http://www.bhmpics.com/view-road-traffic-2560x1600.html

Introduction to Computing

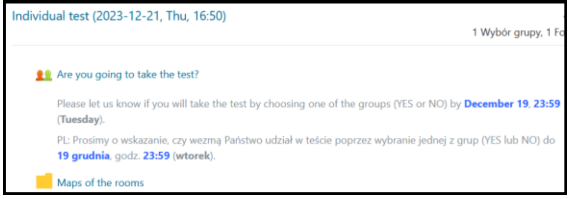
Jerzy Nawrocki
jerzy.nawrocki@put.poznan.pl

Parallel Processing

About Individual Test

If you:
- don't speak Polish, and
- going to take the test on Dec. 21

then let me know by email by Dec. 19.


Individual test (2023-12-21, Thu, 16:50)

1 Wybór grupy, 1 Fo

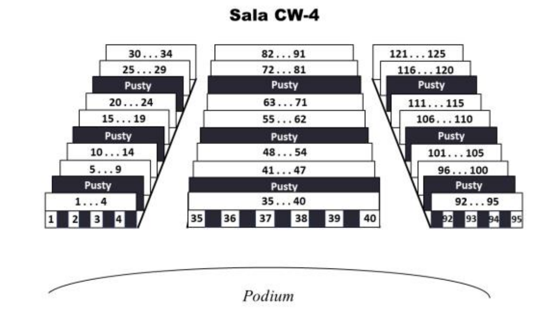
Are you going to take the test?

Please let us know if you will take the test by choosing one of the groups (YES or NO) by **December 19**, **23:59** (**Tuesday**).

PL: Prosimy o wskazanie, czy wezmą Państwo udział w teście poprzez wybranie jednej z grup (YES lub NO) do **19 grudnia**, godz. **23:59** (**wtorek**).

Maps of the rooms

Seat allocation



Each student is assigned to CW-4, CW-8, or CW-11.
Within a room, each student is assigned a seat number.
Pusty = Empty

(3)

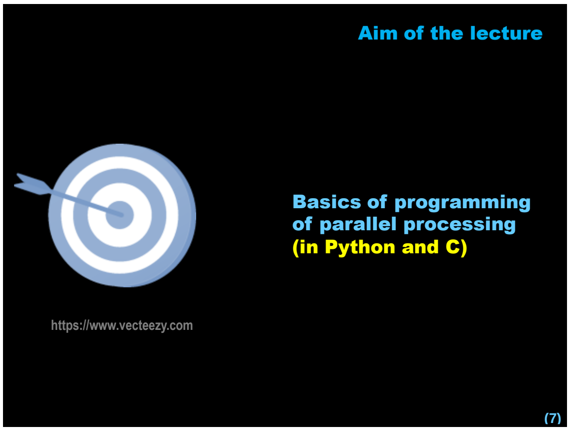
Tentative schedule of lectures

No.	Topic	Date
1	Imperative Programming	2023-10-09
2	Digital Circuits	2023-10-16
3	Computers	2023-10-23
4	Subprograms	2023-11-06
5	Text Processing	2023-11-13
6	Object-oriented Programming	2023-11-20
7	Numerical methods	2023-11-27
8	Computational Complexity	2023-12-04
9	Databases and Machine Learning	2023-12-11
10	Parallel Processing	2023-12-18
11	Computer Networks & Cybersecuri	2024-01-08
12	Software Engineering	2024-01-15
13	Embedded Systems	2024-01-22
14	Professionalism in Computing	2024-01-29
	Individual Test (topics 1-8)	2023-12-21
	Team Contest (topics 1-11)	2024-01-11

(4)

Does parallel processing matter?

(5)

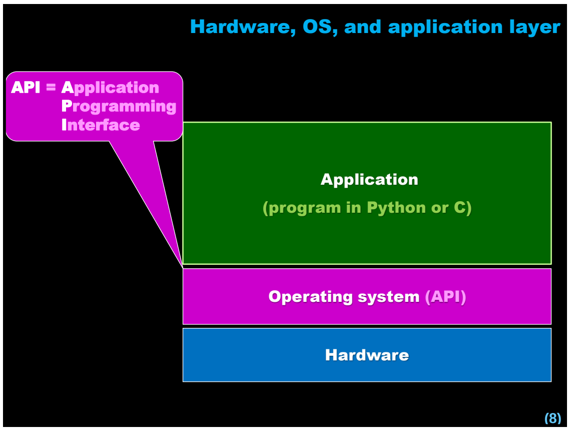


Aim of the lecture

Basics of programming of parallel processing (in Python and C)

https://www.vecteezy.com

(7)



Hardware, OS, and application layer

API = Application Programming Interface

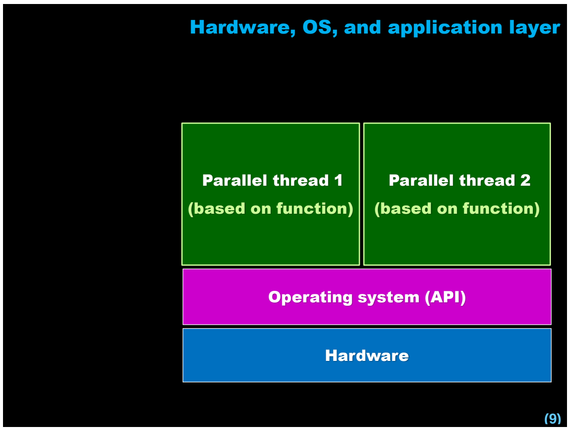
Application

(program in Python or C)

Operating system (API)

Hardware

(8)



Hardware, OS, and application layer

Parallel thread 1

(based on function)

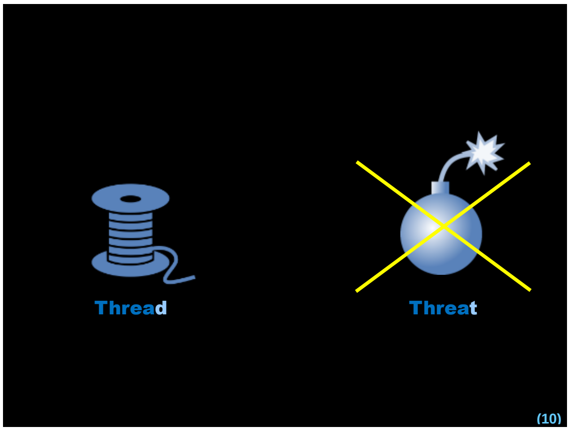
Parallel thread 2

(based on function)

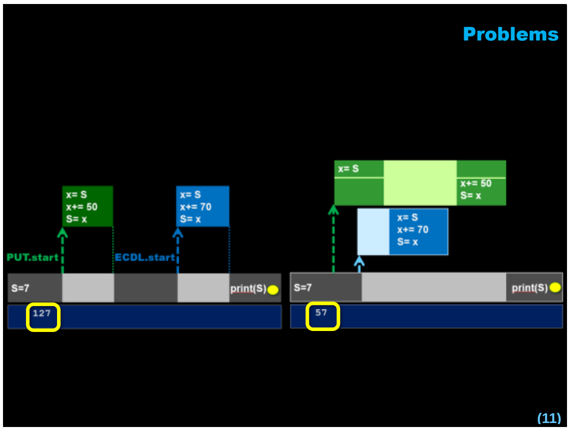
Operating system (API)

Hardware

(9)

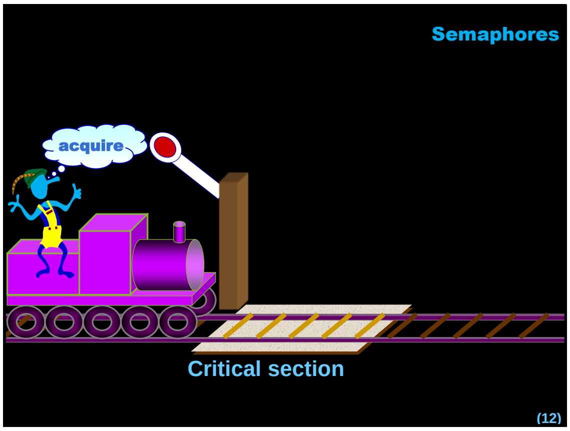


(10)



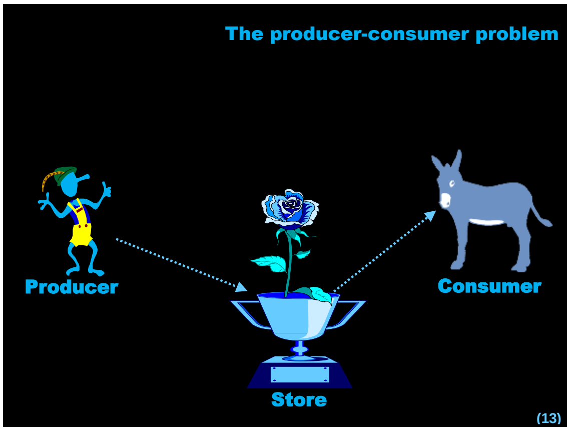
Problems

(11)



Semaphores

(12)



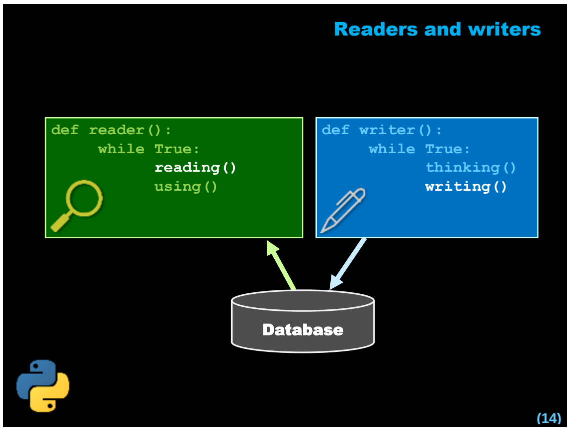
The producer-consumer problem

Producer

Consumer

Store

(13)



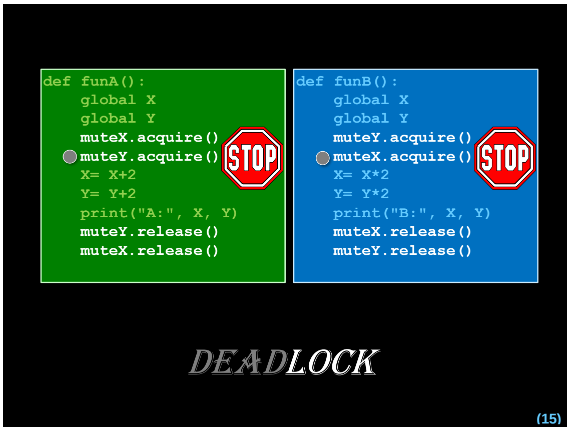
Readers and writers

```
def reader():
    while True:
        reading()
        using()
```

```
def writer():
    while True:
        thinking()
        writing()
```

Database

(14)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

STOP

STOP

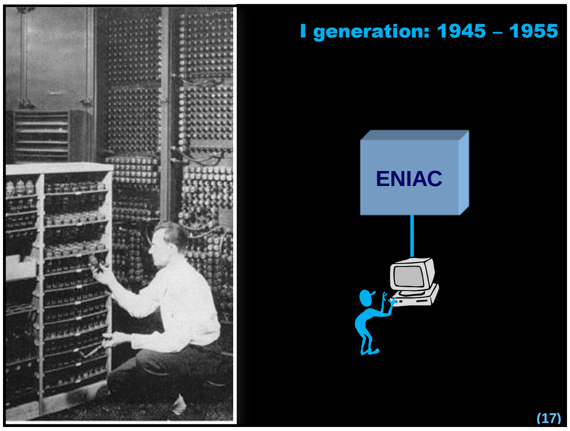
DEADLOCK

(15)

- Operating systems
- Parallel threads
- Indeterminism
- Semaphores
- Producer-consumer problem
- Readers-writers problem
- Deadlock

http://www.bhmpics.com/view-road-traffic-2560x1600.html

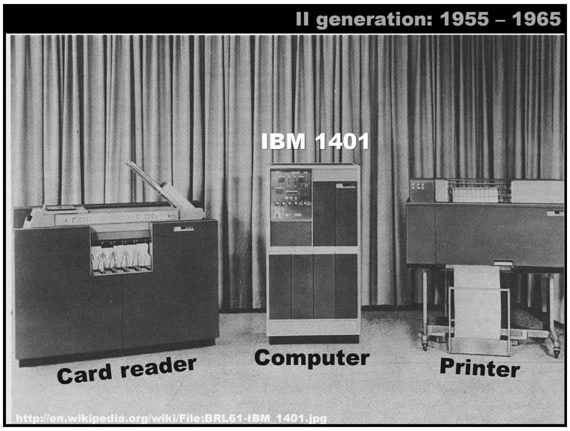
(16)



I generation: 1945 – 1955

ENIAC

(17)



II generation: 1955 – 1965

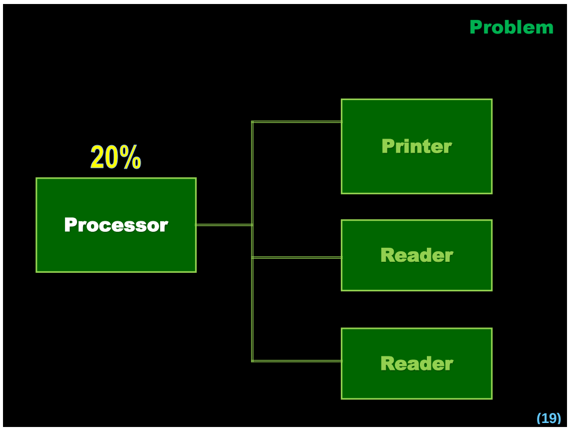
IBM 1401

Card reader

Computer

Printer

http://en.wikipedia.org/wiki/File:BRL61-IBM_1401.jpg



Problem

20%

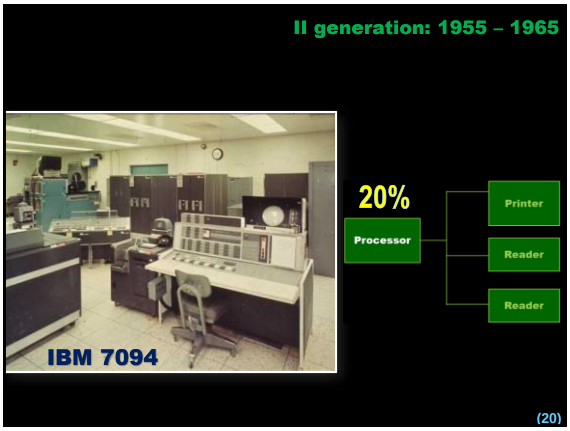
Processor

Printer

Reader

Reader

(19)



II generation: 1955 – 1965

IBM 7094

20%

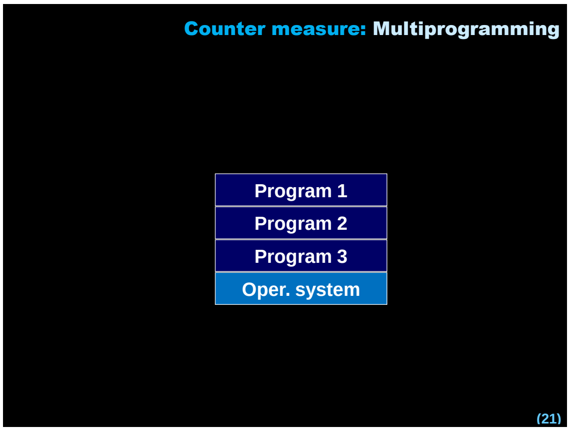
Processor

Printer

Reader

Reader

(20)



Counter measure: Multiprogramming

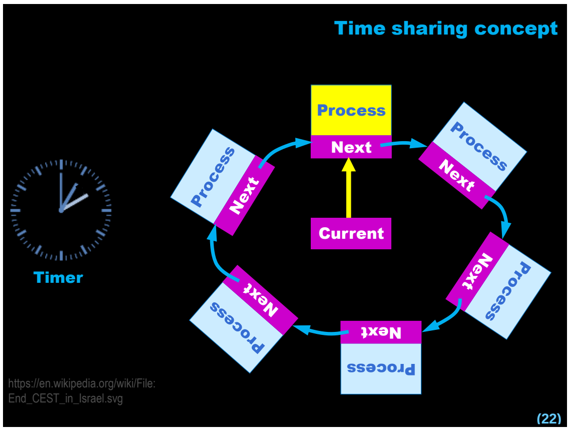
Program 1

Program 2

Program 3

Oper. system

(21)



Time sharing concept

Timer

Process Next

Process Next

Process Next

Process Next

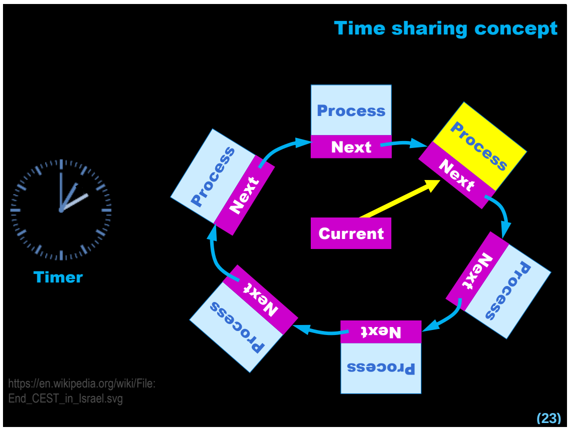
Process Next

Process Next

Current

https://en.wikipedia.org/wiki/File:
End_CEST_in_Israel.svg

(22)



Time sharing concept

Timer

Process Next

Process Next

Process Next

Process Next

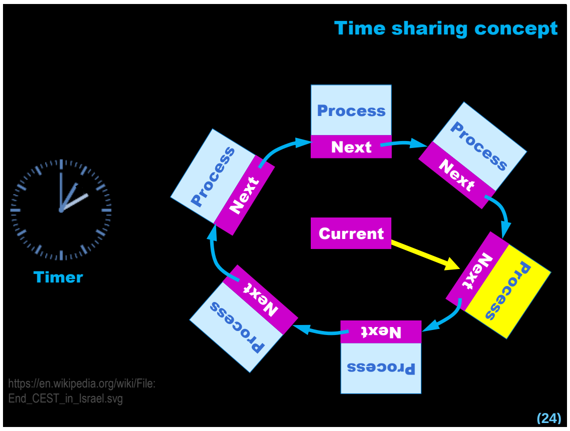
Process Next

Process Next

Current

https://en.wikipedia.org/wiki/File:
End_CEST_in_Israel.svg

(23)



Time sharing concept

Timer

Process Next

Process Next

Process Next

Process Next

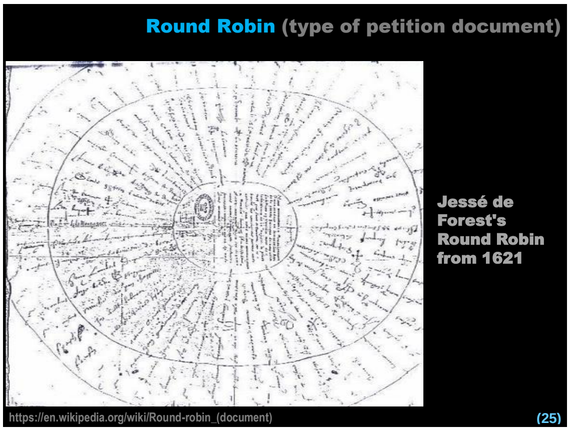
Process Next

Process Next

Current

https://en.wikipedia.org/wiki/File:
End_CEST_in_Israel.svg

(24)

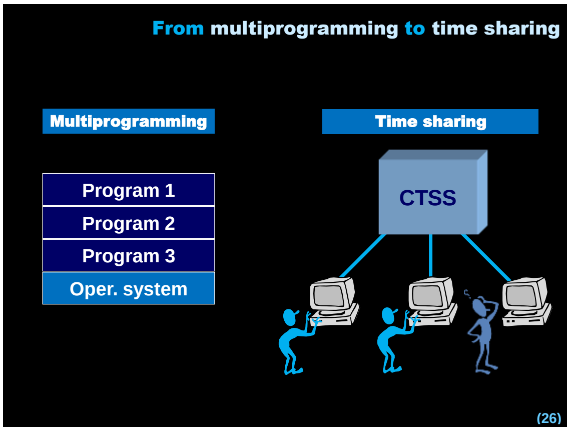


Round Robin (type of petition document)

Jessé de Forest's Round Robin from 1621

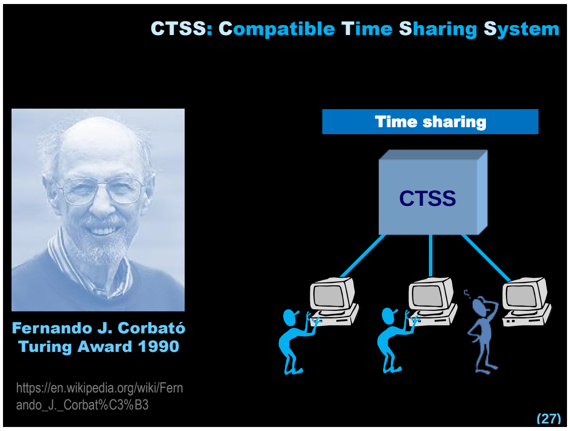
https://en.wikipedia.org/wiki/Round-robin_(document)

(25)



From multiprogramming to time sharing

(26)

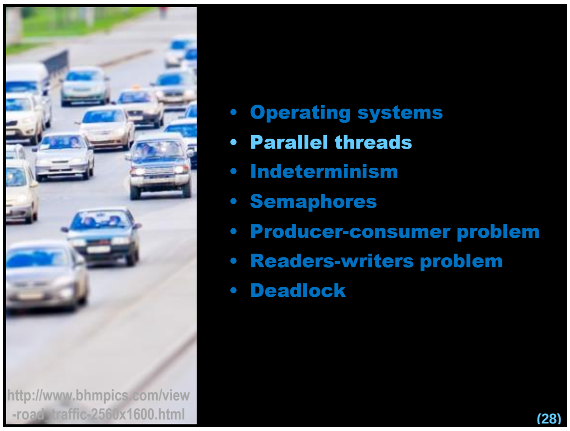


CTSS: Compatible Time Sharing System

Fernando J. Corbató
Turing Award 1990

https://en.wikipedia.org/wiki/Fernando_J._Corbat%C3%B3

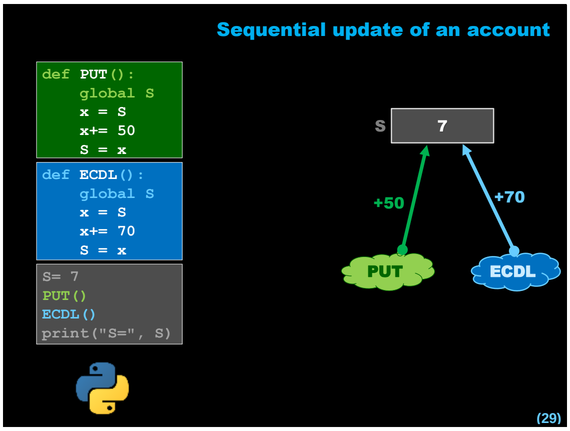
(27)



- Operating systems
- Parallel threads
- Indeterminism
- Semaphores
- Producer-consumer problem
- Readers-writers problem
- Deadlock

http://www.bhmpics.com/view-road_traffic-2560x1600.html

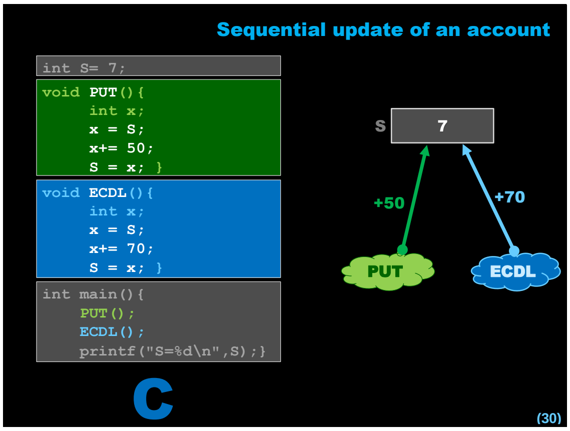
(28)



Sequential update of an account

```
def PUT():
    global S
    x = S
    x+= 50
    S = x
def ECDL():
    global S
    x = S
    x+= 70
    S = x
S= 7
PUT()
ECDL()
print("S=", S)
```

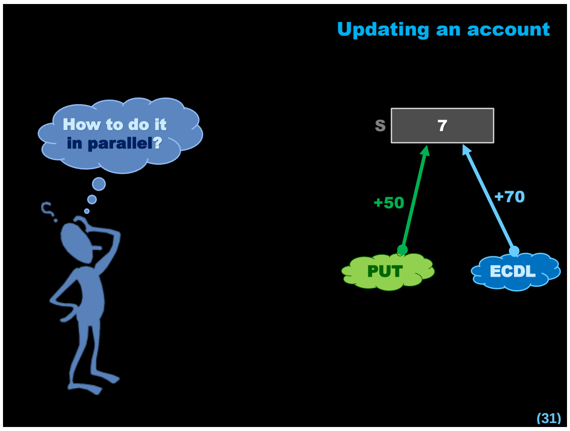
(29)



Sequential update of an account

```
int S= 7;
void PUT(){
     int x;
     x = S;
     x+= 50;
     S = x; }
void ECDL(){
     int x;
     x = S;
     x+= 70;
     S = x; }
int main(){
     PUT();
     ECDL();
     printf("S=%d\n",S);}
```

(30)



Updating an account

How to do it in parallel?

S 7

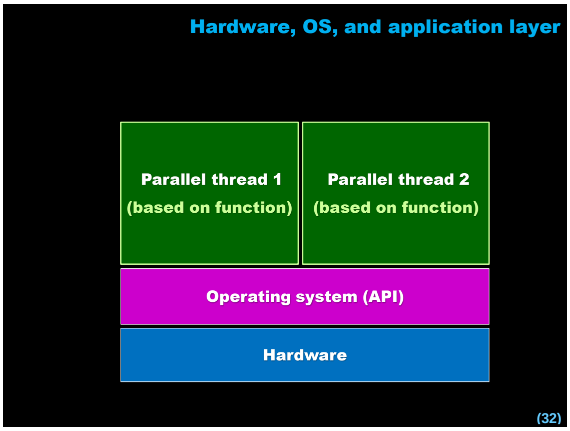
+50

+70

PUT

ECDL

(31)



Hardware, OS, and application layer

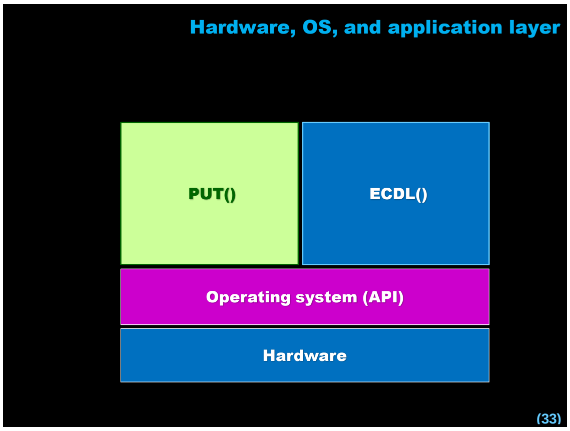
Parallel thread 1 (based on function)

Parallel thread 2 (based on function)

Operating system (API)

Hardware

(32)



Hardware, OS, and application layer

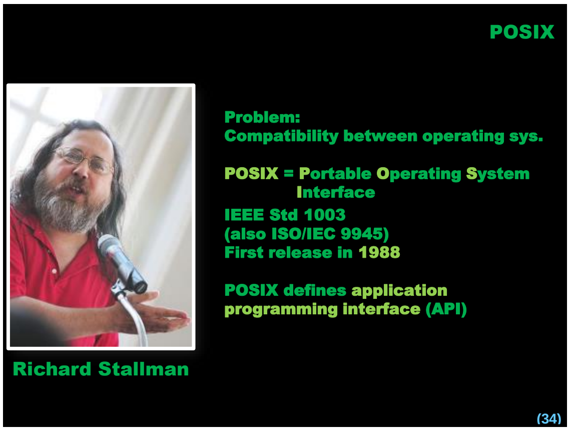
PUT()

ECDL()

Operating system (API)

Hardware

(33)



POSIX

Richard Stallman

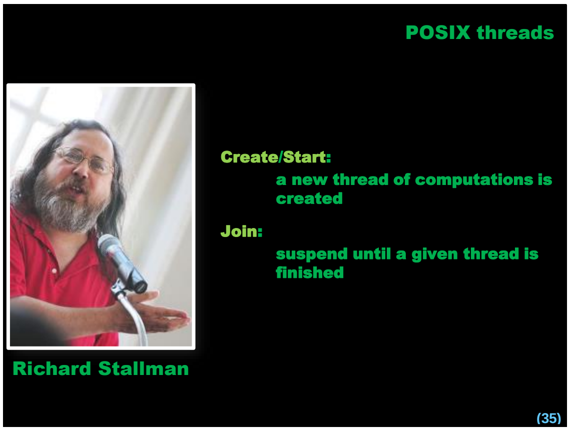
Problem:
Compatibility between operating sys.

POSIX = Portable Operating System Interface

IEEE Std 1003
(also ISO/IEC 9945)
First release in 1988

POSIX defines application programming interface (API)

(34)



POSIX threads

Richard Stallman

Create/Start:
a new thread of computations is created

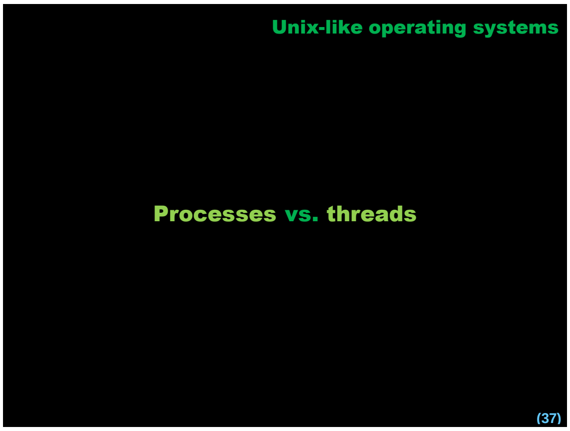
Join:
suspend until a given thread is finished

(35)

Roughly speaking:

Thread = a subprogram that can be executed in parallel
- to the main program and
- to other threads.

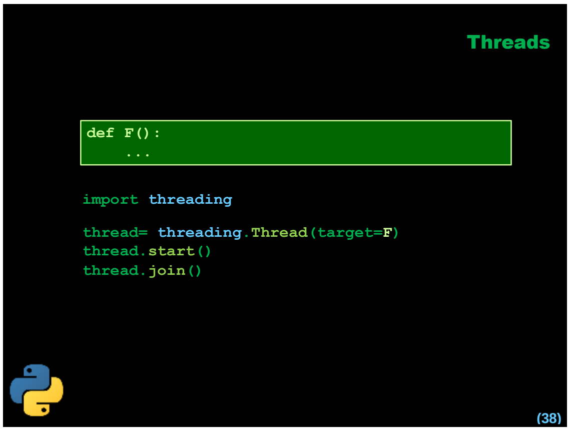
(36)



Unix-like operating systems

Processes vs. threads

(37)

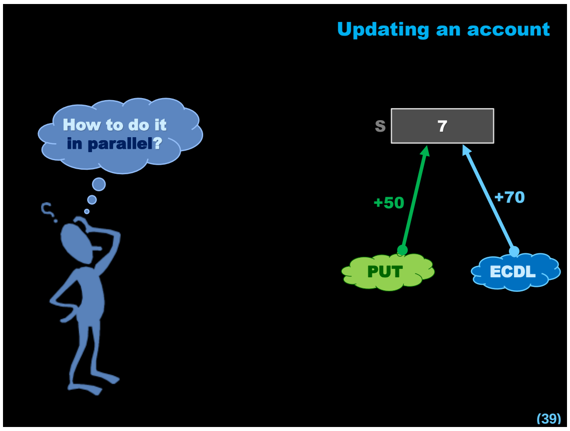


Threads

```
def F():
    ...
```

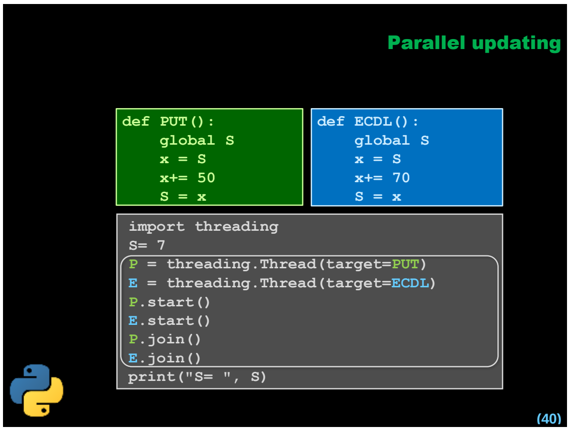
```
import threading

thread= threading.Thread(target=F)
thread.start()
thread.join()
```

(38)



Updating an account

(39)



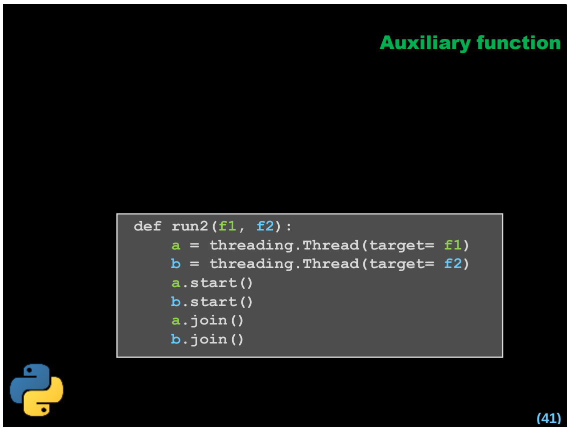
Parallel updating

```
def PUT():
    global S
    x = S
    x+= 50
    S = x
```

```
def ECDL():
    global S
    x = S
    x+= 70
    S = x
```

```
import threading
S= 7
P = threading.Thread(target=PUT)
E = threading.Thread(target=ECDL)
P.start()
E.start()
P.join()
E.join()
print("S= ", S)
```

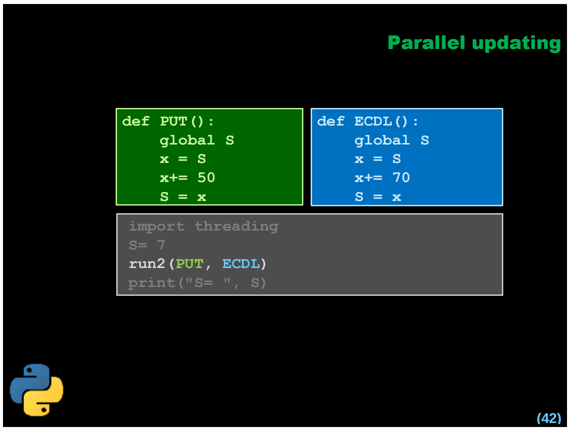
(40)



Auxiliary function

```
def run2(f1, f2):
    a = threading.Thread(target= f1)
    b = threading.Thread(target= f2)
    a.start()
    b.start()
    a.join()
    b.join()
```

(41)



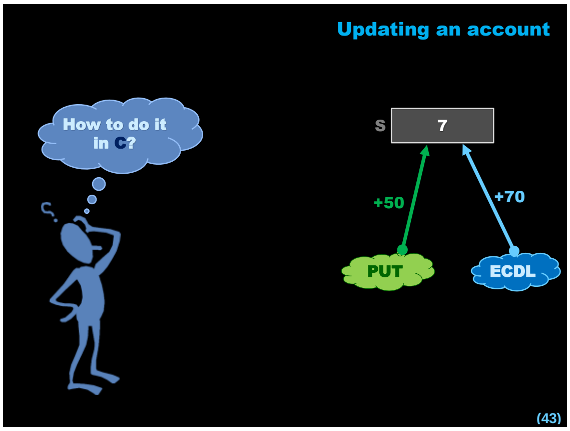
Parallel updating

```
def PUT():
    global S
    x = S
    x+= 50
    S = x
```

```
def ECDL():
    global S
    x = S
    x+= 70
    S = x
```

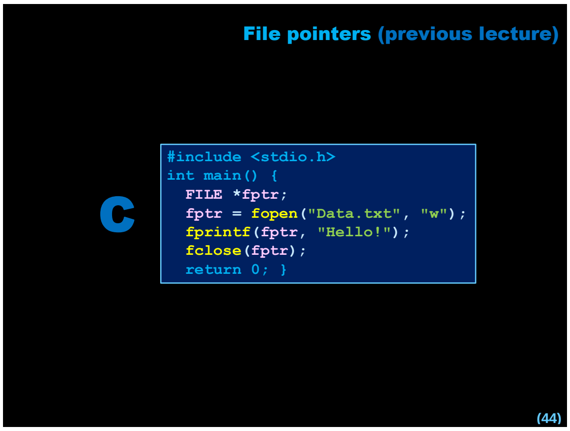
```
import threading
S= 7
run2(PUT, ECDL)
print("S= ", S)
```

(42)



Updating an account

(43)

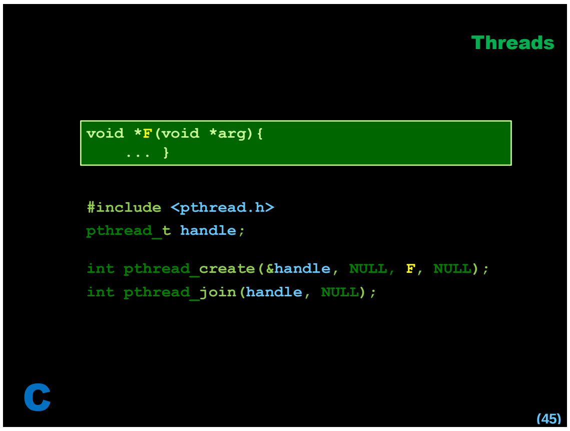


File pointers (previous lecture)

C

```
#include <stdio.h>
int main() {
  FILE *fptr;
  fptr = fopen("Data.txt", "w");
  fprintf(fptr, "Hello!");
  fclose(fptr);
  return 0; }
```

(44)



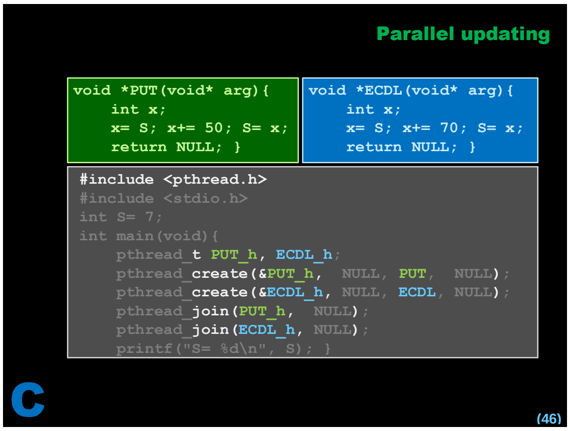
Threads

```
void *F(void *arg){
    ... }
```

```
#include <pthread.h>
pthread_t handle;

int pthread_create(&handle, NULL, F, NULL);
int pthread_join(handle, NULL);
```

C

(45)



Parallel updating

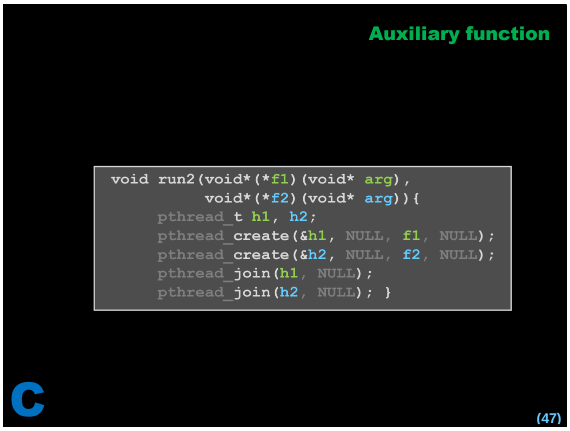
```
void *PUT(void* arg){
    int x;
    x= S; x+= 50; S= x;
    return NULL; }
```

```
void *ECDL(void* arg){
    int x;
    x= S; x+= 70; S= x;
    return NULL; }
```

```
#include <pthread.h>
#include <stdio.h>
int S= 7;
int main(void){
    pthread_t PUT_h, ECDL_h;
    pthread_create(&PUT_h,  NULL, PUT,  NULL);
    pthread_create(&ECDL_h, NULL, ECDL, NULL);
    pthread_join(PUT_h,  NULL);
    pthread_join(ECDL_h, NULL);
    printf("S= %d\n", S); }
```

C

(46)

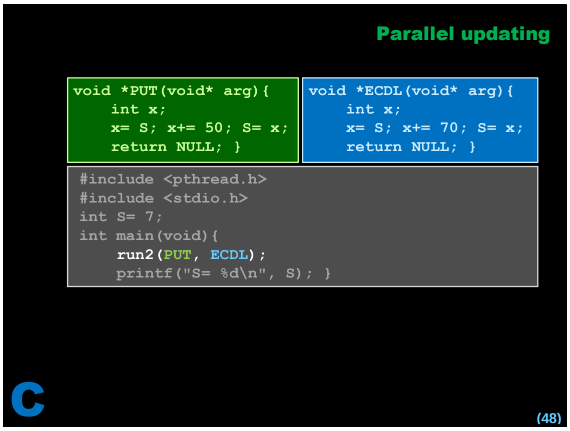


Auxiliary function

```
void run2(void*(*f1)(void* arg),
          void*(*f2)(void* arg)){
    pthread_t h1, h2;
    pthread_create(&h1, NULL, f1, NULL);
    pthread_create(&h2, NULL, f2, NULL);
    pthread_join(h1, NULL);
    pthread_join(h2, NULL); }
```

C

(47)



Parallel updating

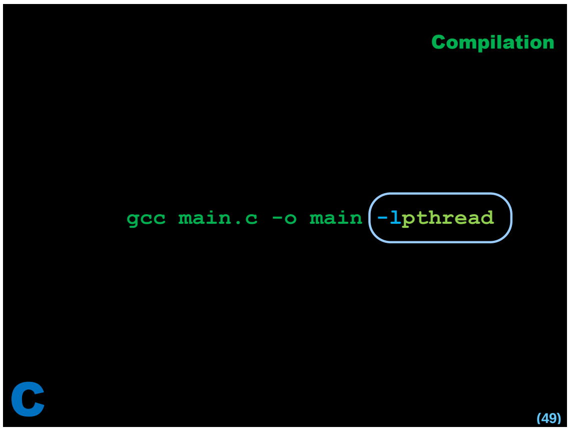
```
void *PUT(void* arg){
    int x;
    x= S; x+= 50; S= x;
    return NULL; }
```

```
void *ECDL(void* arg){
    int x;
    x= S; x+= 70; S= x;
    return NULL; }
```

```
#include <pthread.h>
#include <stdio.h>
int S= 7;
int main(void){
    run2(PUT, ECDL);
    printf("S= %d\n", S); }
```

C

(48)

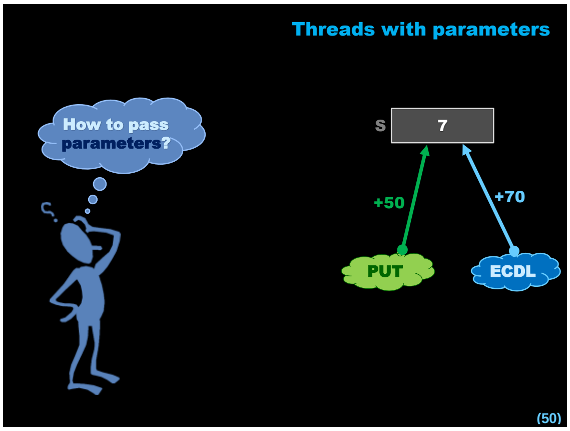


Compilation

```
gcc main.c -o main -lpthread
```

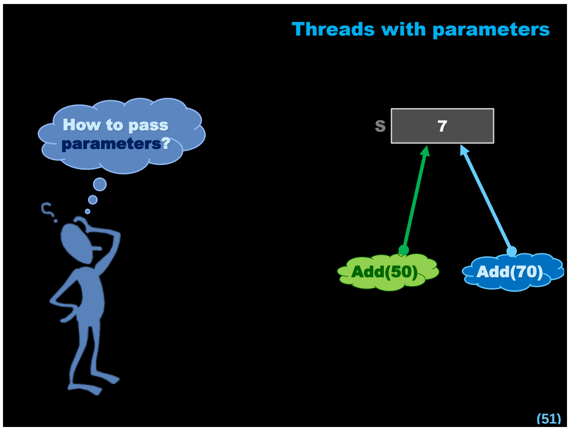
C

(49)



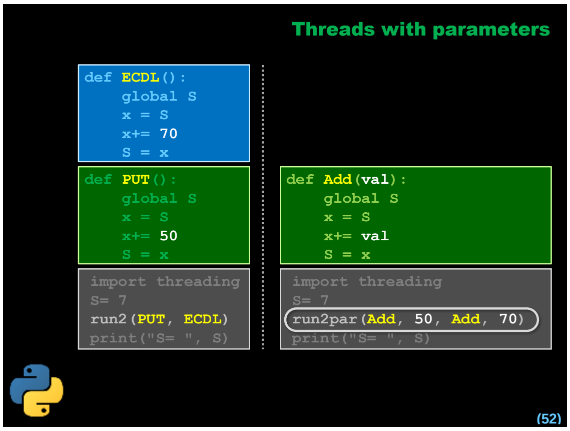
Threads with parameters

(50)



Threads with parameters

(51)



Threads with parameters

```
def ECDL():
    global S
    x = S
    x+= 70
    S = x
```

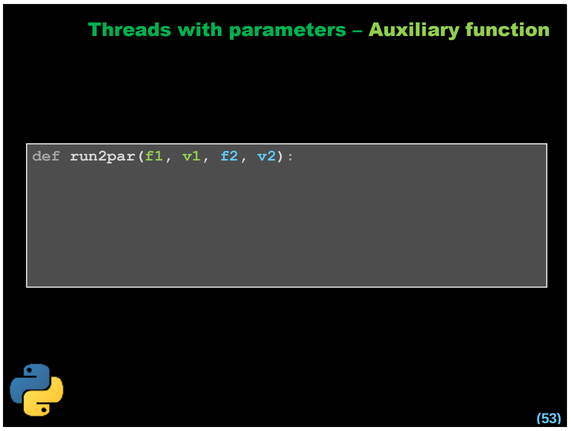
```
def PUT():
    global S
    x = S
    x+= 50
    S = x
```

```
import threading
S= 7
run2(PUT, ECDL)
print("S= ", S)
```

```
def Add(val):
    global S
    x = S
    x+= val
    S = x
```

```
import threading
S= 7
run2par(Add, 50, Add, 70)
print("S= ", S)
```

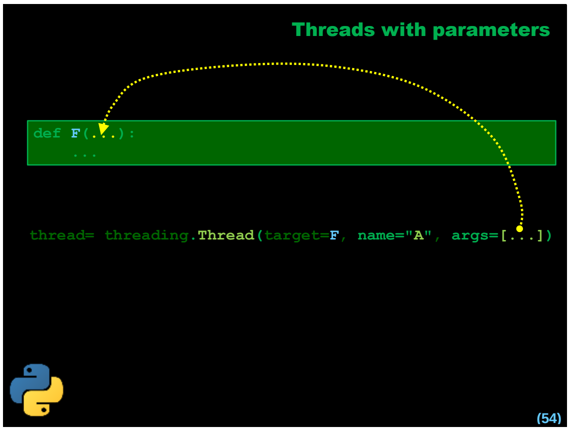
(52)



Threads with parameters – Auxiliary function

```
def run2par(f1, v1, f2, v2):
```

(53)

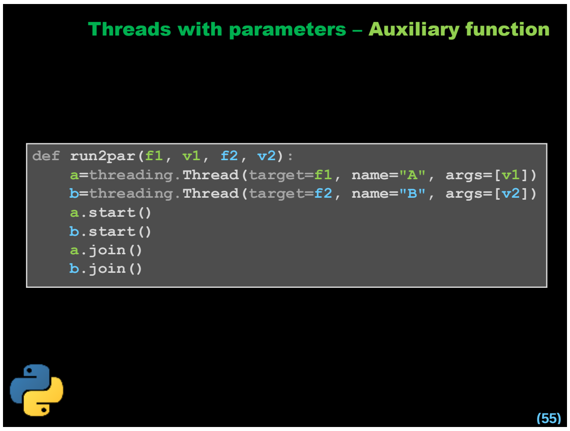


Threads with parameters

```
def F(...):
    ...
```

```
thread= threading.Thread(target=F, name="A", args=[...])
```

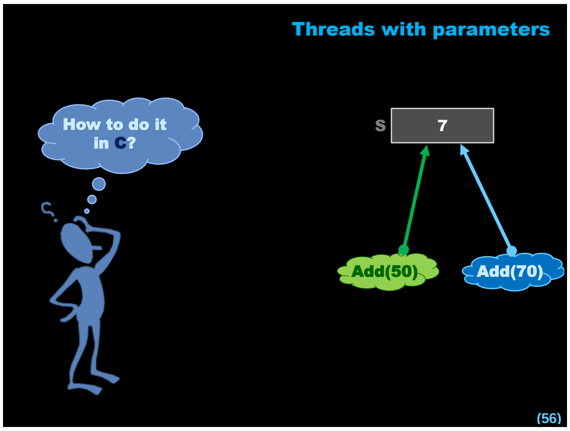
(54)



Threads with parameters – Auxiliary function

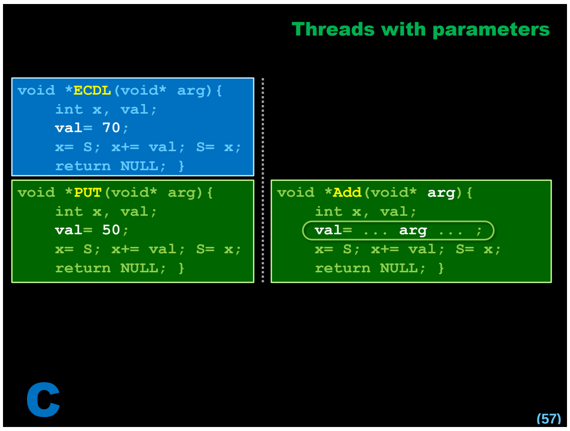
```
def run2par(f1, v1, f2, v2):
    a=threading.Thread(target=f1, name="A", args=[v1])
    b=threading.Thread(target=f2, name="B", args=[v2])
    a.start()
    b.start()
    a.join()
    b.join()
```

(55)



Threads with parameters

(56)



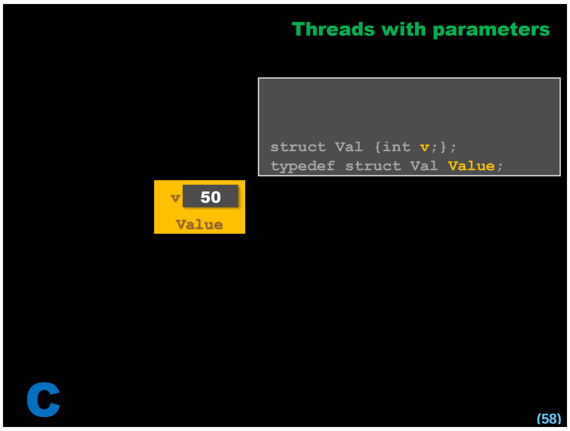
Threads with parameters

```
void *ECDL(void* arg){
    int x, val;
    val= 70;
    x= S; x+= val; S= x;
    return NULL; }
```

```
void *PUT(void* arg){
    int x, val;
    val= 50;
    x= S; x+= val; S= x;
    return NULL; }
```

```
void *Add(void* arg){
    int x, val;
    val= ... arg ... ;
    x= S; x+= val; S= x;
    return NULL; }
```

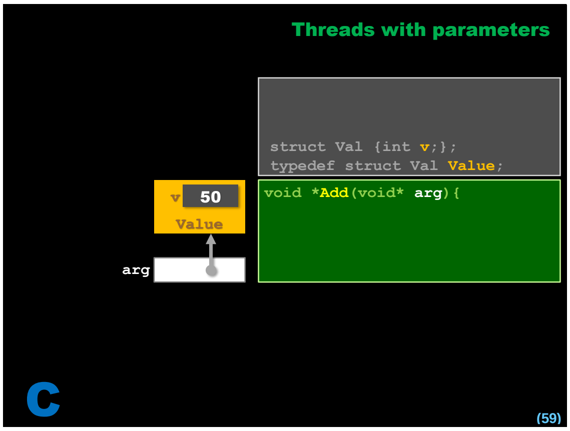
(57)



Threads with parameters

```
struct Val {int v;};
typedef struct Val Value;
```

(58)

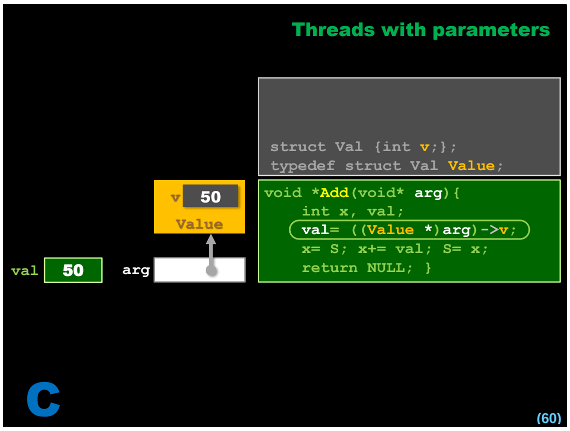


Threads with parameters

```
struct Val {int v;};
typedef struct Val Value;
```

```
void *Add(void* arg){
```

(59)

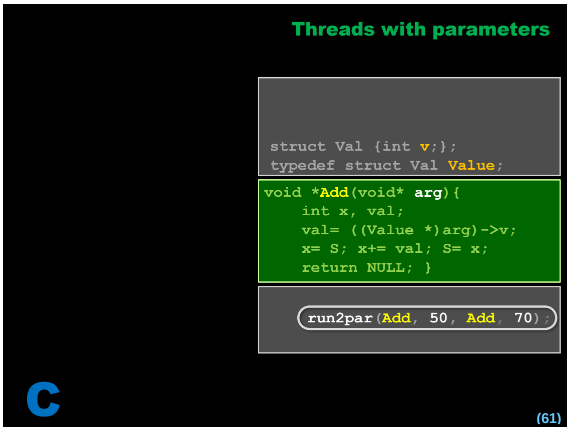


Threads with parameters

```
struct Val {int v;};
typedef struct Val Value;
```

```
void *Add(void* arg){
    int x, val;
    val= ((Value *)arg)->v;
    x= S; x+= val; S= x;
    return NULL; }
```

(60)



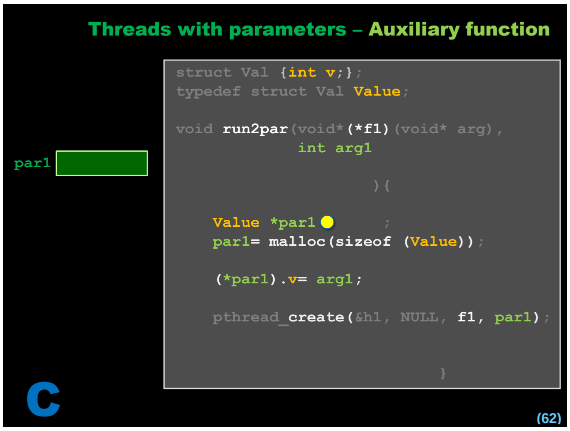
Threads with parameters

```
struct Val {int v;};
typedef struct Val Value;
```

```
void *Add(void* arg){
    int x, val;
    val= ((Value *)arg)->v;
    x= S; x+= val; S= x;
    return NULL; }
```

```
run2par(Add, 50, Add, 70);
```

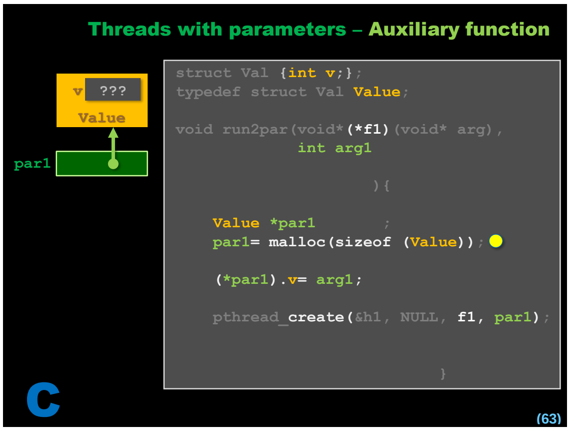
(61)



Threads with parameters – Auxiliary function

```
struct Val {int v;};
typedef struct Val Value;

void run2par(void* (*f1)(void* arg),
             int arg1

                     ){

    Value *par1 ●      ;
    par1= malloc(sizeof (Value));

    (*par1).v= arg1;

    pthread_create(&h1, NULL, f1, par1);

                               }
```

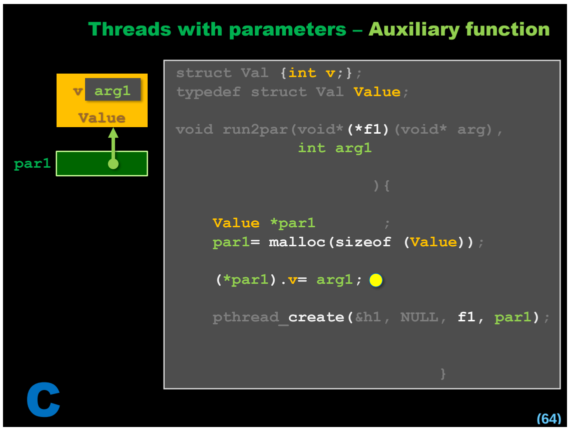
(62)



Threads with parameters – Auxiliary function

```
struct Val {int v;};
typedef struct Val Value;

void run2par(void* (*f1)(void* arg),
             int arg1

                     ){

    Value *par1        ;
    par1= malloc(sizeof (Value)); ●

    (*par1).v= arg1;

    pthread_create(&h1, NULL, f1, par1);

                               }
```

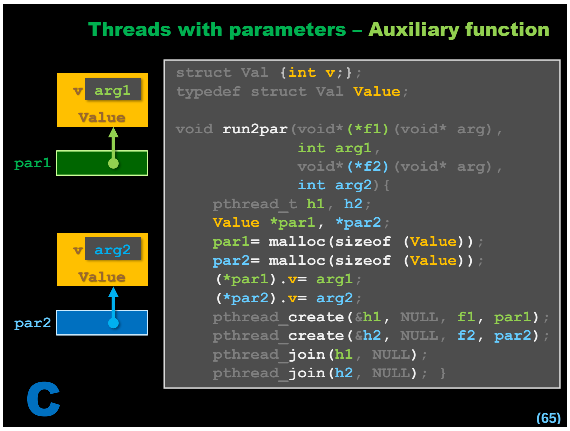
(63)



Threads with parameters – Auxiliary function

```
struct Val {int v;};
typedef struct Val Value;

void run2par(void* (*f1)(void* arg),
             int arg1

                     ){

    Value *par1        ;
    par1= malloc(sizeof (Value));

    (*par1).v= arg1; ●

    pthread_create(&h1, NULL, f1, par1);

                               }
```

(64)



Threads with parameters – Auxiliary function

```
struct Val {int v;};
typedef struct Val Value;

void run2par(void* (*f1)(void* arg),
             int arg1,
             void* (*f2)(void* arg),
             int arg2){
    pthread_t h1, h2;
    Value *par1, *par2;
    par1= malloc(sizeof (Value));
    par2= malloc(sizeof (Value));
    (*par1).v= arg1;
    (*par2).v= arg2;
    pthread_create(&h1, NULL, f1, par1);
    pthread_create(&h2, NULL, f2, par2);
    pthread_join(h1, NULL);
    pthread_join(h2, NULL); }
```

(65)

http://www.bhmpics.com/view-road_traffic-2560x1600.html

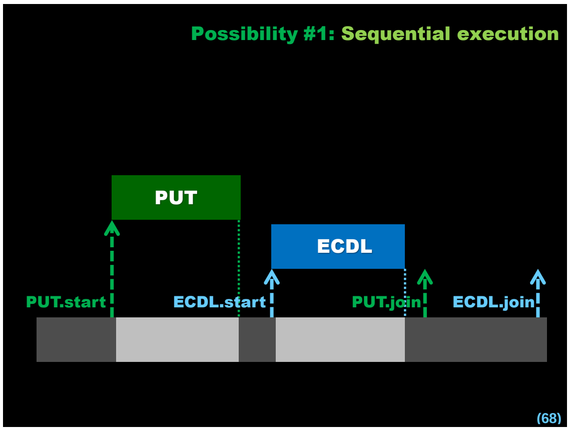
- Operating systems
- Parallel threads
- Indeterminism
- Semaphores
- Producer-consumer problem
- Readers-writers problem
- Deadlock

(66)

Parallelism implies indeterminism

- Sequential execution
- Interleaving

(67)



Possibility #1: Sequential execution

PUT

ECDL

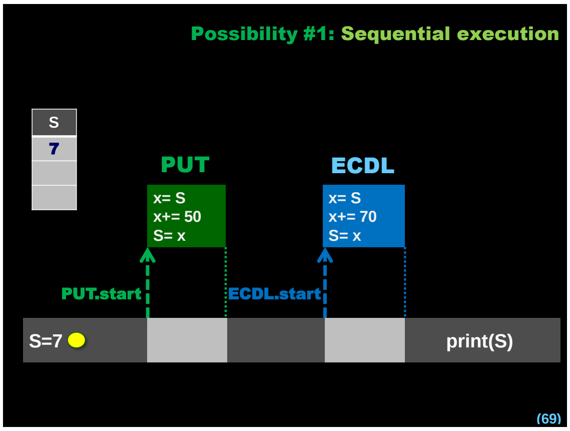
PUT.start

ECDL.start

PUT.join

ECDL.join

(68)



Possibility #1: Sequential execution

S

7

PUT

```
x= S
x+= 50
S= x
```

ECDL

```
x= S
x+= 70
S= x
```

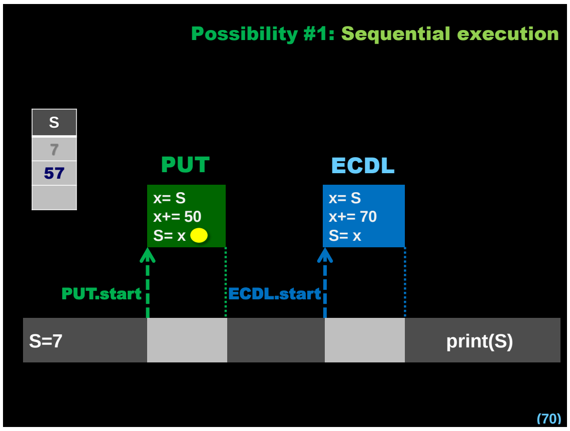
PUT.start

ECDL.start

S=7 ●

print(S)

(69)



Possibility #1: Sequential execution

S

7

57

PUT

```
x= S
x+= 50
S= x ●
```

ECDL

```
x= S
x+= 70
S= x
```

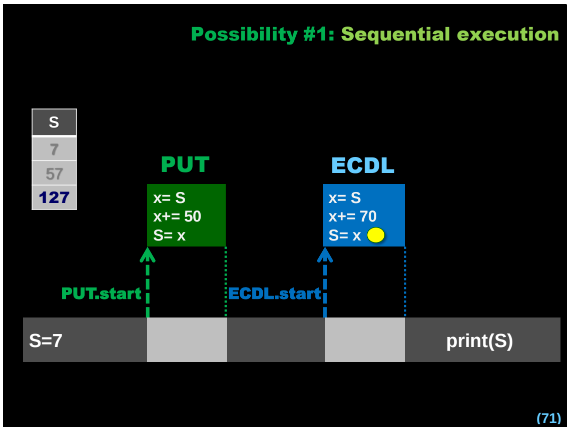
PUT.start

ECDL.start

S=7

print(S)

(70)



Possibility #1: Sequential execution

S

7

57

127

PUT

```
x= S
x+= 50
S= x
```

ECDL

```
x= S
x+= 70
S= x ●
```

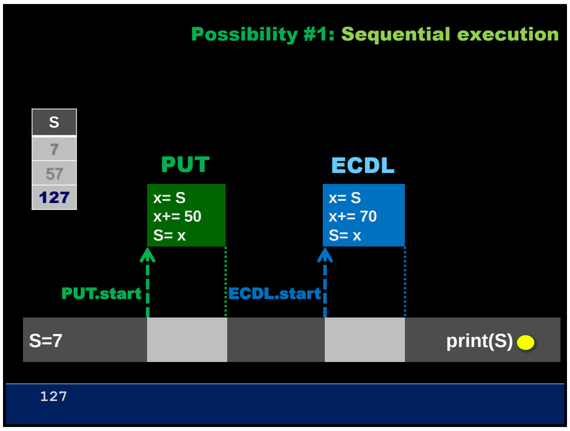
PUT.start

ECDL.start

S=7

print(S)

(71)



Possibility #1: Sequential execution

S

7

57

127

PUT

```
x= S
x+= 50
S= x
```

ECDL

```
x= S
x+= 70
S= x
```

PUT.start

ECDL.start

S=7

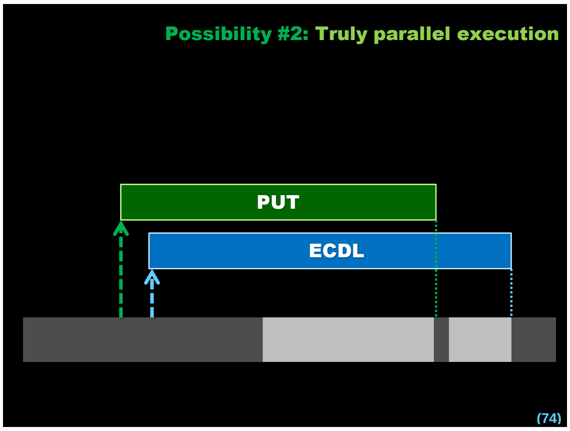
print(S) ●

```
127
```

Parallelism implies indeterminism

- Sequential execution
- Interleaving

(73)

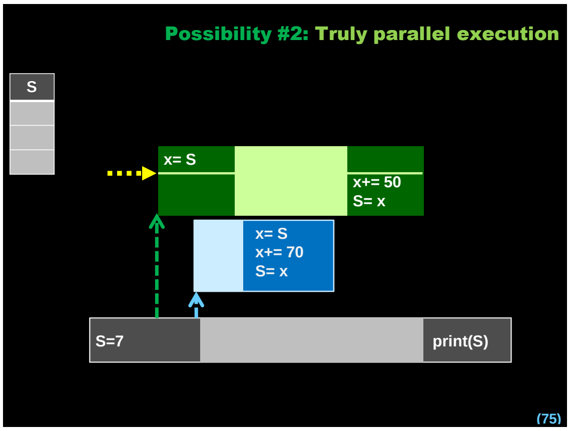


Possibility #2: Truly parallel execution

PUT

ECDL

(74)



Possibility #2: Truly parallel execution

S

x= S

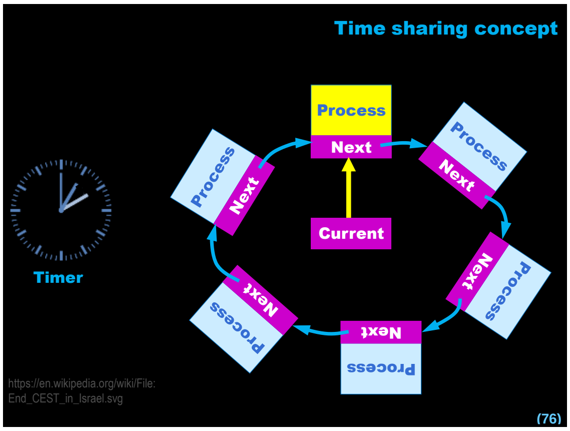
x+= 50
S= x

x= S
x+= 70
S= x

S=7

print(S)

(75)



Time sharing concept

Timer

Process
Next

Current

Process
Next

Process
Next

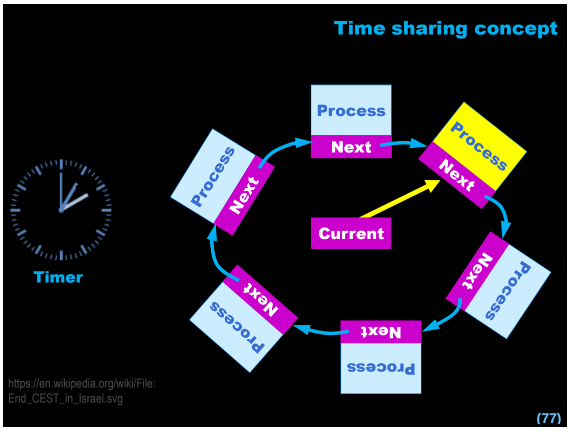
Process
Next

Process
Next

Process
Next

https://en.wikipedia.org/wiki/File:
End_CEST_in_Israel.svg

(76)



Time sharing concept

Timer

Process
Next

Current

Process
Next

Process
Next

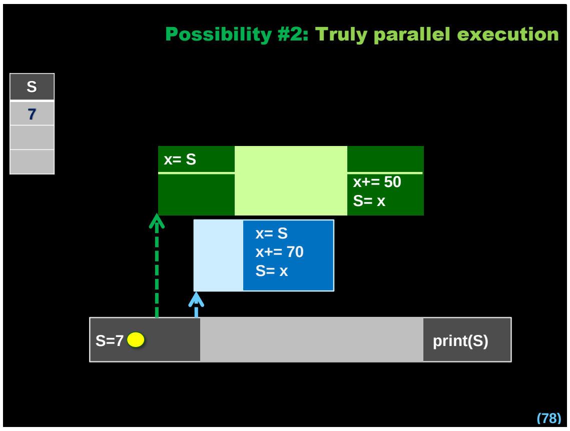
Process
Next

Process
Next

Process
Next

https://en.wikipedia.org/wiki/File:
End_CEST_in_Israel.svg

(77)



Possibility #2: Truly parallel execution

S
7

x= S

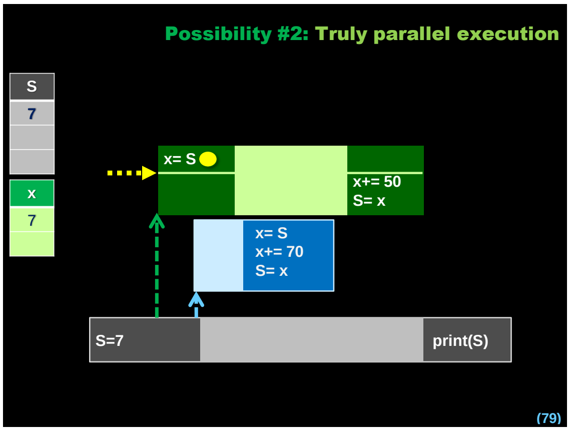
x+= 50
S= x

x= S
x+= 70
S= x

S=7

print(S)

(78)



Possibility #2: Truly parallel execution

S
7

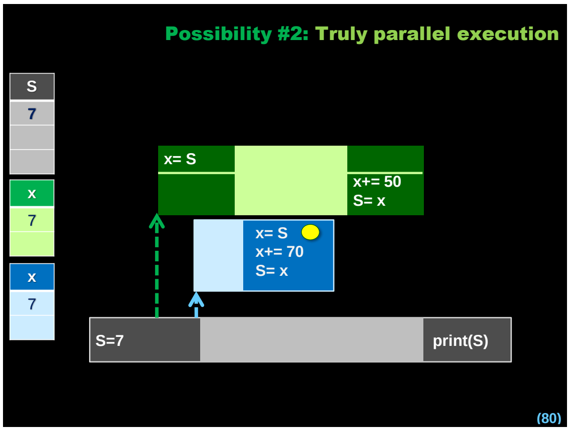
x
7

x= S ●
x+= 50
S= x

x= S
x+= 70
S= x

S=7
print(S)

(79)



Possibility #2: Truly parallel execution

S
7

x
7

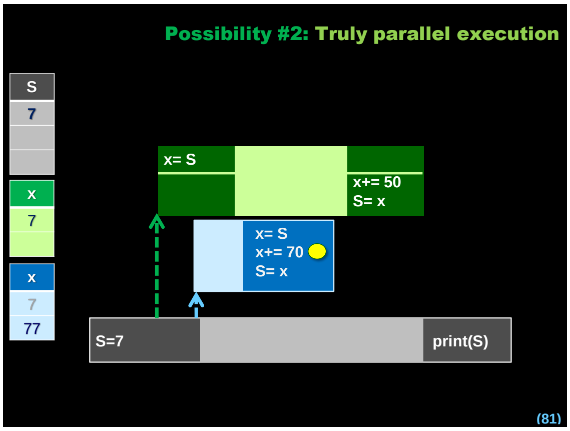
x
7

x= S
x+= 50
S= x

x= S ●
x+= 70
S= x

S=7
print(S)

(80)



Possibility #2: Truly parallel execution

S
7

x
7

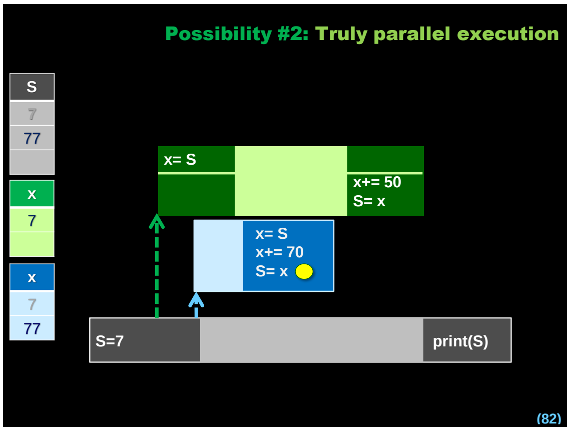
x
7
77

x= S
x+= 50
S= x

x= S
x+= 70 ●
S= x

S=7
print(S)

(81)



Possibility #2: Truly parallel execution

S
7
77

x
7

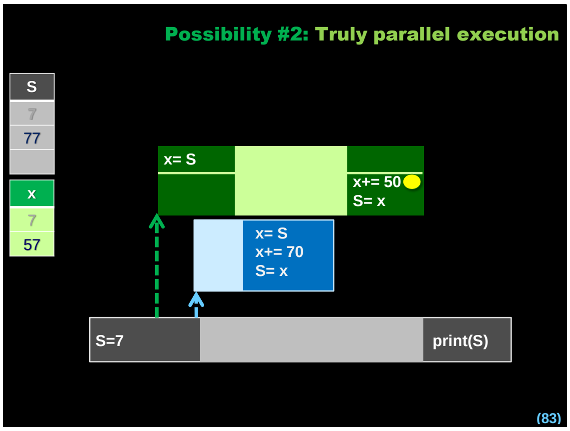
x
7
77

x= S
x+= 50
S= x

x= S
x+= 70
S= x ●

S=7
print(S)

(82)



Possibility #2: Truly parallel execution

S
7
77

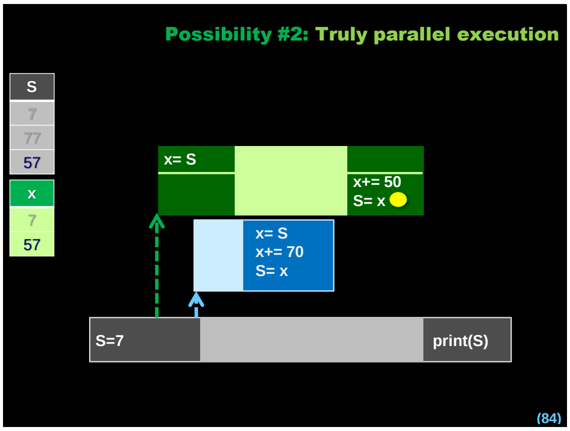
x
7
57

x= S
x+= 50 ●
S= x

x= S
x+= 70
S= x

S=7
print(S)

(83)



Possibility #2: Truly parallel execution

S
7
77
57

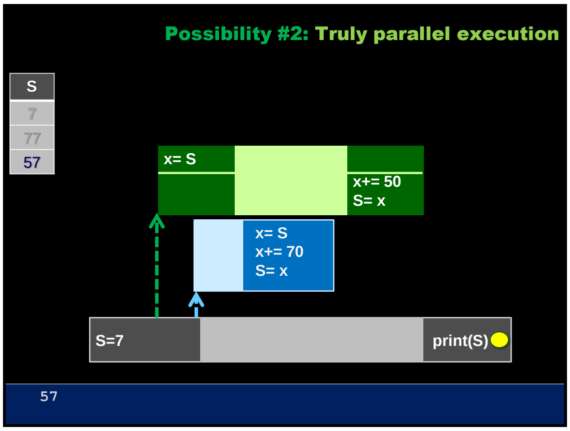
x
7
57

x= S
x+= 50
S= x ●

x= S
x+= 70
S= x

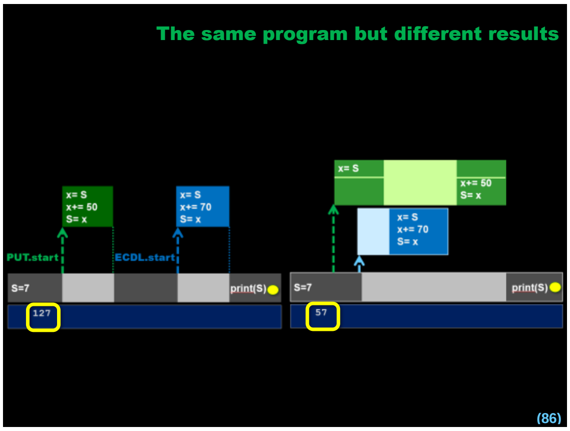
S=7
print(S)

(84)



Possibility #2: Truly parallel execution

```
57
```



The same program but different results

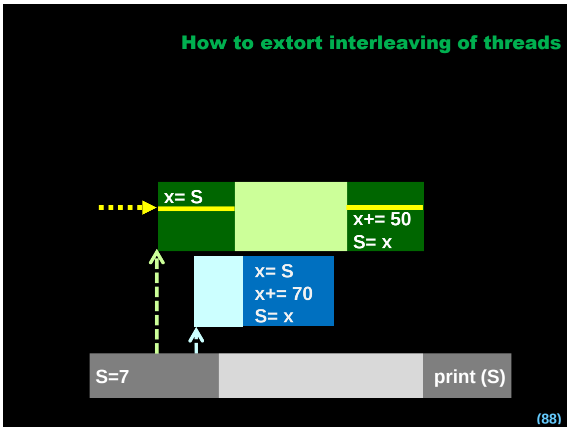
```
127
```

```
57
```

(86)

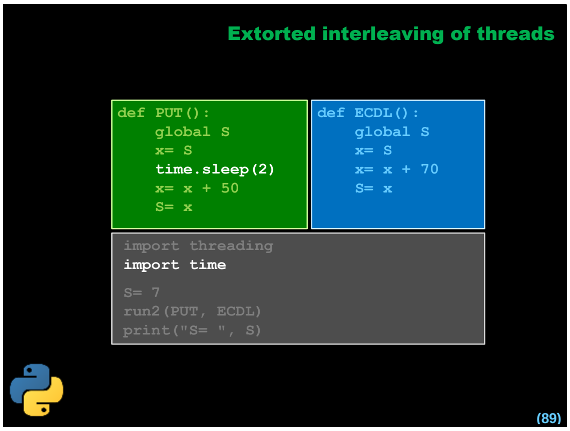
How to extort interleaving of threads?

(87)



How to extort interleaving of threads

(88)



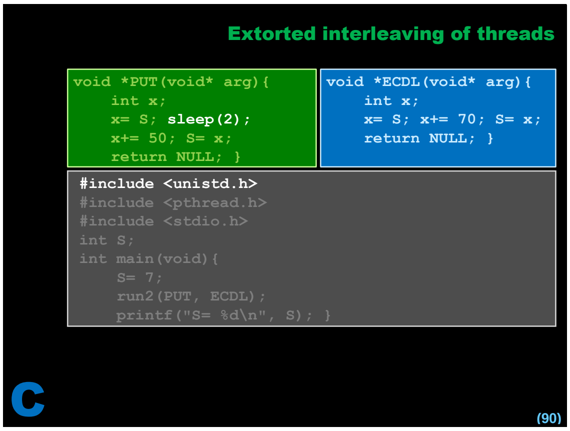
Extorted interleaving of threads

```
def PUT():
    global S
    x= S
    time.sleep(2)
    x= x + 50
    S= x
```

```
def ECDL():
    global S
    x= S
    x= x + 70
    S= x
```

```
import threading
import time

S= 7
run2(PUT, ECDL)
print("S= ", S)
```

(89)



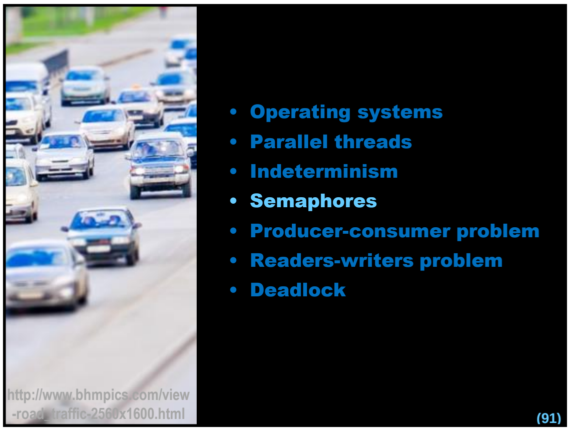
Extorted interleaving of threads

```
void *PUT(void* arg){
    int x;
    x= S; sleep(2);
    x+= 50; S= x;
    return NULL; }
```

```
void *ECDL(void* arg){
    int x;
    x= S; x+= 70; S= x;
    return NULL; }
```

```
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
int S;
int main(void){
    S= 7;
    run2(PUT, ECDL);
    printf("S= %d\n", S); }
```

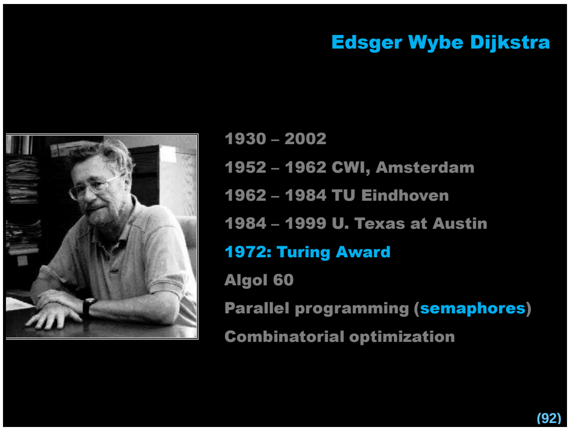
(90)



- Operating systems
- Parallel threads
- Indeterminism
- **Semaphores**
- Producer-consumer problem
- Readers-writers problem
- Deadlock

http://www.bhmpics.com/view-road-traffic-2560x1600.html

(91)



Edsger Wybe Dijkstra

1930 – 2002

1952 – 1962 CWI, Amsterdam

1962 – 1984 TU Eindhoven

1984 – 1999 U. Texas at Austin

1972: Turing Award

Algol 60

Parallel programming (semaphores)

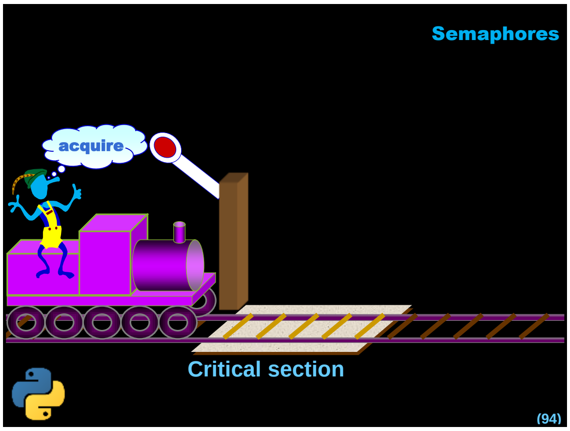
Combinatorial optimization

(92)

Critical section

https://www.istockphoto.com

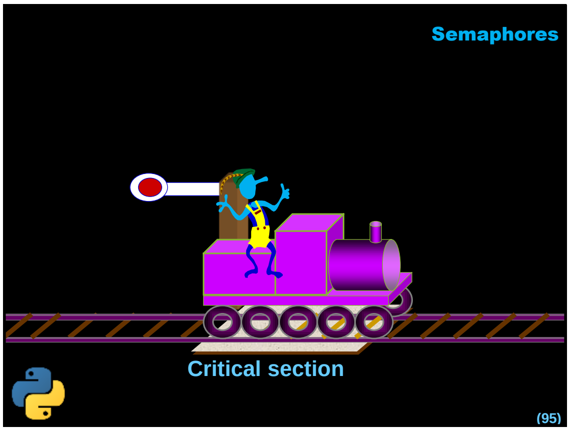
(93)



Semaphores

Critical section

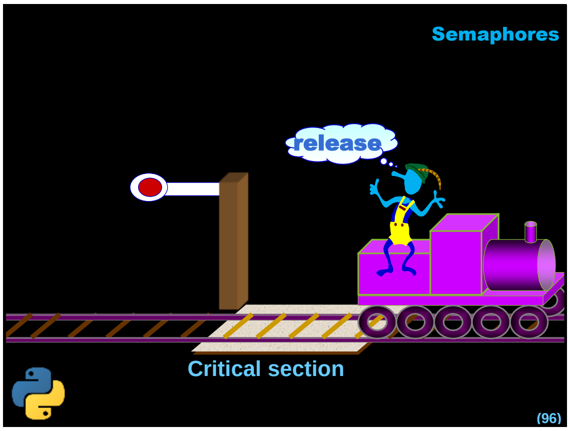
(94)



Semaphores

Critical section

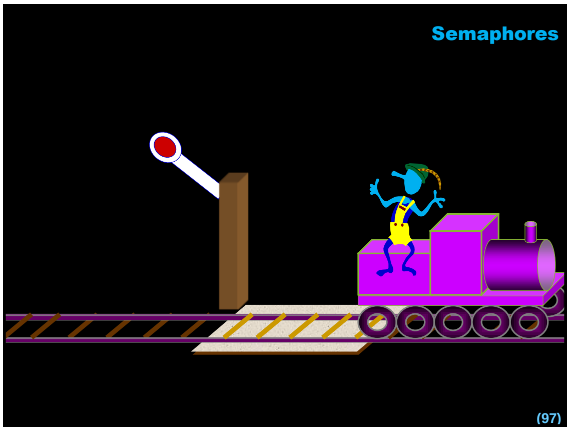
(95)



Semaphores

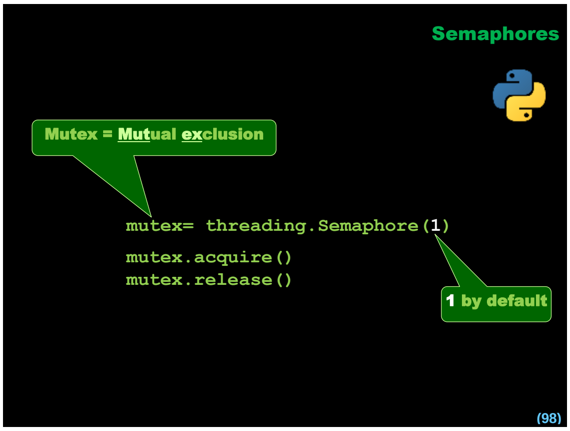
Critical section

(96)



Semaphores

(97)



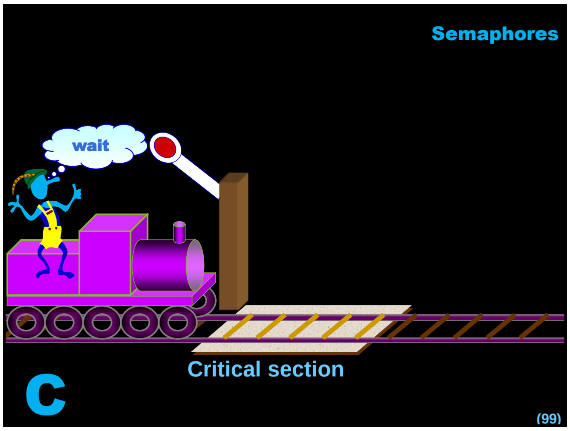
Semaphores

Mutex = Mutual exclusion

```
mutex= threading.Semaphore(1)
mutex.acquire()
mutex.release()
```

1 by default

(98)

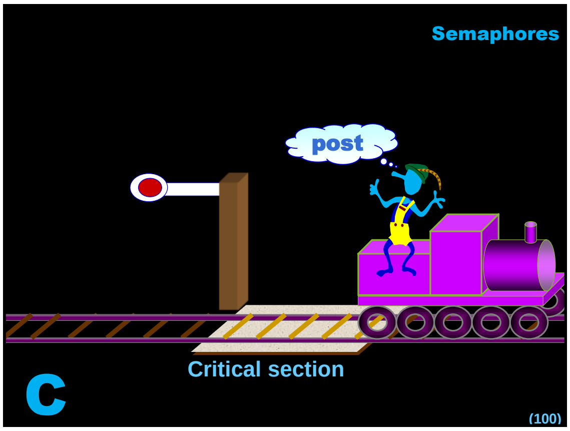


Semaphores

C

Critical section

(99)

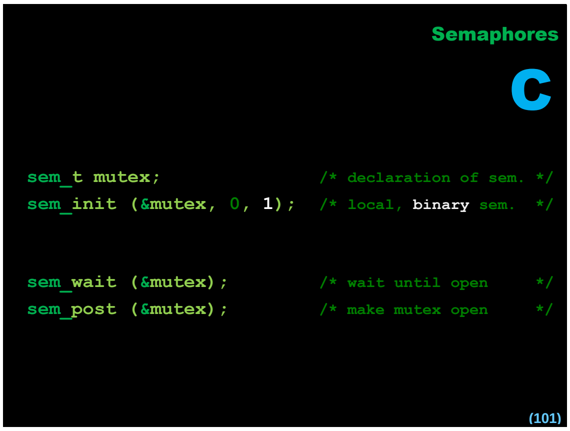


Semaphores

C

Critical section

(100)

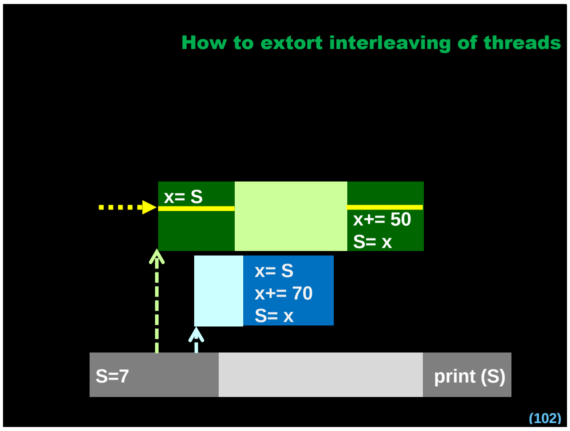


Semaphores

C

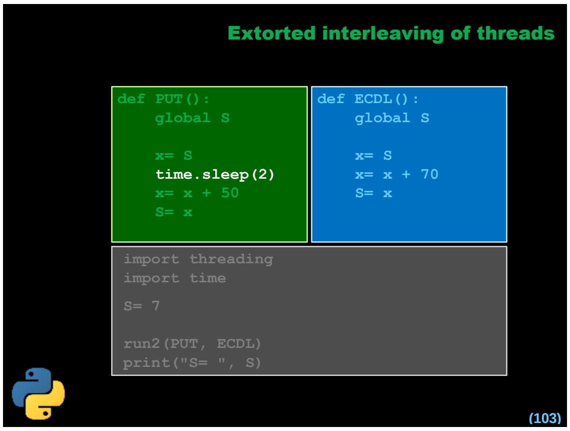
```
sem_t mutex;                /* declaration of sem. */
sem_init (&mutex, 0, 1);    /* local, binary sem.  */


sem_wait (&mutex);          /* wait until open     */
sem_post (&mutex);          /* make mutex open     */
```

(101)



How to extort interleaving of threads

(102)



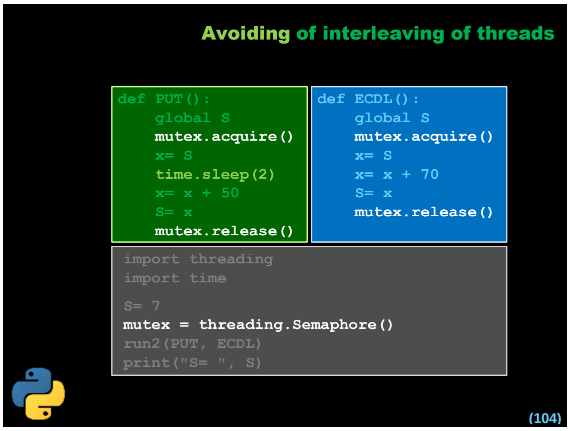
Extorted interleaving of threads

```
def PUT():
    global S

    x= S
    time.sleep(2)
    x= x + 50
    S= x
```

```
def ECDL():
    global S

    x= S
    x= x + 70
    S= x
```

```
import threading
import time

S= 7

run2(PUT, ECDL)
print("S= ", S)
```

(103)



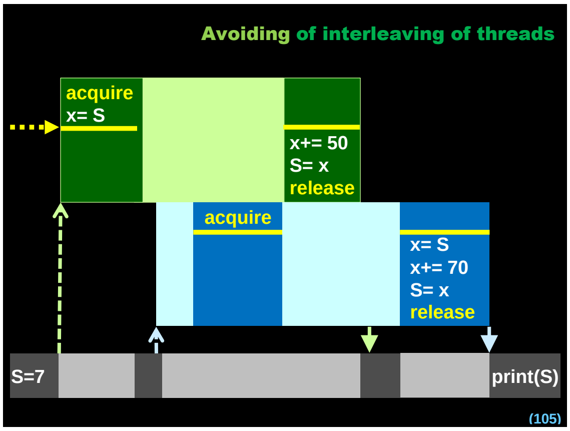
Avoiding of interleaving of threads

```
def PUT():
    global S
    mutex.acquire()
    x= S
    time.sleep(2)
    x= x + 50
    S= x
    mutex.release()
```

```
def ECDL():
    global S
    mutex.acquire()
    x= S
    x= x + 70
    S= x
    mutex.release()
```

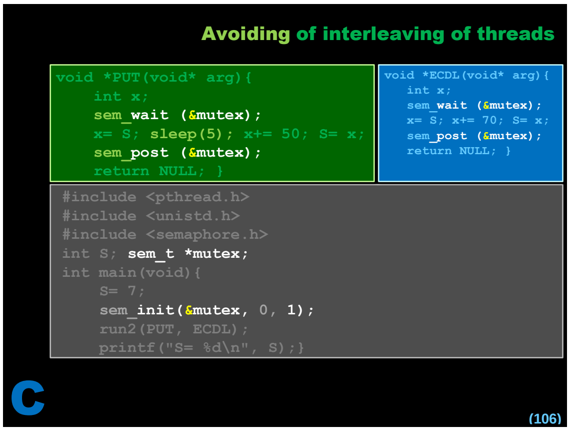
```
import threading
import time

S= 7
mutex = threading.Semaphore()
run2(PUT, ECDL)
print("S= ", S)
```

(104)



Avoiding of interleaving of threads

(105)



Avoiding of interleaving of threads

```
void *PUT(void* arg){
    int x;
    sem_wait (&mutex);
    x= S; sleep(5); x+= 50; S= x;
    sem_post (&mutex);
    return NULL; }
```

```
void *ECDL(void* arg){
    int x;
    sem_wait (&mutex);
    x= S; x+= 70; S= x;
    sem_post (&mutex);
    return NULL; }
```

```
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>
int S; sem_t *mutex;
int main(void){
    S= 7;
    sem_init(&mutex, 0, 1);
    run2(PUT, ECDL);
    printf("S= %d\n", S);}
```

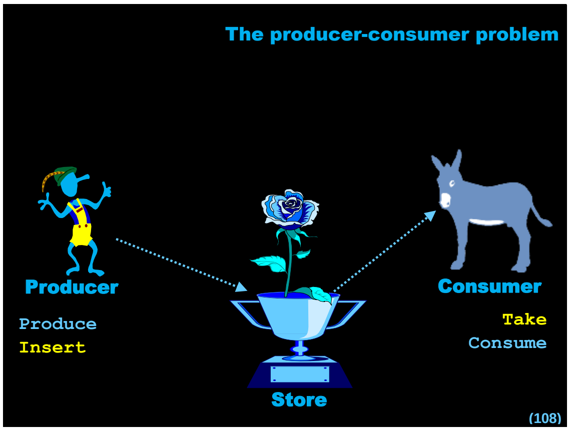
C

(106)

- Operating systems
- Parallel threads
- Indeterminism
- Semaphores
- **Producer-consumer problem**
- Readers-writers problem
- Deadlock

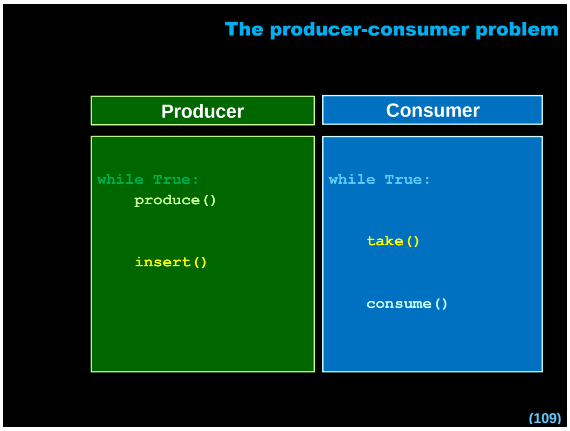
http://www.bhmpics.com/view-road-traffic-2560x1600.html

(107)



The producer-consumer problem

(108)



The producer-consumer problem

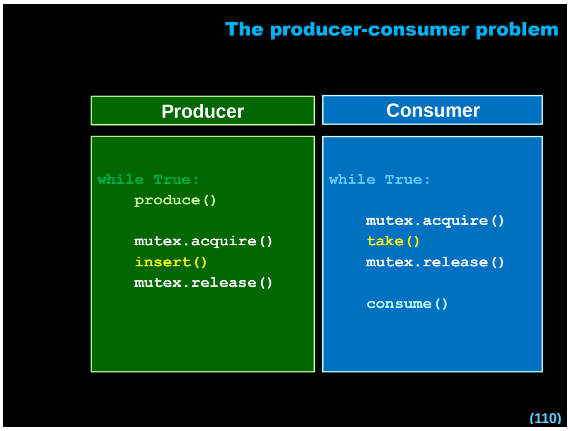
Producer

```
while True:
    produce()

    insert()
```

Consumer

```
while True:

    take()

    consume()
```

(109)



The producer-consumer problem

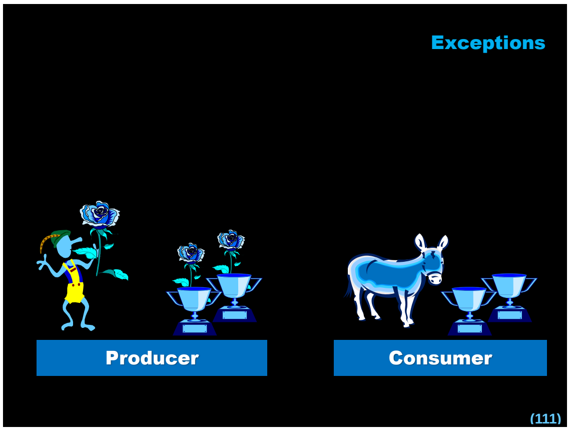
Producer

```
while True:
    produce()

    mutex.acquire()
    insert()
    mutex.release()
```

Consumer

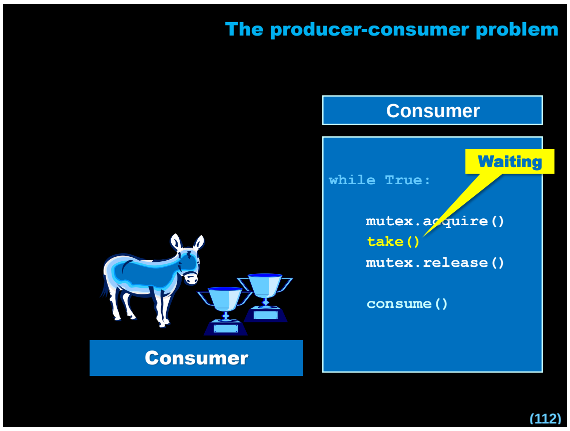
```
while True:

    mutex.acquire()
    take()
    mutex.release()

    consume()
```

(110)



Exceptions

(111)

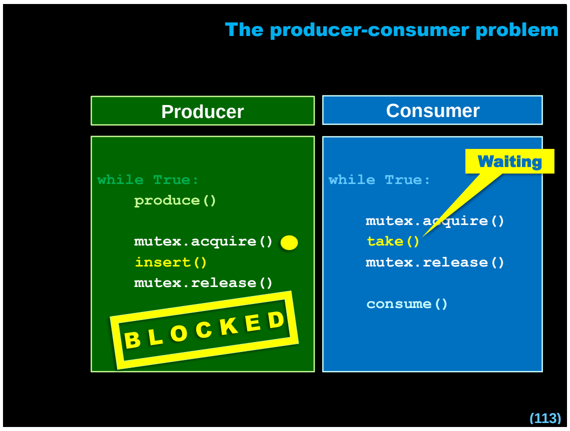


The producer-consumer problem

Consumer

```
while True:

    mutex.acquire()
    take()
    mutex.release()

    consume()
```

(112)



The producer-consumer problem

Waiting
BLOCKED

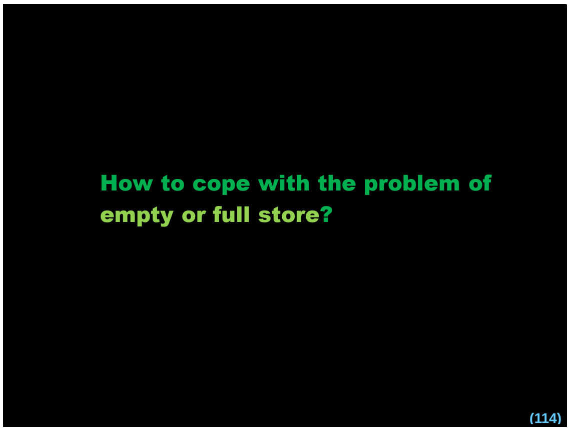
Producer

```
while True:
    produce()

    mutex.acquire()
    insert()
    mutex.release()
```

Consumer

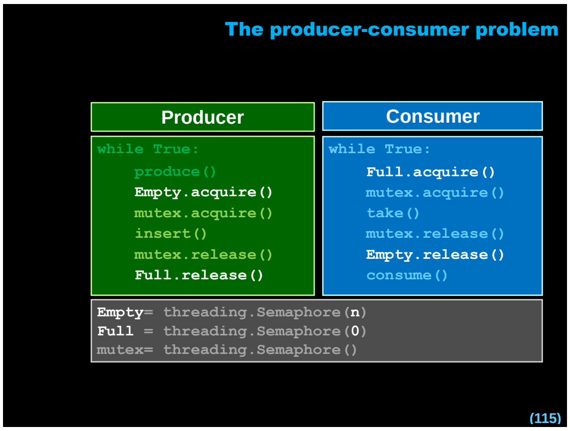
```
while True:

    mutex.acquire()
    take()
    mutex.release()

    consume()
```

(113)



How to cope with the problem of empty or full store?

(114)



The producer-consumer problem

Producer	Consumer

`while True:`
`    produce()`
`    Empty.acquire()`
`    mutex.acquire()`
`    insert()`
`    mutex.release()`
`    Full.release()`
	`while True:`
	`    Full.acquire()`
	`    mutex.acquire()`
	`    take()`
	`    mutex.release()`
	`    Empty.release()`
	`    consume()`

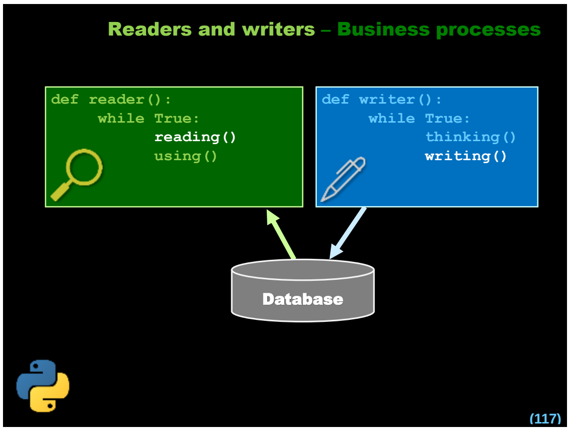
```
Empty= threading.Semaphore(n)
Full  = threading.Semaphore(0)
mutex= threading.Semaphore()
```

(115)

http://www.bhmpics.com/view-road_traffic-2560x1600.html

- Operating systems
- Parallel threads
- Indeterminism
- Semaphores
- Producer-consumer problem
- Readers-writers problem
- Deadlock

(116)

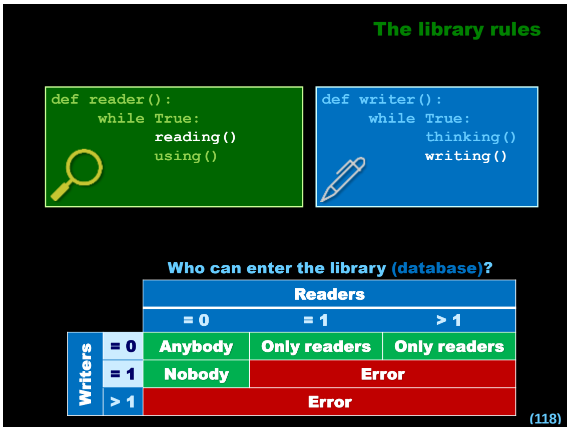


Readers and writers – Business processes

```
def reader():
    while True:
        reading()
        using()
```

```
def writer():
    while True:
        thinking()
        writing()
```

(117)



The library rules

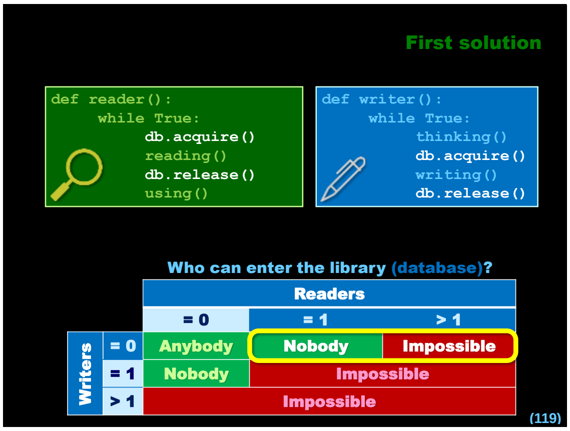
```
def reader():
    while True:
        reading()
        using()
```

```
def writer():
    while True:
        thinking()
        writing()
```

Who can enter the library (database)?

		Readers		
		= 0	= 1	> 1
Writers	= 0	Anybody	Only readers	Only readers
	= 1	Nobody	Error	
	> 1	Error		

(118)



First solution

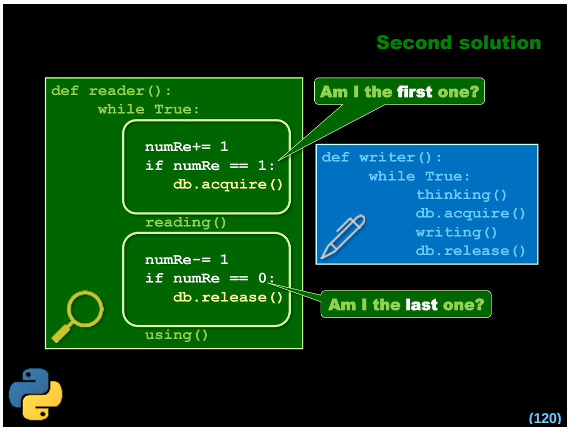
```
def reader():
    while True:
        db.acquire()
        reading()
        db.release()
        using()
```

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

Who can enter the library (database)?

		Readers		
		= 0	= 1	> 1
Writers	= 0	Anybody	Nobody	Impossible
	= 1	Nobody	Impossible	
	> 1	Impossible		

(119)



Second solution

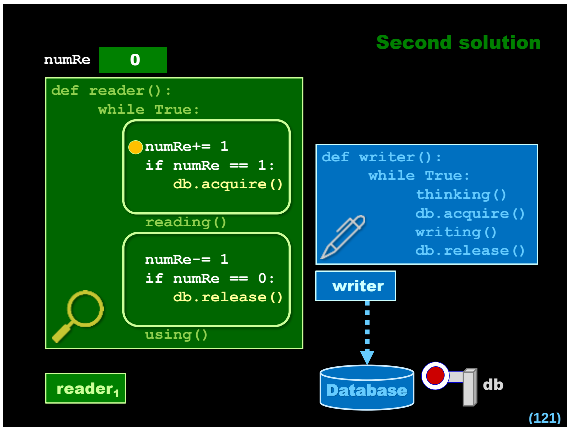
```
def reader():
    while True:
        numRe+= 1
        if numRe == 1:
            db.acquire()
        reading()
        numRe-= 1
        if numRe == 0:
            db.release()
        using()
```

Am I the first one?

Am I the last one?

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

(120)



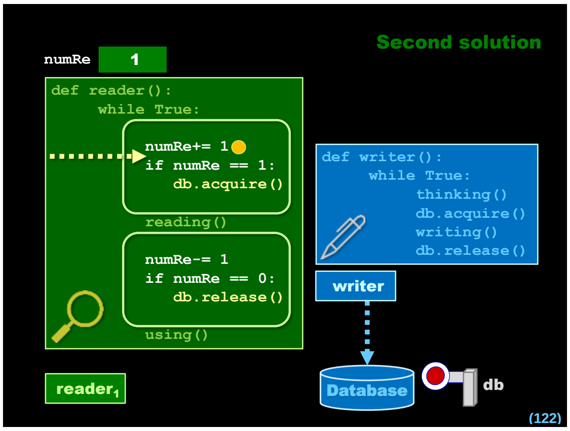
Second solution

numRe 0

```
def reader():
    while True:
        numRe+= 1
        if numRe == 1:
            db.acquire()
        reading()
        numRe-= 1
        if numRe == 0:
            db.release()
        using()
```

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

(121)



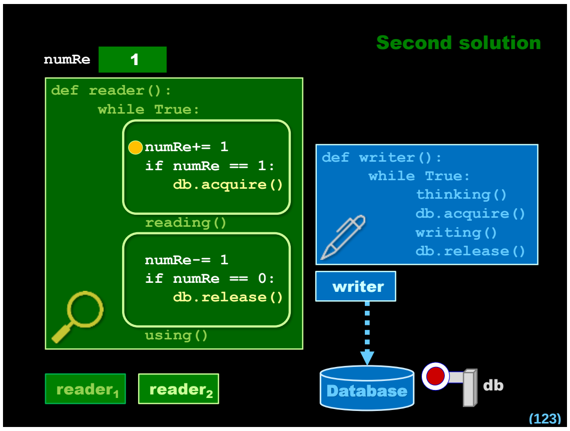
Second solution

numRe 1

```
def reader():
    while True:
        numRe+= 1
        if numRe == 1:
            db.acquire()
        reading()
        numRe-= 1
        if numRe == 0:
            db.release()
        using()
```

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

(122)



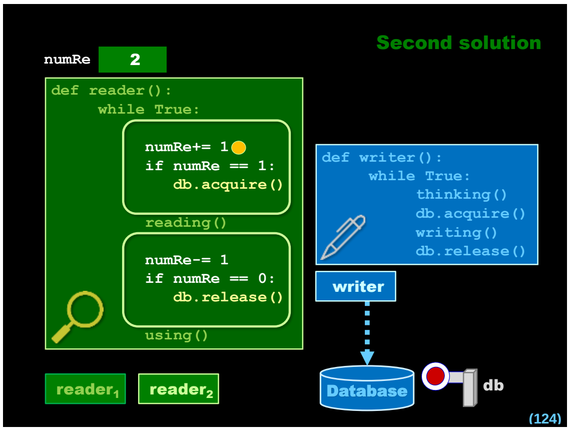
Second solution

numRe 1

```
def reader():
    while True:
        numRe+= 1
        if numRe == 1:
            db.acquire()
        reading()
        numRe-= 1
        if numRe == 0:
            db.release()
        using()
```

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

(123)



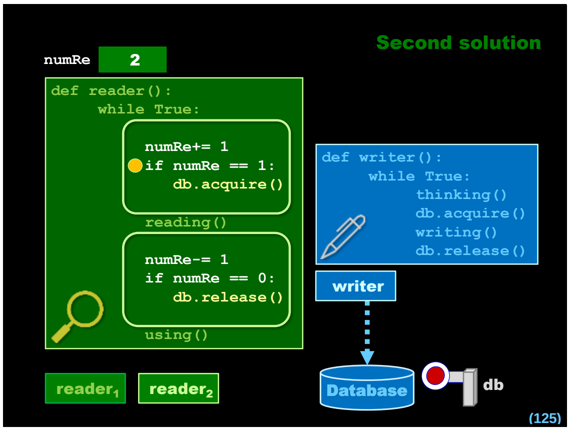
Second solution

numRe 2

```
def reader():
    while True:
        numRe+= 1
        if numRe == 1:
            db.acquire()
        reading()
        numRe-= 1
        if numRe == 0:
            db.release()
        using()
```

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

(124)



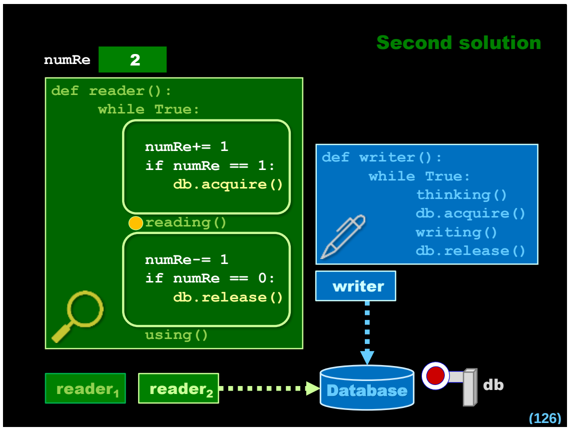
Second solution

numRe 2

```
def reader():
    while True:
        numRe+= 1
        if numRe == 1:
            db.acquire()
        reading()
        numRe-= 1
        if numRe == 0:
            db.release()
        using()
```

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

(125)



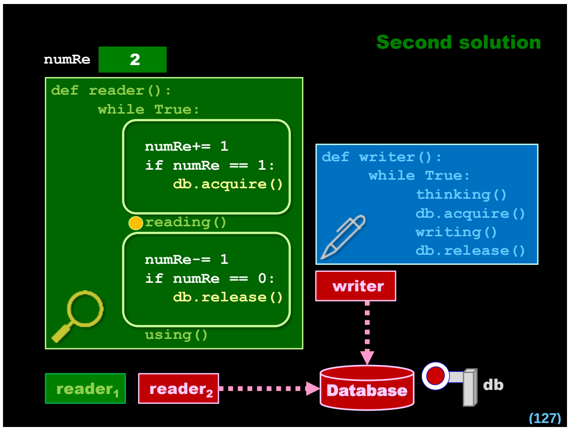
Second solution

numRe 2

```
def reader():
    while True:
        numRe+= 1
        if numRe == 1:
            db.acquire()
        reading()
        numRe-= 1
        if numRe == 0:
            db.release()
        using()
```

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

(126)

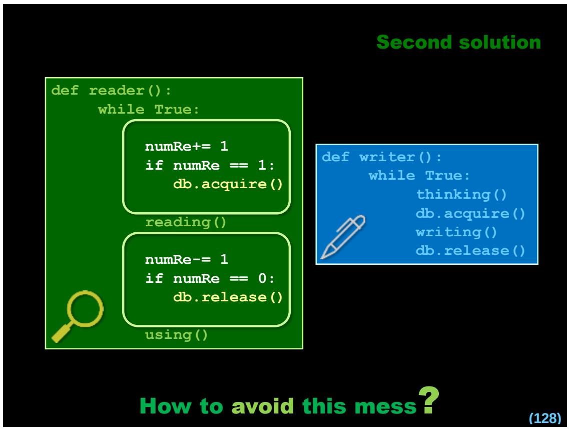


Second solution

```
def reader():
    while True:
        numRe+= 1
        if numRe == 1:
            db.acquire()
        reading()
        numRe-= 1
        if numRe == 0:
            db.release()
        using()
```

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

(127)



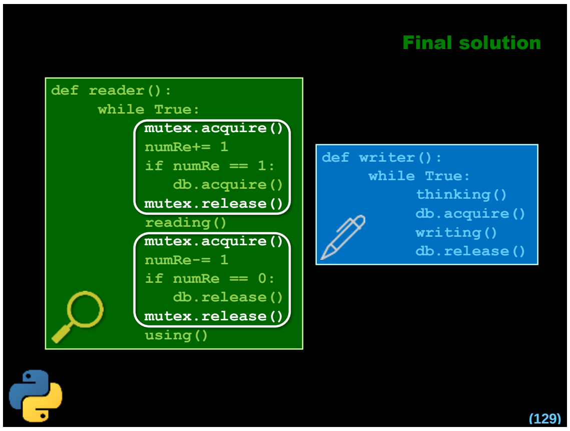
Second solution

```
def reader():
    while True:
        numRe+= 1
        if numRe == 1:
            db.acquire()
        reading()
        numRe-= 1
        if numRe == 0:
            db.release()
        using()
```

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

How to avoid this mess?

(128)

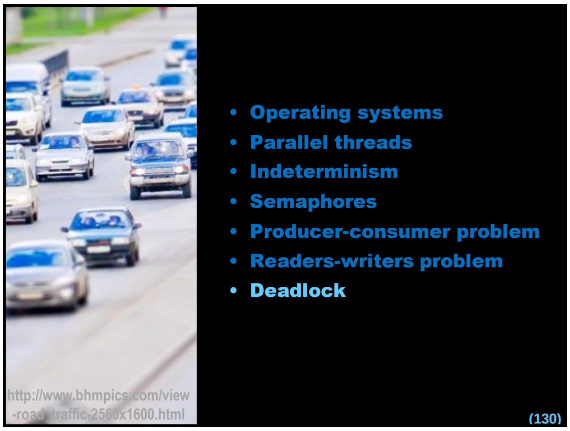


Final solution

```
def reader():
    while True:
        mutex.acquire()
        numRe+= 1
        if numRe == 1:
            db.acquire()
        mutex.release()
        reading()
        mutex.acquire()
        numRe-= 1
        if numRe == 0:
            db.release()
        mutex.release()
        using()
```

```
def writer():
    while True:
        thinking()
        db.acquire()
        writing()
        db.release()
```

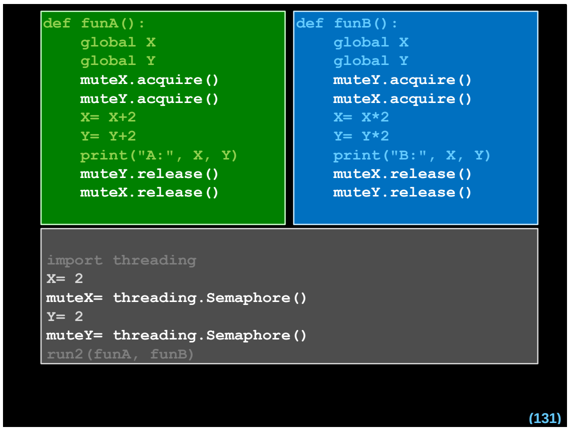
(129)



http://www.bhmpics.com/view-road-traffic-2560x1600.html

- Operating systems
- Parallel threads
- Indeterminism
- Semaphores
- Producer-consumer problem
- Readers-writers problem
- Deadlock

(130)

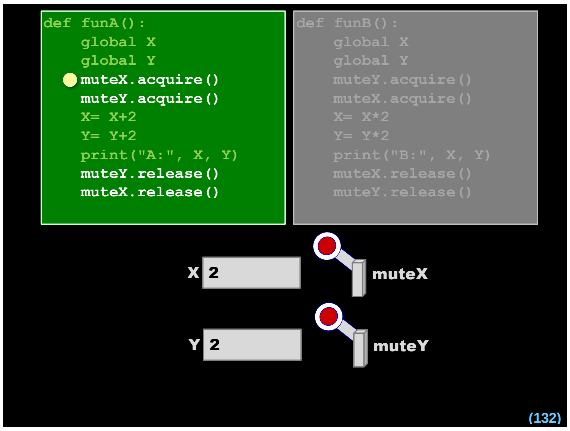


```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

```
import threading
X= 2
muteX= threading.Semaphore()
Y= 2
muteY= threading.Semaphore()
run2(funA, funB)
```

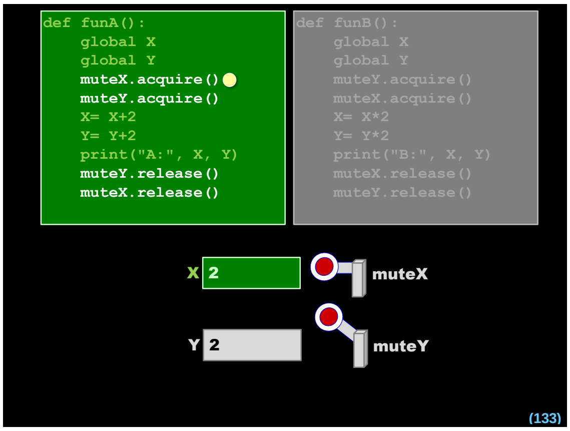
(131)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

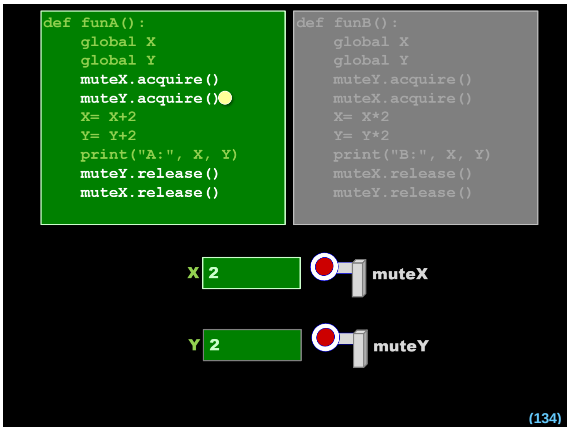
(132)



```
def funA():
    global X
    global Y
    muteX.acquire() ●
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

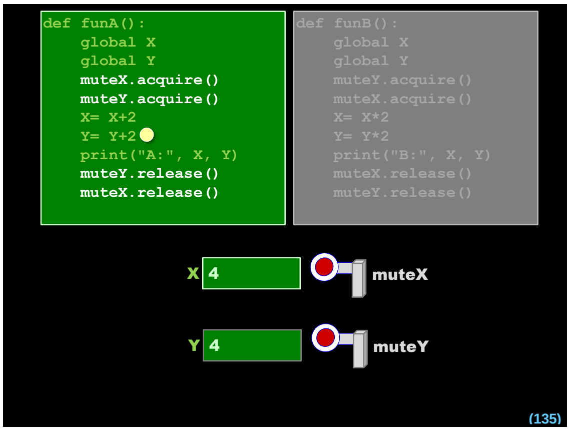
(133)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire() ●
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

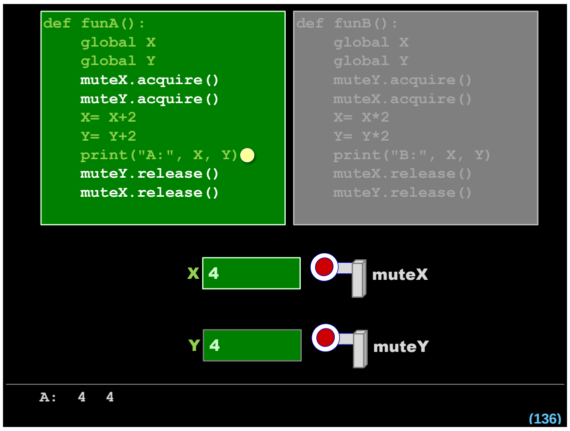
(134)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2 ●
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

(135)

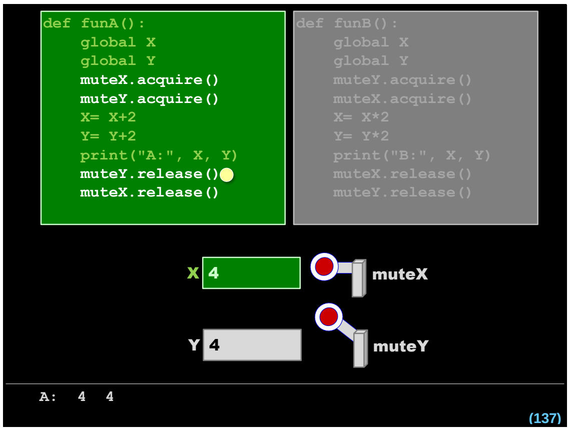


```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y) ●
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

```
A:  4  4
```

(136)

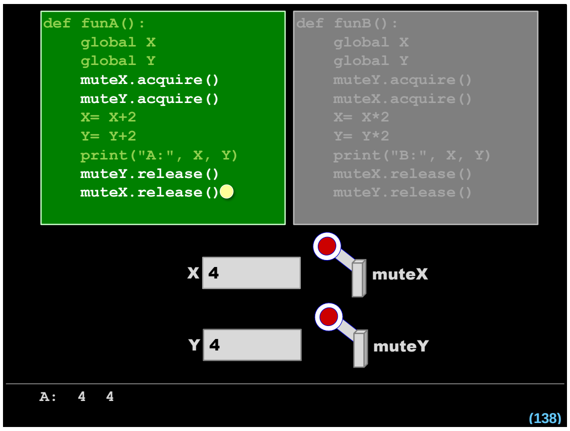


```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release() ●
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

```
A:  4  4
```

(137)

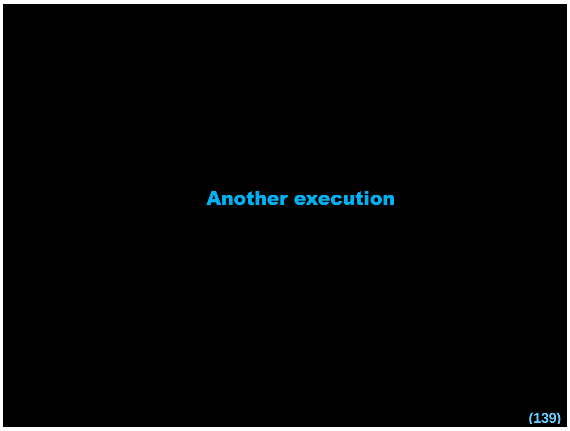


```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release() ●
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

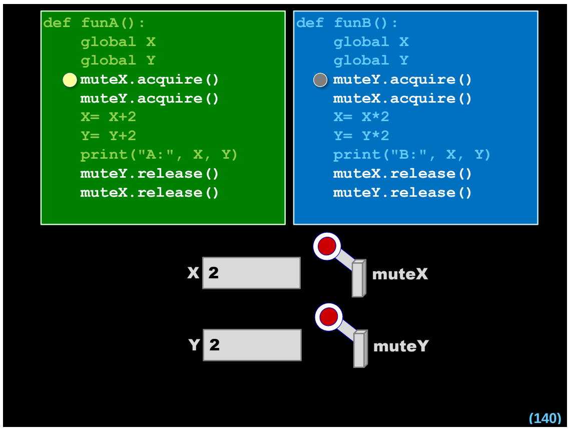
```
A:  4  4
```

(138)



Another execution

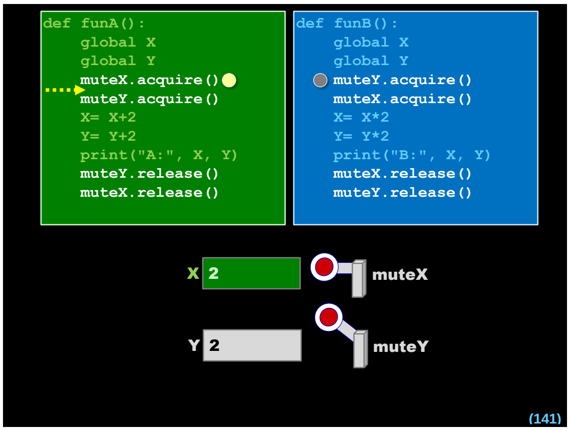
(139)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

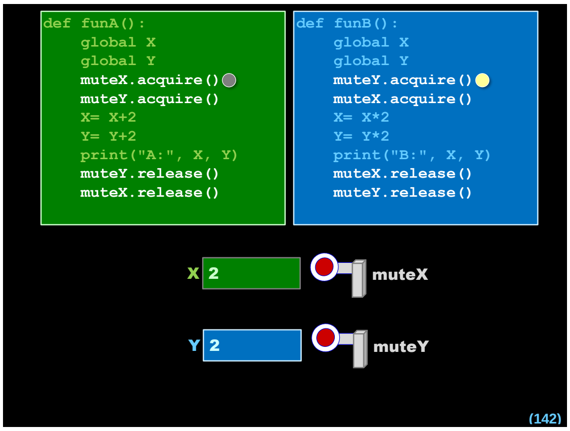
(140)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

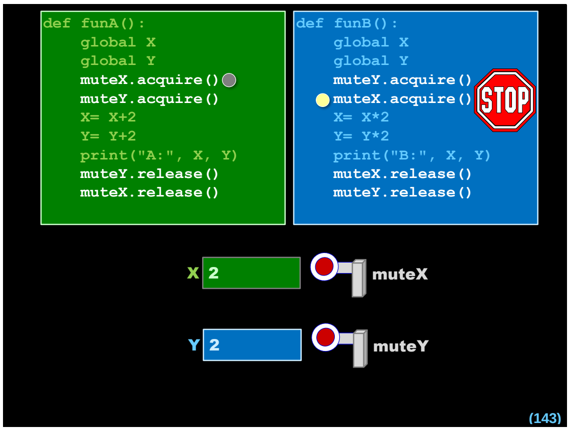
(141)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

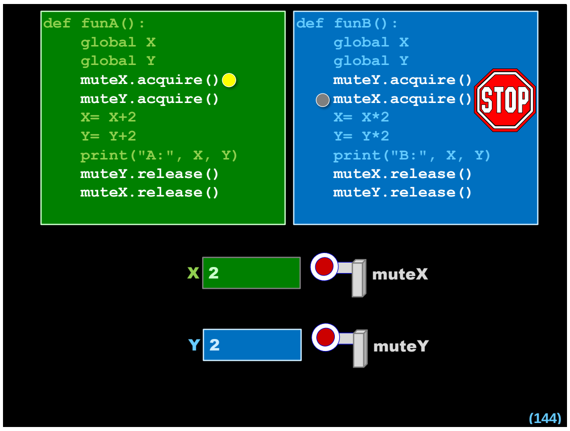
(142)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

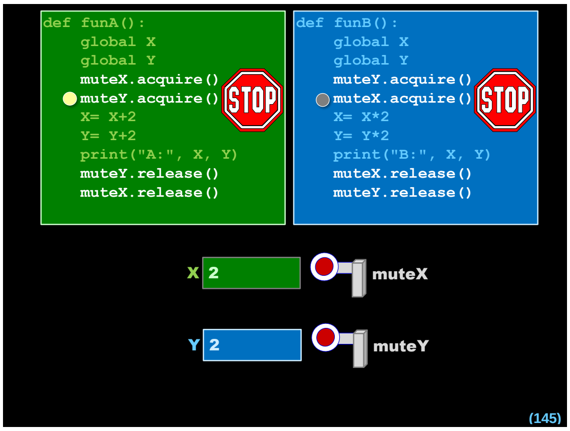
(143)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

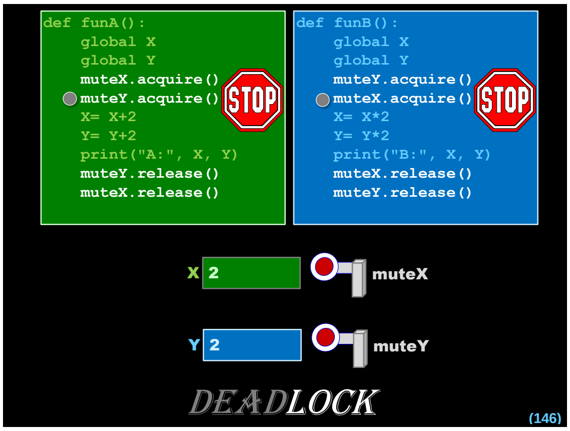
(144)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

(145)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

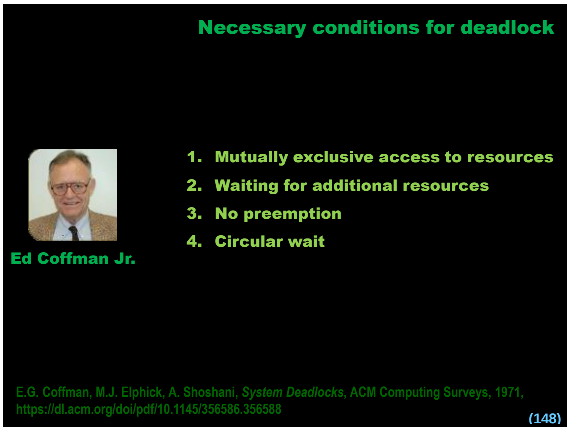
DEADLOCK

(146)

?

How to cope with deadlock?

(147)



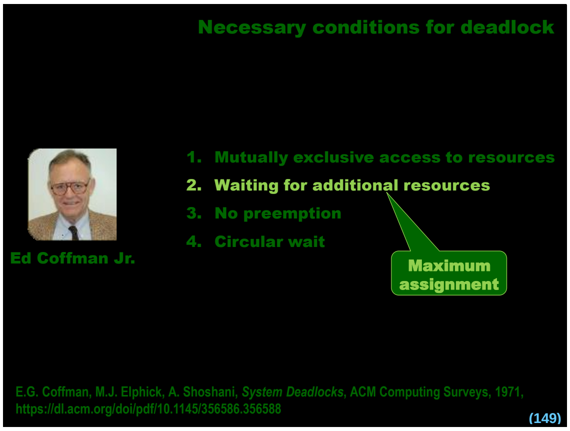
Necessary conditions for deadlock

Ed Coffman Jr.

1. Mutually exclusive access to resources
2. Waiting for additional resources
3. No preemption
4. Circular wait

E.G. Coffman, M.J. Elphick, A. Shoshani, *System Deadlocks*, ACM Computing Surveys, 1971, https://dl.acm.org/doi/pdf/10.1145/356586.356588

(148)



Necessary conditions for deadlock

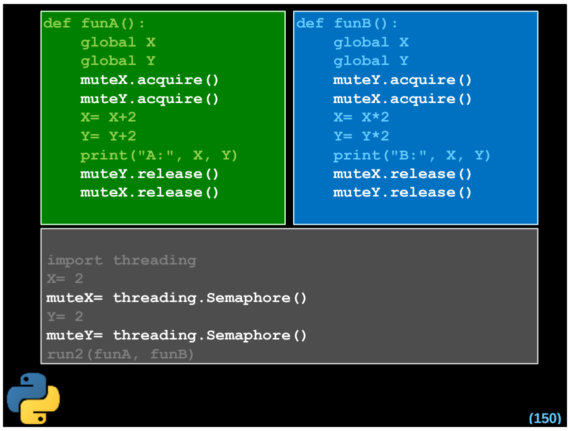
Ed Coffman Jr.

1. Mutually exclusive access to resources
2. Waiting for additional resources
3. No preemption
4. Circular wait

Maximum assignment

E.G. Coffman, M.J. Elphick, A. Shoshani, *System Deadlocks*, ACM Computing Surveys, 1971, https://dl.acm.org/doi/pdf/10.1145/356586.356588

(149)

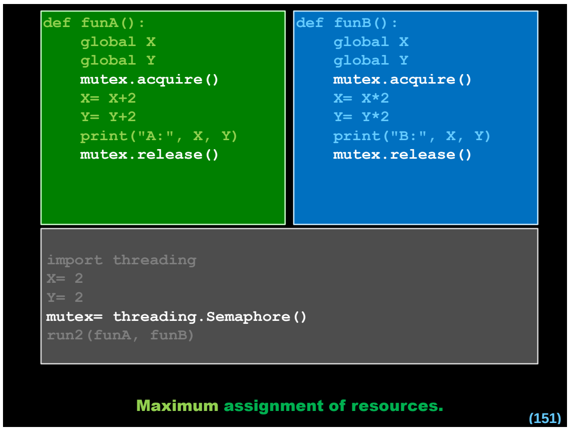


```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

```
import threading
X= 2
muteX= threading.Semaphore()
Y= 2
muteY= threading.Semaphore()
run2(funA, funB)
```

(150)



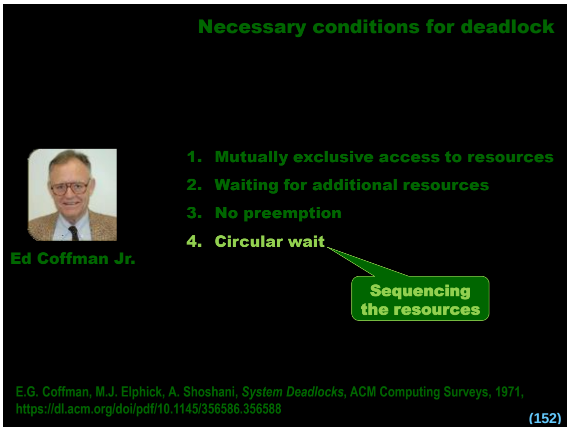
```
def funA():
    global X
    global Y
    mutex.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    mutex.release()
```

```
def funB():
    global X
    global Y
    mutex.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    mutex.release()
```

```
import threading
X= 2
Y= 2
mutex= threading.Semaphore()
run2(funA, funB)
```

Maximum assignment of resources.

(151)



Necessary conditions for deadlock

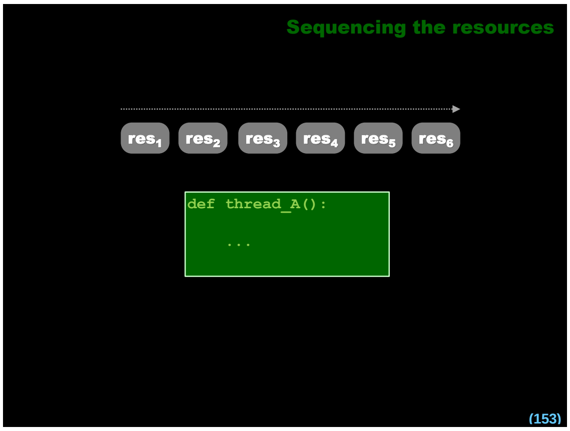
Ed Coffman Jr.

1. Mutually exclusive access to resources
2. Waiting for additional resources
3. No preemption
4. Circular wait

Sequencing the resources

E.G. Coffman, M.J. Elphick, A. Shoshani, *System Deadlocks*, ACM Computing Surveys, 1971, https://dl.acm.org/doi/pdf/10.1145/356586.356588

(152)

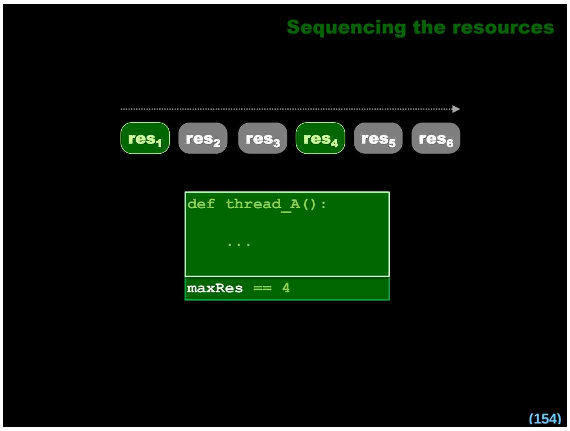


Sequencing the resources

res$_1$ res$_2$ res$_3$ res$_4$ res$_5$ res$_6$

```
def thread_A():
    ...
```

(153)

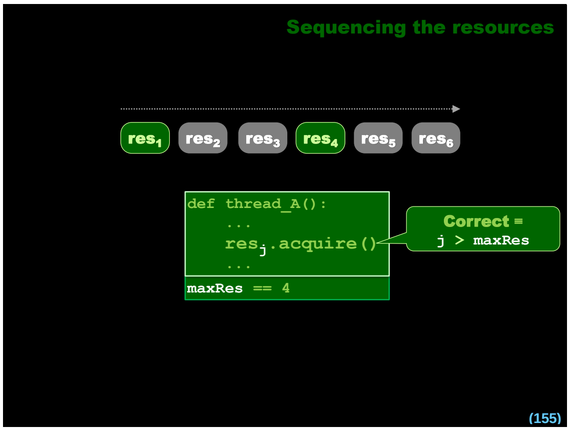


Sequencing the resources

res$_1$ res$_2$ res$_3$ res$_4$ res$_5$ res$_6$

```
def thread_A():
    ...

maxRes == 4
```

(154)



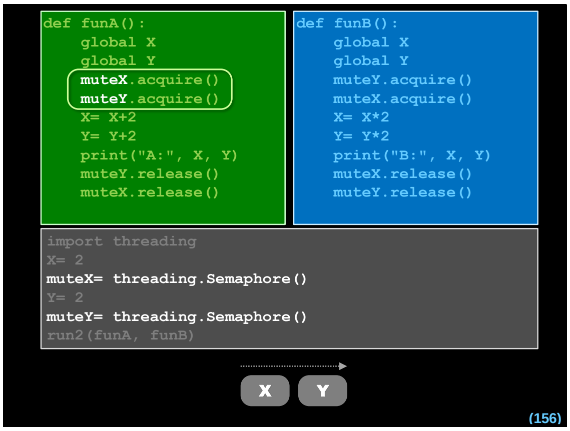
Sequencing the resources

res$_1$ res$_2$ res$_3$ res$_4$ res$_5$ res$_6$

```
def thread_A():
    ...
    res_j.acquire()
    ...
maxRes == 4
```

Correct = j > maxRes

(155)



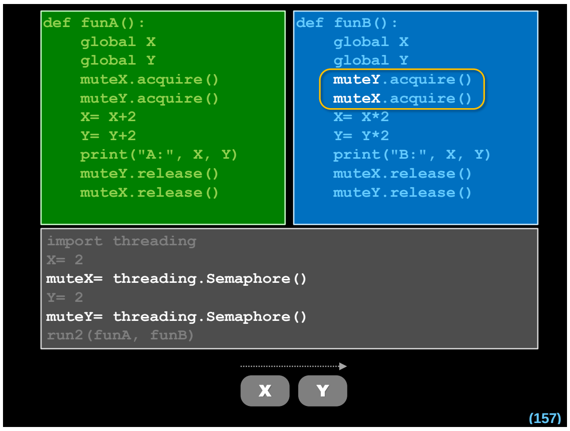
```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

```
import threading
X= 2
muteX= threading.Semaphore()
Y= 2
muteY= threading.Semaphore()
run2(funA, funB)
```

X Y

(156)

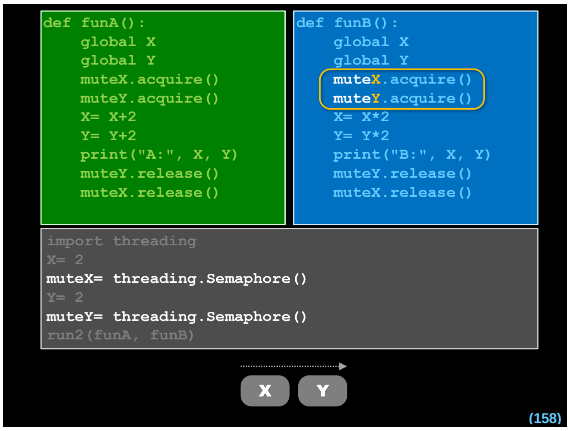


```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

```
import threading
X= 2
muteX= threading.Semaphore()
Y= 2
muteY= threading.Semaphore()
run2(funA, funB)
```

(157)



```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteY.release()
    muteX.release()
```

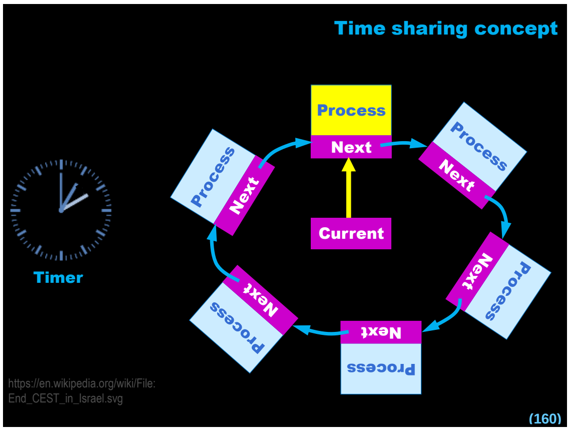
```
import threading
X= 2
muteX= threading.Semaphore()
Y= 2
muteY= threading.Semaphore()
run2(funA, funB)
```

(158)

Summary

http://www.bhmpics.com/view-road_traffic-2560x1600.html

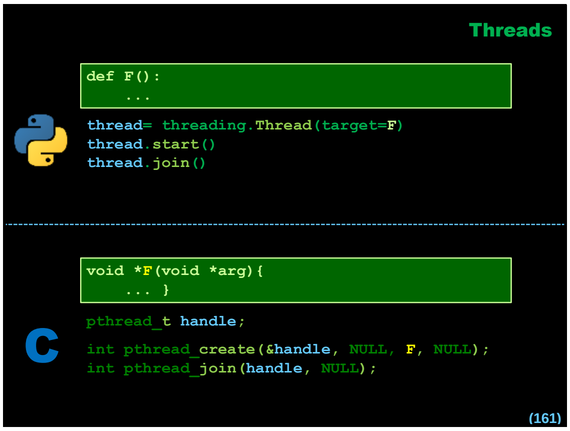
(159)



Time sharing concept

https://en.wikipedia.org/wiki/File:End_CEST_in_Israel.svg

(160)



Threads

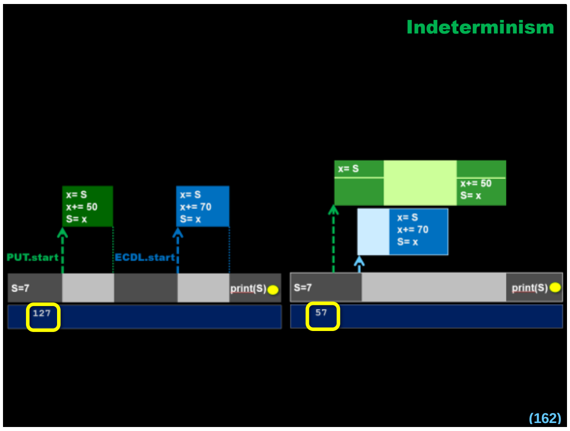
```
def F():
    ...
```

```
thread= threading.Thread(target=F)
thread.start()
thread.join()
```

```
void *F(void *arg){
    ... }
```

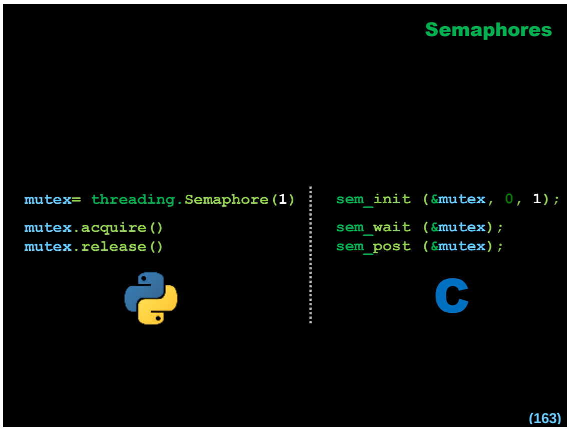
```
pthread_t handle;

int pthread_create(&handle, NULL, F, NULL);
int pthread_join(handle, NULL);
```

(161)



Indeterminism

(162)



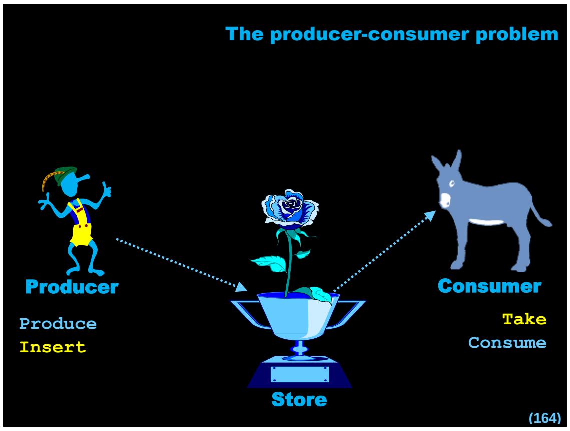
Semaphores

```
mutex= threading.Semaphore(1)

mutex.acquire()
mutex.release()
```

```
sem_init (&mutex, 0, 1);

sem_wait (&mutex);
sem_post (&mutex);
```

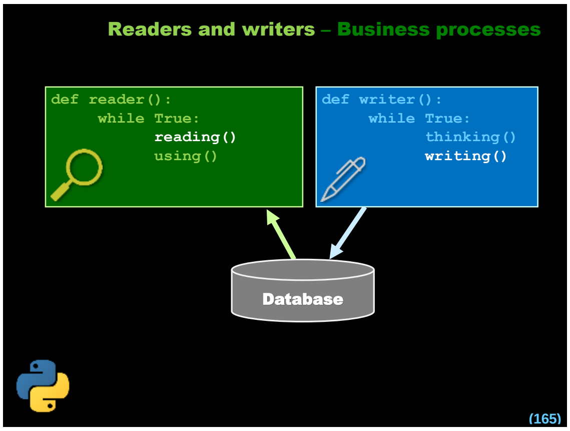
C

(163)



The producer-consumer problem

(164)

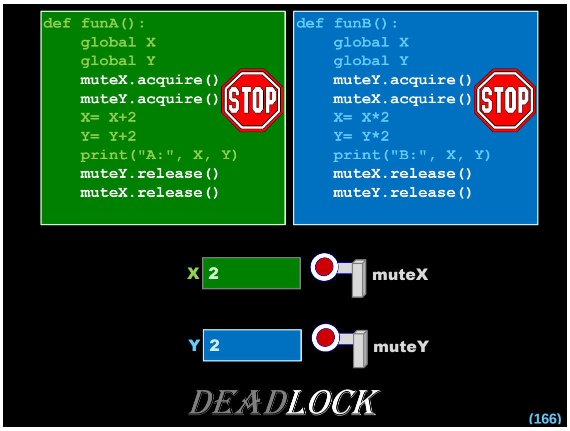


Readers and writers – Business processes

```
def reader():
    while True:
        reading()
        using()
```

```
def writer():
    while True:
        thinking()
        writing()
```

(165)

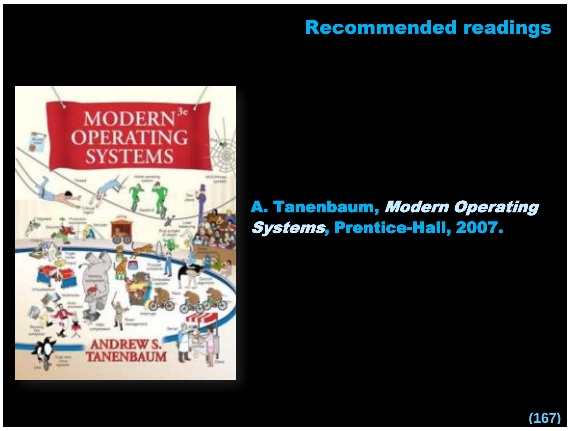


```
def funA():
    global X
    global Y
    muteX.acquire()
    muteY.acquire()
    X= X+2
    Y= Y+2
    print("A:", X, Y)
    muteY.release()
    muteX.release()
```

```
def funB():
    global X
    global Y
    muteY.acquire()
    muteX.acquire()
    X= X*2
    Y= Y*2
    print("B:", X, Y)
    muteX.release()
    muteY.release()
```

DEADLOCK

(166)



Recommended readings

A. Tanenbaum, ***Modern Operating Systems***, Prentice-Hall, 2007.

(167)

A nativity scene in Poznan

Merry Christmas & Happy New Year!