



CS 1550

Week 4 – Project 1 Discussion

Teaching Assistant

Henrique Potter

Recitation TA – Office Hours

- Office Hours (SENSQ 6507)
 - Tuesday:
 - 10:00 am to 11:00 am / 6:00 pm to 7:00 pm
 - Wednesday:
 - 10:00 am to 12:00 pm
 - Friday:
 - 10:00 am to 12:00 pm
- Email
 - potter.hp@pitt.edu
- Slides Website
 - <http://people.cs.pitt.edu/~henriquepotter/>

CS 1550 – Course Servers

Local host

Remote Host

CS 1550 – Course Servers

Local host



Laptop
PC
Notebook
MacBook

Remote Host

CS 1550 – Course Servers

Local host



Laptop
PC
Notebook
MacBook

Remote Host

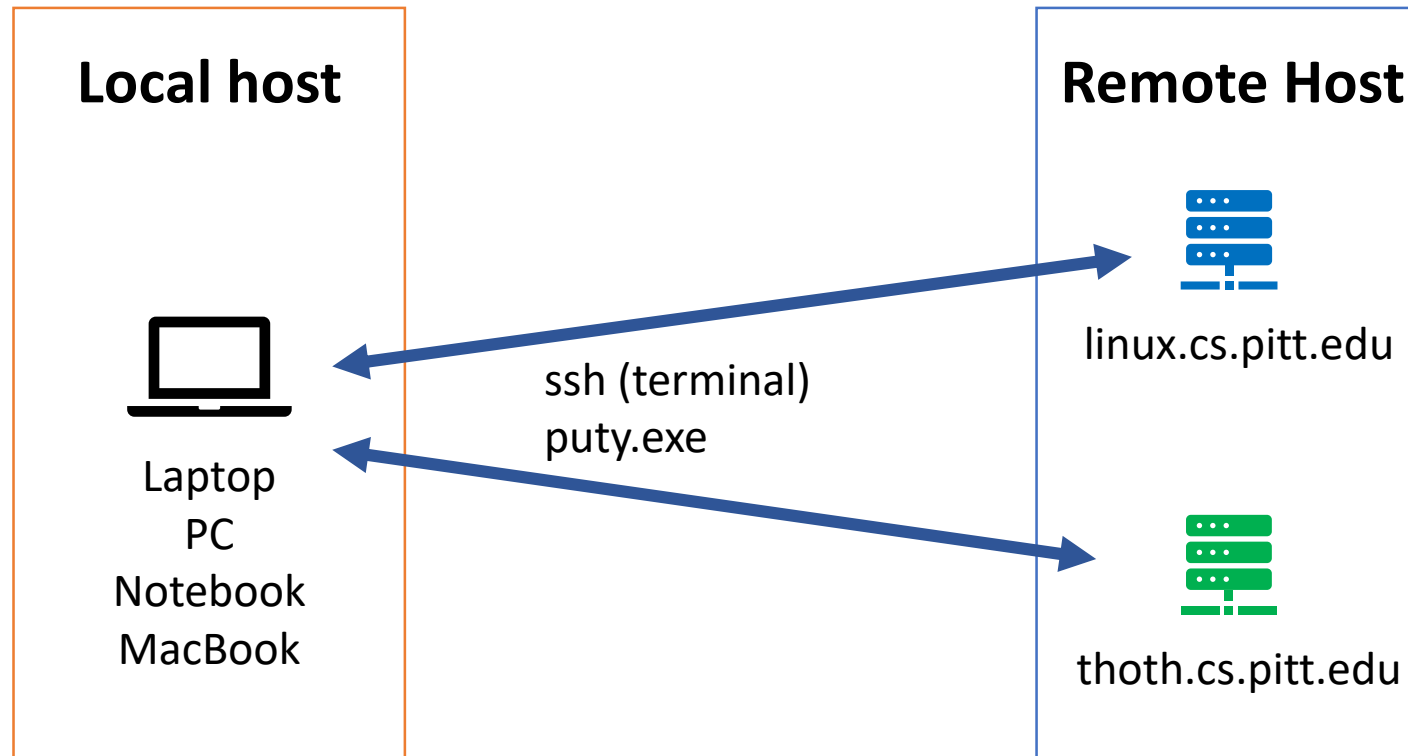


linux.cs.pitt.edu

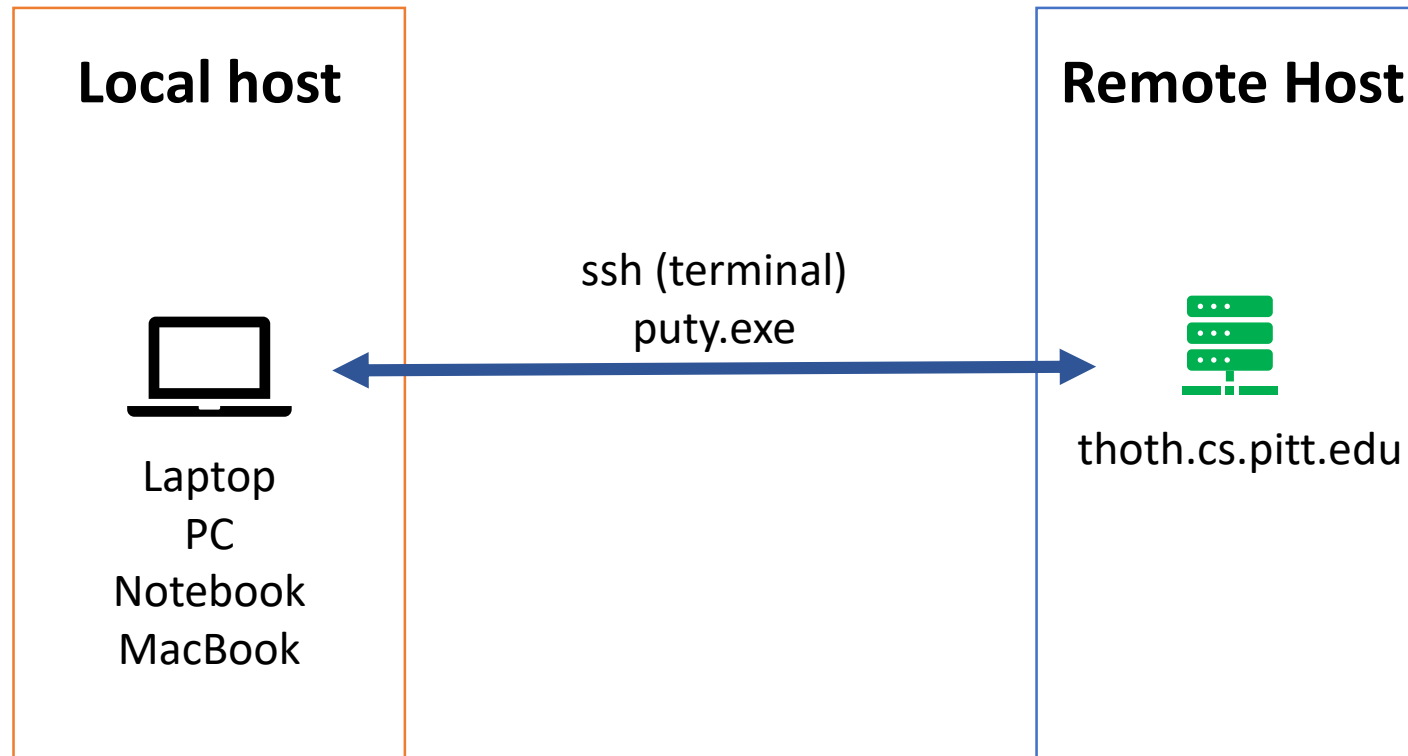


thoth.cs.pitt.edu

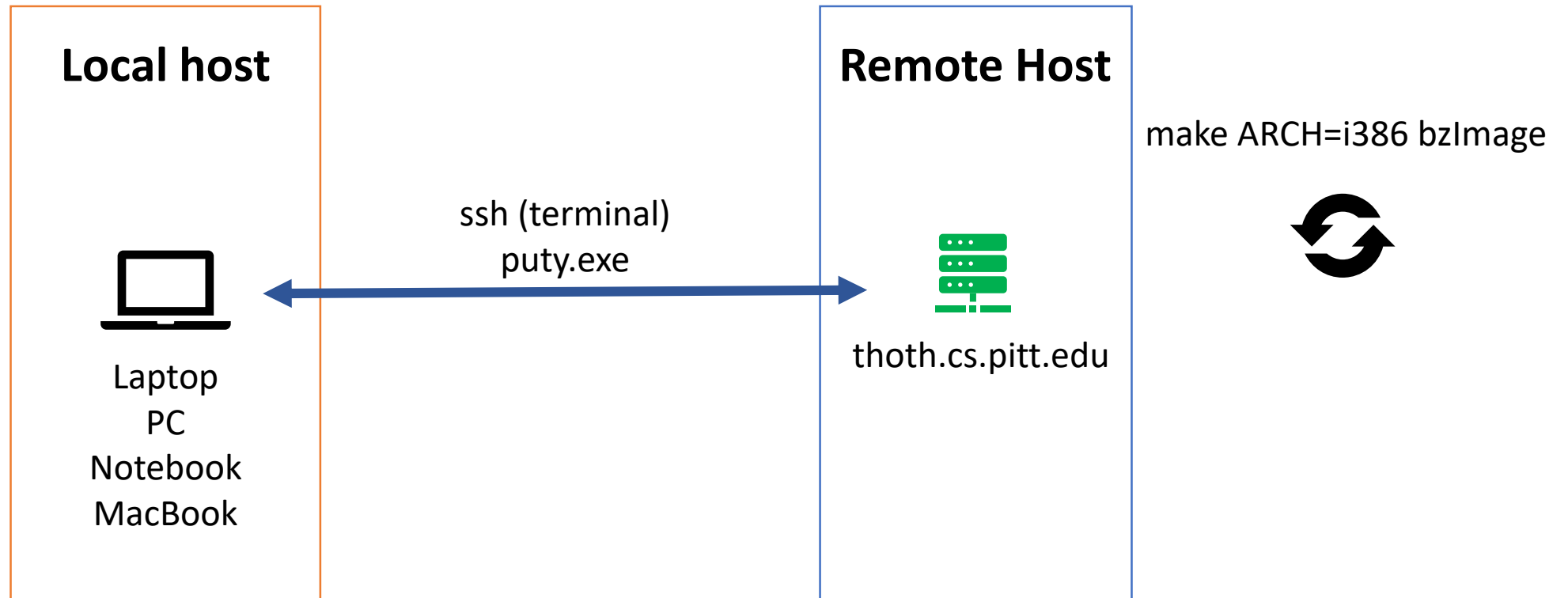
CS 1550 – Course Servers



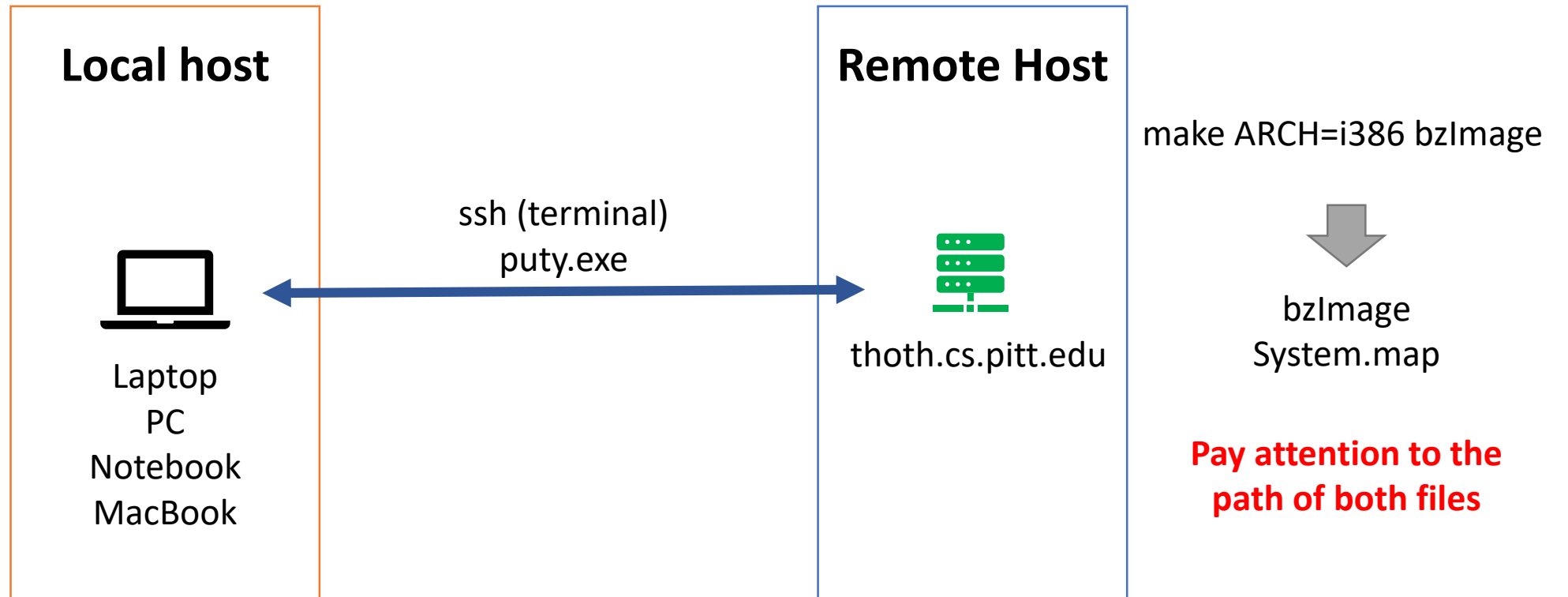
CS 1550 – Project 1 kernel compilation



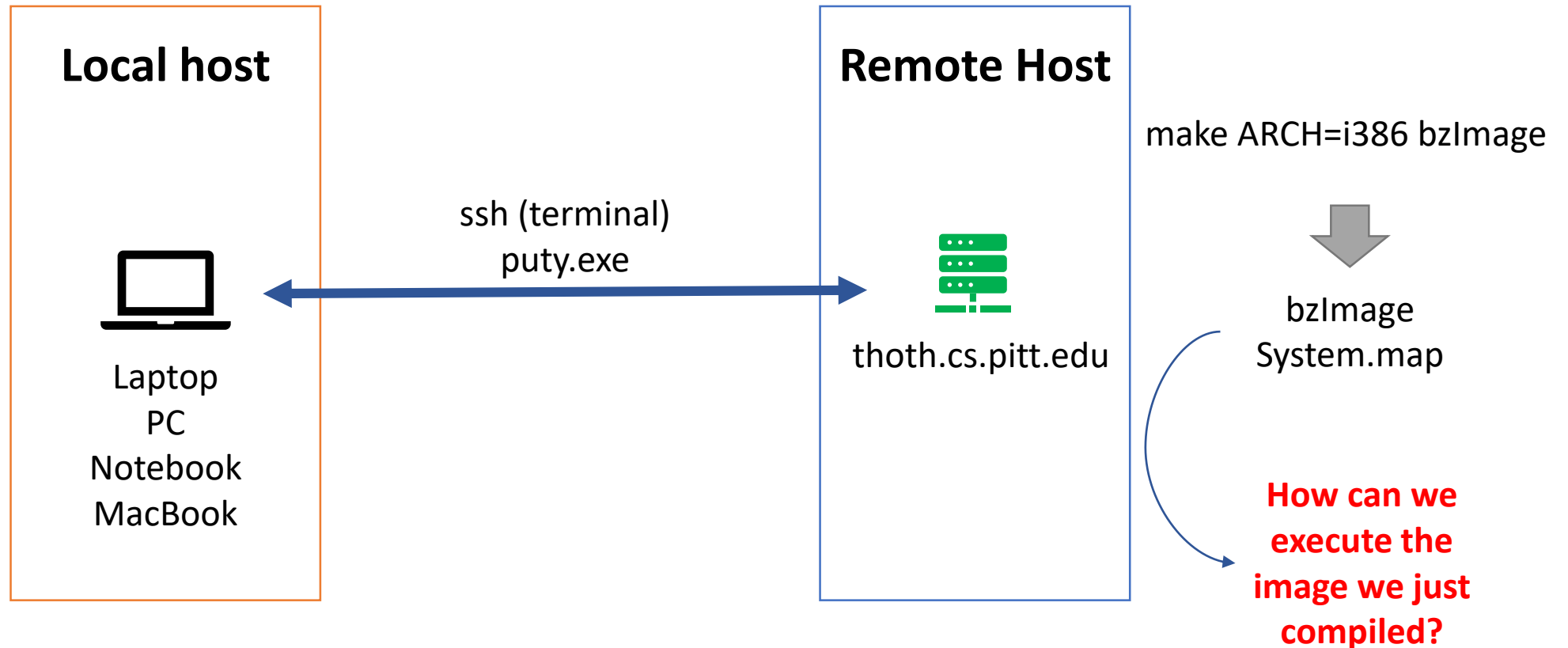
CS 1550 – Project 1 kernel compilation



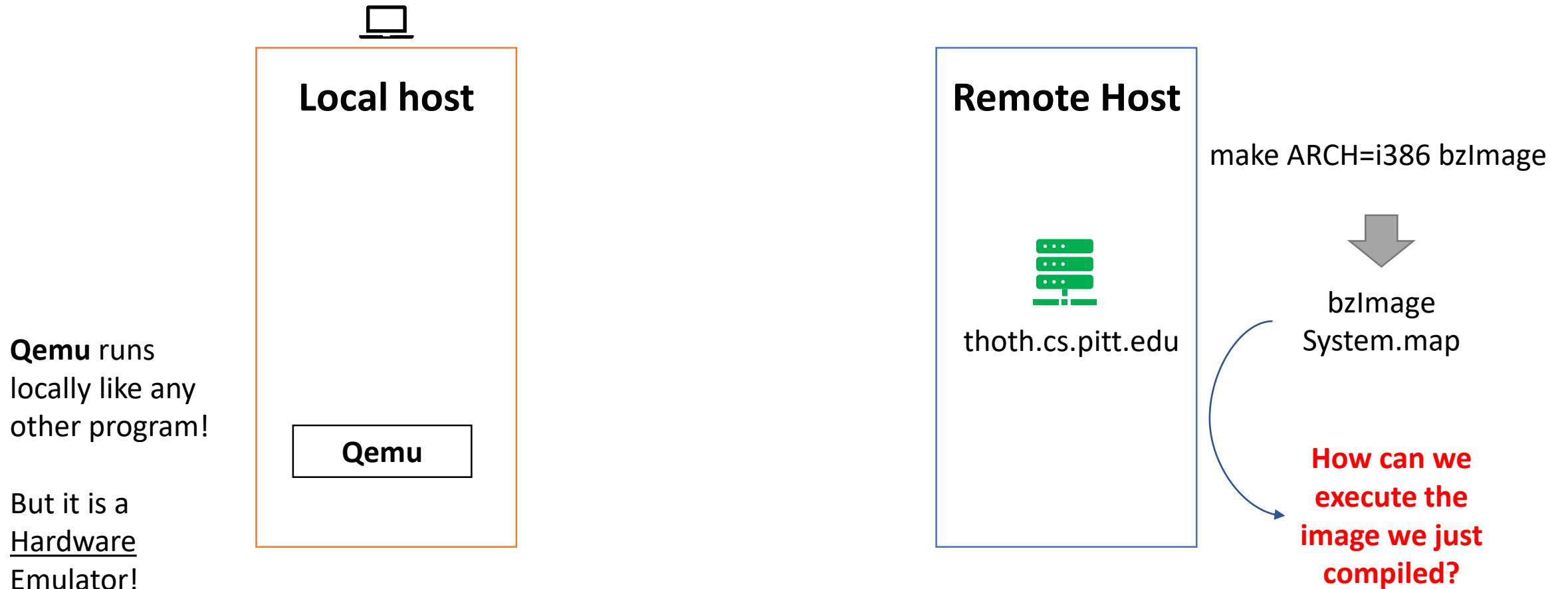
CS 1550 – Project 1 kernel compilation



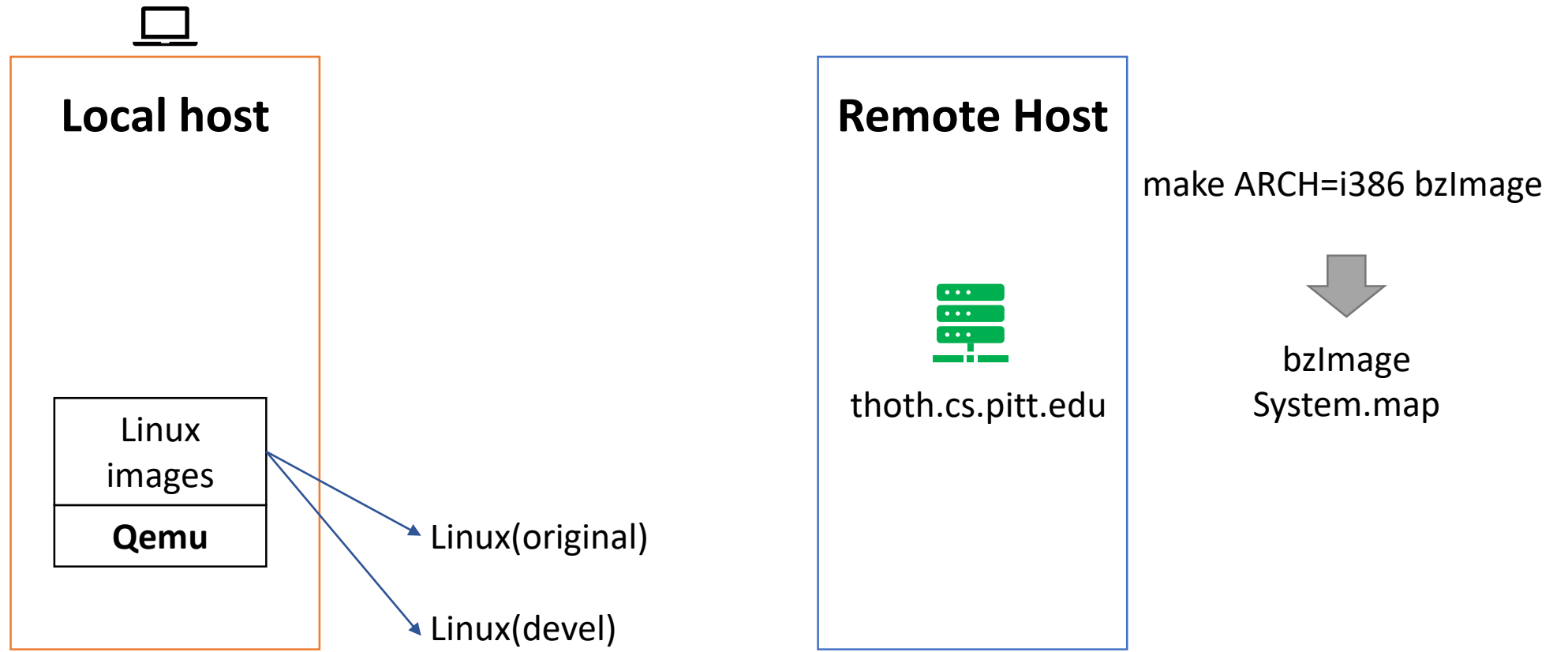
CS 1550 – Project 1 kernel compilation



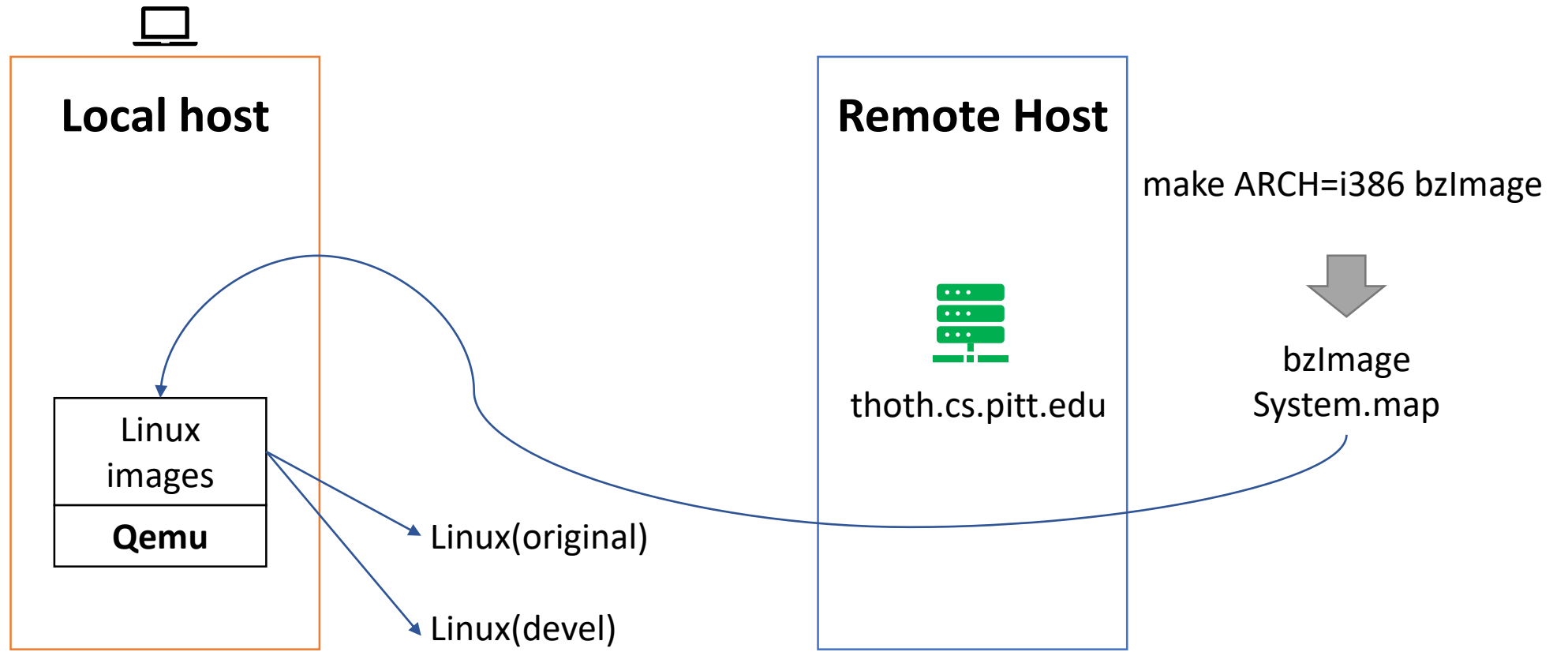
CS 1550 – Project 1 kernel compilation



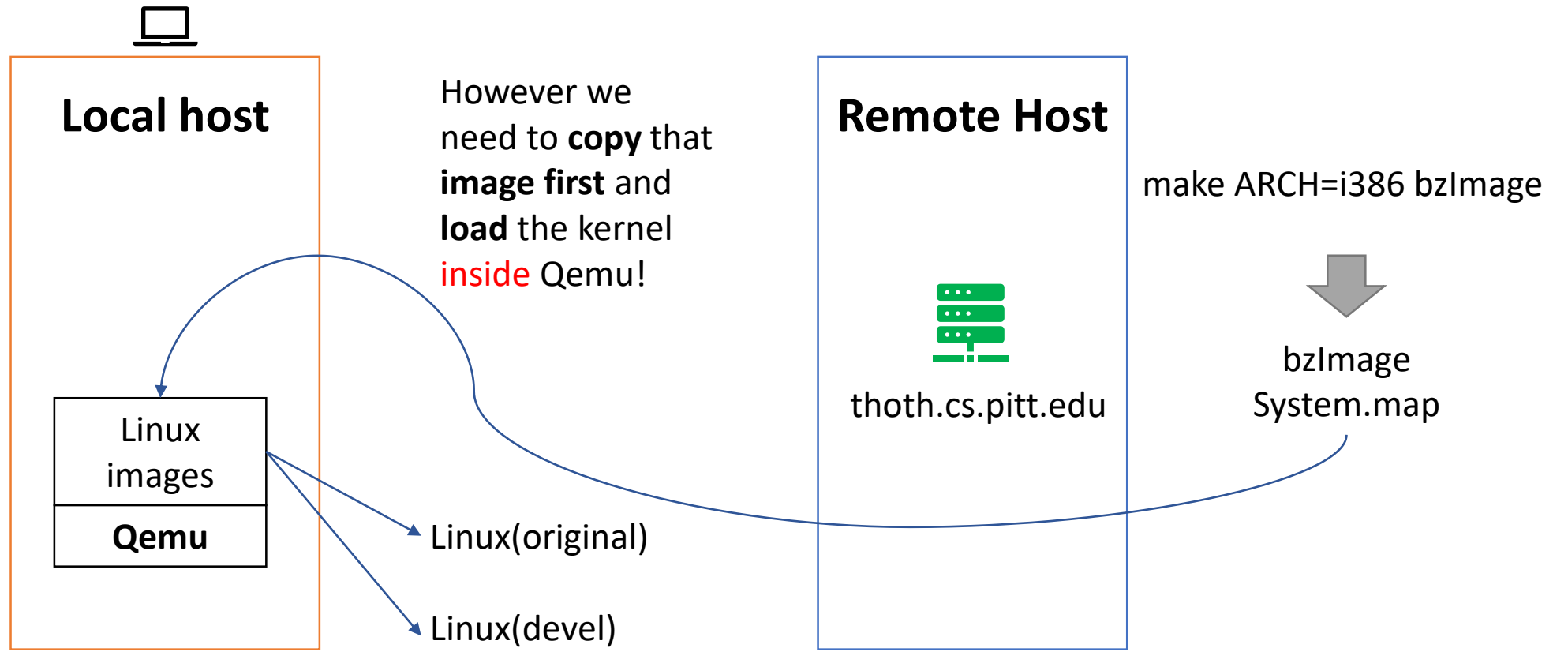
CS 1550 – Project 1 kernel compilation



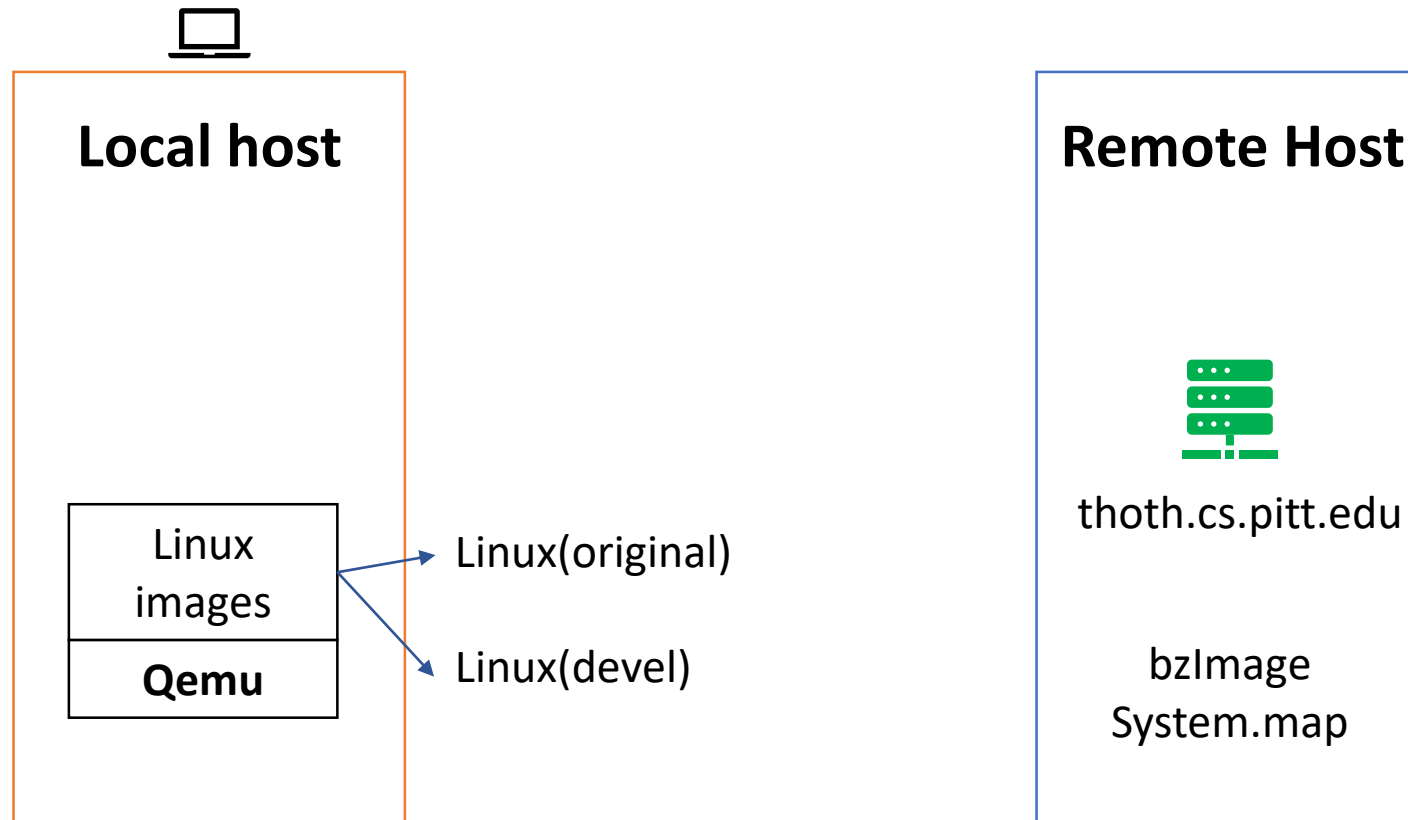
CS 1550 – Project 1 kernel compilation



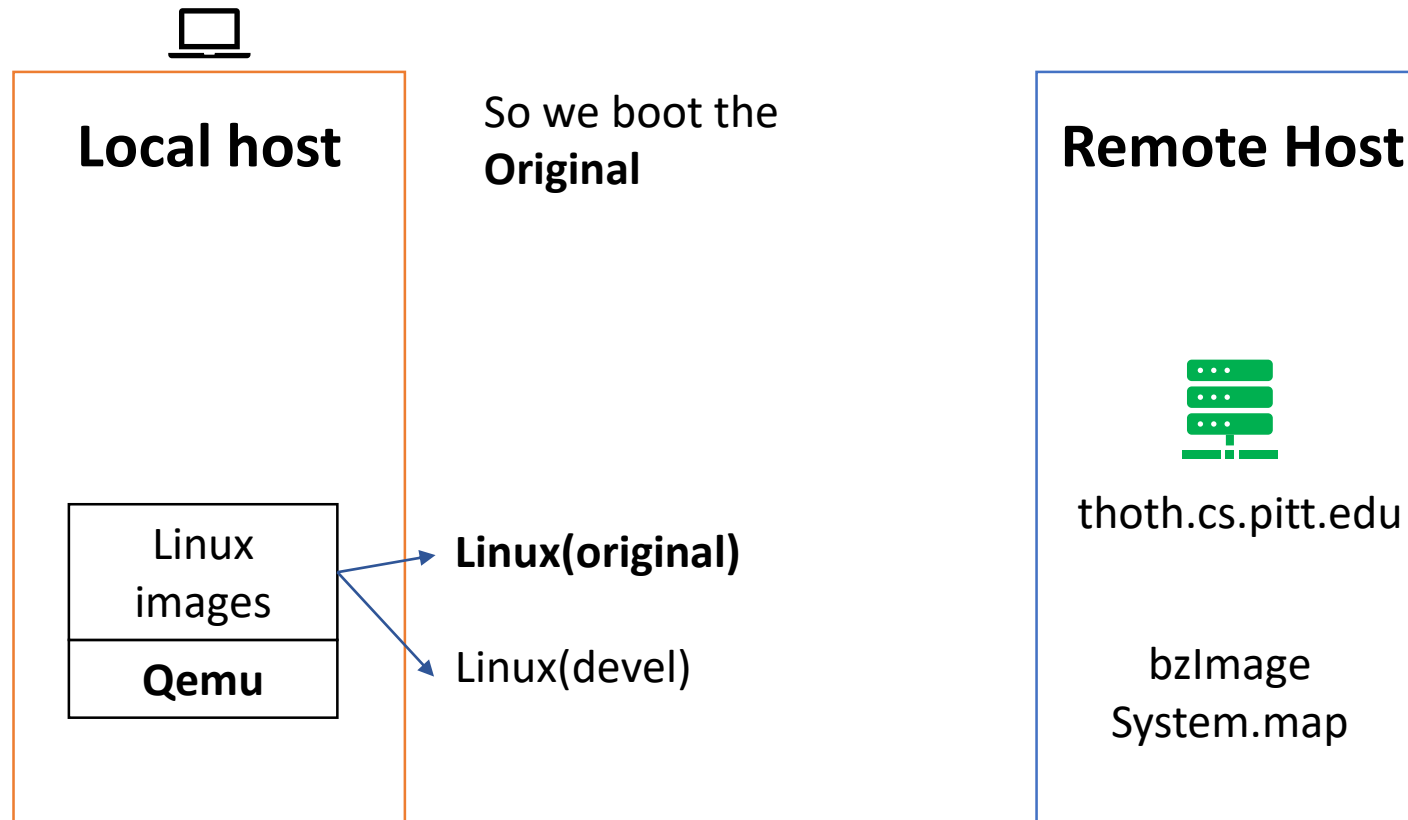
CS 1550 – Project 1 kernel compilation



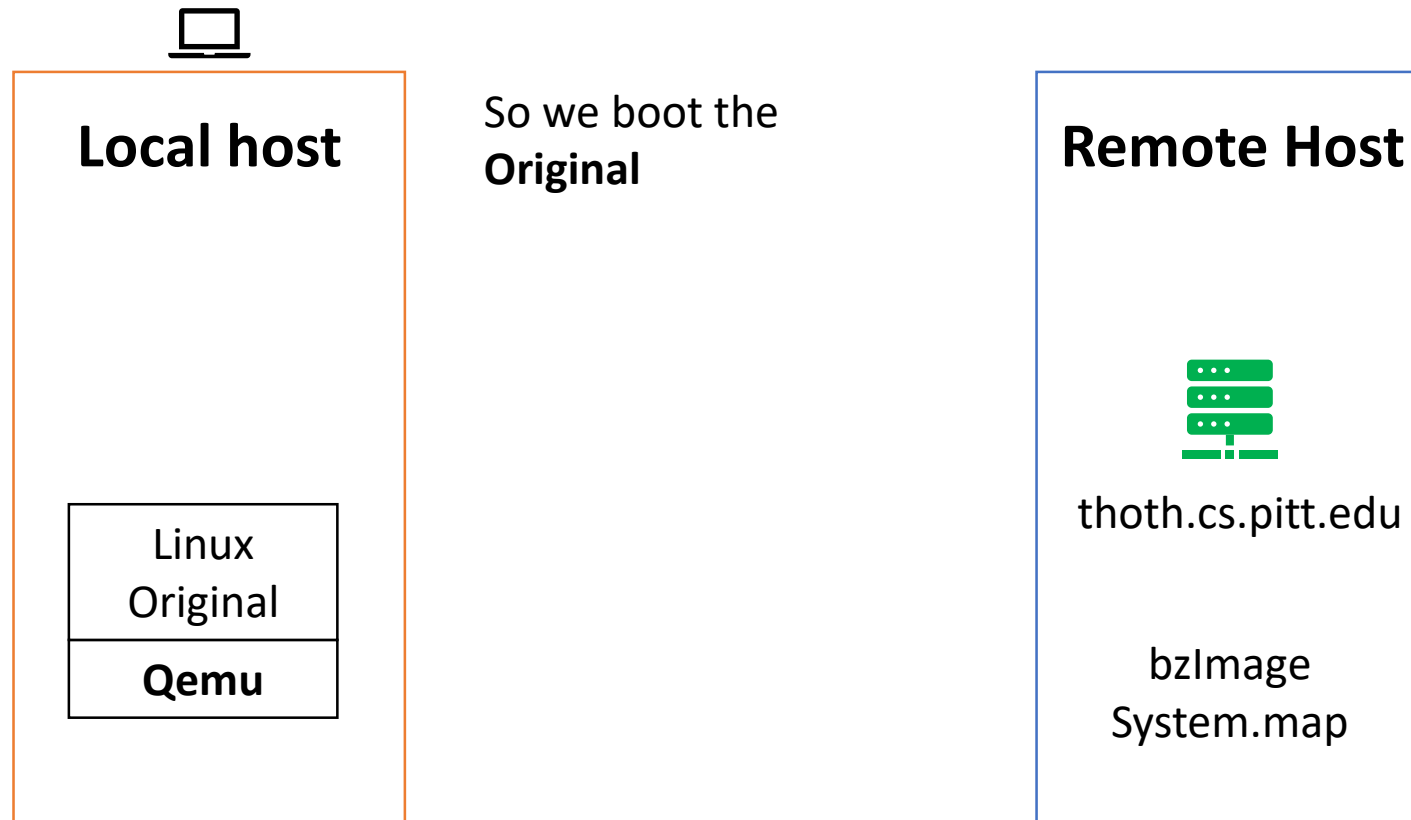
CS 1550 – Project 1 kernel compilation



CS 1550 – Project 1 kernel compilation

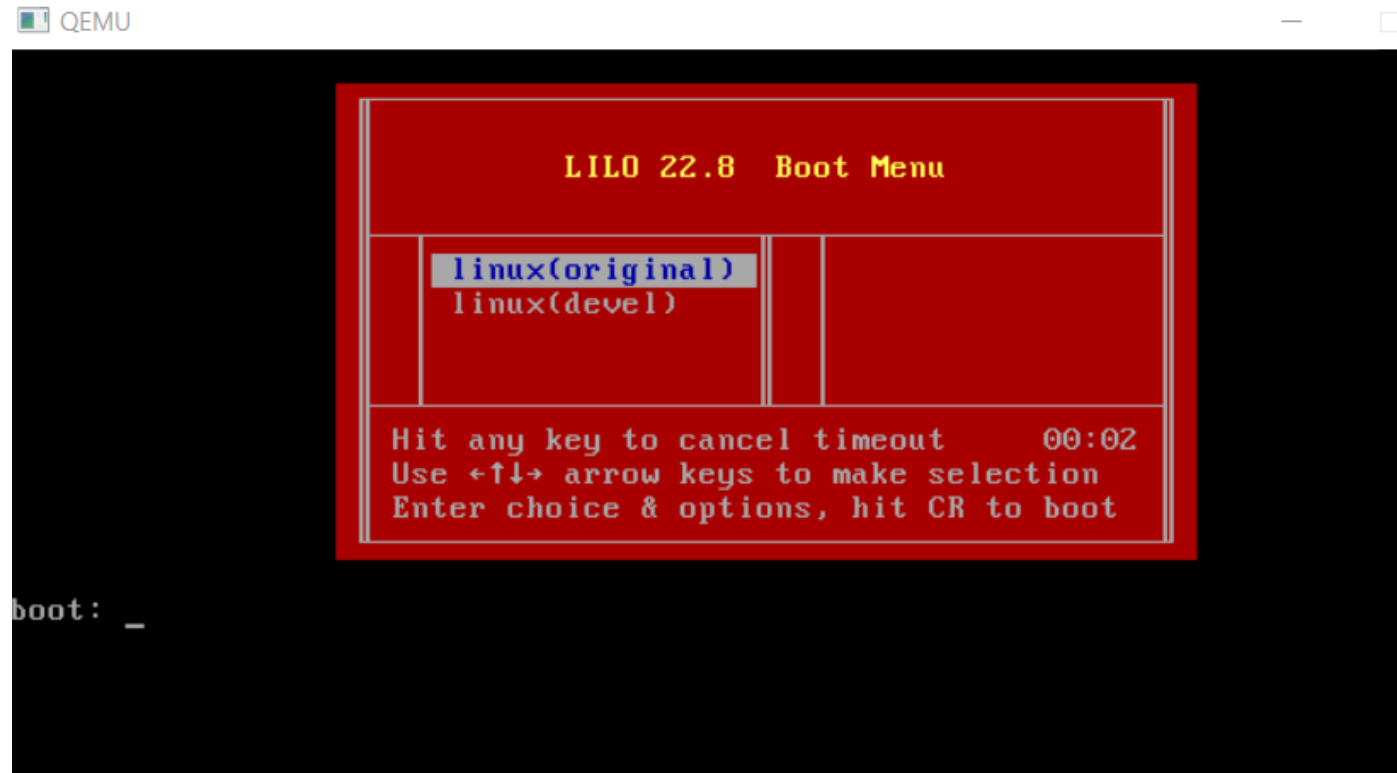


CS 1550 – Project 1 kernel compilation

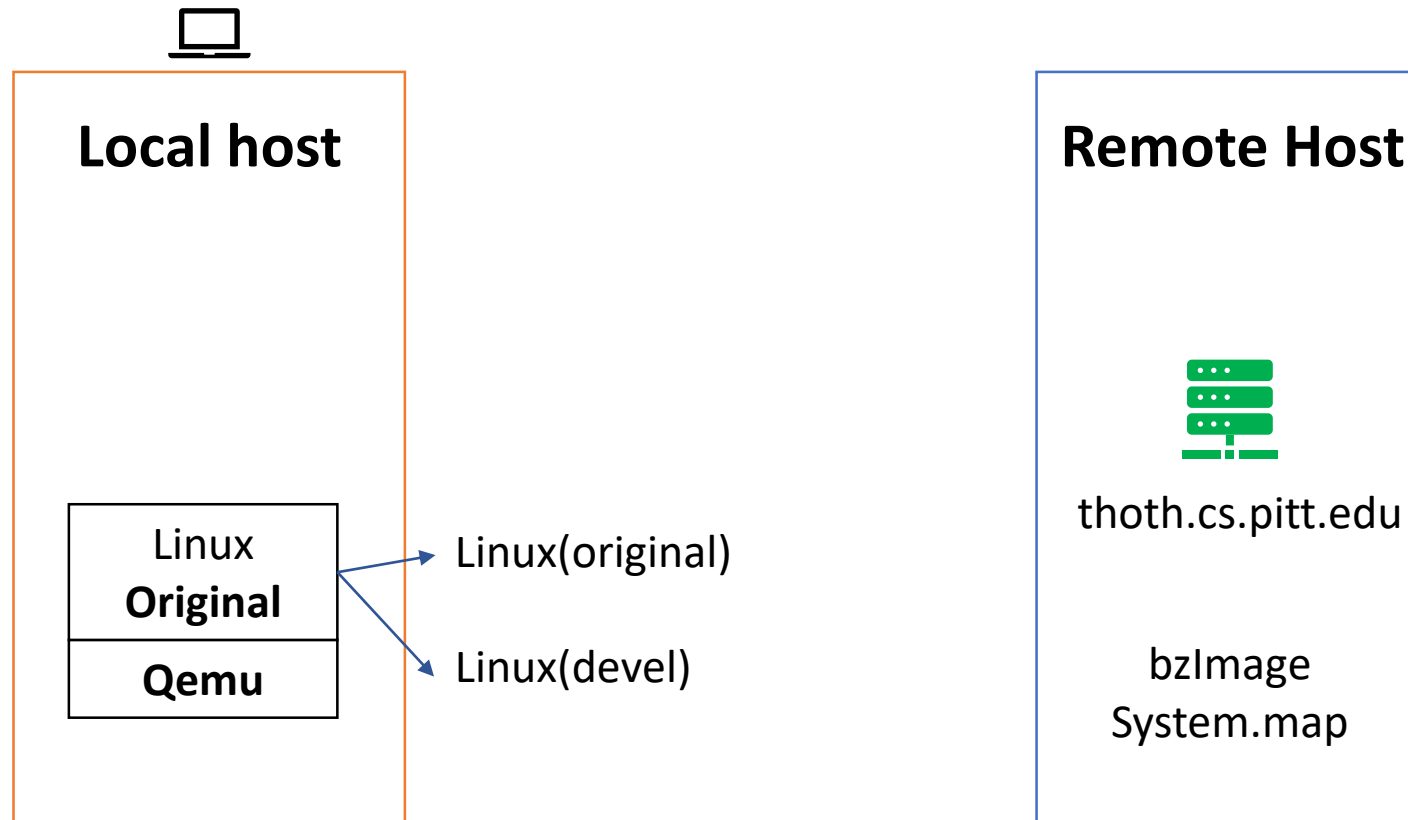


CS 1550 – Project 1 kernel compilation

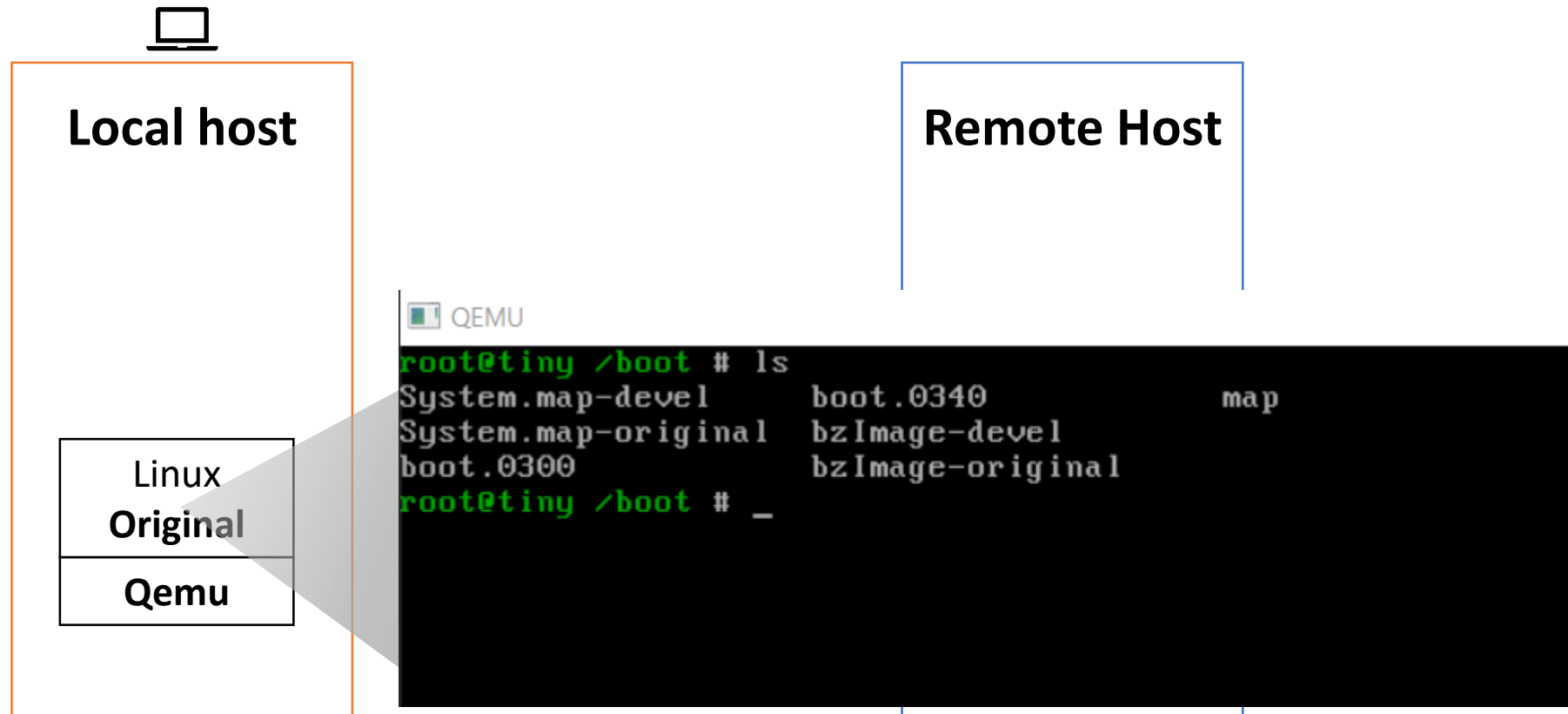
- Choose Linux(original)
 - User '**root**' as **user** and **password**



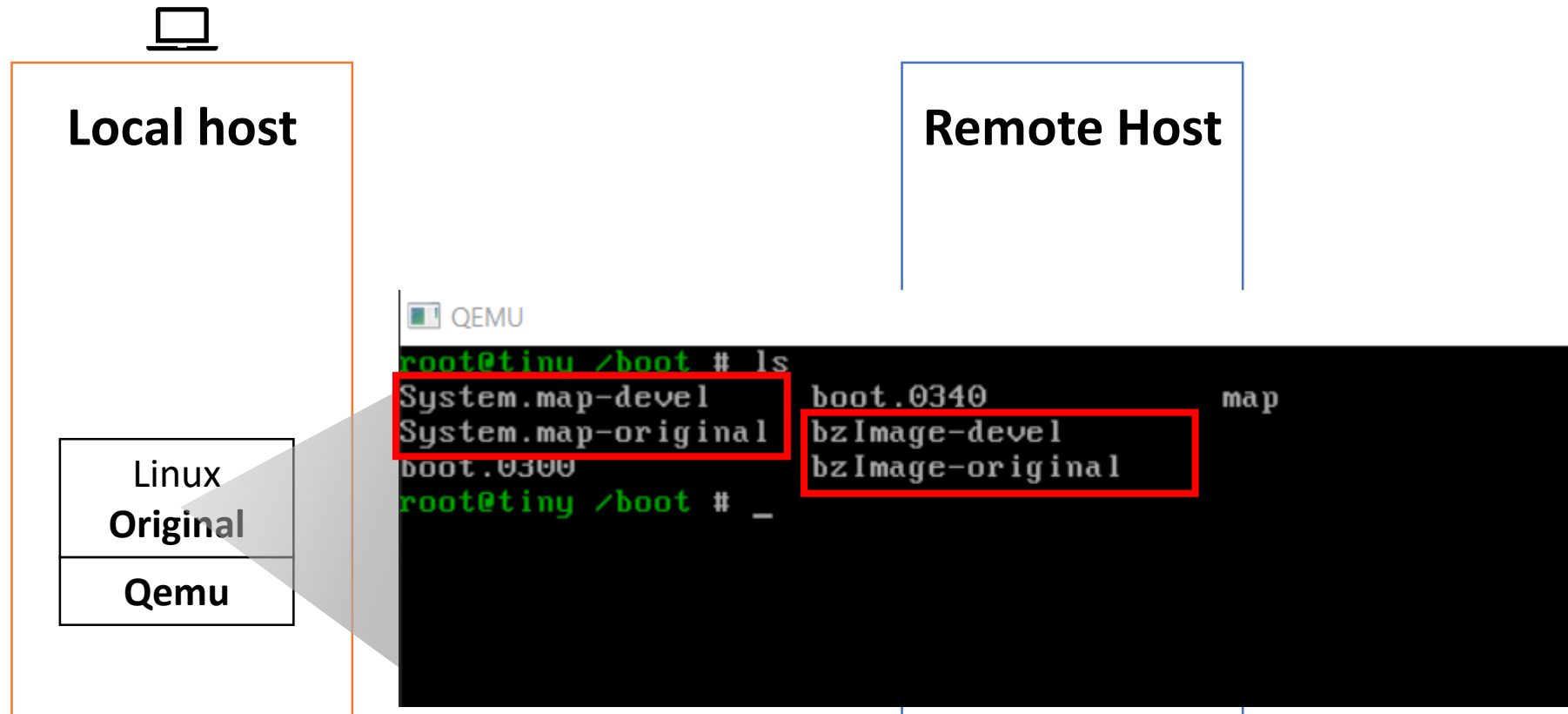
CS 1550 – Project 1 kernel compilation



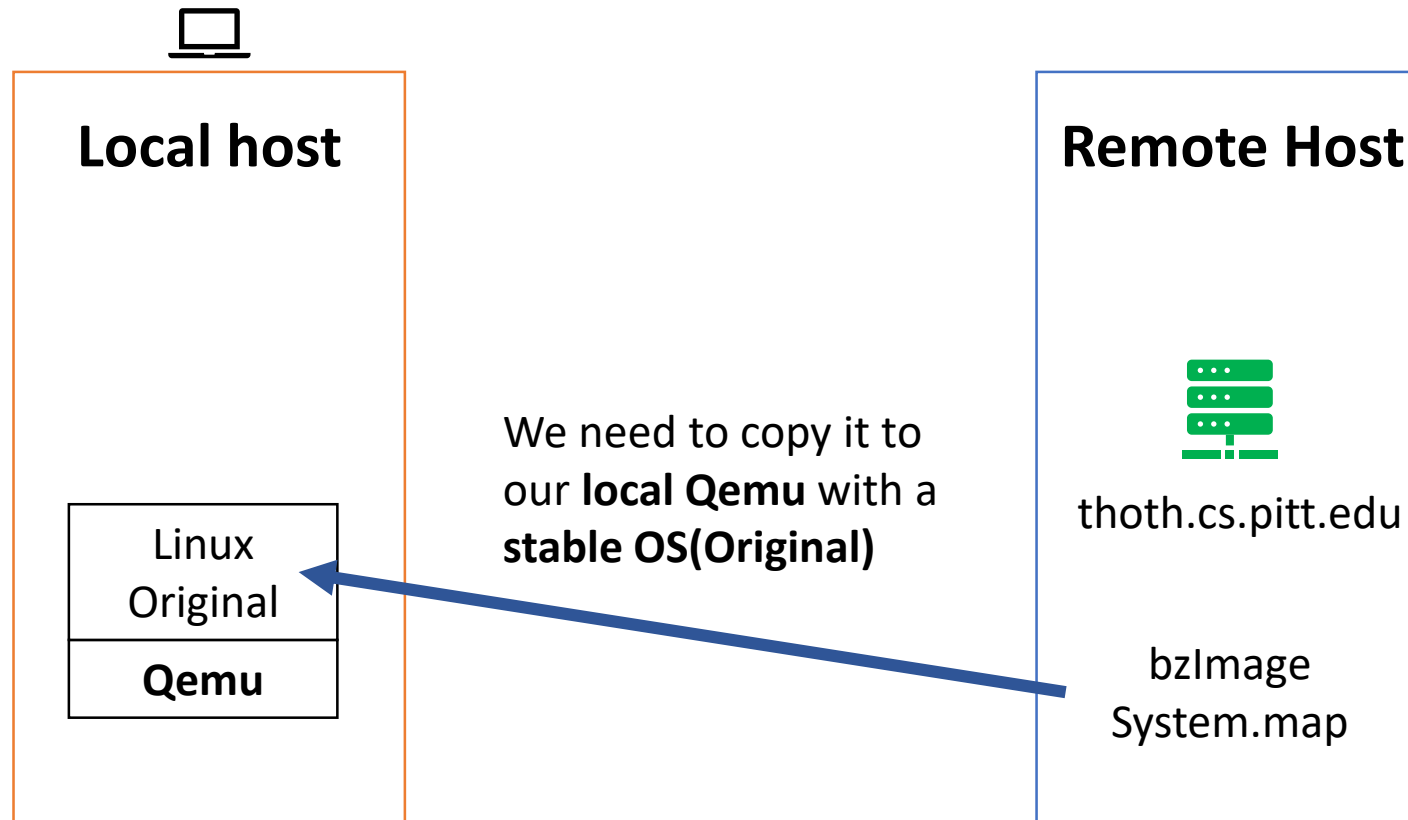
CS 1550 – Project 1 kernel compilation



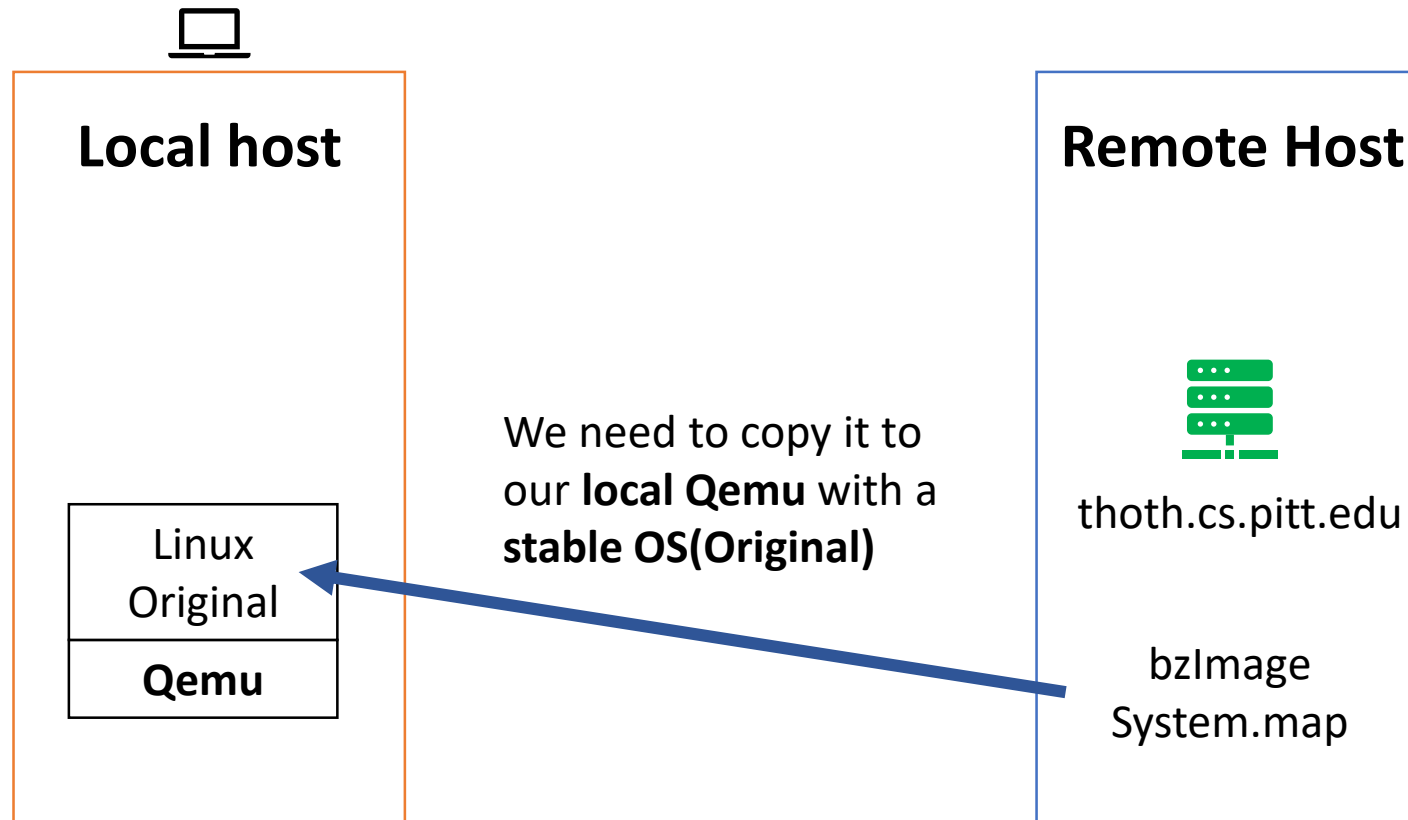
CS 1550 – Project 1 kernel compilation



CS 1550 – Project 1 kernel compilation



CS 1550 – Project 1 kernel compilation



```
scp <user_id>@thoth.cs.pitt.edu:<path_to_the_file>/<file_name> .
```

CS 1550 – Project 1 kernel compilation

scp <user_id>@thoth.cs.pitt.edu:<path_to_the_file>/<file_name> .

CS 1550 – Project 1 kernel compilation



Local host

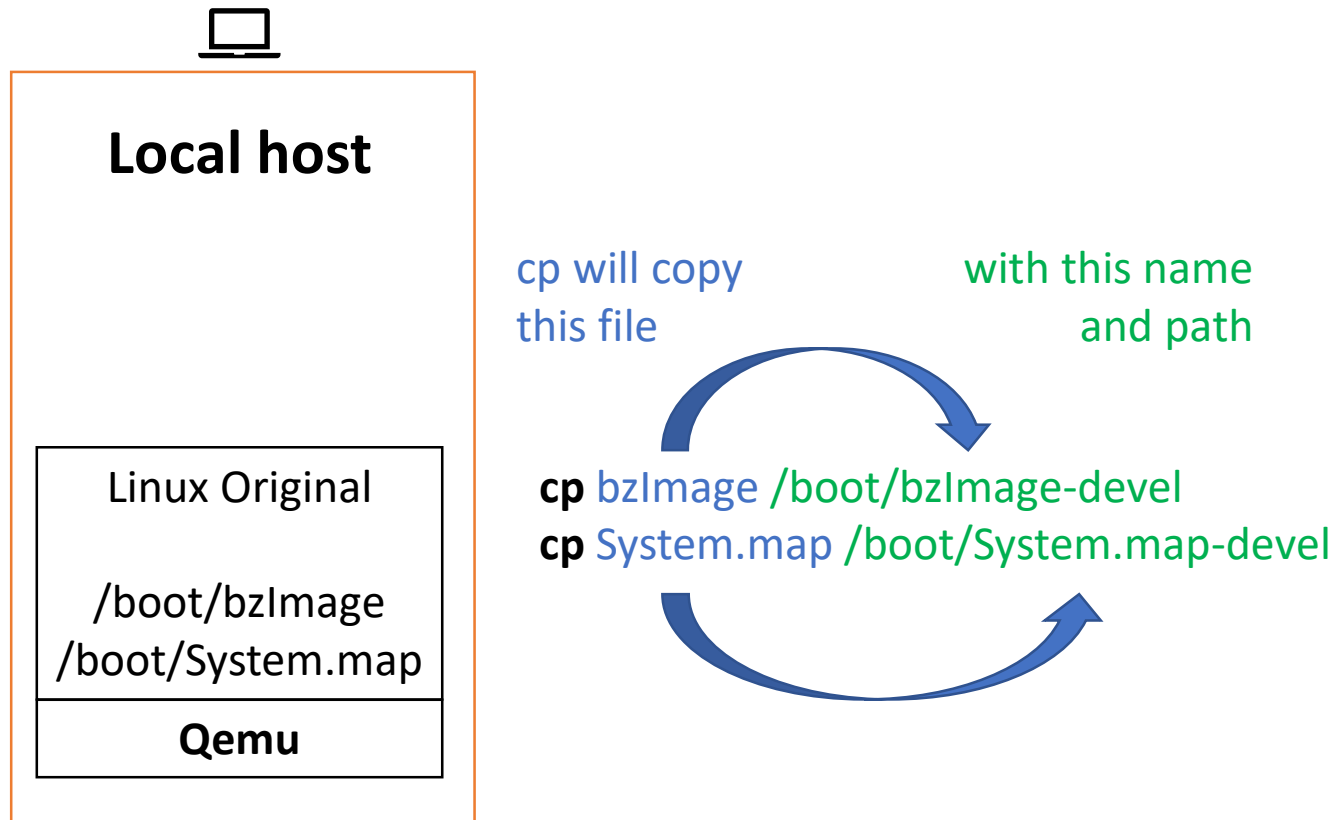
Linux Original

~/bzImage
~/System.map

Qemu

After copying we need to
put it in the **correct folder**
/boot/ and overwrite the
bzImage-devel

CS 1550 – Project 1 kernel compilation



CS 1550 – Project 1 kernel compilation



Local host

Linux Original

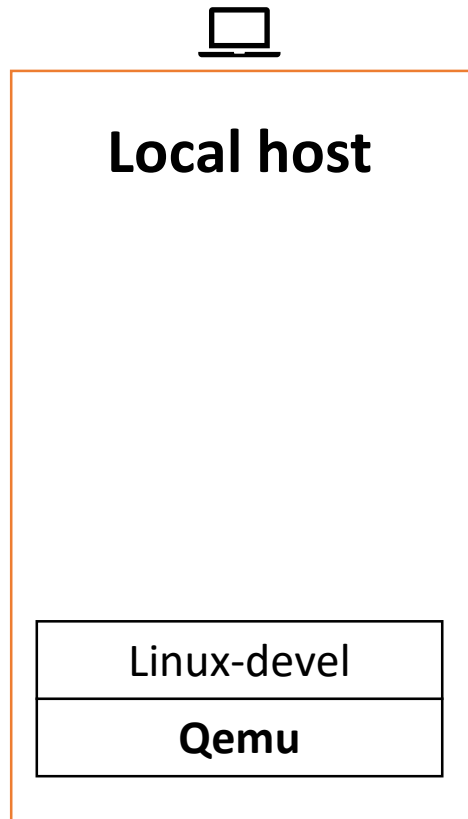
/boot/bzImage
/boot/System.map

Qemu

```
cp bzImage /boot/bzImage-devel  
cp System.map /boot/System.map-devel
```

lilo

CS 1550 – Project 1 kernel compilation

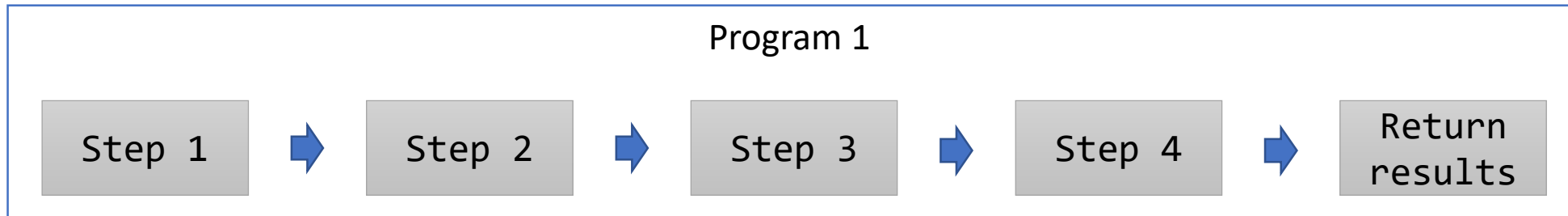


Call **reboot** and choose linux-devel

Synchronization

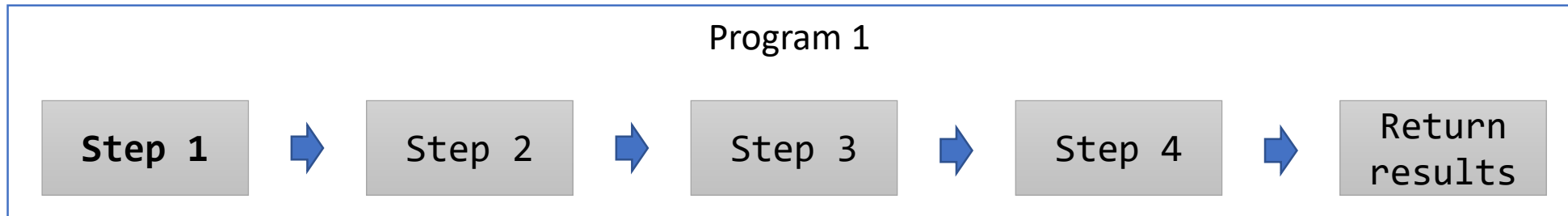
Synchronization

- Each process operates sequentially



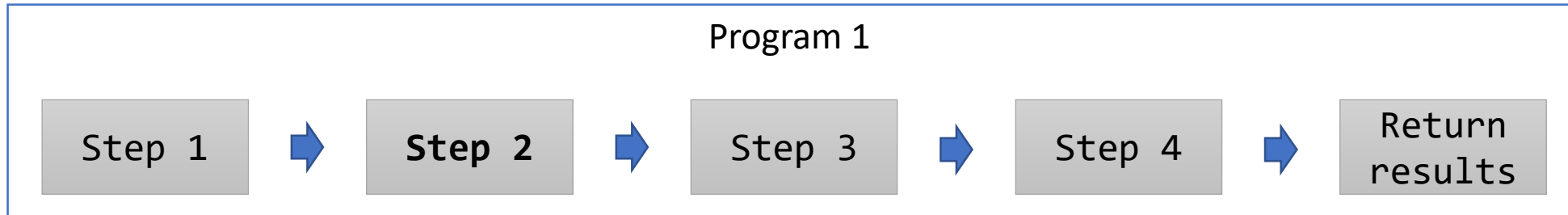
Synchronization

- Each process operates sequentially



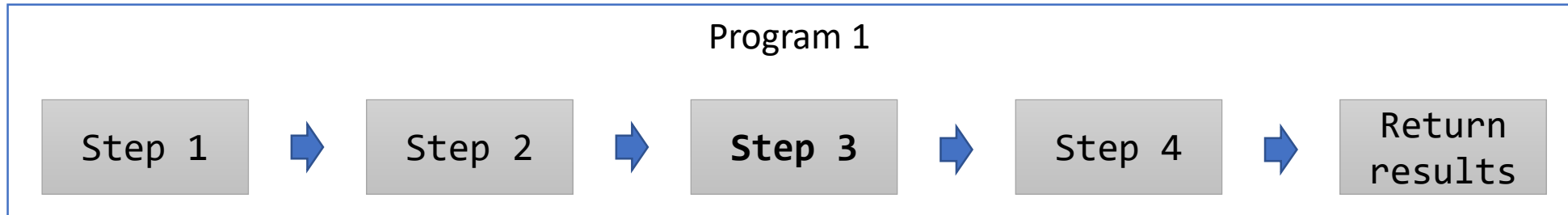
Synchronization

- Each process operates sequentially



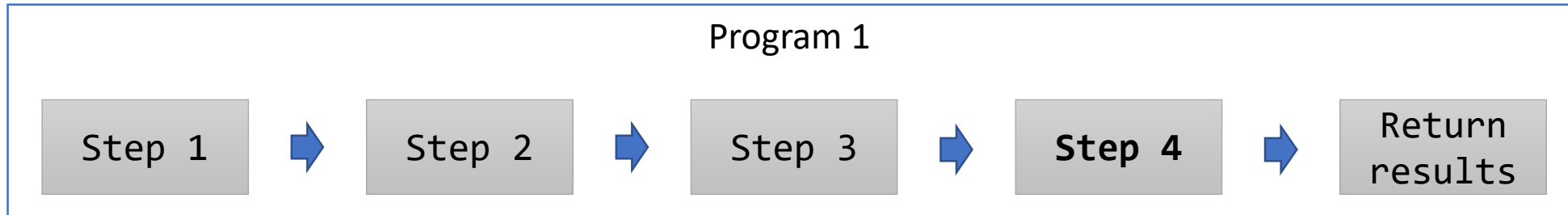
Synchronization

- Each process operates sequentially



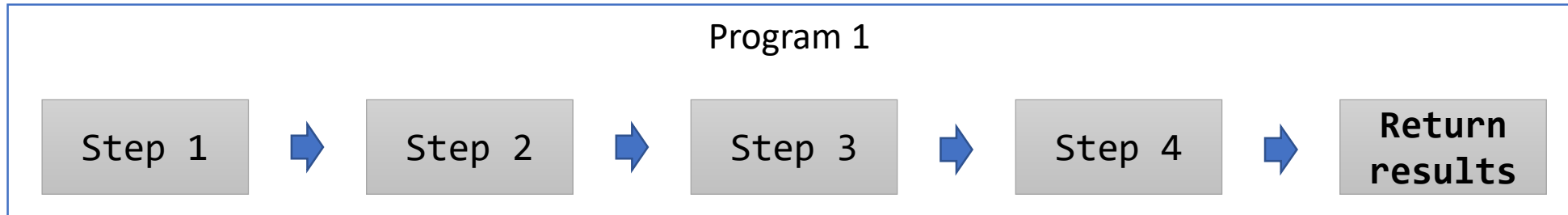
Synchronization

- Each process operates sequentially



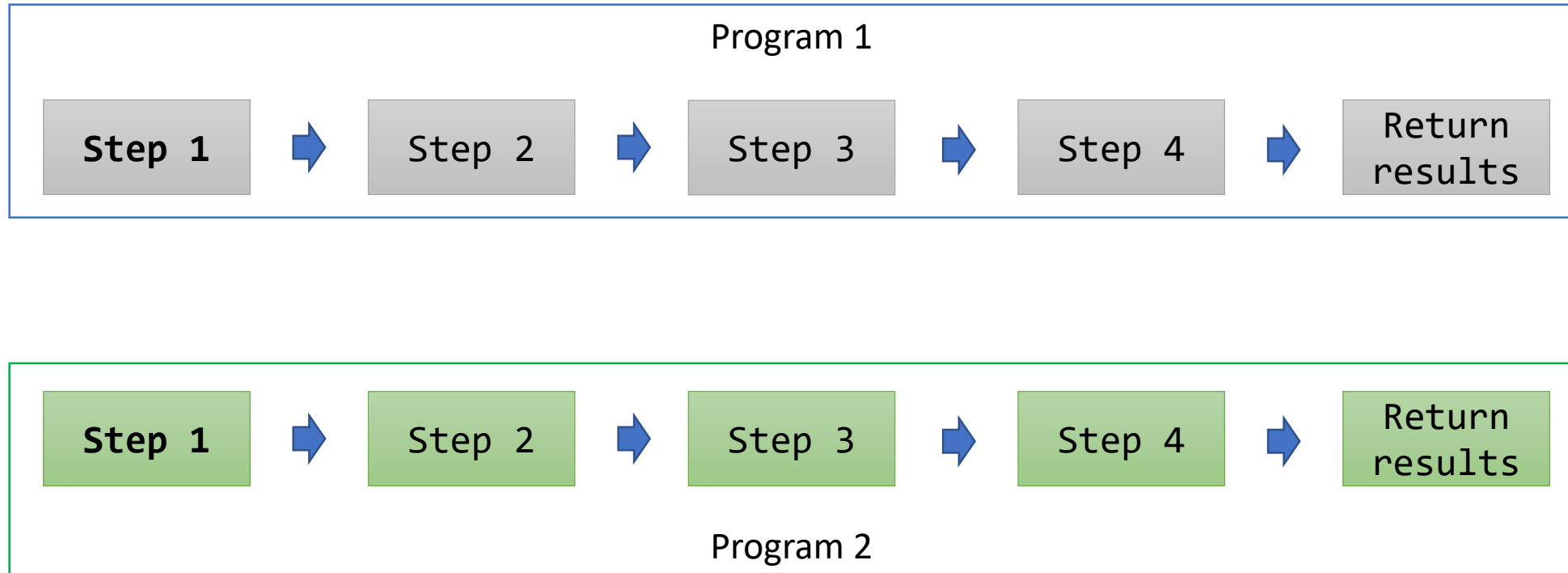
Synchronization

- Each process operates sequentially



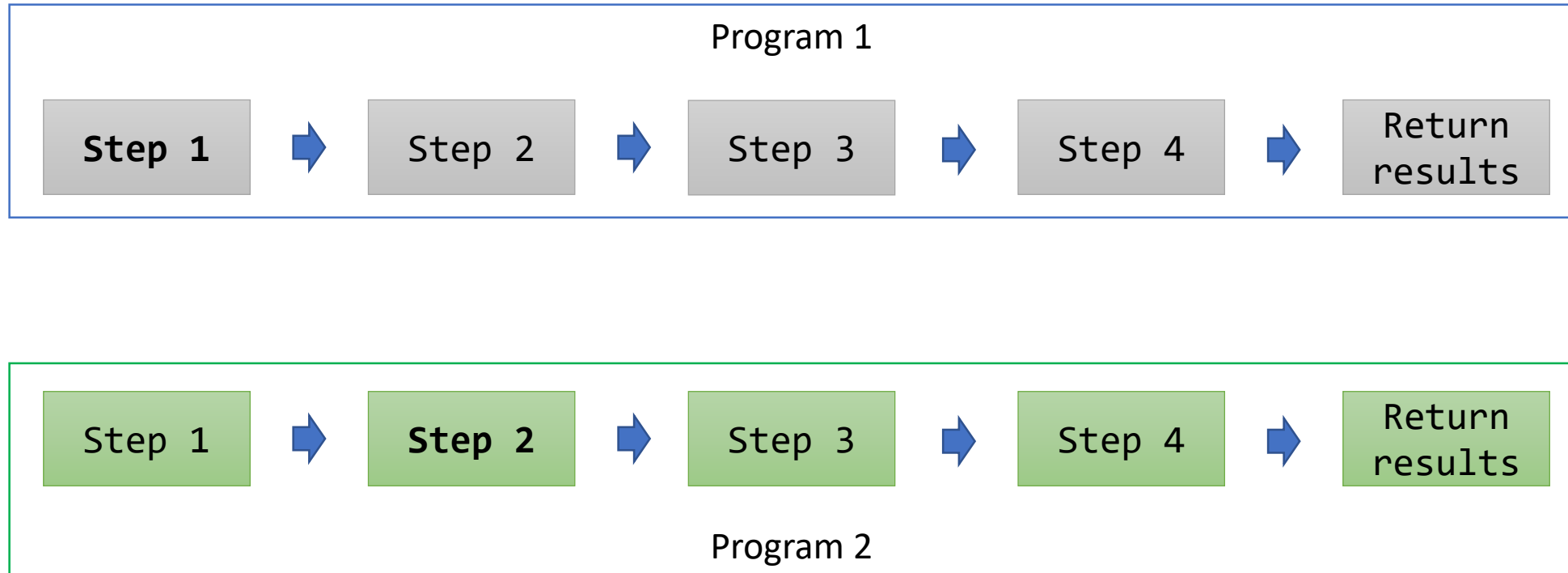
Synchronization

- Each process operates sequentially



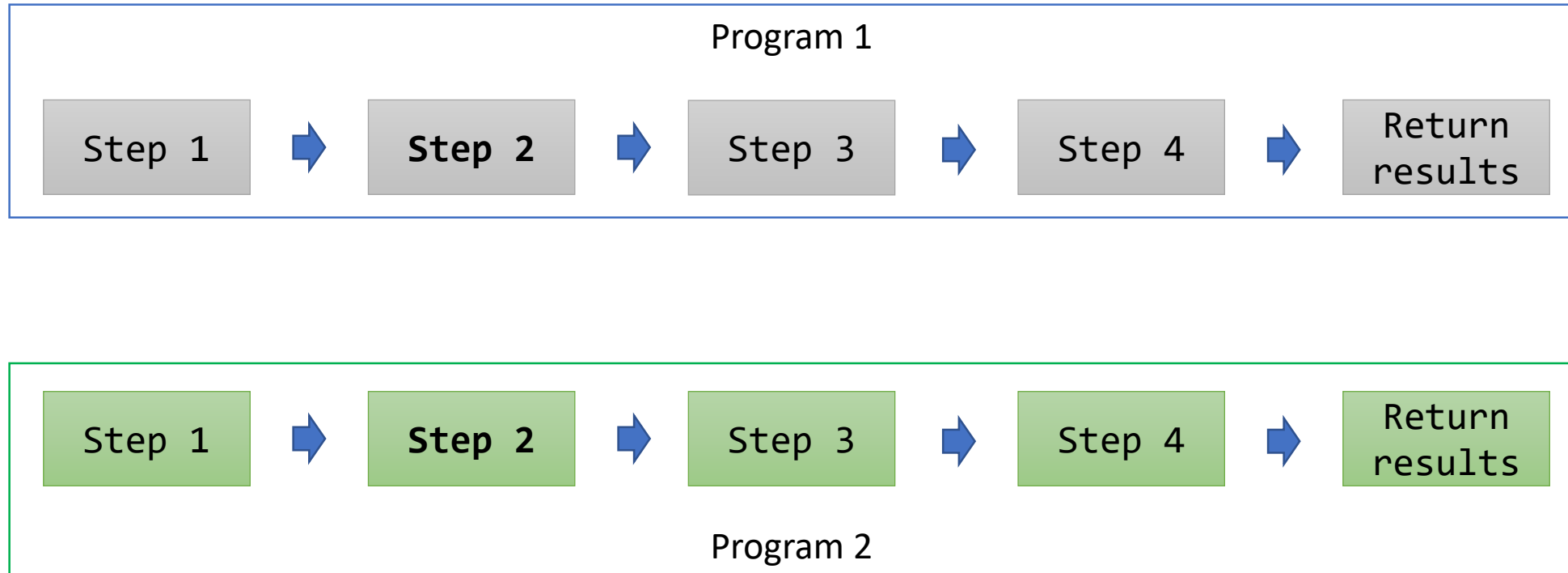
Synchronization

- Each process operates sequentially



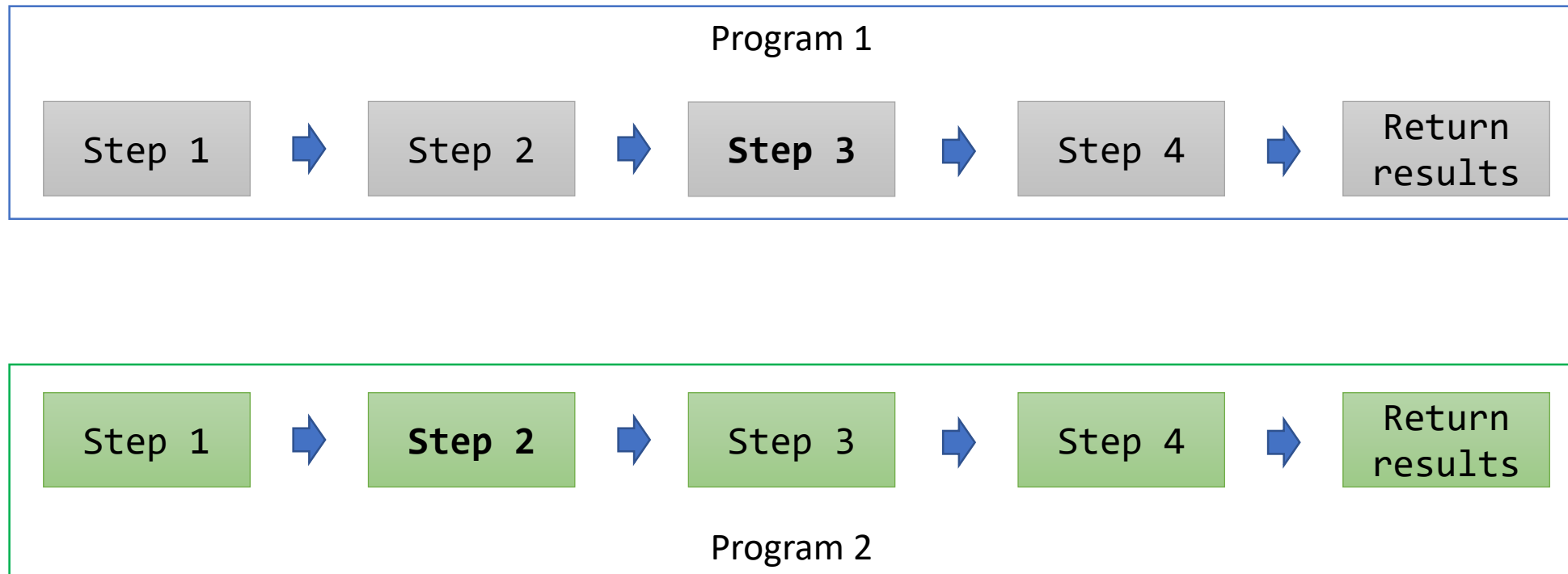
Synchronization

- Each process operates sequentially



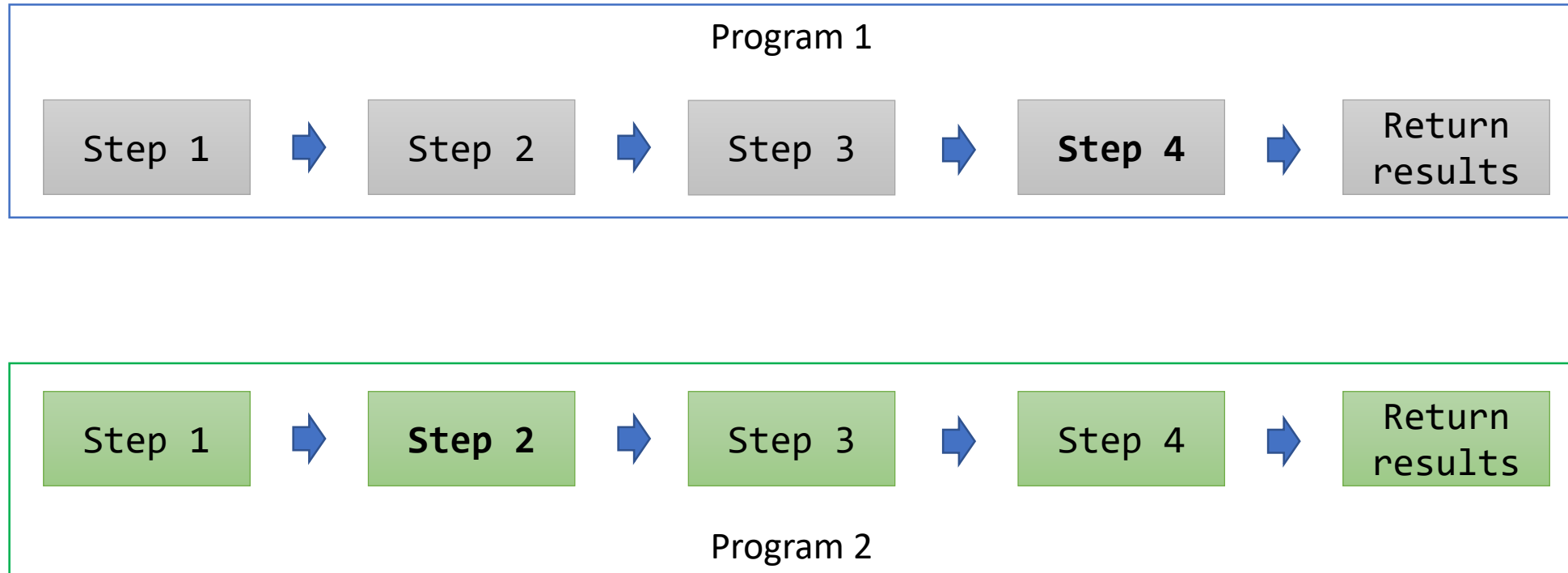
Synchronization

- Each process operates sequentially



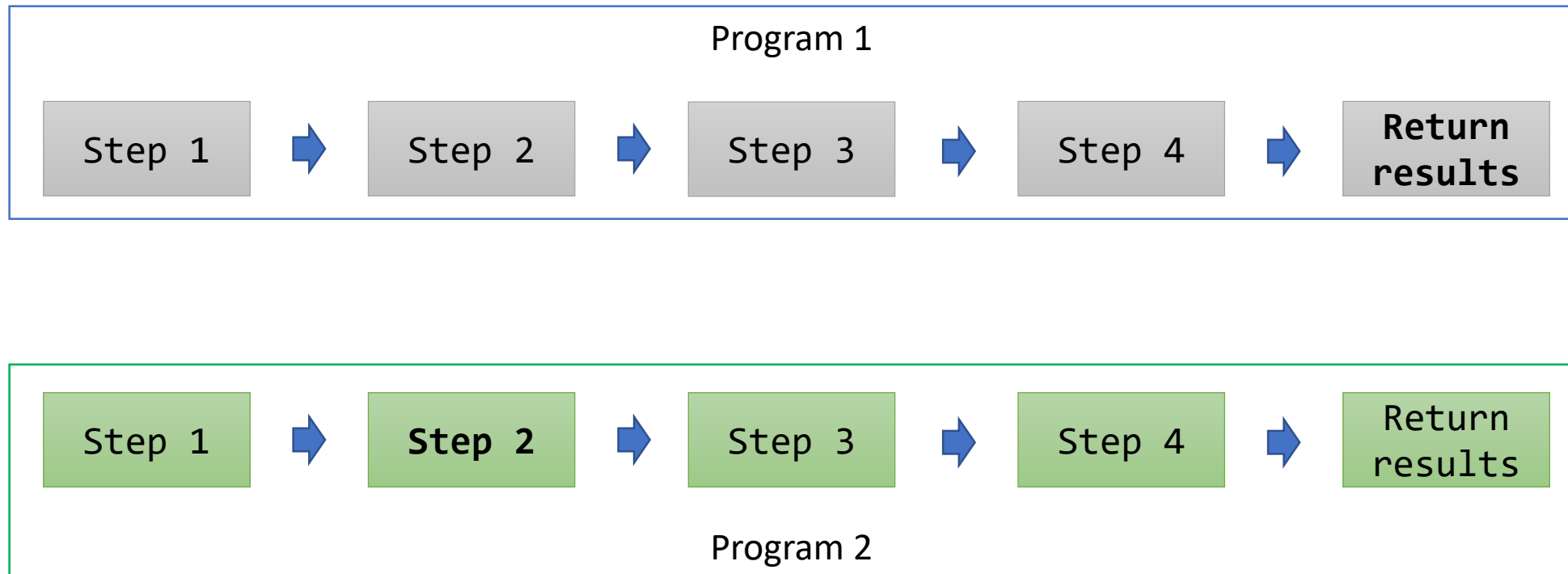
Synchronization

- Each process operates sequentially



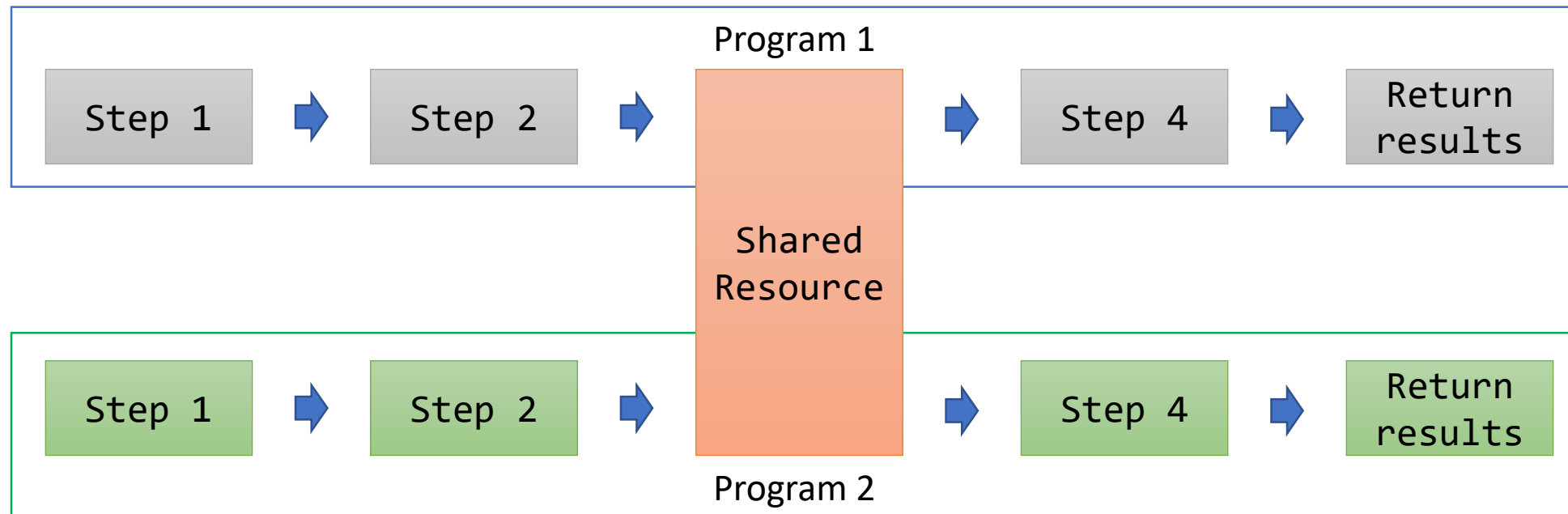
Synchronization

- Each process operates sequentially



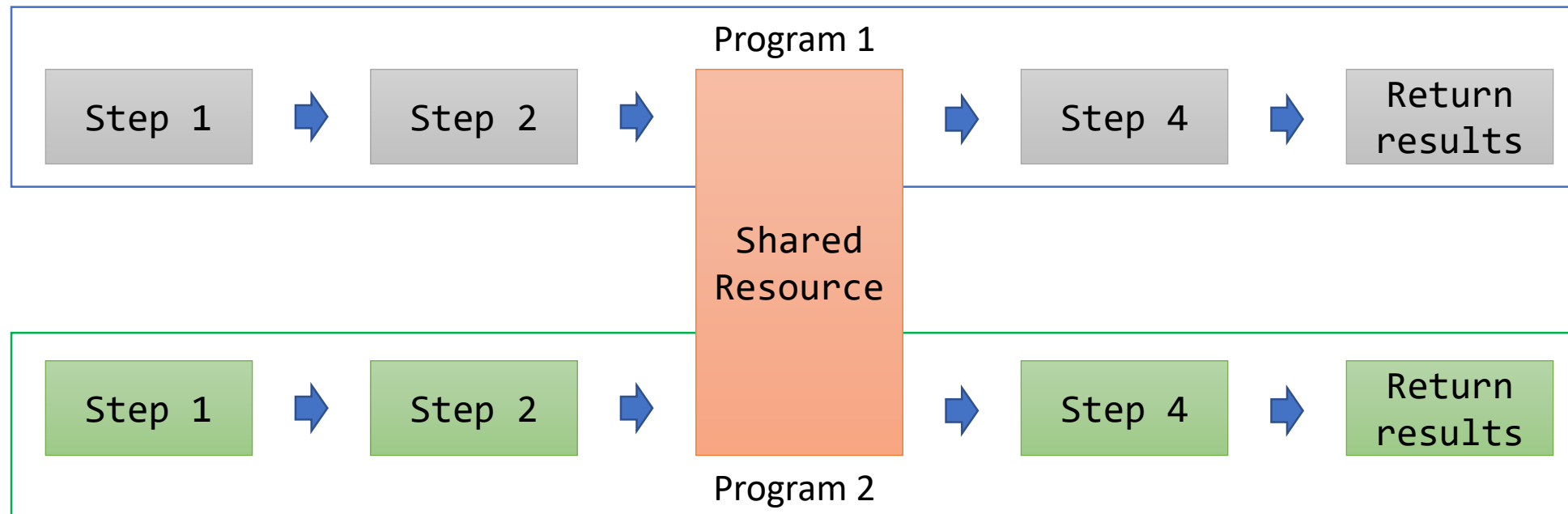
Synchronization

- All is fine until processes want to share data
 - Exchange data between multiple processes



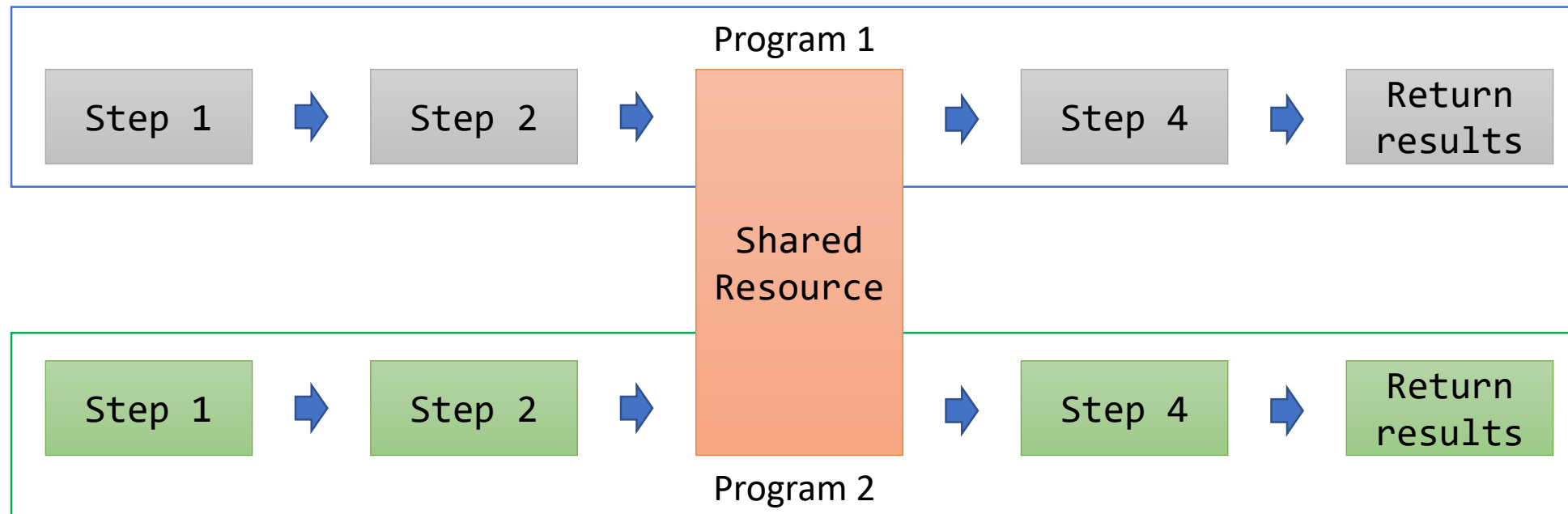
Synchronization

- All is fine until processes want to share data
 - Exchange data between multiple processes
 - Order of execution may affect the output



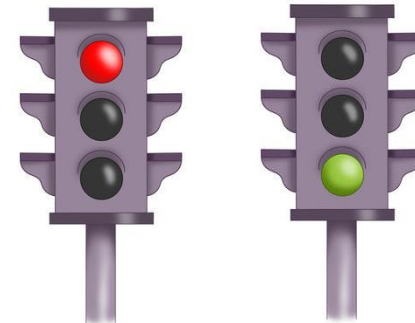
Synchronization

- All is fine until processes want to share data
 - Exchange data between multiple processes
 - Order of execution may affect the output
 - Entering this region requires control



Synchronization

- These issues apply to threads as well
- **Semaphore** is a protected integer variable that can facilitate and restrict access to shared sources in a multi-processing environment.
- **Mutex** allows only one program to enter a critical region. It has the principle of ownership.



Semaphore

- S – Integer (non-negative value at initialization)
- Q – Queue of processes/threads (empty at initialization)
- Two most common kinds of semaphores
 - Counting semaphores
 - Represent multiple resources
 - Binary semaphores
 - Represent two possible states (1 or 0 locked or unlocked)
 - Signaling mechanism

Semaphore – two basic operations

- `down()` / `wait()`
 - Decrements S
 - If S is now negative, the current process is blocked and placed in Q

Semaphore – two basic operations

- `down()` / `wait()`
 - Decrements S
 - If S is now negative, the current process is blocked and placed in Q
- `up()` / `signal()`
 - Increments S
 - If after the increment, S is still ≤ 0 , that means there is still some blocked process in the queue. One of them should be dequeued and becomes unblocked.

Semaphore – pseudo code

```
class Semaphore {  
    int value;  
    ProcessList pl;  
    void down ();  
    void up ();  
};
```

Semaphore – pseudo code

```
class Semaphore {  
    int value;  
    ProcessList pl;  
    void down ();  
    void up ();  
};
```

```
Semaphore code  
Semaphore::down ()  
{  
    value -= 1;  
    if (value < 0) {  
        // add this process to pl  
        Sleep ();  
    }  
}
```

Semaphore – pseudo code

```
class Semaphore {  
    int value;  
    ProcessList pl;  
    void down ();  
    void up ();  
};
```

```
Semaphore code  
Semaphore::down ()  
{  
    value -= 1;  
    if (value < 0) {  
        // add this process to pl  
        Sleep ();  
    }  
}  
Semaphore::up () {  
    Process P;  
    value += 1;  
    if (value <= 0) {  
        // remove a process P  
        // from pl  
        Wakeup (P);  
    }  
}
```

Project 1 – Discussion

- Declare a simple struct that contains an integer value and a queue of processes:

```
struct cs1550_sem
{
    int value;
    //priority queue
};
```

Project 1 – Discussion

- Declare a simple struct that contains an integer value and a queue of processes:

```
struct cs1550_sem
{
    int value;
    //priority queue
};
```

- Make two new system calls that each has the following signatures:

```
asmlinkage long sys_cs1550_down(struct cs1550_sem *sem)
asmlinkage long sys_cs1550_up(struct cs1550_sem *sem)
```

Project 1 - Syscalls and IPC

```
struct cs1550_sem
{
    int value;
    //priority queue
};
```

```
asmlinkage long sys_cs1550_down(struct cs1550_sem *sem)
```

- Here the process can sleep.

```
asmlinkage long sys_cs1550_up(struct cs1550_sem *sem)
```

Project 1 - Syscalls and IPC

```
struct cs1550_sem
{
    int value;
    //priority queue
};
```

```
asm linkage long sys_cs1550_down(struct cs1550_sem *sem)
```

- Here the process can sleep.
- Mark the task as not ready

```
asm linkage long sys_cs1550_up(struct cs1550_sem *sem)
```

Project 1 - Syscalls and IPC

```
struct cs1550_sem
{
    int value;
    //priority queue
};
```

```
asmlinkage long sys_cs1550_down(struct cs1550_sem *sem)
```

- Here the process can sleep.
- Mark the task as not ready
- set the current stat as “TASK_INTERRUPTIBLE”

```
asmlinkage long sys_cs1550_up(struct cs1550_sem *sem)
```


Project 1 - Syscalls and IPC

```
struct cs1550_sem
{
    int value;
    //priority queue
};
```

```
asm linkage long sys_cs1550_down(struct cs1550_sem *sem)
```

- Here the process can sleep.
- Mark the task as not ready
- set the current stat as “TASK_INTERRUPTIBLE”
- Invoke **schedule()** to get next task.

The process can sleep. Mark the task as not ready (but can be awoken by signals)

```
asm linkage long sys_cs1550_up(struct cs1550_sem *sem)
```

Project 1 - Syscalls and IPC

```
struct cs1550_sem
{
    int value;
    //priority queue
};
```

```
asmlinkage long sys_cs1550_down(struct cs1550_sem *sem)
```

```
asmlinkage long sys_cs1550_up(struct cs1550_sem *sem)
```

- wake_up_process(sleeping_task);



Struct that represents a process put to sleep
by the **down()** method

Project 1 - Discussion

- The semaphores need to be implemented as part of the kernel
- We need to do our increment or decrement and the following check on it **atomically**
 - We can use spin locks for that

Project 1 - Discussion

- The semaphores need to be implemented as part of the kernel
- We need to do our increment or decrement and the following check on it **atomically**
 - We can use spin locks for that
- Create a spinlock with a provided macro:
DEFINE_SPINLOCK(sem_lock);

Project 1 - Discussion

- The semaphores need to be implemented as part of the kernel
- We need to do our increment or decrement and the following check on it **atomically**
 - We can use spin locks for that
- Create a spinlock with a provided macro:
DEFINE_SPINLOCK(sem_lock);
- We can then surround our critical regions with the following:
spin_lock(&sem_lock);
// critical region
spin_unlock(&sem_lock);

Project 1 - Tips

- Using **kmalloc** to allocate memory
 - Synopsis: void * kmalloc (size_t size, gfp_t flags);
 - <https://www.kernel.org/doc/html/docs/kernel-api/API-kmalloc.html>
- printk(), you may want to use for printing out debugging messages from the kernel.
- In general, you can use some library standard C functions, but not all. If they do an OS call, they may not work.

Project 1 - Tips

- Copy the test programs to verify your Semaphore Implementation:
 - [Trafficsim test program for project 1](#)


trafficsim.c

trafficsim-mutex.c

trafficsim-strict-order.c

Project 1 – Building and running test programs

```
gcc -m32 -o trafficsim -I /u/OSLab/USERNAME/linux-2.6.23.1/include/ trafficsim.c
```



Tell gcc to look for the new include files

Cannot run our test program on thoth.cs.pitt.edu

Test the program under QEMU

- Installed the modified kernel
- Copy the test program to QEMU
- Then just run it

CS 1550 – Project 1 kernel compilation



Local host

Linux-devel

Qemu

If you want to run the test programs
You should **compile them** in thoth
and **copy to your Linux-devel** with
the **scp command**

Remote Host



thoth.cs.pitt.edu

trafficsim.c
trafficsim-mutex.c
...

Project 1 – Files for submission

- Syscalls you will modify the files
 - Actual implementation
 - linux-2.6.23.1/kernel/**sys.c**
 - Syscall Number map
 - linux-2.6.23.1/arch/i386/kernel/**syscall_table.S**
 - Exposes syscall number to C programs
 - linux-2.6.23.1/include/asm/**unistd.h**
- A header file named **sem.h**
 - All required declarations into the file.
 - Should be in the same folder as the test case file when compiling.

Project 1 – Files for submission

- Syscalls you will modify the files
 - Actual implementation
 - linux-2.6.23.1/kernel/**sys.c**
 - Syscall Number map
 - linux-2.6.23.1/arch/i386/kernel/**syscall_table.S**
 - Exposes syscall number to C programs
 - linux-2.6.23.1/include/asm/**unistd.h**
- A header file named **sem.h**
 - All required declarations into the file.
 - Should be in the same folder as the test case file when compiling.

Remote Host



thoth.cs.pitt.edu

bzlImage
System.map
sys.c
syscall_table.S
unistd.h
sem.h (you will create)

Project 1 – Files for submission

- Syscalls you will modify the files
 - Actual implementation
 - linux-2.6.23.1/kernel/**sys.c**
 - Syscall Number map
 - linux-2.6.23.1/arch/i386/kernel/**syscall_table.S**
 - Exposes syscall number to C programs
 - linux-2.6.23.1/include/asm/**unistd.h**
- A header file named **sem.h**
 - All required declarations into the file.
 - Should be in the same folder as the test case file when compiling.



CS 1550

Week 4 – Project 1 Discussion

Teaching Assistant

Henrique Potter