Imperative programming

6. Scope and life

Zoltán Porkoláb

Scope and life

- Scope – the area in the program where a name has the same meaning
  - Where can I use variable x ?
  - Where can I call function fahr2cels(double) ?
- Life – the time during execution while a memory area is valid
  - Can I use variable x after I left the block ?
- Scope and life are defined by the place (external, internal) of declaration and by the storage class

Scope

```
int i = 0;     // global, external linkage
static int k; // global, internal linkage
extern int n; // global, external linkage, defined elsewhere

static int g(int par) // global, internal linkage
{
  i = par;    // global i
  return i;
}
void f(void)  // global, external linkage
{
  int i = 1;  // local, internal linkage, hides global i
  static int j = 2;  // local, internal linkage
  {
    int i = 3;// local, internal linkage, hides outer local i
    j = 4;    // outer local j
    k = 5;    // global static k
    n = g(i); // global g, inner local i, global n def.elsewhere
  }
  i = 7;       // local i
}
```

Scope of global k

```
int i = 0;     // global, external linkage
static int k; // global, internal linkage
extern int n; // global, external linkage, defined elsewhere

static int g(int par) // global, internal linkage
{
  i = par;     // global i
  return i;
}
void f(void)  // global, external linkage
{
  int i = 1;  // local, internal linkage, hides global i
  static int j = 2;  // local, internal linkage
  {
    int i = 3;// local, internal linkage, hides outer local i
    j = 4;     // outer local j
    k = 5;     // global static k
    n = g(i); // global g, inner local i, global n def.elsewhere
  }
  i = 7;         // local i
}
```

Scope of g()

```
int i = 0;     // global, external linkage
static int k; // global, internal linkage
extern int n; // global, external linkage, defined elsewhere

static int g(int par) // global, internal linkage
{
  i = par;    // global i
  return i;
}
void f(void)  // global, external linkage
{
  int i = 1;  // local, internal linkage, hides global i
  static int j = 2;  // local, internal linkage
  {
    int i = 3;// local, internal linkage, hides outer local i
    j = 4;    // outer local j
    k = 5;    // global static k
    n = g(i); // global g, inner local i, global n def.elsewhere
  }
  i = 7;       // local i
}
```

Scope of par in g()

```
int i = 0;     // global, external linkage
static int k; // global, internal linkage
extern int n; // global, external linkage, defined elsewhere

static int g(int par) // global, internal linkage
{
  i = par;     // global i
  return i;
}
void f(void)  // global, external linkage
{
  int i = 1;  // local, internal linkage, hides global i
  static int j = 2;  // local, internal linkage
  {
    int i = 3;// local, internal linkage, hides outer local i
    j = 4;     // outer local j
    k = 5;     // global static k
    n = g(i); // global g, inner local i, global n def.elsewhere
  }
  i = 7;         // local i
}
```

Scope of local i

```
int i = 0;     // global, external linkage
static int k; // global, internal linkage
extern int n; // global, external linkage, defined elsewhere

static int g(int par) // global, internal linkage
{
  i = par;     // global i
  return i;
}
void f(void)  // global, external linkage
{
  int i = 1;  // local, internal linkage, hides global i
  static int j = 2;  // local, internal linkage
  {
    int i = 3;// local, internal linkage, hides outer local i
    j = 4;     // outer local j
    k = 5;     // global static k
    n = g(i); // global g, inner local i, global n def.elsewhere
  }
  i = 7;         // local i
}
```

Scope of inner local i

```
int i = 0;     // global, external linkage
static int k; // global, internal linkage
extern int n; // global, external linkage, defined elsewhere

static int g(int par) // global, internal linkage
{
  i = par;     // global i
  return i;
}
void f(void)  // global, external linkage
{
  int i = 1;  // local, internal linkage, hides global i
  static int j = 2;  // local, internal linkage
  {
    int i = 3;// local, internal linkage, hides outer local i
    j = 4;     // outer local j
    k = 5;     // global static k
    n = g(i); // global g, inner local i, global n def.elsewhere
  }
  i = 7;         // local i
}
```

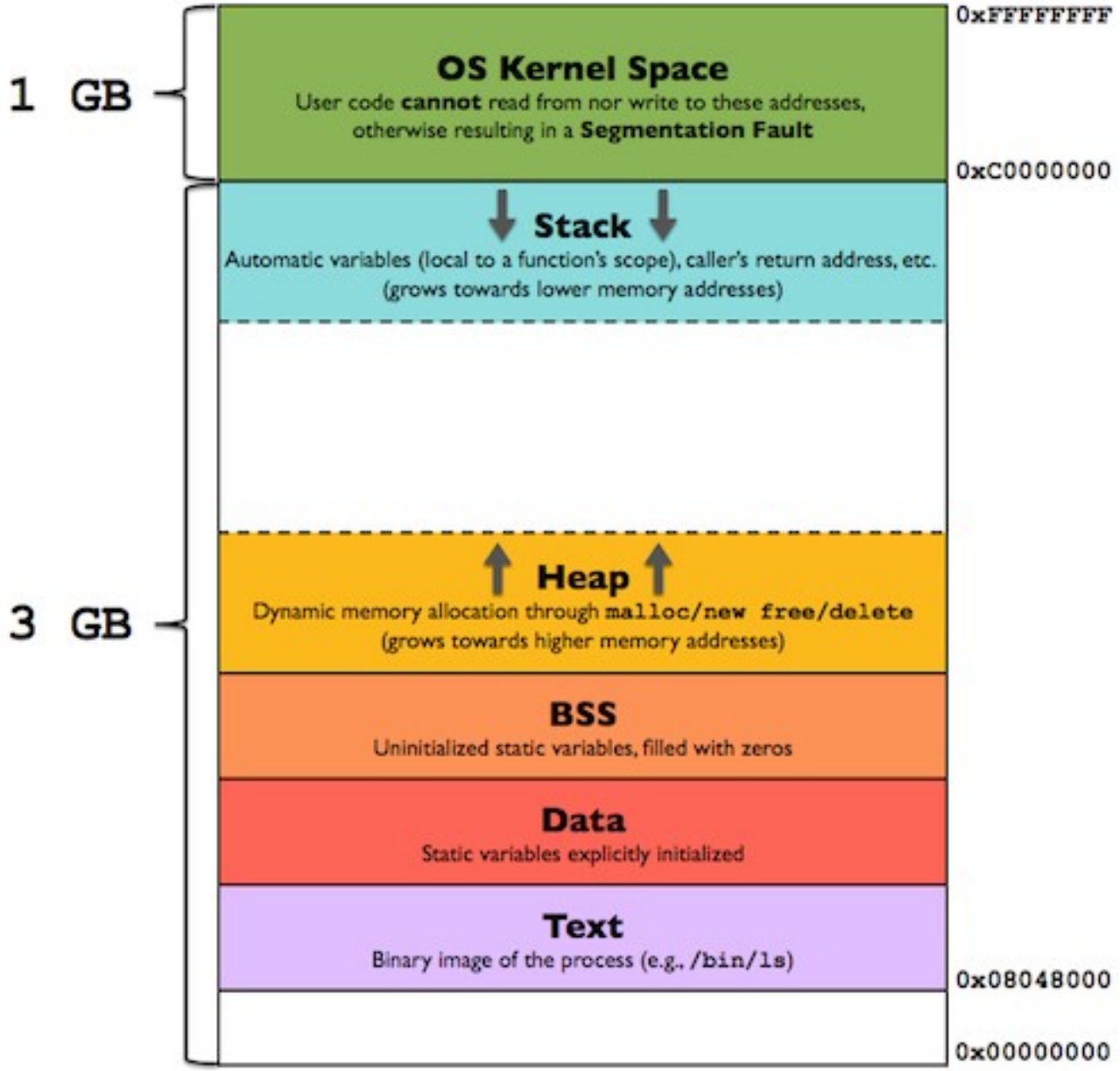
Scope of global i

```
int i = 0;     // global, external linkage
static int k; // global, internal linkage
extern int n; // global, external linkage, defined elsewhere

static int g(int par) // global, internal linkage
{
  i = par;     // global i
  return i;
}
void f(void)  // global, external linkage
{
  int i = 1;  // local, internal linkage, hides global i
  static int j = 2;  // local, internal linkage
  {
    int i = 3;// local, internal linkage, hides outer local i
    j = 4;     // outer local j
    k = 5;     // global static k
    n = g(i); // global g, inner local i, global n def.elsewhere
  }
  i = 7;         // local i
}
```

Life

- For memory used during the execution of program
- From now until when is valid to use a memory area
  - Accessing a memory area out of lifetime is a serious error
  - Can cause wrong program behavior or crash
- Lifetime categories in C
  - Static – from the beginning of the program until the end
  - Automatic – during the execution of a block
  - Dynamic – the user request the memory and the user give it back

Typical memory layout



Static lifetime

- Memory is allocated in the **global area**
- Memory allocated at the start of the program
- Available during the whole program execution
- Deallocated at the end of the program
- if not initialized explicitly static variables are zero initialized
- Typical for
  - Global variables with external linkage
  - Global static variables with internal linkage
  - Local static variables (initialized only the first time)

Static lifetime

```
int i = 42;   // global variable, static lifetime, init 42
static int k; // global variable, static lifetime, init 0
extern int n; // global variable, static lifetime

void f(void)  // global, external linkage
{
  static int cnt = 0;  // local variable, static lifetime
                       // initialized only once, first time
  if ( 0 == ++cnt % 10 )
  {
    printf( "f() is called %d times\n", cnt);
  }
}
int main()
{
  for ( int i = 0; i < 100000; ++i)
  {
    f();
    printf( "%d",cnt);  // cnt lives here but not visible
  }
}
```

Automatic lifetime

- Memory is allocated in the **stack**
- Begins when the execution reaches the definition in a block
- Ends when the execution leaves the block (and we loose the value)
- Initialization happens each time we execute the block
- If not initialized the value is **undefined** and error to read the variable
- Lifetime is valid when we execute inner blocks of called functions
- Typical for
  - Local not-static variable

Automatic lifetime

```
int  f(void)  // global, external linkage
{
  int j = 42; // initialized every time we are here

  ++j;    // ok, 43

  return j;  // ok 43 returns
}    // lifetime of j ends here

int main()
{
  int i = f();           // ok, f() returns 43
  printf( "%d\n", i);    // prints 43
}
```

Automatic lifetime

```
int *f(void)  // global, external linkage
{
  int i;  // local variable, uninitialized
  int j = 42; // initialized every time we are here

  ++i;    // undefined behavior, i is uninitialized
  ++j;    // ok, 43

  return &j;  // error, j is out of lifetime when f returns
}

int main()
{
  int *ptr = f();  // illegal, j is out of lifetime
  printf( "%d\n", *ptr);   // likely run-time error
}
```

Dynamic lifetime

- Memory is allocated in the **free store** (heap)
- User requests memory by **malloc( )** or **realloc( )**
- Size parameter is in unit of **bytes**, and should be greater than 0
- If allocation is unsuccessful, NULL pointer returns
- **Always check the return value of malloc() and realloc( ) !**
- User must deallocate the memory by **free( )**
- Realloc may deallocate old memory and returns new (copying the content) or extend the existing buffer

Dynamic lifetime

```
// read n numbers and print them in reverse order
int  reverse(void)
{
  int n;
  scanf("%d", &n);  // read n from stdin

  int *buf = (int *)malloc(n*sizeof(int)); // try to allocate int[n]

  if ( NULL == buf )  // no memory
  {
    fprintf( stdout, "No memory\n");
    return 0;  // return false
  }
  for (int i = 0; i < n; ++i)
  {
    scanf("%d", &buf[i]);   // read elements from stdin
  }
  for (int i = n-1; i >= 0; --i)
  {
    printf("%d, ", buf[i]); // print elements in reverse order
  }
  free(buf); // deallocate heap memory
  return 1;  // return true
}
```

Dynamic lifetime

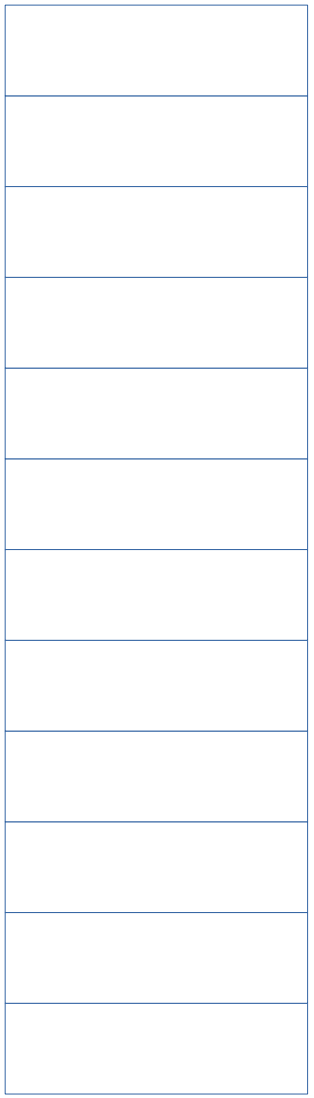
```
int  f(void)
{
  int ch;
  int i = 0;
  int capacity = 16;
  char *buf = (char *)malloc(capacity); // try to allocate char[16]

  if ( NULL == buf )  // allocation unsuccessful
  {
    fprintf(stderr,"no memory");
    return -1;
  }
  while ( EOF != (ch = getchar()) )
  {









    buf[i++] = ch;   // read n integers into dynamic memory
  }
  // do something with buf                          and free!!!
}
```

Dynamic lifetime

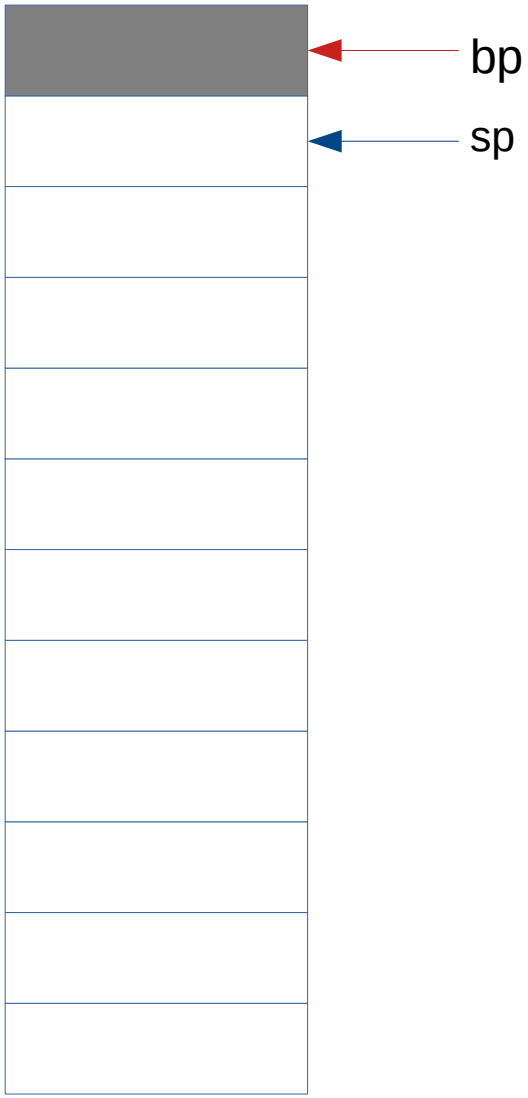
```
int  f(void)
{
  int ch;
  int i = 0;
  int capacity = 16;
  char *buf = (char *)malloc(capacity); // try to allocate char[16]

  if ( NULL == buf )  // allocation unsuccessful
  {
    fprintf(stderr,"no memory");
    return -1;
  }
  while ( EOF != (ch = getchar()) )
  {
    if ( i == capacity ) // if buffer is full, grow buffer
    {
      char *newbuf = (char *)realloc(buf,2*capacity);  // duplicate the size
      if ( newbuf )  // reallocation s successful
      {
         capacity *= 2;
         buf = newbuf;  // newbuf is not necessary is the same as buf
      }
      else  // reallocation is unsuccessful, but the old memory is still valid
      {
         break;   // buf is valid
      }
    }
    buf[i++] = ch;   // read n integers into dynamic memory
  }
  // do something with buf                                  and free!!!
}
```

C stack

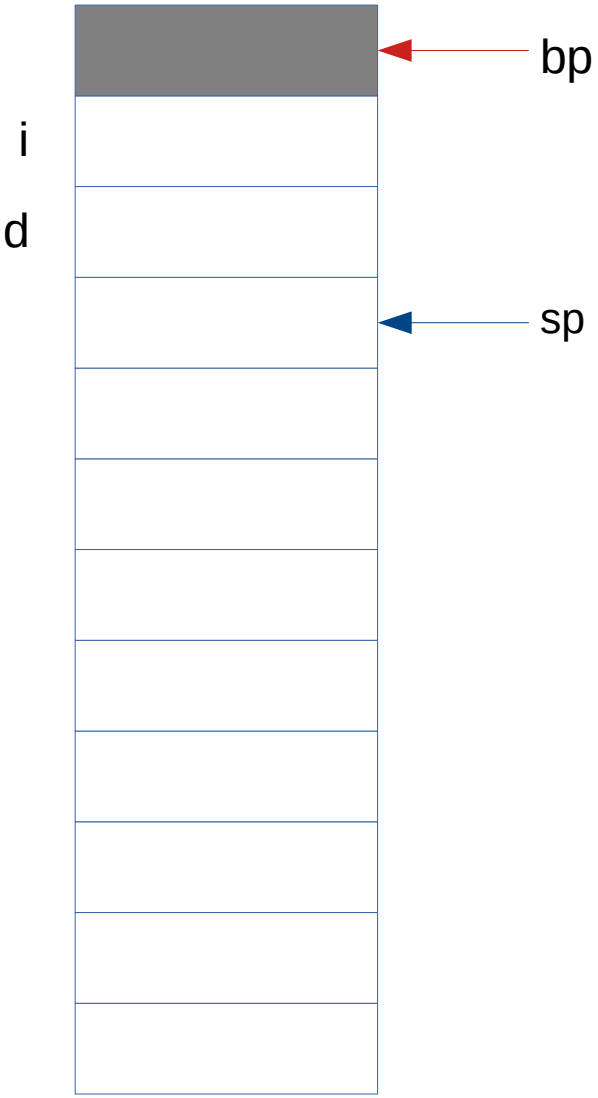
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



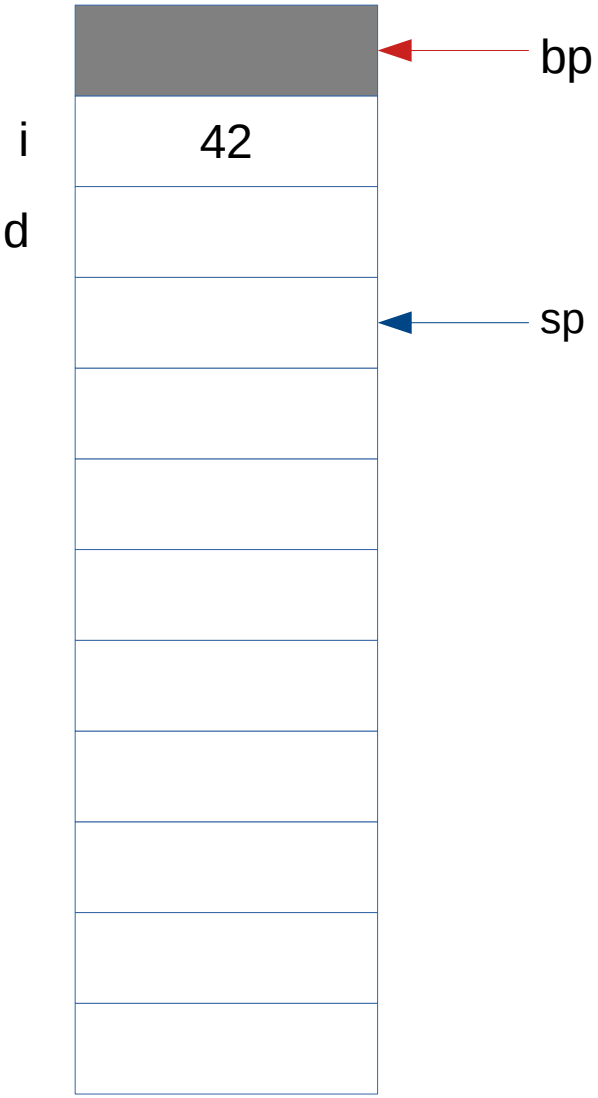
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



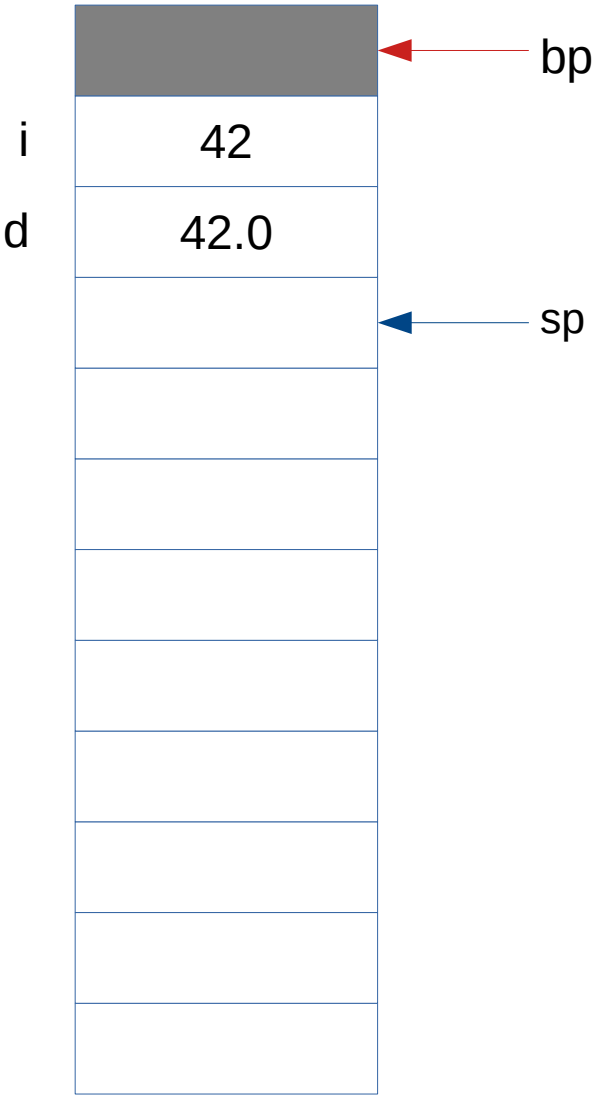
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



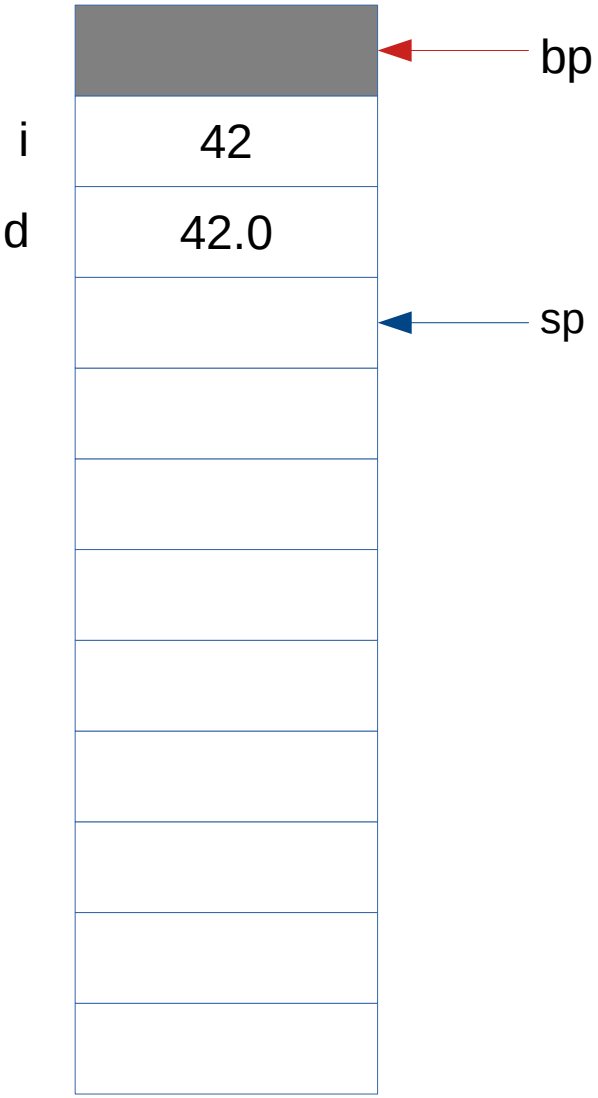
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



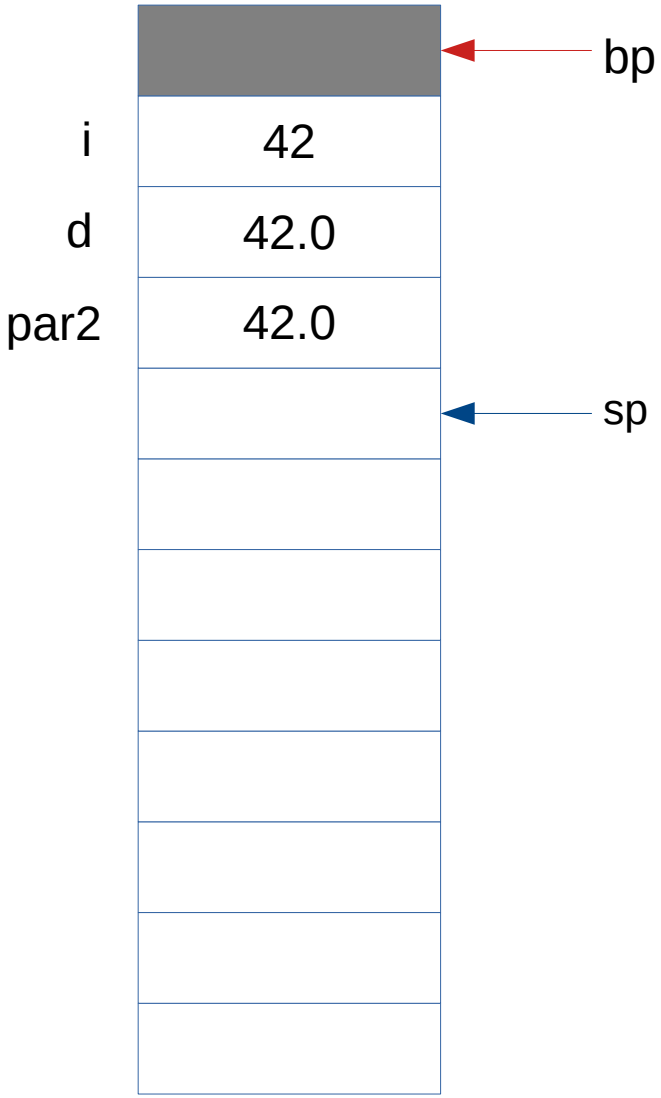
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



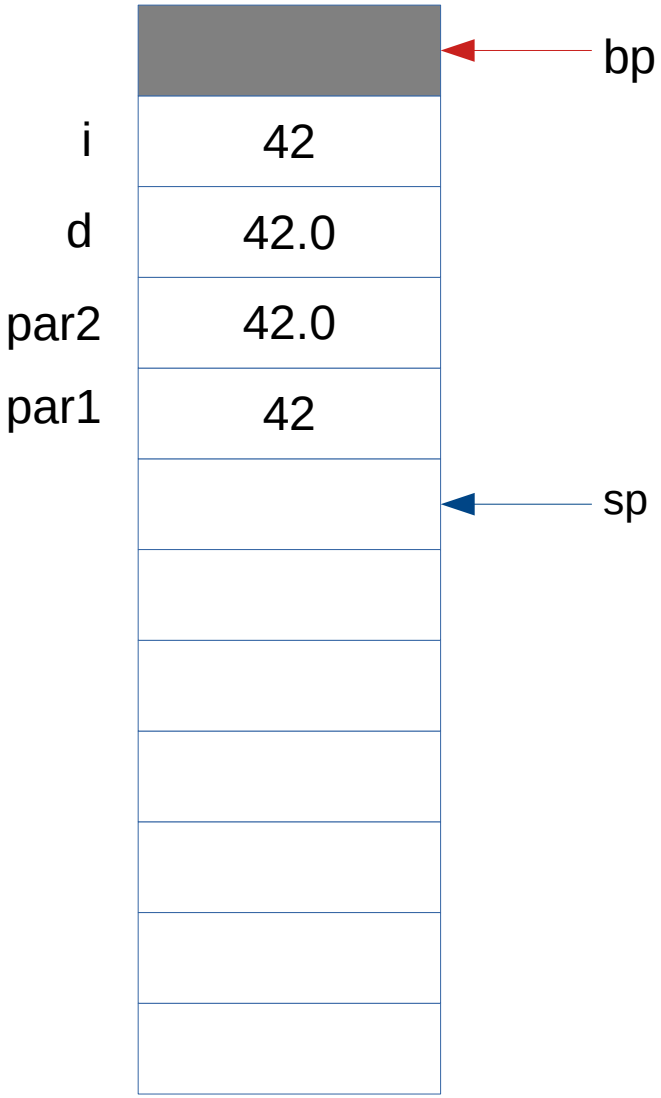
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



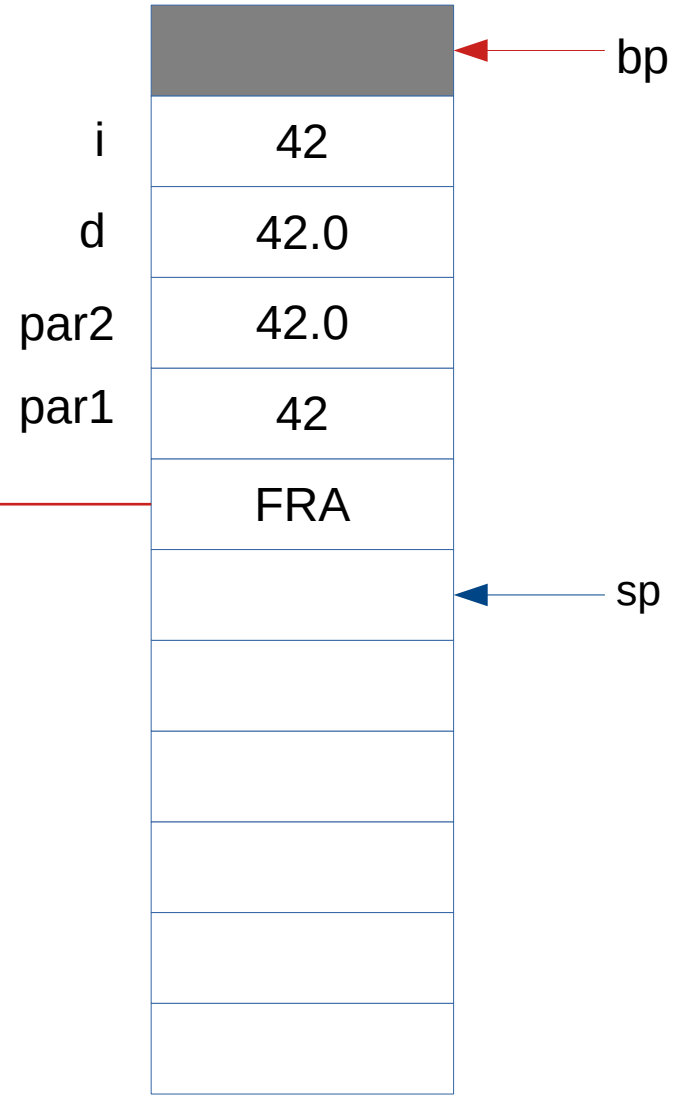
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



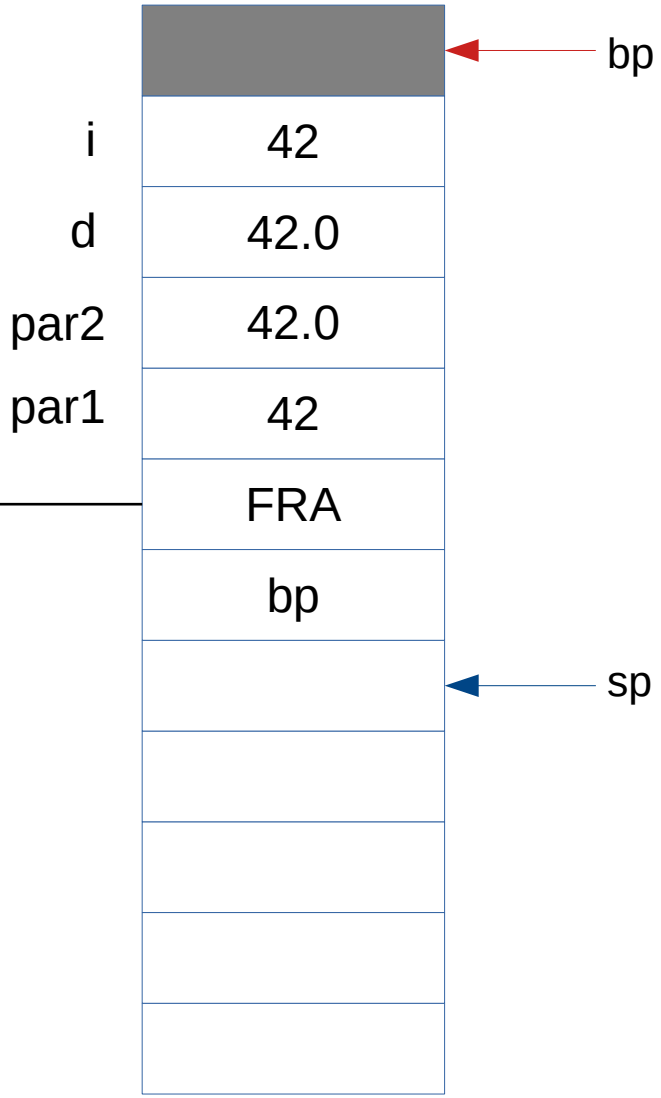
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



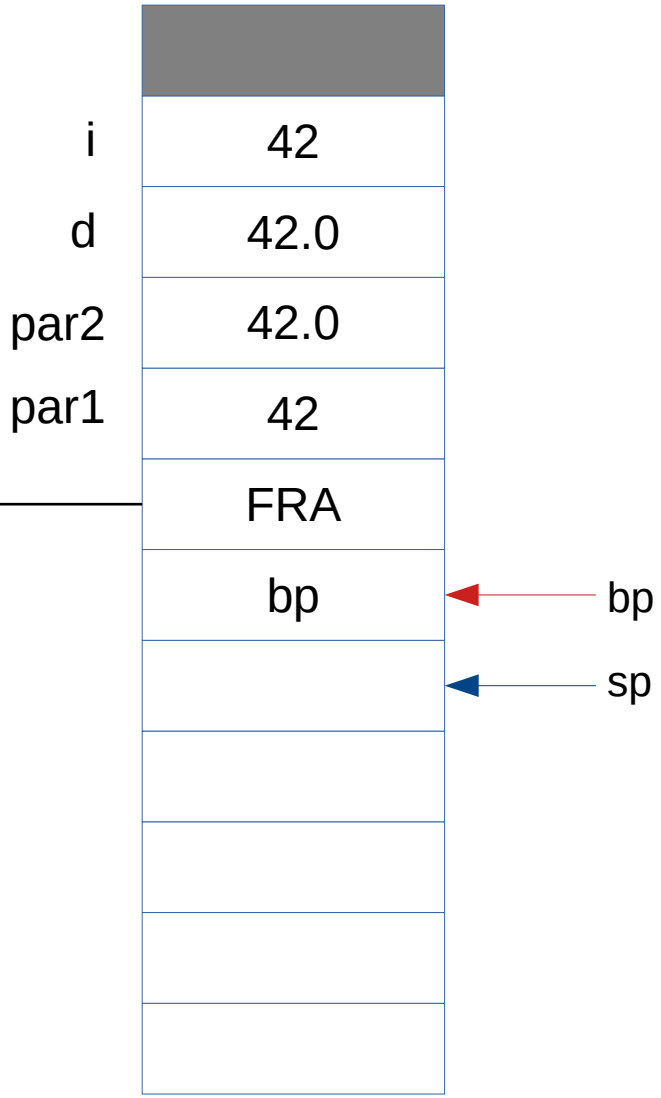
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



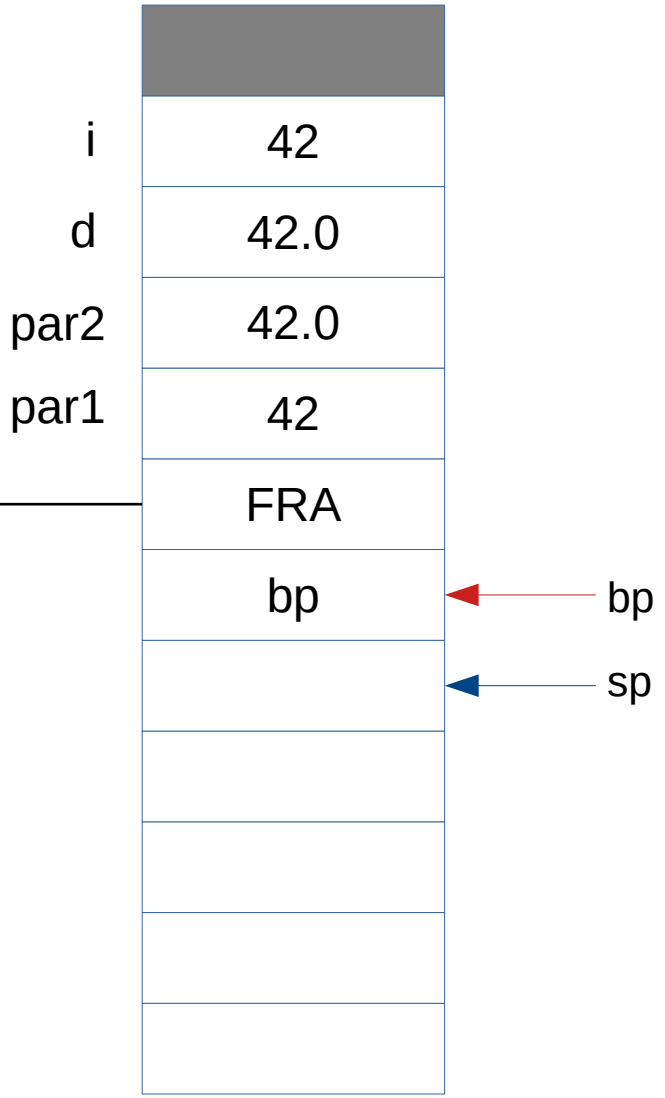
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



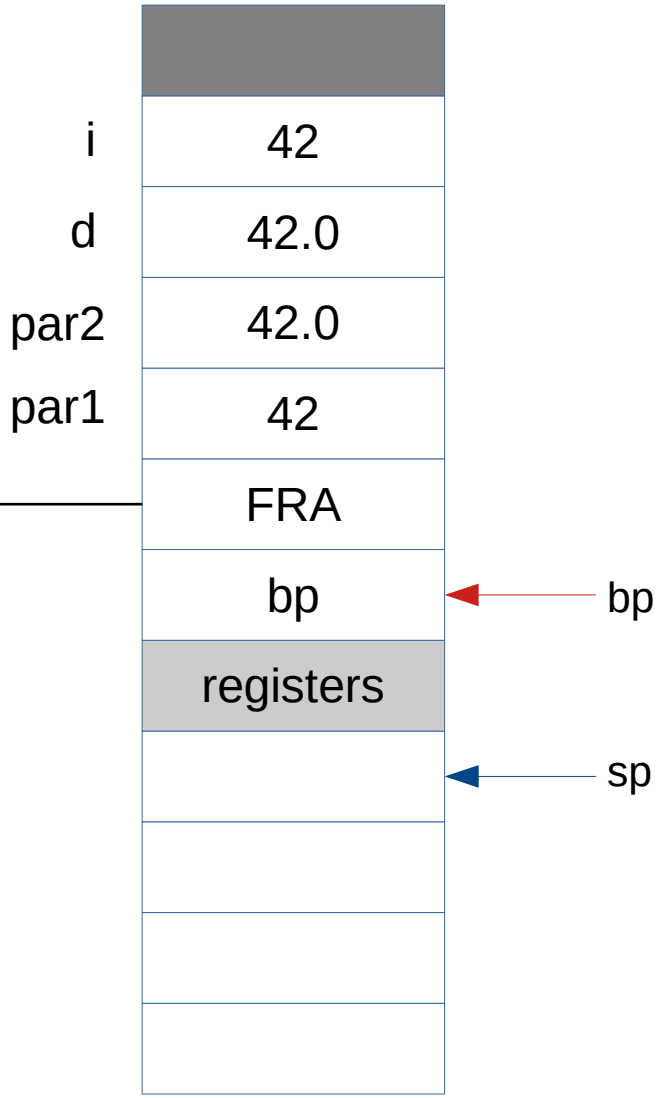
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



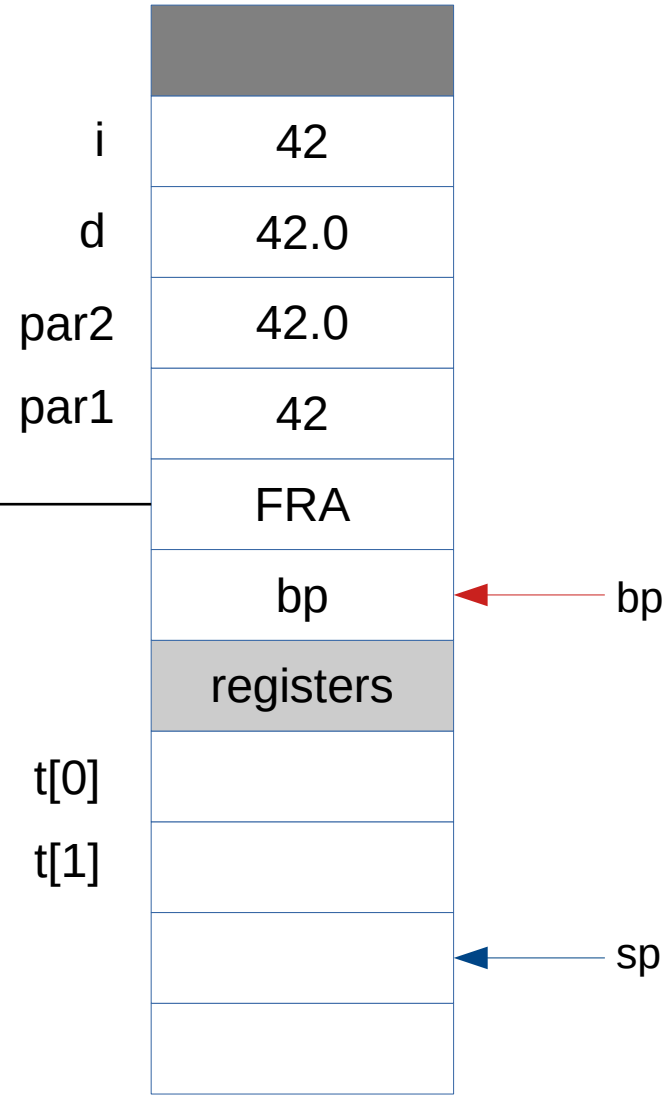
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



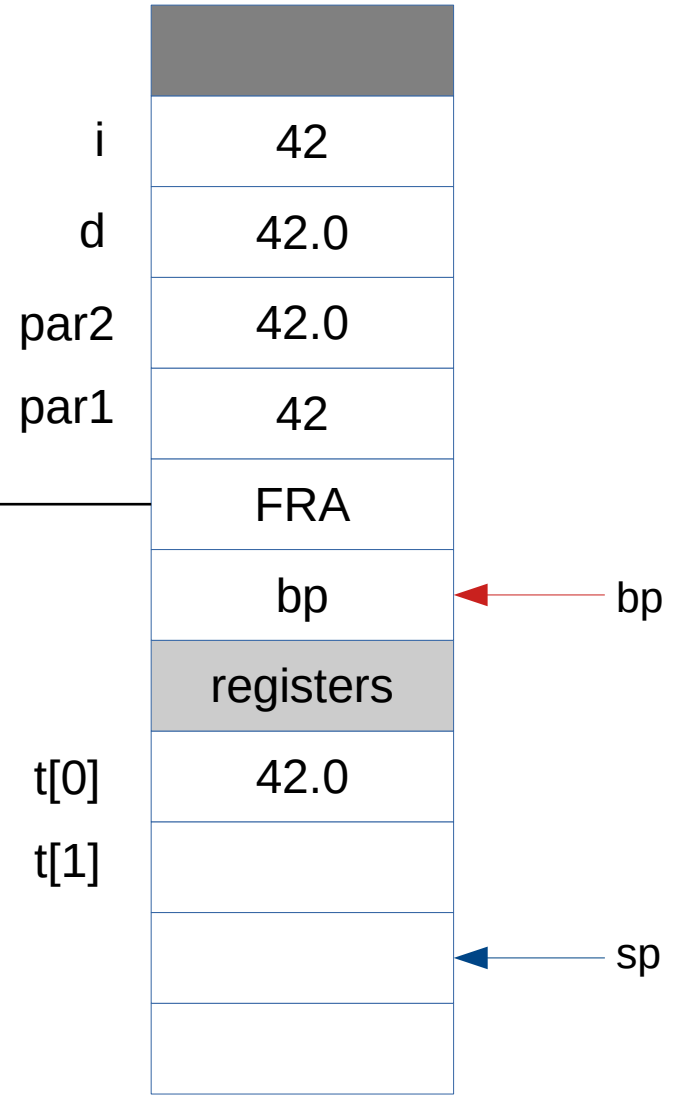
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



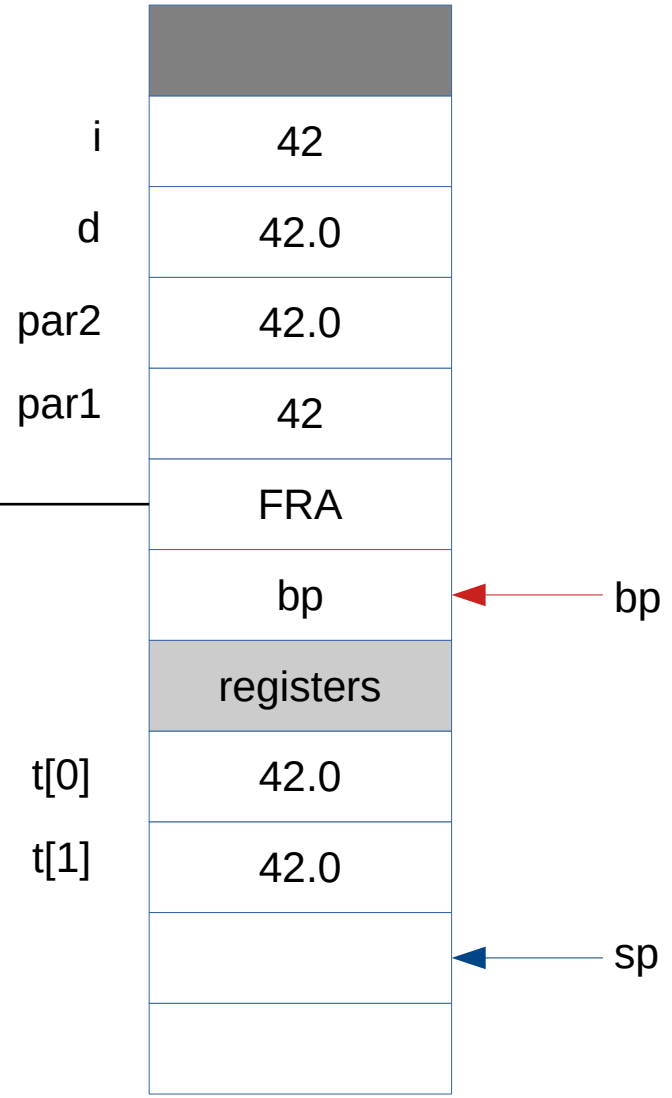
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



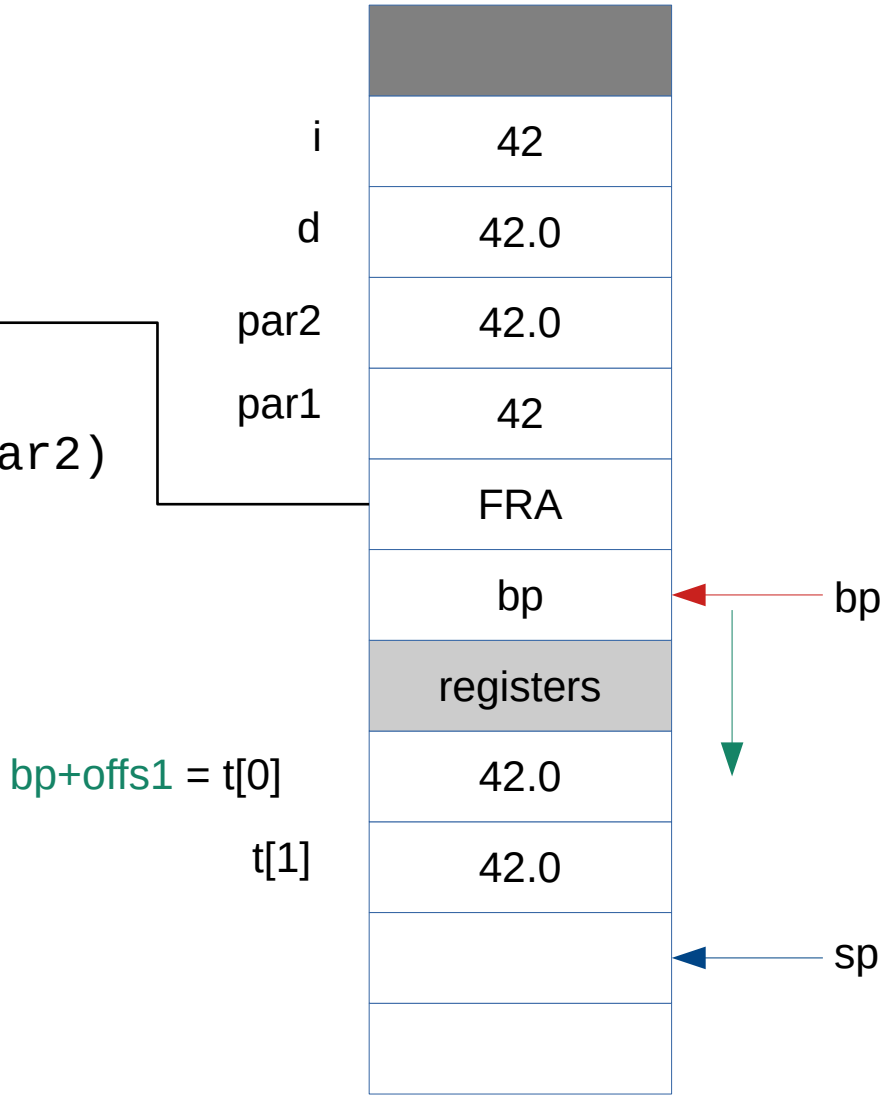
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



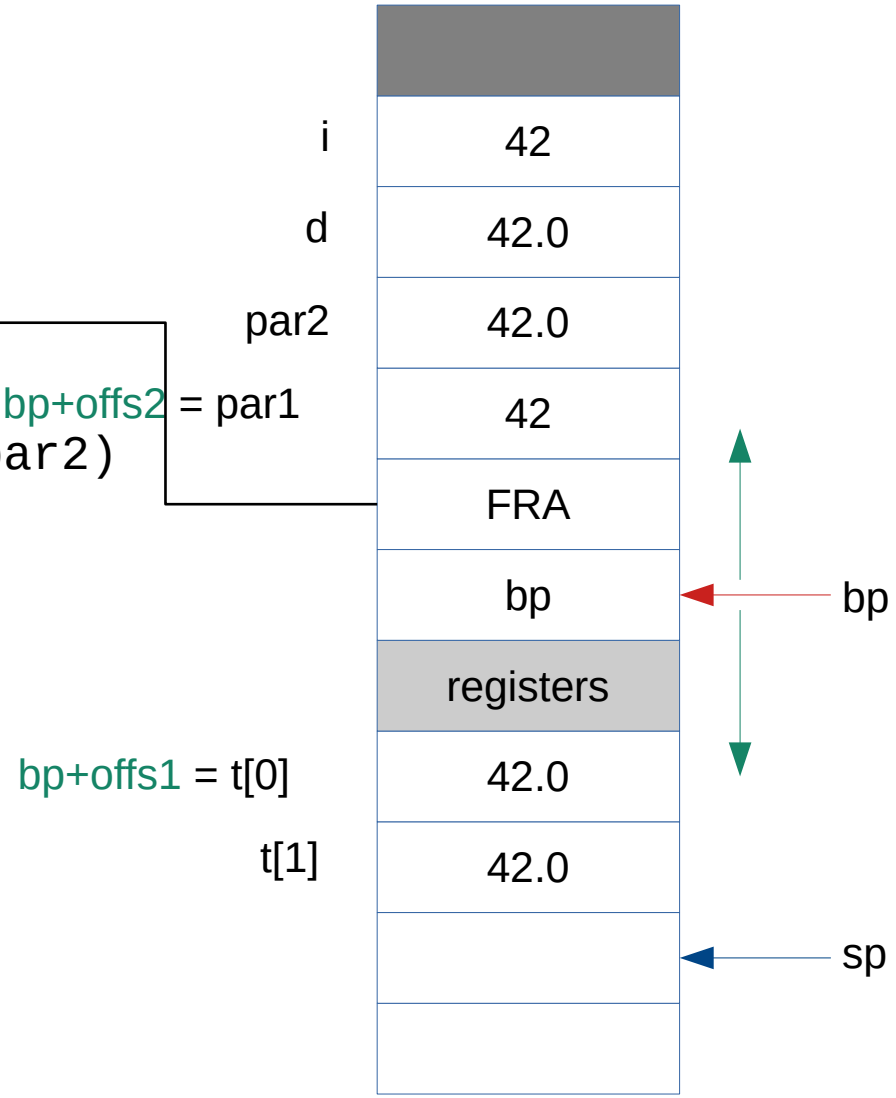
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



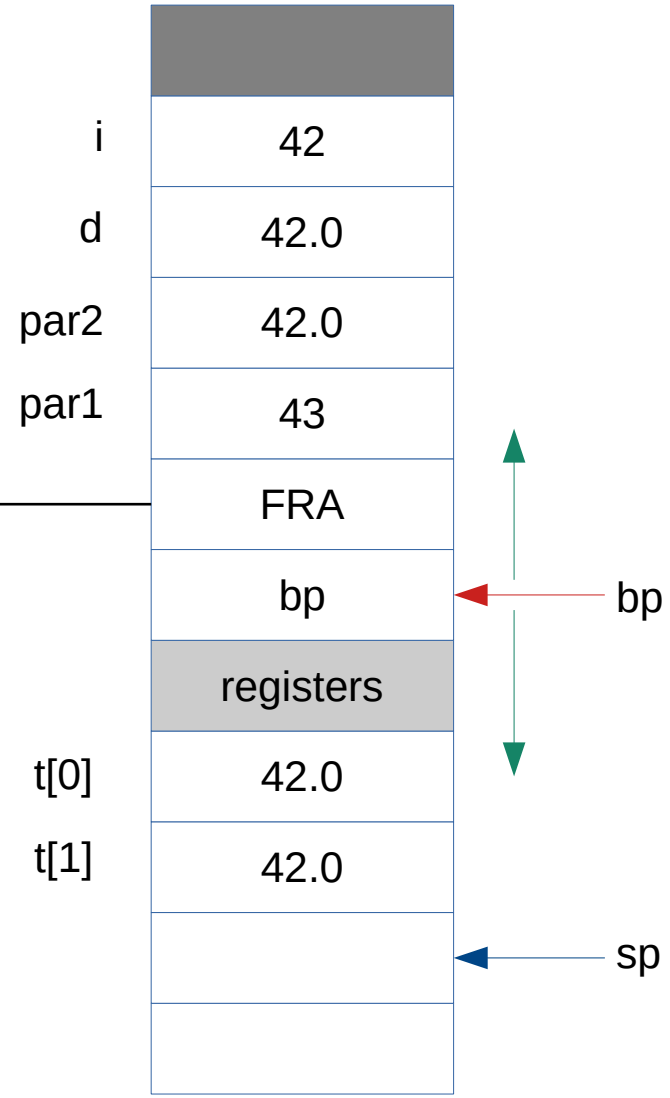
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



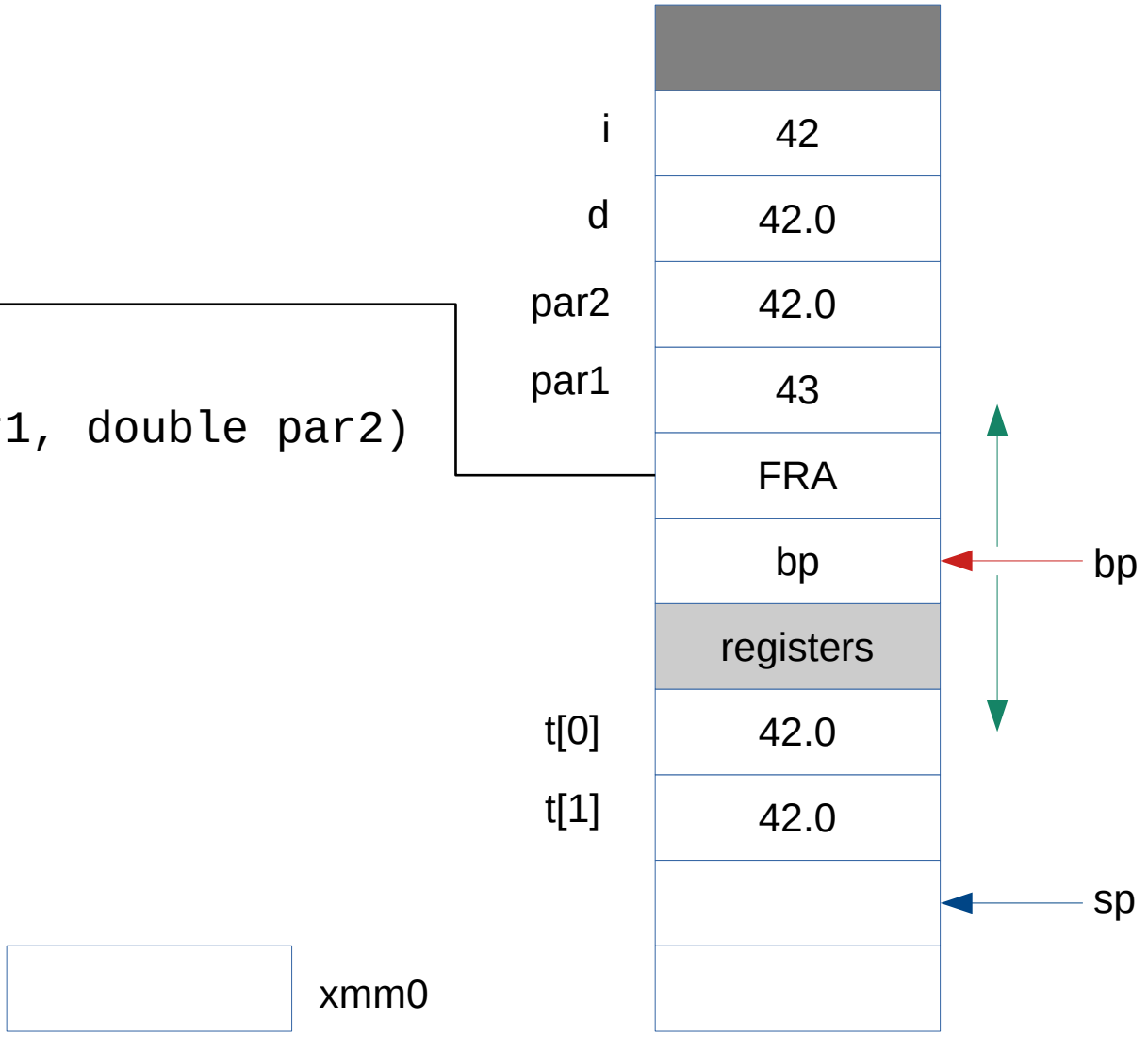
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



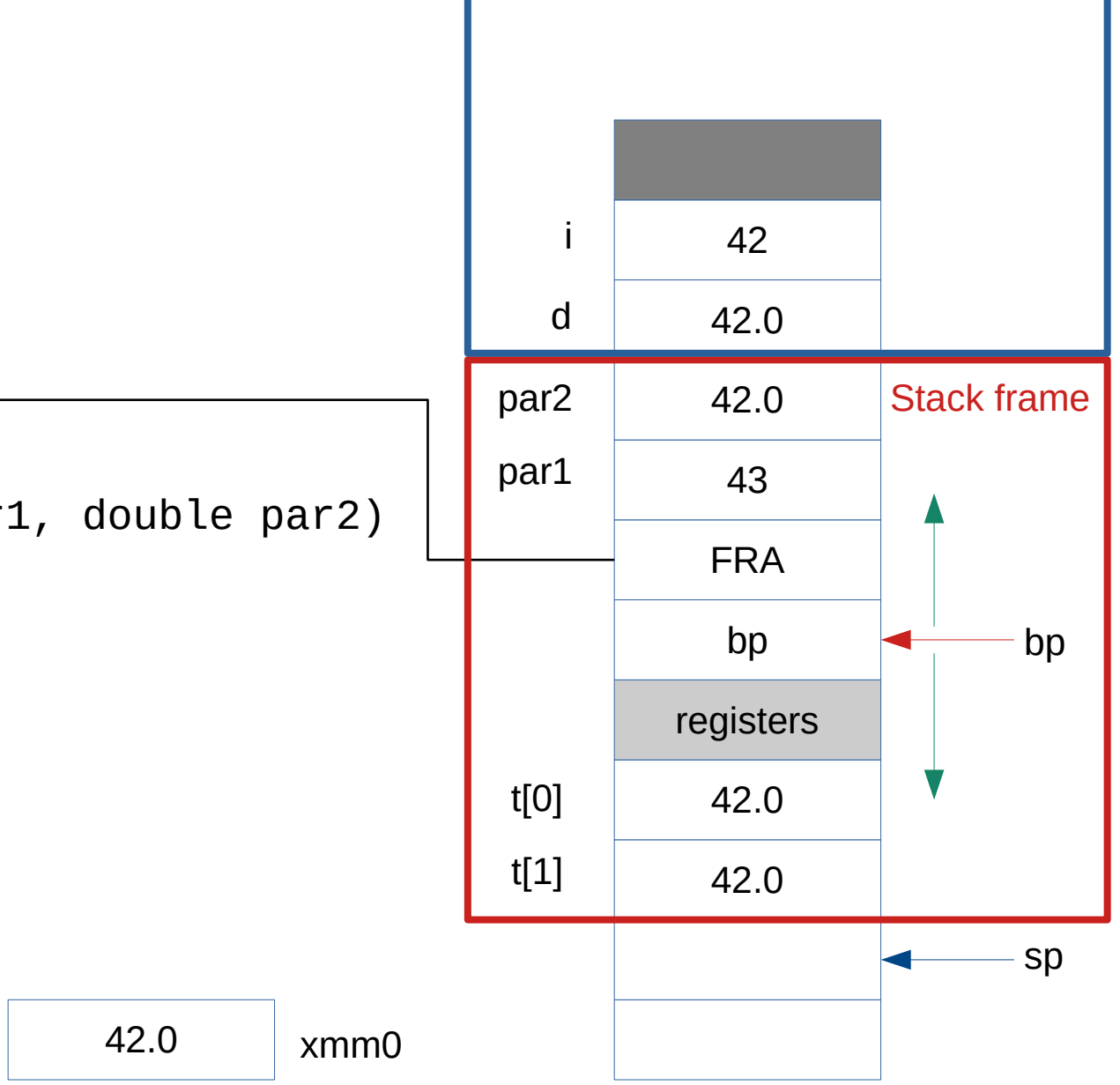
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



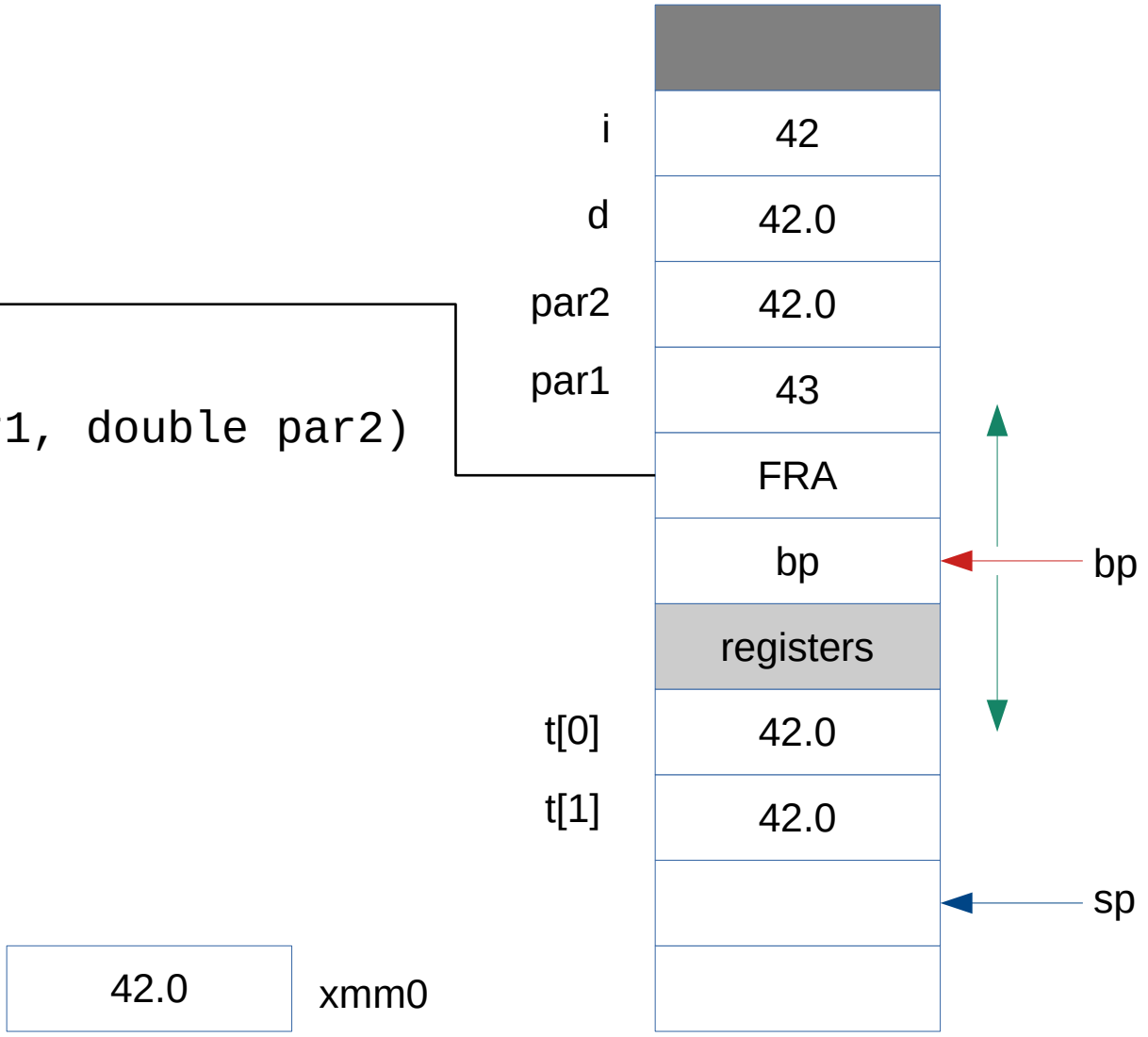
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



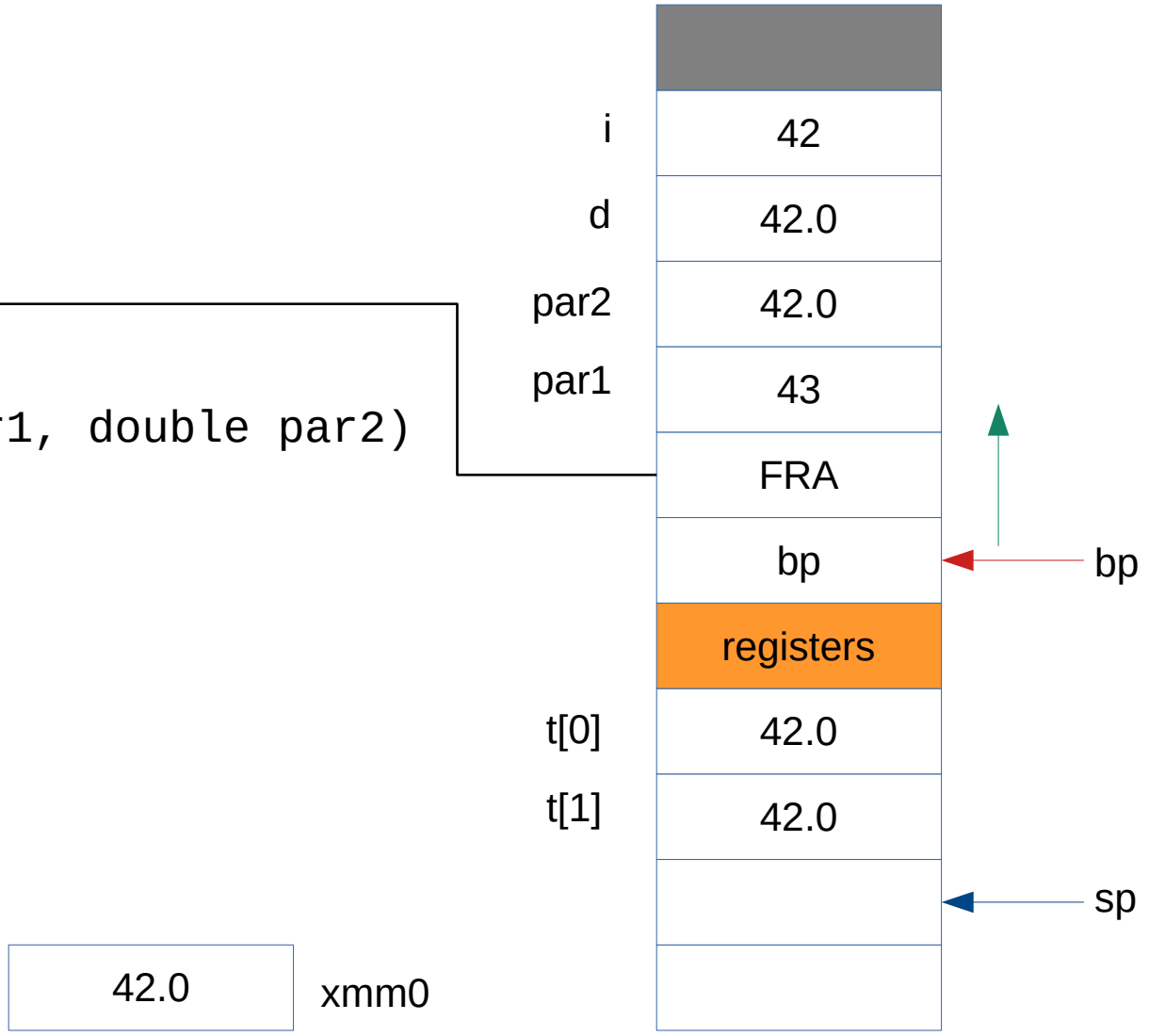
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



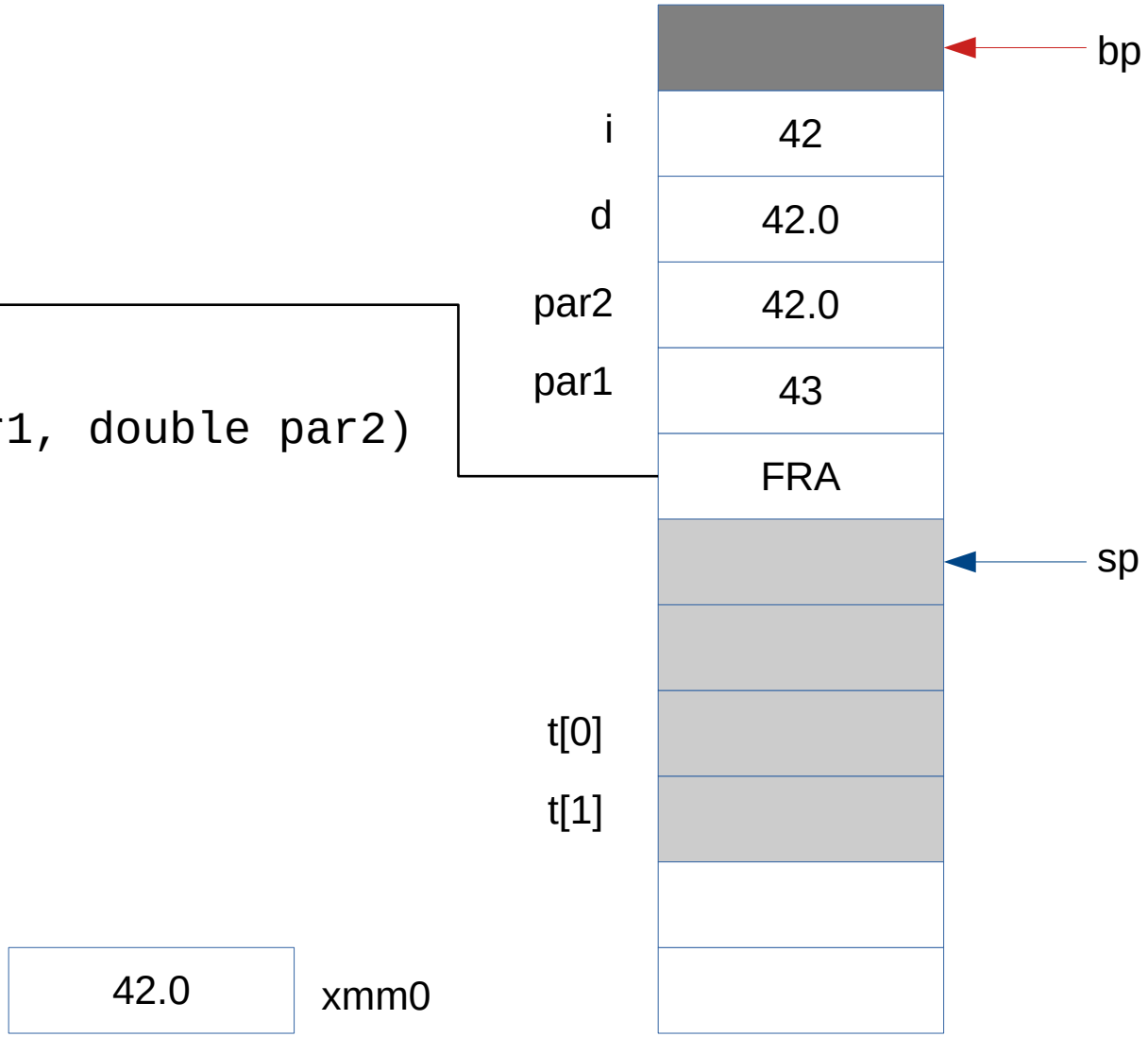
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



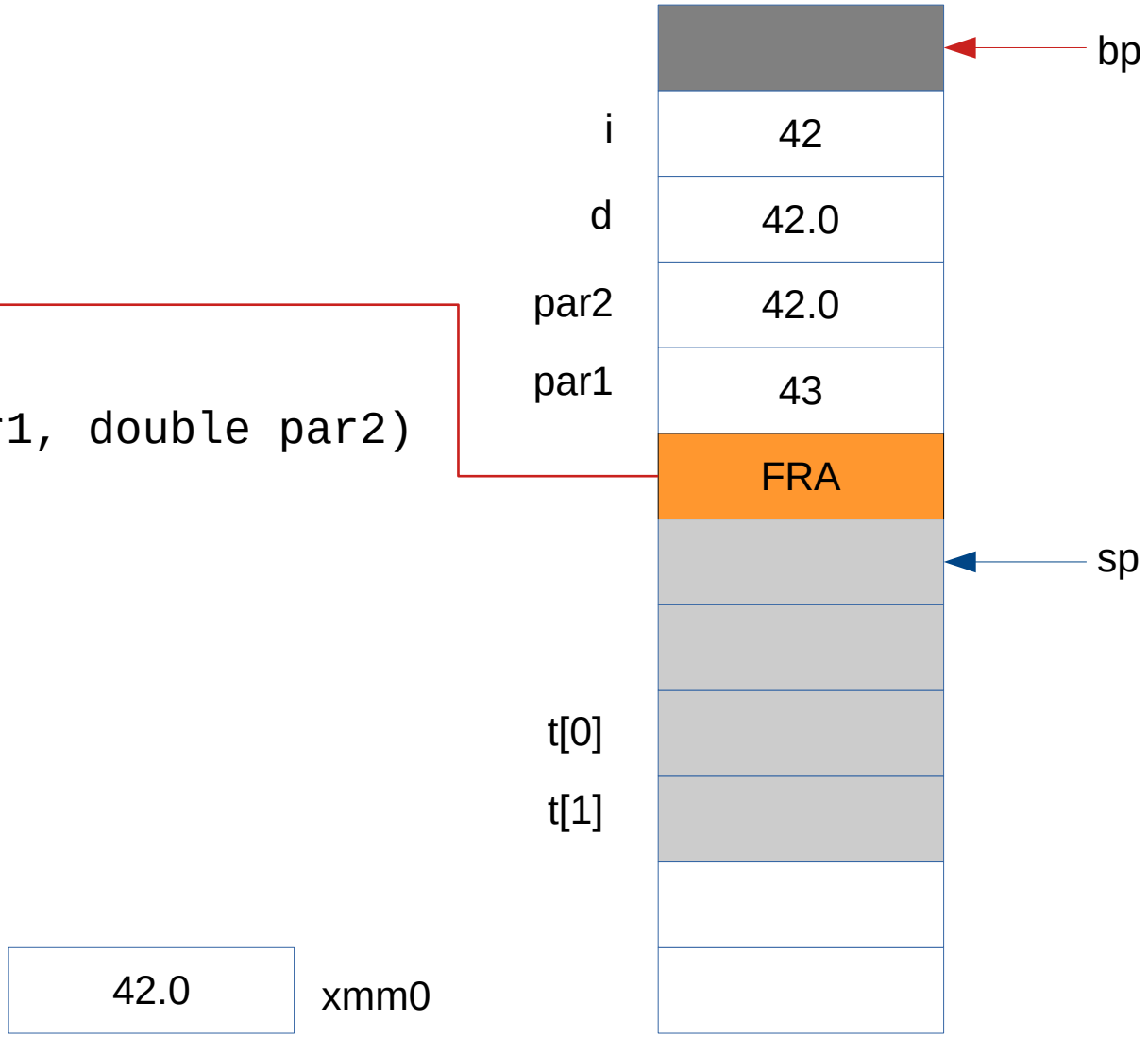
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



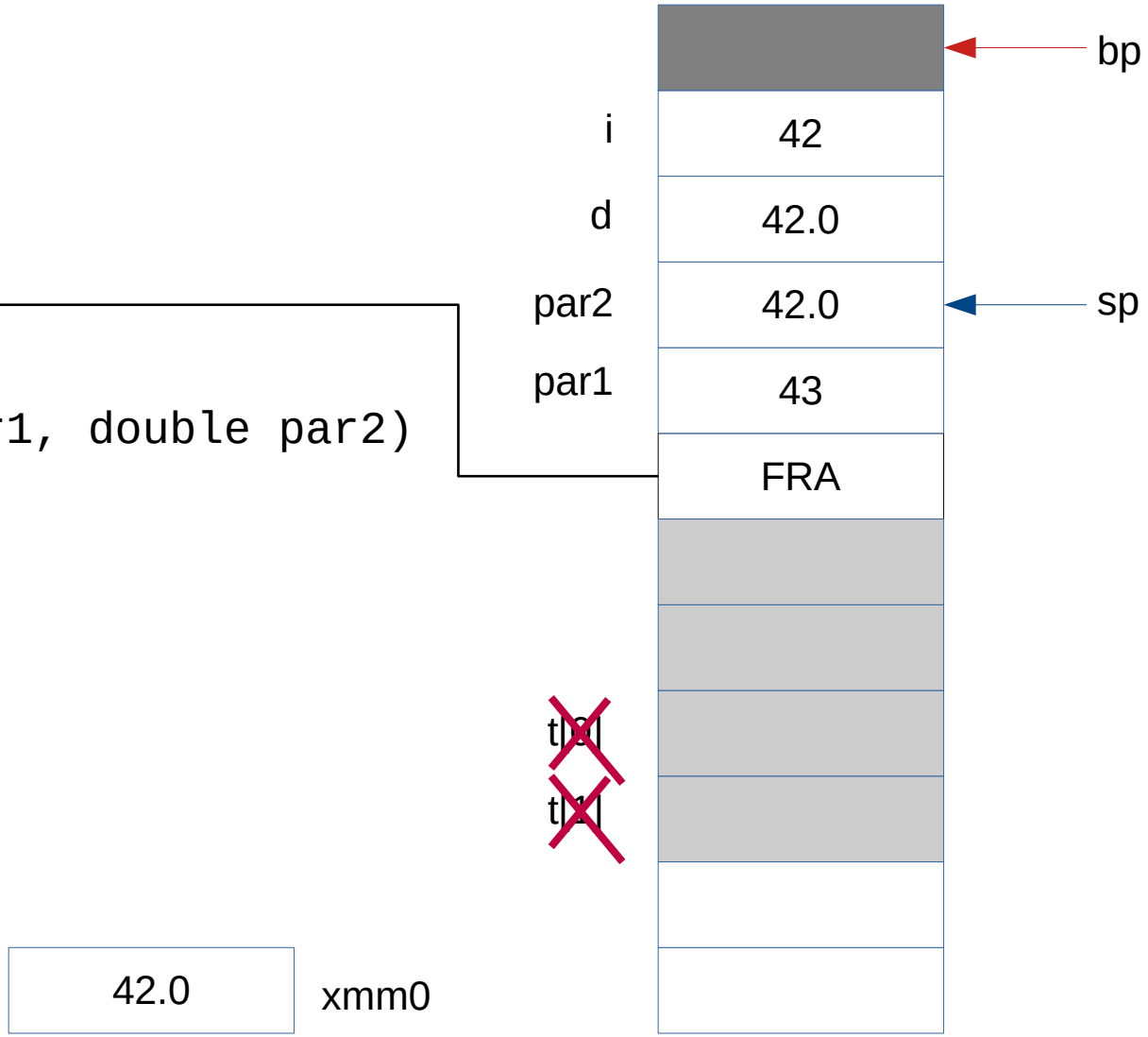
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



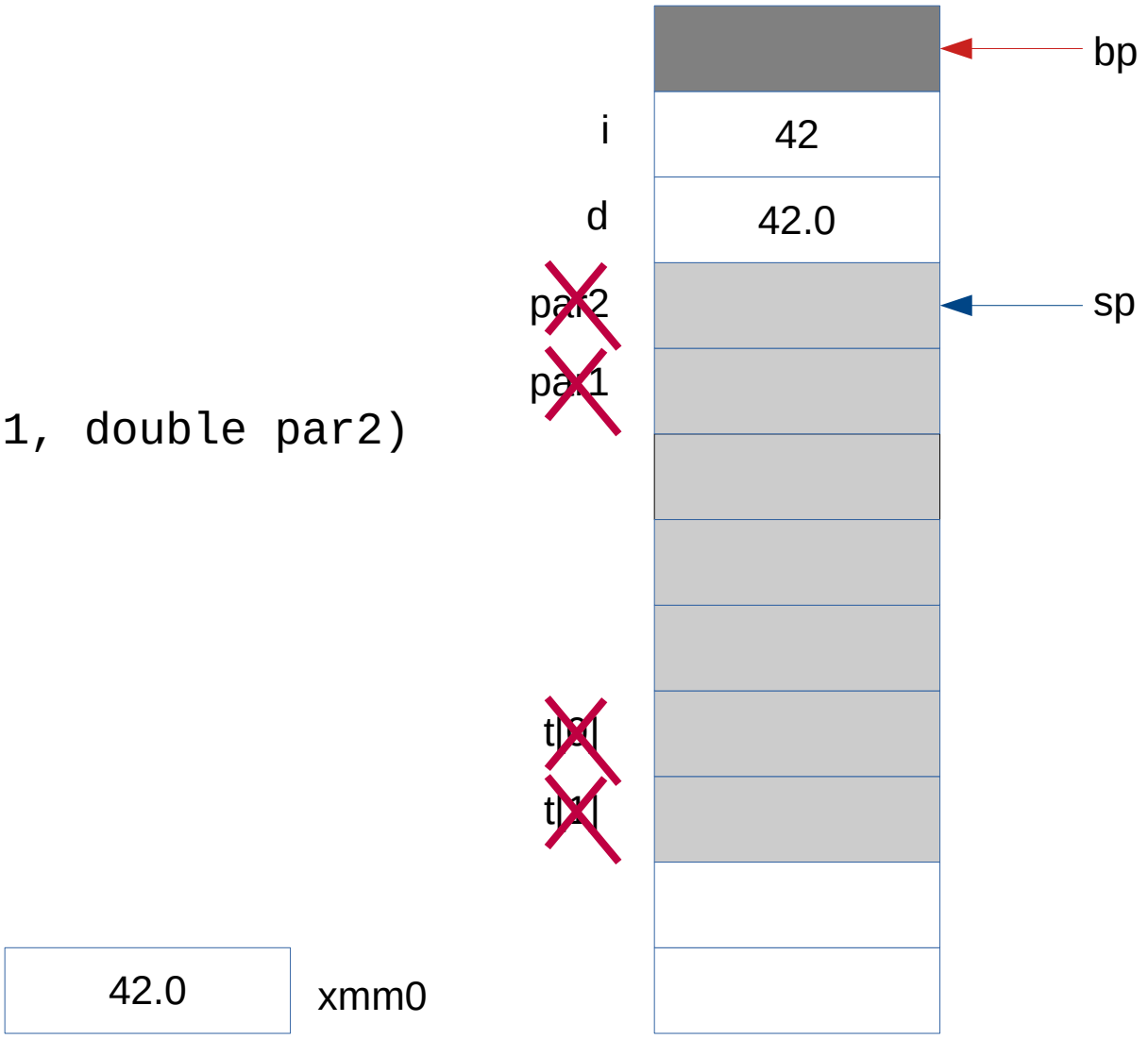
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



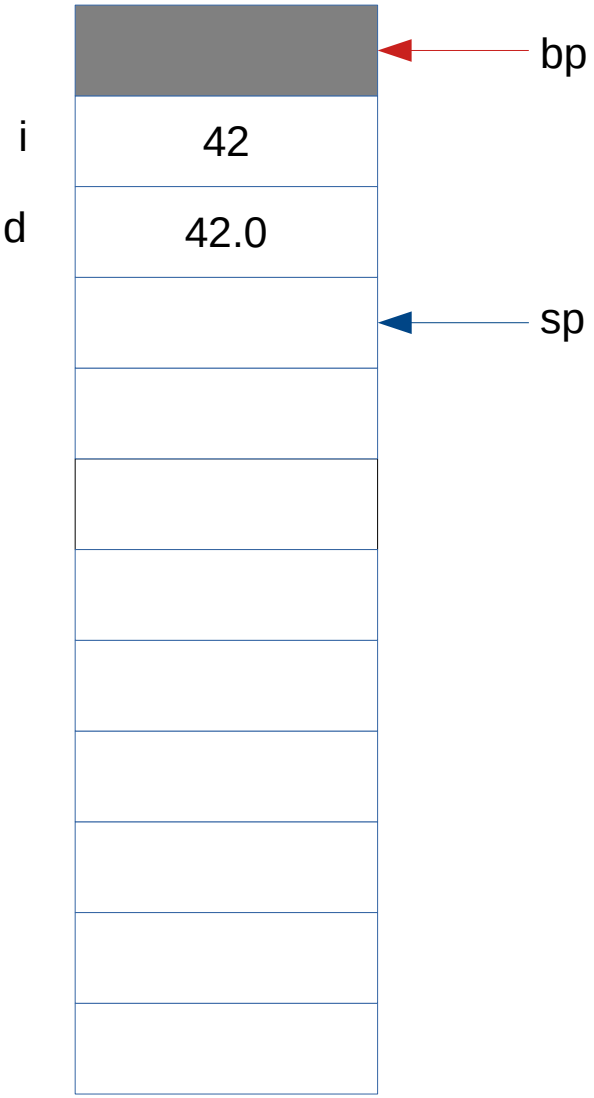
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



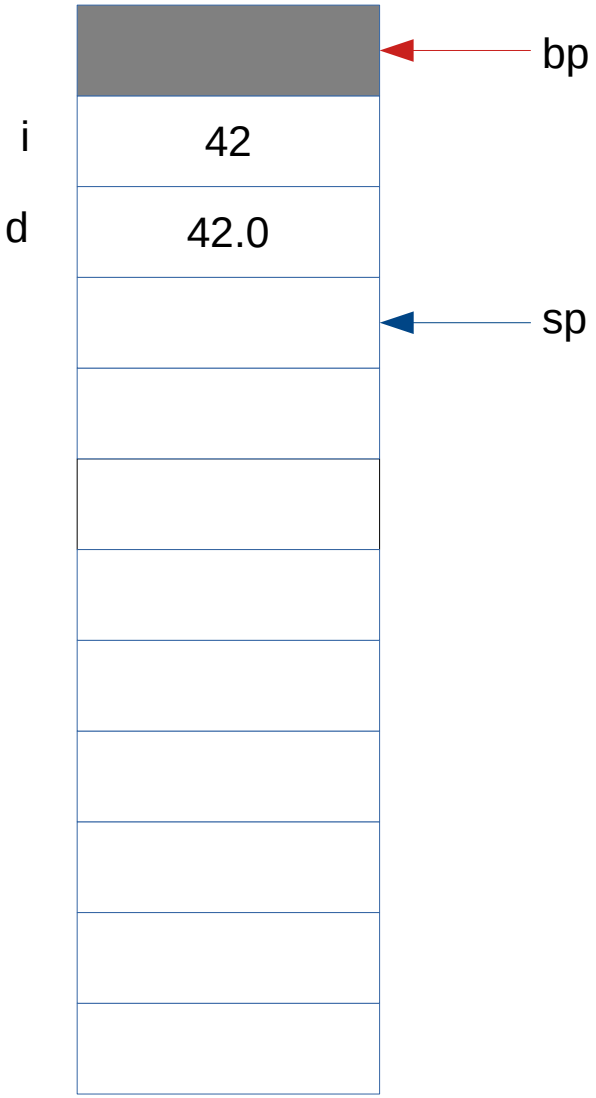
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



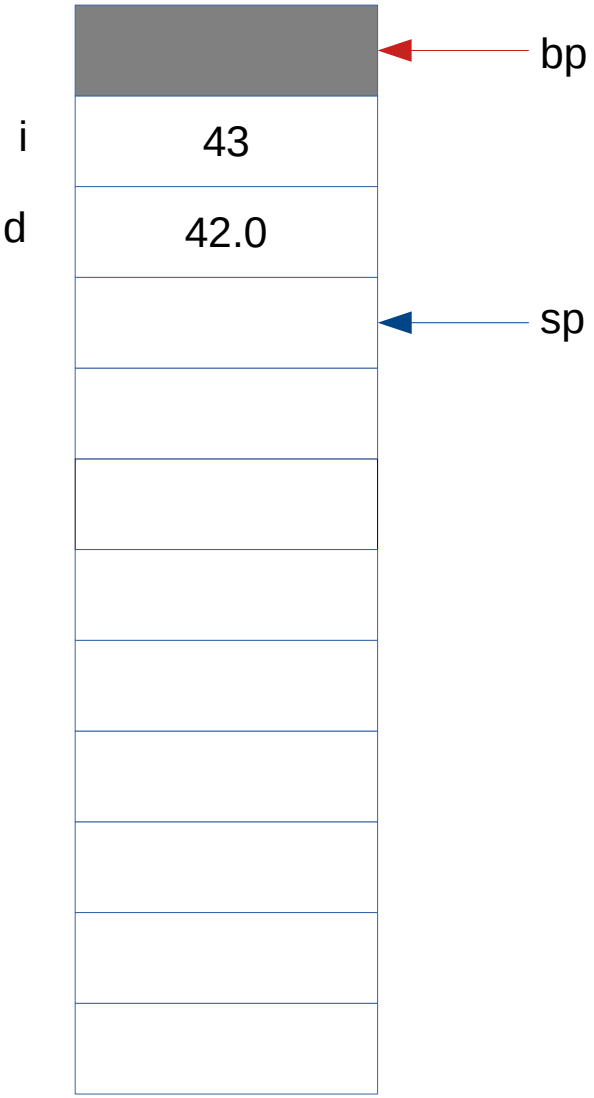
```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



```
void f(void)
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



Lifetime traps and pitfalls

Lifetime traps and pitfalls

- Write a function which prints a question to the stdout
- Then reads the answer from stdin and returns with that

```
void f(void)
{
  printf( "%d\n", answer("How are you? "));
}
```

```
$ a.out
How are you?
```

Lifetime traps and pittfals

- Write a function which prints a question to the stdout
- Then reads the answer from stdin and returns with that

```
void f(void)
{
  printf( "%d\n", answer("How are you? "));
}
```

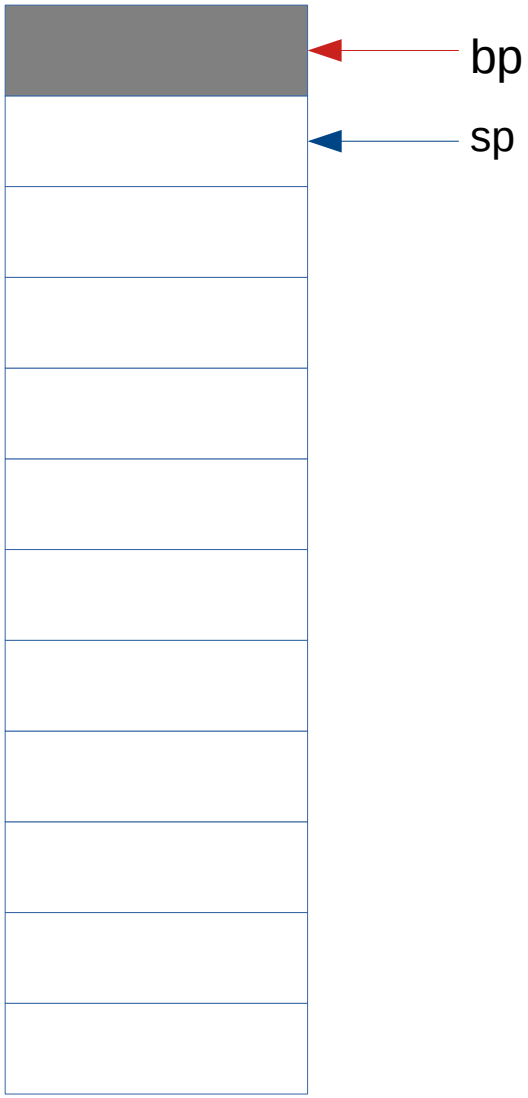
```
$ a.out
How are you? Thanks, fine.
Thanks, fine.
$
```

```
void f(void)
{
   printf(”answer = %s\n”,
        answer(”How are you?”));
}
```

```
char *answer( char *question)
{
   char buffer[20];
   printf(”%s ”, question);
   gets(buffer);
   return buffer;
}
```

.rodata

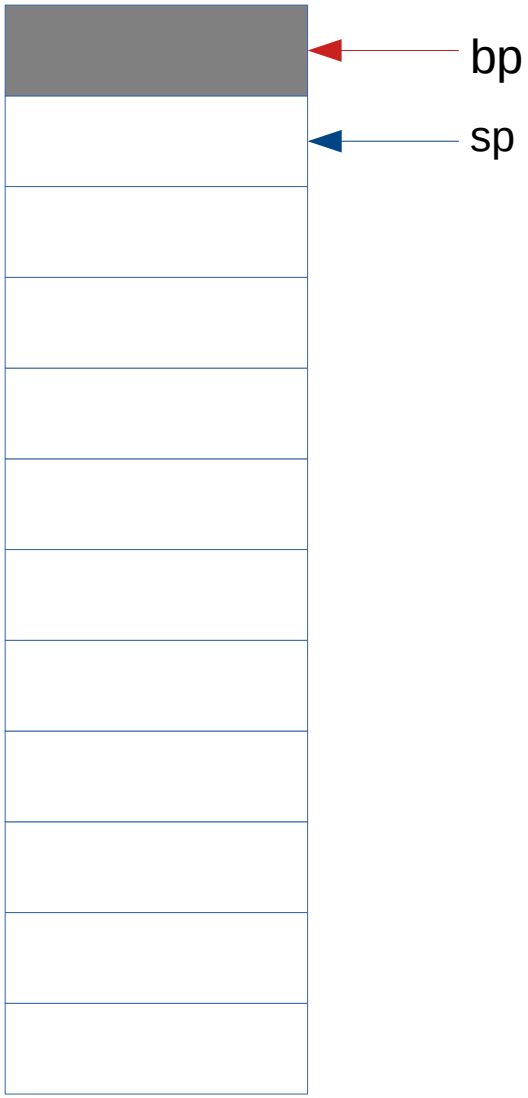
\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H




```
void f(void)
{
   printf(”answer = %s\n”,
        answer(”How are you?”));
}



char *answer( char *question)
{
   char buffer[20];
   printf(”%s ”, question);
   gets(buffer);
   return buffer;
}
```

.rodata

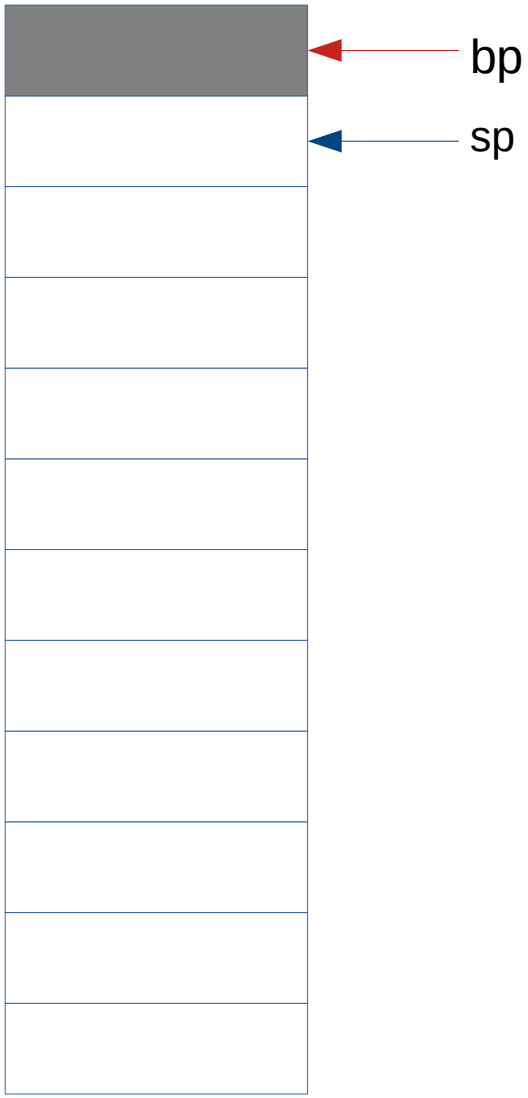
\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H




```
void f(void)
{
   printf(”answer = %s\n”,
        answer(”How are you?”));
}



char *answer( char *question)
{
   char buffer[20];
   printf(”%s ”, question);
   gets(buffer);
   return buffer;
}
```

.rodata

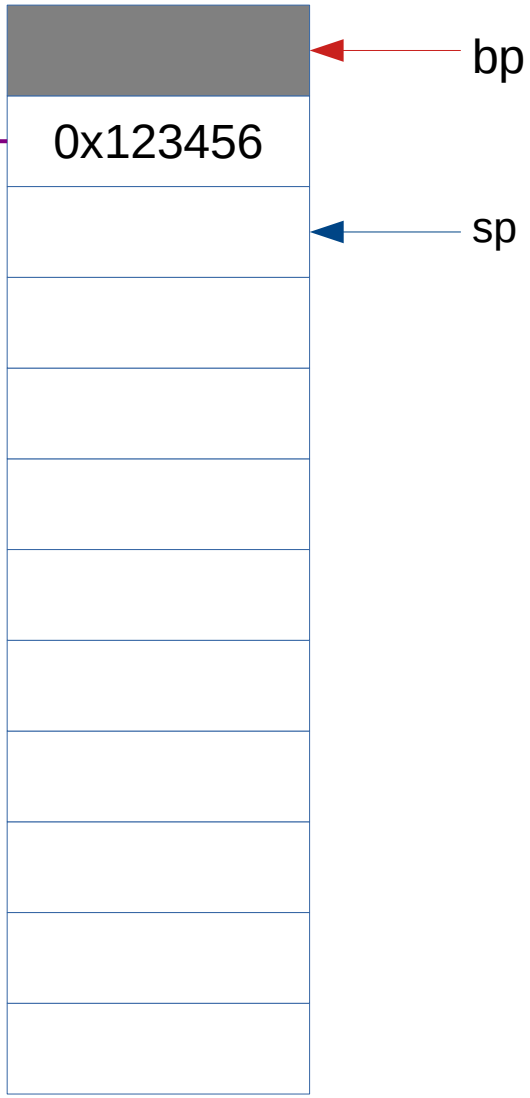
\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H



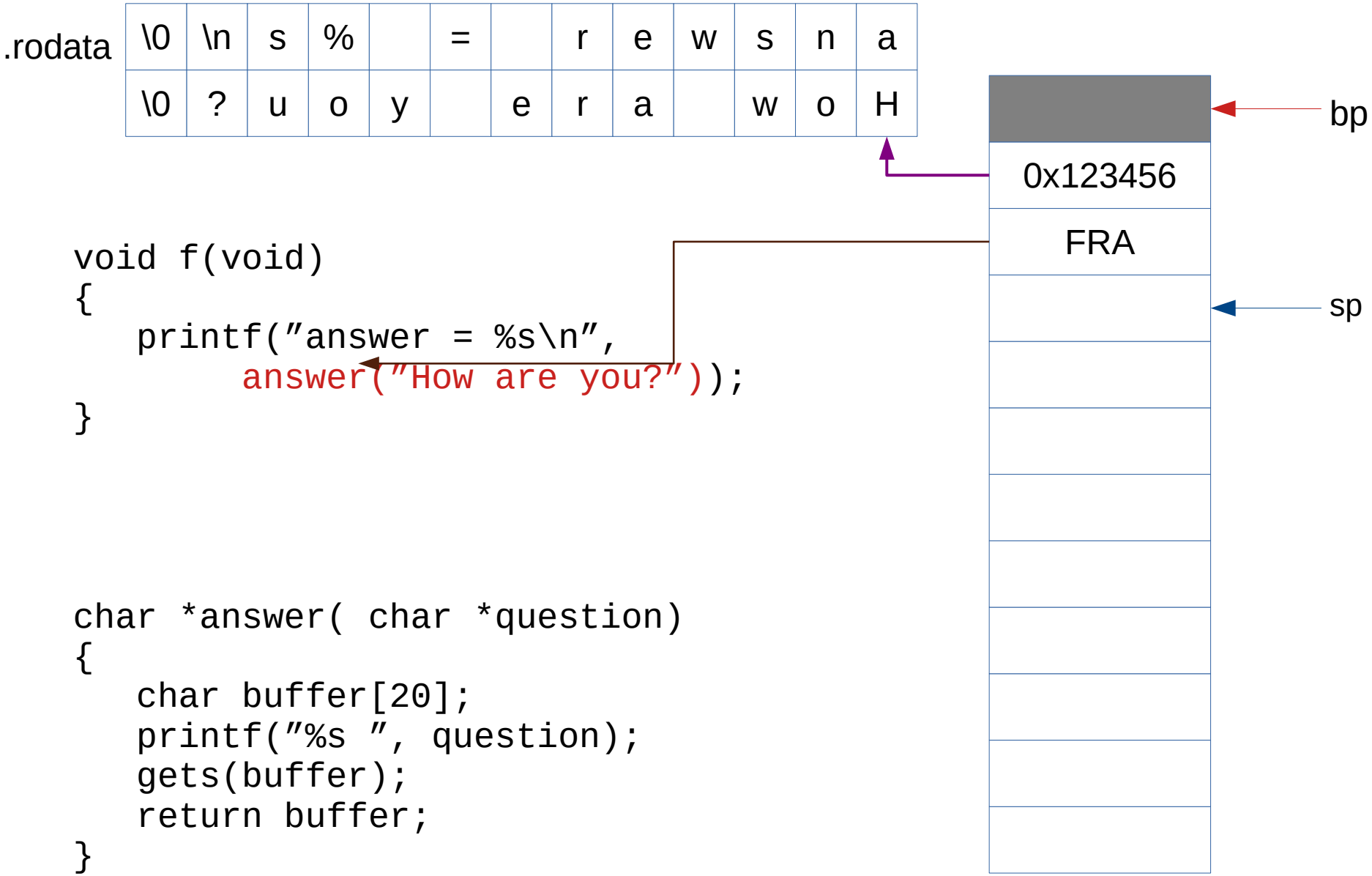

```
void f(void)
{
   printf(”answer = %s\n”,
        answer(”How are you?”));
}



char *answer( char *question)
{
   char buffer[20];
   printf(”%s ”, question);
   gets(buffer);
   return buffer;
}
```

.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

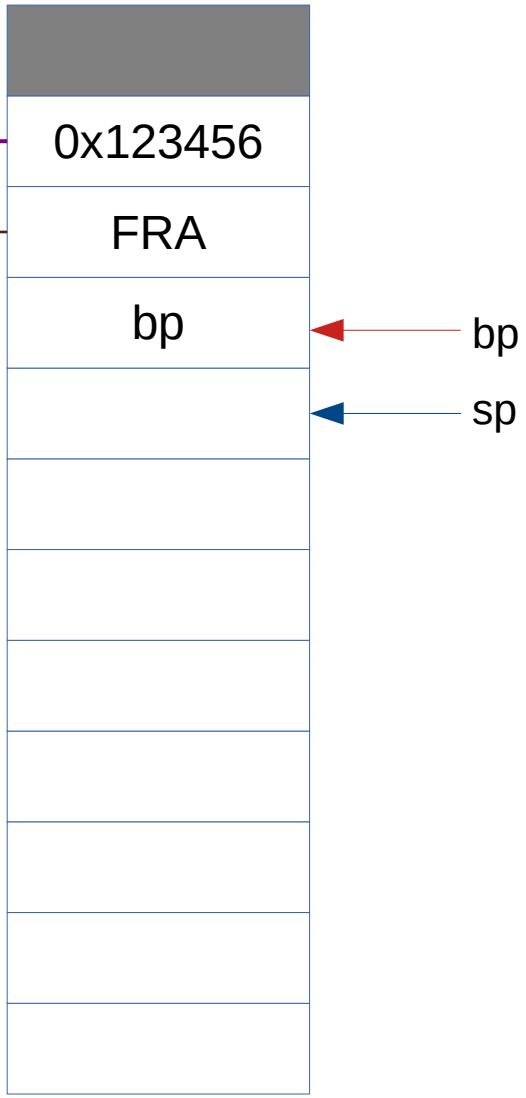



```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}



char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   gets(buffer);
   return buffer;
}
```


.rodata
\0 \n s % = r e w s n a
\0 ? u o y e r a w o H
0x123456
FRA
bp
sp
void f(void)
{
printf(”answer = %s\n”,
answer(”How are you?”));
}
char *answer(char *question)
{
char buffer[20];
printf(”%s ”, question);
gets(buffer);
return buffer;
}

.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

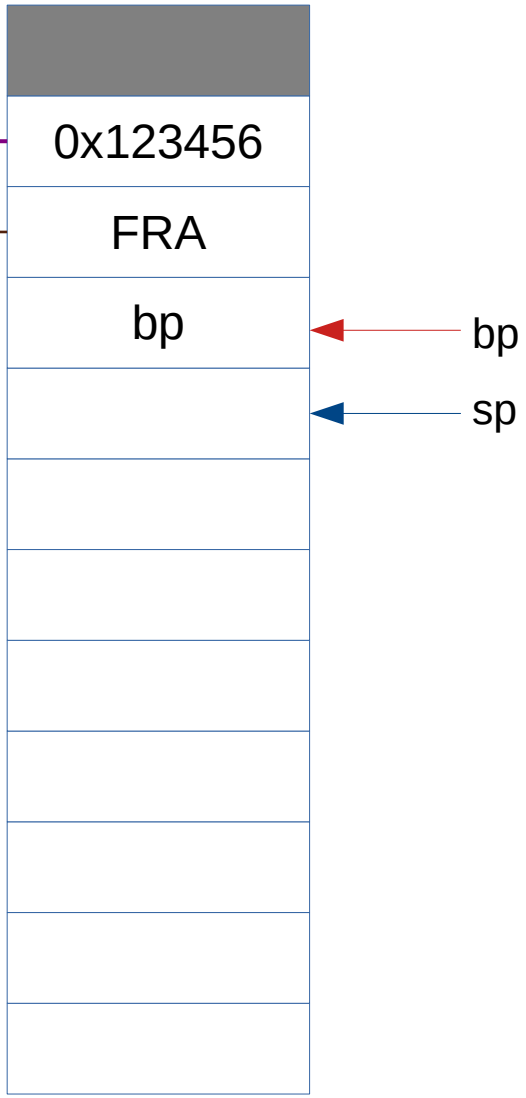



```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   gets(buffer);
   return buffer;
}
```

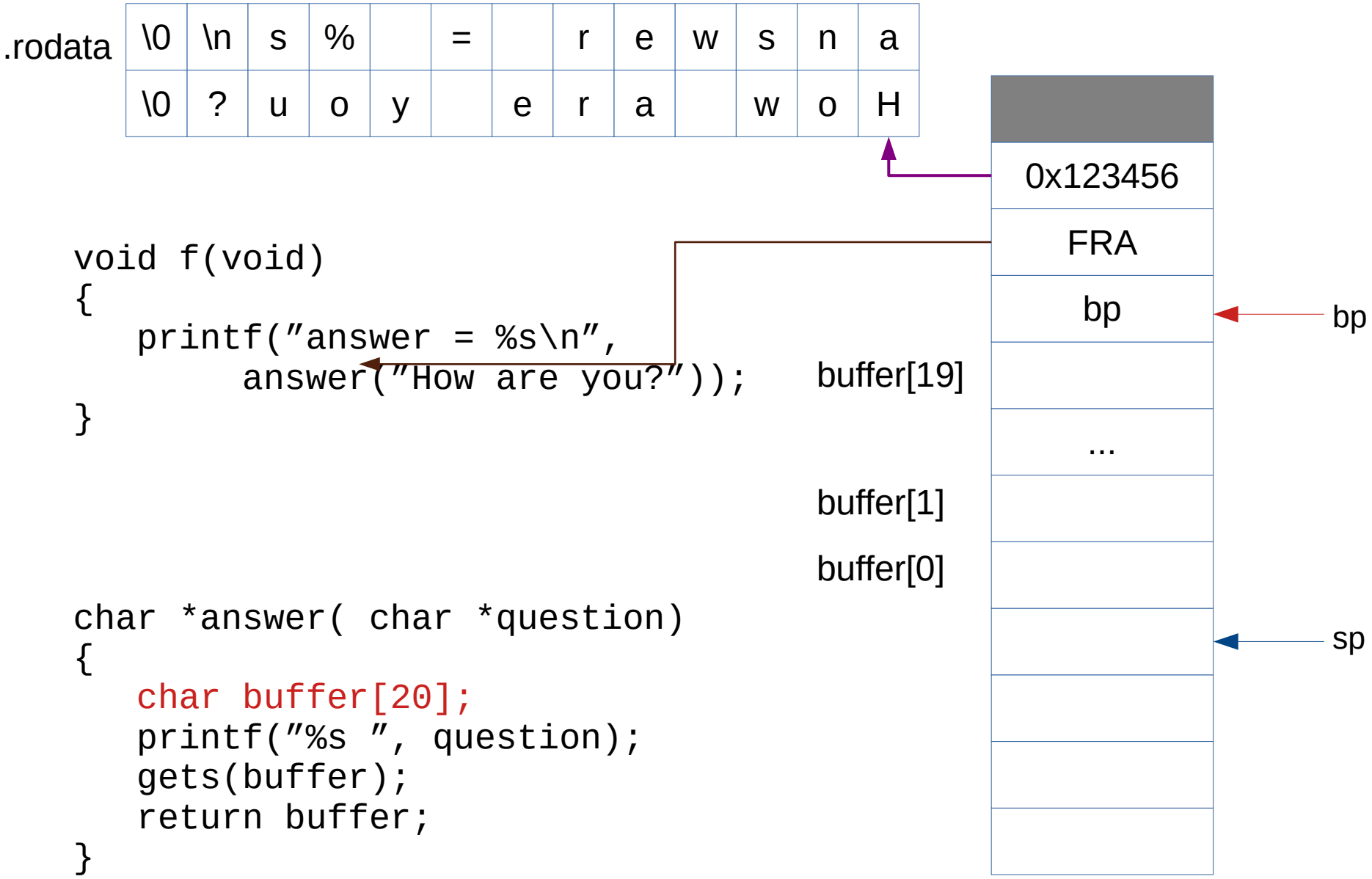
.rodata

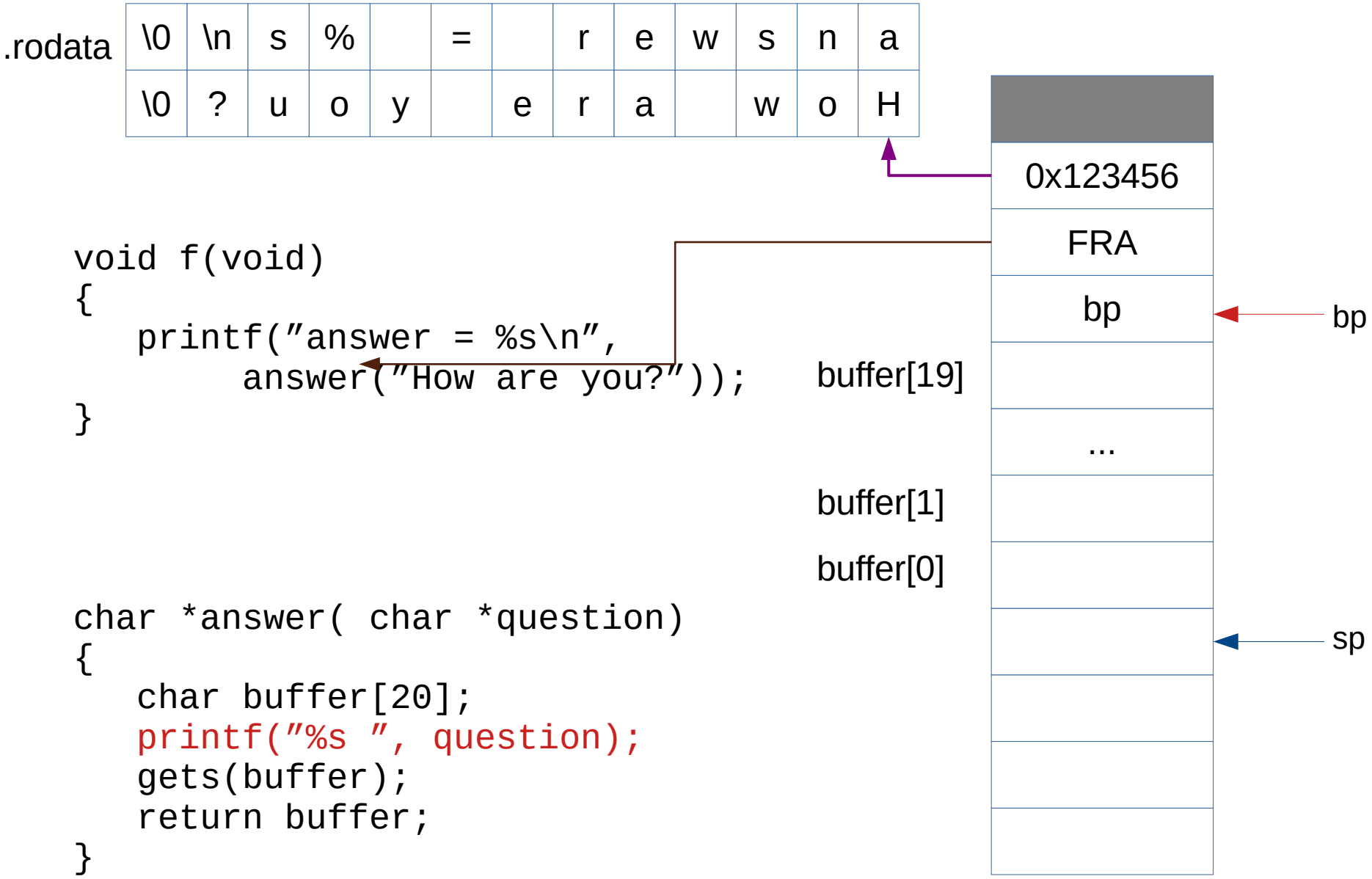
\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H




```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   gets(buffer);
   return buffer;
}
```


.rodata
\0 \n s % = r e w s n a
\0 ? u o y e r a w o H
0x123456
FRA
bp
bp
buffer[19]
...
buffer[1]
buffer[0]
sp
void f(void)
{
printf(”answer = %s\n”,
answer(”How are you?”));
}
char *answer(char *question)
{
char buffer[20];
printf(”%s ”, question);
gets(buffer);
return buffer;
}

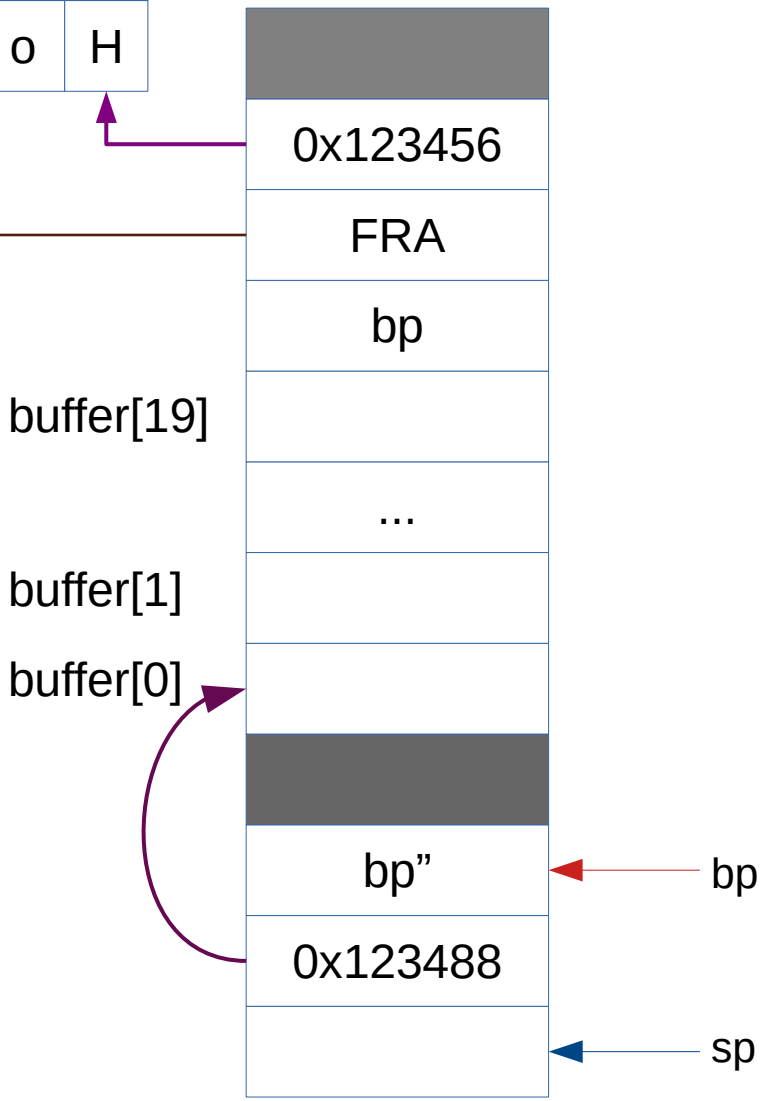

.rodata
\0 \n s % = r e w s n a
\0 ? u o y e r a w o H
0x123456
FRA
bp
bp
buffer[19]
...
buffer[1]
buffer[0]
sp
void f(void)
{
printf(”answer = %s\n”,
answer(”How are you?”));
}
char *answer(char *question)
{
char buffer[20];
printf(”%s ”, question);
gets(buffer);
return buffer;
}

.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   gets(buffer);
   return buffer;
}
```

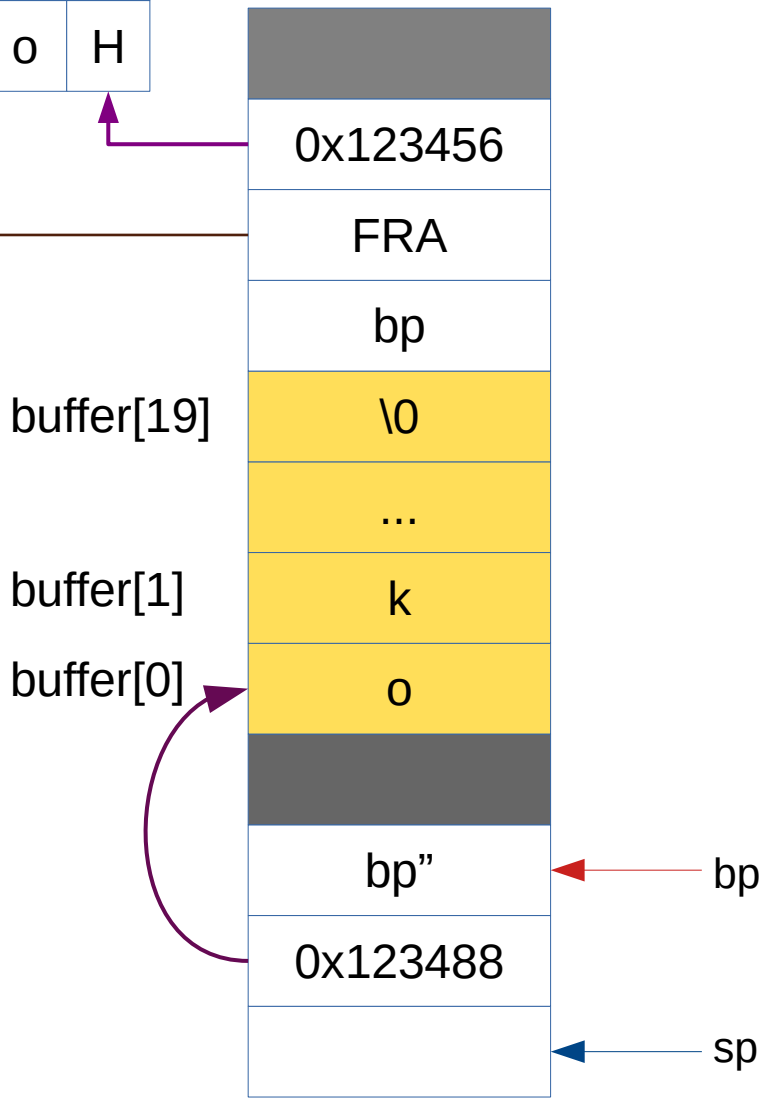


.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
    printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
    char buffer[20];
    printf("%s ", question);
    gets(buffer);
    return buffer;
}
```

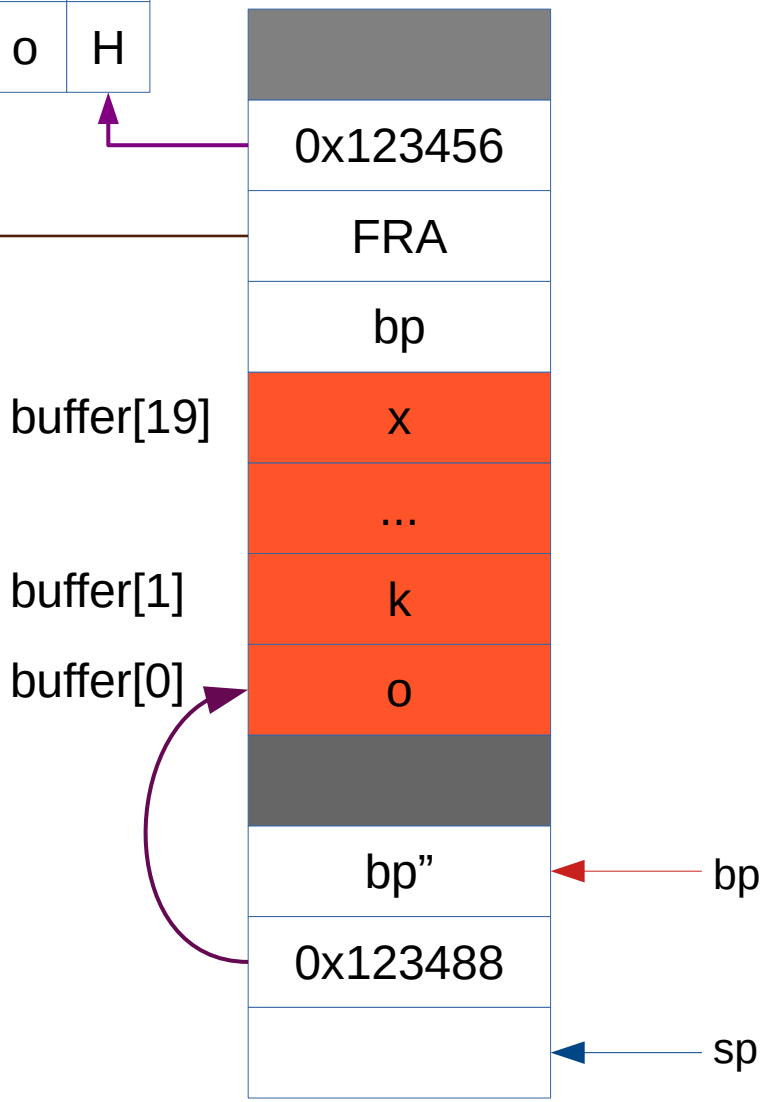


.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   gets(buffer);
   return buffer;
}
```

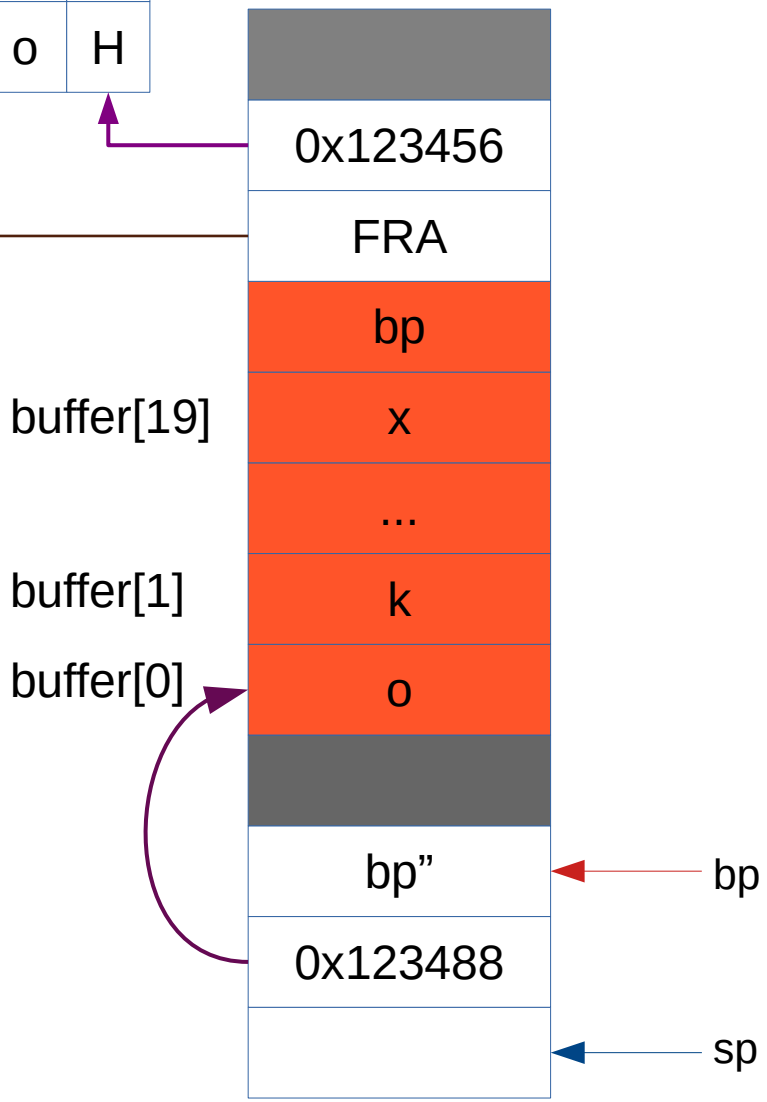


.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   gets(buffer);
   return buffer;
}
```

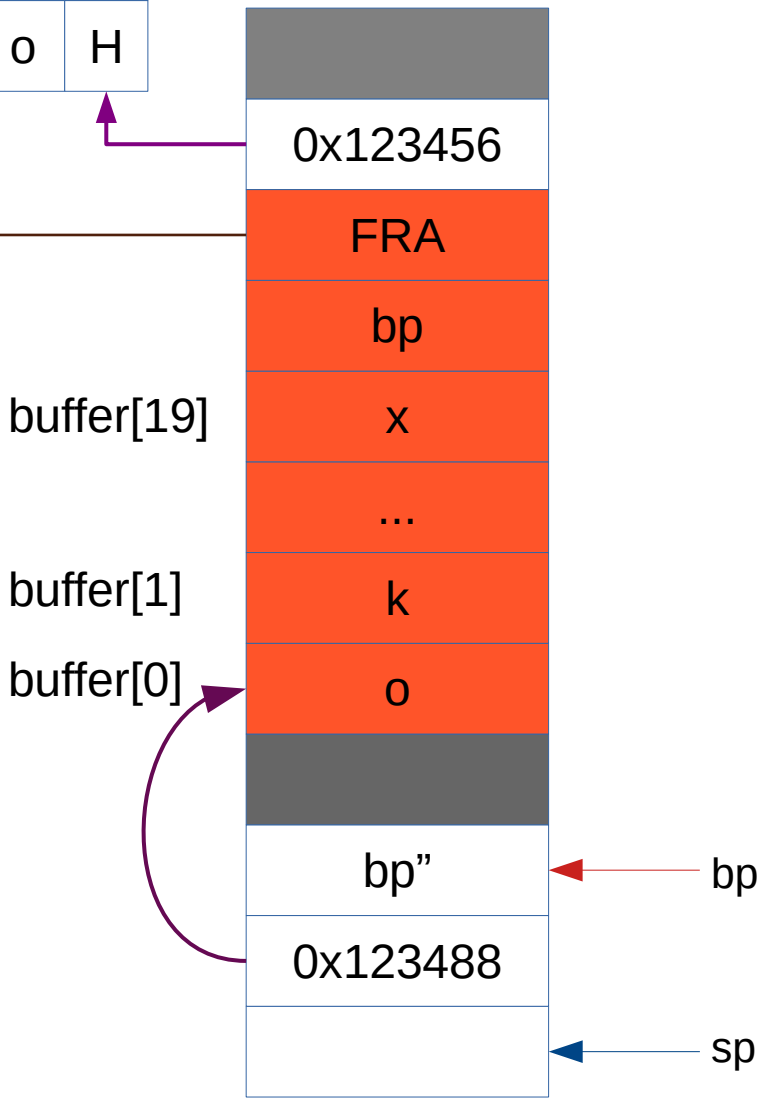


.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   gets(buffer);
   return buffer;
}
```

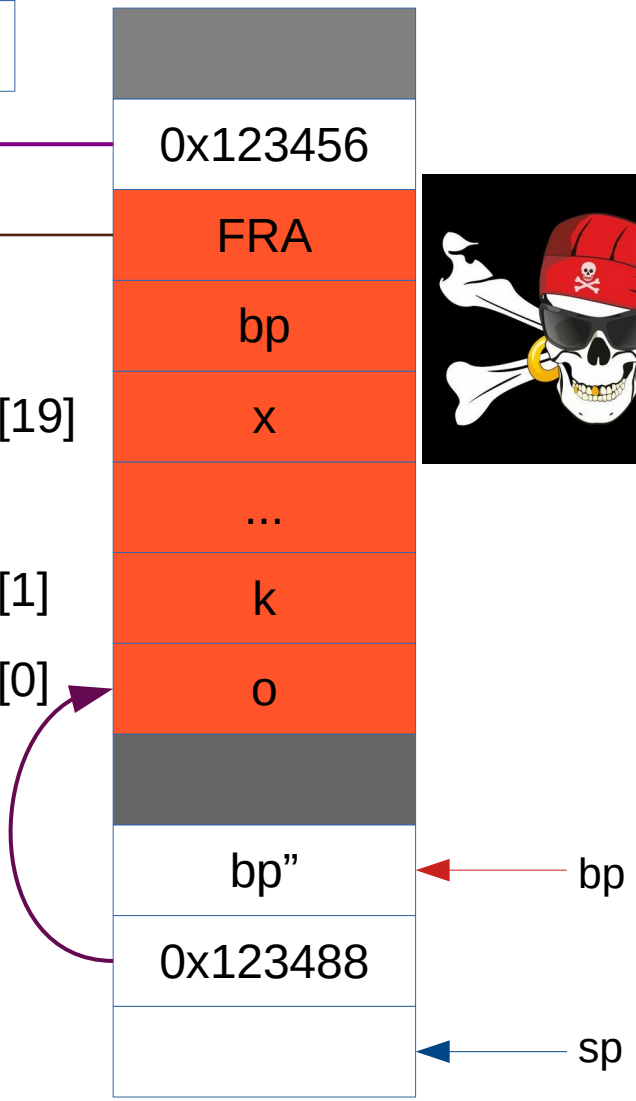


.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
    printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
    char buffer[20];
    printf("%s ", question);
    gets(buffer);
    return buffer;
}
```



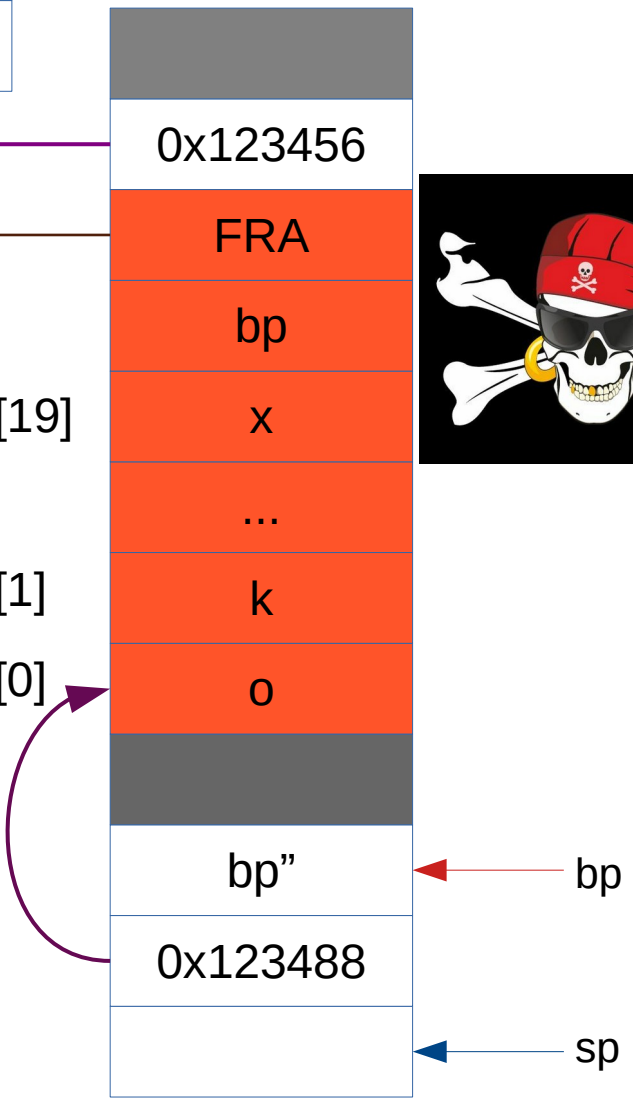
BUFFER OVERFLOW!

.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
    printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
    char buffer[20];
    printf("%s ", question);
    gets(buffer);
    return buffer;
}
```



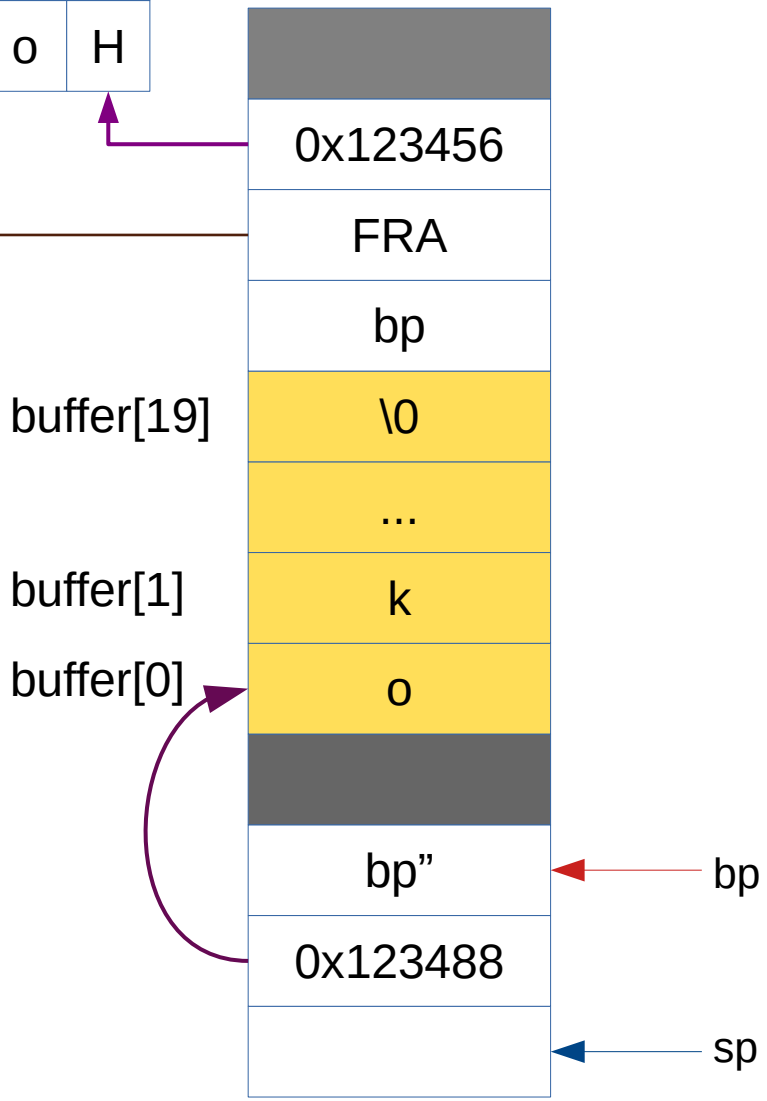
NEVER USE gets( ) !!!

.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
    printf("answer = %s\n",
         answer("How are you?"));
}
```

```
char *answer( char *question)
{
    char buffer[20];
    printf("%s ", question);
    fgets(buffer,20,stdin);
    return buffer;
}
```

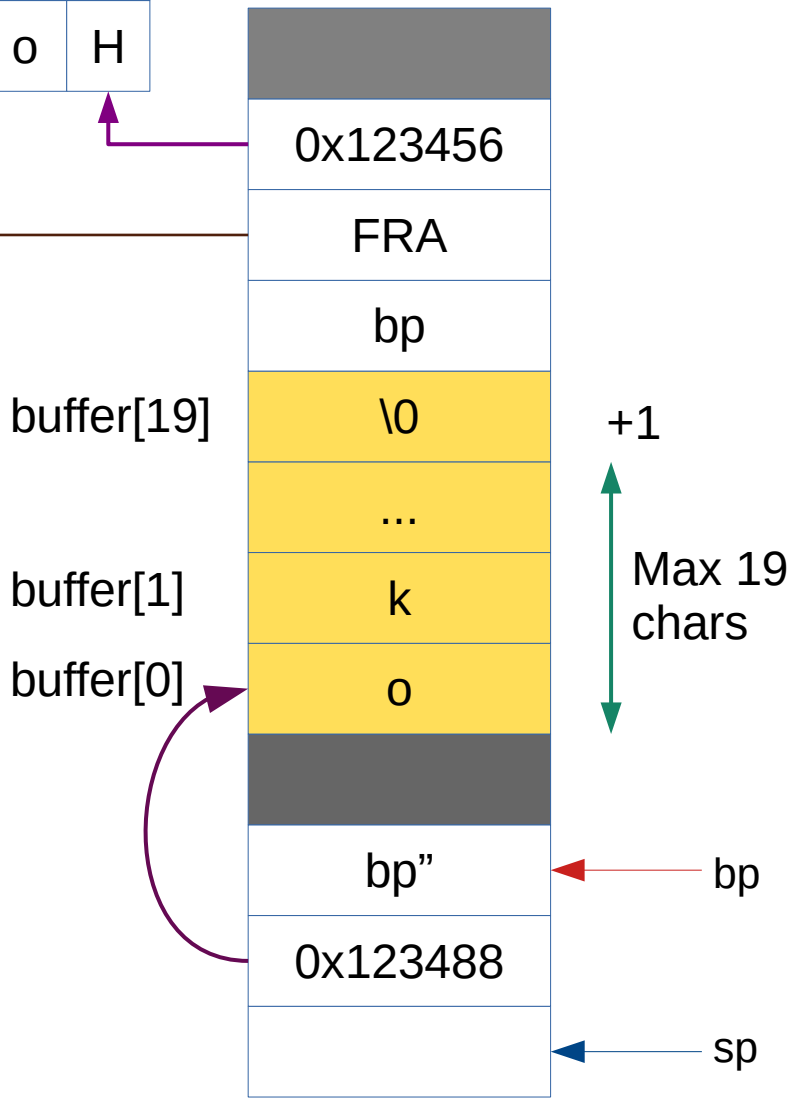


.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
    printf("answer = %s\n",
         answer("How are you?"));
}
```

```
char *answer( char *question)
{
    char buffer[20];
    printf("%s ", question);
    fgets(buffer,20,stdin);
    return buffer;
}
```

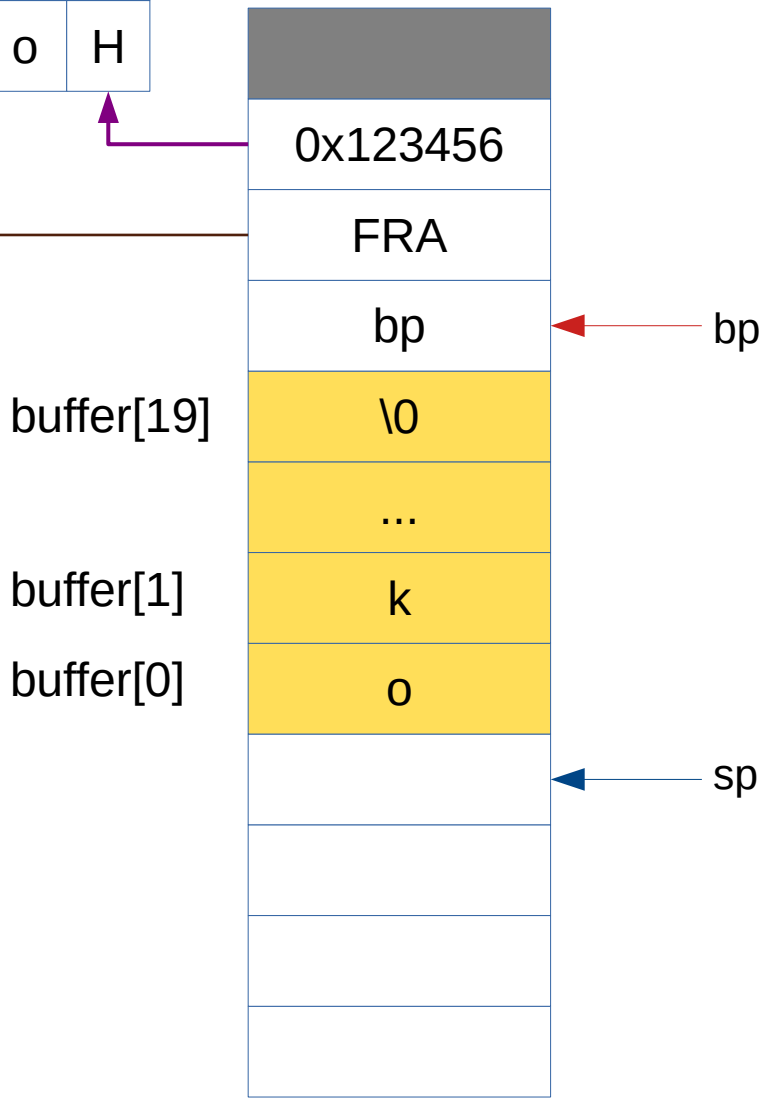


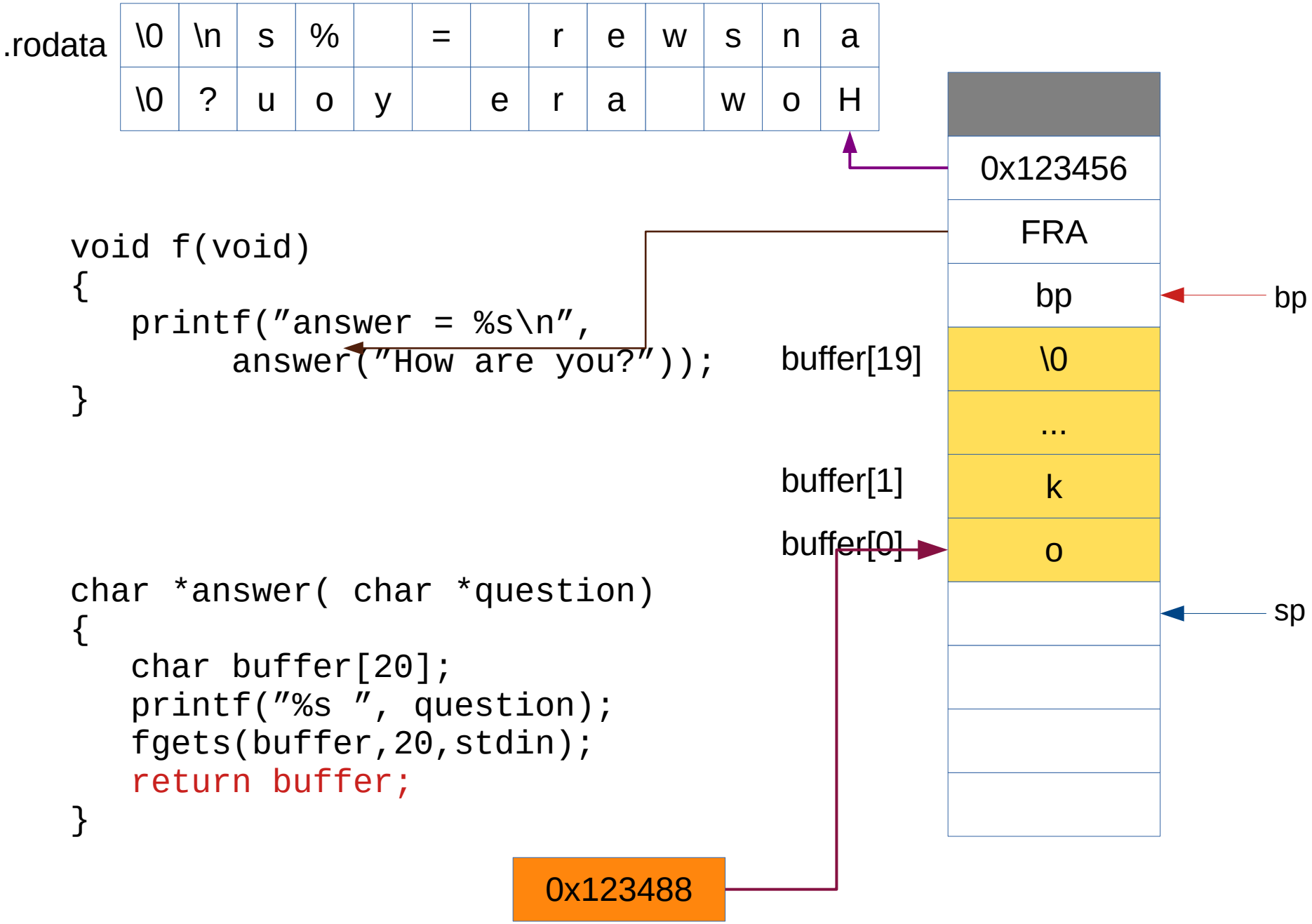
.rodata

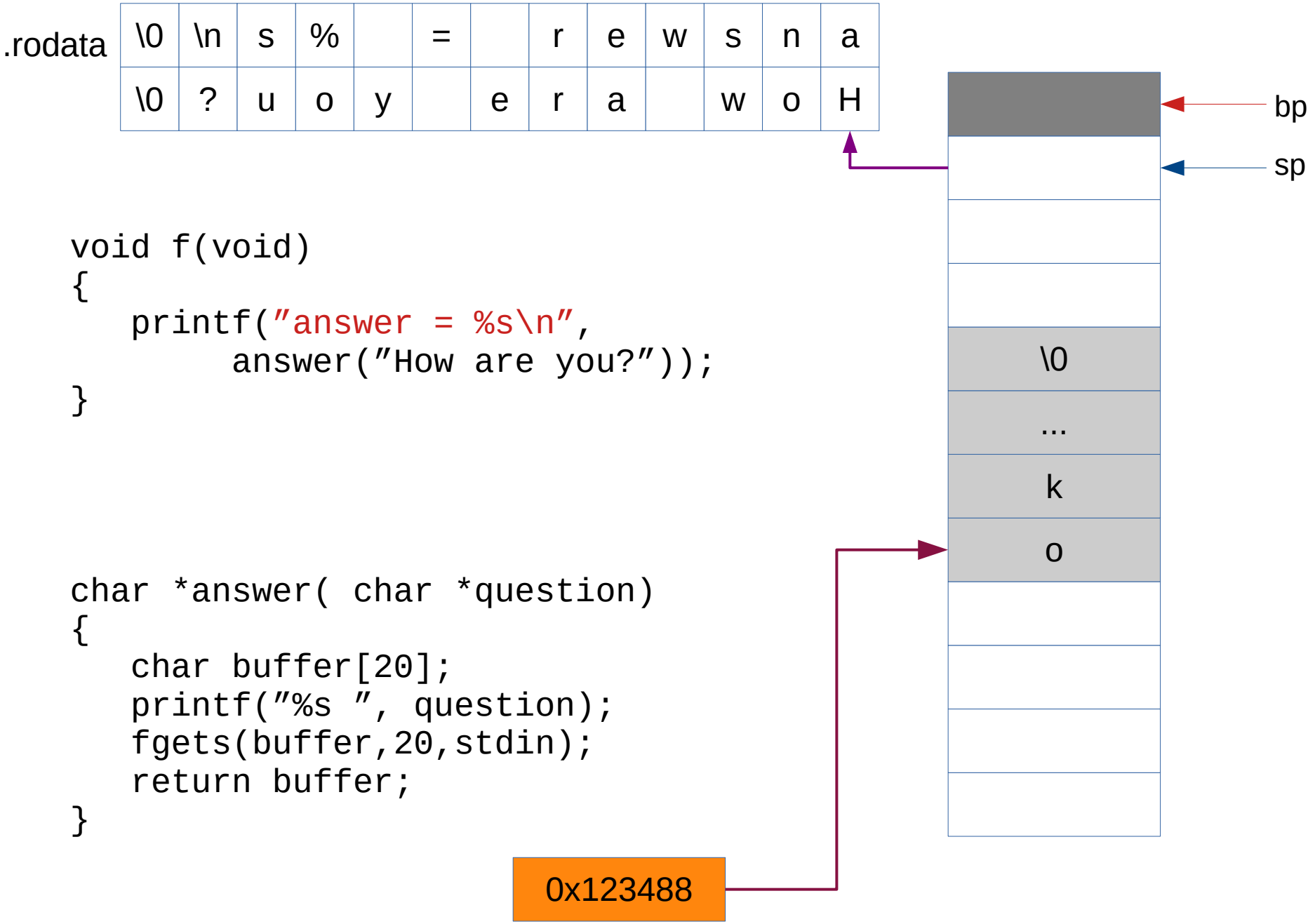
\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```




.rodata
\0 \n s % = r e w s n a
\0 ? u o y e r a w o H
0x123456
FRA
bp
bp
buffer[19]
\0
...
buffer[1]
k
buffer[0]
o
sp
0x123488
void f(void)
{
printf("answer = %s\n",
answer("How are you?"));
}
char *answer(char *question)
{
char buffer[20];
printf("%s ", question);
fgets(buffer,20,stdin);
return buffer;
}




```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}
```

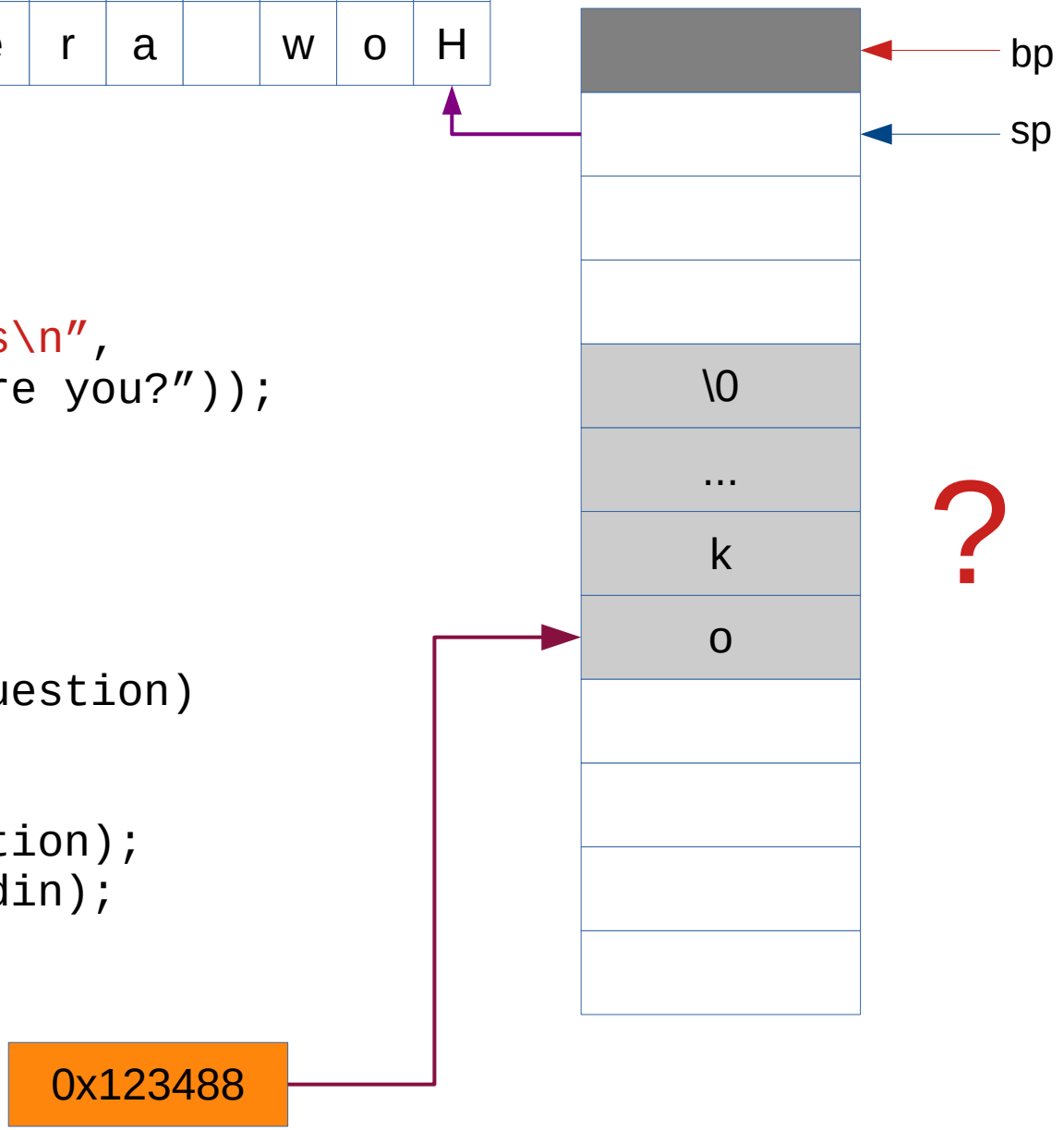
```
char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```

.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}
```

```
char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```

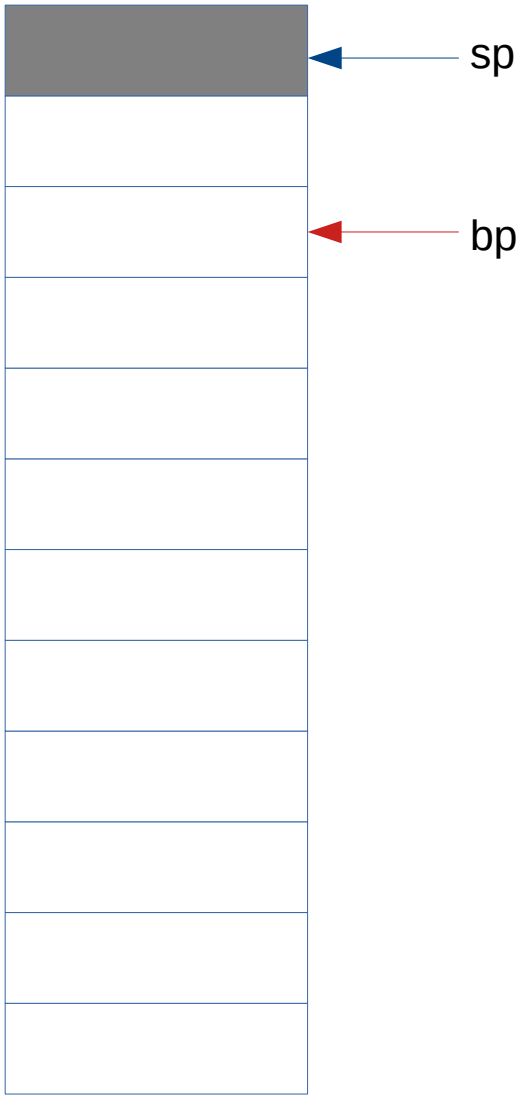


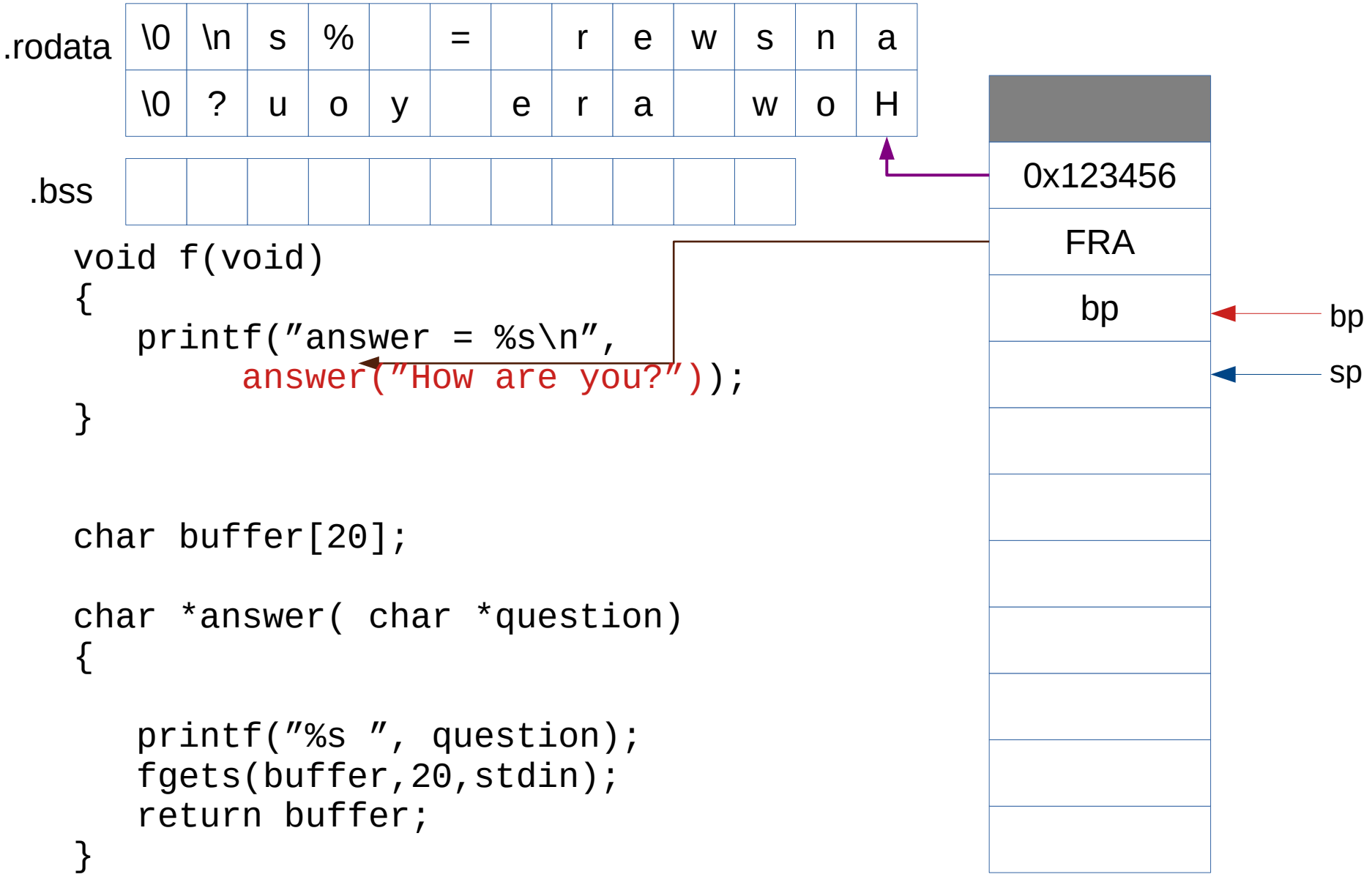
.rodata

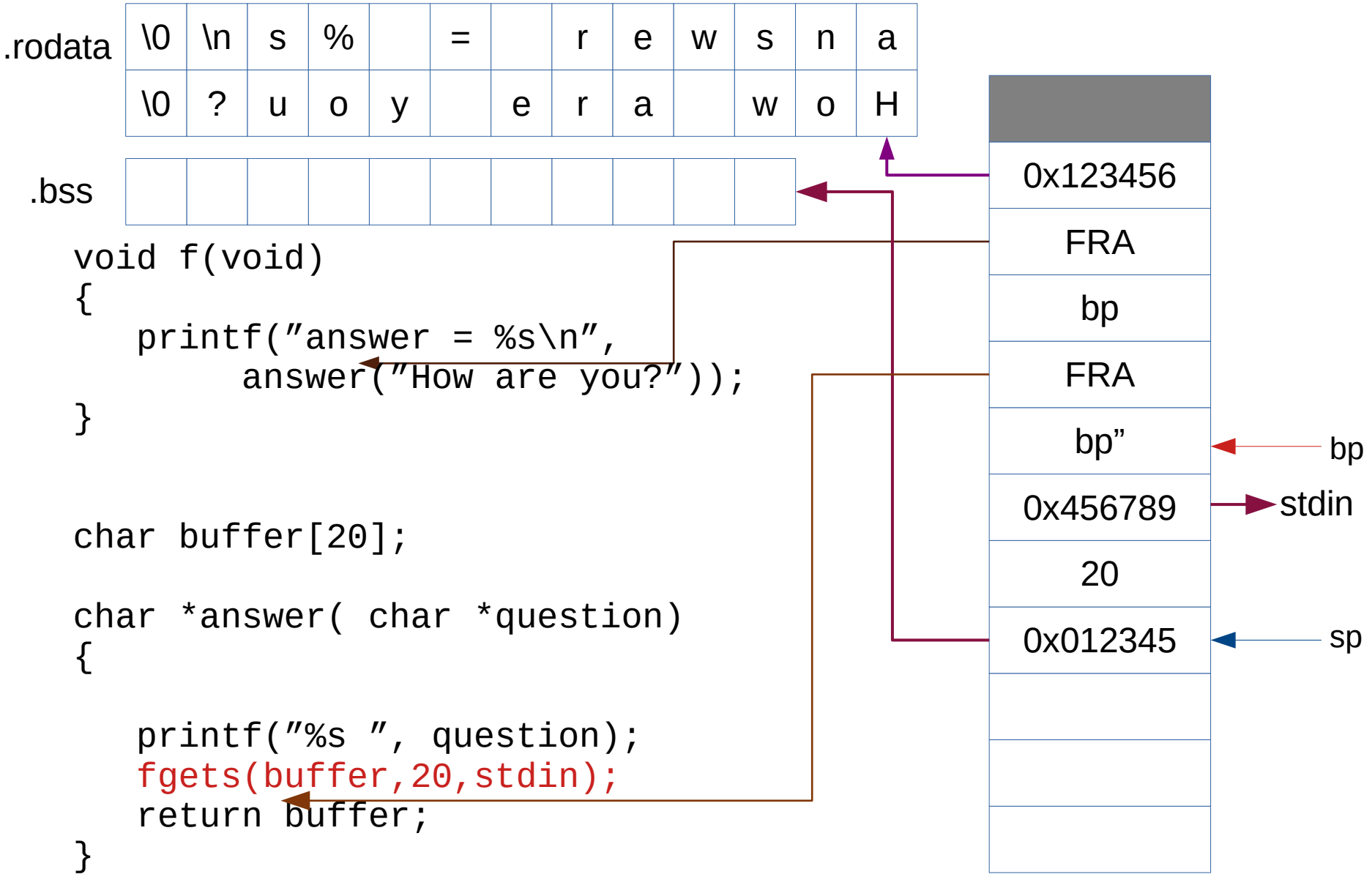
\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

.bss

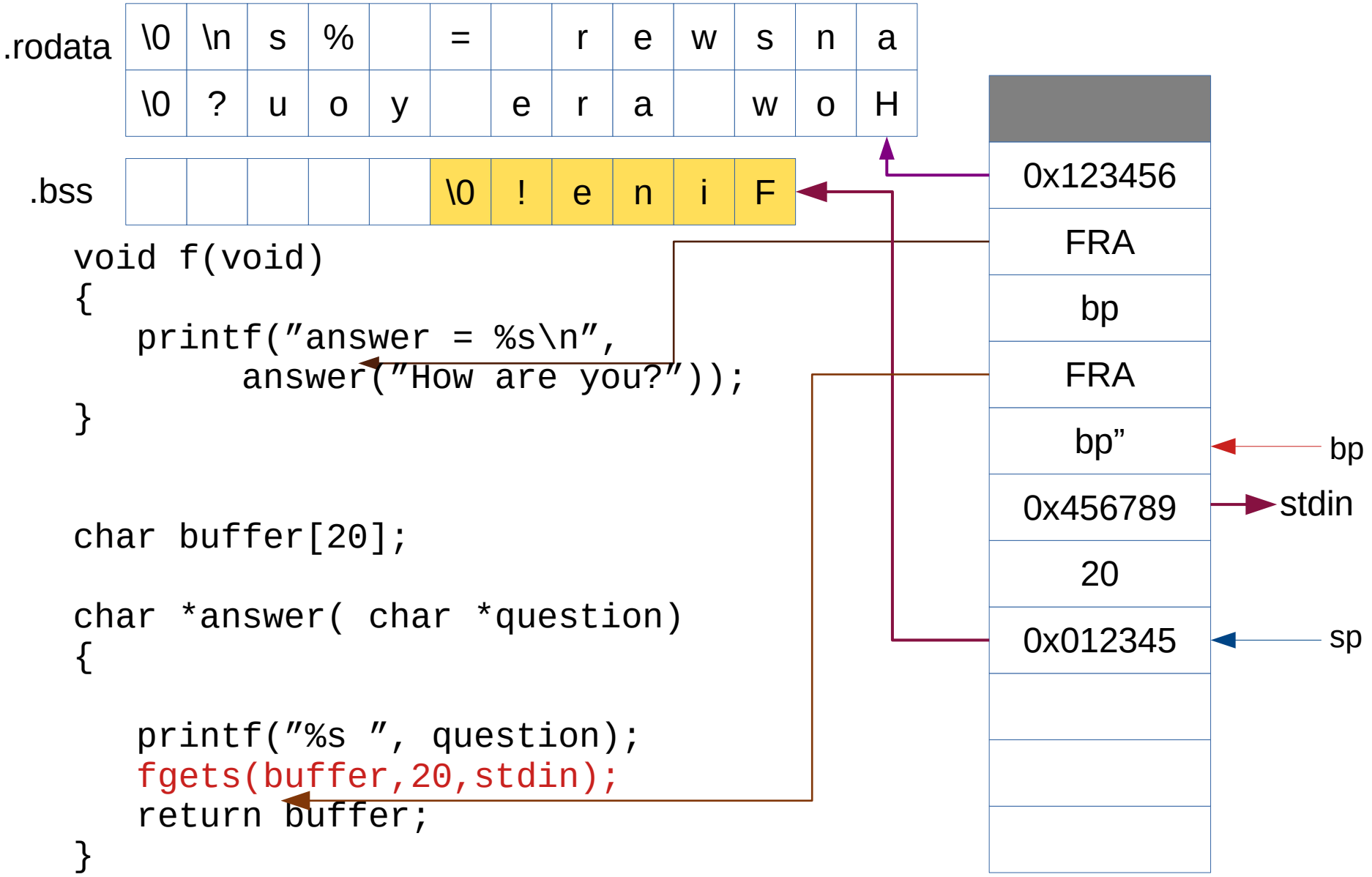

```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}


char buffer[20];

char *answer( char *question)
{
   char buffer[20];
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```

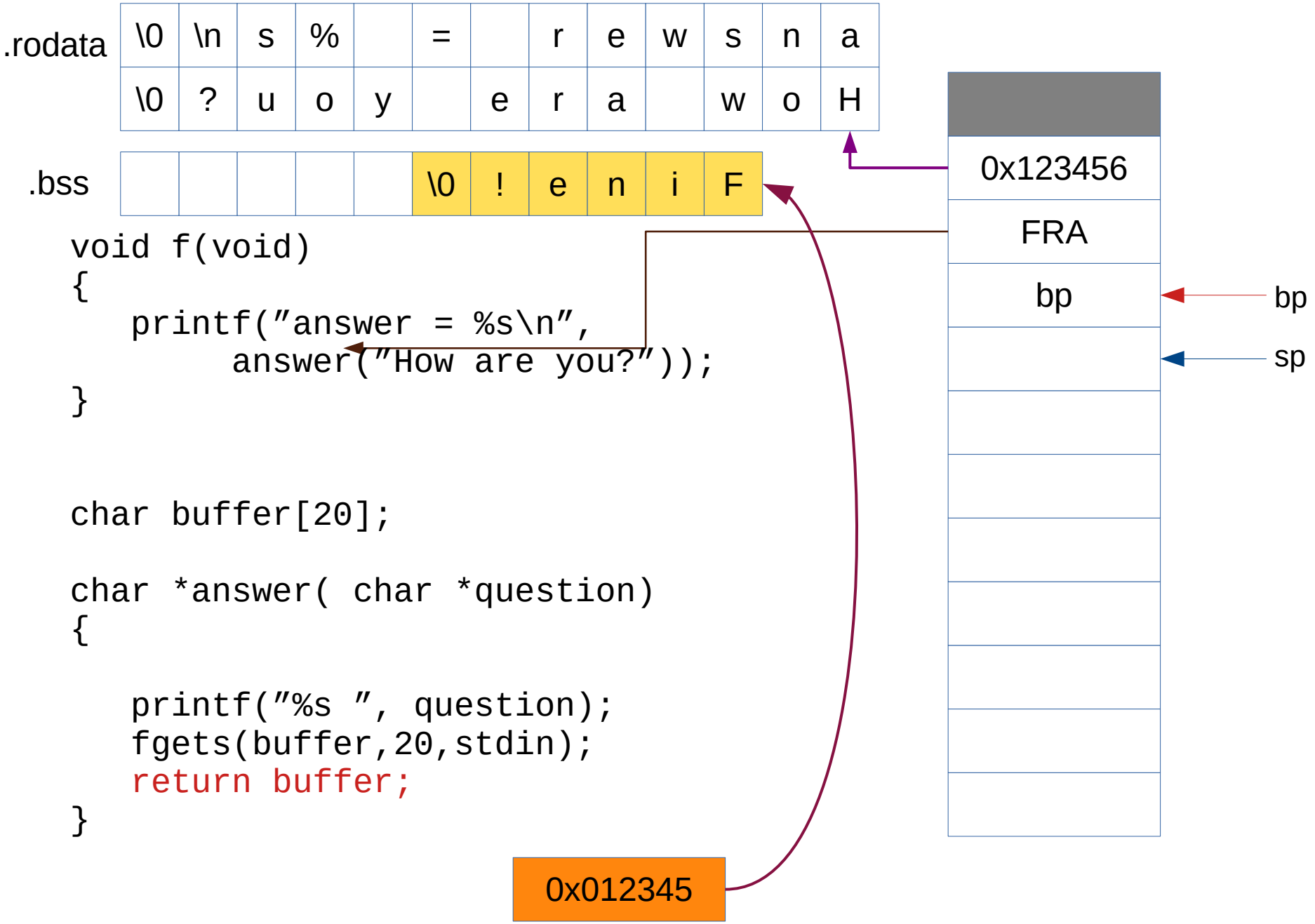



.rodata
\0 \n s % = r e w s n a
\0 ? u o y e r a w o H
.bss
0x123456
FRA
bp
bp
sp
void f(void)
{
printf("answer = %s\n",
answer("How are you?"));
}
char buffer[20];
char *answer(char *question)
{
printf("%s ", question);
fgets(buffer,20,stdin);
return buffer;
}

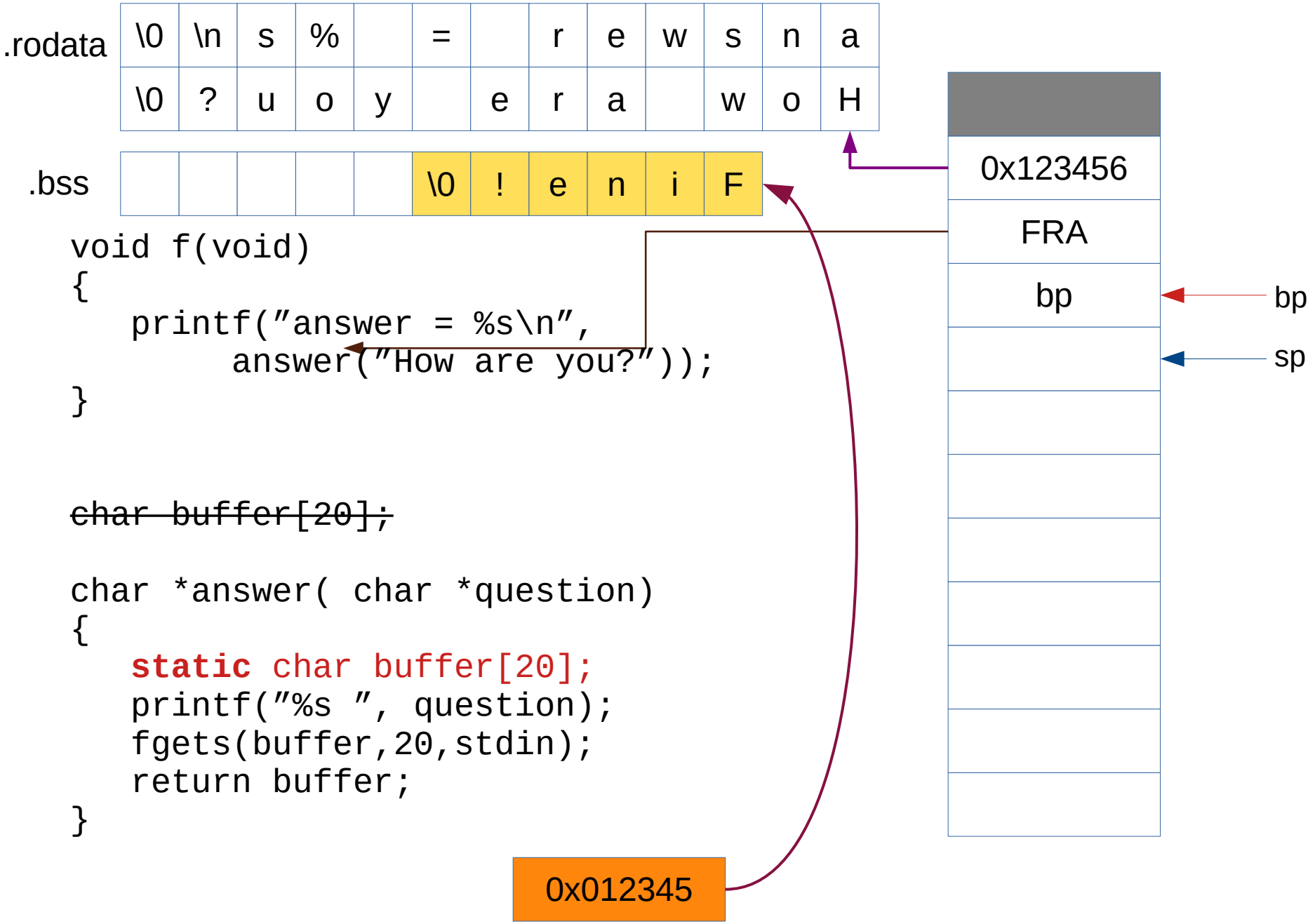



```
void f(void)
{
    printf(”answer = %s\n”,
        answer(”How are you?”));
}


char buffer[20];

char *answer( char *question)
{

    printf(”%s ”, question);
    fgets(buffer,20,stdin);
    return buffer;
}
```


.rodata
\0 \n s % = r e w s n a
\0 ? u o y e r a w o H
.bss
\0 ! e n i F
0x123456
FRA
bp
FRA
bp”
0x456789
20
0x012345
bp
stdin
sp
void f(void)
{
printf(”answer = %s\n”,
answer(”How are you?”));
}
char buffer[20];
char *answer(char *question)
{
printf(”%s ”, question);
fgets(buffer,20,stdin);
return buffer;
}




```
void f(void)
{
   printf("answer = %s\n",
        answer("How are you?"));
}


char buffer[20];

char *answer( char *question)
{

   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



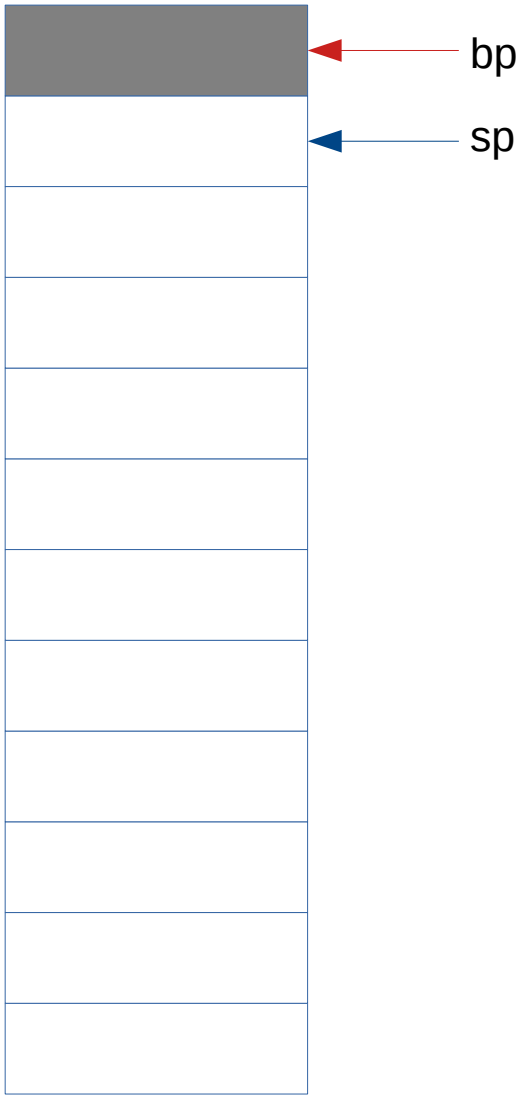

```
void f(void)
{
    printf("answer = %s\n",
        answer("How are you?"));
}


char buffer[20];

char *answer( char *question)
{
    static char buffer[20];
    printf("%s ", question);
    fgets(buffer,20,stdin);
    return buffer;
}
```

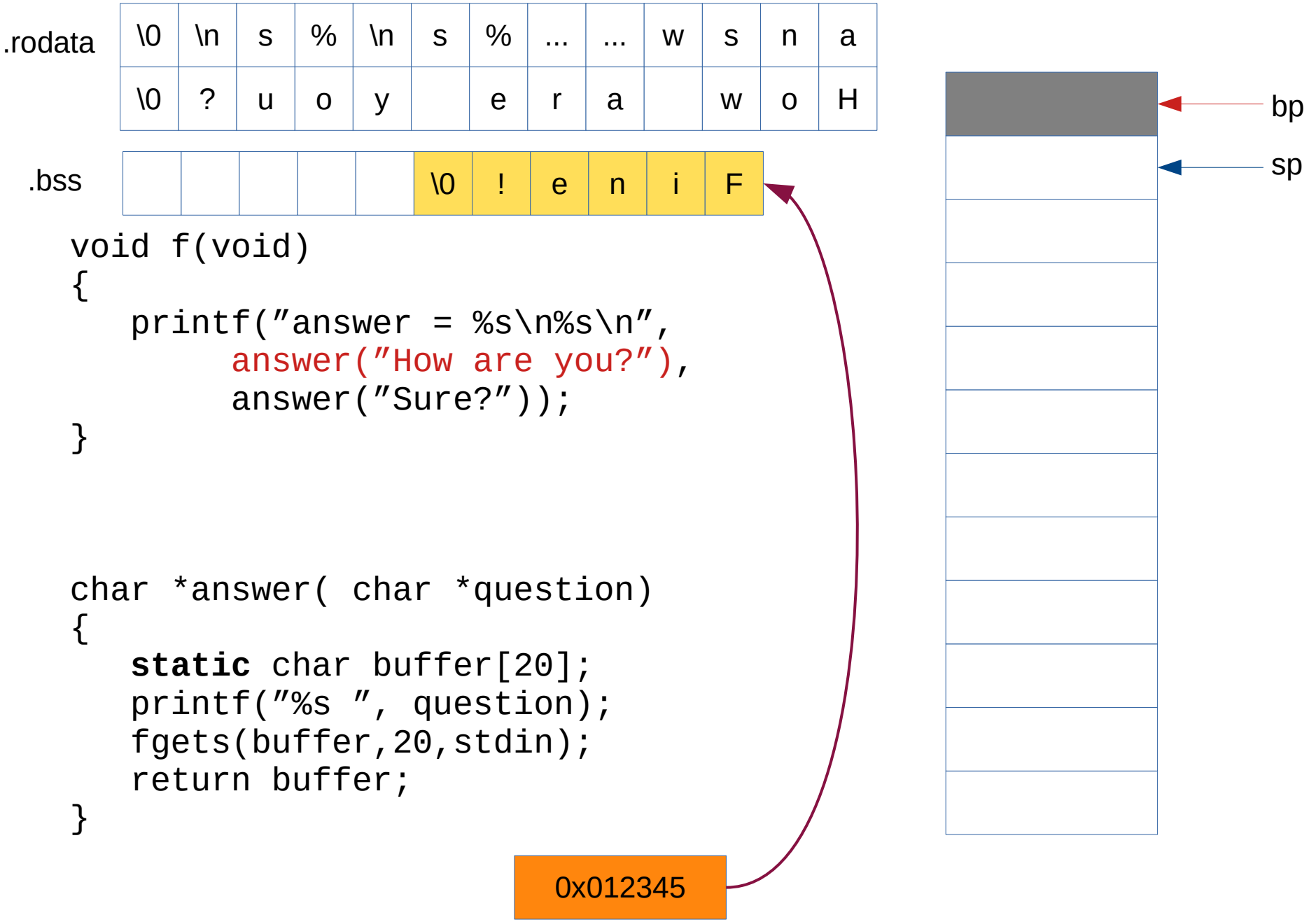
.rodata

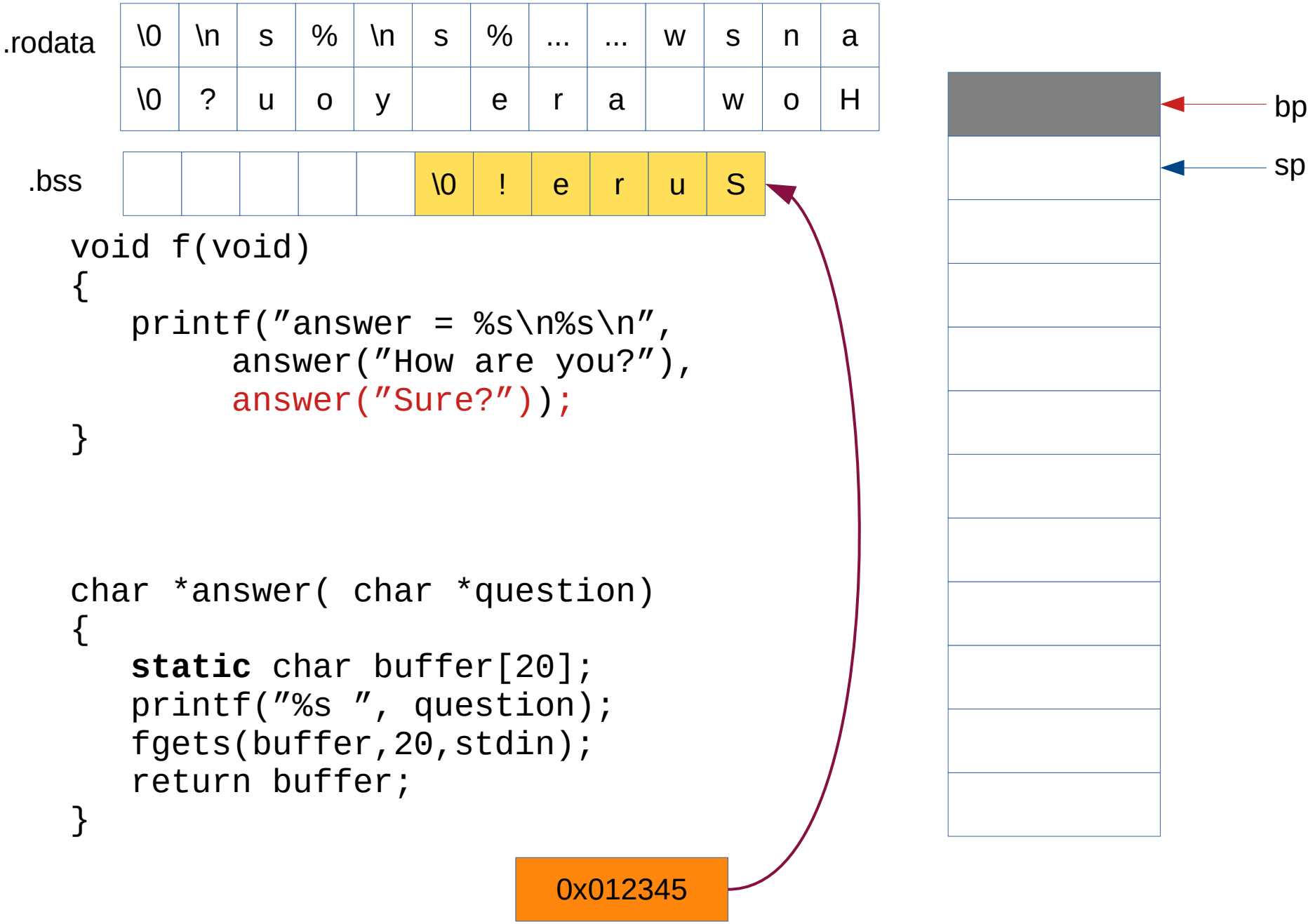
\0	\n	s	%	\n	s	%	...	...	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

.bss

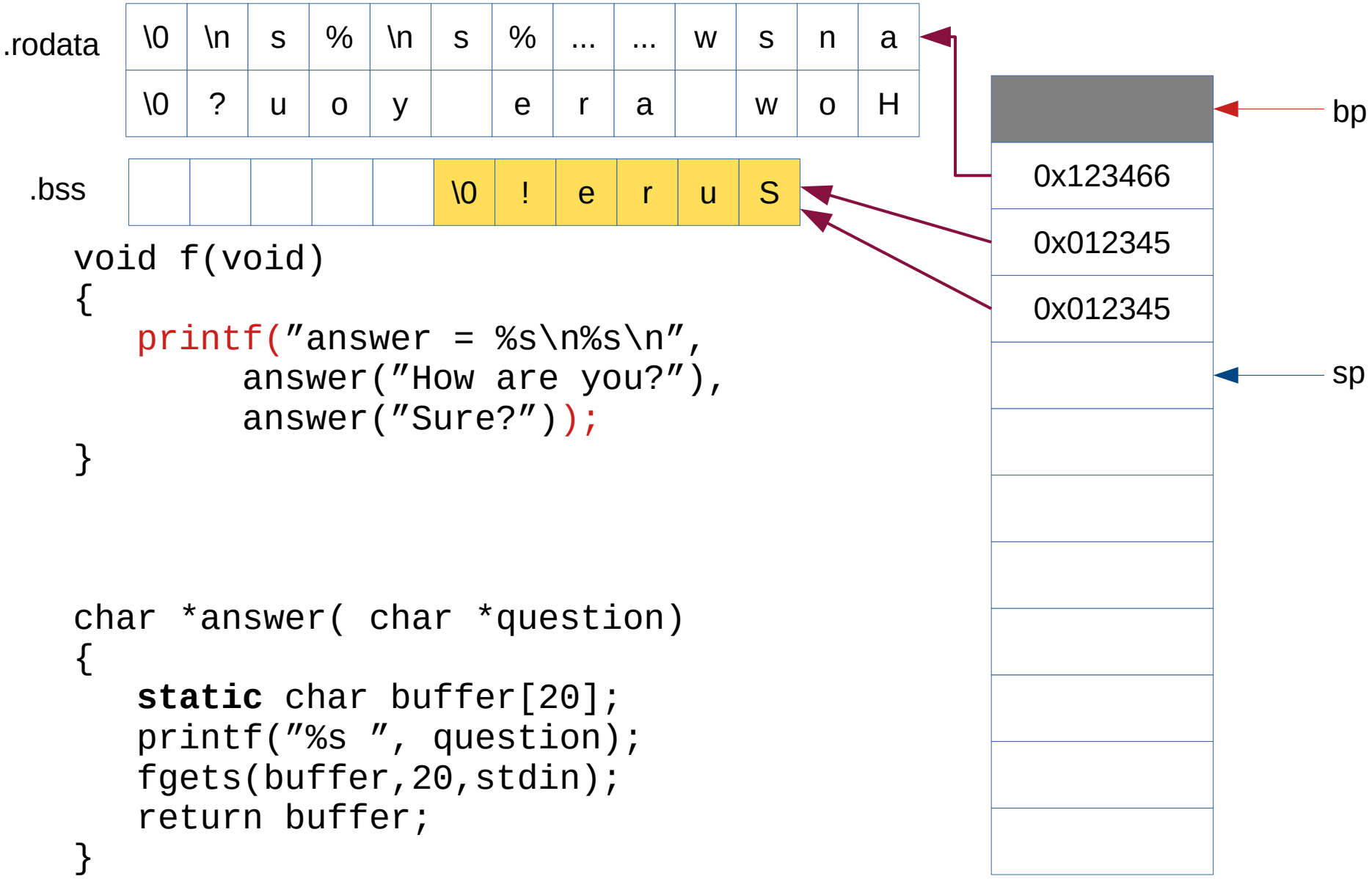

```
void f(void)
{
   printf("answer = %s\n%s\n",
        answer("How are you?"),
        answer("Sure?"));
}



char *answer( char *question)
{
   static char buffer[20];
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```




.rodata
\0 \n s % \n s % ... ... w s n a
\0 ? u o y e r a w o H
.bss
\0 ! e n i F
bp
sp
void f(void)
{
printf("answer = %s\n%s\n",
answer("How are you?"),
answer("Sure?"));
}
char *answer(char *question)
{
static char buffer[20];
printf("%s ", question);
fgets(buffer,20,stdin);
return buffer;
}
0x012345

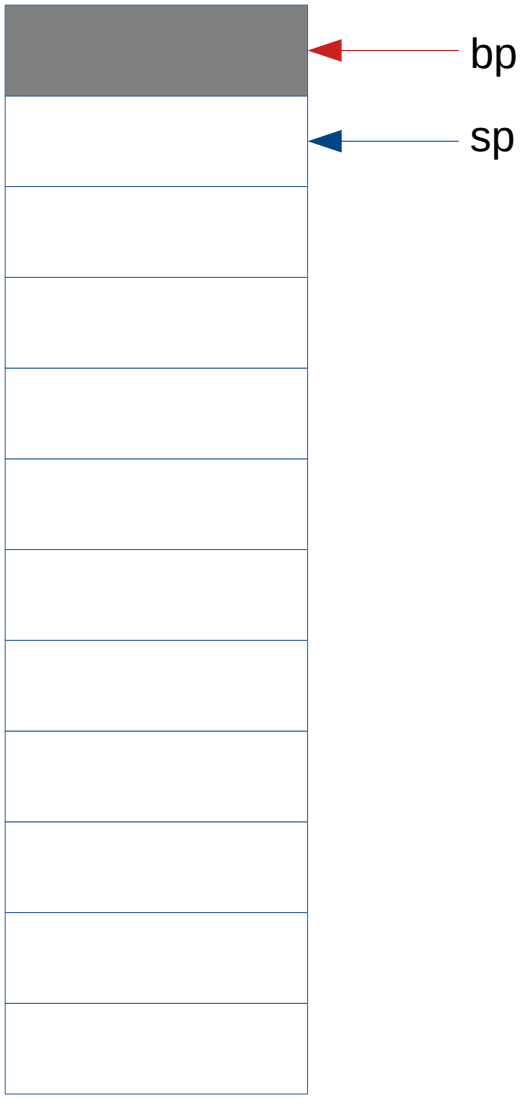



```
void f(void)
{
    printf("answer = %s\n%s\n",
        answer("How are you?"),
        answer("Sure?"));
}



char *answer( char *question)
{
    static char buffer[20];
    printf("%s ", question);
    fgets(buffer,20,stdin);
    return buffer;
}
```

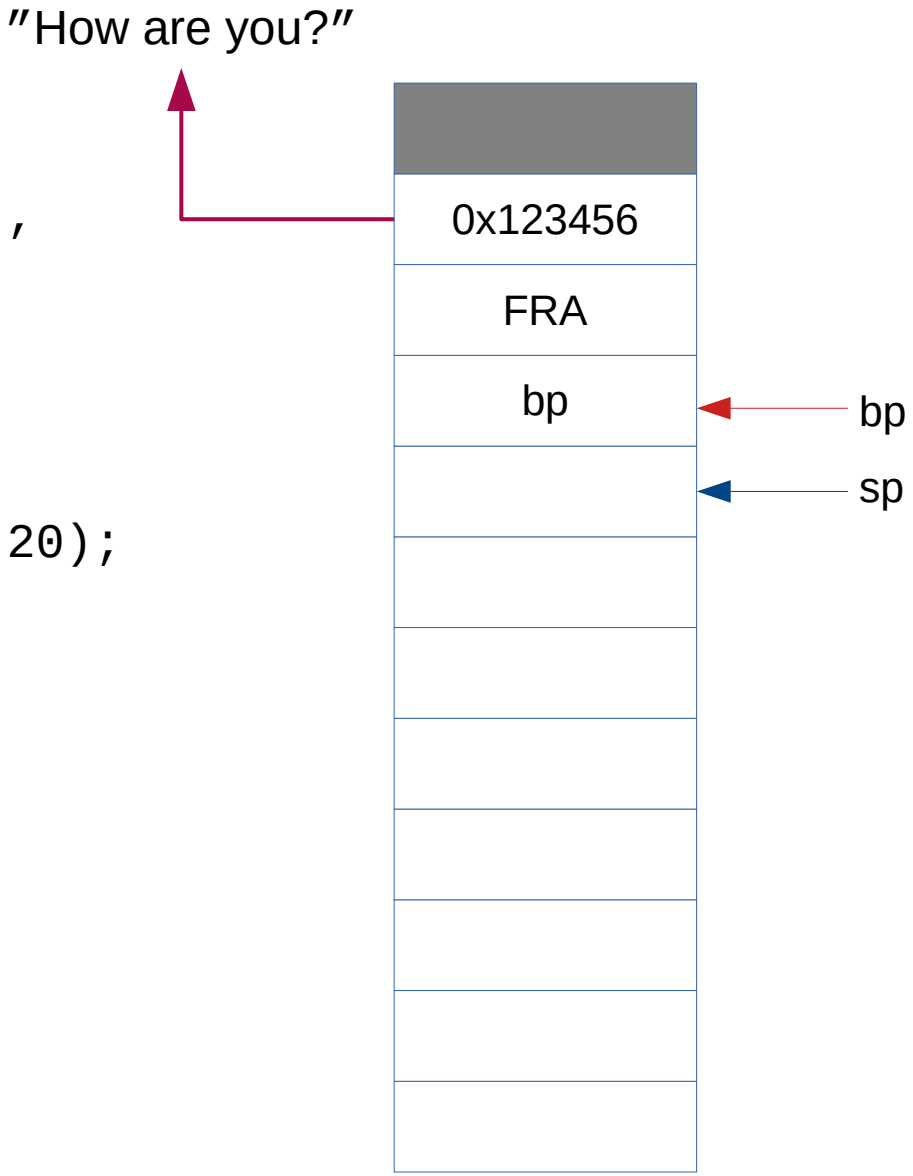



```
void f(void)
{
   printf("answer = %s\n%s\n",
        answer("How are you?"),
        answer("Sure?"));
}



char *answer( char *question)
{
   static char buffer[20];
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```

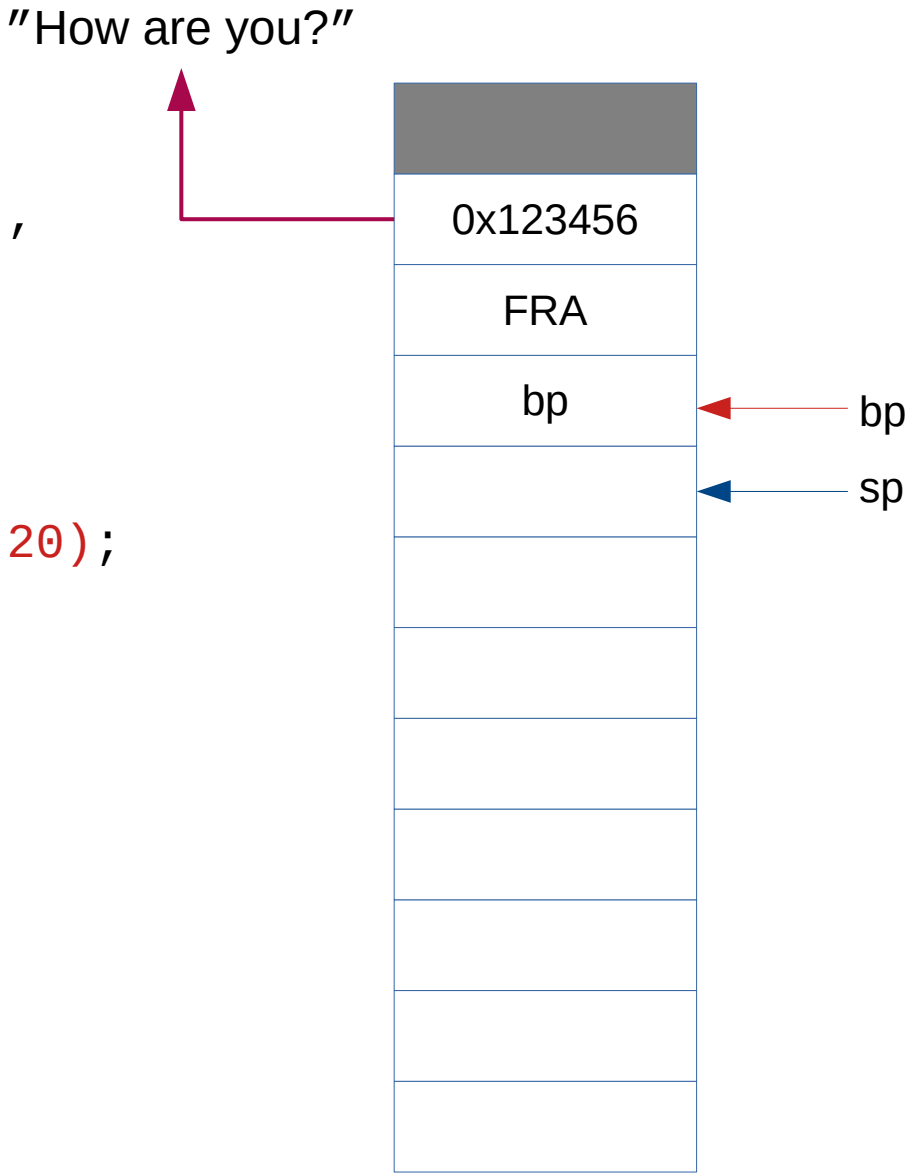
```
void f(void)
{
   printf(”answer = %s\n%s\n”,
        answer(”How are you?”),
        answer(”Sure?”));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return ”No memory”;
   printf(”%s ”, question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



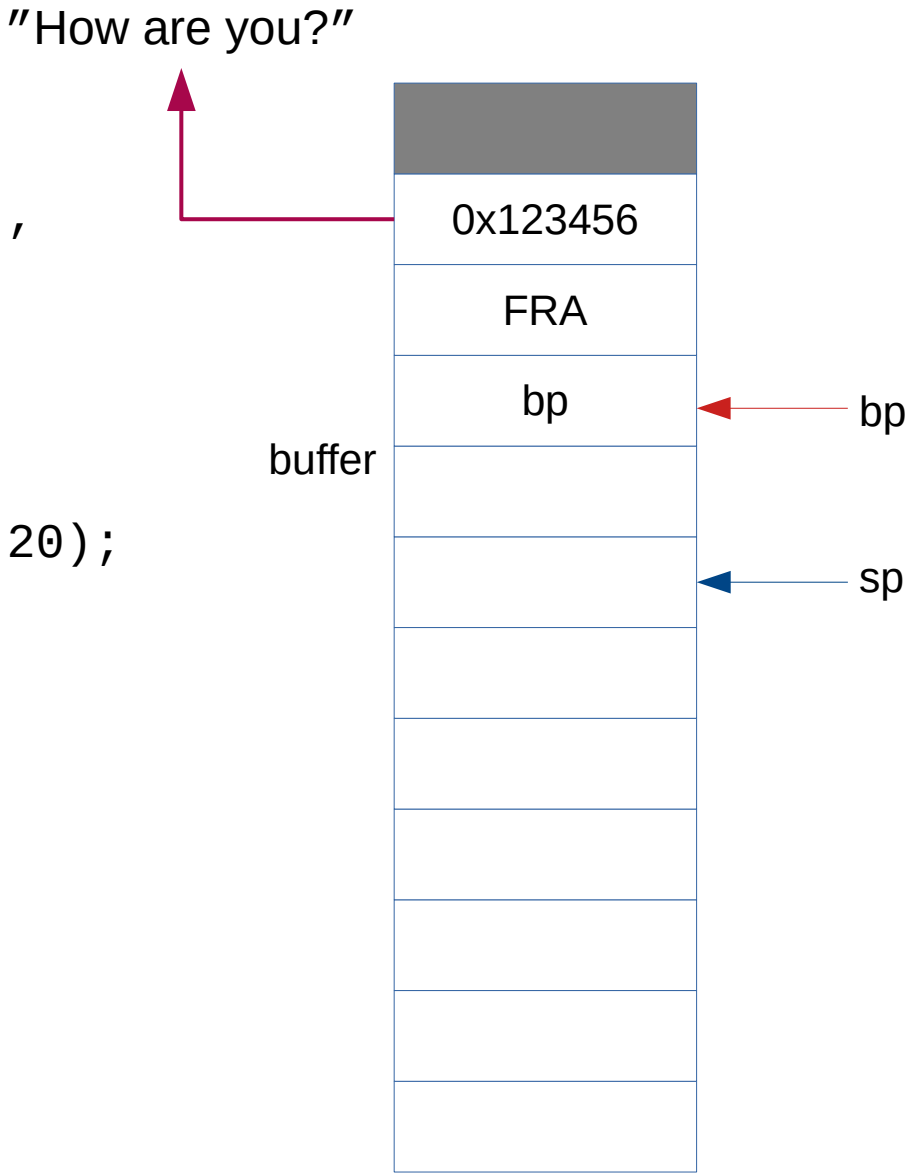
```
void f(void)
{
   printf("answer = %s\n%s\n",
        answer("How are you?"),
        answer("Sure?"));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return "No memory";
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



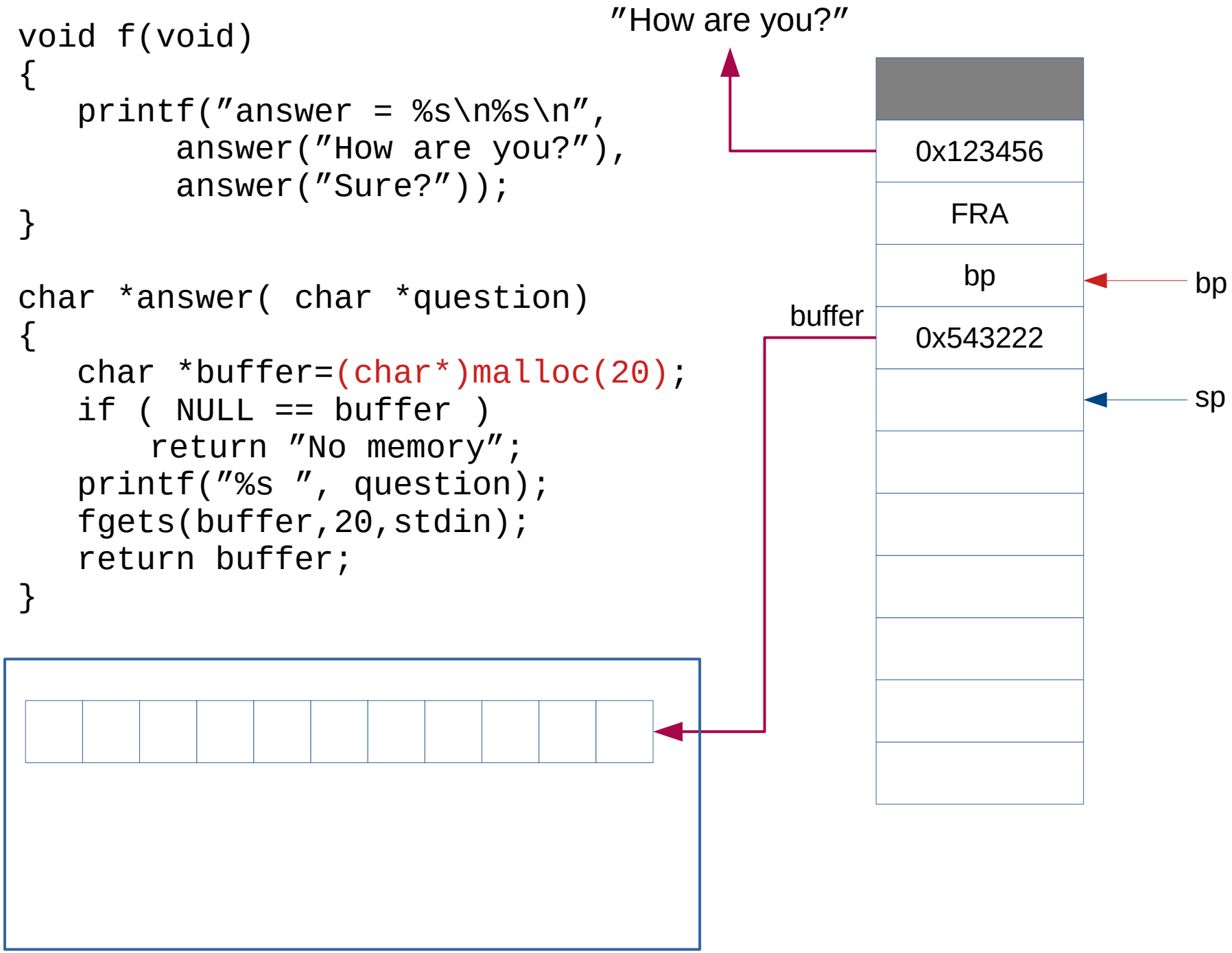
```
void f(void)
{
   printf(”answer = %s\n%s\n”,
        answer(”How are you?”),
        answer(”Sure?”));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return ”No memory”;
   printf(”%s ”, question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



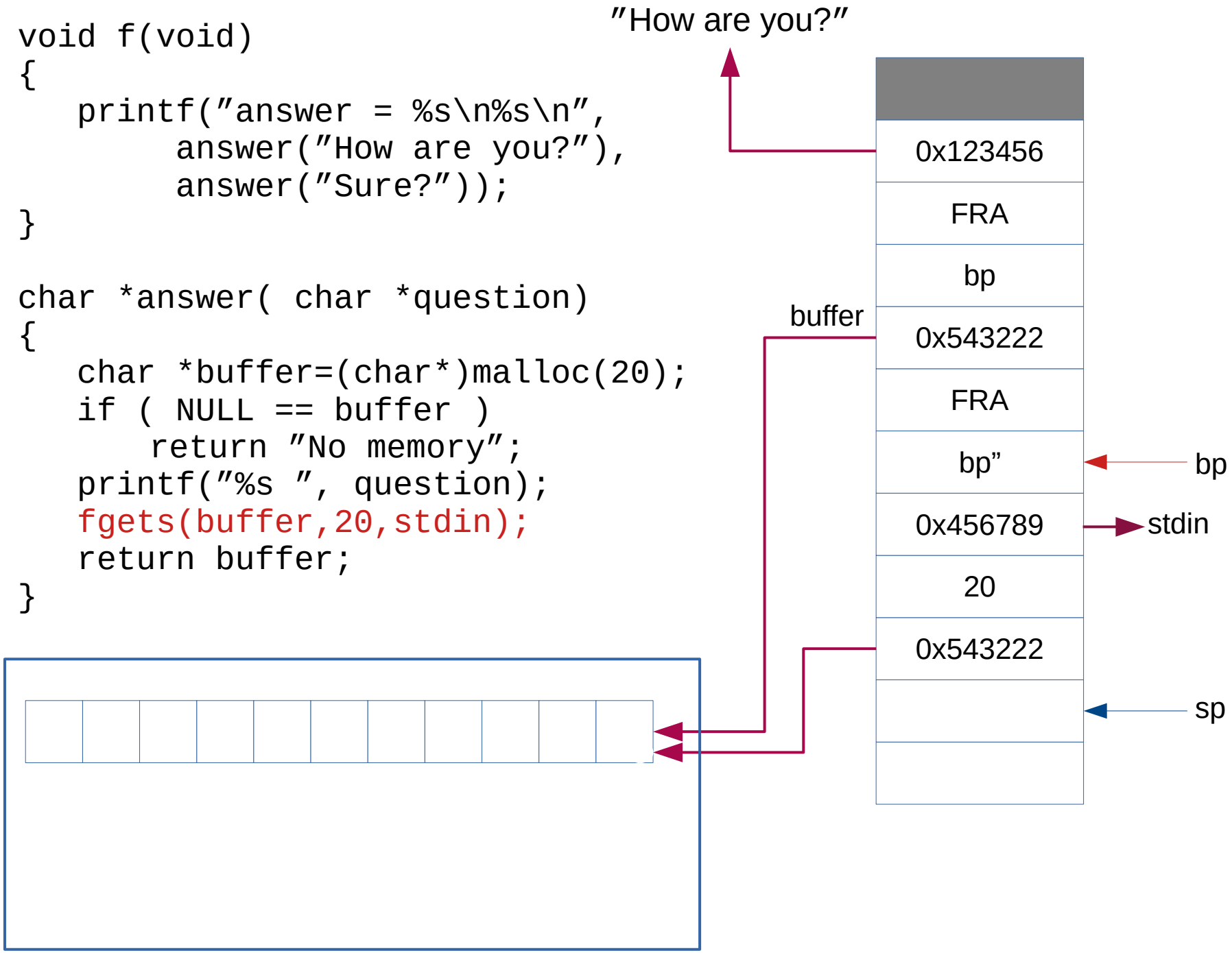
```
void f(void)
{
   printf(”answer = %s\n%s\n”,
        answer(”How are you?”),
        answer(”Sure?”));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return ”No memory”;
   printf(”%s ”, question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



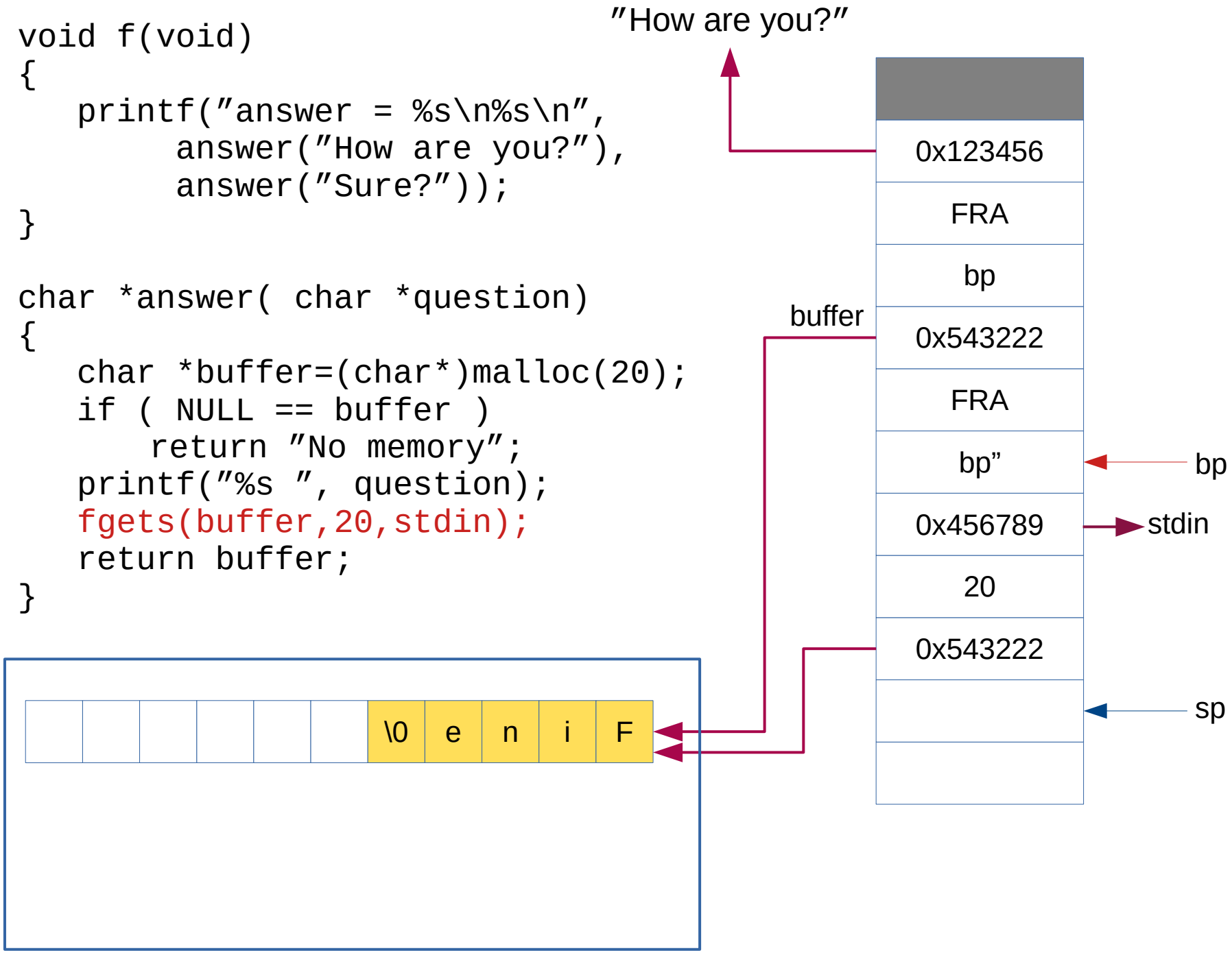
```
void f(void)
{
   printf("answer = %s\n%s\n",
        answer("How are you?"),
        answer("Sure?"));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return "No memory";
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



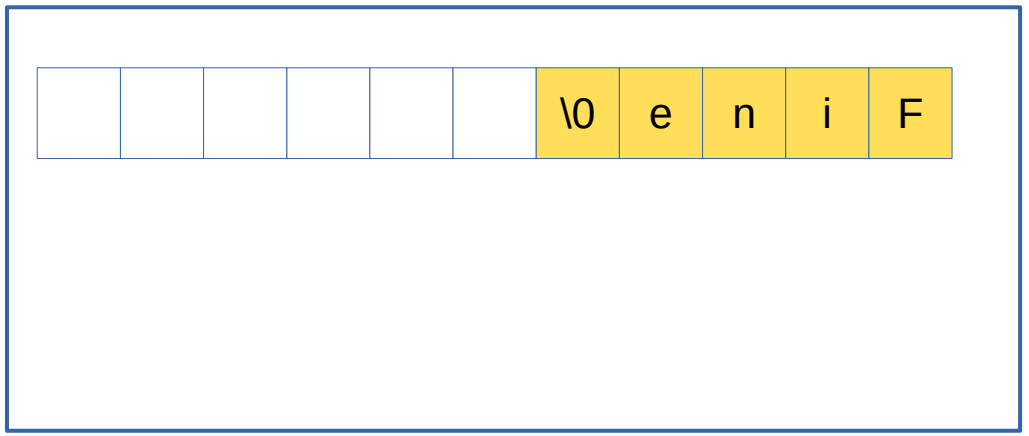
```
void f(void)
{
   printf(”answer = %s\n%s\n”,
        answer(”How are you?”),
        answer(”Sure?”));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return ”No memory”;
   printf(”%s ”, question);
   fgets(buffer,20,stdin);
   return buffer;
}
```

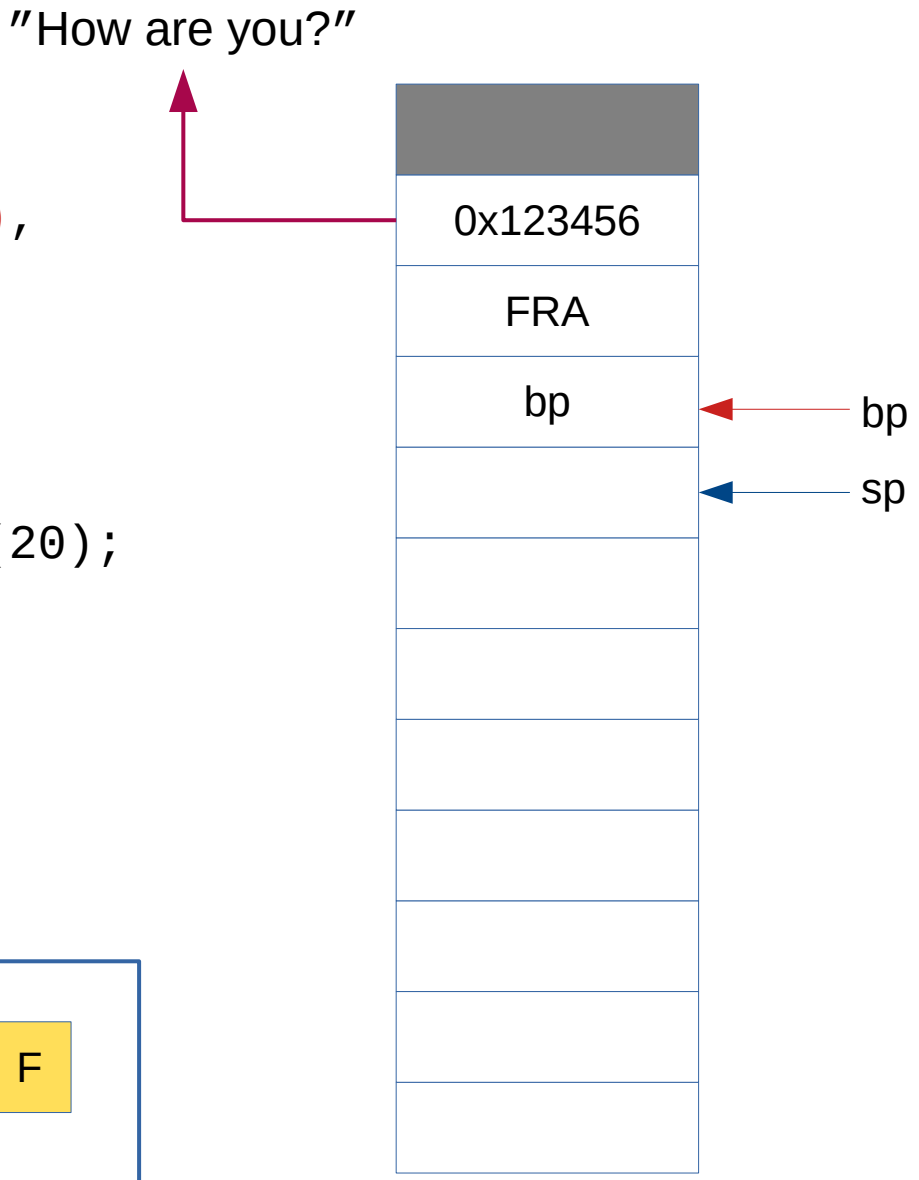


```
void f(void)
{
   printf("answer = %s\n%s\n",
        answer("How are you?"),
        answer("Sure?"));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return "No memory";
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```

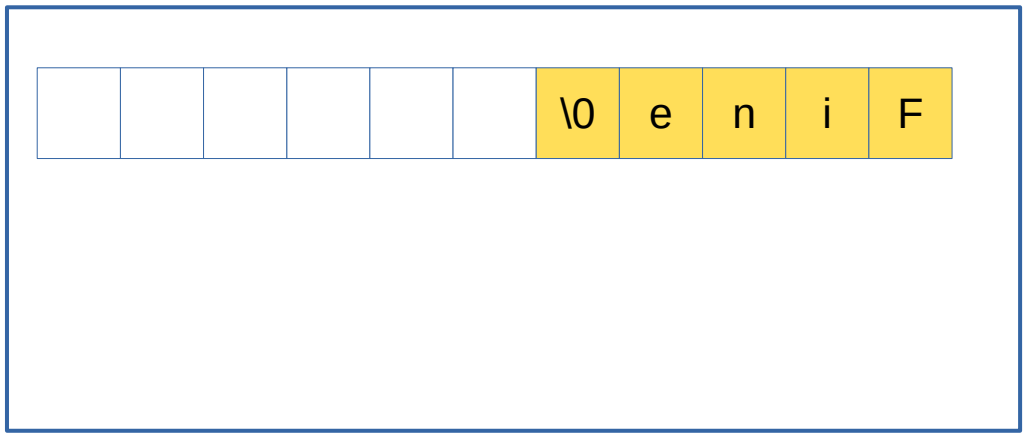


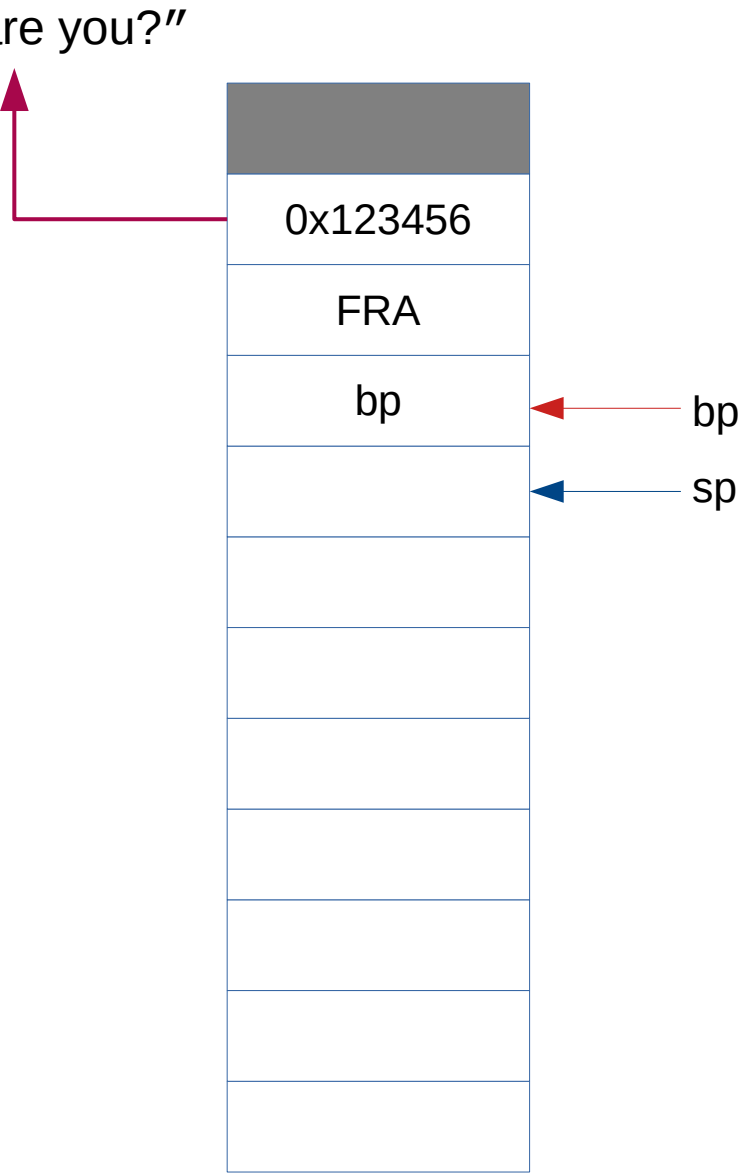
```
void f(void)
{
   printf(”answer = %s\n%s\n”,
        answer(”How are you?”),
        answer(”Sure?”));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return ”No memory”;
   printf(”%s ”, question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



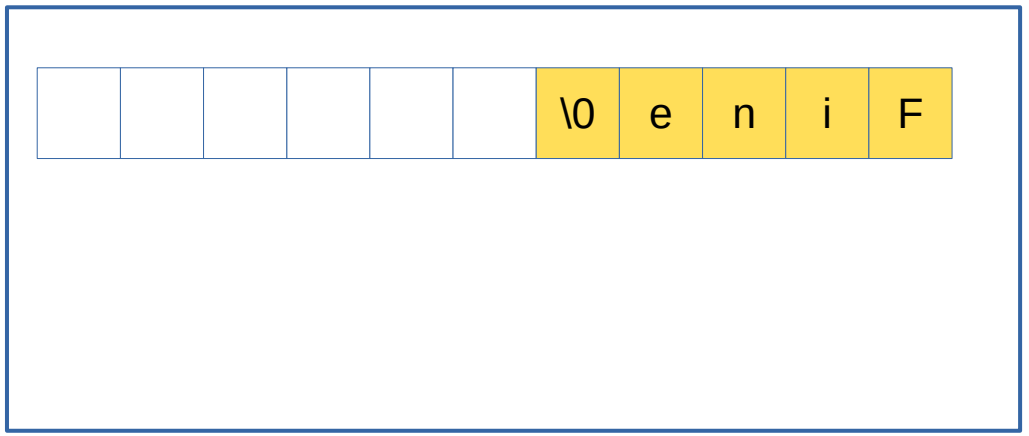


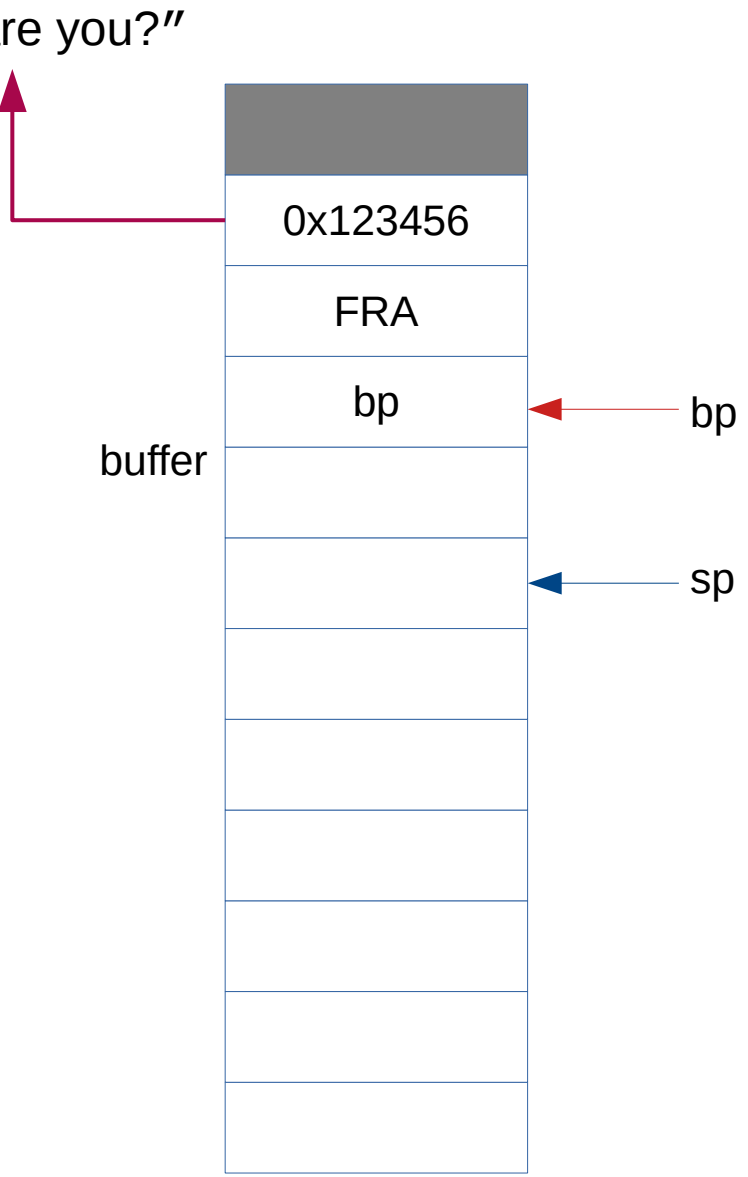
```
void f(void)
{
   printf(”answer = %s\n%s\n”,
        answer(”How are you?”),
        answer(”Sure?”));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return ”No memory”;
   printf(”%s ”, question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



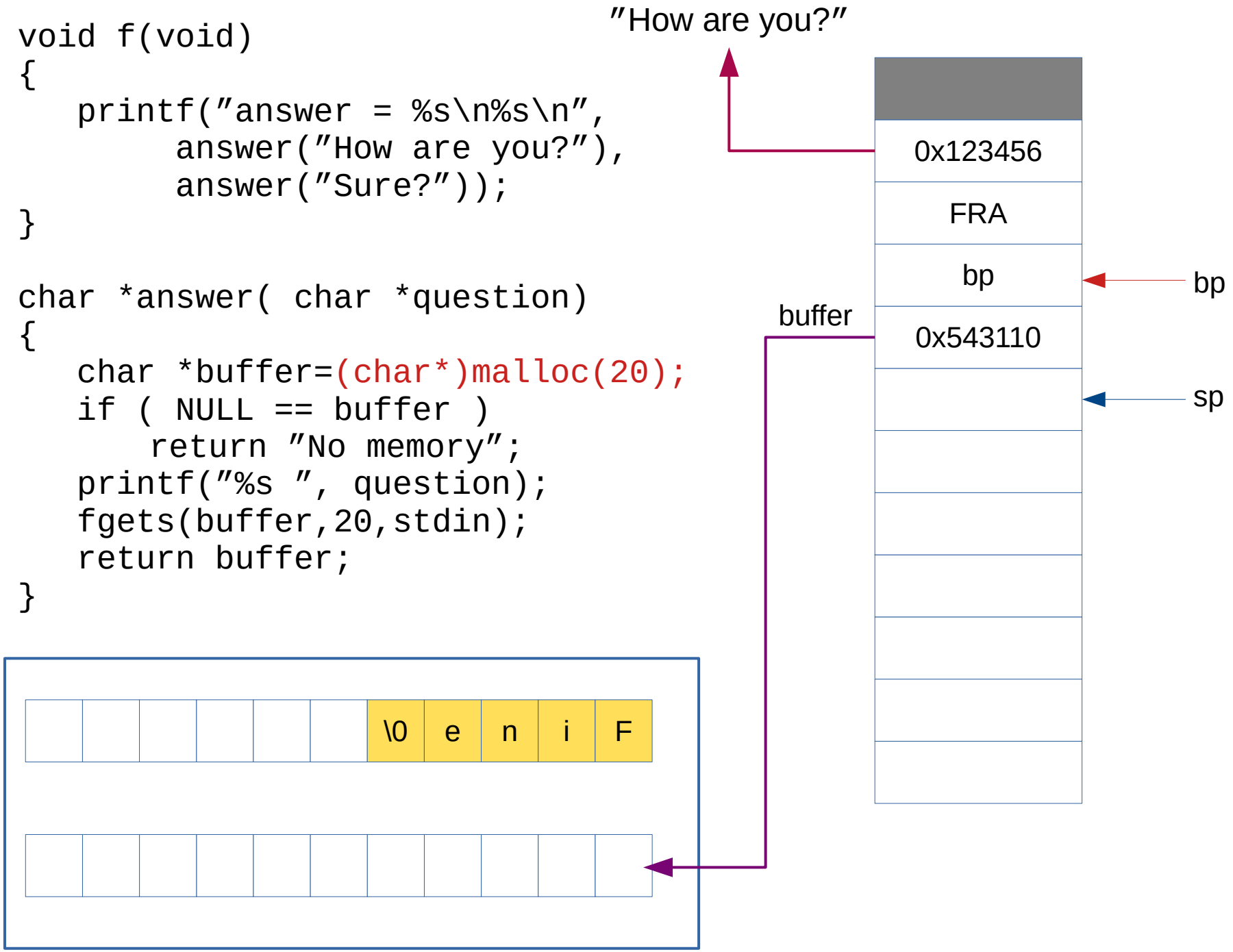


```
void f(void)
{
   printf(”answer = %s\n%s\n”,
        answer(”How are you?”),
        answer(”Sure?”));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return ”No memory”;
   printf(”%s ”, question);
   fgets(buffer,20,stdin);
   return buffer;
}
```

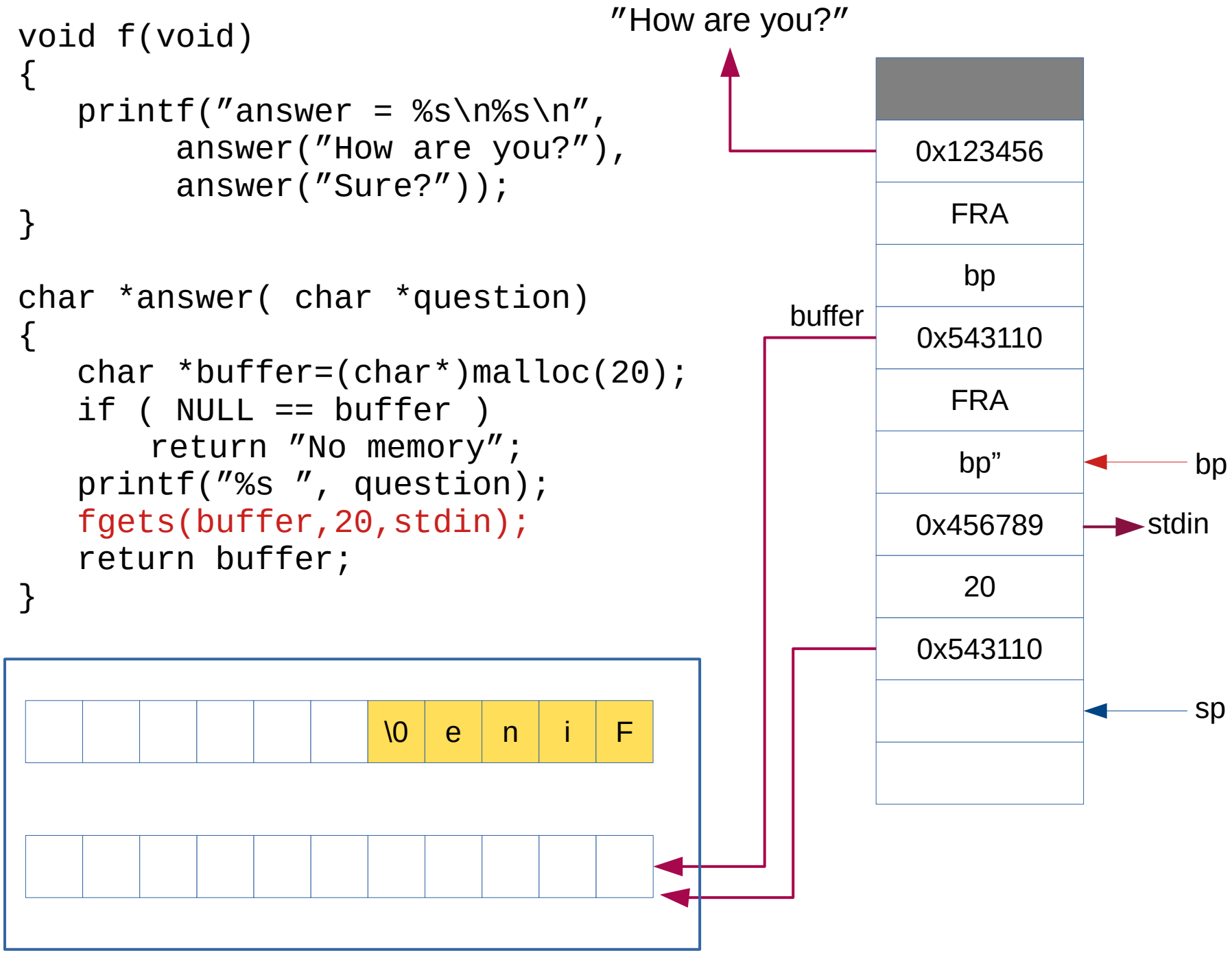




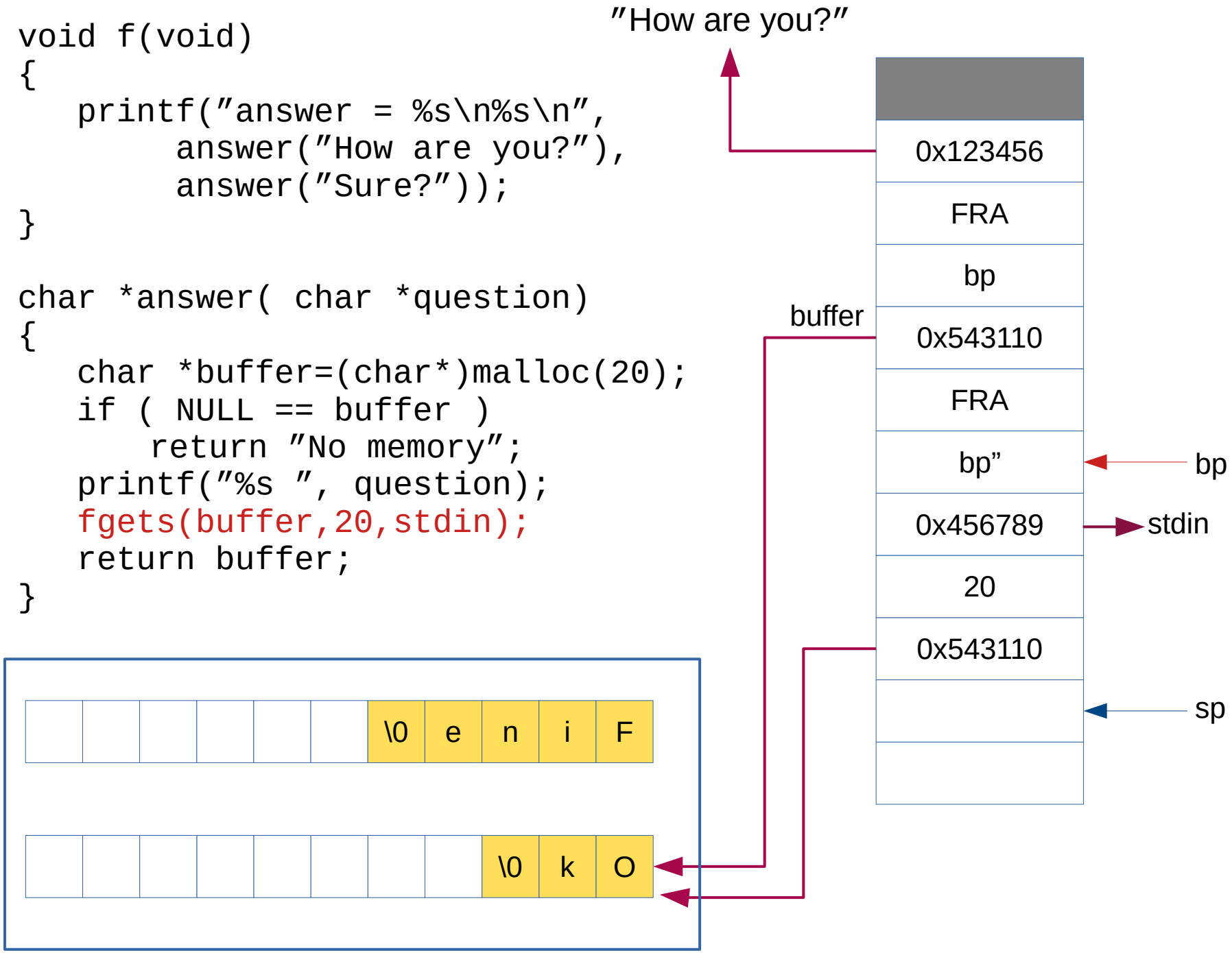
```
void f(void)
{
   printf(”answer = %s\n%s\n”,
        answer(”How are you?”),
        answer(”Sure?”));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return ”No memory”;
   printf(”%s ”, question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



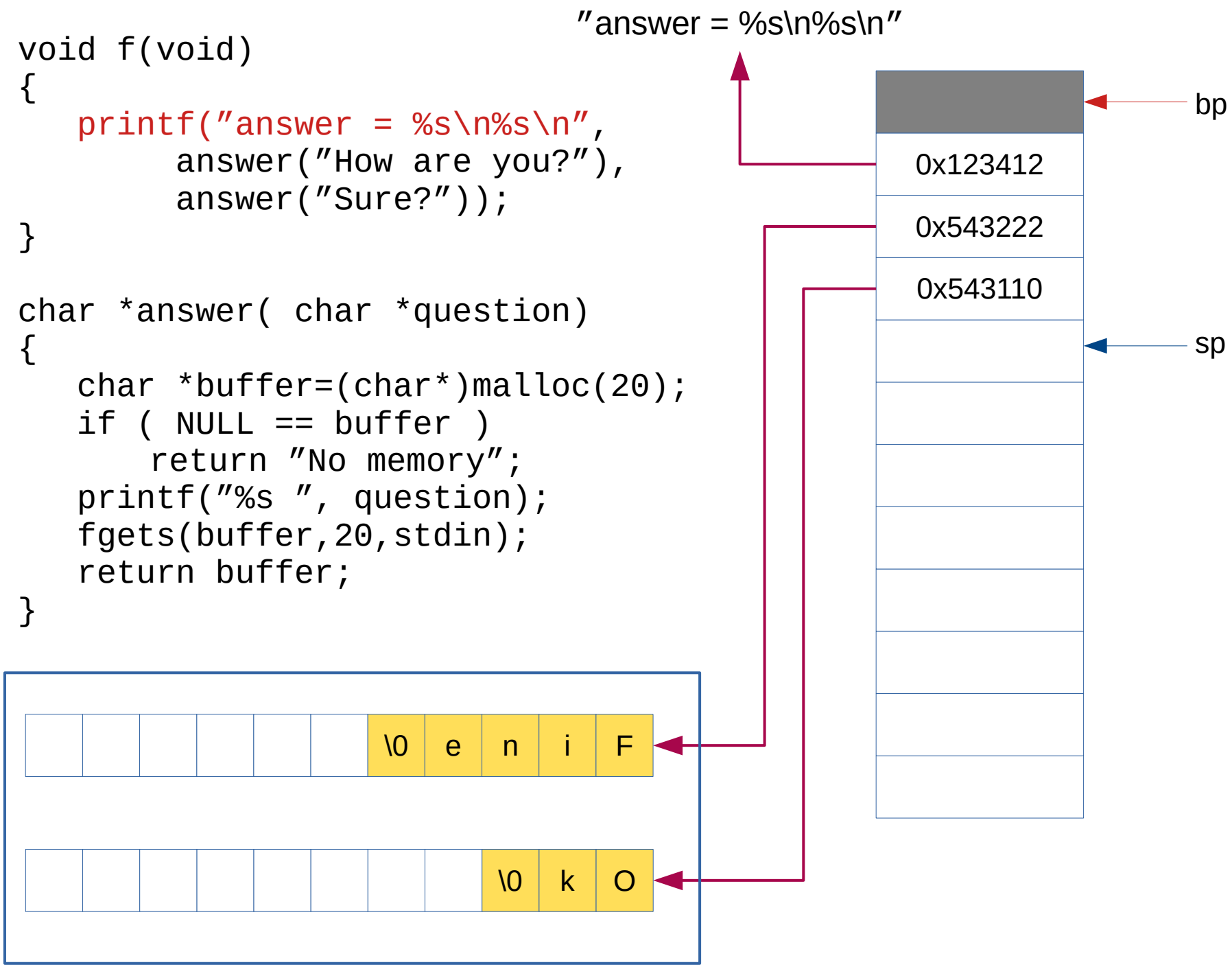
```
void f(void)
{
   printf("answer = %s\n%s\n",
        answer("How are you?"),
        answer("Sure?"));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return "No memory";
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



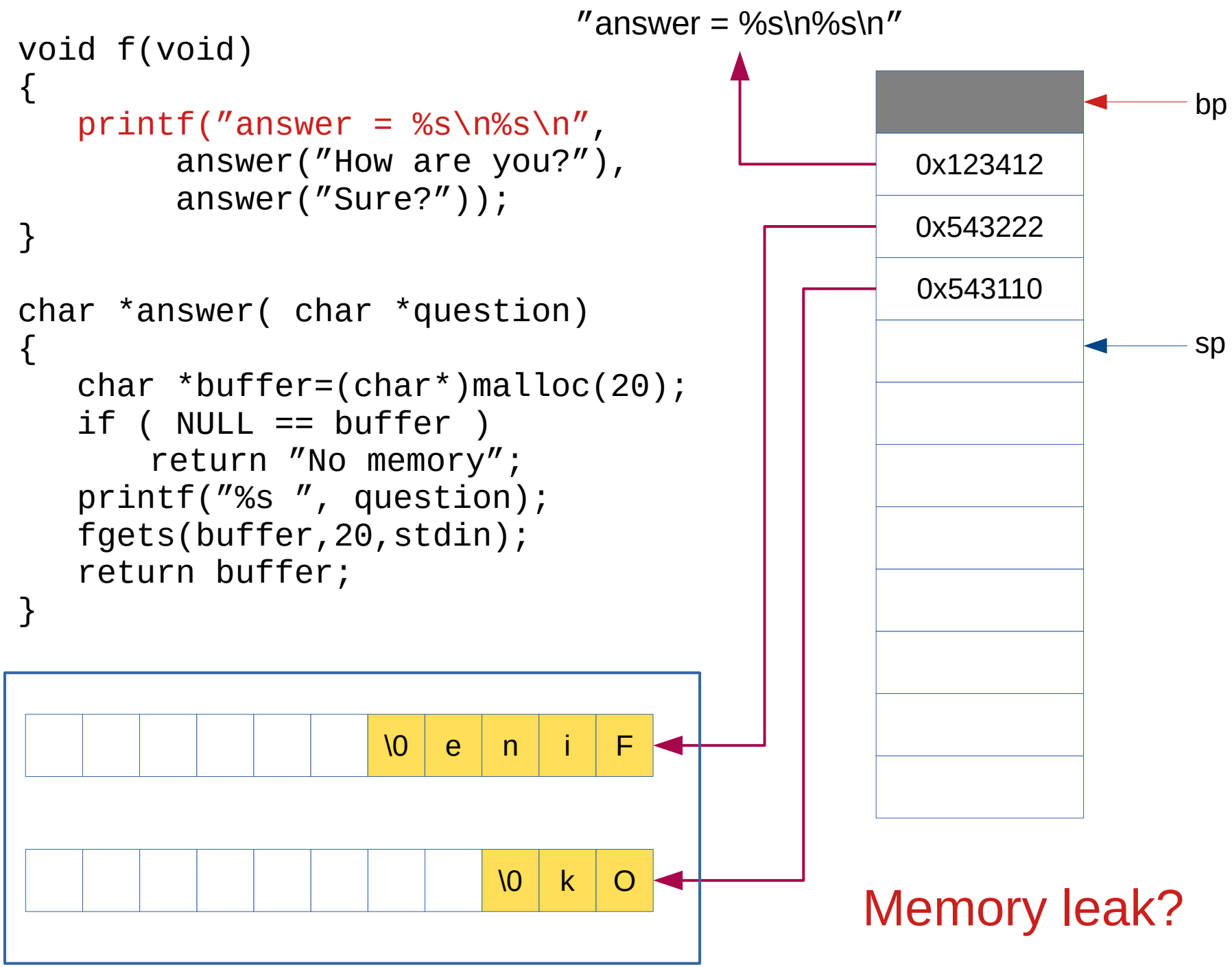
```
void f(void)
{
   printf(”answer = %s\n%s\n”,
        answer(”How are you?”),
        answer(”Sure?”));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return ”No memory”;
   printf(”%s ”, question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



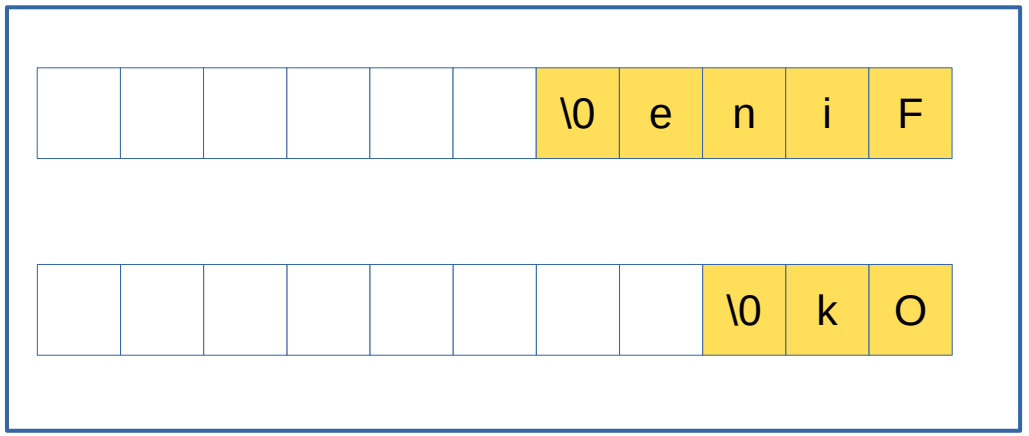
```
void f(void)
{
   printf("answer = %s\n%s\n",
        answer("How are you?"),
        answer("Sure?"));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return "No memory";
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```

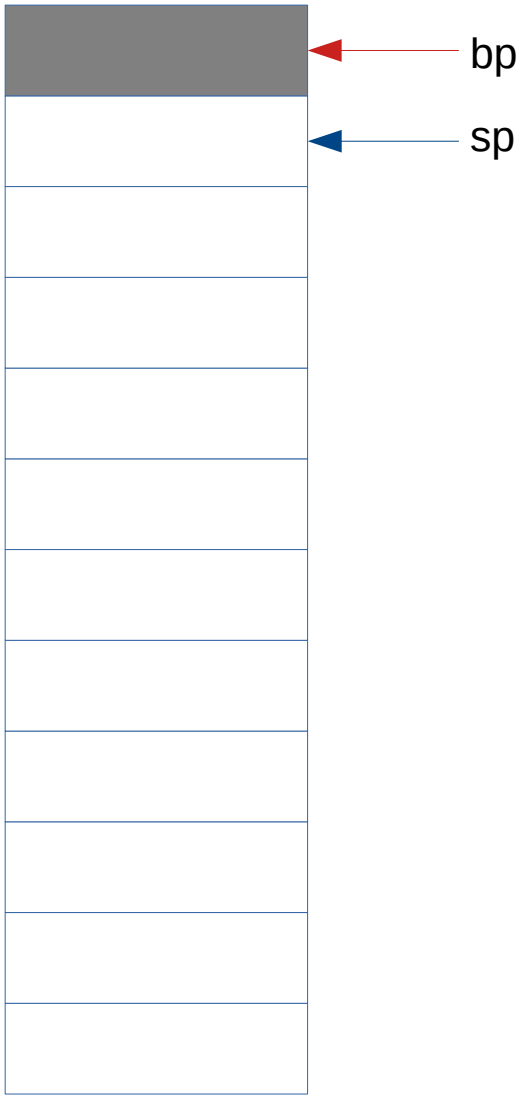


```
void f(void)
{
   printf("answer = %s\n%s\n",
        answer("How are you?"),
        answer("Sure?"));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return "No memory";
   printf("%s ", question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



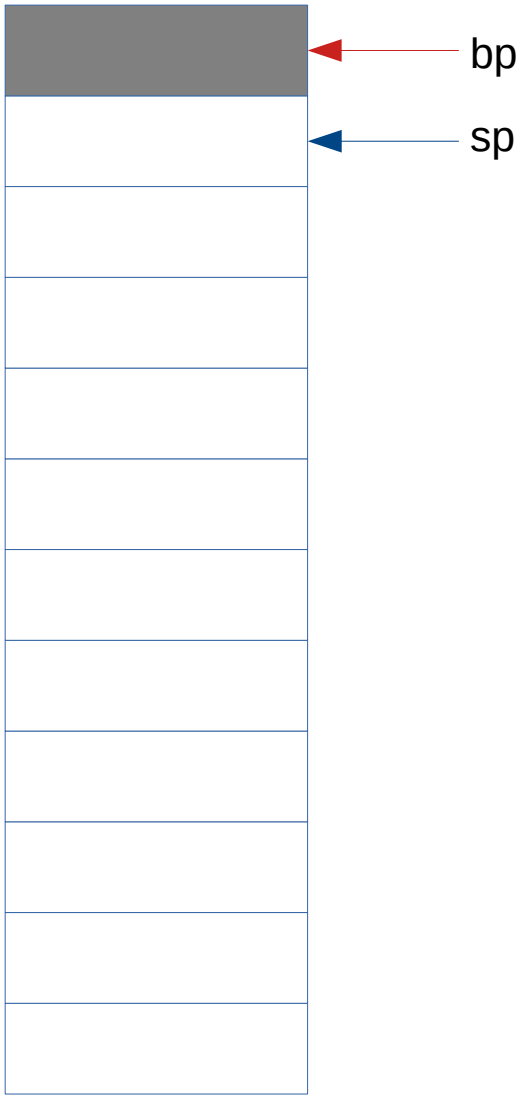
```
void f(void)
{
   printf(”answer = %s\n%s\n”,
        answer(”How are you?”),
        answer(”Sure?”));
}

char *answer( char *question)
{
   char *buffer=(char*)malloc(20);
   if ( NULL == buffer )
       return ”No memory”;
   printf(”%s ”, question);
   fgets(buffer,20,stdin);
   return buffer;
}
```



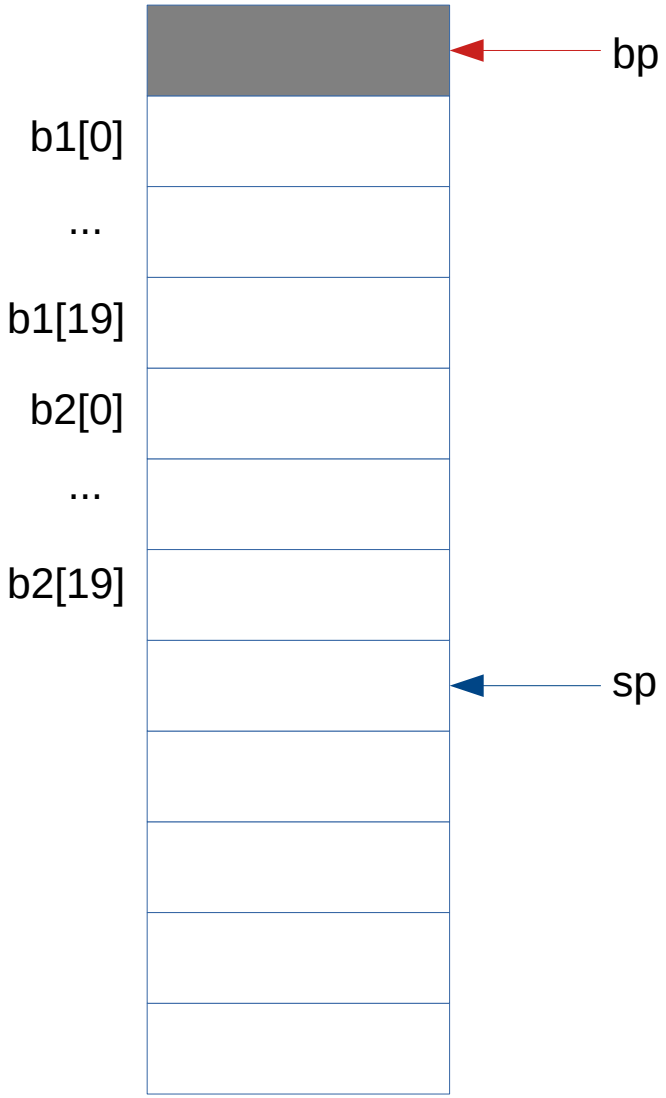


Memory leak?

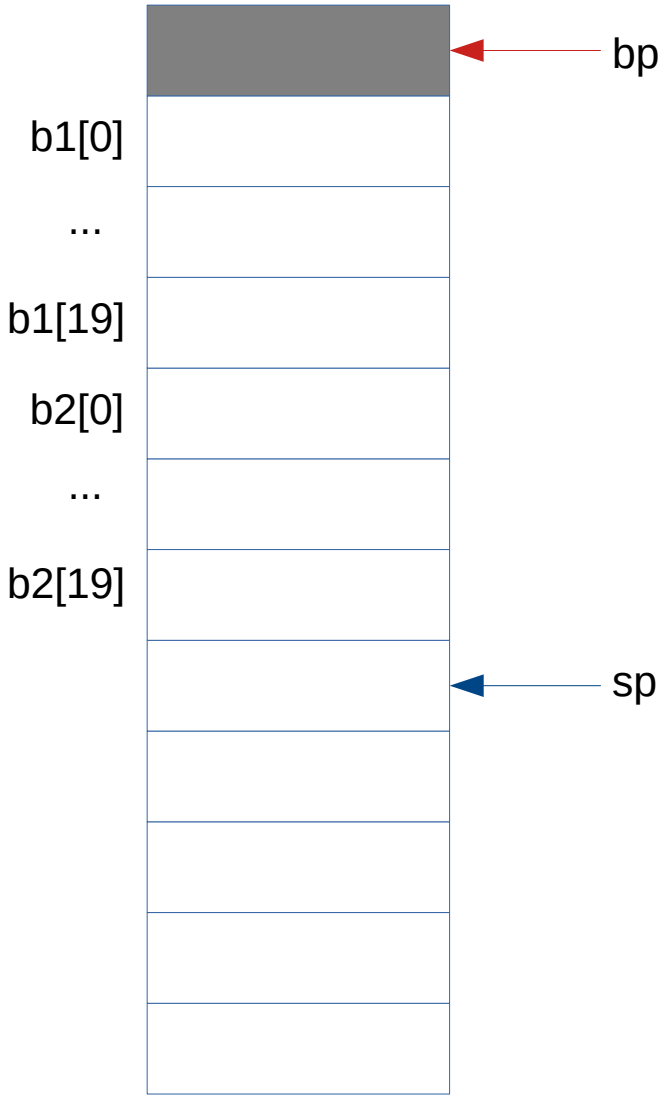
```
void f(void)
{
   char b1[20];
   char b2[20];

    printf(”answer = %s\n%s\n”,
        answer(”How are you?”,b1,20),
        answer(”Sure?”, b2, 20));
}

char *answer( char *question,
              char *buffer, int len)
{
   printf(”%s ”, question);
   fgets(buffer,len,stdin);
   return buffer;
}
```



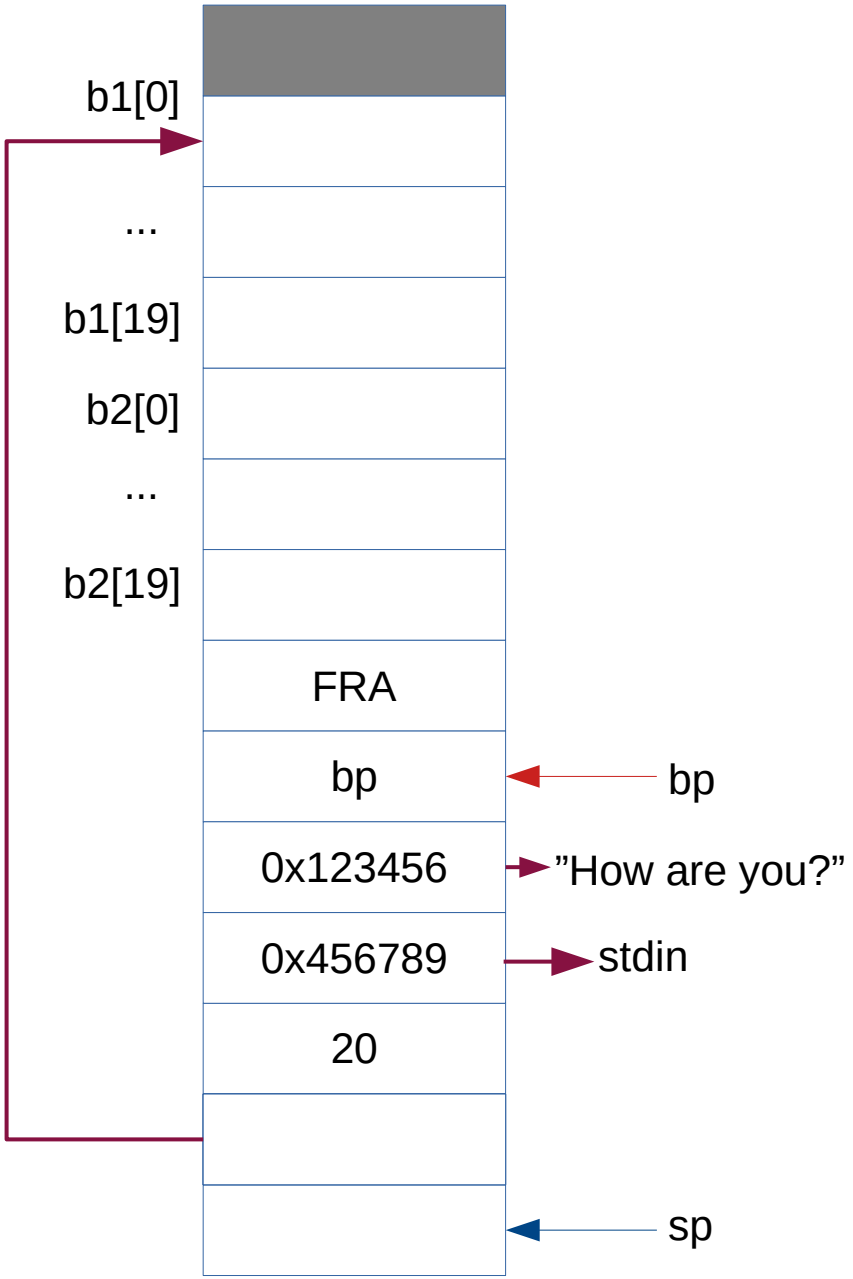
```
void f(void)
{
   char b1[20];
   char b2[20];

    printf(”answer = %s\n%s\n”,
        answer(”How are you?”,b1,20),
        answer(”Sure?”, b2, 20));
}

char *answer( char *question,
              char *buffer, int len)
{
   printf(”%s ”, question);
   fgets(buffer,len,stdin);
   return buffer;
}
```



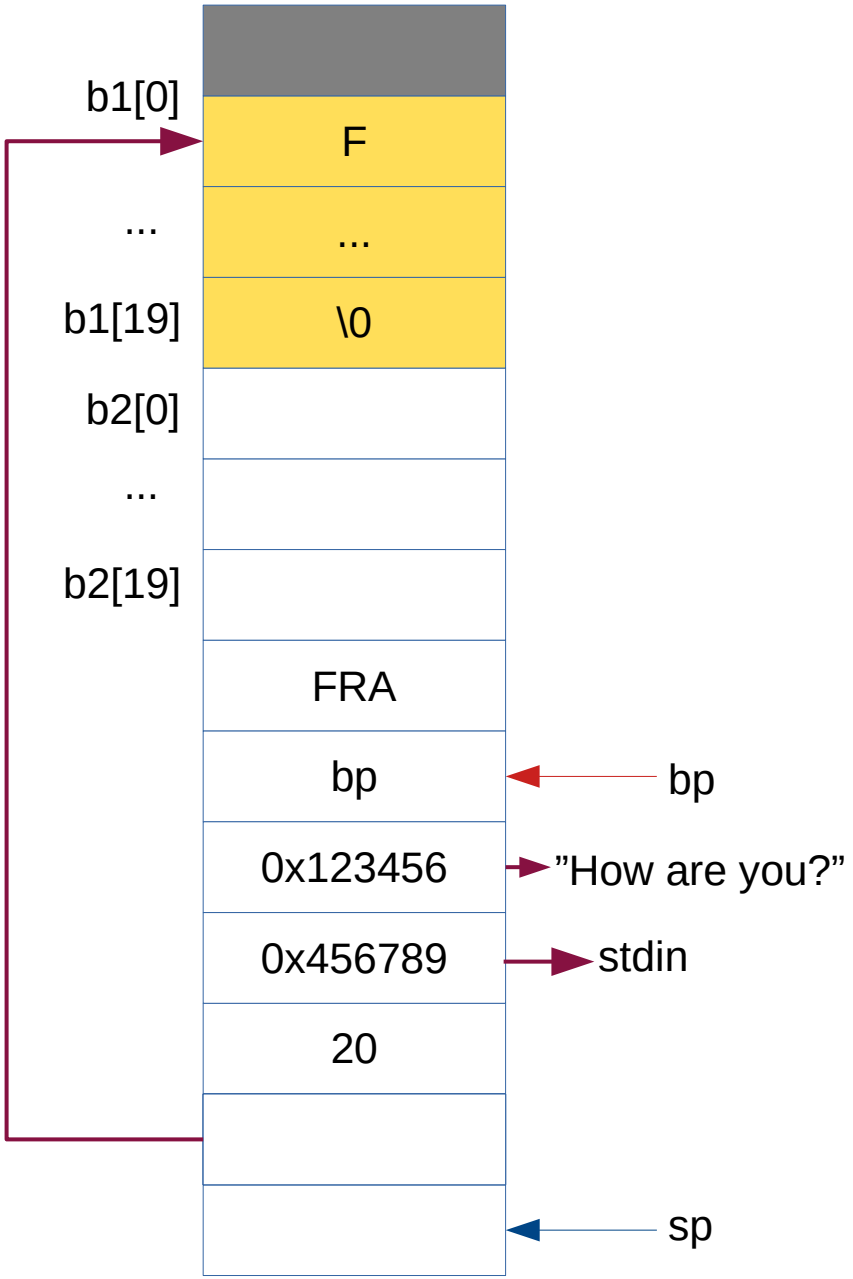
```
void f(void)
{
   char b1[20];
   char b2[20];

    printf(”answer = %s\n%s\n”,
        answer(”How are you?”,b1,20),
        answer(”Sure?”, b2, 20));
}

char *answer( char *question,
              char *buffer, int len)
{
   printf(”%s ”, question);
   fgets(buffer,len,stdin);
   return buffer;
}
```



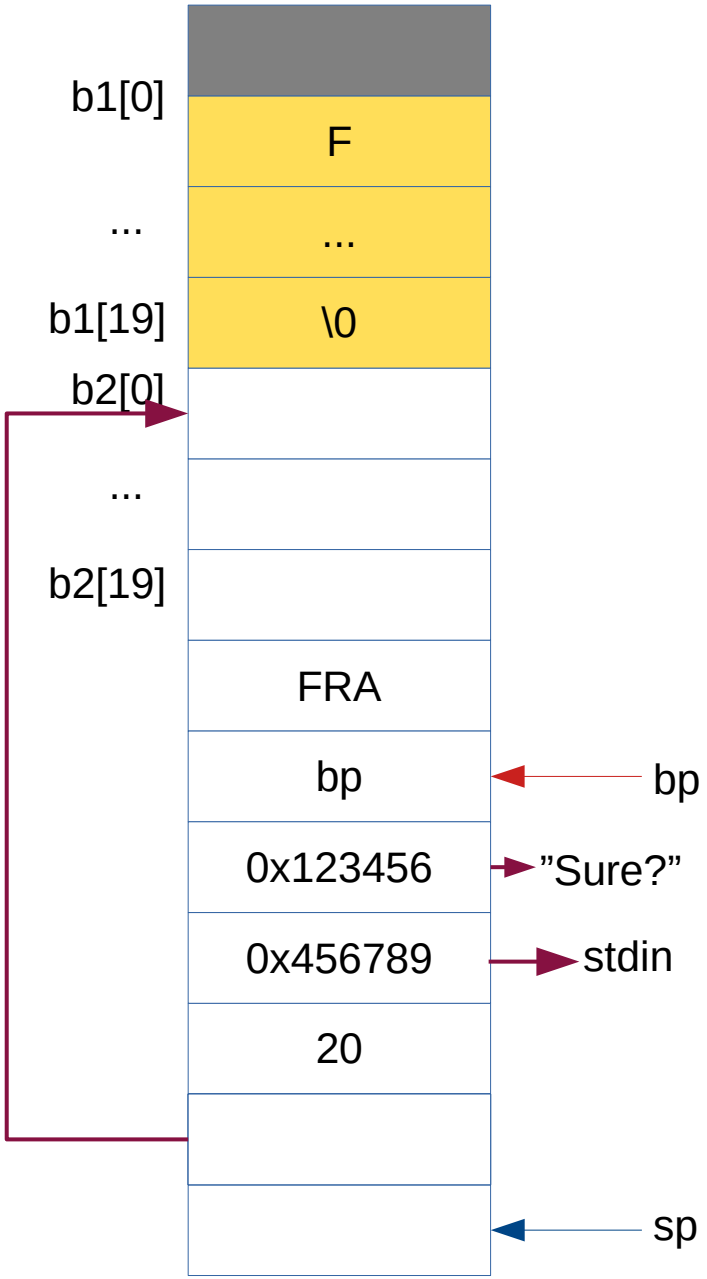
```
void f(void)
{
   char b1[20];
   char b2[20];

    printf("answer = %s\n%s\n",
        answer("How are you?",b1,20),
        answer("Sure?", b2, 20));
}

char *answer( char *question,
              char *buffer, int len)
{
   printf("%s ", question);
   fgets(buffer,len,stdin);
   return buffer;
}
```



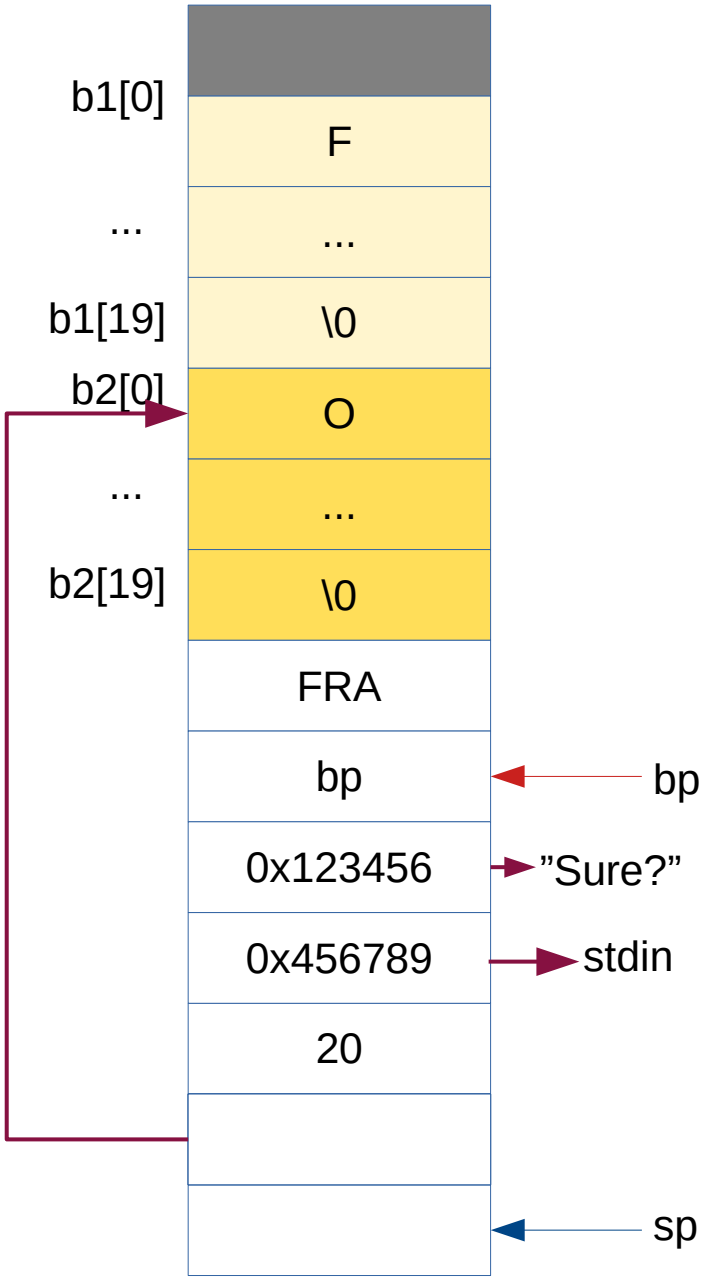
```
void f(void)
{
   char b1[20];
   char b2[20];

    printf("answer = %s\n%s\n",
        answer("How are you?",b1,20),
        answer("Sure?", b2, 20));
}

char *answer( char *question,
              char *buffer, int len)
{
   printf("%s ", question);
   fgets(buffer,len,stdin);
   return buffer;
}
```



```
void f(void)
{
   char b1[20];
   char b2[20];

    printf("answer = %s\n%s\n",
        answer("How are you?",b1,20),
        answer("Sure?", b2, 20));
}

char *answer( char *question,
              char *buffer, int len)
{
   printf("%s ", question);
   fgets(buffer,len,stdin);
   return buffer;
}
```



```
void f(void)
{
   char b1[20];
   char b2[20];

    printf("answer = %s\n%s\n",
        answer("How are you?",b1,20),
        answer("Sure?", b2, 20));
}

char *answer( char *question,
              char *buffer, int len)
{
   printf("%s ", question);
   fgets(buffer,len,stdin);
   return buffer;
}
```



Lifetime traps and pittfals

- Decide who is the **owner** of the memory
- Use the most simple implementation
- Avoid global variables and static lifetime
- Use dynamic memory only when must

```
char *answer( char *question, char *buffer, int len)
{
  printf(”%s ”, question);
  fgets(buffer, len, stdin);
  return buffer;
}
void f(void)
{
  char answ[80];
  printf( "%d\n", answer("How are you? ",answ,80));
}
```

Lifetime traps and pittfals

- Decide who is the **owner** of the memory
- Use the most simple implementation
- Avoid global variables and static lifetime
- Use dynamic memory only when must

```
char *answer( char *question, char *buffer, int len)
{
  printf(”%s ”, question);
  fgets(buffer, len, stdin);
  return buffer;
}
void f(void)
{
  char answ[80];
  printf( "%d\n", answer("How are you? ",answ,sizeof(answ)));
}
```