

Function Overview (1A)

Copyright (c) 2010 - 2017 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice / LibreOffice.

3 Return Types of Functions

```
int func1( int a ) {  
    a *= 999;  
    return a;  
}
```

```
int func2( int a ) {  
    if (a < 0) return -a;  
    else      return a ;  
}
```

```
void func3( int a ) {  
    printf(“%d \n”, a) ;  
    // return;  
}
```

```
S = func1(100);  
S = func1(100);
```

```
S = func2(100);  
S = func2(100);
```

```
func3( 100 );
```

3 Return Types of Functions – Errors and Warnings

```
int func1( int a ) {  
    a *= 999;  
}
```

return val missing

```
S = func1(100);  
S = func1(100);
```

```
int func2( int a ) {  
    if (a < 0) -a;  
    else return a ;  
}
```

return val missing

```
S = func2(100);  
S = func2(100);
```

```
void func3( int a ) {  
    printf(“%d \n”, a) ;  
    return a ;  
}
```

return; must be used

```
func3( 100 );  
S = func3(100);
```

void : no return value

2 Passing Types of Functions

```
void val_func( int a ) {  
    X = a; // input  
    a = Y; // meaningless  
}
```

```
int m;  
val_func( m );
```


```
int *n;  
val_func( *n );
```

```
void ref_func( int *p ) {  
    X = *p; // input  
    *p = Y; // output  
}
```


```
int m;  
ref_func( &m );
```

```
int *n;  
ref_func( n );
```


Extra Outputs of Functions




```
void vfunc( int a ) {  
    X = a; // input  
    a = Y; // meaningless  
}
```



```
void rfunc( int *p ) {  
    X = *p; // input  
    *p = Y; // output  
}
```

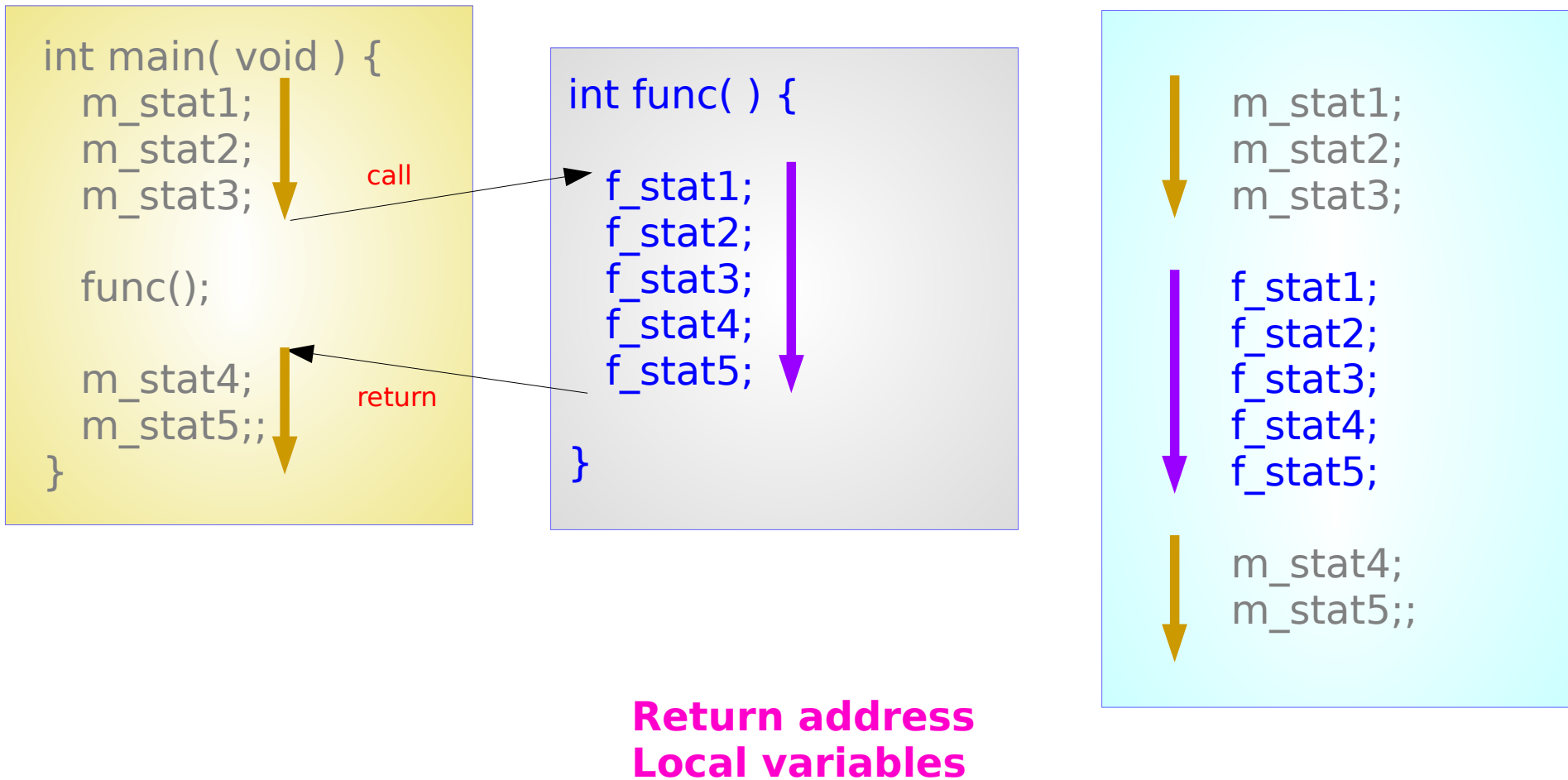


```
void rfunc( const int *p ) {  
    X = *p; // input  
    *p = Y; // prohibited  
}
```

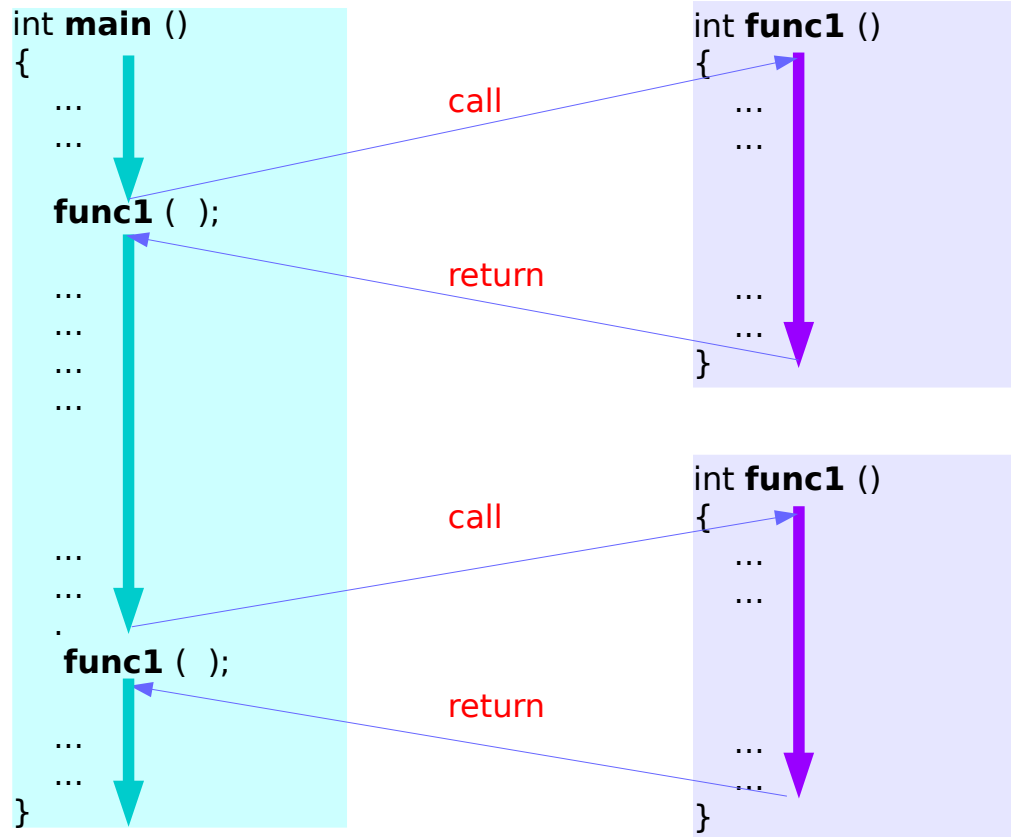


```
void rfunc( int *p ) {  
    *p = Y; // output  
}
```

Function Calls and Control Transfers

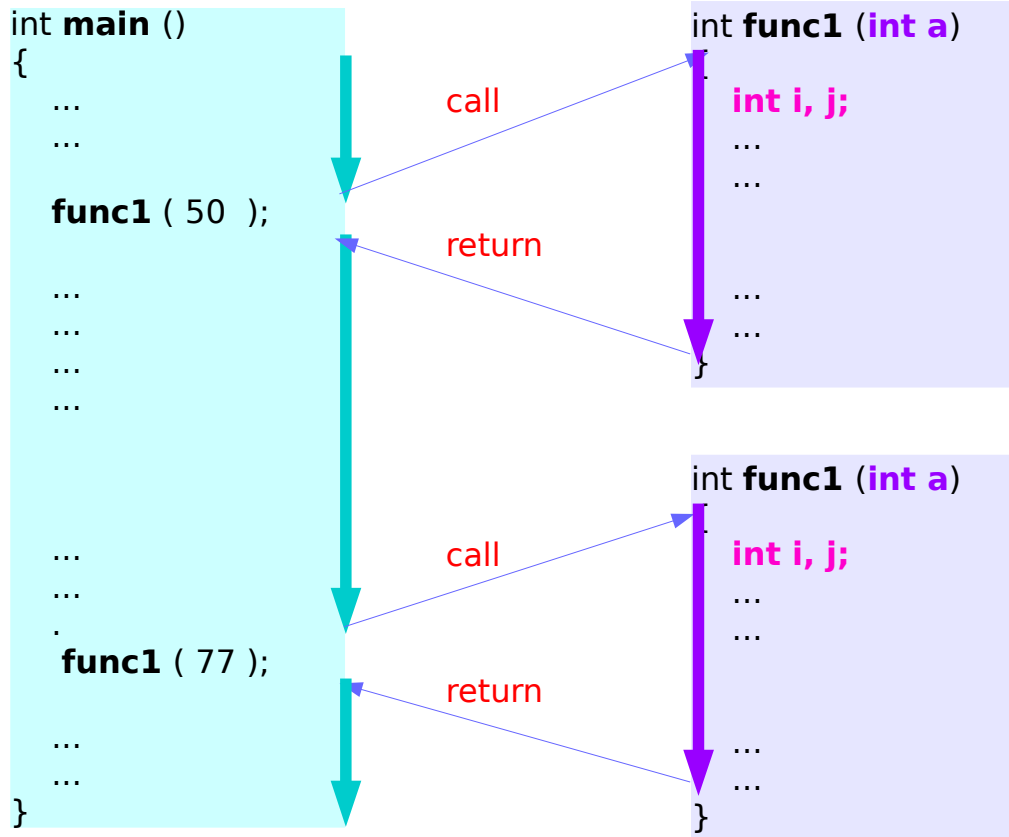


Return Address



each invocation of the same function can have different return addresses

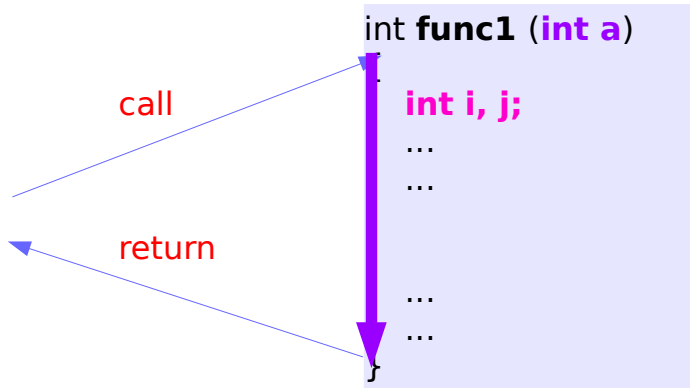
Local Variables



in each invocation of the same function the local variables usually have different values

These local variables are used only when the control is in the function. (when the function is active)

Local Variables



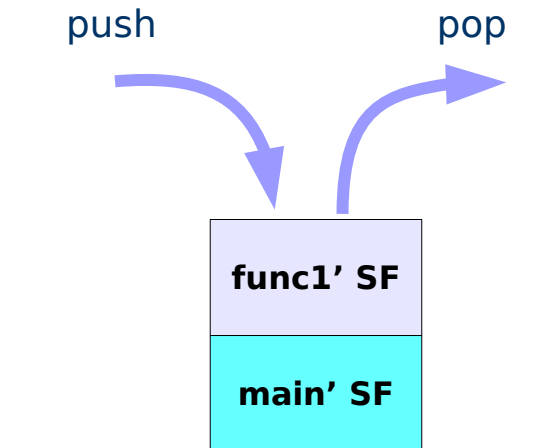
From the beginning and to the end of a function call (while the function is active)

- its return address
- its local variables

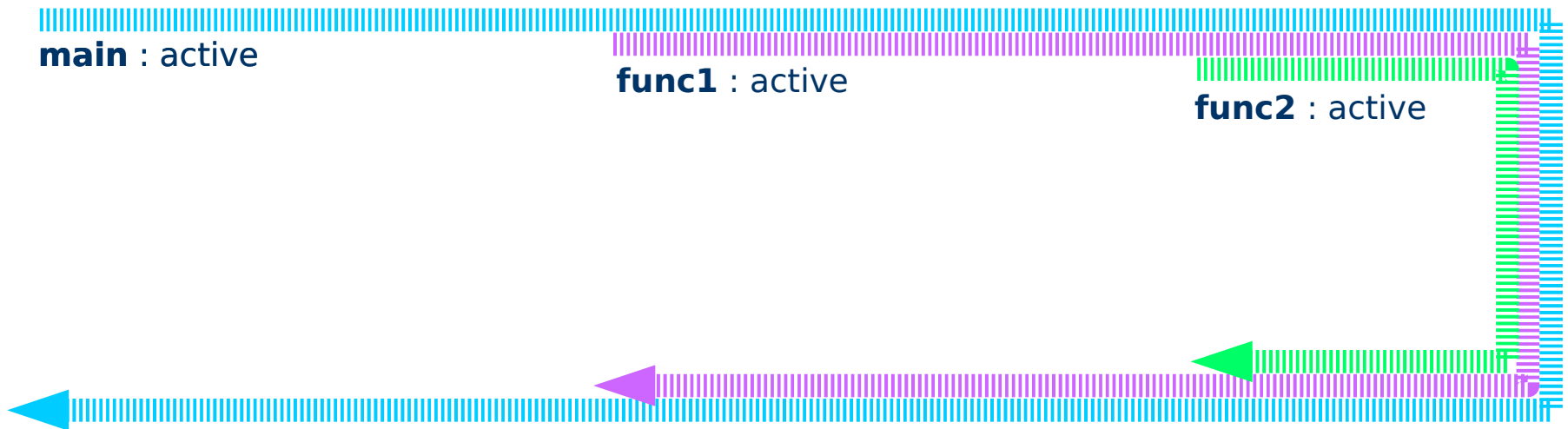
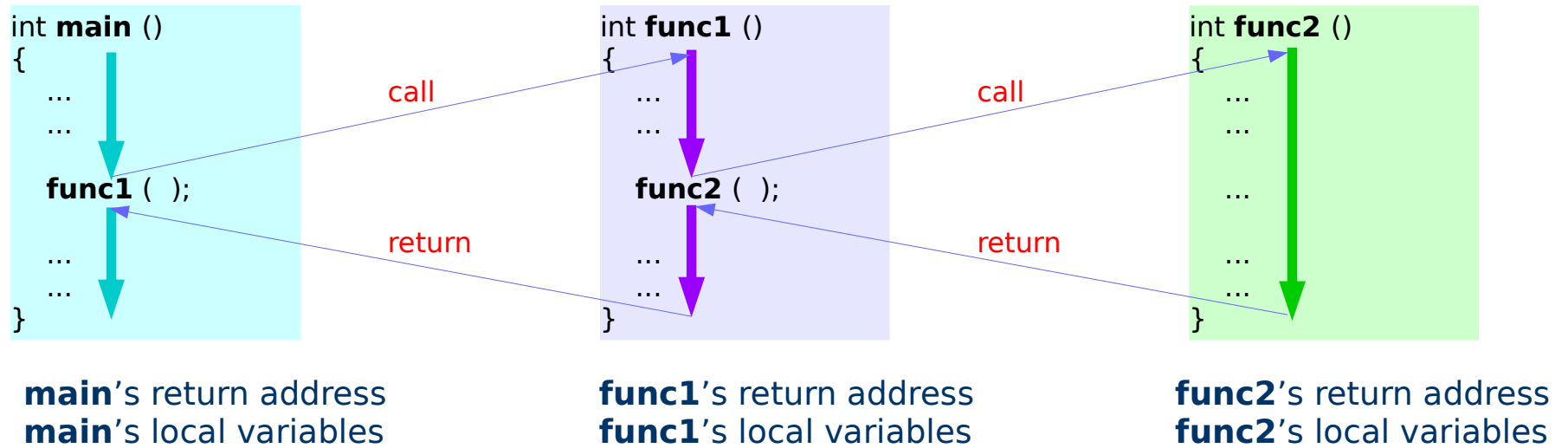
must be preserved

Each function has its own Stack Frame where each function store its return address and local variables

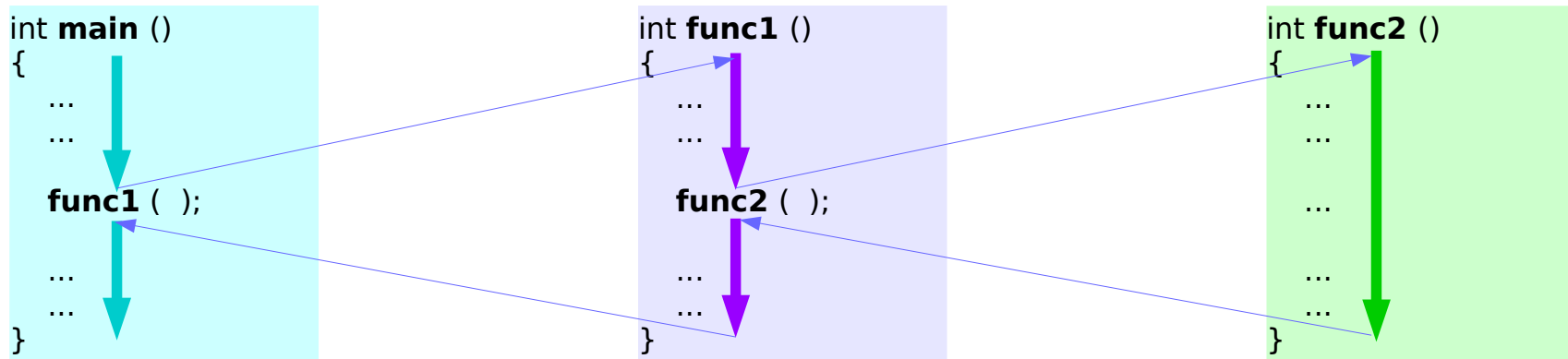
Stack Data Structure (Last In First Out)



Nested function calls



Nested function calls and stack frames

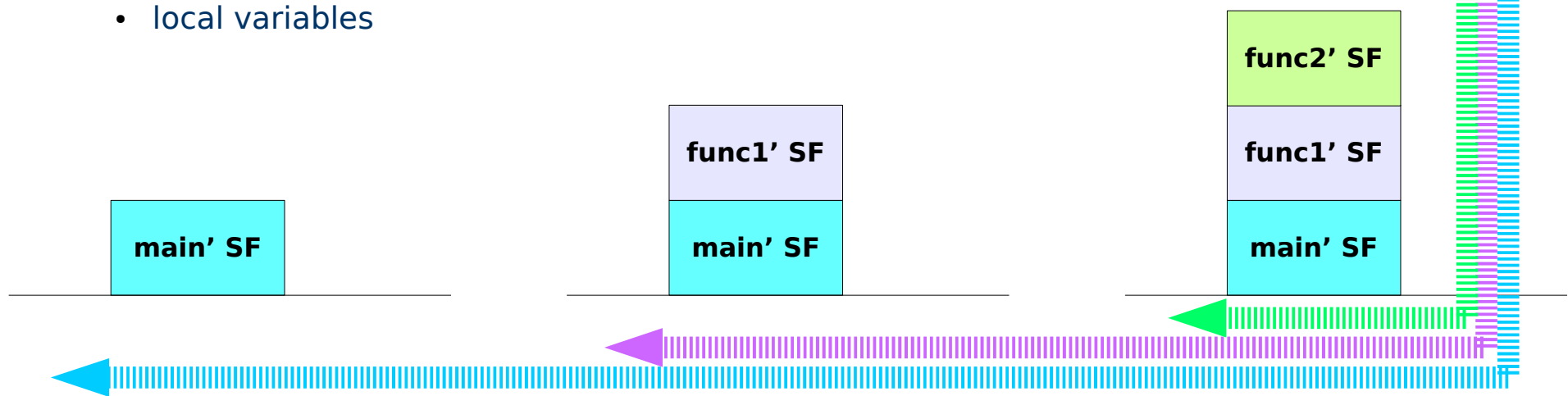


main : active

func1 : active

func2 : active

- return address
- local variables



Task: Finding 3 Partial Sums

$$S_n = \sum_{k=1}^n k$$

$$S_1 = \sum_{k=1}^1 k$$
$$S_2 = \sum_{k=1}^2 k$$
$$S_3 = \sum_{k=1}^3 k$$

$$S_1 = 1$$

```
printf("S1 = %d \n", S1);
```

$$S_2 = 1 + 2$$

```
printf("S2 = %d \n", S2);
```

$$S_3 = 1 + 2 + 3$$

```
printf("S3 = %d \n", S3);
```

Finding 3 Partial Sums – 3 for loops

$$S_1 = \sum_{k=1}^{\textcircled{1}} k = 1$$

$$S_2 = \sum_{k=1}^{\textcircled{2}} k = 1 + 2$$

$$S_3 = \sum_{k=1}^{\textcircled{3}} k = 1 + 2 + 3$$

```
S1 = 0;  
for (k=1; k<=1; ++k) S1 += k;
```

```
printf("S1 = %d \n", S1);
```

```
S2 = 0;  
for (k=1; k<=2; ++k) S2 += k;
```

```
printf("S2 = %d \n", S2);
```

```
S3 = 0;  
for (k=1; k<=3; ++k) S3 += k;
```

```
printf("S3 = %d \n", S3);
```

3 blocks with local variables

1 ⇒ n;

```
{ // block 1
  int n ⇐;
  int k, S = 0;
  for (k=1; k<=n; ++k) S += k;
}
```

S1 ⇐ S;

```
printf("S1 = %d \n", S1);
```

2 ⇒ n;

```
{ // block 2
  int n ⇐;
  int k, S = 0;
  for (k=1; k<=n; ++k) S += k;
}
```

S2 ⇐ S;

```
printf("S2 = %d \n", S2);
```

3 ⇒ n;

```
{ // block 3
  int n ⇐;
  int k, S = 0;
  for (k=1; k<=n; ++k) S += k;
}
```

S3 ⇐ S;

```
printf("S3 = %d \n", S3);
```

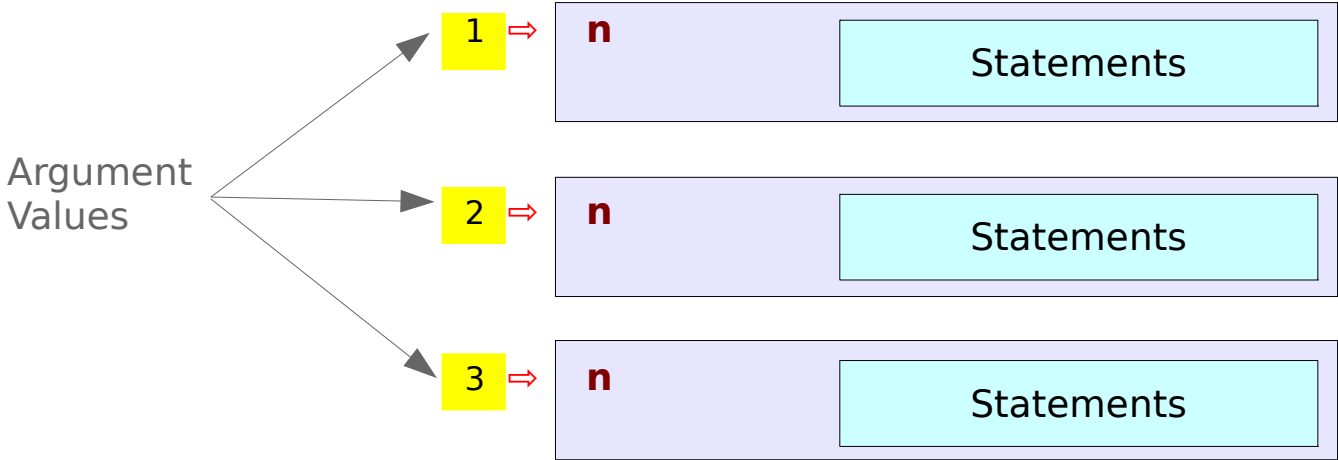
Local Variables

```
int n ;
int k, S = 0; X 3
```

the same named variables
with different values

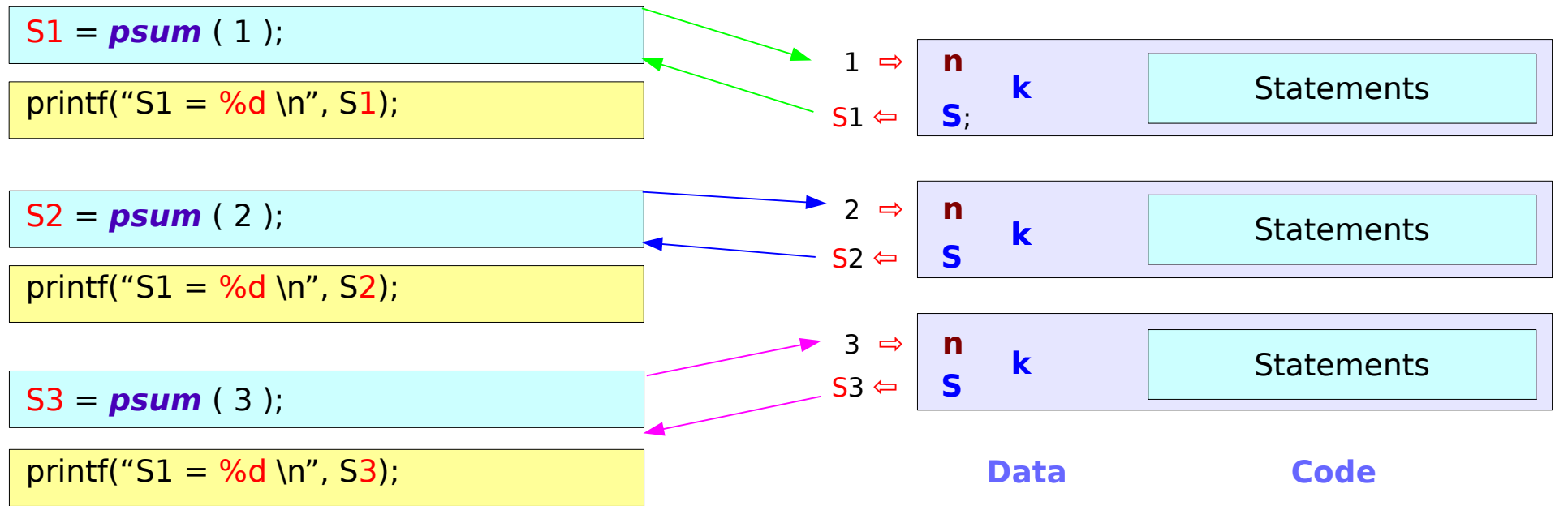
Arguments and Parameters

```
S1 = psum ( 1 );  
S2 = psum ( 2 );  
S2 = psum ( 3 );
```



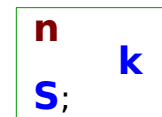
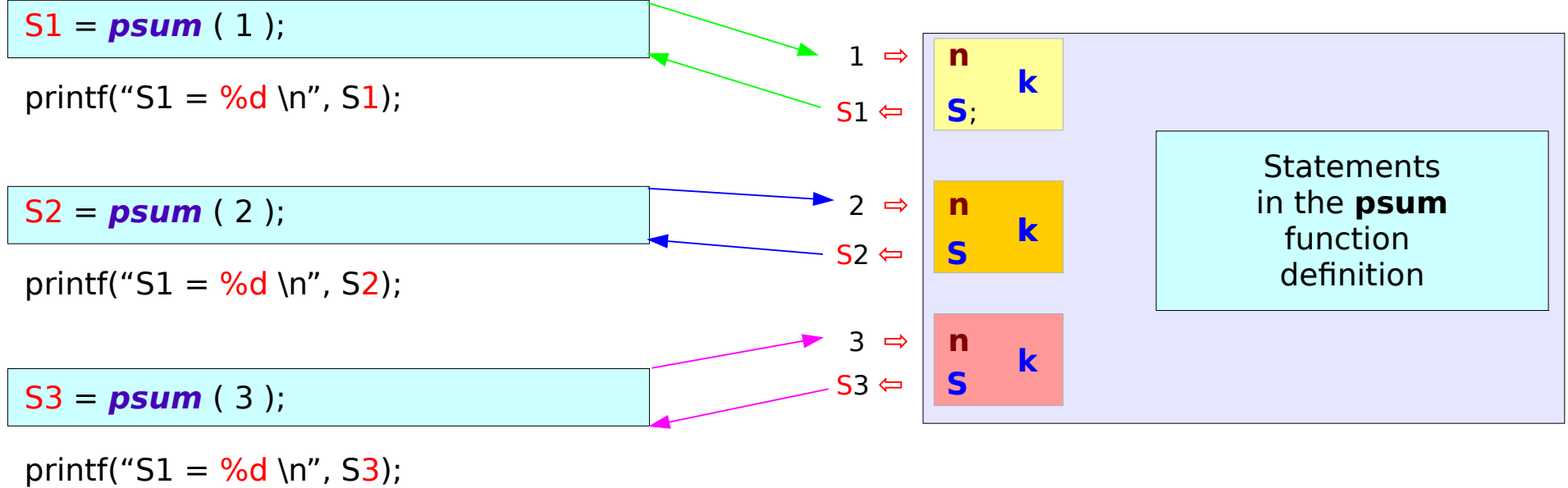
(formal)
**Parameter
Variable n**

Data and Code



Data	Code
different local variables	3 copies of statements
activation	redundancy
resides <u>stack</u> memory area	only <u>one</u> set resides on the memory

Local Variables



Active only when `psum()` is called

Function Prototype and Definition

```
█ → ;  
{  
    int n = input_n;  
    int k, S = 0;  
    for (k=1; k<=n; ++k) S += k;  
}  
← S;
```

Function Prototype

```
← int psum (int n) ;
```

Function Definition

```
int psum (int n)  
{  
    int k, S = 0;  
    for (k=1; k<=n; ++k) S += k;  
    return S;  
}
```

Function Prototype & Definition in a File

src1.c

```
int psum (int n) ;
```

```
int main (void)
```

```
{
```

```
    int S1, S2, S3;
```

```
    S1 = psum ( 1 );
```

```
    printf("S1 = %d \n", S1);
```

```
    S2 = psum ( 2 );
```

```
    printf("S2 = %d \n", S2);
```

```
    S3 = psum ( 3 );
```

```
    printf("S3 = %d \n", S3);
```

```
    return 0;
```

```
}
```

```
int psum (int n)
```

```
{
```

```
    int k, S = 0;
```

```
    for (k=1; k<=n; ++k) S += k;
```

```
    return S;
```

```
}
```

To inform the compiler that **psum** is the **name of a function** which has one integer type input and whose output type is integer

Since **psum** identifier is declared, **psum** can be used here.

What the function **psum** actually does is defined here.

```
gcc -o run src1.c
```

```
./run
```

Only Function Definition in a File

src2.c

```
int psum (int n)
{
    int k, S = 0;
    for (k=1; k<=n; ++k) S += k;
    return S;
}
```

```
int main (void)
{
    int S1, S2, S3;

    S1 = psum ( 1 );
    printf("S1 = %d \n", S1);
    S2 = psum ( 2 );
    printf("S2 = %d \n", S2);
    S3 = psum ( 3 );
    printf("S3 = %d \n", S3);

    return 0;
}
```

The function **psum** is defined here.

Since **psum** identifier is declared (actually the function is defined), **psum** can be used here.

```
gcc -o run src2.c
```

```
./run
```

One File Examples

src1.c

```
int psum (int n) ;
```

```
int main (void)
```

```
{  
    int S1, S2, S3;  
  
    S1 = psum ( 1 );  
    printf("S1 = %d \n", S1);  
    S2 = psum ( 2 );  
    printf("S2 = %d \n", S2);  
    S3 = psum ( 3 );  
    printf("S3 = %d \n", S3);  
  
    return 0;  
}
```

```
int psum (int n)
```

```
{  
    int k, S = 0;  
    for (k=1; k<=n; ++k) S += k;  
    return S;  
}
```

src2.c

```
int psum (int n)
```

```
{  
    int k, S = 0;  
    for (k=1; k<=n; ++k) S += k;  
    return S;  
}
```

```
int main (void)
```

```
{  
    int S1, S2, S3;  
  
    S1 = psum ( 1 );  
    printf("S1 = %d \n", S1);  
    S2 = psum ( 2 );  
    printf("S2 = %d \n", S2);  
    S3 = psum ( 3 );  
    printf("S3 = %d \n", S3);  
  
    return 0;  
}
```

Two File Examples

src3.c

```
int psum (int n) ;
```

```
int main (void)
```

```
{  
    int S1, S2, S3;  
  
    S1 = psum ( 1 );  
    printf("S1 = %d \n", S1);  
    S2 = psum ( 2 );  
    printf("S2 = %d \n", S2);  
    S3 = psum ( 3 );  
    printf("S3 = %d \n", S3);  
  
    return 0;  
}
```

src4.c

```
int psum (int n)
```

```
{  
    int k, S = 0;  
    for (k=1; k<=n; ++k) S += k;  
    return S;  
}
```

gcc -c src3.c → src3.o

gcc -c src4.c → src4.o

gcc -o run src3.o src4.o

./run

Header File Examples

src5.h

```
int psum (int n) ;
```

src5.c

```
#include "src4.h" ←
```

```
int main (void)
```

```
{  
    int S1, S2, S3;  
  
    S1 = psum ( 1 );  
    printf("S1 = %d \n", S1);  
    S2 = psum ( 2 );  
    printf("S2 = %d \n", S2);  
    S3 = psum ( 3 );  
    printf("S3 = %d \n", S3);  
  
    return 0;  
}
```

src6.c

```
int psum (int n)
```

```
{  
    int k, S = 0;  
    for (k=1; k<=n; ++k) S += k;  
    return S;  
}
```

```
gcc -c src5.c → src5.o
```

```
gcc -c src6.c → src6.o
```

```
gcc -o run src5.o src6.o
```

```
./run
```


Function Definitions (1)

```
int func1 (void)
{
}

```

```
int func2 (void)
{
}

```

```
int main (void)
{
}

```

functions are defined
outside the main function

Function Definitions (2)

```
int func1 (void)
```

```
{
```

```
int func3 (void)
```

```
{
```

```
}
```

```
}
```

```
int main (void)
```

```
{
```

```
int func2 (void)
```

```
{
```

```
}
```

```
}
```

Nested function definitions
are not allowed

References

- [1] Essential C, Nick Parlante
- [2] Efficient C Programming, Mark A. Weiss
- [3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
- [4] C Language Express, I. K. Chun