Smith College
Computer Science

# CSC231—Assembly

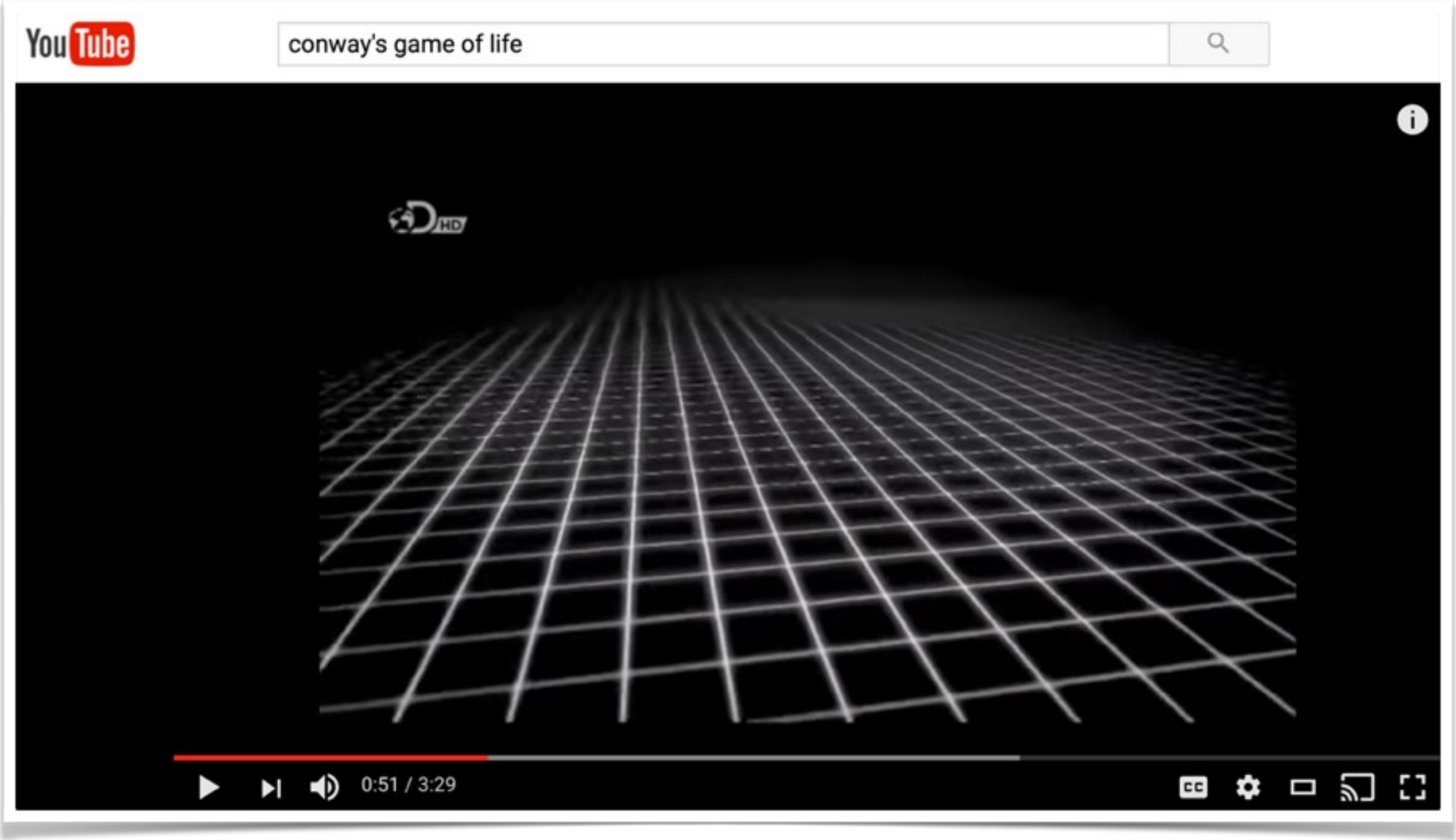
Week #9 — Spring 2017

Dominique Thiébaut
dthiebaut@smith.edu

# 2 Videos to Watch at a Later Time…

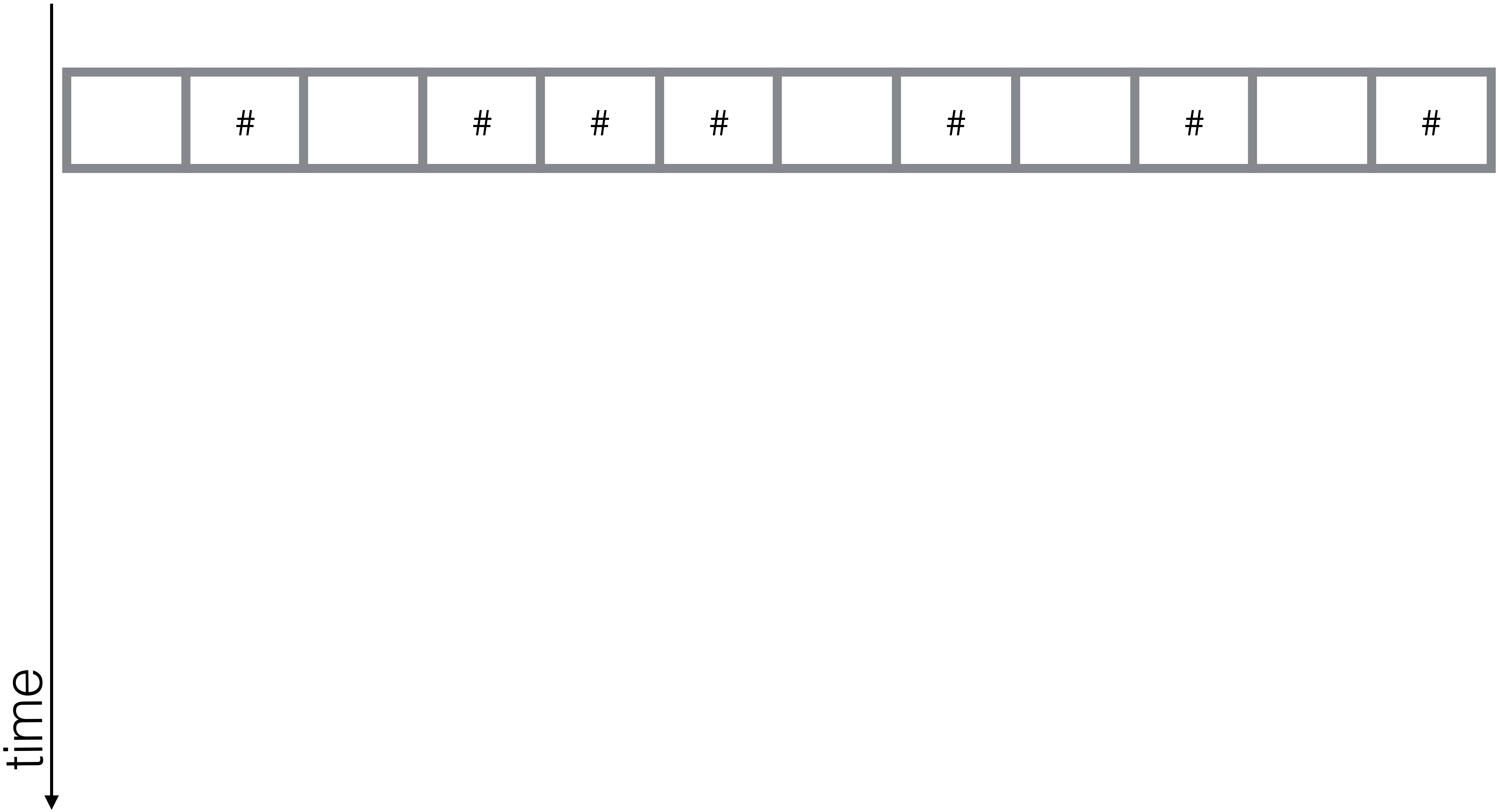
https://www.youtube.com/watch?v=FdMzngWchDk

https://www.youtube.com/watch?v=k2IZ1qsx4CM



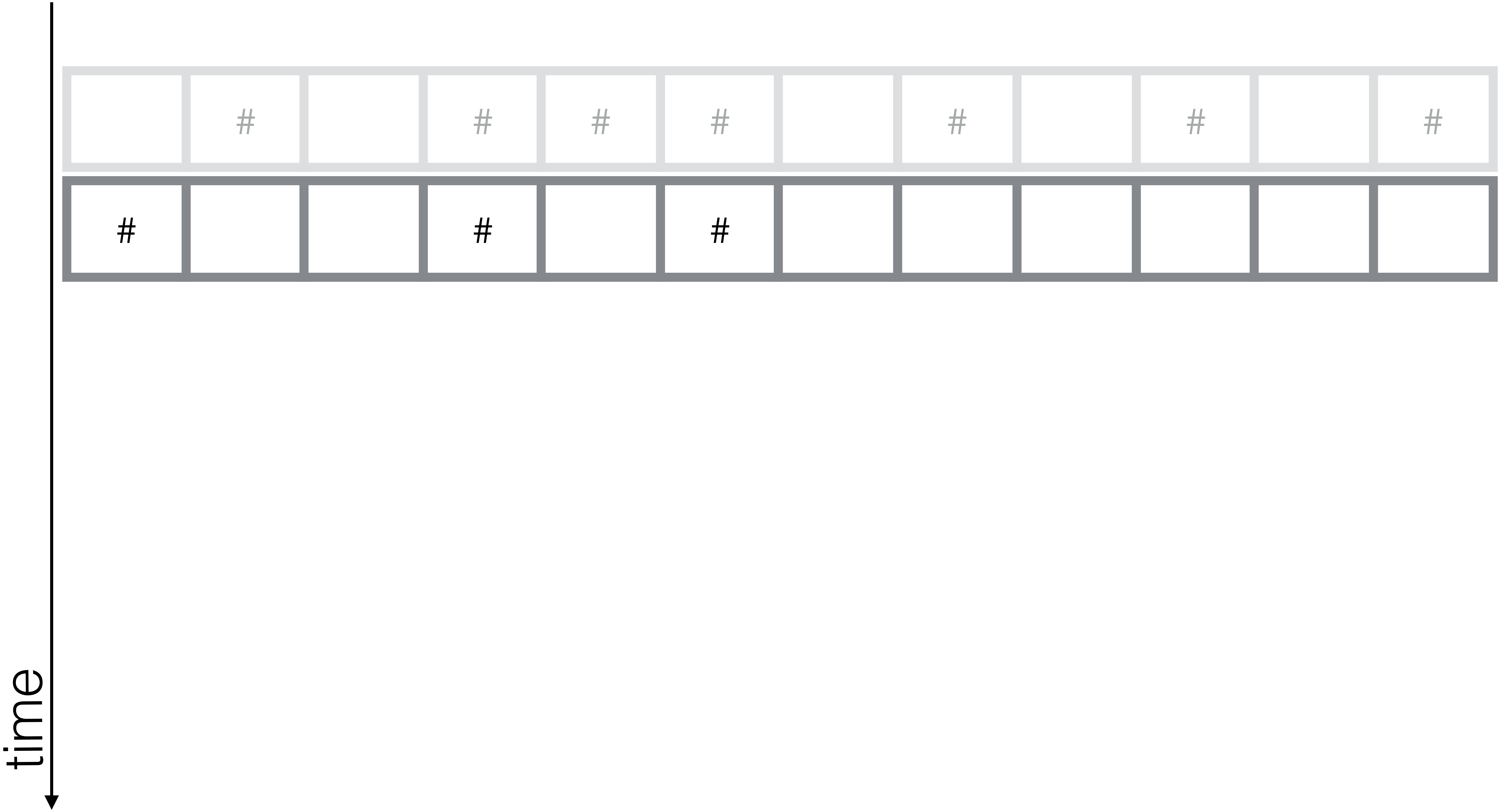
https://www.youtube.com/watch?v=CgOcEZinQ2I

# A 1-D Version



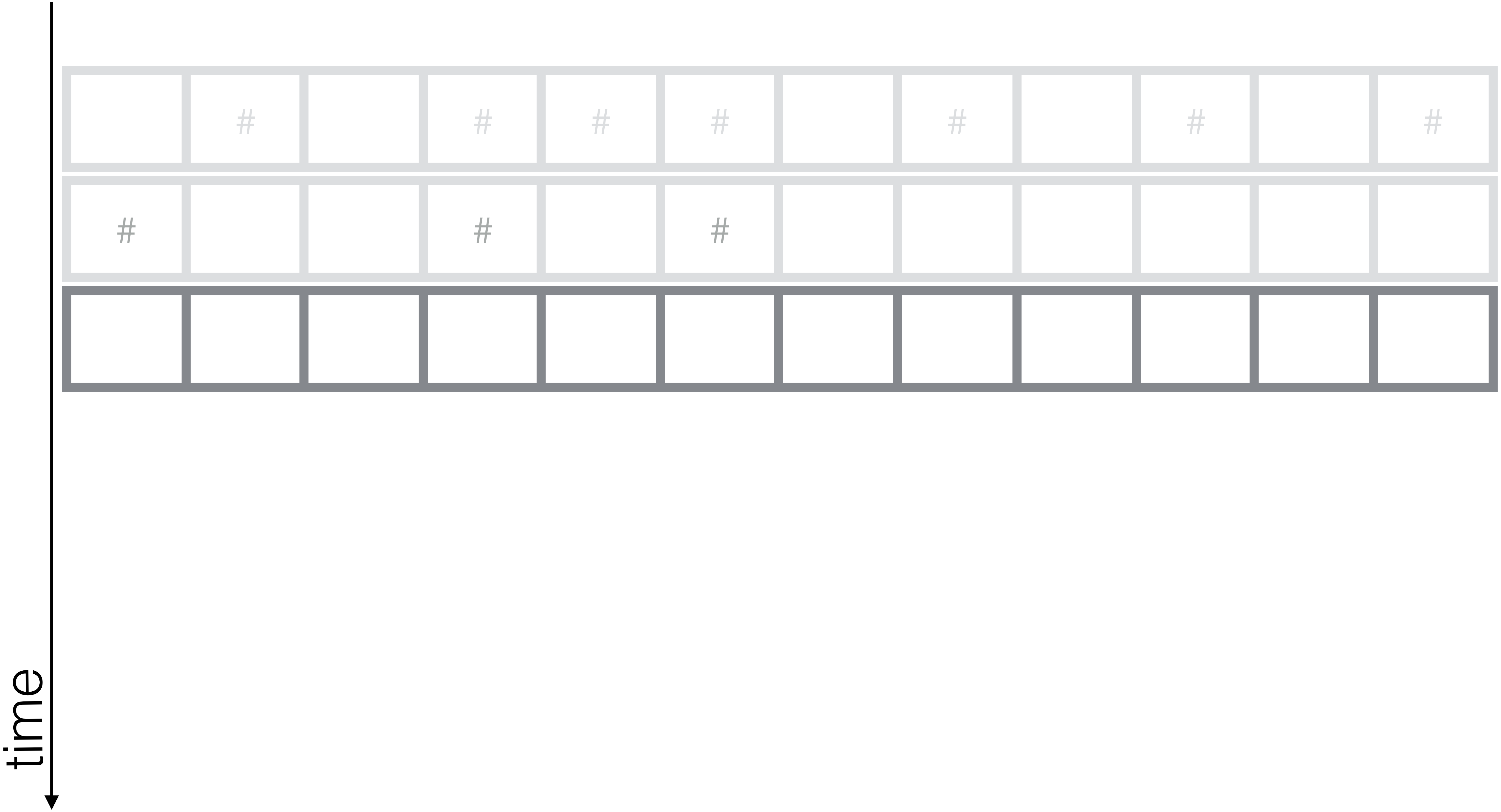
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | # | | # | # | # | | # | | # | | # |

# A 1-D Version



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | # | | # | # | # | | # | | # | | # |
| # | | | # | | # | | | | | | |

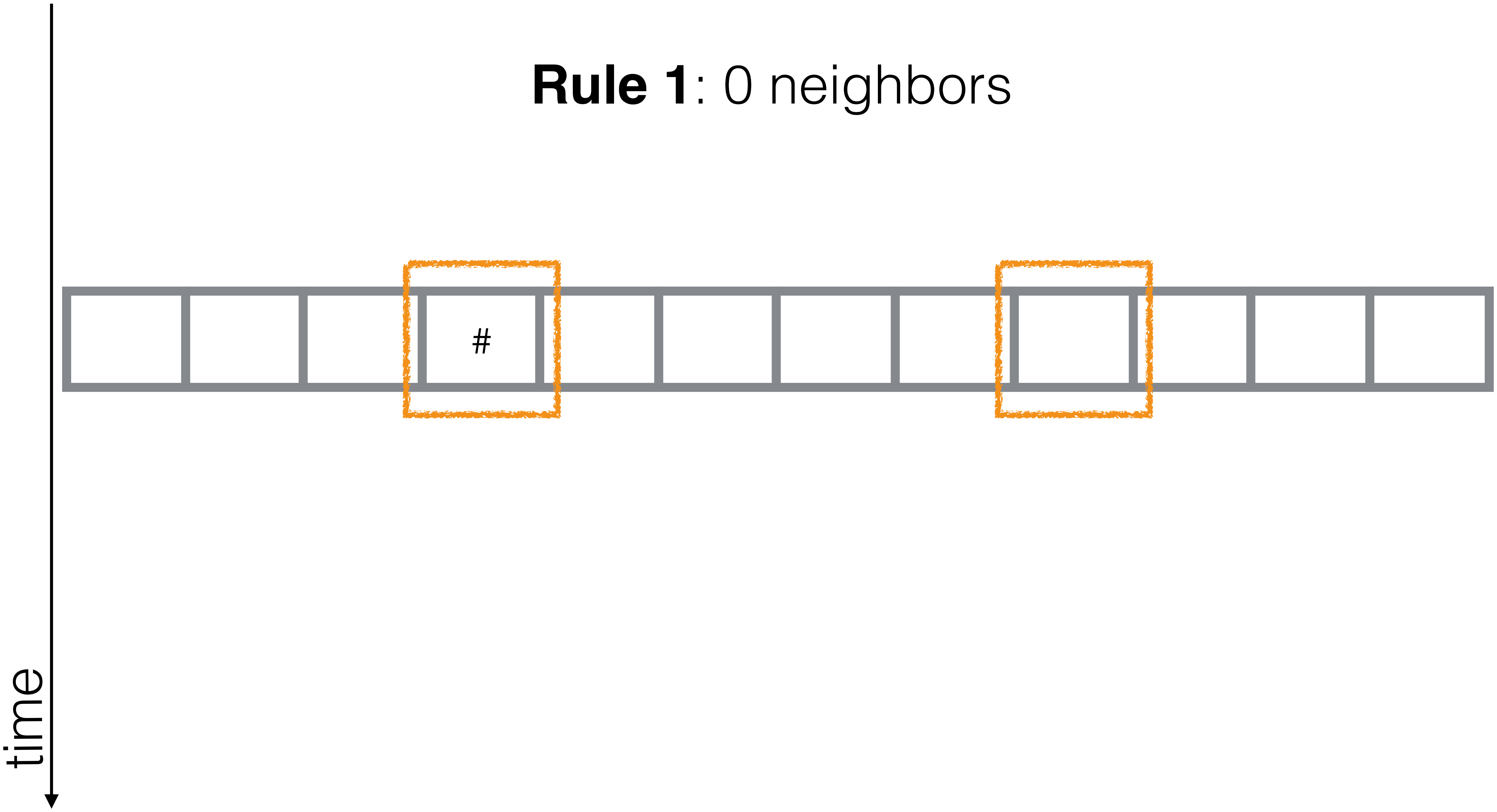
# A 1-D Version



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | # | | # | # | # | | # | | # | | # |
| # | | | # | | # | | | | | | |
| | | | | | | | | | | | |

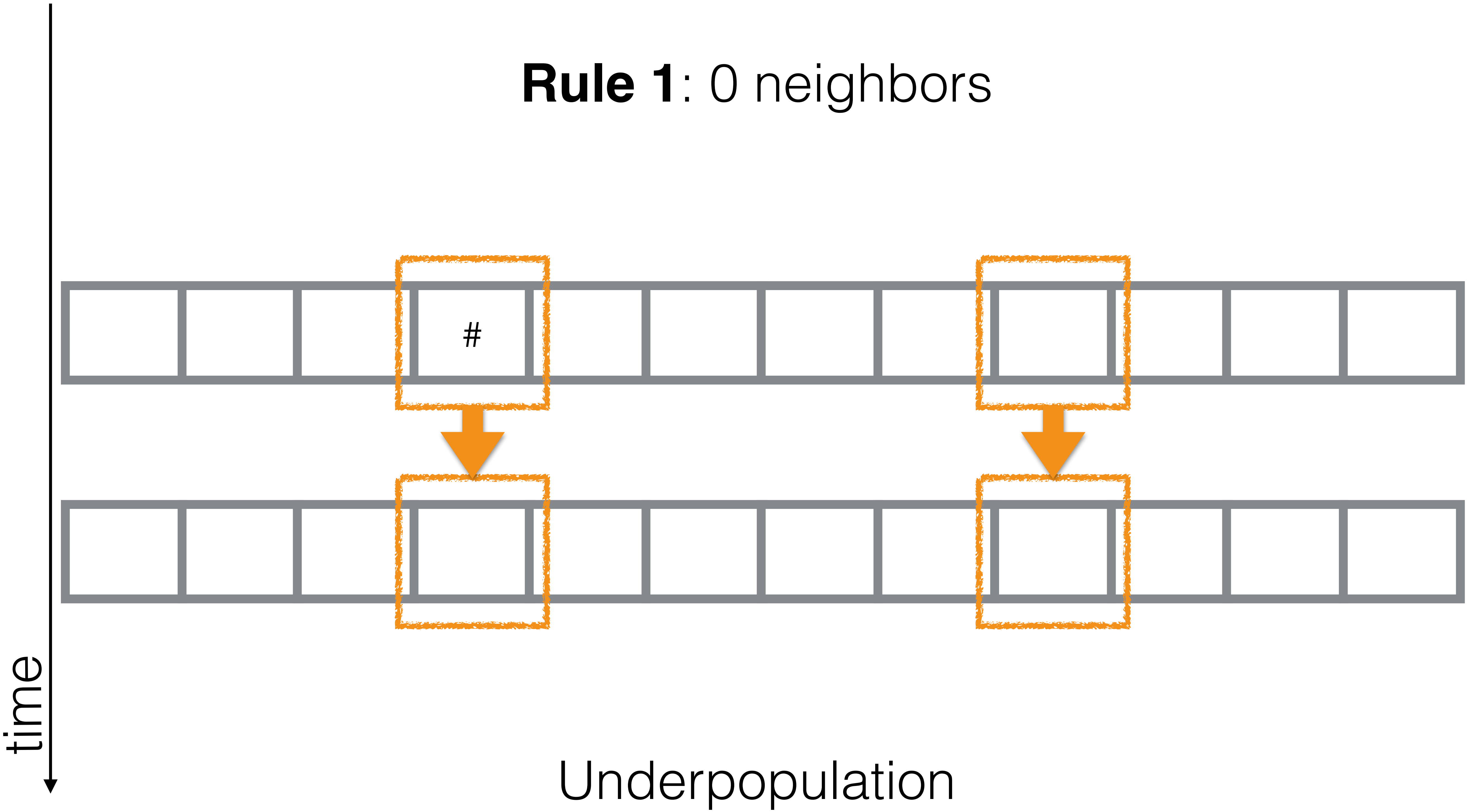
# Rules of Life

**Rule 1**: 0 neighbors



# Rules of Life

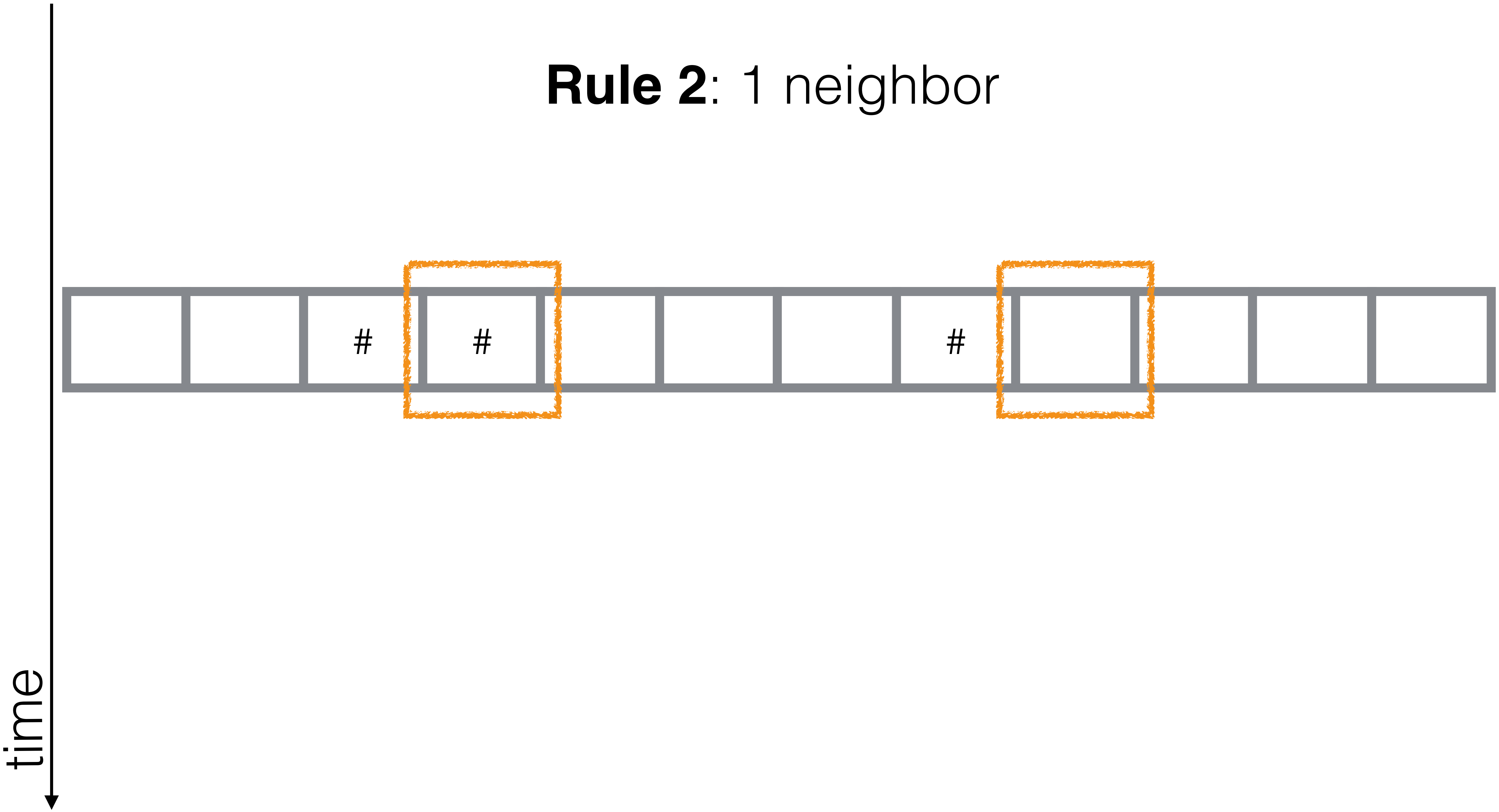
**Rule 1**: 0 neighbors



Underpopulation

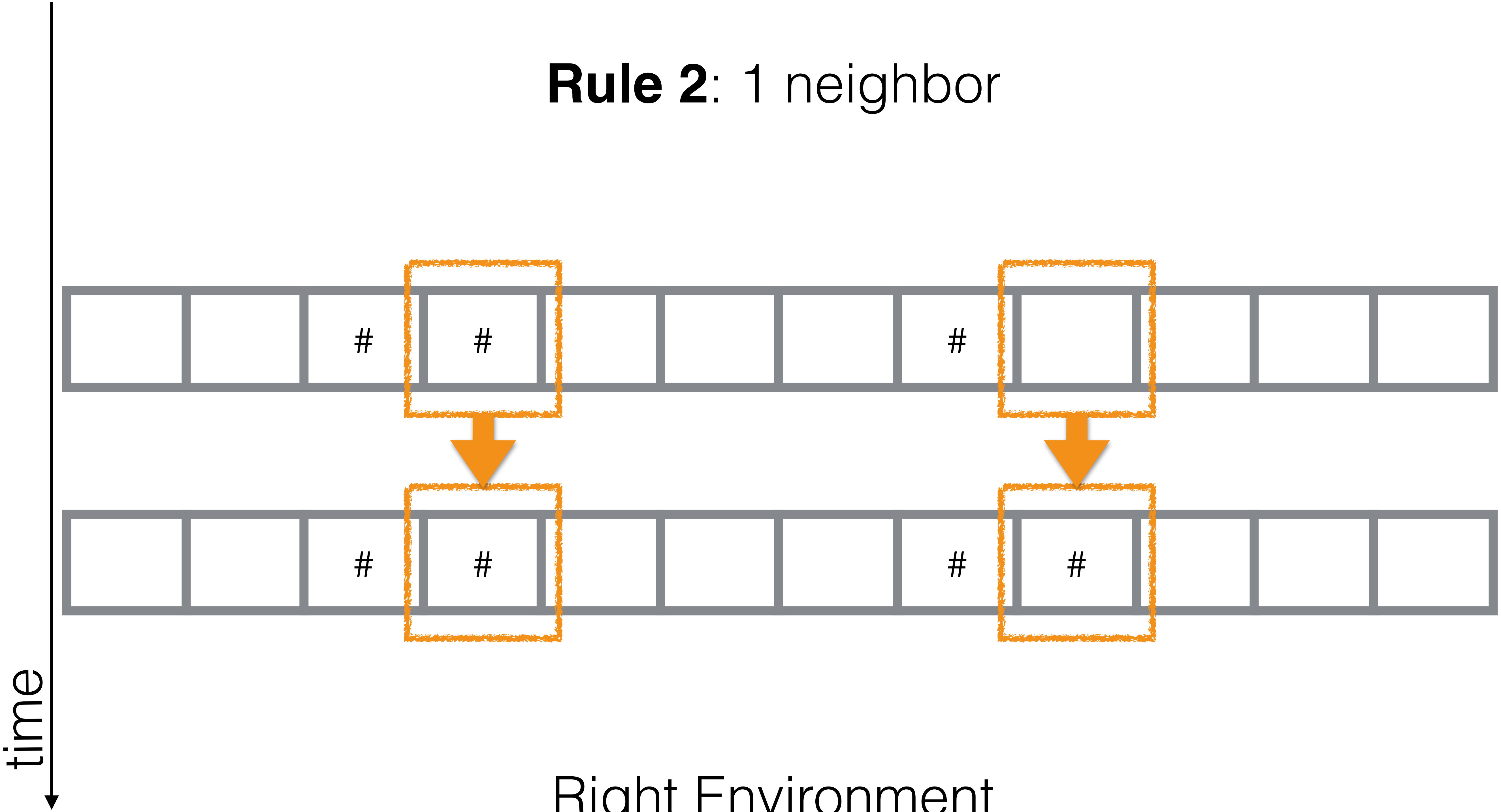
# Rules of Life

**Rule 2**: 1 neighbor



# Rules of Life

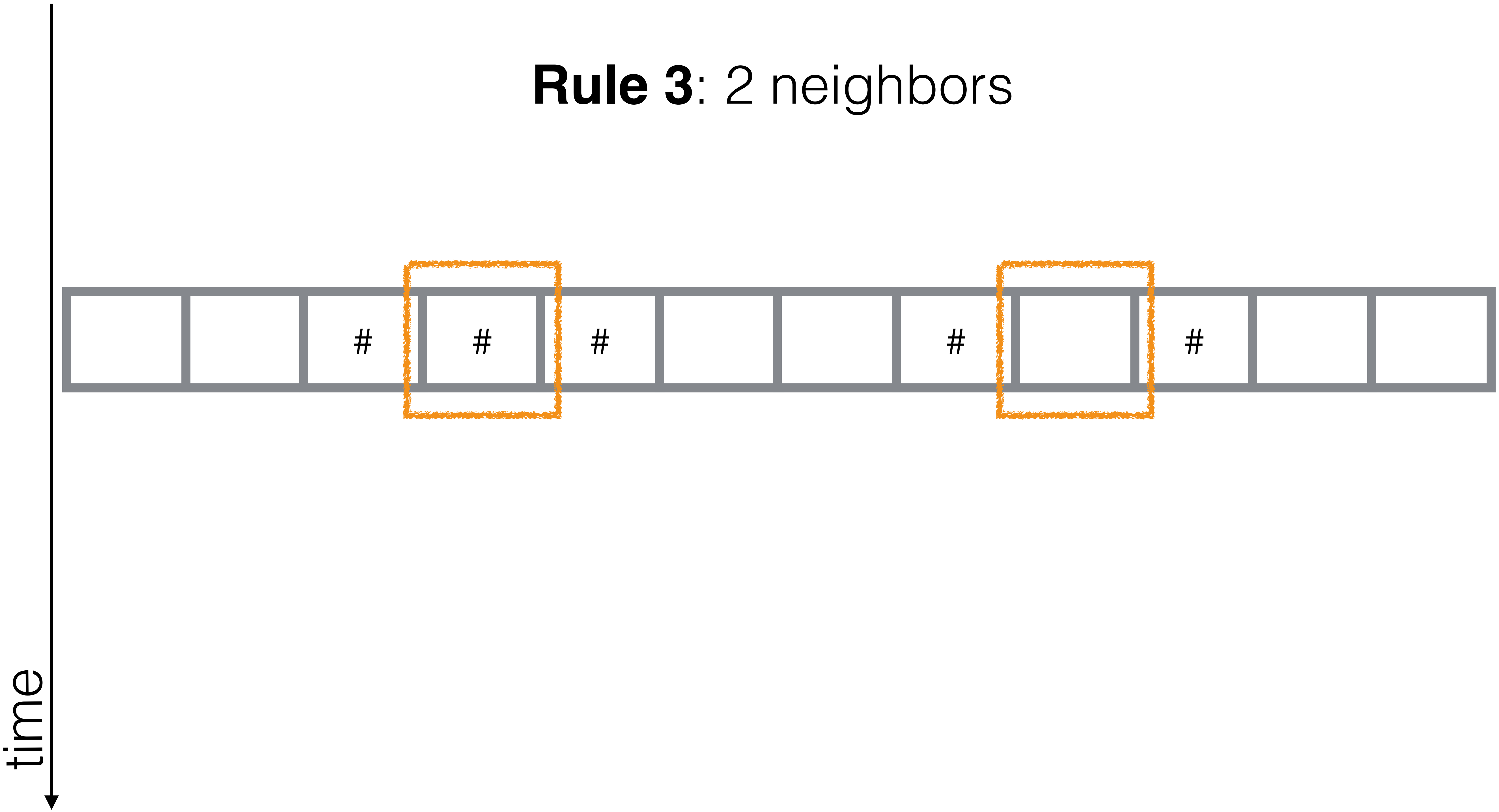
## **Rule 2**: 1 neighbor



Right Environment

# Rules of Life

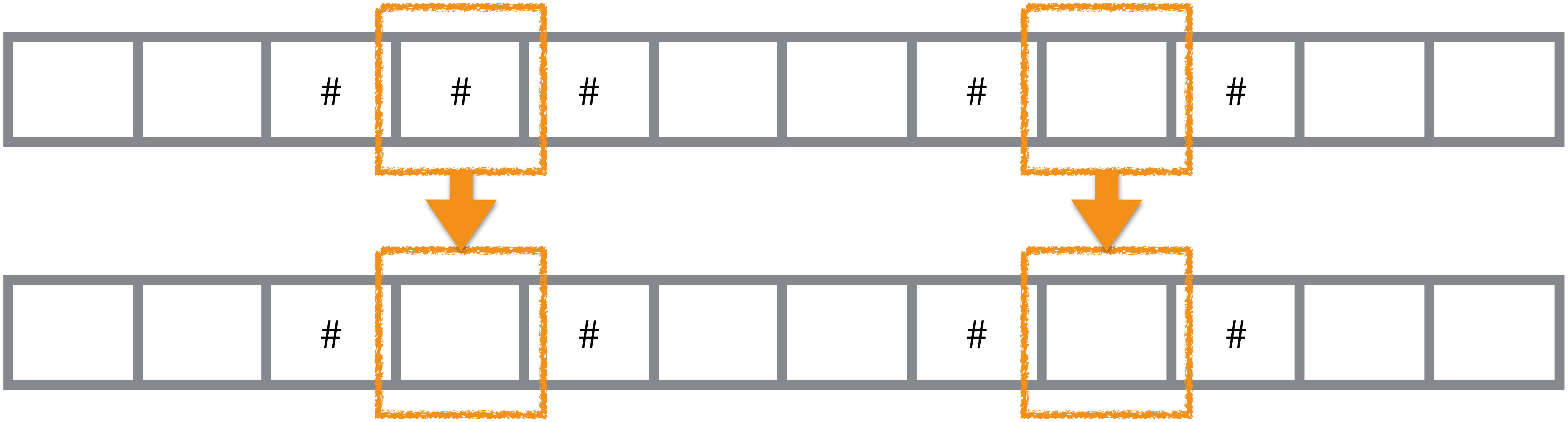
**Rule 3**: 2 neighbors



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | # | # | # | | | # | | # | | |

# Rules of Life

**Rule 3**: 2 neighbors



|  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | # | # | # |  |  | # |  | # |  |  |
|  |  | # |  | # |  |  | # |  | # |  |  |

Overpopulation

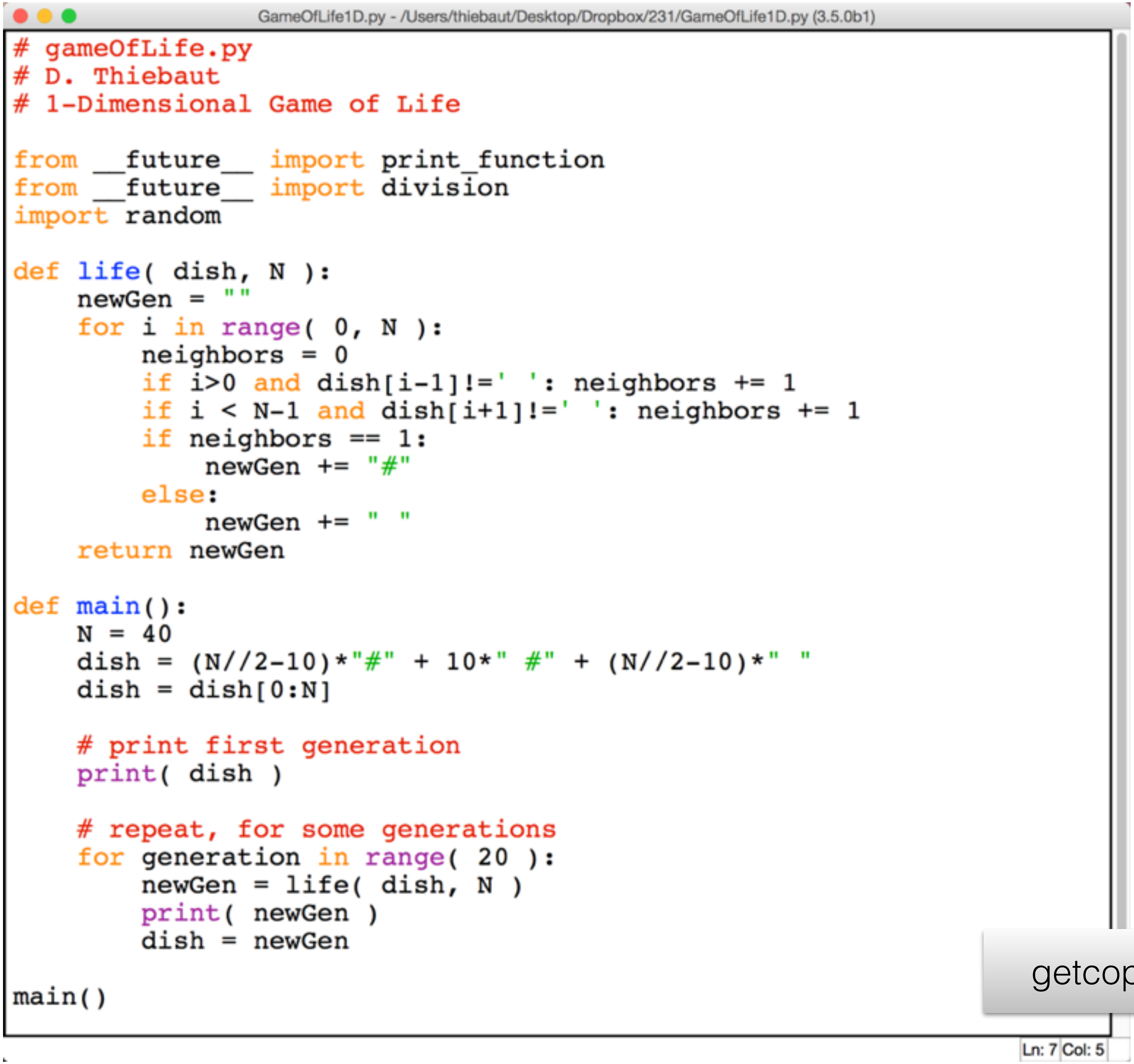
**Problem of the Day(s)**:
Implement 1D Game of Life
in Assembly!

# How to Approach This?

https://img.clipartfest.com/db77689f2cfc577629ec3ff678465323_managed-it-services-nj-it-person-with-question-mark-clipart_4100-6000.jpeg

# **#Step 1**: Write Algorithm in an More Comfortable Language…

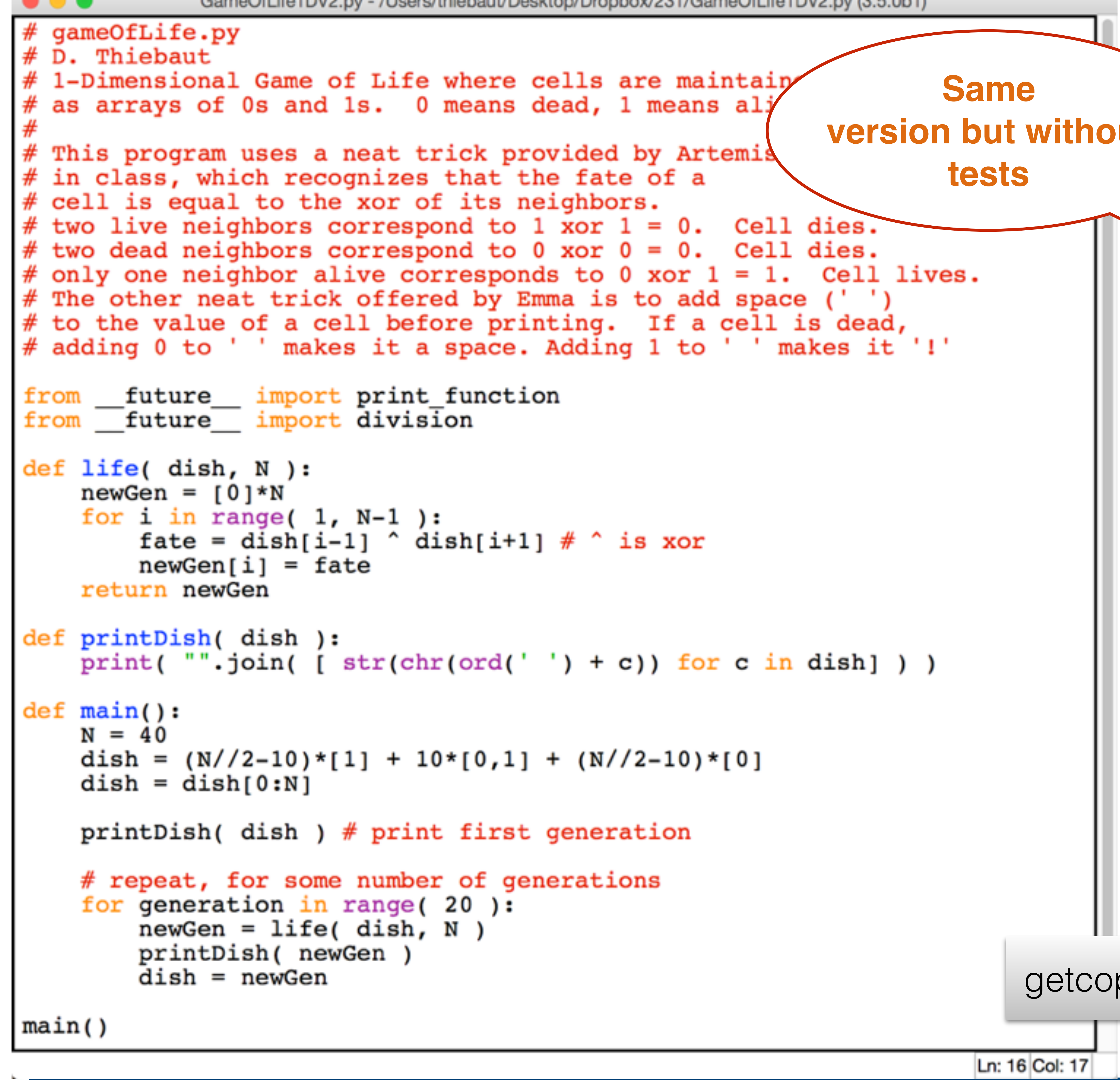
# Game of Life
# Python: V1


GameOfLife1D.py - /Users/thiebaut/Desktop/Dropbox/231/GameOfLife1D.py (3.5.0b1)
Ln: 7 Col: 5

```python
# gameOfLife.py
# D. Thiebaut
# 1-Dimensional Game of Life

from __future__ import print_function
from __future__ import division
import random

def life( dish, N ):
    newGen = ""
    for i in range( 0, N ):
        neighbors = 0
        if i>0 and dish[i-1]!=' ': neighbors += 1
        if i < N-1 and dish[i+1]!=' ': neighbors += 1
        if neighbors == 1:
            newGen += "#"
        else:
            newGen += " "
    return newGen

def main():
    N = 40
    dish = (N//2-10)*"#" + 10*" #" + (N//2-10)*" "
    dish = dish[0:N]

    # print first generation
    print( dish )

    # repeat, for some generations
    for generation in range( 20 ):
        newGen = life( dish, N )
        print( newGen )
        dish = newGen

main()
```

getcopy GameOfLife.py

# Game of Life Python: V2

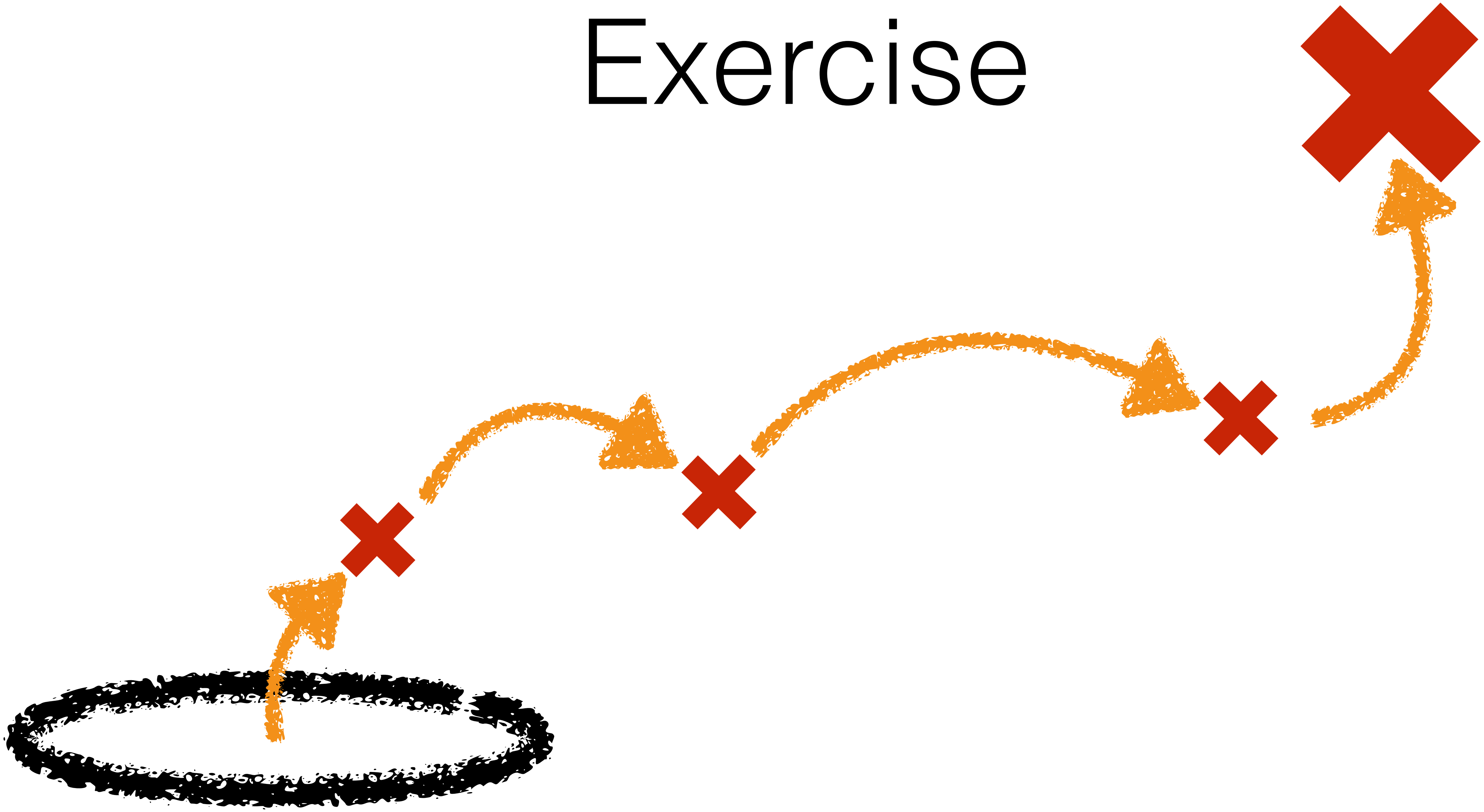

GameOfLife1DV2.py - /Users/thiebaut/Desktop/Dropbox/231/GameOfLife1DV2.py (3.5.0b1)

```
# gameOfLife.py
# D. Thiebaut
# 1-Dimensional Game of Life where cells are maintain
# as arrays of 0s and 1s.  0 means dead, 1 means ali
#
# This program uses a neat trick provided by Artemis
# in class, which recognizes that the fate of a
# cell is equal to the xor of its neighbors.
# two live neighbors correspond to 1 xor 1 = 0.  Cell dies.
# two dead neighbors correspond to 0 xor 0 = 0.  Cell dies.
# only one neighbor alive corresponds to 0 xor 1 = 1.  Cell lives.
# The other neat trick offered by Emma is to add space (' ')
# to the value of a cell before printing.  If a cell is dead,
# adding 0 to ' ' makes it a space. Adding 1 to ' ' makes it '!'

from __future__ import print_function
from __future__ import division

def life( dish, N ):
    newGen = [0]*N
    for i in range( 1, N-1 ):
        fate = dish[i-1] ^ dish[i+1] # ^ is xor
        newGen[i] = fate
    return newGen

def printDish( dish ):
    print( "".join( [ str(chr(ord(' ') + c)) for c in dish] ) )

def main():
    N = 40
    dish = (N//2-10)*[1] + 10*[0,1] + (N//2-10)*[0]
    dish = dish[0:N]

    printDish( dish ) # print first generation

    # repeat, for some number of generations
    for generation in range( 20 ):
        newGen = life( dish, N )
        printDish( newGen )
        dish = newGen

main()
```

Same version but without tests

getcopy GameOfLife_V2.py

# Develop Assembly Program as a Class Exercise



**We stopped here last time…**

# If-statements in Assembly

- **Jmp**: the jump instruction
- **flags** register
- **conditional** jumps (jne, je, jgt, jge, jlt, jle, ja, jb…)

# Jumping around…

```
 Start:
            mov     ebx, Table               ;
            jmp     there                    ;

here:       mov     al, 1                    ;
            mov     ecx, N                   ;

there:      mov     byte[ebx+esi], al        ;
            inc     esi                      ;
            add     al, al                   ;
            jmp     here                     ;
```

# Jumping around…

```
_Start:
        mov     ebx, Table          ;
        jmp     there               ;

here:   mov     al, 1               ;
        mov     ecx, N              ;

there:  mov     byte[ebx+esi], al   ;
        inc     esi                 ;
        add     al, al              ;
        jmp     here                ;
```

# Jumping around…

```
_Start:
        mov     ebx, Table              ;
        jmp     there                   ;

here:   mov     al, 1                   ;
        mov     ecx, N                  ;

there:  mov     byte[ebx+esi], al       ;
        inc     esi                     ;
        add     al, al                  ;
        jmp     here                    ;
```

# Jumping around…

```
_Start:
         mov      ebx, Table           ;
         jmp      there                ;

here:    mov      al, 1                ;
         mov      ecx, N               ;

there:   mov      byte[ebx+esi], al    ;
         inc      esi                  ;
         add      al, al               ;
         jmp      here                 ;
```

# Jumping around…

```
_Start:
        mov     ebx, Table              ;
        jmp     there                   ;

here:   mov     al, 1                   ;
        mov     ecx, N                  ;

there:  mov     byte[ebx+esi], al       ;
        inc     esi                     ;
        add     al, al                  ;
        jmp     here                    ;
```

# Jumping around…

```
_Start:
        mov     ebx, Table          ;
        jmp     there               ;

here:   mov     al, 1               ;
        mov     ecx, N              ;

there:  mov     byte[ebx+esi], al   ;
        inc     esi                 ;
        add     al, al              ;
        jmp     here                ;
```

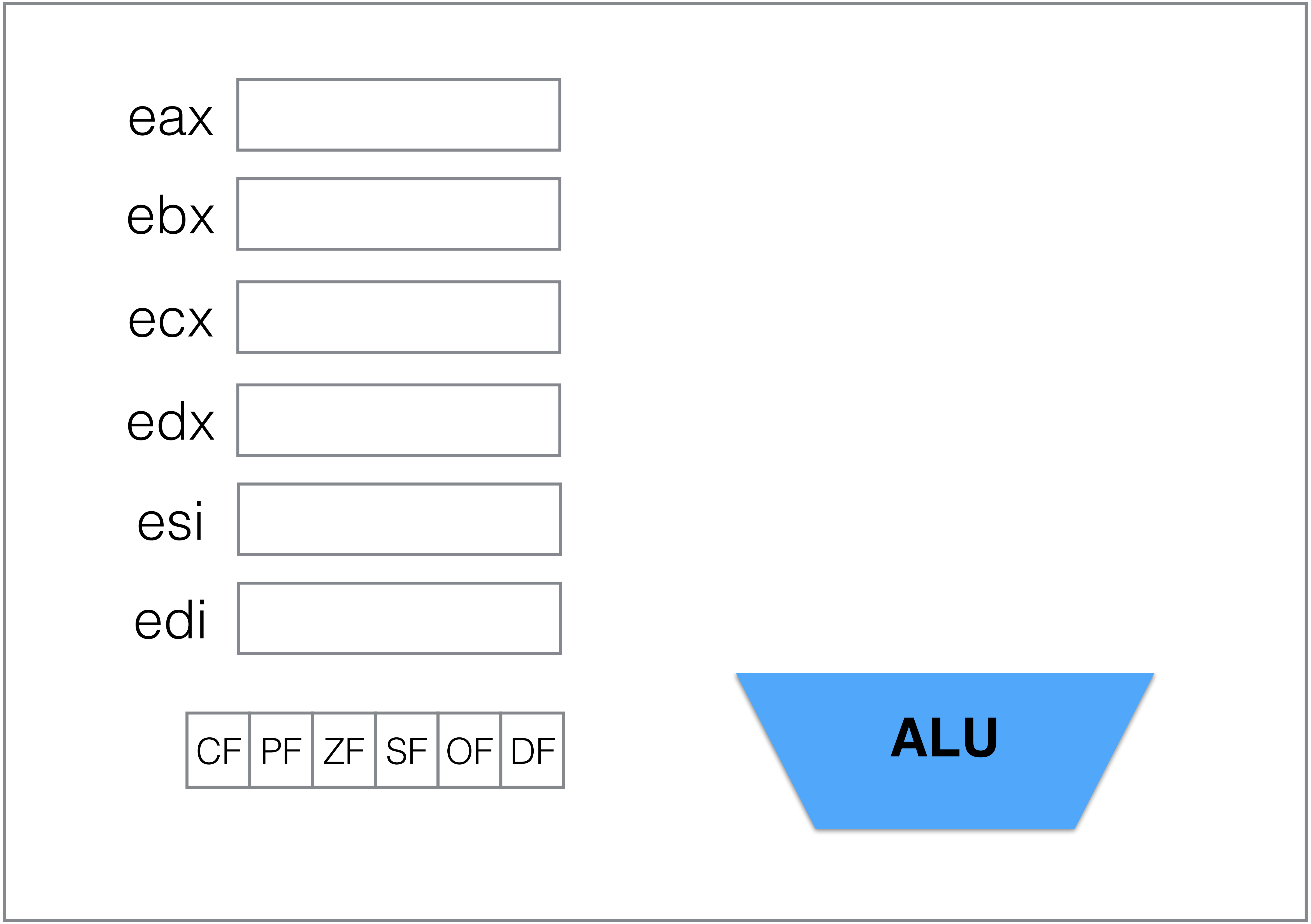
# Jumping around...

```
_Start:
        mov     ebx, Table          ;
        jmp     there               ;

here:   mov     al, 1               ;
        mov     ecx, N              ;

there:  mov     byte[ebx+esi], al   ;
        inc     esi                 ;
        add     al, al              ;
        jmp     here                ;
```

# Jumping around…

```
_Start:
        mov     ebx, Table              ;
        jmp     there                   ;

here:   mov     al, 1                   ;
        mov     ecx, N                  ;

there:  mov     byte[ebx+esi], al       ;
        inc     esi                     ;
        add     al, al                  ;
        jmp     here                    ;
jmp there          ; "mov eip,there"
```

# Flags Register



eax

ebx

ecx

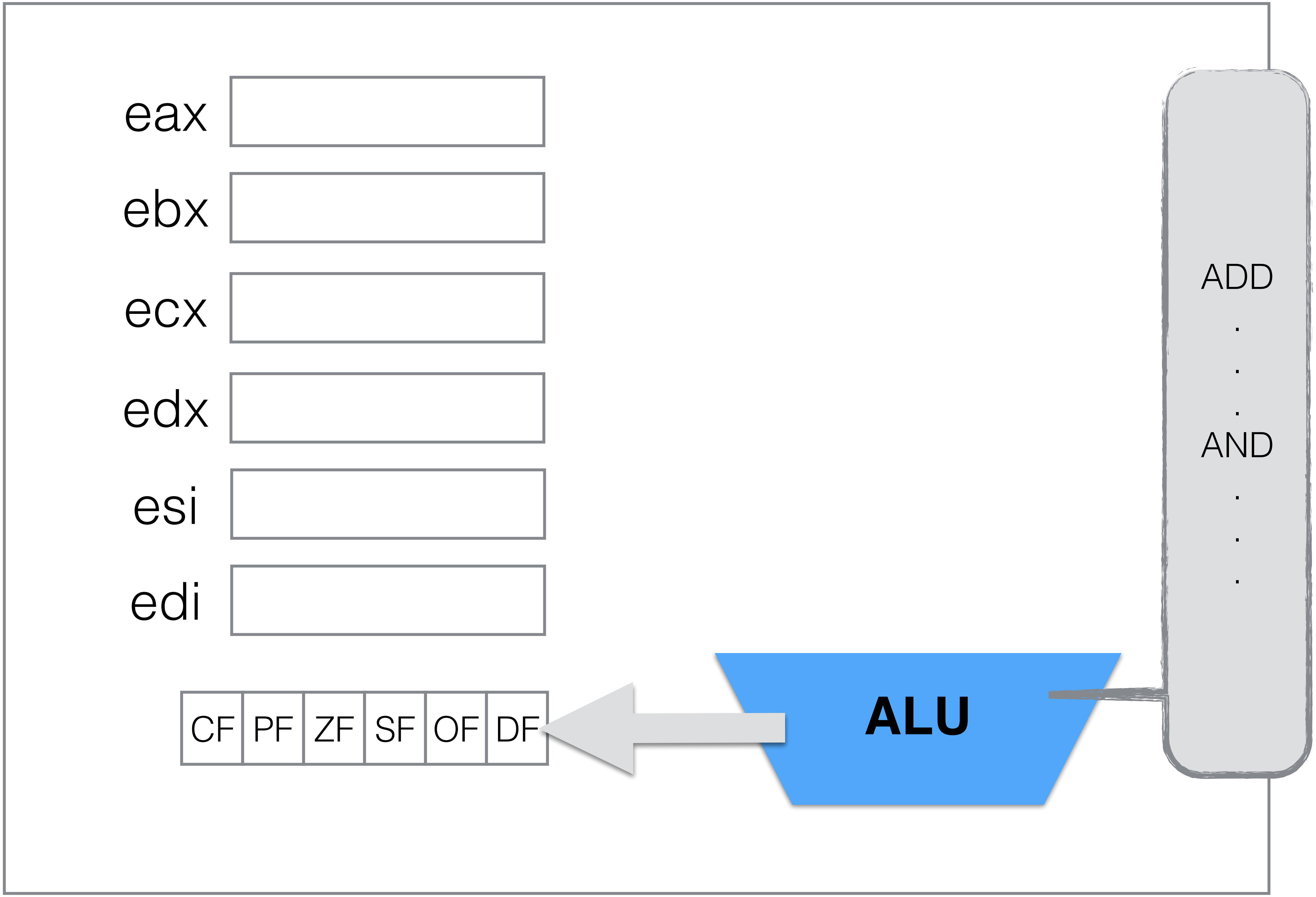
edx

esi

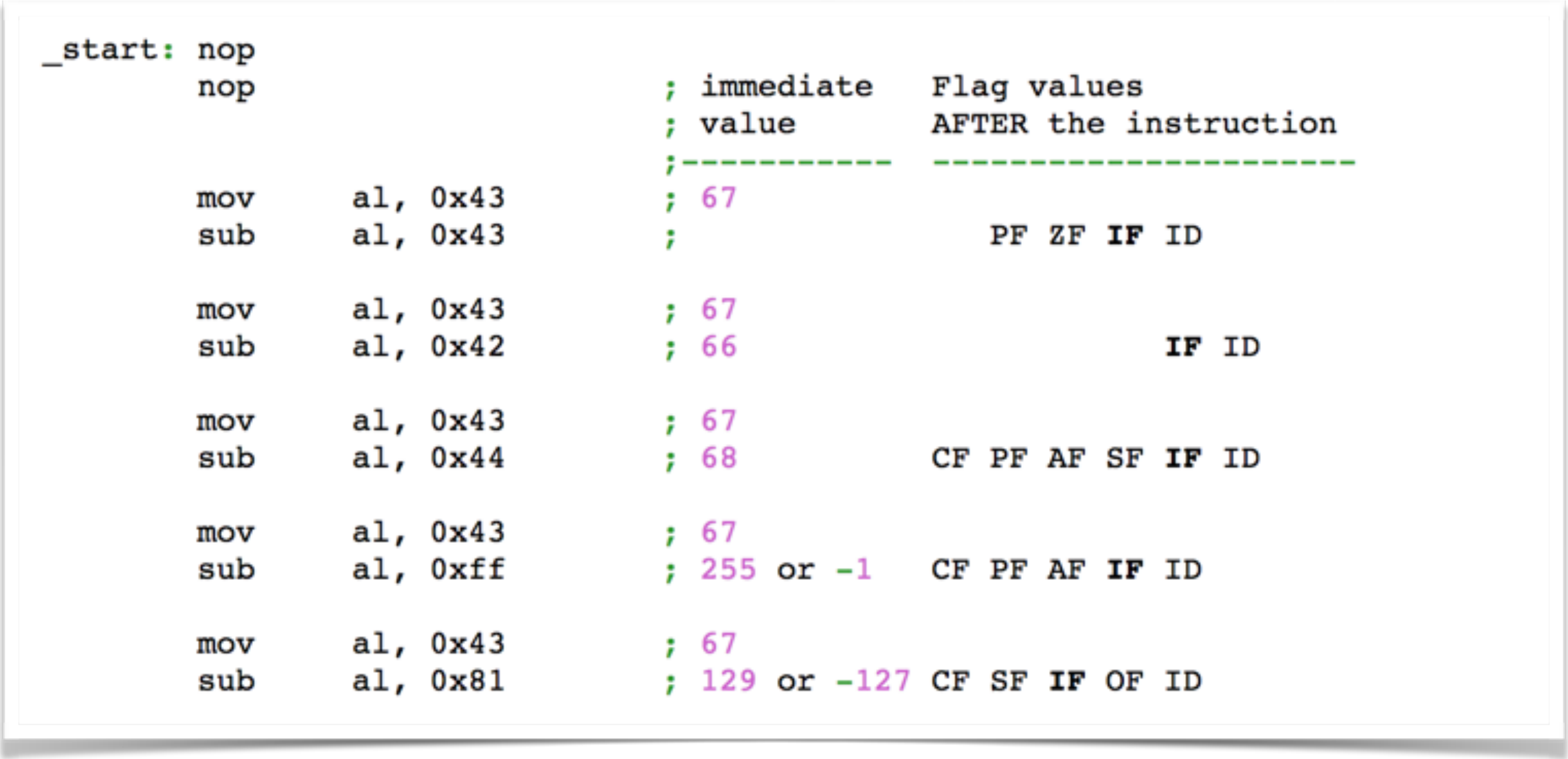
edi

| CF | PF | ZF | SF | OF | DF |
|---|---|---|---|---|---|

ALU


eax
ebx
ecx
edx
esi
edi
CF
PF
ZF
SF
OF
DF
ALU
ADD
.
.
.
AND
.
.
.

# Examples



```
_start: nop
        nop                     ; immediate   Flag values
                                ; value       AFTER the instruction
                                ;-----------  ----------------------
        mov     al, 0x43        ; 67
        sub     al, 0x43        ;                PF ZF IF ID

        mov     al, 0x43        ; 67
        sub     al, 0x42        ; 66                      IF ID

        mov     al, 0x43        ; 67
        sub     al, 0x44        ; 68          CF PF AF SF IF ID

        mov     al, 0x43        ; 67
        sub     al, 0xff        ; 255 or -1   CF PF AF IF ID

        mov     al, 0x43        ; 67
        sub     al, 0x81        ; 129 or -127 CF SF IF OF ID
```

# Conditional Jumps

# Example with Jnz

```
_Start:
        mov     ecx, 10
for:    . . .


        dec     ecx         ;ecx <— ecx - 1
        jnz     for         ;if previous op didn't result in 0 in ALU
                            ; then jump
        . . .               ; else continue here…
```

# Family of Conditional Jumps

- **JE, JZ**
- **JNE, JNZ**
- **JG, JGE, JNL**
- **JL, JLE, JNG**

# Family of Conditional Jumps

EAX: 0xFFFF FFFF
EBX: 0x0000 0001 } Which is greater?

- **JE, JZ**
- **JNE, JNZ**
- **JG, JGE, JNL**
- **JL, JLE, JNG**

# Family of Conditional Jumps

EAX: 0xFFFF FFFF
EBX: 0x0000 0001 } Which is greater?

- **JE, JZ**
- **JNE, JNZ**
- **JG, JGE, JNL**
- **JL, JLE, JNG**

- **JE, JZ**
- **JNE, JNZ**
- **JA, JAE, JNB**
- **JB, JBE, JNA**

# How do we compare two quantities?

```
; if (a==b)
;       c = 3
; else
;       c = -1

        mov     eax, dword[a]           ;eax <— a
        mov     ebx, dword[b]           ;ebx <— b
        sub     eax, ebx                ;eax <— a - b, set flags
        jne     else                    ;if ZF flag not set, go to else

then:   mov     dword[c], 3             ;otherwise, a==b, set c to 3
        jmp     done                    ;and skip else part

else:   mov     dword[c], -1            ;a!=b, set c to -1

done:   . . .
```

**sub** is ok, but it modifies the dest operand

```
; if (a==b)
;       c = 3
; else
;       c = -1

        mov     eax, dword[a]           ;eax <— a
        mov     ebx, dword[b]           ;ebx <— b
        ~~sub~~     eax, ebx                ;eax <— a - b, set flags
        jne     else                    ;if ZF flag not set, go to else

then:   mov     dword[c], 3             ;otherwise, a==b, set c to 3
        jmp     done                    ;and skip else part

else:   mov     dword[c], -1            ;a!=b, set c to -1

done:   . . .
```

**cmp** is better, it subtracts eax from ebx, ***but does not modify eax***

```
; if (a==b)
;       c = 3
; else
;       c = -1

        mov     eax, dword[a]           ;eax <– a
        mov     ebx, dword[b]           ;ebx <– b
        cmp     eax, ebx                ;eax <– a - b, set flags
        jne     else                    ;if ZF flag not set, go to else

then:   mov     dword[c], 3             ;otherwise, a==b, set c to 3
        jmp     done                    ;and skip else part

else:   mov     dword[c], -1            ;a!=b, set c to -1

done:   . . .
```

# Another example with cmp

```
; int a, c   // signed ints
; if (a < 10)
;     c = 3
; else
;     c = -1

        mov     eax, dword[a]        ;eax <– a
        cmp     eax, 10              ;eax <– a - 10, set flags
        jnl     else                 ;if not less than 10, go to else

then:   mov     dword[c], 3          ;a<10, set c to 3
        jmp     done                 ;and skip else part

else:   mov     dword[c], -1         ;a >= 10, set c to -1

done:   . . .
```

# Another example with cmp

or…

```
; int a, c   // signed ints
; if (a < 10)
;     c = 3
; else
;     c = -1

        mov     eax, dword[a]        ;eax <– a
        cmp     eax, 10              ;eax <– a - 10, set flags
        jl      then                 ;if a<10 go to then

else:   mov     dword[c], -1         ;otherwise, a>=10, set c to -1
        jmp     done                 ;and skip then part

then:   mov     dword[c], 3          ;a < 10, set c to 3

done:   . . .
```

# Exercise

Translate this for-loop in assembly

```
; int sum = 0
; for (int i=0; i<20; i+=2 ) {
;    sum += i;
; }
```

# Exercise 2

Translate this for-loop in assembly

```
; unsigned int i, sum = 0
; for (i=0; i<4000000000; i+=2 ) {
;     sum += 1;
; }
```