INTRODUCTION TO POINTERS

Memory representations are not entierly accurate, they are simplified for understanding purposes about working, not architecture

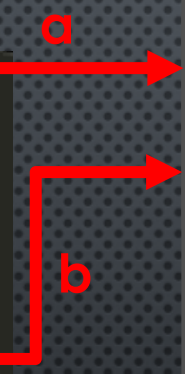
HOW MEMORY WORKS ?

Code :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int main(){
5      int a = 0;
6      int b = 3;
7
8      printf("%x", &b);
9  }
```

Output :

```
$sh  ./a.out
ff00fa00
```



Process Memory

0xff00fa03	Start of stack frame
0xff00fa02	0
0xff00fa00	3
0xff000000	End of stack frame

What is a pointer ?

- A pointer is a variable : has an adress
- A pointer contains the adress of another variable
- The adress contained comes from the context where the pointer is initialized

HOW POINTERS WORKS ?

Code :

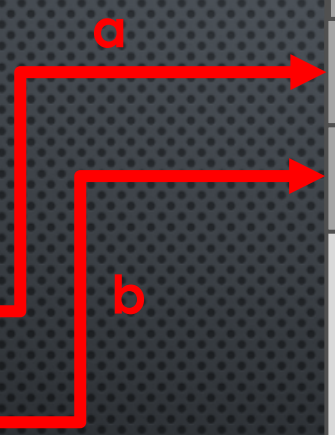
```c
#include <stdlib.h>
#include <stdio.h>

int main(){

    // variable
    int a = 0;

    // pointer to the variable
    int* b = &a;

    printf("%d", *b);
}
```



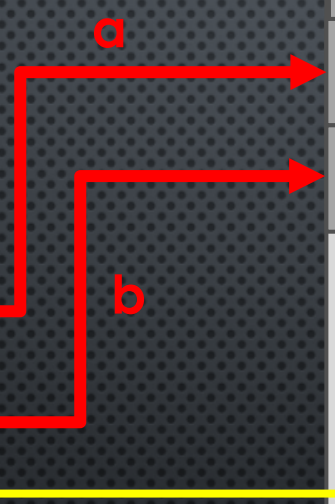
Process Memory

0xff00fa03	Start of stack frame
0xff00fa02	0
0xff00fa00	ff00fa02
0xff000000	End of stack frame

HOW POINTERS WORKS ?

Code :

```
1   #include <stdlib.h>
2   #include <stdio.h>
3
4   int main(){
5
6       // variable
7       int a = 0;
8
9       // pointer to the variable
10      int* b = &a;
11
12      printf("%d", *b);
13  }
```



Process Memory

0xff00fa03	Start of stack frame
0xff00fa02	0
0xff00fa00	ff00fa02
0xff000000	End of stack frame

b

HOW POINTERS WORKS ?

Code :

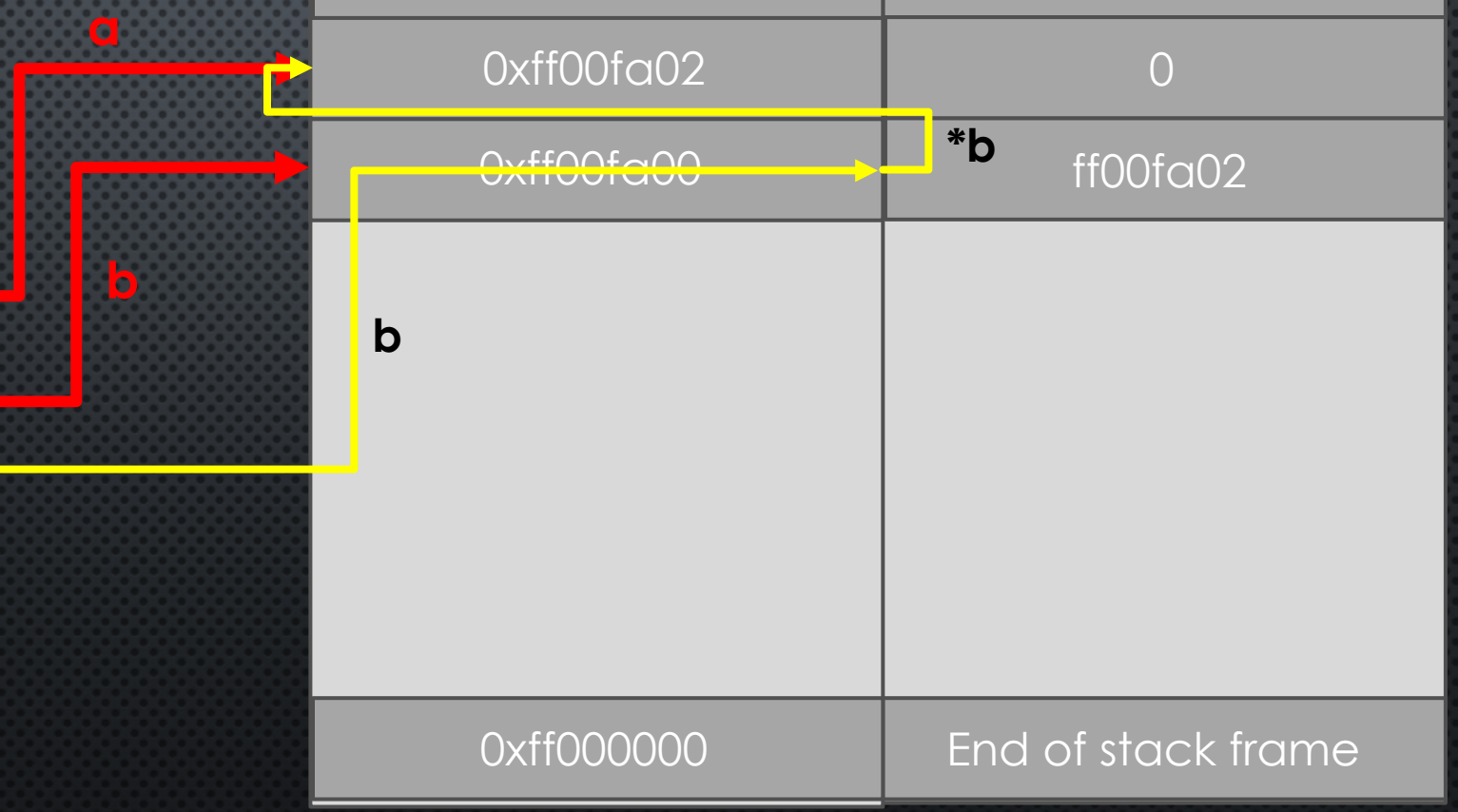
```c
#include <stdlib.h>
#include <stdio.h>

int main(){

    // variable
    int a = 0;

    // pointer to the variable
    int* b = &a;

    printf("%d", *b);
}
```



HOW POINTERS WORKS ?

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int main(){

    // variable
    int a = 0;

    // pointer to the variable
    int* b = &a;

    printf("%d", *b);
}
```

Output :

```
$sh   ./a.out
0
```

What is a pointer ?

- A pointer is a variable : has an adress
- A pointer contains the adress of another variable
- The adress contained comes from the context where the pointer is initialized

What for ?

- When you call a function : new stackframe, cannot access variables from the caller context
- You pass variables as arguments of the function : same problem, because you create a copy of the original variable in the new stackframe
- Solution : pointer, contains adress of the variable in the caller's stackframe

A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int pow(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    int c = pow(a,b);
    printf("%d", c);

}

int pow(int a, int b)
{
    int i;
    int c = 1;
    for(i=0; i<b; i++)
    {
        c*=b
    }
    return c;
}
```

Main Context

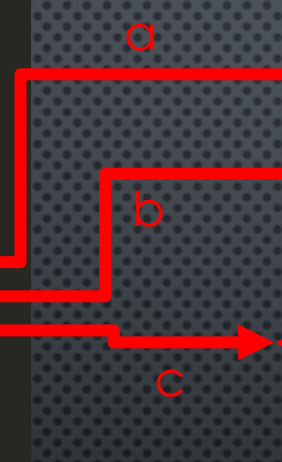
Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	NULL
0xff000016	Adress of pow function
0xff00000e	
0xff000006	
0xff000004	
0xff000002	
0xff000000	

A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```
1   #include <stdlib.h>
2   #include <stdio.h>
3
4   int pow(int a, int b);
5
6   int main()
7   {
8       int a = 4;
9       int b = 5;
10      int c = pow(a,b);
11      printf("%d", c);
12
13  }
14
15  int pow(int a, int b)
16  {
17      int i;
18      int c = 1;
19      for(i=0; i<b; i++)
20      {
21          c*=b
22      }
23      return c;
24  }
```



Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	NULL
0xff000016	Adress of pow function
0xff00000e	
0xff000006	
0xff000004	
0xff000002	
0xff000000	

A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int pow(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    int c = pow(a,b);
    printf("%d", c);

}

int pow(int a, int b)
{
    int i;
    int c = 1;
    for(i=0; i<b; i++)
    {
        c*=b
    }
    return c;
}
```

« Caller » Context

Pow Context

Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	NULL
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call Adress to return when pow end
0xff000006	
0xff000004	
0xff000002	
0xff000000	

A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int pow(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    int c = pow(a,b);
    printf("%d", c);

}

int pow(int a, int b)
{
    int i;
    int c = 1;
    for(i=0; i<b; i++)
    {
        c*=b
    }
    return c;
}
```

Copy of a

Copy of b

Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	NULL
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call
0xff000006	4
0xff000004	5
0xff000002	0
0xff000000	End of stack 1024

A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int pow(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    int c = pow(a,b);
    printf("%d", c);

}

int pow(int a, int b)
{
    int i;
    int c = 1;
    for(i=0; i<b; i++)
    {
        c*=b
    }
    return c;
}
```

Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	1024
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call
0xff000006	4
0xff000004	5
0xff000002	0
0xff000000	End of stack 1024

A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int pow(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    int c = pow(a,b);
    printf("%d", c);

}

int pow(int a, int b)
{
    int i;
    int c = 1;
    for(i=0; i<b; i++)
    {
        c*=b
    }
    return c;
}
```

Output :

```
$sh   ./a.out
1024
```

Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	1024
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call
0xff000006	4
0xff000004	5
0xff000002	0
0xff000000	End of stack 1024

BUT WHAT ABOUT WITH POINTERS ?

WITH POINTERS :

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int pow(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    int* c = &a;
    pow(c,b);
    printf("%d", a);
}

void pow(int* a, int b)
{
    int i;
    int c = *a;
    for(i=0; i<b; i++)
    {
        (*a)*=c
    }
}
```

Main Context

Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	0xff00001c
0xff000016	Adress of pow function
0xff00000e	
0xff000006	
0xff000004	
0xff000002	
0xff000000	

BEFORE THE FOR LOOP

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int pow(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    int* c = &a;
    pow(c,b);
    printf("%d", a);
}

void pow(int* a, int b)
{
    int i;
    int c = *a;
    for(i=0; i<b; i++)
    {
        (*a)*=c
    }
}
```

« Caller » Context

Pow Context

Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	0xff00001c
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call
[illegible]	0xff00001c
0xff000004	5
0xff000002	0
0xff000000	End of stack 4

BEFORE THE FOR LOOP

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int pow(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    int* c = &a;
    pow(c,b);
    printf("%d", a);
}

void pow(int* a, int b)
{
    int i;
    int c = *a;
    for(i=0; i<b; i++)
    {
        (*a)*=c
    }
}
```

« Caller » Context

Pow Context

Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	0xff00001c
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call
0xff000006	0xff00001c
0xff000004	5
0xff000002	0
0xff000000	End of stack 4

AFTER THE FOR LOOP

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int pow(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    int* c = &a;
    pow(c,b);
    printf("%d", a);
}

void pow(int* a, int b)
{
    int i;
    int c = *a;
    for(i=0; i<b; i++)
    {
        (*a)*=c
    }
}
```

« Caller » Context

Pow Context

Process Memory

0xff00001c	Main stack frame 1024
0xff00001a	5
0xff000018	0xff00001c
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call
0xff000006	0xff00001c
0xff000004	5
0xff000002	0
0xff000000	End of stack 4

AFTER THE FOR LOOP

Code :

```c
#include <stdlib.h>
#include <stdio.h>

int pow(int a, int b);

int main()
{
    int a = 4;
    int b = 5;
    int* c = &a;
    pow(c,b);
    printf("%d", a);
}

void pow(int* a, int b)
{
    int i;
    int c = *a;
    for(i=0; i<b; i++)
    {
        (*a)*=c
    }
}
```

Output :

```
$sh   ./a.out
1024
```

Process Memory

0xff00001c	Main stack frame 1024
0xff00001a	5
0xff000018	0xff00001c
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call
0xff000006	0xff00001c
0xff000004	5
0xff000002	0
0xff000000	End of stack 4