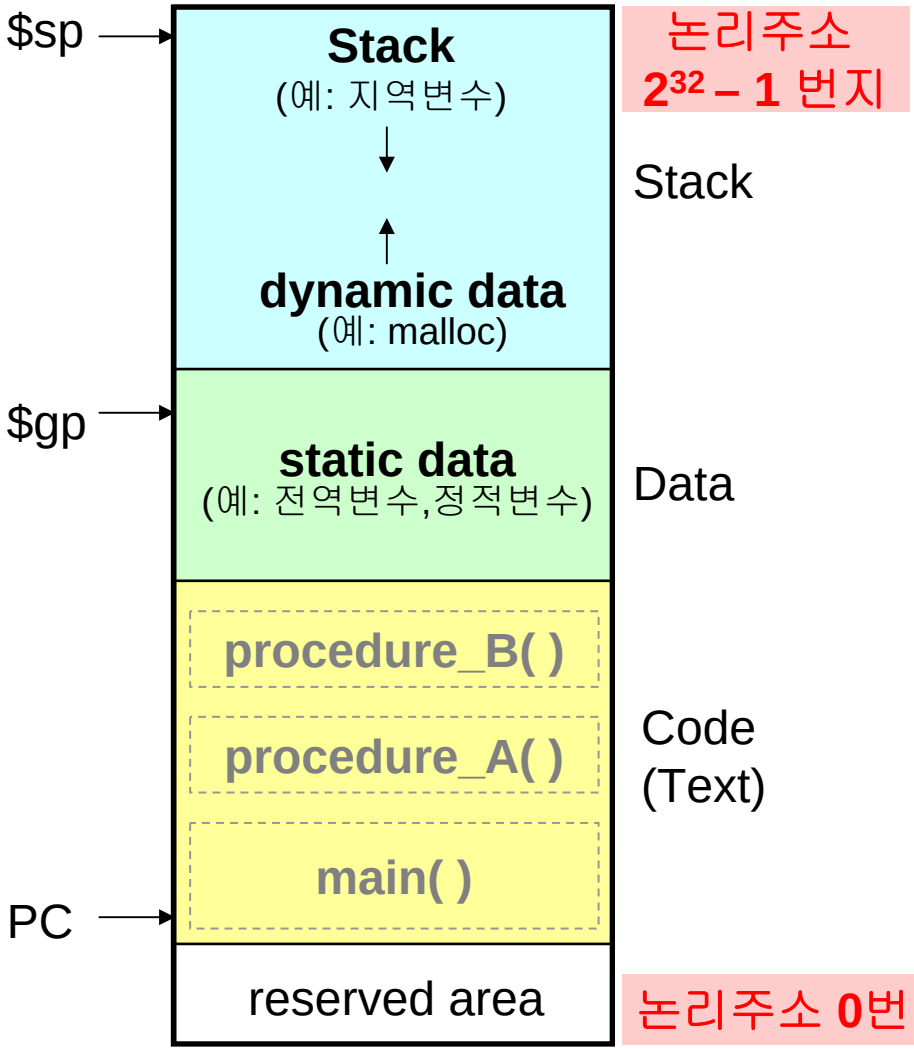
Procedure Call in MIPS

Procedure Call in Program Execution




```
int n=1;
int procedure_B(int x)
{     static int c=0;
      c++;
      return (x+c);
}
int procedure_A(int y)
{     int b = procudure_B(y);
      return (n+b);
}
void main( )
{     int *a = malloc(sizeof(int));
      *a = procedure_A(3);
       printf("%d", *a);
}
```

Procedure Call

- **What should be done?**
  1. 호출 이후에도 값이 유지되어야 하는 register의 저장
  2. return address의 저장
  3. argument의 전달

- **Anything else?**
  - 지역변수를 위한 공간 할당 필요
    (nested call을 지원할 수 있어야 함)

→ **stack**

Stack

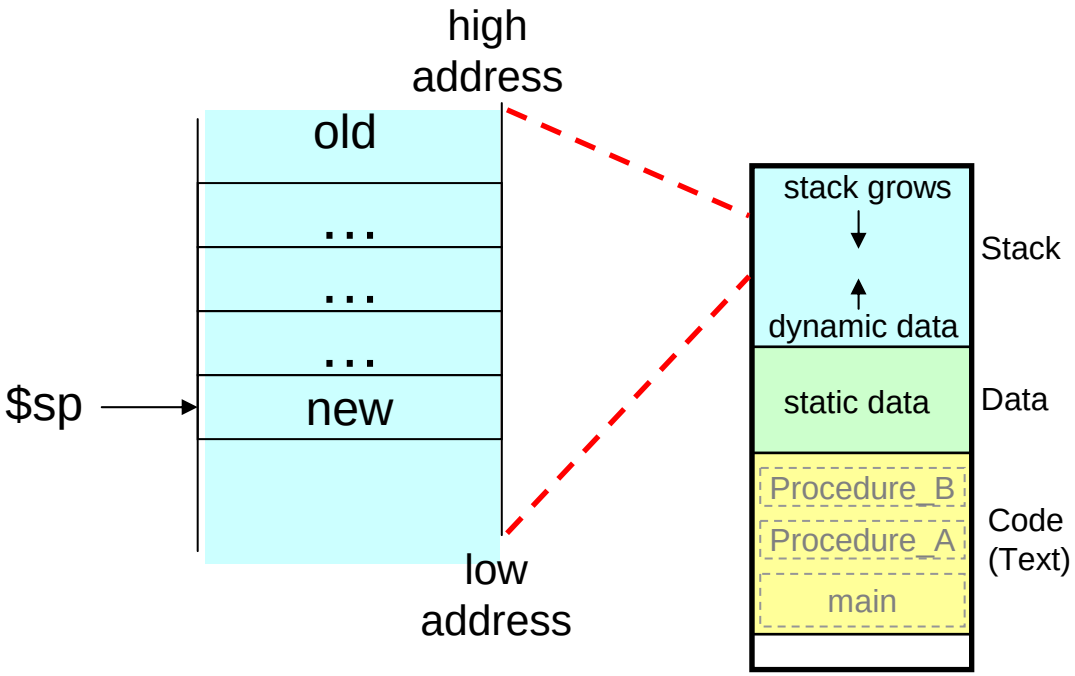
- **Last In First Out**
- **stack pointer ($sp)** 레지스터가 **stack**의 가장 최근 저장 위치를 가리킴

- **MIPS**의 **stack**
  - Stack에 정보 저장

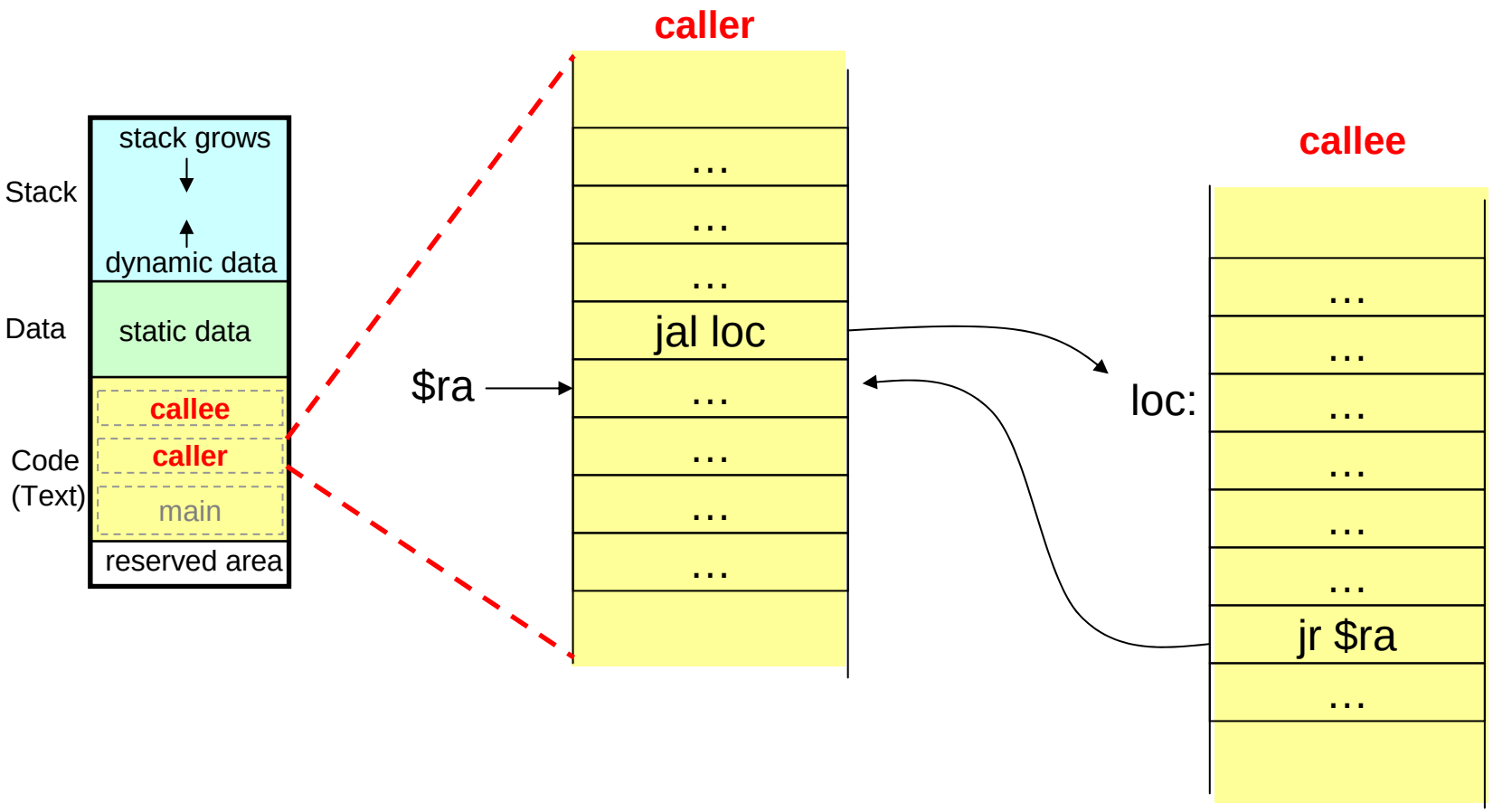
```
sub     $sp, $sp, 4
sw      $t0, 0($sp)
```

  - Stack에서 정보 인출

```
lw      $t0, 0($sp)
add     $sp, $sp, 4
```

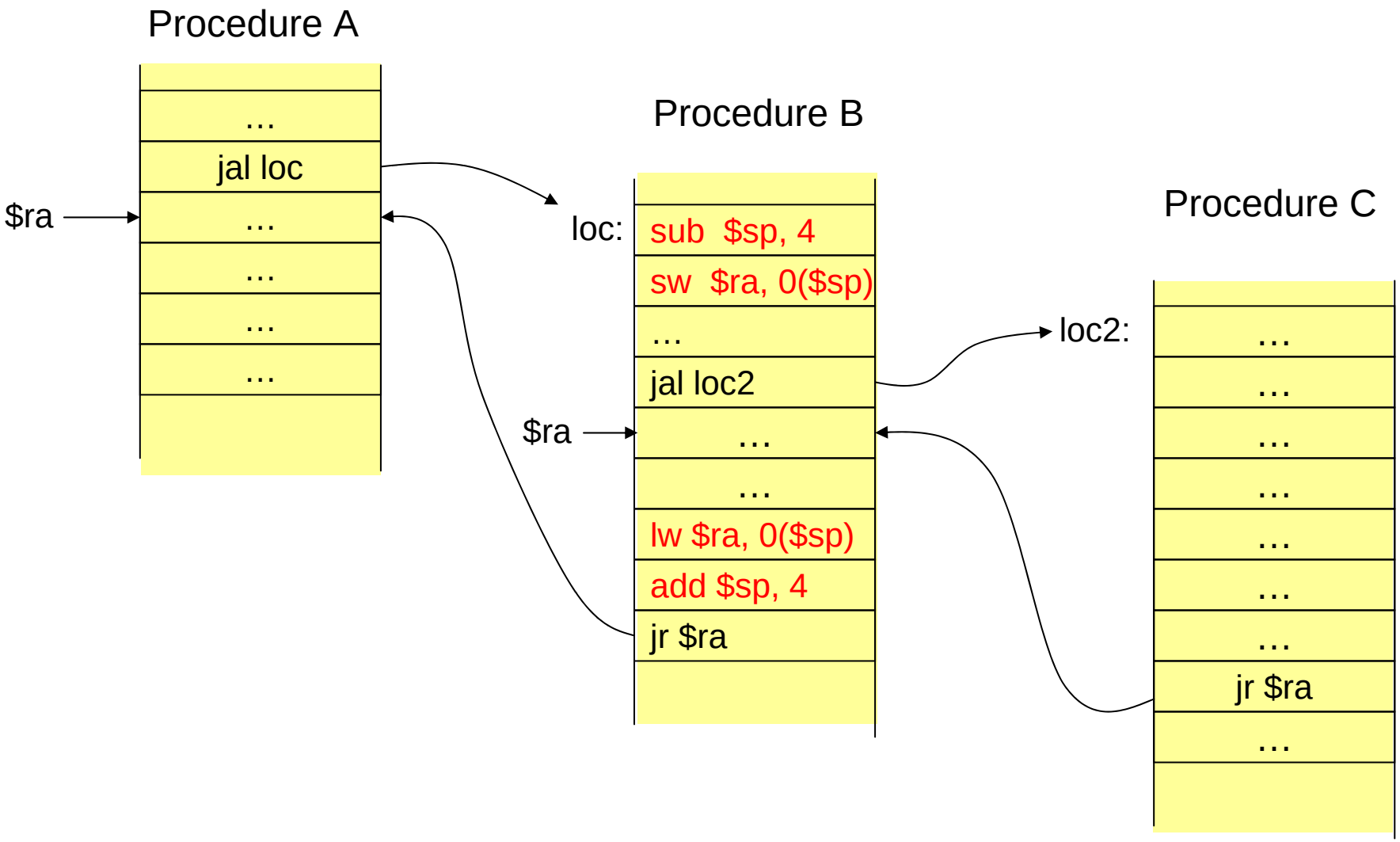


MIPS Procedure Call



Callee가 다른 procedure를 호출할 경우에는?

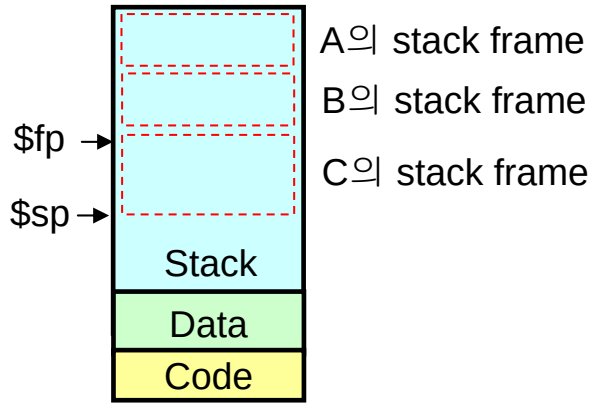
MIPS Procedure Call



Stack Frame

- **Stack frame**
  - procedure가 사용 중인 데이터를 stack 상의 자신의 frame에 저장
  - frame pointer ($fp)를 기본 위치로 하여 procedure가 그 데이터를 접근
  - stack은 expression evaluation의 용도로도 사용되므로 $sp는 procedure 사용 중에도 값이 변할 수 있음

- **stack frame**에 저장되는 내용
  - 전달되는 argument 중 4번째부터 (3개까지는 register a0~a3에 저장)
  - save된 register들의 값
  - 그 procedure의 local variable들



Procedure A → Procedure B→
Procedure C 호출

Register의 저장

- 변수들을 최대한 **register**에 **mapping**해서 사용
  - register는 memory보다 10배 이상 빠름
  - procedure 호출시 register를 새로운 지역변수에 mapping해서 사용
  - procedure 호출시 register의 기존 내용 저장 필요

- **callee save** 또는 **caller save**
  - MIPS의 경우
    - Callee-save register
      - ✓ Saved $s0~$s7
      - ✓ Frame pointer $fp
    - Caller-save register
      - ✓ Temporary $t0~$t9
      - ✓ Argument $a0~$a3
      - ✓ Return value $v0~$v1
      - ✓ Return address $ra

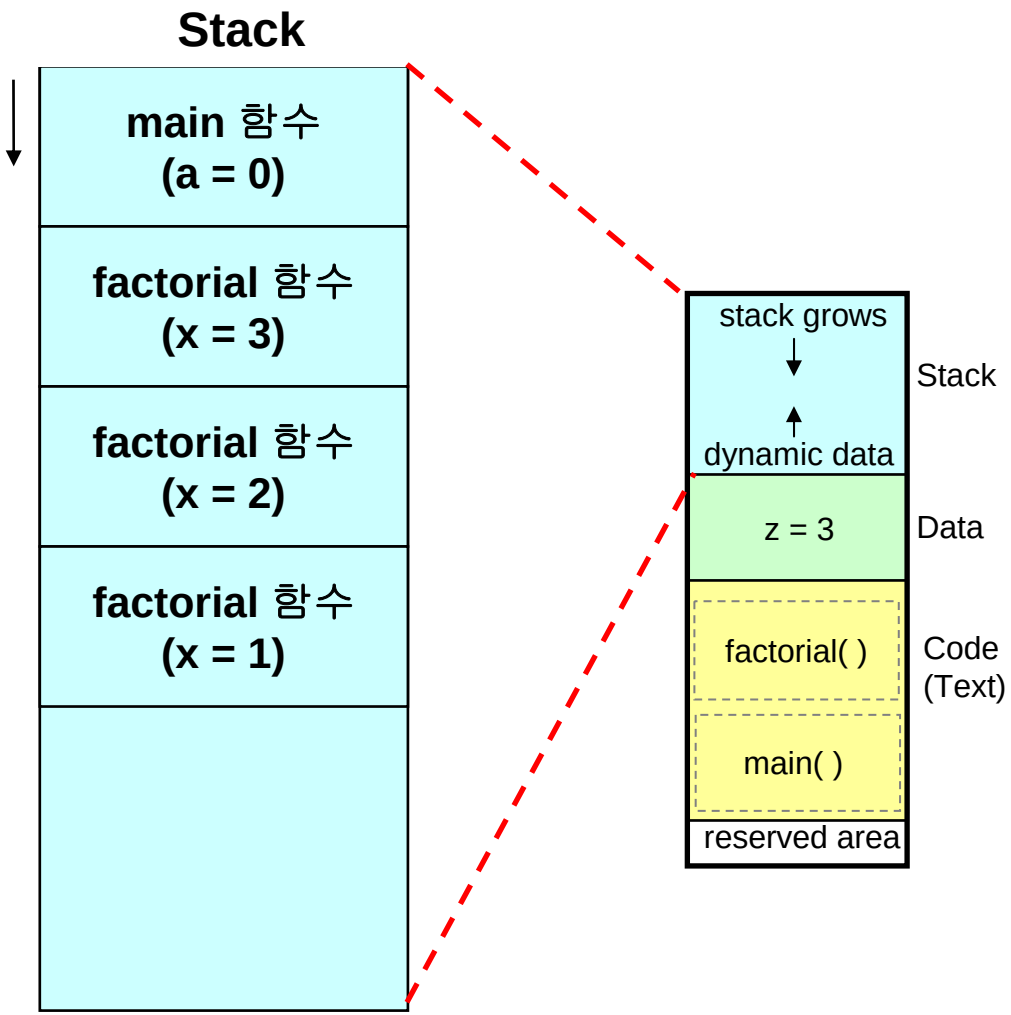
Procedure Call Actions

- **Caller**
  - caller-save register들 중 사용중인 것을 스택에 저장
    - $a0~$a3, $t0~$t9, $v0~$v1, $ra
  - 전달할 argument의 set up
    - 앞의 4개는 $a0~$a3에, 나머지는 stack에 저장
  - **jal** instruction을 이용해서 callee 호출
    - return address는 $ra에 자동 저장됨
- **Callee**
  - frame 크기를 계산하여 stack에 그 크기만큼의 공간 할당
    - $sp ← $sp – frame size
  - callee-save register 중 사용할 레지스터를 스택에 저장
    - $s0~$s7
    - $fp: 현 stack frame의 frame pointer를 가리켜야 하므로 save
  - $fp 값을 현재 stack frame의 시작 위치로 설정

Procedure Call Actions

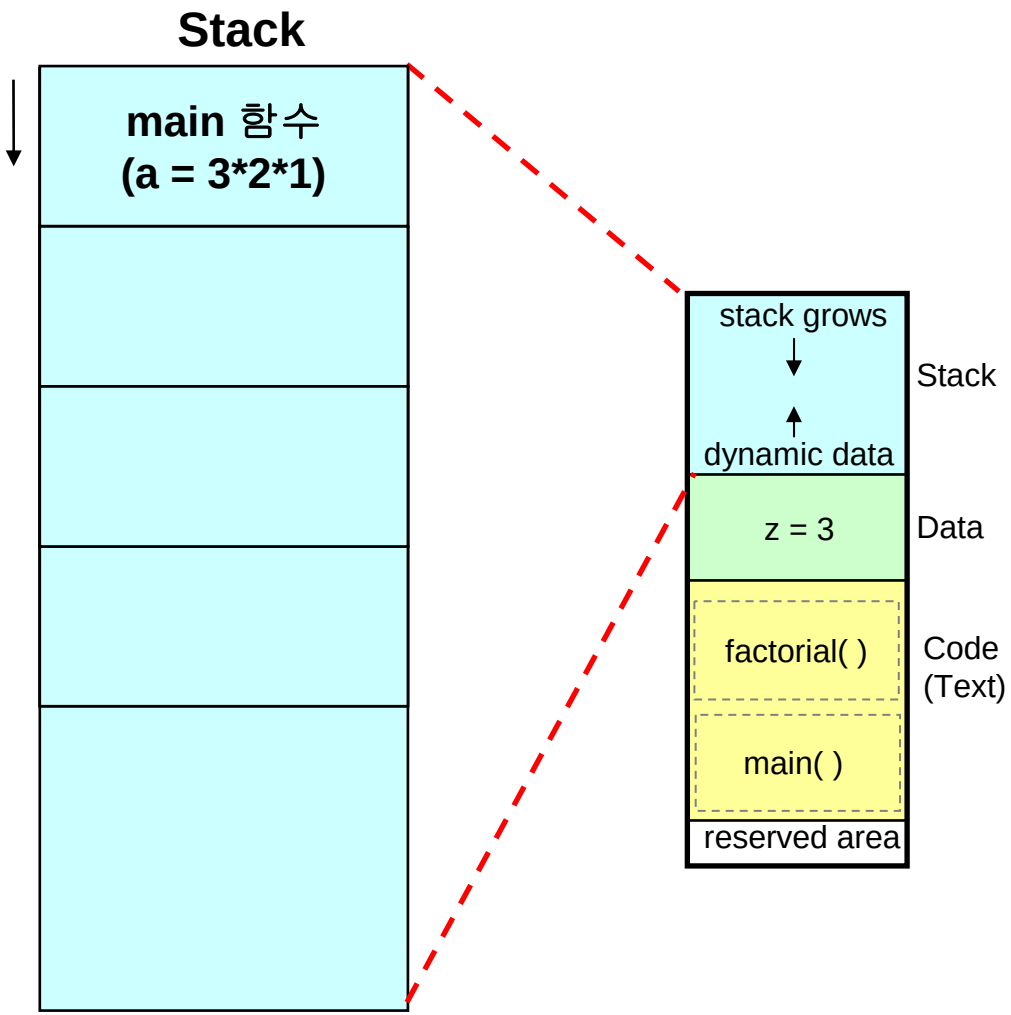
- **callee returns**
  - return value가 있는 경우 $v0~$v1에 그 값을 저장
  - 저장했던 callee-save register 값을 복원
    - $s0~$s7, $fp
  - stack에서 이 procedure의 frame을 비움
    - $sp ← $sp + frame size
  - jump $ra를 이용하여 return

Example of Procedure Call




```
int z = 3;
int factorial(int x)
{    if(x>1) x = x*factorial(x–1);
     return(x);
}
void main( )
{     int a=0;
       a = factorial(z);
        printf("%d", a);
}
```

Example of Procedure Call




```
int z = 3;
int factorial(int x)
{    if(x>1) x = x*factorial(x–1);
     return(x);
}
void main( )
{     int a=0;
       a = factorial(z);
        printf(“%d”, a);
}
```