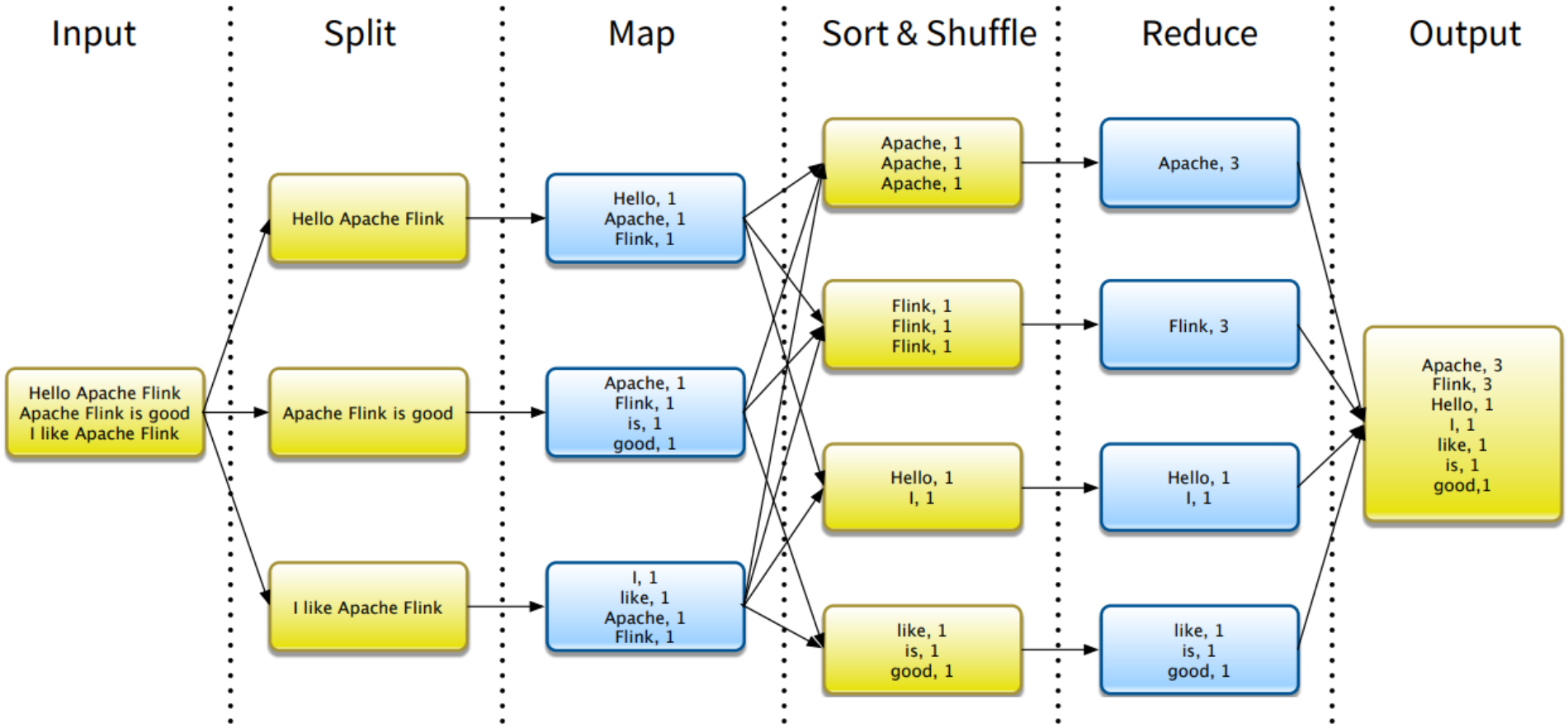
Apache Flink를 이용한 streaming data 처리

강사 : 윤성국

를 알아보기 전에 먼저 MapReduce를 대충 알아봅시다

근데 이제 하둡을 곁들인

MapReduce 예시 – word count



hadoop map reduce의 경우

- map과 reduce만 고려하면 됩니다.
- 문제는 map하고 reduce만 있습니다.
- 파이프라인 강제 : split – map – sort – shuffle –reduce
- 중간결과 디스크에 저장 : 겁나게 느림
- 스트림 데이터 그런거 없음 -> 배치데이터만 처리

hadoop의 문제를 처리하려고 나온 spark

- 디스크 대신 메모리를 적극적으로 이용해서 속도문제 개선
- 하지만 spark도 넘지 못한 streaming data의 벽
- 그래서 나온 spark streaming -> 하지만 레알 스트리밍이 아님 (작게 쪼개 놓은 batch 였을 뿐)
- 그래서 나온 apache storm

그래서 나온 apache storm

- Very low latency, true streaming, mature and high throughput
- Excellent for non-complicated streaming use cases
- no state management

하지만

- No advanced features like Event time processing, aggregation, windowing, sessions, watermarks, etc
- Atleast-once(메시지 전달 성공을 위해 여러번 시도) guarantee

그래서 나온(2) apache Flink 역사

- 베를린 대학의 차세대 빅데이터 플랫폼 구축을 목표로 개발
- 초기에는 스트라토스피어라는 이름으로 시작
- 네펠레라는 논문을 기초로 하고 있음

Nephele: Efficient Parallel Data Processing in the Cloud

Daniel Warneke
Technische Universität Berlin
Berlin, Germany
daniel.warneke@tu-berlin.de

Odej Kao
Technische Universität Berlin
Berlin, Germany
odej.kao@tu-berlin.de

ABSTRACT

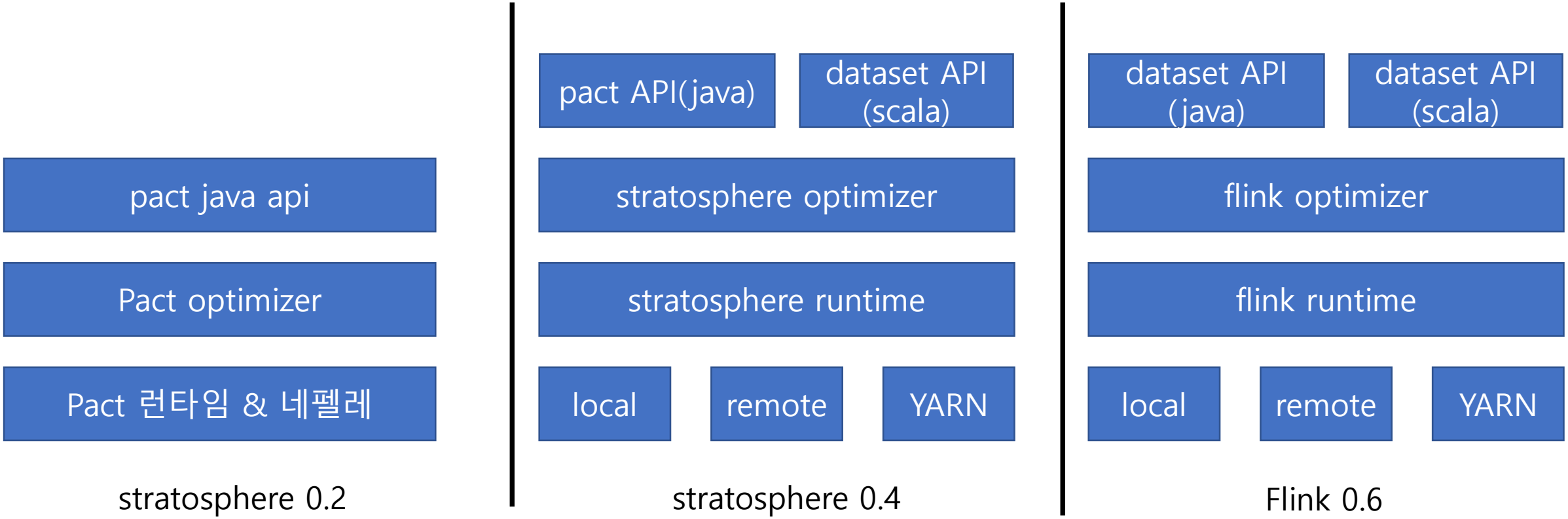
In recent years Cloud Computing has emerged as a promising new approach for ad-hoc parallel data processing. Major cloud computing companies have started to integrate frameworks for parallel data processing in their product portfolio, making it easy for customers to access these services and to deploy their programs. However, the processing frameworks which are currently used stem from the field of cluster computing and disregard the particular nature of a cloud. As a result, the allocated compute resources may be inadequate for big parts of the submitted job and unnecessarily increase processing time and cost. In this paper we discuss the opportunities and challenges for efficient parallel data processing in clouds and present our ongoing research project Nephele. Nephele is the first data processing framework to explicitly exploit the dynamic resource allocation offered by today's compute clouds for both, task scheduling and execution. It allows assigning the particular tasks of a processing job to different types of virtual machines and takes care of their instantiation and termination during the job execution. Based on this new framework, we perform evaluations on a compute cloud system and compare the results to the existing data processing framework Hadoop.

1. INTRODUCTION

Today a growing number of companies have to process huge amounts of data in a cost-efficient manner. Classical representatives for these companies are operators of Internet search engines, like Google, Yahoo or Microsoft. The vast amount of data they have to deal with every day has made traditional database solutions prohibitively expensive [5]. Instead, these companies have popularized an architectural paradigm that is based on a large number of shared-nothing commodity servers. Problems like processing crawled documents, analyzing log files or regenerating a web index are split into several independent subtasks, distributed among the available nodes and computed in parallel.

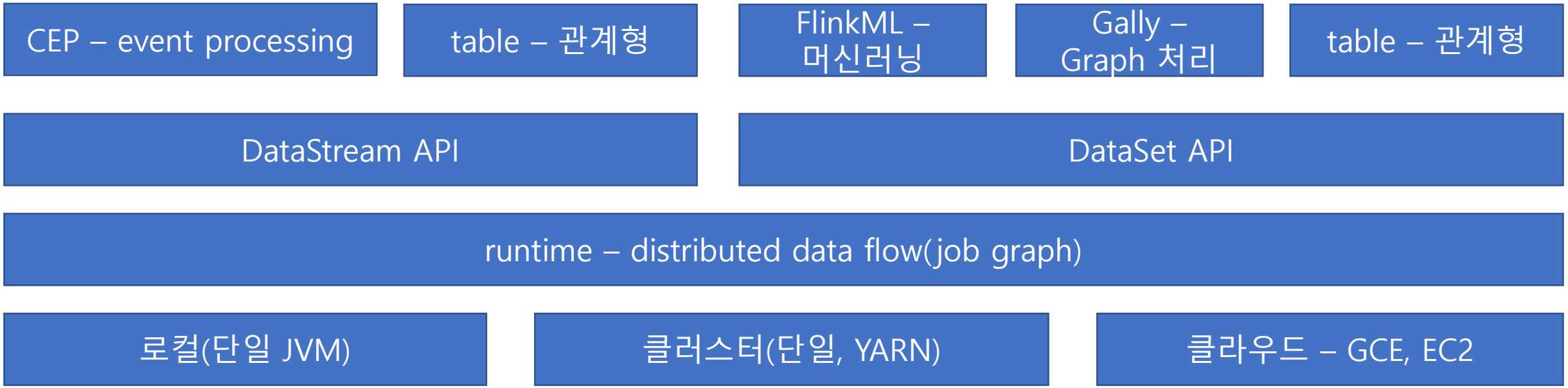
In order to simplify the development of distributed applications on top of such architectures, many of these companies have also built customized data processing frameworks in recent years. Examples are Google's MapReduce engine [9], Microsoft's Dryad [13], or Yahoo!'s Map-Reduce-Merge [6]. They can be classified by terms like high throughput computing (HTC) or many-task computing (MTC), depending on the amount of data and the number of tasks involved in the computation [19]. Although these systems differ in design, the programming models they provide share similar objectives, namely hiding the hassle of parallel pro-

apache Flink 역사

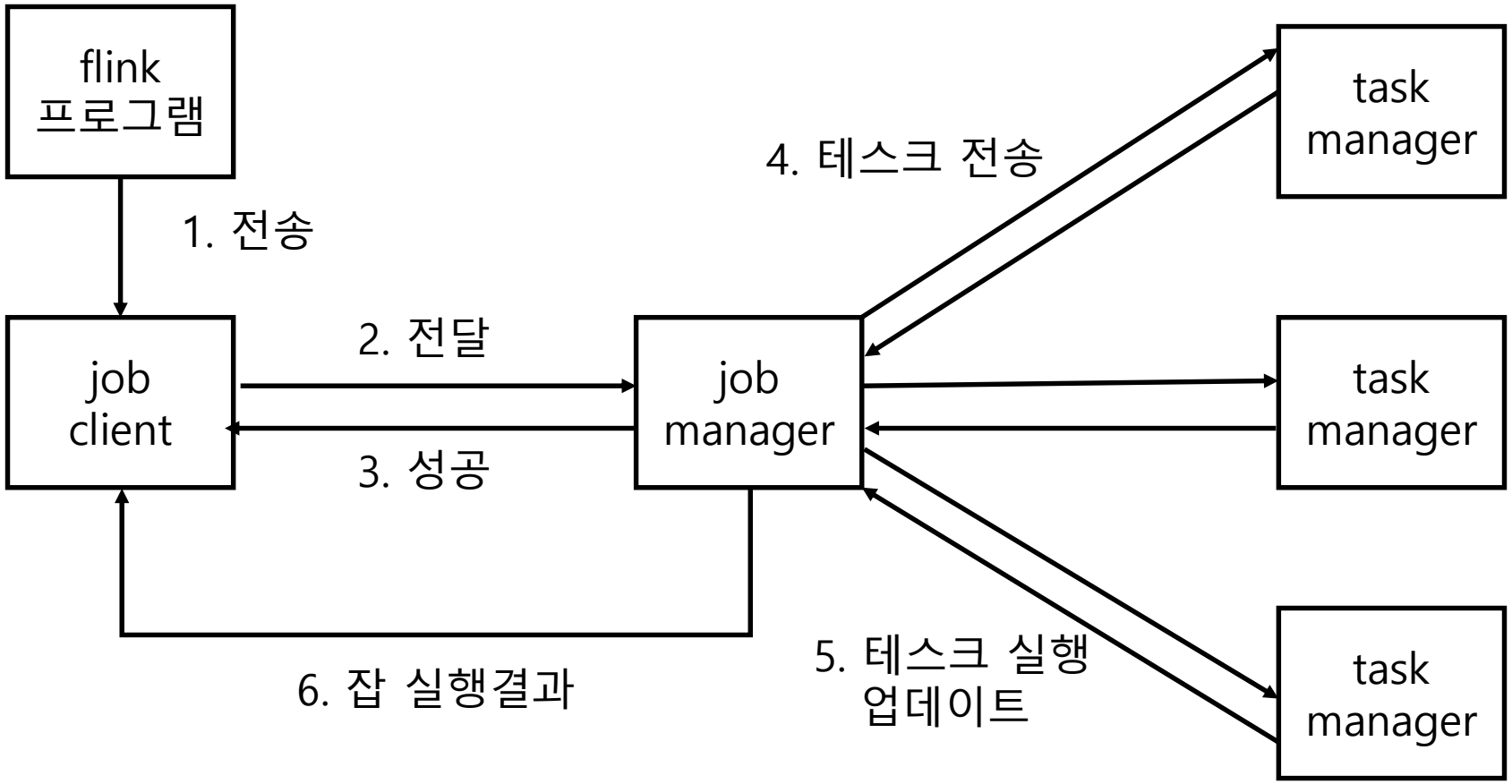


0.7 버전부터 스트리밍 API 제공(scala도 같이 제공)

Flink Architecture



분산형 구조

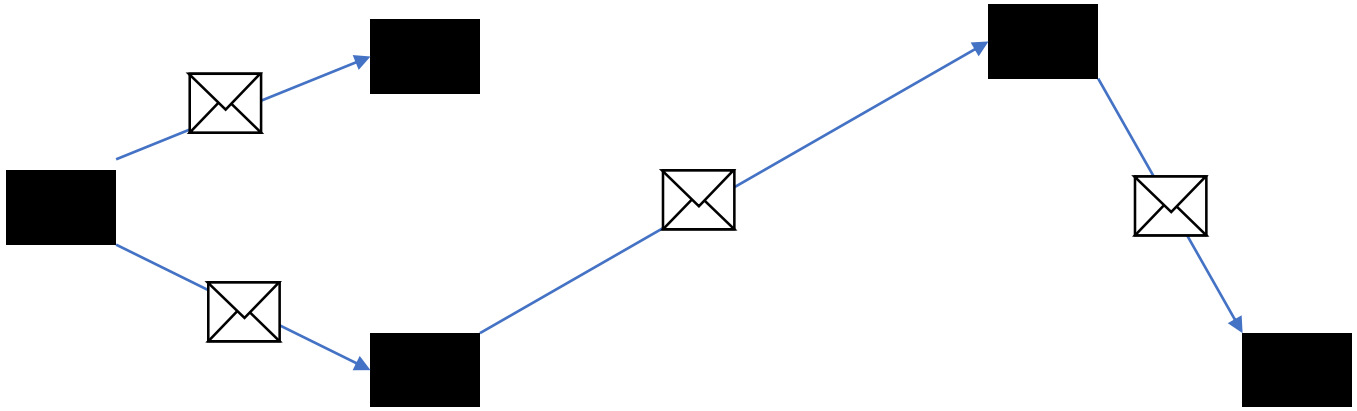

flink
프로그램
1. 전송
job
client
2. 전달
3. 성공
job
manager
4. 테스크 전송
task
manager
task
manager
task
manager
5. 테스크 실행
업데이트
6. 잡 실행결과

job manager

- 마스터 프로세스로서 실행을 관장하고 관리
- task scheduling, checkpoint management, failure recovery
- 여러 개의 job manager 실행이 가능하며 역할공유 가능
- multi일 경우 리더가 필요하며 리더가 죽을경우 스탠바이가 리더
- 구성요소
  - Actor system
  - Scheduler
  - Check pointing

AKKA Actor system

- scheduling, env config, logging 등의 역할
- 모든 actor를 초기화하도록 thread pool 구성
- hierarchy 구조, 새로운 actor는 해당 parent(supervisor)에 할당
- 메일 박스를 통해 메시지를 읽는다(로컬 공유메모리, 원격 RPC)
- state와 behavior를 지닌 컨테이너

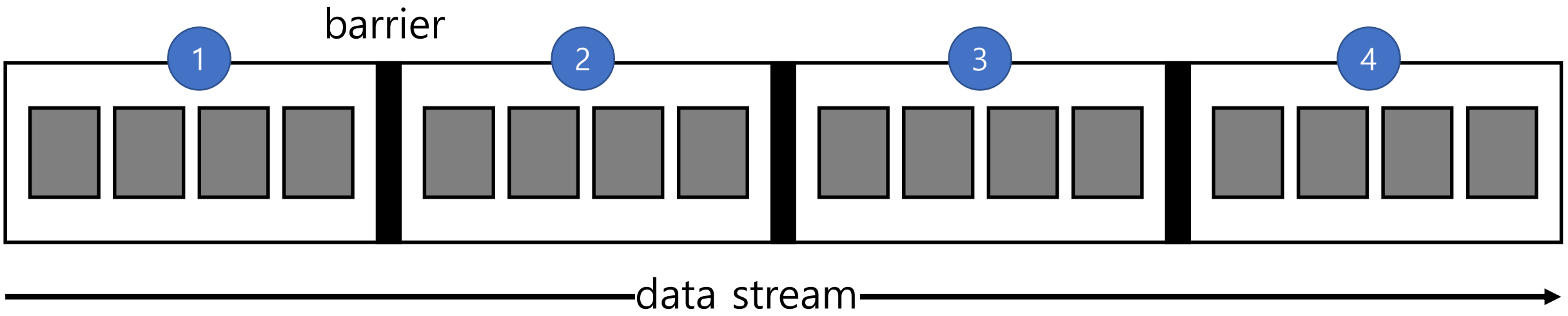


Scheduler

- executor는 task slot으로 정의
- 각 task manager는 적어도 하나 이상의 task slot관리
- 내부적으로 어떤 task가 slot을 공유해야 하는지 결정
- 어느 task가 특정 slot에 있어야 하는지 결정
- slotSharingGroup과 CoLocationGroup에서 결정

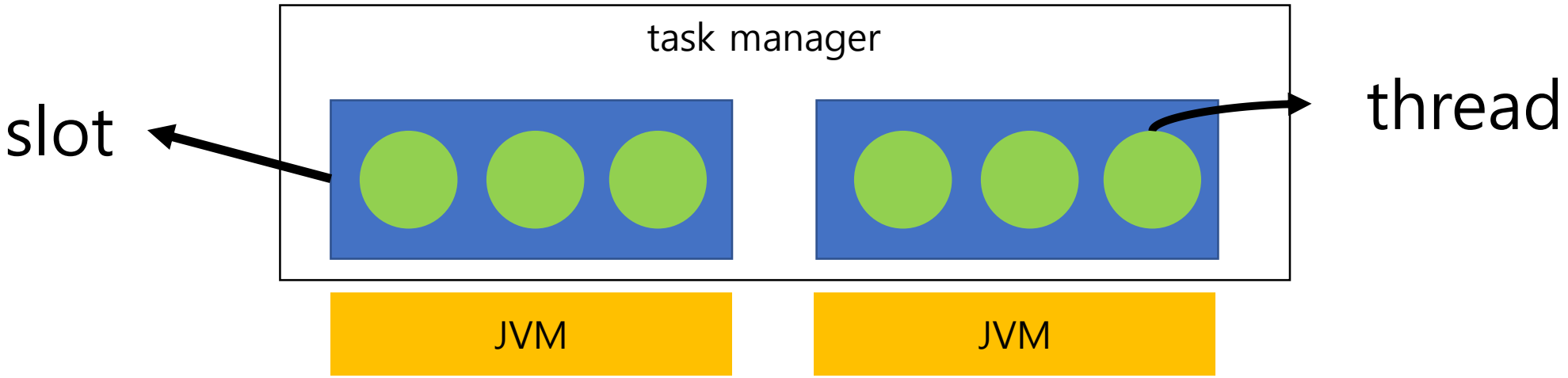
checkpoint

- 일관성 있는 fault-tolerance를 제공하는 핵심기능
- 일관성 있는 snapshot을 만들고 유지
- 장애가 발생하면 executor중단하고, 가장 최신 checkpoint 재실행
- stream barrier는 snapshot의 핵심요소(flow에 영향 받지 않게 함)



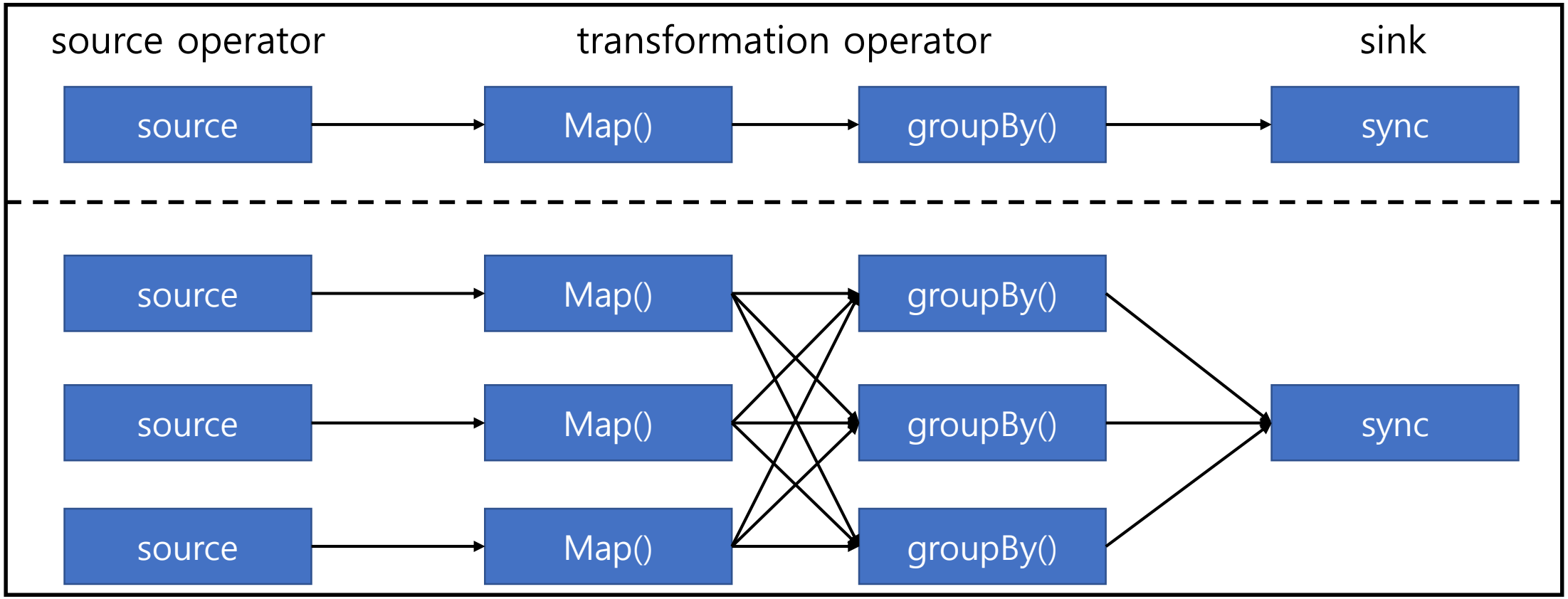
task manager

- worker node, jvm 내의 하나 이상의 thread에서 task 실행
- task를 병렬로 실행하는 것은 task slot를 통해 결정
- task는 task slot에 할당된 resource set으로 표현
  (4개의 slot이면 각 slot에는 memory 25%씩 할당
- 동일 slot 내의 thread는 동일 jvm공유
- 동일 jvm에 있는 task는 TCP connection, heart beat message 공유

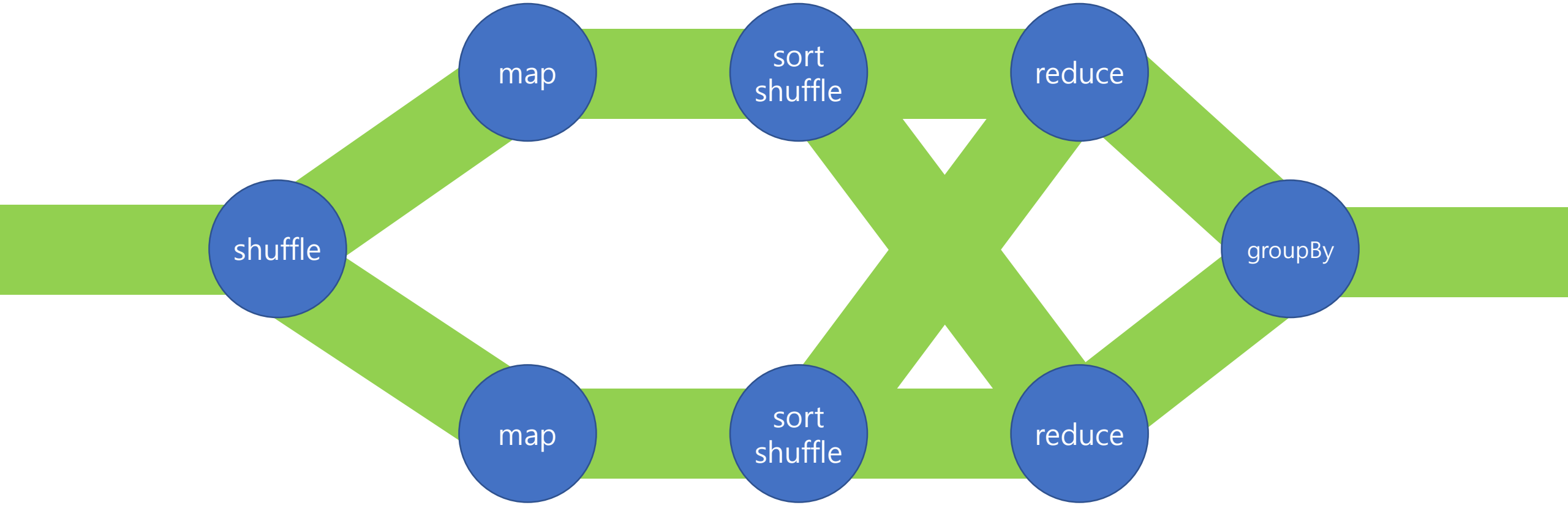


job client

- 프로그램을 받아 data flow 생성하고 job manager에게 전달



Streaming Dataflow Engine



Flink는 batch 데이터도 처리합니다.

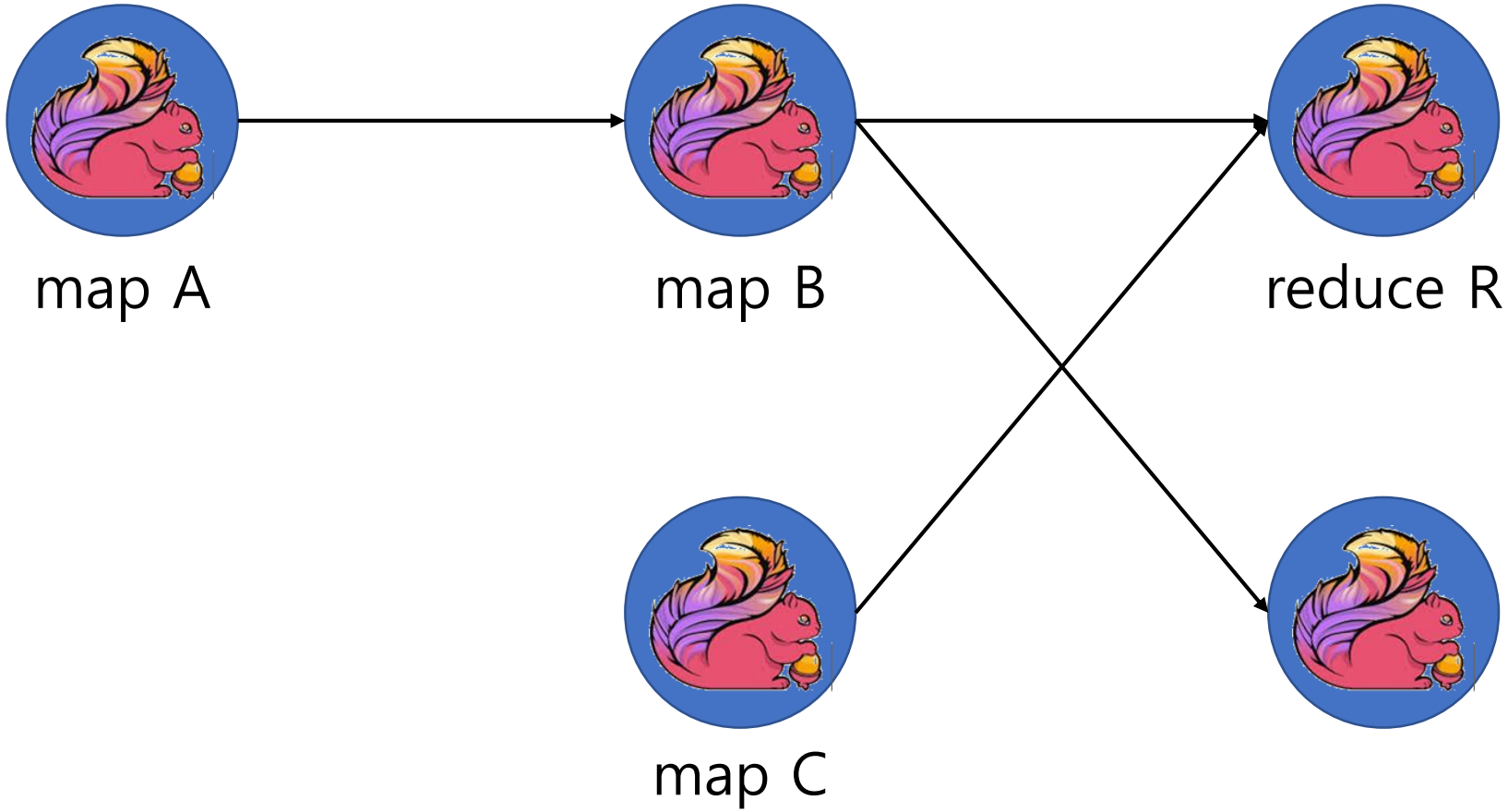
- batch data를 바라보는 Flink의 자세

Batch = special case of streaming

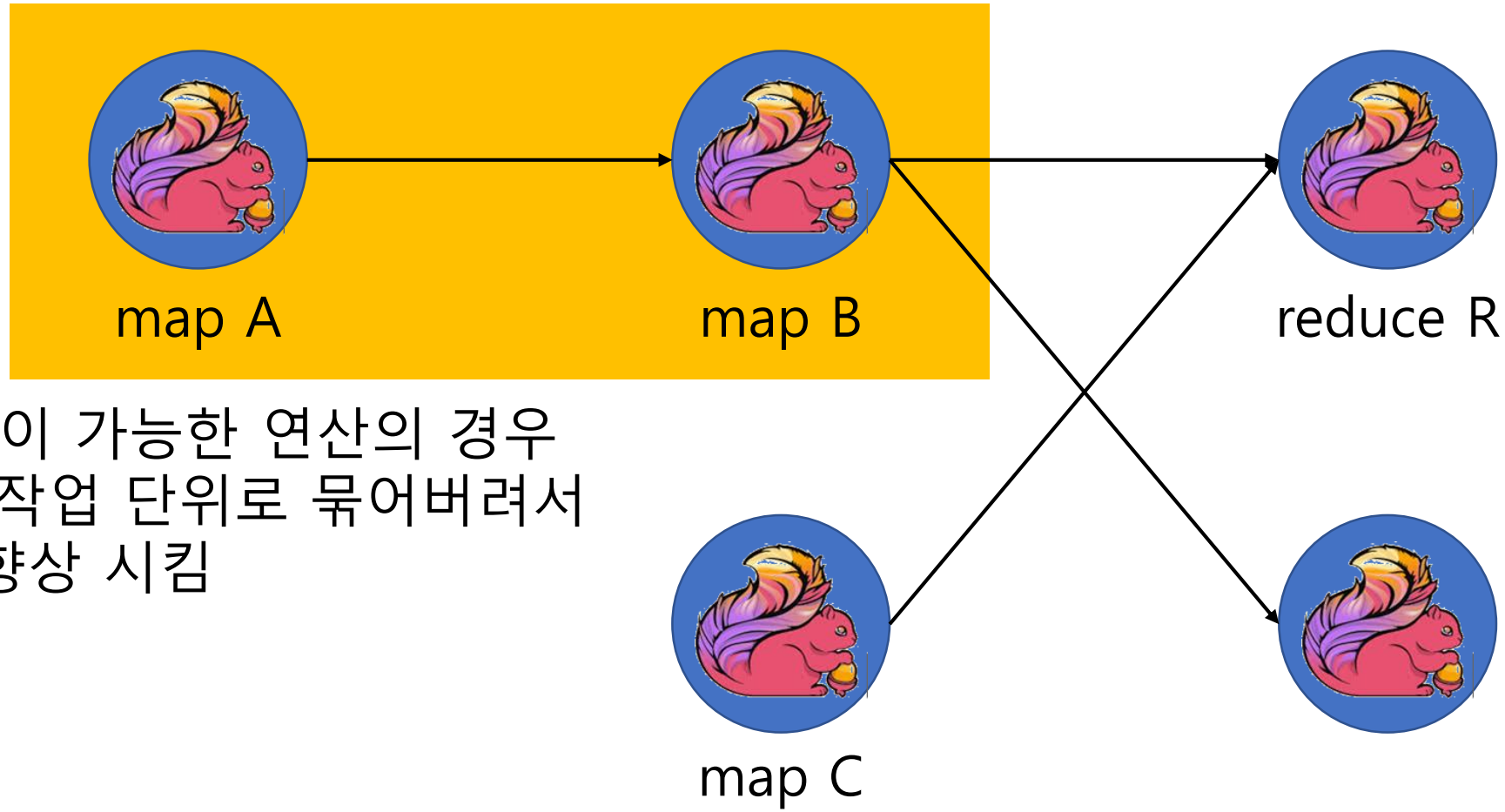
Apache Flink의 특징들

- 세련된 Java, Scala, Python API 제공
- Web UI, Scala Shell 등 다양한 도구 지원
- 그래프 처리, 기계 학습 등을 위한 라이브러리 제공
- 똑똑한 Flink 런타임의 자동 최적화
- 정확하게 한번씩만 처리하는 state 유지 방식
- 유연한 스트리밍 윈도우
- 자체 메모리 관리 지능
- 오픈소스 진영에서 가장 진보된 스트리밍 데이터 처리 시스템

Flink의 pipeline 최적화

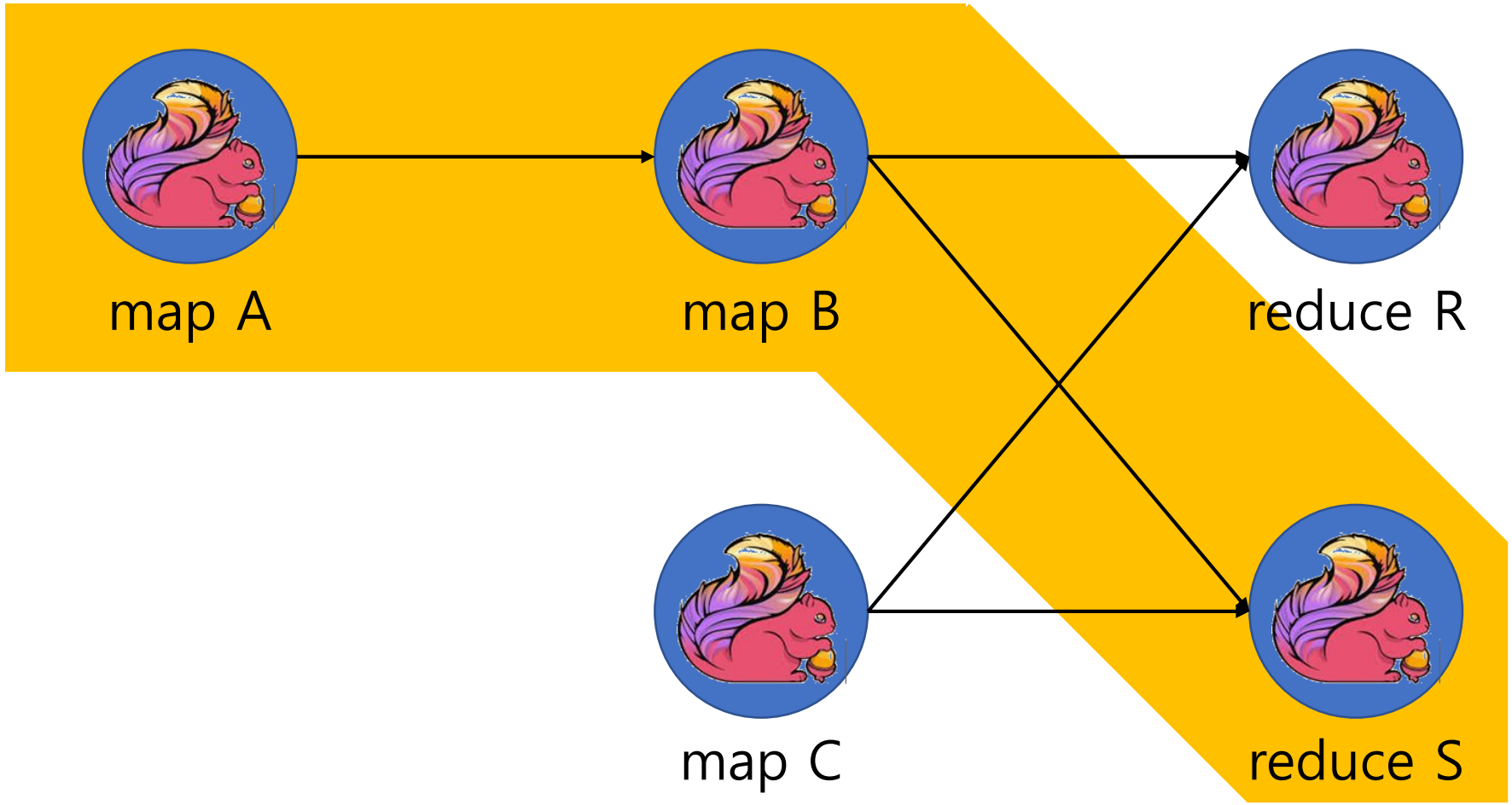


Flink의 pipeline 최적화



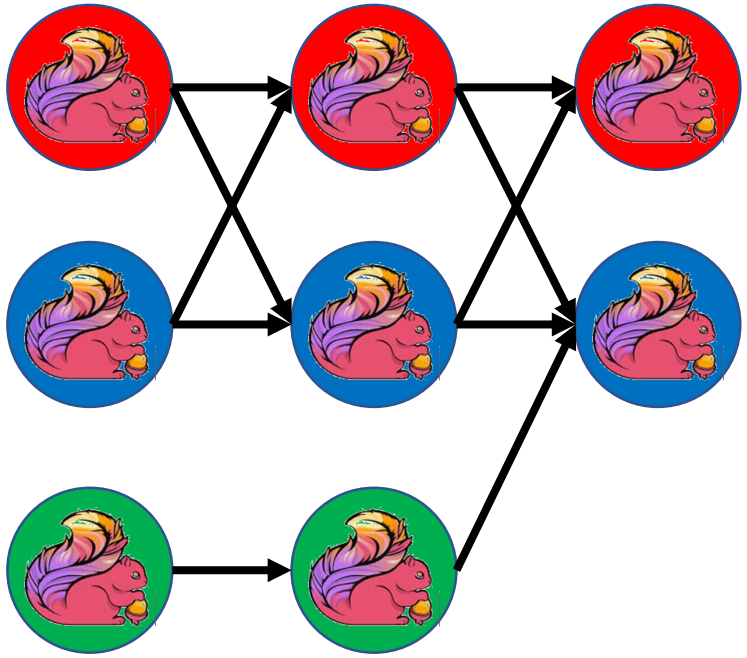
chaining이 가능한 연산의 경우
그냥 한 작업 단위로 묶어버려서
성능을 향상 시킴

Flink의 pipeline 최적화

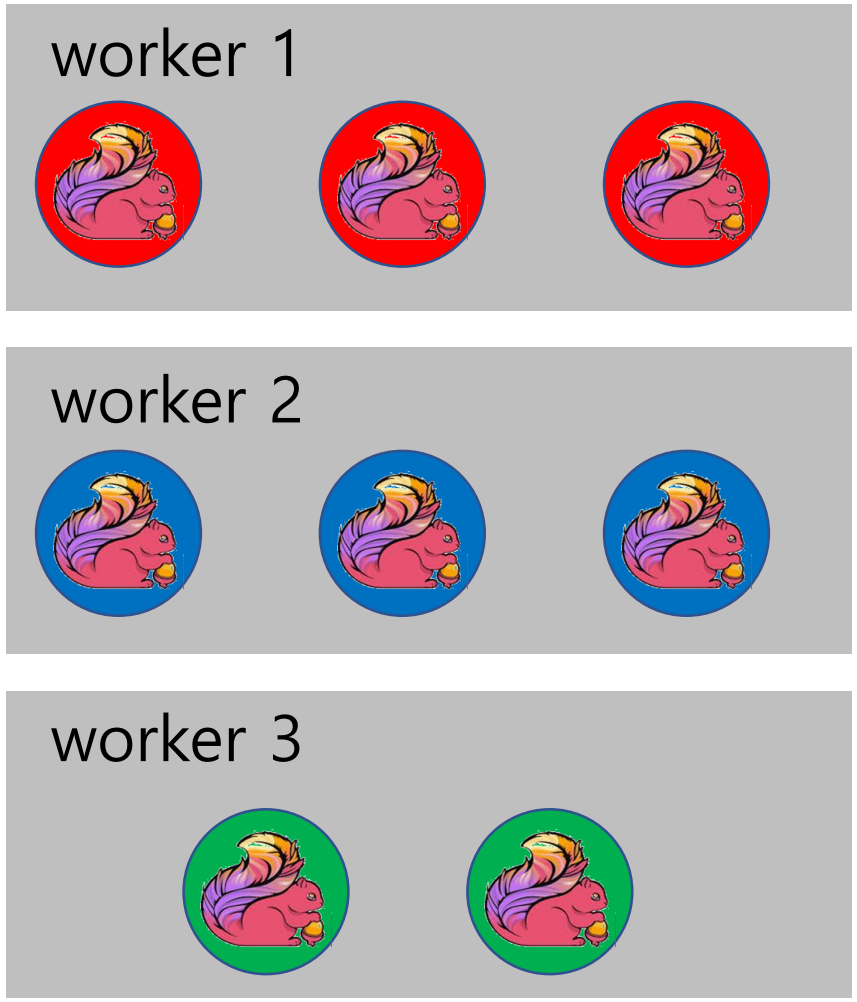


shuffle이 적게
일어나도록 할당

Flink의 pipeline 최적화



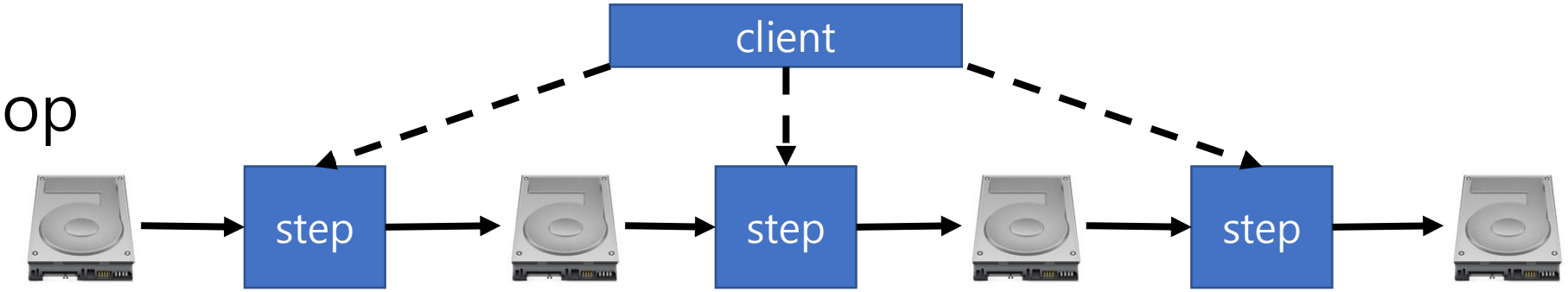
Job Graph



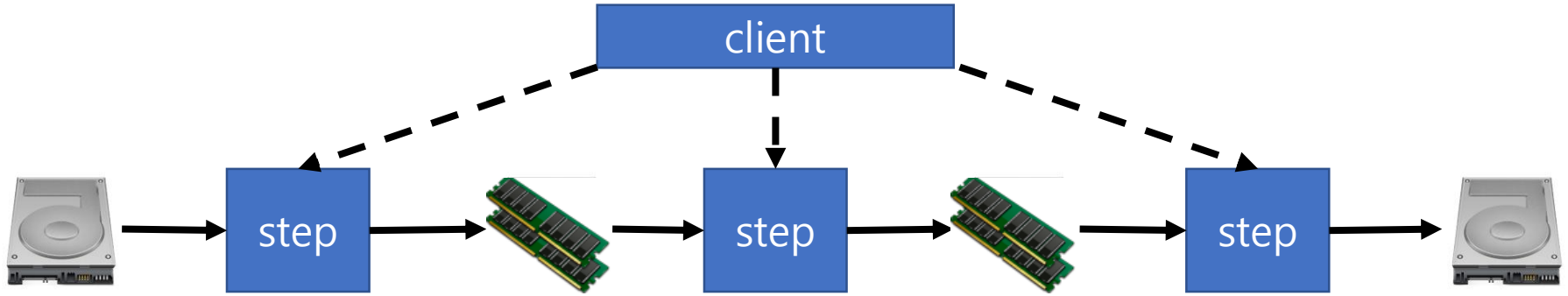
Execution Graph

Flink의 native iteration

- hadoop

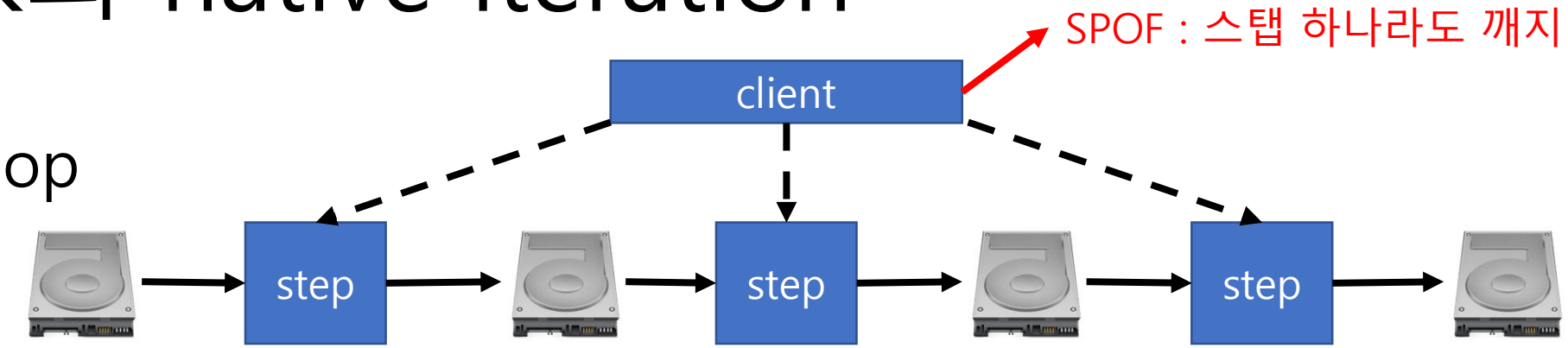


- spark

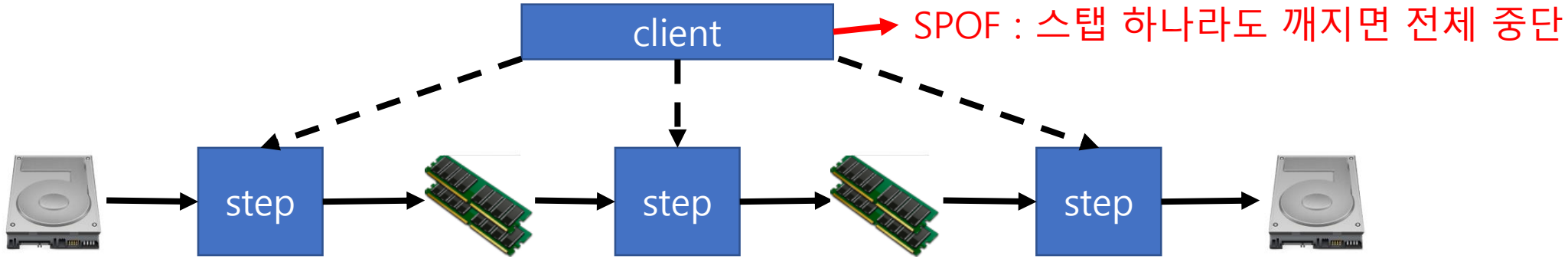


Flink의 native iteration

- hadoop

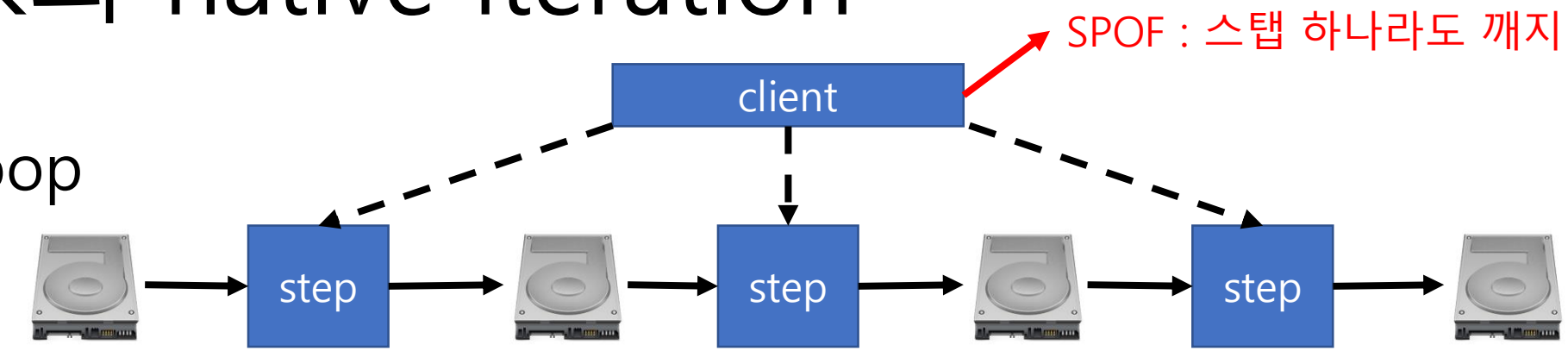


- spark

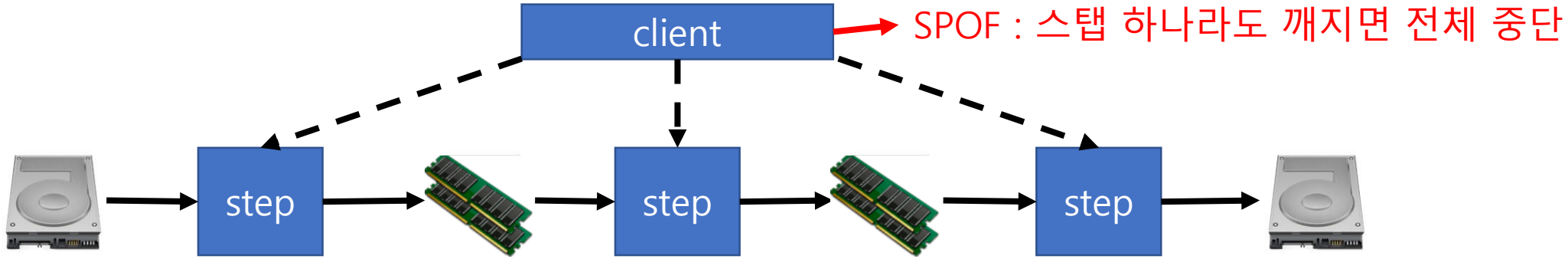


Flink의 native iteration

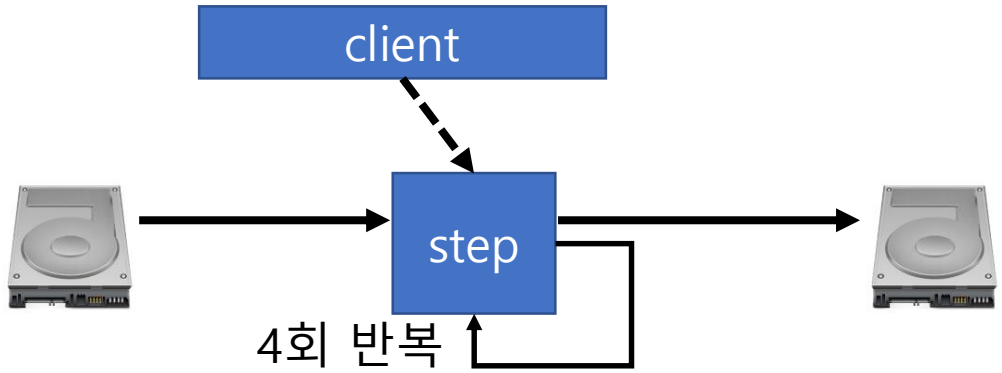
- hadoop



- spark



- Flink



자체 메모리 관리 근데 이제 GC를 곁들이지 않은

user code
Deserialized data

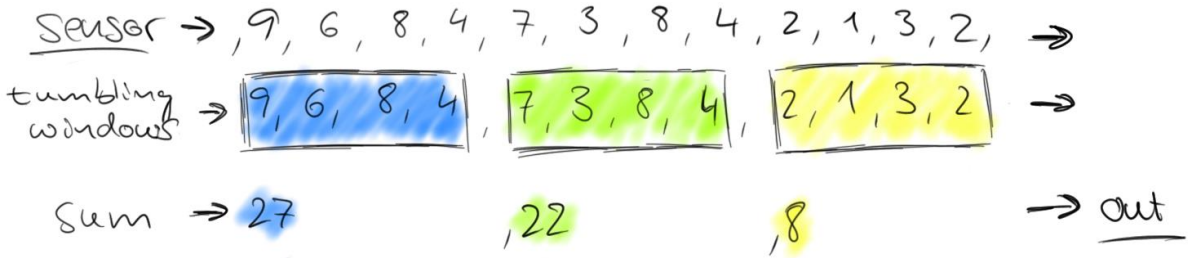
hashing
sorting
caching

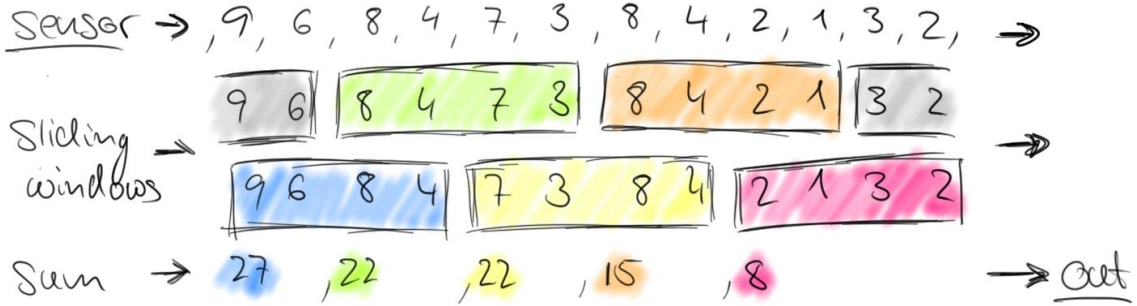
shuffles
Broadcasts

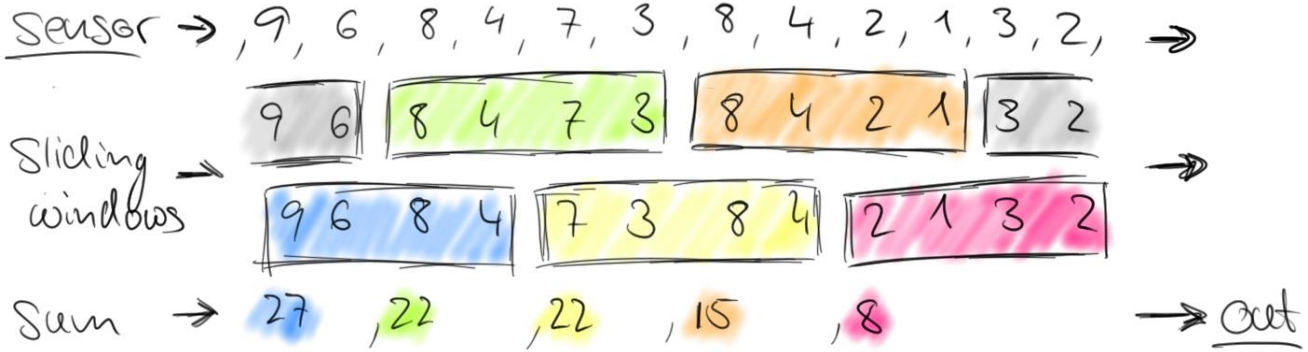
- Flink는 메모리를 직접 관리
  - GC로 인한 성능감소 적음
- SerDe오버헤드 줄이기 위해 바이너리 데이터에서 직접연산
  - 타입 정보를 미리 파악
  - ~~성능변태~~
- 메모리 튜닝이 거의 필요없음
- Spark는 버전 1.4부터 지원

유연한 streaming window

- data driven window
- time, count, session info를 바탕으로 window 디자인
- event stream에서 특정패턴 추적하도록 customizing 가능







Flink API

- DataStream API
- batch processing API
- Table API
- Complex Event Processing
- FlinkML
- Gelly – graph API