

# Data Structure

Week 8  
KyuDong SIM

# 1. 이번 주 실습 내용

- Sparse Matrix (Linked List)

# Sparse Matrix Data Structure Approach

- 구조 부분과 데이터 부분을 나눔
  - Head vs. Entry
- Union 구조를 활용하여 선택적으로 사용
  - Head에서는 next 포인터만 사용
  - Entry에서는 Entry 정보 사용
- 각각의 Head 들은 배열로 저장
  - hdnode

```
typedef enum { Head, Entry } tagField;

typedef struct matrixNode* matrixPtr;

typedef struct entryNode
{
    int row;
    int col;
    int value;
} entryNode;

typedef struct matrixNode
{
    matrixPtr down;
    matrixPtr right;
    tagField tag;
    union
    {
        matrixPtr next;
        entryNode entry;
    } u;
} matrixNode;

matrixPtr Hn[MAX_SIZE];
```

# Sparse Matrix Data Structure

- tagField : Head, Entry 구분
  - Head
  - Entry
- entryNode : Entry 인 경우 사용하는 정보
  - row, col, value
- matrixNode : 기본 구조
  - down : 자기 아래에 있는 노드
  - right : 자기 왼쪽에 있는 노드
  - tag : 이 구조 구분(Head 인지 Entry 인지)
  - u : Head, Entry에 따라 다른 데이터 사용
- hdnode : Head 만 모여있는 구조

```
typedef enum { Head, Entry } tagField;

typedef struct matrixNode* matrixPtr;

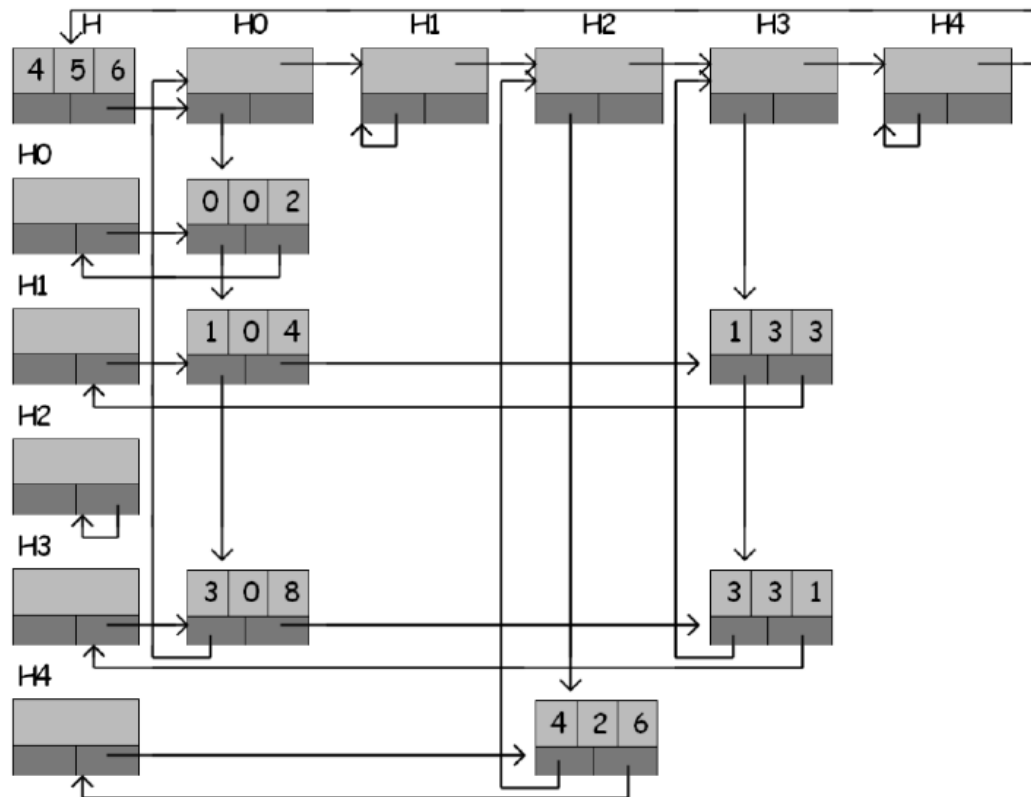
typedef struct entryNode
{
    int row;
    int col;
    int value;
} entryNode;

typedef struct matrixNode
{
    matrixPtr down;
    matrixPtr right;
    tagField tag;
    union
    {
        matrixPtr next;
        entryNode entry;
    } u;
} matrixNode;

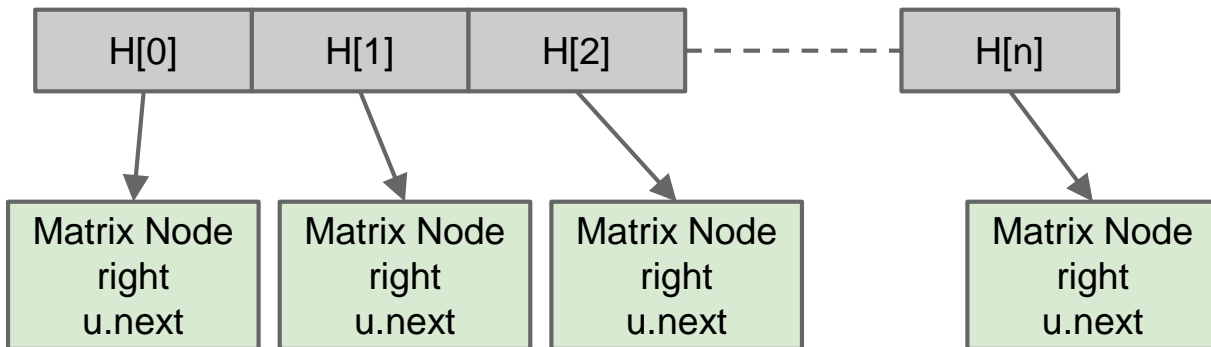
matrixPtr Hn[MAX_SIZE];
```

# Implement in Book

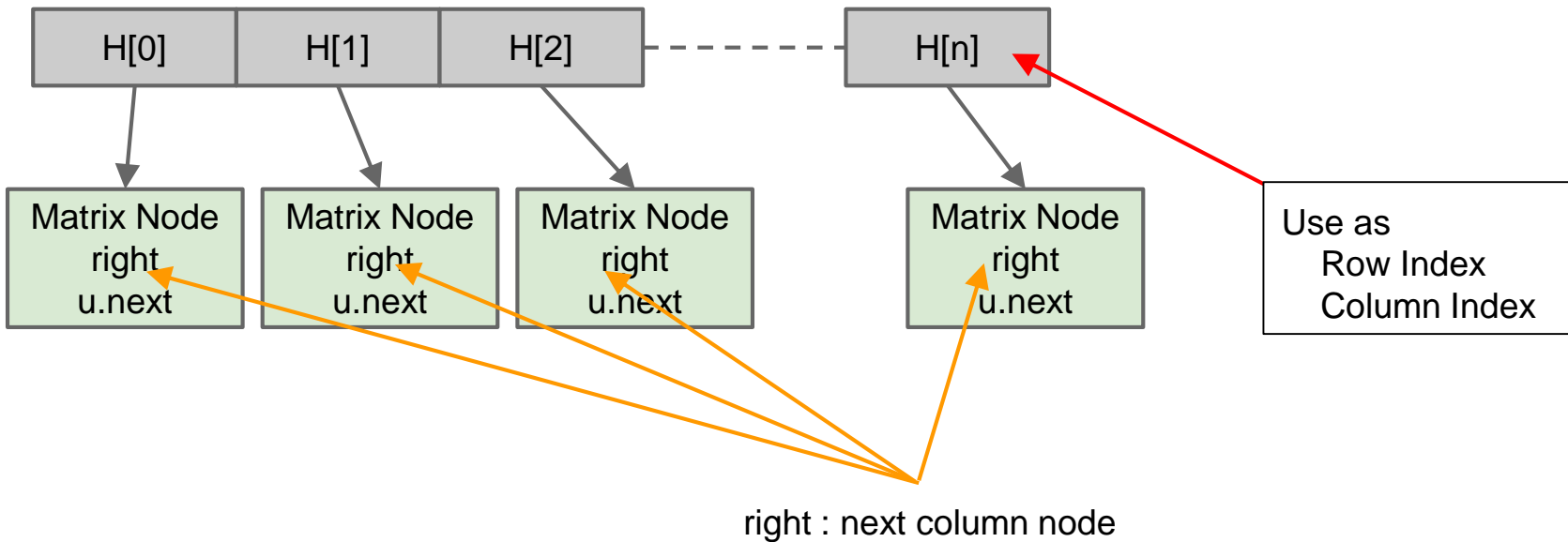
- 교재 구조 설명
- Circular 구조 사용
- H0~H4는 모두 동일한것
  - u.next 와 right 두개의 포인터를 사용하여 구분



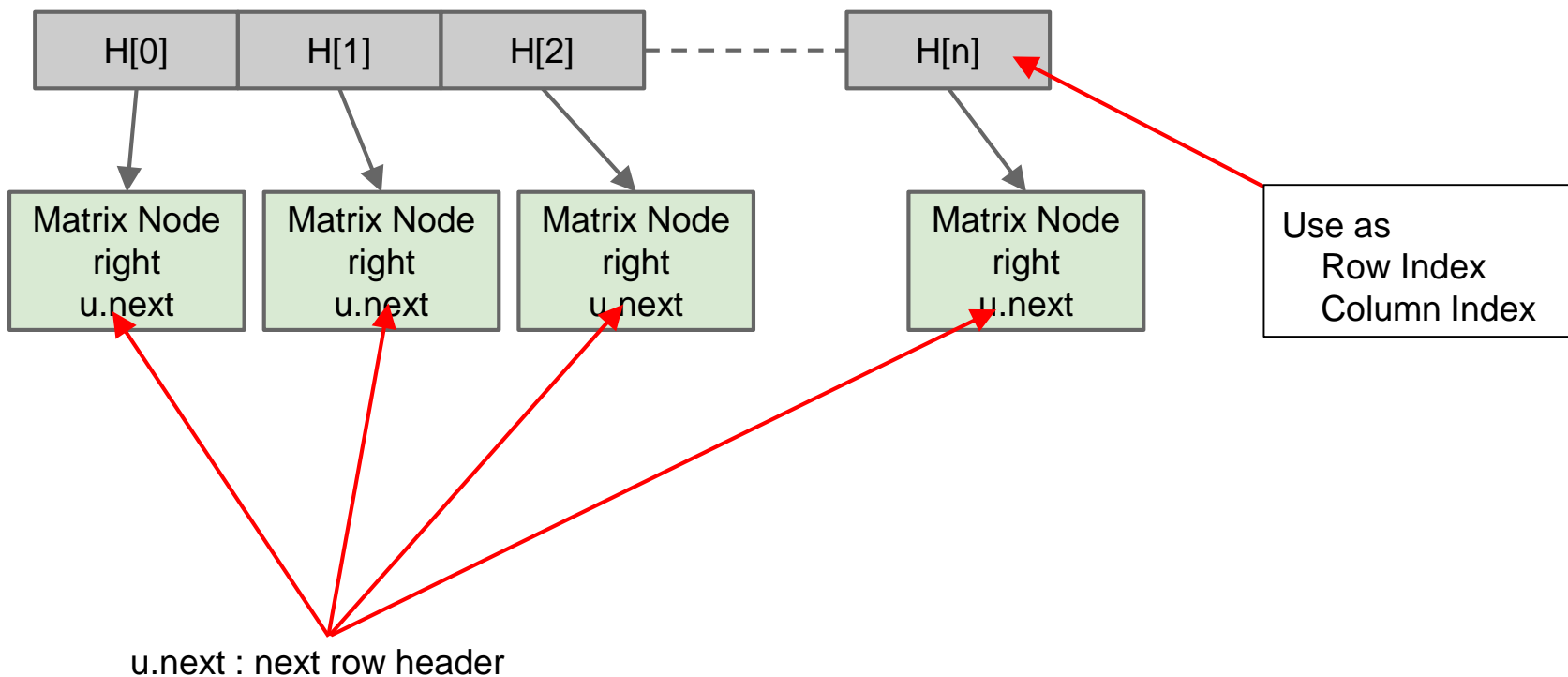
# Implement in Book



# Implement in Book

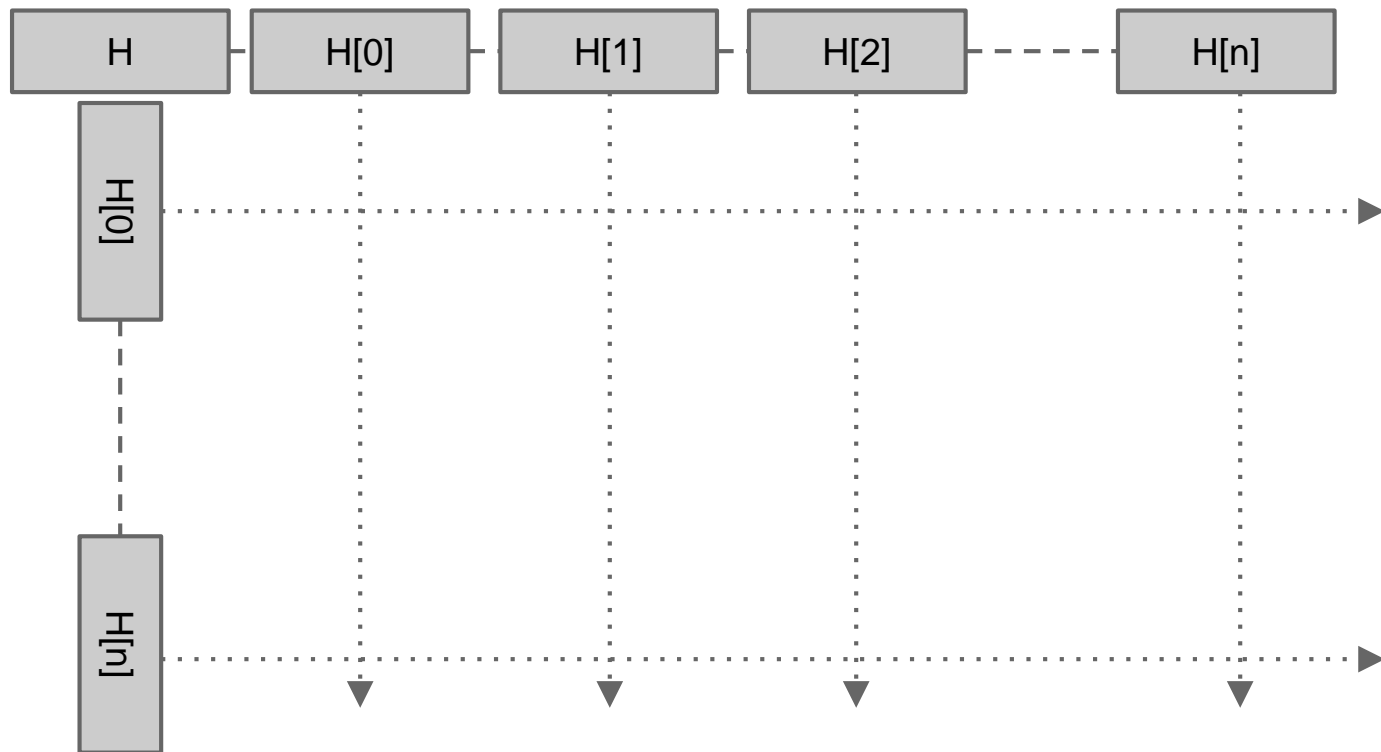


# Implement in Book

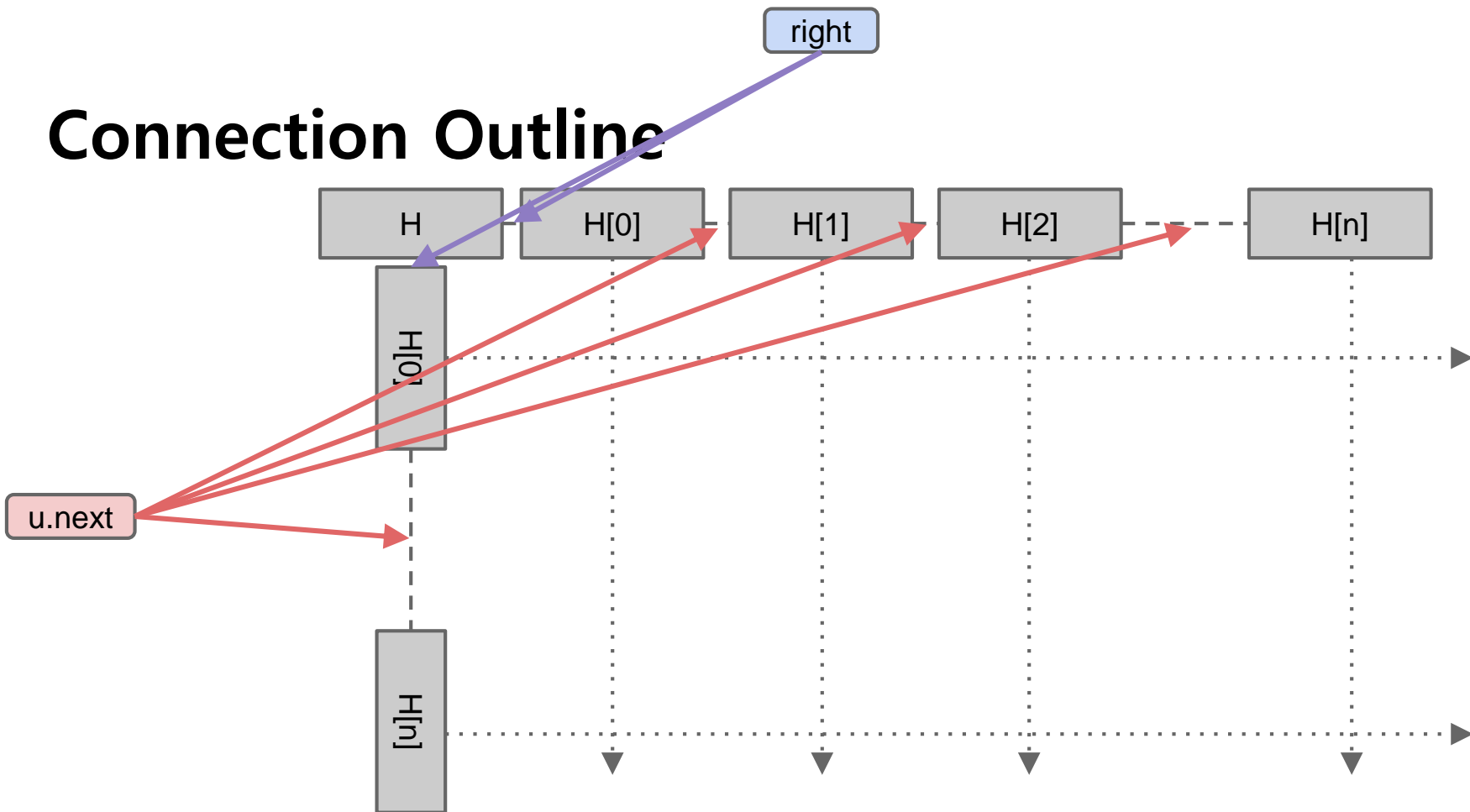




# Connection Outline

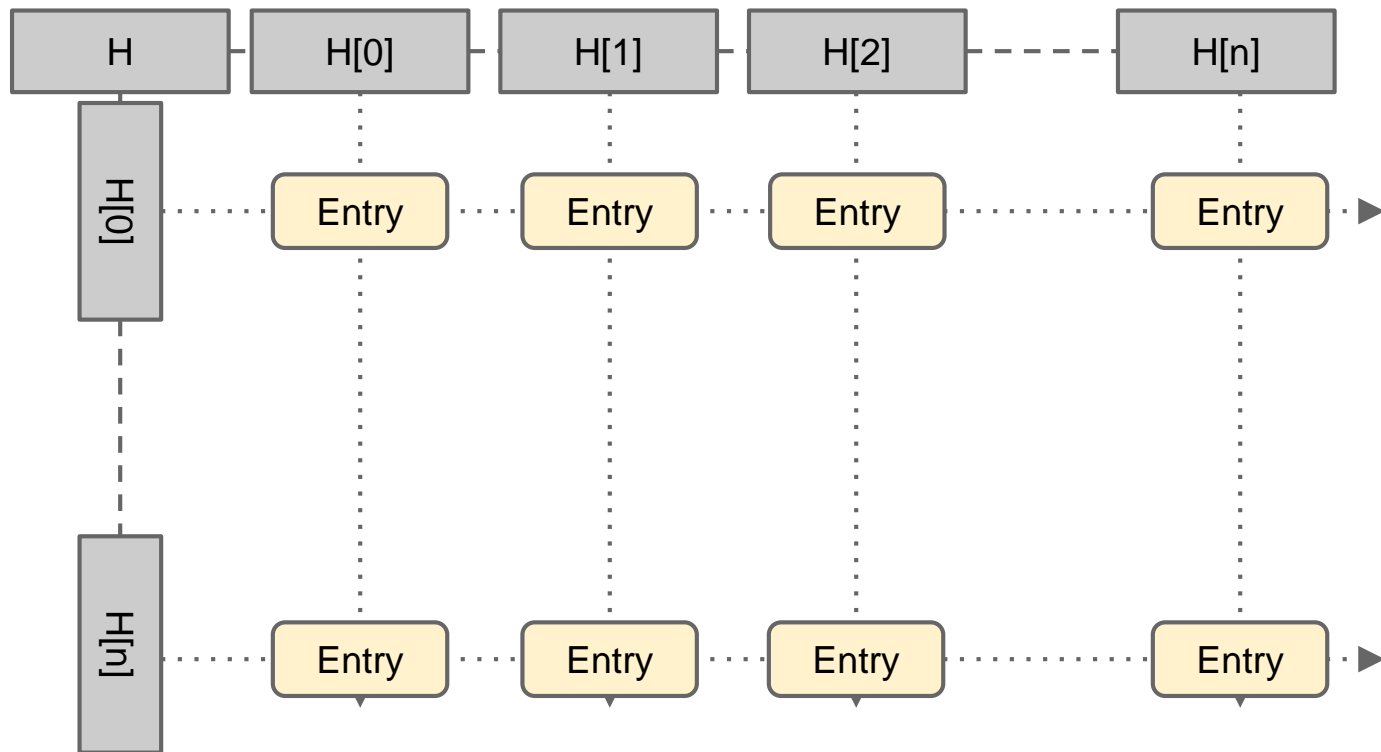


# Connection Outline

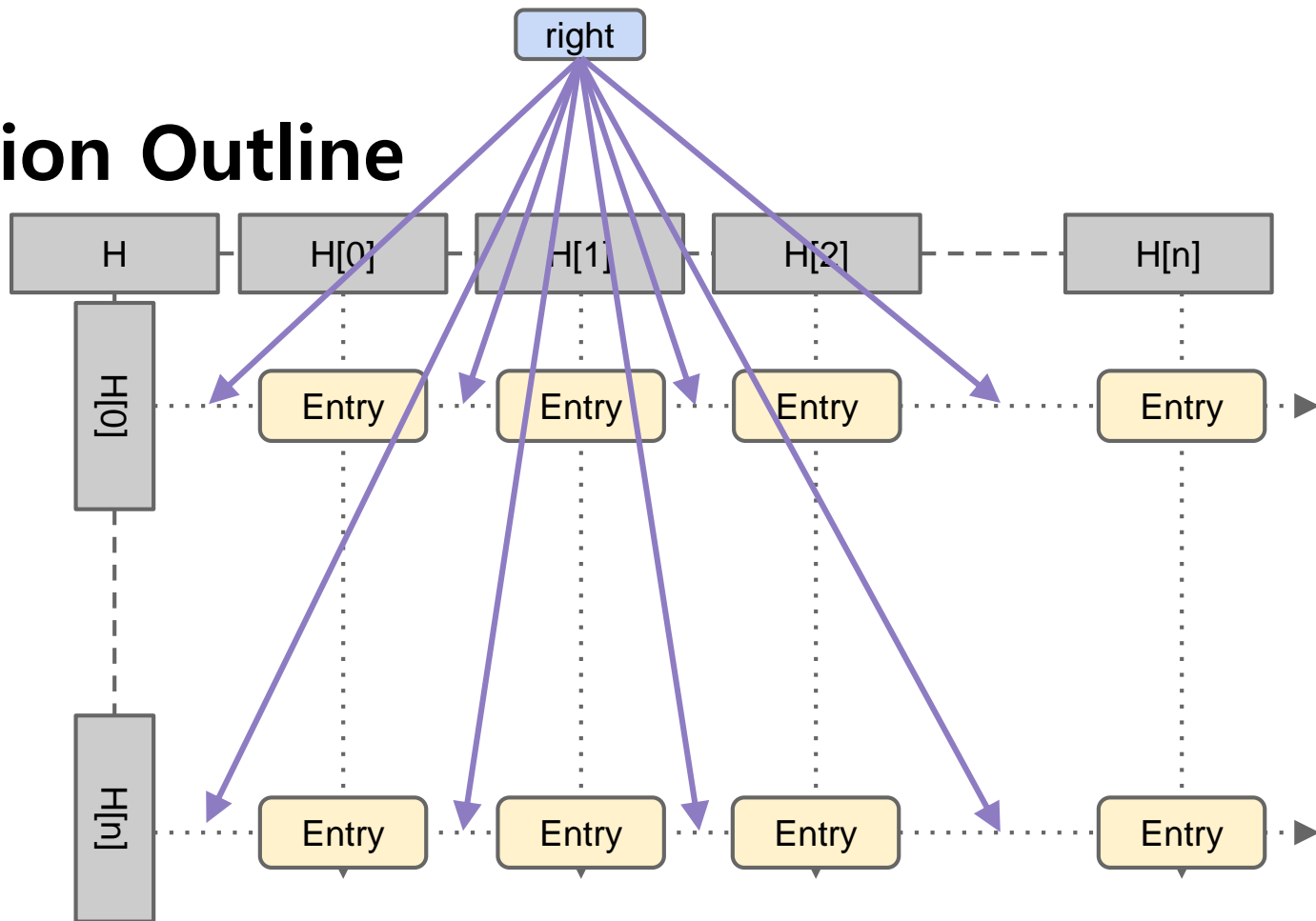


right

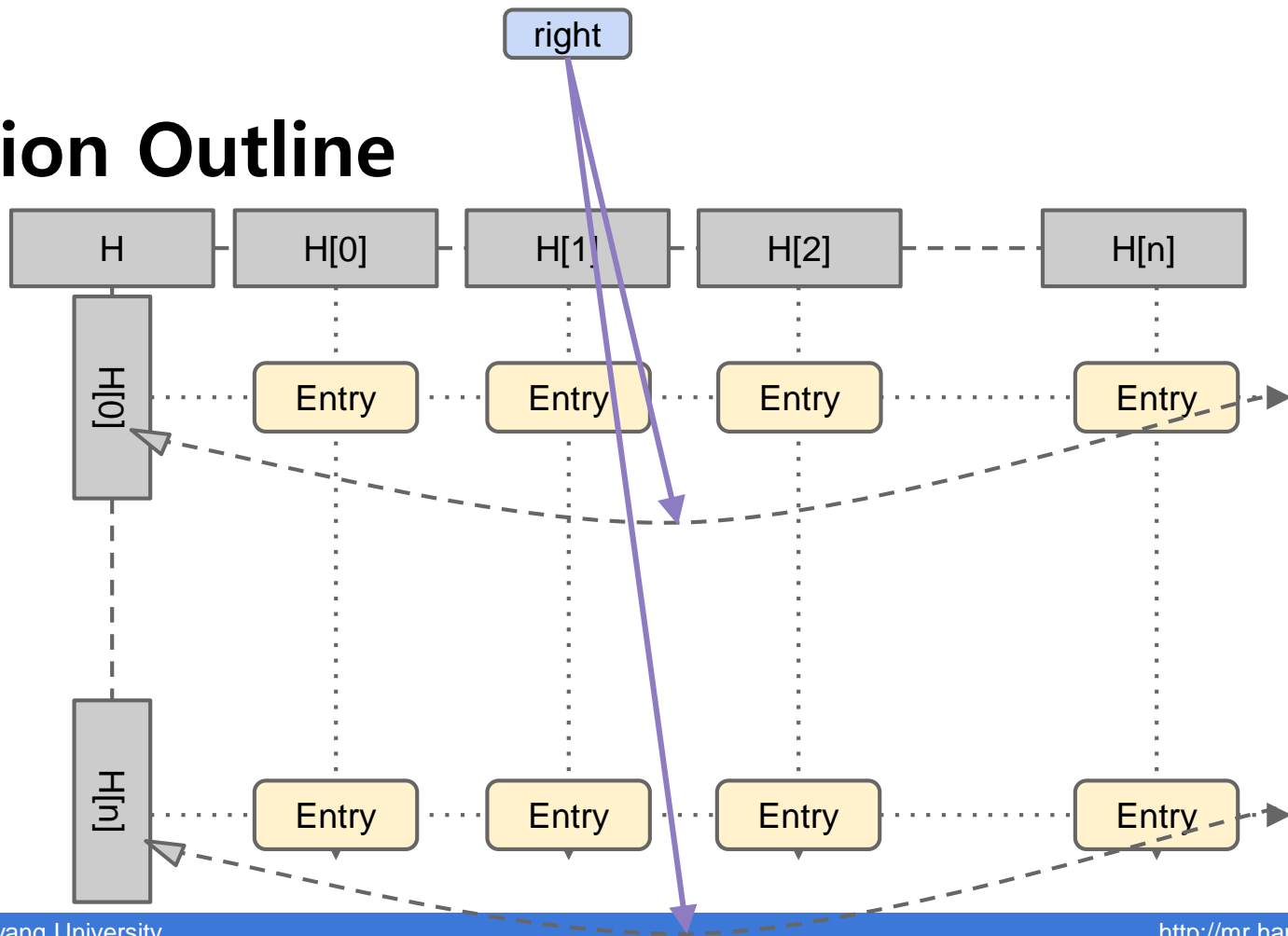
# Connection Outline



# Connection Outline

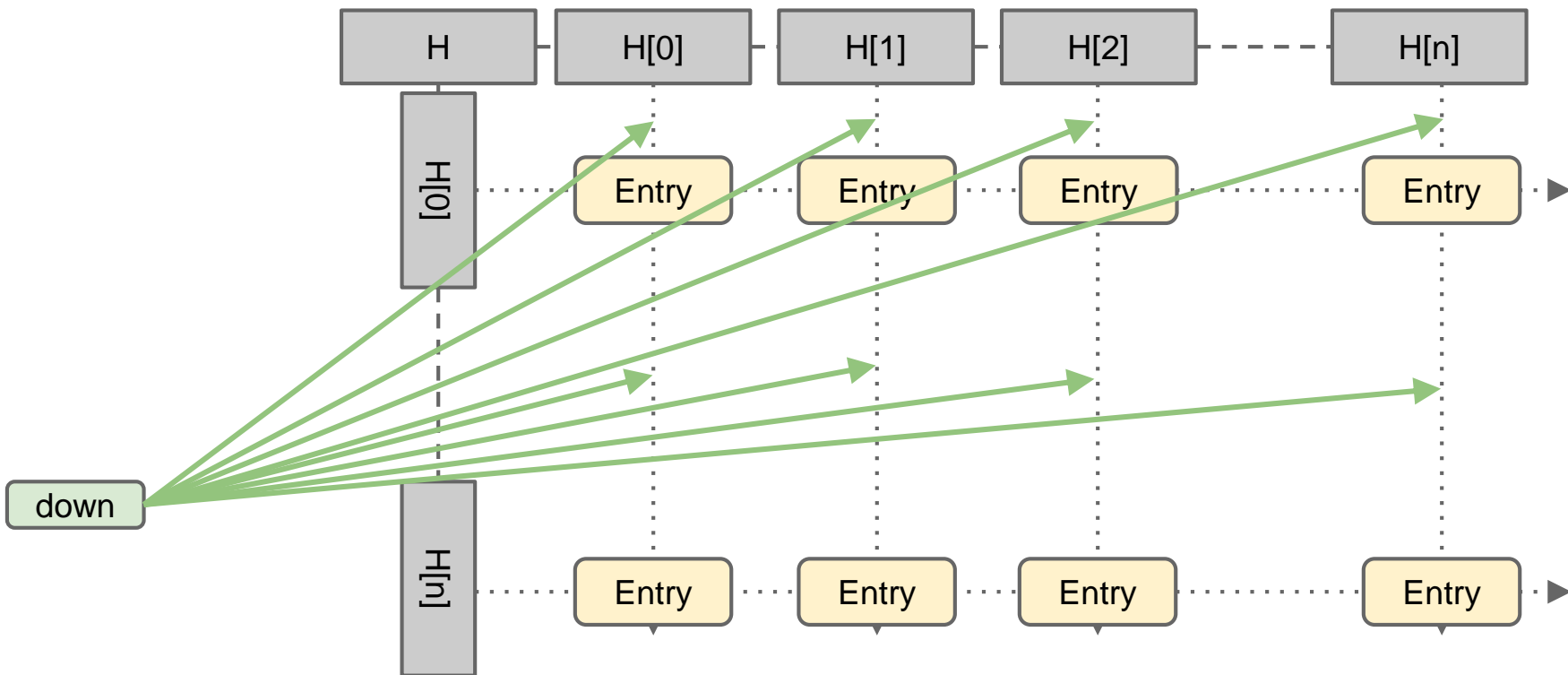


# Connection Outline

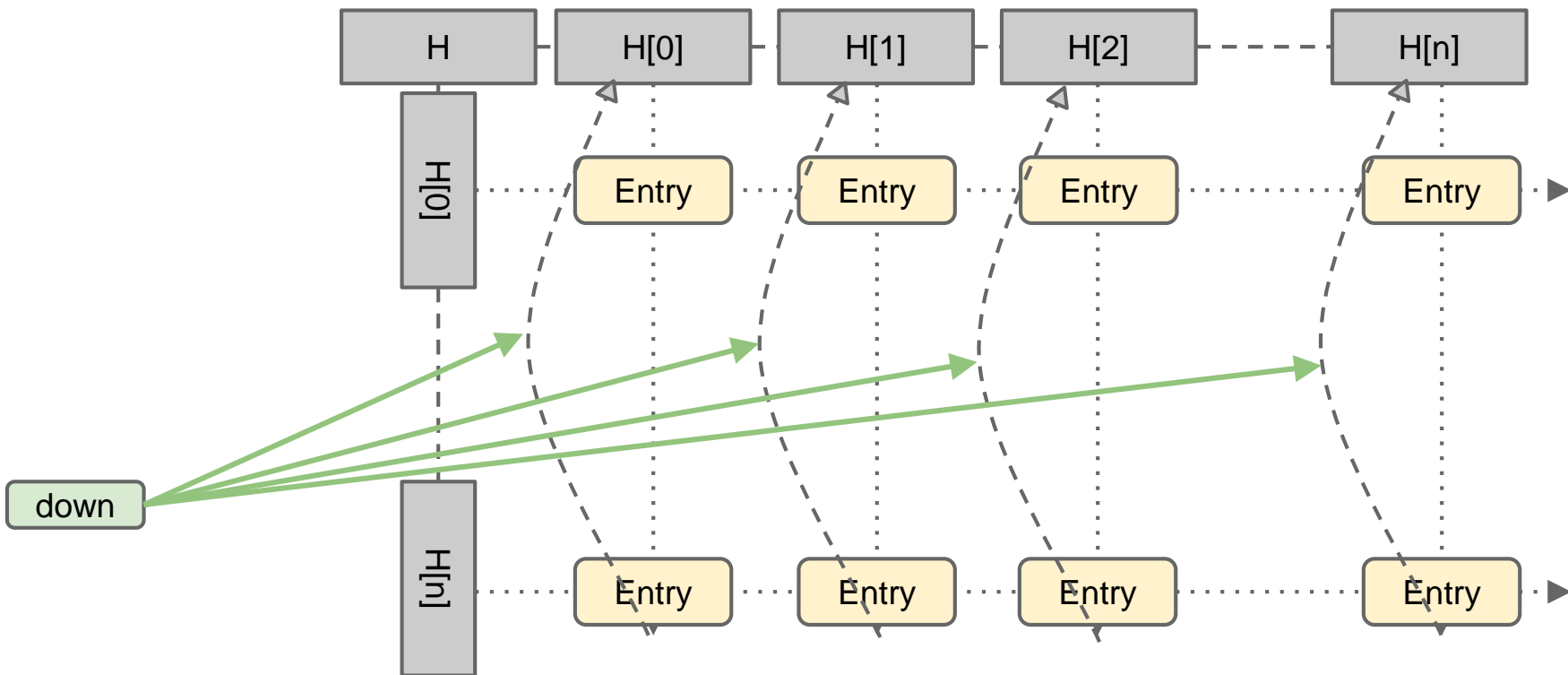


right

# Connection Outline



# Connection Outline



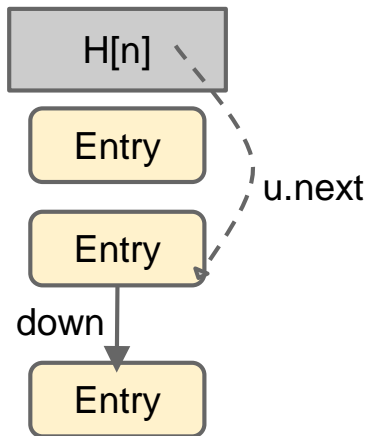
# Insert Data (Matrix Read)

1. row 와 col 중에 큰 값 찾기 → 사용될 head의 개수
2. 기본 Entry 구조 생성 → H 노드
3. H 노드에 행렬 기본 정보 저장 (row, col 크기)
4. Term 개수만큼 Head 노드 생성
5. 각 Term을 노드에 삽입
  - a. 같은 row인 경우 직전에 추가한 노드의 right에 붙임
  - b. 다른 row인 경우 다른 row에 해당하는 Head의 right에 붙임
  - c. col에 해당하는 Head 노드의 u.next를 임시 포인터 저장소로 사용하여 col들을 연결
6. 임시로 사용한 col에 해당하는 Head노드의 u.next의 down을 자기 자신으로 설정(Circular구조)
7. 각 Head의 u.next는 다음 Head로 연결
  - a. 마지막 Head의 u.next는 H 노드
  - b. H의 right는 첫번째 Head



# Insert Data Tricks

- Term 개수만큼 Head 노드 생성
  - row/col Index를 활용하기 위해 포인터 배열에 할당하여 사용
- col에 해당하는 Head 노드의 u.next를 임시 포인터 저장소로 사용
  - Entry가 추가될 때 마다 자기 바로 위에 있는 Entry의 down을 연결하기 위해 임시로 사용
  - 마지막에 u.next를  $H[n+1]$  노드에 연결



# Insert Data

```
matrixPointer mread(void)
{
    /* read in a matrix and set up its linked representation.
       An auxiliary global array hdnnode is used */
    int numRows, numCols, numTerms, numHeads, i;
    int row, col, value, current_row;
    matrixPointer temp, last, node;

    printf("Enter the number of rows, columns
           and number of nonzero terms: ");
    scanf("%d%d%d", &numRows, &numCols, &numTerms);
    numHeads = (numCols>numRows)? numCols : numRows;
    /* set up head node for the list of header nodes */
    node = new_node();    node->tag = entry;
    node->u.entry.row = numRows;
    node->u.entry.col = numCols;

    if (!numHeads) node->right = node;
    else { /* initialize the head nodes */
        for (i=0; i<numHeads; i++) {
            term= newNode();
            hdnnode[i] = temp;    hdnnode[i]->tag = head;
            hdnnode[i]->right = temp;    hdnnode[i]->u.next = temp;
        }
        current_row= 0;
        last= hdnnode[0]; /* last node in current row */
    }
}
```

```

for (i=0; i<numTerms; i++) {
    printf("Enter row, column and value:");
    scanf("%d%d%d", &row, &col, &value);
    if (row>currentRow) { /* close current row */
        last->right= hdnode[currentRow];
        currentRow= row; last=hdnode[row];
    }
    MALLOC(temp, sizeof(*temp));
    temp->tag=entry; temp->u.entry.row=row;
    temp->u.entry.col = col;
    temp->u.entry.value = value;
    last->right = temp; /*link into row list */
    last= temp;
    /* link to column list */
    hdnode[col]->u.next->down = temp;
    hdnode[col]=>u.next = temp;
}
/*close last row */
last->right = hdnode[currentRow];
/* close all column lists */
for (i=0; i<numCols; i++)
    hdnode[i]->u.next->down = hdnode[i];
/* link all head nodes together */
for (i=0; i<numHeads-1; i++)
    hdnode[i]->u.next = hdnode[i+1];
hdnode[numHeads-1]->u.next= node;
node->right = hdnode[0];
}
return node;
}

```

# Print Data

```
void mwrite(matrixPointer node)
{ /* print out the matrix in row major form */
    int i;
    matrixPointer temp, head = node->right;
    /* matrix dimensions */
    printf("\n num_rows = %d, num_cols= %d\n",
        node->u.entry.row,node->u.entry.col);
    printf("The matrix by row, column, and
        value:\n\n");
    for (i=0; i<node->u.entry.row; i++) {
        /* print out the entries in each row */
        for (temp=head->right;temp!=head;temp=temp->right)
            printf("%5d%5d%5d\n", temp->u.entry.row,
                temp->u.entry.col, temp->u.entry.value);
        head= head->u.next; /* next row */
    }
}
```

# Erase Data

```
void merase(matrixPointer *node)
{
    /* erase the matrix, return the nodes to the heap */
    matrixPointer x, y, head = (*node)->right;
    int i;
    /* free the entry and header nodes by row */
    for (i=0; i<(*node)->u.entry.row; i++) {
        y=head->right;
        while (y!=head) {
            x = y; y = y->right; free(x);
        }
        x= head; head= head->u.next; free(x);
    }
    /* free remaining header nodes */
    y = head;
    while (y!=*node) {
        x = y; y = y->u.next; free(x);
    }
    free(*node); *node = NULL;
}
```

1. row 기준으로 Entry 노드들 및 Head 노드를 할당 해제
  - a. row의 right에 있는 Entry 노드들
  - b. Circular 이므로 마지막 노드 다음 노드가 Head임을 활용
2. 노드가 남아있는 경우 (Col이 큰 경우) 남은 것들 제거
3. 최종 H 노드 제거

# Insert Data Tip

- 책에 있는 hdnode는 큰 의미 없음
  - 동적 할당한 matrixNode를 row 또는 col 값인 Index로 접근하기 위해 만든 임시 배열
  - 최종적으로는 Linked List 형태로만 연결이 되어있음
- 과제 샘플 Input의 경우 크기가 4096 by 4096이므로 MAX\_SIZE 4096 이상으로

# Sparse Matrix

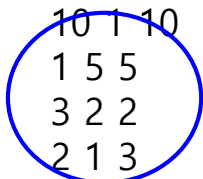
- 결과 값이 정렬 되어 있을 필요는 없음
  - ex)  
10 1 10  
1 5 5  
3 2 2  
2 1 3
- 단, 중복은 있으면 안됨
  - ex)  
1 5 1  
1 5 6  
2 2 3

# Sparse Matrix (input)

- 결과 값이 정렬 되어 있을 필요는 없음

◦ ex)

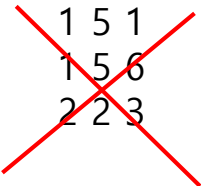
10 1 10  
1 5 5  
3 2 2  
2 1 3



- 단, 중복은 있으면 안됨

◦ ex)

~~1 5 1  
1 5 6  
2 2 3~~





# Lab 6: 결과 예

```
=====
1. Insert Matrix
2. Print Matrix
3. Erase Matrix
4. Exit Program
=====
===Select Menu ? 1
Read Matrix A
Enter num of rows, cols, nonzero terms : 5 5 3
Enter row col value : 1 2 3
Enter row col value : 2 3 4
Enter row col value : 3 4 5
=====
1. Insert Matrix
2. Print Matrix
3. Erase Matrix
4. Exit Program
=====
===Select Menu ? 2
Print matrix A
Matrix Information - Row Count : 5, Col Count : 5
1      2      3
2      3      4
3      4      5
```

## Insert

명령 입력

크기와 데이터 크기 입력

데이터 입력

## Print

# 제출 및 알림

수업 중 확인 or 메일제출 (이름, 학번, 소스코드)

메일 제출 :

주소 : (89kdsim@naver.com)

기한 : ~2016-04-27