Systems Software HS15

Lab Exercises

Claudio Mura
claudio@ifi.uzh.ch

Practical notes on PThreads Mutexes

Mutual Exclusion with Pthreads

- Main tool: *mutex lock*
  - two states: locked or unlocked
  - locking/unlocking operations are atomic
- Pthread implementation: `pthread_mutex_t`
  - initialization: `pthread_mutex_init( attr )`
    - alternative: `pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;`
  - lock/unlock operations:

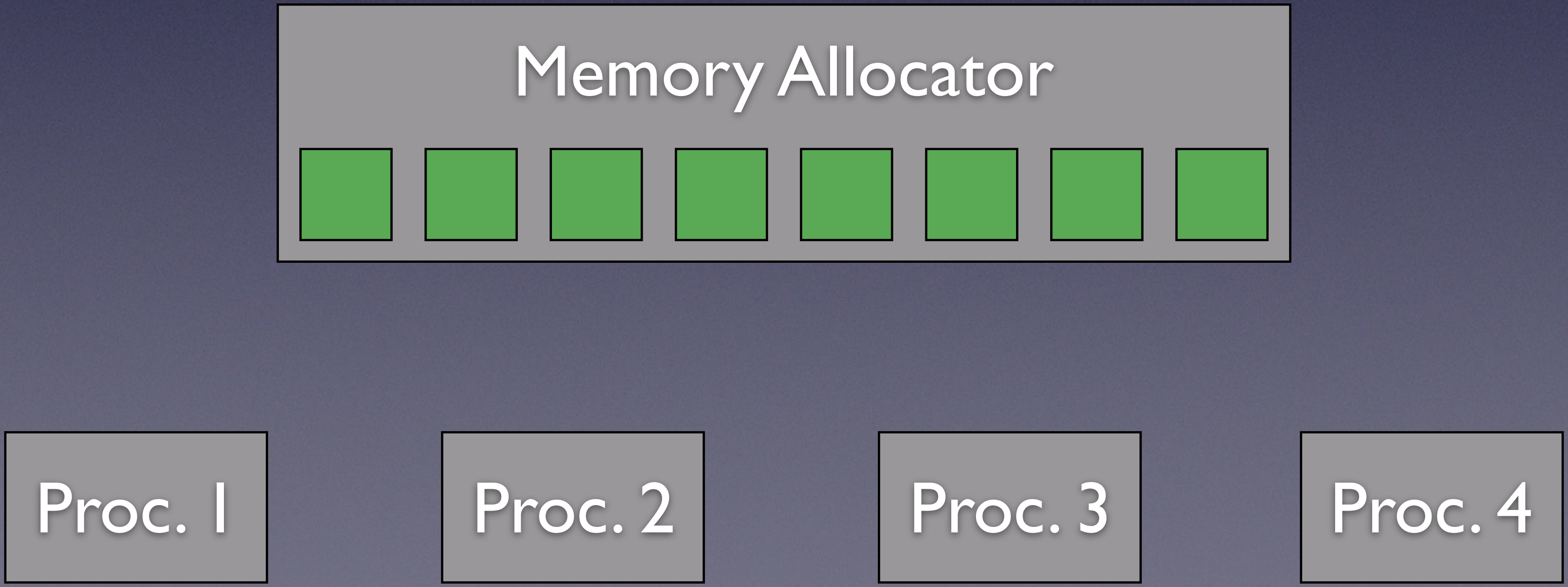
    `pthread_mutex_lock()` / `pthread_mutex_unlock()`
  - unlock must be called by the thread currently owning the lock!
- Note: `pthread_mutex_lock()` is blocking!
  - alternative: `pthread_mutex_trylock()`
  - returns immediately if mutex is locked

Exercise 6

Simulating the Behaviour of a Memory Allocator in a Multi-process Environment

Our Toy Environment

- Available memory divided into *B* blocks of fixed size
  - managed by a memory allocator
  - a block can be unallocated or assigned to a process
- *P* processes running on the system
  - need some extra memory blocks to perform computation
  - request blocks to the memory allocator, release them when done

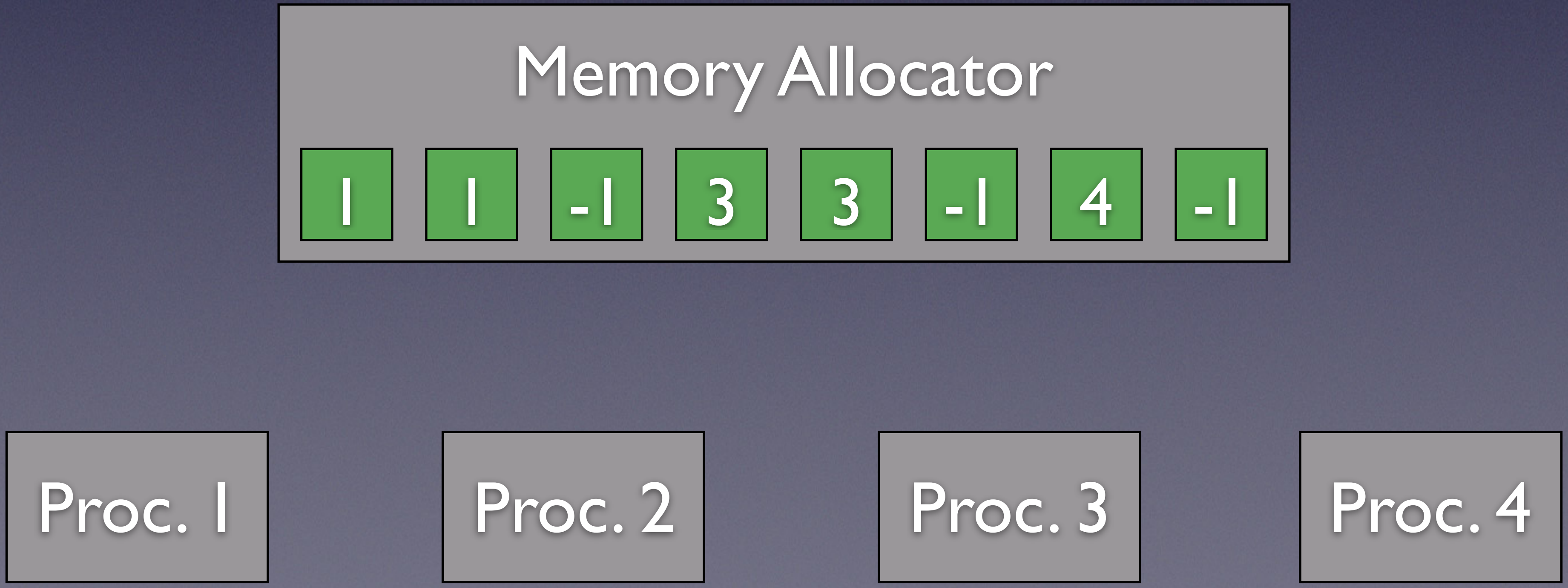


Behaviour of a Process

- Every *T* msecs performs some compute-intensive task
  - requests blocks (no. depends on the task) from the allocator
  - if allocation ok, spend some time computing, then release blocks
  - if allocation fails (no blocks available), abort and do nothing
- Spends rest of the time sleeping

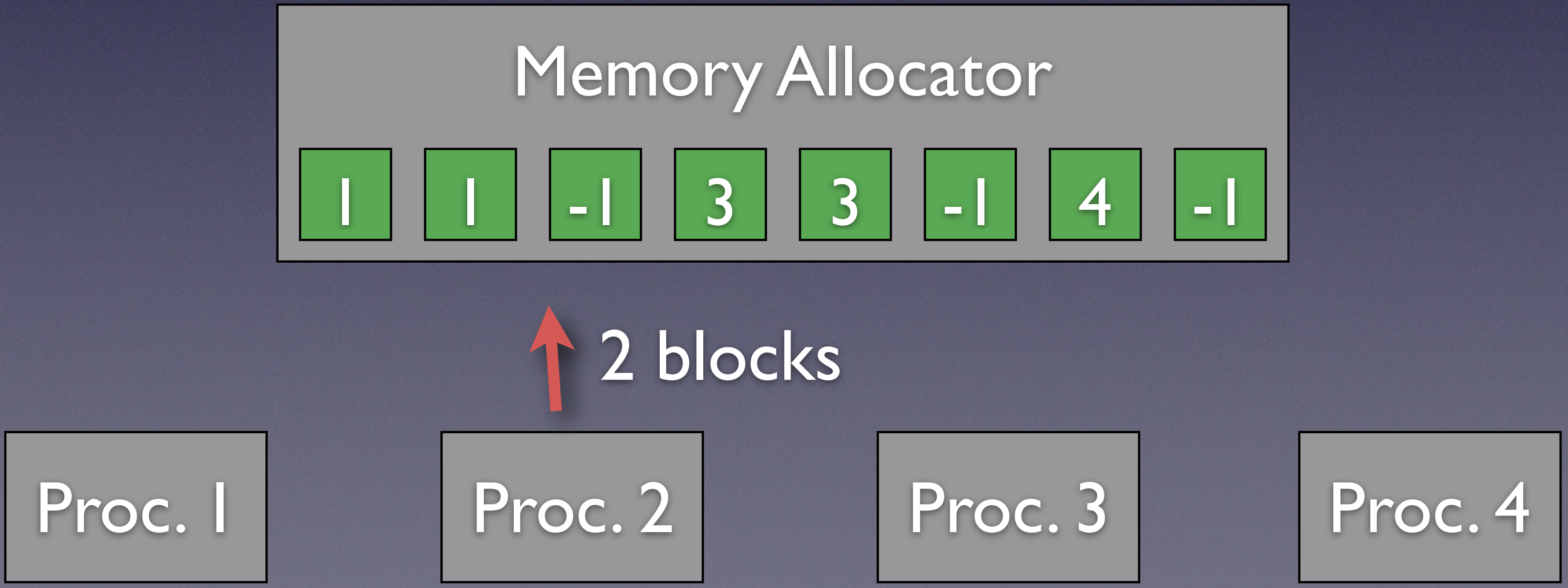
Behaviour of the Allocator

- Keeps a list with information on each block
  - allocation status: free (-1) or allocated to a process (id of process)
- Answers allocation requests of processes
  - check if the number of blocks requested is available
  - if yes, assigns blocks and marks them; if not, does nothing; in both cases, notifies the process of the outcome



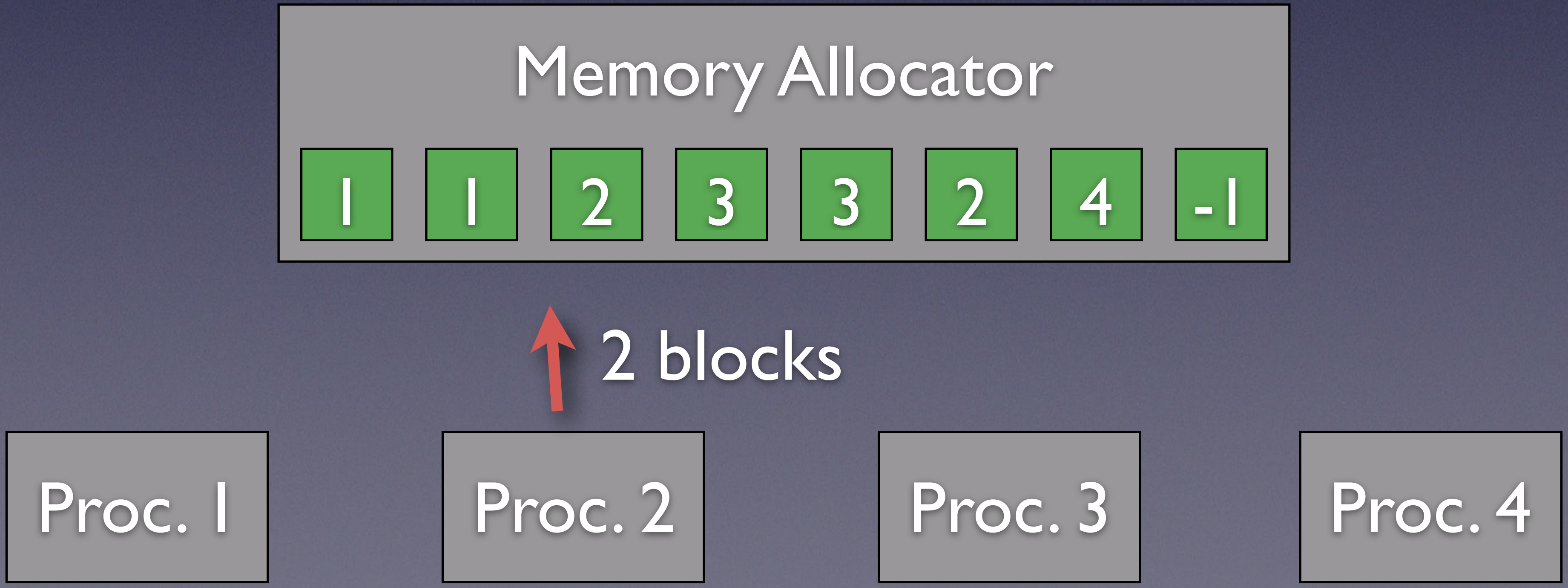
Behaviour of the Allocator

- Keeps a list with information on each block
  - allocation status: free (-1) or allocated to a process (id of process)
- Answers allocation requests of processes
  - check if the number of blocks requested is available
  - if yes, assigns blocks and marks them; if not, does nothing; in both cases, notifies the process of the outcome



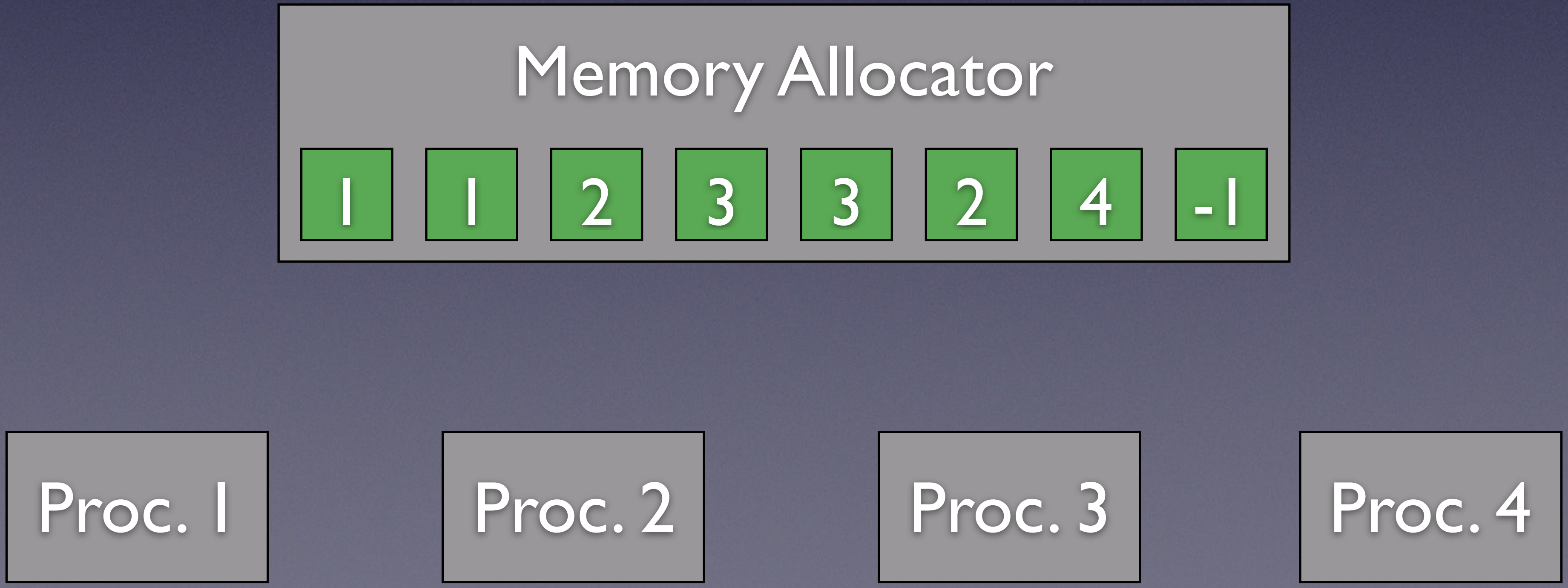
Behaviour of the Allocator

- Keeps a list with information on each block
  - allocation status: free (-1) or allocated to a process (id of process)
- Answers allocation requests of processes
  - check if the number of blocks requested is available
  - if yes, assigns blocks and marks them; if not, does nothing; in both cases, notifies the process of the outcome



Behaviour of the Allocator

- Keeps a list with information on each block
  - allocation status: free (-1) or allocated to a process (id of process)
- Answers allocation requests of processes
  - check if the number of blocks requested is available
  - if yes, assigns blocks and marks them; if not, does nothing; in both cases, notifies the process of the outcome



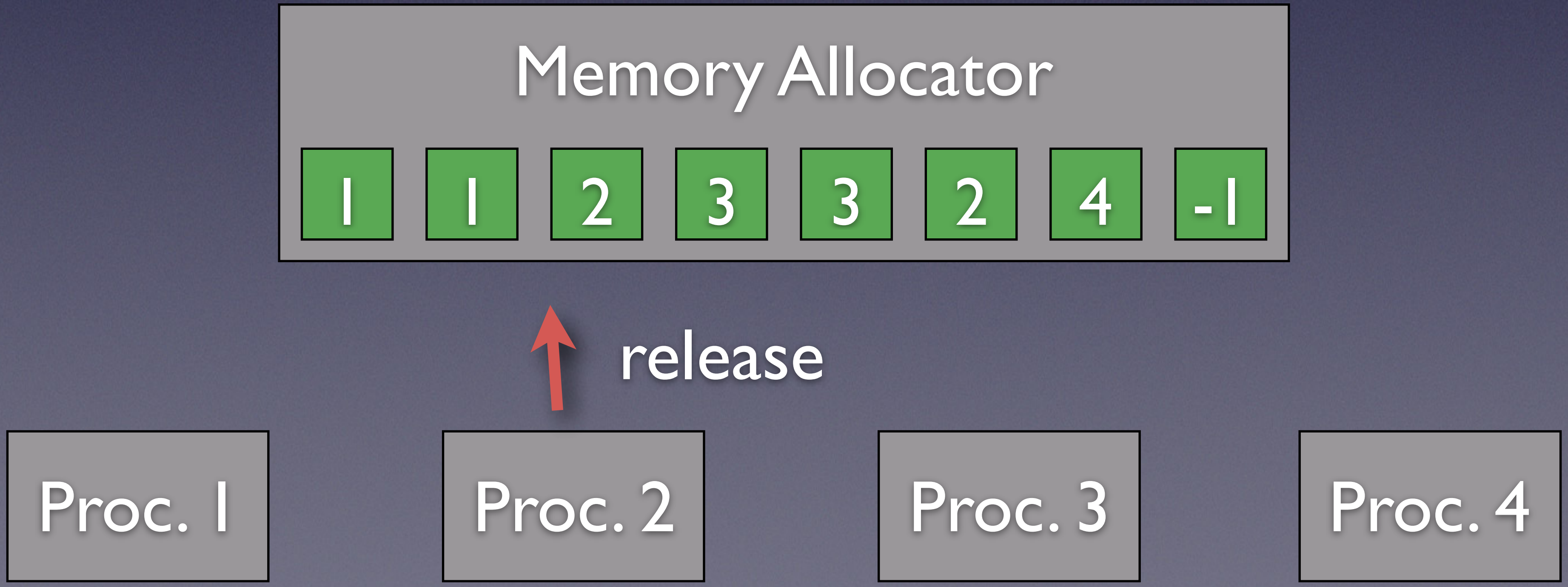
Memory Allocator

1	1	2	3	3	2	4	-1

Proc. 1 Proc. 2 Proc. 3 Proc. 4

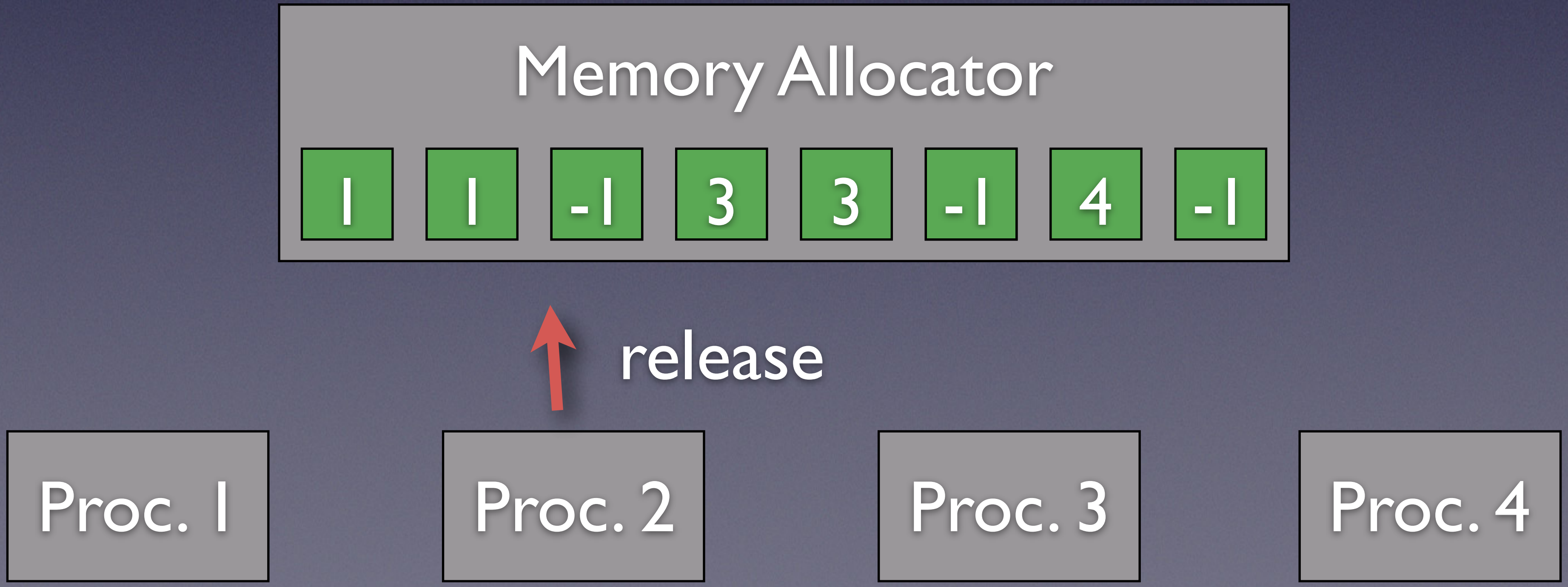
Behaviour of the Allocator

- Keeps a list with information on each block
  - allocation status: free (-1) or allocated to a process (id of process)
- Answers allocation requests of processes
  - check if the number of blocks requested is available
  - if yes, assigns blocks and marks them; if not, does nothing; in both cases, notifies the process of the outcome



Behaviour of the Allocator

- Keeps a list with information on each block
  - allocation status: free (-1) or allocated to a process (id of process)
- Answers allocation requests of processes
  - check if the number of blocks requested is available
  - if yes, assigns blocks and marks them; if not, does nothing; in both cases, notifies the process of the outcome



Simulation

- Implementing behaviour of processes:
  - use one **thread** (PThreads) for every process
    - each thread identified by an id
  - sleep time after work (attempt): T ms, T random in [T_{min} , T_{max}]
  - number of blocks needed: W, random in [W_{min} , W_{max}]
  - simulate computing: sleep for T' ms, T' random in [T_{min} , W x T_{max}]

```
do
   generate random number W in the range [ $W_{min}$ , $W_{max}$ ]
   request allocation of W blocks of memory
   if allocation successful
      sleep for some random time T' in the range [ $T_{min}$ , W x $T_{max}$ ]
      release allocated blocks
   end
   sleep for some random time T in the range [ $T_{min}$ , $T_{max}$ ]
while simulation is running
```

Simulation

- Implementing behaviour of allocator:
  - implement the list of block entries as `std::vector< int >`
    - each entry: -1 (block not allocated) or id of thread using the block
  - provide 2 functions/methods to allocate and release memory blocks
  - ensure that accesses to the vector are properly synchronized!
    - use a `pthread_mutex_t` (and nothing else!)

Additional Details

- Command line arguments (in order):
  - # processes *P* in the simulated system (i.e. threads to be created)
  - # memory blocks *B* (i.e. size of the `std::vector< int >`)
  - values W_{min} , W_{max} , T_{min} , T_{max}
- Making threads sleep: use `nanosleep()` function
- Provide console output to show how simulation evolves
  - warning: terminal is a shared resource! use mutex
- Terminate simulation when user presses 'e' + return
  - make sure threads terminate gracefully