하둡 (Hadoop) 실행 단계 겹치기

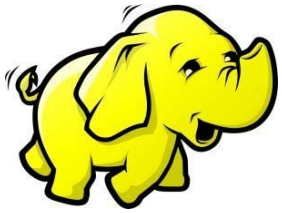
김동원
POSTECH

목차

빅데이터 & 오늘 발표

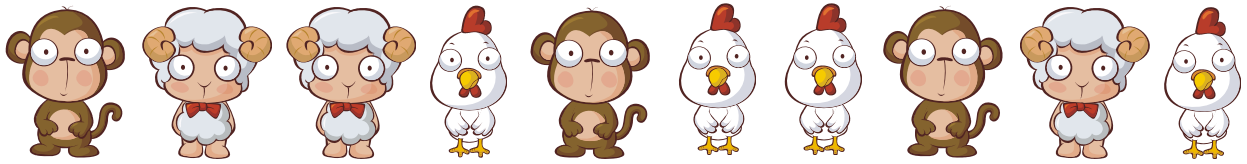
- **빅데이터** 정의 (정보통신산업진흥원, ETRI)
  - "기존 데이터에 비해 **너무 커서**, 기존 방법이나 도구로 수집, 저장, 검색, 분석, 시각화하기 어려운 정형 또는 비정형 데이터"

- 오늘 발표
  - **빅데이터 처리 프로그램**의 성능 향상
    - 하둡 (Hadoop)



동물 별 평균 몸무게 측정

1

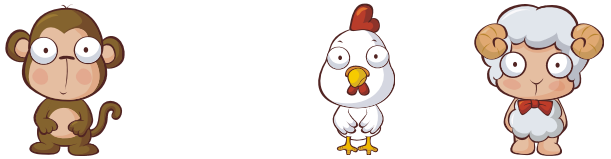


2

동물	무게
원숭이	[7, 8, 9]
양	[150, 145, 155]
닭	[3.3, 3.6, 3.9, 3.6]

3

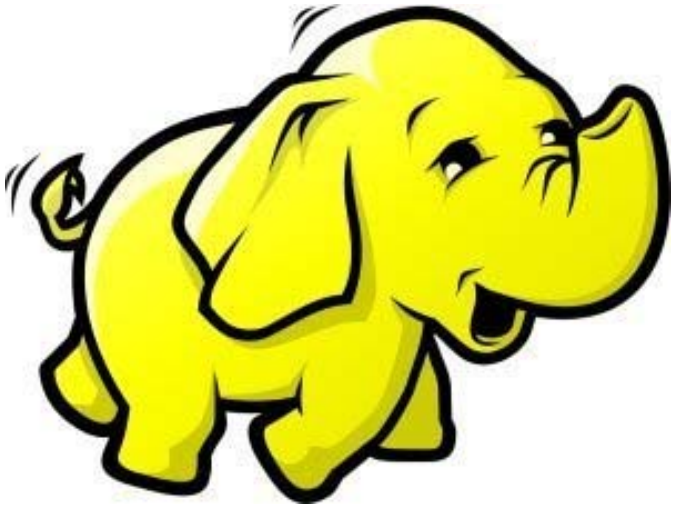
평균 몸무게 =
무게 합/개체 수



샘플 – **세상 모든** 원숭이, 닭, 양

- 문제 자체의 난이도는 쉬움
- 5분 안에 계산해야 한다면?
  - 10,000대의 컴퓨터를 사용하면 5분 안에 가능!
  - 복잡해짐 (작업 스케쥴링, 동기화, 오류 제어, …)

하둡 (Hadoop)



1. 프로그래밍 모델

   - 사용자는 **작은 데이터**를 처리하듯 프로그래밍 가능
     - Map과 Reduce라는 함수만 작성하면 끝!

2. 하둡 런타임 (runtime)
   - 잡일 처리 당번
     - 작업 스케쥴링
     - 작업들 동기화
     - 오류 처리
     - …

하둡 실행 단계들

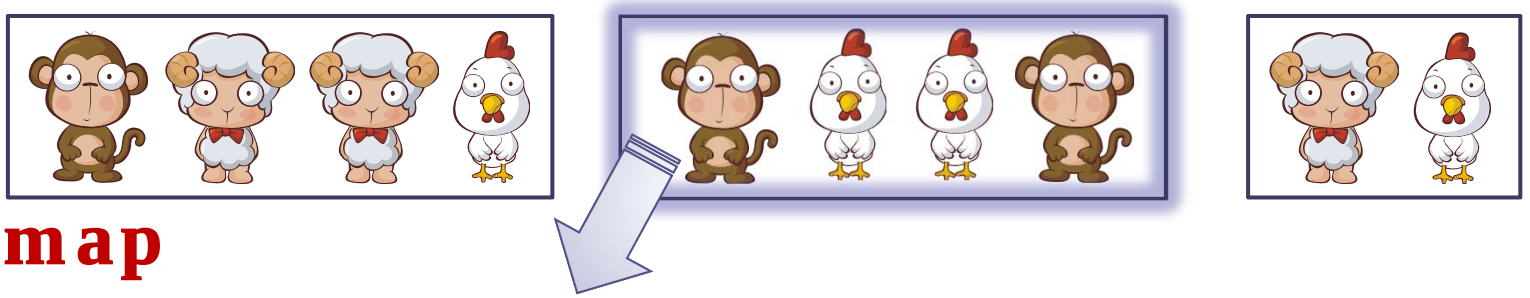
1) **쌍** 만들기 (Mapping)
   - 사용자 정의 **Map** 함수 적용
   - (key, value) 형태의 **쌍**들을 추출

2) 모으기 (Shuffling)
   - 런타임이 **같은 Key를 가지는 쌍**끼리 모아줌

3) 줄이기 (Reducing)
   - 사용자 정의 **Reduce** 함수 적용
   - **값은 Key**를 가지는 **쌍**들에 대한 연산 수행

(key, value) 형태의 **쌍**들을 추출

1) 쌍 만들기 (Mapping)



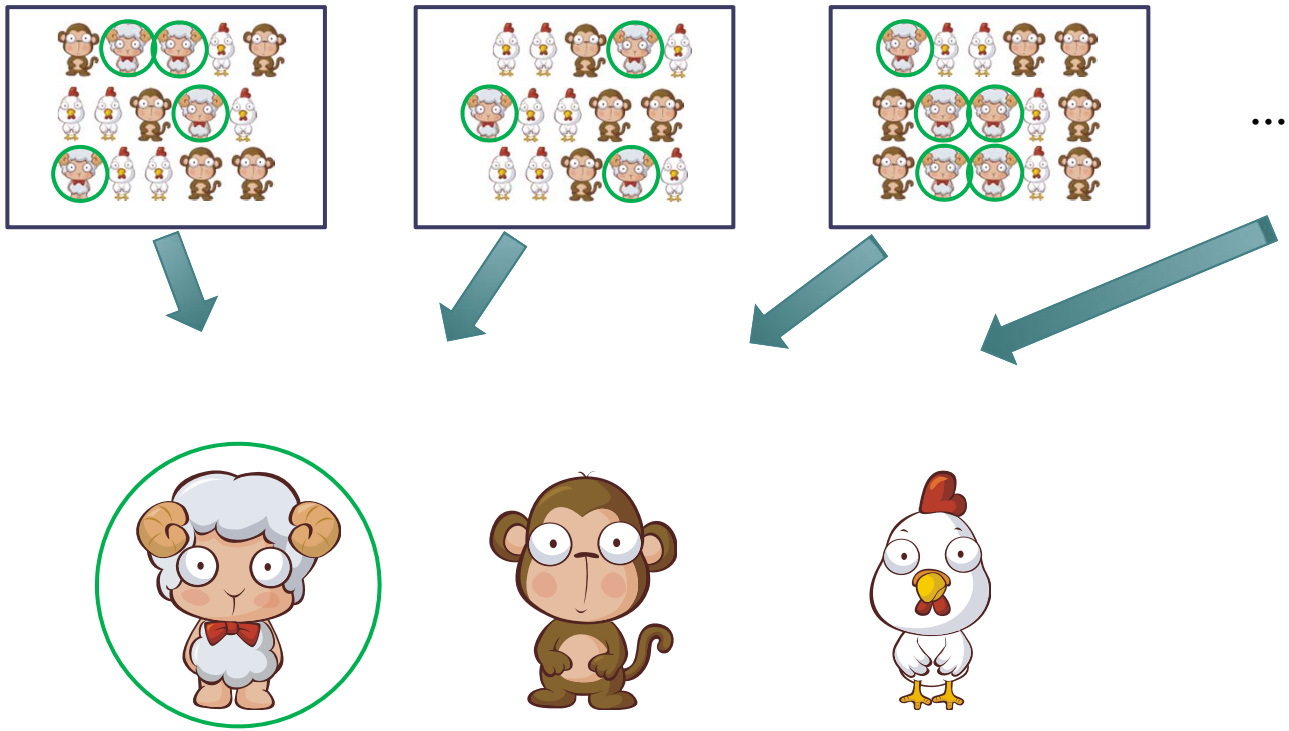
- **map**
  - 호출 : **map**([원숭이2, 닭2, 닭3, 원숭이3])
  - 결과 : [(원숭이, 8 kg), (닭, 3.6 kg), (닭, 3.9 kg), (원숭이, 9 kg)]

- def **map**(animals):
  for animal **in** animals:
  collect(animal.kind, animal.weight)

겨우 2줄!

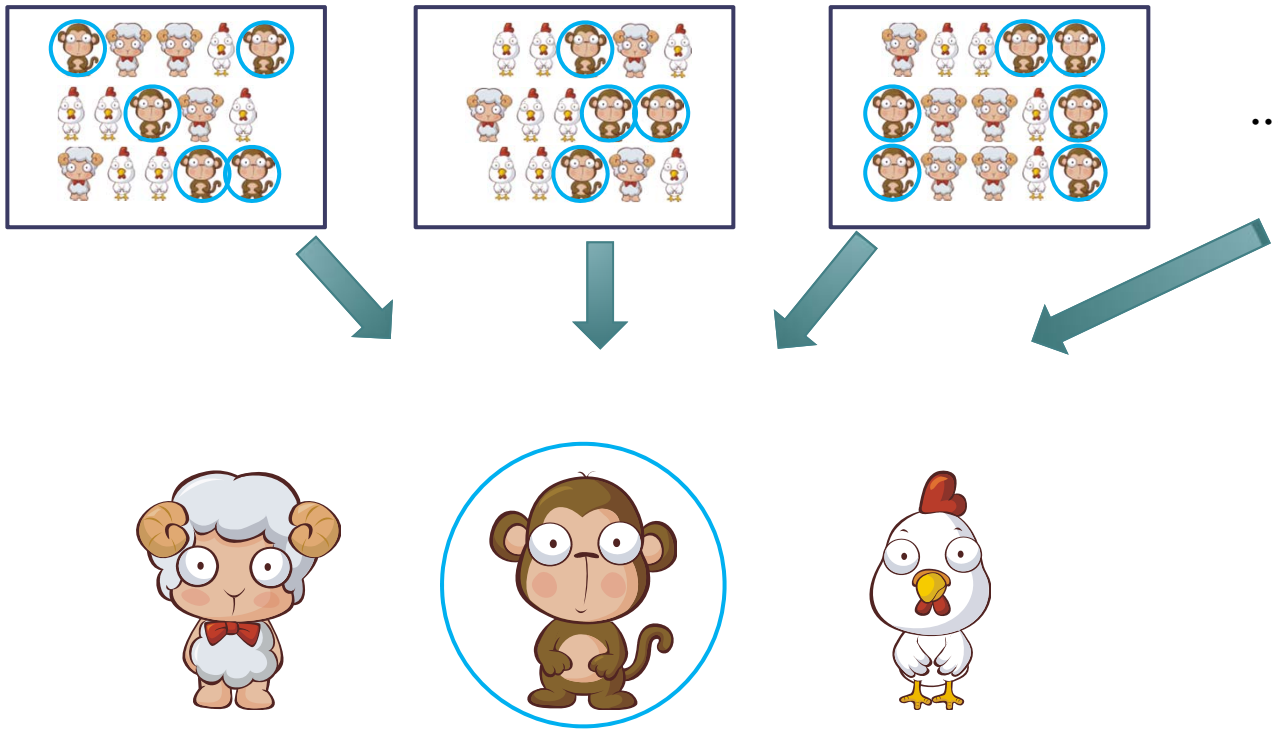
런타임이 **같은 Key를 가지는 쌍**끼리 모아줌

2) 모으기 (Shuffling)



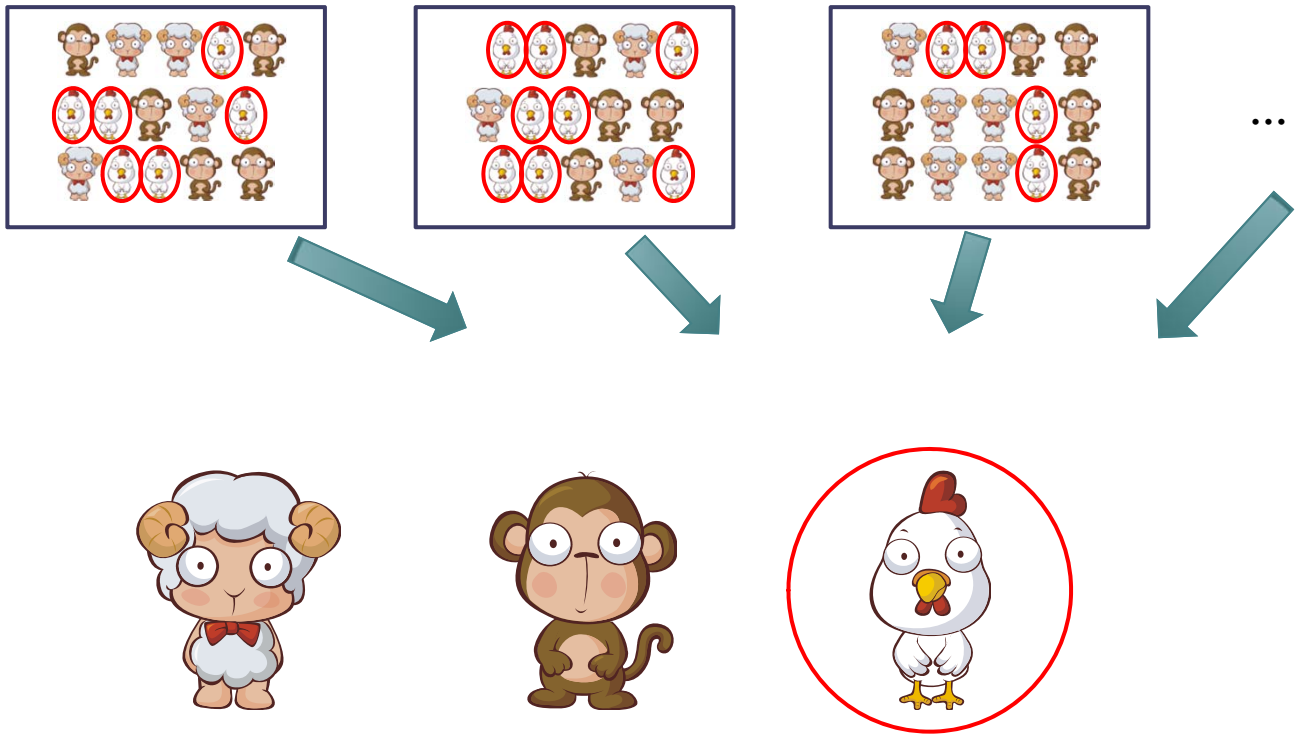
런타임이 **같은 Key를 가지는 쌍**끼리 모아줌

2) 모으기 (Shuffling)



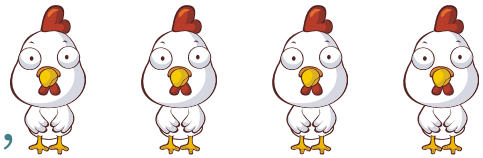
런타임이 **같은 Key를 가지는 쌍**끼리 모아줌

2) 모으기 (Shuffling)



값은 Key를 가지는 모든 **쌍** 대한 연산 수행

3) 줄이기 (Reducing)

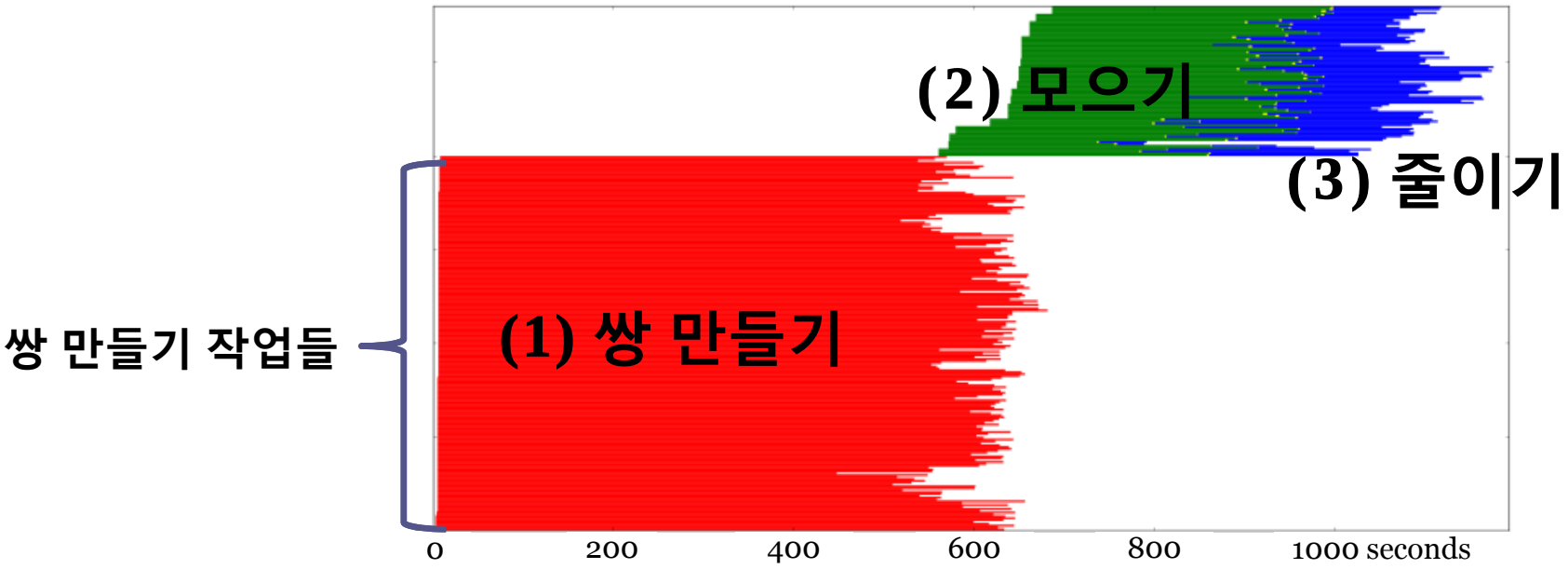
- **Reduce**
  - 호출 : **reduce**(닭, 
    [3.3, 3.9, 3.6, 3.9])
  - 결과 : 닭, (3.3 + 3.9 + 3.6 + 3.9)/4

겨우 1줄!

- def **reduce**(kind, weights):
  write(kind, sum(weights)/len(weights))

목차

하둡 런타임의 한 가지 문제! (여러 문제 중..)

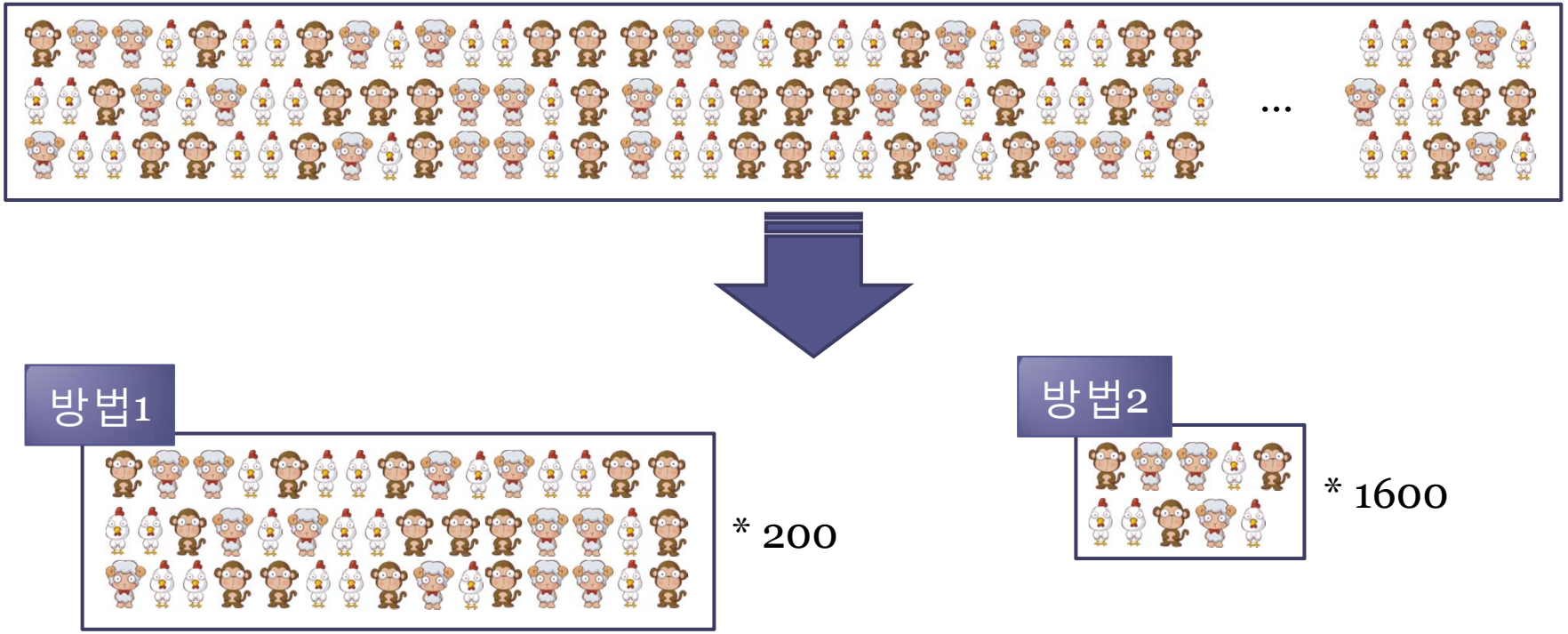


- **쌍 만들기 작업 수**가 성능에 큰 영향을 미침
  - 하나의 **입력 조각**을 하나의 **쌍 만들기 작업**이 처리

목표

1) 쌍 만들기 작업 수와 무관하게 성능 잘 나오게 하기
2) ??

입력을 나누는 방법



- 하나의 **입력 조각**을 하나의 **쌍 만들기 작업**이 처리

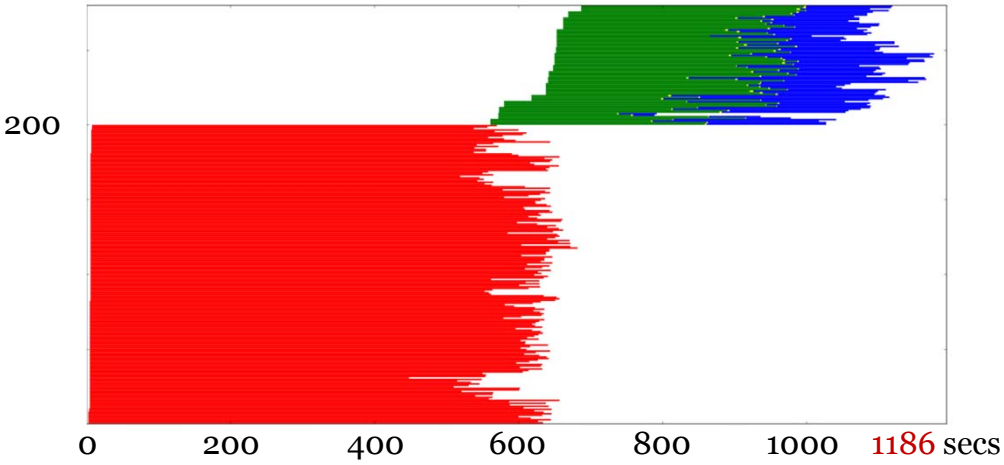
쌍 만들기 작업 수와 하둡 성능

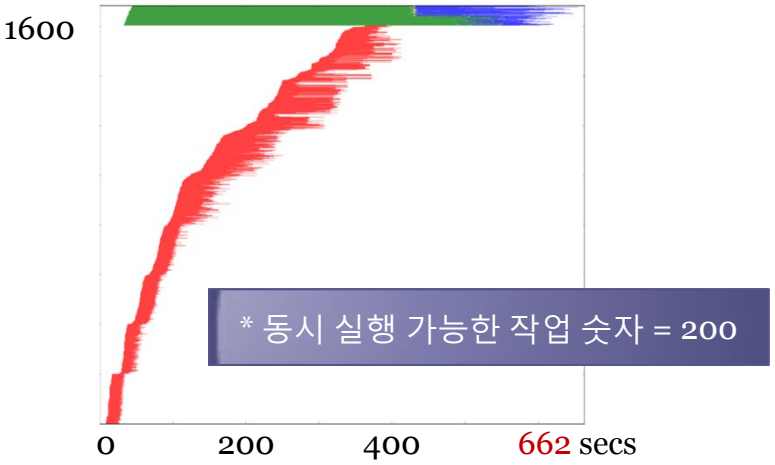
방법1

* 200

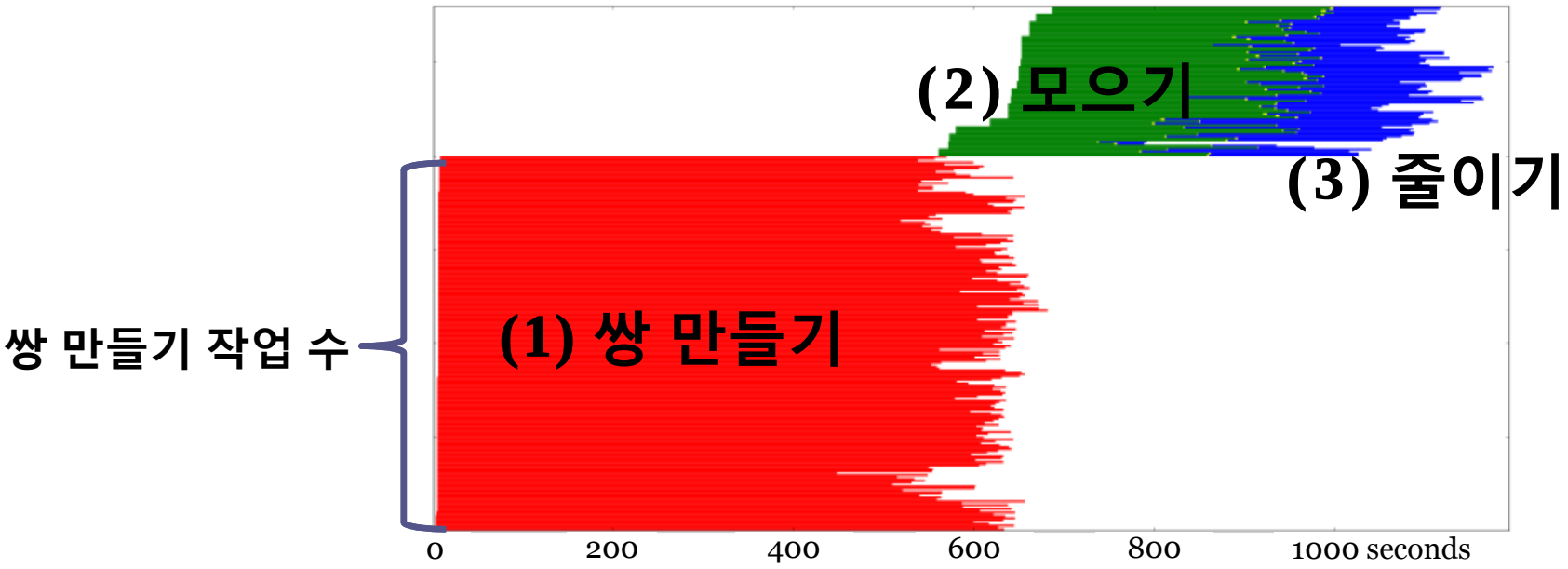
방법2

* 1600



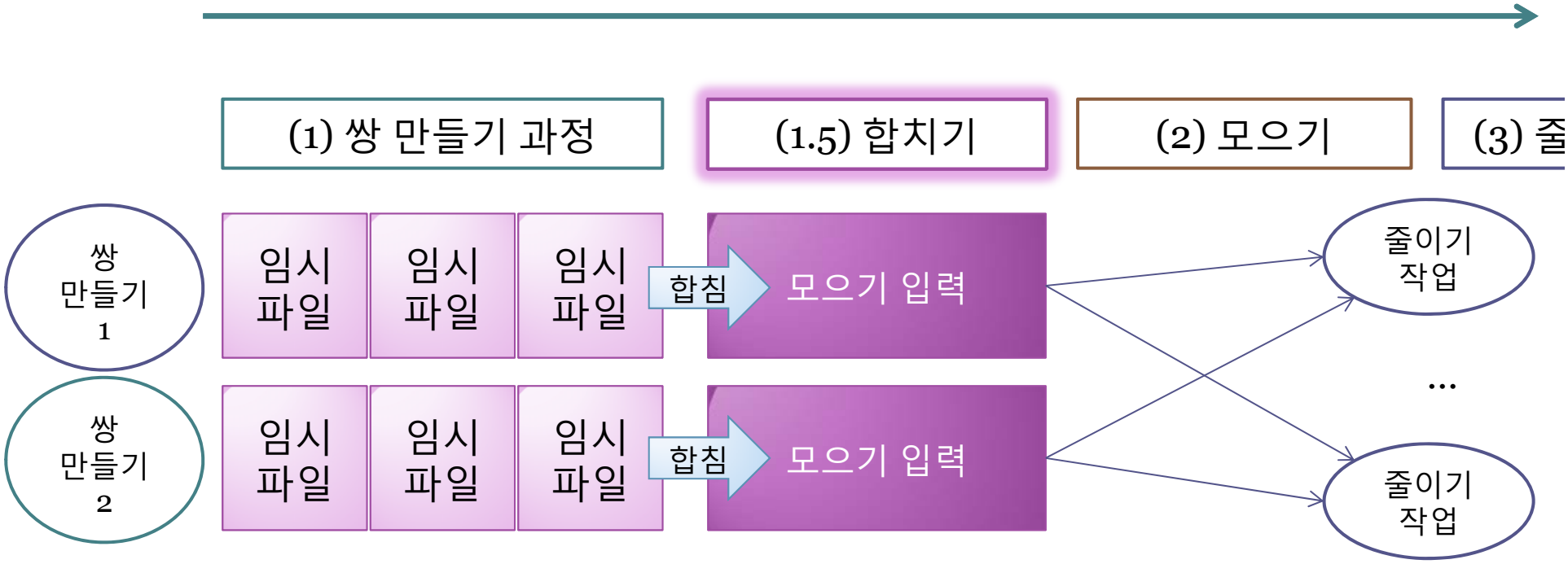


하둡 런타임 특징



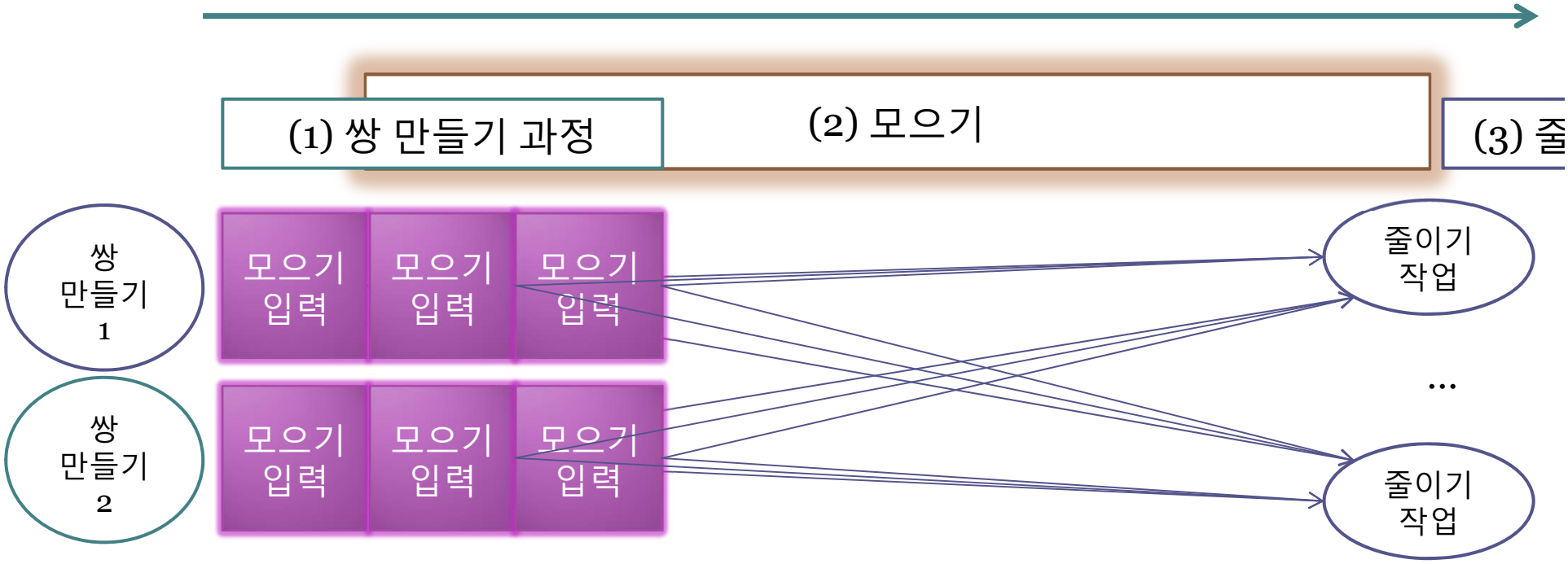
- '**(2) 모으기**'는 몇 개의 '**(1) 쌍 만들기**'가 끝난 후에 시작

하둡 실행 단계들 (약간 기술적으로..)



- (1) 쌍 만들기, (1.5) 합치기, (2) 모으기가 동시에 진행이 안됨

적은 쌍 만들기 작업들로 과정 겹치기

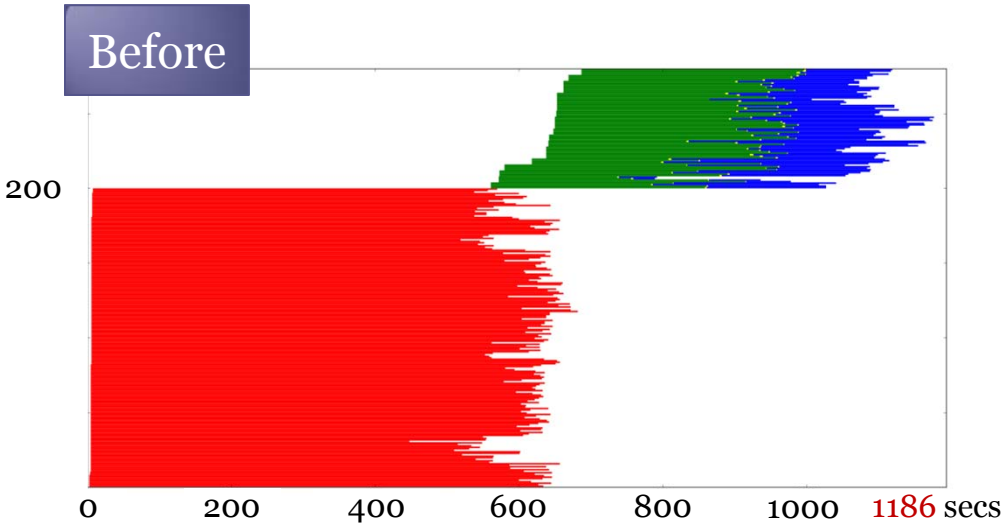


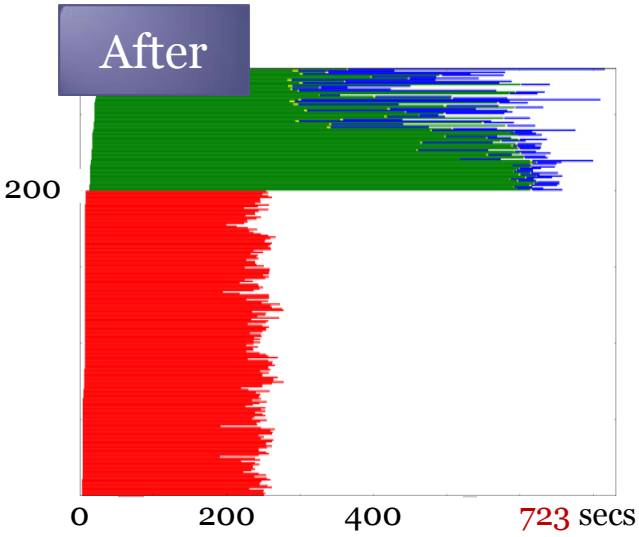
- 입력 조각을 작게 하지 않더라도
  (1) 쌍 만들기, (2) 모으기가 **동시에** 진행됨

적은 쌍 만들기 작업들로 과정 겹치기

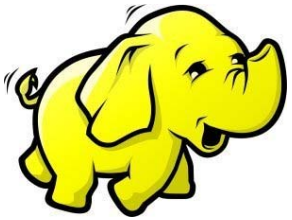
- Before/After 모두 200개의 쌍 만들기 작업들 생성

* 200





왜 하둡은 (1.5) **합치기**를 할까?



- **오류 복구 알고리즘**을 쉽게 만들기 위해서!

- **오류 복구 알고리즘**의 중요성
  - **7일** 동안 꼬박 계산을 해야 하는 작업이 있는데, **하나의 쌍 만들기 작업** 결과가 **6일 째 밤 손실**되었다면?
    - 처음부터 다시? 7일 동안?
    - **그 쌍 만들기 작업만** 다시 실행!

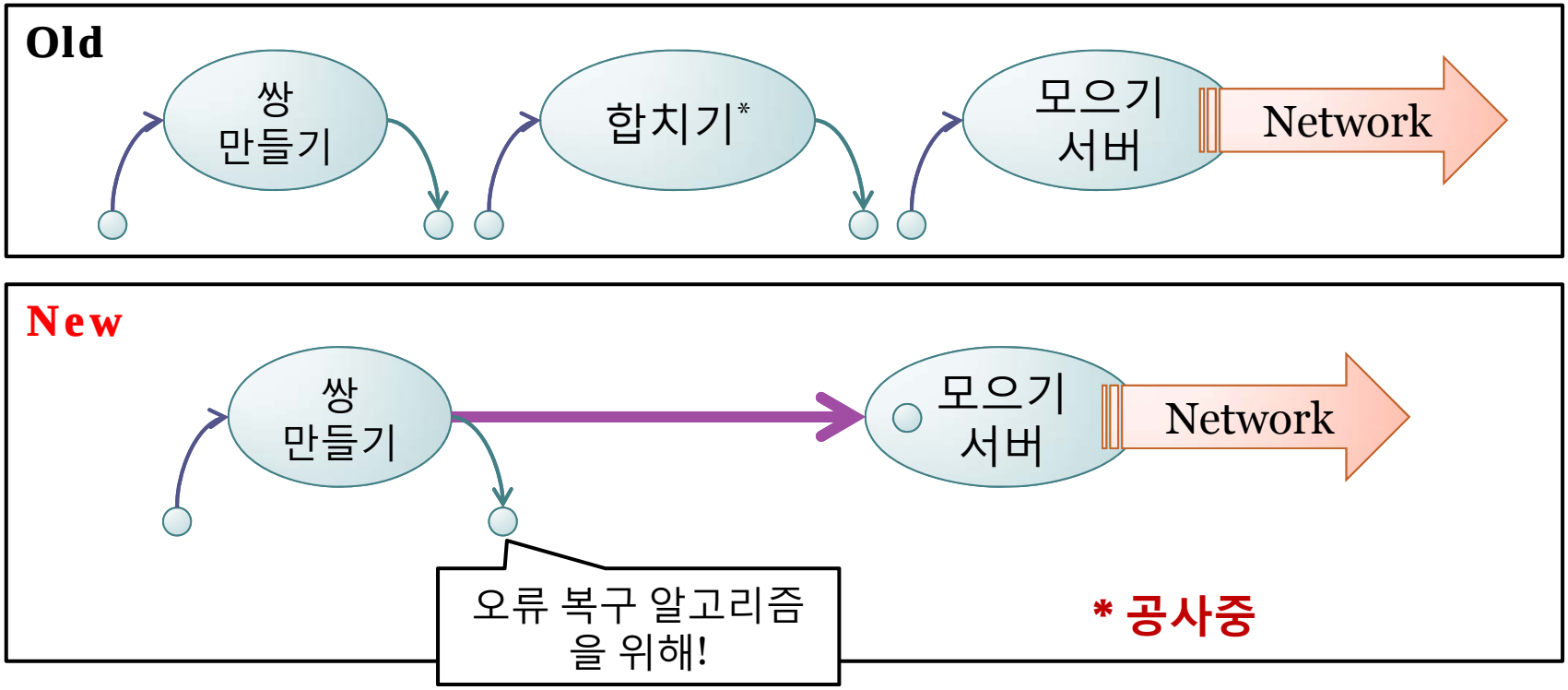
제안의 장/단점 : “살을 주고 뼈를 친다!”

- 살(단점)
  - 오류 복구 알고리즘이 ***약간*** 복잡해짐
- 뼈(장점)
  - 사용자 편의성
    - 입력을 몇 개의 조각으로 나눌지 걱정 하지 않아도, 항상 좋은 성능이 나옴

목표

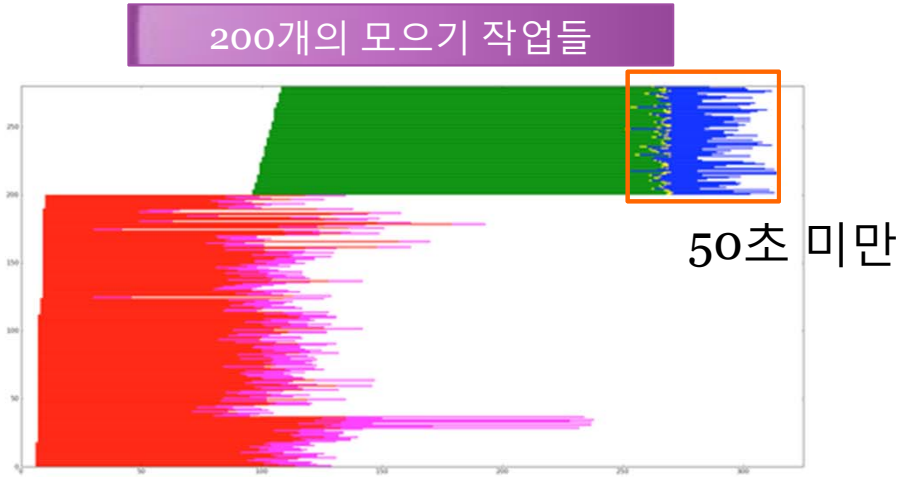
1) 쌍 만들기 작업 수와 무관하게 성능 잘 나오게 하기
2) 단계를 겹칠 때, 디스크를 효율적으로 사용하기
   - **단순히 겹치는 것 이상**으로 성능 향상 가능!

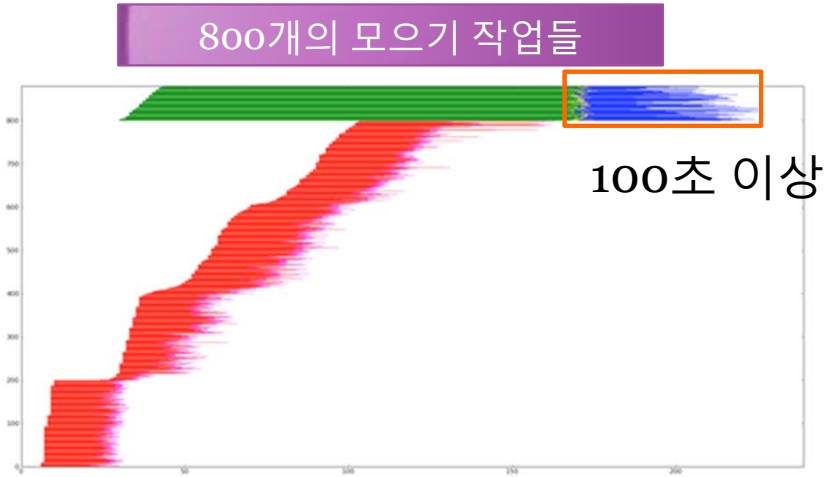
디스크 접근 횟수 줄이기


Old
쌍 만들기
합치기*
모으기 서버
Network
New
쌍 만들기
모으기 서버
Network
오류 복구 알고리즘 을 위해!
* 공사중

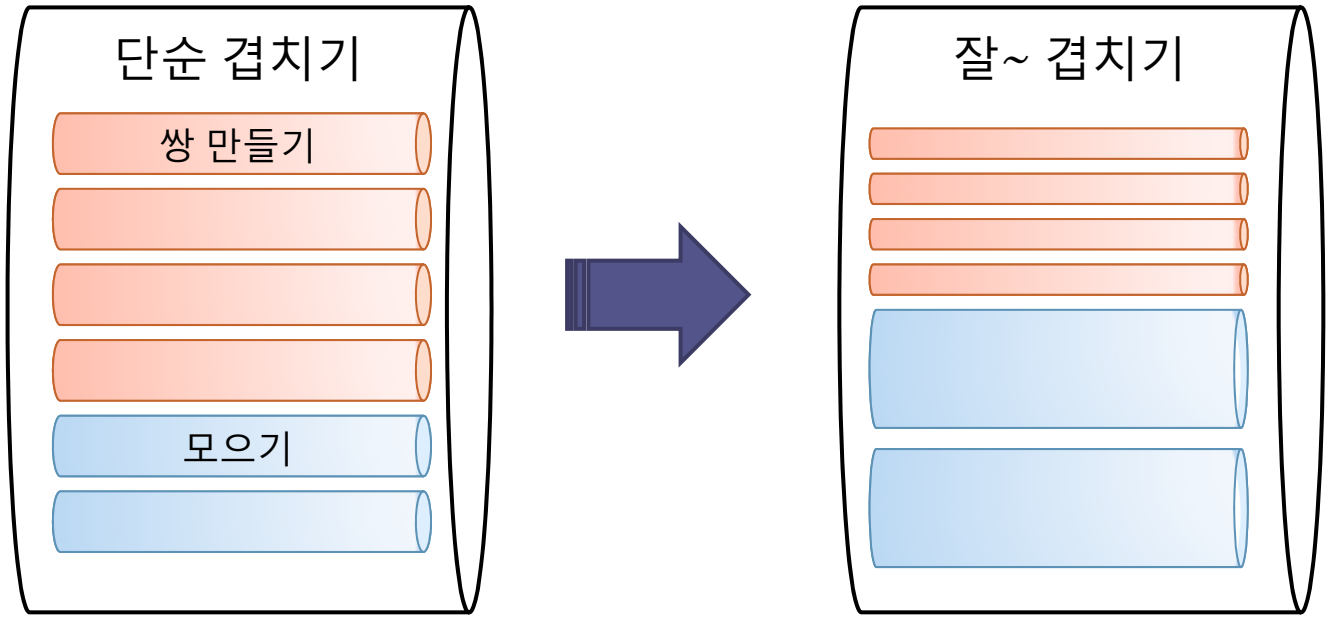
파일 단편화 방지하기

- **단순한 단계 겹치기**는 많은 Thread들이 무분별하게 Disk에 접근하는 것을 막지 못함
- 이 때 **(3) 모으기** 입력파일은 File fragmentation 정도가 심해짐





작업 우선 순위에 따른 Bandwidth 할당



목표

1) 쌍 만들기 작업 수와 무관하게 성능 잘 나오게 하기
2) 단계를 겹칠 때, 디스크를 효율적으로 사용하기
   - **단순히 겹치는 것 이상**으로 성능 향상 가능!
     1) 디스크 접근 횟수 줄이기
     2) 파일 단편화 방지하기
     3) 작업 우선 순위에 따른 Bandwidth 할당

끝