Exercise Session 3

Systems Programming and Computer Architecture

Fall Semester 2024

Disclaimer

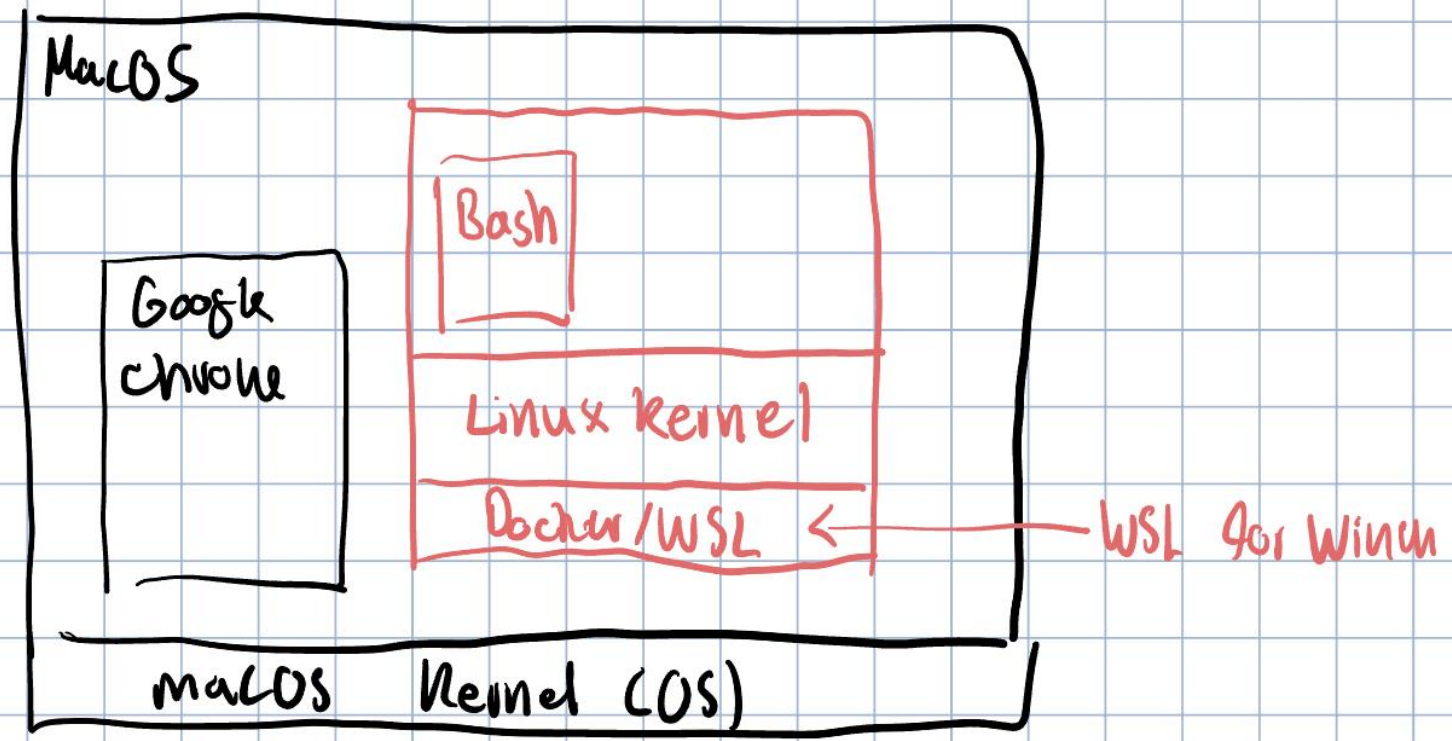
Systems@ETH Zürich

- **Website**: n.ethz.ch/~falkbe/
- **(Extra) Demos on GitHub**: github.com/falkbe
- My exercise slides have **additional slides** (which are not official part of the course) **having a blue heading**: they are there to complement and go into more depth where I found appropriate
- For the exam **only** the official exercise slides are relevant, if in doubt always check the ones on the official moodle page

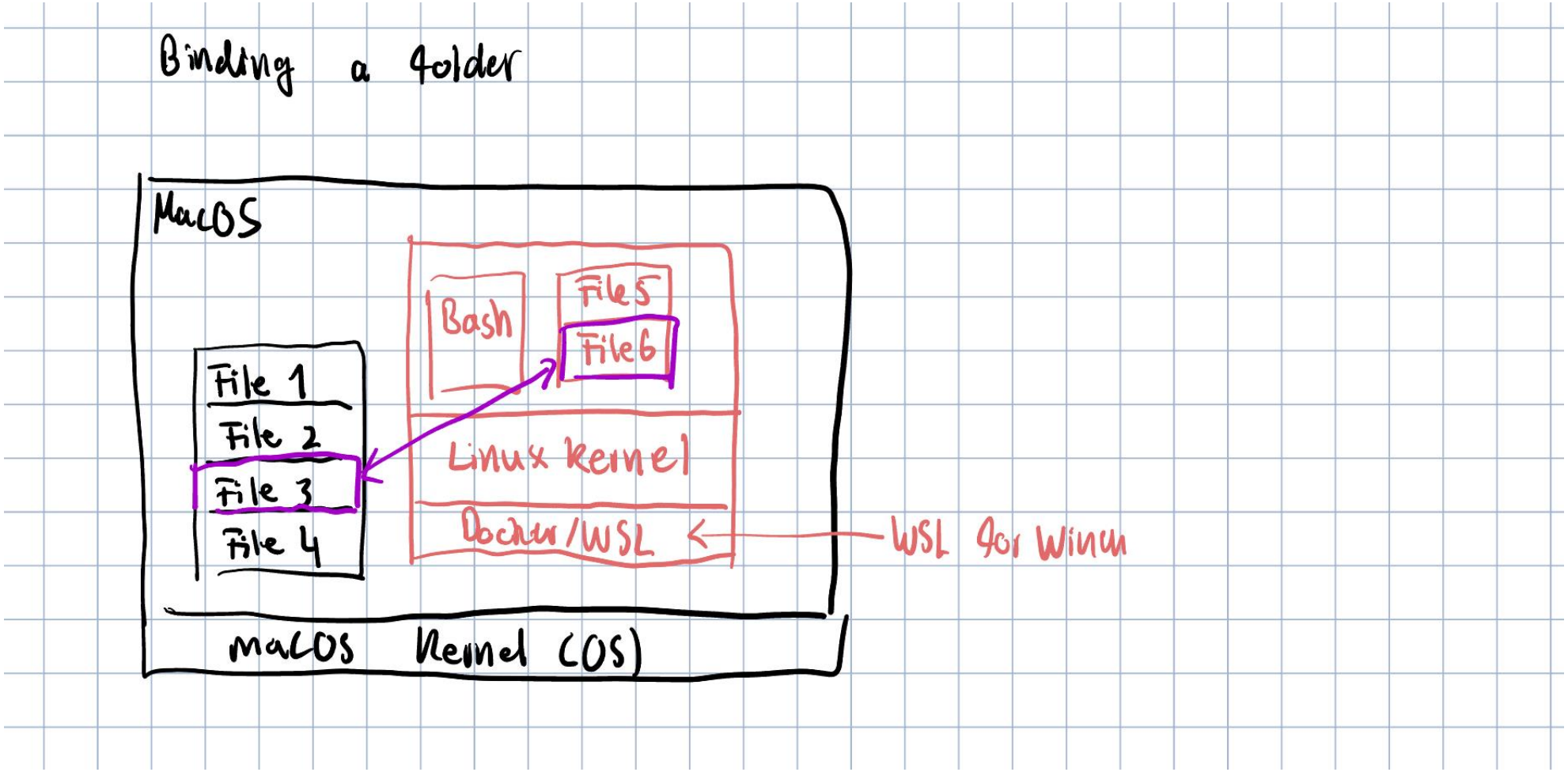
Question

- Who actually knows basics of the terminal (i.e. can change into directories, create files, edit files)?
- Who knows how to **compile a C file** (i.e. used gcc)?

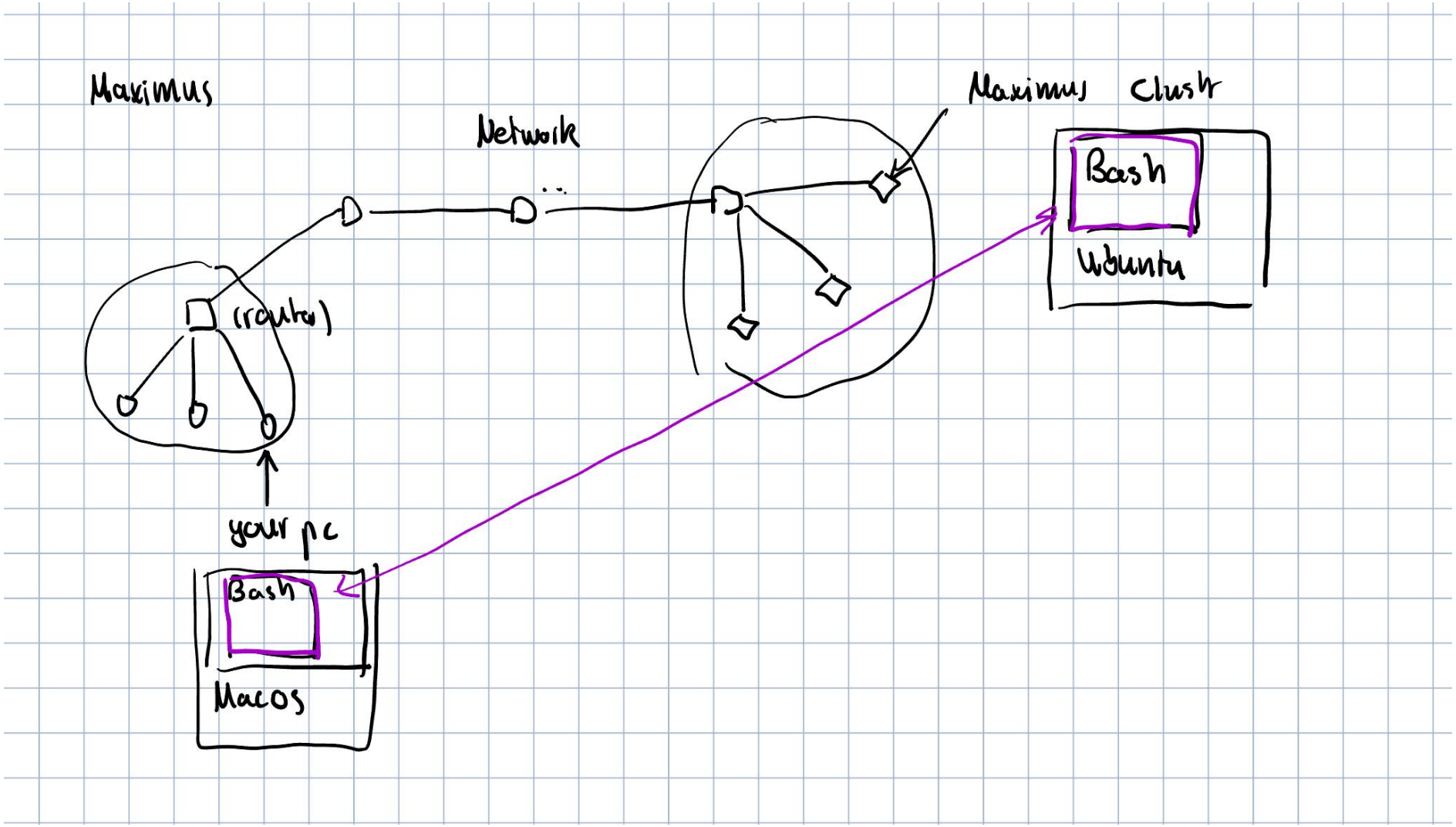
Docker, WSL, Maximus, what are they?



Docker, WSL, Maximus, what are they?



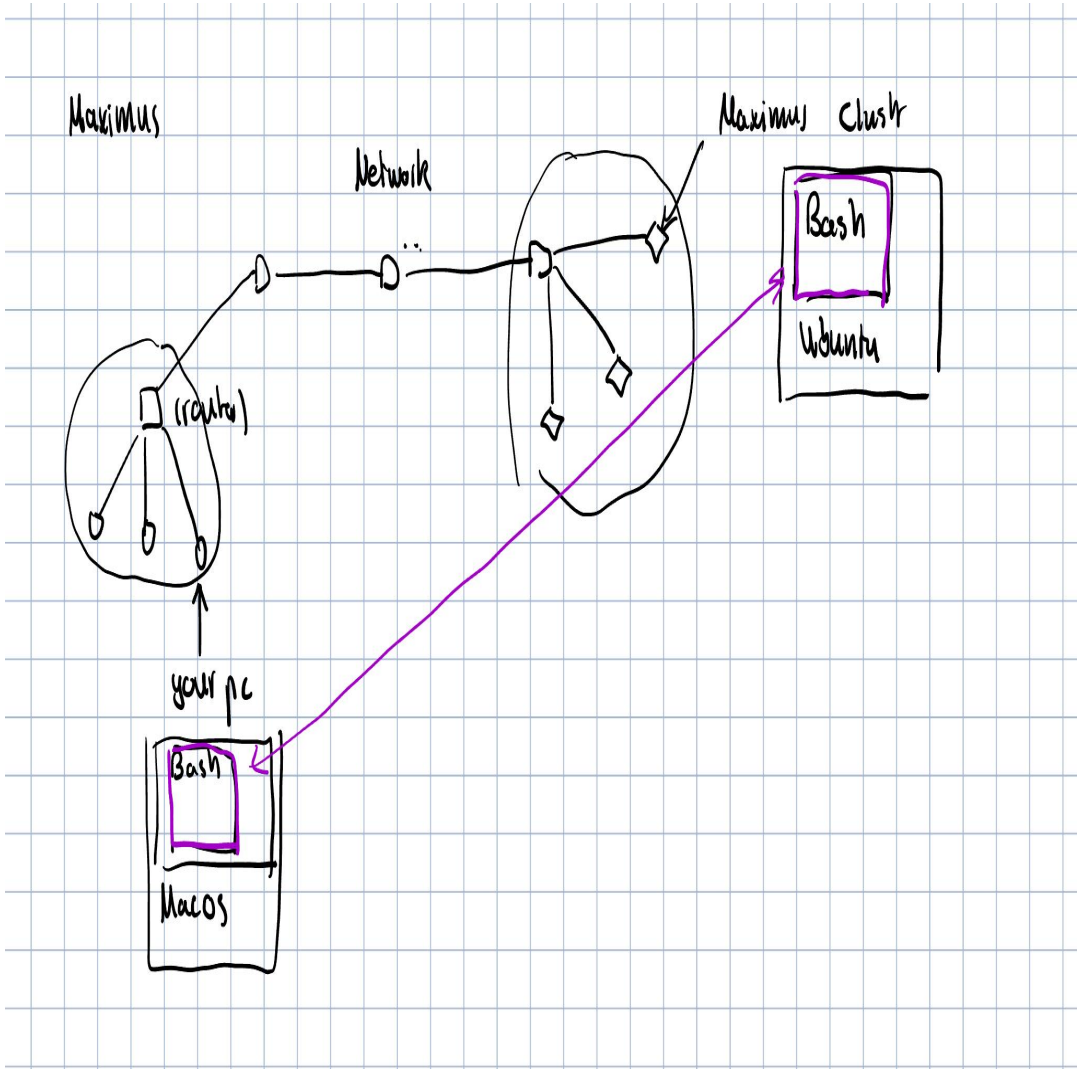
Docker, WSL, Maximus, what are they?



Docker, WSL, Maximus, what are they?

Systems@ETH Zürich

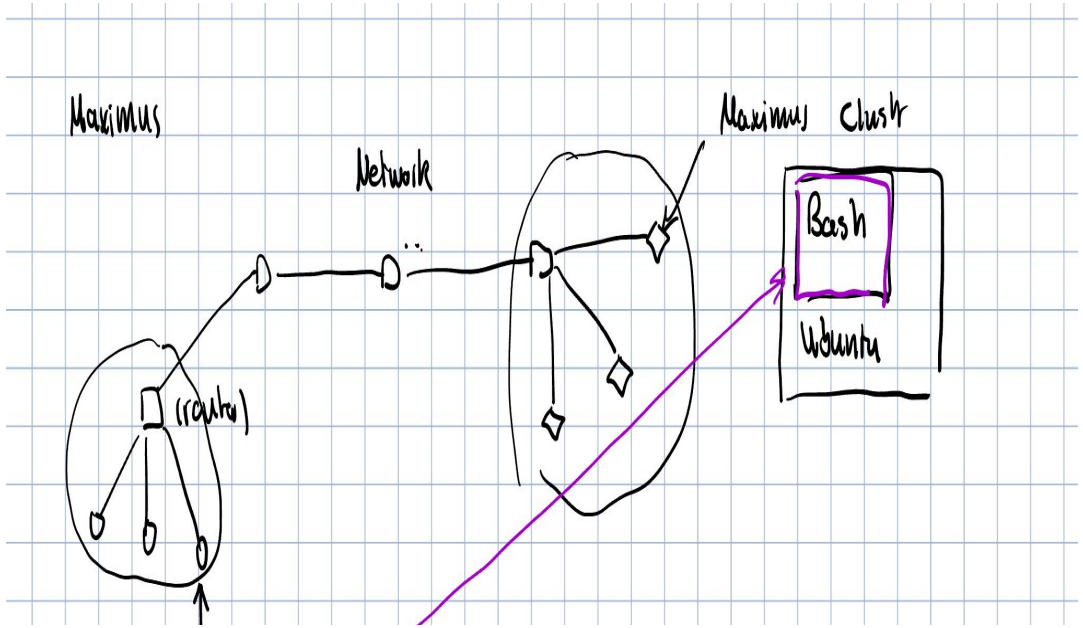
- Where is maximus? **You learn that in Computer Networks**



Docker, WSL, Maximus, what are they?

Systems@ETH Zürich

- Where is maximus? **You learn that in Computer Networks**




```
traceroute to maximus.inf.ethz.ch (129.132.216.39), 64 hops max, 48 byte packets
 1  rou-hg-1-student-net-hg-dock-1-a.ethz.ch (10.5.80.1)  4.651 ms  3.310 ms  2.653 ms
 2  rou-bgw-lee-student-net (10.1.5.21)  4.921 ms  3.192 ms  4.327 ms
 3  rou-fw-lee-student-net (10.1.5.17)  4.106 ms  3.761 ms  3.936 ms
 4  rou-fw-hci-rz-fw.ethz.ch (192.33.92.186)  3.910 ms  4.633 ms  3.122 ms
 5  rou-bgw-hci-service-inf-isg (10.1.18.38)  5.771 ms  4.675 ms  3.347 ms
 6  0.0.0.0 (0.0.0.0)  4.259 ms  3.933 ms  3.945 ms
 7  rou-dcz1-dg-service-inf-isg-dcz1-server-2-a.ethz.ch (129.132.14.193)  5.296 ms  5.495 ms  4.414 ms
 8  maximus.inf.ethz.ch (129.132.216.39)  4.379 ms  3.963 ms  7.915 ms
```

Why would you care?

- Different executables on different machines: s.t. you can recreate our correct code (or our errors) you need to compile on the **system**
- **Segfault demo**

Where are we in the course?

Systems@ETH Zürich

- **1. Programming Language C**
  - Looked at basic syntax of C
  - Yesterday and today: pointers, manual memory allocation : **dynamic memory allocation** (heart of C)
  - Future: how to **implement** dynamic memory allocation (lists)
- **2. Assembly, Variadic Functions, Linking and Loading, Compiler**
- **3. Computer Architecture**: DDCA and PPROG, Devices

Agenda

- Last week's assignment
- More on C-Programing (Assignment 2)
  - File I/O
- Lecture Recap: Pointers
- Quiz: Pointers

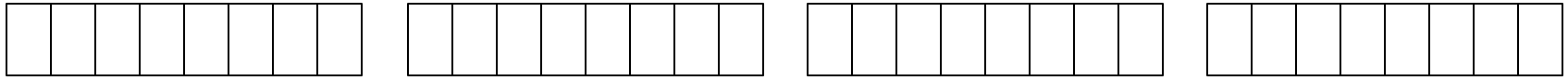
Last Week's Assignment

Bitwise Operations

Slides: BitOps

Bit Lab – bitCount(x)

- Naïve Approach:
  - `(x & 1) + ((x >> 1) & 1) + … + ((x >> 31) & 1)`
  - requires: 31 +, 32 &, 31 >> →94 operators!!
- Another Idea:
  - Divide 32 bits into segements of «bit counters»



accumumulate bit 25 to 32

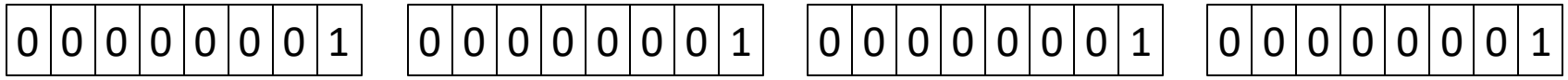
accumumulate bit 17 to 24

accumumulate bit 9 to 16

accumumulate bit 1 to 8

Bit Lab - bitCount(x) with counters

- Mask: `unsigned int m = 1 + (1 << 8) + (1 << 16) + (1 << 24)`



- Accumulate:

```
unsigned int counts = (x & m) + ((x >> 1) & m) + … ((x >> 7) & m)
```

- Sum up:

```
unsigned int sum = (counts & 0xFF) + ((counts >> 8) & 0xFF) +
       ((counts >> 16) & 0xFF) + ((counts >> 24) & 0xFF)
```

6

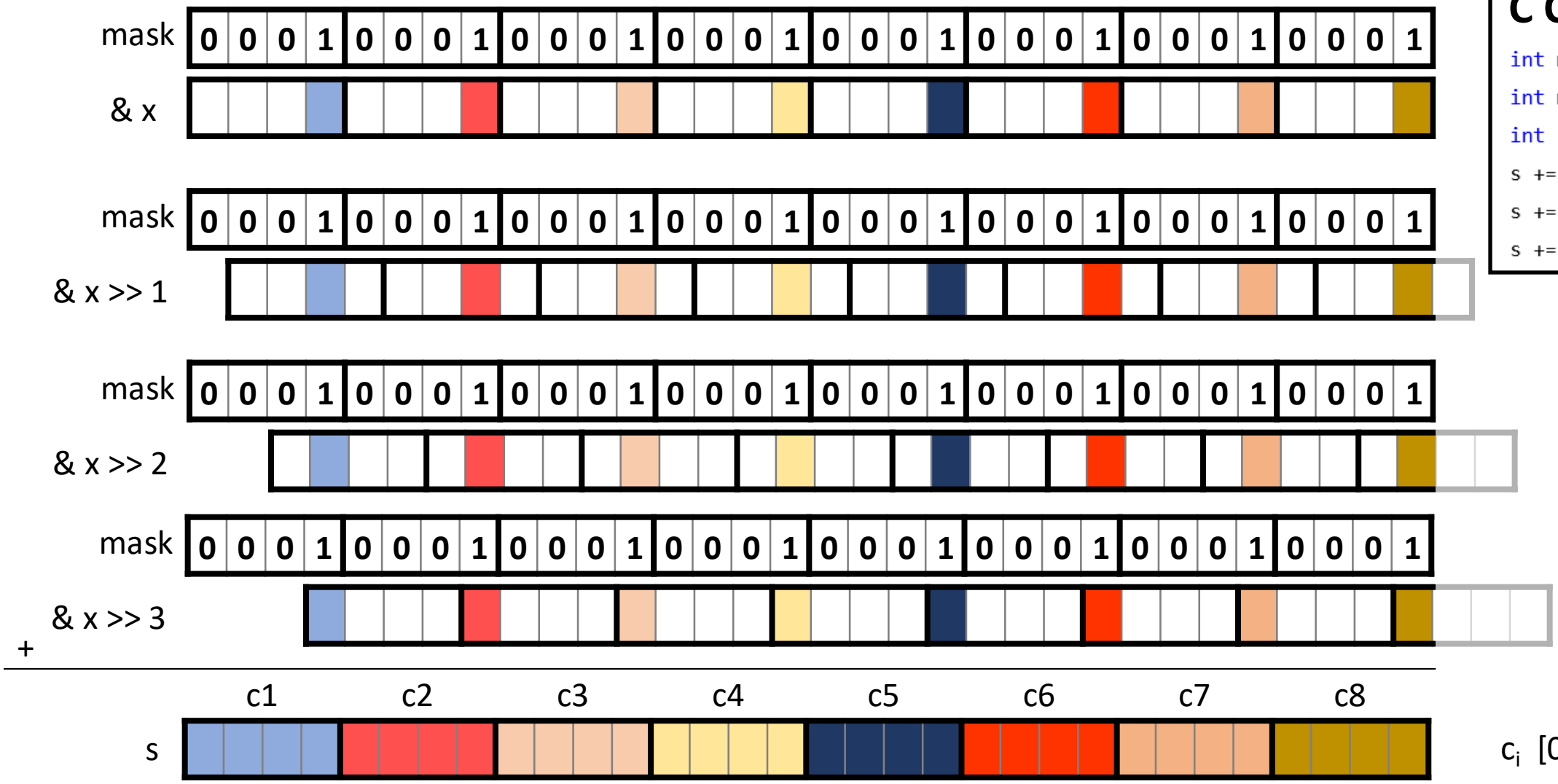
22

10

38

Bit Lab - bitCount(x) – better solution?

Accumulate in eight 4-bit Counters (rather than 4 8-bit Counters)



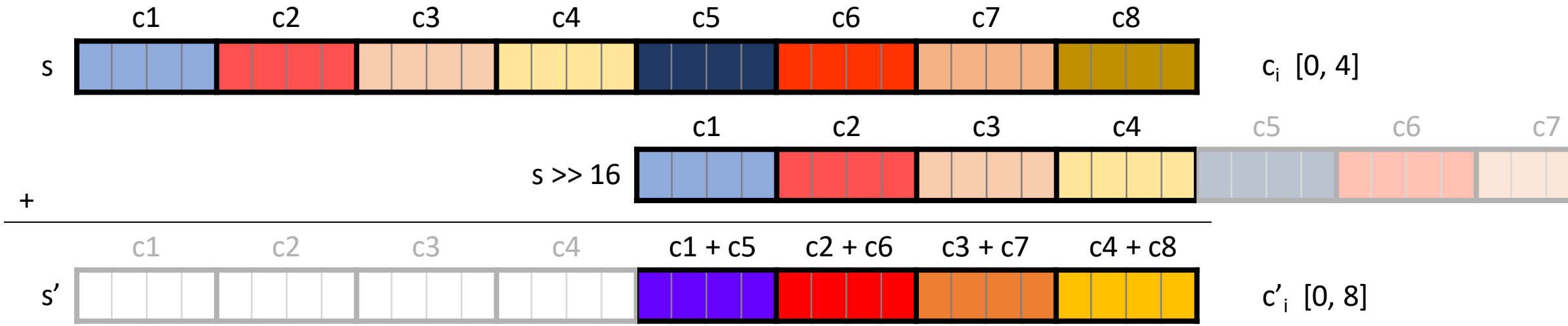
C Code

```
int m1 = 0x11 | (0x11 << 8);
int mask = m1 | (m1 << 16);
int s = x & mask;
s += x>>1 & mask;
s += x>>2 & mask;
s += x>>3 & mask;
```

c_i [0, 4]

Step 1: Reduce from 8 to 4 Counters

Combine 4 upper counters with 4 lower counters



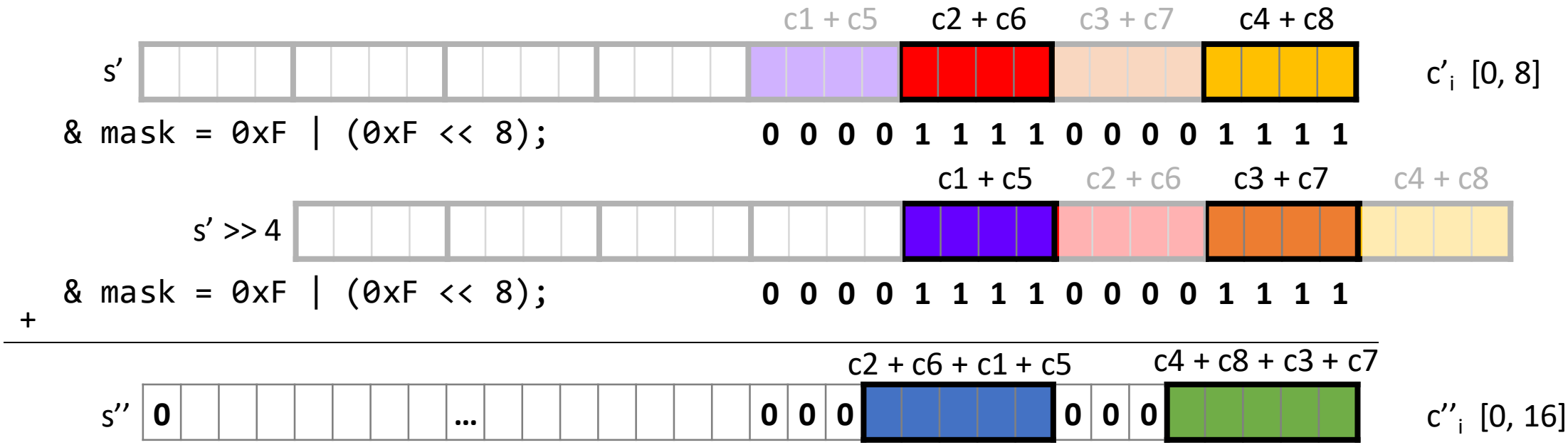
Goal 1 Counter: c1 + c2 + c3 + c4 + c5 + c6

C Code

```
s = s + (s >> 16);
```

Step 2: Reduce from 4 to 2 Counters

Combine 1st with 2nd and 3rd with 4th



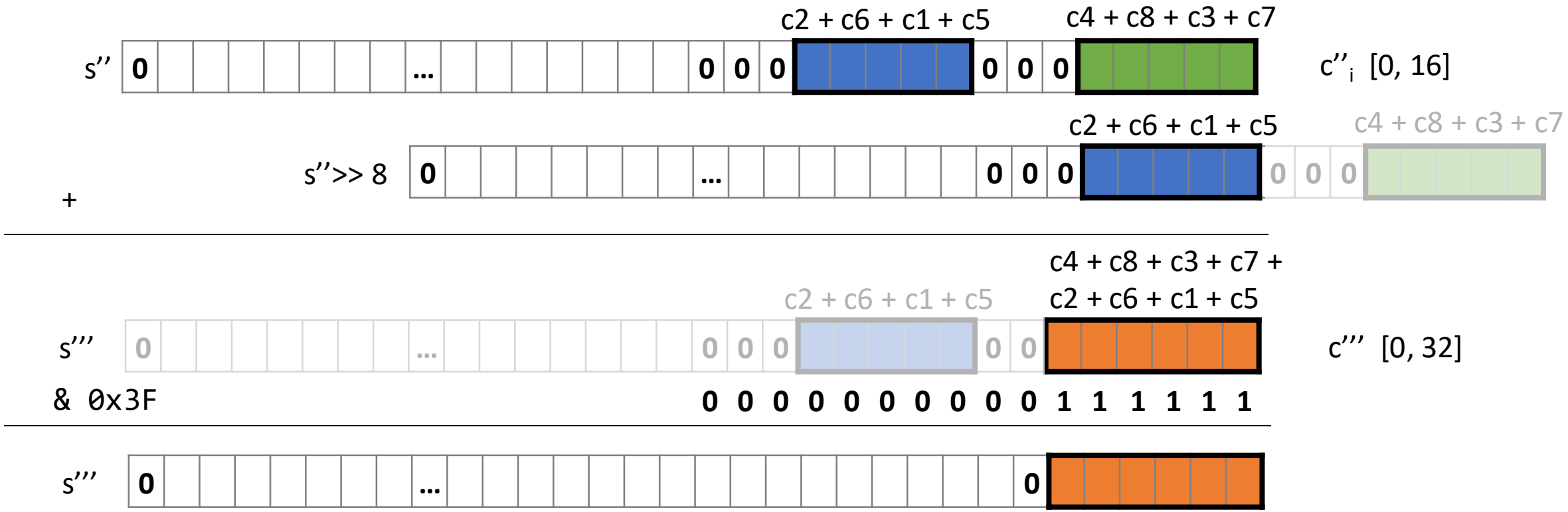
Goal 1 Counter: c1 + c2 + c3 + c4 + c5 + c6

C Code

```
mask = 0xF | (0xF << 8);
s = (s & mask) + ((s >> 4) & mask);
```

Step 3: Reduce from 2 to 1 Counter

Combine the two remaining counters



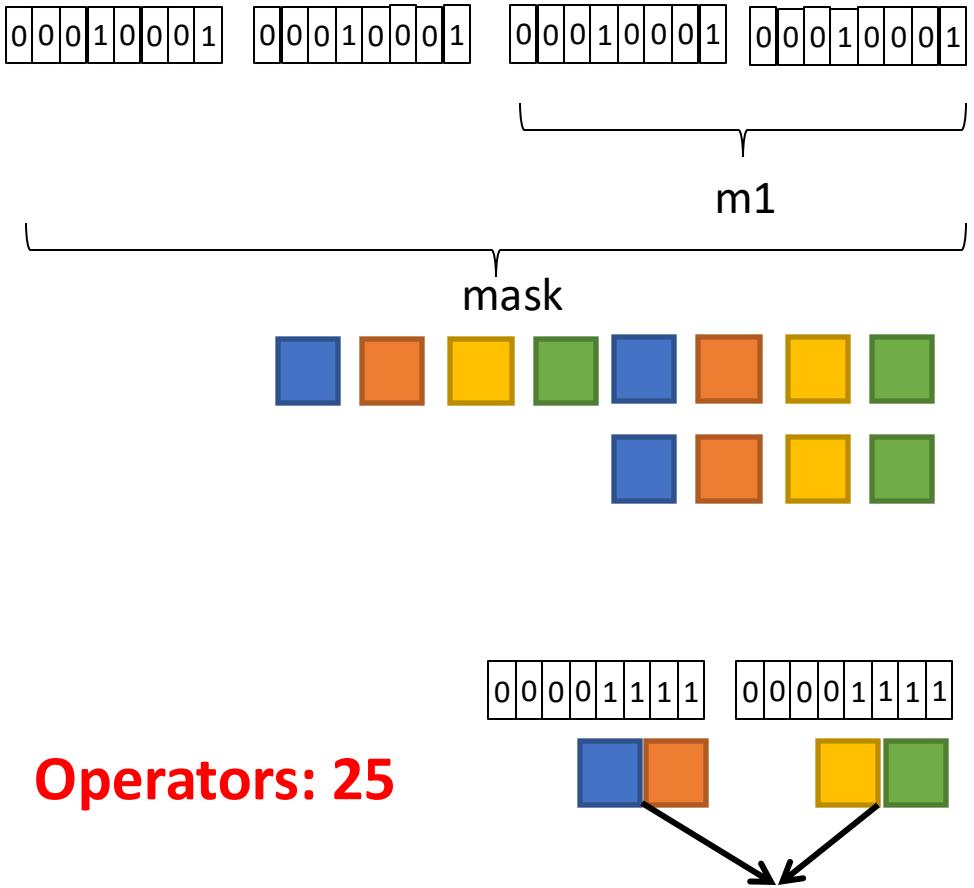
Goal 1 Counter: c1 + c2 + c3 + c4 + c5 + c6

C Code

```
return (s + (s>>8)) & 0x3F;
```

Bit Lab - bitCount(x) – better solution?

```
int bitCount(int x) {
    /* Sum 8 groups of 4 bits each */
    int m1 = 0x11 | (0x11 << 8);
    int mask = m1 | (m1 << 16);
    int s = x & mask;
    s += x>>1 & mask;
    s += x>>2 & mask;
    s += x>>3 & mask;
    /* Now combine high and low order sums */
    s = s + (s >> 16);
    /* Low order 16 bits now consists of 4 sums,
       each ranging between 0 and 8.
       Split into two groups and sum */
    mask = 0xF | (0xF << 8);
    s = (s & mask) + ((s >> 4) & mask);
    return (s + (s>>8)) & 0x3F;
}
```



Bit Lab

Systems@ETH Zürich

Questions for Assignment 1?

Remark Assignment 1

Systems@ETH Zürich

- Need to understand the **basics** of bit manipulation (shifting, or, and, xor, signed, unsigned numbers etc.)
- **Exam** will most likely **not** ask the tricky questions as in assignment 1 (implement with 2 ops, or divide and conquer adding)

Preview: Assignment 2

More on C-Programming

and short overview of

assignment 02

Integer Data Types

- You may use the types defined in C99's `stdint.h`

```c
#include <stdint.h>

void main(int argc, char *argv[]) {
  uint8_t x = 22;
  int16_t y = 4434;
  int64_t z = 1 << 44;
}
```

- Keep in mind not to overflow by using a too small representation for the value

http://www.cplusplus.com/reference/cstdint/
http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/stdint.h.html

Recap: C99 stdint.h

Systems@ETH Zürich

C99 extended integer types

```
#include <stdint.h>

int8_t          a;
int16_t         b;
int32_t         c;
int64_t         d;

uint8_t         x;
uint16_t        y;
uint32_t        z;
uint64_t        w;
```

Signed integers, precise size in bits

Unsigned integers,

1. Reverse an array

Write a C program that has a function that:

- accepts an array of unsigned integers and a length
- reverses the elements of the array in place
- returns void (nothing)

```
…
void reverse(unsigned int *arr, size_t len) {
        …
}
…
```

Choose Your Types Wisely

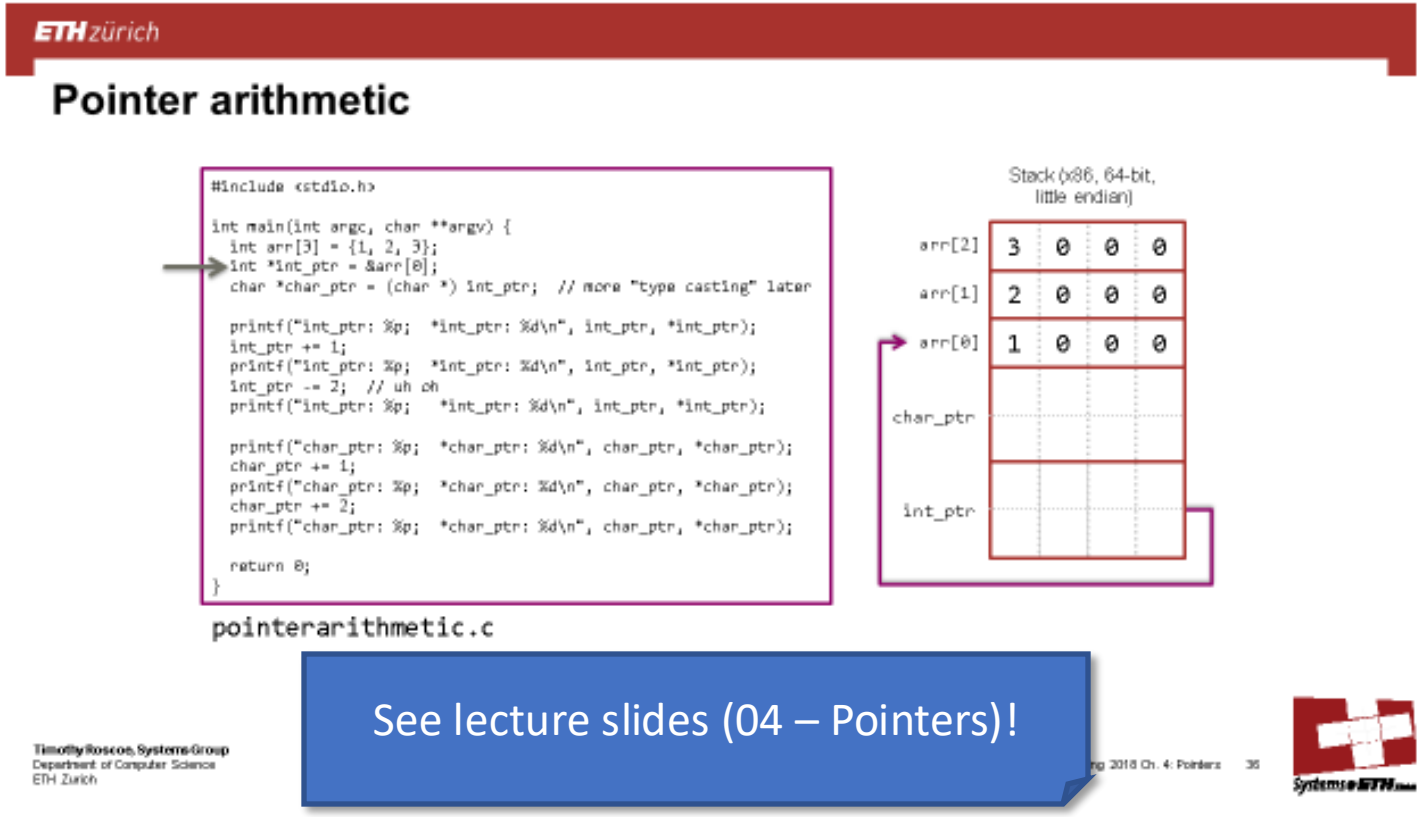
```
…
void reverse_array(unsigned int* array, size_t length)
{
  for (int i = 0; i < length; i++) {
    /* … */
  }
}
…
```

Use the –Wall (or –Wextra) flag every time !!!!

```
test.cc: In function ‘void reverse_array(int, unsigned)’:
test.cc:6:22: warning: comparison between signed and
unsigned integer expressions [-Wsign-compare]
    for (int i = 0; i < length; i++) {
```

2. Box-and-arrow diagram

- Use a box-and-arrow diagram for the given program to explain what it prints out.
- Pen & Paper Exercise: hand it in manually or scan it



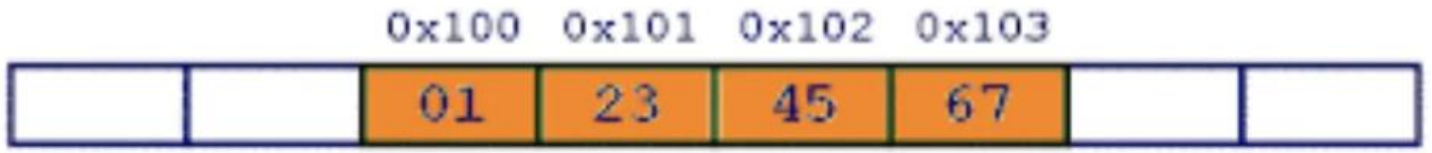
3. Little vs. big endian

Write a C program that prints out whether the computer it is running on is little endian or big endian. (hint: pointers and casts)

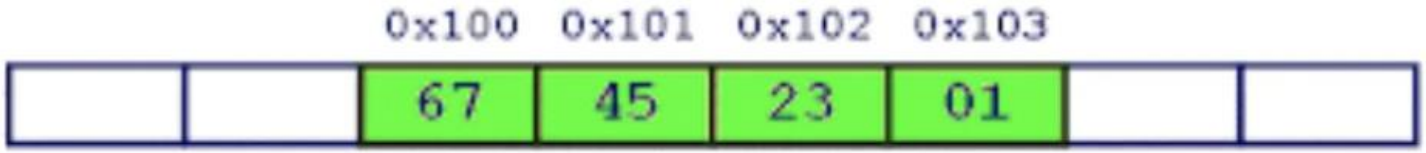
```
…
// returns true if little endian, 0 if big endian
bool is_little_endian() {
        …
}
```

Recap DDCA: Little vs Big Endian

- Example: 0x01234567



Big Endian



Little Endian

4. First whitespace-separated word

Implement a function that extracts the first word from a string

```
…
word find_first_word(char *input_string) {
  word result = {.word_string = NULL, .length = 0};
  // Student TODO: implement here
  return result;
}
…
typedef struct word {
  char *word_string;
  size_t length;
} word;
```

Recap: Strings in C

Systems@ETH Zürich

- string-demo

```
 8   char str1[] = "str1";
 9   char* str2 = "str2";
10   char str3[5] = {'s','t','r','3','\0'};
11
```

Recap: Sizeof vs strlen

Systems@ETH Zürich

- **sizeof**: Determines the size (in bytes) of a data type or variable at compile time. It gives the amount of memory allocated for the object, data type, or array.
  - Sizeof(array) = arraylength * sizeof(element)
  - Sizeof(pointer) = 8
- **strlen**: Calculates the length of a string (i.e., the number of characters before the null terminator \0) at runtime.

Systems@ETH Zürich

string-demo

Initializing Memory
(First whitespace-separated word)

Systems@ETH Zürich

Usage of Malloc

```
#include <string.h>

char *data = (int*)malloc(10 * (sizeof(char)));
if (data == NULL) {
  printf("No memory");
  exit(1);
}
// ensure memory is zeroed out
memset(data, 0, 10 * sizeof(char));
```

malloc() can fail => returns NULL

No guarantee for zeroed memory

Remember, use compiler flags

gcc <FILES> -Wall -Wpedantic -Wextra -Werror -std=c99 -Wmissing-prototypes

5. Complex numbers

For this you must implement a complex number module with following functions:

- add
- subtract
- multiply
- Divide

Recap: Structs in C

Systems@ETH Zürich

Structured data

```
struct typename {
    type name;
    type name;
     …
    type name;
};
```

```
// New structured data type called "struct Point"

struct Point {
    int x;
    int y;
};

struct Point origin = { 0, 0 };
```

- struct: a C type that contains a set of fields
  - a bit like a *class*, but no methods or constructors
  - instances can be allocated on stack or heap

Recap: Difference "." and "->"?

Systems@ETH Zürich

Using `structs`

- Use "." to refer to fields in a struct
- Use "->" to refer to fields through a pointer to a struct

```
struct Point {
  int x, y;
};

int main(int argc, char **argv) {
  struct Point p1 = {0, 0};  // p1 on the stack
  struct Point *p1_ptr = &p1;

  p1.x = 1;
  p1_ptr->y = 2;
  return 0;
}
```

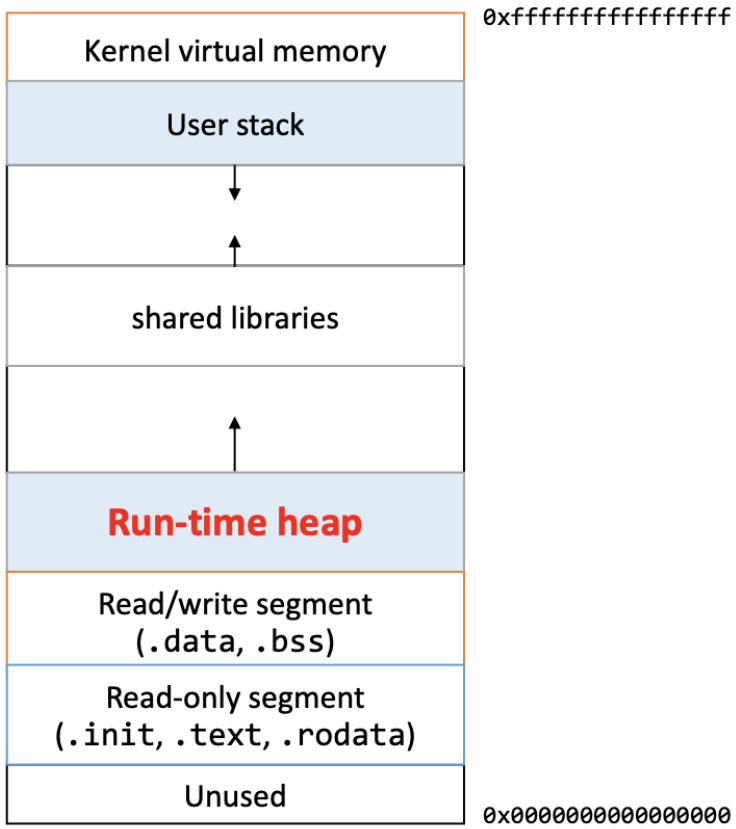
`simplestruct.c`

`p->x`
is nicer than
`(*p).x`

Recap: Stack and Heap

Where does all this memory come from?

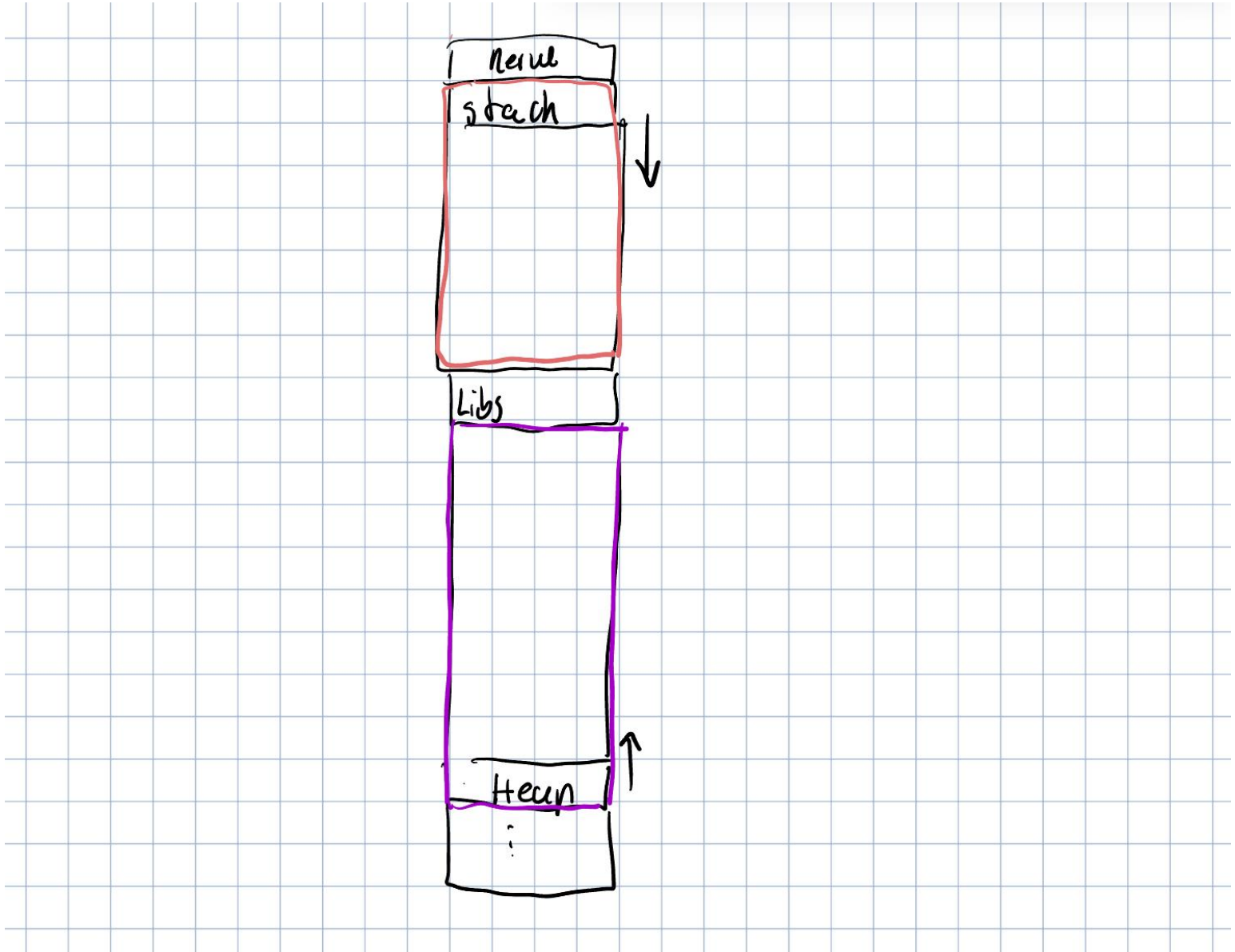
- The heap (or “free store”)
  - Large pool of unused memory, used for dynamically allocated data structures
  - `malloc()` allocates chunks of memory in the heap, `free()` returns them
  - `malloc()` maintains bookkeeping data in the heap to track allocated blocks.



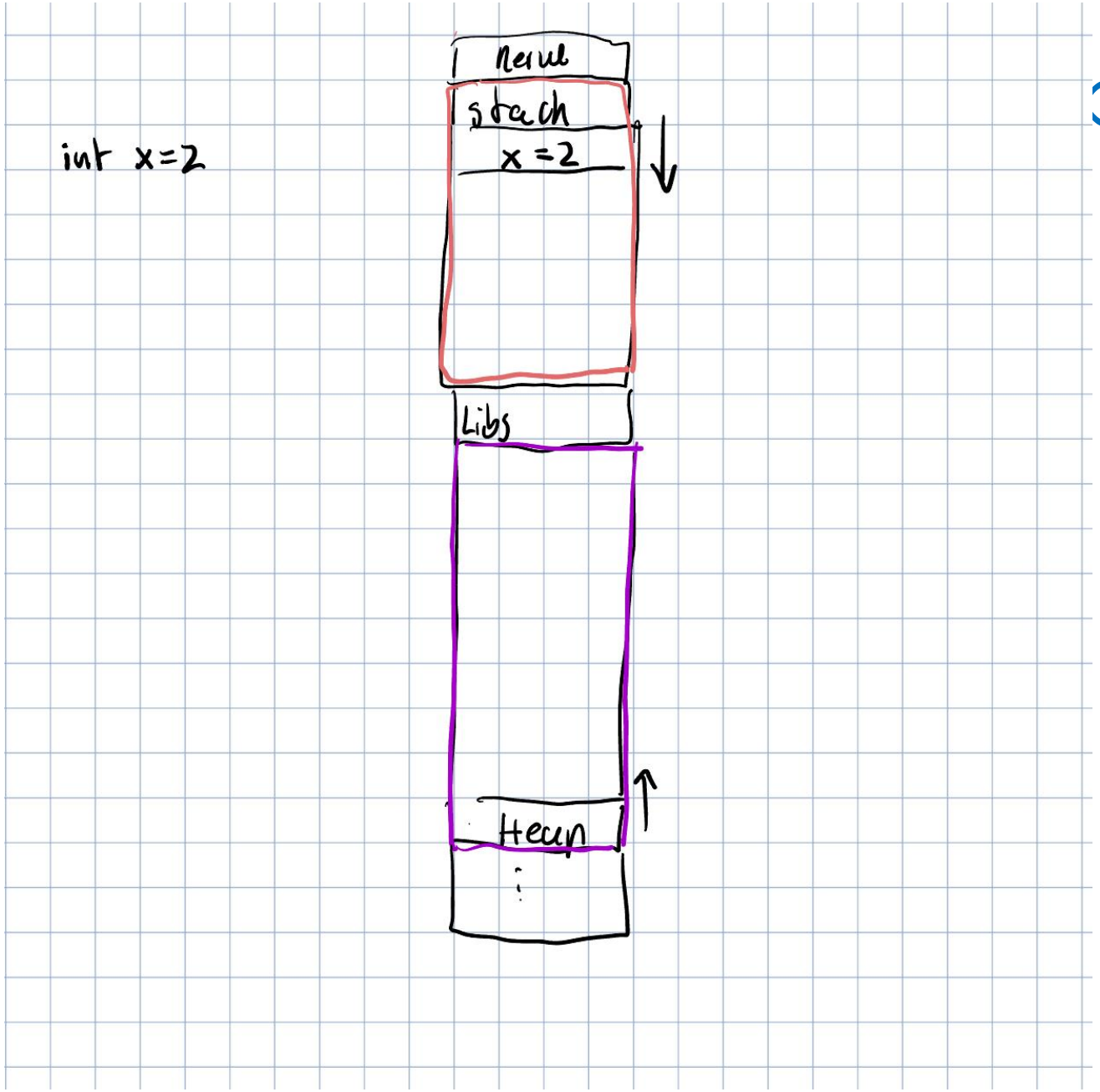
Systems@ETH Zürich

Int on the heap vs stack

Recap: Structs in C, Stack and Heap



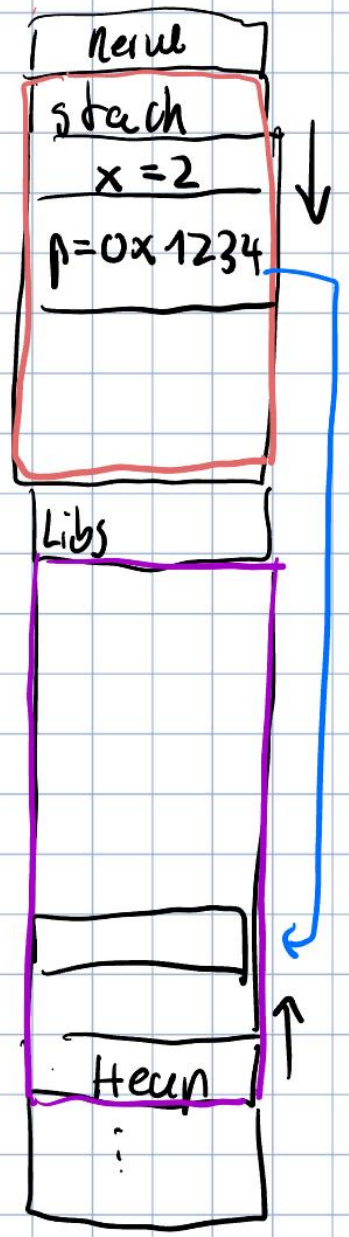
Recap



Systems@ETH Zürich

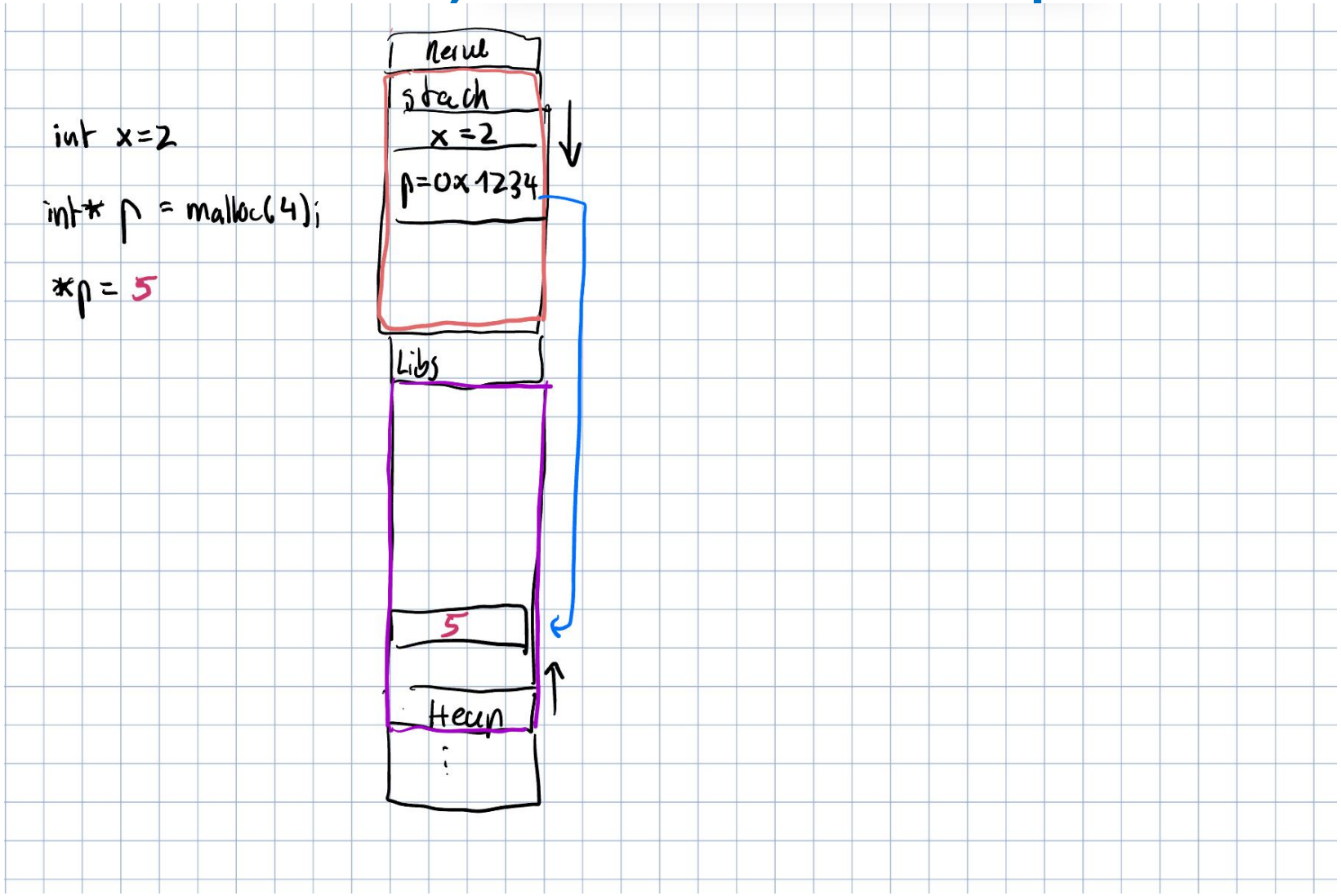
Recap: S

```
int x=2

int* p = malloc(4);
```



Systems@ETH Zürich

Recap: Structs in C, Stack and Heap



Struct on the heap

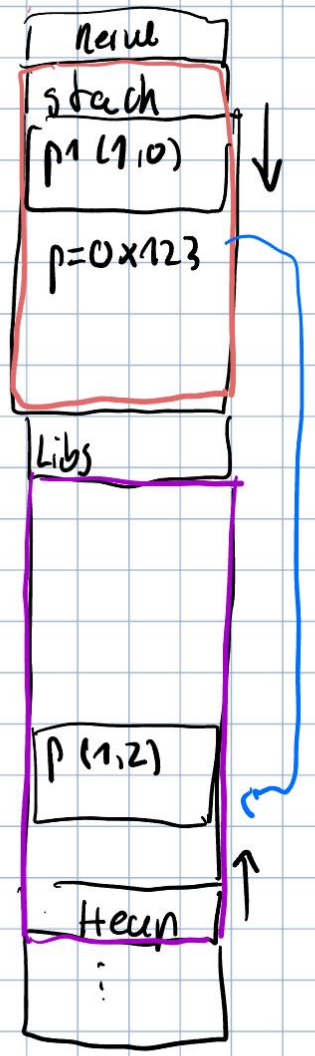
Recap

```
struct Point p1 = {0,0};
struct Point* p = malloc(sizeof(struct));
```

Systems@ETH Zürich

Recap: Structs in C, Stack and Heap

```
struct Point p1 = {0,0};
struct Point* p = malloc(sizeof(struct));
*p = struct Point
p->x = 1;
p->y = 2;
p1.x = 1
```



Recap: Structs in C, Stack and Heap

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

struct Point {
    int x,y;
};

int main(void) {
    struct Point p1 = { .x: 0, .y: 0};
    struct Point* p = (struct Point*) malloc( size: sizeof(struct Point));
    p->x=1;
    p->y=1;

    free(p);

    return 0;
}
```

Recap: Difference “.”, “->” clear?

Systems@ETH Zürich

Using `structs`

- Use "." to refer to fields in a struct
- Use "->" to refer to fields through a pointer to a struct

```
struct Point {
  int x, y;
};

int main(int argc, char **argv) {
  struct Point p1 = {0, 0};  // p1 on the stack
  struct Point *p1_ptr = &p1;

  p1.x = 1;
  p1_ptr->y = 2;
  return 0;
}
```

`simplestruct.c`

`p->x`
is nicer than
`(*p).x`

Systems@ETH Zürich

Note: doesnt have to be on the heap to have a pointer to it

6. Binary Search Tree

Implement the three functions insert(), lookup() and delete()
but be careful during allocations:

```
tree_node* alloc_tree_node() {
        tree_node* ret = (tree_node*)malloc(sizeof(tree_node));
        ret->key = -1;
        ret->value = -1
        ret->left = NULL;
        ret->right = NULL;
        return ret;
}
```

- malloc() can fail, returns NULL.
- Correct code must not leak memory.

Cleanup code

```
tree_node* alloc_tree_node() {
        tree_node* ret = (tree_node*)malloc(sizeof(tree_node));

        if (!ret)
             return NULL;

        ret->key = -1;
        ret->value = -1
        ret->left = NULL;
        ret->right = NULL;


        return ret;
}
```

7. Word Count - wc

What is wc?

wc is a Unix utility that displays the count of characters, words and lines present in a file.

Implement this unix utility step by step while solving one problem at a time.

Start from the given code-shell in wc.c and then just add the missing components.

8. File I/O (File descriptors, read, write)

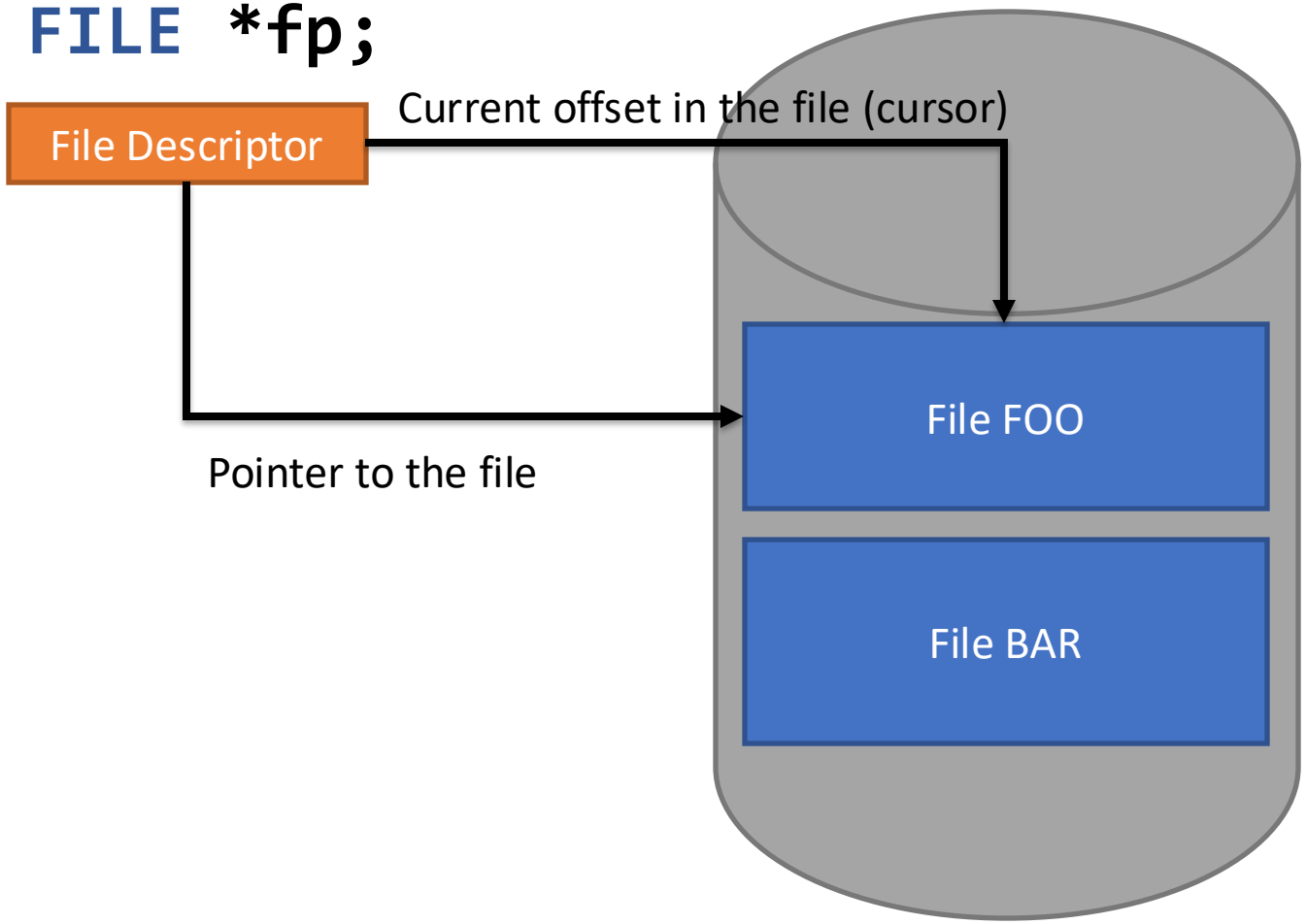
Accessing Files

Systems@ETH Zürich

- In general, you cannot access the file directly
- You need support from the operating system to open/read/write/close a file.
  - This is called a system call (syscall) -> Lecture Computer Systems
- All file related declarations are in the stdio.h
  - This functions are wrappers around system calls, e.g., *fread()* usually calls the system call *read()*.

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

The File Descriptor



- To get a file descriptor you must open a file

```
FILE *fopen(const char *filename,
            const char *mode);
```

- The opening may fail. Check return value!
- Close the file in the end

```
int fclose(FILE *fp);
```

(Text) File Opening Modes

Mode	Read	Write	File Not Exists	File Exists
r	Yes, from beginning	No	Error	FILE* descriptor returned
w	No	Yes, from the beginning	New file created, FILE* Descriptor returned	FILE* descriptor returned
a	No	Yes, from the end (append)	New file crated, FILE* descriptor returned	FILE* descriptor returned
r+	Yes, from beginning	Yes, from beginning	Error	FILE* descriptor returned.
w+	Yes, from beginning	Yes, From beginning	New file created, FILE* descriptor returned	Delete file contents (overwrite), File* descriptor returned
a+	Yes, from beginning	Yes, from the end (append only)	New file crated, FILE* descriptor returned	FILE* descriptor returned
.. b	Binary flag that can be added for binary IO			

(Text) File Opening Modes

Mode	Read	Write	File Not Exists	File Exists
r	Yes, from beginning		Error	FILE* descriptor returned
w	No		New file created, FILE* descriptor returned	FILE* descriptor returned
a	No	Yes, from the end (append)	New file created, FILE* descriptor returned	FILE* descriptor returned
r+	Yes, from beginning	Yes, from beginning	Error	FILE* descriptor returned.
w+	Yes, from beginning	Yes, From beginning	New file created, FILE* descriptor returned	Delete file contents (overwrite), File* descriptor returned
a+	Yes, from beginning	Yes, from the end (append only)	New file crated, FILE* descriptor returned	FILE* descriptor returned
.. b	Binary flag that can be added for binary IO			

What you (probably) want for the assignment

Reading a Text File (Example)

```
FILE *fp;
int c;
int n = 0;
fp=fopen("myfile.txt","r");
if (fp ==NULL) {
  printf("Error opening file");
} else {
  do {
    c = fgetc (fp);
    if (c == ‘A') {
      n++;
    }
  } while (c != EOF);
  fclose (fp);
}
printf(“%i“, n);
```

Alternatives…

```
int fgetc(FILE *fp);
char *fgets(char *str, int count, FILE *fp);
size_t fread(void *buf, size_t size,
             size_t count, FILE *fp);
```

Writing a Text File (Example)

```c
FILE *fp;
char name];

printf(“Enter your name: ");
fgets (name, 256, stdin);

fp=fopen("myfile.txt",“a");

if (fp ==NULL) {
  printf("Error opening file");
} else {
  if ( fputs (name, fp) < 0) {
    printf("Error writing file");
  }
  fclose (fp);
}
```

Alternatives…

```c
int fputc(int character, FILE *fp);
int *fputs(char *str, FILE *fp);
size_t fwrite(void *ptr, size_t size,
              size_t count, FILE *fp);
```

Own Example: readfile.c, .h

```h
int print(char[]);
```

```c
#include <stdio.h>
#include "readfile.h"

int print(char fn[]){
  FILE *fp;
  char c;
  fp = fopen(fn, "r");
  if(fp==NULL) return -1;

  while( (c = getc(fp)) != EOF ) {
    printf("%c", c);
  }

  printf("\n");

  fclose(fp);
  return 0;
}
```

Own Example: writefile.c

```c
#include <stdio.h>
#include "readfile.h"


int main(int argc, char** argv){
  print("read.txt");


  FILE *fp;
  fp = fopen("write.txt", "w"); //w for writemode
  if(fp==NULL) return -1;

  char name[256];

  printf("Name: ");
  fgets(name, 256, stdin);

  if( (fputs(name, fp)) < 0) return -1;

  fclose(fp);

  print("write.txt");

  return 0;
}
```

Systems@ETH Zürich

file-demo

Always keep in mind

Systems@ETH Zürich

- All those functions will change the state of the file descriptor by advancing the cursor position

- Sometimes not all data requested is read/written
  - Keep track of how many bytes are processed
  - Loop until finished

Reading Formatted strings

- `int fscanf(FILE *stream, const char *format, ...);`
  - Version of scanf() that reads from given `FILE*`.
  - `int a; char b[10]; fscanf(stdin, "%d %s", &a, &b);`
  - Also, `sscanf(const char *str, const char *fmt, ...)` may be useful

- Format string:
  - %s Read string
  - %d Read integer

File IO: Resources

- Reference and Examples
  http://www.cplusplus.com/reference/cstdio/

- Read the man pages!
  - man 3 getc
  - man 3 isspace
  - man 3 scanf
- http://stackoverflow.com

9. Function pointers basics

Implement the function array_apply() that:

- accepts a function pointer, an array of integers and the arrays length
- invokes the pointed-to function with each of the elements in the array as an argument
- overrides the current array element with the return value of the called function

```
void array_apply(Function func, int *values, size_t length) {
…
}

int pow2(int a) {
      return a*a;
}
```

10. Function pointer

Systems@ETH Zürich

This last part will help you get even more familiar with function pointers.

- write a function to lexicographically compare first names
- write another function for comparison of last names
- use apply and a callback to call a function on every element of an array
- BONUS: implement your own mysort()

Function Pointers conretely

```c
#include <stdio.h>

// Two simple functions
void fun1() {
  printf("function1\n");
}
void fun2() {
  printf("function2\n");
}

// A function that receives a simple function
// as parameter and calls the function
void wrapper(void (*fun)()) {
  fun();
}

int main(){
    wrapper(fun1);
    wrapper(fun2);
    return 0;
}

```

Usecase?

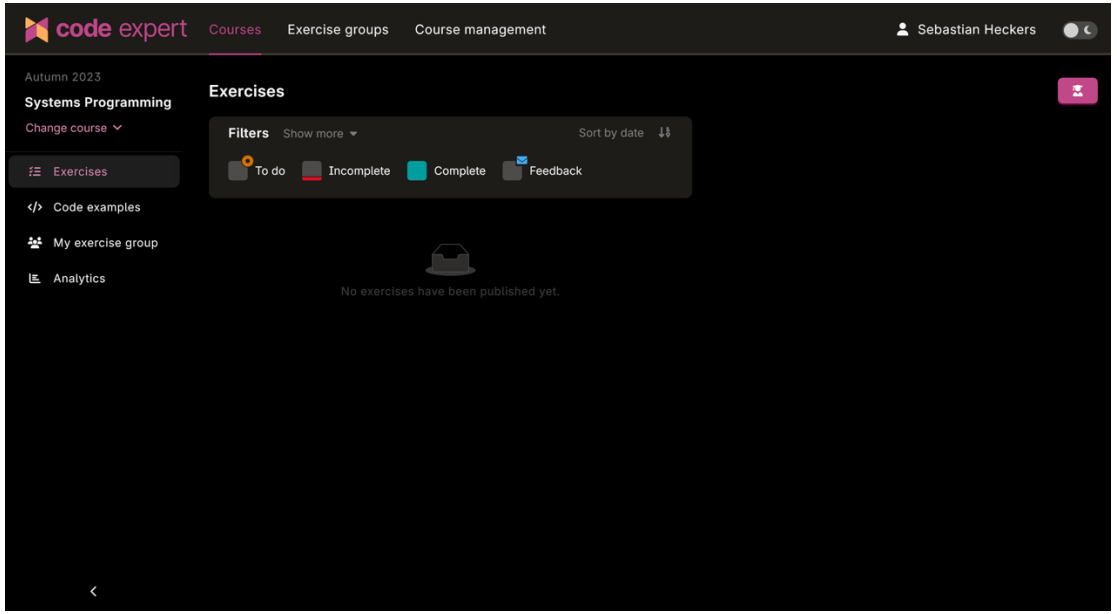

```c
int add(int a, int b);
int multiply(int a, int b);
void performOperation(int (*operation)(int, int), int x, int y);

int main() {
    int num1, num2;
    char operationChoice;

    // User input
    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    printf("Choose op (+,*): ");
    scanf(" %c", &operationChoice);

    // Declare a function pointer
    int (*operation)(int, int) = NULL;

    // Select operation based on user choice
    if (operationChoice == '+') {
        operation = add; // Point to add function
    } else if (operationChoice == '*') {
        operation = multiply; // Point to multiply function
    } else {
        printf("Invalid operation!\n");
        return 1; // Exit if invalid operation
    }

    // Perform the operation
    performOperation(operation, num1, num2);

    return 0;
}
```

```c
// Function definitions
int add(int a, int b) {
    return a + b; // Return sum
}

int multiply(int a, int b) {
    return a * b; // Return product
}

void performOperation(int (*operation)(int, int), int x,
    int result = operation(x, y); // Call the function th
    printf("Result: %d\n", result);
}
```

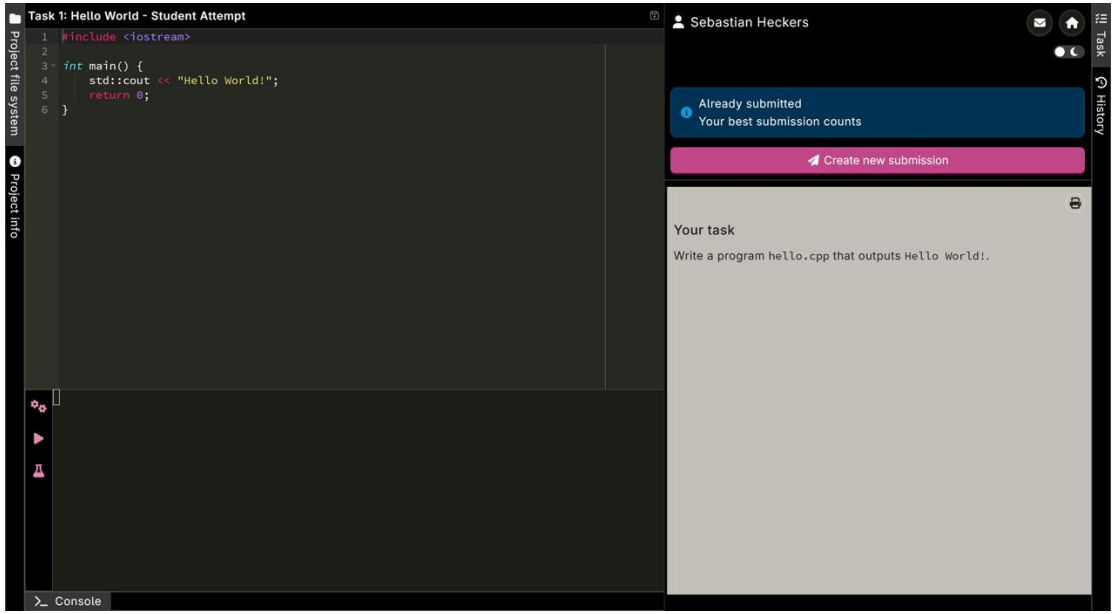
Function pointer demo

Assignment 2

All programming exercises are on CodeExpert, except exercise 2 (Box-and-arrow diagram) and exercise 3 (Little vs. big endian).



Start page with the exercises



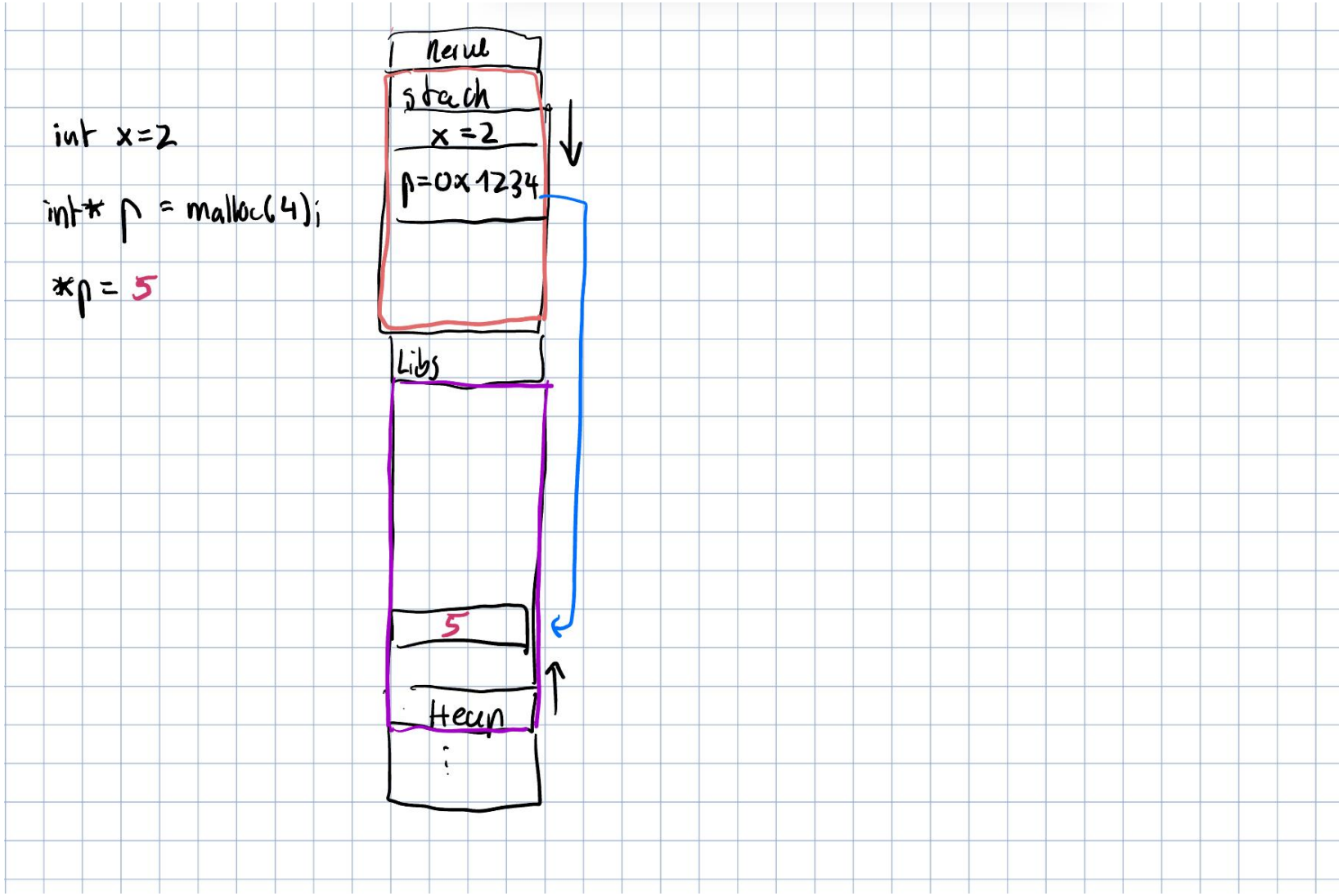
Exercise in the IDE

More information about CodeExpert: https://docs.expert.ethz.ch/

Short Quiz about pointers

to get you started thinking of pointers

Recap: Pointers



Recap: Pointers

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

int main(void) {
    int x = 2;
    printf("addr x: %p\n\n", &x);

    int* p = malloc( size: sizeof(int));
    *p=5;

    printf("dereferences val of p: %d\n", *p);
    printf("val of p itself: %p\n", p);
    printf("addr of p itself: %p\n", &p);

    free(p);
    return 0;
}
```

```
/Users/benediktfalk/CLionProjects/untitled8
addr x: 0x7ff7bc78f548

dereferences val of p: 5
val of p itseöf: 52215856
addr of p itseöf: -1132923584

Process finished with exit code 0
```

Recap: Pointer Arithmetic

- For pointer p, p=p+1 depends on **of what type of pointer it is**
- Sizeof(int)=4
- int* p ⇔ 0x0
- p++ ⇔ 0x4

- For char, sizeof(char)=1
- char* ptr ⇔ 0x5
- ptr++ ⇔ 0x6

Recap: Pointer Arithmetic

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

int main(void) {
    printf("sizeof(int): %d\n", sizeof(int));
    int *p = (int*) 0x0;
    printf("%p\n",p);
    p++;
    printf("%p\n\n",p);

    printf("sizeof(int): %d\n", sizeof(char));
    char *ptr = (char*) 0x5;
    printf("%p\n",ptr);
    ptr++;
    printf("%p\n",ptr);


    return 0;
}
```

```
/Users/benediktfalk/CLionProjects/
sizeof(int): 4
0x0
0x4

sizeof(int): 1
0x5
0x6
```

Recap: Pointer Arithmetic

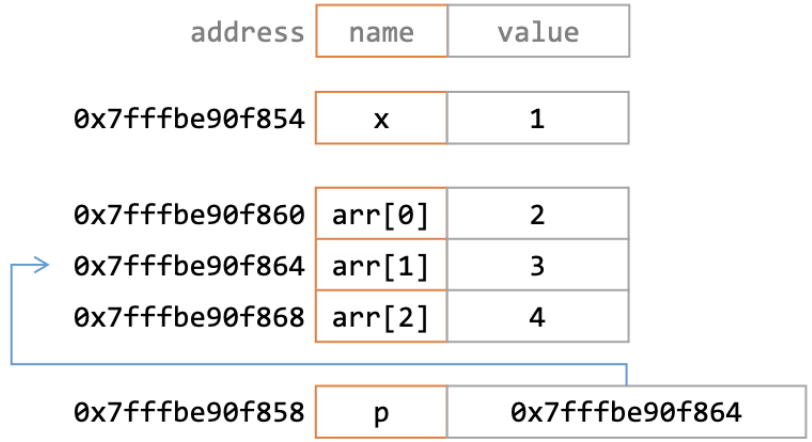
Pointer arithmetic

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];


  *p += 1;      // *p = *p + 1
  p += 1;
  *p += 1;

  return 0;
}
```



Recap: Pointer Arithmetic

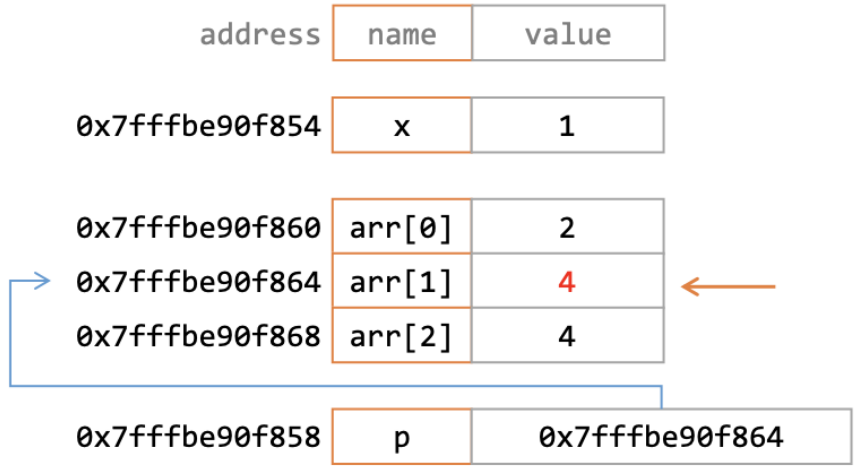
Pointer arithmetic

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];


  *p += 1;
  p += 1;
  *p += 1;

  return 0;
}
```



Recap: Pointer Arithmetic

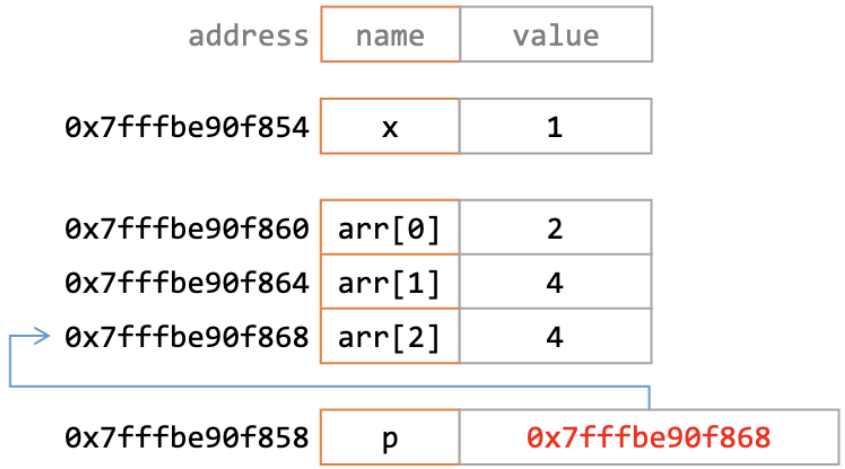
Pointer arithmetic

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];


  *p += 1;
  p += 1;
  *p += 1;

  return 0;
}
```



Recap: Pointer Arithmetic

Pointer arithmetic

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int x = 1;
  int arr[3] = {2, 3, 4};
  int *p = &arr[1];

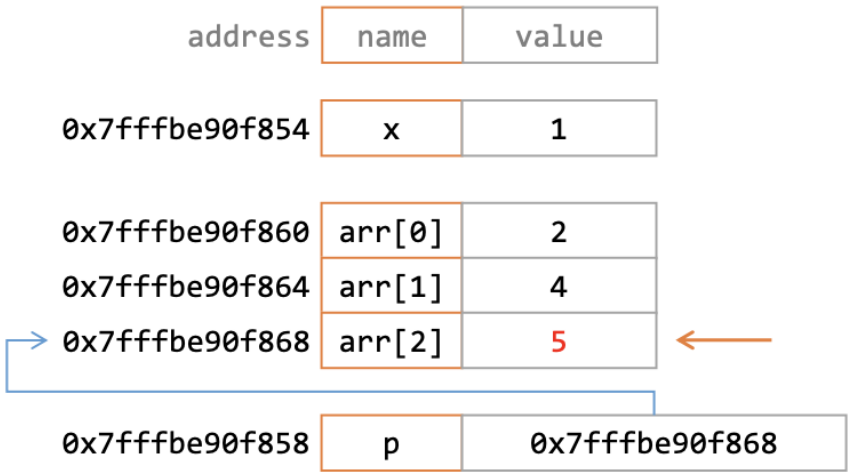

  *p += 1;
  p += 1;
  *p += 1;

  return 0;
}
```

- Pointers are typed:

```
int *int_ptr;
char *char_ptr;
int **ptr_ptr;
```

- Pointer arithmetic follows pointer type



Recap: Pointers and Arrays

In fact…

- `A[i]` is *always* rewritten `*(A+i)` in the compiler

```
int a[10];
assert(a == &(a[0]));
assert(a[5] == 5[a]);
```

```
int get_2(int i)
{
    int *p = a;
    return i[p];
}
void set_2(int i, int v)
{
    int *p = a;
    i[p] = v;
}
```

Quiz: Simple Pointer

```
int a[] = { 0,1,2,3,4 };

int i, *p;
for (p = &a[0], i=0; p+i <= a+4; p++,i++)
     printf("*(p+i) = %d", *(p+i));
```

Solution: Simple Pointer

```
*p+i = 0   *p+i = 2   *p+i = 4
```

Quiz: Arrays and Pointers

```
int a[] = { 0, 1, 2, 3, 4 };
int *p[] = {a, a+1, a+2, a+3, a+4 };
int **pp = p;

main() {
printf("…", a, *a, **a);
printf("…", p, *p, **p);
printf("…", pp, *pp, **pp);
}
```

Solution: Arrays and Pointers

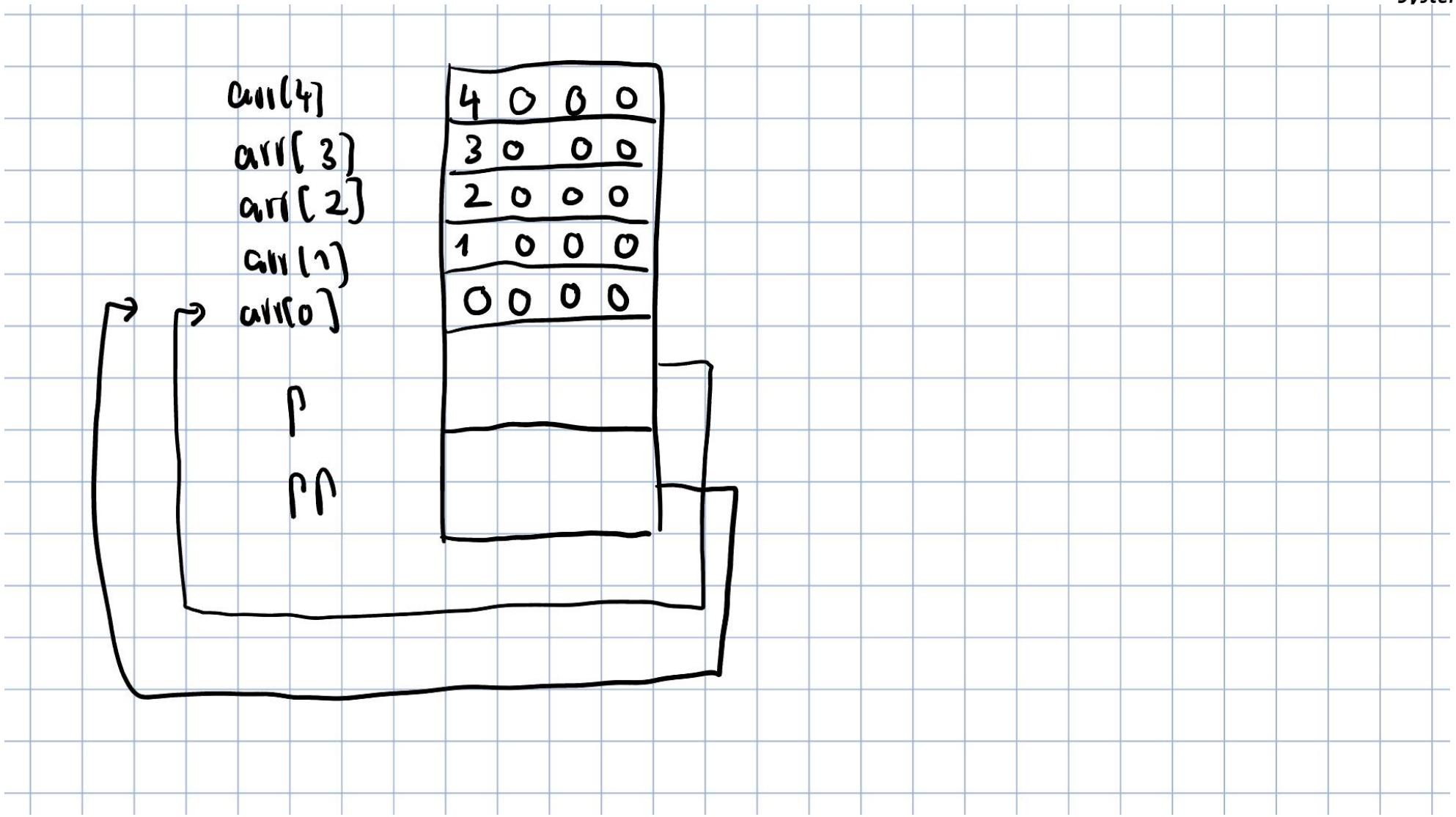
Systems@ETH Zürich

- ```
  a = address of a
  *a = 0
  **a = Segmentation Fault (Null pointer dereference)
  ```

- ```
  p = address of p
  *p = address of a
  **p = 0
  ```

- ```
  pp = address of p
  *pp = address of a
  **pp = 0
  ```
  ```

Box and Arrow



Code

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

int main(void) {
    int a[] = { [0]: 0, [1]: 1, [2]: 2, [3]: 3, [4]: 4};
    int *p[] = { [0]: a, [1]: a+1, [2]: a+2, [3]: a+3, [4]: a+4};
    int **pp=p; //poits to the SAME value as p points to, i.e. a

    printf("%p\n", a);
    printf("%p\n", p);
    printf("%p\n", pp);
    return 0;
}
```

```
/Users/benediktfalk/CLionProjects/untitled8/
0x7ff7b9132530
0x7ff7b9132500
0x7ff7b9132500
```

Recap: Sizeof vs strlen

Systems@ETH Zürich

- **sizeof**: Determines the size (in bytes) of a data type or variable at compile time. It gives the amount of memory allocated for the object, data type, or array.
  - Sizeof(array) = arraylength * sizeof(element)
  - Sizeof(pointer) = 8
- **strlen**: Calculates the length of a string (i.e., the number of characters before the null terminator \0) at runtime.

What does it print?

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

int main(int argc, char *argv[])
{
char s1[100] = "Simonetta Sommaruga";
char s2[100] = "Guy Parmelin";
char *s3 = "Ignazio Cassis";
printf("strlen(s1) = %lu\n", strlen(s1));
return 0;
}
```

Compiled and executed on a 64-bit Linux machine

Answer `strlen(s1) = 19`

Systems Programming and Computer Architecture Exam, Autumn Term 2022, Question 9

What does it print?

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

int main(int argc, char *argv[])
{
char s1[100] = "Simonetta Sommaruga";
char s2[100] = "Guy Parmelin";
char *s3 = "Ignazio Cassis";
printf("sizeof(s2) = %lu\n", sizeof(s2));
return 0;
}
```

Compiled and executed on a 64-bit Linux machine

Answer `sizeof(s2) = 100`

Systems Programming and Computer Architecture Exam, Autumn Term 2022, Question 9

What does it print?

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

int main(int argc, char *argv[])
{
char s1[100] = "Simonetta Sommaruga";
char s2[100] = "Guy Parmelin";
char *s3 = "Ignazio Cassis";
printf("sizeof(s3) = %lu\n", sizeof(s3));
return 0;
}
```

Compiled and executed on a 64-bit Linux machine

Answer `sizeof(s3) = 8`

Systems Programming and Computer Architecture Exam, Autumn Term 2022, Question 9

What does it print?

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

int main(int argc, char *argv[])
{
char s1[100] = "Simonetta Sommaruga";
char s2[100] = "Guy Parmelin";
char *s3 = "Ignazio Cassis";
printf("strlen(s3) = %lu\n", strlen(s3));
return 0;
}
```

Compiled and executed on a 64-bit Linux machine

Answer `strlen(s3) = 14`

Systems Programming and Computer Architecture Exam, Autumn Term 2022, Question 9

What does it print?

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

int main(int argc, char *argv[])
{
char s1[100] = "Simonetta Sommaruga";
char s2[100] = "Guy Parmelin";
char *s3 = "Ignazio Cassis";
printf("s1[strlen(s1)] = %d\n", s1[strlen(s1)]);
return 0;
}
```

Compiled and executed on a 64-bit Linux machine

Answer `s1[strlen(s1)] = 0`

Systems Programming and Computer Architecture Exam, Autumn Term 2022, Question 9

What does it print?

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

int main(int argc, char *argv[])
{
char s1[100] = "Simonetta Sommaruga";
char s2[100] = "Guy Parmelin";
char *s3 = "Ignazio Cassis";
s3 += 4;
printf("s3 = ’%s’\n", s3);
return 0;
}
```

Answer `s3 = ’zio Cassis’`

Compiled and executed on a 64-bit Linux machine

Systems Programming and Computer Architecture Exam, Autumn Term 2022, Question 9

What does it print?

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

int main(int argc, char *argv[])
{
char s1[100] = "Simonetta Sommaruga";
char s2[100] = "Guy Parmelin";
char *s3 = "Ignazio Cassis";
*(s1 + 4) = '\0';
printf("s1 = '%s'\n", s1);
return 0;
}
```

Answer `s1 = 'Simo'`

Compiled and executed on a 64-bit Linux machine

Systems Programming and Computer Architecture Exam, Autumn Term 2022, Question 9

What does it print?

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

int main(int argc, char *argv[])
{
char s1[100] = "Simonetta Sommaruga";
char s2[100] = "Guy Parmelin";
char *s3 = "Ignazio Cassis";
*(s2 + 6) = 0;
printf("strlen(s2) = %lu\n", strlen(s2));
return 0;
}
```

Answer `strlen(s2) = 6`

Compiled and executed on a 64-bit Linux machine

Systems Programming and Computer Architecture Exam, Autumn Term 2022, Question 9

See you
next week!