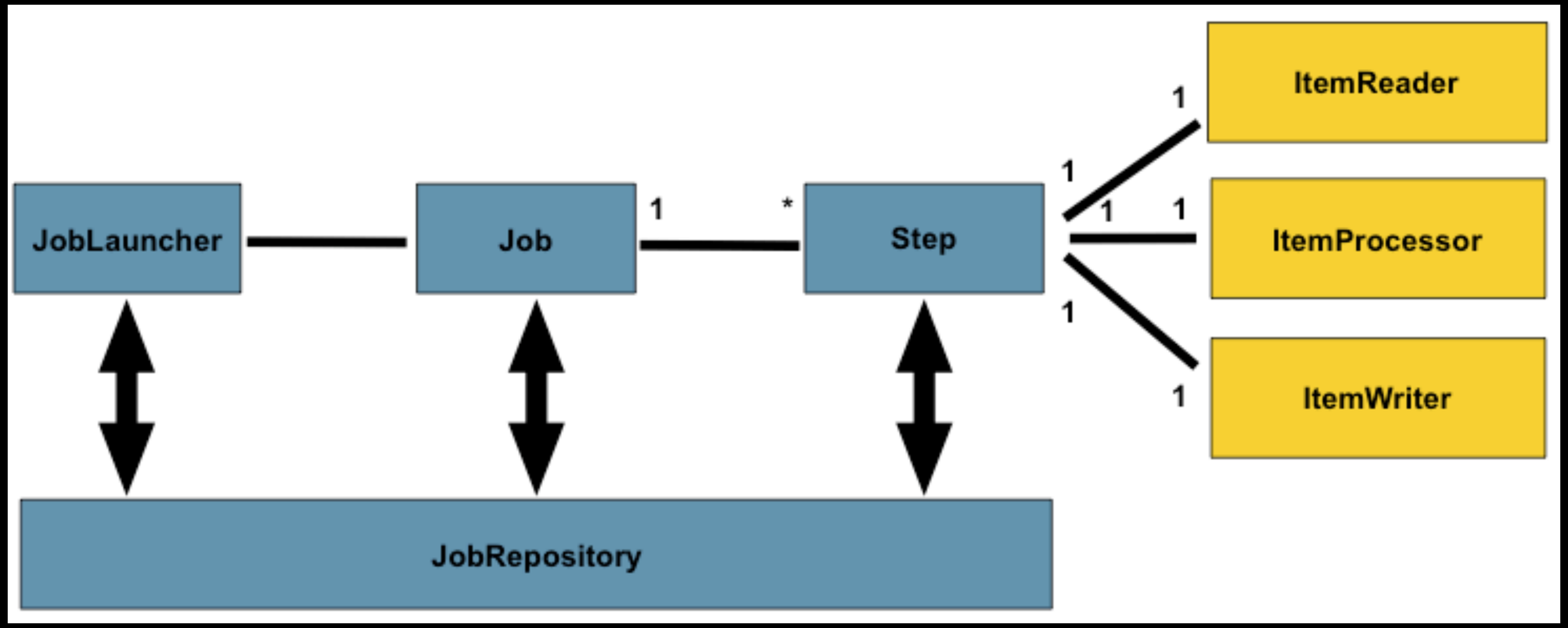
1만명 유저 테스트

1s164ms vs 7s642ms

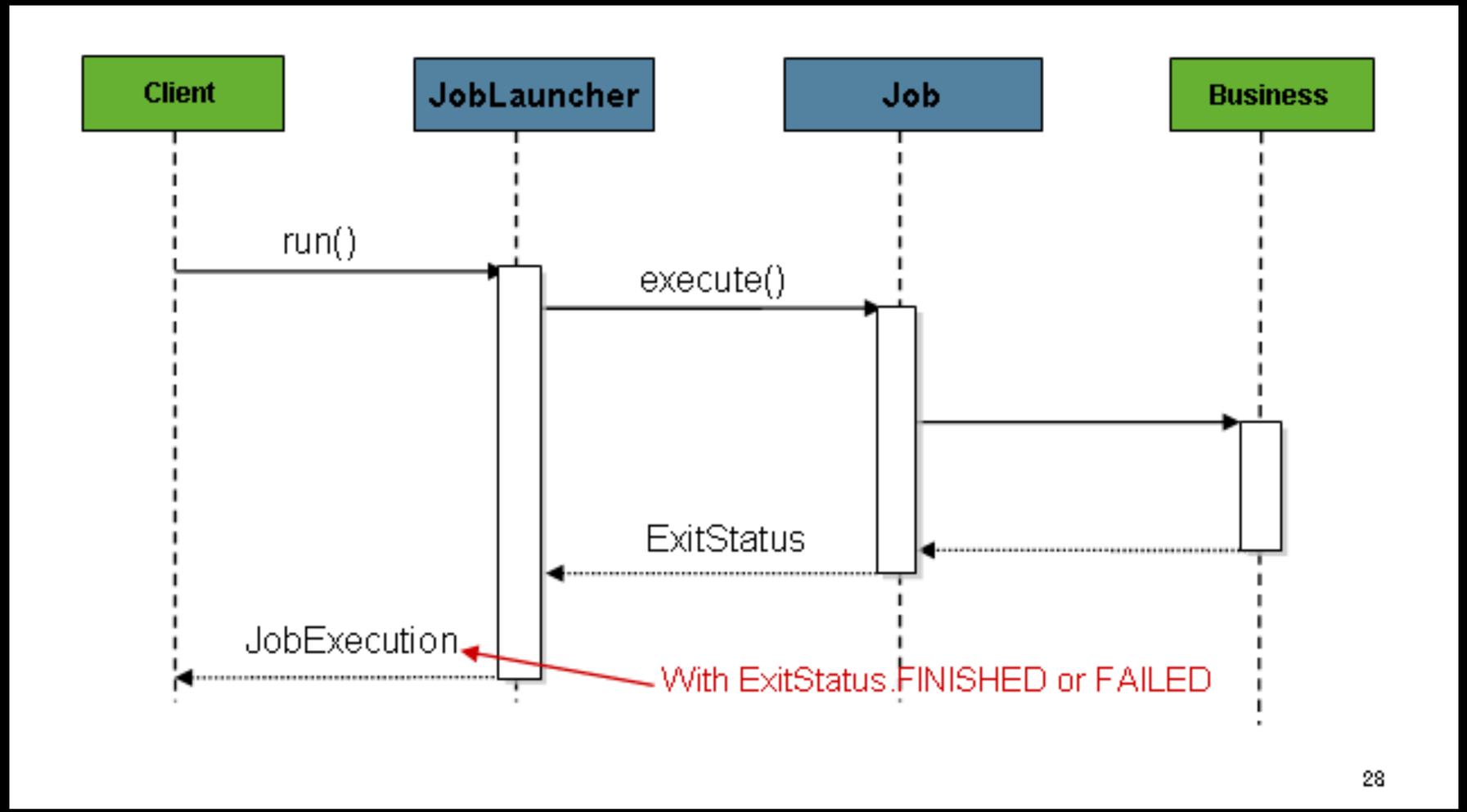
1십만명 유저 테스트

7s960ms vs 17s101ms

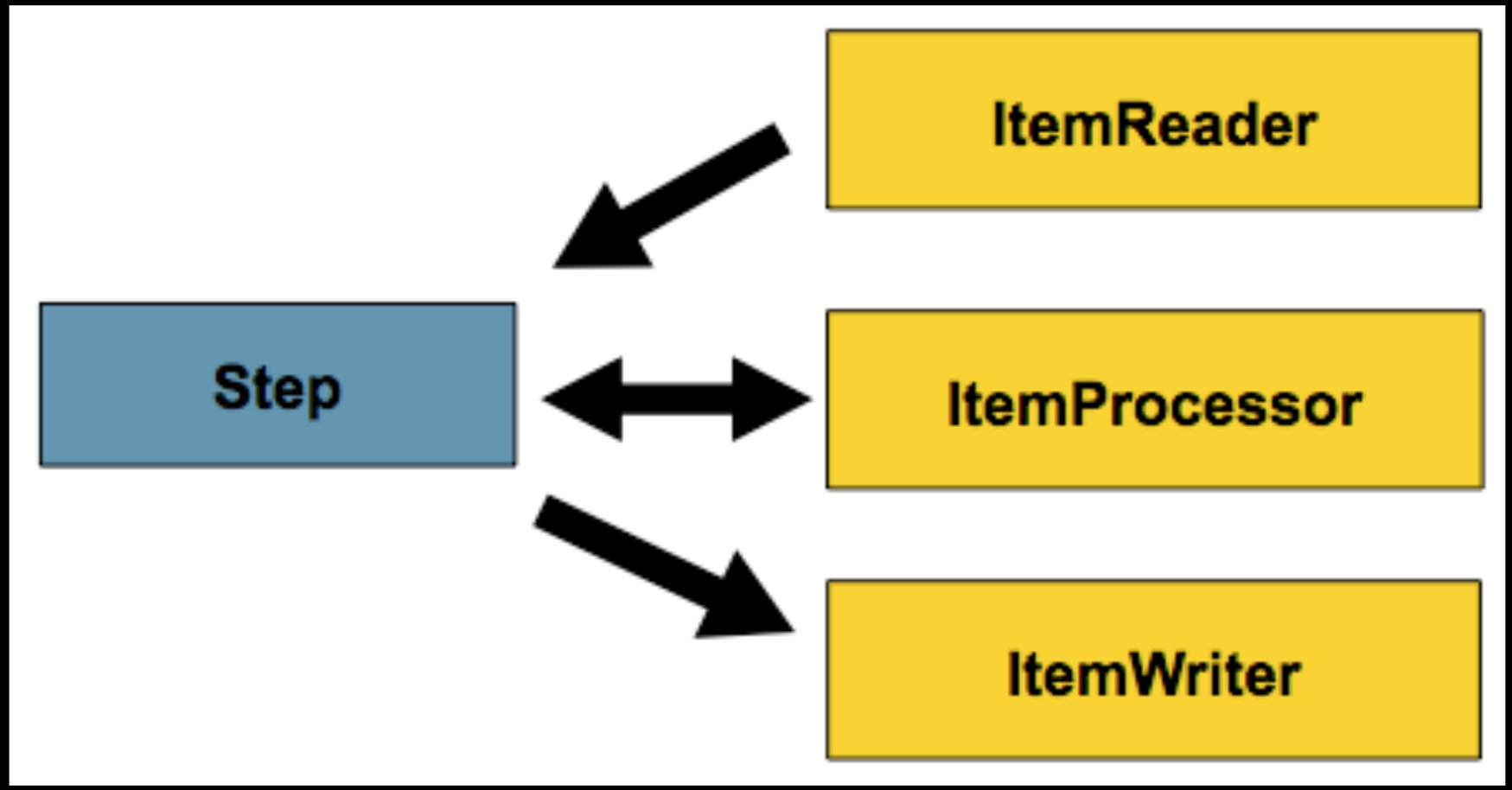
Batch Stereotypes



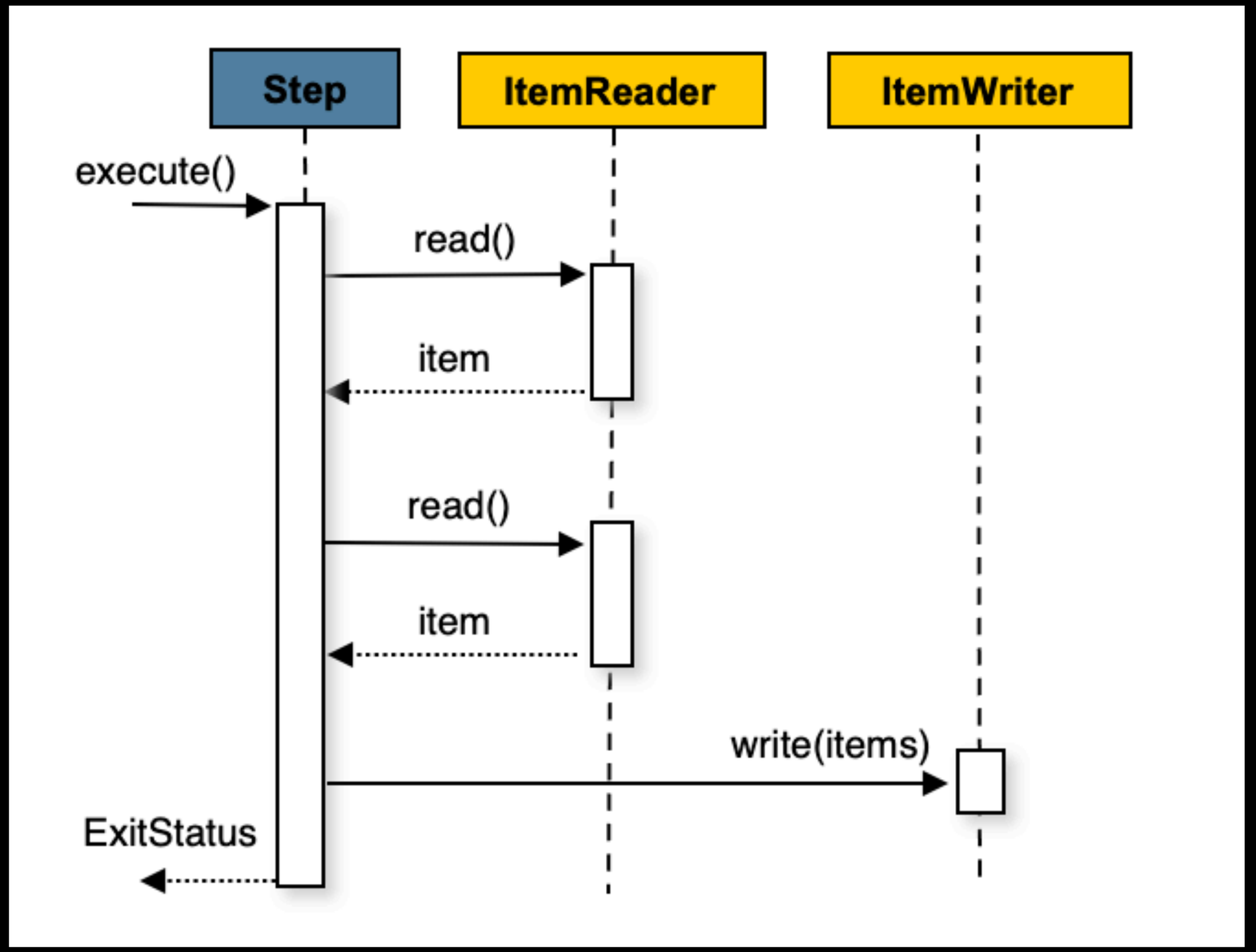
Job Launcher Sequence



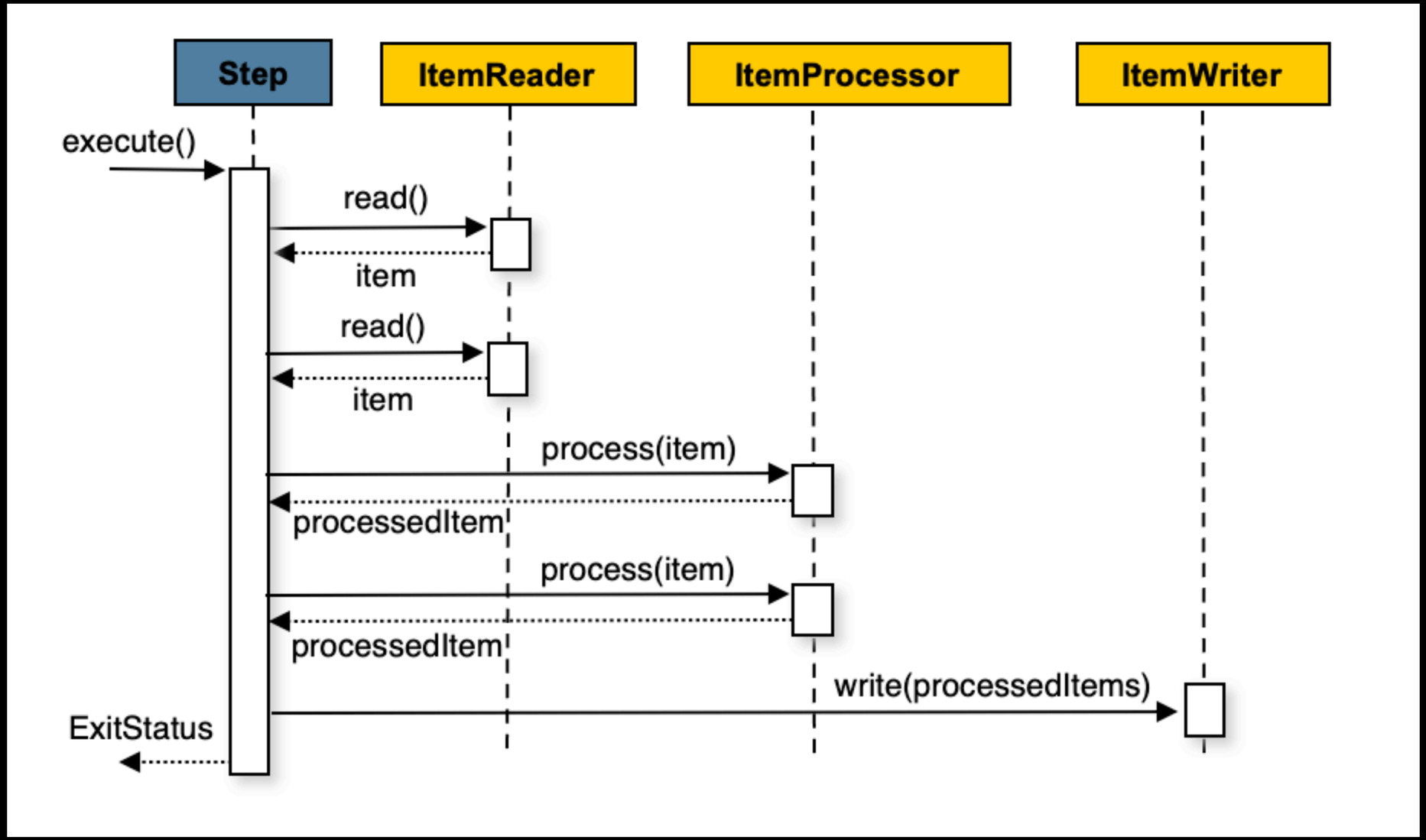
Step



Step - Chunk-oriented Processing



Step - Chunk-oriented Processing with Item Processor



심화 - ItemReader 구현 클래스

심화 - ItemWriter 구현 클래스

심화 - Composited

https://sonegy.wordpress.com/2011/12/28/springbatch에서-compositeitemwriter활용/

심화 - Chunk

심화 - Parallel

심화 - Transactional

심화 - Intercepter Listener

심화 - Annotation Listener

주의할 점 및 굴팁

멀티쓰레드로 배치를 구현하는 것이라면 **PagingItemReader 로 구현해야 Thread Safe** 해요.

대량의 데이터가 아니고 멀티쓰레드 환경이 아니라면 CursorItemReader 로 구현하는 것이 좋아요.

멀티쓰레드 환경일 때에는 반드시 saveState 값을 false 로 설정해두어 실패하면 무조건 처음부터 다시 실행될 수 있도록 해주면 좋아요.

심화 - Retry

https://cheese10yun.github.io/spring-batch-basic/

https://deeplify.dev/back-end/spring/batch-tutorial

http://wiki.gurubee.net/pages/viewpage.action?pageId=4949437

https://velog.io/@kihwanyu/Spring-Batch-정리

https://12bme.tistory.com/365