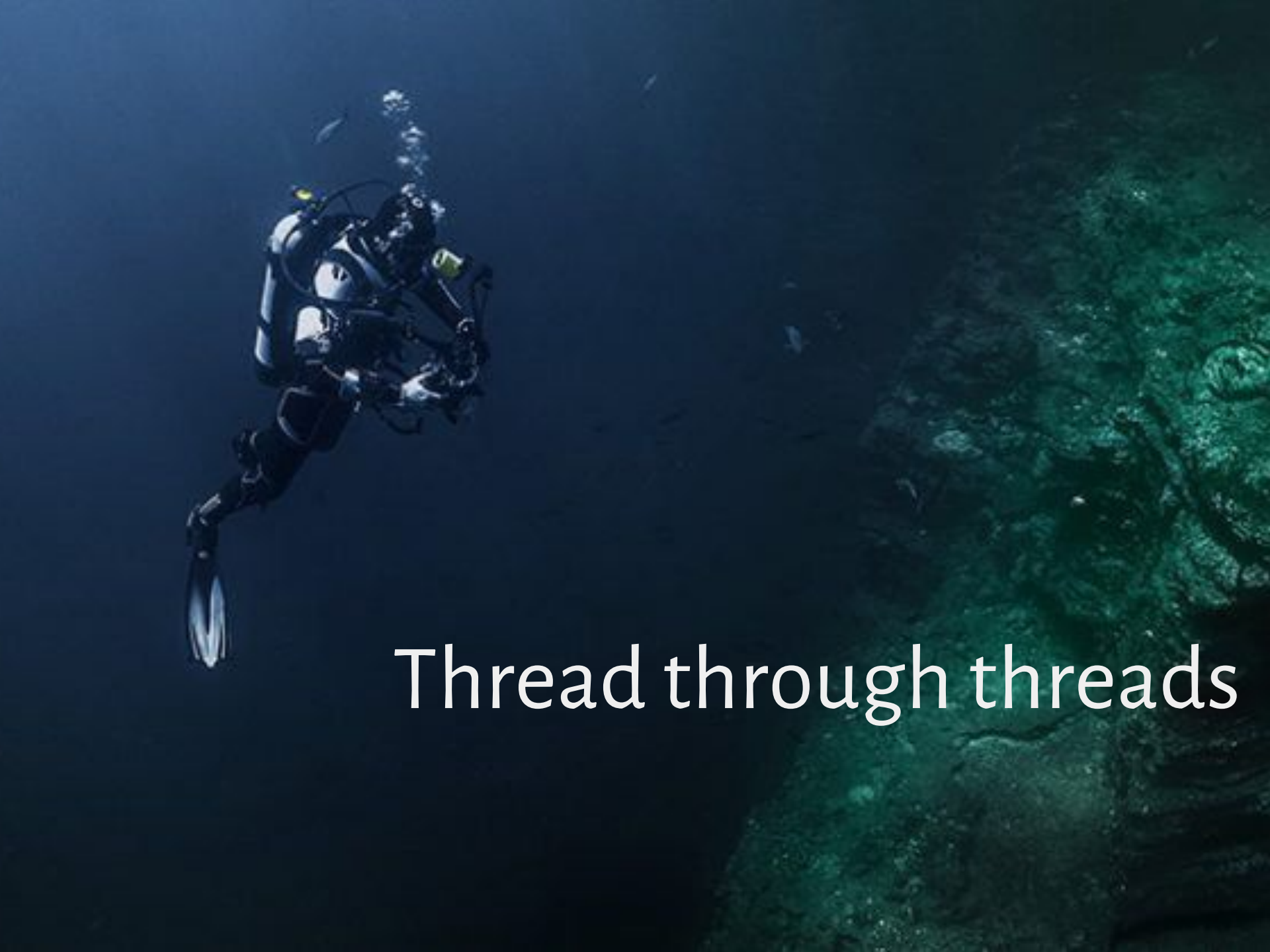
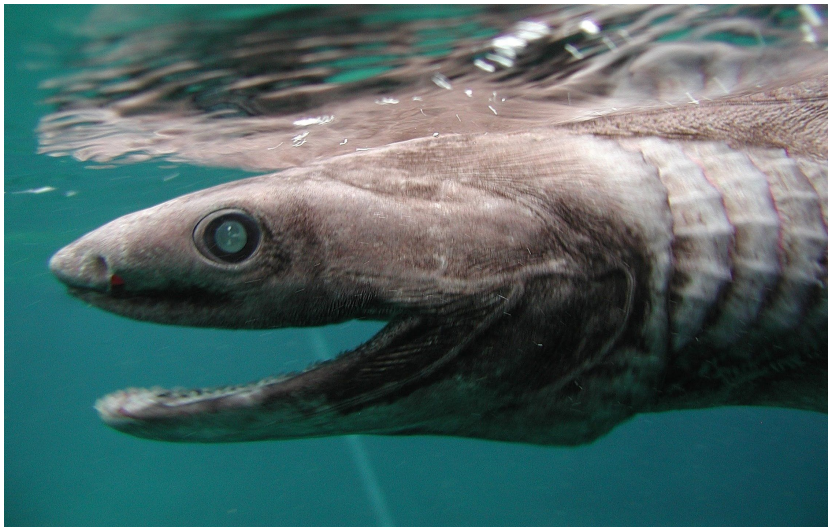
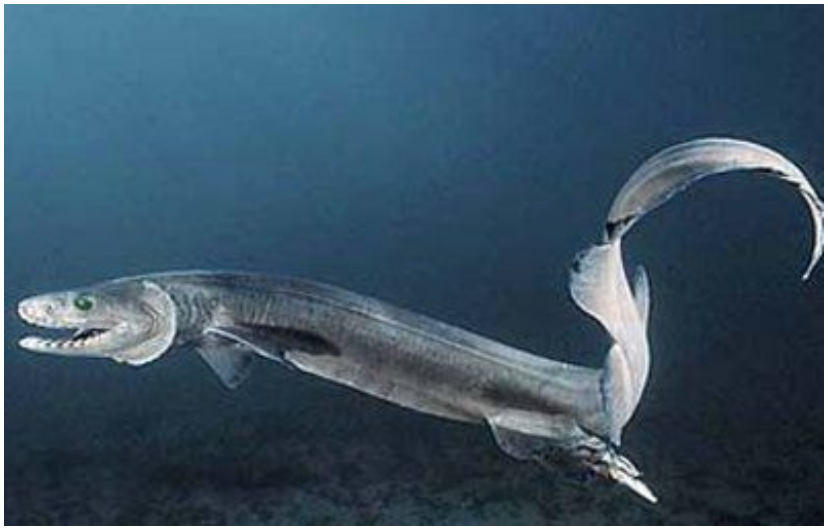

Thread through threads

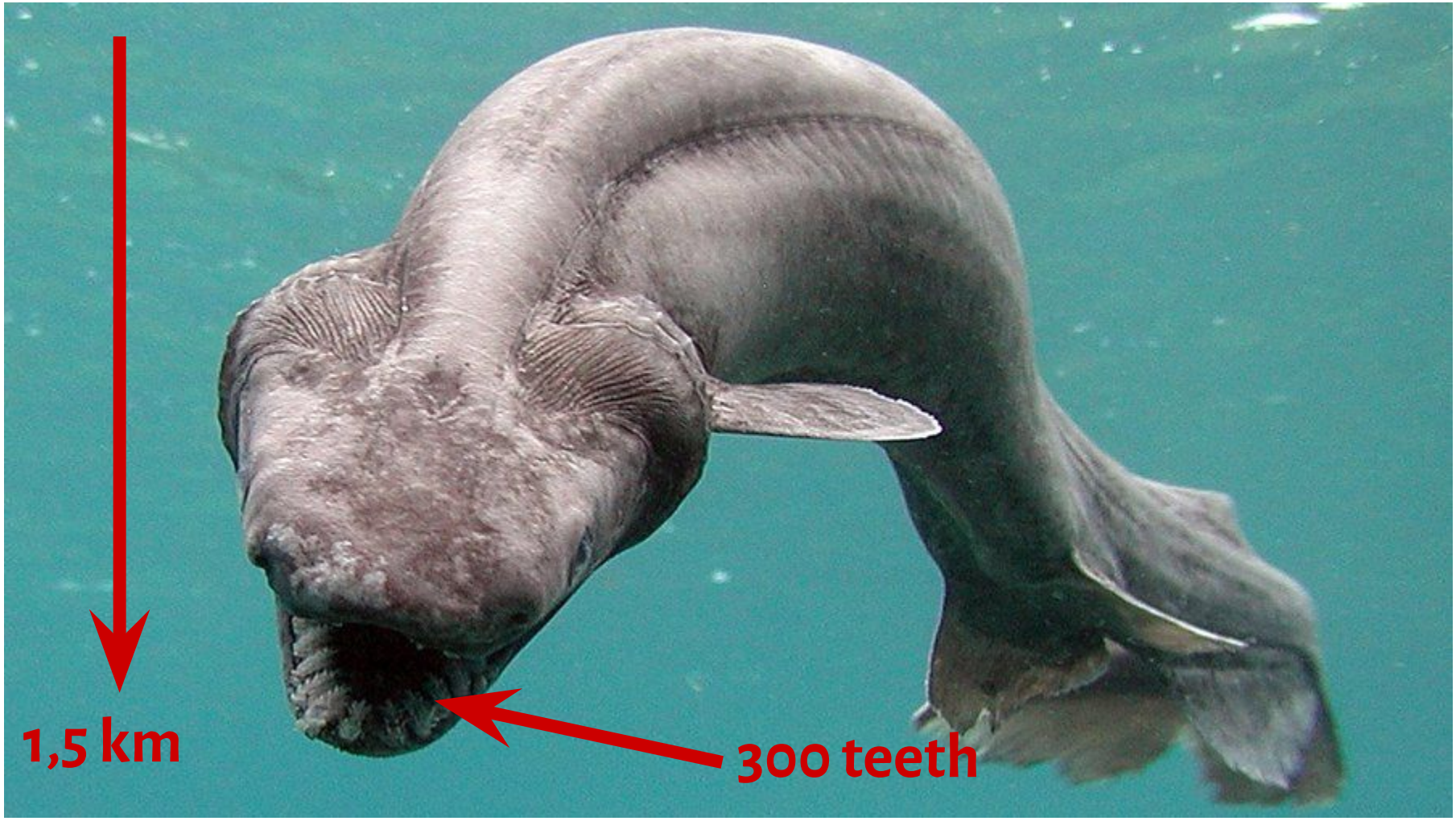
Frilled Shark

Frilled Shark





Frilled Shark



Frilled Shark

- ★ preys of size of half of their body
- ★ pregnancy of 3,5 years
- ★ no changes for 80 million years

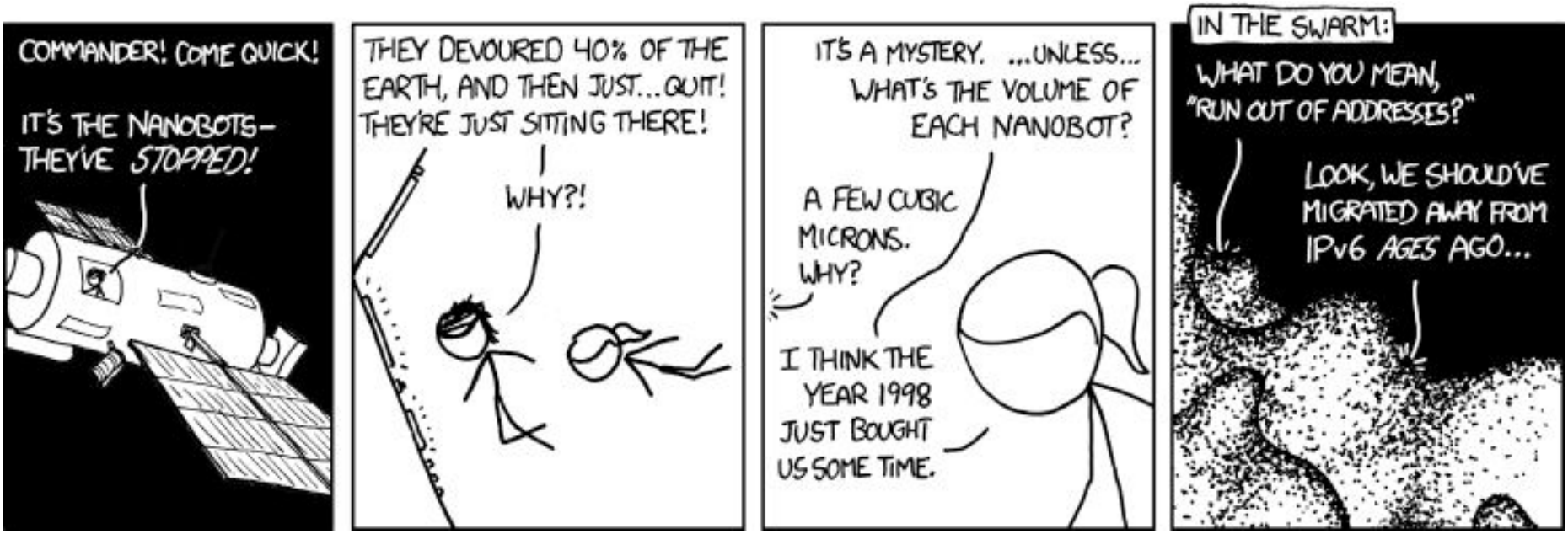
https://sharkopedia.discovery.com/types-of-sharks/frilled-shark/#frilled-sharks-are-known-as-living-fossils

Assignment #1

Make sure that:

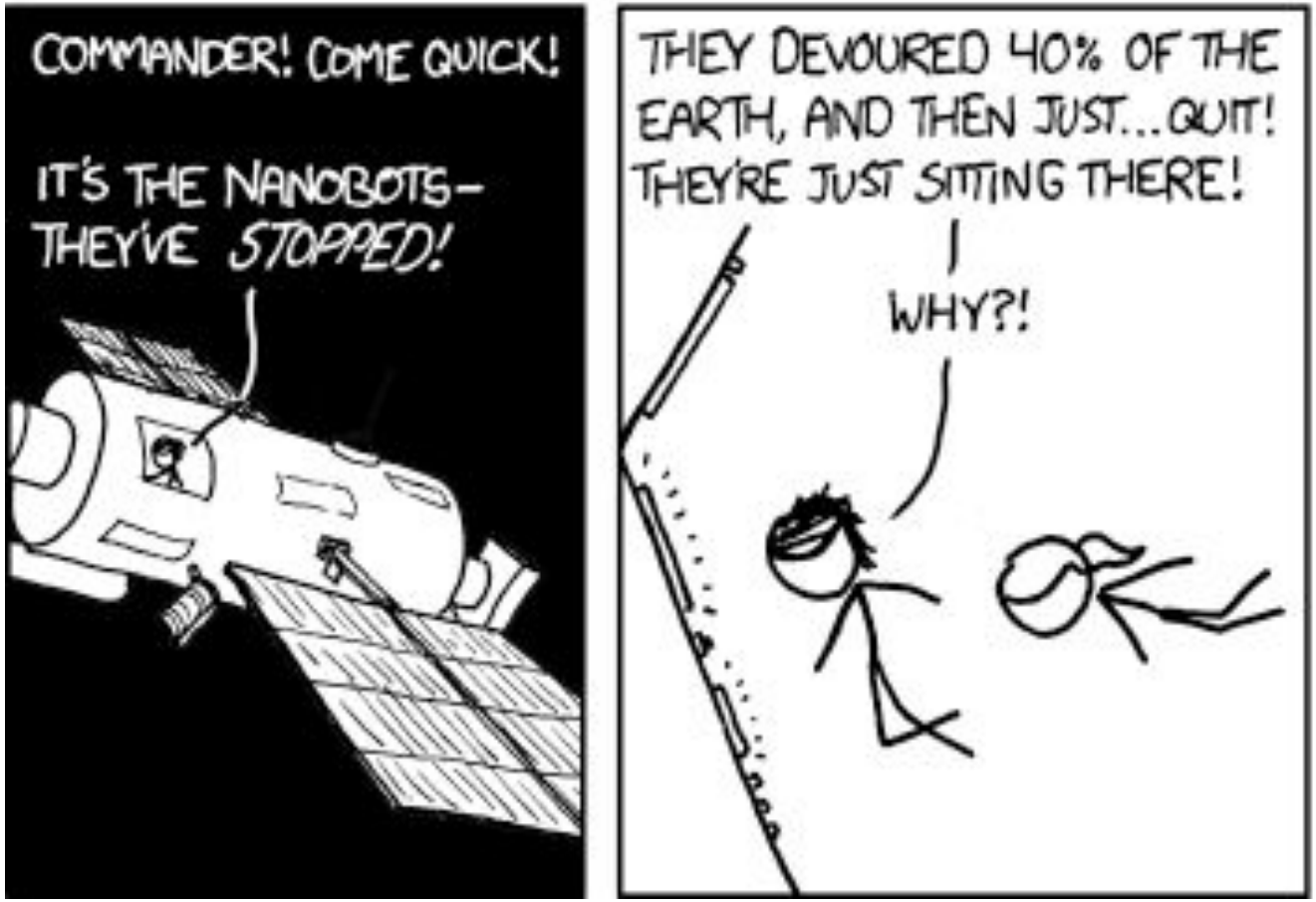
- ★ uninitialized data are kept in .bss section
- ★ input bytes are buffered
- ★ correctness of permutations is verified immediately

Is 2^{64} enough?



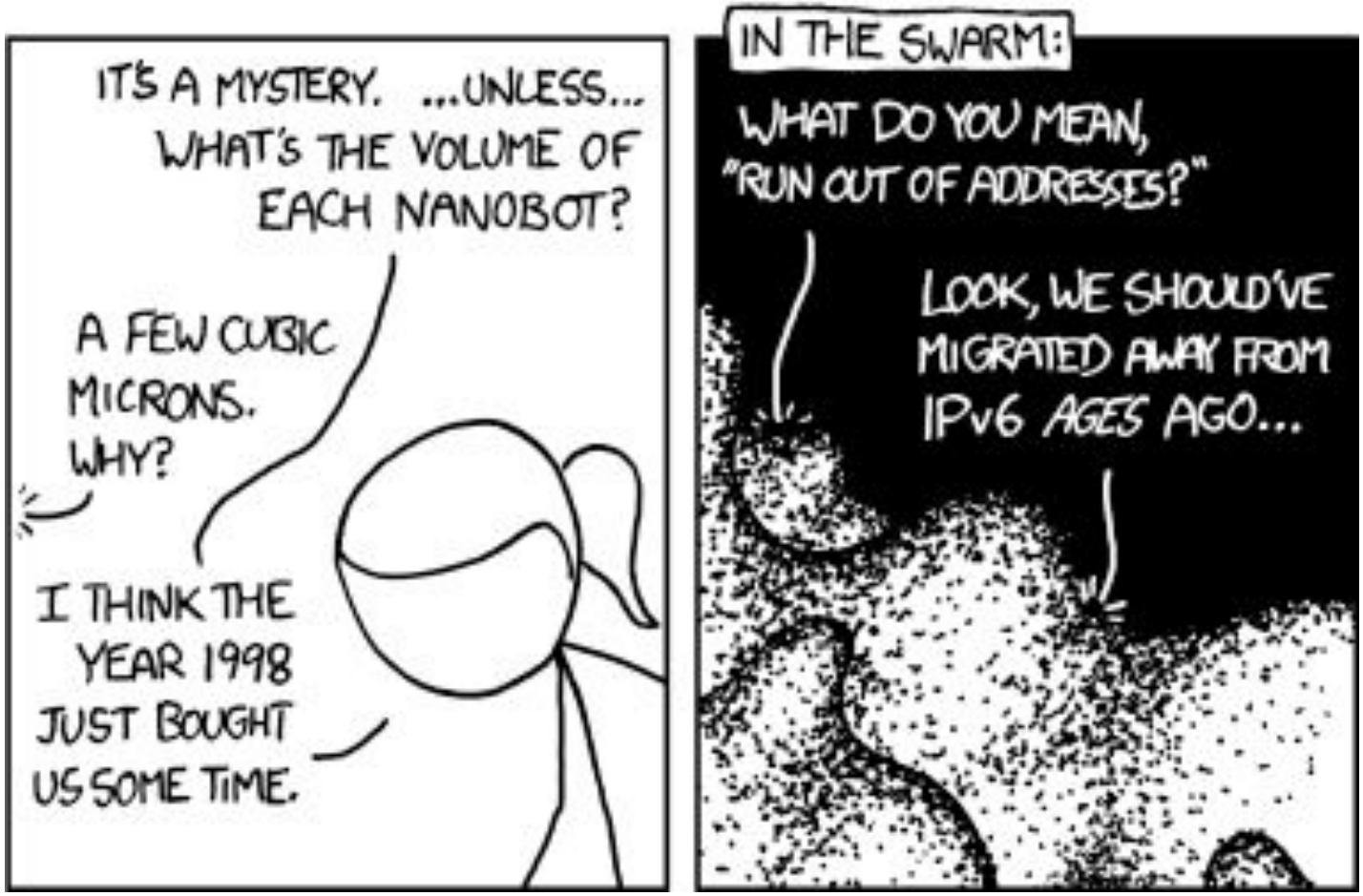
https://xkcd.com/865/

Is 2^{64} enough?

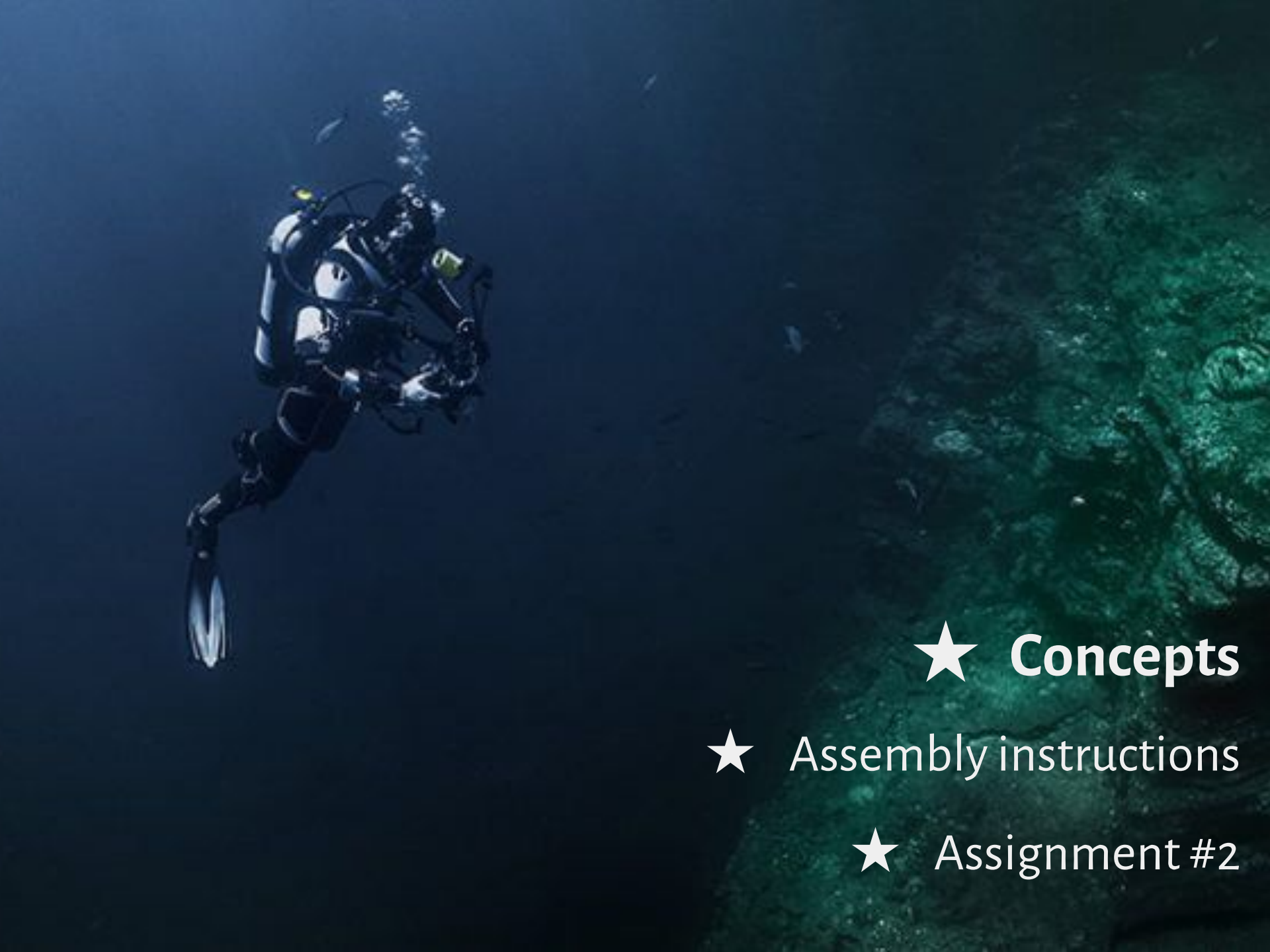


https://xkcd.com/865/

Is 2^{64} enough?



https://xkcd.com/865/


★ Concepts
★ Assembly instructions
★ Assignment #2

A concept of a *process*

*A **computer program** is a passive collection of instructions, while a **process** is the actual execution of those instructions.*

https://en.wikipedia.org/wiki/Process_(computing)

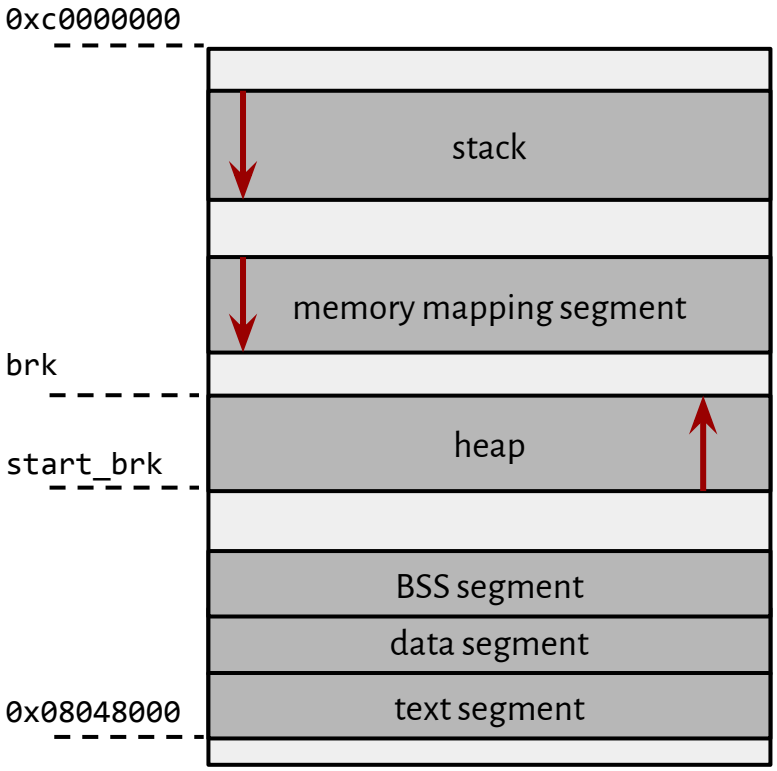
A concept of a **process**

a group of related resources

Andrew S. Tanenbaum: *Operating Systems Design and Implementation*, 3rd Edition; p. 64 - 68

A concept of a **process**

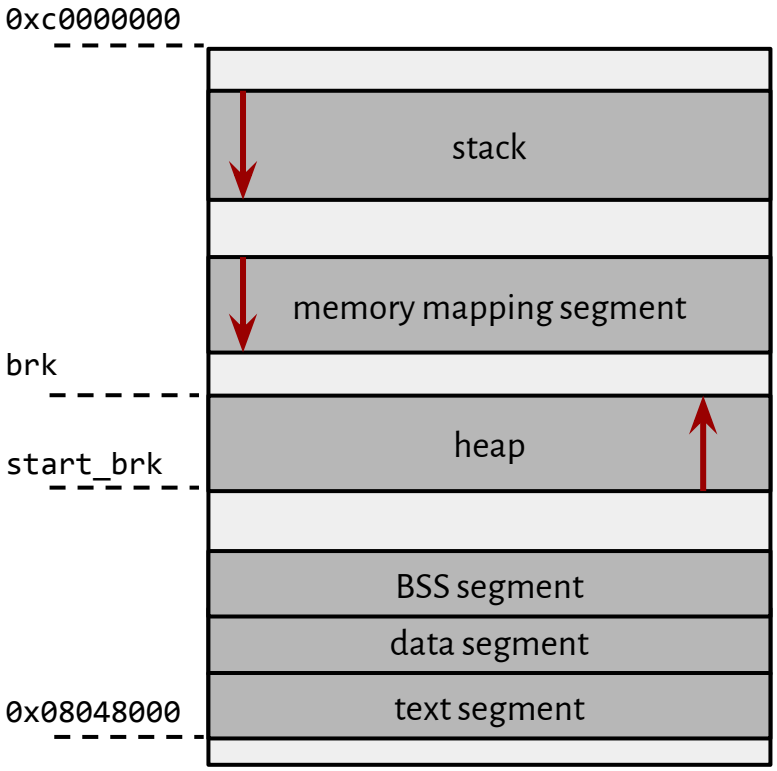
a group of related resources



Andrew S. Tanenbaum: *Operating Systems Design and Implementation*, 3rd Edition; p. 64 - 68

A concept of a **process**

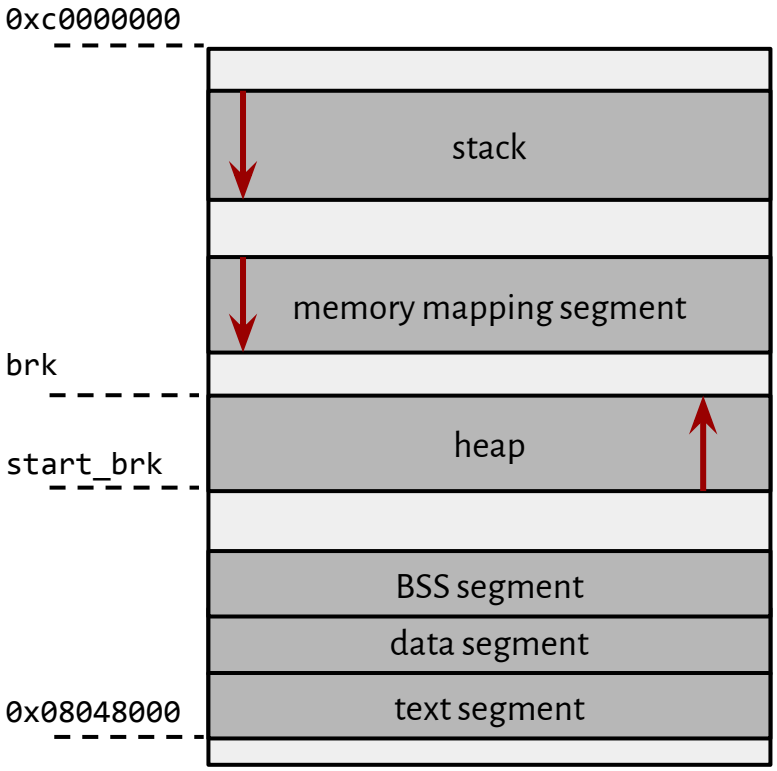
a group of related resources



Andrew S. Tanenbaum: *Operating Systems Design and Implementation*, 3rd Edition; p. 64 - 68

A concept of a **process**

a group of related resources



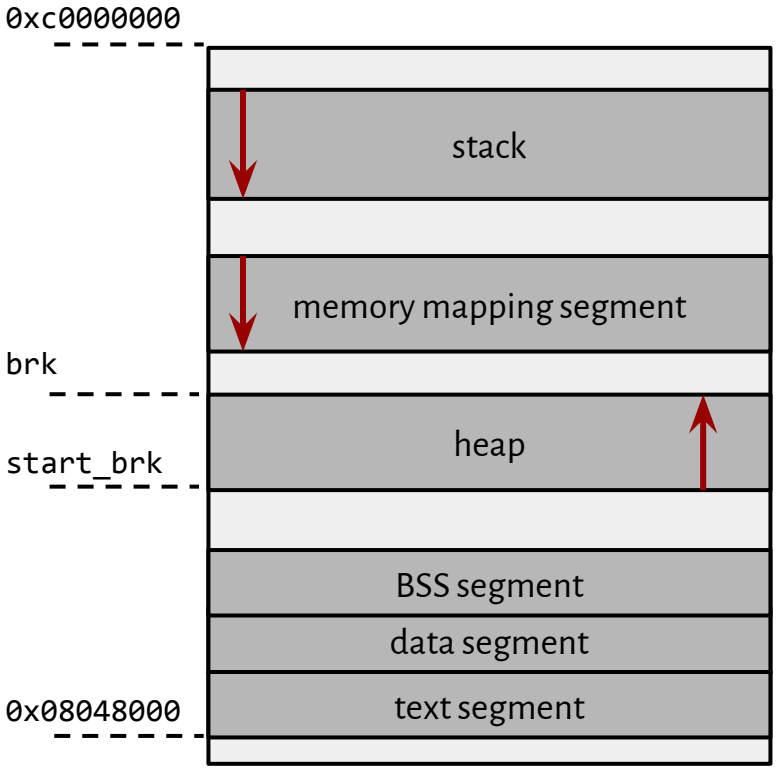
Andrew S. Tanenbaum: *Operating Systems Design and Implementation*, 3rd Edition; p. 64 - 68

A concept of a **thread**

execution of instructions within a process

A concept of a **thread**

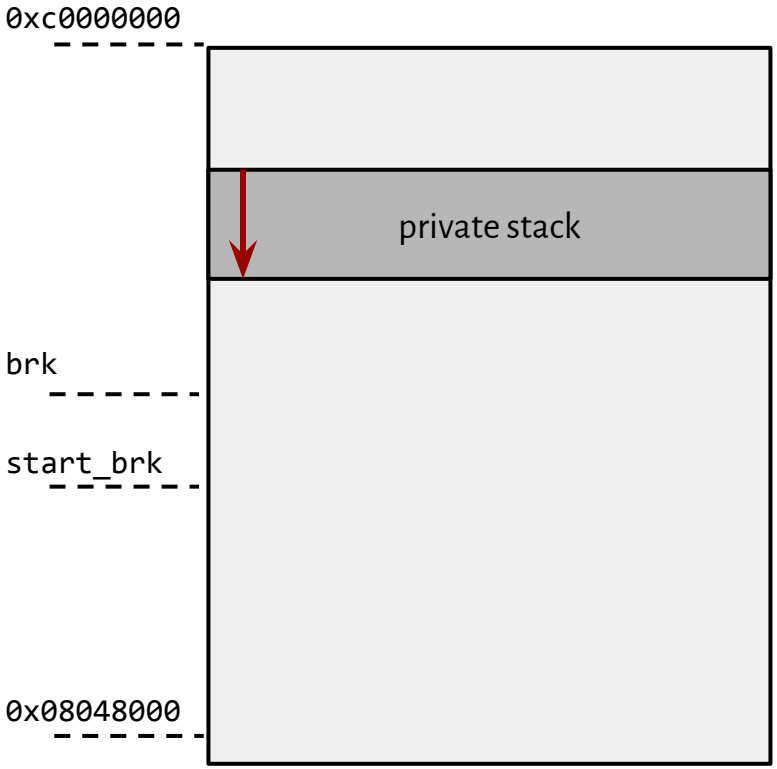
execution of instructions within a process



shared resources

A concept of a **thread**

execution of instructions within a process



private resources

Let's see!

simple.c

```
$ gcc -pthread simple.c -o simple; ./simple

$ cat /proc/29298/maps
```

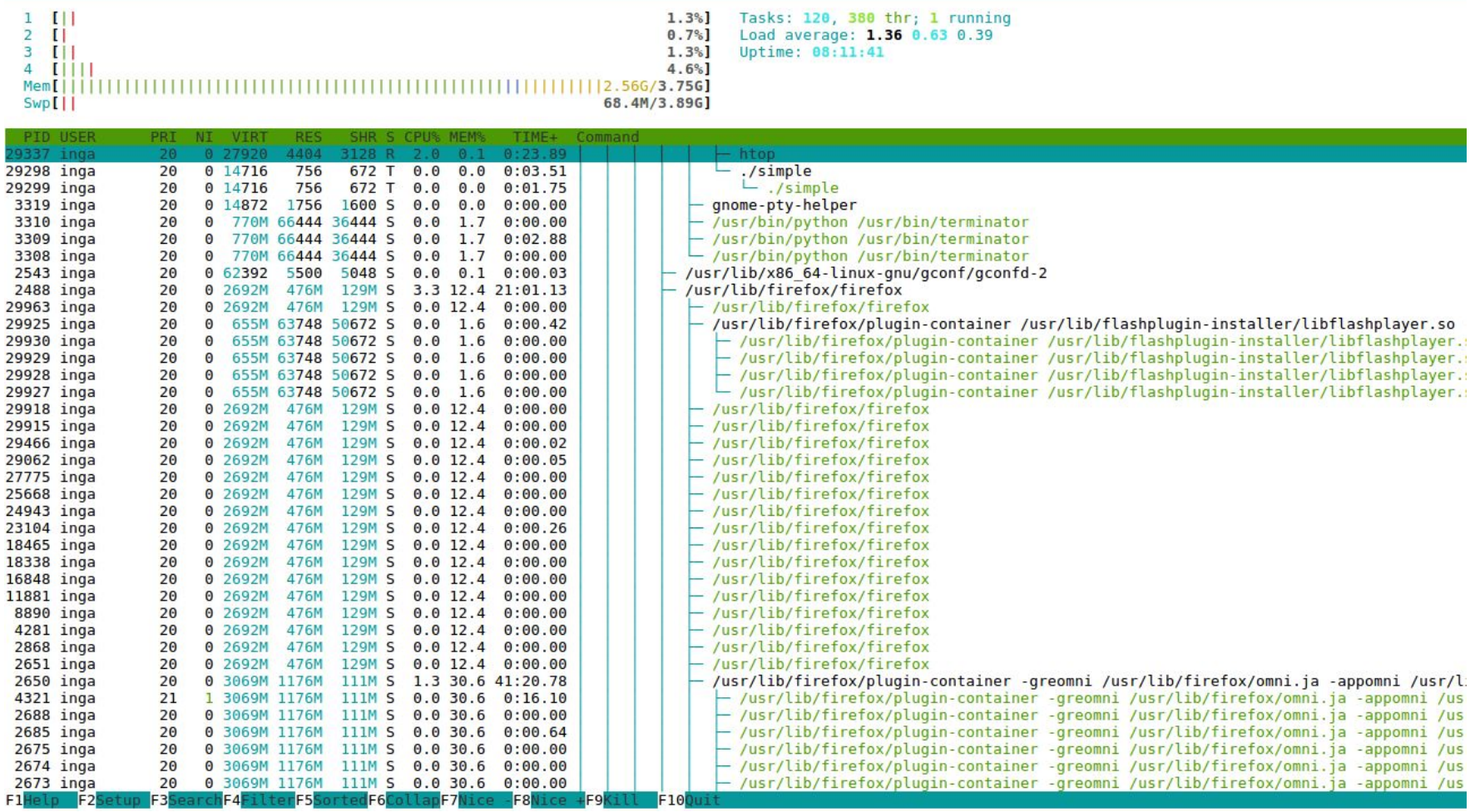
Let's see!

simple.c

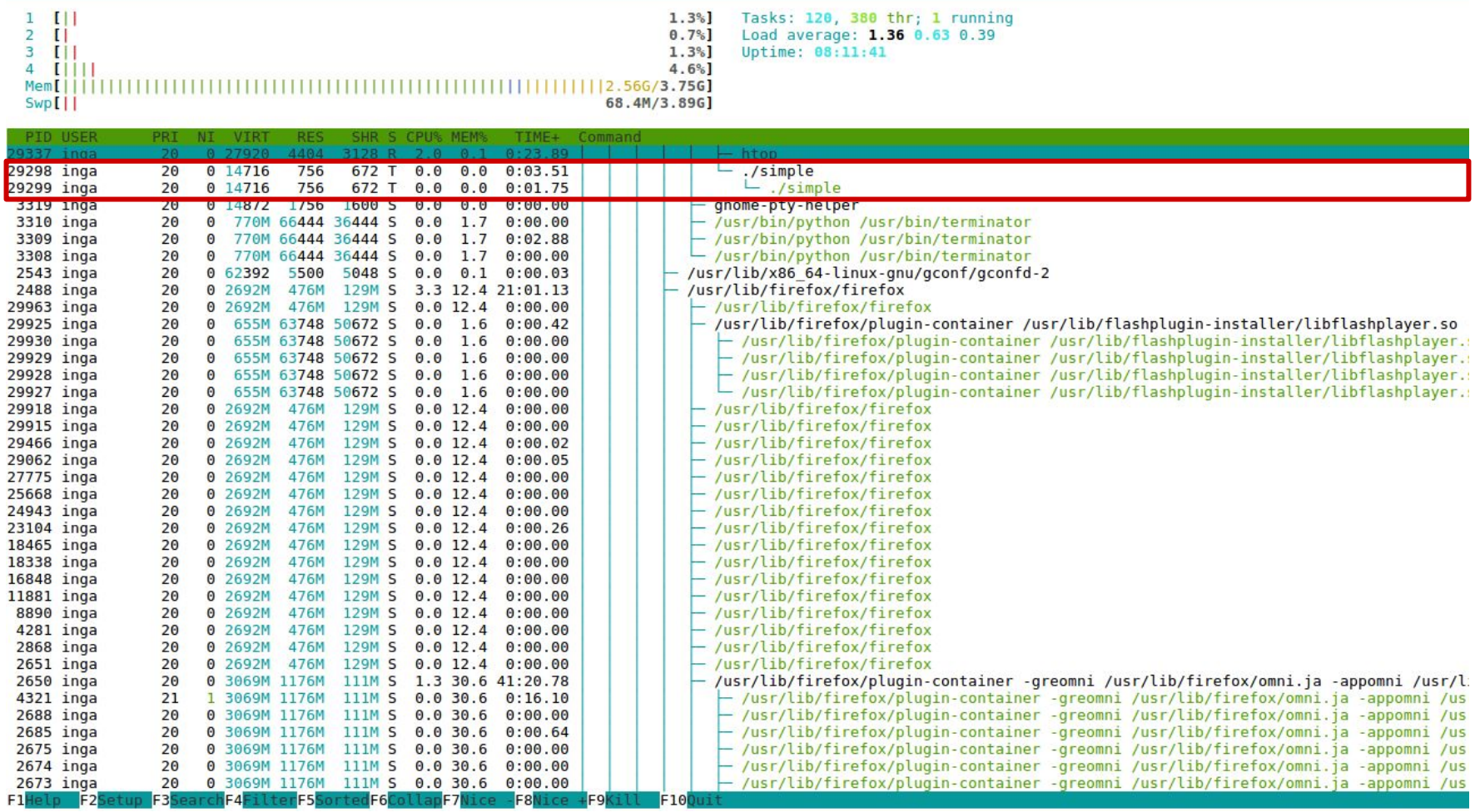
```
$ gcc -pthread simple.c -o simple; ./simple

$ cat /proc/29298/maps
```

```
00400000-00401000 r-xp 00000000 08:02 43010148                   /home/inga/SO/3/simple
00600000-00601000 r--p 00000000 08:02 43010148                   /home/inga/SO/3/simple
00601000-00602000 rw-p 00001000 08:02 43010148                   /home/inga/SO/3/simple
019d0000-019f1000 rw-p 00000000 00:00 0                          [heap]
7ff49b60f000-7ff49b610000 ---p 00000000 00:00 0
7ff49b610000-7ff49be10000 rw-p 00000000 00:00 0
[...]
7ff49c61d000-7ff49c61e000 rw-p 00000000 00:00 0
7ffffb6c9000-7ffffb6ea000 rw-p 00000000 00:00 0                  [stack]
7ffffb779000-7ffffb77b000 r--p 00000000 00:00 0                  [vvar]
7ffffb77b000-7ffffb77d000 r-xp 00000000 00:00 0                  [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0          [vsyscall]
```

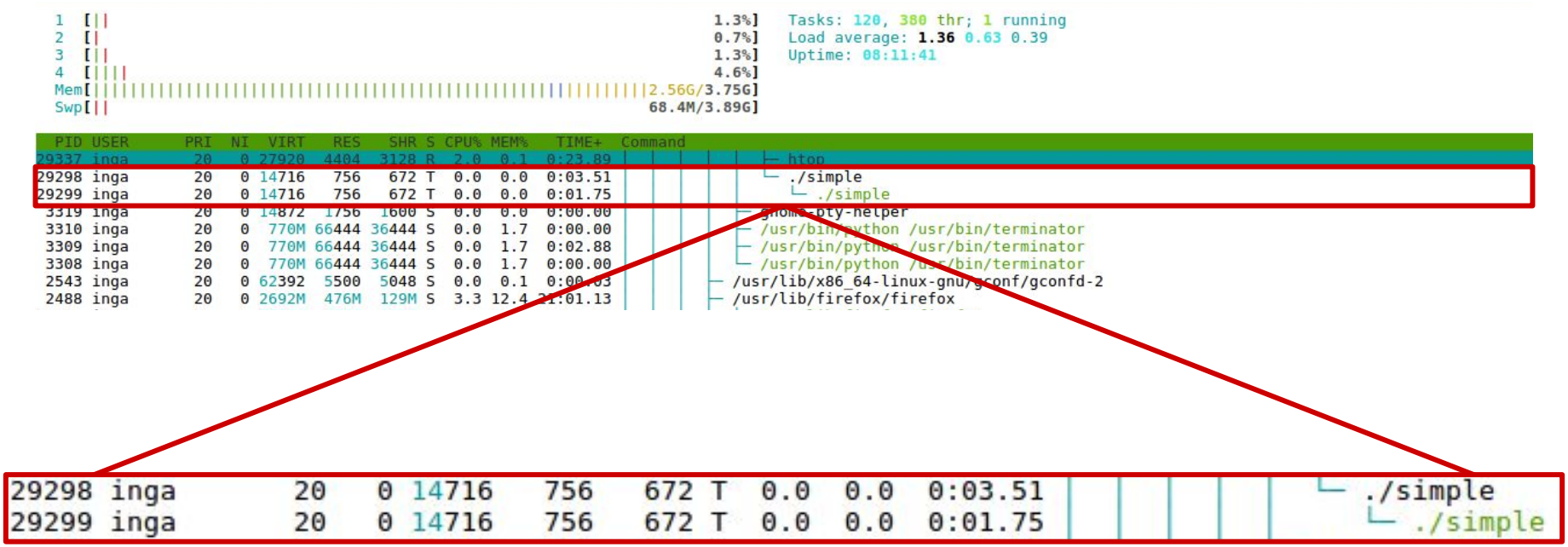
My adventure with **htop**



My adventure with **htop**



My adventure with **htop**



Why PIDs are different?

http://opensourceforu.com/2011/08/light-weight-processes-dissecting-linux-threads/

What are threads in Linux?

processes that share certain resources

What are threads in Linux?

tasks *that share certain resources*

What are threads in Linux?

tasks *that share certain resources*

```
$ strace ./simple
```

```
clone(child_stack=0x7f18a5423ff0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7f18a54249d0,
tls=0x7f18a5424700, child_tidptr=0x7f18a54249d0) = 29299
child thread pid is 29299
```

What are threads in Linux?

tasks *that share certain resources*

```
$ strace ./simple
```

```
clone(child_stack=0x7f18a5423ff0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7f18a54249d0,
tls=0x7f18a5424700, child_tidptr=0x7f18a54249d0) = 29299
child thread pid is 29299
```

```
$ cat /proc/29298/maps
```

```
$ cat /proc/29299/maps
```

What are threads in Linux?

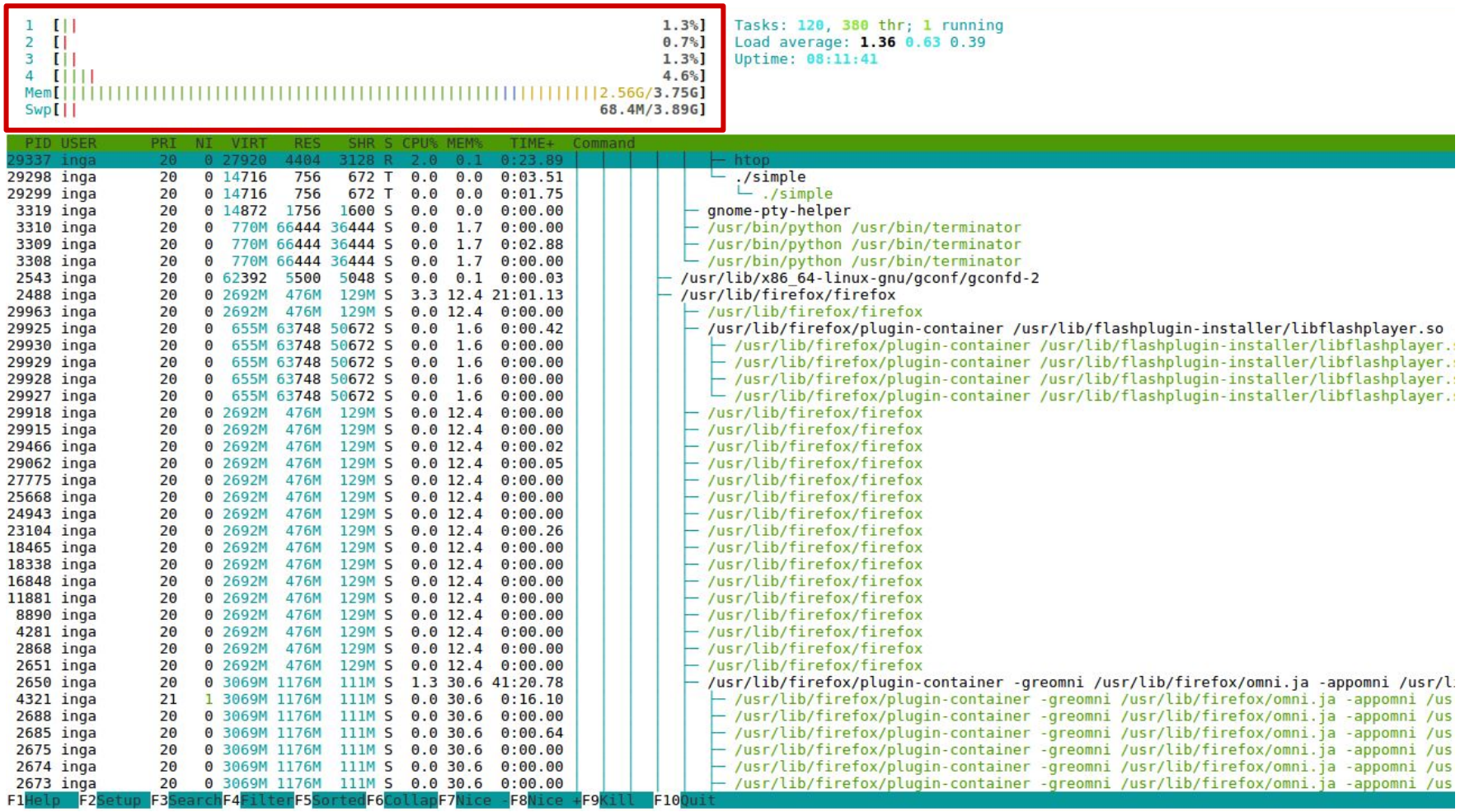
processes that share certain resources

```
$ strace ./simple
```

```
clone(child_stack=0x7f18a5423ff0,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETT
LS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7f18a54249d0,
tls=0x7f18a5424700, child_tidptr=0x7f18a54249d0) = 29299
child thread pid is 29299
```

```
$ cat /proc/29298/maps

$ cat /proc/29299/maps

$ ls -s /proc/29299
```

My adventure with **htop**




★ Concepts
★ Assembly instructions
★ Assignment #2

Assembly - atomic instructions

**lock **

<instr>

Assembly - atomic instructions

**lock **

<instr>

xchg dst, src

dst' = src

src' = dst

Assembly - atomic instructions

**lock **

<instr>

xchg dst, src

```
dst' = src
src' = dst
```

cmpxchg dst, src

```
if rax == dst:
    dst' = src
else:
    rax' = dst
```

Scenario

```
void *inc_thread(void *);


typedef struct {
  int *value;
  int count;
} thread_data_t;
```

```
$ cd /home/students/inf/PUBLIC/SO/scenariusze/3
```

```
$ make
```

```
$ ./inc_thread_test_naive  3 10000000
```

Scenario

```
global inc_thread

section .text

align 8
inc_thread:
  mov     rsi, [rdi]      ; value
  mov     ecx, [rdi + 8]  ; count
  jmp     end
count_loop:
  inc     dword [rsi]     ; ++*value
count_test:
  sub     ecx, 1          ; --count
  jge     loop            ; skok, gdy count >= 0
  xor     eax, eax        ; return NULL
  ret
```

lock

Scenario

```
global inc_thread

section .text

align 8
inc_thread:
  mov     rsi, [rdi]      ; value
  mov     ecx, [rdi + 8]  ; count
  jmp     end
count_loop:
  inc     dword [rsi]     ; ++*value
count_test:
  sub     ecx, 1          ; --count
  jge     loop            ; skok, gdy count >= 0
  xor     eax, eax        ; return NULL
  ret
```

lock

xchg

Spinlock

```
wait:
    if (spinlock == 0) {
        spinlock = 1
        go to critical_section
        spinlock = 0
    } else {
        go to wait
    }

critical_section:
…
```

Spinlock

```
wait:
    if (spinlock == 0) {
        spinlock = 1
        go to critical_section
        spinlock = 0
    } else {
        go to wait
    }

critical_section:
…
```

Spinlock

```
wait:
    if (spinlock == 0) {
        spinlock = 1
        go to critical_section
        spinlock = 0
    } else {
        go to wait
    }

critical_section:
…
```

Spinlock

```
wait:
    if (spinlock == 0) {
        spinlock = 1
        go to critical_section
        spinlock = 0
    } else {
        go to wait
    }

critical_section:
…
```

Spinlock

```
wait:
    if (spinlock == 0) {
        spinlock = 1
        go to critical_section
        spinlock = 0
    } else {
        go to wait
    }

critical_section:
…
```

What does need to be atomic?

Spinlock

```
wait:
    if (spinlock == 0) {
        spinlock = 1
        go to critical_section
        spinlock = 0
    } else {
        go to wait
    }

critical_section:
…
```

atomic

Scenario

```
global inc_thread

section .text

align 8
inc_thread:
  mov     rsi, [rdi]      ; value
  mov     ecx, [rdi + 8]  ; count
  jmp     end
count_loop:
  inc     dword [rsi]     ; ++*value
count_test:
  sub     ecx, 1          ; --count
  jge     loop            ; skok, gdy count >= 0
  xor     eax, eax        ; return NULL
  ret
```

lock

xchg

cmpxchg

Scenario

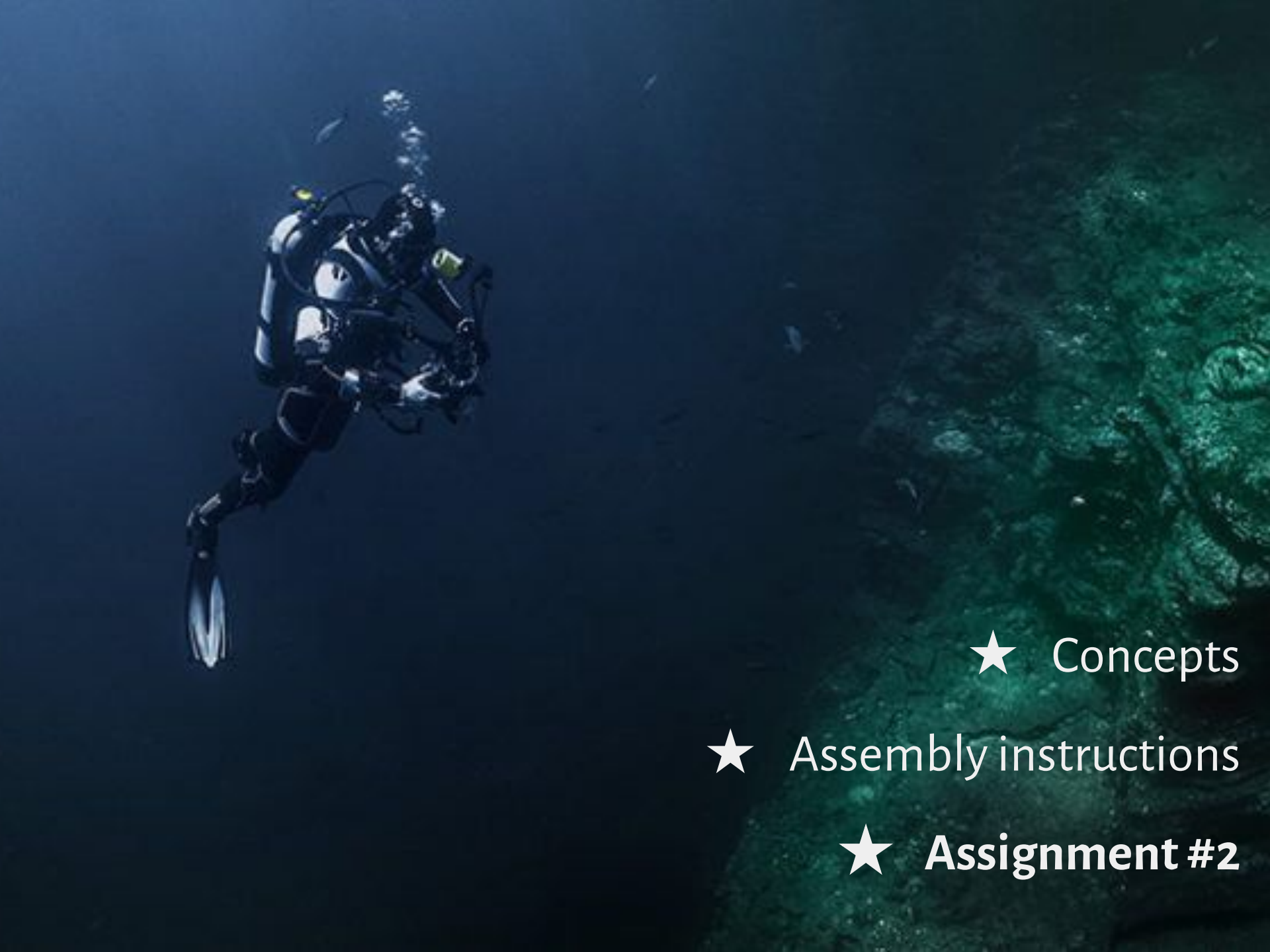
```
global inc_thread

section .text

align 8
inc_thread:
  mov     rsi, [rdi]      ; value
  mov     ecx, [rdi + 8]  ; count
  jmp     end
count_loop:
  inc     dword [rsi]     ; ++*value
count_test:
  sub     ecx, 1          ; --count
  jge     loop            ; skok, gdy count >= 0
  xor     eax, eax        ; return NULL
  ret
```

lock

xchg

cmpxchg

bts, btr


★ Concepts
★ Assembly instructions
★ **Assignment #2**

Assignment #2

Solutions expected by **26 March** 2018, 8 p.m.

Put your solutions here:

https://svn.mimuw.edu.pl/repos/SO/studenci/login/zadanie2

If you want to consult your solution let me know before:

23 March 2018, 8 p.m.

Assignment #2

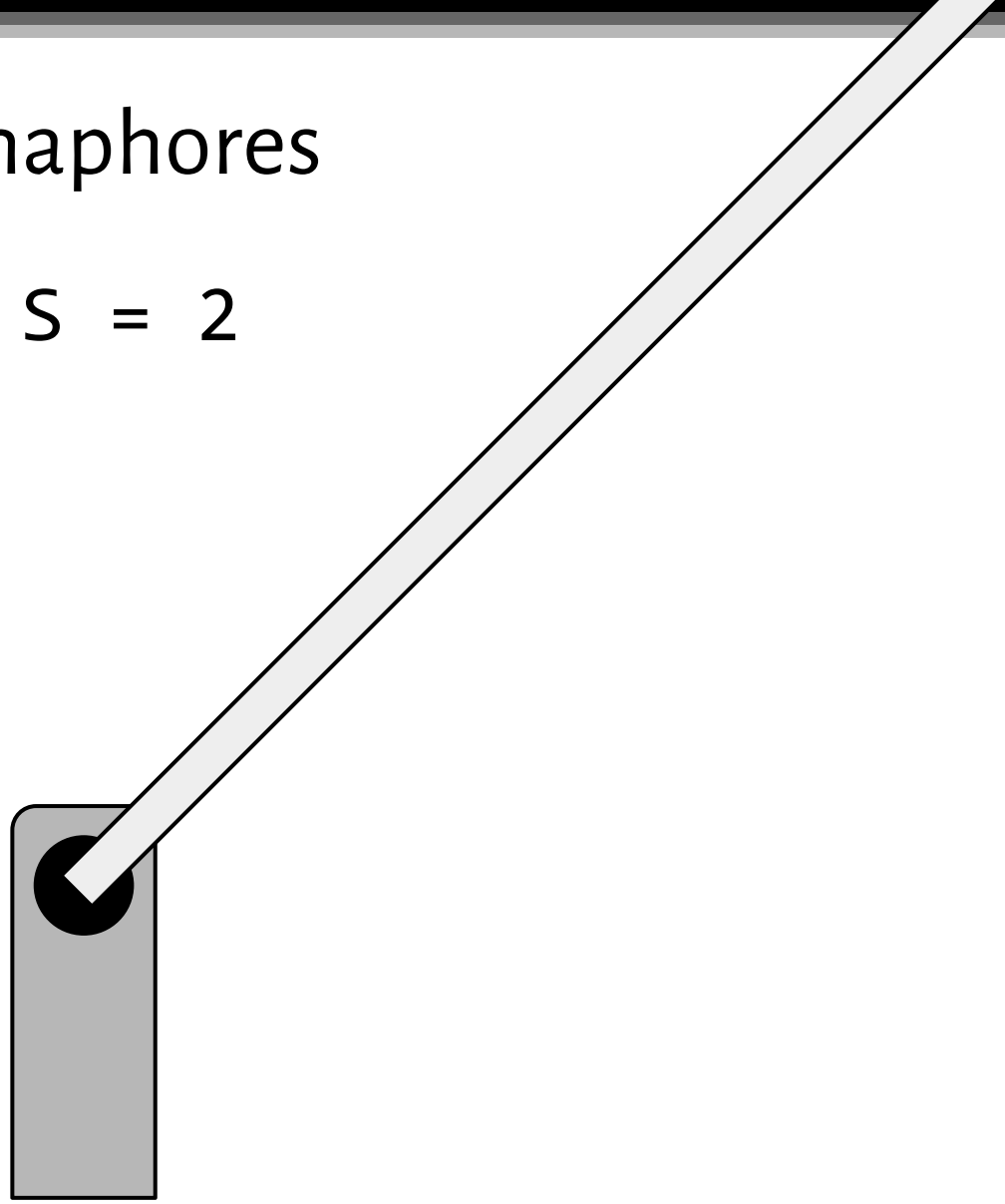
Zaimplementuj w asemblerze x86_64 **uogólniony semafor**, który może być używany w programach napisanych w języku C. Implementacja semafora ma się składać z trzech funkcji, które w C będą widziane jako:

```
void proberen(int32_t *semaphore, int32_t value);
void verhogen(int32_t *semaphore, int32_t value);
uint64_t proberen_time(int32_t *semaphore, int32_t value);
```

Parametr **semaphore** jest adresem zmiennej przechowującej wartość semafora.
Parametr **value** jest liczbą całkowitą z przedziału od 1 do $2^{31} - 1$.

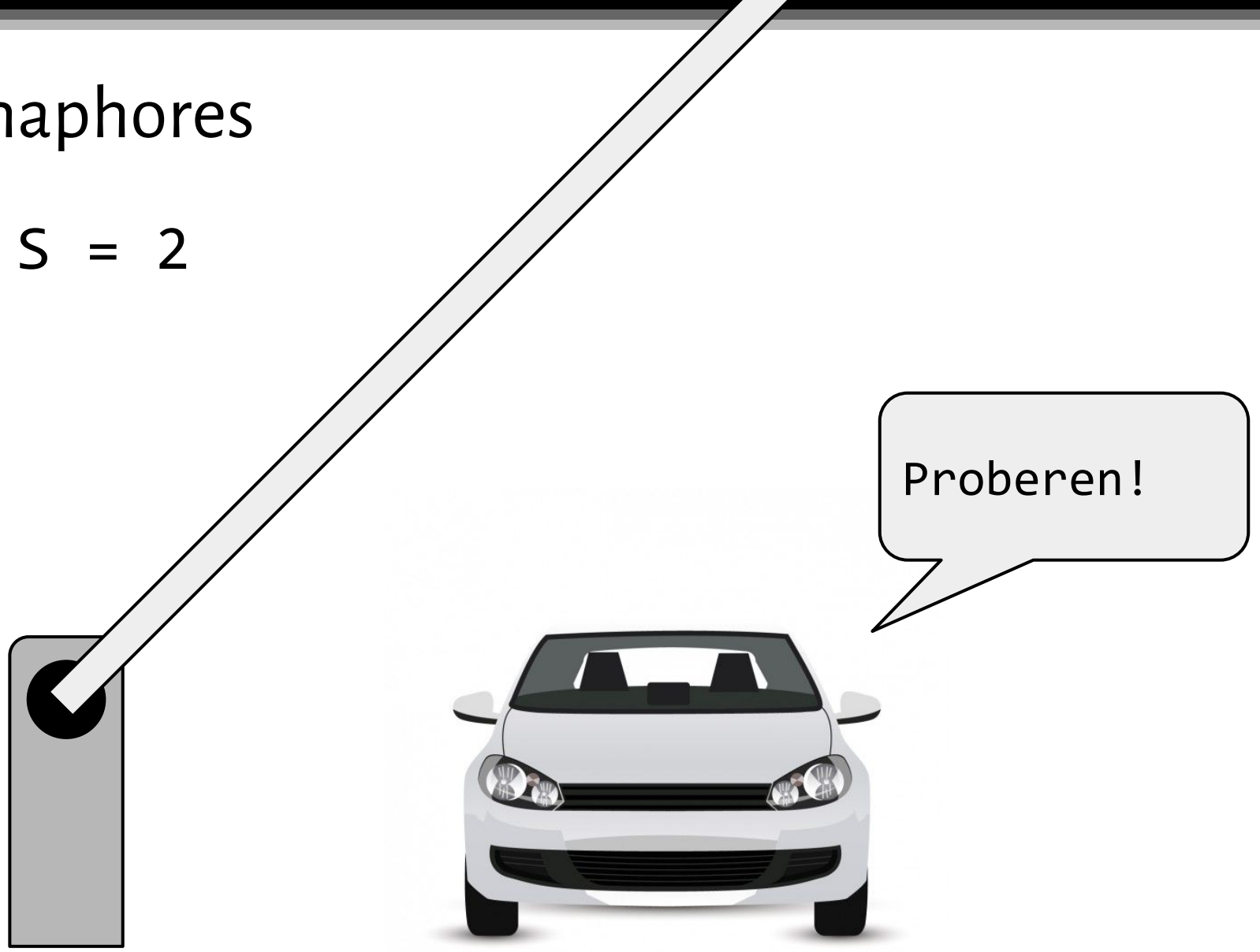
Semaphores

S = 2



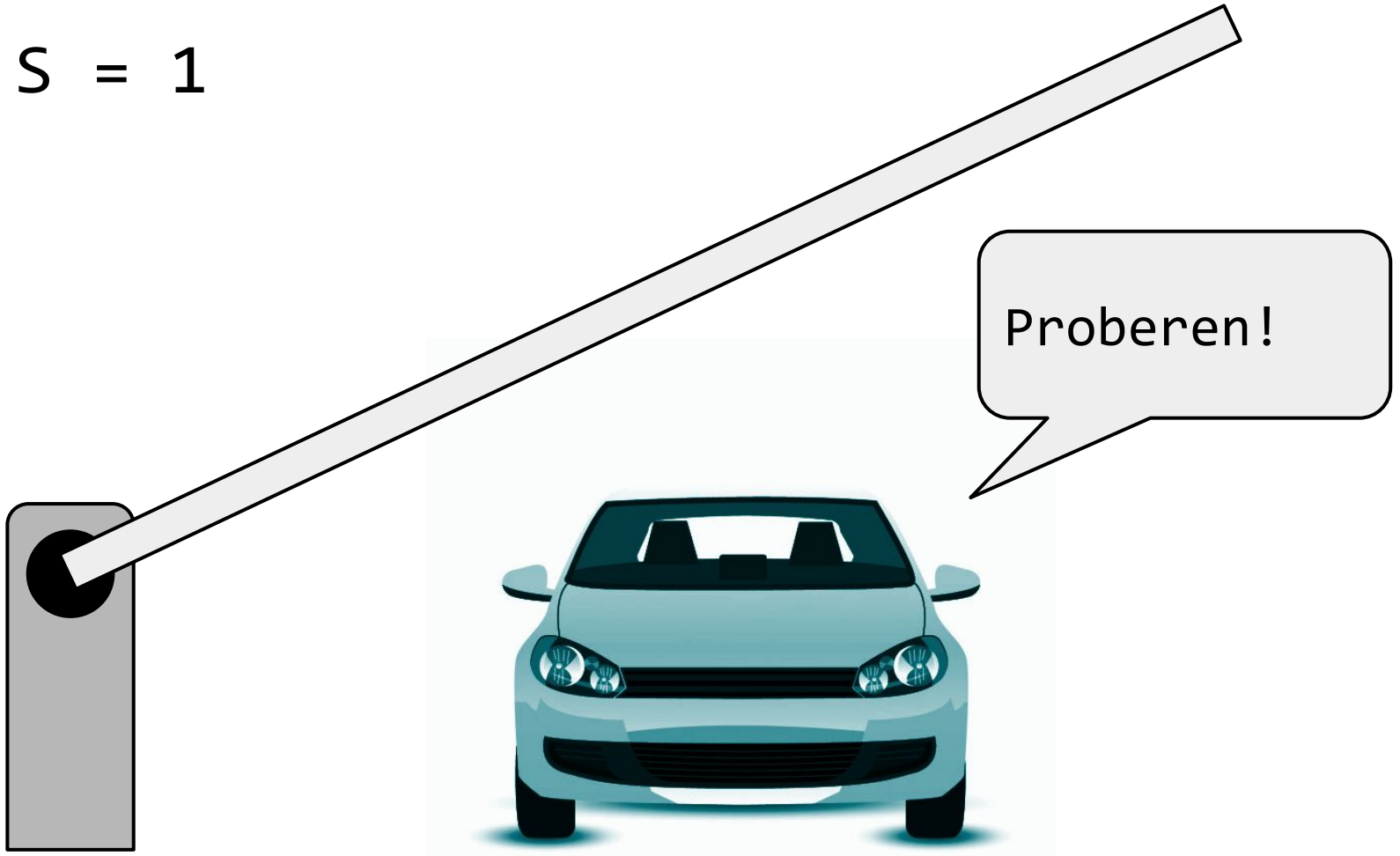
Semaphores

S = 2



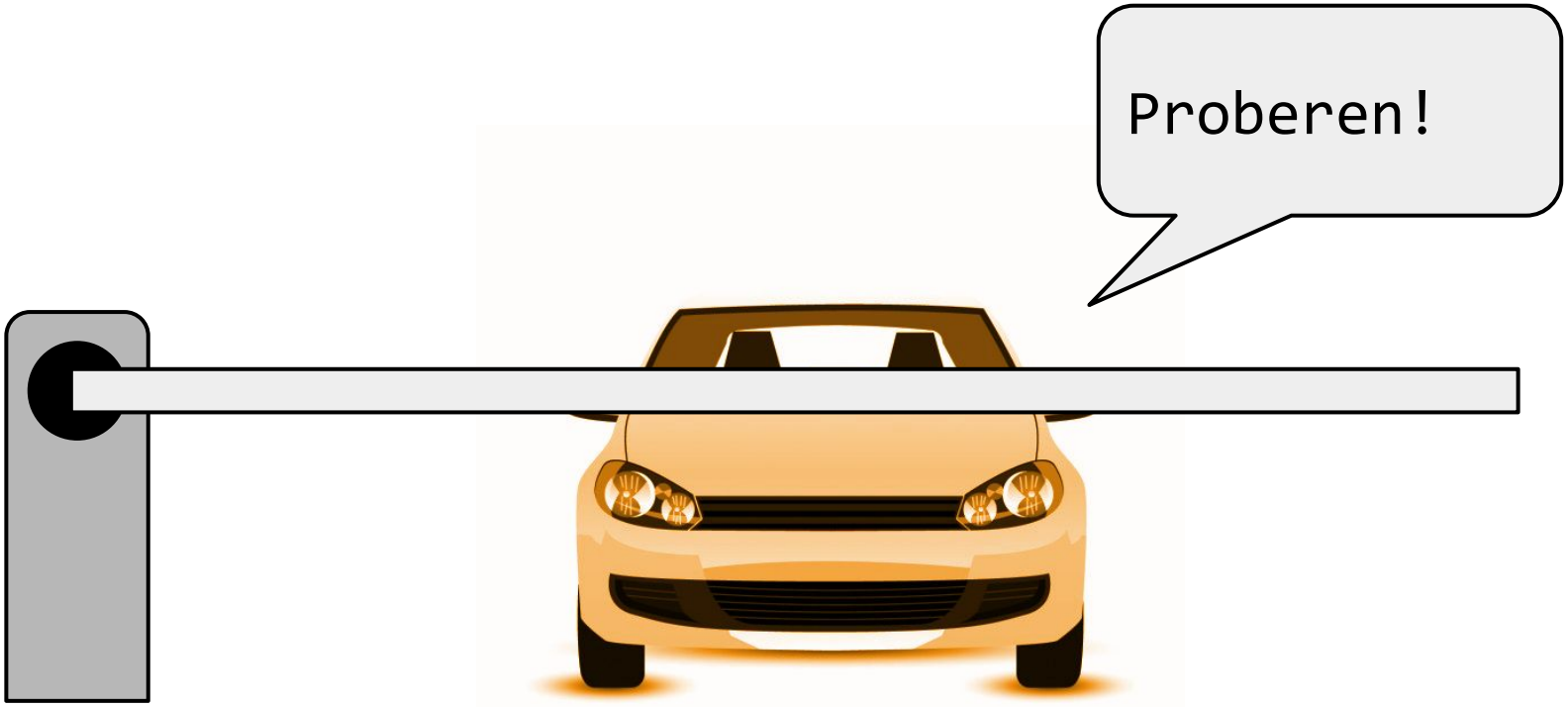
Semaphores

S = 1



Semaphores

```
S = 0
```

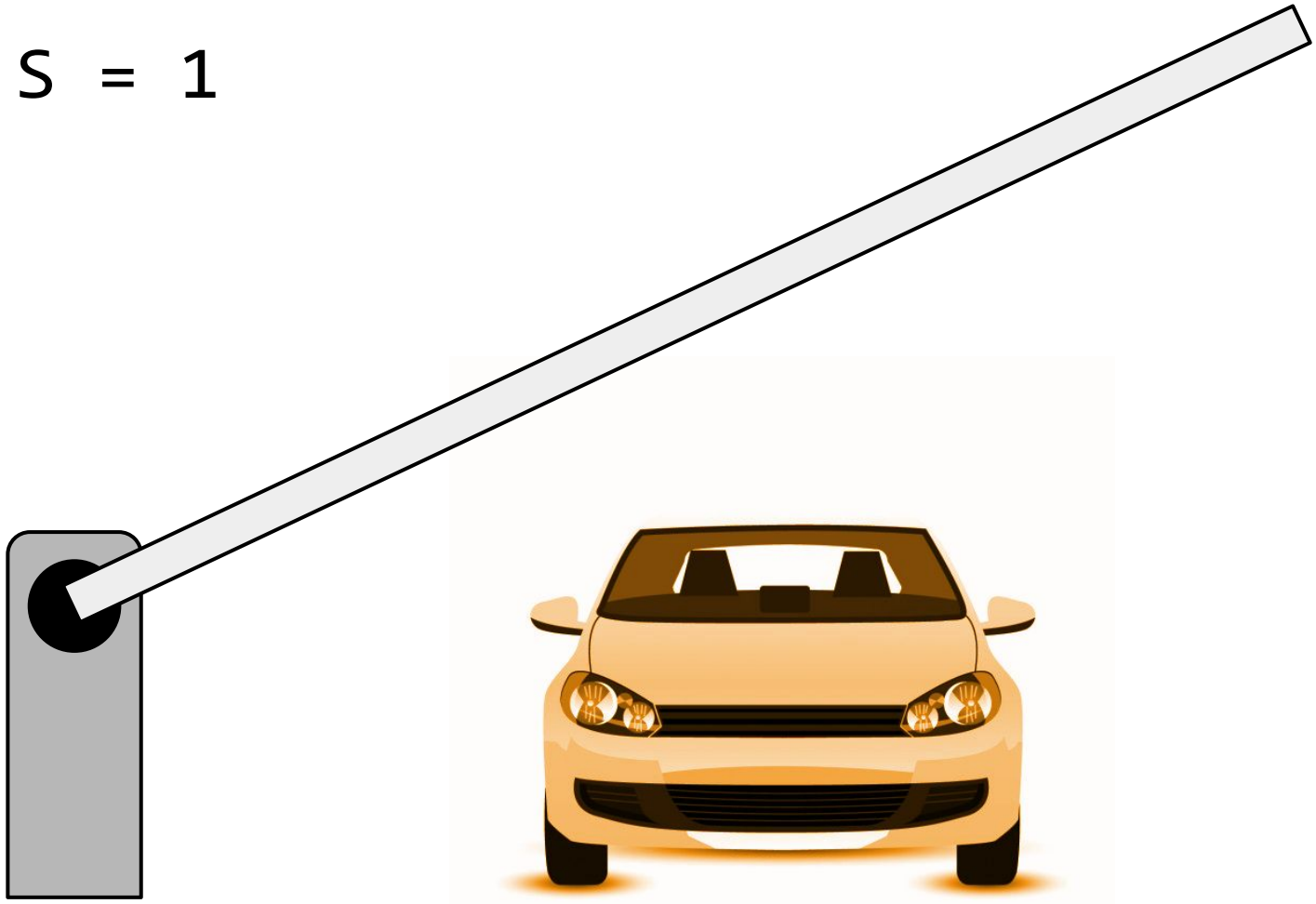


Semaphores

```
S = 0
```

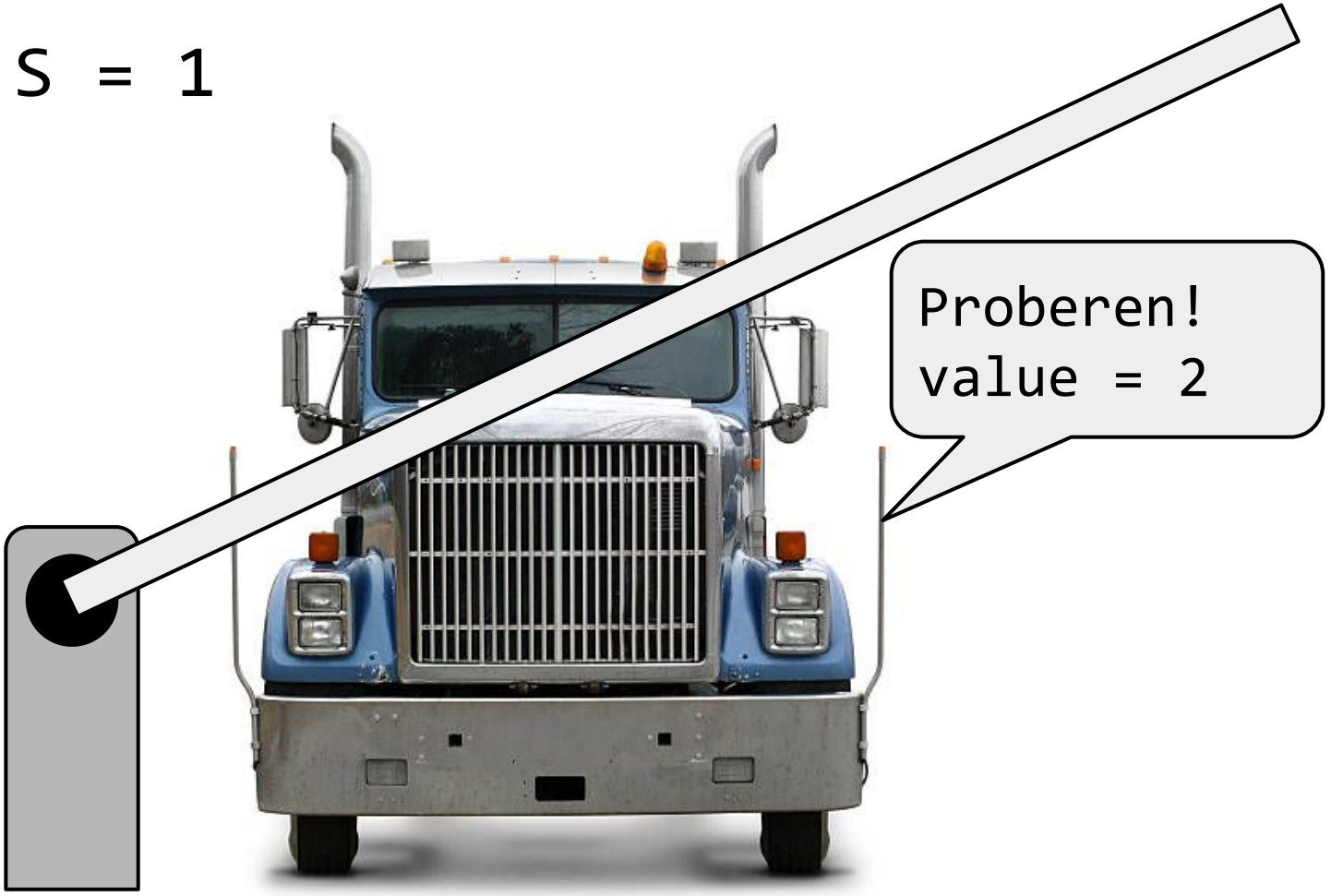
Semaphores

S = 1



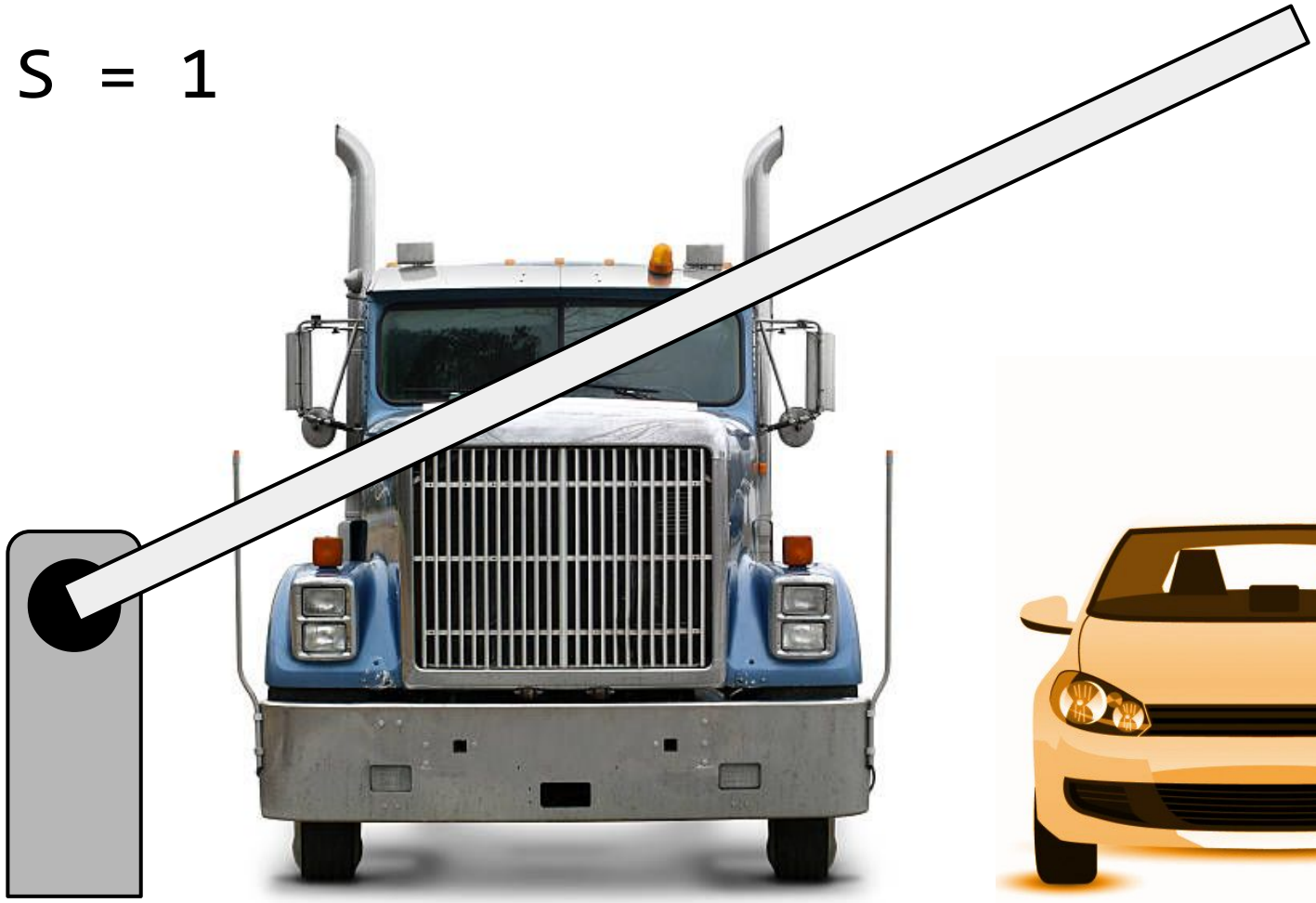
Semaphores

S = 1



Semaphores

S = 1



Assignment #2

Zaimplementuj w asemblerze x86_64 **uogólniony semafor**, który może być używany w programach napisanych w języku C. Implementacja semafora ma się składać z trzech funkcji, które w C będą widziane jako:

```
void proberen(int32_t *semaphore, int32_t value);
void verhogen(int32_t *semaphore, int32_t value);
uint64_t proberen_time(int32_t *semaphore, int32_t value);
```

Parametr **semaphore** jest adresem zmiennej przechowującej wartość semafora.
Parametr **value** jest liczbą całkowitą z przedziału od 1 do $2^{31} - 1$.

Assignment #2

Zaimplementuj w asemblerze x86_64 **uogólniony semafor**, który może być używany w programach napisanych w języku C. Implementacja semafora ma się składać z trzech funkcji, które w C będą widziane jako:

```
void proberen(int32_t *semaphore, int32_t value);
void verhogen(int32_t *semaphore, int32_t value);
uint64_t proberen_time(int32_t *semaphore, int32_t value);
```

Parametr **semaphore** jest adresem zmiennej przechowującej wartość semafora. Parametr **value** jest liczbą całkowitą z przedziału od 1 do $2^{31} - 1$.

Funkcja **proberen_time** woła funkcję proberen z tymi samymi argumentami, które dostała, i mierzy (zwraca) czas przebywania sterowania w proberen. W tym celu wała dwukrotnie funkcję:

```
uint64_t get_os_time(void);
```

Assignment #2

How to implement get_os_time?

https://stackoverflow.com/questions/8594277/clock-precision-in-time-h

Makefile

```
life: lifemain.c life.o
    gcc -o life lifemain.c life.o

%.o: %.asm
    nasm -f elf64 -F dwarf -g $<
```