그림 1

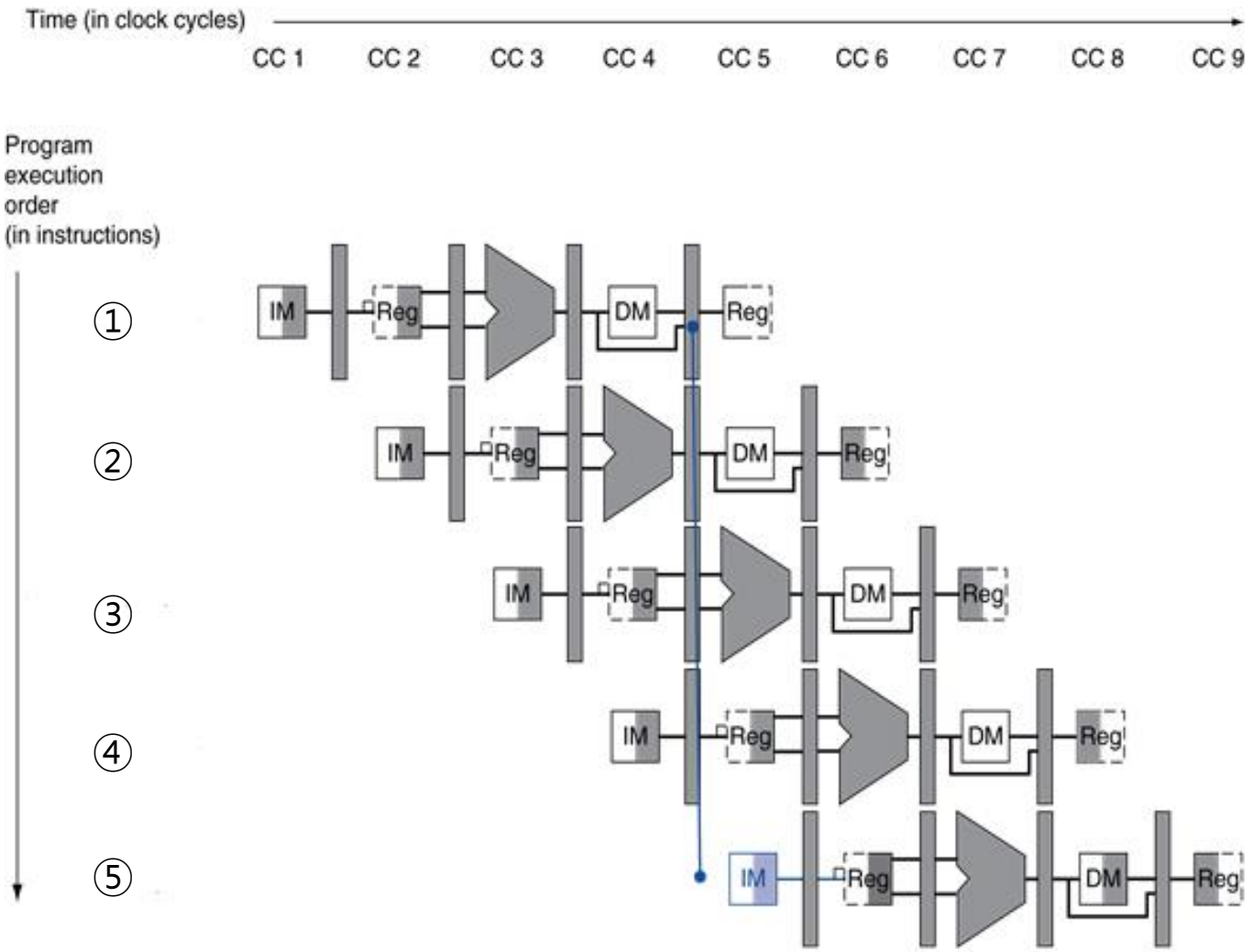

Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
①
②
③
④
⑤
IM
Reg
DM
Reg

그림 2

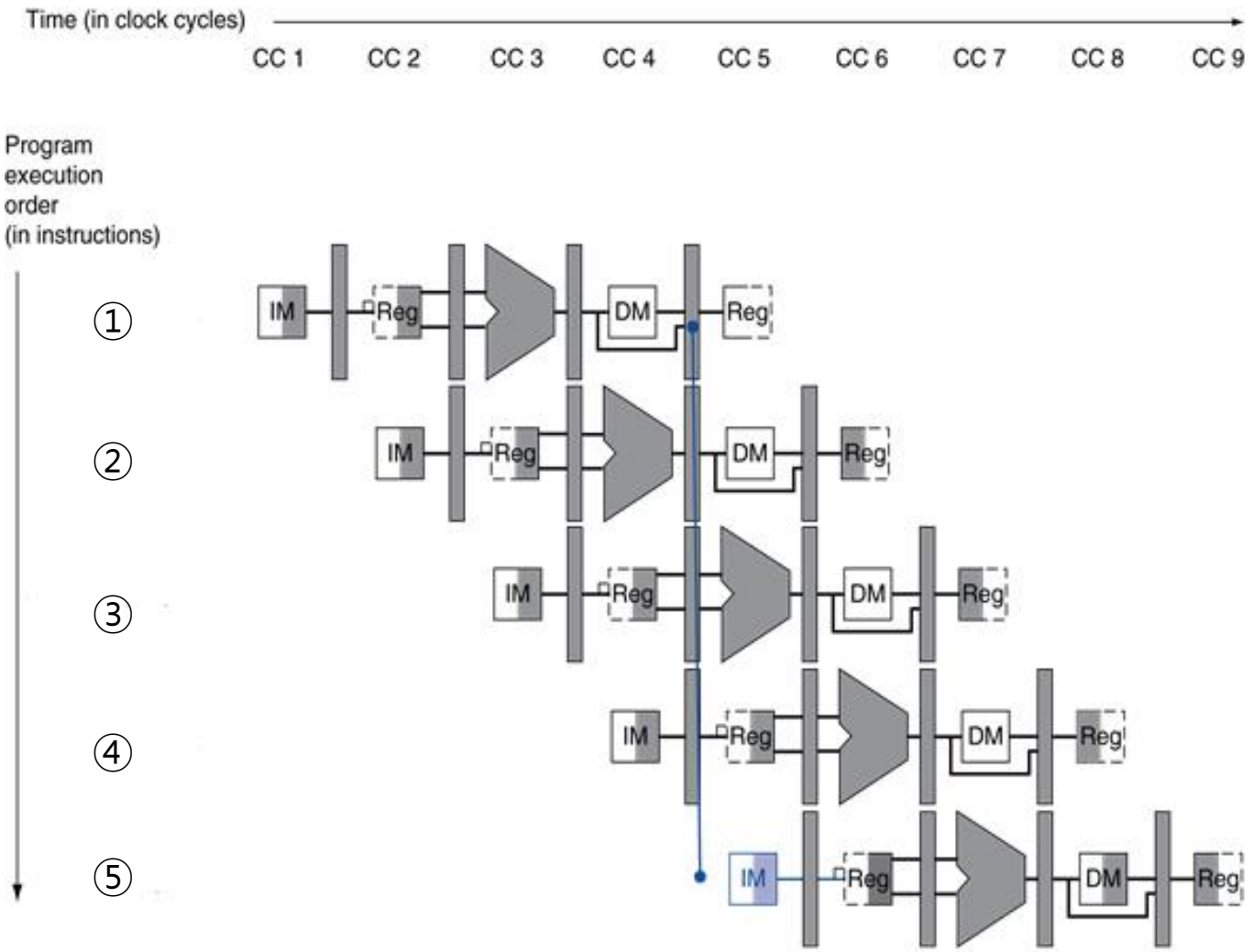

Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
①
②
③
④
⑤
IM
Reg
DM
Reg

그림 3

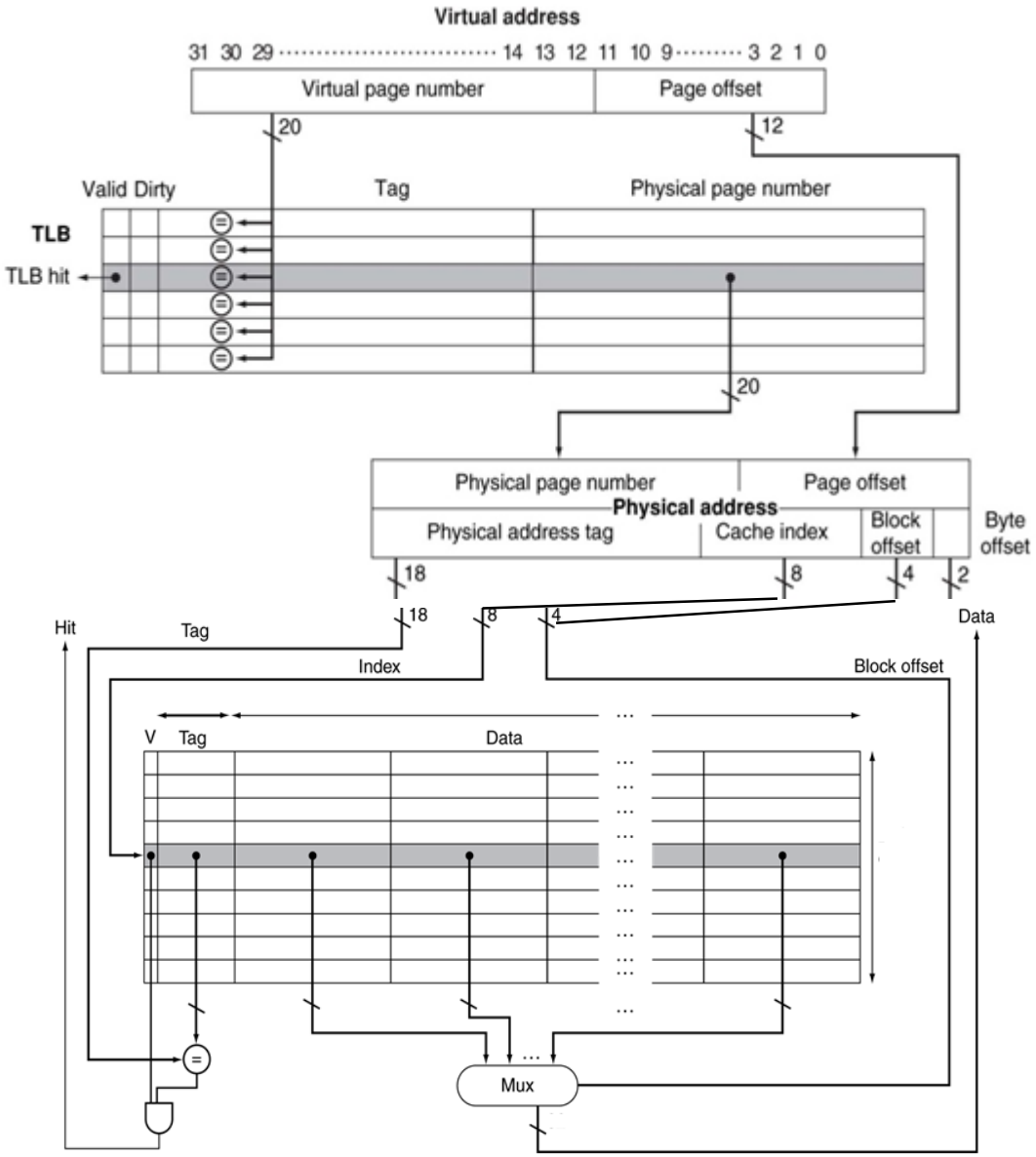

Virtual address
31 30 29 ····· 14 13 12 11 10 9 ····· 3 2 1 0
Virtual page number
Page offset
20
12
Valid Dirty
Tag
Physical page number
TLB
TLB hit
20
Physical page number
Page offset
Physical address
Physical address tag
Cache index
Block offset
Byte offset
18
8
4
2
18
8
4
Hit
Tag
Data
Index
Block offset
V
Tag
Data
Mux

그림 4

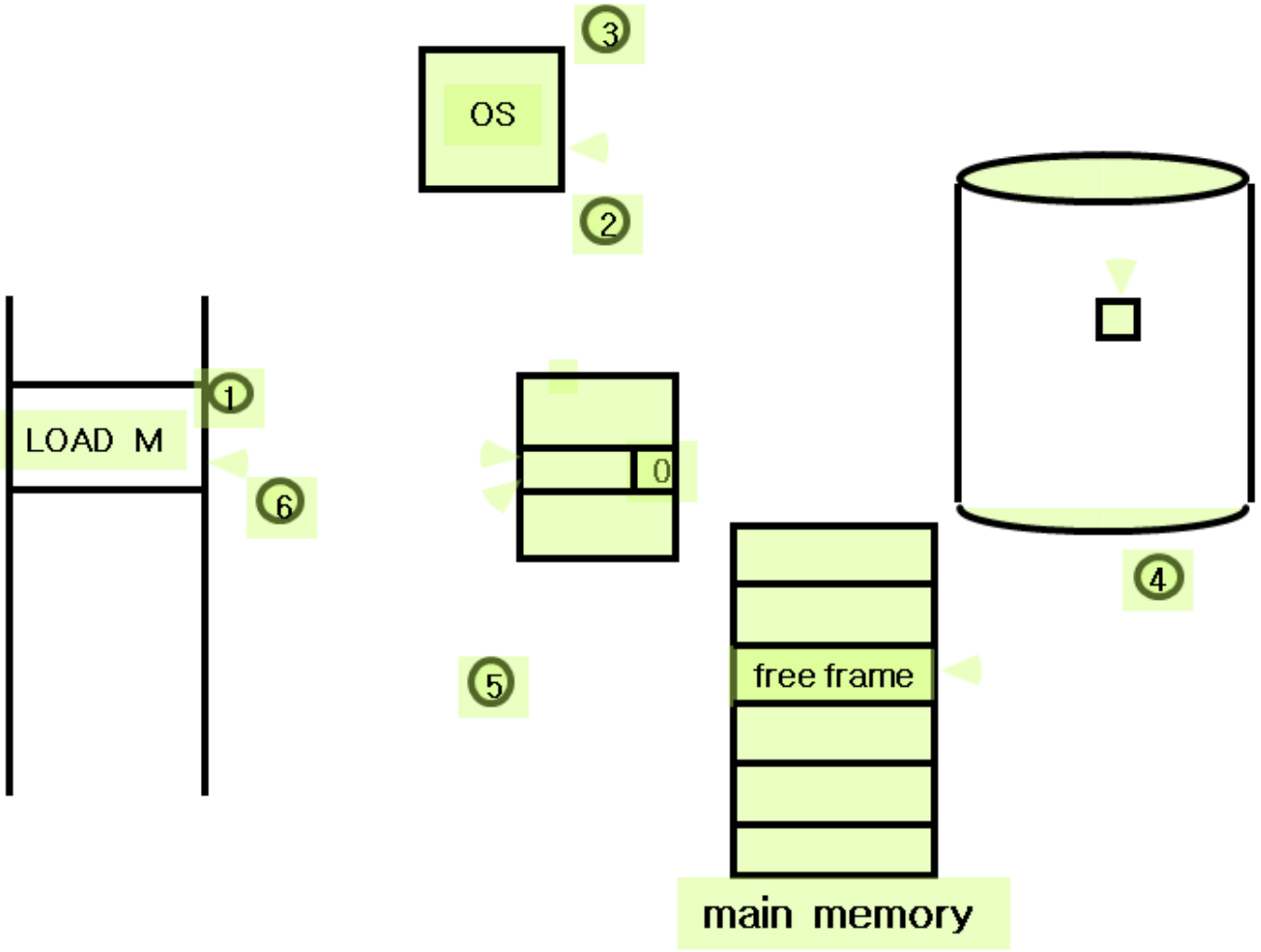
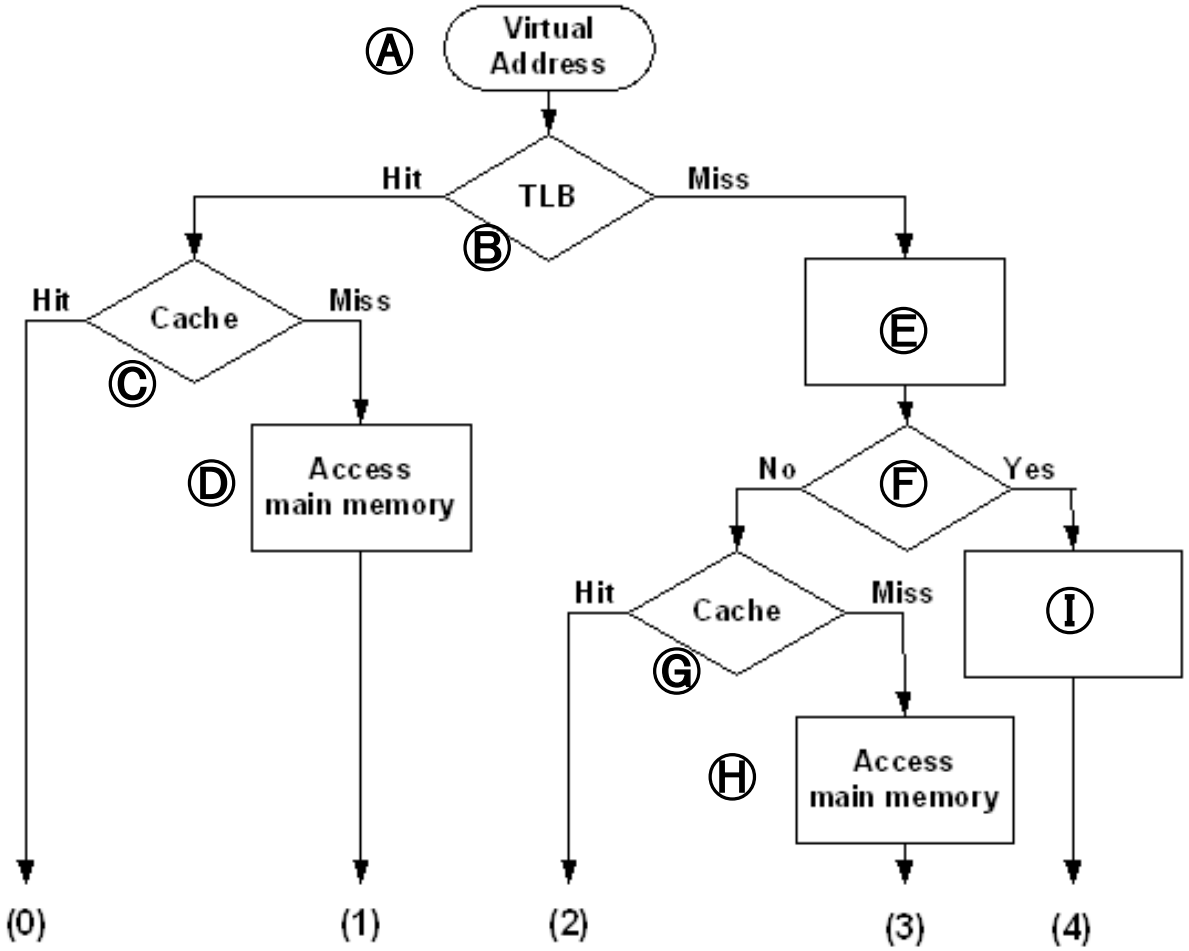


그림 5


Ⓐ
Virtual Address
Hit
TLB
Miss
Ⓑ
Hit
Cache
Miss
Ⓒ
Ⓓ
Access main memory
Ⓔ
No
Ⓕ
Yes
Hit
Cache
Miss
Ⓖ
Ⓗ
Access main memory
Ⓘ
(0)
(1)
(2)
(3)
(4)

보조자료(1/3): Select Algorithm동작

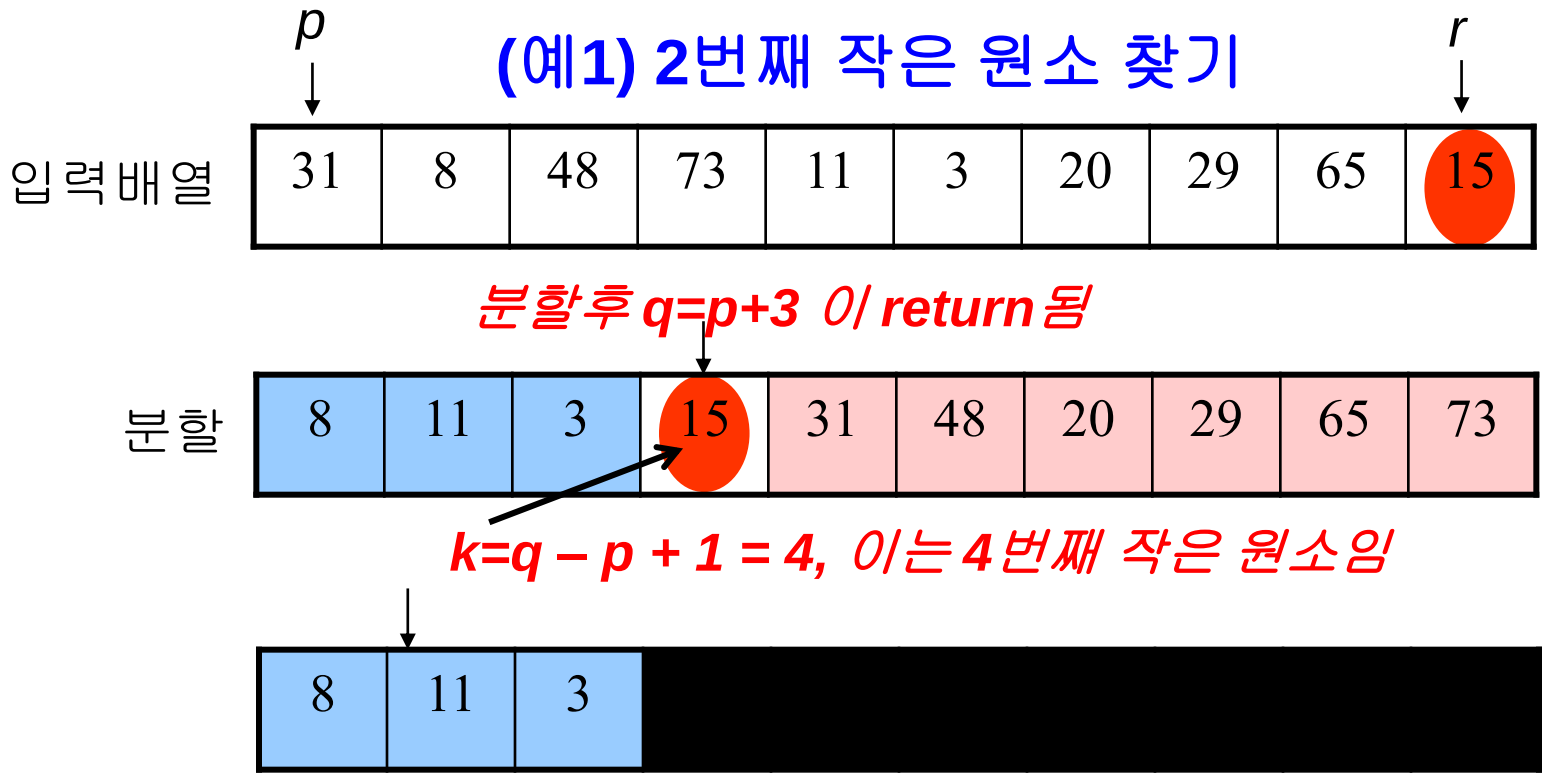
알고리즘 동작원리

1) Quicksort의 partition을 활용

2) Partition후 return 값 "q"=기준원소의 위치값을 이용하여 해당 partition을 선택한 뒤 recursive call

3) 최종적으로 원소 찾기

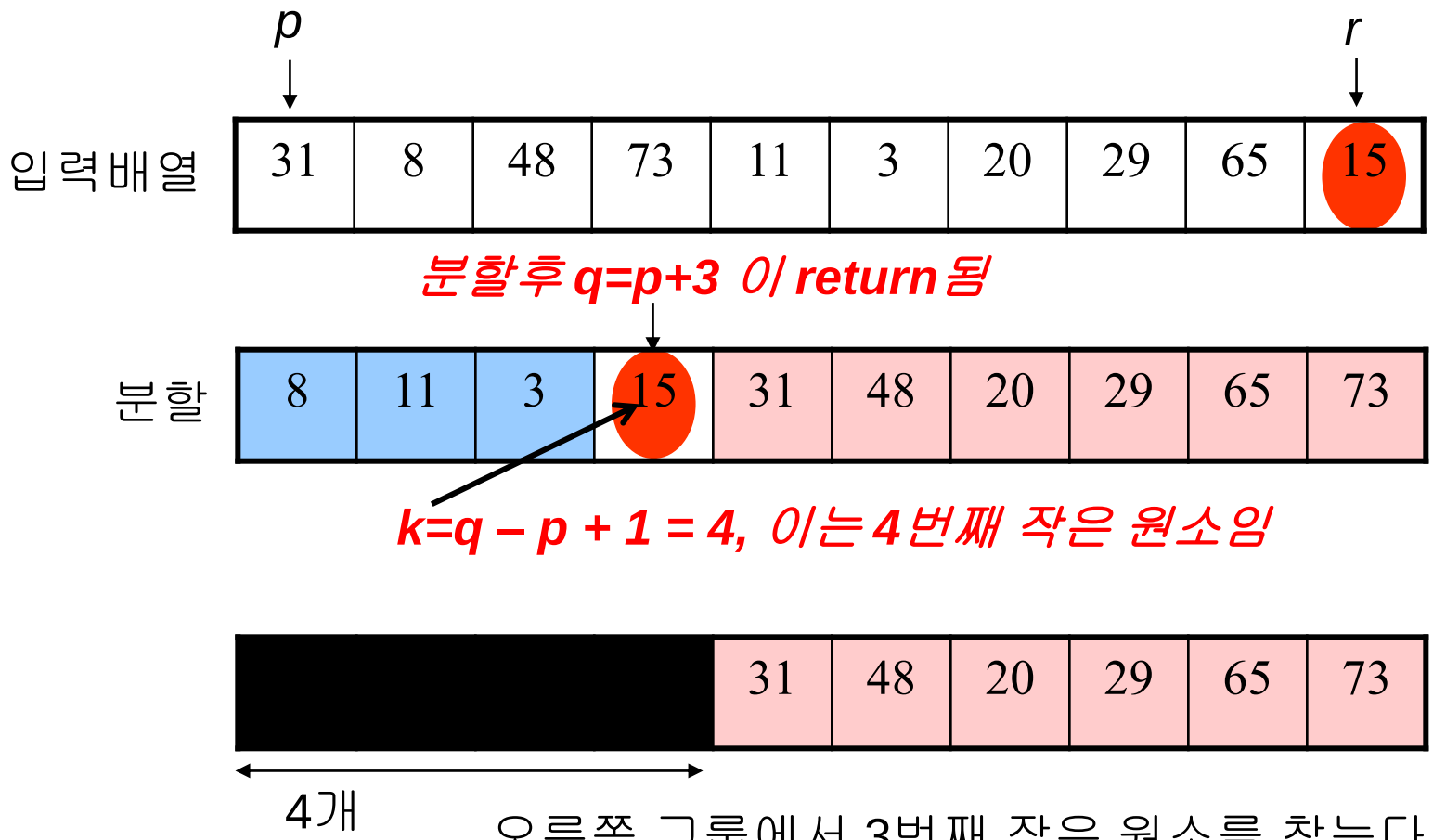
아래 예제에서는 맨 마지막 원소를 pivot로 사용



왼쪽 그룹에서 2번째 작은 원소를 찾는다

보조자료(2/3): Select Algorithm동작

(예2) 7번째 작은 원소 찾기



보조자료(3/3): quick sort

```
quicksort( array, first, last ){
partition( array, first, last );
quicksort( array, first, j-1 );
quicksort( array, j+1, last);
}
```

```
void quicksort(int array[8],int first,int last){
   int i, j, pivot, temp;

   if(first<last){
      pivot=first;
      i=first;
      j=last;

      while(i<j){
         while(array[i]<=array[pivot]&&i<last)
            i++;
         while(array[j]>array[pivot])
            j--;
         if(i<j){
            temp=array[i];
            array[i]=array[j];
            array[j]=temp;
         }
      }

      temp=array[pivot];
      array[pivot]=array[j];
      array[j]=temp;
      quicksort(array,first,j-1);
      quicksort(array,j+1,last);

   }
}
```

Partition