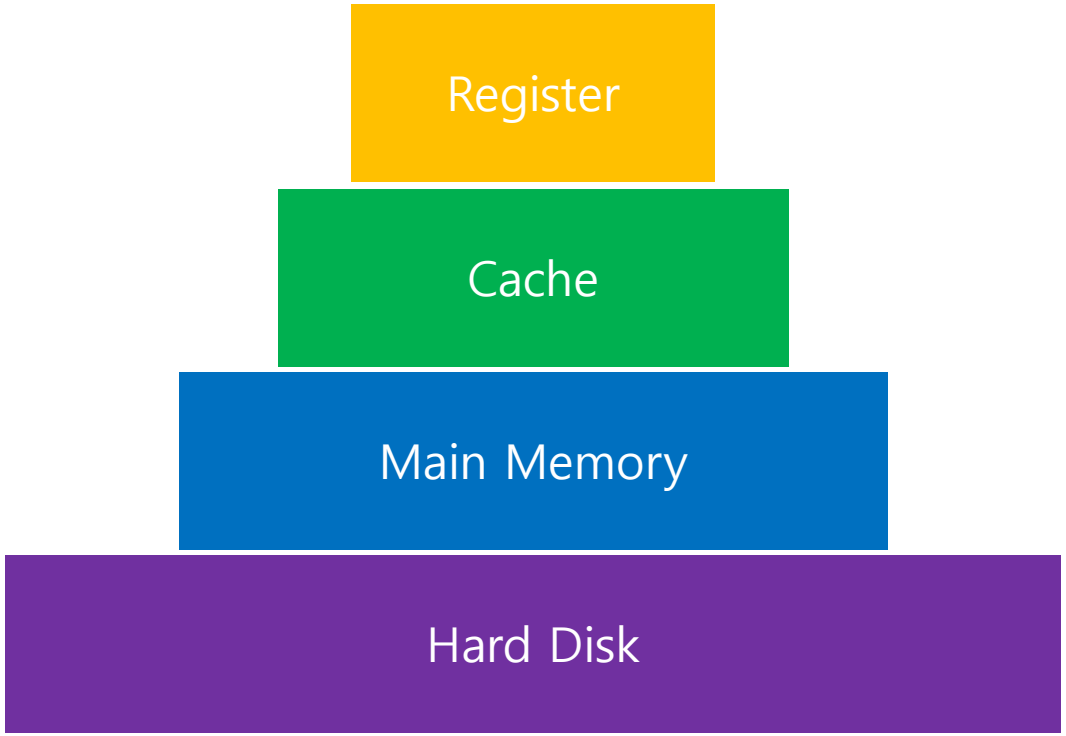
B-tree

Why B-tree??

균형 이진 트리는 완벽하지 않은가?
- 탐색에서 최악의 경우에도 O(log2n)이 걸리는
  균형 이진 트리도 굉장히 큰 성과
- 하지만 데이터가 메인 메모리에 저장되어 있거나
- 데이터베이스의 경우 하드디스크에 저장되어 있으면
- 데이터의 연산보다는 메모리 접근에 시간이 엄청나게 소요된다
- 이 문제를 해결하기 위해 나온 자료구조가 바로 B-tree

메모리 계층 구조



1 cycle

3 cycles

20-100 cycles

500,000-5,000,000 cycles

데이터를 가져올 때

메인 메모리에서 캐시로 가져올 때 **64 bytes~ 128 bytes**

캐시 라인

블록 단위로 가져온다!!!

하드 디스크에서 메인 메모리로 가져올 때 **4096 bytes**

페이지 프레임

균형 블랙 트리의 높이

균형 이진 트리의 높이 : log2(n+1)

데이터의 개수가 1,000,000개면

최대 높이가 20이므로

최악의 경우 20번의 메모리 접근이 필요

메모리 접근 횟수를 줄이자!

균형 이진 트리의 이점을 가지면서
메모리 접근 횟수를 줄일 수 없을까?

트리의 높이를 줄이면 비교 연산 횟수는 늘지만
메모리 접근 횟수는 줄일 수 있다.

M-원 탐색 트리(m-way search tree)

1. 최대 m개의 서브 트리를 가진다.
2. 한 노드에서 key는 정렬되어 있다.
3. 한 원소의 왼쪽 서브 트리의 모든 key는 원소의 key보다 작다.
4. 한 원소의 오른쪽 서브 트리의 모든 key는 원소의 key보다 크다.
5. 서브 트리도 m-원 탐색 트리이다.

B-tree

차수가 m인 B-tree의 정의

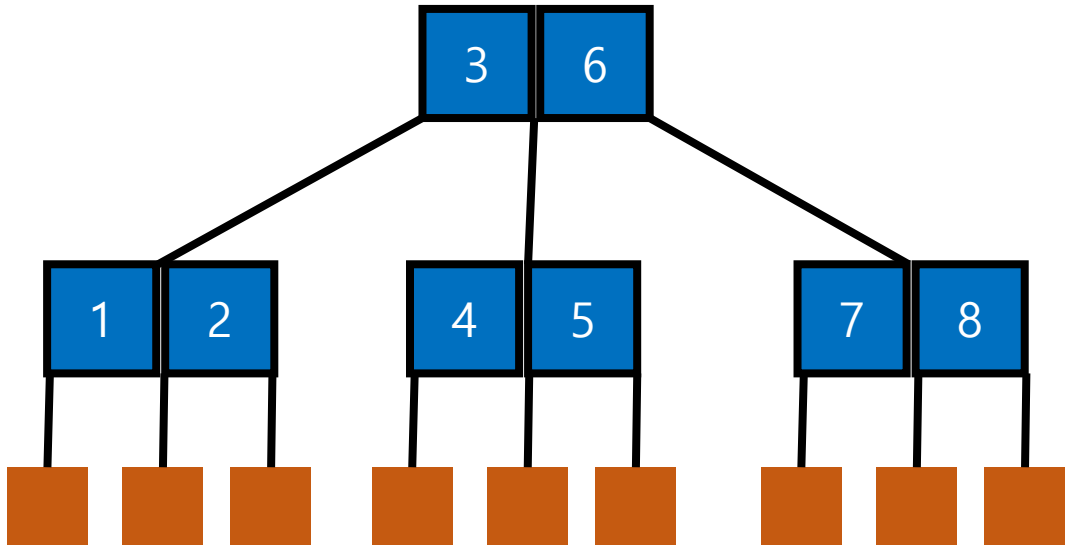
1. 2<= 루트 노드의 서브 트리 <= m
2. $\lceil m/2 \rceil$ <= 루트, 외부 노드를 제외한 모든 노드의 서브 트리 <= m
3. 모든 외부 노드는 같은 레벨이다.

2-3 트리(2-3 tree)

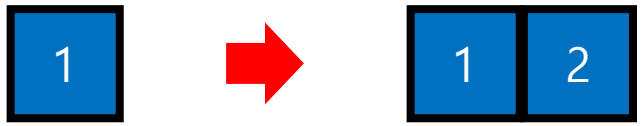
m=3일 때, 2-3 tree라고도 한다.

[3/2] <= 서브 트리 수 <= 3

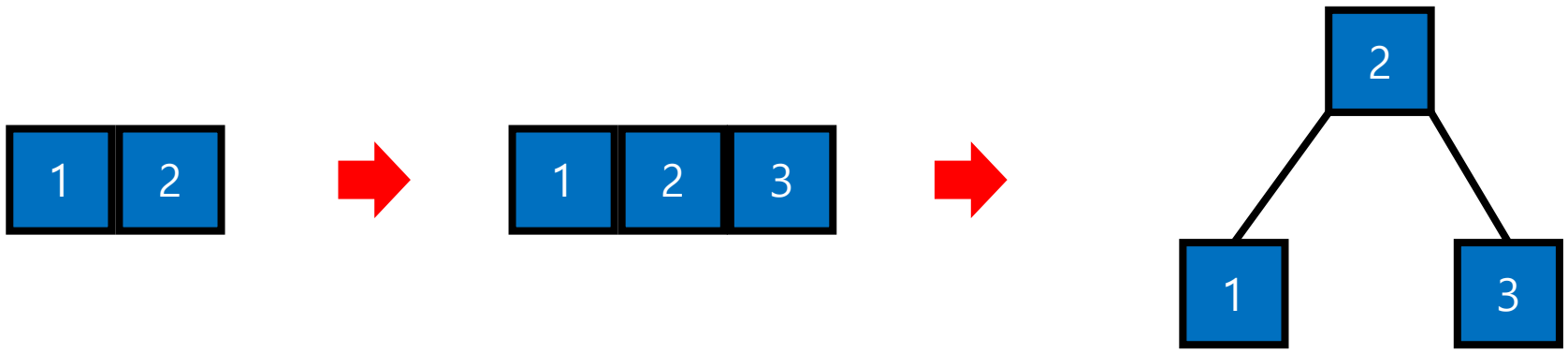
1 <= 노드 원소 수 <= 2



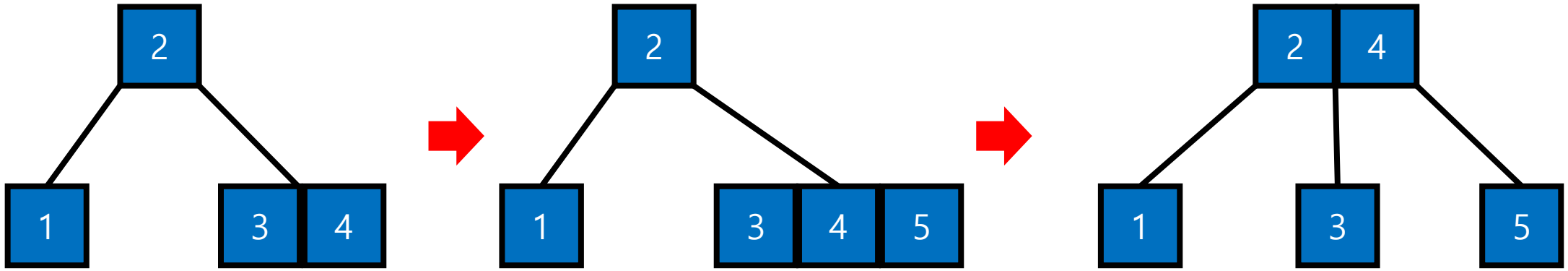
Insert 예제



Insert - split

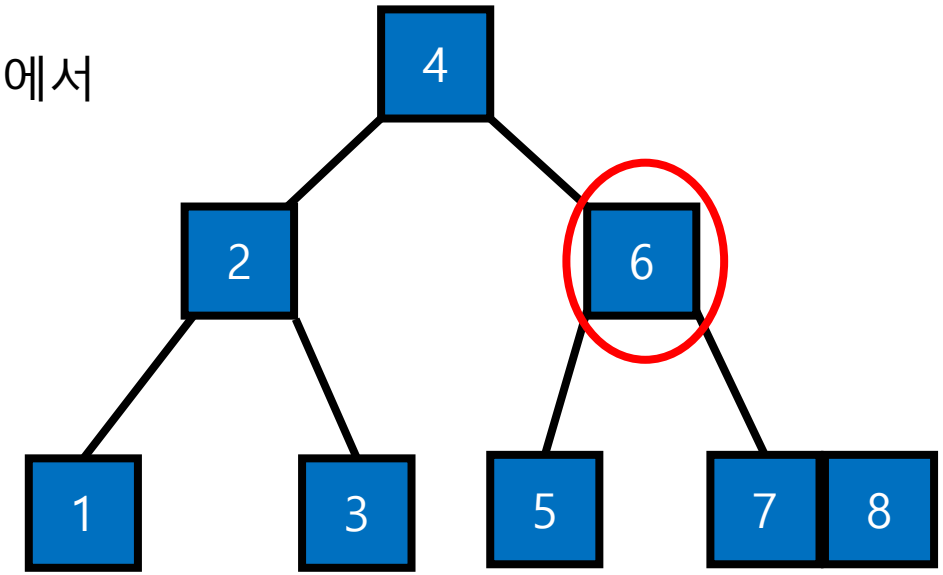


Insert - split

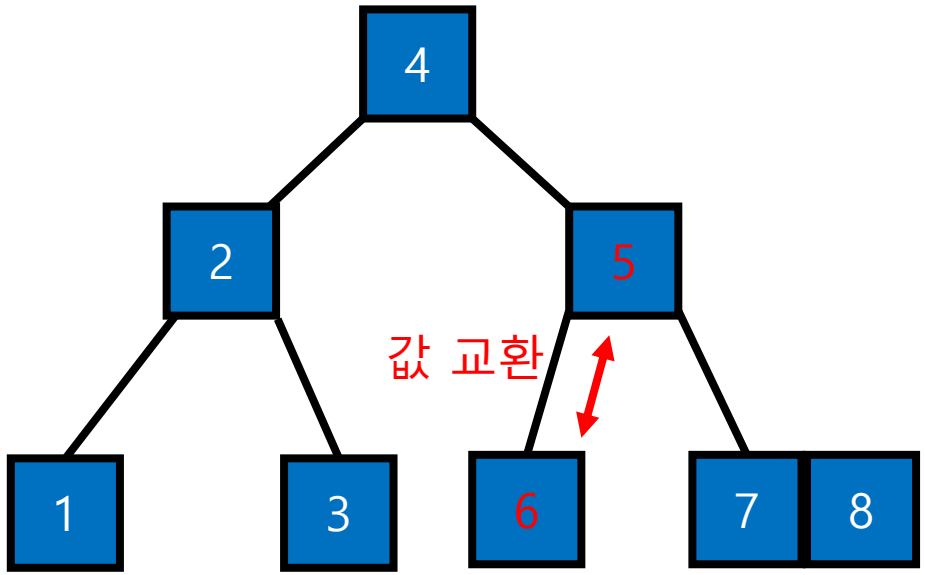


Delete -donate

삭제는 항상 리프 노드에서 이루어진다!
-> 대체 노드를 찾는다!
대체 노드는 왼쪽 서브 트리에서
가장 큰 노드!

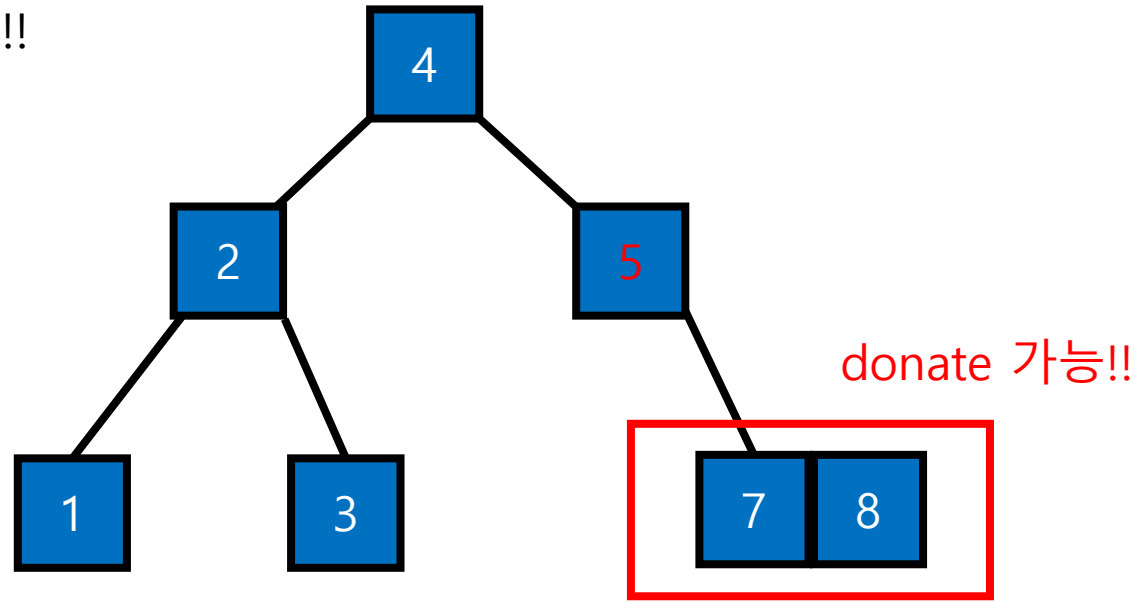


Delete - donate

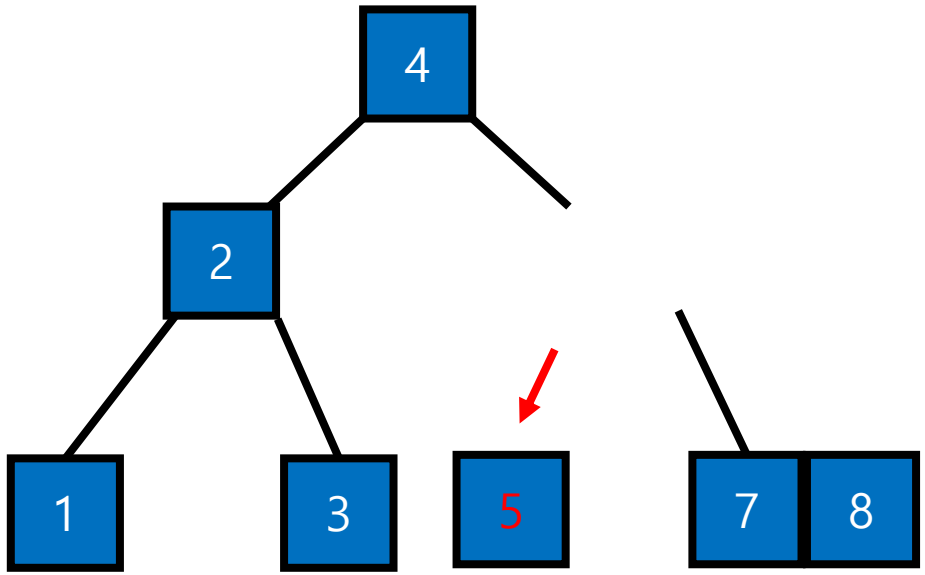


Delete – donate 1

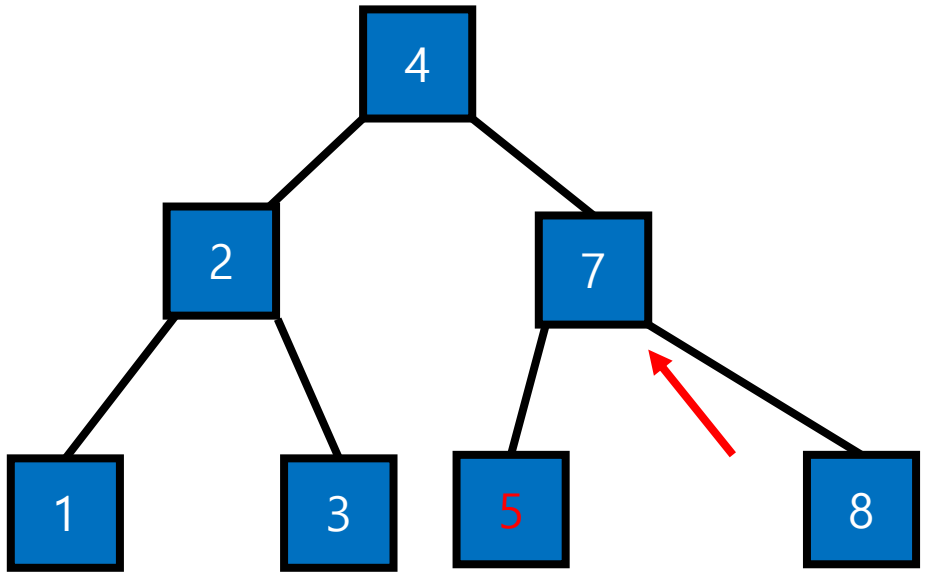
모든 외부 노드는
같은 레벨이다는 규칙에 위반!!
→ Donate 시도!!



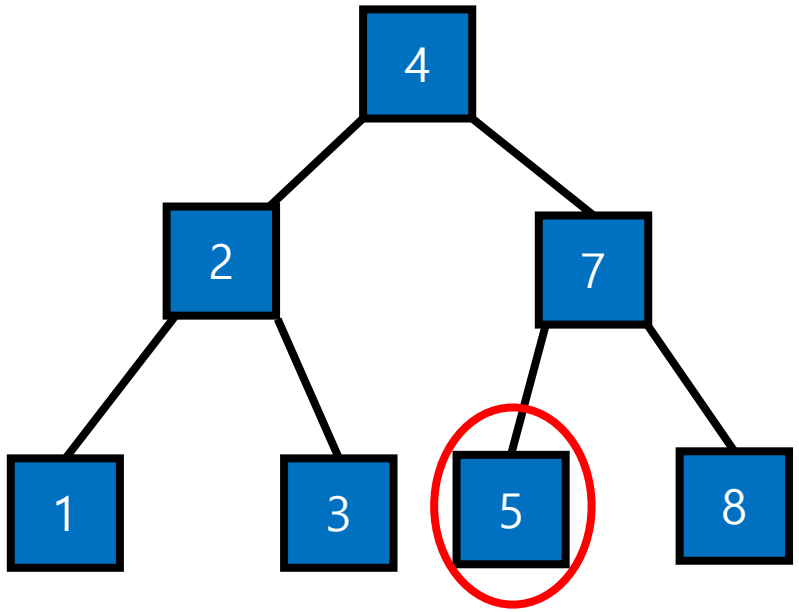
Delete – donate 2



Delete – donate 3



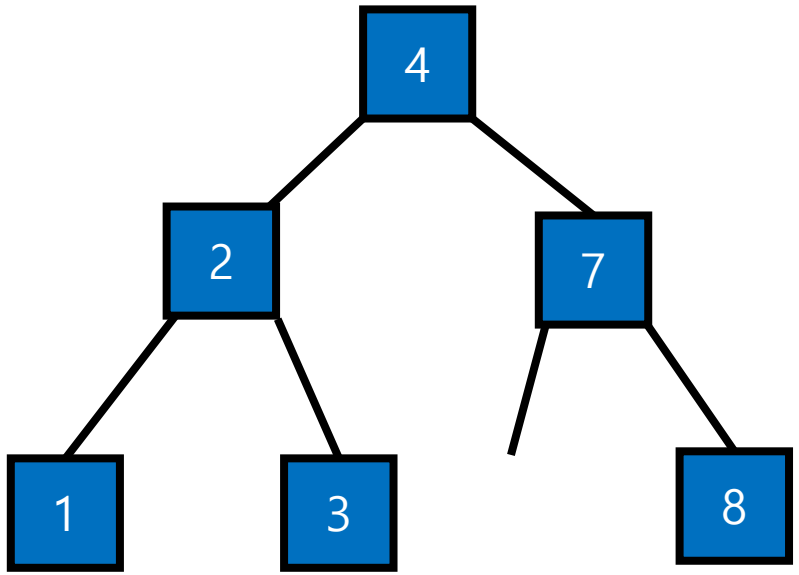
Delete – merge



Delete – merge

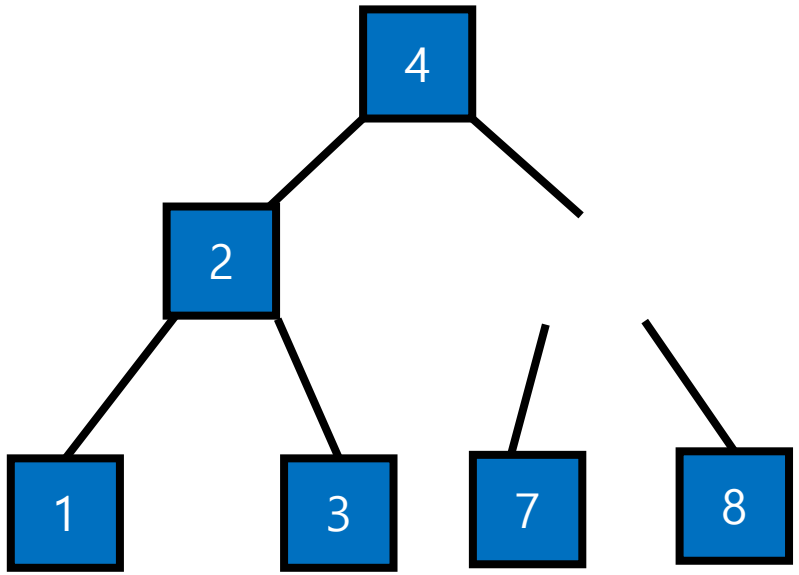
Donate 시도!

- 오른쪽 노드도 원소가 하나뿐이므로 불가능!



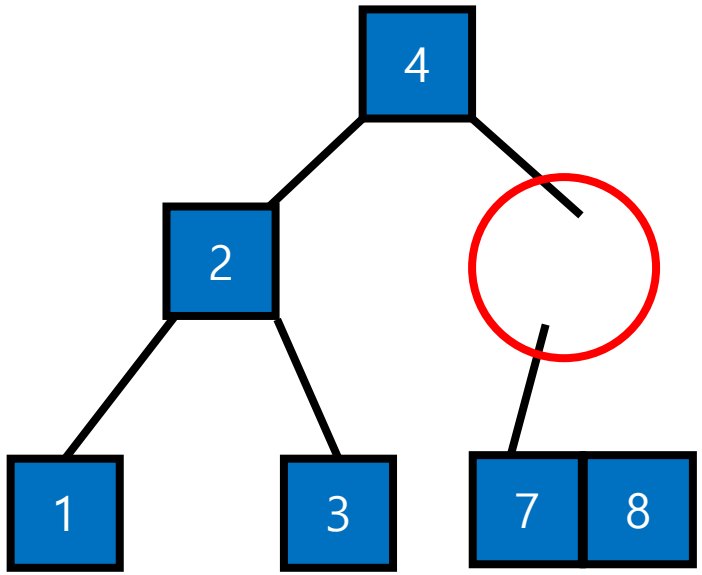
Delete – merge 1

부모의 원소가 내려와
왼쪽 노드에 붙는다.



Delete – merge 2

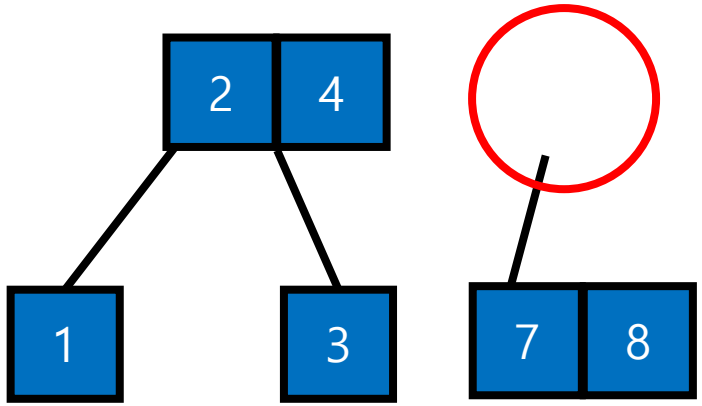
오른쪽 노드가 붙는다.



최소 원소 수를 채우지 못하므로
다시 donate 시도!
왼쪽 형제 노드도 원소가 2 하나뿐이므로
불가능!

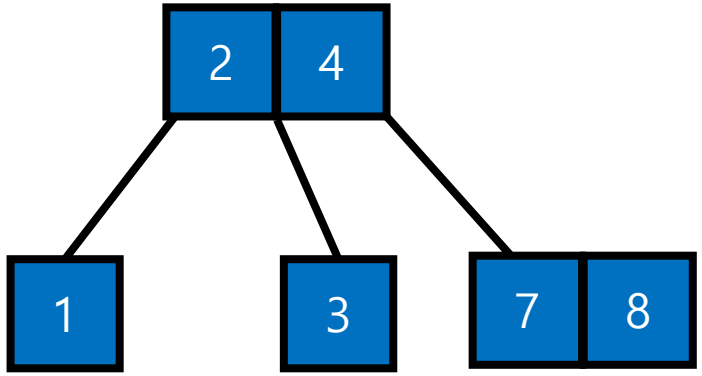
Delete – merge 1

부모의 원소가 내려와
왼쪽 노드에 붙는다.



Delete – merge 2

오른쪽 노드가 붙는다.



B-tree의 응용
데이터베이스

```
CREATE INDEX idx_lastname
ON Employees(LastName);
```

PRIMARY KEY 아님

인덱스 생성 SQL을 실행하면
해당 컬럼을 key로 B-tree를 생성한다.

B-tree의 응용
데이터베이스

```
SELECT * FROM Employees
WHERE LastName='john'
```

인덱스 생성 후 위와 같은 SQL을 실행하면
탐색을 B-tree에서 수행하기 때문에 매우 빠르다!!
→ 이진 탐색을 수행하므로
인덱스를 만들어 두지 않았다면 선형 탐색을 수행한다