Operating System Design

Threads

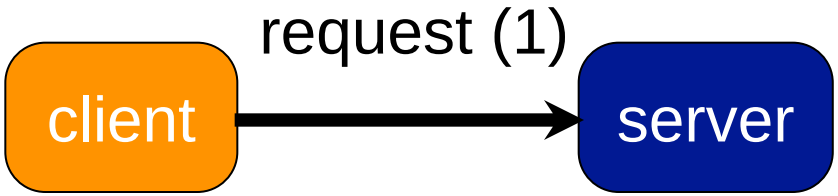
Neda Nasiriani

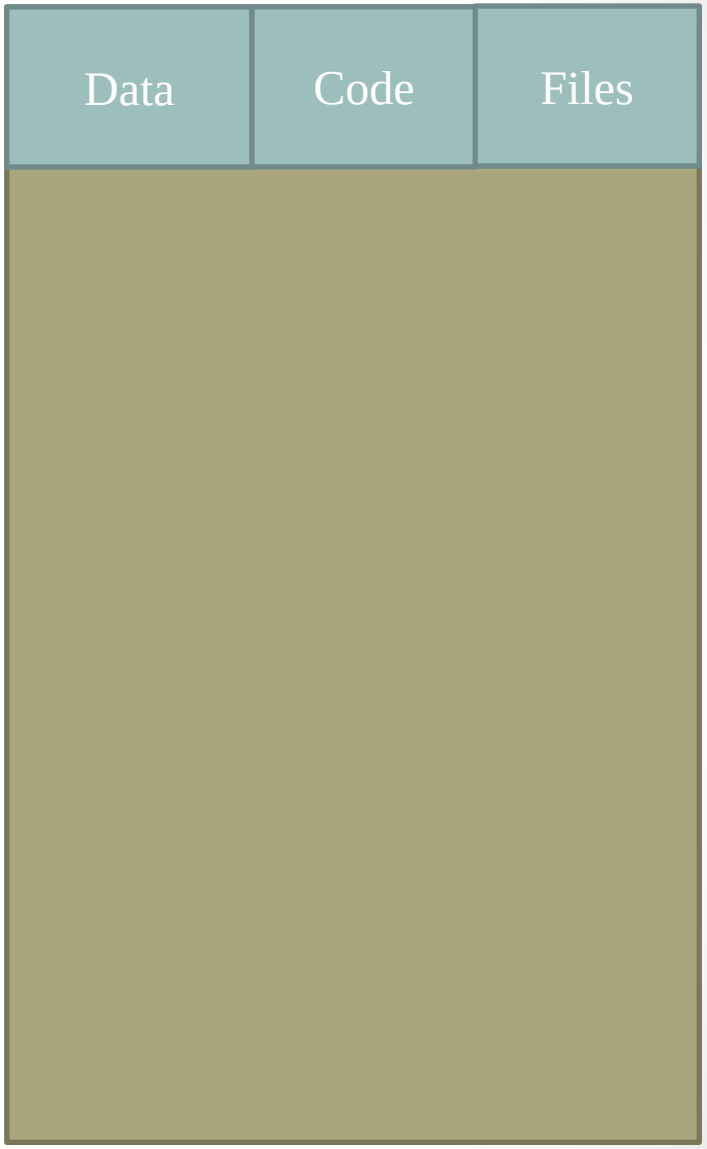
Fall 2018

Web Server Example

- Assume the web server is servicing search request (google search engine)
- Each request to be served is of similar nature (repetitive code) and has to work on the same information (repetitive data)
- What if we could have multiple executions of the same search code within the same process?
- Let’s see how it looks!

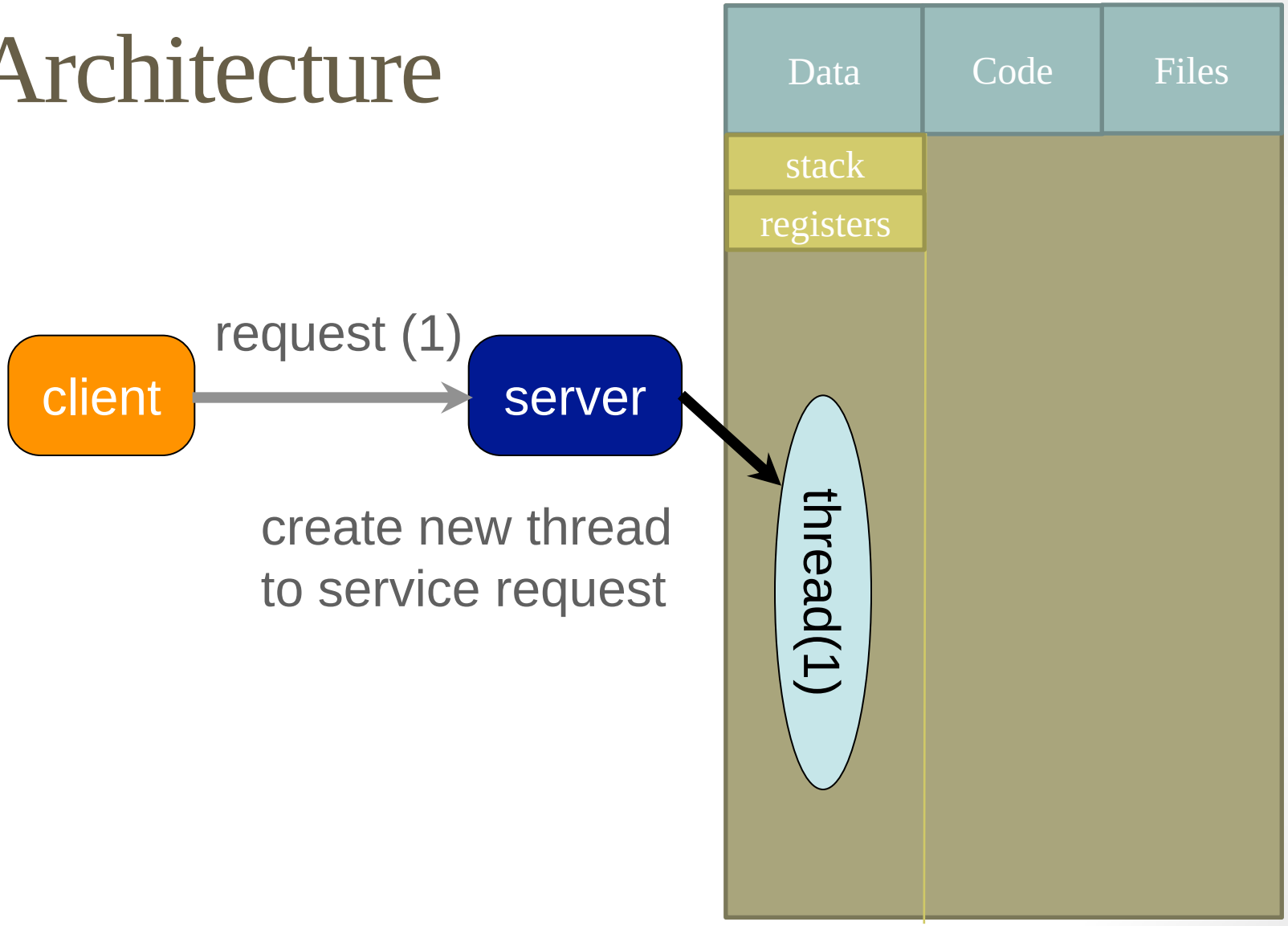
Multithreaded Server Architecture





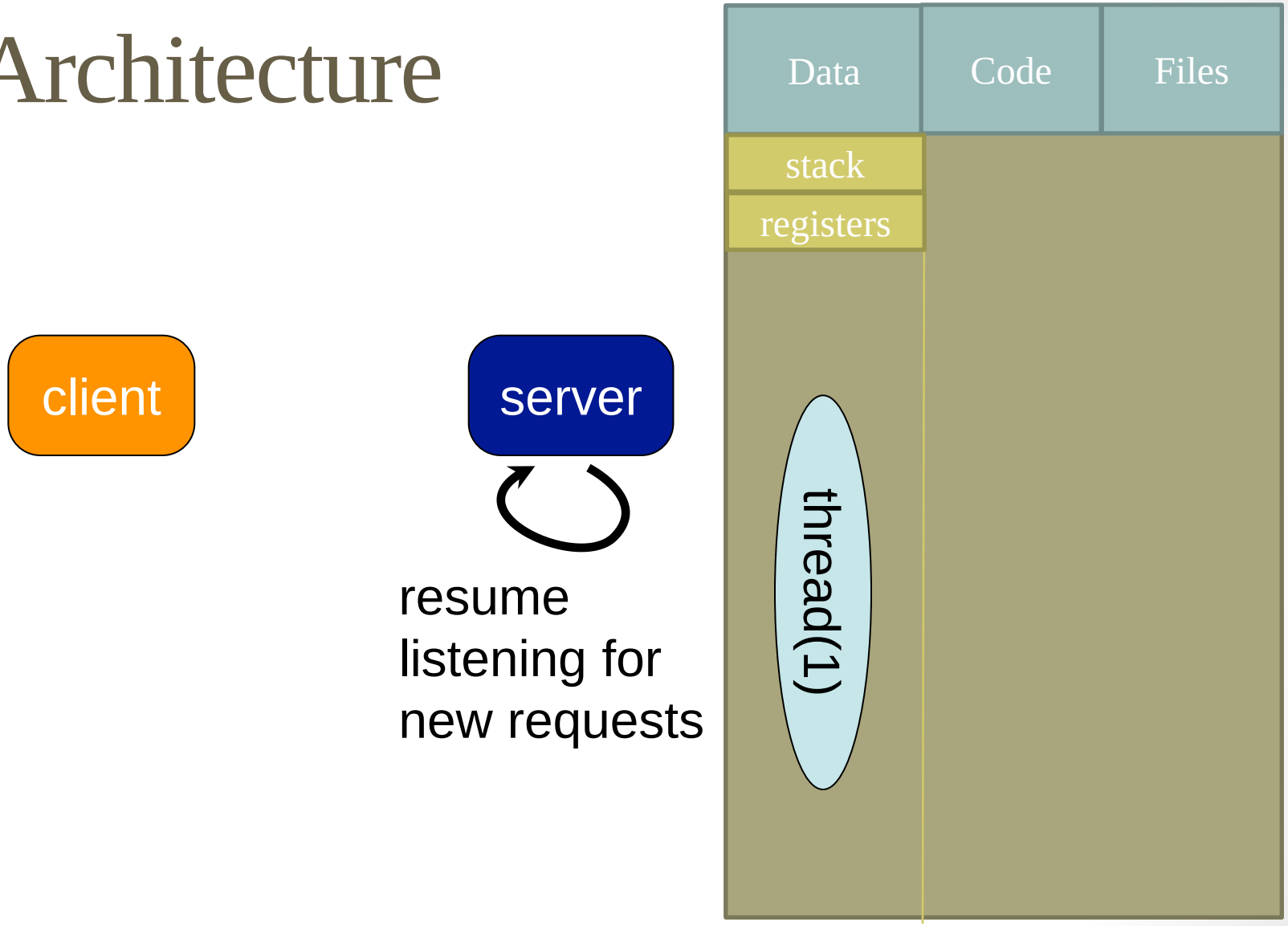
Based on slides from Luiz F. Perrone

Multithreaded Server Architecture



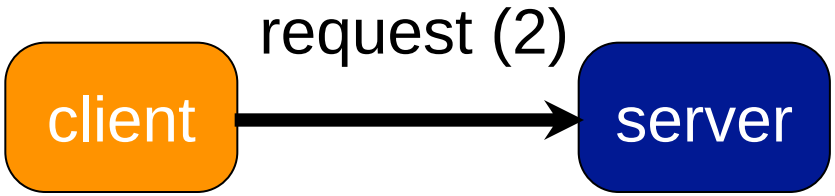
Based on slides from Luiz F. Perrone

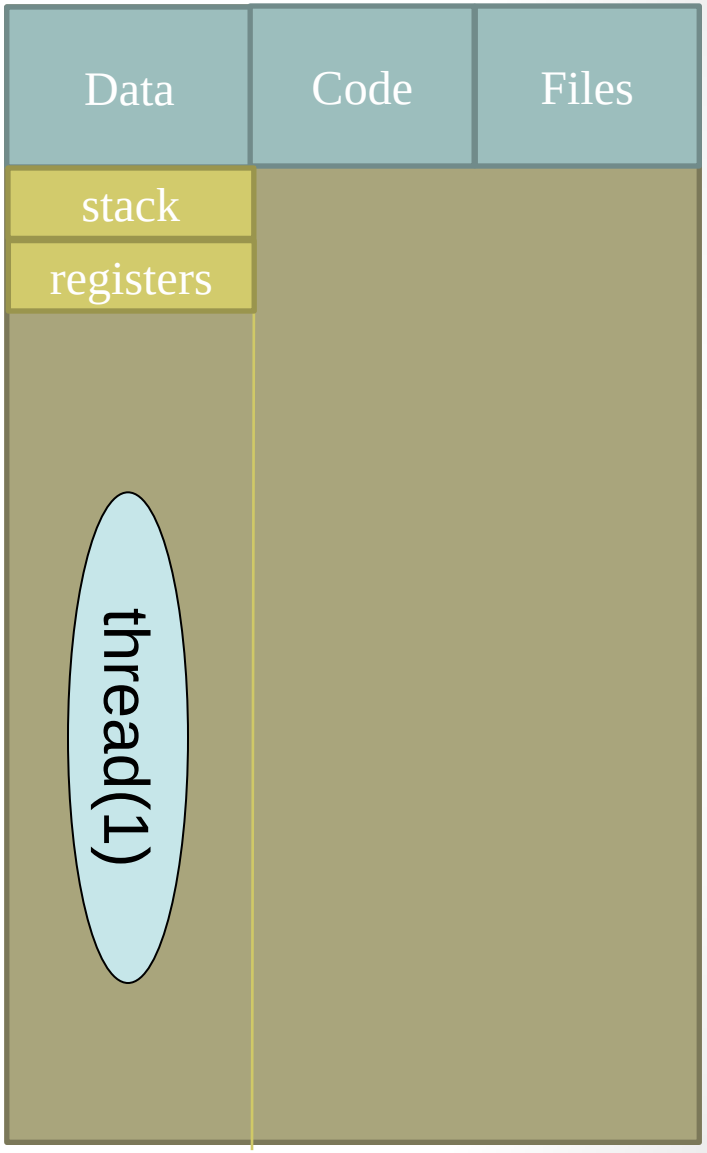
Multithreaded Server Architecture



Based on slides from Luiz F. Perrone

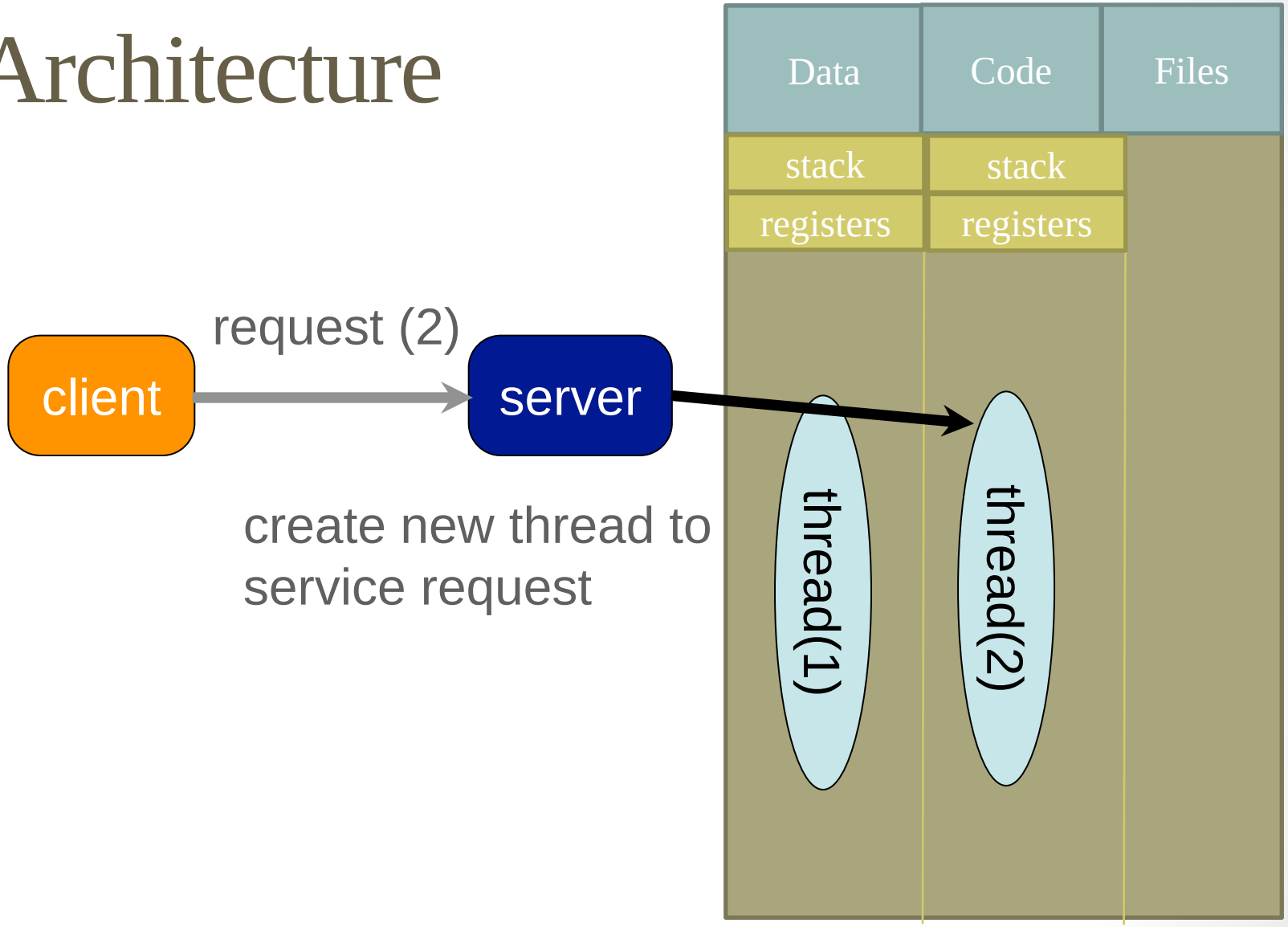
Multithreaded Server Architecture





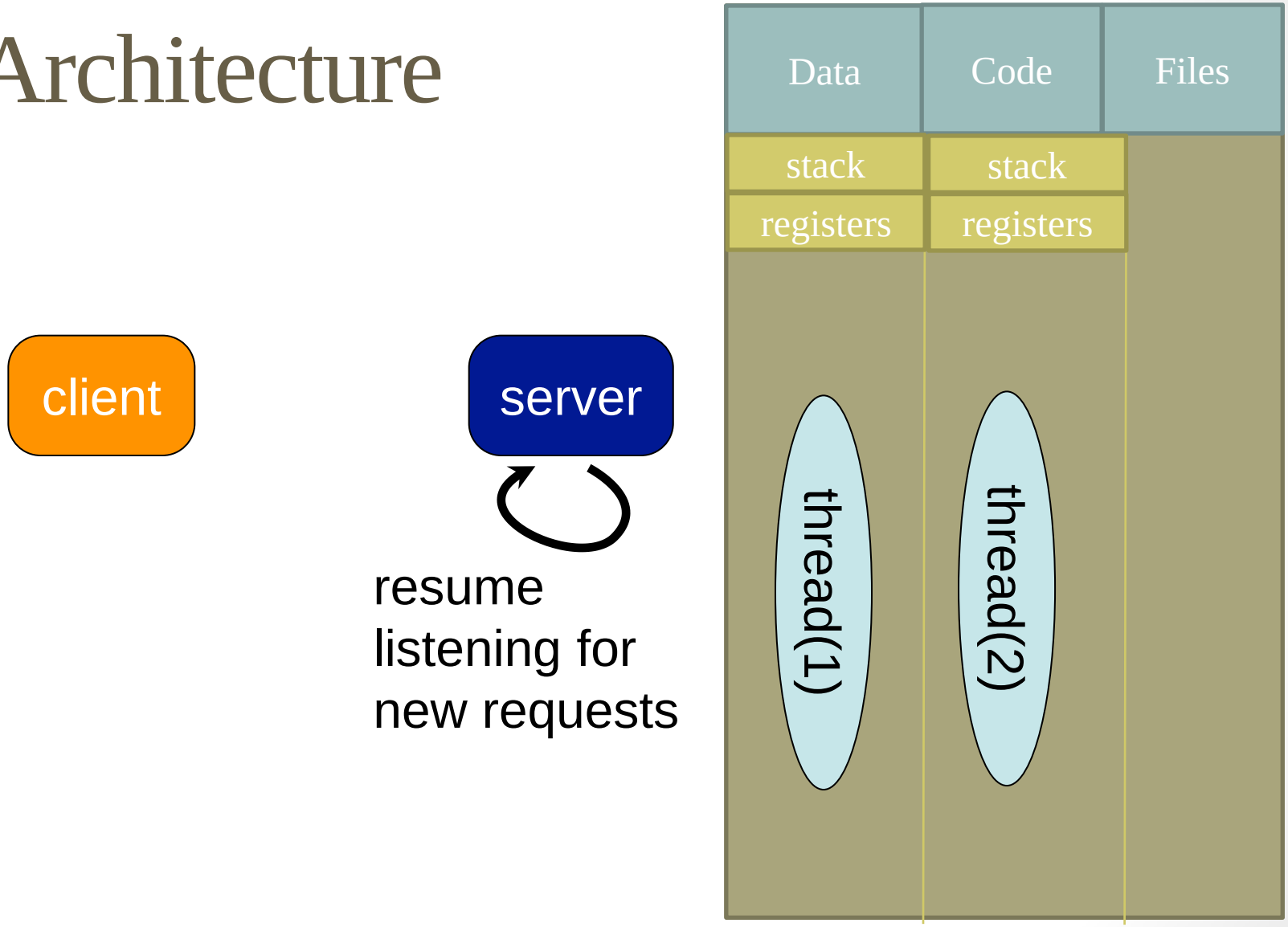
Based on slides from Luiz F. Perrone

Multithreaded Server Architecture



Based on slides from Luiz F. Perrone

Multithreaded Server Architecture



Based on slides from Luiz F. Perrone

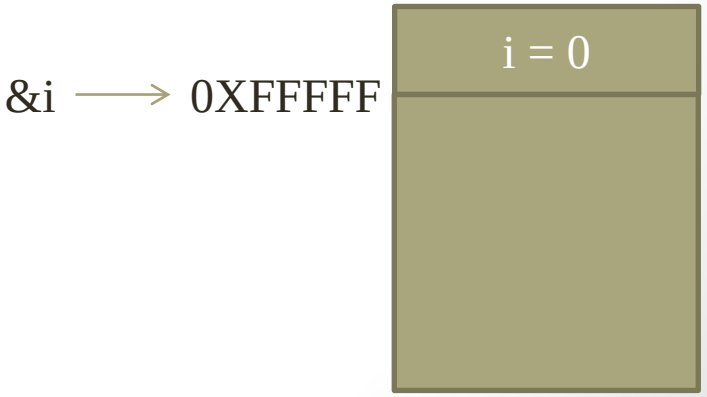
Threads vs. Processes

- What are the advantages of Threads over Processes
  - Light weight
  - Processes are costly to create (around 30X time more time)
  - Context switching processes can take up to 5X more time
  - More Efficient
  - Sharing data is easier
  - PCBs are large data types while TCB are way smaller
- What are the disadvantages and challenges of Threads
  - We need consistency when accessing the shared data
  - We should implement synchronization method among threads accessing the shared data
  - Can complicates execution

Pointers in C

- What is a pointer in C?
  - It is the address to a location in memory
  - C pointers should know what is the data type that they are referring to, it could be an integer, char, another pointer, …. Except for void * which can be any pointer to any data type.
  - Why?
    - Different variables types take up different amounts of memory (int: 2 bytes, long: 4 bytes, char: 1 byte,…)
    - Program needs to know what is a pointer referring to (for pointer operations)
- How can we find the address of a variable?
  - Using & operand
  - Example:

    int i = 0;



Pointers in C

- The beauty of C is that we can define data types for pointers
  - How can we declare a pointer?
    - Using * operand before the variable name specifies that it is a pointer

```
char  *str;
int   *pi;
float *pf;
struct employee_node *new_node;
```

- int *pi tells the compiler that pi is a pointer and *pi is an integer.
- What are pointers initialized to at declaration?
  - NULL -> be careful!!!!
  - Pointers are dynamically assigned and hence need to allocate memory for them.
  - So if you need a variable to send as a status parameter to a function (e.g., wait function) you should declare an int and then send the pointer to that variable.

Function Pointer in C

- What is the use of function pointers?
  - We saw that for creating threads we can send a pointer to a function so the thread can start executing that function. Isn't this neat?
- Function pointer, points to a chunk of code
- How should we declare a function pointer?
  - We need to declare the type of function that we are pointing to
  - We can specify function type by its return value type and its input arguments and their types (basically the function prototype)
- Consider the following function
  - void ToUpper(char *);
- How can we define a function pointer for this function?
  - void (*funcptr) (char *);

Function Pointer in C

- Consider the following function
  - int sum(int a, int b);
- How can we define a function pointer for this function?
  - int (*pf) (int, int);
  - You have to substitute the name of the function with (*var_name) for defining a function pointer to a function
  - (*var_name) specifies a pointer to a function
  - How can we assign a function to a function pointer
    - pf = sum
    - pf = &sum
  - How can you invoke the function pointer?
    - pf (5,4)
    - (*pf)(5,4)
    - What about *pf(5,4)?

Create Threads

```
NAME
       pthread_create - create a new thread

SYNOPSIS
       #include <pthread.h>

       int pthread_create(pthread_t *thread,
                          const pthread_attr_t *attr,
                          void *(*start_routine) (void *),
                          void *arg);

       Compile and link with -pthread.
```

- (*start_routine) is a functin pointer to a function that returns a void * and has one argument of type void *
- pthread_t is a unsigned long (%lu)

Based on slides from Luiz F. Perrone

Thread Termination

- If any thread within a process calls exit, then the entire **process** terminates
- A thread can exit in three ways
  1. Return from the start routine. The return value is the thread's exit code
  2. The thread can be canceled by another thread in the same process
  3. The thread can call pthread_exit

NAME

pthread_exit – terminate calling thread

void pthread_exit (void *retval);

Based on slides from Luiz F. Perrone

Thread Return Value

- If a thread has return values from its start routine, it can send it to other threads in the process by calling pthread_exit (void* retval) or simply returning a pointer of type void * to the return value
- retval is a type-less pointer like the input argument for pthread_create
- How can other threads access this value?
  - If a thread needs an input argument from another thread it can use

NAME

pthread_join – calling thread will block until the specific thread calls pthread_exit

pthread_join (pthread_t tid, void **retval_ptr);

retval_ptr has the return value of the thread with ID tid

FYR: Thread Return Value (continued)

- The typeless pointer passed to pthread_create and pthread_exit can be used to pass more than a single value. The pointer can be used to pass the address of a structure containing more complex information.
- Be careful that the memory used for the structure is still valid when the caller has completed.
  - If the input structure was allocated on the caller's stack, for example, the memory contents might have changed by the time the structure is used.
  - If a thread allocates an output structure on its stack and passes a pointer to this structure to pthread_exit, then the stack might be destroyed and its memory reused for something else by the time the caller of pthread_join tries to use it.

Other Thread Operations

NAME
pthread_self – returns the thread ID
pthread_t pthread_self (void);

NAME
pthread_equal – compare thread IDs
pthread_equal (pthread_t tid1, pthread_t tid2);

- Just as every process has a process ID, every thread has a thread ID. Unlike the process ID, which is unique in the system, the thread ID has significance only within the context of the process to which it belongs.
- pthread_t data type can be implemented as a structure. Therefore, a function must be used to compare two thread IDs.

Example

```
/* COMPILE WITH: gcc thread-ex.c -lpthread -o thread-ex */
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADS 5
#define SLEEP_TIME 3

void *sleeping(void *); /* forward declaration to thread routine */

int main(int argc, char *argv[]) {
int i;
pthread_t tid[NUM_THREADS]; /* array of thread IDs */
for ( i = 0; i < NUM_THREADS; i++)
  pthread_create(&tid[i], NULL, sleeping,(void *)SLEEP_TIME);

for ( i = 0; i < NUM_THREADS; i++)
  pthread_join(tid[i], NULL);

printf("main() reporting that all %d threads have terminated\n", i);
return (0);
} /* main */
```

Passing arguments to threads

```
void * sleeping(void *arg) {
  int sleep_time = (int)arg;
  printf("thread %ld sleeping %d seconds ...\n",
  pthread_self(), sleep_time);
  sleep(sleep_time);
  printf("\nthread %ld awakening\n", pthread_self());
  return (NULL);
}
```

- A thread can take parameter(s) pointed by its **arg** and
- can return a pointer to some memory location that stores its results. Gotta be careful with these pointers!!!

Redo the Activity!

Example

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

int main(){
    int * a = (int *)malloc(sizeof(int));
    void * (*func)(void *) = &sum;

    *a = 100;
    printf("function pointer %d\n", (*func)((void *)a));
    pthread_t id;
    int err;
    err = pthread_create(&id, NULL, func,(void *)a);
    printf("thread create ret val %d\n",err);
    pthread_join(id, NULL);
return 0;
}
```

Passing arguments to threads

```
void * sum(void * a){
    int * val = (int *)a;
    int result = 0;
    for (int i=0; i <= *val;i++){
        result += i;
    }
    printf("Running thread %lu value in thread %d \n",pthread_self(),result);
    return (void *) &result;
}
```

- A thread can take parameter(s) pointed by its **arg** and
- can return a pointer to some memory location that stores its results. Gotta be careful with these pointers!!!