```
# YouTube Link: https://www.youtube.com/watch?v=iYJNmuD4McE

# A lock or mutex is a sychronization mechanism for enforcing
# limits on access to a resource in an environment where there
# are many threads of execution.

# More on locks:
# https://en.wikipedia.org/wiki/Lock_(computer_science)

import time
from multiprocessing import Process, Lock, Value


def add_500_no_mp(total):
    for i in range(100):
        time.sleep(0.01)
        total += 5
    return total


def sub_500_no_mp(total):
    for i in range(100):
        time.sleep(0.01)
        total -= 5
    return total


def add_500_no_lock(total):
    for i in range(100):
        time.sleep(0.01)
        total.value += 5


def sub_500_no_lock(total):
    for i in range(100):
        time.sleep(0.01)
        total.value -= 5


def add_500_lock(total, lock):
    for i in range(100):
        time.sleep(0.01)
        lock.acquire()
        total.value += 5
        lock.release()


def sub_500_lock(total, lock):
    for i in range(100):
```

```
        time.sleep(0.01)
        lock.acquire()
        total.value -= 5
        lock.release()


#if __name__ == '__main__':
#
#    total = 500
#    print(total)
#    total = add_500_no_mp(total)
#    print(total)
#    total = sub_500_no_mp(total)
#    print(total)

#if __name__ == '__main__':
#
#    total = Value('i', 500)
#    add_proc = Process(target=add_500_no_lock, args=(total,))
#    sub_proc = Process(target=sub_500_no_lock, args=(total,))
#
#    add_proc.start()
#    sub_proc.start()
#
#    add_proc.join()
#    sub_proc.join()
#    print(total.value)

if __name__ == '__main__':

    total = Value('i', 500)
    lock = Lock()
    add_proc = Process(target=add_500_lock, args=(total, lock))
    sub_proc = Process(target=sub_500_lock, args=(total, lock))

    add_proc.start()
    sub_proc.start()

    add_proc.join()
    sub_proc.join()
    print(total.value)




import time
from multiprocessing import Process, Value, Lock
```

```
def func(val, lock):
    for i in range(500):
        time.sleep(0.01)
        with lock:
            val.value += 1

if __name__ == '__main__':
    v = Value('i', 0)
    lock = Lock()
    procs=Process(target=func, args=(v, lock))

    procs.start()
    procs.join()

    print(v.value)

import time
from multiprocessing import Process, Value

def func(val):
    for i in range(500):
        time.sleep(0.01)
        val.value += 1

if __name__ == '__main__':
    v = Value('i', 0)
    procs = Process(target=func, args=(v,))

    procs.start()
    procs.join()

    print(v.value)

'''
43
```

When you use Value you get a ctypes object in shared memory that by default is synchronized using RLock. When you use Manager you get a SynManager object that controls a server process which allows object values to be manipulated by other processes. You can create multiple proxies using the same manager; there is no need to create a new manager in your loop:

Use Manager to create multiple shared objects, including dicts and lists.
Use Manager to share data across computers on a network.
Use Value or Array when it is not necessary to share information across a network and the types in ctypes are sufficient for your needs.
Value is faster than Manager.
'''

```
import threading
def worker1(v):
        #with v.get_lock():
        for i in range(1000000):
                ctypes_int.value += 1

def worker2(v):
        #with v.get_lock():
        for i in range(1000000):
                ctypes_int.value += 2

ctypes_int = threading.Value("i", 0)
print(ctypes_int.value)
process1 = threading.Thread(
   target=worker1, args=[ctypes_int])
process2 = threading.Thread(
   target=worker2, args=[ctypes_int])

process1.start()
process2.start()
process1.join()
process2.join()

print (ctypes_int.value)
```