안정해시 설계

System Design Interview
Chapter 5

윤석윤 cid.yoon.dev@gmail.com

수평적 규모 확장성 달성을 위한 Feature

- 이를 통해 우리가 해결하려고 하는것

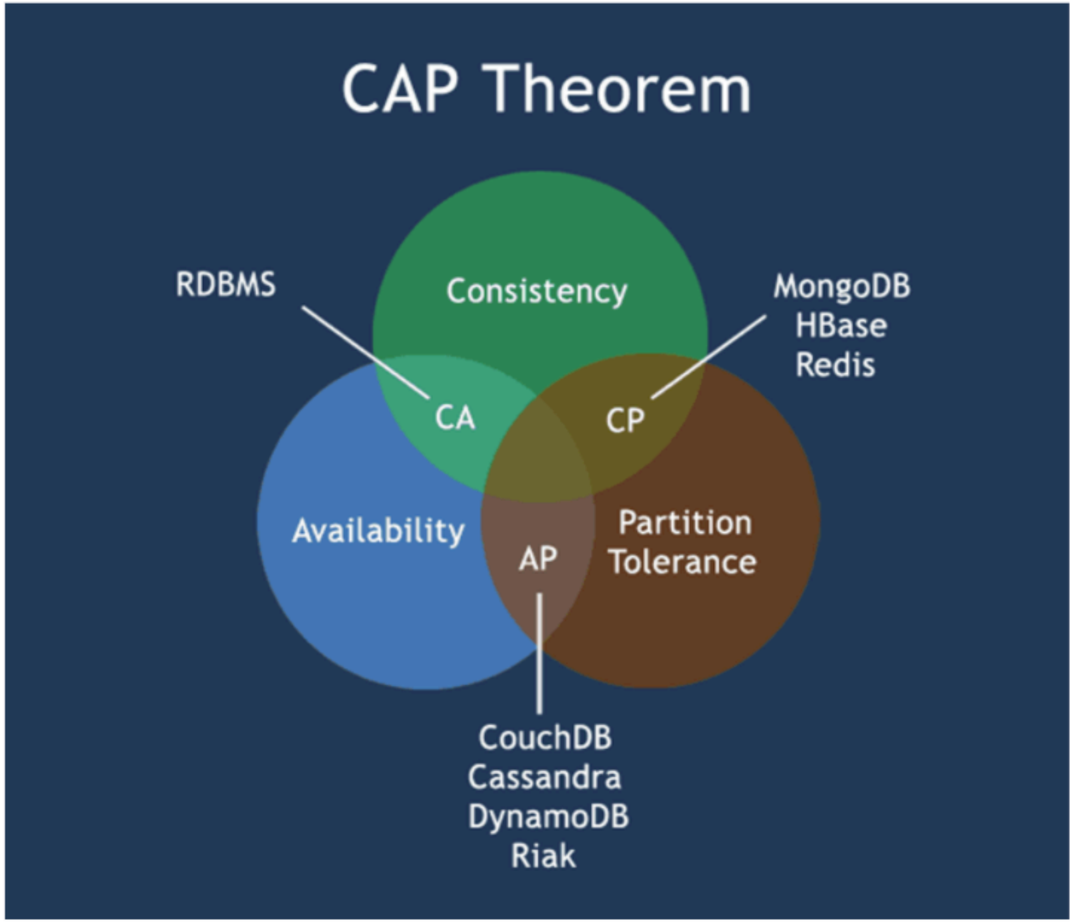
수평적 규모 확장성 달성을 위한 Feature

- 이를 통해 우리가 해결하려고 하는것
  - 요청 또는 데이터를 서버에 균등하게 나누는것
  - 이를 위해 보편적으로 사용하는 기술
  - 그전에 교양

CAP Theorem

http://ndc.vod.nexoncdn.co.kr/NDC2018/slides/NDC2018_0089/index.html

- 2000년, Eric Brewer*
- 튼튼한(robust) 분산 시스템 설계시 기술적인 선택에 도움을 주는 정리
  - 일관성 (Consistency)
  - 가용성 (Availability)
  - 분할용인 (Partition Tolerance)
- C+A+P를 동시에 모두 만족하는 시스템은 없음
  - 2개 선택하세요
- NoSQL과는 사실상 관련 없음



IMG From: https://docs.deistercloud.com/Technology.50/NoSQL/index.xml

샤딩(P)

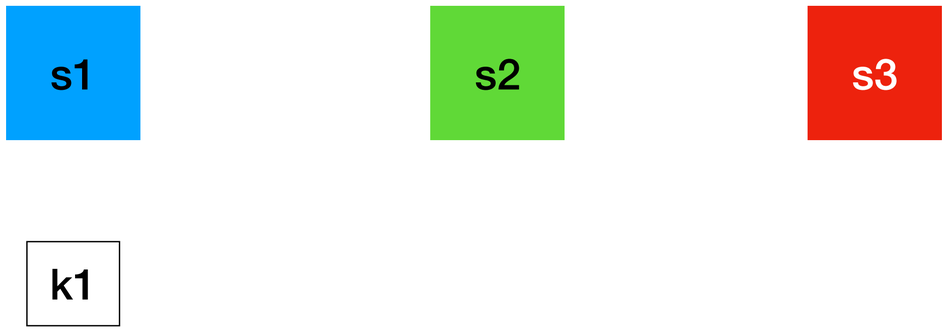
데이터셋이 매우 크거나 질의 처리량이 높다면 복제만으론 부족해요

- 데이터를 파티션으로 쪼갤 필요가 있음
  - 표준 용어는 파티셔닝
  - 대용량 데이터베이스를 의도적으로 작은 단위로 쪼개는 방법
  - 결과적으로 각 파티션은 작은 그 자체로 작은 데이터베이스

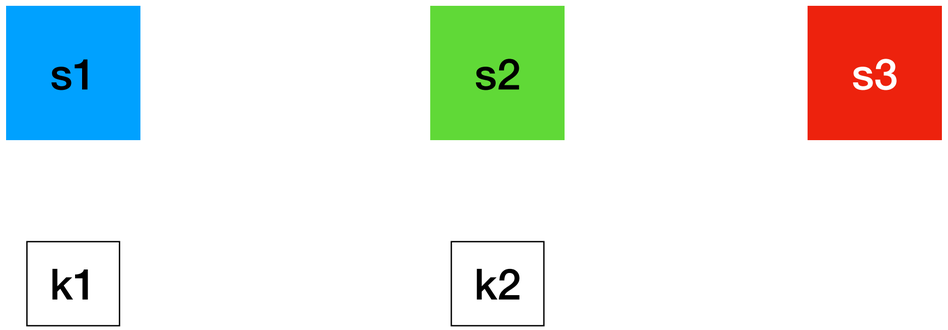
해시 키 재배치

- serverIndex = hash(key) % N
  - N은 서버의 개수
  - 분산을 위한 Modular
  - 사용자는 반드시 키 값에 따라 정확한 서버에 요청을 해야만 원하는 답을 얻을 수 있다

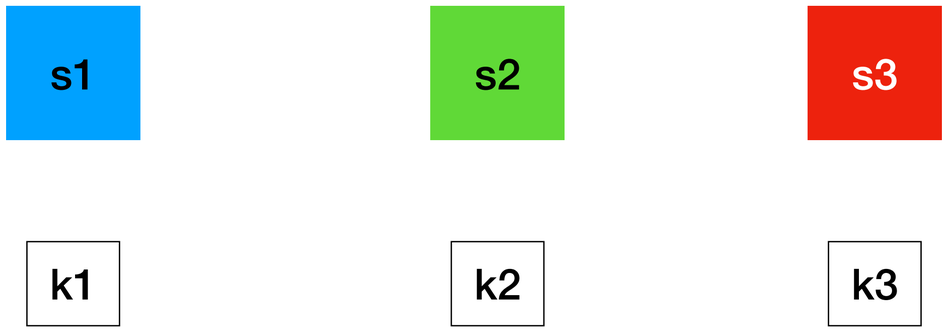
- serverIndex = hash(key) % N



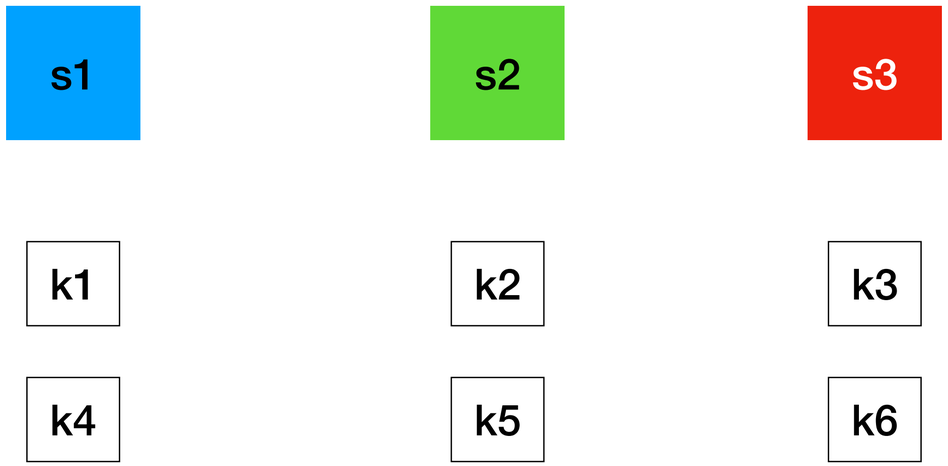
- serverIndex = hash(key) % N



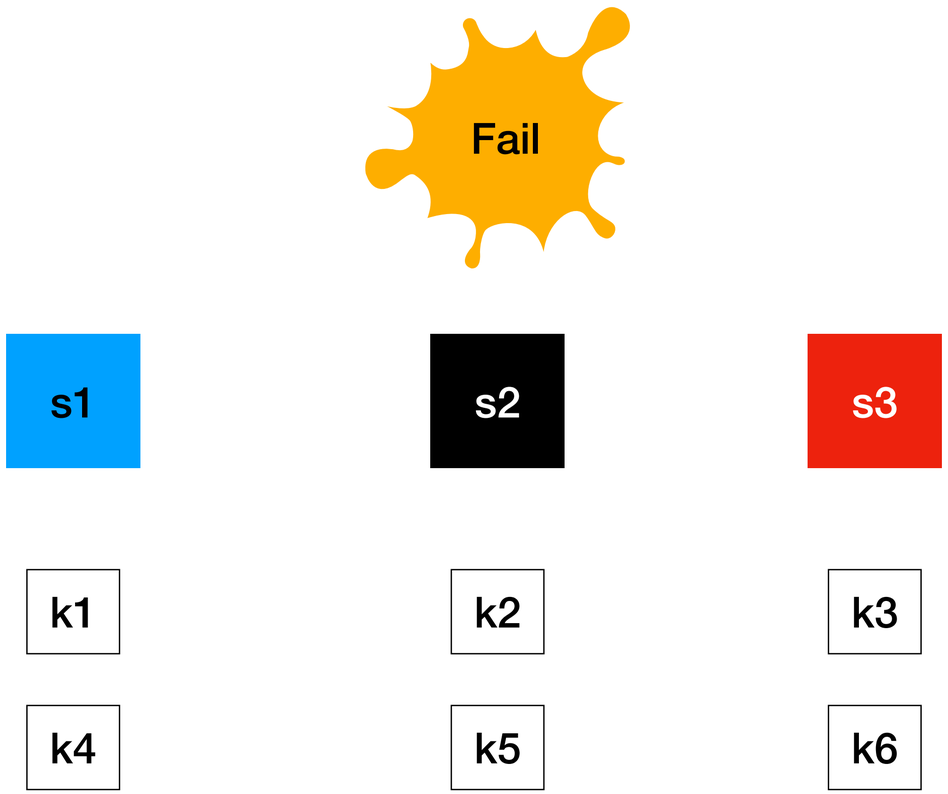
- serverIndex = hash(key) % N



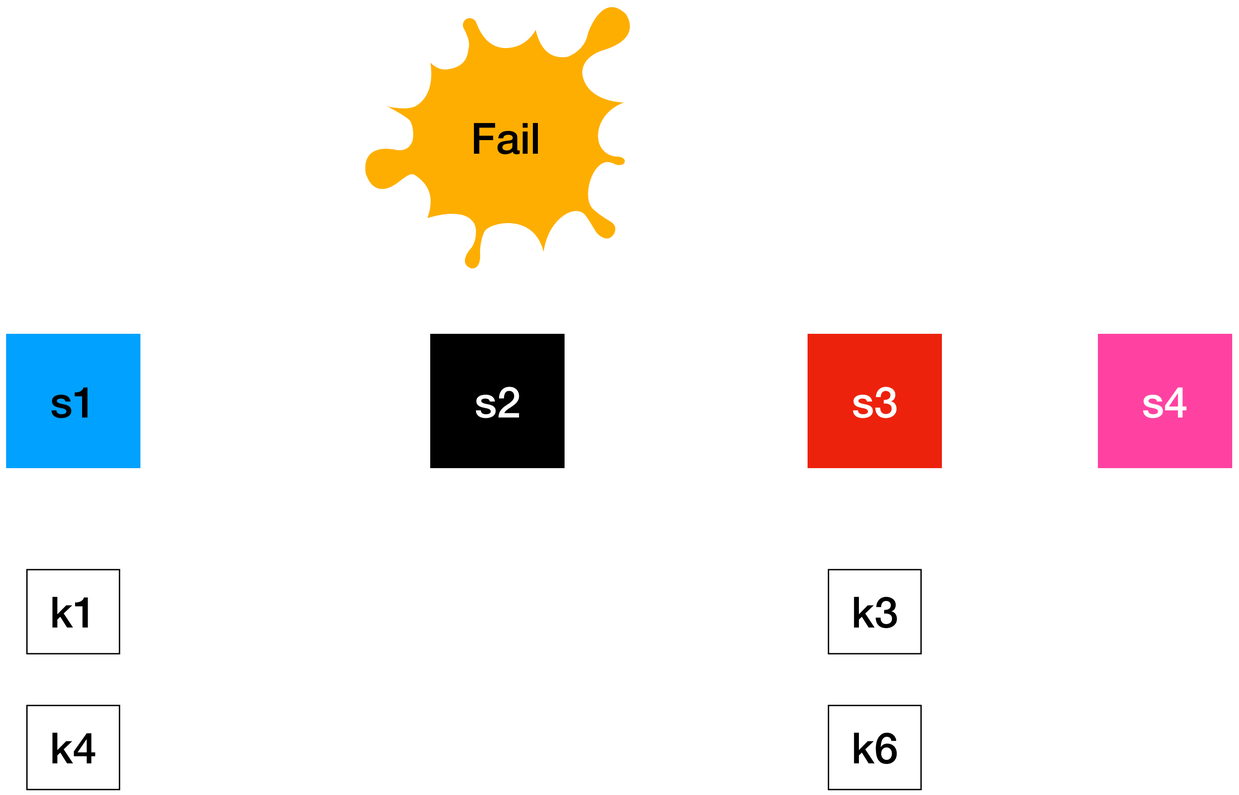
- serverIndex = hash(key) % N



- serverIndex = hash(key) % N



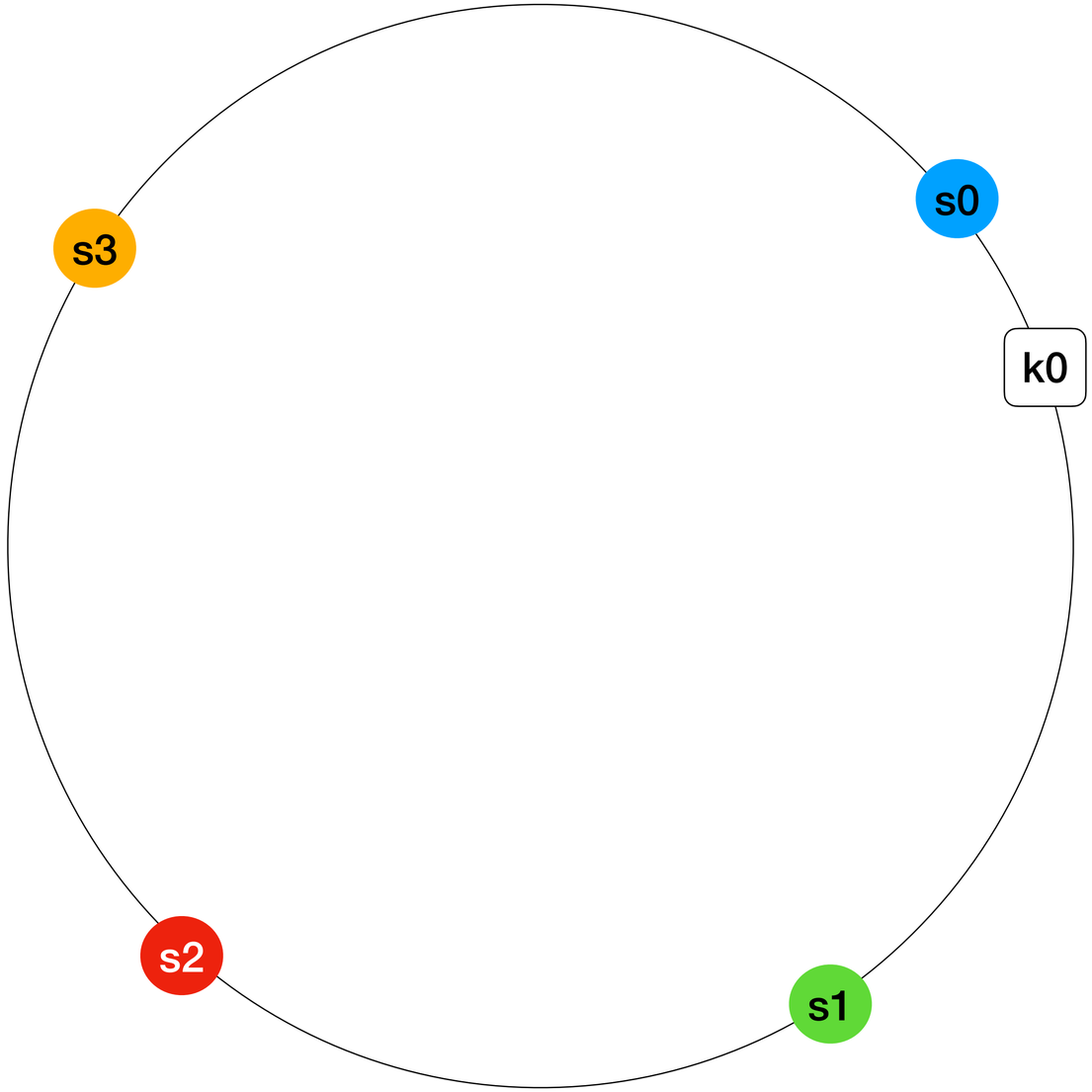
- serverIndex = hash(key) % N

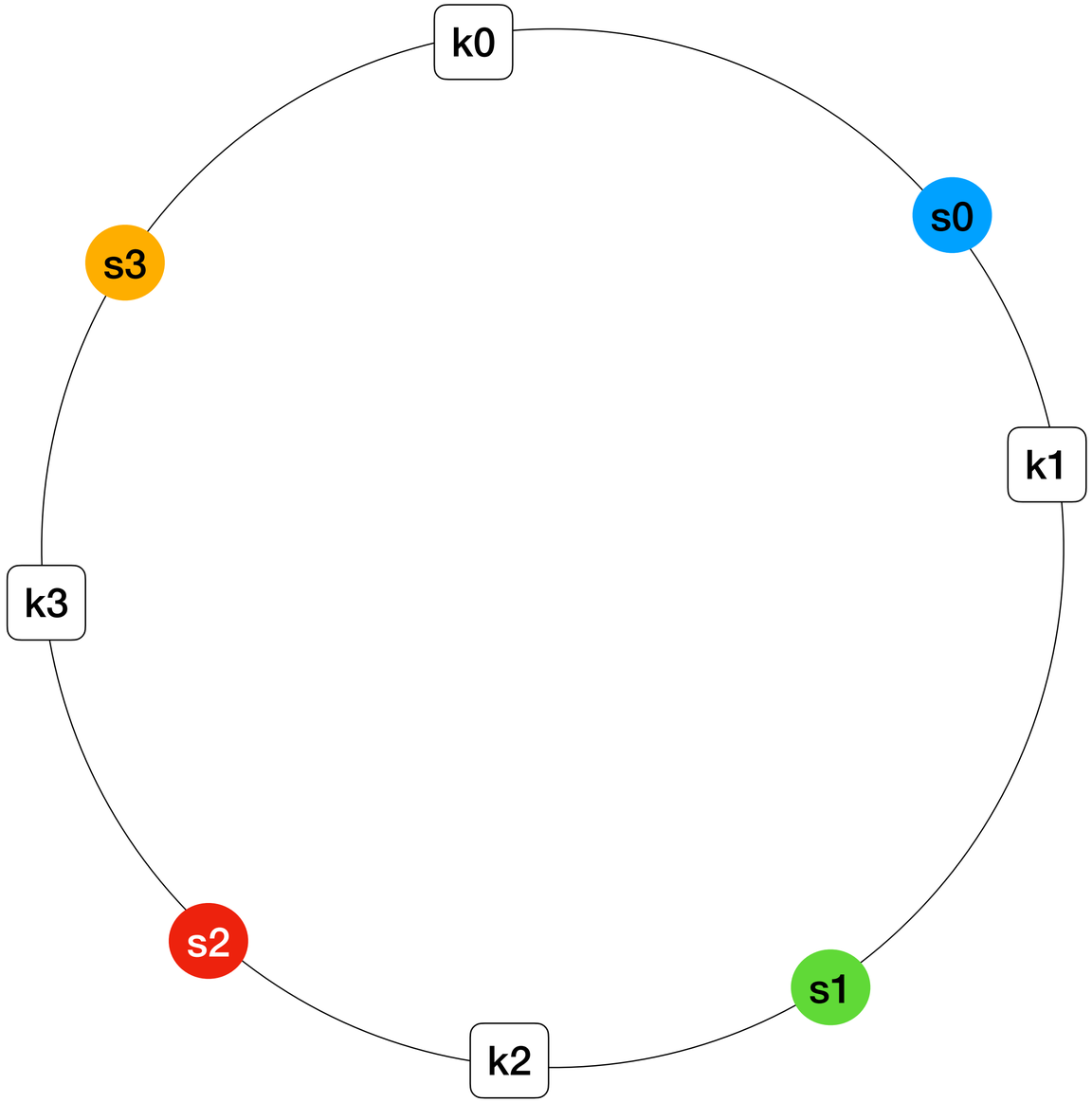


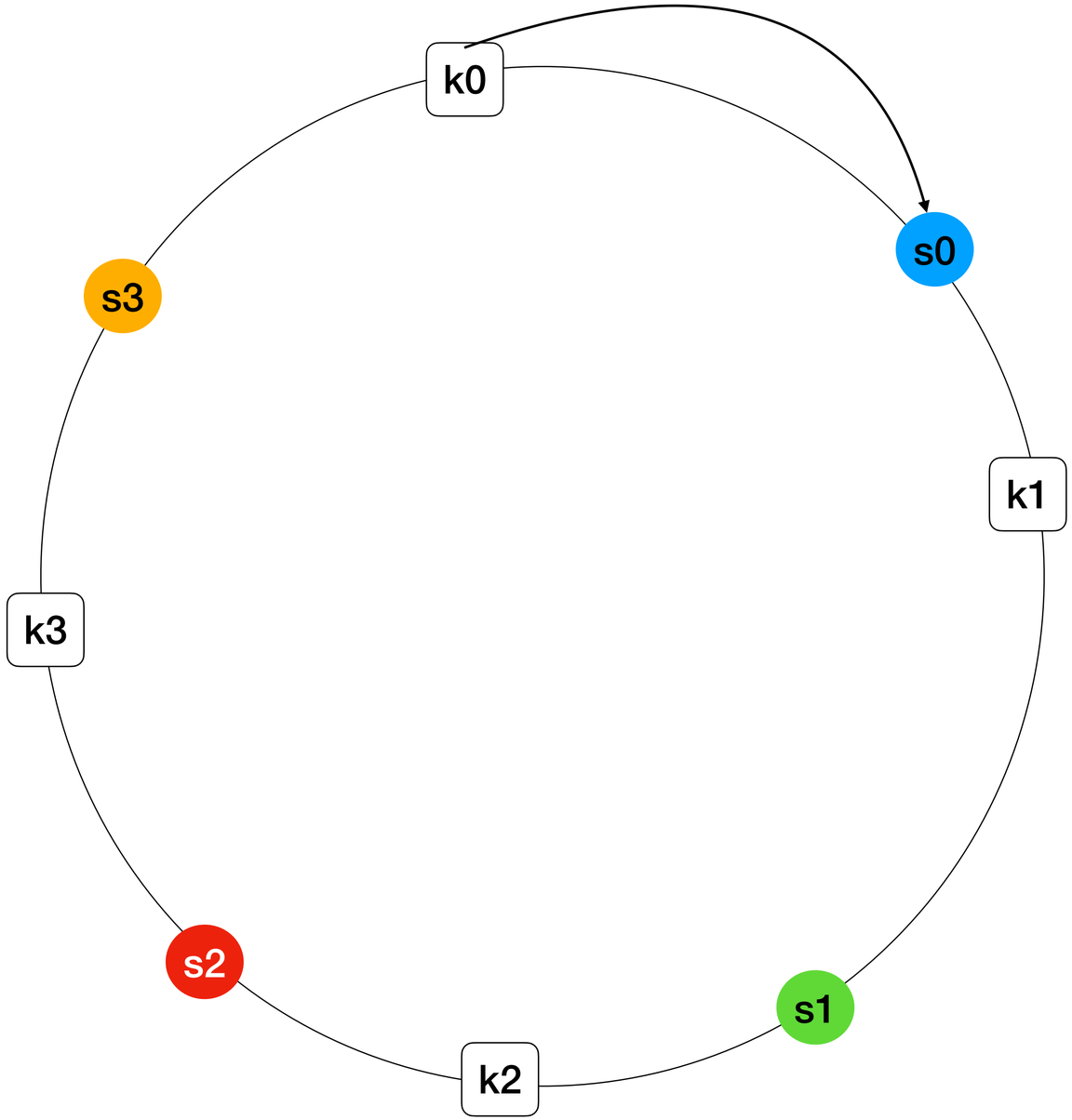
안정 해시

K/N 개의 키만 재배치 하는 해시 기술

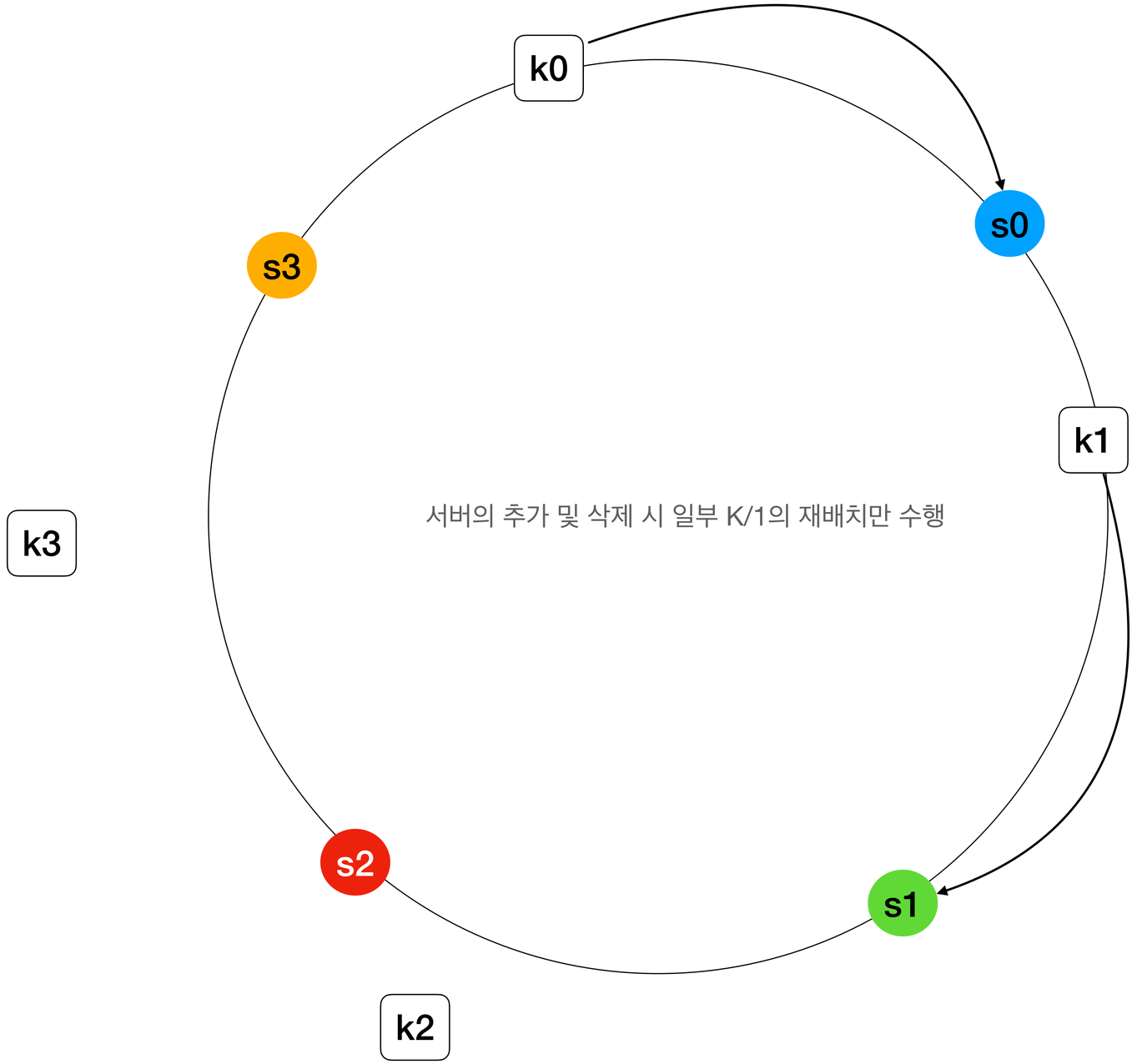
- K = 키의 개수, N은 슬롯 개수
  - 대부분의 전통적 해시 테이블은 슬롯의 수가 바뀌면 거의 대부분 키를 재배치
  - 아~주 넓은 해시를 가질 수 있는 캐시 테이블 생성
  - 방향을 가진 순회를 위해 링 구조로 표현
  - **modular 연산을 사용하지 않고 있음**


s0
k0
s1
s2
s3


k0
s0
k1
s1
k2
s2
k3
s3

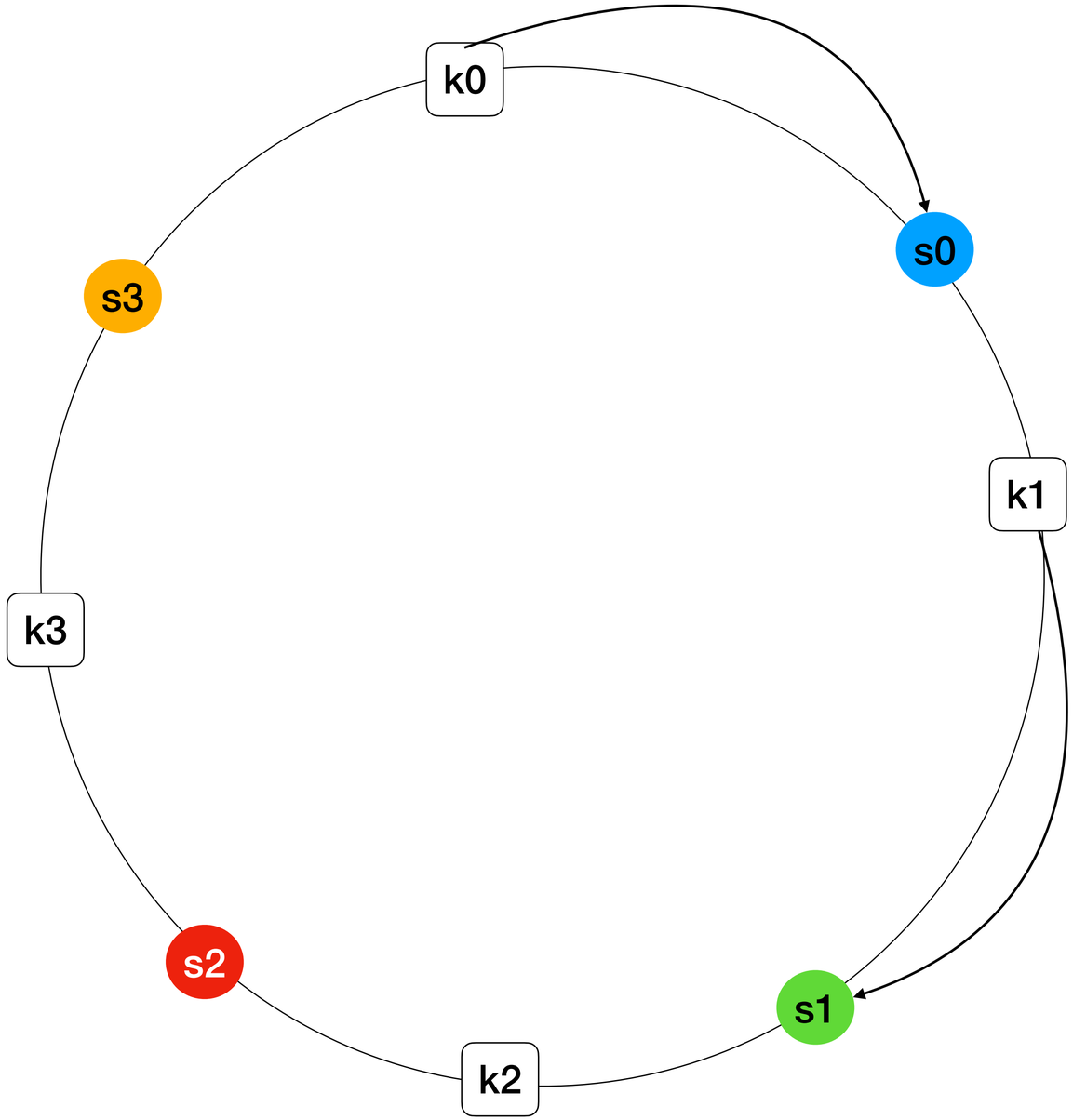


시계방향(혹은 반시계 방향)으로 링을 탐색



시계방향(혹은 반시계 방향)으로 링을 탐색

처음 만나는 서버에 값을 저장



시계방향(혹은 반시계 방향)으로 링을 탐색

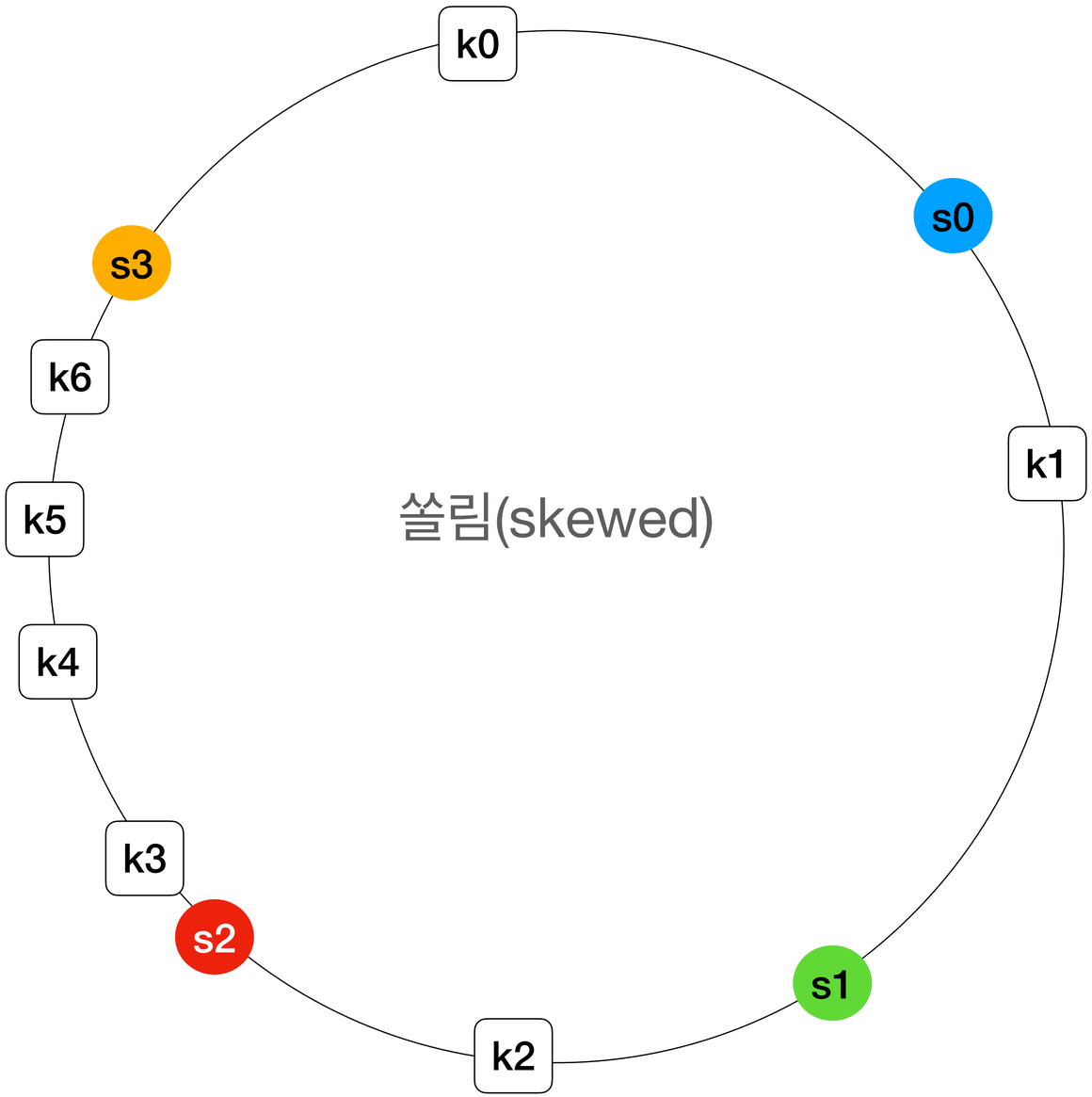
처음 만나는 서버에 값을 저장

안정 해시

기본 구현법의 두가지 문제

- 서버와 키를 균등 분포(uniform distribution) 해시 함수를 사용해 해시 링에 배치
- 키의 위치에서 링을 검색 방향으로 탐색하다가 만나는 최초의 서버가 키가 저장될 서버

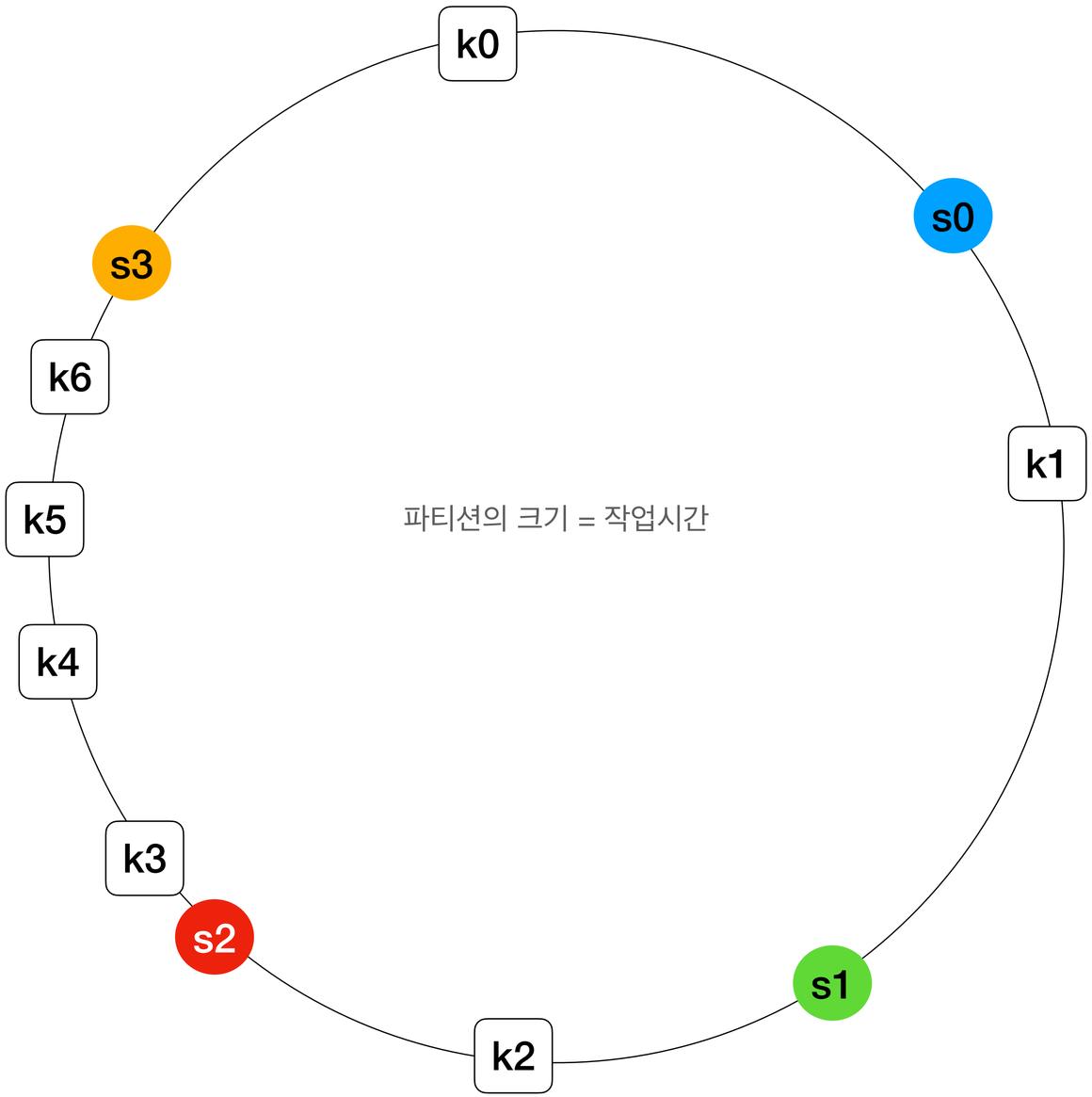
- 문제
  - 추가 및 삭제 시 파티션의 크기를 균등하게 유지하는게 불가
  - 키의 균등 분포를 달성하기 힘듦

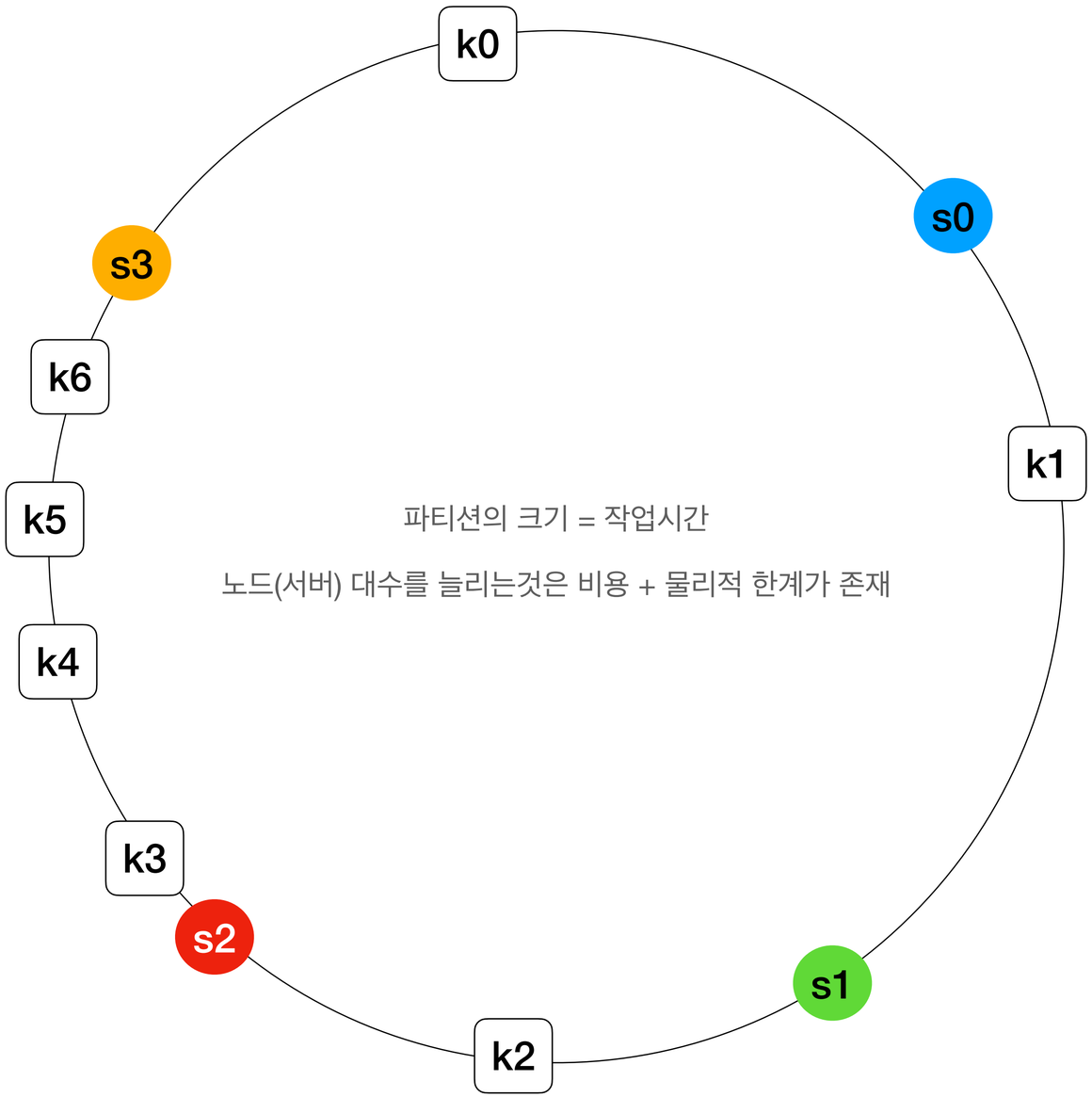

k0
s0
k1
k2
s1
s2
k3
k4
k5
k6
s3
쏠림(skewed)

쏠림(skewed)의 최소화
파티셔닝의 목적

- 핫스팟(불균형적으로 높은 부하를 받는 노드)
  - 데이터의 질의 및 저장에 발생하는 부하를 여러 장비에 균일하게 분배하는 것
  - 분산될 노드가 적은 경우 쏠림 및 부하 분산에 어려움이 발생

- 예시
  - 유명 연예인을 팔로우 하는 경우
  - 유사 시간때에 특정 데이터를 조회/저장 하는 경우


k0
s0
k1
k2
s1
s2
k3
k4
k5
k6
s3
파티션의 크기 = 작업시간


k0
s0
k1
k2
s1
s2
k3
k4
k5
k6
s3
파티션의 크기 = 작업시간
노드(서버) 대수를 늘리는것은 비용 + 물리적 한계가 존재

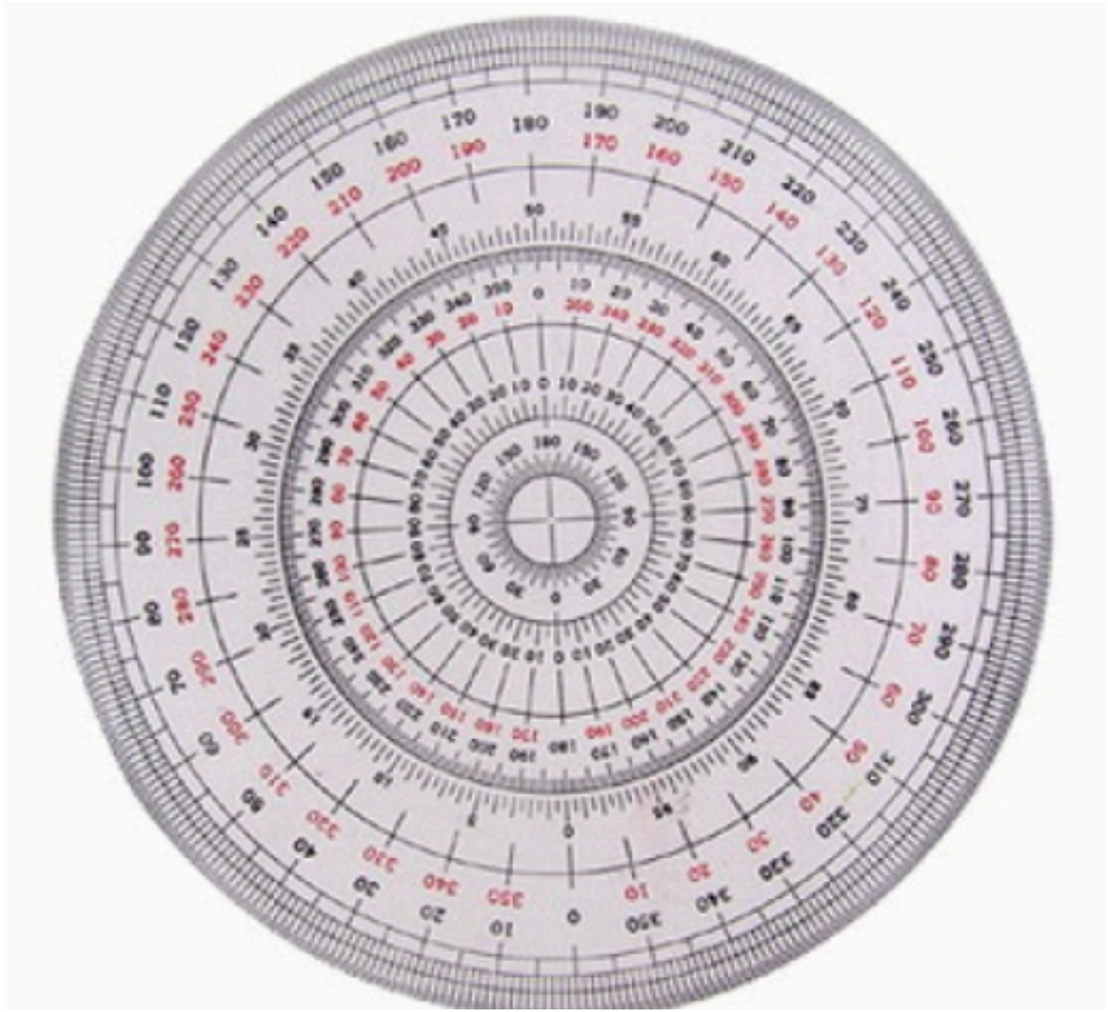
안정 해시

가상 노드(virtual node)

- 실제 노드 또는 서버를 가리키는 노드
  - 서버는 링 위에 여러 개의 가상 노드를 가질 수 있다
  - 가상 노드의 개수를 늘리면 키의 분포는 점점 균등 해진다(표준 편차)

안정 해시

가상 노드(virtual node)



안정 해시

가상 노드(virtual node)

- 각도기의 표시처럼 정렬된 값의 가상 노드 목록을 생성
  - 가상노드=파티션
  - 파티션을 노드 대수보다 많이 만들고 각 노드에 여러 파티션을 할당하는 것

- 저장 및 검색
  - 정렬된 서버 값 목록을 탐색(또는 이진 검색)하여 더 큰 첫번째 서버를 찾기
  - 해당 값을 찾지 못하는 경우 목록의 첫번째 값에 저장

Server

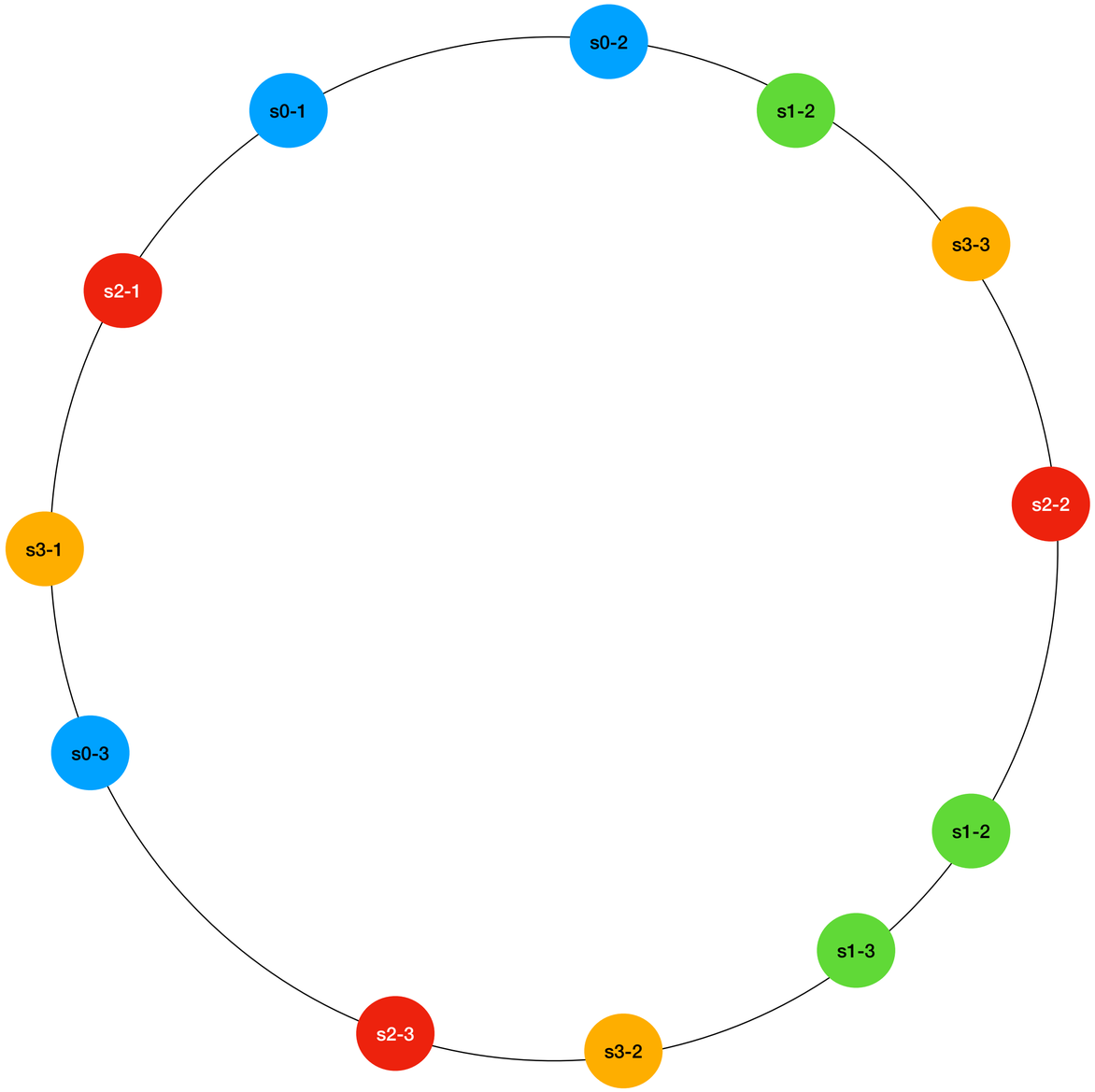
s0 s1

s2 s3

Key

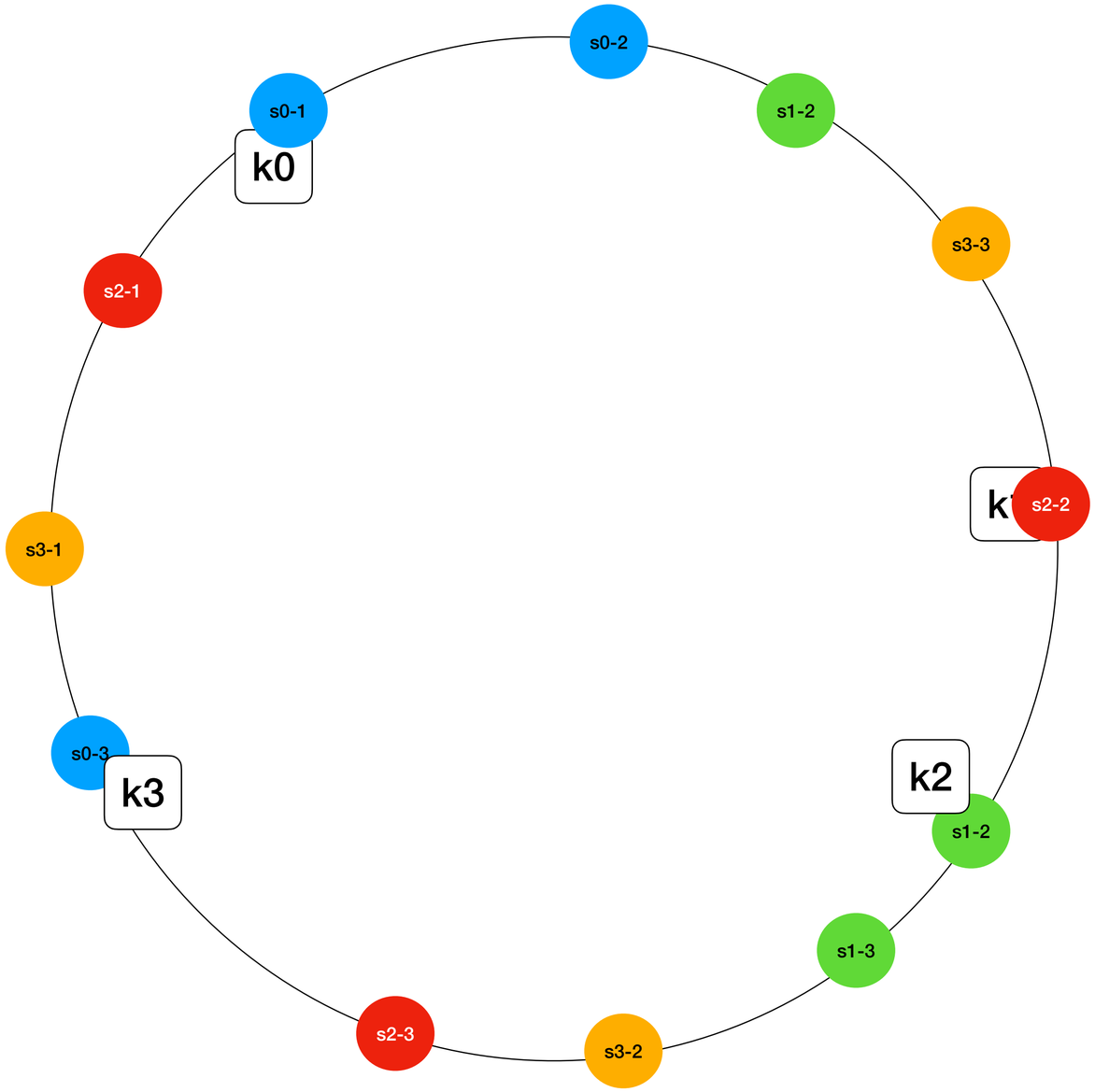
k0 k1

k2 k3



Server

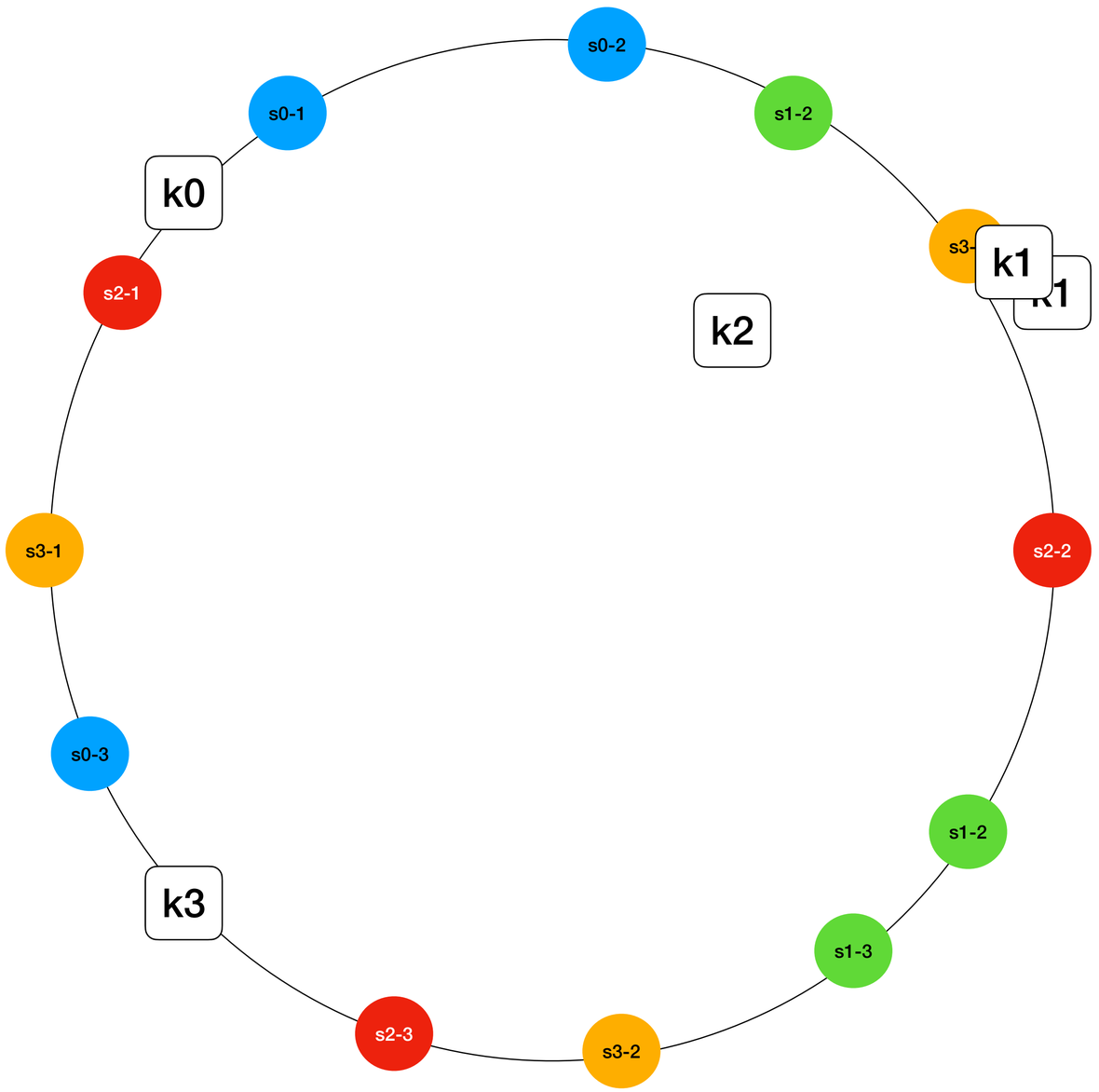

s0
s1
s2
s3


s0-2
s0-1
k0
s1-2
s3-3
s2-1
s2-2
s3-1
s0-3
k3
k2
s1-2
s1-3
s2-3
s3-2

Server

s0 s1

s2 s3

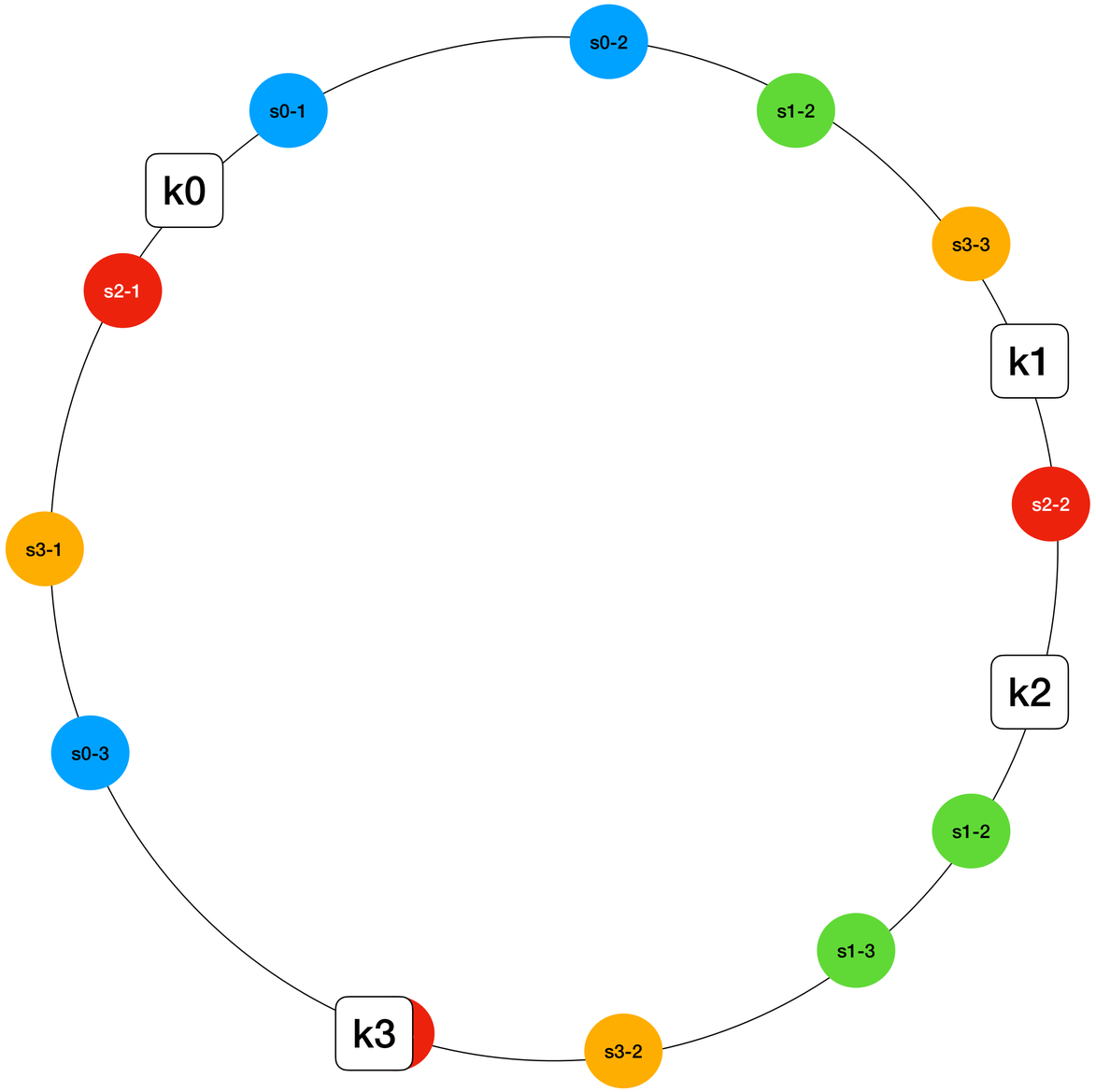


Key	Hash	Slot
s0-1	2222	1
s0-2	3333	2
s1-2	4444	3
s3-3	5555	4
s2-2	6666	5

Server

s0 s1

s2 s3



Key	Hash	Slot
s0-1	2222	1
s0-2	3333	2
s1-2	4444	3
s3-3	5555	4
s2-2	6666	5

Key	Hash	Slot	Serv
K0	2012	0-1	S0
K1	5888	2-2	S2
K2	6688	1-2	S1
k3	10101	0-3	S0

안정 해시

리밸런싱 과정

- 노드의 추가 및 삭제
  - 새 노드는 파티션이 다시 균일하게 분배될 때까지 기존 노드에서 파티션 몇개를 가져온다. 제거되는 경우는 이와 반대의 현상이 발생

- 파티션은 노드 사이에서 통째로 이동
  - 개수는 바뀌지 않고 할당된 키도 변경되지 않음
  - 유일한 변화는 노드에 어떤 파티션이 할당되는가? 이다

안정 해시
기타

- 운영 용이성
  - 파티션 개수가 고정되면 운영이 단순해짐
  - 고정 파티션을 사용하는 데이터베이스는 분할 지원을 하지 않는 경우가 많음
- 고려할 부분
  - **가상 노드는 데이터를 저장할 공간을 더 많이 필요하기에** 적절한 trade off가 필요
  - 초기 설정된 파티션 수가 사용 가능한 노드 대수의 최대치가 되기에 미래에 증가할 것을 수용하기 위해 충분히 큰 값으로 선택해야 함
  - 고비용 리소스를 사용하는 경우 노드 할당을 더 많이 해 주는 것도 가능

CAP 보다는 PACECL

P는 필수

- 분할 내성? 보다는 분할 용인

  The system continues to operate despite arbitrary message loss or failure of part of the system

  The network will be allowed to lose arbitrarily many messages sent from one node to another

  - http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
  - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.67.6951&rep=rep1&type=pdf

- 마치 P를 포기하고 CA를 선택할 수 있는가?
  - 절대로 장애가 나지 않는 네트워크를 구성하지 않는 이상 불가능
  - P는 선택이 아닌 필수, C와 A가 비대칭

PAECL

> *if there is a partition (P) how does the system tradeoff between availability and consistency (A and C); else (E) when the system is running as normal in the absence of partitions, how does the system tradeoff between latency (L) and consistency (C)?*