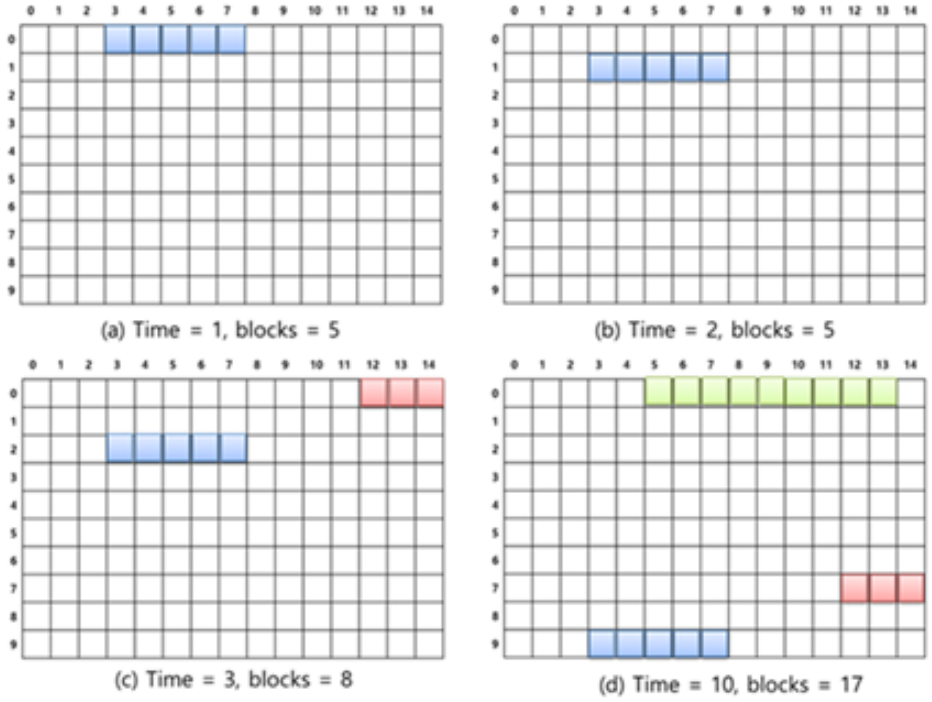
[22.12.02] 블록제거게임

TS블록제거게임

김 태 현

문 제

- R * C 크기의 격자판이 주어진다.
- 블록은 가로방향으로 격자판 가장 윗부분에 놓아지며 시간이 1 지날 때마다 모든 블록이 한 칸씩 밑으로 내려온다.
  더 이상 내려갈 수 없는 경우, 제거되어 없어진다.
- 추가적으로 블록을 제거하는 기능이 있다.
  이 기능은 각 열 마다 바닥에서 가장 가까운 블록 하나를 제거한다.



dropBlocks() <= 1,000
removeBlocks() <= 400

void init(int R, int C)

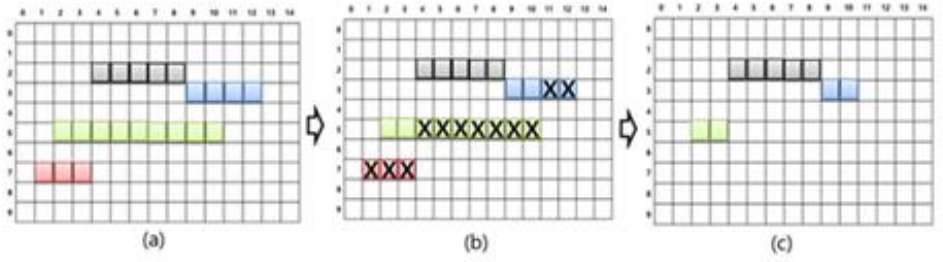
R*C 크기의 격자판이 주어진다.
10 <= R <= 50
10 <= C <= 10,000

int dropBlocks(int timestamp, int col, int len)

timestamp 시각에 col 열에 길이 len인 블록이 내려온다.
남아있는 블록의 총 개수를 반환한다.

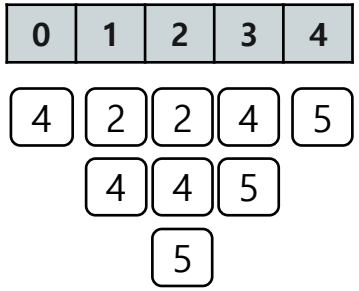
int removeBlocks(int timestamp)

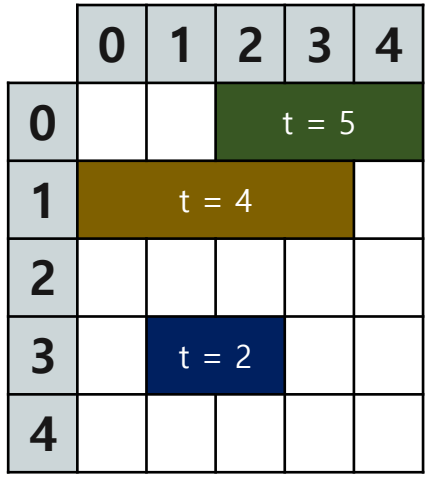
timestamp 시각에 모든열의 가장 아래위치한 블록을 제거하고
남아있는 블록의 총 개수를 반환한다.



블록 관리

1. 열 단위 낱개로 관리





2. 연속된 블록 한 개의 노드로 관리

노드 : {time, col_start, col_end} or {time, col, len}

2, 1, 2 | 4, 0, 3 | 5, 2, 4

노드를 관리하는 Container
set? list? vector?

=> 대개 이런 문제의 경우 노드 단위로 관리

블록 관리

필요한 로직

1) 가장 먼저 들어온 노드들부터 바닥을 지나면 삭제 : *시간순 관리, 노드 삭제*
2) 모든 열의 바닥에 가장 가까운 블록 삭제 : *시간순 관리, 노드 분리/삭제*

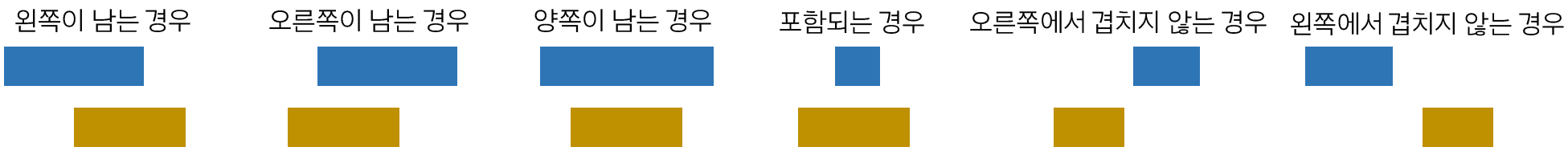
시간은 증가하는 방향으로 주어지므로
블록이 들어오는 순서대로 관리해주면 시간순 관리 가능
: list, vector, (set)

중간 원소 삭제/분리(삽입)
: list, (set)

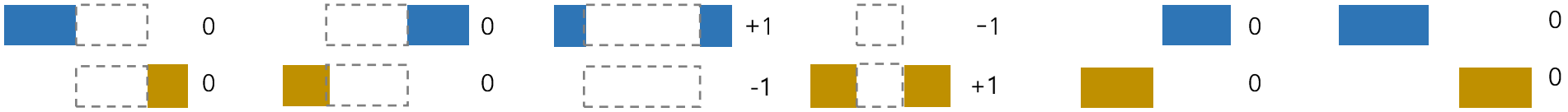
노드 개수?
50개를 넘지 않는다.
그렇기 때문에 사실 어떤 container를 사용해도 상관 없음

노드 개수가 50개를 넘지 않는 이유

두 노드가 겹치게 되는 경우 4가지 + 겹치지 않는 경우 2가지 (파랑 기준)



겹치는 부분 삭제 후 노드 개수 변화량

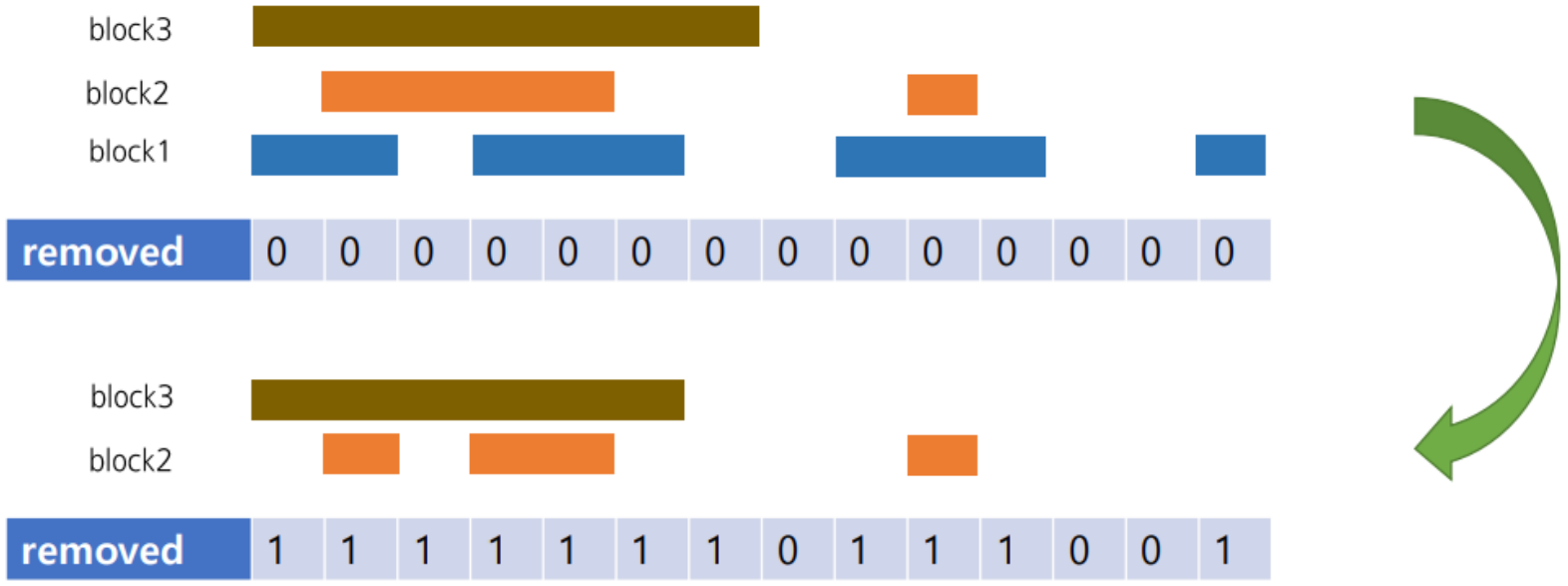


행이 최대 50이므로 기본 노드 최대 50개이다.
겹치는 부분을 지울 때, 총 노드개수의 합은 고정된다.

모든 열의 바닥과 가까운 블록 제거

1. **모든 열을 check 해가며 제거**
   - bool removed[10000] = 1/0 or
   - int removed[10000] = rcnt (rcnt는 함수호출때마다 증가)

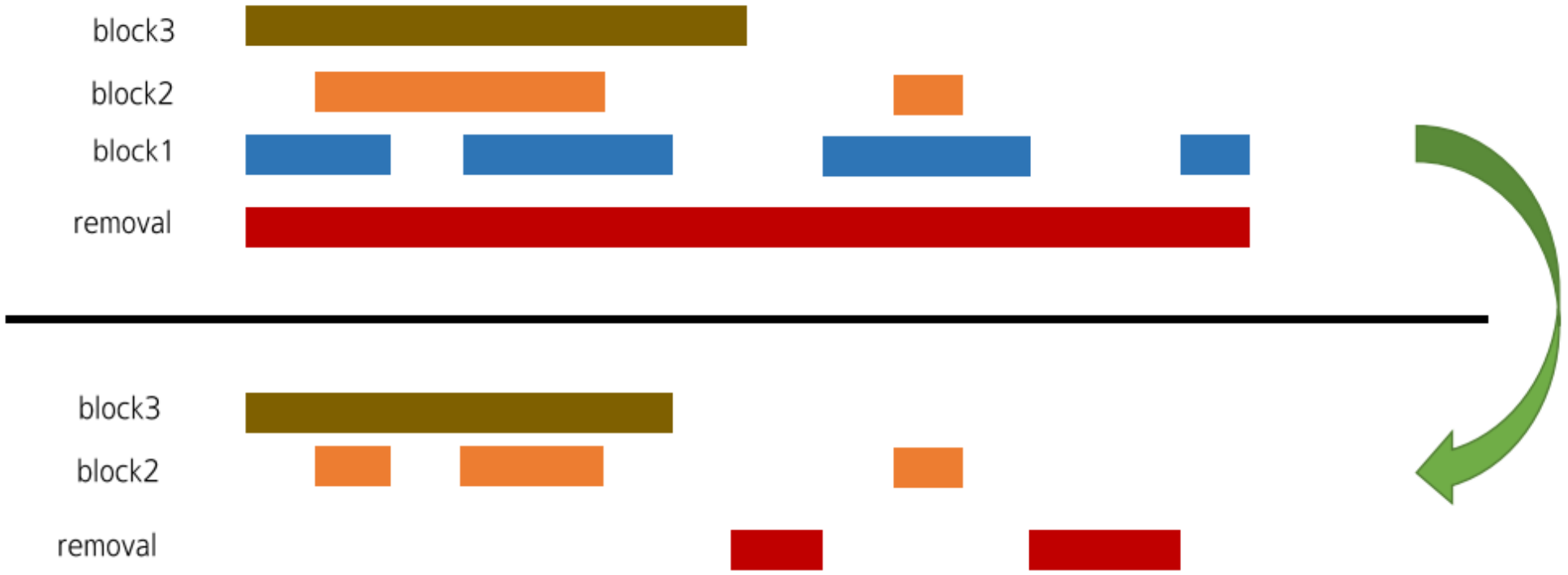
앞에 노드부터 해당 열을 전부 순회하며 이미 지워진 열의 노드들만 남긴다.
=> 기존 노드는 삭제한 상황에서 남는 노드들만 새롭게 추가



모든 열의 바닥과 가까운 블록 제거

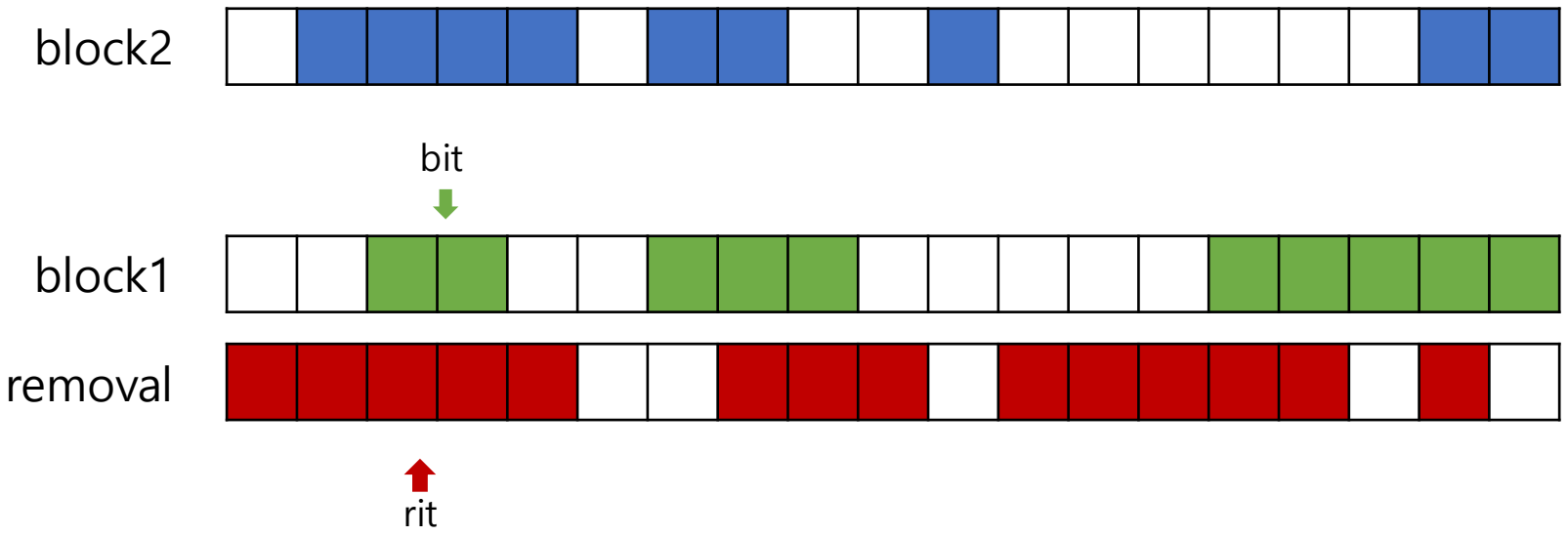
2. 지워야하는 연속된 열을 한 개의 노드로 관리하여 겹치는 부분 제거

- 열의 전체 구간을 한 개의 노드로 갖는 removal를 list로 생성 : {col_start, col_end} or {col, len}
- 앞의 block들부터 removal와 겹치는 구간을 제거/분할하며 새롭게 노드들을 구성해나간다.



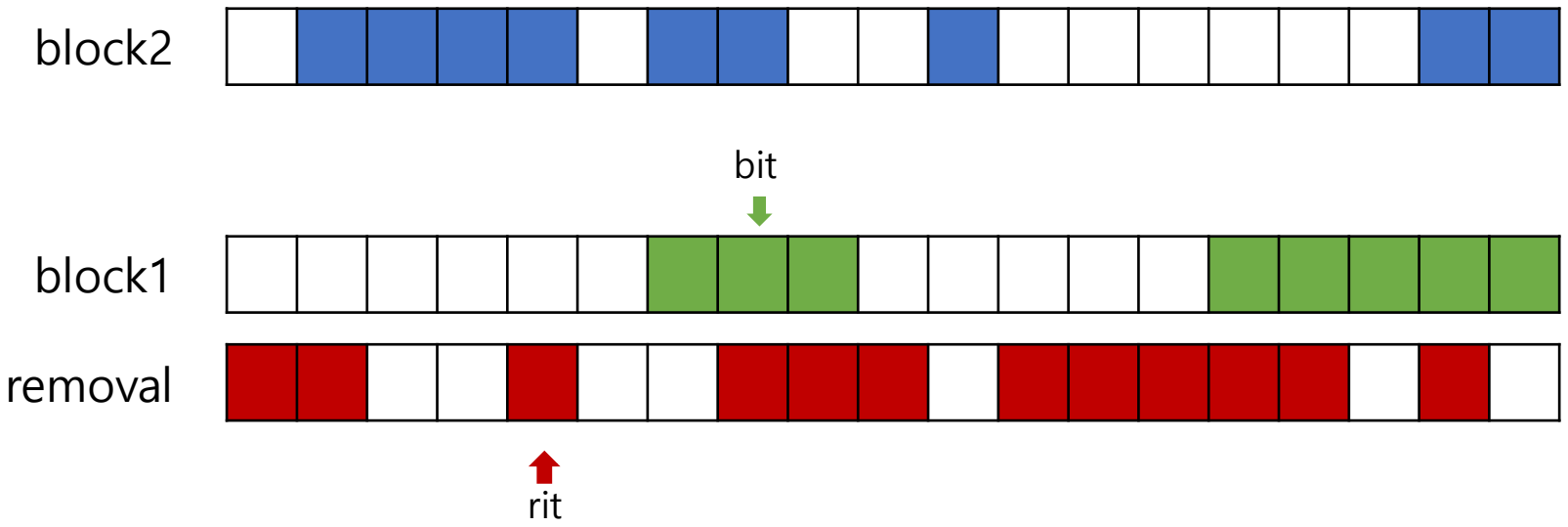
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



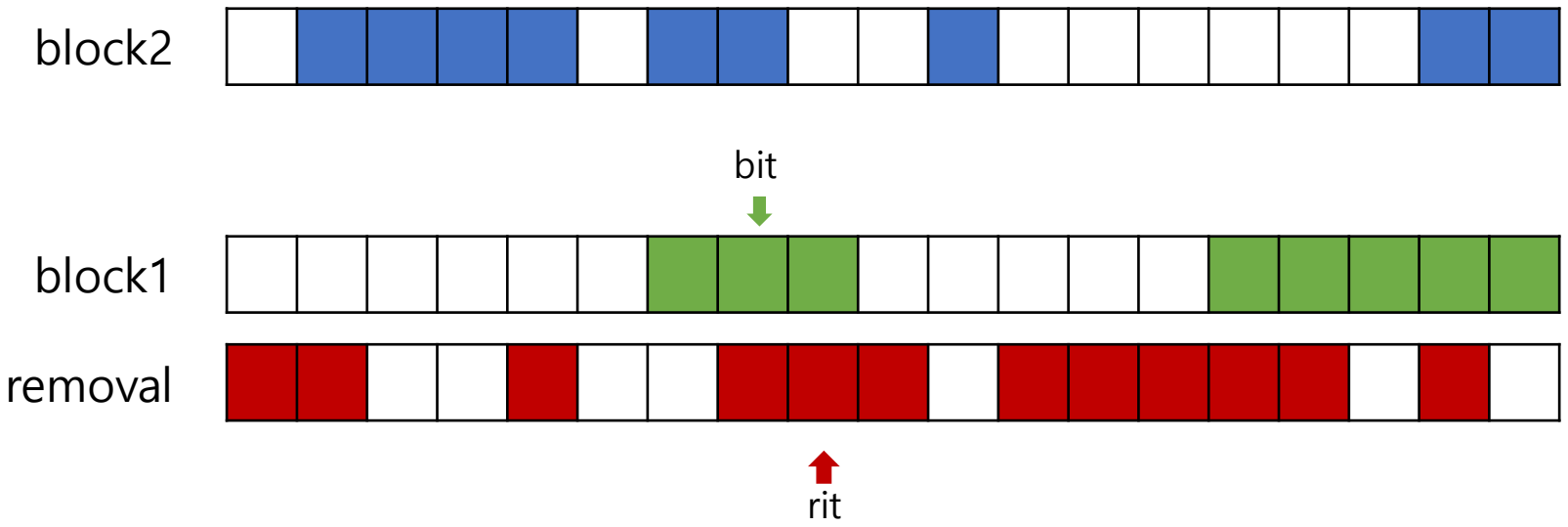
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



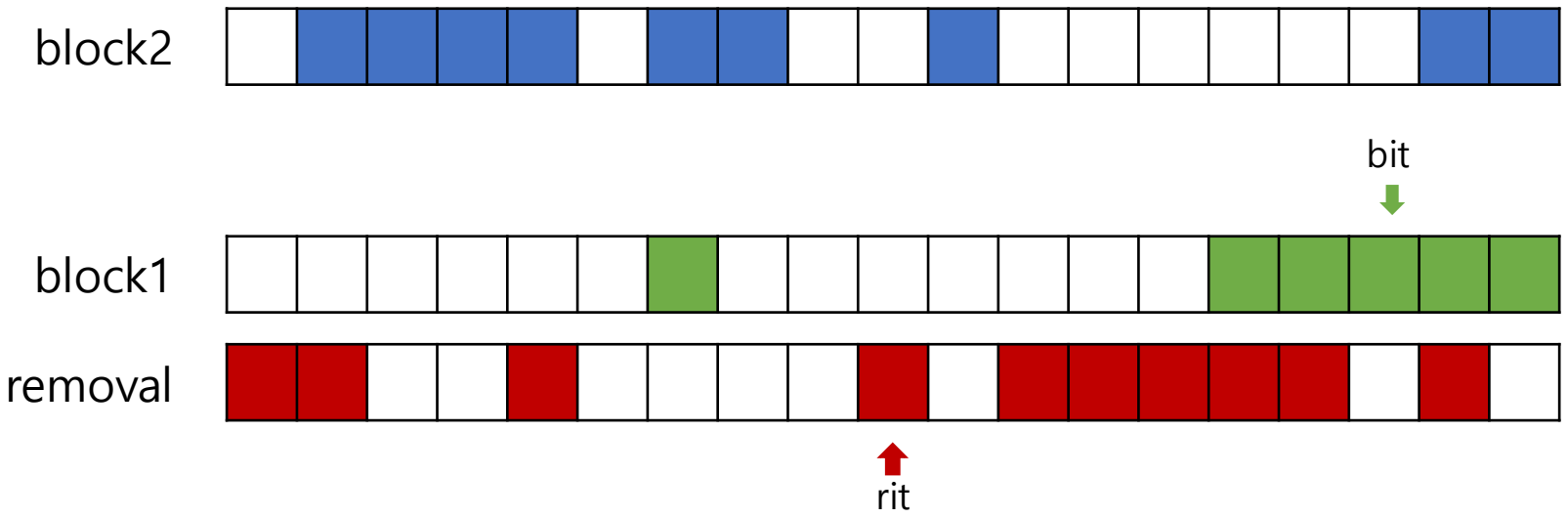
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



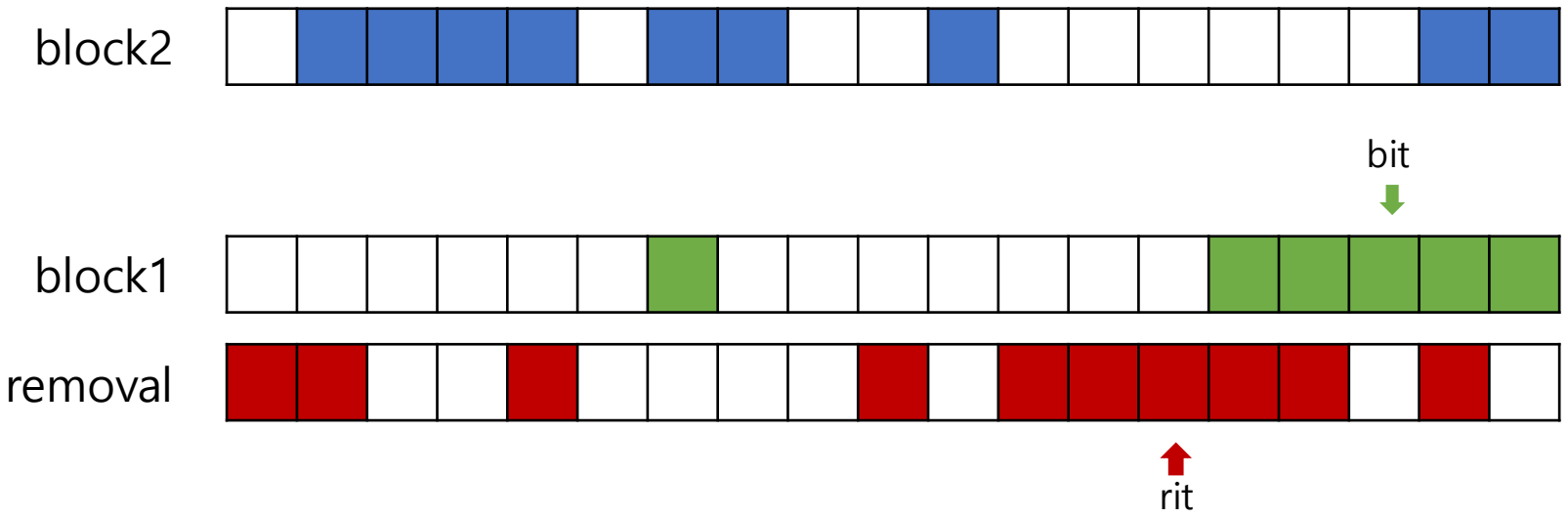
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



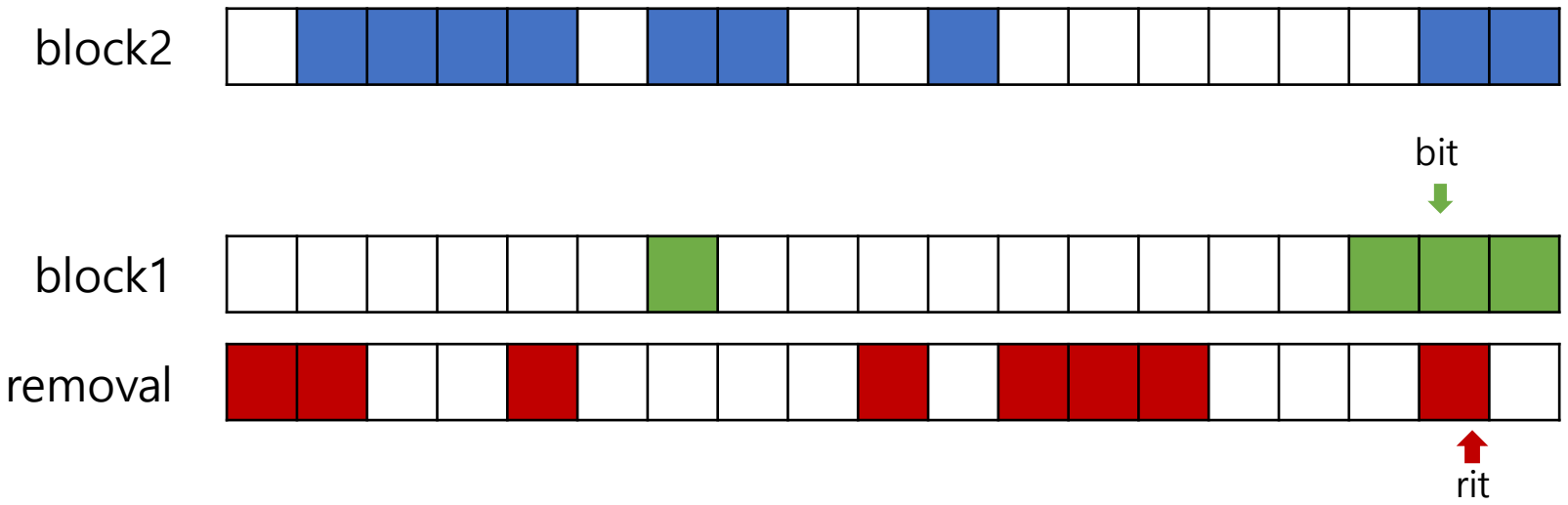
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



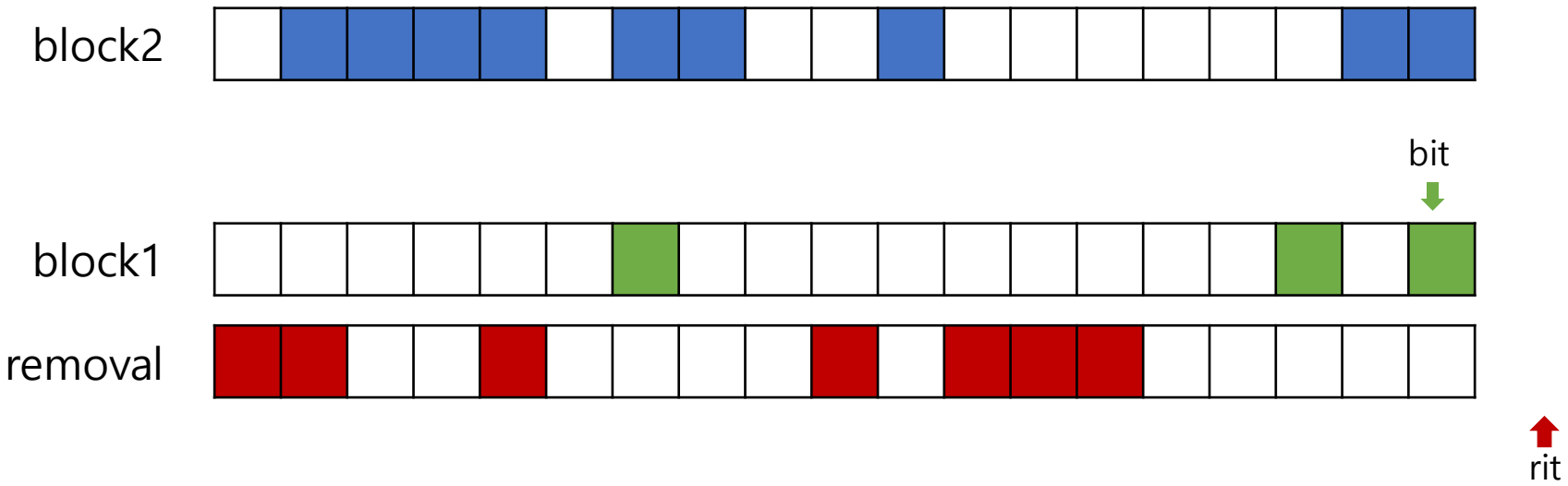
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



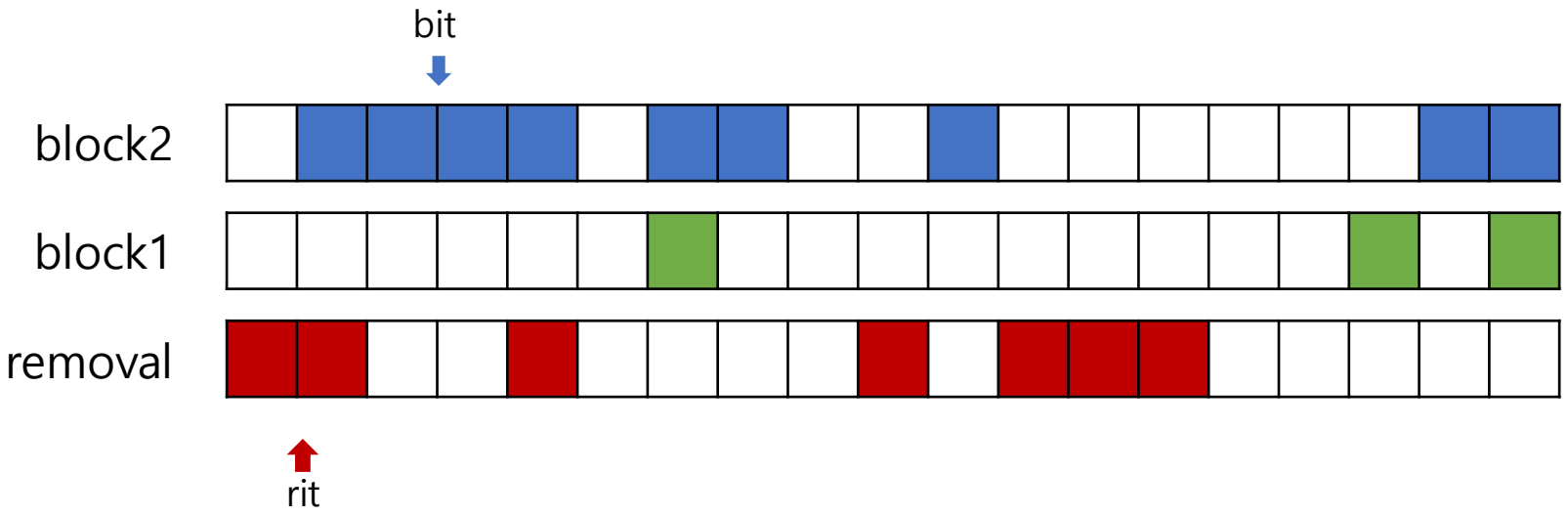
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



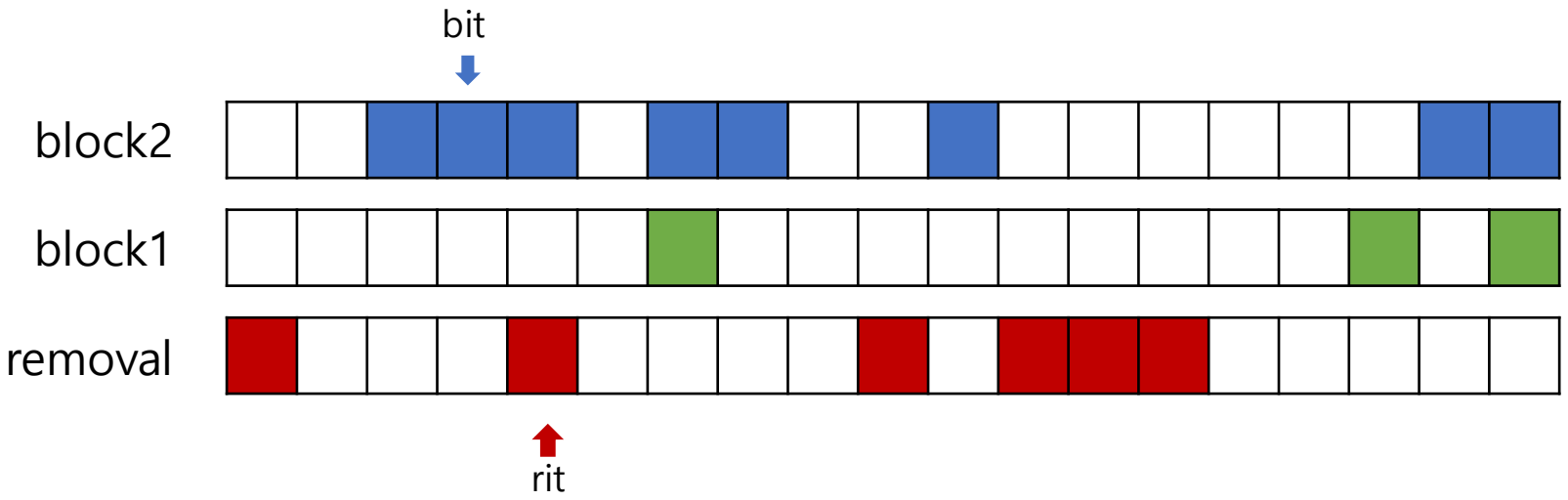
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



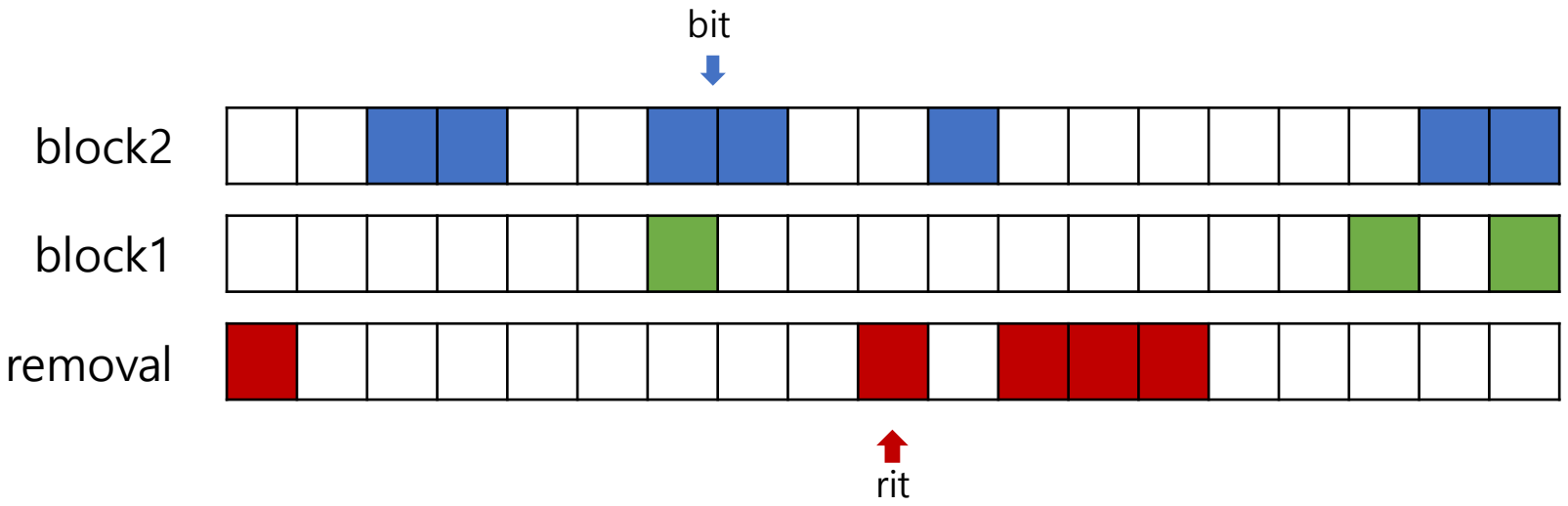
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



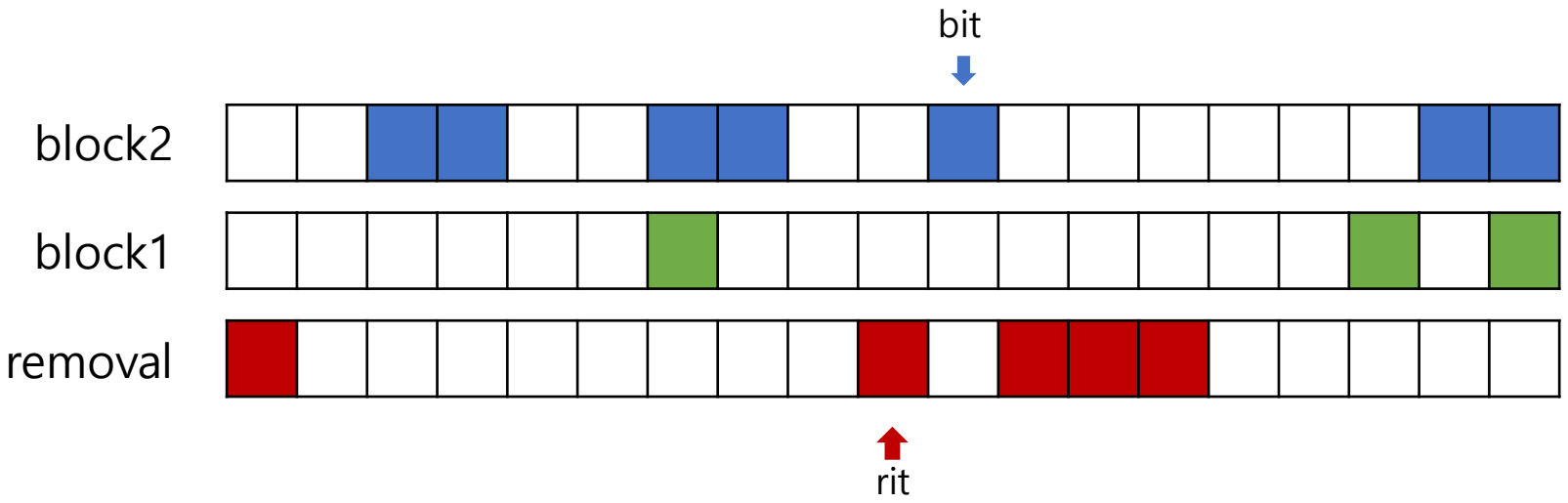
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



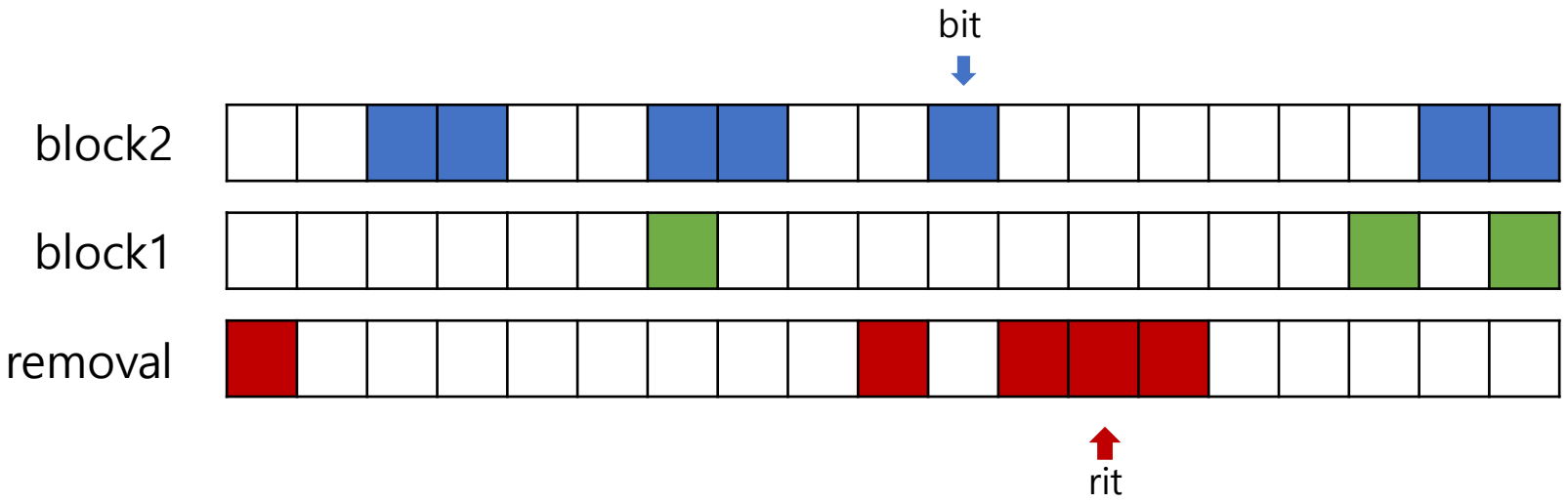
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



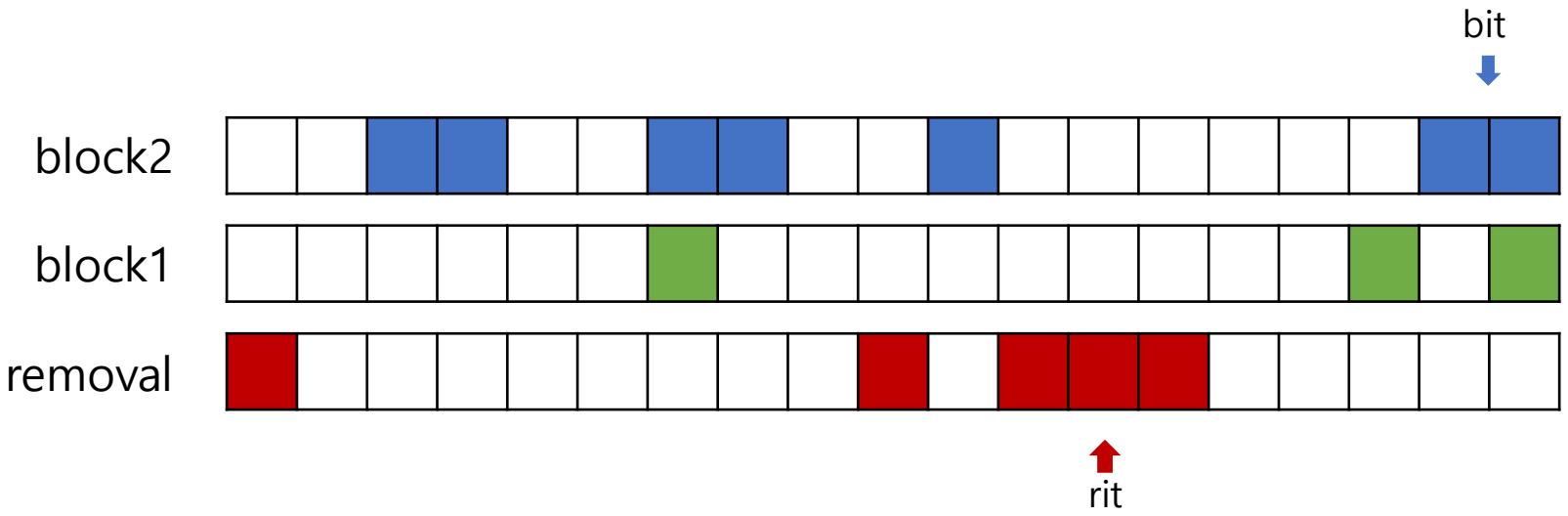
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



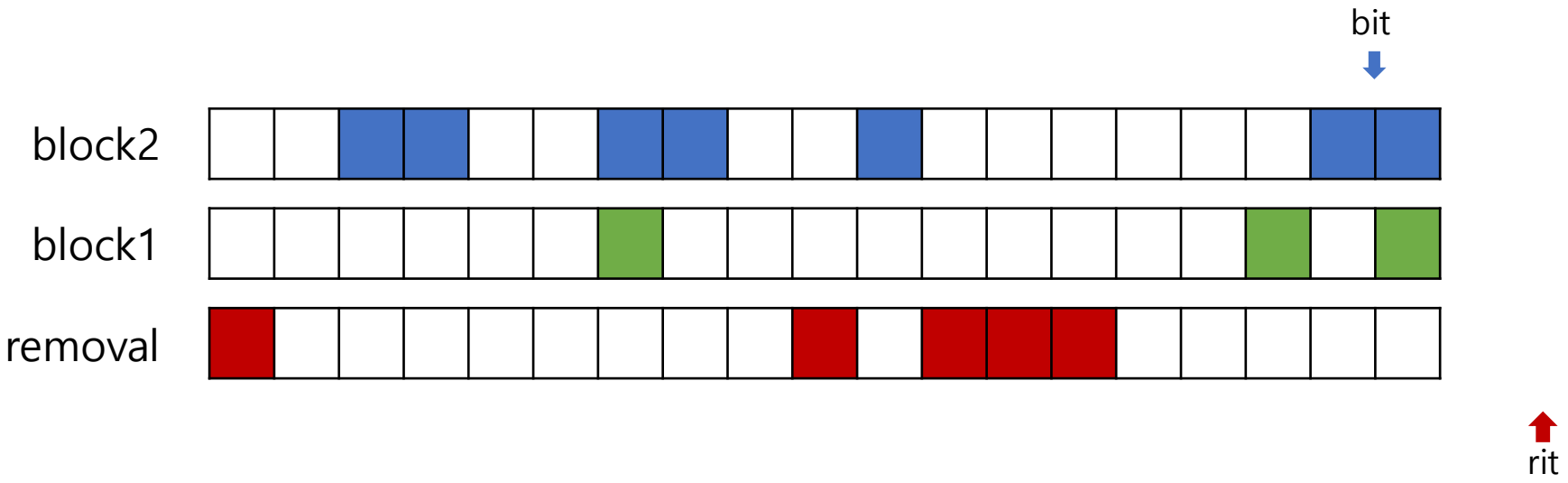
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



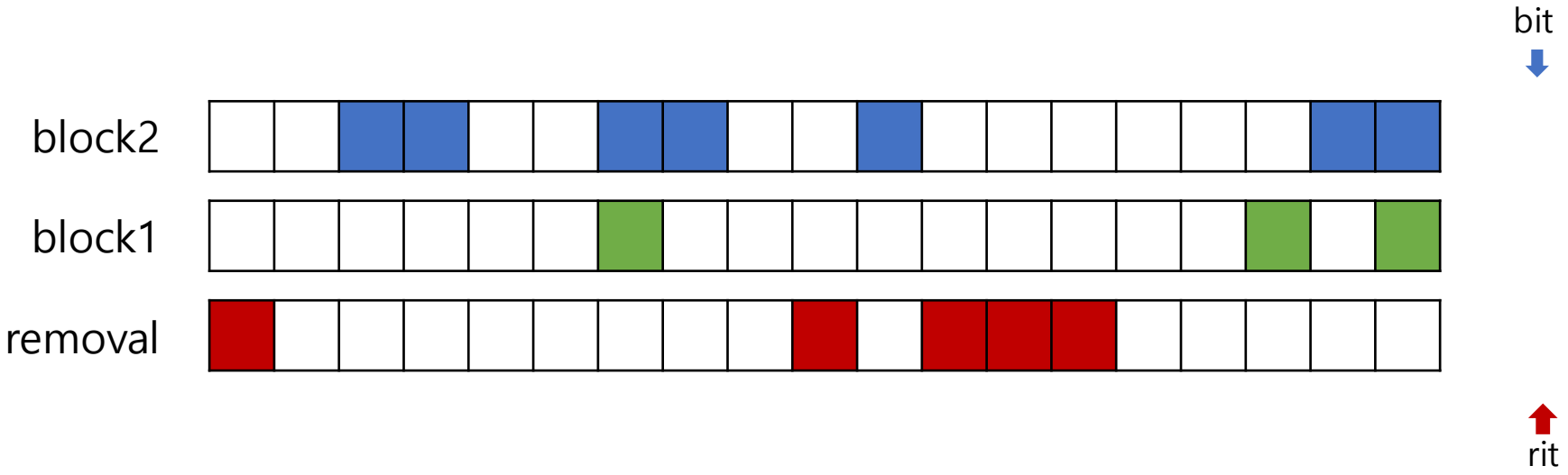
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



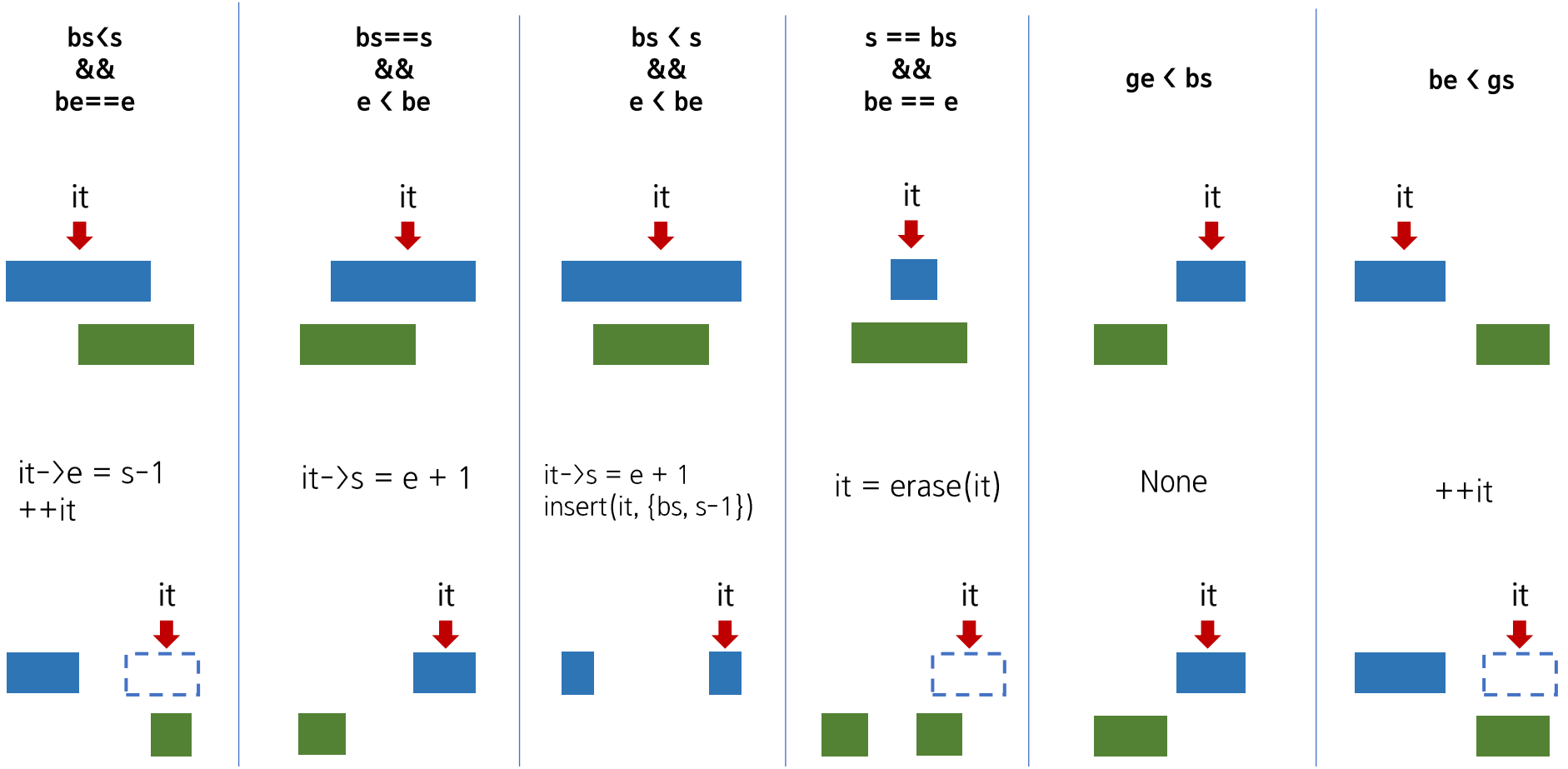
block, removal 겹치는 부분 삭제 방식

동일한 라인의 block들 단위로 removal의 맨앞에서부터 순차적으로 처리
block 라인(drop time)이 바뀌면 removal를 다시 맨앞에서부터 진행



두 노드의 겹치는 부분을 삭제하는 방법

파랑색 노드(it) = {bs,be}　　녹색 노드 = {gs,ge}　　겹치는 부분 = {s,e} = {max(bs,gs), min(be,ge)}




감사합니다